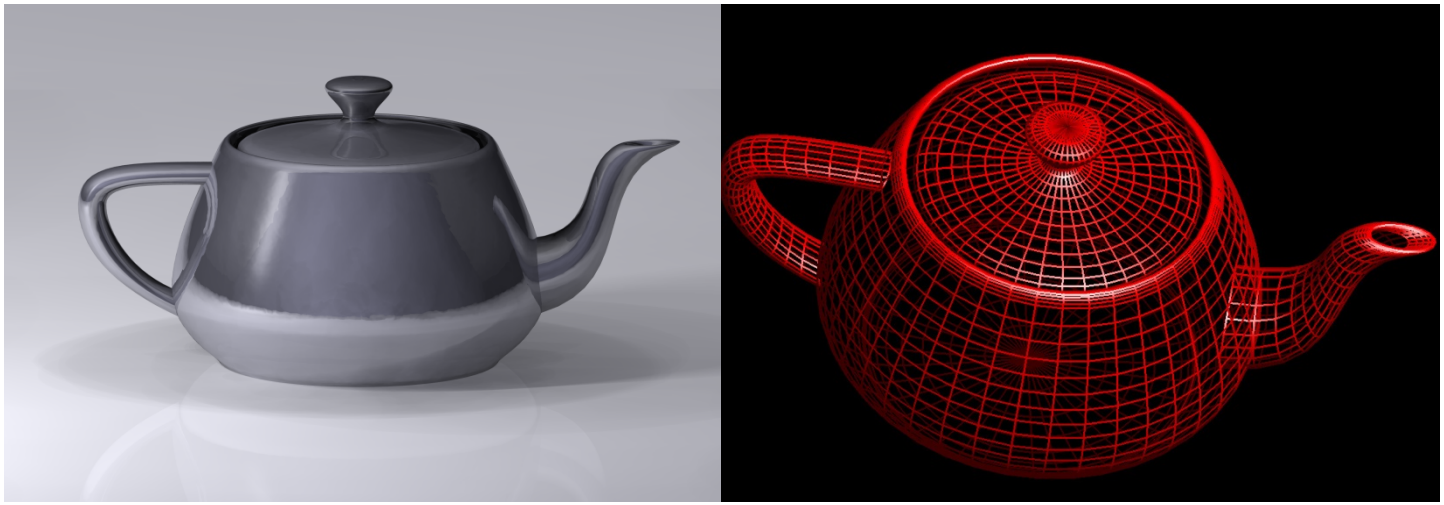
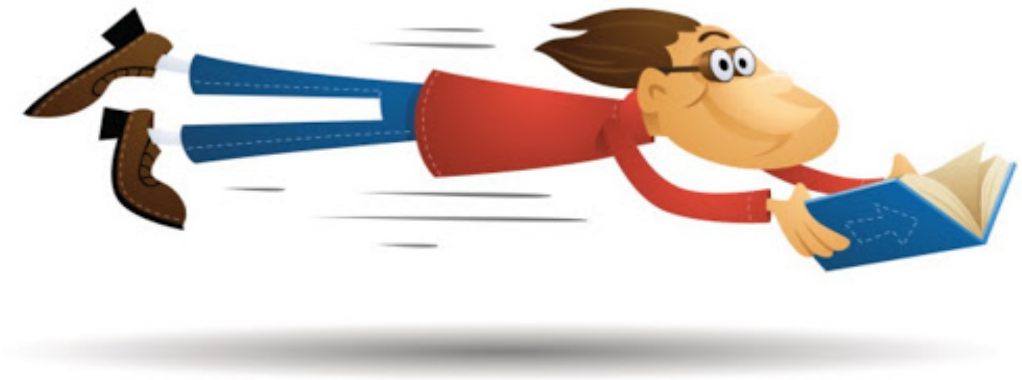


Foundations of Computer Graphics

Saurabh Ray

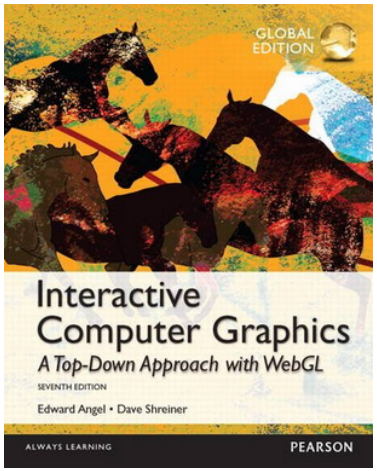
Reading



Reading for Lecture 3: Sections $2.10, 3.1, 3.6 - 3.10$.

