Physically-Based Animation

Chapter 4

Introduction

Animators are often more concerned with the general quality of the motion than with precisely controlling the position and orientation of each object in each frame. Such is the case with **physically based animation**. In physically based animation, forces are typically used to maintain relationships among geometric elements. Note that these forces may or may not be physically accurate – animation is not necessarily concerned with being accurate but rather than with believability, sometimes referred to as physical realism. Some forces may not relate to physics at all – they may model constraints that an animator may wish to enforce on the motion, such as directing a ball to go toward making contact with a bat. For purposes of this chapter, the use of forces to control motion will be referred to as physically based animation, whether or not the forces are actually trying to model some aspects of physics

Chapter 4

Introduction

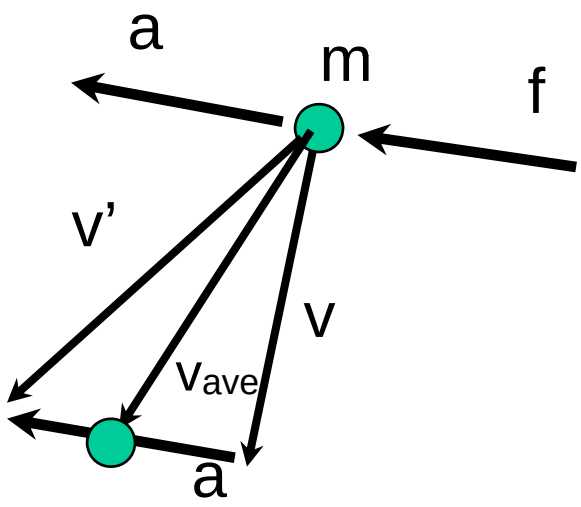
When modeling motion that is produced by an underlying mechanism, such as the principles of physics, a decision has to be made concerning the level at which to model the process. For example, wrinkles in a cloth can be modeled by trying to characterize common types of wrinkles and where they typically occur on a piece of cloth. On the other hand, physics (stresses and strains) of the individual threads of cloth could be modeled in sufficient detail to give rise to naturally occurring wrinkles. Modeling the surface characteristics of the process (the former case) is usually computationally less expensive.

Chapter 4

Introductory video.

Chapter 4

Physics Review: force, mass, acceleration velocity, position



$$f = ma$$

$$a = f / m$$

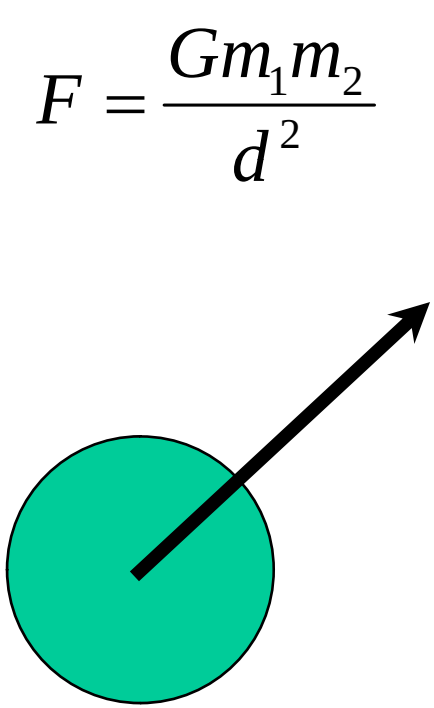
$$v' = v + a \cdot \Delta t$$

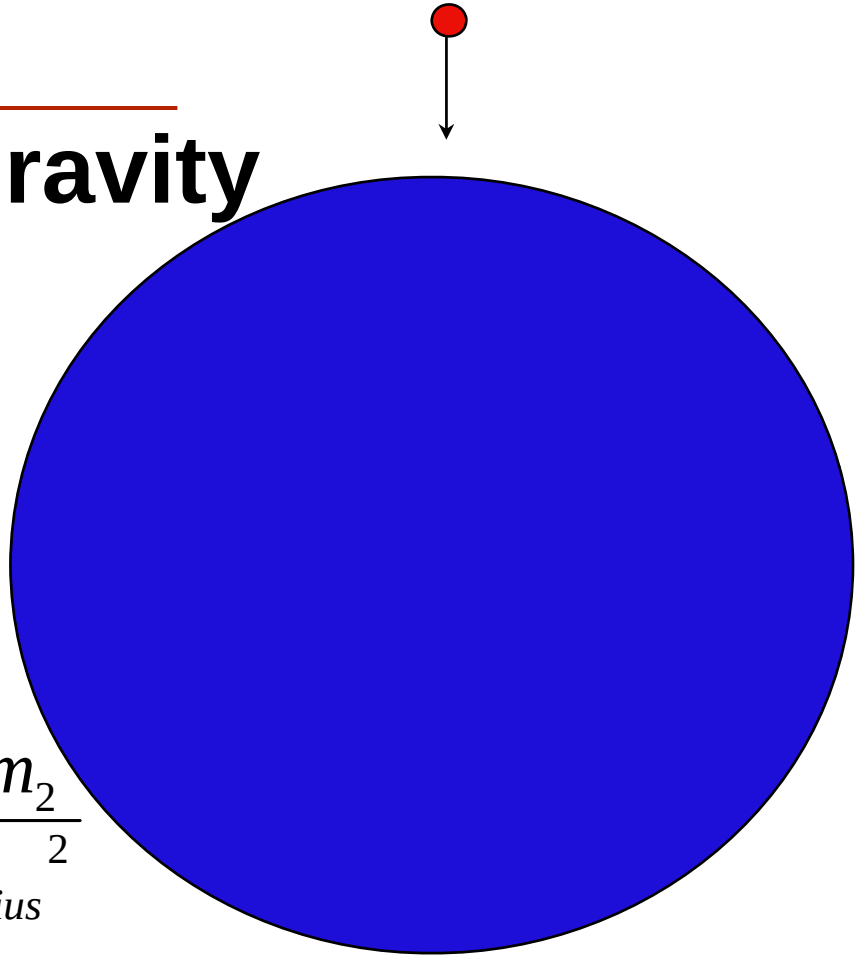
$$p' = p + \frac{(v + v')}{2}\Delta t = p + v\Delta t + \frac{1}{2}a\Delta t^2$$

Chapter 4

Physics Review: Gravity

$$F = \frac{Gm_1m_2}{d^2}$$





$$F = \frac{Gm_{earth}m_2}{d_{earth-radius}^{\ 2}}$$

$$a = \frac{F}{m_2} = 9.8m/s^2$$

Chapter 4

Physics Review: Other forces

$$f_s = sf_N$$ Static friction

$$f_k = kf_N$$ Kinetic friction

$$f_{vis} = -Knv$$ Viscosity

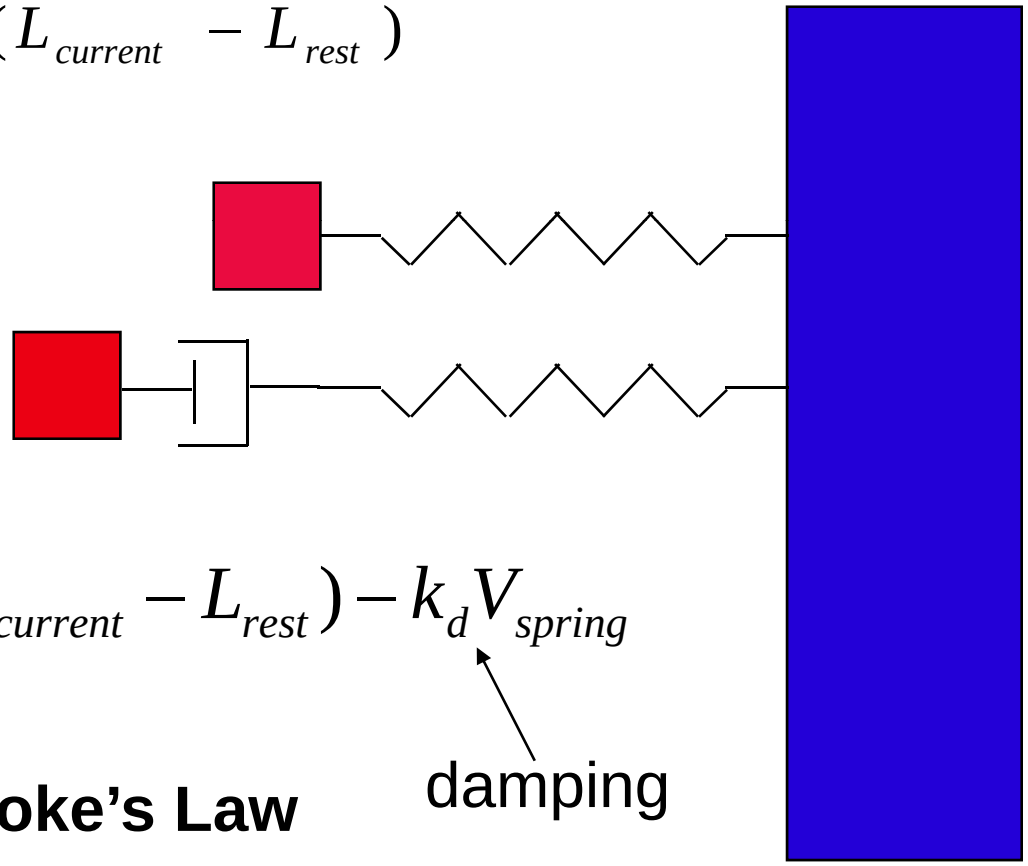
$$K = 6\pi r$$ For sphere

Chapter 4

Physics Review: Spring-damper

$$F = k_s(L_{current} - L_{rest})$$

stiffness



$$F = k_s(L_{current} - L_{rest}) - k_d V_{spring}$$

damping

Hooke's Law

Chapter 4

Physics Review: Momentum

conservation of momentum (mv)

In a closed system, momentum is conserved

$$P = \Sigma mv = \Sigma m'v'$$

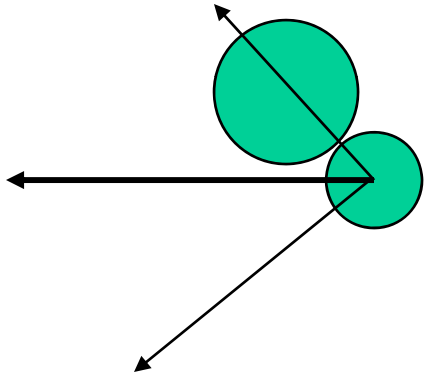
After collision has same momentum as before collision (minus loss of energy due to friction)

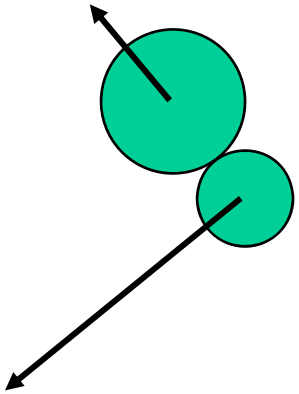
Can be used to solve for velocities after collision

Chapter 4

Physics Review: Momentum

$$\Sigma mv = \Sigma m'v'$$





$$F = ma = \frac{d(mv)}{dt} = \frac{dP}{dt}$$

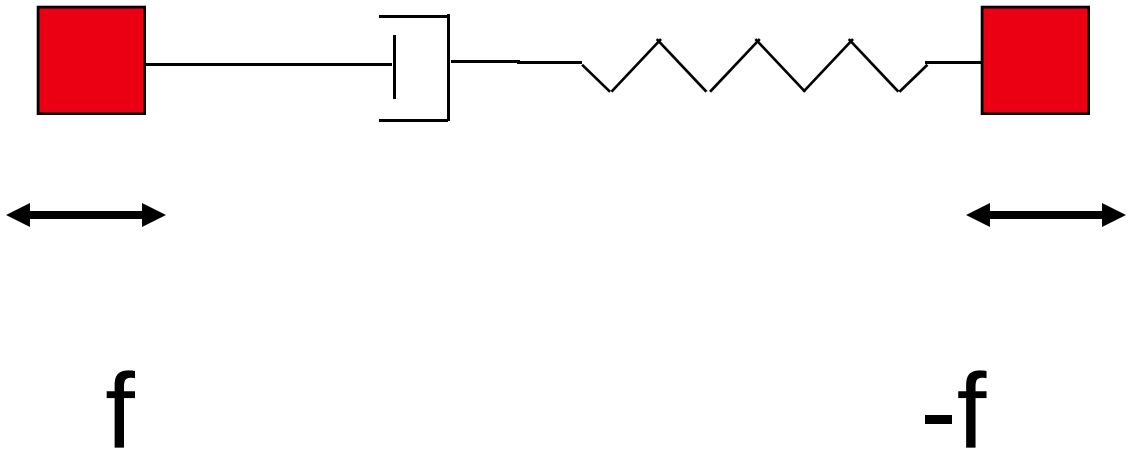
Chapter 4

Spring Meshes

Probably the most common approach to modeling flexible shapes is the spring-mass-damper model. The most straightforward strategy is to model each vertex of an object as a point in mass and each edge of the object as a spring. Each spring's rest length is set equal to the original length of the edge. A mass can be arbitrarily assigned to an object by the animator and the mass evenly distributed among the object's vertices. If there is an uneven distribution of vertices throughout the object definition, then masses can be assigned to vertices in an attempt to more evenly distribute mass.

Chapter 4

Spring-mass-damper system



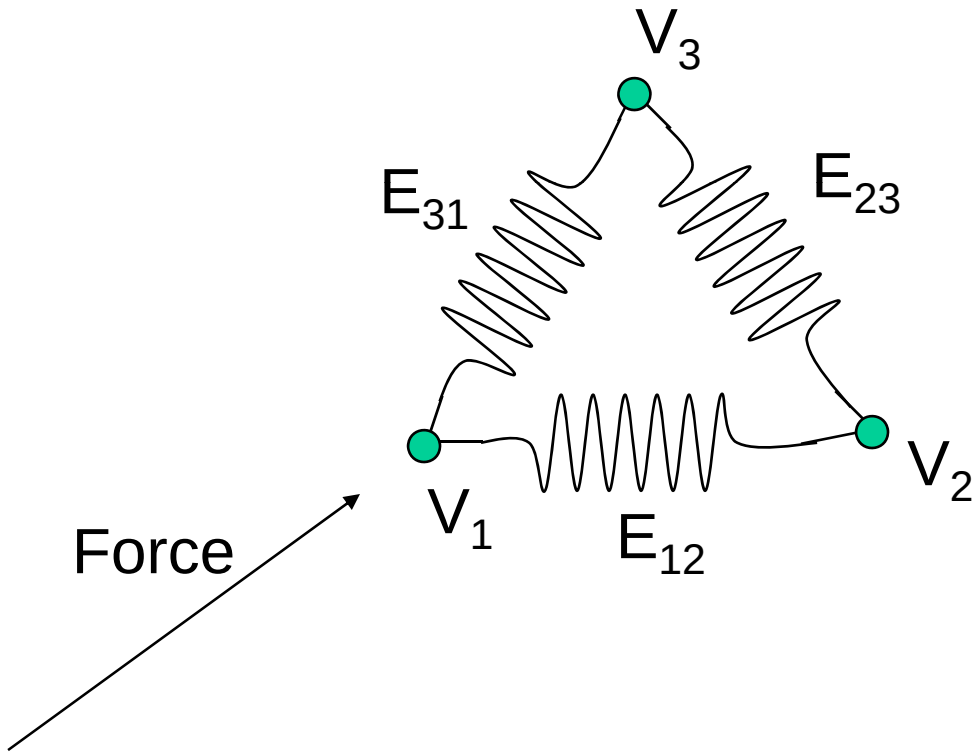
Chapter 4

Spring Meshes

As forces are applied to specific vertices of the object, either because of collision, gravity, wind, or explicitly scripted forces, vertices will be displaced relative to other vertices of the object. This displacement will induce spring forces. Which will impart forces to adjacent vertices as well as reactive forces back to the initial vertex. These forces will result in further displacements, which will induce more spring forces throughout the object, resulting in more displacements, and so on. The result will be an object that is wiggling and jiggling as a result of the forces propagating along the edge springs.

Chapter 4

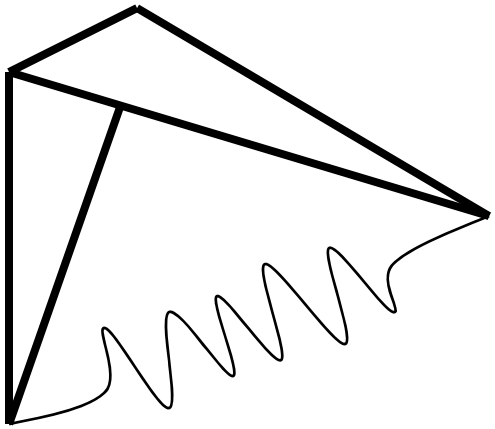
Spring-mass system



Chapter 4

Angular springs

If specific angles between adjacent faces (dihedral angles) are desired, then angular springs (and dampers) can be applied.



Linear spring between vertices

Dihedral angular spring

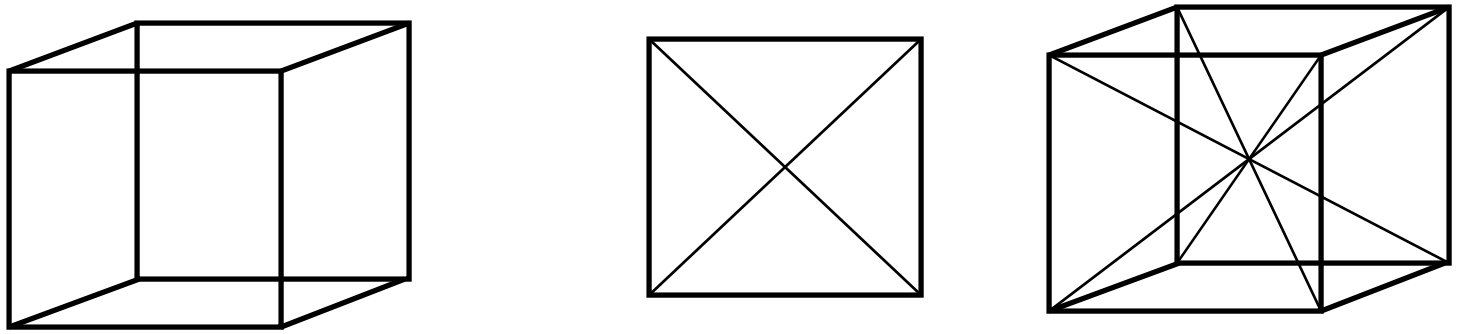
Chapter 4

Virtual Springs

Virtual springs introduce forces into the system that do not directly model physical elements. These can be used to control the motion of objects in a variety of circumstances. Virtual springs with zero rest lengths can be used to constrain one object to lie on the surface of another, for example, or with non-zero rest lengths, to maintain separation between moving objects. Virtual springs can also be used to help maintain the shape of an object:

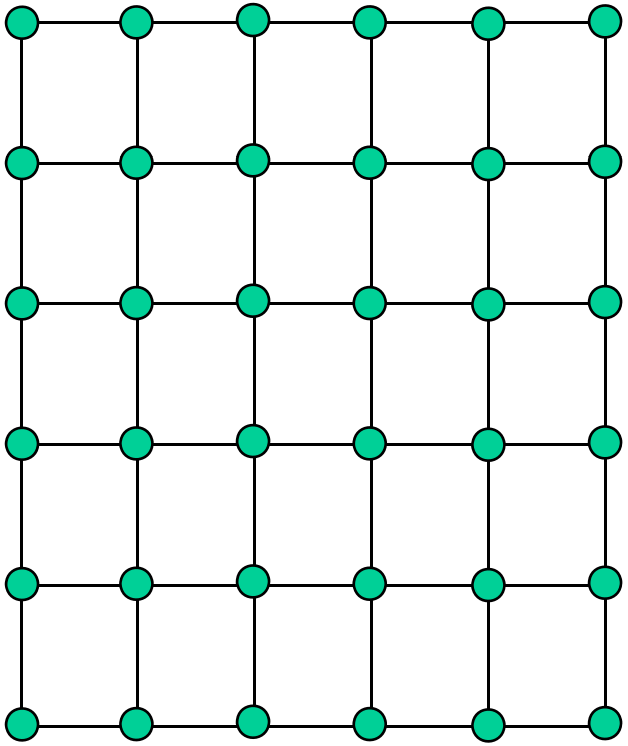
Chapter 4

“Virtual” edge springs system



Chapter 4

Spring mesh



Each vertex is a point mass

Each edge is a spring-damper

Angular springs connect every other mass point

Global forces: gravity, wind

Chapter 4

Proportional Derivative Controllers

Proportional derivative controllers (PDCs) are another type of virtual spring used to keep a control variable and its derivative within the neighborhood of desired values. For example, to maintain a joint angle and joint velocity close to desired values, the user can introduce a torque into a system:

$$\tau = k_s(\theta(t) - \theta_d) - k_d(\dot{\theta}(t) - \dot{\theta}_d)$$

Dampers and PDCs are commonly used to keep system parameters close to desired values.

