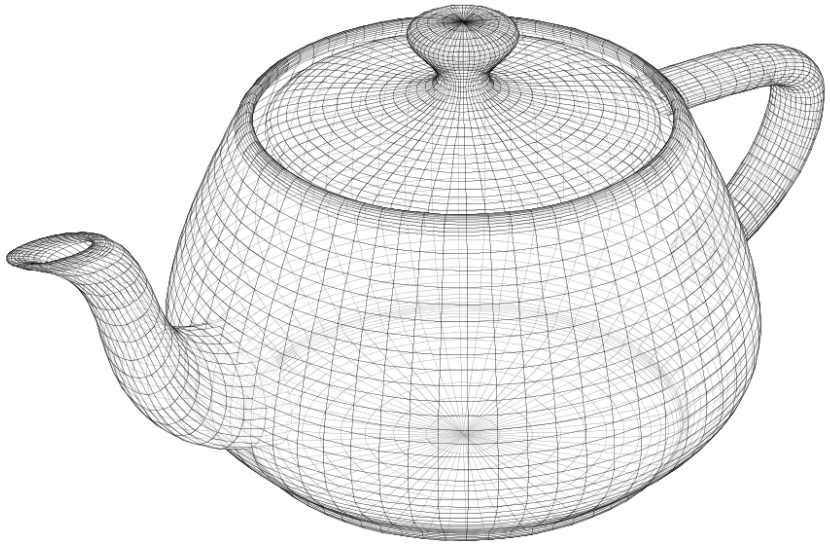
Animation in WebGL
Part 2



CS 418: Interactive Computer Graphics

Professor Eric Shaffer

ILLINOIS

Agenda for Today

- Debugging tips
- Review animation
- Understand what you are seeing
- MP1 Exercise – Changing coordinates in the vertex buffer

Debugging Tips: Chrome Development Tools

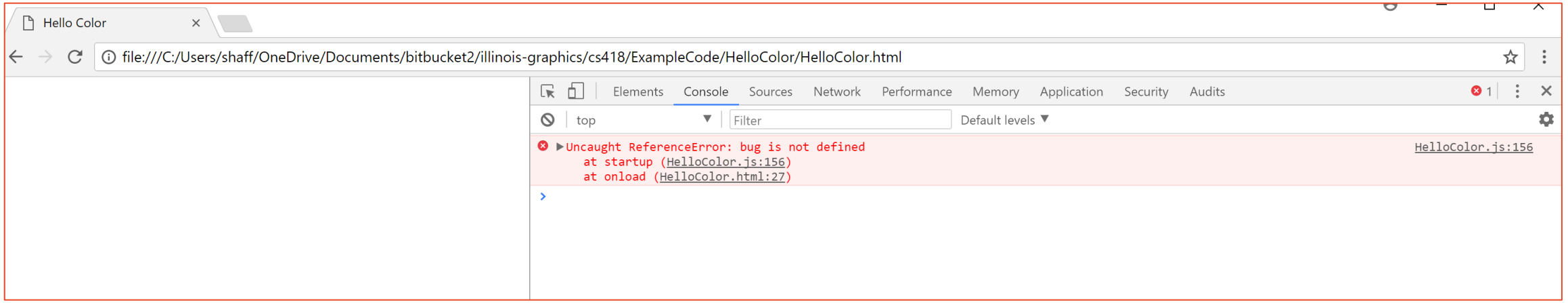
- Recommended Development Environment
  http://www.google.com/chrome

- Built-in debugging environment within Google Chrome browser
- Use this to set breakpoints, examine variables, etc.
- Check out the chrome devtools overview
  https://developer.chrome.com/devtools

- Use Ctrl + Shift + I (or Cmd + Opt + I on Mac) to open the DevTools.
- Use Ctrl + Shift + J (or Cmd + Opt + J on Mac) to open the DevTools and bring focus to the Console.
- Use Ctrl + Shift + C (or Cmd + Shift + C on Mac) to open the DevTools in Inspect Element mode, or toggle Inspect Element mode if the DevTools are already open.