Reading for Lecture 4: Sections 4.1, 4.3.

I suggest reading whole chapters even though not everything in the book was discussed in class.



Dev Tools

You can use chrome developer tools as a simple IDE.

In Chrome and press `Ctrl+Shift+I` to bring up the developer tools. (`Cmd+Opt+I` on Mac.)

Using "**Add folder to workspace**" add the folder where your code resides.

Now you can edit both html and javascript files from the sources tab.

Press `Ctrl + S` to save. **No autosave!**

WebGL Example: Draw a triangle

File: DrawTriangle.html

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">

<title>Draw Triangle</title>

<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="DrawTriangle.js"></script>

<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
void main(){
    gl_Position = vPosition;
}
</script>
```

WebGL Example: Draw a triangle

```
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;

void main(){
    gl_FragColor = vec4( 1.0, 0.0, 1.0, 1.0 );
}
</script>
</head>
<body>
   <canvas id="gl-canvas" width="512" height="512">
    HTML5 Canvas not supported!
   </canvas>
</body>
</html>
```

WebGL Example: Draw a triangle

File: DrawTriangle.js

```
"use strict";
var gl; // global variable

window.onload = function init() {
    // Set up WebGL
    var canvas = document.getElementById("gl-canvas");
    gl = WebGLUtils.setupWebGL( canvas );
    if(!gl){alert("WebGL setup failed!");}

    // Clear canvas
    gl.clearColor(0.0, 1.0, 0.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);

    // Load shaders and initialize attribute buffers
    var program = initShaders( gl, "vertex-shader", "fragment-shader" );
    gl.useProgram( program );
```

WebGL Example: Draw a triangle

x and y coordinates of three vertices.

```
// Load data into a buffer
var vertices = [ 0.0, 0.5, 0.4, -0.7, 0.6, 0.9];
var vBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);

// Do shader plumbing
var vPosition = gl.getAttribLocation(program, "vPosition");
gl.vertexAttribPointer(vPosition, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);

