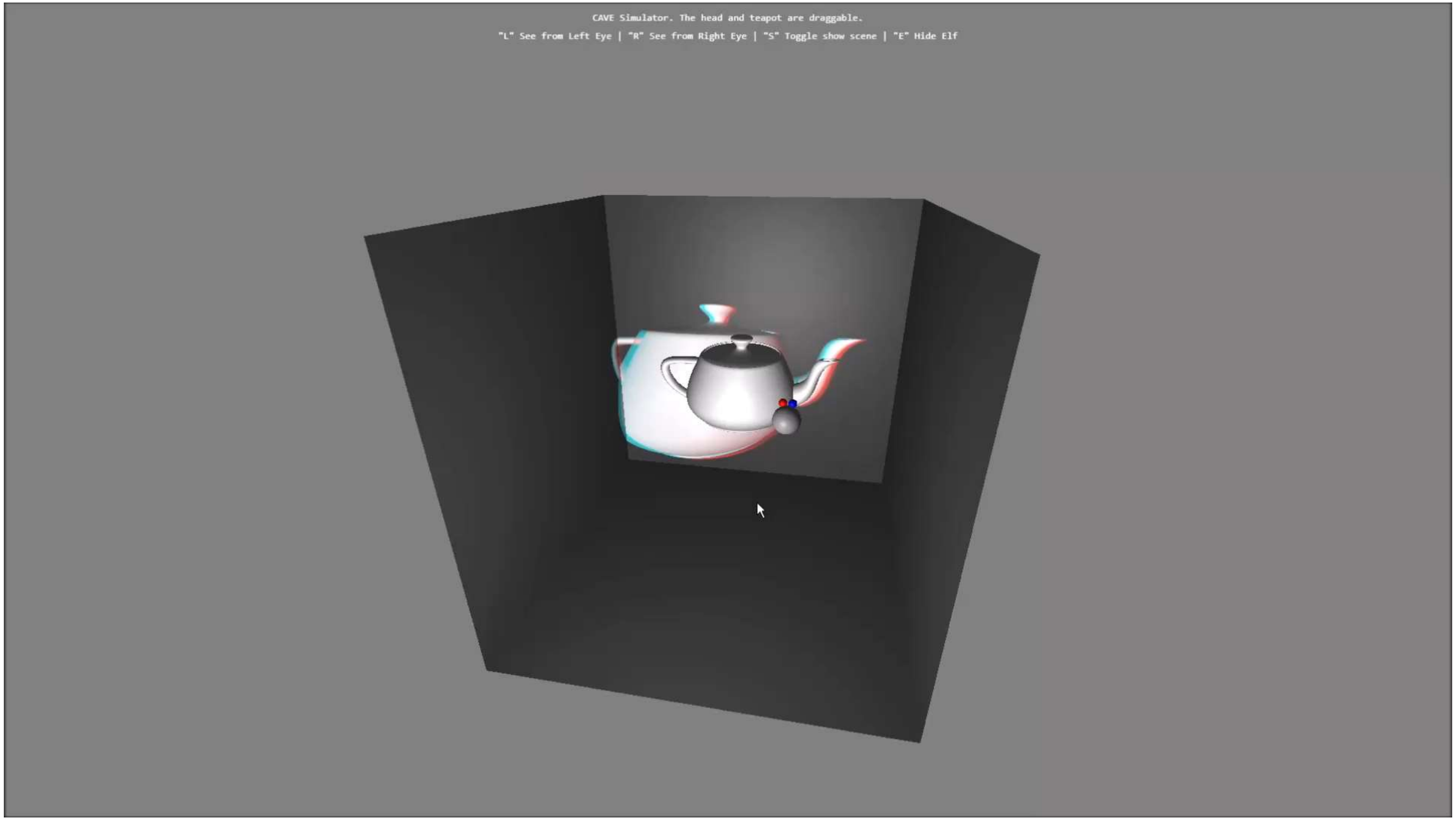
Stereoscopy project

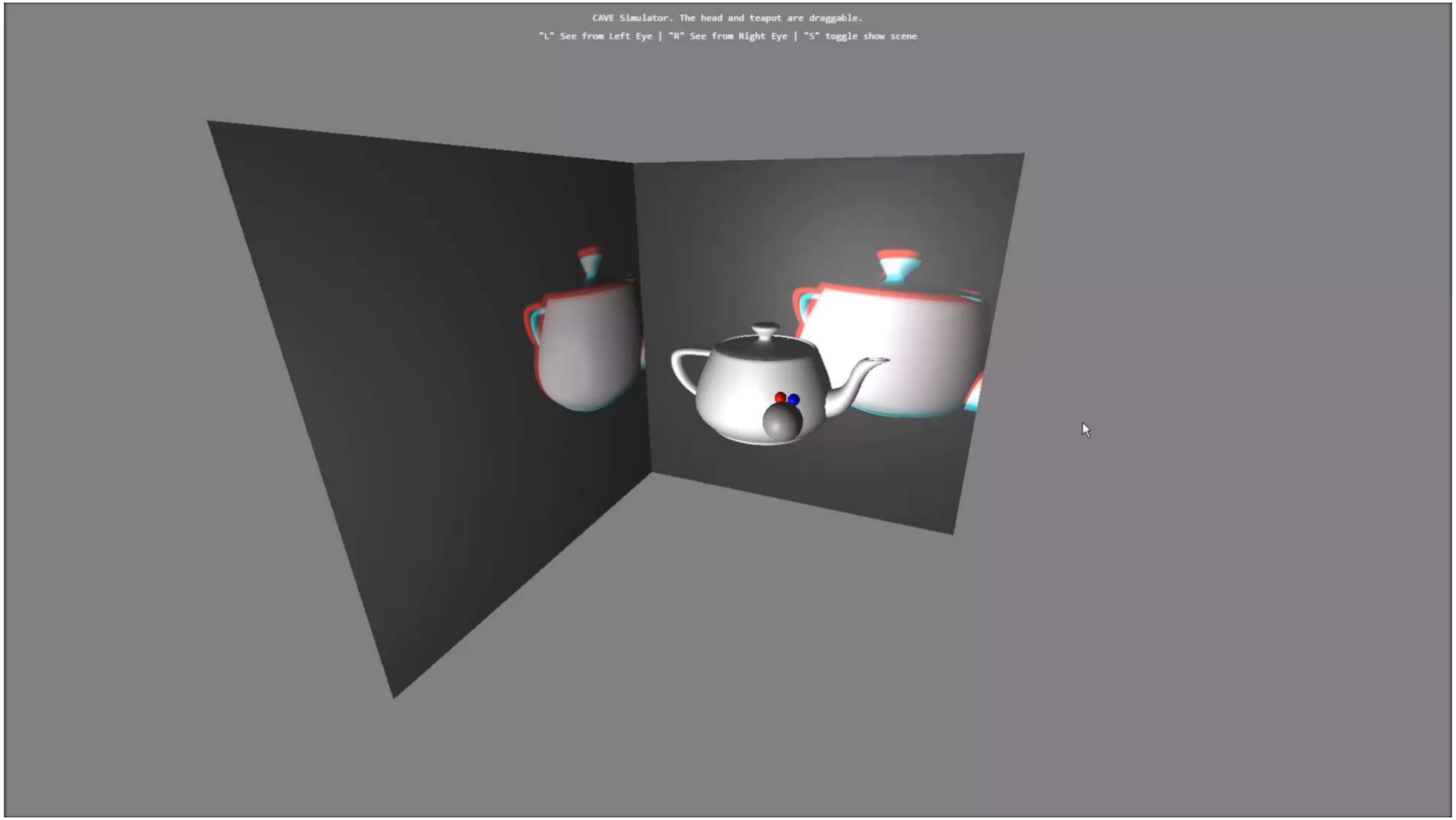
C. Andujar
Nov 2024

Project description

Goal – Implement a CAVE simulator

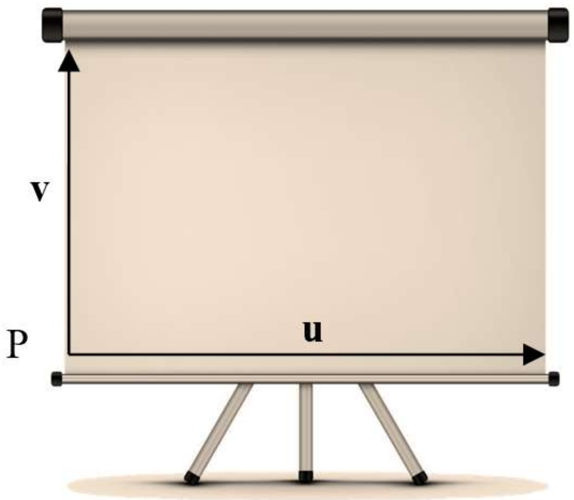


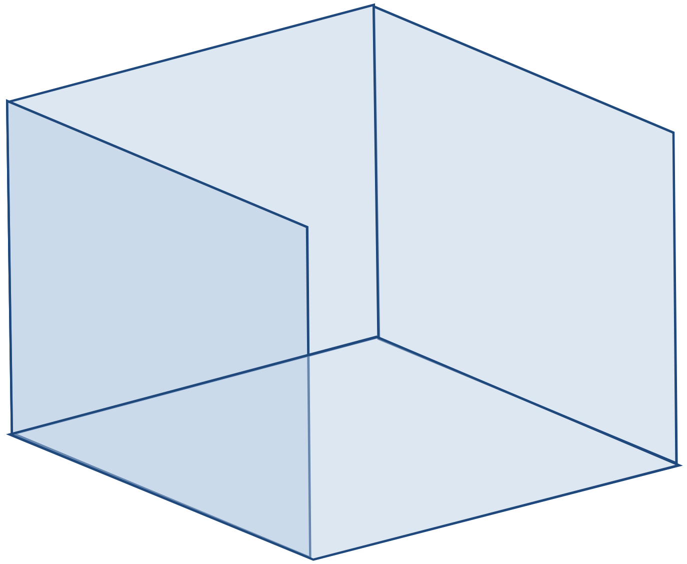
Stating point: threeJS example



Display surface

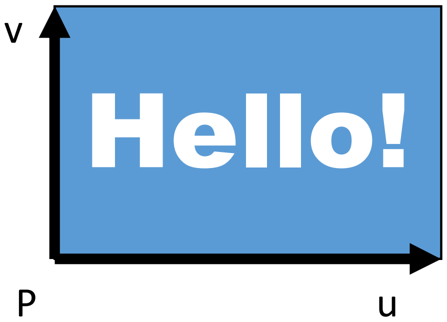
- Projection-based systems consist of a collection of display surfaces.
- Example: CAVE with 4 display surfaces (3 walls + floor).
- Each rectangular display surface can be represented with a point **P** and two orthogonal vectors **u**, **v**.

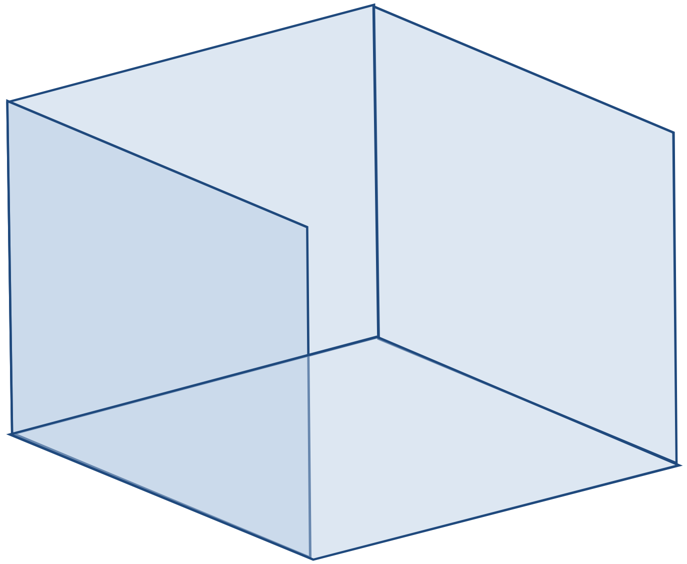




Display surface - assumptions

