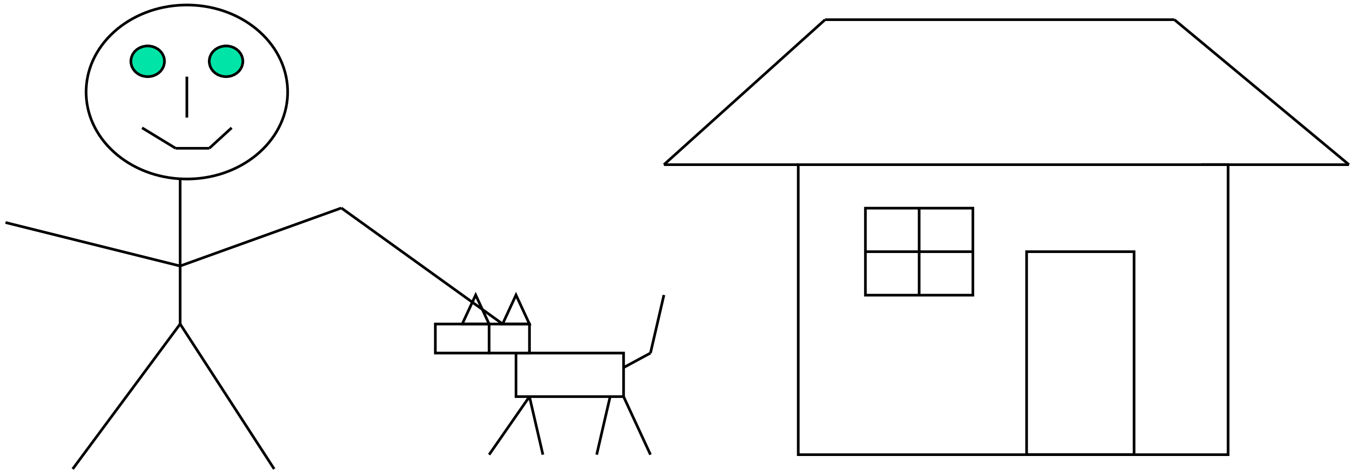
Drawing and Coordinate Systems



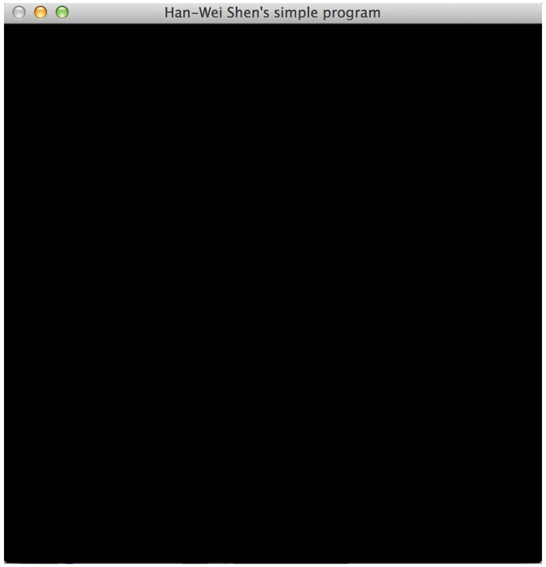
What are the visible coordinates range?

- For 2D drawing, the visible range of the display window is from [-1,-1] to [1,1] (for 3D, the z is also from -1 to 1, but we will talk about it later)
- In other words, you need to transform your points to this range so that they will be visible
- This is called "Normalized Device Coordinate (NDC) system

But how to map the NDC to the display window?

- A pixel in a window is referenced as two integers (i,j)
- This is called the screen coordinate (SC) system

[500,500]



[0,0]

** Remember GLUT uses the upper left corner as [0,0]

From NDC to SC

- Just do a linear mapping from

  [-1,-1] x [1,1] to [0,0] x [I_{max}, J_{max}]
- That is, assume (x,y) is in NDC, (i,j) is in SC, then

