Review

CSE169: Computer Animation

Instructor: Steve Rotenberg

UCSD, Winter 2017

Final

- The final is on Thursday, March 23, at 7:00 pm
- There will be 15 questions and worth 15% of the total grade
- It will cover the material from the second half of the quarter, but may use math we learned throughout the whole quarter

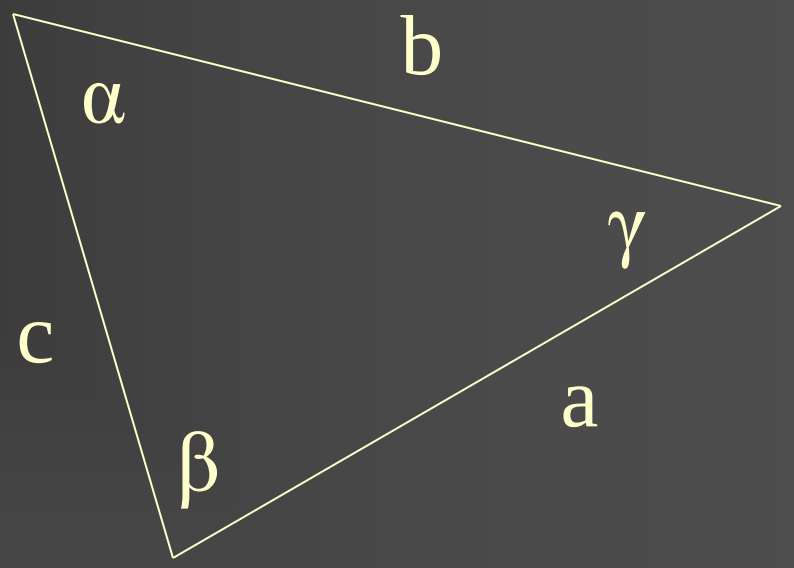
IK: Analytical Methods

- For specific chain configurations, one can implement custom analytical IK solvers.
- Analytical solvers rely on heuristics and direct solutions to matrix and trigonometric equations (i.e., they use a lot of matrix inversion and inverse trig functions)
- These methods can be extremely fast and well behaved, but their main drawback is their lack of generality. They also become more and more difficult to implement as the chains become more complex.

IK: Laws of Sines and Cosines

- Law of Sines:

$$\frac{a}{\sin\alpha} = \frac{b}{\sin\beta} = \frac{c}{\sin\gamma}$$



- Law of Cosines:

$$c^2 = a^2 + b^2 - 2ab\cos\gamma$$

Locomotion

- Locomotion (walking, running, turning…) is one of the most essential animation processes for any character, so it is really nice to be able to understand and automate
- A gait is a particular sequence of lifting and placing the feet during locomotion
- Gaits are described by various parameters such as their period, number of legs, and the step timings of each leg
- Common quadruped gaits include walk, trot, canter, gallop, and pace. Hexapod gaits include the back to front wave gait and tripod gait.
- Other types of locomotion include swimming, flying, gliding, brachiation (swinging), slithering (snakes…), and more…

Gaits

- A *gait* refers to a particular sequence of lifting and placing the feet during legged locomotion (gallop, trot, walk, run…)
- Each repetition of the sequence is called a *gait cycle*
- The time taken in one complete cycle is the *gait period*
- The inverse of the period is the *gait frequency* (1/period)
- Normally, in one gait cycle, each leg goes through exactly one complete step cycle

Gait Phase

- We can think of the *gait phase* a value that ranges from 0 to 1 as the gait cycle proceeds
- We can choose 0 as being any arbitrary point within the cycle (such as when the back left foot begins its step)
- The phase is like a clock that keeps going round and round (0…1, 0…1, 0…1)
- For a particular gait, the stepping of the legs and all other motion of the character can be described relative to the gait phase

Step Cycle

- In one gait cycle, each individual leg goes through a complete *step cycle*
- Each leg's step cycle is phase shifted relative to the main gait cycle
- The step cycle is broken into two main stages
  - *Support stage* (foot on ground)
  - *Transfer stage* (foot in the air)
- The amount of time a leg spends in the support stage is the *support duration* (& likewise for *transfer duration*)

$$SupportDuration + TransferDuration = GaitPeriod$$

Duty Factor

- The relative amount of time a foot spends on the ground is called the *duty factor*

$$DutyFactor = \frac{SupportDuration}{GaitPeriod}$$

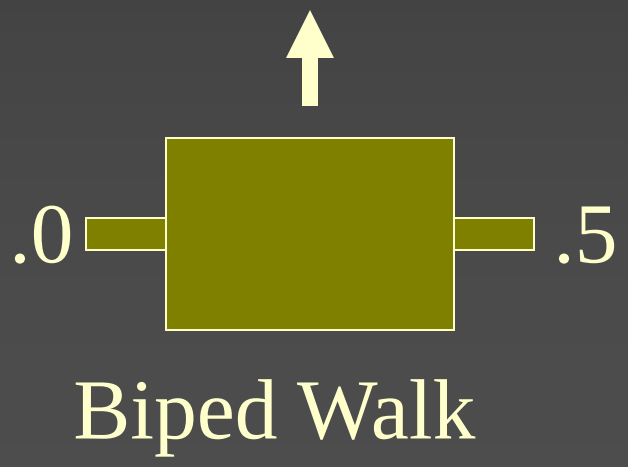
- For a human walking, the duty factor will be greater than 0.5, indicating that there is an overlap time when both feet are on the ground
- For a run, the duty factor is less than 0.5, indicating that there is a time when both feet are in the air and the body is undergoing ballistic motion

Step Phase

- The *step phase* is a value that ranges from 0 to 1 during an individual leg's step cycle
- We can choose 0 to indicate the moment when the foot begins to lift (i.e., the beginning of the transfer phase)
- The foot contacts the ground and comes to rest when the phase equals 1 minus the duty factor

Step Trigger

- Each leg's step cycle is phase shifted relative to the main gait cycle
- This phase shift is called the *step trigger*
- The trigger is the phase within the main gait cycle where a particular leg begins its step cycle



Particle Systems

Kinematics of Particles

- We will define an individual particle's 3D position over time as **r**(t)
- By definition, the velocity is the first derivative of position, and acceleration is the second

$$\mathbf{r} = \mathbf{r}(t)$$

$$\mathbf{v} = \frac{d\mathbf{r}}{dt}$$

$$\mathbf{a} = \frac{d\mathbf{v}}{dt} = \frac{d^2\mathbf{r}}{dt^2}$$

Mass and Momentum

- We can associate a mass m with each particle. We will assume that the mass is constant

$$m = m_0$$

- We will also define a vector quantity called momentum (**p**), which is the product of mass and velocity

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

Newton's First Law

- Newton's First Law states that a body in motion will remain in motion and a body at rest will remain at rest- unless acted upon by some force
- This implies that a free particle moving out in space will just travel in a straight line

$$\mathbf{a} = 0$$

$$\mathbf{v} = \mathbf{v}_0 \qquad \mathbf{p} = \mathbf{p}_0 = m\mathbf{v}_0$$

$$\mathbf{r} = \mathbf{r}_0 + \mathbf{v}_0 t$$

Force

- Force is defined as the rate of change of momentum

$$\mathbf{f} = \frac{d\mathbf{p}}{dt}$$

- We can expand this out:

$$\mathbf{f} = \frac{d(m\mathbf{v})}{dt} = \frac{dm}{dt}\mathbf{v} + m\frac{d\mathbf{v}}{dt} = m\frac{d\mathbf{v}}{dt}$$

$$\mathbf{f} = m\mathbf{a}$$

Newton's Second Law

- Newton's Second Law says:

$$\mathbf{f} = \frac{d\mathbf{p}}{dt} = m\mathbf{a}$$

- This relates the kinematic quantity of acceleration to the physical quantity of force

Newton's Third Law

- Newton's Third Law says that any force that body A applies to body B will be met by an equal and opposite force from B to A

$$\mathbf{f}_{AB} = -\mathbf{f}_{BA}$$

- Put another way: every action has an equal and opposite reaction
- This is very important when combined with the second law, as the two together imply the conservation of momentum

Conservation of Momentum

- Any gain of momentum by a particle must be met by an equal and opposite loss of momentum by another particle. Therefore, the total momentum in a closed system will remain constant
- We will not always explicitly obey this law, but we will implicitly obey it
- In other words, we may occasionally apply forces without strictly applying an equal and opposite force to anything, but we will justify it when we do

