Before we Start - Terms

- **Kinematics** – motion, don’t care how it happens
- **Dynamics** – motion, do care how it happens – forces
- **Inverse kinematics** – what joint angles will get me where I want (pose, or finger location)
- **Forward kinematics** – given joint angles where will my fingers be
- **Inverse dynamics** – how much force (torque) must be applied to get the movement we want (i.e. walking)
- **Forward dynamics** – Given forces, what happens over time.
What Kind of Objects?

- Want to be able to easily find collisions (intersections)
  - Spheres
What Kind of Objects?

- Want to be able to easily find collisions (intersections)
  - Spheres
    - Is $r_1 + r_2 < d$?
    - Squared?
What Kind of Objects?

- Want to be able to easily find collisions (intersections)
  - Spheres
  - Convex
What Kind of Objects?

- Want to be able to easily find collisions (intersections)
  - Spheres
  - Convex polygonal
    - Pt-line
    - Pt-pt
    - Line-line
  - Costly, but some tricks.
  - Pt inside poly simple
What Kind of Objects?

- Want to be able to easily find collisions (intersections)
  - Spheres
  - Convex
  - Concave
What Kind of Objects?

- Want to be able to easily find collisions (intersections)
  - Spheres
  - Convex
  - Concave
    - Very expensive
    - Difficult
      - Pt inside poly not trivial
    - Some tricks
      - Use convex-hull first
Problems

- In our physical world things collide, if they don’t break apart, we have a bounce
  - Newton’s 1st and 3rd laws?
- So worrying only about collisions is easy, don’t need point inside test, not penetration right?
Problems

- In our physical world things collide, if they don’t break apart, we have a bounce
  - Newton’s 1\textsuperscript{st} and 3\textsuperscript{rd} laws?
- So worrying only about collisions is easy, don’t need point inside test, not penetration right?
Problems

- In our physical world things collide, if they don’t break apart, we have a bounce
  - Newton’s 1st and 3rd laws?
- So worrying only about collisions is easy, don’t need point inside test, not penetration right?
Problems

- In our physical world things collide, if they don’t break apart, we have a bounce
  - Newton’s 1st and 3rd laws?
- So worrying only about collisions is easy, don’t need point inside test, not penetration right?
- We can lower \(dt\). Will this work?
Problems

- In our physical world things collide, if they don’t break apart, we have a bounce
  - Newton’s 1st and 3rd laws?
- So worrying only about collisions is easy, don’t need point inside test, not penetration right?
- We can lower dt. Will this work?
- We can detect intersection, then backup simulation to collision, then determine bounce.
  - Easy with one ball, difficult with many
Besides Collisions?

- What happens here?
- What does it depend on?
Besides Collisions?

- Friction.
- Rotation
  - Angular velocity
- Momentum
Example from Ch 14 on Physics

- Lot’s of formulas for physics
- Create a simple demo that will allow objects to move
  - Forward dynamics
- Simple collisions
  - Cause forces to changes
Formulas

- **Velocity** \( v = \Delta s/\Delta t \)
- **Acceleration** \( a = \Delta v/\Delta t \)
- **Newton’s 2\text{nd} law** \( F = ma \)
- **Momentum** \( p = mv \)
- **Impulse** \( I = F\Delta t = m\Delta v = \Delta p \)
- **Conservation of momentum** \( \Delta p_1 = -\Delta p_2 \)
- **Kinetic Energy** \( E = \frac{1}{2} m v^2 \)
- **Restitution coefficient** \( e = (v_{2f} - v_{1f})/(v_1 - v_2) = \sqrt{(h/H)} \)
- **Conservation of kinetic energy**
Using conservation of momentum and kinetic energy

- \( e = (v_{2f} - v_{1f})/(v_1 - v_2) = \sqrt{(h/H)} \)
- \( E = \frac{1}{2} m v^2 \) must be the same for initial and final velocities
- The total momentum can’t change after a collision.

- \( v_{1f} = ((e+1)m_2 v_2 + v_1(m_1 - e m_2))/(m_1 + m_2) \)
- \( v_{2f} = ((e+1)m_1 v_1 + v_2(m_1 - e m_2))/(m_1 + m_2) \)
class PhysicalObject
{
    public Vector3 Position;
    public Vector3 Velocity;
    public float Mass;
    public float Radius;
    public Matrix World;
    public Color Color;
}
Declarations

private Model sphere;
private PhysicalObject[] spheres = new PhysicalObject[5];
private Vector3 friction;
private float e; //coefficient of restitution

private InputHandler input;
private FirstPersonCamera camera;
public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";

    input = new InputHandler(this);
    Components.Add(input);
    camera = new FirstPersonCamera(this);
    Components.Add(camera);
}

