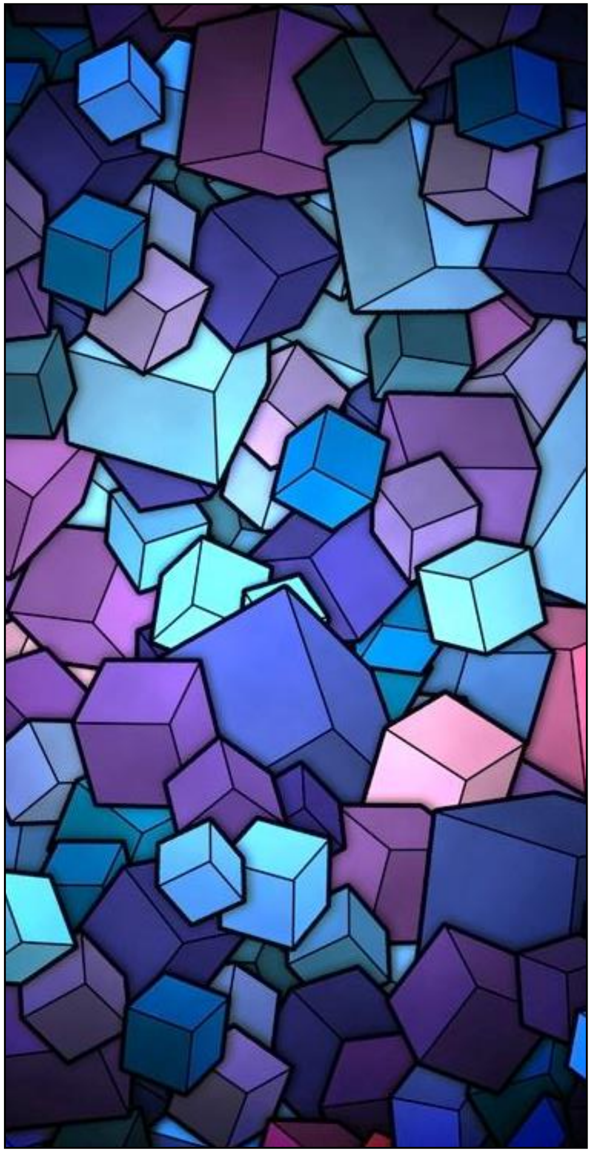
Transformations

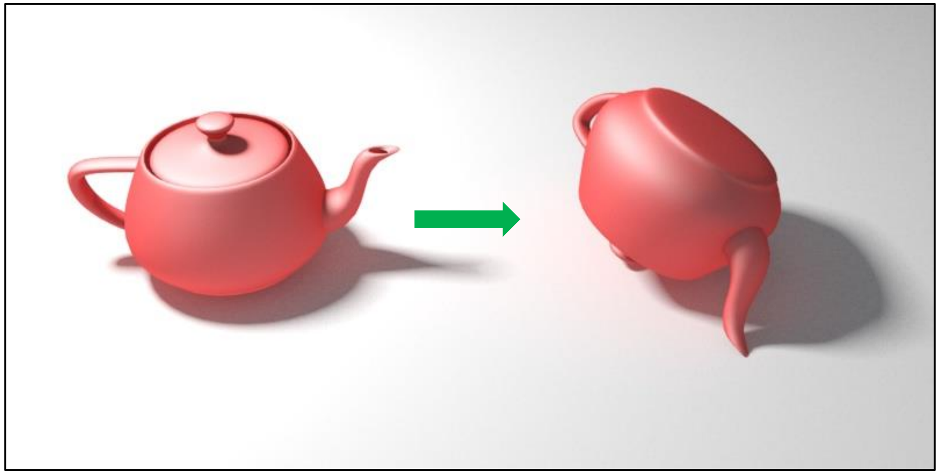


Georgios Papaioannou – 2016

ABOUT TRANSFORMATIONS

Transformations

- They are operators on vectors and points of a corresponding vector or affine space
- They alter the coordinates of shape vertices
- They are basic building blocks of geometric design:
  - Help us manipulate shapes to produce new ones
  - Help us express relations between coordinate systems in a virtual world



Affine Transformations

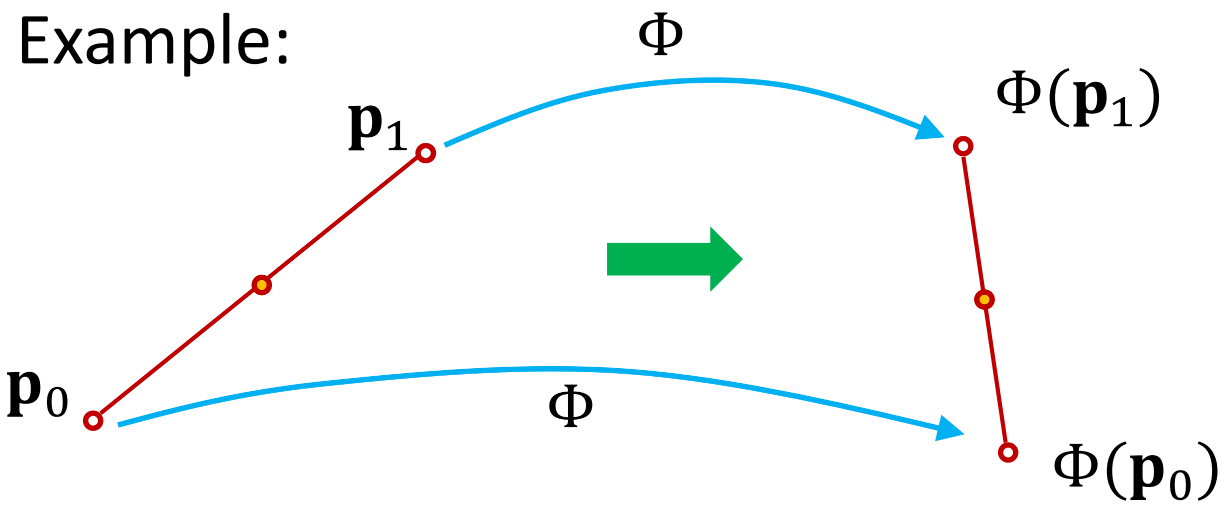
- An affine transformation Φ on an affine space is a transformation that preserve affine combinations

$$\mathbf{p} = \sum_{i=0}^{n} a_i \mathbf{p}_i \Longrightarrow \Phi(\mathbf{p}) = \sum_{i=0}^{n} a_i \Phi(\mathbf{p}_i)$$

- For shapes in $\mathbb{E}^2$ and $\mathbb{E}^3$ this is an important property:
- To transform a shape we only need to transform its defining vertices

Affine Transformations on Vertices

- Example:



- The midpoint of the transformed endpoints is the transformed midpoint
  – Similarly, all transformed points on the line segment can be linearly interpolated form the transformed endpoints

Affine Transformations in 2D and 3D

- Mappings of the form $\Phi(\mathbf{p}) = \mathbf{A} \cdot \mathbf{p} + \vec{\mathbf{t}}$ are affine transformations in $\mathbb{E}^2$ and $\mathbb{E}^3$
- 2D:
  - $\mathbf{A}$ is a 2X2 matrix and
  - $\vec{\mathbf{t}}$ is an offset vector in matric column form: $\vec{\mathbf{t}} = \left[t_x \; t_y\right]^T$
- 3D:
  - $\mathbf{A}$ is a 3X3 matrix and
  - $\vec{\mathbf{t}}$ is an offset vector in matric column form: $\vec{\mathbf{t}} = \left[t_x \; t_y \; t_z\right]^T$

Linear Transformations

- Linear transformations are affine transformations with the following properties:
  - Preserve additivity: $\Phi(\mathbf{p}+\mathbf{q}) = \Phi(\mathbf{p}) + \Phi(\mathbf{q})$
  - Preserve scalar multiplication: $\Phi(c\mathbf{p}) = \mathrm{c}\Phi(\mathbf{p})$

- Important:
  - The affine transformation $\Phi(\mathbf{p}) = \mathbf{A} \cdot \mathbf{p} + \vec{\mathbf{t}}$ is not linear (why?)
  - But the transformation $\Phi(\mathbf{p}) = \mathbf{A} \cdot \mathbf{p}$ is!

2D TRANSFORMATIONS

Geometric Transformations in 2D

- The 4 common transformations that are used in computer graphics are:
  - Translation $\mathrm{T}(\mathbf{p}) = \mathbf{I}\mathbf{p} + \vec{\mathbf{t}}$
  - Rotation $\mathrm{R}(\mathbf{p}) = \mathbf{R}_{\theta}\mathbf{p}$
  - Scaling $\mathrm{S}(\mathbf{p}) = \mathbf{S}_{sx,sy}\mathbf{p}$
  - Shearing $\mathrm{Sh}(\mathbf{p}) = \mathbf{Sh}_{sx,sy}\mathbf{p}$

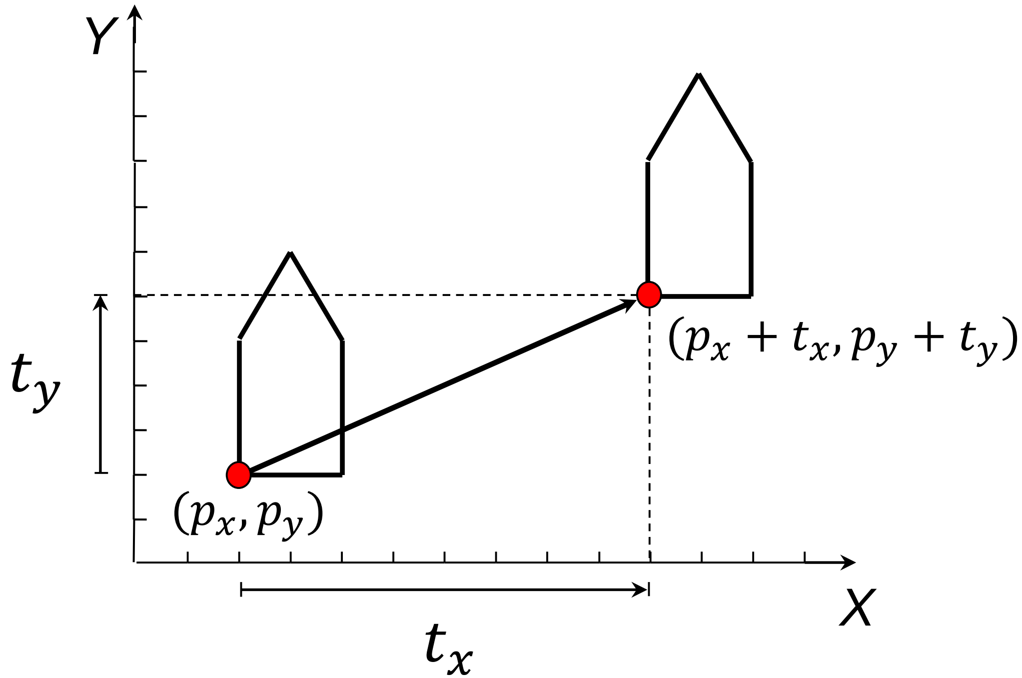
- All of the above transformations are invertible, i.e. given $\Phi(\mathbf{p})$, there always exists the inverse transformation $\Phi^{-1}(\mathbf{p})$:

$$\mathbf{p}' = \Phi(\mathbf{p}) \Leftrightarrow \mathbf{p} = \Phi^{-1}(\mathbf{p}')$$

2D Translation

- Moves a point on the plane

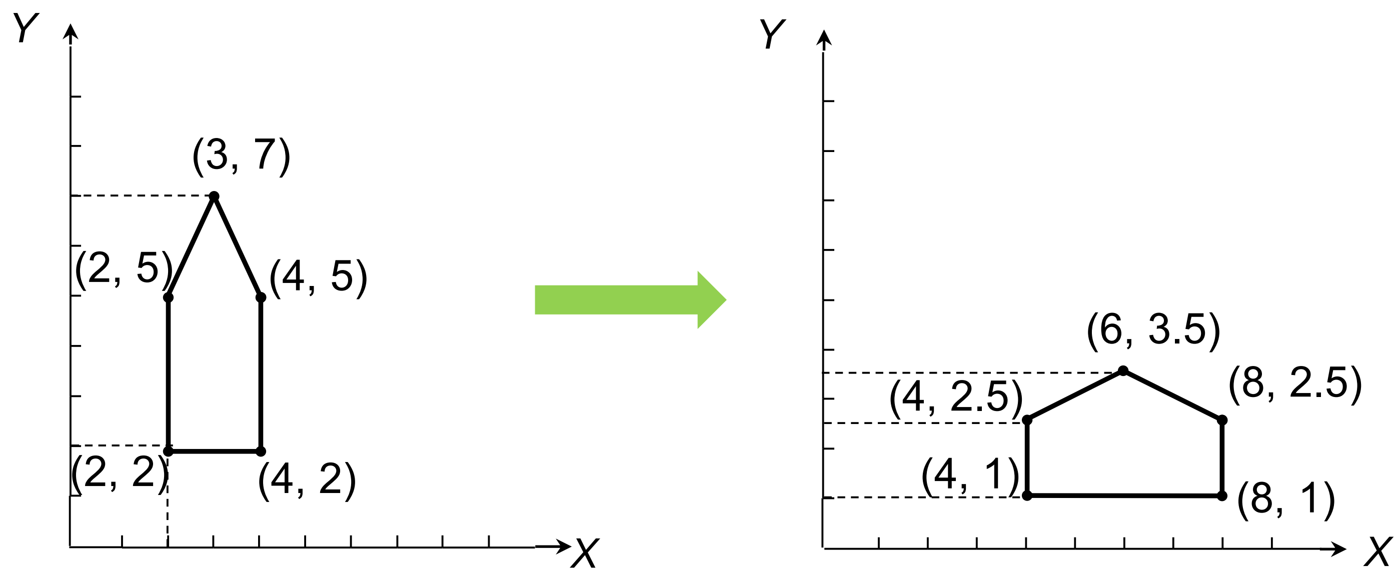
$$\boxed{\mathbf{p}' = \mathbf{I}\mathbf{p} + \vec{\mathbf{t}} = \mathbf{p} + \vec{\mathbf{t}}}$$



2D Scaling

$$\mathbf{p}' = \mathbf{S}_{s_x,s_y}\mathbf{p} \qquad \mathbf{S}_{s_x,s_y} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

- When $s_x = s_y$, then the scaling is isotropic (preserves angles)

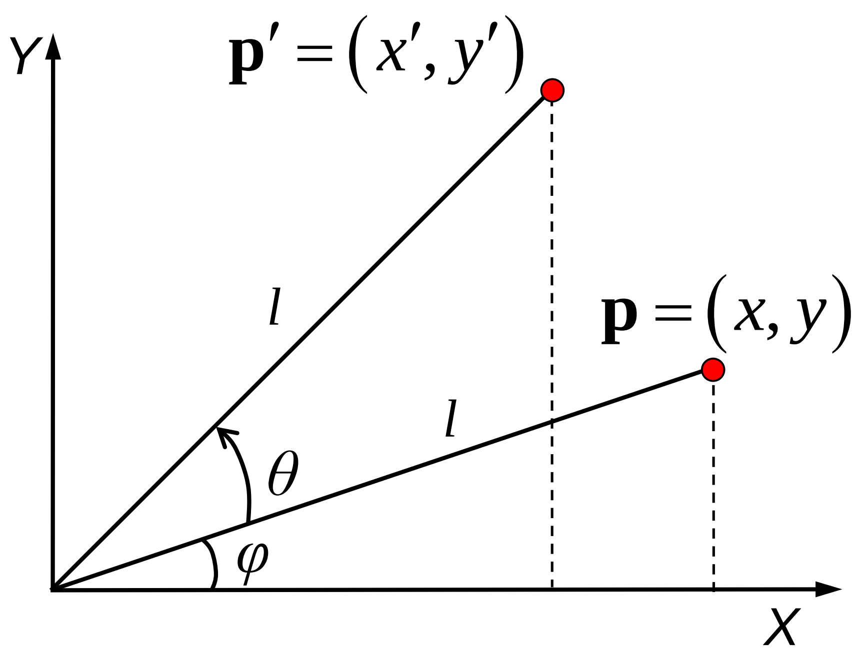


2D Rotation

- Rotates a point around the origin by angle θ

$$x' = l\cos(\varphi+\theta) = l(\cos\varphi\cos\theta - \sin\varphi\sin\theta) = x\cos\theta - y\sin\theta$$

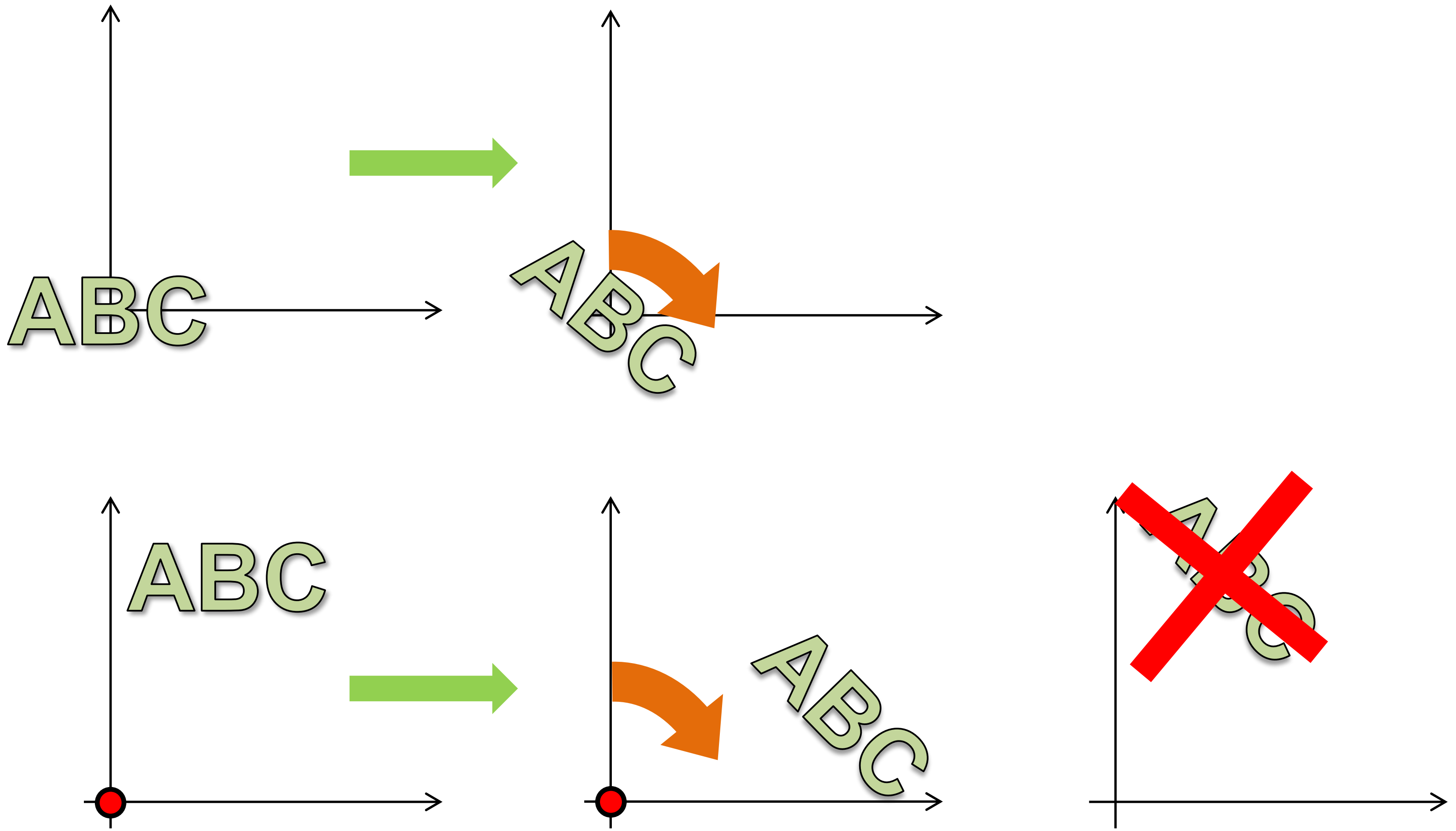
$$y' = l\sin(\varphi+\theta) = l(\cos\varphi\sin\theta + \sin\varphi\cos\theta) = x\sin\theta + y\cos\theta$$



$$\mathbf{p}' = \mathbf{R}_\theta \mathbf{p}$$

$$\mathbf{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

2D Rotation - Examples



Rotations are always relative to the coordinate system origin!

