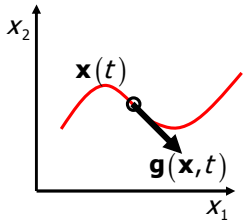
Physically Based Animation

Ordinary Differential Equations
Particle Dynamics
Rigid-Body Dynamics
Collisions

Particle

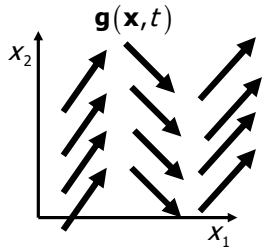
A single particle in 2-D moving in a flow field

- Position $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$
- Velocity $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$, $\mathbf{v} = \frac{d\mathbf{x}}{dt}$
- The flow field function dictates particle velocity $\mathbf{v} = \mathbf{g}(\mathbf{x}, t)$



Vector Field

The flow field $\mathbf{g}(\mathbf{x},t)$ is a vector field that defines a vector for any particle position $\mathbf{x}$ at any time t.



How would a particle move in this vector field?

Differential Equations

The equation $\mathbf{v} = \mathbf{g}(\mathbf{x}, t)$ is a first order differential equation:

$$\frac{d\mathbf{x}}{dt} = \mathbf{g}(\mathbf{x}, t)$$

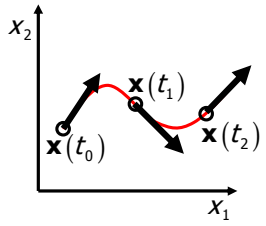
The position of the particle is computed by integrating the differential equation:

$$\mathbf{x}(t) = \mathbf{x}(t_0) + \int_{t_0}^{t} \mathbf{g}(\mathbf{x}, t)\, dt$$

For most interesting cases, this integral cannot be computed analytically.

Numeric Integration

Instead we compute the particle's position by numeric integration: starting at some initial point $\mathbf{x}(t_0)$ we step along the vector field to compute the position at each subsequent time instant. This type of a problem is called an **initial value problem**.



Euler's Method

Euler's method is the simplest solution to an initial value problem. Euler's method starts from the initial value and takes small time steps along the flow:

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t\,\mathbf{g}(\mathbf{x}, t)$$

Why does this work?

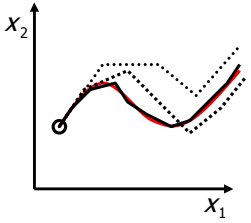
Let's look at a Taylor series expansion of function $\mathbf{x}(t)$:

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t \frac{d\mathbf{x}}{dt} + \frac{\Delta t^2}{2}\frac{d^2\mathbf{x}}{dt^2} + \cdots$$

Disregarding higher-order terms and replacing the first derivative with the flow field function yields the equation for the Euler's method.

Other Methods

Euler's method is the simplest numerical method. The error is proportional to Δt^2. For most cases, the Euler's method is inaccurate and unstable requiring very small steps.



Other methods:

- Midpoint (2nd order Runge-Kutta)
- Higher order Runge-Kutta (4th order, 6th order)
- Adams
- Adaptive Stepsize

Particle in a Force Field

What is a motion of a particle in a force field?

The particle moves according to Newton's Law:

$$\frac{d^2\mathbf{x}}{dt^2} = \frac{\mathbf{f}}{m} \quad (\mathbf{f} = ma)$$

The mass m of a particle describes the particle's inertial properties: heavier particles are easier to move than lighter particles. In general, the force field $\mathbf{f}(\mathbf{x}, \mathbf{v}, t)$ may depend on the time t and particle's position $\mathbf{x}$ and velocity $\mathbf{v}$.

Second-Order Differential Equations

Newton's Law yields an ordinary differential equation of second order:

$$\frac{d^2\mathbf{x}(t)}{dt^2} = \frac{\mathbf{f}(\mathbf{x},\mathbf{v},t)}{m}$$

A clever trick allows us to reuse the same numeric differentiation solvers for first-order differential equations. If we define a new phase space vector **y**, which consists of particle's position **x** and velocity **v**, then we can construct a new first-order differential equation whose solution will also solve the second-order differential equation.

$$\mathbf{y} = \begin{bmatrix}\mathbf{x}\\ \mathbf{v}\end{bmatrix}, \quad \frac{d\mathbf{y}}{dt} = \begin{bmatrix}d\mathbf{x}/dt\\ d\mathbf{v}/dt\end{bmatrix} = \begin{bmatrix}\mathbf{v}\\ \mathbf{f}/m\end{bmatrix}$$

Particle Systems

How would we compute the motion of a particle system consisting of *n* particles?

