Homework Policy

- There are **four** late days without excuses just for homework assignments. (No, you can not use it in your final projects)
- All homework assignments are due before class (1:30 PM EST)
- All homework assignments have 100 base pts and at most 20 extra pts.
- TAs penalize late submissions for 20 pts per day.
- TAs will always grade the latest submission
- [Code of Academic Integrity](#)
Homework Assignments

- Rigid Body Simulation (two weeks)
  - Assign on Feb. 9
  - Due on Feb. 23 before class

- Deformable Body Simulation (two weeks)
  - Assign on Feb. 23
  - Due on Mar. 9 before class

- Fluid Simulation (three weeks including spring break)
  - Assign on Mar. 9
  - Due on Mar. 30 before class
Meet your TAs

Xinjie Ma

Ying Li

Harmony Li

Yu Wang
Rigid Bodies Simulation

CIS 563 SPRING 2015, HW1 RECITATION
Overview

- Demo
  - https://www.youtube.com/watch?v=cznCbHlrYLY
  - [Executable]
Overview – Main Simulation Loop

- An intuitive approach:
  - Update pos & vel
  - Process Collision
  - Process Contact

- Problem?
Overview – Main Simulation Loop

- Adjusted approach
  - Process Collision (No Collision)
  - Update Velocity (Block gains downwards velocity due to gravity)
  - Process Contact (Stop normal motion by processing contact)
  - Update Position (Slides down with no bounce)
ResolveCollision reflects the velocity using the coefficient of restitution.

ResolveContact cancels the normal direction velocity when two objects are stacking.
Collision parameters:

- $\varepsilon$: coefficient of restitution
  
  \[ \text{vel}_{n,new} = -\varepsilon \times \text{vel}_{n,old} \]

- $\mu$: coefficient of friction
  
  Kinetic Friction:
  
  \[ F_f = \mu \times F_n \text{ OR } J_f = \mu \times J_n \]

  Static Friction:
  
  \[ F_f \leq \mu \times F_n \text{ OR } J_f \leq \mu \times J_n \]
Overview – Impulse

\[ J = \int F \, dt \]

- Impulse-momentum theorem:
  \[ \Delta p = m \Delta v = J \]
  \[ v_{\text{new}} = v_{\text{old}} + \frac{J}{m} \]

- Angular impulse:
  \[ \Delta L = I \Delta \omega = r \times J \]
  \[ \omega_{\text{new}} = \omega_{\text{old}} + I^{-1} (r \times J) \]

- Velocity Change at contact point:
  \[ u_{\text{new}} = u_{\text{old}} + KJ, \text{ where } K = \frac{1}{m} \delta + r^* \! T \! I^{-1} r^* \]

- How to get this \( r^* \): [LINK]
Overview – Impulse

- How to get this Impulse?
  - Collision Only Model
  - Collision + Friction Model (Static and Kinetic Friction)
Data Structure - Class RigidBody

- key members:
  - State:
    - Matrix4 m_transformation
    - Vector3 m_velocity;
    - Vector3 m_angularVelocity
    - Vector3 m_queuedDeltaVelocity;
    - Vector3 m_queuedDeltaAngularVelocity;
  - Material:
    - float density;
    - float restitution;
    - float friction;
    - Vector3 color;
Data Structure – Class Rigid Body

- Key Subclasses:
  - Sphere
  - Cube
  - Ground
Key Methods:
- Lots of get/set attribute methods
  - Material
  - Bounding box
  - Mass
  - Initial Tensor
  - States (pos/ori/vel/avel)
- Advance position/velocity
- Apply impulses
- Save/Restore current state
Data Structure - Class RigidBody

- Key Methods Example: Inertial Tensor
  - List of moments of inertia
    - virtual Vector3 GetNormalLocalInertialTensor() const = 0;
    - Vector3 GetLocalInertialTensor() const;
    - virtual Matrix3 GetInertialTensor() const;

\[
I_{xx} = M \int_V (y^2 + z^2) dV \\
I_{xy} = -M \int_V xy dV
\]

... but are Constant in Body Space

\[
I(t) = R(t)I_{body}R(t)^T
\]
**Data Structure - Struct Intersection**

- **Key Members:**
  - `RigidBody* bodyA;` // this body
  - `RigidBody* bodyB;` // the body this penetrates into
  - `Vector3 contactPoint;` // the point where contact will be applied (in world space)
  - `Vector3 contactPointA;` // the same point in bodyA space
  - `Vector3 contactPointB;` // the same point in bodyB space
  - `Vector3 outVector;` // the vector of restitution for BodyA (in world space)
  - Out Vector is Normalized.
Data Structure - Struct Intersection

Key Members:

- ContactPoint
- ContactPointA
- ContactPointB
- OutVector
- Body A
- Body B
Data Structure – Struct Intersection

Key Methods:

- void FindIntersection(RigidBody * bodyA, RigidBody * bodyB, std::vector<Intersection> & intersections);
- void FindIntersectionBoxBox(Box * box, Box * box2, std::vector<Intersection> & intersections);
- void FindIntersectionBoxGround(Box * box, Ground * ground, std::vector<Intersection> & intersections);
- void FindIntersectionBoxSphere(Box * box, Sphere * sphere, std::vector<Intersection> & intersections);
- void FindIntersectionGroundSphere(Ground * ground, Sphere * sphere, std::vector<Intersection> & intersections);
- void FindIntersectionSphereSphere(Sphere * sphere, Sphere * sphere2, std::vector<Intersection> & intersections);
Data Structure – Class World

- **Key Members:**
  - `std::vector<RigidBody*> m_bodies;`

- **Key Methods:**
  - `void Simulate(float dT);`
  - `void ResolveCollisions(float dT);`
  - `void AdvanceVelocities(float dT);`
  - `void ResolveContacts(float dT);`
  - `void AdvancePositions(float dT);`
  - `static void ResolveIntersection(Intersection &i, float epsilon, bool immediate = false);`

- **Save/Restore Entire States**
Data Structure – Class XMLWorldVisitor

- Support Ground/Box/Sphere.
- Support Pos/Vel/Ori/Avel
- Support all elements described in Material.h

<table>
<thead>
<tr>
<th>float density;</th>
<th>float friction;</th>
</tr>
</thead>
<tbody>
<tr>
<td>float restitution;</td>
<td>Vector3 color;</td>
</tr>
</tbody>
</table>
void World::Simulate(float dT)
{
    // LOOK this is the basis of simulation
    ResolveCollisions(dT);
    AdvanceVelocities(dT);
    ResolveContacts(dT);
    AdvancePositions(dT);
}
ResolveCollision reflects the velocity using the coefficient of restitution.

ResolveContact cancels the normal direction velocity when two objects are stacking.
$u'_{rel,n} = -\varepsilon u_{rel,n}$

-1 0 1

Contact:
Slow down the velocity and finally cancel it

Collision:
Bounce it back
Work Flow – Resolve Collision

- **Iteration Loop**

  ```
  {
    Save state;
    Advance state (position and velocity) and find intersections;
    Restore state;
    Resolve interaction and enqueue impulses with positive coefficients of restitution;
    Apply queued impulses.
  }
  ```
Work Flow – Resolve Collision
Apply external acceleration caused by:
  - Gravity
  - Other field forces
  - Etc…

Missing TODO in your framework:

```cpp
void World::AdvanceVelocities(float dT) {
    // TODO: add gravity
}
```
Work Flow – Resolve Contact

- Iteration Loop
  
  ```
  { 
  Save state;
  Advance position and find intersections;
  Restore state;
  Resolve interaction and apply impulses with negative coefficients of restitution;
  }
  ```

Keep in mind: Resolve contact is always used for stacking objects.
Update position and orientation using current velocity and angular velocity.
WorkFlow – Missing Part

- Resolve Interaction
- Find Intersection

Work Flow – Resolve Contact

- Iteration Loop
  
  ```
  { 
    Save state;
    Advance position and find intersections;
    Restore state;
    Resolve interaction and apply impulses with negative coefficients of restitution;
  }
  ```

Work Flow – Resolve Collision

- Iteration Loop
  
  ```
  { 
    Save state;
    Advance state and find intersections;
    Restore state;
    Resolve interaction and enqueue impulses with positive coefficients of restitution;
    Apply queued impulses.
  }
  ```
void World::ResolveIntersection(Intersection &i, float epsilon, bool immediate)

- Nonconvex Rigid Bodies with Stacking Section 6 and 7.
void World::FindIntersections(std::vector<Intersection> & intersections)
  Brute Force Algorithm is given. (with time complexity of $O(n^2)$)
  Any algorithm faster than this is acceptable.
  Sweep and Prune
  Sweep and Prune + PCA (Principal Component Analysis)
  Oct-tree
  Kd-tree
  Etc.
Key idea: if two objects collide with each other, their projections on an arbitrary axis must overlap with each other.
Extra - Make it faster

- Get it faster & faster & faster!!!
  - 20 pts Extra credit for fastest three competitors.

- FindIntersections() is the best place that you should be working on
- Other code optimizations are welcome
- RunBenchmark(std::string const& name, int frames) in benchmark.cpp is provided for you to test.
Extra - Make it more stable

- Stacking problem. (Shock propagation)
  - Method set/has_infiniteMass has been provided to you.
  - Section 8.2 of the stacking paper.
Extra – Interesting scene for your demo-reel!
Extra – Advanced Rendering

- Export to .obj sequence
- Render it using advanced tools such as Maya.
Start Boom Blox...

- **Things need **TODO**:**
  - There are 7 places you need to fill in the start code. (6 places commented with “TODO”, another one is adding gravity in World::AdvanceVelocities)
    - Intersection.cpp (3 TODO)
    - World.cpp (2 TODO)
    - Box.cpp (1 TODO)

- **Things need to **LOOK** at:**
  - There are lots of places you can extend in the code commented with “LOOK”
Acknowledgement

- Ben Sunshine-Hill
  - For creating this framework
- Xinjie Ma
  - For making most of these slides
LET'S BLOW SOME $HIT UP!
LET’S BLOW SOME $HIT UP!

Good Luck 😊
KEEP CALM AND CODE C++