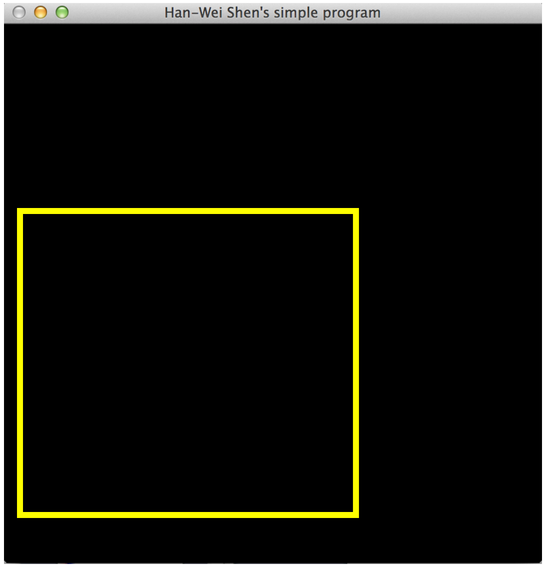
  $$i = (x - (-1))/2.0 \ * \ I_{max}$$

  $$j = (y - (-1))/2.0 \ * \ J_{max}$$

You can draw to any sub-area in the window

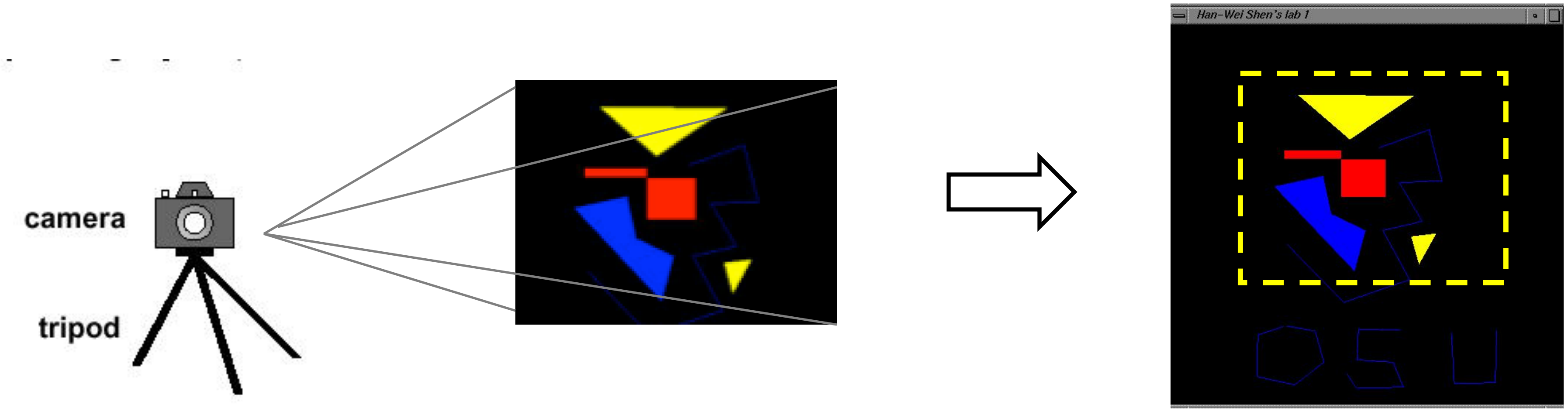
- If you do not want to use the entire window, you can define a sub-area called 'Viewport' as your drawing area
- Your drawing will only show up in the viewport
- Your points will be mapped from NDC to viewport

The yellow box is called viewport



Viewport Mapping

- Convert the vertex coordinates from the normalized device coordinates (NDC) to the screen space
- The NDC has the range of (-1,1) in both X and Y for everything that is visible



From NDC to Viewport

- Just do a linear mapping from

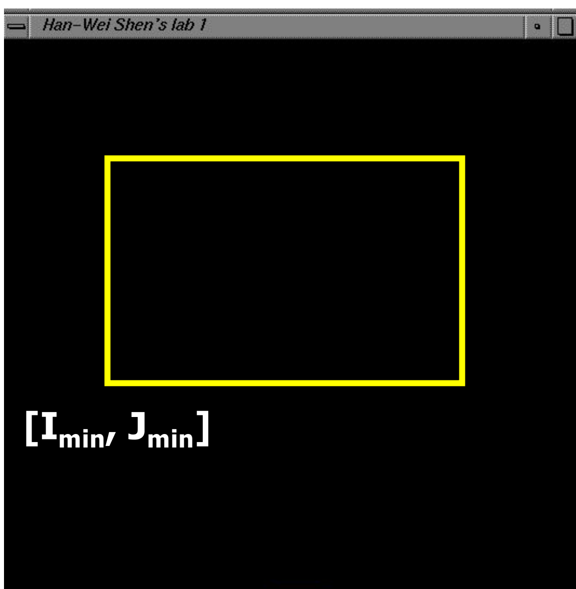
  $[-1,-1] \times [1,1]$ to $[I_{min}, J_{min}] \times [I_{max}, J_{max}]$
- Assume (x,y) is in NDC, (i,j) is in SC, then

  $$i = (x - (-1))/2.0 \;\; * \;\; (I_{max}-I_{min}) + I_{min}$$

  $$j = (y - (-1))/2.0 \;\; * \;\; (J_{max}-J_{min}) + J_{min}$$

Viewport in WebGL

- Viewport: the rectangular region in the screen for displaying the graphical objects defined
- Viewport is defined using the screen
  m in pixels



