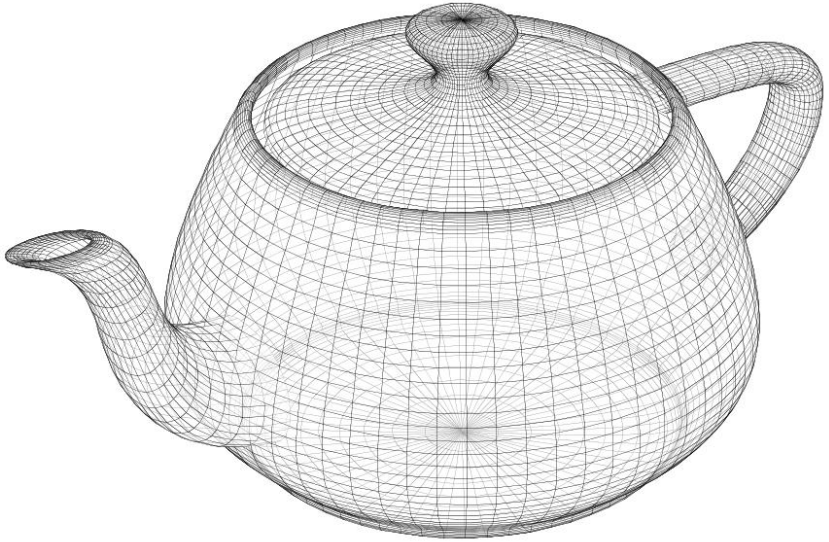
Animation in WebGL

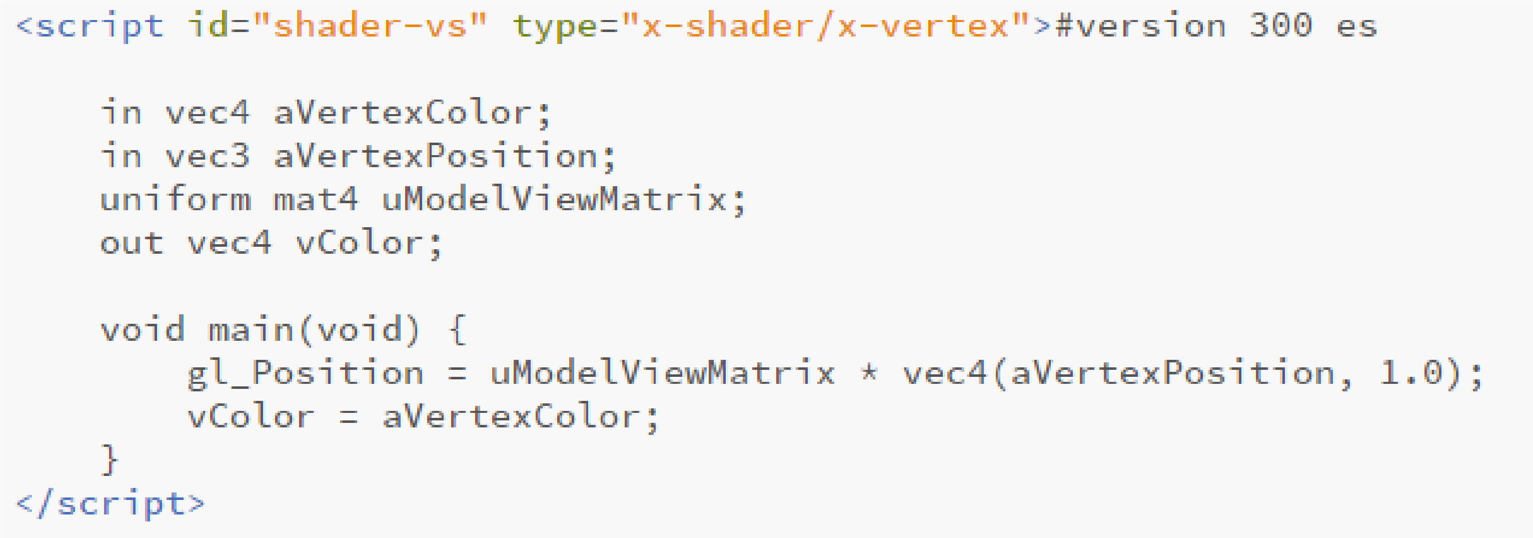


CS 418: Interactive Computer Graphics

Eric Shaffer

ILLINOIS

Transforming Geometry in the Vertex Shader



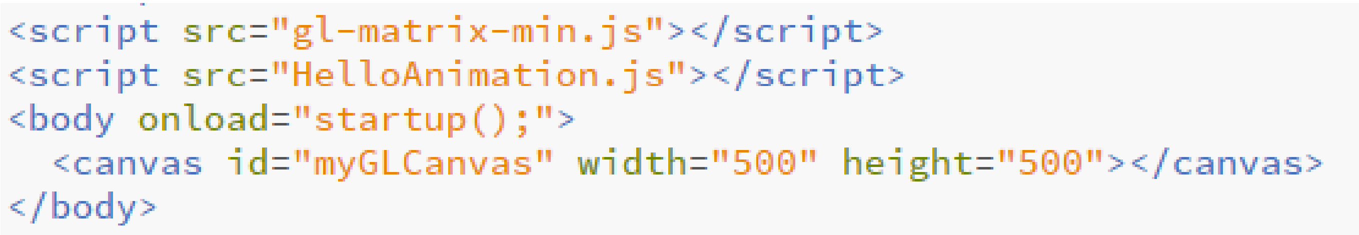
```
<script id="shader-vs" type="x-shader/x-vertex">#version 300 es

    in vec4 aVertexColor;
    in vec3 aVertexPosition;
    uniform mat4 uModelViewMatrix;
    out vec4 vColor;

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

We can use the 3D transformations we’ve learned to alter geometry in the the vertex shader.

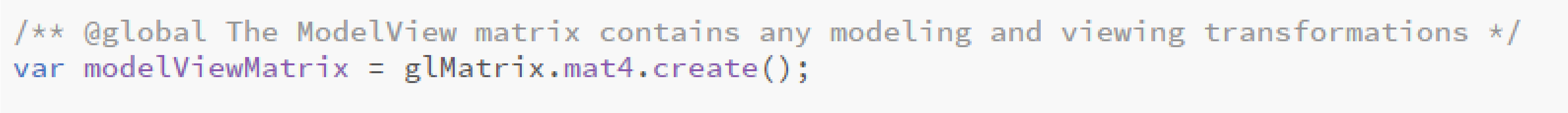