Forces on a Particle

- Usually, a particle will be subjected to several simultaneous vector forces from different sources
- All of these forces simply add up to a single total force acting on the particle

$$\mathbf{f}_{total} = \sum \mathbf{f}_i$$

Particle

$\mathbf{r}$: *position*

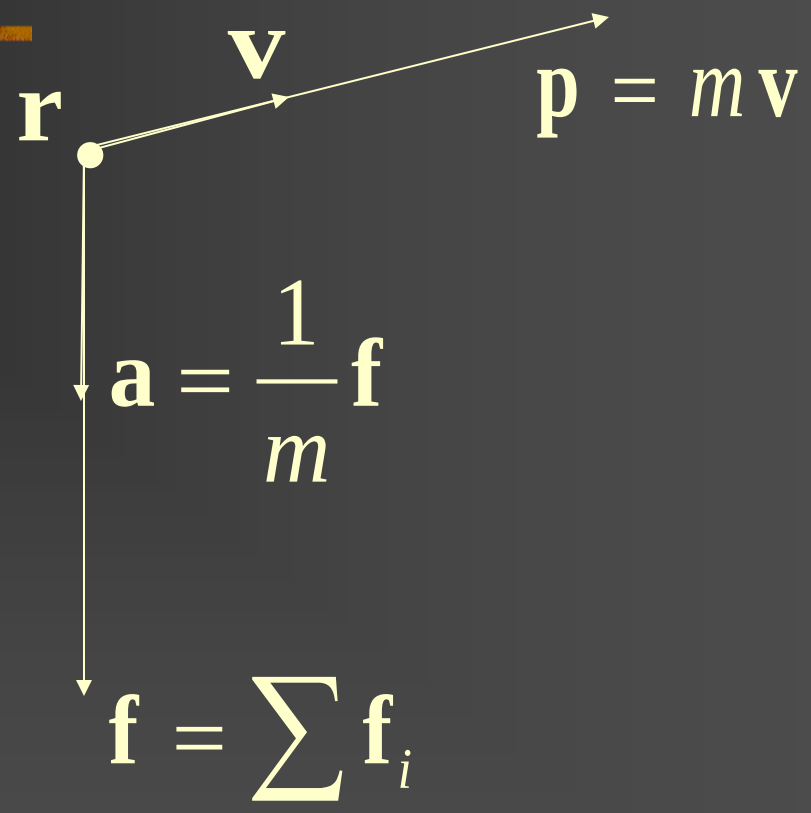
$\mathbf{v}$: *velocity*

$\mathbf{a}$: *acceleration*

m : *mass*

$\mathbf{p}$: *momentum*

$\mathbf{f}$: *force*



Particle Simulation

1. Compute all forces acting within the system in the current configuration (making sure to obey Newton's third law)
2. Compute the resulting acceleration for each particle (**a**=**f**/m) and integrate over some small time step to get new positions
3. Check for collisions and correct positions & velocities as necessary

- Repeat

General Newtonian Simulation

- Many types of simulations can be fit into this overall approach:

1. Compute Forces
2. Integrate Motion
3. Enforce Constraints

- Repeat

- Note that ‘constraints’ may include various things like collisions, articulations, or geometric properties such as fluid incompressibility

Cloth Simulation

1. Compute Forces
   - For each particle: Apply gravity
   - For each spring-damper: Compute & apply forces
   - For each triangle: Compute & apply aerodynamic forces

2. Integrate Motion
   - For each particle: Apply forward Euler integration

3. Enforce Constraints
   - For each particle: Check for collisions with ground
     and apply position correction & impulse

Forward Euler Integration

- Forward Euler integration is about the simplest possible way to do numerical integration

$$x_{n+1} = x_n + x'_n \Delta t$$

- It works by treating the linear slope of the derivative at a particular value as an approximation to the function at some nearby value
- The gradient descent algorithm we used for inverse kinematics used Euler integration

Forward Euler Integration

- For particles, we are actually integrating twice to get the position

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{a}_n \Delta t$$

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_{n+1} \Delta t$$

which expands to

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \left(\mathbf{v}_n + \mathbf{a}_n \Delta t\right)\Delta t$$

$$= \mathbf{r}_n + \mathbf{v}_n \Delta t + \mathbf{a}_n \left(\Delta t\right)^2$$

Euler Integration

- Once we've computed all of the forces in the system, we can use Newton's Second Law (f=ma) to compute the acceleration

$$\mathbf{a}_n = \frac{1}{m}\mathbf{f}_n$$

- Then, we use the acceleration to advance the simulation forward by some time step Δt, using the simple Euler integration scheme

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{a}_n \Delta t$$

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_{n+1} \Delta t$$

Forces

Uniform Gravity

- If we are near the Earth's surface, we can think of the ground as a flat plane (instead of a big sphere) and treat gravity as a constant downward acceleration

$$\mathbf{f}_{gravity} = m\mathbf{g}_0$$

$$\mathbf{g}_0 = \begin{bmatrix} 0 & -9.8 & 0 \end{bmatrix} \quad \frac{m}{s^2}$$

Non-Uniform Gravity

- If we are far away enough from the objects such that the inverse square law of gravity is noticeable, we can use Newton's Law of Gravitation:

$$\mathbf{f}_{gravity} = \frac{Gm_1m_2}{d^2}\mathbf{e} \qquad \mathbf{e} = \frac{\mathbf{r}_1 - \mathbf{r}_2}{\left|\mathbf{r}_1 - \mathbf{r}_2\right|}$$

$$G = 6.673 \times 10^{-11} \qquad \frac{m^3}{kg \cdot s^2}$$

Non-Uniform Gravity

- The law describes an equal and opposite force exchanged between two bodies, where the force is proportional to the product of the two masses and inversely proportional to their distance squared. The force acts in a direction **e** along a line from one particle to the other (in an attractive direction)

$$\mathbf{f}_{gravity} = \frac{G m_1 m_2}{d^2}\mathbf{e} \qquad \mathbf{e} = \frac{\mathbf{r}_1 - \mathbf{r}_2}{\left|\mathbf{r}_1 - \mathbf{r}_2\right|}$$

Aerodynamic Drag

- Aerodynamic interactions are actually very complex and difficult to model accurately
- We can use a reasonable simplification to describe the total aerodynamic drag force on an object:

$$\mathbf{f}_{aero} = \frac{1}{2}\rho\left|\mathbf{v}\right|^2 c_d a\mathbf{e} \qquad \mathbf{e} = -\frac{\mathbf{v}}{\left|\mathbf{v}\right|}$$

- Where ρ is the density of the air (or water…), c_d is the coefficient of drag for the object, a is the cross sectional area of the object, and **e** is a unit vector in the opposite direction of the velocity

Aerodynamic Force

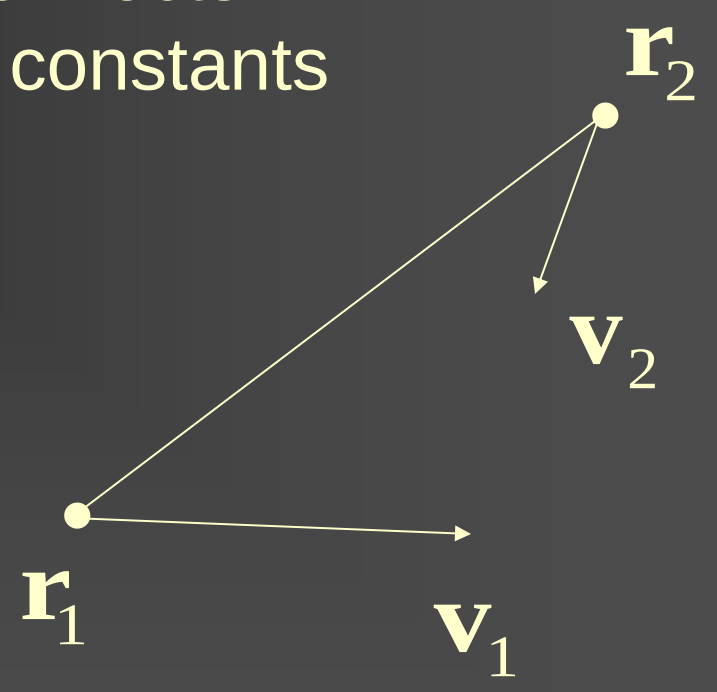
- If we want to compute the aerodynamic force on a flat surface with normal **n**, we can use:

$$\mathbf{f}_{aero} = -\frac{1}{2}\rho\left|\mathbf{v}\right|^2 c_d a\mathbf{n}$$

