**Dublin City University School of Electronic Engineering**

EE497 – 3D Interface Technologies Assignment 1

building

sphere

base

column

Plane parts

Lighting used in the program is directional light, ambient light and point light.

|  |
| --- |
| **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; }  */\* function to add Directional Light  x, y, z: location of the group (beacon and the spot light) in the scene  r, g, b: color of the light  \*/* **function** *createDirectionalLight*(x, y, z, r, g, b) {   **var** lightNumber = *getNextLightNumber*();  **var** directionalLight = **new osg**.**Light**(lightNumber);   **if**(**typeof** x === **'undefined'**) x = 1.0;  **if**(**typeof** y === **'undefined'**) y = -1.0;  **if**(**typeof** z === **'undefined'**) z = 1.0;  directionalLight.setPosition([x, y, z, 0.0]);   **if**(**typeof** r === **'undefined'**) r = 0.8;  **if**(**typeof** g === **'undefined'**) g = 0.8;  **if**(**typeof** b === **'undefined'**) b = 0.8;  directionalLight.setDiffuse([r, g, b, 1.0]);  directionalLight.setSpecular([r, g, b, 1.0]);  directionalLight.setAmbient([0.0, 0.0, 0.0, 1.0]);   **var** lightSource = **new osg**.*LightSource*();  lightSource.setLight(directionalLight);  **return** lightSource; }  */\* function to add point Light  r, g, b: color of the light  c, l, g: attenuation  \*/* **function** *createPointLight*(r, g, b, c, l, q) {   **var** lightNumber = *getNextLightNumber*();  **var** pointLight = **new osg**.**Light**(lightNumber);  pointLight.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;  pointLight.setDiffuse([r, g, b, 1.0]);  pointLight.setSpecular([r, g, b, 1.0]);  pointLight.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;  pointLight.setConstantAttenuation(c);  pointLight.setLinearAttenuation(l);  pointLight.setQuadraticAttenuation(q);   **var** lightSource = **new osg**.*LightSource*();  lightSource.setLight(pointLight);   **return** lightSource; } |

Point lighting is used in the case of the moon that emits light as it rotates around the scene.

Lights are added to the aircraft enabling the aircraft to light up the areas it vests.

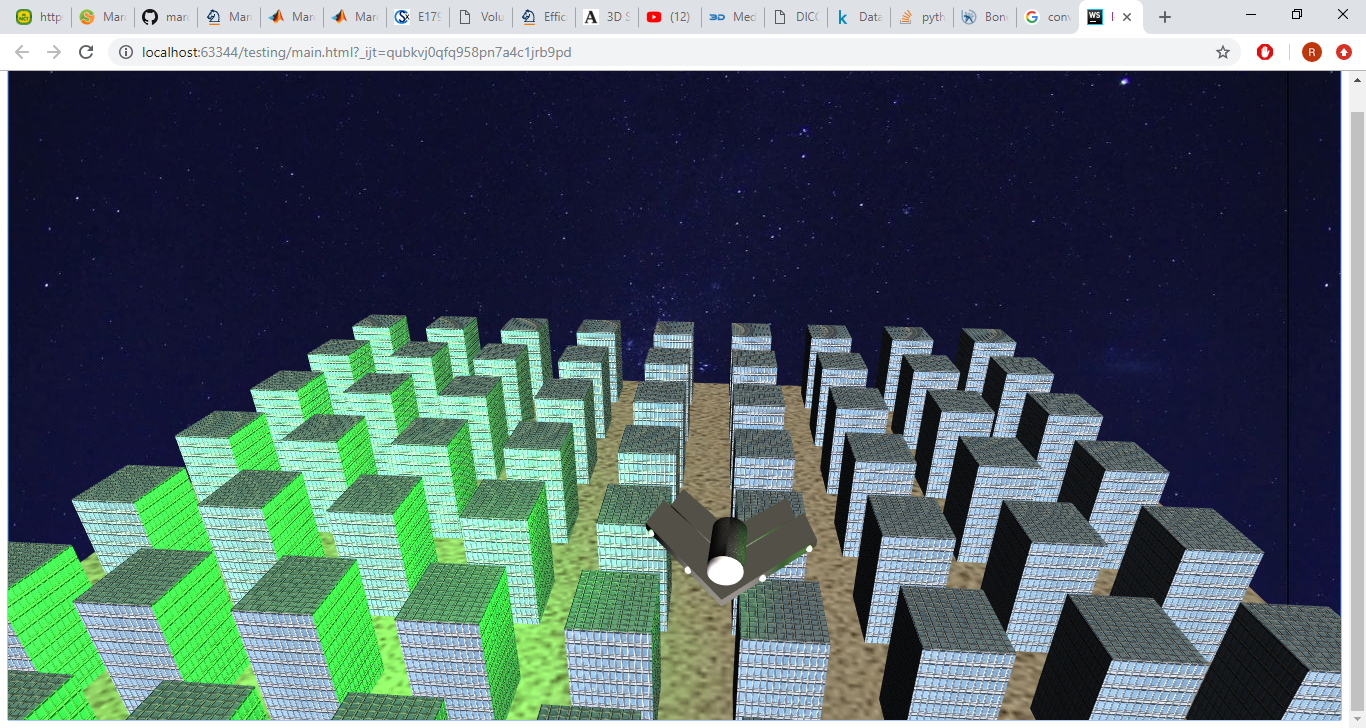
Texture mapping was applied to the building and the landing pad for the aircraft. Code of textured building which are used to display a terrain in which the plane are used sample code of textured building are

