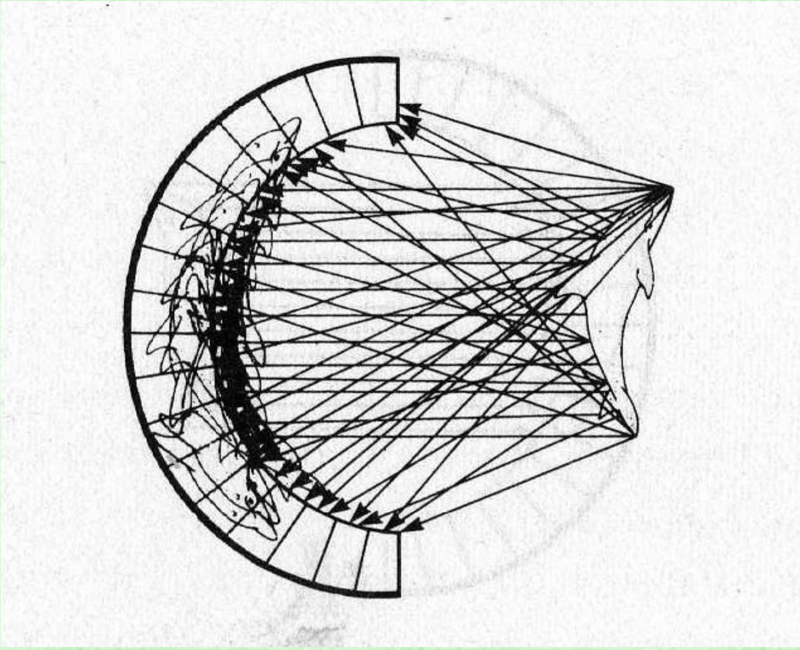
The Virtual World

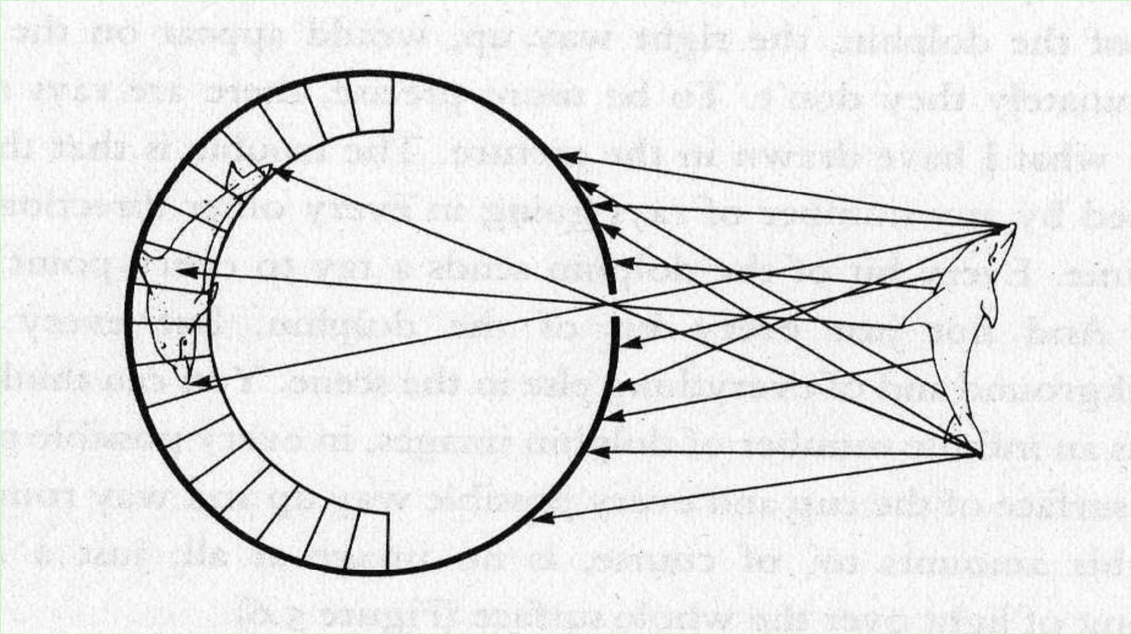
Building a Virtual World

- Goal: mimic human vision in a virtual world (with a computer)
  - Cheat for efficiency, using knowledge about light and the eye (e.g. from the last lecture)
- Create a virtual camera: place it somewhere and point it at something
- Put film (containing pixels, with RGB values ranging from 0-255) into the camera
  - Taking a picture creates film data as the final image
- Place objects into the world, including a floor/ground, walls, ceiling/sky, etc.
  - Two step process: (1) make objects (geometric modeling), (2) place objects (transformations)
  - Making objects is itself a two-step process: (1) build geometry (geometric modeling), (2) paint geometry (texture mapping)
- Put lights into the scene (so that it's not completely dark)
- Finally, snap the picture:
  - "Code" emits light from (virtual) light sources, bounces that light off of (virtual) geometry, and follows that bounced light into the (virtual) camera and onto the (virtual) film
  - We will consider 2 methods (scanline rendering and ray tracing) for the taking this picture

Pupil

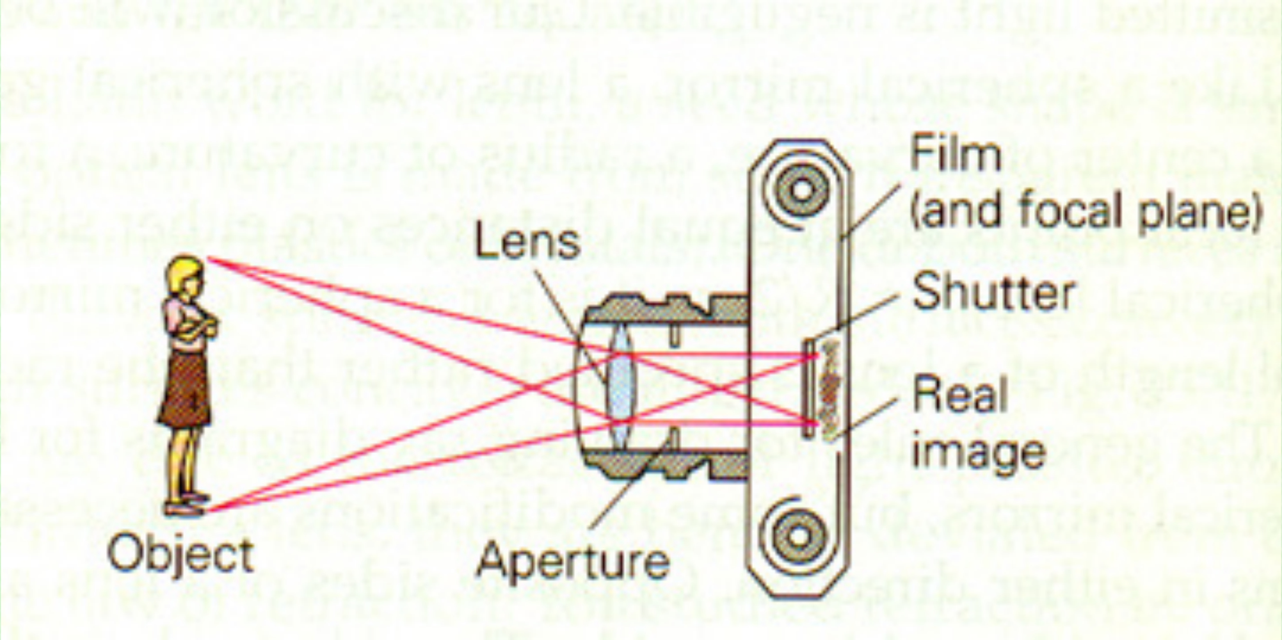
- Light emanates off of every point of an object outwards in every direction
  - That's why we can all see the same spot on the same object
  - Light leaving that spot/point (on the object) is entering each of our eyes
- Without a pupil, light from every point on an object would hit the same cone on our eye, averaging/blurring the light information
- The (small) pupil restricts the entry of light so that each cone only receives light from a small region on the object, giving interpretable spatial detail