- Instead of opposing the velocity, the force pushes against the normal of the surface
- Note: This is a major simplification of real aerodynamic interactions, but it's a good place to start

Spring-Dampers

- The basic spring-damper connects two particles and has three constants defining its behavior
  - Rest length: l_0
  - Spring constant: k_s
  - Damping factor: k_d



Spring-Dampers

- The basic linear spring force in one dimension is:

$$f_{spring} = -k_s x = -k_s \left(l_0 - l \right)$$

- The linear damping force is:

$$f_{damp} = -k_d v = -k_d \left(v_1 - v_2 \right)$$

- We can define a spring-damper by just adding the two:

$$f_{sd} = -k_s \left(l_0 - l \right) - k_d \left(v_1 - v_2 \right)$$

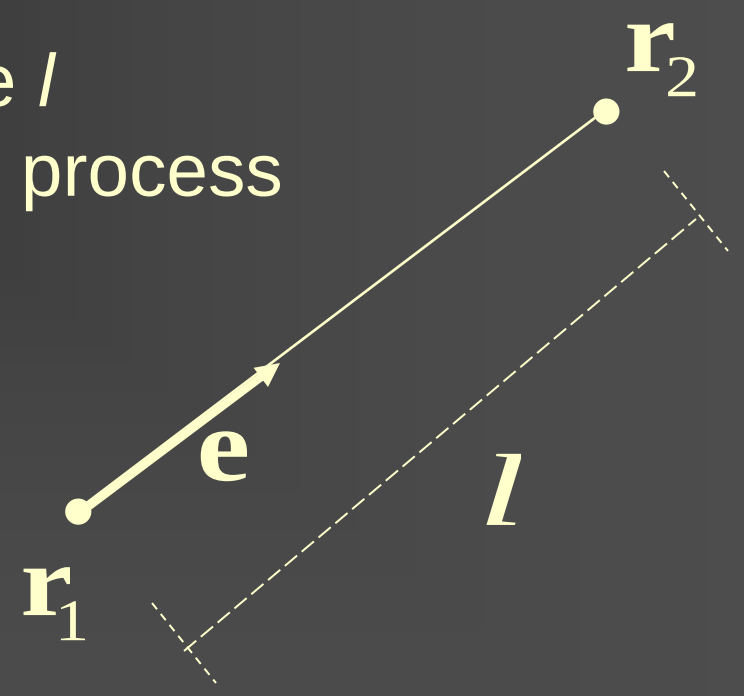
Spring-Damper Force

- We start by computing the unit length vector **e** from $\mathbf{r}_1$ to $\mathbf{r}_2$
- We can compute the distance *l* between the two points in the process

$$\mathbf{e}^* = \mathbf{r}_2 - \mathbf{r}_1$$

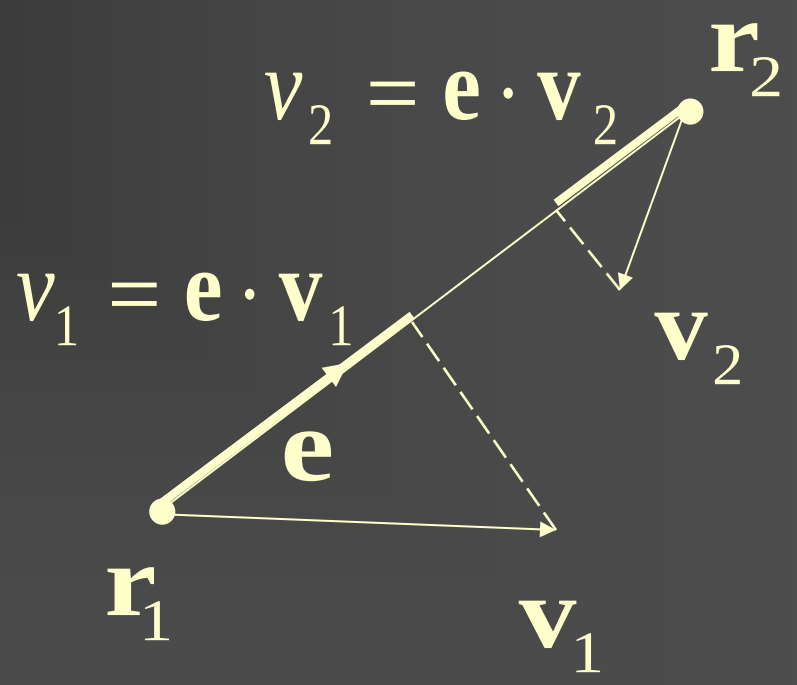
$$l = \left|\mathbf{e}^*\right|$$

$$\mathbf{e} = \frac{\mathbf{e}^*}{l}$$



Spring-Dampers

- Next, we find the 1D velocities



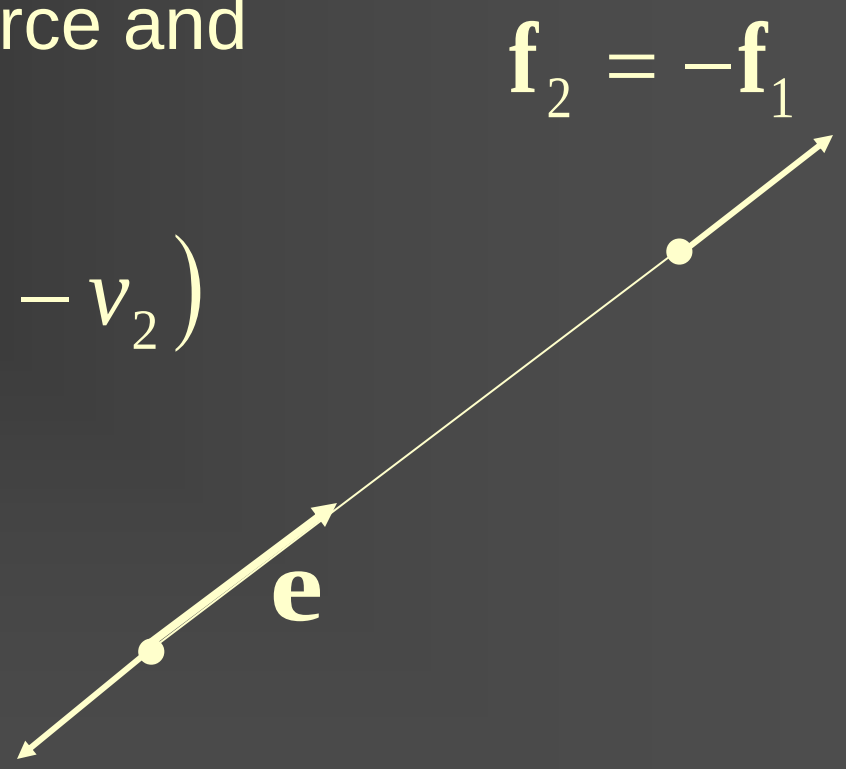
Spring-Dampers

- Now, we can find the 1D force and map it back into 3D

$$f_{sd} = -k_s\left(l_0 - l\right) - k_d\left(v_1 - v_2\right)$$

$$\mathbf{f}_1 = f_{sd}\mathbf{e}$$

$$\mathbf{f}_2 = -\mathbf{f}_1$$



$$\mathbf{f}_1 = f_{sd}\mathbf{e}$$

Friction

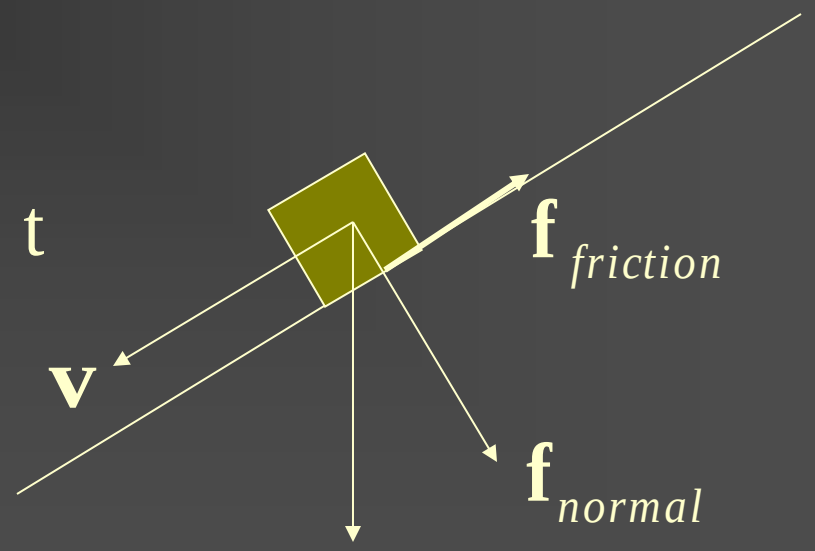
- The Coulomb friction model says:

$$\mathbf{f}_{dynamic} = \mu_d \left|\mathbf{f}_{normal}\right| \mathbf{e}$$

$$\mathbf{f}_{static} \leq \mu_s \left|\mathbf{f}_{normal}\right| \mathbf{e}$$

μ_d : dynamic friction coefficient

μ_s : static friction coefficient



Rigid Bodies

Cross Product & Hat Operator

$$\mathbf{a} \times \mathbf{b} = [a_y b_z - a_z b_y \quad a_z b_x - a_x b_z \quad a_x b_y - a_y b_z]$$

