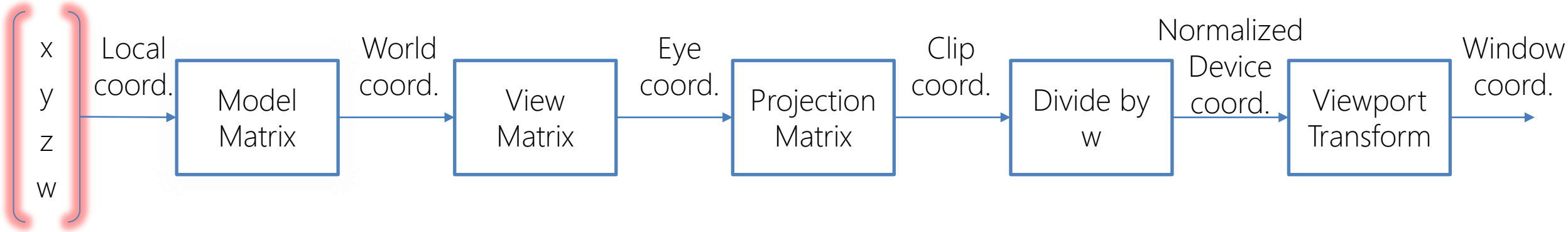
POLITECNICO
MILANO 1863

Transformations and Animations

Computer Graphics 2021

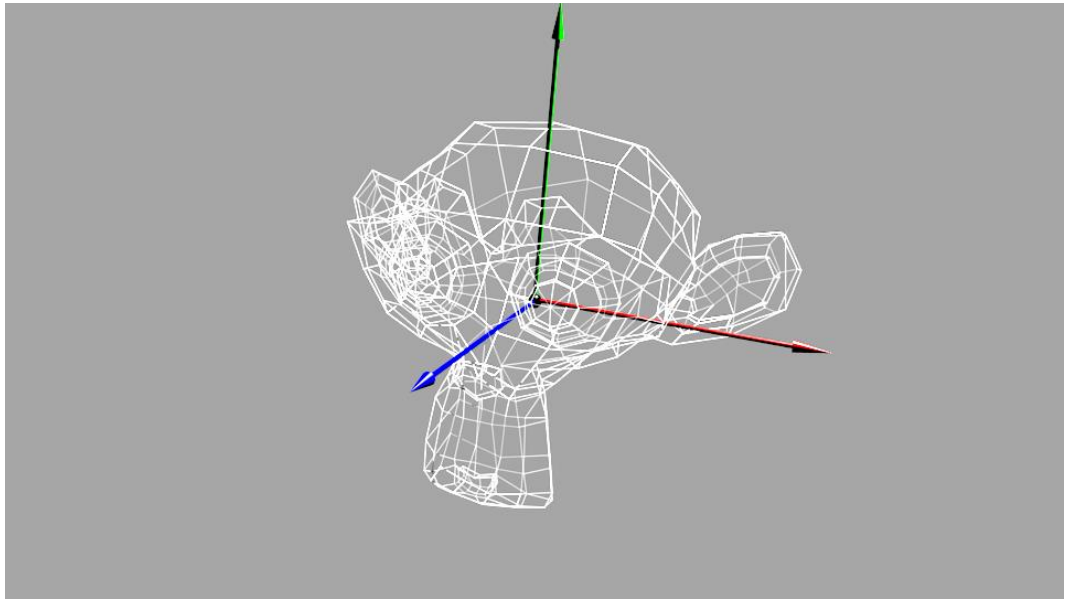
Erica Stella (erica.stella@polimi.it)

Pipeline of OpenGL/WebGL transformations



Local coordinates: x,y,z of the vertices defined relative to the object's center

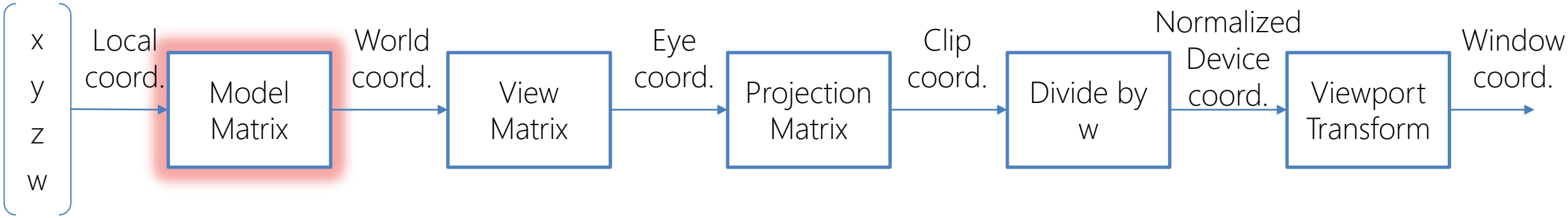
Add w=1 to get **homogeneous coordinates**



Hello Suzanne!

