Intro to Game Physics Architecture

Jeff Morgan
jeffrey.morgan@digipen.edu

Contents

- Overview
- Terms
- Before you get started
- The Update Loop
- Game play
- Optimizations
- Misc
- Common pitfalls
- Resources

Terms

- Integration
- Collision Detection
  - Broad phase
  - Narrow phase
  - Manifold
  - Contact vs. Collision
- Collision Resolution
  - Impulse
  - Constraint
- Rigid Body

More Terms

- Collision Normal
- Penetration Depth
- Collider
- Collider Pair
- Ray cast
- InList
- Profiling
- Stacking

Before You Start

- MATH, MATH, MATH
- DEBUG Features
- Modular Design
- Fix Your Time Step

MATH, MATH, MATH

- Game Physics = Math (and lies)
  - Everything you do uses vector and matrix math
- Test it before you build your engine
  - Seriously
- Know your games coordinate system
  - {Graphics, Physics, AI} all need to be the same

DEBUG FEATURES

- Very important
  - Even if you “don’t” need it, you’ll use it
    - But you do need it, from the beginning
    - Make this a priority early
- Debug Draw
  - Being able to draw lines, circles, points
    - otherwise, you have to debug numbers ☹
- Frame Control
  - Pause, Step, and Continue
- Sandbox / Testbed

Modular Design

- You are hacking together physics
  - How you represent “physics” is going to change
    - The physics engine doesn’t have to
- Encapsulate those changes
  - Colliders
  - Materials
  - Rigid Bodies
  - Integrators, Resolvers, Collision Detectors

Fix Your Time Step

- Use the same Δt EVERY physics update
  - Decouple it from the games update
  - Eat up discrete time chunks every frame
    - Puts you ahead of your engine
- This makes your physics much more stable
- Makes your physics mostly deterministic
  - For replays, bug tracking, networking
- GafferOnGames: “Fix Your Timestep”

The Update Loop

- Integrate
- Optional: Broad Phase
  - Keep in mind, but wait to implement
- Detect Collisions
- Resolve Collisions
- Clean Up

Integration

- Moving your physics object
- Many types
  - Euler (Implicit, Semi-Implicit, …)
  - Verlet (Velocity, LeapFrog, …)
  - RK2, RK4
- There are trade offs to each integrator
  - Speed vs. Accuracy
  - ProTip: integrate more times with a faster integrator

Integration Example

Simple Integration Example:

- Note* You might want to make this a non-member function for ease of swapping integrators

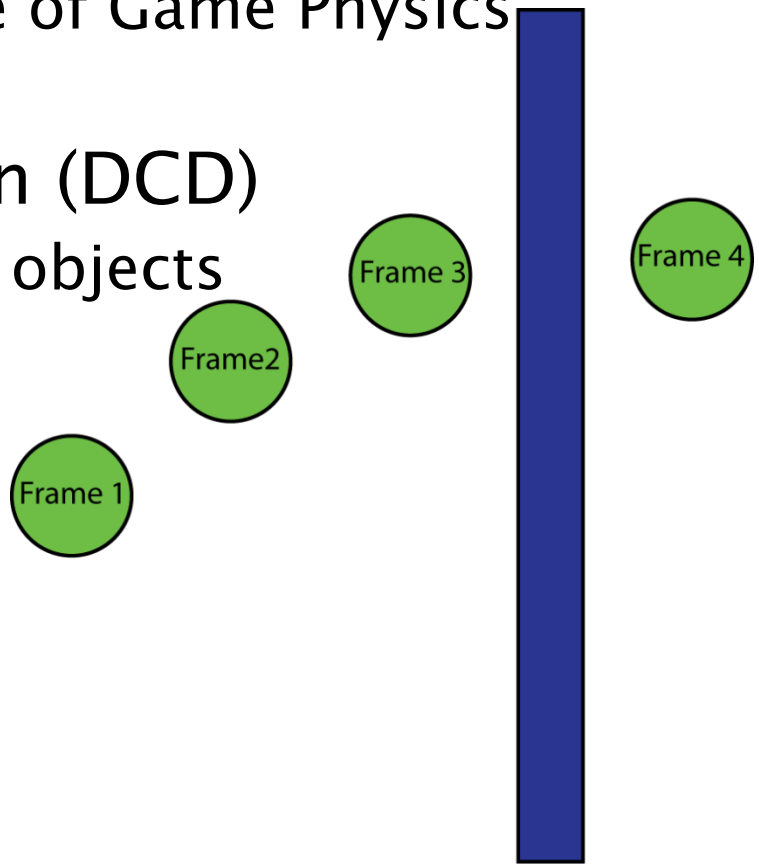
```
void RigidBody::IntegrateEuler(real dt )
{
  //Integrate velocity
  mVelocity += mForceAccumulator * mInverseMass * dt;

  //Integrate position
  mCenterMass += mVelocity * dt;

  //Integration orientation is for another lecture
}
```