//Draw
gl.drawArrays(gl.TRIANGLES,0,3);
};
```

create a buffer and make it current

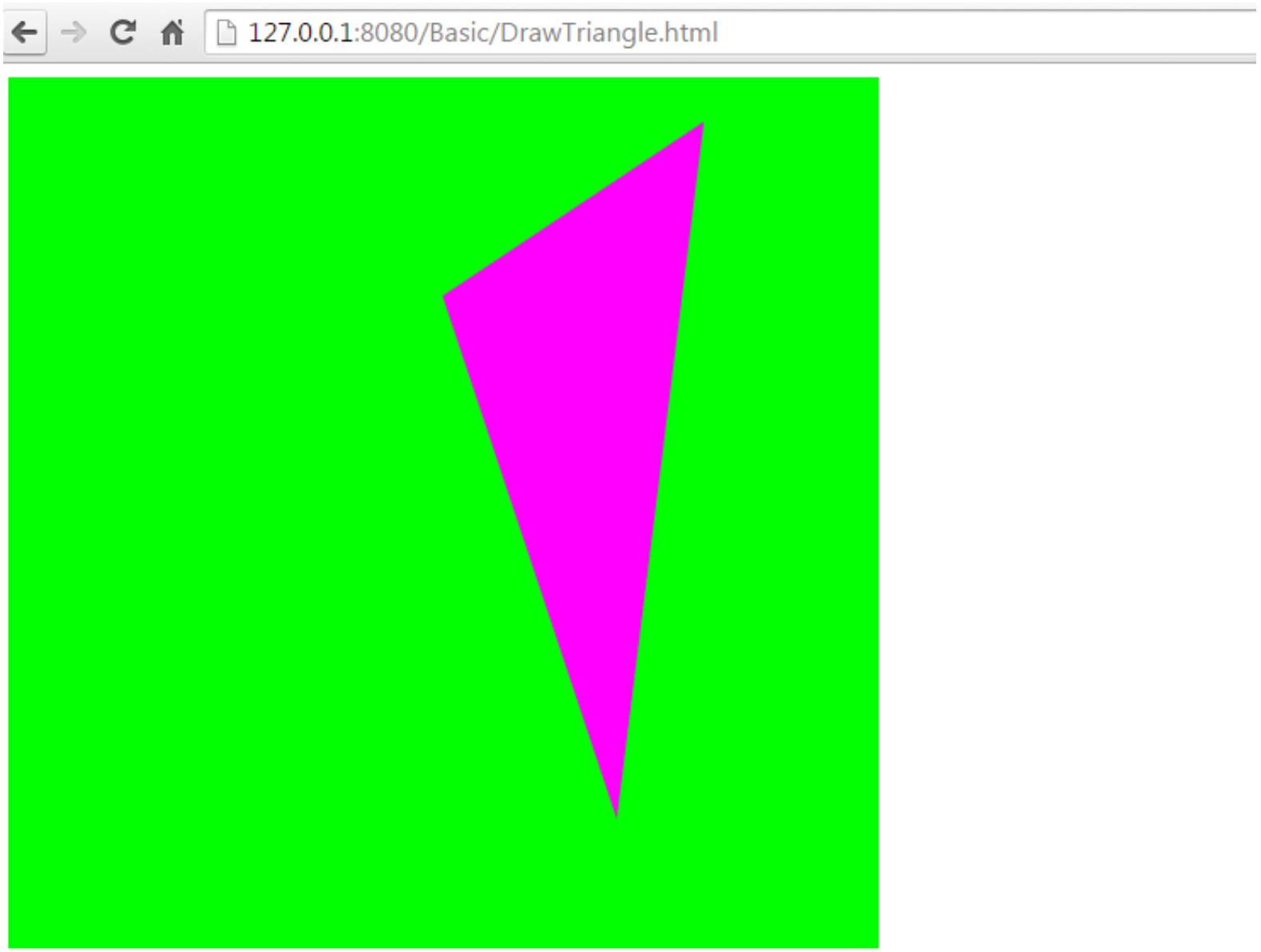
Transfer data

telling the shader how to read the data

WebGL requires C style arrays. Javascript arrays are objects.

WebGL Example: Draw a triangle

Output:



drawTriangle

Look at the code in `drawTriangle.html` and `drawTriangle.js`

Questions:

Which part of the code runs in the CPU? Which part runs in the GPU?

What is `gl`?

We have two coordinates for each vertex. How does the GPU know this?

Why do we use `new Float32Array(vertices)` in `gl.bufferData(...)`?

Where are we setting the color of the triangle?

What will happen if we change the 0.6 in the `vertices` array to 1.6?

`vPosition` is of type `vec4` in the vertex shader but we are passing only two coordinates. Where are the other two set?

Why is the last argument of `gl.drawArrays(...)` 3 and not 1?

Coding Practice

Tasks:

Draw an equilateral triangle whose centroid is at $(0, 0)$.

Draw a square with side length 0.4 whose center is at $(0, 0)$.

Change the vertex shader so that the square moves to the right by 0.3.

Change the color of the square to blue.

Draw two triangles but put their data in different buffers in the GPU.

Drawing two triangles with separate buffers

```
...
// set up first buffer
var vertices1 = [ 0.2, 0.4, 0.4, -0.7, 0.6, 0.9];
var vBuffer1 = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer1);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices1), gl.STATIC_DRAW);

//set up second buffer
var vertices2 = [0.1, 0.6, -0.3, 0.9, -0.4, 0.5];
var vBuffer2 = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer2);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices2), gl.STATIC_DRAW);
```

Drawing two triangles with separate buffers