2D Shearing

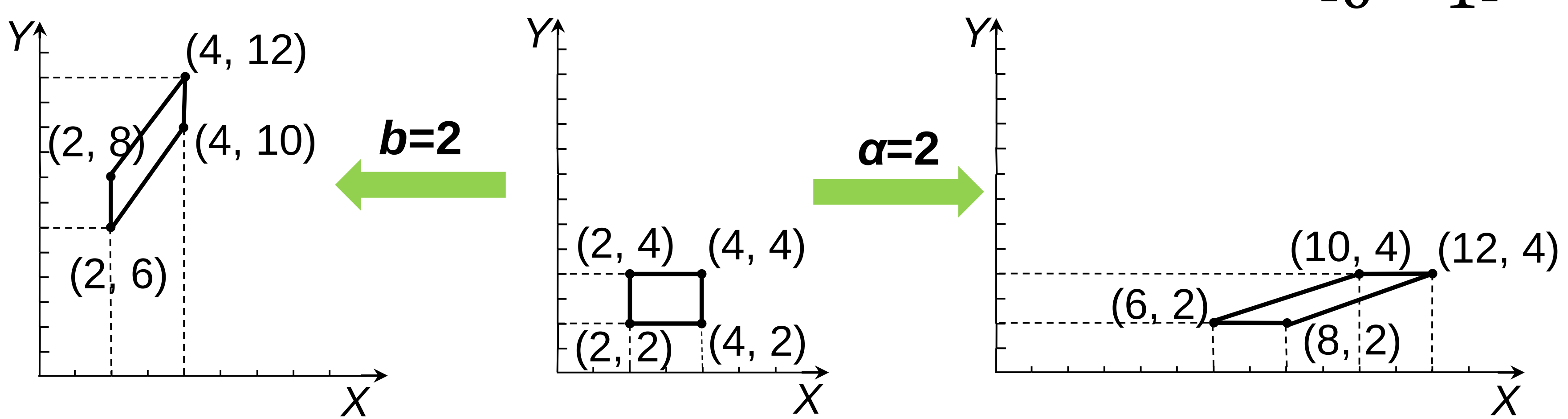
- Skews the shape by translating a point in one axis proportionally to its coordinate on the other axis

$$\mathbf{p}' = \mathbf{Sh}_{y,b}\mathbf{p}$$

$$\mathbf{Sh}_{y,b} = \begin{bmatrix} 1 & 0 \\ b & 1 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{Sh}_{x,a}\mathbf{p}$$

$$\mathbf{Sh}_{x,a} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}$$



Composite Transformations

- Useful transformations in computer graphics and visualization rarely consist of a single basic affine transformation
- Transformation composition is the stacking of operators (function composition):

$$\Phi \circ \Gamma(\mathbf{p}) = \Phi(\Gamma(\mathbf{p}))$$

- We can efficiently compute composite linear transformations

Composite Linear Transformations (1)

- A useful property of linear transformations is that a composite transformation can be expressed as matrix multiplication:

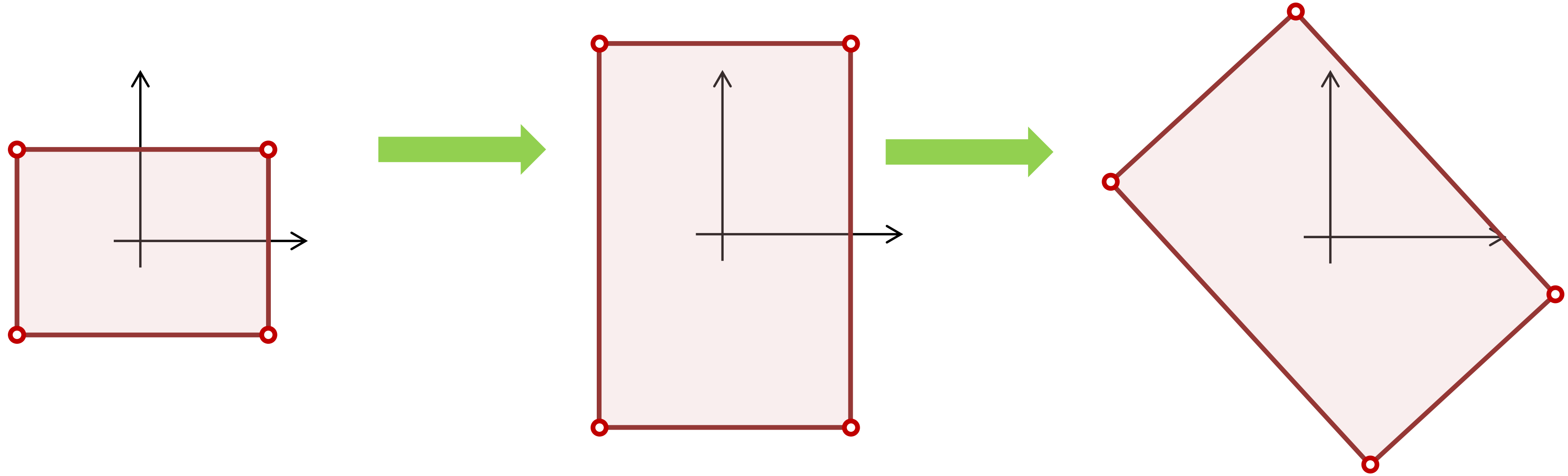
$$\boxed{\Phi \circ \Gamma(\mathbf{p}) = \mathbf{\Phi} \cdot \mathbf{\Gamma} \cdot \mathbf{p}}$$

- In graphics, it allows the efficient computation of multiple composite transformations

Composite Linear Transformations (2)

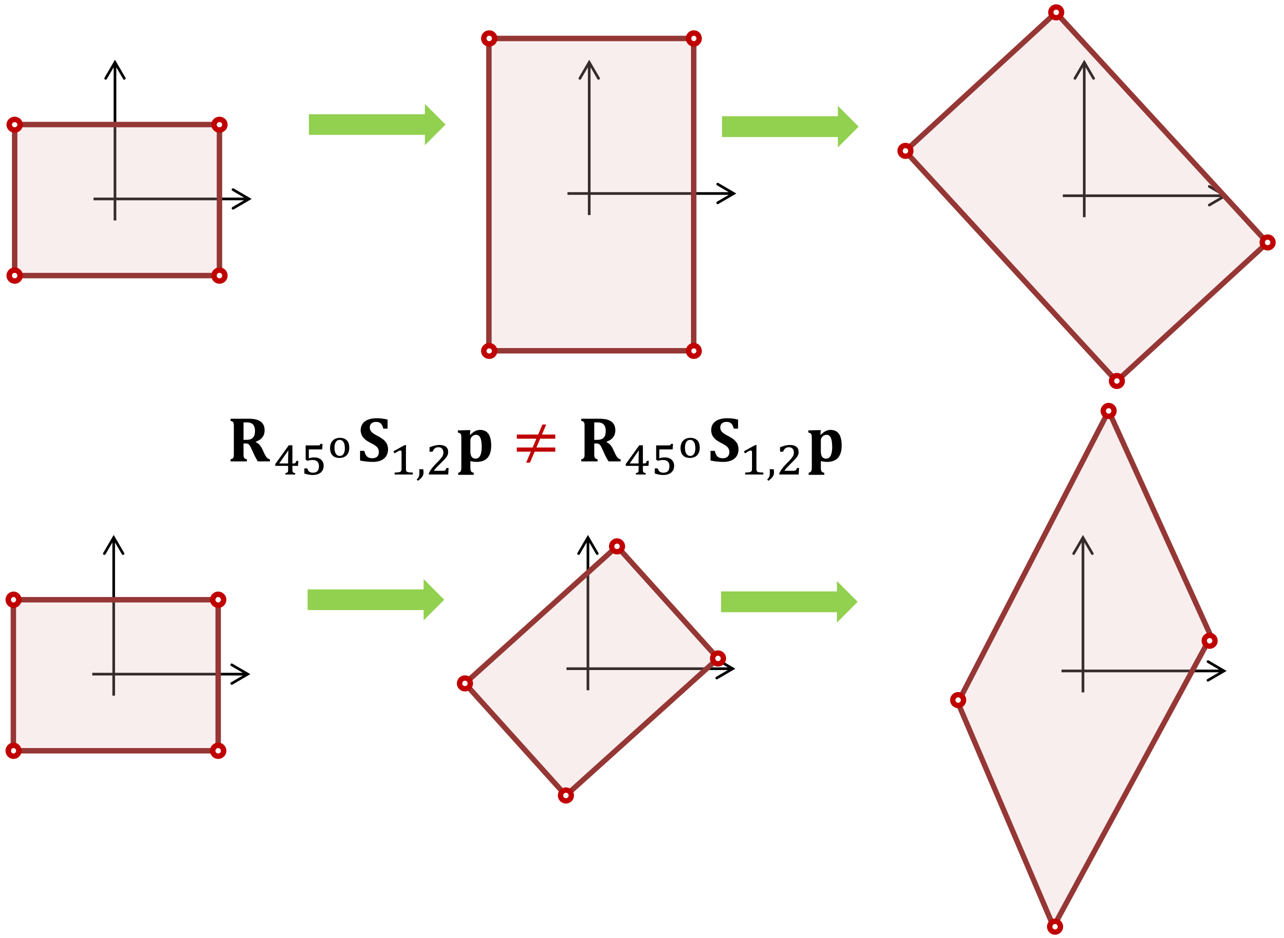
- Example:

$$R_{45^o}\left(S_{1,2}(\mathbf{p})\right) = \mathbf{R}_{45^o}\mathbf{S}_{1,2}\mathbf{p}$$



Composite Linear Transformations (3)

- Transformations are not commutative in general!



$$\mathbf{R}_{45^o}\mathbf{S}_{1,2}\mathbf{p} \neq \mathbf{R}_{45^o}\mathbf{S}_{1,2}\mathbf{p}$$

Composite Linear Transformations (4)

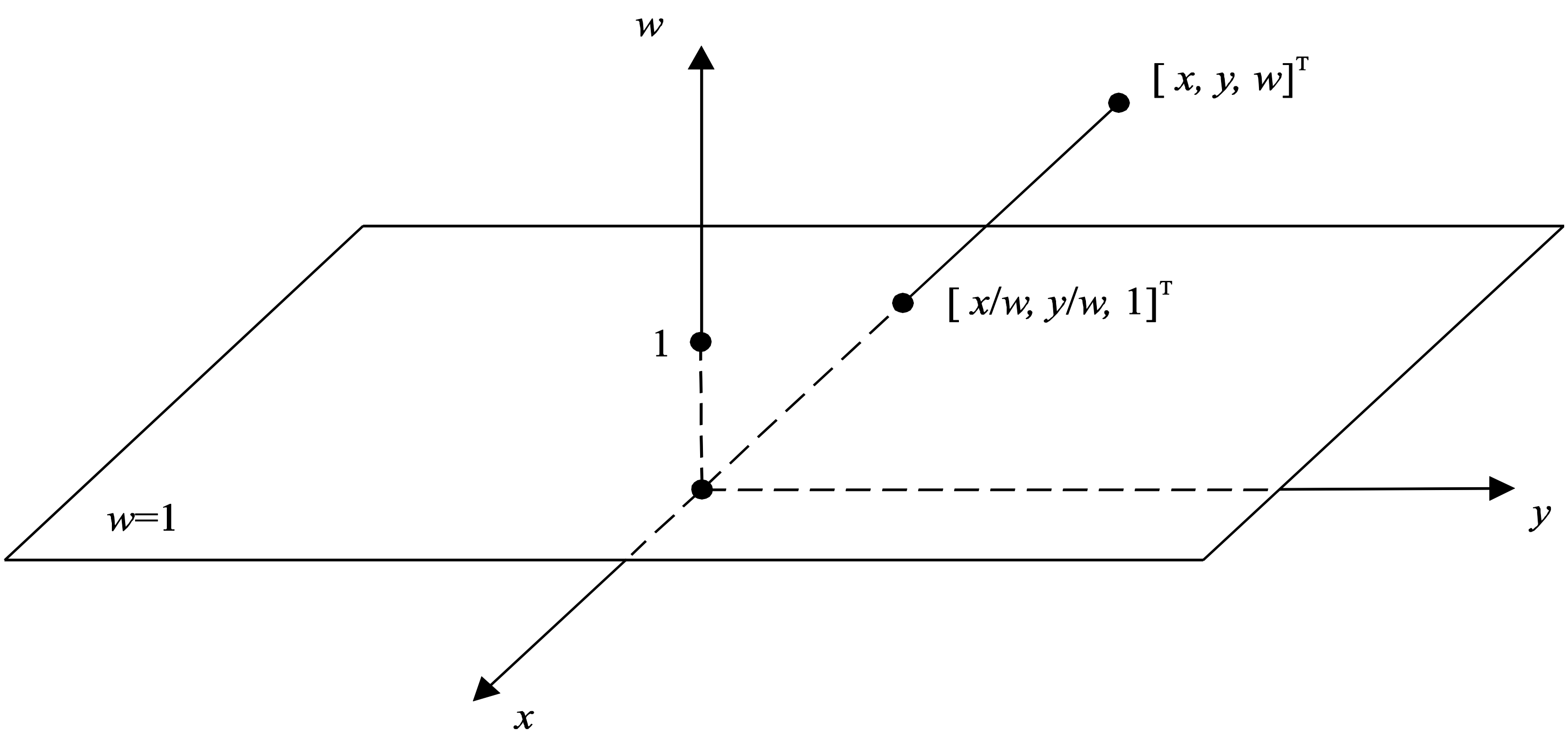
- Unfortunately, translation cannot be expressed as a linear transformation and is therefore impossible to express it as a matrix multiplication
- We must convert the transformation to a linear one

Homogeneous Coordinates (1)

- With homogeneous coordinates, we augment the dimensionality of the space by one
- So $[x, y]^T$ coordinates become $[x, y, w]^T$
- Similarly, all transformations are now expressed as 3X3 matrices
  - For w=1 we get the basic representation of a point: $[x, y, 1]^T$
  - All points which are multiples of each other are equivalent
  - Typically, we work with the basic representation of points

Homogeneous Coordinates (2)

- Therefore the 2D space becomes a plane (slice) embedded in 3D space at w=1



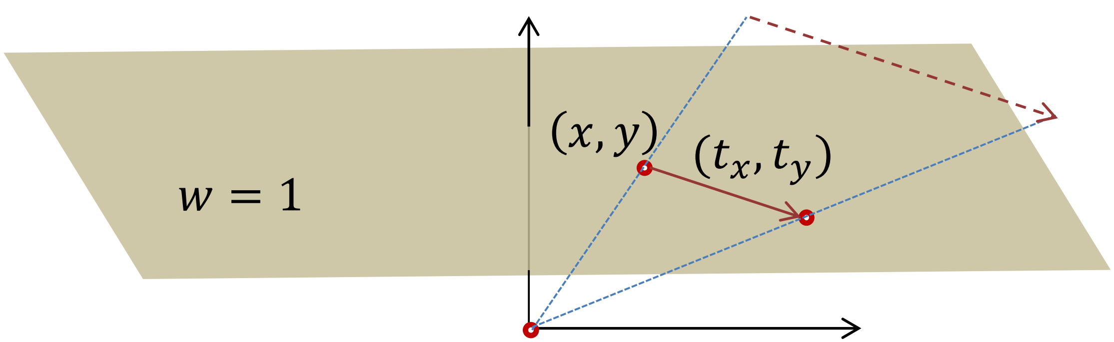
Homogeneous Coordinates (3)

- Points on the homogeneous 2D plane define an affine space and not a vector space
  - Adding two vectors results in a vector outside the plane (remember we also add the w coordinates!)
- The origin of our homogeneous coordinate system is typically (0,0,1) (or (0,0,w) in general)
- Since addition is not defined in our space, how is translation expressed?

