- Download the lab5 template

- You will subdivide the canvas into 3 regions (each ono has different background color)
  - One renders the scene from the **front** side with **perspective** camera model
  - Another one renders the scene from the **front** side with **orthographic** camera model
  - The last one renders the scene from the **back** side with **perspective** camera model

- Make sure all results without distortion

- User can drag to rotate the scene and the three sub views are linked

- Or you can check this video
  - https://www.youtube.com/watch?v=xylPM9S5j2I&ab_channel=Ko-ChihWang

Look into the scene from back side (perspective camera model)

Orthographic camera model (front side view)

Perspective camera model (front side view)
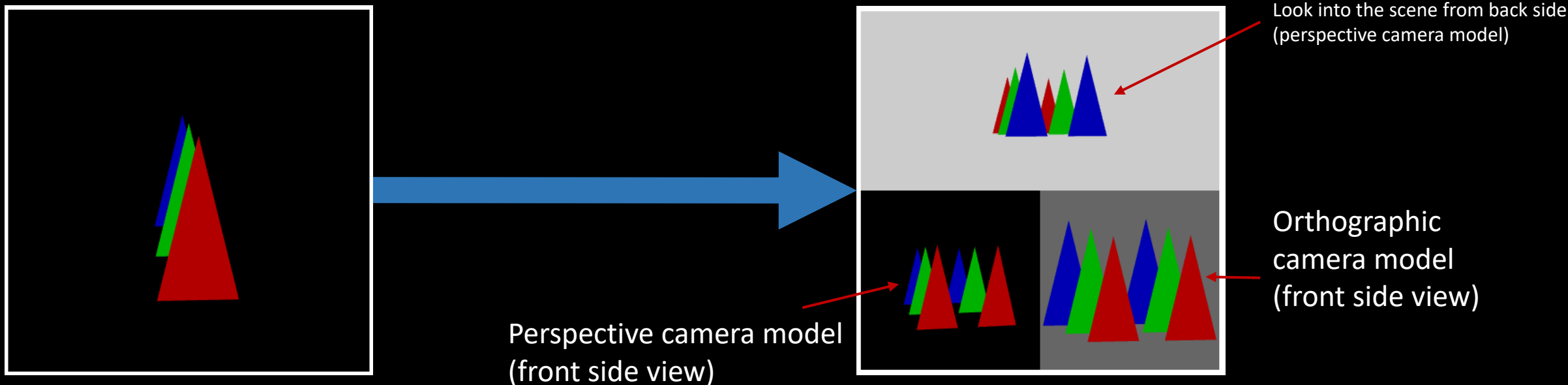
Triangles and Colors

- Here are three triangles (a set)

```javascript
var vertices = new Float32Array(//9 vertices (three triangles)
    [
        0.0, 1.0, -2.0, //x, y, z of the 1st vertex of the 1st triangle
        -0.5, -1.0, -2.0,
        0.5, -1.0, -2.0,

        0.0, 1.0, -0.0,
        -0.5, -1.0, -0.0,
        0.5, -1.0, -0.0,

        0.0, 1.0, 2.0,
        -0.5, -1.0, 2.0,
        0.5, -1.0, 2.0,
    ]
);

var colors = new Float32Array(//9 vertices (three triangles)'s color
    [
        0.7, 0.0, 0.0, //r, g, b of the 1st vertex of the 1st triangle
        0.7, 0.0, 0.0,
        0.7, 0.0, 0.0,

        0.0, 0.7, 0.0,
        0.0, 0.7, 0.0,
        0.0, 0.7, 0.0,

        0.0, 0.0, 0.7,
        0.0, 0.0, 0.7,
        0.0, 0.0, 0.7,
    ]
);
```

Viewpoint Change (Rotation)

