Rigid Body Motion

A Comprehensive Introduction

Motivation

Rigid Body Motion
Andreas Kirsch

Motiviation

- Rigid Body Motion is all around us

- Rigid Body Motion is **the** game physics concept in use

- Bullet, Havok, ODE all use this concept

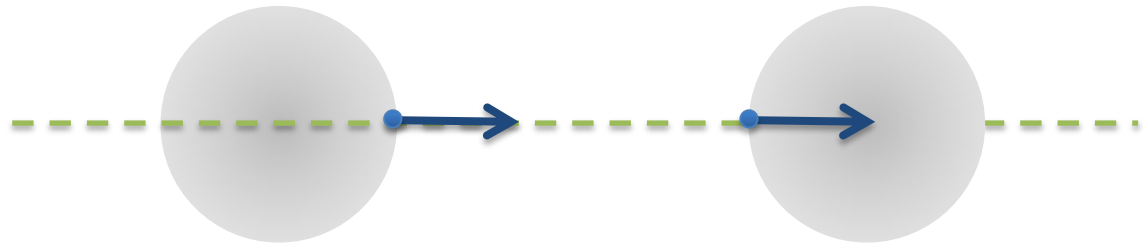**You** need to know about it if you want to work with and understand physics engines!

tum.3D
computer graphics & visualization

Overview

tum.3D
computer graphics & visualization

Rigid Body

Definition

A **rigid body (Starrkörper)** is a solid body that is not deformable.

Rigid Body

Soft Body

tum.3D
computer graphics & visualization
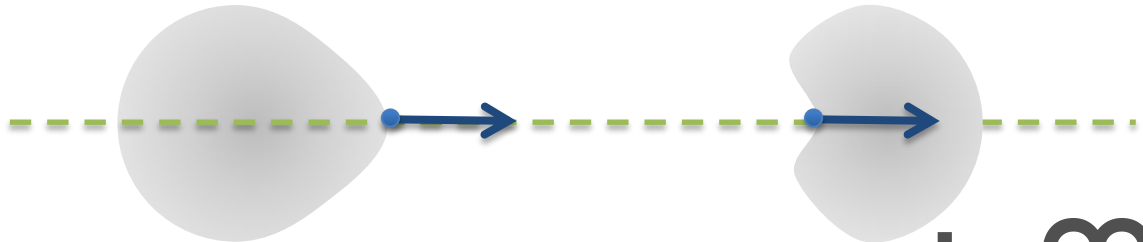
Game Engine Example

Rigid Body Update Code

```
for each (RigidBody* body in scene->rigidBodies) {
    auto contacts = scene->determineContacts(body);
    auto fieldForces = scene->determineFieldForces(body);
    auto totalForce = computeTotalForce(contacts, fieldForces);
    auto totalMoment = computeTotalMoment(contacts);
    body->linearMomentum.integrate(timeStep, totalForce);
    body->angularMomentum.integrate(timeStep, totalMoment);
    body->velocity = body->linearMomentum / body->mass;
    body->angularVelocity = body->invInertiaTensor *
                                 body->angularMomentum;
    body->position.integrate(timeStep, body->velocity);
    body->rotation.integrate(timeStep, body->angularVelocity);
}
```

tum3D
computer graphics & visualization

Where We Are

Rigid Body Update Code

```cpp
for each (RigidBody* body in scene->rigidBodies) {
    auto contacts = scene->determineContacts(body);
    auto fieldForces = scene->determineFieldForces(body);
    auto totalForce = computeTotalForce(contacts, fieldForces);
    auto totalMoment = computeTotalMoment(contacts);
    body->linearMomentum.integrate(timeStep, totalForce);
    body->angularMomentum.integrate(timeStep, totalMoment);
    body->velocity = body->linearMomentum / body->mass;
    body->angularVelocity = body->invInertiaTensor *
                                body->angularMomentum;
    body->position.integrate(timeStep, body->velocity);
    body->rotation.integrate(timeStep, body->angularVelocity);
}
```

tum.3D
computer graphics & visualization

Short Math Primer

Vector

Definition

A vector is a matrix with a single column:

$$\mathbf{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

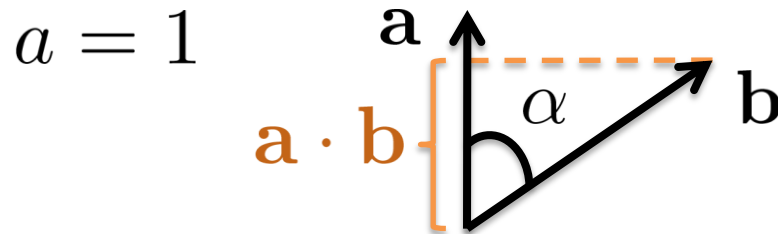v or $\|\mathbf{v}\|$ denotes the magnitude of the vector.

This is just a stub to show the notation : -)

You all know everything about vertices and matrices from your Linear Algebra lecture.

tum3D
computer graphics & visualization

Dot Product

Definition

The **dot product (Skalarproduct)** of two vectors $\mathbf{a} \cdot \mathbf{b}$ is the scalar $a\,b\cos\alpha$, where α is the angle between the two vectors.


If one of the vectors is unit length, ie its length is 1, the result of the dot product is just the projection of the other vector onto the unit vector.

Note: If $\mathbf{a} \perp \mathbf{b}$, then $\mathbf{a} \cdot \mathbf{b} = 0$.

computer graphics & visualization

Cross Product

Definition

The **cross product (Kreuzprodukt)** of two vectors $\mathbf{a} \times \mathbf{b}$ is a vector that is **perpendicular** to both $\mathbf{a}$ and $\mathbf{b}$. The length of the resulting vector is $a\,b\sin\alpha$, where α is the angle between $\mathbf{a}$ and $\mathbf{b}$.
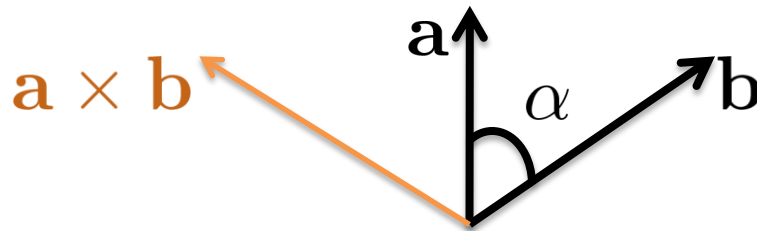
You can use the right-hand rule (Rechte-Hand-Regel) to visualize the cross product.

Note: $\mathbf{0} \times \mathbf{v} = 0$ and $(\lambda \mathbf{v}) \times \mathbf{v} = 0$
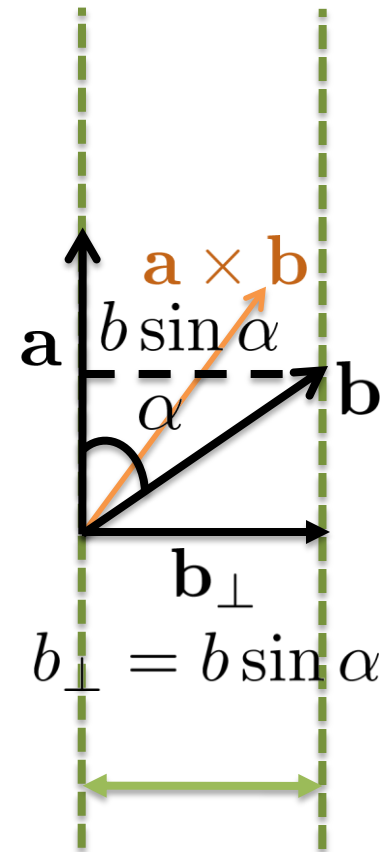
Cross Product

Also note: $\forall v : x \times v = 0 \implies x = 0$

Let's look at the magnitude formula again:

$\|\mathbf{a} \times \mathbf{b}\| = a\,b\sin\alpha$

What does it mean? If we only look at $b\sin\alpha$, where does it appear in the drawing?

This means that the magnitude of the cross product only depends on the distance of the end point of $\mathbf{b}$ to the line though $\mathbf{a}$!