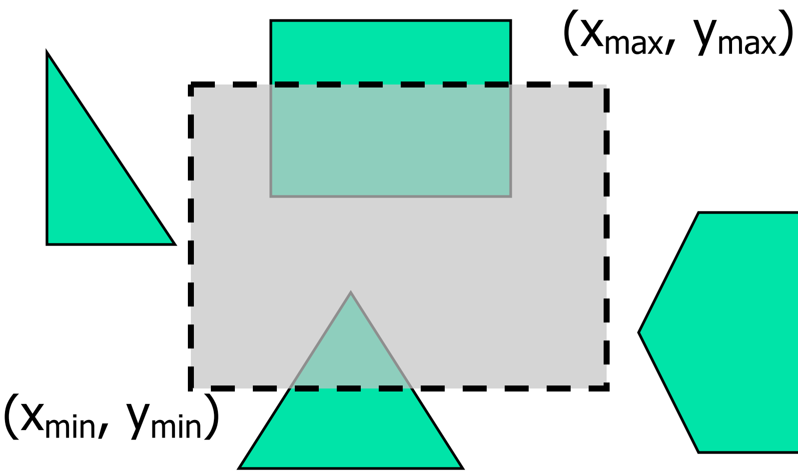
gl.Viewport(int I_{min}, int J_{min},
int width),
int height));

call this function before drawing (calling glBegin() and glEnd())

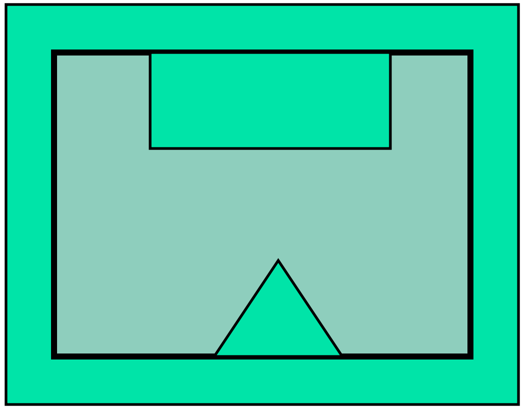
Choose the visible ranges from your data

- As we discussed, only points that are mapped to the [-1,-1] – [1,1] in NDC space are visible
- But what if you want to view different portions of your data?
  - Transform them into the [-1,-1]-[1,1] range by setting up the X_{min}, X_{max}, Y_{min}, Y_{max} range and perform a linear transformation
  - Let me call this 'Projection'

(A Very Simple) 2D Project



Projection



Viewport Mapping

$$X = 2 * (x - x_{min})/(x_{max} - x_{min}) - 1$$
$$Y = 2 * (y - y_{min})/(y_{max} - y_{min}) - 1$$

$$i = (x - (-1))/2.0 * (I_{max} - I_{min}) + I_{min}$$
$$j = (y - (-1))/2.0 * (J_{max} - J_{min}) + J_{min}$$

2D Projection in OpenGL

- Use Ortho() to get a projection matrix to perform the task mentioned in the previous slide
- Pass the matrix as a uniform to the vertex shader
- Multiple to the vertex position

2D Projection in OpenGL

- Use Ortho() to get a projection matrix to perform the task mentioned in the previous slide
- Pass the matrix as a uniform to the vertex shader
- Multiple to the vertex position
- Transform-ortho2D.html/js in github

2D Projection in OpenGL

- Use Ortho() to get a projection matrix to perform the task mentioned in the

```
mat4.identity(mvMatrix);
console.log('Z angle = '+ Z_angle);
mvMatrix = mat4.rotate(mvMatrix, degToRad(Z_angle), [0, 0, 1]);

mat4.identity(pMatrix);
mat4.ortho(0, 100, 0, 100, -1, 1, pMatrix); //orthographic projection, range: [-80, 100]x[-80, 100]
```