- The idea here is to rotate the whole scene instead of moving the camera

- After we know how much we should rotate the scene from mouse information, we multiply this information into "modelMatrix"

```
var modelMatrix1 = new Matrix4();
var frontViewMatrix = new Matrix4();
var pespProjMatrix = new Matrix4();
var transformMat = new Matrix4();
var mouseLastX, mouseLastY;
var mouseDragging = false;
var angleX = 0, angleY = 0;
var canvas, gl;
```

```javascript
function main(){
    //////Get the canvas context
    canvas = document.getElementById('webgl');
    gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);
    gl.useProgram(program);

    /////prepare attribute reference of the shader
    program.a_Position = gl.getAttribLocation(program, 'a_Position');
    program.a_Color = gl.getAttribLocation(program, 'a_Color');
    program.u_mvpMatrix = gl.getUniformLocation(program, 'u_mvpMatrix');
    if(program.a_Position<0 || program.a_Color<0 || program.u_mvpMatrix < 0)
        console.log('Error: f(program.a_Position<0 || program.a_Color<0 || .....');

    /////create vertex buffer of rotating point, center points, rotating triangle for later use
    triangles = initVertexBufferForLaterUse(gl, vertices, colors);

    gl.enable(gl.DEPTH_TEST);
    gl.enable(gl.SCISSOR_TEST);//enable scissor test to only apply background clear on one viewport

    frontViewMatrix.setLookAt(0, 0, -10, 0, 0, 100, 0, 1, 0);
    pespProjMatrix.setPerspective(30, canvas.width/canvas.height, 1, 100);

    canvas.onmousedown = function(ev){mouseDown(ev)};
    canvas.onmousemove = function(ev){mouseMove(ev)};
    canvas.onmouseup = function(ev){mouseUp(ev)};
}
```
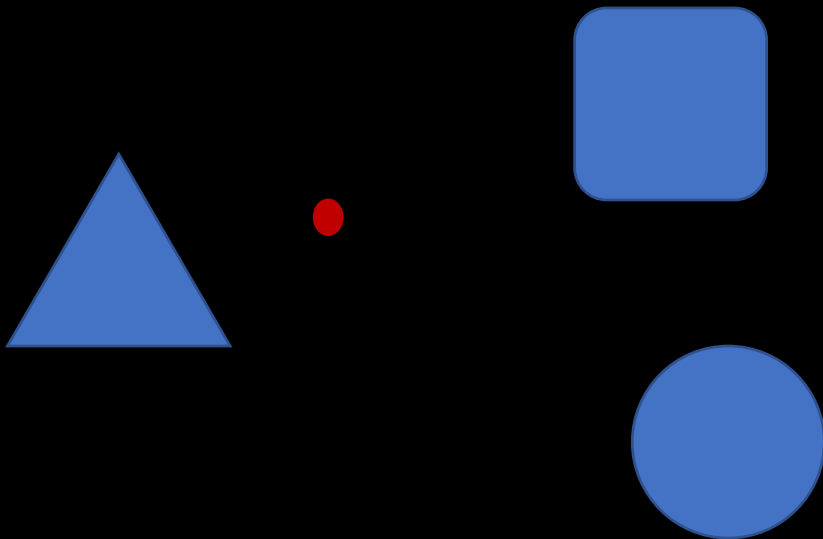
Viewpoint Change (Rotation)

What does "rotate the scene" mean?

