CMSC 498M: Chapter 11
Game Physics

Reading:

- **Game Physics**, by David H. Eberly, 2004
- **Physics for Game Developers**, by David M. Bourg, 2002
- **Bullet Physics** (http://bulletphysics.org/)

Overview:

- Basic physical concepts
- Kinematics for particles and rigid bodies
- Kinetics
- Collisions and motion constraints
- The Bullet physics library

Overview

Basic physical concepts

Kinematics for particles

Kinematics for rigid bodies

Kinetics

Collisions

Motion Constraints

The Bullet Physics Library

Game Physics: Basic Concepts

Basic Issues:

- **Kinematics**: The study of motion (ignoring forces). How does acceleration affect velocity? How does velocity affect position?
  - **Particle**: A point-mass. Body rotation ignored
  - **Rigid body**: Rotation of the body needs to be considered
- **Force**: Objects change motion only when forces are applied
  - **Contact vs. field forces**: Hitting a baseball vs. gravity or magnetism
  - **Torque**: Force that induces rotation
  - **Environmental sources**: Friction, buoyancy, drag/lift
- **Kinetics**: (also called **Dynamics**) The effect of force on motion
- **Non-rigid Objects**:
  - **Joints and constraints**: Rag-doll physics, mass-spring systems
  - **Flexible objects**: Soft bodies, meshes, cloth, hair
- **Complex Motion**: Fluid dynamics, smoke, particle systems
- **Collisions**: Detection and response

Applications of Physics in Games

Flight simulators
Sports/Racing
Combat Simulation
many others…

Major League Baseball 2K6


EF2000


WRC4


LiveForSpeed


Age of Empires III

Game Physics: Kinetics

Kinetics: How do forces effect the motion of an object

Projectiles: (e.g., bullets, cannon balls)
- Body **rotation may be negligible** or totally ignored
- Effects due to **gravity, wind, air resistance**

Aircraft: (e.g., for flight simulators)
- **Full 3D motion** modeling
- Effects due to **lift, drag, turbulence**
- Control issues (how do ailerons, flaps, rudder affect motion)

Ships: (and floating objects)
- **Buoyancy and flotation**
- Motion resistance and **fluid dynamics**

Cars: (e.g. for racing games)
- Friction, **road resistance, breaking, skidding**, **drifting**
- Effects of road **banking**
- **Crashing** and **tumbling**

Game Physics: Mathematics

Units and measures:
- **Type-checking** for physicists
- **English or metric**? Important to keep these straight

3-dimensional geometry:
- **Basic representations**: Scalars, points, vectors, matrices, tensors (we won't discuss tensors)
- **3D geometric processing**:
  - Linear/affine transformations
  - Dot and cross product of vectors
  - Rotation and quaternions

Calculus: Although derivations involve understanding of calculus, most computations just use basic constructs

Differential calculus: finite differences

Integral calculus: finite summations

Differential equations: simulated using small time steps

Physical Quantities and Units

Basic Quantities:

Measure	English	Metric (SI)
Mass	Slug	Kilogram (kg)
Length	Foot (ft)	Meter (m)
Time	Second (s)	Second (s)

Examples of Other Quantities:

Measure	English	Metric (SI)
Force	Pound (lb)	Newton (N)
Pressure	lb/ft^2	N/m^2
Velocity	ft/s	m/s

Basics: Newton's Laws

Isaac Newton's Laws of Motion: (circa 1687)

- **Law I**: A body tends to **remain at rest** or continue to move in a **straight line at constant velocity**, unless it is acted upon by an external force (Inertia)
- **Law II**: The **acceleration** of a body is proportional to the resultant **force** acting on the body, and this acceleration is in the same direction as the force (F = ma)
- **Law III**: For every force acting on a body (**action**) there is an equal and oppositely directed reacting force (**reaction**)

Relevance: Most of game physics involves implementing **Newton's laws**

Rigid Body Physics

Rigid Bodies:

- **No moving parts**, no hinges, no flexibility
- Very **simple** to analyze/model
- Unlike a point masses, need to consider **rotation**
- Complex objects can often be modeled as **systems of rigid bodies**

Rigid Body Properties:

Mass: (scalar)

- the **amount of matter**
- the degree of **resistance to change in motion** (inertial mass)

Center of Mass (or gravity): (point/vector)

- **central point** about which rotations occur
- need not lie within the body (if the body is nonconvex)

(Mass) **Moment of Inertia**: (scalar)

- the **resistance to rotational motion** about a given axis (scalar form)

Overview

Basic physical concepts

Kinematics for particles

Kinematics for rigid bodies

Kinetics

Collisions

Motion Constraints

The Bullet Physics Library

Kinematics: Speed and Velocity

Speed and Velocity:

Average Speed: Let s denote the object's position and t denote time. Assuming motion along a line, speed is the change in **position** Δs over some **time interval** Δt:

$$v = \frac{\Delta s}{\Delta t}.$$

Units: Speed is often measured in feet per second (ft/s), miles per hour (mi/h), meters per second (m/s), etc

Instantaneous speed: If speed varies with time, we need to consider the limit for **differential** (infinitely small) **time intervals**:

$$v = \lim_{\Delta t \to 0} \frac{\Delta s}{\Delta t} = \frac{ds}{dt}.$$

Velocity: Is a **vector-valued** quantity, whose **magnitude is the speed** and whose direction indicates the **direction of motion**

Kinematics: Acceleration

Acceleration: Change in speed over time

Average Acceleration: Change in **speed** Δv over some **time** Δt

$$a = \frac{\Delta v}{\Delta t}$$

Instantaneous acceleration: If speed varies with time, we need to consider the limit for **differential** (infinitely small) time intervals:

$$a = \lim_{\Delta t \to 0} \frac{\Delta v}{\Delta t} = \frac{dv}{dt}$$

Units: Acceleration is measured in ft/s^2, mi/h^2, m/s^2, etc

Kinematics: Relationships

Relating Position, Velocity, and Acceleration:

Position and Velocity: By definition, v(t) = ds/dt. Suppose that an object moves from position s_0 to s_1 during the time period t_0 to t_1. We have:

$$ds = v(t)\,dt$$

$$\int_{s_0}^{s_1} ds = \int_{t_0}^{t_1} v(t)\,dt$$

$$\Delta s = s_1 - s_0 = \int_{t_0}^{t_1} v(t)\,dt$$

We sometimes drop the parameter t and just write "v" here.

Velocity and Acceleration: By similar argument we have:

$$\Delta v = v_1 - v_0 = \int_{t_0}^{t_1} a(t)\,dt$$

All three: We also have:

$$a = \frac{dv}{dt} = \frac{d^2s}{dt^2} \quad \text{and} \quad v(t)\,dv = a(t)\,ds$$

Overview

Basic physical concepts

Kinematics for particles

Kinematics for rigid bodies

Kinetics

Collisions

Motion Constraints

The Bullet Physics Library

Angular Velocity

Rigid Body Rotation:

- So far we have only discussed **translation**, which would be fine if all objects were treated as **particles** (point-mass)
- For a complete understanding, we must consider **rotation**
- Rotation occurs about:
  - the object's **center of mass** and
  - some **axis of rotation** (which may change depending on forces)

Plane Kinematics:

- All rotation occurs about a **fixed axis of rotation** in 3-space (i.e, on a plane orthogonal to that axis)
- Good enough for many $2\frac{1}{2}$-dimensional games (e.g. Mario Bros)

General Kinematics:

