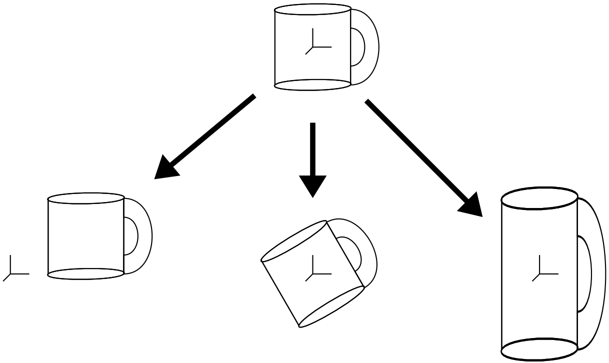
3D Transformations

CMSC 435/634

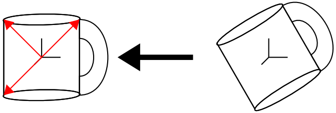
Transformation

Webster: The operation of changing one configuration or expression into another in accordance with a mathematical rule



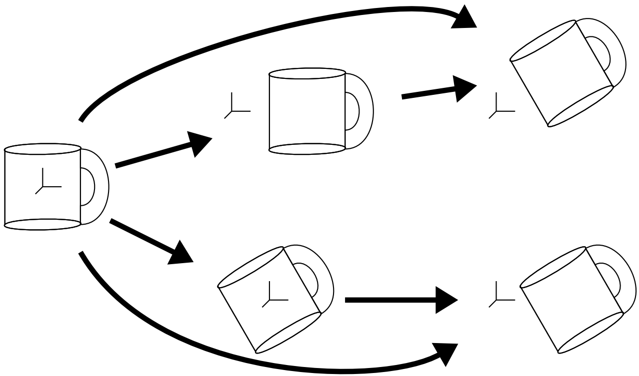
Using Transformation

- Points on object represented as vector offset from origin
- Transform is a vector to vector function
  - $\vec{p'} = f(\vec{p})$
- Relativity:
  - From $\vec{p'}$ point of view, object is transformed
  - From $\vec{p}$ point of view, coordinate system changes
- Inverse transform, $\vec{p} = f^{-1}(\vec{p'})$



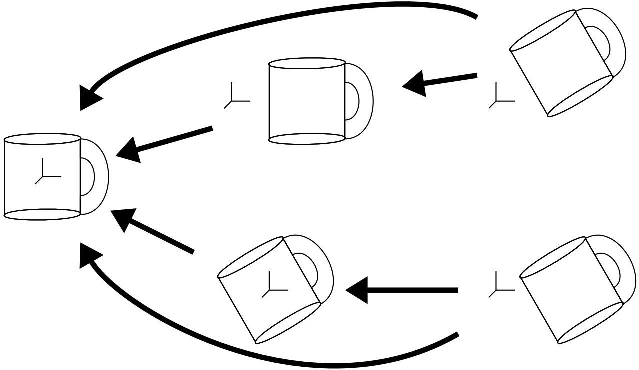
Composing Transforms

- Order matters
  - $R(T(\vec{p})) = R \circ T(\vec{p})$
  - $T(R(\vec{p})) = T \circ R(\vec{p})$



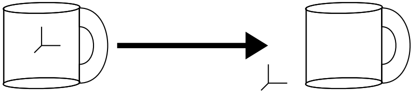
Inverting Composed Transforms

- Reverse order
  - $(R \circ T)^{-1}(\vec{p'}) = T^{-1}(R^{-1}(\vec{p'}))$
  - $(T \circ R)^{-1}(\vec{p'}) = R^{-1}(T^{-1}(\vec{p'}))$



Translation

- $\vec{p'} = \vec{p} + \vec{t}$
- $\begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \end{bmatrix}$
- $\vec{t}$ says where $\vec{p}$-space origin ends up ($\vec{p'} = \vec{0} + \vec{t}$)
- Composition: $\vec{p'} = (\vec{p} + \vec{t_0}) + \vec{t_1} = \vec{p} + (\vec{t_0} + \vec{t_1})$

