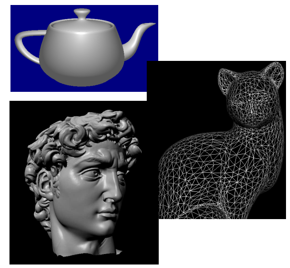
Lecture 7
Camera and Basic Viewing Projections

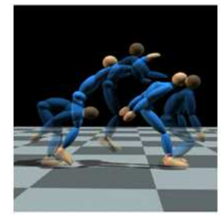
CS174A

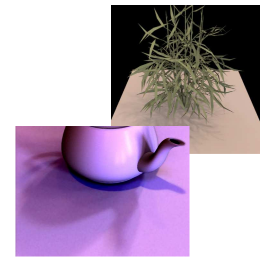
Animation Recap

- 3D Graphics Pipeline

Modeling → Animation → Rendering







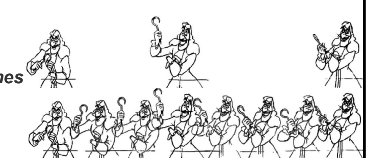
Traditional (Manual) Animation

Every frame is created individually by a human

- That's 24 frames/sec at traditional movie speeds
  - Roughly 130,000 frames for a 1.5 hr movie

A general pipeline evolved to support efficiency

- Start with a ***storyboard***
  - A set of drawings outlining the animation
- Senior artists sketch important frames – ***Keyframes***
  - Typically occur when motion changes



- Lower-paid artists draw the rest of the frames – ***in-betweens***
- All line drawings are painted on ***cels***
  - Generally composed in layers, hence the use of acetate
  - Background changes infrequently, so it can be reused
- Photograph finished ***cel-stack*** onto film

Computer Generated Animations

Physical Simulations

- We usually want realistic looking motion
  - People are extremely experienced at observing body language
  - They pick up on unnatural human motion instantly
- Some of the methods we've discussed can achieve realism
  - If our animator makes good enough key frames
  - Or we write good enough procedural scripts
  - Or we strap a bunch of sensors on an actor
- But there's another good alternative
  - Why not just ***simulate the relevant physical laws*** ?
  - Then we'll know that the motion is natural
  - And we'll still have decent control over it

Basic Particles

- Properties
  - mass
  - Position, velocity, acceleration
  - color
  - temperature
  - age
- Differential equations govern these properties
- Collisions and other constrains directly modify position and/or velocity

Dynamics

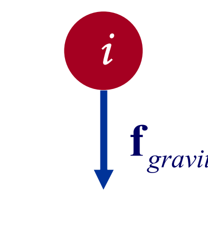
- Basic governing equation
  - Newton's Laws of Physics

$$\mathbf{f} = m\mathbf{a} \rightarrow \frac{d^2\mathbf{x}}{dt^2} = \frac{\mathbf{f}}{m} \quad \text{or} \quad \ddot{\mathbf{x}} = \frac{\mathbf{f}}{m}$$

  - And in general we must solve them numerically (discretize time)
- **f** is a sum of a number of forces due to
  - **Gravity**: constant downward force proportional to mass
  - Simple drag (**damping force**) : force proportional to negative velocity
  - **Particle interactions**: particles mutually attract and/or repell
  - **Wind forces**
  - **User interaction**

External Forces

- Gravitational force

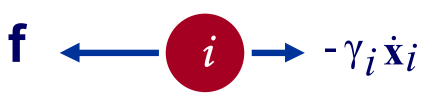


$$\mathbf{f}_{gravity} = m_i\mathbf{a} \qquad \mathbf{a} = \begin{bmatrix} 0 \\ 0 \\ -9.8m/s^2 \\ 0 \end{bmatrix}$$

Damping Force

- Behaves like viscous drag on all motion

$$\mathbf{f}_{\text{damping}} = -\gamma_i \dot{\mathbf{x}}_i$$



- γ_i is the damping coefficient

Particle interactions - Discrete Fluid Model

The total force, $\mathbf{g}_i$, on a particle, i, due to all other particles (important in fluid modeling)

$$\mathbf{g}_i(t) = \sum_{j \neq i} \mathbf{g}_{ij}(t)$$

$$\mathbf{g}_{ij}(t) = m_i m_j (\mathbf{x}_i - \mathbf{x}_j)\left(-\overbrace{\frac{\alpha}{(r_{ij} + \varsigma)^a}}^{\text{attraction term}} + \overbrace{\frac{\beta}{(r_{ij})^b}}^{\text{repulsion term}} \right)$$

$a=2$ and $b=4$

α and β determine the strength of the attraction and repulsion forces

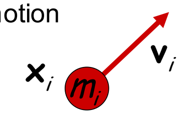
$r_{ij} = \left\| \mathbf{x}_j - \mathbf{x}_i \right\|$

ς minimum required separation between particles

Particle Dynamics

Set of particles modeled as point masses in motion

- m_i : mass of particle i
- $\mathbf{x}_i$: position of particle i
- $\mathbf{v}_i$: velocity of particle i



Compute positions from Newton's second law

$$\mathbf{f}_i(t) = m_i \mathbf{a}_i(t)$$

$\mathbf{f}_i$: sum of all forces acting on particle

$$\mathbf{a}_i^t = \frac{f_{i,total}^t}{m_i}$$

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \Delta t\, \mathbf{a}_i^t$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \underbrace{\Delta t\, \mathbf{v}_i^{t+1}}_{\text{Translate by } \Delta t\, \mathbf{v}_i^{t+1}}$$

