
# Programming with OpenGL
# Part 2: Complete Programs

Ed Angel

Professor of Emeritus of Computer Science

University of New Mexico


# Objectives

- Build a complete first program
  - Introduce shaders
  - Introduce a standard program structure
- Simple viewing
  - Two-dimensional viewing as a special case of three-dimensional viewing
- Initialization steps and program structure


# Program Execution

- WebGL runs within the browser
  - complex interaction among the operating system, the window system, the browser and your code (HTML and JS)
- Simple model
  - Start with HTML file
  - files read in asynchronously
  - start with onload function
    - event driven input


# Coordinate Systems

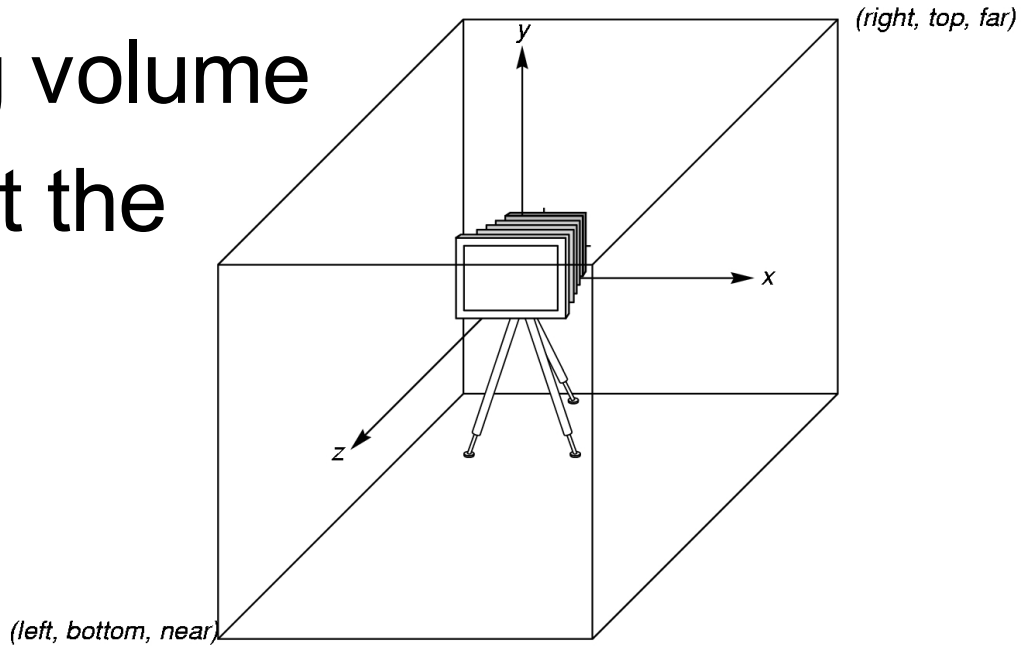
- The units in **`points`** are determined by the application and are called *object, world, model* or *problem coordinates*
- Viewing specifications usually are also in object coordinates
- Eventually pixels will be produced in *window coordinates*
- WebGL also uses some internal representations that usually are not visible to the application but are important in the shaders
- Most important is *clip coordinates*


# Coordinate Systems and Shaders

- Vertex shader must output in clip coordinates
- Input to fragment shader from rasterizer is in window coordinates
- Application can provide vertex data in any coordinate system but shader must eventually produce gl_Position in clip coordinates
- Simple example uses clip coordinates

