

CSE 167:
Introduction to Computer Graphics
Lecture #7: Projection

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2019

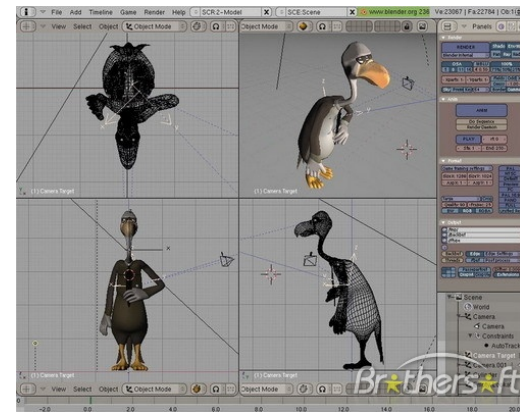
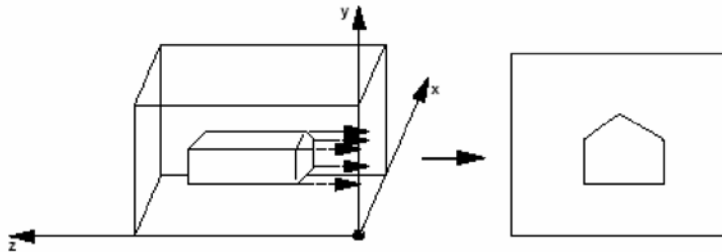


Projection



Projection

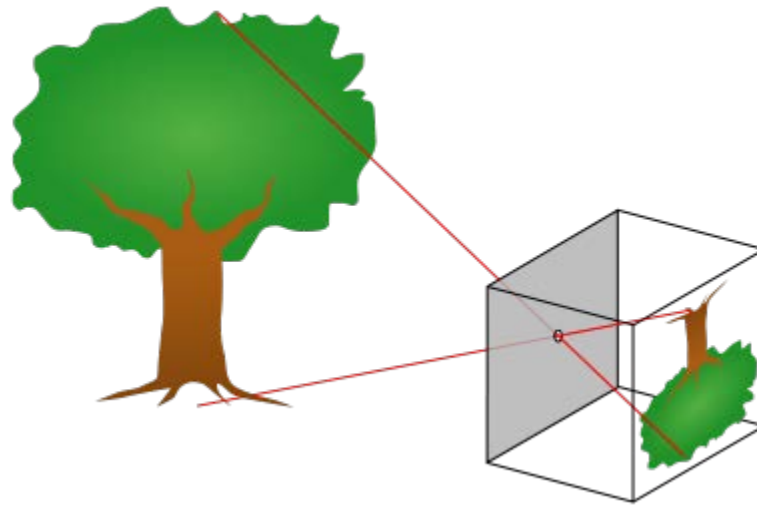
- ▶ Goal:
Given 3D points (vertices) in camera coordinates, determine corresponding image coordinates
- ▶ Transforming 3D points into 2D is called Projection
- ▶ Typically one of two types of projection is used:
 - ▶ Orthographic Projection (=Parallel Projection)