Initialize()

protected override void Initialize()
{
    for (int i = 0; i < spheres.Length; i++)
    {
        spheres[i] = new PhysicalObject();
        InitializeValues();
        base.Initialize();
    }
}
InitializeValues()

private void InitializeValues()
{
    e = 0.95f;
    friction = new Vector3(-0.025f);

    if (spheres.Length < 2)
        throw (new ApplicationException("Must have at least 2 objects to do a collision!")
        );

    //we can setup our positions and velocity in 3D values as well ...
    //with some careful plotting we can get some nice results
    /*
    spheres[0].Position = new Vector3(-25.0f, 5.0f, -430);
    spheres[0].Velocity = new Vector3(50.0f, -8.0f, 10);
    spheres[0].Mass = 6.0f;
    spheres[0].Color = Color.Silver;
    */
InitializeValues()

spheres[0].Position = new Vector3(-90.0f, 0, -400.0f);
spheres[0].Velocity = new Vector3(160.0f, 0, 0);
spheres[0].Mass = 1.0f;
spheres[0].Color = Color.Silver;

for (int i = 1; i < spheres.Length; i++) {
    spheres[i].Position = new Vector3(25.0f + (i * 25), 0, -400);
    spheres[i].Velocity = new Vector3(-5.0f, 0, 0);
    spheres[i].Mass = 4.0f;
    spheres[i].Color = Color.Red;
}
spheres[spheres.Length - 1].Mass = 6.0f;
protected override void Update(GameTime gameTime)
{
        this.Exit();
    float elapsedTime = (float)gameTime.ElapsedGameTime.TotalSeconds;
    if (input.KeyboardState.WasKeyPressed(Keys.Enter) ||
        input.ButtonHandler.WasButtonPressed(0, InputHandler.ButtonType.Start))
        InitializeValues();
    for (int i = 0; i < spheres.Length; i++)
    {
        spheres[i].World = Matrix.CreateScale(spheres[i].Mass) *
            Matrix.CreateTranslation(spheres[i].Position);
        Vector3 trans, scale;
        Matrix rot;
        MatrixDecompose(spheres[i].World, out trans, out scale, out rot);
        spheres[i].Radius = scale.Length();
        ApplyFriction(ref spheres[i].Velocity);
    }
}
Update()

for (int a = 0; a < spheres.Length; a++)
    {
        for (int b = a + 1; b < spheres.Length; b++)
            {
                if (a == b) continue; //don't check against yourself
                //float distance = (spheres[a].Position - spheres[b].Position).Length();
                float distance = Vector3.DistanceSquared(spheres[a].Position, spheres[b].Position);
                //if (distance < collisionDistance * collisionDistance)
                float tmp = 1.0f / (spheres[a].Mass + spheres[b].Mass);
                float collisionDistance = distance - ((spheres[a].Radius + spheres[b].Radius) * (spheres[a].Radius + spheres[b].Radius));
                if (collisionDistance <= 0)
                    {
                        Vector3 velocity1 = ((e + 1.0f) * spheres[b].Mass * spheres[b].Velocity + spheres[a].Velocity * (spheres[a].Mass - (e * spheres[b].Mass))) * tmp;
                        Vector3 velocity2 = ((e + 1.0f) * spheres[a].Mass * spheres[a].Velocity + spheres[b].Velocity * (spheres[b].Mass - (e * spheres[a].Mass))) * tmp;
                        spheres[a].Velocity = velocity1;
                        spheres[b].Velocity = velocity2;
                    }
                spheres[a].Position = spheres[a].Position + (elapsedTime * (spheres[a].Velocity));
            }
    }
base.Update(gameTime);
protected override void Draw(GameTime gameTime) {
    GraphicsDevice.Clear(Color.CornflowerBlue);

    for (int i = 0; i < spheres.Length; i++)
        DrawModel(ref sphere, ref spheres[i].World, spheres[i].Color);

    // TODO: Add your drawing code here

    base.Draw(gameTime);
}
private void DrawModel(ref Model m, ref Matrix world, Color color) {
    Matrix[] transforms = new Matrix[m.Bones.Count];
    m.CopyAbsoluteBoneTransformsTo(transforms);

    foreach (ModelMesh mesh in m.Meshes) {
        foreach (BasicEffect be in mesh.Effects) {
            be.EnableDefaultLighting();
            be.PreferPerPixelLighting = true;

            be.AmbientLightColor = color.ToVector3();
            be.Projection = camera.Projection;
            be.View = camera.View;
            be.World = world * mesh.ParentBone.Transform;
        }
        mesh.Draw();
    }
}
private void ApplyFriction(ref Vector3 velocity)   {
  if (velocity.X < 0)
    velocity.X -= friction.X;
  if (velocity.X > 0)
    velocity.X += friction.X;

  if (velocity.Y < 0)
    velocity.Y -= friction.Y;
  if (velocity.Y > 0)
    velocity.Y += friction.Y;

  if (velocity.Z < 0)
    velocity.Z -= friction.Z;
  if (velocity.Z > 0)
    velocity.Z += friction.Z;
}
Collision Demo