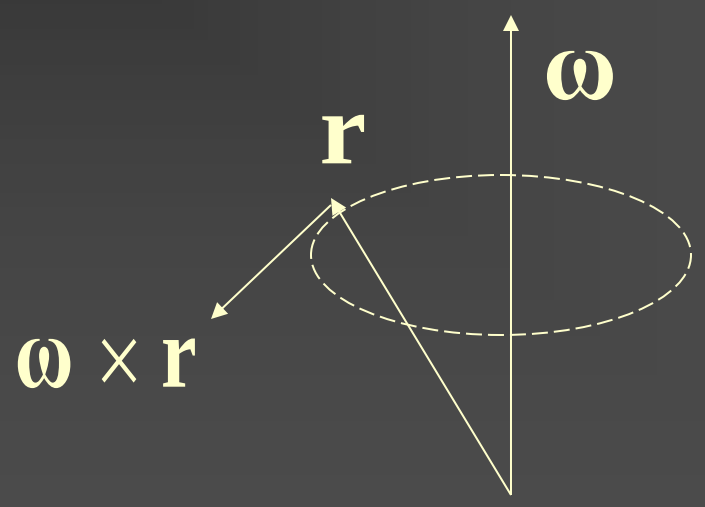
$$\mathbf{a} \times \mathbf{b} = \hat{\mathbf{a}} \cdot \mathbf{b}$$

$$\hat{\mathbf{a}} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}$$

Derivative of a Rotating Vector

- Let's say that vector **r** is rotating around the origin, maintaining a fixed distance
- At any instant, it has an angular velocity of **ω**

$$\frac{d\mathbf{r}}{dt} = \boldsymbol{\omega} \times \mathbf{r}$$



Derivative of Rotating Matrix

- If matrix **A** is a rigid 3x3 matrix rotating with angular velocity **ω**
- This implies that the **a**, **b**, and **c** axes must be rotating around **ω**
- The derivatives of each axis are **ω**x**a**, **ω**x**b**, and **ω**x**c**, and so the derivative of the entire matrix is:

$$\frac{d\mathbf{A}}{dt} = \boldsymbol{\omega} \times \mathbf{A} \qquad = \hat{\boldsymbol{\omega}} \cdot \mathbf{A}$$

Product Rule

- The product rule of differential calculus can be extended to vector and matrix products as well

$$\frac{d(\mathbf{a}\cdot\mathbf{b})}{dt}=\frac{d\mathbf{a}}{dt}\cdot\mathbf{b}+\mathbf{a}\cdot\frac{d\mathbf{b}}{dt}$$

$$\frac{d(\mathbf{a}\times\mathbf{b})}{dt}=\frac{d\mathbf{a}}{dt}\times\mathbf{b}+\mathbf{a}\times\frac{d\mathbf{b}}{dt}$$

$$\frac{d(\mathbf{A}\cdot\mathbf{B})}{dt}=\frac{d\mathbf{A}}{dt}\cdot\mathbf{B}+\mathbf{A}\cdot\frac{d\mathbf{B}}{dt}$$

Eigenvalue Equation

- Lets say we have a known matrix **M** and we want to know if there is any vector **x** and scalar *s* such that

$$\mathbf{M}\mathbf{x} = s\mathbf{x}$$

- This is known as an eigenvalue equation, and for a NxN matrix, there should be up to N eigenvectors $\mathbf{x}_i$ and N eigenvalues s_i that satisfy the equation
- If **M** is a symmetric matrix (i.e., $\mathbf{M}^T = \mathbf{M}$) then all of the eigenvalues will be real numbers and the eigenvectors will be real, orthonormal vectors (otherwise, some of the eigenvalues/eigenvectors will be complex)

Symmetric Matrix

- If we have a symmetric matrix $\mathbf{M}$, we can *diagonalize* it:

$$\mathbf{M}_0 = \mathbf{A}^T \cdot \mathbf{M} \cdot \mathbf{A}$$

- Where $\mathbf{M}_0$ is a diagonal matrix and $\mathbf{A}$ is an orthonormal (pure rotation) matrix
- The columns of $\mathbf{A}$ are the eigenvectors of $\mathbf{M}$ and the diagonal elements in $\mathbf{M}_0$ are the corresponding eigenvalues
- The *symmetric Jacobi algorithm* is a simple and effective matrix algorithm for computing this diagonalization

Angular Momentum

- The linear momentum of a particle is $\mathbf{p} = m\mathbf{v}$
- We define the *moment of momentum* (or *angular momentum*) of a particle at some offset **r** as the vector $\mathbf{L} = \mathbf{r} \times \mathbf{p}$
- Like linear momentum, angular momentum is conserved in a mechanical system
- If the particle is constrained only to rotate so that the direction of **r** is changing but the length is not, we can re-express its velocity as a function of angular velocity $\boldsymbol{\omega}$:

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

- This allows us to re-express **L** as a function of $\boldsymbol{\omega}$:

$$\mathbf{L} = \mathbf{r} \times \mathbf{p} = \mathbf{r} \times (m\mathbf{v}) = m\mathbf{r} \times \mathbf{v} = m\mathbf{r} \times (\boldsymbol{\omega} \times \mathbf{r})$$
$$\mathbf{L} = -m\mathbf{r} \times (\mathbf{r} \times \boldsymbol{\omega})$$
$$\mathbf{L} = -m\hat{\mathbf{r}} \cdot \hat{\mathbf{r}} \cdot \boldsymbol{\omega}$$

Rotational Inertia

$$\mathbf{L} = -m\hat{\mathbf{r}} \cdot \hat{\mathbf{r}} \cdot \boldsymbol{\omega}$$

- We can re-write this as:

$$\mathbf{L} = \mathbf{I} \cdot \boldsymbol{\omega} \quad where \quad \mathbf{I} = -m\hat{\mathbf{r}} \cdot \hat{\mathbf{r}}$$

- We’ve introduced the *rotational inertia matrix* $\mathbf{I}$, which relates the angular momentum of a rotating particle to its angular velocity

Rigid Body Rotational Inertia

$$\mathbf{I} = \begin{bmatrix} \int \rho\left(r_y^2 + r_z^2\right) d\Omega & -\int \rho r_x r_y \, d\Omega & -\int \rho r_x r_z \, d\Omega \\ -\int \rho r_x r_y \, d\Omega & \int \rho\left(r_x^2 + r_z^2\right) d\Omega & -\int \rho r_y r_z \, d\Omega \\ -\int \rho r_x r_z \, d\Omega & -\int \rho r_y r_z \, d\Omega & \int \rho\left(r_x^2 + r_y^2\right) d\Omega \end{bmatrix}$$

$$\mathbf{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix}$$

Rotational Inertia

- The rotational inertia matrix $\mathbf{I}$ is a 3x3 symmetric matrix that is essentially the rotational equivalent of mass
- It relates the angular momentum of a system to its angular velocity by the equation

$$\mathbf{L} = \mathbf{I} \cdot \boldsymbol{\omega}$$

- This is similar to how mass relates linear momentum to linear velocity, but rotation adds additional complexity

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

Rotational Inertia

- The center of mass of a rigid body behaves like a particle- it has position, velocity, momentum, etc., and it responds to forces through **f**=*m***a**
- Rigid bodies also add properties of rotation. These behave in a similar fashion to the translational properties, but the main difference is in the velocity-momentum relationships:

$$\mathbf{p} = m\mathbf{v} \quad vs. \quad \mathbf{L} = \mathbf{I}\boldsymbol{\omega}$$

