
CMSC 498M: Chapter 8b Physics: Kinetics and Integration

Reading:

- **Real-Time Game Physics**, by Graham Rhodes, 2005 (Chapt 4.3 in "Introduction to Game Development" by S. Rabin).
- **Physics for Game Developers**, by David M. Bourg, 2002.

Overview:

- Force and Torque
- Kinetics and the laws of force and motion
- Collision detection and response
- Numerical integration (Euler and Verlet)
- Motion constraints

Chapter 8, Slide 1
Copyright © David Mount and Amitabh Varshney

Overview

- **Torque**
- **Kinetics** (for particles and rigid bodies)
- **Collision Detection and Response**
- **Integration**
- **Motion Constraints**

Chapter 8, Slide 2
Copyright © David Mount and Amitabh Varshney

Force and Torque

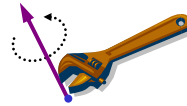
Force:

- Induces **linear acceleration** (a) in accordance with Newton's 2nd Law.
- **Units:** lbs, Newtons.



Torque:

- **Twisting** force.
- Induces **angular acceleration:** α .
- Forces inducing torque are applied at some **distance** from the axis of rotation. Turning a 1-foot long wrench has a different effect than turning a 2-foot long wrench.
- **Units:** foot-lbs, Newton-meters.



Chapter 8, Slide 3
Copyright © David Mount and Amitabh Varshney

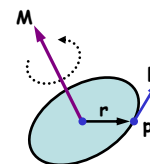
Force and Torque

Torque:

- Torque is represented by a **vector**, denoted by M .
- As with all rotations, the direction is along the **axis of rotation**, and directed according to the **right-hand rule**.
- Consider a **force F** applied to a **point p** and let **r be the vector** from the center of rotation to p . Then:

$$M = r \times F$$

- When multiple forces are involved, the total torque is the vector **sum of individual torques**.
- Note that the center of rotation **need not** coincide with the center of mass. (Because of various force constraints.)
- All the **laws of kinetics** that we will present **apply equally to torque** and angular velocity.



Chapter 8, Slide 4
Copyright © David Mount and Amitabh Varshney

Overview

- Torque
- **Kinetics** (for particles and rigid bodies)
- Collision Detection and Response
- Integration
- Motion Constraints

Chapter 8, Slide 5
Copyright © David Mount and Amitabh Varshney

Kinetics

Kinetics: (also called Dynamics)

- Explains the effects of **forces** on the **motion** of objects.
- Not to be confused with **kinematics**, which does not involve force.

Fundamental Equations of Motion: (Newton's 2nd Law)

Force: $F = ma$.

Torque: $M = I\alpha$.

When force and torque vary as **functions of time** these are more accurately given in terms of **derivatives:**

Force: $F(t) = m (dv/dt)$.

Torque: $M(t) = I (d\omega/dt)$.

(Where I is **angular momentum** with respect to axis of rotation.)

Kinetics for Games:

- Given a set of **objects and forces** acting on them (both applied and reactive), determine new **accelerations, velocities, and positions**.

Chapter 8, Slide 6
Copyright © David Mount and Amitabh Varshney

Kinetics

Typical Steps in a Kinetic Simulation System:

1. Calculate **body properties**: mass, center of mass, moment of inertia.
2. For each body, identify/quantify all **forces/torques**:
 - a. This will generally involve **collision detection** and **collision response**, to be discussed later.
 - b. Calculate the **vector sums** of all these forces.
3. Solve **equations of motion** to determine **new object accelerations**.
4. **Integrate accelerations** over time to determine **new velocities**.
5. **Integrate velocities** over time to determine **new positions**.
6. **Render** scene with new object positions.



