Making Bullet deterministic **without** recreating everything!

Shahin Rabbani

Bullet Physics does not demonstrate deterministic behaviour by default. This is due to a number of reasons, most importantly making Bullet fast and stable. There are a lot of places where warm staring and cache variables are used to stay close to the stable simulated trajectory. Randomized methods are also often used to numerically solve the constraints that help them converge faster. While all these make the simulation stable and fast, a non-deterministic simulator is particularly bad for a type of research that involves saving and rewinding the simulation state. Many researchers simply opt for destroying and recreating everything (mostly dynamic world and constraint solver), but this is obviously not the least expensive approach. Things even get in the way of recreation if the rigid body pointers are passed around and spread throughout the code, which makes a safe dynamic world recreation very difficult.

In this document I show how a self-contained cache clean-up code in addition to a few adjustments in the simulator parameters can do the job.

Things to do:

1. Use fixed time step instead of variable time step when taking simulation step. This is the easiest yet very effective fix.
2. Avoid internal interpolation of the transformations when using fixed time step. Either make fixed and variable time steps the same with max internal sub step set to 2, or make them integer multiplicands and compute the maximum internal sub step to (variable_dt/fixed_dt) + 1.]
3. Make sure the following flags in `btSolverMode` in `btContactSolverInfo.h` is cleared:
    a. `SOLVER_RANDMIZE_ORDER`
    b. `SOLVER_USE_WARMSTARTING`

    This will make `btSequentialImpulseConstraintSolver` deterministic.
4. Use the clear cache code (available in the code section of this document). Place it before and after each simulation step. Also, include it in the physics reset function.
5. The only difference between reset() and the clear collision cache would be setting the local time variable in the solver to zero in reset() (`m_dynamicsWorld->m_localTime = 0;`). Usually this variable does not play a huge role. But due to the round off error, if fixed time step is used, for the very first time the local time gets very small, but negative, value, which does not make sense (it is a minor bug in Bullet that is %99.99 of time negligible). Because the local time is used to compute transformations, forcing it to be 0 helps always starting from the same state. Another option is to take one speculative simulation step at the very beginning to make the local time get some (safe) non-zero value.
6. btTransformUtil was not part of the clean-up, but it has some cache to predict the transformation. Good to know for the future.

7. Continuous collision detection did not cause any problem during the experimentation for determinism. But it always remain a place to keep an eye on as it uses predictive methods.
8. Load/Save control parameters from/to a file:
    a. Float should be fine for data transfer. But use max precision when loading/saving.
    b. Use hexa_float (or binary) for IO instead of readable text format in the decimal base. Yes… even rounding to the 6th decimal when converting from binary to decimal affects the simulation drastically, particularly when dealing with a character locomotion control that is one heck of a chaotic system!

Hunting places (usual suspects):

1. `btCollisionWorld`
2. `btPersistentManifold`
3. `btCompoundCompoundCollisionAlgorithm`
4. `btHashedOverlappingPairCache`
    a. `cleanProxyFromPairs`
    b. `processAllOverlappingPairs`
5. `btDbvtBroadphase`
6. `btNNCGConstraintSolver`
7. `btCollisionDispatcher`
    a. `->btCollisionPairCallback::getNearCallback`
    b. `dispatchAllCollisionPairs`
8. `btGImpactCollisionAlgorithm`
9. `btCollisionObject (m_broadphaseHandle)`
10. `btRigidBody::predictIntegratedTransform`

Useful sources and discussion:

- Step simulation parameters in Bullet
  https://stackoverflow.com/questions/22825391/stepsimulation-parameters-in-bullet-physics
  https://gafferongames.com/post/fix_your_timestep/
- Simulation Tick Callbacks
  http://www.bulletphysics.org/mediawiki-1.5.8/index.php/Simulation_Tick_Callbacks
- Multi-threading and determinism
  http://www.bulletphysics.org/Bullet/phpBB3/viewtopic.php?f=9&t=10232&p=35983&hilit=deterministic#p35983
- Bullet determinism
  http://www.bulletphysics.org/Bullet/phpBB3/viewtopic.php?f=9&t=11187&p=37704&hilit=deterministic#p37704
  http://bulletphysics.org/Bullet/phpBB3/viewtopic.php?t=7889
- Collision Detection and Callbacks in Bullet

http://www.bulletphysics.org/mediawiki-1.5.8/index.php?title=Collision_Detection_and_Physics_FAQ
http://www.bulletphysics.org/mediawiki-1.5.8/index.php/Collision_Callbacks_and_Triggers

- Persistent Manifold in Bullet
  http://www.bulletphysics.org/Bullet/phpBB3/viewtopic.php?f=4&t=8632
- Continuous Collision Detection (CCD)
  https://gamedev.stackexchange.com/questions/11961/how-can-i-enable-ccd-in-bullet-physics
  http://gamedevs.org/uploads/continuous-collision-detection-and-physics.pdf Erwin Coumans, Sony Computer Entertainment August 2005, Draft revision 5.
  http://www.panda3d.org/manual/index.php/Bullet_Continuous_Collision_Detection
  http://www.imperial.ac.uk/pls/portallive/docs/1/20673698.PDF Collision Overload: Reducing the Impact in Real-time Physics Final Report Ian Robert Ballantyne Supervisor: Tony Field Second Marker: Paul Kelly.

