

# EE497 – 3D INTERFACE TECHNOLOGIES

## ASSIGNMENT 1

CIARAN O'DONNELL

STUDENT NO: 15414048

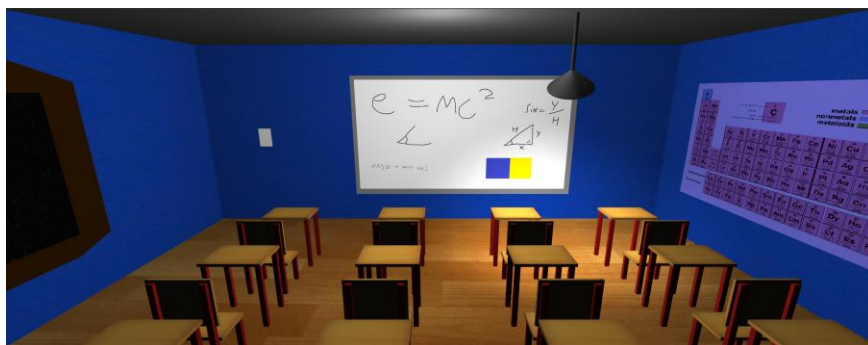
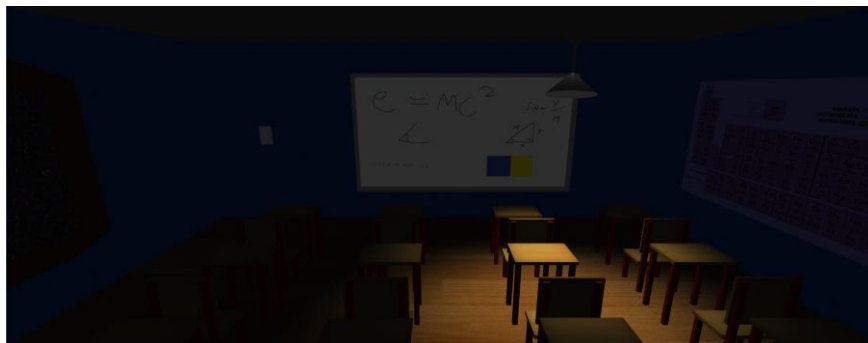
DATE: 14<sup>TH</sup>/MARCH/2019

### Introduction:

This goal of this assignment is to develop software using OSGJS to create a 3D scene that displays some of the fundamental aspects of 3D interfaces. For this assignment I have opted to create a classroom scene that incorporates different elements to demonstrate these aspects of 3D interfaces.

The assignment should show the use of scene graph design, design of custom geometry, use of lighting, texture mapping, animation and interaction. I have divided this report to show how each of these elements are displayed in my scene.

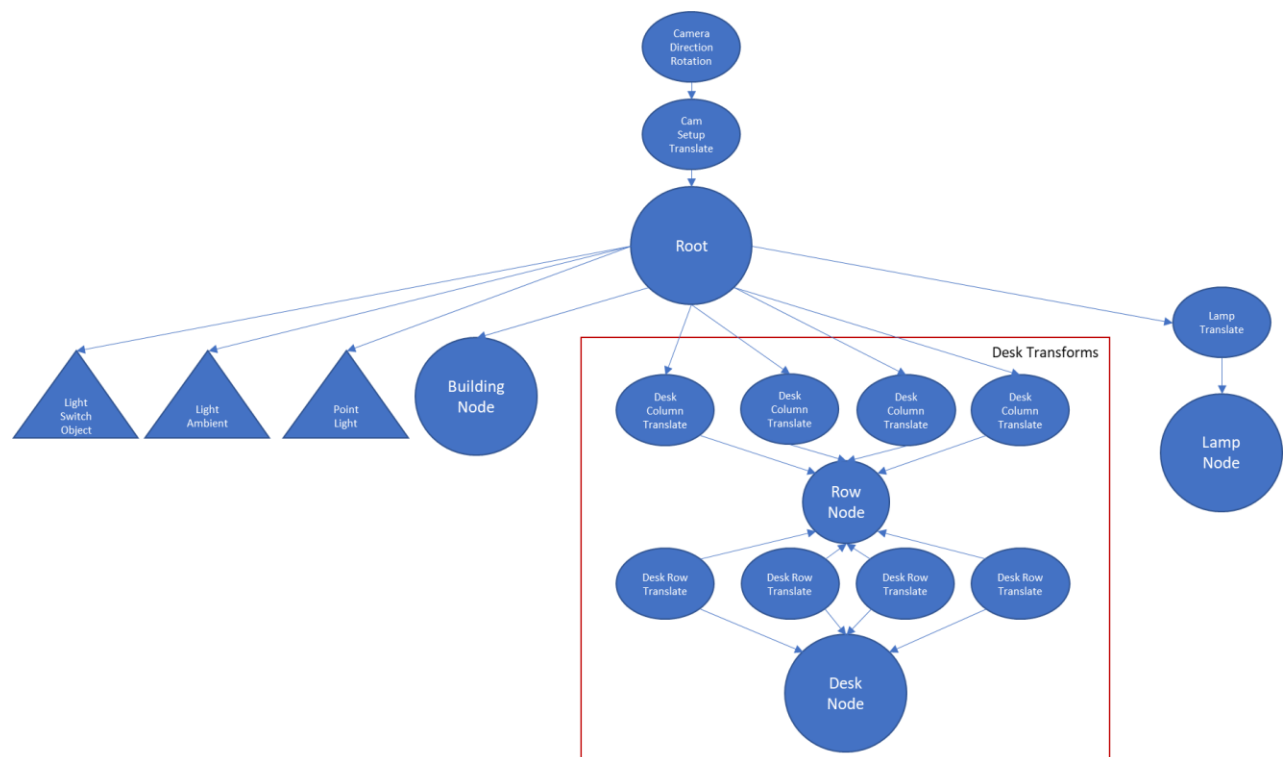
The final scene includes a classroom with a grid of desks and chairs, along with a swinging lamp, and an interactable light switch to turn on further lighting in the scene.



## Scene Graph Design

In order to create this scene many assets had to be added to the scene and placed (translated and rotated) appropriately. The best way to do this is via scene graph design.

