GEOMETRIC TRANSFORMATIONS AND VIEWING

2D and 3D

2D TRANSFORMATIONS HOMOGENIZED

Transformation	Matrix
Scaling	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Rotation	$\begin{bmatrix} \cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Translation	$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$

- These 3 transformations are all affine transformations

EXAMPLES

- Scaling: *Scale by 15 in the x direction, 17 in the y*

- Rotation: *Rotate by 123°*

- Translation: *Translate by -16 in the x, +18 in the y*

$$\begin{bmatrix} 1 & 0 & -16 \\ 0 & 1 & 18 \\ 0 & 0 & 1 \end{bmatrix}$$

2D INVERSE TRANSFORMATIONS

- How do we find the inverse of a transformation?
- Take the inverse of the transformation matrix (thanks to homogenization, they're all invertible!):

Transformation	Matrix Inverse	Does it make sense?
Scaling	$\begin{bmatrix} 1/s_x & 0 & 0 \\ 0 & 1/s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	If you scale something by factor X, the inverse is scaling by 1/X
Rotation	$\begin{bmatrix} \cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Not so obvious, but can use math! Rotation Matrix is orthonormal, so inverse is just the transpose
Translation	$\begin{bmatrix} 1 & 0 & -dx \\ 0 & 1 & -dy \\ 0 & 0 & 1 \end{bmatrix}$	If you translate by X, the inverse is translating by -X

COMPOSITION OF TRANSFORMATIONS (2D) (1/2)

- We now have a number of tools at our disposal, we can combine them!
- An object in a scene uses many transformations in sequence, how do we represent this in terms of functions?

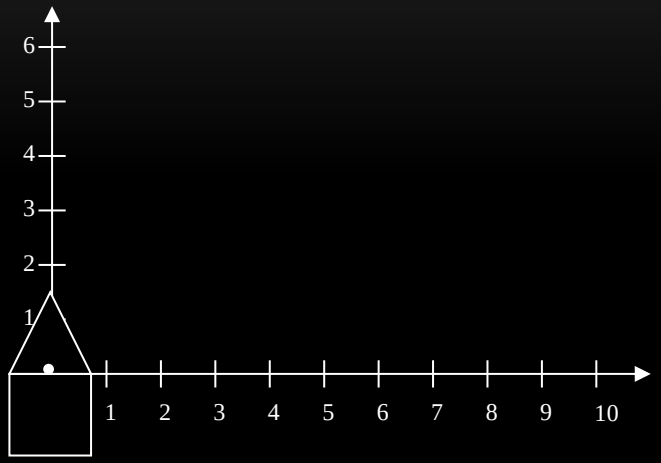
- A transformation is a function; by associativity we can compose functions: $(f \circ g)(i)$
- This is the same as first applying g to some input i then applying f: $(f(g(i)))$
- Consider our functions f and g as matrices (M_1 and M_2 respectively) and our input as a vector ($\boldsymbol{v}$)
- Our composition is equivalent to $M_1 M_2 v$

COMPOSITION OF TRANSFORMATIONS (2D) (2/2)

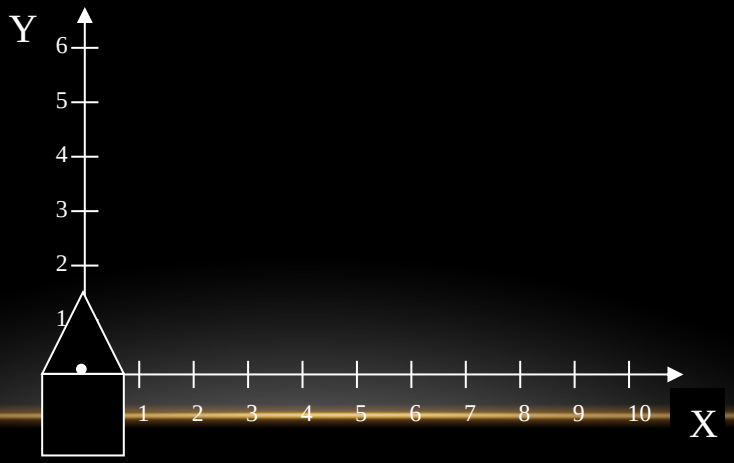
- We can now form compositions of transformation matrices to form a more complex transformation
- For example, $\boldsymbol{TRSv}$, which scales point, then rotates, then translates:
- $$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
- Note that we apply the matrices in sequence right to left, but practically, given associativity, we can compose them and apply the composite to all the vertices in, say, a mesh.
- **Important: Order Matters!**
- Matrix Multiplication is **not commutative.**
- Let's do some math…!!!!