Added codes:

State snapshot/rewind

1. btRigidBody

```
inline void setTotalForce(const btVector3& totalForce)
{
	m_updateRevision++;
	m_totalForce = totalForce;
}

inline void setTotalTorque(const btVector3& totalTorque)
{
	m_updateRevision++;
	m_totalTorque = totalTorque;
}
```

2. dmRigidBody

```
DynamicState& dmRigidBody::getState()
{
	m_dynamicState.set(getLinearVelocity(),
	getAngularVelocity(),
	getTotalForce(),
	getTotalTorque(),
	getWorldTransform());
	return m_dynamicState;
};

void dmRigidBody::setState(const DynamicState& dynamicState)
{
	if (isStaticObject())	return;

	setWorldTransform(dynamicState.m_transform);
	setLinearVelocity(dynamicState.m_linVel);
```

```
        setAngularVelocity(dynamicState.m_angVel);

        //Extra care
        btDefaultMotionState* myMotionState =
        (btDefaultMotionState*)getMotionState();

        myMotionState->m_graphicsWorldTrans = dynamicState.m_transform;
        setCenterOfMassTransform(dynamicState.m_transform);
        setInterpolationWorldTransform(dynamicState.m_transform);
        forceActivationState(ACTIVE_TAG);
        activate();
        setActivationState(DISABLE_DEACTIVATION);
        setDeactivationTime(btScalar(2e7));

        clearForces();
        // setTotalForce(m_snapshotState.m_force);
        // setTotalTorque(m_snapshotState.m_torque);
}

void dmRigidBody::snapshotState()
{
        m_snapshotState = getState();
}

void dmRigidBody::rewindState()
{
        if (isStaticObject())      return;

        setWorldTransform(m_snapshotState.m_transform);
        setLinearVelocity(m_snapshotState.m_linVel);
        setAngularVelocity(m_snapshotState.m_angVel);

        clearForces();
        //setTotalForce(m_snapshotState.m_force);
        //setTotalTorque(m_snapshotState.m_torque);
}
```

Clearing collision cache (right before and after taking a simulation step)

```
void PhysicsMediator::clearCollisionCache()
{
        if (m_dynamicsWorld)
        {
                int numObjects = m_dynamicsWorld->getNumCollisionObjects();

                ///create a copy of the array, not a reference!
                                btCollisionObjectArray copyArray =
                m_dynamicsWorld->getCollisionObjectArray();

                for (int i = 0; i < numObjects; i++)
                {
                        btCollisionObject* colObj = copyArray[i];
                        btRigidBody* body = btRigidBody::upcast(colObj);
                        if (body)
                        {
```

```
                        //removed cached contact points (this is not
                        necessary if all objects have been removed from
                        the dynamics world)
                        m_dynamicsWorld->getBroadphase()-
                        >getOverlappingPairCache()-
                        >cleanProxyFromPairs(colObj-
                        >getBroadphaseHandle(), m_dynamicsWorld-
                        >getDispatcher());

                        //remove the broadphase from the collision
                        object
                        btBroadphaseProxy* bp = colObj-
                        >getBroadphaseHandle();
                        if (bp)
                        {
                               // only clear the cached algorithms
                               m_dynamicsWorld->getBroadphase()-
                               >getOverlappingPairCache()-
                               >cleanProxyFromPairs(bp, m_dynamicsWorld-
                               >getDispatcher());
                               m_dynamicsWorld->getBroadphase()-
                               >destroyProxy(bp, m_dynamicsWorld-
                               >getDispatcher());
                               colObj->setBroadphaseHandle(0);
                        }
                }
        }


/*************************************************************************/
      /* Recreating broadphase (Not computatinally very expensive. The only
      thing that is better to recreate rather than get cleaned up!)

/*************************************************************************/
              vector<btPersistentManifold*> manifoldArray;
              int numManifolds = m_dynamicsWorld->getDispatcher()-
              >getNumManifolds();
              for (int i = 0; i < numManifolds; i++)
              {
                     btPersistentManifold* contactManifold =
                     m_dynamicsWorld->getDispatcher()-
                     >getManifoldByIndexInternal(i);
                     manifoldArray.push_back(contactManifold);
              }

              for (int i = 0; i < manifoldArray.size(); i++)
              {
                     m_dynamicsWorld->getDispatcher()-
                     >releaseManifold(manifoldArray[i]);
              }

              m_dynamicsWorld->getBroadphase()->~btBroadphaseInterface();
              btDbvtBroadphase* broadphase = new btDbvtBroadphase();
              m_dynamicsWorld->setBroadphase(broadphase);

/*************************************************************************/
      /* End of Recreating stuff
```

```
/*****************************************************************************/

		for (int i = 0; i < numObjects; i++)
		{
			btCollisionObject* colObj = copyArray[i];
			btRigidBody* body = btRigidBody::upcast(colObj);
			if (body)
			{
				//Add the broadphase back
				//calculate new AABB
				btTransform trans = colObj->getWorldTransform();

				btVector3	minAabb;
				btVector3	maxAabb;
				colObj->getCollisionShape()->getAabb(trans,
				minAabb, maxAabb);

				int type = colObj->getCollisionShape()-
				>getShapeType();

				dmRigidBody* dmBody = dmRigidBody::upcast(body);

				short int collisionFilterGroup = dmBody-
				>m_cInfo.m_collisionGroup;//
				btBroadphaseProxy::DefaultFilter
				short int collisionFilterMask = dmBody-
				>m_cInfo.m_collidesWith;///collision mask

				colObj->setBroadphaseHandle(m_dynamicsWorld-
				>getBroadphase()->createProxy(
					minAabb,
					maxAabb,
					type,
					colObj,
					collisionFilterGroup,
					collisionFilterMask,
					m_dynamicsWorld->getDispatcher(), 0
				));
			}
		}
		///reset some internal cached data in the broad phase
		m_dynamicsWorld->getBroadphase()->resetPool(m_dynamicsWorld-
		>getDispatcher());
		m_dynamicsWorld->getConstraintSolver()->reset();
	}
}
```