Homogeneous Transformations (1)

- Translation in our augmented, homogeneous space can be expressed as a linear transformation:
  - (it is actually a skew (shearing) transformation of x,y w.r.t. w in 3D)

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \xRightarrow{w=1} (x', y') = (x, y) + (t_x, t_y)$$



Homogeneous Transformations (2)

- Now all geometric transformations can be performed by matrix multiplication alone!
- We can arbitrarily combine transformations in a unified manner

$$R_{\theta_1}\left(T_{\mathbf{v}_1}\left(R_{\theta_2}\left(T_{\mathbf{v}_2}\left(S_{s_1,s_2}(\dots(\mathbf{p})\dots)\right)\right)\right)\right) = \mathbf{R}_{\theta_1}\mathbf{T}_{\mathbf{v}_1}\mathbf{R}_{\theta_2}\mathbf{T}_{\mathbf{v}_2}\mathbf{S}_{s_1,s_2}\dots\mathbf{p}$$

Homogeneous 2D Transformations

- In matrix form (3X3) the homogeneous 2D transformations become:

$$\mathbf{T}_{t_x,t_y} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{Sh}_{x,a} = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{S}_{s_x,s_y} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{Sh}_{y,b} = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}_{\theta} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

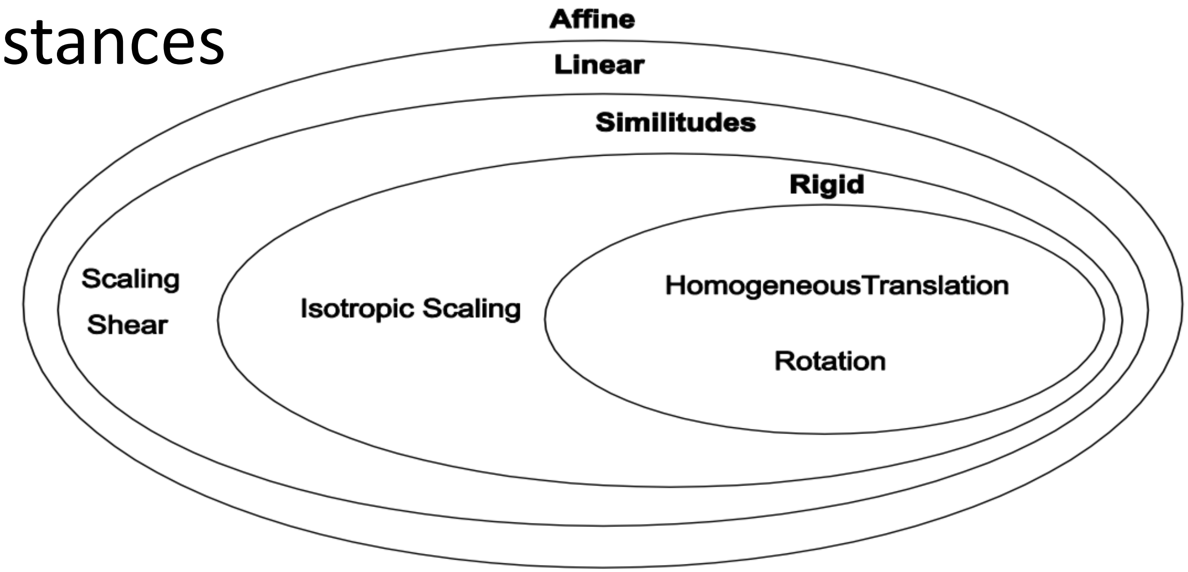
Representing Shape Transformations

- In the following, we will apply transformations to entire shapes
- This is equivalent to applying the transformations on each defining vertex of the shape
- Notation:

$$\mathbf{M}_1 \cdot \mathbf{M}_2 \cdot \underbrace{Rect}_{\text{shape}} = \mathbf{M}_1 \cdot \mathbf{M}_2 \cdot \underbrace{[\mathbf{v}_1\ \mathbf{v}_2\ \mathbf{v}_3\ \mathbf{v}_4]}_{\text{shape vertices}} = \mathbf{M}_1 \cdot \mathbf{M}_2 \cdot \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Categorizing the Transformations

- Affine: preserve linear combinations
- Linear: can be expressed by a concatenation of matrices, preserve parallel lines
- Similitudes: preserve ratios of distances and angles
- Rigid: preserve distances



Inverse Transformations

- The inverse of geometric transformation is the inverse matrix $\mathbf{M}^{-1}$ of the original transformation $\mathbf{M}$
- The inverse of a concatenated transformation is the concatenation of the inverse matrices in reverse order:

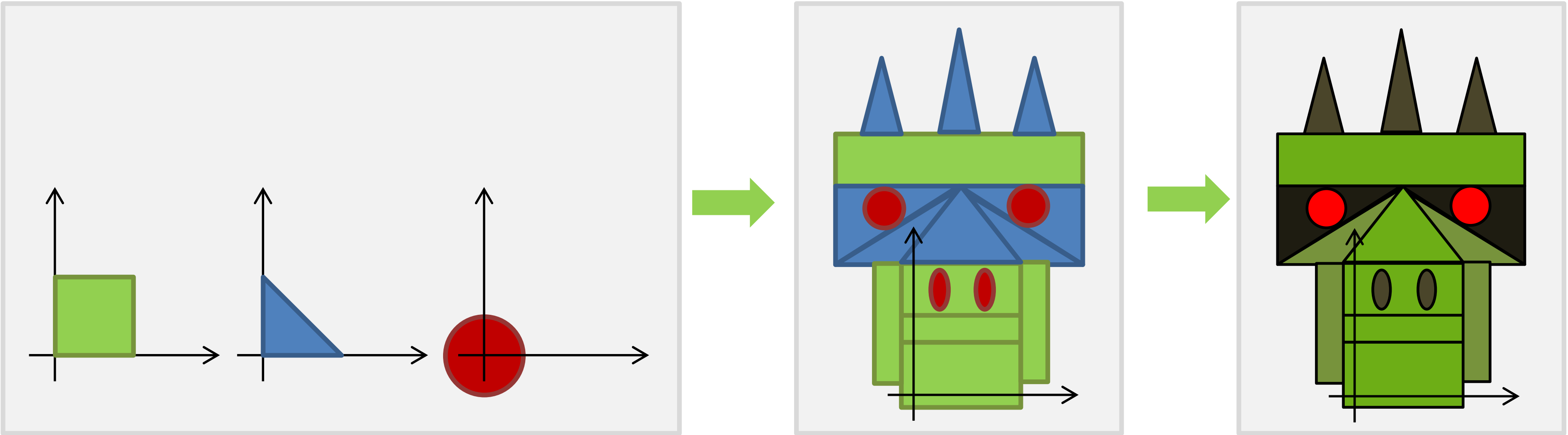
$$(\mathbf{M}_1\mathbf{M}_2\mathbf{M}_3 \ldots \mathbf{M}_n)^{-1} = \mathbf{M}_n^{-1} \ldots \mathbf{M}_3^{-1}\mathbf{M}_2^{-1}\mathbf{M}_1^{-1}$$

- Inverse of standard transformations:

$$\mathbf{R}_\theta^{-1} = \mathbf{R}_{-\theta} \qquad \mathbf{S}_{s_x,s_y}^{-1} = \mathbf{S}_{\frac{1}{s_x},\frac{1}{s_y}} \qquad \mathbf{T}_{t_x,t_y}^{-1} = \mathbf{T}_{-t_x,-t_y}$$

Shape Composition

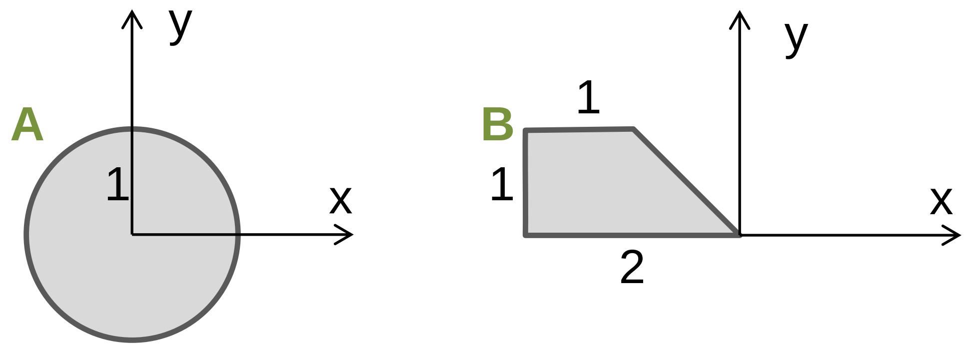
- We can use geometric transformations to create complex shapes from simple primitive ones
  – We use the transformations to modify instances of the original shapes and arrange them in their pose in the composite object



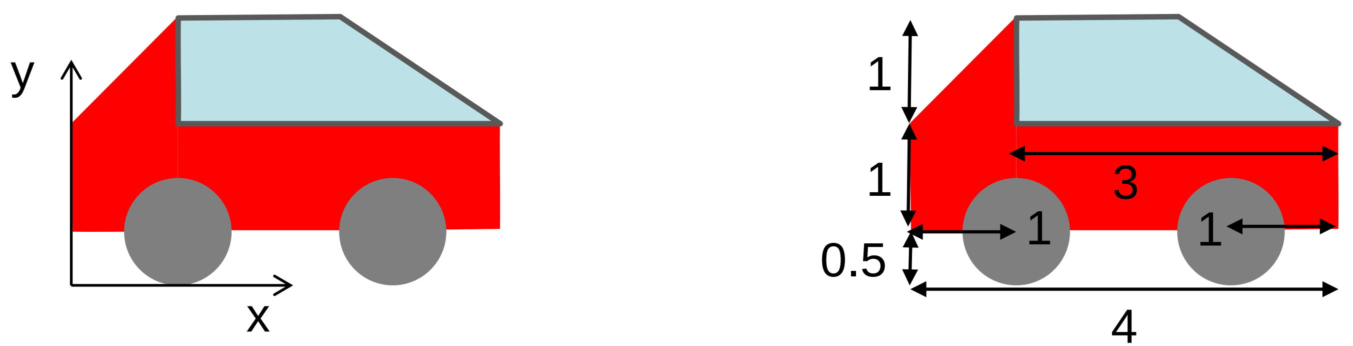
Original primitives (object space coords) Composite object Colored object

2D Transformation Example (1)

- Given the following primitives

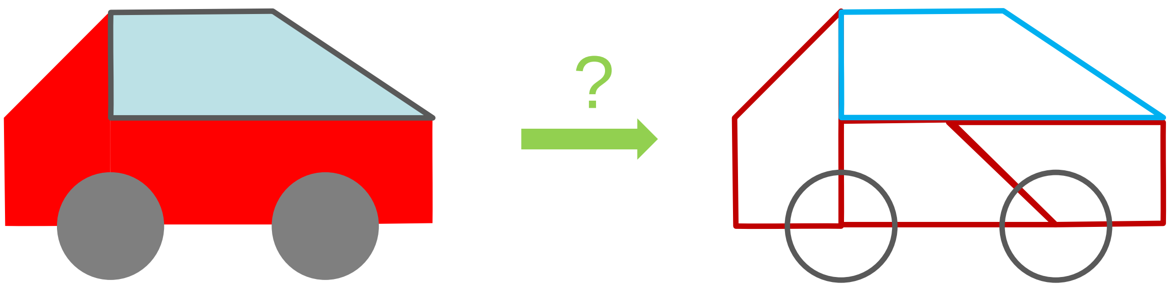


- Build this:



2D Transformation Example (2)

- First, let us try to decompose the target shape into primitives:

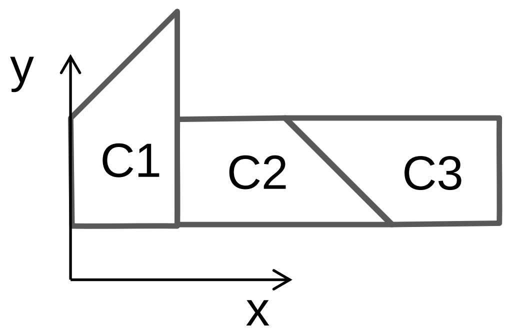


Three shape groups:
- Chassis
- Wheels
- Windshield

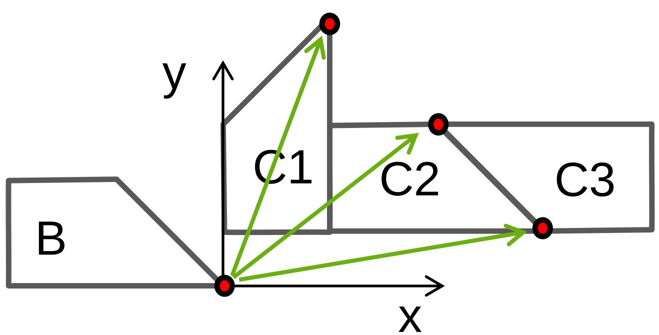
- Observe that in 2D we can order primitives in different layers to hide parts of the geometry
- Second, locate the origin of the resulting shape and compare it to the one of the primitives

2D Transformation Example (3)

- Chassis (3XB parts)

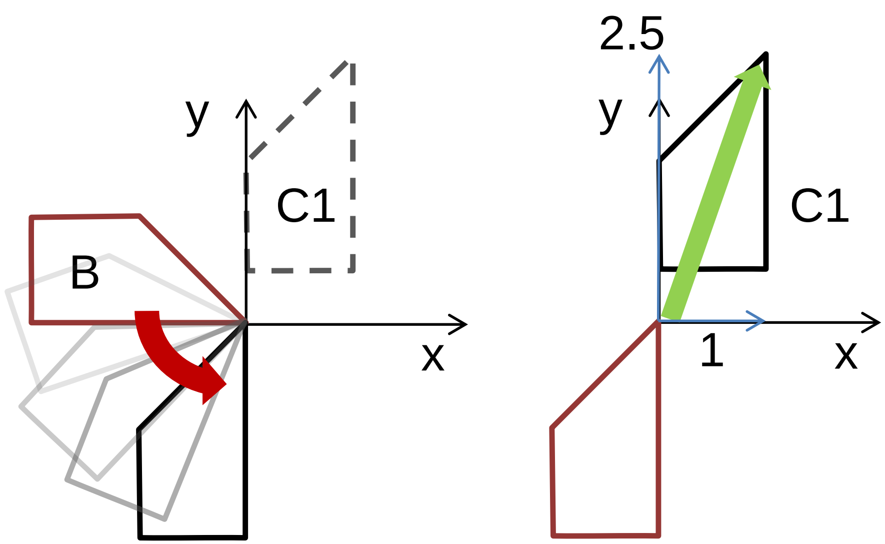


- Looks like no scaling is required, just translations and rotations
- Now observe where the pivot point should end in each one:



2D Transformation Example (4)

- C1: Rotate +90 degrees (remember, it is a CCW system)
- Then translate by (1, 2.5)



- The transformation then is:

$$C1 = \mathbf{T}_{(1.0,2.5)} \mathbf{R}_{\frac{\pi}{2}} \mathrm{B}$$

Important!

Rightmost transformations are applied first to the shapes (vertices)

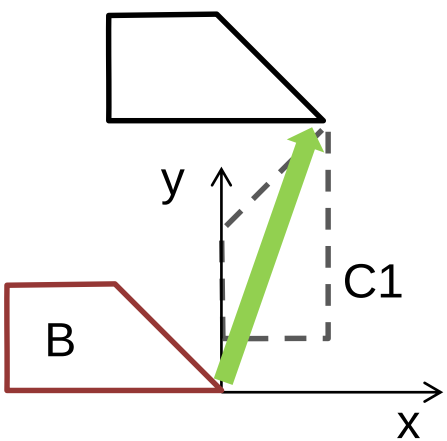
Here, we first rotate then translate

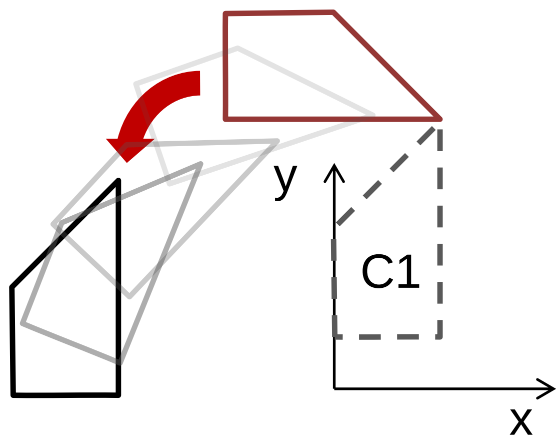
2D Transformation Example (5)

Important!

Order of transformation does matter
(remember: transformations are relative to origin)

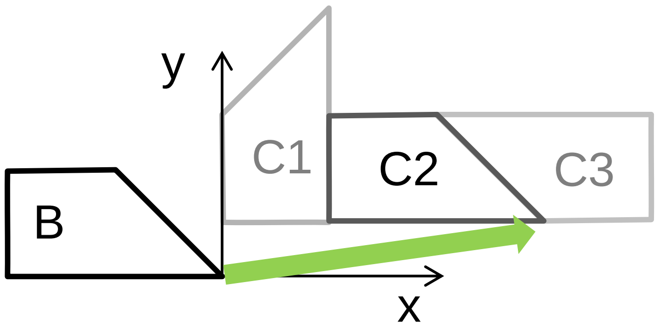
$$\mathbf{R}_{\frac{\pi}{2}}\mathbf{T}_{(1.0,2.5)}\mathrm{B} \neq \mathrm{C1}$$





2D Transformation Example (6)

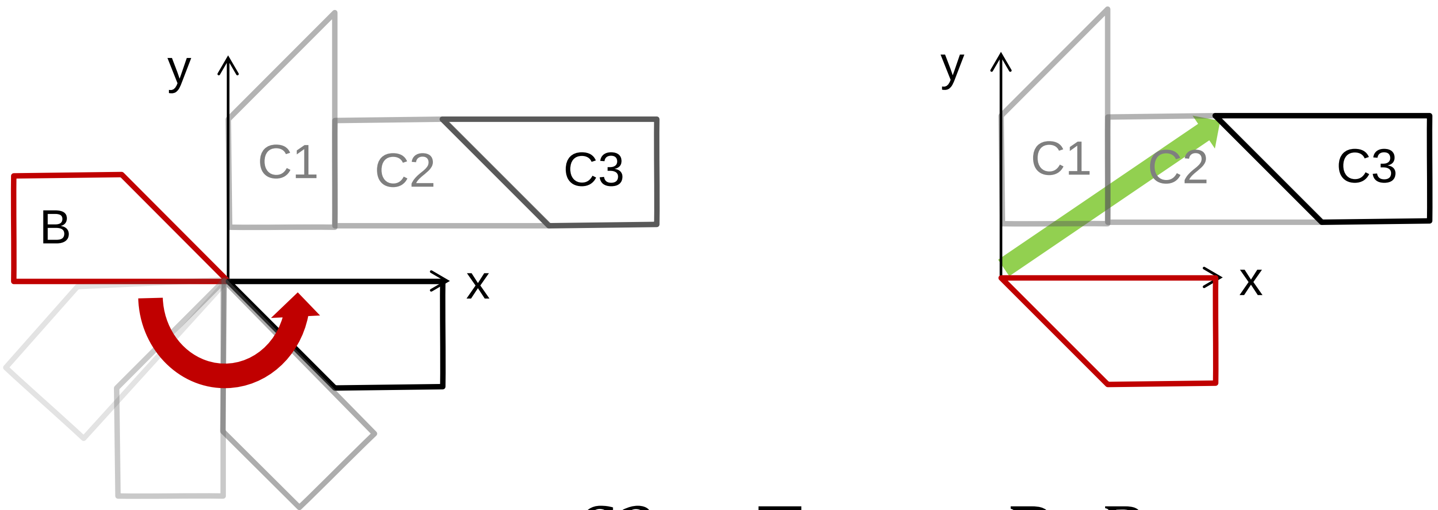
- Now lets do C2: This only requires a translation



$$C2 = \mathbf{T}_{(3.0,0.5)}B$$

2D Transformation Example (7)

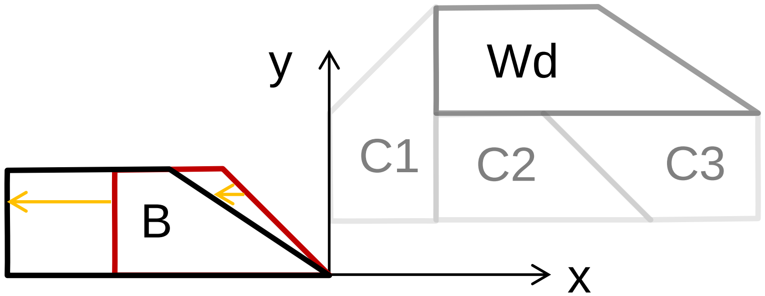
- C3: Rotate 180 degrees, then translate

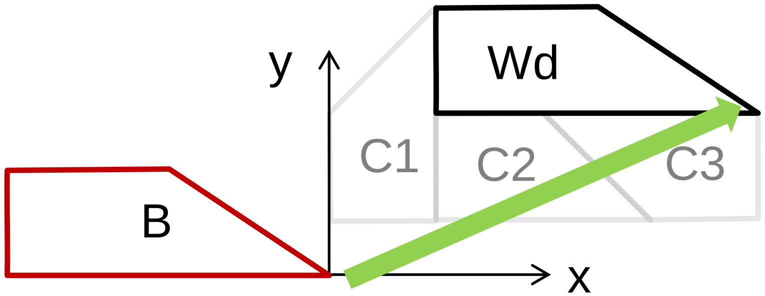


$$C3 = \mathbf{T}_{(2.0,1.5)}\mathbf{R}_{\pi}\mathrm{B}$$

2D Transformation Example (8)

- The windshield Wd is a single piece
- Although it is not a rotated version of B, it has a slightly different scale in the X axis, giving it a more slanted slope



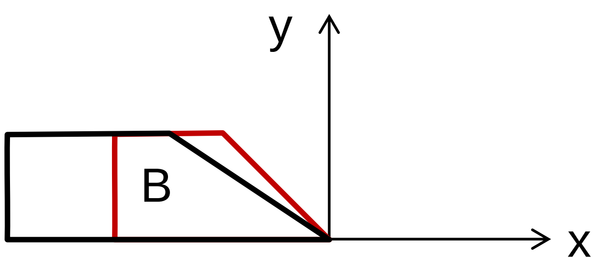


$$\mathrm{Wd} = \mathbf{T}_{(4.0,1.5)}\mathbf{S}_{1.5,1.0}\mathrm{B}$$

2D Transformation Example (9)

Important!