A Uniform variable is one that is the same for all the vertices being processed by the shader.

In the code at the left, we use Uniform 4x4 matrix to transform the coordinates of each vertex.

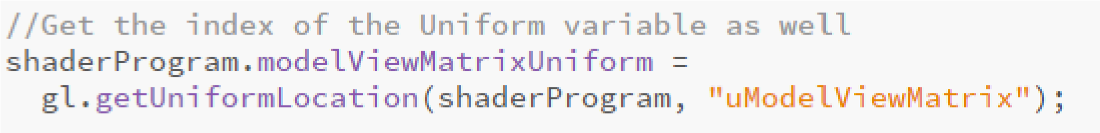
Getting the Transformation to the Shader



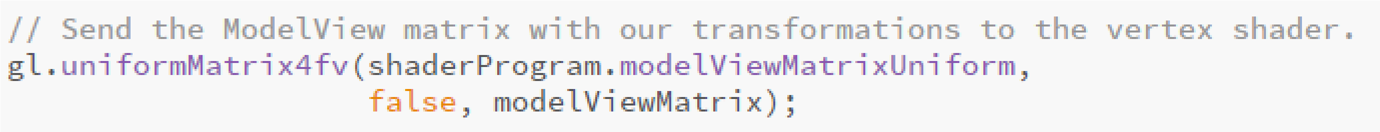
```
<script src="gl-matrix-min.js"></script>
<script src="HelloAnimation.js"></script>
<body onload="startup();">
  <canvas id="myGLCanvas" width="500" height="500"></canvas>
</body>
```



```
/** @global The ModelView matrix contains any modeling and viewing transformations */
var modelViewMatrix = glMatrix.mat4.create();
```