- Multiple to the vertex position
- Transform-ortho2D.html/js in github

2D Projection in OpenGL

- Use Ortho() to get a projection matrix to perform the task mentioned in the previous slide
- Pass the matrix as a uniform to the vertex shader

```
gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
```

- Transform-ortho2D.html/js in github

2D Projection in OpenGL

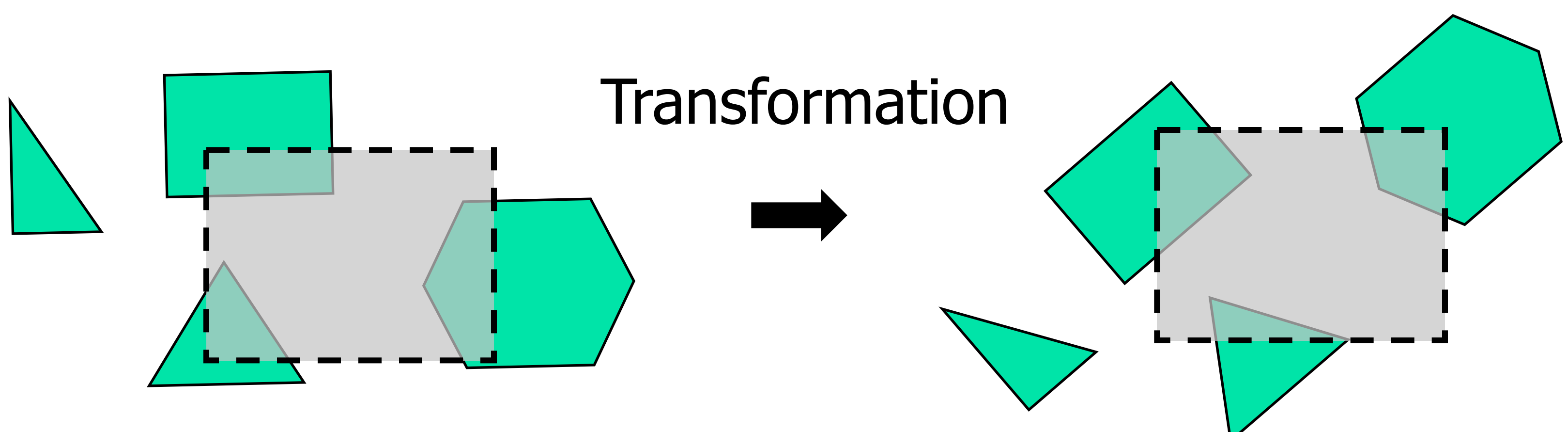
- Use Ortho() to get a projection matrix

```
uniform mat4 uMVMatrix;
uniform mat4 uPMatrix;

 void main(void) {

  gl_PointSize = 10.0;

   //gl_Position = uMVMatrix*vec4(aVertexPosition, 1.0);
   gl_Position = uPMatrix* uMVMatrix*vec4(aVertexPosition, 1.0);
```

