Bullet Physics

Daniel Sun

What is a Rigid Body?

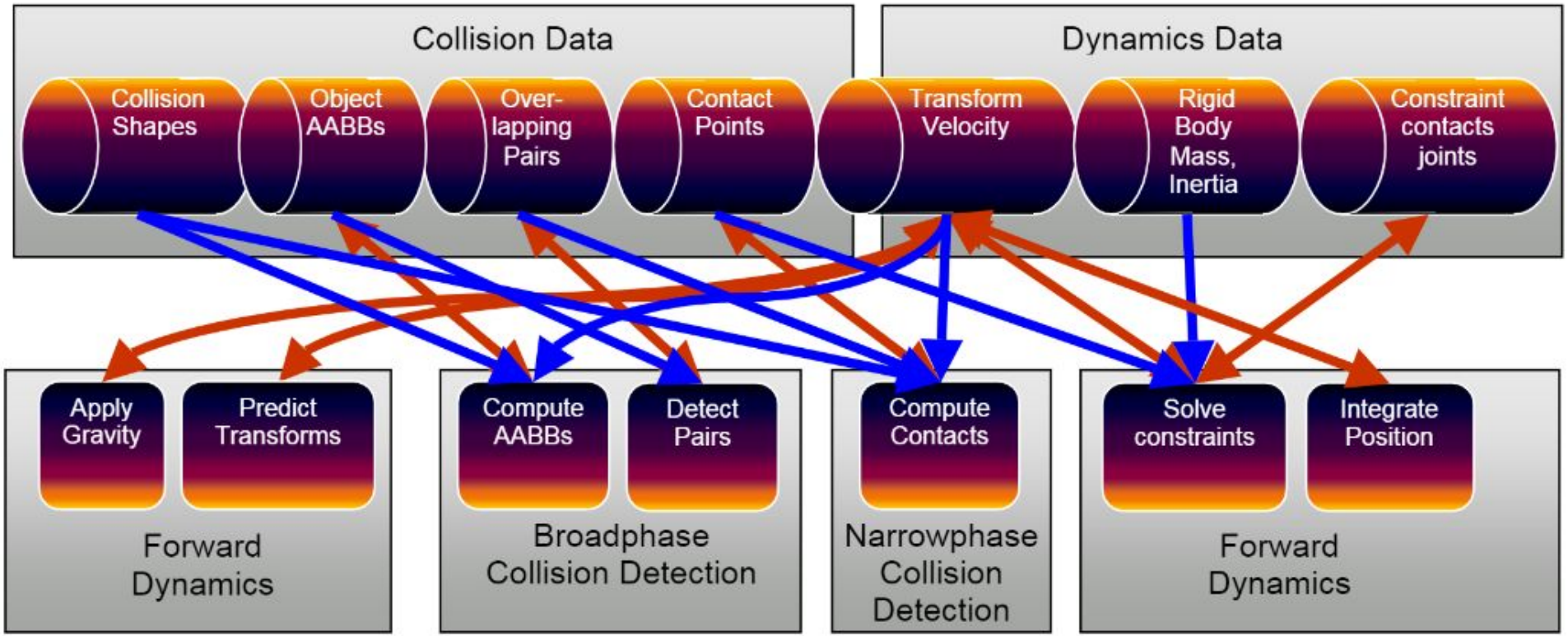
“A body is formally regarded as rigid if the distance between any set of two points in it is always constant. In reality no body is perfectly rigid. When equal and opposite forces are applied to a body, it is always deformed slightly. The body’s own tendency to restore the deformation has the effect of applying counterforces to whatever is applying the forces, thus obeying Newton’s third law. Calling a body rigid means that the changes in the dimensions of the body are small enough to be neglected, even though the force produced by the deformation may not be neglected.” - Britannica

- A rigid body is a body that does not deform
- Convenient ‘model’ as it simplifies simulations

Bullet Physics - An Introduction

- Bullet Physics is a C++ library that provides real time physics simulation, including soft-body and cloth simulations
- Only the rigid body simulation features are used in this physics project
- Commonly used in games and game engines
- Features for robotics
- Open source

Bullet Simulation Pipeline



Gory Technical Details (Code Analysis)

```
// collision configuration contains default setup for memory, collision setup
btDefaultCollisionConfiguration* configuration = new btDefaultCollisionConfiguration();

// default collision dispatcher, responsible for specialized collision algorithms based on shape
btCollisionDispatcher* dispatcher = new btCollisionDispatcher(collisionConfiguration);

// finds and creates a list of objects that will or might collide
btBroadphaseInterface* broadphase = new btDbvtBroadphase();

// default constraint solver, integrates velocity and position
btSequentialImpulseConstraintSolver* solver = new btSequentialImpulseConstraintSolver();

// the collision world is responsible for the physics simulation, this one only implements rigid body
dynamics, but there are others that support soft body simulation as well
btDiscreteDynamicsWorld* world = new btDiscreteDynamicsWorld(dispatcher, broadphase, solver, configuration);

// sets the gravity to -10 m/s downwards
world->setGravity(btVector3(0, -10, 0));

// update the simulation
world->stepSimulation(1.0f / 60.0f)
```

btDiscreteDynamicsWorld::stepSimulation()

```
// calculate number of simulation steps, usually one or zero by default
m_fixedTimeStep = fixedTimeStep;
m_localTime += timeStep;
if (m_localTime ≥ fixedTimeStep)
{
	numSimulationSubSteps = int(m_localTime / fixedTimeStep);
	m_localTime -= numSimulationSubSteps * fixedTimeStep;
}

…

// applies gravity to every object
applyGravity();

for (int i = 0; i < clampedSimulationSteps; i++)
{
	// actual update function
	internalSingleStepSimulation(fixedTimeStep);


	// interpolate position and velocities
	synchronizeMotionStates();
}
```

btDiscreteDynamicsWorld::internalSingleStepSimulation()

```
// apply gravity, predict motion (without taking into consideration collision or constraints etc.)
predictUnconstraintMotion(timeStep);

// continuous collision detection
createPredictiveContacts(timeStep);

// collision detection
performDiscreteCollisionDetection();

// performance: objects put into groups, which are disabled and enabled based on activity within the group
calculateSimulationIslands();

// solve contact and other joint constraints, update velocities and transforms
getSolverInfo().m_timeStep = timeStep;
solveConstraints(getSolverInfo());

// integrate transforms
integrateTransforms(timeStep);

// performance: ‘disables’ objects if inactive (at rest and not affected by forces) for too long
updateActivationState(timeStep);
```

btTransformUtil::integrateTransform

```
// btTransformUtil::integrateTransform()
// d = vt, adds the displacement to the current position
predictedTransform.setOrigin(curTrans.getOrigin() + linvel * timeStep);

// crazy rotation calculations that I don't understand using quaternions
// Exponential map: "Practical Parameterization of Rotations Using the Exponential Map", F. Sebastian Grassia
float fAngle2 = angvel.length2(), fAngle = 0;
if (fAngle2 > SIMD_EPSILON)
        fAngle = btSqrt(fAngle2);

//limit the angular motion
if (fAngle * timeStep > ANGULAR_MOTION_THRESHOLD)
        fAngle = ANGULAR_MOTION_THRESHOLD / timeStep;

// use Taylor's expansions of sync function
btVector3 axis;
if (fAngle < 0.001)
        axis = angvel * (0.5 * timeStep - (timeStep * timeStep * timeStep) * 0.020833333333 * fAngle * fAngle);
else
        axis = angvel * (btSin(0.5 * fAngle * timeStep) / fAngle);

btQuaternion dorn(axis.x(), axis.y(), axis.z(), btCos(fAngle * timeStep * 0.5));
btQuaternion orn0 = curTrans.getRotation();
btQuaternion predictedOrn = dorn * orn0; predictedOrn.safeNormalize();
if (predictedOrn.length2() > SIMD_EPSILON)
        predictedTransform.setRotation(predictedOrn);
else
        predictedTransform.setBasis(curTrans.getBasis());
```

Collision Detection

- Naive method: for each object check if it collides with all other objects. Slow! $O(n^2)$
- Split into two phases: broadphase and narrowphase
- Broadphase: “rough” collision detection, fast
- Bullet physics supports many collision shapes, all of which have specialized algorithms to detect whether a collision has occurred or not
- The algorithms tend to be slow, so they should be avoided unless necessary
- The broadphase collision phase creates a list of pairs of objects that may potentially collide
- The narrowphase collision phase goes through the list and uses the collision algorithms to check if the collision occurs or not

btDbvtBroadphase

- Implements a ‘dynamic bounding volume hierarchy’
- A bounding volume is a region of space that encapsulates the rigid bodies in the simulation
- Bounding volumes are created or merged based on the density of rigid bodies
- The advantage of this is that any rigid body can only potentially collide with another rigid body in the same bounding volume, which saves computation time
- Potential collisions are determined using axis aligned bounding boxes (AABB)
- The edges of the bounding boxes are parallel to the axes (hence ‘axis aligned’)
- Easy to calculate collision + fast, with no need to use specialized algorithms

