



# Modeling Transformations

Michael Kazhdan

(601.457/657)

# Announcements



Recall: Midterm will be March 3<sup>rd</sup>.



# Overview

- Ray-Tracing so far
- Modeling transformations



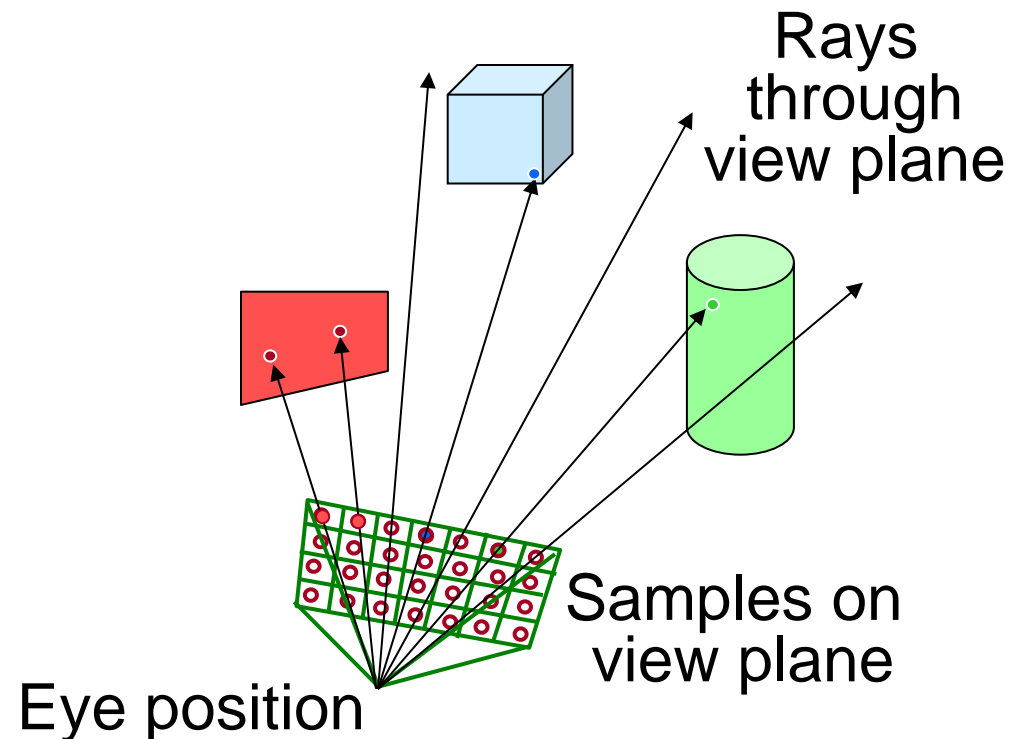
# Ray Tracing

```
Image RayTrace( Camera camera , Scene scene , int width , int height , int depth , float cutoff )
{
    Image image( width , height );
    for( int i=0 ; i<width ; i++ ) for( int j=0 ; j<height ; j++ )
    {
        Ray< 3 > ray = ConstructRayThroughPixel( camera , i , j );
        image[i][j] = GetColor( scene , ray , depth, Color( cutOff , cutOff , cutOff ) );
    }
    return image;
}
```



# Ray Tracing

- For each sample ...
  - Construct ray from eye position through view plane
  - Compute color contribution of the ray





# Ray Tracing

```
Image RayTrace( Camera camera , Scene scene , int width , int height , int depth , float cutoff )
{
    Image image( width , height );
    for( int j=0 ; j<height ; j++ ) for( int i=0 ; i<width ; i++ )
    {
        Ray< 3 > ray = ConstructRayThroughPixel( camera , i , j );
        image[i][j] = GetColor( scene , ray , depth, Color( cutOff , cutOff , cutOff ) );
    }
    return image;
}
```



# Constructing Ray Through a Pixel

- 2D Example:

The ray passing through the  $i$ -th pixel is defined by:

$$\text{Ray}(t) = e + t \cdot \vec{v}$$

- $e$ : camera position
- $\vec{v}$ : direction to the  $i$ -th pixel:

$$\vec{v} = \frac{p[i] - e}{\|p[i] - e\|}$$

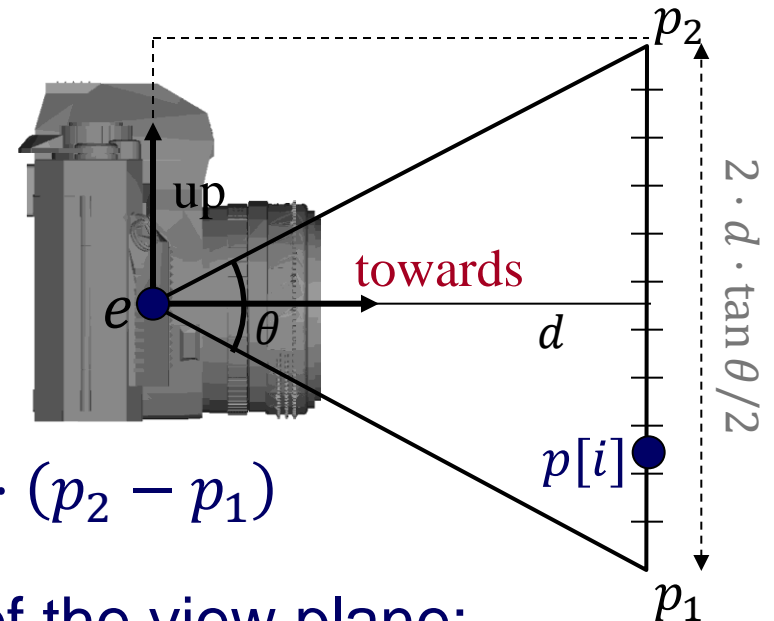
- $p[i]$ :  $i$ -th pixel location:

$$p[i] = p_1 + \left( \frac{i + 0.5}{\text{height}} \right) \cdot (p_2 - p_1)$$

- $p_1$  and  $p_2$  are the endpoints of the view plane:

$$p_1 = e + d \cdot \text{towards} - d \cdot \tan \theta/2 \cdot \text{up}$$

$$p_2 = e + d \cdot \text{towards} + d \cdot \tan \theta/2 \cdot \text{up}$$



# Ray Tracing



```
Image RayTrace( Camera camera , Scene scene , int width , int height , int depth , float cutoff )
{
    Image image( width , height );
    for( int j=0 ; j<height ; j++ ) for( int i=0 ; i<width ; i++ )
    {
```

```
        Color GetColor( Scene scene , Ray< 3 > ray , int d , Color cutOff )
        {
            HitInformation hit;
            Color c(0,0,0);

            if( !d || (cutOff[0]>1 && cutOff[1]>1 && cutOff[2]>1) ) return c;

            if( Intersect( scene , ray , hit ) ≥ 0 )
            {
                c = GetEmissive( hit );
                for( i=0 ; i<lightNum ; i++ ) c += light[i].getAmbientDiffuseAndSpecular( ray , hit );
                if( Dot( ray.direction , hit.normal )<0 )
                {
                    Ray< 3 > reflectedRay = ReflectedRay( ray , hit.p , hit.n );
                    c += GetColor( scene , reflectedRay , d-1 , cutOff /hit.spec ) * hit.spec;
                }
                Ray< 3 > refractedRay = RefractedRay( ray , hit.p , hit.n , hit.ir );
                c += GetColor( scene , refractedRay , d-1 , cutOff /hit.tran ) * hit.tran;
            }
            return c;
        }
    }
}
```



# Ray Tracing



```
Image RayTrace( Camera camera , Scene scene , int width , int height , int depth , float cutoff )
{
    Image image( width , height );
    for( int j=0 ; j<height ; j++ ) for( int i=0 ; i<width ; i++ )
    {
```

```
        Color GetColor( Scene scene , Ray< 3 > ray , int d , Color cutOff )
        {
            HitInformation hit;
            Color c(0,0,0);

            if( !d || (cutOff[0]>1 && cutOff[1]>1 && cutOff[2]>1) ) return c;

            if( Intersect( scene , ray , hit )≥0 )
            {
                c = GetEmissive( hit );
                for( i=0 ; i<lightNum ; i++ ) c += light[i].getAmbientDiffuseAndSpecular( ray , hit );
                if( Dot( ray.direction , hit.normal )<0 )
                {
                    Ray< 3 > reflectedRay = ReflectedRay( ray, hit.p, hit.n );
                    c += GetColor( scene , reflectedRay , d-1 , cutOff /hit.spec ) * hit.spec;
                }
                Ray< 3 > refractedRay = RefractedRay( ray, hit.p , hit.n , hit.ir );
                c += GetColor( scene , refractedRay , d-1 , cutOff /hit.tran ) * hit.tran;
            }
            return c;
        }
    }
}
```



# Ray-Scene Intersection

- Intersections with geometric primitives
  - Sphere
  - Triangle
  - Groups of primitives (scene)
- Acceleration techniques
  - Bounding volume hierarchies
  - Spatial partitions
    - » Uniform grids
    - » Octrees
    - » BSP trees