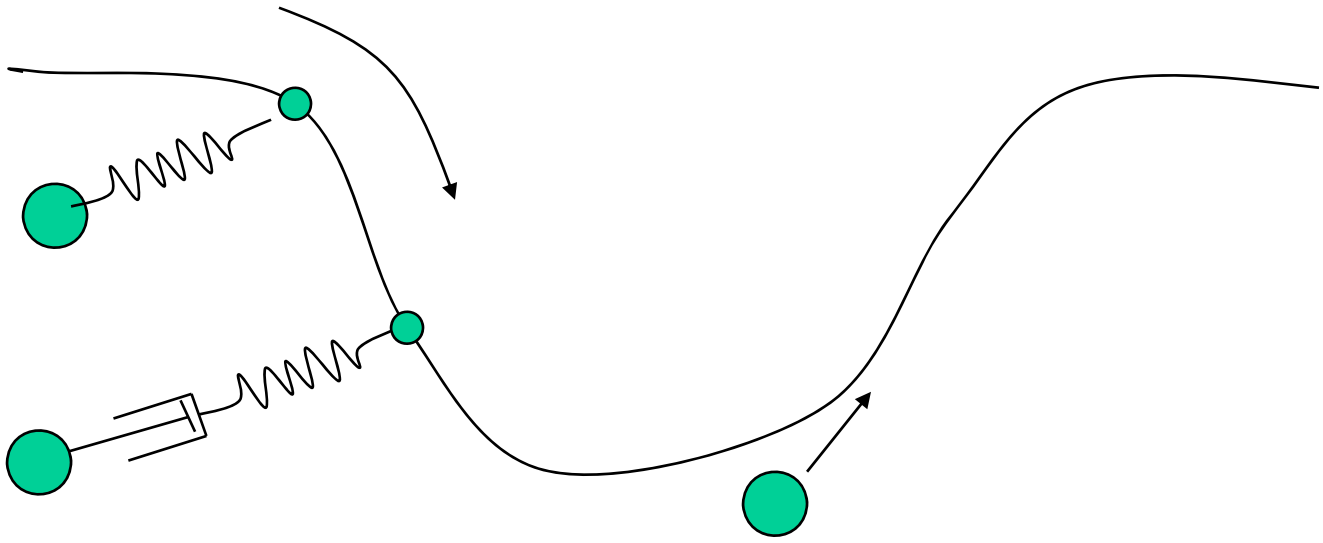
Chapter 4

Proportional (Derivative) Controllers

e.g., particle reacts to other forces while trying to maintain position on curve – virtual spring

$$d = x(t) - p(t)$$

$$F = k_s d$$



$$F = k_s d - k_d v_{relative}$$

Chapter 4

Relaxation

If a force acts on one of the vertices of the spring mesh, the system is no longer in its equilibrium which causes the vertices to move. This leads to a system of differential equations: $a = \dot{v} = \ddot{p}$

$$f(p) = m \cdot a = m \cdot \ddot{p}$$

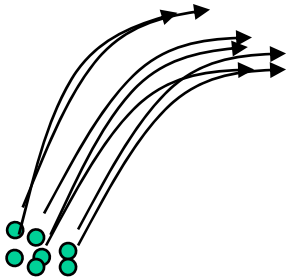
In the simplest case (Euler integration), we can solve this system by computing the forces for every vertex and use it to update the velocity and then the position. Since this will most likely induce forces to other vertices we need to repeat this step until the system reaches its equilibrium again.

Chapter 4

Particle systems

A particle system is a collection of a large number of point-like elements. Particle systems are often animated as a simple physical simulation. Because of the large number of elements, simplifying assumptions are used in both the rendering and the calculation of their motions. Common assumptions are:

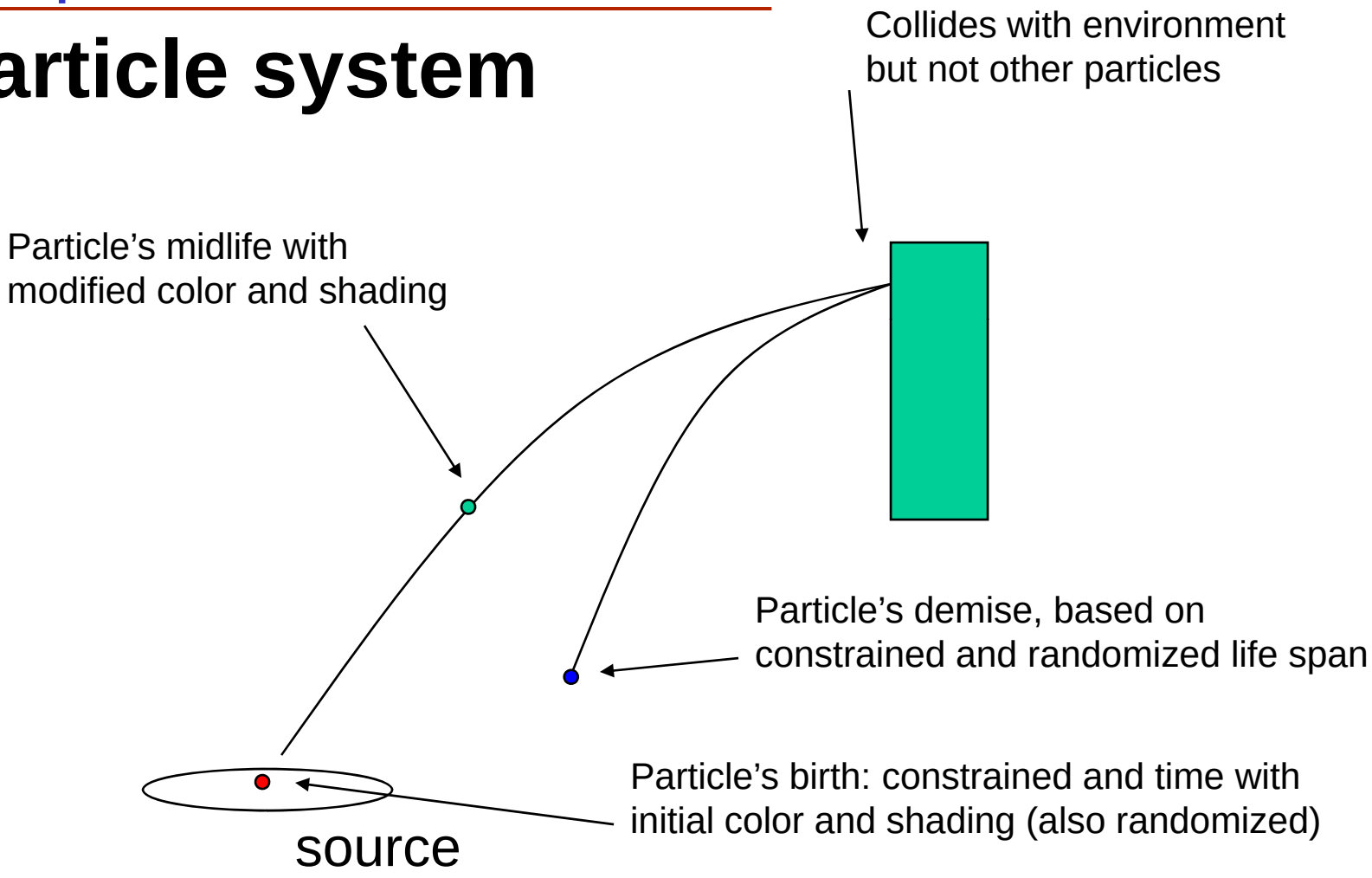
Particles:



Do collide with the environment
Do not collide with other partcles
Do not cast shadows on other particles
Might cast shadows on environment
Do not reflect light - usually emit it

Chapter 4

Particle system



Chapter 4

Particle Generation

Particles are typically generated according to a controlled stochastic process. For each frame, a random number of particles are generated using some user-specified distribution.

Particle Attributes

The attributes of a particle determine its motion status, its appearance, and its life in the particle system. Typical attributes are position, velocity, shape parameters, color, transparency, or lifetime.

Chapter 4

Particle Termination

At each new frame, each particle's lifetime attribute is decremented by one. When the attribute reaches zero, the particle is removed from the system.

Particle Animation

Typically, each active particle in a particle system is animated throughout its life. This animation includes not only its position and velocity but also its display attributes, such as color, shape, etc. To animate the particle's position in space, the user considers forces and computes the resultant particle acceleration.

Chapter 4

Particle system implementation

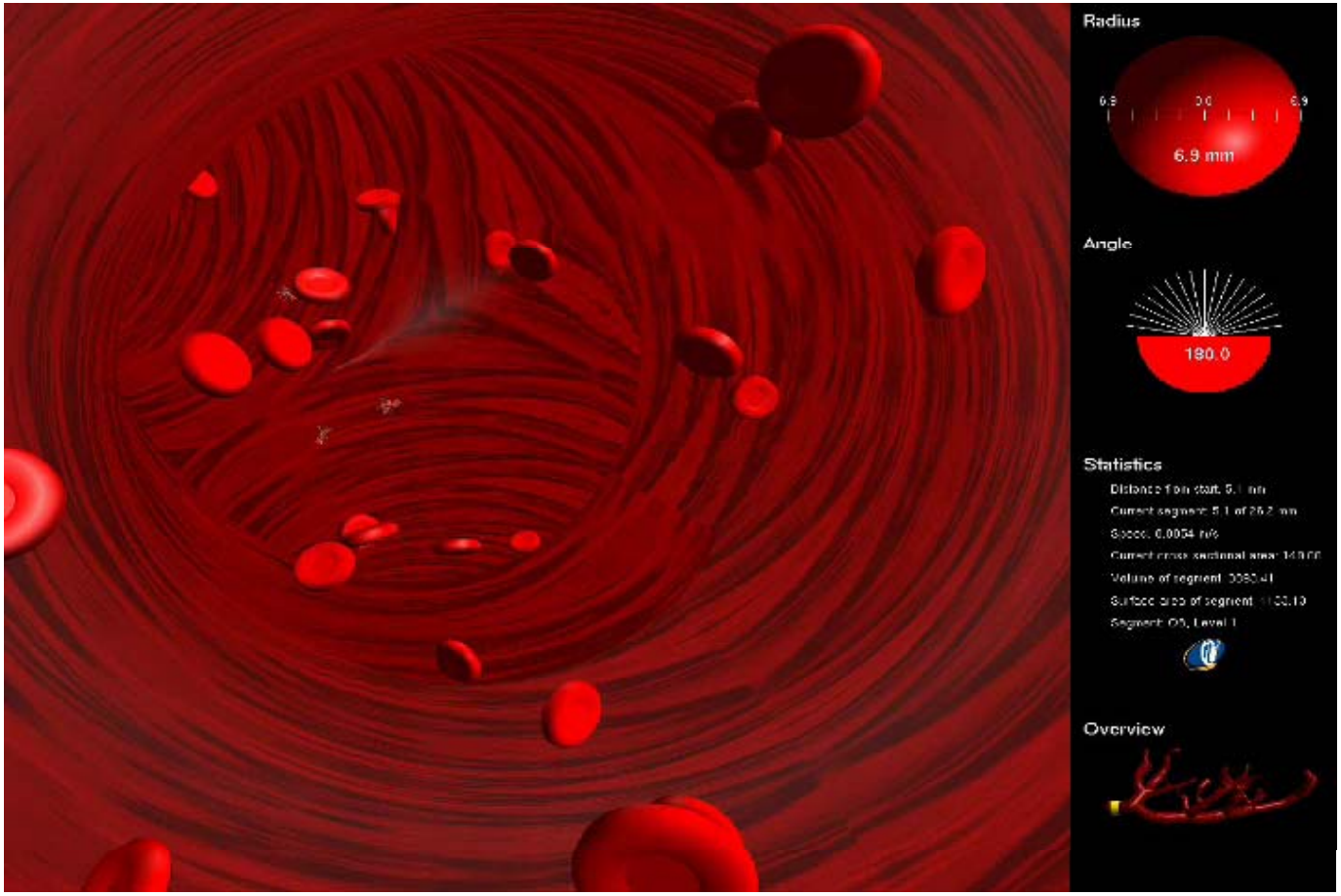
STEPS

1. for each particle
    1. if dead, reallocate and assign new attributes
    2. animate particle, modify attributes
2. render particles

Use constrained randomization to keep control of the simulation while adding interest to the visuals

Chapter 4

Particle simulation: example



Chapter 4

Rigid Body Simulation

A common objective in computer animation is to create realistic-looking motion. A major component of realistic motion is the physically based reaction of rigid bodies to commonly encountered forces such as gravity, viscosity, friction, and those resulting from collisions. Creating realistic motion with key-frame techniques can be a daunting task. However, the equations of motion can be incorporated into an animation system to automatically calculate these reactions. This can eliminate considerable tedium – if the animator is willing to relinquish precise control over the motion of some objects.

Chapter 4

Example: Free Fall

To understand the basics in modeling physically based motion, the motion of a point in space will be considered first. The position of the point at discrete time steps is desired, where the interval between these time steps in some uniform Δt. To update the position of the point over time, its position, velocity, and acceleration are used, which are modeled as functions of time.

Consider a point with an initial position of *(0,0)*, with an initial velocity of *(100,100)* feet per second, and under the force of gravity resulting in a uniform acceleration of *(0,-32)* feet per second per second.

Chapter 4

Integration

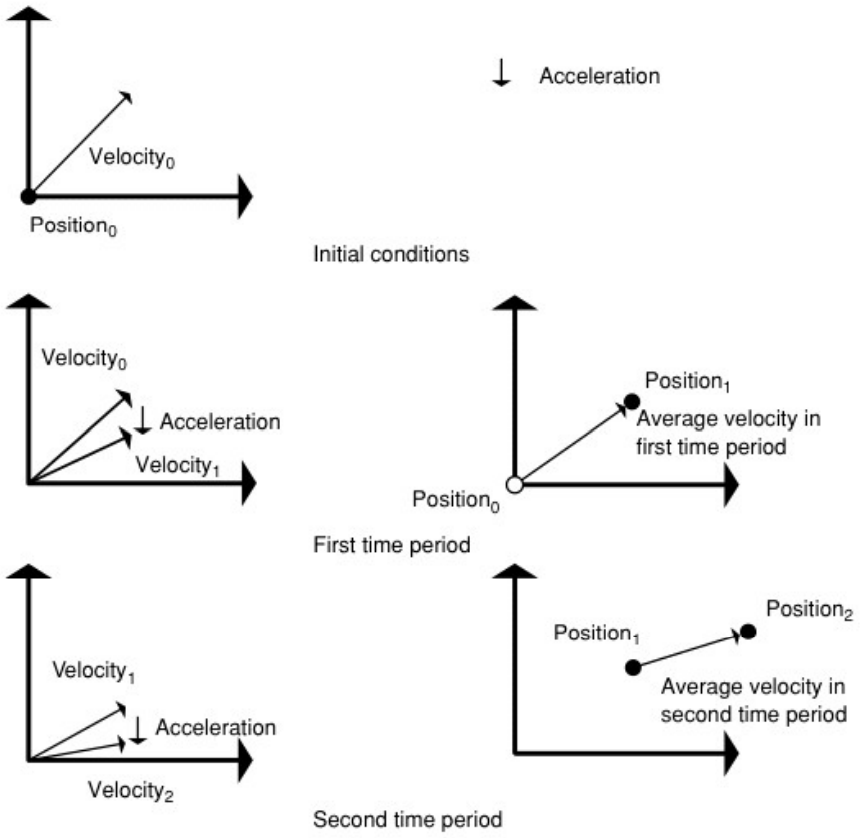
Given acceleration, compute velocity & position by integrating over time

$$f = ma$$

$$a = f / m$$

$$v' = v + a \cdot t$$

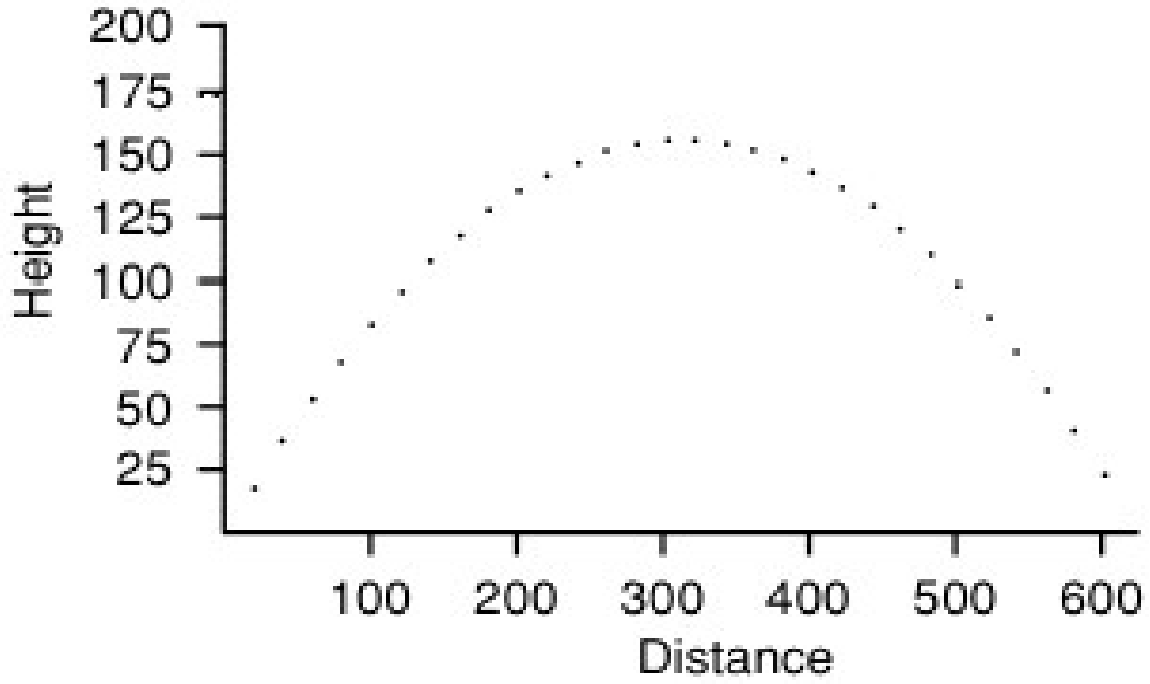
$$p' = p + vt + \frac{1}{2}at^2$$



Chapter 4

Projectile

given initial velocity under gravity

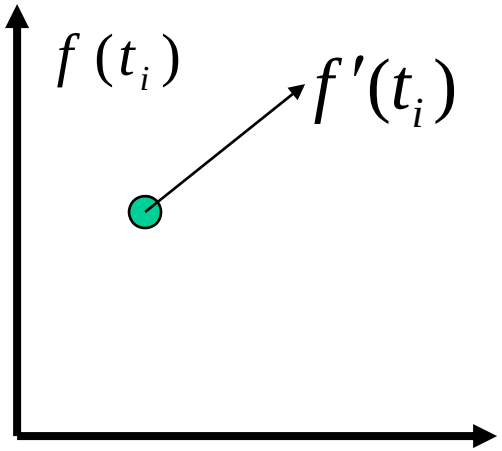


Chapter 4

Euler integration

For arbitrary function, f(t)

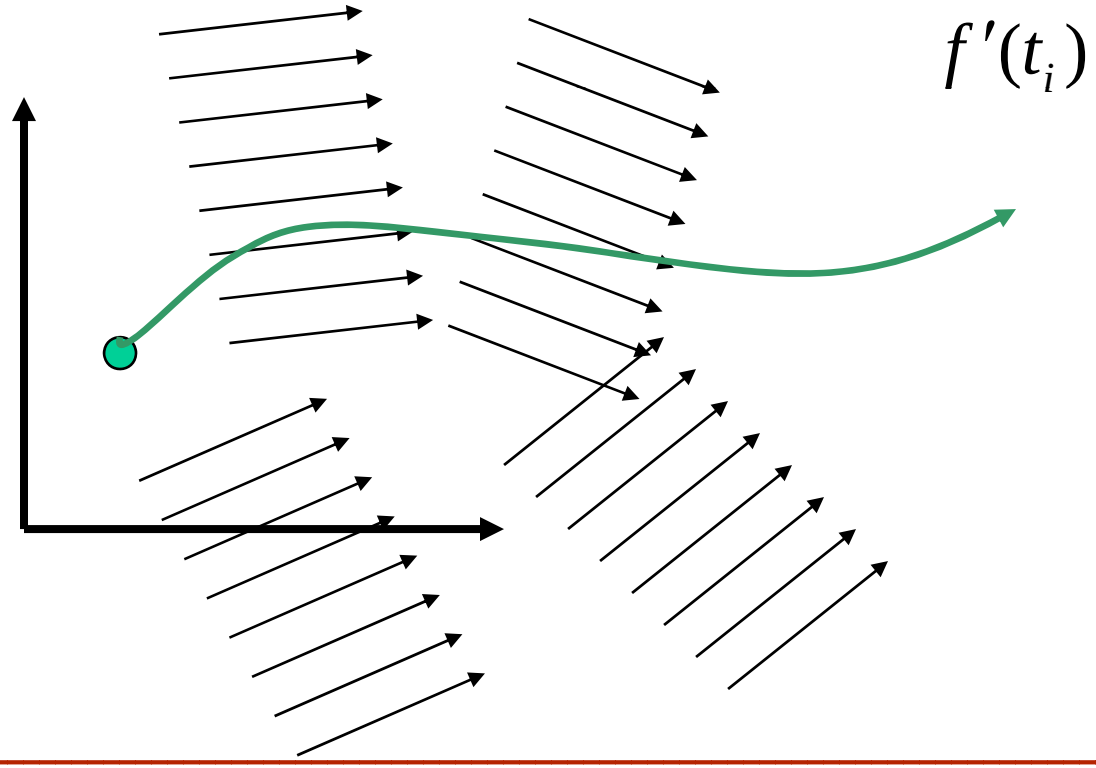
$$f(t_{i+1}) = f(t_i) + f'(t_i) \cdot \Delta t$$



Chapter 4

Integration – derivative field

For arbitrary function, f(t)



Chapter 4

Integration

As seen from the previous example, this configuration relates to solving a differential equation, i.e. integration. The assumption that acceleration remains constant over the delta time step is incorrect in most rigid body simulations; many forces continually vary as the object changes its position and velocity over time. As a result, Euler integration is not necessarily accurate for this type of problem.

