

CSE 167:
Introduction to Computer Graphics
Lecture #4: Vertex Transformation

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2012

Announcements

- ▶ Project 2 due Friday, October 12
- ▶ Remaining homework grades should go on-line today or tomorrow
 - ▶ Compare to your notes and let us know if they don't match
- ▶ To get graded early, please see the TA Sid during his office hours

Lecture Overview

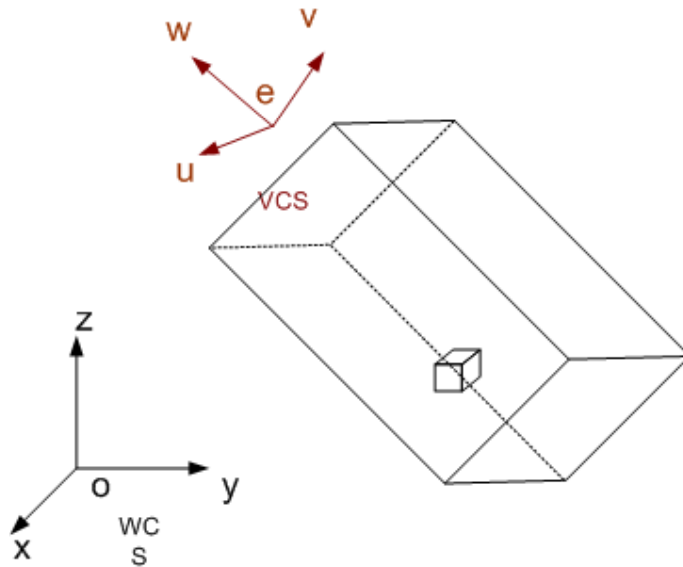
- ▶ **View Volumes**
- ▶ Vertex Transformation
- ▶ Rendering Pipeline

View Volumes

- ▶ View volume = 3D volume seen by camera

Orthographic view volume

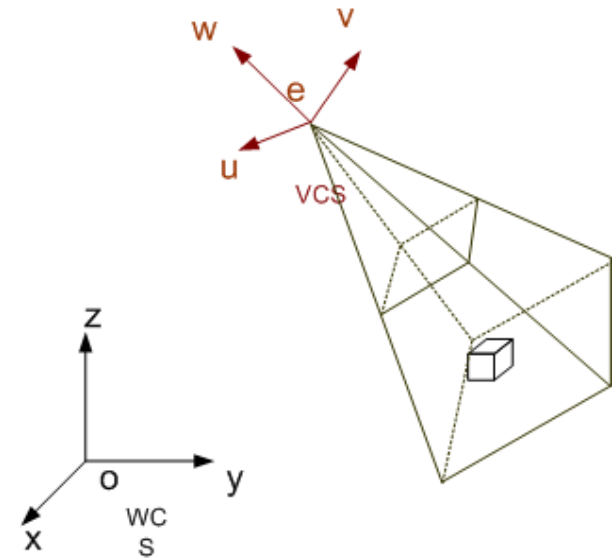
Camera coordinates



World coordinates

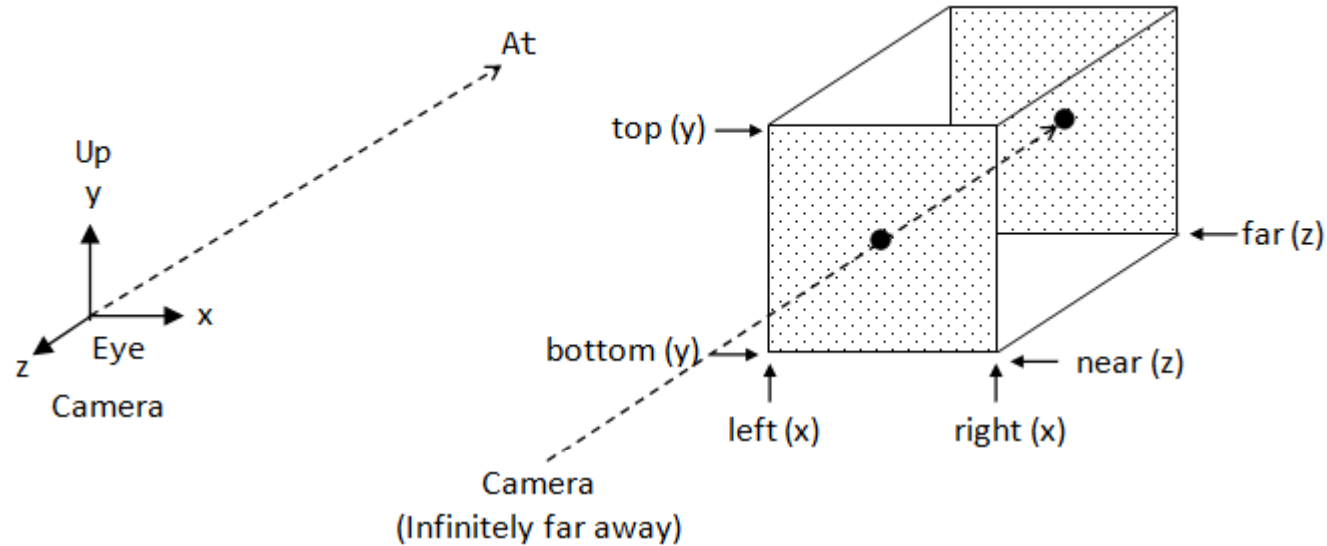
Perspective view volume

Camera coordinates



World coordinates

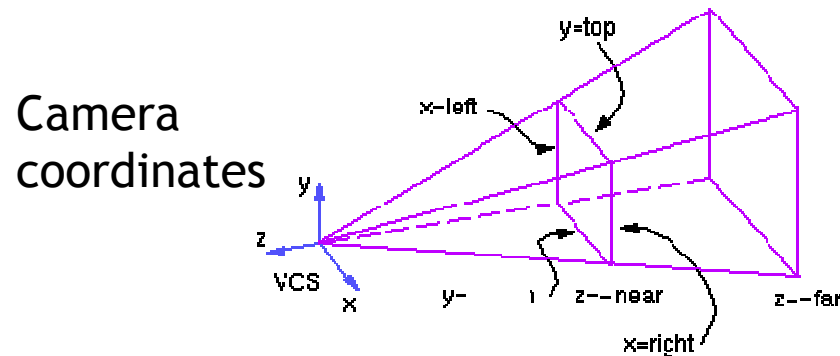
Orthographic View Volume



- ▶ Specified by 6 parameters:
 - ▶ Right, left, top, bottom, near, far
- ▶ Or, if symmetrical:
 - ▶ Width, height, near, far

