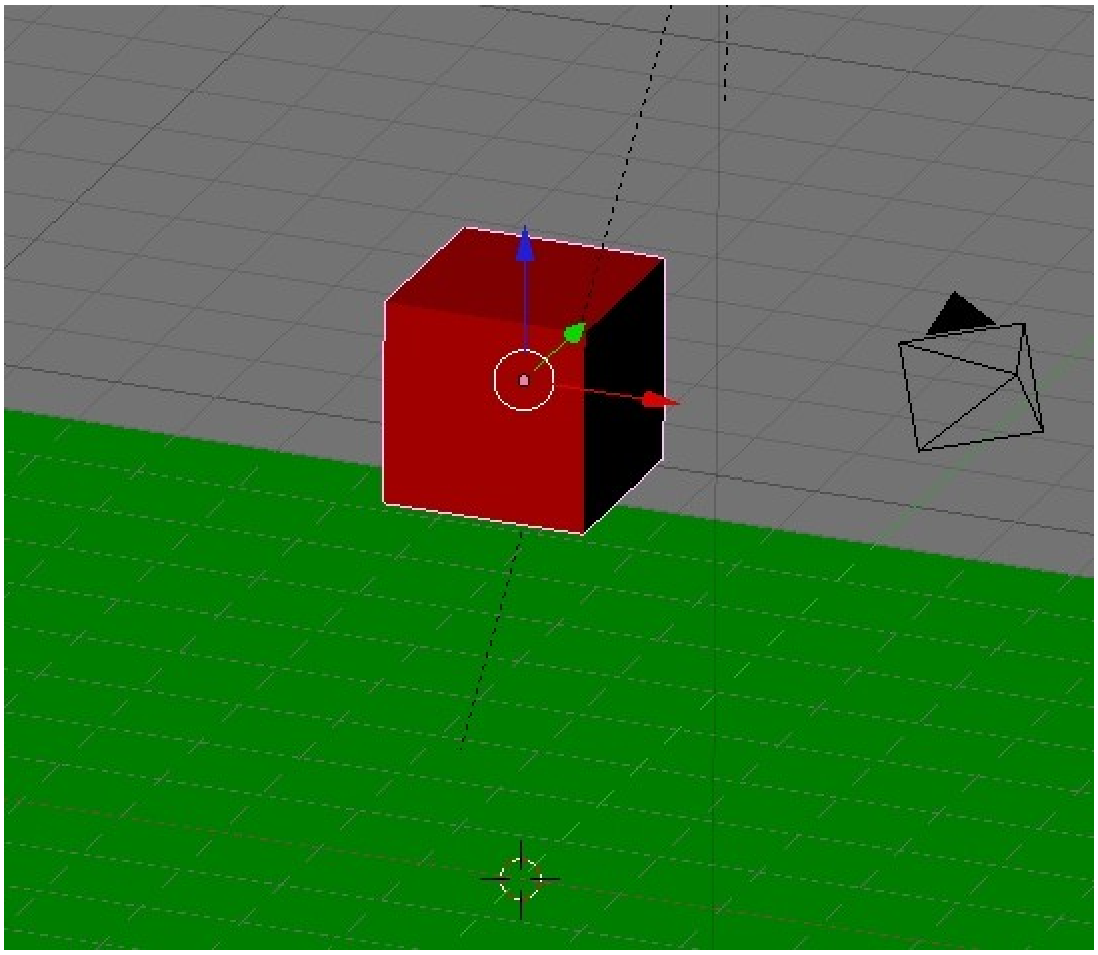
Blender Springs and Dampers (021; 29.09.2009; *blender*)

Blender Game Engine extension scripts for dynamic Springs and Dampers.

