Hierarchical Transformation

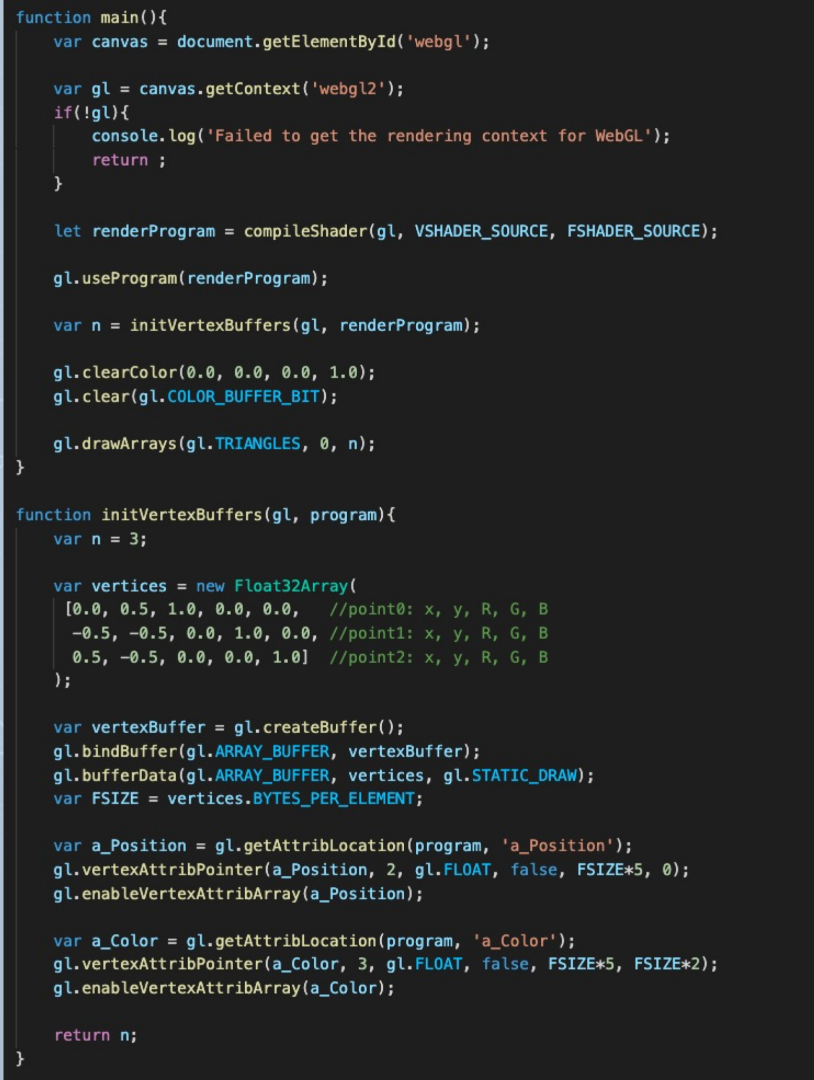
CSU0021: Computer Graphics

Reuse VBO

Reuse VBO

- This is how we create and use VBO in Ex02-3
- What if
  - we have different shapes to draw on the same frame and move them on different frames
  - E.g. you have a triangle and a rectangle in the scene, and they move.

- We can keep recreating buffers and pass data to buffers for all shapes. But, it is not a good idea to repeatedly recreate buffers.



```
function main(){
    var canvas = document.getElementById('webgl');

    var gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    let renderProgram = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);

    gl.useProgram(renderProgram);

    var n = initVertexBuffers(gl, renderProgram);

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);

    gl.drawArrays(gl.TRIANGLES, 0, n);
}

function initVertexBuffers(gl, program){
    var n = 3;

    var vertices = new Float32Array(
      [0.0, 0.5, 1.0, 0.0, 0.0,    //point0: x, y, R, G, B
       -0.5, -0.5, 0.0, 1.0, 0.0,  //point1: x, y, R, G, B
       0.5, -0.5, 0.0, 0.0, 1.0]   //point2: x, y, R, G, B
    );

    var vertexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
    var FSIZE = vertices.BYTES_PER_ELEMENT;

    var a_Position = gl.getAttribLocation(program, 'a_Position');
    gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE*5, 0);
    gl.enableVertexAttribArray(a_Position);

    var a_Color = gl.getAttribLocation(program, 'a_Color');
    gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, FSIZE*5, FSIZE*2);
    gl.enableVertexAttribArray(a_Color);

    return n;
}
```

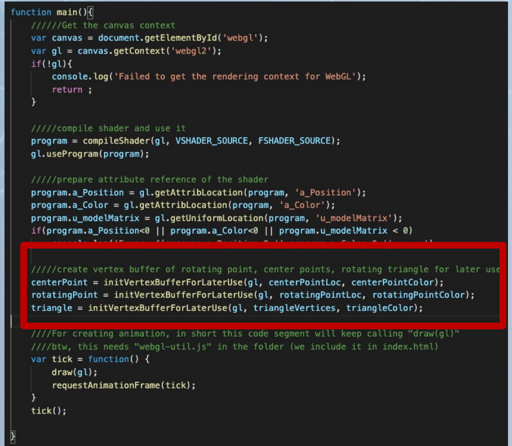
Reuse VBO

- We have shown how to reuse VBO in Lab03
  - Create VBOs for shapes and repeatedly use it for rendering in the following frames

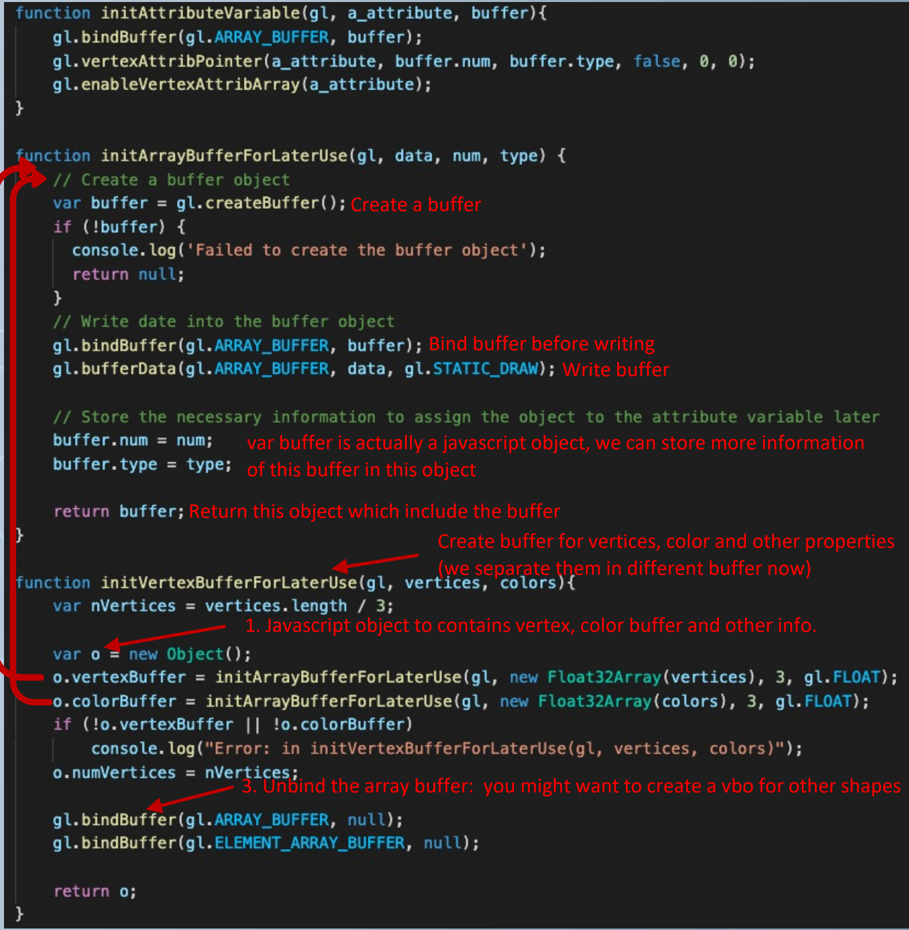
- It involves these self-defined function in WebGL.js
  - initAttributeVariable()
  - initArrayBufferForLaterUse()
  - initVertexBufferForLaterUse()
  - And, of course, main() and draw()

Reuse VBO (Prepare)

- Recall: we have 5 steps to use VBO
  - Create a buffer: **gl.createBuffer()**
  - Bind the buffer: **gl.bindBuffer()**
  - Write vertices information to the buffer: **gl.bufferData()**
  - Assign the buffer to an “attribute” variable in vertex shader: **gl.vertexAttribPointer()**
  - Enable the attribute variable: **gl.enableVertexAttributeArray()**



```
function main(){
    //////Get the canvas context
    var canvas = document.getElementById('webgl');
    var gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    /////compile shader and use it
    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);
    gl.useProgram(program);

    /////prepare attribute reference of the shader
    program.a_Position = gl.getAttribLocation(program, 'a_Position');
    program.a_Color = gl.getAttribLocation(program, 'a_Color');
    program.u_modelMatrix = gl.getUniformLocation(program, 'u_modelMatrix');
    if(program.a_Position<0 || program.a_Color<0 || program.u_modelMatrix < 0)
        console.log('Error: f(program.a_Position<0 || program.a_Color<0 || .....');

    /////create vertex buffer of rotating point, center points, rotating triangle for later use
    centerPoint = initVertexBufferForLaterUse(gl, centerPointLoc, centerPointColor);
    rotatingPoint = initVertexBufferForLaterUse(gl, rotatingPointLoc, rotatingPointColor);
    triangle = initVertexBufferForLaterUse(gl, triangleVertices, triangleColor);

    ////For creating animation, in short this code segment will keep calling "draw(gl)"
    ////btw, this needs "webgl-util.js" in the folder (we include it in index.html)
    var tick = function() {
        draw(gl);
        requestAnimationFrame(tick);
    }
    tick();

}
```



```
function initAttributeVariable(gl, a_attribute, buffer){
    gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
    gl.vertexAttribPointer(a_attribute, buffer.num, buffer.type, false, 0, 0);
    gl.enableVertexAttribArray(a_attribute);
}

function initArrayBufferForLaterUse(gl, data, num, type) {
    // Create a buffer object
    var buffer = gl.createBuffer();
    if (!buffer) {
      console.log('Failed to create the buffer object');
      return null;
    }
    // Write date into the buffer object
    gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
    gl.bufferData(gl.ARRAY_BUFFER, data, gl.STATIC_DRAW);

    // Store the necessary information to assign the object to the attribute variable later
    buffer.num = num;
    buffer.type = type;

    return buffer;
}

function initVertexBufferForLaterUse(gl, vertices, colors){
    var nVertices = vertices.length / 3;

    var o = new Object();
    o.vertexBuffer = initArrayBufferForLaterUse(gl, new Float32Array(vertices), 3, gl.FLOAT);
    o.colorBuffer = initArrayBufferForLaterUse(gl, new Float32Array(colors), 3, gl.FLOAT);
    if (!o.vertexBuffer || !o.colorBuffer)
      console.log("Error: in initVertexBufferForLaterUse(gl, vertices, colors)");
    o.numVertices = nVertices;

    gl.bindBuffer(gl.ARRAY_BUFFER, null);
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, null);

    return o;
}
```

Create a buffer

Bind buffer before writing

Write buffer

var buffer is actually a javascript object, we can store more information of this buffer in this object

Return this object which include the buffer

Create buffer for vertices, color and other properties (we separate them in different buffer now)

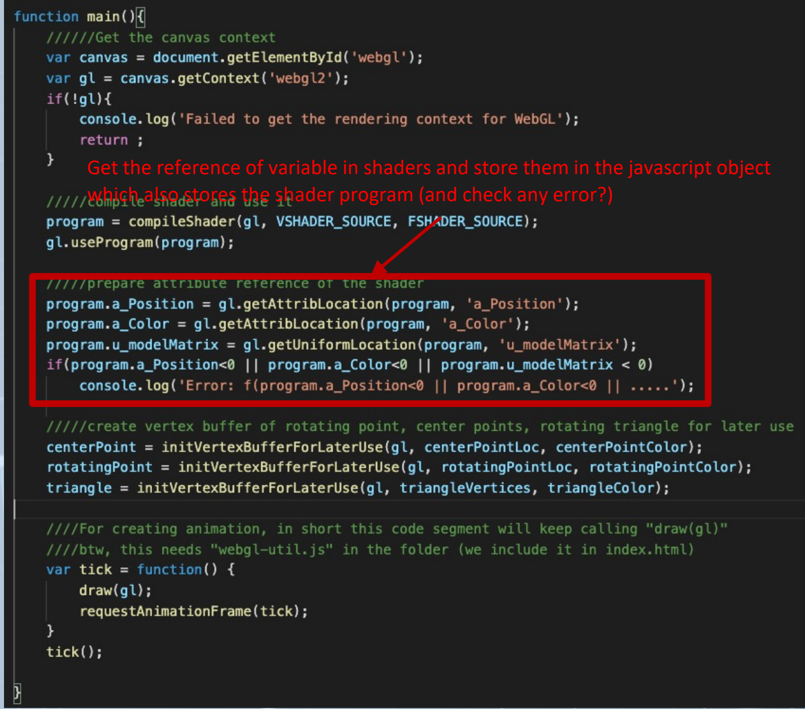
1. Javascript object to contains vertex, color buffer and other info.

2.

3. Unbind the array buffer: you might want to create a vbo for other shapes

Reuse VBO (Prepare)

- Recall: we have 5 steps to use VBO
  - Create a buffer: **gl.createBuffer()**
  - Bind the buffer**: gl.bindBuffer()**
  - Write vertices information to the buffer: **gl.bufferData()**
  - Assign the buffer to an “attribute” variable in vertex shader: **gl.vertexAttribPointer()**
  - Enable the attribute variable: **gl.enableVertexAttributeArray()**



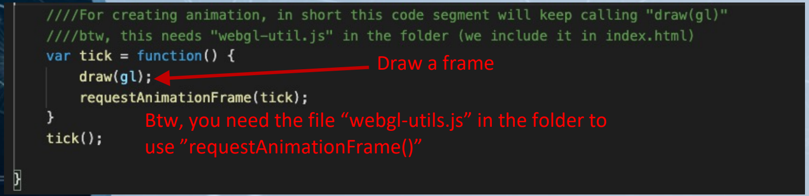
```
function main(){
    //////Get the canvas context
    var canvas = document.getElementById('webgl');
    var gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    /////compile shader and use it
    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);
    gl.useProgram(program);

    /////prepare attribute reference of the shader
    program.a_Position = gl.getAttribLocation(program, 'a_Position');
    program.a_Color = gl.getAttribLocation(program, 'a_Color');
    program.u_modelMatrix = gl.getUniformLocation(program, 'u_modelMatrix');
    if(program.a_Position<0 || program.a_Color<0 || program.u_modelMatrix < 0)
        console.log('Error: f(program.a_Position<0 || program.a_Color<0 || .....');

    /////create vertex buffer of rotating point, center points, rotating triangle for later use
    centerPoint = initVertexBufferForLaterUse(gl, centerPointLoc, centerPointColor);
    rotatingPoint = initVertexBufferForLaterUse(gl, rotatingPointLoc, rotatingPointColor);
    triangle = initVertexBufferForLaterUse(gl, triangleVertices, triangleColor);

    ////For creating animation, in short this code segment will keep calling "draw(gl)"
    ////btw, this needs "webgl-util.js" in the folder (we include it in index.html)
    var tick = function() {
        draw(gl);
        requestAnimationFrame(tick);
    }
    tick();
}
```

Get the reference of variable in shaders and store them in the javascript object which also stores the shader program (and check any error?)

Get references of attribute variables in the vertex shader

Reuse VBO (USE)

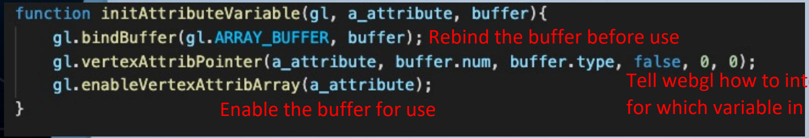
- Recall: we have 5 steps to use VBO
  - Create a buffer: **gl.createBuffer()**
  - Bind the buffer: **gl.bindBuffer()**
  - Write vertices information to the buffer: **gl.bufferData()**
  - Assign the buffer to an "attribute" variable in vertex shader: **gl.vertexAttribPointer()**
  - Enable the attribute variable: **gl.enableVertexAttributeArray()**



```
////For creating animation, in short this code segment will keep calling "draw(gl)"
////btw, this needs "webgl-util.js" in the folder (we include it in index.html)
var tick = function() {
    draw(gl);
    requestAnimationFrame(tick);
}
tick();
}
```

Draw a frame

Btw, you need the file "webgl-utils.js" in the folder to use "requestAnimationFrame()"

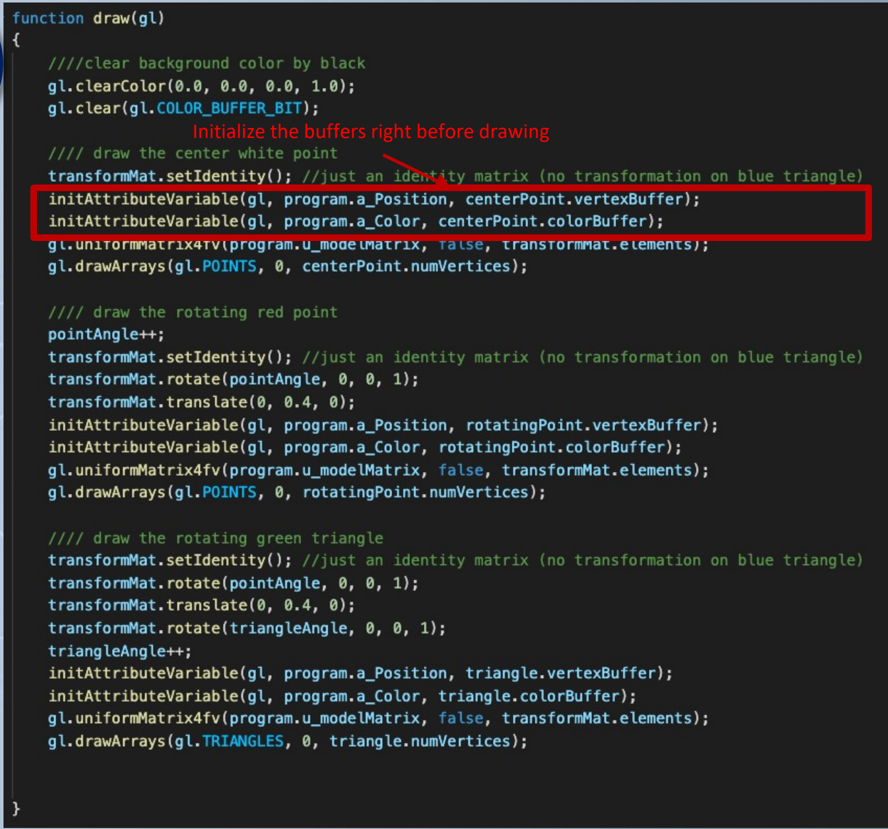


```
function initAttributeVariable(gl, a_attribute, buffer){
    gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
    gl.vertexAttribPointer(a_attribute, buffer.num, buffer.type, false, 0, 0);
    gl.enableVertexAttribArray(a_attribute);
}
```

Rebind the buffer before use

Tell webgl how to interpret the buffer and for which variable in shader

Enable the buffer for use



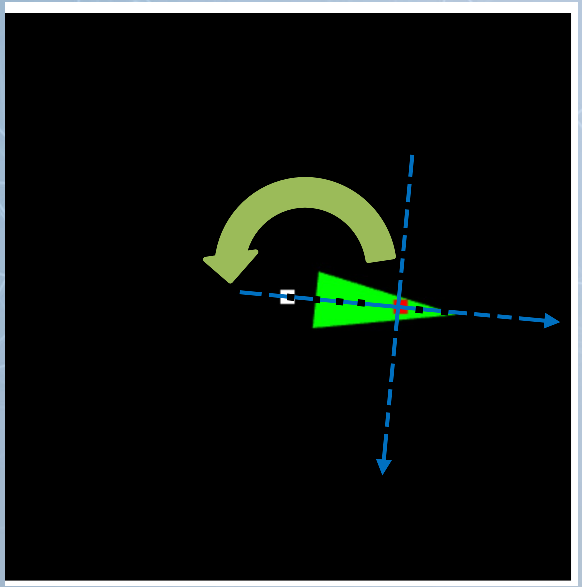
```
function draw(gl)
{
    ////clear background color by black
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);

    //// draw the center white point
    transformMat.setIdentity(); //just an identity matrix (no transformation on blue triangle)
    initAttributeVariable(gl, program.a_Position, centerPoint.vertexBuffer);
    initAttributeVariable(gl, program.a_Color, centerPoint.colorBuffer);
    gl.uniformMatrix4fv(program.u_modelMatrix, false, transformMat.elements);
    gl.drawArrays(gl.POINTS, 0, centerPoint.numVertices);

    //// draw the rotating red point
    pointAngle++;
    transformMat.setIdentity(); //just an identity matrix (no transformation on blue triangle)
    transformMat.rotate(pointAngle, 0, 0, 1);
    transformMat.translate(0, 0.4, 0);
    initAttributeVariable(gl, program.a_Position, rotatingPoint.vertexBuffer);
    initAttributeVariable(gl, program.a_Color, rotatingPoint.colorBuffer);
    gl.uniformMatrix4fv(program.u_modelMatrix, false, transformMat.elements);
    gl.drawArrays(gl.POINTS, 0, rotatingPoint.numVertices);

    //// draw the rotating green triangle
    transformMat.setIdentity(); //just an identity matrix (no transformation on blue triangle)
    transformMat.rotate(pointAngle, 0, 0, 1);
    transformMat.translate(0, 0.4, 0);
    transformMat.rotate(triangleAngle, 0, 0, 1);
    triangleAngle++;
    initAttributeVariable(gl, program.a_Position, triangle.vertexBuffer);
    initAttributeVariable(gl, program.a_Color, triangle.colorBuffer);
    gl.uniformMatrix4fv(program.u_modelMatrix, false, transformMat.elements);
    gl.drawArrays(gl.TRIANGLES, 0, triangle.numVertices);

}
```

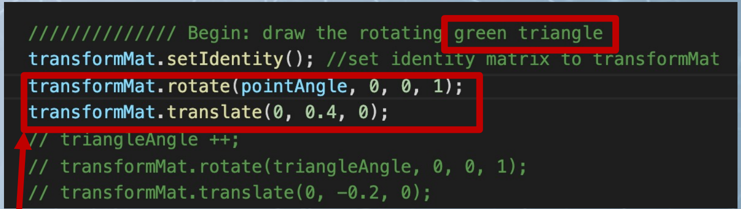
Initialize the buffers right before drawing

Time to review Lab03 again (5mins)

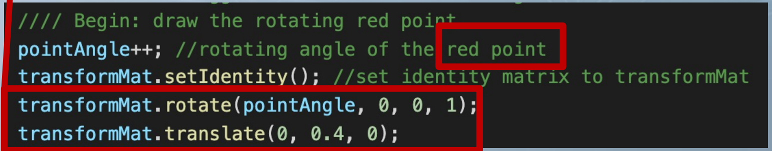
Lab03 (Review)

- 1. move the origin of the local coordinate of the triangle to the red point
  - triangle rotate around the white point **without** self spin





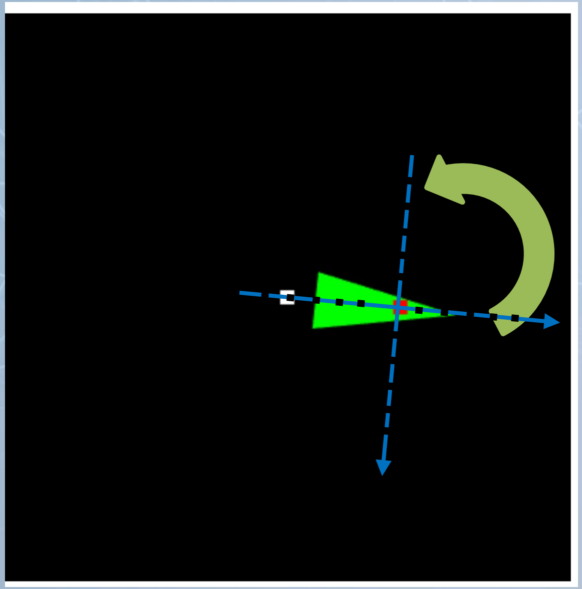
```
/////////////// Begin: draw the rotating green triangle
transformMat.setIdentity(); //set identity matrix to transformMat
transformMat.rotate(pointAngle, 0, 0, 1);
transformMat.translate(0, 0.4, 0);
// triangleAngle ++;
// transformMat.rotate(triangleAngle, 0, 0, 1);
// transformMat.translate(0, -0.2, 0);
```

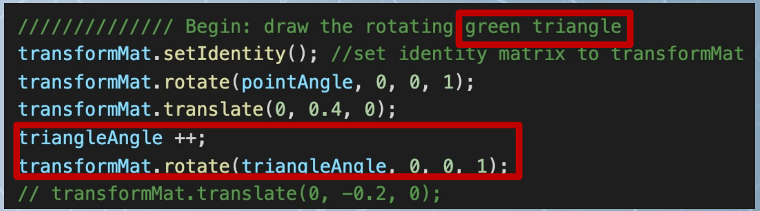


```
//// Begin: draw the rotating red point
pointAngle++; //rotating angle of the red point
transformMat.setIdentity(); //set identity matrix to transformMat
transformMat.rotate(pointAngle, 0, 0, 1);
transformMat.translate(0, 0.4, 0);
```

Lab03 (Review)

- 2. make the blue coordinate system rotate
  - triangle rotate around the white point **with** self spin

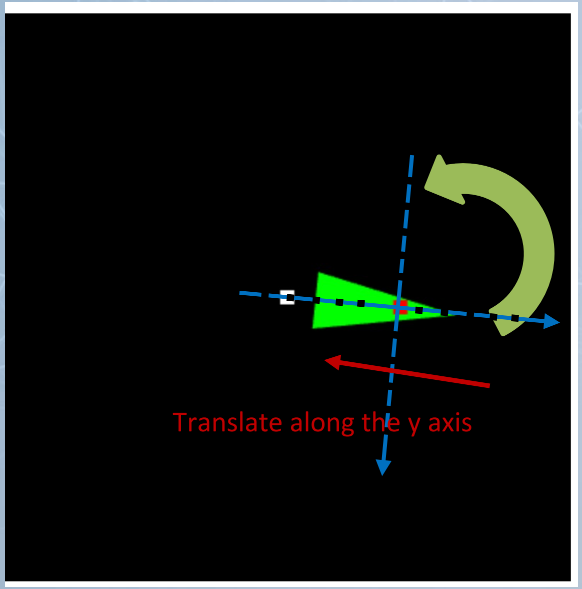




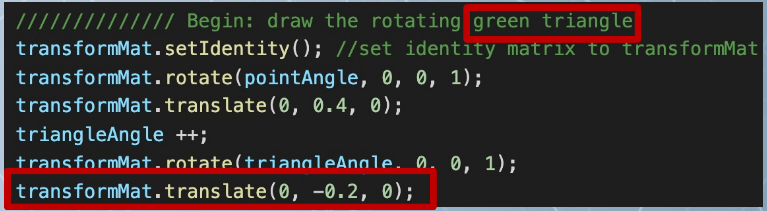
```
/////////////// Begin: draw the rotating green triangle
transformMat.setIdentity(); //set identity matrix to transformMat
transformMat.rotate(pointAngle, 0, 0, 1);
transformMat.translate(0, 0.4, 0);
triangleAngle ++;
transformMat.rotate(triangleAngle, 0, 0, 1);
// transformMat.translate(0, -0.2, 0);
```

Lab03 (Review)

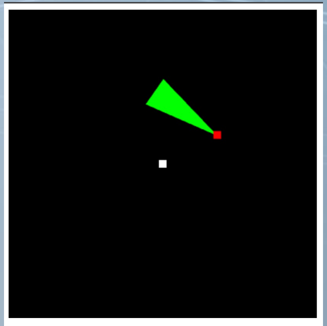
- 3. use translation to move the tip of the triangle on the top of the red point
  - done



Translate along the y axis



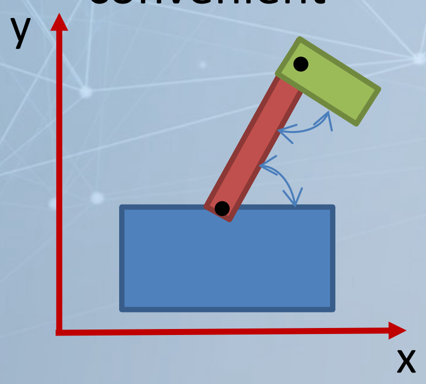
```
////////////// Begin: draw the rotating green triangle
transformMat.setIdentity(); //set identity matrix to transformMat
transformMat.rotate(pointAngle, 0, 0, 1);
transformMat.translate(0, 0.4, 0);
triangleAngle ++;
transformMat.rotate(triangleAngle, 0, 0, 1);
transformMat.translate(0, -0.2, 0);
```



Hierarchical Transformation

Object with Multiple Components

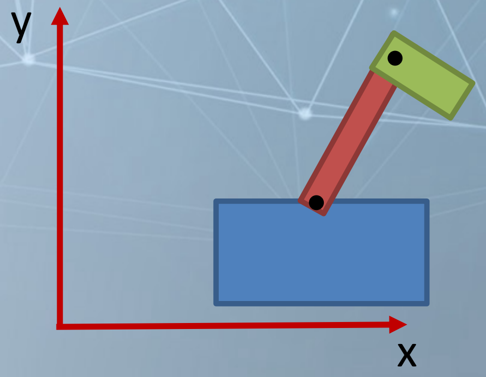
- What if you want to move an object consists of multiple components
  - Ex: robot
  - 2D example here
- You can transform each part independently, but this is not convenient



Translate blue by tx = 5

Translate red by tx = 5

Translate green by tx = 5



Multiple Transformations (Refresh Our Memory)

translate(2,1) -> rotate(45degrees)->scale(1, 2)

$$\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\frac{PI}{4}) & -\sin(\frac{PI}{4}) & 0 \\ \sin(\frac{PI}{4}) & \cos(\frac{PI}{4}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.828 \\ 6.656 \\ 1 \end{bmatrix}$$

Order here is defined by the order of matrix multiplication

Coordinate in object space

Where to draw in world space

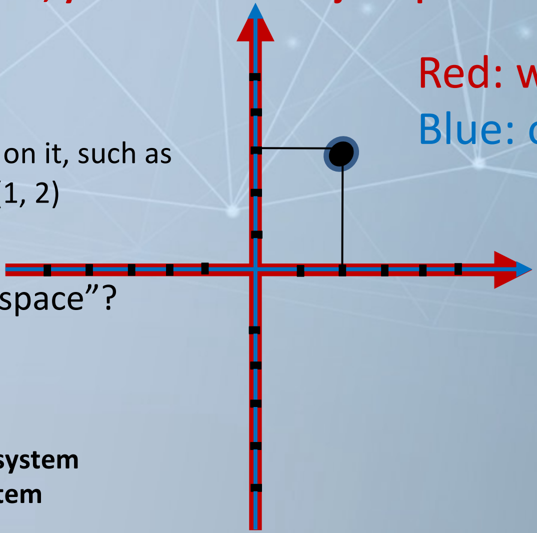
One Way to Interpret Transformation (Refresh Our Memory)

- Imagine that there are two coordinate system
  - World space (never change)
  - Object space (change by transformation)
    - When you draw any object, you draw the object according to the object coordinate system
- These two systems overlap perfectly if you do not apply any transformation
- **When you apply transformation, you move the "object space"**

(x=2.0, y=3.0) in "object space"

If we want to apply some transformations on it, such as translate(2,1) -> rotate(45degrees)->scale(1, 2)

Where should we draw it in "world space"?