Chapter 4

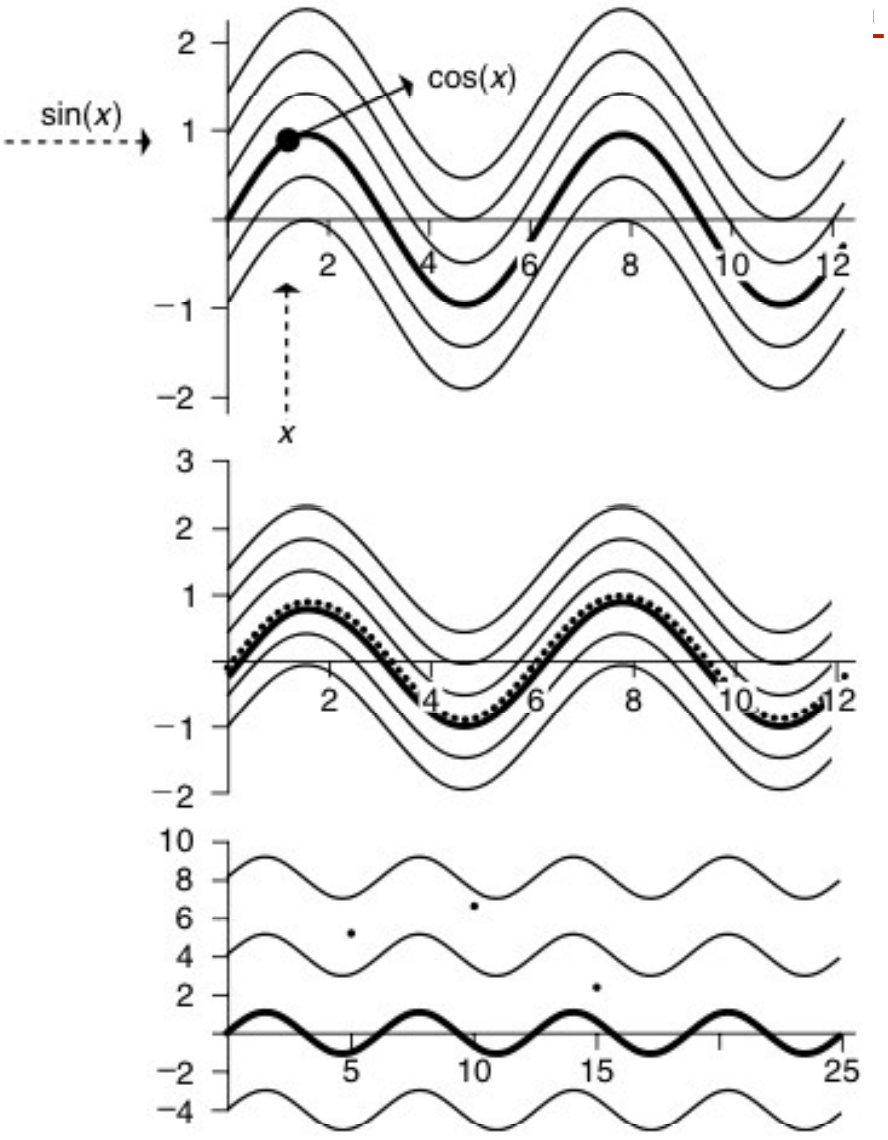
Example Integration

Apply integration method to the first derivative of the cosine function. Hence, this should yield the sine function. With increasing step size, the error of the Euler integration becomes larger.

Step size

$$\Delta x = 0.2$$

$$\Delta x = 5$$

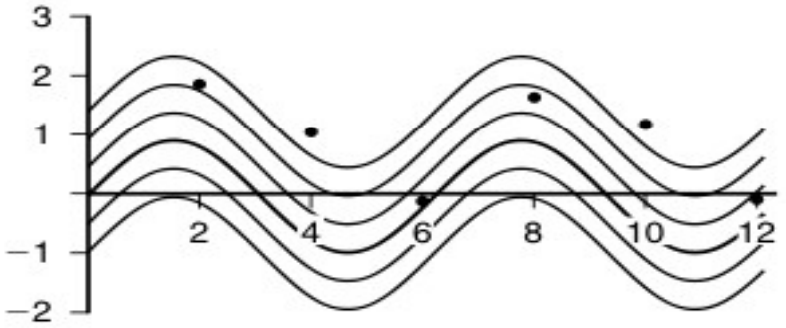


Chapter 4

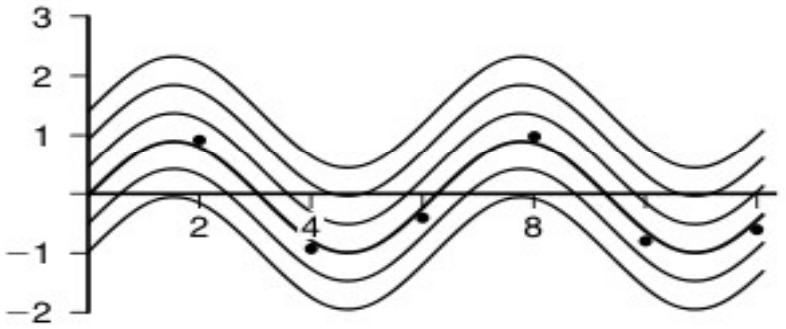
Step size

$$\Delta x = 2$$

Euler Integration



Midpoint Method



Chapter 4

Numeric Integration Methods

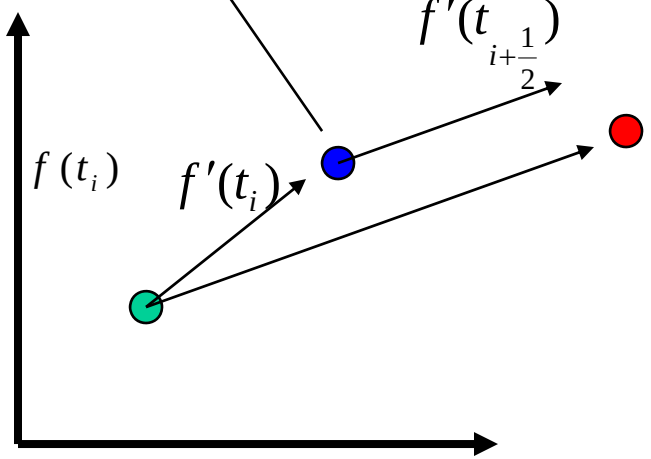
(explicit or forward) Euler Integration
2nd order Runga Kutta Integration (Midpoint Method)
4th order Runga Kutta Integration
Implicit (backward) Euler Integration
Semi-implicit Euler Integration

Chapter 4

Runge Kutta Integration: 2nd order Aka Midpoint Method

For unknown function, f(t); known f ’(t)

$$f(t_{i+\frac{1}{2}}) = f(t_i) + \frac{1}{2} f'(t_i) \cdot \Delta t$$

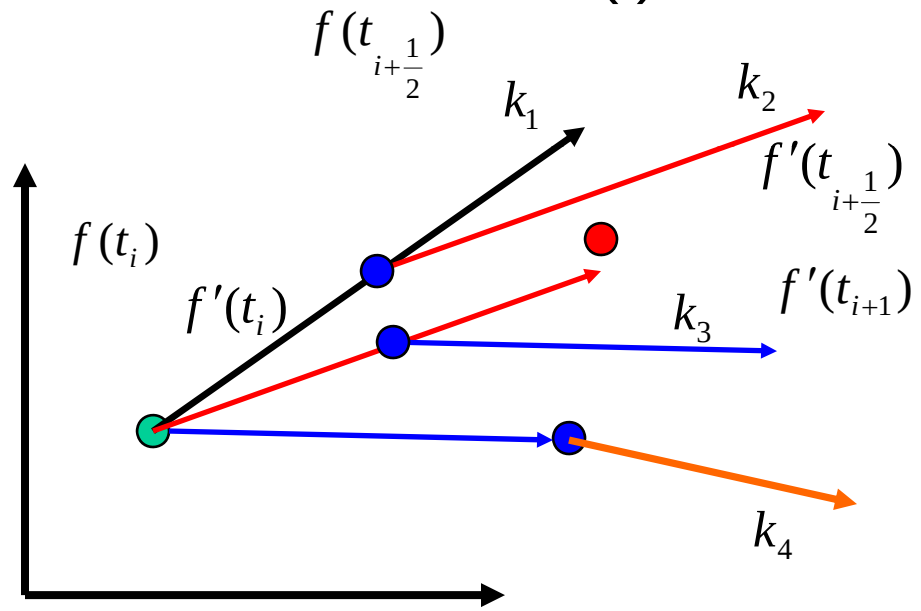


$$f(t_{i+1}) = f(t_i) + f'(t_{i+\frac{1}{2}}) \cdot \Delta t$$

Chapter 4

Runge Kutta Integration: 4th order

For unknown function, f(t); known f ’(t)



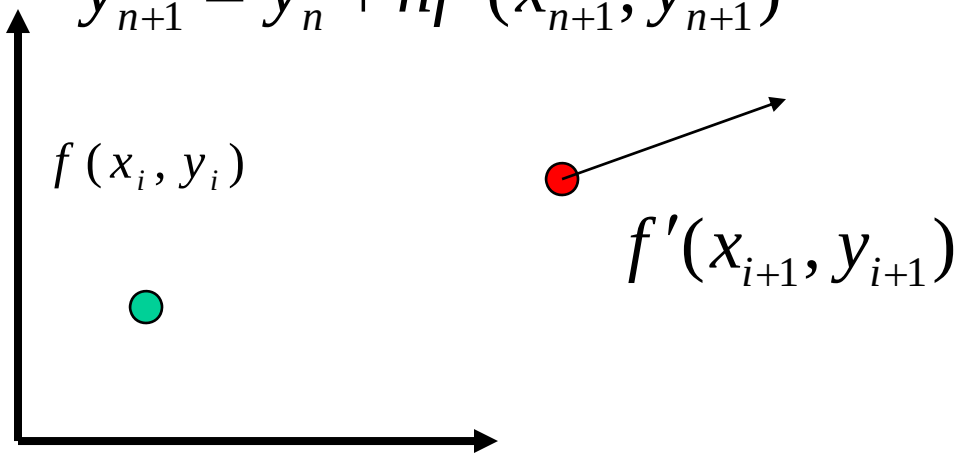
$$f(t_{i+1}) = f(t_i) + h\left(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4\right)$$

Chapter 4

Implicit Euler Integration

For arbitrary function, f(x,y), find next point whose derivative updates last value to this value: required numeric method (e.g. Newton-Raphson)

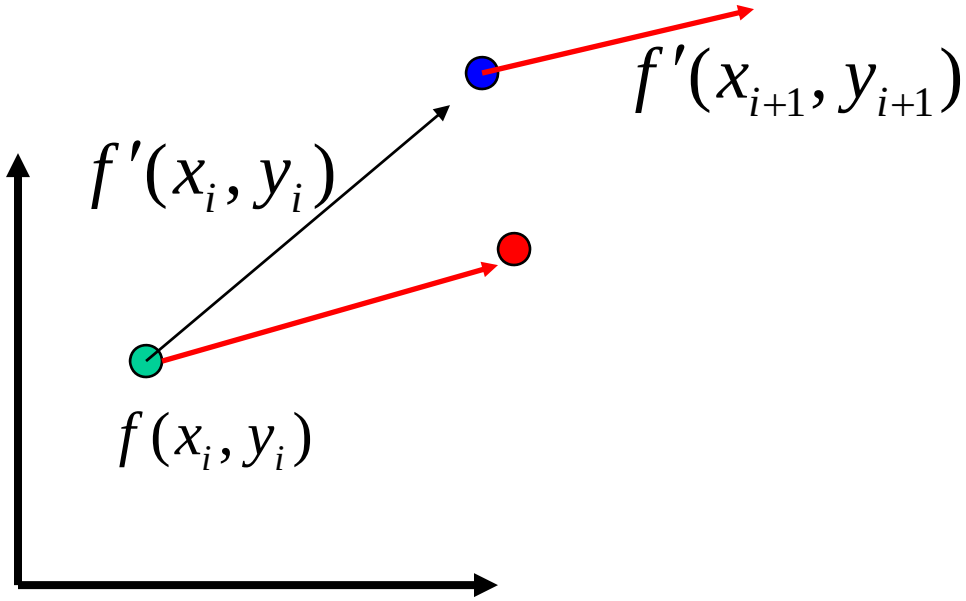
$$y_{n+1} = y_n + hf'(x_{n+1}, y_{n+1})$$



Chapter 4

Semi-Implicit Euler Integration

$$y_{i+1} = y_i + hf'(x_{i+1}, y_i + hf'(x_i, y_i))$$



Chapter 4

Methods specific to update position

Heun Method
Verlet Method
Leapfrog Method

Chapter 4

Heun Method

$$v(t_{i+1}) = v(t_i) + a(t_i)\Delta t$$

$$x(t_{i+1}) = x(t_i) + \frac{1}{2}(v(t_i) + v(t_{i+1}))\Delta t$$

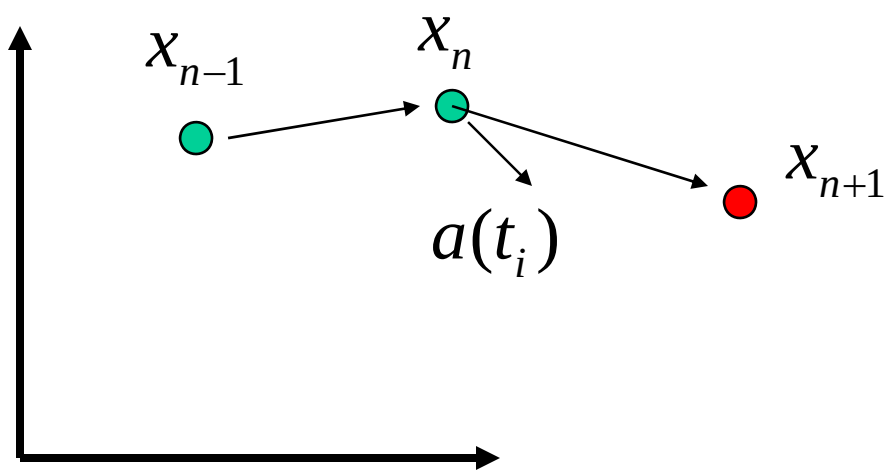
$$= x(t_i) + v(t_i)\Delta t + \frac{1}{2}a(t_i)\Delta t^2$$ (by plugging in the the first equation)

Chapter 4

Verlet Method

$$x_{n+1} = 2x_n - x_{n-1} + a(t_i)\Delta t^2$$

$$x_{n+1} = x_n + (x_n - x_{n-1}) + a(t_i)\Delta t^2$$



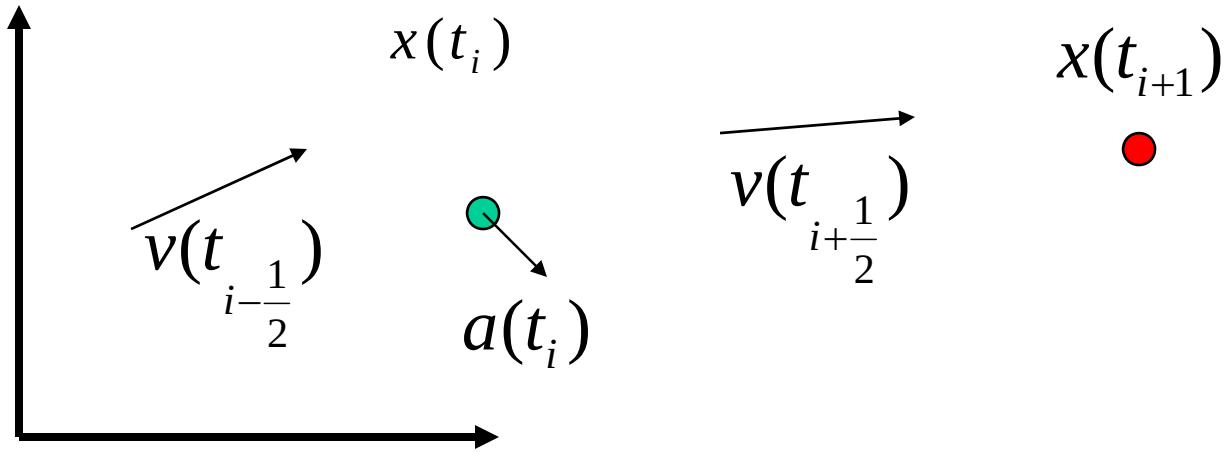
Chapter 4

Leapfrog Method

$$v(t_{i+\frac{1}{2}}) = v(t_{i-\frac{1}{2}}) + a(t_{i-1})\Delta t$$

$$x(t_{i+1}) = x(t_i) + v(t_{i+\frac{1}{2}})\Delta t$$

$$v(t_{i+\frac{3}{2}}) = v(t_{i+\frac{1}{2}}) + a(t_i)\Delta t$$



Chapter 4

Further Readings

Many books have been written about the subject of integration; *Numerical Recipes: The art of Scientific Computing*, by Press et al., is a good place to start. The previous explanations are intended merely to demonstrate that better options than Euler integration exist and that, at the very least, a Runge-Kutta method should be considered. There are also higher ordered Runge-Kutta methods (5 or 6) available, some even with an adaptive error estimate. Using this estimate, a suitable step-size is determined that ensures a certain error bound.

Chapter 4

Bodies in Collision

When an object starts to move in any kind of environment other than complete void, chances are that sooner or later it will bump into something. If nothing is done about this in an computer animation, the object will penetrate and then pass through other objects. Other types of contact include objects sliding against and resting on each other. All of these types of contact require the calculation of forces in order to accurately simulate the reaction of one object to another.

