WebGL

- HTML5 + WebGL
- Javascript + GLSL ES 2.0 (OpenGL Shading Language)
- Vertex and Fragment Shaders

Dividing Work

- HTML5:
  - performs interactive tasks
  - event queue
  - callback functions
- WebGL:
  - 3D graphics
  - graphics pipeline

example1.html

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" >
<title>Example 1</title>


<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;

void
main()
{
    gl_Position = vPosition;
}
</script>

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

void
main()
{
    gl_FragColor = vec4( 1.0, 0.0, 0.0, 1.0 );
}
</script>
```

example1.html

```
<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="example1.js"></script>
</head>

<body>
<div>
recursive steps  0 <input id="slider" type="range"
 min="0" max="5" step="1" value="0" />
5
</div>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```

example1.js

```
var canvas;
var gl;

var points = [];

var numTimesToSubdivide = 0;

var bufferId;

function init()
{
    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    //
    //  Initialize our data for the Sierpinski Gasket
    //

    // First, initialize the corners of our gasket with three points.

    //
    //  Configure WebGL
    //
    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );

    //  Load shaders and initialize attribute buffers

    var program = initShaders( gl, "vertex-shader", "fragment-shader" );
    gl.useProgram( program );
```

example1.js

```
    // Allocate space for the data that goes to the GPU

    bufferId = gl.createBuffer();
    gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
    gl.bufferData( gl.ARRAY_BUFFER, 8*Math.pow(3, 11), gl.STATIC_DRAW );



    // Associate shader variables with our data buffer

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

    document.getElementById("slider").onchange = function(event) {
        numTimesToSubdivide = event.target.value;
        render();
    };


    render();
};
```

example1.js

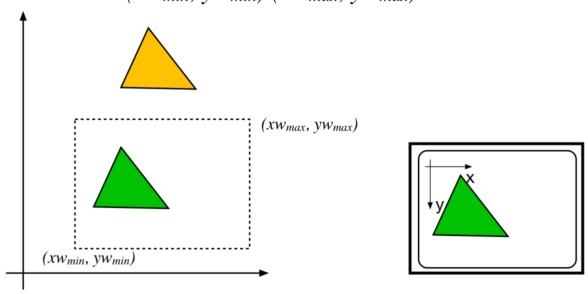
```
window.onload = init;

function render()
{
    var vertices = [
        vec2( -1, -1 ),
        vec2(  0,  1 ),
        vec2(  1, -1 )
    ];
    points = [];
    divideTriangle( vertices[0], vertices[1], vertices[2],
                    numTimesToSubdivide);

    gl.bufferSubData(gl.ARRAY_BUFFER, 0, flatten(points));
    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.drawArrays( gl.TRIANGLES, 0, points.length );
    points = [];
    //requestAnimFrame(render);
}
```

example1.js

```
function triangle( a, b, c )
{
    points.push( a, b, c );
}

function divideTriangle( a, b, c, count )
{

    // check for end of recursion

    if ( count == 0 ) {
        triangle( a, b, c );
    }
    else {

        //bisect the sides

        var ab = mix( a, b, 0.5 );
        var ac = mix( a, c, 0.5 );
        var bc = mix( b, c, 0.5 );

        --count;

        // three new triangles

        divideTriangle( a, ab, ac, count );
        divideTriangle( c, ac, bc, count );
        divideTriangle( b, bc, ab, count );
    }
}
```

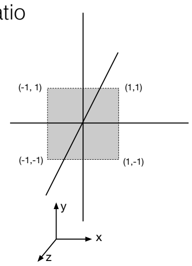
Window

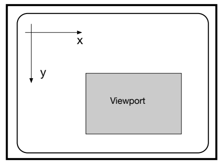
- what part of infinite world is visible on the screen?
- window: (xw_{min}, yw_{min}) (xw_{max}, yw_{max})



Viewport

- Viewport specifies where on the screen the window will appear
- `gl.viewport(x, y, width, height)`
- Aspect ratio





Window -> Viewport

- window: (xw_{min}, yw_{min}) (xw_{max}, yw_{max})
- viewport: (xv_{min}, yv_{min}) (xv_{max}, yv_{max})
- transform from world to screen: (x_w, y_w) -> (x_v, y_v)

$$x_v = xv_{\min} + \frac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}}(x_w - xw_{\min})$$

- can be simplified to:

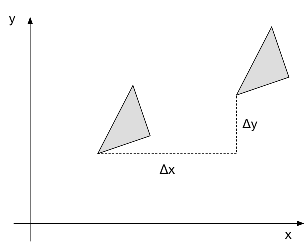
$$x_v = \alpha x_w + \beta$$

Transformations

- can perform geometric transformations on graphics primitives

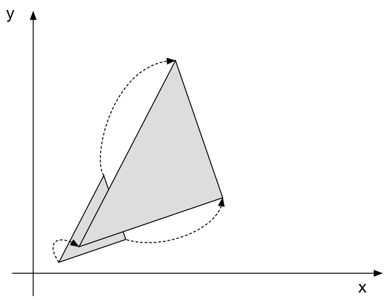
Translation

$$x' = x + \Delta x$$
$$y' = y + \Delta y$$



Scale (about origin)