Perspective View Volume

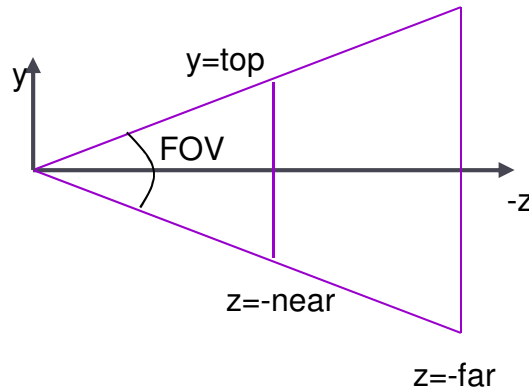
General view volume



- ▶ Defined by 6 parameters, in camera coordinates
 - ▶ Left, right, top, bottom boundaries
 - ▶ Near, far clipping planes
- ▶ Clipping planes to avoid numerical problems
 - ▶ Divide by zero
 - ▶ Low precision for distant objects
- ▶ Usually symmetric, i.e., left=-right, top=-bottom

Perspective View Volume

Symmetrical view volume



- ▶ Only 4 parameters
 - ▶ Vertical field of view (FOV)
 - ▶ Image aspect ratio (width/height)
 - ▶ Near, far clipping planes

$$\text{aspect ratio} = \frac{\text{right} - \text{left}}{\text{top} - \text{bottom}} = \frac{\text{right}}{\text{top}}$$

$$\tan(\text{FOV} / 2) = \frac{\text{top}}{\text{near}}$$

Canonical View Volume

- ▶ **Goal: create projection matrix so that**
 - ▶ User defined view volume is transformed into canonical view volume: cube $[-1,1] \times [-1,1] \times [-1,1]$
 - ▶ Multiplying corner vertices of view volume by projection matrix and performing homogeneous divide yields corners of canonical view volume
- ▶ **Perspective and orthographic projection are treated the same way**
- ▶ **Canonical view volume is last stage in which coordinates are in 3D**
 - ▶ Next step is projection to 2D frame buffer

