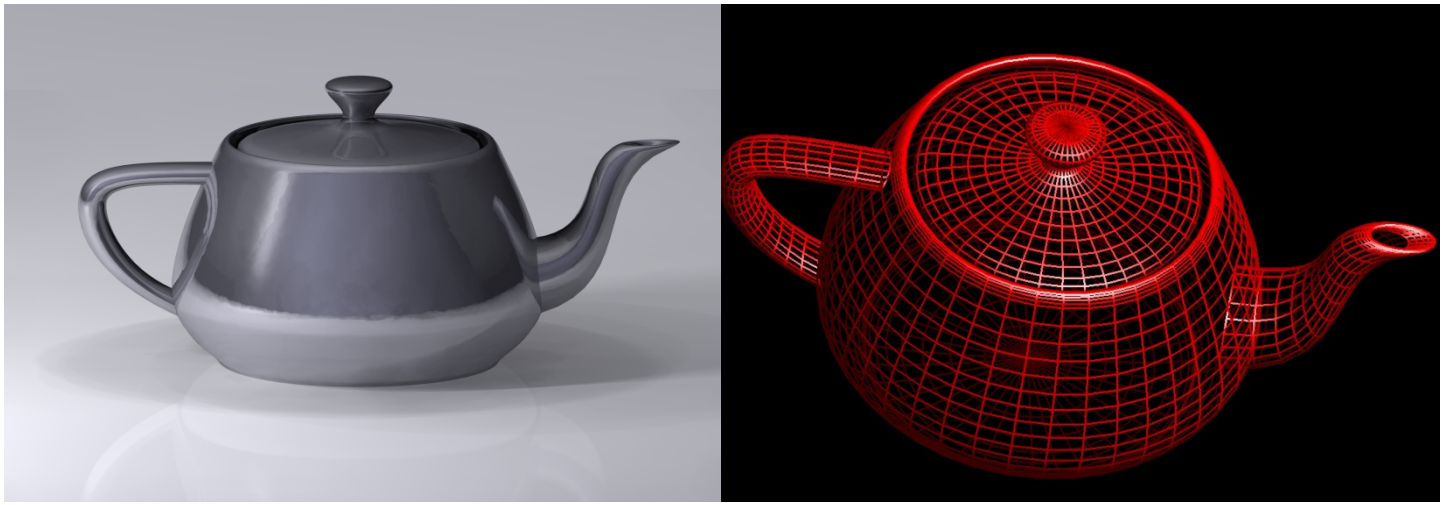


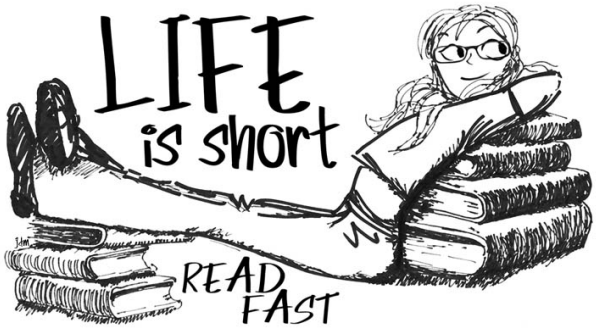
Foundations of Computer Graphics

Saurabh Ray

Reading

Reading for Lecture 4: Sections 3.6-3.10, 4.1.

Reading for Lecture 5: Section 4.3.



Code on github: https://github.abudhabi.nyu.edu/sr194/CG-2020

Resources for Learning Javascript.

Slides in NYU Classes Resources/Reading.

https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.pdf

Learn to use a Javascript debugger.

Install webgl inspector and learn to use it.

not urgent

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

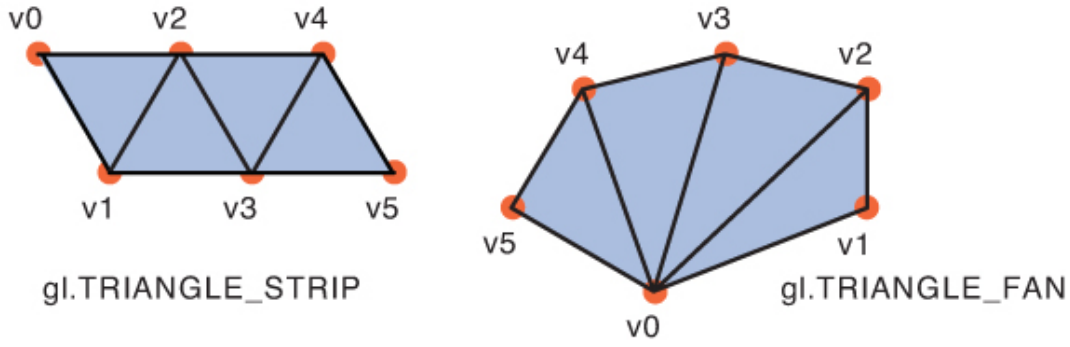
```
<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 fColor;
uniform float t;
void main(){
        float x = vPosition.x;
        float y = vPosition.y;
        float x1 = x*cos(t) - y*sin(t);
        float y1 = x*sin(t) + y*cos(t);
 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

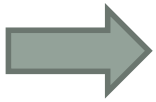
We can also pass the `event` object as a parameter to the event handler. This is in fact the preferred way.