Deformable Models

Continuum mechanics

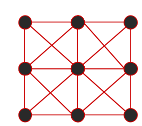
- Deformable solid models
  - Cloth
  - Rubber
  - Soft tissues (muscle, skin, hair, …)
- Fluid models
  - Water (oceans, puddles, rain, …)
- Gas-like models
  - Steam, smoke, fire, …

Deformable Solids: Mass-Spring-Damper Systems

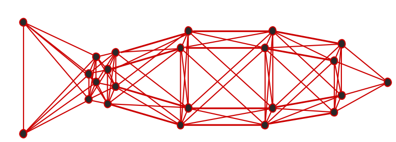
Useful for building deformable models

1-dimensional:

2-dimensional:



3-dimensional:



System Dynamics / Total Force Computation

1. For each nodal mass sum up all the forces:

$$\mathbf{F}_{i,\text{total}} = -\gamma_i \dot{\mathbf{x}}_i + \mathbf{s}_i + \mathbf{f}_i$$

nodes interact with each other through spring forces

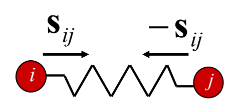
- γ_i is damping coefficient
- $\mathbf{s}_i$ total internal force on the node *i* due to neighboring nodes connected by springs
- $\mathbf{f}_i$ is the external force at node *i* *(ie., gravity, interaction forces)*

2. Compute the acceleration, velocity and position from Newton's 2nd Law of Dynamics

$$\mathbf{F}_{i,\text{total}} = m_i \ddot{\mathbf{x}}_i$$

Simple Ideal Spring

- Ideal zero length spring
- Force pulls points together

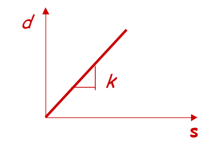


$$\mathbf{s}_{ij} = k_{ij}\underbrace{(\mathbf{x}_i - \mathbf{x}_j)}_{\text{distance}}$$

spring constant

$$\mathbf{f}_i^t = \mathbf{f}_i^{t-1} + \mathbf{s}_{ij}^t$$

$$\mathbf{f}_j^t = \mathbf{f}_j^{t-1} - \mathbf{s}_{ij}^t$$



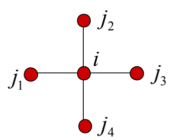
- Strength proportional to distance

Internal Non-zero Length Spring Forces

Spring Forces:

- $\mathbf{s}_i(t)$ total force on the node *i* due to springs connecting it to neighboring nodes $j \in N_i$

$$\mathbf{s}_i(t) = \sum_{j \in N_i} \mathbf{s}_{ij}$$



- the force spring *ij* exerts on node *i*

$$\mathbf{s}_{ij} = k_{ij} e_{ij} \frac{\mathbf{d}_{ij}}{\|\mathbf{d}_{ij}\|}$$

$\mathbf{d}_{ij} = \mathbf{x}_j - \mathbf{x}_i$ node distance/separation

$\|\mathbf{d}_{ij}\|$ actual spring length

$e_{ij} = \|\mathbf{s}_{ij}\| - l_{ij}$ spring deformation, l_{ij} natural spring length

k_{ij} is the spring constant for the spring connecting node ***i*** and node ***j***

Integrating the Equations of Motion Through Time

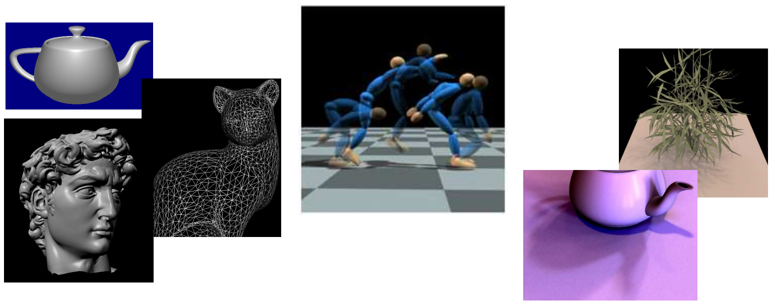
The explicit Euler time-integration method

- For each node *i* do:
  - Step 1: $\mathbf{a}_i^t = \dfrac{F_{i,total}^t}{m_i}$
  - Step 2: $\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \Delta t\, \mathbf{a}_i^t$
  - Step 3: $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \underbrace{\Delta t\, \mathbf{v}_i^{t+1}}_{\text{Translate by } \Delta t\, \mathbf{v}_i^{t+1}}$

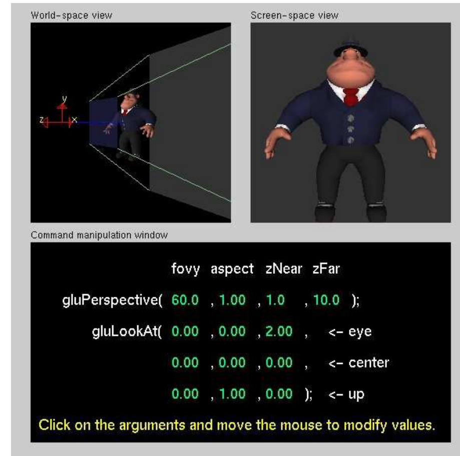
Course Outline

- 3D Graphics Pipeline

Modeling → Animation → Rendering



Camera and Basic Viewing Projections



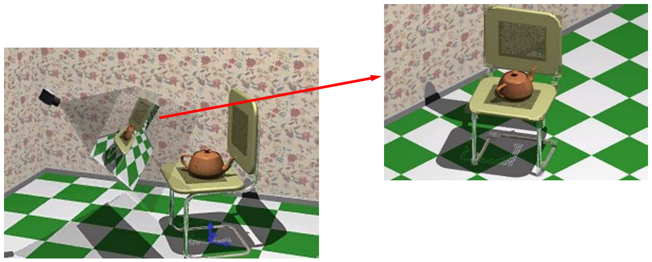
Lecture Outline

- ***Camera Transformations***
- Projections
  - Orthographic projection (simpler)
  - Orthographic Viewing Cube
  - Perspective projection, basic idea
  - Perspective Viewing Frostrum
  - Reading chapter 7

