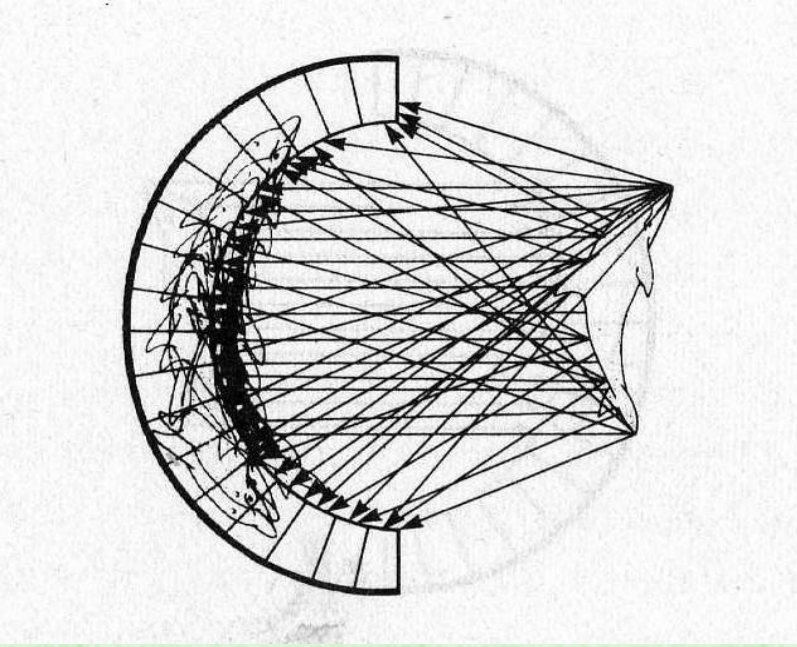
The Virtual World

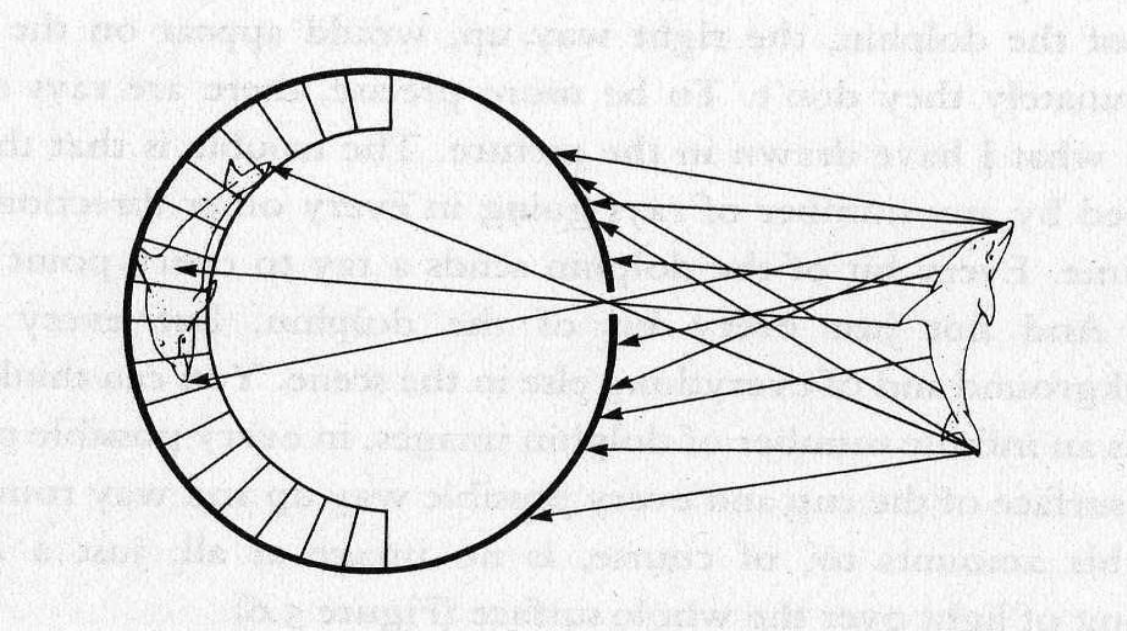
Building a Virtual World

- Goal: mimic human vision in a virtual world (with a computer)
  - Cheat for efficiency, using knowledge about light physics and human perception (e.g. from lecture 2)
- Create a virtual camera: place it somewhere and point it at something
- Put film (containing pixels, each with RGB values ranging from 0-255) into the camera
- Place objects into the world, including a floor/ground, walls, ceiling/sky, etc.
  - Two step process: (1) make objects, (2) place objects (transformations)
  - Making objects is itself a two-step process: (1a) build geometry (geometric modeling), (1b) paint geometry (texture mapping)
- Put lights into the scene (so 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
  - Taking a picture creates film data as the final image
  - We will consider two methods (scanline rendering and ray tracing) for taking a picture