```
//Get the index of the Uniform variable as well
shaderProgram.modelViewMatrixUniform =
  gl.getUniformLocation(shaderProgram, "uModelViewMatrix");
```



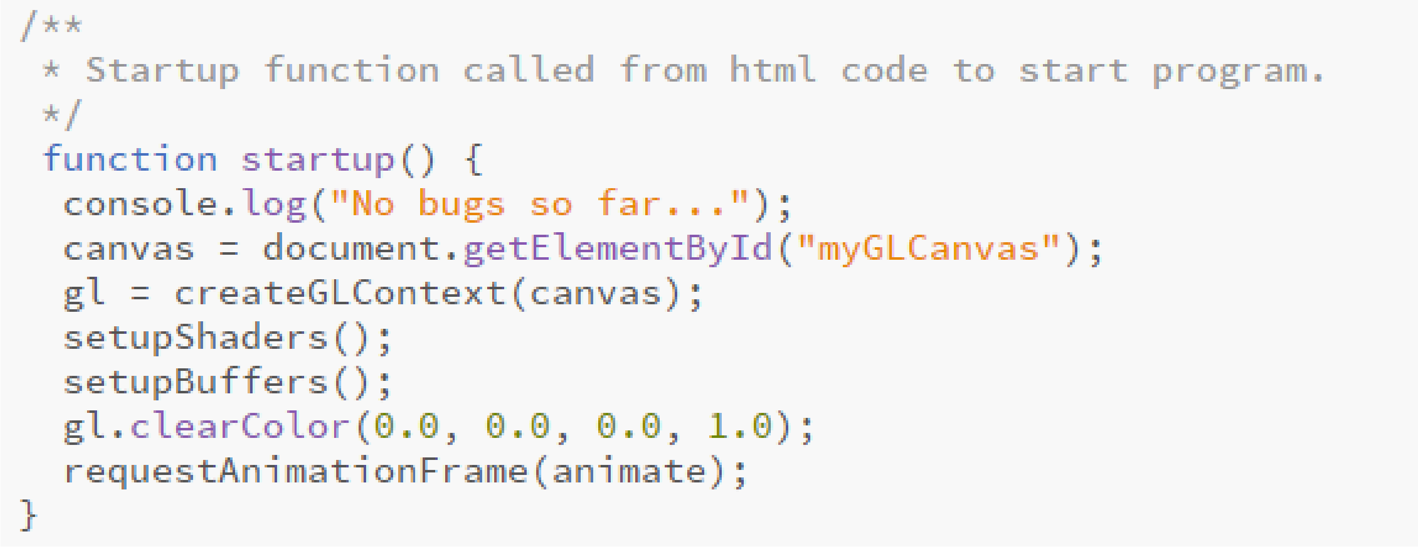
```
// Send the ModelView matrix with our transformations to the vertex shader.
gl.uniformMatrix4fv(shaderProgram.modelViewMatrixUniform,
                    false, modelViewMatrix);
```

We will use the glMatrix library for vector and matrix math inside JavaScript code.

There are several steps to using the library and to create a matrix to be sent to the shader.

1. Include the library in the HTML file using <script> tags
2. We create a matrix
3. When we initialize the shader, we get a handle for the Uniform matrix in the shader
4. In the draw() function, we use the handle to send it to the shader