|  |
| --- |
| **var** buildingHeight = 100; **var** buildingWidth = 50; **var** buildingDepth = 50; **for** (**var** x = -350; x < 400; x += 100){  **for** (**var** y = -400; y < 500; y += 100) {  **var** translateMatrix = **new osg**.**Matrix**.create();  translateMatrix = **osg**.**Matrix**.makeTranslate(6\*(width+2), 0, -20, translateMatrix);  **var** translateMatrixTransform = **new osg**.*MatrixTransform*();  translateMatrixTransform.setMatrix(translateMatrix);  root.addChild(translateMatrixTransform);  **var** building = **osg**.*createTexturedBoxGeometry*(x, y, buildingHeight/2 - 5, buildingWidth, buildingDepth, buildingHeight);  building.getOrCreateStateSet().setTextureAttributeAndModes(0, **osg**.**Texture**.createFromURL(**"res/textures/building.jpg"**));  translateMatrixTransform.addChild(building);  } } |

The custom geometry was used to create the wings of the aircraft which includes code for normal allowing the wings to have a texture mapped onto it.

|  |
| --- |
| **function** *Wings*() {  **var** depth = 5;   *// vertex definition* **var** vertexAttribArray = **new osg**.*BufferArray*(**osg**.*BufferArray*.**ARRAY\_BUFFER**, **null** , 3);  vertexAttribArray.setElements(**new *Float32Array***(  [ -1.0, 0.0, 0.0,  5, 0.0, 0.0,  4, 0.0, 2,  -3.0, 0.0, 2.0,  -3.0, 0.0, -5.0,  -1.0, 0.0, -6.0]));   **var** vertexAttribArrayBehind = **new osg**.*BufferArray*(**osg**.*BufferArray*.**ARRAY\_BUFFER**, **null** , 3);  vertexAttribArrayBehind.setElements(**new *Float32Array***(  [ -1.0, depth, 0.0,  -1.0, depth, -6.0,  -3.0, depth, -5.0,  -3.0, depth, 2.0,  4.0, depth, 2.0,  5.0, depth, 0.0]));   **var** vertexAttribArrayDepth = **new osg**.*BufferArray*(**osg**.*BufferArray*.**ARRAY\_BUFFER**, **null** , 3);  vertexAttribArrayDepth.setElements(**new *Float32Array***(  [  4,0,2, *// 0* 4,depth,2, *// 1* 5,0,0, *// 2* 5,depth,0, *// 3* -1,0,0, *// 4* -1,depth,0, *// 5* -3,0,2, *// 6* -3,depth,2, *// 7* -3,0,-5, *// 8* -3,depth,-5, *// 9* -1,0,-6, *// 10* -1,depth,-6 *// 11* ]));   **var** indices = **new osg**.*BufferArray*(**osg**.*BufferArray*.**ELEMENT\_ARRAY\_BUFFER**, **null**, 1 );  indices.setElements(**new *Uint16Array***(  [  1,0,3,  3,0,2,   5,3,2,  5,2,4,   1,6,0,  1,7,6,   7,9,8,  7,8,6,   9,11,10,  9,10,8,   11,5,4,  11,4,10  ]));   *// normals definition for the shape (front, back and depth)* **var** normalsGeometry = **new osg**.*BufferArray*(**osg**.*BufferArray*.**ARRAY\_BUFFER**, **null** , 3);  normalsGeometry.setElements(**new *Float32Array***(  [ 0, -1, 0,  0, -1, 0,  0, -1, 0,  0, -1, 0,  0, -1, 0,  0, -1, 0  ]));   **var** normalsGeometryBehind = **new osg**.*BufferArray*(**osg**.*BufferArray*.