

OgreOde Walking Character

From Ogre Wiki

Creating a walking character using OgreOde

Original version by SuperMegaMau

Note: This tutorial includes code and algorithms based on the author's knowledge and may be incorrect or less accurate, so please correct all the problems you find. And ... sorry my english, I'm not very good at it yet :)

Contents

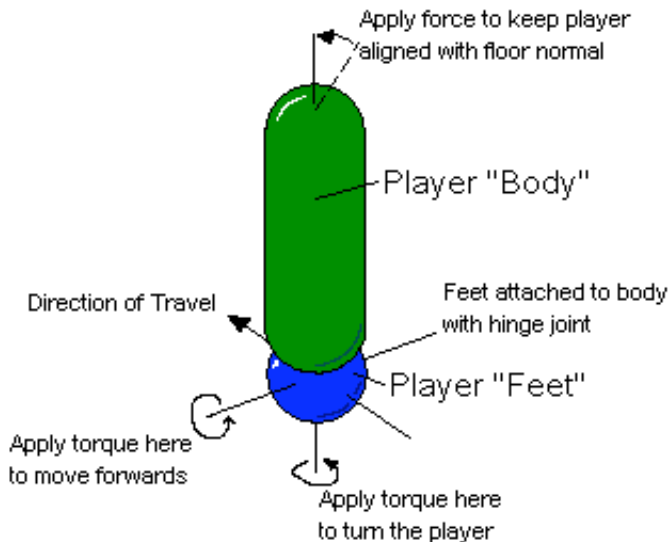
- 1 Introduction
- 2 Creating the physics model
 - 2.1 Creating the character
 - 2.2 Getting AABB
 - 2.3 Create a new space
 - 2.4 Create the Sphere (feet)
 - 2.5 Create the Capsule (torso)
 - 2.6 Create the Joint
- 3 Moving the Character
 - 3.1 Moving forward and backward
 - 3.2 Rotating left and right
- 4 Getting the Character up
- 5 Known issues

Introduction

I believe I'm not the first one to ask himself how to create a moving character using OgreOde. After searching the forum and wiki, I realize that there are other forum members that find this info useful. The tutorial explains how to create a moving character that walks through the terrain and through other meshes (houses, trees, etc).

Creating the physics model

So, I will follow the scheme found at Monster's page to represent the character:



I will assume you know Ogre basics specially related to SceneNodes, meshes and AlignedBoxes, and you got OgreOde working with the terrain SceneManager.

Creating the character

First you have to create a SceneNode to hold your character (mesh), in this case I will use the ninja mesh included in the examples. Just create two SceneNodes and attach one to the other. I'll explain why this two nodes later...

```
Entity* ninja = mSceneMgr->createEntity("ninja","ninja.mesh");
SceneNode* ninjaNode = mSceneMgr->getRootSceneNode()->createChildSceneNode("ninja");
SceneNode* modelNode = ninjaNode->createChildSceneNode("ninja_model");
modelNode->attachObject(ninja);
ninjaNode->setScale(0.05,0.05,0.05);
```

As you probably notice, the ninjaNode is scaled to a very small size, this is because if we consider a very large mesh, we get a very slow physics simulation (don't really know why...). Then again if we use a very small mesh, we realize that sometimes half the character passes trough the terrain because of ODE accuracy due to performance.

Getting AABB

Now we get the AxisAlignedBox to align our mesh at the right position later. With the following code we also determine the size of the sphere we are going to use to represent the character's feet.

```
AxisAlignedBox aab = modelNode->getAttachedObject("ninja")->getBoundingBox();
Ogre::Vector3 min = aab.getMinimum()*ninjaNode->getScale();
Ogre::Vector3 max = aab.getMaximum()*ninjaNode->getScale();
Ogre::Vector3 center = aab.getCenter()*ninjaNode->getScale();
Ogre::Vector3 size(fabs(max.x-min.x),fabs(max.y-min.y),fabs(max.z-min.z));
float radius = (size.x>size.z)?size.z/2.0f:size.x/2.0f;
```

