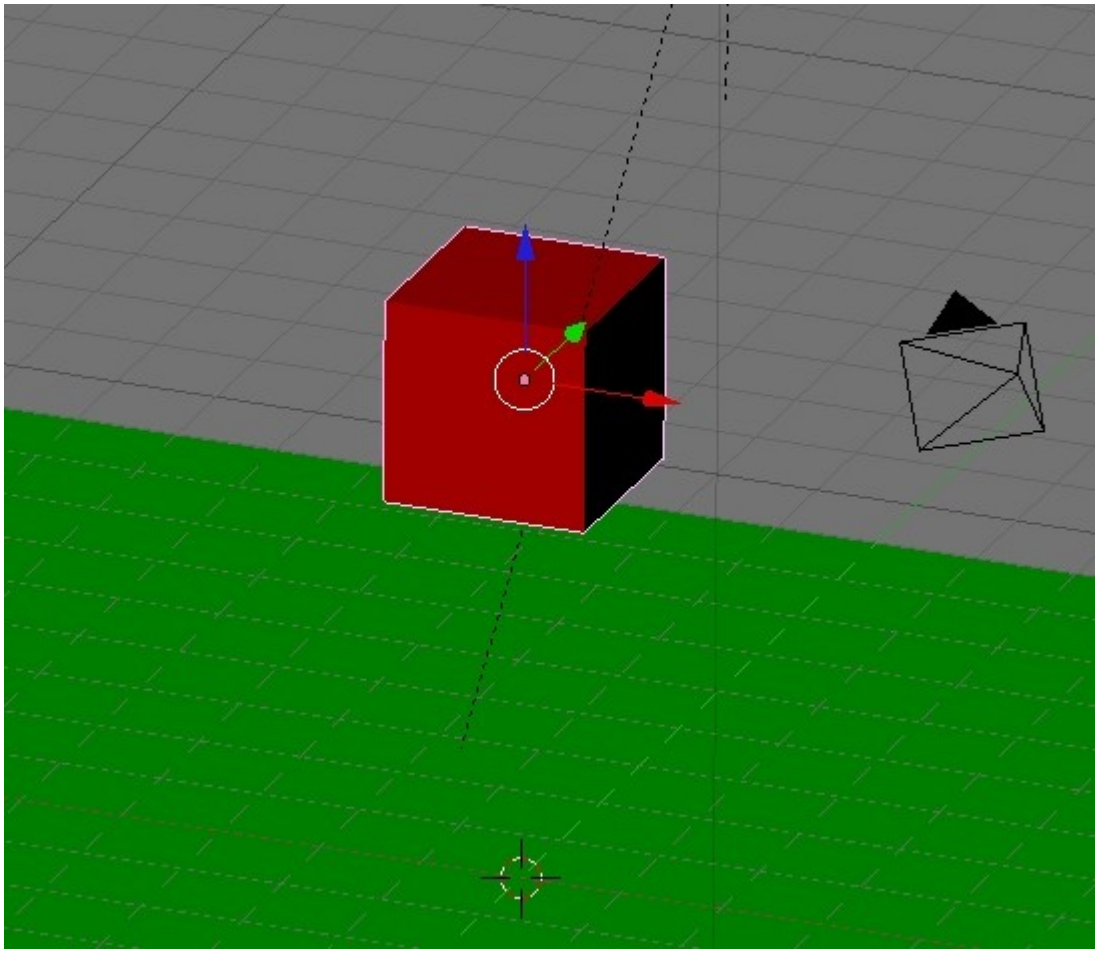
Blender Springs and Dampers (021; 29.09.2009; *blender*)

Rozszerzenie możliwości Blender Game Engine o wstawianie sprężyn i tłumików opartych o skrypty Python.

