

# CSE4204

## LAB-4 : Perspective projection, camera transformation and multiple shader

Mohammad Imrul Jubair



# Modularize your code

```
function init() {  
    var canvas = document.getElementById("webglcanvas");  
    gl = canvas.getContext("webgl");  
    initGL();  
    model();  
    draw();  
}
```

Canvas and WebGL  
context

Create and Compile  
Shaders

Associate the shader  
variable

Define model + color  
and store it in buffer

Draw object

```
var vertexShaderSource = ``;  
var fragmentShaderSource = ``;
```

//Global variables

```
function model() {  
  
function draw() {  
  
    function createProgram(gl,  
                           vertexShaderSource,  
                           fragmentShaderSource) {  
  
function initGL() {  
  
function init() {  
    init();  
}
```

# Modularize your code

```
function initGL() {  
    var prog = createProgram( gl, vertexShaderSource, fragmentShaderSource );  
    gl.useProgram(prog);  
  
    a_coords_location = gl.getAttribLocation(prog, "a_coords");  
    a_coords_buffer = gl.createBuffer();  
  
    a_colors_location = gl.getAttribLocation(prog, "a_colors");  
    a_colors_buffer = gl.createBuffer();  
  
    u_rotate_location = gl.getUniformLocation(prog, "u_rotate");  
    u_rotate_buffer = gl.createBuffer();  
  
    rotateX_location = gl.getUniformLocation(prog, "u_Rx");  
    rotateY_location = gl.getUniformLocation(prog, "u_Ry");  
}
```

Canvas and WebGL  
context

Create and Compile  
Shaders

Associate the shader  
variable

Define model + color  
and store it in buffer

Draw object

```
vertexShaderSource = ``;  
fragmentShaderSource = ``;  
  
// Global variables  
  
function model() {  
  
    function draw() {  
  
        function createProgram(gl,  
                                vertexShaderSource,  
                                fragmentShaderSource) {  
  
            function initGL() {  
  
                function init() {  
  
                }  
            }  
        }  
    }  
}
```

# Modularize your code

```
function initGL() {  
    var prog = createProgram( gl, vertexShaderSource, fragmentShaderSou  
    gl.useProgram(prog);  
}
```

```
function createProgram(gl, vertexShaderSource, fragmentShaderSource) {  
    var vsh = gl.createShader( gl.VERTEX_SHADER );  
    gl.shaderSource( vsh, vertexShaderSource );  
    gl.compileShader( vsh );  
  
    var fsh = gl.createShader( gl.FRAGMENT_SHADER );  
    gl.shaderSource( fsh, fragmentShaderSource );  
    gl.compileShader( fsh );  
  
    var prog = gl.createProgram();  
    gl.attachShader( prog, vsh );  
    gl.attachShader( prog, fsh );  
    gl.linkProgram( prog );  
  
    return prog;  
}
```

```
var vertexShaderSource = ``;  
var fragmentShaderSource = ``;
```

//Global variables

```
function model() {
```

```
function draw() {
```

```
function createProgram(gl,  
    vertexShaderSource,  
    fragmentShaderSource) {
```

```
function initGL() {
```

```
function init() {
```

```
    t();
```

Canv

Cre

Assoc

Defin  
and s

Draw object

# Modularize your code

```
function initGL() {  
    var prog = createProgram( gl, vertexShaderSource, fragmentShaderSource );  
    gl.useProgram(prog);  
  
    a_coords_location = gl.getAttribLocation(prog, "a_coords");  
    a_coords_buffer = gl.createBuffer();  
  
    a_colors_location = gl.getAttribLocation(prog, "a_colors");  
    a_colors_buffer = gl.createBuffer();  
  
    u_rotate_location = gl.getUniformLocation(prog, "u_rotate");  
    u_rotate_buffer = gl.createBuffer();  
  
    rotateX_location = gl.getUniformLocation(prog, "u_Rx");  
    rotateY_location = gl.getUniformLocation(prog, "u_Ry");  
}
```

