CSC 4356
Interactive Computer Graphics
Lecture 9: 3D Viewing (Part 2)

Jinwei Ye

http://www.csc.lsu.edu/~jye/CSC4356/

Tue & Thu: 10:30 - 11:50am
218 Tureaud Hall

Transformation Recap

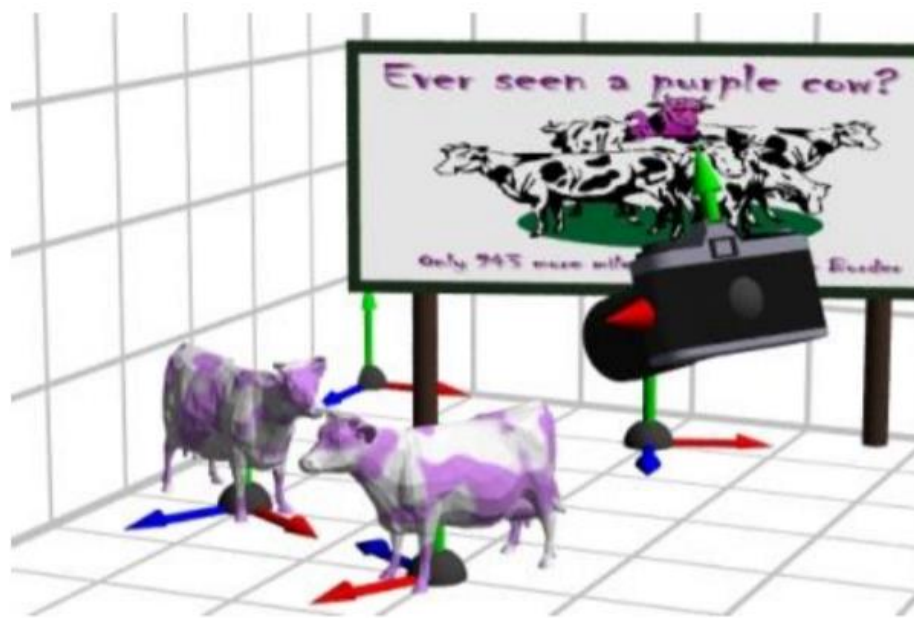
- Model (geometric) transformation
  - Arrange objects in the world coordinate
- Projection transformation
  - Map 3D objects to 2D image in the camera coordinate

Transformation Recap

- Model (geometric) transformation
  - Arrange objects in the world coordinate
- Projection transformation
  - Map 3D objects to 2D image in the camera coordinate

Viewing Transformation

- Map points from world coordinate to camera/eye coordinate
- Use the MODELVIEW matrix stack in OpenGL
  - Same stack as model transformation

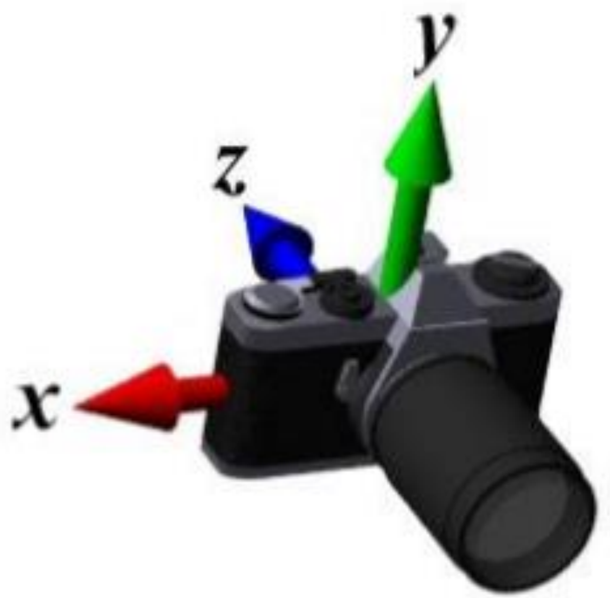


"Framing" the Picture

- Reorient the entire scene such that the camera is located at the origin
  - OpenGL assumes camera at origin
- Greatly simply the projection steps