NOT COMMUTATIVE

Translate by x=6, y=0 then rotate by 45º

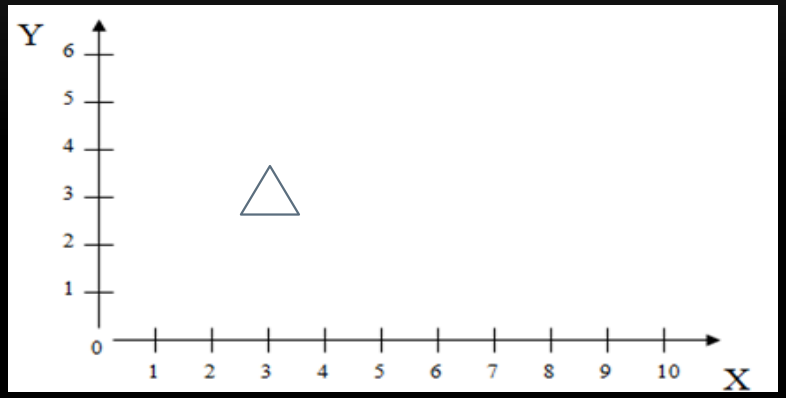


Rotate by 45º then translate by x=6, y=0

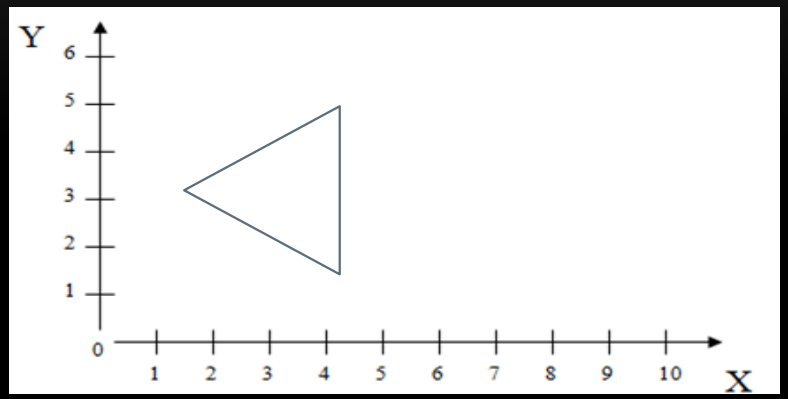


COMPOSITION (AN EXAMPLE) (2D) (1/2)

- Start:



Goal:



- Important concept: Make the problem simpler
- Translate object to origin first, scale , rotate, and translate back $\boldsymbol{T^{-1}RST}$

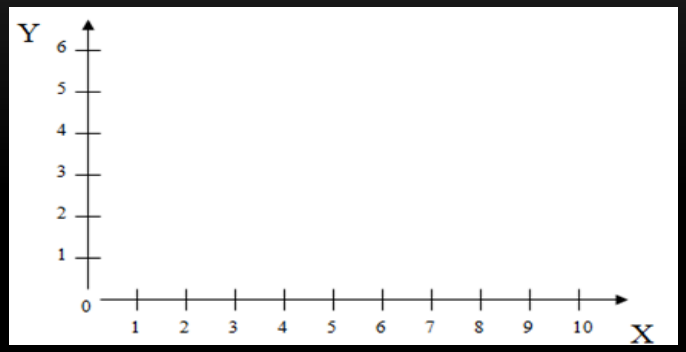
$$\begin{bmatrix}1 & 0 & 3\\ 0 & 1 & 3\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}cos90 & -sin90 & 0\\ sin90 & cos90 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}3 & 0 & 0\\ 0 & 3 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & -3\\ 0 & 1 & -3\\ 0 & 0 & 1\end{bmatrix}$$