# Ray Tracing



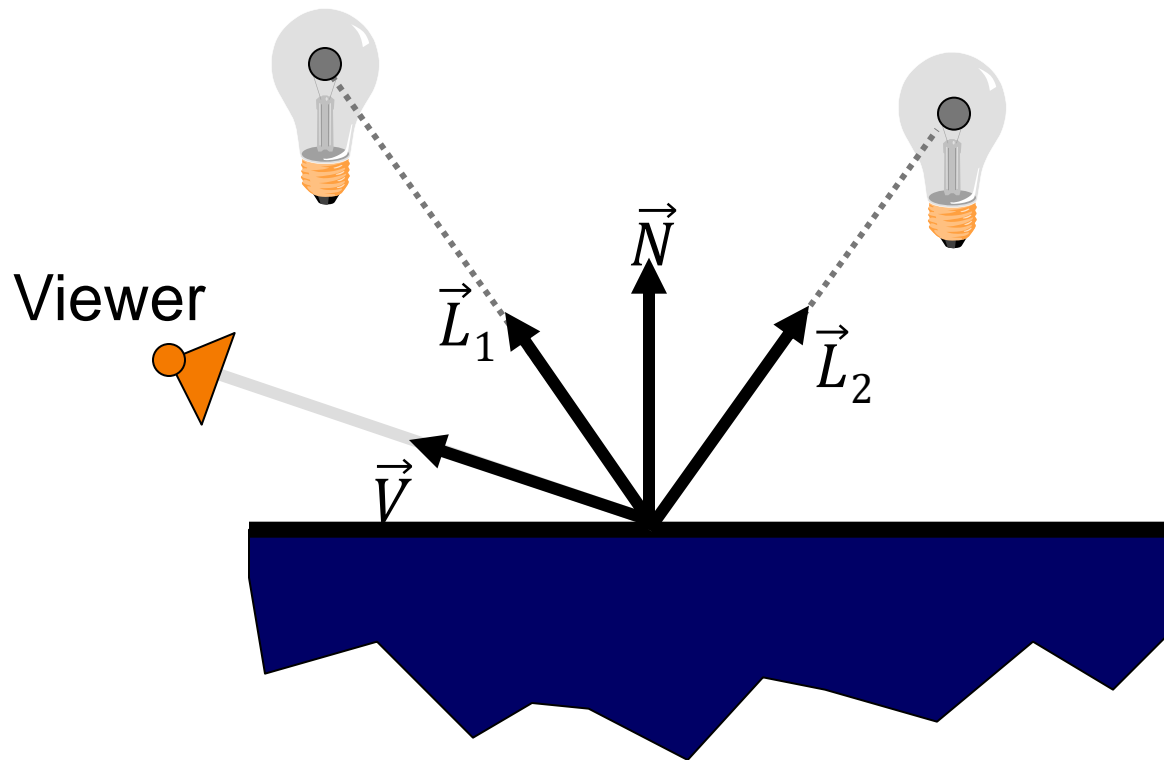
```
Image RayTrace( Camera camera , Scene scene , int width , int height , int depth , float cutoff )
{
    Image image( width , height );
    for( int j=0 ; j<height ; j++ ) for( int i=0 ; i<width ; i++ )
    {
```

```
        Color GetColor( Scene scene , Ray< 3 > ray , int d , Color cutOff )
        {
            HitInformation hit;
            Color c(0,0,0);

            if( !d || (cutOff[0]>1 && cutOff[1]>1 && cutOff[2]>1) ) return c;

            if( Intersect( scene , ray , hit ) ≥ 0 )
            {
                c = GetEmissive( hit );
                for( i=0 ; i<lightNum ; i++ ) c += light[i].getAmbientDiffuseAndSpecular( ray , hit );
                if( Dot( ray.direction , hit.normal )<0 )
                {
                    Ray< 3 > reflectedRay = ReflectedRay( ray , hit.p , hit.n );
                    c += GetColor( scene , reflectedRay , d-1 , cutOff /hit.spec ) * hit.spec;
                }
                Ray< 3 > refractedRay = RefractedRay( ray , hit.p , hit.n , hit.ir );
                c += GetColor( scene , refractedRay , d-1 , cutOff /hit.tran ) * hit.tran;
            }
            return c;
        }
    }
}
```

# Surface Illumination Calculation



$$I = I_E + \sum_L \left[ K_A \cdot I_L^A + \left( K_D \cdot \langle \vec{N}, \vec{L} \rangle + K_S \cdot \langle \vec{V}, \vec{R} \rangle^n \right) \cdot I_L \cdot S_L \right]$$



# Ray Tracing

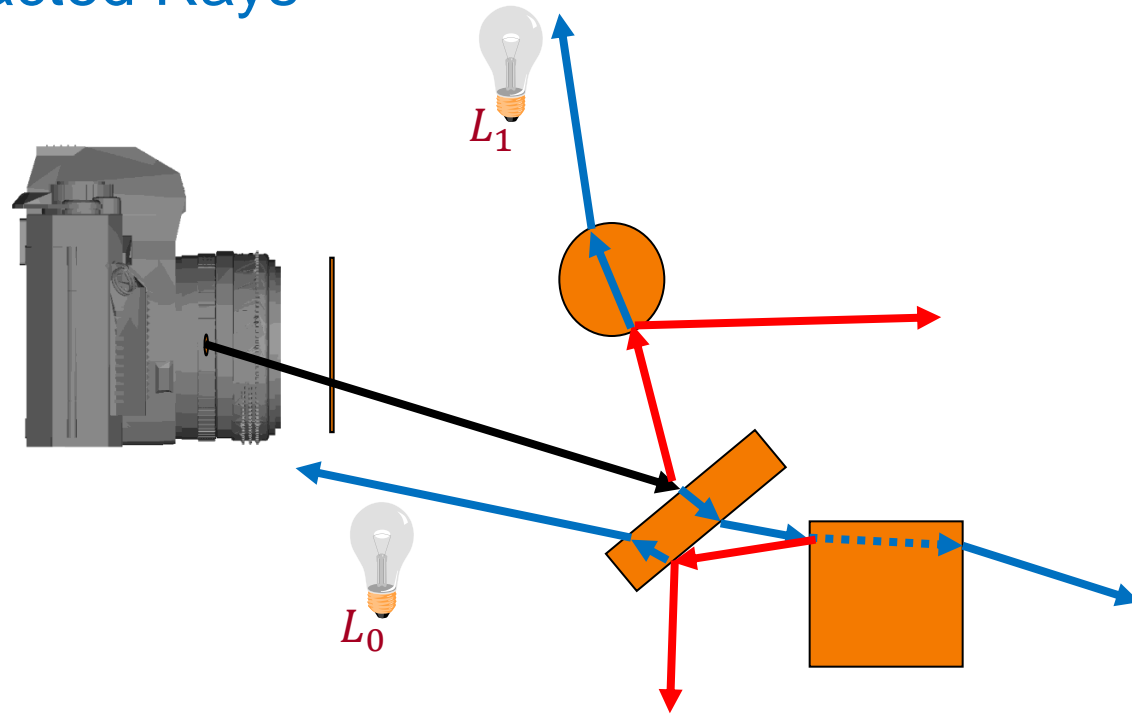
```
Image RayTrace( Camera camera , Scene scene , int width , int height , int depth , float cutoff )  
{  
    Image image( width , height );  
    for( int j=0 ; j<height ; j++ ) for( int i=0 ; i<width ; i++ )  
    {
```

```
        Color GetColor( Scene scene , Ray< 3 > ray , int d , Color cutOff )  
        {  
            HitInformation hit;  
            Color c(0,0,0);  
  
            if( !d || (cutOff[0]>1 && cutOff[1]>1 && cutOff[2]>1) ) return c;  
  
            if( Intersect( scene , ray , hit ) ≥ 0 )  
            {  
                c = GetEmissive( hit );  
                for( i=0 ; i<lightNum ; i++ ) c += light[i].getAmbientDiffuseAndSpecular( ray , hit );  
                if( Dot( ray.direction , hit.normal )<0 )  
                {  
                    Ray< 3 > reflectedRay = ReflectedRay( ray , hit.p , hit.n );  
                    c += GetColor( scene , reflectedRay , d-1 , cutOff /hit.spec ) * hit.spec;  
                }  
                Ray< 3 > refractedRay = RefractedRay( ray , hit.p , hit.n , hit.ir );  
                c += GetColor( scene , refractedRay , d-1 , cutOff /hit.tran ) * hit.tran;  
            }  
            return c;  
        }  
    }  
}
```



# Ray Tracing (Recursive)

- Consider the contribution of:
  - Reflected Rays
  - Refracted Rays



$$I = I_E + \sum_L [K_A \cdot I_L^A + (K_D \cdot \langle \vec{N}, \vec{L} \rangle + K_S \cdot \langle \vec{V}, \vec{R} \rangle^n) \cdot I_L \cdot S_L] + K_S \cdot I_R + K_T \cdot I_T$$