1. Body Properties



2a. Identify Forces



2b. Sum Forces



5. New Position

Chapter 8, Slide 7
Copyright © David Mount and Amitabh Varshney

Typical (Generic) Physics Engine Code

```
#include <...> // your standard includes
#include <SomePhysicsEngine.h> // your favorite physics engine
int main( ... ) {
    SPE_World* world; // create the root physics object
    SPE_Box* box = world->createBox(); // create a box
    box->setPosition( 0, 5, 0 ); // initialize box position
    box->setSize( 1, 1, 1 ); // ... dimensions
    box->setMass( 1 ); // ... mass
    box->setLinearVelocity( 2, 3, -1 ); // ... and velocity
    for ( int i = 0; i < 100; i++ ) { // run 100 simulation steps
        box->setForce( 0, -9.8f, 0 ); // apply some force (e.g. gravity)
        world->update( 0.02f ); // advance time 0.02sec
        SPE_Vector3 pos = box->getPosition(); // get current box location
        // ...insert code to redraw everything...
    }
    world->cleanup(); // done---clean up
}
```

Chapter 8, Slide 8
Copyright © David Mount and Amitabh Varshney

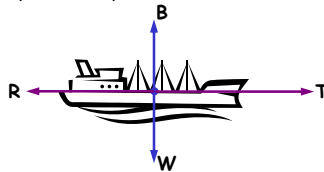
Particle Kinetics: An Example

Particle Kinetics: A simple 2-d example to see how kinetics works.

- Consider a **ship** of mass m floating slowly on surface of ocean.
- We treat the object as a **particle** in order to **ignore rotation**.

Physical forces:

- Propeller generates a constant **thrust force** T along $+x$ direction.
- Ship's **weight** due to gravity is W .
- **Buoyancy force** is B , where $B = -W$.
- **Fluid drag** results in a force R that resists velocity. Assuming slow (viscous) flow, $R = c \cdot v$, where c is the **coefficient of drag**.



Chapter 8, Slide 9
Copyright © David Mount and Amitabh Varshney

Particle Kinetics: An Example

Initial State:

Time: t . Initial time is $t_0 = 0$.

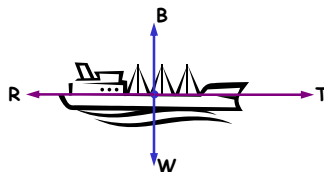
Position: $s = s(t)$. Initial position is at origin, so $s_0 = 0$.

Velocity: $v = v(t)$. Initial velocity is given as v_0 .

Acceleration: $a = a(t)$. Acceleration will be computed as a function of forces.

Objective: Solve for position, $s(t)$.

Approach: Compute velocity from net forces.



Chapter 8, Slide 10
Copyright © David Mount and Amitabh Varshney

Particle Kinetics: An Example

Equations of Motion:

- **Vertical forces:** cancel, and so can be ignored.
- **Net horizontal force:** $F(t) = T - R = T - c \cdot v(t)$.

Step 1: Solve for velocity, $v(t)$:

- Using $F(t) = m (dv/dt)$, we have $F(t) dt = m dv$. Subbing for $F(t)$:

$$(T - c \cdot v(t)) dt = m dv$$

$$\int_0^t dt = \int_{v_0}^{v(t)} \frac{m}{T - c \cdot v(t)} dv$$

$$t - 0 = \left[-\frac{m}{c} \ln(T - c \cdot v(t)) \right]_{v_0}^{v(t)}$$

$$t = -\frac{m}{c} (\ln(T - c \cdot v(t)) - \ln(T - c \cdot v_0))$$

$$\frac{c \cdot t}{m} = \ln \left(\frac{T - c \cdot v_0}{T - c \cdot v(t)} \right)$$

Chapter 8, Slide 11
Copyright © David Mount and Amitabh Varshney

Particle Kinetics: An Example

Step 1: Solve for Velocity:

- To solve for $v(t)$, take exponentials:

$$\frac{c \cdot t}{m} = \ln \left(\frac{T - c \cdot v_0}{T - c \cdot v(t)} \right)$$

$$e^{c \cdot t / m} = \frac{T - c \cdot v_0}{T - c \cdot v(t)}$$

$$T - c \cdot v(t) = (T - c \cdot v_0) e^{-c \cdot t / m}$$

$$v(t) = \frac{T}{c} - \left(\frac{T}{c} - v_0 \right) e^{-c \cdot t / m} \quad \leftarrow \text{Final result}$$

- This yields the desired **velocity**, as a function of time.
- Observe that **in the limit**, the velocity approaches T/c .

