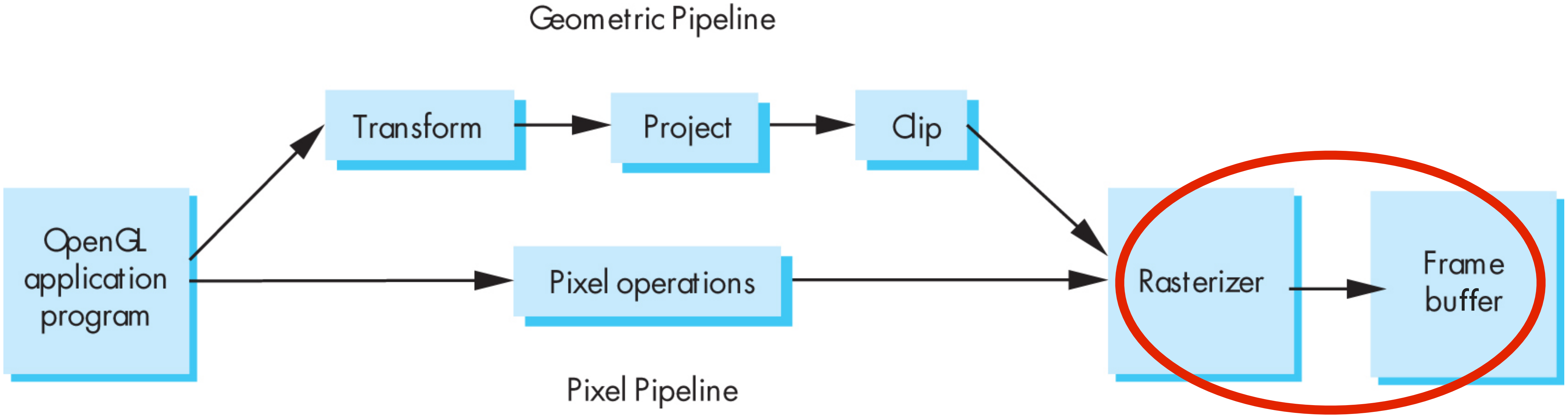
CS130 : Computer Graphics

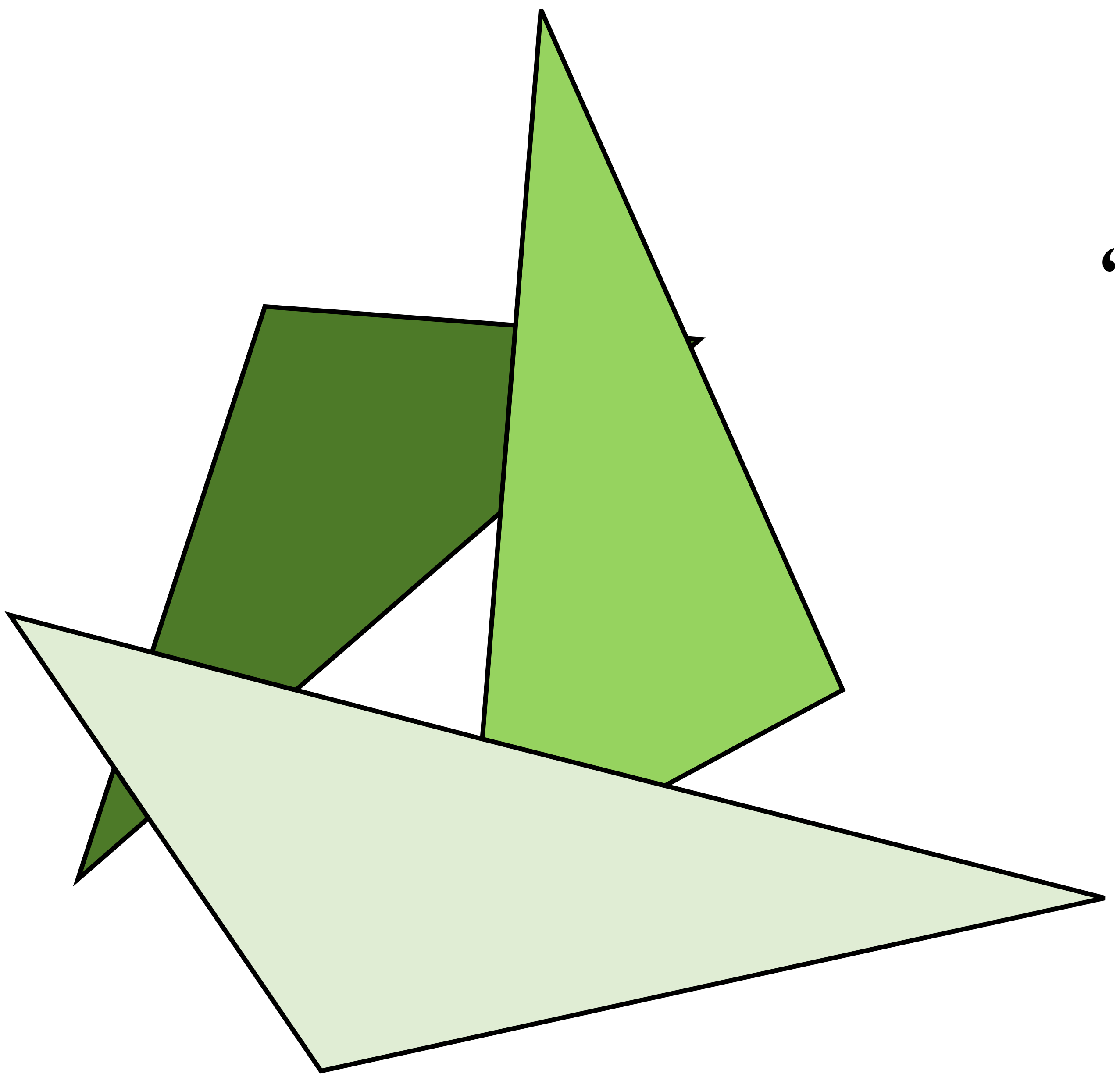
Lecture 5: Viewing Transformations

Tamar Shinar
Computer Science & Engineering
UC Riverside

Hidden Surface Removal

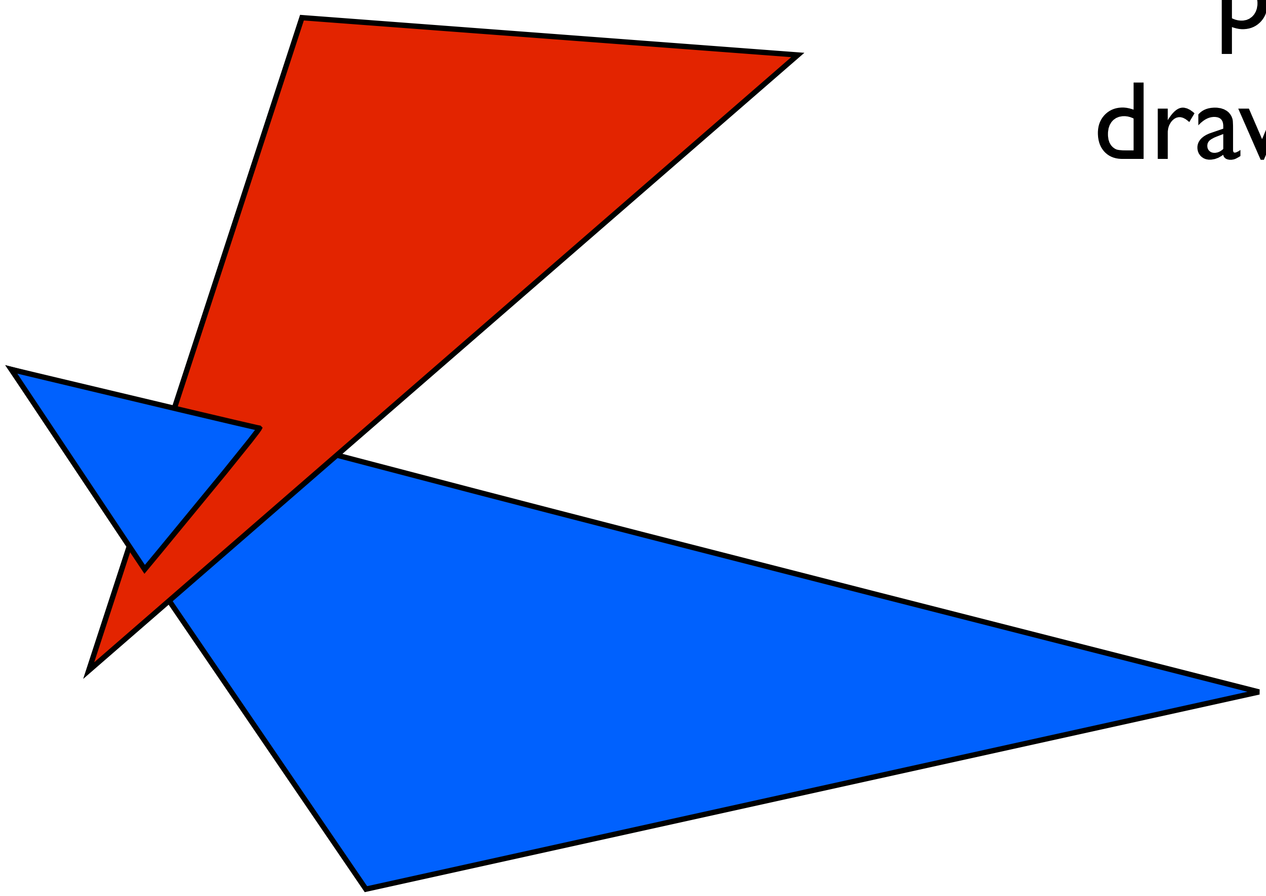


Occlusion



"painter's algorithm"
draw primitives in
back-to-front order

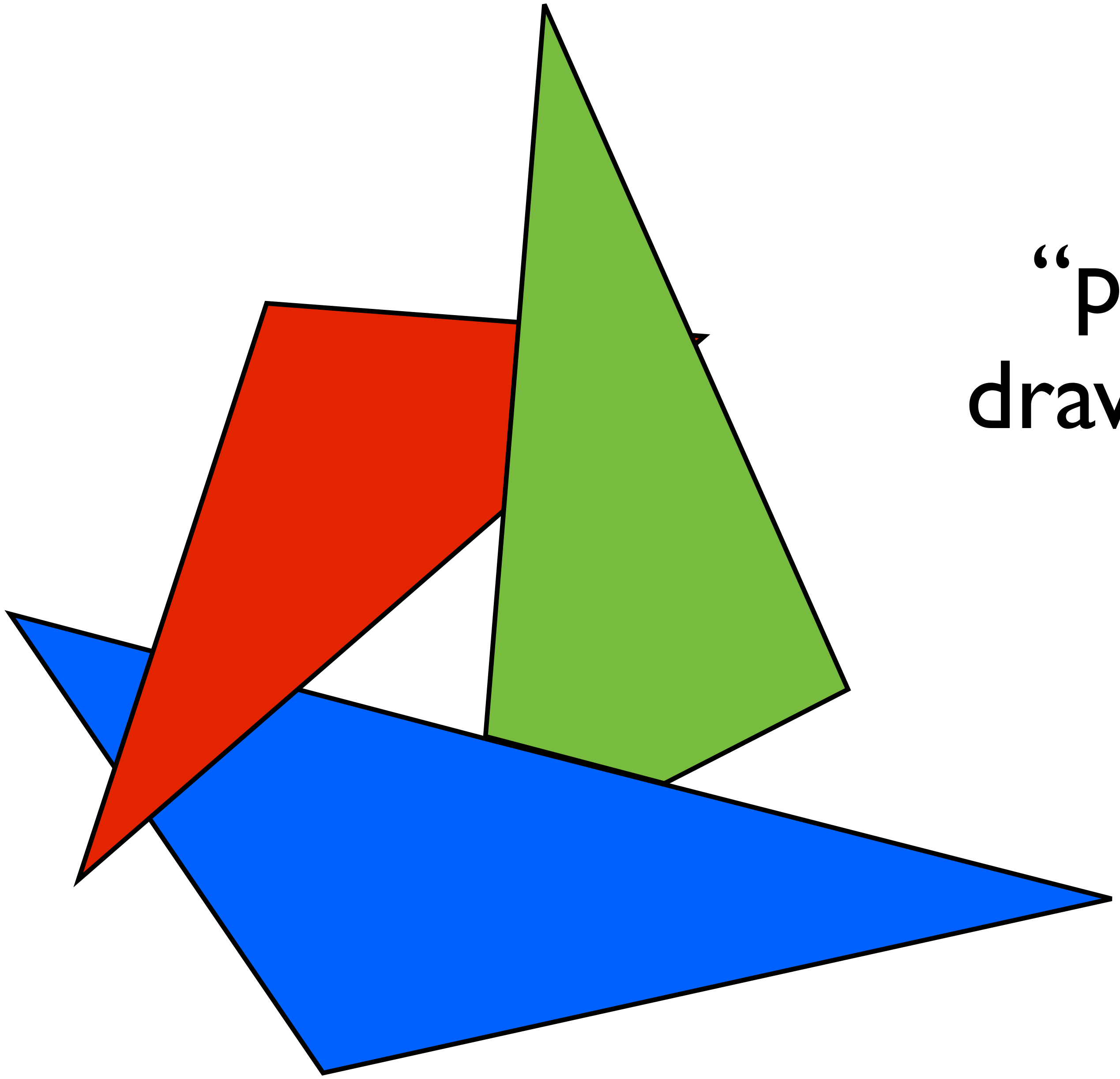
Occlusion



"painter's algorithm"