Camera/Eye Space

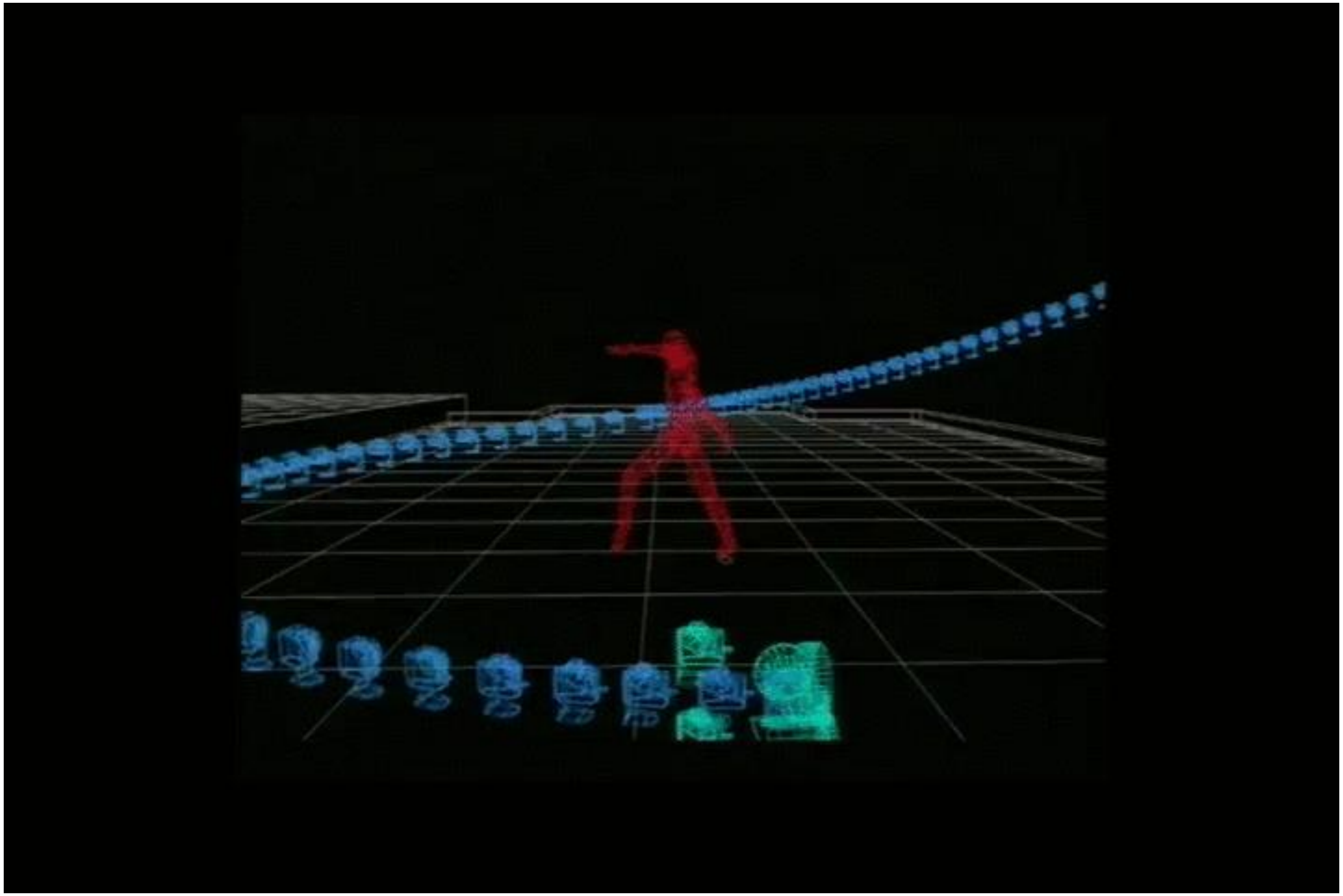
- Origin is located at the center of projection (COP) for perspective projection
- Image plane is parallel to the x-y plane
- Camera is viewing towards the –z direction



Notes on Camera Space

- Although the goal is to transform the world space to camera space, it is more natural to think of camera as an object positioned in the world space
- In this way, we make it easy to change viewpoint
  - Simply change the camera coordinate in the world space
- Useful for generating cool visual effects

