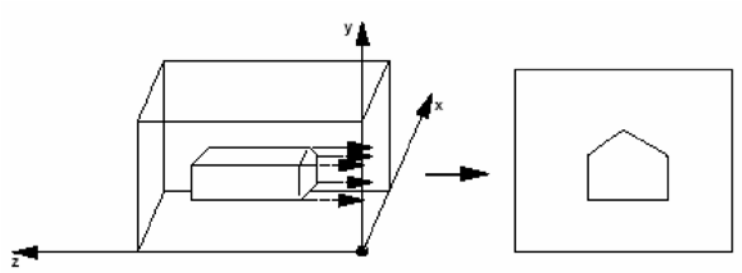
CSE 167:
Introduction to Computer Graphics
Lecture #6: Projection

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2020

Projection

Projection

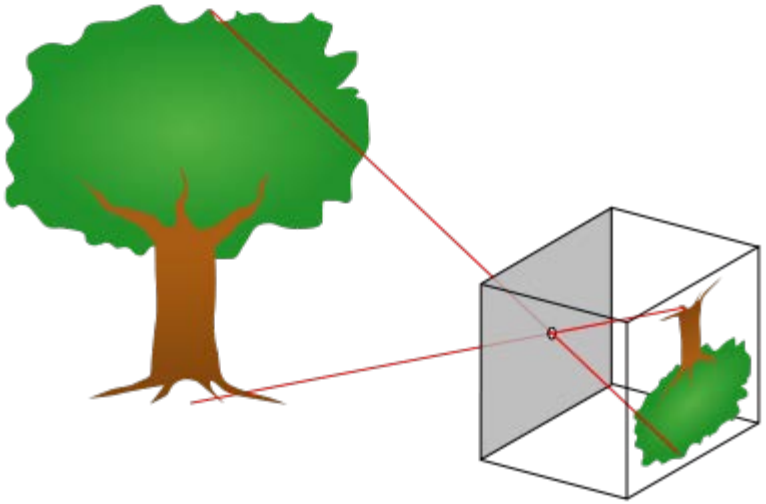
- Goal:
  Given 3D points (vertices) in camera coordinates, determine corresponding image coordinates
- Transforming 3D points into 2D is called Projection
- Typically one of two types of projection is used:
  - Orthographic Projection (=Parallel Projection)