Red: world space

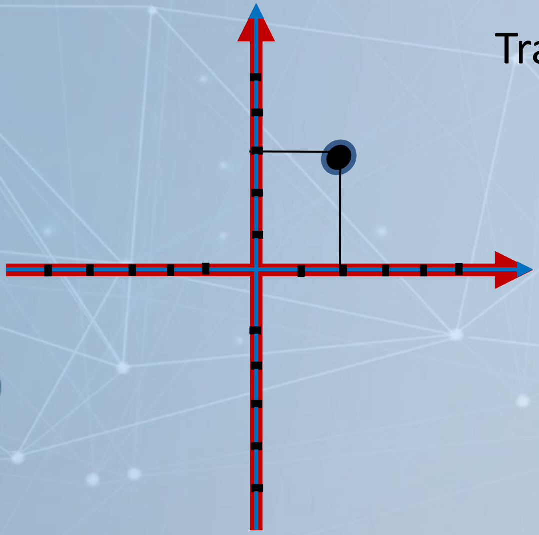
Blue: object space

The transformation of the blue coordinate system is based on the current blue coordinate system

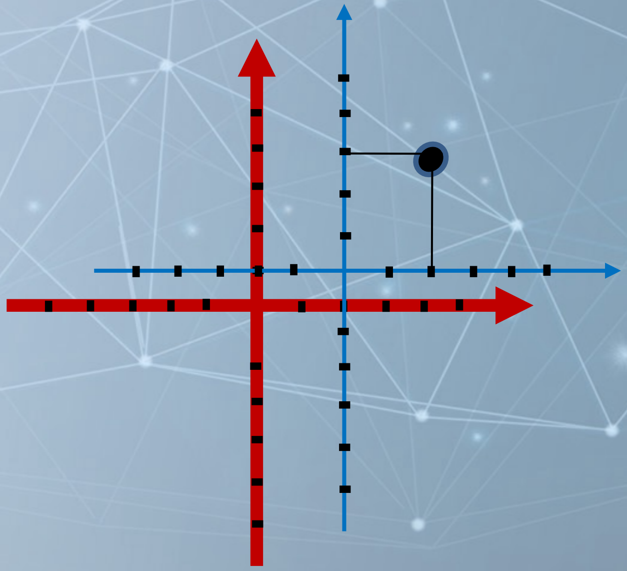
One Way to Interpret Transformation (Refresh Our Memory)

Red: world space

Blue: object space



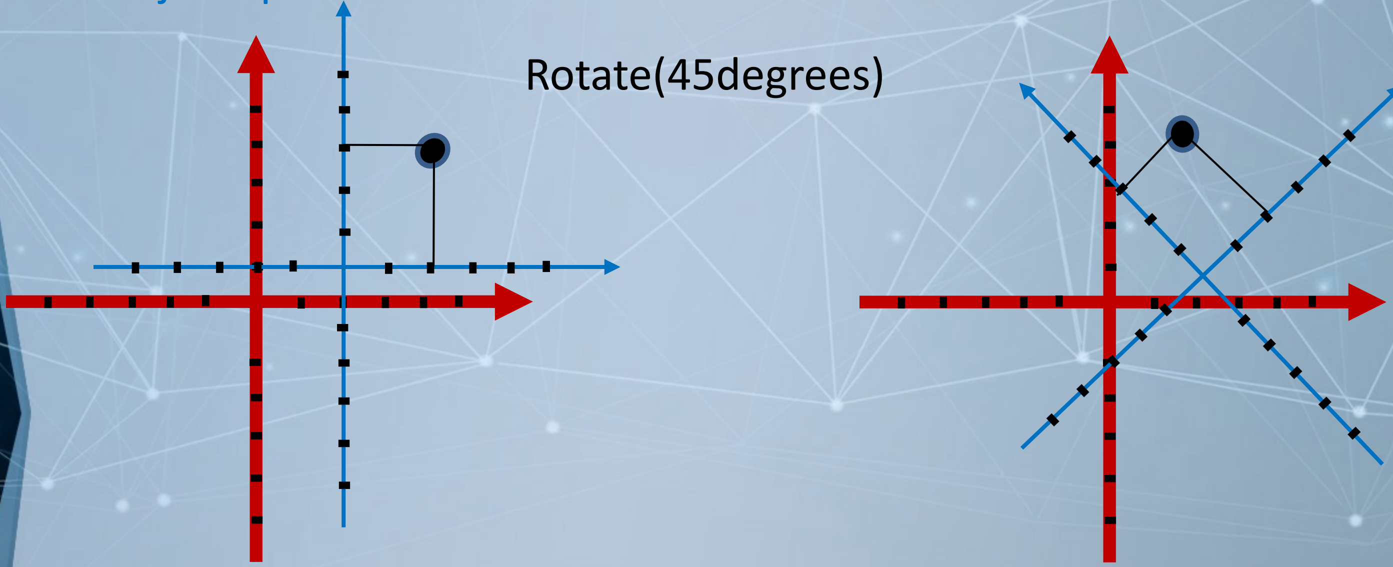
Translate(tx=2, ty=1)



One Way to Interpret Transformation (Refresh Our Memory)

Red: world space

Blue: object space

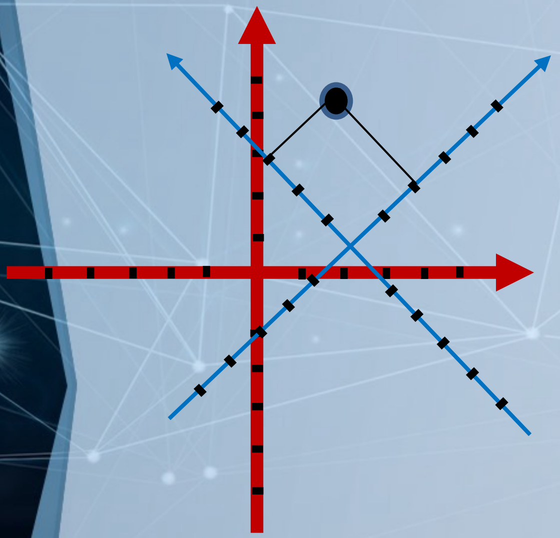


Rotate(45degrees)

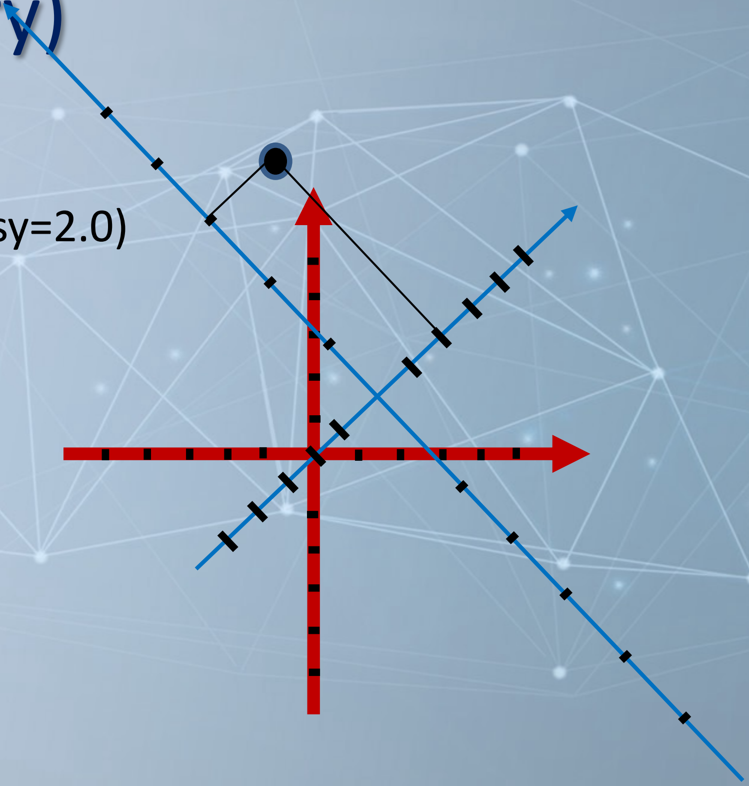
One Way to Interpret Transformation (Refresh Our Memory)

Red: world space

Blue: object space



Scale(sx=1.0, sy=2.0)



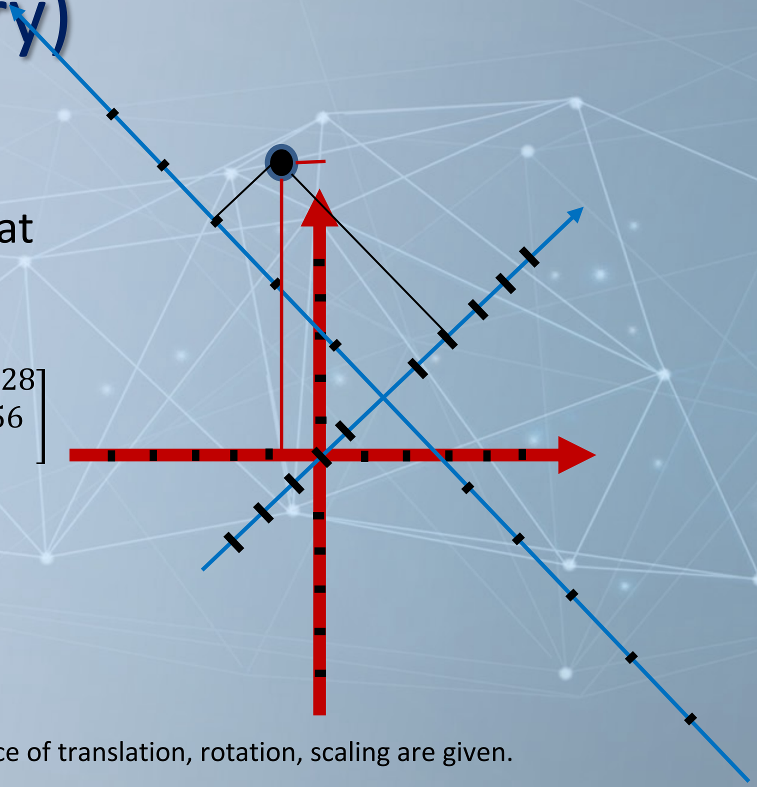
One Way to Interpret Transformation (Refresh Our Memory)

Red: world space

Blue: object space

Draw a point at
(x=2.0, y=3.0)

$$\begin{bmatrix}1 & 0 & 2\\ 0 & 1 & 1\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos(\frac{PI}{4}) & -\sin(\frac{PI}{4}) & 0\\ \sin(\frac{PI}{4}) & \cos(\frac{PI}{4}) & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}2\\ 3\\ 1\end{bmatrix} = \begin{bmatrix}-0.828\\ 6.656\\ 1\end{bmatrix}$$



Not only for you to understand what happens when a sequence of translation, rotation, scaling are given.

But also help to come up with a sequences of translations, rotations, scalings for a complex transformation

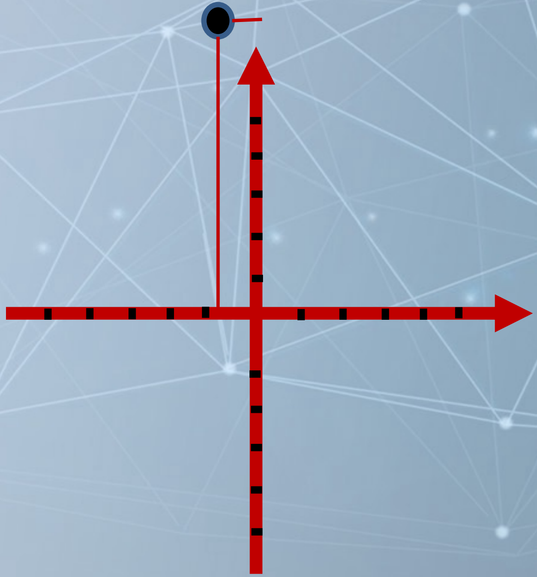
One Way to Interpret Transformation (Refresh Our Memory)

Red: world space

Blue: object space

Draw a point at (x=2.0, y=3.0)

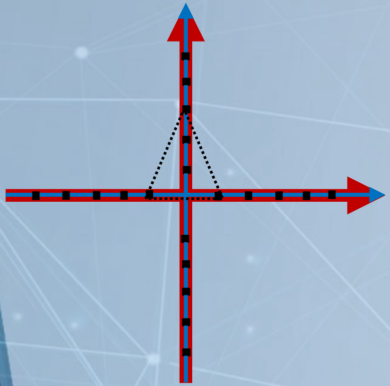
$$\begin{bmatrix}1 & 0 & 2\\0 & 1 & 1\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos(\frac{PI}{4}) & -\sin(\frac{PI}{4}) & 0\\\sin(\frac{PI}{4}) & \cos(\frac{PI}{4}) & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\0 & 2 & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}2\\3\\1\end{bmatrix}=\begin{bmatrix}-0.828\\6.656\\1\end{bmatrix}$$

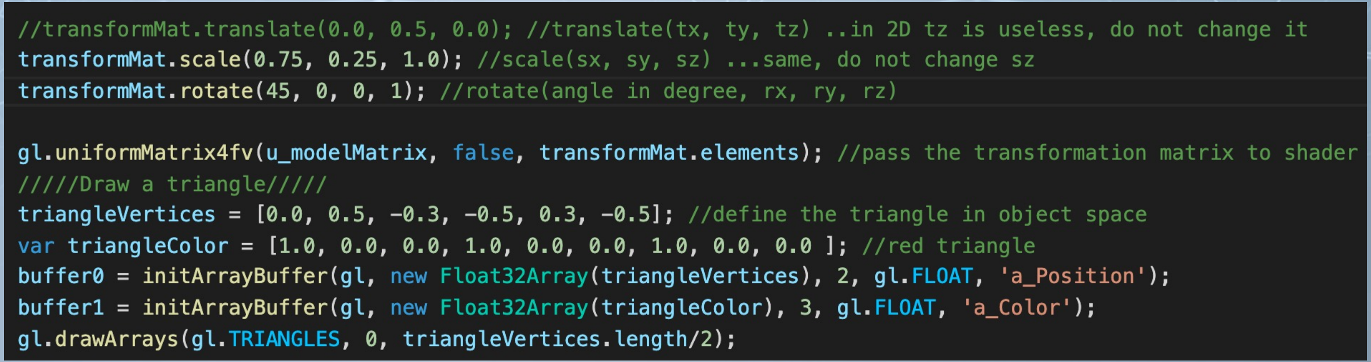


Not only for you to understand what happens when a sequence of translation, rotation, scaling are given.

But also help to come up with a sequences of translations, rotations, scalings for a complex transformation

Ex3-1 (if, non-uniform scale -> rotate)

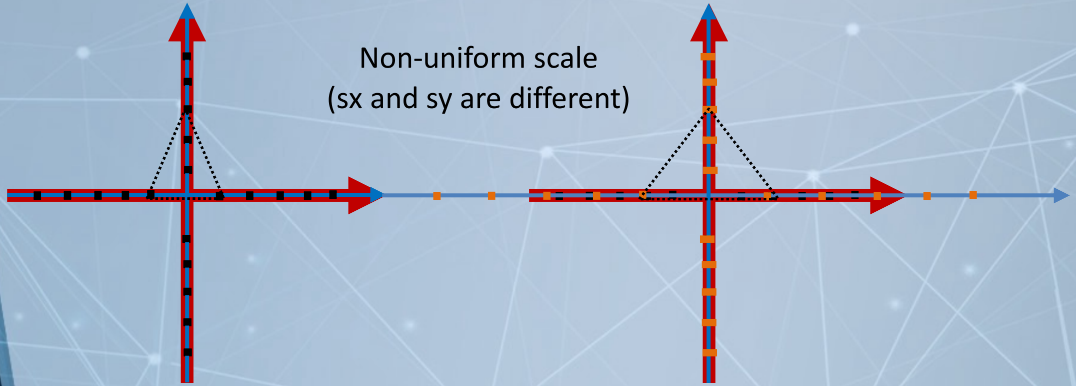




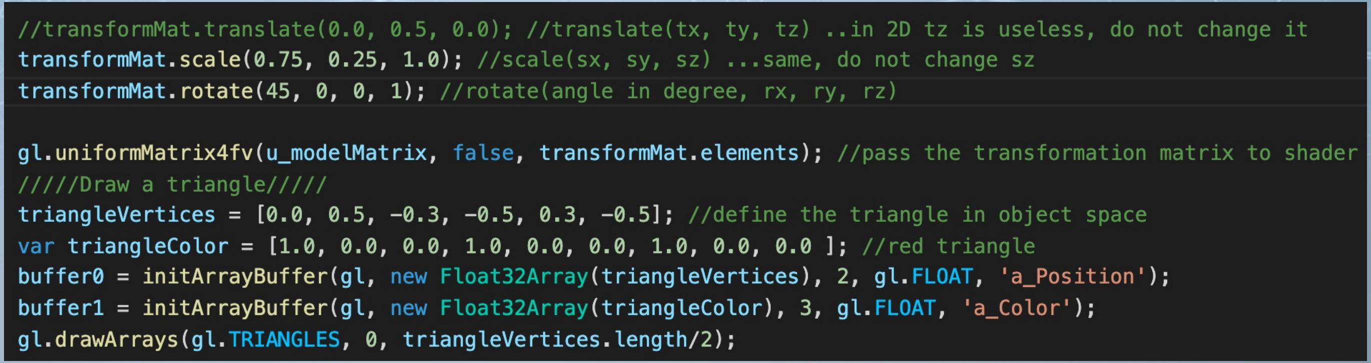
```
//transformMat.translate(0.0, 0.5, 0.0); //translate(tx, ty, tz) ..in 2D tz is useless, do not change it
transformMat.scale(0.75, 0.25, 1.0); //scale(sx, sy, sz) ...same, do not change sz
transformMat.rotate(45, 0, 0, 1); //rotate(angle in degree, rx, ry, rz)

gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements); //pass the transformation matrix to shader
/////Draw a triangle/////
triangleVertices = [0.0, 0.5, -0.3, -0.5, 0.3, -0.5]; //define the triangle in object space
var triangleColor = [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0 ]; //red triangle
buffer0 = initArrayBuffer(gl, new Float32Array(triangleVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(triangleColor), 3, gl.FLOAT, 'a_Color');
gl.drawArrays(gl.TRIANGLES, 0, triangleVertices.length/2);
```

Ex3-1 (if, non-uniform scale -> rotate)

Orange: ticks of blue axes



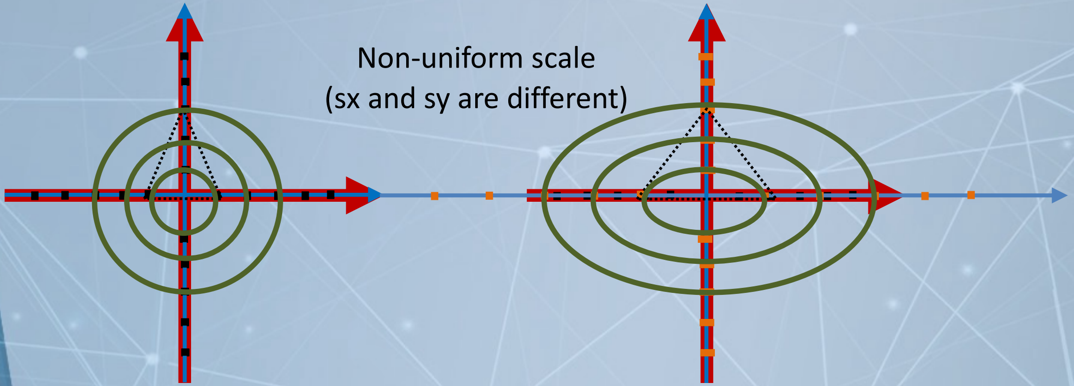
Non-uniform scale
(sx and sy are different)



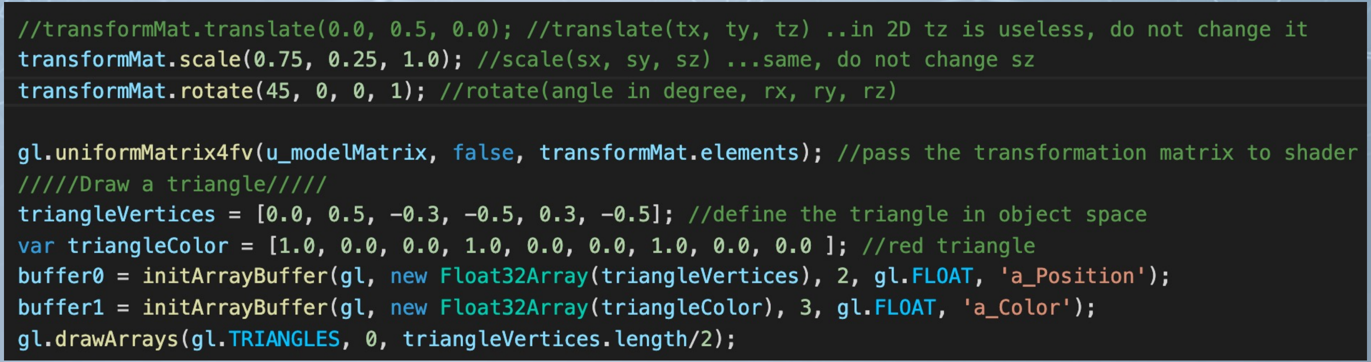
```
//transformMat.translate(0.0, 0.5, 0.0); //translate(tx, ty, tz) ..in 2D tz is useless, do not change it
transformMat.scale(0.75, 0.25, 1.0); //scale(sx, sy, sz) ...same, do not change sz
transformMat.rotate(45, 0, 0, 1); //rotate(angle in degree, rx, ry, rz)

gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements); //pass the transformation matrix to shader
/////Draw a triangle/////
triangleVertices = [0.0, 0.5, -0.3, -0.5, 0.3, -0.5]; //define the triangle in object space
var triangleColor = [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0 ]; //red triangle
buffer0 = initArrayBuffer(gl, new Float32Array(triangleVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(triangleColor), 3, gl.FLOAT, 'a_Color');
gl.drawArrays(gl.TRIANGLES, 0, triangleVertices.length/2);
```

Ex3-1 (if, non-uniform scale -> rotate)

Orange: ticks of blue axes

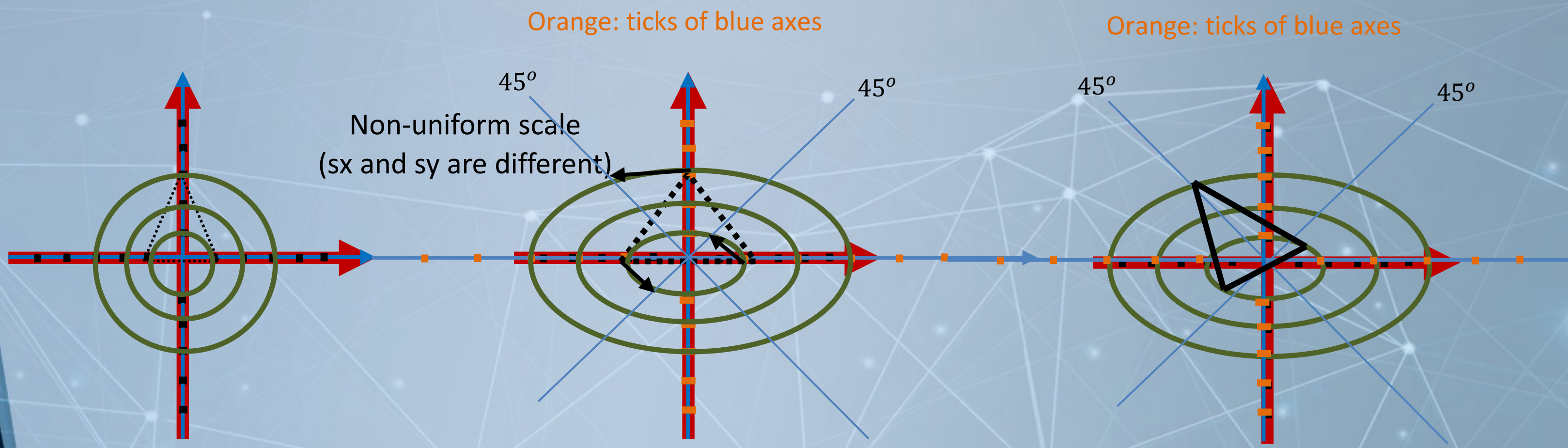


Non-uniform scale
(sx and sy are different)



```
//transformMat.translate(0.0, 0.5, 0.0); //translate(tx, ty, tz) ..in 2D tz is useless, do not change it
transformMat.scale(0.75, 0.25, 1.0); //scale(sx, sy, sz) ...same, do not change sz
transformMat.rotate(45, 0, 0, 1); //rotate(angle in degree, rx, ry, rz)

gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements); //pass the transformation matrix to shader
/////Draw a triangle/////
triangleVertices = [0.0, 0.5, -0.3, -0.5, 0.3, -0.5]; //define the triangle in object space
var triangleColor = [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0 ]; //red triangle
buffer0 = initArrayBuffer(gl, new Float32Array(triangleVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(triangleColor), 3, gl.FLOAT, 'a_Color');
gl.drawArrays(gl.TRIANGLES, 0, triangleVertices.length/2);
```

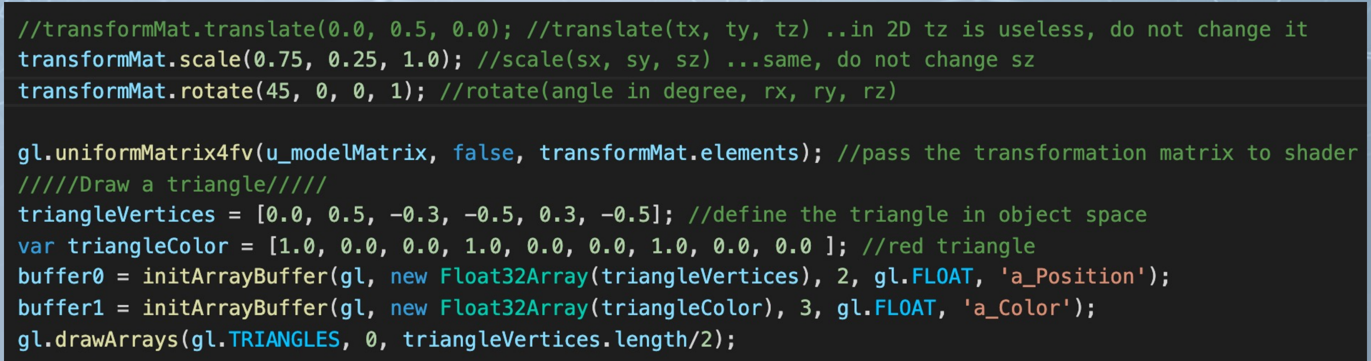
Ex3-1 (if, non-uniform scale -> rotate)



Orange: ticks of blue axes

Non-uniform scale
(sx and sy are different)

Orange: ticks of blue axes

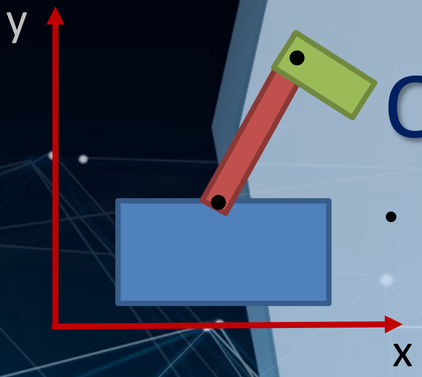


```
//transformMat.translate(0.0, 0.5, 0.0); //translate(tx, ty, tz) ..in 2D tz is useless, do not change it
transformMat.scale(0.75, 0.25, 1.0); //scale(sx, sy, sz) ...same, do not change sz
transformMat.rotate(45, 0, 0, 1); //rotate(angle in degree, rx, ry, rz)

gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements); //pass the transformation matrix to shader
/////Draw a triangle/////
triangleVertices = [0.0, 0.5, -0.3, -0.5, 0.3, -0.5]; //define the triangle in object space
var triangleColor = [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0 ]; //red triangle
buffer0 = initArrayBuffer(gl, new Float32Array(triangleVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(triangleColor), 3, gl.FLOAT, 'a_Color');
gl.drawArrays(gl.TRIANGLES, 0, triangleVertices.length/2);
```

One Way to Interpret Transformation

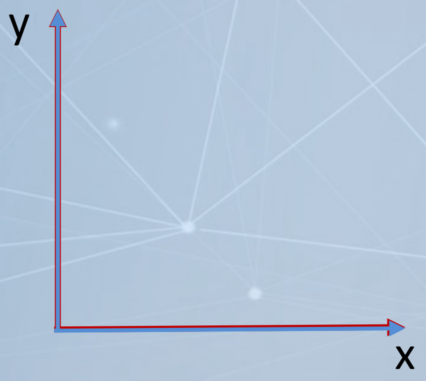
- Be careful if you use this way to interpret the operation of doing rotation/translation after **non-uniform** scaling. It may not be so intuitive!

