# Real-Time Physics Simulation

Alan Hazelden

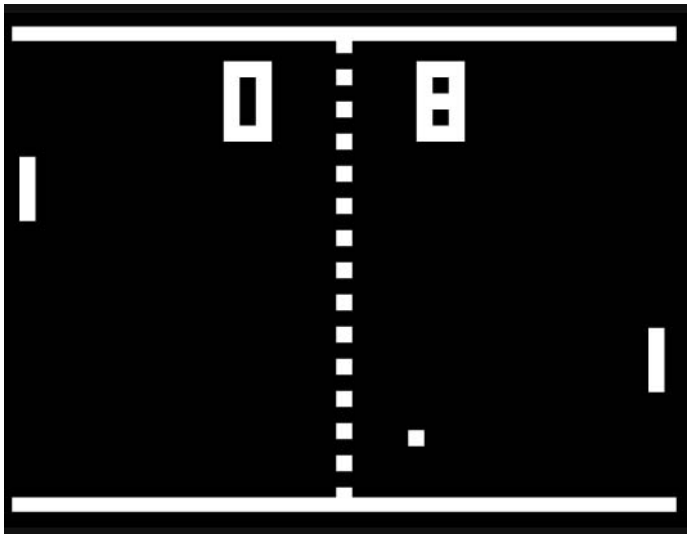
alan@draknek.org
http://www.draknek.org/

# Who am I?

- Studied Computer Science 2005-2009
  - 3rd and 4th year projects: physics engines
- Warwick Game Design exec member
  - Participated in many 48 hour competitions
- Probably the best C++ programmer here

# Physics in games

- Which games use physics?




# Physics in games

- Which games use physics?
  - Almost all of them!
  - Almost always need non-penetration
  - Almost always need collision detection
  - Almost always need collision resolution
- A physics engine provides all these
  - To some approximation of reality
  - But you may not want reality

# What is a physics engine?

- Simulates movement of objects
  - Position; orientation
  - Velocity; rotational velocity
- Models constraints between objects
  - Most common: non-penetration
  - Also: joints, friction, springs, buoyancy
- Here's one I made earlier...

# Accuracy vs. efficiency

- True physics is computationally ridiculous
- So we simplify things
  - Move objects and then resolve problems
  - Simplify collision geometry
  - ”Sleep” non-moving objects
- If we can fake something, we probably should

# Structure of a physics engine

- Broadphase
  - Determines which objects could potentially be colliding
- Generate contacts
  - Performs collision detection and finds contacts
- Resolve contacts
  - Find new (valid) positions for all objects

# Broadphase (collision culling)

- Brute force collision testing would take $O(n^2)$ comparisons
- We can rule some collisions out very quickly
  - Bounding boxes
  - Exploiting spacial coherence
  - Exploiting temporal coherence

# Broadphase (collision culling)

- Many implementations:
  - Bounding boxes for all pairs
  - Regular grid
  - Quadtree/Octree
  - BSP tree (binary space partitioning)
  - Hierarchy of bounding shapes
  - Sort and sweep algorithm

# Collision detection

- Bad collision detection means bad physics
- Different levels of collision detection:
  - Intersection
    - Are these two shapes touching?
  - Collision
    - If these two shapes touch, tell me how and where
  - Temporal collision
    - Tell me how, where and also when

# Contact generation

- Information generally needed:
  - Contact point
  - Contact normal
  - Amount of penetration
- For convex shapes, I use separating axis
- Concave shapes more complicated

# Collision resolution

- Resolve contacts so there is no penetration
  - In a physically realistic manner!
- Solving one contact may make another worse
- Need to find new positions and velocities
  - For all objects involved