Visual Effect: Bullet Time

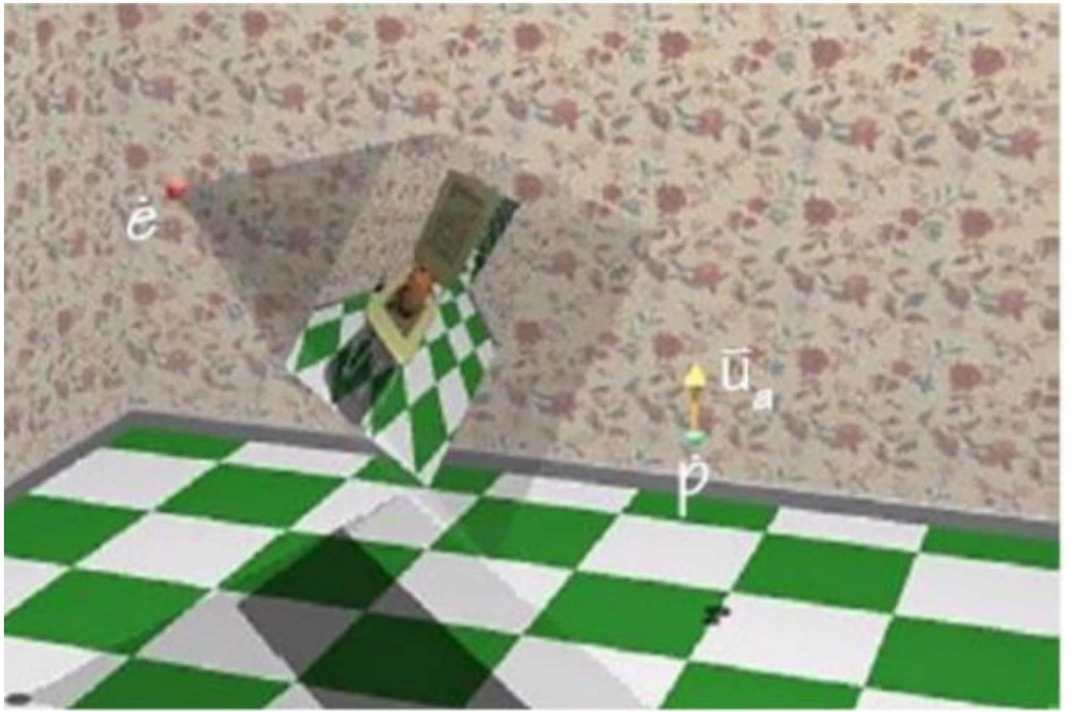


Goal of Viewing Transformation

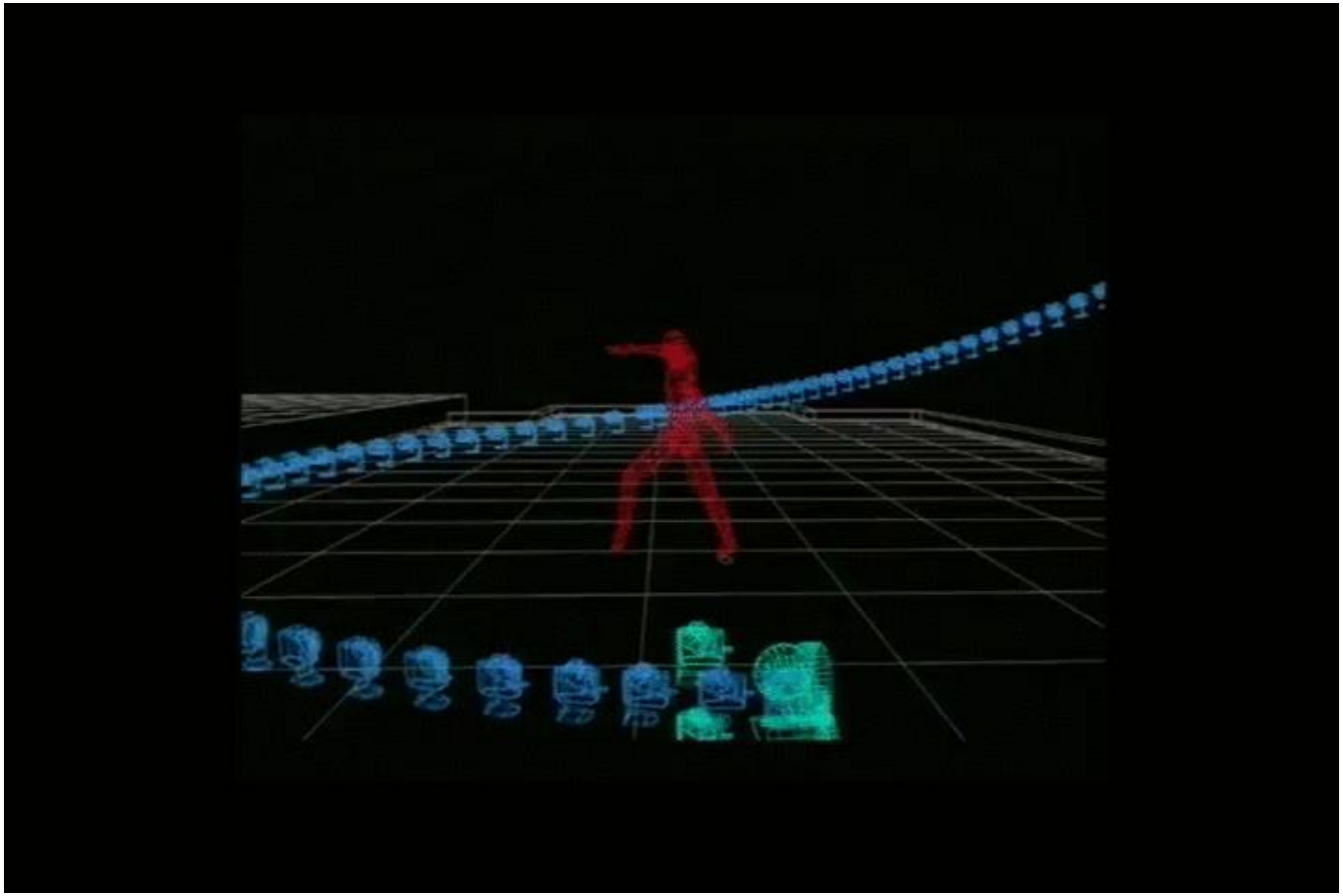
- Define the camera/eye space
  - Specify the position and orientation of the viewing camera
- Establish mapping between the two coordinate system
  - World space to camera space
  - Rotation & Translation

