Anatomy of a physics engine

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

Who am I?

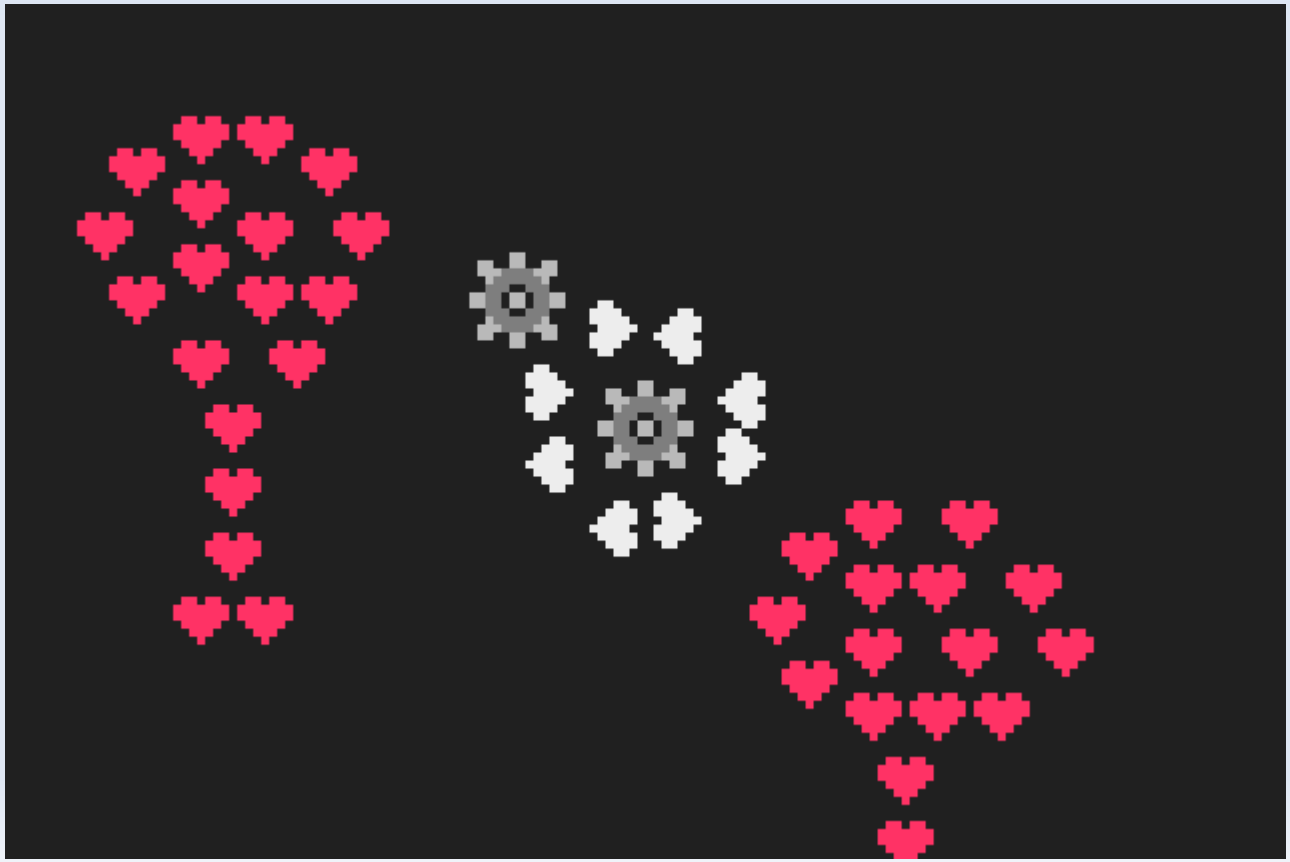
- Studied Computer Science 2005-2009
  - 3rd and 4th year projects: physics engines
- Making games since 2006
  - Joined Warwick Game Design
- Now:
  - Part-time game developer
  - Part-time web developer
- More Alan facts at www.draknek.org

My games

- These Robotic Hearts of Mine

  Narrative-based puzzle game

  http://www.draknek.org/games/hearts/

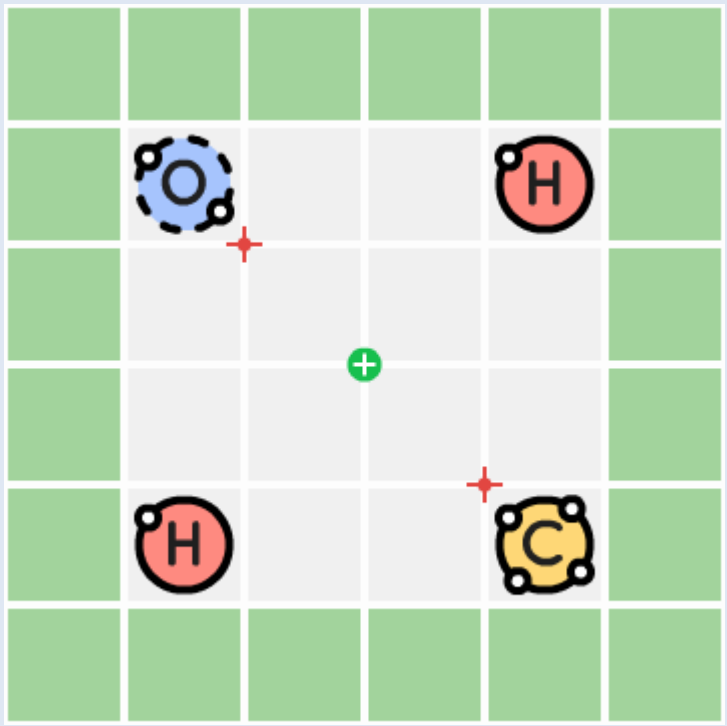


My games

- Sokobond

  Chemistry-themed puzzle game

  http://www.sokobond.com/



Also

- Current maintainer for FlashPunk

  Popular open-source flash game engine

  http://www.flashpunk.net/

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...