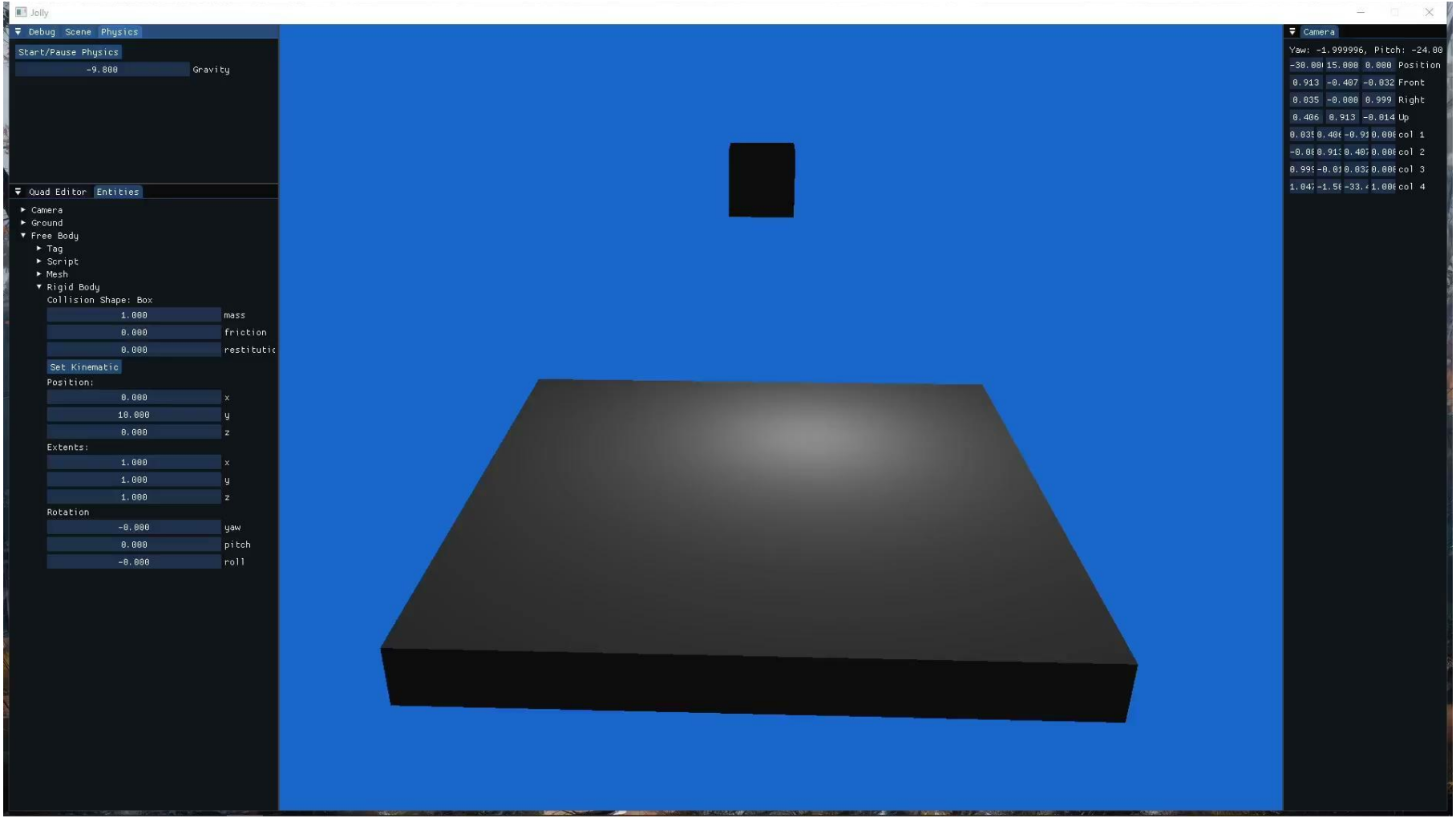
btSequentialImpulseConstraintSolver

- Updates velocities and position:

```
// btSequentialImpulseConstraintSolver::initSolverBody()
solverBody->m_worldTransform = rb->getWorldTransform();

// I don't know why inverse mass is a vector, possibly for mass distributions in the collision shapes?
solverBody->internalSetInvMass(btVector3(rb->getInvMass(), rb->getInvMass(), rb->getInvMass()) *
rb->getLinearFactor());
solverBody->m_originalBody = rb;
solverBody->m_angularFactor = rb->getAngularFactor();
solverBody->m_linearFactor = rb->getLinearFactor();
solverBody->m_linearVelocity = rb->getLinearVelocity();
solverBody->m_angularVelocity = rb->getAngularVelocity();

// definition of impulse: I = ft, rearranging: v = f/m * t
// newton's second law: a = f/m, so this is really v = a * t
solverBody->m_externalForceImpulse = rb->getTotalForce() * rb->getInvMass() * timeStep;

// torque, not investigated in the physics project
solverBody->m_externalTorqueImpulse = rb->getTotalTorque() * rb->getInvInertiaTensorWorld() * timeStep;
```

btSequentialImpulseConstraintSolver::writeBackBodies()

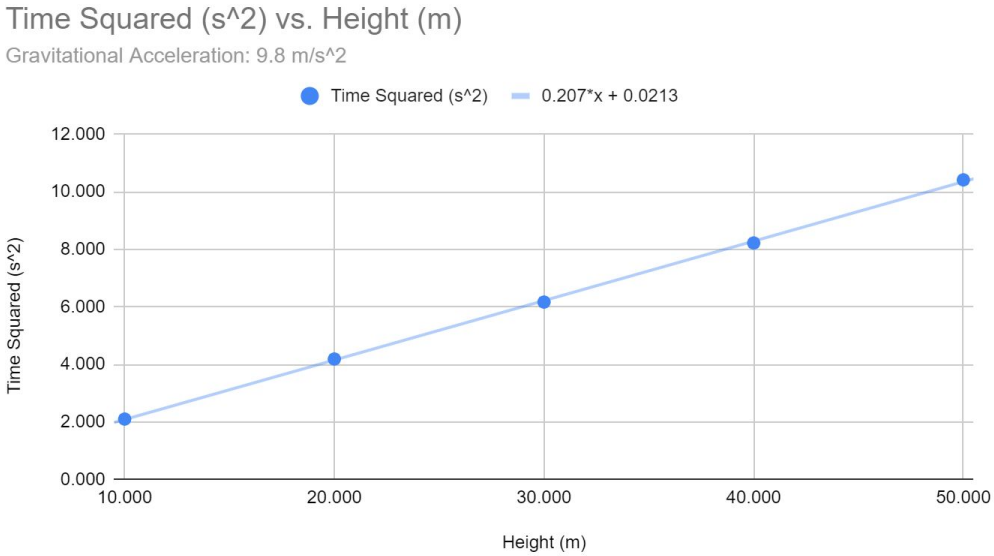
```
for (int i = iBegin; i < iEnd; i++)
{
	btRigidBody* body = m_tmpSolverBodyPool[i].m_originalBody;
	if (body)
	{
		// update the velocities and transforms
		m_tmpSolverBodyPool[i].writebackVelocityAndTransform(infoGlobal.m_timeStep, infoGlobal.m_splitImpulseTurnErp);

		// from previous slide: m_externalForceImpulse was calculated to be the delta velocity, here it is simply added
		// to the previous velocity
		m_tmpSolverBodyPool[i].m_originalBody->setLinearVelocity(
			m_tmpSolverBodyPool[i].m_linearVelocity +
			m_tmpSolverBodyPool[i].m_externalForceImpulse);

		// same idea as above
		m_tmpSolverBodyPool[i].m_originalBody->setAngularVelocity(
			m_tmpSolverBodyPool[i].m_angularVelocity +
			m_tmpSolverBodyPool[i].m_externalTorqueImpulse);

		m_tmpSolverBodyPool[i].m_originalBody->setWorldTransform(m_tmpSolverBodyPool[i].m_worldTransform);
		m_tmpSolverBodyPool[i].m_originalBody->setCompanionId(-1);
	}
}
```


Jolly
Debug Scene Physics
Start/Pause Physics
-9.800
Gravity
Quad Editor Entities
Camera
Ground
Free Body
Tag
Script
Mesh
Rigid Body
Collision Shape: Box
1.000
mass
0.000
friction
0.000
restitutic
Set Kinematic
Position:
0.000
x
10.000
y
0.000
z
Extents:
1.000
x
1.000
y
1.000
z
Rotation
-0.000
yaw
0.000
pitch
-0.000
roll
Camera
Yaw: -1.999996, Pitch: -24.00
-30.00 15.000 0.000 Position
0.913 -0.407 -0.032 Front
0.035 -0.000 0.999 Right
0.406 0.913 -0.014 Up
col 1
col 2
col 3
col 4