Concatenating the phase space positions of all *n* particles yields a large first-order ordinary differential equation. For particles in 3-D, there will be 6*n* equations (3 equations for position and 3 equations for velocity of each particle):

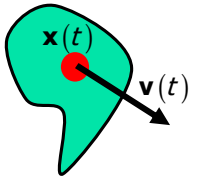
$$\begin{bmatrix}x_1^1\\ x_2^1\\ x_3^1\\ v_1^1\\ v_2^1\\ v_3^1\end{bmatrix} \xrightarrow{\frac{d}{dt}} \begin{bmatrix}\mathbf{x}_1\\ \mathbf{v}_1\\ \mathbf{x}_2\\ \mathbf{v}_2\\ \vdots\\ \mathbf{x}_n\\ \mathbf{v}_n\end{bmatrix} = \begin{bmatrix}\mathbf{v}_1\\ \mathbf{f}_1/m\\ \mathbf{v}_2\\ \mathbf{f}_2/m\\ \vdots\\ \mathbf{v}_n\\ \mathbf{f}_n/m_n\end{bmatrix}$$

Particle Animation

```
AnimateParticles(n, y0, t0, tf)
{
        y = y0
        t = t0
        DrawParticles(n, y)
        while(t != tf) {
                f = ComputeForces(y, t)
                dydt = AssembleDerivative(y, f)
                {y, t} = ODESolverStep(6n, y, dy/dt)
                DrawParticles(n, y)
        }
}
```


Rigid-Body Dynamics

We could compute the motion of a rigid-body by computing the motion of all constituent particles. However, a rigid body does not deform and position of few of its particles is sufficient to determine the state of the body in a phase space. We'll start with a special particle located at the body's center of mass.



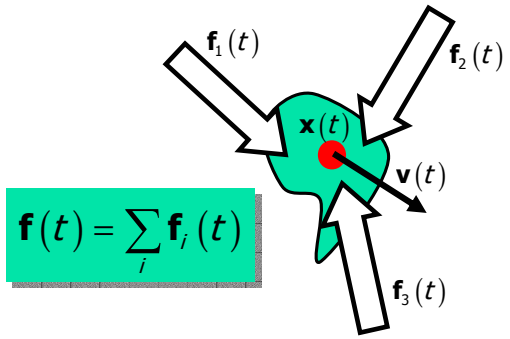
$$\mathbf{y}(t) = \begin{bmatrix}\mathbf{x}(t)\\ ?\\ \mathbf{v}(t)\\ ?\end{bmatrix}$$

Rigid-Body Equation of Motion

The equation of motion describes the dynamics of a rigid body: the motion of a body subjected to external forces. We already know how to write a portion of this coupled first-order differential equation:

$$\frac{d}{dt}\mathbf{y}(t) = \frac{d}{dt}\begin{bmatrix}\mathbf{x}(t)\\ ?\\ M\mathbf{v}(t)\\ ?\end{bmatrix} = \begin{bmatrix}\mathbf{v}(t)\\ ?\\ \mathbf{f}(t)\\ ?\end{bmatrix}$$

Net Force



$$\mathbf{f}(t) = \sum_i \mathbf{f}_i(t)$$

Rest of the State

$$\mathbf{y}(t) = \begin{bmatrix}\mathbf{x}(t)\\ ? \leftarrow \mathbf{R}(t)\\ \mathbf{v}(t)\\ ? \leftarrow \boldsymbol{\omega}(t)\end{bmatrix}$$

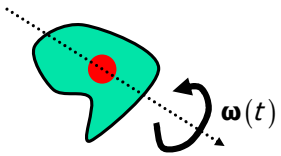
If we knew the orientation of the body, we could compute the position of any body point. Let's represent the orientation with a rotation matrix $\mathbf{R}(t)$. A point $\mathbf{p}^b$ in body coordinates is transformed to world coordinates with the following equation:

$$\mathbf{p}(t) = \mathbf{R}(t)\mathbf{p}^b + \mathbf{x}(t)$$

The last component is the rate of change of the rotation matrix, which we'll call angular velocity $\boldsymbol{\omega}(t)$.