Demo

(A demo is worth a thousand pictures)

Large Polygon Collider
4th year group project 2008-2009

http://lpc.draknek.org/

What's the point?

- Games
  - 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 or may not want reality

How does this relate to graphics?

- Same areas of maths
  - Vectors
  - Matrices
- Some shared algorithms
  - Collision pruning/visibility culling
  - Point-in-polygon test
- Interactive technology
  - Real-time requirements
  - Always needs to be faster

Accuracy vs. efficiency

- True physics is computationally ridiculous
- We want plausibility not accuracy
- So for a real-time system we simplify things
  - Move objects and then resolve problems
  - Simplify collision geometry
  - “Sleep” non-moving objects
- If we can fake something, we probably should

Two types of physics engine

- Mass-aggregate systems
  - Everything is a particle
  - Soft-body physics
  - Fluid simulation
  - Good for GPUs
- Rigid body simulators
  - Everything has position **and** orientation
  - Good for solid objects

Structure of a physics engine

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

1. 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

2. 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

3. Collision Resolution

- Given the collision information:
  - Find the new state of all bodies
  - New velocities
  - New positions

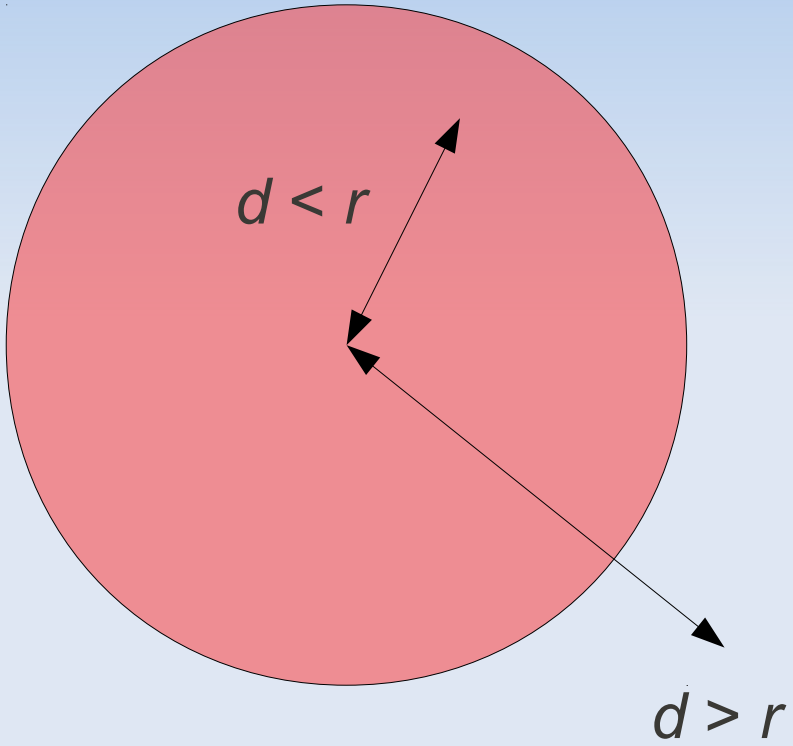
- In a particle system (no rotation), simple:
  - Conservation of momentum
  - Coefficient of restitution