The scene graph has been split into 4 separate scene graphs that nest within each other as the design is overly complex to show as a single scene graph.



This is the root scene graph and is highest in the hierarchy. It is where the entire scene extends from. The two camera transforms at the top are simply a method to move the scene to a more pleasing angle for the user, and to select a viewing angle for the user. It will be explained in more detail how this was achieved later in the report. This scene graph also shows how a grip of desks was attained by translating the desks into rows and then into columns to attain a 4\*4 grid of desks. This was achieved via use of a for loop too create four transforms and add a desk to each and then do the same again for adding each row to a column.

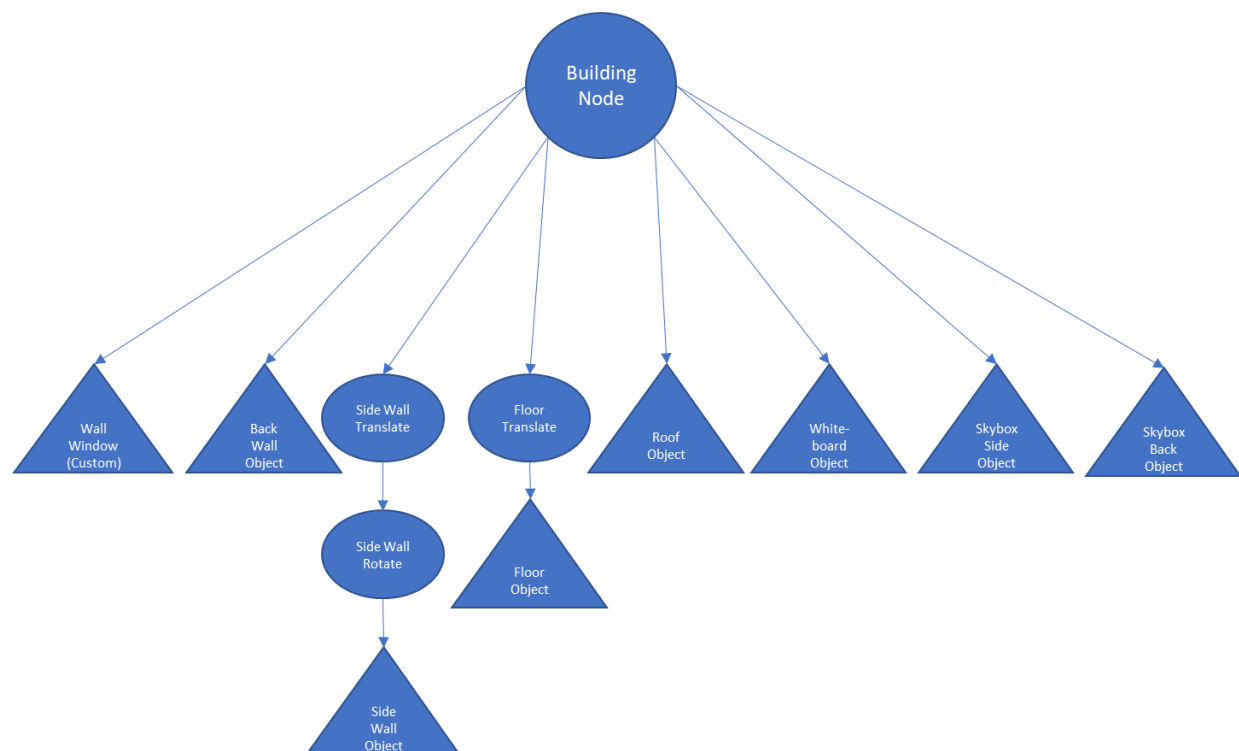
```

for(i=0; i<3; i++){ //creates a row of desks four times
    var MTRX = new osg.Matrix.create();
    MTRX = osg.Matrix.makeTranslate(0, -(i*15)-20, -8, MTRX); //i dictates spacing
    var Trn = new osg.MatrixTransform();
    Trn.setMatrix(MTRX);
    Trn.addChild(createDesk());
    Deskrow.addChild(Trn); //adds desk to DeskRow node
}
for(j=0; j<4; j++){ //adds each row of four desks to column
    var MTRXcol = new osg.Matrix.create();
    MTRXcol = osg.Matrix.makeTranslate(j*15-20, 0, 0, MTRXcol); //j dictates spacing
    var Trncol = new osg.MatrixTransform();
    Trncol.setMatrix(MTRXcol);
    Trncol.addChild(Deskrow); //adds each row
    root.addChild(Trncol); //adds to root
}

```

This allows a grid of desks be added quickly without specifying the placement of them all separately.

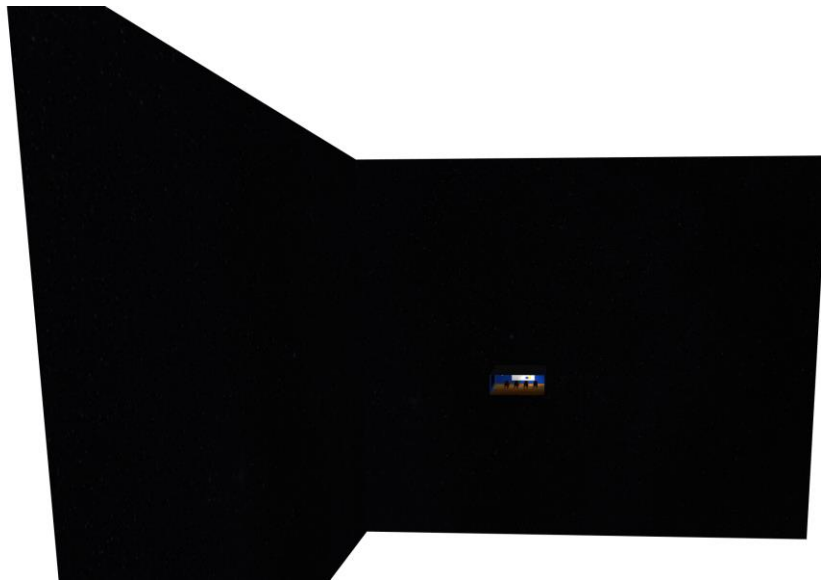
The next scene graph is of the Building node. Within this scene graph is the design of the general building/environment the other assets are placed in.