$b_\perp = b\sin\alpha$

tum.3D
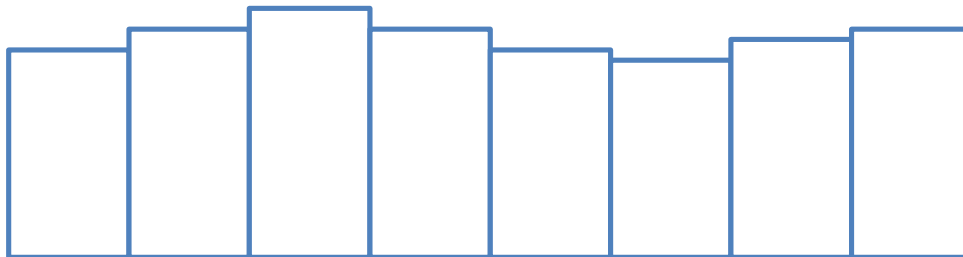computer graphics & visualization

Calculus

Don't be afraid of 'complicated' integrals like:

$$\int_{\Omega} f \, dx$$

Just assume that we work on discrete functions and treat them as a sum (which they are):

$$\sum_{x \in \Omega} f(x) \, \Delta x$$

Calculus

Likewise we can think of the derivative of a function at a certain point as the slope of the tangent on the curve at that point

$$f(x) = x * sin(x^2) + 1$$
$$f'(x) = sin(x^2) + 2x^2 * cos(x^2)$$

(2.8137, 3.8081)

(1.3552, 2.3076)

(0, 0)

x-axis

y-axis

(2.1945, -1.1828)

tum.3D
computer graphics & visualization
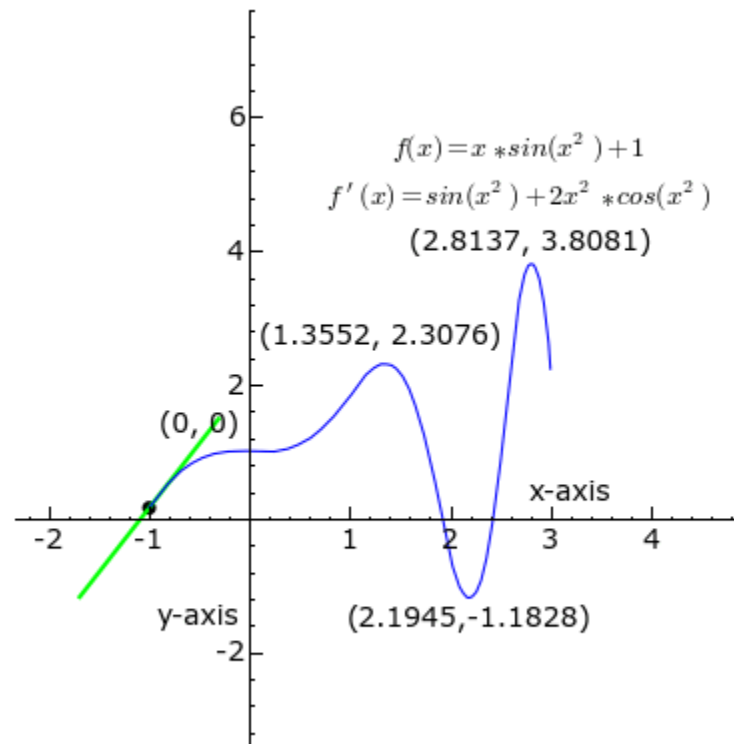
Branches of Mechanics

There are **three** branches of mechanics that are of interest to us:

Definition

Kinematics (Kinematik) describes the motion of objects without regard for the reason of their movement.

(from Wikipedia)

Definition

Dynamics (Dynamik) or also called **Kinetics (Kinetik)** examines the correlation between an object's motion and the reasons for it.

(from Wikipedia)

Definition

Statics (Statik) analyzes the motion of objects without regard for the reason of their movement.

(from Wikipedia)

tum.3D
computer graphics & visualization

Kinematics

Rigid Body Motion
Andreas Kirsch

Linear Motion

Let's look at the movement of a simple **particle (Massenpunkt)** .
We identify a particle i by its **position** $\mathbf{r_i}$.

$$\mathbf{r_i} = \begin{pmatrix} r_{ix} \\ r_{iy} \\ r_{iz} \end{pmatrix}$$

The **velocity** $\mathbf{v_i}$ of a particle i is its change of position over time:

$$\mathbf{v_i} = \frac{d\mathbf{r_i}}{dt} = \dot{\mathbf{r}}_\mathbf{i}$$

The **speed (Tempo)** of a particle is simply the magnitude of its velocity: v_i .

Likewise the **acceleration** $\mathbf{a_i}$ is the change of velocity over time:

$$\mathbf{a_i} = \frac{d\mathbf{v_i}}{dt} = \dot{\mathbf{v}}_\mathbf{i} = \ddot{\mathbf{r}}_\mathbf{i}$$

tum.3D
computer graphics & visualization

Angular Motion in 2D

Imagine that a particle moves on a circle around the origin.

We can exactly describe the particle's position by the angle ϕ between the particle and the x-axis at any moment of time.

Like with **linear motion**, we can examine the change of ϕ over time and get the **angular velocity** ω:

$$\omega = \frac{d\phi}{dt} = \dot{\phi}$$

Similarly the **angular acceleration** α is the change of ω over time:

$$\alpha = \frac{d\omega}{dt} = \dot{\omega} = \ddot{\phi}$$

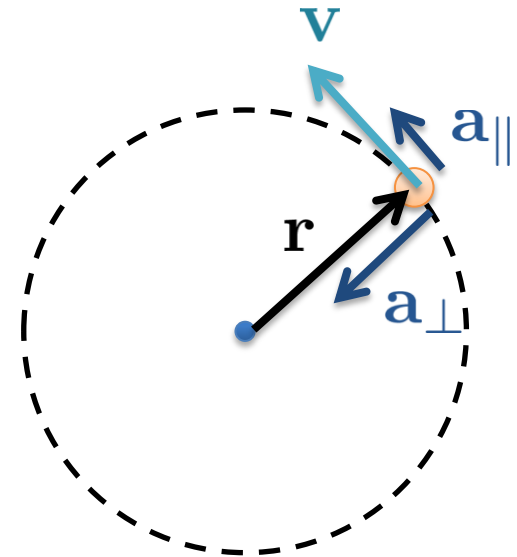tum.3D
computer graphics & visualization

Example: Circular Movement

$$\mathbf{r} = r \begin{pmatrix} \cos \phi \\ \sin \phi \end{pmatrix}$$

$$\Rightarrow \mathbf{v} = \dot{\mathbf{r}} = r \begin{pmatrix} -\sin \phi \\ \cos \phi \end{pmatrix} \cdot \dot{\phi} = r\,\omega \begin{pmatrix} -\sin \phi \\ \cos \phi \end{pmatrix}$$

And $\mathbf{v} \perp \mathbf{r}$ because $\mathbf{v} \cdot \mathbf{r} = r^2\,\omega\,(-\sin\phi\,\cos\phi + \sin\phi\,\cos\phi) = 0$

$$\mathbf{a} = \dot{\mathbf{v}} = r\,\dot{\omega} \begin{pmatrix} -\sin \phi \\ \cos \phi \end{pmatrix} + r\,\omega \begin{pmatrix} -\cos \phi \\ -\sin \phi \end{pmatrix} \dot{\phi}$$

$$= \underbrace{r\,\alpha \begin{pmatrix} -\sin \phi \\ \cos \phi \end{pmatrix}}_{\mathbf{a}_\parallel} \underbrace{-r\,\omega^2 \begin{pmatrix} \cos \phi \\ \sin \phi \end{pmatrix}}_{\mathbf{a}_\perp}$$

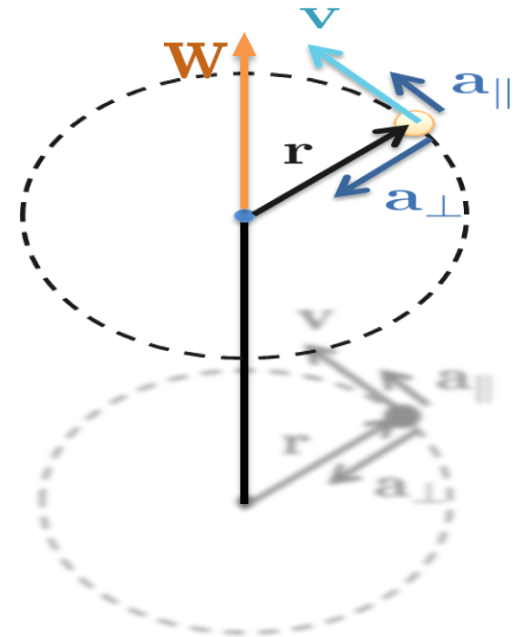tum.3D
computer graphics & visualization

Example: Circular Movement in 3D

If we look at the same movement in 3D using $\mathbf{w}$ as rotation axis, it is clear that the magnitudes of $\mathbf{v}$, $\mathbf{a}_{\parallel}$ and $\mathbf{a}_{\perp}$ stay the same. However, we need to come up with the direction in a new way.