- This interpretation just helps you to come up with a sequence of transformations to move an object, or imagine where an object goes if a sequence of transformation is given
  - The way to interpret the transformation is not unique. You can find any way you are comfortable with to interpret the transformations

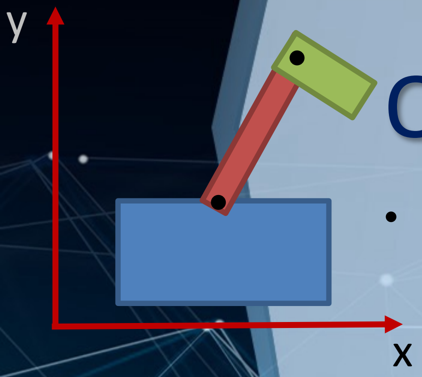


Object with Multiple Components

- The first question is “how do you draw this robot?”
  - The concept – move the object space by transformation to draw

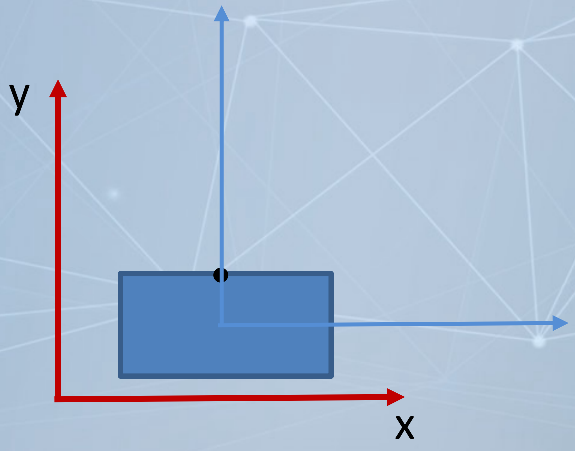


Blue coordinate: object space



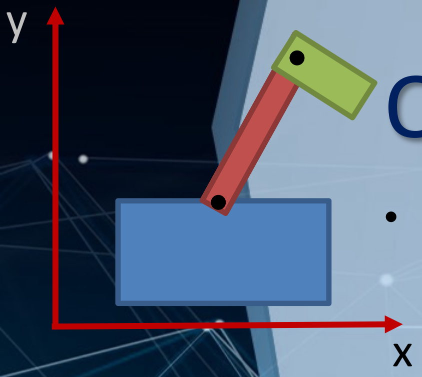
Object with Multiple Components

- The first question is “how do you draw this robot?”
  - The concept – move the object space by transformation to draw

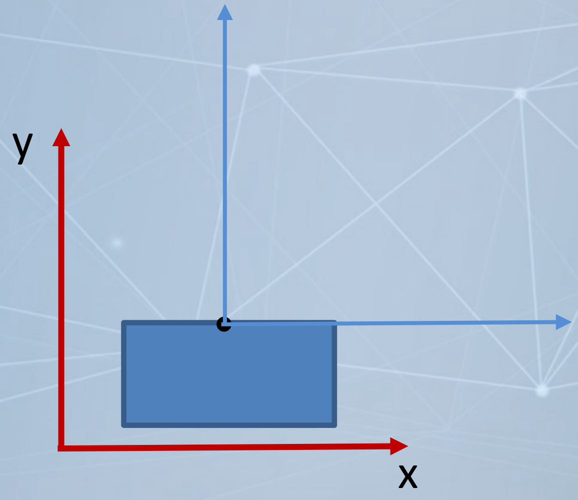


Translate(tx_1, ty_1)
Draw blue block

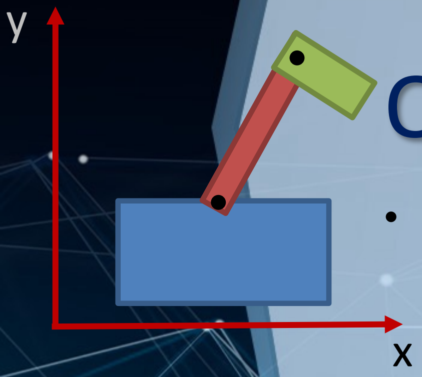
Object with Multiple Components



- The first question is “how do you draw this robot?”
  - The concept – move the object space by transformation to draw

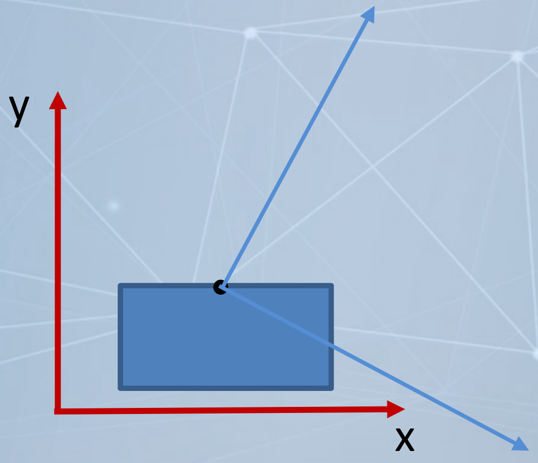


Translate(tx_1, ty_1)
Draw blue block
Translate(tx_2, ty_2)

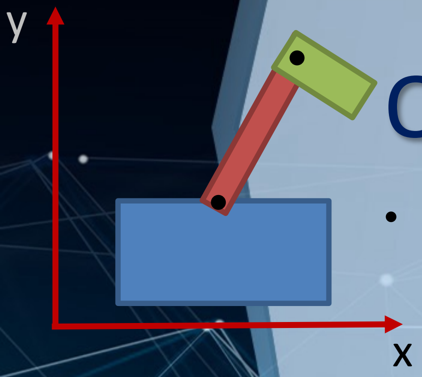


Object with Multiple Components

- The first question is “how do you draw this robot?”
  - The concept – move the object space by transformation to draw

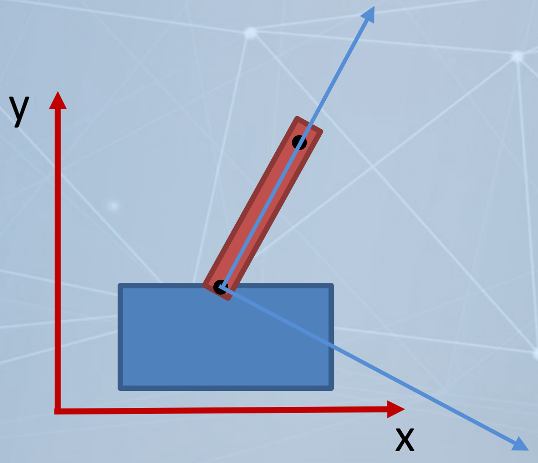


Translate(tx_1, ty_1)
Draw blue block
Translate(tx_2, ty_2)
Rotate(θ_1)

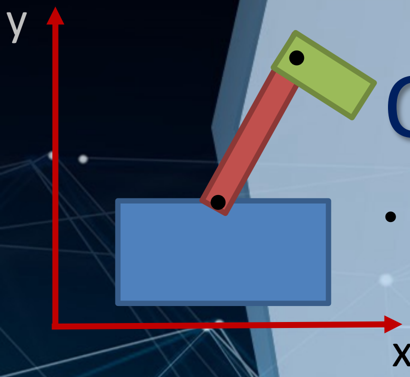


Object with Multiple Components

- The first question is “how do you draw this robot?”
  - The concept – move the object space by transformation to draw

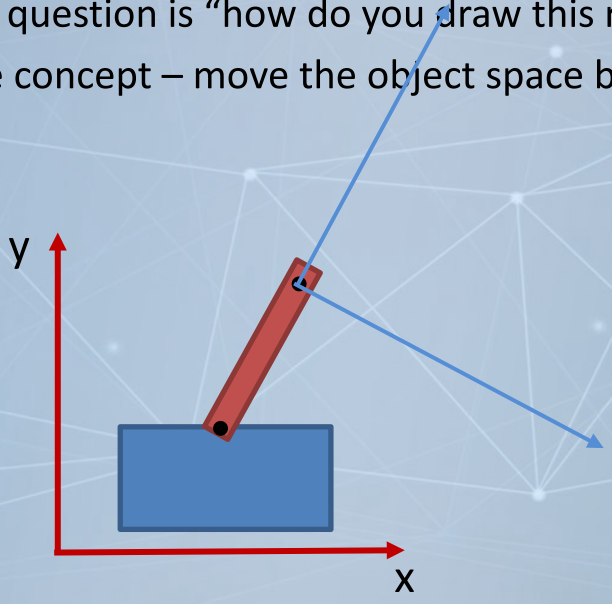


Translate(tx_1, ty_1)
Draw blue block
Translate(tx_2, ty_2)
Rotate(θ_1)
Draw red block



Object with Multiple Components

- The first question is “how do you draw this robot?”
  - The concept – move the object space by transformation to draw



Translate(tx_1, ty_1)

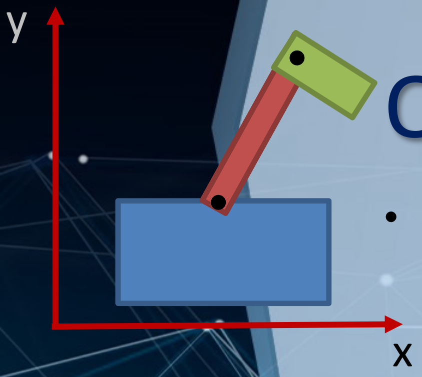
Draw blue block

Translate(tx_2, ty_2)

Rotate(θ_1)

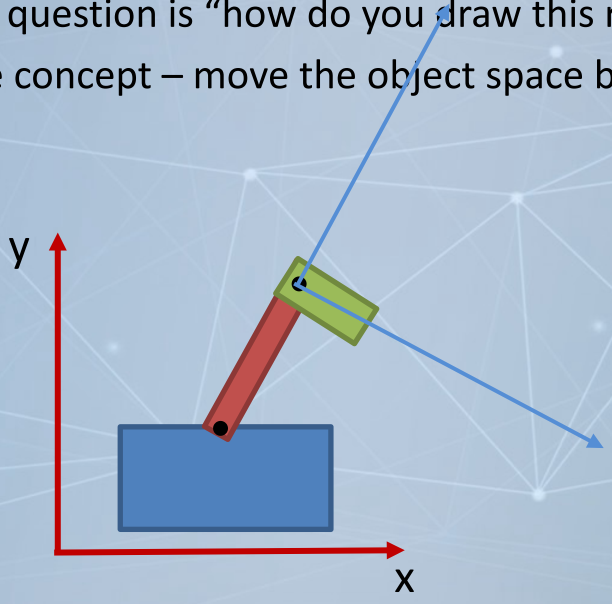
Draw red block

Translate(tx_3, ty_3)

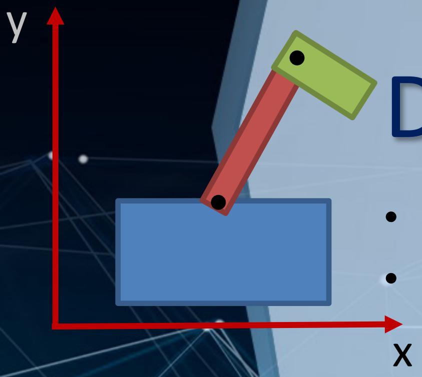


Object with Multiple Components

- The first question is “how do you draw this robot?”
  - The concept – move the object space by transformation to draw

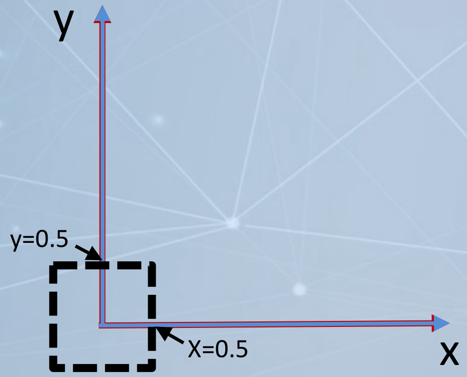


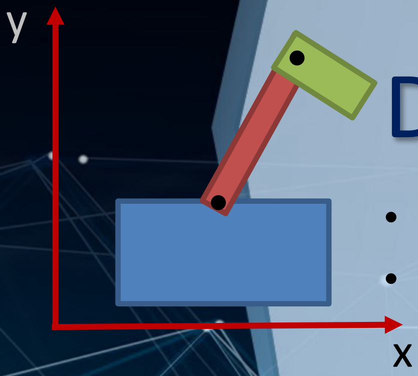
Translate(tx_1, ty_1)
Draw blue block
Translate(tx_2, ty_2)
Rotate(θ_1)
Draw red block
Translate(tx_3, ty_3)
Draw green block



Draw With Same Object Model

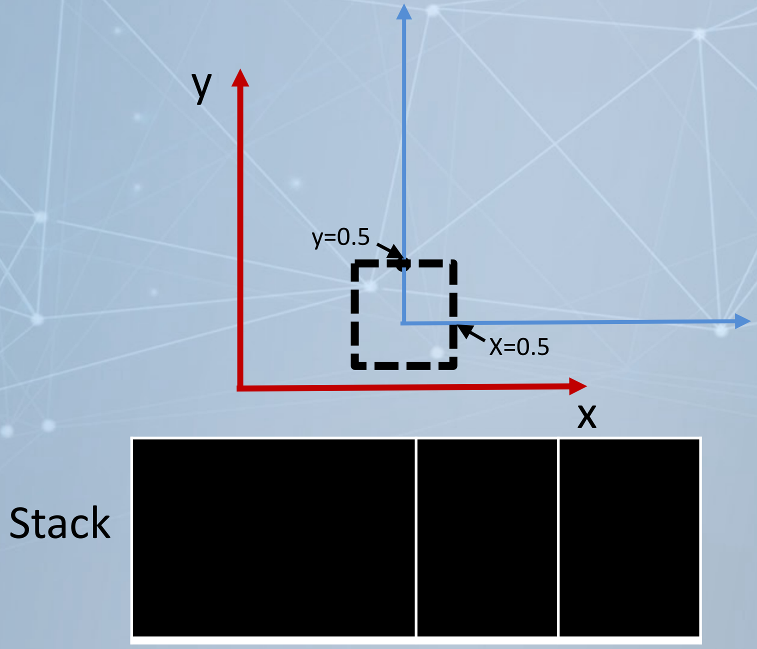
- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**



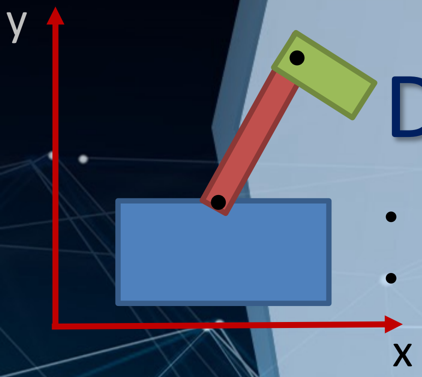


Draw With Same Object Model

- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**

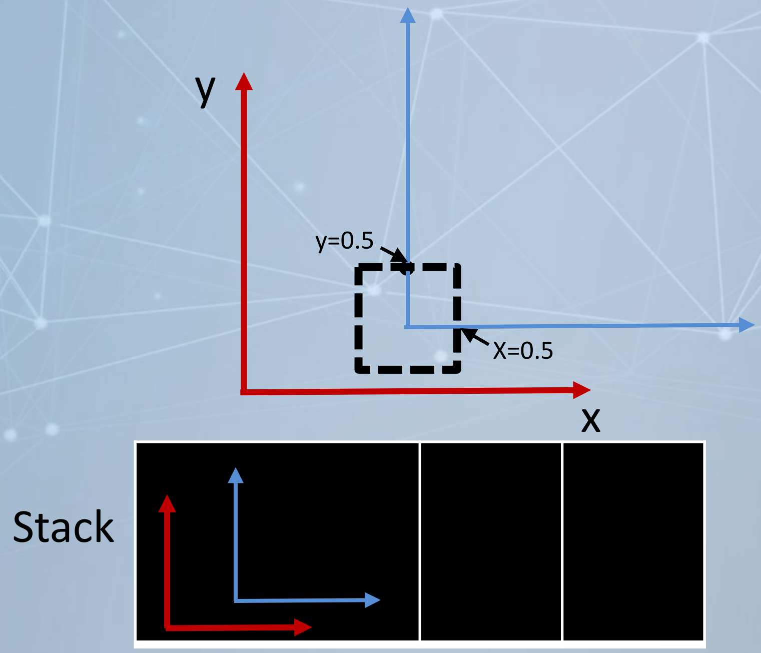


Translate(tx_1, ty_1)

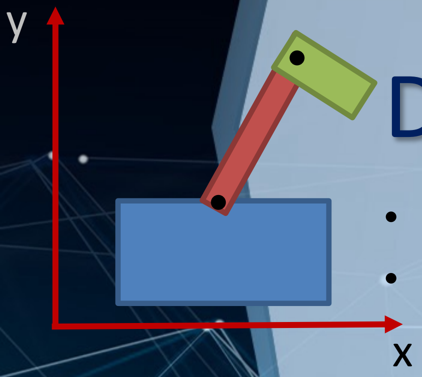


Draw With Same Object Model

- We skip some details – the detailed information to draw each block
- **Let's make the question harder – what if we always say "we want to draw a rectangle with edge length 1" when drawing blocks**

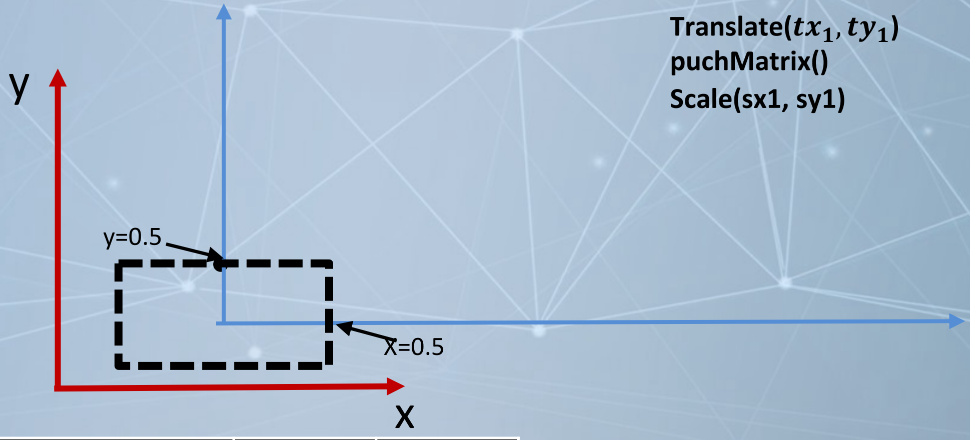


Translate(tx_1, ty_1)
pushMatrix() → Keep current state in stack



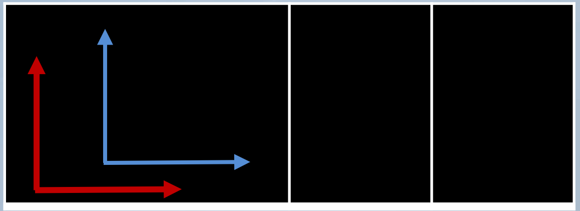
Draw With Same Object Model

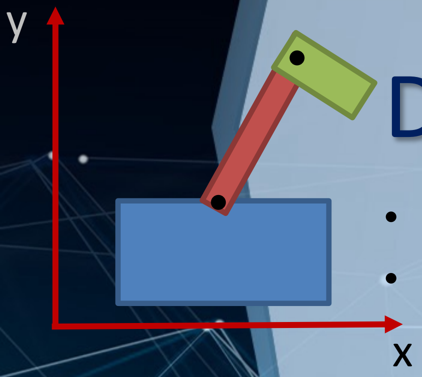
- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**



Translate(tx_1, ty_1)
puchMatrix()
Scale(sx1, sy1)

Stack

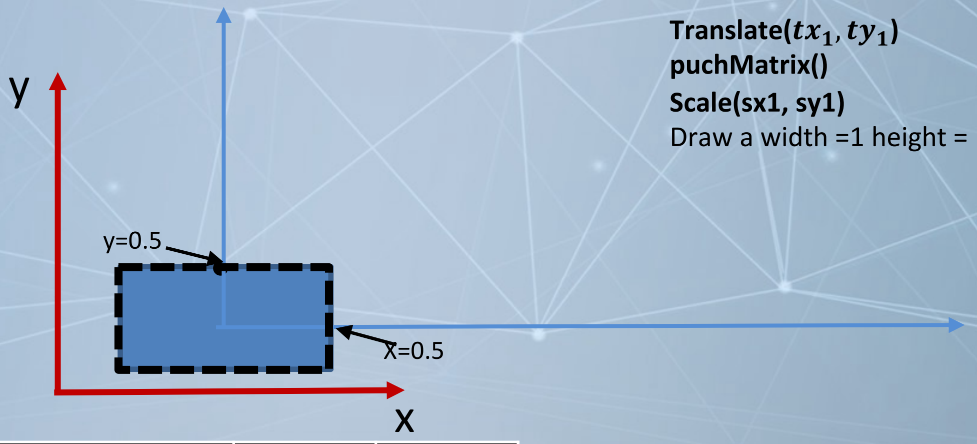




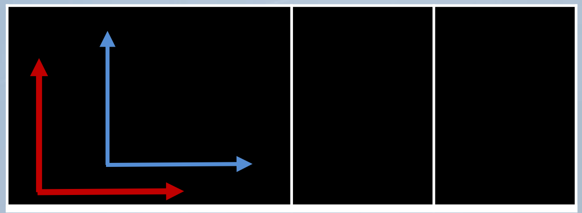
Draw With Same Object Model

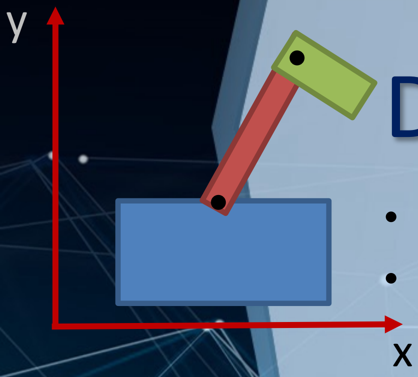
- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**

Translate(tx_1, ty_1)
puchMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block



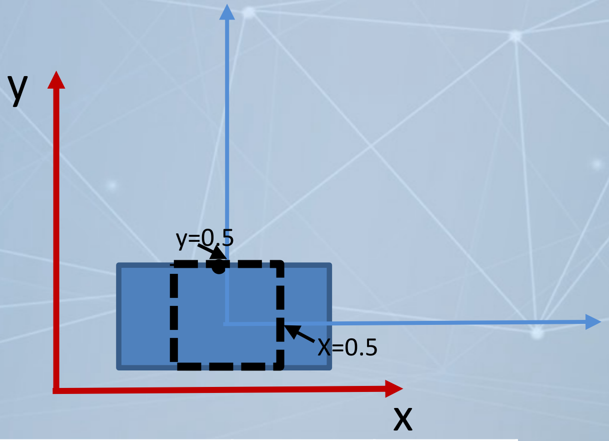
Stack





Draw With Same Object Model

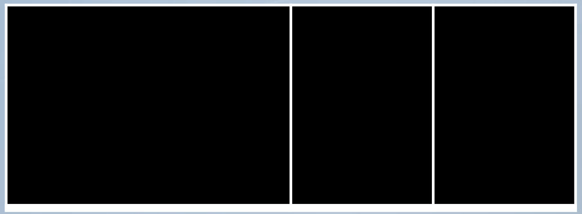
- We skip some details – the detailed information to draw each block
- **Let's make the question harder – what if we always say "we want to draw a rectangle with edge length 1" when drawing blocks**

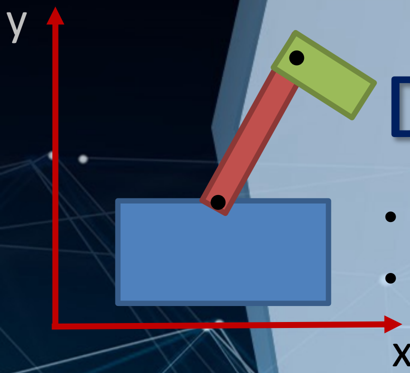


Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()

Restore the state from stack

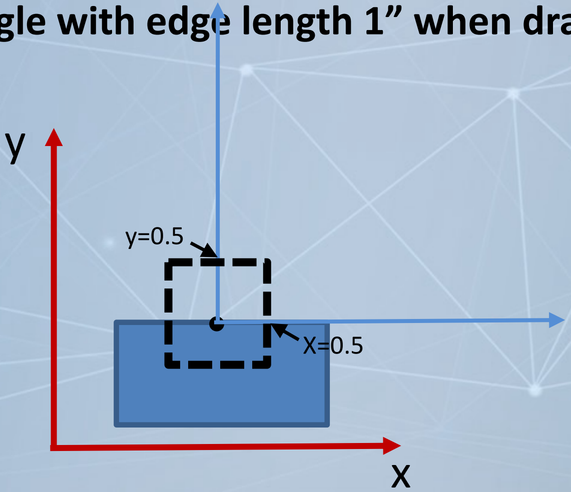
Stack





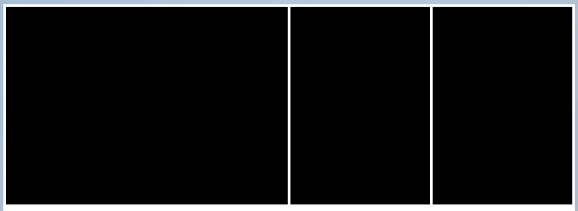
Draw With Same Object Model

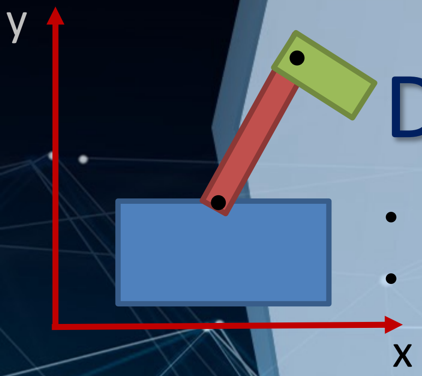
- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**



Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)

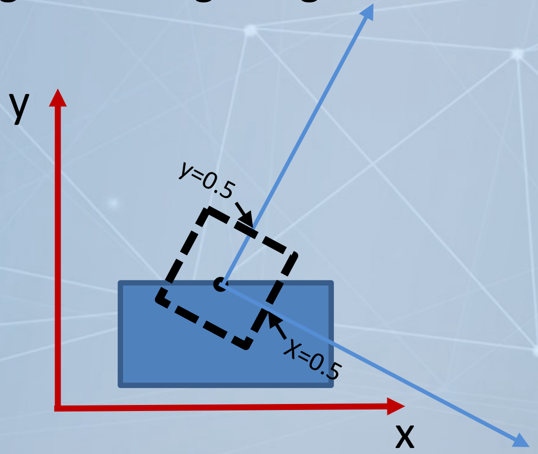
Stack



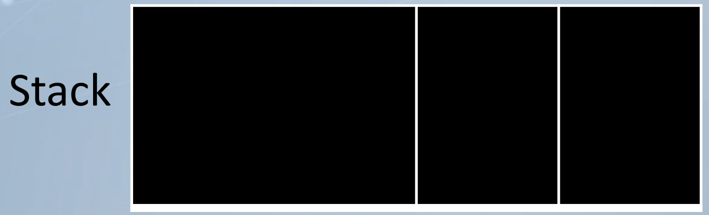


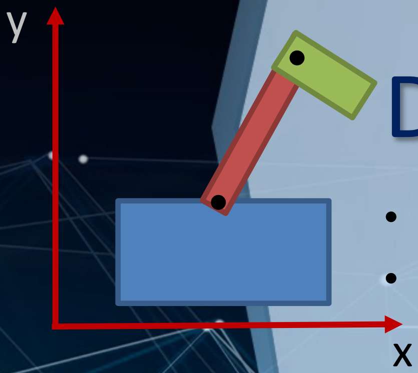
Draw With Same Object Model

- We skip some details – the detailed information to draw each block
- **Let's make the question harder – what if we always say "we want to draw a rectangle with edge length 1" when drawing blocks**



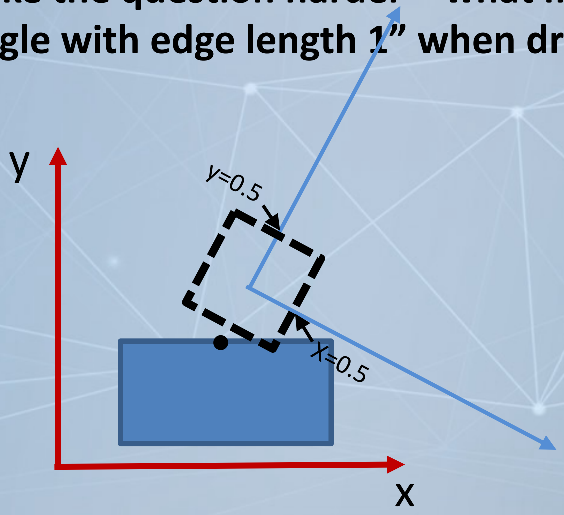
```
Translate($tx_1, ty_1$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1$)
```



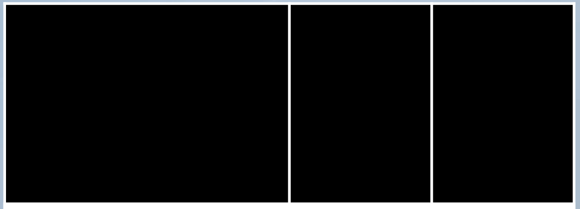


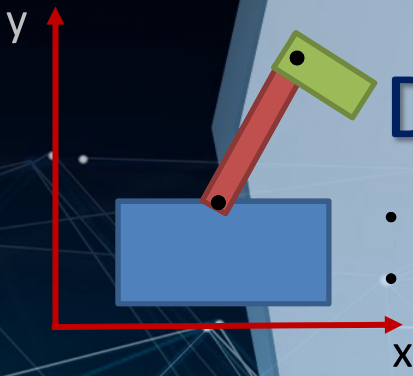
Draw With Same Object Model

- We skip some details – the detailed information to draw each block
- **Let's make the question harder – what if we always say "we want to draw a rectangle with edge length 1" when drawing blocks**



