

# LABORATORIO DI REALTÀ AUMENTATA

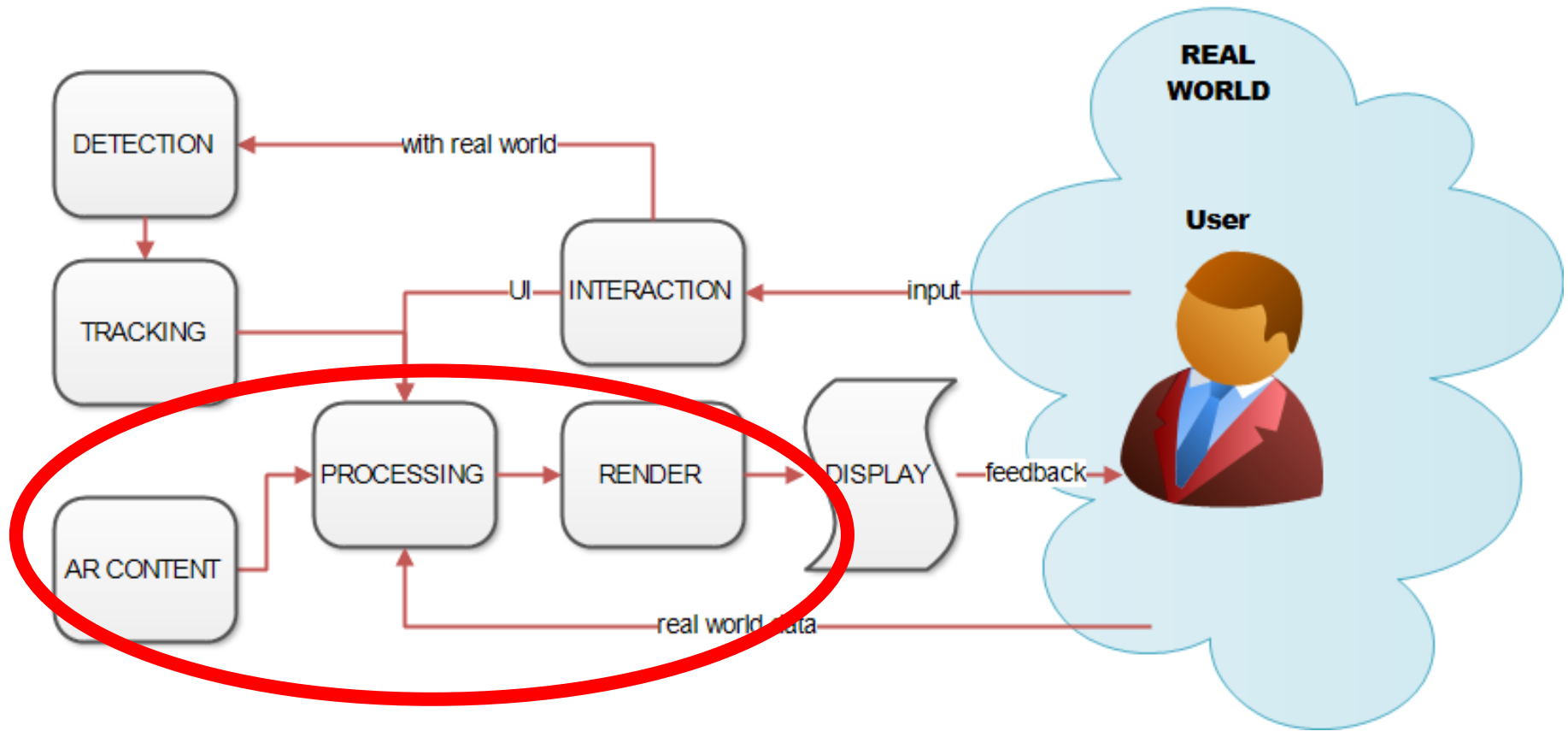
Claudio Piciarelli

Università degli Studi di Udine  
Corso di Laurea in Scienze e Tecnologie Multimediali



# Project: 3D graphics

# Architecture of an AR system



# Canvas and 3D

- We already know HTML5's canvas has a 2D graphics context ( `canvas.getContext("2d")` )
- There is also a 3d context, which is accessible only through the WebGL API. It allows to display interactive 3d graphics without the use of external plugins
- Check for support: <http://get.webgl.org/>

# WebGL



3D graphics contents directly rendered in web pages. Initially based on...