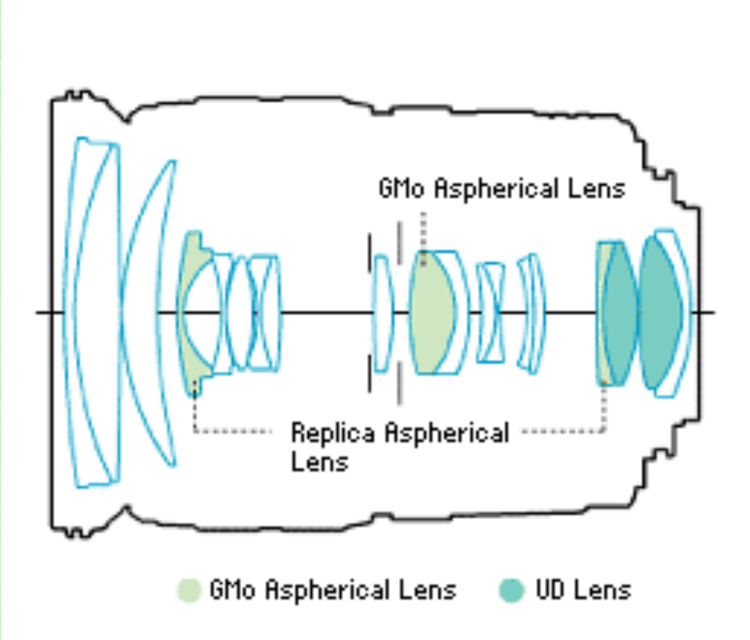




Aperture

- Cameras are similar to the eye (with mechanical as opposed to biological components)
- Instead of cones, the camera has mechanical pixels
- Instead of a pupil, the camera has a small (adjustable) aperture for light to pass through
- Cameras also typically have a hefty/complex lens system





Aside: Lens Flare

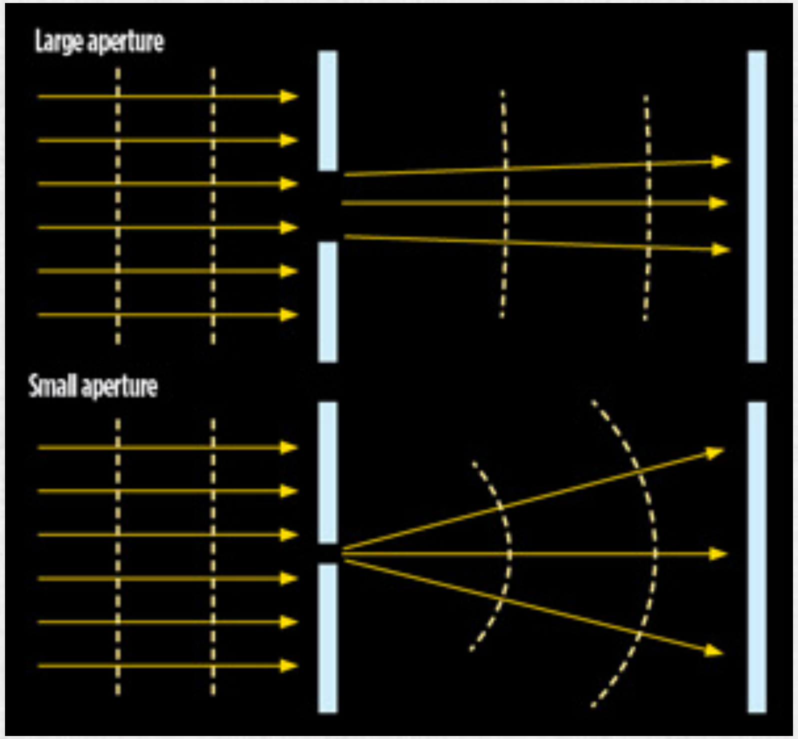
- Many camera complexities are (often) not properly accounted for in virtual worlds
- Thus, certain effects (such as depth of field, motion blur, chromatic aberration, lens flare, etc.) have to be approximated/modeled in other ways (as we will discuss later)
- Example: Lens flare is caused by a complex lens system reflecting/scattering light
  - This depends on material inhomogeneities in the lenses, the geometry of lens surfaces, absorption/dispersion of lenses, antireflective coatings, diffraction, etc.