Lecture Overview

- ▶ View Volumes
- ▶ **Vertex Transformation**
- ▶ Rendering Pipeline

Projection Matrix

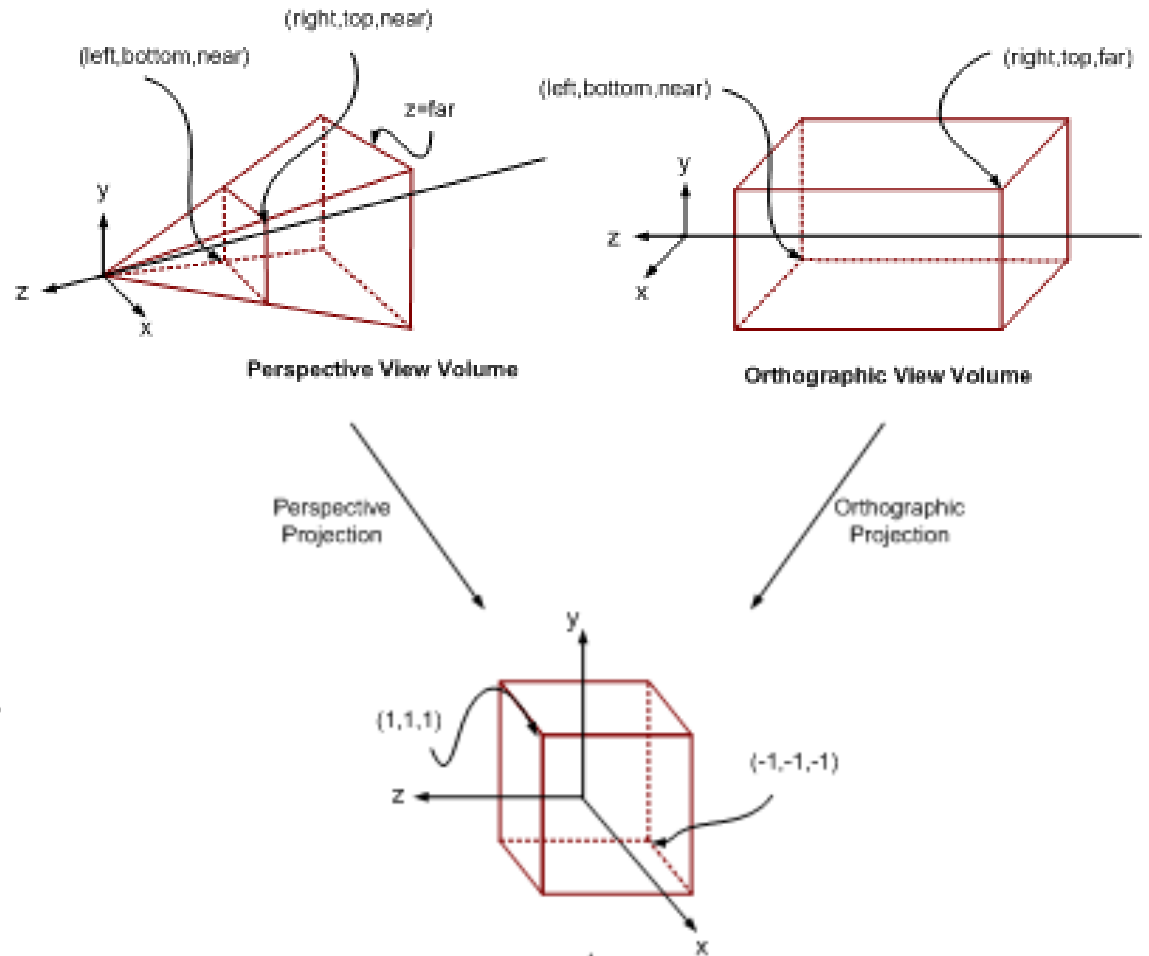
Camera coordinates



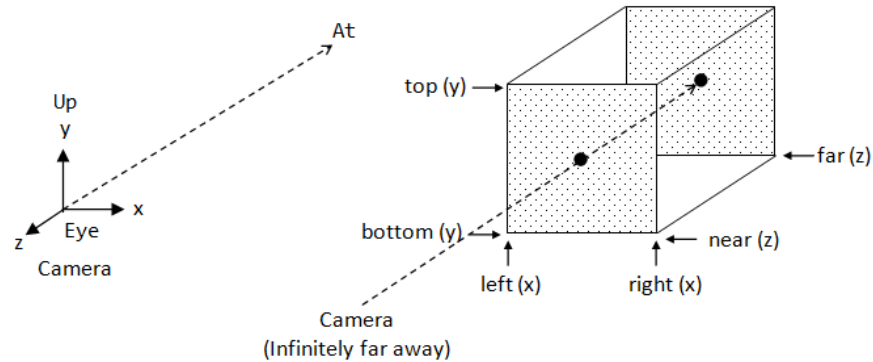
Projection matrix



Canonical view volume



Orthographic Projection Matrix



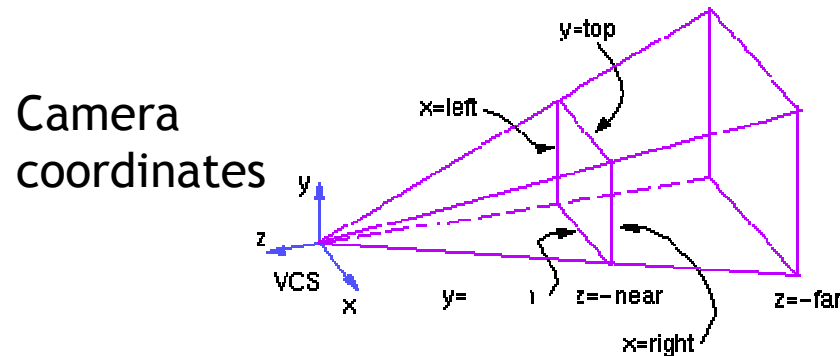
In OpenGL:
glOrtho(left, right, bottom, top, near, far)

$$\mathbf{P}_{ortho}(right, left, top, bottom, near, far) = \begin{bmatrix} \frac{2}{right - left} & 0 & 0 & -\frac{right + left}{right - left} \\ 0 & \frac{2}{top - bottom} & 0 & -\frac{top + bottom}{top - bottom} \\ 0 & 0 & \frac{2}{far - near} & -\frac{far + near}{far - near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{P}_{ortho}(width, height, near, far) = \begin{bmatrix} \frac{2}{width} & 0 & 0 & 0 \\ 0 & \frac{2}{height} & 0 & 0 \\ 0 & 0 & \frac{2}{far - near} & -\frac{far + near}{far - near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective Projection Matrix

- ▶ General view frustum with 6 parameters



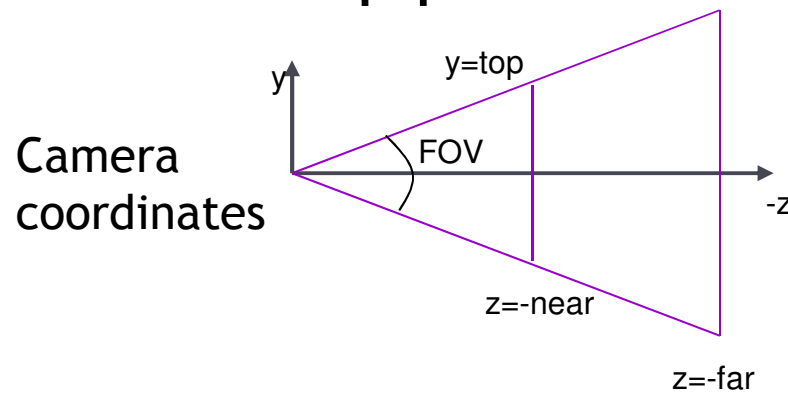
$$\mathbf{P}_{persp}(left, right, top, bottom, near, far) =$$

$$\begin{bmatrix} \frac{2near}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & \frac{-(far+near)}{far-near} & \frac{-2far \cdot near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

In OpenGL:
glFrustum(left, right, bottom, top, near, far)

Perspective Projection Matrix

- ▶ Symmetrical view frustum with field of view, aspect ratio, near and far clip planes



$$\mathbf{P}_{persp}(FOV, aspect, near, far) = \begin{bmatrix} \frac{1}{aspect \cdot \tan(FOV / 2)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(FOV / 2)} & 0 & 0 \\ 0 & 0 & \frac{near + far}{near - far} & \frac{2 \cdot near \cdot far}{near - far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

In OpenGL:

`gluPerspective(fov, aspect, near, far)`

Viewport Transformation

- ▶ After applying projection matrix, scene points are in *normalized viewing coordinates*
 - ▶ Per definition within range $[-1..1] \times [-1..1] \times [-1..1]$
- ▶ Next is projection from 3D to 2D (not reversible)
- ▶ Normalized viewing coordinates can be mapped to image (=pixel=frame buffer) coordinates
 - ▶ Range depends on window (view port) size:
 $[x_0...x_1] \times [y_0...y_1]$
- ▶ Scale and translation required:

$$\mathbf{D}(x_0, x_1, y_0, y_1) = \begin{bmatrix} (x_1 - x_0)/2 & 0 & 0 & (x_0 + x_1)/2 \\ 0 & (y_1 - y_0)/2 & 0 & (y_0 + y_1)/2 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Complete Vertex Transformation

