CSE-170 Computer Graphics

Lecture 5

Transformations (II)

Dr. Renato Farias
rfarias2@ucmerced.edu

Reminder

- Consult our book!
  - Chapter 2: Miscellaneous Math
    - Quadratic equations, trigonometry, vectors, etc.
  - Chapter 6: Linear Algebra
    - Matrices
  - Chapter 7: Transformation Matrices
    - Linear and Affine transformations
  - Chapter 8: Viewing
    - Viewing, projective, perspective transformations

Types of Transformations

Transformations

- Linear
  - rotations, scalings, shears
- Affine
  - Linear + Translation
  - preserves collinearity and ratios of distances (midpoints remain midpoints, etc.)
  - preserves barycentric combinations or affine combinations (we will see them soon)
  - may not preserve angles or lengths
- Projections
  - parallel (orthogonal or oblique) projections
  - perspective projections

Transformations

- Affine and projective transformations can be computed by multiplying 4x4 matrices with vectors in homogeneous coordinates

$$T(\underline{\mathbf{v}}) = \begin{pmatrix} e_{11} & e_{12} & e_{13} & e_{14} \\ e_{21} & e_{22} & e_{23} & e_{24} \\ e_{31} & e_{32} & e_{33} & e_{34} \\ e_{41} & e_{42} & e_{43} & e_{44} \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ v_w \end{pmatrix}$$

Affine Transformations

- Translation
  T(*a*,*b*,*c*)

$$T(a,b,c)=\begin{pmatrix}1&0&0&a\\0&1&0&b\\0&0&1&c\\0&0&0&1\end{pmatrix}$$

- Scaling
  S(*r*,*s*,*t*)

$$S(r,s,t)=\begin{pmatrix}r&0&0&0\\0&s&0&0\\0&0&t&0\\0&0&0&1\end{pmatrix}$$

- Rotation $R_x(\theta)$, $R_y(\theta)$, $R_z(\theta)$

$$R_x(\theta)=\begin{pmatrix}1&0&0&0\\0&\cos(\theta)&-\sin(\theta)&0\\0&\sin(\theta)&\cos(\theta)&0\\0&0&0&1\end{pmatrix}\quad R_y(\theta)=\begin{pmatrix}\cos(\theta)&0&\sin(\theta)&0\\0&1&0&0\\-\sin(\theta)&0&\cos(\theta)&0\\0&0&0&1\end{pmatrix}\quad R_z(\theta)=\begin{pmatrix}\cos(\theta)&-\sin(\theta)&0&0\\\sin(\theta)&\cos(\theta)&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$$

Change of Basis
is also a Transformation

Change of Orthonormal Basis

- Given:

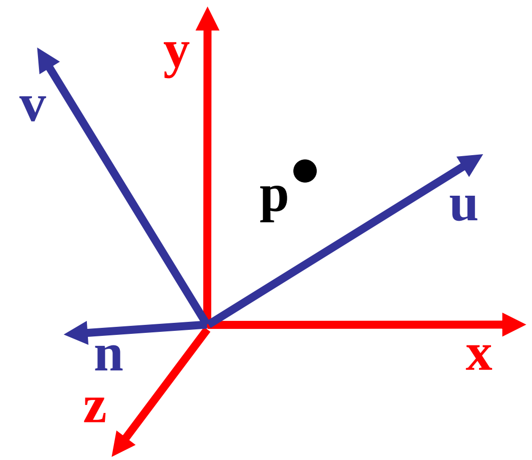
  coordinate frames

  xyz and **uvn**,

  and point $\mathbf{p}_{xyz} = (x,y,z)$

- Find:

  $\mathbf{p}_{uvn} = (u,v,n)$

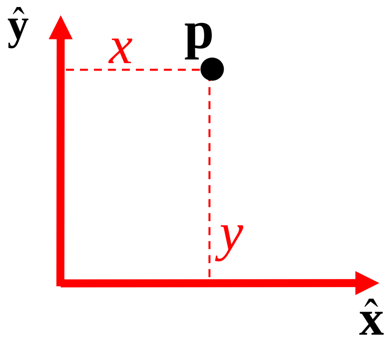


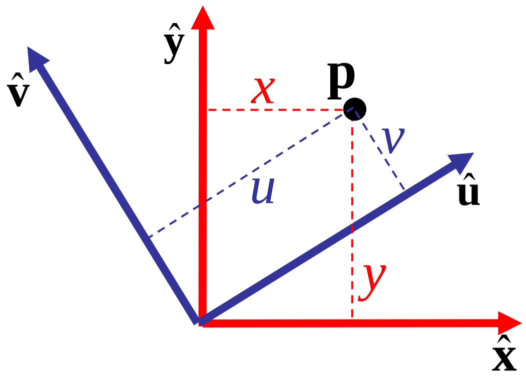
Change of Orthonormal Basis

- ...are affine transformations

  2D ex: transform between frames with same origin

  $\mathbf{p} = (x,y)$ and $\mathbf{p} = (u,v)$

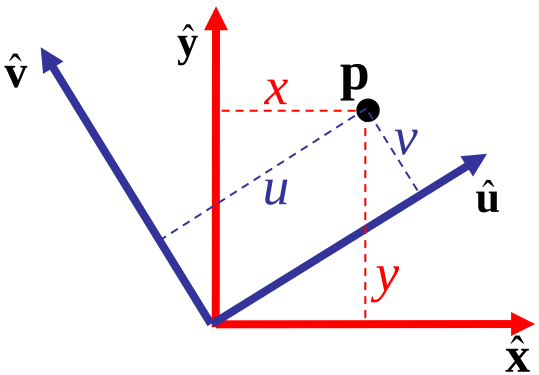


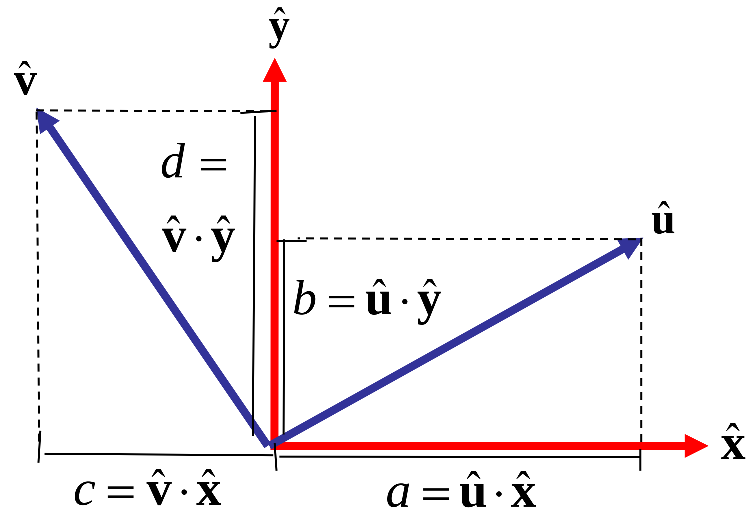


Change of Orthonormal Basis

$$\Rightarrow \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & c \\ b & d \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \Rightarrow \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \longleftarrow$$

Result: a matrix multiplication relates (x,y) to (u,v)
(matrix is a rotation transformation, its inverse is the transpose)





Change of Orthonormal Basis

- Just write a transformation as follows:
  - The lines in the matrix are the coordinates of the new frame written with respect to the old frame:

$$\begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ n_x & n_y & n_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} u \\ v \\ n \end{pmatrix} \Rightarrow \mathbf{M} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} u \\ v \\ n \end{pmatrix}$$

- To transform back, use the inverse:

$$\mathbf{M} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} u \\ v \\ n \end{pmatrix} \Rightarrow \mathbf{M}^{-1} \begin{pmatrix} u \\ v \\ n \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Change of Orthonormal Basis