- We have a vector **p** for linear momentum and vector **L** for angular momentum
- We also have a vector **v** for linear velocity and vector $\boldsymbol{\omega}$ for angular velocity
- In the linear case, the velocity and momentum are related by a single scalar *m*, but in the angular case, they are related by a matrix $\mathbf{I}$
- This means that linear velocity and linear momentum always line up, but angular velocity and angular momentum don't
- Also, as $\mathbf{I}$ itself changes as the object rotates, the relationship between $\boldsymbol{\omega}$ and **L** changes
- This means that a constant angular momentum may result in a non-constant angular velocity, thus resulting in the tumbling motion of rigid bodies

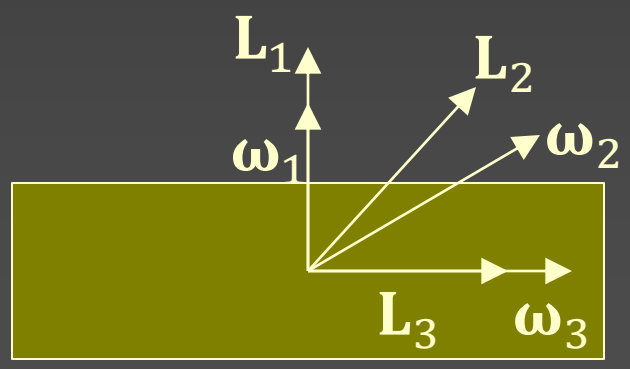
Rotational Inertia

$$\mathbf{L} = \mathbf{I}\boldsymbol{\omega}$$

- Remember eigenvalue equations of the form **Ax**=*b***x** where given a matrix **A**, we want to know if there are any vectors **x** that when transformed by **A** result in a scaled version of the x (i.e., are there vectors who's direction doesn't change after being transformed?)
- A symmetric 3x3 matrix (like $\mathbf{I}$) has 3 real eigenvalues and 3 orthonormal eigenvectors
- If the angular momentum **L** lines up with one of the eigenvectors of $\mathbf{I}$, then $\boldsymbol{\omega}$ will line up with **L** and the angular velocity will be constant
- Otherwise, the angular velocity will be non-constant and we will get tumbling motion
- We call these eigenvectors the *principal axes* of the rigid body and they are constant relative to the geometry of the rigid body
- Usually, we want to align these to the x, y, and z axes when we initialize the rigid body. That way, we can represent the rotational inertia as 3 constants (which happen to be the 3 eigenvalues of $\mathbf{I}$)

Principal Axes

- We see three example angular momentum vectors **L** and their corresponding angular velocities $\boldsymbol{\omega}$, all based on the same rotational inertial matrix $\mathbf{I}$
- We can see that $\mathbf{L}_1$ and $\mathbf{L}_3$ must be aligned with the principal axes, as they result in angular velocities in the same direction as the angular momentum



Principal Axes & Inertias

- If we diagonalize the **I** matrix, we get an orientation matrix **A** and a constant diagonal matrix $\mathbf{I}_o$
- The matrix **A** rotates the object from an orientation where the principal axes line up with the *x*, *y*, and *z* axes
- The three values in $\mathbf{I}_o$, (namely $\mathbf{I}_x$, $\mathbf{I}_y$, and $\mathbf{I}_z$) are the principal inertias. They represent the resistance to torque around the corresponding principal axis (in a similar way that mass represents the resistance to force)

Diagonalization of Rotational Inertial

$$\mathbf{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix}$$

$$\mathbf{I} = \mathbf{A} \cdot \mathbf{I}_0 \cdot \mathbf{A}^T \qquad where \qquad \mathbf{I}_0 = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}$$

Particle Dynamics

$\mathbf{x}$	Position
$\mathbf{v} = \frac{d\mathbf{x}}{dt}$	Velocity
$\mathbf{a} = \frac{d\mathbf{v}}{dt} = \frac{d^2\mathbf{x}}{dt^2}$	Acceleration
m	Mass
$\mathbf{p} = m\mathbf{v}$	Momentum
$\mathbf{f} = \frac{d\mathbf{p}}{dt} = m\mathbf{a}$	Force

Rigid Body Dynamics

$\mathbf{A}$	Orientation (3x3 matrix)
$\boldsymbol{\omega}$	Angular Velocity (vector)
$\bar{\boldsymbol{\omega}} = \frac{d\boldsymbol{\omega}}{dt}$	Angular Acceleration (vector)
$\mathbf{I} = \mathbf{A} \cdot \mathbf{I}_0 \cdot \mathbf{A}^T$	Rotational Inertia (3x3 matrix)
$\mathbf{L} = \mathbf{I}\boldsymbol{\omega}$	Momentum (vector)
$\boldsymbol{\tau} = \frac{d\mathbf{L}}{dt} = \mathbf{r} \times \mathbf{f} = \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega} + \mathbf{I}\bar{\boldsymbol{\omega}}$	Torque (vector)

Newton-Euler Equations

$$\mathbf{f} = m\mathbf{a}$$

$$\boldsymbol{\tau} = \boldsymbol{\omega} \times \mathbf{I} \cdot \boldsymbol{\omega} + \mathbf{I} \cdot \overline{\boldsymbol{\omega}}$$

Rigid Body Simulation

- Each frame, we can apply several forces to the rigid body, that sum up to one total force and one total torque

$$\mathbf{f} = \sum \mathbf{f}_i \qquad \boldsymbol{\tau} = \sum \mathbf{r}_i \times \mathbf{f}_i$$

- We can then integrate the force and torque over the time step to get the new linear and angular momenta

$$\mathbf{p}' = \mathbf{p} + \mathbf{f}\Delta t \qquad \mathbf{L}' = \mathbf{L} + \boldsymbol{\tau}\Delta t$$

- We can then compute the linear and angular velocities from those:

$$\mathbf{v} = \frac{1}{m}\mathbf{p}' \qquad \boldsymbol{\omega} = \mathbf{I}^{-1}\mathbf{L}'$$

- We can now integrate the new position and orientation:

$$\mathbf{x}' = \mathbf{x} + \mathbf{v}\Delta t \qquad \mathbf{A}' = \mathbf{A} \cdot Rotate(\boldsymbol{\omega}\Delta t)$$

Kinematics of an Offset Point

- The kinematic equations for a fixed point on a rigid body are:

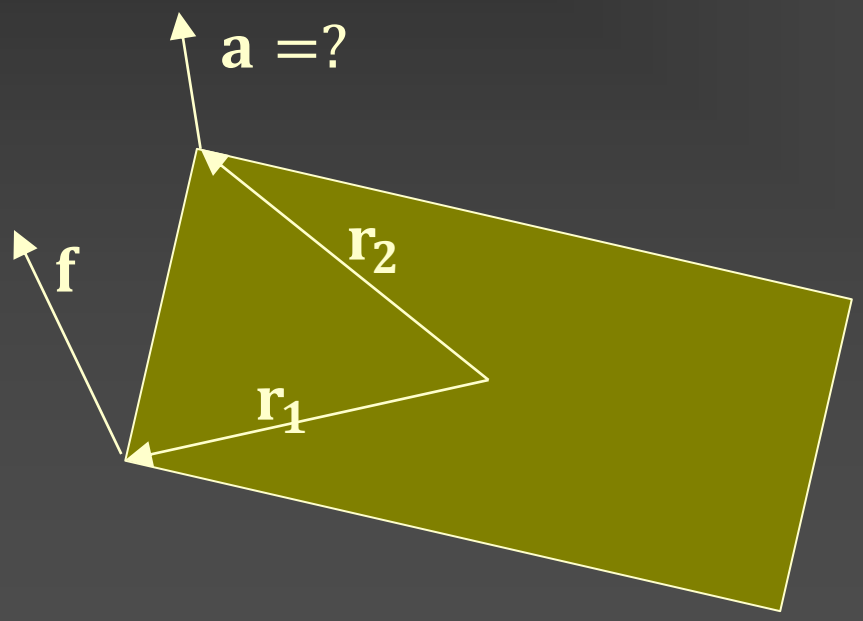
$$\mathbf{x} = \mathbf{x}_{cm} + \mathbf{r}$$

$$\mathbf{v} = \mathbf{v}_{cm} + \boldsymbol{\omega} \times \mathbf{r}$$

$$\mathbf{a} = \mathbf{a}_{cm} + \overline{\boldsymbol{\omega}} \times \mathbf{r} + \boldsymbol{\omega} \times \left(\boldsymbol{\omega} \times \mathbf{r} \right)$$

Offset Forces

- If we apply a force **f** to a rigid body at offset $\mathbf{r}_1$, what is the resulting acceleration at a offset $\mathbf{r}_2$?