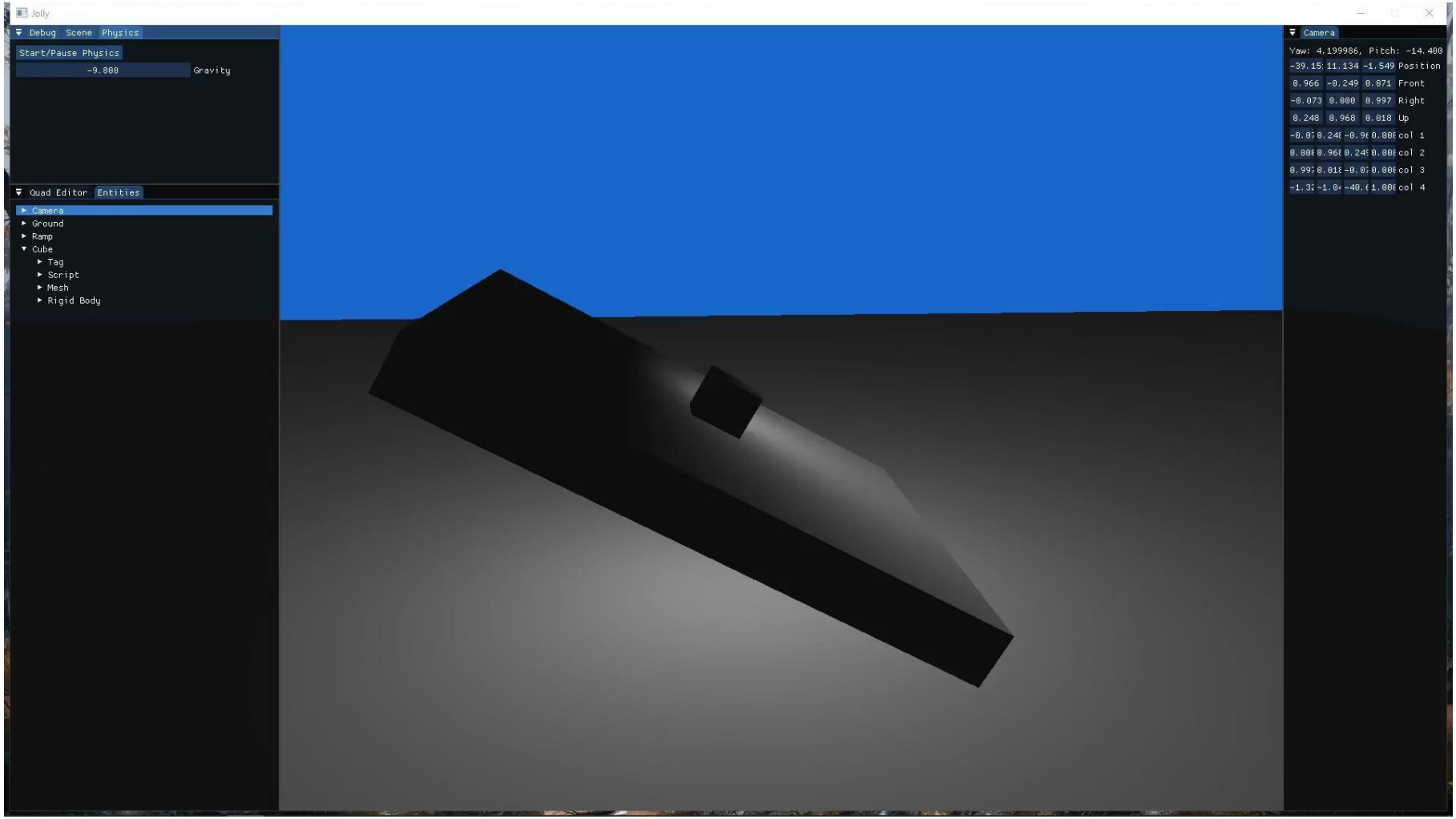
Gravity

Figure 1: Time Squared vs. Height

- Bullet physics was able to accurately simulate the motion of an object in free fall
- The raw data collected in the experiment shows some peculiar results, such as slight penetration
- The velocity of the object varied slightly when it was supposed to be at rest, likely due to correcting the penetration
- The simulation was found to improperly use the update frequencies to update positions, due to some fluctuations in the frequency



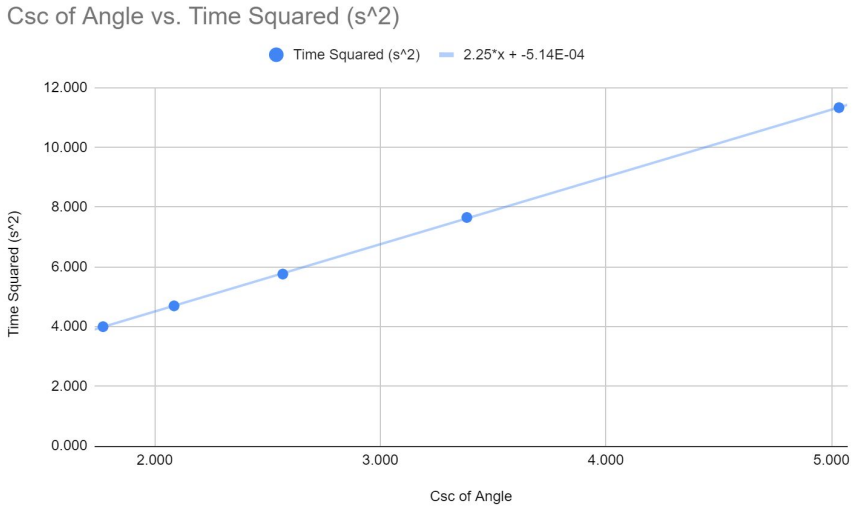
Slope represents 2 * inverse gravitational acceleration in s^2/m, the theoretical value is 0.204


Jolly
Debug
Scene
Physics
Start/Pause Physics
-9.800
Gravity
Quad Editor
Entities
Camera
Ground
Ramp
Cube
Tag
Script
Mesh
Rigid Body
Camera
Yaw: 4.199986, Pitch: -14.400
-39.15 11.134 -1.549 Position
0.966 -0.249 0.071 Front
-0.073 0.000 0.997 Right
0.248 0.968 0.018 Up
col 1
col 2
col 3
col 4

Force of Gravity on an Incline Plane

Figure 2: Csc of Angle vs. Time Squared

- Bullet physics was able to accurately simulate the motion of an object on a ramp



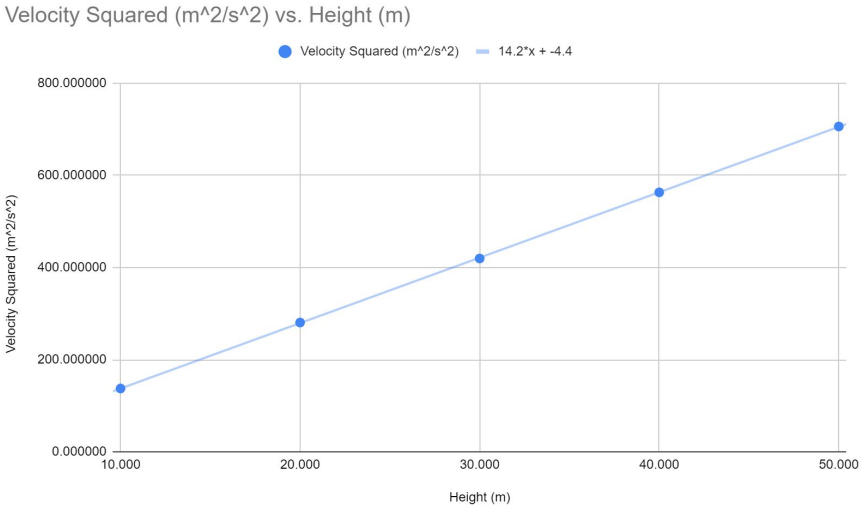
Slope represents 2 * distance over gravitational acceleration, the theoretical value is 2.25


Jolly
Debug
Scene
Physics
Start/Pause Physics
-9.800
Gravity
Quad Editor
Entities
Camera
Ground
Ramp
Cube
Camera
Yaw: -16.399948, Pitch: -15.4
-118.3
54.208
-33.88
Position
0.925
-0.267
-0.272
Front
0.282
-0.000
0.959
Right
0.256
0.964
-0.075
Up
0.28
0.25
-0.9
0.00
col 1
-0.0
0.96
0.26
0.00
col 2
0.95
-0.0
0.27
0.00
col 3
65.9
-24.
-114.
1.00
col 4