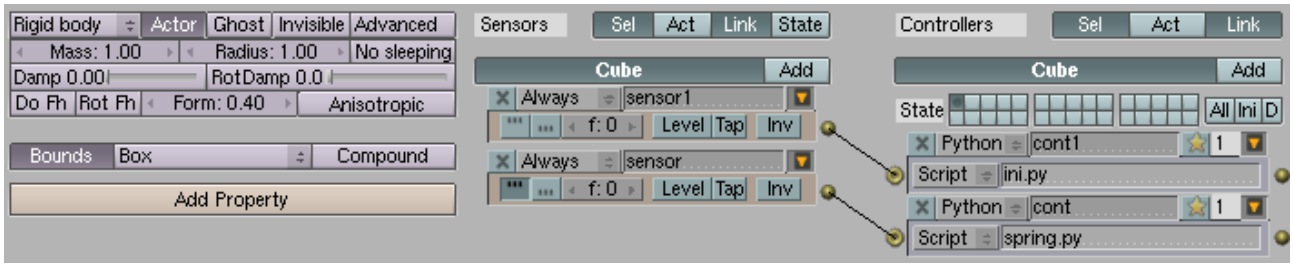
Postanowiłem rozszerzyć standardowe możliwość silnika fizyki w Blender Game Engine o wstawianie sprężyn i tłumików oddziałujących siłowo na łączone nimi elementy. Jest to istotne rozszerzenie możliwości modelowania układów dynamicznych z zastosowaniem wiązań opisanych w artykule WiązaniaBGE.

Sprężyna liniowa.

Utwórzmy scenę z obiektami Cube i Plane jak na poniższym rysunku:



Dla prostopadłościanu ustawmy typ Rigid Body i dodajmy kostki logiki:



W text editor tworzymy skrypt ini.py o treści:

```
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
```

oraz skrypt spring.py o treści:

```
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)
```

Jak to działa?

Skrypt ini.py wywołany jednorazowo na początku symulacji:

- wczytuje do zmiennych globalnych obiekty, które mają być połączone sprężyną,
- odczytuje położenia środków ciężkości obiektów,
- zapisuje jako zmienną globalną odległość między obiektami - długość początkową sprężyny,
- tworzy zmienną globalną przechowującą wartość sztywności sprężyny.

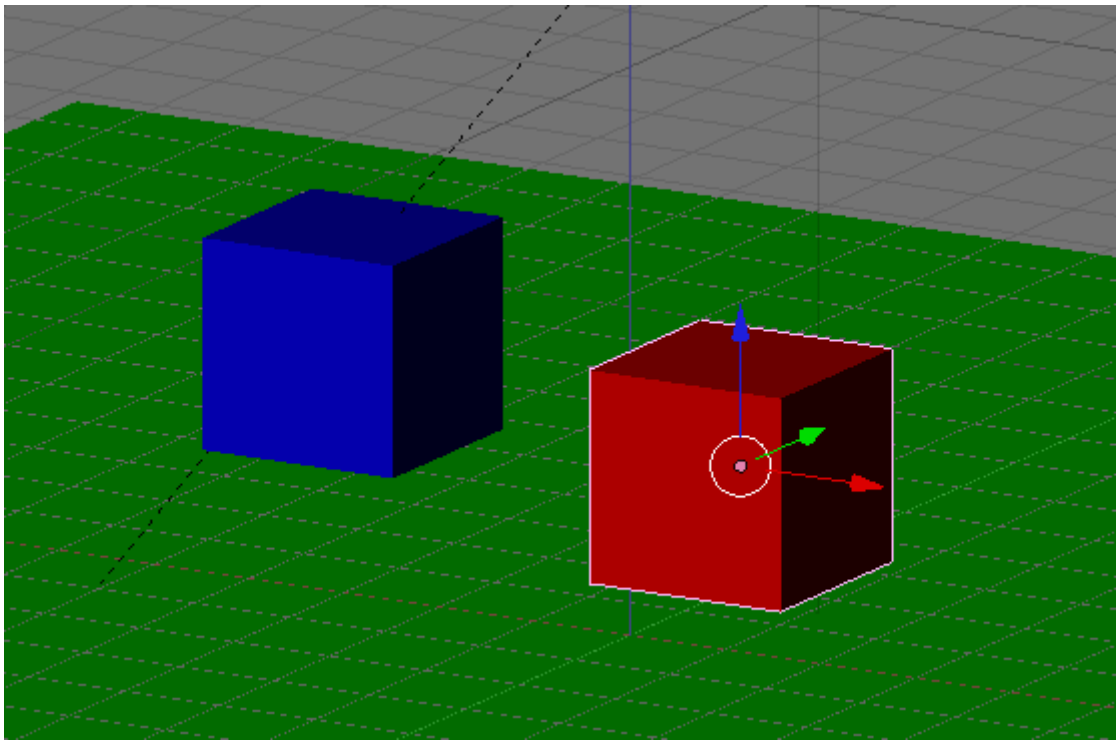
Skrypt spring.py wywoływany w każdym kroku symulacji:

- określa aktualne położenie obu obiektów i oblicza odległość między nimi,
- oblicza siłę generowaną przez sprężynę, która wynosi:
  sztywność sprężyny * (długość aktualna – długość początkowa),
- tworzy globalne wektory sił i dodaje je do obiektów.

