What is a Physics Engine?

- What are the things our Physics Engine should handle?
- How should it coordinate with other aspects of the game engine?
Integrating a Physics Engine

- Physics engine called during by game loop
- Runs simulation based on a time step
- Provides updated position and orientation
Simulation Objects

• Rigid bodies are the basic physics objects in Bullet
• Have “rigid body” properties:
  • Mass
  • Motion state
  • Local inertia
  • Collision shape
• Three types of rigid bodies:
  • Dynamic (controlled by rigid body dynamics)
  • Static (no mass, for collision detection only)
  • Kinematic (user-controlled, no mass)
Collision Shapes

• Collision shapes are bounding box that define physics objects
• Reuse of shapes among collision bodies saves memory
• Primitive shapes include spheres, boxes, cylinders, capsules etc
• Mesh shapes include convex hulls, convex triangle meshes, heightfield terrain etc
• Primitive shapes provide an efficiency/accuracy tradeoff
• Meshes better for arbitrary geometry that requires higher degree of accuracy
Simulator Class Example

class Simulator {
protected:
    btDefaultCollisionConfiguration* collisionConfiguration;
    btCollisionDispatcher* dispatcher;
    btBroadphaseInterface* overlappingPairCache;
    btSequentialImpulseConstraintSolver* solver;
    btDiscreteDynamicsWorld* dynamicsWorld;
    btAlignedObjectArray<btCollisionShape*> collisionShapes;
    std::deque<GameObject*> objList;

public:
    Simulator();
    ~Simulator();

    void addObject(GameObject* o);
    bool removeObject(GameObject* o);
    void stepSimulation(const Ogre::Real elapsedTime,
                        int maxSubSteps = 1, const Ogre::Real fixedTimestep = 1.0f/60.0f);
Simulator::Simulator() {
    collisionConfiguration = new btDefaultCollisionConfiguration();
    dispatcher = new btCollisionDispatcher(collisionConfiguration);
    overlappingPairCache = new btDbvtBroadphase();
    solver = new btSequentialImpulseConstraintSolver();
    dynamicsWorld = new btDiscreteDynamicsWorld(dispatcher,
                                               overlappingPairCache,
                                               solver,
                                               collisionConfiguration);
    dynamicsWorld->setGravity(btVector3(0.0, -0.098, 0.0));

    // Add collision shapes to reuse among rigid bodies
}

void Simulator::addObject (GameObject* o) {
    objList.push_back(o);
    dynamicsWorld->addRigidBody(o->getBody());
}
Simulation World and Objects

- `btDefaultCollisionConfiguration` defines default setup for memory and collision
- `btCollisionDispatcher` defines default thread dispatcher (not parallel)
- `btDbvtBroadphase` checks for number of collisions to resolve (can also use `btAxis3Sweep`)
- `btSequentialImpulseConstraintSolver` defines default constraint solver (not parallel)
- `btDiscreteDynamicsWorld` creates a world based on simulation settings
- `btRigidBody` (`btCollisionObject`) manages collision detection for an object:
  - Shape, axis-aligned bounding box (AABB), and transform
Connecting to Ogre…

- Ogre renders entities based on scene node transforms
- Bullet models physical interactions of rigid bodies and forces
- What should a Game Object used by both libraries include?
class GameObject {
protected:
    Ogre::String name;
    Ogre::SceneManager* sceneMgr;
    Simulator* simulator;
    Ogre::SceneNode* rootNode;
    Ogre::Entity* geom;
    btCollisionShape* shape;
    btScalar mass;
    btRigidBody* body;
    btTransform tr;
    btVector3 inertia;
    OgreMotionState* motionState;
Simulation Loop vs Game Loop

• How do we know when something changes in one or the other?
• What should we do when this change happens?
Bullet MotionStates

• MotionStates provide an interface between the simulation and game loop
  • Used for moving objects
  • Movements in simulation are passed to rendered objects in the scene
• Static objects do not need motion states
• Kinematic objects communicate movement from game loop to Bullet
  • Motion states used in reverse
Using MotionStates

• btDefaultMotionState provides basic motion state functionality
• Use motion state to initialize object position when it enters the simulation (getWorldTransform)
• Call motion state during simulation to move body in rendering world (setWorldTransform)
class OgreMotionState : public btMotionState {
protected:
    Ogre::SceneNode* mVisibleobj;
    btTransform mPos1;

public:
    OgreMotionState(const btTransform &initialpos, Ogre::SceneNode* node) {
        mVisibleobj = node;
        mPos1 = initialpos;
    }

    virtual ~OgreMotionState() {}

    //Provides flexibility in terms of object visibility
    void setNode(Ogre::SceneNode* node) {
        mVisibleobj = node;
    }
}
Ogre-Bullet Interface cont’d

virtual void getWorldTransform(btTransform &worldTrans) const {
    worldTrans = mPos1;
}

virtual void setWorldTransform(const btTransform &worldTrans) {
    if (mVisibleobj == nullptr)
        return; // silently return before we set a node
    btQuaternion rot = worldTrans.getRotation();
    mVisibleobj->setOrientation(rot.w(), rot.x(), rot.y(), rot.z());
    btVector3 pos = worldTrans.getOrigin();
    mVisibleobj->setPosition(pos.x(), pos.y(), pos.z());
};
Collision Callbacks

• ContactTest tests object against all other objects and calls ContactResultCallback
  
  • simulator->getWorld()>
  contactTest(body, contactCallback);

• Implement ContactResultCallback to determine:
  • Which collisions should result in a hit
  • What information about the hit to store/use
Note on OgreBullet…

• Wrapper designed to integrate Bullet physics engine into Ogre graphics engine
  • Out-dated and not necessarily supported by later versions of Ogre
  • Generally I’d advise against using it, but you’ll almost certainly see it while searching for “Ogre” and “Bullet” :)
Further References

- Bullet Wiki (http://bulletphysics.org/mediawiki-1.5.8/index.php/Main_Page)
- Bullet SDK (http://bulletphysics.org/Bullet/BulletFull/annotated.html)
- Ogre + Bullet – Beginner’s Tutorial (http://oramind.com/ogre-bullet-a-beginners-basic-guide/)