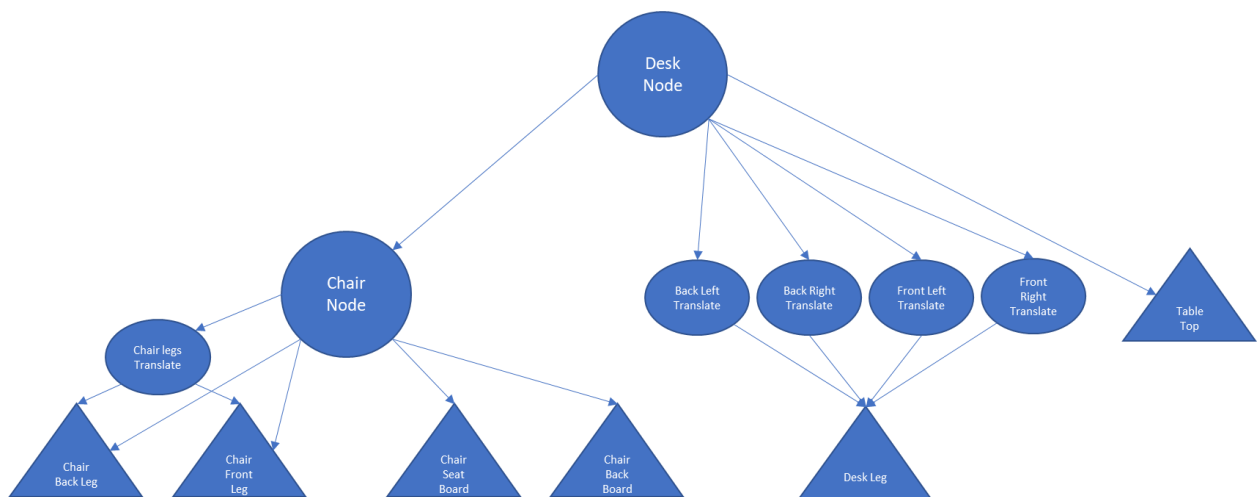
This scene graph is a little simpler than the last. It could be simplified further using smarter design for instance the side wall and floor could have been translated and rotated on their creation, however rotating and translating these objects was a good learning experience on how these processes are carried out. All these objects were created using pre-determined functions to create objects in osgjs except the windowed wall which is custom geometry. This will be explained further later in the report. This scene graph also includes the skyboxes

which are two cube objects used to give the impression of a night's sky looking out the window. It is important that the material of these skyboxes only have an ambient element as they should not interact with lighting.

```
var skyboxmat = new osg.Material();  
skyboxmat.setAmbient([0.5, 0.5, 0.5, 1]);  
skyboxmat.setDiffuse([0, 0, 0, 1]);
```



The next scene graph is the desk scene graph is of the Desk Node. This describes the design of the desk and chair assets used in the final scene.



Again, this scene graph is rather simple, however we can see that the chair group node is nested within the desk group node. This was not necessary, but it simplified the process of adding the chairs to the grid of desks described above as they would be added as part of the desk automatically. Also, there can also be seen here two different approaches taken in the design of the desk and chair. The desk has a leg and table top shape node created and the leg is translated four times, for each position a leg must stand. However, the chair has the legs position specified on creation and then they are translated once across to the other side to create the four legs.

```
var chairMTRX = new osg.Matrix.create(); //translate leftside legs to rightside
chairMTRX = osg.Matrix.makeTranslate(4, 0, 0, chairMTRX);
var chairTrn = new osg.MatrixTransform();
chairTrn.setMatrix(chairMTRX);
chairnode.addChild(chairTrn);
chairTrn.addChild(chairlegbck);
chairTrn.addChild(chairlegfrnt);

var frontleftMTRX = new osg.Matrix.create(); //translation for each table leg
frontleftMTRX = osg.Matrix.makeTranslate(-2.6, -2, -3.5, frontleftMTRX);
var frontleftTrn = new osg.MatrixTransform();
frontleftTrn.setMatrix(frontleftMTRX);
desknode.addChild(frontleftTrn);
frontleftTrn.addChild(leg);

var frontrightMTRX = new osg.Matrix.create();
frontrightMTRX = osg.Matrix.makeTranslate(2.6, -2, -3.5, frontrightMTRX);
var frontrightTrn = new osg.MatrixTransform();
frontrightTrn.setMatrix(frontrightMTRX);
desknode.addChild(frontrightTrn);
frontrightTrn.addChild(leg);

var backleftMTRX = new osg.Matrix.create();
backleftMTRX = osg.Matrix.makeTranslate(-2.6, 2, -3.5, backleftMTRX);
var backleftTrn = new osg.MatrixTransform();
backleftTrn.setMatrix(backleftMTRX);
desknode.addChild(backleftTrn);
backleftTrn.addChild(leg);

var backrightMTRX = new osg.Matrix.create();
backrightMTRX = osg.Matrix.makeTranslate(2.6, 2, -3.5, backrightMTRX);
var backrightTrn = new osg.MatrixTransform();
backrightTrn.setMatrix(backrightMTRX);
desknode.addChild(backrightTrn);
backrightTrn.addChild(leg);
```

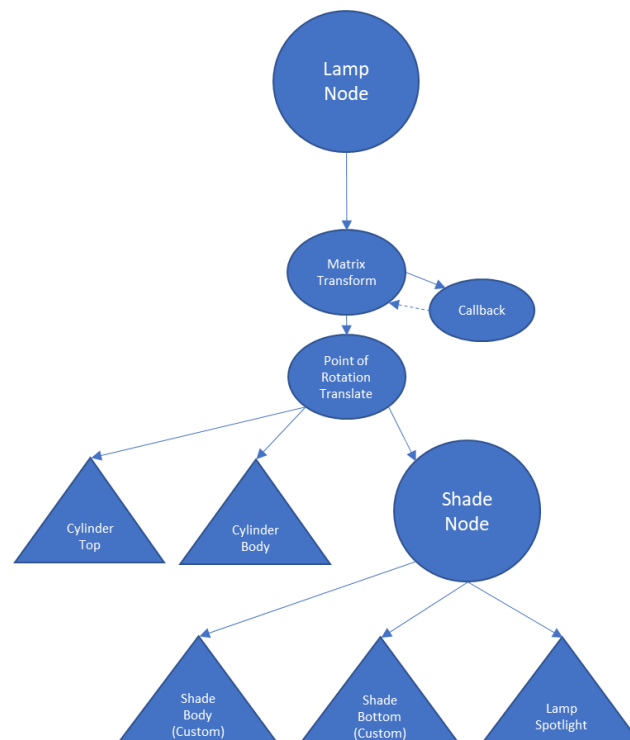
Here we can see the 4 translations for the table and the one translation for the chair.

```
var chairlegfrnt = osg.createTexturedBoxGeometry(-2, -6, -5, 0.5, 0.5, 3.5);
chairlegfrnt.getOrCreateStateSet().setAttributeAndModes(legmaterial); //chair leg created and
placed at front left

var chairlegbck = osg.createTexturedBoxGeometry(-2, -10, -3, 0.5, 0.5, 9.5);
chairlegbck.getOrCreateStateSet().setAttributeAndModes(legmaterial); //chair leg created and
placed at front right
```