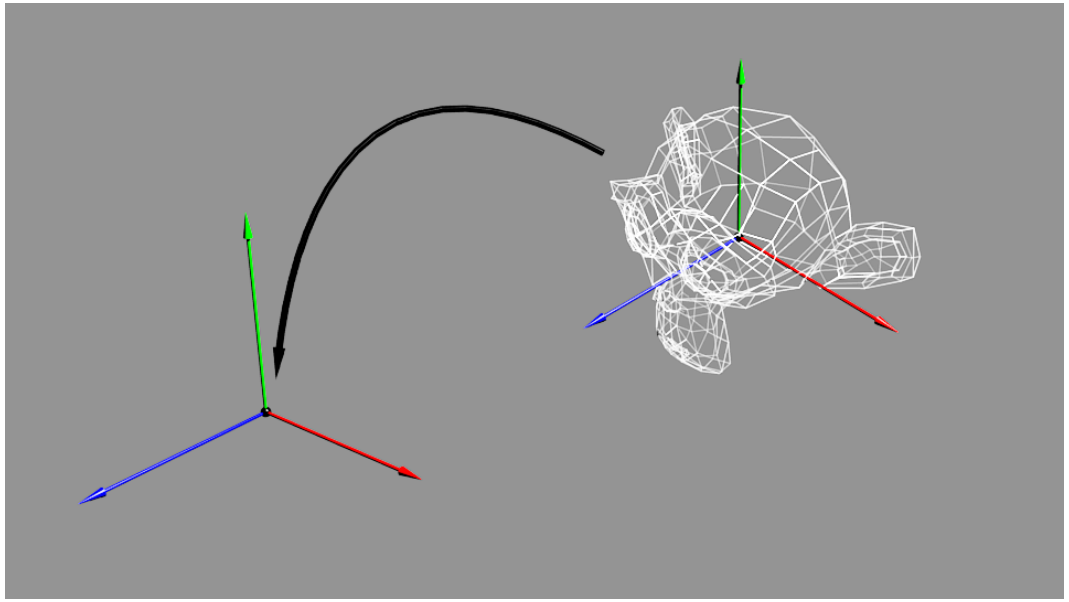
Source: http://www.opengl-tutorial.org

Pipeline of OpenGL/WebGL transformations



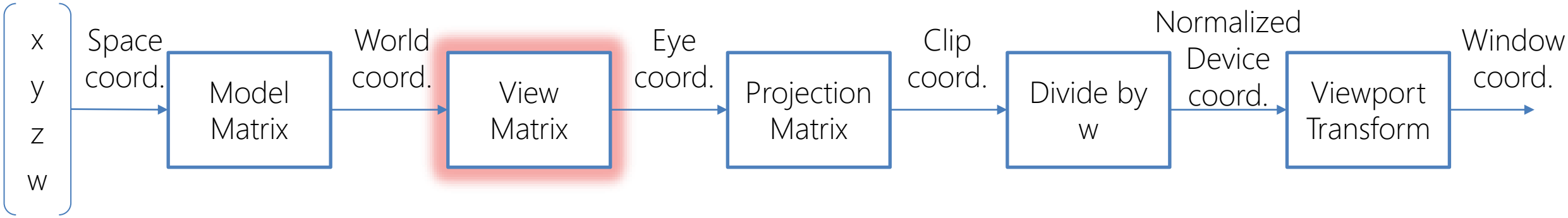
Model/World Matrix:

Moves the model in the world space



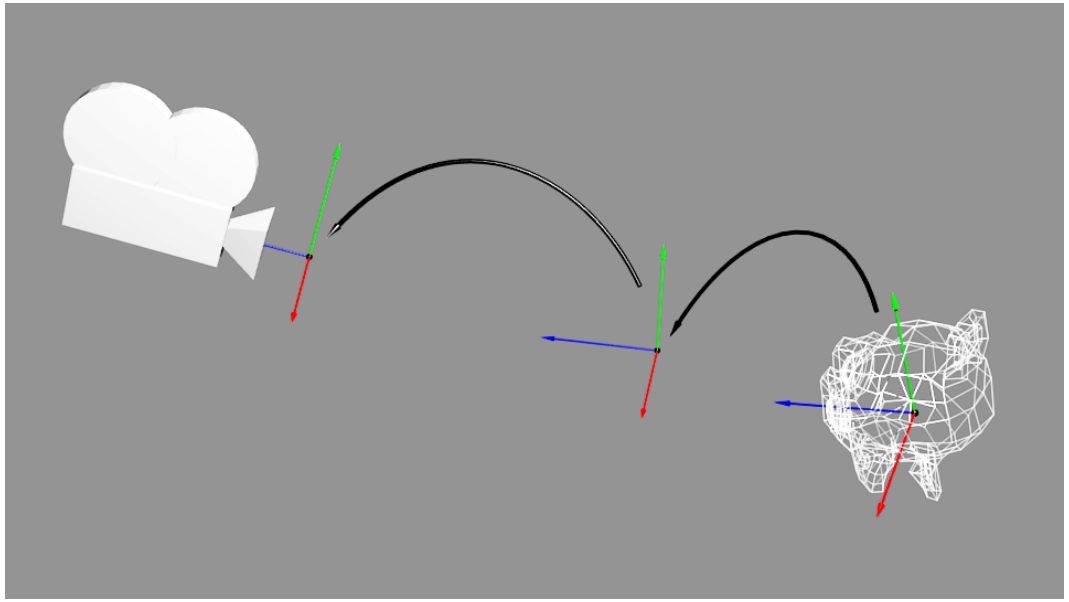
Source: http://www.opengl-tutorial.org

Pipeline of OpenGL/WebGL transformations



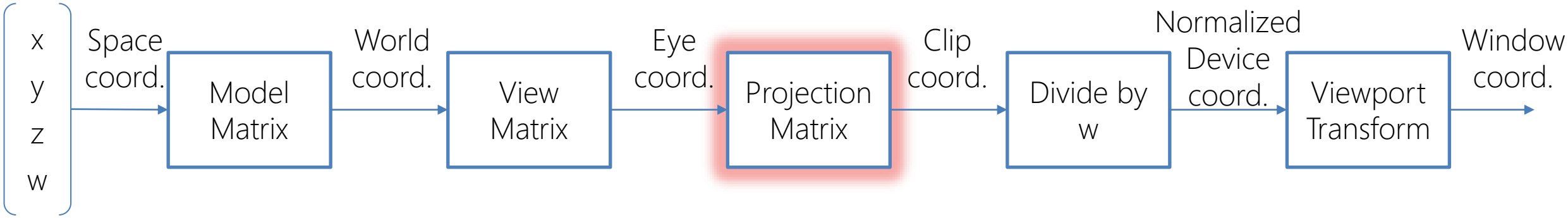
View Matrix:

Sets the position of the camera and moves the model in Camera space (eye coords)