API for mobile devices. Based on...



cross-platform API initially developed by SGI since 1991

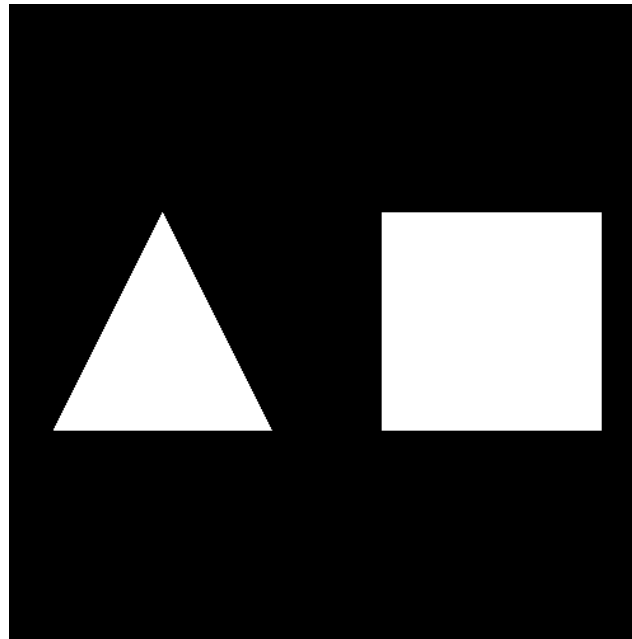
# WebGL: pros

---

- ❑ Extremely powerful
- ❑ Extremely fast, since it can directly access the graphics hardware

# WebGL: cons

- Extremely COMPLEX



- Check the tutorial for drawing these shapes:

<http://dzeek.net/dzeek/molythio/webgl/tutorial/lesson1.html>

# WebGL made easy: Three.js

- How can we use the powerful features of WebGL, avoiding the hassle of a complex API?
- By using an high-level 3D graphics library which hides all the WebGL complexity to the user
- **Three.js**
- <http://threejs.org/>



# Three.js resources

- A very nice introduction to Three.js

<http://davidscottlyons.com/threejs/presentations/frontporch14/index.html#slide-0>

- Examples for almost every Three.js function

<http://stemkoski.github.io/Three.js/>

- Three.js website has many examples too

<http://threejs.org/>

# Preparing for the lesson

- First of all, download three.js from the original website (warning: ~300 MB download!) or take the basic library file from the elearning website
- We will use the compact version named three.min.js
- Copy this file in the same folder of your web pages
- Current version: r127

# Preparing for the lesson

- Create a basic HTML page with a canvas

```
<!doctype html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Three.js test</title>
  <script src="three.min.js"></script>
</head>
<body>
  <canvas width="640" height="480" id="mycanvas"></canvas>
</body>
</html>
```

# Setting up three.js

```
<script>
window.onload = function() {
  // setup three.js
  var mycnv = document.getElementById("mycanvas");
  var renderer = new THREE.WebGLRenderer({canvas: mycnv});
  var scene = new THREE.Scene();
  var camera = new THREE.PerspectiveCamera( 75, mycnv.width
                                          / mycnv.height, 0.1, 1000 );

  scene.add(camera);
}
</script>
```

# Renderer

```
var renderer = new THREE.WebGLRenderer({canvas: mycnv});
```

- The renderer creates a 2D image (visible on the screen) of the 3D world, as seen from the camera
- `{canvas: mycnv}` – the optional input is an object with a single property called “canvas”. It’s the canvas where to draw
- Alternative solution (does not require the onload event):

```
var renderer = new THREE.WebGLRenderer();  
renderer.setSize( window.innerWidth, window.innerHeight );  
document.body.appendChild( renderer.domElement );
```

# Scene

```
var scene = new THREE.Scene();
```

- The `scene` is a container for the objects to be visualized
- It also contains the two other fundamental elements for an object to be visible
  - ▣ Cameras
  - ▣ lights

# Camera

```
var camera = new THREE.PerspectiveCamera( 75, mycnv.width  
                                           / mycnv.height, 0.1, 1000 );  
scene.add(camera);
```