Integration

```
class RigidBody
{
   //…

   Vec3 mCenterMass;        //Current center of mass
   Vec3 mVelocity;          //Current velocity
   Vec3 mForceAccumulator; //All accumulated forces (gravity)
   Vec3 mAngularVelocity;  //Angular velocity
   Mat3 mOrientation;       //Current orientation
   real mInverseMass;       // 1.0f / mass
   //Inverse Inertia Tensor is the objects resistance to rotation
   Mat3 mInvInertiaTensorM; //Model space
   Mat3 mInvInertiaTensorW; //World space

   PhysicsMaterial* mMaterial; //More on this later

   //…
};
```

Collision Detection

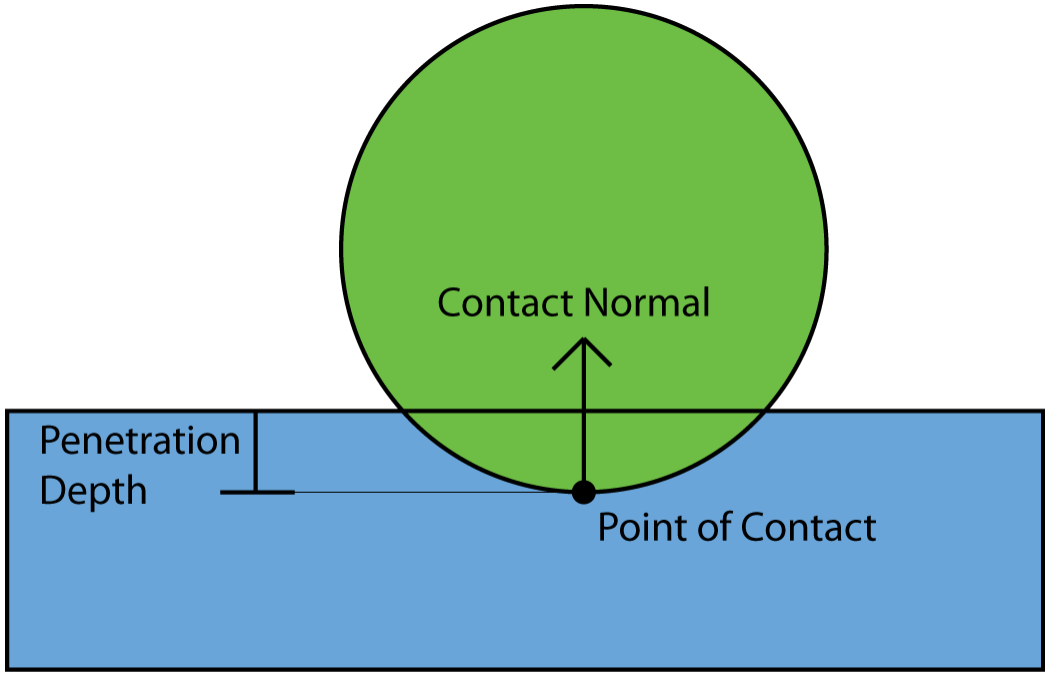
- There are “two” types of physics
  - ProTip: There’s only ONE type of Game Physics
- Discrete Collision Detection (DCD)
  - Every frame we’ll teleport the objects



