Witold Alda

Computer Graphics. Lab 2. Lighting in Three.js (revision 97)

Elements of the three.js library

The scene in Three.js library is built using objects. The scene itself is also an object - container - in which other objects are placed, both geometric and the other, such as cameras and light sources.

Basic objects which we create, are defined using built-in classes. They are:

- Rendereres (we have at least two of them: Canvas renderer and WebGL renderer, the latter needs better hardware.
- Geometry shapes – ready made (cube, sphere, etc.) or created manually as a polygon mesh.
- Objects which represent an observer (*camera*), its position, point it is looking at, viewing angle, etc.
- Light sources which illuminate the scene.
- Materials attached to geometry forms, which define their color, reflection of light, etc.

Three.js objects have their own properties and methods. Good documentation can be found on ***threejs.org*** page. We'll use it often.

In the exercises we use some of predefined objects.
Following are available (this is not a complete list , however):

- 2D:
  - PlaneGeometry
  - CircleGeometry
  - ShapeGeometry
- 3D:
  - BoxGeometry
  - SphereGeometry
  - CylinderGeometry
  - TorusGeometry
  - TorusKnotGeometry
  - PolyhedronGeometry
  - IcosahedronGeometry
  - OctahedronGeometry
  - TetraHedronGeometry

Every shape has its specific parameters – check in the documentation.

Object placed in the scene belong to *Mesh* class. They consist of shape (geometry) and material, e.g.:

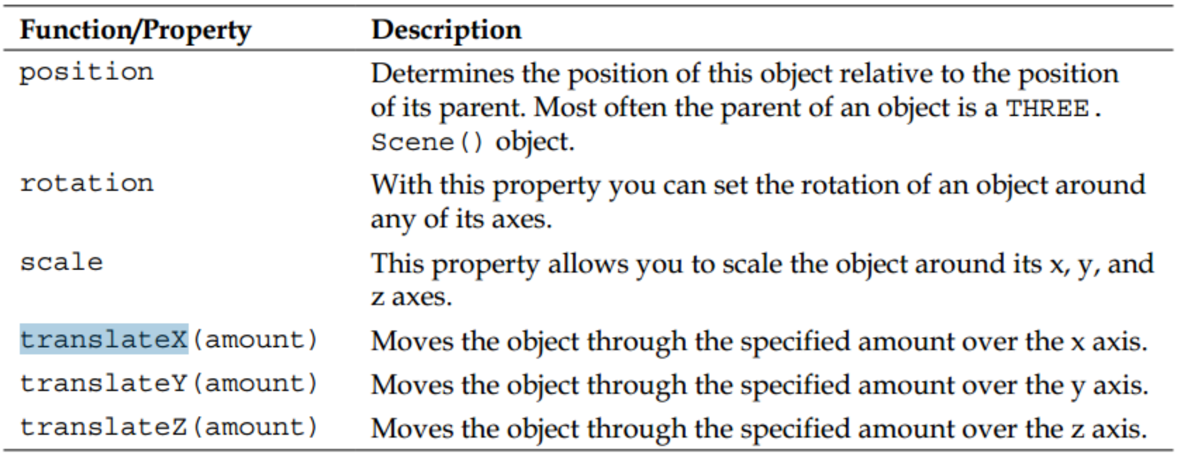
```
var cubeGeometry = new THREE.BoxGeometry(4,4,4);
var cubeMaterial = new THREE.MeshLambertMaterial({color:0xff0000});
var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
```

Materials describe surface parameters, such as color, light reflection, glossiness, transparency, opacity, etc. The list of available materials in three.js contains 16 items revision 97. For us most important currently are `MeshLambertMaterial` i `MeshPhongMaterial` , and perhaps still `MeshBasicMaterial`.

`MeshBasicMaterial` is the simplest one and it does not react to lights. It can, however, keep color, also quite complex, set by usage of color maps.

`MeshLambertMaterial` keeps capabilities of `MeshBasicMaterial`. Aditionaliy it reflects light in a diffused manner. `MeshPhongMaterial` can also reflect specular light with highlights. There is one more difference between these two materials. `MeshLambertMaterial` calculates illumination in mesh vertices and interpolates it linearly between them, while `MeshPhongMaterial` calculates lighting separately in every pixel, based on interpolated normal vectors

We can easily modify *Mesh* objects using suitable attributes.



Function/Property	Description
`position`	Determines the position of this object relative to the position of its parent. Most often the parent of an object is a `THREE.Scene()` object.
`rotation`	With this property you can set the rotation of an object around any of its axes.
`scale`	This property allows you to scale the object around its x, y, and z axes.
`translateX(amount)`	Moves the object through the specified amount over the x axis.
`translateY(amount)`	Moves the object through the specified amount over the y axis.
`translateZ(amount)`	Moves the object through the specified amount over the z axis.

Object position can be established in three ways.
First:

```
cube.position.x=10;
cube.position.y=3;
cube.position.z=1;
```

Second:

```
cube.position.set(10,3,1);
```

Third:

```
cube.position=new THREE.Vector3(10,3,1);
```

Lighting Models in three.js.

Three.js library is based on objects. In case of lighting, these objects are light sources and materials which describe reflection attributes.