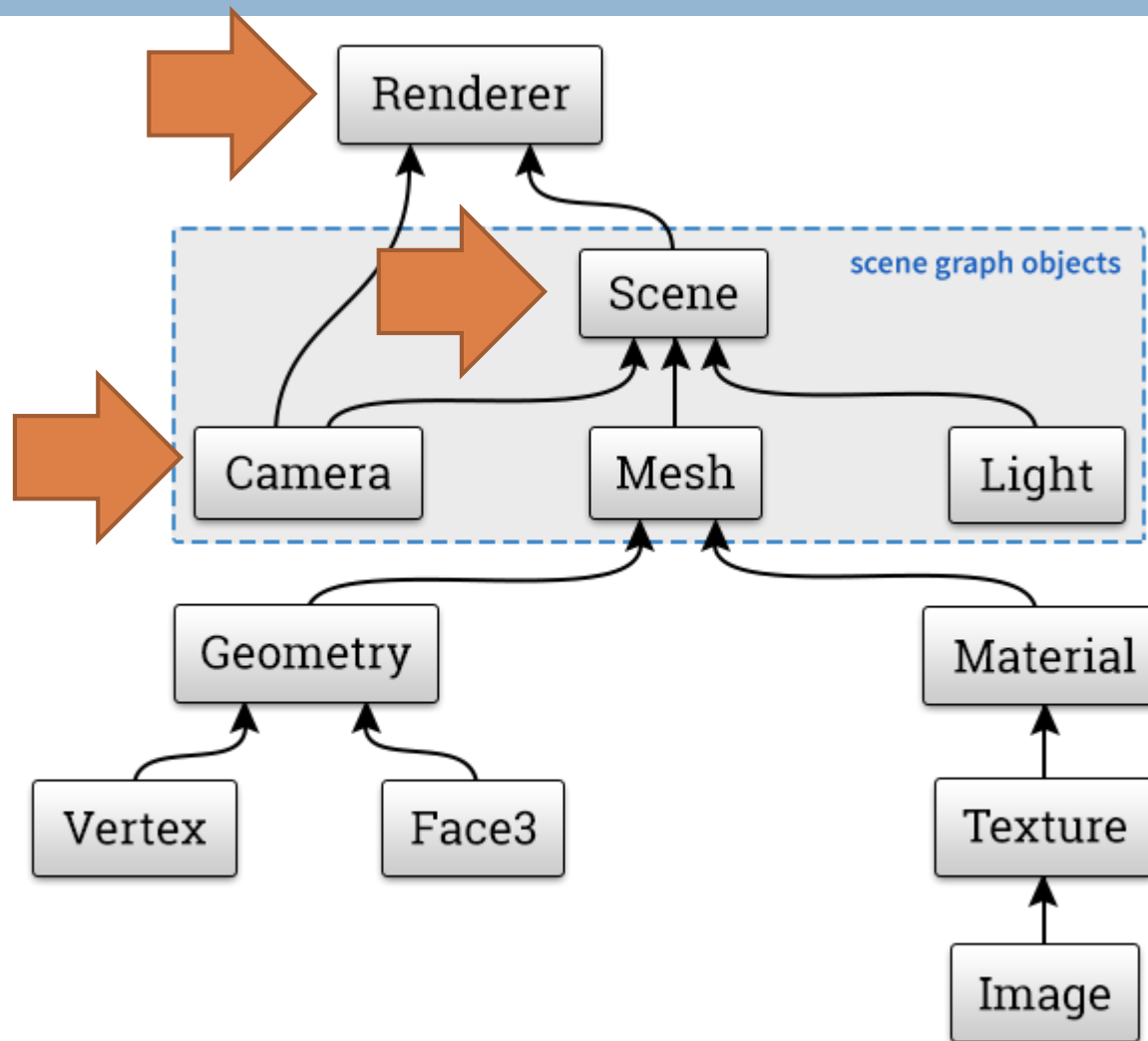
- The camera it's the point of view from which you observe the 3D world
- We use a perspective camera (orthographic cameras are available too!)
- We specify the field-of-view (75°, common for many cameras), and the image ratio
- The last two parameters are the near and far clipping planes

# Camera – side note

- Note that the camera is added to the scene
- You'll notice that this is not the case for some examples on the web
- In the past, adding the camera to the scene was not mandatory, however this omission is now deprecated



# Scene structure



# Adding an object

```
// create a cube
var geometry = new THREE.BoxGeometry( 1, 1, 1 );
var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
var cube = new THREE.Mesh( geometry, material );
scene.add( cube );
```

# Geometry

- A geometry describes the 3D **shape** of an object

```
var geometry = new THREE.BoxGeometry( 1, 1, 1 );
```

- This creates a cube (actually, a polyhedron made of triangles)

# Material

- A material defines the **appearance** of an object

```
var material = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
```

- This creates a basic green material
- Are you familiar with hexadecimal RGB color notation?

# Mesh

- A **mesh** is a 3D object: it is composed of a geometry and a material

```
var cube = new THREE.Mesh( geometry, material );  
scene.add( cube );
```

- A mesh is what you add to the scene if you want to see your object

# Render loop

```
// render loop
function renderloop() {
    requestAnimationFrame( renderloop );
    renderer.render( scene, camera );
}
renderloop();
```

- The render loop is a function that is repeatedly called each time a frame needs to be drawn

# Render

```
requestAnimationFrame( renderloop );
```

- With this line, the render() function sets itself as the function to be called to render the next frame
- This creates an endless rendering loop