Canvas and WebGL  
context

Create and Compile  
Shaders

Associate the shader  
variable

Define model + color  
and store it in buffer

Draw object

```
vertexShaderSource = ``;  
fragmentShaderSource = ``;  
  
// Global variables  
  
function model() {  
  
    function draw() {  
  
        function createProgram(gl,  
                                vertexShaderSource,  
                                fragmentShaderSource) {  
  
            function initGL() {  
  
                function init() {  
  
                }  
            }  
        }  
    }  
}
```

# Modularize your code

```
function model() {  
  
    coords = new Float32Array( [ // Front face  
                                -0.5, -0.5, 0.5,  
                                0.5, -0.5, 0.5,  
                                0.5, 0.5, 0.5,  
                                -0.5, 0.5, 0.5] );  
  
    colors = new Float32Array( [1.0, 0.0, 0.0,  
                                1.0, 0.0, 0.0,  
                                1.0, 0.0, 0.0,  
                                1.0, 0.0, 0.0] );  
  
    indices = new Uint8Array([ 0, 1, 2,      0, 2, 3]);  
}
```

Canvas and WebGL  
context

Create and Compile  
Shaders

Associate the shader  
variable

Define model + color  
and store it in buffer

Draw object

```
var vertexShaderSource = ``;  
var fragmentShaderSource = ``;
```

//Global variables

```
function model() {  
  
    function draw() {  
  
        function createProgram(gl,  
                                vertexShaderSource,  
                                fragmentShaderSource) {  
  
            function initGL() {  
  
                function init() {  
  
                    init();  
                }  
            }  
        }  
    }  
}
```

# Modularize your code

```
function draw() {
    var rad = thetaX*Math.PI/180;
    var rotateMatX = new Float32Array( [1.0, 0.0, 0.0, 0.0,
                                         0.0, Math.cos(rad), Math.sin(rad), 0.0,
                                         0.0, -Math.sin(rad), Math.cos(rad), 0.0,
                                         0.0, 0.0, 0.0, 1.0] );

    var rad = thetaY*Math.PI/180;
    var rotateMatY = new Float32Array( [Math.cos(rad), 0.0, -Math.sin(rad), 0.0,
                                         0.0, 1.0, 0.0, 0.0,
                                         Math.sin(rad), 0.0, Math.cos(rad), 0.0,
                                         0.0, 0.0, 0.0, 1.0] );

    gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
    gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
    gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
    gl.enableVertexAttribArray(a_coords_location);

    gl.bindBuffer(gl.ARRAY_BUFFER, a_colors_buffer);
    gl.bufferData(gl.ARRAY_BUFFER, colors, gl.STATIC_DRAW);
    gl.vertexAttribPointer(a_colors_location, 3, gl.FLOAT, false, 0, 0);
    gl.enableVertexAttribArray(a_colors_location);

    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, bufferInd);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, indices, gl.STATIC_DRAW);

    gl.uniformMatrix4fv(u_matrix_rotateX_location, false, rotateMatX);
    gl.uniformMatrix4fv(u_matrix_rotateY_location, false, rotateMatY);

    glClearColor(1.0,1.0,1.0,1.0);
    gl.enable(gl.DEPTH_TEST);
    gl.enable(gl.CULL_FACE);
    gl.glClearColor(1.0,1.0,1.0,1.0);
    gl.glEnable(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    gl.drawElements(gl.TRIANGLES, 3*12, gl.UNSIGNED_BYTE, 0);
}
```

Canvas and WebGL  
context

Create and Compile  
Shaders

Associate the shader  
variable

Define model + color  
and store it in buffer

Draw object

```
vertexShaderSource = ``;
```

```
fragmentShaderSource = ``;
```

```
var variables
```

```
function model() {
```

```
function draw() {
```

```
function createProgram(gl,
    vertexShaderSource,
    fragmentShaderSource) {
```

```
function initGL() {
```

```
function init() {
```

```
;
```

