Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

Computer Viewing
Projection

Ed Angel
Professor Emeritus of Computer Science
University of New Mexico

Objectives

- Introduce the mathematics of projection
- Add WebGL projection functions in MV.js

Projections and Normalization

- The default projection in the eye (camera) frame is orthogonal
- For points within the default view volume

$$x_p = x$$
$$y_p = y$$
$$z_p = 0$$

- Most graphics systems use *view normalization*
  - All other views are converted to the default view by transformations that determine the projection matrix
  - Allows use of the same pipeline for all views

Homogeneous Coordinate Representation

The University of New Mexico

default orthographic projection

$$x_p = x$$
$$y_p = y$$
$$z_p = 0$$
$$w_p = 1$$

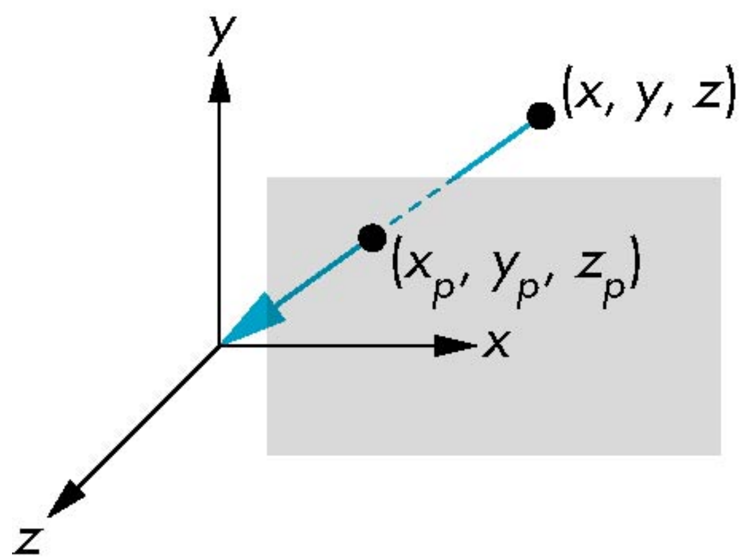
$$\mathbf{p}_p = \mathbf{M}\mathbf{p}$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In practice, we can let $\mathbf{M} = \mathbf{I}$ and set the z term to zero later