**ARRAY\_BUFFER**, **null** , 3);  normalsGeometryBehind.setElements(**new *Float32Array***(  [ 0, 1, 0,  0, 1, 0,  0, 1, 0,  0, 1, 0,  0, 1, 0,  0, 1, 0  ]));   **var** normalsGeometryDepth = **new osg**.*BufferArray*(**osg**.*BufferArray*.**ARRAY\_BUFFER**, **null** , 3);  normalsGeometryDepth.setElements(**new *Float32Array***(  [ 1, 0, 0,  1, 0, 0,  ***Math***.cos(***Math***.PI/4), 0 , ***Math***.cos(***Math***.PI/4),  ***Math***.cos(***Math***.PI/4), 0 , ***Math***.cos(***Math***.PI/4),  ***Math***.cos(***Math***.PI/4), 0 , -***Math***.cos(***Math***.PI/4),  ***Math***.cos(***Math***.PI/4), 0 , -***Math***.cos(***Math***.PI/4),  -***Math***.cos(***Math***.PI/4), 0 , ***Math***.cos(***Math***.PI/4),  -***Math***.cos(***Math***.PI/4), 0 , ***Math***.cos(***Math***.PI/4),  -***Math***.cos(***Math***.PI/4), 0 , -***Math***.cos(***Math***.PI/4),  -***Math***.cos(***Math***.PI/4), 0 , -***Math***.cos(***Math***.PI/4),  0,0,-1,  0,0,-1  ]));   *// front view of the shape* **var** geometry = **new osg**.**Geometry**();  geometry.setVertexAttribArray(**'Vertex'**, vertexAttribArray);  geometry.setVertexAttribArray(**'Normal'**, normalsGeometry);  geometry.getPrimitives().push(**new osg**.*DrawArrays*(**osg**.**PrimitiveSet**.TRIANGLE\_FAN, 0, 6));   *// rear view of the shape* **var** geometryBehind = **new osg**.**Geometry**();  geometryBehind.setVertexAttribArray(**'Vertex'**, vertexAttribArrayBehind);  geometryBehind.setVertexAttribArray(**'Normal'**, normalsGeometryBehind);  geometryBehind.getPrimitives().push(**new osg**.*DrawArrays*(**osg**.**PrimitiveSet**.TRIANGLE\_FAN, 0, 6));   *// depth view of the shape* **var** geometryDepth = **new osg**.**Geometry**();  geometryDepth.setVertexAttribArray(**'Vertex'**, vertexAttribArrayDepth);  geometryDepth.setVertexAttribArray(**'Normal'**, normalsGeometryDepth);  geometryDepth.getPrimitives().push(**new osg**.*DrawElements*(**osg**.**PrimitiveSet**.TRIANGLES, indices));   *// combine all views of the shape (creation of a group node)* **var** transform = **new osg**.*MatrixTransform*();  transform.addChild(geometry);  transform.addChild(geometryBehind);  transform.addChild(geometryDepth);   *// rotate and scale matrices for the wing so as to align it with the body of the plane* **var** transformMatrix = **new osg**.**Matrix**.create();  **osg**.**Matrix**.makeRotate(***Math***.PI/2,1,0,0,transformMatrix);  **var** scaleMatrix = **new osg**.**Matrix**.create();  **osg**.**Matrix**.makeScale(10,1,10, scaleMatrix);  **osg**.**Matrix**.preMult(transformMatrix,scaleMatrix);  transform.setMatrix(transformMatrix);   **return** transform; } |

Image of the aircraft can be seen below.



The action reaction of the program is done using the key input function which in turn control the rotation and the direction the plane is pointing to and move in the x and y directions.