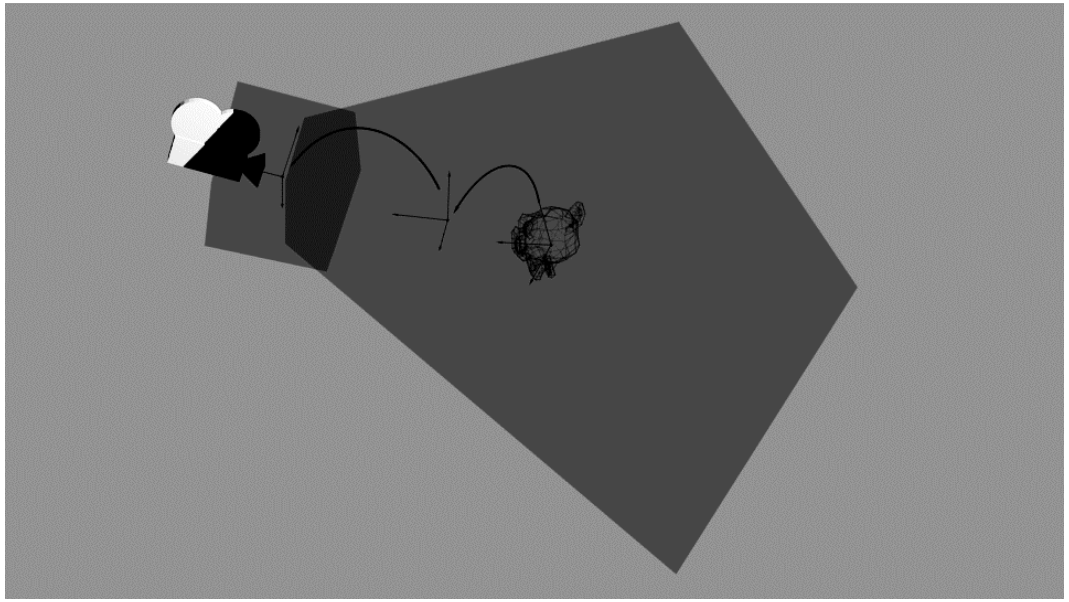
Source: http://www.opengl-tutorial.org

Pipeline of OpenGL/WebGL transformations



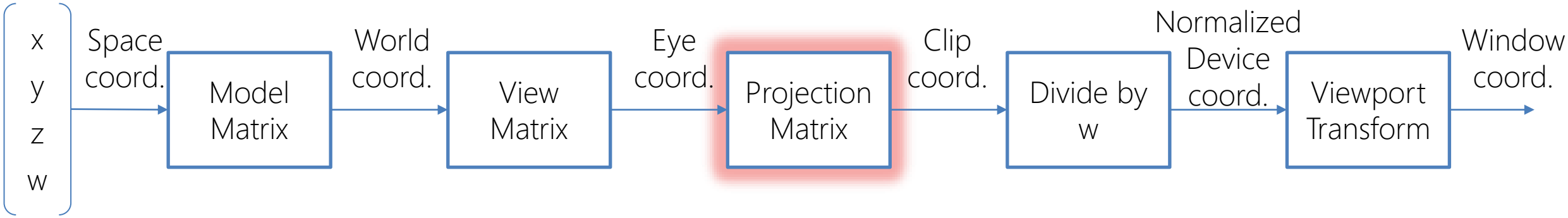
Projection Matrix:

Defines how the scene is perceived by the camera



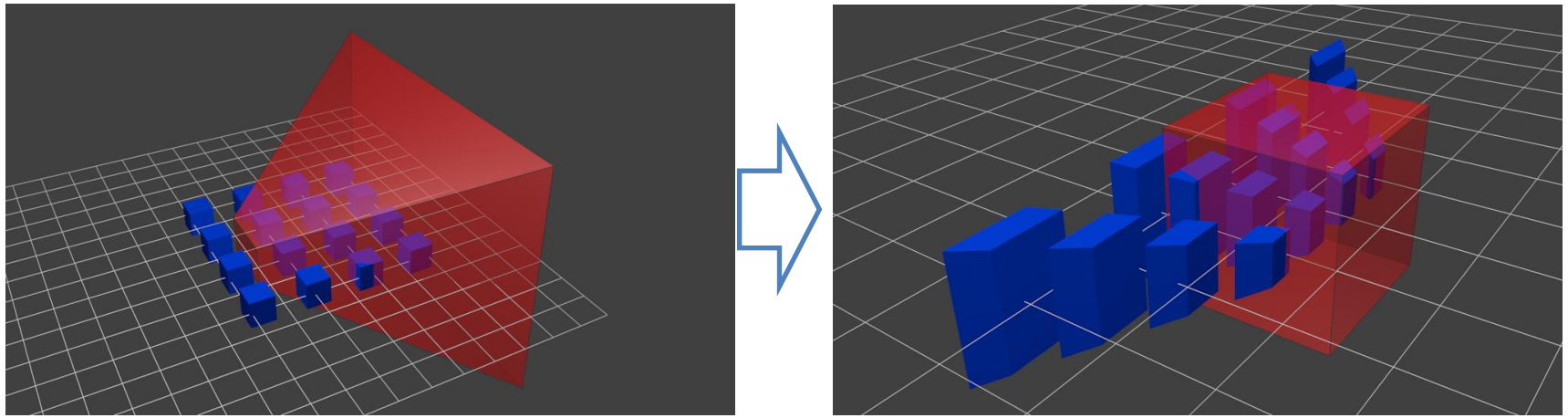
Source: http://www.opengl-tutorial.org

Pipeline of OpenGL/WebGL transformations



Projection Matrix:

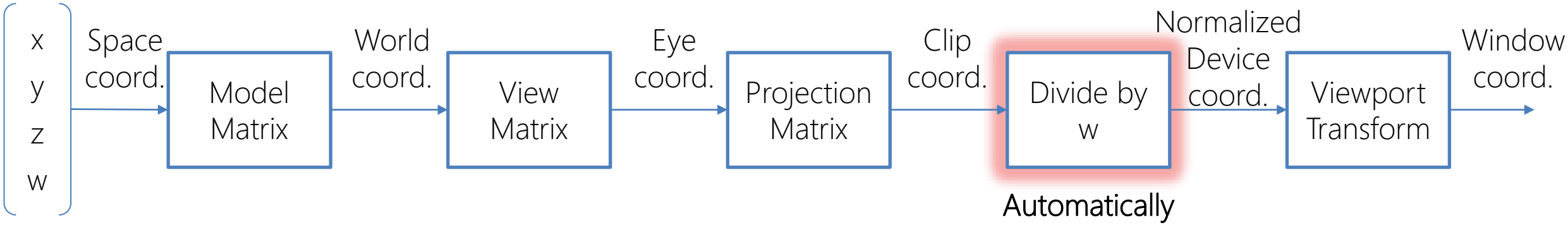
Defines how the scene is perceived by the camera



gl_Position set in the Vertex Shader must be in Clip Space!

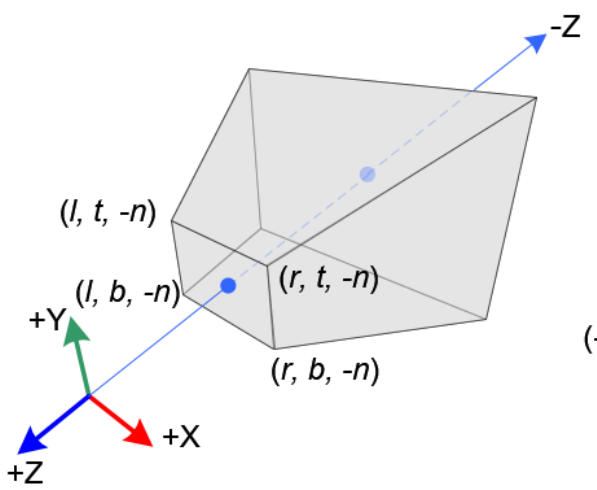
Source: http://www.opengl-tutorial.org

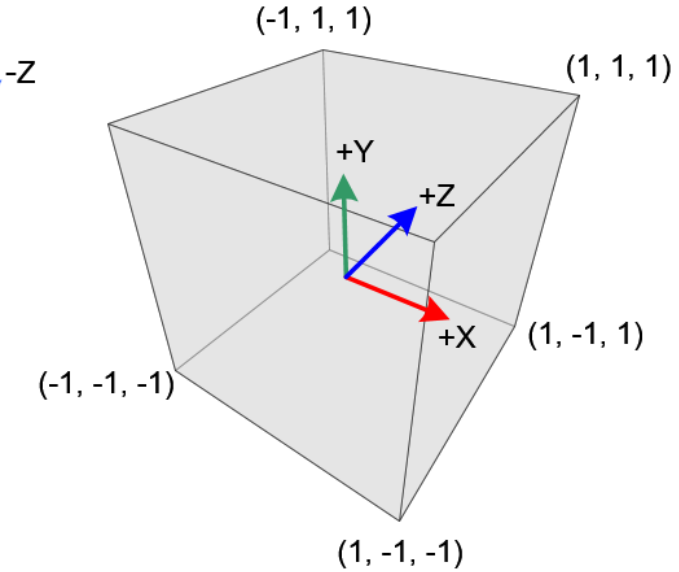
Pipeline of OpenGL/WebGL transformations