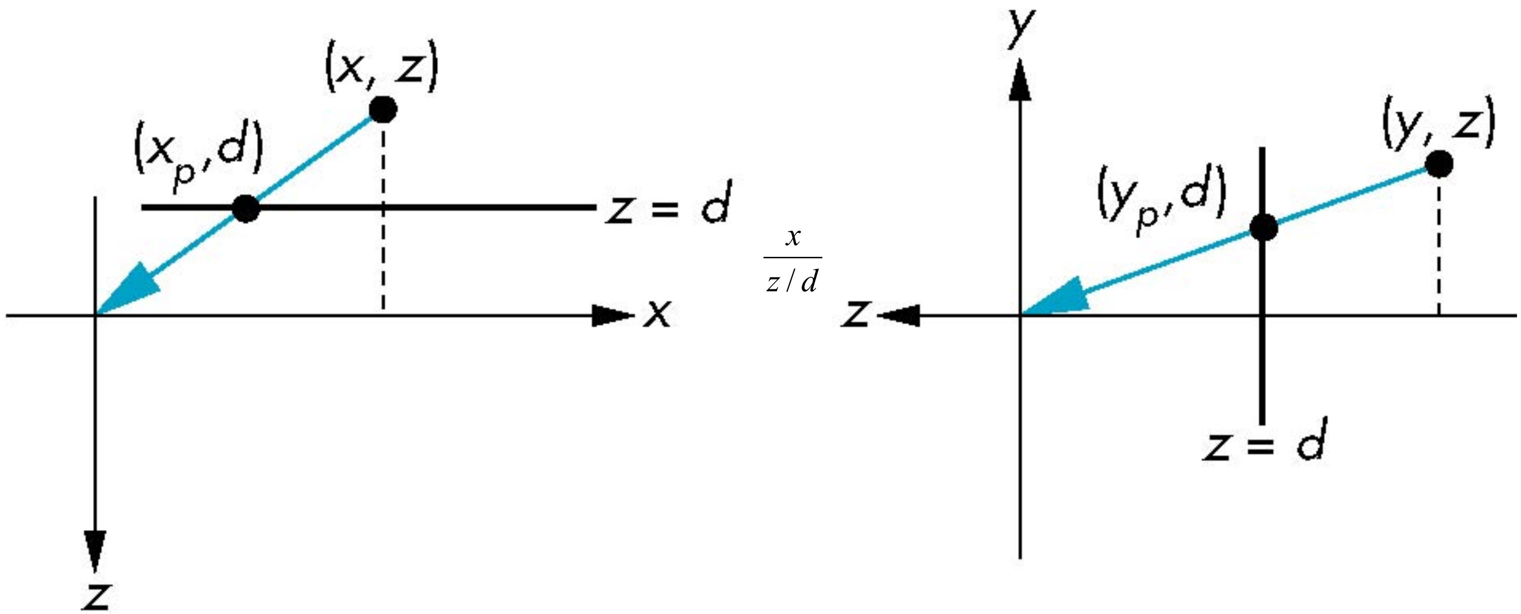
Simple Perspective

- Center of projection at the origin
- Projection plane $z = d,\ d < 0$



Perspective Equations

Consider top and side views



$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

The University of New Mexico

Homogeneous Coordinate Form

consider $\mathbf{q} = \mathbf{M}\mathbf{p}$ where $\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$

$$\mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \Rightarrow \quad \mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

Perspective Division

- However $w \neq 1$, so we must divide by w to return from homogeneous coordinates
- This *perspective division* yields

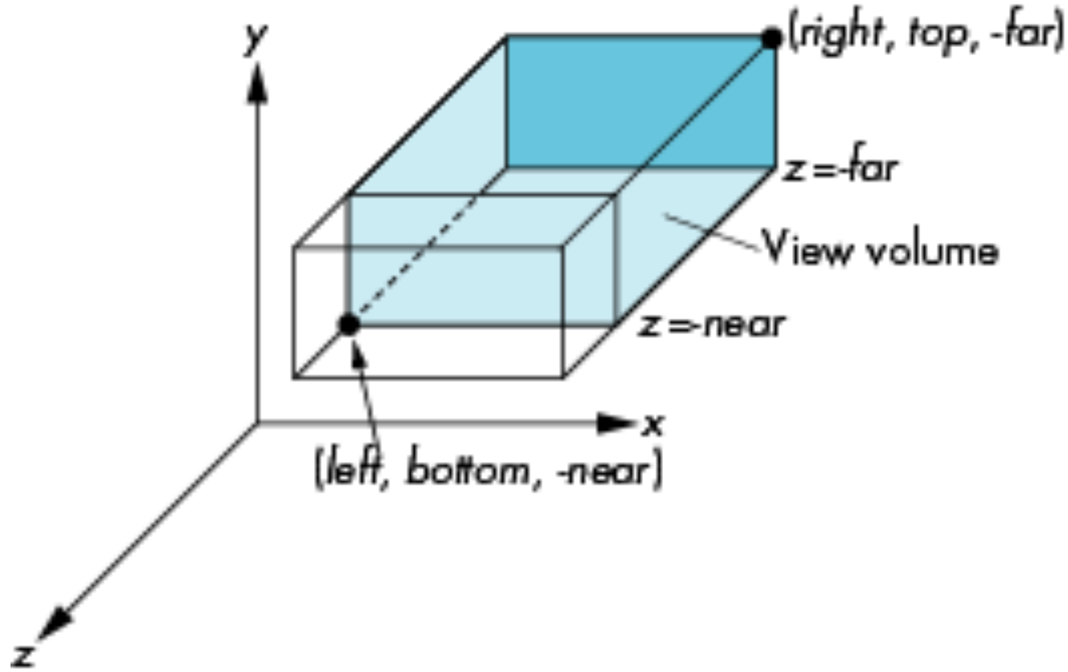
$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

the desired perspective equations

- We will consider the corresponding clipping volume with mat.h functions that are equivalent to deprecated OpenGL functions