Motivation

- We have used transforms to place objects in a scene, but all that is in 3D. We still need to make a 2D picture
- How do we do this? Do what eyes/cameras do. Project 3D to 2D.
- This lecture
  - viewing transforms - where is the camera, what is it pointing at?
    - where is the camera, what is it pointing at?
  - Perspective/orthograpic projection: 3D to 2D
    - flatten to image

Rendering a 3D Scene From the Point of View of a Virtual Camera



Rendering Pipeline

Scene graph
Object geometry

Modelling Transforms

Viewing Transform

Projection

Rendering Pipeline

Scene graph
Object geometry

Modelling Transforms

Viewing Transform

Projection

- result
  - all vertices of scene in shared 3D arbitrary world coordinate system

Rendering Pipeline

Scene graph
Object geometry

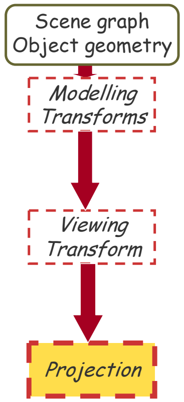
Modelling Transforms

Viewing Transform

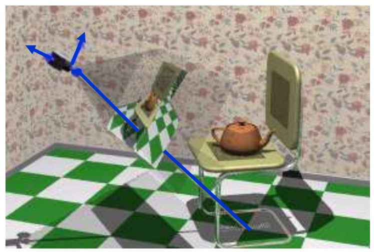
Projection

- result
  - scene vertices in 3D view (camera) coordinate system

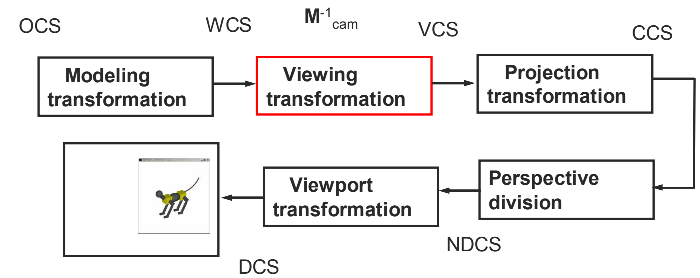
Rendering Pipeline



- result
  - 2D screen coordinates of clipped vertices



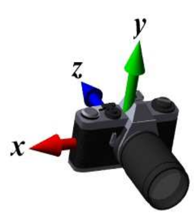
Detailed Graphics Pipeline



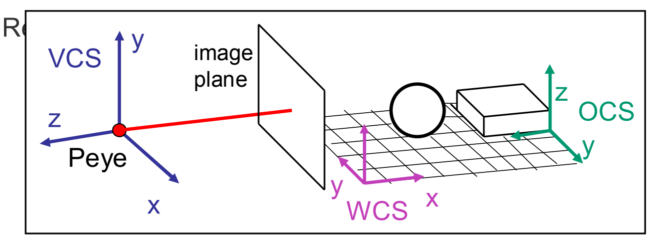
OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system

OpenGL Convention

In world coordinates, the camera system is defined as follows:



Camera Transformation

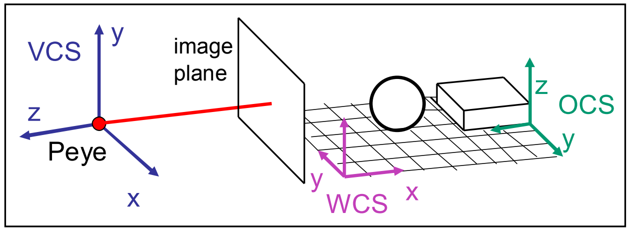


$$P_{\mathrm{WCS}} = \mathrm{M}_{\mathrm{mod}} P_{\mathrm{OCS}}$$

$$P_{\mathrm{WCS}} = \mathrm{M}_{\mathrm{cam}} P_{\mathrm{VCS}} \quad \Rightarrow \quad P_{\mathrm{VCS}} = \mathrm{M}_{\mathrm{cam}}^{-1} P_{\mathrm{WCS}}$$

$$P_{\mathrm{VCS}} = \underbrace{\mathrm{M}_{\mathrm{cam}}^{-1} \mathrm{M}_{\mathrm{mod}}}_{\text{Modelview Transformation}} P_{\mathrm{OCS}}$$

Viewing Transformation



object	world	viewing
OCS	**WCS**	**VCS**

→ **modeling transformation** → **viewing transformation** →

$\mathbf{M}_{mod}$ $\mathbf{M}^{-1}_{cam}$

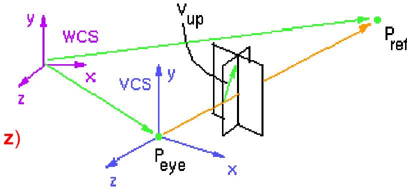
OpenGL ModelView matrix

Defining $\mathbf{M}_{cam}$

Given:

Eye point P_{eye}
Reference point P_{ref}
Up vector $\mathbf{v}_{up}$

($\mathbf{v}_{up}$ is not necessarily orthogonal to **z**)



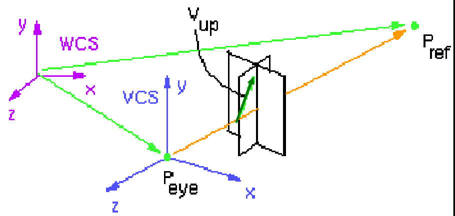
To build $\mathbf{M}_{cam}$ we need to define a camera coordinate system (O, **i**, **j**, **k**)