- ▶ Mapping a 3D point in object coordinates to pixel coordinates:

$$\mathbf{p}' = \mathbf{DPC}^{-1}\mathbf{M}\mathbf{p}$$

|
Object space

- ▶ **M**: Object-to-world matrix
- ▶ **C**: camera matrix
- ▶ **P**: projection matrix
- ▶ **D**: viewport matrix

Complete Vertex Transformation

- ▶ Mapping a 3D point in object coordinates to pixel coordinates:

$$\mathbf{p}' = \mathbf{DPC}^{-1} \mathbf{M} \mathbf{p}$$

|
| Object space
|
| World space

- ▶ **M**: Object-to-world matrix
- ▶ **C**: camera matrix
- ▶ **P**: projection matrix
- ▶ **D**: viewport matrix

Complete Vertex Transformation

- ▶ Mapping a 3D point in object coordinates to pixel coordinates:

$$\mathbf{p}' = \mathbf{DPC}^{-1}\mathbf{M}\mathbf{p}$$

Object space
World space
Camera space

- ▶ **M**: Object-to-world matrix
- ▶ **C**: camera matrix
- ▶ **P**: projection matrix
- ▶ **D**: viewport matrix

Complete Vertex Transformation

- ▶ Mapping a 3D point in object coordinates to pixel coordinates:

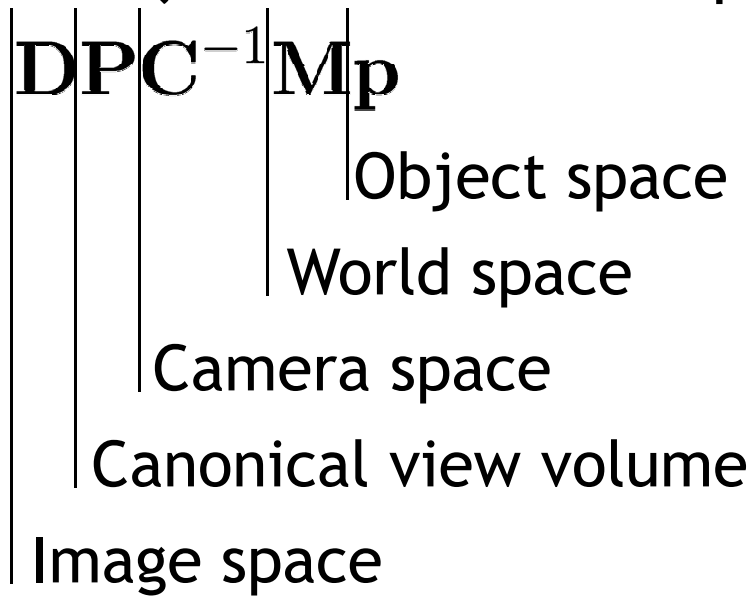
$$\mathbf{p}' = \mathbf{D} \mathbf{P} \mathbf{C}^{-1} \mathbf{M} \mathbf{p}$$

Object space
World space
Camera space
Canonical view volume

- ▶ **M**: Object-to-world matrix
- ▶ **C**: camera matrix
- ▶ **P**: projection matrix
- ▶ **D**: viewport matrix

Complete Vertex Transformation

- ▶ Mapping a 3D point in object coordinates to pixel coordinates: $\mathbf{p}' = \mathbf{DPC}^{-1}\mathbf{M}\mathbf{p}$



- ▶ **M**: Object-to-world matrix
- ▶ **C**: camera matrix
- ▶ **P**: projection matrix
- ▶ **D**: viewport matrix

Complete Vertex Transformation

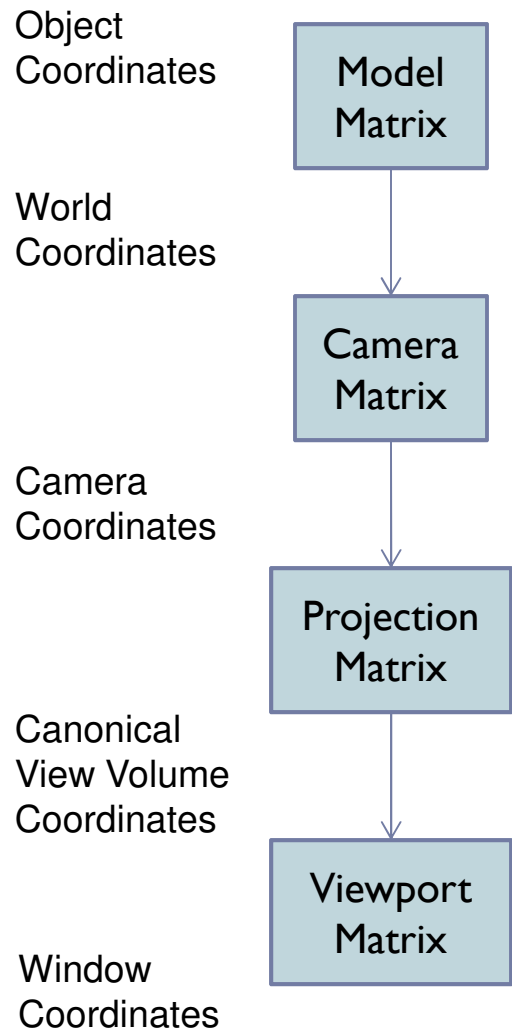
- ▶ Mapping a 3D point in object coordinates to pixel coordinates:

$$\mathbf{p}' = \mathbf{DPC}^{-1}\mathbf{Mp}$$

$$\mathbf{p}' = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} \quad \text{Pixel coordinates: } \begin{matrix} x'/w' \\ y'/w' \end{matrix}$$