- 3D rotation: **Euler angles** and **quaternions**

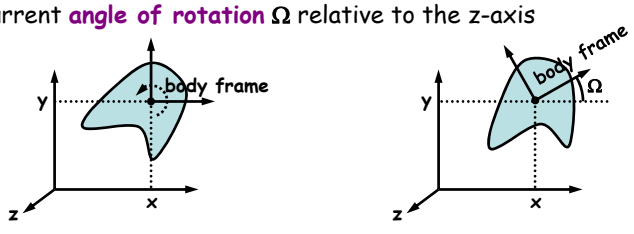
Plane Kinematics: Object State

Local Coordinate Frame:

- We select a coordinate frame so that **z-axis** is aligned with the **axis of rotation**

Object Orientation: To specify an object's location in space:

- Location of its **center of mass**: (x, y, z) coordinates in local frame.
- Current **angle of rotation** Ω relative to the z-axis



Plane Kinematics: Angular Velocity & Acceleration

Each of the quantities for translational motion has a **counterpart** in angular motion (m = meters, s = seconds, r = radians)

Translational Motion			Angular Motion		
Quantity	**Symbol**	**Dims.**	**Quantity**	**Symbol**	**Dims.**
Position	s	m	Angle of orientation	Ω	r
Velocity	v	m/s	Angular velocity	ω	r/s
Acceleration	a	m/s^2	Angular acceleration	α	r/s^2

Plane Kinematics: Angular Velocity & Acceleration

Analogs:

- The equations relating translational (linear) motion have **counterparts** for angular motion:

Translational motion	Angular motion
$v = ds/dt$	$\omega = d\Omega/dt$
$a = dv/dt = d^2s/dt^2$	$\alpha = d\omega/dt = d^2\Omega/dt^2$
$s = \int v\, dt$	$\Omega = \int \omega\, dt$
$v = \int a\, dt$	$\omega = \int \alpha\, dt$
$v\, dv = a\, ds$	$\omega\, d\omega = \alpha\, d\Omega$

Overview

Basic physical concepts

Kinematics for particles

Kinematics for rigid bodies

Kinetics

Collisions

Motion Constraints

The Bullet Physics Library

Kinetics (a.k.a. Dynamics)

Kinematics:

- The study of motion in the **absence of force**

Kinetics (**Dynamics**):

- How do we **integrate forces** with mass to determine **motion**?

Force Basics:

Contact force: from impact, friction, buoyancy, pressure

Field force: from gravity and electromagnetism

Newton's Third Law: Forces come in **pairs** (action and reaction). We will usually compute just one, and the other is its negation.

Examples:

Springs – useful for handling collisions

Friction – braking, skidding, sliding

Dampers – motion resistors

Bouyancy - floating

Kinetics (a.k.a. Dynamics)

Fundamental Equations of Motion:

Determined by **Newton's 2nd Law**

Force: $F = ma$

Torque: $M = I\alpha$

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

If force and torque vary over **time** these are given as **derivatives**:

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

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

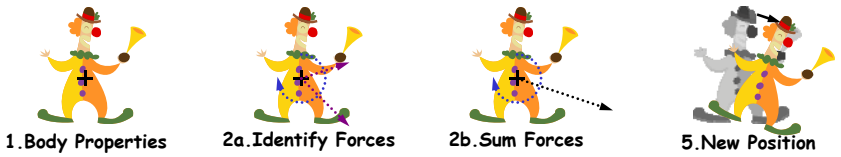
Kinetics for Games:

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

Kinetics (a.k.a. Dynamics)

