Graphics and Visualization

WebGL Lab

1. Overview

This lab session shows you how to use ThreeJS, a JavaScript 3D library based on WebGL. You will create a first person viewing control and use ray casting to pick an object in 3D space.
WebGL: https://www.khronos.org/webgl
ThreeJS: https://threejs.org
To check WebGL compatibility for your browser: https://caniuse.com/#feat=webgl or checking at runtime: https://threejs.org/docs/#manual/introduction/WebGL-compatibility-check

2. Objectives

- Able to create 3D scene and add some 3D objects in ThreeJS
- Object picking using ray casting technique
- Control camera view

3. Project

a. Preparation

- A computer
- ThreeJS lastest version (this lab uses r89)
- An up-to-date browser (Google Chrome 64 64-bit is used in this lab)

b. Part 1: Creating a scene

All you need is located here: https://threejs.org/docs/index.html#manual/introduction/Creating-a-scene

Basically, we have to init a scene (THREE.Scene), a camera (e.g., THREE.PerspectiveCamera), a renderer (THREE.Renderer) in a JavaScript function

```
var gScene, gCamera, gRenderer;
// Make it global because we need them
```

```
// not only in init function but also other functions

// Reserved for ray casting
var gMouse = new THREE.Vector2();

init(); // Execute Init function

function init() {

// Add an element which contains our renderer's DOM element
var canvasContainer = document.createElement("div");
document.body.appendChild(canvasContainer);

gScene = new THREE.Scene();
gCamera = new THREE.PerspectiveCamera(
              70, // Fov
              window.innerWidth/window.innerHeight, // Aspect
              1, // Near
              1000 // Far
              );
gCamera.position.z = 1.5; // Moving the camera in z axis

// Add light in case of using MeshLambertMaterial, otherwise
// can not see anything
// If MeshBasicMaterial is used, it isn't necessary
var light = new THREE.DirectionalLight(0xffffff, 1);
light.position.set(1, 1, 1).normalize();
gScene.add(light);

gRenderer = new THREE.WebGLRenderer();
gRenderer.setSize(
                  window.innerWidth, // Width
                  window.innerHeight // Height
                  );

canvasContainer.appendChild(gRenderer.domElement);
// Put DOM element to declared container

}
```

So now, initialization have been done but we have not rendered anything. The canvas needs to be updated frame by frame by animate loop.

```
function animate() {
```

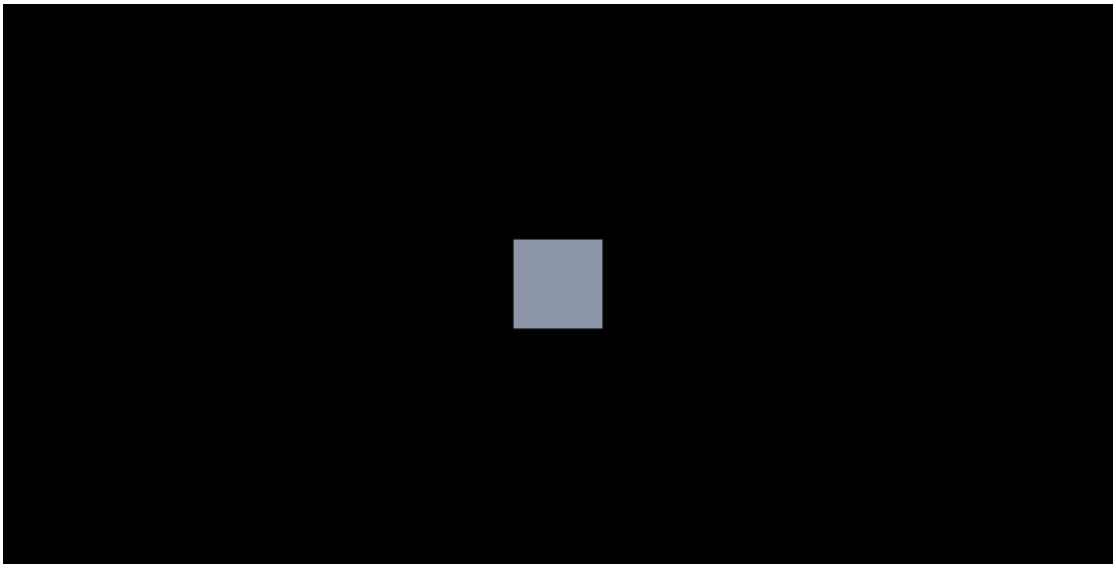
```
      requestAnimationFrame(animate);
      gRenderer.render(gScene, gCamera);
}
animate(); // Trigger render loop
```