- ▶ **M**: Object-to-world matrix
- ▶ **C**: camera matrix
- ▶ **P**: projection matrix
- ▶ **D**: viewport matrix

The Complete Vertex Transformation



Complete Vertex Transformation in OpenGL

- ▶ Mapping a 3D point in object coordinates to pixel coordinates:

OpenGL `GL_MODELVIEW` matrix

$$\mathbf{p}' = \mathbf{D}\mathbf{P}\mathbf{C}^{-1}\mathbf{M}\mathbf{p}$$

OpenGL `GL_PROJECTION` matrix

- ▶ **M**: Object-to-world matrix
- ▶ **C**: camera matrix
- ▶ **P**: projection matrix
- ▶ **D**: viewport matrix

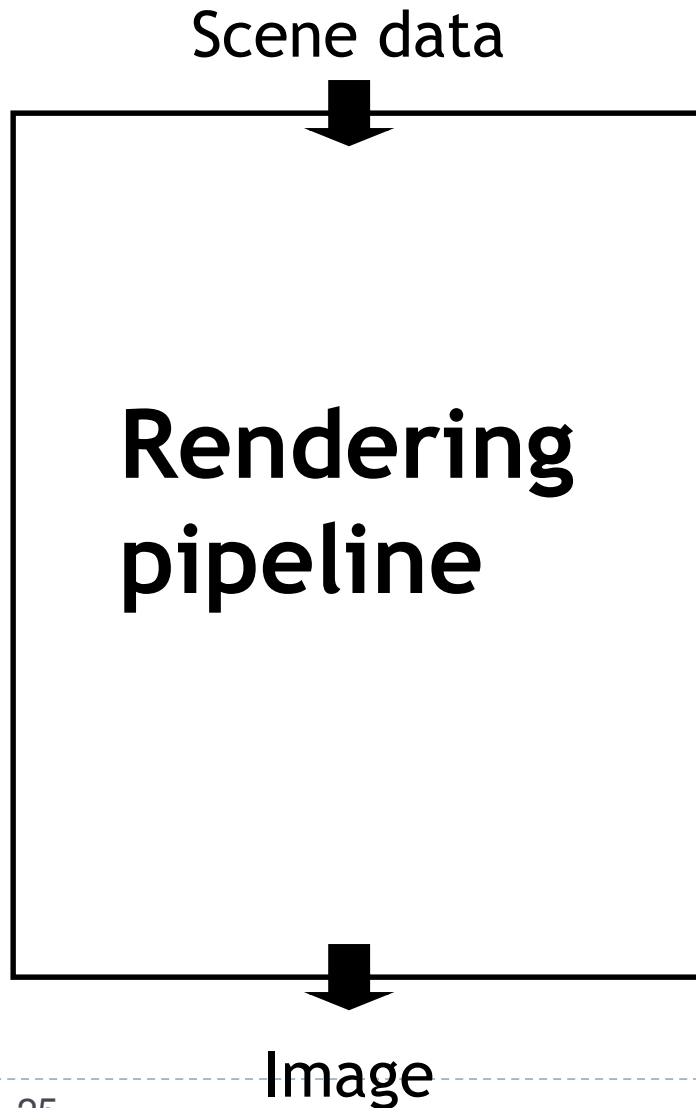
Complete Vertex Transformation in OpenGL

- ▶ **GL_MODELVIEW, $C^{-1}M$**
 - ▶ Defined by programmer
- ▶ **GL_PROJECTION, P**
 - ▶ Utility routines to set it by specifying view volume: `glFrustum()`, `glPerspective()`, `glOrtho()`
 - ▶ Do not use utility functions in homework project 2
- ▶ **Viewport, D**
 - ▶ Specify implicitly via `glViewport()`
 - ▶ No direct access with equivalent to `GL_MODELVIEW` or `GL_PROJECTION`

Lecture Overview

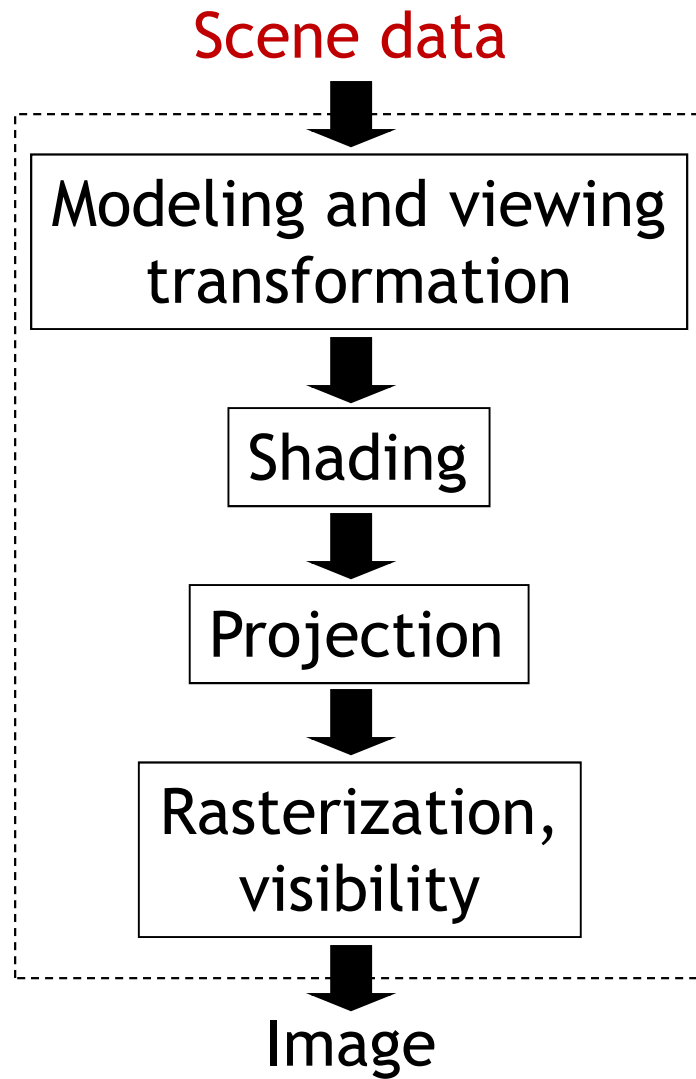
- ▶ View Volumes
- ▶ Vertex Transformation
- ▶ **Rendering Pipeline**