draw primitives in back-to-front order

problem:
triangle
intersection

Occlusion



"painter's algorithm" draw primitives in back-to-front order

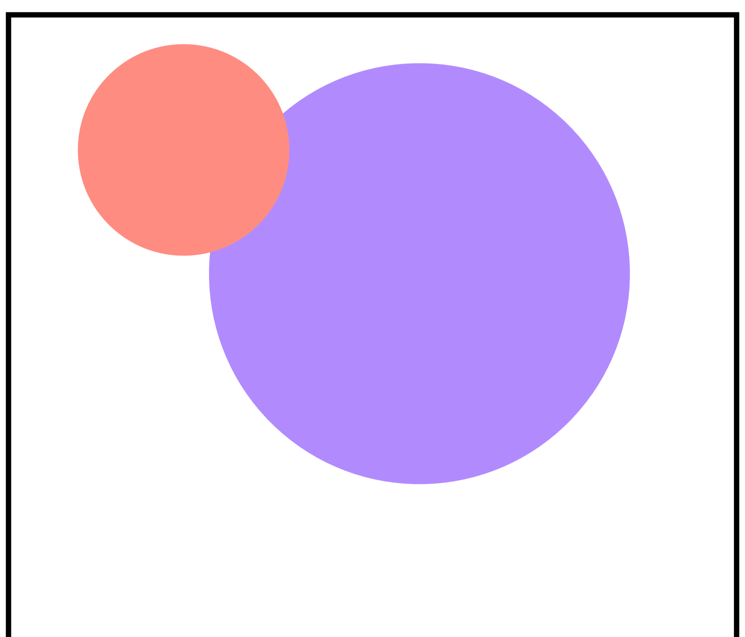
problem: occlusion cycle

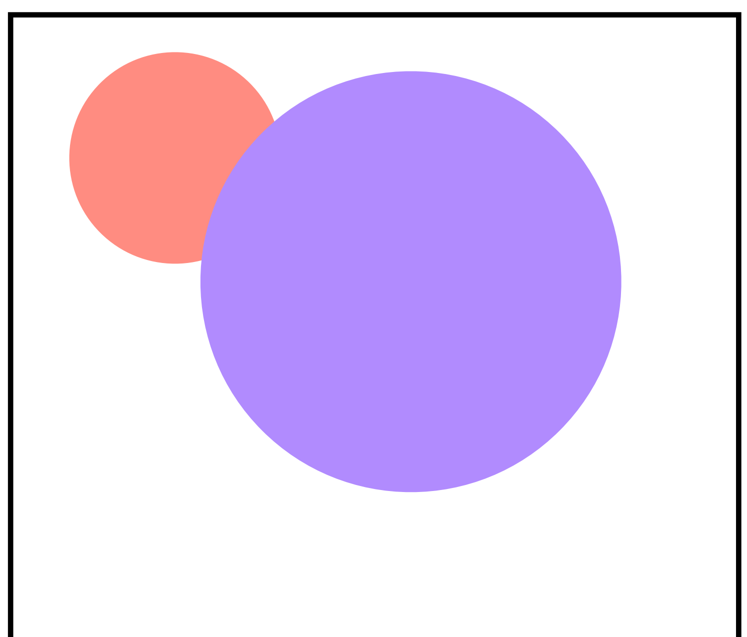
Use a *z-buffer* for hidden surface removal

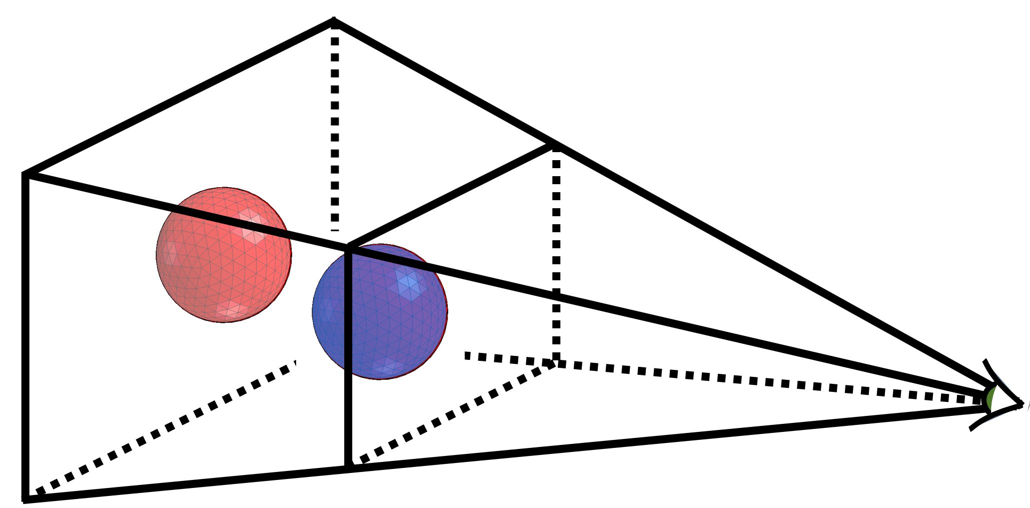
at each pixel, record distance to the closest object that has been drawn in a *depth* buffer