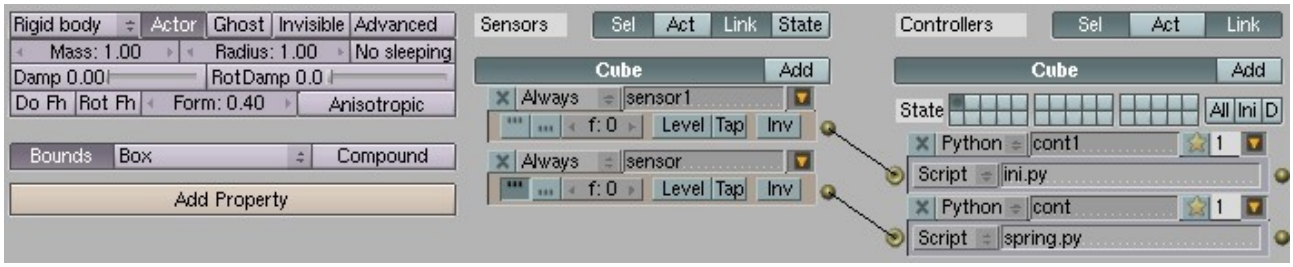
Gotowa scena do pobrania: http://myinventions.pl/021/BlenderLinearSpring.blend.

Sprężyna 3D.

Aby sprężyna działała przestrzennie w przypadku łączenia dwóch ruchomych elementów Rigid Body stwórzmy scenę z elementami Cube1, Cube2 oraz Plane.



Ustalmy obiekty jako Rigid Body z kolizją Bounds/Box. Tylko dla jednego obiektu ustawmy logikę:



W text editor tworzymy skrypt ini.py o treści:

```
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
```

oraz skrypt spring.py o treści:

```
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)
```

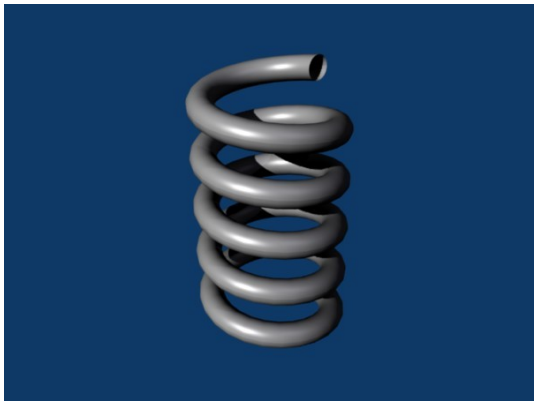
Niezbędne było teraz obliczanie odległości między obiektami wzdłuż każdego kierunku globalnego układu współrzędnych, obliczenie siły w sprężynie i rzutowanie jej na 3 składowe w celu dodania do obiektu jako wektor.

Gotowa scena do pobrania: http://myinventions.pl/021/Blender3dSpring.blend.

Wizualizacja sprężyny.

Przydatne było by stworzenie widzialnej sprężyny rozciągającej się między ruchomymi obiektami połączonymi wyżej opisaną metodą.

Jeśli chodzi o wygenerowanie mesh'a sprężyny, można zrobić to np. odpowiednio operując wyciągnięciem profilu po krzywe. Ja polecam zastosować skrypt Parametric Object (http://www.blinken.com/blender-plugins.php).



Równania parametryczne powierzchni sprężyny:

$$x = (R + r\cos(2\pi v))\sin(2\pi u),$$
$$y = (R + r\cos(2\pi v))\cos(2\pi u),$$
$$z = \frac{h}{2\pi}u + r\sin(2\pi v),$$
$$u \in (0, k),$$
$$v \in (0, 1),$$

gdzie:
R – *promień spirali*,
r – *promień drutu*,
h – *wysokość sprężyny*,
k – *liczba zwojów sprężyny.*

Dla uzyskania prostej wizualizacji wystarczy połączyć sprężynę do jednego obiektu wiązaniem kulistym (Ball) a do drugiego suwliwie (odpowiednio ustawione wiązanie 6DOF).