c. Part 2: Adding 3D object

To create an object, we should define a geometry, a material and a mesh in init function as following:

```
var geometry = new THREE.BoxGeometry(
                                    1,  // Width
                                    1,  // Height
                                    1); // Depth
var material = new THREE.MeshLambertMaterial(
                          { color: Math.random() * 0xffffff });
var cube = new THREE.Mesh(geometry, material);
cube.position.z = -30; // Moving out this cube in z axis

gScene.add(cube);
```

You may see your browser's screen likes the bellow image, color will be different because we generate ramdomly:



d. Part 3: Moving camera

It is very simple by setting camera position uses ThreeJS's interfaces:
Object3D.**rotateX**, Object3D.**rotateY**, Object3D.**rotateZ**, Object3D.**translateX**, Object3D.**translateY**, Object3D.**translateZ** because THREE.Perspective camera is based on THREE.Object3D.
To create first person viewing control, we need to add some event listeners so as to change camera position according to user's input. (refer to appendix)

```
//For instance, we have already set 'keydown' event listener
function onKeyDown(e){
e.preventDefault();
```

```
switch (e.keyCode) {
      case 38: // Arrow up key
            gCamera.position.x -= 1;
            // gCamera.translateX(-1);
            break;
      case 40: // Arrow down key
            gCamera.position.x += 1;
            // gCamera.translateX(1);
            break;
}
}
```

e. Part 4: Picking object

In ThreeJS, THREE.Raycaster is implemented to support this task. However, you should understand mouse coordinate and image screen coordinate because conversion is required

Firstly, we should add mouse down event listener

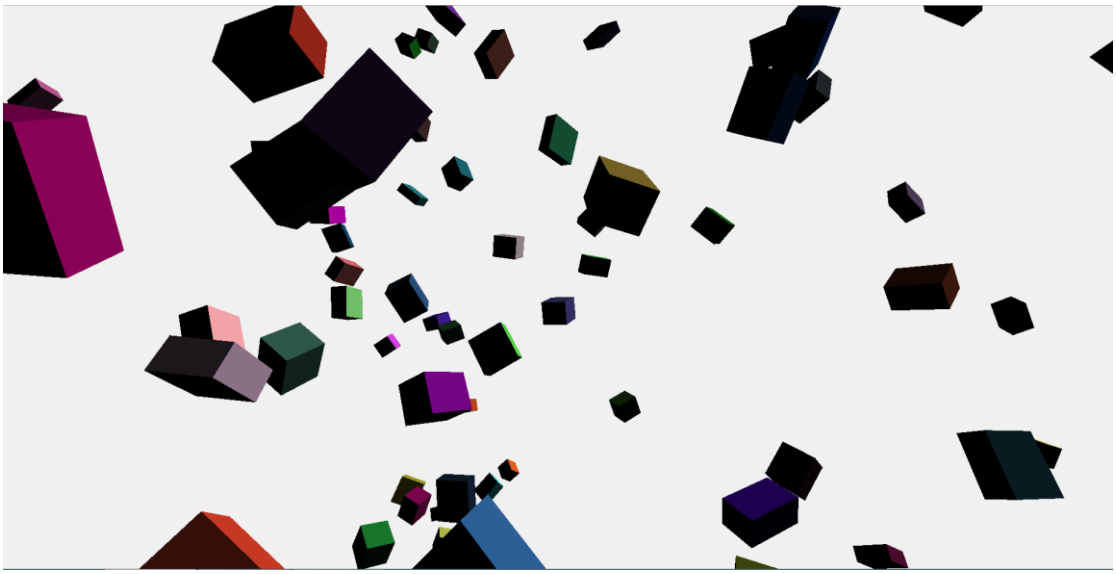
```
function init() {
...
document.addEventListener('mousedown', onMouseDown, false);
...
}
```

And then, onMouseDown function must be declared, we will change color of the picked object randomly.

```
function onMouseDown (event) {

event.preventDefault();

// Conversion from mouse coordinate to image screen coordinate
gMouse.x = (event.clientX / window.innerWidth) * 2 - 1;
gMouse.y = -(event.clientY / window.innerHeight) * 2 + 1;

// Find intersections
var raycaster = new THREE.Raycaster();
raycaster.setFromCamera(gMouse, gCamera);
var intersects = raycaster.intersectObjects(gScene.children);
if (intersects.length > 0) {
intersects[0].object.material.color.setHex(Math.random()* 0xffffff );
// First element is the nearest object in front of camera
}
}
```

f. The last part: Do it youself

Let's modify it for more interesting things. You should add more objects by generating randomly position, scale, rotation.