Pupil

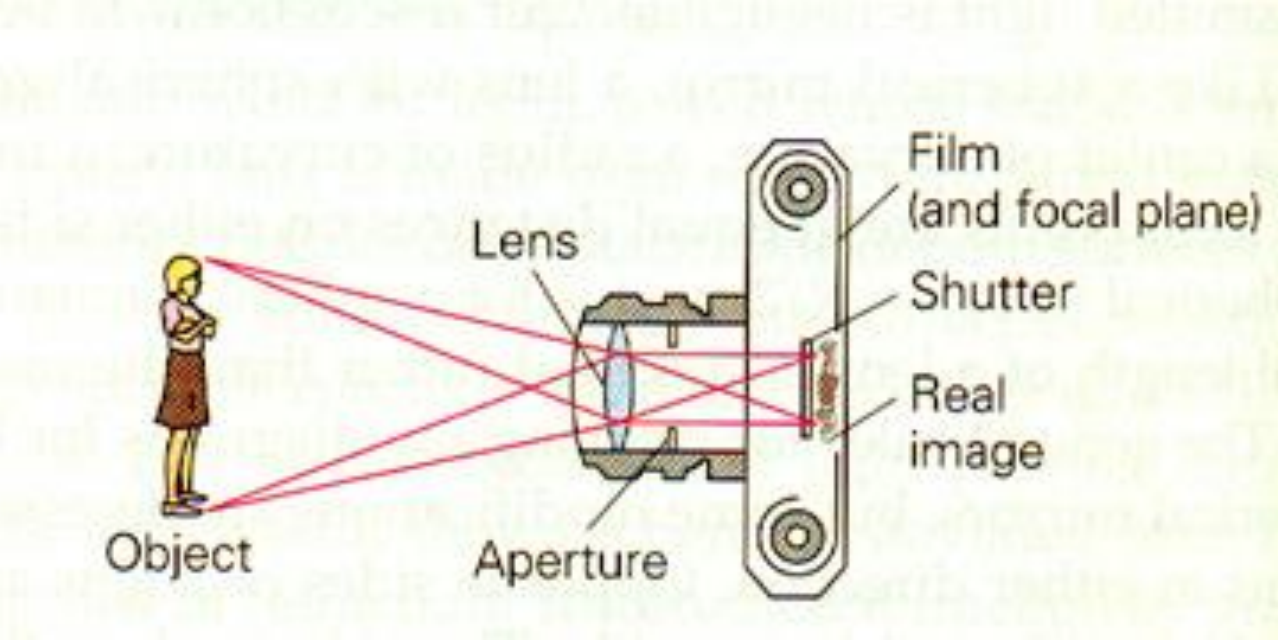
- Light emanates from every point of an object outwards in every direction
  - That's why we can all see the same spot on the same object
  - Light leaving that spot/point (on the object) is entering each of our eyes
- Without a pupil, light from every point on an object would hit the same cone on our 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

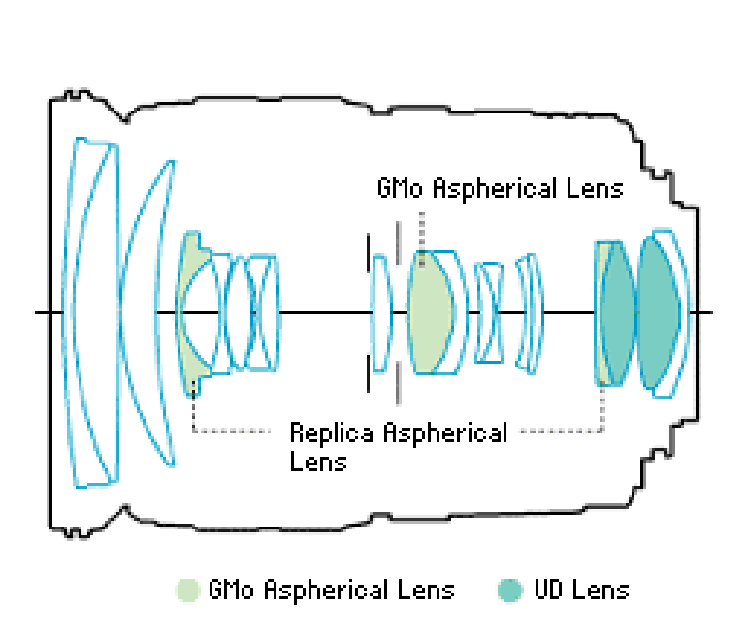




Aperture

- 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



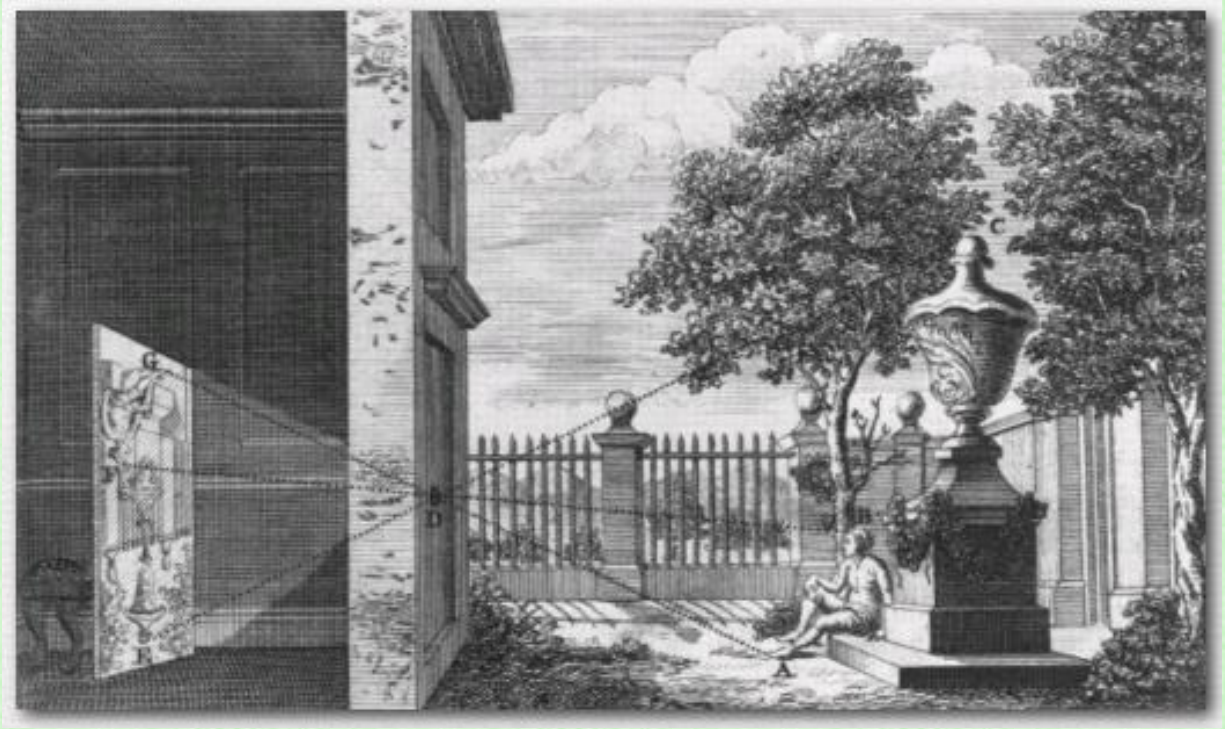


Aside: Lens Flare

- Many camera complexities are (typically) not properly accounted for in virtual worlds
- Thus, certain effects (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 that reflects and scatters light
  - It depends on material inhomogeneities in the lenses, the imperfect geometry of lens surfaces, absorption/dispersion of lenses, antireflective coatings, diffraction, etc.