# Overview



- Raytracing so far
- Modeling transformations



# Modeling Transformations

- Specify transformations for objects
  - Allows definitions of objects in own coordinate systems
  - Allows use of object definition multiple times in a scene





# Overview

- 2D Transformations
  - Basic 2D transformations
  - Matrix representation
  - Matrix composition
- 3D Transformations
  - Basic 3D transformations
  - Same as 2D

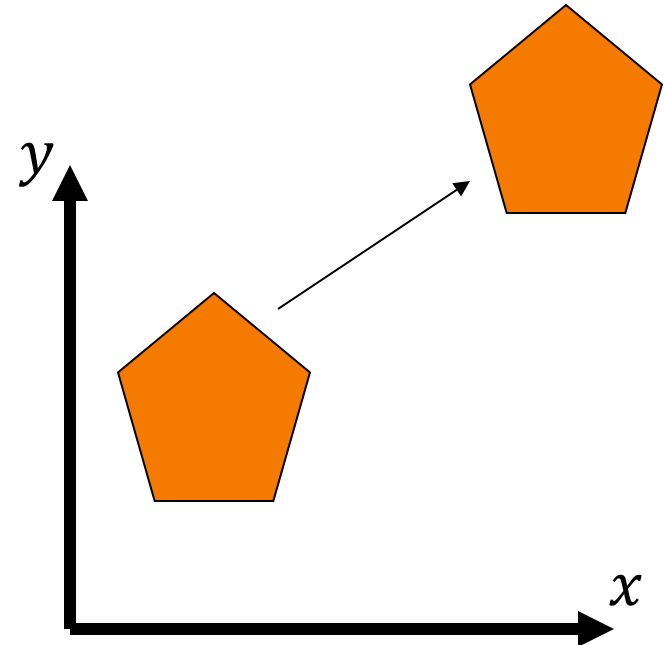


# Simple 2D Transformations

## Translation

$$p' = p + t$$

$$\begin{bmatrix} p'_x \\ p'_y \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$



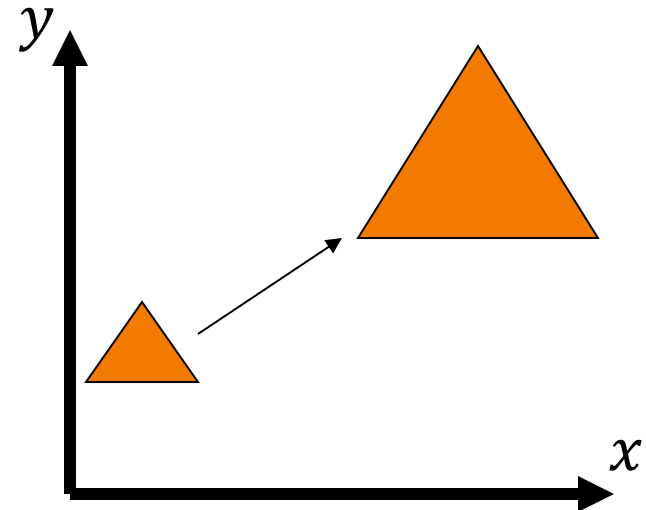


# Simple 2D Transformations

## Scale

$$p' = S \cdot p$$

$$\begin{bmatrix} p'_x \\ p'_y \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix}$$



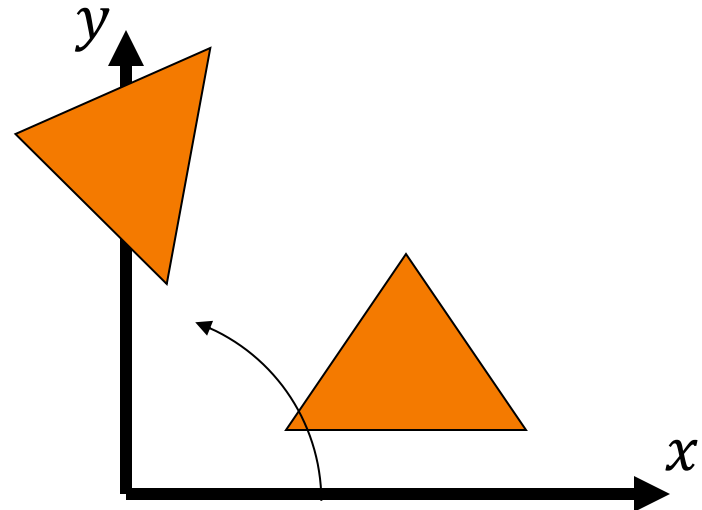


# Simple 2D Transformation

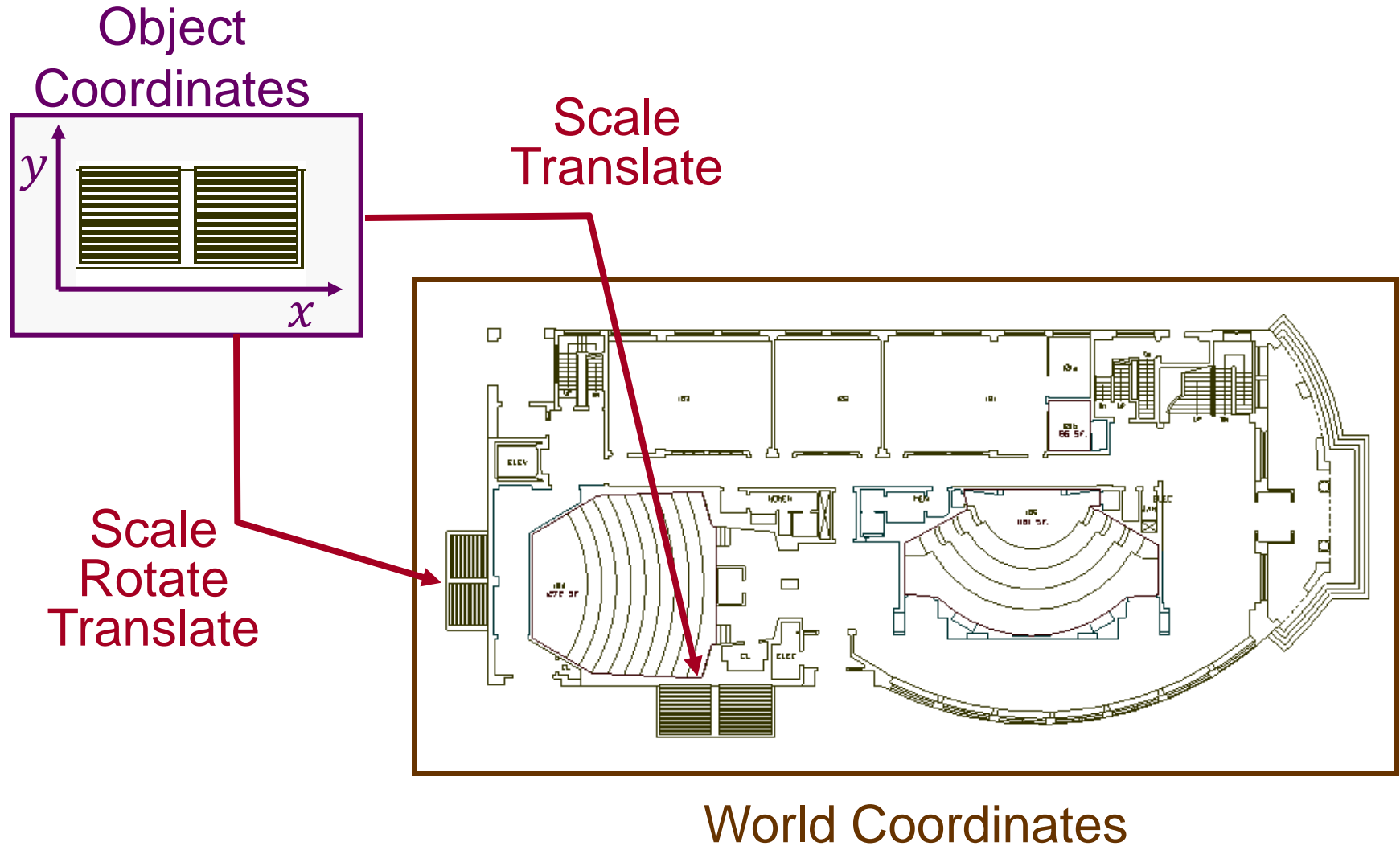
## Rotation

$$p' = R \cdot p$$

$$\begin{bmatrix} p'_x \\ p'_y \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix}$$