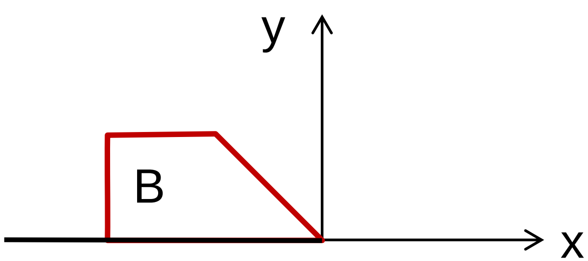
If you want to leave one coordinate unchanged, use a scaling factor of 1



$\mathbf{S}_{1.5,1.0}\mathrm{B}$

Wrong!

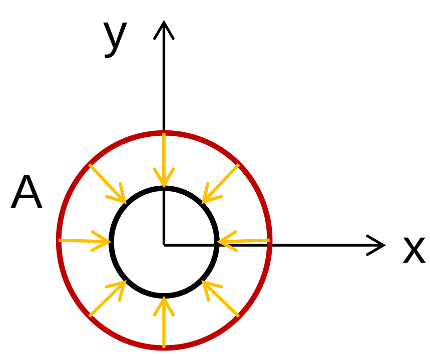
Never, **ever** zero the scaling factors!
This collapses the shape and the operation is irreversible (try inverting the corresponding scaling matrix…)



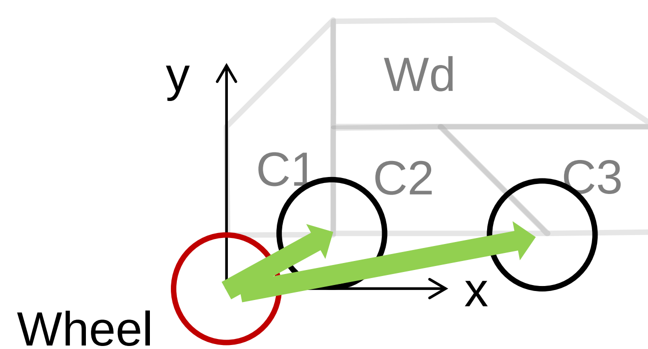
$\mathbf{S}_{1.5,0.0}\mathrm{B}$

2D Transformation Example (10)

- The wheels are easy to add since they are identical and only require a uniform scaling and translation
- Room for optimization:
  - Create a “wheel” object (by scaling once the original shape A)
  - Then only perform the translation:



$$\text{Wheel} = \mathbf{S}_{0.5,0.5}A$$

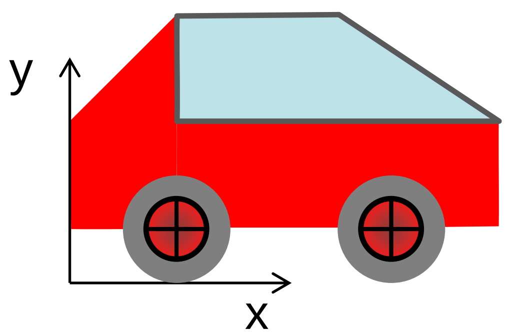


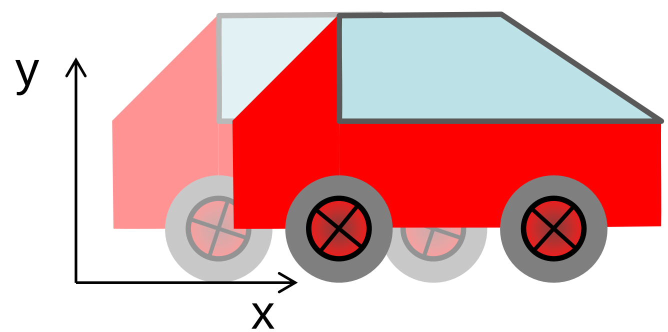
$$\text{W1} = \mathbf{T}_{(1.0,0.5)}\text{Wheel}$$

$$\text{W2} = \mathbf{T}_{(3.0,0.5)}\text{Wheel}$$

Let us Add Some Animation (1)

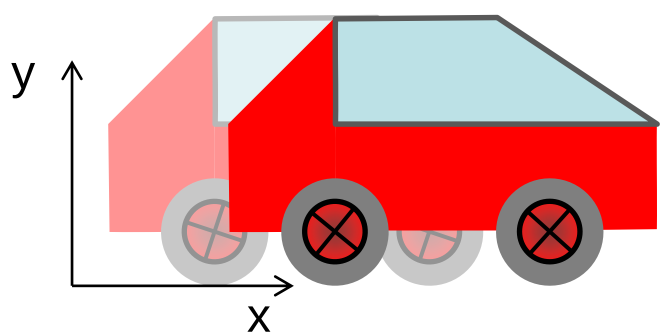
- What if instead of the flat-colored wheels we had a more interesting shape that would look better if rotated?
- What if the car could also move forward?





Let us Add Some Animation (2)

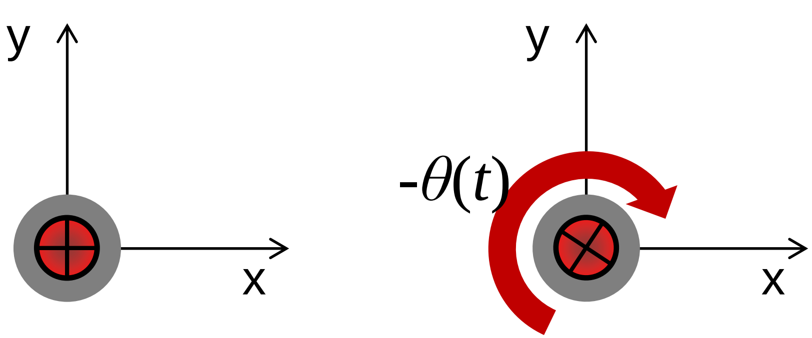
- Again, we need to decompose the problem and prioritize the motion:
  - Clearly, the wheels must be rotated around their axis
  - This must take place before moving them off the origin
  - The wheels should move in par with the rest of the car →
  - Therefore, the translation of the car should happen after the composition of the entire vehicle



Let us Add Some Animation (3)

- To spin the wheels, we must apply a rotation to the "wheel" entity, before translation
- Uniform scale and rotation can be safely interchanged
- We rotate according to a user-defined angle $\theta(t)$
- So, the new Wheel is:

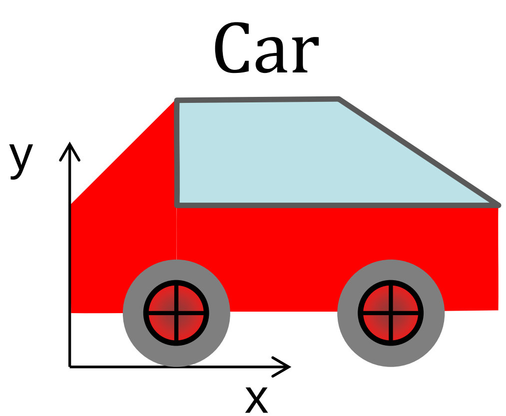
$$\text{Wheel} = \mathbf{R}_{-\theta(t)} \mathbf{S}_{0.5,0.5} \mathrm{A}$$



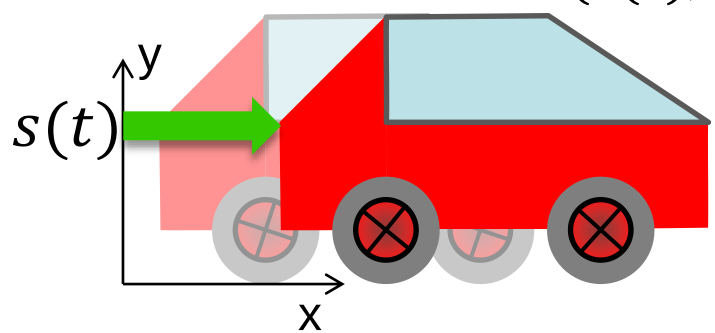
Note: Now the angle is negative, since this is a CW rotation

Let us Add Some Animation (4)

- Now to more efficiently apply the forward motion to the entire car, let's:
  - First group all of its components
  - Then apply the translation to the "car" group

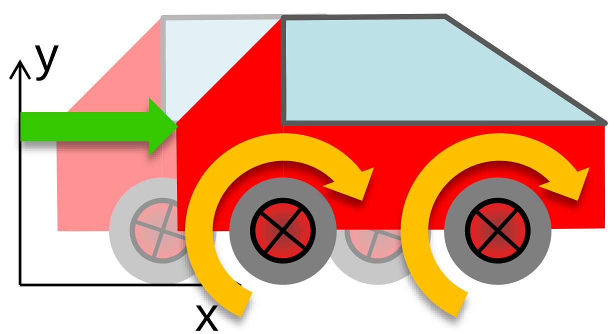


$$MovingCar = \mathbf{T}_{(s(t),0.0)}\mathrm{Car}$$



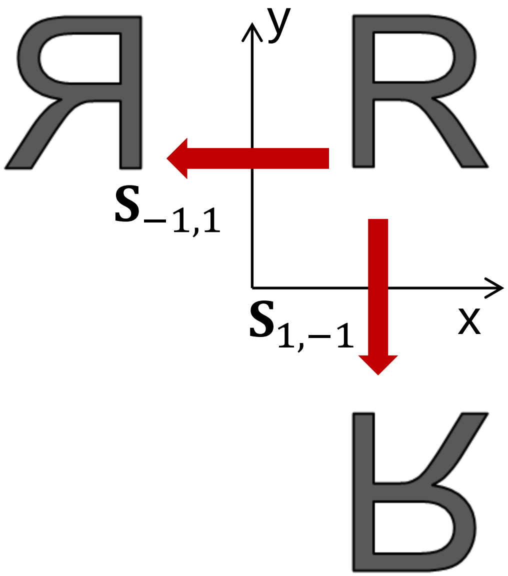
Let us Add Some Animation (5)

- Congratulations! You have just made your first transformation hierarchy, i.e. dependent transformations
- More on this in the Scene Management chapter



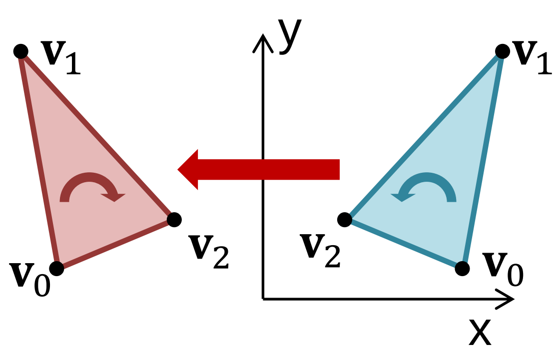
Mirroring (1)

- Via the scaling transformation, we can perform a switch of sides of the coordinates along an axis, by negating the scaling factor:



Mirroring (2)

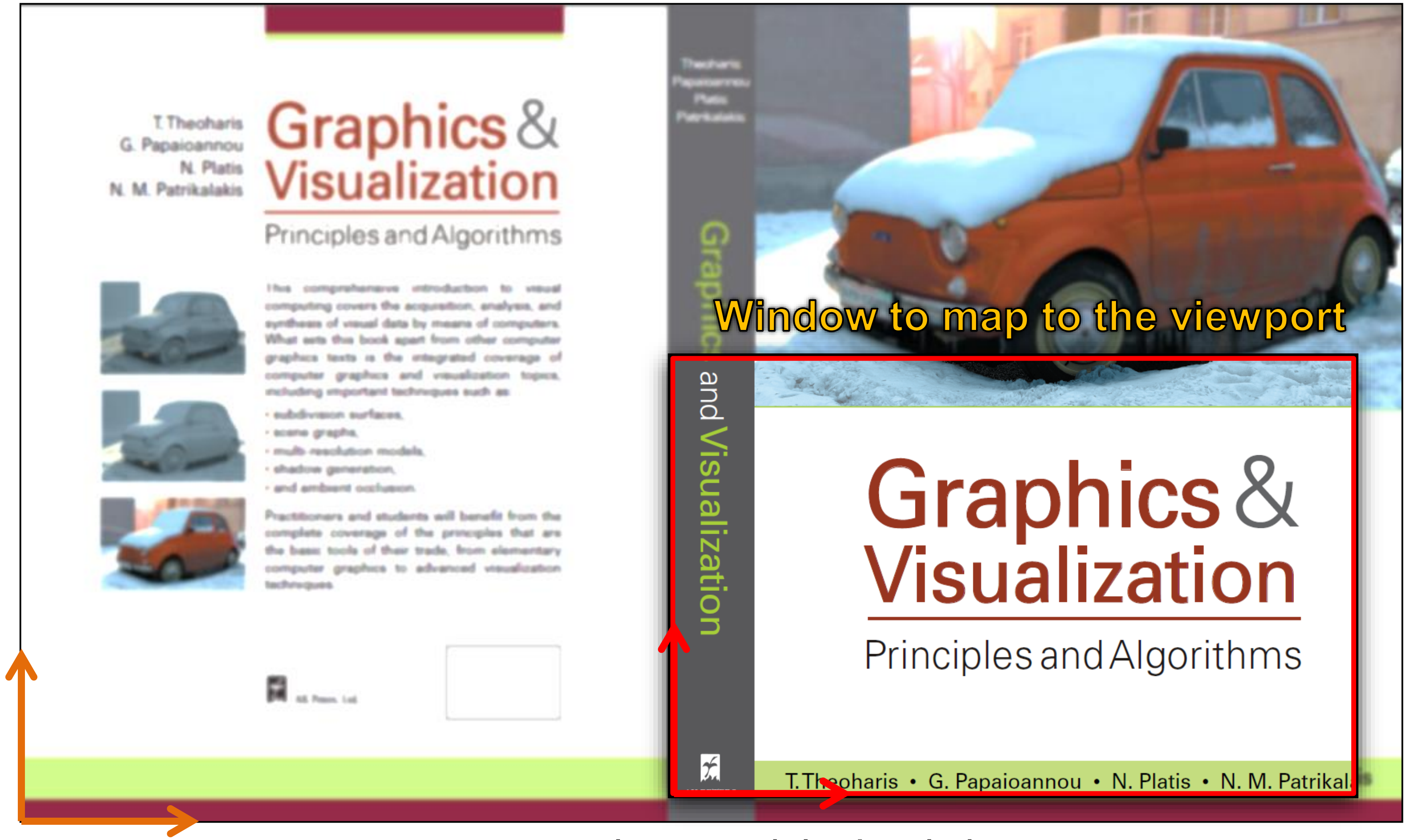
- However, caution must be exercised because mirroring changes the order of the vertices from CCW to CW and vice versa
- This can seriously impact many algorithms that depend on the correct ordering of the vertices (see Rasterization and shading)
- So, mirroring is best avoided, unless a re-ordering of the vertices can be done



Viewport Transformation (1)

- Shape coordinates on the 2D canvas or image plane are expressed relative to a global absolute coordinate system, which is independent of the output device size and resolution
  - E.g. a pdf document page has the 2D origin at one corner and may be measured in real units, such as centimeters
- The viewport transformation maps the coordinate system of the 2D canvas (image plane) to that of the actual viewport that the image should be generated in

Viewport Transformation (2)



Global reference system and units (e.g. cm)

Viewport (pixel units)