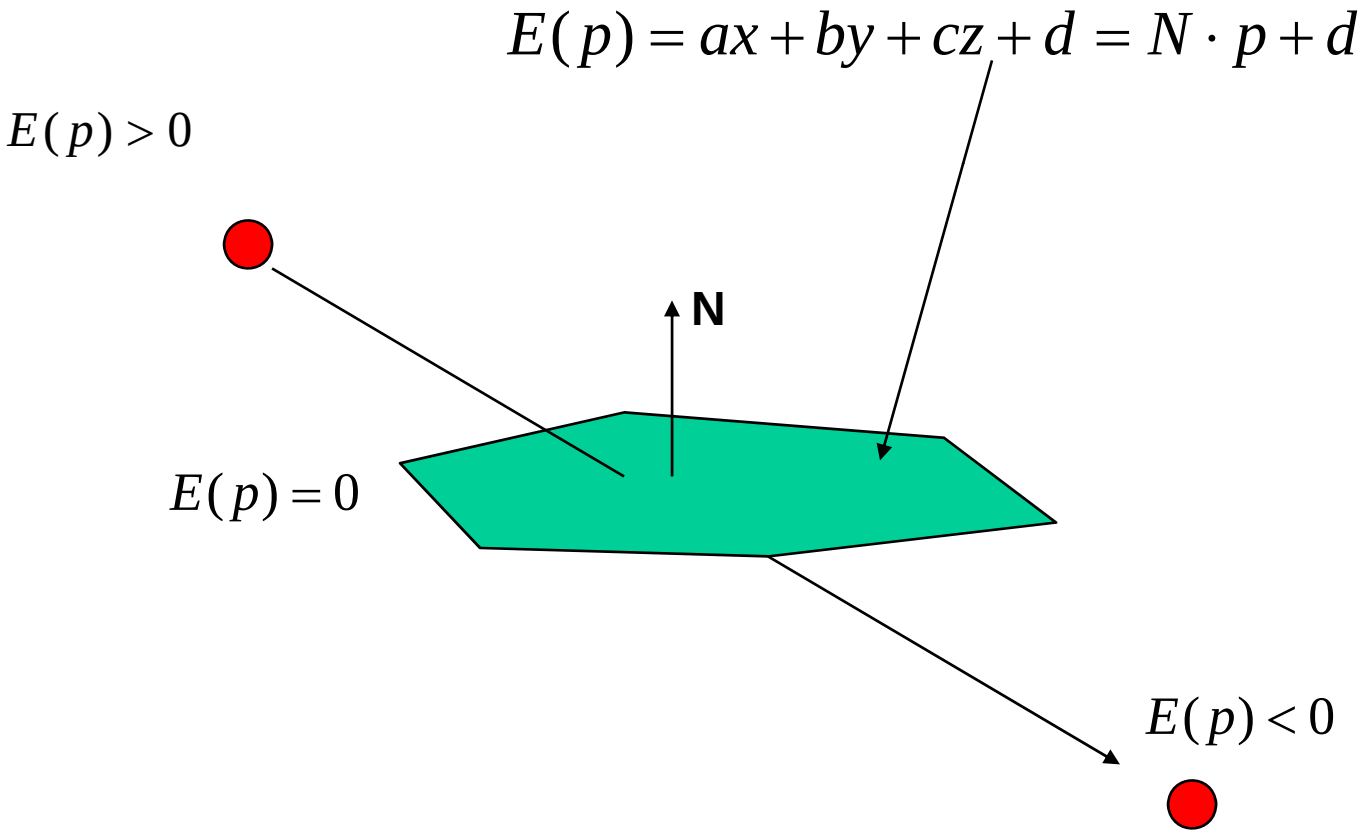
Chapter 4

Collision handling detection & response

- detection
  - Particle-plane collision detection
  - Polyhedron-polyhedron collision detection
    - overlap of Bounding volumes
    - Vertex inside polyhedron test
      - Concave case
      - Convex case
    - Edge-face intersection test
- response
  - kinematic response
  - Penalty method
  - Impulse force of collision

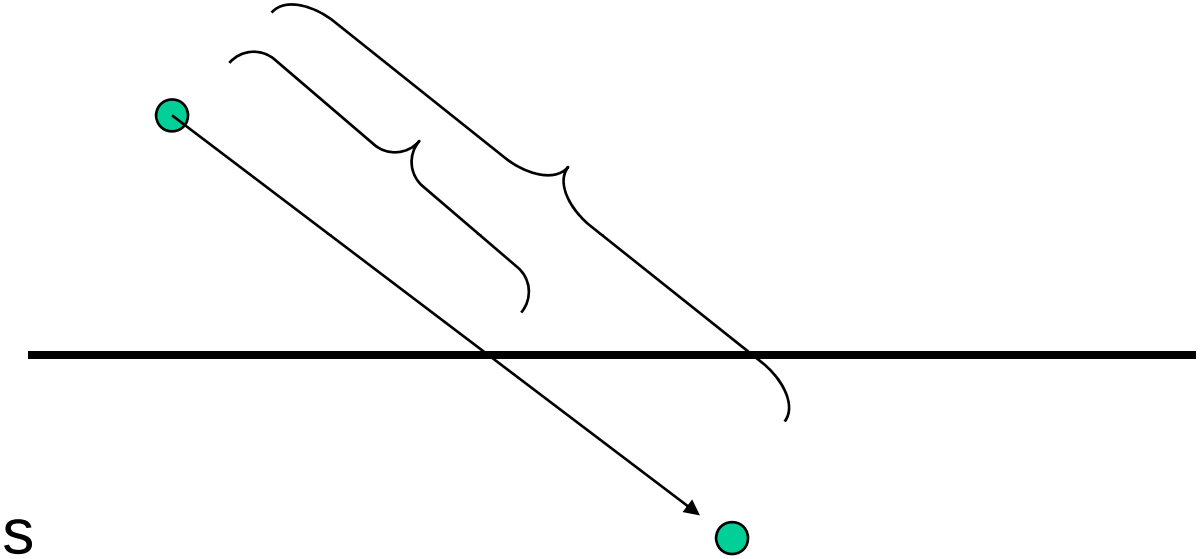
Chapter 4

Collision detection: point-plane



Chapter 4

Collision detection: time of impact



2 options

Consider collision at next time step

Compute fractional time at which collision actually occurred

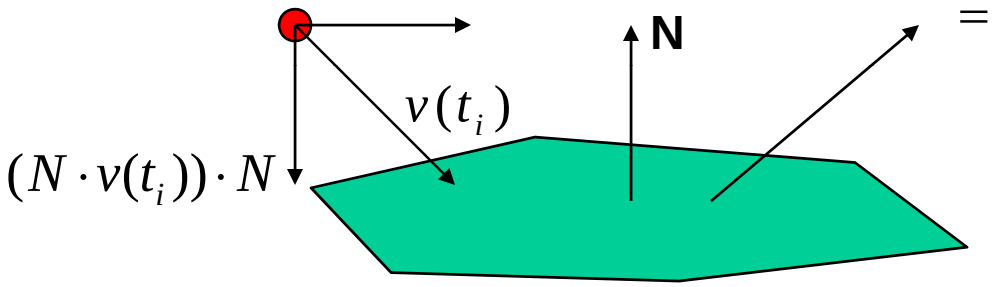
Tradeoff: accuracy v. complexity

Chapter 4

Collision response: kinematic

$$v(t_i) - (N \cdot v(t_i)) \cdot N$$

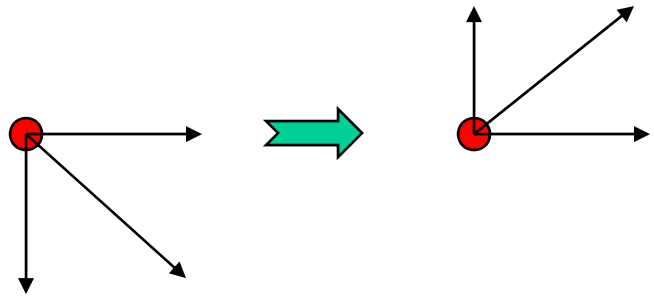
$$v(t_{i+1}) = v(t_i) - N \bullet v(t_i) \cdot N - k(N \bullet v(t_i)) \cdot N$$
$$= v(t_i) - (1+k)(N \bullet v(t_i)) \cdot N$$



k: damping factor
k=1 indicates no energy loss

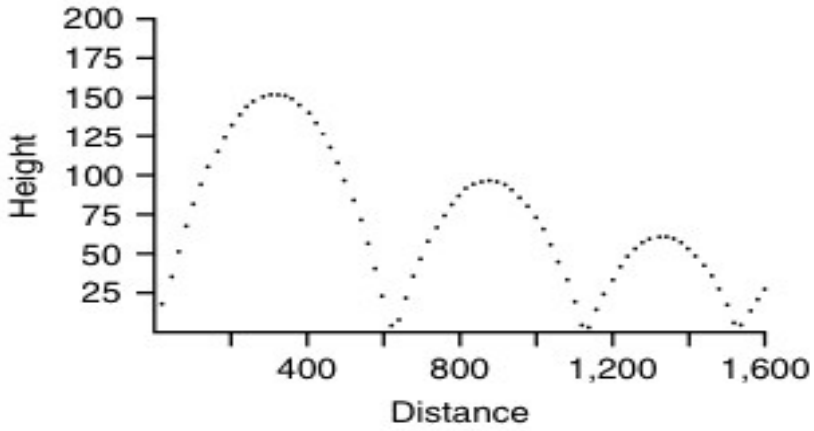
Negate component of velocity in direction of normal

No forces involved!



Chapter 4

Collision response: damped



Damping factor = 0.8

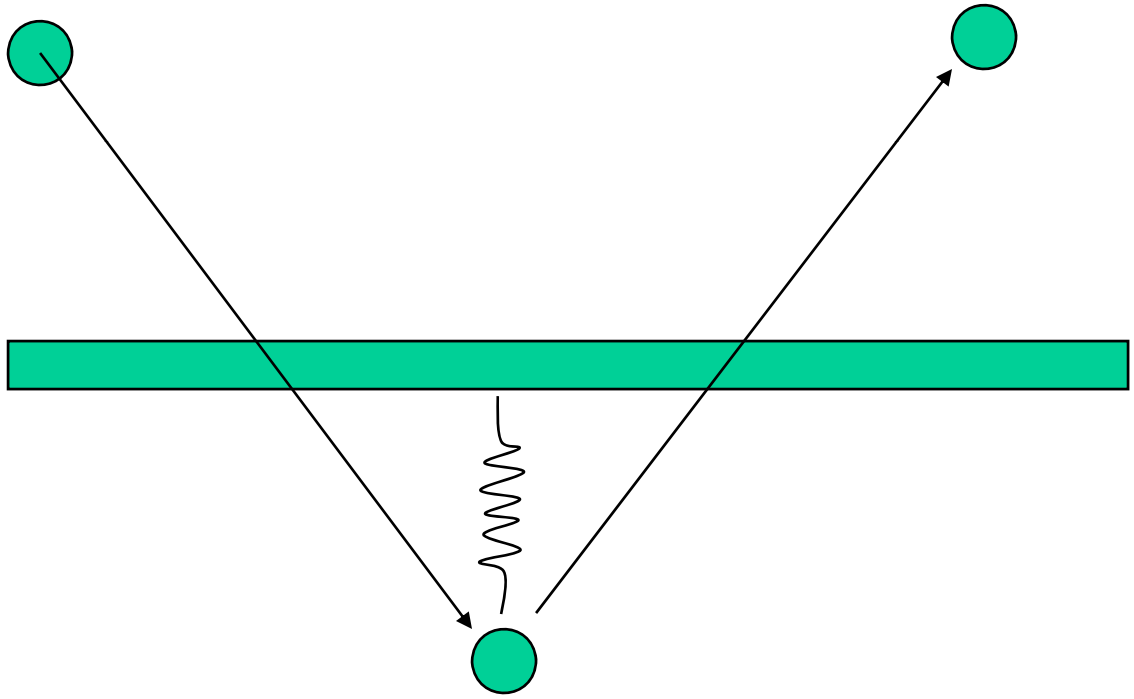
Chapter 4

The Penalty Method

As the name suggests, a points is penalized for penetrating another object. In this case, a spring, with zero rest length, is momentarily attached from the offending point to the surface it penetrated in such a way so as to impart a restoring force on the offending point. For now, it is assumed that the surface it penetrates is immovable and thus does not have to be considered as far as collision response is concerned. The closest point on the penetrated surface to the penetrating point is used as the point of attachment.

Chapter 4

Collision response – penalty method



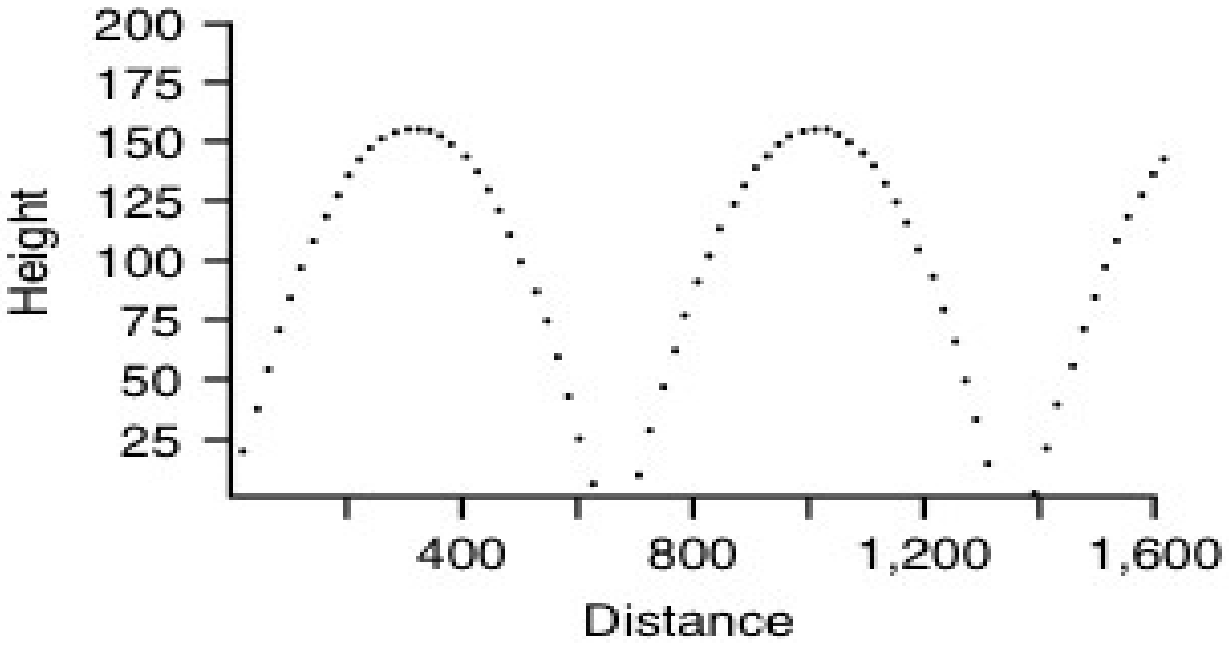
Chapter 4

The Penalty Method

The spring, therefore imparts a force on the point in the direction of the penetrated surface normal and with a magnitude according to Hooke's law ($F=-kd$). A mass assigned to the point is used to compute a resultant acceleration ($a=F/m$), which contributes an upward velocity to the point. When this upward velocity is combined with the point's original motion, the point's downward motion will be stopped and reversed by the spring, while any component of its motion tangent to the surface will be unaffected. While this is easy to implement, this approach is not ideal because it is difficult to control and does not correct the collision immediately.

Chapter 4

Collision response: penalty



Chapter 4

Collision detection: polyhedra

In environments in which objects are modeled as polyhedra, at each time step each polyhedra is tested for possible penetration against every other polyhedron. A collision is implied whenever the environment transitions from a non-penetration state to a state in which a penetration is detected. A test for penetration at each time step can miss some collisions because of the discrete temporal sampling, but it is sufficient for many applications. The geometric structure of a polyhedra can be quite complex and computing the intersection quite costly. Finding fast and efficient tests identify penetration can help speed up the algorithm.

Chapter 4

Collision detection: polyhedra

Order tests according to computational complexity and power of detection

1. test bounding volumes for overlap

2. test for vertex of one object inside of other object

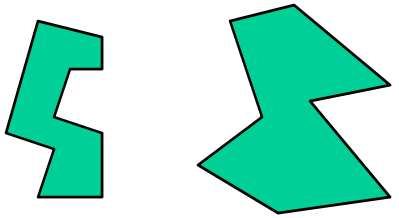
3. test for edge of one object intersecting face of other object

Chapter 4

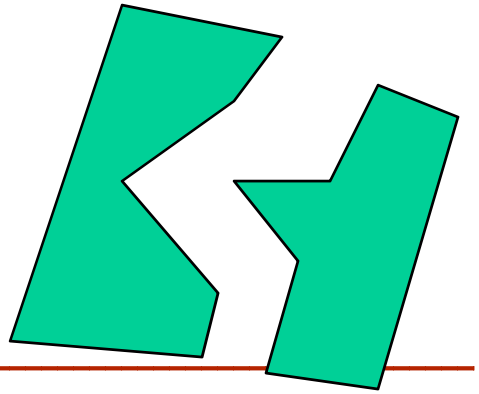
Collision detection: bounding volumes

Don’t do vertex/edge intersection testing if there’s no chance of an intersection between the polyhedra

Want a simple test to remove easy cases



Tradeoff complexity of test with power to reject non-intersecting polyhedra
(goodness of fit of bounding volume)



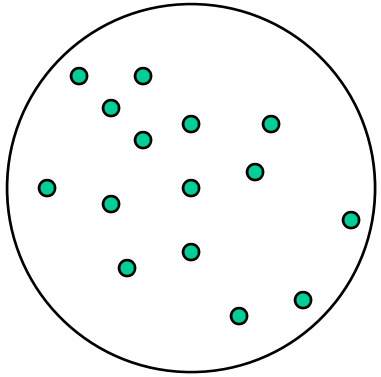
Chapter 4

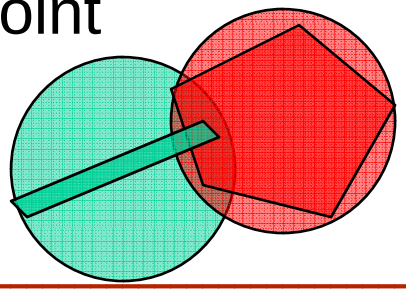
Bounding Spheres

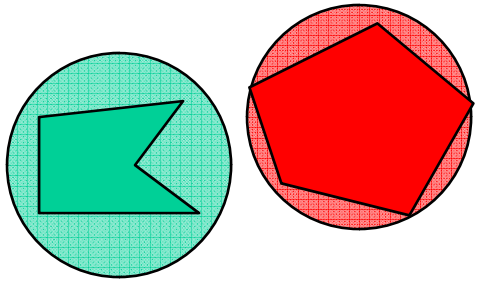
Compute bounding sphere of vertices

Compute in object space and transform with object

1. Find min/max pair of points in each dimension
2. use maximally separated pair – use to create initial bounding sphere (midpoint is center)
3. for each vertex adjust sphere to include point





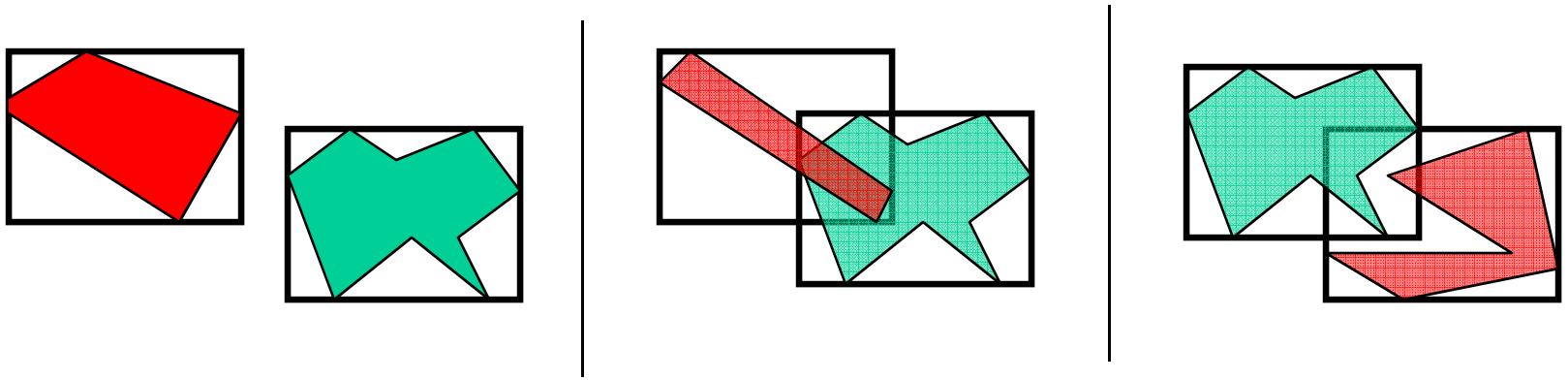


Chapter 4

Bounding Boxes

Axis-aligned (AABB): use min/max in each dimension

Oriented (OBB): e.g., use AABB in object space and transform with object. Vertex is inside of OBB iff on inside of 6 planar equations (use normal vectors e.g. always pointing to inside of OBB)

