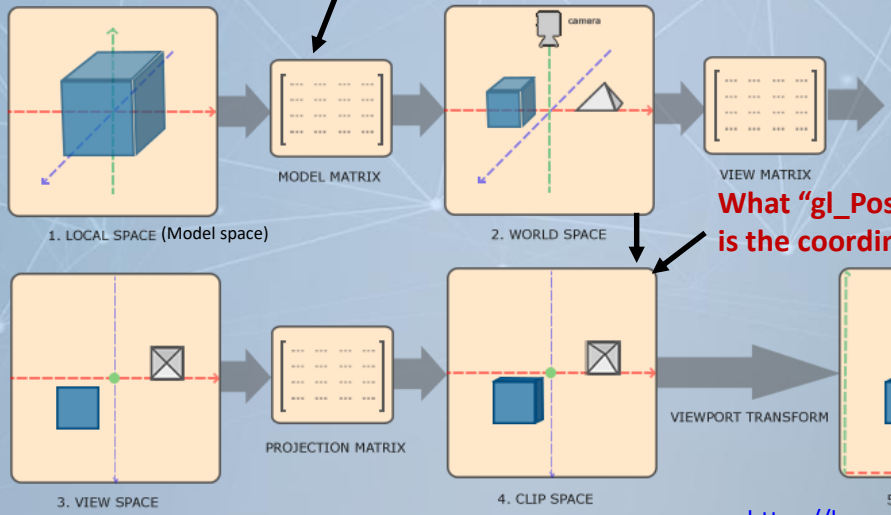
Space Transformation Pipeline (3D)

CSU0021: Computer Graphics

Coordinate System (Space)

- We will talk about all these matrices/transformations one by one
  - This space transformation pipeline exactly tells you where to show a 3D point in the 2D screen space

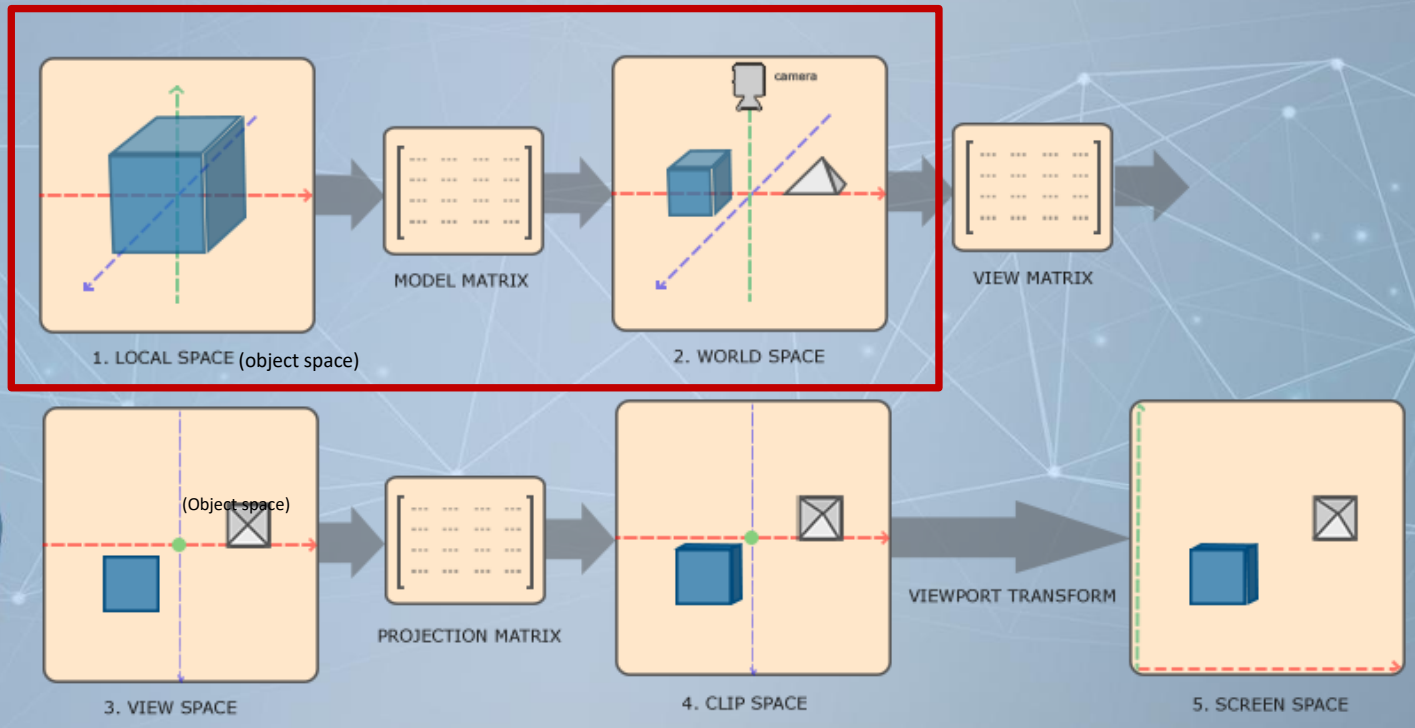
What we learn in the past two weeks is this one (object space to world space)



we assume our viewport is the same as the screen space in the past four weeks

https://learnopengl.com/Getting-started/Coordinate-Systems
http://www.codinglabs.net/article_world_view_projection_matrix.aspx

Model Matrix (Model to World Space)



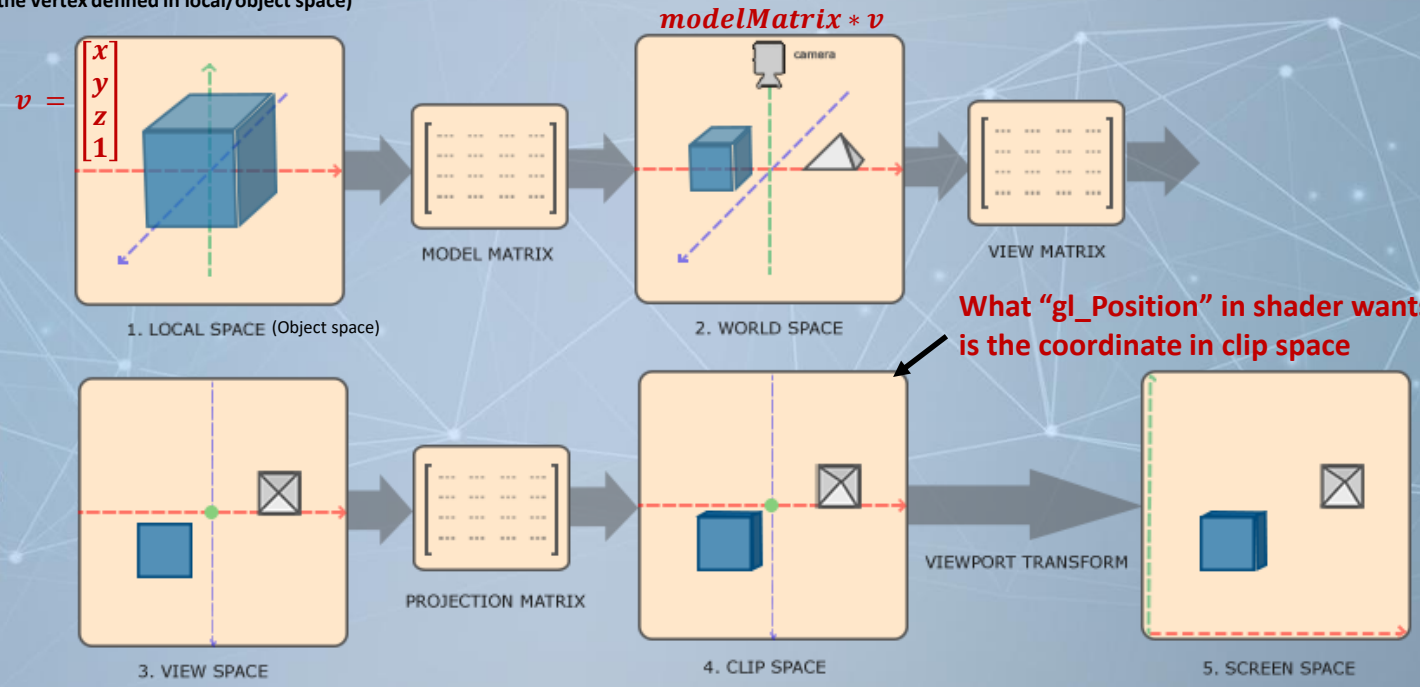
https://learnopengl.com/Getting-started/Coordinate-Systems
http://www.codinglabs.net/article_world_view_projection_matrix.aspx

Model Matrix (Model to World Space)

- Convert each model (e.g. a teapot's) local coordinates to world space
  - The model matrix usually consists of **rotation, translation and scaling matrix** (what we learn in the past two weeks)
- Each teapot has its own model matrix to transform their vertices to world space
- All vertices of a teapot use the same model matrix
  - $v_{tp1}^{W} = M_{tp1}^{M} * v_{tp1}^{O}$
  - $v_{tp2}^{W} = M_{tp2}^{M} * v_{tp2}^{O}$
  - $v_{tp3}^{W} = M_{tp3}^{M} * v_{tp3}^{O}$

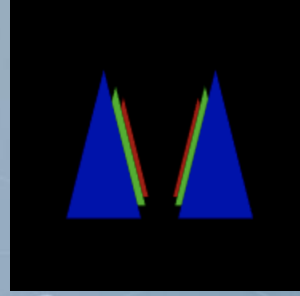
Coordinate System (Space)

v is what the coordinate you put in the VBO and pass to shader
(that is the vertex defined in local/object space)

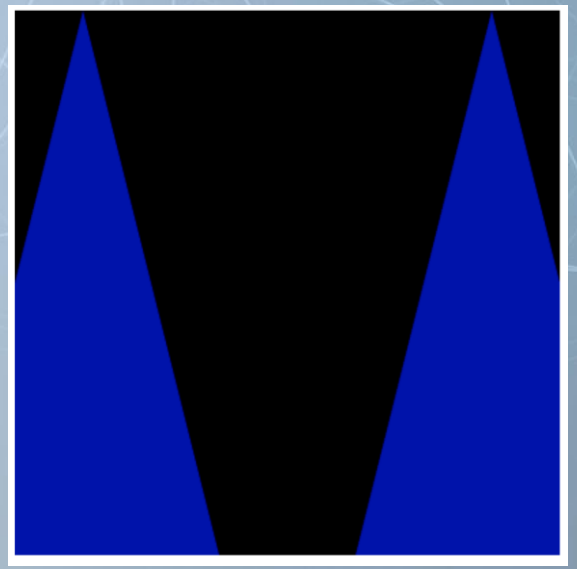


https://learnopengl.com/Getting-started/Coordinate-Systems
http://www.codinglabs.net/article_world_view_projection_matrix.aspx

Example (Ex05-1)



- In the end of today, we will draw six triangles
- In Ex05-1, we only have the model matrix and we can only see a portion of two triangles



- Files:
  - Index.html
  - WebGL.js
  - Cuon-matrix.js

Example (Ex05-1)

Shaders

```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec4 a_Color;
    uniform mat4 u_ModelMatrix;
    varying vec4 v_Color;
    void main(){
        gl_Position = u_ModelMatrix * a_Position;
        v_Color = a_Color;
    }
`;

var FSHADER_SOURCE = `
    precision mediump float;
    varying vec4 v_Color;
    void main(){
        gl_FragColor = v_Color;
    }
`;
```

initVertexBuffers() in WebGL.js

```
function initVertexBuffers(gl, program){
    var vertices = new Float32Array(//9 vertices (three triangles)
        [
            0.0, 1.0, -4.0, 0.7, 0.0, 0.0, // x, y, z, r, g, b of the first vertex
            -0.5, -1.0, -4.0, 0.7, 0.0, 0.0,
            0.5, -1.0, -4.0, 0.7, 0.0, 0.0,

            0.0, 1.0, -2.0, 0.0, 0.7, 0.0,
            -0.5, -1.0, -2.0, 0.0, 0.7, 0.0,
            0.5, -1.0, -2.0, 0.0, 0.7, 0.0,

            0.0, 1.0, 0.0, 0.0, 0.0, 0.7,
            -0.5, -1.0, 0.0, 0.0, 0.0, 0.7,
            0.5, -1.0, 0.0, 0.0, 0.0, 0.7,
        ]
       );

    var n = 9;

    var vertexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
    var FSIZE = vertices.BYTES_PER_ELEMENT;
    var a_Position = gl.getAttribLocation(program, 'a_Position');
    gl.vertexAttribPointer(a_Position, 3, gl.FLOAT, false, FSIZE*6, 0);
    gl.enableVertexAttribArray(a_Position);

    var a_Color = gl.getAttribLocation(program, 'a_Color');
    gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, FSIZE*6, FSIZE*3);
    gl.enableVertexAttribArray(a_Color);

    return n;
}
```

The first triangle (red)

The second triangle (blue)

The third triangle (blue)