It's obvious that $\mathbf{a}_{\perp}$ has the direction $-\mathbf{r}$.

But what about $\mathbf{v}$ and $\mathbf{a}_{\parallel}$?

$\mathbf{v}$ and $\mathbf{a}_{\parallel}$ are collinear to $\mathbf{w} \times \mathbf{r}$!

tum.3D
computer graphics & visualization

Example: Rotation around Axis

Now if we have a rotation around any axis (through the origin) in 3D, we can treat the rotation as a circular movement by only looking at the $\mathbf{r}_\perp$ component.
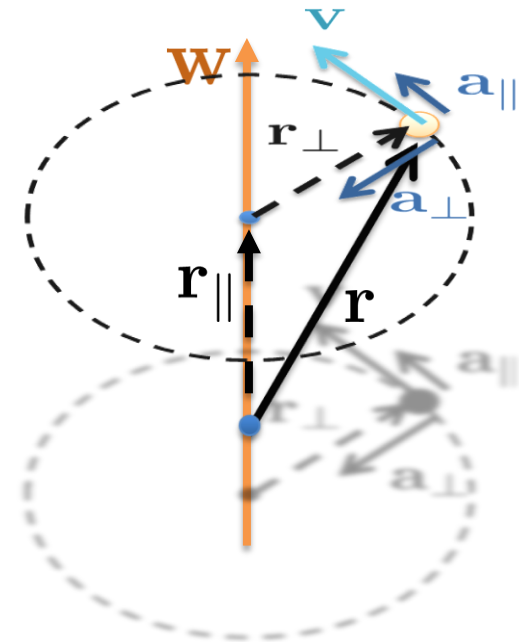
Here is the nice thing: the cross product does this already for us!

$$\mathbf{w} \times \mathbf{r} = \mathbf{w} \times \mathbf{r}_\perp$$

For the magnitude we get:

$$\|\mathbf{w} \times \mathbf{r}\| = w\, r_\perp$$

Now remember that we had $v = \omega\, r$ in the 2D case, so if we set w to ω, everything gets very simple!

Example: Rotation around Axis

So if we set:

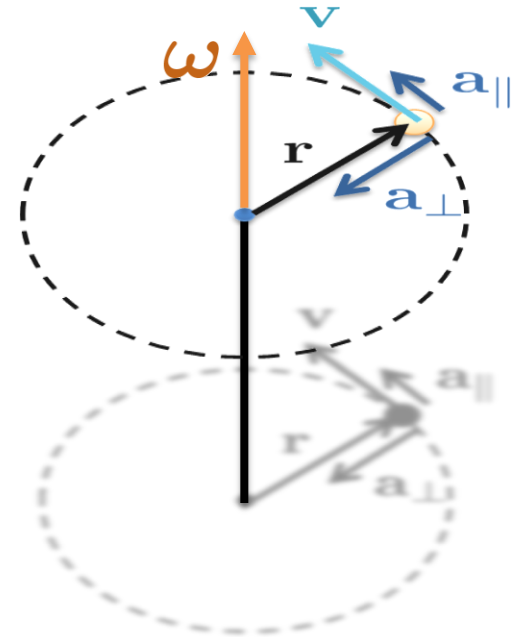$$\mathbf{v} = \boldsymbol{\omega} \times \mathbf{r}$$

What is the acceleration?

$$\mathbf{a} = \dot{\mathbf{v}} = \frac{d}{dt}\left(\boldsymbol{\omega} \times \mathbf{r}\right)$$

$$\mathbf{a} = \dot{\boldsymbol{\omega}} \times \mathbf{r} + \boldsymbol{\omega} \times \dot{\mathbf{r}}$$

$$\mathbf{a} = \boldsymbol{\alpha} \times \mathbf{r} + \boldsymbol{\omega} \times \mathbf{v}$$

$$\mathbf{a} = \underbrace{\boldsymbol{\alpha} \times \mathbf{r}}_{=\mathbf{a}_\parallel} + \underbrace{\boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r})}_{=\mathbf{a}_\perp}$$

tum.3D
computer graphics & visualization

Angular Motion in 3D

In 3D we can't use a single angle to describe the orientation. Instead we use a rotation matrix R.

The derivative of a matrix is again a matrix, so instead we're using something simpler: The change of the rotation matrix from one moment to the next is nothing else but a rotation itself.

We can describe this rotation using the rotation axis ω from the previous slides and its magnitude is the amount of rotation.
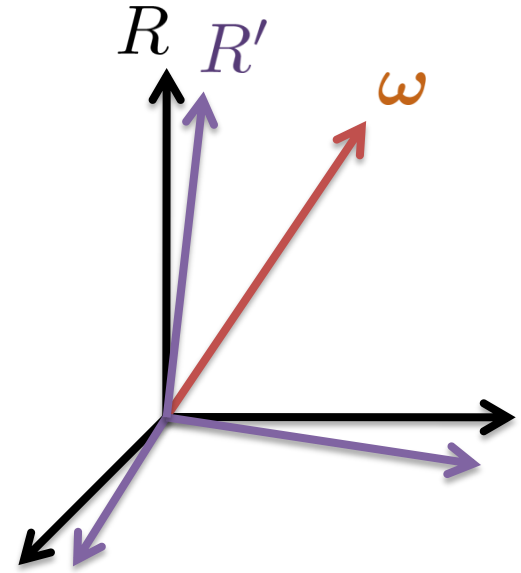
Angular Motion in 3D

It can be shown that in this case $\dot{R} = \omega \times R$, so we can use this representation.

Now we know how to get from $\boldsymbol{\omega}$ to $\dot{R}$ and can use the formulas that we have deduced when rotating a point around an axis:

$$\mathbf{v} = \boldsymbol{\omega} \times \mathbf{r}$$

$$\mathbf{a} = \boldsymbol{\alpha} \times \mathbf{r} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r})$$

to get convert from angular quantities to linear ones.

Summary

Linear Motion:

- Position $\mathbf{r}$

- Velocity $\mathbf{v}$

- Acceleration $\mathbf{a}$

Angular Motion:

- Orientation ϕ or R

- Angular Velocity ω

  - $\mathbf{v} = \boldsymbol{\omega} \times \mathbf{r}$

- Angular Acceleration α

  - $\mathbf{a} = \boldsymbol{\alpha} \times \mathbf{r} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r})$

Statics

Rigid Body Motion
Andreas Kirsch

Force

Definition

A **force (Kraft)** is any influence that causes a free body to undergo an acceleration.

A force has both **magnitude** and **direction**, making it a vector quantity.