  - Perspective Projection: most commonly used

Perspective Projection

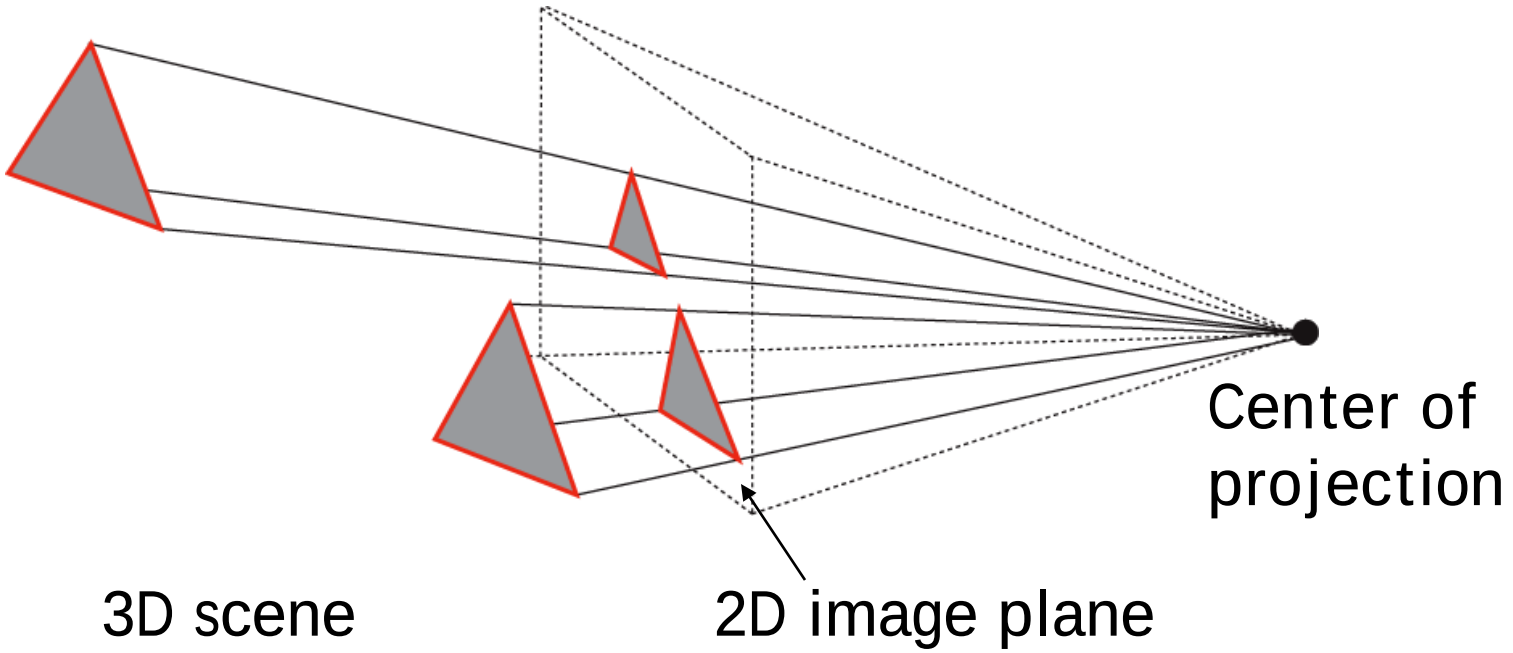
- Most common for computer graphics
- Simplified model of human eye, or camera lens (*pinhole camera*)