Inverse Mass Matrix

$$\mathbf{M}^{-1} = \begin{bmatrix} 1/m & 0 & 0 \\ 0 & 1/m & 0 \\ 0 & 0 & 1/m \end{bmatrix} - \widehat{\mathbf{r}_2} \cdot \mathbf{I}^{-1} \cdot \widehat{\mathbf{r}_1}$$

- We call $\mathbf{M}^{-1}$ an 'inverse mass matrix', (and we can call **M** the mass matrix)
- It lets us apply a force at $\mathbf{r}_1$ and find the resulting acceleration at $\mathbf{r}_2$ in a f=ma format
- It also lets us apply an impulse at $\mathbf{r}_1$ and find the resulting change in velocity
- Note: $\mathbf{r}_1$ can equal $\mathbf{r}_2$, allowing us to find the resulting acceleration at the same offset where we apply the force

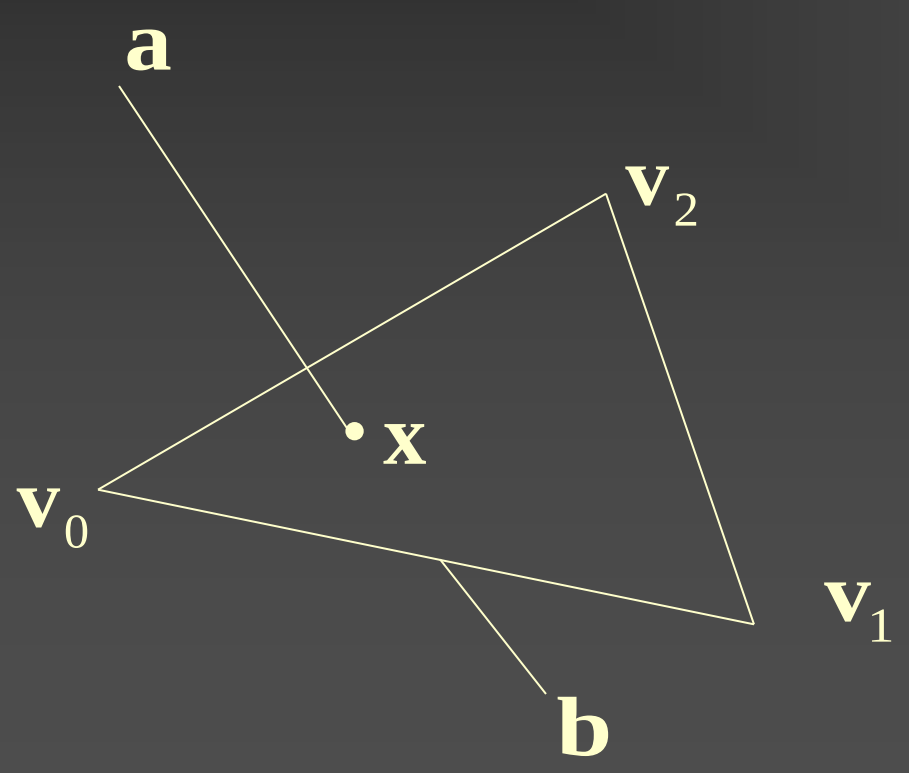
Collisions

Physics: Collisions

- Collision detection is a big part of physics simulation
- Technically, detection of collisions is a geometry problem, while the response to a collision is a physics problem
- For general purpose collision detection, we typically have a pair of moving objects, and we need to determine if they collide, and when and where exactly they hit
- Objects are built up from various primitives such as triangles, spheres, cylinders, etc.
- At the heart of collision detection is primitive-primitive testing. Other important components are optimization data structures (often bounding volume hierarchies) and pair reduction

Segment vs. Triangle

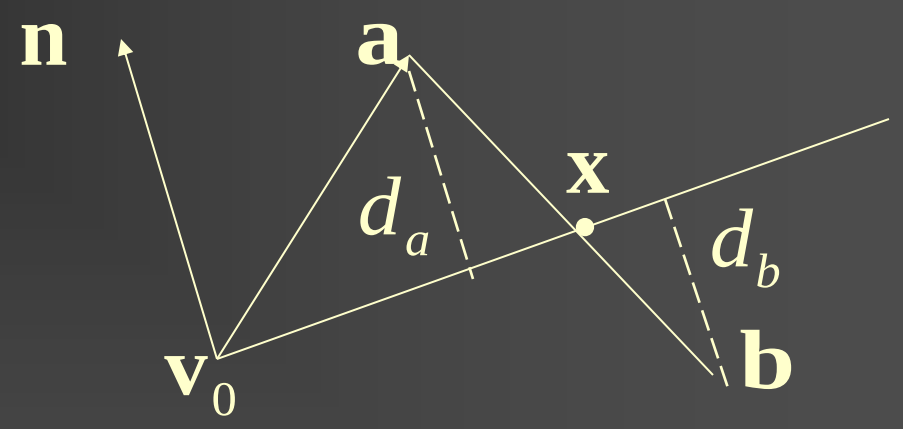
- Does segment **ab** intersect triangle $\mathbf{v}_0\mathbf{v}_1\mathbf{v}_2$?



Segment vs. Triangle

- First, compute signed distances of a and b to plane

$$d_a = (\mathbf{a} - \mathbf{v}_0) \cdot \mathbf{n}$$
$$d_b = (\mathbf{b} - \mathbf{v}_0) \cdot \mathbf{n}$$



- Reject if both are above or both are below triangle
- Otherwise, find intersection point **x**

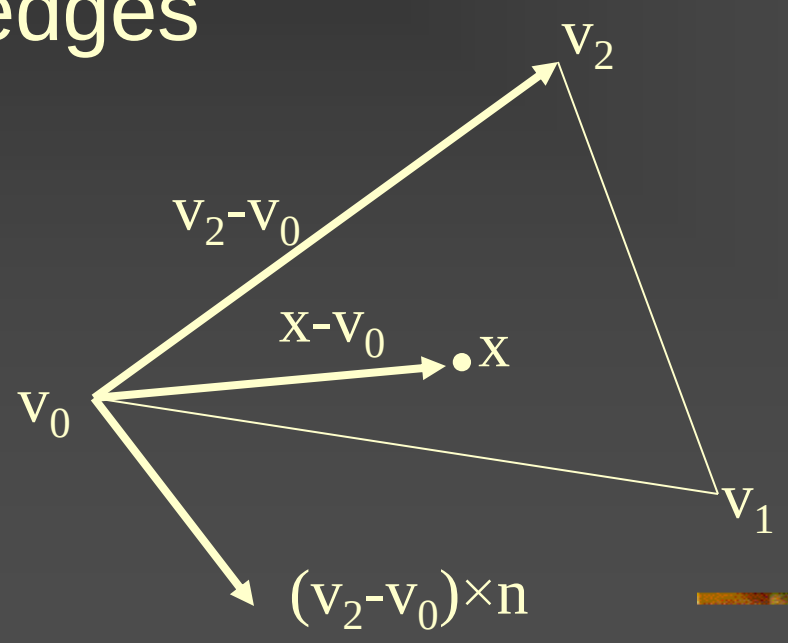
$$\mathbf{x} = \frac{d_a \mathbf{b} - d_b \mathbf{a}}{d_a - d_b}$$

Segment vs. Triangle

- Is point x inside the triangle?

$$(x-v_0)\cdot((v_2-v_0)\times n) > 0$$

- Test all 3 edges



Optimization Structures

- BV, BVH (bounding volume hierarchies)
  - Octree
  - KD tree
  - BSP (binary separating planes)
  - OBB tree (oriented bounding boxes- a popular form of BVH)
- Uniform grid
- Hashing
- Dimension reduction

Pair Reduction

- At a minimum, any moving object should have some sort of bounding sphere (or other simple primitive)
- Before a pair of objects is tested in any detail, we can quickly test if their bounding spheres intersect
- When there are lots of moving objects, even this quick bounding sphere test can take too long, as it must be applied N^2 times if there are N objects
- Reducing this N^2 problem is called *pair reduction*
- Pair testing isn't a big issue until N>50 or so…
- Note that the spatial hash table we discussed in the SPH lecture is used for pair reduction