(from Wikipedia)

It also has a **point of action (Angriffspunkt)**: the point where the force is applied; and a **line of action (Wirkungslinie)**: the line along the force's direction through the point of action.

tum.3D
computer graphics & visualization

Force

line of action

body

force

point of action
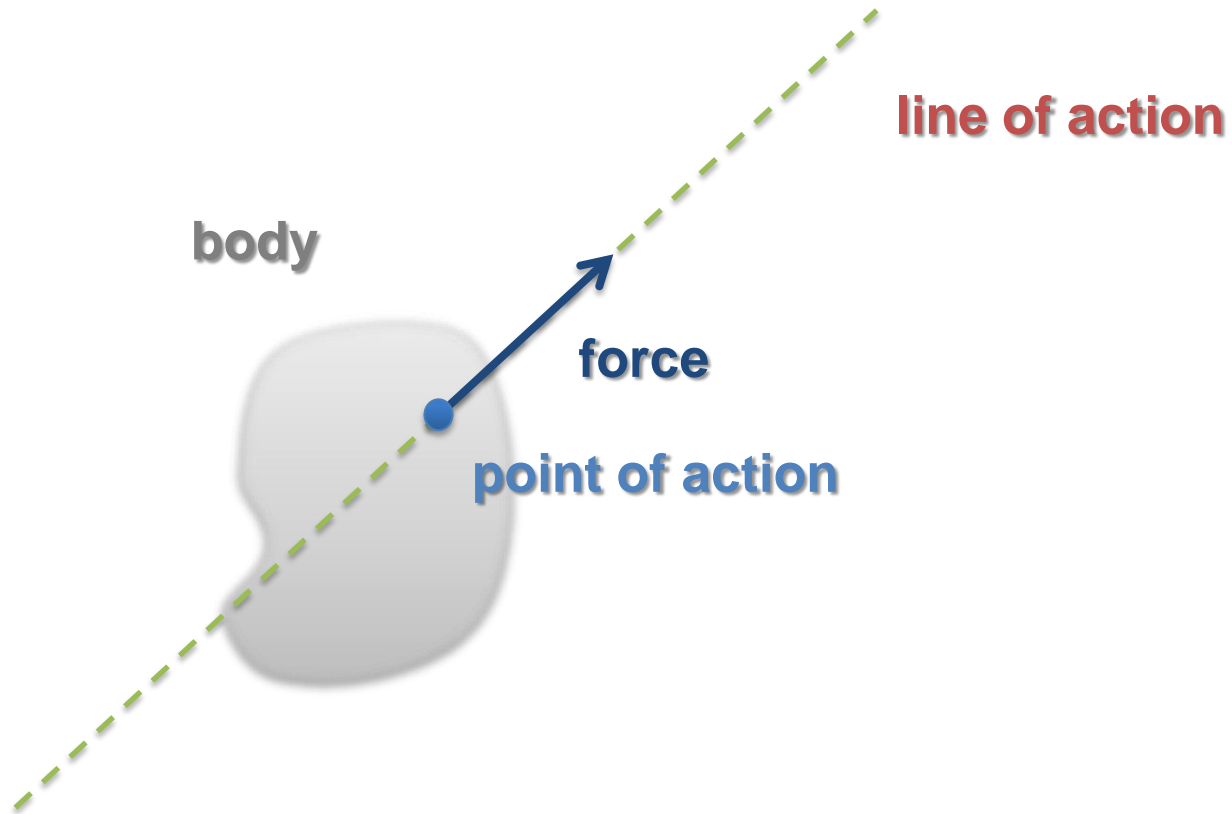
Newton's Laws of Motion

Second Law

The acceleration of a body is proportional to the net force acting on the body and this acceleration is in the same direction as the net force.

$$\mathbf{F} = m\,\mathbf{a}$$

Third Law

For every force acting on a body, there is an equal and opposite reacting force.

$$-actio = reactio$$
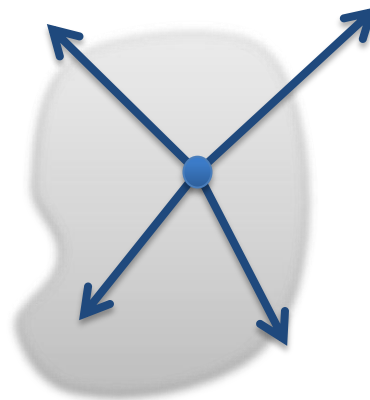
tum.3D

computer graphics & visualization

Equilibrium of Forces

Definition

A set of forces $\mathfrak{F}$ that act **on the same point of action** is said to be **in equilibrium (Gleichgewicht)** if the sum of the forces is zero:

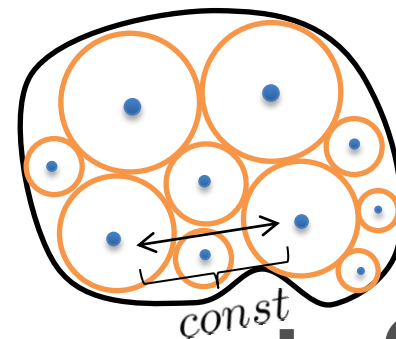$$\sum_{\mathbf{F} \in \mathfrak{F}} \mathbf{F} = 0$$

Rigid Bodies

> **Definition**
>
> A rigid body (Starrkörper) is a solid body that is not deformable.

If we take a body to be made-up of particles, then the definition means that the distance between any two particles always remains constant.

This, of course, is an idealization, because every body can be deformed if enough stress is applied.
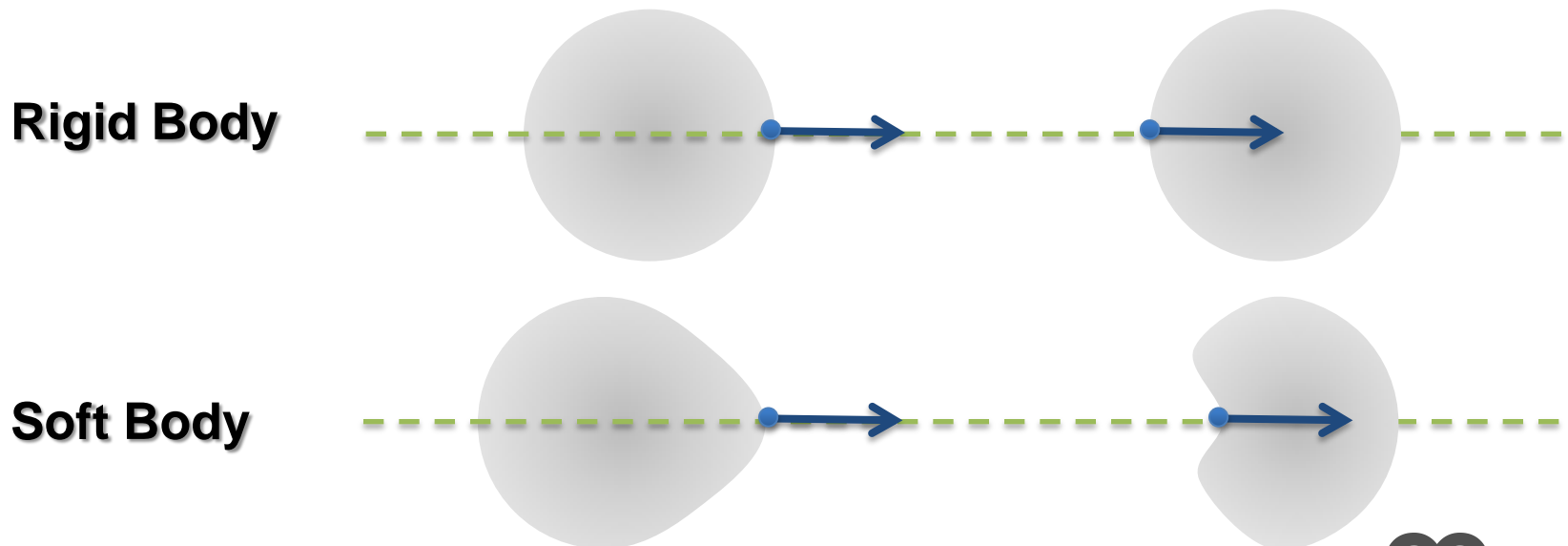
Rigid Bodies

Because of this rigidness for a rigid body only its position and orientation in space need to be stored.

They suffice to describe its instantaneous state.

Another special property of rigid bodies is that the point of action for a force is irrelevant – only the line of action matters.