- **P** corresponds to the origin of the OpenGL window coordinate system, i.e. the projector will map the pixel at (0,0) to P,
- **u** and **v** vectors correspond respectively to the x and y axes of the OpenGL window coordinate system.

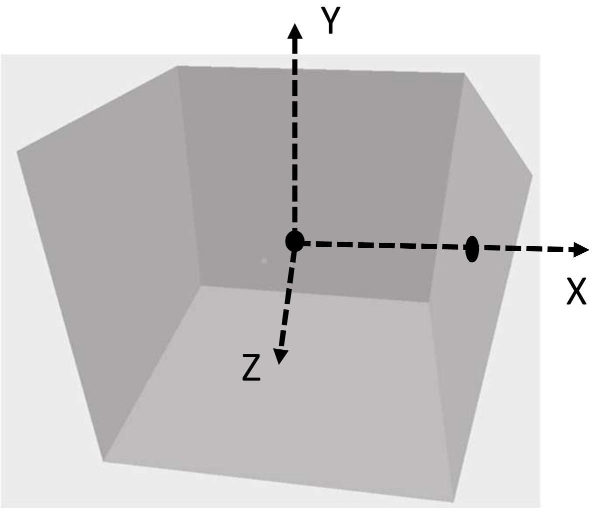




Display surface – representation (JS)

```
class DisplaySurface
{
    constructor(name, origin, u_vector, v_vector)
    {
        this.name = name;
        this.origin = origin;
        this.u = u_vector;
        this.v = v_vector;
    }
    ...
```

Example: 3x3x3m CAVE with 4 screens




```
var frontScreen = new DisplaySurface("Front",
   new THREE.Vector3(-150.0, -150.0, -150.0),
   new THREE.Vector3( 300.0,    0.0,    0.0),
   new THREE.Vector3(   0.0,  300.0,    0.0));

var leftScreen = new DisplaySurface("Left",
   new THREE.Vector3(-150.0, -150.0,  150.0),
   new THREE.Vector3(   0.0,    0.0, -300.0),
   new THREE.Vector3(   0.0,  300.0,    0.0));
```

…

Coordinate system: at the middle of the CAVE (see Figure). Length units: cm

Part 1 – view matrix

View matrix

Write a function that, given

- a display surface and
- the (x,y,z) coordinates of the eye

(both expressed w.r.t. the chosen coordinate system), returns

- the 4x4 viewing matrix to be used to render a scene onto the display surface from the given eye position.

