CSE528 Computer Graphics: Theory, Algorithms, and Applications

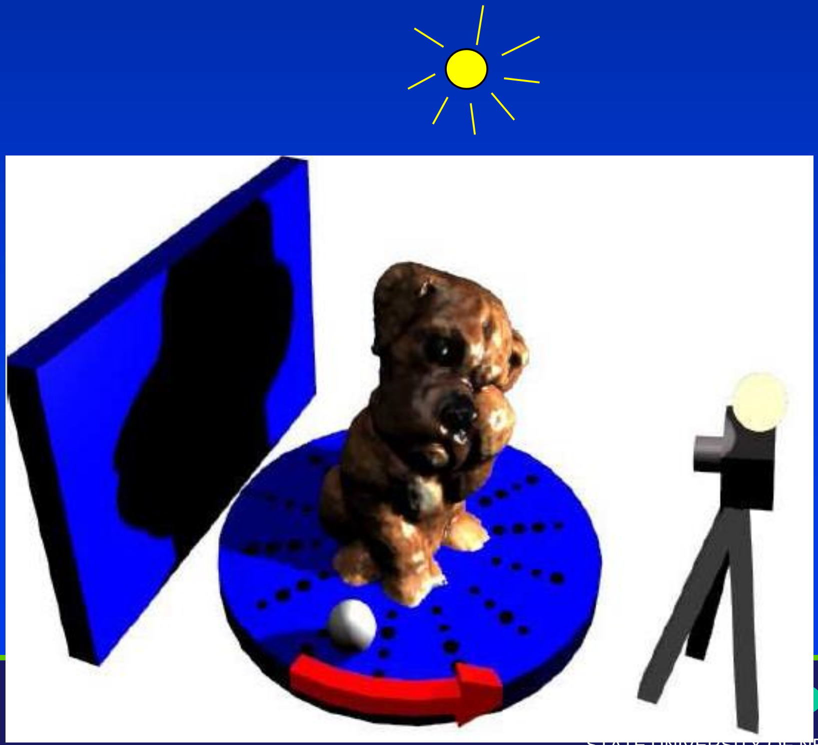
Hong Qin
Department of Computer Science
Stony Brook University (SUNY at Stony Brook)
Stony Brook, New York 11794-2424
Tel: (631)632-8450; Fax: (631)632-8334
qin@cs.stonybrook.edu
http://www.cs.stonybrook.edu/~qin