ILLINOIS

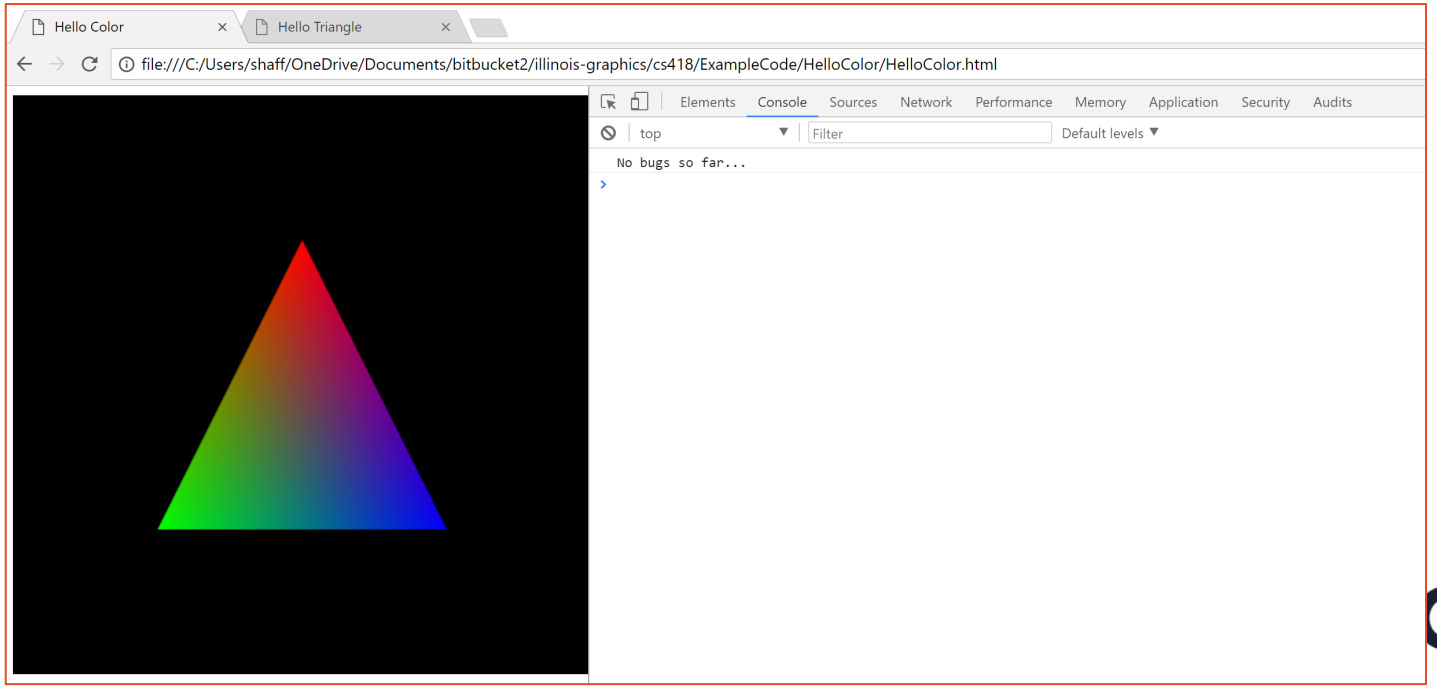
Dev Tools: How to Look at the Console…



ILLINOIS

Dev Tools: How to Print to the Console

```
function startup() {

    console.log("No bugs so far...");
    canvas = document.getElementById("myGLCanvas");
    gl = createGLContext(canvas);
    setupShaders();
    setupBuffers();
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    draw();
}
```



Most Important Debugging Tips Ever

Test as you write code…

Test each new significant piece of functionality

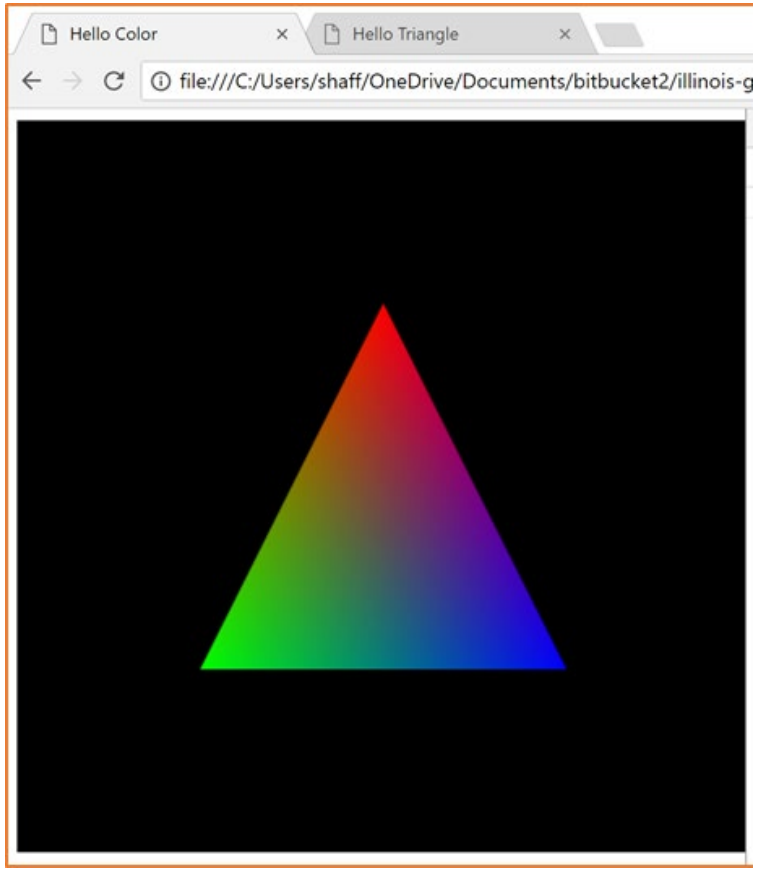
Write your test code so that it is reusable

WebGL: What do you see?

What is the default view volume in WebGL?

Where is the default eyepoint?