Requesting an Animation Frame



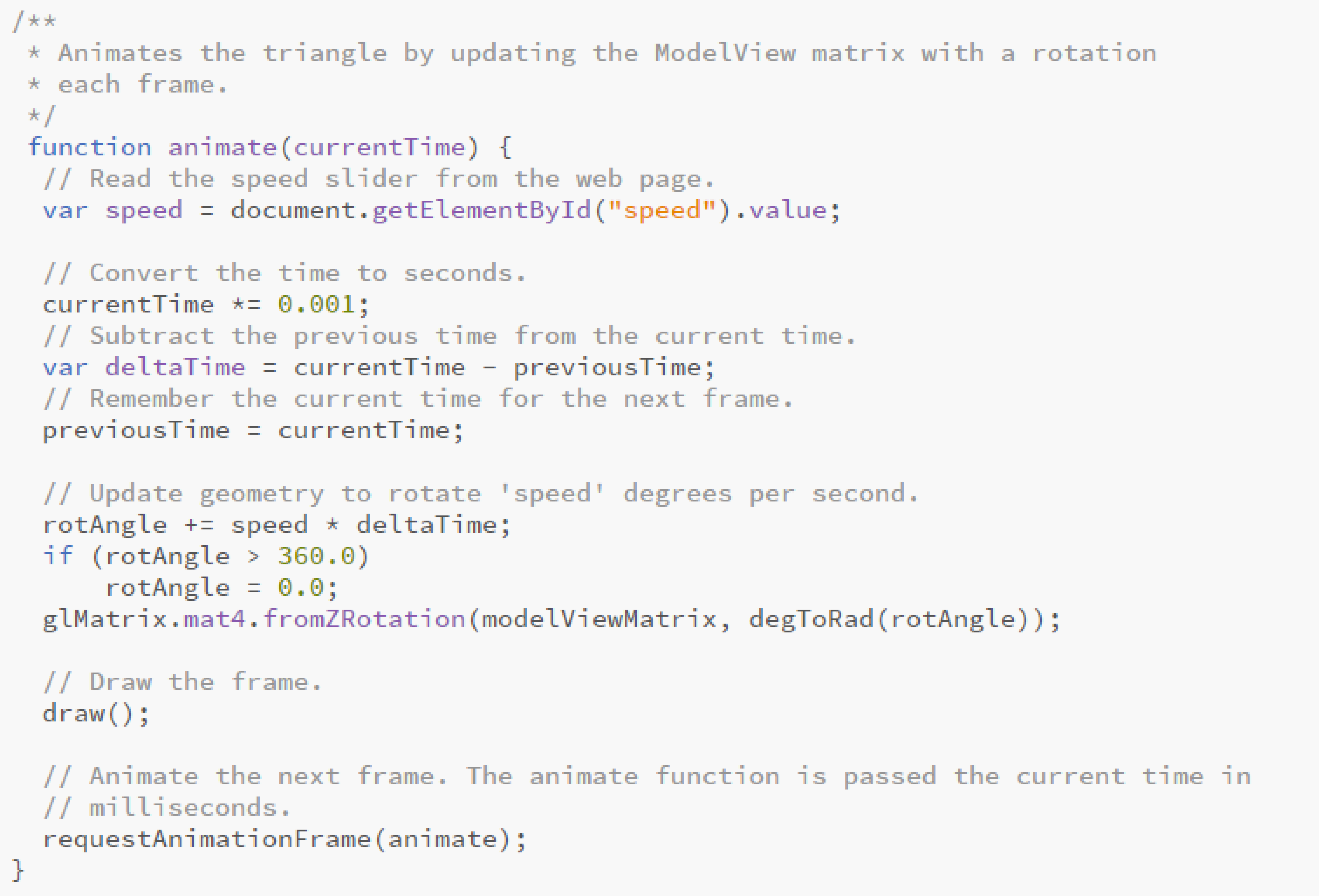
```
/**
 * Startup function called from html code to start program.
 */
 function startup() {
  console.log("No bugs so far...");
  canvas = document.getElementById("myGLCanvas");
  gl = createGLContext(canvas);
  setupShaders();
  setupBuffers();
  gl.clearColor(0.0, 0.0, 0.0, 1.0);
  requestAnimationFrame(animate);
}
```

`requestAnimationFrame`
is a function that tells the broswer you want to animate and gives it a function to call before the the next repaint.

The number of callbacks is usually 60 times per second, but will generally match the display refresh rate.

https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

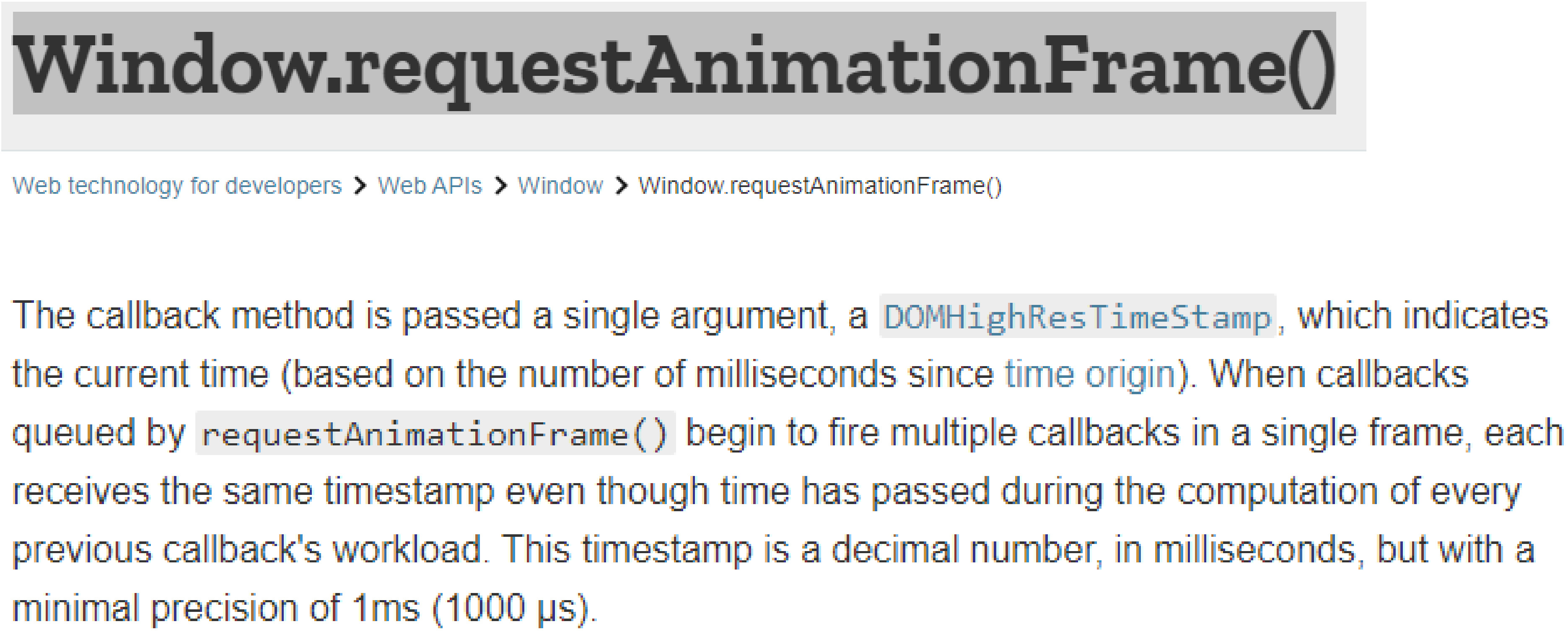
Animation