Example (Ex05-1)

main() in WebGL.js

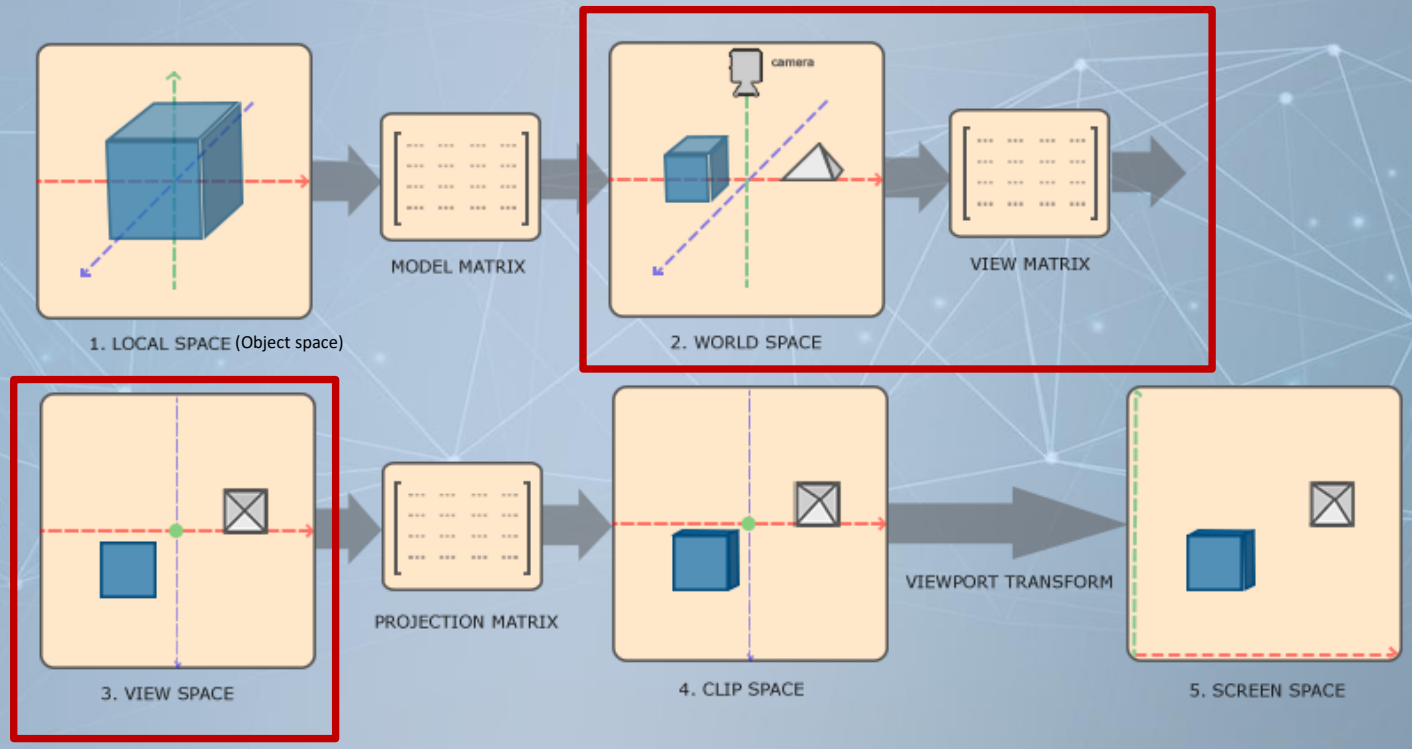
```
function main(){
    //////Get the canvas context
    var canvas = document.getElementById('webgl');
    var gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);

    gl.useProgram(program);

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);

    var n = initVertexBuffers(gl, program);

    var u_ModelMatrix = gl.getUniformLocation(program, 'u_ModelMatrix');

    var modelMatrix = new Matrix4();

    modelMatrix.setTranslate(0.75, 0, 0);
    gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
    gl.drawArrays(gl.TRIANGLES, 0, n);

    modelMatrix.setTranslate(-0.75, 0, 0);
    gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
    gl.drawArrays(gl.TRIANGLES, 0, n);
}
```

translate the triangle set to draw
(do it twice with different translation
to draw two sets of triangles)

Let’s try and think (5mins)

- Take a close look at triangle vertices array (x, y of each triangle are the same, but z is different)
- Can you apply more transformations to the model matrix and let you can see the all triangles?
- I already show you the long pipeline of 3D rendering, but we do not have the view matrix, projection matrix and viewport in the code. If so, what happens in the rendering pipeline?

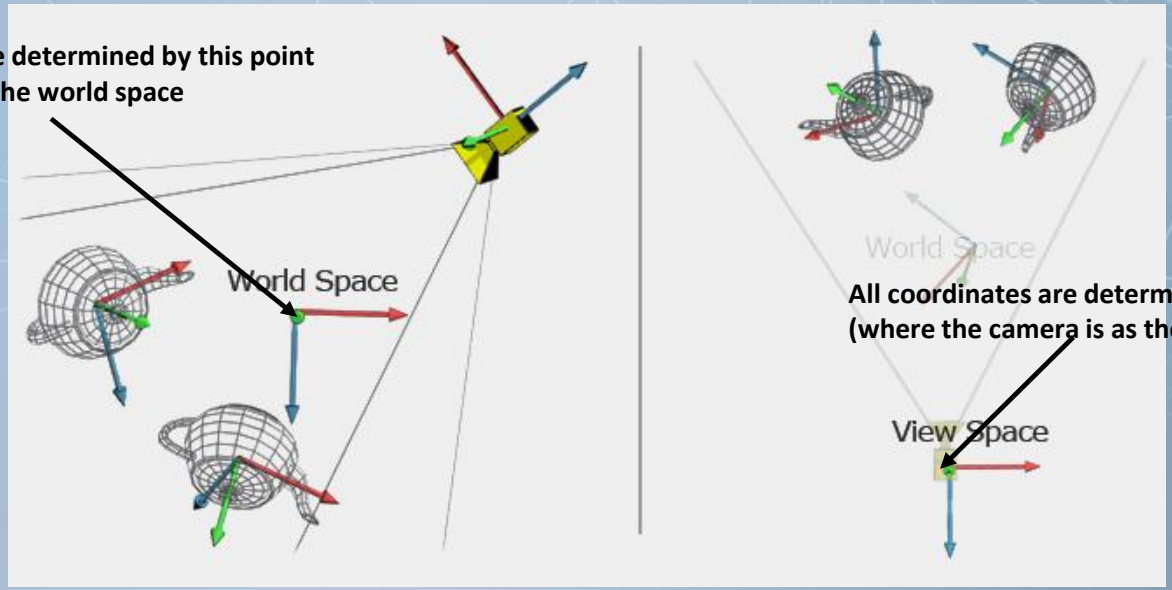
View Matrix (World to View Space)



https://learnopengl.com/Getting-started/Coordinate-Systems
http://www.codinglabs.net/article_world_view_projection_matrix.aspx

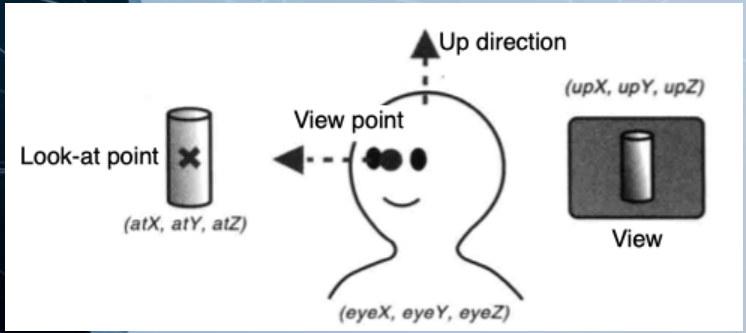
View Matrix (World to View Space)

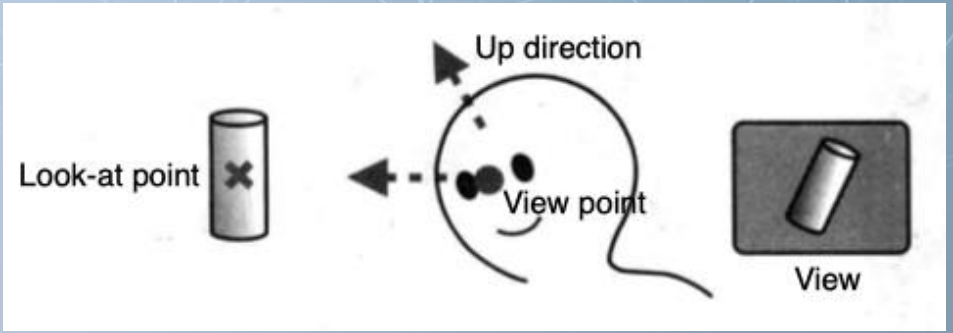
- This transformation is going to use the position of the camera as the origin to recalculate new coordinates of all vertices



View Matrix (World to View Space)

- Transform coordinates from the world space to the camera (eye) space
  - Make the following transformation (project and viewport) easily
- How to define the view space?
  - Camera position
  - Camera looking direction
  - Camera up direction

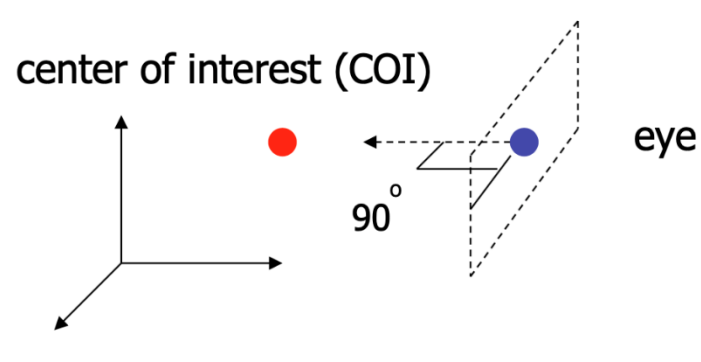




Build the View Space Coordinate System

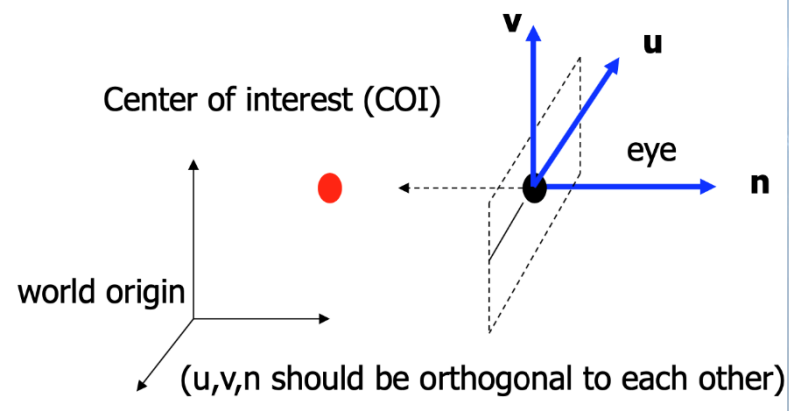
- Known: eye position, look-at point (COI), view-up vector
- To find out: new origin and three basis vectors

Assumption: the direction of view is orthogonal
to the view plane (the plane objects will project into)



Build the View Space Coordinate System

- Origin: eye position
- Three basis vectors
  - Normal vector (n) of the view plane
  - The other two vectors are the ones (u and v) that span the viewing plane



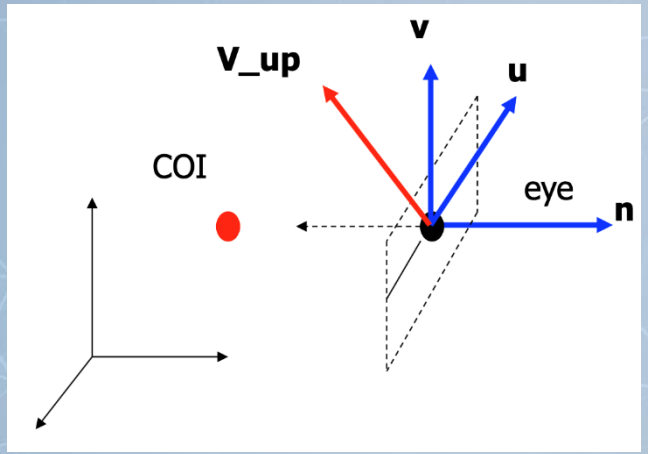
n is pointing away from the world because we use right hand coordinate system

