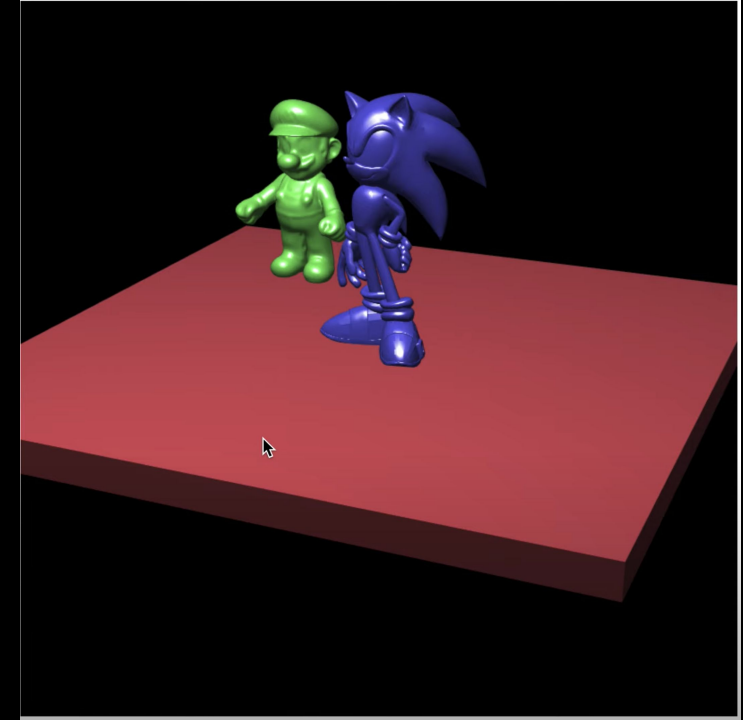




Lab 7

- Download the lab7 template
- You will load the mario and sonic model and create a cube to render the scene
 - You should use scale() to make the cube look like a ground
 - Mario and sonic should stand right on the ground
 - Maio and sonic should be rescaled to the proper size
 - Mario is static
 - User can use sliders to move and rotate the sonic
 - Nice illumination on all objects
- Please check this video, this is the result you should have
 - <https://www.youtube.com/watch?v=olb1jCNwDGE&list=PLsld7efYPyAah0Z64j9DpedSVAcvzOSKb&index=8>



Normal Vector Calculation

- I have provided the normal vector calculation function for you.
- You can pass all vertices of an object to it and it will result normal vectors on each vertex to you.
 - Input vertices: an array. All vertices of all triangles of the object are separated (e.g. a cube: 36 input vertices)

```
///// normal vector calculation (for the cube)
function getNormalOnVertices(vertices){
  var normals = [];
  var nTriangles = vertices.length/9;
  for(let i=0; i < nTriangles; i ++ ){
    var idx = i * 9 + 0 * 3;
    var p0x = vertices[idx+0], p0y = vertices[idx+1], p0z = vertices[idx+2];
    idx = i * 9 + 1 * 3;
    var p1x = vertices[idx+0], p1y = vertices[idx+1], p1z = vertices[idx+2];
    idx = i * 9 + 2 * 3;
    var p2x = vertices[idx+0], p2y = vertices[idx+1], p2z = vertices[idx+2];

    var ux = p1x - p0x, uy = p1y - p0y, uz = p1z - p0z;
    var vx = p2x - p0x, vy = p2y - p0y, vz = p2z - p0z;

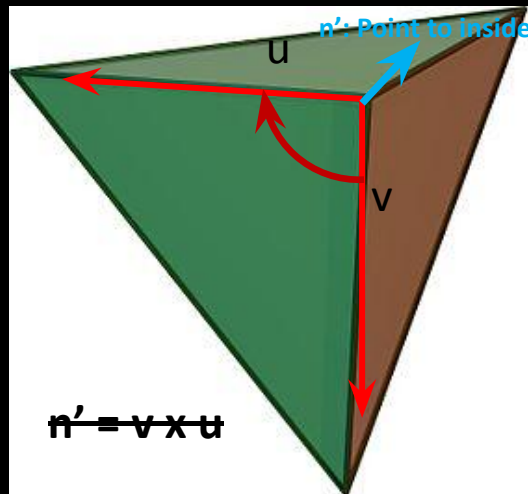
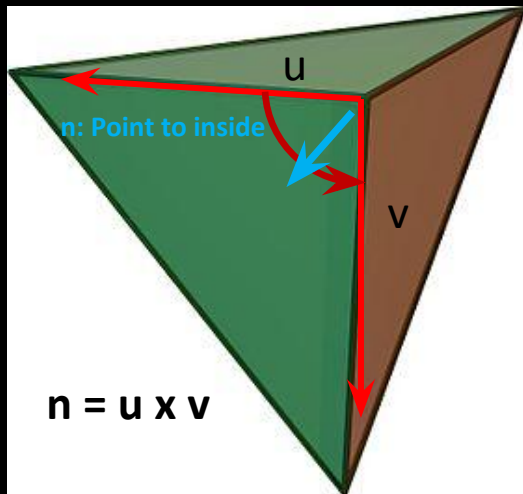
    var nx = uy*vz - uz*vy;
    var ny = uz*vx - ux*vz;
    var nz = ux*vy - uy*vx;

    var norm = Math.sqrt(nx*nx + ny*ny + nz*nz);
    nx = nx / norm;
    ny = ny / norm;
    nz = nz / norm;

    normals.push(nx, ny, nz, nx, ny, nz, nx, ny, nz);
  }
  return normals;
}
```