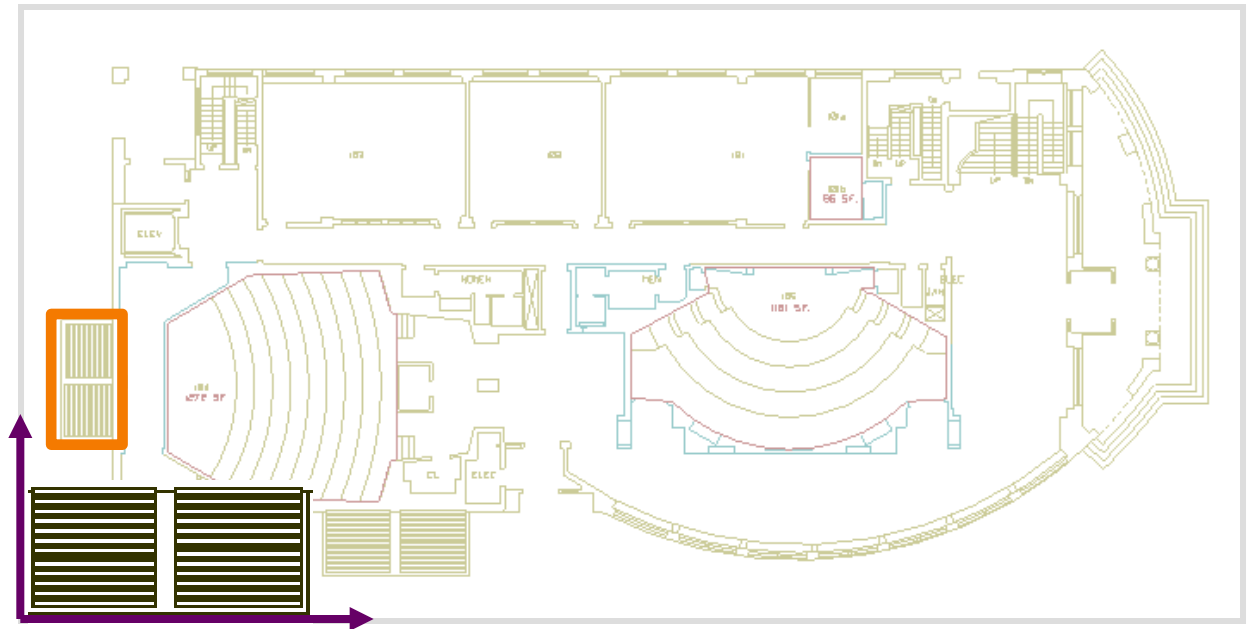
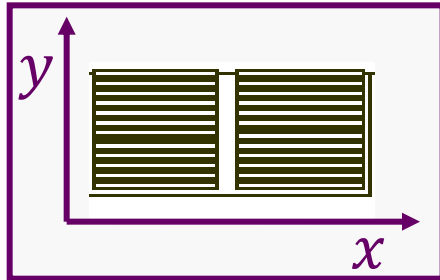
# 2D Modeling Transformations