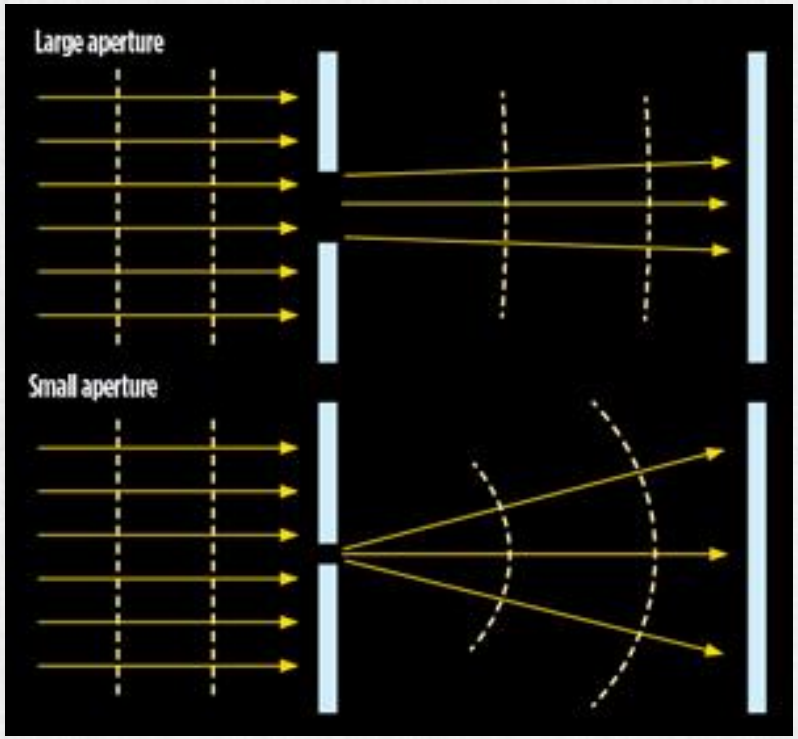
Pupil/Aperture Size

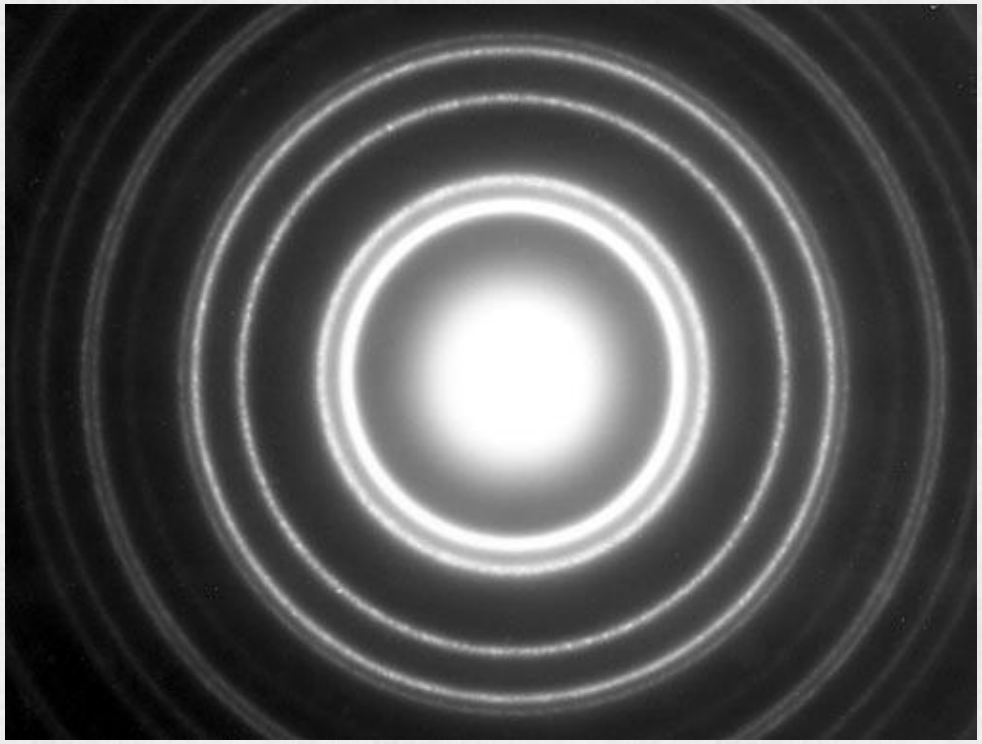
- The pupil/aperture has a finite size, big enough for light to pass through it
- When too small, not enough light enters and the image is too dark/noisy
  - In addition, light can diffract (instead of traveling in straight lines) distorting the image
- When too large, light from a large area of an object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