Rigid Body

Soft Body

tum.3D
computer graphics & visualization
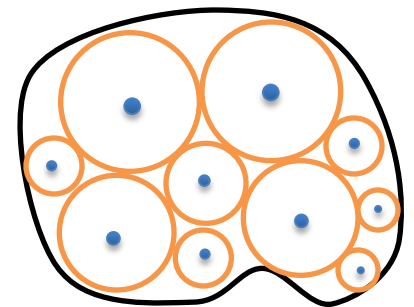
Rigid Bodies – Particle Model

We already know particles. They have mass but no orientation. A group of particles, however, has orientation.

We can use a **set of particles** $\mathbf{B}$ to model a rigid body. Each particle has its own properties: mass, position, velocity, etc. But they are all linked together by their constant distance to each other. We will use this to deduce many important properties of rigid bodies.

We will see later that many properties of the whole rigid body are made up of the sum of its particles.

The mass of the rigid body for example is:

$$m = \sum_{i \in \mathbf{B}} m_i = \int_{\mathbf{B}} dm$$

Rigid Body Motion
Andreas Kirsch

tum.3D
computer graphics & visualization

Where We Are

Rigid Body Update Code

```
for each (RigidBody* body in scene->rigidBodies) {
    auto contacts = scene->determineContacts(body);
    auto fieldForces = scene->determineFieldForces(body);
    auto totalForce = computeTotalForce(contacts, fieldForces);
    auto totalMoment = computeTotalMoment(contacts);
    body->linearMomentum.integrate(timeStep, totalForce);
    body->angularMomentum.integrate(timeStep, totalMoment);
    body->velocity = body->linearMomentum / body->mass;
    body->angularVelocity = body->invInertiaTensor *
                            body->angularMomentum;
    body->position.integrate(timeStep, body->velocity);
    body->rotation.integrate(timeStep, body->angularVelocity);
}
```

tum.3D
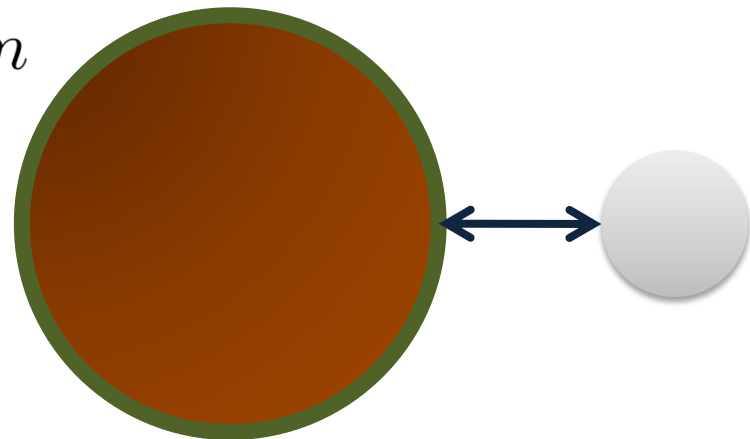computer graphics & visualization

Field Forces

> **Definition**
>
> A **field force (Feldkraft)** is a force that is not transferred through direct contact but through a **force field (Kraftfeld)**.

Example: Gravity

$$F_g = 9.81 \frac{\mathrm{m}}{s^2}\, m$$

tum.3D
computer graphics & visualization

Contact Forces

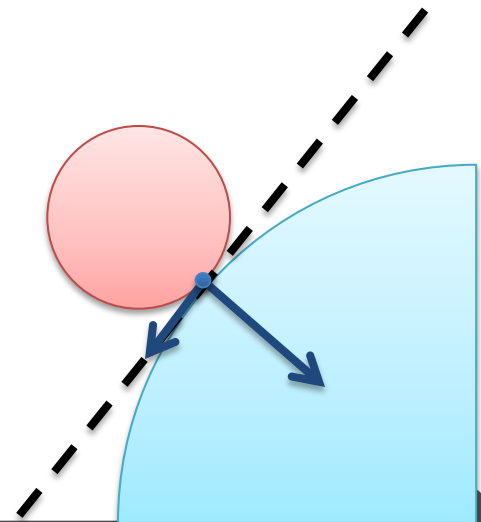**Definition**

A **contact force (Kontaktkraft)** is a force that is transmitted through a contact point of two objects that touch.

Because of $actio = -reactio$ both bodies apply the same force (but in opposite directions) on each other.

It consists of a **normal force (Normalkraft)** perpendicular to the contact plane and **a friction force (Reibungskraft)** that lies on the contact plane.

Where We Are

Rigid Body Update Code

```cpp
for each (RigidBody* body in scene->rigidBodies) {
    auto contacts = scene->determineContacts(body);
    auto fieldForces = scene->determineFieldForces(body);
    auto totalForce = computeTotalForce(contacts, fieldForces);
    auto totalMoment = computeTotalMoment(contacts);
    body->linearMomentum.integrate(timeStep, totalForce);
    body->angularMomentum.integrate(timeStep, totalMoment);
    body->velocity = body->linearMomentum / body->mass;
    body->angularVelocity = body->invInertiaTensor *
                                body->angularMomentum;
    body->position.integrate(timeStep, body->velocity);
    body->rotation.integrate(timeStep, body->angularVelocity);
}
```

tum.3D
computer graphics & visualization

Moment (Motivation)

Law of the Lever

Law of the Lever (Hebelgesetz)

A level is in equilibirum if the forces applied on each end is inversely proportional to the distance of the points of action to the pivot:

$$\mathbf{F}_1\, h_1 = \mathbf{F}_2\, h_2$$
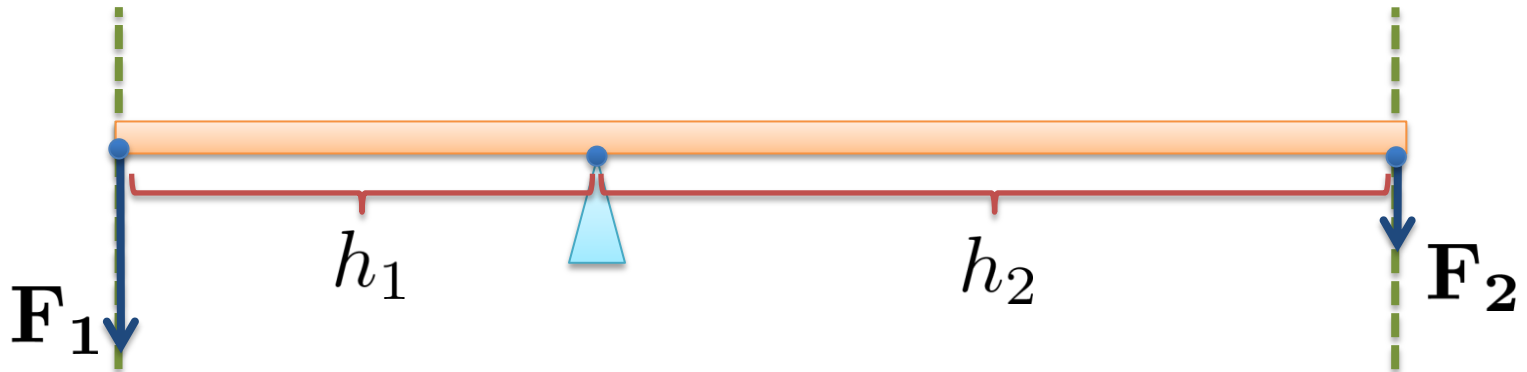
Moment

Moment (Drehmoment) is a measure for the ability of a force to rotate an object.
It is always defined with respect to a point of reference $\mathbf{r_x}$:

$$\mathbf{M}_{\mathbf{F}}^{(\mathbf{x})} = (\mathbf{r_F} - \mathbf{r_x}) \times \mathbf{F} = \mathbf{r}_{\mathbf{F}}^{(\mathbf{x})} \times \mathbf{F}$$
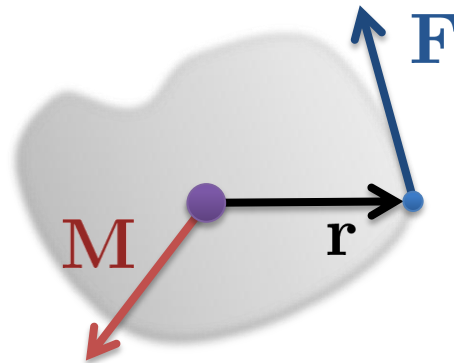
Moment