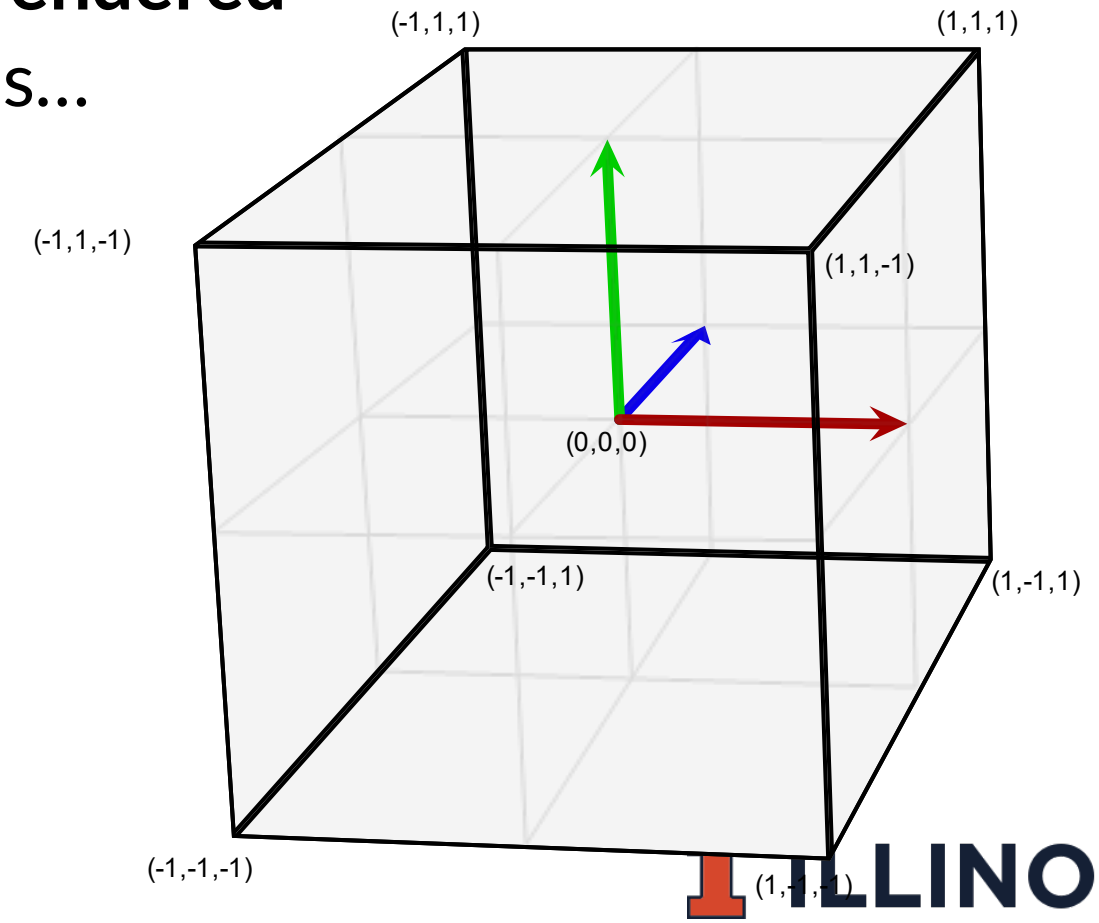
In what direction are you looking?



ILLINOIS

WebGL View Volume

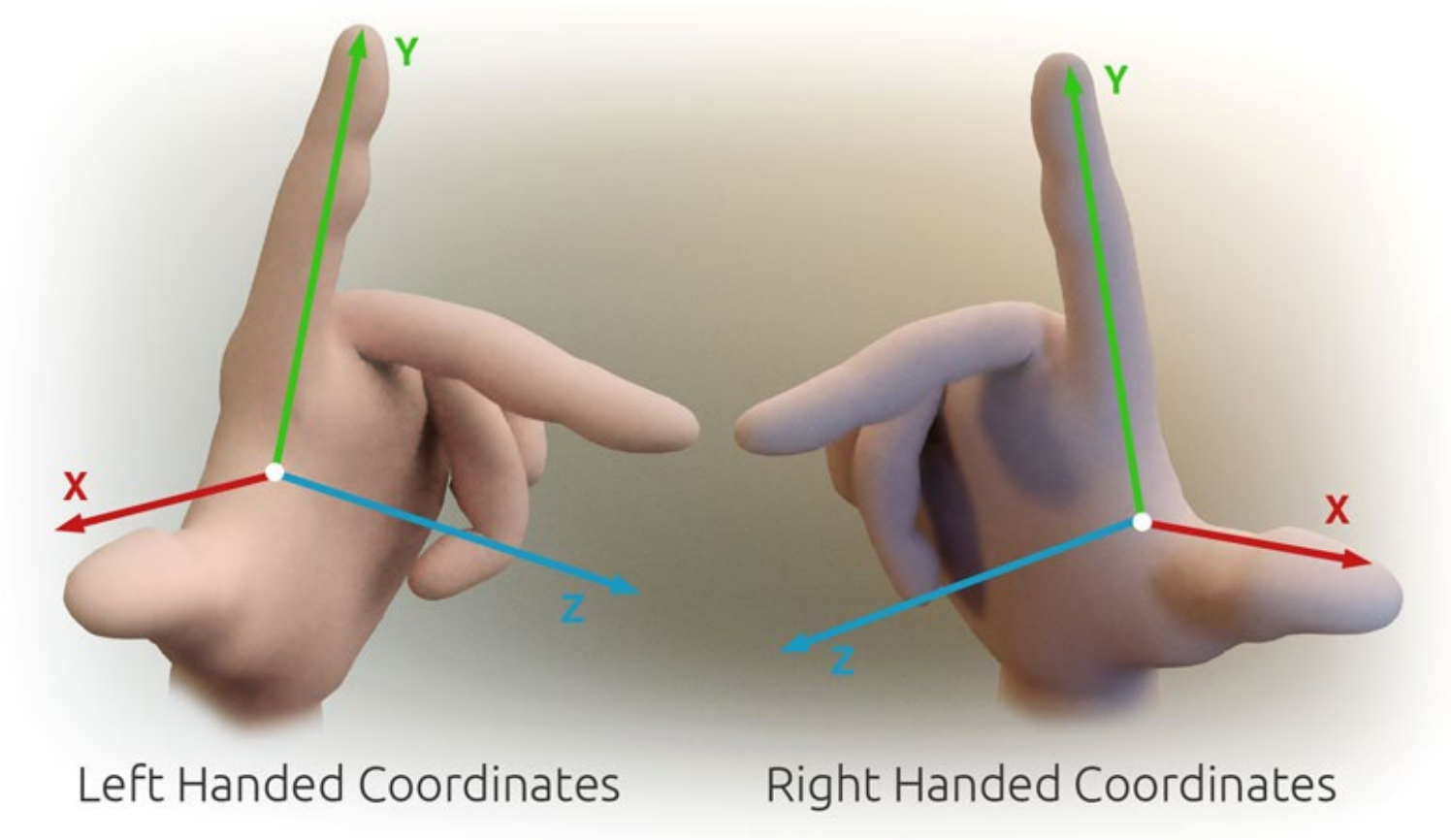
- View volume is a box running between [-1,1] in **x**, **y**, and **z**
- Geometry outside volume **will not be rendered**
- WebGL default eyepoint is on the z axis…