Chapter 8, Slide 12
Copyright © David Mount and Amitabh Varshney

Particle Kinetics: An Example

Step 2: Solve for Position:

- To obtain the **position**, we **integrate velocity** $v(t)$ over time. Recall that $v(t) = ds/dt$, and so:

$$v(t) dt = ds$$

$$\int_0^t \frac{T}{c} - \left(\frac{T}{c} - v_0 \right) e^{-c\tau/m} d\tau = \int_0^{s(t)} ds$$

$$\left[\frac{T}{c} \tau + \left(\frac{T}{c} - v_0 \right) \frac{m}{c} e^{-c\tau/m} \right]_0^t = s(t) - 0$$

$$\frac{T}{c} t + \left(\frac{T}{c} - v_0 \right) \frac{m}{c} (e^{-c\tau/m} - 1) = s(t) \quad \leftarrow \text{Final result}$$

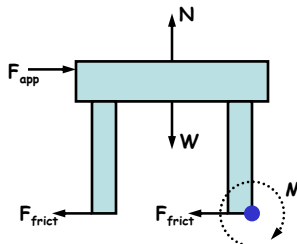
- This provides the desired **position** $s(t)$, as a function of time t .

Chapter 8, Slide 13
Copyright © David Mount and Amitabh Varshney

Rigid-Body Kinetics

How do we generalize this for **rigid bodies**?

- For rigid bodies we need to consider **rotational forces** as well.
- Calculate the **forces** acting on the object.
- Determine the total **resulting torques**.
- Use the (rotational) motion equation, $M(t) = I (d\omega/dt)$, to determine **angular velocity** as a function of time.
- Integrate to obtain **angular position**.



Chapter 8, Slide 14
Copyright © David Mount and Amitabh Varshney

Overview

- Torque
- Kinetics (for particles and rigid bodies)
- **Collision Detection and Response**
- Integration
- Motion Constraints

Chapter 8, Slide 15
Copyright © David Mount and Amitabh Varshney

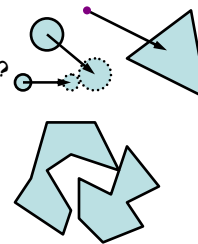
Collisions

Collisions:

- An important **source of forces** in games.
- Can be **expensive**, since if there are n objects, there are $O(n^2)$ **possible colliding pairs** to test.
- Requires **good data algorithms** to minimize calculations.

Collision Detection:

- **Geometric primitives:** (By solving linear or quadratic equations.)
 - When does a moving **point hit a plane**?
 - When does a moving **line hit another line**?
 - When does a moving **sphere hit another sphere**?
- Combine these for more **complex objects**:
 - When does a polyhedral object hit another polyhedral object?

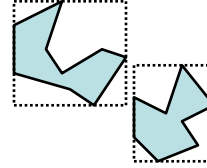


Chapter 8, Slide 16
Copyright © David Mount and Amitabh Varshney

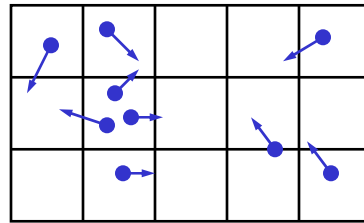
Collision Detection

Efficient Collision Detection:

Bounding volumes: To simplify calculations, first test whether bounding volumes (e.g. box or sphere) intersect. If so, then apply **more complex tests**.



Spatial Index: Use a spatial index (grid, quadtree, kd-tree, BSP-tree) to **reduce** the number of intersecting pairs that need to be tested.



Chapter 8, Slide 17
Copyright © David Mount and Amitabh Varshney

Collision Response: Kinetic Energy

Collision Response:

- How do objects **react** to a collision?
- Depends on **masses** and **velocities**.
- Also depends on the degree to which **kinetic energy** is conserved.

Kinetic Energy:

- Defined to be $\frac{1}{2}mv^2$. ←..... **We'll treat velocity as a scalar for now.**
- **Units:** $\text{kg m}^2/\text{s}^2 = 1 \text{ Newton-meter} = 1 \text{ joule (J)}$.
- A collision is said to be **elastic** if kinetic energy is **preserved**.
- Otherwise it is **inelastic** or **plastic**.