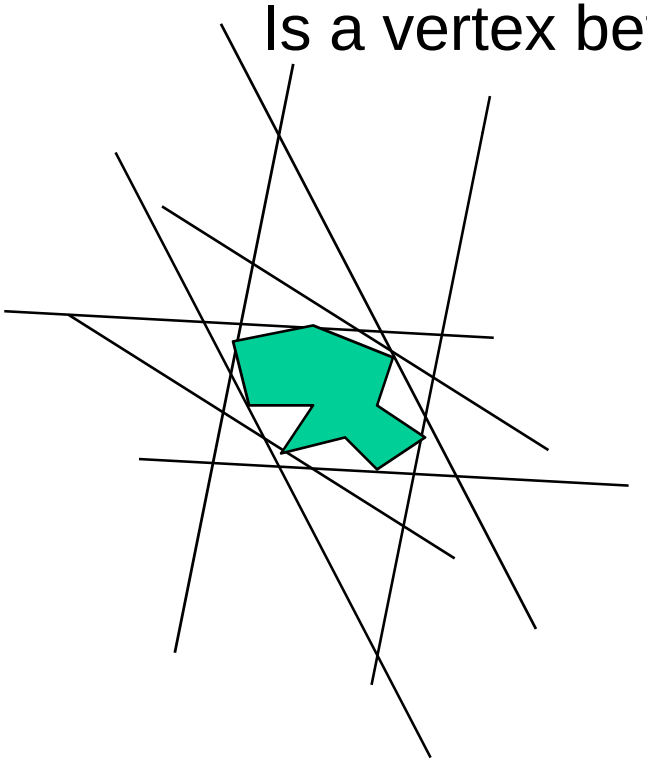


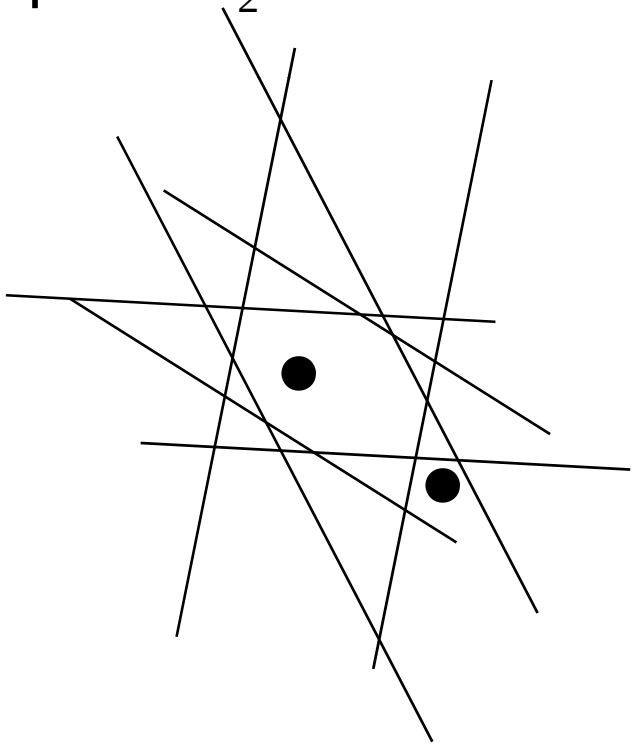
Chapter 4

Bounding Slabs

For better fit bounding polyhedron: use arbitrary (user-specified) collection of bounding plane-pairs

Is a vertex between each pair? $d_2 < N \bullet P < d_1$

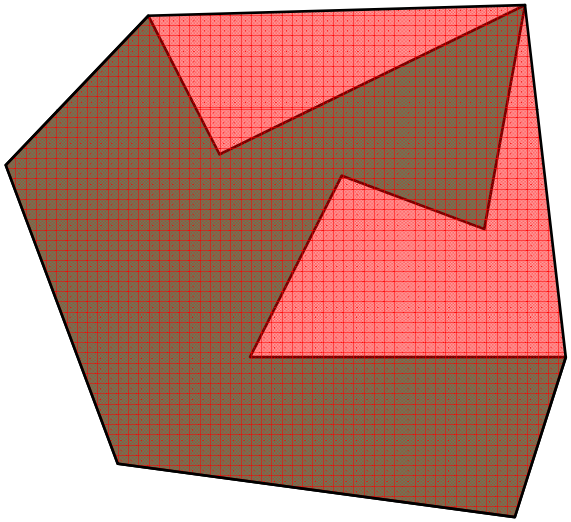




Chapter 4

Convex Hull

Best fit convex polyhedron to concave polyhedron but takes some (one-time) computation



1. Find highest vertex, V1
2. Find remaining vertex that minimizes angle with horizontal plane through point. Call edge L
3. Form plane with this edge and horizontal line perpendicular to L at V1
4. Find remaining vertex that for triangle that minimizes angle with this plane. Add this triangle to convex hull, mark edges as *unmatched*
5. For each unmatched edge, find remaining vertex that minimizes angle with the plane of the edge's triangle

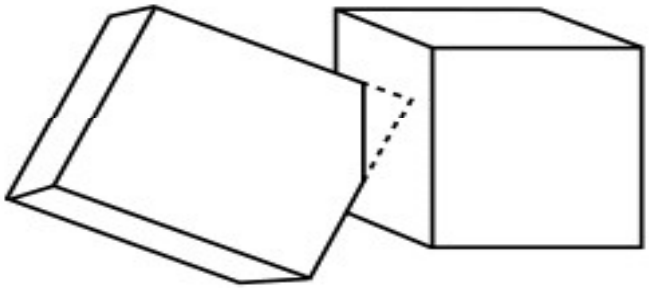
Chapter 4

Collision detection: polyhedra

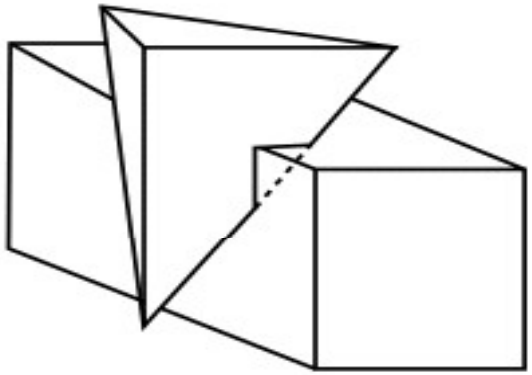
1. test bounding volumes for overlap

2. test for vertex of one object inside of other object

3. test for edge of one object intersecting face of other object



Vertex inside a polyhedron



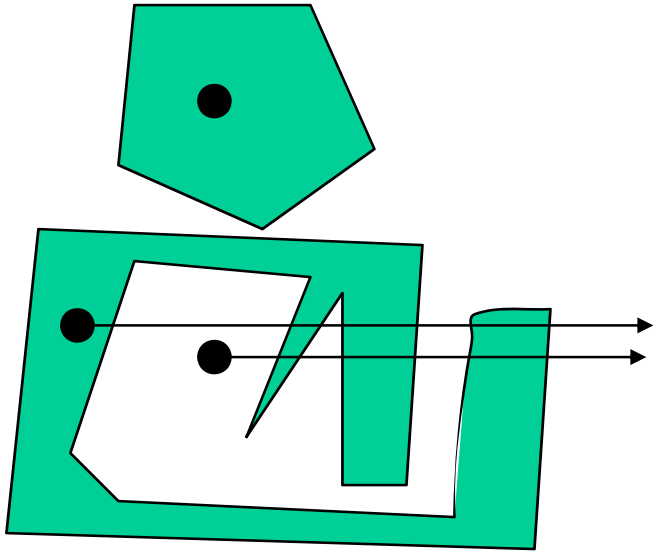
Object penetration without a vertex of one object contained in the other

Chapter 4

Collision detection: polyhedra

Intersection = NO
For each vertex, V, of object A
    if (V is inside of B) intersection = YES
For each vertex, V, of object B
    if (V is inside of A) intersection = YES

A vertex is inside a convex polyhedron if it's on the 'inside' side of all faces

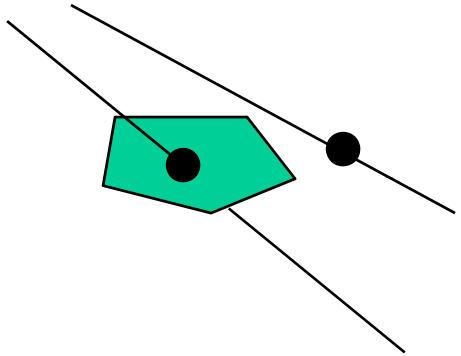


A vertex is inside a concave polyhedron if a semi-infinite ray from the vertex intersects an odd number of faces

Chapter 4

Collision detection: polyhedra

Edge intersection face test
Finds ALL polyhedral intersections
But is most expensive test



If vertices of edges are on opposite side of plane of face

Calculate intersection of edge with plane

Test vertex for inside face (2D test in plane of face)

Chapter 4

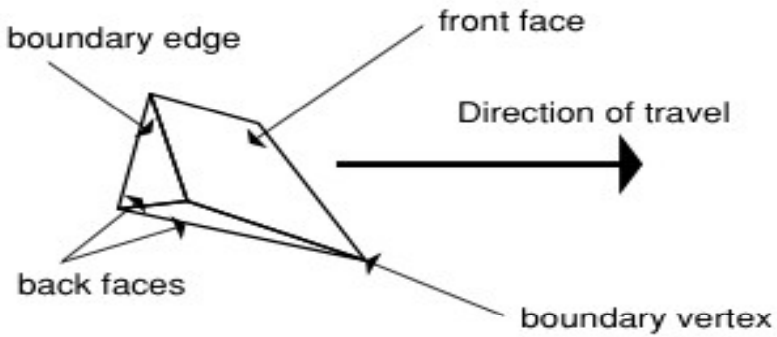
Collision detection: swept volume

In certain limited cases, a more precise determination of collision can be performed. For example, if the movement of one object with respect to another is a linear path, then the volume swept by one object can be determined and intersected with the other object, whose position is assumed to be static relative to the moving object. The volume swept by the object is defined by the original back faces, the translated positions of the front faces, and new faces and edges that are extruded in the direction of travel. This volume is intersected with the static object.

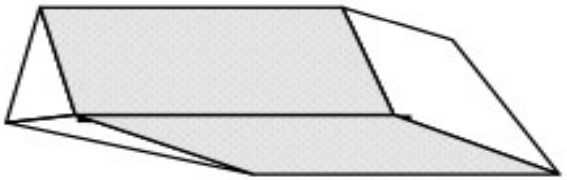
Chapter 4

Collision detection: swept volume

Time & relative direction of travel sweeps out a volume
Only tractable in simple cases (e.g. linear translation)



a) Original object

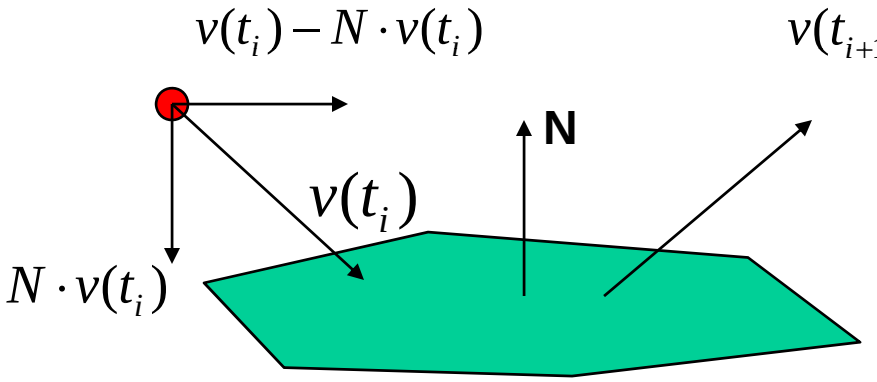


b) Swept volume

If part of an object is in the volume, it was intersected by object

Chapter 4

Collision reaction
Coefficient of restitution



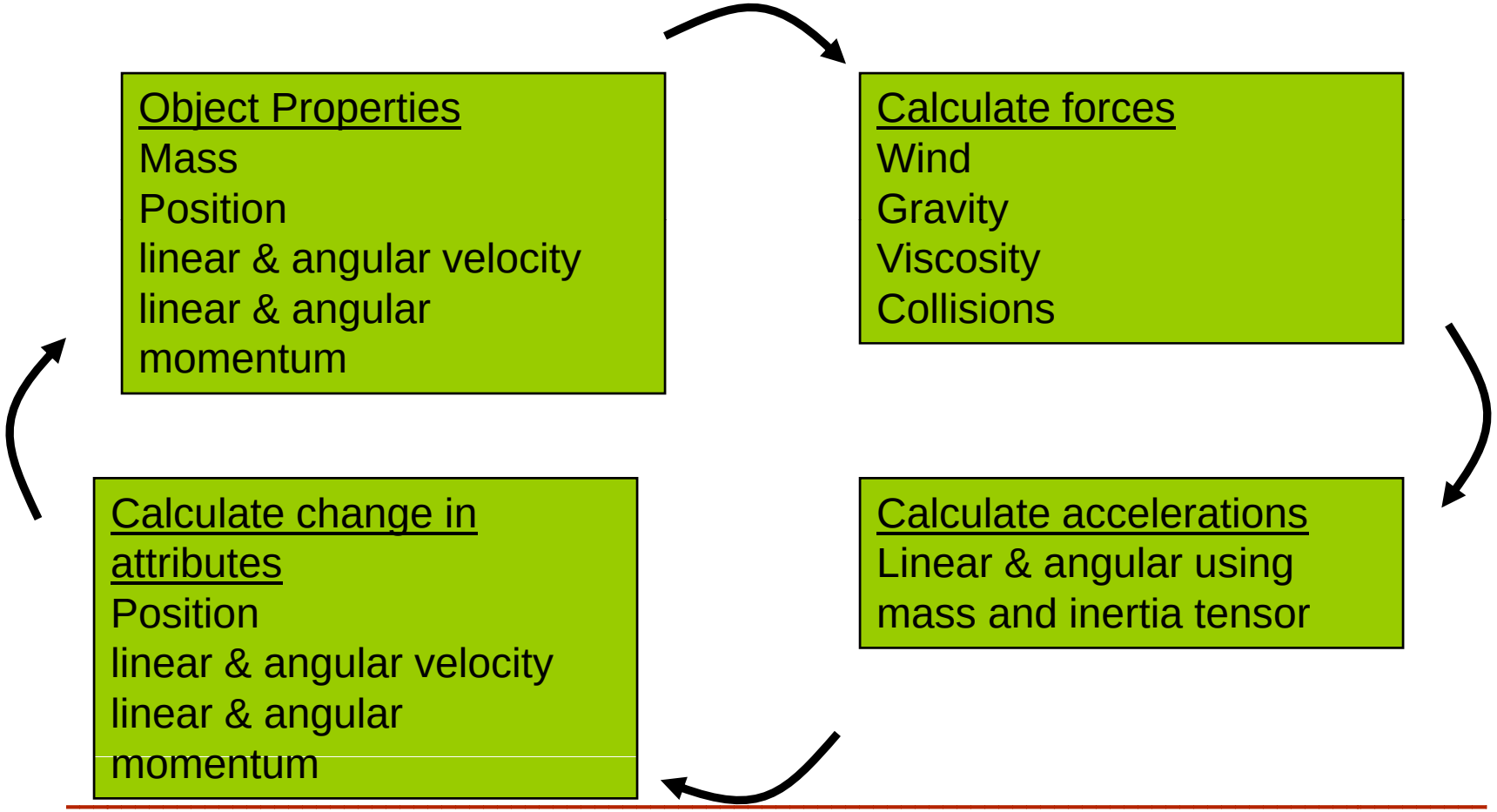
$$v(t_{i+1}) = v(t_i) - N \bullet v(t_i) - k(N \bullet v(t_i))$$
$$= v(t_i) - (1+k)N \bullet v(t_i)$$

k – coefficient of restitution

But now want to add angular velocity contribution to separation velocity

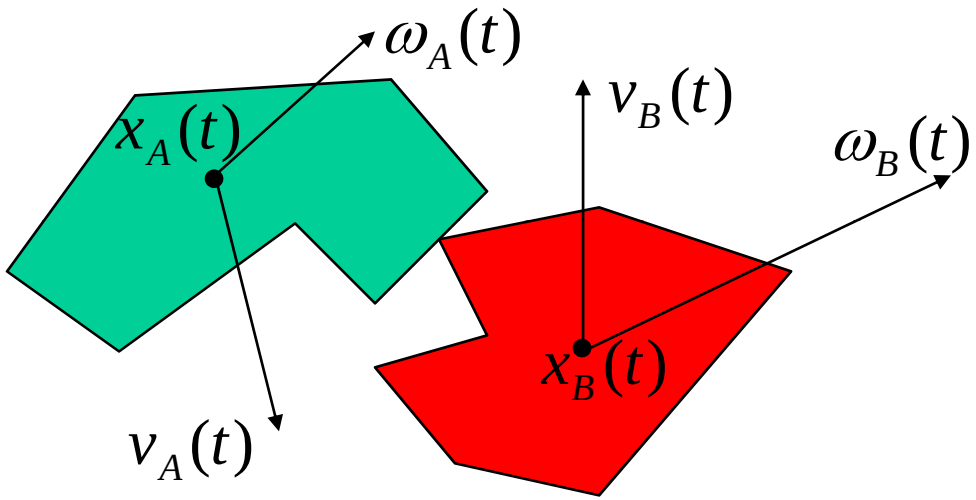
Chapter 4

Rigid body simulation



Chapter 4

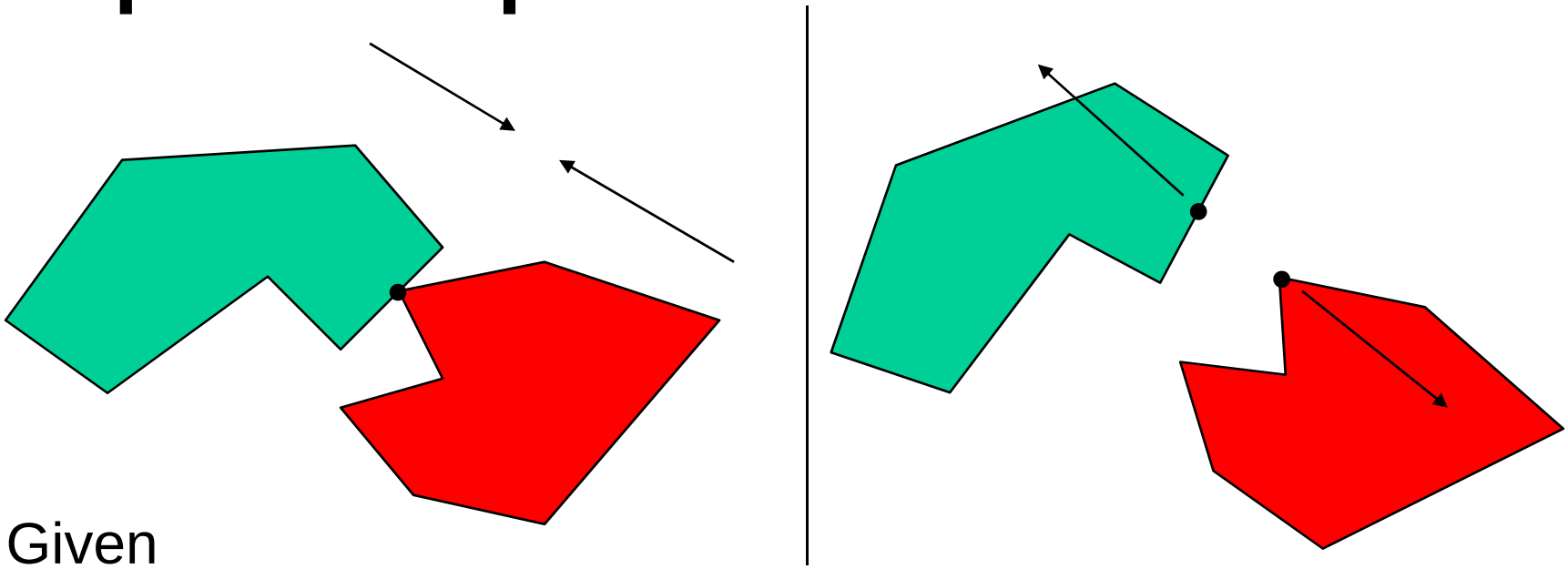
Impulse response



How to compute the collision response of two rotating rigid objects?

Chapter 4

Impulse response



Given

Separation velocity is to be negative of colliding velocity

Compute

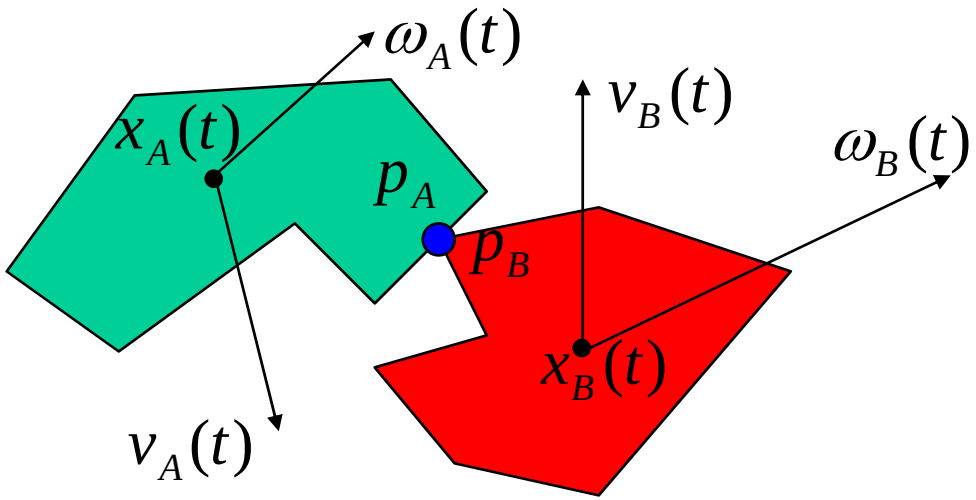
Impulse force that produces sum of linear and angular velocities that produce desired separation velocity

Chapter 4

Rigid body simulation

Separation velocity

$$v_{rel}^{+} = -\varepsilon v_{rel}^{-}$$



To characterize the elasticity of the collision response, the user selects the coefficient of restitution, which relates the relative velocity before the collision to the velocity after the collision.

