Chapter 4: Getting Started with WebGL and Threejs

K. Jünemann
Department Informations- und Elektrotechnik
HAW Hamburg

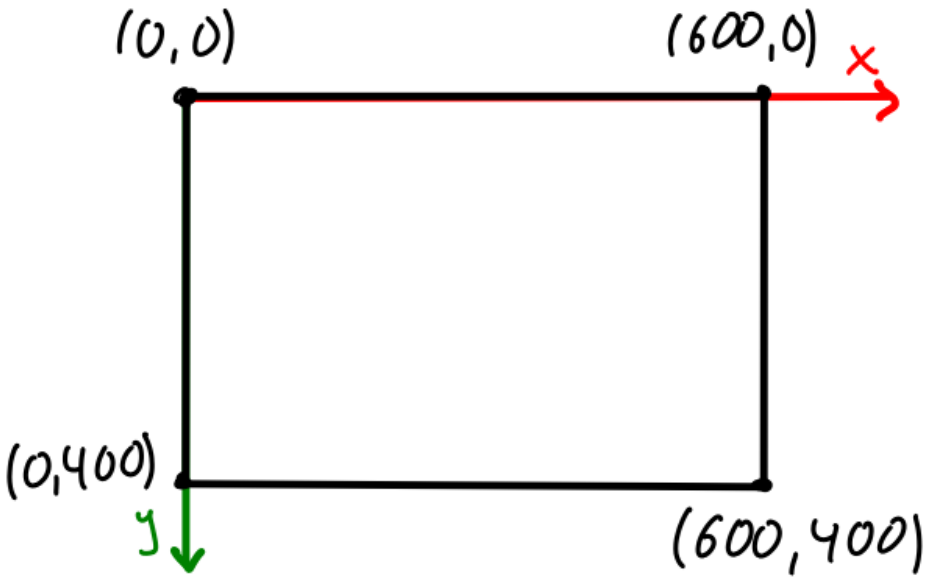
Content

The canvas tag

Provides screen space for graphics:

```
<canvas id="mycanvas"
        width="600" height="400"/>
```

- id-attribute useful for access from Javascript
- width- and height-unit: pixel



- pixel position described specified with (x, y)-coordinates

Canvas tag: Javascript API

Javascript access with DOM-API:

```
const canvas = document.getElementById("mycanvas");
```

Modern browsers provide two types of drawing contexts:

- 2D (not our topic):

```
const ctx = canvas.getContext("2d");
```

- 3D (webgl):

```
const ctx = canvas.getContext("webgl2");
```

Drawing context `ctx` provides access to corresponding graphics API.

The 2d drawing API

- Drawing state: defines *how* to draw

```
ctx.strokeStyle="red";
ctx.lineWidth=10;
ctx.lineCap="round";
...
```

- Path: defines *what* to draw

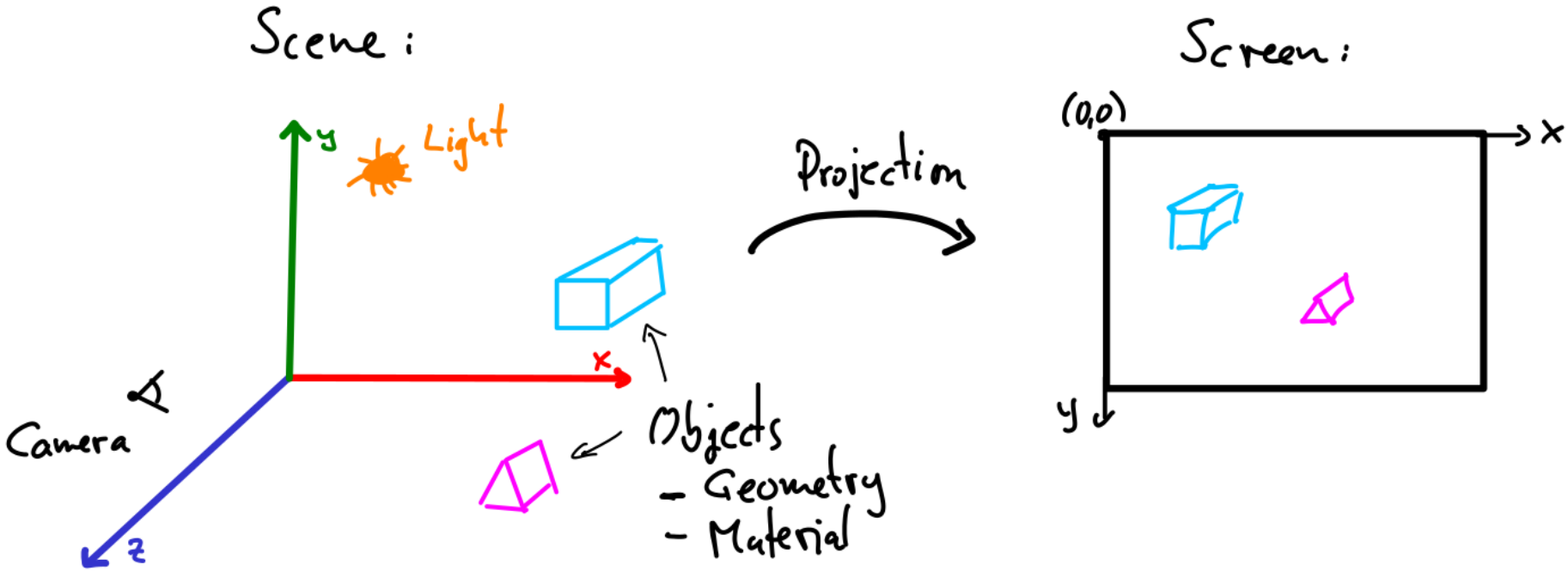
```
ctx.beginPath();
ctx.moveTo(10, 20);
ctx.lineTo(100, 200);
...
ctx.stroke();   // render path on screen
```

