Building a Virtual World

- Goal: mimic human vision in a virtual world (with a computer)
  - Cheating 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 look at the same spot on the same object and see it
  • Light leaving that point/spot on the object is 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 from a small region on the object, giving interpretable spatial detail
Aperture

• Cameras are similar to the eye, except 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 not often properly accounted for in virtual worlds
• Thus, some complex effects like depth of field, motion blur, chromatic aberration, lens flare, etc. have to be approximated-modeled in other ways (as we will discuss later)
• Particularly complex is lens flare, which is caused by light being reflected/scattered by lenses in a complex lens system
• This is caused in part by material inhomogeneities in the lenses, and depends on the geometry of a lens surface 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 will enter and the image is too dark/noisy to interpret
    • In addition, light might diffract (instead of traveling in straight lines) distorting the image
  • If too large, light from a large area of an object hits the same cone (causing blurring)
• Luckily, the virtual world 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)
• 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

• 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
• 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!
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 looks terrible)
- Set up the scene like a real-world scene!
- Get very familiar with the virtual camera!
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 $\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ and move it around in the virtual world.
- An object is just a collection of points, and as such the methods for handling a single point 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 may require rotation, translation, resizing of the object.
- When taking a (virtual) picture, points on the object are projected onto the film, which we refer to as screen space.
- Unlike rotation/translation/resizing, the projection onto screen space is highly nonlinear and the source of undesirable distortion.
Rotation

• Given a 3D point, \( \hat{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \)

• In 2D, one can rotate the point counter-clockwise about the origin via:

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

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

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

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

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

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

- 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, $R_y(\theta_y)R_x(\theta_x)\vec{x}$, gives a different results than rotating about the y-axis and then the x-axis, $R_x(\theta_x)R_y(\theta_y)\vec{x}$
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 \( 0 \leq \alpha \leq 1 \) determines all the points on the line segment

- Multiplying points on the line segment by a rotation matrix \( R \) gives:
  \[
  R\vec{u}(\alpha) = R((1 - \alpha)\vec{p} + \alpha\vec{q}) = (1 - \alpha)R\vec{p} + \alpha R\vec{q}
  \]

- \( R\vec{u}(0) = R\vec{p} \) and \( R\vec{u}(1) = R\vec{q} \), and \( 0 \leq \alpha \leq 1 \) determines all the points on the new rotated line segment 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)

- \( \|R\vec{u}(\alpha) - R\vec{p}\|_2^2 = \|R(\vec{u}(\alpha) - \vec{p})\|_2^2 = (\vec{u}(\alpha) - \vec{p})^T R^T R (\vec{u}(\alpha) - \vec{p}) = \|\vec{u}(\alpha) - \vec{p}\|_2^2 \) shows that the distance between any two rotated points is equivalent to the distance between the two original (un-rotated) points
Angles are Preserved

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

- $\mathbf{R}\mathbf{u} \cdot \mathbf{R}\mathbf{v} = \|\mathbf{R}\mathbf{u}\|_2 \|\mathbf{R}\mathbf{v}\|_2 \cos(\hat{\theta})$
- $\mathbf{R}\mathbf{u} \cdot \mathbf{R}\mathbf{v} = (\mathbf{u}^T \mathbf{R}^T \mathbf{R} \mathbf{v}) = \mathbf{u}^T \mathbf{v} = \|\mathbf{u}\|_2 \|\mathbf{v}\|_2 \cos(\theta) = \|\mathbf{R}\mathbf{u}\|_2 \|\mathbf{R}\mathbf{v}\|_2 \cos(\theta)$
- So, the angle $\theta$ between $\mathbf{u}$ and $\mathbf{v}$ is the same as the the angle $\hat{\theta}$ between $\mathbf{R}\mathbf{u}$ and $\mathbf{R}\mathbf{v}$.
• 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 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 (again) 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 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 any homogenous coordinates by its fourth component (i.e. \( w \)) gives \( \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \) or \( \begin{pmatrix} \vec{x} \\ 1 \end{pmatrix} \).

- We convert all of our 3D points to the form \( \vec{x}_H = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \), i.e. \( w = 1 \), to deal with translations.

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

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

  To produce the same result for $\begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix}$, use the 4x4 matrix

\[
\begin{pmatrix}
M_{3\times3} & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix} =
\begin{pmatrix}
M_{3\times3}\mathbf{x} \\
1
\end{pmatrix}
\]

- Similarly, for a vector $\begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ 0 \end{pmatrix}$,

\[
\begin{pmatrix}
M_{3\times3} & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
\mathbf{u}_1 \\
\mathbf{u}_2 \\
\mathbf{u}_3 \\
0
\end{pmatrix} =
\begin{pmatrix}
M_{3\times3}\mathbf{u} \\
0
\end{pmatrix}
\]
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

\[
\begin{pmatrix}
I_{3x3} & t_1 \\
t_2 & t_2 \\
t_3 & t_3 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix} = \begin{pmatrix} \vec{x} + \vec{t} \end{pmatrix}
\]

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

- For a vector

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

which 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

• 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
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
- $M_2 M_1$ transforms the teapot from its object space to world space (on the world space table)
Using Transformations

• Create objects (or parts of objects) in convenient coordinate systems
• Assemble objects from their parts (using transformations)
• Then, 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)

• **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 into the desired orientation
  • Afterwards, translate the object to the desired 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

\[
\frac{x}{z} = \frac{x'}{h} \quad \text{and} \quad \frac{y}{z} = \frac{y'}{h}
\]
Matrix Form

- Express the screen space result in homogeneous coordinates:

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

- 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 (so it can be added to the matrix multiplication stack)!
Perspective Projection

- The third equation in the linear system is $z'w' = az + b$ or $z'z = az + b$
- $z$ values are not needed, since the 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)
- If $z = n$ is the near clipping plane and $z = f$ is the far clipping plane, these clipping planes are preserved via $z' = n$ and $z' = f$ (respectively)
- 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