If you have multiple objects in the scene and the
red dot is where your camera looks at,
all objects should rotate around the red dot.

```
function mouseDown(ev){                        Enable mouse dragging
    var x = ev.clientX;                        and calculate where user presses the mouse
    var y = ev.clientY;
    var rect = ev.target.getBoundingClientRect();
    if( rect.left <= x && x < rect.right && rect.top <= y && y < rect.bottom){
        mouseLastX = x;
        mouseLastY = y;
        mouseDragging = true;
    }
}

function mouseUp(ev){
    mouseDragging = false;                     Disable mouse dragging
}

function mouseMove(ev){                         When user starts to drag,
    var x = ev.clientX;                         calculate how much we should rotate the scene
    var y = ev.clientY;                         along the X-axis and Y-axis
    if( mouseDragging ){
        var factor = 100/canvas.height; //100 determine the spped you rotate the object
        var dx = factor * (x - mouseLastX);
        var dy = factor * (y - mouseLastY);

        angleX += dx;                           How much we should rotate the scene
        angleY += dy;
    }
    mouseLastX = x;
    mouseLastY = y;

    //call drawOneViewPort three times to draw the three views
    modelMatrix1.setRotate(-angleY, 1, 0, 0);  Set the rotation matrix to rotate scene
    modelMatrix1.rotate(angleX, 0, 1, 0);
    modelMatrix1.translate(0, 0, 0);

    //this only draw one set of triangles because we pass "null" for the last argument
    drawOneViewport(gl, 0, 0, canvas.width, canvas.height,
                    0, 0, 0,
                    pespProjMatrix, frontViewMatrix, modelMatrix1, null );

}
```

drawOneViewport() draws "one" viewport for you

```
function drawOneViewport(gl, viewportX, viewportY, viewportWidth, viewportHeight,
                             bgColorR, bgColorG, bgColorB,
                             projMatrix, viewMatrix, modelMatrixTriangleSet1, modelMatrixTriangleSet2 ){

    gl.viewport(viewportX, viewportY, viewportWidth, viewportHeight);
    gl.scissor(viewportX, viewportY, viewportWidth, viewportHeight);
    //scissor: make the background clear only apply to this region

    ////clear background color and depth buffer
    gl.clearColor(bgColorR, bgColorG, bgColorB, 1.0);
    gl.clear(gl.DEPTH_BUFFER_BIT | gl.COLOR_BUFFER_BIT)


    //draw a set of triangles
    transformMat.set(projMatrix);
    transformMat.multiply(viewMatrix);
    transformMat.multiply(modelMatrixTriangleSet1);
    initAttributeVariable(gl, program.a_Position, triangles.vertexBuffer);
    initAttributeVariable(gl, program.a_Color, triangles.colorBuffer);
    gl.uniformMatrix4fv(program.u_mvpMatrix, false, transformMat.elements);
    gl.drawArrays(gl.TRIANGLES, 0, triangles.numVertices);


    if( modelMatrixTriangleSet2 != null){//if we have the second modelMatrix
        //draw the second set of triangles
        transformMat.set(projMatrix);
        transformMat.multiply(viewMatrix);
        transformMat.multiply(modelMatrixTriangleSet2);
        initAttributeVariable(gl, program.a_Position, triangles.vertexBuffer);
        initAttributeVariable(gl, program.a_Color, triangles.colorBuffer);
        gl.uniformMatrix4fv(program.u_mvpMatrix, false, transformMat.elements);
        gl.drawArrays(gl.TRIANGLES, 0, triangles.numVertices);
    }

}
```

Viewport info.