Create a new space

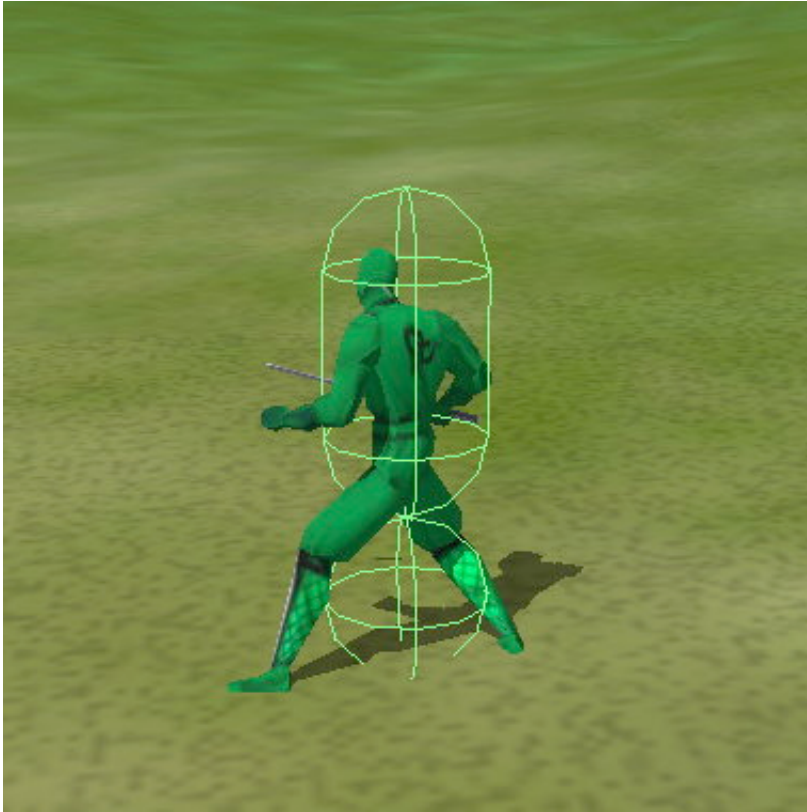
We need to create a new space for the character to live in, and disable the internal collisions

```
OgreOde::SimpleSpace* dollSpace = new OgreOde::SimpleSpace(_world->getDefaultSpace());  
dollSpace->setInternalCollisions(false);
```

Create the Sphere (feet)

So, at this point we have two SceneNodes, the "ninjaNode" and the "modelNode". The ninjaNode is the node that represents your character, and the modelNode is where you really attach the mesh. This approach is due to the fact that you can't modify the OgreOde::Body position, so we create the body using the ninjaNode, then to get it where we want to, we set the position of the mesh relative to the ninjaNode and OgreOde::Body. Using this, method, we always get the right relation between the mesh and the body position.





The left shot is from the 1 node method and the right one is from the 2 node mode. As you can notice, in the left image, the character floats by the terrain at a distance of half the feet sphere. This is because the center of the mesh is defined be at this feet, and the is `OgreOde::body`, automatically set to this point. So we need the second node to force the character mesh to be at the right position, like the right image shows.

The following code creates a sphere representing the feet of our character. "radius" is the radius of the feet sphere. We need a Sphere geometry and a Transformation Geometry to translate our sphere to the desired position.

```

OgreOde::Body* dollFeetBody = new OgreOde::Body("feet");
dollFeetBody->setMass(OgreOde::SphereMass(70*2.5,radius));
OgreOde::SphereGeometry* feetGeom = new OgreOde::SphereGeometry(radius);
OgreOde::TransformGeometry* feetTrans = new OgreOde::TransformGeometry(dollSpace);
modelNode->translate(Vector3(0,-radius/ninjaNode->getScale().y,0));
feetTrans->setBody(dollFeetBody);
feetTrans->setEncapsulatedGeometry(feetGeom);
ninjaNode->attachObject(dollFeetBody);

```

Create the Capsule (torso)