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



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
}
```


Overview

Basic physical concepts

Kinematics for particles

Kinematics for rigid bodies

Kinetics

Collisions

Motion Constraints

The Bullet Physics Library

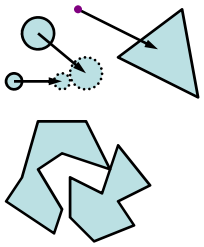
Collisions

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:

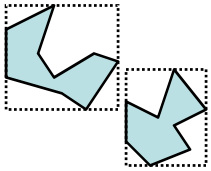
- **Geometric primitives**: (By solving linear or quadratic equations.)
  - When does a moving **point hit a plane**?
  - When does a moving **line hit another line**?
  - When does a moving **sphere hit another sphere**?
- Combine these for more **complex objects**:
  - When does a polyhedral object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



Collision Detection

Efficient Collision Detection:

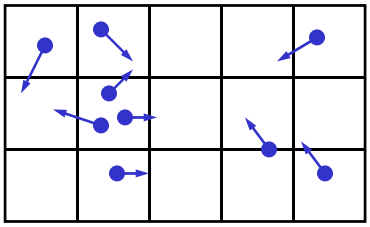
Bounding volumes: To simplify calculations, first test whether bounding volumes (e.g. box or sphere) intersect. If so, then apply **more complex tests**



Examples:

AABB: Axis-aligned bounding box

MEB: Minimum enclosing ball



Spatial Index: Use a spatial index to **reduce** the number of intersecting pairs that need to be tested. Examples:

- **square grid**
- **quadtree** (more adaptive)
- **BSP-tree** (even more adaptive, but more complex)

Collision Detection

Bounding-Volume Hierarchies:

- Good for **complex objects**
- Create a **tree** of enclosing shapes (e.g., AABBs)

Recursive Intersection Detection:

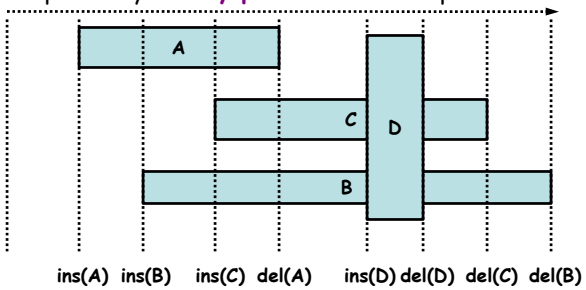
- Starting with the two tree roots, determine whether their **bounding boxes overlap**
- If they do, **recursively check their children** for overlaps
- Stop at the **leaf level**