```
// Do shader plumbing
var vPosition = gl.getAttribLocation(program, "vPosition");
gl.enableVertexAttribArray(vPosition);

//Draw first triangle
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer1); //set appropriate buffer as
current
gl.vertexAttribPointer(vPosition, 2, gl.FLOAT, false, 0, 0); // uses
currently bound buffer
gl.drawArrays(gl.TRIANGLES,0,3);

//Draw second triangle
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer2); //set appropriate buffer as
current
gl.vertexAttribPointer(vPosition, 2, gl.FLOAT, false, 0, 0); // uses
currently bound buffer
gl.drawArrays(gl.TRIANGLES,0,3);
```

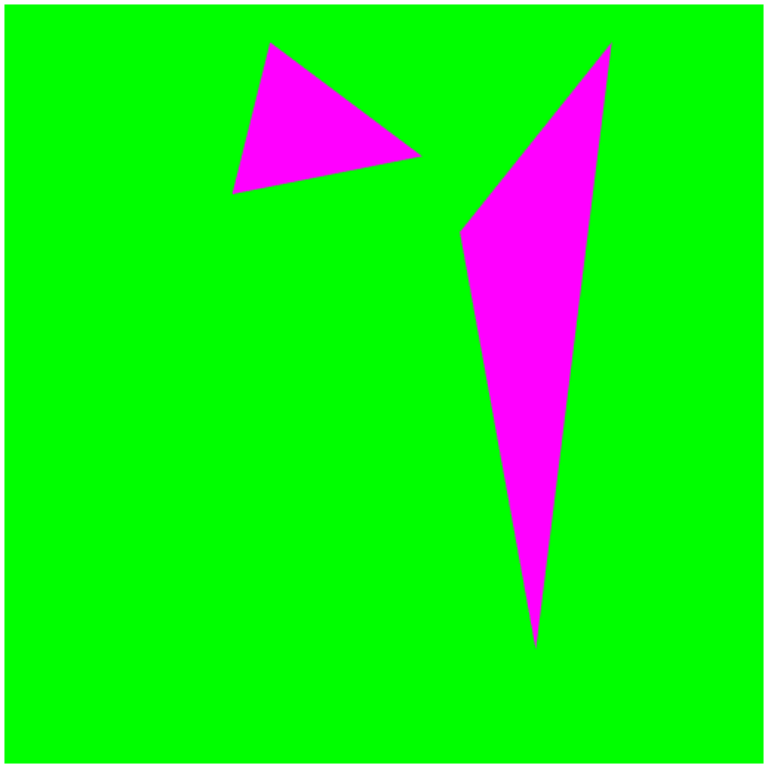
Once per variable!

Once for each buffer!

The function **vertexAttribPointer** *uses the buffer currently bound to ARRAY_BUFFER.*

Drawing two triangles with separate buffers

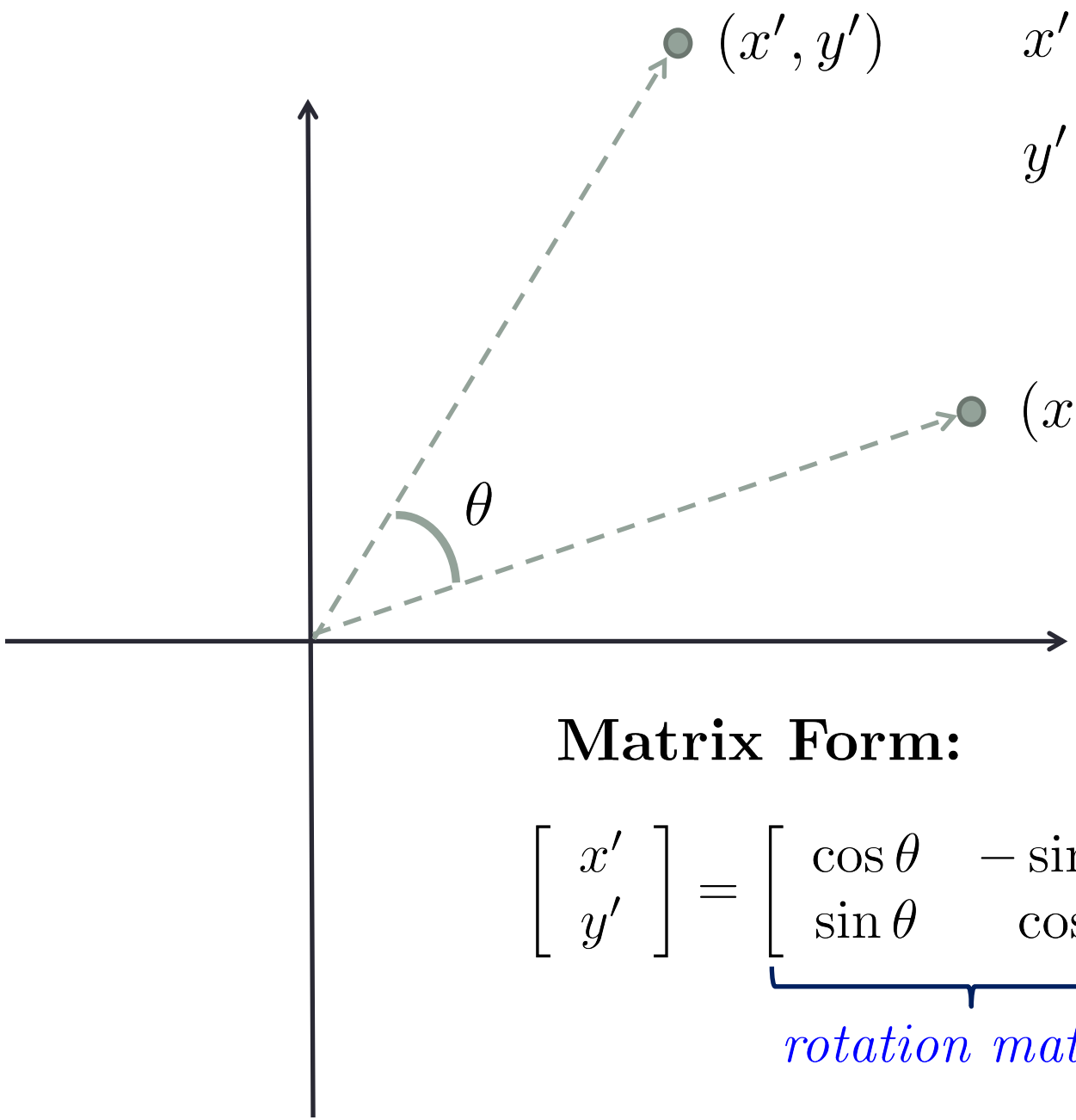
Output:



Next Task:

Change the vertex shader so that the entire picture is 1.5 times the original size and is rotated by 30 degrees in the counter-clockwise direction around the origin.

Rotation by θ (counter-clockwise)



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

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

Matrix Form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}}_{\textit{rotation matrix}} \begin{bmatrix} x \\ y \end{bmatrix}$$

requestAnimationFrame

Typically, what we render on the screen changes with time.

So, what we do is write a function which is called again and again.

However, we don't call this function again and again ourselves.
Rather we request the browser to do it whenever it is free next.

This is done as follows: `requestAnimationFrame(render);`

Note that `requestAnimationFrame` is not a WebGL function.

`requestAnimationFrame` also passes the time, in milliseconds, since the webpage was loaded to the `render` function.

So, we typically write the render function as follows:

```
function render(now) { ... }
```

something that depends on time i.e., 'now'

Rotating Triangle

We created the equilateral triangle earlier. Now, we want to it to rotate continuously around the origin.

Idea: Modify the vertex coordinates depending on the time, and send the updated coordinates to the GPU using `gl.bufferData`.

Better Idea: Send the current time to the GPU and do the time dependent transformations there.

Uniform Variables

Note that *time* is not something that changes from vertex to vertex. Such variables are called `uniform` variables.

We get the location of uniform variables using:

```
gl.getUniformLocation(program, "variableName");
```

We set a single floating point uniform variable by:

```
gl.uniform1f(location, value);
```

Back to coding: Implement the rotating triangle.

How do we change the speed?

How can we rotate it about the y-axis?

Types of Variables in Shaders

attribute variables: information that varies from vertex to vertex
Used in the vertex shader

uniform variables: data that is the same for all vertices and fragments.
Can be used in both vertex and fragment shaders.

varying variables: assigned in vertex shaders, *interpolated* in fragment shaders.
Must be declared in both shaders.

Colorful Rotating Triangle

Modify the the rotating triangle code as follows:

Create an array called `colors` in which we put color information.

```
var colors = [1,0,0, 0,1,0, 0,0,1];
```

Transfer the color data to a new buffer in the GPU.

Create an attribute variable called `vColor` in the vertex shader.

```
attribute vec4 vColor;
```

Create a varying variable called `fColor` in the both shaders.

```
varying vec4 fColor;
```

In the vertex shader: `fColor = vColor`.

In the fragment shader: `gl_FragColor = fColor`.

Code for Colored Rotating Triangle

```
"use strict";

// global variables
var gl;
var vertices, vBuffer;
var colors, cBuffer;
var ut;

window.onload = function init() {
        // Set up WebGL
        var canvas = document.getElementById("gl-canvas");
        gl = WebGLUtils.setupWebGL( canvas );
        if(!gl){alert("WebGL setup failed!");}

        // Set Clear Color
        gl.clearColor(0.0, 0.0, 0.0, 1.0);

        // Load shaders and initialize attribute buffers
        var program = initShaders( gl, "vertex-shader", "fragment-shader" );
        gl.useProgram( program );
        // Load data into buffers
        vertices = [];
        var r =0.7;
        for(var t = 0; t <  2*Math.PI; t+=2*Math.PI/3){
            vertices.push(r*Math.cos(t), r*Math.sin(t));
        }

        vBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);

        colors = [1,0,0, 0,1,0, 0,0,1];
        cBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, cBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
```

Code for Colored Rotating Triangle

```
        // Do shader plumbing
        var vPosition = gl.getAttribLocation(program, "vPosition");
        gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
        gl.vertexAttribPointer(vPosition, 2, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(vPosition);

        var vColor = gl.getAttribLocation(program, "vColor");
        gl.bindBuffer(gl.ARRAY_BUFFER, cBuffer);
        gl.vertexAttribPointer(vColor, 3, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(vColor);

        // Note that the function vertexAttribPointer needs to know
        // the buffer from which the data for the variable (referred
        // to by the first argument) comes from. We therefore need to
        // have the appropriate buffer bound as the current buffer.

        ut = gl.getUniformLocation(program, "t");

        requestAnimationFrame(render);
};


function render(now){
        requestAnimationFrame(render);

        var t = 0.001*now;
        gl.uniform1f(ut,t);
        gl.clear(gl.COLOR_BUFFER_BIT);
        gl.drawArrays(gl.TRIANGLES,0,3);
}
```

Code for Colored Rotating Triangle

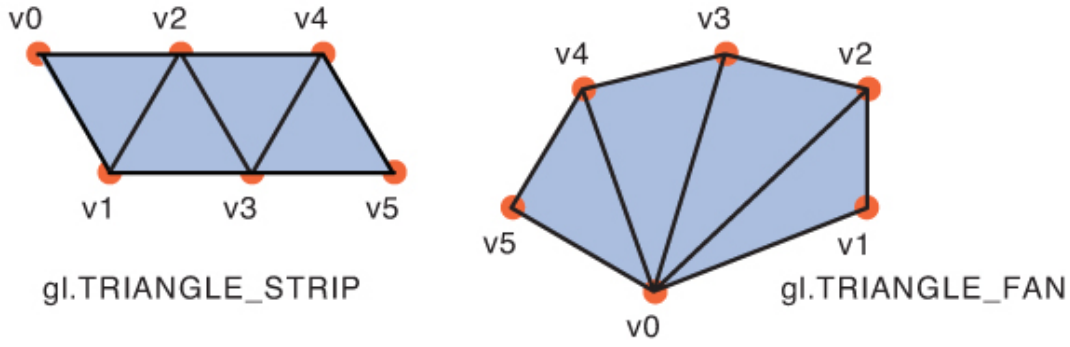
```
<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 fColor;
uniform float t;
void main(){
        float x = vPosition.x;
        float y = vPosition.y;
        float x1 = x*cos(t) - y*sin(t);
        float y1 = x*sin(t) + y*cos(t);
        gl_Position = vec4(x1,y1,0,1);
        fColor = vColor;
}
</script>