N = eye – COI

$n = N / \mid N \mid$

Build the View Space Coordinate System

- How to get u and v



u first

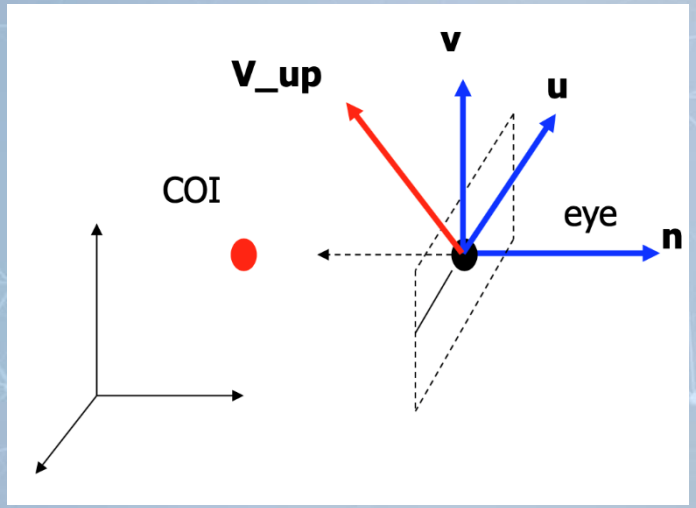
垂直

u is a vector that is perpendicular to the plane spanned by n and view up vector

$$U = v_{up} \times n$$
$$u = U / |U|$$

Build the View Space Coordinate System

- How about v

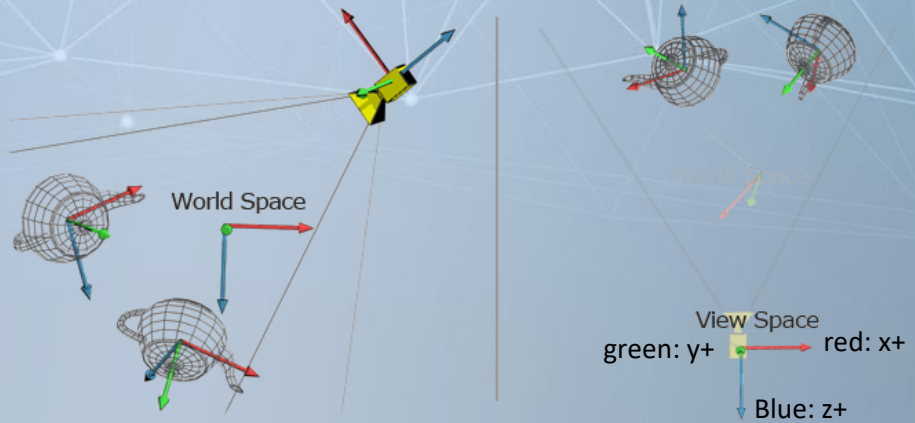


$v = n \times u$

v is already normalized

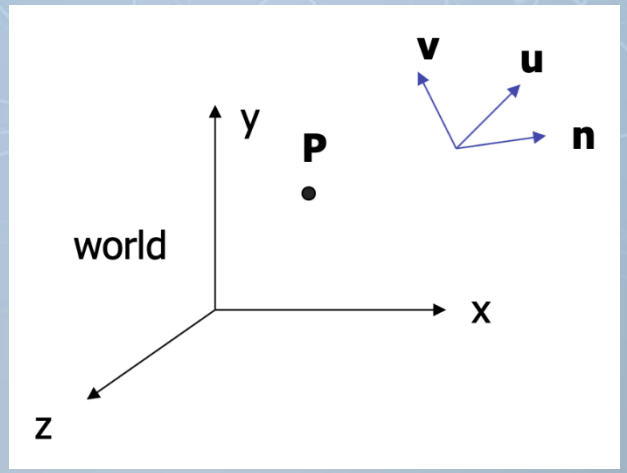
World to View Space in Mathematics

- Transform coordinates from the world space to the camera (eye) space
  - Make the following transformation (project and viewport) easily
- If you have the view matrix, you can transform any coordinates in world space to camera (eye) space (all coordinates/objects in a view share one view matrix)
  - $v^V = M^V * v^W$
  - The view matrix (M^V) essentially consists of a translation and a rotation matrix



World to View Space in Mathematics

- $\boldsymbol{v^V = M^V * v^W} = M^V * M^M * v^O$
- Why not $M^M * M^V * v^O$?
  - Come up with the transformation sequence to **move** view space coordinate frame to world



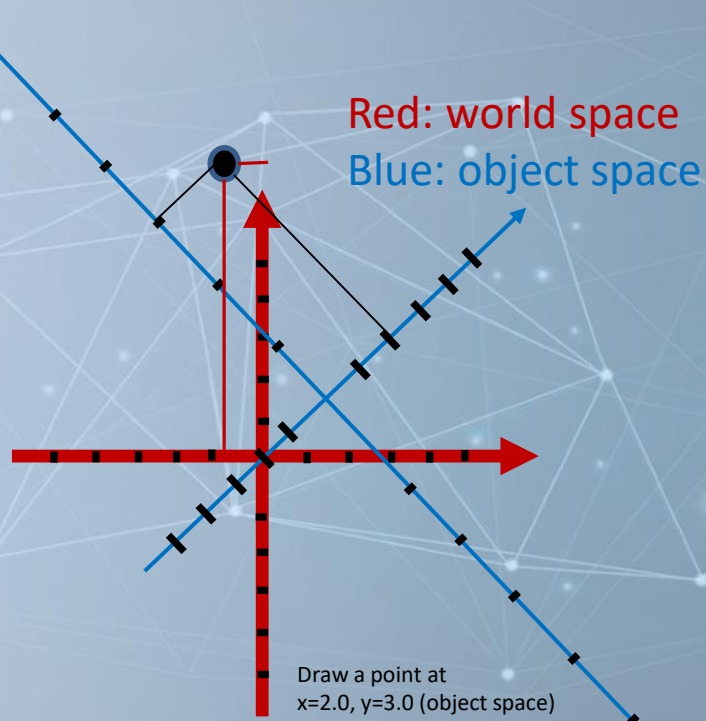
World to View Space in Mathematics

- Transformations describes the relation between two coordinate systems (space)
- If we know a point coordinate defined in blue space and know the relative position between red and blue spaces. And ask what the coordinate of the point in the red space
  - A sequence of transformations can move the red coordinate system to blue one (and we know the order of multiplication of matrices).

known p_{blue}, what is p_{red}?
Transformations to move red coordinate system to blue one

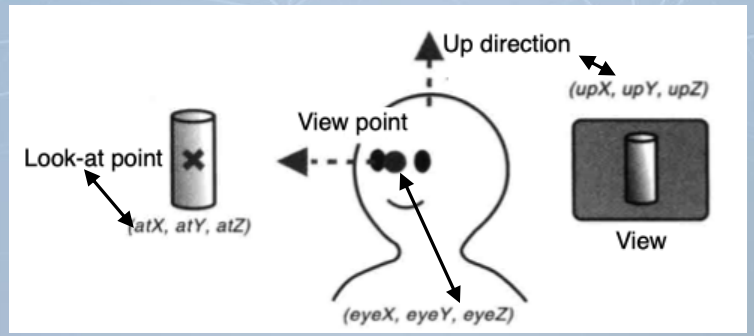
known p_{obj}, what is p_{view}?
Transformations to move view coordinate system to object one.

So, $p_{view} = M^{view} * M^{model} * p_{obj}$



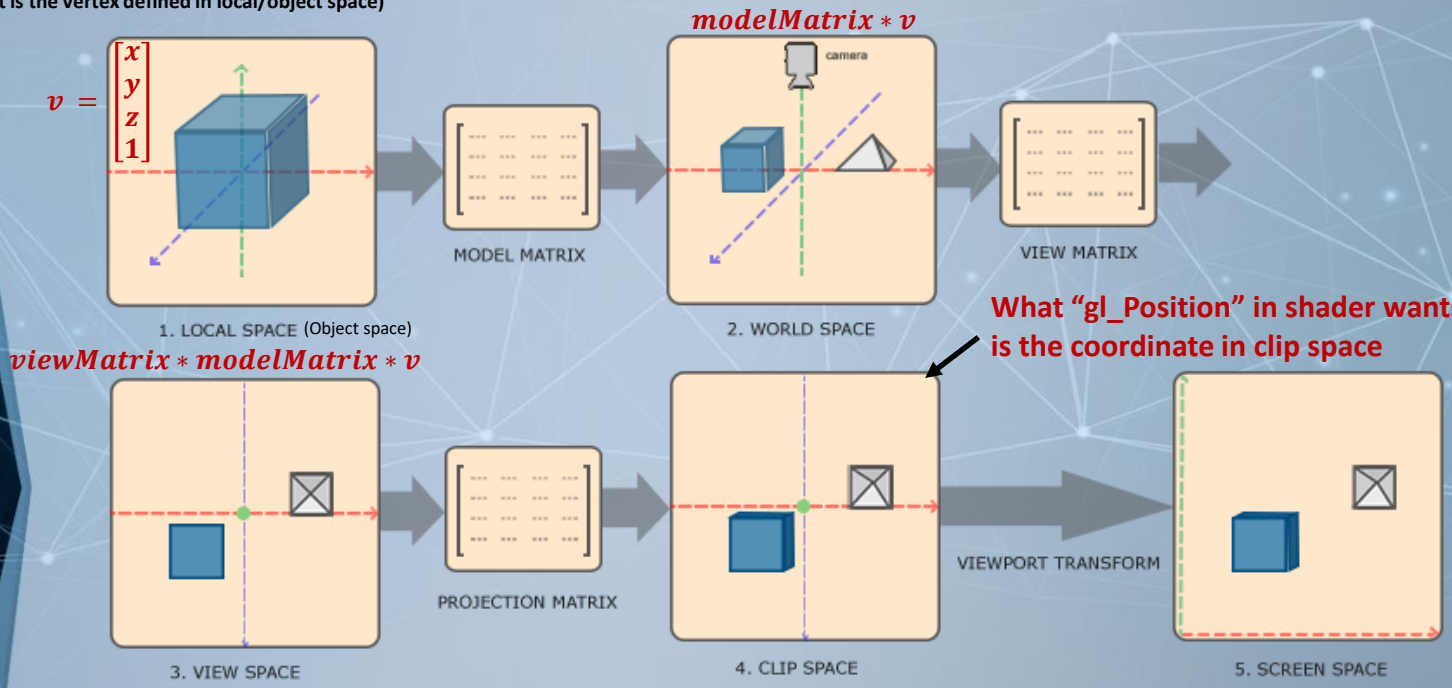
Build View Matrix by cuon-matrix.js

- How to create a view matrix?
  - ”setLookAt()” in cuon-matrix.js
- Matrix4.setLookAt(eyeX, eyeY, eyeZ, atX, atY, atZ, upX, upY, upZ)
  - eyeX, eyeY, eyeZ: camera position in world space
  - atX, atY, atZ: looking target in world space
  - upX, upY, upZ: up direction vector in world space



Coordinate System (Space)

v is what the coordinate you put in the VBO and pass to shader
(that is the vertex defined in local/object space)



https://learnopengl.com/Getting-started/Coordinate-Systems
http://www.codinglabs.net/article_world_view_projection_matrix.aspx

Example (Ex05-2)

- Files:
  - Index.html
  - WebGL.js
  - Cuon-matrix.js

Example (Ex05-2)

Shader

```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec4 a_Color;
    uniform mat4 u_ModelMatrix;
    uniform mat4 u_ViewMatrix;
    varying vec4 v_Color;
    void main(){
        gl_Position = u_ViewMatrix * u_ModelMatrix * a_Position;
        v_Color = a_Color;
    }
`;

var FSHADER_SOURCE = `
    precision mediump float;
    varying vec4 v_Color;
    void main(){
        gl_FragColor = v_Color;
    }
`;
```

```
function main(){
    //////Get the canvas context
    var canvas = document.getElementById('webgl');
    var 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);

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);

    var n = initVertexBuffers(gl, program);

    var u_ModelMatrix = gl.getUniformLocation(program, 'u_ModelMatrix');
    var u_ViewMatrix = gl.getUniformLocation(program, 'u_ViewMatrix');

    var modelMatrix = new Matrix4();
    var viewMatrix = new Matrix4();

    viewMatrix.setLookAt(0, 0, 1, 0, 0, -100, 0, 1, 0);
    gl.uniformMatrix4fv(u_ViewMatrix, false, viewMatrix.elements);

    modelMatrix.setTranslate(0.75, 0, 0);
    gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
    gl.drawArrays(gl.TRIANGLES, 0, n);

    modelMatrix.setTranslate(-0.75, 0, 0);
    gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
    gl.drawArrays(gl.TRIANGLES, 0, n);
}
```

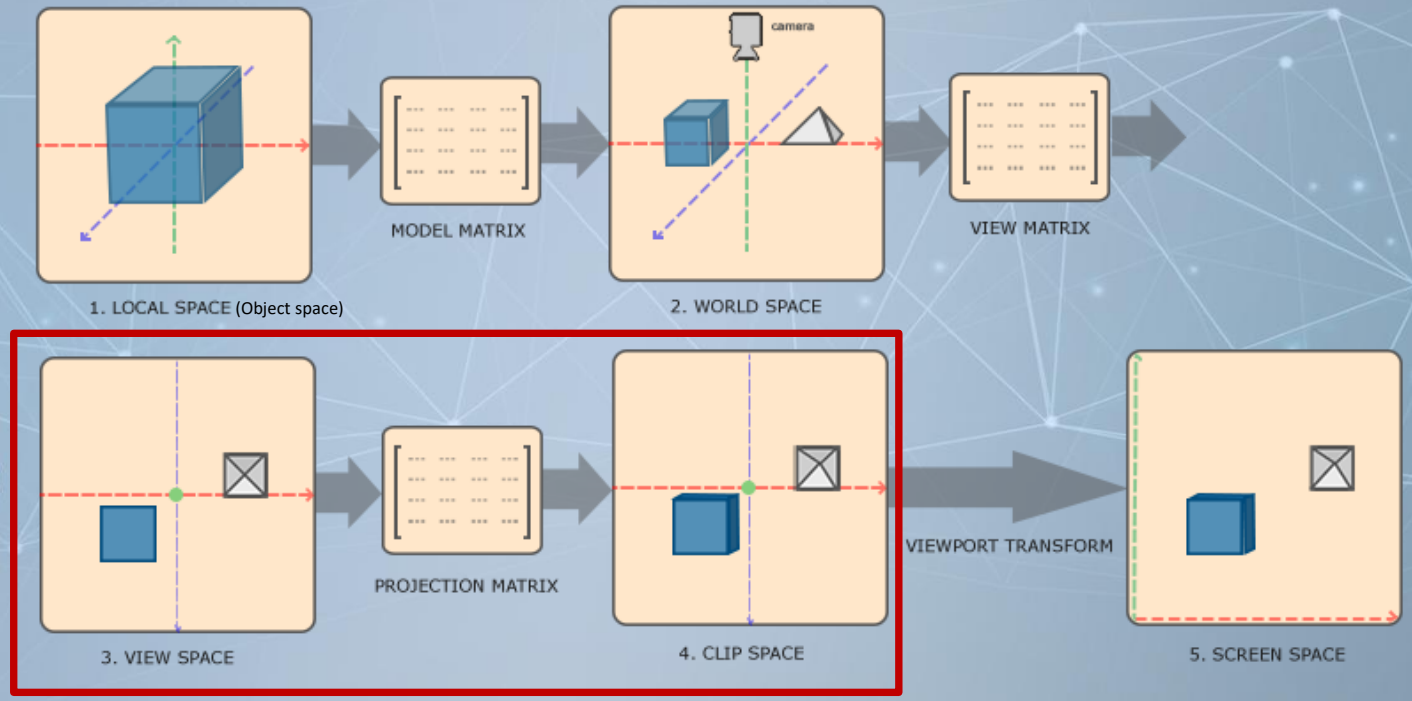
Let’s try and think (5mins)

- Try to modify the arguments in setLookAt() function to see the scene from other view angles (you will be able to see other triangles)
  - Make sure you know what is going on

- What you see if you modify setLookAt() to
  - setLookAt(0, 0, 1, 0, 0, -100, 1, 0, 0);
  - setLookAt(-0.5, -0.5, -0.5, 0, 0, 0, 0, 1, 0);
  - setLookAt(-1, -1, -4.6, 0, 0, 0, 0, 1, 0);
  - You can try more!!