Use a *z-buffer* for hidden surface removal

at each pixel, record distance to the closest object that has been drawn in a *depth* buffer


without z-buffer


with z-buffer



Use a *z-buffer* for hidden surface removal

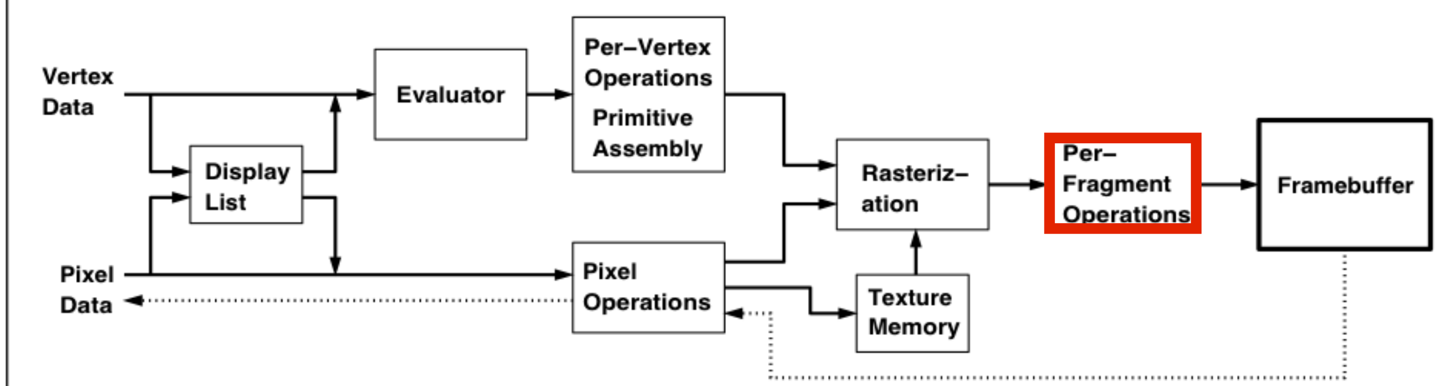
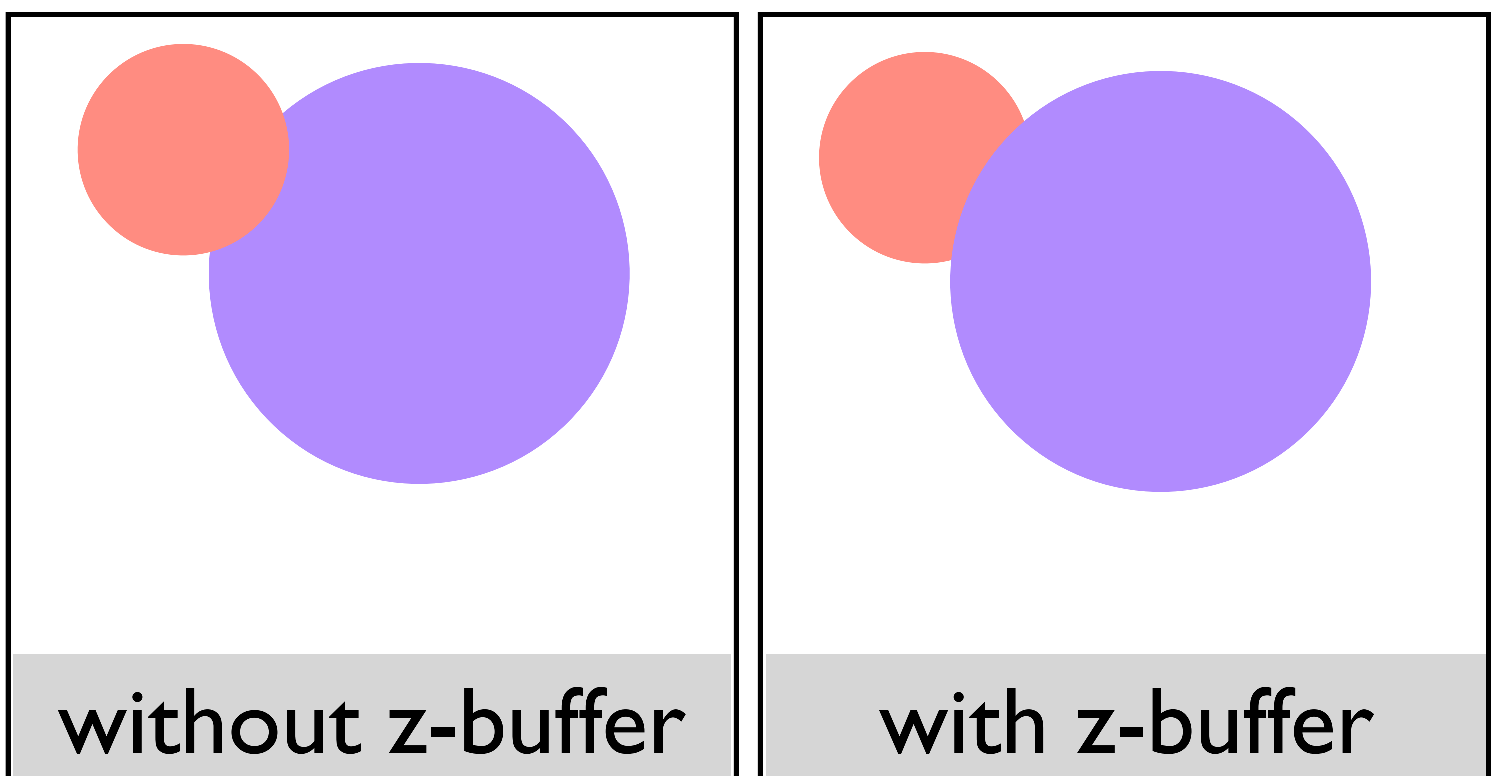
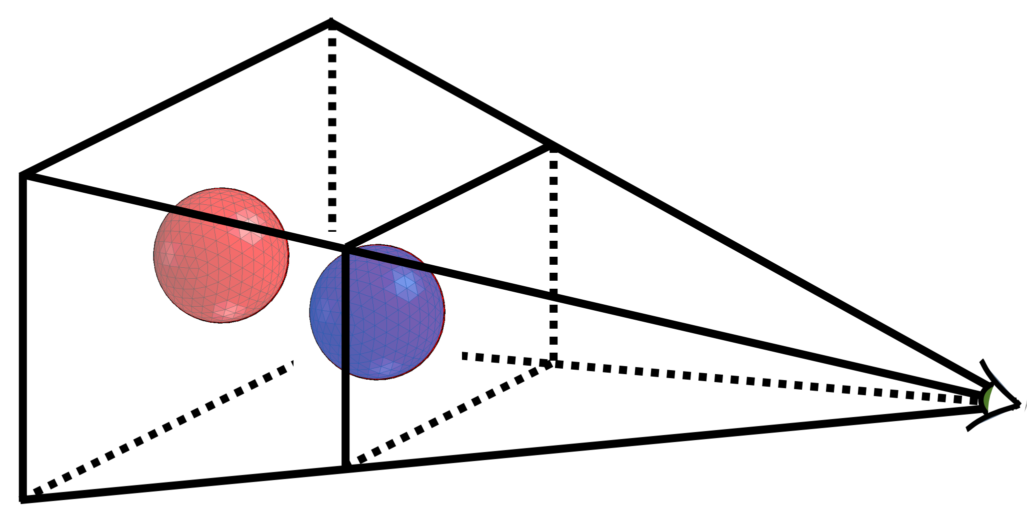
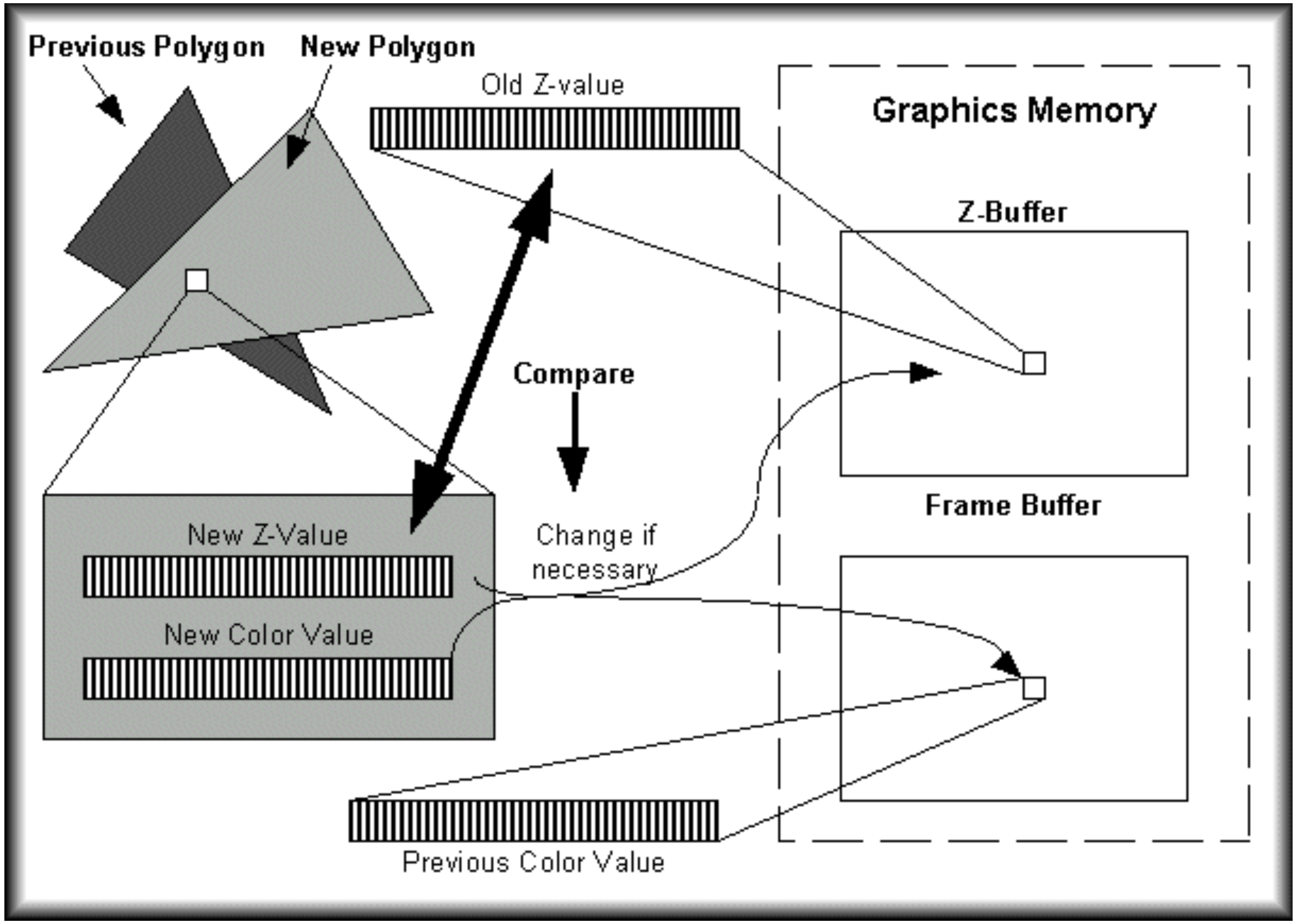