WebGL View Volume

The default view volume is called ***clip space***

WebGL clip space is a ***left-handed coordinate system***



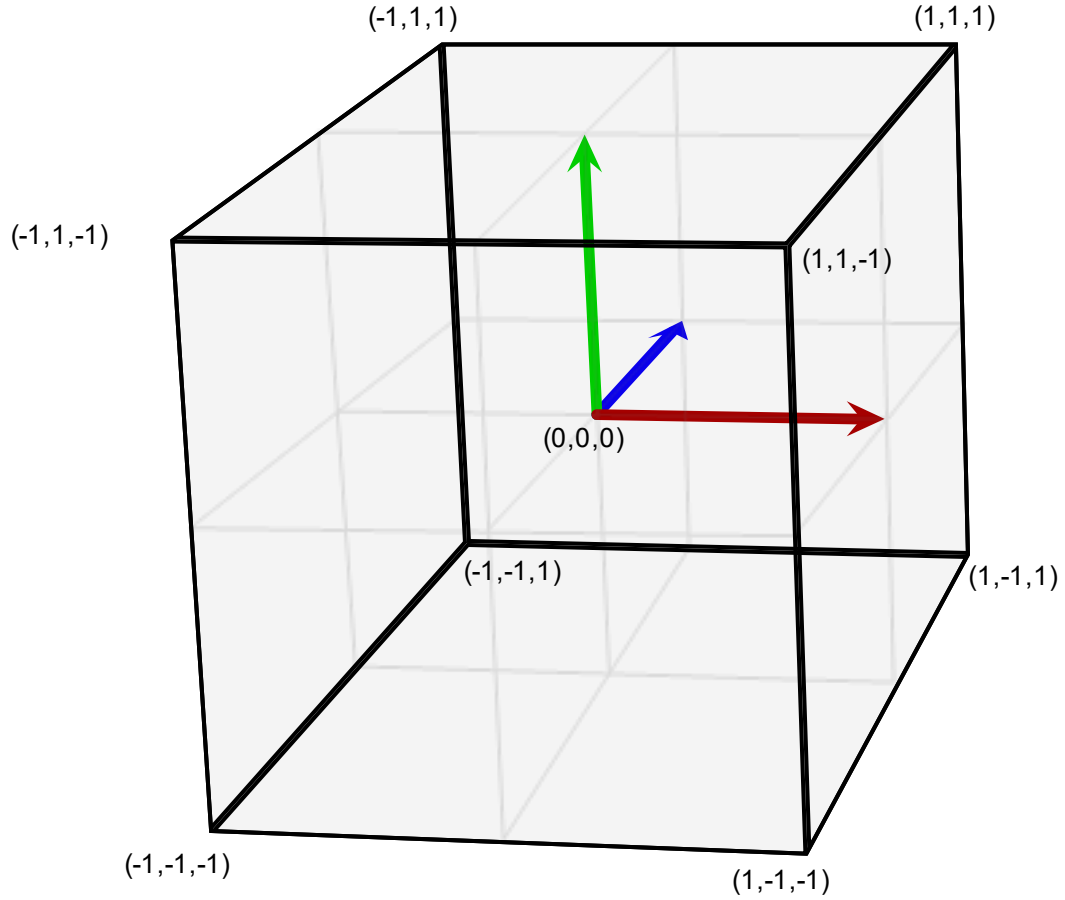
Many (most?) programmers don't realize this…they think WebGL is a right-handed system…this is because the tradition is to model in right-handed system looking down the -Z axis and then use a matrix transformation to invert the Z axis….

Which is +Z in relation to +X and +Y ?