- From the original Ex05-2 code
  - Modify setLookAt() to setLookAt(0, 0, 2, 0, 0, -100, 0, 1, 0)
  - Can you still see the triangles? (why)

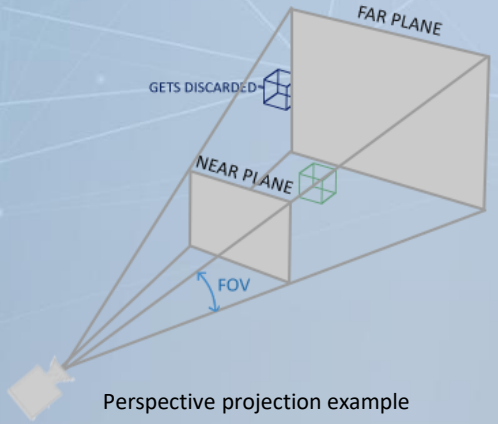
Projection Matrix (View to Clip Space)



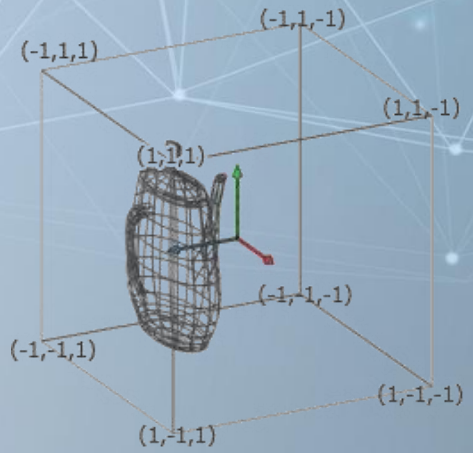
https://learnopengl.com/Getting-started/Coordinate-Systems
http://www.codinglabs.net/article_world_view_projection_matrix.aspx

Projection Matrix (View to Clip Space)

- Projection: map object/coordinates from 3D space to 2D screen
  - project everything in the visible volume into **clip space**
    - Clip space is described by **normalized device coordinate (NDC)** : a cube whose x, y, z axis ranges are between **(-1,-1,-1) to (1,1,1)**
  - Transform objects' coordinates from view space to clip space
    - Why is it called clip space? After the projection matrix is defined, only the coordinates inside the region defined by the projection matrix will be included in the clip space

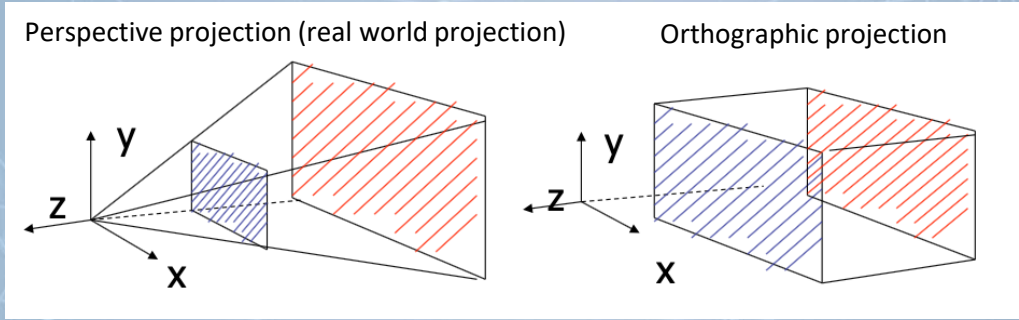


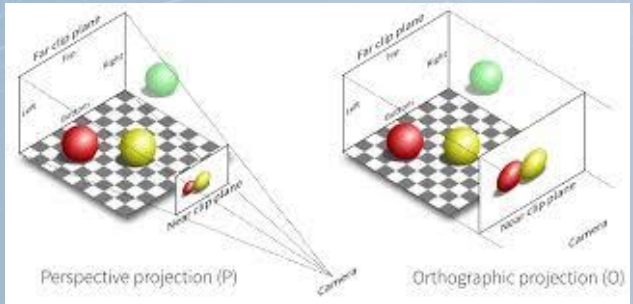
Perspective projection example

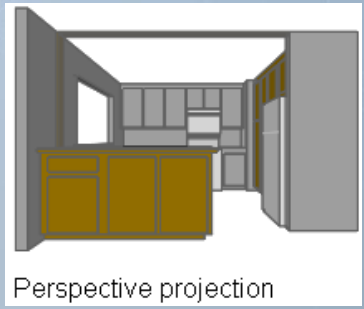


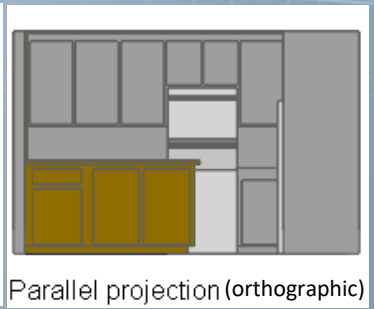
Projection Matrix (View to Clip Space)

- Two types of projection:
  - **perspective** and **orthographic** projection



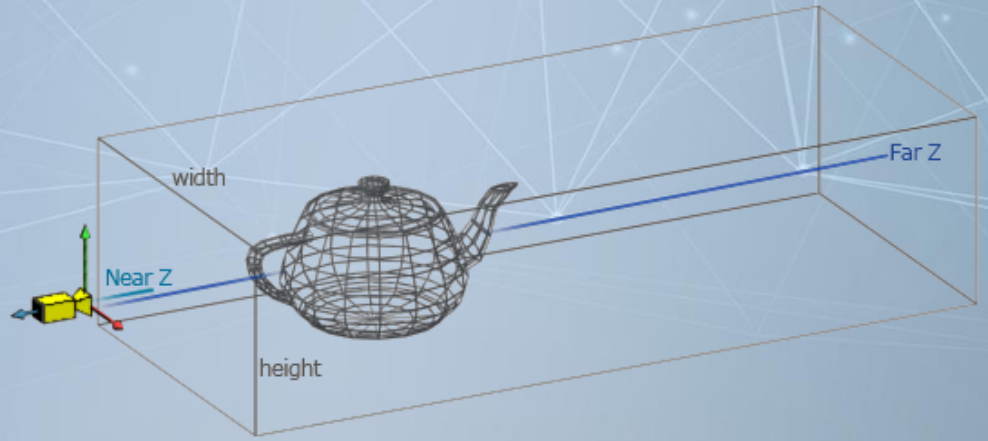






Orthographic Projection Matrix (View to Clip Space)

- **Orthographic project**: all objects appear at the same scale
- To do the orthographic projection, you should define the size of area that camera can see (width for x-axis, height for y-axis, z-near and z-far for z-axis)



Orthographic Projection Matrix (View to Clip Space)

Vertices in clip space

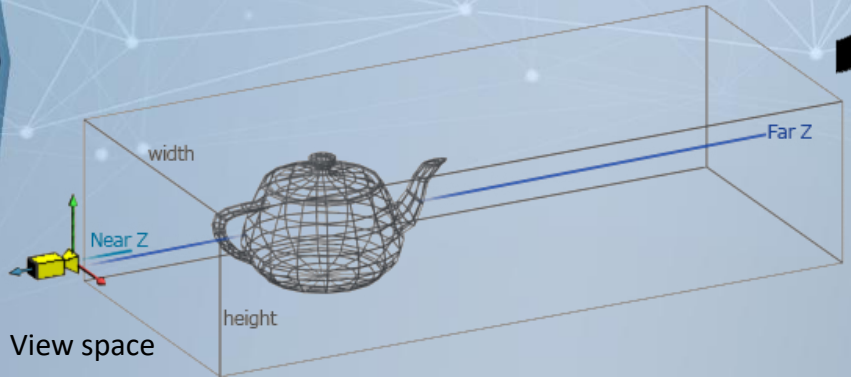
$$v^C = M^P * v^V$$

Projection matrix

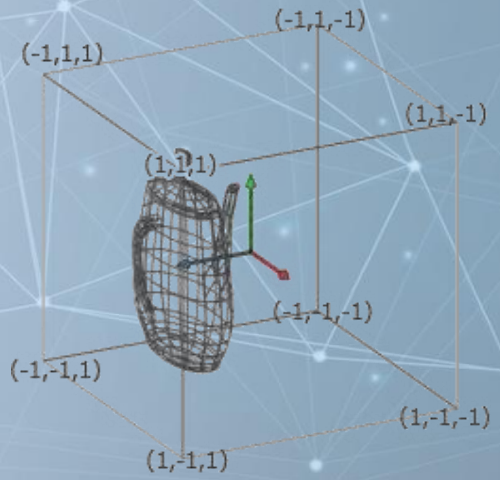
Vertices in view space

Orthographic projection matrix
(simple version: if xmin == -xmax and ymin==-ymax)

$$\begin{bmatrix} \frac{1}{width} & 0 & 0 & 0 \\ 0 & \frac{1}{height} & 0 & 0 \\ 0 & 0 & -\frac{2}{Z_{far} - Z_{near}} & -\frac{Z_{far} + Z_{near}}{Z_{far} - Z_{near}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



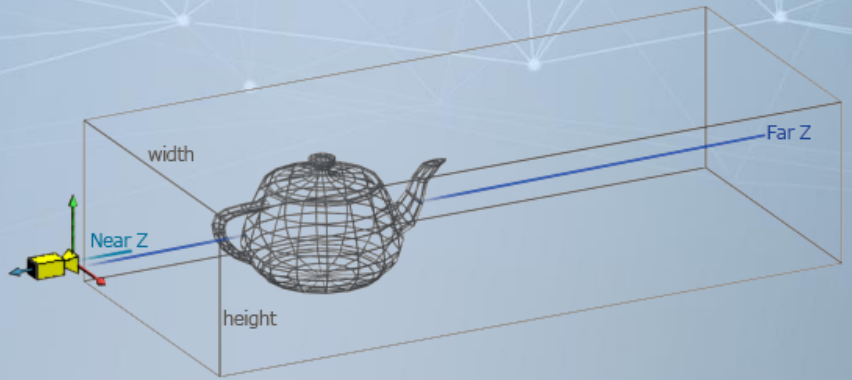
View space



Clip space

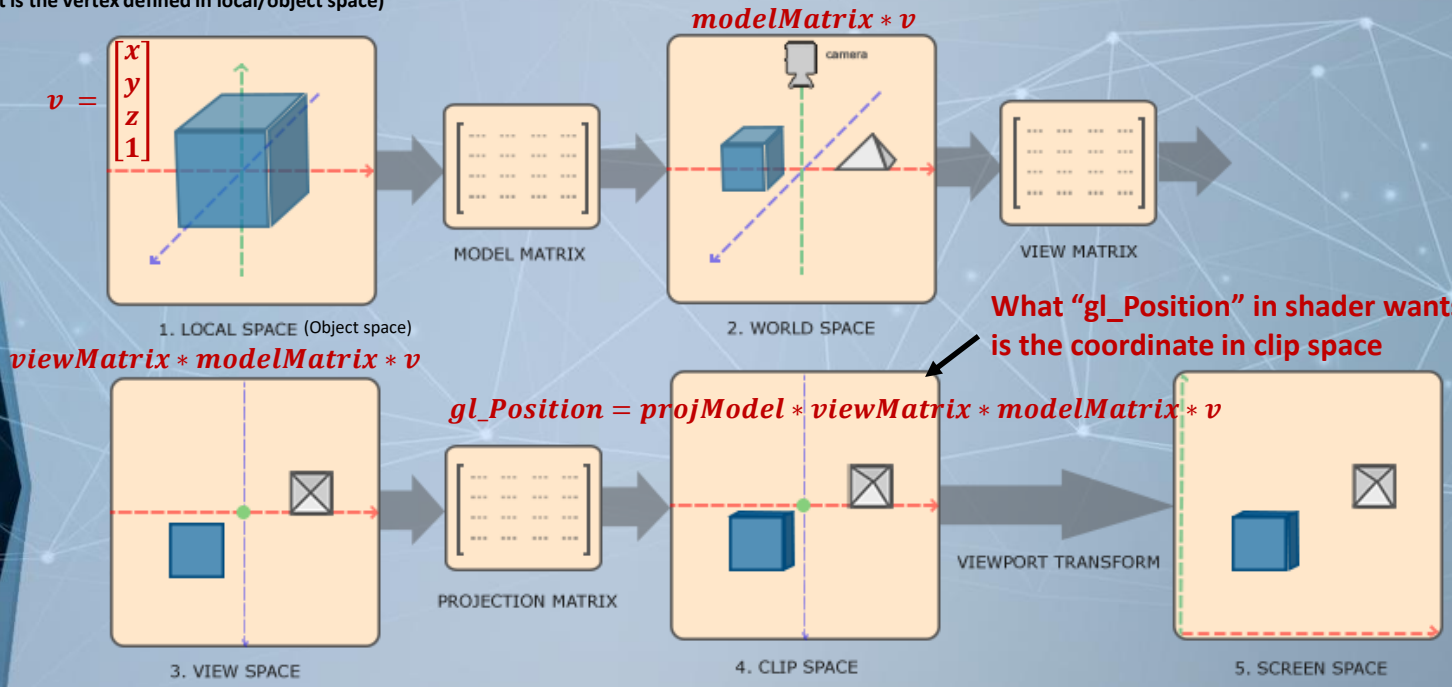
Orthographic Projection Matrix (View to Clip Space)