View matrix

```
viewMatrix(eye)
{
        // to be written by you!
        var target = new THREE.Vector3(0,0,0);
        var upVector = new THREE.Vector3(0,1,0);
        var mat = new THREE.Matrix4();
        mat = mat.lookAt(eye, target, upVector); // this lookAt creates only rotation!
        var translate = new THREE.Matrix4().makeTranslation(-eye.x, -eye.y, -eye.z);
        mat = mat.multiplyMatrices(mat, translate);
        return mat;
}
```

View matrix

For example, for eye = (50, 20, 100), and the front display above, your code should return the following matrix:

```
1        0        0        -50
0        1        0        -20
0        0        1       -100
0        0        0          1
```

Part 2 – projection matrix

Projection matrix

Write a function that, given

- a display surface,
- the (x,y,z) coordinates of the eye, and
- znear, zfar clipping planes,

returns

- the 4x4 projection matrix to be used to draw a scene onto the display surface from the given eye position.

Projection matrix

```
projectionMatrix(eye, znear, zfar)
{
  // to be written by you!
  var left = -1;
  var right = 1;
  var bottom = -1;
  var top = 1;
  return new THREE.Matrix4().makePerspective(
         left, right, top, bottom, znear, zfar); // order!
}
```

Part 3 – CAVE simulator

CAVE simulator features

- Define a collection of display surfaces, and (x,y,z) positions for the user's left and right eye.
- Draw the display surfaces as quads/boxes.
- Draw the position of left and right eyes (e.g. as spheres)
- Draw the projection of the scene onto each display surface, for the left (in red) and right (blue) eyes.
- Provide some interaction with the scene. The user should be able to move the left/right eyes, change iod, and move/scale the scene.
- You might want to create a scene with objects at multiple locations.

Notice that a subset of the features above are already implemented in the provided ThreeJS example

Part 4 – WebXR support


three.js
docs
examples
xr
reset
exit
vr / handinput / pointerdrag
vr / handinput / pressbutton
vr / layers
three.js vr - handinput - point and drag
(Oculus Browser 15.1+)
reset
exit
ENTER VR

ThreeJS Example

https://threejs.org/

three.js docs examples

en

Manual

Getting Started
Creating a scene
Installation
Browser support
WebGL compatibility check
How to run things locally
Typescript setup
Drawing lines
Creating text
Loading 3D models
FAQ
Useful links

Next Steps
How to update things
How to dispose of objects
How to create VR content
How to use post-processing
Matrix transformations
Animation system

Build Tools
Testing with NPM

Reference

Creating a scene

The goal of this section is to give a brief introduction to three.js. We will start by setting up a scene, with a spinning cube. A working example is provided at the bottom of the page in case you get stuck and need help.

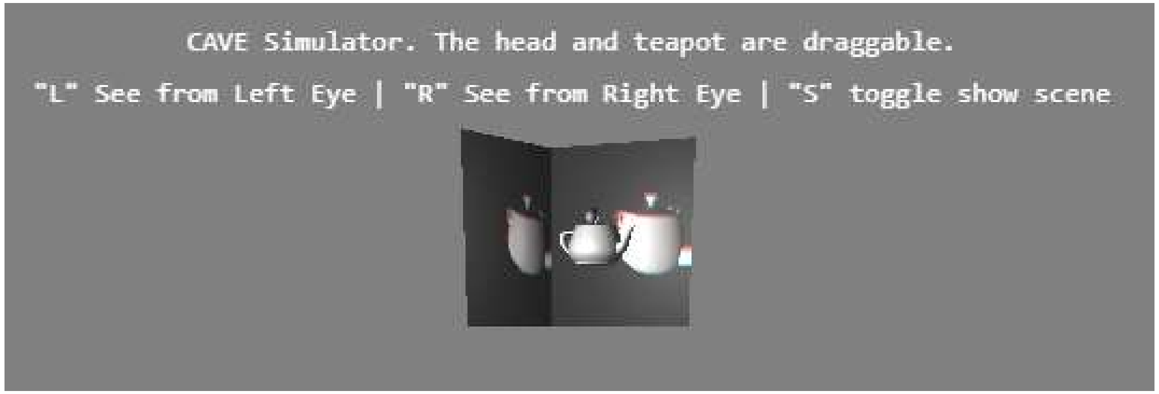
Before we start

Before you can use three.js, you need somewhere to display it. Save the following HTML to a file on your computer, along with a copy of three.js in the js/ directory, and open it in your browser.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>My first three.js app</title>
        <style>
            body { margin: 0; }
            canvas { display: block; }
        </style>
    </head>
    <body>
        <script src="js/three.js"></script>
        <script>
            // Our Javascript will go here.
        </script>
    </body>
</html>
```

That's all. All the code below goes into the empty <script> tag.

HTML part




```
<html>
<head>
 <meta charset="utf-8">
 <title> CAVE simulator </title>
 <link type="text/css" rel="stylesheet" href="main.css">
</head>