For the upper part of the character, we use a capsule (caped cylinder).

```

OgreOde::Body* dollTorsoBody = new OgreOde::Body("torso");
dollTorsoBody->setMass(OgreOde::CapsuleMass(70*2.5,radius,Vector3::UNIT_Y,radius));
dollTorsoBody->setAffectedByGravity(false);
dollTorsoBody->setDamping(0,50000);
OgreOde::TransformGeometry* torsoTrans = new OgreOde::TransformGeometry(dollSpace);
OgreOde::CapsuleGeometry* torsoGeom = new OgreOde::CapsuleGeometry(radius,size.y-4*radius,dollSpace);

```

```
torsoGeom->setPosition(Ogre::Vector3(0,size.y-((size.y-4*radius)/2+2*radius),0)); //can't find a good w
torsoGeom->setOrientation(Quaternion(Degree(90),Vector3::UNIT_X));
torsoTrans->setBody(dollTorsoBody);
torsoTrans->setEncapsulatedGeometry(torsoGeom);
ninjaNode->attachObject(dollTorsoBody);
```

The above geometries are placed at the same space as the feet geometries, so we can disable internal collisions. Set the angular damping to a relatively high value (each case is different). we must disable the gravity for this part of the character, to keep him from fall over.

Create the Joint

The only thing left to do is to connect the two bodies together by a hinge joint. A simple representation of a hinge joint in this case a the front wheel of a bicycle.

```
OgreOde::HingeJoint* joint = new OgreOde::HingeJoint();
joint->attach(dollTorsoBody,dollFeetBody);
joint->setAxis(Ogre::Vector3::UNIT_X); //set the rotation axis
```

Note: Don't forget then to keep track of all bodies and joints so you can get them later. You can use ogre stacks or make your own.

Moving the Character

To rotate and move your character you can use different ways, I decided to make the movement by setting the body orientation instead of apply forces and torques to change this direction.

Moving forward and backward

The following code can be executed when you press a key, like KC_UP. Now you need to get your torso body, get it from your stack or hashTable. and get its orientation. I use:

```
OgreOde::Body* torso = torsoBodies->getObject("ninja");
Quaternion q = torso->getOrientation();
```

Then apply an angular velocity to the feet sphere like this:

```
OgreOde::Body* feet = feetBodies->getObject("ninja");
feet->wake();
feet->setAngularVelocity(q*Ogre::Vector3(10*cos(1),0,10*sin(1)));
```

10 is the angular velocity we use, and it must be multiplied for the trigonometrical functions in order to get the

correct relation between them so the velocity is applied in the right direction.

Rotating left and right

This code must be executed when you press a key as well, KC_RIGHT for example.

```
OgreOde::Body* torso = torsoBodies->getObject("ninja");
Quaternion q1 = torso->getOrientation();
Quaternion q2(Degree(-4),Ogre::Vector3::UNIT_Y);
torso->setOrientation(q1*q2);
```

Using Degree(-4) the character rotates to the right, using a positive number the character rotates left. As you probably notice, I always set and get orientations from the torso bodies. I didn't thought much about it, but don't think that is any problem if you get this orientations from the feet body.

Note: When you release ke movement key, you must apply an angular and linear velocity of 0 so the player stops moving, or else the feet sphere continues to roll affected by gravity.

```
feetbody->setAngularVelocity(Vector3(0,0,0));
feetBody->setLinearVelocity(Vector3(0,feetBody->getLinearVelocity().y,0));
```

Getting the Character up

At last, we need to be sure that our character don't fall over, so we need to reset his vertical orientation once in a while:

```
OgreOde::Body* torso = torsoBodies->getObject("ninja");
Quaternion q = torso->getOrientation();

Vector3 x = q.xAxis();
Vector3 y = q.yAxis();
Vector3 z = q.zAxis();

torso->wake();
torso->setOrientation(Quaternion(x,Vector3::UNIT_Y,z));
```

Known issues

Part of this code is not as much elegant as I like it to be, and because of this (or not) it has its flaws. The way I reset the vertical orientation sometimes produces strange behavior. Although I set the angularVelocity to 0, the character never stops if I release the key when I'm at an irregular surface.

Retrieved from "http://www.ogre3d.org/wiki/index.php/OgreOde_Walking_Character"

Categories: Animation | ODE | Physics

- This page was last modified 16:46, 11 December 2006.
- Source code is public domain, other content is available under GNU Free Documentation License 1.2.