# 2D Modeling Transformations



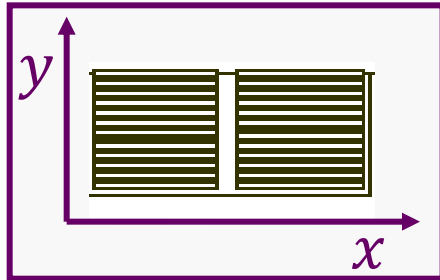
Object  
Coordinates



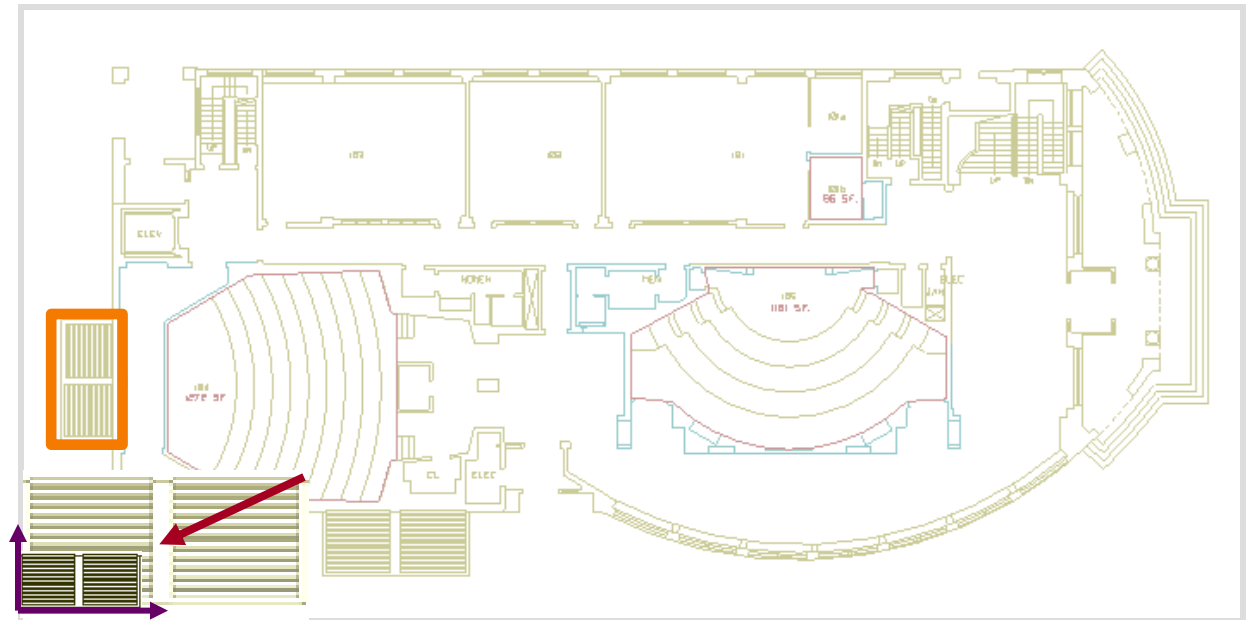


# 2D Modeling Transformations

Object  
Coordinates



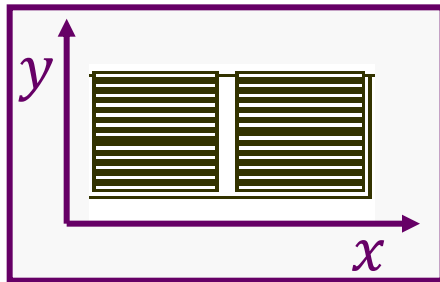
Scale .3, .3



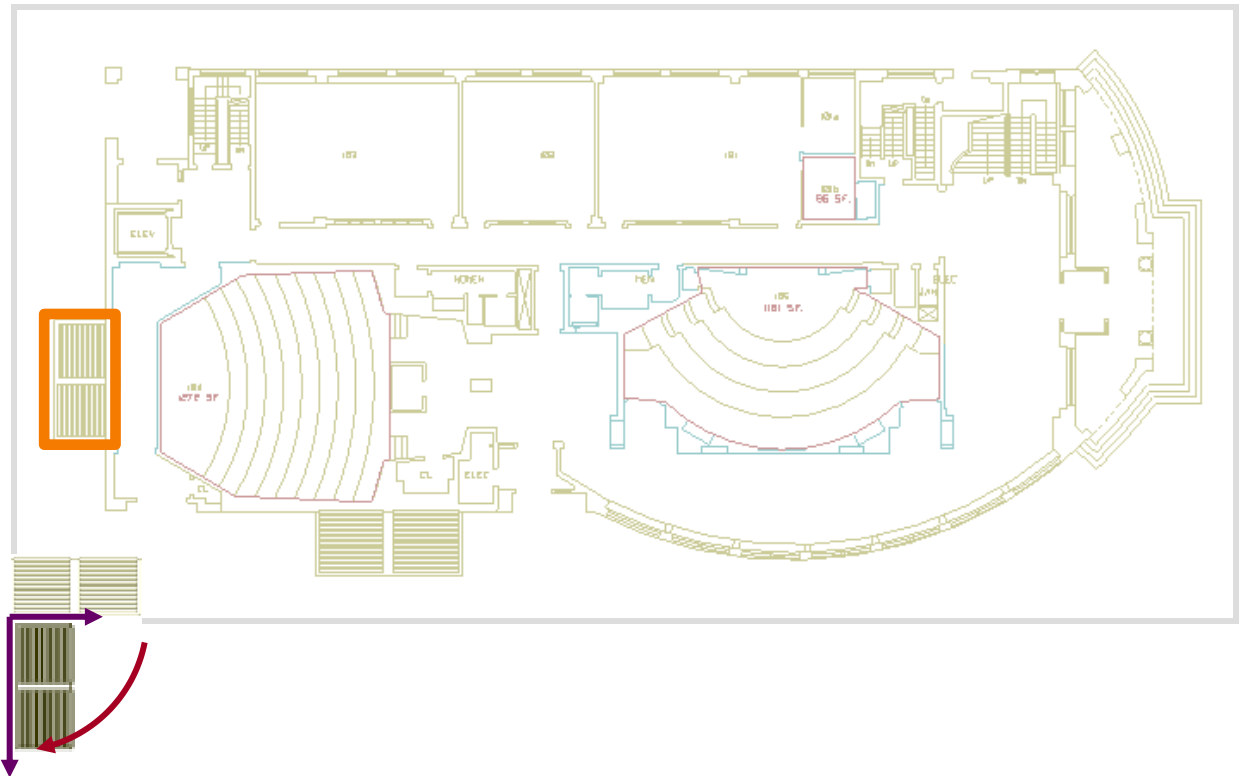
# 2D Modeling Transformations



Object  
Coordinates



Scale .3, .3  
Rotate -90

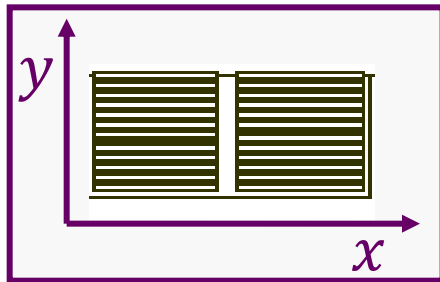




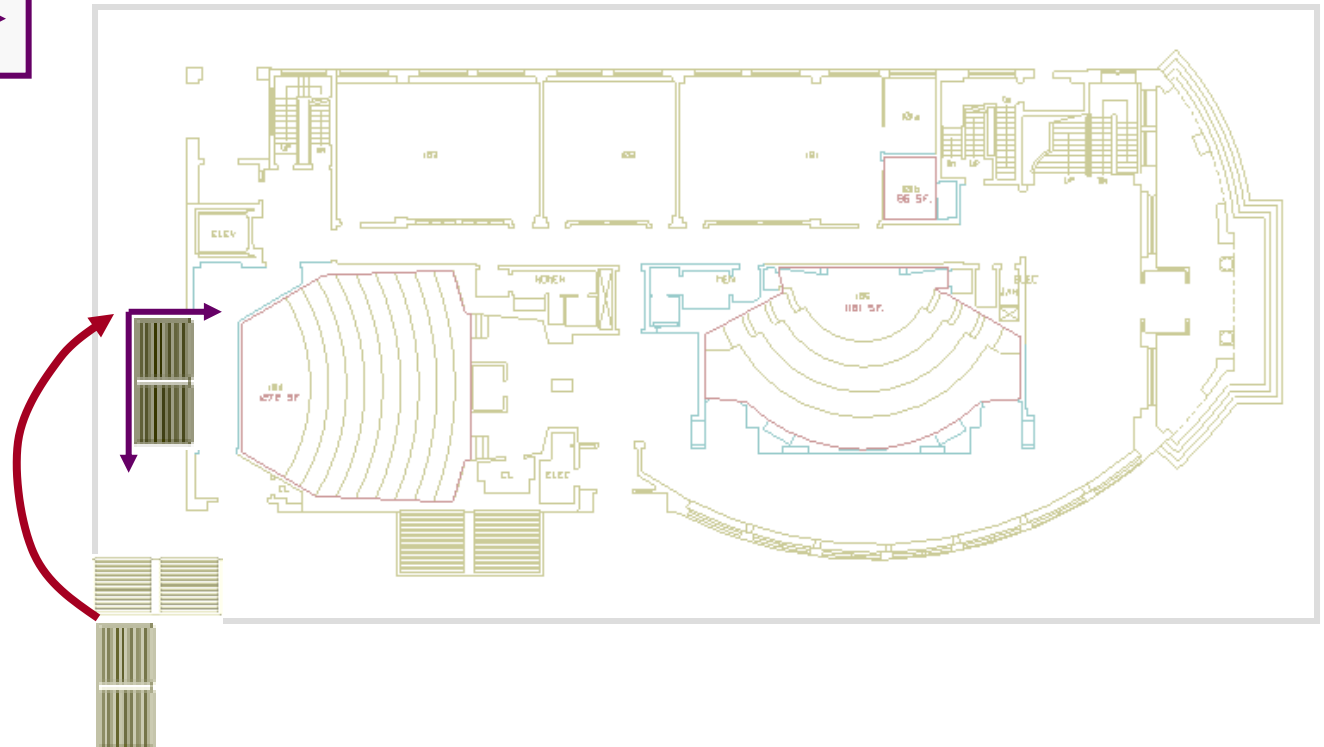


# 2D Modeling Transformations

Object  
Coordinates



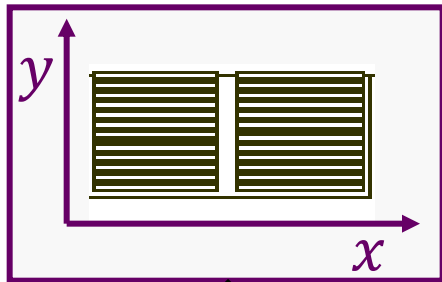
Scale .3, .3  
Rotate -90  
Translate 3, 5





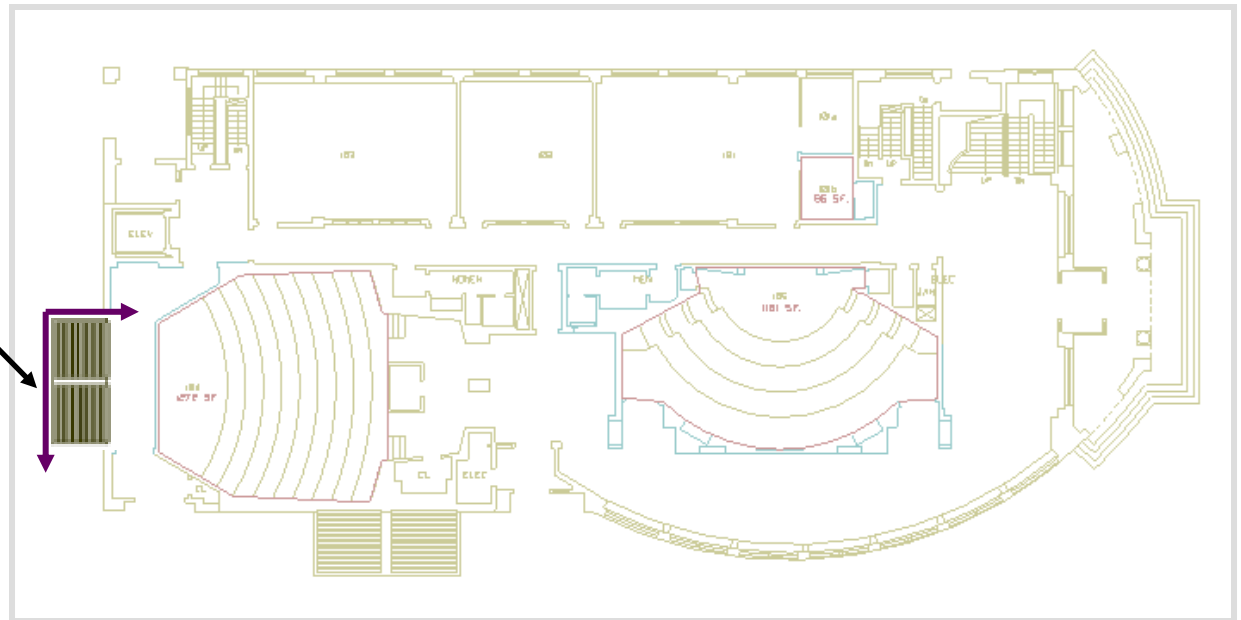
# 2D Modeling Transformations

Object  
Coordinates



The composition take us from  
object to world coordinates

Scale .3, .3  
Rotate -90  
Translate 3, 5





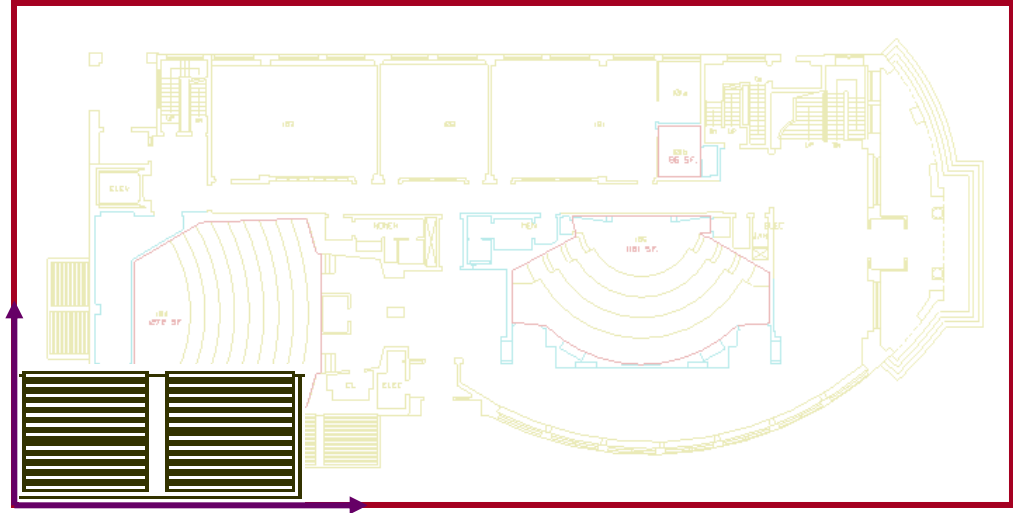
# Composing 2D Transformations

- Translation:

- $x' = x + t_x$
  - $y' = y + t_y$

- Scale:

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



- Rotation:

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



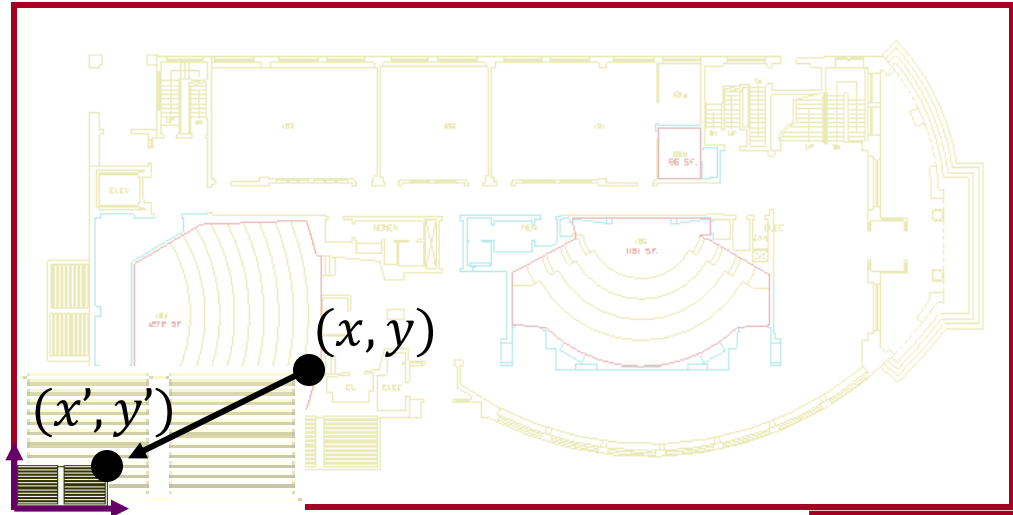
# Composing 2D Transformations

- Translation:

- $x' = x + t_x$
  - $y' = y + t_y$

- Scale:

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



- Rotation:

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

$$\begin{aligned} x' &= x \cdot s_x \\ y' &= y \cdot s_y \end{aligned}$$



# Composing 2D Transformations

- Translation:

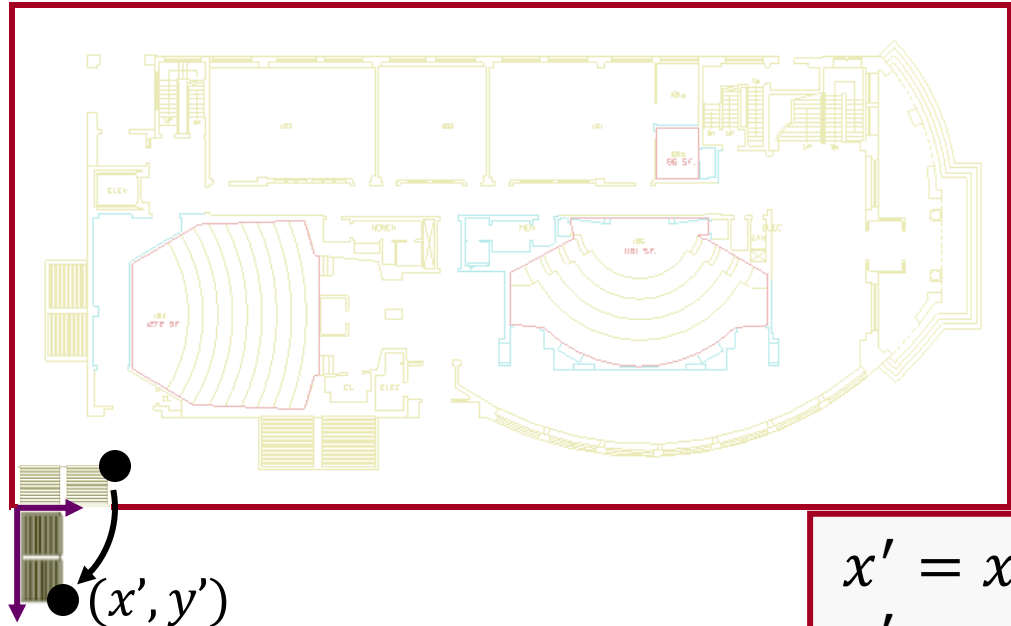
- $x' = x + t_x$
  - $y' = y + t_y$

- Scale:

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

- Rotation:

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



$$\begin{aligned} x' &= x \cdot s_x \\ y' &= y \cdot s_y \end{aligned}$$

$$\begin{aligned} x' &= (x \cdot s_x) \cdot \cos \theta - (y \cdot s_y) \cdot \sin \theta \\ y' &= (x \cdot s_x) \cdot \sin \theta + (y \cdot s_y) \cdot \cos \theta \end{aligned}$$



# Composing 2D Transformations

- Translation:

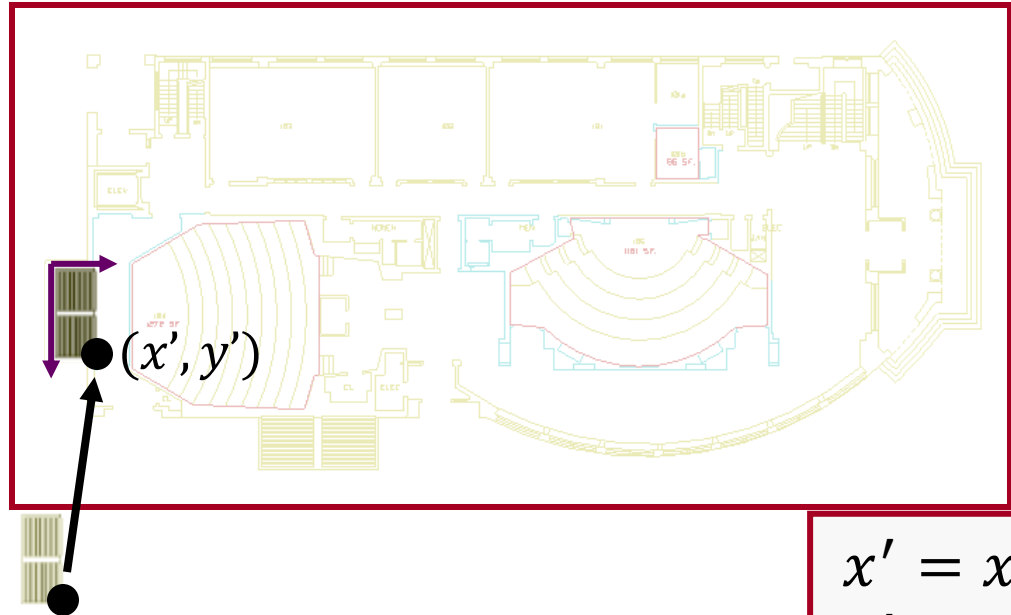
- $x' = x + t_x$
  - $y' = y + t_y$

- Scale:

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

- Rotation:

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



$$\begin{aligned}x' &= x \cdot s_x \\y' &= y \cdot s_y\end{aligned}$$

$$\begin{aligned}x' &= (x \cdot s_x) \cdot \cos \theta - (y \cdot s_y) \cdot \sin \theta \\y' &= (x \cdot s_x) \cdot \sin \theta + (y \cdot s_y) \cdot \cos \theta\end{aligned}$$

$$\begin{aligned}x' &= (x \cdot s_x) \cdot \cos \theta - (y \cdot s_y) \cdot \sin \theta + t_x \\y' &= (x \cdot s_x) \cdot \sin \theta + (y \cdot s_y) \cdot \cos \theta + t_y\end{aligned}$$



# Composing 2D Transformations

- Translation: Naïve composition makes the expression more complicated as more transformations are applied!

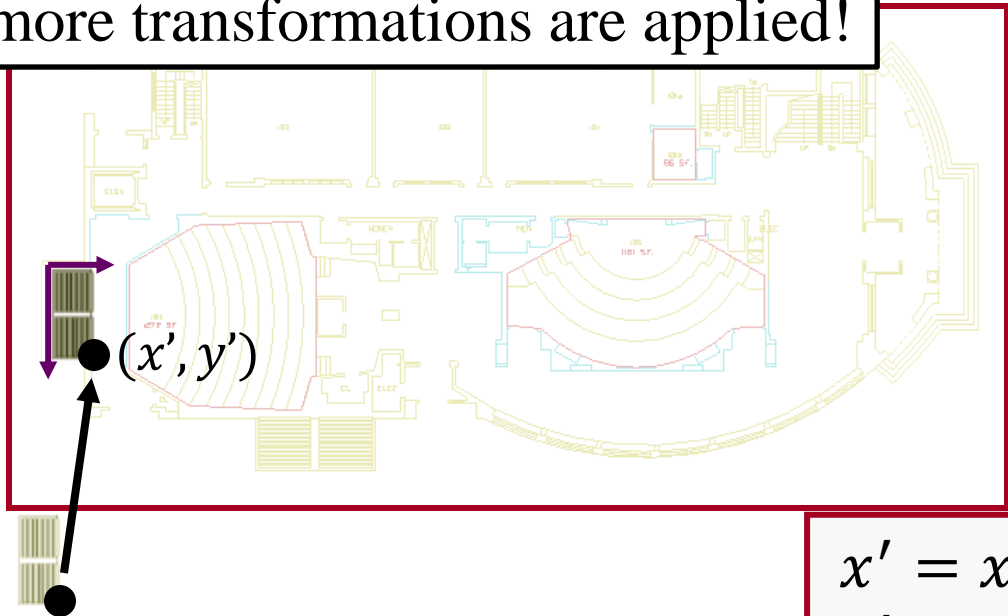
- $x' = x + t_x$
  - $y' = y + t_y$

- Scale:

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

- Rotation:

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



$$\begin{aligned} x' &= x \cdot s_x \\ y' &= y \cdot s_y \end{aligned}$$

$$\begin{aligned} x' &= (x \cdot s_x) \cdot \cos \theta - (y \cdot s_y) \cdot \sin \theta \\ y' &= (x \cdot s_x) \cdot \sin \theta + (y \cdot s_y) \cdot \cos \theta \end{aligned}$$

$$\begin{aligned} x' &= (x \cdot s_x) \cdot \cos \theta - (y \cdot s_y) \cdot \sin \theta + t_x \\ y' &= (x \cdot s_x) \cdot \sin \theta + (y \cdot s_y) \cdot \cos \theta + t_y \end{aligned}$$



# Overview

- 2D Transformations
  - Basic 2D transformations
  - Matrix representation
  - Matrix composition
- 3D Transformations
  - Basic 3D transformations
  - Same as 2D





# Matrix Representation

- Represent 2D transformation by a matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Multiply matrix by column vector



apply transformation to point

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \Leftrightarrow \begin{aligned} x' &= a \cdot x + b \cdot y \\ y' &= c \cdot x + d \cdot y \end{aligned}$$



# Matrix Representation

- Transformation composition amounts to matrix multiplication:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix}$$

⇒ The composition is still a matrix



# 2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

## 2D Scale around (0,0)?

$$\begin{aligned}x' &= s_x \cdot x \\ y' &= s_y \cdot y\end{aligned}$$

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

## 2D Rotate around (0,0)?

$$\begin{aligned}x' &= \cos \theta \cdot x - \sin \theta \cdot y \\ y' &= \sin \theta \cdot x + \cos \theta \cdot y\end{aligned}$$

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

## 2D Mirror over Y axis?

$$\begin{aligned}x' &= -x \\ y' &= y\end{aligned}$$

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



# 2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

## 2D Scale around (0,0)?

$$x' = s_x \cdot x$$

$$y' = s_y \cdot y$$

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

Like scale with  
negative scale values

## 2D Mirror over Y axis?

$$x' = -x$$

$$y' = y$$

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



# 2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

## 2D Translation?

$$x' = x + t_x$$

$$y' = y + t_y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



# 2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

## 2D Translation?

$$x' = x + t_x$$

$$y' = y + t_y$$

NO!

Only linear 2D transformations  
can be represented with a  $2 \times 2$  matrix



# Linear Transformations

- Linear transformations are combinations of ...
  - Scale, and
  - Rotation
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
- Properties of linear transformations:
  - Satisfies:  $T(s_1 \cdot p_1 + s_2 \cdot p_2) = s_1 \cdot T(p_1) + s_2 \cdot T(p_2)$ 
    - ⇒ Origin maps to origin
    - ⇒ Lines map to lines
    - ⇒ Preserves (weighted) average
    - ⇒ Parallel lines remain parallel
    - ⇒ Closed under composition



# Linear Transformations

- Linear transformations are combinations of ...

- Scale, and
- Rotation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Properties of linear transformations:

- Satisfies:  $T(s_1 \cdot p_1 + s_2 \cdot p_2) = s_1 \cdot T(p_1) + s_2 \cdot T(p_2)$

⇒ Origin maps to origin

⇒ Lines map to lines

⇒ Preserves (weighted) average

⇒ Parallel lines remain parallel

⇒ Closed under composition

Translations do not map the origin to the origin





# 2D Translation

- Treat 2D positions as 3D positions by adding a third, *homogenous*, coordinate with a “1”:

$$(x, y) \rightarrow (x, y, 1)$$

- Represent translations using a 3x3 matrix:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow \begin{aligned} x' &= x + t_x \\ y' &= y + t_y \end{aligned}$$



# Basic 2D Transformations

- Basic 2D transformations as  $3 \times 3$  matrices

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

Translate

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

Scale

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

Rotate

# Homog. Coordinates: $(x, y, 1) \leftrightarrow (x, y)$



More generally:

- For  $w \neq 0$  we associate  $(x, y, w) \leftrightarrow \left(\frac{x}{w}, \frac{y}{w}\right)$

What about when  $w = 0$ ?

Consider the limit:

$$\lim_{w \rightarrow \infty} (x, y, w) \leftrightarrow \lim_{w \rightarrow \infty} \left(\frac{x}{w}, \frac{y}{w}\right)$$

$\Rightarrow$  In the limit this is the *ideal* point at infinity in direction  $(x, y)$

# Homog. Coordinates: $(x, y, 1) \leftrightarrow (x, y)$



More generally:

- For  $w \neq 0$  we associate  $(x, y, w) \leftrightarrow \left(\frac{x}{w}, \frac{y}{w}\right)$
- We associate  $\lim_{t \rightarrow \infty} (tx, ty) \leftrightarrow (x, y, 0)$
- $(0, 0, 0)$  is not allowed

$\Rightarrow$  In addition to supporting translation, homogenous coordinates provide a way for describing geometry at infinity.

# Homog. Coordinates: $(x, y, 1) \leftrightarrow (x, y)$



More generally:

- For  $w \neq 0$  we associate  $(x, y, w) \leftrightarrow \left(\frac{x}{w}, \frac{y}{w}\right)$
- We associate  $\lim_{t \rightarrow \infty} (tx, ty) \leftrightarrow (x, y, 0)$
- $(0, 0, 0)$  is not allowed

## Note:

As defined, the points  $(a, b, 0)$  and  $(-a, -b, 0)$  represent the same point at infinity.

## Warning:

OpenGL distinguishes these as the two end-points of the line with equation  $bx = ay$ , allowing for representation of directional light sources.



# Affine Transformations

- Affine transformations are combinations of ...
  - Linear transformations, and
  - Translations

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Properties of affine transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Preserves (weighted) average
  - Parallel lines remain parallel
  - Closed under composition



# Affine Transformations

- Affine transformations are combinations of ...
  - Linear transformations, and
  - Translations

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Properties of affine transformations
  - **Original** Note that with affine transformations  $(x, y, 1)$  has to map to  $(x', y', 1)$
  - Lines map to lines
  - Preserves (weighted) average
  - Parallel lines remain parallel
  - Closed under composition



# Projective Transformations

- Projective transformations ...
  - Affine transformations, and
  - Projective warps

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Properties of projective transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - (Weighted) average is not necessarily preserved
  - Parallel lines do not necessarily remain parallel
  - Closed under composition





# Projective Transformations

- Projective transformations ...
  - Affine transformations, and
  - Projective warps

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Projective Note that under projective transformations
  - **Origin**  $(x, y, 1)$  does **not** have to map to  $(x', y', 1)$
  - Lines map to lines
  - (Weighted) average is not necessarily preserved
  - Parallel lines do not necessarily remain parallel
  - Closed under composition



# Overview

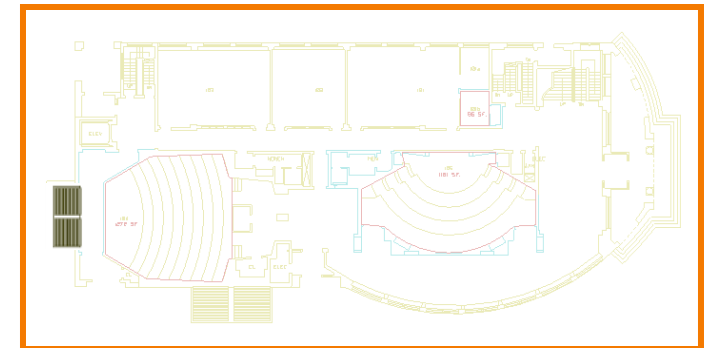
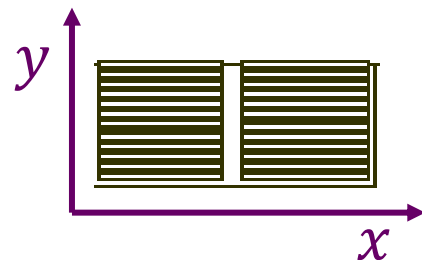
- 2D Transformations
  - Basic 2D transformations
  - Matrix representation
  - Matrix composition
- 3D Transformations
  - Basic 3D transformations
  - Same as 2D



# Matrix Composition

- Transformations can be combined by matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{pmatrix} \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
$$p' = T(t_x, t_y) \circ R(\theta) \circ S(s_x, s_y) \quad p$$