<body>
 <div id="info"> CAVE Simulator. The head and teapot are draggable.
 <br /> "L" See from Left Eye | "R" See from Right Eye | "S" toggle show scene <br />
 </div>
 <script>
   …

 </script>
</body>
</html>
```

JS part: different options (as script, as module)

```
import * as THREE from
'https://threejs.org/build/three.module.js';
import { OrbitControls } from
'https://threejs.org/examples/jsm/controls/OrbitContr
ols.js';
import { DragControls } from
'https://threejs.org/examples/jsm/controls/DragContro
ls.js';
import { TeapotBufferGeometry } from
'https://threejs.org/examples/jsm/geometries/TeapotBu
fferGeometry.js';
```

JS part

```
class DisplaySurface
{
  constructor(name, origin, u_vector, v_vector)
  { … }
  viewMatrix(eye)
  {
       // to be written by you!
  }
  projectionMatrix(eye, znear, zfar)
  {
       // to be written by you!
  }
}
```

JS part

```
var renderer, scene, camera;
var displaySurfaces, displaySurfaceScene
var displaySurfaceTargets;
var eyeCenter, eyeScene;
var orbitControl;
```

JS part

```
function createRenderer()
{
  renderer = new THREE.WebGLRenderer();
  renderer.autoClear = false;
  renderer.setSize(innerWidth, innerHeight);
  document.body.appendChild(renderer.domElement);
}
```

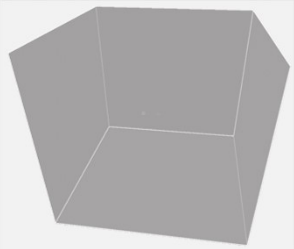
JS part

```
function createDisplaySurfaces()
{
    displaySurfaces = [];
    // FRONT SCREEN
    var frontScreen = new DisplaySurface("Front",
        new THREE.Vector3(-150.0, -150.0, -150.0),
        new THREE.Vector3( 300.0,    0.0,    0.0),
        new THREE.Vector3(   0.0,  300.0,    0.0));
    displaySurfaces.push(frontScreen);
    ...
}
```

JS part

```
function createDisplaySurfaceTargets()
{
   const SIZE = 1024;  // texture resolution
   displaySurfaceTargets = [];

   for (var v of displaySurfaces)
    displaySurfaceTargets.push(
     new THREE.WebGLRenderTarget(SIZE, SIZE));
}
```

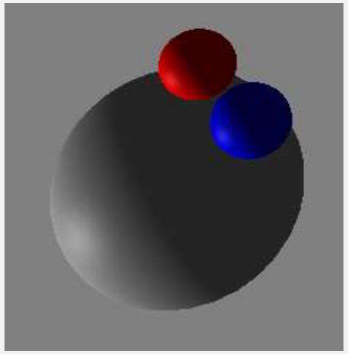
JS part



```
function createDisplaySurfaceScene()
{
  displaySurfaceScene = new THREE.Scene();
  // add display surfaces
  for (var [index, displaySurface] of displaySurfaces.entries())
  {
    ...
    var geom = new THREE.BoxGeometry(u.length(), v.length(), 0.01);
    var mat = new THREE.MeshPhongMaterial( {map:display...Targets[index].texture});
    var cube = new THREE.Mesh( geom, mat);
    ...
    displaySurfaceScene.add(cube);
  }
  createLights(displaySurfaceScene);
}
```

JS part

```
function createEyeScene()
{
    var IPD = 6.8;
    eyeCenter = new THREE.Vector3(50, 20, 50);
    // eye positions relative to the head
    var eyeL = new THREE.Vector3( - IPD/2, 10, -6);
    var eyeR = new THREE.Vector3( + IPD/2, 10, -6);
    eyeScene = new THREE.Scene();
    // add sphere representing head
    var geometry = new THREE.SphereGeometry( 10, 32, 22 );
    var material = new THREE.MeshPhongMaterial( { color: 0xaaaaaa } );
    var head = new THREE.Mesh( geometry, material );
    eyeScene.add(head);
    // add spheres representing L/R eyes
    …
}
```



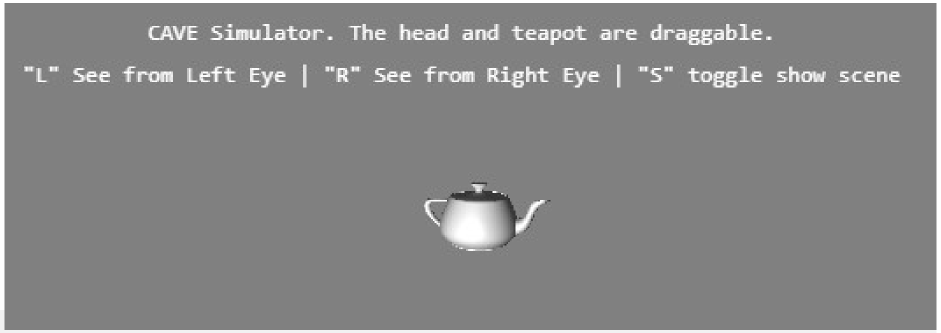
JS part