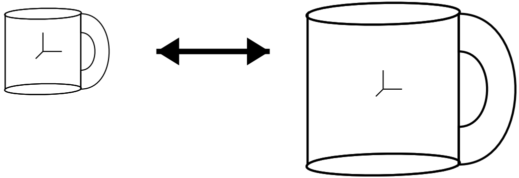


Linear Transforms

- $$\begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$
- Matrix says where $\vec{p}$-space axes end up
  - $$\begin{bmatrix} a \\ d \\ g \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$
    $$\begin{bmatrix} b \\ e \\ h \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$
    $$\begin{bmatrix} c \\ f \\ i \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$
  - Composition: $\vec{p'} = M\ (N\ \vec{p}) = (M\ N)\vec{p}$

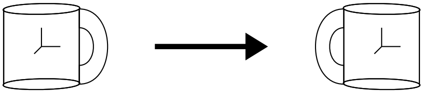
Common case: Scaling

- $\begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix} = \begin{bmatrix} s_x\ p_x \\ s_y\ p_y \\ s_z\ p_z \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$
- Inverse: $\begin{bmatrix} 1/s_x & 0 & 0 \\ 0 & 1/s_y & 0 \\ 0 & 0 & 1/s_z \end{bmatrix}$

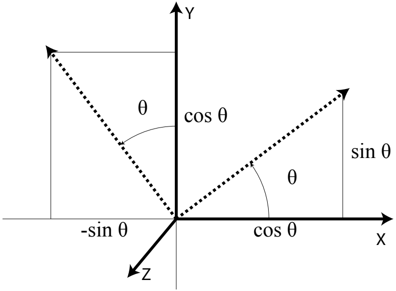


Common case: Reflection

- Negative scaling
- $\begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix} = \begin{bmatrix} -p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$

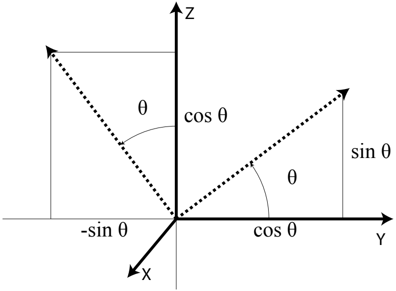


Common case: Rotation



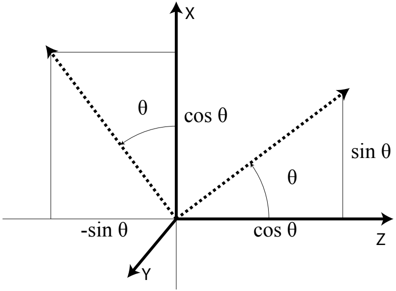
- Rotate around Z: $\vec{p'} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \vec{p}$
- Orthogonal, so $M^{-1} = M^T$

Common case: Rotation



- Rotate around X: $\vec{p'} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \vec{p}$
- Orthogonal, so $M^{-1} = M^T$

Common case: Rotation



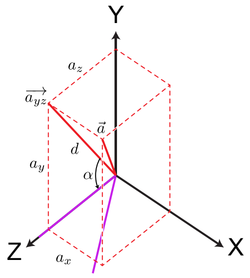
- Rotate around Y: $\vec{p'} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \vec{p}$
- Orthogonal, so $M^{-1} = M^T$

Composing Transforms

- Scale by s along axis $\hat{a}$
  - Rotate to align $\hat{a}$ with Z
  - Scale along Z
  - Rotate back