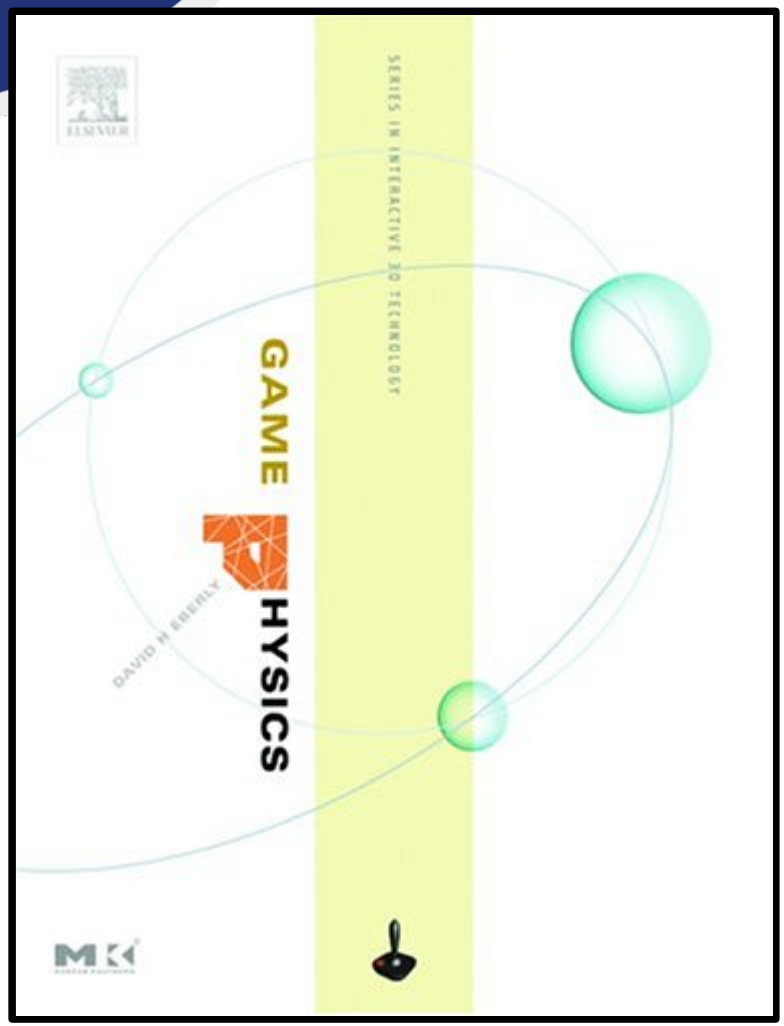
# Creating a physics engine

- Do you hate yourself?
- Do you have several years of your life to spare?
- Requirements:
  - Excellent maths skills
  - Excellent programming skills
  - Excellent patience
- Incredibly rewarding
  - Eventually