Figure 1. Block diagram of OpenGL.



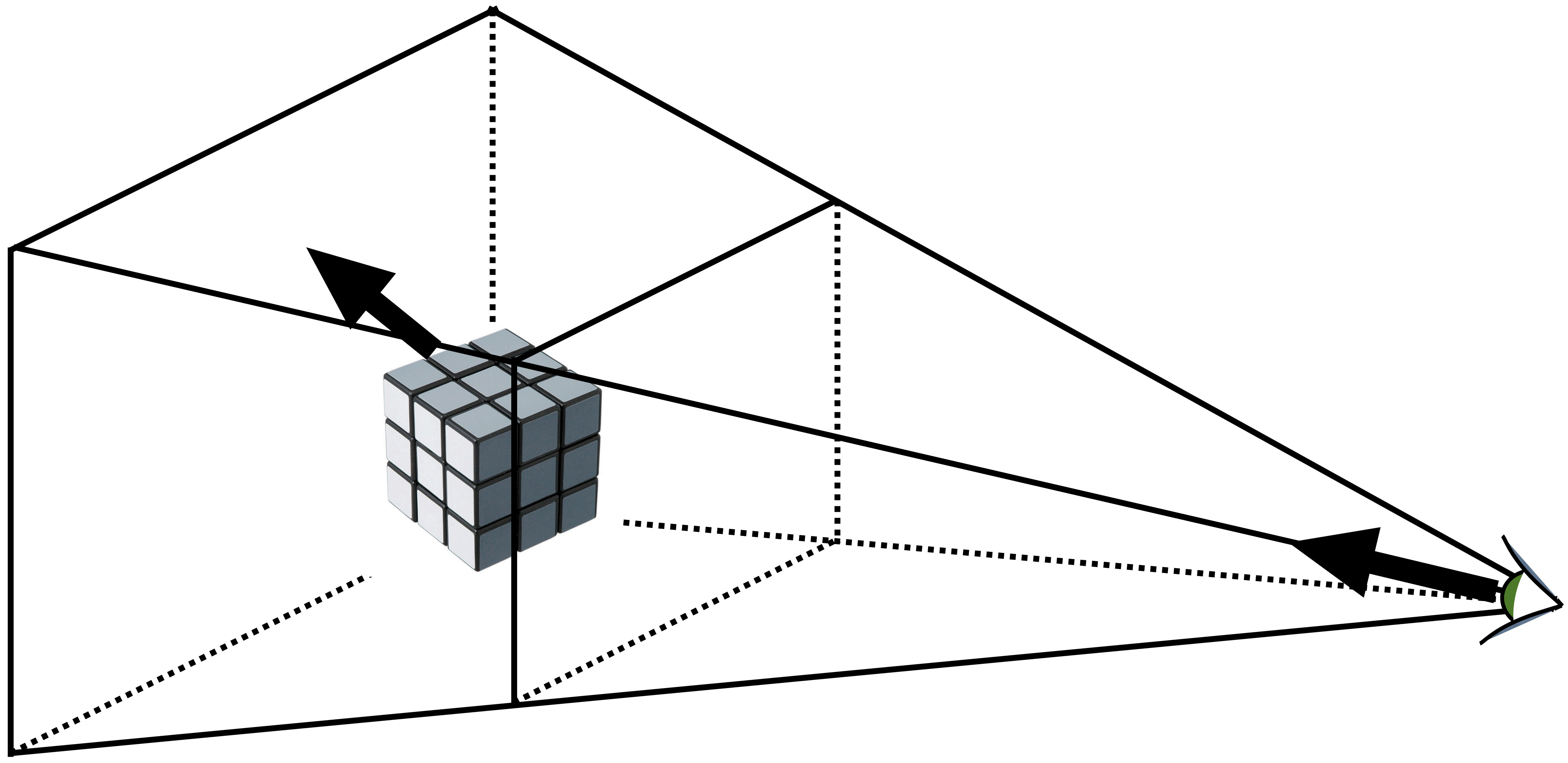


Use a *z-buffer* for hidden surface removal



http://www.beyond3d.com/content/articles/41/

Backface culling: another way to eliminate hidden geometry



Hidden Surface Removal in OpenGL

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

glEnable(GL_DEPTH_TEST);