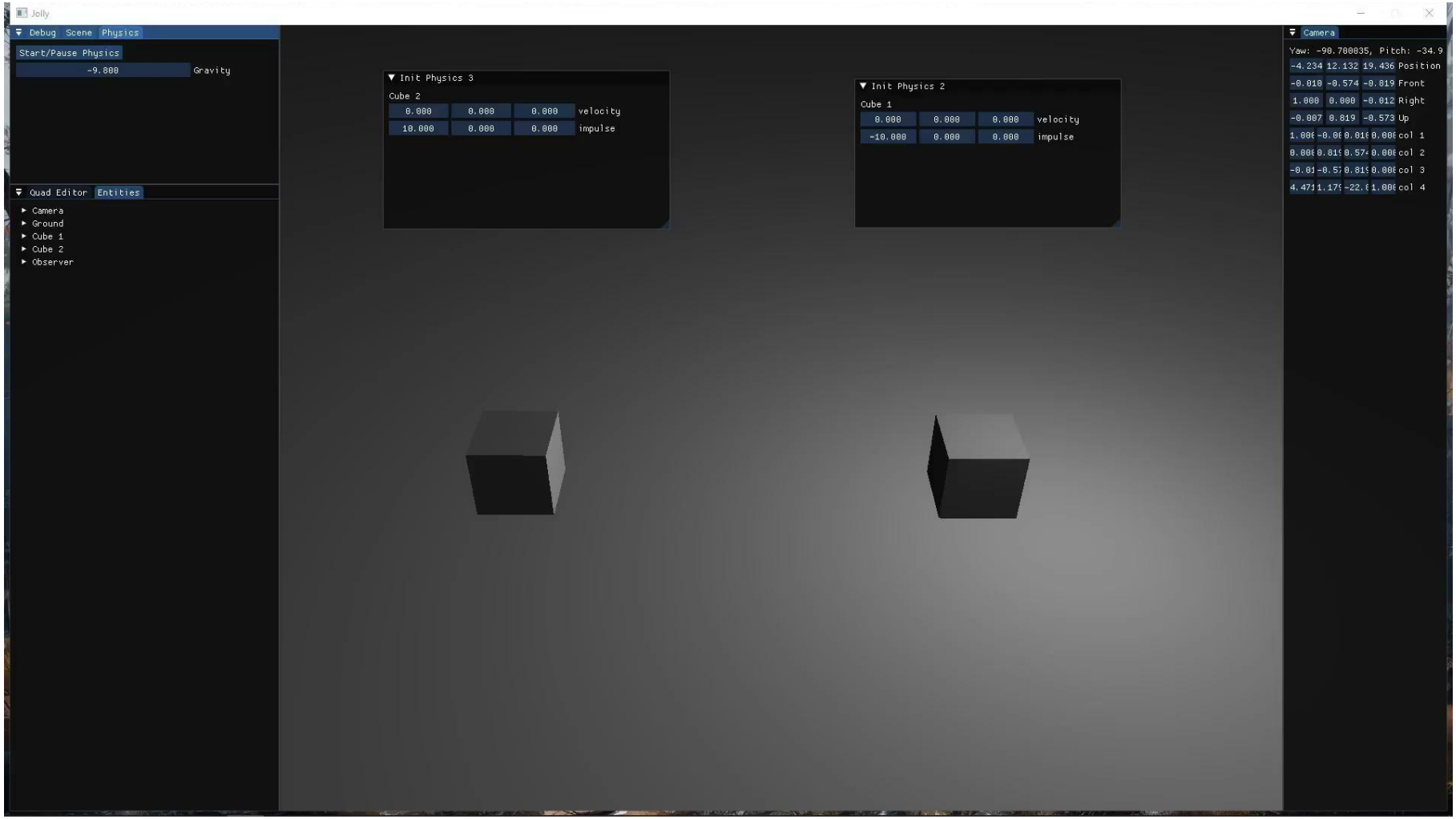
Mechanical Energy

Figure 3: Velocity Squared vs. Height

- Due to error in the experimental design, it is inconclusive whether the physics simulation properly conserved energy
- There was zero uncertainty in the experimental values, which is good for the reproducibility
- There is still some slight penetration, which results in a little bit of fluctuation in the vertical velocity component.



Slope represents 2 * gravitational acceleration, the theoretical value is 19.6

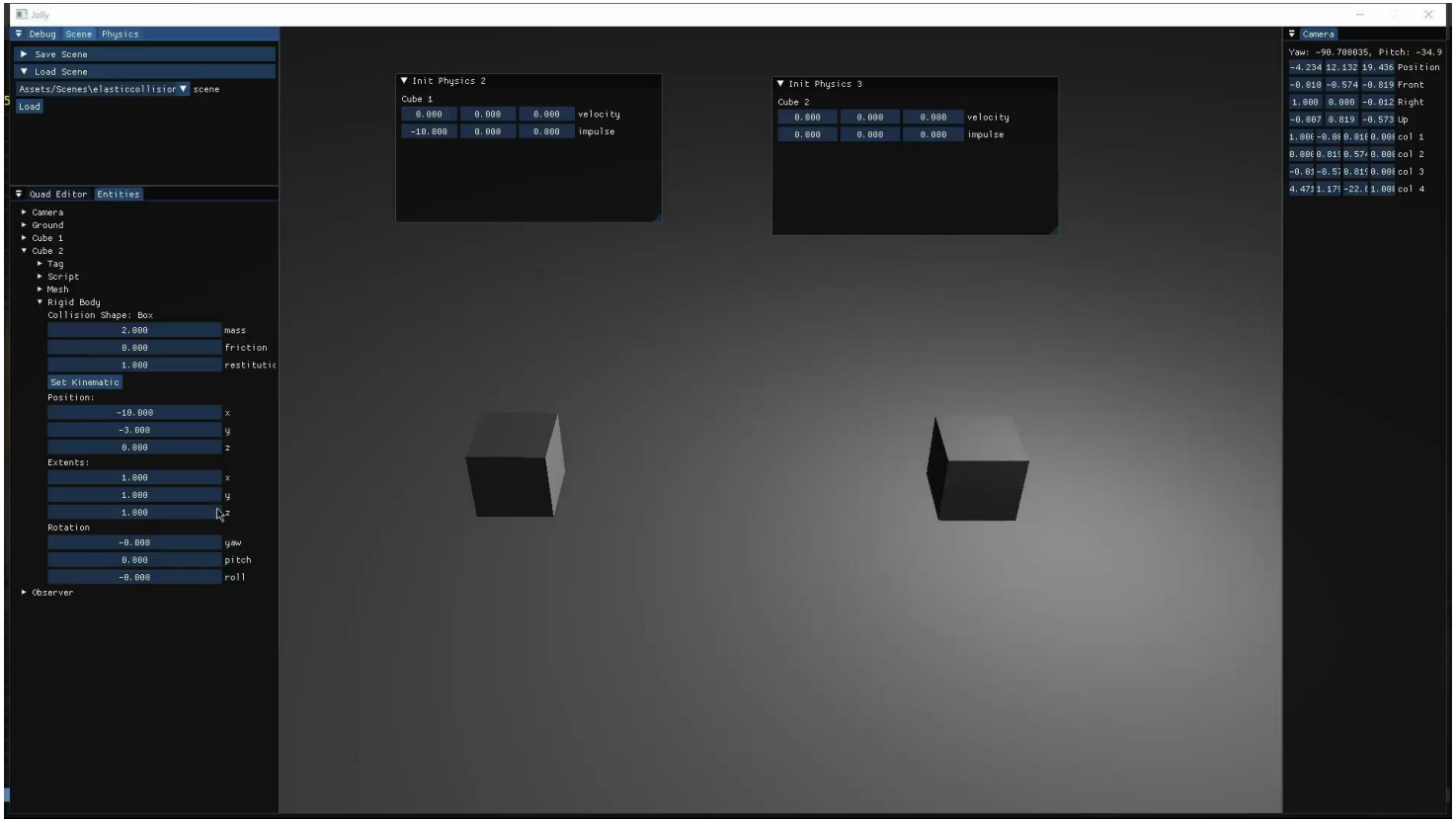

Jolly
Debug Scene Physics
Start/Pause Physics
-9.800
Gravity
Quad Editor Entities
Camera
Ground
Cube 1
Cube 2
Observer
Init Physics 3
Cube 2
0.000 0.000 0.000 velocity
10.000 0.000 0.000 impulse
Init Physics 2
Cube 1
0.000 0.000 0.000 velocity
-10.000 0.000 0.000 impulse
Camera
Yaw: -90.700035, Pitch: -34.9
-4.234 12.132 19.436 Position
-0.010 -0.574 -0.819 Front
1.000 0.000 -0.012 Right
-0.007 0.819 -0.573 Up
col 1
col 2
col 3
col 4