- Change of basis often needs an affine transformation
  - If the frames do not have the same origin point, a translation is combined

Change of Orthonormal Basis

- Inverse:
  - If the transformation is only a rotation, then:

$$\mathbf{M}^{-1} = \mathbf{M}^{T}$$

  - If it encodes a rotation and a translation, you may use the "translation trick":
    - take the transpose only of the 3x3 submatrix, and then negate the translation components of the 4x4 matrix

Camera Transformations

- Camera transformations are equivalent to a change of basis
  - Coordinates of the objects in the scene are transformed with respect to a camera frame of reference, for ex:

$$\begin{pmatrix} U_x & U_y & U_z & 0 \\ V_x & V_y & V_z & 0 \\ N_x & N_y & N_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{pmatrix} = \begin{pmatrix} x_{camera} \\ y_{camera} \\ z_{camera} \\ 1 \end{pmatrix}$$

  - The popular lookAt() function builds a matrix very similar to the one above
    - but it also incorporates translation, to account for an arbitrary point of view

Viewing Transformations

Projections

- Let's review the usual transformations in our rendering pipeline:

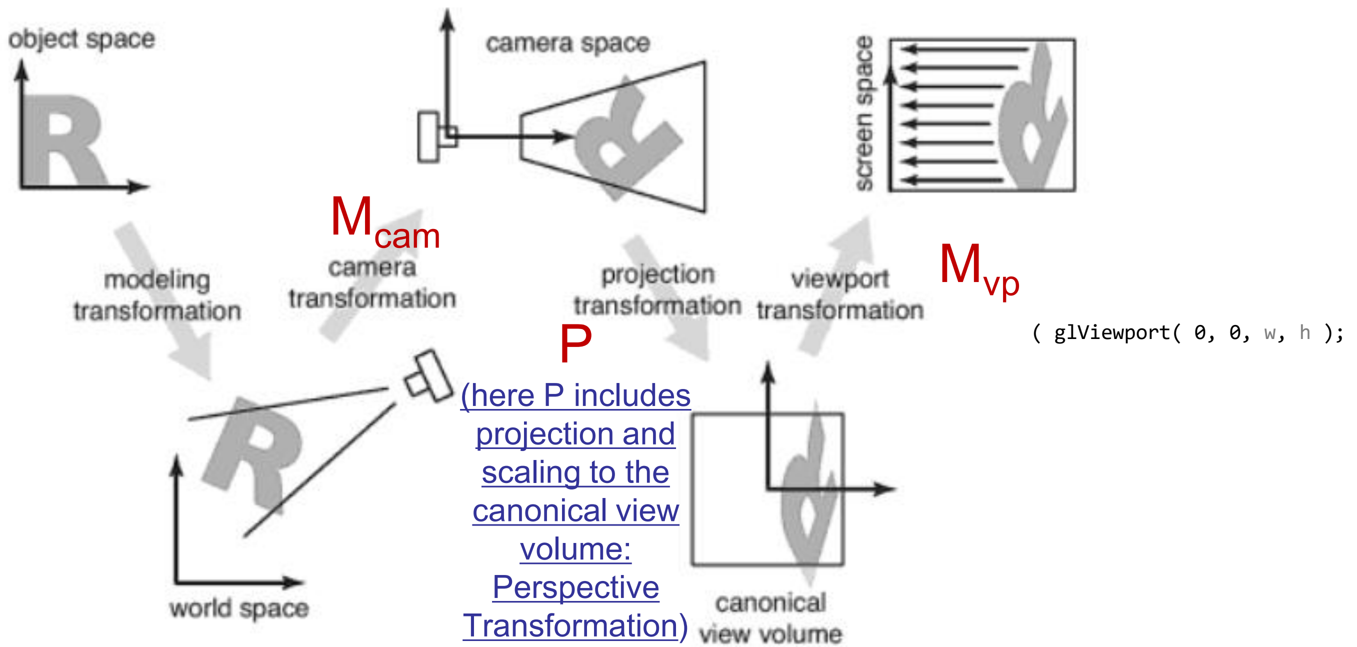


Figure 7.2. The sequence of spaces and transformations that gets objects from their original coordinates into screen space.