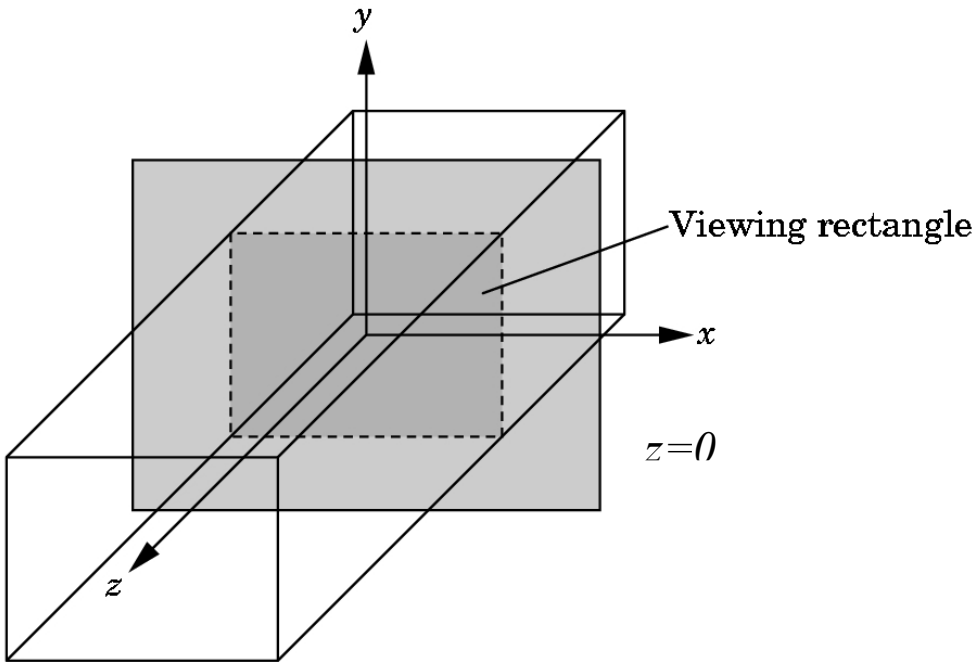

# WebGL Camera

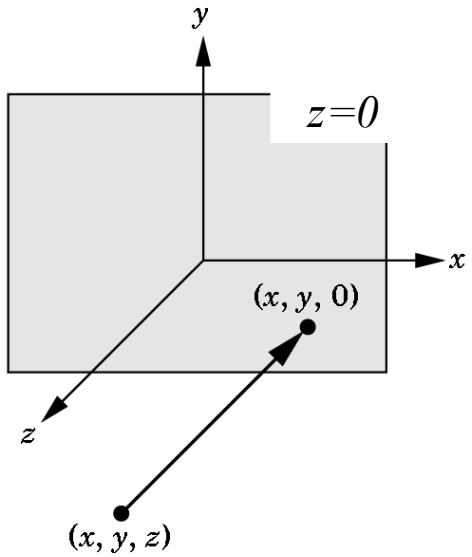
- WebGL places a camera at the origin in object space pointing in the negative $z$ direction
- The default viewing volume is a box centered at the origin with sides of length 2




# Orthographic Viewing

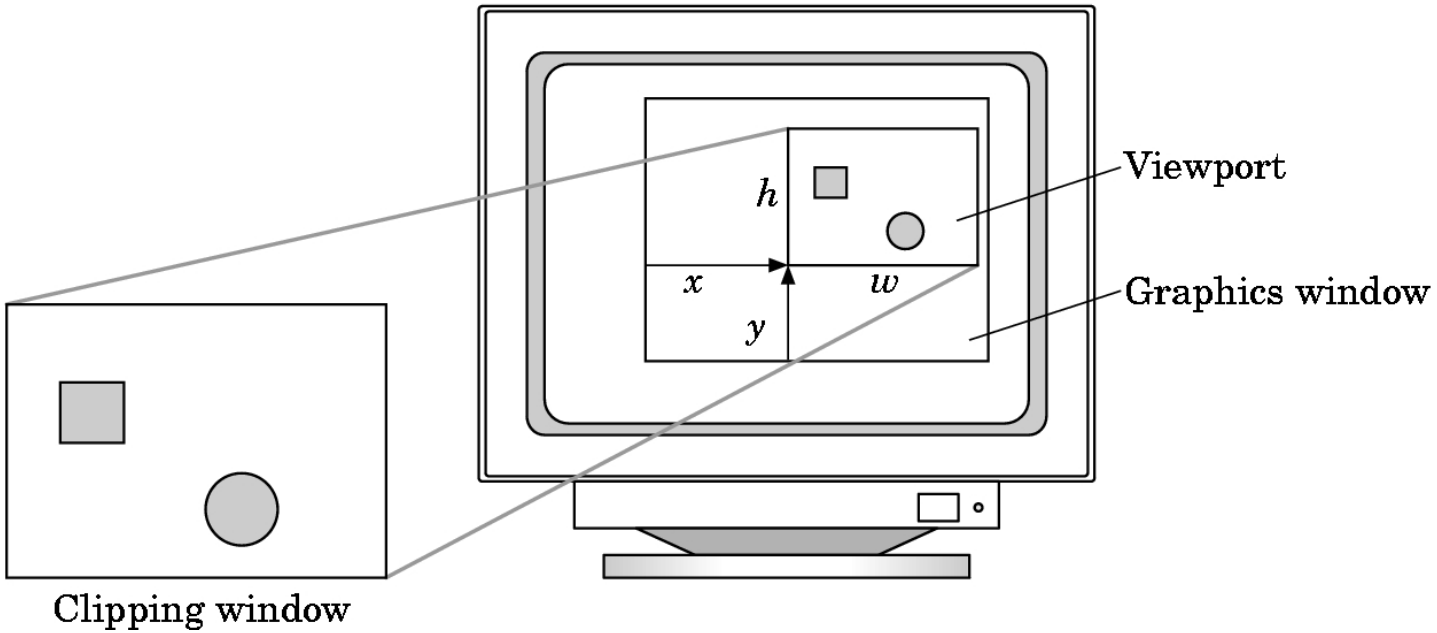
In the default orthographic view, points are projected forward along the $z$ axis onto the plane $z=0$