ILLINOIS

WebGL View Volume

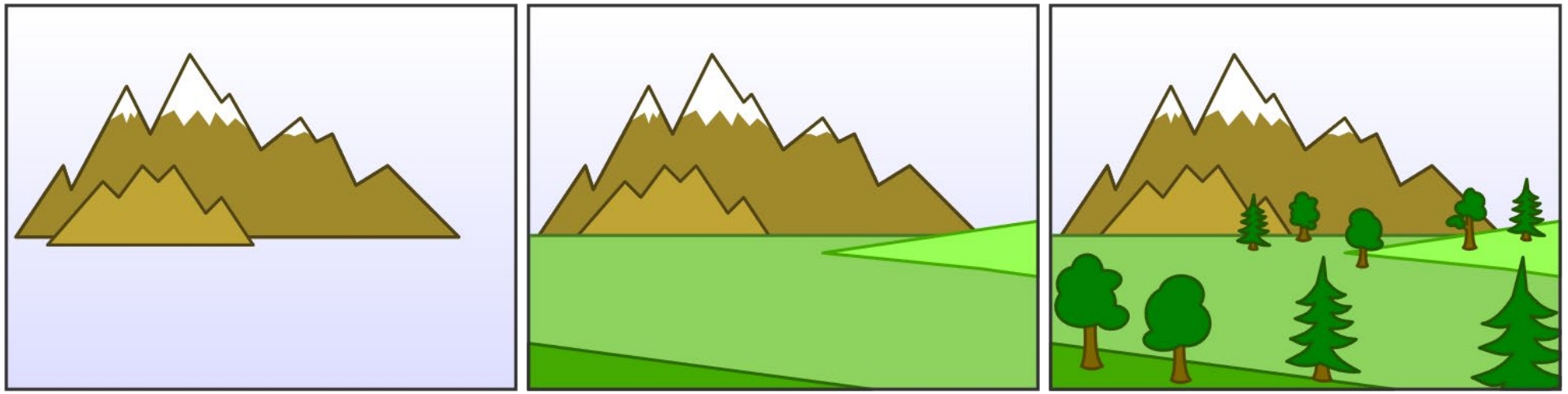
When WebGL renders, all geometry is projected onto Z=0 plane



ILLINOIS

Which Surfaces are Visible?

The default is that you see the surface you drew last…like painting



By Zapyon - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=14256921

Hidden Surface Removal

You can draw in any order and have WebGL remove hidden surfaces

```
/**
 * Startup function called from html code to start program.
 */
 function startup() {
  canvas = document.getElementById("myGLCanvas");
  gl = createGLContext(canvas);
  setupShaders();
  setupBuffers();
  gl.clearColor(0.0, 0.0, 0.0, 1.0);
  gl.enable(gl.DEPTH_TEST);
  draw();
  tick();
}
```

Hidden surface removal is also known as ***depth-testing*** or ***z-buffering***.

ILLINOIS

Hidden Surface Removal

If you are animating, you need to re-initialize the depth buffer each frame

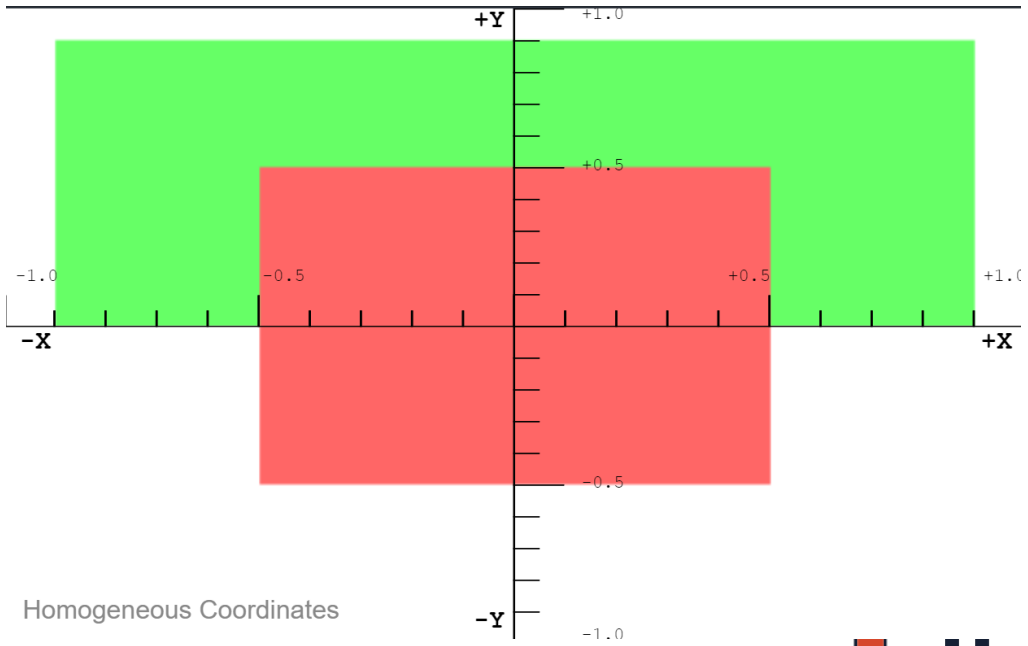
```
/**
 * Draw call that applies matrix transformations to model and draws model in frame
 */
function draw() {
  gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  glMatrix.mat4.identity(mvMatrix);
  glMatrix.mat4.identity(pMatrix);


  gl.bindBuffer(gl.ARRAY_BUFFER, vertexPositionBuffer);
  gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
                         vertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexColorBuffer);
  gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
                           vertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

  setMatrixUniforms();
  gl.drawArrays(gl.TRIANGLE_FAN, 0, vertexPositionBuffer.numberOfItems);
}
```

Hidden Surface Removal

In clip space, the z coordinate is depth.