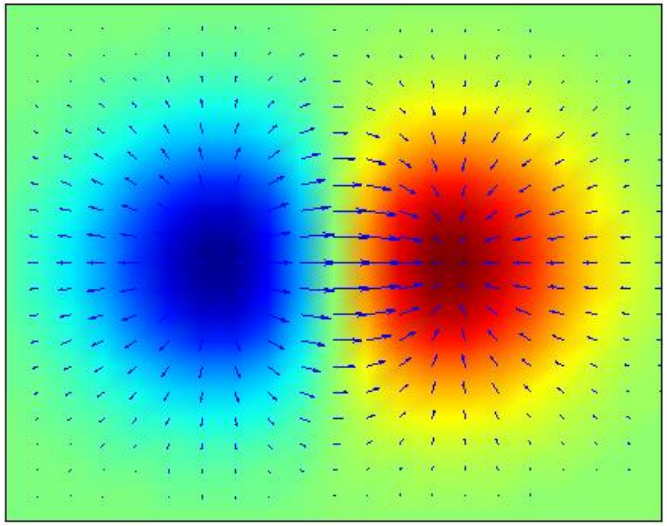
Fluid Dynamics

Gradient

- The *gradient* is a generalization of the concept of a derivative

$$\nabla s = \begin{bmatrix} \frac{\partial s}{\partial x} & \frac{\partial s}{\partial y} & \frac{\partial s}{\partial z} \end{bmatrix}$$

- When applied to a *scalar* field, the result is a *vector* pointing in the direction the field is increasing
- In 1D, this reduces to the standard derivative (slope)



Divergence

- The *divergence* of a vector field is a measure of how much the vectors are expanding

$$\nabla \cdot \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}$$

- For example, when air is heated in a region, it will locally expand, causing a positive divergence in the area of expansion
- The divergence operator works on a vector field and produces a scalar field as a result

Curl

- The *curl* operator produces a new vector field that measures the rotation of the original vector field

$$\nabla \times \mathbf{v} = \left[\frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z} \quad \frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x} \quad \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}\right]$$

- For example, if the air is circulating in a particular region, then the curl in that region will represent the axis of rotation
- The magnitude of the curl is twice the angular velocity of the vector field

Laplacian

- The *Laplacian* operator is a measure of the second derivative of a scalar or vector field

$$\nabla^2 = \nabla \cdot \nabla = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$$

- Just as in 1D where the second derivative relates to the curvature of a function, the Laplacian relates to the curvature of a field
- The Laplacian of a scalar field is another scalar field:

$$\nabla^2 s = \frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} + \frac{\partial^2 s}{\partial z^2}$$

- And the Laplacian of a vector field is another vector field

$$\nabla^2 \mathbf{v} = \frac{\partial^2 \mathbf{v}}{\partial x^2} + \frac{\partial^2 \mathbf{v}}{\partial y^2} + \frac{\partial^2 \mathbf{v}}{\partial z^2}$$

Del Operations

- Del: $\nabla = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \end{bmatrix}$
- Gradient: $\nabla s = \begin{bmatrix} \frac{\partial s}{\partial x} & \frac{\partial s}{\partial y} & \frac{\partial s}{\partial z} \end{bmatrix}$
- Divergence: $\nabla \cdot \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}$
- Curl: $\nabla \times \mathbf{v} = \begin{bmatrix} \frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z} & \frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x} & \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \end{bmatrix}$
- Laplacian: $\nabla^2 s = \frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} + \frac{\partial^2 s}{\partial z^2}$

Frame of Reference

- When describing fluid motion, it is important to be consistent with the frame of reference
- In fluid dynamics, there are two main ways of addressing this
- With the *Eulerian* frame of reference, we describe the motion of the fluid from some fixed point in space
- With the *Lagrangian* frame of reference, we describe the motion of the fluid from the point of view moving with the fluid itself
- Eulerian simulations typically use a fixed grid or similar structure and store velocities at every point in the grid
- Lagrangian simulations typically use particles that move with the fluid itself. Velocities are stored on the particles that are irregularly spaced throughout the domain
- We will stick with an Eulerian point of view today, but we will look at Lagrangian methods in the next lecture when we discuss particle based fluid simulation

Advection

- Let us assume that we have a velocity field $\mathbf{v}(\mathbf{x})$ and we have some scalar field $s(\mathbf{x})$ that represents some scalar quantity that is being transported through the velocity field
- For example, $\mathbf{v}$ might be the velocity of air in the room and s might represent temperature, or the concentration of some pigment or smoke, etc.
- As the fluid moves around, it will transport the scalar field with it. We say that the scalar field is *advected* by the fluid
- The rate of change of the scalar field at some location is:

$$\frac{ds}{dt} = -\mathbf{v} \cdot \nabla s$$

Convection

- The velocity field **v** describes the movement of the fluid down to the molecular level
- Therefore, all properties of the fluid at a particular point should be advected by the velocity field
- This includes the property of velocity itself!
- The advection of velocity through the velocity field is called *convection*

$$\frac{d\mathbf{v}}{dt} = -\mathbf{v} \cdot \nabla \mathbf{v}$$

- Remember that $d\mathbf{v}/dt$ is an acceleration, and since $\mathbf{f}=m\mathbf{a}$, we are really describing a force

Diffusion

- Lets say that we put a drop of red food coloring in a motionless water tank. Due to random molecular motion, the red color will gradually diffuse throughout the tank until it reaches equilibrium
- This is known as a *diffusion* process and is described by the diffusion equation

$$\frac{ds}{dt} = k\nabla^2 s$$

- The constant *k* describes the rate of diffusion
- Heat diffuses through solids and fluids through a similar process and is described by a diffusion equation

Viscosity

- *Viscosity* is essentially the diffusion of velocity in a fluid and is described by a diffusion equation as well:

$$\frac{d\mathbf{v}}{dt} = \mu \nabla^2 \mathbf{v}$$

- The constant μ is the coefficient of viscosity and describes how viscous the fluid is. Air and water have low values, whereas something like syrup would have a relatively higher value
- Some materials like modeling clay or silly putty are extremely viscous fluids that can behave similar to solids
- Like convection, viscosity is a force. It resists relative motion and tries to smooth out the velocity field

Pressure Gradient

- Fluids flow from high pressure regions to low pressure regions in the direction of the *pressure gradient*

$$\frac{d\mathbf{v}}{dt} = -\nabla p$$

- The difference in pressure acts as a force in the direction from high to low (thus the minus sign)

Transport Equations

- Advection: $\frac{ds}{dt} = -\mathbf{v} \cdot \nabla s$
- Convection: $\frac{d\mathbf{v}}{dt} = -\mathbf{v} \cdot \nabla \mathbf{v}$
- Diffusion: $\frac{ds}{dt} = k\nabla^2 s$
- Viscosity: $\frac{d\mathbf{v}}{dt} = \mu\nabla^2 \mathbf{v}$
- Pressure: $\frac{d\mathbf{v}}{dt} = -\nabla p$

Navier-Stokes Equation

- The complete *Navier-Stokes equation* describes the strict conservation of mass, energy, and momentum within a fluid
- Energy can be converted between potential, kinetic, and thermal states
- The full equation accounts for fluid flow, convection, viscosity, sound waves, shock waves, thermal buoyancy, and more
- However, simpler forms of the equation can be derived for specific purposes. Fluid simulation, for example, typically uses a limited form known as the incompressible flow equation

Incompressibility

- Real fluids have some degree of compressibility. Gasses are very compressible and even liquids can be compressed some
- Sound waves in a fluid are caused by compression, as are supersonic shocks, but generally, we are not interested in modeling these fluid behaviors
- We will therefore assume that the fluid is incompressible and we will enforce this as a constraint
- Incompressibility requires that there is zero divergence of the velocity field everywhere

$$\nabla \cdot \mathbf{v} = 0$$

- This is actually very reasonable, as compression has a negligible affect on fluids moving well below the speed of sound

Navier-Stokes Equation

- The incompressible Navier-Stokes equation describes the forces on a fluid as the sum of convection, viscosity, and pressure terms:

$$\frac{d\mathbf{v}}{dt} = -\mathbf{v} \cdot \nabla \mathbf{v} + \mu \nabla^2 \mathbf{v} - \nabla p$$