Divide by w:

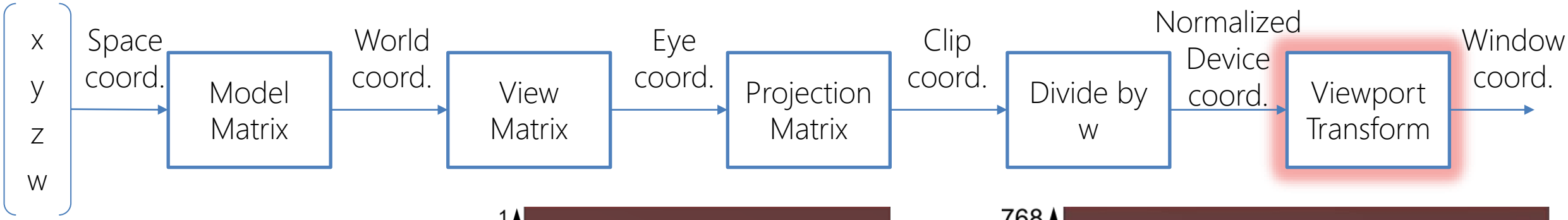
From 4D clip coordinates to 3D Normalized Device Coordinates (NDC):
x ∈ (-1,0,1.0), y ∈ (-1,0,1.0)
z ∈ (-1,0,1.0)





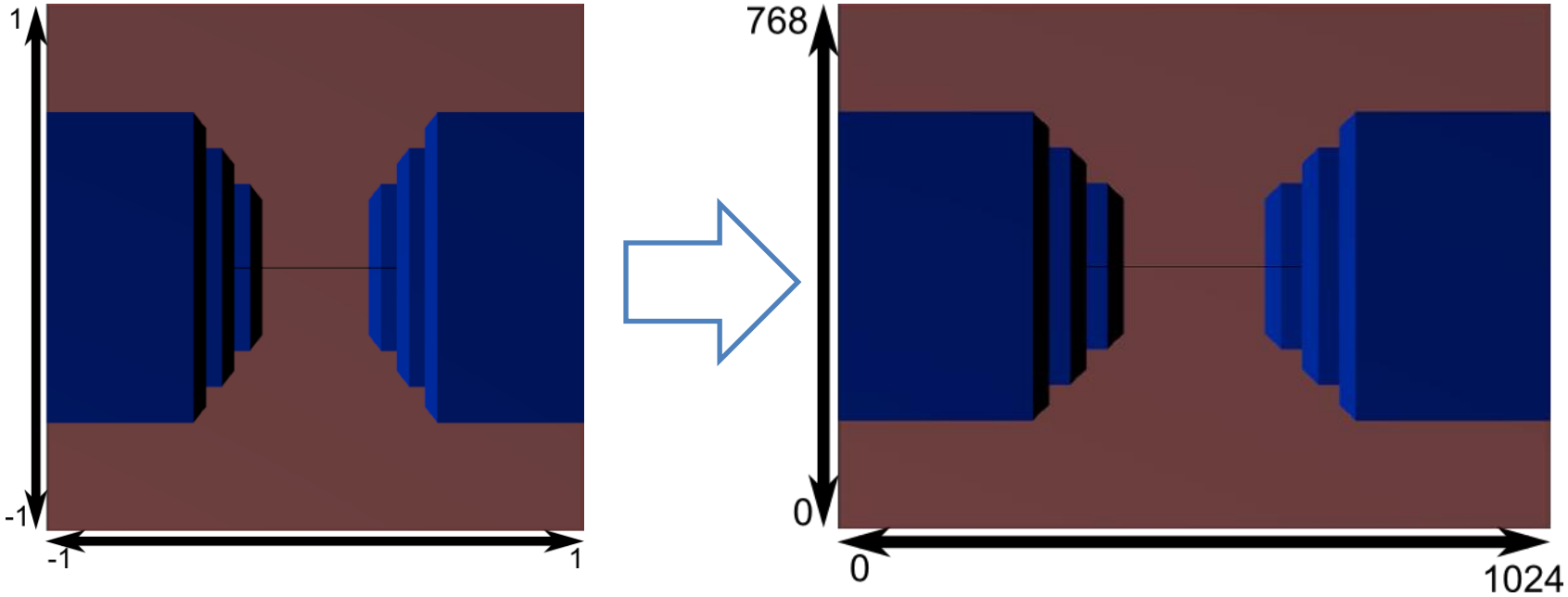
Source: http://www.songho.ca/opengl/gl_projectionmatrix.html

Pipeline of OpenGL/WebGL transformations



Viewport transform:

Transforms NDC to the screen space




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

Source: http://www.opengl-tutorial.org

Transformations

```
#version 300 es

in vec4 a_position;

uniform mat4 matrix;

void main() {

  gl_Position = matrix * a_position;
}
```

```
#version 300 es

precision mediump float;

out vec4 outColor;

void main() {

  outColor = vec4(1.0,0.0,0.0,1.0);
}
```

Matrices in WebGL

- To pass a matrix as a uniform variable to a GLSL program, we need to use the function **uniformMatrix[234]fv(location, transpose, value)**
  - The number chosen between [234] stands for the dimension of the matrix (2x2, 3x3, 4x4)
  - The values are assumed to be float (f) and are passed in a single array (v)
  - The location is obtained with the usual **`getUniformLocation(…)`**
  - Transpose MUST be gl.FALSE
- To create and manipulate matrices, we will use the functions provided by our *utils.js* script (you will find it in the exercise folders)