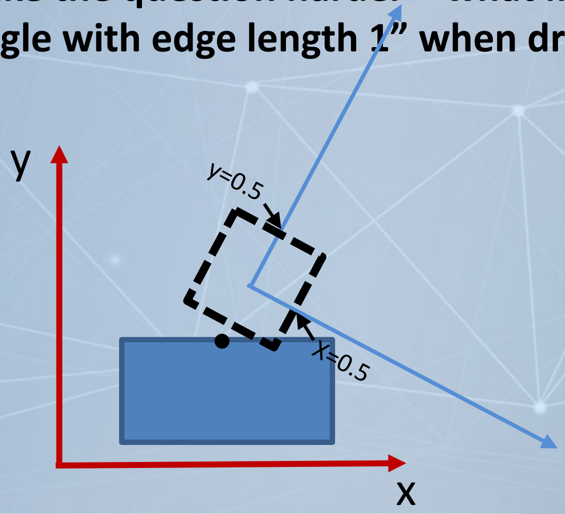
Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1)
Translate(tx_3, ty_3)


Stack



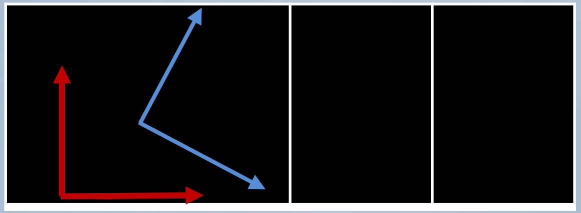
Draw With Same Object Model

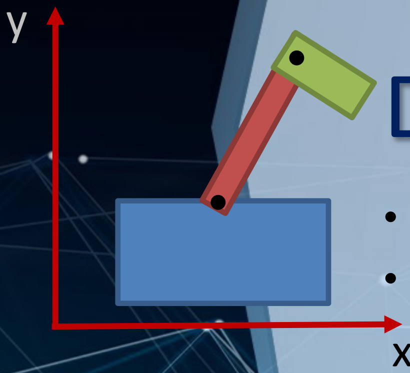
- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**



```
Translate($tx_1, ty_1$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1$)
Translate($tx_3, ty_3$)
pushMatrix()
```

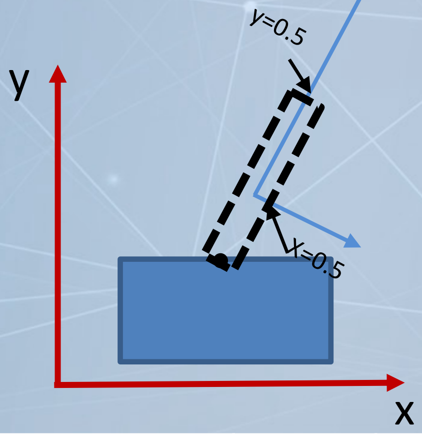
Stack





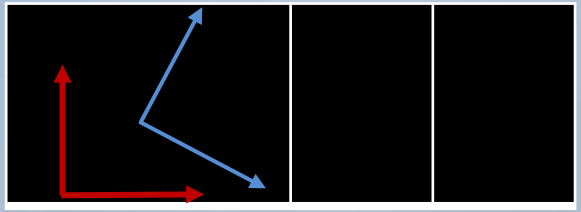
Draw With Same Object Model

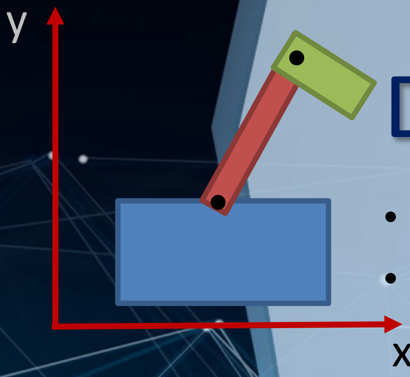
- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**



Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1)
Translate(tx_3, ty_3)
pushMatrix()
Scale(sx_2, sy_2)

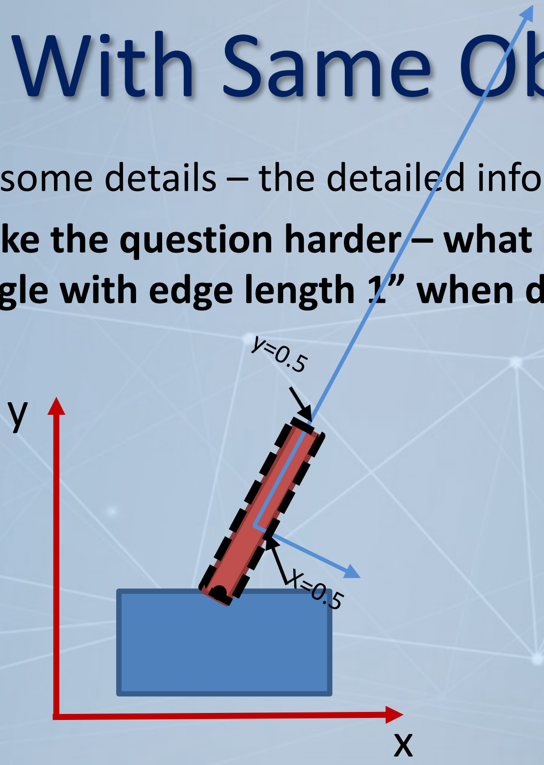
Stack





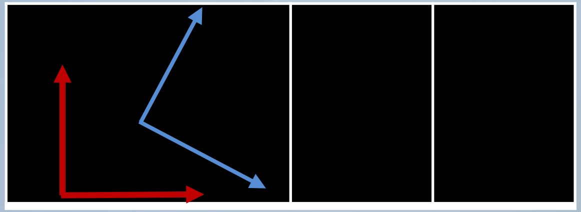
Draw With Same Object Model

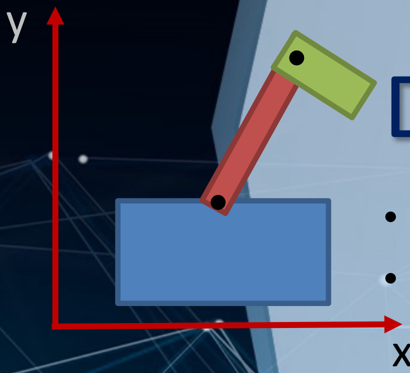
- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**



```
Translate($tx_1, ty_1$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1$)
Translate($tx_3, ty_3$)
pushMatrix()
Scale($sx_2, sy_2$)
Draw width =1 height = 1 red block
```

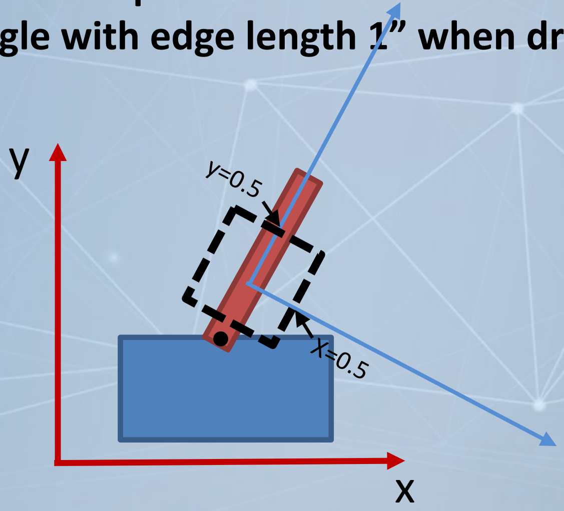
Stack



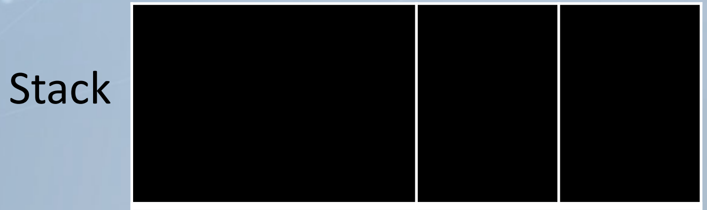


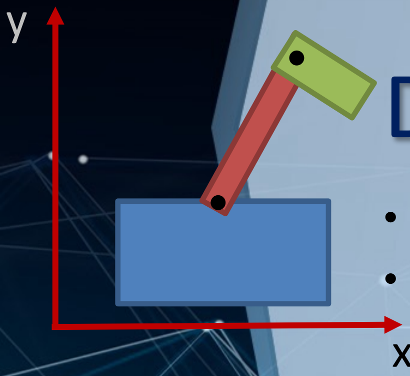
Draw With Same Object Model

- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**



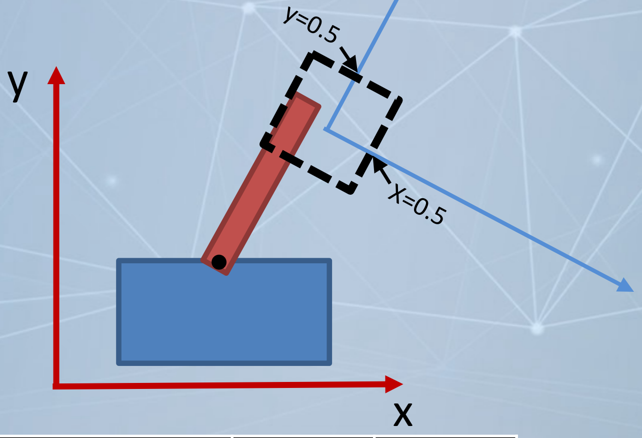
```
Translate($tx_1, ty_1$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1$)
Translate($tx_3, ty_3$)
pushMatrix()
Scale($sx_2, sy_2$)
Draw width =1 height = 1 red block
popMatrix()
```





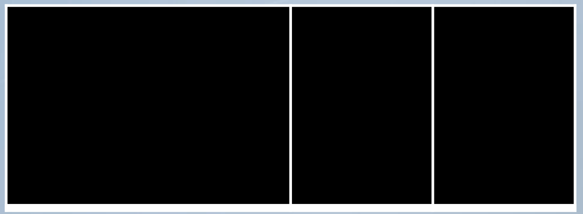
Draw With Same Object Model

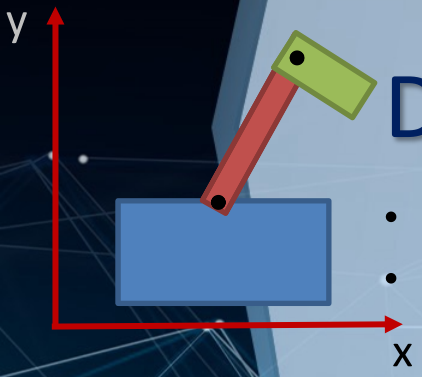
- We skip some details – the detailed information to draw each block
- **Let's make the question harder – what if we always say "we want to draw a rectangle with edge length 1" when drawing blocks**



```
Translate($tx_1, ty_1$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1$)
Translate($tx_3, ty_3$)
pushMatrix()
Scale($sx_2, sy_2$)
Draw width =1 height = 1 red block
popMatrix()
Translate($tx_4, ty_4$)
```

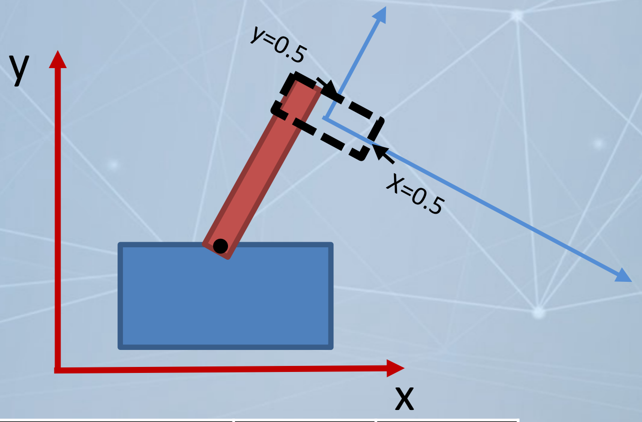
Stack





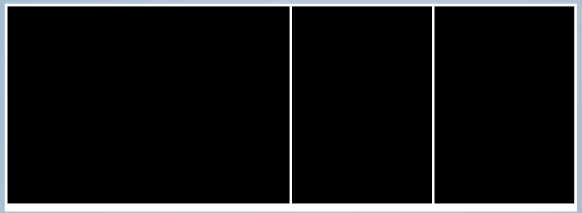
Draw With Same Object Model

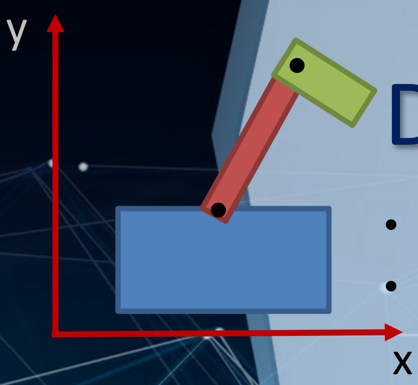
- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**



```
Translate($tx_1, ty_1$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1$)
Translate($tx_3, ty_3$)
pushMatrix()
Scale($sx_2, sy_2$)
Draw width =1 height = 1 red block
popMatrix()
Translate($tx_4, ty_4$)
Scale($sx_3, sy_3$)
```

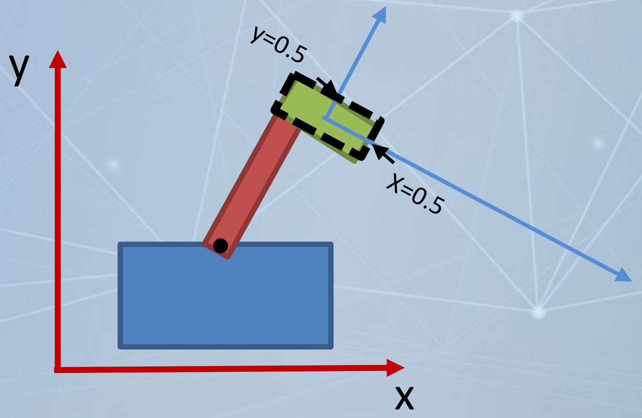
Stack





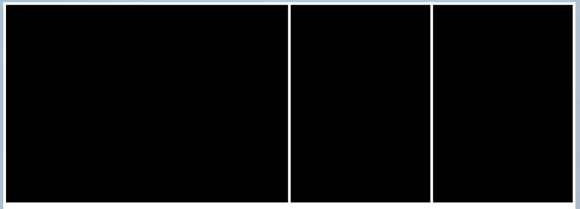
Draw With Same Object Model

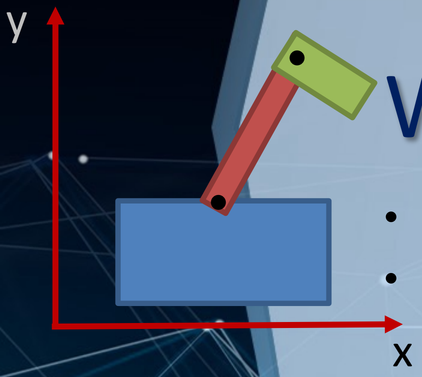
- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**



```
Translate($tx_1, ty_1$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1$)
Translate($tx_3, ty_3$)
pushMatrix()
Scale($sx_2, sy_2$)
Draw width =1 height = 1 red block
popMatrix()
Translate($tx_4, ty_4$)
Scale($sx_3, sy_3$)
Draw width =1 height = 1 green block
```

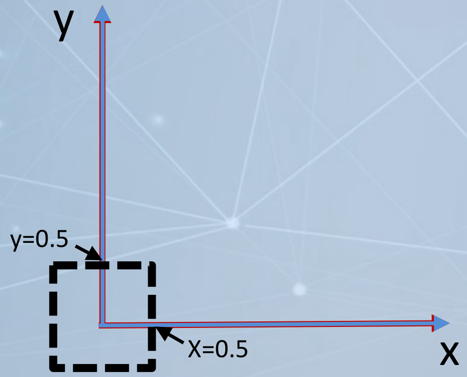
Stack



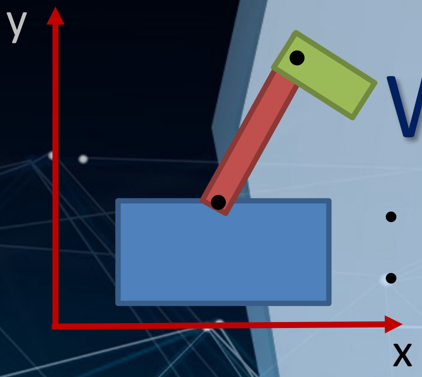


What Happen Mathematically?

- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**

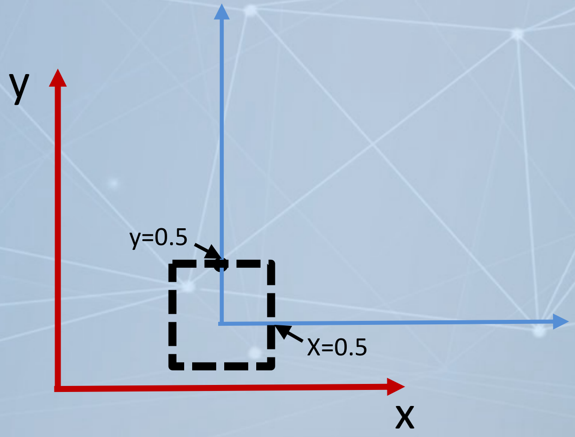


Initially, we have an identity matrix (I)

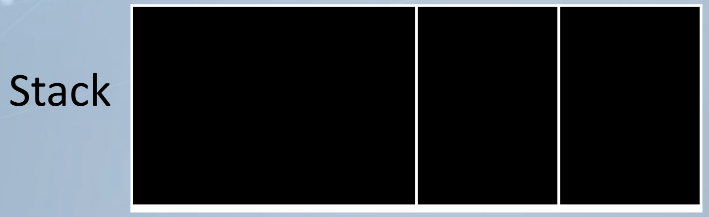


What Happen Mathematically?

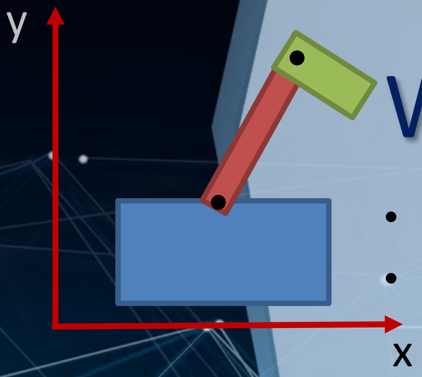
- We skip some details – the detailed information to draw each block
- **Let's make the question harder – what if we always say "we want to draw a rectangle with edge length 1" when drawing blocks**



Translate(tx_1, ty_1) → $\boldsymbol{M}_{\{tx_1,ty_1\}}$

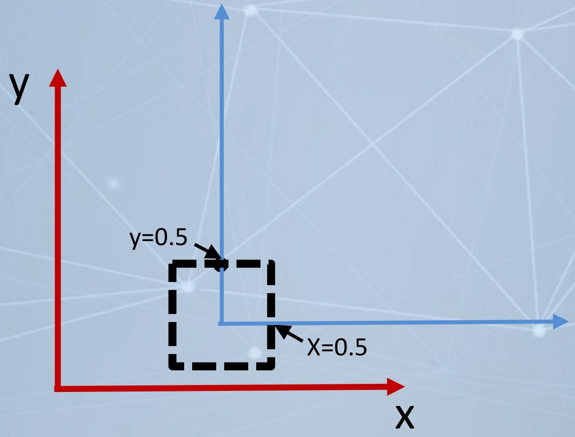


$I * \boldsymbol{M}_{\{tx_1,ty_1\}}$



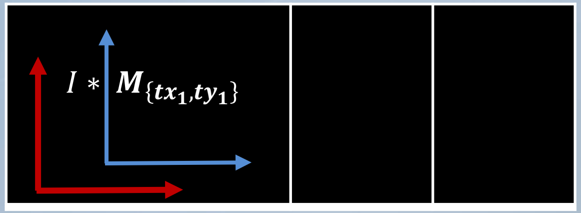
What Happen Mathematically?

- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**

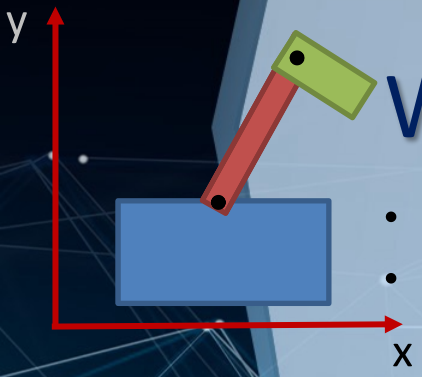


Translate(tx_1, ty_1)
pushMatrix() → Keep current state in stack

Stack

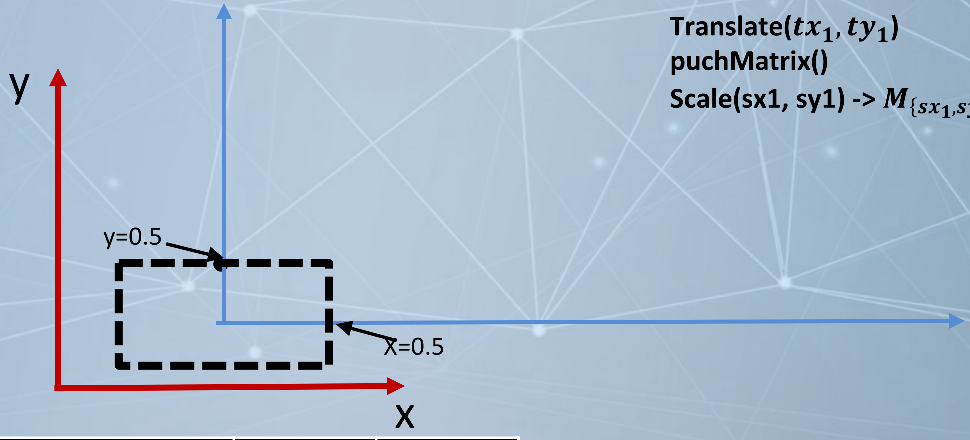


$I * M_{\{tx_1,ty_1\}}$



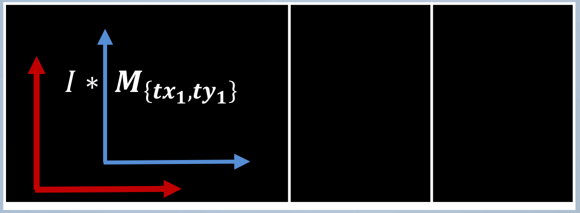
What Happen Mathematically?

- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**



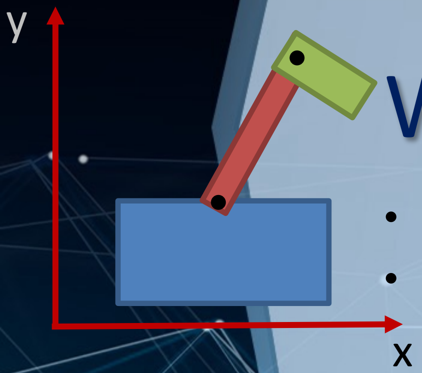
Translate(tx_1, ty_1)
puchMatrix()
Scale(sx1, sy1) -> $M_{\{sx_1,sy_1\}}$

Stack



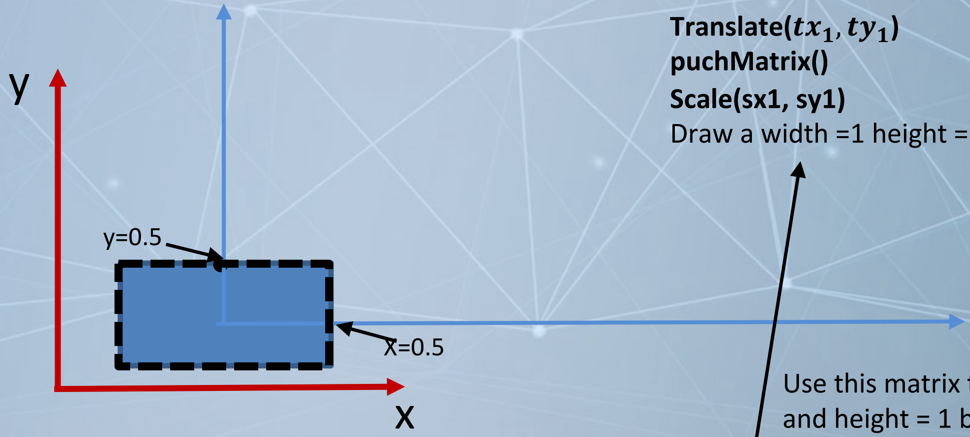
$I * M_{\{tx_1,ty_1\}}$

$I * M_{\{tx_1,ty_1\}} * M_{\{sx_1,sy_1\}}$



What Happen Mathematically?

- We skip some details – the detailed information to draw each block
- **Let's make the question harder – what if we always say "we want to draw a rectangle with edge length 1" when drawing blocks**

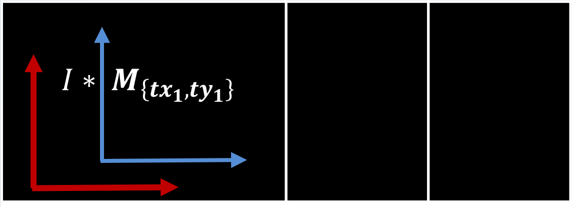


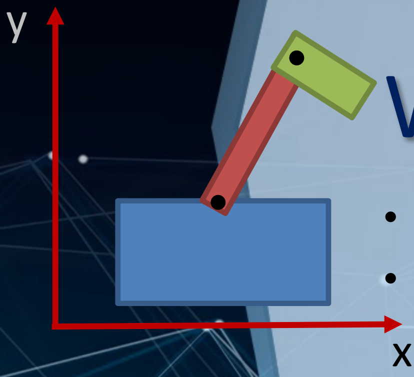
Translate(tx_1, ty_1)
puchMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block

Use this matrix to transform a width=1 and height = 1 blue block and draw

$I * \boldsymbol{M}_{\{tx_1,ty_1\}} * \boldsymbol{M}_{\{sx_1,sy_1\}}$

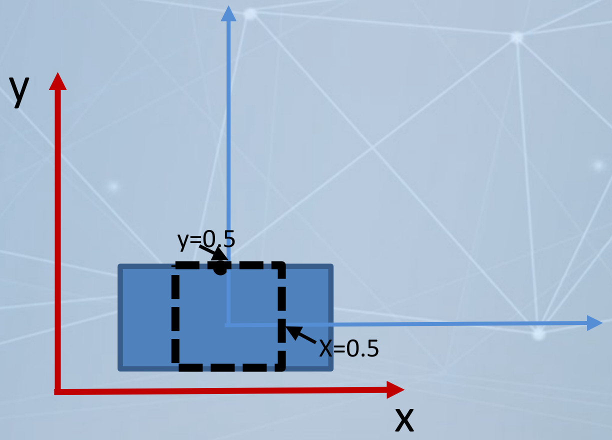
Stack





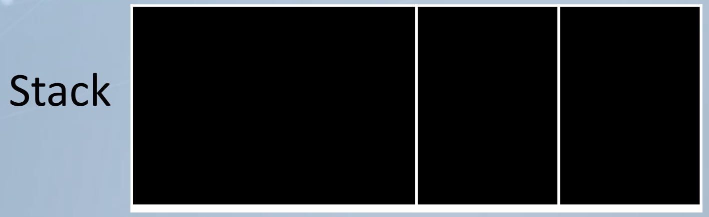
What Happen Mathematically?

- We skip some details – the detailed information to draw each block
- **Let's make the question harder – what if we always say "we want to draw a rectangle with edge length 1" when drawing blocks**

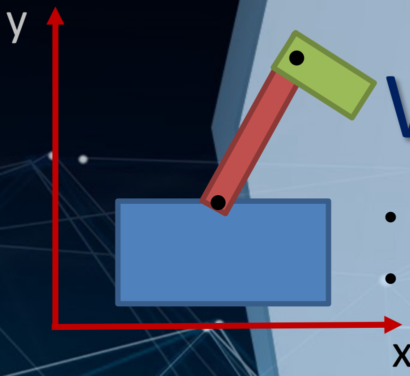


Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()

Restore the state from stack

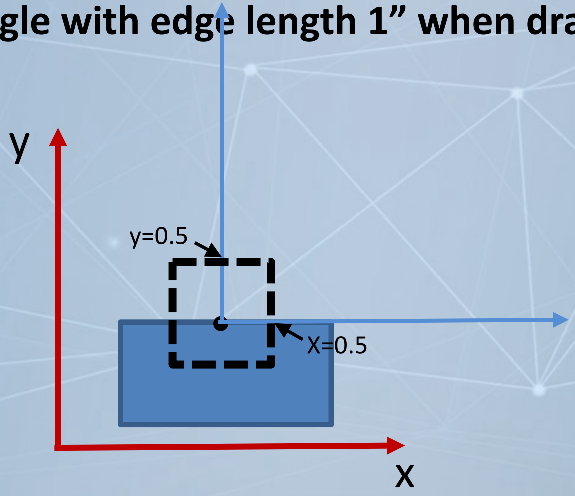


$I * M_{\{tx_1,ty_1\}}$



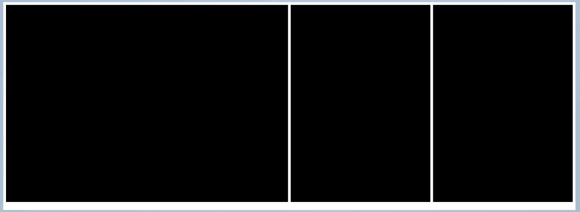
What Happen Mathematically?

- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**

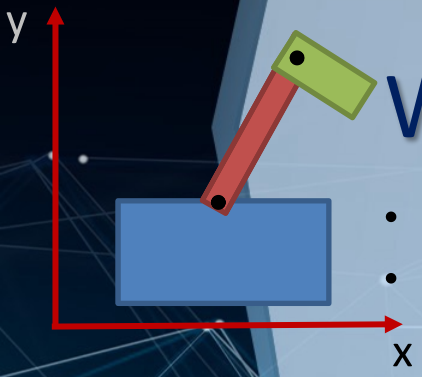


Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2) -> $M_{\{tx_2,ty_2\}}$

Stack

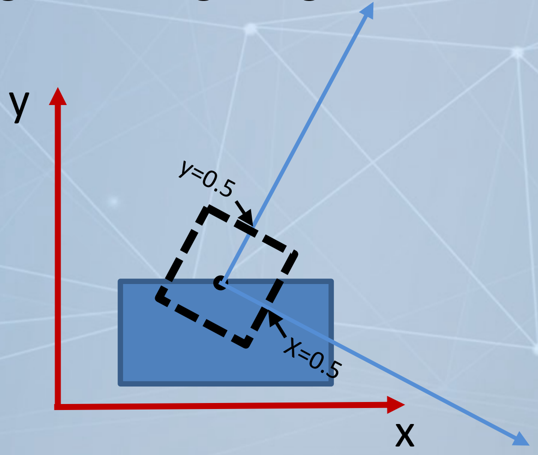


$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}}$$



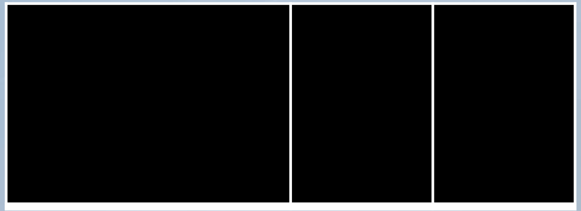
What Happen Mathematically?

- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**

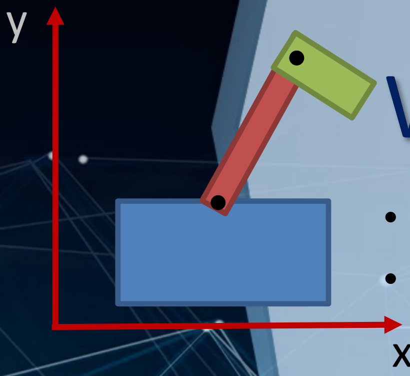


Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1) -> $M_{\{\theta_1\}}$

Stack

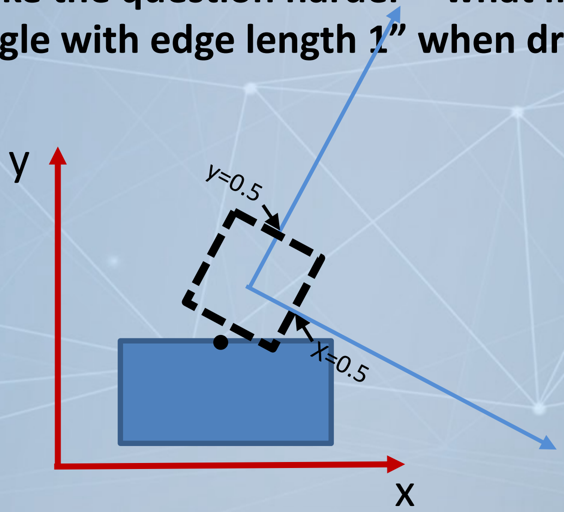


$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}} * M_{\{\theta_1\}}$$



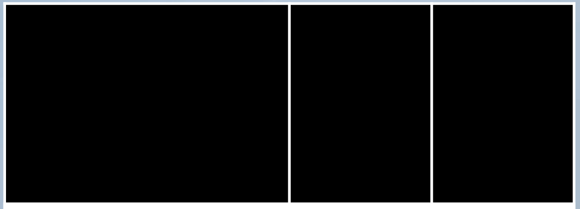
What Happen Mathematically?

- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**

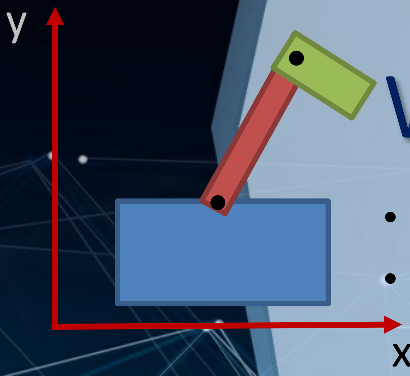


Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1)
Translate(tx_3, ty_3) -> $M_{\{tx_3,ty_3\}}$

Stack

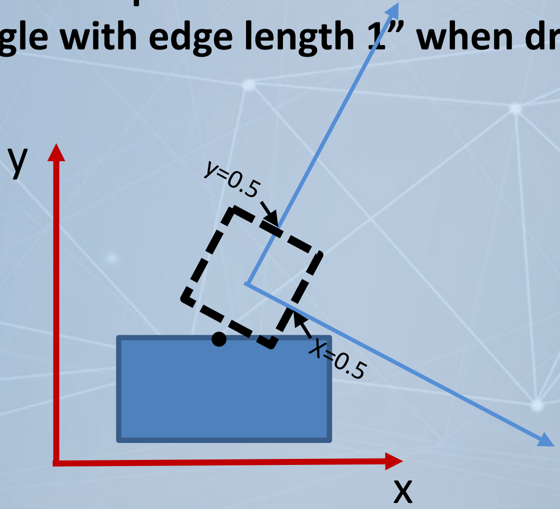


$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}} * M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}$$



What Happen Mathematically?

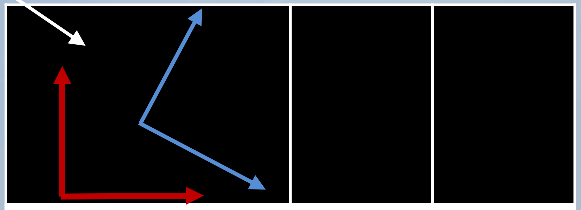
- We skip some details – the detailed information to draw each block
- **Let's make the question harder – what if we always say "we want to draw a rectangle with edge length 1" when drawing blocks**



```
Translate($tx_1, ty_1$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1$)
Translate($tx_3, ty_3$)
pushMatrix()
```

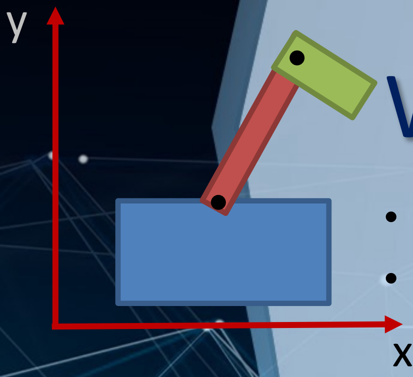
$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}}{}^{*} M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}$

Stack

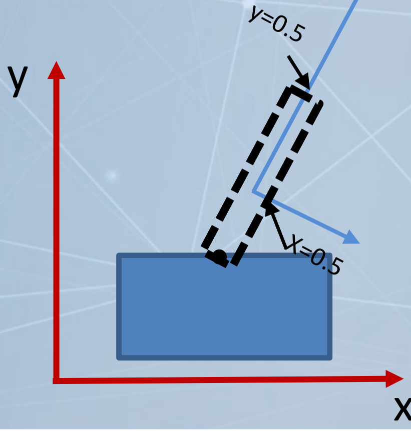


$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}}{}^{*} M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}$

What Happen Mathematically?



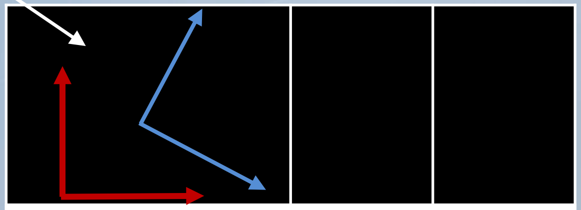
- We skip some details – the detailed information to draw each block
- **Let's make the question harder – what if we always say "we want to draw a rectangle with edge length 1" when drawing blocks**



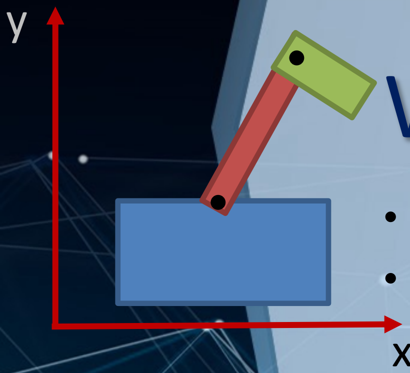
Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1)
Translate(tx_3, ty_3)
pushMatrix()
Scale(sx_2, sy_2) -> $M_{\{sx_2,sy_2\}}$

$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}}{}^* M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}$$

Stack

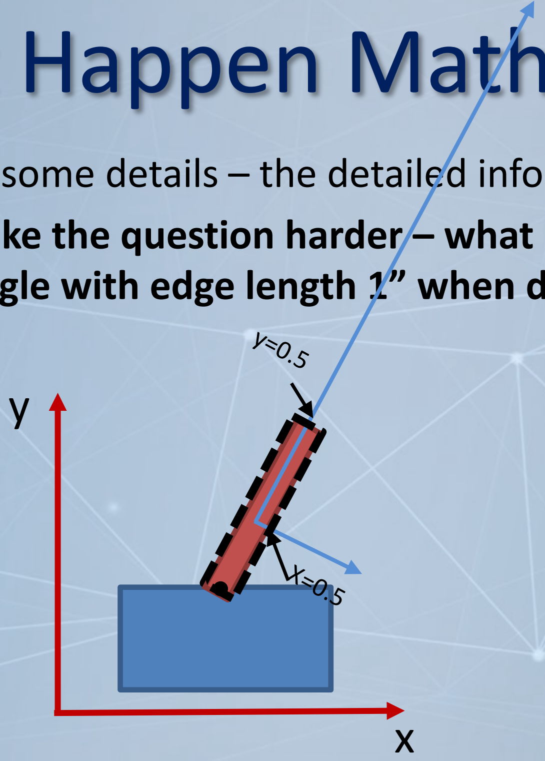


$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}}{}^* M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}{}^* M_{\{sx_2,sy_2\}}$$



What Happen Mathematically?

- We skip some details – the detailed information to draw each block
- **Let's make the question harder – what if we always say "we want to draw a rectangle with edge length 1" when drawing blocks**



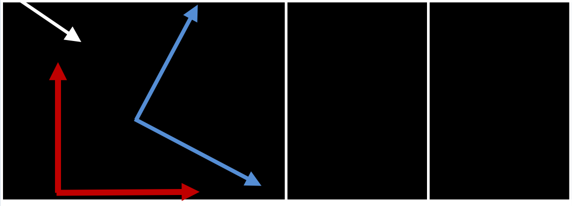
```
Translate($tx_1, ty_1$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1$)
Translate($tx_3, ty_3$)
pushMatrix()
Scale($sx_2, sy_2$)
Draw width =1 height = 1 red block
```

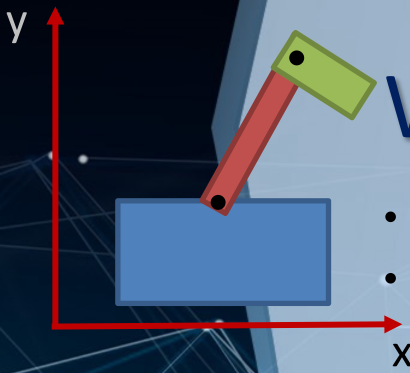
Use this matrix to transform a width=1 and height = 1 red block and draw

$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}}{}^{*} M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}{}^{*} M_{\{sx_2,sy_2\}}$$

$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}}{}^{*} M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}$$

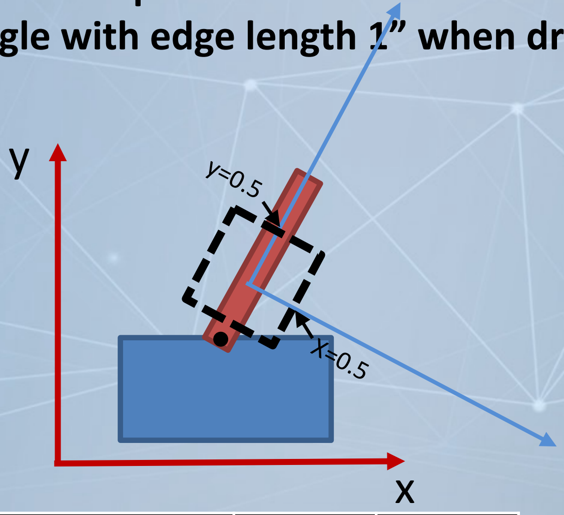
Stack





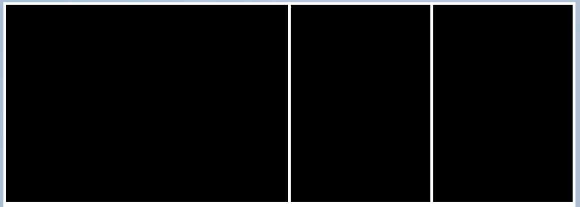
What Happen Mathematically?

- We skip some details – the detailed information to draw each block
- **Let's make the question harder – what if we always say "we want to draw a rectangle with edge length 1" when drawing blocks**

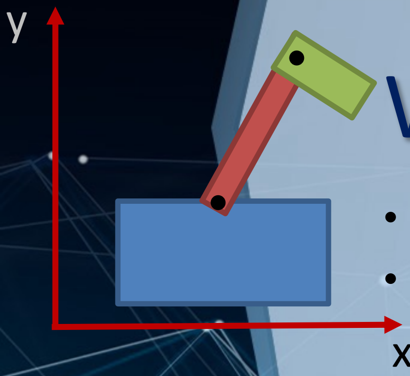


Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1)
Translate(tx_3, ty_3)
pushMatrix()
Scale(sx_2, sy_2)
Draw width =1 height = 1 red block
popMatrix()

Stack

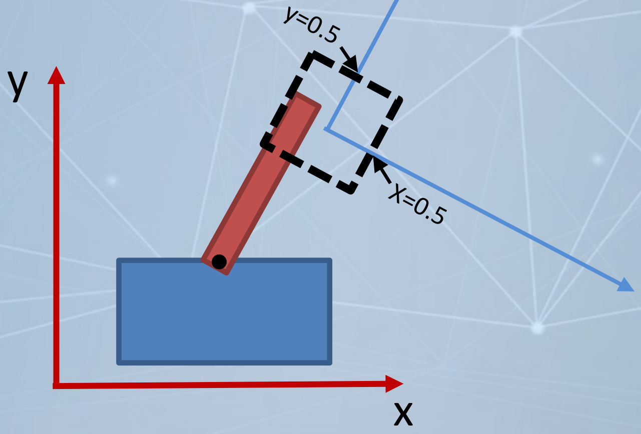


$$I * \boldsymbol{M}_{\{tx_1,ty_1\}} * \boldsymbol{M}_{\{tx_2,ty_2\}} * \boldsymbol{M}_{\{\theta_1\}} * \boldsymbol{M}_{\{tx_3,ty_3\}}$$



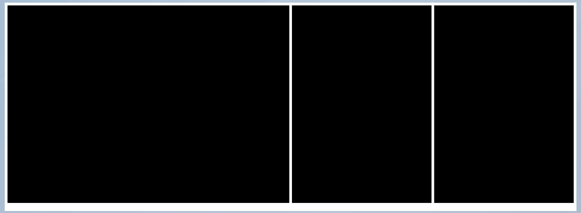
What Happen Mathematically?

- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**

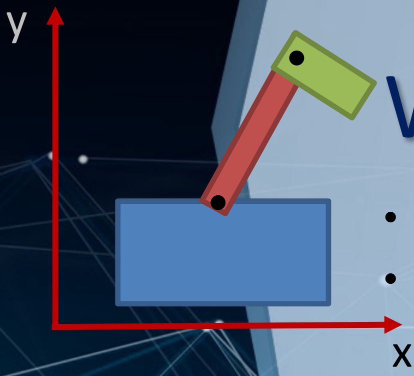


```
Translate($tx_1, ty_1$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1$)
Translate($tx_3, ty_3$)
pushMatrix()
Scale($sx_2, sy_2$)
Draw width =1 height = 1 red block
popMatrix()
Translate($tx_4, ty_4$) -> $M_{\{tx_4,ty_4\}}$
```

Stack

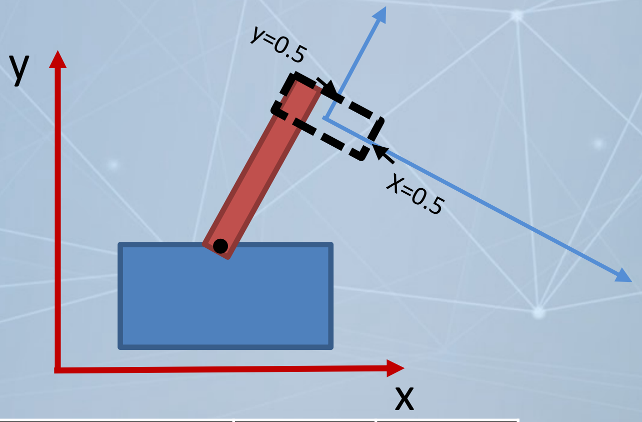


$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}} * M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}} * M_{\{tx_4,ty_4\}}$$

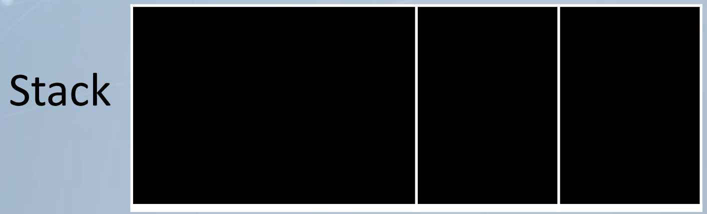


What Happen Mathematically?

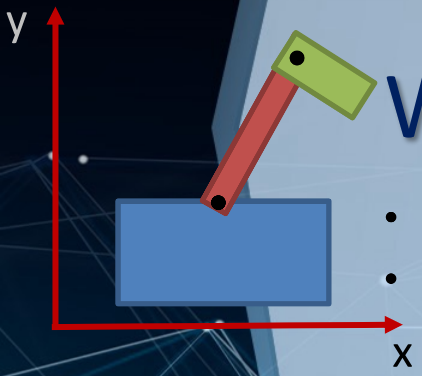
- We skip some details – the detailed information to draw each block
- **Let's make the question harder – what if we always say "we want to draw a rectangle with edge length 1" when drawing blocks**



```
Translate($tx_1, ty_1$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1$)
Translate($tx_3, ty_3$)
pushMatrix()
Scale($sx_2, sy_2$)
Draw width =1 height = 1 red block
popMatrix()
Translate($tx_3, ty_3$)
Scale($sx_3, sy_3$) -> $M_{\{sx_3,sy_3\}}$
```

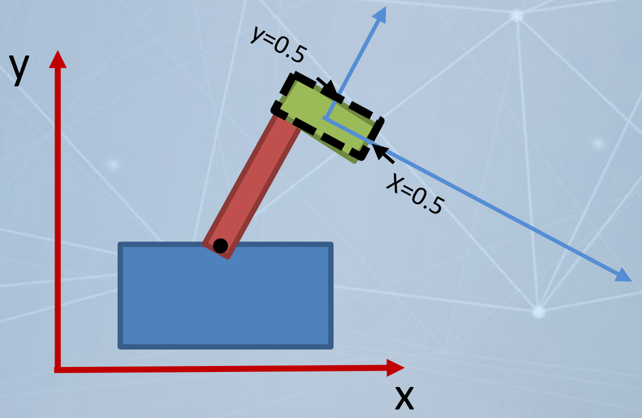


$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}} * M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}} * M_{\{tx_4,ty_4\}} * M_{\{sx_3,sy_3\}}$$



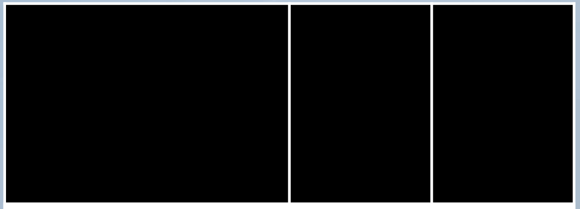
What Happen Mathematically?

- We skip some details – the detailed information to draw each block
- **Let’s make the question harder – what if we always say “we want to draw a rectangle with edge length 1” when drawing blocks**



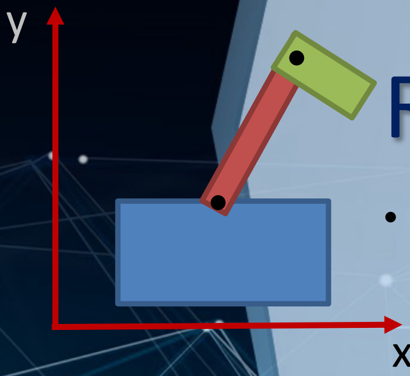
```
Translate($tx_1, ty_1$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1$)
Translate($tx_3, ty_3$)
pushMatrix()
Scale($sx_2, sy_2$)
Draw width =1 height = 1 red block
popMatrix()
Translate($tx_3, ty_3$)
Scale($sx_3, sy_3$)
Draw width =1 height = 1 green block
```

Stack



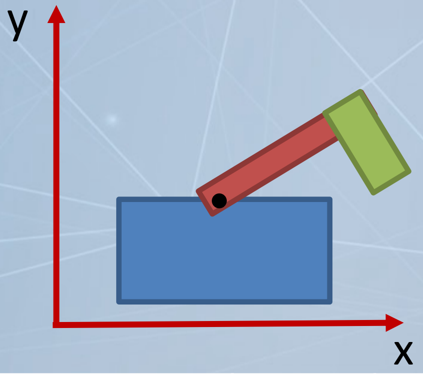
Use this matrix to transform a width=1 and height = 1 green block and draw

$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}}{}^* M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}{}^* M_{\{tx_4,ty_4\}}{}^* M_{\{sx_3,sy_3\}}$$

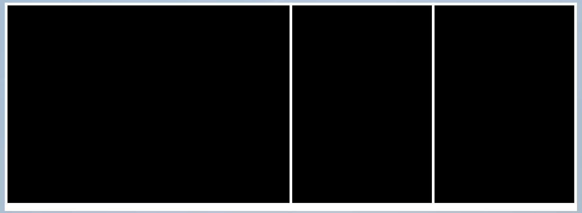


Rotate the Red Arm?

- Simply change Rotate($\boldsymbol{\theta_1}$)



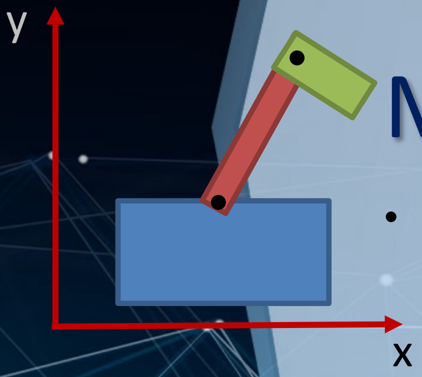
Stack



```
Translate($tx_1, ty_1$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1 + \nabla\theta$)
Translate($tx_3, ty_3$)
pushMatrix()
Scale($sx_2, sy_2$)
Draw width =1 height = 1 red block
popMatrix()
Translate($tx_3, ty_3$)
Scale($sx_3, sy_3$)
Draw width =1 height = 1 green block
```

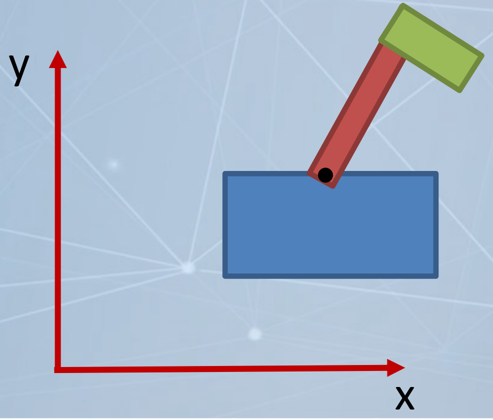
$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}} {}^{*} M_{\{\theta_1+\nabla\theta\}} * M_{\{tx_3,ty_3\}} {}^{*} M_{\{tx_4,ty_4\}} {}^{*} M_{\{sx_3,sy_3\}}$$

Transformation matrix to draw the green component

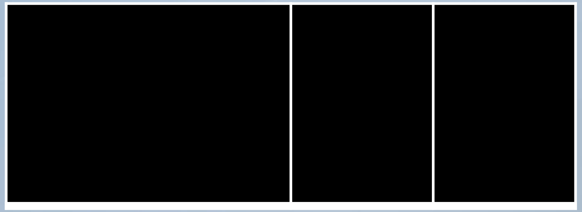


Move the Robot

- Add a translation operation in the beginning



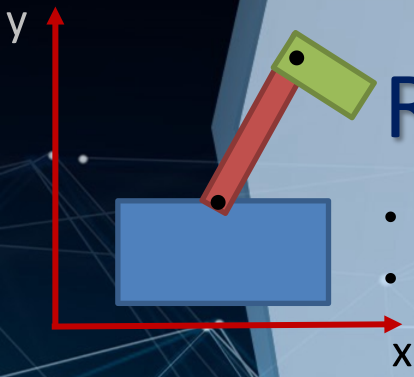
Stack



```
Translate($tx_{rm}, ty_{rm}$)
Translate($tx_1, ty_1$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1$)
Translate($tx_3, ty_3$)
pushMatrix()
Scale($sx_2, sy_2$)
Draw width =1 height = 1 red block
popMatrix()
Translate($tx_3, ty_3$)
Scale($sx_3, sy_3$)
Draw width =1 height = 1 green block
```

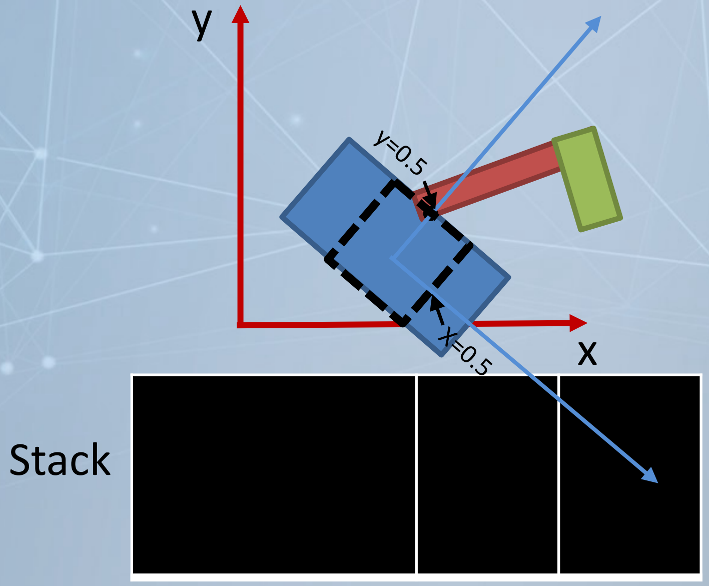
$$I * M_{\{tx_{rm}, ty_{rm}\}} * M_{\{tx_1, ty_1\}} * M_{\{tx_2, ty_2\}} * M_{\{\theta_1\}} * M_{\{tx_3, ty_3\}} * M_{\{tx_4, ty_4\}} * M_{\{sx_3, sy_3\}}$$

Transformation matrix to draw the green component



Rotate the whole Robot

- Add a rotation operation
- Where to insert the rotation operation?



```
Translate($tx_1, ty_1$)
Rotate($\theta_{rr}$)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate($tx_2, ty_2$)
Rotate($\theta_1$)
Translate($tx_3, ty_3$)
pushMatrix()
Scale($sx_2, sy_2$)
Draw width =1 height = 1 red block
popMatrix()
Translate($tx_3, ty_3$)
Scale($sx_3, sy_3$)
Draw width =1 height = 1 green block
```

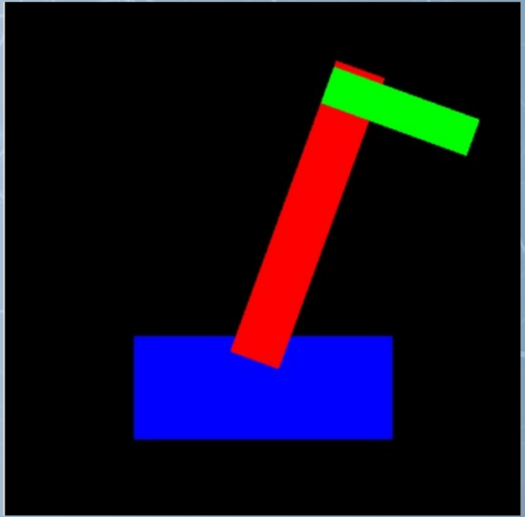
$$I * \boldsymbol{M}_{\{tx_1,ty_1\}} * \boldsymbol{M}_{\{\theta_{rr}\}} * \boldsymbol{M}_{\{tx_2,ty_2\}} * \boldsymbol{M}_{\{\theta_1\}} * \boldsymbol{M}_{\{tx_3,ty_3\}} * \boldsymbol{M}_{\{tx_4,ty_4\}} * \boldsymbol{M}_{\{sx_3,sy_3\}}$$

Transformation matrix to draw the green component

Example (Ex04-1)

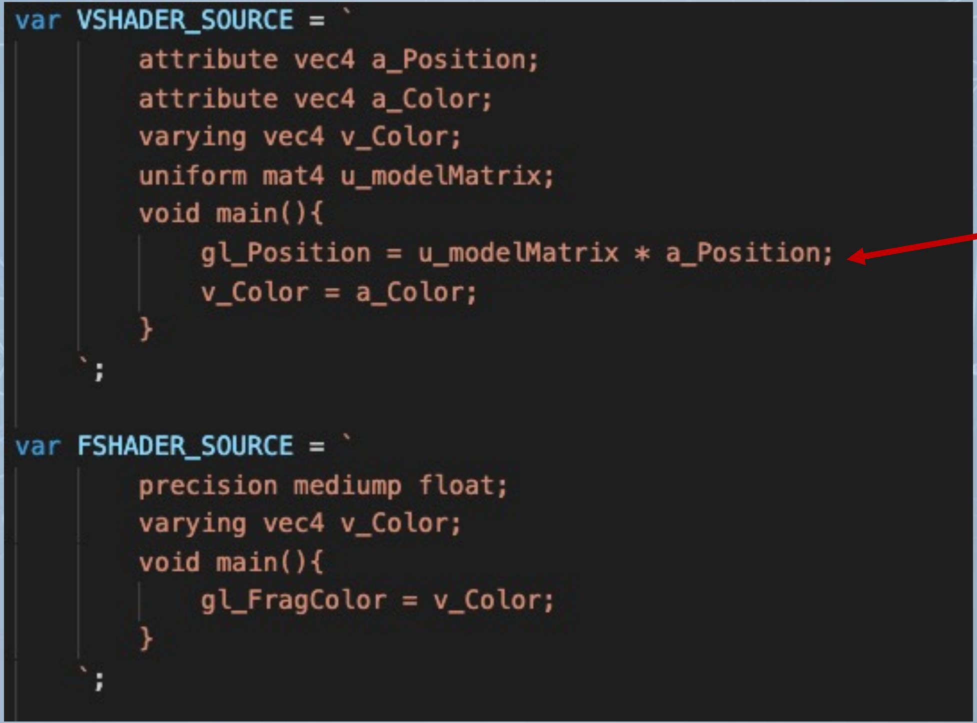
- Let’s repeat the same operations in code