- How to create an orthographic projection matrix?
  - ”setOrtho()” in cuon-matrix.js
- Matrix4.setOrtho(xmin, xmax, ymin, ymax, znear, zfar)
  - xmin, xmax: boundaries along the x-axis of the clip space (define width)
  - ymin, ymax: boundaries along the y-axis of the clip space (define height)
  - znear, zfar: boundaries along the z-axis of the clip space



Coordinate System (Space)

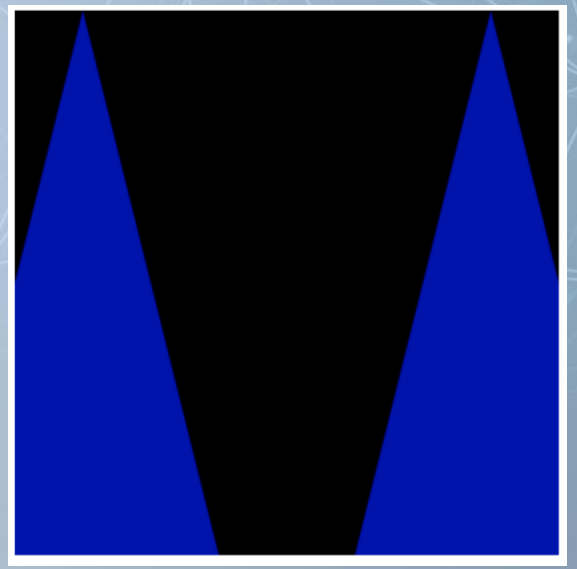
v is what the coordinate you put in the VBO and pass to shader
(that is the vertex defined in local/object space)



https://learnopengl.com/Getting-started/Coordinate-Systems
http://www.codinglabs.net/article_world_view_projection_matrix.aspx

Example (Ex05-3)

- Files:
  - Index.html
  - WebGL.js
  - Cuon-matrix.js



Example (Ex05-3)

Shader

```
var VSHADER_SOURCE = `
        attribute vec4 a_Position;
        attribute vec4 a_Color;
        uniform mat4 u_ModelMatrix;
        uniform mat4 u_ViewMatrix;
        uniform mat4 u_ProjMatrix;
        varying vec4 v_Color;
        void main(){
            gl_Position = u_ProjMatrix*u_ViewMatrix * u_ModelMatrix * a_Position;
            v_Color = a_Color;
        }
    `;

var FSHADER_SOURCE = `
    precision mediump float;
    varying vec4 v_Color;
    void main(){
        gl_FragColor = v_Color;
    }
`;
```

We can see the blue triangles
even if we set the third argument of setLookAt() to 2

main() in WebGL.js

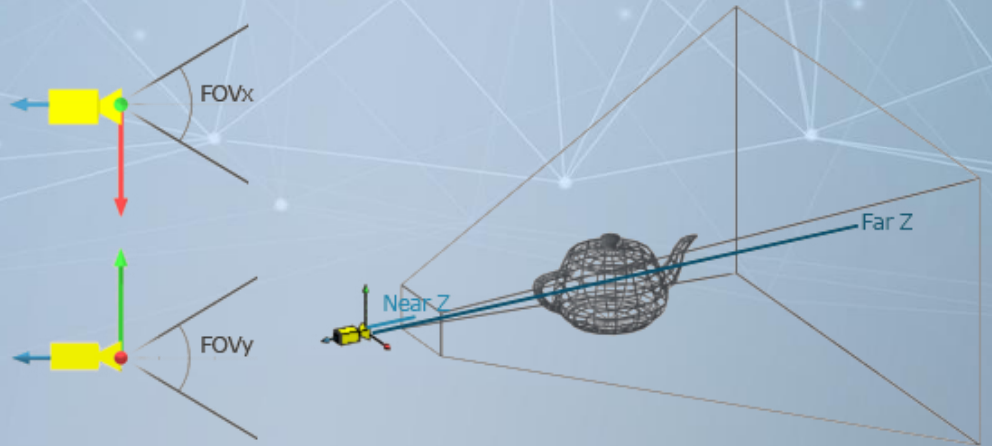
```
function main(){
    //////Get the canvas context
    var canvas = document.getElementById('webgl');
    var gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);

    gl.useProgram(program);

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);

    var n = initVertexBuffers(gl, program);

    var u_ModelMatrix = gl.getUniformLocation(program, 'u_ModelMatrix');
    var u_ViewMatrix = gl.getUniformLocation(program, 'u_ViewMatrix');
    var u_ProjMatrix = gl.getUniformLocation(program, 'u_ProjMatrix');


    var modelMatrix = new Matrix4();
    var viewMatrix = new Matrix4();
    var projMatrix = new Matrix4();

    viewMatrix.setLookAt(0, 0, 2, 0, 0, -100, 0, 1, 0);
    projMatrix.setOrtho(-1, 1, -1, 1, -10, 10);

    gl.uniformMatrix4fv(u_ViewMatrix, false, viewMatrix.elements);
    gl.uniformMatrix4fv(u_ProjMatrix, false, projMatrix.elements);

    modelMatrix.setTranslate(0.75, 0, 0);
    gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
    gl.drawArrays(gl.TRIANGLES, 0, n);

    modelMatrix.setTranslate(-0.75, 0, 0);
    gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
    gl.drawArrays(gl.TRIANGLES, 0, n);
}
```

Let’s try and think (5mins)

- What happen if you modify setOrtho() to
  - setOrtho(0, 1, -1, 1, -10, 10);
  - setOrtho(-10, 10, -10, 10, -10, 10);
  - setLookAt(5, 5, 5, 0, 0, 0, 0, 1, 0) and setOrtho(-5, 5, -5, 5, -20, 20);
  - Try any argument you want, make sure you understand why what you see happens

Perspective Projection Matrix (View to Clip Space)

- **Perspective:** objects which are far away are smaller than those nearby
- To do the perspective projection, you should define the fov of y axis (field of view), aspect ratio (width/height), z-near and z-far



Perspective Projection Matrix (View to Clip Space)

Vertices in clip space

$$v^C = M^P * v^V$$

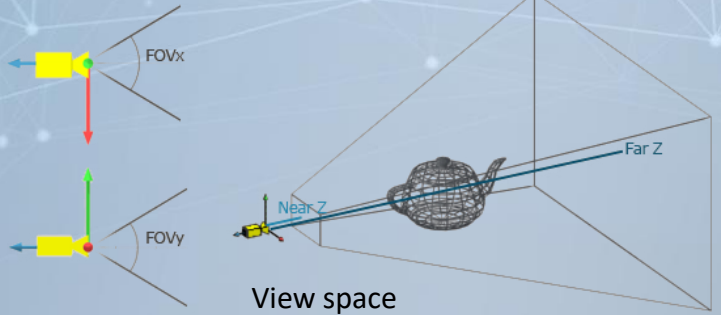
Projection matrix

Vertices in view space

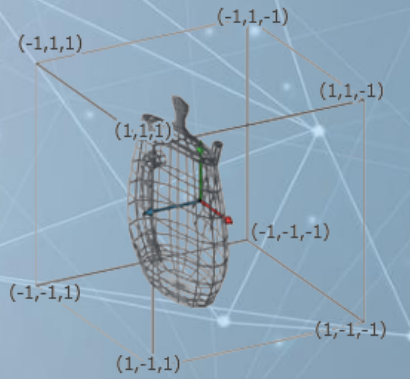
- fov-x can be derived by fov-y and aspect ratio

Perspective projection matrix

$$\begin{bmatrix} \tan^{-1}(\frac{FOVx}{2}) & 0 & 0 & 0 \\ 0 & \tan^{-1}(\frac{FOVy}{2}) & 0 & 0 \\ 0 & 0 & -\frac{Z_{far}+Z_{near}}{Z_{far}-Z_{near}} & -\frac{2\,(Z_{near}\,Z_{far})}{Z_{far}-Z_{near}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



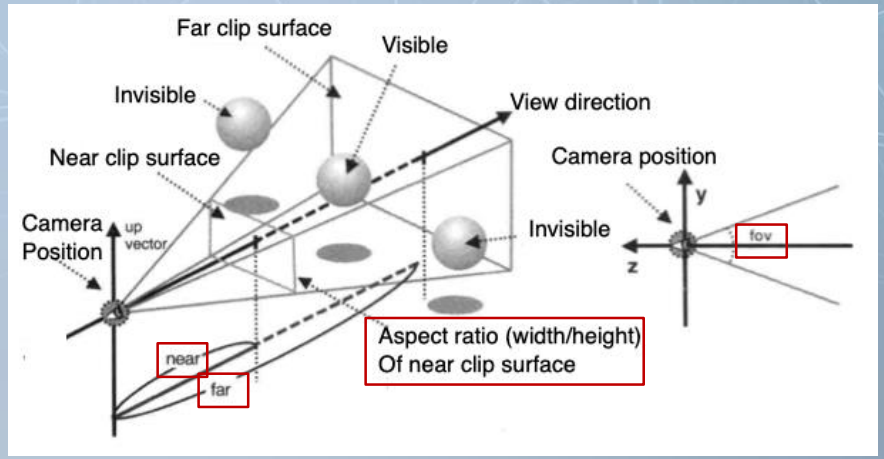
View space



Clip space

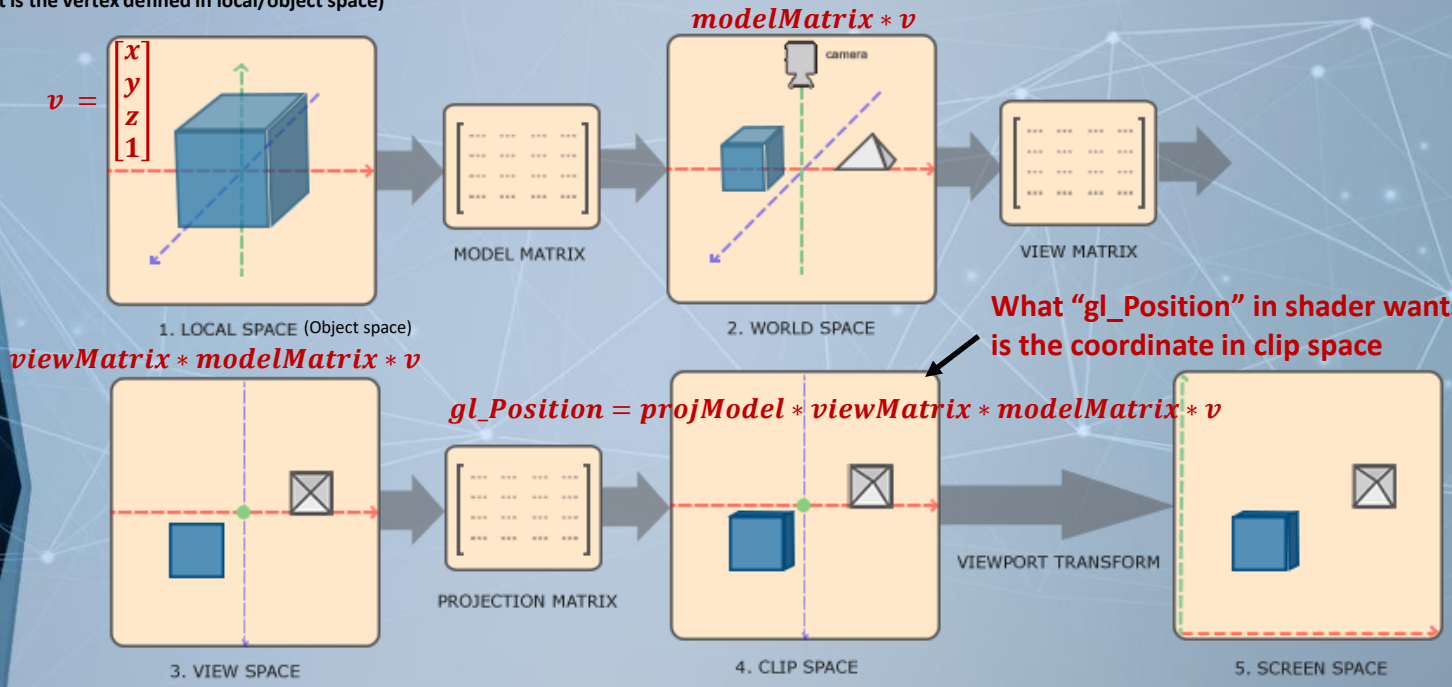
Perspective Projection Matrix (View to Clip Space)