Chapter 4

Inertia Tensor

Aka Angular mass

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

Terms of the matrix are calculated by integrating over the object's mass:

$$I_{xx} = \int (y^2 + z^2) dm \qquad I_{xy} = -\int xy dm$$

$$I_{yy} = \int (x^2 + z^2) dm \qquad I_{xz} = -\int xz dm$$

$$I_{zz} = \int (x^2 + y^2) dm \qquad I_{yz} = -\int yz dm$$

Chapter 4

Inertia Tensor

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

Discrete version for particle collection

$$I_{xx} = \Sigma m_i (y_i^2 + z_i^2) \qquad I_{xy} = -\Sigma m_i x_i y_i$$

$$I_{yy} = \Sigma m_i (x_i^2 + z_i^2) \qquad I_{xz} = -\Sigma m_i x_i z_i$$

$$I_{zz} = \Sigma m_i (x_i^2 + y_i^2) \qquad I_{yz} = -\Sigma m_i y_i z_i$$

m_i: mass at particle i

Chapter 4

Inertia Tensor

Symmetric wrt axes

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

Cuboid

$$I = \frac{1}{12}\begin{bmatrix} M(b^2+c^2) & 0 & 0 \\ 0 & M(a^2+c^2) & 0 \\ 0 & 0 & M(a^2+b^2) \end{bmatrix}$$

Sphere

$$I = \frac{2MR^2}{5}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

M: mass of object

Chapter 4

Inertia Tensor

Translation and rotation applied to inertia tensor:

$$I_{traslated} = \begin{bmatrix} I_{xx} + M(Y^2 + Z^2) & -I_{xy} - MXY & -I_{xy} - MXZ \\ -I_{xy} - MXY & I_{yy} + M(Y^2 + Z^2) & -I_{xy} - MYZ \\ -I_{xy} - MXZ & -I_{xy} - MYZ & I_{zz} + M(Y^2 + Z^2) \end{bmatrix}$$

$$I_{rotated} = RI_{object}R^{-1}$$

M: mass of object; (X,Y,Z): translation

Chapter 4

State of the Object

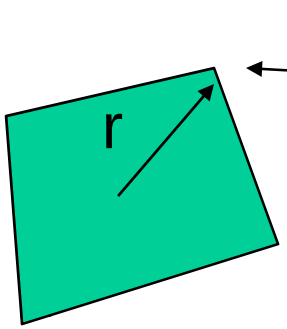
The state of an object can be described as a vector composed of its current position *x(t)*, current rotation *R(t)*, current linear momentum *P(t)*, and current angular momentum *L(t)*:

$$S(t) = \begin{bmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{bmatrix}$$

Other object attributes: mass *M*, inertia tensor I_{object}^{-1}

Chapter 4

The Equations



$$\tau = r \times F$$

$$I_{rotated}^{-1} = R I_{object}^{-1} R^{-1}$$

$$v(t) = \frac{P(t)}{M}$$

$$w(t) = I(t)^{-1} L(t)$$

$$\frac{d}{dt} S(t) = \frac{d}{dt} \begin{bmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ w(t)^{*} R(t) \\ F(t) \\ \tau(t) \end{bmatrix}$$

If using rotation matrix, will need to orthonormalize updated rotation matrix

Chapter 4

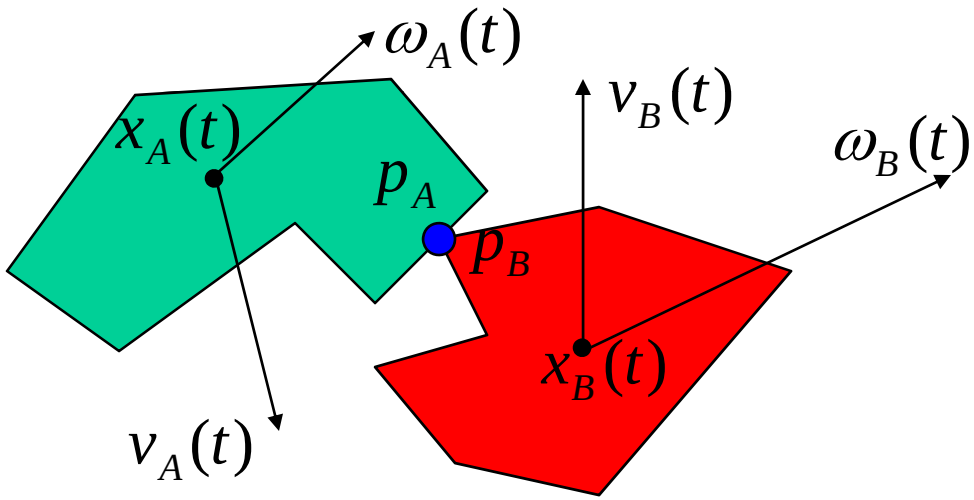
Update linear and angular velocities as a result of impulse force

$$v_A^+ = v_A^- + \frac{jn}{M_A}$$

$$v_B^+ = v_B^- + \frac{jn}{M_B}$$

$$\omega_A^+ = \omega_A^- + I_A^{-1}(t)(r_A \times jn)$$

$$\omega_B^+ = \omega_B^- + I_B^{-1}(t)(r_B \times jn)$$



j is the (unknown) impulse
n is the normal to the surface of contact

Chapter 4

Velocities of points of contact

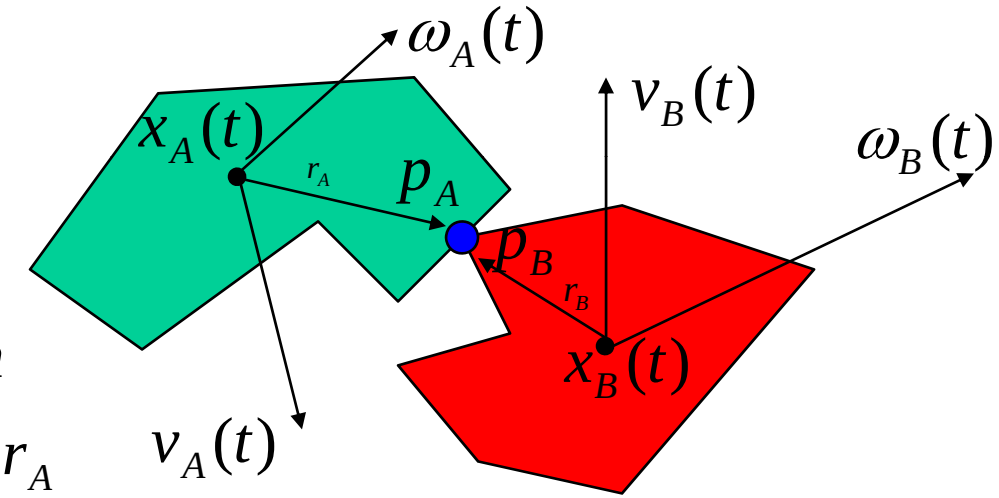
$$r_A = p_A - x_A(t)$$

$$r_B = p_B - x_B(t)$$

$$v_{rel} = (\dot{p}_A(t) - \dot{p}_B(t)) \cdot n$$

$$\dot{p}_A(t) = v_A(t) + \omega_A(t) \times r_A$$

$$\dot{p}_B(t) = v_B(t) + \omega_B(t) \times r_B$$



Chapter 4

Rigid body simulation

$$v_{rel}^{+} = n \cdot (\dot{p}_A^{+}(t) - \dot{p}_B^{+}(t))$$

$$v_{rel}^{+} = n \cdot (v_A^{+}(t) + \omega_A^{+}(t) \times r_A - v_B^{+}(t) - \omega_B^{+}(t) \times r_B)$$

$$\varepsilon v_{rel}^{-} = n \cdot (v_A^{-} + \frac{jn}{M_A} + (\omega_A^{-} + I_A^{-1}(r_A \times jn)) \times r_A - v_B^{-} - \frac{jn}{M_B} - (\omega_A^{-} + I_A^{-1}(r_A \times jn)) \times r_B)$$

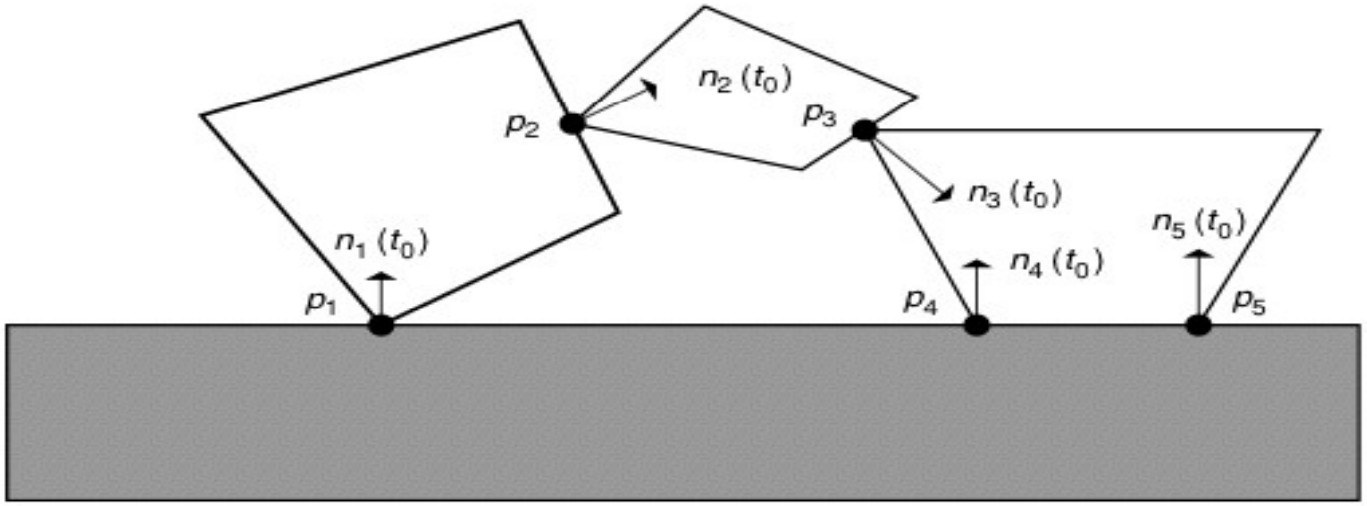
$$j = \frac{-((1+\varepsilon)v_{rel}^{-})}{\frac{1}{M_A} - \frac{1}{M_B} + n \cdot (I_A^{-1}(r_A \times n) \times r_A - I_B^{-1}(r_B \times n) \times r_B)}$$

Chapter 4

Resting Contact

Computing the forces involved in resting contact is one of the more difficult dynamics problems for computer animation. The solution requires access to quadratic programming, the implementation of which is beyond the scope of this course. An example situation in which several objects are resting on one another is shown on the next slide.

Chapter 4
Resting contact



Complex situations: need to solve for forces that prevent penetration, push objects apart, if the objects are separating, then the contact force is zero

Chapter 4

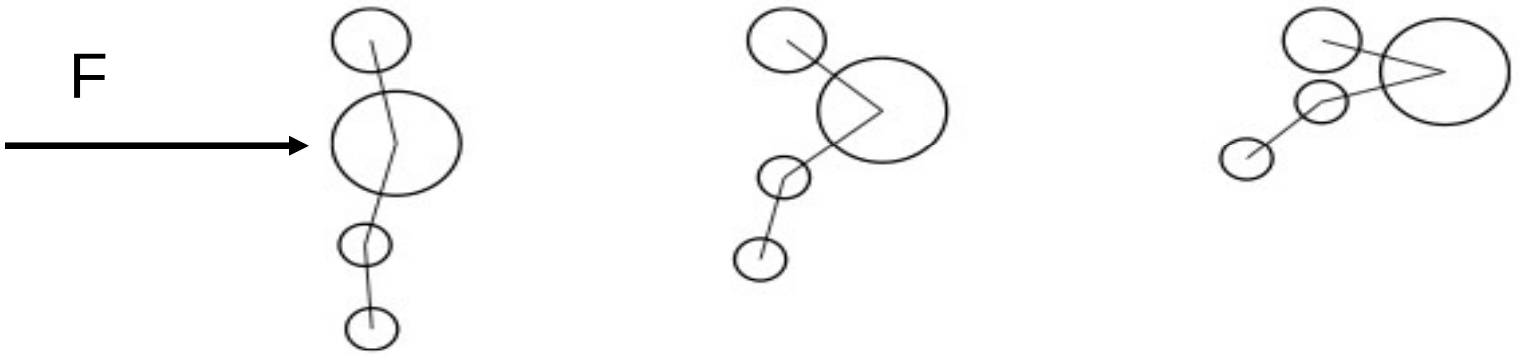
Dynamics of Linked Hierarchies

Constrained dynamics
The Featherstone equations

Chapter 4

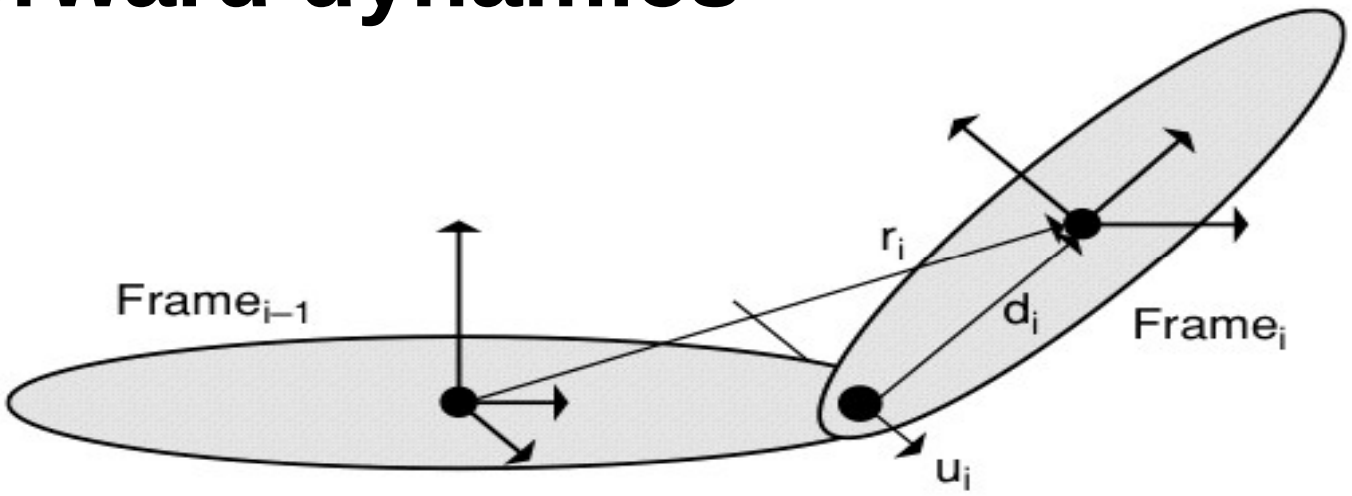
Constrained dynamics

Apply force to one component, other components repositioned, from near to far, to satisfy distance constraints



Chapter 4

Forward dynamics



Vectors relating one coordinate frame to the next: u_i is the axis of revolution associated with Frame$_i$, r_i is the displacement vector form the center of Frame$_{i-1}$ to the center of Frame$_i$; d_i is the displacement vector form the axis of revolution to the center of Frame$_i$.

Preliminaries

Links numbered 0 to n

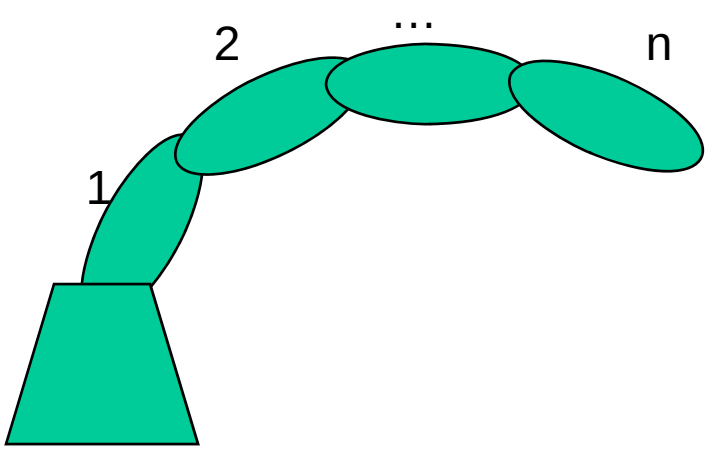
Fixed base: link 0; Outermost like: link n

Joints numbered 1 to n

Link i has inboard joint, Joint i

Each joint has 1 DoF

Vector of joint positions: $q=(q_1,q_2,\ldots q_n)^T$



The Problem

- Given:
  - the **positions q and velocities** $\dot{q}$ of the n joints of a serial linkage,
  - the **external forces** acting on the linkage,
  - and the **forces and torques** being applied by the joint actuators
- Find: The resulting **accelerations** of the joints: $\ddot{q}$

First Determine equations that give absolute motion of all links

Given: the joint positions q, velocities and accelerations

Compute: for each link the linear and angular velocity and acceleration relative to an inertial frame

Notation

v_i Linear velocity of link i

a_i Linear acceleration of link i

ω_i Angular velocity of link i

α_i Angular acceleration of link i

Joint variables

q_i joint position

$\dot{q}_i$ joint velocity

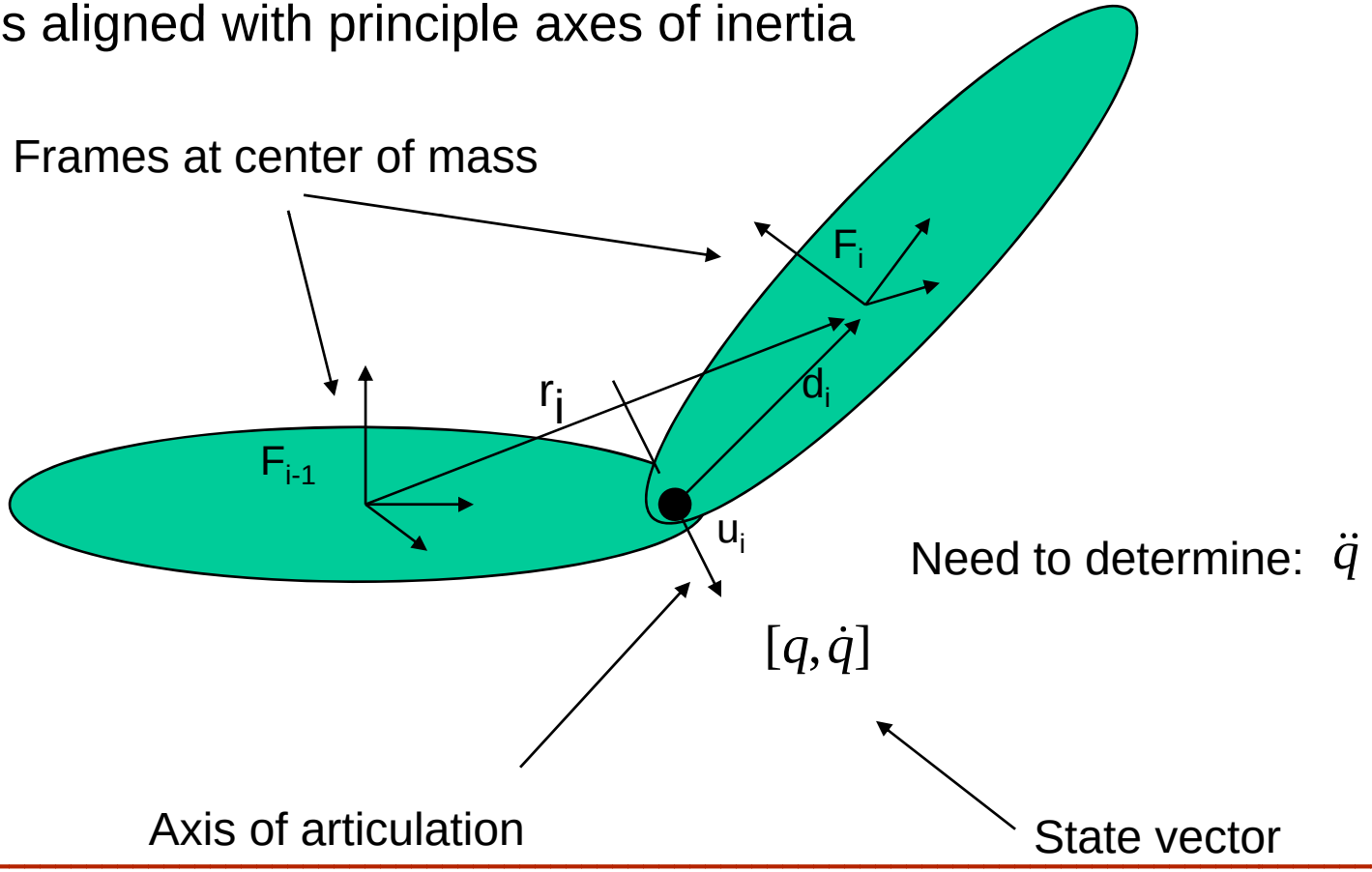
u_i Unit vector in direction of the axis of joint i

r_i vector from origin of F_{i-1} to origin of F_i

d_i vector from axis of joint i to origin of F_i

Basic terms

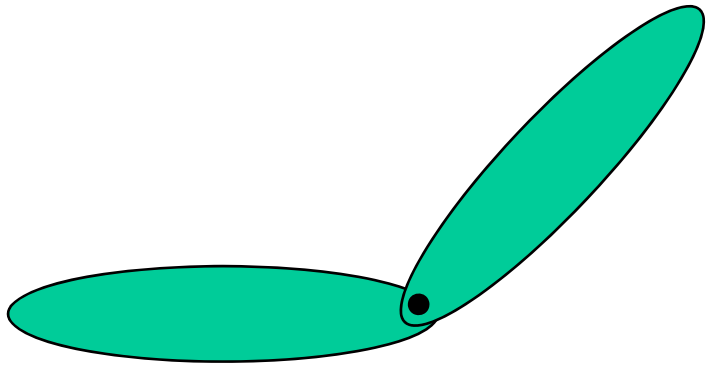
- F_i – body frame of link i
- Origin at center of mass
- Axes aligned with principle axes of inertia