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

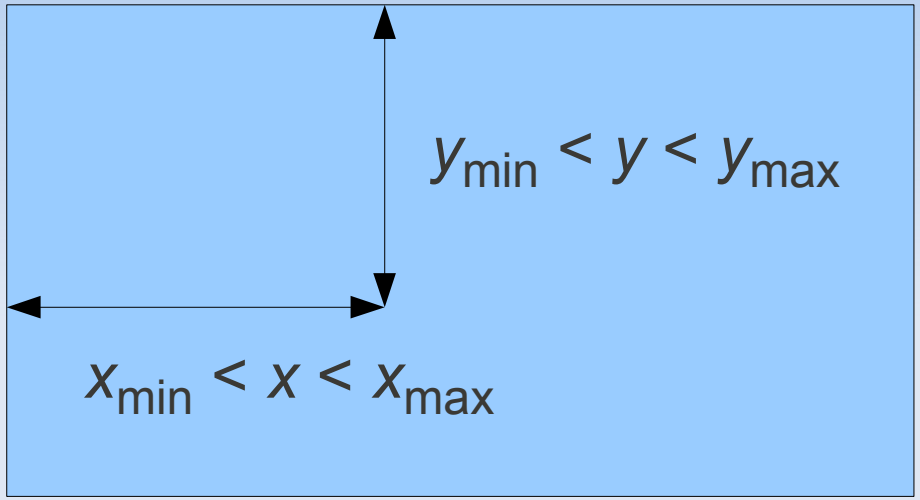
Intersection tests

- **Point in circle**



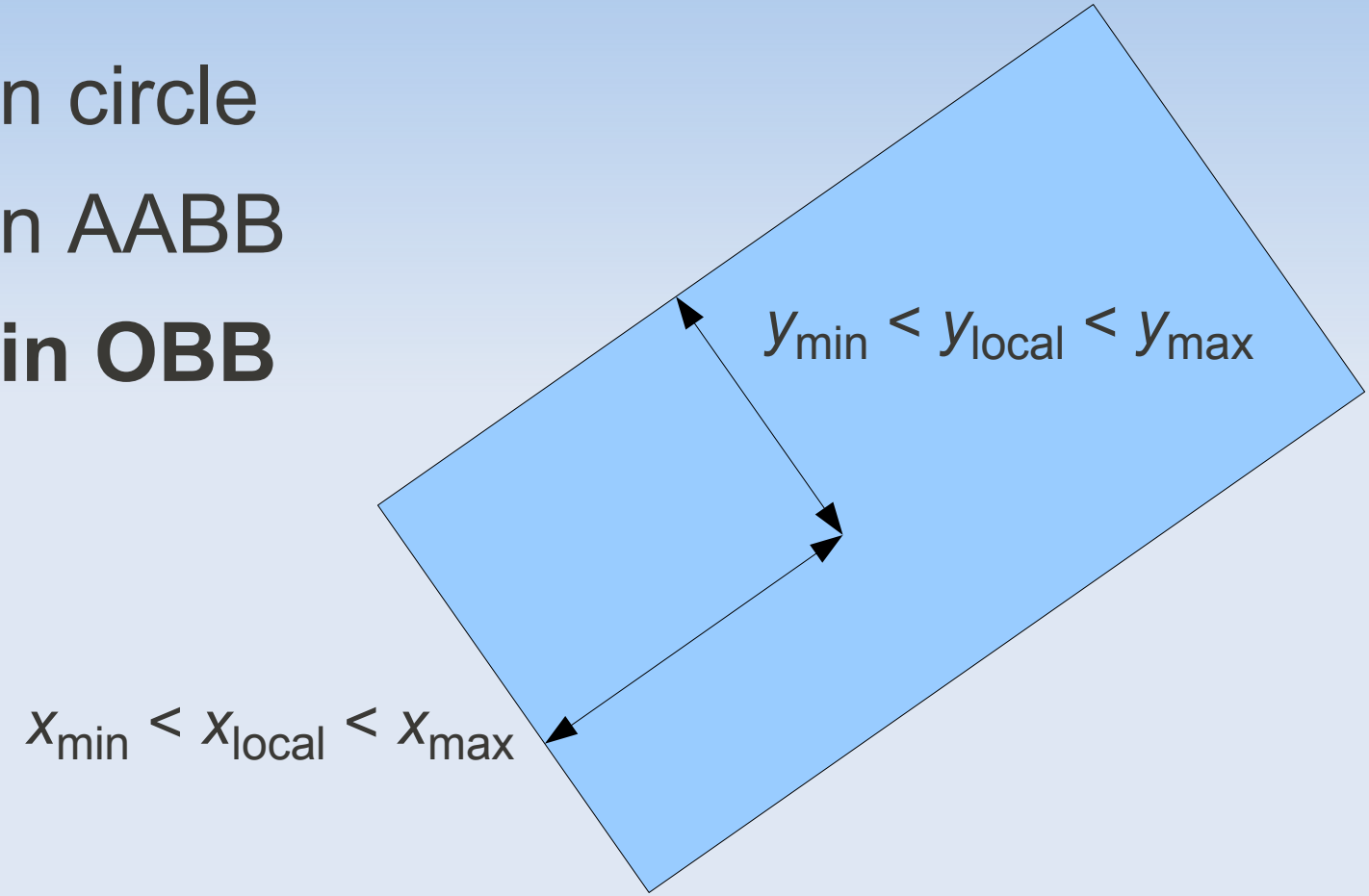
Intersection tests

- Point in circle
- **Point in AABB**



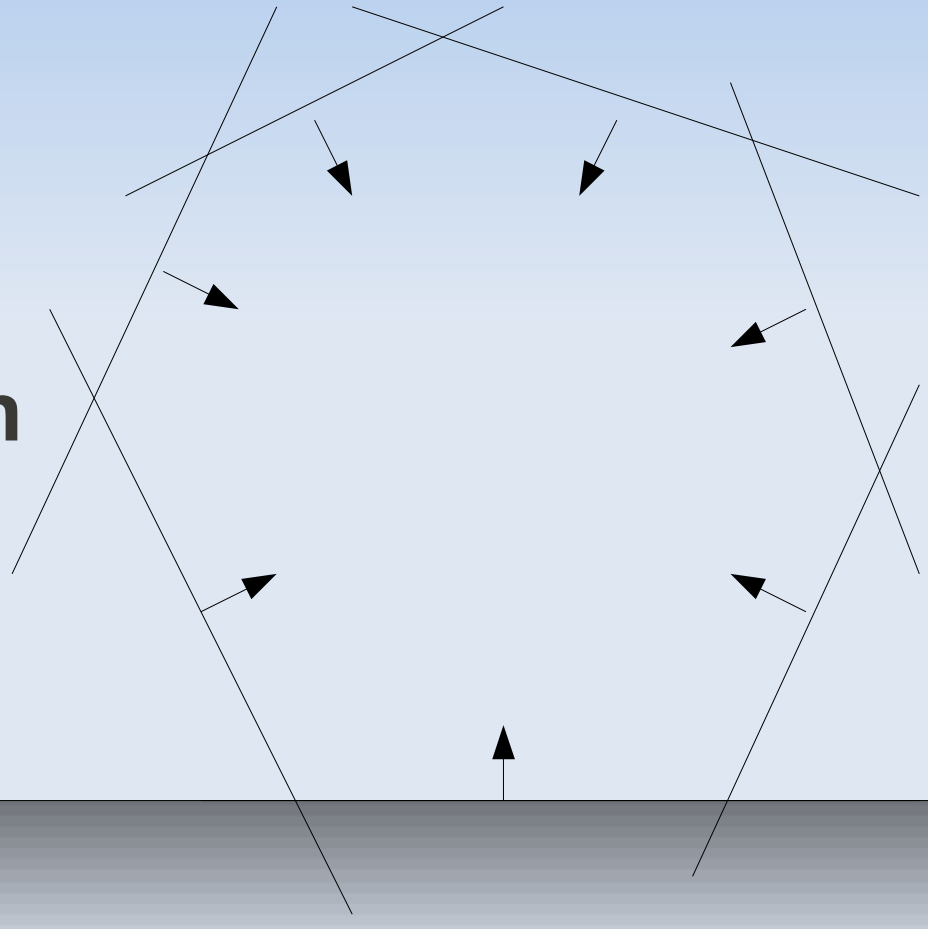
Intersection tests

- Point in circle
- Point in AABB
- **Point in OBB**



