Documentation Note:

How osgBullet Handles Center of Mass (COM) and Scale

The first problem—For collision shapes such as spheres, boxes, and cylinders, Bullet uses their center as the center of mass (COM). If you attempt to use one of these collision shapes to represent an OSG object that is not modeled with the origin at its center, the rigid body dynamics will appear to misbehave because the COM is not at the center of the body.

The second problem —For collision shapes such as convex hulls and triangle meshes (TMs), Bullet treats the local collision shape origin as the COM. If you have an OSG object that isn't modeled with the COM at the origin and you make such a collision shape directly from that data, the rigid body dynamics will appear to misbehave because the COM is not at the center of the body.

The larger problem —Sometimes you just need an easy way to specify an arbitrary COM (for example, when modeling a loaded die).

An even worse problem —Bullet transforms consist of a normalized basis and an origin, which means Bullet collision shapes can't be scaled by a btTransform. The collision shape must be created in the scaled space. If your application has a scale transform above a rigid body subgraph, you need a mechanism to synchronize an unscaled OSG subgraph with a scaled collision shape.

The good news —osgBullet has facilities for minimizing these issues and supporting arbitrary COM and scale for any collision shape. This document decribes the osgBullet interfaces that support this functionality.

CollisionShapes.cpp/.h

This low-level set of routines converts between OSG scene graphs and Bullet collision shapes. Functions convert in either direction, from OSG to Bullet or from Bullet to OSG.

When converting from OSG to a simple sphere, box, or cylinder, only the OSG extents, not their spatial location, are considered. The resulting parametric collision shape is always centered on the origin for purposes of computing COM. It is the calling code's responsibility to handle arbitrary COM using btCompoundShape, if desired.

When converting from OSG to a convex hull or TM, osgBullet uses the OSG vertices literally to create the collision shape, so the collision shape COM corresponds to the OSG model origin. Before calling into these conversion routines, calling code is responsible for transforming the source OSG model so that its origin corresponds to the desired COM. Alternately, the calling code can handle arbitrary COM using btCoumpoundShapefor sphere, box, and cylinder above.

When converting from a Bullet collision shape to an OSG representation, the calling code can specify an arbitrary btTransform to position and orient the resulting OSG Geometry. This is primarily to support converting btCompoundShape collision shapes to OSG.

So what about scaling?
When converting from OSG to a sphere, box, or cylinder, the OSG subgraph can contain a scale, and the resulting collision shape will reflect that scaling.

However, the routines for converting from OSG to a convex hull or TM, as well as the routines for converting from any Bullet shape to OSG, do not support scale. The calling code is responsible for scaling any input data before converting that data with these functions.

OSGToCollada

High-level OSGToCollada uses the routines in CollisionShapes.cpp/.h to convert from an OSG scene graph to a Bullet representation, alternatively writing the result as a Collada .dae file. Calling code can convert to Bullet, then retrieve the resulting btRigidBody.

OSGToCollada has two methods to support COM, setCenterOfMass() and setAutoComputeCenterOfMass().

setCenterOfMass()
Use the setCenterOfMass() method to specify the desired COM in the scene graph's local coordinate system. The coordinate units of the COM must be in the same coordinate space as the subgraph passed into setSceneGraph().

setCenterOfMass() automatically disables auto compute (see next section).

setAutoComputeCenterOfMass()
If set to true, OSGToCollada uses the center of the scene graph bounding sphere as the COM. If set to false, OSGToCollada uses the scene graph local origin, or the COM specified in setCenterOfMass(). The default value is true, but it is automatically disabled by a call to setCenterOfMass().

Great, but what's happening internally?
To create a collision shape centered on an arbitrary COM, OSGToCollada creates a MatrixTransform that translates to the arbitrary COM. It sets this MatrixTransform as the scene graph parent and uses it as the new root node.

Next, it flattens all transforms (including the new COM MatrixTransform it just created). After this is complete, the model's geometry is in an absolute coordinate system with the arbitrary COM at the origin, and all transforms in the scene graph are identity.

What happens next depends on the value of setOverall(), as well as the specified result collision shape type.

If overall is true and the collision shape type is a convex hull or TM, the scene graph is ready for a direct translation to Bullet. OSGToCollade essentially calls the appropriate low-level function in CollisionShape.cpp to create the collision shape.

If overall is true and the collision shape type is a sphere, box, or cylinder, OSGToCollada uses the low-level CollisionShapes.cpp function to perform the conversion, and it adds the resulting collision shape as a child shape to a btCompoundShape. It sets the child transform to the specified COM offset, so the child shape center is positioned correctly with respect to the COM.

If overall is false, OSGToCollada always builds a btCompoundShape. (This document is currently incomplete. Information on this subject will be added in the future.)

So what about scaling?
The calling application specifies the local scale using setScale(). When OSGToCollada flattens transforms, it uses the following formula (in OSG's row-major, pre-multiply notation):

$$v' = v\, M\, (-Tc)\, S$$

where v is an xyz vertex.
v' is the transformed vertex.
M is the subgraph's accumulated model transform.
$-Tc$ is a translation by the inverse COM offset (either specified or computed automatically).
S is the specified local scaling factor.