This is only possible as when the chair legs are created they are also positioned on the left side.

The final scene graph is the one for the swinging lamp. This scene graph shows the use of a call back function. This function is used to dynamically update the rotation of this scene to give the lamp a swinging effect. The call back function updates the matrix controlling the rotation. We also see the use of more custom geometry in this scene which will be explained in further detail later in the report.



## Custom Geometry

Two elements of this scene were created using custom geometry. The first of these is the windowed wall of the room. This was created using indexed geometry and the triangle primitive to create the object. The indexed points were worked out by hand before being hard coded into the object.

```
var l = 80;
var w = 120;
var h = 30;

{
    //windowed wall
    var vertexAttribArray = new osg.BufferArray(osg.BufferArray.ARRAY_BUFFER, null
, 3);
    vertexAttribArray.setElements(new Float32Array(//creates the points of the
object
[   -l/2, -0, -h/2,
      -l/2, 0, h/2,
      -l/2, 0, 0,
      -l/2, -w/4, h/4,
      -l/2, -w/4, -h/4,
      -l/2, -(w/4)*2, h/2,
      -l/2, -(w/4)*2, -h/2,
      -l/2, -(w/4)*3, h/4,
      -l/2, -(w/4)*3, -h/4,
      -l/2, -w, h/2,
      -l/2, -w, 0,
      -l/2, -w, -h/2]));

    var indices = new osg.BufferArray(osg.BufferArray.ELEMENT_ARRAY_BUFFER, null, 1
);
    indices.setElements(new Uint16Array(
        [ 0, 2, 4,//indices for each triangle
          4, 2, 3,
          3, 2, 1,
          1, 5, 3,
          5, 7, 3,
          5, 9, 7,
          10, 7, 9,
          10, 8, 7,
          11, 8, 10,
          11, 6, 8,
          8, 6, 4,
          4, 6, 0]));

    var normalArray = new osg.BufferArray(osg.BufferArray.ARRAY_BUFFER, null, 3);
    normalArray.setElements(new Float32Array(
        [1, 0, 0,
          1, 0, 0,
          1, 0, 0,
          1, 0, 0,
          1, 0, 0,
          1, 0, 0,
          1, 0, 0,
          1, 0, 0,
          1, 0, 0,
          1, 0, 0,
          1, 0, 0,
          1, 0, 0,
          1, 0, 0]
        ));

    var windowwall = new osg.Geometry();
    windowwall.setVertexAttribArray('Vertex', vertexAttribArray);
```

```
windowwall.setVertexAttribArray('Normal', normalArray);
windowwall.getPrimitives().push(new osg.DrawElements(osg.PrimitiveSet.TRIANGLES, indices));
```

With the wall being a flat surface, the normal data was added as being perpendicular to the ZY plane at all points.



The second piece of custom geometry created for this project was the lamp shade. This was created using the triangle strip primitive. It was created via a for loop that added a point at the high inner part of the shade and then the low outer part of the shade for each loop. Then a step was taken around the circumference of the shade and the next loop ran.

```
function createShade() {
    var angle = 0.0;
    var angleIncrement = (2 * Math.PI) / faces;
    var shadecoordinates = new Array(faces*3+2);
    //array has 1points(3coords) for each face +2 for first face.

    shadecoordinates[0] = (shaderadius); //create first point at bottom outer shade
    shadecoordinates[1] = (0);
    shadecoordinates[2] = (-2);
    for(var f=0; f<(faces/2)+1; f++) {
        // Generate a coordinate for each face
        //coord at bottom and coord at top of shade per loop
        shadecoordinates[f*6+3] = (radius*Math.cos(angle-angleIncrement));
        shadecoordinates[f*6+4] = (radius*Math.sin(angle-angleIncrement));
        shadecoordinates[f*6+5] = (0); //creates each high point at top inner shade
        angle = angle-angleIncrement; //steps around shade by increment

        shadecoordinates[f*6+6] = (shaderadius*Math.cos((angle-angleIncrement)));
        shadecoordinates[f*6+7] = (shaderadius*Math.sin((angle-angleIncrement)));
        shadecoordinates[f*6+8] = (-2); //creates each subsequent low outer point
        angle = angle-angleIncrement; //steps around shade by increment
    }
    var shadenormals = new Array(faces*3+2);
    shadenormals[0] = Math.cos(angle); //does the same for normals
    shadenormals[1] = Math.sin(angle);
    shadenormals[2] = 0.65;

    for(var f=0; f<(faces/2)+1; f++)
    {
        // Generate normal for each coord
        var n1x = Math.cos(angle);
        var n1y = Math.sin(angle);
        angle -= angleIncrement;
        var n2x = Math.cos(angle);
        var n2y = Math.sin(angle);
        angle -= angleIncrement;
        // Populate the coordinates array
        shadenormals[f*6+3] = n1x;
        shadenormals[f*6+4] = n1y;
        shadenormals[f*6+5] = 0.65;
        shadenormals[f*6+6] = n2x;
```



```

        shadenormals[f*6+7] = n2y;
        shadenormals[f*6+8] = 0.65;
    }
    var normalAttribArray = new osg.BufferArray(osg.BufferArray.ARRAY_BUFFER, null, 3);
    normalAttribArray.setElements(new Float32Array(shadenormals)); //sets normals array to
    attrib
    var vertexAttribArray = new osg.BufferArray(osg.BufferArray.ARRAY_BUFFER, null, 3);
    vertexAttribArray.setElements(new Float32Array(shadecoordinates)); //sets vert array to
    attrib
    var geometry = new osg.Geometry();
    geometry.setVertexAttribArray('Vertex', vertexAttribArray); //sets vertexes
    geometry.setVertexAttribArray('Normal', normalAttribArray); //sets normals
    geometry.getPrimitives().push(new osg.DrawArrays(osg.PrimitiveSet.TRIANGLE_STRIP, 0,
    faces+2));
    return geometry; //draws them as trianglestrip and returns object.
}