Small z values come before large z values (negative before positive).

https://jsfiddle.net/mff99yu5



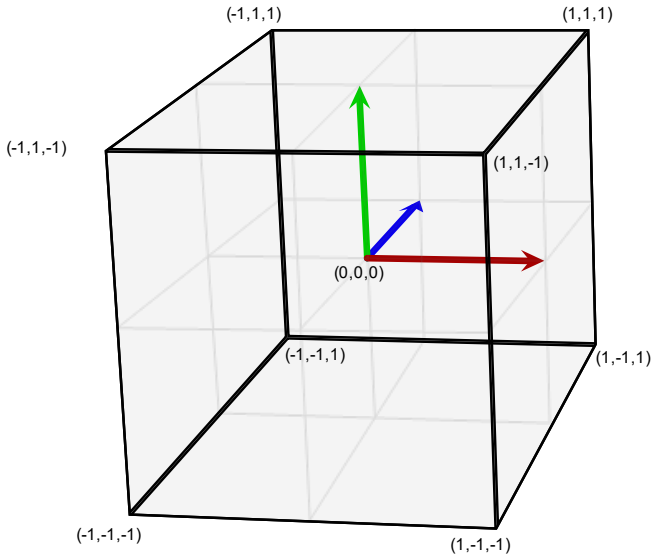
ILLINOIS

Changing the View Volume

Suppose we wanted to model in a view volume like $[-100,100]^3$

…bcause you hate decimal points

How could you use code to let you do this?



Clipspace

ILLINOIS

WebGL View Volume

- How can you work with more convenient coordinates (e.g. [-100, 100])?
  - Let's call those **world coordinates….**
  - View volume coordinates are often called **clip coordinates**

You can change coordinate systems pretty easily…

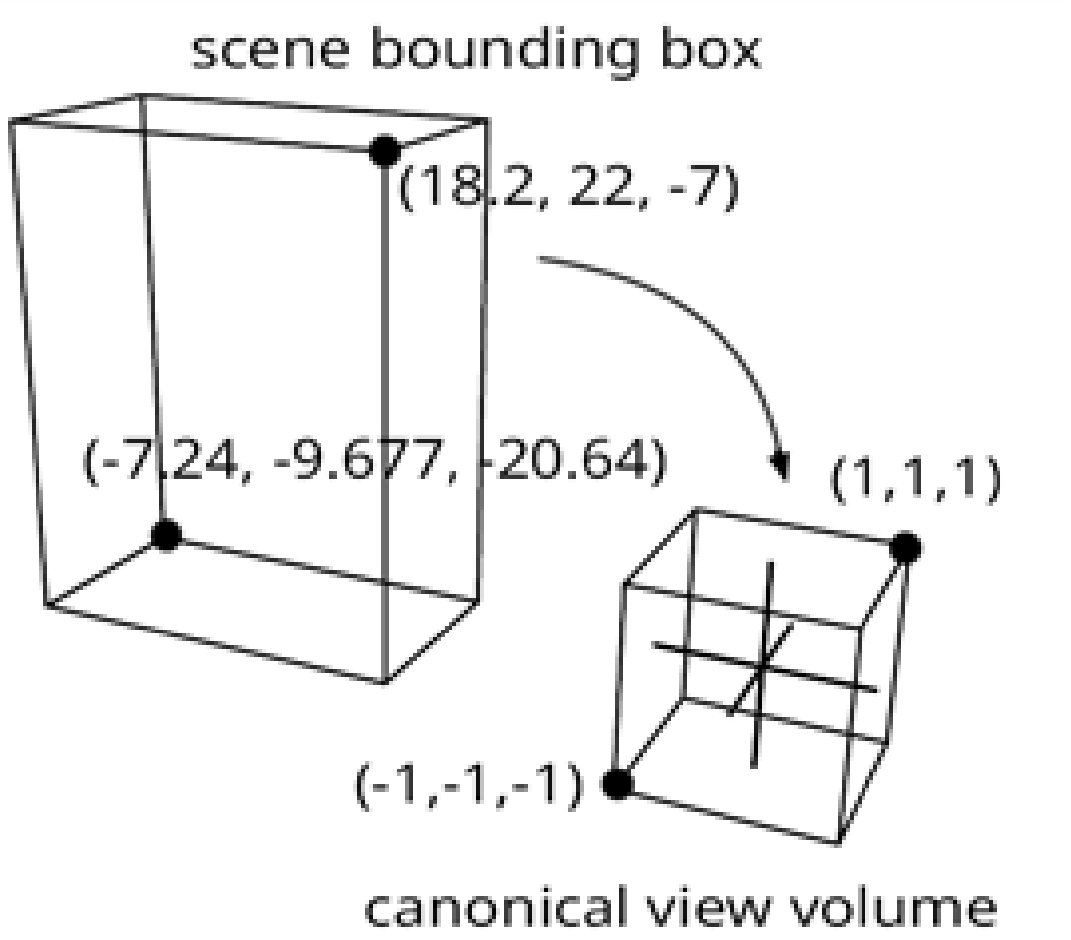
1. Specify your geometry in whatever world coordinates you want
2. Transform the coordinates into clip coordinates
    1. In other words…scale down all your geometry to fit in the view volume
    2. To do so we will use an **affine transformation**