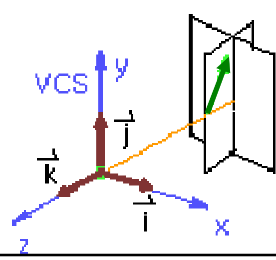
Camera Coordinate System

$$\mathbf{k} = \frac{P\text{eye} - P\text{ref}}{|P\text{eye} - P\text{ref}|}$$

$$\mathbf{i} = \frac{\mathbf{v}\text{up} \times \mathbf{k}}{|\mathbf{v}\text{up} \times \mathbf{k}|}$$

$$\mathbf{j} = \mathbf{k} \times \mathbf{i}$$



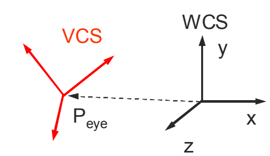


Building $\mathbf{M}_{cam}$

Change of basis

Our reference system is WCS,
we know the camera parameters with
respect to the world

Align WCS with VCS



$$\mathrm{M}\text{cam} = \begin{bmatrix} 1 & 0 & 0 & Peye_x \\ 0 & 1 & 0 & Peye_y \\ 0 & 0 & 1 & Peye_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i_x & j_x & k_x & 0 \\ i_y & j_y & k_y & 0 \\ i_z & j_z & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P_{\text{WCS}} = \mathrm{M}_{\text{cam}} P_{\text{VCS}}$$

Building M_{cam} Inverse

Invert the smart way

$$\mathrm{M}_{\text{cam}}^{-1} = \left(\begin{bmatrix} 1 & 0 & 0 & P\text{eye}_x \\ 0 & 1 & 0 & P\text{eye}_y \\ 0 & 0 & 1 & P\text{eye}_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i_x & j_x & k_x & 0 \\ i_y & j_y & k_y & 0 \\ i_z & j_z & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \right)^{-1}$$

Building M_{cam} Inverse

Invert the smart way

$$\begin{aligned}\mathrm{M}_{\text{cam}}^{-1} &= \left(\begin{bmatrix} 1 & 0 & 0 & P\text{eye}_x \\ 0 & 1 & 0 & P\text{eye}_y \\ 0 & 0 & 1 & P\text{eye}_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i_x & j_x & k_x & 0 \\ i_y & j_y & k_y & 0 \\ i_z & j_z & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \right)^{-1} \\ &= \begin{bmatrix} i_x & j_x & k_x & 0 \\ i_y & j_y & k_y & 0 \\ i_z & j_z & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & 0 & P\text{eye}_x \\ 0 & 1 & 0 & P\text{eye}_y \\ 0 & 0 & 1 & P\text{eye}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}\end{aligned}$$

Building M_{cam} Inverse

Invert the smart way

$$\begin{aligned}\mathrm{M}_{\text{cam}}^{-1} &= \begin{bmatrix} i_x & j_x & k_x & 0 \\ i_y & j_y & k_y & 0 \\ i_z & j_z & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & 0 & P\text{eye}_x \\ 0 & 1 & 0 & P\text{eye}_y \\ 0 & 0 & 1 & P\text{eye}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \\ &= \begin{bmatrix} \boxed{\begin{matrix} i_x & i_y & i_z \\ j_x & j_y & j_z \\ k_x & k_y & k_z \end{matrix}}^{\text{Transpose}} & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \begin{bmatrix} \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} & \boxed{\begin{matrix} -P\text{eye}_x \\ -P\text{eye}_y \\ -P\text{eye}_z \end{matrix}}^{\text{Negate}} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \\ P_{\text{vcs}} &= \mathrm{M}_{\text{cam}}^{-1} P_{\text{wcs}}\end{aligned}$$

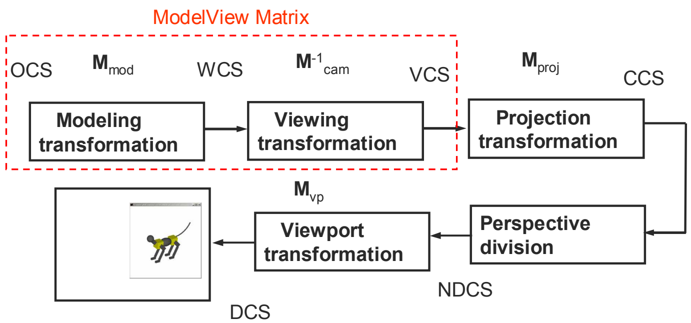
Camera Specification in OpenGL

gluLookAt (eye_x, eye_y, eye_z, ref_x, ref_y, ref_z, up_x, up_y, up_z)