- Files
  - Index.html
  - WebGL.js
  - Cuon-matrix.js



Example (Ex04-1)

- Shaders (same as Ex03-2)



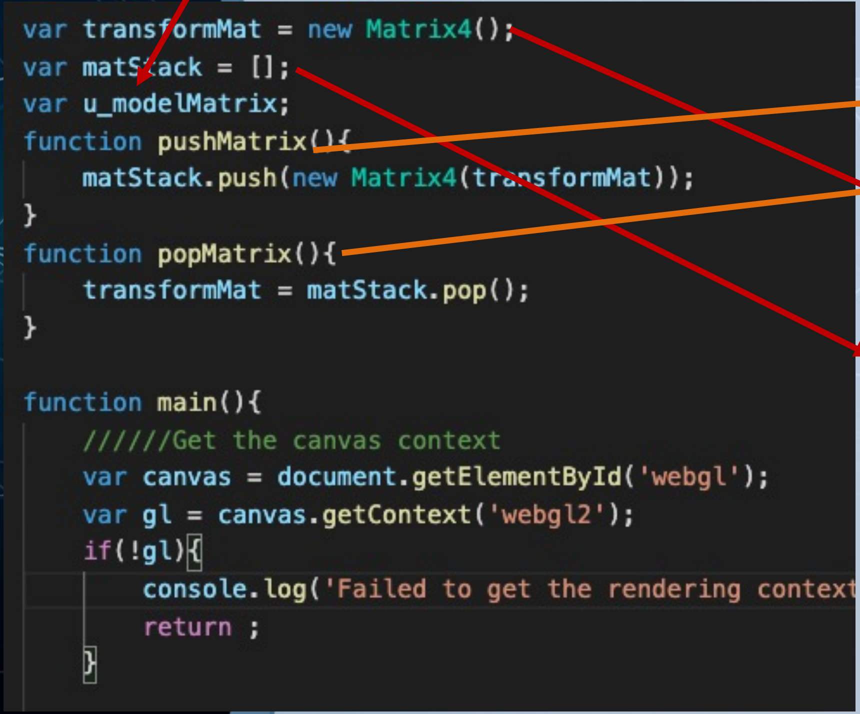
```
var VSHADER_SOURCE = `
        attribute vec4 a_Position;
        attribute vec4 a_Color;
        varying vec4 v_Color;
        uniform mat4 u_modelMatrix;
        void main(){
            gl_Position = u_modelMatrix * a_Position;
            v_Color = a_Color;
        }
    `;

var FSHADER_SOURCE = `
        precision mediump float;
        varying vec4 v_Color;
        void main(){
            gl_FragColor = v_Color;
        }
    `;
```

Use u_modelMatrix to transform

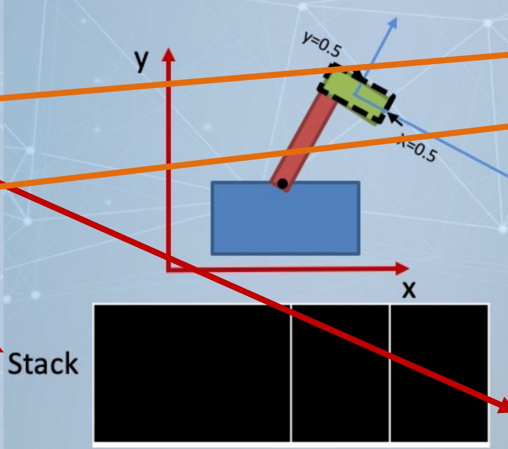
Reference which points to u_modelMatrix in shader

Example (Ex04-1)

- WebGL.js



```
var transformMat = new Matrix4();
var matStack = [];
var u_modelMatrix;
function pushMatrix(){
    matStack.push(new Matrix4(transformMat));
}
function popMatrix(){
    transformMat = matStack.pop();
}

function main(){
    //////Get the canvas context
    var canvas = document.getElementById('webgl');
    var gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context
        return ;
    }
```



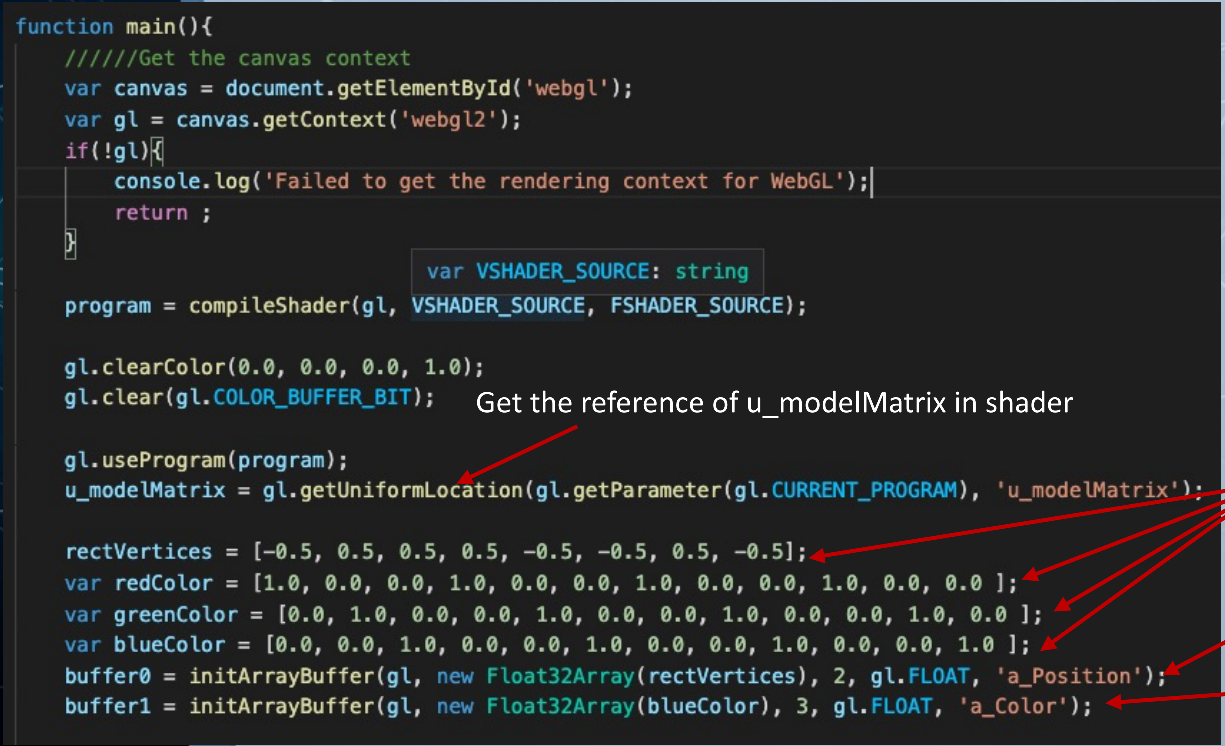
Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1)
Translate(tx_3, ty_3)
pushMatrix()
Scale(sx_2, sy_2)
Draw width =1 height = 1 red block
popMatrix()
Translate(tx_3, ty_3)
Scale(sx_3, sy_3)
Draw width =1 height = 1 green block

Use this matrix to transform a width=1 and height = 1 green block and draw

$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}} * M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}} * M_{\{tx_4,ty_4\}} * M_{\{sx_3,sy_3\}}$$

Example (Ex04-1)

- WebGL.js



```
function main(){
    //////Get the canvas context
    var canvas = document.getElementById('webgl');
    var gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);

    gl.useProgram(program);
    u_modelMatrix = gl.getUniformLocation(gl.getParameter(gl.CURRENT_PROGRAM), 'u_modelMatrix');

    rectVertices = [-0.5, 0.5, 0.5, 0.5, -0.5, -0.5, 0.5, -0.5];
    var redColor = [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0 ];
    var greenColor = [0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0 ];
    var blueColor = [0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0 ];
    buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
    buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');
```

Get the reference of u_modelMatrix in shader

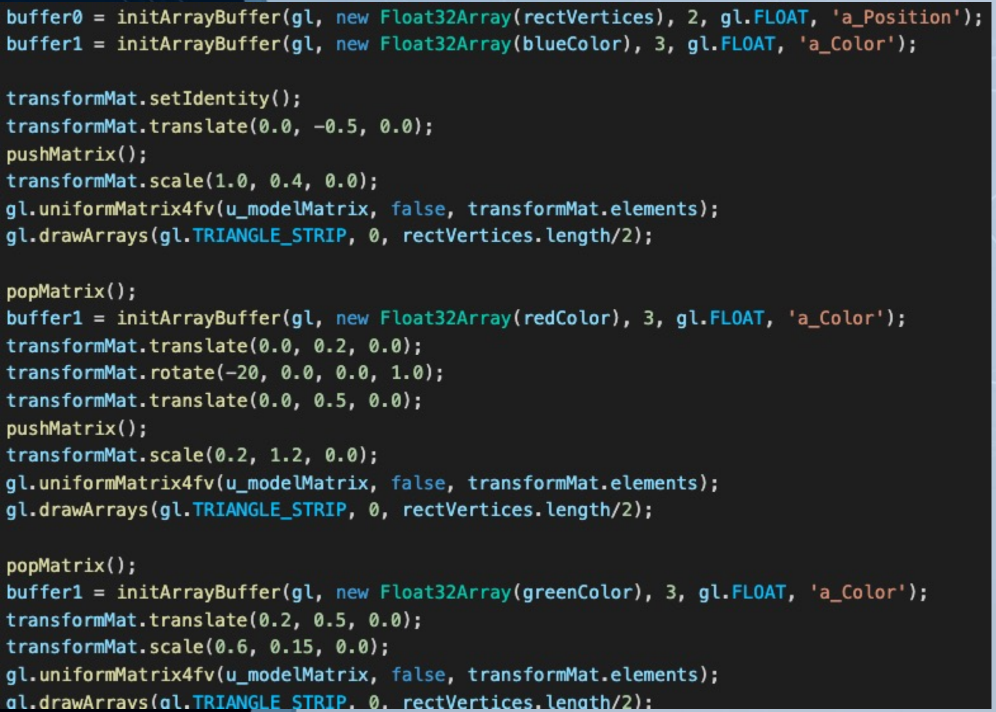
Setup a unit square and three different colors

Create and initialize VBO (vertex)

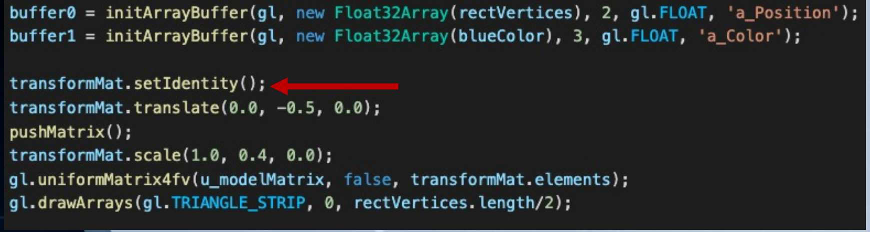
Create and initialize VBO (the first color we want)

Example (Ex04-1)

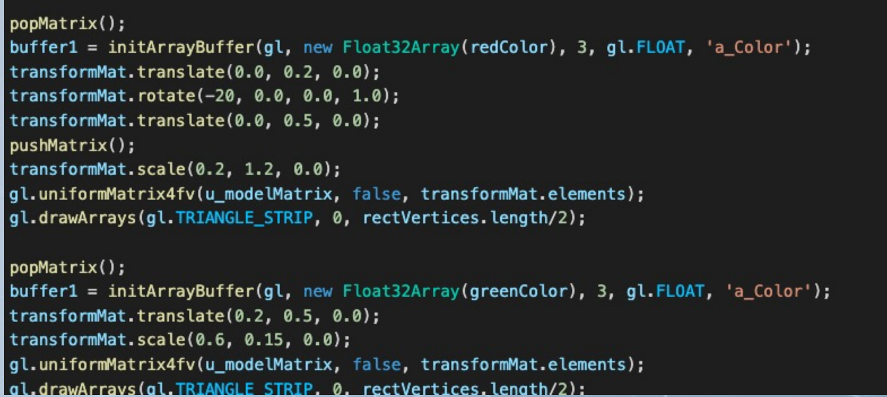
- WebGL.js



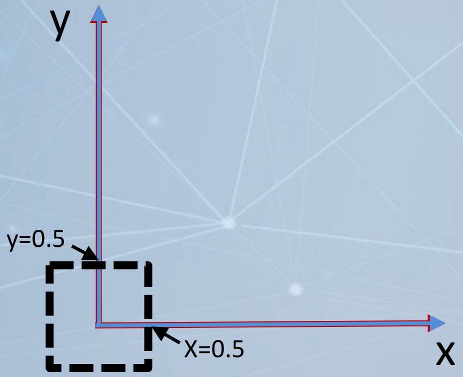
```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



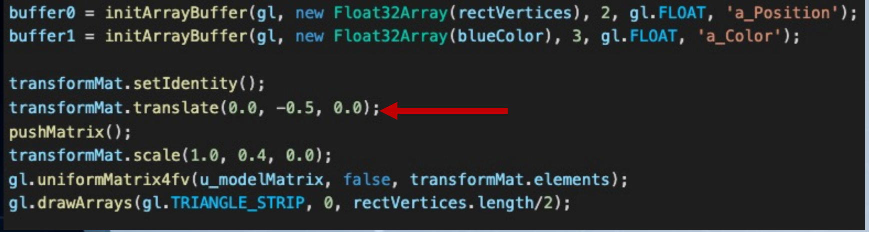
```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



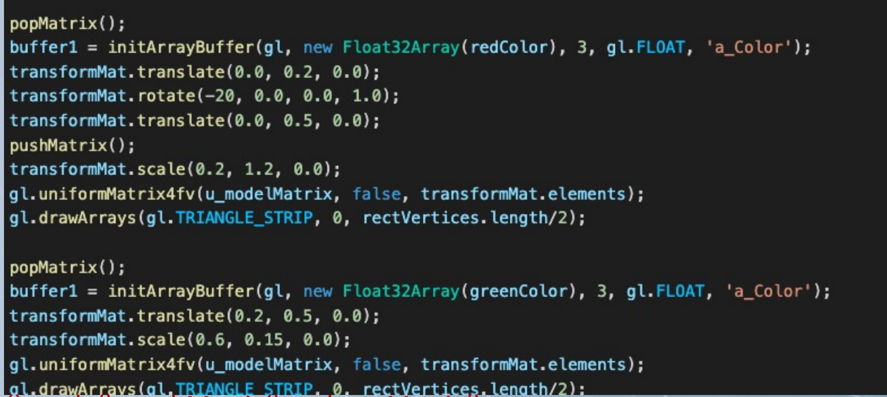
```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



Initially, we have an identity matrix (I)



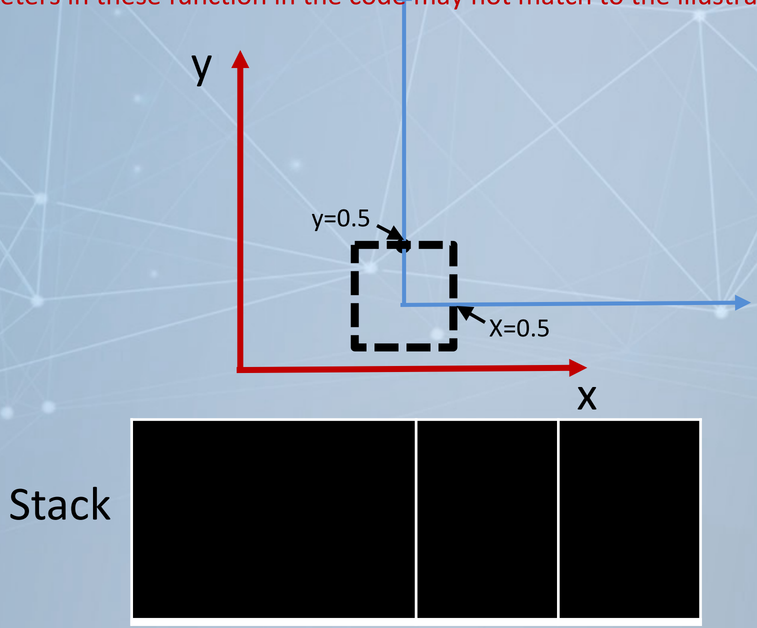
```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

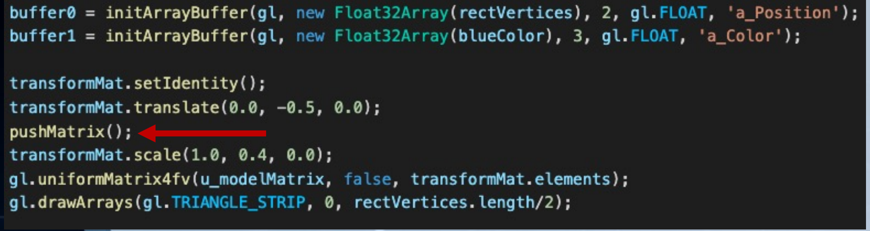
I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix(). The parameters in these function in the code may not match to the illustration.

Translate(tx_1, ty_1) → $M_{\{tx_1,ty_1\}}$

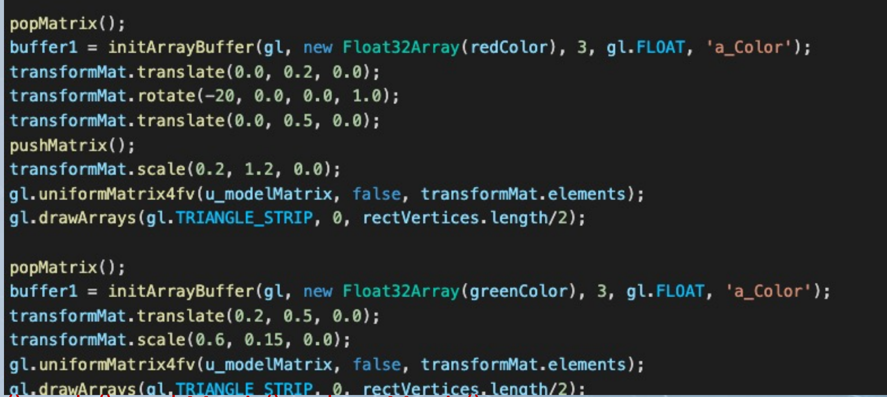


$I * M_{\{tx_1,ty_1\}}$

y

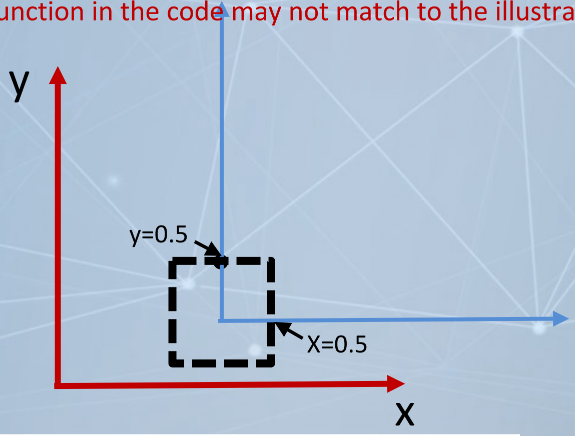


```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



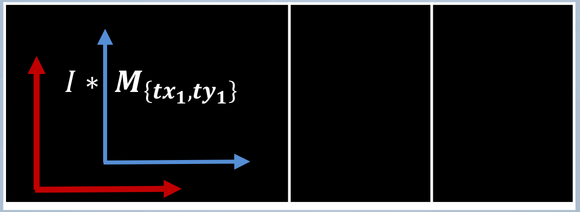
```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix().
The parameters in these function in the code may not match to the illustration.

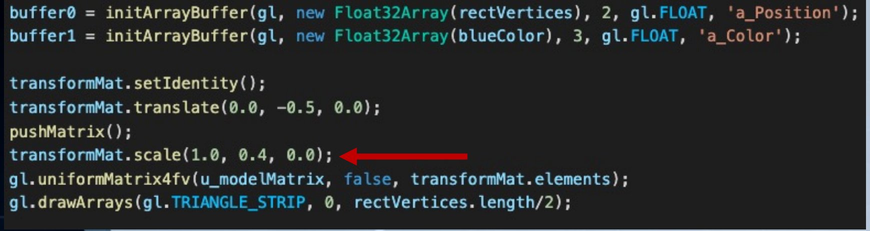


Translate(tx_1, ty_1)
pushMatrix() → Keep current state in stack

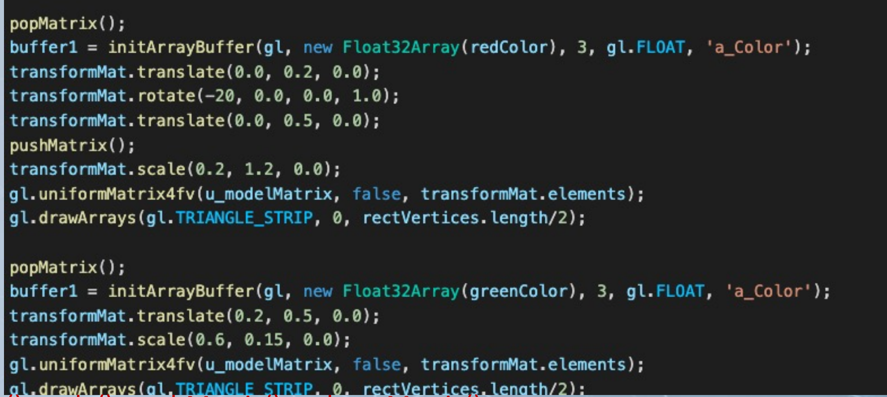
Stack



$I * M_{\{tx_1,ty_1\}}$



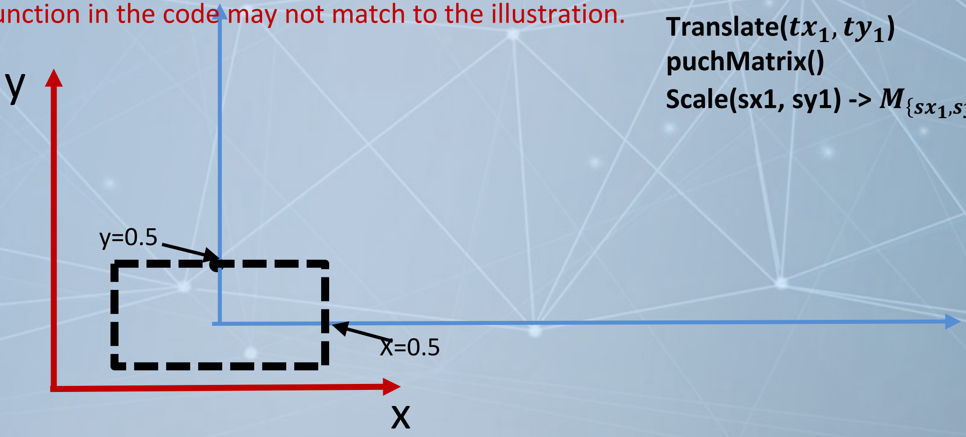
```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



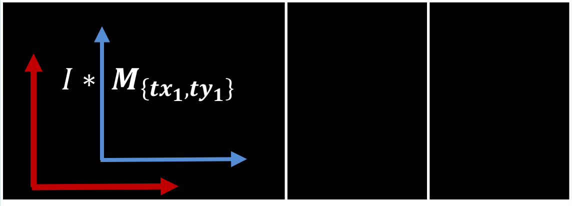
```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix().
The parameters in these function in the code may not match to the illustration.

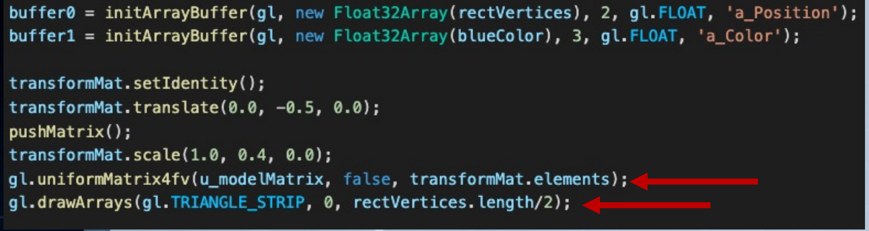
Translate(tx_1, ty_1)
puchMatrix()
Scale(sx1, sy1) -> $M_{\{sx_1,sy_1\}}$



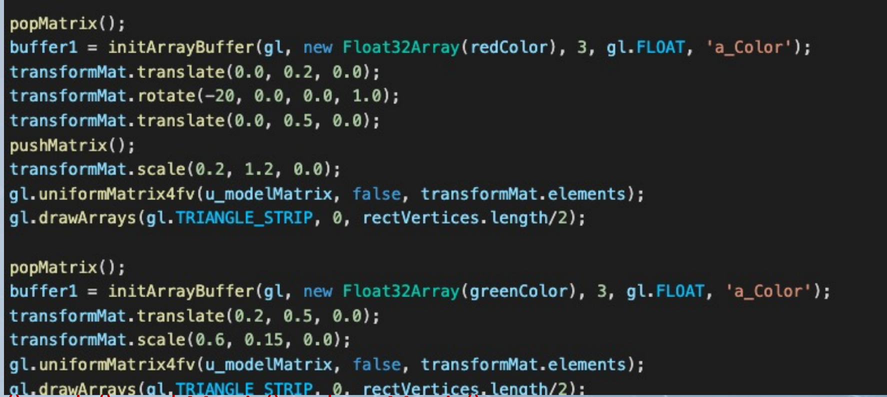
Stack



$I * M_{\{tx_1,ty_1\}} * M_{\{sx_1,sy_1\}}$

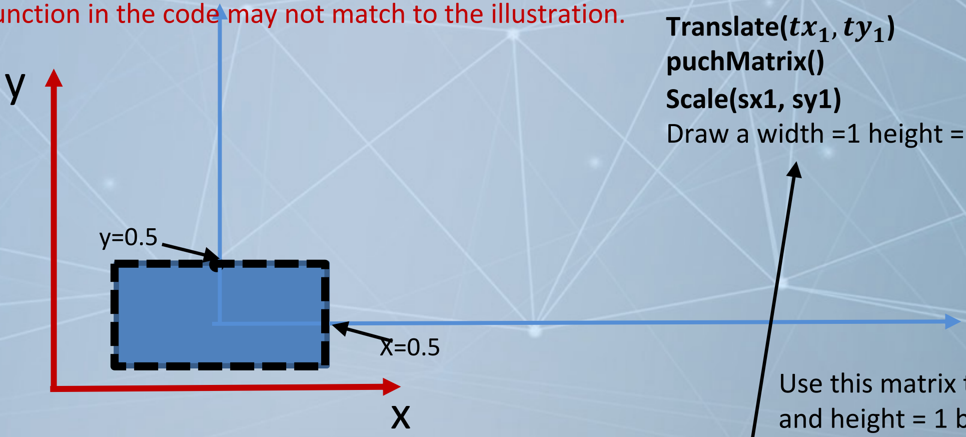


```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix(). The parameters in these function in the code may not match to the illustration.

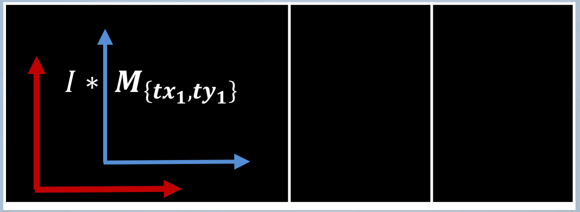


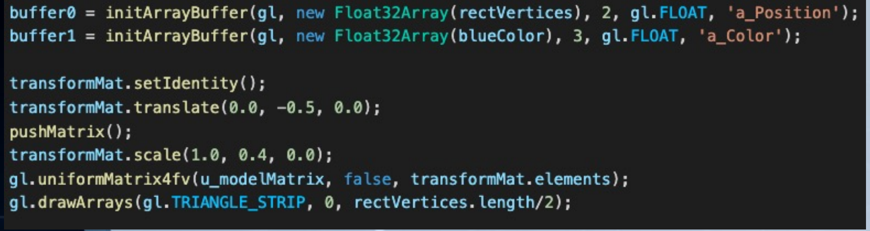
Translate(tx_1, ty_1)
puchMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block

Use this matrix to transform a width=1 and height = 1 blue block and draw

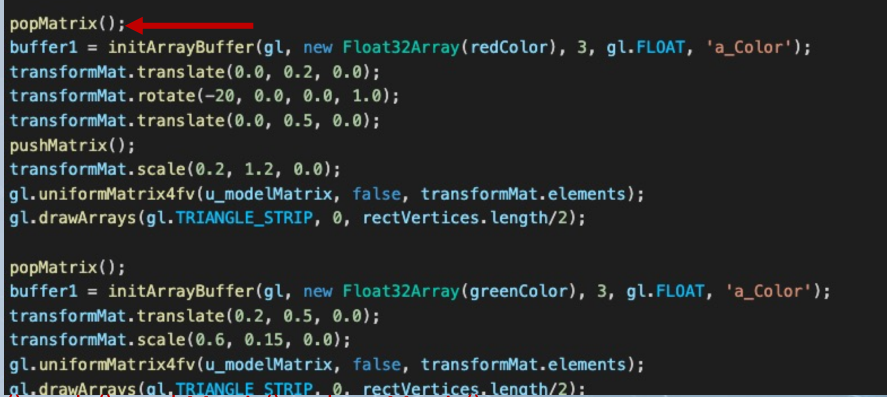
$$I * M_{\{tx_1,ty_1\}} * M_{\{sx_1,sy_1\}}$$

Stack



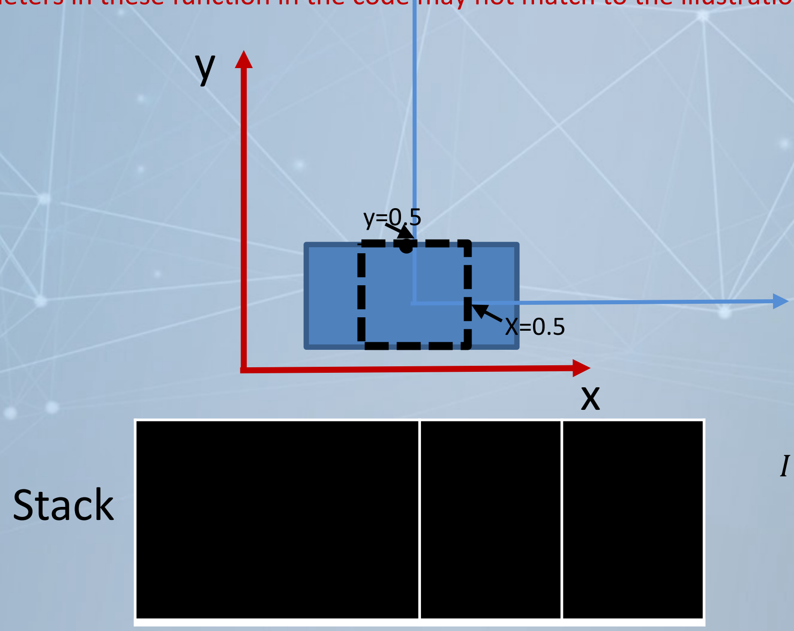


```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

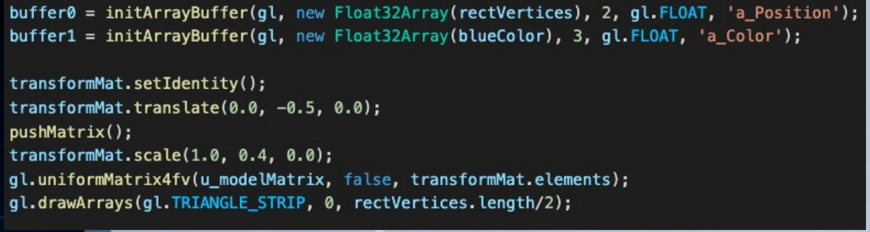
I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix().
The parameters in these function in the code may not match to the illustration.