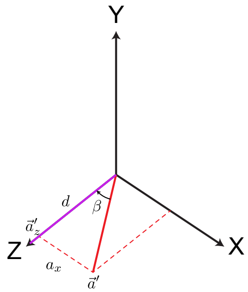
Rotate by α around X into XZ plane

- Projection of $\hat{a}$ onto YZ: $\overrightarrow{a_{yz}} = \begin{bmatrix} 0 \\ a_y \\ a_z \end{bmatrix}$
- length $d = \sqrt{(a_y)^2 + (a_z)^2}$
- So $\cos\alpha = a_z/d$, $\sin\alpha = a_y/d$
- $R_X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & a_z/d & -a_y/d \\ 0 & a_y/d & a_z/d \end{bmatrix}$
- Result $\hat{a}' = \begin{bmatrix} a_x \\ 0 \\ d \end{bmatrix}$



Rotate by -β around Y to Z axis

- $\hat{a}' = \begin{bmatrix} a_x \\ 0 \\ d \end{bmatrix}$
- length $= 1$
- So $\cos\beta = d$, $\sin\beta = a_x$
- $R_Y = \begin{bmatrix} d & 0 & -a_x \\ 0 & 1 & 0 \\ a_x & 0 & d \end{bmatrix}$
- Result $\hat{a}'' = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$



Composing Transforms

- Scale by s along Z: $S_Z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix}$
- Scale by s along axis $\hat{a}$
  - Rotate to align $\hat{a}$ with XZ plane
  - Rotate to align $\hat{a}$ with Z axis
  - Scale along Z
  - Undo Z-axis alignment rotation
  - Undo XZ-plane alignment rotation
  - $\vec{p'} = R_X^{-1} R_Y^{-1} S_Z R_Y R_X \vec{p}$

Affine Transforms

- Affine = Linear + Translation
- Composition? $A\,(B\;\vec{p} + \vec{t_0}) + \vec{t_1} = A\;B\;\vec{p} + A\;\vec{t_0} + \vec{t_1}$
- Yuck!

Homogeneous Coordinates

- Add a '1' to each point
- $$\begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$
- $\vec{p'}_x = (a\ p_x + b\ p_y + c\ p_z) + t_x$
- $\vec{p'}_y = (d\ p_x + e\ p_y + f\ p_z) + t_y$
- $\vec{p'}_z = (g\ p_x + h\ p_y + i\ p_z) + t_z$
- $1 = (0p_x + 0p_y + 0p_z) + 1$

Homogeneous Coordinates

- $$\begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$
- $\vec{p'} = \left[\begin{array}{ccc|c} \vec{x} & \vec{y} & \vec{z} & \vec{t} \end{array}\right] \vec{p}$
  - $\vec{t}$ says where the $\vec{p}$-space origin ends up
  - $\vec{x}$, $\vec{y}$, $\vec{z}$ say where the $\vec{p}$-space axes end up
- Composition: Just matrix multiplies!

Composing Transforms

- Rotate by θ about line between $\vec{p}_0$ and $\vec{p}_1$:
  - Translate $\vec{p}_0$ to origin
  - Rotate to align $\vec{p}_1 - \vec{p}_0$ with Z
  - Rotate by θ around Z
  - Undo $\vec{p}_1 - \vec{p}_0$ rotation
  - Undo translation
- $T^{-1} R_X^{-1} R_Y^{-1} R_Z(\theta) R_Y R_X T$

Vectors

- Transform by *Jacobian Matrix*
- Matrix of partial derivatives
  - $$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a\ x + b\ y + c\ z + t_x \\ d\ x + e\ y + f\ z + t_y \\ g\ x + h\ y + i\ z + t_z \end{bmatrix}$$
  - $$J = \begin{bmatrix} \partial x'/\partial x & \partial x'/\partial y & \partial x'/\partial z \\ \partial y'/\partial x & \partial y'/\partial y & \partial y'/\partial z \\ \partial z'/\partial x & \partial z'/\partial y & \partial z'/\partial z \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$
- Use upper-left 3x3, or 0 for final coordinate:
  - $$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \text{ or } \left[\begin{array}{ccc|c} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