- ▶ Perspective Projection: most commonly used

Perspective Projection

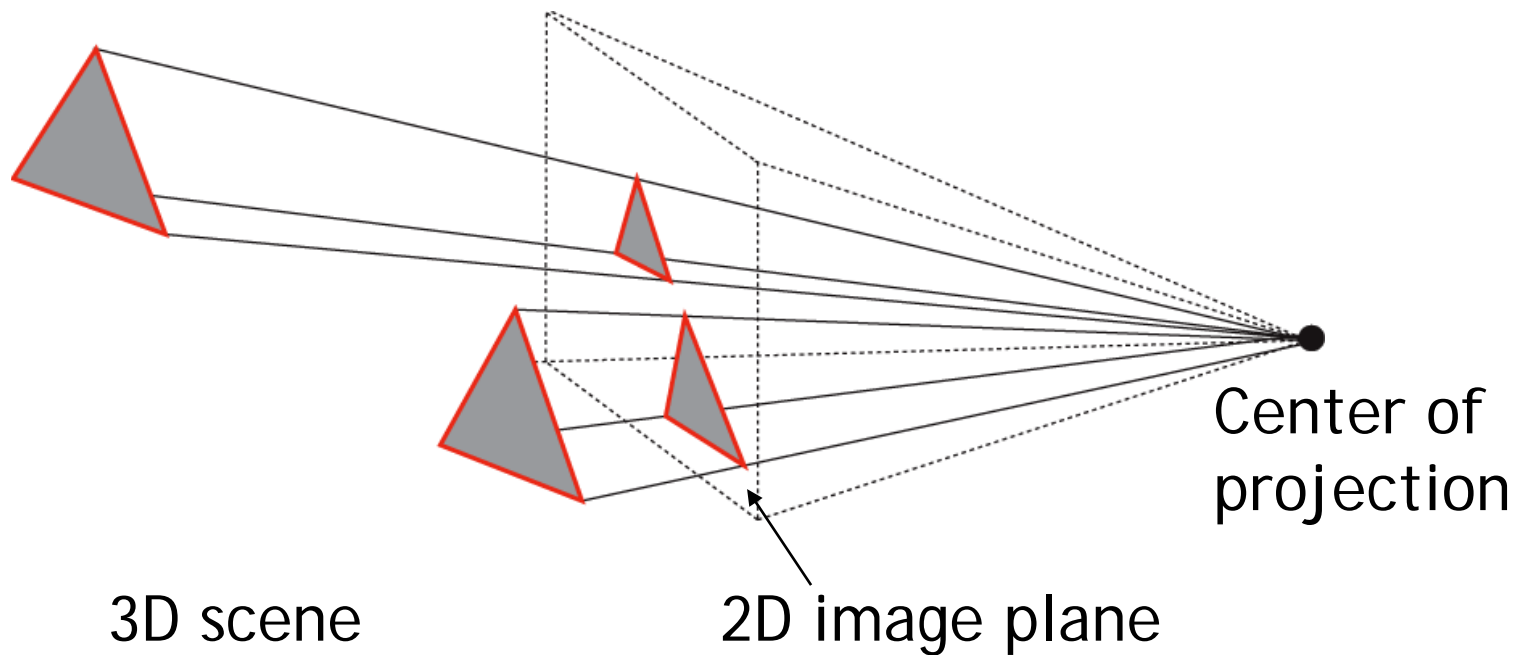
- ▶ Most common for computer graphics
- ▶ Simplified model of human eye, or camera lens (*pinhole camera*)



- ▶ Things farther away appear to be smaller
- ▶ Discovery attributed to Filippo Brunelleschi (Italian architect) in the early 1400's

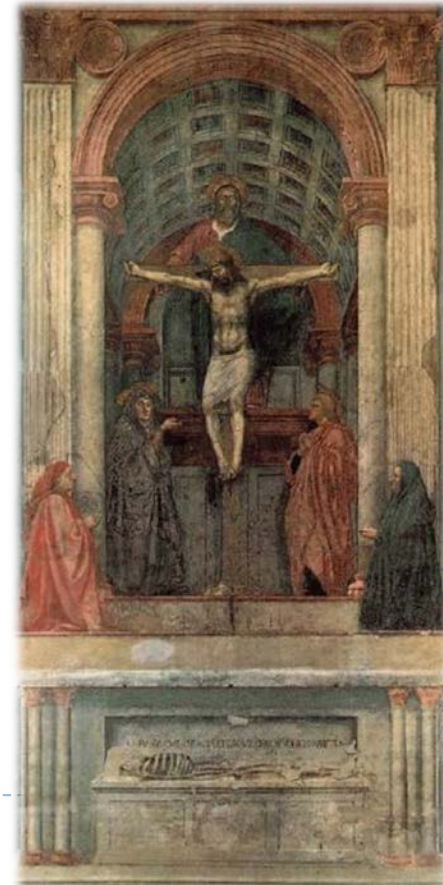
Perspective Projection

- ▶ Project along rays that converge in center of projection



Perspective Projection

Parallel lines are no longer parallel, converge in one point



Earliest example:

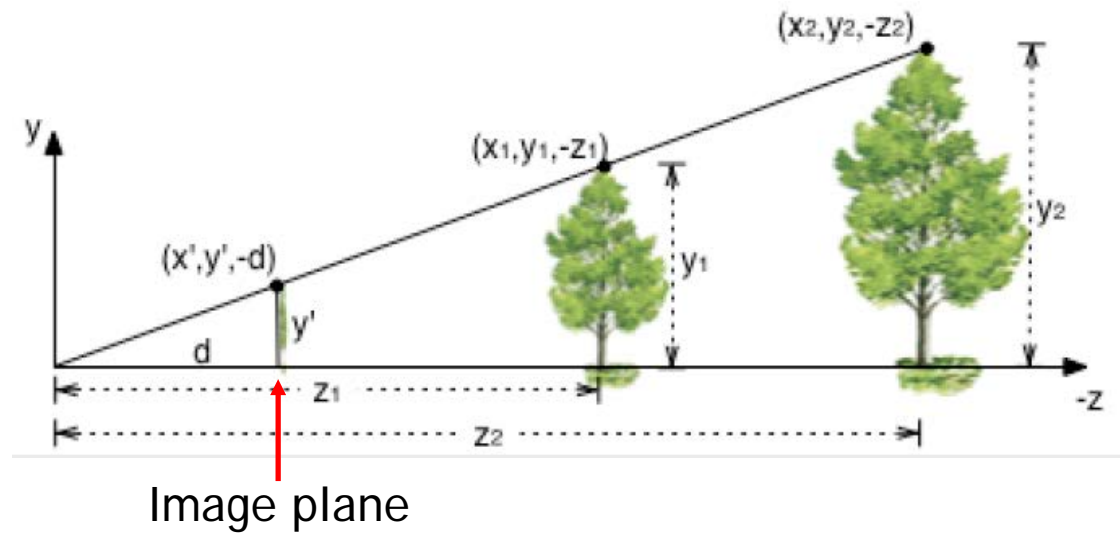
La Trinitá (1427) by Masaccio

Perspective Projection

From law of ratios in similar triangles follows:

$$\frac{y'}{d} = \frac{y_1}{z_1} \rightarrow y' = \frac{y_1 d}{z_1}$$

Similarly: $x' = \frac{x_1 d}{z_1}$



By definition: $z' = d$

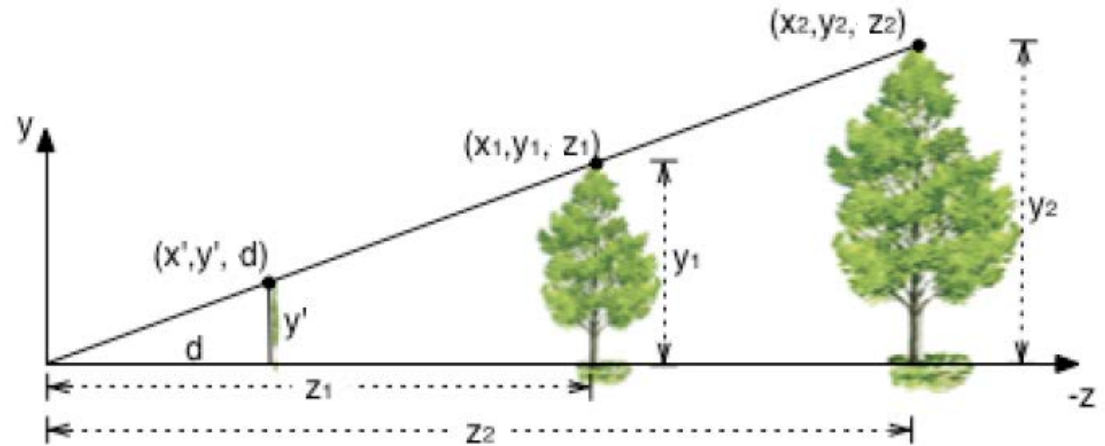
- ▶ We can express this using homogeneous coordinates and 4x4 matrices as follows

Perspective Projection

$$x' = \frac{x_1 d}{z_1}$$

$$y' = \frac{y_1 d}{z_1}$$

$$z' = d$$



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \rightarrow \begin{bmatrix} xd/z \\ yd/z \\ d \\ 1 \end{bmatrix}$$

Projection matrix

Homogeneous division

Perspective Projection

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} = \begin{bmatrix} xd/z \\ yd/z \\ d \\ 1 \end{bmatrix}$$

Projection matrix P

- ▶ Using projection matrix, homogeneous division seems more complicated than just multiplying all coordinates by d/z , so why do it?
- ▶ It will allow us to:
 - ▶ Handle different types of projections in a unified way
 - ▶ Define arbitrary view volumes

