Project 4: Rigid Body Piñata

Due: Mar 16, 2018, 11:55PM

In this project, you will develop a rigid body simulator which is able to compute the rigid body motion and detect and handle collisions. To demonstrate your simulator, you will simulate the process of beating up a piñata with two pieces of jawbreaker candy made of rigid body. The user can hit the piñata from different directions with keyboard commands and see the rigid bodies inside move around, colliding with the piñata and with each other. When the piñata takes enough beating, it breaks and the rigid bodies fall out. The skeleton code provides you a box-like piñata hanging in the air, an interface to use the collision detector provided by DART, and the code for simulating linear motion of a rigid body. The motion of the piñata (moving and breaking) is simulated by DART. Your job is to compute the angular motion of the rigid bodies correctly and handle colliding contact based on the information provided by the collision detector.

Requirements:

1. Simulate rigid bodies in two different shapes: sphere and cube. Use a quaternion to represent orientation of a rigid body. The code for linear motion is provided in the skeleton code, but the angular motion is not.
2. Implement "colliding contact" between a rigid body and the piñata and between rigid bodies themselves. You can assume that the piñata has infinite mass and inertia.

About the collision detector:

The default collision detector provided for this project is a very basic one. It can only detect collisions for spheres and boxes. If you want to use more sophisticated collision detectors to handle arbitrary meshes, you will need to download and install Bullet collision detector. Contact the TA or the instructor if you want to use different collision detectors.

The collision detector provided here will return an array of contacts at every time step. Here is a list of APIs you might need.

```
int nContacts = mCollisionDetector->getNumContacts();
// Get the number of contacts at the current time step
```

```
Vector3d point = mCollisionDetector->getContact(0).point;
// Get the world coordinate of the first contact point at the current time step
```

```
Vector3d normal = mCollisionDetector->getContact(2).normal;
// Get the normal vector of the third contact point at the current time step. The vector is expressed in the world space.
```

```
RigidBody* A = mCollisionDetector->getContact(1).rb1;
// Get the pointer to the rigid body A involved in the second contact. If A is a null pointer, that means it is the pinata. Otherwise, it will return a non-null pointer.
```

```
RigidBody* B = mCollisionDetector->getContact(1).rb2;
// Get the pointer to the rigid body B involved in the second contact. If B is a null pointer, that means it is the pinata. Otherwise, it will return a non-null pointer.
```

```
Vector3d pVelocity = mCollisionDetector->getContact(0).pinataVelocity;
```

```
// Get the velocity of the colliding point on the pinata in the world space. If neither rb1 nor rb2
is the pinata, it will return (0, 0, 0).
```

About quaternions:
Eigen library provides the data type of Quaternion and many useful operators. For example:

```
// Create an identify quaternion
Eigen::Quaterniond q1;
q1.setIdentity();

// Convert a quaternion to a rotation matrix
Eigen::Matrix3d rotMatrix = q.toRotationMatrix();

// Normalize a quaternion
q.normalize();

// Access the scalar part and the vector part of the quaternion
double w = q.w();
Eigen::Vector3d v = q.vec();

// Add two quaternions
Eigen::Quaterniond sum;
sum.w() = q1.w() + q2.w();
sum.vec() = q1.vec() + q2.vec();

// Multiply a scalar with a quaternion
Eigen::Quaterniond scaledQ;
scaledQ.w() = 2.0 * q.w();
scaledQ.vec() = 2.0 * w.vec();
```

Skeleton code:
You will use the skeleton code provided in DART to complete this project.
Step 1:
Install DART following the bonus project instructions (If you have not done so).

Step 2:
Install the app directory and the piñata.skel file included in the .zip file that is part of this assignment as described in the readme file included in the .zip file

Step 3:
Rebuild DART, including the new app, following the directions at the above link.

UI control provided by the skeleton code:
space bar: simulation on/off
'p': playback/stop
'[' and ']': play one frame backward and forward
'v': visualization of contact on/off
‘1-4’: Hit the piñata from different directions
Left click: rotate camera
Right click: pan camera

Shift + Left click: zoom camera

Extra points:

1. Add rigid bodies on the fly using a keyboard command. When the user hits ‘a’, spawn a rigid body in a random location inside of the pinata. Make sure that the initial location is collision-free (2 points).
2. Implement a collision handling routine to deal with resting contacts. This can be very challenging (3 points).