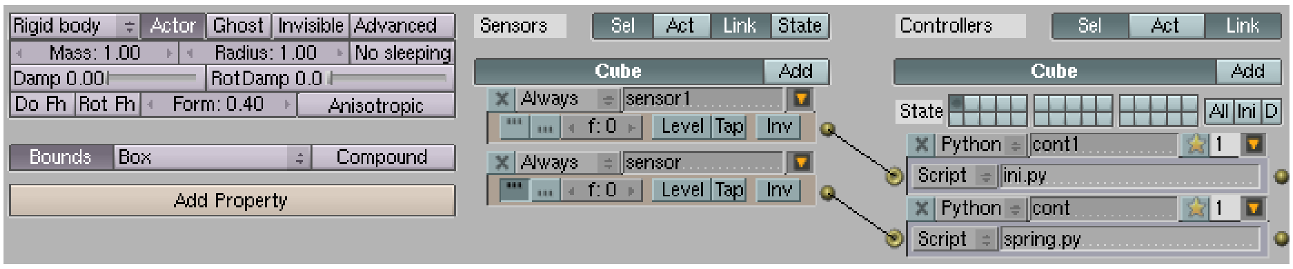
I have decided to extend standard Blender Game Engine with springs and dampers. It is useful when working with mechanisms.

Linear spring.

Create scene with Cube and Plane like this:



Set cube as Rigid Body and put game logic:


Rigid body | Actor | Ghost | Invisible | Advanced
Mass: 1.00 | Radius: 1.00 | No sleeping
Damp 0.00 | RotDamp 0.0
Do Fh | Rot Fh | Form: 0.40 | Anisotropic
Bounds | Box | Compound
Add Property

Sensors | Sel | Act | Link | State
Cube | Add
Always | sensor1 | f: 0 | Level | Tap | Inv
Always | sensor | f: 0 | Level | Tap | Inv

Controllers | Sel | Act | Link
Cube | Add
State | All | Ini | D
Python | cont1 | 1
Script | ini.py
Python | cont | 1
Script | spring.py

Put in text editor python script ini.py with this content:

```
import GameLogic

scene = GameLogic.getCurrentScene()

GameLogic.Object1=scene.objects['OBPlane']
GameLogic.Object2=scene.objects['OBCube']

Loc1=GameLogic.Object1.localPosition
Loc2=GameLogic.Object2.localPosition

GameLogic.IniLength=Loc2[2]-Loc1[2]

GameLogic.Stiff=20
```

and spring.py:

```
import GameLogic

Loc1=GameLogic.Object1.localPosition
Loc2=GameLogic.Object2.localPosition

Length=Loc2[2]-Loc1[2]

Force=(Length-GameLogic.IniLength)*GameLogic.Stiff

Force1=[0,0,Force]
Force2=[0,0,-Force]

GameLogic.Object1.applyForce(Force1, False)
GameLogic.Object2.applyForce(Force2, False)
```

How it works?

ini.py script started once at the beginning of simulation:

- put in global variable objects that will be connected with spring,
- read objects centers location,
- put in global variable distance between objects centers – initial spring length,
- put in global variable spring stiffness.

spring.py started every time-step:

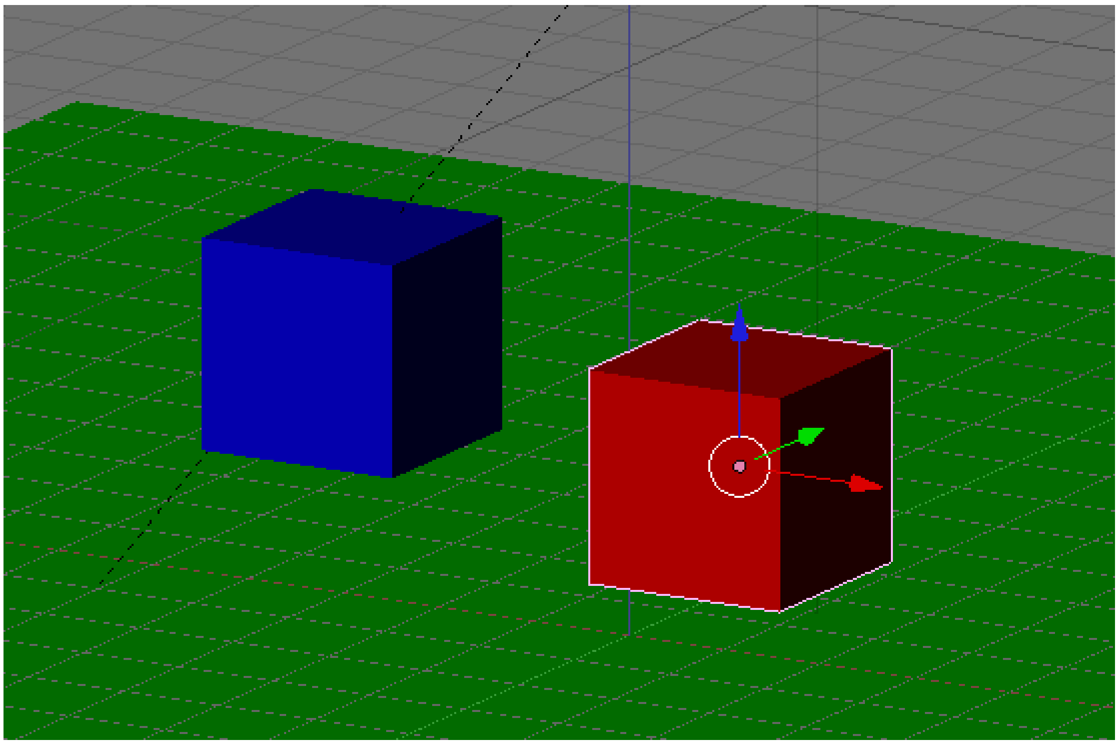
- read objects centers locations and calculate distance between them,
- calculate force in spring:

  spring stiffness * (actual length – initial length),
- create global force vectors and add them to objects.

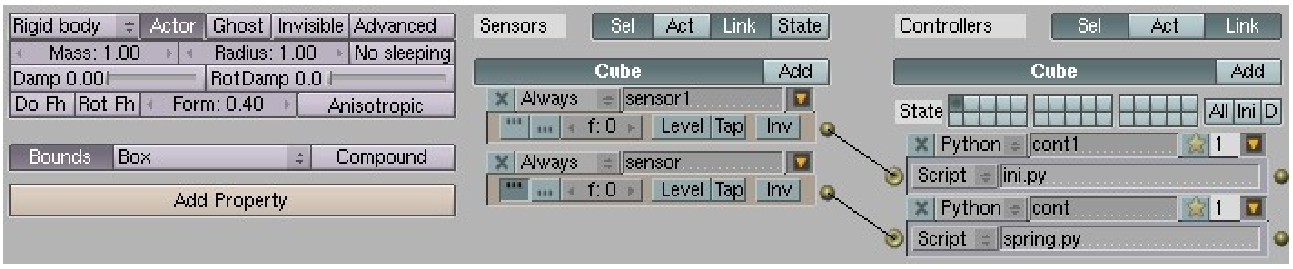
Download scene: http://en.myinventions.pl/021/BlenderLinearSpring.blend.

3D spring.

For complex spring witch working with two rigid body objects create scene witch Cube1, Cube2 and Plane objects.



Just for one rigid body object with Bounds/Box collision make logic:


Rigid body | Actor | Ghost | Invisible | Advanced
Mass: 1.00 | Radius: 1.00 | No sleeping
Damp 0.00 | RotDamp 0.0
Do Fh | Rot Fh | Form: 0.40 | Anisotropic
Bounds | Box | Compound
Add Property

Sensors | Sel | Act | Link | State
Cube | Add
Always | sensor1 | f: 0 | Level | Tap | Inv
Always | sensor | f: 0 | Level | Tap | Inv