Aside: Diffraction

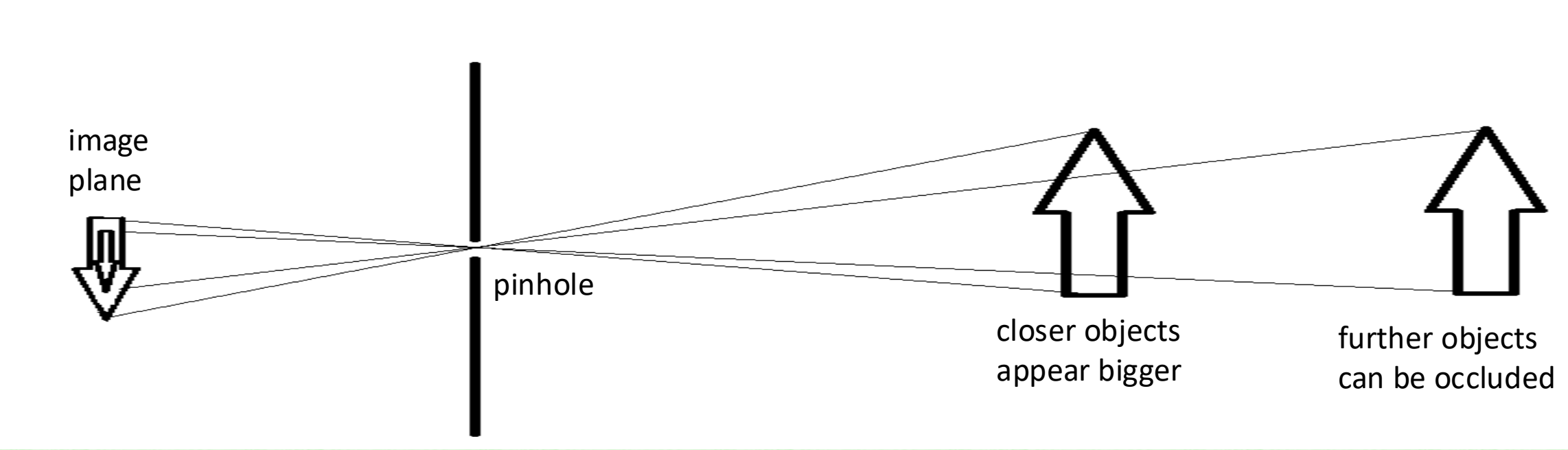
- Light spread out as it goes through small openings
- This happens when the camera aperture is too small (diffraction limited)
- It leads to constructive/destructive interference of light waves (the Airy disk effect)





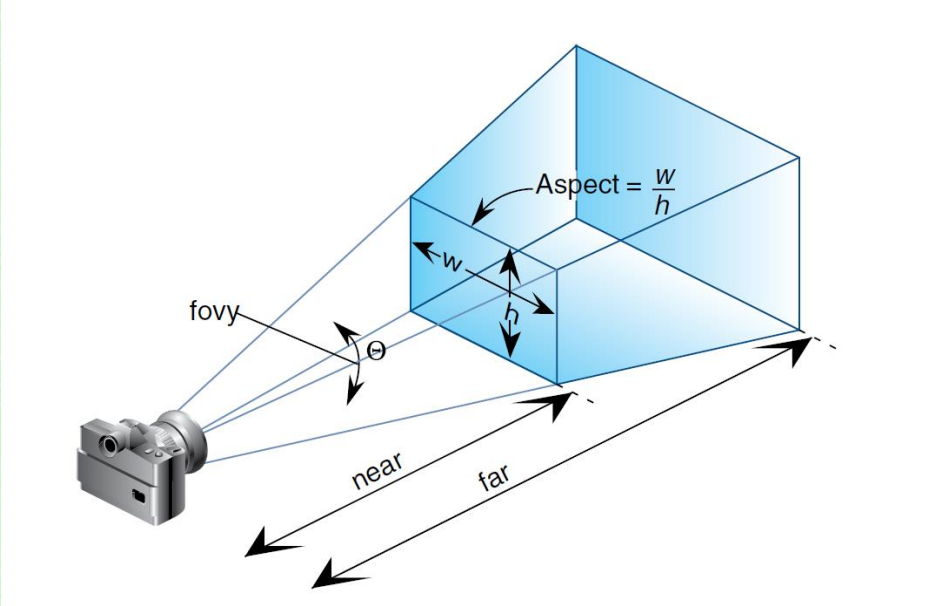
Pinhole Camera

- 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

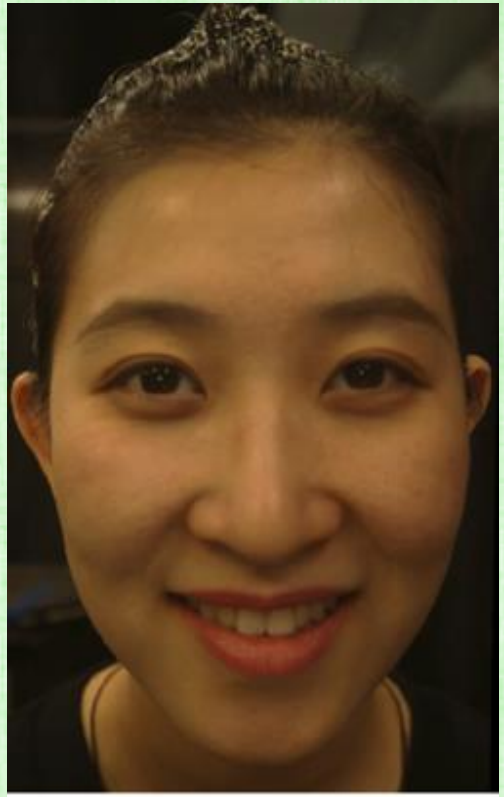
- Trick: Move the film out in front of the pinhole, so that the image is not upside down
- Only render (compute an image for) objects further away from the camera than the film plane
- Add a back clipping plane, for efficiency
- The volume between the film (front clipping plane) and the back clipping plane is called 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 grade deductions for poor camera placement, fisheye, etc. (because distortion looks bad)
- 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 by 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 cones see this:

Dealing with Objects

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

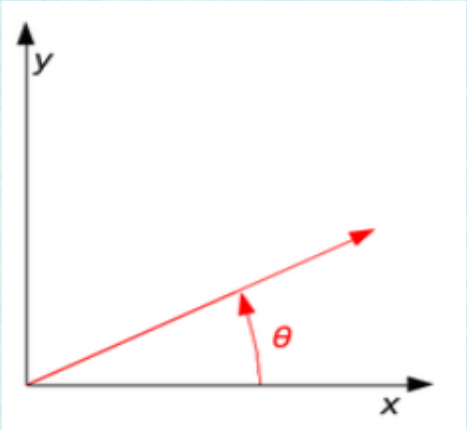
- Objects are created in a reference space, which we refer to as object space
- After creation, we place objects into the virtual scene, which we refer to as world space
- This may require rotation, translation, resizing of the object