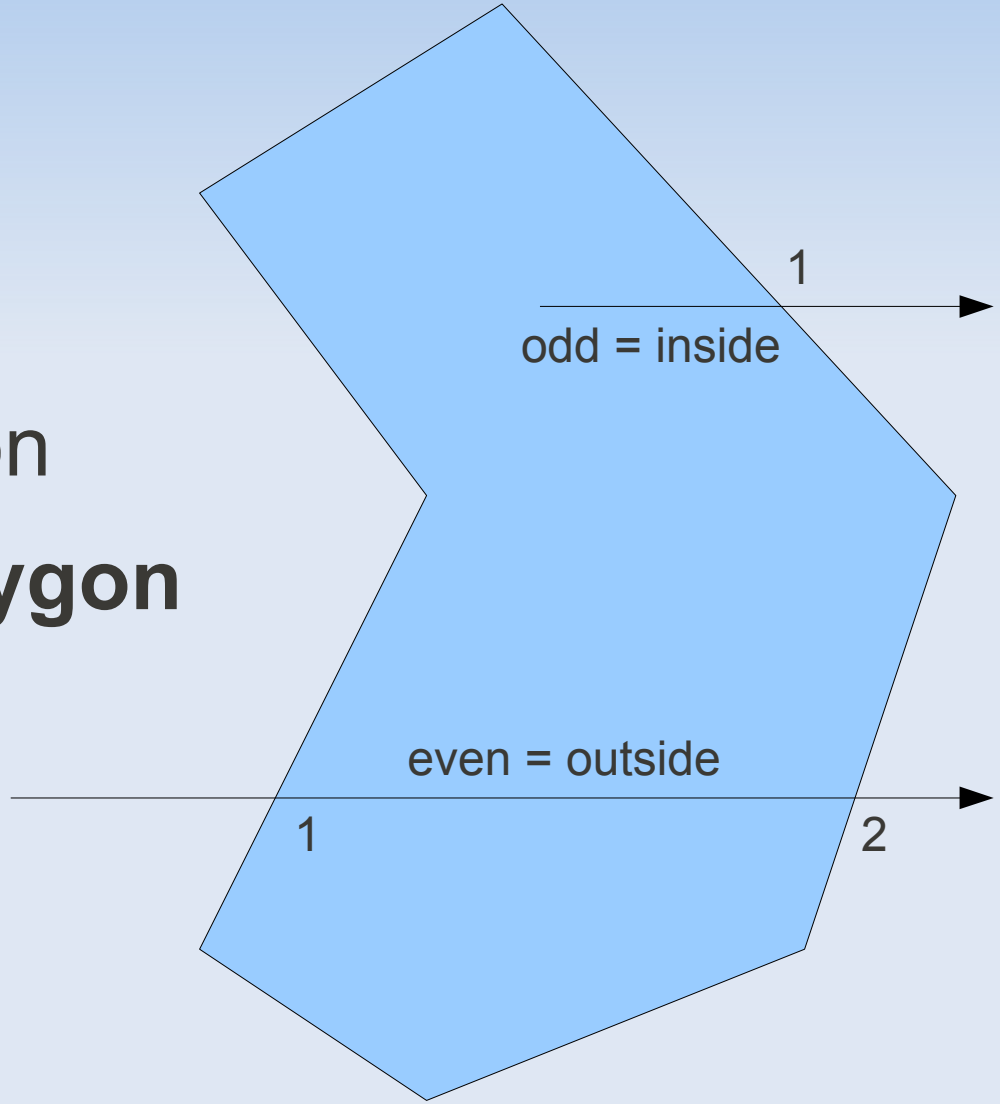
Intersection tests

- Point in circle
- Point in AABB
- Point in OBB
- **Point in convex polygon**



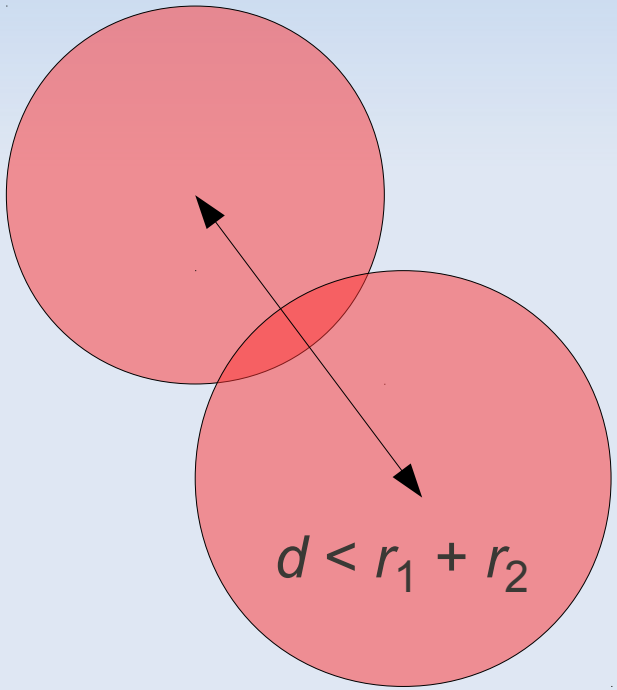
Intersection tests

- Point in circle
- Point in AABB
- Point in OBB
- Point in convex polygon
- **Point in concave polygon**



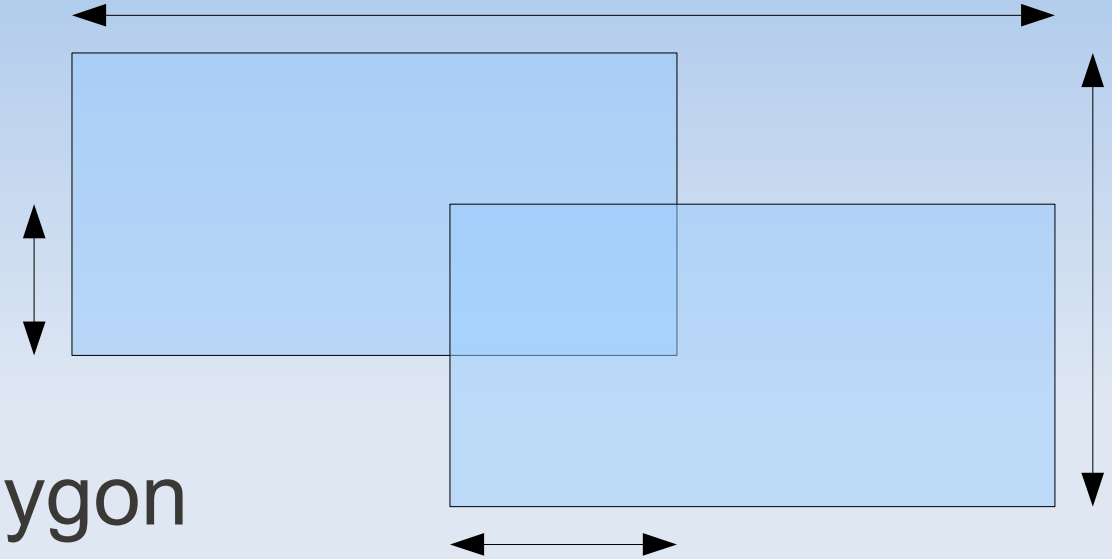