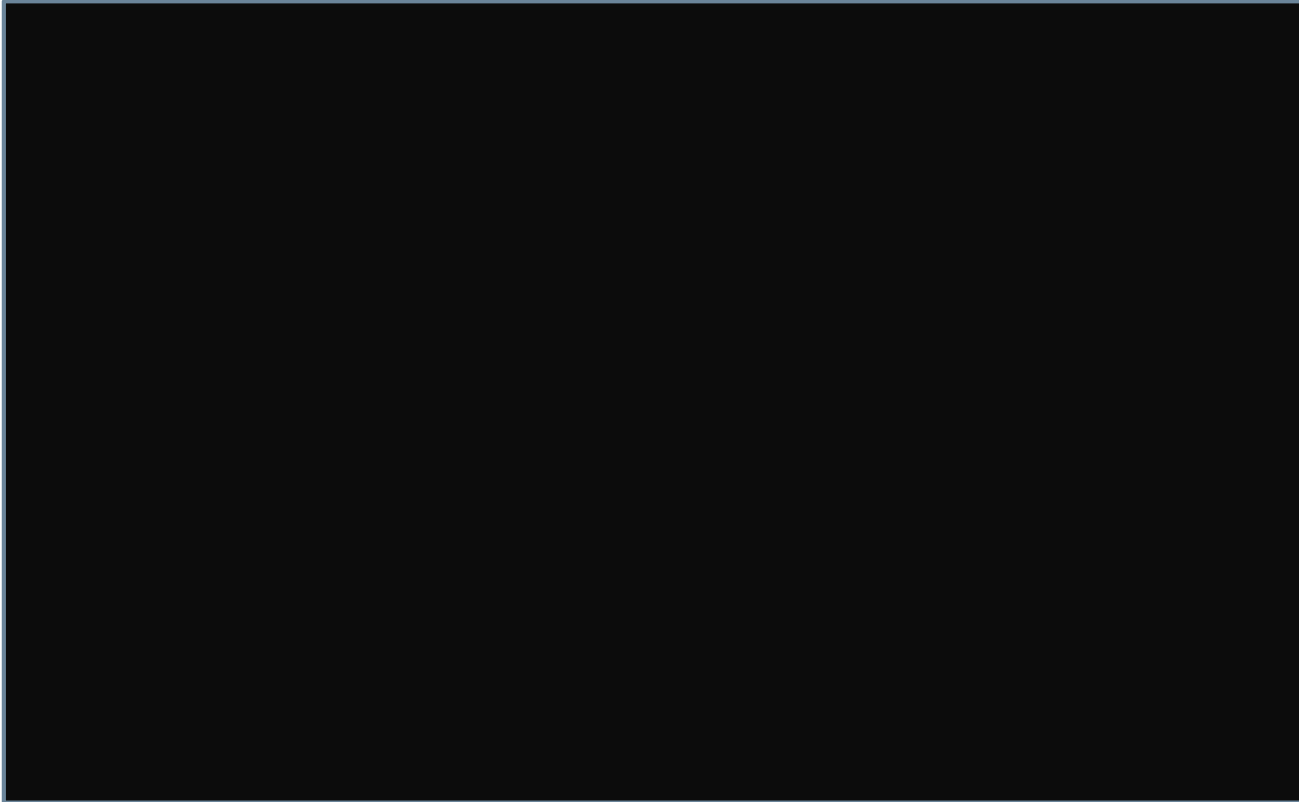
# Render

```
renderer.render( scene, camera );
```

- The second line tells the renderer object to render the scene “scene”, as seen from camera “camera”
- The output will be drawn in the canvas associated to the renderer



# Let's try it!

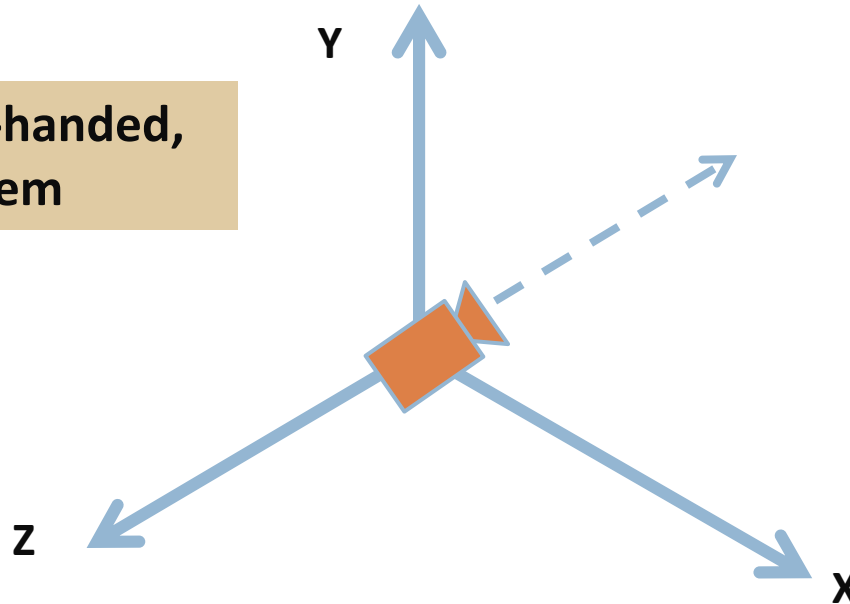


- Ops...