$\mathbf{M} = \mathbf{r} \times \mathbf{F}$ is perpendicular to the force vector and the vector from the point of reference, and the magnitude of the moment vector is the force's magnitude times the distance between the point of reference and the line of action of the force:

$$M = h\, F$$
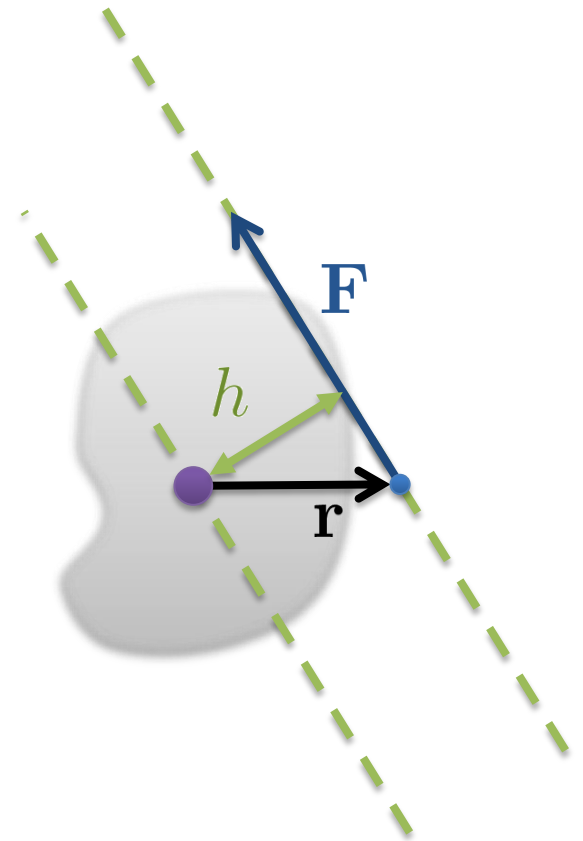
Net Force and Net Torque

If we have forces $\mathbf{F}_1, \mathbf{F}_2, ..., \mathbf{F}_n$ (with points of action $\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_n$ etc), then we can calculate the net force and net moment in respect to a point of reference $\mathbf{x}$ with:

$$\mathbf{F} = \sum_i \mathbf{F}_i$$

$$\mathbf{M}^{(\mathbf{x})} = \sum_i \mathbf{M}_{\mathbf{F}_i}^{(\mathbf{x})}$$

That is, we can simply add up the forces and moments to get the net force and net moment.
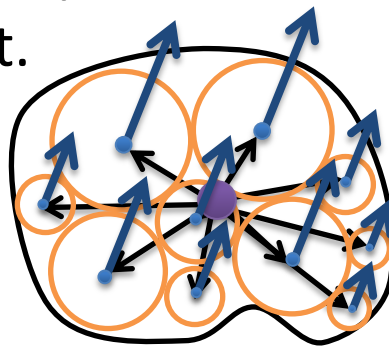
Center of Mass (Motivation)

What is a good choice for our point of reference?

It would be nice to choose it in such a way that applying a force to it won't induce any actual moment on an unconstrained rigid body.

This means that every particle as well as the whole rigid body will experience the same acceleration. So if any force $\mathbf{F}$ is applied to the point of reference, the whole rigid body should be accelerated by $\mathbf{a} = \frac{\mathbf{F}}{m}$. m is the mass of the whole rigid body, ie $m = \sum_i m_i$. Then:

$$\mathbf{F}_i = m_i \, \mathbf{a} = \frac{m_i}{m} \mathbf{F}$$
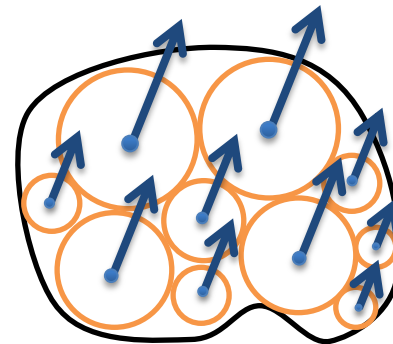
Center of Mass (Deduction)

$$\sum_i \mathbf{M}_i \quad (\text{net moment}) = \mathbf{M} = 0$$

$$\sum_i (\mathbf{r_i} - \mathbf{r}) \times \mathbf{F}_i = 0$$

$$\sum_i (\mathbf{r_i} - \mathbf{r}) \times \frac{m_i}{m} \mathbf{F} = \left(\sum_i \frac{m_i}{m} (\mathbf{r_i} - \mathbf{r}) \right) \times \mathbf{F} = 0$$
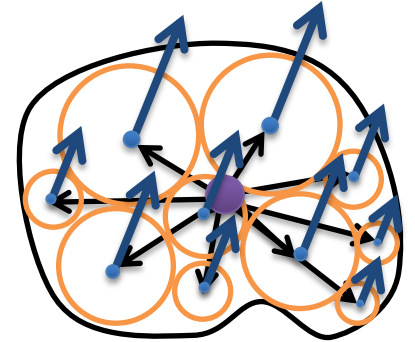
Because the last statement has to hold for any force $\mathbf{F}$, it follows that $\sum_i \frac{m_i}{m} (\mathbf{r_i} - \mathbf{r}) = 0$, that is:

$$\sum_i \frac{m_i}{m} (\mathbf{r_i} - \mathbf{r}) = \sum_i \frac{m_i}{m} \mathbf{r_i} - \sum_i \frac{m_i}{m} \mathbf{r} = \sum_i \frac{m_i}{m} \mathbf{r_i} - \mathbf{r} = 0$$

$$\mathbf{r} = \sum_i \frac{m_i}{m} \mathbf{r_i} = \frac{1}{m} \int \mathbf{r} \, dm$$

Center of Mass

Definition

The **center of mass ((Masse-)Schwerpunkt) $\mathbf{cm}$** of a rigid body is the mass-weighted average of all particle positions:

$$\mathbf{r}_{cm} = \sum_i \frac{m_i}{m}\, \mathbf{r_i} = \frac{1}{m} \int \mathbf{r}\, dm$$

The center of mass is the perfect point of reference to use when calculating the net moment for an unconstrained rigid body. (The net force is independent of a point of reference.)

tum.3D
computer graphics & visualization

Where We Are

Rigid Body Update Code

```
for each (RigidBody* body in scene->rigidBodies) {
    auto contacts = scene->determineContacts(body);
    auto fieldForces = scene->determineFieldForces(body);
    auto totalForce = computeTotalForce(contacts, fieldForces);
    auto totalMoment = computeTotalMoment(contacts);
    body->linearMomentum.integrate(timeStep, totalForce);
    body->angularMomentum.integrate(timeStep, totalMoment);
    body->velocity = body->linearMomentum / body->mass;
    body->angularVelocity = body->invInertiaTensor *
                            body->angularMomentum;
    body->position.integrate(timeStep, body->velocity);
    body->rotation.integrate(timeStep, body->angularVelocity);
}
```

tum.3D
computer graphics & visualization

Kinetics

Linear Momentum

Definition

The **linear momentum (Impuls)** $\mathbf{p}$ of a particle i is defined as:

$$\mathbf{p}_i = m\,\mathbf{v}_i$$

The linear momentum of a rigid body is the sum of the linear momentums of all its particles:

$$\mathbf{p} = \sum_i \mathbf{p}_i = \int \mathbf{v}\,dm = m\,\mathbf{v}_{cm}$$

The linear momentum of a system of bodies is a **conserved quantity (Erhaltungsgröße)**, ie the total amount never changes inside the system.

This is called **conservation of momentum (Impulserhaltung)**.