Elastic Collision

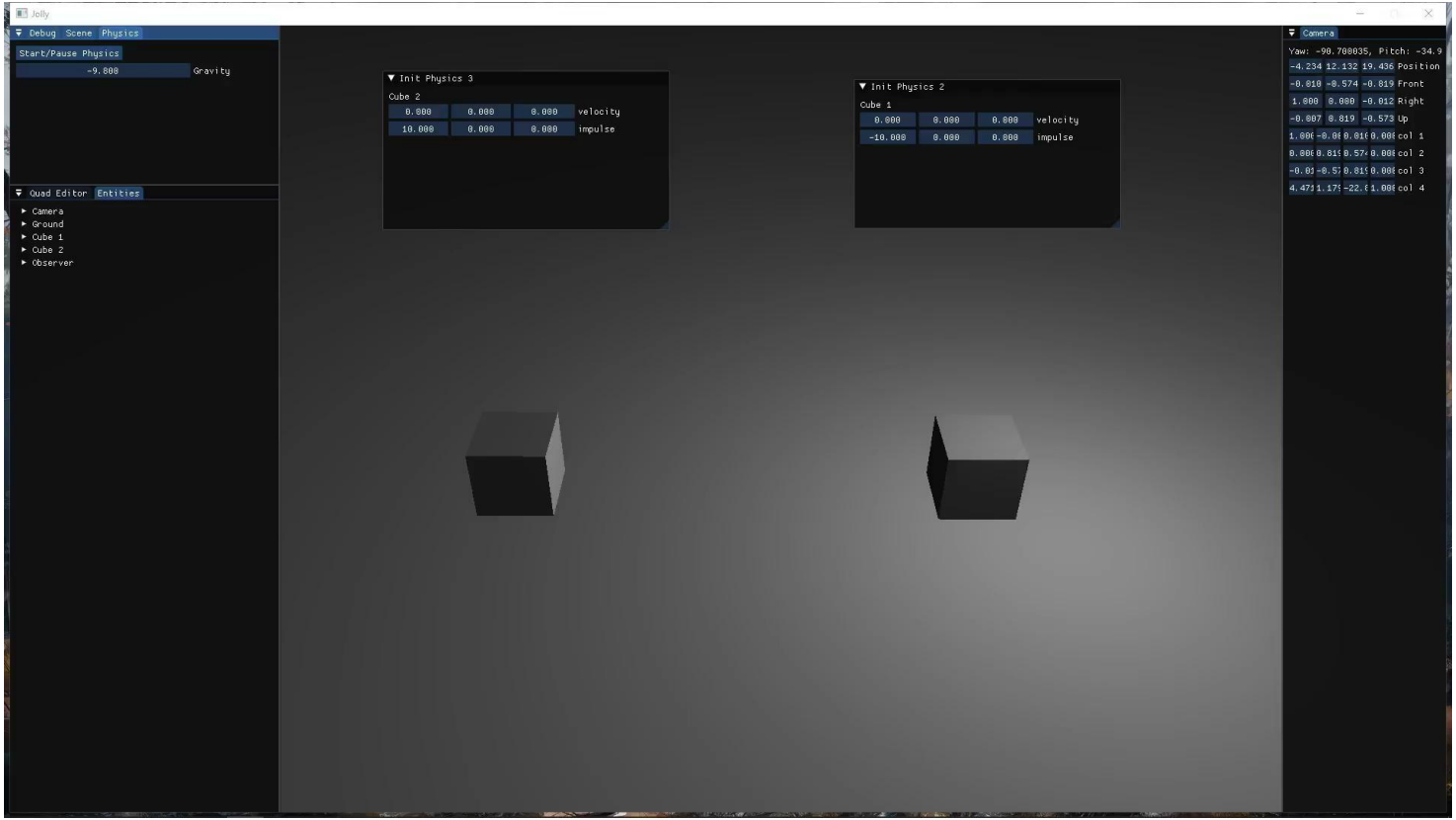
Table 1: Initial and Final Momentum and Initial and Final Energy

- The physics simulation was able to conserve momentum
- The physics simulation was unable to conserve kinetic energy in two test cases
- The bullet user guide recommends using inelastic collision over elastic collision, indicating that this area requires improvement

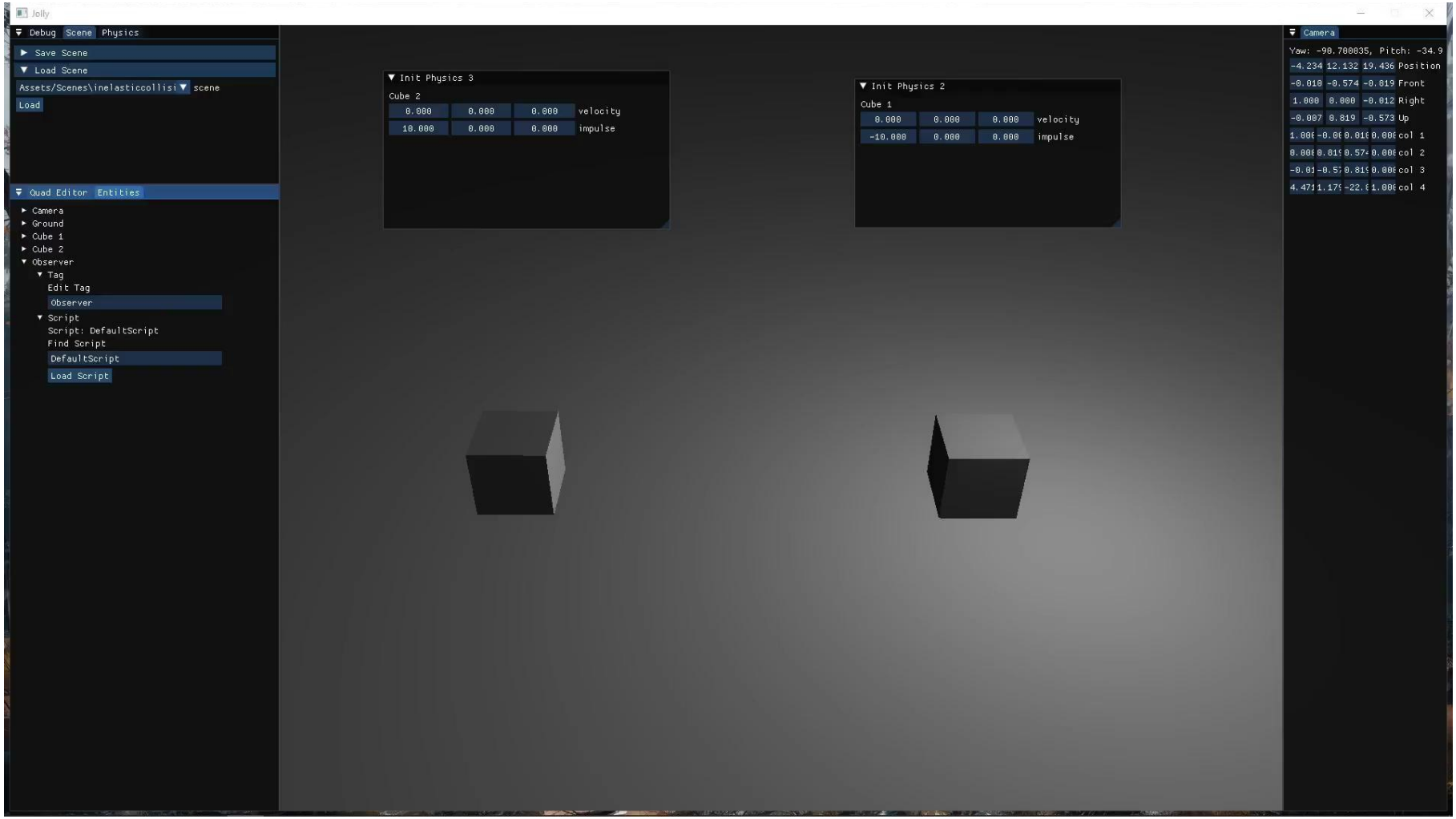
Initial Momentum (Ns)	Final Momentum (Ns)	Initial Energy (J)	Final Energy (J)
0	0	25	24.99314047
0	0	100	99.98576451
0	0.000001	75	37.10975214
-10	-10.000001	50	32.03576259
-20	-20	200	200.0179473



A significant amount of kinetic energy was lost. Somehow an upwards impulse was applied on the cubes which results in some rotation.



This result occurs because the cubes rotate just before the collision. Energy is lost, but the cubes also eventually lose their angular momentum and their original rotations are restored.