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

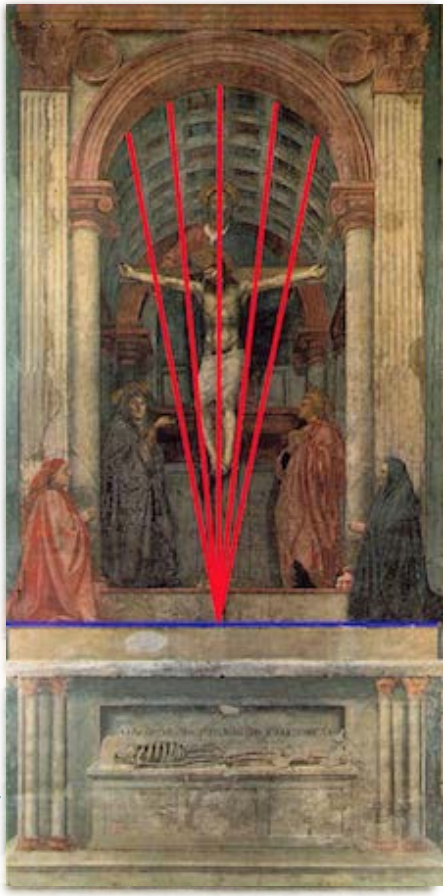
Perspective Projection

- Project along rays that converge in center of projection



Perspective Projection

Parallel lines are
no longer parallel,
converge in one point



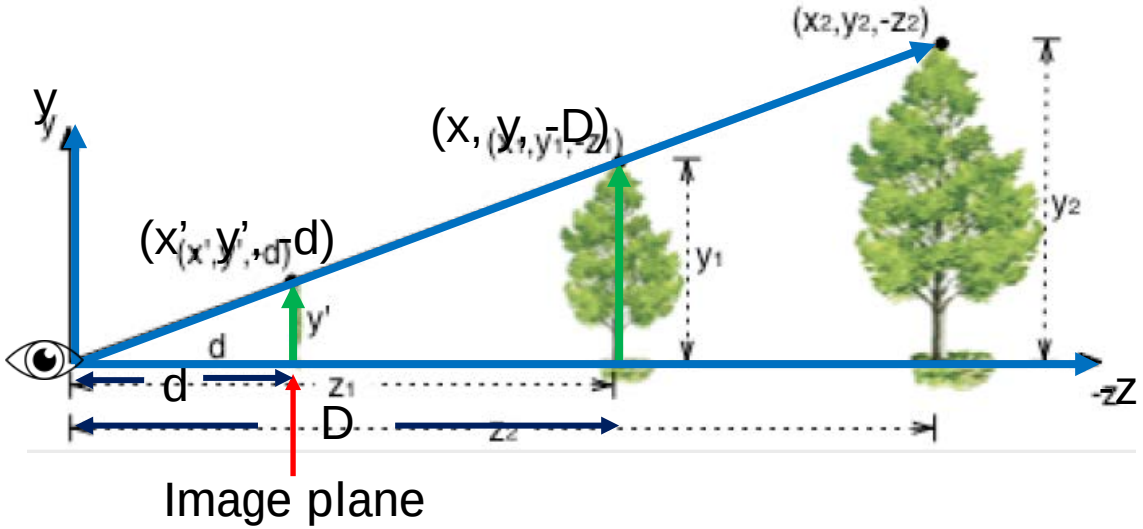
Earliest example:
La Trinitá (1427) by Masaccio

Perspective Projection

From law of ratios in similar triangles follows:

$$\frac{y'}{d} = \frac{y}{D} \rightarrow y' = \frac{yd}{D}$$

Similarly: $x' = \frac{xd}{D}$



By definition: $z' = -d$

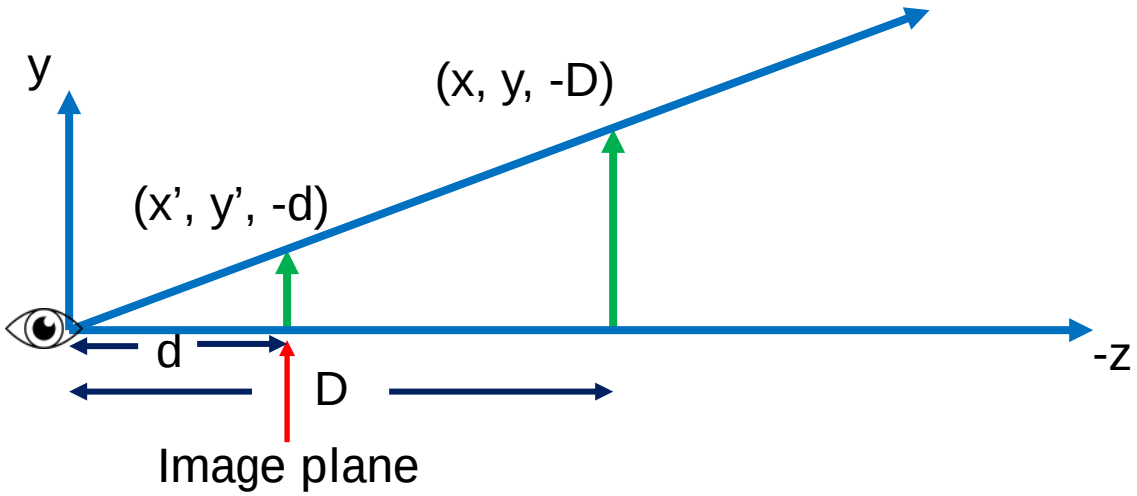
- We can express this using homogeneous coordinates and 4x4 matrices as follows