- Get the code

<https://rb.gy/7e2em9>



# Multiple shaders

```
function init() {  
    var canvas = document.getElementById("webglcanvas");  
    gl = canvas.getContext("webgl");  
  
    model();  
  
    initGL_1();  
    draw_1();  
  
    initGL_2();  
    draw_2();  
}
```

```
init();
```

# Multiple shaders

```
var vertexShaderSource_1 =  
  
`attribute vec3 a_coords;  
attribute vec3 a_colors;  
uniform mat4 u_RotY;  
uniform mat4 u_RotX;  
uniform mat4 u_Scale;  
uniform mat4 u_Trans;  
varying vec3 v_color;  
  
void main() {  
  
    mat4 M = u_Trans*u_RotX*u_RotY*u_Scale;  
    gl_Position = M*vec4(a_coords, 1.0);  
    v_color = a_colors;  
}`;
```

```
var fragmentShaderSource_1 =  
  
`precision mediump float;  
varying vec3 v_color;  
void main() {  
    gl_FragColor = vec4(v_color, 1.0);  
}`;
```

```
var vertexShaderSource_2 =  
  
`attribute vec3 a_coords;  
attribute vec3 a_colors;  
uniform mat4 u_RotY;  
uniform mat4 u_RotX;  
uniform mat4 u_Scale;  
uniform mat4 u_Trans;  
varying vec3 v_color;  
  
void main() {  
    mat4 M = u_RotX*u_RotY*u_Scale*u_Trans;  
    gl_Position = M*vec4(a_coords, 1.0);  
    v_color = a_colors;  
}`;
```

```
var fragmentShaderSource_2 =  
  
`precision mediump float;  
varying vec3 v_color;  
void main() {  
    gl_FragColor = vec4((v_color.g + v_color.b)/2.0,  
                        (v_color.r + v_color.b)/2.0,  
                        (v_color.r + v_color.g)/2.0,  
                        1.0);  
}`;
```

# Multiple shaders

```
function initGL_1() {  
    var prog1 = createProgram( gl, vertexShaderSource_1, fragmentShaderSource_1 );  
    gl.useProgram(prog1);  
  
    a_coords_location = gl.getAttribLocation(prog1, "a_coords");  
    a_coords_buffer = gl.createBuffer();  
  
    a_colors_location = gl.getAttribLocation(prog1, "a_colors");  
    a_colors_buffer = gl.createBuffer();  
  
    bufferInd = gl.createBuffer();  
  
    u_matrix_rotateX_location = gl.getUniformLocation(prog1, "u_RotX");  
    u_matrix_rotateY_location = gl.getUniformLocation(prog1, "u_RotY");  
    u_matrix_scale_location = gl.getUniformLocation(prog1, "u_Scale");  
    u_matrix_trans_location = gl.getUniformLocation(prog1, "u_Trans");  
}
```

# Multiple shaders

```
function initGL_2() {  
    var prog2 = createProgram( gl, vertexShaderSource_2, fragmentShaderSource_2 );  
    gl.useProgram(prog2);  
  
    a_coords_location = gl.getAttributeLocation(prog2, "a_coords");  
    a_coords_buffer = gl.createBuffer();  
  
    a_colors_location = gl.getAttributeLocation(prog2, "a_colors");  
    a_colors_buffer = gl.createBuffer();  
  
    bufferInd = gl.createBuffer();  
  
    u_matrix_rotateX_location = gl.getUniformLocation(prog2, "u_RotX");  
    u_matrix_rotateY_location = gl.getUniformLocation(prog2, "u_RotY");  
    u_matrix_scale_location = gl.getUniformLocation(prog2, "u_Scale");  
    u_matrix_trans_location = gl.getUniformLocation(prog2, "u_Trans");  
}
```