```



There was also a bottom circle added to this which was a simple circle to hide the invisible sides of the lamp shape.

```

var angle = 0.0;
var angleIncrement = (2*Math.PI)/faces;
var coordinates = new Array(faces*3*3);

for(var f=0; f<faces; f++) {
    // Generate the four coordinates required for each face
    var x1 = shaderadius * Math.cos(angle);
    var y1 = shaderadius * Math.sin(angle);
    angle += angleIncrement;
    var x2 = shaderadius * Math.cos(angle);
    var y2 = shaderadius * Math.sin(angle);
    coordinates[f*3*3] = 0;
    coordinates[f*3*3+1] = 0;
    coordinates[f*3*3+2] = -height/2.0-2;
    coordinates[f*3*3+3] = x2;
}

```

```

        coordinates[f*3*3+4] = y2;
        coordinates[f*3*3+5] = -height/2.0-2;
        coordinates[f*3*3+6] = x1;
        coordinates[f*3*3+7] = y1;
        coordinates[f*3*3+8] = -height/2.0-2;
    }
    var normals = new Array(faces*3*3);
    for(var f=0; f<faces; f++) {
        // Generate the four coordinates required for each face
        var x1 = Math.cos(angle);
        var y1 = Math.sin(angle);
        angle += angleIncrement;
        var x2 = Math.cos(angle);
        var y2 = Math.sin(angle);

        normals[f*3*3] = 0;
        normals[f*3*3+1] = 0;
        normals[f*3*3+2] = -1;
        normals[f*3*3+3] = 0;
        normals[f*3*3+4] = 0;
        normals[f*3*3+5] = -1;
        normals[f*3*3+6] = 0;
        normals[f*3*3+7] = 0;
        normals[f*3*3+8] = -1;
    }
    var vertexAttribArray = new osg.BufferArray(osg.BufferArray.ARRAY_BUFFER, null, 3);
    vertexAttribArray.setElements(new Float32Array(coordinates));
    var normalAttribArray = new osg.BufferArray(osg.BufferArray.ARRAY_BUFFER, null, 3);
    normalAttribArray.setElements(new Float32Array(normals));
    var geometry = new osg.Geometry();
    geometry.setVertexAttribArray('Vertex', vertexAttribArray);
    geometry.setVertexAttribArray('Normal', normalAttribArray);
    geometry.getPrimitives().push(new osg.DrawArrays(osg.PrimitiveSet.TRIANGLES, 0, 3*faces));
    return geometry;
}

```

This bottom piece of geometry is near identical to that in the supplied code for this module.

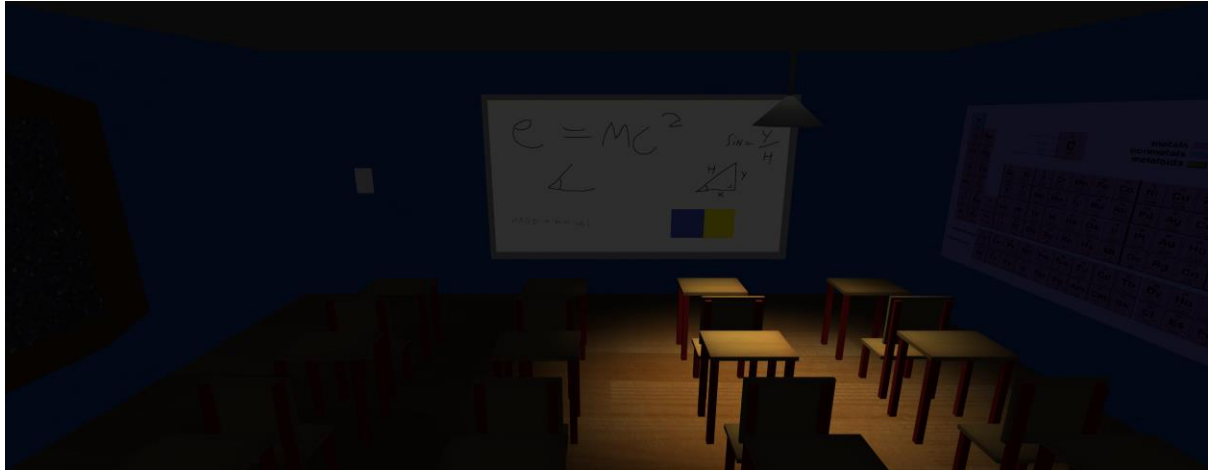
## Lighting

This scene uses a total of three lights. The first and simplest is an ambient light. This is used as a very low light level so that the scene and skyboxes are still visible even when the other lights are not interacting with them. This light is added directly to the root scene.

```
var nextLightNumber = 0;
function getNextLightNumber() {
    return nextLightNumber++;
}

function createAmbientLight(r, g, b) {
    var lightNumber = getNextLightNumber();
    var ambientLight = new osg.Light(lightNumber);
    ambientLight.setPosition([0, 0, 0, 1]);
    if(typeof r === 'undefined') r = 0.8;
    if(typeof g === 'undefined') g = 0.8;
    if(typeof b === 'undefined') b = 0.8;
    ambientLight.setDiffuse([0.0, 0.0, 0.0, 1.0]);
    ambientLight.setSpecular([0.0, 0.0, 0.0, 1.0]);
    ambientLight.setAmbient([r, g, b, 1.0]);
    var lightSource = new osg.LightSource();
    lightSource.setLight(ambientLight);
    return lightSource;
}
```

```
root.addChild(createAmbientLight(0.2, 0.2, 0.2)); //create dark ambient light
```



This light's effects can be seen here where we can see items in the scene that are not directly lit by the lamp.