```
function createScene()
{
    scene = new THREE.Scene();

    var geometry = new TeapotBufferGeometry( 40, 15);
    var material = new THREE.MeshPhongMaterial( { color: 0xffffff } );
    var teapot = new THREE.Mesh(geometry, material);
    teapot.name = "Teapot";
    teapot.position.z -= 70;
    scene.add( teapot );

    createLights(scene);
}
```

JS part

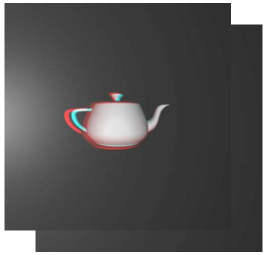
```
function createCamera()
{
    camera = new THREE.PerspectiveCamera( 75,
             window.innerWidth/window.innerHeight, 0.1, 10000 );
    camera.position.set( 100, 100, 300 );
    camera.lookAt( 0, 0, 0 );
}
```

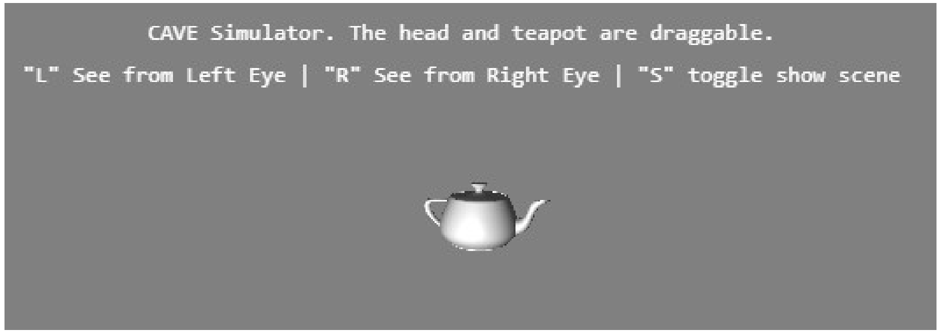
JS part




```
var animate = function () {
    ...
    // 1. render scene objects
    renderer.clear();
    renderer.render(scene, camera);
    ...
```

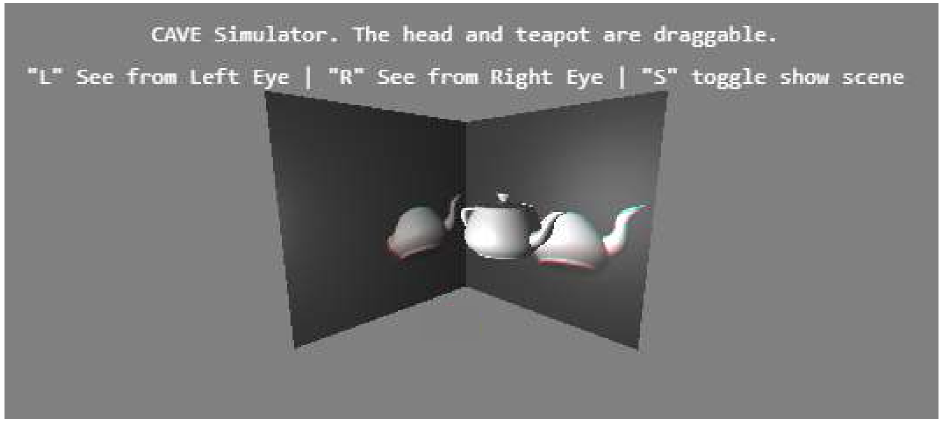
JS part






```
// 2. render scene objects onto a texture, for each target
for (let [index, displaySurface] of displaySurfaces.entries())
{
    renderer.setRenderTarget(displaySurfaceTargets[index]);
    renderer.clear();
    // left eye on RED channel
    gl.colorMask(1, 0, 0, 0);
    var view = displaySurface.viewMatrix(eye);
    var proj = displaySurface.projectionMatrix(eye, 1, 10000);
    var leftCamera = cameraFromViewProj(view, proj);
    renderer.render(scene, leftCamera);
    // right eye on GREEN, BLUE channels
    …
}
renderer.setRenderTarget(null);
```

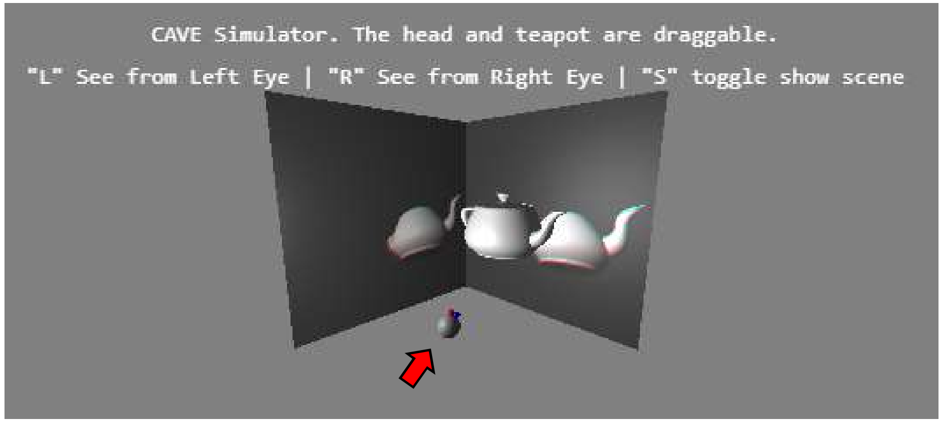
JS part