# Object positions

- Problem: the box and cameras are created at the origin (0,0,0). The camera lies inside the box!
- By default cameras looks toward  $-Z$  direction

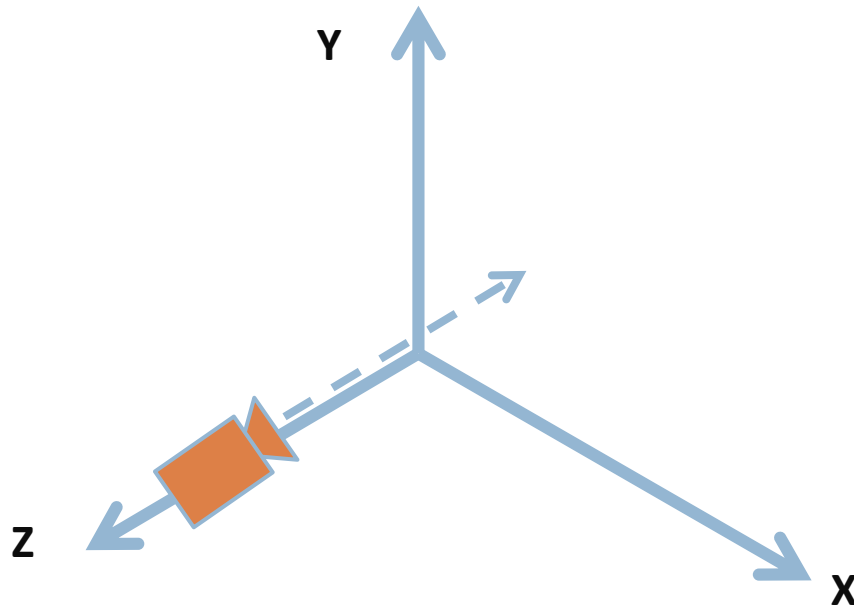
Three.js uses a right-handed, Y-up coordinate system



# Object positions

- Solution: move the camera along the positive Z axis

```
camera.position.z = 3;
```

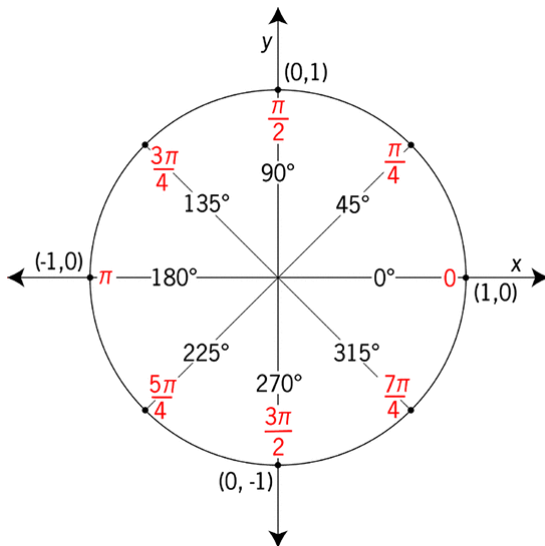


# Apply transformations

- Objects can be translated, rotated and scaled
- What's the effect of these transformations?

- ▣ `Cube.position.x = 10;`
- ▣ `Cube.rotation.z += 0.1;`
- ▣ `Cube.scale.set(2, 2, 2);`

These are properties of the Object3D object. Anything that can be inserted in a scene derives from it



Angles are expressed in **radians**

`Three.Math.degToRad()` converts degrees to radians

# Moving the object

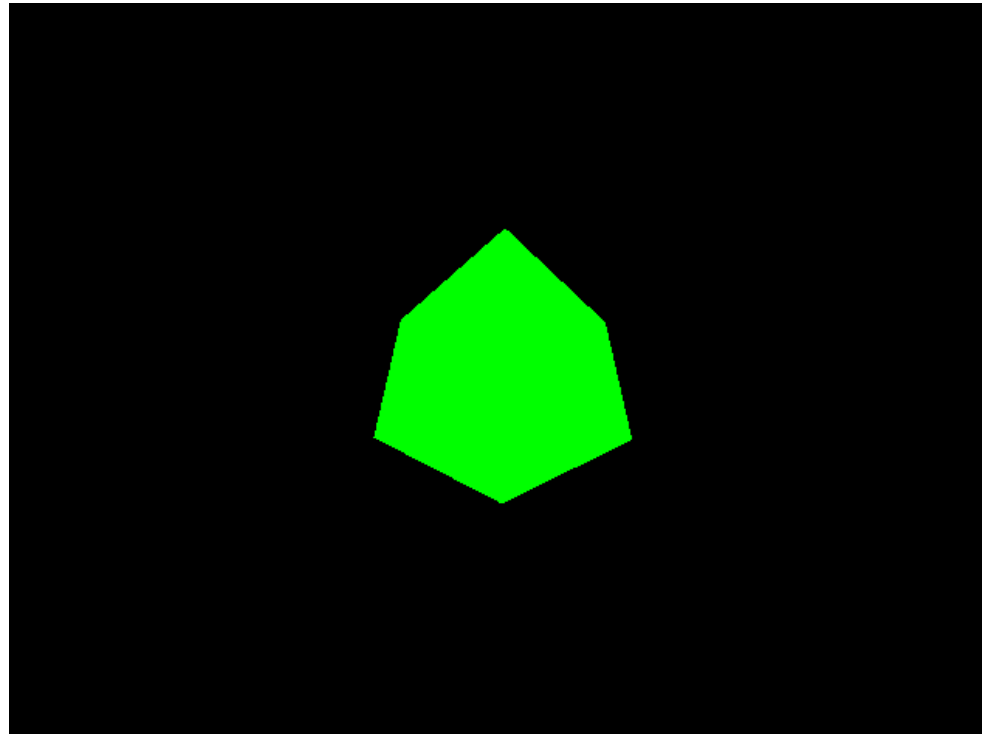
- Try this render loop:

```
// render loop
function render() {
    requestAnimationFrame( render );
    cube.rotation.z += 0.01;
    cube.rotation.x += 0.01;
    renderer.render( scene, camera );
}
render();
```