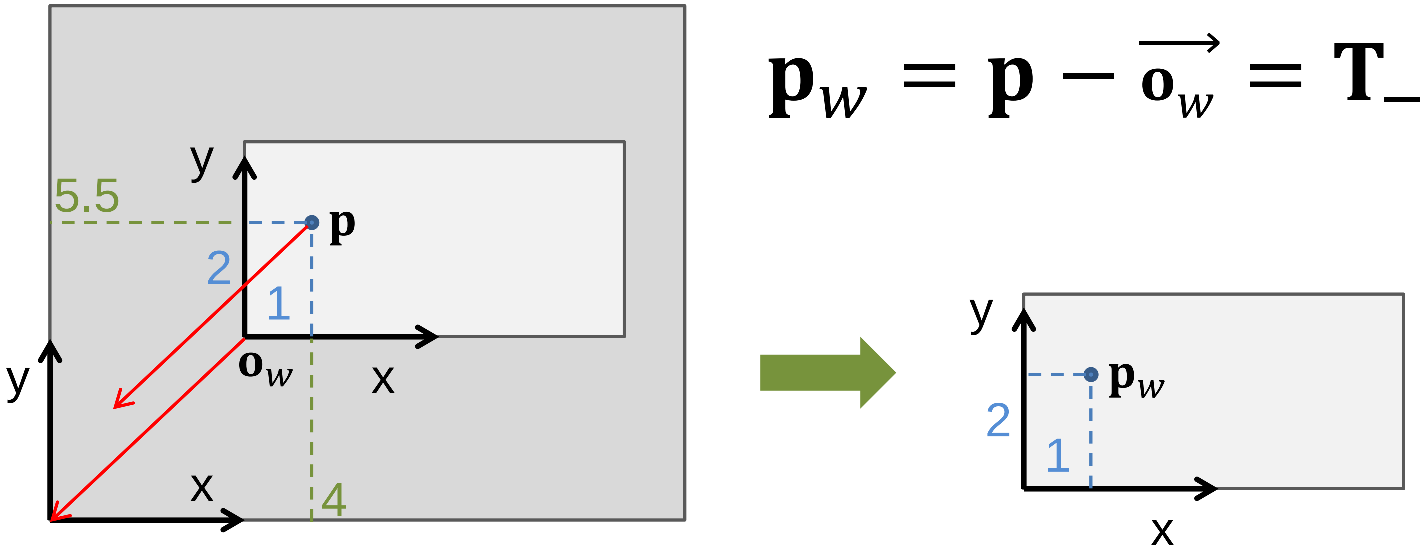
Viewport Transformation (3)

- What steps does the viewport transformation involve?
  - Definitely, we must first express the shapes relative to the corner of the window
  - We must scale the units
  - We must then express the contents of the window relative to the viewport’s shifted location in the image buffer (or screen)

Viewport Transformation (4)

- Express the shapes relative to the corner of the window:
  - "subtract" the window corner from the point coordinates → "move" the point and the window so that the two coordinate systems coincide:

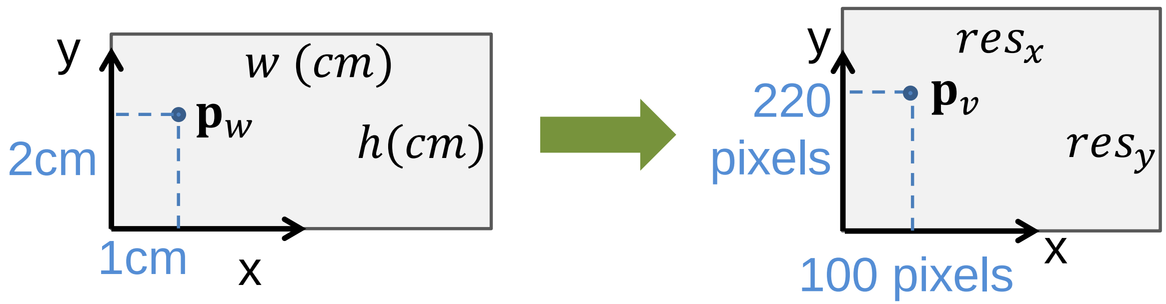
$$\mathbf{p}_w = \mathbf{p} - \overrightarrow{\mathbf{o}_w} = \mathbf{T}_{-\mathbf{o}_w}\mathbf{p}$$



Viewport Transformation (5)

- Now we must map the canvas units to the viewport size. Two options usually:
  - We are given a fixed “points-per-unit” metric (e.g. dpi – dots per inch), which is directly the scaling factor (can be different in x and y)
  - We are given the final resolution of the actual window, in “points” (pixels), in which case, we must derive the x, y scaling factors:

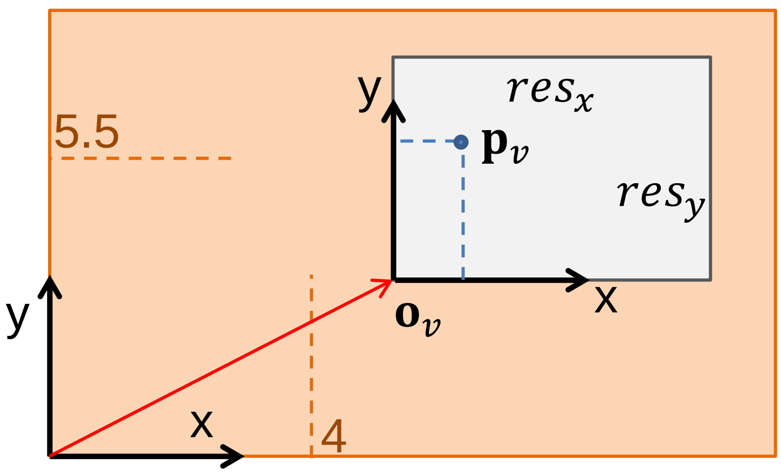
$$\mathbf{p}_v = \mathbf{S}_{\frac{res_x}{w},\frac{res_y}{h}}\mathbf{p}_w$$

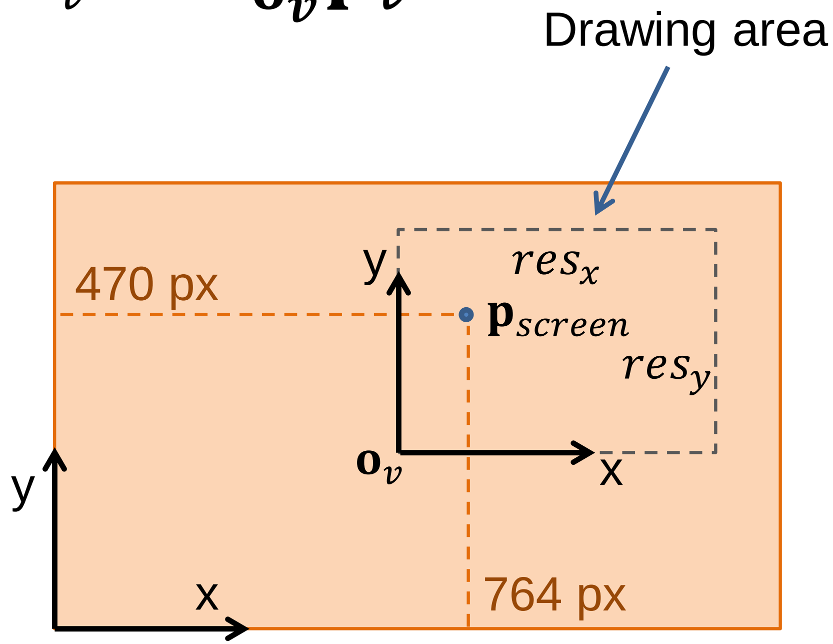


Viewport Transformation (6)

- Finally, we must (optionally) express the viewport coordinates w.r.t. the screen or drawing buffer:

$$\mathbf{p}_{screen} = \mathbf{p}_v + \overrightarrow{\mathbf{o}_v} = \mathbf{T}_{\mathbf{o}_v}\mathbf{p}_v$$





3D TRANSFORMATIONS

About 3D Transformations

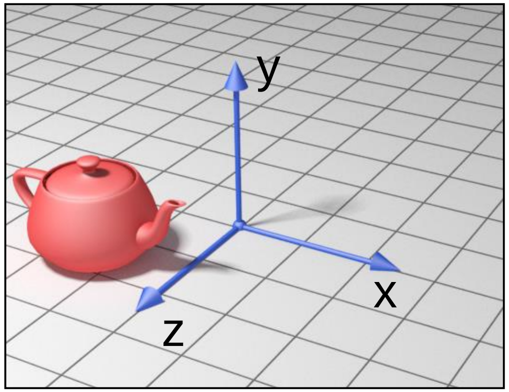
- Going to 3D, means adding one more coordinate, the z direction
- All 3D vectors are now expressed as 4-element columns in homogeneous coordinates
- All transformations become 4X4 matrixes
- Nothing else changes

A Third Dimension. Now What?

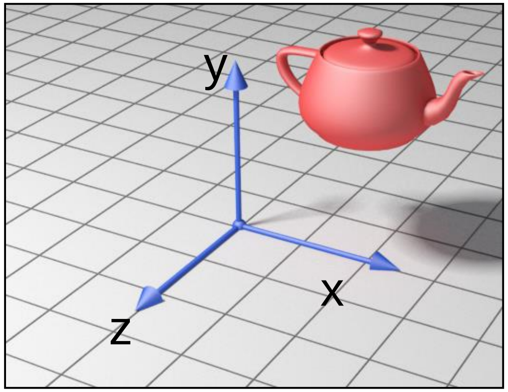
- Translation and scaling are augmented by a z coordinate
- Rotation:
  - We now have three coordinate axes to rotate around
  - In 2D, shapes revolved around a “z” axis perpendicular to the plane
  - In 3D, this becomes the rotation around Z
  - … and we also introduce a rotation around X and around Y

3D Geometric Transformations (1)

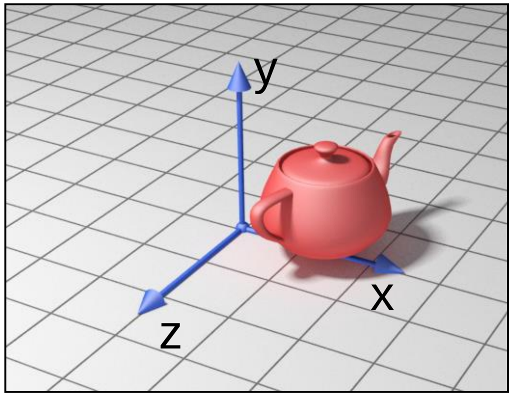
Translation:



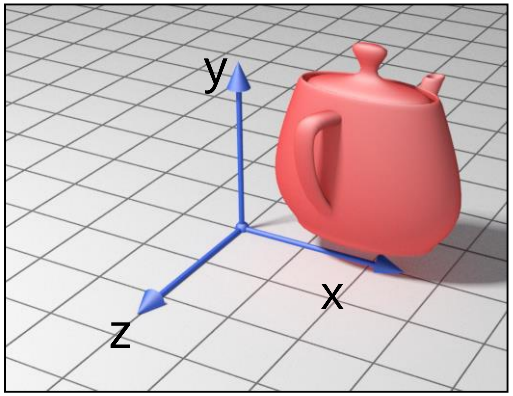
$$\mathbf{T}_{t_x,t_y,t_z} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



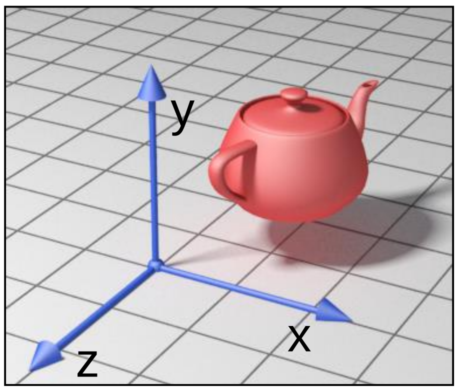
Scaling:



$$\mathbf{S}_{s_x,s_y,s_z} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

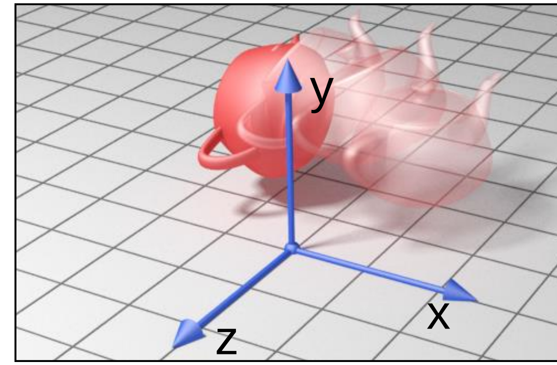


3D Geometric Transformations (2)



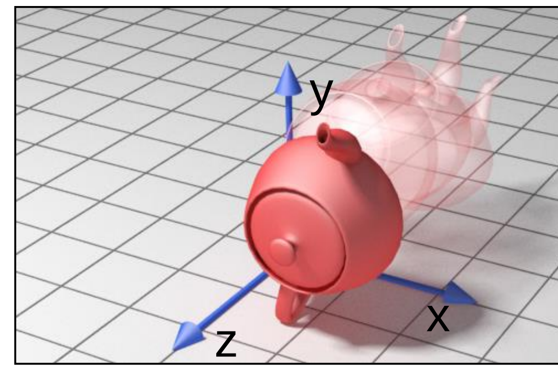
Rotation around Z:

$$\mathbf{R}_{z,\theta} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



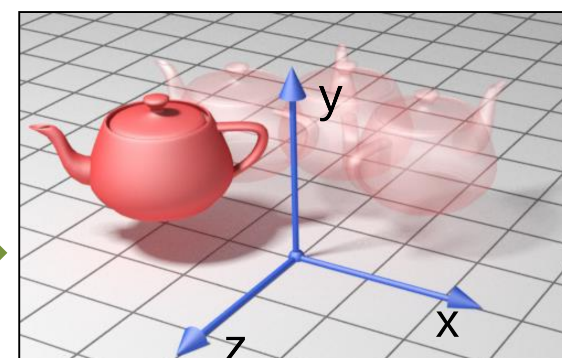
Rotation around X:

$$\mathbf{R}_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



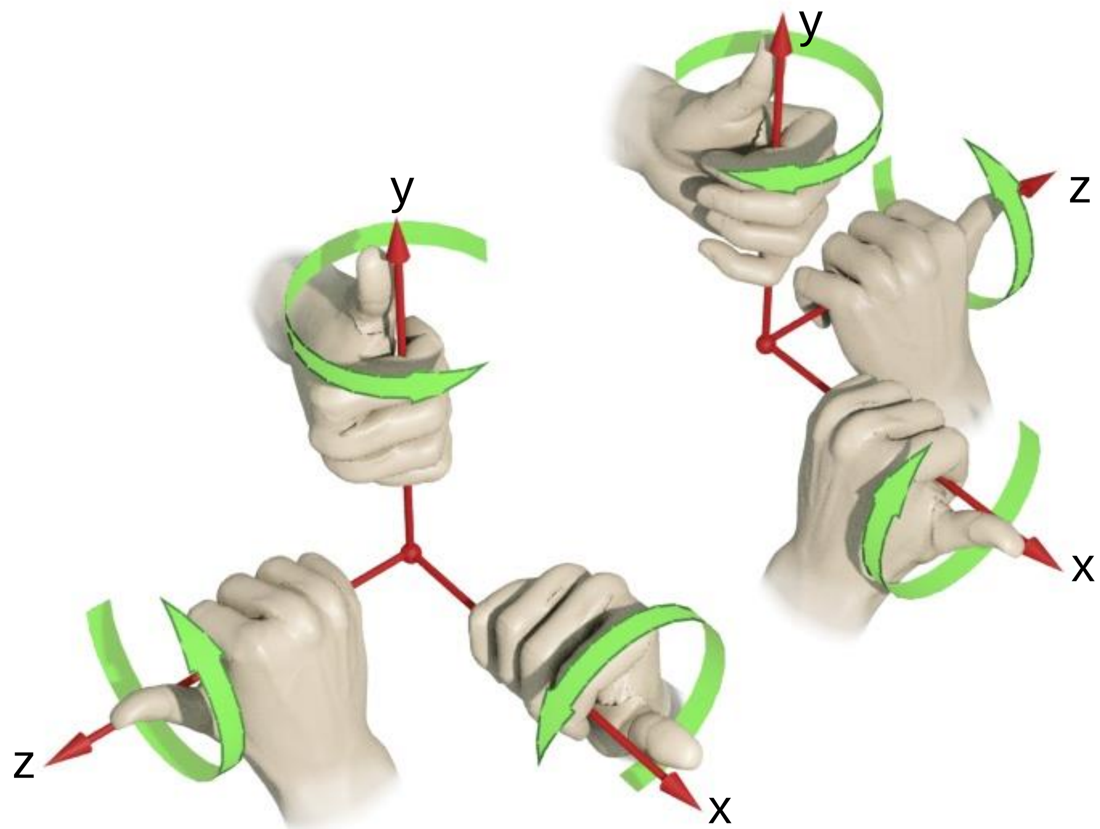
Rotation around Y:

$$\mathbf{R}_{y,\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



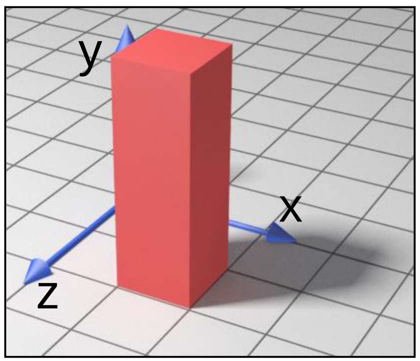
Rule of Thumb for Rotations

- Positive angles follow the curled hand, when thumb lies along the positive axis direction

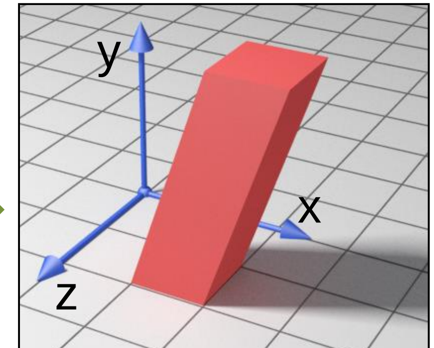


3D Geometric Transformations (3)

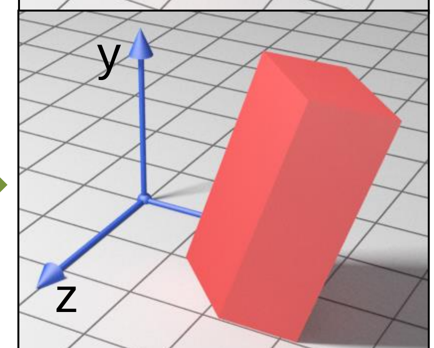
- Shearing: Many skew combinations. Examples:



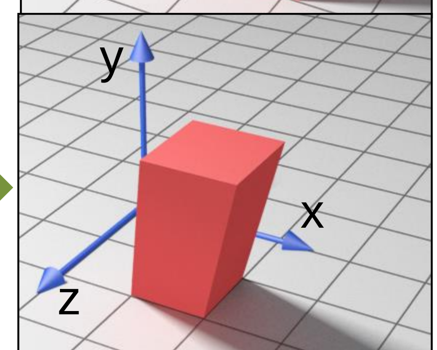
$$\mathbf{Sh}_{y \to x} = \begin{bmatrix} 1 & a & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\mathbf{Sh}_{y,z \to x} = \begin{bmatrix} 1 & a & b & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