Collision Detection

- Collision Detection is the process of checking two solids for intersections and returning relevant data so that we can resolve the collision

```
struct CollisionData
{
  //The colliding objects
  Collider* Objects[2];

  Vec3 ContactPoint;
  Vec3 Normal;
  real Penetration;
  //…
};
```



Collision Detection

```
class Collider
{
  //…

  u32 mId;       //Each collider should have a unique id
  Aabb mAabb;    //The objects bounding box
  real mVolume; //The volume of the object
  Vec3 mHalfExtents; //Vector from center to corner
  Vec3 mPosition;    //Relative position
  Mat3 mOrientationMtx; //Relative orientation

  //…
}
```

Collision Resolution

- Once you have collision you need to resolve them
  - In an impulse engine, you just push the two objects apart based on their mass and velocity
  - This uses the CollisionData structs from earlier

- Resting Contact vs. Collision
  - Micro Collisions
  - Non-Convex Rigid Bodies with Stacking (pdf)

Clean Up

- After resolving collisions you should inform the rest of your game about what happened
  - Send Messages / Events
    - OnCollision (with da)
    - OnSleep
    - OnEnter/OnExit a region
- You don't want this to happen during your physics step
  - What if logic deletes or moves an object?

Update Loop Example

```
//Integrate all objects
IntegrateObjects(mColliders);

//Query the broad phase for possible collisions
ObjectPairVector possibleCollisions;
mDynamicBroadPhase->GetCollisions &possibleCollisions);
mStaticBroadPhase->GetCollisions(&mColliders, &possibleCollisions);

//Walk through each possible collision
for(uint i = 0; i < possibleCollisions.size(); ++i)
{
	CollisionReport report;
	//Check if the two objects collided
	if DetectCollision(possibleCollisions[i].A, possibleCollisions[i].B, &report) == true)
	{
		//Allocate a copy of the report and store it elsewhere
		//NOTE* You should use a memory manager for these allocations as there will be a large amount of them
		per frame
		StoreCollisionReport new CollisionReport(report));
	}
}

//At a later time…
for(uint i = 0; i < mCollisionReports.size(); ++i)
{
	ResolveColision(CollisionReports[i]);
}

//When you are all done...
DispatchMessages();
```

Gameplay

- Make others want to use your engine
  - Designers can make use of collisions and filters
  - AI can make use of ray cast, kinematics
- Collisions are both logical and physical
  - A bomb might hit something
    - Physics detects the collision
    - Game Logic handles the collision with an explosion
- Your main character can be controlled with physics!

Ray Cast

- What is ray casting
  - A ray is a point and a direction
    - $P + t\mathbf{d}, t >= 0$
  - A ray can return the first object it hits, multiple objects, or objects of certain types

- Uses
  - Line of Sight (LOS)
  - Fast moving objects (bullets)
  - Continuous Collision Detection (CCD)
  - Movement for AI (feelers)
  - …and more