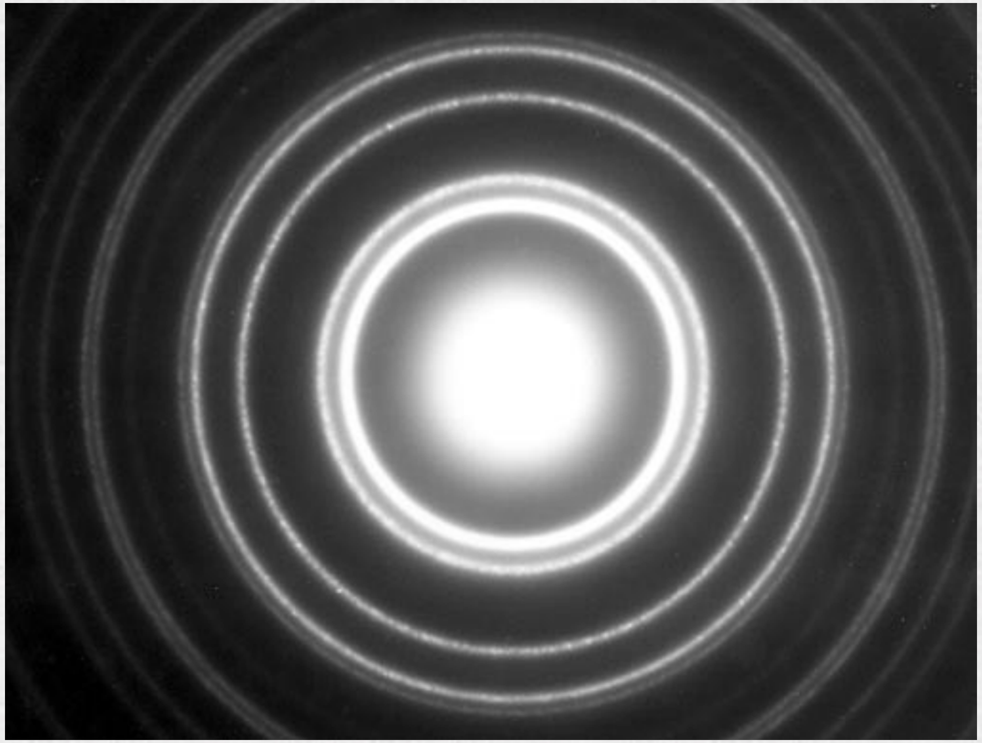
Pinhole Camera

- The pupil/aperture has to have a finite size in order for light to be able to pass through it
- When too small, not enough light enters and the image is too dark/noisy to interpret
  - In addition, light can diffract (instead of traveling in straight lines) distorting the image
- When too large, light from a large area of an object hits the same cone (causing blurring)
- Luckily, the virtual camera can use a single point for the aperture (without worrying about dark or distorted images)

Aside: Diffraction

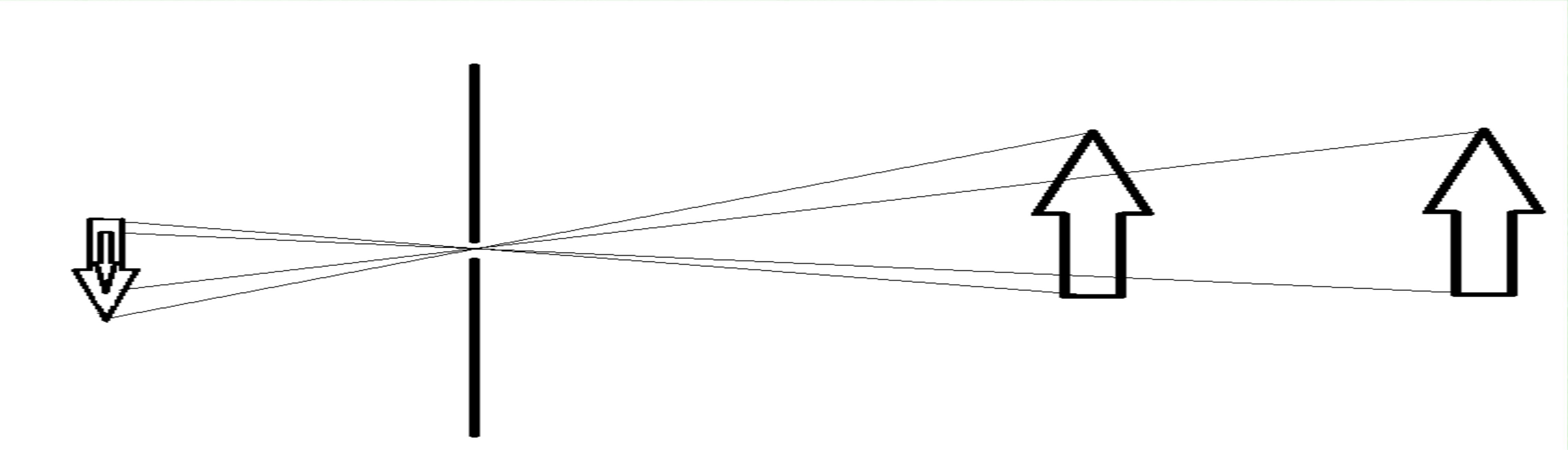
- Light spread out as it goes through small openings
- This happens when the camera aperture is too small (diffraction limited)
- It leads to constructive/destructive interference of light waves (the Airy disk effect)





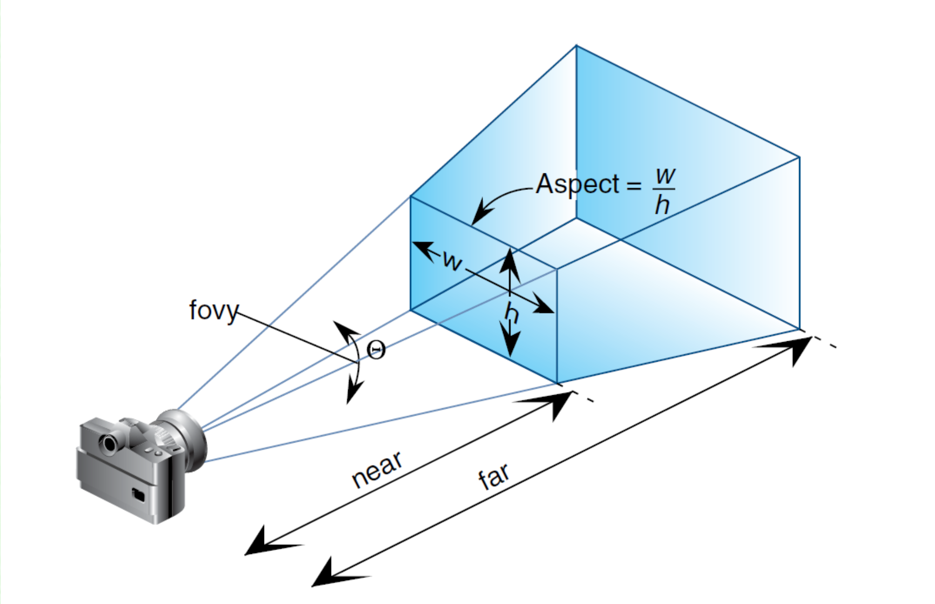
Pinhole Camera (a theoretical approximation)

- Light leaving any point travels in straight lines
- We only care about the lines that hit the pinhole (a single point)
  - Using a single point gives infinite depth of field (everything is in focus, no blurring)
- An upside-down image is formed by the intersection of these lines with an image plane
- More distant objects subtend smaller visual angles and appear smaller
- Objects occlude objects behind them



Virtual Camera

- Trick: Move the film out in front of the pinhole, so that the image is not upside down
- Only render (compute an image for) objects further away from the camera than the film plane
- Add a back clipping plane for efficiency
- Volume between the film (front clipping plane) and the back clipping plane is the viewing frustum (shown in blue)
  - Make sure that the near/far clipping planes have enough space between them to contain the scene
  - Make sure objects are inside the viewing frustum
  - Do not set the near clipping plane to be at the camera aperture!