# Multiple shaders

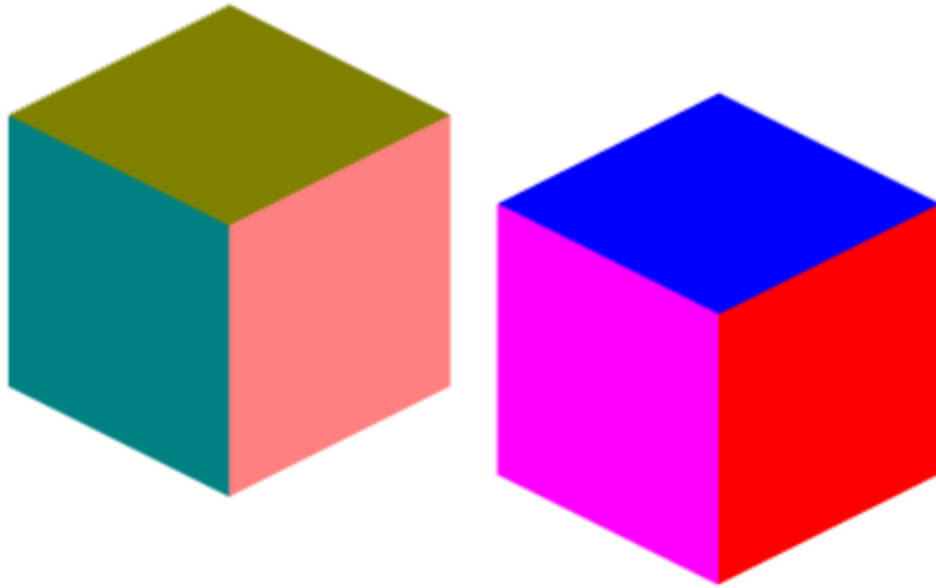
```
function draw_1 () {
```

```
    gl.clearColor(1.0,1.0,1.0,1.0);  
    gl.enable(gl.DEPTH_TEST);  
    gl.enable(gl.CULL_FACE);  
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);  
  
    gl.drawElements(gl.TRIANGLES, 3*12, gl.UNSIGNED_BYTE, 0);  
}
```

```
function draw_2 () {
```

```
    gl.drawElements(gl.TRIANGLES, 3*12, gl.UNSIGNED_BYTE, 0);  
}
```