The resulting matrix *post-multiplies* the modeling transformation matrix M

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(ex, ey, ez, rx, ry, rz, ux, uy, uz);
// modeling transformations go here
```

Transformations in the Pipeline



Summary: Modelview Transformations

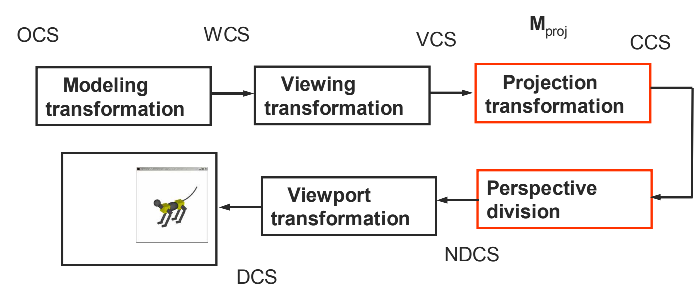
- *Camera transformation as a change of basis*

$$P_{\text{VCS}} = \underbrace{\mathrm{M}_{\text{cam}}^{-1}\mathrm{M}_{\text{mod}}}_{\text{Modelview Transformation}} P_{\text{OCS}}$$

Outline

- Camera Transformations
- ***Projections***
  - ***Orthographic projection (simpler)***
  - Orthographic Viewing Cube
  - Perspective projection, basic idea
  - Perspective Viewing Frostrum

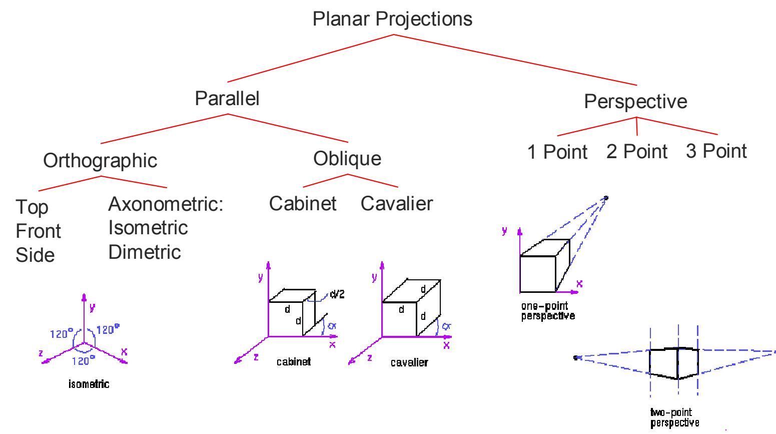
Graphics Pipeline



Projections

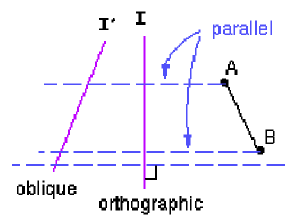
- To lower dimensional space (here 3D → 2D)
- Preserve straight lines
- Trivial example: Drop one coordinate (Orthographic)

Taxonomy and Examples

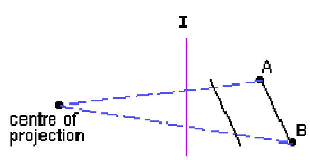


Basic Projections

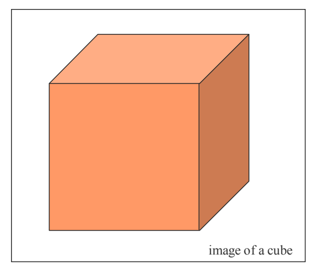
Parallel

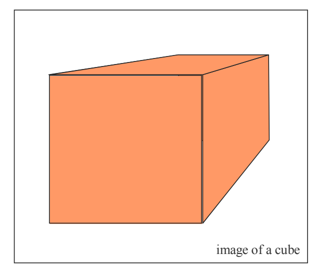


Perspective



Basic Projections





Orthographic Projection

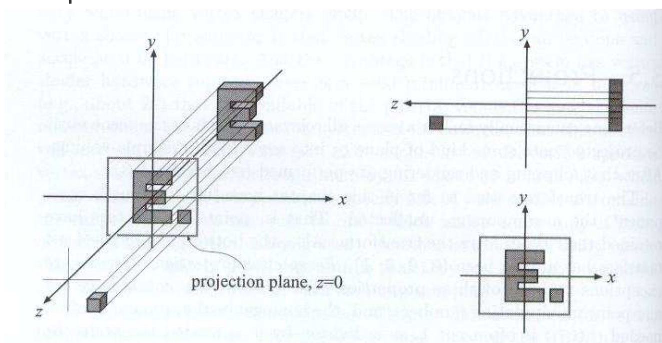
- Drop z-coord
- Parallel lines remain parallel
- Useful for technical drawings etc.

Perspective Projection

- Parallel lines are no longer parallel
- Most common computer graphics, art, visual system

Orthographic Example

- Simply project onto xy plane, then
- Drop z coordinate



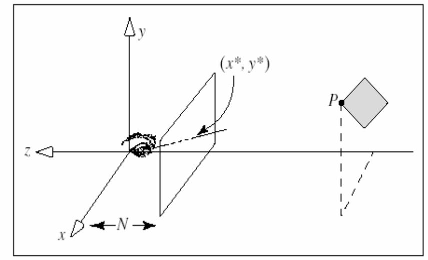
Camera Coordinate System

Camera at (0,0,0)

Looking down –z axis

Image plane = near plane

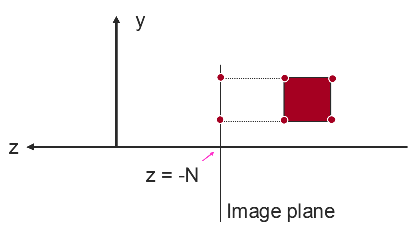
Image plane at z = –N



Orthographic Projection

$P'_x = P_x$
$P'_y = P_y$
$P'_z = -N$

Matrix Form:



$$\begin{bmatrix} P'_x \\ P'_y \\ P'_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -N \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

Outline

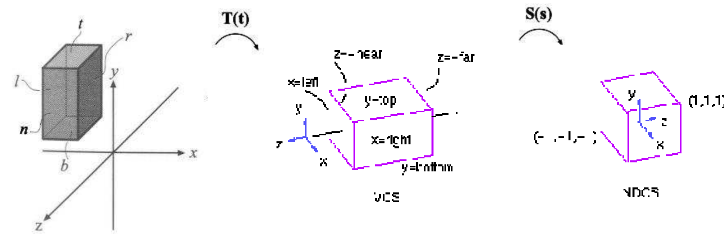
- Camera Transformations
- Projections
  - Orthographic projection (simpler)
  - ***Orthographic Viewing Cube***
  - Perspective projection, basic idea
  - Perspective Viewing Frostrum

Motivation

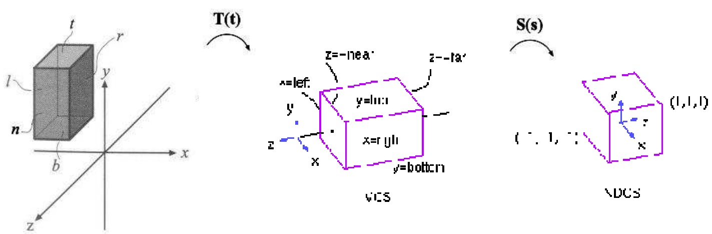
- Viewing volumes are used for clipping (determines if an object is a candidate to be rendered)
- Restricts domain of z stored for visibility test

Orthographic Matrix

- First center cuboid by translating