# Viewports

- Do not have use the entire window for the image: **`gl.viewport(x,y,w,h)`**
- Values in pixels (window coordinates)




# Transformations and Viewing

- In WebGL, we usually carry out projection using a projection matrix (transformation) before rasterization
- Transformation functions are also used for changes in coordinate systems
- Pre 3.1 OpenGL had a set of transformation functions which have been deprecated
- Three choices in WebGL
  - Application code
  - GLSL functions
  - MV.js


# First Assignment: Tessellation and Twist

- Consider rotating a 2D point about the origin

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

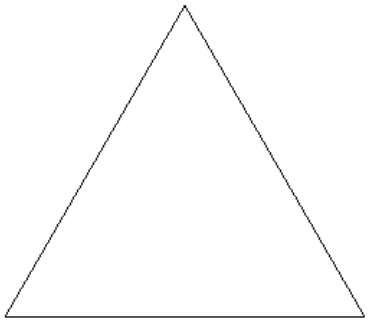
- Now let amount of rotation depend on distance from origin giving us **twist**

$$x' = x\cos(d\theta) - y\sin(d\theta)$$

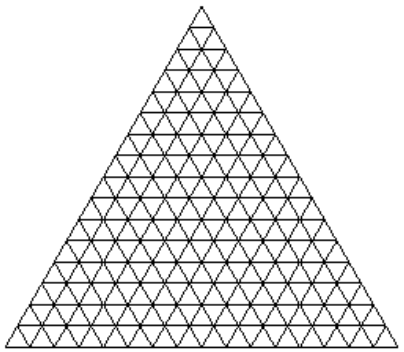
$$y' = x\sin(d\theta) + y\cos(d\theta)$$

$$d \propto \sqrt{x^2 + y^2}$$

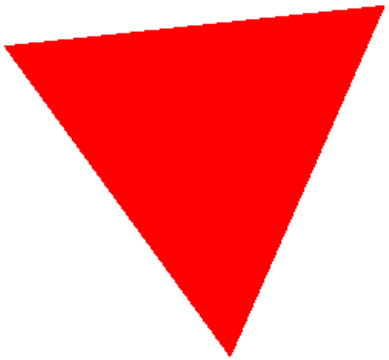

# Example



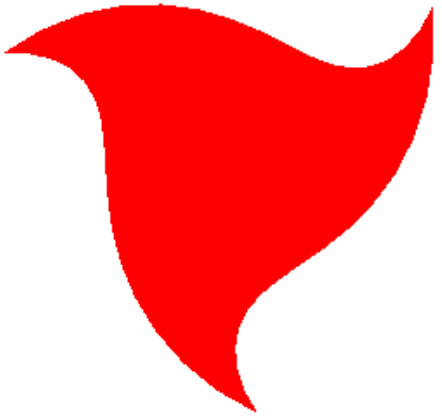
triangle



tessellated triangle



twist without tessellation



twist after tessellation