The Virtual World
Building a Virtual World

- Goal: to mimic human vision in a virtual world (with the computer)
- Will cheat a bit for efficiency, using what we know about light and our eye (e.g. from the last lecture)
- Set up a virtual camera somewhere in the virtual world, and point it in some direction
- 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 into scene (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 --- that is, code emits light from virtual light sources, bounces that light off of virtual geometry, and follows that light into the camera and onto the film
  - We will consider both scanline rendering and ray tracing for the taking this picture
Pupil

- Light emanates off every point of an object outwards in every direction
  - That’s why we can all look at the same spot on the same object and see it
  - Light is leaving from that point/spot on the object and entering each of our eyes
- Without a pupil, light from every part of an object would hit the same cone on our eye, blurring the image
- The small pupil restricts the entry of light so that each cone only receives light emanating from a small region on the object, giving interpretable spatial detail
Aperture

- Digital cameras are similar to the human eye, except with mechanical as opposed to biological components
- Instead of cones, the camera has pixels
- Instead of a pupil, the camera has a small (adjustable) aperture that light passes through
- The camera also typically has a hefty/complex lens system
Aside: Lens Flare

• Many digital camera complexities are not often accounted for in virtual worlds
• Thus, some complex effects like depth of field, motion blur, chromatic aberration, lens flare, etc. have to be modeled in other ways (as we will see later)
• Particularly complex is lens flare, which is caused by light being reflected/scattered by lenses in the lens system
• This is caused in part by material inhomogeneities in the lens, and depends on the geometry of lens surfaces and characteristic planes, absorption/dispersion of lens elements, antireflective coatings, diffraction, etc.
Pinhole Camera

• The pupil/aperture has to have a finite size in order for light to get through
  • If 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
  • A larger pupil/aperture allows light from a larger area of an object to hit the same cone causing some amount of blurring
• In our virtual world, we have the luxury of using a single point for the aperture, without worrying about dark or distorted imagery
Aside: Diffraction

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

- Light leaving any point travels in straight lines
- We only care about the lines that hit the pinhole (a single point)
- Infinite depth of field; i.e., 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 World Camera

- 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
- 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 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 at the camera aperture!
Cameras 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 will look terrible)
- Set up the scene like a real world scene!
- Get very familiar with the virtual camera!
Eye Distortion?

• Your eye has the same distortion, or even worse because it has slightly curved “film”

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

• If your skeptical about all this processing, remember 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 learn to move it around in the virtual world

• An object is just an infinite collection of points, and as such the methods for handling a single point readily extend to handling entire objects

• Typically objects are created in a reference space, which we refer to as object space
• After creation, we place objects into the scene, which we refer to as world space
• This often requires rotation, translation, resizing of the object

• Finally, when taking a picture, the points on the object are projected onto the film, which we refer to as screen space
• Unlike rotation/translation/resizing, this projection onto screen space is highly nonlinear and the source of undesirable distortions
Rotation

- Let’s start with a single 3D point \( \vec{x} = (\hat{x}, \hat{y}, \hat{z}) \) and learn to move it around in the virtual world.
- In 2D, one can rotate a point clockwise about the origin as follows:

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

- This is equivalent to rotating any 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

- In order to rotate a 3D point around the x-axis, y-axis, or z-axis (respectively), one multiplies by:

\[
R_x(\theta) = \begin{pmatrix}
1 & 0 & 0 \\
0 & \cos \theta & -\sin \theta \\
0 & \sin \theta & \cos \theta
\end{pmatrix} \quad R_y(\theta) = \begin{pmatrix}
\cos \theta & 0 & \sin \theta \\
0 & 1 & 0 \\
-\sin \theta & 0 & \cos \theta
\end{pmatrix} \quad R_z(\theta) = \begin{pmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

- 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, \( R_y(\theta_y)R_x(\theta_x)\hat{x} \), gives a different result than rotating about the y-axis and then the x-axis, \( R_x(\theta_x)R_y(\theta_y)\hat{x} \)
- That is, \( R_y(\theta_y)R_x(\theta_x)\hat{x} \neq R_x(\theta_x)R_y(\theta_y)\hat{x} \) 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 \( \hat{p} \) and \( \hat{q} \) and the line segment between them:

\[
\tilde{u}(\alpha) = (1 - \alpha)\hat{p} + \alpha\hat{q}
\]

• Here, \( \tilde{u}(0) = \hat{p} \) and \( \tilde{u}(1) = \hat{q} \) and \( 0 \leq \alpha \leq 1 \) determine all the points on the line segment

• Multiplying points on the line segment by a rotation matrix \( R \) gives:

\[
R\tilde{u}(\alpha) = R((1 - \alpha)\hat{p} + \alpha\hat{q}) = (1 - \alpha)R\hat{p} + \alpha R\hat{q}
\]

• Here, \( R\tilde{u}(0) = R\hat{p} \) and \( R\tilde{u}(1) = R\hat{q} \) and \( 0 \leq \alpha \leq 1 \) determine all the points on the new rotated line segment connecting the rotated points \( R\hat{p} \) and \( R\hat{q} \)

• I.e., one only need rotate the endpoints and construct the new line segment connecting them

• \( \|R\tilde{u}(\alpha) - R\hat{p}\|_2^2 = \|R(\tilde{u}(\alpha) - \hat{p})\|_2^2 = (\tilde{u}(\alpha) - \hat{p})^T R^T R (\tilde{u}(\alpha) - \hat{p}) = \|\tilde{u}(\alpha) - \hat{p}\|_2^2 \) shows that the distance between rotated points is equivalent to the distance between original 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 $\theta$ is the angle between them.