Coefficient of Restitution:

- Most collisions are **neither purely elastic or purely plastic**.
- **Coefficient of Restitution** (ϵ): Degree of elasticity in a collision. This is a property of the two **materials** involved.
- $\epsilon \cong 1$: Highly elastic (ping-pong balls). $\epsilon \cong 0$: Highly plastic (soft clay).

Chapter 8, Slide 18
Copyright © David Mount and Amitabh Varshney

Collision Response: Impulse Force

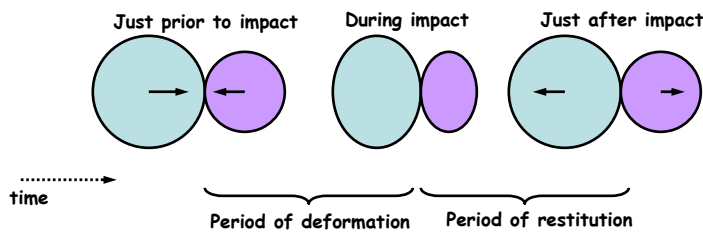
Impulse Force: A force that acts over a very short period of time.

Linear Impulse: Induces linear motion.

Torque Impulse: Induces rotational motion.

Impulse-Momentum Principle:

- When two objects collide, they are in contact for a **very short period**, and each exerts an **impulse** force on the other.
- During collision time interval the objects **deform**. This results in some **loss of kinetic energy**. Then they **restore** to prior shape.



Chapter 8, Slide 19
Copyright © David Mount and Amitabh Varshney

Collision Response: Linear Impulse

Response to Impulse:

- Recall: $F(t) = m (dv/dt)$, that is, $F(t) dt = m dv$.
- Let $[t^-, t^+]$ denote the **small time interval** in which colliding objects are in contact.
- **Integrating** over time:

$$\int_{t^-}^{t^+} F(t) dt = m \int_{t^-}^{t^+} dv = m(v^+ - v^-)$$

where v^- and v^+ are velocities at t^- and t^+ , respectively.

Linear Impulse (Λ): Defined to be the **integral of force** over the time interval.

$$\Lambda = \int_{t^-}^{t^+} F(t) dt$$

Equation of Linear Impulse: (Relates change in **momentum** to impulse force.)

$$m \cdot v^+ = m \cdot v^- + \Lambda$$

Chapter 8, Slide 20
Copyright © David Mount and Amitabh Varshney

Collision Response: Example

Example: (Particles: no rotation)

- Consider two **spherical bodies** of masses m_1 and m_2 , that collide with respective **velocities** v_1^- and v_2^- . What are the velocities v_1^+ and v_2^+ **after collision**?



- If we knew the **linear impulse** Λ , we could use the equations of linear impulse to solve for v_1^+ and v_2^+ .

$$m_1 \cdot v_1^+ = m_1 \cdot v_1^- + \Lambda$$

$$m_2 \cdot v_2^+ = m_2 \cdot v_2^- - \Lambda$$

Observe that due to **reaction**, we use $-\Lambda$ for object 2.

Chapter 8, Slide 21
Copyright © David Mount and Amitabh Varshney

Collision Response: Example

Example:

What's missing? The coefficient of restitution (ϵ).

Simple Definition: Velocities are measured along line of contact (n):

$$\epsilon = -\frac{v_1^+ - v_2^+}{v_1^- - v_2^-} \qquad \epsilon = -\frac{(v_1^+ - v_2^+) \cdot n}{(v_1^- - v_2^-) \cdot n}$$

General (vector) Form:

Intuition: $(v_1 - v_2)$ is the **relative velocity** of the two objects. This fraction indicates the degree to which the relative velocities are **preserved** after collision.