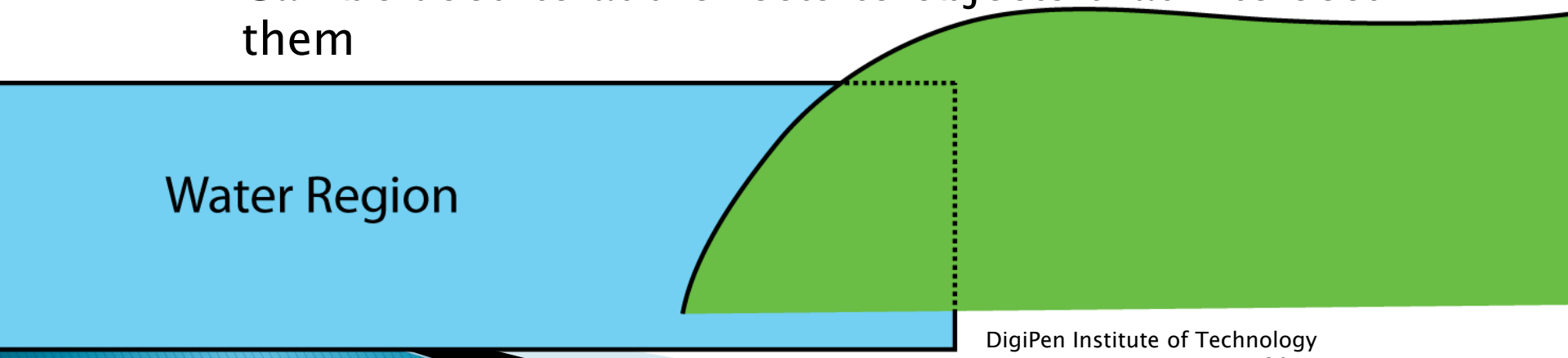
Collision Masks

- A way to stop certain types of objects from colliding
  - Each object gets a bit field that corresponds to a type
  - Flags are masked against each other to see if the types can collide

- Example
  - In our freshman game, we didn't want players and NPCs to collide

Triggers, Ghost, and Regions

- Trigger/Ghost collisions
  - A purely logical collision
    - A collision message is sent out for logic to handle, but physics doesn't resolve the collision
- Regions
  - Similar to Triggers, but used as a component of an obj.
    - Can be used to add effects to objects that intersect them



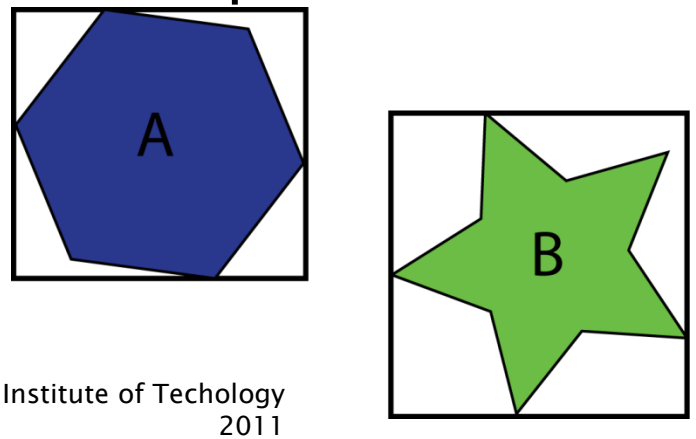
Extras and Optimizations

- ProTip: Optimize after your core engine is solid

- A few types:
  - Broad Phase
  - Sleeping objects
  - Islanding
  - Materials

Broad Phase

- Broad Phase is used to remove unrelated objects from costly collision test
  - In out example, Object A can not possible be in contact with Object B, so we won't send it to the collision detector
- You should start of with a n^2 algorithm!
  - Use it to test against any new broad phases



Broad Phase

- There are MANY types of broad phases
  - Sweep And Prune (SAP)
    - Also called Sort and Sweep
  - AABB trees
  - KD trees
  - Binary Search Partition Tree (BSP)
  - …and more
- There's no best broad phase
  - ProTip: SAP and AABB Tree are good to know

Broad Phase

- Two types of broad phases
  - Dynamic
    - Designed for moving objects
    - Updated every frame
  - Static
    - Designed for non-moving objects
    - Usually built once when the world is created

```
struct ColliderPair
{
  //...

  Collider* mColliders[2];
};
```

- Getting information from your Broad Phase.
  - Dynamic – Pass in an empty array of collider pairs. The broad phase should fill it out with object pairs that are close enough to be tested for collision
  - Static – Pass in an array of all the dynamic objects as well as an empty array of collider pairs. The broad phase should fill it out with possible collisions

Sleeping Objects

- ProTip: If an object hasn't moved in a while and isn't likely to, don't update it. ☺
- This takes some work, but it isn't too hard.
  - One technique uses a running total of how much an object moved over the last few frames
    - If that value drops below a threshold, put the object to sleep
- Waking and sleeping objects can be tricky
  - Islanding helps!