Key Elements of Cameras and Geometric Coordinate Systems

Image Formation

- Camera
- Light, shape, reflectance, texture

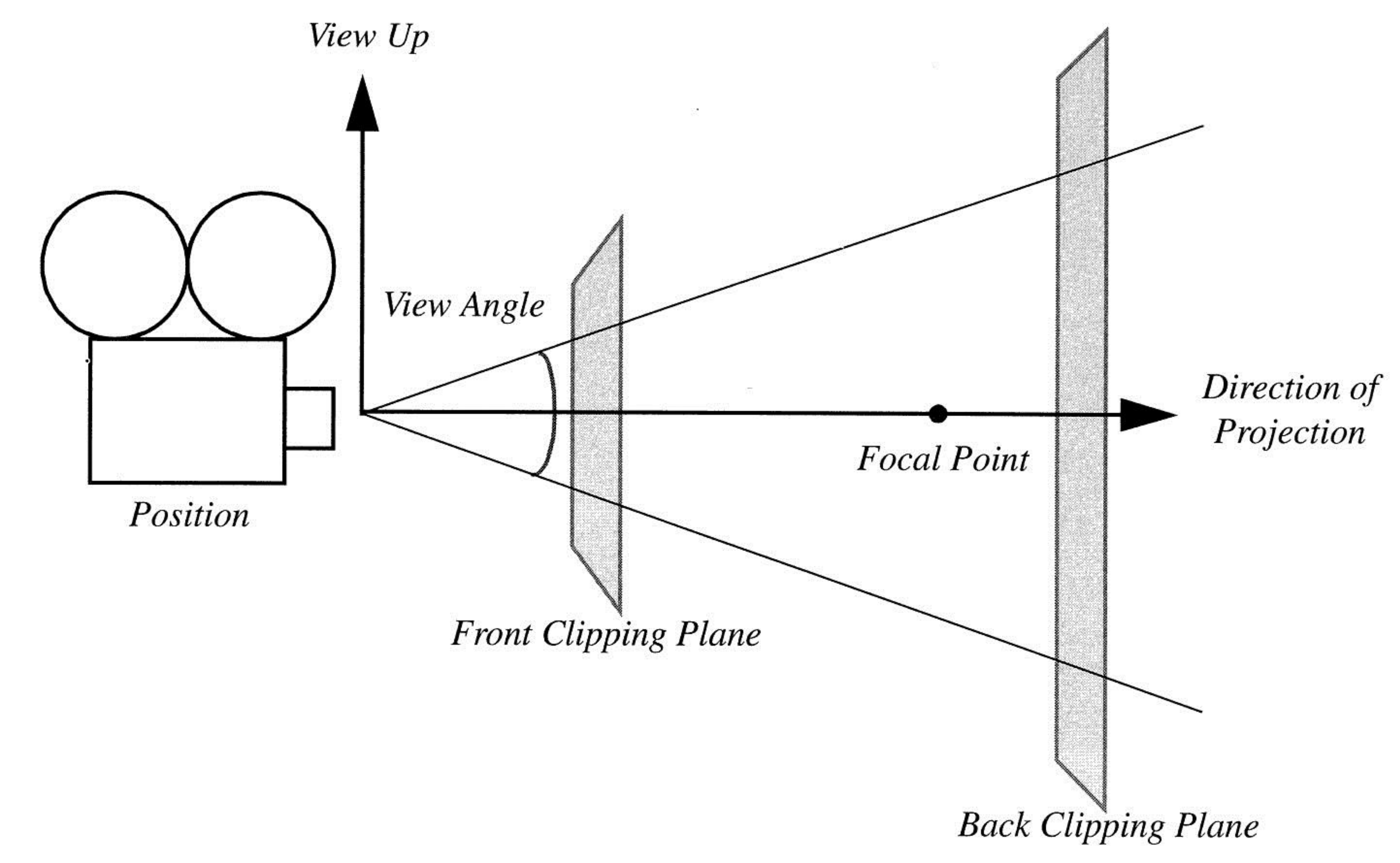
Image formation



Cameras

- We have light sources that illuminate 3D objects (or datasets) in our virtual scene within the graphics system
- Light rays interact with surface properties and generate colors according to the illumination model
- But how do we view the scene, select the position and orientation of the viewpoint?
- This is where the virtual camera comes in