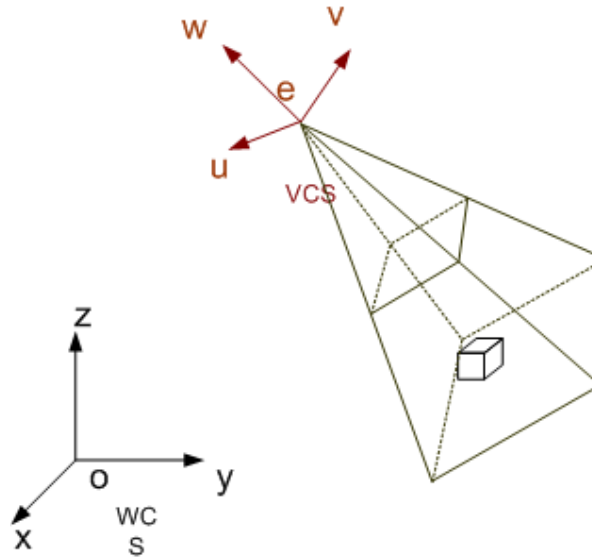
Topics

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

View Volume

- ▶ View volume = 3D volume seen by camera

Camera coordinates



World coordinates

Projection Matrix

Camera coordinates

*Projection
matrix*

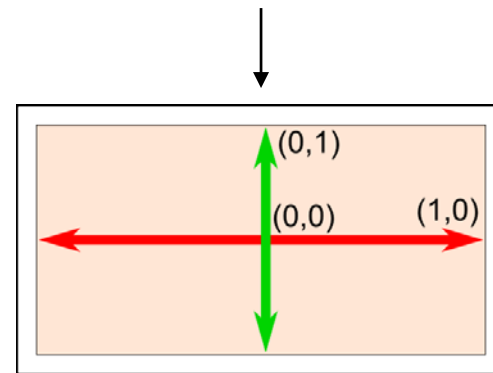
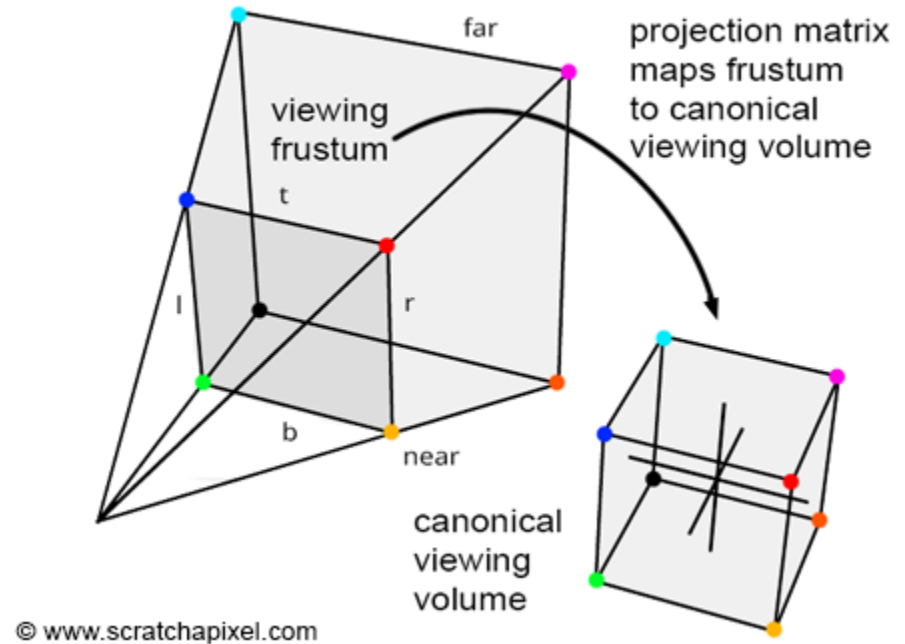