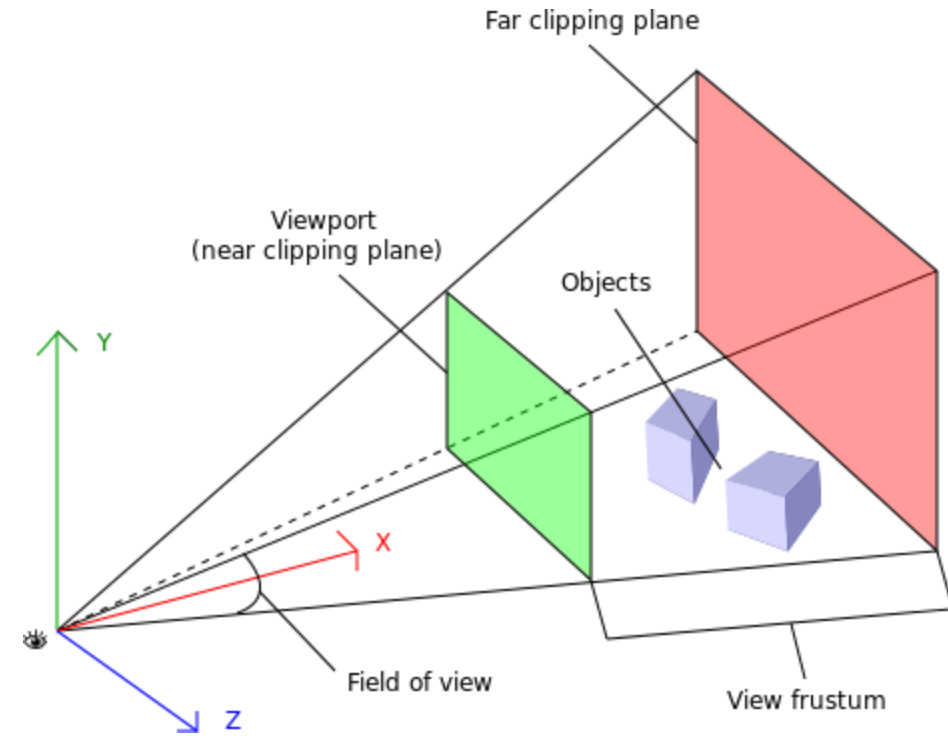
# Multiple shaders



- Get the code

<https://rb.gy/pkhinfo>

# Perspective Projection



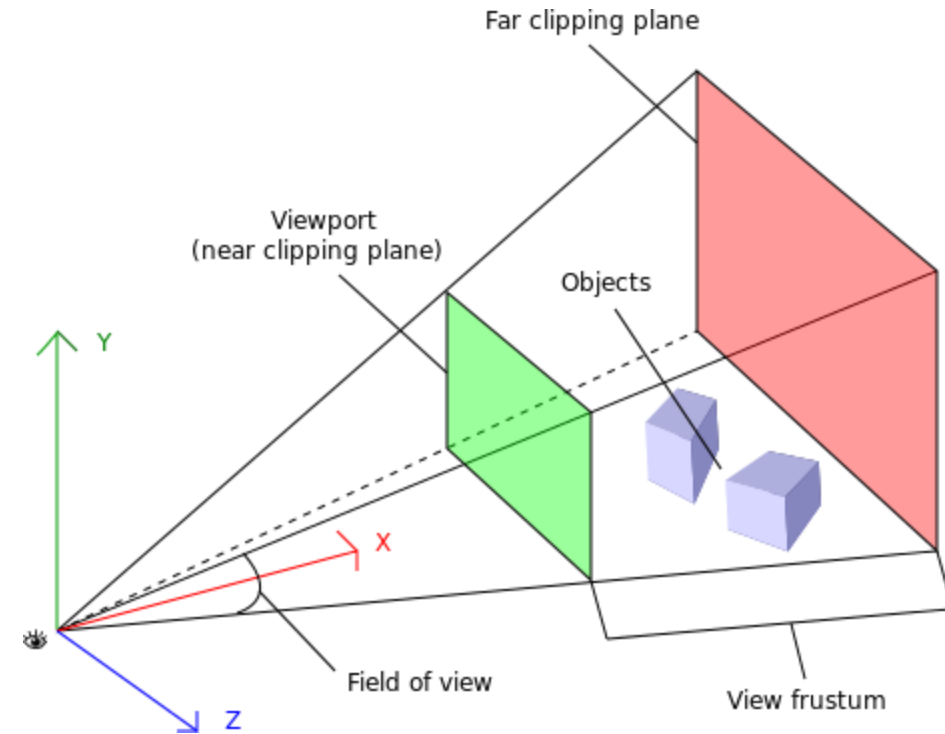
<https://www.oreilly.com/library/view/webgl-up-and/9781449326487/ch01.html>



# Perspective Projection

$$\text{persMat} = \begin{bmatrix} \frac{1}{\text{aspect} * \tan(\frac{fov}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{fov}{2})} & 0 & 0 \\ 0 & 0 & -\frac{far + near}{far - near} & -\frac{2 * far * near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