Camera Distortion depends on Distance

- Do not put the camera too close to objects of interest!
  - Significant/severe deductions for poor camera placement, fisheye, etc. (because the image looks terrible)
- Set up the scene like a real-world scene!
- Get very familiar with the virtual camera!


@160CM


@25CM

Eye Distortion?

- Your eye also has distortion

- Unlike a camera, you don’t actually see the signal received on the cones
- Instead, you perceive an image (highly) processed by your brain
- Your eyes constantly move around obtaining multiple images for your brain to work with
- You have two eyes, and see two images (in stereo), so triangulation can be used to estimate depth and to undo distortion

- If your skeptical about all this processing, remember that your eye sees this:

Dealing with Objects

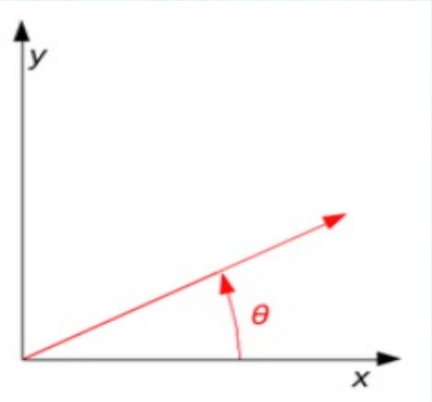
- Let's start with a single 3D point $\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ and move it around in the virtual world
- An object is just a collection of points, so methods for handling a single point extend to handling entire objects

- Typically, objects are created in a reference space, which we refer to as <u>object space</u>
- After creation, we place objects into the scene, which we refer to as <u>world space</u>
- This may require rotation, translation, resizing of the object

- When taking a (virtual) picture, points on the object are projected onto the 2D film plane, which we refer to as <u>screen space</u>
- Unlike rotation/translation/resizing, the projection onto screen space is highly nonlinear and the source of undesirable distortion

Rotation

- Given a 3D point, $\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$
- In 2D, one can rotate a point counter-clockwise about the origin via:



$$\begin{pmatrix} x^{new} \\ y^{new} \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = R(\theta) \begin{pmatrix} x \\ y \end{pmatrix}$$

- This is equivalent to rotating a 3D point around the z-axis using (i.e. multiplying by):

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation

- To rotate a 3D point around the x-axis, y-axis, z-axis (respectively), multiply by:

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Matrix multiplication doesn't commute, i.e. $AB \neq BA$, so the order of rotations matters!
- Rotating about the x-axis and then the y-axis, $R_y(\theta_y)R_x(\theta_x)\vec{x}$, is different than rotating about the y-axis and then the x-axis, $R_x(\theta_x)R_y(\theta_y)\vec{x}$
  - $R_y(\theta_y)R_x(\theta_x)\vec{x} \neq R_x(\theta_x)R_y(\theta_y)\vec{x}$ because $R_y(\theta_y)R_x(\theta_x) \neq R_x(\theta_x)R_y(\theta_y)$

Line Segments are Preserved

- Consider two points $\vec{p}$ and $\vec{q}$ and the line segment between them:

$$\vec{u}(\alpha) = (1 - \alpha)\vec{p} + \alpha\vec{q}$$

- $\vec{u}(0) = \vec{p}$ and $\vec{u}(1) = \vec{q}$, and $0 \leq \alpha \leq 1$ specifies all the points on the line segment

- Multiplying points on the line segment by a rotation matrix R gives:

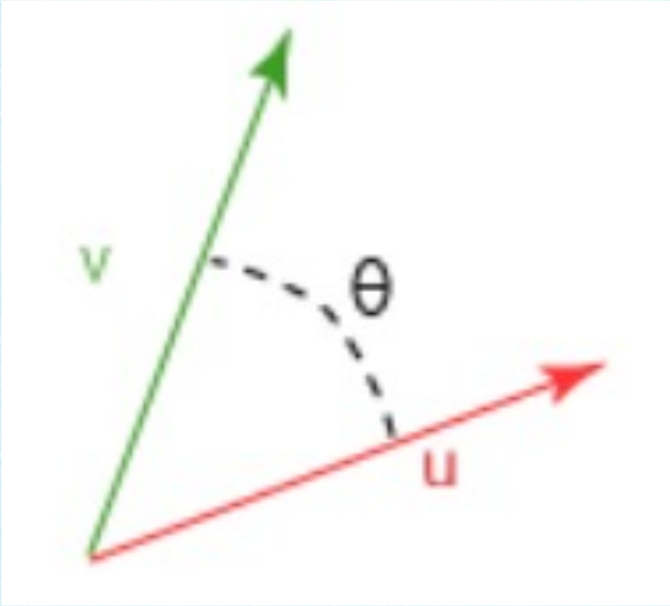
$$R\vec{u}(\alpha) = R\big((1 - \alpha)\vec{p} + \alpha\vec{q}\big) = (1 - \alpha)R\vec{p} + \alpha R\vec{q}$$

- $R\vec{u}(0) = R\vec{p}$ and $R\vec{u}(1) = R\vec{q}$, and $0 \leq \alpha \leq 1$ specifies all the points connecting $R\vec{p}$ and $R\vec{q}$
  - i.e., only need to rotate the endpoints in order to construct the new line segment (connecting them)

- $\|R\vec{p}_1 - R\vec{p}_2\|_2^2 = \|R(\vec{p}_1 - \vec{p}_2)\|_2^2 = (\vec{p}_1 - \vec{p}_2)^T R^T R(\vec{p}_1 - \vec{p}_2) = \|\vec{p}_1 - \vec{p}_2\|_2^2$ shows that the distance between two rotated points is equivalent to the distance between the two original (un-rotated) points

Angles are Preserved

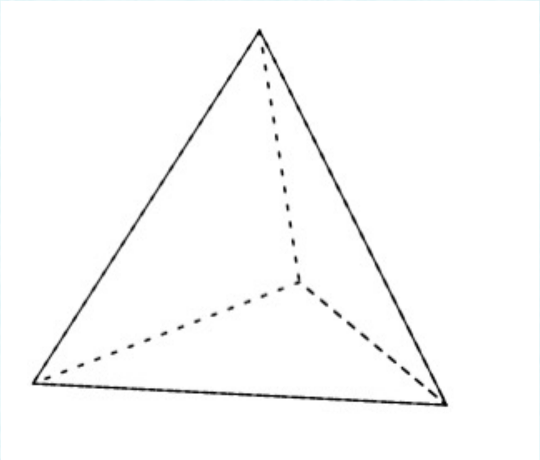
- Consider two line segments $\vec{u}$ and $\vec{v}$ with $\vec{u} \cdot \vec{v} = \|\vec{u}\|_2 \, \|\vec{v}\|_2 \cos(\theta)$ where θ is the angle between them



