

SensablePlugin

SensablePlugin

From SOFAWiki

Contents

- 1 Sensable Plugin
 - 1.1 Installing the Sensable Plugin
 - 1.2 Using the Plugin
 - 1.2.1 Method 1
 - 1.2.1.1 Example Scene
 - 1.3 NewOmniDriver Attributes
 - 1.4 EnslavementForceFeedback Attributes
 - 1.5 Scenes at different scales

Sensable Plugin

Installing the Sensable Plugin

- Download the OpenHaptics SDK (<http://www.sensable.com/products-openhaptics-toolkit.htm>)
- The OpenHaptics SDK comes with both the Phantom Drivers and the OpenHaptics libraries. First install the drivers, then the SDK.
- Run the Phantom Test program that came with the SDK, and use it to check that the Omni is working. It is also recommended that you use it to calibrate your Omni.
- In your Sofa CMake configuration, select **SOFA-PLUGIN_SENSABLE** and configure. If you installed OpenHaptics in the default location, CMake should find all the libraries and directories automatically.

If CMake doesn't find them for you, it will return an error and you can set them manually. The HD and HL libraries are found in OpenHaptics/Academic/3.1/lib/"your system's subdirectories", while the HDU library is found in OpenHaptics/Academic/3.1/utilities/lib/"your system's subdirectories".

- Once all the variables are set, you can Configure, Generate and compile as usual.

Using the Plugin

Method 1

One method of using the Omni in a scene is to control a rigid object directly with the Omni, and compute the force feedback based on the penetration of that rigid object with any other object in the scene.

The Sensable Plugin contains two important components.

The **NewOmniDriver** interfaces with the Omni device, getting the tool's location and sending it the appropriate force feedback.

The **EnslavementForceFeedback** computes the force feedback values, based on the collision detection that Sofa already does in a scene.

Example Scene

In the examples directory of the Sensable Plugin, there are a number of example scenes. We will examine **SimpleBox.scn**.

This scene has two main objects. A long curved tool is controlled by the Omni, and a simple box serves as something for us to feel.

Taking a look at the scene in more detail, we see our collision pipeline:

```
<CollisionPipeline name="pipeline" depth="6" verbose="0"/>
<BruteForceDetection name="detection" />
<CollisionResponse name="response" response="default" />
<MinProximityIntersection name="proximity" alarmDistance="0.8" contactDistance="0.5" />
```

This is what does the collision detection for the scene.

Next, we see our **NewOmniDriver**:

```
<NewOmniDriver name="Omni Driver" listening="true" tags="Omni" forceScale="0.5" scale="500" permanent="
```

We will discuss all the attributes in more detail later, but the important one to note now is the **tags** attribute. The NewOmniDriver needs to find the MechanicalObject that it will control, and the ForceFeedback that will calculate the feedback for it. It will do this by looking for the components that have the same tag as in, in this case **Omni**.

Next we see the **Instrument** node. First we see the MechanicalObject:

```
<MechanicalObject template="Rigid" name="instrumentState" tags="Omni" />
```

Here, the template type is important. The NewOmniDriver is looking for a MechanicalObject with the template **Rigid**. Also, we see that it has the matching tag, so that the NewOmniDriver knows that this is the MechanicalObject that it should be controlling.

The UniformMass object simply gives the MechanicalObject some mass.

In the **VisualModel** node, the **OglModel** loads the mesh that is used to visualize the instrument.

```
<OglModel template="ExtVec3f" name="InstrumentVisualModel" fileMesh="data/mesh/dental_instrument.obj"
```

Note the translation and rotation attributes. We will come back to them later. The material attribute just gives the instrument a different look than the box.

The **RigidMapping** connects the MechanicalObject we saw earlier with the OglModel. This keeps the visualization of the instrument in sync with the movement of the Omni.

In the **Collision Model** node, there are a number of important details.

```
<MechanicalObject template="Vec3d" name="Particle" position="0 0 0" />
<Point name="ParticleModel" contactStiffness="2" />
<RigidMapping template="MechanicalMapping<Rigid,Vec3d>" name="MM->CM mapping" object1="instrumen
  <EnslavementForceFeedback name="forcefeedback" tags="Omni" collisionModel1="@ParticleModel" collision
```

First, we see the MechanicalObject named **Particle**, and the attribute **position** with the value "0 0 0". Here we are making defining a single point that will be used for our collisions. This point represents the tip of the Omni. We place it at "0 0 0" because this lines it up with the Omni properly.