Intersection tests

- Point in circle
- Point in AABB
- Point in OBB
- Point in convex polygon
- Point in concave polygon
- **Circle-circle**



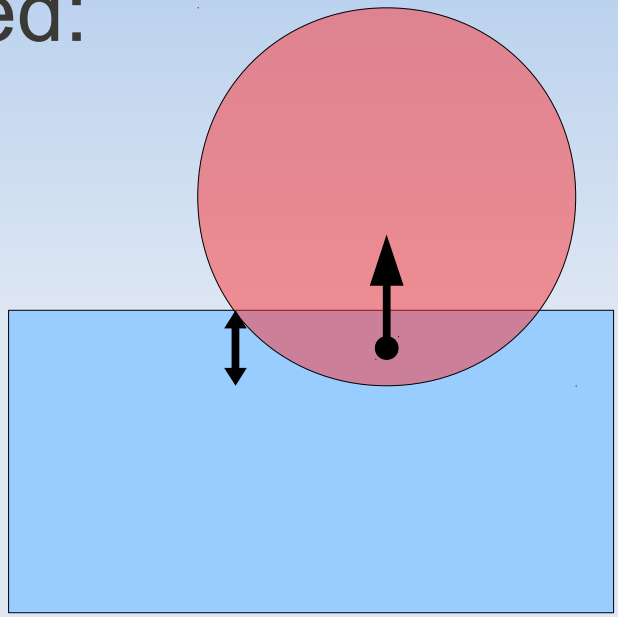
Intersection tests

- Point in circle
- Point in AABB
- Point in OBB
- Point in convex polygon
- Point in concave polygon
- Circle-circle
- **AABB-AABB**