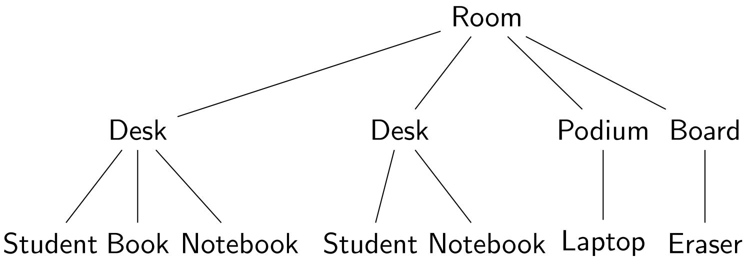
Normals

- Normal should remain perpendicular to tangent vectors
- $\vec{n} \cdot \vec{v} = \vec{n'} \cdot \vec{v'} = 0$
- $\begin{bmatrix} n_x & n_y & n_z \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \left(\begin{bmatrix} n_x & n_y & n_z \end{bmatrix} J^{-1} \right) \left(J \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \right) = 0$
- $\vec{n'} = \vec{n} J^{-1}$
- Multiply by inverse on right
- OR multiply *column* normal by inverse transpose
  - $\vec{n'} = (J^{-1})^T \vec{n}$
  - $(J^{-1})^T = J$ if J is orthogonal (only rotations)

Coordinate System / Space

- Origin + Axes
- Reference frame
- Convert by matrix
- OpenGL convention (we use this!): Points are columns
  - $\vec{p}_{table} = TableFromPencil\ \vec{p}_{pencil}$
  - $\vec{p}_{room} = RoomFromTable\ TableFromPencil\ \vec{p}_{pencil}$
  - $\vec{p}_{room} = RoomFromPencil\ \vec{p}_{pencil}$
- Same thing in D3D convention (Points are rows, everything transposed)
  - $\vec{p}_{table} = \vec{p}_{pencil}\ PencilToTable$
  - $\vec{p}_{room} = \vec{p}_{pencil}\ PencilToTable\ TableToRoom$
  - $\vec{p}_{room} = \vec{p}_{pencil}\ PencilToRoom$

Nesting


Room
Desk
Desk
Podium
Board
Student
Book
Notebook
Student
Notebook
Laptop
Eraser

Matrix Stack

- Remember transformation, return to it later
- Push a copy, modify the copy, pop
- Keep matrix and update matrix and inverse
- Push and pop both matrix and inverse together

code	**stack** (start with Identity)
`transform(WorldFromRoom);`	WfR
`push;`	WfR WfR
`transform(RoomFromDesk);`	WfD WfR
`push;`	WfD WfD WfR
`transform(DeskFromStudent);`	WfS WfD WfR
`pop;`	WfD WfR
`push;`	WfD WfD WfR
`transform(DeskFromBook);`	WfB WfD WfR
`...`	

Common Spaces

- Object / Model
  - Logical coordinates for modeling
  - May have several more levels
- World
  - Common coordinates for everything
- View / Camera / Eye
  - eye/camera at $(0, 0, 0)$, looking down Z (or -Z) axis
  - planes: left, right, top, bottom, near/hither, far/yon
- Normalized Device Coordinates (NDC) / Clip
  - Visible portion of scene from $(-1, -1, -1)$ to $(1, 1, 1)$
  - Sometimes 0 to 1 (D3D uses $(-1, -1, 0)$ to $(1, 1, 1)$)
- Raster / Pixel / Viewport
  - $0, 0$ to x-resolution, y-resolution
- Device / Screen
  - May translate or scale to fit actual screen

WorldFromModel / ViewFromModel

- WorldFromModel
  - All shading and rendering in World space
  - Transform all objects and lights
- ViewFromModel
  - World can be any common space, might as well use View space
  - Serves just as well for single view
  - Old OpenGL used to have a MODELVIEW transform built in
- Ray tracing implicitly does World $\rightarrow$ Raster