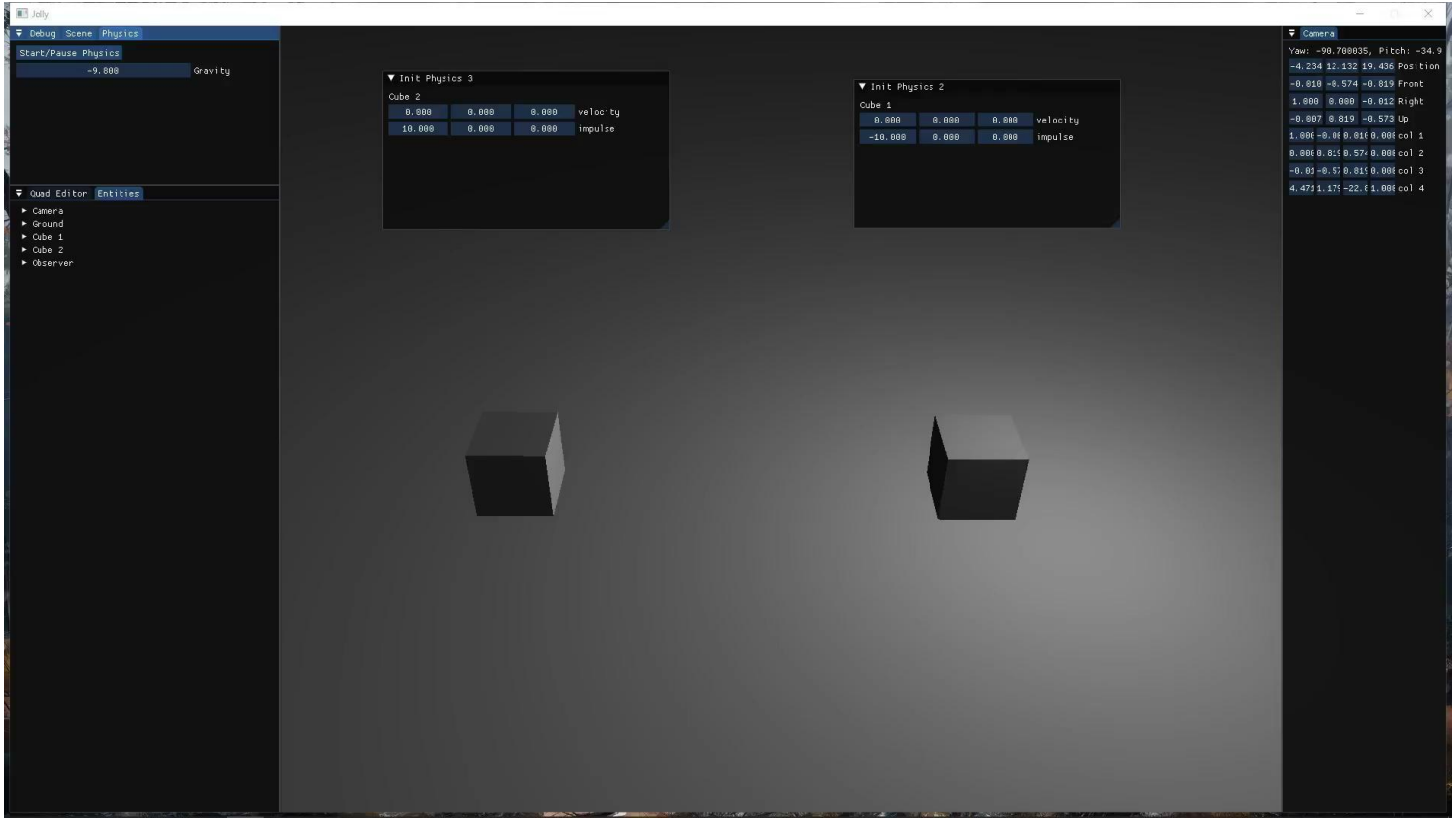

Jolly
Debug Scene Physics
Save Scene
Load Scene
Assets/Scenes\inelasticcollisi scene
Load
Quad Editor Entities
Camera
Ground
Cube 1
Cube 2
Observer
Tag
Edit Tag
Observer
Script
Script: DefaultScript
Find Script
DefaultScript
Load Script
Init Physics 3
Cube 2
0.000 0.000 0.000 velocity
10.000 0.000 0.000 impulse
Init Physics 2
Cube 1
0.000 0.000 0.000 velocity
-10.000 0.000 0.000 impulse
Camera
Yaw: -90.700035, Pitch: -34.9
-4.234 12.132 19.436 Position
-0.010 -0.574 -0.819 Front
1.000 0.000 -0.012 Right
-0.007 0.819 -0.573 Up
col 1
col 2
col 3
col 4

Inelastic Collision

Table 2: Initial and Final Momentum and Initial and Final Energy

- The physics simulation was able to conserve momentum fairly well
- Kinetic energy was not conserved as expected, more importantly: energy was not added to the system
- In an inelastic collision, it is expected that the objects stick together, however this was not observed in the simulation

Initial Momentum (Ns)	Final Momentum (Ns)	Initial Energy (J)	Final Energy (J)
0	0	100	0.000183743447
0	0.1053122	300	0.06099812158
-20	-19.999999	200	66.67951922
-15	-15.000001	206.25	37.56732212
15	15	112.5	37.53567702



The objects fail to “stick” together as expected. While their velocities are the same, one cube has a greater mass than the other, so momentum was not conserved.