- Then scale into unit cube



Transformation Matrix



$$
\begin{array}{cc} \text{Scale} & \text{Translation (centering)} \end{array}
$$

$$
M_O = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

Transformation Matrix

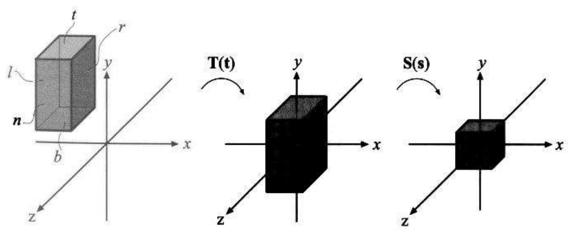
$$
\begin{array}{cc} \text{Scale} & \text{Translation (centering)} \end{array}
$$

$$
M_O = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

$$
M_O = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

Caveats

- Looking down –z, f and n are negative (n > f)
- OpenGL convention: positive n, f, negate internally



Orthographic Transformation - Final Result

$$\mathbf{M}_O = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Opengl Implementation

$$\mathbf{M}_O = \begin{bmatrix} \frac{2}{r-l} & & & -\frac{r+l}{r-l} \\ & \frac{2}{t-b} & & -\frac{t+b}{t-b} \\ & & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ & & & 1 \end{bmatrix}$$

- Looking down –z, f and n are negative (n > f)
- OpenGL convention: positive n, f, negate internally

Final Result

$$\mathbf{M}_O = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad glm::ortho = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \boxed{\frac{-2}{f-n}} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

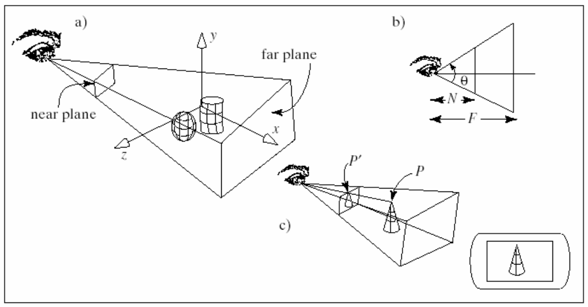
- Looking down –z, f and n are negative (n > f)
- OpenGL convention: positive n, f, negate internally

Outline

- Camera Transformations
- Projections
  - Orthographic projection (simpler)
  - Orthographic Viewing Cube
  - ***Perspective projection, basic idea***
  - Perspective Viewing Frostrum

Perspective Projection

- Most common computer graphics, art, visual system
- Further objects are smaller (size, inverse distance)
- Parallel lines not parallel; converge to single point



Pinhole Camera

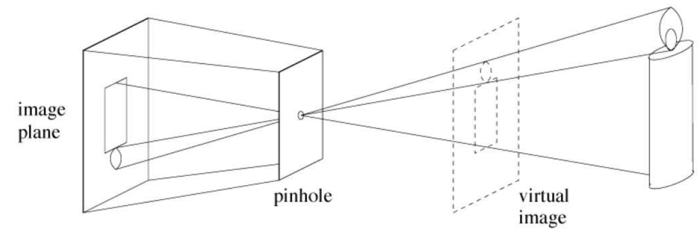
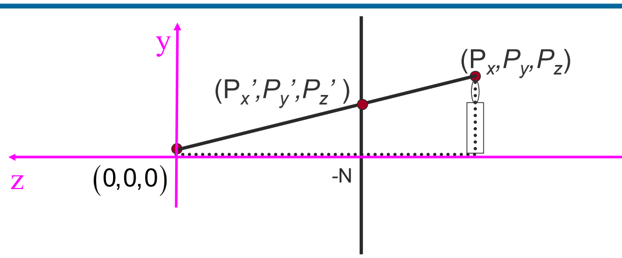


Image from D. Forsyth

- Center of Projection (one point)
- Very common model in graphics
  (but real cameras use lenses; a bit more complicated)

Overhead View of Our Screen



Looks like we've got some nice similar triangles here?

$$P_z' = -n \qquad \frac{P_x}{P_z} = \frac{P_x'}{P_z'} \quad \Rightarrow \quad P_x' = \frac{NP_x}{-P_z}$$

$$\frac{P_y}{P_z} = \frac{P_y'}{P_z'} \quad \Rightarrow \quad P_y' = \frac{NP_y}{-P_z}$$

$$\boxed{\begin{bmatrix} P_x' \\ P_y' \\ P_z' \\ 1 \end{bmatrix} = \begin{bmatrix} P_x N/(-P_z) \\ P_y N/(-P_z) \\ -N \\ 1 \end{bmatrix}}$$