World Coordinate Precision

- Floating point precision is not enough for large worlds
- Float precision of x is (next lower power of 2) * $2^{-23} \approx x * 10^{-7}$
  - Earth radius $6.378 * 10^6 m$; UMBC at $39.2498^\circ N, 76.7115^\circ W$
  - $\therefore X = 1.135 * 10^6 m$; $Y = 4.807 * 10^6 m$; $Z = 4.035 * 10^6 m$
  - Position resolution: $X \pm 0.125m$; $Y \pm 0.5m$; $Z \pm 0.25m$
- Use doubles ... or recenter world space
  - UnrealEngine: *Translated World* or *Large World Coordinates (LWC)*
  - Translated World: origin at camera since floating point precision is better near origin
    - Errors are farther away where they're harder to see
  - LWC: include a tile translation exactly representable as float (0's in least significant bits) and float world positions relative to tile.
- Only need to worry about this for **huge** worlds

ViewFromWorld

- Also called Viewing or Camera transform
- LookAt
  - $\overrightarrow{from}, \overrightarrow{to}, \overrightarrow{up}$
  - $\hat{w} = \mathrm{normalize}(\overrightarrow{to} - \overrightarrow{from}); \hat{u} = \mathrm{normalize}(\hat{w} \times \overrightarrow{up}); \hat{v} = \hat{u} \times \hat{w}$
  - $\left[\begin{array}{c|c|c|c} \hat{u} & \hat{v} & \hat{w} & \overrightarrow{from} \end{array}\right]$
- Roll / Pitch / Yaw (use without roll for FPS)
  - Translate to camera center, rotate around camera
  - $R_z\ R_x\ R_y\ T$
  - Can have gimbal lock when first and last axes align
- Orbit
  - Rotate around object center, translate out
  - $T\ R_z\ R_x\ R_y$
  - Also can have gimbal lock

NDCFromView

- Also called *Projection* transform
- Orthographic / Parallel
  - Translate & Scale to view volume
  - $$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
- Perspective
  - More complicated...

RasterFromNDC

- Also called *Viewport* transform
- $[-1,1],[-1,1],[-1,1] \to [0,n_x],[0,n_y],[0,n_z]$
- or $\to [-\frac{1}{2}, n_x - \frac{1}{2}], [-\frac{1}{2}, n_y - \frac{1}{2}], [-\frac{1}{2}, n_z - \frac{1}{2}]$
  - Translate by $(1,1,1)$: $(-1,-1,-1) \to (0,0,0); (1,1,1) \to (2,2,2)$
  - Scale by $(n_x/2, n_y/2, n_z/2)$: $(2,2,2) \to (n_x, n_y, n_z)$
  - (if needed) Translate by $(-\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2})$ — puts pixel centers at integer coordinates

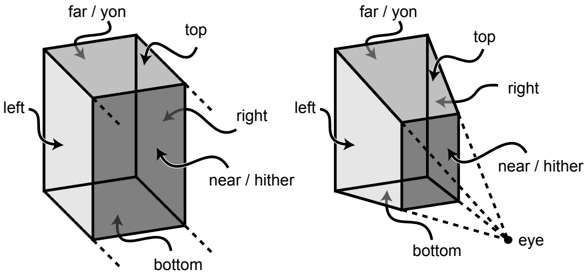
$$\begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y}{2} \\ 0 & 0 & \frac{n_z}{2} & \frac{n_z}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & \frac{n_z}{2} & \frac{n_z-1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ScreenFromRaster

- Usually just a translation
  - Some game consoles include scaling for performance
  - More complicated for tiled displays, domes, etc.
- Usually handled by windowing system

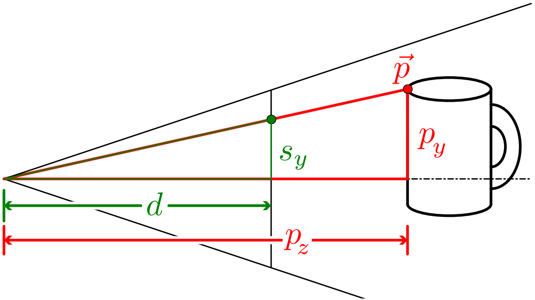
Perspective View Frustum

- *Orthographic* view volume is a rectangular volume
- Perspective is a truncated pyramid or *frustum*