Matrices in WebGL

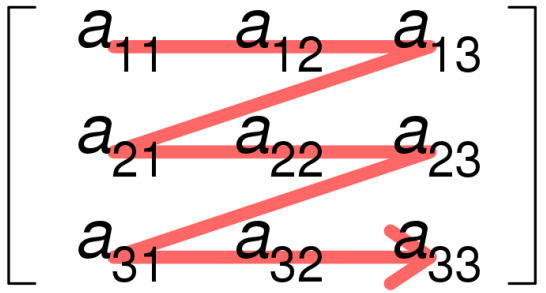
- Matrices in WebGL are specified in column-major order
- Our *utils* script specifies them in row-major order

```
MakeTranslateMatrix:
function(dx,dy,dz) {
  return [1,0,0,dx,
          0,1,0,dy,
          0,0,1,dz,
          0,0,0,1];
}
```

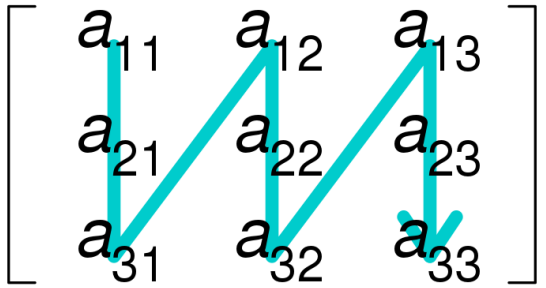
=

```
MakeTranslateMatrix: function(dx,dy,dz) {
  return [1,0,0,dx,0,1,0,dy,0,0,1,dz,0,0,0,1];
}
```

Row-major order



Column-major order



Source: https://en.wikipedia.org/wiki/Row-_and_column-major_order

Matrices in WebGL

- If we pass a matrix like this to WebGL, it will interpret it in the WRONG way!

```
gl.uniformMatrix4fv(matrixLocation, gl.FALSE, MakeTranslateMatrix(0.1,0.2,0.3));
```

```
[1,0,0,0,
 0,1,0,0,
 0,0,1,0,
 dx,dy,dz,1];
```

WRONG!!

- We need to transpose the matrix before passing it

```
var matrix = MakeTranslateMatrix(0.1,0.2,0.3);
gl.uniformMatrix4fv(matrixLocation, gl.FALSE, utils.transposeMatrix(matrix));
```

Transformations - example

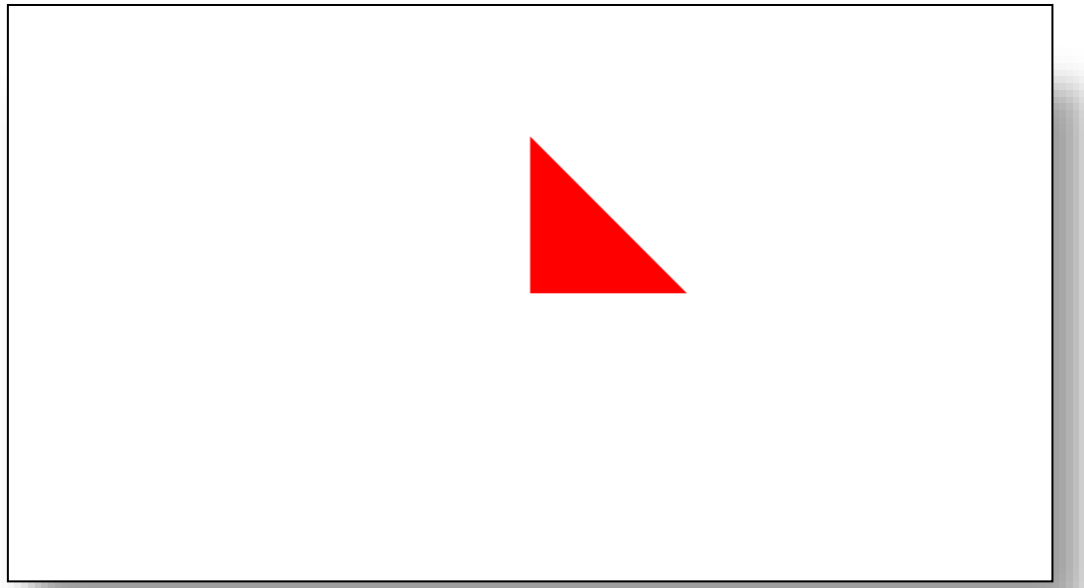
```
[...]
  // look up where the vertex data needs to go.
  var positionAttributeLocation = gl.getAttribLocation(program, "a_position");
  // look up where the matrix needs to go.
  var matrixLocation = gl.getUniformLocation(program, "matrix");

[...]

  gl.enableVertexAttribArray(positionAttributeLocation);

  gl.vertexAttribPointer(positionAttributeLocation, 4, gl.FLOAT, false, 0, 0);

  var matrix = utils.MakeTranslateMatrix(0.3,-0.7,0.0);
  //After gl.useProgram(..)
  //We transpose the matrix because of the column-major order
  gl.uniformMatrix4fv(matrixLocation, gl.FALSE, utils.transposeMatrix(matrix));
```

Example of library to perform matrix calculations without using our utils script: https://github.com/toji/gl-matrix

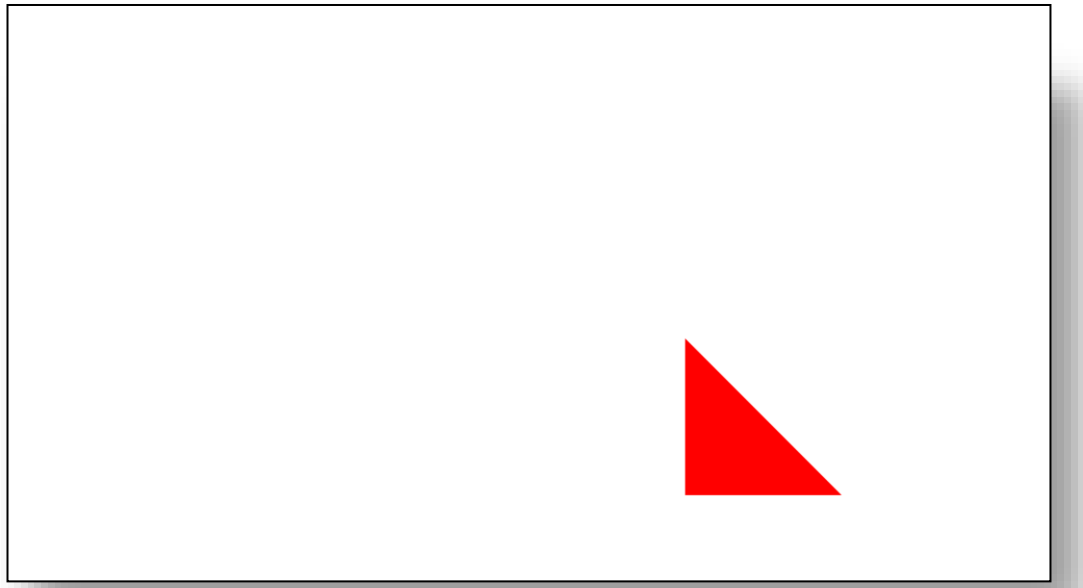
Doc: http://glmatrix.net/docs/

Transformations

Without transformations



With translation



Going in 3D → Let's draw a cube!

```
#version 300 es

in vec3 a_position;

uniform mat4 matrix;

void main() {

  gl_Position = matrix *
                vec4(a_position,1.0);
}
```

```
#version 300 es

precision mediump float;

out vec4 outColor;

void main() {

  outColor = vec4(1.0,0.0,0.0,1.0);

}
```