Angular Velocity

Angular velocity is a vector $\boldsymbol{\omega}(t)$ that encodes both the axis of rotation (with its direction) and the speed of rotation (with its speed)*.



How are the orientation matrix $\mathbf{R}(t)$ and the angular velocity $\boldsymbol{\omega}(t)$ related? The obvious relationship is not sensible:

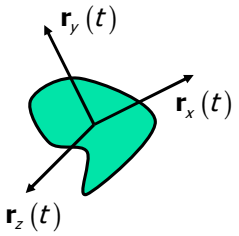
[matrix] $\frac{d}{dt}\mathbf{R}(t) \neq \boldsymbol{\omega}(t)$ [vector]

*This may sound like a quaternion but be careful any vector $\boldsymbol{\omega}$ with magnitude 2π would represent identical orientation. That's a singularity.

Rotation Matrix

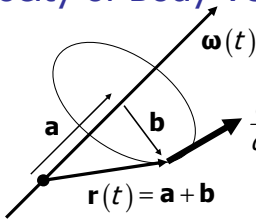
The columns of the rotation matrix describe the world-space directions of the body-space coordinate axes.

$$\mathbf{R}(t) = \begin{bmatrix} | & | & | \\ \mathbf{r}_x(t) & \mathbf{r}_y(t) & \mathbf{r}_z(t) \\ | & | & | \end{bmatrix}$$



If we can determine the rate of change of an arbitrary body vector then we can apply the sample equation to compute the rate of change of each column in the rotation matrix.

Velocity of Body Vector



$$\frac{d}{dt}\mathbf{r}(t) = \boldsymbol{\omega}(t) \times \mathbf{b}$$
$$= \boldsymbol{\omega}(t) \times (\mathbf{a} + \mathbf{b})$$
$$= \boldsymbol{\omega}(t) \times \mathbf{r}(t)$$

The tip of the vector **r** moves with the speed of $|\boldsymbol{\omega}(t)||\mathbf{b}|$ and the direction of the velocity vector is perpendicular to the radius **b** and the axis of rotation $\boldsymbol{\omega}(t)$. Therefore, we can write the velocity vector as the cross product between the angular velocity $\boldsymbol{\omega}(t)$ and the radius vector **b**. Because the vectors $\boldsymbol{\omega}(t)$ and **a** are collinear, we can rewrite the velocity as a function of angular velocity $\boldsymbol{\omega}(t)$ and the original vector $\mathbf{r}(t)$.

Angular Velocity and Orientation

The orientation rate of change and the angular velocity are related by applying the same formula on each column of the rotation matrix. We introduce a special operator to make the expression shorter to write.

$$\frac{d}{dt}\mathbf{R}(t) = \begin{bmatrix} | & | & | \\ \frac{d}{dt}\mathbf{r}_x(t) & \frac{d}{dt}\mathbf{r}_y(t) & \frac{d}{dt}\mathbf{r}_z(t) \\ | & | & | \end{bmatrix}$$
$$= \begin{bmatrix} | & | & | \\ \boldsymbol{\omega}(t) \times \mathbf{r}_x(t) & \boldsymbol{\omega}(t) \times \mathbf{r}_y(t) & \boldsymbol{\omega}(t) \times \mathbf{r}_z(t) \\ | & | & | \end{bmatrix}$$
$$= \boldsymbol{\omega}(t) \odot \mathbf{R}(t)$$

Rigid-Body Equation of Motion

Need to relate rate of change of angular velocity to applied forces. This term must depend on the mass distribution.

$$\frac{d}{dt}\mathbf{y}(t) = \frac{d}{dt}\begin{bmatrix} \mathbf{x}(t) \\ \mathbf{R}(t) \\ M\mathbf{v}(t) \\ \boldsymbol{\omega}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{v}(t) \\ \omega(t) \odot \mathbf{R}(t) \\ \mathbf{f}(t) \\ ? \end{bmatrix}$$