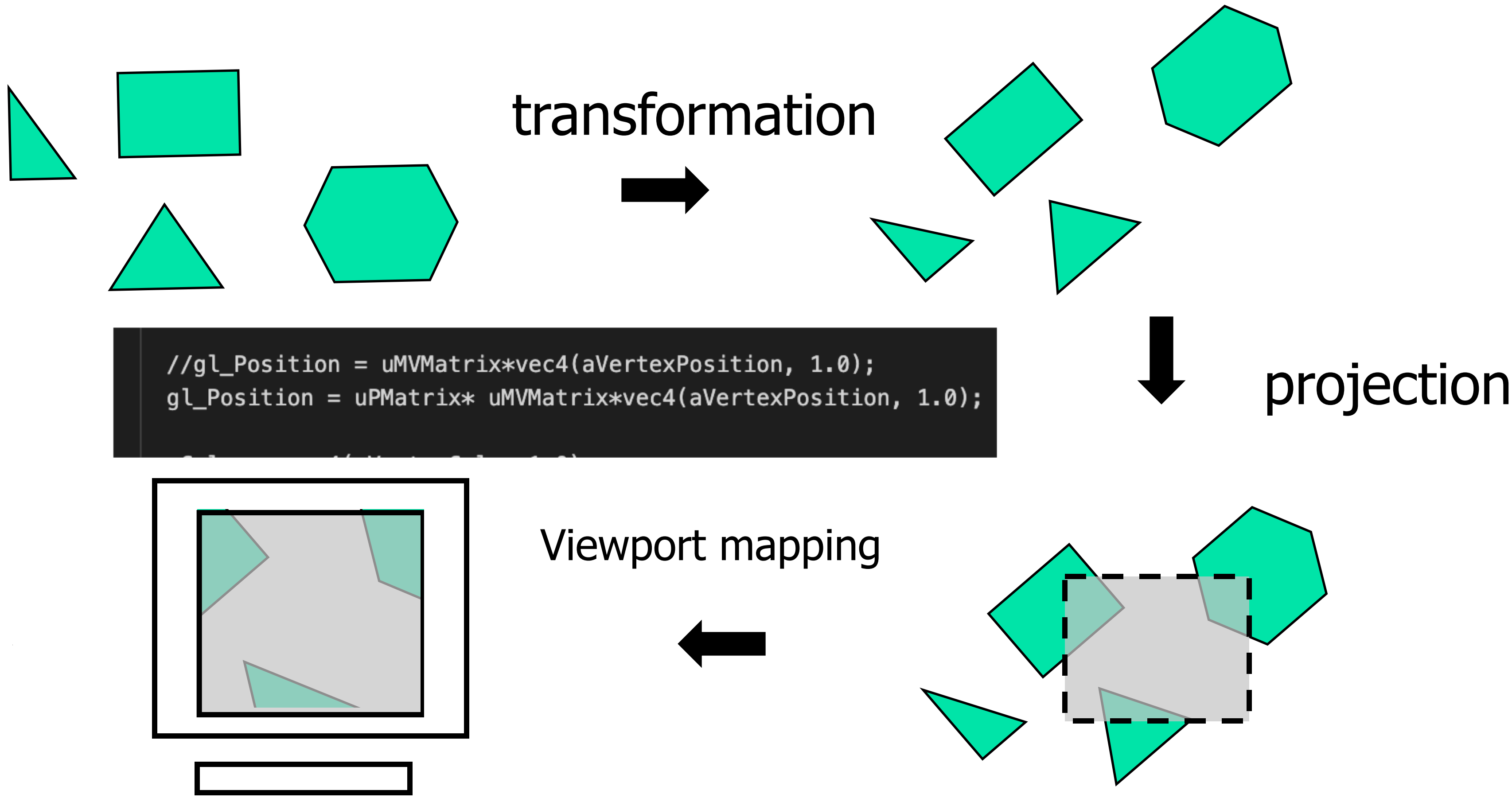
- Multiple to the vertex position
- Transform-ortho2D.html/js in github

Transformations

- Then of course, you can arbitrarily transform your points (rotation, scaling, translation) before performing such a projection



Transformation/Projection/Mapping Pipeline



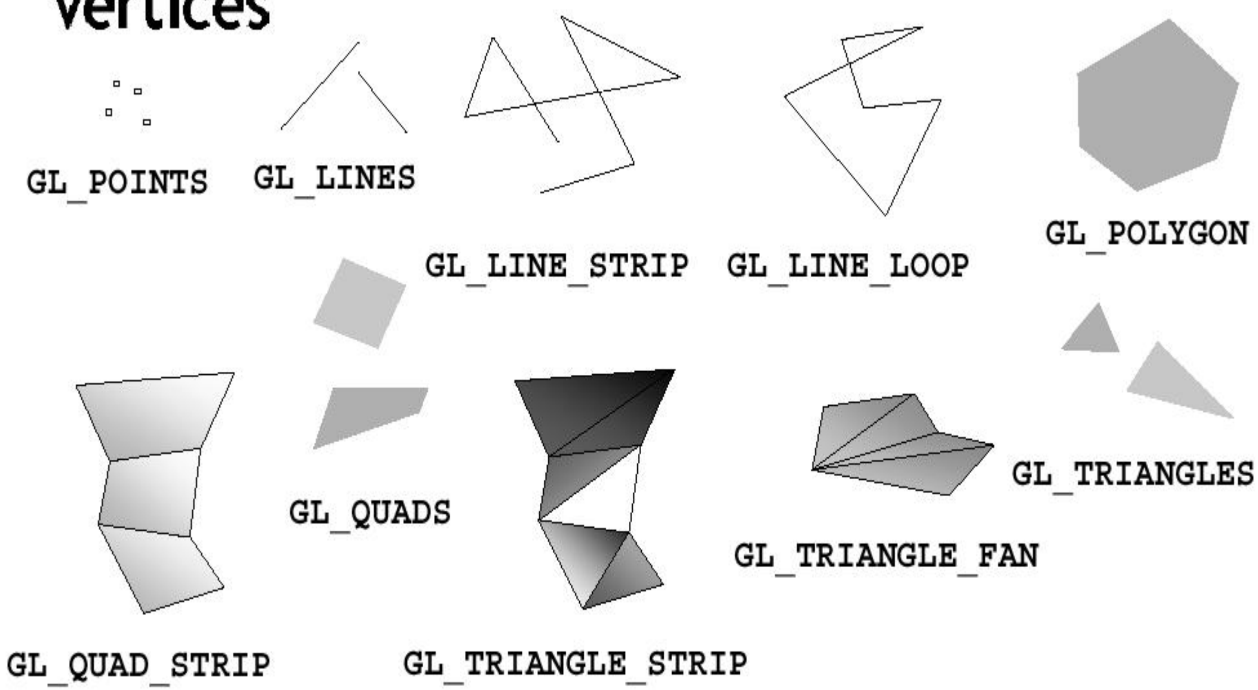
Transformation/Projection/Mapping Pipeline