- Taking a (virtual) picture projects the object's points onto the 2D film plane, which we refer to as screen space
- Unlike rotation/translation/resizing, projection onto screen space is highly nonlinear and a source of distortion

Rotation

- Given a 3D point, $\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$
- In 2D, 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 is different than rotating about the y-axis and then the x-axis:
  - $R_y(\theta_y)R_x(\theta_x)\vec{x} \neq R_x(\theta_x)R_y(\theta_y)\vec{x}$ 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 each point on the line segment is specified by some $\alpha \in [0,1]$

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

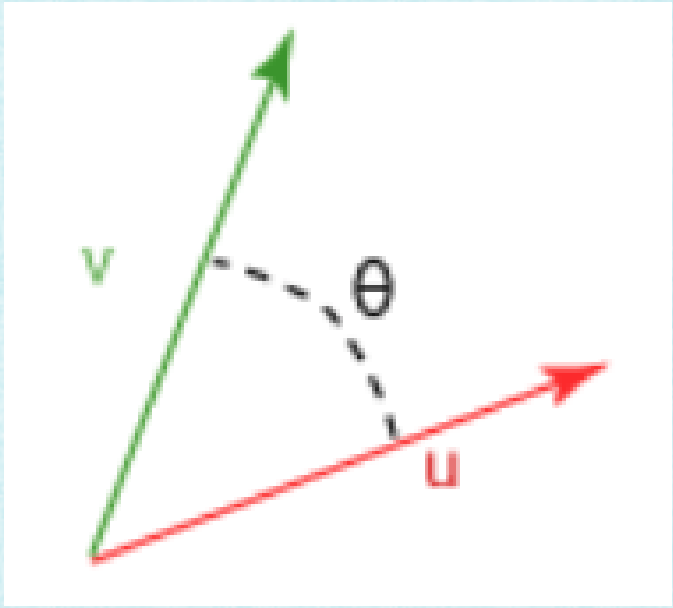
$$R\vec{u}(\alpha) = (1-\alpha)R\vec{p} + \alpha R\vec{q}$$

- So, that point lives on the line segment between $R\vec{p} = R\vec{u}(0)$ and $R\vec{q} = R\vec{u}(1)$; in fact, it's the same distance from $R\vec{p}$ and $R\vec{q}$ that it is was from $\vec{p}$ and $\vec{q}$
  - 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 rotated endpoints is equivalent to the distance between the original (un-rotated) endpoints

Angles are Preserved

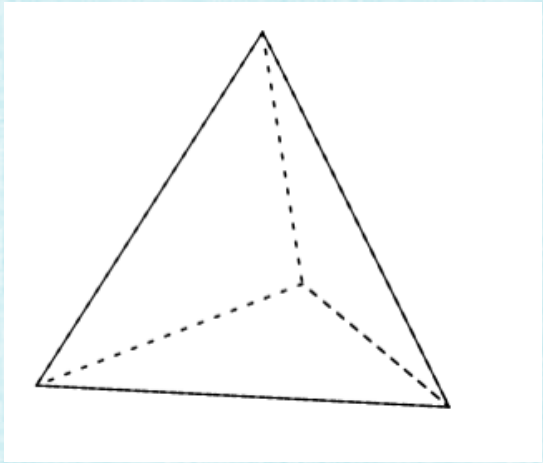
- Consider line segments $\vec{u}$ and $\vec{v}$ with $\vec{u} \cdot \vec{v} = \|\vec{u}\|_2 \, \|\vec{v}\|_2 \cos(\theta)$ where θ is the angle between them



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

Shape is Preserved

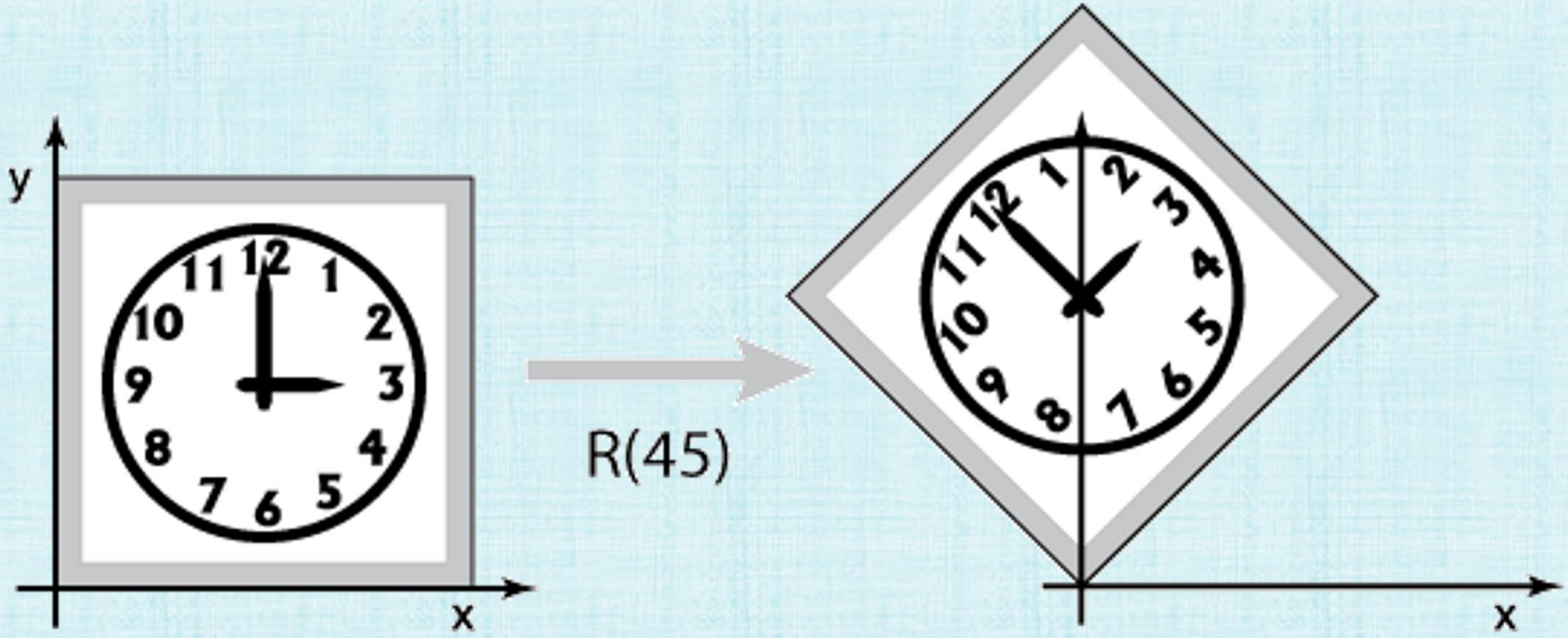
- In continuum mechanics, material deformation is measured by a geometric strain tensor
- The 6 unique entries in the nonlinear Green strain tensor are computed by comparing an undeformed tetrahedron to its deformed counterpart
- Given a tetrahedron in 3D, it is fully determined by one point and 3 line segments (the dotted lines in the figure)