Rendering Pipeline

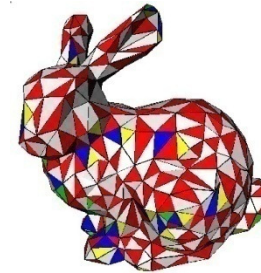


- ▶ Hardware and software which draws 3D scenes on the screen
- ▶ Consists of several stages
 - ▶ Simplified version here
- ▶ Most operations performed by specialized hardware (GPU)
- ▶ Access to hardware through low-level 3D API (OpenGL, DirectX)
- ▶ All scene data flows through the pipeline at least once for each frame

Rendering Pipeline

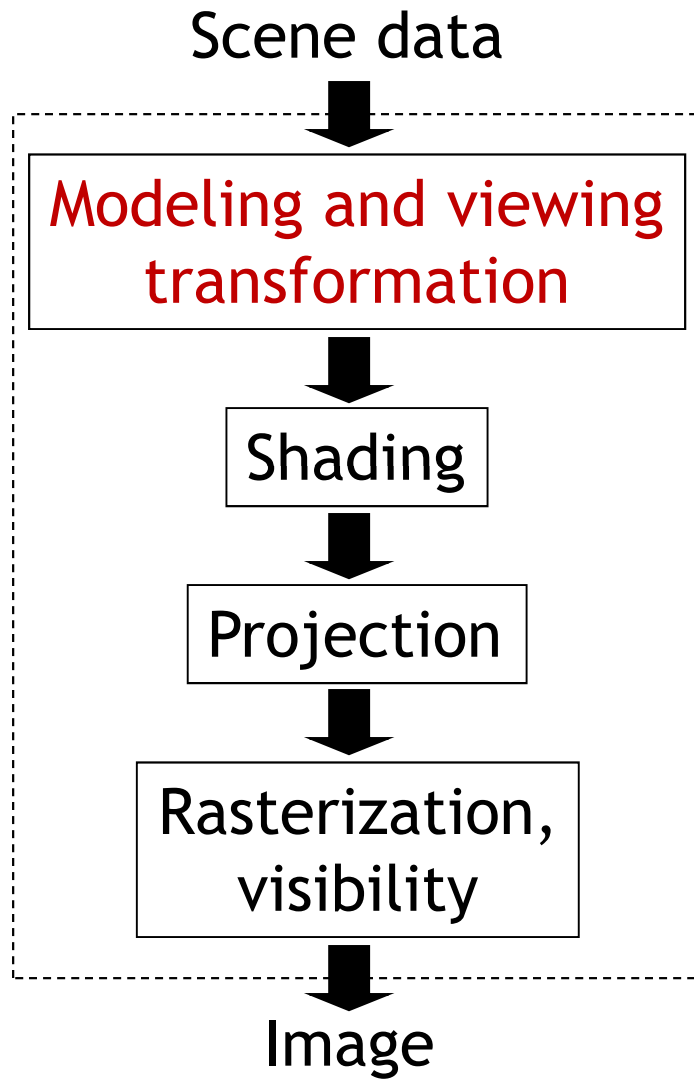


- ▶ Textures, lights, etc.
- ▶ Geometry
 - ▶ Vertices and how they are connected
 - ▶ Triangles, lines, points, triangle strips
 - ▶ Attributes such as color



- ▶ Specified in object coordinates
- ▶ Processed by the rendering pipeline one-by-one

Rendering Pipeline

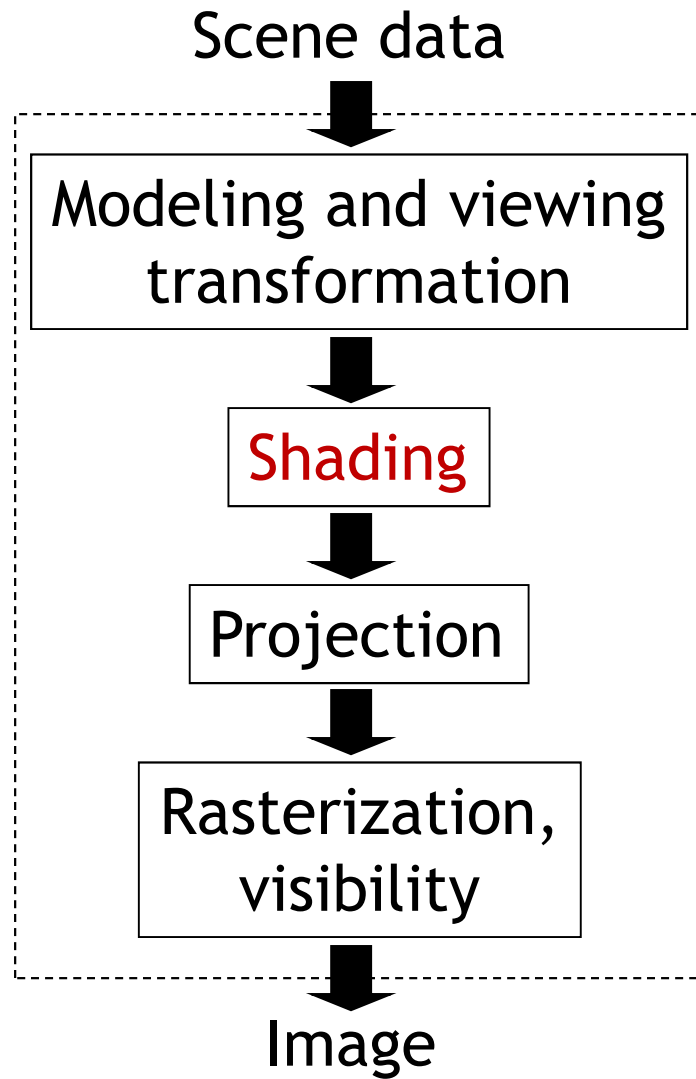


- ▶ Transform object to camera coordinates
- ▶ Specified by `GL_MODELVIEW` matrix in OpenGL
- ▶ User computes `GL_MODELVIEW` matrix as discussed