Islanding

- Also called Contact Graphs
  - Contact graphs also have directionality
- Islands group together related contact groups
  - This allows you to thread contact resolution
  - Can also be used to detect special kind of contacts
    - I checked to see if my player was indirectly involved in a collision with terrain
  - Islanding really helps with waking up objects in a cluster around a collision

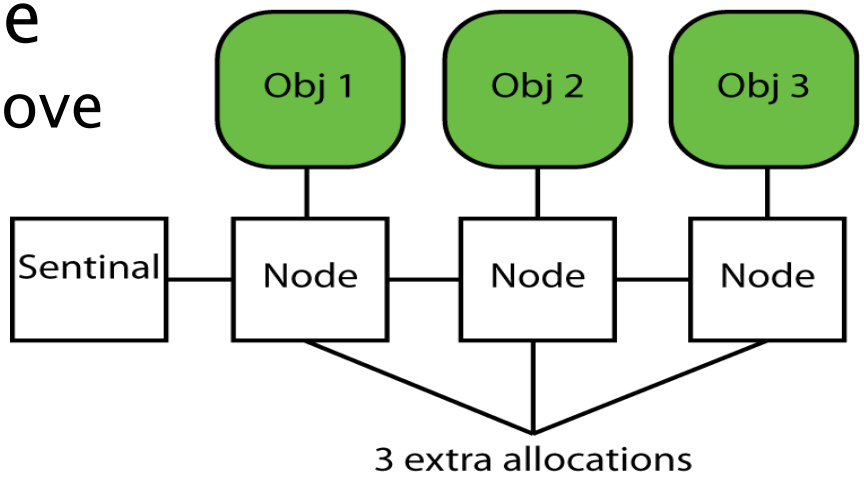
Materials

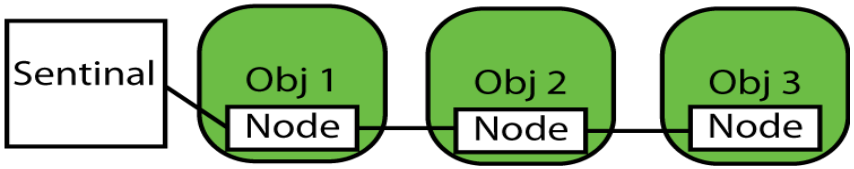
- I really with I had done this in my first engine
  - ProTip: Modular design is a good thing
- Objects share these materials
  - stored as a pointer to the material
- Data
  - Restitution is a value between 0 and 1 determining the amount of energy retained on a collision
  - Static and Dynamic Friction
  - Density

```
class PhysicsMaterial
{
  //...

  real mRestitution;
  real mStaticFriction;
  real mDynamicFriction;
  real mDensity;
};
```

Misc.

- InList is super awesome
  - Constant time add / remove
  - No extra allocations
  - The “Node” is stored as a member variable of the object





Misc.

- Profile your code
  - VerySleepy is free
  - You can write your own simple profiler
- Profiling allows you to see exactly what is slow about your code
  - No guesswork necessary

Misc.

- Constraints
  - Are a mathemagical type of collision resolution
  - They allow you to model very cool things, in a unified way.
    - Hinges, Joints, Pulleys
  - The are a lot more difficult to implement and should only be tackled after you have done impulses.

Common Pitfalls

- Collision Detection
  - It will be the source of most of the engines problems
    - Even problems that don't appear like a collision detection issue
- Stacking
  - Things like Jitter, Drift or Bouncing will occur

Resources

- The Orange Book,
  - Real-Time Collision Detection
- The other orange book
  - Game Physics Engine Development
- Advanced Character Physics, pdf

- Physics club!
  - Seriously

Special Thanks

- Thanks to Josh Claeys for all of the code samples, picture, and a few of the paragraphs on these slides
  - Seriously ☺

Questions?