Witold Alda

Computer Graphics. Lab 6.

Calculating collisions with physi.js library

In this exercise we use the **Physi.js** library one of more popular libraries for physical simulations. There are a few alternative physical libraries which work with WebGL/Three.js, among them: **cannon.js** and **babylon.js** (whith much bigger capacities).
This wxcersize is not strictly graphics, but is important in animations and video games.
o

Building a three.js scene with Physijs

Creating a scene using Physijs is simple and consists only of a few steps.
The library itself together with supporting files can be downloaded from

`http://chandlerprall.github.io/Physijs/`.

Short description of the library is available on: https://github.com/chandlerprall/Physijs/wiki

Usage of Physijs can as follows:

1. We import the library:

```
<script type="text/javascript" src="physi.js"></script>
```

2. We start *worker* mode for simulation.

Physical simulations involving collisions are usually computationally intensive. The problem arises when we want to do rendering at the same time. Simulation can adversely affect the speed of rendering. Moreover it's even more difficult when executed in Web browser.
Fortunately is possible with modern Web browsers, as they can hold execution of multiple threads simultaneously. In Physijs we perform physical calculations in the background thread through the use of Web workers mechanism available in most browsers. Information on Web workers can be found on:
`http://www.whatwg.org/specs/web-apps/current-work/multipage/workers.html`

In Physijs we need to identify what we want to run in background (i.e. worker mode) and what physics engine we use (in our case ammo.js, Physijs is just a wrapper). The engine itself, ammo.js, can be found at
`https://github.com/kripken/ammo.js/`

In the code we put:
```
Physijs.scripts.worker = 'physijs_worker.js';
Physijs.scripts.ammo = 'ammo.js';
```

3. In the next step we need to make a scene in which we will be using physics.

W replace `THREE.Scene` by `Physijs.Scene` to add gravity as a new scene attribute:

```
var scene = new Physijs.Scene();
scene.setGravity(new THREE.Vector3(0, -10, 0));
```

4. Subsequently we add object which are simulated. They are ordinary Three.js objects ‘wrapped’ by Physijs, i.e. with new attributes added.

```
var stoneGeom = new THREE.CubeGeometry(0.6,6,2);
var stone = new Physijs.BoxMesh(stoneGeom,
new THREE.MeshPhongMaterial({color: 0xff0000}));
scene.add(stone);
```

In the above code we replace a `THREE.Mesh` object with a `Physijs.BoxMesh` object, which enables collision simulation. We can see that instead of single `Mesh` function, `Physijs` supplies us with several different functions adjusted to the shape of the object. This is because in fact we generate two objects: one with precise geometry, and the second for collision purposes. These two objects do not need to be identical.
`Physi.js` classes are gathered below:

Shapes handled by Physi.js

Name	Description
`Physijs.PlaneMesh`	This mesh can be used to create a zero-thickness plane. You could also use a `BoxMesh` object for this along with a `THREE.CubeGeometry` property with a low height.
`Physijs.BoxMesh`	If you have geometries that look like a cube, use this mesh. For instance, this is a good match for the `THREE.CubeGeometry` property.
`Physijs.SphereMesh`	For spherical shapes use this geometry. This geometry is a good match for the `THREE.SphereGeometry` property.
`Physijs.CylinderMesh`	With the `THREE.Cylinder` property you can create various cylindrical shapes. Physijs provides multiple meshes depending on the shape of the cylinder. The `Physijs.CylinderMesh` should be used for a normal cylinder having a same top and bottom radius.
`Physijs.ConeMesh`	If you specify the top radius as `0` and use a positive value for the bottom radius, you can use the `THREE.Cylinder` property to create a cone. If you want to apply physics to such an object the best fit from Physijs is the `ConeMesh` class.
`Physijs.CapsuleMesh`	A capsule is just like a `THREE.Cylinder` property, but with a rounded top and bottom. We'll show you how to create a capsule in Three.js, further down in this section.
`Physijs.ConvexMesh`	A `Physijs.ConvexMesh` is a rough shape you can use for more complex objects. It creates a convex (just like the `THREE.ConvexGeometry` property) to approximate the shape of complex objects.

`Physijs.ConcaveMesh`	While the `ConvexMesh` is a rough shape, a `ConcaveMesh` is a more detailed representation of your complex geometry. Note that the performance penalty of using a `ConcaveMesh` is very high. Usually it is better to either create separate geometries with their own specific Physijs mesh, or group them together (like we do with the floors shown in the previous examples).
`Physijs.HeightfieldMesh`	This mesh is a very specialized one. With this mesh you can create a height field from a `THREE.PlaneGeometry` property. Look at example `03-shapes.html` for an example of this mesh.

5. We can start simulation in the following way:

```
render = function() {
requestAnimationFrame(render);
renderer.render(scene, camera);
scene.simulate();}
```

Examples

In the attached file we've got three examples. We plan to modify only one of them (balls.html). The others, which come from physijs homepage (collisions.html i shapes.html), we'll browse only . Please look at the parameters in the objetcs' definitions, using methods: `Physijs.BoxMesh` and `Physijs.createMaterial.`

We can see parameters .4, .99 and 0 which mean friction, restitution and gravity. Please make experiments with different values.

```
plane = new Physijs.BoxMesh(
      new THREE.CubeGeometry(100, 100, 2, 10, 10),
      Physijs.createMaterial(
        new THREE.MeshLambertMaterial({
          color: 0xeeeeee
        }),
        .4,             // friction
        .99             // restitution
      ),
      0                 // gravity
    );
```

To do:

Please modify the scen with ball by adding obstacles, changing positions of the obstacles, changing the frequency of balls generation, or something else. You may generate a box (or a drawer) which could be filled with balls, which would eventually fall to another box and so on. Whatever you like…