\[
R\vec{u} \cdot R\vec{v} = (R\vec{u})^T R^T R\vec{v} = \vec{u}^T \vec{v} = ||\vec{u}||_2 ||\vec{v}||_2 \cos(\theta) = ||R\vec{u}||_2 ||R\vec{v}||_2 \cos(\theta)
\]
- So, the angle $\theta$ between $\vec{u}$ and $\vec{v}$ is the same as the the angle $\theta$ between $R\vec{u}$ and $R\vec{v}$.
In continuum mechanics, one measures the deformation of a material by a tensor called the strain. 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 has the form $S = \begin{pmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{pmatrix}$ and can both scale and shear the object.
- Generally speaking, shearing an object changes lengths/angles creating significant distortion.
- When $s_1 = s_2 = s_3$, one has pure scaling of the form $S = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{pmatrix} = sI$.
- The distributive law of matrix multiplication guarantees that line segments map to line segments, and $\|S\vec{u}(\alpha) - S\vec{p}\|_2 = s\|\vec{u}(\alpha) - \vec{p}\|_2$ 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, when using uniform scaling, objects grow/shrink but look the same as far as proportions are concerned (they are mathematically similar).
Scaling (or Resizing)

- Non-uniform
- Uniform
- Uniform
Homogenous Coordinates

• In order to deal with transformations via matrix multiplication, one uses homogeneous coordinates.

• The homogeneous coordinates of a 3D point \( \mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \) are \( \mathbf{x}_H = \begin{pmatrix} xw \\ yw \\ zw \\ w \end{pmatrix} \) for any \( w \neq 0 \).

• Dividing any homogenous coordinates by its fourth component gives \( \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \) or \( \begin{pmatrix} \tilde{x} \\ 1 \end{pmatrix} \).

• We convert all our 3D points to the form \( \mathbf{x}_H = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \) where \( w = 1 \) to deal with translations.

• For vectors \( \mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \), the homogenous coordinates are \( \mathbf{u}_H = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ 0 \end{pmatrix} \) or \( \begin{pmatrix} \tilde{u} \\ 0 \end{pmatrix} \).
Homogenous Coordinates

• Let $M_{3\times3}$ be a 3x3 rotation or scaling matrix (as discussed previously)
• Then the transformation of a point $\vec{x}$ is given by $M_{3\times3}\vec{x}$

• To produce the same result for $(\vec{x})_1$, use the 4x4 matrix

$$
\begin{pmatrix}
M_{3\times3} & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix}
= (M_{3\times3}\vec{x})_1
$$

• Similarly, for a vector

$$
\begin{pmatrix}
M_{3\times3} & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2 \\
u_3 \\
0
\end{pmatrix}
= (M_{3\times3}\vec{u})$$
Translation

• To translate a point $\vec{x}$ by some amount $\vec{t} = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}$ one multiplies by a 4x4 matrix

$$\begin{pmatrix} I_{3x3} & t_1 \\ t_2 & t_2 \\ t_3 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \vec{x} + \vec{t} \end{pmatrix}$$

where the 3x3 identity is $I_{3x3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

• For a vector

$$\begin{pmatrix} I_{3x3} & t_1 \\ t_2 & t_2 \\ t_3 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} \vec{u} \\ 0 \end{pmatrix},$$

which has no effect (as desired)

• Translations preserves line segments and angles between them, and thus shapes
Shape is Preserved

- We can translate entire objects without changing them
Composite Transforms

• Suppose one wants to 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
- $M_2$ transforms the table from its object space to world space
- $M_2M_1$ transforms the teapot from its object space to world space
Using Transformations

• Create objects (or parts of objects) in convenient coordinate systems
• Assemble objects from their parts
• Then, transform the assembled object into the scene
• Can make multiple copies (even of different sizes) of the same object (simply) by adding another transform stack (and efficiently, i.e. without creating a new copy of the object)

• Helpful Hint: Always compute composite transforms for objects or sub-objects, and apply the single composite transform to all relevant points (it’s a lot faster)
• Helpful Hint: Orientation is best done first: place the object at the center of the target coordinate system, and rotate it to the desired orientation. Afterwards, translate the object to the correct location.
Screen Space Projection

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

\[
\begin{align*}
\frac{x}{z} &= \frac{x'}{h} \\
x' &= h \frac{x}{z} \\
y' &= h \frac{y}{z}
\end{align*}
\]
Matrix Form

- Express the screen space result in homogeneous coordinates as

\[
\begin{pmatrix}
  x'w' \\
y'w' \\
z'w' \\
w'
\end{pmatrix}
\]

- Setting \( w' = z \) gives the desired \( \frac{1}{z} \) when dividing by \( w' \)

- Consider the following transformation

\[
\begin{pmatrix}
x'w' \\
y'w' \\
z'w' \\
w'
\end{pmatrix} = \begin{pmatrix} h & 0 & 0 & 0 \\ 0 & h & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & 1 & 0 \end{pmatrix}
\begin{pmatrix}
x \\ y \\ z \\ 1
\end{pmatrix}
\]

- 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 nonlinear \( \frac{1}{z} \) to be expressed with linear matrix multiplication!
Perspective Projection

- The third equation in the linear system is $z'w' = az + b$ or $z'z = az + b$, but $z$ values are not needed since all projected points lie on $z = h$ image plane.
- However, computing $z'$ as a monotonically increasing function of $z$ allows it to be used to determine occlusions (for alpha channel transparency).
- If $z = n$ is the near clipping plane and $z = f$ is the far clipping plane, these clipping planes can be preserved in $z'$ by setting $z' = n$ and $z' = f$.
- This gives 2 equations in 2 unknowns: $n^2 = an + b$ and $f^2 = af + b$ leading to $a = n + f$ and $b = -fn$.
- This transforms the viewing frustum into an orthographic volume in screen space.