Projection matrix of the camera of this viewport

Background color of the viewport

view matrix of the camera of this viewport

modelMatrix (include scene rotation info.) for the first set of triangles

modelMatrix (include scene rotation info.) for the second set of triangles

Where to draw the first set of triangles

Where to draw the second set of triangles

Scissor Test

Without enabling scissor test and set scissor region,
gl.clear() will clear whole canvas

```javascript
function main(){
    //////Get the canvas context
    canvas = document.getElementById('webgl');
    gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);
    gl.useProgram(program);

    /////prepare attribute reference of the shader
    program.a_Position = gl.getAttribLocation(program, 'a_Position');
    program.a_Color = gl.getAttribLocation(program, 'a_Color');
    program.u_mvpMatrix = gl.getUniformLocation(program, 'u_mvpMatrix');
    if(program.a_Position<0 || program.a_Color<0 || program.u_mvpMatrix < 0){
        console.log('Error: f(program.a_Position<0 || program.a_Color<0 || .....');
    }

    /////create vertex buffer of rotating point, center points, rotating triangle for later use
    triangles = initVertexBufferForLaterUse(gl, vertices, colors);

    gl.enable(gl.DEPTH_TEST);
    gl.enable(gl.SCISSOR_TEST);//enable scissor test to only apply background clear on one viewport

    frontViewMatrix.setLookAt(0, 0, -10, 0, 0, 100, 0, 1, 0);
    pespProjMatrix.setPerspective(30, canvas.width/canvas.height, 1, 100);

    canvas.onmousedown = function(ev){mouseDown(ev)};
    canvas.onmousemove = function(ev){mouseMove(ev)};
    canvas.onmouseup = function(ev){mouseUp(ev)};
}
```

```javascript
function drawOneViewport(gl, viewportX, viewportY, viewportWidth, viewportHeight,
                         bgColorR, bgColorG, bgColorB,
                         projMatrix, viewMatrix, modelMatrixTriangleSet1, modelMatrixTriangleSet2 ){

    gl.viewport(viewportX, viewportY, viewportWidth, viewportHeight);
    gl.scissor(viewportX, viewportY, viewportWidth, viewportHeight);
    //scissor: make the background clear only apply to this region

    ////clear background color and depth buffer
    gl.clearColor(bgColorR, bgColorG, bgColorB, 1.0);
    gl.clear(gl.DEPTH_BUFFER_BIT | gl.COLOR_BUFFER_BIT)

    //draw a set of triangles
    transformMat.set(projMatrix);
    transformMat.multiply(viewMatrix);
    transformMat.multiply(modelMatrixTriangleSet1);
    initAttributeVariable(gl, program.a_Position, triangles.vertexBuffer);
    initAttributeVariable(gl, program.a_Color, triangles.colorBuffer);
    gl.uniformMatrix4fv(program.u_mvpMatrix, false, transformMat.elements);
    gl.drawArrays(gl.TRIANGLES, 0, triangles.numVertices);

    if( modelMatrixTriangleSet2 != null){//if we have the second modelMatrix
        //draw the second set of triangles
        transformMat.set(projMatrix);
        transformMat.multiply(viewMatrix);
        transformMat.multiply(modelMatrixTriangleSet2);
        initAttributeVariable(gl, program.a_Position, triangles.vertexBuffer);
        initAttributeVariable(gl, program.a_Color, triangles.colorBuffer);
        gl.uniformMatrix4fv(program.u_mvpMatrix, false, transformMat.elements);
        gl.drawArrays(gl.TRIANGLES, 0, triangles.numVertices);
    }
}
```

Hints

- What you should do is just call drawOneViewport() 3 times (because you are going to draw three viewport)

- And, give each call proper "viewport", "background", "projectionMatrix", "viewMatrix" and "modelMatrix" information

```javascript
function mouseDown(ev){
    var x = ev.clientX;
    var y = ev.clientY;
    var rect = ev.target.getBoundingClientRect();
    if( rect.left <= x && x < rect.right && rect.top <= y && y < rect.bottom){
        mouseLastX = x;
        mouseLastY = y;
        mouseDragging = true;
    }
}

function mouseUp(ev){
    mouseDragging = false;
}

function mouseMove(ev){
    var x = ev.clientX;
    var y = ev.clientY;
    if( mouseDragging ){
        var factor = 100/canvas.height; //100 determine the spped you rotate the object
        var dx = factor * (x - mouseLastX);
        var dy = factor * (y - mouseLastY);

        angleX += dx;
        angleY += dy;
    }
    mouseLastX = x;
    mouseLastY = y;

    //call drawOneViewPort three times to draw the three views
    modelMatrix1.setRotate(-angleY, 1, 0, 0);
    modelMatrix1.rotate(angleX, 0, 1, 0);
    modelMatrix1.translate(0, 0, 0);

    //this only draw one set of triangles because we pass "null" for the last argument
    drawOneViewport(gl, 0, 0, canvas.width, canvas.height,
                    0, 0, 0,
                    pespProjMatrix, frontViewMatrix, modelMatrix1, null );
}
```

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.