# Output

- Maybe not what we expected

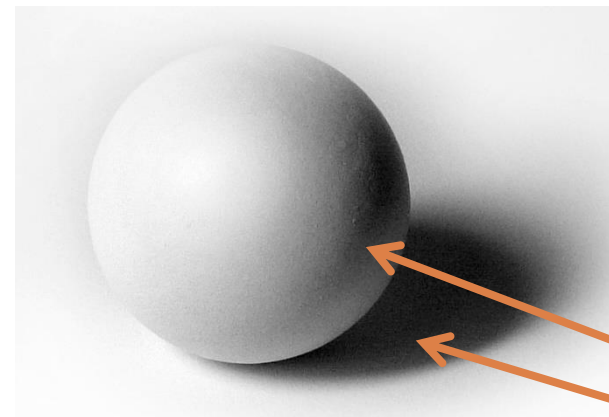


# Use a Lambert material

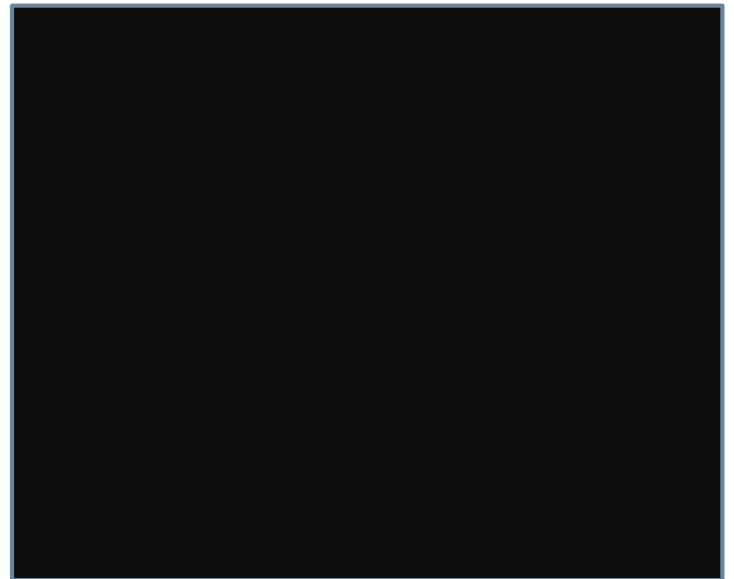
- Change the material to a MeshLambertMaterial, which can handle core shadows

```
var material = new THREE.MeshLambertMaterial({color: 0x00ff00});
```

- Everything dark again...



Core shadow  
Cast shadow



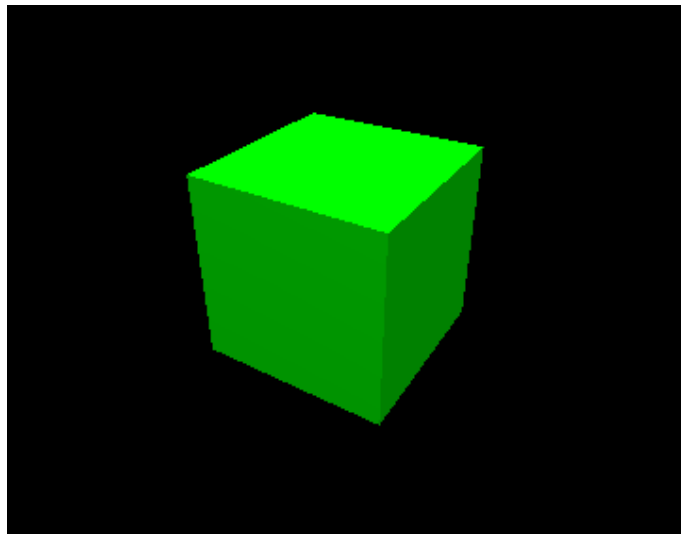
# Lights

- To see anything (except objects with BasicMaterial) you need a light
- Several types
  - ▣ Ambient light
  - ▣ Directional light
  - ▣ Hemisphere light
  - ▣ Point light
  - ▣ Spot light
- Let's use a point light and an ambient light