Chapter 8, Slide 22
Copyright © David Mount and Amitabh Varshney

Collision Response: Example

Putting it together: Combining these equations we have the following (vector) equations.

$$m_1 \cdot \mathbf{v}_1^+ = m_1 \cdot \mathbf{v}_1^- + \Lambda \quad (1)$$

$$m_2 \cdot \mathbf{v}_2^+ = m_2 \cdot \mathbf{v}_2^- - \Lambda \quad (2)$$

$$(\mathbf{v}_1^+ - \mathbf{v}_2^+) \cdot \mathbf{n} = -\varepsilon((\mathbf{v}_1^- - \mathbf{v}_2^-) \cdot \mathbf{n}) \quad (3)$$

Solving for impulse yields:

$$\Lambda = - \left(\frac{m_1 m_2 (1 + \varepsilon) ((\mathbf{v}_1^- - \mathbf{v}_2^-) \cdot \mathbf{n})}{m_1 + m_2} \right) \mathbf{n}$$

Vector dot product
Final impulse

Substituting Λ into (1) and (2) yields the final velocities \mathbf{v}_1^+ and \mathbf{v}_2^+ after collision.

What about Angular Impulse (Torque)? Handled analogously.

Chapter 8, Slide 23
Copyright © David Mount and Amitabh Varshney

Simple Motion Simulator

We have all the tools we need for a simple motion simulator.

- Assume motion **between** collision events can be computed **explicitly**.
- If this is not true we need to resort to **integration** (later).

Simple Motion Simulator: A single step of the simulation.

```

t ← getCurrentTime()
collidingPairs ← getCollidingPairs()
for each (p,q) in collidingPairs {
    calculate exact time tpq at which collision occurs
    calculate impulse force Λ // as in previous example
    p.veloc ← p.veloc + Λ / p.mass // by equation of linear impulse
    q.veloc ← q.veloc - Λ / q.mass
}
for each (object p)
    update p's position based on velocities and elapsed time
    
```

Chapter 8, Slide 24
Copyright © David Mount and Amitabh Varshney

Overview

- Torque
- Kinetics (for particles and rigid bodies)
- Collision Detection and Response
- **Integration**
- Motion Constraints

Chapter 8, Slide 25
Copyright © David Mount and Amitabh Varshney

Numerical Simulation

Symbolic Integration:

- The simple motion simulator described earlier works fine if we can solve the motion equations through **symbolic integration** or differentiation.
- This is often **not** the case, because motion is subject to **complex motion constraints**.

Numerical Integration:

- Approximating differential constraints in **small time steps**.
- Can deal with **complex** constraint systems.
- Tradeoffs between **physical accuracy** and **computational expense**.
- **Stability** can be an issue. **Vicious cycle**:
 - To increase **accuracy**, make time steps **smaller**.
 - Smaller time steps require **more total steps**.
 - Small per-step **errors accumulate**, reducing the overall accuracy.

Chapter 8, Slide 26
Copyright © David Mount and Amitabh Varshney

Euler Integration: Derivation

Euler Integration:

- Simplest and best known method of motion integration.
- Easy to implement.
- Not very accurate.

Derivation:

Taylor's Theorem: Consider a function $f(x)$, and derivatives f' , f'' , f''' , ..., $f^{(k)}$, which are assumed to be continuous. Then:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \dots + \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + O((x - x_0)^{k+1})$$

The last term is called the **remainder term** and represents the error due to **truncating** the series.

Chapter 8, Slide 27
Copyright © David Mount and Amitabh Varshney

Euler Integration: Derivation

Derivation (cont):

- To apply this to motion simulation, let:

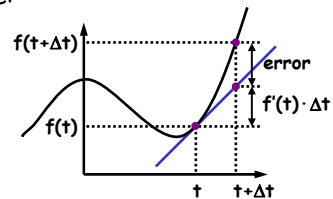
- $x_0 = t$,
- $x - x_0 = \Delta t$,
- $x = t + \Delta t$.

- We have:

$$f(t + \Delta t) = f(t) + f'(t)\Delta t + \frac{f''(t)}{2!}\Delta t^2 + \dots + \frac{f^{(k)}(t)}{k!}\Delta t^k + O(\Delta t^{k+1})$$

- Keeping just the **first two terms** we have:

$$f(t + \Delta t) = f(t) + f'(t)\Delta t + O(\Delta t^2)$$



Chapter 8, Slide 28
Copyright © David Mount and Amitabh Varshney

Euler Integration

Why is this useful?

- Given the value of a **function** at some time and its current **derivative**, we can **approximate** its value after a small time step.
- Error decreases **quadratically** with the time step.