```
/**
 * Animates the triangle by updating the ModelView matrix with a rotation
 * each frame.
 */
 function animate(currentTime) {
  // Read the speed slider from the web page.
  var speed = document.getElementById("speed").value;

  // Convert the time to seconds.
  currentTime *= 0.001;
  // Subtract the previous time from the current time.
  var deltaTime = currentTime - previousTime;
  // Remember the current time for the next frame.
  previousTime = currentTime;

  // Update geometry to rotate 'speed' degrees per second.
  rotAngle += speed * deltaTime;
  if (rotAngle > 360.0)
      rotAngle = 0.0;
  glMatrix.mat4.fromZRotation(modelViewMatrix, degToRad(rotAngle));

  // Draw the frame.
  draw();

  // Animate the next frame. The animate function is passed the current time in
  // milliseconds.
  requestAnimationFrame(animate);
}
```

With each frame, we update a global variable that is used to set the rotation matrix sent to the shader.

requestAnimationFrame() and Time

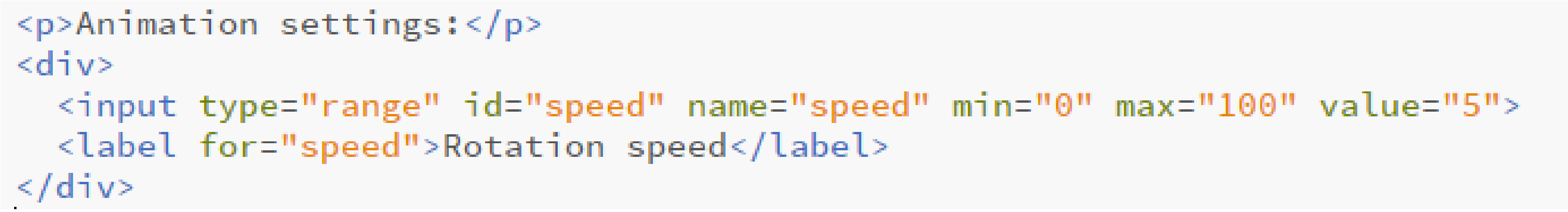


Window.requestAnimationFrame()

Web technology for developers > Web APIs > Window > Window.requestAnimationFrame()

The callback method is passed a single argument, a `DOMHighResTimeStamp`, which indicates the current time (based on the number of milliseconds since time origin). When callbacks queued by `requestAnimationFrame()` begin to fire multiple callbacks in a single frame, each receives the same timestamp even though time has passed during the computation of every previous callback's workload. This timestamp is a decimal number, in milliseconds, but with a minimal precision of 1ms (1000 µs).