All objects must be defined in the -ve z axis. V



# Perspective Projection

```
`attribute vec3 a_coords;
attribute vec3 a_colors;
uniform mat4 u_RotY;
uniform mat4 u_RotX;
uniform mat4 u_Scale;
uniform mat4 u_Trans;
uniform mat4 u_Pers;
varying vec3 v_color;

void main() {

    mat4 M = u_Trans*u_RotX*u_RotY*u_Scale;
    gl_Position = u_Pers*M*vec4(a_coords, 1.0);
    v_color = a_colors;
}`;
```

# Perspective Projection

```
u_matrix_pers_location = gl.getUniformLocation(prog, "u_Pers");
```

```
var aspect = 1.0;
```

```
var fov = 45.0;
```

```
var far = 5.0;
```

```
var near = 2.0;
```

```
var pa = 1.0 / (aspect * Math.tan((fov/2) * Math.PI/180));
```

```
var pb = 1.0 / (Math.tan((fov/2) * Math.PI/180));
```

```
var pc = -(far+near) / (far-near);
```

```
var pd = -(2.0*far*near) / (far-near);
```

```
var persMat = new Float32Array( [pa,    0.0,    0.0,    0.0,  
                                0.0,    pb,    0,    0.0,  
                                0.0,    0.0,    pc,    -1.0,  
                                0.0,    0.0,    pd,    0.0] );
```

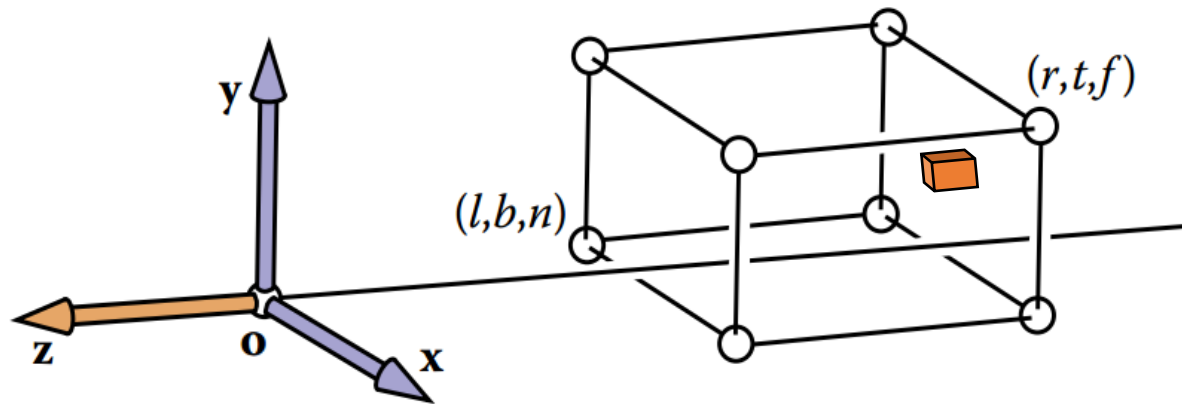
```
gl.uniformMatrix4fv(u_matrix_pers_location, false, persMat);
```