Define Camera Space

- Eye point: camera position (COP)
- Look-at point: center of the image
- Up vector: upwards orientation in the image



Visual Effect: Bullet Time



- Specify camera path by simply changing the eye point

Viewing Transformation: Derivation

- Let's first derive the rotation matrix $\mathbf{R}_v$ of the viewing transformation
- Look-at direction:

$$\begin{bmatrix} l_x \\ l_y \\ l_z \end{bmatrix} = \begin{bmatrix} lookat_x \\ lookat_y \\ lookat_z \end{bmatrix} - \begin{bmatrix} eye_x \\ eye_y \\ eye_z \end{bmatrix}$$

$$\hat{l} = \frac{\vec{l}}{\sqrt{l_x^2 + l_y^2 + l_z^2}}$$

First Constraint

- Camera is viewing towards –z direction
- So we expect our desired rotation matrix to map the look-at direction to the vector $[0, 0, -1]^T$

$$\begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} = \mathbf{R}_v \begin{bmatrix} \hat{l}_x \\ \hat{l}_y \\ \hat{l}_z \end{bmatrix}$$

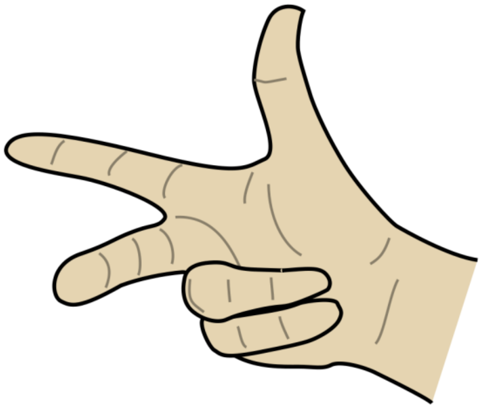
Second Constraint

- There is another special vector that we can compute
- If we find the cross product between the look-at vector with our up vector, we will get a vector that points to the right

$$\vec{r} = \vec{l} \times \overrightarrow{up}$$

- We expect the right vector, when normalized, will transform to the vector $[1, 0, 0]^T$

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \mathbf{R}_v \frac{\vec{r}}{\sqrt{r_x^{\,2} + r_y^{\,2} + r_z^{\,2}}}$$