The next light in the scene is that of the lamp. This light is a spotlight and is created within the lamp shade node so that the light will rotate with the lamp. The spotlight type has r g b elements as well as a direction (x y z), attenuation (c l q), a cut off and a blend. It may also have a position, but it is best to simply translate it after it is created.

```

function createSpotLight(x, y, z, r, g, b, c, l, q, cutoff, blend) {
    var lightNumber = getNextLightNumber();
    var spotLight = new osg.Light(lightNumber);
    spotLight.setPosition([0, 0, 0, 1]);

    if(typeof x === 'undefined') x = 0.0;
    if(typeof y === 'undefined') y = 1.0;
    if(typeof z === 'undefined') z = 0.0;
    spotLight.setDirection([x, y, z]);

    if(typeof r === 'undefined') r = 0.8;
    if(typeof g === 'undefined') g = 0.8;
    if(typeof b === 'undefined') b = 0.8;
    spotLight.setDiffuse([r, g, b, 1.0]);
    spotLight.setSpecular([r, g, b, 1.0]);
    spotLight.setAmbient([0.0, 0.0, 0.0, 1.0]);

    if(typeof c === 'undefined') c = 1.0;
    if(typeof l === 'undefined') l = 0.0;
    if(typeof q === 'undefined') q = 0.0;
    spotLight.setConstantAttenuation(c);
    spotLight.setLinearAttenuation(l);
    spotLight.setQuadraticAttenuation(q);

    if(typeof cutoff === 'undefined') cutoff = 25.0;
    spotLight.setSpotCutoff(cutoff);

    if(typeof blend === 'undefined') blend = 1.0;
    spotLight.setSpotBlend(blend);

    var lightSource = new osg.LightSource();
    lightSource.setLight(spotLight);
    return lightSource;
}

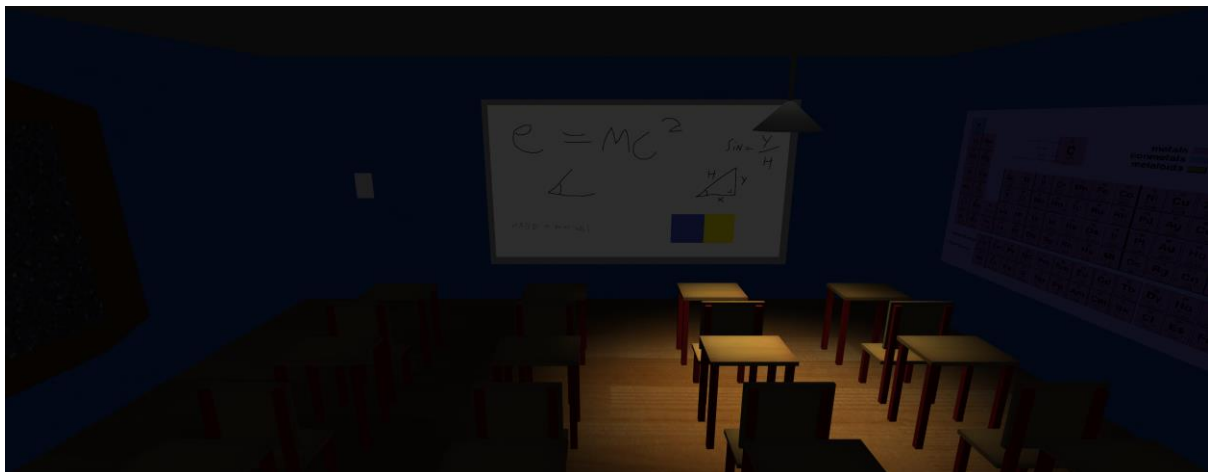
```

```

var light = createSpotLight(0, 0, -20, .8, .8, .8, 1, 0, 0, 45, 1);
shadenode.addChild(light);

```

Having a light source that moves through the scene makes it very easy to see how the light interacts with objects within the scene, and this can again be seen in this scene:



The final light source is a point light. A point light was used as a directional light could not cast light on all three walls, floor and ceiling at once, and another ambient light would leave no areas dark (such as the underside of desks) so a point light was used. The point light was added to the root node, it is controllable via the light switch on the wall of the class room.

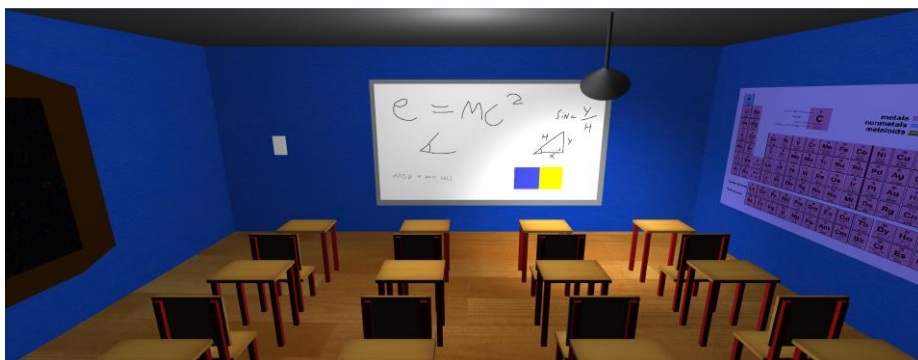
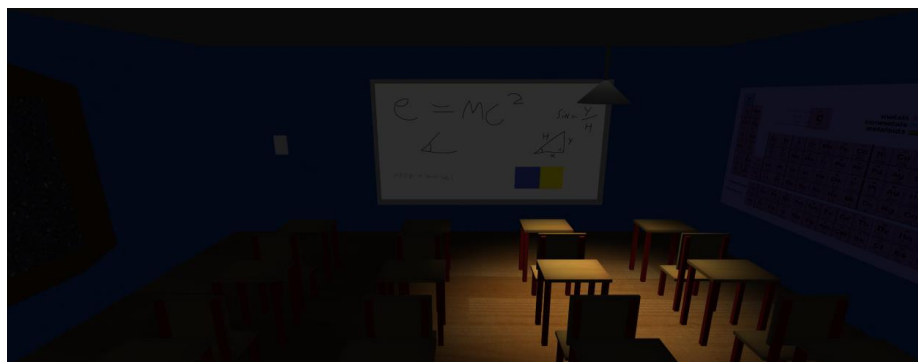
```
var lights = new osg.LightSource();//create light source
var LightOn = new osg.Light(getNextLightNumber());//create light object
LightOn.setPosition([0, -30, 10, 1]);//set pos
LightOn.setDiffuse([0, 0, 0, 1.0]);//no light to start
LightOn.setSpecular([0, 0, 0, 1.0]);//no specular ever (dont want ball shaped shine)
LightOn.setAmbient([0.0, 0.0, 0.0, 1.0]);
LightOn.setConstantAttenuation(1);
LightOn.setLinearAttenuation(0);
LightOn.setQuadraticAttenuation(0)
```

```
lights.setLight(LightOn);//set light source to light object
root.addChild(lights);
```