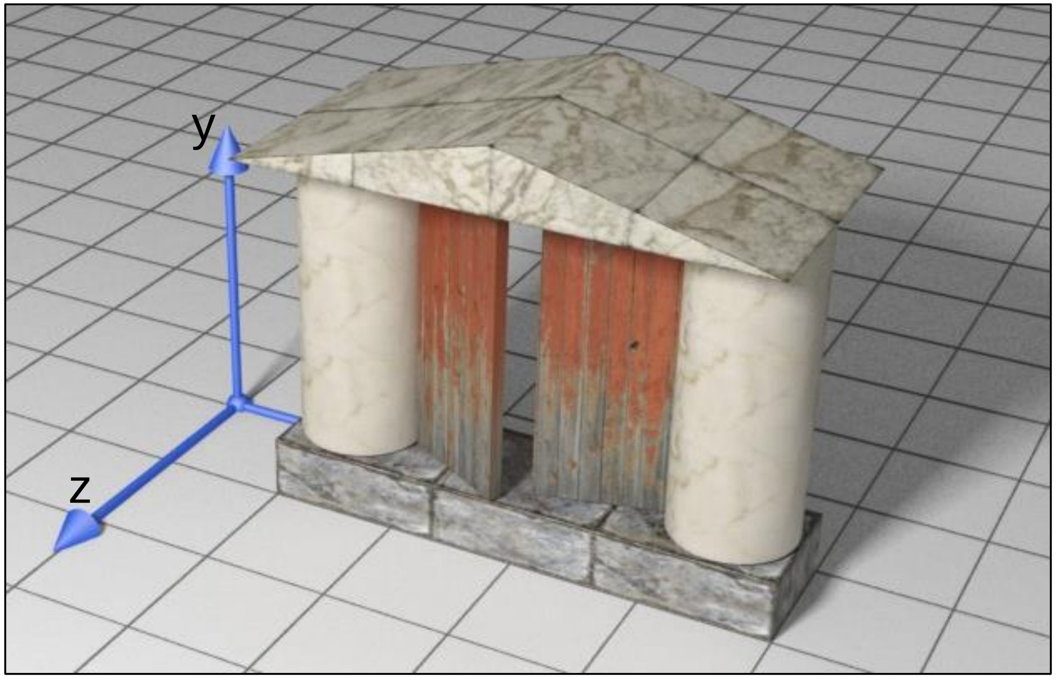


$$\mathbf{Sh}_{y \to x,z} = \begin{bmatrix} 1 & a & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & b & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

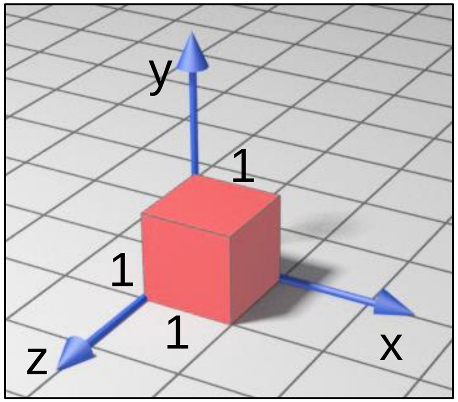


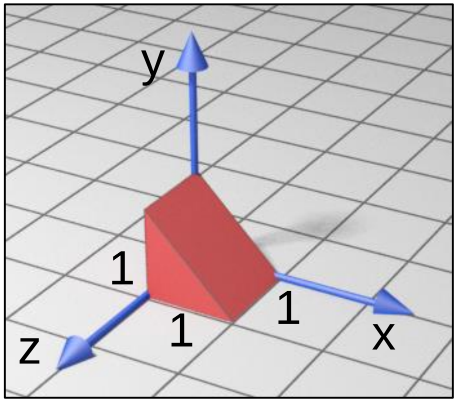
3D Transformations – Example (1)

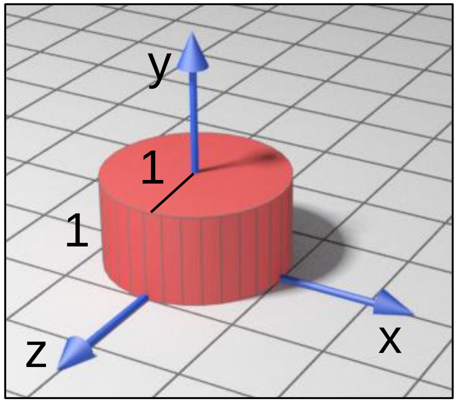
Build this:



Out of these:

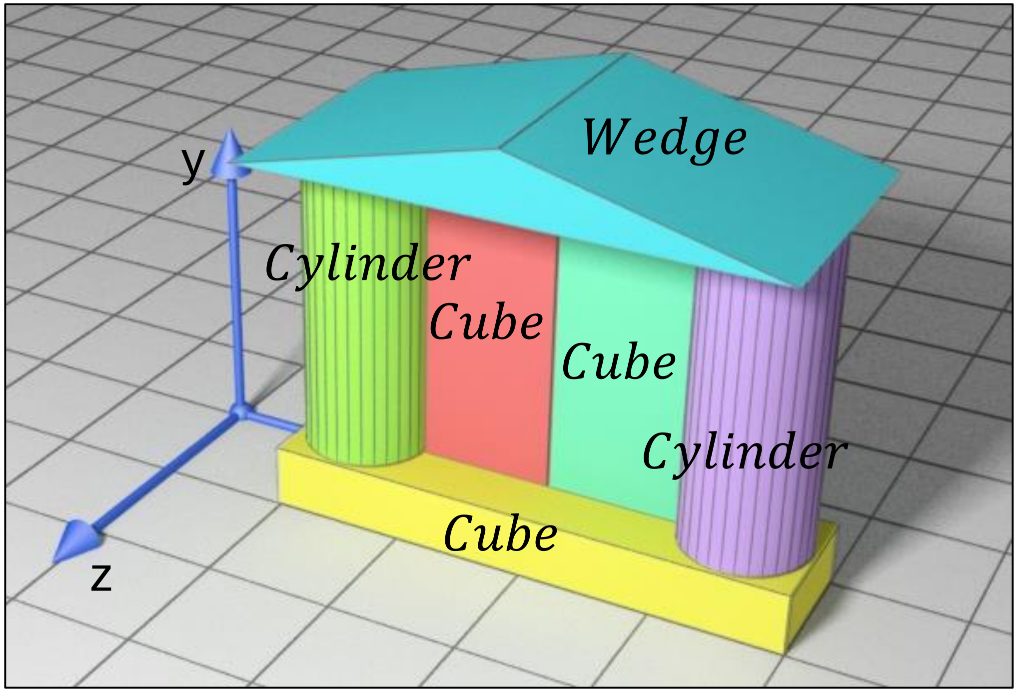


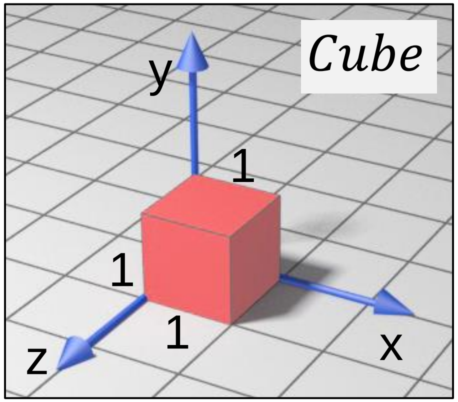


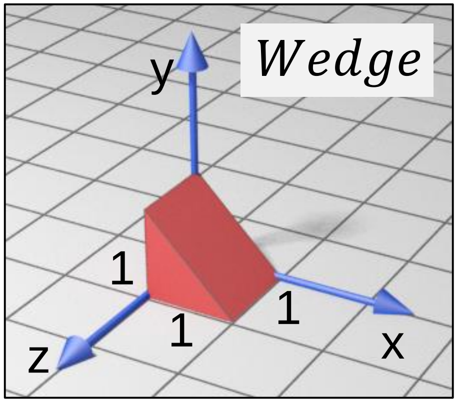


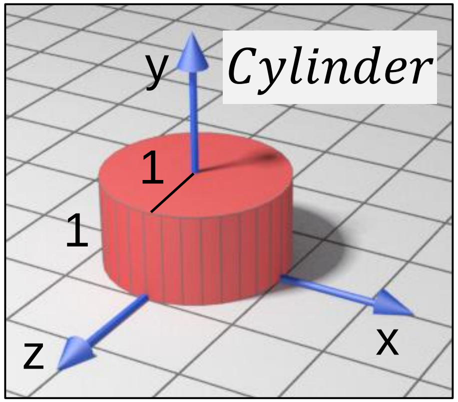
3D Transformations – Example (2)

- First, let's identify the elements of the structure:

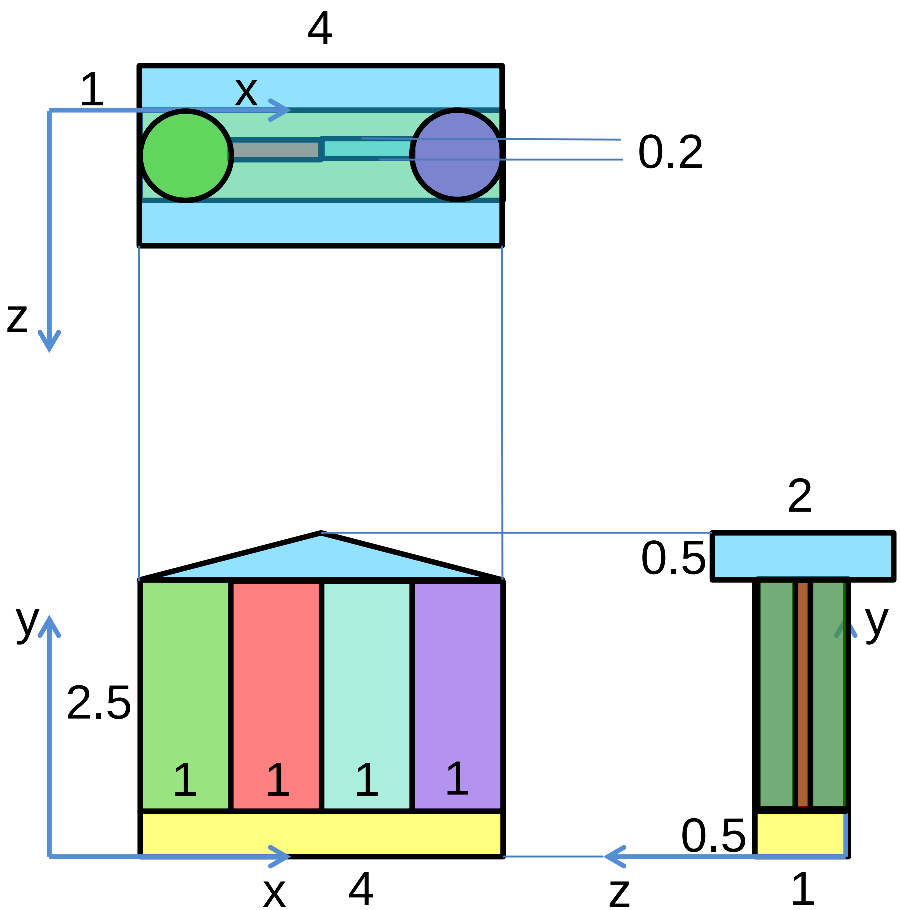


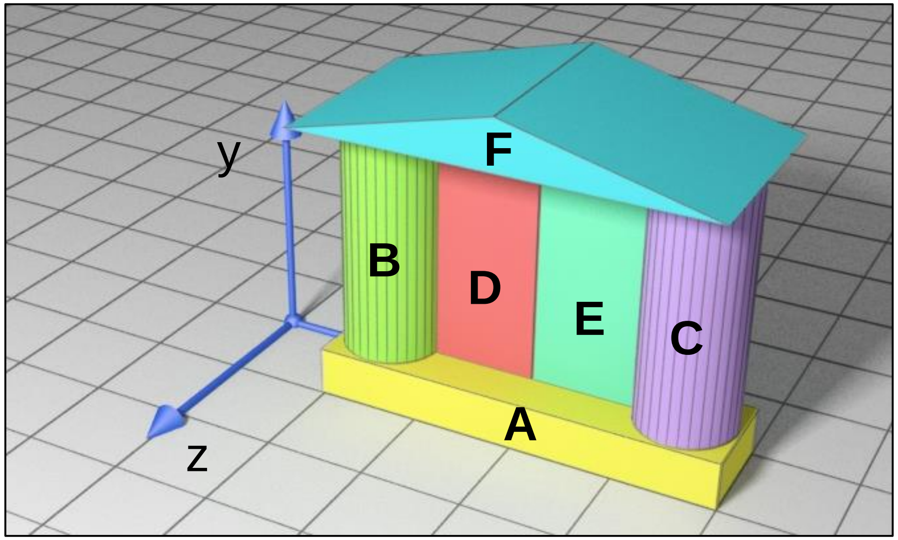






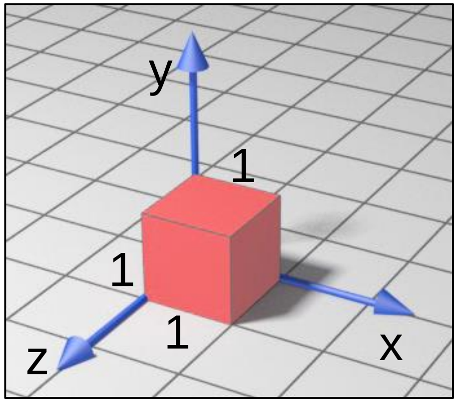
3D Transformations – Example (3)



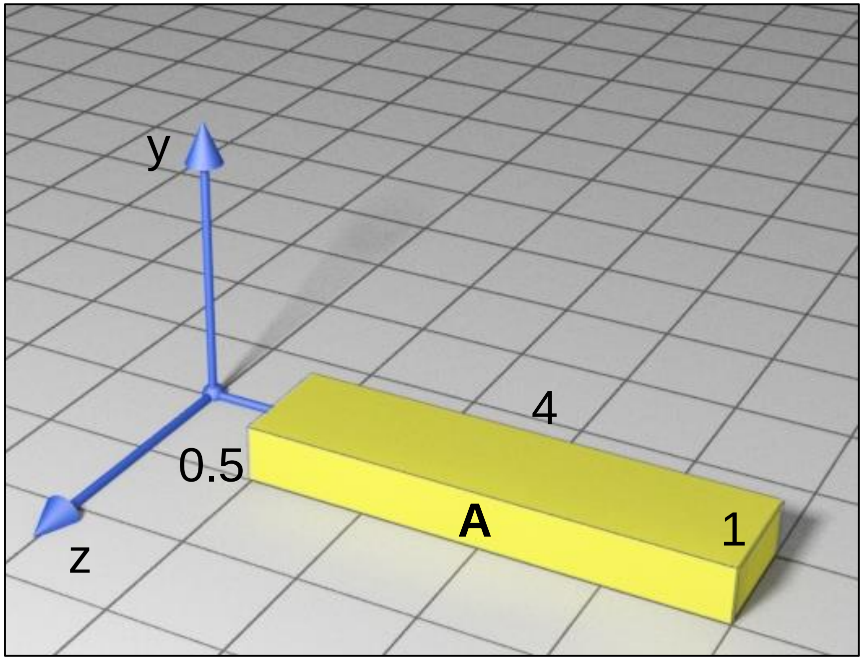


- It is also some times more convenient to think in 2D in order to decompose the transformations into simpler steps

3D Transformations – Example (4)

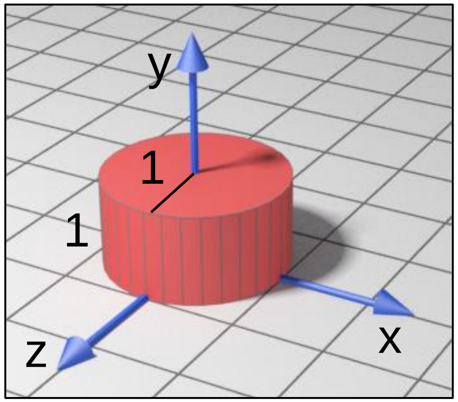


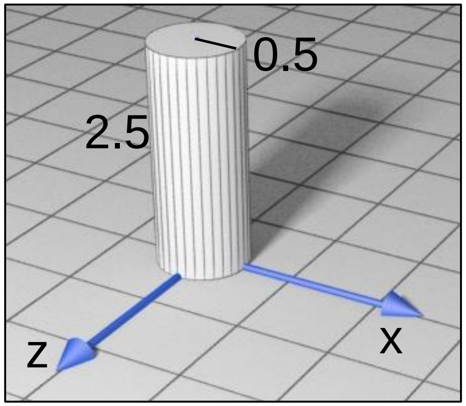
$$A = \mathbf{T}_{(1,0,0)}\mathbf{S}_{(4,0.5,1)}Cube$$

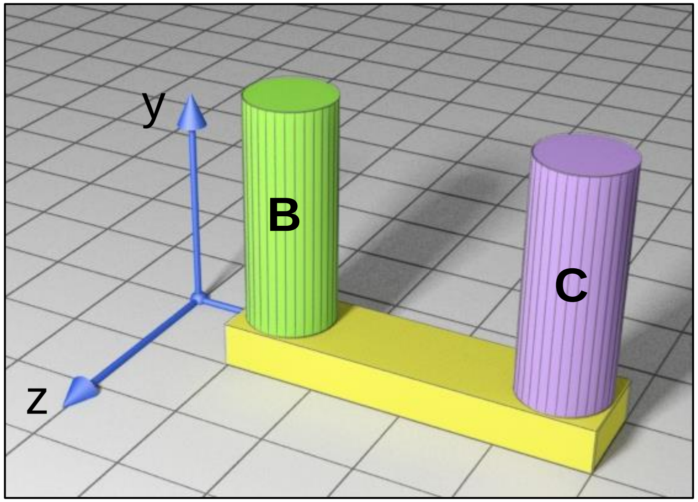


- Since one of the Cube corners is already at the origin, it is more convenient to first scale and then translate the piece to form part A