Figure 4: Position and Velocity vs. Time

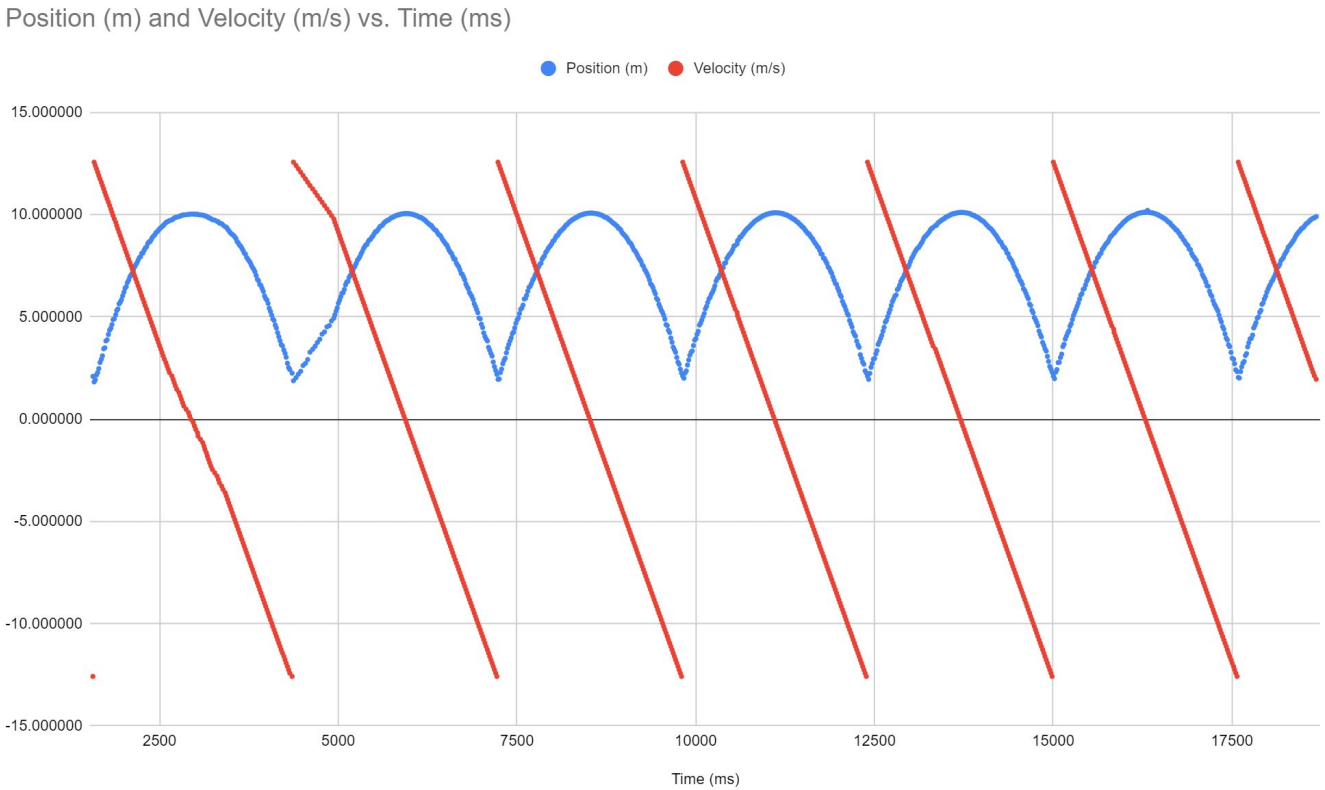
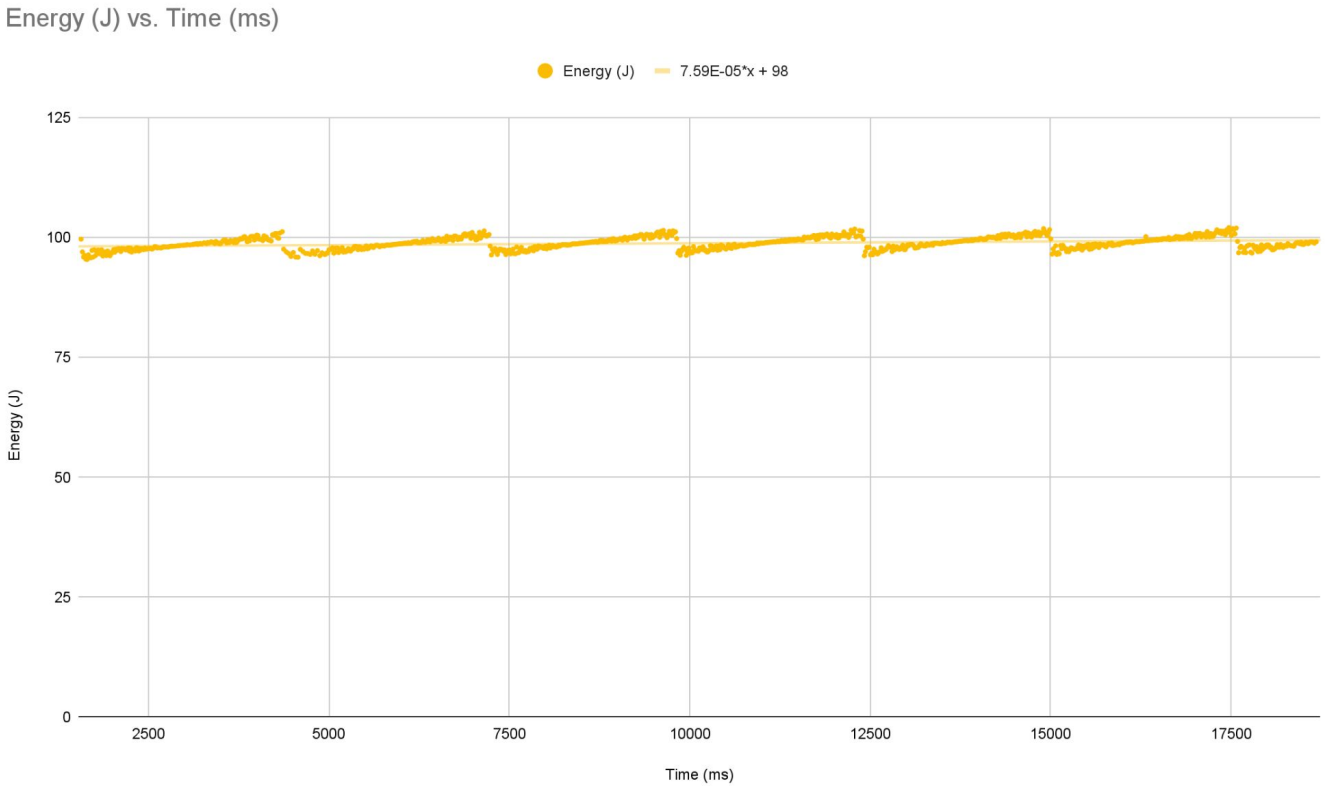
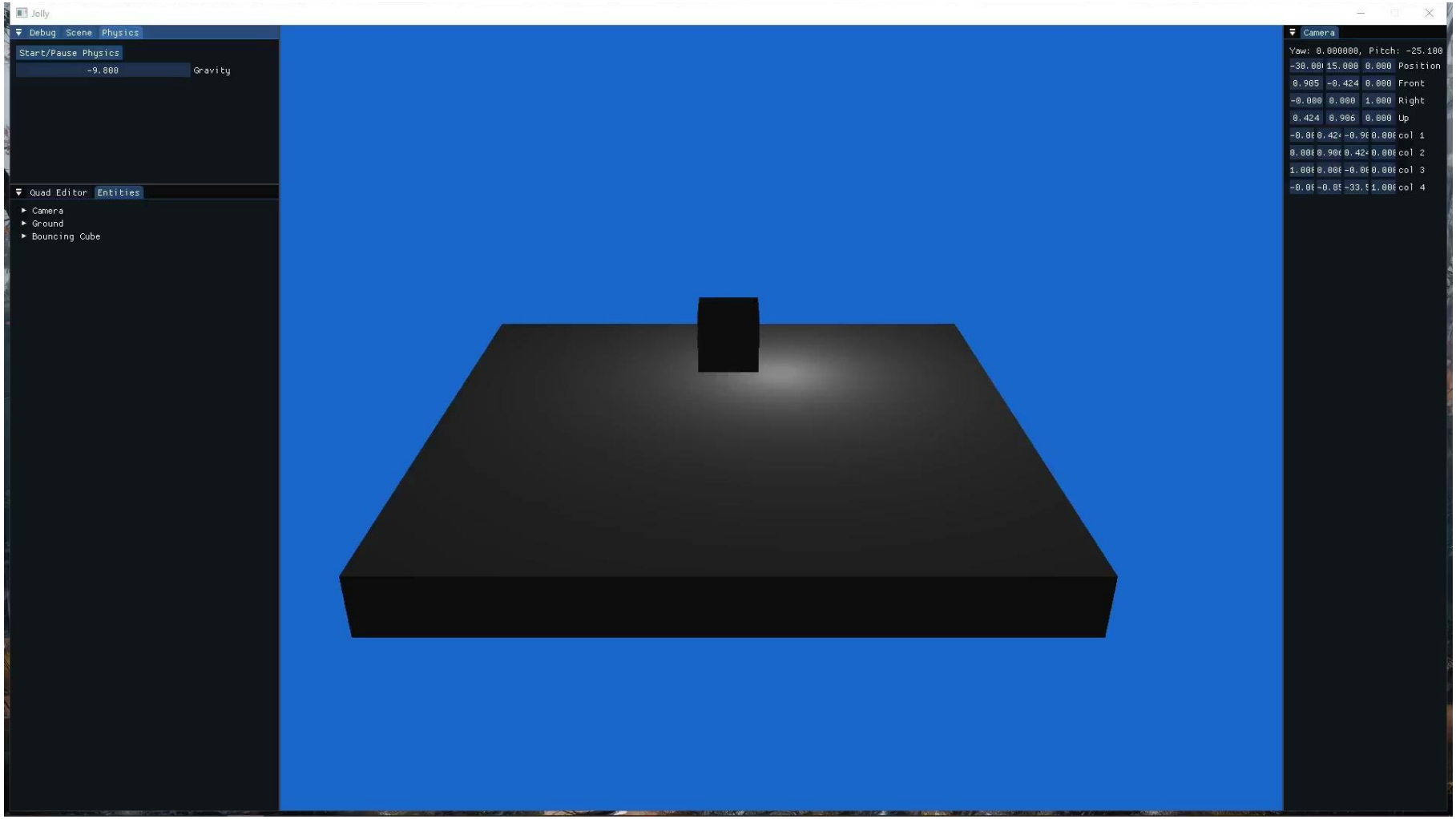


Figure 5: Energy vs. Time



Elastic Collision Stress Test

- Energy in the simulation is gained and lost cyclically
- Average energy increases over time, the cube reaches higher and higher heights each time
- Mechanical energy is gained as the object falls, and is lost due to the collisions
- After roughly 42 seconds the cube somehow gains angular momentum and the simulation becomes chaotic
- The physics simulation fails to accurately simulate the cyclic motion