```
//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);

});
```

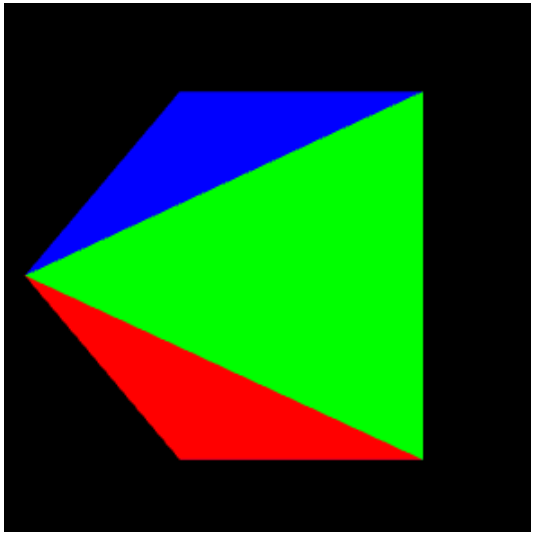
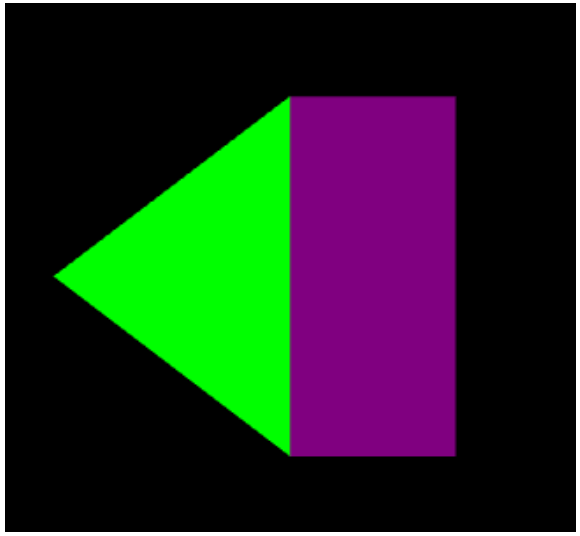
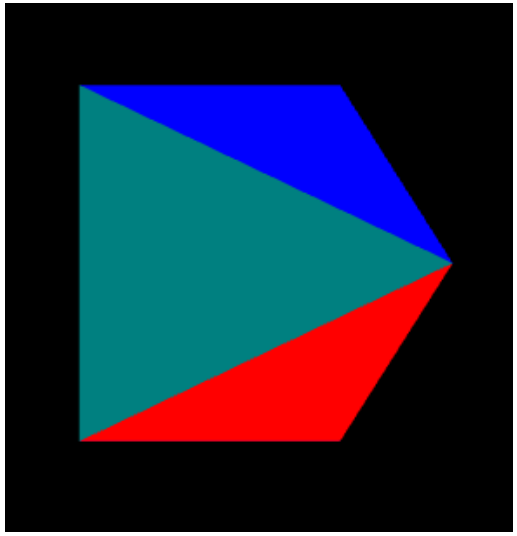
Adding Interaction

Keyboard input:

```
window.addEventListener("keydown", function(event){
    var code = event.keyCode;
    var key = String.fromCharCode(code);
    console.log("You pressed " + key + ", key code: ", +code );
});
```

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

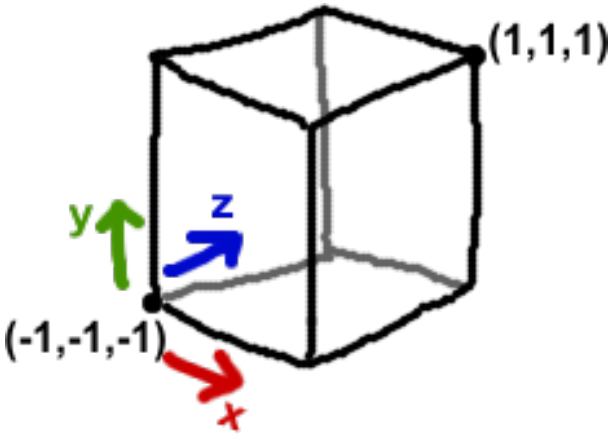
3D Objects

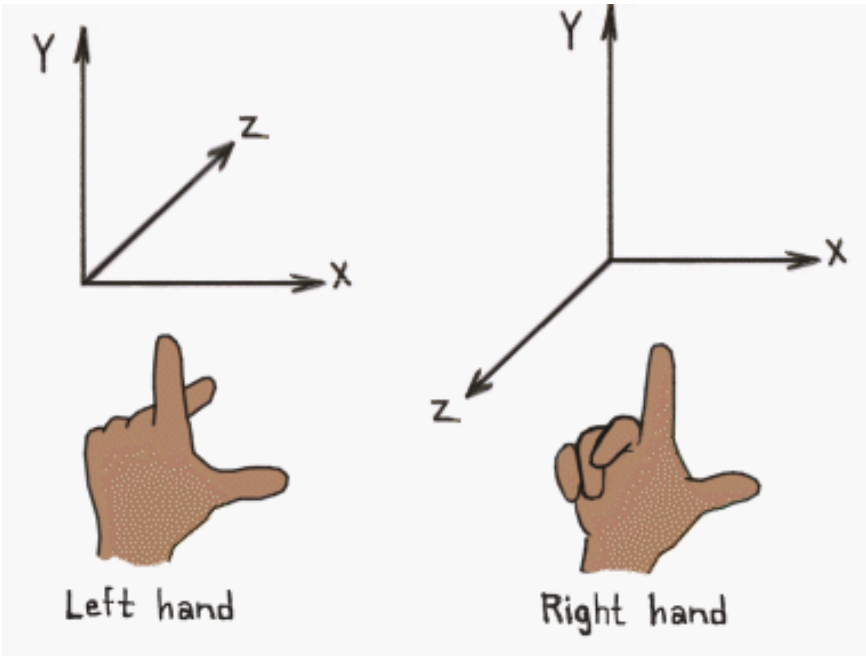




Pyramid with a square base

WebGL Coordinate System

Left Handed Coordinate system!





In mathematics, right handed coordinate system is the convention.

Pyramid with a square base

Vertex position data:

```
var s = 0.4;
var a = vec3(-s,-s,s);
var b = vec3(s,-s,s);
var c = vec3(s,s,s);
var d = vec3(-s,s,s);
var e = vec3(0,0,-s);
var vertices = [a,b,e,b,c,e,c,d,e,d,a,e,a,b,c,a,c,d];
```

`vec3` is a function defined in `MV.js`. Returns an array of length 3.

`vec2` and `vec4` are similar functions defined in the file.

`vertices` is now an array of arrays.

To get a C-type array, we use the function `flatten` also defined in `MV.js`

```
var vBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
gl.bufferData(gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW);
```

Pyramid with a square base

Colors data:

```
var R = vec3(1,0,0);
var G = vec3(0,1,0);
var B = vec3(0,0,1);
var X = vec3(0.0,0.5,0.5);
var Y = vec3(0.5, 0, 0.5);
var colors = [R,R,R,G,G,G,B,B,B,X,X,X,Y,Y,Y,Y,Y,Y];


var cBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, cBuffer);
gl.bufferData(gl.ARRAY_BUFFER, flatten(colors), gl.STATIC_DRAW);
```

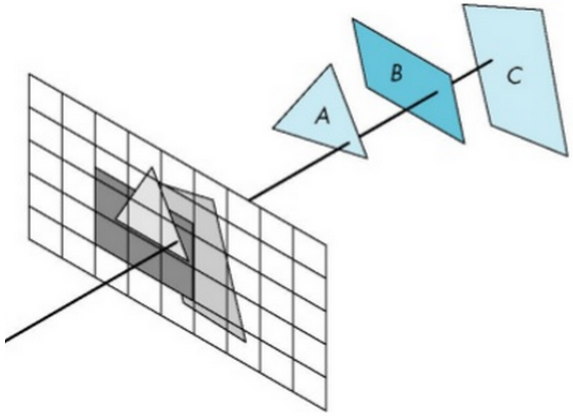
Visibility

So far, we are just drawing a bunch of triangles.

How do we know what is visible and what is not?

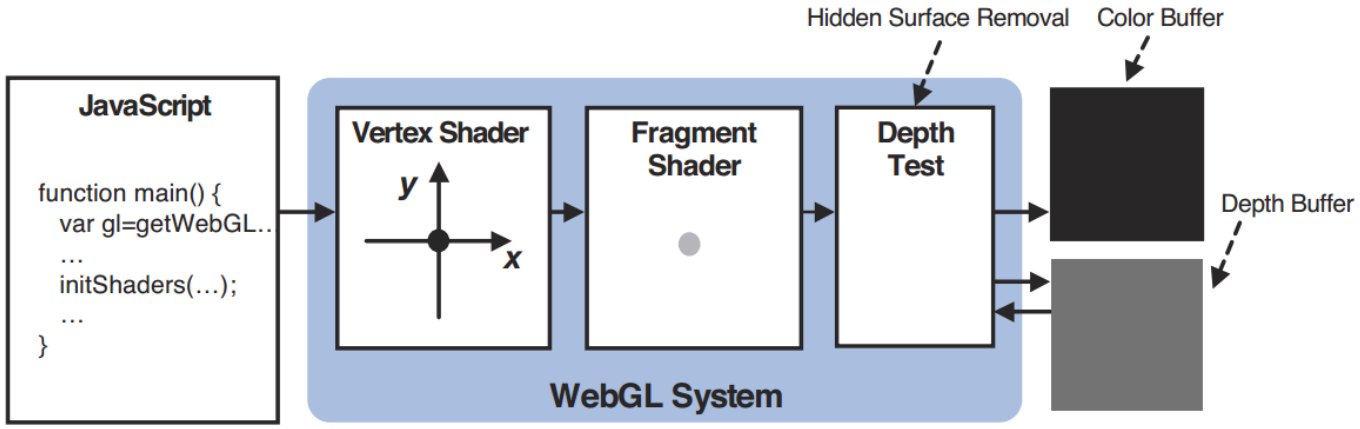
Ans: by looking at the z-coordinates

A fragment with a larger z value is farther from us than a fragment with a smaller z value.



We write the color of a fragment into the color buffer, only if its depth is smaller than the current value in the depth buffer.

Visibility



How do we know the z-coordinate of a fragment?

Ans: Interpolation

The depth of a fragment used by WebGL is not exactly its z-coordinate. It is an increasing function of the z-coordinate and lies in the range $[0, 1]$.

Depth buffering is enabled in WebGL by: `gl.enable(gl.DEPTH_TEST);`

We also need to clear the depth buffer before drawing:

```
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
```

Points and Vectors

Point: a location
a fixed place in the geometric world
Example: South Pole

Vector: direction with magnitude
think of it as displacement or motion between two points
Example: 2km North

Points and Vectors are two different concepts.

Operations on Points and Vectors

Vector + Vector?	2km North + 2km East = ?	Ans: $2\sqrt{2}$ km North-East
Point + Vector?	Abu Dhabi + 100km North-East $\approx$ Dubai	
Vector + Point?	Same as Point + Vector	
Point + Point?	Abu Dhabi + Dubai = ?	*Nonsense*
Point − Point?	Dubai − Abu Dhabi $\approx$ 100km North-East	
Point − Vector?	Dubai − 100km North-East $\approx$ Abu Dhabi − 100km North-East = 100 km South-West	
Vector − Point?	*Nonsense*	

Operations on Points and Vectors

$2\times$ Vector? $2\times$ 1 km East = 2 km East

-3 $\times$ Vector? -3 $\times$ 1 km East = 3 km West

You can multiply a vector with any real number.

Can you multiply a point by a number? What is $2\times$ Abu Dhabi ?

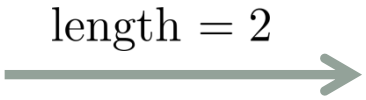
Nonsense

Representing Vectors by Arrows

Recall that vectors have a direction and a magnitude.

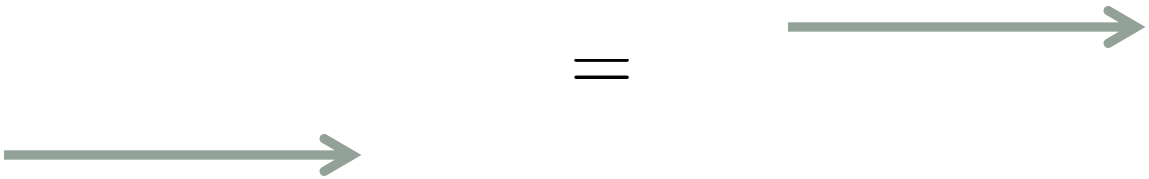
We can therefore represent a vector with an arrow s.t.

- the direction of the arrow shows the direction of the vector
- the length of the arrow shows the magnitude

2km East: 

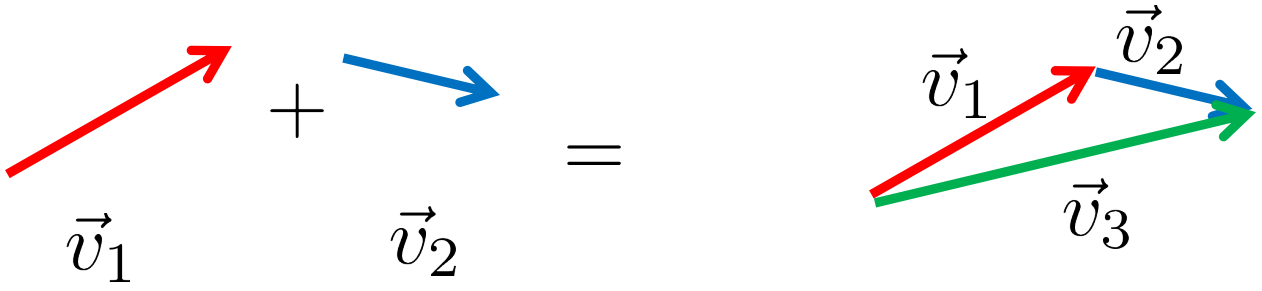
Note that the starting point of the arrow does not matter.

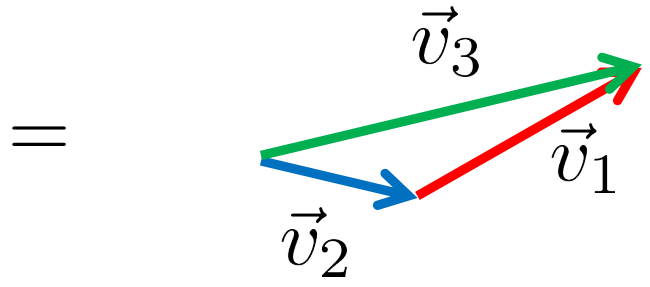
Only its direction and length matter.



Adding Vectors

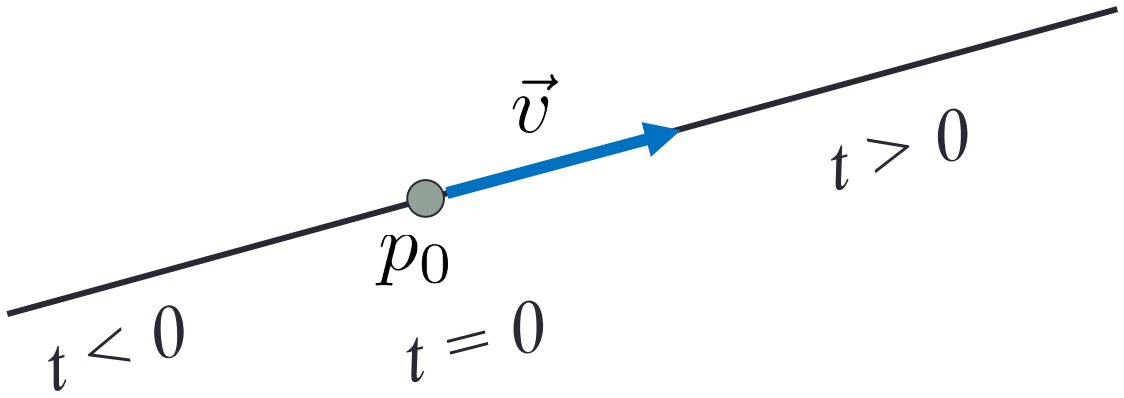
How do you add vectors?





Parametric equation of a line

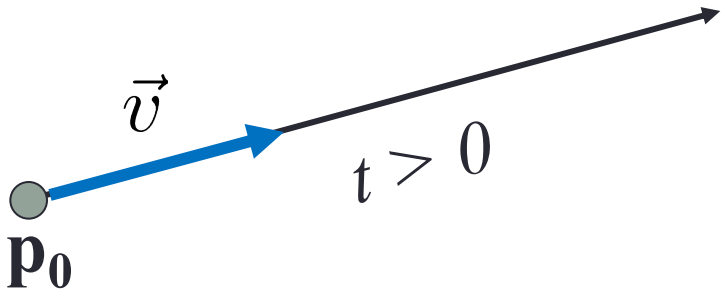
To define a line, we need a point p_0 on it and a vector $\vec{v}$ along it.



$$\ell(t) = p_0 + t\vec{v}, \qquad t \in (-\infty, +\infty)$$

This works in any dimension!

Parametric equation of a ray



$$\ell(t) = p_0 + t\vec{v}, \qquad t \in [0, +\infty)$$