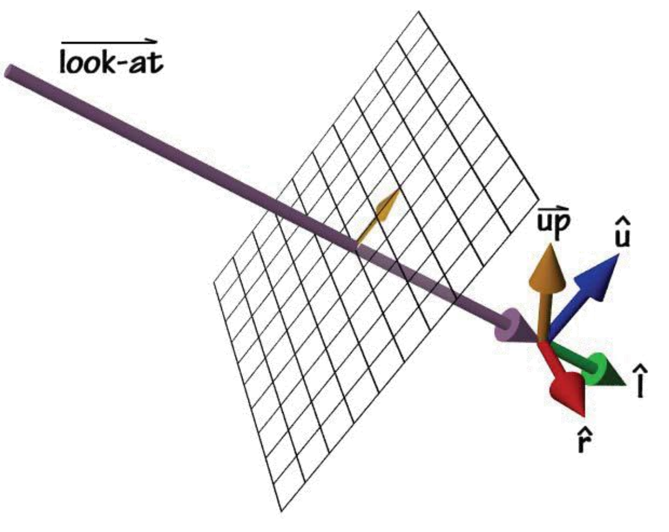
Third Constraint

- Finally, from these two vectors we can synthesize a third vector that is perpendicular to both the look-at and right vectors. It is also oriented in the up direction

$$\vec{u} = \vec{r} \times \vec{l}$$

- We expect this vector, when normalized, will transform to the vector $[0, 1, 0]^T$

$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \mathbf{R}_v \frac{\vec{u}}{\sqrt{{u_x}^2 + {u_y}^2 + {u_z}^2}}$$



Putting Them All Together

- Now lets consider all of these constraints together

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{R}_v \begin{bmatrix} \hat{r} & \hat{u} & -\hat{l} \end{bmatrix}$$

- In order to compute the matrix, $\mathbf{R}_v$, we need only compute the inverse of the matrix formed by concatenating our 3 special vectors.
- How to compute the inverse?

Inverse is Transpose

- Remember that each of our vectors are unit length (we normalized them). Also, each vector is perpendicular to the other two. These two conditions on a matrix makes it, **orthogonal**. Rotations are also orthogonal. Orthonormal matrices have the unique property that:

$$\text{if } \mathbf{M} \text{ is Orthonormal, } \mathbf{M}^{-1} = \mathbf{M}^{\mathrm{T}}$$

- Therefore, the rotation component of our viewing transformation is just the transpose of the matrix formed by our selected vectors as rows.

$$\mathbf{R}_v = \begin{bmatrix} \hat{r}^T \\ \hat{u}^T \\ -\hat{l}^T \end{bmatrix}$$

Translation

- The rotation that we just derived is specified about the origin in world space.
- Therefore, before we can apply this rotation, we need to translate all world-space coordinates so that the eye point is at the origin.
- Translation is simply to move the origin of the world coordinate to the eye position

$$\mathbf{T}_{-eye} = \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewing Transformation

- Composing these transformations (translation and rotation) gives our viewing transformation matrix V

$$\mathbf{V} = \mathbf{R}_v \mathbf{T}_{-eye} = \begin{bmatrix} \hat{r}_x & \hat{r}_y & \hat{r}_z & 0 \\ \hat{u}_x & \hat{u}_y & \hat{u}_z & 0 \\ -\hat{l}_x & -\hat{l}_y & -\hat{l}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \hat{r}^T & -\hat{r} \cdot \overrightarrow{eye} \\ \hat{u}^T & -\hat{u} \cdot \overrightarrow{eye} \\ -\hat{l}^T & \hat{l} \cdot \overrightarrow{eye} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix}$$