- $R\vec{u} \cdot R\vec{v} = \|R\vec{u}\|_2 \, \|R\vec{v}\|_2 \cos(\hat{\theta})$
- $R\vec{u} \cdot R\vec{v} = \vec{u}^T R^T R \vec{v} = \vec{u}^T \vec{v} = \|\vec{u}\|_2 \, \|\vec{v}\|_2 \cos(\theta) = \|R\vec{u}\|_2 \, \|R\vec{v}\|_2 \cos(\theta)$
- So, the angle θ between $\vec{u}$ and $\vec{v}$ is the same as the the angle $\hat{\theta}$ between $R\vec{u}$ and $R\vec{v}$

Shape is Preserved

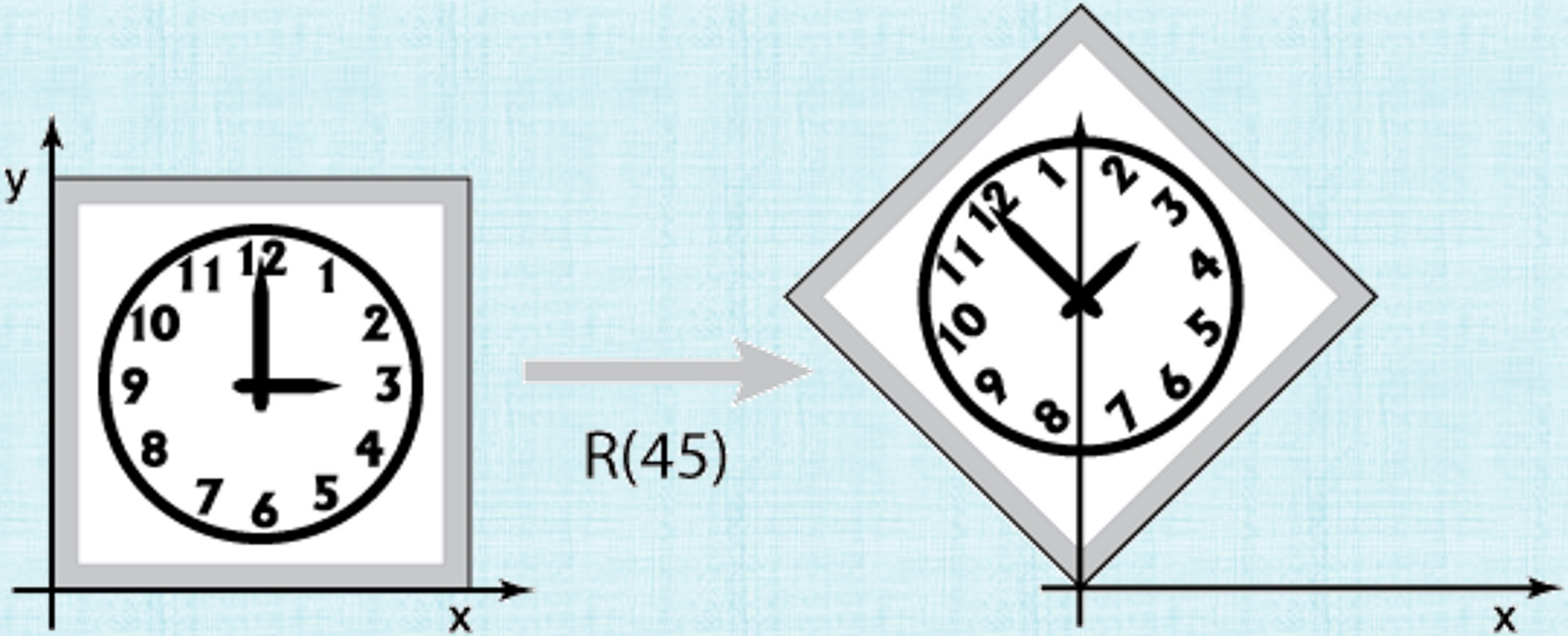
- In continuum mechanics, material deformation is measured by a strain tensor
- The six unique entries in the nonlinear Green strain tensor are computed by comparing an undeformed tetrahedron to its deformed counterpart
- Given a tetrahedron in 3D, it is fully determined by one point and three line segments (the dotted lines in the figure)



- The 3 lengths of these three line segments and the 3 angles between any two of them are used to compare the undeformed tetrahedron to its deformed counterpart
- Since we proved these were all identical under rotations, rotations are shape preserving