Perspective Projection

$$x' = \frac{xd}{D}$$

$$y' = \frac{yd}{D}$$

$$z' = -d$$



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

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

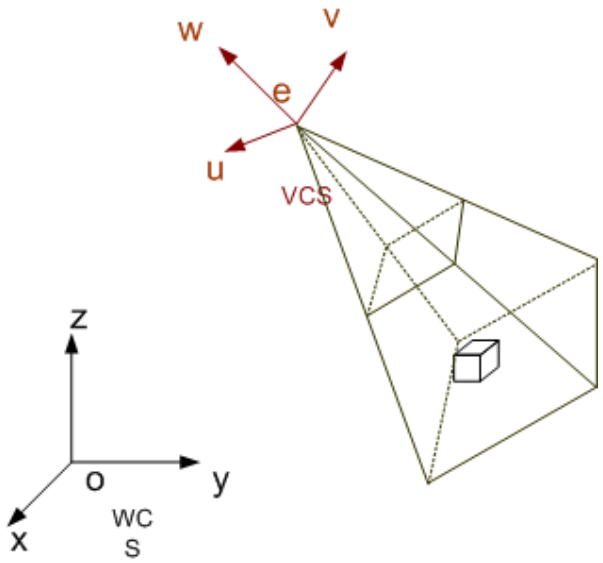
Topics

- View Volumes
- Vertex Transformation
- Rendering Pipeline
- Culling

View Volume

- View volume = 3D volume seen by camera

Camera coordinates



World coordinates

Projection Matrix

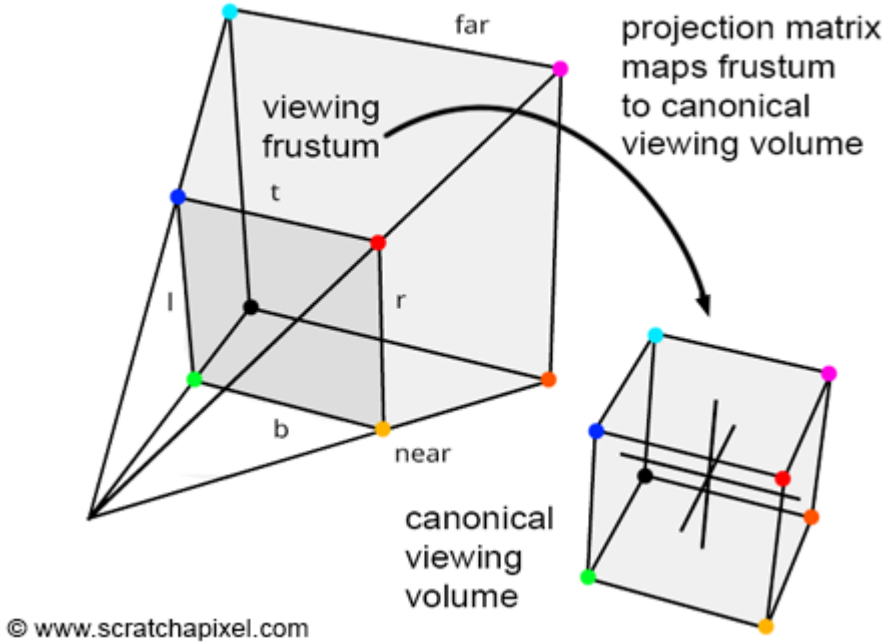
Camera coordinates

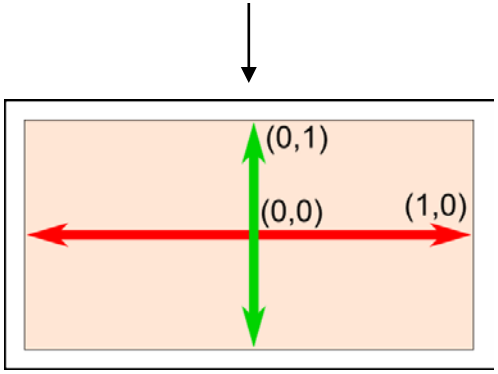
Projection matrix

Canonical view volume

Viewport transformation

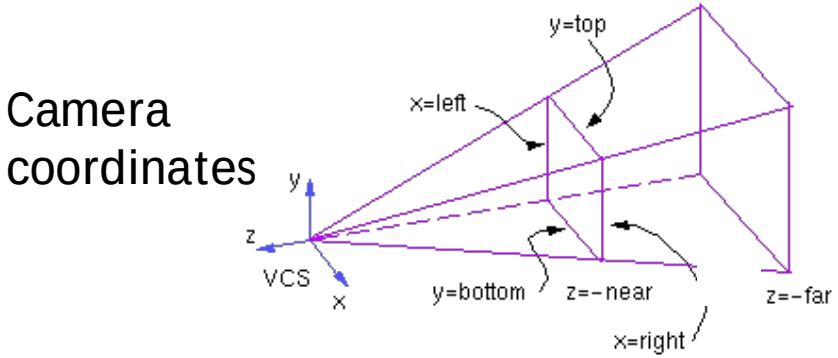
Image space
(pixel coordinates)

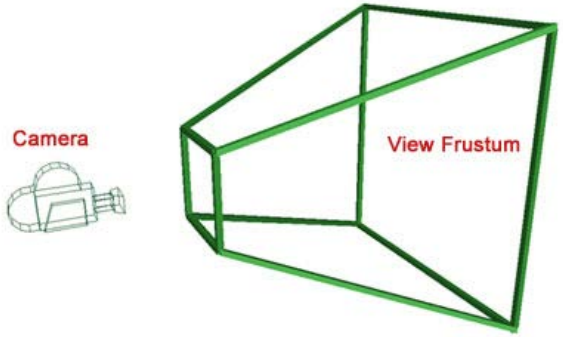




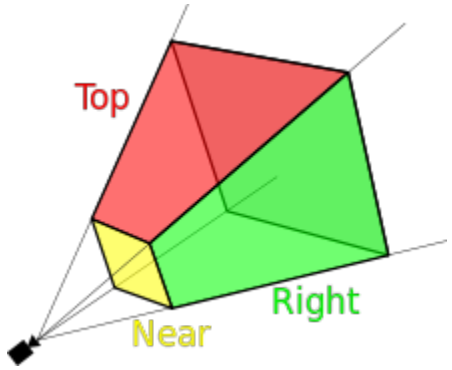
Perspective View Volume