Euler Integration:

- $s(t)$: **Position** at time t ,
- $v(t)$: **Velocity** at time t ,
- $a(t)$: **Acceleration** at time t
(computed from forces using $a(t) = F(t)/\text{mass}$).
- (These are all **vector quantities**, which we'll treat as scalars.)

- Update rules:

$$s(t + \Delta t) \cong s(t) + s'(t)\Delta t \cong s(t) + v(t)\Delta t$$

$$v(t + \Delta t) \cong v(t) + v'(t)\Delta t \cong v(t) + a(t)\Delta t$$

$$a(t + \Delta t) = F(t)/\text{mass}$$

Chapter 8, Slide 29
Copyright © David Mount and Amitabh Varshney

Euler Integration: Pseudocode

Initialization:

```
p.pos ← p.initialPosition( )  
p.veloc ← p.initialVelocity( )
```

Update: (Compute state at time $t+\Delta t$ from values at time t)

```
p.pos ← p.pos +  $\Delta t$ ·p.veloc  
p.veloc ← p.veloc +  $\Delta t$ ·p.force/p.mass  
p.force ← p.calculateSumOfForces( )
```

Advantages:

- Easy to implement.
- Very fast.

Disadvantages:

- Not very accurate:** Errors can accumulate rapidly.
- Inconsistency:** Velocities and positions can be inconsistent.
- Instability:** Can be unstable for stiff equations.

Chapter 8, Slide 30
Copyright © David Mount and Amitabh Varshney

Overview

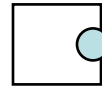
- Torque
- Kinetics (for particles and rigid bodies)
- Collision Detection and Response
- Integration
- **Motion Constraints**

Chapter 8, Slide 31
Copyright © David Mount and Amitabh Varshney

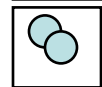
Constraints

Real motion is subject to numerous constraints:

- Objects should not intersect **obstacles** in the environment.



- Objects should not intersect **each other**.



- **Link constraints**: Two objects must be at a fixed distance.
(E.g., wrist to elbow joint).



- **Angle constraints**: The angle determined by three objects should be within a certain range.
(E.g., wrist-elbow-shoulder angle from 0 and 180 degrees.)



Chapter 8, Slide 32
Copyright © David Mount and Amitabh Varshney

Constraints

How can we maintain constraints within our integrator?

Penalty springs: Insert stiff springs that offer resistance when constraints are violated. (**Problem:** Possible instability.)

Zero-Tolerance Policy: Process events exactly at the time they occur. (**Problem:** Computationally expensive.)

Relaxation: When constraints are violated, move to a state where the violation is lessened. (A good compromise.)

Chapter 8, Slide 33
Copyright © David Mount and Amitabh Varshney

Constraints: Relaxation

Example 1: An object's x-coordinate must lie within [0,100].

- $x \leftarrow \text{updatePosition}()$
- if $(x < 0) x \leftarrow 0$; if $(x > 100) x \leftarrow 100$;

Example 2: Objects p_1 and p_2 must be at distance d of each other.

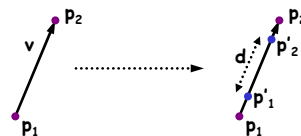
- $p_1 \leftarrow p_1.\text{updatePosition}()$
- $p_2 \leftarrow p_2.\text{updatePosition}()$
- $v \leftarrow p_2 - p_1$
- $d' \leftarrow \text{sqrt}(v \cdot v)$
- $\gamma \leftarrow (d' - d)/(2d')$
- $p'_1 \leftarrow p_1 - \gamma v$
- $p'_2 \leftarrow p_2 + \gamma v$

[vector from p_1 to p_2 .]

[length of vector v]

[correction factor]

[corrected position]



Chapter 8, Slide 34
Copyright © David Mount and Amitabh Varshney

Constraints: Relaxation

Example 3: Obstacle avoidance for the link object p_1p_2 .

Idea 1: **Translate** the segment along the shortest vector from the obstacle.

Idea 2: **Rotate** the segment about its farther endpoint to avoid the obstacle.

Idea 3: A **combination** of the above.

Example 4: Articulated body.

- Individual **joints** are modeled as particles.
- Connecting **bones** are modeled as distance constraints.
- Joint **angles** are modeled as 3-point angle constraints.