$$\mathbf{P}_{camera} = \mathbf{C}^{-1} \mathbf{M} \mathbf{P}_{object}$$

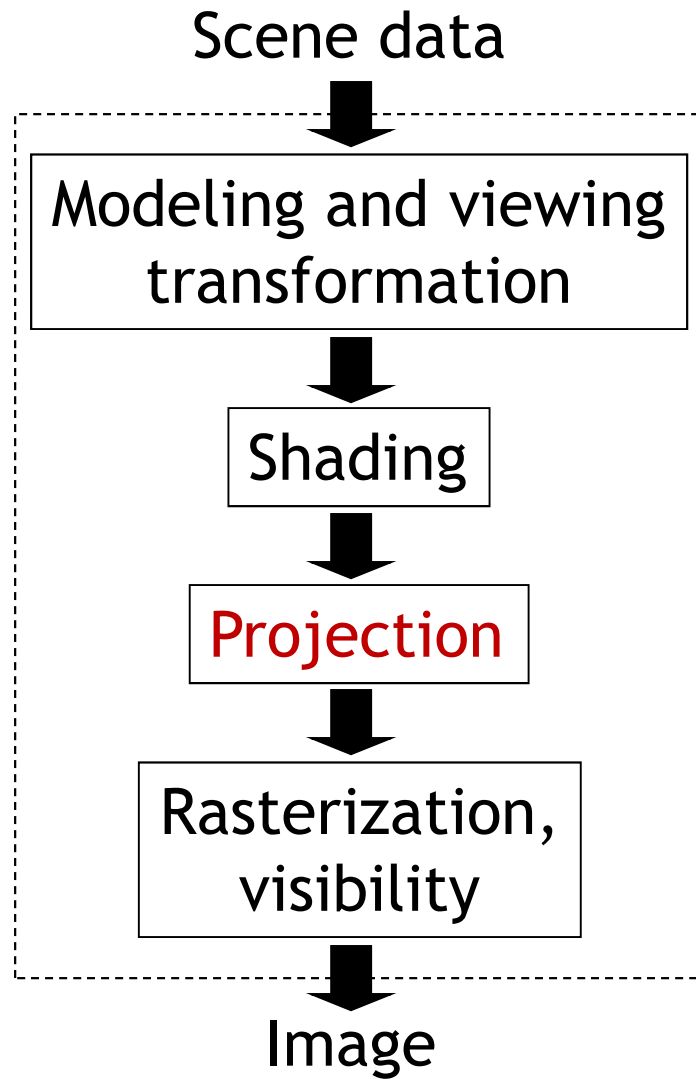
MODELVIEW matrix

Rendering Pipeline



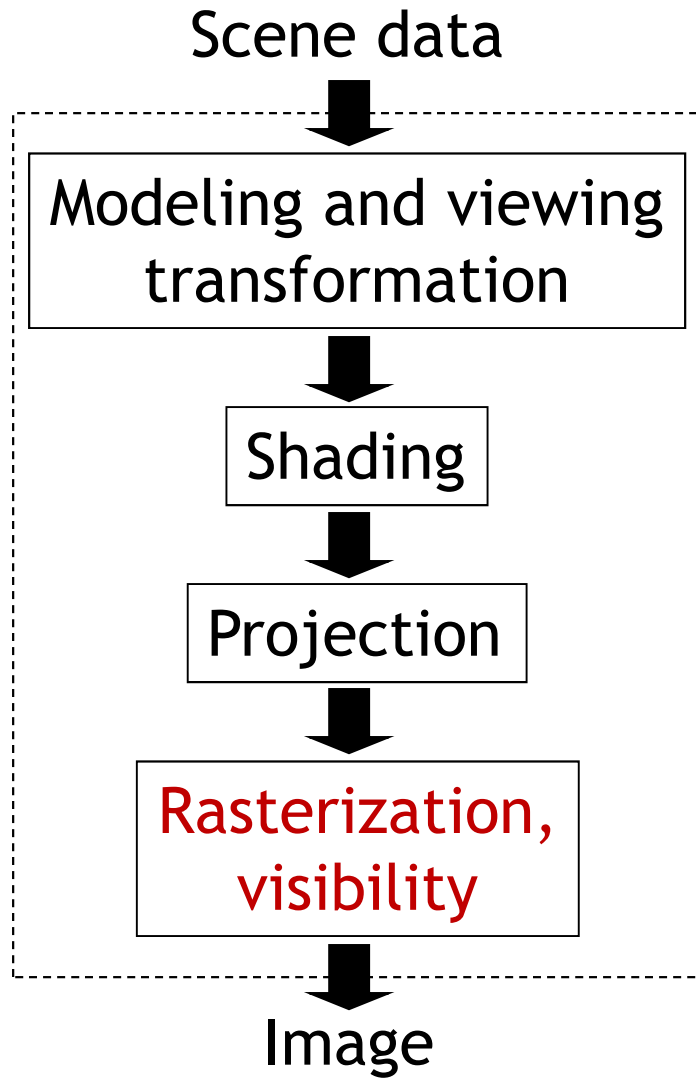
- ▶ Look up light sources
- ▶ Compute color for each vertex
- ▶ Covered later in the course