3D Transformations – Example (5)







$$Column = \mathbf{S}_{(0.5,2.5,0.5)} Cylinder$$

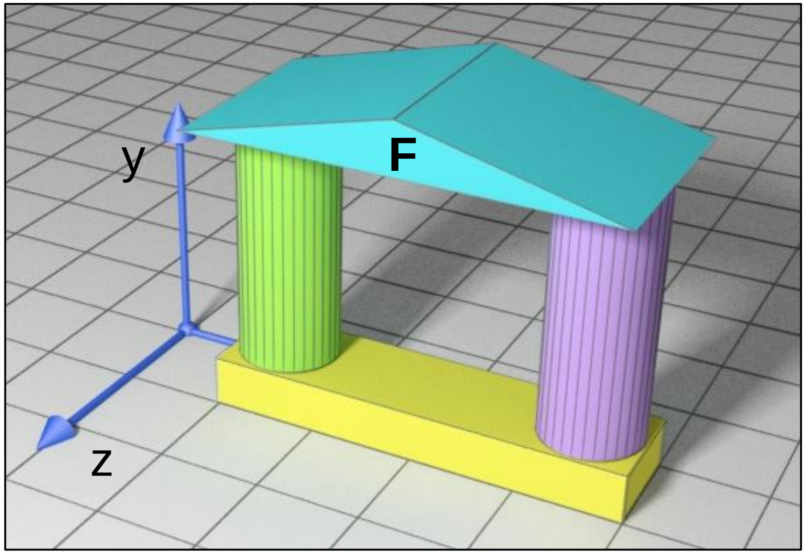
$$B = \mathbf{T}_{(1.5,0.5,0.5)} Column$$

$$C = \mathbf{T}_{(4.5,0.5,0.5)} Column$$

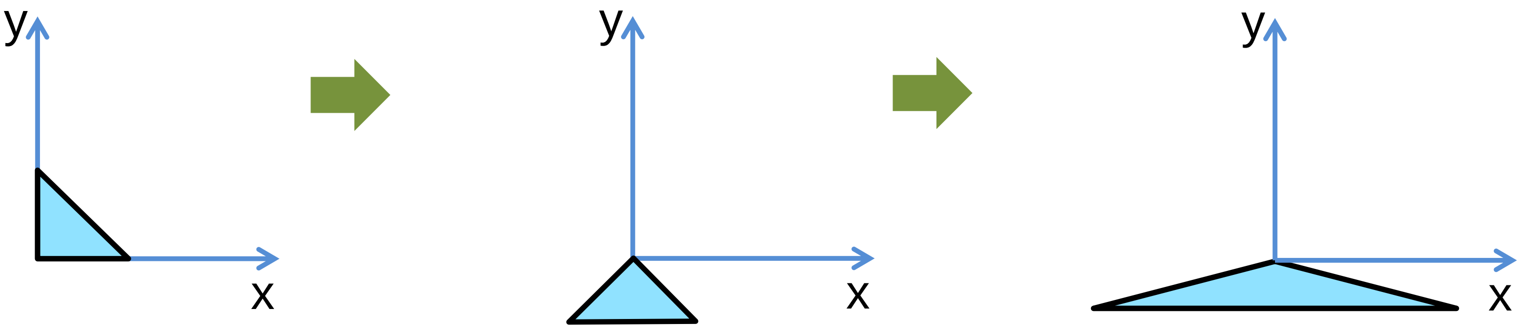
- We have 2 identical parts. We create deformed cylinder to match the column shape and then two *instances* of the same object are placed in their final position

3D Transformations – Example (6)

- The wedge is not conveniently oriented for scaling, since we need to scale along the hypotenuse

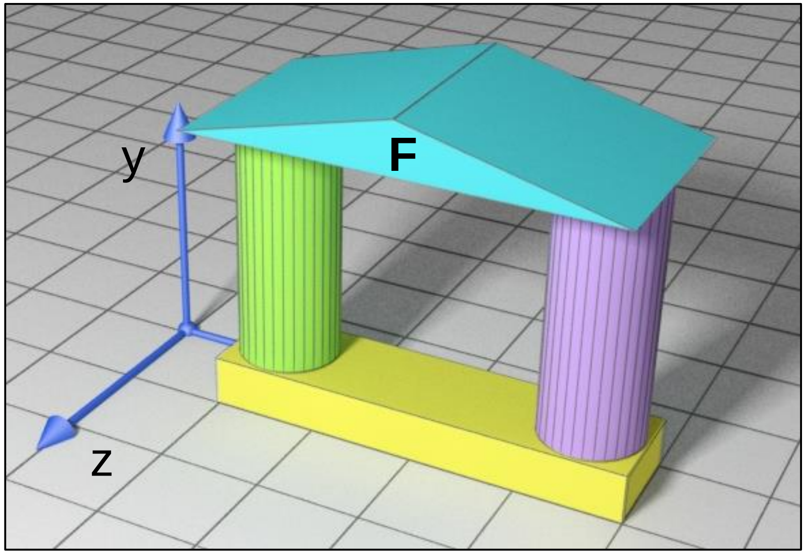


$$\mathbf{R}_{z,\frac{-3\pi}{4}} Wedge \qquad \mathbf{S}_{\frac{4}{\sqrt{2}},\frac{1}{\sqrt{2}},2}\mathbf{R}_{z,\frac{-3\pi}{4}} Wedge$$

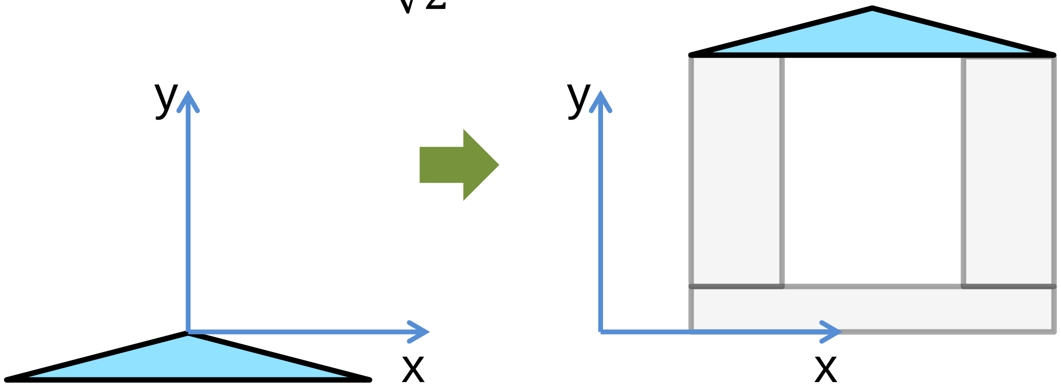


3D Transformations – Example (6)

- The wedge is not conveniently oriented for scaling, since we need to scale along the hypotenuse

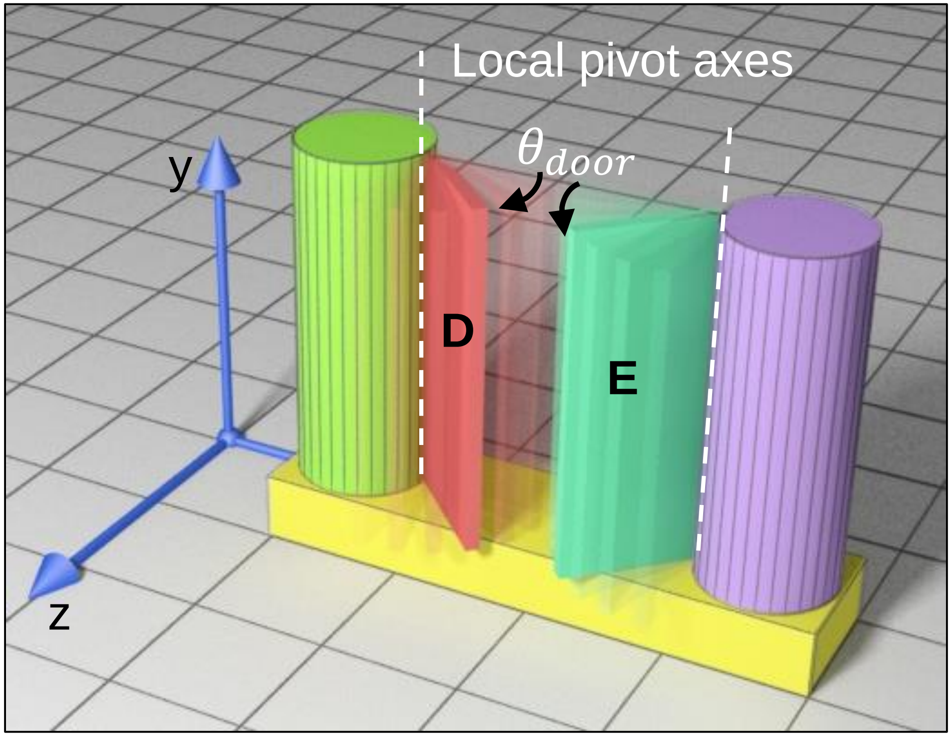


$$\mathbf{T}_{(3,3.5,-0.5)}\mathbf{S}_{\frac{4\sqrt{2}}{2},\frac{1}{\sqrt{2}},2}\mathbf{R}_{z,\frac{-3\pi}{4}}Wedge$$



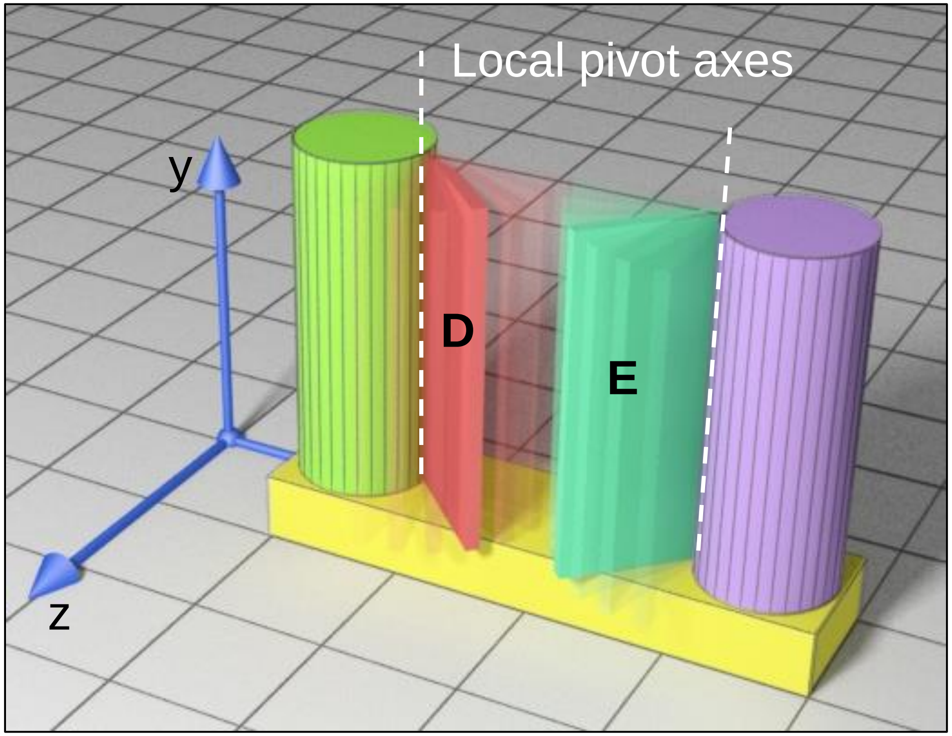
3D Transformations – Example (7)

- The two doors must parametrically swing open
- The door rotation is defined according to a local pivot axis
- The two rotations are of exactly opposite angle

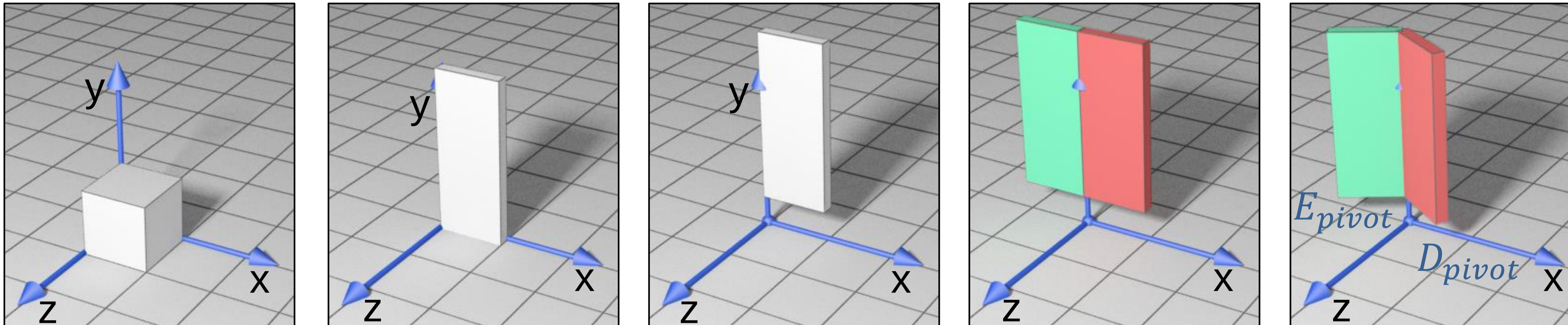


3D Transformations – Example (8)

- More convenient to first scale the cube then
- Translate it so that the Y axis coincides with the local pivot axis
- Then move the parts to their final position



3D Transformations – Example (9)



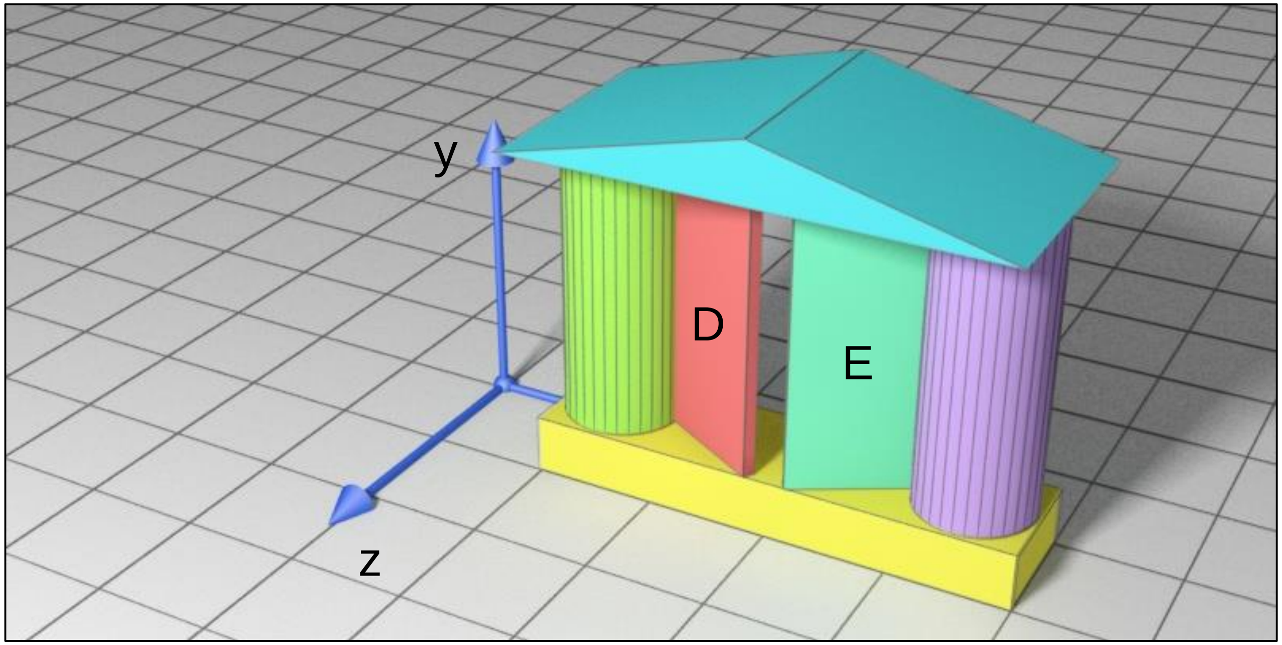
$$\mathbf{S}_{1,2.5,0.2}Cube$$

$$Door' = \mathbf{T}_{(0,0.5,-0.1)}\mathbf{S}_{1,2.5,0.2}Cube$$

$$D_{pivot} = \mathbf{R}_{y,-\theta_{door}}Door'$$

$$E_{pivot} = \mathbf{R}_{y,\theta_{door}}\mathbf{R}_{y,\pi}Door' = \mathbf{R}_{y,\pi+\theta_{door}}Door'$$

3D Transformations – Example (9)

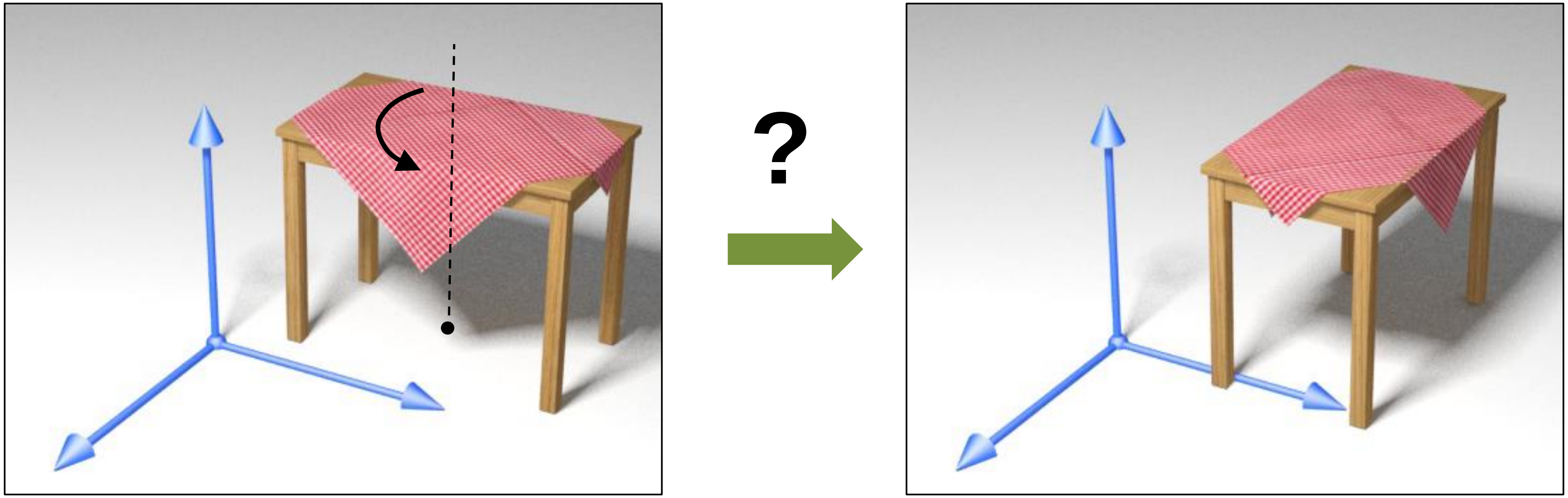


$$D = \mathbf{T}_{(2,0,0.5)}\mathbf{R}_{y,-\theta_{door}}\mathbf{T}_{(0,0.5,-0.1)}\mathbf{S}_{1,2.5,0.2}Cube$$

$$E = \mathbf{T}_{(4,0,0.5)}\mathbf{R}_{y,\pi+\theta_{door}}\mathbf{T}_{(0,0.5,-0.1)}\mathbf{S}_{1,2.5,0.2}Cube$$

Application – Transformation About Pivot (1)

- Very often, we require an arbitrary transformation relative to a user-defined pivot point and not the origin of the coordinate system