Perspective Transform

- Ray tracing
  - Given screen (s_x, s_y), parameterize all points $\vec{p}$
- Perspective Transform
  - Given $\vec{p}$, find (s_x, s_y)
  - Use similar triangles
  - $s_y/d = p_y/p_z$ So $s_y = d\ p_y/p_z$



Homogeneous Equations

- Same total degree for every term
- Introduce a new redundant variable
- Plane equation
  - $\mathrm{a}\,X + \mathrm{b}\,Y + \mathrm{c} = 0$
  - $X = x/w, Y = y/w$
  - $\mathrm{a}\,x/w + \mathrm{b}\,y/w + \mathrm{c} = 0$
  - $\rightarrow \mathrm{a}\,x + \mathrm{b}\,y + \mathrm{c}\,w = 0$
- Quadric
  - $\mathrm{a}\,X^2 + \mathrm{b}\,X\,Y + \mathrm{c}\,Y^2 + \mathrm{d}\,X + \mathrm{e}\,Y + \mathrm{f} = 0$
  - $X = x/w, Y = y/w$
  - $\mathrm{a}\,x^2/w^2 + \mathrm{b}\,x\,y/w^2 + \mathrm{c}\,y^2/w^2 + \mathrm{d}\,x/w + \mathrm{e}\,y/w + \mathrm{f} = 0$
  - $\rightarrow \mathrm{a}\,x^2 + \mathrm{b}\,x\,y + \mathrm{c}\,y^2 + \mathrm{d}\,x\,w + \mathrm{e}\,y\,w + \mathrm{f}\,w^2 = 0$

Homogeneous Coordinates

- Rather than $(x, y, z, 1)$, use (x, y, z, w)
- Real 3D point is $(X, Y, Z) = (x/w, y/w, z/w)$
- Can represent Perspective Transform as 4x4 matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ p_z/d \end{bmatrix} \rightarrow \begin{bmatrix} d\ p_x/p_z \\ d\ p_y/p_z \\ d \end{bmatrix}$$

Homogeneous Depth

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ p_z/d \end{bmatrix} \rightarrow \begin{bmatrix} d\, p_x/p_z \\ d\, p_y/p_z \\ d \end{bmatrix}$$

- Lose depth information
- Can't get $d\, p_z'/p_z = p_z$
  - Plus $x/z, y/z, z$ isn't linear
- Use *Projective Geometry*

Projective Geometry

- If (x, y, z) lie on a plane, $(x/z, y/z, 1/z)$ also lie on a plane
- $1/z$ is strictly ordered: if $z_1 < z_2$, then $1/z_1 > 1/z_2$
- New matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ 1 \\ p_z \end{bmatrix} \to \begin{bmatrix} p_x/p_z \\ p_y/p_z \\ 1/p_z \end{bmatrix}$$

Getting Fancy

- Tuning transform output
  - Field of view (x/y scale)
  - Near/far range (z scale and translate)

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & d \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} a\,p_x \\ b\,p_y \\ c\,p_z + d \\ -p_z \end{bmatrix} \rightarrow \begin{bmatrix} -a\,p_x/p_z \\ -b\,p_y/p_z \\ -c - d/p_z \end{bmatrix}$$

- $b = 1/\tan(yfov/2); a = 1/\tan(xfov/2) = b * height/width;$
- OpenGL convention: Solve for $(0,0,-n) \rightarrow (0,0,-1); (0,0,-f) \rightarrow (0,0,1)$
  - $c = (n+f)/(n-f);\ d = (2\,n\,f)/(n-f)$
- D3D convention: Solve for $(0,0,n) \rightarrow (0,0,0); (0,0,f) \rightarrow (0,0,1)$
  - $c = f/(n-f);\ d = n\,f/(f-n)$