The light is changed dynamically via updating the light types diffuse variable on each click to either set it to zero or 0.8 accordingly.

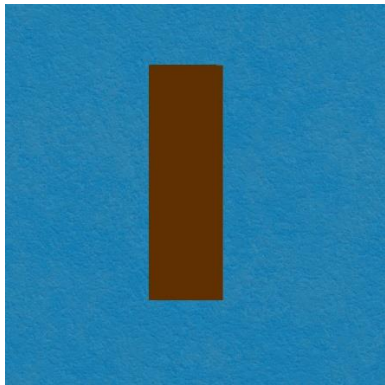
```
lswitch.onpick = function (){//on click of lightswitch
    if(0==lightlevel){//if light off
        lightlevel = 1;//set on
        LightOn.setDiffuse([0.8, 0.8, 0.8, 1.0]);}
    else{
        lightlevel = 0;//set off
        LightOn.setDiffuse([0.0, 0.0, 0.0, 1.0]);}
};
```

The difference between light levels can be seen here:



# Texture Mapping

For the majority of the textures in the scene they were simply wrapped in a texture as mapping was not necessary. The only object directly texture mapped is the windowed wall. This is a simple rectangular wall so in order to make the process of wrapping this object a texture that did not fit the shape very well was used.

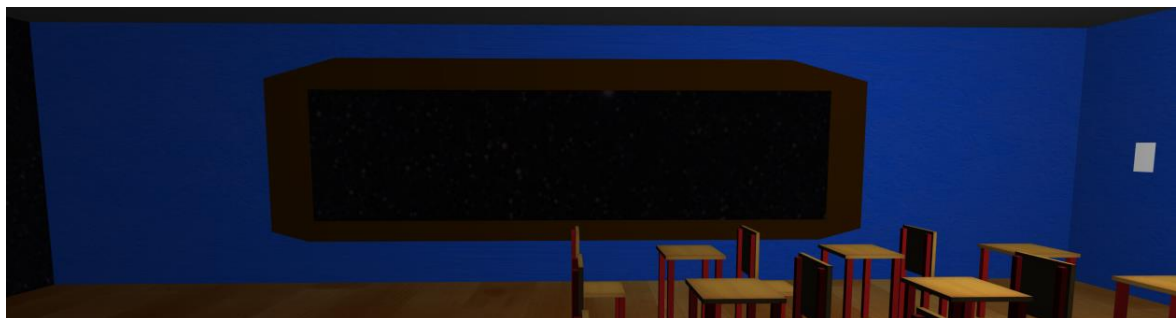


By setting the points on the wall around the window to the center of this texture it was possible to stretch the texture out in order to get the desired window frame effect.

```
var textureArray = new osg.BufferArray(osg.BufferArray.ARRAY_BUFFER, null, 2);
textureArray.setElements(new Float32Array(
    [
        1, 0, //outer points set as normal
        1, 1,
        1, 0.5,
        0.50, 0.70, //points around window centred to meet brown in texture.
        0.50, 0.33,
        0.48, 1,
        0.48, 0,
        0.45, 0.70,
        0.45, 0.33,
        0, 1, //outer points set as normal
        0, 0.5,
        0, 0
    ]
));
```

```
windowwall.setVertexAttribArray('TexCoord0', textureArray);
windowwall.getOrCreateStateSet().setTextureAttributeAndMode(0,
    osg.Texture.createFromURL("ClassRoom/windowwall.jpgg"));
```

This gave the desired effect:



## Animation

The animation in this project is of the lamp rotating over and back. This was achieved via a call back function in the lamps node that updated a transform matrix controlling the rotating of the lamp. The lamp was also translated so the tip of the lamp pole was the point of rotation giving a swinging effect.

```
var matrixTransform = new osg.MatrixTransform();
matrixTransform.addChild(polenode); //rotation matrix
var updateCallback = new SimpleUpdateCallback();
matrixTransform.addUpdateCallback(updateCallback);

var root = new osg.Node();
root.addChild(matrixTransform); //add rotation matrix to root
return root;
```

```
var SimpleUpdateCallback = function() {};
var dir = 1;

SimpleUpdateCallback.prototype = {
  // rotation angle
  angle: 0,

  update: function(node, nodeVisitor) {
    var currentTime = nodeVisitor.getFrameStamp().getSimulationTime();
    var dt = currentTime - node._lastUpdate;
    if (dt < 0) return true;
    node._lastUpdate = currentTime;

    // rotation
    var matrix = node.getMatrix();
    osg.Matrix.makeRotate(this.angle, 1.0, -0.3, 0.0, matrix); //rotate by angle both x and y
    if(this.angle < 0.8 && dir == 1) { //up to angle 0.8
      this.angle += 0.01;
    }
    else { //
      this.angle -= 0.01;
      dir = 0; //change direction
    }
    if(this.angle < -0.8) dir = 1; //back to -0.8 than chang
    return true;
  }
};
```





## Interaction

The user can interact with this scene in two ways. The first is by turning in and off a light using the light switch within the scene. This uses a simple on click function that once clicked updates the point lights diffuse attribute to either on or off and is designed as such.

```
lswitch.onpick = function (){//on click of lightswitch
    if(0==lightlevel){//if light off
        lightlevel = 1;//set on
        LightOn.setDiffuse([0.8, 0.8, 0.8, 1.0]);} //set light level
    else{
        lightlevel = 0;//set off
        LightOn.setDiffuse([0.0, 0.0, 0.0, 1.0]);} //set level 0
};
```