Normal Vector Calculation

- Normal vector can be calculated by "cross product" of two vectors which come from two edge of the triangle
 - https://en.wikipedia.org/wiki/Cross_product
- Key to correctly calculate normal vector (let's focus on the green triangle)
 - We can get normal vector which point to outside of the object or inside of the object
 - What we want is the normal vector which **points to outside** of the object



```
////// normal vector calculation (for the cube)
function getNormalOnVertices(vertices){
  var normals = [];
  var nTriangles = vertices.length/9;
  for(let i=0; i < nTriangles; i ++ ){
    var idx = i * 9 + 0 * 3;
    var p0x = vertices[idx+0], p0y = vertices[idx+1], p0z = vertices[idx+2];
    idx = i * 9 + 1 * 3;
    var p1x = vertices[idx+0], p1y = vertices[idx+1], p1z = vertices[idx+2];
    idx = i * 9 + 2 * 3;
    var p2x = vertices[idx+0], p2y = vertices[idx+1], p2z = vertices[idx+2];

    var ux = p1x - p0x, uy = p1y - p0y, uz = p1z - p0z;
    var vx = p2x - p0x, vy = p2y - p0y, vz = p2z - p0z;

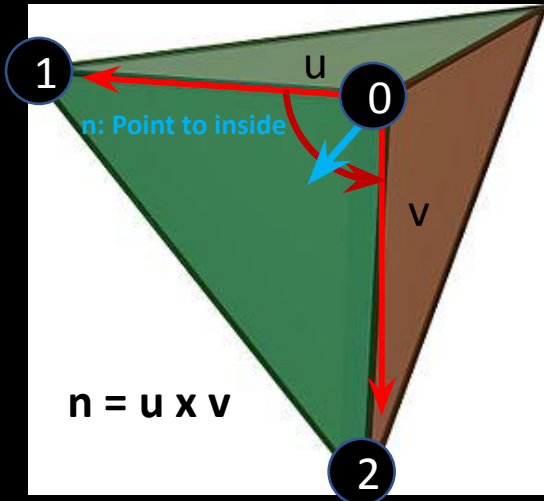
    var nx = uy*vz - uz*vy;
    var ny = uz*vx - ux*vz;
    var nz = ux*vy - uy*vx;

    var norm = Math.sqrt(nx*nx + ny*ny + nz*nz);
    nx = nx / norm;
    ny = ny / norm;
    nz = nz / norm;

    normals.push(nx, ny, nz, nx, ny, nz, nx, ny, nz);
  }
  return normals;
}
```

Normal Vector Calculation

- By the normal vector calculation function I provide
 - You should give the vertices of a triangles in counter clock wise order (when you look at the triangle from outside)
- $\mathbf{n} = \mathbf{u} \times \mathbf{v}$
 - $n_x = u_y * v_z - u_z * v_y$
 - $n_y = u_z * v_x - u_x * v_z$
 - $n_z = u_x * v_y - u_y * v_x$



```
///// normal vector calculation (for the cube)
function getNormalOnVertices(vertices){
    var normals = [];
    var nTriangles = vertices.length/9;
    for(let i=0; i < nTriangles; i ++ ){
        var idx = i * 9 + 0 * 3;
        var p0x = vertices[idx+0], p0y = vertices[idx+1], p0z = vertices[idx+2];
        idx = i * 9 + 1 * 3;
        var p1x = vertices[idx+0], p1y = vertices[idx+1], p1z = vertices[idx+2];
        idx = i * 9 + 2 * 3;
        var p2x = vertices[idx+0], p2y = vertices[idx+1], p2z = vertices[idx+2];

        var ux = p1x - p0x, uy = p1y - p0y, uz = p1z - p0z;
        var vx = p2x - p0x, vy = p2y - p0y, vz = p2z - p0z;

        var nx = uy*vz - uz*vy;
        var ny = uz*vx - ux*vz;
        var nz = ux*vy - uy*vx;

        var norm = Math.sqrt(nx*nx + ny*ny + nz*nz);
        nx = nx / norm;
        ny = ny / norm;
        nz = nz / norm;

        normals.push(nx, ny, nz, nx, ny, nz, nx, ny, nz);
    }
    return normals;
}
```