Going in 3D → Let's draw a cube!

```
var vertices = [// Vertex #:
 0.4, 0.4,-0.4, //  0
[...]
-0.4,-0.4, 0.4,  // 21
-0.4, 0.4, 0.4,  // 22
-0.4, 0.4,-0.4   // 23
];

var indices = [ // Face #:
        0,  1,  2,//  0
        [...]
        13, 22, 14,    //  9
        15, 16, 17,    // 10
        16, 23, 17     // 11
];
```

cubeDefinition.js

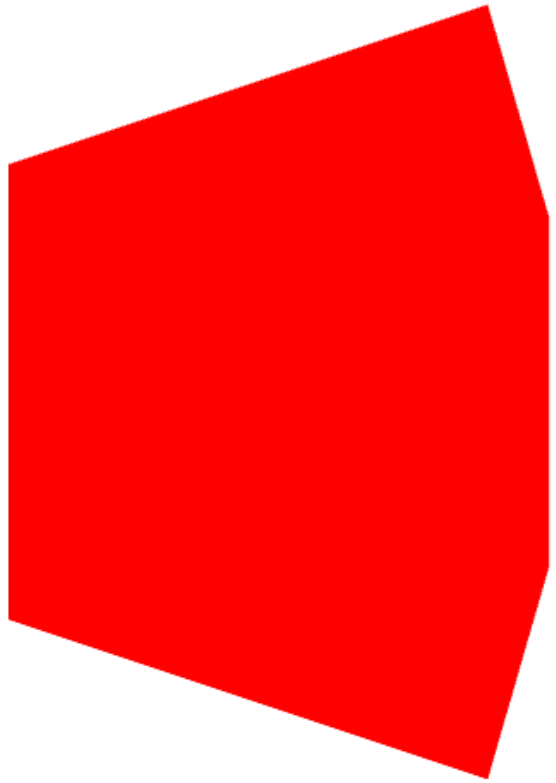
Going in 3D → Let's draw a cube!

```
<!DOCTYPE html>
<html lang="en-US">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style type="text/css">
      body { margin: 0; background-color: gray; }
      canvas {display: block; background-color: white; }
    </style>
    //Pay attention to the order of the scripts! The script always comes last since it
    //uses variables/functions from previous scripts
    <script type="text/javascript" src="lib/utils.js"></script>
    <script type="text/javascript" src="lib/cubeDefinition.js"></script>
    <script type="text/javascript" src="lib/script.js"></script>
</head>
<body>
  <canvas id="c"></canvas>
</body>
</html>
```

Going in 3D → Let's draw a cube!

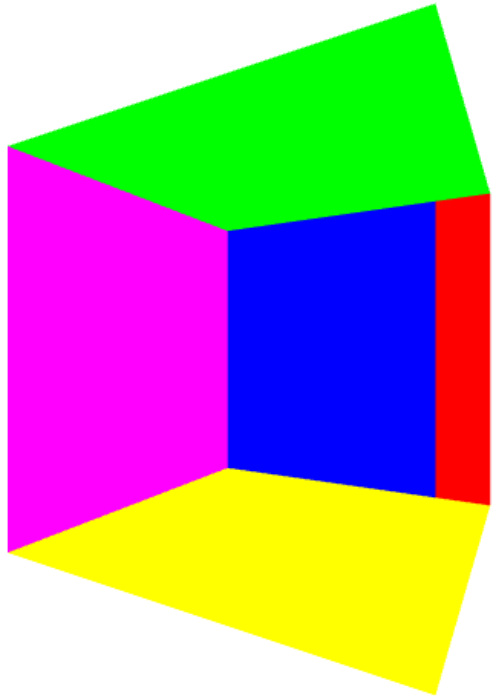
```
[...]
  var w = gl.canvas.width;
  var h = gl.canvas.height;
  //fovy, aspect ratio, near plane, far plane
  var perspectiveMatrix = utils.MakePerspective(90, w/h, 0.1, 100.0);
  //Camera position x,y,z, elev (x axis), ang (y axis)
  var viewMatrix = utils.MakeView(0.5, 0.0, 1.0, 0.0, -30.0);
  //perspectiveMatrix * viewMatrix
  var projectionMatrix = utils.multiplyMatrices(perspectiveMatrix, viewMatrix);

  gl.uniformMatrix4fv(matrixLocation, gl.FALSE, utils.transposeMatrix(projectionMatrix));

  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
  gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED_SHORT, 0 );
```

Going in 3D → Let's draw a cube!



Colouring the cube

We can add color the cube by defining a second attribute as for each vertex of the triangle (**`colors`** array in **`cubeDefinition.js`**), but that is the output



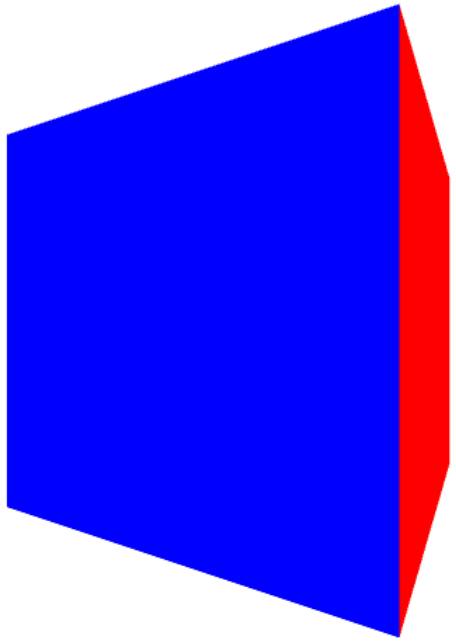
Colouring the cube

We need to enable depth testing (once during initialization)

```
gl.enable(gl.DEPTH_TEST);
```

And when we clear the color buffer, we also need to clear the depth buffer (every frame before drawing)

```
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
```