Rendering Pipeline

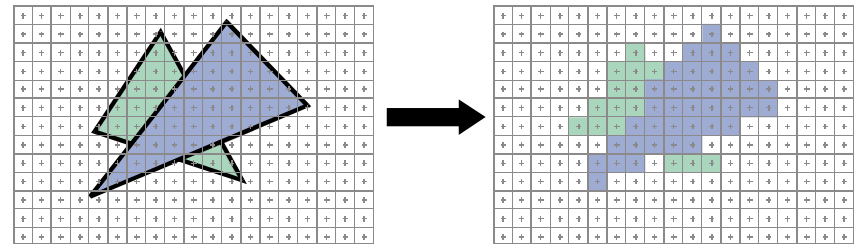


- ▶ Project 3D vertices to 2D image positions
- ▶ `GL_PROJECTION` matrix

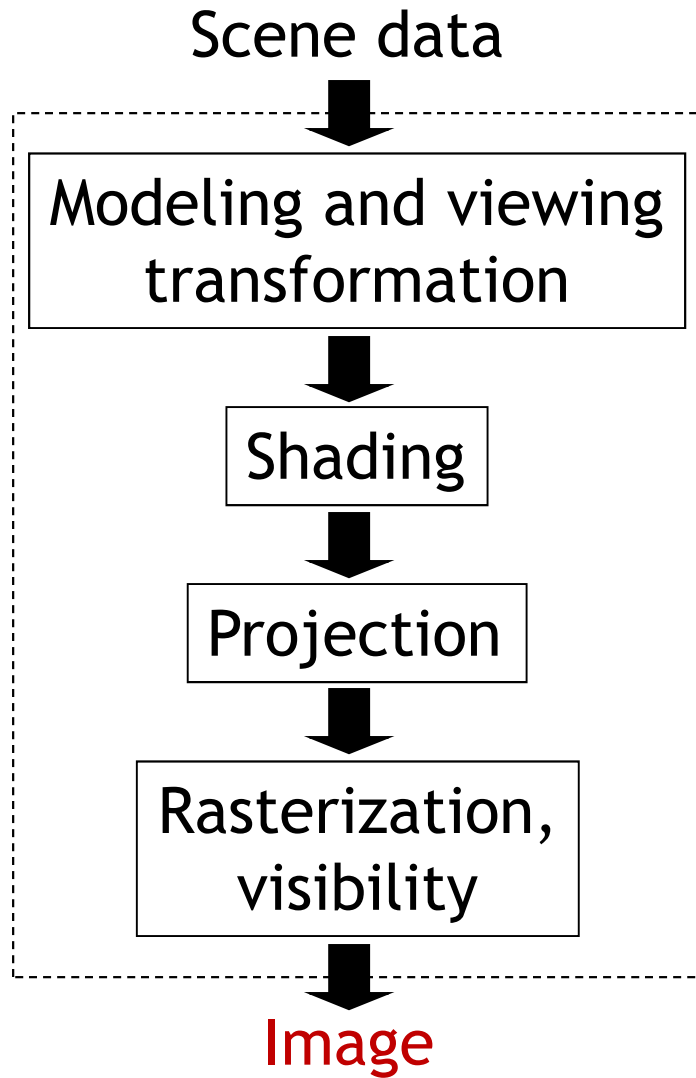
Rendering Pipeline



- ▶ Draw primitives (triangles, lines, etc.)
- ▶ Determine what is visible

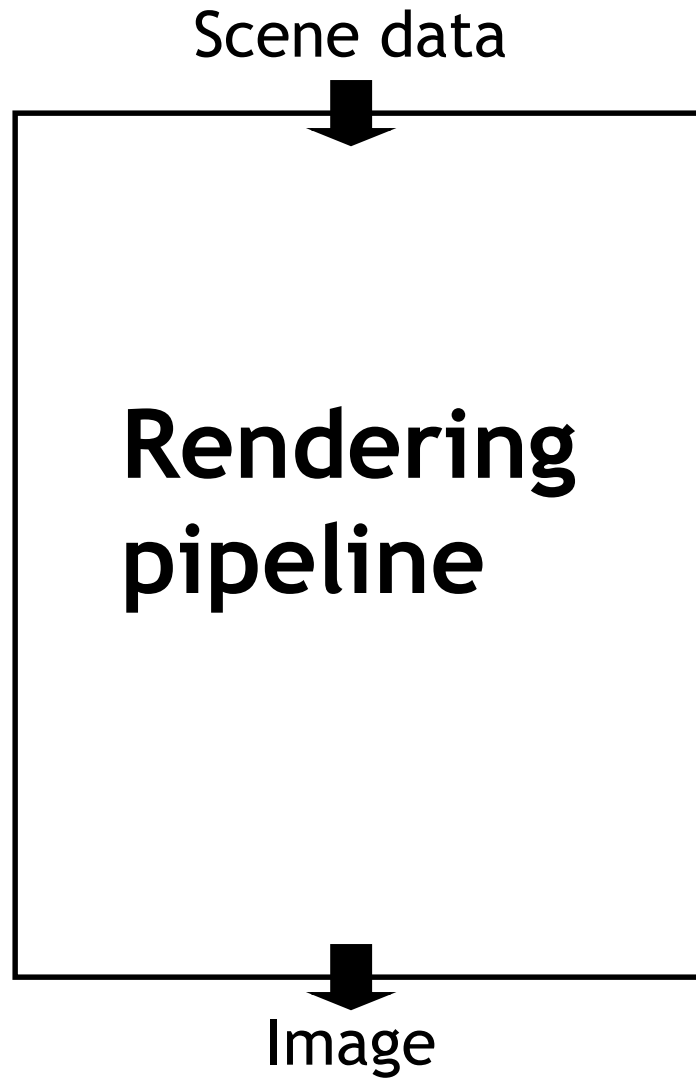


Rendering Pipeline



▶ Pixel colors

Rendering Engine



- ▶ Additional software layer encapsulating low-level API
- ▶ Higher level functionality than OpenGL
- ▶ Platform independent
- ▶ Layered software architecture common in industry
 - ▶ Game engines
 - ▶ Graphics middleware