In Homogeneous Matrix Form

Reminder:

$$\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} \rightarrow \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} \xrightarrow{\times w} \underbrace{\begin{bmatrix} wP_x \\ wP_y \\ wP_z \\ w \end{bmatrix}}_{\text{(a line in 4D space)}} \xrightarrow[\div w]{\text{homogenize}} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}$$

Perspective projection:

$$\begin{bmatrix} P_x' \\ P_y' \\ P_z' \\ 1 \end{bmatrix} = \begin{bmatrix} P_x N/(-P_z) \\ P_y N/(-P_z) \\ -N \\ 1 \end{bmatrix}$$

In Homogeneous Matrix Form

Reminder:

$$\begin{bmatrix}P_x\\P_y\\P_z\end{bmatrix} \rightarrow \begin{bmatrix}P_x\\P_y\\P_z\\1\end{bmatrix} \xrightarrow[\times w]{} \begin{bmatrix}wP_x\\wP_y\\wP_z\\w\end{bmatrix} \xrightarrow[\div w]{\text{homogenize}} \begin{bmatrix}P_x\\P_y\\P_z\\1\end{bmatrix} \rightarrow \begin{bmatrix}P_x\\P_y\\P_z\end{bmatrix}$$

(a line in 4D space)

Perspective projection:

$$\begin{bmatrix}P'_x\\P'_y\\P'_z\\1\end{bmatrix} = \begin{bmatrix}P_xN/(-P_z)\\P_yN/(-P_z)\\-N\\1\end{bmatrix} \xrightarrow{\times -P_z/N} \boxed{\begin{bmatrix}P_x\\P_y\\P_z\\-P_z/N\end{bmatrix} = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&-1/N&0\end{bmatrix}\begin{bmatrix}P_x\\P_y\\P_z\\1\end{bmatrix}}$$

Therefore:

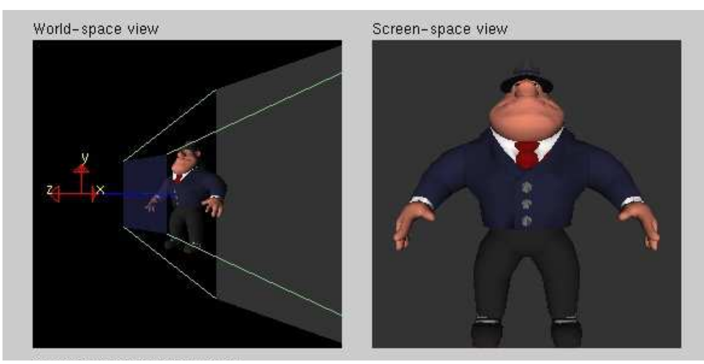
$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&-1/N&0\end{bmatrix}\begin{bmatrix}P_x\\P_y\\P_z\\1\end{bmatrix} \xrightarrow[\div -P_z/N]{\text{and then: homogenize}} \begin{bmatrix}P'_x\\P'_y\\P'_z\\1\end{bmatrix}$$

Homogenization step:
"Perspective Division"
(divide by $w = -P_z/N$)

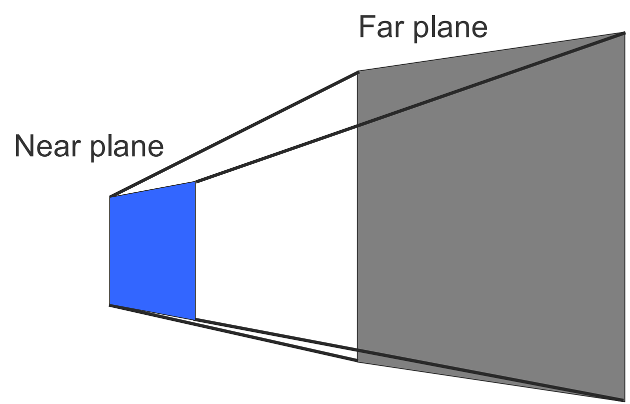
Outline

- Camera Transformations
- Projections
  - Orthographic projection (simpler)
  - Orthographic Viewing Cube
  - Perspective projection, basic idea
  - ***Perspective Viewing Frustrum***

Perspective: Viewing Frustrum



Viewing Frustum

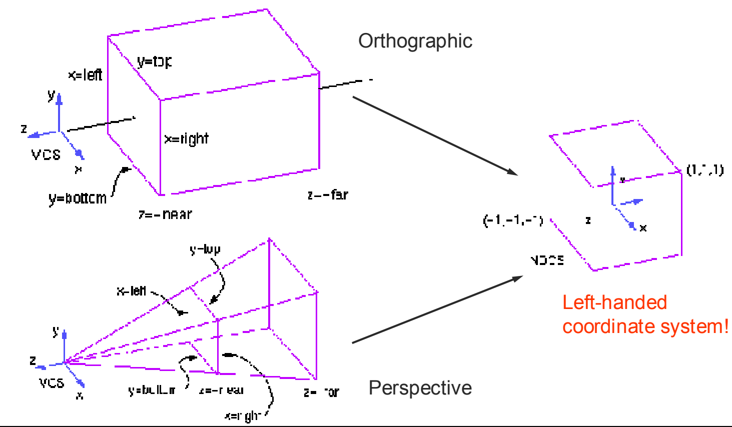


Motivation

- Viewing volumes are used for clipping (determines if an object is a candidate to be rendered)
- Restricts domain of z stored for visibility test

Canonical View Volumes

- standardized viewing volume representation



Why Canonical View Volumes?

- permits standardization
  - clipping
    - easier to determine if an arbitrary point is enclosed in volume with canonical view volume vs. clipping to six arbitrary planes
  - rendering
    - projection and rasterization algorithms can be reused