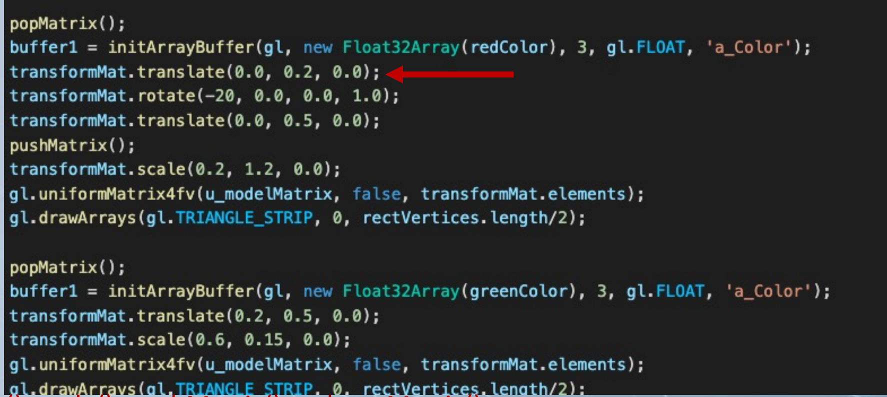
Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()

Restore the state from stack

$$I * M_{\{tx_1,ty_1\}}$$

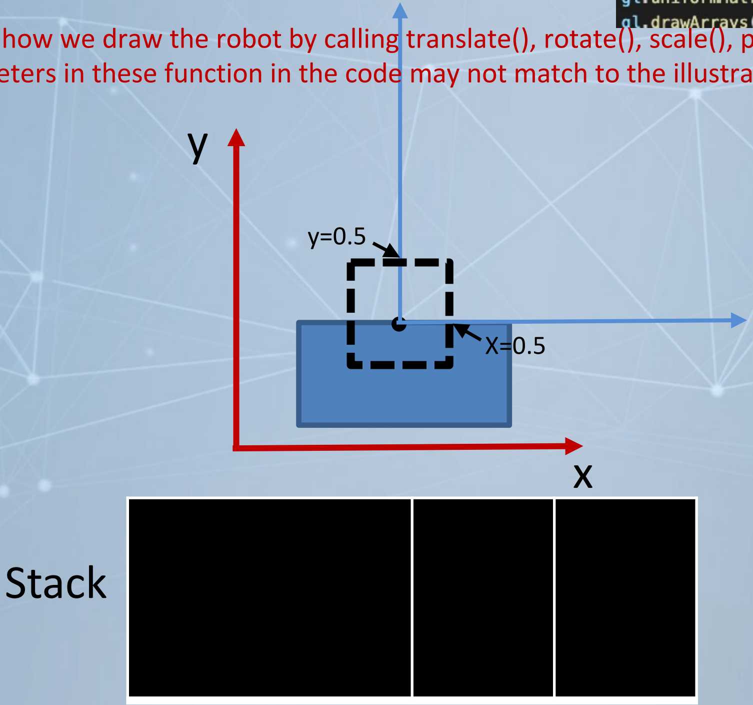


```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



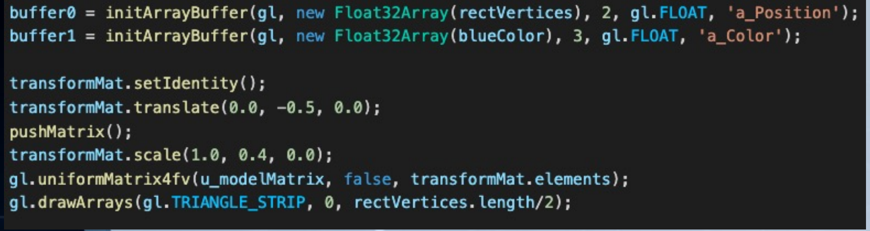
```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix().
The parameters in these function in the code may not match to the illustration.

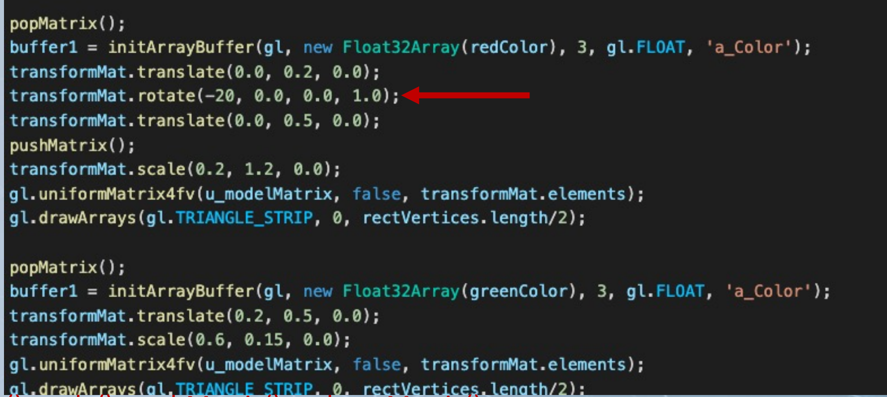


Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2) -> $M_{\{tx_2, ty_2\}}$

$$I * M_{\{tx_1, ty_1\}} * M_{\{tx_2, ty_2\}}$$

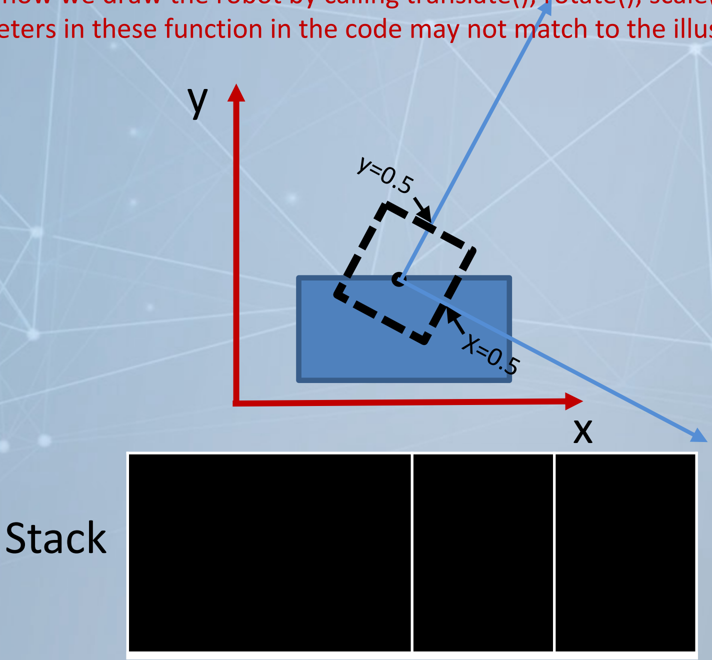


```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix(). The parameters in these function in the code may not match to the illustration.



Translate(tx_1, ty_1)

pushMatrix()

Scale(sx1, sy1)

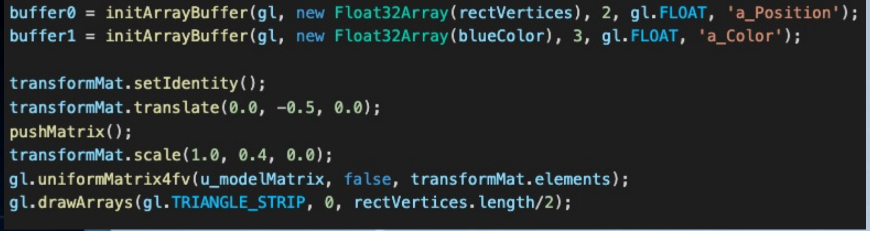
Draw a width =1 height = 1 blue block

popMatrix()

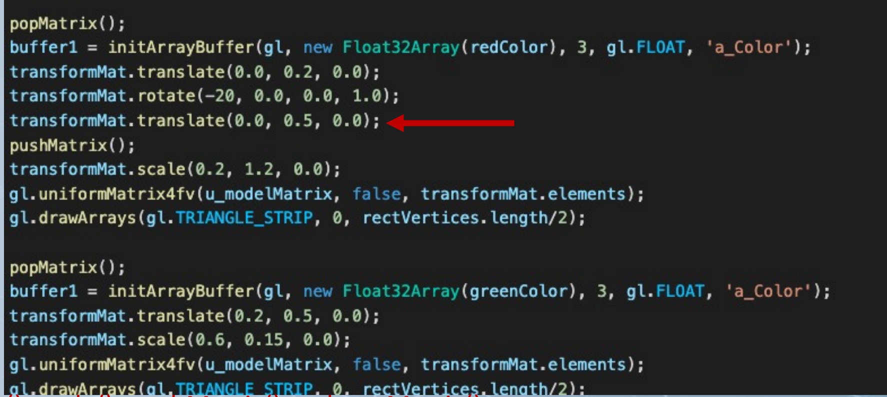
Translate(tx_2, ty_2)

Rotate(θ_1) -> $M_{\{\theta_1\}}$

$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}} * M_{\{\theta_1\}}$$

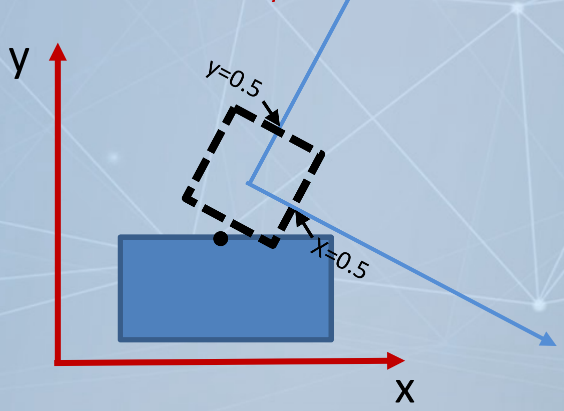


```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



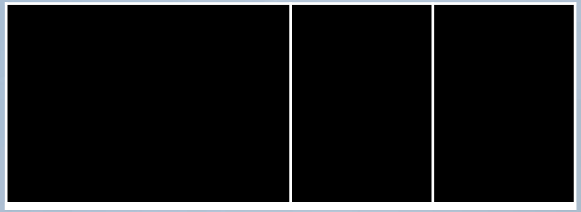
```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix().
The parameters in these function in the code may not match to the illustration.

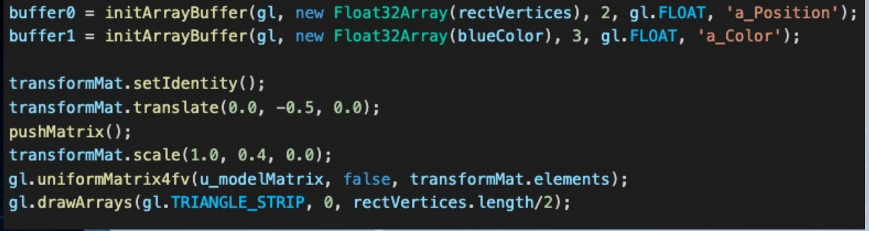


Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1)
Translate(tx_3, ty_3) -> $M_{\{tx_3,ty_3\}}$

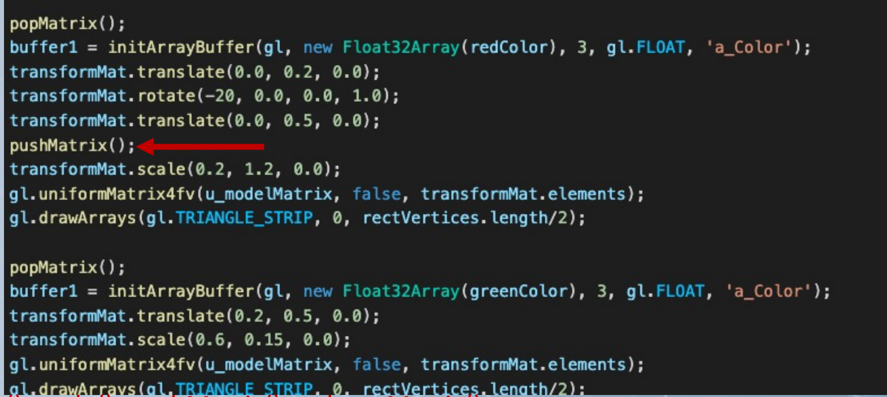
Stack



$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}} * M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}$$



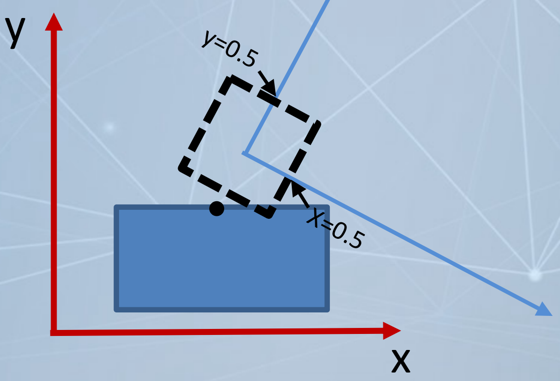
```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



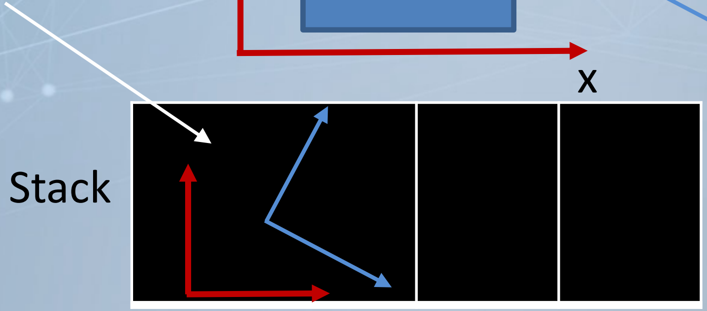
```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix().
The parameters in these function in the code may not match to the illustration.

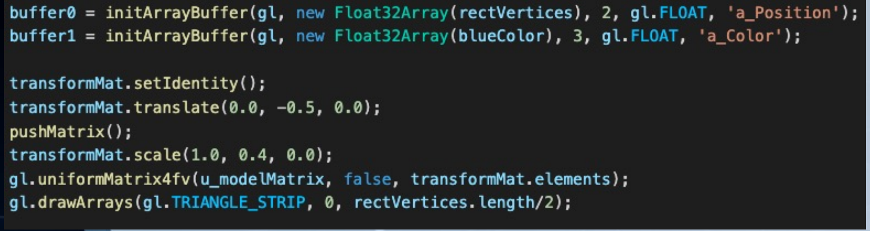
Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1)
Translate(tx_3, ty_3)
pushMatrix()



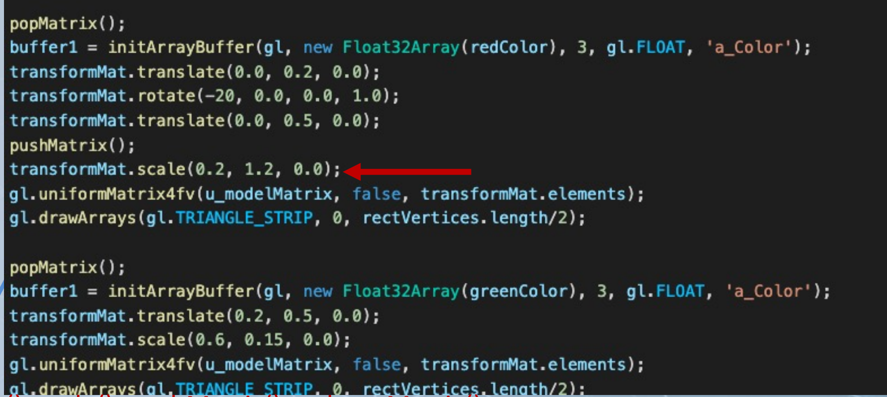
$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}} * M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}$



$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}} * M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}$

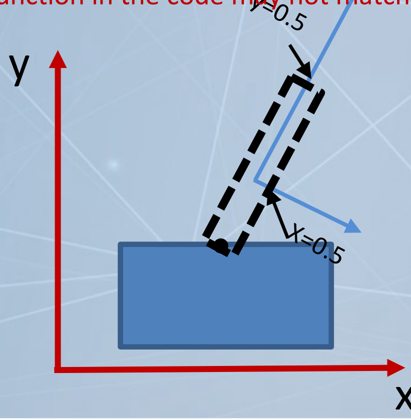


```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

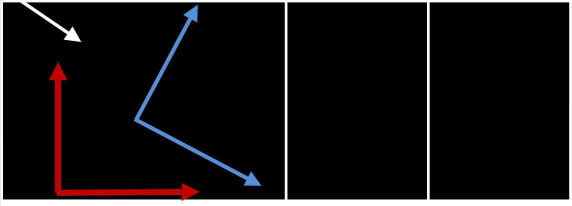
I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix().
The parameters in these function in the code may not match to the illustration.



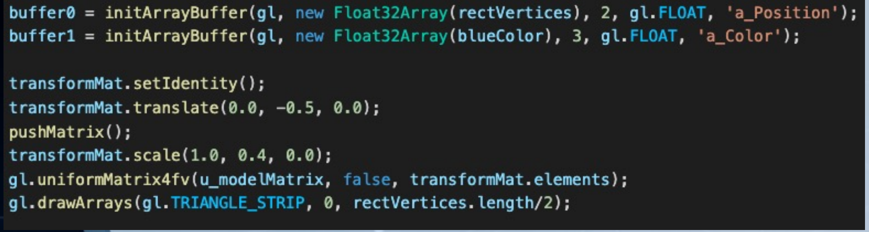
Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1)
Translate(tx_3, ty_3)
pushMatrix()
Scale(sx_2, sy_2) -> $M_{\{sx_2,sy_2\}}$

$I * \boldsymbol{M}_{\{tx_1,ty_1\}} * \boldsymbol{M}_{\{tx_2,ty_2\}} * \boldsymbol{M}_{\{\theta_1\}} * \boldsymbol{M}_{\{tx_3,ty_3\}}$

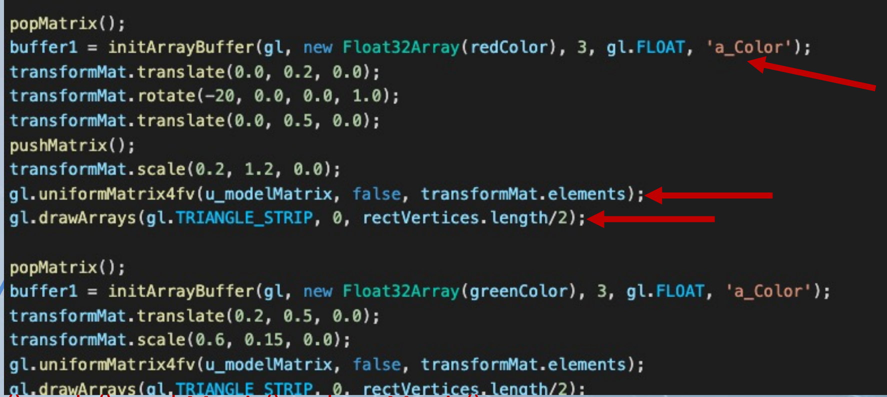
Stack



$I * \boldsymbol{M}_{\{tx_1,ty_1\}} * \boldsymbol{M}_{\{tx_2,ty_2\}} * \boldsymbol{M}_{\{\theta_1\}} * \boldsymbol{M}_{\{tx_3,ty_3\}} * \boldsymbol{M}_{\{sx_2,sy_2\}}$

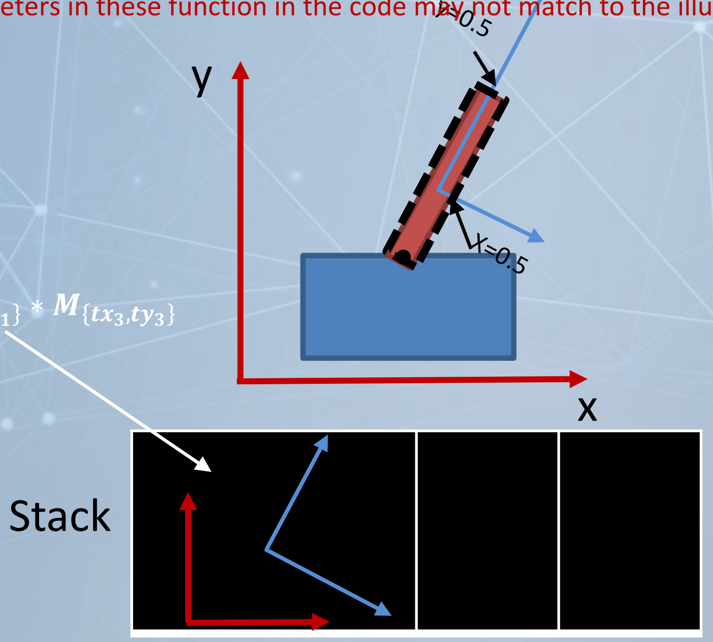


```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix().
The parameters in these function in the code may not match to the illustration.

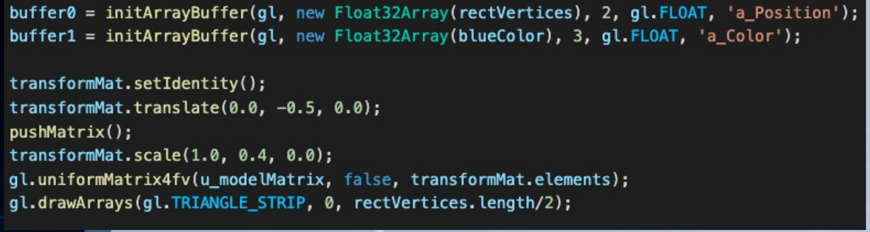


$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}} * M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}$

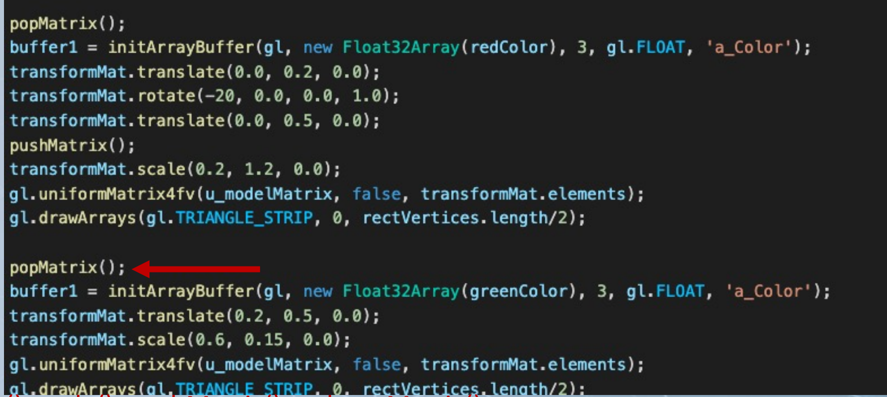
Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1)
Translate(tx_3, ty_3)
pushMatrix()
Scale(sx_2, sy_2)
Draw width =1 height = 1 red block

Use this matrix to transform a width=1 and height = 1 red block and draw

$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}} * M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}} * M_{\{sx_2,sy_2\}}$

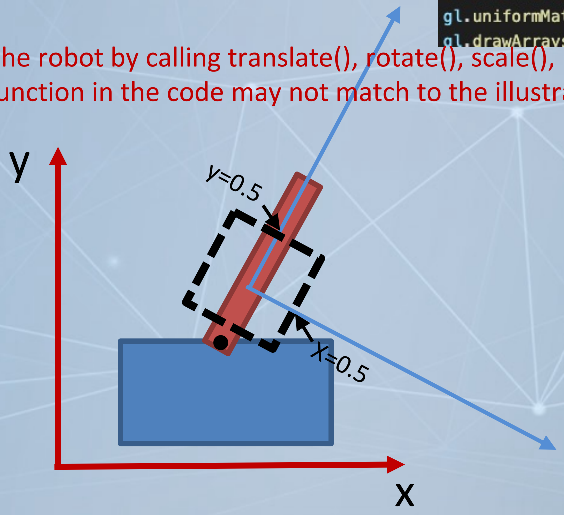


```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

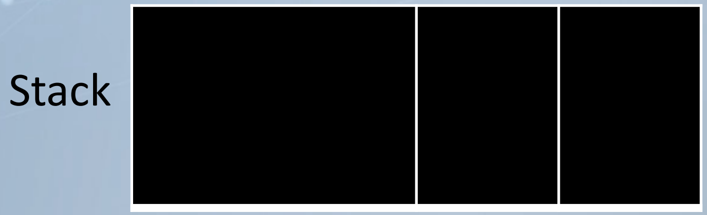


```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

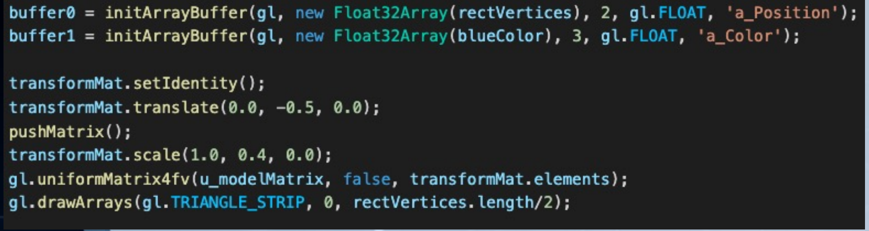
I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix(). The parameters in these function in the code may not match to the illustration.



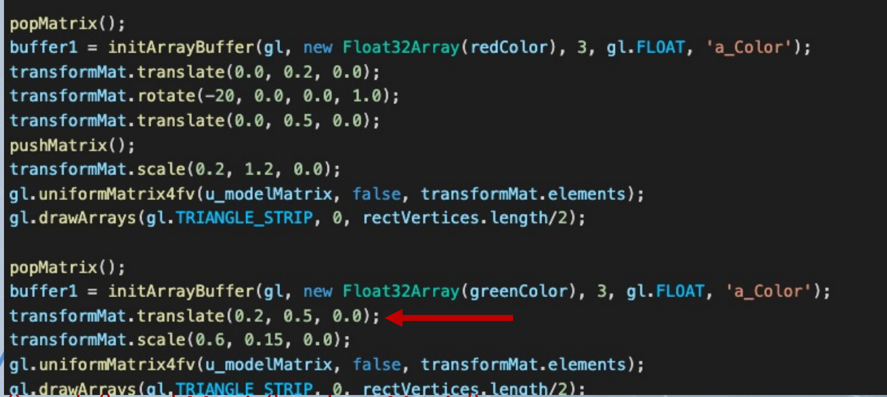
Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1)
Translate(tx_3, ty_3)
pushMatrix()
Scale(sx_2, sy_2)
Draw width =1 height = 1 red block
popMatrix()



$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}} * M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}$$

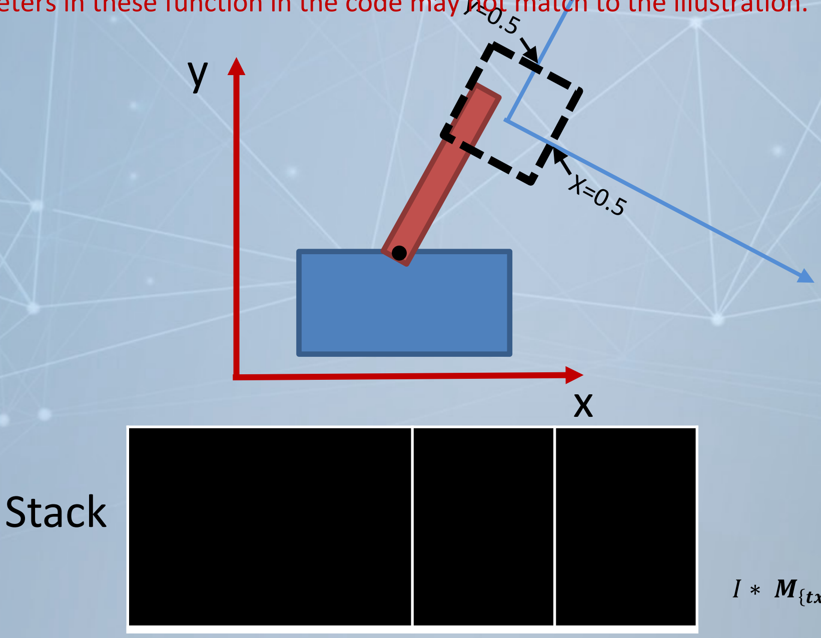


