Eye space is the coordinate system at the camera: x right, y up, z out (i.e. looking down -z)
The setup

• Once we’ve applied the modelview transform, we have moved all the objects into eye space

• Next, we define a viewing volume and apply a projection transform to map objects to the screen

• This is where perspective comes into play
Projection Transform

- The idea: take the viewing volume and transform it to Homogeneous Clip Coordinates (HCC)
  - HCC is a cube centered at origin, extending $[-1, 1]$ in all dimensions
  - OpenGL reverses $Z$ axis (far away = $+Z$ now)
- Gives a consistent workspace to apply perspective
- Clipping happens in these coordinates as well
**Geometry pipeline**

- **Object coordinates**
- **Eye coordinates**
- **Homogeneous clip coordinates**
- **Screen coordinates**
- **Normalized device coordinates**

Diagram:
- Model & View Transform
- Vertex Shading
- Projection
- Clipping
- Screen Mapping
Orthographic Projection

- Project everything in the viewing volume onto a flat plane, without perspective
- Parallel lines stay parallel, like blueprints
Orthographic projection

- This projection can be computed with a matrix
Building orthographic projections

- First, translate center to origin
- Then, scale around origin to correct size
$\mathbf{P}_{\text{ortho}}(r, l, b, t, n, f) = \begin{bmatrix}
\frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\
0 & \frac{2}{t-b} & 0 & -\frac{r-l}{t+b} \\
0 & 0 & \frac{-2}{f-n} & -\frac{t-b}{f+n} \\
0 & 0 & 0 & \frac{f-n}{f-n} \\
\end{bmatrix}$

- $f$ is far clipping plane, $n$ is near clipping plane
- Inverts Z coordinate
What about perspective?

- We want objects that are farther away to be smaller.
- This means that they need to be scaled in X and Y based on their Z coordinate.
Problem!

- Wait, we want to scale one coordinate based on the value of another coordinate...
- That’s not linear!
  - Any particular scaling matrix is linear, but scaling that depends on a coordinate is not
- We can’t use a matrix to represent that anymore
Real projective space

\(< x, y, z, w > \rightarrow \left< \frac{x}{w}, \frac{y}{w}, \frac{z}{w} \right>

• The \(w\) in homogeneous coordinates isn’t a regular coordinate, it’s a divisor for all the others

• Every point is actually a line along \(w\)

• All points on that line are equivalent

  • e.g. \(< 1, 0, -2, 1 > = < 5, 0, -10, 5 >\)
The actual division by \( w \) is done towards the end of the pipeline, after clipping.

- Turns HCC into Normalized Device Coordinates (NDC for short)
- We can put Z coordinates into \( w \), causing thing to shrink the farther away they are
- We can represent that much as a matrix just fine
What happens to Z?

- We only care about X and Y in screen space for positions.
- However, Z is necessary for its depth information.
- If we don’t change its values, it will get divided out by $w$ and turn into 1.
- So, we rescale Z in the matrix such that it equals -1 at the near plane and 1 at the far plane.
Perspective projection

- Do nonlinear mapping using a matrix, putting distance scaling values into $w$ coordinate
- Adjust $Z$ coordinates so we don’t lose them after divide
Perspective derivation

• Start off by putting Z in $\omega$ coordinate
• Want near plane points to get rescaled to $[-1, 1]$
• Want Z values to be rescaled to $[-1, 1]$
• Must keep in mind that we’re dividing by Z after all the linear transforms
glFrustum

\[ P_{frustum}(r, l, b, t, n, f) = \begin{bmatrix}
\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\
0 & 0 & \frac{f+n}{f-n} & \frac{2fn}{f-n} \\
0 & 0 & -1 & 0
\end{bmatrix} \]

- Reverses Z coordinate, like glOrtho
- Puts reversed Z into \( w \), so objects will be scaled by it when perspective divide happens
Asymmetric Frustums

- Very useful when doing tiled renderings, or splitting work between CPUs / GPUs
\[ \text{gluPerspective} \]

\[
P_{\text{persp}}(f_{\text{ovy}}, a, n, f) = \begin{bmatrix}
\frac{\cot(f_{\text{ovy}}/2)}{a} & 0 & 0 & 0 \\
0 & \cot(f_{\text{ovy}}/2) & 0 & 0 \\
0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\
0 & 0 & -1 & 0
\end{bmatrix}
\]

- Same deal, just a different way of picking the X and Y scaling terms
After $W$ divide
Z resolution

- We did a linear transform on Z, then divided it by its old value
- The values we get are no longer linear in depth
- Lose resolution as near and far separate
Infinite Far Plane

\[ P_{inf}(l, r, b, t, n) = \begin{bmatrix}
\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
0 & \frac{2n}{t-b} & \frac{r-l}{t-b} & 0 \\
0 & 0 & -1 & -2n \\
0 & 0 & -1 & 0 \\
\end{bmatrix} \]

- Take limit of frustum as far goes to infinity
- Z resolution is absolute garbage
- Useful in specific situations
FIGURES COURTESY...

- Real-Time Rendering, 3rd ed. [RTR]
  - Tomas Akenine-Moller, Eric Haines, Naty Hoffman
  - Eric Lengyel
  - Edward Angel, Dave Shreiner
- Wikipedia [WP]
- songho.ca [SH]