→ Final matrix for perspective viewing: $M=M_{vp}\ P\ M_{cam}$

→ Projections "project 3D objects onto a 2D plane"

Camera Transformation

- Typical parameters:
  - eye position, center, up vector
- Then:

  Gaze direction $\mathbf{z}$ = ||eye-center||

  $\mathbf{x}$ = up × $\mathbf{z}$

  $\mathbf{y} = \mathbf{z} \times \mathbf{x}$

$$\mathbf{M}_{cam} = \begin{pmatrix} \mathbf{x}.x & \mathbf{x}.y & \mathbf{x}.z & 0 \\ \mathbf{y}.x & \mathbf{y}.y & \mathbf{y}.z & 0 \\ \mathbf{z}.x & \mathbf{z}.y & \mathbf{z}.z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -\mathbf{e}.x \\ 0 & 1 & 0 & -\mathbf{e}.y \\ 0 & 0 & 1 & -\mathbf{e}.z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

→ see Chapter 8, and gluLookAt() for reference:

https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/gluLookAt.xml

Projections

- Are you looking at 2D or 3D objects…?

Projections

- Here the transparent window is equivalent to the "near plane" of the viewing frustum

- Photographer's position is the camera eye position

- The image on the window plane is the result of projections

Projections

- Are these two angles 90 degrees angles…?

Projections

- In scene coordinates they are 90 degrees
  - in window coordinates they are not

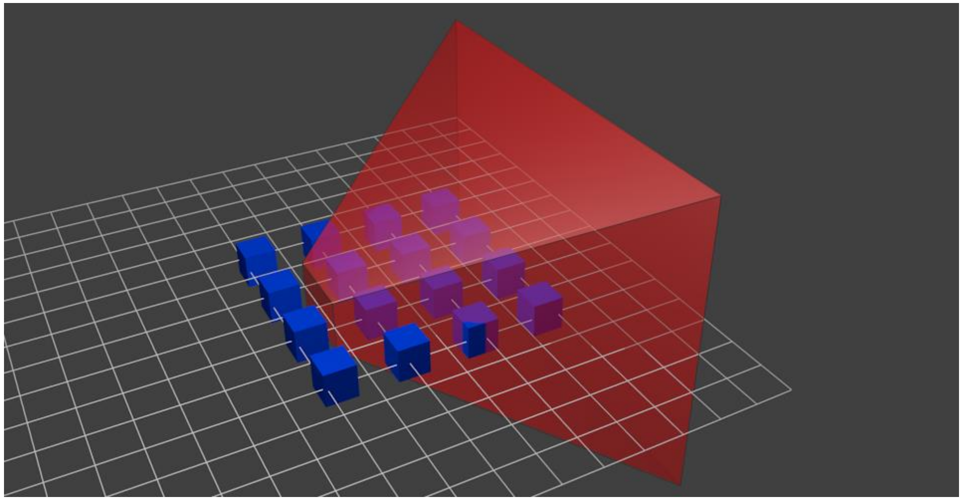
- The complete viewing transformation has:
  - “camera” and “perspective” transformations