```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



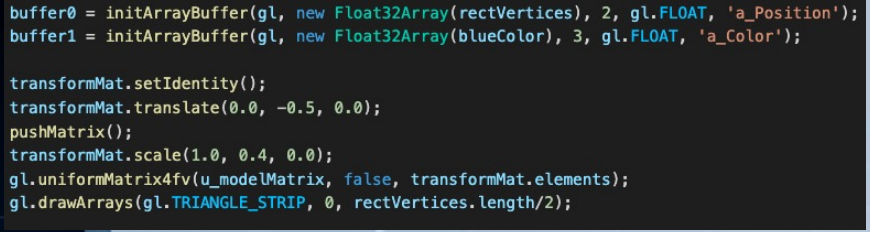
```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix().
The parameters in these function in the code may not match to the illustration.

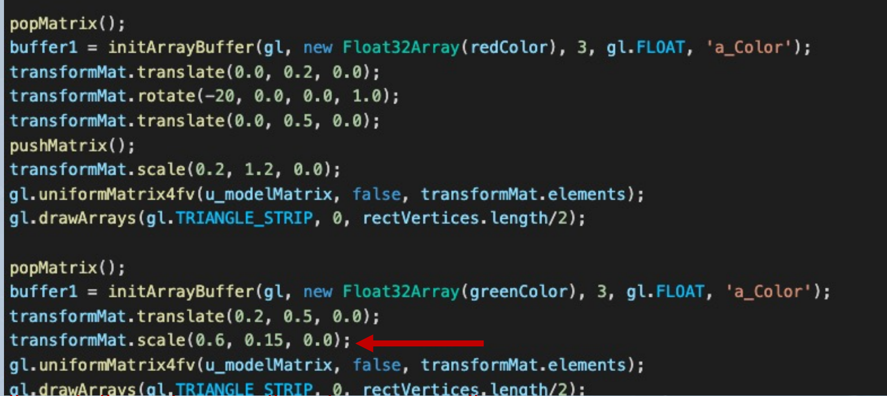


Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1)
Translate(tx_3, ty_3)
pushMatrix()
Scale(sx_2, sy_2)
Draw width =1 height = 1 red block
popMatrix()
Translate(tx_4, ty_4) -> $M_{\{tx_4,ty_4\}}$

$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}}^{*} M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}^{*} M_{\{tx_4,ty_4\}}$$

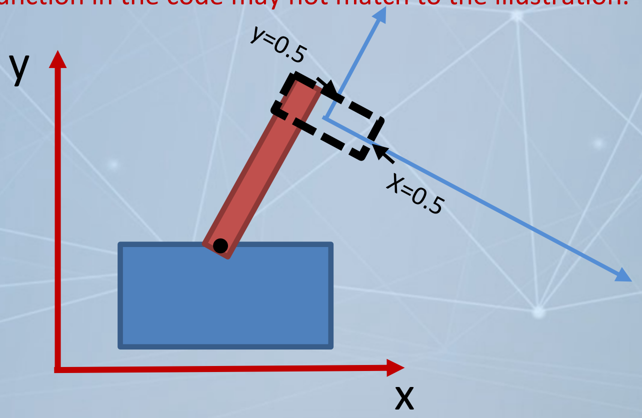


```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```



```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

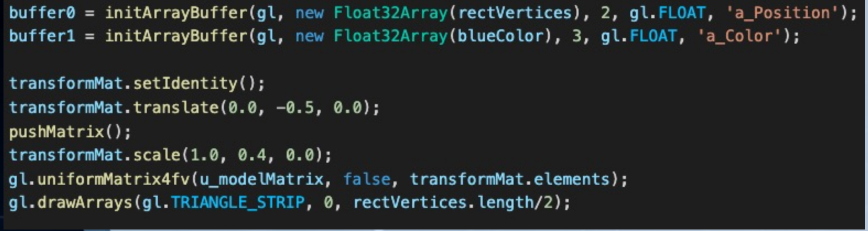
I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix().
The parameters in these function in the code may not match to the illustration.



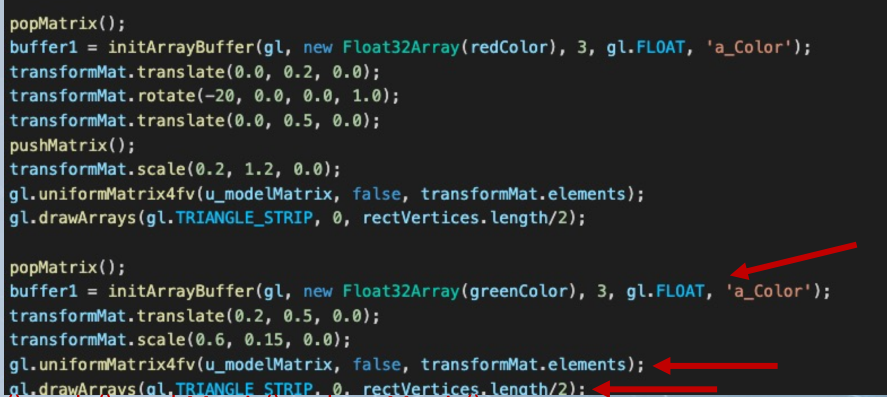
Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1)
Translate(tx_3, ty_3)
pushMatrix()
Scale(sx_2, sy_2)
Draw width =1 height = 1 red block
popMatrix()
Translate(tx_4, ty_4)
Scale(sx_3, sy_3) -> $M_{\{sx_3,sy_3\}}$

$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}}{}^* M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}}{}^* M_{\{tx_4,ty_4\}}{}^* M_{\{sx_3,sy_3\}}$$

y

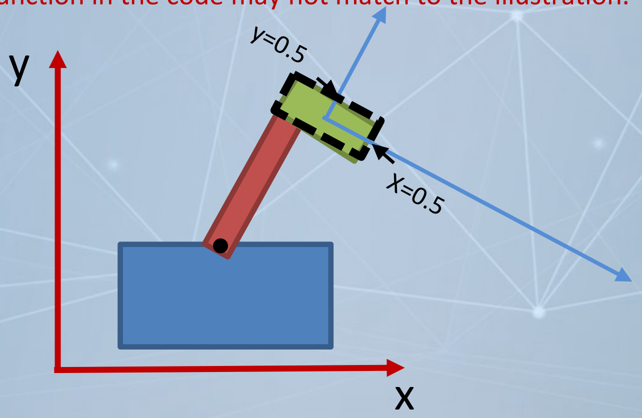


```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

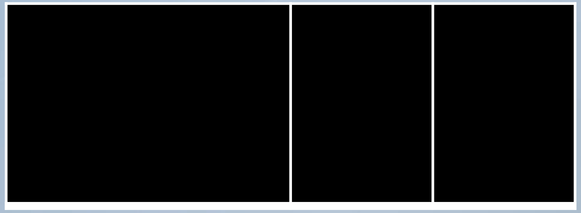


```
popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

I just show how we draw the robot by calling translate(), rotate(), scale(), pushMatrix() and popMatrix().
The parameters in these function in the code may not match to the illustration.



Stack



Translate(tx_1, ty_1)
pushMatrix()
Scale(sx1, sy1)
Draw a width =1 height = 1 blue block
popMatrix()
Translate(tx_2, ty_2)
Rotate(θ_1)
Translate(tx_3, ty_3)
pushMatrix()
Scale(sx_2, sy_2)
Draw width =1 height = 1 red block
popMatrix()
Translate(tx_4, ty_4)
Scale(sx_3, sy_3)
Draw width =1 height = 1 green block

Use this matrix to transform a width=1 and height = 1 green block and draw

$$I * M_{\{tx_1,ty_1\}} * M_{\{tx_2,ty_2\}} * M_{\{\theta_1\}} * M_{\{tx_3,ty_3\}} * M_{\{tx_4,ty_4\}} * M_{\{sx_3,sy_3\}}$$

Let’s try (5mins)

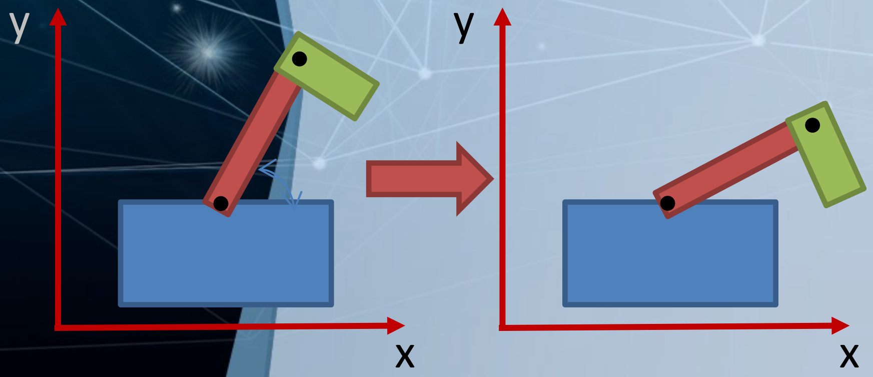
- Check the code and the slides above, make sure you understand all the details

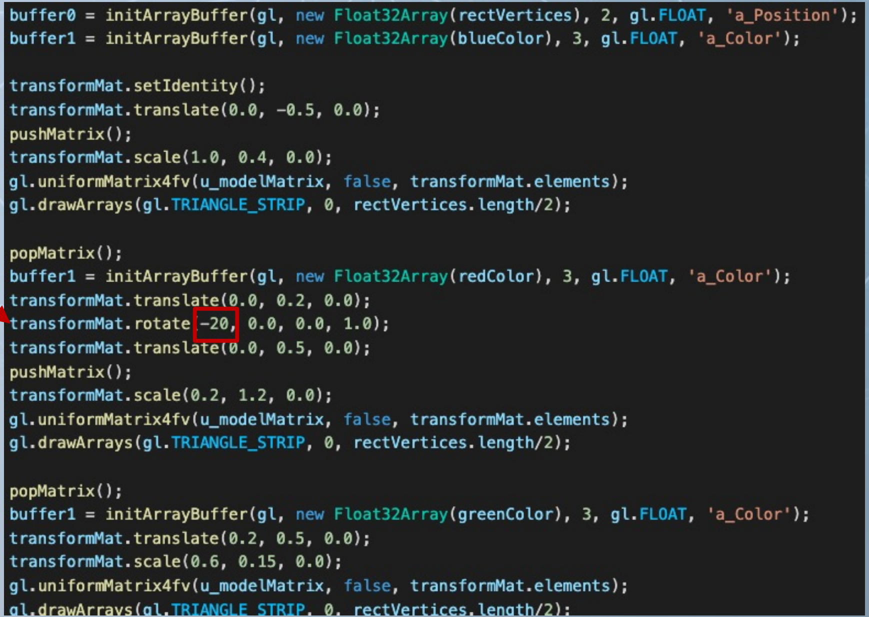
- You can modify some numbers to verify your understanding

True Power of Hierarchical Transformation

- It becomes very easy to transform a part of an object
  - Ex: rotate a part of an object

Just modify the angle here



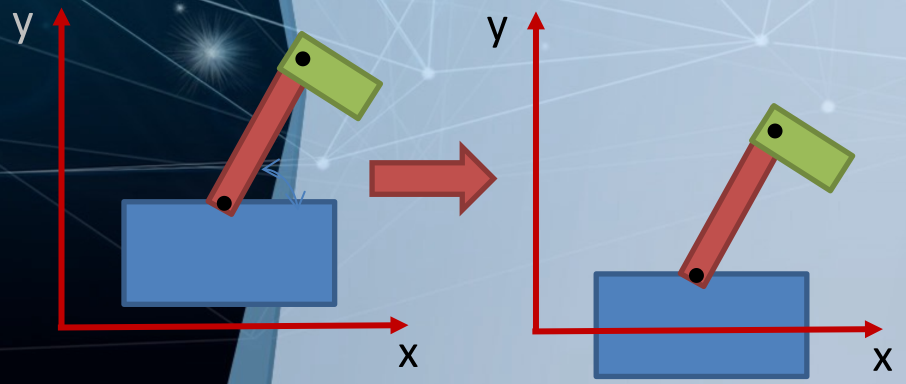


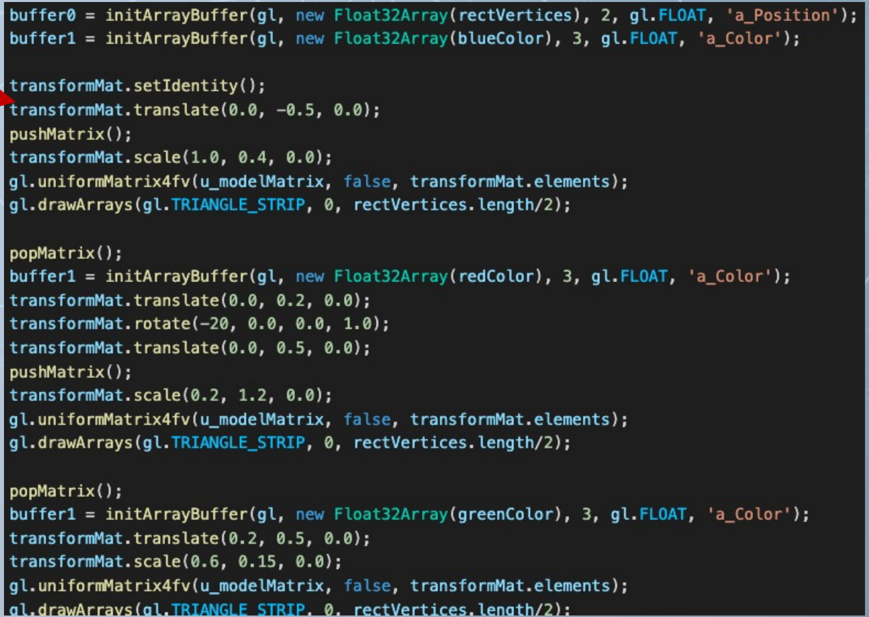
```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

True Power of Hierarchical Transformation

- It becomes very easy to transform a part of an object
  - Ex: translate(move) whole object

Insert a translate() here, and modify its “tx” and “ty” by what you want it to moves





```
buffer0 = initArrayBuffer(gl, new Float32Array(rectVertices), 2, gl.FLOAT, 'a_Position');
buffer1 = initArrayBuffer(gl, new Float32Array(blueColor), 3, gl.FLOAT, 'a_Color');

transformMat.setIdentity();
transformMat.translate(0.0, -0.5, 0.0);
pushMatrix();
transformMat.scale(1.0, 0.4, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(redColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.0, 0.2, 0.0);
transformMat.rotate(-20, 0.0, 0.0, 1.0);
transformMat.translate(0.0, 0.5, 0.0);
pushMatrix();
transformMat.scale(0.2, 1.2, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);

popMatrix();
buffer1 = initArrayBuffer(gl, new Float32Array(greenColor), 3, gl.FLOAT, 'a_Color');
transformMat.translate(0.2, 0.5, 0.0);
transformMat.scale(0.6, 0.15, 0.0);
gl.uniformMatrix4fv(u_modelMatrix, false, transformMat.elements);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, rectVertices.length/2);
```

Let’s try (5mins)

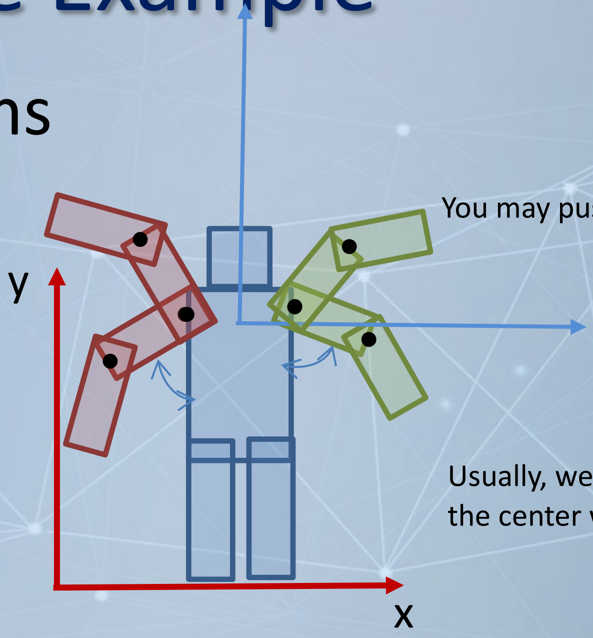
- Still use Ex04-1

- Try what we learn in the two previous slides
  - Modify angle in rotate() to rotate the arm
  - Add a translate() to move the whole robot

- Try any thing you want to try

One More Example

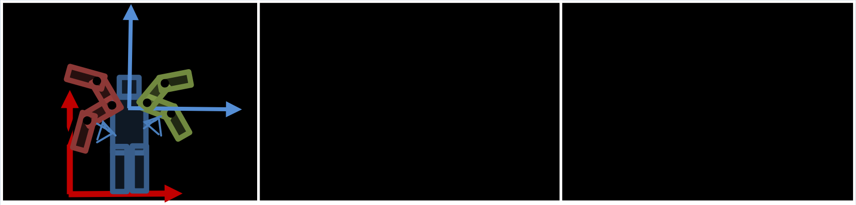
- Robot arms



You may push the current active matrix into stack

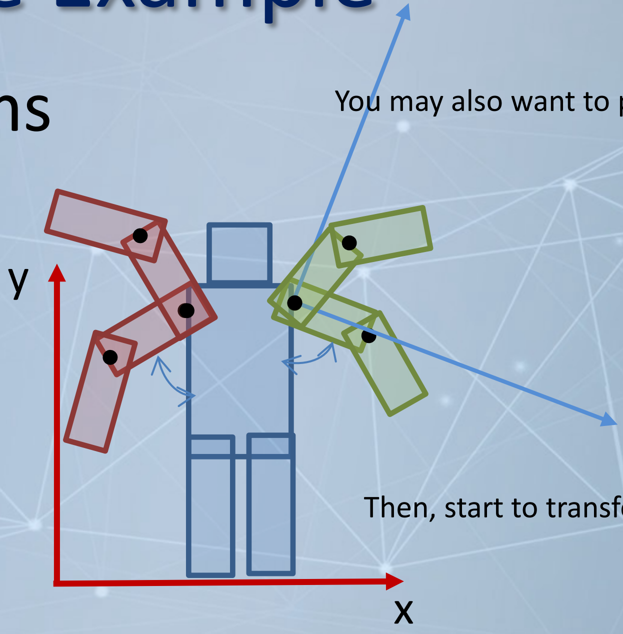
Usually, we firstly move the object coordinate to the center where we want to put the object

Stack



One More Example

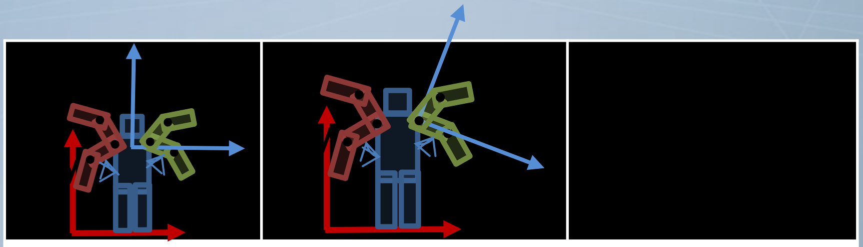
- Robot arms



You may also want to put this active matrix into the stack, too

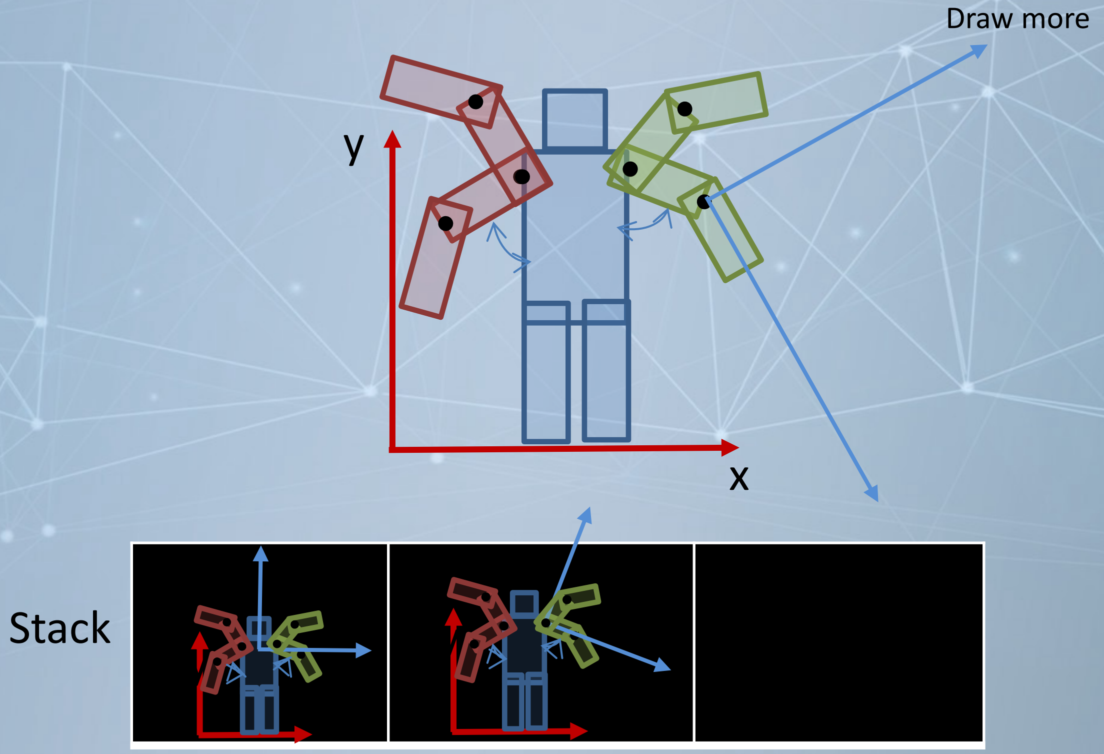
Then, start to transform the object coordinate and draw

Stack



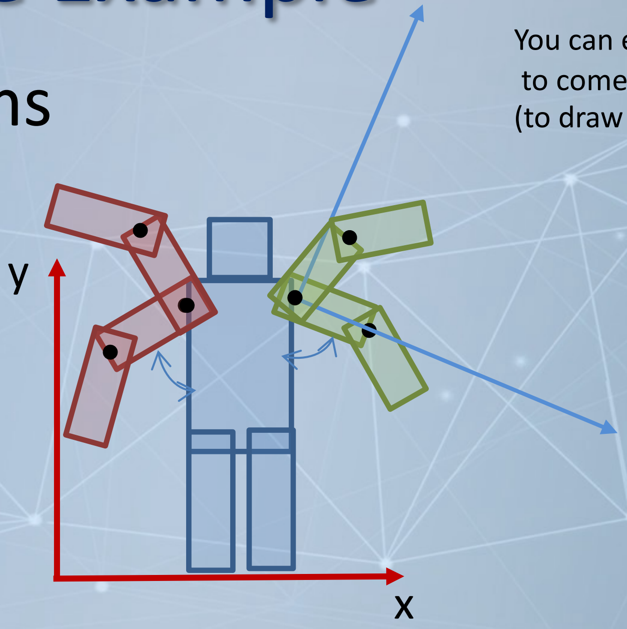
One More Example

- Robot arms



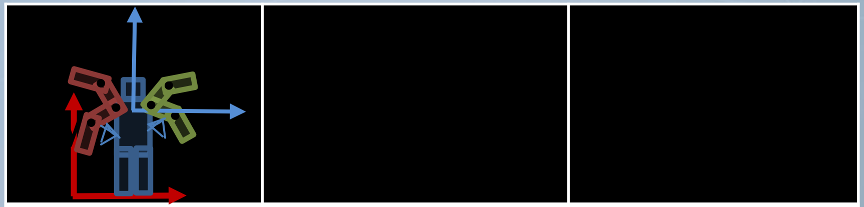
One More Example

- Robot arms



You can easily pop the matrix from the stack
to come back a state you want
(to draw another green arm)

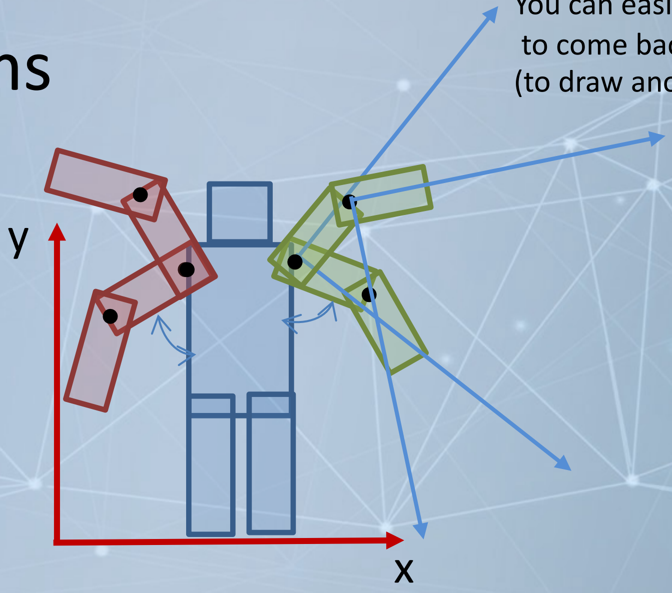
Stack



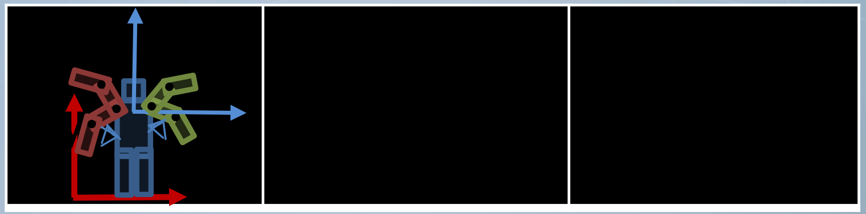
One More Example

- Robot arms

You can easily pop the matrix from the stack
to come back a state you want
(to draw another green arm)

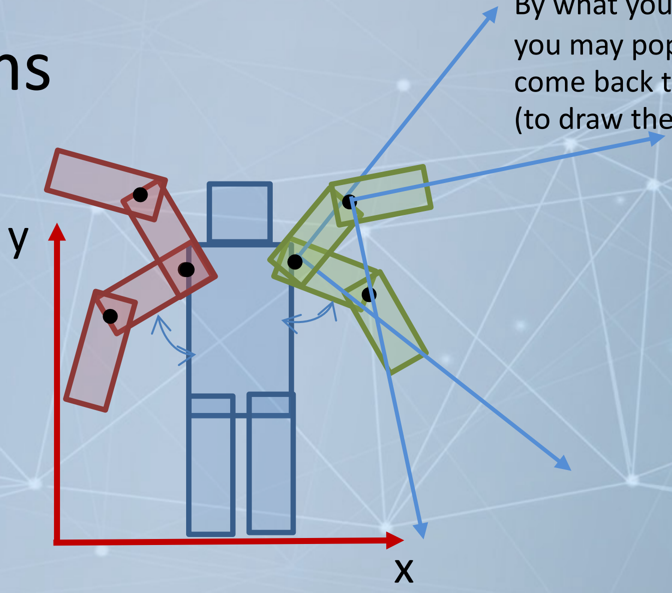


Stack



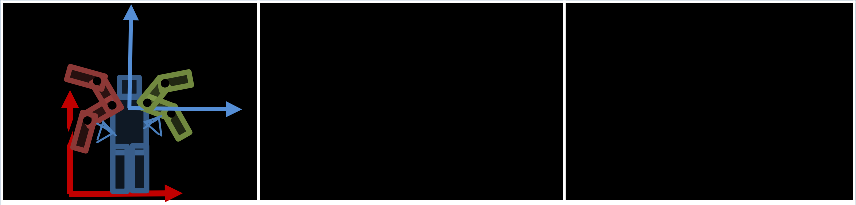
One More Example

- Robot arms



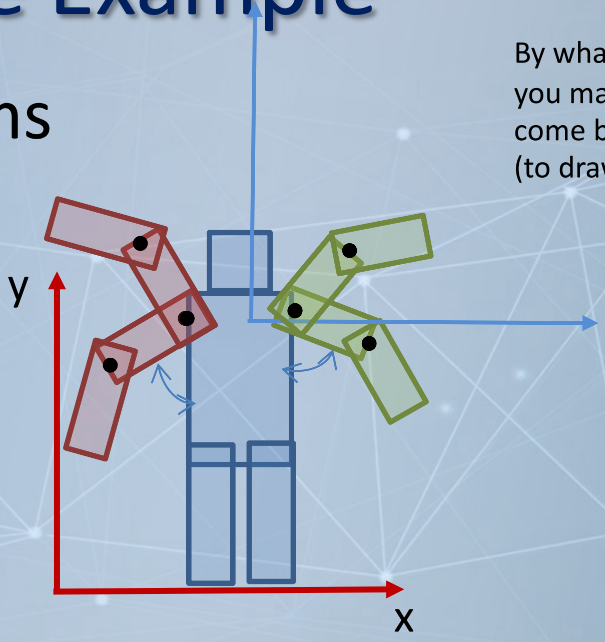
By what you push into the matrix,
you may pop out matrix from the stack to
come back to the center of the object
(to draw the red arms)

Stack



One More Example

- Robot arms

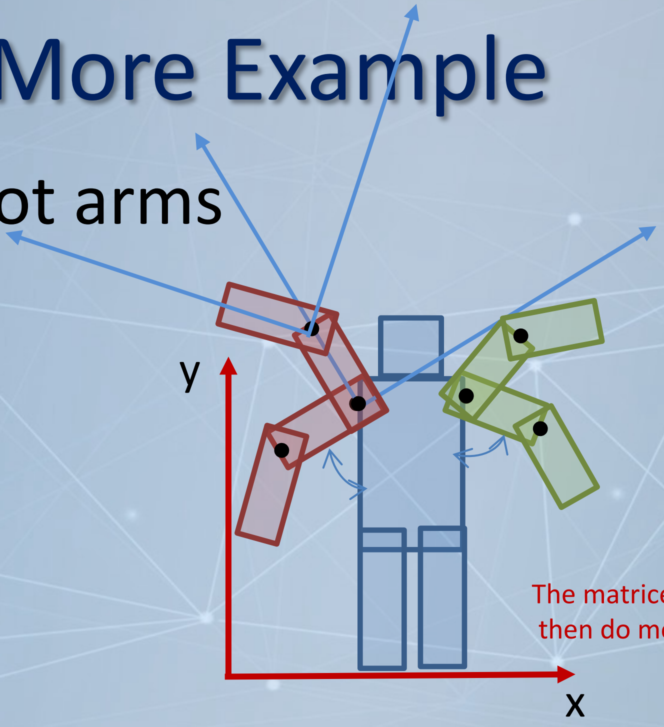


By what you push into the matrix,
you may pop out matrix from the stack to
come back to the center of the object
(to draw the red arms)

Stack

One More Example

- Robot arms



By what you push into the matrix,
you may pop out matrix from the stack to
come back to the center of the object
(to draw the red arms)

The matrices in the stack help you go back some key points/joints,
then do more transformations to draw other part of your object

Stack