- Apply to all vertices

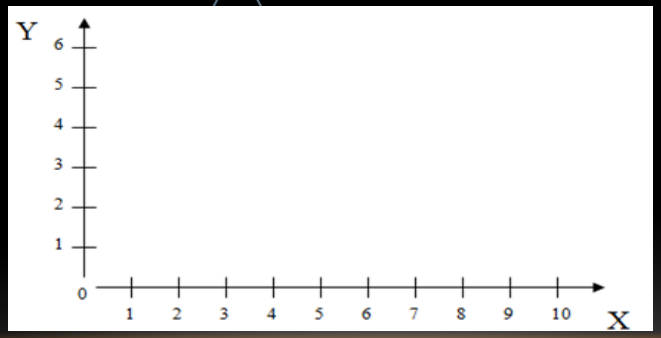
Rotate 90°
Uniform Scale 3x
Both around object's center, not the origin

COMPOSITION (AN EXAMPLE) (2D) (2/2)

- $\boldsymbol{T^{-1}RST}$



- But what if we mixed up the order? Let's try $\boldsymbol{RT^{-1}ST}$
  - $\begin{bmatrix} cos90 & -sin90 & 0 \\ sin90 & cos90 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$



- Oops! We managed to scale it properly but when we rotated it we rotated the object about the origin, not its own center, shifting its position…**Order Matters!**

ASIDE: TRANSFORMING COORDINATE AXES

- We understand linear transformations as changing the position of vertices relative to the standard axes
- Can also think of transforming the coordinate axes themselves
- Just as in matrix composition, be careful of which order you modify your coordinate system

EXAMPLE IN 3D!

- Let's take some 3D object, say a cube, centered at (2,2,2)
- Rotate in object's space by 30° around x axis, 60° around y and 90° around z
- Scale in object space by 1 in the x, 2 in the y, 3 in the z
- Translate by (2,2,4)
- Transformation Sequence: $\boldsymbol{T T_0^{-1} S_{xy} R_{xy} R_{xz} R_{yz} T_0}$, where $\boldsymbol{T_0}$ translates to $(0,0)$
- $$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos90 & sin90 & 0 & 0 \\ -sin90 & cos90 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\begin{bmatrix} cos60 & 0 & sin60 & 0 \\ 0 & 1 & 0 & 0 \\ -sin60 & 0 & cos60 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos30 & sin30 & 0 \\ 0 & -sin30 & cos30 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

COMPUTER VIEWING

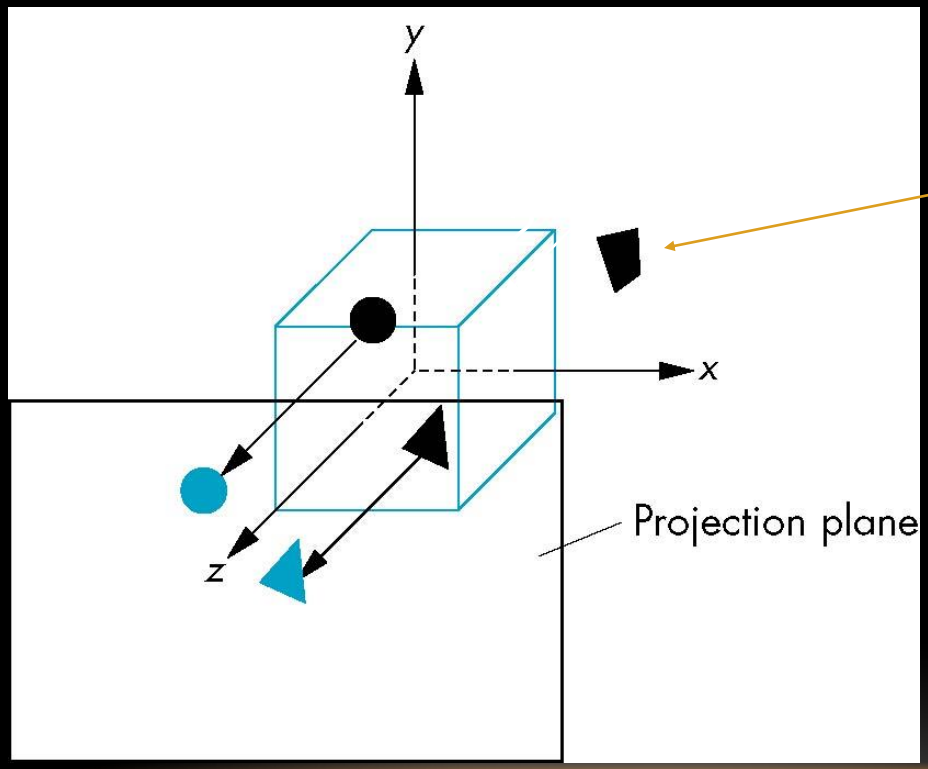
- There are three aspects of the viewing process, all of which are implemented in the pipeline,
  - Positioning the camera
    - Setting the model-view matrix
  - Selecting a lens
    - Setting the projection matrix
  - Clipping
    - Setting the view volume