- How to create an orthographic projection matrix?
  - ”setPerspective()” in cuon-matrix.js
- Matrix4. setPerspective(**fov, aspect, near, far**)
  - fov: angle of field of view in degree along the y-axis
  - aspect: ratio of width and height (width/height) of the near clipping surface
  - znear, zfar: boundaries along the z-axis of the clip space



Coordinate System (Space)

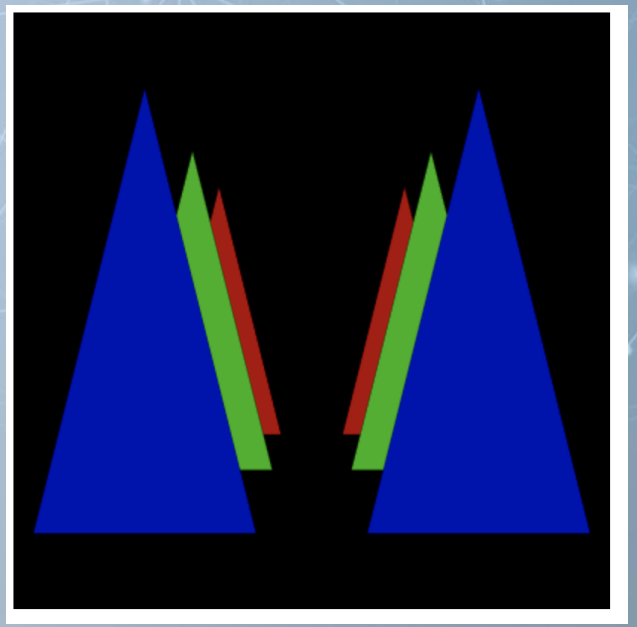
v is what the coordinate you put in the VBO and pass to shader
(that is the vertex defined in local/object space)



https://learnopengl.com/Getting-started/Coordinate-Systems
http://www.codinglabs.net/article_world_view_projection_matrix.aspx

Example (Ex05-4)

- Files:
  - Index.html
  - WebGL.js
  - Cuon-matrix.js



Example (Ex05-4)

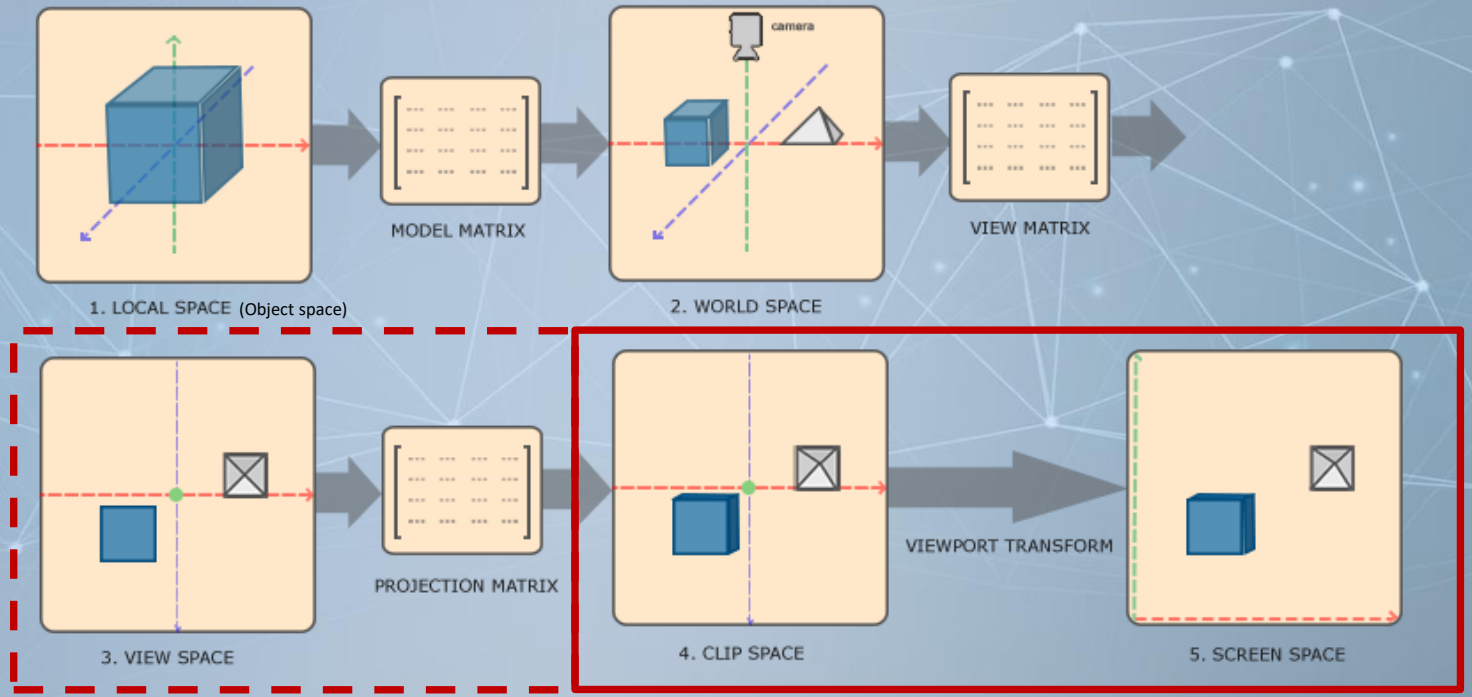
main() in WebGL.js

```
function main(){
    //////Get the canvas context
    var canvas = document.getElementById('webgl');
    var 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);

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);

    var n = initVertexBuffers(gl, program);

    var u_ModelMatrix = gl.getUniformLocation(program, 'u_ModelMatrix');
    var u_ViewMatrix = gl.getUniformLocation(program, 'u_ViewMatrix');
    var u_ProjMatrix = gl.getUniformLocation(program, 'u_ProjMatrix');


    var modelMatrix = new Matrix4();
    var viewMatrix = new Matrix4();
    var projMatrix = new Matrix4();

    viewMatrix.setLookAt(0, 0, 5, 0, 0, -100, 0, 1, 0);
    projMatrix.setPerspective(30, canvas.width/canvas.height, 1, 100);


    gl.uniformMatrix4fv(u_ViewMatrix, false, viewMatrix.elements);
    gl.uniformMatrix4fv(u_ProjMatrix, false, projMatrix.elements);

    modelMatrix.setTranslate(0.75, 0, 0);
    gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
    gl.drawArrays(gl.TRIANGLES, 0, n);

    modelMatrix.setTranslate(-0.75, 0, 0);
    gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
    gl.drawArrays(gl.TRIANGLES, 0, n);
}
```

Let’s try and think (5mins)

- Now, the depth matters
  - Try to modify setLookAt() to setLookAt(0,0,2, 0,0,-100,0,1,0)
- Go back to try Ex05-3
  - If you modify the third argument of setLookAt(), does what you see change?
- change fov (the first argument) in setPerspective()?
- change aspect (the second argument) in setPerspective() to 2 or 0.5?

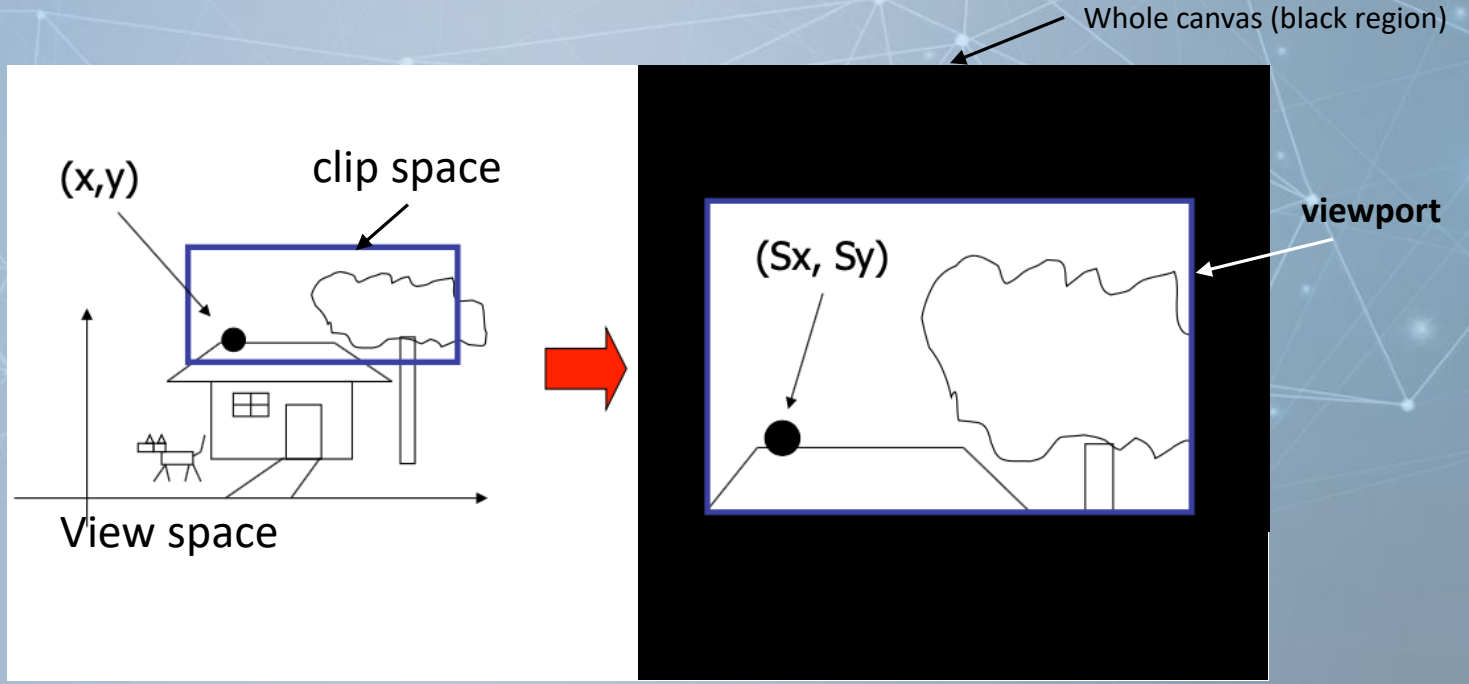
Viewport Transformation (Clip to Screen Space)



https://learnopengl.com/Getting-started/Coordinate-Systems
http://www.codinglabs.net/article_world_view_projection_matrix.aspx

Viewport Transformation (Clip to Screen Space)

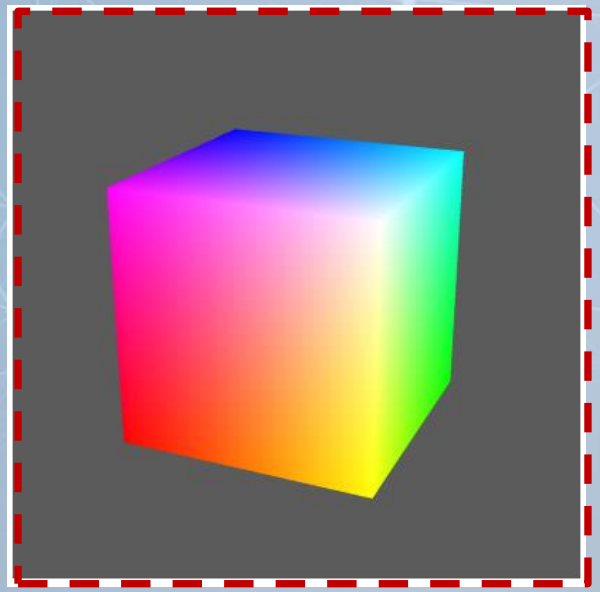
- You can map clip space (x-y plane) to arbitrary sub-region of your canvas
- How to calculate (sx, sy) from (x,y) (ignore z in clip space here)



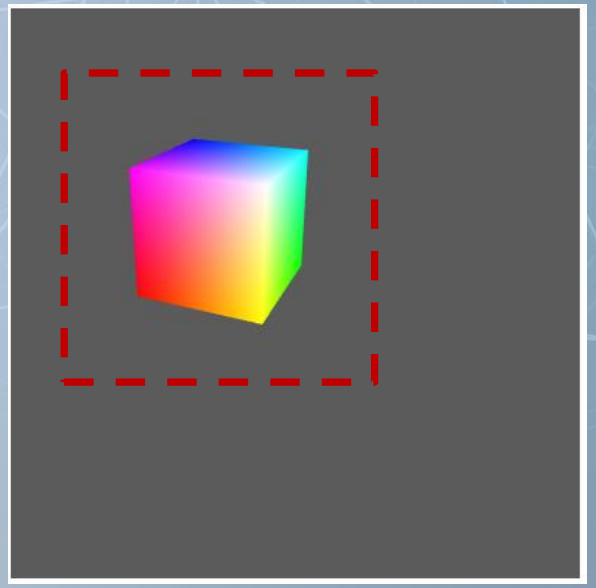
Viewport Transformation (Clip to Screen Space)