The second element of interactivity in this scene is the camera control via the keyboard. This is set up to give the user three separate viewing angles. The user can rotate the scene left or right using the arrow keys. It is done by having an event listener for a key press.

```
var KEY_CODE_LEFT_ARROW = 37;
var KEY_CODE_RIGHT_ARROW = 39;

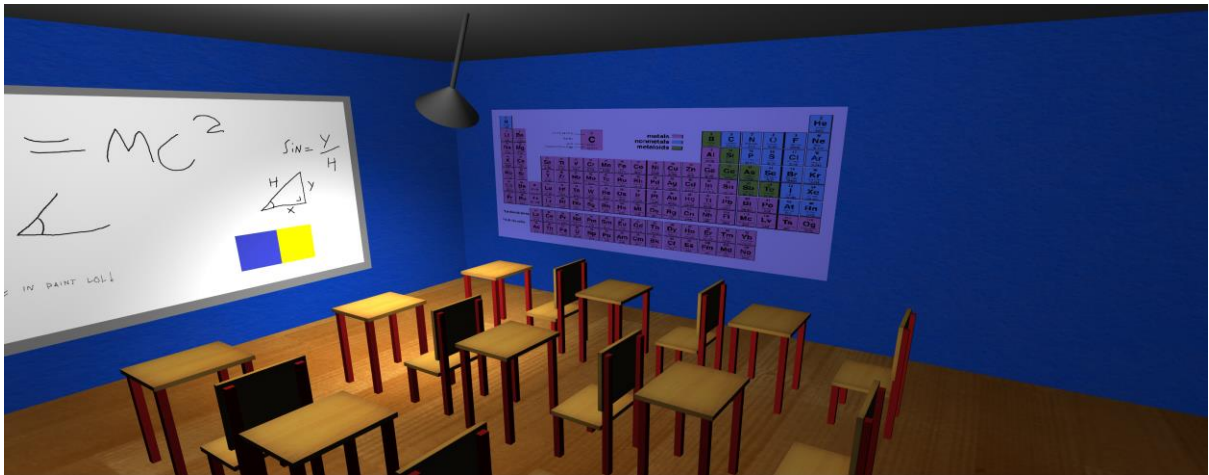
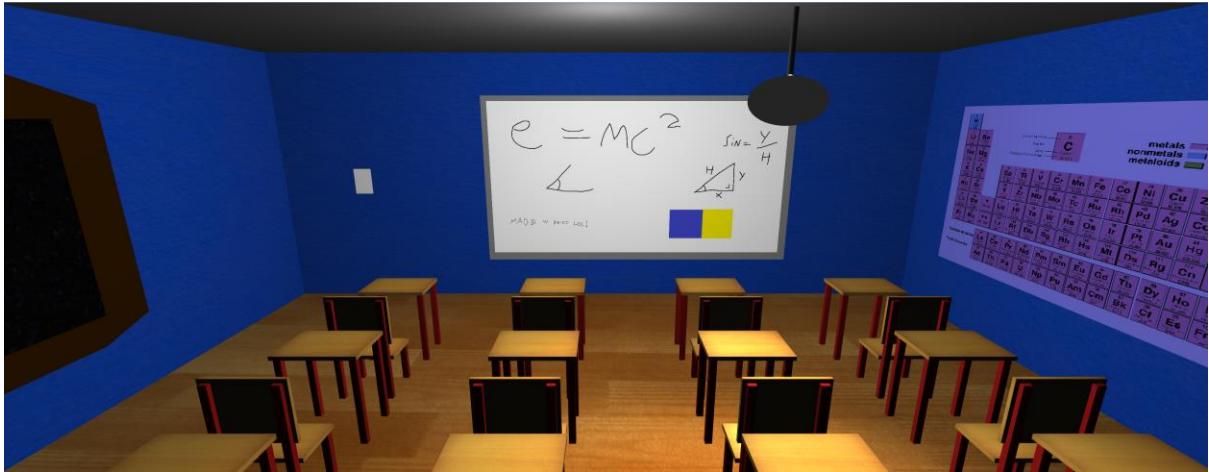
var leftRightAngle = 0;
var leftRightMatrix;
var leftRightTransform;
```


```
function init() {
    disableDefaultLight();
    enablePicking();
    var canvas = document.getElementById("canvas");
    canvas.addEventListener("keydown", keyPressed);
    canvas.focus();
}
```

Once the key is pressed it updates the matrix transform to rotate the scene either left or right.


```
var currentpos = 1;
function keyPressed(event) {
    var keyCode = event.keyCode;
    if(keyCode == KEY_CODE_LEFT_ARROW && currentpos>0) {//if not looking left
        //alert("left");
        leftRightAngle -= 1;//rotate left
        currentpos--;//update view
    }
    else if(keyCode == KEY_CODE_RIGHT_ARROW && currentpos<2) {//if not looking right
        //alert("right");
        leftRightAngle += 1;//rotate right
        currentpos++;//update view
    }
    osg.Matrix.makeRotate(leftRightAngle, 0.0, 0.0, 1.0, leftRightMatrix);//update matrix
    leftRightTransform.setMatrix(leftRightMatrix);//set matrix
}
```

This creates the following three angles to view the scene:





[1]



[3]

## REFERENCES

- [1] seamless-pixels.blogspot, "seamless-pixels.blogspot," [Online]. Available: <http://seamless-pixels.blogspot.com/2012/07/blue-wall-paint-stucco-plaster-texture.html>. [Accessed 14 March 2019].
- [2] G. Hodan, "publicdomainpictures.net," [Online]. Available: <https://www.publicdomainpictures.net/en/view-image.php?image=67167&picture=stars-in-the-night-sky>. [Accessed 14 March 2019].
- [3] Dmarcus100, "Wikipedia," 11 November 2016. [Online]. Available: [https://commons.wikimedia.org/wiki/File:Periodic\\_Table\\_Of\\_Elements\\_2016.jpg](https://commons.wikimedia.org/wiki/File:Periodic_Table_Of_Elements_2016.jpg). [Accessed 14 March 2019].