4. Appendix:

Mouse move event

```
function init() {
...
document.addEventListener('mousemove', onMouseMove, false);
...
}

function onMouseMove(event) {
event.preventDefault();
var movementX = event.movementX;
var movementY = event.movementY;
... // Rotating, translating camera position
    // For instance,
    // gCamera.rotateY(0.002*movementX);
}
```

Key pressed event

```
var gMoveForwardReq = false,
    gMoveLeftReq = false,
    gMoveRightReq = false,
    gMoveBackwardReq = false;

function init() {
...
document.addEventListener('keyup', onKeyUp, false);
```

```
document.addEventListener('keydown', onKeyDown, false);
...
}

function onKeyDown (event) {
      switch(event.keyCode) {

      case 38: // up
      case 87: // w
            gMoveForwardReq = true;
            break;

      case 37: // left
      case 65: // a
            gMoveLeftReq = true;
            break;

      case 40: // down
      case 83: // s
            gMoveBackwardReq = true;
            break;

      case 39: // right
      case 68: // d
            gMoveRightReq = true;
            break;

      }
}

function onKeyUp (event) {
      switch(event.keyCode) {

      case 38: // up
      case 87: // w
            gMoveForwardReq = false;
            break;

      case 37: // left
      case 65: // a
            gMoveLeftReq = false;
            break;

      case 40: // down
      case 83: // s
```

```
            gMoveBackwardReq = false;
            break;

      case 39: // right
      case 68: // d
            gMoveRightReq = false;
            break;

      }
}

var gPrevTime = performance.now();
var gDelta = performance.now();

function animate() {

var time = performance.now();
gDelta = (time - gPrevTime) / 10;

if (gMoveRightReq) {
// Change camera position or rotation based on gDelta
}
if (gMoveLeftReq) {
// Change camera position or rotation based on gDelta
}
if (gMoveForwardReq) {
// Change camera position or rotation based on gDelta
}
if (gMoveBackwardReq) {
// Change camera position or rotation based on gDelta
}

...

gPrevTime = time;

}
```

Proof of translating mouse coordinates to camera coordinates

We assumed that WebGL canvas is full page. Width and height are window.innerWidth, window.innerHeight respectively

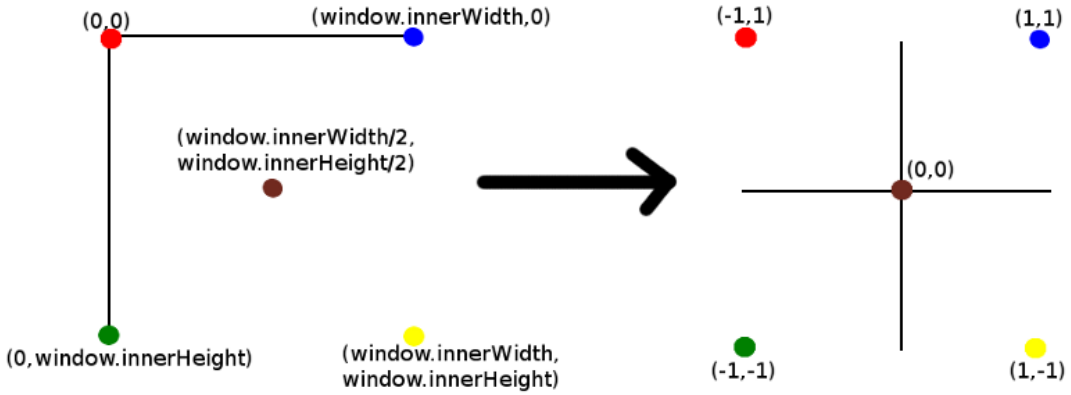


Figure 1. Top-left coordinates to the cartesian coordinates (http://barkofthebyte.azurewebsites.net)

Therefore, we have the following mapping:

$$(x', y') = (\frac{2x}{innerWidth} - 1, -\frac{2y}{innerHeight} + 1)$$