Basic Camera Attributes

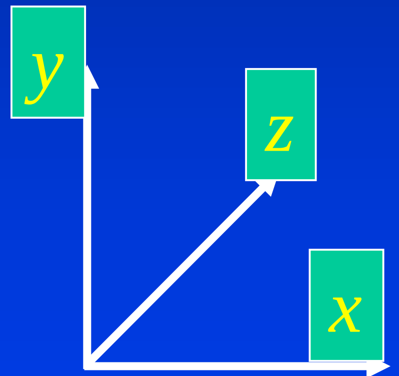


Basic Camera Attributes

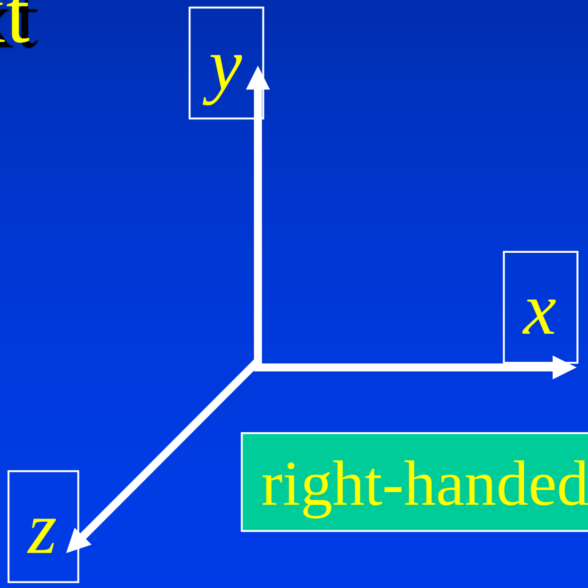
- *Position* – given in (x,y,z) coordinates
- *Up-vector* – orients the camera, given in (x,y,z)
- *Direction of projection* – points the camera in some (x,y,z) direction; also called *viewing direction*
- Why is the up-vector needed if we have a direction of projection?
- Why is the direction of projection needed if we have an up-vector?

Coordinate Systems

- Two kinds of Cartesian coordinate systems: right-handed and left-handed
- Use whichever coordinate system seems most natural in the given context



left-handed system



right-handed system

Coordinate Systems

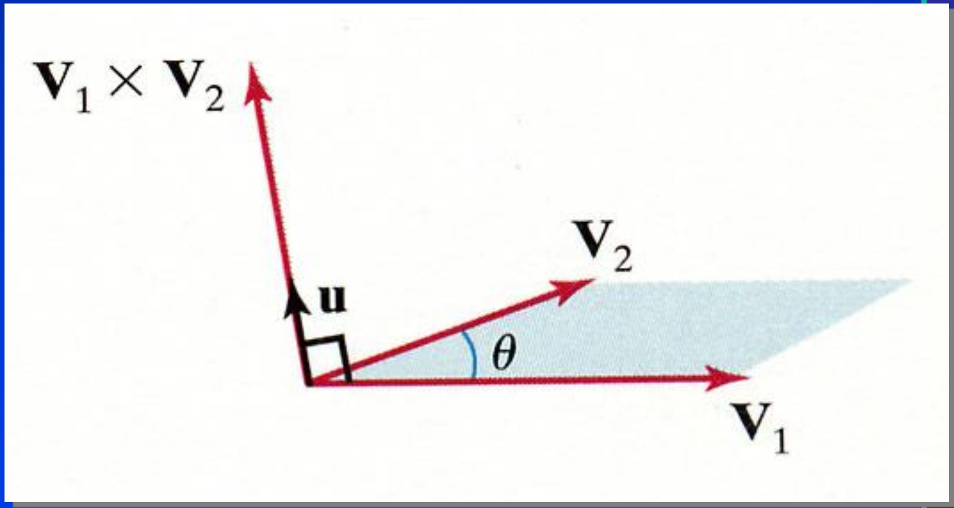
- You might be familiar with different types of coordinate systems:
  - Cartesian
  - Polar
  - Spherical
  - Cylindrical
- Computer graphics and visualization applications use several distinct coordinate systems: *model*, *world*, *view* and *display*
- Usually they use Cartesian coordinates

Basic Camera Attributes

- *Front and back clipping planes* – determine which objects *might* be visible
- Planes perpendicular to viewing direction
- Specified as distances along viewing direction
- Also called *near and far clipping planes*
- Objects on near side of front clipping plane and on far side of back clipping plane are invisible
- Objects between the clipping planes may occlude each other and may be fully visible, partially visible, or invisible