THE OPENGL CAMERA

- In OpenGL, initially the object and camera frames are the same
  - Default model-view matrix is an identity
- The camera is located at origin and points in the negative z direction
- OpenGL also specifies a default view volume that is a cube with sides of length 2 centered at the origin
  - Default projection matrix is an identity

DEFAULT PROJECTION

Default projection is orthogonal



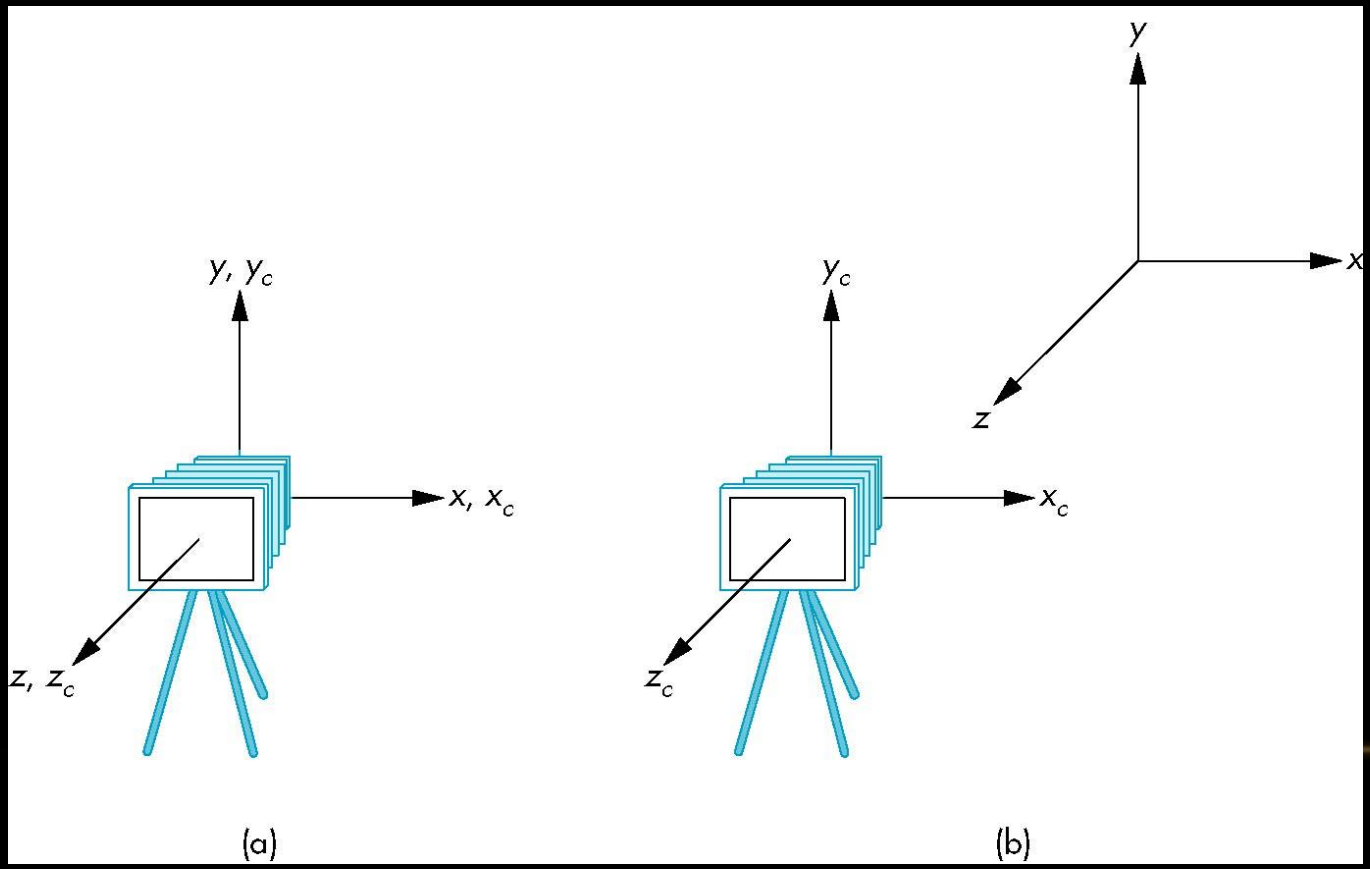
MOVING THE CAMERA FRAME

- If we want to visualize object with both positive and negative z values we can either
  - Move the camera in the positive z direction
    - Translate the camera frame
  - Move the objects in the negative z direction
    - Translate the world frame
- Both of these views are equivalent and are determined by the model-view matrix
  - Want a translation (`Translate(0.0,0.0,-d);`)
  - `d > 0`

MOVING CAMERA BACK FROM ORIGIN

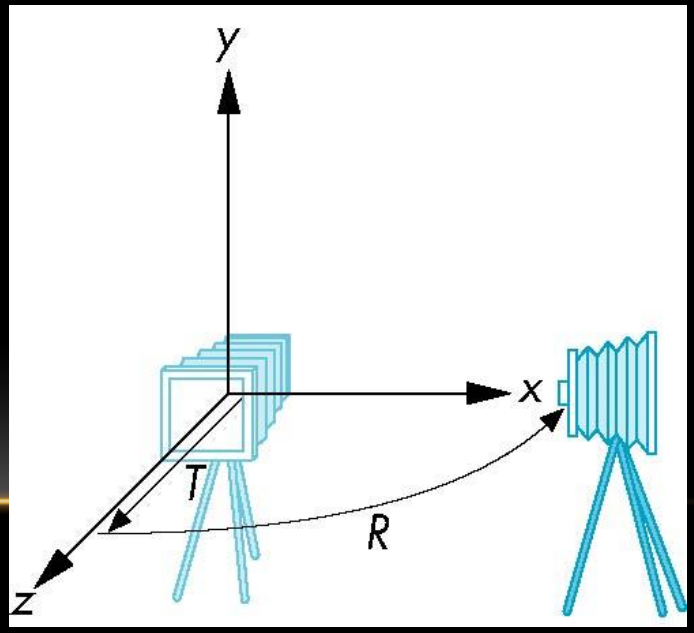
frames after translation by –d

d > 0



MOVING THE CAMERA

- We can move the camera to any desired position by a sequence of rotations and translations
- Example: side view
  - Rotate the camera
  - Move it away from origin
  - Model-view matrix C = TR



PROJECTIONS AND NORMALIZATION

- The default projection in the eye (camera) frame is orthogonal
- For points within the default view volume

$$x_p = x$$
$$y_p = y$$
$$z_p = 0$$

- Most graphics systems use *view normalization*
  - All other views are converted to the default view by transformations that determine the projection matrix
  - Allows use of the same pipeline for all views

HOMOGENEOUS COORDINATE REPRESENTATION

default orthographic projection

$x_p = x$

$y_p = y$

$z_p = 0$

$w_p = 1$

$$\mathbf{p}_p = \mathbf{M}\mathbf{p}$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In practice, we can let $\mathbf{M} = \mathbf{I}$ and set the z term to zero later