Controllers | Sel | Act | Link
Cube | Add
State | All | Ini | D
Python | cont1 | 1
Script | ini.py
Python | cont | 1
Script | spring.py

ini.py content:

```
import GameLogic

scene = GameLogic.getCurrentScene()

GameLogic.Object1=scene.objects['OBCube1']
GameLogic.Object2=scene.objects['OBCube2']

Loc1=GameLogic.Object1.localPosition
Loc2=GameLogic.Object2.localPosition

IniLengthX=Loc2[0]-Loc1[0]
IniLengthY=Loc2[1]-Loc1[1]
IniLengthZ=Loc2[2]-Loc1[2]
GameLogic.IniLength=(IniLengthX**2+IniLengthY**2+IniLengthZ**2)**(0.5)

GameLogic.Stiff=100
```

spring.py content:

```
import GameLogic

Loc1=GameLogic.Object1.localPosition
Loc2=GameLogic.Object2.localPosition

LengthX=Loc2[0]-Loc1[0]
LengthY=Loc2[1]-Loc1[1]
LengthZ=Loc2[2]-Loc1[2]
Length=(LengthX**2+LengthY**2+LengthZ**2)**(0.5)

Force=(Length-GameLogic.IniLength)*GameLogic.Stiff

ForceX=Force*LengthX/Length
ForceY=Force*LengthY/Length
ForceZ=Force*LengthZ/Length

Force1=[ForceX,ForceY,ForceZ]
Force2=[-ForceX,-ForceY,-ForceZ]

GameLogic.Object1.applyForce(Force1, False)
GameLogic.Object2.applyForce(Force2, False)
```

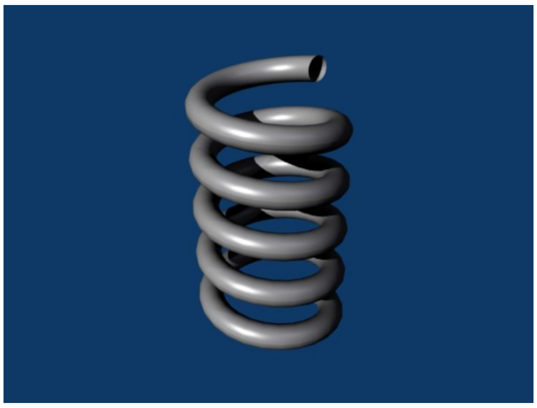
There were necessary to calculate distance in 3 dimensions and project spring force to three components vector.

Finished scene: http://en.myinventions.pl/021/Blender3dSpring.blend.

Spring visualization.

It will be helpful to create spring mesh that represents spring just generated.

You can make spring mesh extruding circle through spiral curve. I propose to use Parametric Object script (see: http://www.blinken.com/blender-plugins.php).



Parametric representation of spring surface:

$$\begin{aligned}
&x=(R+r\cos(2\pi v))\sin(2\pi u),\\
&y=(R+r\cos(2\pi v))\cos(2\pi u),\\
&z=\frac{h}{2\pi}u+r\sin(2\pi v),\\
&u\in(0,k),\\
&v\in(0,1),\\
\\
&\mathit{where:}\\
&R-\mathit{spiral\ radius},\\
&r-\mathit{wire\ radius},\\
&h-\mathit{spring\ height},\\
&k-\mathit{coils\ number}.
\end{aligned}$$

For simple visualization you can connect spring with one Ball and one 6DOF constraint on the ends. For dynamic stretching you probably could use armature. I make simple script that scales spring every frame. You will see it later.

Damping in Blender Game Engine.

You should notice how Damp and RotDamp parameters affect objects in Game Engine.