The range control in HTML



```
<p>Animation settings:</p>
<div>
  <input type="range" id="speed" name="speed" min="0" max="100" value="5">
  <label for="speed">Rotation speed</label>
</div>
```

- Creates a slider
- Values are 0 to 100
- We can access current value in JS code in `animate` function

- Can learn more about adding elements for user interaction:

  https://developer.mozilla.org/en-US/docs/Learn/Forms

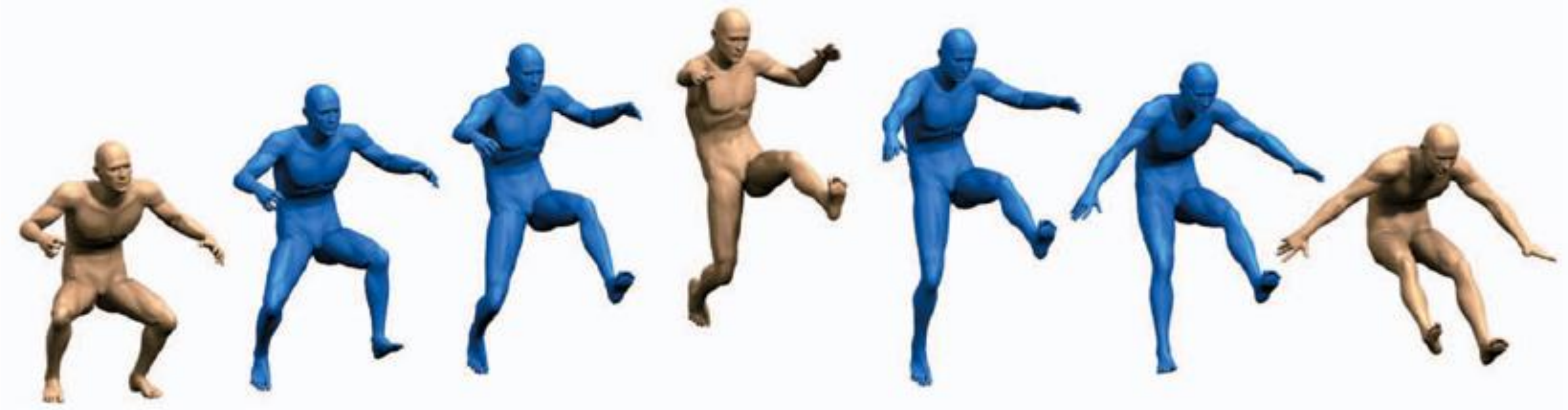
Using Wall Clock Time

How can the elapsed time variable be useful?

You can base your transformations off of elapsed time instead of frame count

Framerates can vary…for things like game physics time can provide a better experience

Interpolation and Key Frames



In animation a **key frame** is a typically an artist generated geometry

Intermediate frames can be calculated using interpolation

Linear Interpolation

Probably most common mathematical operation in computer graphics

Affectionately referred to as ***lerp***

Given two points A and B, lerp generates intermediate positions on a straight line

$$lerp(A, B, t) = (1 - t)A + tB = A + t(B - A)$$

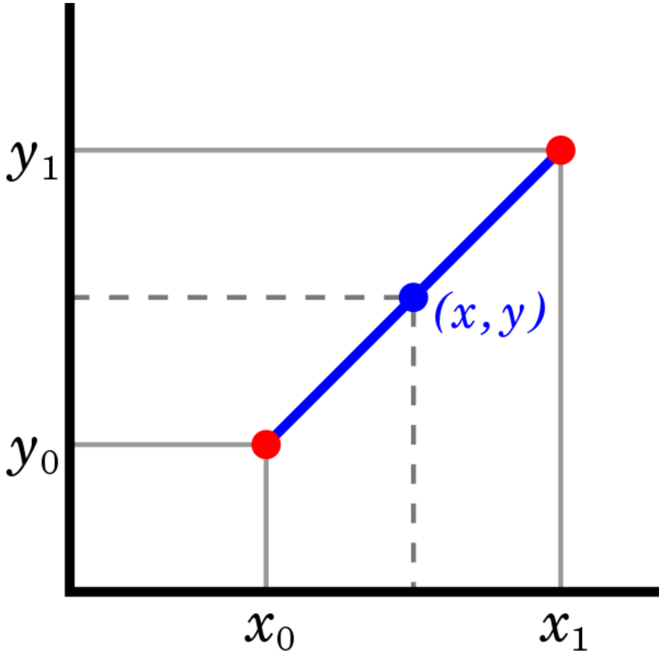
This is a **parametric equation**. The parameter is the variable **t**.

Think of ***t*** as time. At time 0, where are we? How about at time 1?

Linear Interpolation

If we wanted to code up lerp for 2D points, how do you do it?