- In addition, we also have the incompressibility constraint:

$$\nabla \cdot \mathbf{v} = 0$$

Numerical Representation of Fields

- A scalar or vector field represents a continuously variable value across space that can have infinite detail
- Obviously, on the computer, we can’t truly represent the value of the field everywhere to this level, so we must use some form of approximation
- A standard approach to representing a continuous field is to sample it at some number of discrete points and use some form of interpolation to get the value between the points
- There are several choices of how to arrange our samples:
  - Uniform grid
  - Hierarchical grid
  - Irregular mesh
  - Particle based

Finite Difference First Derivatives

- If we have a scalar field $s(\mathbf{x},t)$ stored on a uniform grid, we can approximate the partial derivative along x at grid cell i as:

$$\frac{\partial s_i}{\partial x} \approx \frac{s_{i+1} - s_{i-1}}{x_{i+1} - x_{i-1}} = \frac{s_{i+1} - s_{i-1}}{2\Delta x}$$

- Where cell i+1 is the cell in the +x direction and cell i-1 is in the –x direction
- Also Δx is the cell size in the x direction
- All of the partial derivatives in the gradient, divergence, and curl can be computed in this way

Finite Difference Second Derivative

- The second derivative can be computed in a similar way:

$$\frac{\partial^2 s_i}{\partial x^2} \approx \frac{s_{i+1} - 2s_i + s_{i-1}}{\Delta x^2}$$

- This can be used in the computation of the Laplacian
- Remember, these are based on the assumption of a uniform grid. To calculate the derivatives on irregular meshes or with particle methods, the formulas get more complex

Smooth Particle Hydrodynamics

- Particle based fluid simulation is often referred to as *smooth particle hydrodynamics* or SPH
- Some of the original work was done for simulating galactic gas dynamics by astrophysicists
- The technique was introduced to the computer graphics community around 2003
- In recent years, advances in the techniques as well as increases in GPU computational power have made large-scale SPH simulations possible
- The technique has proven very effective, especially for simulating very dynamic situations with lots of splashing and interaction with complex surfaces

Spatial Hash Table

- A spatial hash table operates very much like a standard hash table, where a hashing function maps some key (like a string) to an integer, which is then mod'ed into an array of slots. Items can be added, removed, or accessed through the table in constant time
- The spatial hash table is essentially the same thing, but it uses a 3D position to map to a grid cell which is then hashed into the table
- The table stores occupied cells, each of which may contain several particles, but will be limited to some maximum number due to the physics
- If more than one occupied cell maps to the same table entry, then the table entry can simply contain a linked list of cells. In practice, if the table size is anywhere near the number of particles, then this will happen very rarely
- The paper refers to a 'compact hashing' scheme that uses some additional tricks to keep the memory size manageable

Hash Function

- A point in infinite space is mapped into a finite list of cells using a hash function such as:

$$c = \left[\left(\left\lfloor\frac{x}{d}\right\rfloor \cdot p_1\right) \text{xor} \left(\left\lfloor\frac{y}{d}\right\rfloor \cdot p_2\right) \text{xor} \left(\left\lfloor\frac{z}{d}\right\rfloor \cdot p_3\right)\right] \%m$$

- With *d* being the cell spacing, *m* the hash table size, and p_1, p_2, and p_3 being large prime numbers such as 73856093, 19349663, and 83492791

Marching Cubes

- Most surface extraction techniques are based on the *marching cubes* algorithm or some variation of it
- With this approach, we first create a virtual grid around our particles where the grid size is a little smaller than the particle radius
- We then evaluate a ‘distance’ or ‘density’ type function on this grid, where each point computes some density value based on the particle nearby
- The surface is implicitly defined as being the set of points where the density value is some constant (i.e., an *isosurface*). An isosurface is a 3D version of the 2D isocurves one finds on a topographic map
- To find the surface, we loop through each cell and examine the 8 values on the corners of the cell. If some of the values of the cell are above the isosurface value and some are below, then we know that the surface passes through the cell. We then triangulate that small section of the surface and repeat for all cells

Marching Cubes

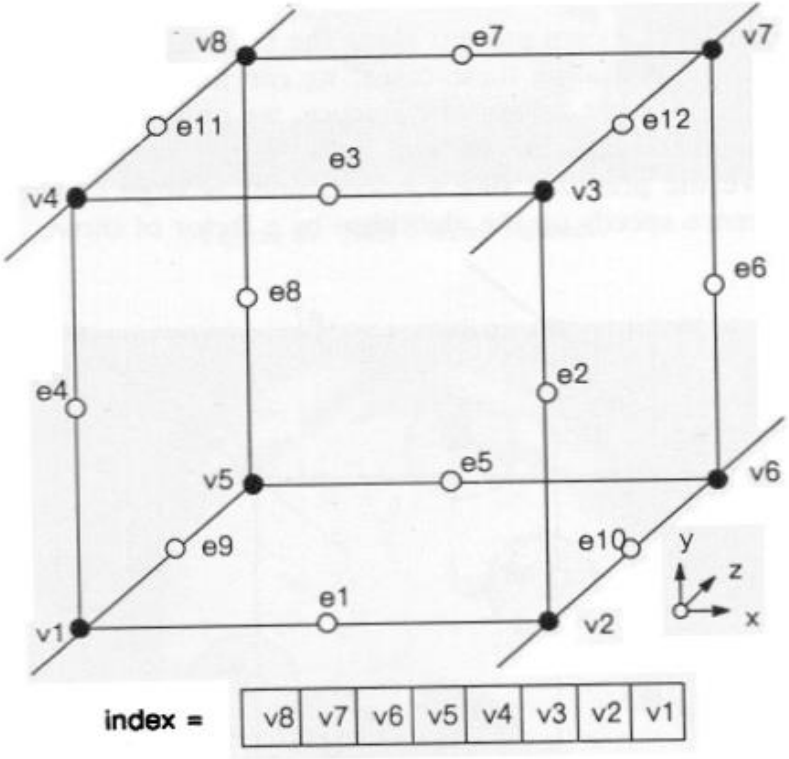


Figure 4. Cube Numbering.

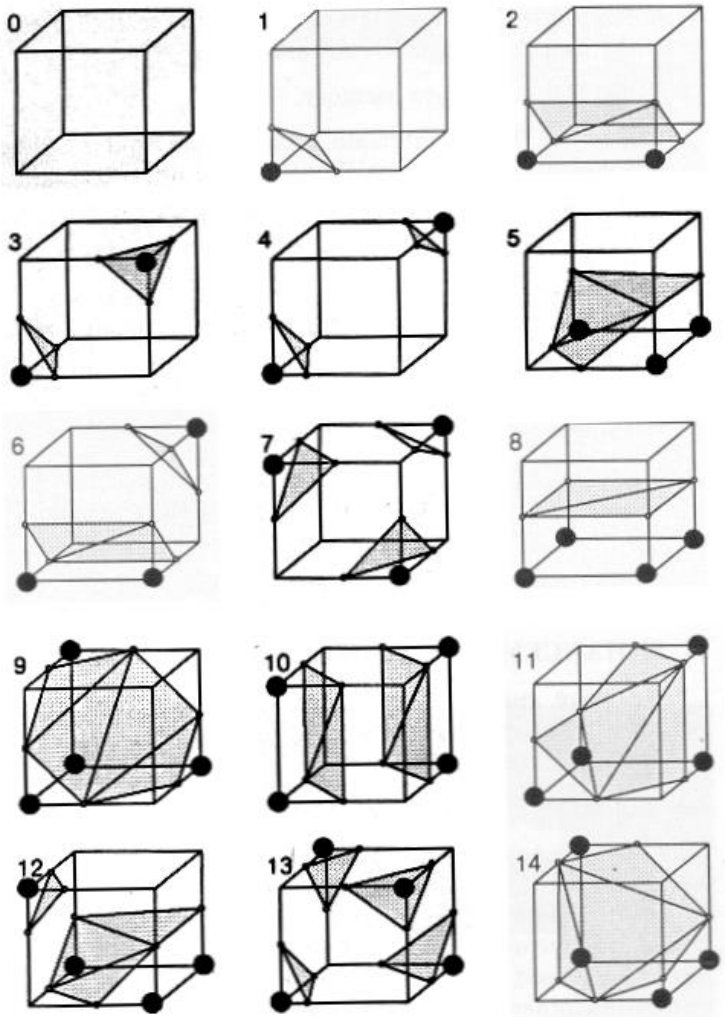


Figure 3. Triangulated Cubes.