glEnable(GL_CULL_FACE);
```

For a perspective transformation, there is more precision in the depth buffer for z-values closer to the near plane

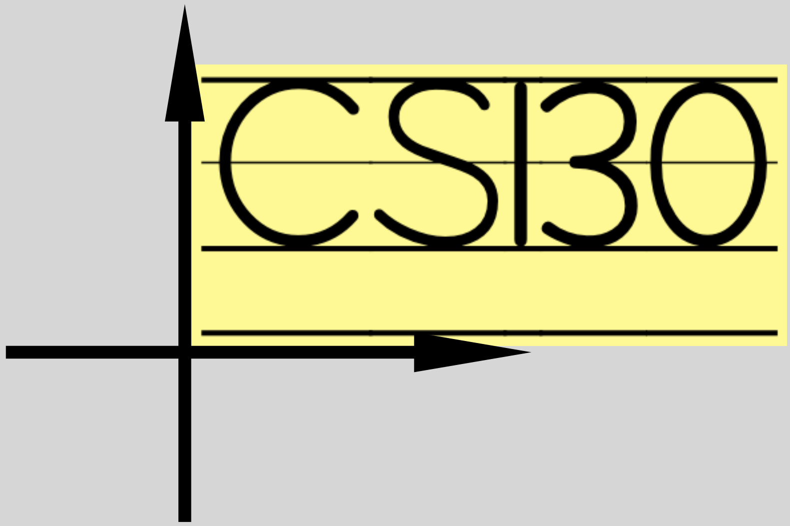
Transformation Matrices
<whiteboard>

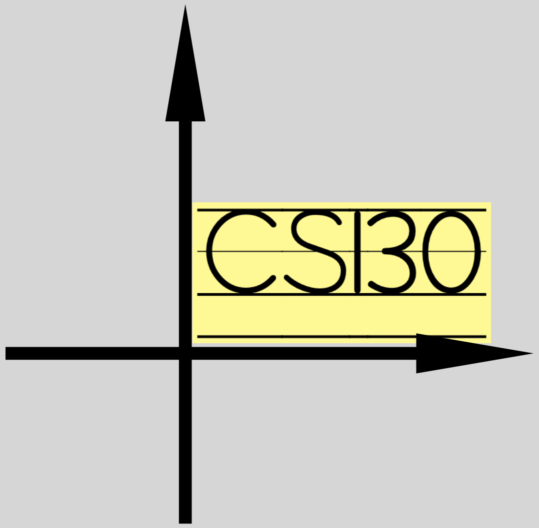
2D Transformations

Uniform Scale

$$\begin{pmatrix} s & 0 \\ 0 & s \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} sx \\ sy \end{pmatrix}$$

$$\begin{pmatrix} .5 & 0 \\ 0 & .5 \end{pmatrix}$$

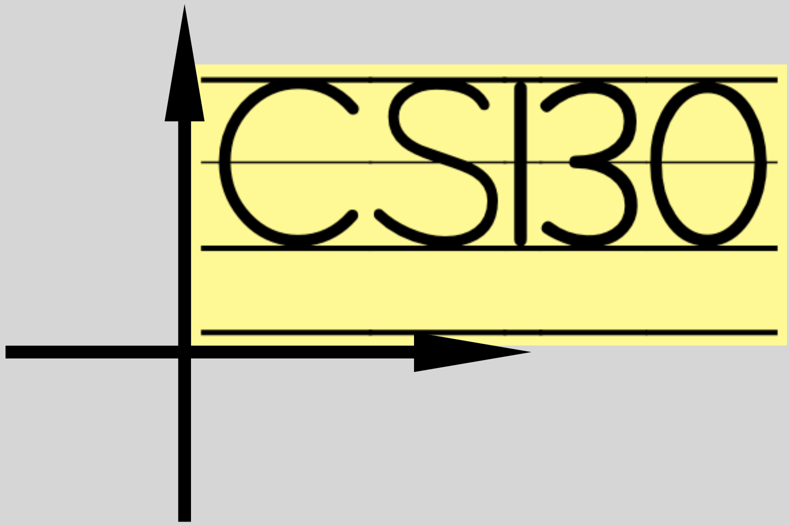




Nonuniform Scale

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