$$P' = \mathbf{V}P = \mathbf{R}_v \mathbf{T}_{-eye} P$$

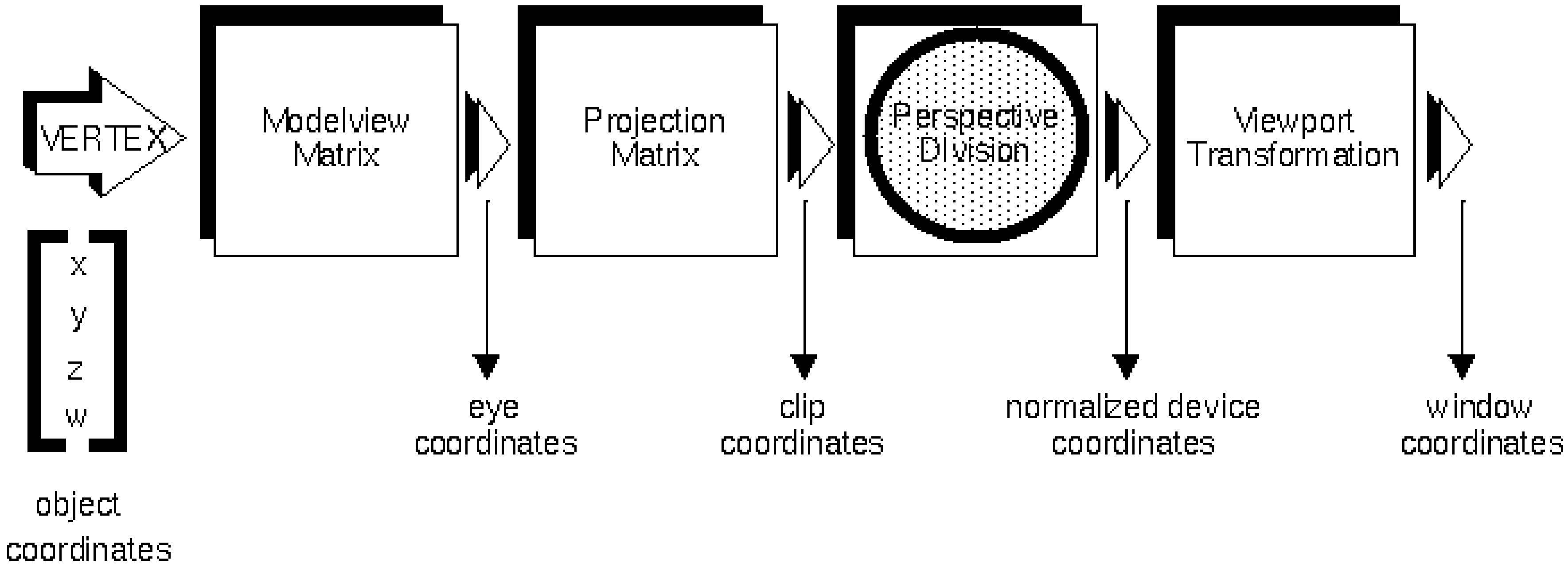
Viewing Transformation in OpenGL

- OpenGL provides a function for computing viewing transformations specified in terms of world space coordinates in its utility library (glu):

  ```
  gluLookAt(double eyex, double eyey,
  double eyez, double centerx, double
  centery, double centerz, double
  upx, double upy, double upz);
  ```

- It computes the same transformation that we derived and composes it with the current matrix (Modelview matrix)
- Viewing transformation is after model transformation

Transformation Pipeline



Model Transformation

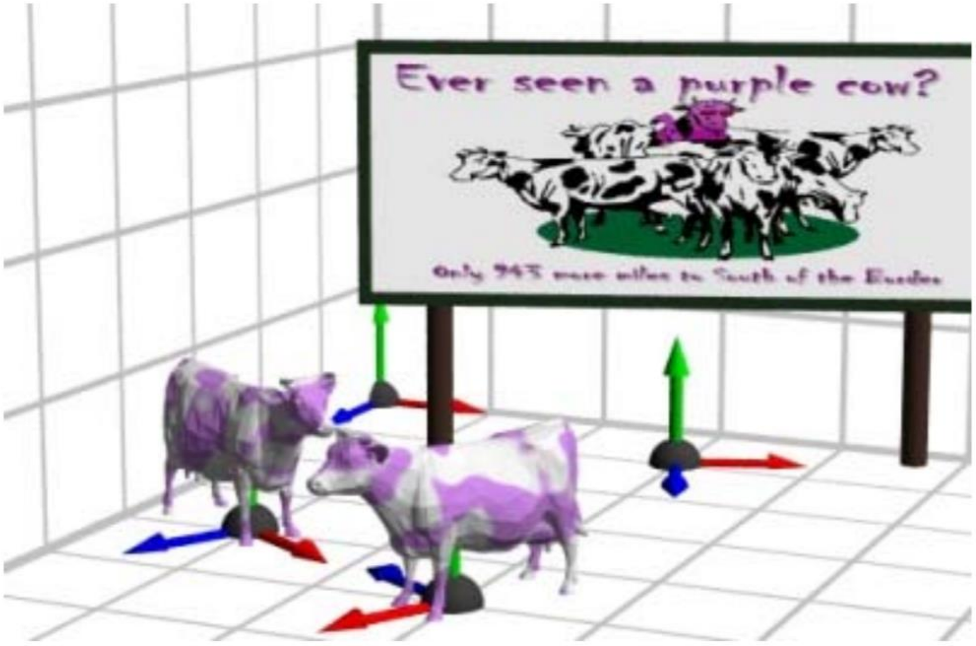
- We start with 3-D models defined in their own model space

$$\vec{m}_1^{\,t}, \vec{m}_2^{\,t}, \vec{m}_3^{\,t}, \ldots, \vec{m}_n^{\,t}$$

- Modeling transformations orient models within a common coordinate frame called world space