Contact generation

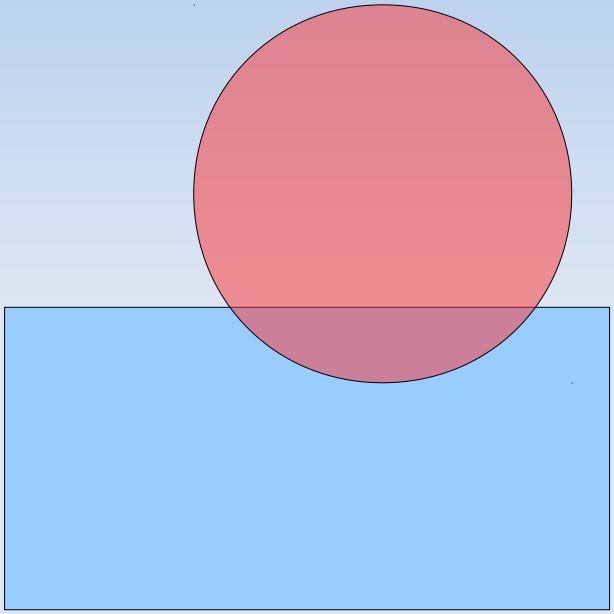
- Information generally needed:
  - Contact point
  - Contact normal
  - Amount of penetration



- For convex shapes in 2D, this isn't too hard
  - Concave shapes more difficult
  - 3D **much** more difficult

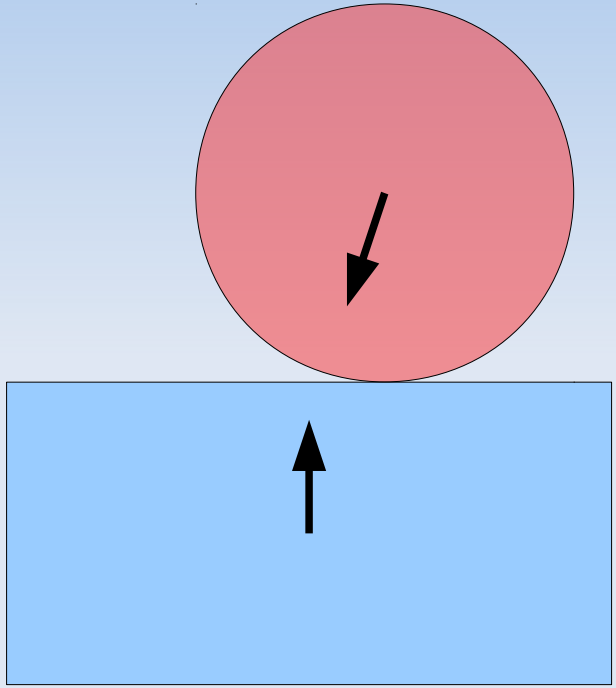
Collision resolution

- Remove penetration



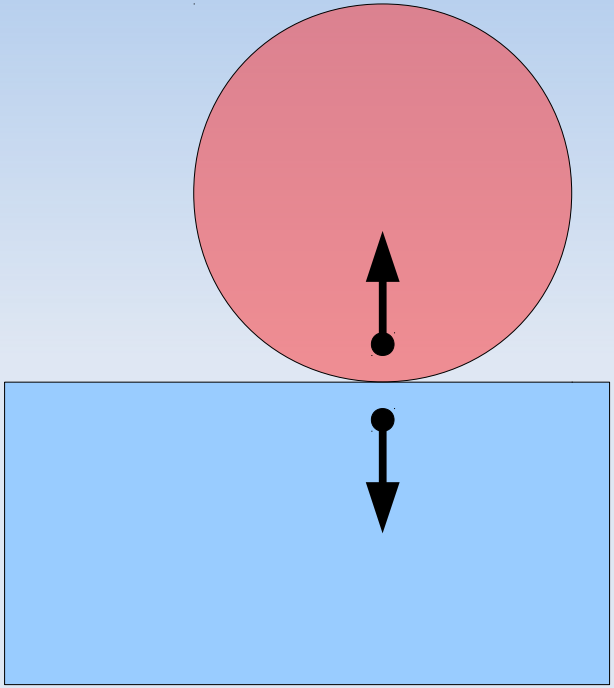
Collision resolution

- Remove penetration
- Calculate new velocities



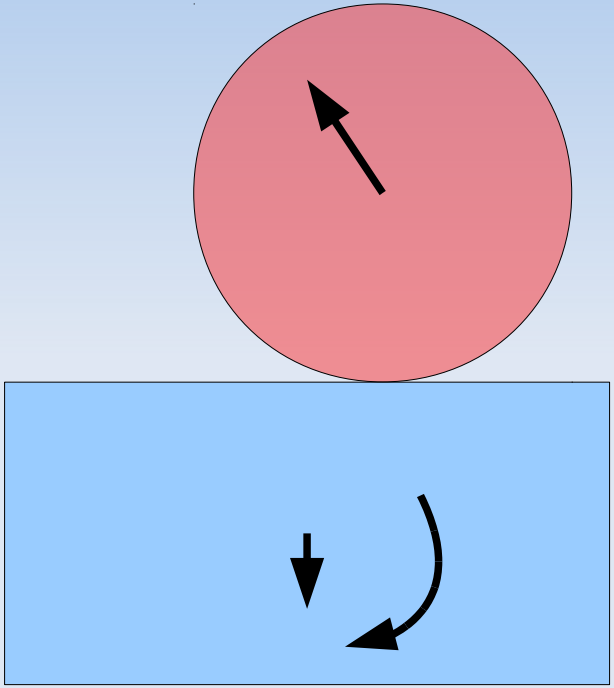
Collision resolution

- Remove penetration
- Calculate new velocities
  - Apply impulse at contact
  - Conservation of momentum
  - Coefficient of restitution