- Get the code

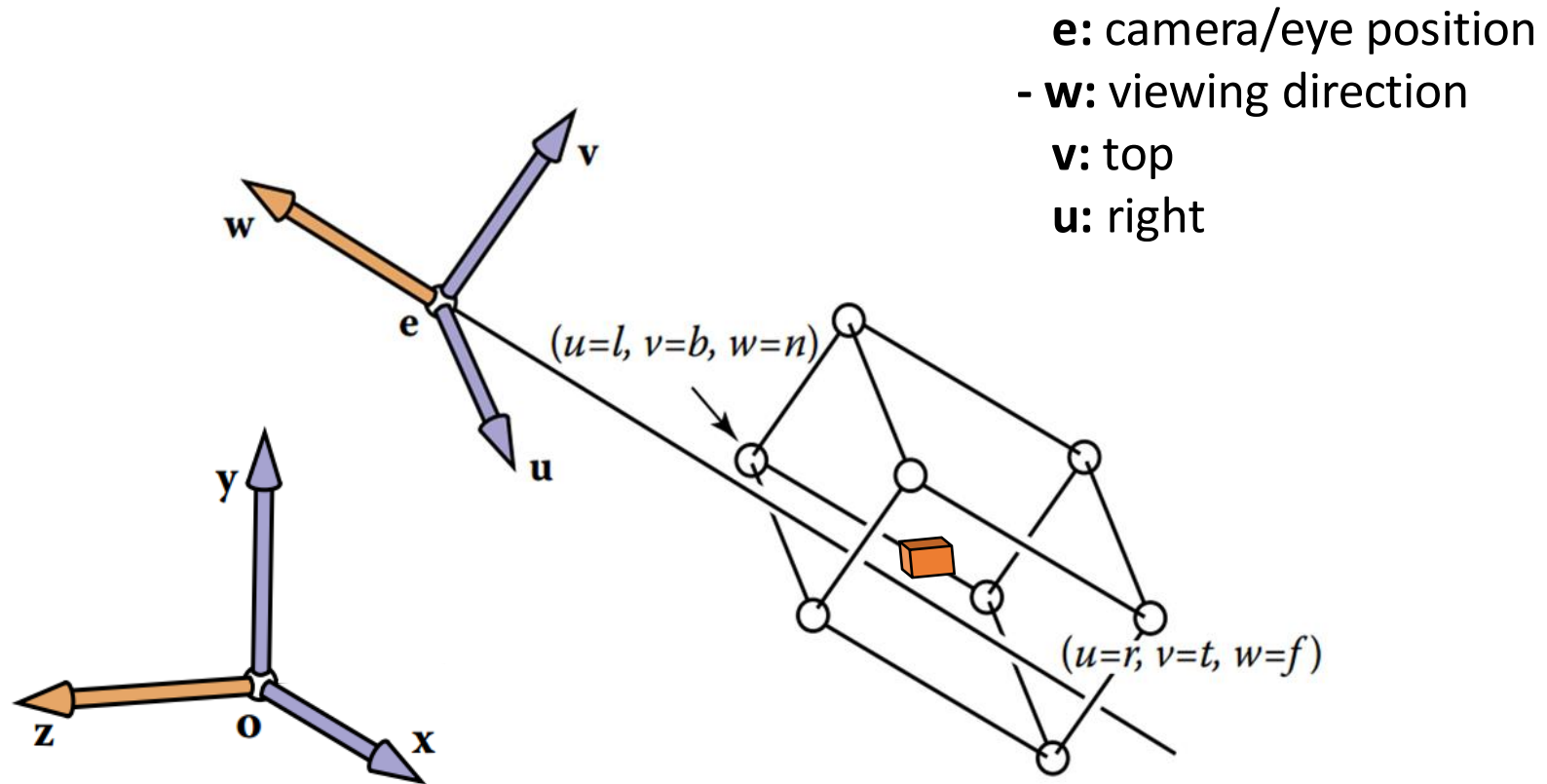
<https://rb.gy/wmjdjdic>

# Camera

- We'd like to be able to change the viewpoint in 3D and look in any direction.