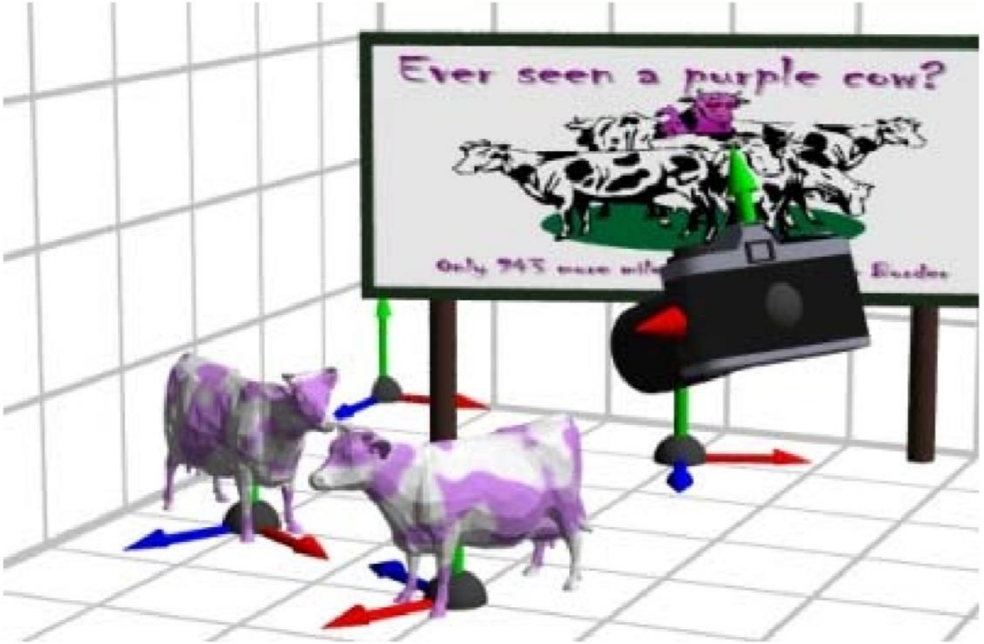
$$\vec{w}^{\,t}$$

- All objects, light sources, and the viewer/camera live in the world space



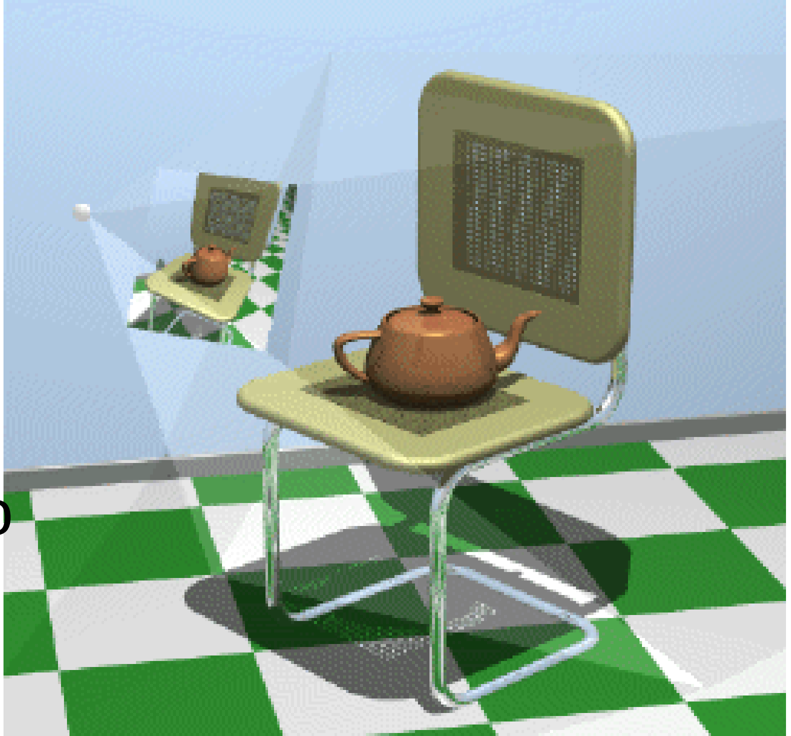
Viewing Transformation

- Another change of coordinate systems
- Transform points from world space into eye space
- Viewing position is transformed to the origin
- Viewing direction is oriented along –z direction
- Together with Model Transformation, they are called the Modelview Transformation