In three.js we have seven types of light sources defined:

Nazwa	Opis
`AmbientLight`	Environment light

PointLight	
Spotlight	Point light, limited to a cone
DirectionalLight	
HemisphereLight	Combined lighting which emulates sky light reflection from the terrain.
AreaLight	Lighting rectangle
LensFlare	Flashes

We concentrate only on three of them

AmbientLight

Can be introduced in the following way:

```
var ambiColor = "#0c0c0c";
var ambientLight = new THREE.AmbientLight(ambiColor);
scene.add(ambientLight);
...
var controls = new function() {
this.ambientColor = ambiColor ;
}
var gui = new dat.GUI();
gui.addColor(controls, 'ambientColor').onChange(function(e) {
ambientLight.color = new THREE.Color(e);
});
```

Class inheritance is as follows:

`Object3D → Light → AmbientLight`

Constructor example for `AmbientLight`:

```
var light = new THREE.AmbientLight( 0xff8080, 1);
scene.add( light );
```

Constructor parameters are:

```
AmbientLight( color : Integer, intensity : Float)
```

where:

Atribute	**Description**
color	RGB color, default is white 0xFFFFFF
intensity	Intensity of light, by default 1.0

PointLight

Pointlight lights in every direction. Its basic attributes are as follow:

Attribute	Description
`color`	RGB color
`intensity`	Intensity of light, by default 1.0
`distance`	Distance at which intensity falls to 0. If distance=0, the light reaches infinity.
`position`	Of the point light of course.
`visible`	If *true* the light source is active.

```
      var pointColor = "#ccffcc";
      var pointLight = new THREE.PointLight(pointColor);
      pointLight.distance = 100;
      pointLight.position.set( 50, 50, 50 );
      scene.add(pointLight);
```

Increasing intensity:

```
      pointLight.intensity = 2.4;
```

Light control using dat.GUI

```
var controls = new function() {
            this.intensity = 1;
      }
      var gui = new dat.GUI();
      gui.add(controls, 'intensity', 0, 3).onChange(function (e) {
            pointLight.intensity = e;
      });
```

SpotLight

SpotLight jest often used in realistic scenes.
Beside attributes typical to PointLight, SpotLight has a few of its own:

Attribute	Description
`castShadow`	If *true*, it casts shadows
`Shadow.camera.near`	Form what distance from the light shadows are created
`Shadow.camera.far`	To what distance from the light shadows are created
`Shadow.camera.fov`	Angle for shadow calculation
`target`	`SpotLight` direction.
`shadowBias`	Can offset the position of the shadow
`angle`	Light cone angle (Pi/3 – default)
`penumbra`	Intensity decrease along the radius
`shadow.camera`	Lines show camera frustum

`shadow.map.width`	Number of pixels used to make shadows
`shadow.map.height`	Number of pixels used to make shadows

Introducing SpotLight is simple :

```
var pointColor = "#ffffff";
var spotLight = new THREE.SpotLight(pointColor);
spotLight.position.set(-40, 60, -10);
spotLight.castShadow = true;
spotLight.target = plane;
scene.add(spotLight);
```

In the latest version of three.js shadow parameters have changed.

To do…

In this lab we limit ourselves to modifications of only one example (the first, 01ambient and spot-light). However it's useful to browse other examples as well.

The scene displays three geometry objects: a plane, of *PlaneGeometry* class, a cube from *BoxGeometry*, a sphere from *SphereGeometry*.

In the example we use a „torch-like" source, namely a `SpotLight` and material which reflects diffuse light, the `MeshLambertMaterial`.

We also introduce a simple animation. In the example, changing positions, rotations, etc. are assigned to the objects in `render()` function. Subsequently, a sequence of these changes (animation) is achieved by calling:

```
requestAnimationFrame(render);
```

inside the render() function.
Using `requestAnimationFrame(render);` begins the animation with 60 fps (if only the computer can proceed it fast enough).

In this exercise, please, change different lighting parameters. You may use a Three.js documentation from threejs.org . In particular, please:

- Change `SpotLight` parameters: its position and FOV angle to smaller. Setting `shadowCameraVisible` to *true*, enables to display lines showing the shadowing area. Please get the image of the light circle on the plane and the objects. This should be done for both materials, i.e.: `MeshLambertMaterial` and `MeshPhongMaterial`. It's worth looking that they behave completely different. For `MeshLambertMaterial` we need a fine mesh (light is interpolated between vertices). `MeshPhongMaterial` works directly on pixels and does not need any mesh. Please play with changing these parameters and compare the results with both materials. Choose the one, which you like.
- Please add another light source, directional or spotlight, but with bigger FOV angle.
- Check which parameters control shadow casting and shadow receiving. Can you cast a cube shadow on the sphere?

- Please change the material from `MeshLambertMaterial` to `MeshPhongMaterial` in order to get specular lights on the cube and sphere.
- Increase `shadow.map.width` and `shadow.map.height` parameters to get better results . You may also play with `shadow.camera.near`, `shadow.camera.far` and `shadow.camera.fov` . Make the shadows look nice and natural.
- Please add a transparent cone to a spotlight. This cone should mimic light which we usually see in the fog, when light is diffused on the droplets of water. Partial transparency of such a cone can be achieved using `transparent` and `opacity` parameters, when defining materials e.g.

  ```
  new THREE.MeshLambertMaterial( { opacity:0.6, color: 0x44ff44,
  transparent:true } );
  ```

  Try to make the cone move along with the light source.

- Please add more moving objects and more moving, preferabely color, light sources.