Shape is Preserved

- Thus, we can rotate entire objects without changing them

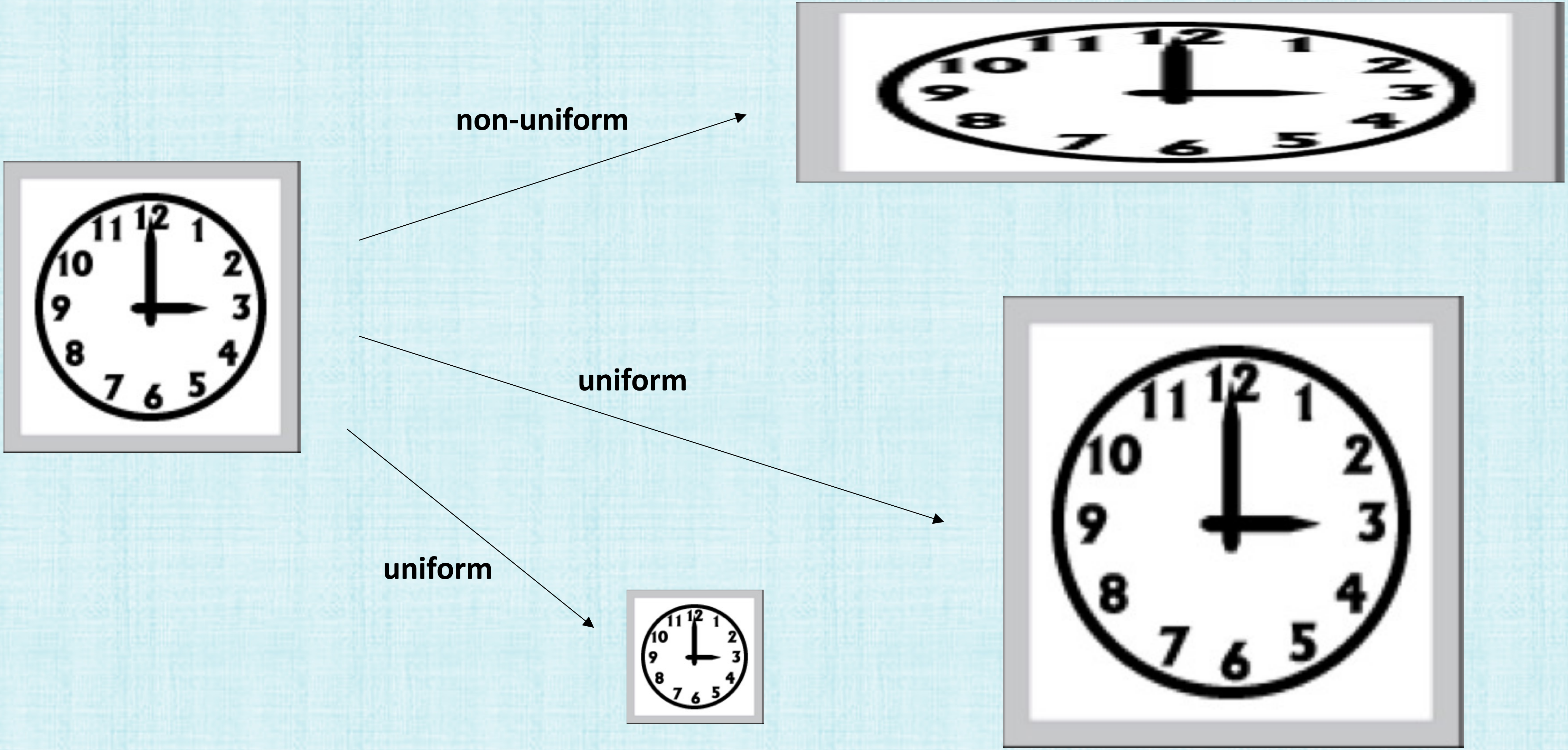


Scaling (or Resizing)

- A scaling matrix $S = \begin{pmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{pmatrix}$ can both scale and shear the object
  - Shearing changes lengths/angles creating significant distortion

- When $s_1 = s_2 = s_3$, then $S = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{pmatrix} = sI$ is pure scaling
- The distributive law of matrix multiplication (again) guarantees that line segments map to line segments
- $\|S\vec{p}_1 - S\vec{p}_2\|_2^2 = s\|\vec{p}_2 - \vec{p}_2\|_2^2$ implies that the distance between scaled points is increased/decreased by a factor of s
- $S\vec{u} \cdot S\vec{v} = \mathrm{s}^2\vec{u} \cdot \vec{v} = s^2\|\vec{u}\|_2 \, \|\vec{v}\|_2 \cos(\theta) = \|S\vec{u}\|_2 \, \|S\vec{v}\|_2 \cos(\theta)$ shows that angles between line segments are preserved
- Thus, uniform scaling grows/shrinks objects proportionally (they are mathematically <u>similar</u>)

Scaling (or Resizing)



Homogenous Coordinates

- In order to use matrix multiplication for transformations, homogeneous coordinates are required
- The homogeneous coordinates of a 3D point $\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ are $\vec{x}_H = \begin{pmatrix} xw \\ yw \\ zw \\ w \end{pmatrix}$ for any $w \neq 0$
- Dividing homogenous coordinates by the fourth component (i.e. w) gives $\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$ or $\begin{pmatrix} \vec{x} \\ 1 \end{pmatrix}$
- 3D points are converted to $\vec{x}_H = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$, with $w = 1$, to deal with translations
- Vectors $\vec{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$ have homogenous coordinates $\vec{u}_H = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ 0 \end{pmatrix}$ or $\begin{pmatrix} \vec{u} \\ 0 \end{pmatrix}$

Homogenous Coordinates

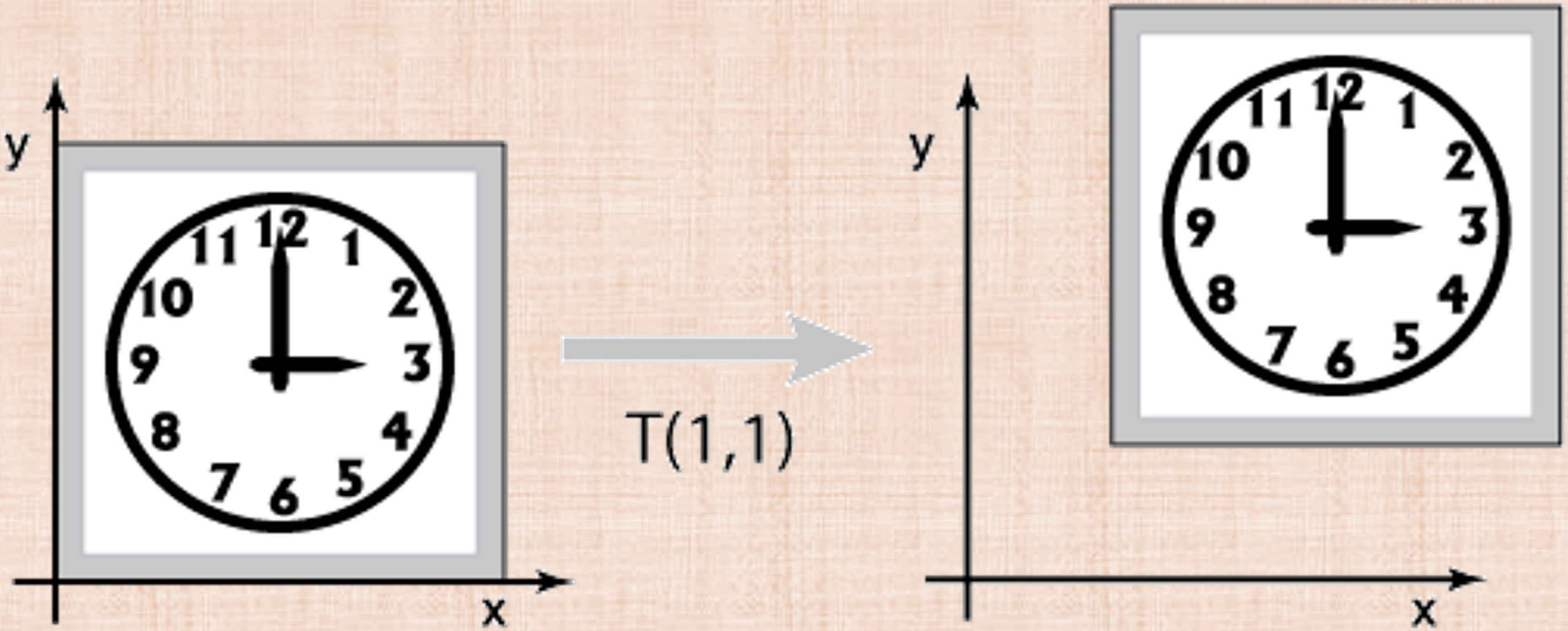
- Let M_{3x3} be a 3x3 rotation or scaling matrix (as discussed previously)
- The transformation of a point $\vec{x}$ is given by $M_{3x3}\vec{x}$
- To obtain the same result for $\begin{pmatrix} \vec{x} \\ 1 \end{pmatrix}$, use a 4x4 matrix $\begin{pmatrix} & M_{3x3} & & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} M_{3x3}\vec{x} \\ 1 \end{pmatrix}$
- Similarly, for a vector $\begin{pmatrix} & M_{3x3} & & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ 0 \end{pmatrix} = \begin{pmatrix} M_{3x3}\vec{u} \\ 0 \end{pmatrix}$