Coding Up the Transformation

- One way to move geometry is to apply an affine transformation to it.
- We can encode that transformation using a matrix multiplication
- The transformation can be implemented in the vertex shader
- In the JavaScript we create a matrix using the glMatrix library
- That matrix is then passed to the vertex shader as a Uniform variable

- Let’s look at the code……

The ortho Transformation

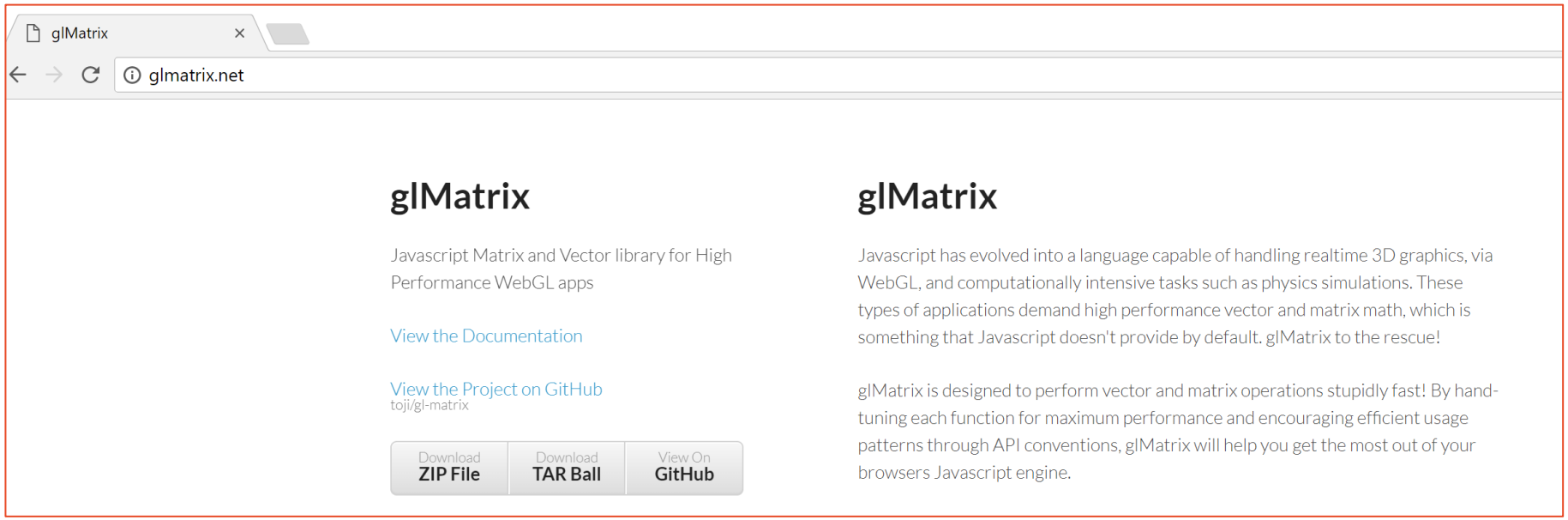


The ortho transformation maps any rectangular axis-aligned viewing volume to the WebGL view volume

It's basically just a translation and scale

The glmatrix Library

- A javascript library that supports 3D matrix and vector operations
- To use it grab a copy from glmatrix.net



ILLINOIS

The glmatrix Library

- Add a line to your html file with the location of the glmatrix library

```
<script src="gl-matrix-min.js"></script>
<script type="text/javascript" src="webgl-utils.js"></script>
<script src="Discussion1Demo.js"></script>
<body onload="startup();">
  <canvas id="myGLCanvas" width="500" height="500"></canvas>
</body>

</html>
```

- In this example, library is in the same folder/directory as html file
  - Also, a set of utility JavaScript code `webgl-utils.js` is being used as well

ILLINOIS

Transforming a Vertex in a Shader

```
<script id="shader-vs" type="x-shader/x-vertex">
   attribute vec3 aVertexPosition;
   attribute vec4 aVertexColor;

    uniform mat4 uPMatrix;
    varying vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix*vec4(aVertexPosition, 1.0);
        vColor = aVertexColor;
    }
</script>
```

- This vertex shader code expects a matrix to be loaded from the application
- The matrix is provided as a **uniform** variable

ILLINOIS

Transforming a Vertex in a Shader

```
<script id="shader-vs" type="x-shader/x-vertex">
   attribute vec3 aVertexPosition;
   attribute vec4 aVertexColor;

    uniform mat4 uPMatrix;
    varying vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix*vec4(aVertexPosition, 1.0);
        vColor = aVertexColor;
    }
</script>
```

- In GLSL a **uniform** is a variable that is the same for all executions of a shader program in a given draw call
- In GLSL an **attribute** is a variable that may be different for all executions of a shader program in a given draw call

Creating an ortho Matrix in JavaScript

First, when you create the shader program, get the uniform location

```
shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
```

Create the matrix

```
var pMatrix = glMatrix.mat4.create();
glMatrix.mat4.ortho(pMatrix, left, right , bottom , top , near, far);
```