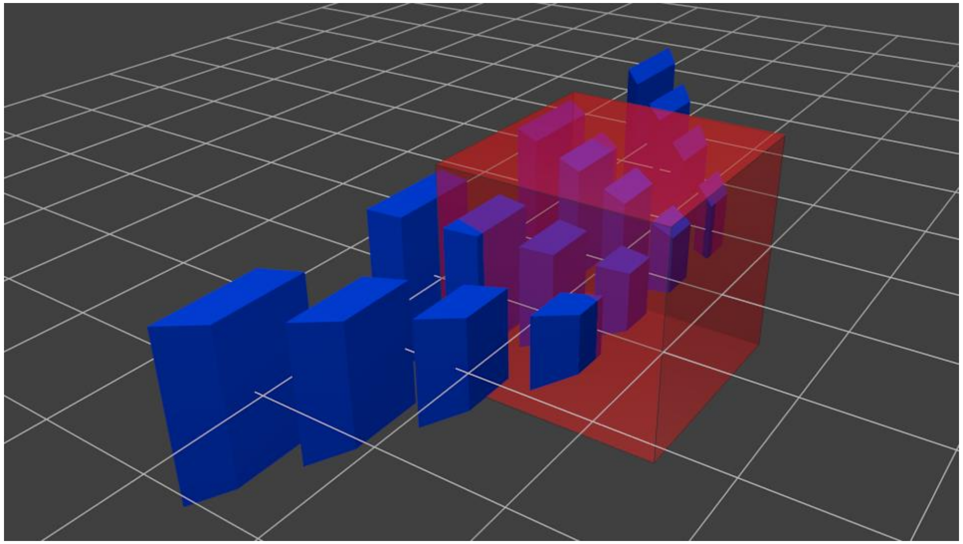
Perspective Camera Transformation

- Camera perspective projection in OpenGL uses:
  - position along z of the “near plane”
  - position along z of the “far plane”
  - eye position at (0,0,0)

- Includes scaling scene to normalized coordinates
- We have to compute it, there is no included camera model

Perspective Camera Transformation





(pictures from opengl-tutorial.org)

Perspective Camera Transformation

- Projection along z inside viewing frustrum:

  fov: field of view angle

  aspect: screen aspect ratio

  zNear, zFar: near and far planes

$$f = cotangent\left(\frac{fovy}{2}\right)$$

The generated matrix is

$$\begin{pmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{zFar + zNear}{zNear - zFar} & \frac{2 \times zFar \times zNear}{zNear - zFar} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

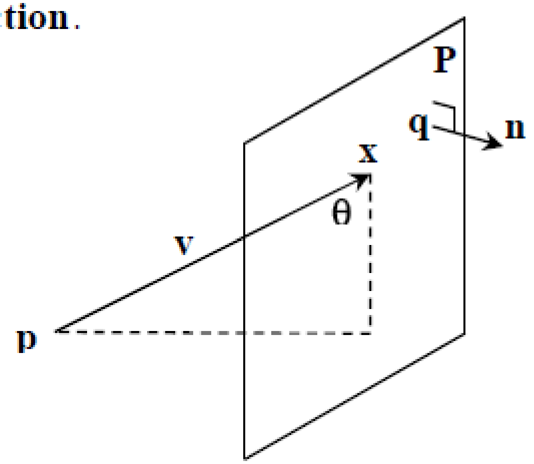
(The above definition is from the GLU library of the old 2.1 OpenGL version, use it only for reference: https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/gluPerspective.xml)

Other Perspective Transformations

- To be seen next class:

Parallel projection of **p** by direction **v** into plane **P (q,n)**, **q** a point, **n** the normal.
If θ=90 => **orthographic projection, otherwise oblique projection**.

$$\mathbf{x} = \begin{pmatrix} \begin{pmatrix} \mathbf{v}\cdot\mathbf{n} & 0 & 0 \\ 0 & \mathbf{v}\cdot\mathbf{n} & 0 \\ 0 & 0 & \mathbf{v}\cdot\mathbf{n} \end{pmatrix} - \left(\mathbf{v}\mathbf{n}^{T}\right) & (\mathbf{q}\cdot\mathbf{n})\mathbf{v} \\ 0 \quad 0 \quad 0 & \mathbf{v}\cdot\mathbf{n} \end{pmatrix} \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix}$$



For a **perspective projection**, direction **v** now becomes **-p**, which is the line from **p** to the coordinate system origin.