```
function lerp(Out, A, B, t){
        Out[0] = A[0] + (B[0]-A[0])*t;
        Out[1] = A[1] + (B[1]-A[1])*t;
}
```



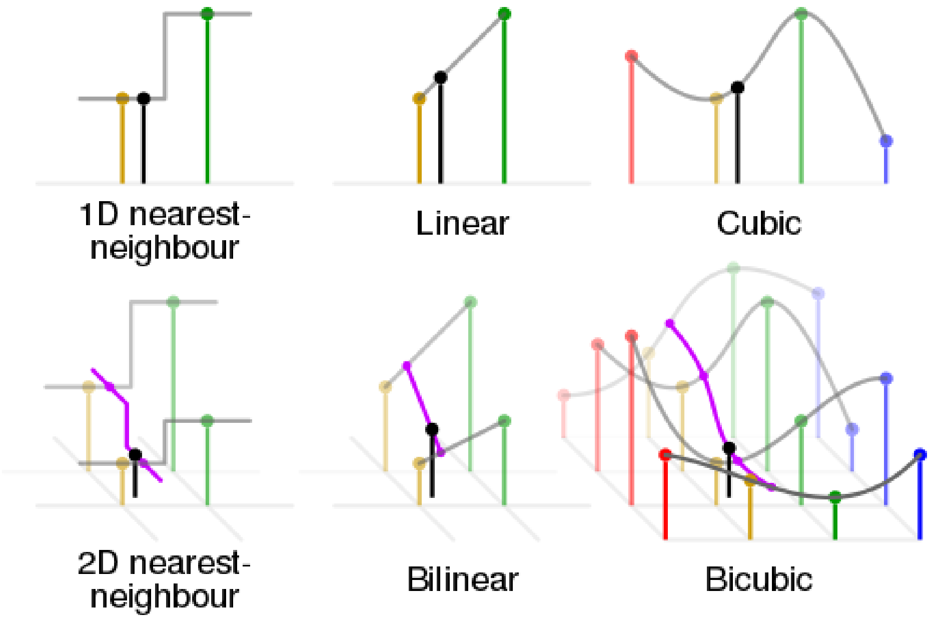
For 3D, just add a line to compute Out[2]

Maybe obvious here…
but keep in mind that more complicated math
that we will see usually generalizes to 3D similarly

Linear Interpolation

Can also be thought of as reconstructing a function from sample points

We will see some other ways of doing this as well…


1D nearest-neighbour
Linear
Cubic
2D nearest-neighbour
Bilinear
Bicubic