Projection Transformation

- Define a three dimensional viewing frustum
- Eliminate objects that are outside the viewing frustum
- Normalize the viewing frustum into a cube (NDC)
- Project the objects into 2D image
- Transformation from eye space to screen space



Analogous to Photography

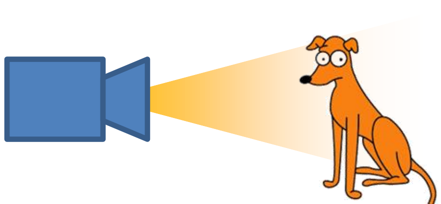
- Model transformation
  - Pose your model!

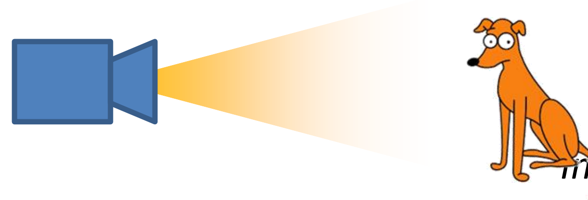


Image source: https://holmeslewismodels.wordpress.com/

Analogous to Photography

- Viewing transformation
  - Position your camera





Images from Pinterest

Analogous to Photography

- Projection transformation
  - Adjust your lens settings
  - gluperspective()

    vert fov: Field of view

    near plane: focal length

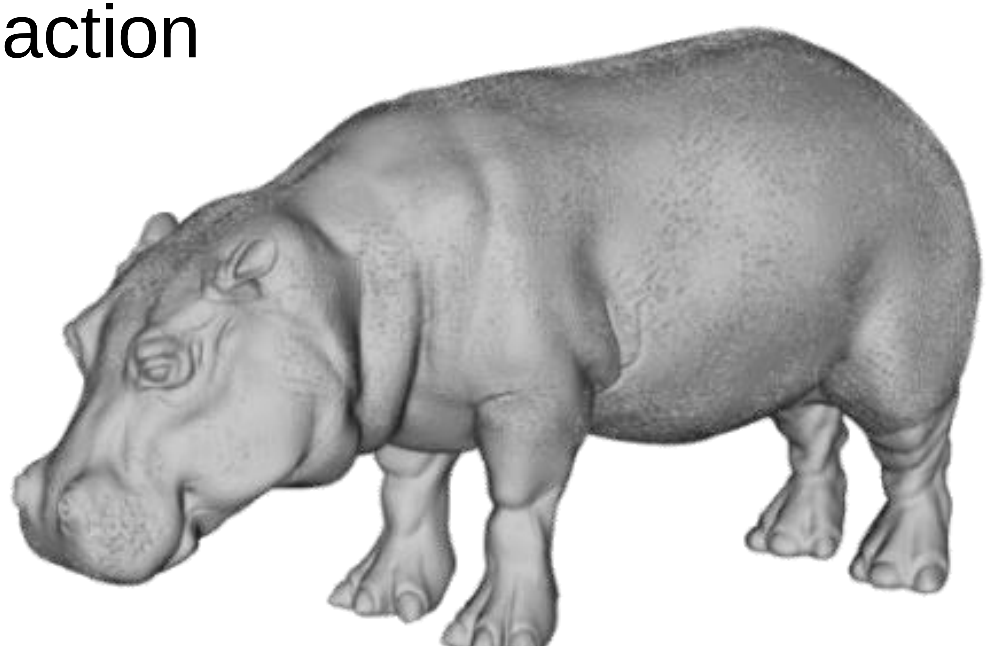
    far plane: infinity

Focal Length and FoV

Focal Length and FoV

Next Time…

- Build a 3D world
  - 3D model representation
  - data format
  - User interaction



Programming Assignment 1

- Due today at midnight! (11:59pm)
  - If you want to use free late days, please notify your TA. Otherwise, penalty will be taken per late day.
- **What to submit?**
  - .cpp file (your source code)
  - .exe file (executable)
  - Report that explains your implementation
- **Where to submit?**
  - classes.csc.lsu.edu
  - Log in using your account and password
  - Upload files to folder “prog1”
  - Use “p_copy” to submit and “verify” to confirm