Canonical view volume

*Viewport
transformation*

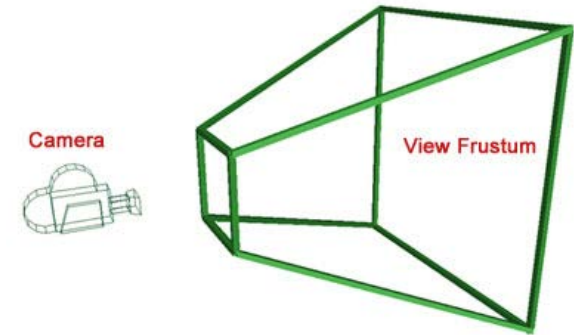
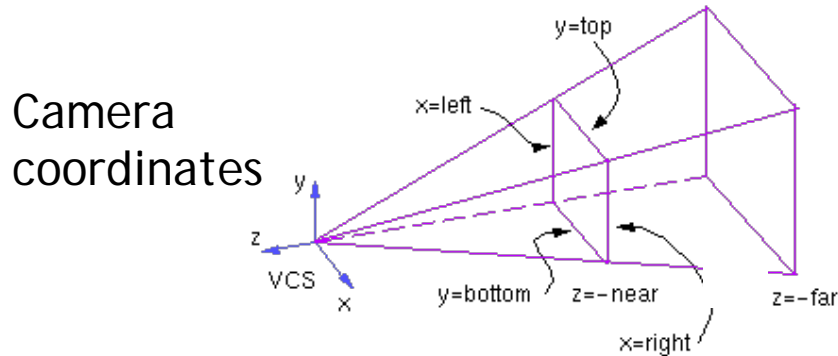


Image space
(pixel coordinates)

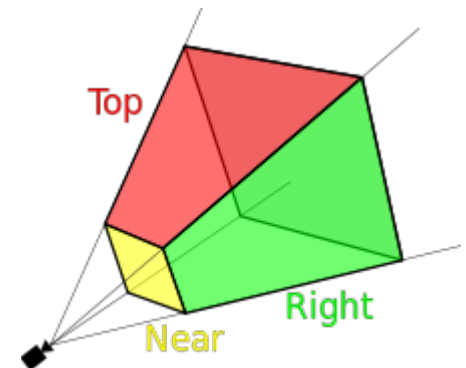


Perspective View Volume

General view volume

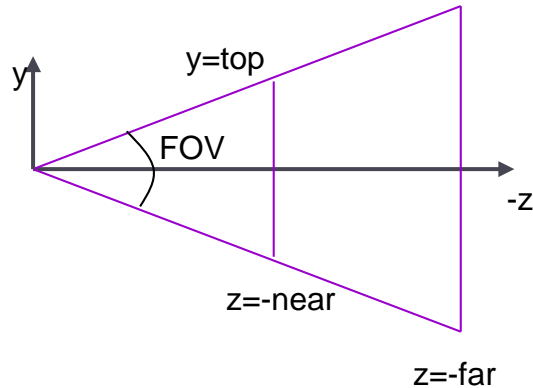


- ▶ Defined by 6 parameters, in camera coordinates
 - ▶ Left, right, top, bottom boundaries
 - ▶ Near, far clipping planes
- ▶ Clipping planes to avoid numerical problems
 - ▶ Divide by zero
 - ▶ Low precision for distant objects
- ▶ Usually symmetric, i.e., $\text{left} = -\text{right}$, $\text{top}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



Perspective View Volume

Symmetrical view volume



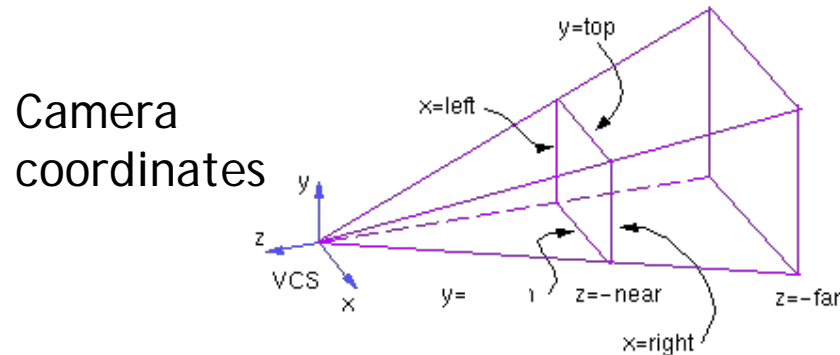
- ▶ Only 4 parameters
 - ▶ Vertical field of view (FOV)
 - ▶ Image aspect ratio (width/height)
 - ▶ Near, far clipping planes