Good tutorial: `https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial`

WebGL

- standardized Javascript API for 3D graphics applications (initial release 2011)
- it is a specialized version of Open GL (initial release 1992)
- standardized by Khronos group, consisting of Intel, AMD, Nvidia and many others
- implemented by most 'big' browsers
- WebGL provides functionality to execute code (shader programs) on graphics hardware units
- provides no functionality that's directly useful for writing 3D applications
- WebGL is *low-level*!

The elements of a 3D application



Elements of a 3D application

1. A renderer performing the projection
2. A scene defining a coordinate system
3. A camera
4. One or more light sources
5. Objects defined through geometry and material

three.js

- WebGL asks for libraries sitting on top of it!
- One of them is `three.js`
  - initiated in 2010 by Ricardo Cabello (Mr.doob)
- Scene definition is done with Javascript, runs on CPU
- Shader code generation happens deep inside the library

A first three.js example

Step 1: Setting up a renderer

```
// At the beginning of the app
const canvas = document.getElementById("mycanvas");
const renderer = new THREE.WebGLRenderer({canvas:canvas
    });

// ...

// at the end of the app
renderer.render(scene, camera);  // do the work
```

- All pieces can be configured in 1000 ways.
- All work done in `renderer.render` function, usually inside `requestAnimationFrame` call (see chapter 6).

A first three.js example

Step 2: Define a scene

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

- A scene is a container to which all parts of a 3D application have to be *added*.
- The scene has to be passed to the `render` function to specify *what* to render.
- A scene defines an (invisible) coordinate system: the world coordinate system. Optionally show coordinate system:

```
const axesHelper = new THREE.AxesHelper( 5 );
scene.add( axesHelper );
```

A first three.js example

Step 3: Define a camera

```
const camera = new THREE.PerspectiveCamera( 75,
               canvas.width / canvas.height,
               0.1, 500 );
```

- The camera has to be passed to the `render` function to specify *how* to render.
- three.js contains various camera models (see chapter 9).
- Optionally add mouse control:

```
const controls = new OrbitControls( camera, renderer
    .domElement);
// ...
// in render loop
controls.update();
```

A first three.js example

Step 4: Add light source, e.g.

```
const light = new THREE.PointLight();
light.intensity = 200.0;
light.position.set(0,0,10);
scene.add(light);
```

- Light sources are invisible!
- `three.js` contains various types of light (see chapter 10).

A first three.js example

Step 5: Add objects

```
const knotGeometry = new THREE.TorusKnotGeometry
      (5,1,160,100);
const knotMat = new THREE.MeshStandardMaterial
      ({color:'#55ff22',
        metalness:0.5,
        roughness:0.2});
const knot = new THREE.Mesh(knotGeometry, knotMat);
scene.add(knot);
```

- A *mesh* object consists of a geometry and a material
  - the geometry defines the shape of a object (see chapter 5)
  - the material describes how the object looks like (see chapter 10)
- Besides mesh objects `three.js` contains other types of objects.