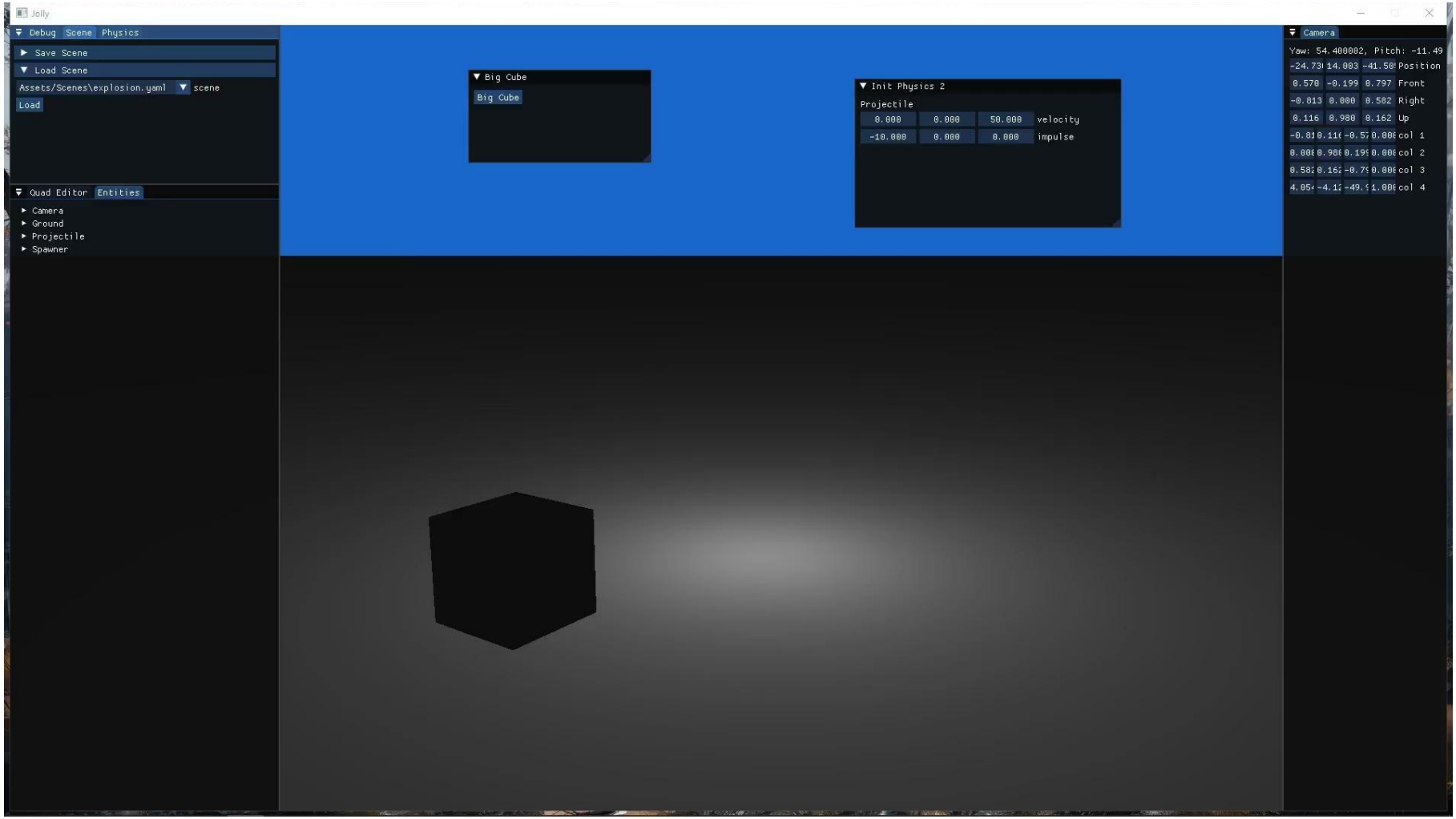

Jolly
Debug
Scene
Physics
Start/Pause Physics
-9.800
Gravity
Quad Editor
Entities
Camera
Ground
Bouncing Cube
Camera
Yaw: 0.000000, Pitch: -25.100
-30.000 15.000 0.000 Position
0.905 -0.424 0.000 Front
-0.000 0.000 1.000 Right
0.424 0.906 0.000 Up
col 1
col 2
col 3
col 4

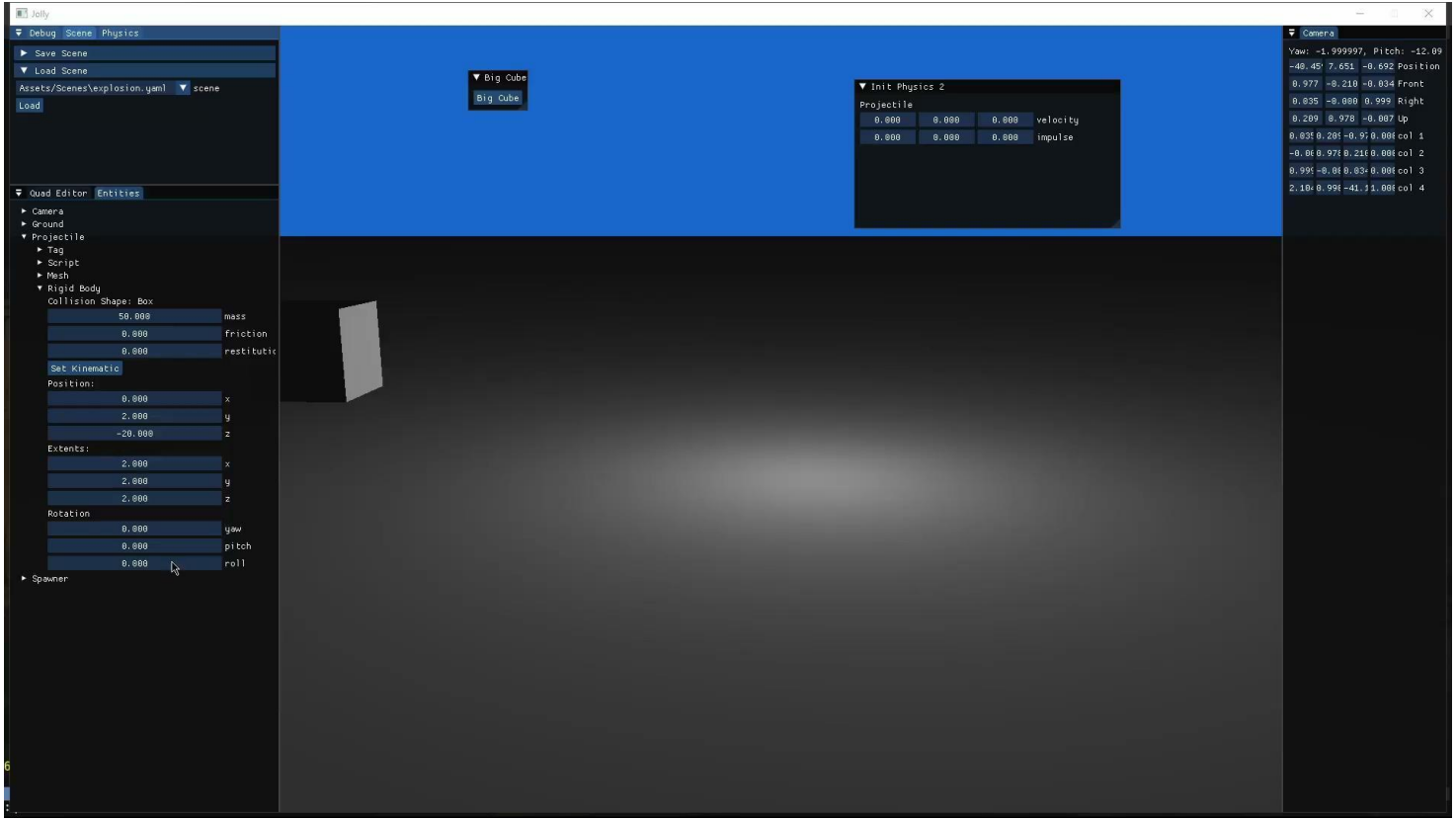
Inelastic Collision Stress Test

Table 3: Time and Momentum

- A cube with a mass of 50 kg and a velocity 50 m/s collides with 125 cubes arranged into a larger cube shape, with a total mass of 125 kg
- The physics simulation was able to conserve momentum

Time (ms)	Z-Momentum (Ns)	X-Momentum (Ns)
10966	2500.057257	0.054508
11066	2499.076544	0.025972
11166	2499.905953	-0.022007
11266	2500.053545	0.026968
11366	2499.905311	0.014252
11466	2500.015561	-0.020217
11566	2499.994381	-0.002033


Jolly
Debug
Scene
Physics
Save Scene
Load Scene
Assets/Scenes\explosion.yaml
scene
Load
Quad Editor
Entities
Camera
Ground
Projectile
Spawner
Big Cube
Big Cube
Init Physics 2
Projectile
0.000
0.000
50.000
velocity
-10.000
0.000
0.000
impulse
Camera
Yaw: 54.400002, Pitch: -11.49
-24.73
14.003
-41.50
Position
0.570
-0.199
0.797
Front
-0.813
0.000
0.582
Right
0.116
0.980
0.162
Up
col 1
col 2
col 3
col 4



Strange rotations occur during the simulation. As mentioned previously, the simulation did not update properly using the given timestep, resulting in the “slow-motion” effect.

Limitations

- Fails to conserve energy
- Slight penetration occurs during collision
- Bullet uses a “collision margin.” This is an offset between shapes so that interpenetration does not occur. May be noticeable on some collision shapes
- Objects do not “stick together” after an inelastic collision
- Bullet recommends using smaller masses, as larger masses may cause problems for the simulation
- Does not properly take into account delta times, which is important for a real time physics simulation
- Setting the friction of each object yields surprising results, as the frictional forces applied does not follow the expectation of $F_f = \mu F_n$

Real World Application

- Despite the lengthy list of limitations, Bullet remains popular for robotics training
- A paper published by a group of authors including Eric Coumans, developer of the Bullet physics library, explores the augmentation of the physics simulation using neural networks
- The technique seeks to close the “sim to real” gap by using the neural network to dynamically apply forces to account for unsimulated phenomena, such as air resistance and deformation from the stress of heavy loads

https://youtu.be/J4KylEbFM8I

Rapidly Advancing Field

- Only recently was a technique for simulating contact forces properly introduced
- The technique guarantees penetration-free simulation with zero parameter tuning
- Able to simulate the twisting of a nut and bolt, the sprocket and chain of a bike, pistons, and a complex locking box
- Comes at greater computational cost, with a timestep unacceptable for real time simulation
- The technique is unable to conserve energy, and restitution (elastic vs. inelastic) cannot be controlled.

Conclusion

As Two Minute Papers commonly remarks:

“The first law of papers says that research is a process. Do not look at where we are, look at where we will be two more papers down the line.”

With the pace and improvements to these simulations, it is inevitable that existing limitations will be overcome, as well as improvements to computational friendliness.

Works Cited

Coumans, Eric, et al. “bullet3.” *Github*, https://github.com/bulletphysics/bullet3.

Ferguson, Zachary. “Intersection-free Rigid Body Dynamics.” 2021, https://ipc-sim.github.io/rigid-ipc/.

Goodstein, David L. “Rigid Bodies.” *Britannica*, https://www.britannica.com/science/mechanics/Rigid-bodies.

Heiden, Eric, et al. “NeuralSim: Augmenting Differentiable Simulators with Neural Networks.” *arXiv.org*, 2021, https://arxiv.org/abs/2011.04217.

“The Tale of the Unscrewable Bolt.” *YouTube*, uploaded by Two Minute Papers, 21 Sep 2021, https://www.youtube.com/watch?v=CfJ074h9K8s.