<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;
varying vec4 fColor;
void main(){
    gl_FragColor = fColor;
}
</script>
```

Back to Square One

We created our square using two triangles.

We had to repeat coordinates of the two shared vertices.

Two ways to avoid it:



A more general way: index buffers (later).

Adding Interaction

WebGL does not support interaction directly. Reason: Portability.

Interaction is done using Javascript and HTML.

An ***event driven model*** is used. Events occur and we respond to them via functions called ***event handlers***.

Examples of events: mouse click, key press, mouse movement

Clicking of a button on the screen and loading a page are also events.

We have been using an event handler all along. Where?

```
window.onload = function init() { ...  }
```

Adding Interaction

In the colored rotating triangle program, we want add a button to flip the direction of rotation and a slider to control speed.

Step 1: Create a button and a slider using HTML.

```
<body>
   <canvas id="gl-canvas" width="512" height="512">
    HTML5 Canvas not supported!
   </canvas>

   <button id="Flip Button"> Flip Direction </button>

   <input id="Speed Slider" type="range"
     min="0", max="10", step="0.1" > Speed Control
   </input>
                  ...
</body>
```

Adding Interaction

Step 2: Add event listeners using Javascript

```
var direction = 1;
var speed = 5;

window.onload = function init() {
                    ...
        // Button
        var button = document.getElementById("Flip Button");
        button.onclick = function(){direction*= -1;};

        //Slider
        var slider = document.getElementById("Speed Slider");
        slider.onchange = function(){ speed = event.srcElement.value;};
                    ...
};

function render(now){
            requestAnimationFrame(render);

            gl.uniform1f(ut, direction*speed*0.001*now);

            // Clear canvas and draw
            gl.clear(gl.COLOR_BUFFER_BIT);
            gl.drawArrays(gl.TRIANGLES,0,3);
}
```

two global variables

Where did we define this object?
It is the `window.event` *object.*

Adding Interaction

Two equivalent ways for adding event listeners:

```
// Button
var button = document.getElementById("Flip Button");
button.onclick = function(){direction*= -1;};


// Alternate form:
button.addEventListener("click", function(){direction*= -1;});
```

Adding Interaction

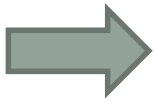
We can also pass the `event` object as a parameter to the event handler. This is in fact the preferred way.

```
//Mouse click on canvas
canvas.addEventListener("click", function(event){
   console.log( "You have clicked position: ("
            + event.offsetX + "," + event.offsetY + ")" );

});
```

`event.offsetX, event.offsetY` : offsets in pixels from top left corner

Offsets

(x_o, y_o)



Normalized Device Coordinates

$$(-1 + \frac{2x_o}{w}, -1 + \frac{2(h-y_o)}{h})$$

Here w and h are the canvas width and height respectively in pixels.

Adding Interaction

```
function getMousePosition(canvas, event){
    return {
          x: -1+2*event.offsetX/canvas.width,
          y: -1+2*(canvas.height- event.offsetY)/canvas.height
    };
}


//Mouse click on canvas
canvas.addEventListener("click", function(event){
   console.log( "You have clicked position: ("
          + event.offsetX + "," + event.offsetY + ")" );

   var pos = getMousePosition(canvas, event);
   console.log("NDC Coordinates:" + pos.x + "," + pos.y);

});
```

Adding Interaction

Keyboard input:

```
// Key press in the window
window.addEventListener("keydown", function(event){

    var key = String.fromCharCode(event.keyCode);
    alert("You pressed: " + key);

});
```

For special keys, you need to check the key code.