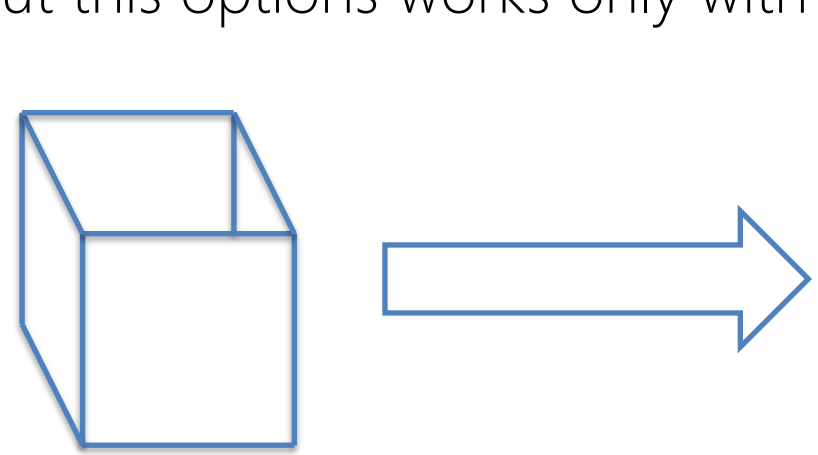


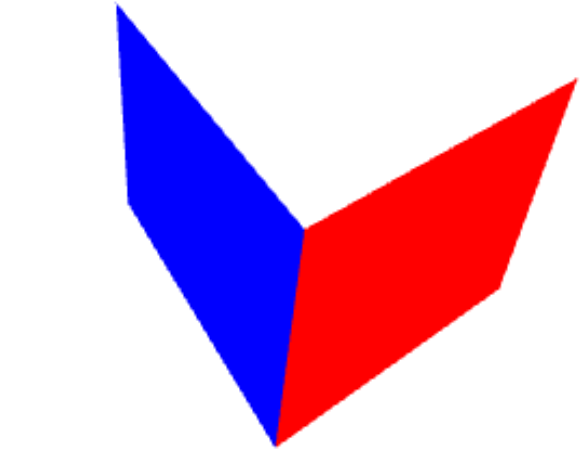
Backface Culling

Another option is to enable backface culling to get rid of facets which do not front-face the camera

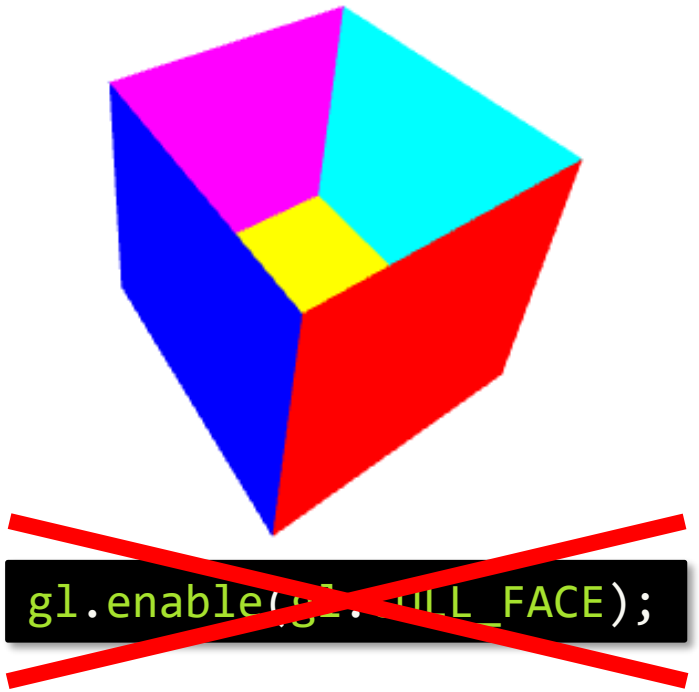
```
gl.enable(gl.CULL_FACE);
```

But this options works only with (closed) 2-manifold surfaces





```
gl.enable(gl.CULL_FACE);
```



~~gl.enable(gl.CULL_FACE);~~

Move the cube with the keyboard

To move the cube with the keybord we add a javascript listener

```
//cx, cy, and cz are the coordinates of the camera
//position, while elevation and angle refer to its
//rotation
function keyFunction(e){ //e is the event

      if (e.keyCode == 37) {  // Left arrow
        cx-=delta;
      }
      if (e.keyCode == 39) {  // Right arrow
        cx+=delta;
      }
      if (e.keyCode == 38) {  // Up arrow
        cz-=delta;
      }
      if (e.keyCode == 40) {  // Down arrow
        cz+=delta;
      }
      if (e.keyCode == 107) { // Add
        cy+=delta;
      }
      if (e.keyCode == 109) { // Subtract
        cy-=delta;
      }
      if (e.keyCode == 65) {  // a
        angle-=delta*10.0;
      }
      if (e.keyCode == 68) {  // d
        angle+=delta*10.0;
      }
      if (e.keyCode == 87) {  // w
        elevation+=delta*10.0;
      }
      if (e.keyCode == 83) {  // s
        elevation-=delta*10.0;
      }

  }

  window.addEventListener("keyup", keyFunction, false);
```

https://keycode.info/

Move the cube with the keyboard

- Moving the cube means we have to draw everything again, otherwise we won't see the change in position
- We need to change a bit the organization of the code

```
//Here are global variables like canvas, program,
matrices, handles…

function main() {
//Initialisation:
    • Getting canvas and webgl context
    • Creating shaders and program
    • Retrieving handles to attributes and uniforms
    • Calling drawScene() function
}
function drawScene() {
//Logic that needs to be repeated every time before
drawing a new frame:
    • Setting up VBOs (we will see in a while that we
      won't need to do this every frame with VAOs)
    • gl.useProgram()
    • Updating and passing uniforms to the GLSLprogram
    • Draw call
}
```