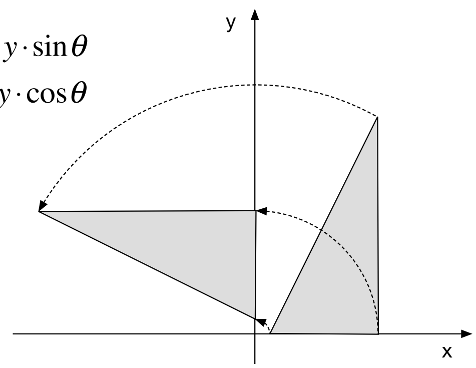
$$x' = s_x \cdot x$$
$$y' = s_y \cdot y$$



- what if s_x less than 1?
- what if s_x is negative?

Rotation (about origin)

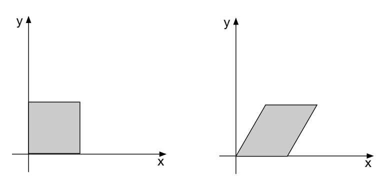
$$x' = x \cdot \cos\theta - y \cdot \sin\theta$$
$$y' = x \cdot \sin\theta + y \cdot \cos\theta$$



Shear

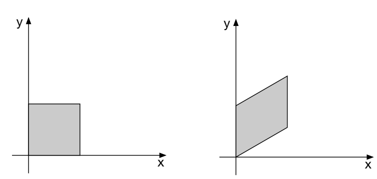
Along x:

$$x' = x + \alpha \cdot y$$
$$y' = y$$



Along y:

$$x' = x$$
$$y' = \beta \cdot x + y$$



Want Matrix Representation

why?

$$\left(TM_3\left(TM_2\left(TM_1\right)\right)\right)\begin{bmatrix} x \\ y \end{bmatrix} = (TM_3 TM_2 TM_1)\begin{bmatrix} x \\ y \end{bmatrix} = TM_{combined}\begin{bmatrix} x \\ y \end{bmatrix}$$

Using Matrices

Scaling

$$x' = s_x \cdot x \qquad y' = s_y \cdot y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}$$

Rotation

$$x' = x \cdot \cos\theta - y \cdot \sin\theta \qquad y' = x \cdot \sin\theta + y \cdot \cos\theta$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}$$

Using Matrices

Translation?

$$x' = x + \Delta x \qquad y' = y + \Delta y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}$$

- can't do it :-(
- want:

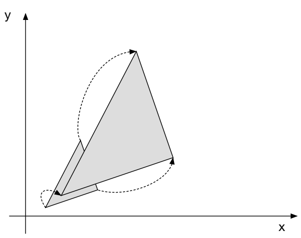
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

Homogeneous Coordinates

- want to represent all transformations with a matrix
- P(x, y) ⇔ P(w·x, w·y, w), w ≠ 0
- i.e. go up 1 more dimension
- we can always go back by dividing by w
- let's use w = 1
- eg. P(3, 4) ⇔ P(3, 4, 1)

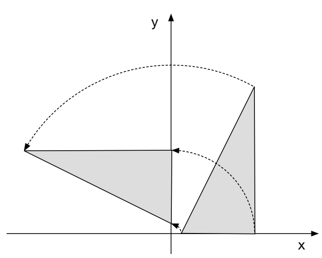
Scaling

$$x' = s_x \cdot x$$
$$y' = s_y \cdot y$$



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation (about origin)

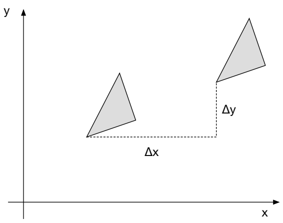


$$x' = x \cdot \cos\theta - y \cdot \sin\theta$$
$$y' = x \cdot \sin\theta + y \cdot \cos\theta$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translation

$$x' = x + \Delta x$$
$$y' = y + \Delta y$$



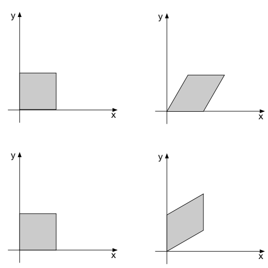
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shear

Along x: $x' = x + \alpha \cdot y$

$y' = y$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \alpha & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



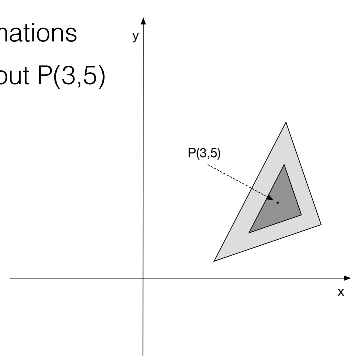
Along y: $x' = x$

$y' = \beta \cdot x + y$

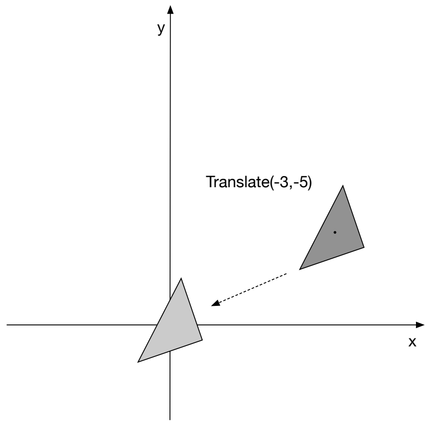
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \beta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scale/Rotate about P(x,y)