Note that the application must specify the COM in local (unscaled) coordinates. If an object has COM at $x = 3$ and is scaled by 0.5, then the application must tell OSGToCollada that the COM is $x = 3$, not $x = 1.5$. After scaling, the effective COM will be $x = 1.5$.

MotionState

In the process of creating the collision shape and rigid body, OSGToCollada alters the scene graph. As a result, the typical usage case for OSGToCollada is as an offline process (creating .dae files), or the calling code typically makes a deep copy of the scene graph to pass to OSGToCollada, which it discards after the conversion to a Bullet collision shape.

In both of these use cases, the application is left with the problem of syncing a scene graph not centered on the COM with a COM-centered collision shape. Bullet must be informed of the difference in these locations, so it can position the collision shape and rigid body correctly. osgBullet must subtract this difference from the Bullet transform before it can drive the actual scene graph model.

The MotionState class is responsible for reconciling both the COM and the scaling difference. It is essentially a linear algebra problem.

Given: Consider the initial scene graph consisting of a hierarchy of transformations and a subgraph to be used as a rigid body. The scene graph is expressed algebraically as

object *M | S [P]*

where *M* is the local model transform of the object to be treated as a rigid body.
| is the boundary between the rigid body and the owning scene graph.
S is the parent scale factor.
[P] is the parent transform excluding scale.

Note that an application typically inserts an osgBullet::AbsoluteModelTransform node at the boundary | between the rigid body and the owning scene graph, and Bullet drives the transformation of the OSG visual representation using the AbsoluteModelTransform node. Alternatively, the application can reparent the rigid body subgraph to a MatrixTransform attached to the scene graph root.

Using the scene graph above as input, osgBullet must create a collision shape that is scaled and translated by the inverse COM

shape = object *M (-Tc) S*

In order for Bullet to perform a physics simulation using this collision shape, the initial Bullet world transform for the rigid body is given by

Tsc [P]

where *Tsc* is a translation by the scaled COM offset (computed as a component-wise multiplication of the *xyz* scale vector *S* with the *xyz* COM translation *Tc*).
[P] is the orthonormalized parent transform (excluding scale).

Note that the COM offset is scaled. As a thought proof, imagine an object with a COM offset of 3 and a local scale of 2. OSGToCollada creates a collision shape with an inverse translation of -6 units (-*Tsc*). To position this scaled collision shape, the object must be translated by 3 * 2 or 6 (*Tsc*).

MotionState supports this initial world transform by accepting parent transform, COM, and scale values from the calling application, and combining them into its _transform member variable. When the simulation starts, Bullet queries the MotionState for the initial world transformation, and MotionState returns the transform *Tsc [P]*.

Finally, given the initial transformation of *Tsc [P]*, what is the formula for transforming the OSG rigid body subgraph that is both unscaled and untranslated by the COM? This new formula must produce the same transformation as the original scene graph

object *M | S [P]* == object *M X (Tsc [P])*

or

S [P] == X Tsc [P]

The solution for the transformation *X* requires little algebra and is practically given by intuition

X = S (-Tsc)

You can check this solution using substitution

S [P] == X Tsc [P]
S [P] == (*S (-Tsc)) Tsc [P]*
S [P] == S (-Tsc Tsc) [P]
S [P] == S [P]

This means if we originally transformed the OSG rigid body with its local model transform *M* using the formula *S [P]*, then the MotionState class can transform it to the same position, scale, and orientation using the formula (*S -Tsc) (Tsc [P])*. As a result, when MotionState receives a new world transform from Bullet, it concatenates (*S -Tsc*) with the incoming world transform (*Tsc [P]*), and sets the result in the OSG rigid body parent Transform node (either a MatrixTransform or an AbsoluteModelTransform).

Note that the sole reason for the complexity is the fact that the OSG rigid body subgraph does not have the scale baked in, and does not have its origin set at the COM. If you could remodel the object to satisfy these criteria, the problem would be simplifed as follows:

The original scene graph hierarchy expressed algebraically is
object *M* | *P*

The collision shape is a duplicate of the local object subgraph:
shape = object *M*

And the initial Bullet world transform is
P

In this simple case, MotionState simply uses *P* to transform both the collision shape and the OSG rigid body subgraph. Unfortunately, application developers frequently are restricted from remodeling geometric data, which is the source of the issue addressed by osgBullet in this document.

A Less-Detailed Summary of MotionState
At initialization time, the application specifies the initial parent transform of the rigid body using setParentTransform(). Note that this method performs an orthonormalization on the incoming matrix before converting it to a btTransform. The application also sets the scale and COM using setScale() and setCOM(). These three routines provide MotionState with the information it needs to compute the initial transformation for the

collision shape, *Tsc [P]*. When the Bullet simulation starts, Bullet queries the MotionState for the initial world transformation, and MotionState returns *Tsc [P]*.

After a simulation step, Bullet passes the new world transform to MotionState. MotionState premultiplies this transform with *S (-Tsc)*, and passes the result to the OSG rigid body subgraph parent Transform node.

As a result, the OSG rigid body subgraph and the collision shape are kept in sync entirely by the initial parent transform, scale, and COM paramaters passed to the MotionState class.

Conclusion

Bullet's requirements for absolute scaling and origin-COM collision shapes create a significant problem for application developers who cannot remodel their geometric data for use in physics simulation. osgBullet's OSGToCollada and MotionState classes provide a solution for this problem.