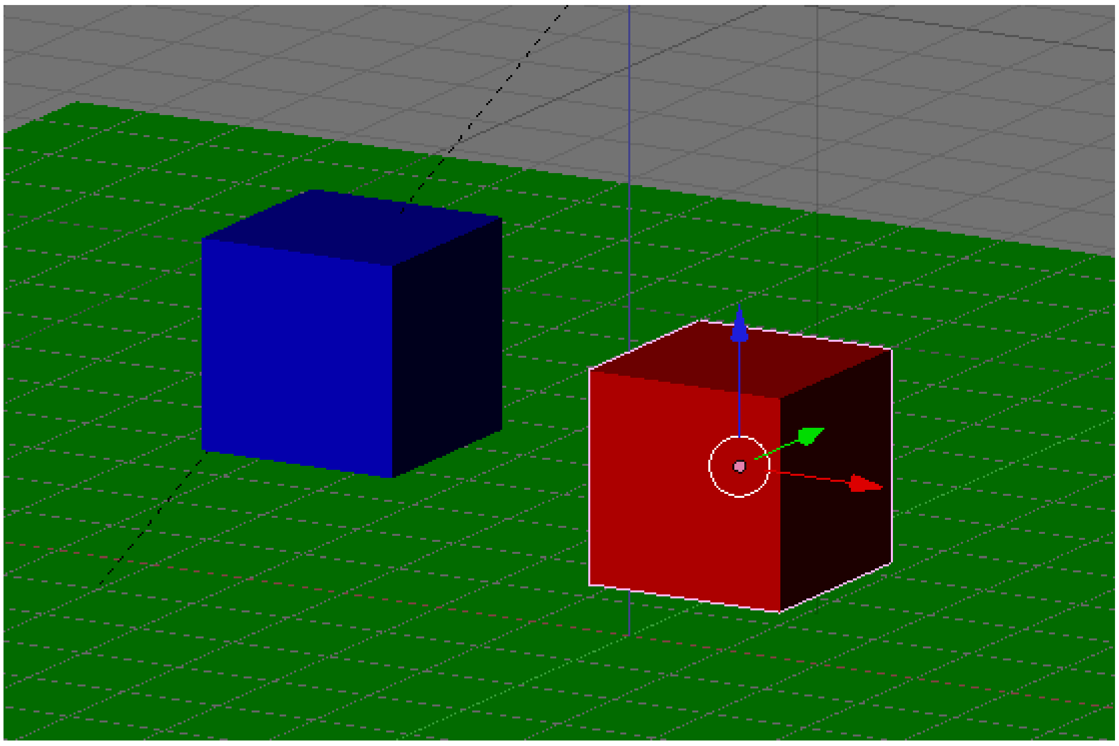
Damp parameter involve only linear movement. Engine add to object damping force opposite to linear velocity. Force is multiplication of velocity, Damp and mass.

RotDamp parameter involve only rotational movement. Engine add to object damping moment of force opposite to angular velocity. Force is multiplication of angular velocity, RotDamp and rotational inertia. Inertia is probably calculated as multiplication of mass and Form factor.

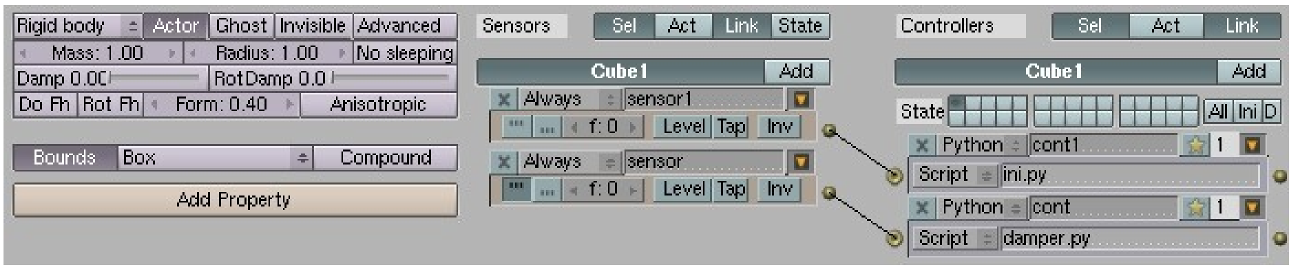
This parameters involve global objects movement, but they have few abilities in mechanics even though they can be set independently.

3D damper.

It will be very similar to spring creation. Start with Cube1, Cube2 and Plane.