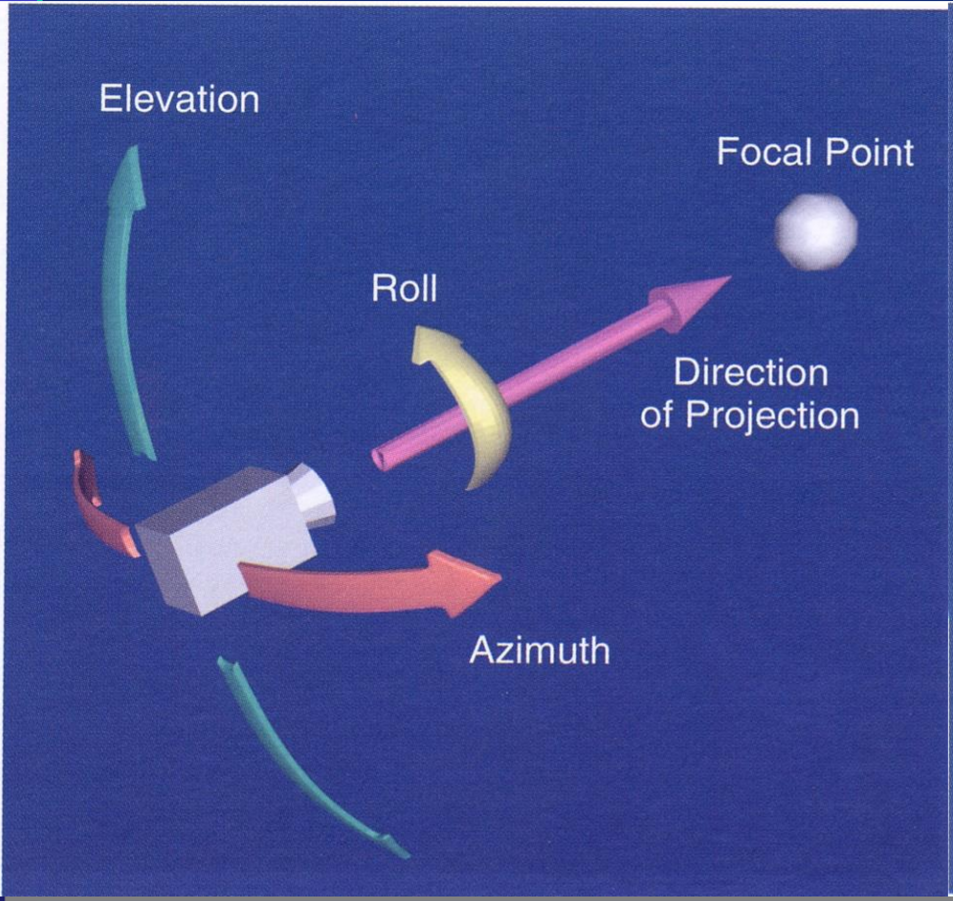
Camera Manipulation

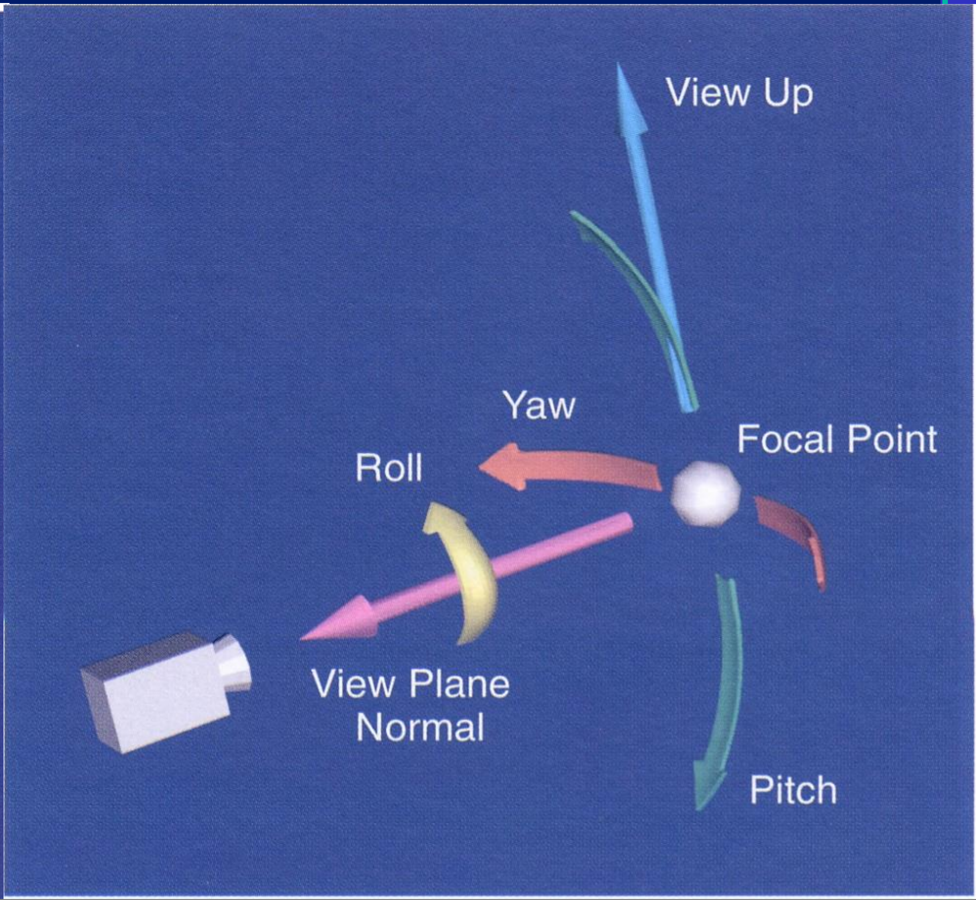
- Changing *azimuth* = rotating camera's position around its view vector w.r.t. focal point
- Changing *elevation* = rotating camera's position around cross-product of view direction and up-vector