Linear Momentum and Force

$$\mathbf{F} = m\,\mathbf{a}$$

$$\mathbf{p} = m\,\mathbf{v}$$

Look at the derivative of $\mathbf{p}$:

$$\dot{\mathbf{p}} = \dot{m}\,\mathbf{v} + m\,\dot{\mathbf{v}}$$

But m is constant and thus $\dot{m} = 0$.
We get:

Derivative of Linear Momentum = Force

The linear momentum changes according to the applied force:

$$\dot{\mathbf{p}} = m\,\dot{\mathbf{v}} = m\,\mathbf{a} = F$$

tum.3D
computer graphics & visualization

Newton's First Law of Motion

> **Newton's First Law**
>
> Every object in a state of uniform motion tends to remain in that state of motion unless an external force is applied to it. This is the concept of **inertia (Trägheit).**

Proof:

This follows directly from the conservation of momentum:

If $\mathbf{F} = 0$ then $\dot{\mathbf{p}} = 0$, and then $\mathbf{p} = m\,\mathbf{v} = const$ and since the mass m is constant, we see that the velocity is constant, too.

tum.3D
computer graphics & visualization

Angular Momentum

The **angular momentum (Drehimpuls)** $\mathbf{L}$ of a particle i is defined as:

$$\mathbf{L}_i^{(\mathbf{x})} = (\mathbf{r}_i - \mathbf{r}_x) \times \mathbf{p}_i = \mathbf{r}_i^{(x)} \times \mathbf{p}_i$$

with respect to a point of reference $\mathbf{x}$.

The angular momentum of a rigid body is the sum of the angular momentum of its particles:

$$\mathbf{L}^{(x)} = \sum_i \mathbf{L}_i^{(\mathbf{x})} = \sum_i \mathbf{r}_i^{(x)} \times \mathbf{p}_i = \int \mathbf{r}^{(x)} \times \mathbf{v_i} \, dm$$

Angular momentum is a **conserved quantity**, too, that is it stays constant for a system of particles (except if outer forces are applied).

tum.3D
computer graphics & visualization

Angular Momentum and Moment

Derivative of Angular Momentum = Moment

Angular momentum changes according to the applied momentum (around the center of mass):

$$\dot{\mathbf{L}}^{(cm)} = \mathbf{M}^{(cm)}$$

Proof Sketch:

$$\dot{L}_{\mathbf{x}}^{(cm)} = \frac{d}{dt}\left(\mathbf{r}_{\mathbf{x}}^{(cm)} \times \mathbf{p}_{\mathbf{x}}\right)$$

$$\dot{L}_{\mathbf{x}}^{(cm)} = \underbrace{\dot{\mathbf{r}}_{\mathbf{x}}^{(cm)} \times \mathbf{p}_{\mathbf{x}}}_{=\mathbf{0}} + \underbrace{\mathbf{r}_{\mathbf{x}}^{(cm)} \times \dot{\mathbf{p}}_{\mathbf{x}}}_{\mathbf{M}_{\mathbf{x}}^{(cm)}} = \mathbf{M}_{\mathbf{x}}^{(cm)}$$

$$\tau = \mathbf{r} \times \mathbf{F}$$
$$\mathbf{L} = \mathbf{r} \times \mathbf{p}$$

tum.3D
computer graphics & visualization
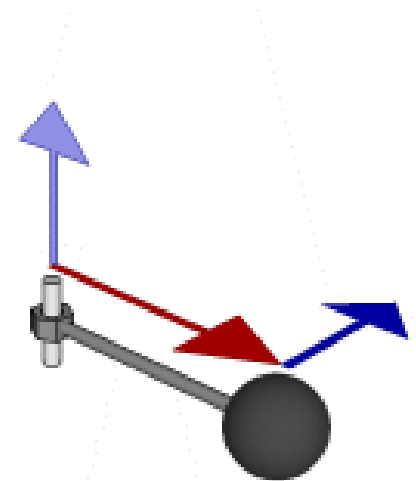
Integration

The problem is that you know the net moment and net torque, but want to know the linear and angular momentum.

From the differential equation we need to evaluate:

$$\mathbf{p}\left(t+dt\right) = \int_{t}^{t+dt} \mathbf{F} dt + \mathbf{p}\left(t\right)$$

$$\mathbf{L}\left(t+dt\right) = \int_{t}^{t+dt} \mathbf{M} dt + \mathbf{L}\left(t\right)$$

The integration steps in a rigid body simulation can be as difficult and complicated as you want.

To solve this you can eg use a simple Euler step:

$$\mathbf{p}(t+\Delta t) = \mathbf{p}(t) + \mathbf{F}(t)\,\Delta t$$

$$\mathbf{L}(t+\Delta t) = \mathbf{L}(t) + \mathbf{M}(t)\,\Delta t$$

tum.3D
computer graphics & visualization

Summary

- Linear Momentum $\mathbf{p} = m\,\mathbf{v}$

- Angular Momentum $\mathbf{L} = \mathbf{r} \times \mathbf{p}$

- Conservation of Linear Momentum

- Conservation of Angular Momentum

- Linear Momentum = Integrated Force: $\dot{\mathbf{p}} = \mathbf{F}$

- Angular Momentum = Integrated Moment: $\dot{\mathbf{L}} = \mathbf{M}$

- Integration = actual problem

tum.3D
computer graphics & visualization

Where We Are

Rigid Body Update Code

```
for each (RigidBody* body in scene->rigidBodies) {
    auto contacts = scene->determineContacts(body);
    auto fieldForces = scene->determineFieldForces(body);
    auto totalForce = computeTotalForce(contacts, fieldForces);
    auto totalMoment = computeTotalMoment(contacts);
    body->linearMomentum.integrate(timeStep, totalForce);
    body->angularMomentum.integrate(timeStep, totalMoment);
    body->velocity = body->linearMomentum / body->mass;
    body->angularVelocity = body->invInertiaTensor *
                            body->angularMomentum;
    body->position.integrate(timeStep, body->velocity);
    body->rotation.integrate(timeStep, body->angularVelocity);
}
```

tum.3D
computer graphics & visualization

Angular Momentum and Angular Velocity

We are interested in the kinematics of a rigid body to animate it correctly on the screen. For this we need to determine quantities like linear and angular velocity.

From the definition of linear momentum $\mathbf{p} = m\,\mathbf{v}$ we can directly identify $\mathbf{v} = \frac{\mathbf{p}}{m}$.

What about angular velocity though?

We only know that $\mathbf{L}^{(cm)} = \sum_i \mathbf{r}_i^{(cm)} \times \mathbf{p_i} = \sum_i \mathbf{r}_i^{(cm)} \times m_i\,\mathbf{v}_i$. But $\mathbf{v}_i$ depends on the angular velocity ω indirectly.

Using this fact, it is possible to transform the equation to

$$\mathbf{L}^{(cm)} = \Phi\,\omega,$$

where Φ is the **inertia tensor (Trägheitstensor)**.

tum.3D
computer graphics & visualization

Inertia Tensor in 2D

In 2D the moment definition can be simplified to scalars (like we did for the moment):

$$L_i^{(cm)} = h_i^{(cm)} p_i \text{ with } h_i^{(cm)} = r_i^{(cm)} \sin \alpha$$