From base outward

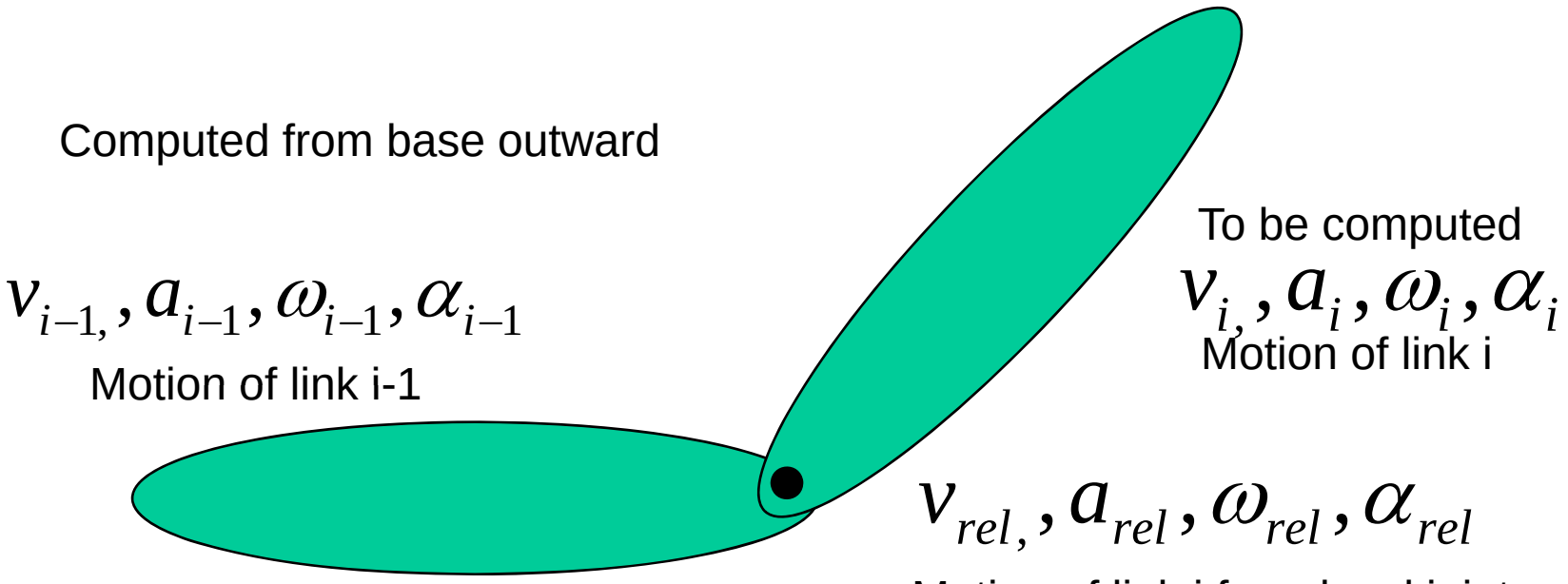
Velocities and accelerations of link i are completely determined by:

- the velocities and accelerations of link i-1
- and the motion of joint i



First – determine velocities and accelerations

From velocity and acceleration of previous link, determine total (global) velocity and acceleration of current link



Compute outward

Angular velocity of link i =

angular velocity of link i-1 plus

angular velocity induced by rotation at joint i

$$\omega_i = \omega_{i-1} + \omega_{rel}$$

Linear velocity =

linear velocity of link i-1 plus

linear velocity induced by rotation at link -1 plus

linear velocity from translation at joint i

$$v_i = v_{i-1} + \omega_{i-1} \times r_i + v_{rel}$$

Compute outward

Angular acceleration propagation $\alpha_i = \alpha_{i-1} + \dot{\omega}_{rel}$

Linear acceleration propagation

$$a_i = a_{i-1} + \alpha_{i-1} \times r_i + \omega_{i-1} \times \dot{r}_i + \dot{v}_{rel}$$

Rewritten, using $\dot{r}_i = v_i - v_{i-1}$ (relative velocity) and $v_i = v_{i-1} + \omega_{i-1} \times r_i + v_{rel}$ (from previous slide)

$$\dot{r}_i = \omega_{i-1} \times r_i + v_{rel}$$

$$a_i = a_{i-1} + \alpha_{i-1} \times r_i + \omega_{i-1} \times (\omega_{i-1} \times r_i) + \omega_{i-1} \times v_{rel} + \dot{v}_{rel}$$

Define w_{rel} and v_{rel} and their time derivatives

Joint velocity vector	Joint acceleration vector
Axis times parametric velocity	Axis times parametric acceleration
$\upsilon_i = \dot{q}_i u_i$	$\xi_i = \ddot{q}_i u_i$ (unkown)

prismatic	revolute
$\omega_{rel} = 0$	$\omega_{rel} = \upsilon_i$
$v_{rel} = \upsilon_i$	$v_{rel} = \upsilon_i \times d_i$

Velocity propagation formulae

(revolute)

linear

$$v_i = v_{i-1} + \omega_{i-1} \times r_i + v_{rel}$$

$$v_{rel} = \upsilon_i \times d_i$$

$$v_i = v_{i-1} + \omega_{i-1} \times r_i + \upsilon_i \times d_i$$

angular

$$\omega_i = \omega_{i-1} + \omega_{rel}$$

$$\omega_{rel} = \upsilon_i$$

$$\omega_i = \omega_{i-1} + \upsilon_i$$

Time derivatives of v_{rel} and w_{rel}

(revolute)

Joint acceleration vector

Change in joint velocity vector

$$\dot{\omega}_{rel} = \xi_i + \omega_{i-1} \times \upsilon_i$$

$$\dot{v}_{rel} = 2\omega_{i-1} \times (\upsilon_i \times d_i) + \xi_i \times d_i + \upsilon_i \times (\upsilon_i \times d_i)$$

From change in joint velocity vector

From joint acceleration vector

From change in change in vector from joint to CoM

Explanation see next slide

Derivation of $\dot{v}_{rel}$

(revolute)

$$\upsilon_i = \dot{q}_i u_i$$

$$\dot{\upsilon}_i = \ddot{q}u_i + \dot{q}\dot{u}_i = \xi_i + \dot{q}\dot{u} \qquad \dot{u}_i = \omega_{i-1} \times u_i$$

$$\dot{\upsilon}_i = \xi_i + \omega_{i-1} \times \upsilon_i$$

$$\frac{d}{dt}(v_{rel}) = \frac{d}{dt}(\upsilon_i \times d_i) = \dot{\upsilon}_i \times d_i + \upsilon_i \times \dot{d}_i$$

$$\dot{d}_i = \omega_i \times d_i = (\omega_{i-1} + \upsilon_i) \times d_i$$

$$\dot{v}_{rel} = 2\omega_{i-1} \times (\upsilon_i \times d_i) + \xi_i \times d_i + \upsilon_i \times (\upsilon_i \times d_i)$$

Acceleration propagation formulae

(revolute)

linear

$$a_i = a_{i-1} + \alpha_{i-1} \times r_i + \omega_{i-1} \times \dot{r}_i + \dot{v}_{rel}$$

Previously derived

$$\dot{v}_{rel} = 2\omega_{i-1} \times (v_i \times d_i) + \xi_i \times d_i + v_i \times (v_i \times d_i)$$

$$\dot{r}_i = \omega_{i-1} \times r_i + v_{rel}$$

$$a_i = a_{i-1} + \alpha_{i-1} \times r_i + \xi_i \times d_i + \omega_{i-1} \times (\omega_{i-1} \times r_i) + 2\omega_{i-1} \times (v_i \times d_i) + v_i \times (v_i \times d_i)$$

angular

$$\alpha_i = \alpha_{i-1} + \dot{\omega}_{rel}$$

$$\dot{\omega}_{rel} = \xi_i + \omega_{i-1} \times v_i$$

$$\alpha_i = \alpha_{i-1} + \xi_i + \omega_{i-1} \times v_i$$

First step in forward dynamics

Use known dynamic state:

Compute absolute linear and angular velocities: $q, \dot{q}$

Remember: Acceleration propagation equations involve unknown joint accelerations

$$v, \omega$$

But first – need to introduce notation to facilitate equation writing

Spatial Algebra

Spatial Algebra

Spatial velocity

$$\hat{v} = \begin{bmatrix} \omega \\ v \end{bmatrix}$$

Spatial acceleration

$$\hat{a} = \begin{bmatrix} \alpha \\ a \end{bmatrix}$$

This spatial notation combines angular and linear components. It has been developed to make the computations more concise. The spatial vectors consists of the angular and linear velocity and acceleration, respectively.

Spatial Transform Matrix

r – offset vector R– rotation

$$\hat{v}_G = {}_G\hat{X}_F \hat{v}_F$$

$$\hat{a}_G = {}_G\hat{X}_F \hat{a}_F$$

Transformation from frame F to frame G:

$${}_G\hat{X}_F = \begin{bmatrix} R & 0 \\ -\tilde{r}R & R \end{bmatrix}$$

(cross product operator)

cross product operator:

$$\tilde{r} = \begin{bmatrix} 0 & -r_z & -r_y \\ r_z & 0 & -r_x \\ r_y & r_x & 0 \end{bmatrix}$$

Spatial Algebra

Spatial force/torque

$$\hat{f} = \begin{bmatrix} f \\ \tau \end{bmatrix}$$

Spatial transpose

$$\hat{x}' = \begin{bmatrix} a \\ b \end{bmatrix}' = \begin{bmatrix} b^T & a^T \end{bmatrix}$$

Spatial joint axis

$$\hat{s}_i = \begin{bmatrix} u_i \\ u_i \times d_i \end{bmatrix}$$

(used later)

Spatial inner product

$$\hat{x}'\hat{y}$$

Compute Serial Link Velocities

(revolute)

For i = 1 to N do

$$\omega_0, v_0, \alpha_0, a_0 \leftarrow 0$$

R←rotation matrix from frame i-1 to i

r←radius vector from frame i-1 to frame i (in frame i coordinates)

$$\omega_i \leftarrow R\omega_{i-1}$$

$$v_i \leftarrow Rv_{i-1} + \omega_i \times r$$

$$\left.\begin{aligned} \omega_i &\leftarrow \omega_i + \dot{q}_i u_i \\ v_i &\leftarrow v_i + \dot{q}_i (u_i \times d_i) \end{aligned}\right\}$$

Specific to revolute joints

end

Spatial formulation of acceleration propagation

Previously: (revolute)

$$\alpha_i = \alpha_{i-1} + \xi_i + \omega_{i-1} \times \upsilon_i$$

$$a_i = a_{i-1} + \alpha_{i-1} \times r_i + \xi_i \times d_i + \omega_{i-1} \times (\omega_{i-1} \times r_i) + 2\omega_{i-1} \times (\upsilon_i \times d_i) + \upsilon_i \times (\upsilon_i \times d_i)$$

Want to put in form:

$$\hat{a}_i = {}_i\hat{X}_{i-1}\hat{a}_{i-1} + \ddot{q}_i\hat{s} + \hat{c}_i$$

Where: ${}_G\hat{X}_F = \begin{bmatrix} R & 0 \\ -\tilde{r}R & R \end{bmatrix}$ $\hat{s}_i = \begin{bmatrix} u_i \\ u_i \times d_i \end{bmatrix}$

Spatial Coriolis force

(revolute)

$$\alpha_i = \alpha_{i-1} + \xi_i + \omega_{i-1} \times \upsilon_i$$

$$a_i = a_{i-1} + \alpha_{i-1} \times r_i + \xi_i \times d_i + \omega_{i-1} \times (\omega_{i-1} \times r_i) + 2\omega_{i-1} \times (\upsilon_i \times d_i) + \upsilon_i \times (\upsilon_i \times d_i)$$

$$\hat{a}_i = {}_i\hat{X}_{i-1}\hat{a}_{i-1} + \ddot{q}_i\hat{s}_i + \hat{c}_i$$

$$\hat{s}_i = \begin{bmatrix} u_i \\ u_i \times d_i \end{bmatrix}$$

These are the terms involving $\xi_i = \ddot{q}_i u_i$

$$\hat{c}_i \leftarrow \begin{bmatrix} \omega_{i-1} \times \upsilon_i \\ \omega_{i-1} \times (\omega_{i-1} \times r_i) + 2\omega_{i-1} \times (\upsilon_i \times d_i) + \upsilon_i \times (\upsilon_i \times d_i) \end{bmatrix}$$

Featherstone algorithm

$\hat{a}_i$ Spatial acceleration of link i

$\hat{f}_i^I$ Spatial force exerted on link i through its inboard joint

$\hat{f}_i^O$ Spatial force exerted on link i through its outboard joint

All expressed in frame i

Forces expressed as acting on center of mass of link i

Serial linkage articulated motion

$$\hat{f}_i^{\,I} = \hat{I}_i^{\,A}\hat{a}_i + \hat{Z}_i^{\,A}$$

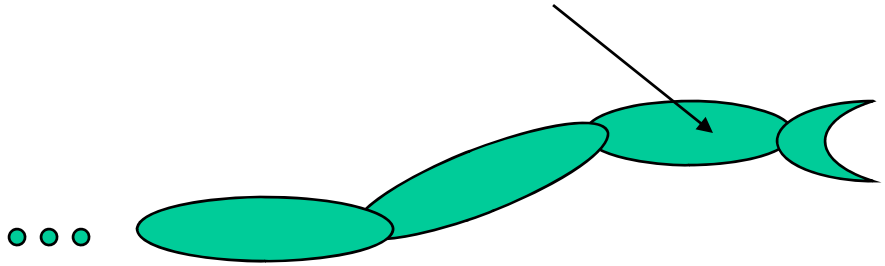
$\hat{I}_i^{\,A}$ Spatial articulated inertia of link I; *articulated* means entire subchain is being considered

$\hat{Z}_i^{\,A}$ Spatial articulated zero acceleration force of link I (independent of joint accelerations); force exerted by inboard joint on link i, if link i is not to accelerate

Develop equations by induction

Base Case

Consider last link of linkage (link n)



Force/torque applied by inboard joint + gravity = inertia*accelerations of link

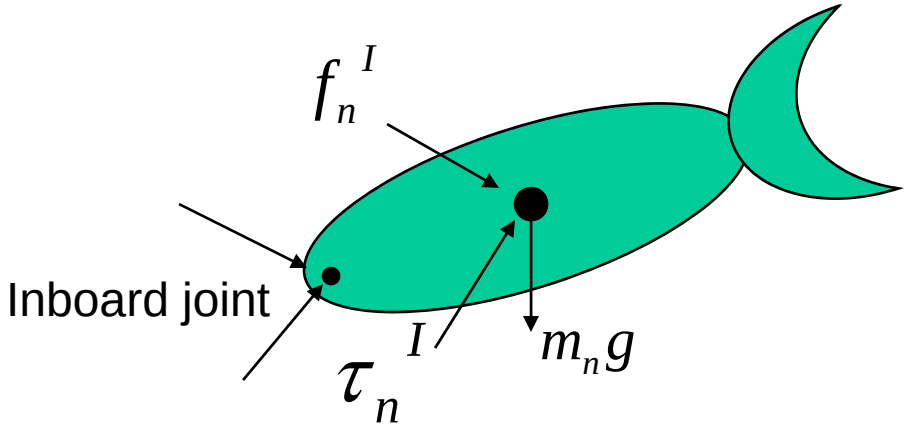
Newton-Euler equations of motion

$$\hat{f}_n^{\,I} + m_n g = m_n a_n$$

$$\tau_n^{\,I} = I_n \alpha_n + \omega_n \times I_n \omega_n$$

Using spatial notation

Link n



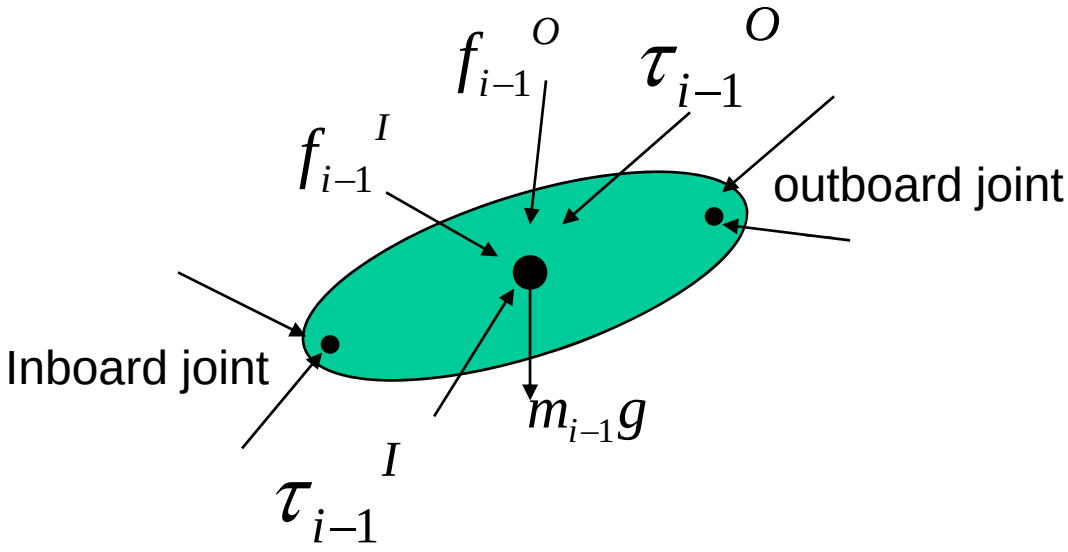
$$\begin{bmatrix} f_n^{\ I} \\ \tau_n^{\ I} \end{bmatrix} = \begin{bmatrix} 0 & M_n \\ I_n & 0 \end{bmatrix} \begin{bmatrix} \alpha_n \\ a_n \end{bmatrix} + \begin{bmatrix} -m_n g \\ \omega_n \times I_n \omega_n \end{bmatrix}$$

$$\hat{f}_n = \hat{I}_n^{\ A} \hat{a}_n + \hat{Z}_n^{\ A}$$

Inductive case

Assume previous is true for link i; consider link i-1

Link i-1



$$\begin{bmatrix} f_{i-1}^{I} \\ \tau_{i-1}^{I} \end{bmatrix} + \begin{bmatrix} f_{i-1}^{O} \\ \tau_{i-1}^{O} \end{bmatrix} = \begin{bmatrix} 0 & M_{i-1} \\ I_{i-1} & 0 \end{bmatrix} \begin{bmatrix} \alpha_{i-1} \\ a_{i-1} \end{bmatrix} + \begin{bmatrix} -m_{i-1}g \\ \omega_{i-1} \times I_{i-1}\omega_{i-1} \end{bmatrix}$$