- The 3 lengths of these 3 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 preserve shape

Shape is Preserved

- Thus, we can rotate entire objects without changing them



Scaling/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 an object
  - Shearing changes lengths/angles creating 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
- $\|S\vec{p}_1 - S\vec{p}_2\|_2^2 = s^2\|\vec{p}_2 - \vec{p}_2\|_2^2$ implies that the distance between scaled points is increased/decreased by a factor of s
- $S\vec{u} \cdot S\vec{v} = s^2\vec{u} \cdot \vec{v} = s^2\|\vec{u}\|_2 \, \|\vec{v}\|_2 \cos(\theta) = \|S\vec{u}\|_2 \, \|S\vec{v}\|_2 \cos(\theta)$ shows that angles between line segments are preserved
- Thus, uniform scaling grows/shrinks objects proportionally (they are mathematically similar)

Scaling (or Resizing)

non-uniform

$s_1 \neq s_2$

uniform

$s > 1$

uniform

$0 < s < 1$

Homogenous Coordinates

- Homogeneous coordinates allow translation to use matrix multiplication (instead of addition)

- 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 $\begin{pmatrix} \vec{x} \\ 1 \end{pmatrix}$

- We convert 3D points to $\vec{x}_H = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$, with $w = 1$

- We convert 3D vectors $\vec{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$ to $\vec{u}_H = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ 0 \end{pmatrix}$ or $\begin{pmatrix} \vec{u} \\ 0 \end{pmatrix}$, noting that there is no dividing by w for vectors

Homogenous Coordinates

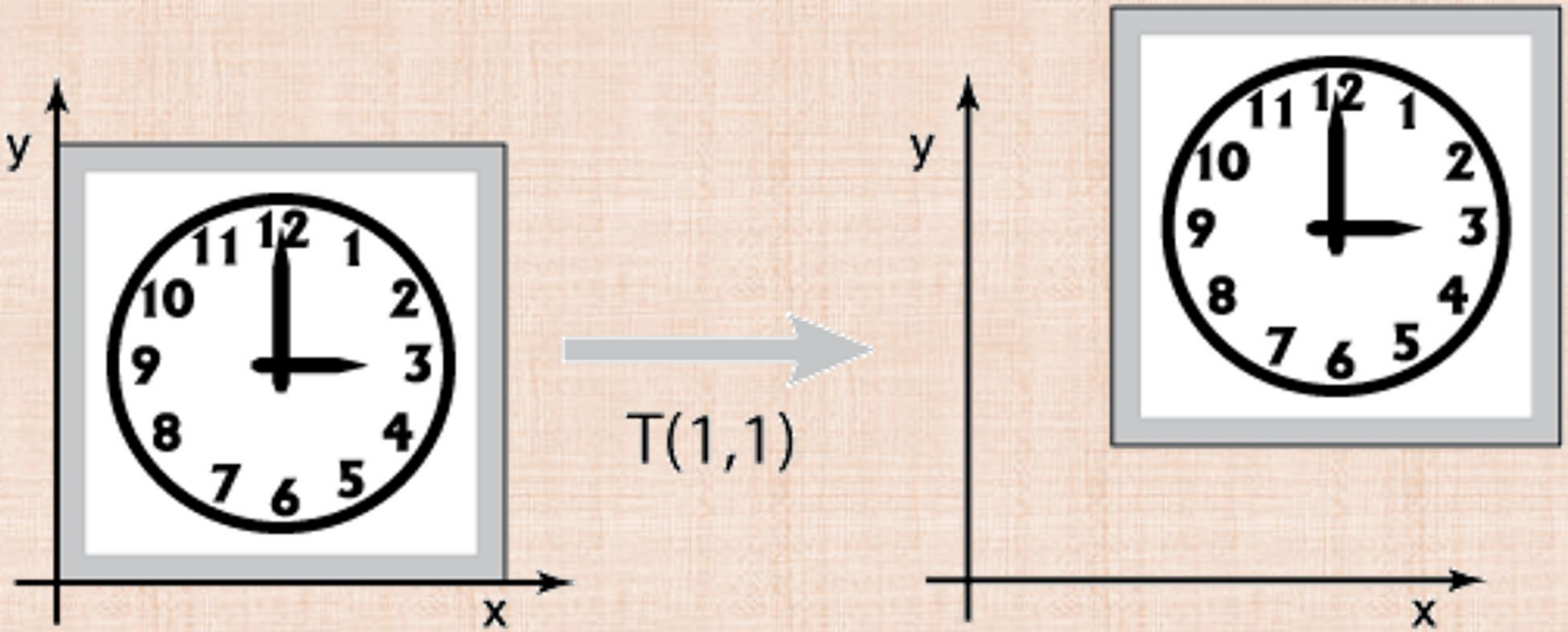
- Let M_{3x3} be a 3x3 rotation or scaling matrix (as discussed previously)
- The transformation of a point $\vec{x}$ is given by $M_{3x3}\vec{x}$
- To obtain the same result for $\begin{pmatrix}\vec{x}\\1\end{pmatrix}$, use a 4x4 matrix $\begin{pmatrix} & M_{3x3} & & \begin{matrix}0\\0\\0\end{matrix}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