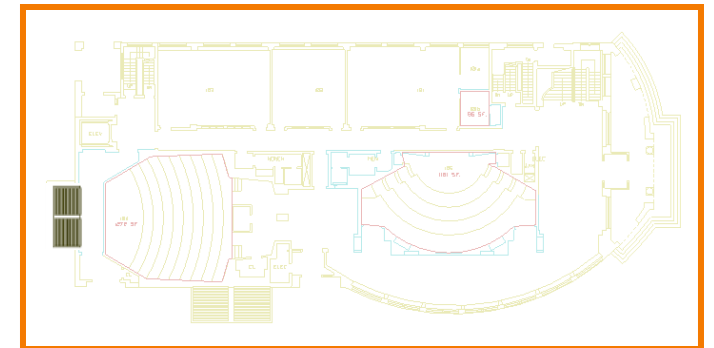
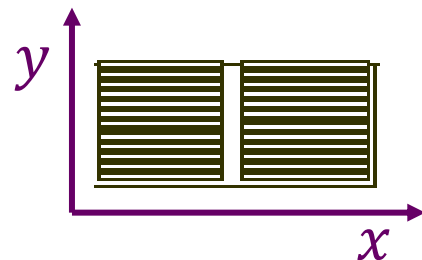




# Matrix Composition

- Transformations can be combined by matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left( \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} s_x \cdot \cos \theta & -s_y \cdot \sin \theta & t_x \\ s_x \cdot \sin \theta & s_y \cdot \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$







# Matrix Composition

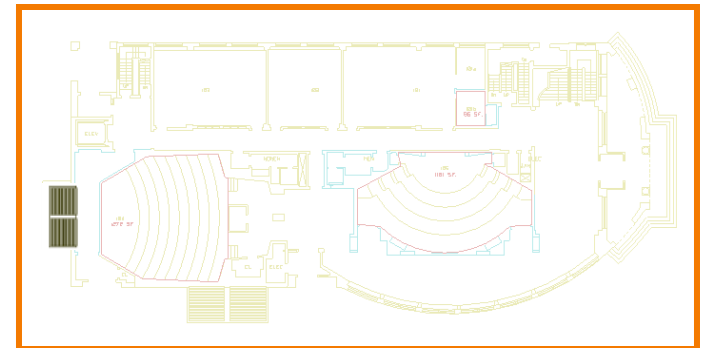
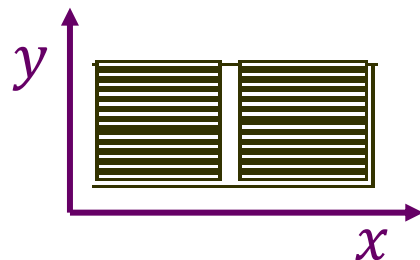
- [NOTE] order of transformations matters
  - » Matrix multiplication is not commutative

$$p' = T \cdot R \cdot S \cdot p$$



“Global”

“Local”





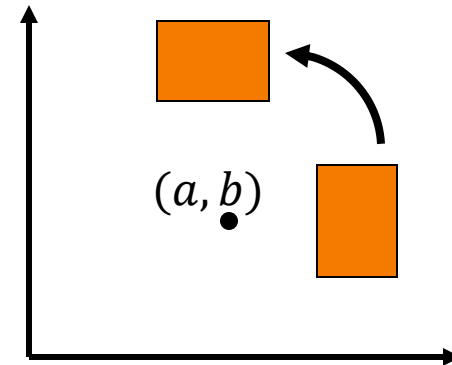
# Matrix Composition

- Rotate by  $\theta$  around arbitrary point  $(a, b)$

- $M = T(a, b) \circ R(\theta) \circ T(-a, -b)$

The trick:

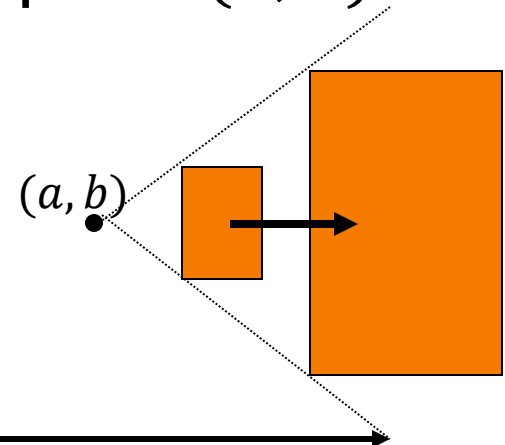
1. Translate  $(a, b)$  to the origin.
2. Do the rotation about origin.
3. Translate back.



- Scale by  $(s_x, s_y)$  around arbitrary point  $(a, b)$

- $M = T(a, b) \circ S(s_x, s_y) \circ T(-a, -b)$

(Use the same trick.)





# Overview

- 2D Transformations
  - Basic 2D transformations
  - Matrix representation
  - Matrix composition
- 3D Transformations
  - Basic 3D transformations
  - Same as 2D





# 3D Transformations

- Same idea as 2D transformations
  - Homogeneous coordinates:  $(x, y, z, w)$
  - $4 \times 4$  transformation matrices

» Affine

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

» Projective

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$



# Basic 3D Transformations

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

Identity

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

Scale

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Translation



# Basic 3D Transformations

## Pitch-Roll-Yaw Convention:

- Any rotation can be expressed as the combination of a rotation about the  $x$ -, the  $y$ -, and the  $z$ -axis.

Rotate around  $z$  axis:

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

Rotate around  $y$  axis:

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

Rotate around  $x$  axis:

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



# Basic 3D Transformations

## Pitch-Roll-Yaw Convention:

- Any rotation can be expressed as the combination of a rotation about the  $x$ -, the  $y$ -, and the  $z$ -axis.

Rotate around  $z$  axis:

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

Rotate around

How do you rotate around an arbitrary axis  $U$  by angle  $\psi$ ?

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

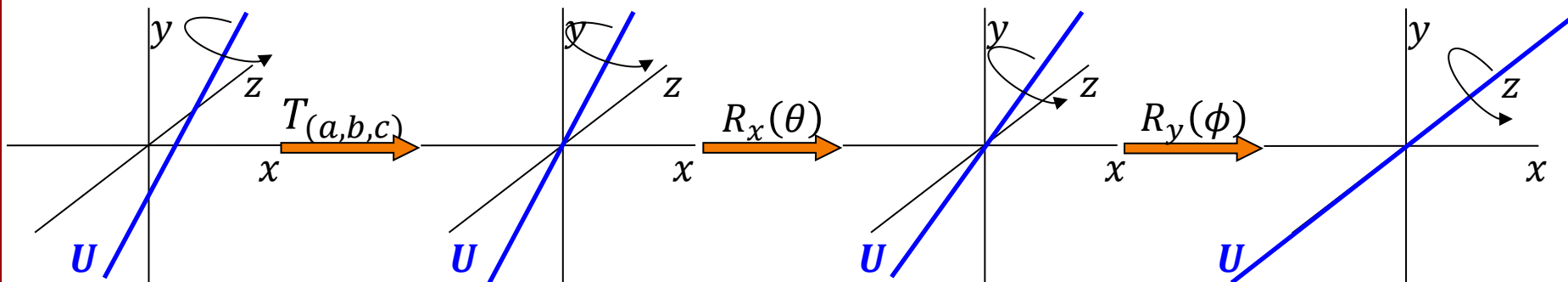
Rotate around  $x$  axis:

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



# Rotation By $\psi$ Around Arbitrary Axis $U$

- Align  $U$  (w.l.o.g.) with the  $z$ -axis:
  - $T_{(a,b,c)}$ : Translate  $U$  by  $(a, b, c)$  to pass through origin
  - $R_x(\theta)$ : Rotate about the  $x$ -axis by  $\theta$  to get  $U$  in the  $xz$ -plane
  - $R_y(\phi)$ : Rotate about the  $y$ -axis by  $\phi$  to align  $U$  with the  $z$ -axis
- $R_z(\psi)$ : Perform rotation by  $\psi$  around the  $z$ -axis.
- Do inverse of original transformation for alignment.





# Rotation By $\psi$ Around Arbitrary Axis $U$

- Align  $U$  (w.l.o.g.) with the  $z$ -axis:
  - $T_{(a,b,c)}$ : Translate  $U$  by  $(a, b, c)$  to pass through origin
  - $R_x(\theta)$ : Rotate about the  $x$ -axis by  $\theta$  to get  $U$  in the  $xz$ -plane
  - $R_y(\phi)$ : Rotate about the  $y$ -axis by  $\phi$  to align  $U$  with the  $z$ -axis
- $R_z(\psi)$ : Perform rotation by  $\psi$  around the  $z$ -axis.
- Do inverse of original transformation for alignment.

$$p' = \underbrace{\left( R_y(\phi) \cdot R_x(\theta) \cdot T_{(a,b,c)} \right)^{-1}}_{\text{Aligning Transformation}} \cdot R_z(\psi) \cdot \underbrace{\left( R_y(\phi) \cdot R_x(\theta) \cdot T_{(a,b,c)} \right)}_{\text{Aligning Transformation}} p$$

Aligning Transformation