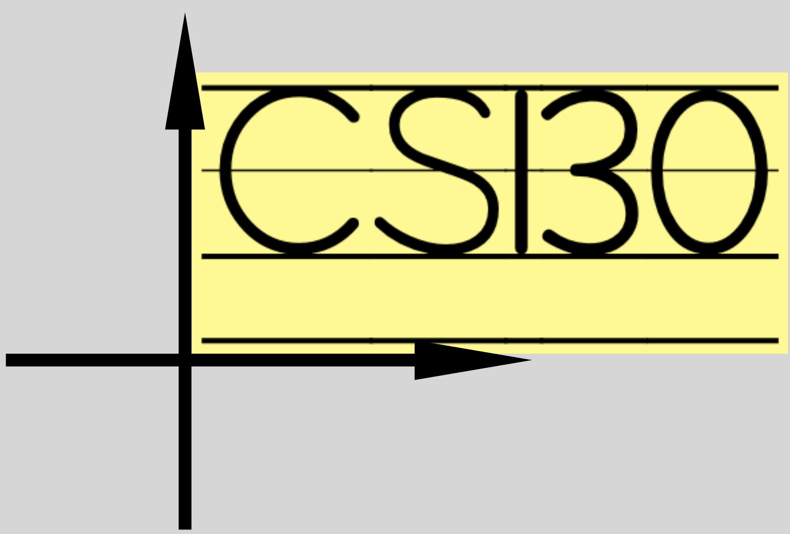
$$\begin{pmatrix} .5 & 0 \\ 0 & 1 \end{pmatrix}$$

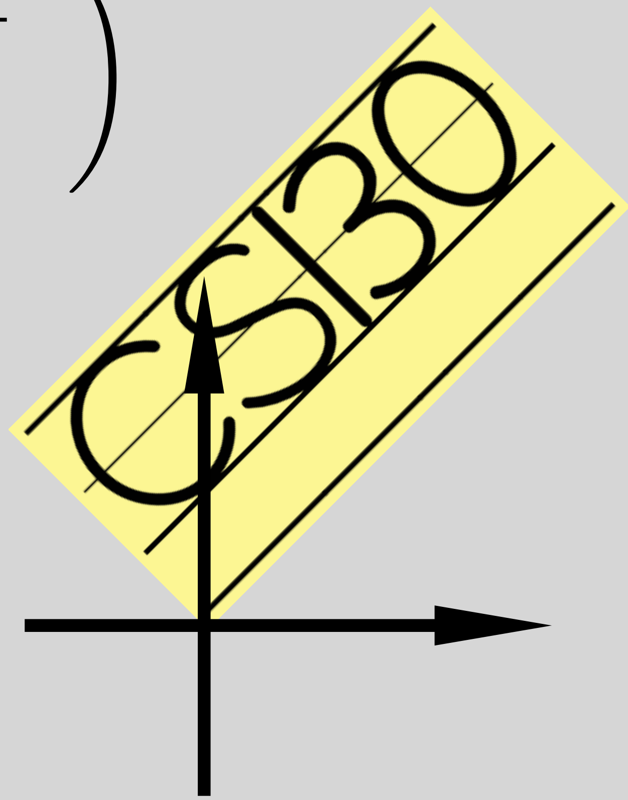


Rotation

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{pmatrix}$$

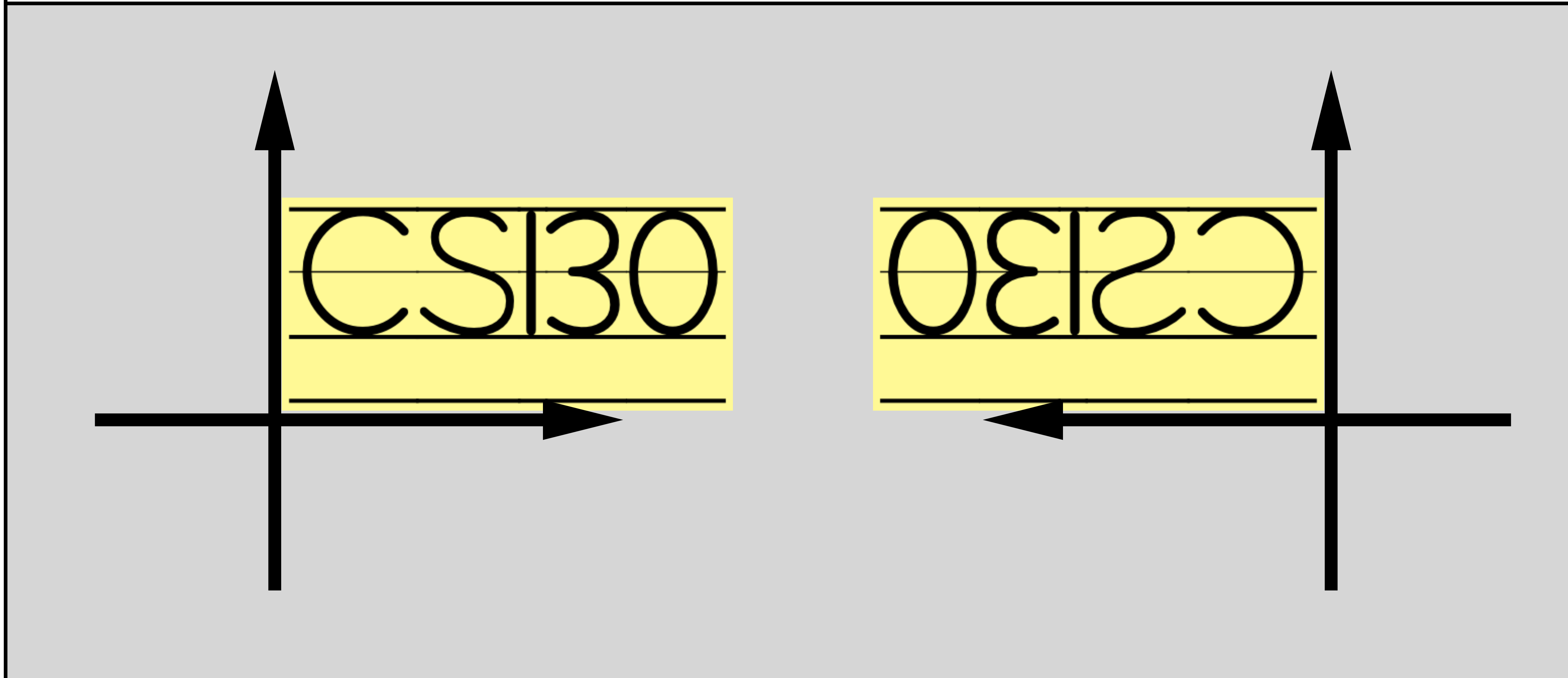
$$\begin{pmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{pmatrix}$$





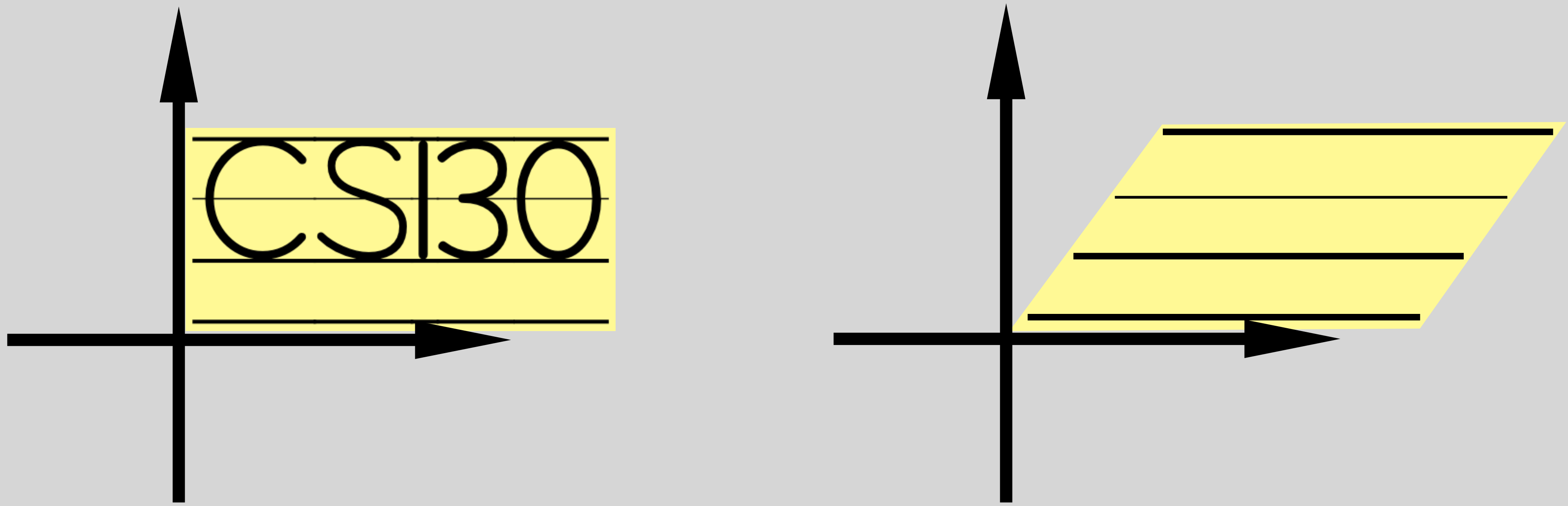
Reflection

$$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -x \\ y \end{pmatrix}$$



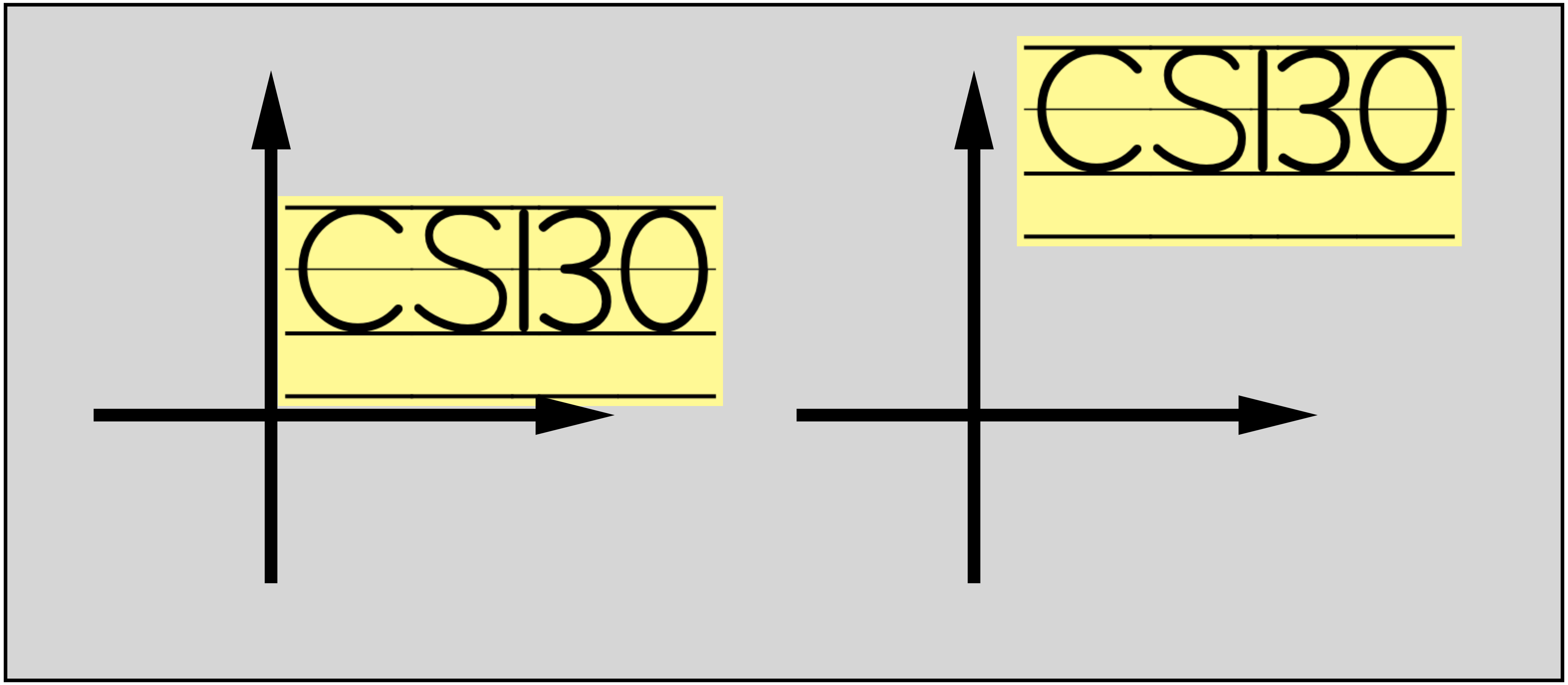
Shear

$$\begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x + ay \\ y \end{pmatrix}$$