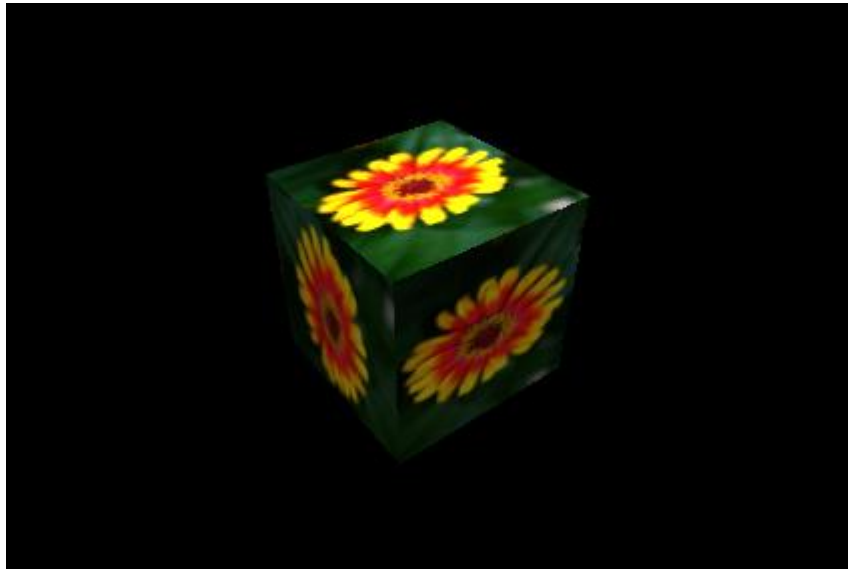
# Adding lights

```
// adding lights
var plight = new THREE.PointLight(0xffffff);
plight.position.set(0,3,3);
scene.add(plight);
var alight = new THREE.AmbientLight(0x808080);
scene.add(alight);
```



# Textures

```
// create the cube  
var geometry = new THREE.BoxGeometry( 1, 1, 1 );  
var texture = new THREE.TextureLoader().load("flower.jpg");  
var material = new THREE.MeshLambertMaterial( { map: texture } );  
var cube = new THREE.Mesh( geometry, material );  
scene.add( cube );
```



Warning: won't work locally  
(see next slide)

# Textures

- If you get the warning “image is not power of two”, either
  - ▣ Resize the image to power-of-two sides (e.g. 512x512)
  - ▣ Or add `texture.minFilter = THREE.LinearFilter;`
- More on texture mapping
  - ▣ <https://solutiondesign.com/blog/-/sdg/webgl-and-three-js-texture-mappi-1/19147>

# How to work locally

- Solution 1: use a local web server
  - ▣ <https://threejs.org/docs/index.html#manual/en/introduction/How-to-run-things-locally>
- Solution 2: force your browser to allow local file loading. Warning: potential security issues! Use it **only** for local development!
  - ▣ Chrome: <https://stackoverflow.com/questions/18586921/how-to-launch-html-using-chrome-at-allow-file-access-from-files-mode>
  - ▣ Firefox: <https://discourse.mozilla.org/t/firefox-68-local-files-now-treated-as-cross-origin-1558299/42493>

# Back to transformations

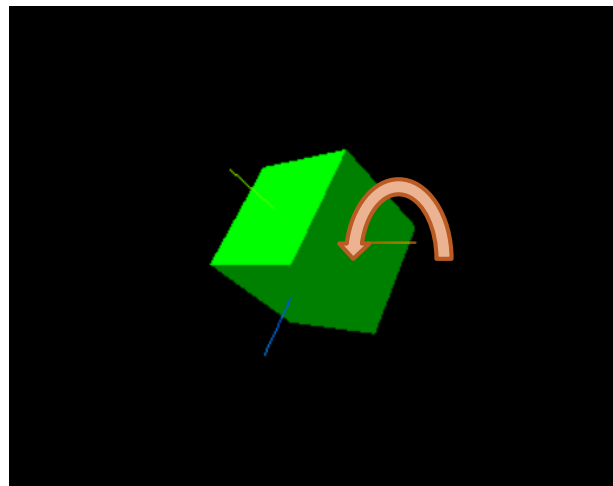
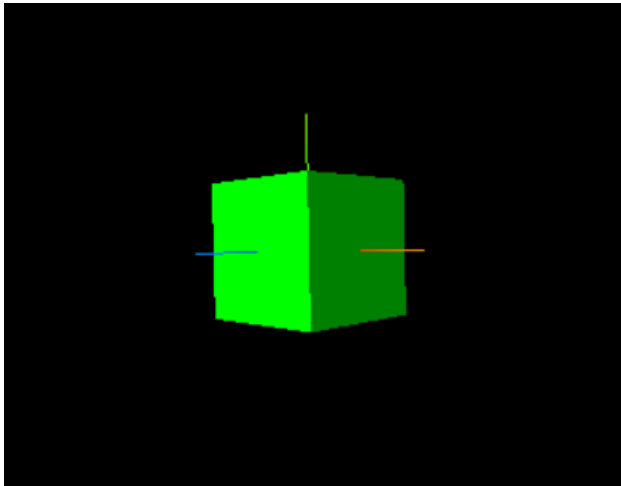
- Now that the object is clearly visible, let's go back to transformations
- **Rule n. 1:** rotations are referred to a **local axis system!**

```
cube.rotation.x = Math.PI/4;  
// render loop  
function renderloop() {  
    requestAnimationFrame( renderloop );  
    cube.rotation.z += 0.01;  
    renderer.render( scene, camera );  
}  
renderloop();
```



