**Overview**

**Collision Detection**
- Broad- vs. Narrow-phase

**Collision Handling & Physics**

**Physics Engines**
- Rigid Bodies and Joints
- Collision Spaces

**Integrating Physics & Game Engines**

---

**Naïve Collision Detection**

```java
void checkCollisions()
{
    Iterator iter1 = theWorld.iterator();
    while (iter1.hasNext())
    {
        //get a world object
        ICollider curObj = (ICollider) iter1.next();
        Iterator iter2 = theWorld.iterator();
        while (iter2.hasNext())
        {
            //get a second object
            ICollider otherObj = (ICollider) iter2.next();
            //insure it's not the SAME object
            if (otherObj != curObj)
            {
                //check for collision and handle it
                if (curObj.collidesWith(otherObj))
                {
                    curObj.handleCollision(otherObj);
                }
            }
        }
    }
}
```

---

**Two-Phase Collision Detection**

**Broad-phase:**

*select pairs of objects that *might* have collided*

**Goals:**
- Try to avoid selecting pairs that can't collide
- Try to make these sections quickly

**Narrow-phase:**

*compare those pairs to see if they actually collided*

**Goals:**
- Accurate comparisons
- Efficient comparisons

---

**Broad-phase (step 1)**

"Sweep And Prune" (SAP) Algorithm

```
Sweep And Prune (cont.)

Identifying "interference" (potential collisions):
- Prune (skip) non-interfering objects
```

---

**Sweep And Prune (cont.)**

Identifying “interference” (potential collisions):
- *Prune* (skip) non-interfering objects

---

---

---

---
For hierarchical objects, using Bounding Volumes:

```java
public class SceneNode {
    ... public boolean collidesWith(SceneNode otherNode) {
        if (!this.getWorldBound().intersects(otherNode.getWorldBound())) {
            return false;  // world BV's don't intersect
        } else if (this.hasChildren()) {
            for (each child of this node) {
                if (child.intersects(otherNode)) {
                    return true;  // found a child that intersects
                }
            }
            return false;  // no child intersects
        } else if (otherNode.hasChildren()) {
            for (each child of otherNode) {
                if (child.collidesWith(this)) {
                    return true;
                }
            }
            return false;
        }
    }
}
```

Collision Handling

(Some) factors to consider
- Position
- Orientation
- Linear velocity (change in position)
- Angular velocity (change in orientation)
- Friction
- Air lift/drag
- Water resistance/buoyancy
- Gravity
- Elasticity

Using Physics In Games

Ad-hoc solutions:
- Constant velocity
  `newPos = curPos + velocity * elapsedTime;`
- Gravity
  `newPosY = curPosY + velocityY * elapsedTime;`
- Springs
  `k = 0.2; // spring constant
  springVector = mass.pos - connectionPos;
  mass.pos = mass.applyForce(-springVector*k);`

(Ordinary) Differential Equations

- Equations defining a relationship between a single-variable function \( x(t) \) and its derivatives
  \[
  \frac{dx(t)}{dt}, \quad \frac{d^2x(t)}{dt^2}, \quad \cdots
  \]
- ex.: Newton’s Second Law of motion
  \[
  F = ma; \quad F(x(t)) = m \frac{d^2x(t)}{dt^2}
  \]
  Force on particle at position \( X \) at time \( t \)
  Acceleration (2nd derivative of position)

Physics Engines

Integrate laws of physics into a game
- Laws are represented by ODE’s
- Physics Engines contain “solvers”

Independent of “gameplay”
- Analogous to how a Game Engine supplies independent “renderers”
- Usable for a wide number of genres:
  - Race simulation
  - FPS action-shooter
  - Virtual world dynamic structures
  - Space travel, planets, etc.
  - Sports

Common Physics Engines

- Havok (www.havok.com)
- Newton (www.newtondynamics.com)
- PhysX (www.nvidia.com/physX)
  - jPhysX (www.jphysX.com)
- Bullet (www.bulletphysics.org)
  - jBullet (jbullet.advel.cz)
- Open Dynamics Engine (ODE) (www.ode.org)
  - ODEJava (odejava.dev.java.net)
Physics Engine Classes

World ("Physics Space")
A container for Bodies, Constraints, CollisionShapes, CollisionHandlers, and Solvers

Body
Represents a single rigid or soft body.
Most common is rigid:
Fixed attributes: Mass, Mass distribution
Dynamic Attributes: Position, Orientation, Velocity, Angular velocity

Constraint ("joint")
Represents a connection between two bodies

Physics Engine Interface

```java
/** Sage interface */
public interface IPHysicsObject {
    public void setTransform(double[] transform);
    public double[] getTransform();
    public float getFriction();
    public void setFriction(float friction);
    public float getLinearDamping();
    public float getAngularDamping();
    public void setDamping(float linearDamping, float angularDamping);
    public float getBounciness();
    public void setBounciness(float value);
    public float[] getLinearVelocity();
    public void setLinearVelocity(float[] velocity);
    public float[] getAngularVelocity();
    public void setAngularVelocity(float[] velocity);
    public float[] getLinearAcceleration();
    public void applyLinearForce(float fx, float fy, float fz);
    public void applyLinearTorque(float fx, float fy, float fz);
    public boolean isDynamic();
}
```

Physics Object Interface

```java
/** Define the interface implemented by all Physics Objects */
public interface IPhysicsObject {
    public void init();
    public void setGravity(float[] gravity_vector);
    public void setGravity(int gravity); // sets gravity explicitly
    public void setFriction(float friction);
    public void setDamping(float linearDamping, float angularDamping);
    public void setBounciness(float bounciness);
    public void applyForce(float fx, float fy, float fz);
    public void applyTorque(float fx, float fy, float fz);
    public void applyForce(float fx, float fy, float fz, float px, float py, float pz);
    public void applyForce(float fx, float fy, float fz, float px, float py, float pz, float ps);
    public void applyForce(float fx, float fy, float fz, float px, float py, float pz, float ps, float pt);
    public void applyTorque(float fx, float fy, float fz, float px, float py, float pz, float ps);
    public void applyTorque(float fx, float fy, float fz, float px, float py, float pz, float ps, float pt);
    public void applyTorque(float fx, float fy, float fz, float px, float py, float pz, float ps, float pt, float pu);
}
```