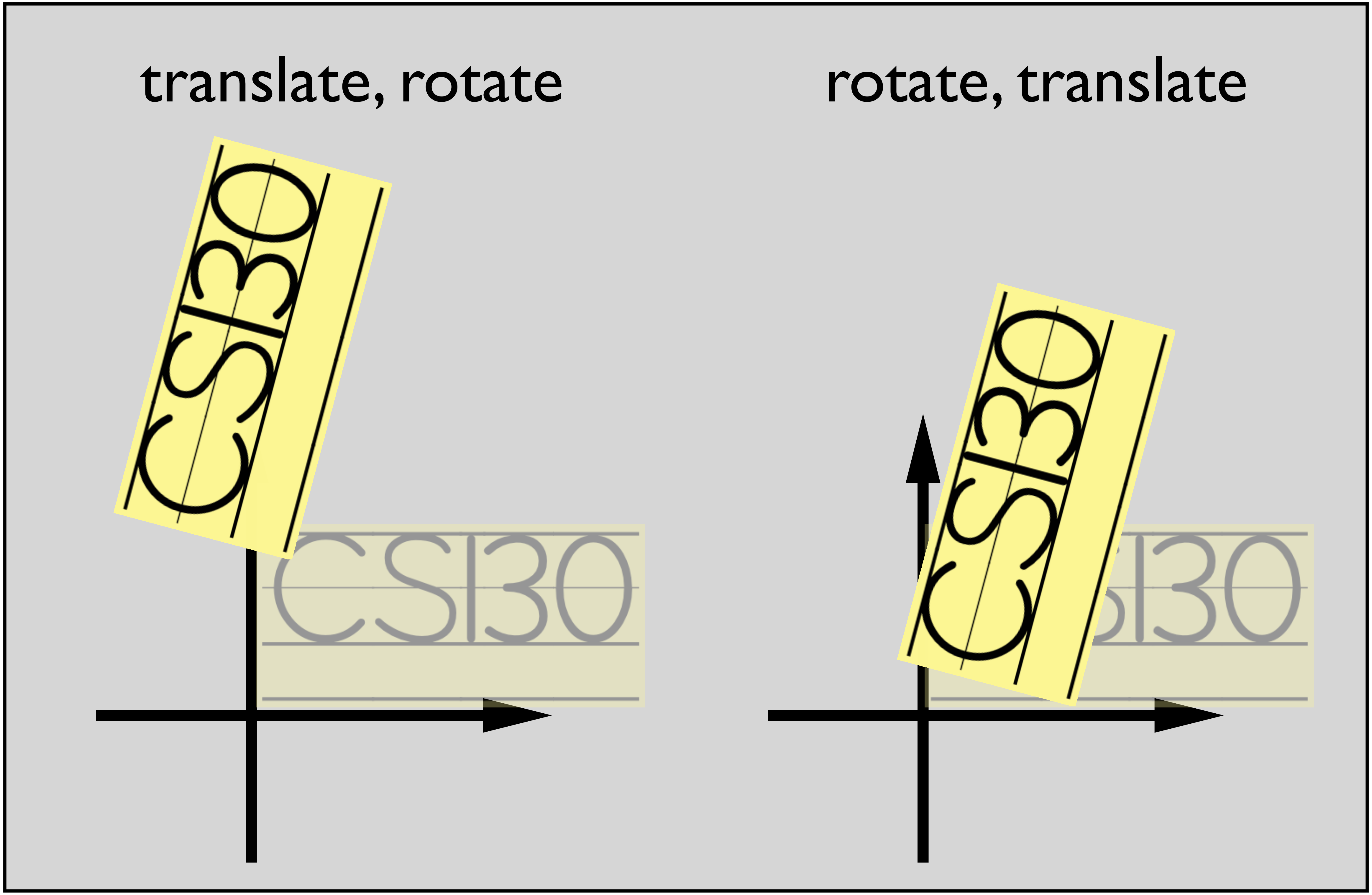


Translation

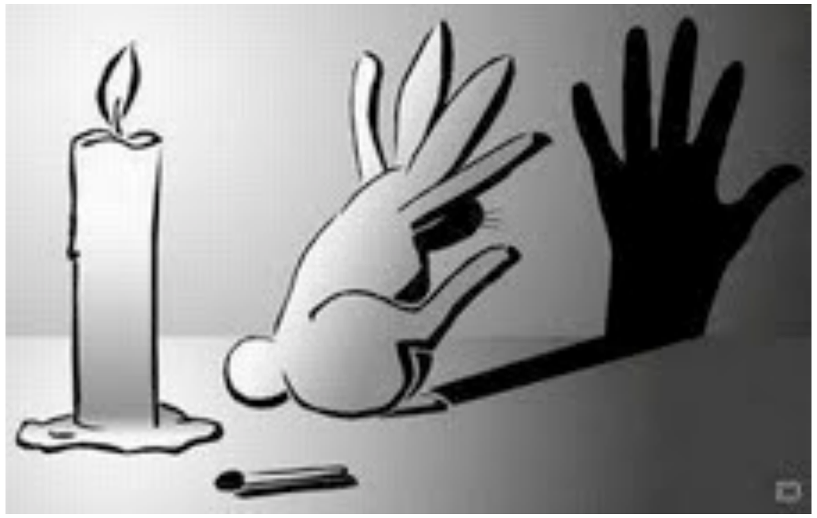
$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}$$



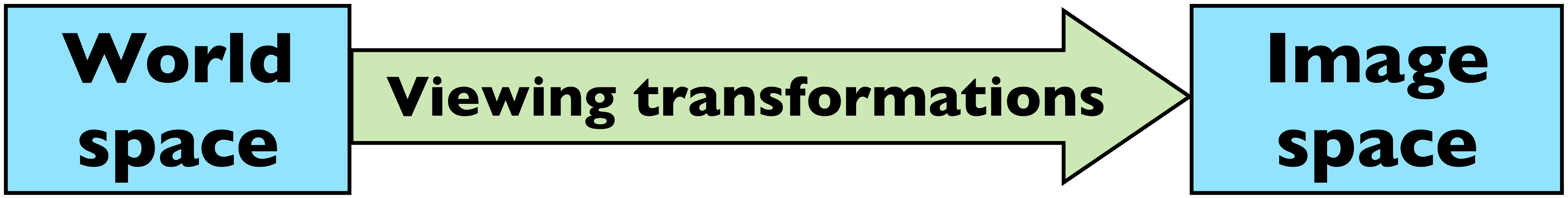
Noncommutativity



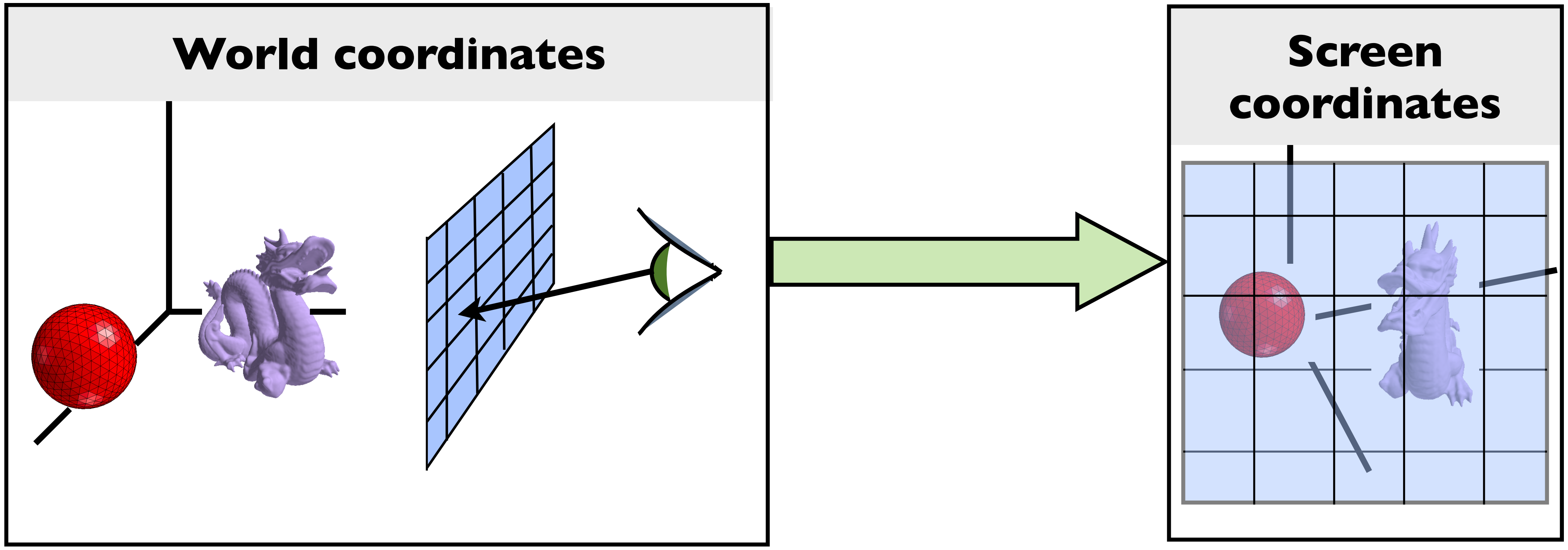
Viewing Transformations



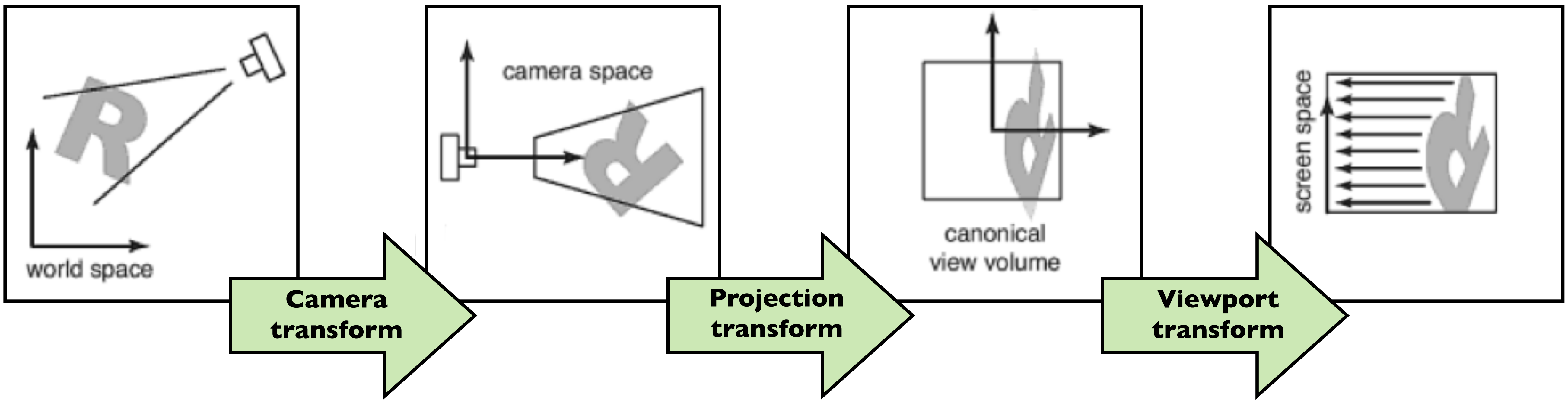
Viewing transformations



- Move objects from their 3D locations to their positions in a 2D view



Decomposition of viewing transforms



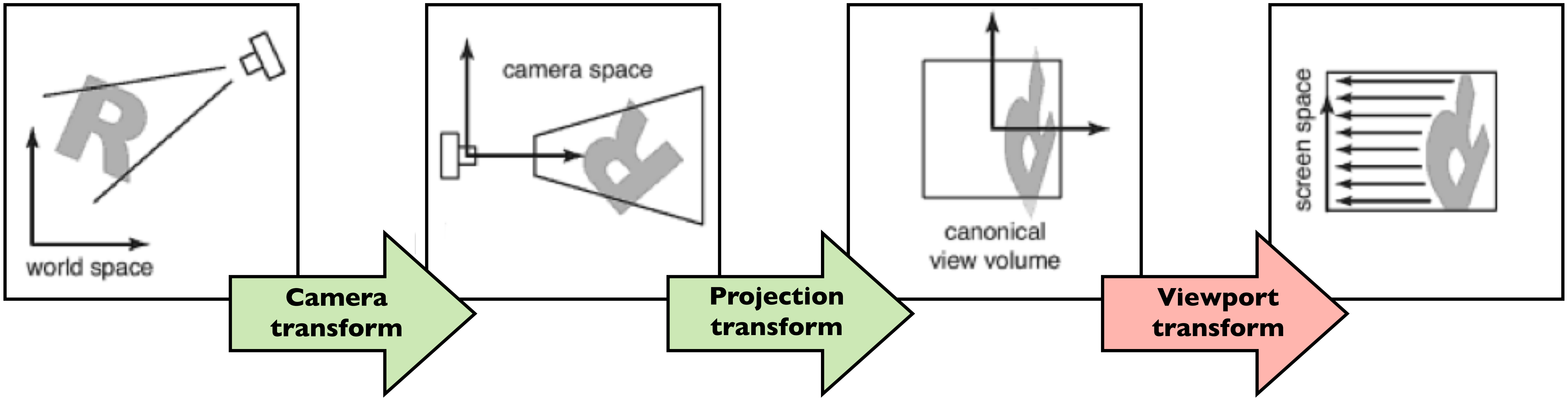
- rigid body transformation