Inductive case

$$\begin{bmatrix} f_{i-1}^{\ I} \\ \tau_{i-1}^{\ I} \end{bmatrix} + \begin{bmatrix} f_{i-1}^{\ O} \\ \tau_{i-1}^{\ O} \end{bmatrix} = \begin{bmatrix} 0 & M_{i-1} \\ I_{i-1} & 0 \end{bmatrix} \begin{bmatrix} \alpha_{i-1} \\ a_{i-1} \end{bmatrix} + \begin{bmatrix} -m_{i-1} g \\ \omega_{i-1} \times I_{i-1} \omega_{i-1} \end{bmatrix}$$

$$\hat{f}_{i-1}^{\ I} = \hat{I}_{i-1} \hat{a}_{i-1} + \hat{Z}_{i-1} - \hat{f}_{i-1}^{\ O}$$

The effect of joint i on link i-1 is equal and opposite to its effect on link i

$$\hat{f}_{i-1}^{\ O} = -{}_{i-1}X_i \ \hat{f}_i^{\ I}$$

Substituting…

$$\hat{f}_{i-1}^{\ I} = \hat{I}_{i-1} \hat{a}_{i-1} + \hat{Z}_{i-1} + {}_{i-1}X_i \ \hat{f}_i^{\ I}$$

Inductive case

$$\hat{f}_{i-1}^{\,I} = \hat{I}_{i-1}\hat{a}_{i-1} + \hat{Z}_{i-1} + {}_{i-1}X_i\,\hat{f}_i^{\,I}$$

Invoking induction on the definition of $\hat{f}_i^{\,I}$

$$\hat{f}_{i-1}^{\,I} = \hat{I}_{i-1}\hat{a}_{i-1} + \hat{Z}_{i-1} + {}_{i-1}X_i\,(\hat{I}_i^{\,A}\hat{a}_i + \hat{Z}_i^{\,A})$$

Inductive case

$$\hat{f}_{i-1}^{\ I} = \hat{I}_{i-1}\hat{a}_{i-1} + \hat{Z}_{i-1} + {}_{i-1}X_i(\hat{I}_i^{\ A}\hat{a}_i + \hat{Z}_i^{\ A})$$

Express a_i in terms of a_{i-1} and rearrange

$$\hat{a}_i = {}_iX_{i-1}\hat{a}_{i-1} + \ddot{q}_i\hat{s}_i + \hat{c}_i$$

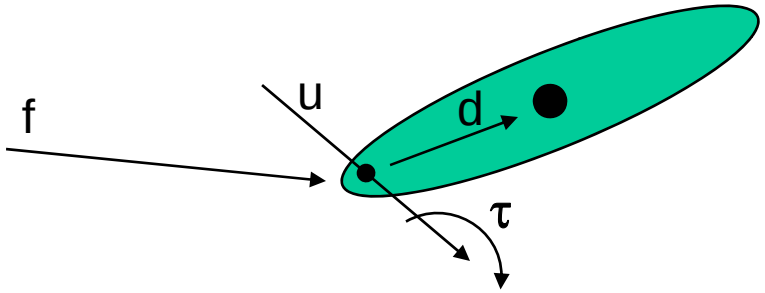
$$\hat{f}_{i-1}^{\ I} = (\hat{I}_{i-1} + {}_{i-1}X_i\hat{I}_i^{\ A}\,{}_iX_{i-1})\hat{a}_{i-1} + \hat{Z}_{i-1} + {}_{i-1}X_i[\hat{Z}_i^{\ A} + \hat{I}_i^{\ A}\hat{c}_i + (\hat{I}_i^{\ A}\hat{s}_i)\ddot{q}_i]$$

Need to eliminate from the right side of the equation $\ddot{q}_i$

Inductive case

$$\hat{f}_{i-1}^{\ I} = (\hat{I}_{i-1} + {}_{i-1}X_i \hat{I}_i^{\ A}\ {}_iX_{i-1})\hat{a}_{i-1} + \hat{Z}_{i-1} + {}_{i-1}X_i[\hat{Z}_i^{\ A} + \hat{I}_i^{\ A}\hat{c}_i + (\hat{I}_i^{\ A}\hat{s}_i)\ddot{q}_i]$$

Magnitude of torque exerted by revolute joint actuator is Q_i



A force f and a torque τ applied to link i at the inboard joint give rise to a spatial inboard force (resolved in the body frame) of

$$\hat{f}_i^{\ I} = \begin{bmatrix} f_i \\ \tau_i - d_i \times f_i \end{bmatrix}$$

$$Q_i = \hat{s}_i' \hat{f}_i^{\ I} = \begin{bmatrix} u_i \\ u_i \times d_i \end{bmatrix}' \begin{bmatrix} f_i \\ \tau_i - d_i \times f_i \end{bmatrix} = f_i \cdot (u_i \times d_i) + (\tau_i - d_i \times f_i) \cdot u_i = \tau_i \cdot u_i$$

Moment of force

Moment of force

Inductive case

$$Q_i = \hat{s}_i' \hat{f}_i^I = \tau_i \cdot u_i$$

previously $\hat{a}_i = {}_i\hat{X}_{i-1}\hat{a}_{i-1} + \ddot{q}_i\hat{s}_i + \hat{c}_i$ and $\hat{f}_i^I = \hat{I}_i^A\hat{a}_i + \hat{Z}_i^A$

$$\hat{f}_i^I = \hat{I}_i^A({}_i\hat{X}_{i-1}\hat{a}_{i-1} + \ddot{q}_i\hat{s}_i + \hat{c}_i) + \hat{Z}_i^A$$

Premultiply both sides by $\hat{s}_i'$ substitute Q_i for s'f , and solve

$$\ddot{q}_i = \frac{Q_i - \hat{s}_i'\hat{I}_i^A{}_i\hat{X}_{i-1}\hat{a}_{i-1} - \hat{s}_i'(\hat{Z}_i^A + \hat{I}_i^A\hat{c}_i)}{\hat{s}_i'\hat{I}_i^A\hat{s}_i}$$

And substitute

$$\ddot{q}_i = \frac{Q_i - \hat{s}_i' \hat{I}_i^A {}_i\hat{X}_{i-1}\hat{a}_{i-1} - \hat{s}_i' (\hat{Z}_i^A + \hat{I}_i^A \hat{c}_i)}{\hat{s}_i' \hat{I}_i^A \hat{s}_i}$$

$$\hat{f}_{i-1}^I = (\hat{I}_{i-1} + {}_{i-1}X_i \hat{I}_i^A {}_i X_{i-1})\hat{a}_{i-1} + \hat{Z}_{i-1} + {}_{i-1}X_i [\hat{Z}_i^A + \hat{I}_i^A \hat{c}_i + (\hat{I}_i^A \hat{s}_i)\ddot{q}_i]$$

$$\hat{f}_{i-1}^I = [\hat{I}_{i-1} + {}_{i-1}X_i (\hat{I}_i^A - \frac{\hat{I}_i^A \hat{s}_i \hat{s}_i' \hat{I}_i^A}{\hat{s}_i' \hat{I}_i^A \hat{s}_i}) {}_i X_{i-1}]\hat{a}_{i-1} +$$

$$\hat{Z}_{i-1} + {}_{i-1}X_i [\hat{Z}_i^A + \hat{I}_i^A \hat{c}_i + \frac{\hat{I}_i^A \hat{s}_i [Q_i - \hat{s}_i' \hat{I}_i^A {}_i\hat{X}_{i-1}\hat{a}_{i-1} - \hat{s}_i' (\hat{Z}_i^A + \hat{I}_i^A \hat{c}_i)]}{\hat{s}_i' \hat{I}_i^A \hat{s}_i}]$$

And form I & Z terms

$$\hat{f}_{i-1}^{\ I} = [\hat{I}_{i-1} + {}_{i-1}\hat{X}_i(\hat{I}_i^{\ A} - \frac{\hat{I}_i^{\ A}\hat{s}_i\hat{s}_i'\hat{I}_i^{\ A}}{\hat{s}_i'\hat{I}_i^{\ A}\hat{s}_i})_i\hat{X}_{i-1}]\hat{a}_{i-1} +$$

$$\hat{Z}_{i-1} + {}_{i-1}\hat{X}_i[\hat{Z}_i^{\ A} + \hat{I}_i^{\ A}\hat{c}_i + \frac{\hat{I}_i^{\ A}\hat{s}_i[Q_i - \hat{s}_i'\hat{I}_i^{\ A}{}_i\hat{X}_{i-1}\hat{a}_{i-1} - \hat{s}_i'(\hat{Z}_i^{\ A} + \hat{I}_i^{\ A}\hat{c}_i)]}{\hat{s}_i'\hat{I}_i^{\ A}\hat{s}_i}]$$

To get into form: $$\hat{f}_i^{\ I} = \hat{I}_i^{\ A}\hat{a}_i + \hat{Z}_i^{\ A}$$

$$\hat{I}_{i-1}^{\ A} = \hat{I}_{i-1} + {}_{i-1}\hat{X}_i(\hat{I}_i^{\ A} - \frac{\hat{I}_i^{\ A}\hat{s}_i\hat{s}_i'\hat{I}_i^{\ A}}{\hat{s}_i'\hat{I}_i^{\ A}\hat{s}_i})_i\hat{X}_{i-1}$$

$$\hat{Z}_{i-1}^{\ A} = \hat{Z}_{i-1} + {}_{i-1}\hat{X}_i[\hat{Z}_i^{\ A} + \hat{I}_i^{\ A}\hat{c}_i + \frac{\hat{I}_i^{\ A}\hat{s}_i[Q_i - \hat{s}_i'\hat{I}_i^{\ A}{}_i\hat{X}_{i-1}\hat{a}_{i-1} - \hat{s}_i'(\hat{Z}_i^{\ A} + \hat{I}_i^{\ A}\hat{c}_i)]}{\hat{s}_i'\hat{I}_i^{\ A}\hat{s}_i}]$$

Ready to put into code

Using

- Loop from inside out to compute velocities previously developed (repeated on next slide)
- Loop from inside out to initialize I, Z, and c variables
- Loop from outside in to propagate I, Z and c updates
- Loop from inside out to compute $\ddot{q}$ using I, Z, c

Compute Serial Link Velocities

(revolute)

// This is code from an earlier slide – loop inside out to compute velocities

For i = 1 to N do

$$\omega_0, v_0, \alpha_0, a_0 \leftarrow 0$$

R←rotation matrix from frame i-1 to i

r←radius vector from frame i-1 to frame i (in frame i coordinates)

$$\omega_i \leftarrow R\omega_{i-1}$$

$$v_i \leftarrow Rv_{i-1} + \omega_i \times r$$

$$\omega_i \leftarrow \omega_i + \dot{q}_i u_i$$

$$v_i \leftarrow v_i + \dot{q}_i (u_i \times d_i)$$

} Specific to revolute joints

end

Init Serial Links

(revolute)

// loop from inside out to initialize Z, I, c variables

For i = 1 to N do

$$\hat{Z}_i^A \leftarrow \begin{bmatrix} -m_i g \\ \omega_i \times I_i \omega_i \end{bmatrix}$$

$$\hat{I}_i^A \leftarrow \begin{bmatrix} 0 & M_i \\ I_i & 0 \end{bmatrix}$$

$$\hat{c}_i \leftarrow \begin{bmatrix} \omega_{i-1} \times v_i \\ \omega_{i-1} \times (\omega_{i-1} \times r_i) + 2\omega_{i-1} \times (v_i \times d_i) + v_i \times (v_i \times d_i) \end{bmatrix}$$

end

Serial Forward Dynamics

// new code with calls to 2 previous routines

Call compSerialLinkVelocities

Call initSerialLinks

// loop outside in to form I and Z for each link

For i = n to 2 do

$$\hat{I}_{i-1}^{A} \leftarrow \hat{I}_{i-1}^{A} +_{i-1}\hat{X}_i[\hat{I}_i^{A} - \frac{\hat{I}_i^{A}\hat{s}_i\hat{s}_i'\hat{I}_i^{A}}{\hat{s}_i'\hat{I}_i^{A}\hat{s}_i}]_i\hat{X}_{i-1}$$

$$\hat{Z}_{i-1}^{A} \leftarrow \hat{Z}_{i-1}^{A} +_{i-1}\hat{X}_i[\hat{Z}_i^{A} + \hat{I}_i^{A}\hat{c}_i + \frac{\hat{I}_i^{A}\hat{s}_i[Q_i - \hat{s}_i'(\hat{Z}_i^{A} + \hat{I}^{A}{}_i\hat{c}_i)]}{\hat{s}_i'\hat{I}_i^{A}\hat{s}_i}]$$

// loop inside out to compute link and joint accelerations

$\hat{a}_0 \leftarrow 0$

For i = 1 to n do

$$\ddot{q}_i = \frac{Q_i - \hat{s}_i'\hat{I}_i^{A}{}_i\hat{X}_{i-1}\hat{a}_{i-1} - \hat{s}_i'(\hat{Z}_i^{A} + \hat{I}_i^{A}\hat{c}_i)}{\hat{s}_i'\hat{I}_i^{A}\hat{s}_i}$$

$$\hat{a}_i =_i \hat{X}_{i-1}\hat{a}_{i-1} + \hat{c}_i + \ddot{q}_i\hat{s}_i$$

Chapter 4

And that’s all there is to it!

Chapter 4

$$\omega_i = \omega_{i-1} + \omega_{rel}$$
$$v_i = v_{i-1} + \omega_{i-1} \times r_i + v_{rel}$$
$$\alpha_i = \alpha_{i-1} + \dot{\omega}_{rel}$$
$$a_i = a_{i-1} + \alpha_{i-1} \times r_i + \omega_{i-1} \times \dot{r}_i + \dot{v}_{rel}$$
$$a_i = a_{i-1} + \alpha_{i-1} \times r_i + \omega_{i-1} \times (\omega_{i-1} \times r_i) + \omega_{i-1} \times v_{rel} + \dot{v}_{rel}$$
$$\dot{\omega}_{rel} = \xi_i + \omega_{i-1} \times \upsilon_i$$
$$\dot{v}_{rel} = \omega_{i-1} \times (\upsilon_i \times d_i) + \xi_i \times d_i + \upsilon_i \times (\upsilon_i \times d_i)$$
$$\frac{d}{dt}(v_{rel}) = \frac{d}{dt}(\upsilon_i \times d_i) = \dot{\upsilon}_i \times d_i + \upsilon_i \times \dot{d}_i$$
$$\dot{d}_i = \omega_i \times d_i = (\omega_{i-1} + \upsilon_i) \times d_i$$
$$\dot{v}_{rel} = 2\omega_{i-1} \times (\upsilon_i \times d_i) + \xi_i \times d_i + \upsilon_i \times (\upsilon_i \times d_i)$$

$$\dot{r}_i = v_i - v_{i-1}$$
$$\dot{r}_i = \omega_{i-1} \times r_i + v_{rel}$$

$$\upsilon_i = \dot{q}_i u_i \quad \omega_{rel} = \upsilon_i$$
$$\xi_i = \ddot{q}_i u_i \quad v_{rel} = \upsilon_i \times d_i$$

$$\dot{\upsilon}_i = \xi_i + \omega_{i-1} \times \upsilon_i$$

$$v_i = v_{i-1} + \omega_{i-1} \times r_i + \upsilon_i \times d_i$$
$$\omega_i = \omega_{i-1} + \upsilon_i$$

Chapter 4

$$a_i = a_{i-1} + \alpha_{i-1} \times r_i + \xi_i \times d_i + \omega_{i-1} \times (\omega_{i-1} \times r_i) + 2\omega_{i-1} \times (\upsilon_i \times d_i) + \upsilon_i \times (\upsilon_i \times d_i)$$

$$\alpha_i = \alpha_{i-1} + \xi_i + \omega_{i-1} \times \upsilon_i$$

Chapter 4

$$\hat{v} = \begin{bmatrix} \omega \\ v \end{bmatrix} \quad \hat{a} = \begin{bmatrix} \alpha \\ a \end{bmatrix} \qquad {}_G\hat{X}_F = \begin{bmatrix} R & 0 \\ -\tilde{r}R & R \end{bmatrix}$$

$$\hat{f} = \begin{bmatrix} f \\ \tau \end{bmatrix} \quad \hat{x}' = \begin{bmatrix} a \\ b \end{bmatrix}' = \begin{bmatrix} b^T & a^T \end{bmatrix}$$

$$\hat{s}_i = \begin{bmatrix} u_i \\ u_i \times d_i \end{bmatrix} \qquad \hat{x}'\hat{y}$$

Chapter 4

Enforcing Soft and Hard Constraints

Energy minimization
Space-time constraints

Chapter 4

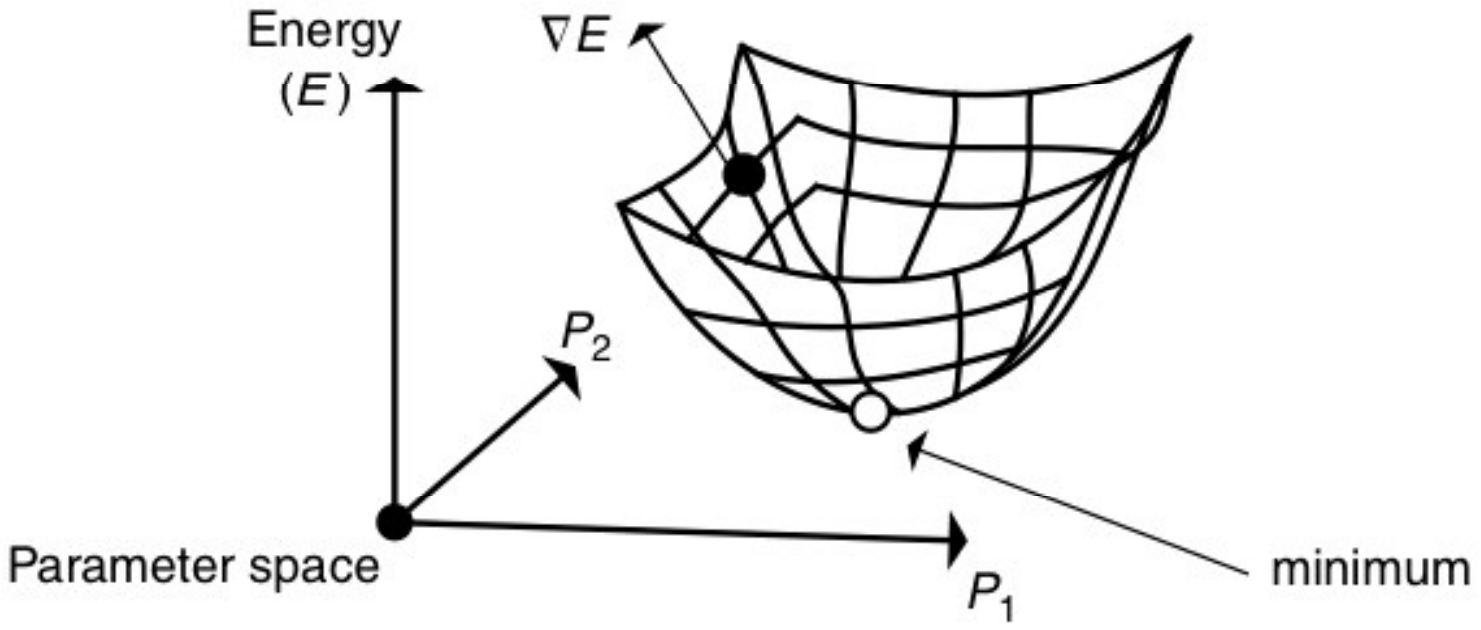
Energy Minimization

The concept of a system's energy can be used in a variety of ways to control the motion of objects. Used in this sense, energy is not defined solely as physically realizable but its free to be defined in whatever form servers the animator. Energy functions can be used to pin objects together, to restore the shape of objects, or to minimize the curvature in a spline as it interpolates points in space.

Energy constraints induce restoring forces based on arbitrary functions of a model's parameters. A constraint is phrased in terms of a non-negative smooth energy function, E, and a local minimum is searched by evaluating the gradient of the energy function and stepping in the direction of the negative of the gradient, $-\nabla E$.

Chapter 4

Energy minimization



Chapter 4

Space-Time Constraints

Space-time constraints view motion as a solution to a constrained optimization problem that takes place over time in space. Hard constraints which include equations of motion as well as non-penetration constraints and locating the object at certain points in time and space are placed on the object. An objective function that is to be optimized is stated; for example, minimize the amount of force required to produce the motion over some time interval.

Chapter 4

Example Space-Time Particle

A time-varying force fucntion, $f(t)$, is responsible for moving a particle located at $x(t)$. Its equation of motion is described by:

$$m\ddot{x}(t) - f(t) - mg = 0$$

Assuming we want to minimize fuel consumption, approximated by $|f|^2$, between times t_0 and t_1 we will have to minimize the following functional:

$$R = \int_{t_0}^{t_1} |f|^2$$