Check "TODO"s

- In main(), fill the vertices xyz of the cube in "cubeVertices"
- Let getNormalVertices() calculates the correct normal vectors for you

```
/////3D model mario
response = await fetch('mario.obj');
text = await response.text();
obj = parseOBJ(text);
for( let i=0; i < obj.geometries.length; i ++ ){
    let o = initVertexBufferForLaterUse(gl,
        obj.geometries[i].data.position,
        obj.geometries[i].data.normal,
        obj.geometries[i].data.texcoord);
    mario.push(o);
}
```

We have loaded and setup two external 3D models for you

```
/////3D model sonic
response = await fetch('sonic.obj');
text = await response.text();
obj = parseOBJ(text);
for( let i=0; i < obj.geometries.length; i ++ ){
    let o = initVertexBufferForLaterUse(gl,
        obj.geometries[i].data.position,
        obj.geometries[i].data.normal,
        obj.geometries[i].data.texcoord);
    sonic.push(o);
}
```

```
////cube
//TODO-1: create vertices for the cube whose edge length is 2.0 (or 1.0 is also fine)
//F: Face, T: Triangle
cubeVertices = [//F1_T1_X, F1_T1_Y, F1_T1_Z, F1_T2_X, F1_T2_Y, F1_T2_Z, //this row for the face z = 1.0
                //F2_T1_X, F2_T1_Y, F2_T1_Z, F2_T2_X, F2_T2_Y, F2_T2_Z, //this row for the face x = 1.0
                //F3_T1_X, F3_T1_Y, F3_T1_Z, F3_T2_X, F3_T2_Y, F3_T2_Z, //this row for the face y = 1.0
                //F4_T1_X, F4_T1_Y, F4_T1_Z, F4_T2_X, F4_T2_Y, F4_T2_Z, //this row for the face x = -1.0
                //F5_T1_X, F5_T1_Y, F5_T1_Z, F5_T2_X, F5_T2_Y, F5_T2_Z, //this row for the face y = -1.0
                //F6_T1_X, F6_T1_Y, F6_T1_Z, F6_T2_X, F6_T2_Y, F6_T2_Z, //this row for the face z = -1.0
                ];
cubeNormals = getNormalOnVertices(cubeVertices);
let o = initVertexBufferForLaterUse(gl, cubeVertices, cubeNormals, null);
cube.push(o);
```

TODO-2 -3 -4

- At the end of main(), we have registered the sliders and mouse events
 - When the slides are dragged, we will get the reading and store values into “moveDistance” and “rotateAngle”, then redraw the scene.

```
mvpMatrix = new Matrix4();
modelMatrix = new Matrix4();
normalMatrix = new Matrix4();

gl.enable(gl.DEPTH_TEST);

draw();//draw it once before mouse move

canvas.onmousedown = function(ev){mouseDown(ev)};
var slider1 = document.getElementById("move");
var slider2 = document.getElementById("rotate");
slider1.oninput = function() {
    moveDistance = this.value/60.0
    draw();
}
slider2.oninput = function() {
    rotateAngle = this.value
    draw();
}
```

TODO-2 -3 -4

- In draw(), we should set up the model matrix (without considering the mouse rotation) into mdlMatrix and call drawOneObject() to draw an object
 - Cube(ground)
 - Mario
 - Sonic

```
/////Call drawOneObject() here to draw all object one by one
//// (setup the model matrix and color to draw)
function draw(){
    gl.clearColor(0,0,0,1);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    let mdlMatrix = new Matrix4(); //model matrix of objects

    //Cube (ground)
    //TODO-1: set mdlMatrix for the cube
    drawOneObject(cube, mdlMatrix, 1.0, 0.4, 0.4);

    //mario
    //TODO-2: set mdlMatrix for mario
    //drawOneObject(mario, mdlMatrix, 0.4, 1.0, 0.4);

    //sonic
    //TODO-3: set mdlMatrix for sonic (include rotation and movement)
    //drawOneObject(sonic, mdlMatrix, 0.4, 0.4, 1.0);

    //obj: the object components
    //mdlMatrix: the model matrix without mouse rotation
    //colorR, G, B: object color
    function drawOneObject(obj, mdlMatrix, colorR, colorG, colorB){
        //model Matrix (part of the mvp matrix)
        modelMatrix.setRotate(angleY, 1, 0, 0); //for mouse rotation
        modelMatrix.rotate(angleX, 0, 1, 0); //for mouse rotation
        modelMatrix.multiply(mdlMatrix);
        //mvp: projection * view * model matrix
       .mvpMatrix.setPerspective(30, 1, 1, 100);
    }
}
```


What You Should Do for “Submission”



Submission Instruction

- Create a folder
 - Put the html and js files in the folder
 - Zip the folder
 - Rename the zip file to your student ID
 - For example, if your student ID is “40312345s”, rename the zip file to “40312345s.zip”
 - Submit the renamed zip file to Moodle
- Make sure
 - you put all files in the folder to zip
 - You submit the zip file with correct name
- You won't get any point if
 - the submitted file does not follow the naming rule,
 - TA cannot run your code,
 - or cannot unzip your zip file.