Translation

- To translate a point $\vec{x}$ by $\vec{t} = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}$, multiply $\begin{pmatrix} & I_{3x3} & & \begin{matrix} t_1 \\ t_2 \\ t_3 \end{matrix} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \vec{x} + \vec{t} \\ 1 \end{pmatrix}$
  - $I_{3x3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ is the 3x3 identity matrix
- For a vector $\begin{pmatrix} & I_{3x3} & & \begin{matrix} t_1 \\ t_2 \\ t_3 \end{matrix} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ 0 \end{pmatrix} = \begin{pmatrix} \vec{u} \\ 0 \end{pmatrix}$ has no effect (as desired)
- Translation preserves line segments and the angles between them (and thus preserves shape)

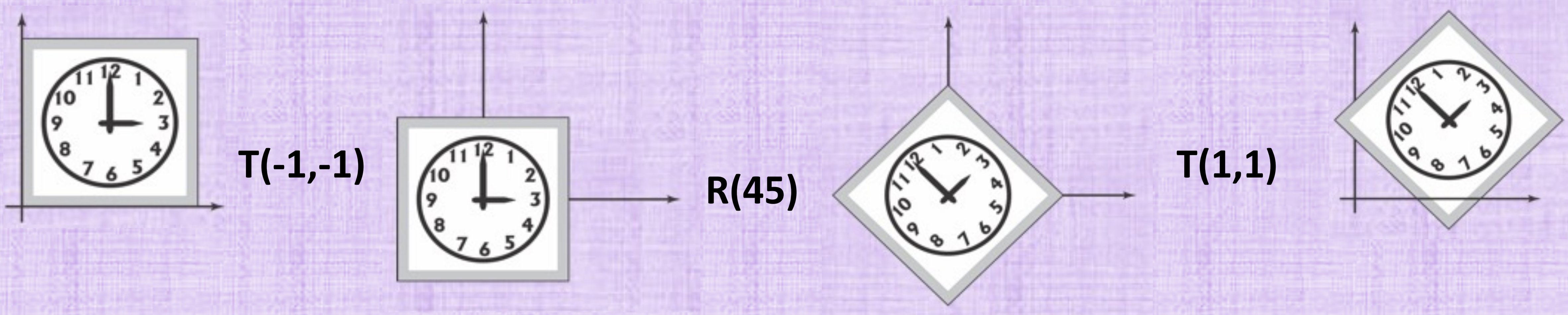
Shape is Preserved

- We can translate entire objects without changing them

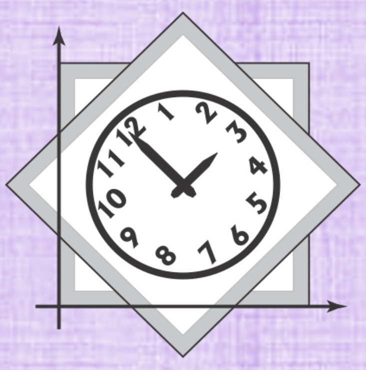


Composite Transforms

- Rotate 45 degrees about the point (1,1)

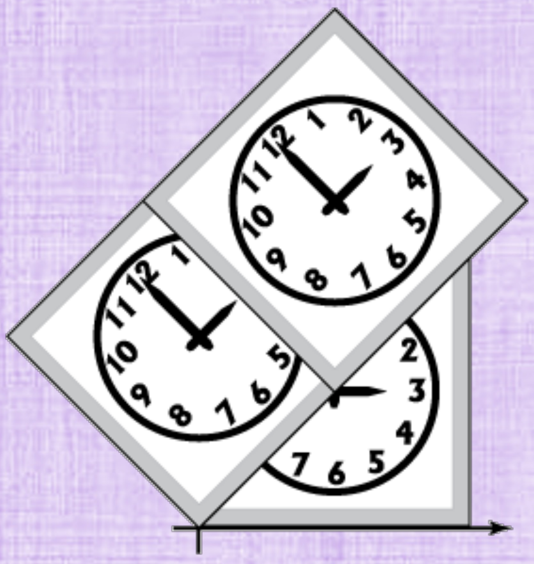


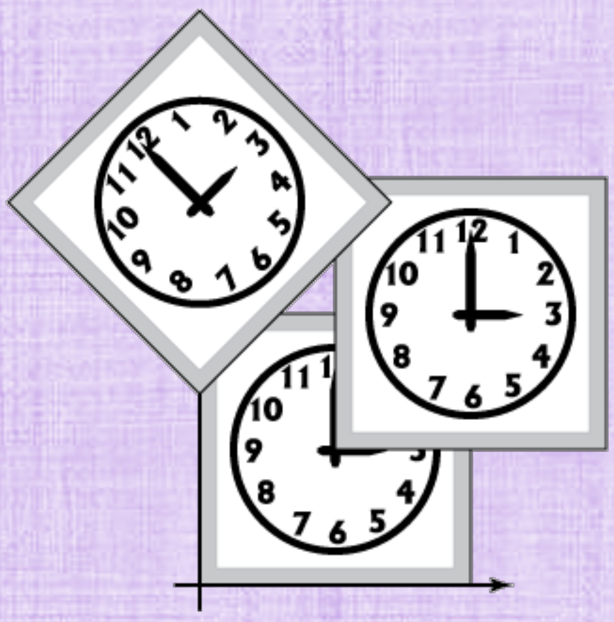
- These transformations can be multiplied together to get a single matrix M=T(1,1)R(45)T(-1,-1) that can be used to multiply every relevant point in the (entire) object:



Order Matters

- Matrix multiplication does not commute: $AB \neq BA$
- The rightmost transform is applied to the points first