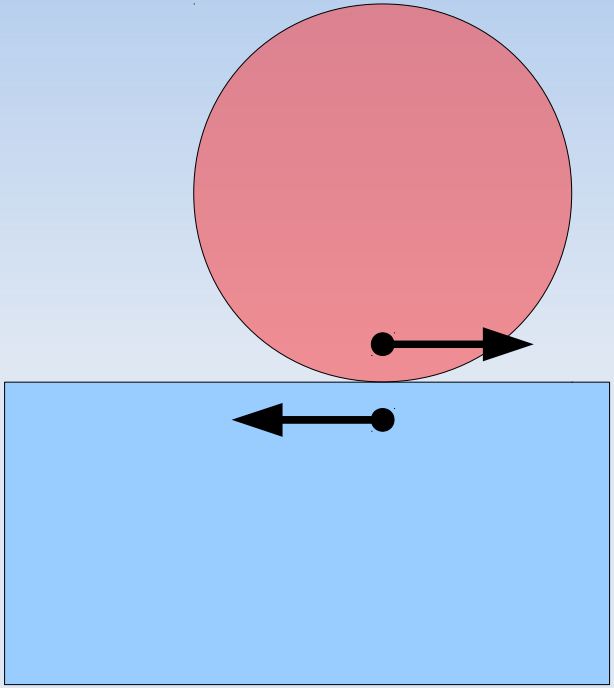
Collision resolution

- Remove penetration
- Calculate new velocities
  - Apply impulse at contact
  - Conservation of momentum
  - Coefficient of restitution
  - Includes rotation



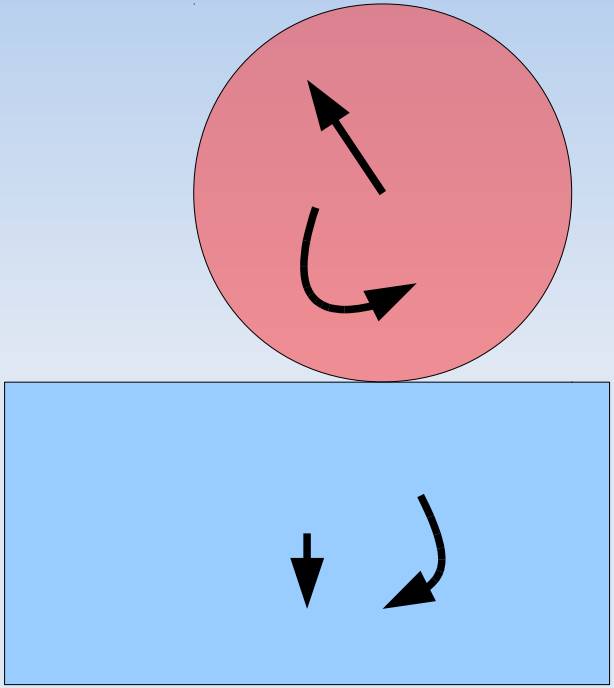
Collision resolution

- Remove penetration
- Calculate new velocities
  - Apply impulse at contact
  - Conservation of momentum
  - Coefficient of restitution
  - Includes rotation
  - Includes friction



Collision resolution

- Remove penetration
- Calculate new velocities
  - Apply impulse at contact
  - Conservation of momentum
  - Coefficient of restitution
  - Includes rotation
  - Includes friction
  - All at once



Collision resolution

- So we can resolve each contact
- But solving one may make another worse
- Could solve simultaneously
  - Build a massive LCP matrix
  - But not in real-time
- Instead, iterate over contacts repeatedly
  - Converge on global solution
  - Can balance computation time against accuracy

Putting it all together

- Every frame:
  - All bodies are moved simultaneously
  - Pairs of potentially colliding bodies are detected
  - Detailed contact information is generated
  - The collision resolver is run
    - Velocities are updated
    - Penetration is removed
  - All bodies are drawn at their new positions

Forget everything I just said.

Forget everything I just said

- That's all fairly useless information
  - You shouldn't make your own physics engine!
  - Box2D already exists
- My perspective as a game developer
  - A lot of this module is not that useful
  - It's been done for you

~~Anatomy of a physics engine~~

Anyone can make games, you should make games!

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

It's easy to make games

- The tools are amazing
  - FlashPunk
  - Unity
  - Stencyl
  - Twine
  - Box2D

It's fun to make games

- What if...
  - You made the snakes in Snake poop
  - The paddles in Pong were shaped like continents
  - Breakout blocks had to be kept onscreen
  - Seeing an enemy made you run away
  - Tetris blocks had an underground fight scene
  - You had a deathmatch game with invisible players

It's fun to make games

- What if...
  - You could only move one step per day
  - 2 players each had a microphone controlling speed
  - You had to hold your breath while underwater
  - A typing game gave increasingly morbid sentences
  - A guitar hero game had only one button
  - You had a videogame that didn't use the screen

Bear in mind

- Your first game will probably suck
  - And that's okay!
  - Make it terrible in a way that is awesome
- Don't be too ambitious
  - Make the simplest possible thing
  - Don't do everything from scratch
- Particles cover a multitude of sins
  - So does screen shake!

MAKE GAMES

Questions?