- place camera at origin

- x, y, z in [-1,1]
- depends on type of projection

- map to pixel coordinates

Viewing transforms depend on: camera position and orientation, type of projection, field of view, image resolution

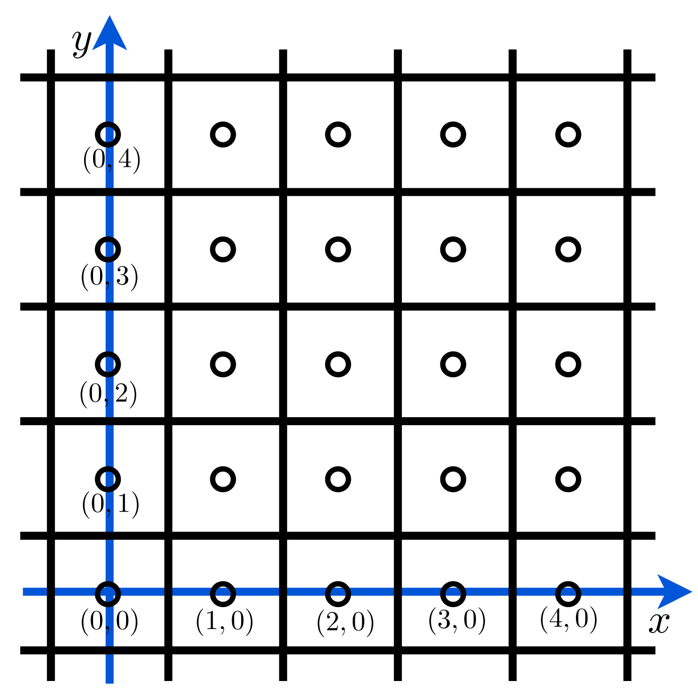
Viewport transform



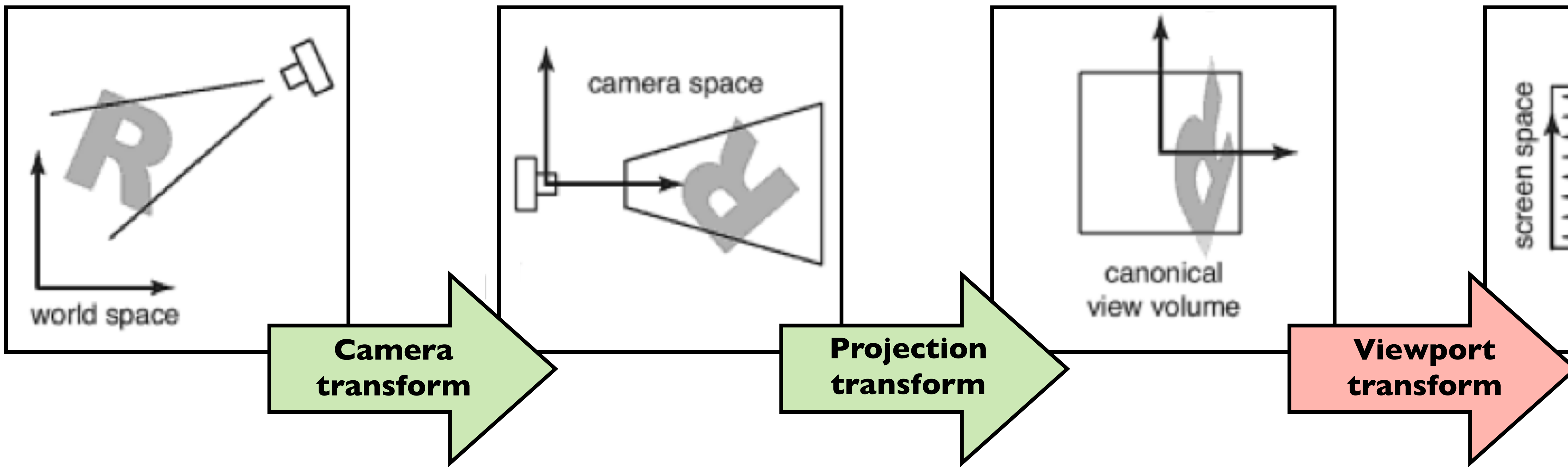
$$(x, y, z) \rightarrow (x', y', z')$$

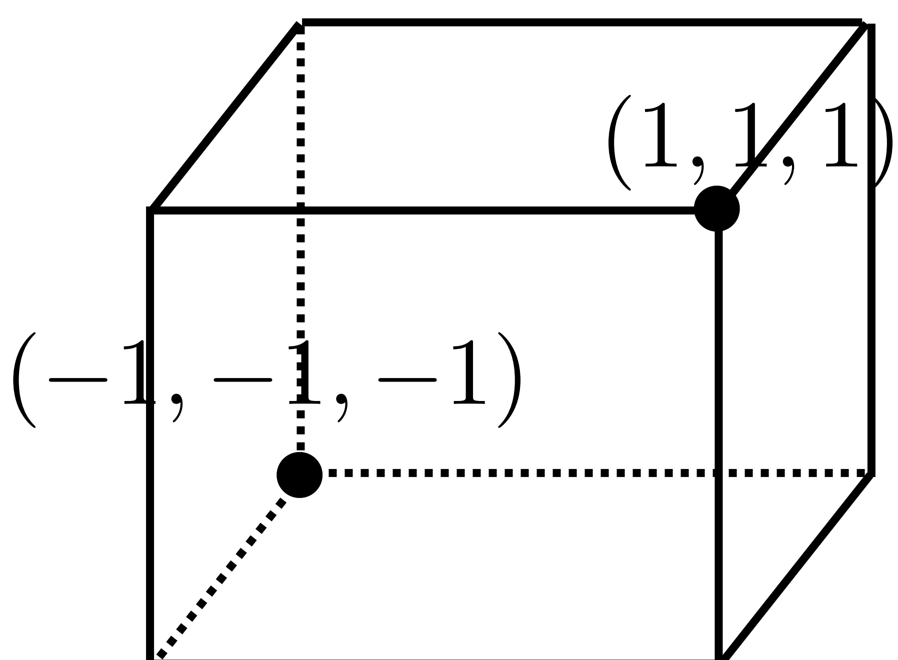
$$(x, y, z) \in [-1, 1]^3$$

$$x' \in [-.5, n_x - .5]$$
$$y' \in [-.5, n_y - .5]$$

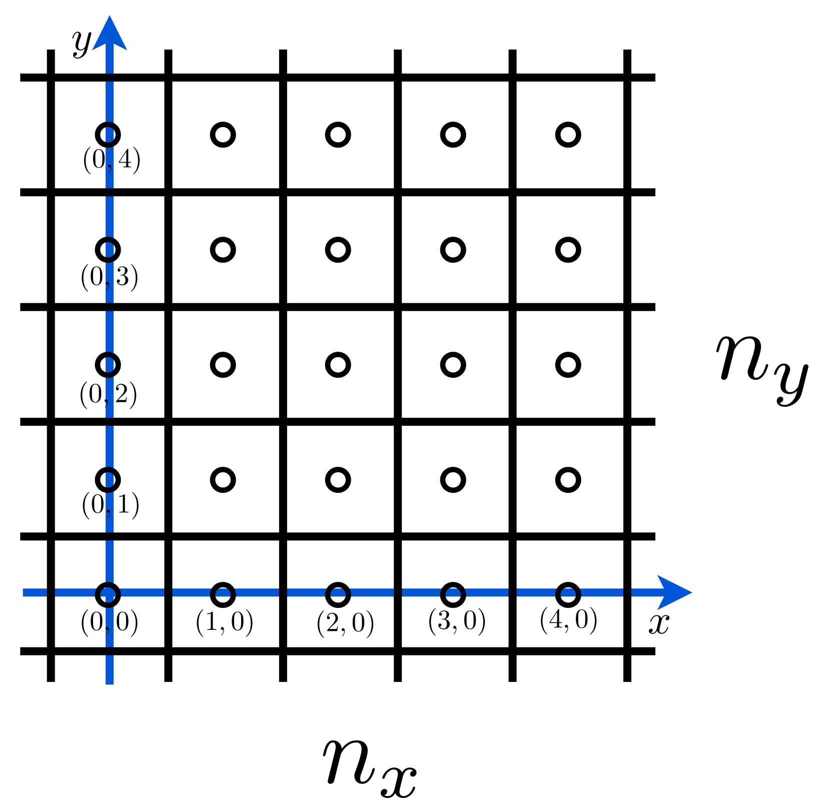


Viewport transform





M_{vp}



<whiteboard>