Rag-Doll Physics:

- Forces applied to any part of the body are transmitted by these constraints to the other joints.
- Result looks like a stick-figure puppet.

Chapter 8, Slide 35
Copyright © David Mount and Amitabh Varshney

Multiple Constraints

Multiple Constrained Objects:

- Fixing one constraint violation may create a **new** constraint violation.

Iterated local relaxation:

- Construct a **list of constraint violations**, involving small sets of objects (e.g. pairs).
- For each violation, compute a **minimal motion** that **resolves** the situation (and so satisfies the constraint).
- When done, go back and **repeat the process**. Repeat for some **fixed number** of iterations or until the system approaches a **stable configuration** (which may or may not be constraint-free.)

Advantages:

- Easy to implement. Often works.

Disadvantages:

- No guarantees.

Chapter 8, Slide 36
Copyright © David Mount and Amitabh Varshney

Summary

Summary:

- Force and Torque
- Kinetics and the laws of force and motion
- Collision detection and response
- Numerical integration
- Motion constraints

Chapter 8, Slide 37
Copyright © David Mount and Amitabh Varshney

Additional Slides

Chapter 8, Slide 38
Copyright © David Mount and Amitabh Varshney

Verlet Integration

Verlet Integration:

- Very **stable**.
- Does **not** require explicit computation of **velocity**, which is useful for objects that do not move continuously.
- **Widely used** (e.g., in molecular simulations).

Update Rule:

- Let $s(t)$ be **position** and let $a(t)$ be **acceleration** at time t .

$$s(t + \Delta t) = 2s(t) - s(t - \Delta t) + a(t)\Delta t^2$$

- **Note:** Unlike Euler integration, we don't need velocity, but we do need to store the **two recent positions**: $s(t)$ and $s(t - \Delta t)$.
- Velocity can be derived:

$$v(t) = \frac{s(t + \Delta t) - s(t - \Delta t)}{2\Delta t}$$

Chapter 8, Slide 39
Copyright © David Mount and Amitabh Varshney

Verlet Integration: Derivation

Verlet Integration Derivation:

- We start with a 2-term Taylor's expansion for $s(t + \Delta t)$ and $s(t - \Delta t)$:

$$s(t + \Delta t) = s(t) + \Delta t \cdot s'(t) + \Delta t^2 \frac{s''(t)}{2!} + \Delta t^3 \frac{s'''(t)}{3!} + O(\Delta t^4),$$

$$s(t - \Delta t) = s(t) - \Delta t \cdot s'(t) + \Delta t^2 \frac{s''(t)}{2!} - \Delta t^3 \frac{s'''(t)}{3!} + O(\Delta t^4).$$

- Summing and rearranging yields:

$$s(t + \Delta t) + s(t - \Delta t) = 2s(t) + \Delta t^2 s''(t) + O(\Delta t^4),$$

$$s(t + \Delta t) = 2s(t) - s(t - \Delta t) + \Delta t^2 s''(t) + O(\Delta t^4).$$

- The second derivative is acceleration. Final **Verlet update rule**:

$$s(t + \Delta t) = 2s(t) - s(t - \Delta t) + \Delta t^2 a(t) + O(\Delta t^4)$$

- **Note:** $O(\Delta t^4)$ error term is **much smaller** than Euler's $O(\Delta t^2)$.

Chapter 8, Slide 40
Copyright © David Mount and Amitabh Varshney

Verlet Integration: Pseudocode

Pseudocode:

- Let p.prev be state at time $t - \Delta t$.
- Let p.curr be state at time t .
- Let p.next be state at time $t + \Delta t$.

Initialization:

```
p.prev.pos ← p.positionAtTime( $t_0$ )  
p.curr.pos ← p.positionAtTime( $t_1$ )  
p.curr.force ← p.forcesAtTime( $t_1$ )
```

Update: (Compute values at time $t+\Delta t$ from values at time t , $t-\Delta t$.)

```
p.next.pos ←  $2 \cdot p.curr.pos - p.prev.pos + (\Delta t^2) \cdot p.curr.force / p.mass$   
p.next.force ← p.calculateSumOfForces( )  
p.prev ← p.curr  
p.curr ← p.next
```