Aby uzyskać sprężynę dynamicznie zmieniającą kształt należało by zapewne zastosować armaturę i wiązanie stretch to. Można również pokusić się o skrypt dynamicznie generujący sprężynę lub tylko dynamicznie rozciągający obiekt. Ten ostatni sposób będzie można przetestować w późniejszych plikach źródłowych.

Tłumienie w Blender Game Engine.

Warto zwrócić uwagę na sposób oddziaływania wartości parametrów Damp i RotDam na ruch obiektów w środowisku Blender Game Engine.

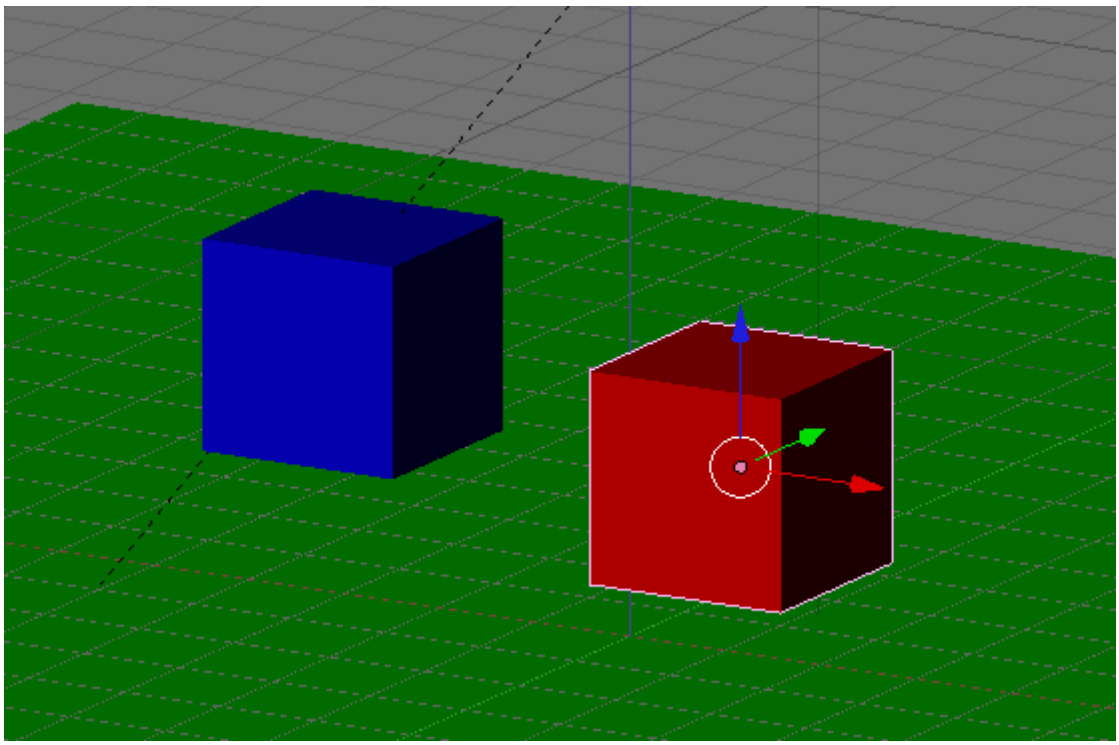
Parametr Damp dotyczy tłumienia tylko w ruchu liniowym. Do obiektu dodawana jest siła przeciwna do wypadkowej prędkości liniowej obiektu, równa jej iloczynowi z parametrem Damp i masą.
Parametr RotDamp dotyczy tłumienia tylko w ruchu obrotowym. Do obiektu dodawany jest moment siły przeciwny do wypadkowej prędkości obrotowej obiektu, równy jej iloczynowi z parametrem RotDamp i momentem bezwładności obiektu. Moment bezwładności najprawdopodobniej liczony jest jako iloczyn masy obiektu i parametru Form.

Parametry te dotyczą wszystkich ruchomych elementów i choć indywidualnie ustalane dla każdego, nie dają zbyt dużych możliwości. Przedstawię zatem sposób stworzenia skryptowego tłumika łączącego dwa elementy.