We then give our particle a Point collision model.

The **RigidMapping**, like the earlier one, keeps are particle in sync with the movement of the Omni. The visual model is also in sync with the motion, and we want to tip of the instrument to correspond with our particle, so that the tip of the instrument is where the collision occurs. That is what the translation and rotation attribute in **InstrumentVisualModel** are for. They line the visual model of the tool up with the particle. We put the translation in the visual model instead of in the particle so that the tip lines up with the movement and rotation of the Omni.

The **EnslavementForceFeedback** listens for any contact that our Particle Model makes with any other model in the scene, then calculates the force feedback for the Omni accordingly. We will look into all the attributes in detail later, but for now notice that again the tag attribute matches that of the NewOmniDriver.

The **Box** node also has a visual model and a collision model. In this case, the same mesh is used in both models. Here, the **DistanceGrid** is the collision model for the box.

NewOmniDriver Attributes

- **listening** - When true, the NewOmniDriver will listen to the ForceFeedback for the computation of the feedback.
- **tags** - The NewOmniDriver will look for a MechanicalObject with template Rigid with a matching tag, and a ForceFeedback with a matching tag.
- **forceScale** - scales the force feedback given to the Omni. This attribute will involve a balance in your scene. If it is too high, you will feel a lot of oscillations and vibrations. Too low and you will be able to push through your surfaces easily.
- **scale** - scales the motion from the Omni. This changes how much motion on the physical Omni it takes to move the tool in the scene.
- **permanent** - True if the force feedback will be applied permanently.

EnslavementForceFeedback Attributes

- **tags** - should correspond with the tags attribute in the NewOmniDriver
- **collisionModel1** - The collision model that the Omni controls. The EnslavementForceFeedback will then gather the collision information for every collision this model has in the scene
- **collisionModel2** - If you are only interested in the collision between collisionModel1 and one other collision model in the scene, you can specify that other model here. The EnslavementForceFeedback will then only gather the collision information for collisions involving both models. **Note:** if you do not want to specify another model, it is important that you specify the attribute as **collisionModel2=""**.
- **relativeStiffness** - To reduce oscillations when contact is made with a surface, the force applied when the instrument is found to be inside the other object is higher than when it is outside. The ratio of these forces is specified by the relativeStiffness. This value will also involve a balance in your scene.

If it is too high, you will feel a lot of oscillations and vibrations. Too low and you will be able to push through your surfaces easily.

- **attractionDistance** - To reduce oscillation when contact is made with a surface, as the instrument gets very close to making contact, it is slightly attracted to the contact point. Once it reaches the contact point, the force feedback will push away from the object, to prevent the instrument from entering. The distance from contact at which this attraction starts is given with attractionDistance.
- **normalsPointOut** - In order for the force feedback to be applied in the appropriate direction, the EnslavementForceField must know if the normals of the other objects in the scene point towards the inside of the object or towards the outside. If they point towards the outside, normalsPointOut should be true, and false if the normals point towards the inside. Note: All the objects (other than the one being controlled by the Omni) need to have their normals pointing in the same direction.
- **contactScale** - This scales the strength of the force feedback depending on the size of the objects in your scene. Larger objects will require a larger contactScale in order for the force feedback to be felt.

Scenes at different scales

The forces computed in your scene will vary depending on the scale of the objects in your scene. For example, the box in SimpleBox.scn is 10x10x10, while the box in SimpleBoxLarge.scn is scaled to be 10 times bigger than that. At larger and smaller scales, a number of your attributes will need to become larger and smaller as well. The attributes that are sensitive to scale are:

- MinProximityIntersection: alarmDistance
- MinProximityIntersection: contactDistance
- NewOmniDriver: scale
- EnslavementForceFeedback: attractionDistance
- EnslavementForceFeedback: contactScale

Look at the differences between these attributes in SimpleBox.scn and SimpleBoxLarge.scn to learn more.

Retrieved from "<https://wiki.sofa-framework.org/wiki/SensablePlugin>"