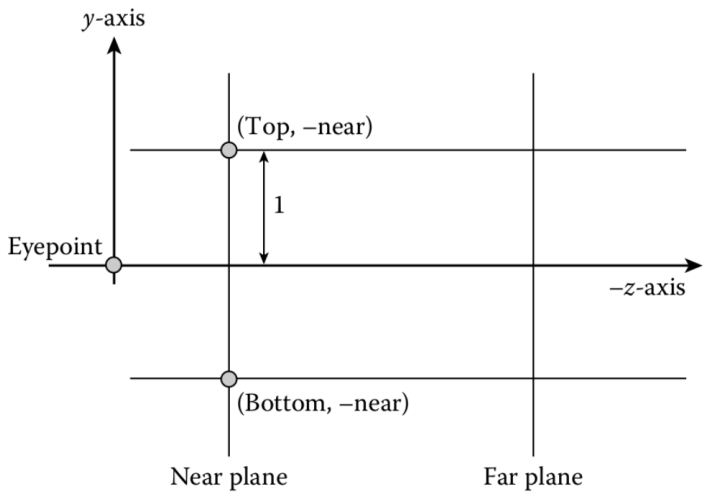
Send the matrix to the GPU before you issue a draw call

```
gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
```

The ortho Matrix

mat4.ortho(out,left,right,bottom,top,near,far)

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



The parameters **left** and **right** are the x coordinates of the planes forming those sides of the view volume
The parameters **top** and **bottom** are the y coordinates of the planes forming those sides of the view volume
The parameters **near** and **far** are distances down the negative z axis measured from the origin

ILLINOIS

Animation…

```
window.requestAnimationFrame(callback);
```

Parameters

`callback`

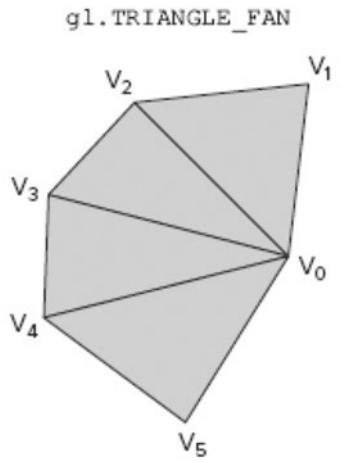
The function to call when it's time to update your animation for the next repaint. The callback function is passed one single argument, a `DOMHighResTimeStamp` similar to the one returned by `performance.now()`, indicating the point in time when `requestAnimationFrame()` starts to execute callback functions.

What would happen if you called requestAnimationFrame without drawing anything new?

ILLINOIS

Drawing Triangle Fans

WebGL offers as special drawing mode to render triangle fans



A triangle fan is a triangulated polygon with a central vertex

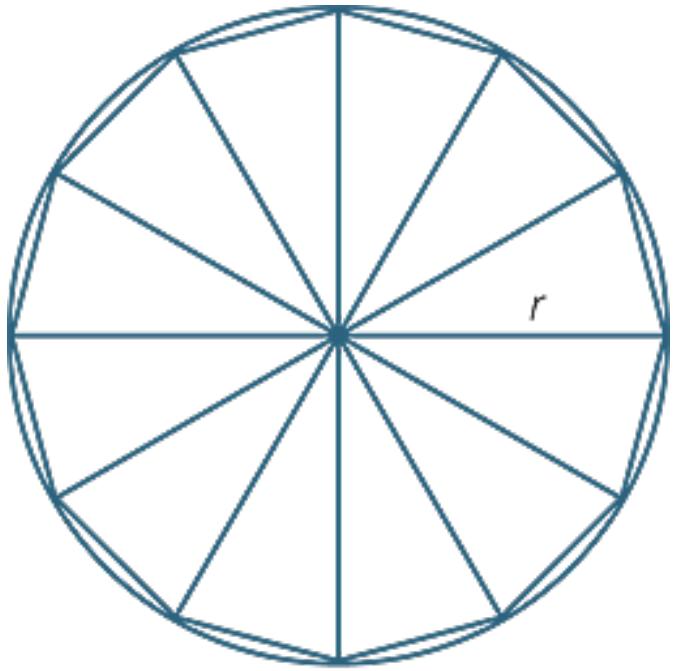
This central vertex forms one corner of every triangle

ILLINOIS

Drawing a Circle

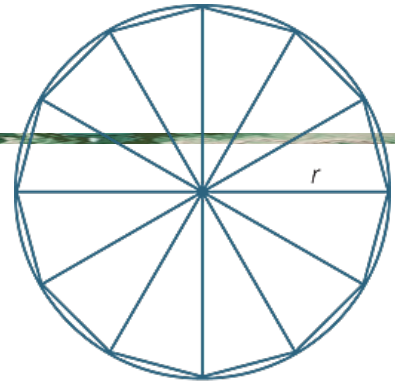
We can ***procedurally generate*** the geometry for a circle

Easy to approximate the shape using a triangle fan



ILLINOIS

Generating the Circle Vertices




```
// Start with vertex at the origin
var triangleVertices = [0.0,0.0,0.0];

//Generate a triangle fan around origin
var radius=0.5
var z=0.0;

for (i=0;i<=numVertices;i++){
    angle = i *  twicePi / numVertices;
    x=(radius * Math.cos(angle));
    y=(radius * Math.sin(angle));
    triangleVertices.push(x);
    triangleVertices.push(y);
    triangleVertices.push(z);
}
```

What would we draw if the loop were **for(i=0;i<numVertices;i++)** ?

Where in the array does the JavaScript push function place the new coordinate?

ILLINOIS