- Just set different viewports

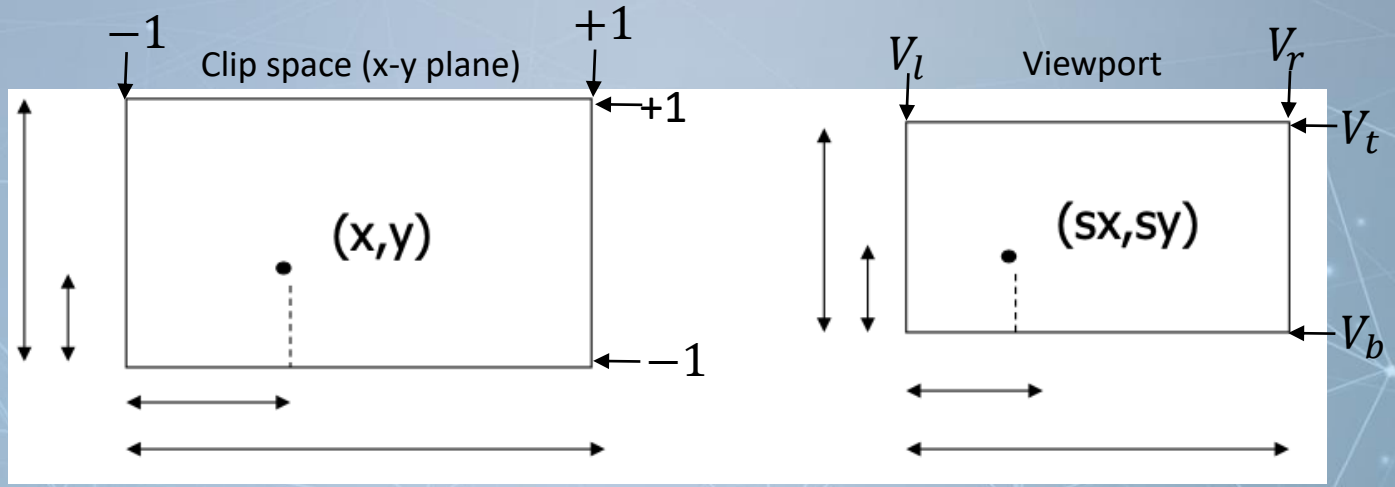
Viewport: whole canvas



Viewport: a portion of the canvas



Viewport Transformation (Clip to Screen Space)

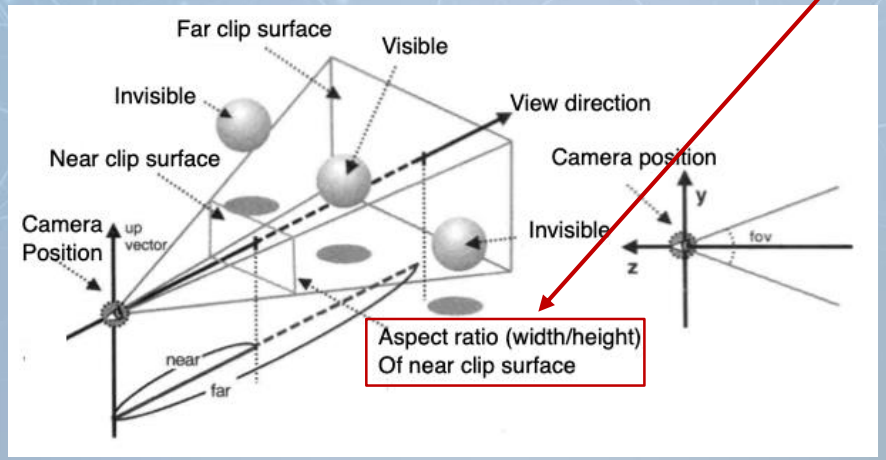


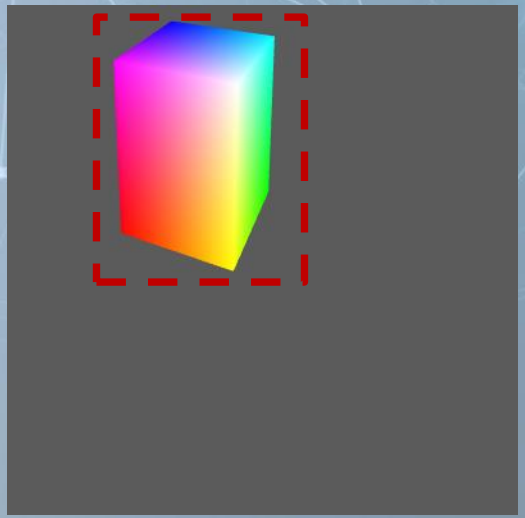
- $s_x = x * \frac{(V_r - V_l)}{1-(-1)} + \frac{(V_r + V_l)}{1-(-1)}$
- $s_y = y * \frac{(V_t - V_b)}{1-(-1)} + \frac{(V_t + V_b)}{1-(-1)}$

Viewport Transformation (Clip to Screen Space)

- Question: when a distortion happens?
  - **Clip space x-y plane in view space** and viewport have different aspect ratio
- What is "**clip space x-y plane in view space** "?

Perspective projection example

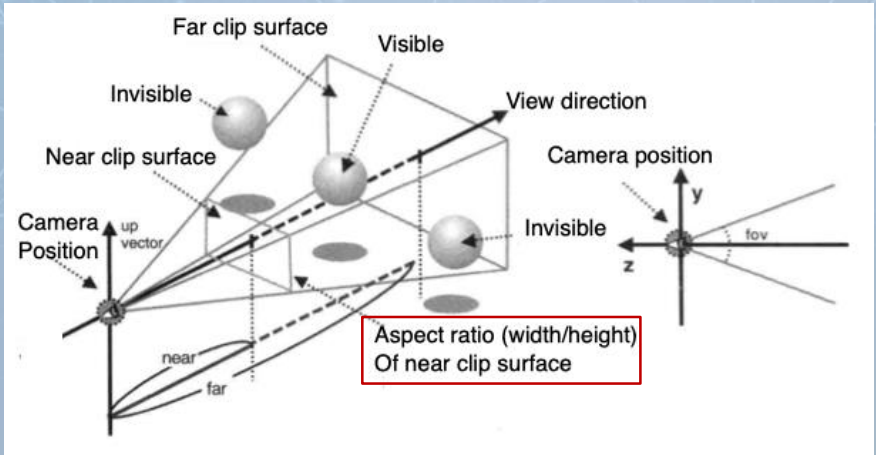


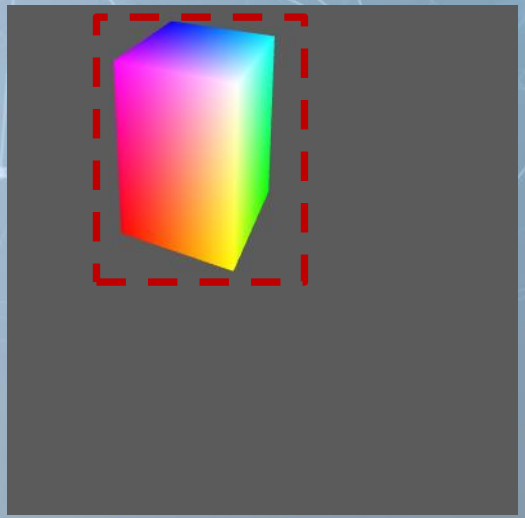


Viewport Transformation (Clip to Screen Space)

- So, if the ratio of the “near clip surface” and “viewport” are different, you will see a distortion.
- Usually (not always), we set a 1:1 aspect ration for the near clip surface and use the whole canvas as the viewport
  - In this case, if the aspect ratio of you canvas is not 1:1, you will see a distortion

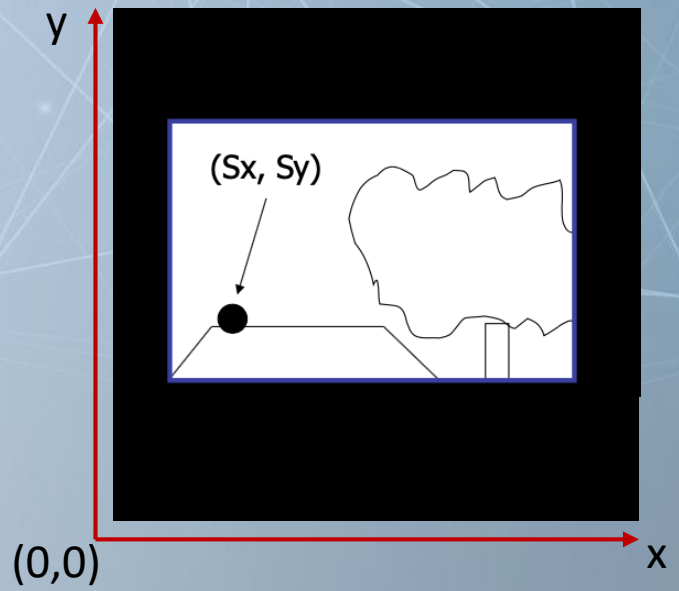
Perspective projection example





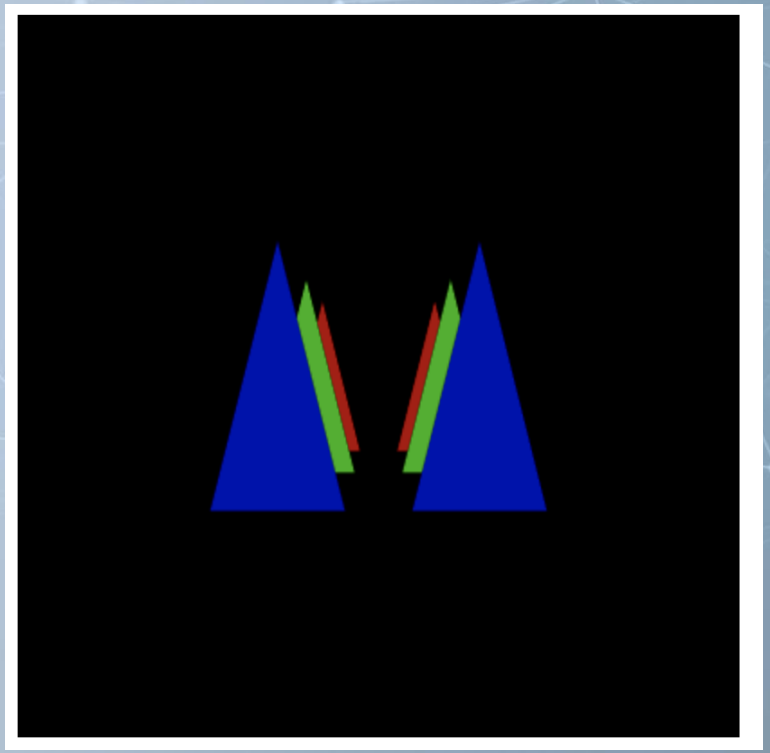
gl.viewport(x, y, width, height)

- https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/viewport
- Unit of x, y, width, height: pixels of the canvas
- Left bottom corner is (0,0)



Example (Ex05-5)

- Files:
  - Index.html
  - WebGL.js
  - Cuon-matrix.js



Example (Ex05-5)

main() in WebGL.js

You see smaller triangles just because we map the clip space to a smaller screen region

```
function main(){
    //////Get the canvas context
    var canvas = document.getElementById('webgl');
    var 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);

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);

    gl.viewport(100, 100, canvas.width/2, canvas.height/2);

    var n = initVertexBuffers(gl, program);

    var u_ModelMatrix = gl.getUniformLocation(program, 'u_ModelMatrix');
    var u_ViewMatrix = gl.getUniformLocation(program, 'u_ViewMatrix');
    var u_ProjMatrix = gl.getUniformLocation(program, 'u_ProjMatrix');


    var modelMatrix = new Matrix4();
    var viewMatrix = new Matrix4();
    var projMatrix = new Matrix4();

    viewMatrix.setLookAt(0, 0, 5, 0, 0, -100, 0, 1, 0);
    projMatrix.setPerspective(30, canvas.width/canvas.height, 1, 100);

    gl.uniformMatrix4fv(u_ViewMatrix, false, viewMatrix.elements);
    gl.uniformMatrix4fv(u_ProjMatrix, false, projMatrix.elements);

    modelMatrix.setTranslate(0.75, 0, 0);
    gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
    gl.drawArrays(gl.TRIANGLES, 0, n);

    modelMatrix.setTranslate(-0.75, 0, 0);
    gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
    gl.drawArrays(gl.TRIANGLES, 0, n);
}
```

Let’s try and think (5mins)

- Try to
  - Change the canvas height to 200 (in index.html)
  - and viewport() to gl.viewport(0, 0, canvas.width, canvas.height);
  - and setPerspective() to projMatrix.setPerspective(30, **1**, 1, 100);
  - Then, run it

- Does the result match your understanding?

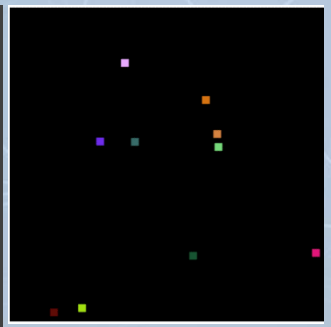
- From the original code of Ex05-5, change setLookAt() to setLookAt(0, 0, -6, 0, 0, 100, 0, 1, 0);
  - Looks like something wrong?