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

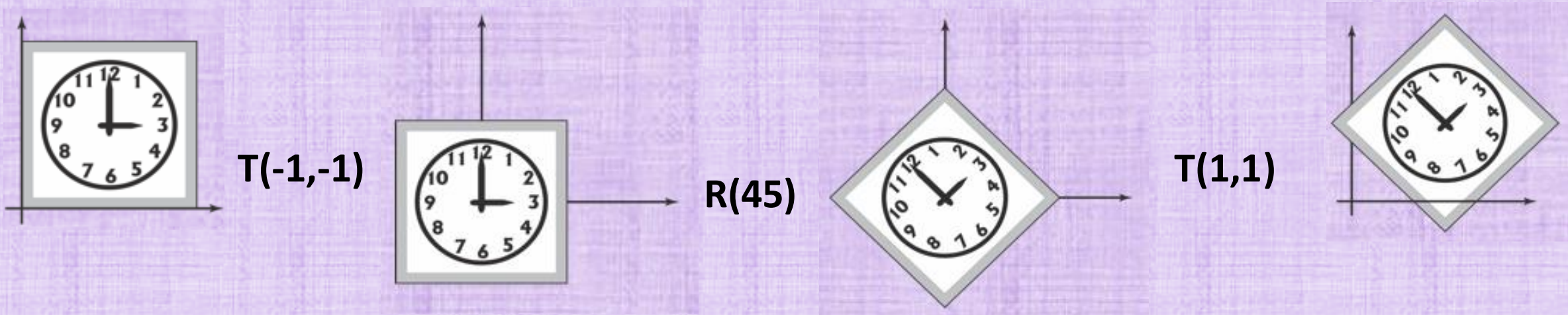
Shape is Preserved

- We can translate entire objects without changing them

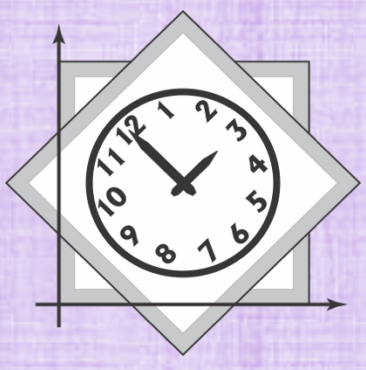


Combining Transforms

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

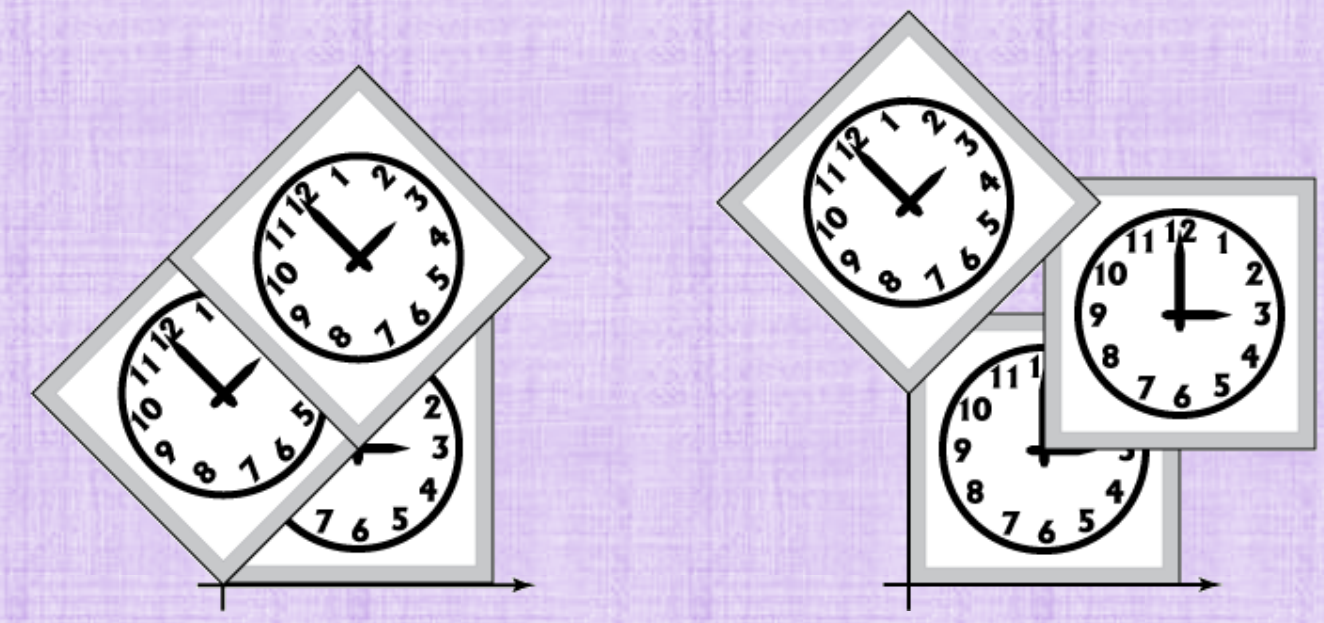


- Multiply these transforms together to get a single matrix M=T(1,1)R(45)T(-1,-1)
- Then, just multiply every relevant point in the (entire) object by M:



Order Matters

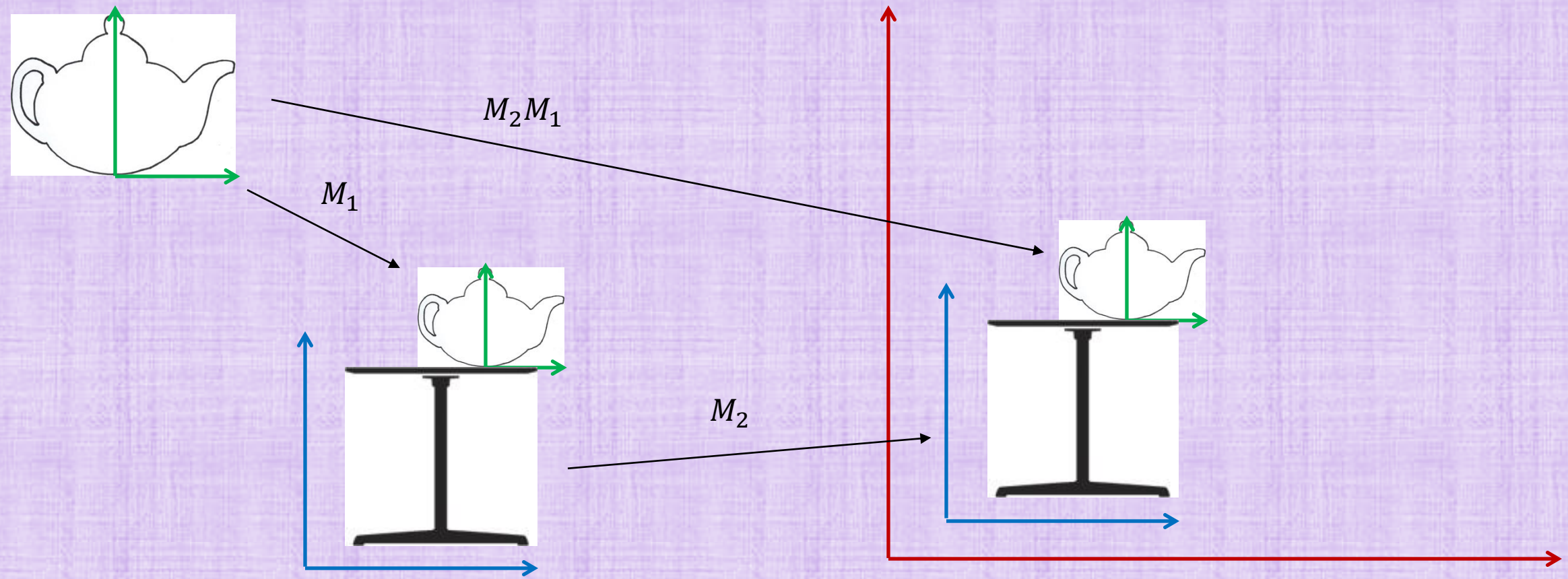
- Matrix multiplication does not commute: $AB \neq BA$
- The rightmost transform is applied to the points first



T(1,1)R(45) **≠** **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

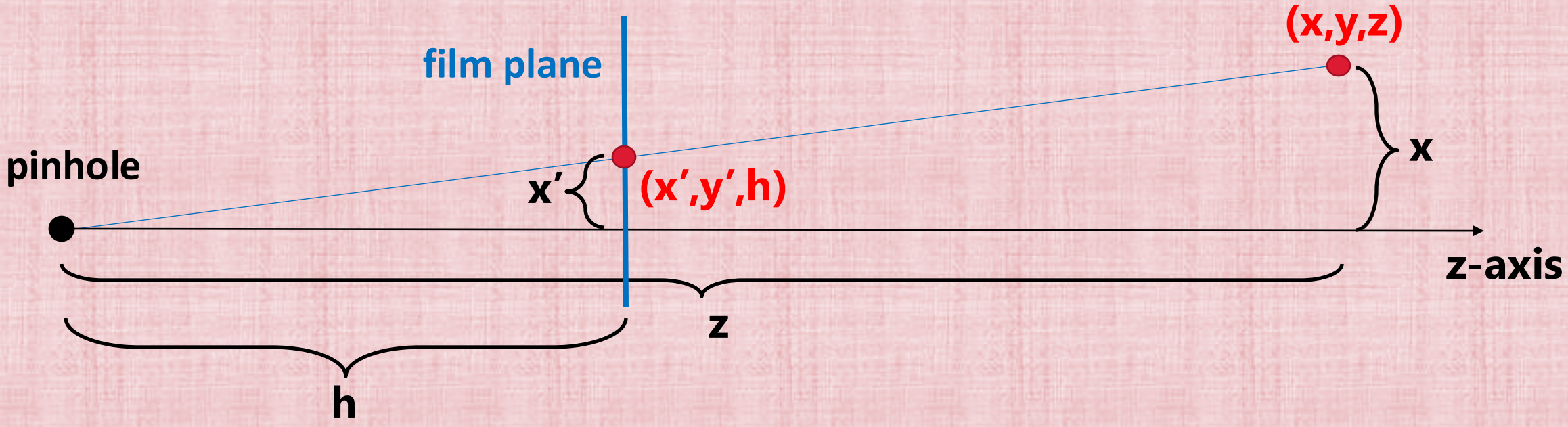
- Create objects (or parts of objects) in convenient modeling coordinate systems
- Assemble objects from their parts (using transformations)
- Transform the assembled object into the scene (via hierarchical transformations)

- Can make multiple copies (even of different sizes) of the same object (simply) by adding another transform stack (avoiding the creation of a new copy of the object geometry)

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

- Helpful Hint: Orientation is best done first:
  - Place an 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



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

- $\frac{1}{z}$ is highly nonlinear, so projection from world space into screen space can create significant distortion

Matrix Form

- Writing the screen space result as $\begin{pmatrix} x'w' \\ y'w' \\ z'w' \\ w' \end{pmatrix}$ gives a $\frac{1}{z}$ 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 via <u>linear</u> matrix multiplication, so the projection can be added to the matrix multiplication stack!

Choosing a and b

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

- The near ($z = n$) and far ($z = f$) clipping planes can be 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 $b = -fn$
- Then, $z' = a + \frac{b}{z} = n + f - \frac{fn}{z}$
- This transforms the viewing frustum into a distorted orthographic volume in screen space