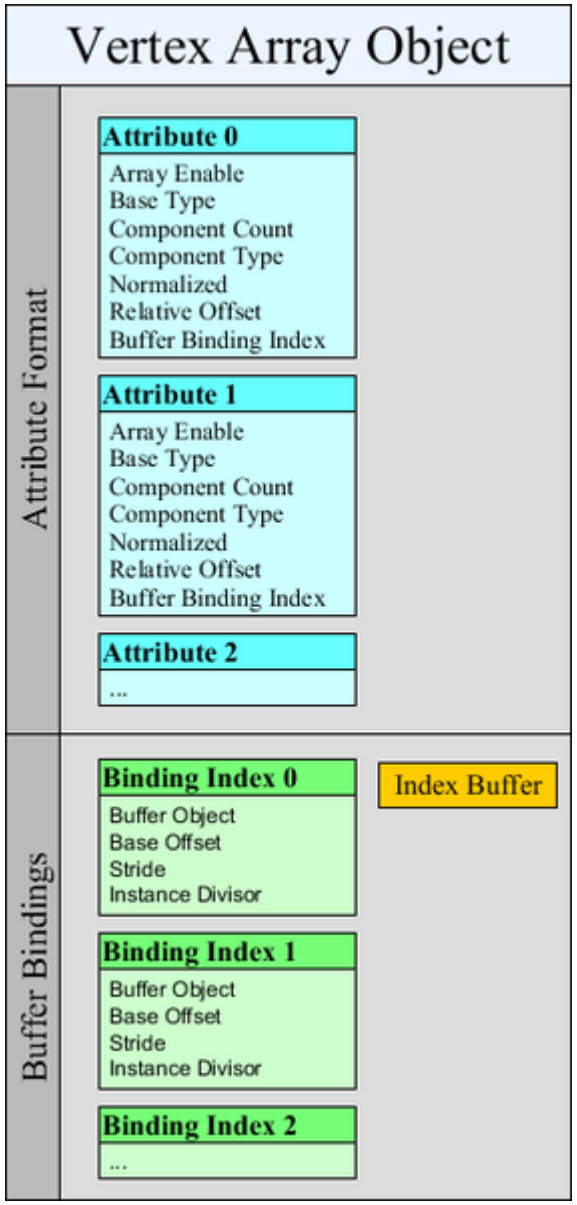
```
function keyFunction(e){
  //Key listeners that modify some parameters e.g.,
  position of the camera
  [..]
  //The requestAnimationFrame function tells the browser
  to call the specified function before the next repaint
  so that we can update the rendered image with the latest
  changes
  window.requestAnimationFrame(drawScene);
}

window.onload = main;
```

Vertex Array Objects (VAO)

What if I need to draw the same object multiple times?

- Option 1: bind, set attributes and enable attributes for each Vertex Buffer Object every time the frame is drawn
- Option 2: Vertex Array Objects:
  - Encapsulates all the states to specify vertex data
  - Stores any subsequent vertex attribute call:
    - glEnableVertexAttribArray
    - glDisableVertexAttribArray.
    - attribute configurations from glVertexAttribPointer.
    - VBO associated with vertex attributes via glVertexAttribPointer.

  A.k.a. ….make these calls once!



VAOs – How to use them

```
[...]
  // Create a vertex array object
  var vao = gl.createVertexArray();
  gl.bindVertexArray(vao);
  //All the calls to set up the VBOs are now stored in the currently-bound vao
  var positionBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
  gl.enableVertexAttribArray(positionAttributeLocation);
  gl.vertexAttribPointer(positionAttributeLocation, 3, gl.FLOAT, false, 0, 0);

  var colorBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
  gl.enableVertexAttribArray(colorAttributeLocation);
  gl.vertexAttribPointer(colorAttributeLocation, 3, gl.FLOAT, false, 0, 0);

  var indexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
  gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(indices), gl.STATIC_DRAW);
```

VAOs – How to use them

```
  drawScene();

  function drawScene() {

    gl.bindVertexArray(vao); /*** No need to repeat all the vbos set up, just bind the right vao ***/
    var aspect = gl.canvas.clientWidth / gl.canvas.clientHeight;
    var zNear = 1;
    var zFar = 2000;
    var fieldOfViewDeg = 40;
    var perspectiveMatrix = utils.MakePerspective(fieldOfViewDeg, aspect, zNear, zFar);

    viewMatrix = utils.MakeView(cx, cy, cz, elevation, angle);
    projectionMatrix = utils.multiplyMatrices(perspectiveMatrix, viewMatrix);

    gl.uniformMatrix4fv(matrixLocation, gl.FALSE, utils.transposeMatrix(projectionMatrix));

    gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED_SHORT, 0 );
  }
}
```

Move the cube with the keyboard – with VAOs

```
//Here are global variables like canvas, program,
matrices, handles…

function main() {
//Initialisation:
    • Getting canvas and webgl context
    • Creating shaders and program
    • Retrieving handles to attributes and uniforms
    • Setting up VBOs with VAOs
    • Calling drawScene() function
}
function drawScene() {
//Logic that needs to be repeated every time before
drawing a new frame:
    • gl.useProgram()
    • Updating and passing uniforms to the GLSLprogram
    • Bind VAOs
    • Draw call
}
```

```
function keyFunction(e){
  //Key listeners that modify some parameters e.g.,
  position of the camera
  [..]
  //The requestAnimationFrame function tells the browser
  to call the specified function before the next repaint
  so that we can update the rendered image with the latest
  changes
  window.requestAnimationFrame(drawScene);
}

window.onload = main;
```

Animations

- What if we want to have continuous animations?

```
//Here are global variables like canvas, program,
matrices, handles…

function main() {
//Initialisation:
    • Getting canvas and webgl context
    • Creating shaders and program
    • Retrieving handles to attributes and uniforms
    • Setting up VBOs with VAOs
    • Calling drawScene() function
}

function animate() {
//Logic for your frame-by-frame update of the animation
goes here
}
```

```
function drawScene() {
//Logic that needs to be repeated every time before
drawing a new frame:
    • animate() function to update the animation
    • gl.useProgram()
    • Updating and passing uniforms to the GLSLprogram
    • Bind VAOs
    • Draw call

    //By recursively calling the drawScene function in
    //the requestAnimationFrame function the animation
    //is updated every frame
    window.requestAnimationFrame(drawScene);
}


function keyFunction(e){
  [..]
}

window.onload = main;
```

Animating the cube

```
var lastUpdateTime = (new Date).getTime(); //At beginning of the script

function animate(){
  var currentTime = (new Date).getTime();
  if(lastUpdateTime){
    //Smooth the animation with the time between frames
    var deltaC = (30 * (currentTime - lastUpdateTime)) / 1000.0;
    cubeRx += deltaC;
    cubeRy -= deltaC;
    cubeRz += deltaC;

    if (flag == 0) cubeS += deltaC/100;
    else cubeS -= deltaC/100;

    if (cubeS >= 1.5) flag = 1;
    else if (cubeS <= 0.5) flag = 0;
  }
  worldMatrix = utils.MakeWorld(cubeTx, cubeTy, cubeTz, cubeRx, cubeRy, cubeRz, cubeS);
  lastUpdateTime = currentTime;
}
```

Animating the cube

```
function drawScene() {
  animate();

  gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);
  gl.clearColor(0, 0, 0, 0);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);
  gl.enable(gl.CULL_FACE);

  gl.bindVertexArray(vao);

  var viewMatrix = utils.MakeView(cx, cy, cz, elevation, angle);
  var projectionMatrix = utils.multiplyMatrices(viewMatrix, worldMatrix);
  var projectionMatrix = utils.multiplyMatrices(perspectiveMatrix, projectionMatrix);
  gl.uniformMatrix4fv(matrixLocation, gl.FALSE, utils.transposeMatrix(projectionMatrix));

  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
  gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED_SHORT, 0 );
  window.requestAnimationFrame(drawScene);
}
```