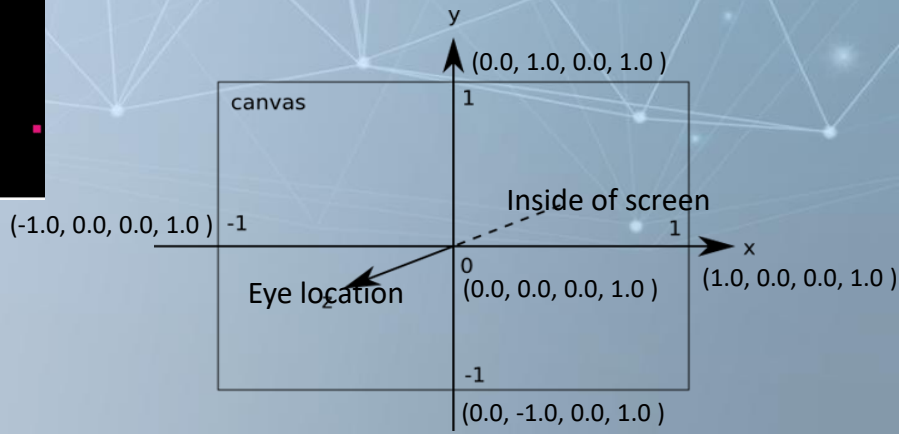
Recall the 2D Drawing

- Do you remember this one when we just draw in 2D in the very beginning (I said the canvas x-y range from -1 to +1, I skip a lot of details). Now, we can review it again.
- gl_position is defined in clip space and we use whole canvas as the viewport (default)

Ex1-3

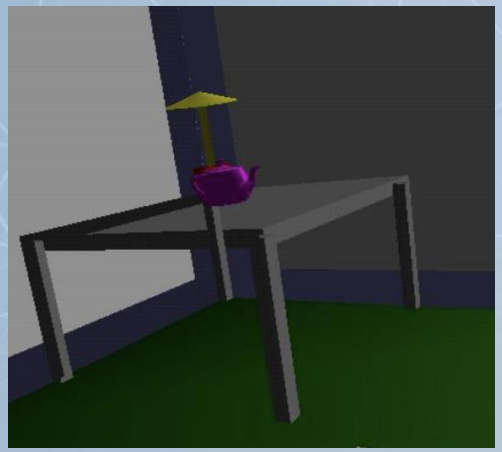
```
var VSHADER_SOURCE = `
      uniform vec4 u_Position;
      void main(){
            gl_Position = u_Position;
            gl_PointSize = 10.0;
      }
      `;

var FSHADER_SOURCE = `
      precision mediump float;
      uniform vec4 u_FragColor;
      void main(){
            gl_FragColor = u_FragColor;
      }
      `;
```



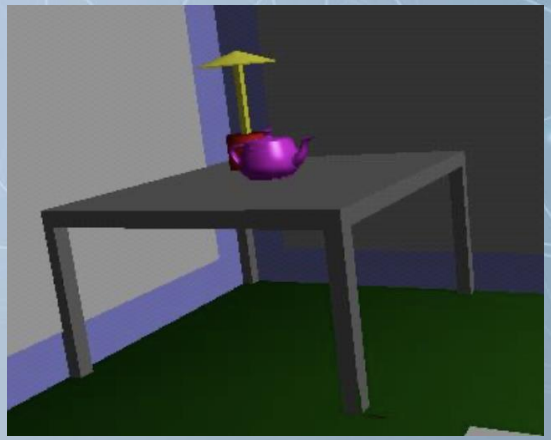


Visibility

- A correct rendering requires correct visibility calculation
- Another visibility example
  - Of course, we want the correct one



Wrong visibility



Correct visibility

Visibility

- Where does WebGL deal with the visibility problem in the rendering pipeline?

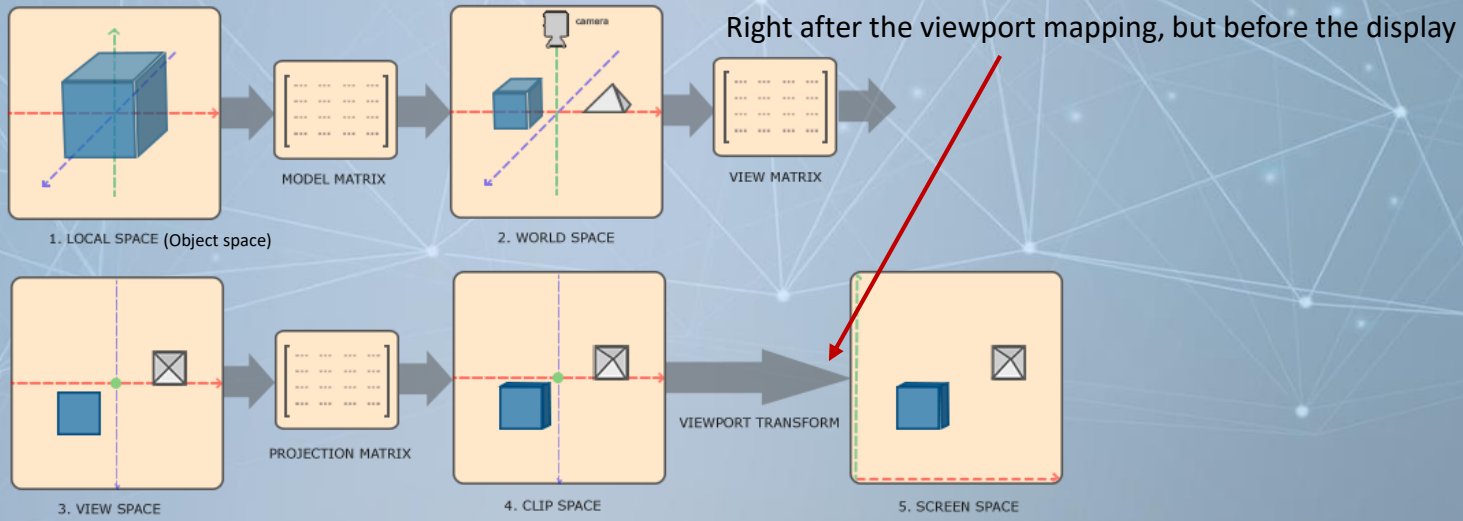
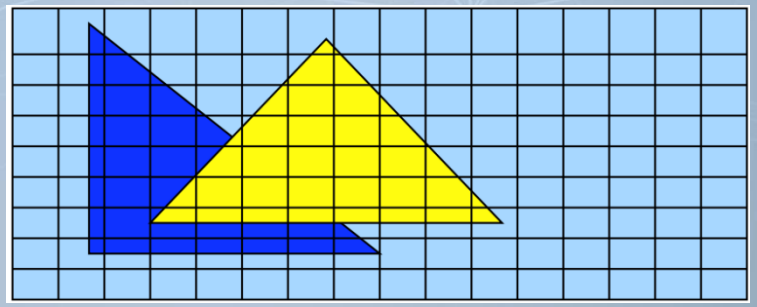


Image Space Approach for Visibility

- Determine which of the n objects is visible to each pixel on the image plane

```
for(each pixel in the image){
        determine the object closest to the pixel
        draw the pixel using the object's color
}
```



Depth Buffer (Z-buffer)

- Method used in most graphics hardware
  - Z-buffer algorithm
  - (There are still other approaches/algorithms to determine the visibility. Z-buffer algorithm is just one of them)

- Basic idea
  - Rasterize every input polygon
  - For every pixel in the polygon interior, calculate corresponding Z value
  - Choose the color of the polygon whose z value closest to the eye to paint the pixel

Z-buffer Algorithm

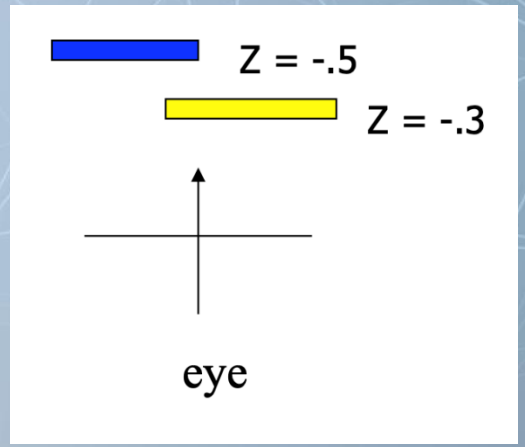
- Step 1: initialize the depth buffer with very large number

-999	-999	-999	-999
-999	-999	-999	-999
-999	-999	-999	-999
-999	-999	-999	-999

Z-buffer Algorithm

- Step 2: WebGL may draw the blue polygon first (we must have depth (z) information of all object in clip space)

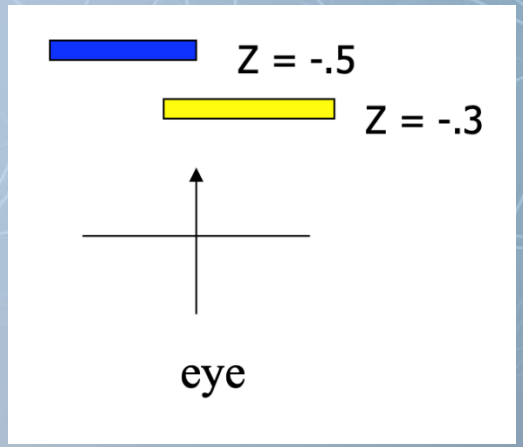
-999	-999	-999	-999
-999	-999	-999	-999
-.5	-.5	-999	-999
-.5	-.5	-999	-999



Z-buffer Algorithm

- Step 2: Then, draw the yellow polygon

-999	-999	-999	-999
-999	-.3	-.3	-999
-.5	-.3	-.3	-999
-.5	-.5	-999	-999

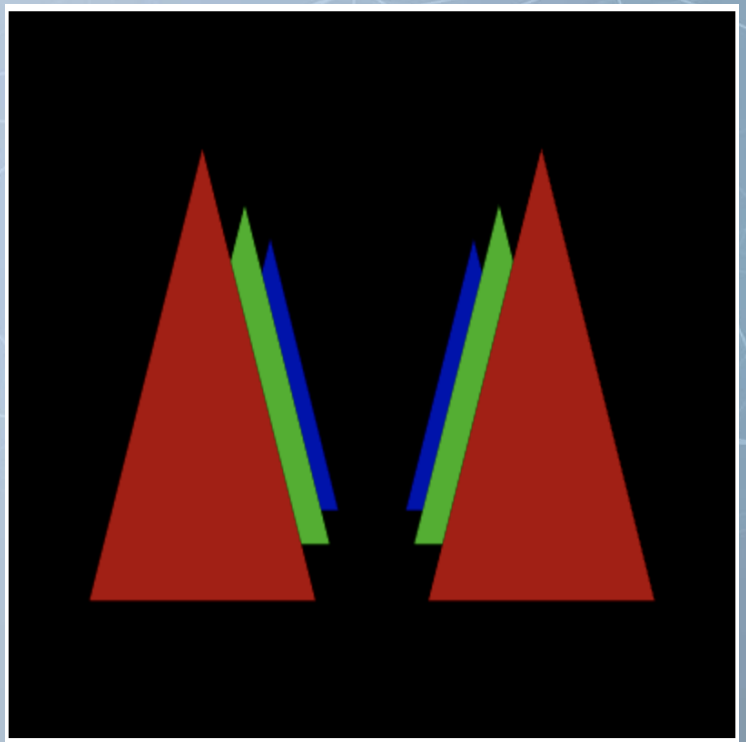


Depth Buffer in WebGL

- It’s simple. Just call
  - **gl.enable(gl.DEPTH_TEST)**
- And, before you draw any new frame, call
  - **gl.clear(gl.DEPTH_BUFFER_BIT)**
  - To clear the depth buffer for new frame

Example (Ex05-6)

- Look into the scene from the other size with correct visibility
- Files:
  - Index.html
  - WebGL.js
  - Cuon-matrix.js



Example (Ex05-6)

- main() in WebGL.js
- We change the setLookAt() to look into the scene from the other side
- You have to call gl.enable(DEPTH_TEST) for once in your program
- To ensure the correct visibility, you have to call gl.clear() to clear depth buffer before you draw a new frame

```
function main(){
    //////Get the canvas context
    var canvas = document.getElementById('webgl');
    var 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);

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST)
    gl.clear(gl.DEPTH_BUFFER_BIT | gl.COLOR_BUFFER_BIT)
    //You must clear the depth buffer (call gl.lear()) for every new frame
    //, and ususally color buffer, too


    gl.viewport(0, 0, canvas.width, canvas.height);

    var n = initVertexBuffers(gl, program);

    var u_ModelMatrix = gl.getUniformLocation(program, 'u_ModelMatrix');
    var u_ViewMatrix = gl.getUniformLocation(program, 'u_ViewMatrix');
    var u_ProjMatrix = gl.getUniformLocation(program, 'u_ProjMatrix');


    var modelMatrix = new Matrix4();
    var viewMatrix = new Matrix4();
    var projMatrix = new Matrix4();

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

    gl.uniformMatrix4fv(u_ViewMatrix, false, viewMatrix.elements);
    gl.uniformMatrix4fv(u_ProjMatrix, false, projMatrix.elements);

    modelMatrix.setTranslate(0.75, 0, 0);
    gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
    gl.drawArrays(gl.TRIANGLES, 0, n);

    modelMatrix.setTranslate(-0.75, 0, 0);
    gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
    gl.drawArrays(gl.TRIANGLES, 0, n);
}
```

Example (Ex05-6)

- gl.clear()
  - E.g. gl.clear(gl.DEPTH_BUFFER_BIT | gl.COLOR_BUFFER_BIT)
  - https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/clear
- The input argument is a mask to indicate what buffers you want to clear
  - gl.DEPTH_BUFFER_BIT
  - gl.COLOR_BUFFER_BIT
  - gl.STENCIL_BUFFER_BIT
- You can bitwise OR these bits to build a mask you want to clear multiple buffers