Just for one rigid body object with Bounds/Box collision make logic:


Rigid body | Actor | Ghost | Invisible | Advanced
Mass: 1.00 | Radius: 1.00 | No sleeping
Damp 0.00 | RotDamp 0.0
Do Fh | Rot Fh | Form: 0.40 | Anisotropic
Bounds | Box | Compound
Add Property

Sensors | Sel | Act | Link | State
Cube1 | Add
Always | sensor1 | f: 0 | Level | Tap | Inv
Always | sensor | f: 0 | Level | Tap | Inv

Controllers | Sel | Act | Link
Cube1 | Add
State | All | Ini | D
Python | cont1 | 1
Script | ini.py
Python | cont | 1
Script | damper.py

ini.py content:

```
import GameLogic

scene = GameLogic.getCurrentScene()

GameLogic.Object1=scene.objects['OBCube1']
GameLogic.Object2=scene.objects['OBCube2']

GameLogic.Damping=5
```

spring.py content:

```
import GameLogic

Loc1=GameLogic.Object1.localPosition
Loc2=GameLogic.Object2.localPosition
V1=GameLogic.Object1.getLinearVelocity(False)
V2=GameLogic.Object2.getLinearVelocity(False)

LengthX=Loc2[0]-Loc1[0]
LengthY=Loc2[1]-Loc1[1]
LengthZ=Loc2[2]-Loc1[2]
Length=(LengthX**2+LengthY**2+LengthZ**2)**(0.5)
Vx=V2[0]-V1[0]
Vy=V2[1]-V1[1]
Vz=V2[2]-V1[2]
V=(Vx*LengthX+Vy*LengthY+Vz*LengthZ)/Length

Force=GameLogic.Damping*V
ForceX=Force*LengthX/Length
ForceY=Force*LengthY/Length
ForceZ=Force*LengthZ/Length

Force1=[ForceX,ForceY,ForceZ]
Force2=[-ForceX,-ForceY,-ForceZ]

GameLogic.Object1.applyForce(Force1, False)
GameLogic.Object2.applyForce(Force2, False)
```

Because for projecting velocity vector we need cosines of this vector angles, we used distances between objects to calculate cosines.

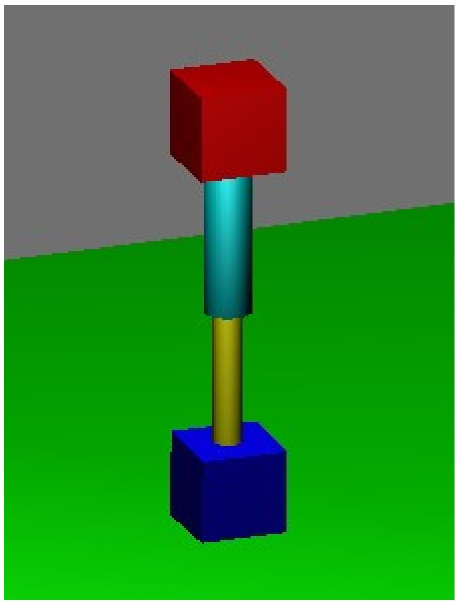
http://en.myinventions.pl/021/Blender3dDamper.blend.

Damper visualization.

Just need two cylinders and three constraints. Connect cubes witch cylinders at centers with Ball constraint and connect cylinders together with linear constraint (6DOF constraint with only LinZ degree of freedom).

http://en.myinventions.pl/021/BlenderDampVisible.blend

.



Kelvin – Voigt model.

The most useful is parallel connection of spring and damper. Just change scripts like this:

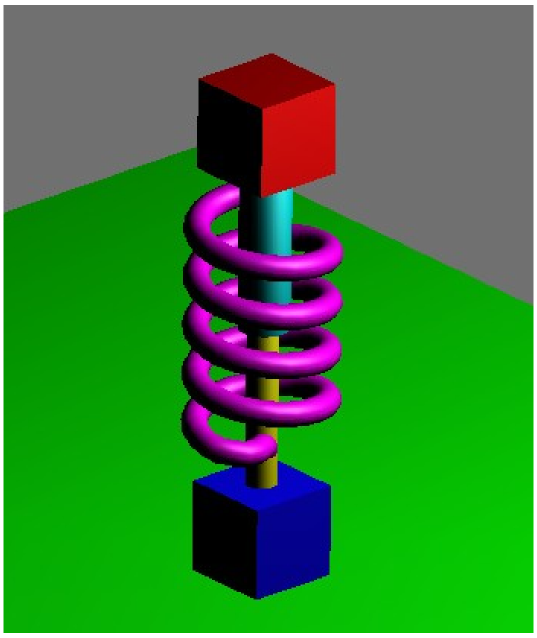
ini.py:

```
import GameLogic

scene = GameLogic.getCurrentScene()

GameLogic.Object1=scene.objects['OBCube1']
GameLogic.Object2=scene.objects['OBCube2']

Loc1=GameLogic.Object1.localPosition
Loc2=GameLogic.Object2.localPosition

IniLengthX=Loc2[0]-Loc1[0]
IniLengthY=Loc2[1]-Loc1[1]
IniLengthZ=Loc2[2]-Loc1[2]
GameLogic.IniLength=(IniLengthX**2+IniLengthY**2+IniLengthZ**2)**(0.5)

GameLogic.Stiff=7
GameLogic.Damping=2
```