- The pipeline is also called transformation pipeline
- It is done as part of geometry (vertex) processing
- Common transformations: rotation, translation, and scaling
- Projection: orthographic and perspective (3D only)
- Different stages in the pipeline correspond to different spaces

Geometry Transformation

Transformations are applied to all geometric primitives , and

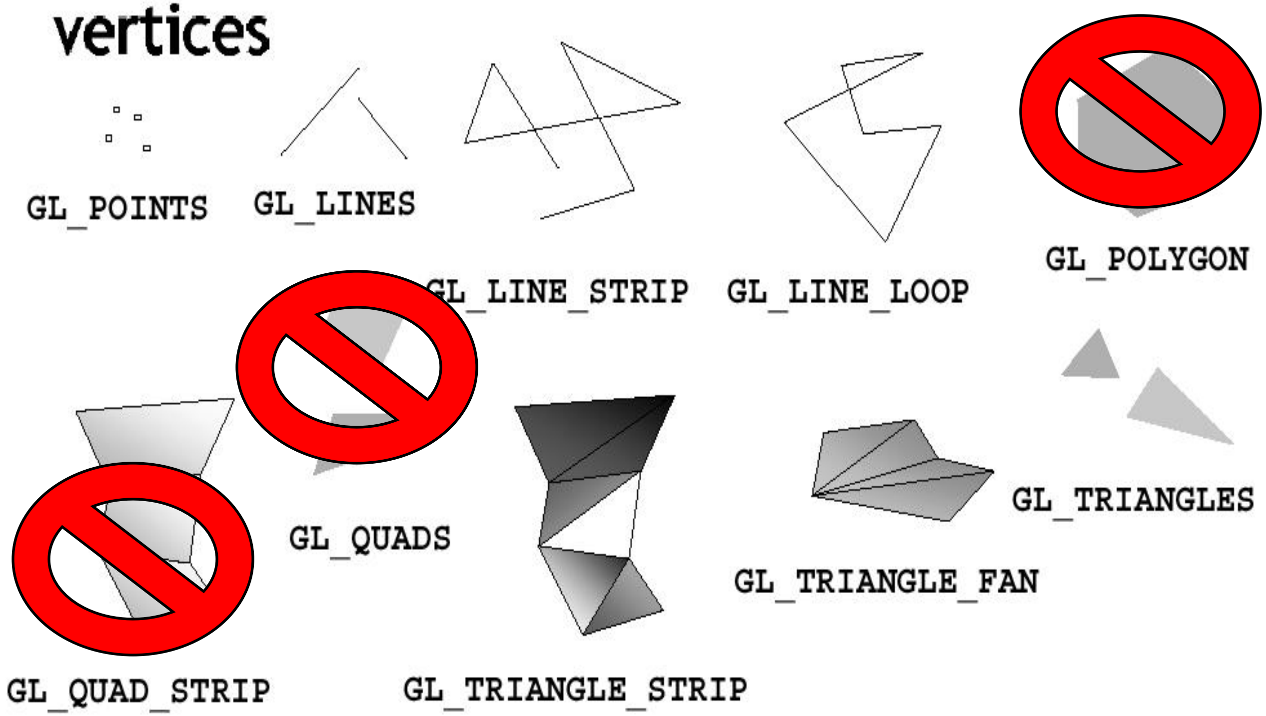
All geometric primitives are specified by vertices



WebGL Geometric Primitives

Transformations are applied to all geometric primitives , and

All geometric primitives are specified by vertices



Transformation & Spaces