If we look at an angular motion, it can also be thought of as circular motion around the center of gravity and since $\mathbf{p_i}$ is collinear with $\mathbf{v}$, we know that it is perpendicular to $\mathbf{r_i}$, this gives $h_i = r_i$. From the slide about circular movement we also know that

$$\mathbf{v_i} = r_i \, \omega \begin{pmatrix} -\sin\phi \\ \cos\phi \end{pmatrix} \Rightarrow v_i = r_i \, \omega \; .$$
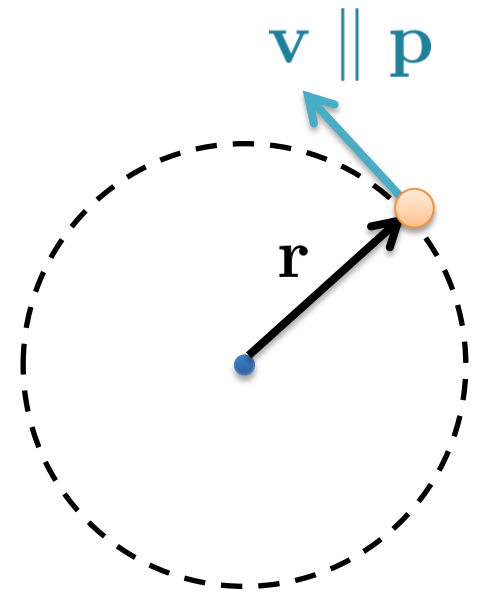
Inertia Tensor in 2D

$$L_i^{(cm)} = h_i^{(cm)} \, p_i \text{ with } h_i^{(cm)} = r_i^{(cm)} \sin \alpha$$

$$\mathbf{v_i} = r_i \, \omega \begin{pmatrix} -\sin\phi \\ \cos\phi \end{pmatrix} \Rightarrow v_i = r_i \, \omega$$

$$L^{(cm)} = \sum_i L_i^{(cm)} = \sum_i r_i \, p_i = \sum_i r_i \, m_i \, v_i = \sum_i m_i \, r_i \, (r_i \, \omega)$$

$$= \sum_i m_i \, r_i^2 \, \omega = \left(\sum_i m_i \, r_i^2 \right) \omega = \Phi \, \omega$$

$$\Phi = \sum_i m_i \, r_i^2 = \int r^2 dm$$

tum.3D
computer graphics & visualization

Inertia Tensor in 2D

Definition

In 2D the inertia tensor is just a scalar. It is the squared distance of all particles to the center of gravity weighted by their mass:

$$\Phi = \sum_i m_i \, r_i^{(cm)^2} = \int r^{(cm)^2} dm$$
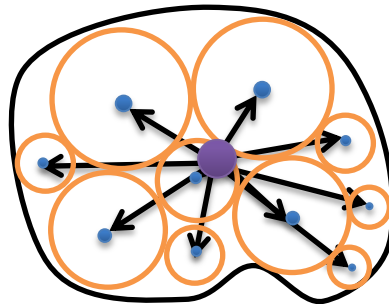
Inertia Tensor in 3D

We can do the same in 3D, but this we'll get a 3x3 matrix. Here is an over-simplified deduction:

$$\mathbf{L} = \sum_i \mathbf{r_i} \times \mathbf{p_i} = \sum_i \mathbf{r_i} \times (m_i\, \mathbf{v_i}) = \sum_i m_i\, \mathbf{r_i} \times \mathbf{v_i}$$

$$= \sum_i m_i\, \mathbf{r_i} \times (\boldsymbol{\omega} \times \mathbf{r_i}) = \sum_i m_i\, \left(r_i^2\, \boldsymbol{\omega} - (\mathbf{r_i} \cdot \boldsymbol{\omega})\, \mathbf{r_i} \right)$$

$$= \sum_i m_i \left(\left(r_{ix}^2 + r_{iy}^2 + r_{iz}^2 \right) \boldsymbol{\omega} - \begin{pmatrix} r_{ix}^2 & r_{ix}\,r_{iy} & r_{ix}\,r_{iz} \\ r_{ix}\,r_{iy} & r_{iy}^2 & r_{iy}\,r_{iz} \\ r_{ix}\,r_{iz} & r_{iy}\,r_{iz} & r_{iz}^2 \end{pmatrix} \boldsymbol{\omega} \right)$$

$$= \sum_i m_i \begin{pmatrix} r_{iy}^2 + r_{iz}^2 & -r_{ix}\,r_{iy} & -r_{ix}\,r_{iz} \\ -r_{ix}\,r_{iy} & r_{ix}^2 + r_{iz}^2 & -r_{iy}\,r_{iz} \\ -r_{ix}\,r_{iz} & -r_{iy}\,r_{iz} & r_{ix}^2 + r_{iy}^2 \end{pmatrix} \boldsymbol{\omega}$$

tum.3D
computer graphics & visualization

Inertia Tensor in 3D

$$= \left(\sum_i m_i \begin{pmatrix} r_{iy}^2 + r_{iz}^2 & -r_{ix}\,r_{iy} & -r_{ix}\,r_{iz} \\ -r_{ix}\,r_{iy} & r_{ix}^2 + r_{iz}^2 & -r_{iy}\,r_{iz} \\ -r_{ix}\,r_{iz} & -r_{iy}\,r_{iz} & r_{ix}^2 + r_{iy}^2 \end{pmatrix} \right) \boldsymbol{\omega} = \Phi\,\boldsymbol{\omega}$$

$$\Phi = \sum_i m_i \begin{pmatrix} r_{iy}^2 + r_{iz}^2 & -r_{ix}\,r_{iy} & -r_{ix}\,r_{iz} \\ -r_{ix}\,r_{iy} & r_{ix}^2 + r_{iz}^2 & -r_{iy}\,r_{iz} \\ -r_{ix}\,r_{iz} & -r_{iy}\,r_{iz} & r_{ix}^2 + r_{iy}^2 \end{pmatrix}$$

tum3D

computer graphics & visualization

Inertia Tensor in 3D

Definition

In 3D the inertia tensor is a 3×3 matrix:

$$\Phi = \sum_i m_i \begin{pmatrix} r_{iy}^2 + r_{iz}^2 & -r_{ix}\, r_{iy} & -r_{ix}\, r_{iz} \\ -r_{ix}\, r_{iy} & r_{ix}^2 + r_{iz}^2 & -r_{iy}\, r_{iz} \\ -r_{ix}\, r_{iz} & -r_{iy}\, r_{iz} & r_{ix}^2 + r_{iy}^2 \end{pmatrix}$$

The deduction is similar to the 2D case, only more heavy-handed.

tum.3D
computer graphics & visualization

Intuition behind the Inertia Tensor

It is possible to rotate many objects in such a way that Φ is diagonal.

$$\Phi = \begin{pmatrix} \Phi_x & 0 & 0 \\ 0 & \Phi_y & 0 \\ 0 & 0 & \Phi_z \end{pmatrix}$$

$$\mathbf{L} = \Phi\,\boldsymbol{\omega} \Leftrightarrow \boldsymbol{\omega} = \Phi^{-1}\,\mathbf{L}$$

The bigger the diagonal entry the more inert the object is regarding a rotation around that axis.

The smaller Φ is in total, the faster the object's rotation axis can change.

Where We Are

Rigid Body Update Code

```
for each (RigidBody* body in scene->rigidBodies) {
    auto contacts = scene->determineContacts(body);
    auto fieldForces = scene->determineFieldForces(body);
    auto totalForce = computeTotalForce(contacts, fieldForces);
    auto totalMoment = computeTotalMoment(contacts);
    body->linearMomentum.integrate(timeStep, totalForce);
    body->angularMomentum.integrate(timeStep, totalMoment);
    body->velocity = body->linearMomentum / body->mass;
    body->angularVelocity = body->invInertiaTensor *
                                body->angularMomentum;
    body->position.integrate(timeStep, body->velocity);
    body->rotation.integrate(timeStep, body->angularVelocity);
}
```

tum.3D
computer graphics & visualization

Where We Are

Rigid Body Update Code

```
for each (RigidBody* body in scene->rigidBodies) {
    auto contacts = scene->determineContacts(body);
    auto fieldForces = scene->determineFieldForces(body);
    auto totalForce = computeTotalForce(contacts, fieldForces);
    auto totalMoment = computeTotalMoment(contacts);
    body->linearMomentum.integrate(timeStep, totalForce);
    body->angularMomentum.integrate(timeStep, totalMoment);
    body->velocity = body->linearMomentum / body->mass;
    body->angularVelocity = body->invInertiaTensor *
                                    body->angularMomentum;
    body->position.integrate(timeStep, body->velocity);
    body->rotation.integrate(timeStep, body->angularVelocity);
}
```

Questions?

computer graphics & visualization

Bibliography

- "Technische Mechanik 1 & 3" by Gross, Hauger, Schröder & Wall

- "Physically Based Modeling" by Pixar (SIGGRAPH 2001)

- "Mathematics for 3D Game Programming & Computer Graphics" by Eric Lengyel

- "Physics for Game Developers" by David M. Bourg

- "Game Physics" by David H. Eberly

tum.3D
computer graphics & visualization

Andreas Kirsch