General view volume



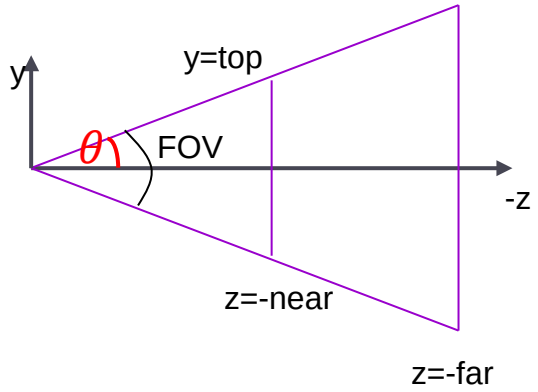


- Defined by 6 parameters, in camera coordinates
  - Left, right, top, bottom boundaries
  - Near, far clipping planes
- Clipping planes to avoid numerical problems
  - Divide by zero (multiplying all coordinates by - d/z)
  - Low precision for distant objects
- Usually symmetric, i.e., left=-right, 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
- Demo link

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

$$\tan(\underbrace{FOV / 2}_{\theta}) = \frac{top}{near}$$

Perspective View Volume

Rule of thumb to calculate projection matrix:

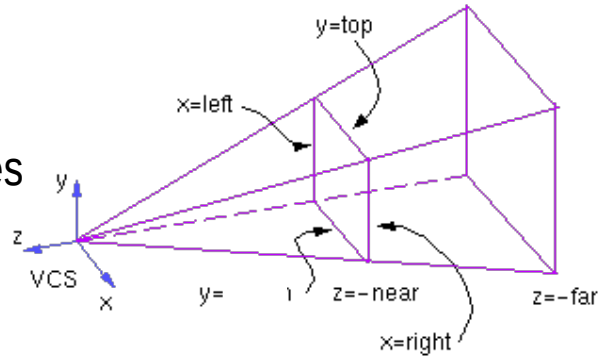
1. Convert the view-frustum to the simple symmetric projection frustum
2. Transform the simple frustum to the canonical view frustum