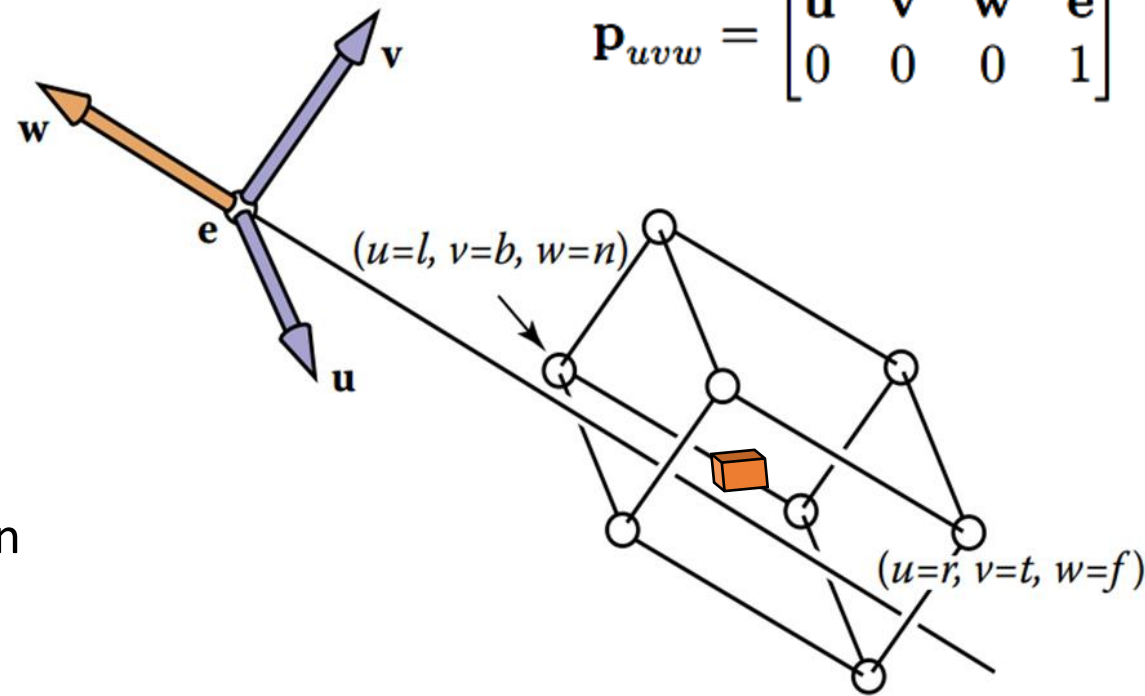
# Camera



# Camera

$$\begin{bmatrix} u_p \\ v_p \\ w_p \\ 1 \end{bmatrix} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

$$\mathbf{p}_{uvw} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \mathbf{p}_{xyz}.$$



- $\mathbf{e}$ : camera/eye position
- $\mathbf{w}$ : viewing direction
- $\mathbf{v}$ : top
- $\mathbf{u}$ : right

# Camera

```
var vertexShaderSource =  
  
    `attribute vec3 a_coords;  
    attribute vec3 a_colors;  
    uniform mat4 u_RotY;  
    uniform mat4 u_RotX;  
    uniform mat4 u_Scale;  
    uniform mat4 u_Trans;  
    uniform mat4 u_Basis;  
    uniform mat4 u_Eye;  
    uniform mat4 u_Pers;  
    varying vec3 v_color;  
  
    void main() {  
  
        mat4 M = u_Trans*u_RotX*u_RotY*u_Scale;  
        mat4 V = u_Basis*u_Eye;  
        mat4 P = u_Pers;  
        mat4 MVP = P*V*M;  
        gl_Position = MVP*vec4(a_coords, 1.0);  
        v_color = a_colors;  
    }`;
```



# Camera

```
u_matrix_basis_location = gl.getUniformLocation(prog, "u_Basis");  
u_matrix_eye_location = gl.getUniformLocation(prog, "u_Eye");
```

```
var basisMat = new Float32Array([ 1, 0, 0, 0,  
                                   0, 1, 0, 0,  
                                   0, 0, 1, 0,  
                                   0, 0, 0, 1]);  
  
var xe = 0.5;  
var ye = 1.0;  
var ze = 3.0;  
var eyeMat = new Float32Array([1, 0, 0, 0,  
                                0, 1, 0, 0,  
                                0, 0, 1, 0,  
                                -xe, -ye, -ze, 1]);
```

```
gl.uniformMatrix4fv(u_matrix_basis_location, false, basisMat);  
gl.uniformMatrix4fv(u_matrix_eye_location, false, eyeMat);
```

- Get the code

<https://rb.gy/4vmkdg>