And the resulting transformation is:

$$\mathbf{x} = \begin{pmatrix} \begin{pmatrix} \mathbf{q}\cdot\mathbf{n} & 0 & 0 \\ 0 & \mathbf{q}\cdot\mathbf{n} & 0 \\ 0 & 0 & \mathbf{q}\cdot\mathbf{n} \end{pmatrix} & 0 \\ 0 \quad 0 \quad 0 & \mathbf{p}\cdot\mathbf{n} \end{pmatrix} \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix}$$

Transformations
Summary and Properties

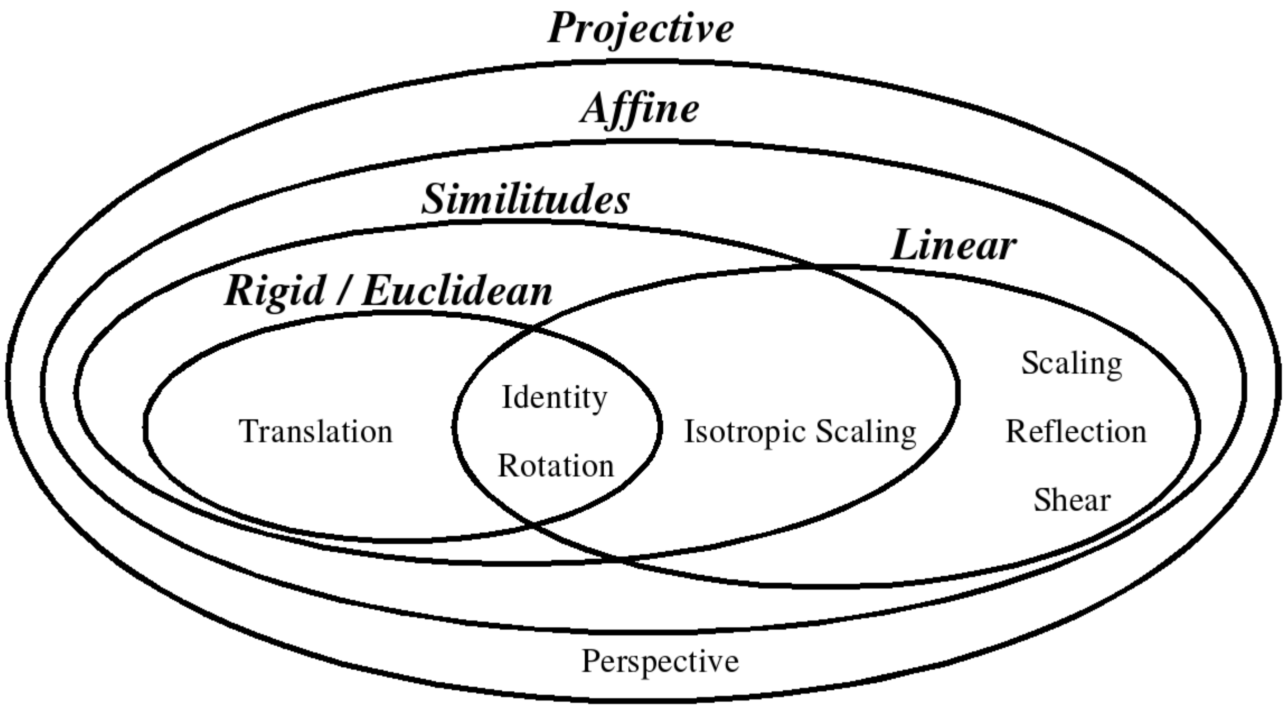
Transformations - Properties

- Preserves…

	Rigid	Linear	Affine	Projective
lengths	✓			
angles	✓			
ratios of distances	✓	✓	✓	
parallel lines	✓	✓	✓	
straight lines	✓	✓	✓	✓

Transformations - Taxonomy

- The different kinds of transformations we saw:



  – Similitudes = Rigid + Uniform Scaling