```
intersect(u, v) {
    if (bounds(u) ∩ bounds(v) = null)      // no overlap - return
        return null
    if (u.isLeaf() and v.isLeaf())         // both leaves - found a pair
        return {u, v}
    if (u.isLeaf()) swap u and v;          // make sure u is not a leaf
                                           // recursively find overlapping pairs
    return intersect(u.leftChild(), v) ∪ intersect(u.rightChild(), v)
}
```


Collision Detection

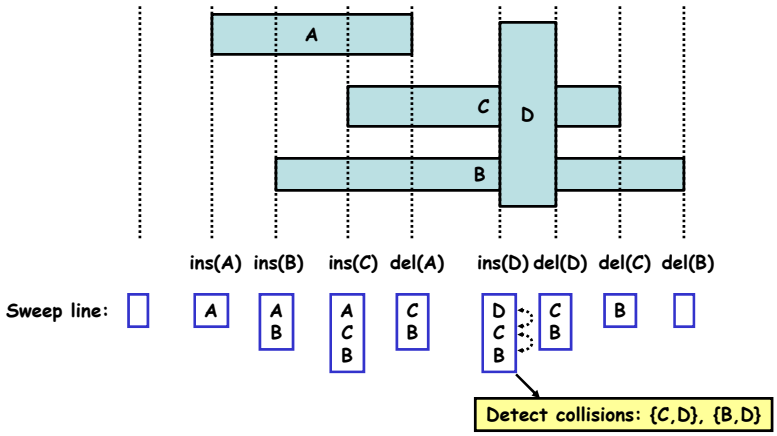
Plane Sweep Algorithm:

- Sweep the plane with a **vertical line**
- Maintain a data structure storing the objects that intersect the **sweep line** ordered from **top to bottom** based on their **top-most point**
- When new objects are **discovered**, add them to the **sweep line**
- Compare only **nearby pairs** on the sweep line for intersection



Collision Detection

Plane Sweep Algorithm:



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**: kg m^2/s^2 = 1 Newton-meter = 1 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
- $\varepsilon \cong 1$: Highly elastic (ping-pong balls). $\varepsilon \cong 0$: Highly plastic (soft clay)

Collision Response: Impulse Force

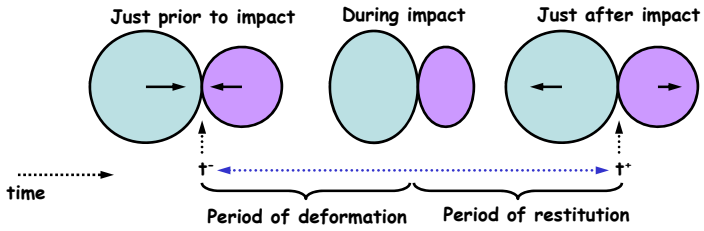
Impulse Force: A force that acts over a very short period of time.

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



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:

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

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

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

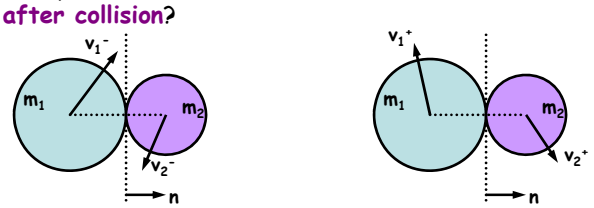
Equation of Linear Impulse: (Relates change in **momentum** to impulse force)

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

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

Collision Response: Example

Example:

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

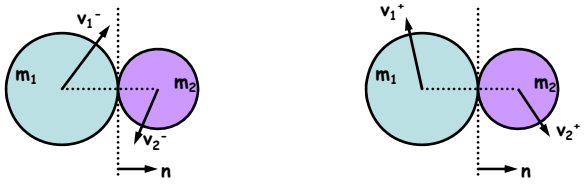
Simple Definition: Velocities are measured along line of contact (n):

$$\varepsilon = -\frac{v_1^+ - v_2^+}{v_1^- - v_2^-}$$

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

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.



Collision Response: Example

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

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

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

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

Solving for impulse yields:

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

Vector dot product

Final impulse

Substituting Λ into (1) and (2) yields the final velocities v_1^+ and v_2^+ after collision

What about Angular Impulse (Torque)? Handled analogously

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 t_pq 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
```

Overview

Basic physical concepts

Kinematics for particles

Kinematics for rigid bodies

Kinetics

Collisions

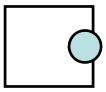
Motion Constraints

The Bullet Physics Library

Motion 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)

Motion 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)

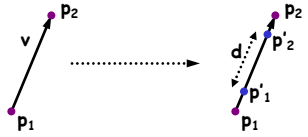
Constraints: Relaxation

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

- $x \leftarrow$ 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

- $p_1 \leftarrow p_1$.updatePosition( )
- $p_2 \leftarrow p_2$.updatePosition( )
- $v \leftarrow p_2 - p_1$ [vector from p_1 to p_2.]
- $d' \leftarrow \text{sqrt}(v \cdot v)$ [length of vector v]
- $\gamma \leftarrow (d' - d)/(2d')$ [correction factor]
- $p'_1 \leftarrow p_1 - \gamma v$ [corrected position]
- $p'_2 \leftarrow p_2 + \gamma v$



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**

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

Overview

Basic physical concepts

Kinematics for particles

Kinematics for rigid bodies

Kinetics

Collisions

Motion Constraints

The Bullet Physics Library

Bullet

Bullet Physics:

- Open-source **library** for **collision detection**, rigid and soft body **dynamics** (what we have called **kinetics**)