After the rotation around x axis, the second rotation will NOT be around *world* z axis. The x rotation also rotated the *local* z axis

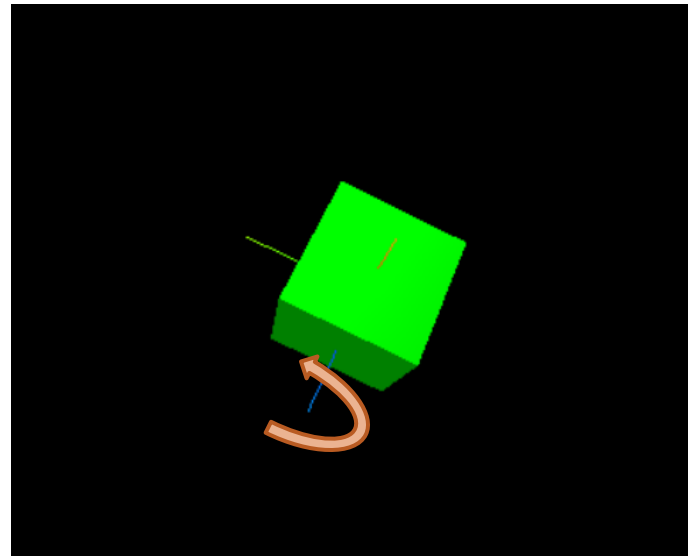
# Local rotations



Rotation around x axis  
also rotates the local  
coordinate system

Starting position  
(camera not in front of the object)

Rotation around the  
local z axis



# Order of rotations

- Now try this

```
cube.rotation.z = Math.PI/4;  
// render loop  
function renderloop() {  
    requestAnimationFrame( renderloop );  
    cube.rotation.x += 0.01;  
    renderer.render( scene, camera );  
}  
renderloop();
```

- Seems to contradict what we just said. It appears to rotate around world x axis
- The problem lies in the **rotation order**

# Order of rotations

- **Rule n. 2:**
- When you specify values in the .rotation vector, they are not applied immediately
- Rather, they are applied at render time in a specific order
- By default, x rotations are applied first. Then y rotations, finally z rotations



# Order of rotations

```
cube.rotation.z = Math.PI/4;  
// render loop  
function render() {  
    requestAnimationFrame( render );  
    cube.rotation.x += 0.01;  
    renderer.render( scene, camera );  
}
```

- Thus, even if we wrote the z rotation first, Three.js first rotates the cube around the x axis, and then around the z axis.
- Order of rotations is important!

# Order of rotations

- ❑ You can change the rotation order
- ❑ In the previous example, see the difference if you add the following line:

```
cube.rotation.order = 'ZXY';
```

# Loading external meshes

- There are several loaders to load 3D models in different formats (e.g. .OBJ files)
- Best choice: GLTF/GLB format, optimized for web apps
- How to get GLTF files:
  - ▣ Download them, e.g. from <https://sketchfab.com>
  - ▣ Convert from other formats (online converters available)
  - ▣ Use GLTF exporter plugins for 3D modeling software (Maya, Blender, 3DSMax, c4d, etc.)
  - ▣ Windows 10 users: use the 3d paint app
- Warning: same local file issues already seen for textures...

# Loading a GLTF mesh

- First we must import the loader, which is distributed as an external file:

```
<script src="GLTFLoader.js"></script>
```

- And we must change the output color space encoding, or the GLTF objects will look too dark:

```
renderer.outputEncoding = THREE.sRGBEncoding;
```



# Loading a GLTF mesh

- You can now use the loader:

```
// load the model
var loader = new THREE.GLTFLoader();
var car;
loader.load('./car.glTF', function(glTF_model){
    car = glTF_model.scene
    scene.add(car);
});
```



(the loaded object is big – roughly 800 units wide. To see it you will need to move the camera and lights outside it)



(Car model from <https://sketchfab.com>)