WebGL Orthogonal Viewing

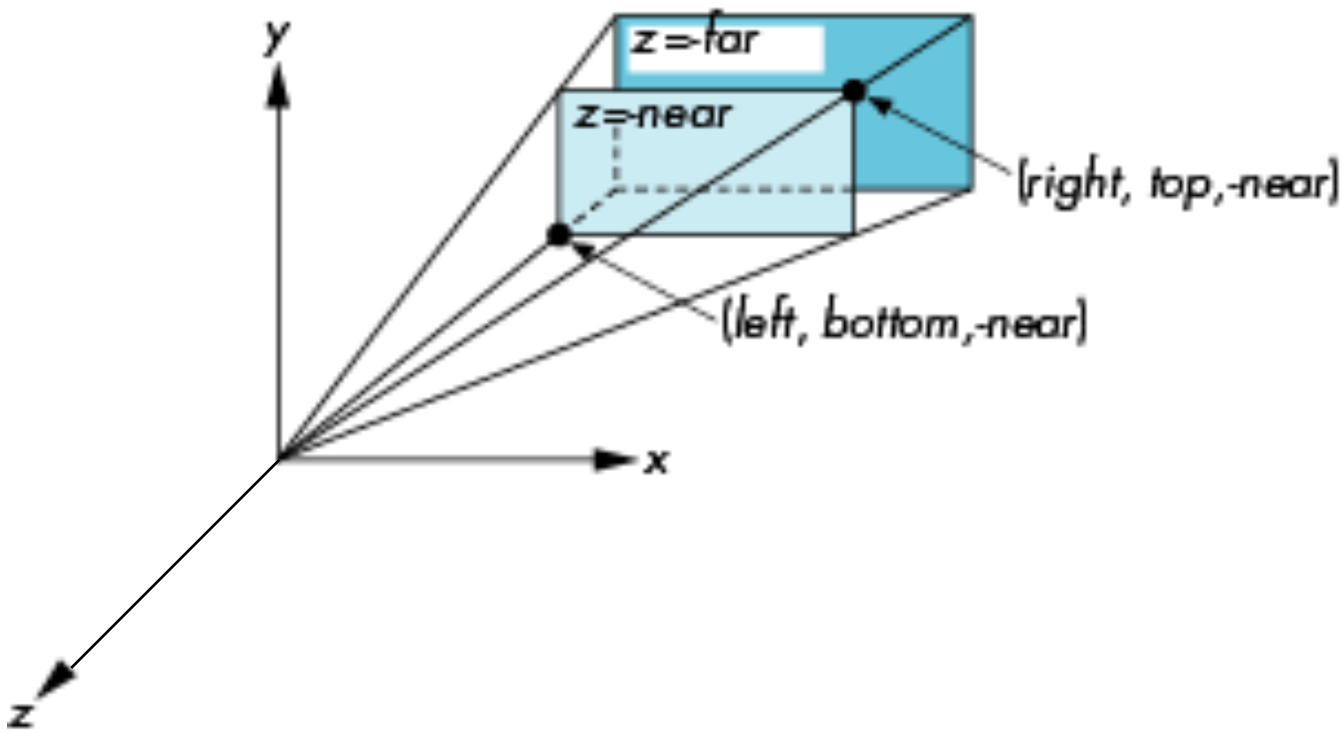
`ortho(left,right,bottom,top,near,far)`



`near` and **`far`** measured <u>from</u> camera

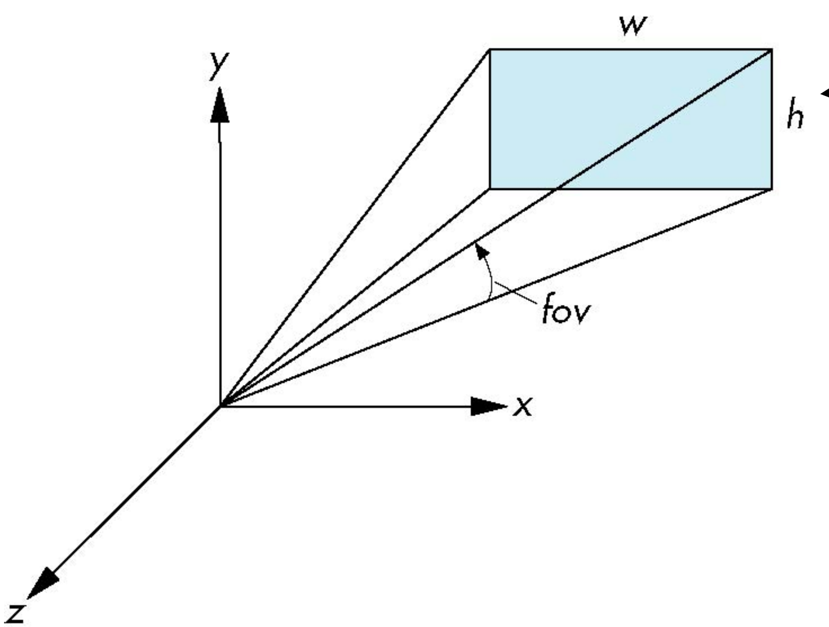
WebGL Perspective

`frustum(left,right,bottom,top,near,far)`



Using Field of View

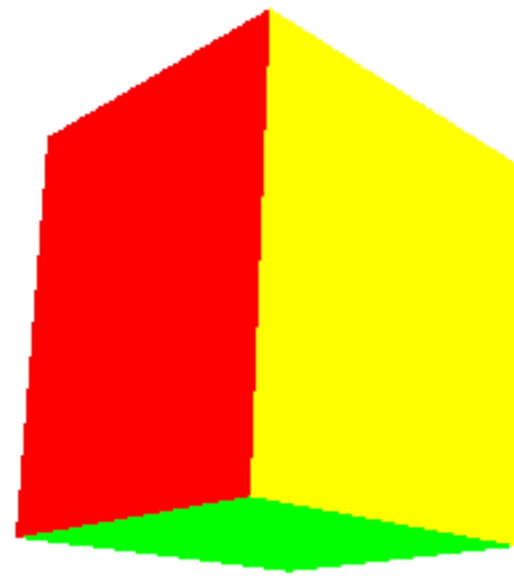
- With **`frustum`** it is often difficult to get the desired view
- **`perpective(fovy, aspect, near, far)`** often provides a better interface

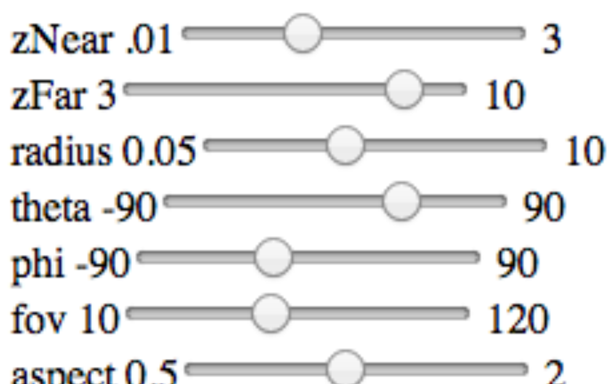


`aspect = w/h`

Computing Matrices

- Compute in JS file, send to vertex shader with gl.uniformMatrix4fv
- Dynamic: update in render() or shader





persepctive2.js

```
var render = function(){
   gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    eye = vec3(radius*Math.sin(theta)*Math.cos(phi),
       radius*Math.sin(theta)*Math.sin(phi), radius*Math.cos(theta));
   modelViewMatrix = lookAt(eye, at , up);
   projectionMatrix = perspective(fovy, aspect, near, far);
   gl.uniformMatrix4fv( modelViewMatrixLoc, false,
        flatten(modelViewMatrix) );
   gl.uniformMatrix4fv( projectionMatrixLoc, false,
       flatten(projectionMatrix) );
   gl.drawArrays( gl.TRIANGLES, 0, NumVertices );
   requestAnimFrame(render);
}
```

vertex shader

```
attribute  vec4 vPosition;
attribute  vec4 vColor;
varying vec4 fColor;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

void main() {
   gl_Position = projectionMatrix*modelViewMatrix*vPosition;
   fColor = vColor;
}
```