SpringDamp.py:

```
import GameLogic

Loc1=GameLogic.Object1.localPosition
Loc2=GameLogic.Object2.localPosition
V1=GameLogic.Object1.getLinearVelocity(False)
V2=GameLogic.Object2.getLinearVelocity(False)

LengthX=Loc2[0]-Loc1[0]
LengthY=Loc2[1]-Loc1[1]
LengthZ=Loc2[2]-Loc1[2]
Length=(LengthX**2+LengthY**2+LengthZ**2)**(0.5)

Vx=V2[0]-V1[0]
Vy=V2[1]-V1[1]
Vz=V2[2]-V1[2]
V=(Vx*LengthX+Vy*LengthY+Vz*LengthZ)/Length

ForceSpring=(Length-GameLogic.IniLength)*GameLogic.Stiff
ForceDamper=GameLogic.Damping*V

ForceX=(ForceSpring+ForceDamper)*LengthX/Length
ForceY=(ForceSpring+ForceDamper)*LengthY/Length
ForceZ=(ForceSpring+ForceDamper)*LengthZ/Length

Force1=[ForceX,ForceY,ForceZ]
Force2=[-ForceX,-ForceY,-ForceZ]

GameLogic.Object1.applyForce(Force1, False)
GameLogic.Object2.applyForce(Force2, False)
```

http://en.myinventions.pl/021/BlenderSpringDamp.blend



See full spring and damper scene with my best visualization:

http://en.myinventions.pl/021/BlenderSpringDampVisible.blend

Using of springs and dampers.

Notice, that springs and dampers are connect with 'virtual ball' constraints witch don't limit system kinematic (don't fix degrees of freedom) and just change dynamic behavior. So it is required to use many Rigid Body Joints constraints (see article – in Polish).

Simulation precision.

I made some tests of Bullet Physics implemented in Blender Game Engine. In my opinion, you can't use this simulation with engineering applications because of non perfect precision (and of course we don't now well how bullet calculate inertia, and constraints are not perfectly rigid). But for artistic and drafts GE is perfect with it's realism and performance.

Fist test: Record object free fall with IPO. You can calculate theoretical fall time and compare it with simulation (simple equation h = g*(t^2)/2). Remember about zero damping and default 60 frames per second, witch you can change with script line:

```
GameLogic.setLogicTicRate(NumberOfFramesPerSecond)
```

This test confirm good precision of calculate of dynamic object with constant acceleration.

Second test: We can check natural frequency of system with object hanged on spring, calculated with equation:

$$f\,[Hz] = \frac{\sqrt{\dfrac{spring\ stiffnes}{mass}}}{2\pi} \quad ,$$

Nonlinear spring.

Typical progressive third order spring have force calculated with script line:

```
Force=((Length-GameLogic.IniLength)**(3))*GameLogic.Stiff
```

Afterword.

My concept of springs and dampers is a bit difficult when we want to have several of it. I thought about Python script generating full spring or damper (constraints, logic bricks, scripts), but Blender 2.49 API haven't access to logic...So be patient and careful when copying or renaming objects.

In this case I'm waiting for new Blender 2.5 API.
Maybe I ought to learn C and C++, take part in Blender and improve game engine with simple springs and dampers panel...?

See Blender Springs and Dampers in action: http://vimeo.com/6814875