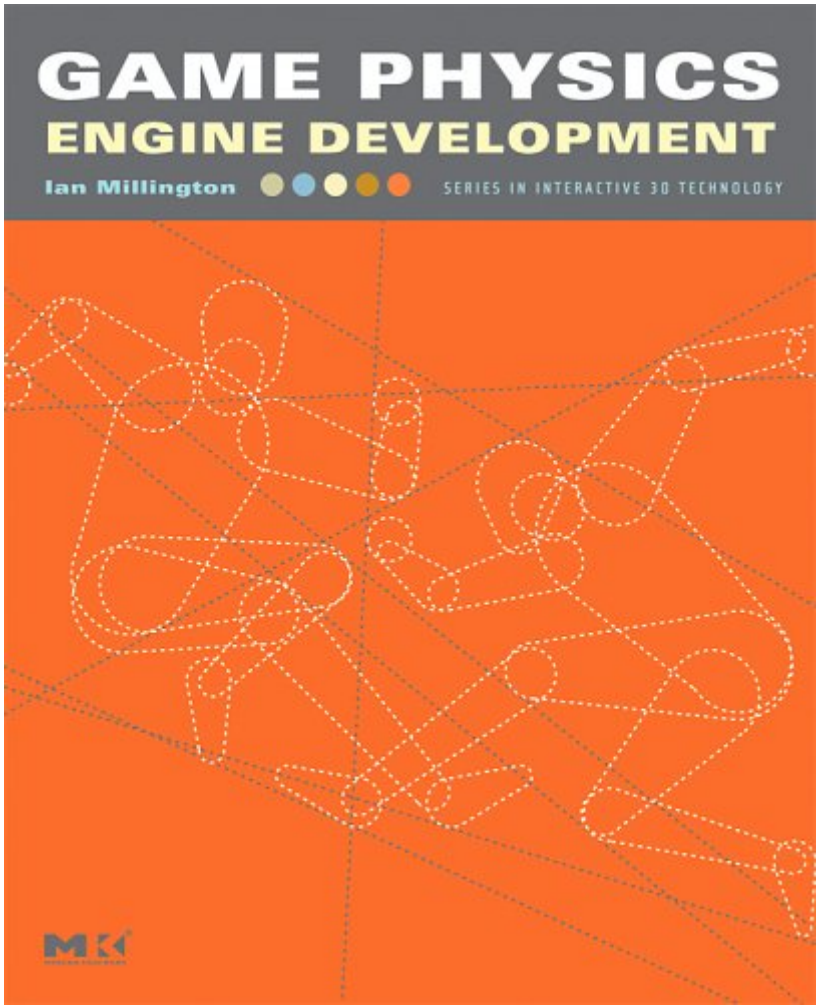
# References


*Real-Time Collision Detection*
Christer Ericson

# References




*Game Physics*
David Eberly




*Game Physics Engine Development*
Ian Millington

# Online resources

- Erin Catto
  - http://www.gphysics.com/
    - Box2D Lite: http://box2d.org/
- Glenn Fiedler
  - http://www.gaffer.org/game-physics
- Wikipedia

# 2D physics engines

- Box2D
  - http://www.box2d.org/
- Chipmunk
  - http://wiki.slembcke.net/main/published/Chipmur
- Farseer
  - http://www.codeplex.com/FarseerPhysics
- Large Polygon Collider
  - http://www.draknek.org/physics/

# 3D physics engines

- Bullet
  - http://www.bulletphysics.com/
- Open Dynamics Engine
  - http://www.ode.org/
- Havok
  - http://www.havok.com/tryhavok
- Large Polygon Collider
  - http://www.draknek.org/physics/ (awful)

# Using a physics engine

- Create a world
- Add bodies to world
- Add shapes to bodies
- Add constraints between bodies

- Apply an impulse to a body to move it

# Using a physics engine

- Static bodies never move
  - Use for level geometry
- Some bodies should not rotate
  - Possibly player should always be upright
- Shouldn't add/remove bodies in a callback
- In general, the player should be roughly one unit tall

# Using Box2D: creating a world

```
b2AABB worldAABB;
worldAABB.lowerBound.Set(-100.0f, -100.0f);
worldAABB.upperBound.Set(100.0f, 100.0f);
b2Vec2 gravity(0.0f, -10.0f);
bool doSleep = true;
b2World world(worldAABB, gravity, doSleep);
```

# Using Box2D: static bodies

```
b2BodyDef groundBodyDef;
groundBodyDef.position.Set(0.0f, -10.0f);
b2Body* ground =
  world.CreateBody(&groundBodyDef);


b2PolygonDef groundShapeDef;
groundShapeDef.SetAsBox(50.0f, 10.0f);
groundBody->CreateShape(&groundShapeDef);
```

# Using Box2D: non-static bodies

```
b2BodyDef bodyDef;
bodyDef.position.Set(0.0f, 4.0f);
b2Body* body = world.CreateBody(&bodyDef);

b2PolygonDef shapeDef;
shapeDef.SetAsBox(1.0f, 1.0f);
shapeDef.density = 1.0f;
shapeDef.friction = 0.3f;
body->CreateShape(&shapeDef);
body->SetMassFromShapes();
```

# Using Box2D: game loop

```
float32 timeStep = 1.0f / 60.0f;
int32 iterations = 10;

while (true)
{
    update();
    world.Step(timeStep, iterations);
    draw();
}
```

# Using Box2D: moving bodies

```
void update ()
{
    b2Vec2 p = body->GetPosition();
    if (Input::left) {
        body->ApplyImpulse(b2Vec2(-1, 0), p);
    }
    If (Input::right) {
        body->ApplyImpulse(b2Vec2(1, 0), p);
    }
}
```

# Using Box2D: drawing bodies

```
void draw ()
{
    b2Vec2 position = body->GetPosition();
    float32 angle = body->GetAngle();
    drawBox(position, angle);
}
```

# Questions?

<Rapturous applause>