Inertia Tensor

$$\mathbf{I}(t) = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$$

diagonal terms

$$I_{xx} = M \int_V \left(y^2 + z^2\right) dV$$

off-diagonal terms

$$I_{xy} = M \int_V xy \, dV$$

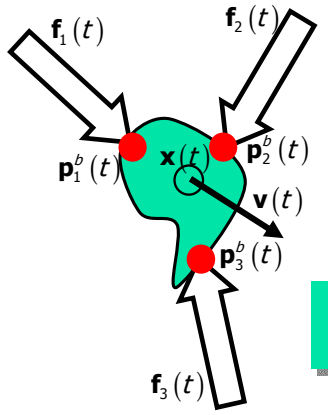
Rigid-Body Equation of Motion

$$\frac{d}{dt}\mathbf{y}(t) = \frac{d}{dt}\begin{bmatrix} \mathbf{x}(t) \\ \mathbf{R}(t) \\ M\mathbf{v}(t) \\ \mathbf{I}(t)\boldsymbol{\omega}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{v}(t) \\ \omega(t) \odot \mathbf{R}(t) \\ \mathbf{f}(t) \\ \boldsymbol{\tau}(t) \end{bmatrix}$$

$M\mathbf{v}(t) \rightarrow$ linear momentum

$\mathbf{I}(t)\boldsymbol{\omega}(t) \rightarrow$ angular momentum

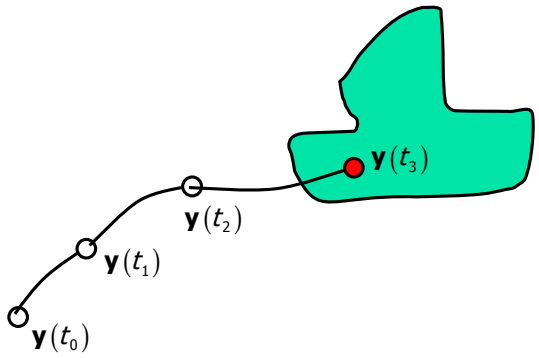
Net Torque



$$\boldsymbol{\tau}(t) = \sum_i \left(\mathbf{p}_i^b - \mathbf{x}(t)\right) \times \mathbf{f}_i(t)$$

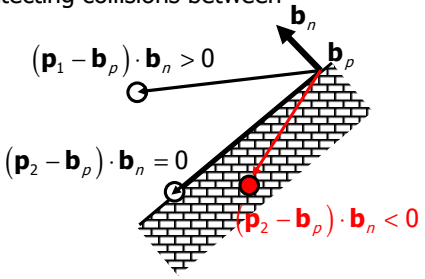
Simulations with Collisions

Simulating motions with collisions requires that we detect them (collision detection) and fix them (collision response).



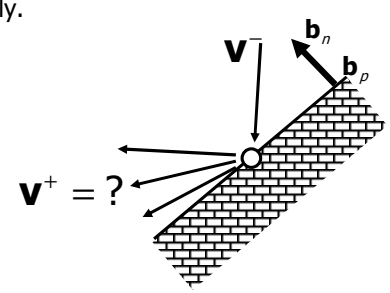
Collision Detection

Plane-Particle collision can be detected by computing the signed-distance function between the boundary and the particle. In this example, collision detection is deceptively simple. Fast collision detection between many curved surface is a difficult problem. Detecting collisions between deformable surfaces is even harder.



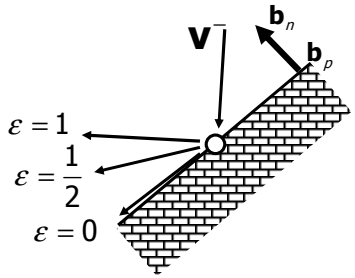
Collision Response

The mechanics of collisions are complicated and many mathematical models have been developed. We'll just look at a one simple model, which assumes that when the collision occurs the two bodes exchange collision impulse instantaneously.



Frictionless Collision Model

$$\mathbf{b}_n \cdot \mathbf{v}^+ = -\varepsilon\left(\mathbf{b}_n \cdot \mathbf{v}^-\right)$$



Animation

```
Animate(y0, t0, tf)
{
        y = y0
        t = t0
        Draw(y)
        while(t != tf) {
                if (Collision(y)) {
                        t = CollisionTime(y)
                        y = ApplyImpulse(y)
                }
                f = ComputeForces(y, t)
                dydt = AssembleDerivative(y, f)
                {y, t} = ODESolverStep(y, dy/dt)
                Draw(y)
        }
}
```

Animation Design

Suppose we'd like to animate a hat flying through the air and landing on a coatrack.

We can compute the hat's motion by techniques described in this lecture, but this won't be enough. We also have to discover what initial position and velocity will result in the hat landing on the coatrack.

Next Time

Controlling Animation