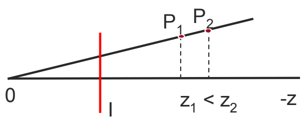
Normalized Device Coordinates

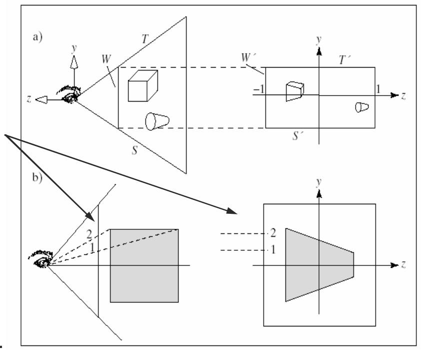
- convention
  - viewing frustum mapped to specific parallelepiped
    - Normalized Device Coordinates (NDC)
    - same as clipping coords
  - only objects inside the parallelepiped get rendered
  - which parallelepiped?
    - depends on rendering system

Derivation: Perspective Transformation Matrix

Warps the view volume and the objects in it

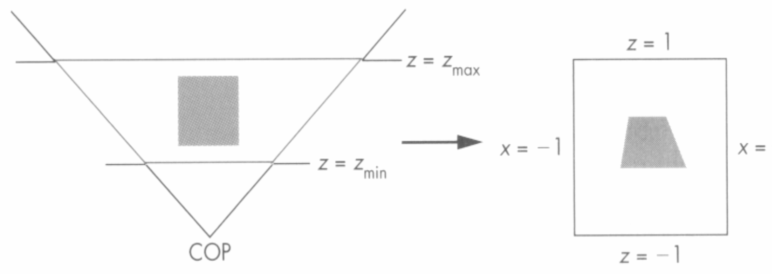
- Eye becomes a point at infinity, and the projection rays become **parallel lines** (i.e., orthographic projection)
- We also want to keep z
  - Why? Pseudodepth



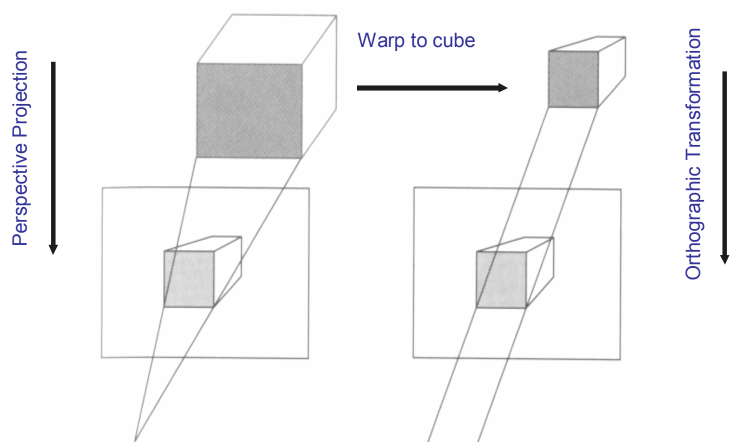


Perspective Normalization

- perspective viewing frustum transformed to cube
- orthographic rendering of cube produces same

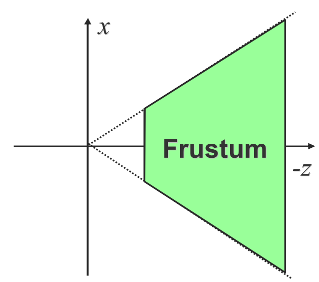


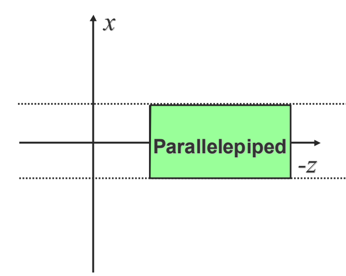
Perspective Projection vs Perspective Warp



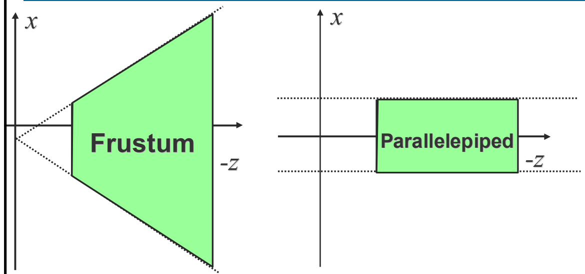
Perspective to Orthographic

- transformation of space
  - center of projection moves to infinity
  - view volume transformed
    - from frustum (truncated pyramid) to parallelepiped (box)





Derivation: Perspective Transformation Matrix



$$\begin{bmatrix} \frac{P_x n}{P_z} \\ \frac{P_y n}{P_z} \\ n+f-\frac{fn}{P_z} \\ 1 \end{bmatrix}$$

Therefore:

$$M_P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{n+f}{n} & -f \\ 0 & 0 & \frac{1}{n} & 0 \end{bmatrix} \quad \text{or} \quad M_P = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Orthographic Transformation - Final Result

Opengl Implementation

$$\mathbf{M}_O = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{M}_O = \begin{bmatrix} \frac{2}{r-l} & & & -\frac{r+l}{r-l} \\ & \frac{2}{t-b} & & -\frac{t+b}{t-b} \\ & & \frac{2}{n-f} & -\frac{f+n}{n-f} \\ & & & 1 \end{bmatrix}$$

- Looking down $-z$, f and n are negative ($n > f$)
- OpenGL convention: positive n, f, negate internally

The Projection Matrix

$$\mathbf{M}_{\text{proj}} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{M}_{\text{proj}} = \mathbf{M}_O \mathbf{M}_P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The Projection Matrix

Opengl Implementation

$$\mathbf{M}_{\text{proj}} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{f+n}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{M}_{\text{proj}} = \mathbf{M}_O \mathbf{M}_P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Graphics Pipeline