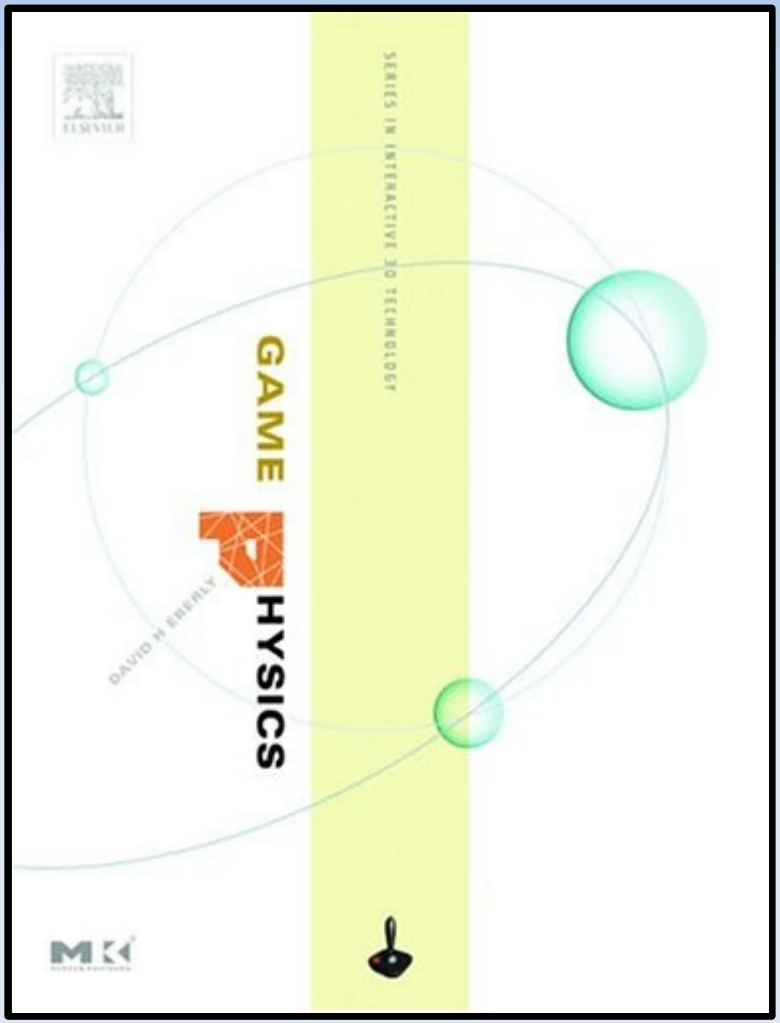
I want to create a physics engine!

- 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

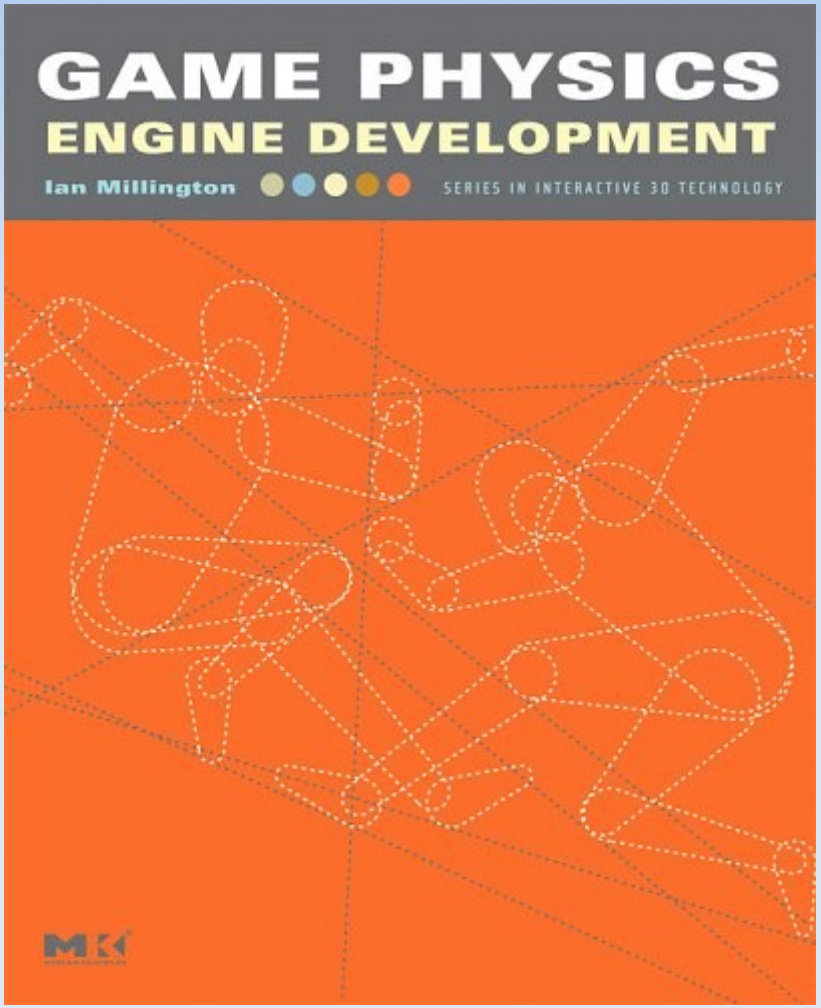
References

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
- Chris Hecker
  - http://chrishecker.com/Rigid_Body_Dynamics
- Thomas Jakobsen
  - http://www.teknikus.dk/tj/gdc2001.htm

2D physics engines

- Box2D
  - http://www.box2d.org/
- Chipmunk
  - http://wiki.slembcke.net/main/published/Chipmunk
- 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
- PhysX
  - https://developer.nvidia.com/physx