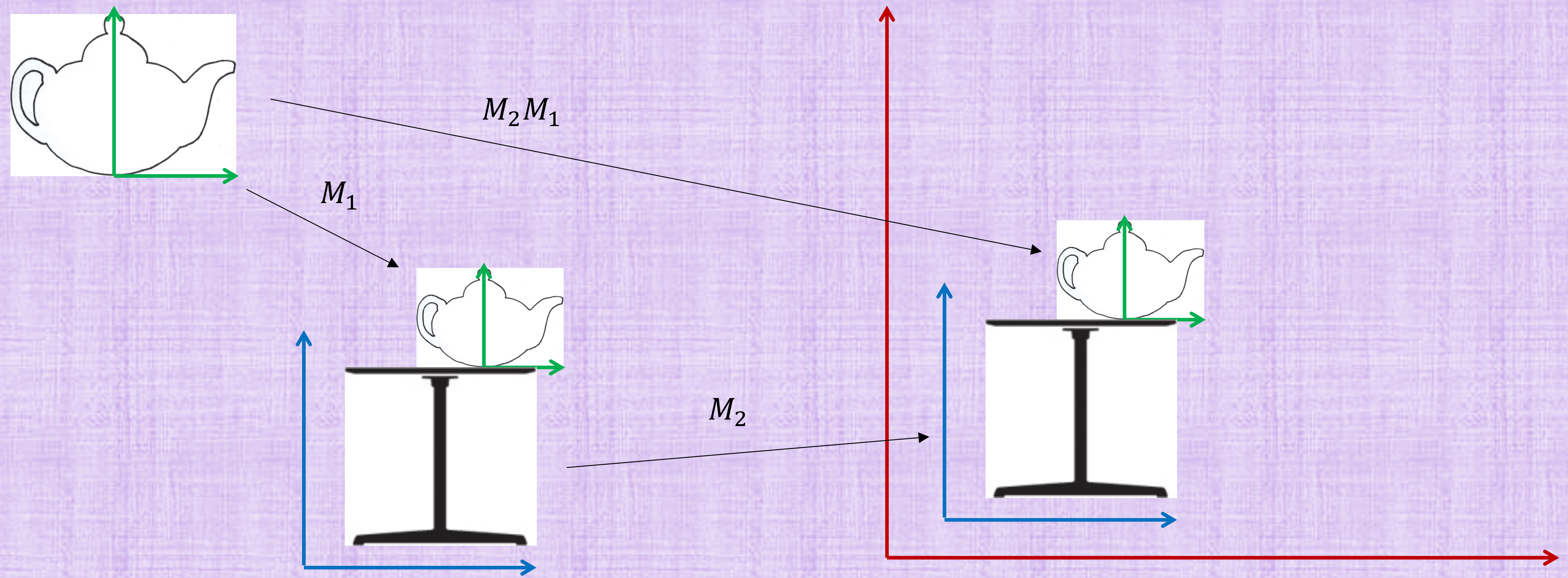




T(1,1)R(45) $\neq$ **R(45)T(1,1)**

Hierarchical Transforms

- M_1 transforms the teapot from its object space to the table's object space (puts it on the table)
- M_2 transforms the table from its object space to world space
- M_2M_1 transforms the teapot from its object space to world space (and onto the table)



Using Transformations

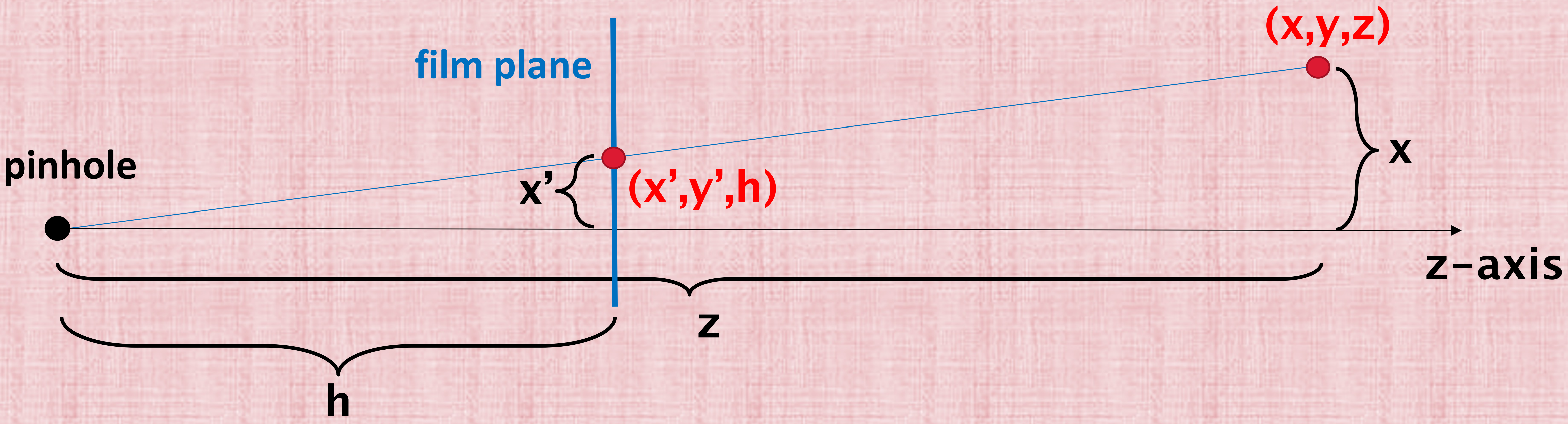
- Create objects (or parts of objects) in convenient coordinate systems
- Assemble objects from their parts (using transformations)
- Transform the assembled object into the scene (via hierarchical transformations)
- Can make multiple copies (even of different sizes) of the same object (simply) by adding another transform stack (efficiently avoiding the creation of a new copy of the object)

- <u>Helpful Hint</u>: Always compute composite transforms for objects or sub-objects, and apply the single composite transform to all relevant points (it's a lot faster)

- <u>Helpful Hint</u>: Orientation is best done first:
  - Place the object at the center of the target coordinate system, and rotate it into the desired orientation
  - Afterwards, translate the object to the desired location

Screen Space Projection

- Projecting geometry from world space into screen space can create significant distortion
- This is because $\frac{1}{z}$ is highly nonlinear



$$\frac{x}{z} = \frac{x'}{h} \Rightarrow x' = h\frac{x}{z} \quad \text{and} \quad \frac{y}{z} = \frac{y'}{h} \Rightarrow y' = h\frac{y}{z}$$

Matrix Form

- Writing the screen space result as $\begin{pmatrix} x'w' \\ y'w' \\ z'w' \\ w' \end{pmatrix}$ gives the desired $\frac{1}{z}$ after dividing by $w' = z$

- Consider: $\begin{pmatrix} x'w' \\ y'w' \\ z'w' \\ w' \end{pmatrix} = \begin{pmatrix} h & 0 & 0 & 0 \\ 0 & h & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$

- This has $w' = z$, $x'w' = hx$ or $x' = \frac{hx}{z}$, and $y'w' = hy$ or $y' = \frac{hy}{z}$ (as desired)

- Homogenous coordinates allows the <u>nonlinear</u> $\frac{1}{z}$ to be expressed with <u>linear</u> matrix multiplication (so it can be added to the matrix multiplication stack!)

Perspective Projection

- The third equation is $z'w' = az + b$ or $z'z = az + b$
- New z values aren't required (projected points all lie on the $z = h$ image plane)
- However, computing z' as a monotonically increasing function of z allows it to be used to determine occlusions (for alpha channel transparency)

- The near ($z = n$) and far ($z = f$) clipping planes are preserved via $z' = n$ and $z' = f$
- 2 equations in 2 unknowns ($n^2 = an + b$ and $f^2 = af + b$); so, $a = n + f$ and $b = -fn$

- This transforms the viewing frustum into an orthographic volume in screen space