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

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

Perspective Projection Matrix

- ▶ General view frustum with 6 parameters

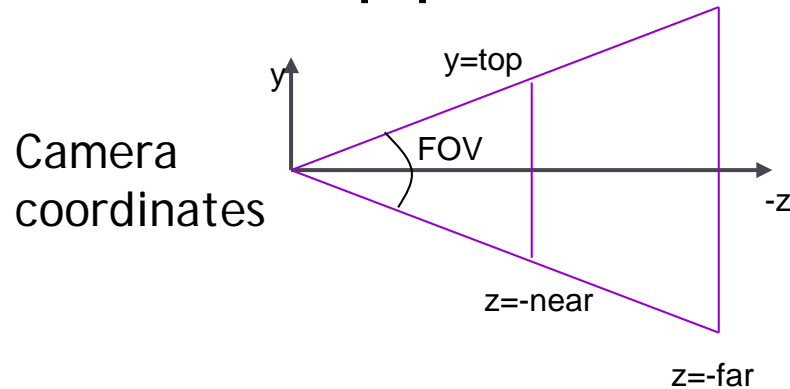


$$\mathbf{P}_{persp}(left, right, top, bottom, near, far) =$$

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

Perspective Projection Matrix

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



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

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

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:

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

Lecture Overview

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

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:

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

|
| 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:

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

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:

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

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:

$$p' = \begin{matrix} \mathbf{D} & \mathbf{P} & \mathbf{C}^{-1} & \mathbf{M} & \mathbf{p} \\ \text{Canonical view volume} & \text{Camera space} & \text{World space} & \text{Object space} & \end{matrix}$$

Image 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:

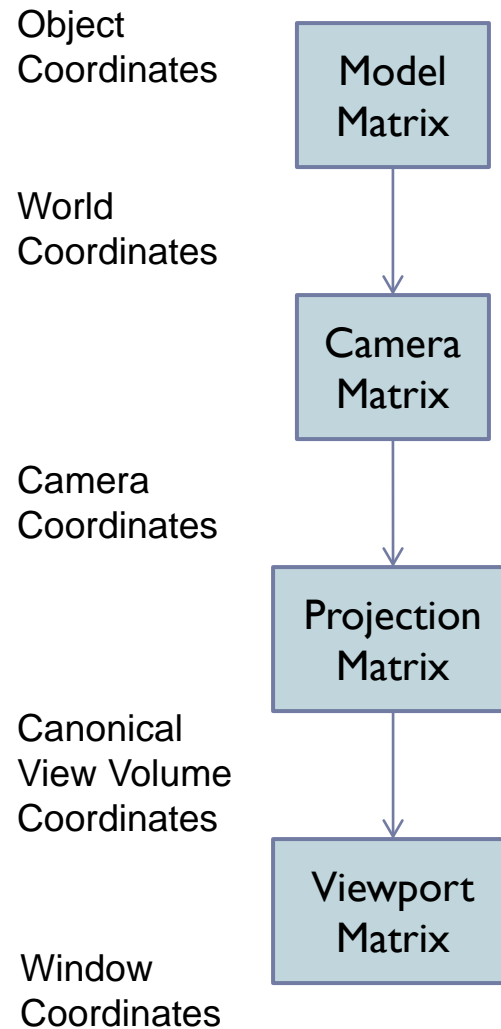
$$\mathbf{p}' = \mathbf{DPC}^{-1}\mathbf{Mp}$$

$$\mathbf{p}' = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

Pixel coordinates: x'/w'
 y'/w'

- ▶ **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:

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

Projection matrix

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

Complete Vertex Transformation in OpenGL

- ▶ **ModelView matrix: $C^{-1}M$**
 - ▶ Defined by the programmer.
 - ▶ Think of the ModelView matrix as where you stand with the camera and the direction you point it.
- ▶ **Projection matrix: P**
 - ▶ Think of the projection matrix as describing the attributes of your camera, such as field of view, focal length, etc.
- ▶ **Viewport, D**
 - ▶ Specify via `glViewport(x, y, width, height)`

Vertex Shader Code

```
in vec4 vertexPosition;
```

```
// ...
```

```
uniform mat4 ModelView, Projection;
```

```
void main() {
```

```
    gl_Position = Projection * ModelView  
* vertexPosition;
```

```
    // ...
```

```
}
```