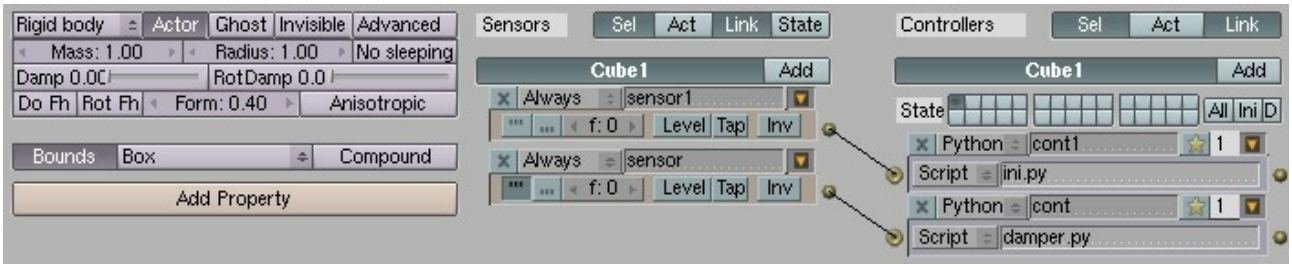
Tłumik 3D.

Stworzenie tłumika będzie bardzo podobne do sprężyny, można wręcz przerobić sprężynę, gdyż należy tylko zastąpić parametr sztywności tłumieniem, oraz obliczenia przemieszczeń odpowiednimi obliczeniami prędkości.

Stwórzmy scenę z elementami Cube1, Cube2 oraz Plane.



Ustalmy obiekty jako Rigid Body z kolizją Bounds/Box. Tylko dla jednego obiektu ustawmy następującą logikę:



Treść skryptu ini.py:

```
import GameLogic

scene = GameLogic.getCurrentScene()

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

GameLogic.Damping=5
```

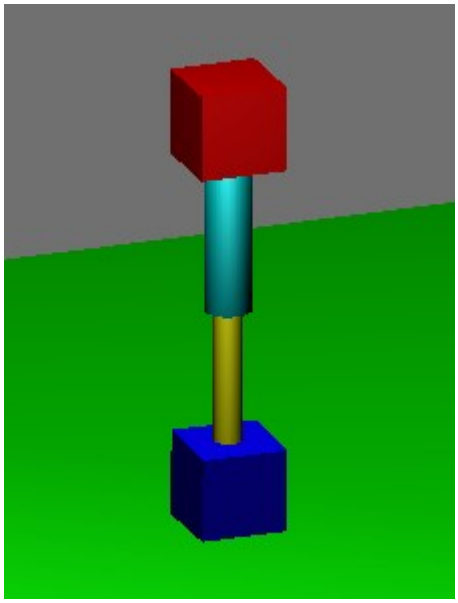
Treść skryptu spring.py:

```
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

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)
```

Ponieważ do rzutowania prędkości niezbędne były cosinusy kątów nachylenia wektora prędkości wypadkowej w obliczeniach potrzebne były również odległości między obiektami.

Plik źródłowy do pobrania: http://myinventions.pl/021/Blender3dDamper.blend.

Wizualizacja tłumika.



Nie będzie to zbyt skomplikowane, wystarczą dwa cylindry i trzy wiązania. Obiekty mocujemy w środkach ciężkości wiązaniami kulistymi odpowiednio do końców dwóch cylindrów, a te łączymy między sobą jednym wiązaniem przesuwnym (wiązanie 6DOF zablokowane wszystko oprócz przesuwu wzdłuż osi cylindrów).

Najlepiej widać to w przykładowym pliku: http://myinventions.pl/021/BlenderDampVisible.blend.

Model Kelvina – Voigt'a.

Najbardziej przydatne będzie jednak połączenie równoległe sprężyny i tłumika w jednym skrypcie. Robimy to podobnie jak we wcześniejszych zagadnieniach, ale skrypty przedstawiają się następująco:

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
```

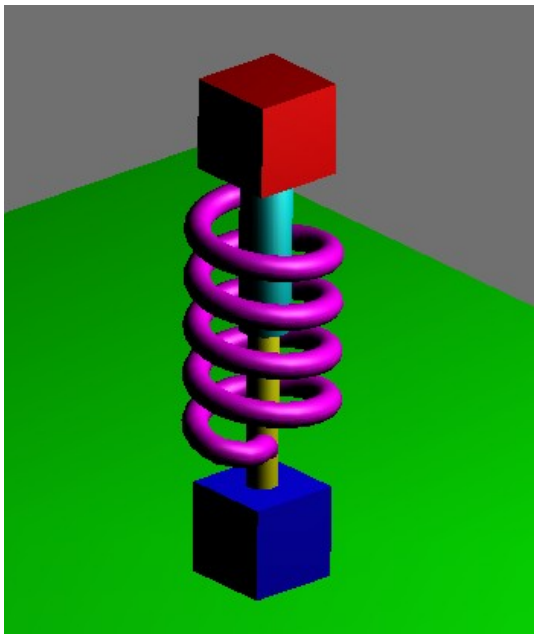
SpringDamp.py:

```
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)
```

Plik źródłowy do pobrania: http://myinventions.pl/021/BlenderSpringDamp.blend

Dodając odpowiednie wizualizacje otrzymać możemy następującą scenę:

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



Użytkowanie sprężyn i tłumików.

Można zauważyć, ze sprężyny i tłumiki przyczepiane są do elementów pewnego rodzaju połączeniami kulistymi. Sprawia to, że w istocie nie ograniczamy kinematyki ruchu tych obiektów (nie odbieramy im stopni swobody), ale tylko powodujemy zmianę ich zachowań dynamicznych. Niezbędne zatem będzie używanie wiązań typu Rigid Body Joints, w czym przydatny jest wcześniejszy artykuł artykuł WiązaniaBGE.

Dokładność symulacji.

Przeprowadziłem kilka testów środowiska Bullet Physics zaimplementowanego w Blender Game Engine i stwierdzam, że silnik obliczeń fizyki choć oparty o modelowanie teoretycznie poprawne, wprowadza uproszczenia na tyle duże, że wyniki symulacji mogą nie pokrywać się z wynikami symulacji numerycznych otrzymanych w programach o zastosowaniu typowo inżynierskim. Jednak dla potrzeb artystycznych i poglądowych otrzymujemy duży realizm z zachowaniem odpowiedniej wydajności.

Przeprowadzić możemy najprostszy test spadku swobodnego pewnego obiektu z zadanej wysokości. Zapisując symulację do IPO możemy określić czas spadku z zadanej wysokości i porównać z wartością teoretyczną (prosty wzór h=g*(t^2)/2), pamiętając, że domyślnie ustalone jest 60klatek na sekundę a zmienić tą liczbę możemy prostym poleceniem do wywołania skryptem:

```
GameLogic.setLogicTicRate(liczbaKlatekNaSekundę)
```

Test ten potwierdza dość dobrą dokładność obliczeń ruchu swobodnego ciała pod działaniem jednej stałej siły grawitacji (pamiętając o zmniejszeniu tłumienia do 0).

Kolejny test dotyczyć może wymodelowanej sprężyny. Wiedząc, że częstotliwość drgań własnych obiektu zawieszonego na sprężynie liniowej równa jest:

$$f\,[Hz] = \frac{\sqrt{\frac{sztywność}{masa}}}{2\pi} \quad ,$$

porównać możemy ją z rzeczywistym efektem symulacji. Wg moich testów błąd tej częstotliwości jest niewielki i zmniejsza się wraz ze wzrostem liczby klatek obliczeń na sekundę.

Sprężyna nieliniowa.

Nic trudnego, typową sprężynę progresywną 3 rzędu otrzymać możemy obliczając siłę w sprężynie we wcześniej używanych skryptach następująco:

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

Posłowie.

Jak widać, wstawienie pojedynczej sprężyny lub tłumika związane jest z odrobiną zachodu i mnożeniem kodów do skryptów. Rozpocząłem planowanie stworzenia skryptu Python, który pozwalałby w prosty sposób generować sprężyny i tłumiki do zastosowania potem w GE. Jednak, niestety, jak na razie Python API w Blenderze nie daje żadnej możliwości kontroli w tworzeniu i ustawianiu kostek logiki Blendera jak również przypisywania do nich skryptów...
Dla większych scen niezbędne jest zatem kopiowanie obiektów i skryptów przy zachowaniu ogromnej ostrożności w nazewnictwie i wiązaniach pomocniczych. W tej sytuacji muszę zaczekać na nowe przebudowane API w Blenderze 2.5.
Gdyby nie brak czasu rozważyłbym możliwość nauczenia się 'tworzenia Blendera' (który pisany jest w języku C), a następnie wspomógł bym prace rozwojowe API Blendera, aby jak najwięcej funkcji Bullet'a mogło zostać wykorzystane w Blenderze z pomocą skryptów. Najlepszym było by oczywiście stworzenie w Blenderze nowego panelu umożliwiającego dodawanie sprężyn i tłumików tak jak zwykłe wiązania wiązania...

Zobacz filmik: http://vimeo.com/6814875