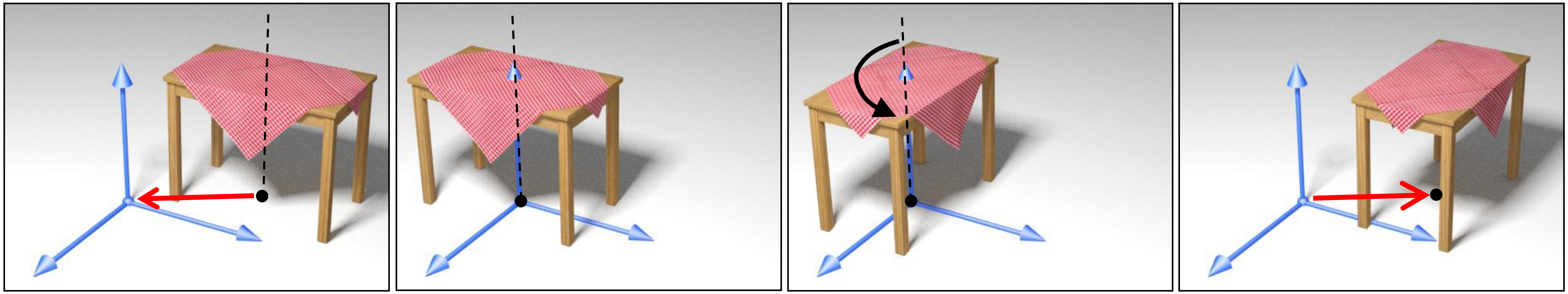


Application – Transformation About Pivot (2)

- Method:
  - Bring the shape and the pivot point to the origin
  - Apply the transformation
  - Bring the shape back

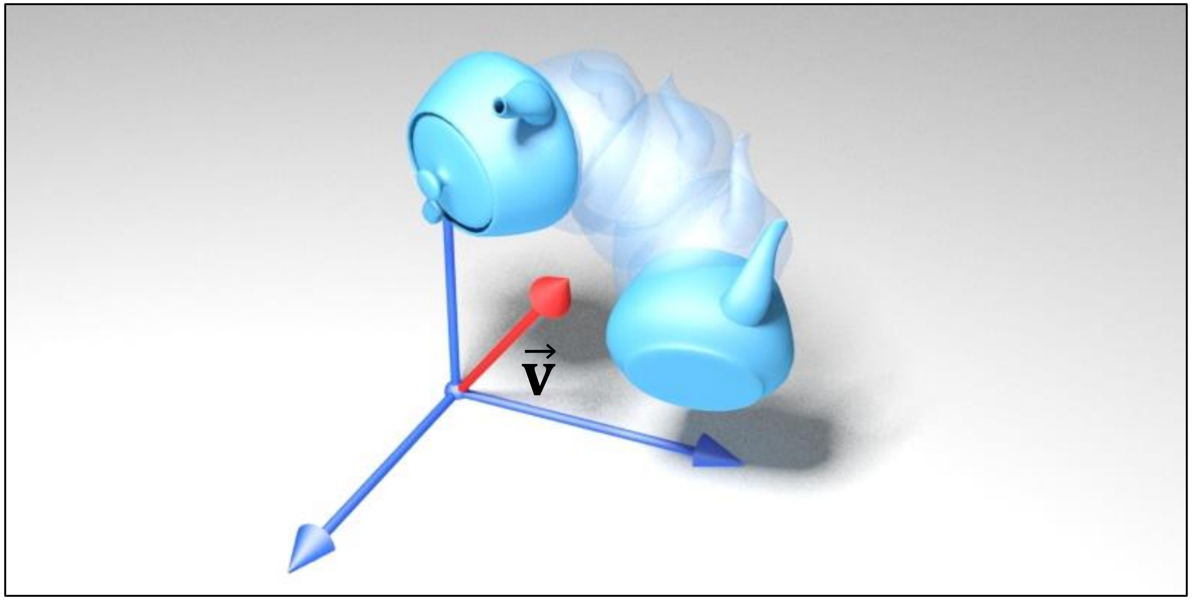
$$\mathbf{M}_{pivot(\mathbf{p})} = \mathbf{T}_{-\mathbf{p}}\mathbf{M}\mathbf{T}_{\mathbf{p}}$$

Note here that we only translated along the x,z coordinates of p, since the y coordinate is unaffected by the particular rotation



Application – Rotation Around Arbitrary Axis (1)

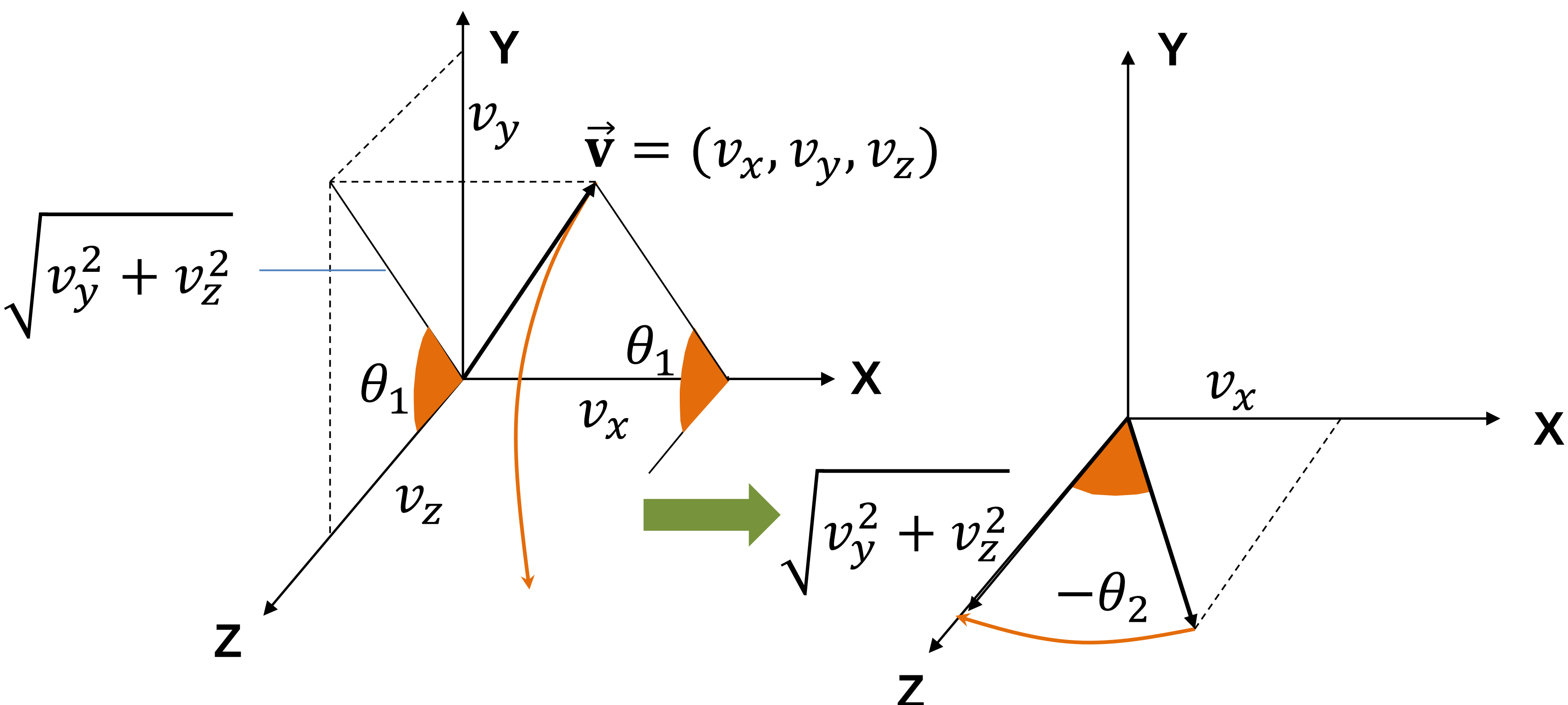
- Sometimes we need to rotate a shape around an arbitrary axis. How can we do that?



- The idea is to convert the arbitrary rotation to an axis-aligned rotation → The arbitrary axis must be forced to coincide with one coord. system axis

Application – Rotation Around Arbitrary Axis (2)

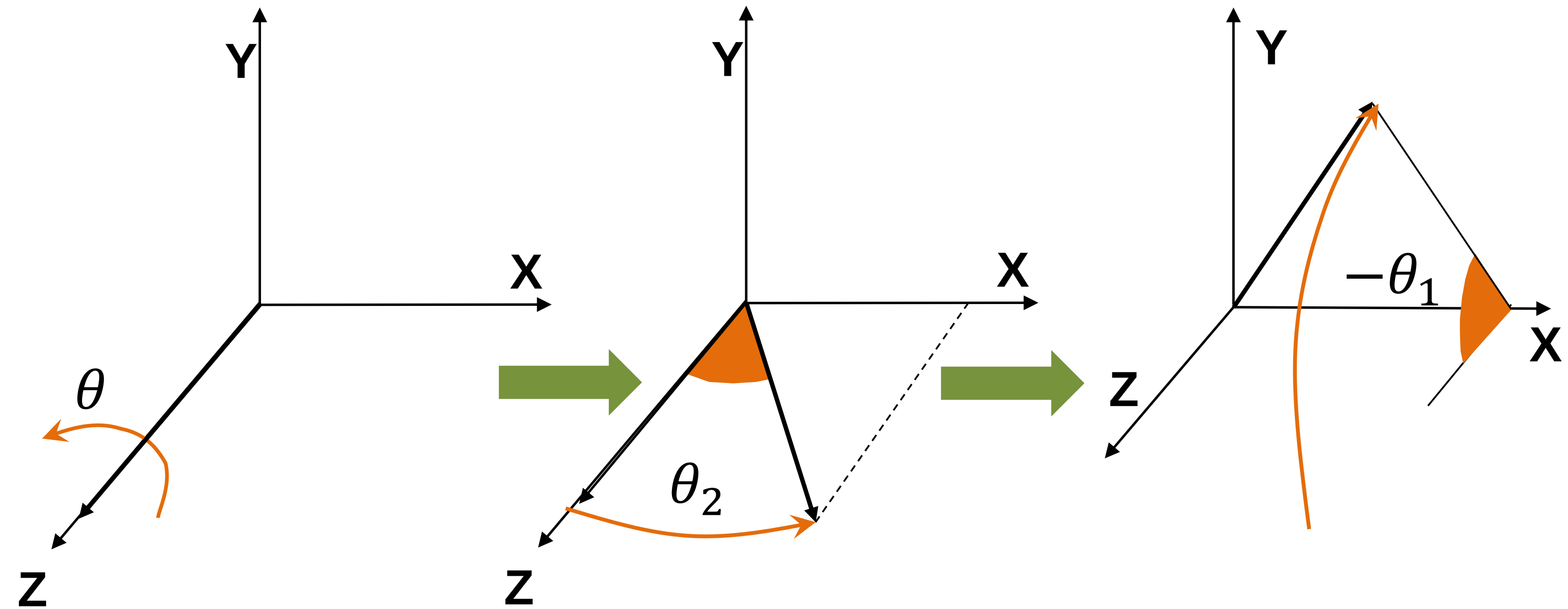
Collapse $\vec{\mathbf{v}}$ axis on e.g. z axis…



$$\theta_1 = \operatorname{atan}(v_y, v_z)$$

$$\theta_2 = \operatorname{atan}\left(v_x, \sqrt{v_x^2 + v_z^2}\right)$$

Application – Rotation Around Arbitrary Axis (3)



Do the rotation around z instead … and revert to the $\vec{\mathbf{v}}$ axis

$$\mathbf{R}_{\vec{\mathbf{v}},\theta} = \mathbf{R}_{x,-\theta_1}\mathbf{R}_{y,\theta_2}\mathbf{R}_{z,\theta}\mathbf{R}_{y,-\theta_2}\mathbf{R}_{x,\theta_1}$$

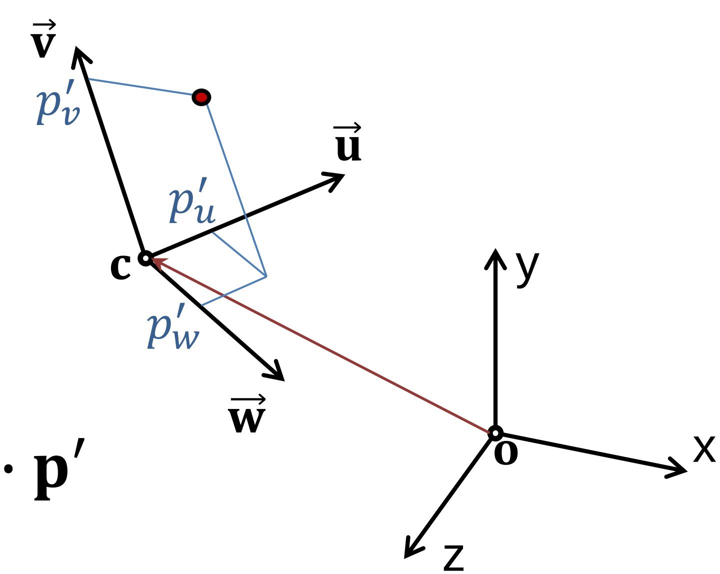
Change of Basis Transformation (1)

- Let $\mathbf{p}' = [\mathbf{p}]_{uvw}$ be the coordinates of $\mathbf{p}$ w.r.t. a coordinate system $\{\mathbf{c}, \vec{\mathbf{u}}, \vec{\mathbf{v}}, \vec{\mathbf{w}}\}$
- By definition, this means that: $\mathbf{p} = p'_u\vec{\mathbf{u}} + p'_v\vec{\mathbf{v}} + p'_w\vec{\mathbf{w}} + \boldsymbol{c}$ or

$$\mathbf{p} = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \begin{bmatrix} p'_u \\ p'_v \\ p'_w \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix}$$

- And in homogeneous coordinates:

$$\mathbf{p} = \begin{bmatrix} u_x & v_x & w_x & c_x \\ u_y & v_y & w_y & c_y \\ u_z & v_z & w_z & c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p'_u \\ p'_v \\ p'_w \\ 1 \end{bmatrix} = \mathbf{T_c} \cdot \mathbf{B} \cdot \mathbf{p}'$$

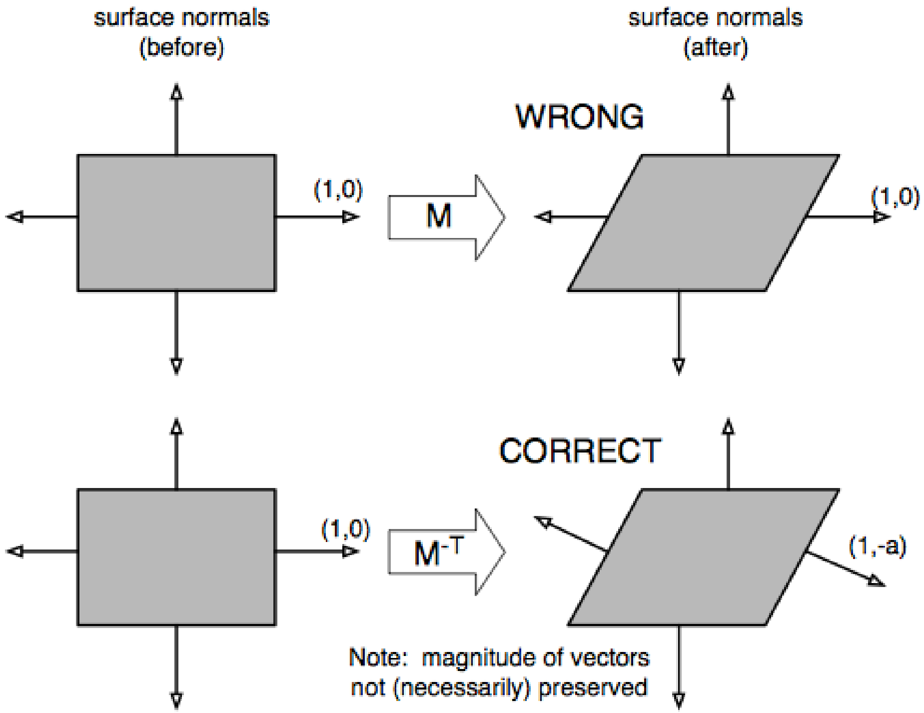


Change of Basis Transformation (2)

- So the transformation $\mathbf{T_c} \cdot \mathbf{B}$ is a rotation followed by translation that expresses the point from $\{\mathbf{c}, \vec{\mathbf{u}}, \vec{\mathbf{v}}, \vec{\mathbf{w}}\}$ to $\{\mathbf{o}, \hat{\mathbf{e}}_x, \hat{\mathbf{e}}_y, \hat{\mathbf{e}}_z\}$
- Therefore, the transformation $\mathbf{B}^{-1} \cdot \mathbf{T_{-c}} = \mathbf{B}^T \cdot \mathbf{T_{-c}}$ expresses the point from $\{\mathbf{o}, \hat{\mathbf{e}}_x, \hat{\mathbf{e}}_y, \hat{\mathbf{e}}_z\}$ to $\{\mathbf{c}, \vec{\mathbf{u}}, \vec{\mathbf{v}}, \vec{\mathbf{w}}\}$

- This pair of change of basis transformations is extremely useful in graphics, since we very often need to move from one coordinate system to another in many computations

Transforming Normals (1)



- Caution should be exercised when transforming "directions"
- It is **wrong** to apply arbitrary transformation matrices directly to normals

Transforming Normals (2)

- What is the correct transformation $\mathbf{M}_n$ given the 3X3 matrix $\mathbf{M}$(excluding translation)?
- Intuitively, we require that after the transformation, the normal is still perpendicular to any tangent vector $\mathbf{v}$:

$$(\mathbf{M}_n\mathbf{n}) \cdot (\mathbf{M}\mathbf{v}) = 0$$

Or, after manipulating the matrices to express the dot product in matrix form:

$$(\mathbf{M}_n\mathbf{n})^T(\mathbf{M}\mathbf{v}) = \mathbf{n}^T\mathbf{M}_n^T\mathbf{M}\mathbf{v} = 0$$

Transforming Normals (3)

$$(\mathbf{M}_n\mathbf{n})^T(\mathbf{M}\mathbf{v}) = \mathbf{n}^T\mathbf{M}_n^T\mathbf{M}\mathbf{v} = 0$$

But by the definition of the tangent vector $\mathbf{v}$: $\mathbf{n}^T\mathbf{v} = 0$ and therefore we require:

$$\mathbf{M}_n^T\mathbf{M} = \mathbf{I} \Rightarrow$$

$$\boxed{\mathbf{M}_n = (\mathbf{M}^{-1})^T}$$

Contributors

- Georgios Papaioannou