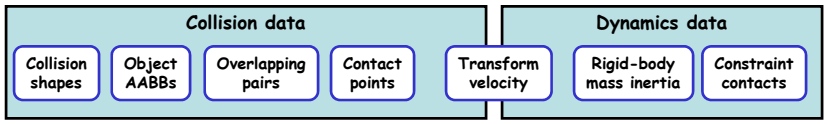
Features:

- Implemented in **C++**. Free for commercial use on **many platforms** including PLAYSTATION 3, XBox 360, Wii, PC, Linux, Mac OSX and iPhone
- Supports **collision detection**. Collision shapes include concave and convex meshes and all basic primitives
- Stable **rigid body dynamics** constraint solver including **vehicle** and **ragdoll** dynamics
- **Soft body dynamics** for cloth, rope and deformable shapes

Bullet Basic Data Structures

Basic Data Structures:

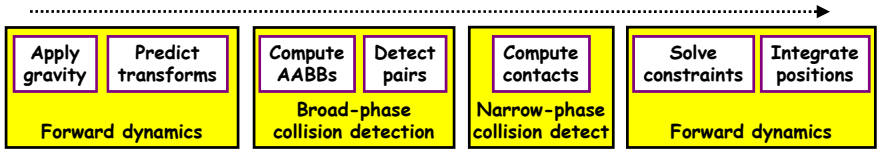
- **Collision Data**: Depend only on **object shape**, not on other physical properties, such as mass and velocity
  - **Collision shapes**: Describe object extents
  - **Axis-aligned Bounding Boxes** (**AABBs**): Provide simple approximations to collision shapes
  - **Overlapping pairs**: Pairs of AABBs that overlap
  - **Contact points**: Computed exact collision points
- **Dynamics Data**: Store **physical properties** such as mass, inertia, contact constraints, and joints.



Bullet Processing Pipeline

Bullet Physics Pipeline: Executed by invoking "stepSimulation"

- **Apply gravity**: Applies gravitational force to objects
- **Predict transforms**: Predict current positions of objects
- **Compute AABBs**: Computes bounding boxes
- **Detect pairs**: Determine which AABBs overlap (possible collisions)
- **Compute contacts**: Determine exact collisions
- **Solve constraints**: Perform collision response with motion constraints
- **Integrate positions**: Calculate new object positions and orientations



Bullet: Basic Data Types

Bullet provides basic **math/geometry data types**:

- **btScalar**: real-valued scalar
- **btVector3**: vector in 3-space (represented in homogeneous coordinates as a 4d vector)
- **btQuaternion**/**btMatrix3x3**: representing rotations/linear transformations in 3-space
- **btTransform**: an affine transformations (3d transformation plus rotation)

…and other **utilities**:

- Memory management
- Clock and performance profiling
- Debugging options

Bullet: Collision Detection

Fast collision detection:

- **btCollisionObject**: a **transformable object**
- **btCollisionShape**: describes the shape of a **collision object** (e.g., box, sphere, convex hull, triangle mesh)
- **btCollisionWorld**: stores **all collision objects** and provides an interface to perform **queries**

Broadphase collision detection: Efficient determination of pairs of **possibly intersecting pairs** based on axis-aligned bounding box (AABB) overlap:

- **btDbvtBroadphase**: **bounding volume hierarchy** based on AABB trees
- **btAxisSweep3** and **bt32BitAxisSweep3**: **3D sweep and prune** approach
- **btCudaBroadphase**: fast **uniform grid** using **GPU** graphics hardware for efficiency

Bullet: Further Topics

Further topics (to be added):

- **Rigid-body Dynamics**: Specifying how objects **respond** to **forces** and **collisions**
- **Motion Constraints**: Specifying **constraints** on the motion of two or more objects (e.g., due to sharing a common joint)
- **Soft-body Dynamics**: Specifying how **flexible bodies** (cloth, e.g.) behave

Summary

Summary:

- Basic physical concepts
- Kinematics for particles and rigid bodies
- Kinetics (Dynamics)
- Collisions and motion constraints
- Bullet physics library