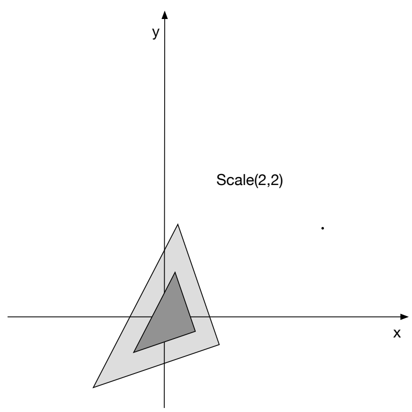
- series of transformations
- eg: Scale(2,2) about P(3,5)



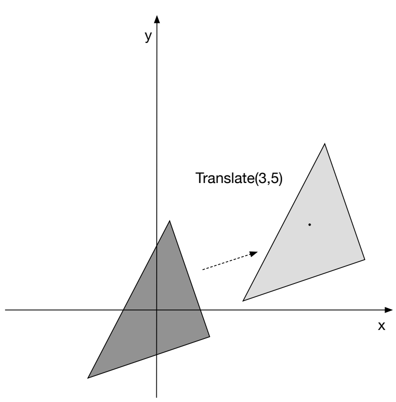
Scale/Rotate about P(x,y)



Scale/Rotate about P(x,y)



Scale/Rotate about P(x,y)



Scale/Rotate about P(x,y)

$$Translate(3,5) \cdot Scale(2,2) \cdot Translate(-3,-5) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 0 & -3 \\ 0 & 2 & -5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

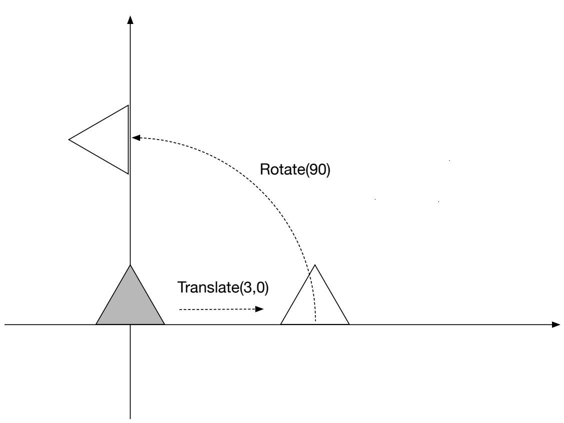
Order of Transformations

$p' = M_1 \cdot p$

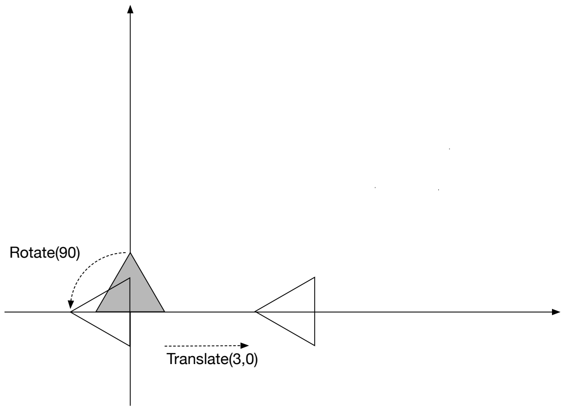
$p' = M_2 M_1 \cdot p$ — pre-multiply

$p' = M_1 M_2 \cdot p$ — post-multiply

Order of Transformations



Order of Transformations



Order of Transformations

$$p' = M_1 \cdot p$$

$$p' = M_2 M_1 \cdot p$$

M_2 happens after M_1

Pre-multiply

$$p' = M_1 M_2 \cdot p$$

M_2 happens before M_1

Post-multiply

Which is Better?

- both are valid
- many graphics systems use **post-multiply**

$$p' = M_1 M_2 \cdot p$$

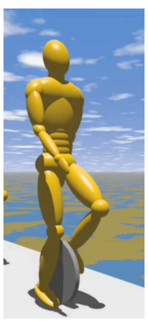
(M_2 happens before M_1)

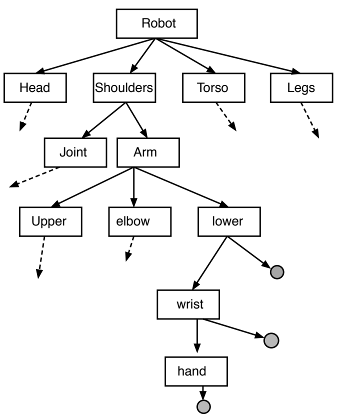
- why?

Stack of Transformation Matricies

- Current Transformation Matrix (CTM)
- InitCTM()
- PushCTM()
- PopCTM()

Hierarchical Models





Hierarchical Models

```
for(i=0; i < 10; i++){
    translate(5,0,0)
    robot()
}
```