```
...
// 3. render display surfaces as (textured) quads
renderer.render(displaySurfaceScene, camera);
```

```
function createDisplaySurfaceScene()
{
  for (var [index, displaySurface] of displaySurfaces.entries())
  {
    var mat = new THREE.MeshPhongMaterial(
                        {map:display...Targets[index].texture}
                                  );
    displaySurfaceScene.add(cube);
  }
}
```

JS part




```
    // 4. render eyes
    renderer.render(eyeScene, camera);
}
```

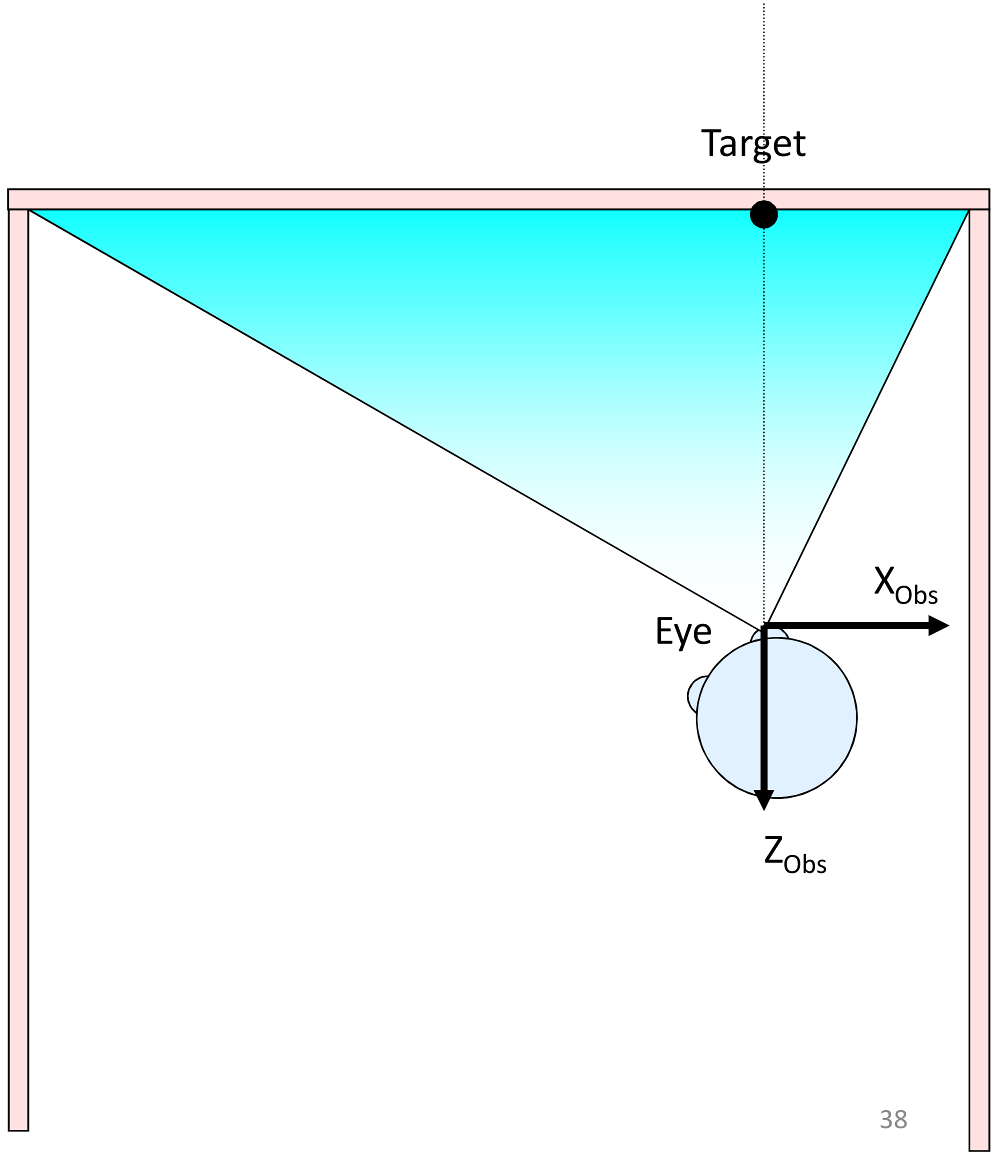
Computing the View matrix

View matrix

- First, compute **eye, target, upVector**
- Then, use lookAt + translation

Hints:

- The vector **eye – target** must be perpendicular to the display surface.
- User the cross product to compute a normal vector of the display surface.



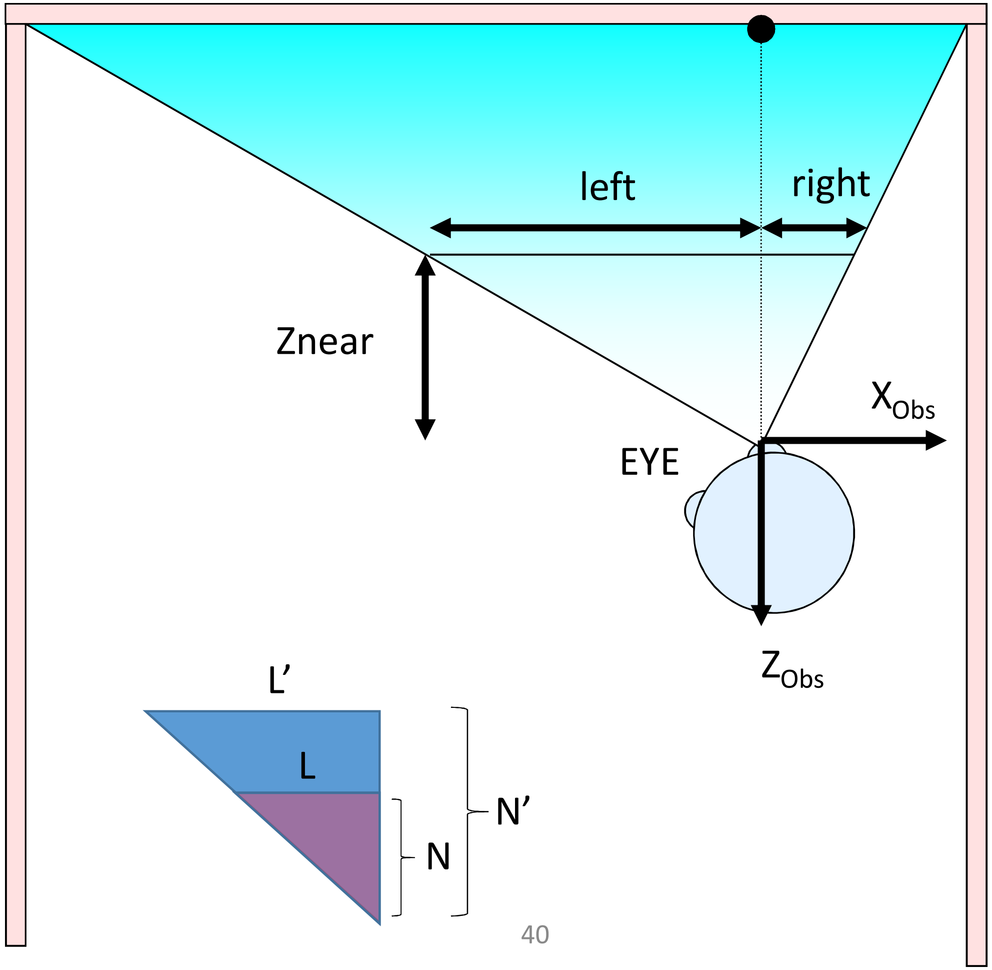
Computing the Projection matrix

Projection matrix

- First, compute **left, right, top, bottom**
- The, use makePerspective

Hints:

- u, v vectors are aligned with eye-space X, Y axes.
- Recall the projection interpretation of the dot product.
- Use properties of similar triangles.



Moving to WebXR

Check the VR examples at ThreeJs.org

```
renderer = new THREE.WebGLRenderer( { antialias: true } );
renderer.setPixelRatio( window.devicePixelRatio );
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.setAnimationLoop( animate );
renderer.shadowMap.enabled = true;
renderer.xr.enabled = true;
renderer.xr.cameraAutoUpdate = false;
```

```
function animate() {

        const delta = clock.getDelta();
        const elapsedTime = clock.elapsedTime;
        renderer.xr.updateCamera( camera );
        world.execute( delta, elapsedTime );
        renderer.render( scene, camera );

}
```

```
// controllers
const controller1 = renderer.xr.getController( 0 );
scene.add( controller1 );

const controller2 = renderer.xr.getController( 1 );
scene.add( controller2 );

const controllerModelFactory = new XRControllerModelFactory();

// Hand 1
const controllerGrip1 = renderer.xr.getControllerGrip( 0 );
controllerGrip1.add( controllerModelFactory.createControllerModel( controllerGrip1 ) );
scene.add( controllerGrip1 );

const hand1 = renderer.xr.getHand( 0 );
hand1.add( new OculusHandModel( hand1 ) );
const handPointer1 = new OculusHandPointerModel( hand1, controller1 );
hand1.add( handPointer1 );
```