Ref: http://info.ee.surrey.ac.uk/Teaching/Courses/eem.cgi/lectures_pdf/lecture3.pdf

Perspective Projection Matrix

- General view frustum with 6 parameters

Camera coordinates

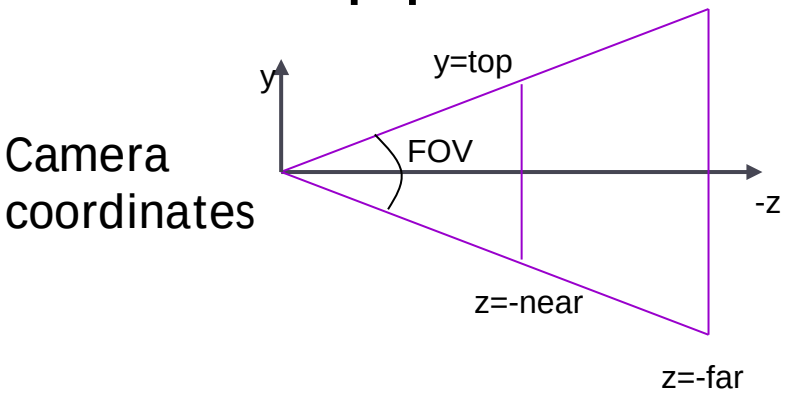


$$\mathbf{P}_{persp}(left, right, top, bottom, near, far) =$$

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

Perspective Projection Matrix

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



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

Ref: https://www.youtube.com/watch?v=ohksz3A00fk&t=79s

Projection Matrix

- How to determine if a matrix is projection matrix?

Canonical View Volume

- Goal: create projection matrix so that
  - User defined view volume is transformed into canonical view volume: cube [-1,1]x[-1,1]x[-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

Canonical View Volume

- Summary so far in a demo

Viewport Transformation

- After applying projection matrix, scene points are in *normalized viewing coordinates*
  - Per definition within range [-1..1] x [-1..1] x [-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: [x0…x1] x [y0…y1]
- 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:

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

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:

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

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:

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

Object space

World space

Camera space

Canonical view volume

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:

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

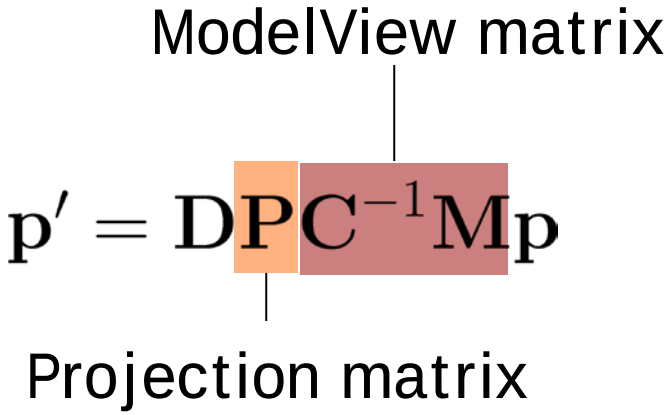
$$\mathbf{p}' = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

Pixel coordinates: $\begin{matrix} x'/w' \\ y'/w' \end{matrix}$

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

Complete Vertex Transformation in OpenGL

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



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

Complete Vertex Transformation in OpenGL

- ModelView matrix: $\mathbf{C^{-1}M}$
  - 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

```
layout (location = 0) in vec3 position;
// ...

uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;

void main() {
  gl_Position = projection * view *
model * vec4(position, 1.0);
    // ...
}
```

The Complete Vertex Transformation