- Cross-product of two vectors provides vector in dir. perpendicular to two original vectors



- Changing *roll* = rotate camera's up-vector about the viewing direction (*twisting* the camera)

Camera Manipulation



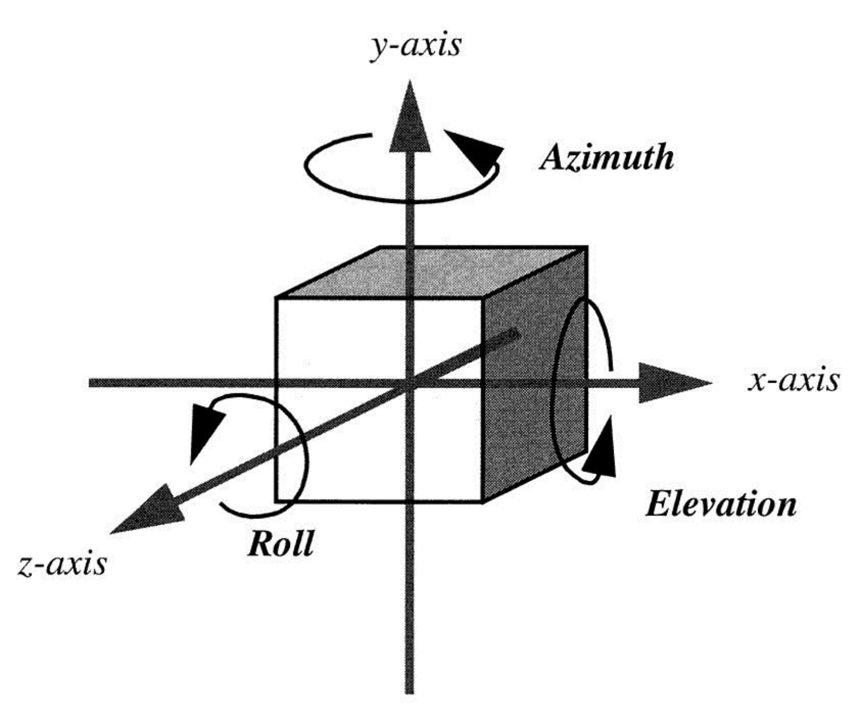


Camera Manipulation

- Changing *yaw* = rotating focal point about the up-vector
- Changing *pitch* = rotating focal point about cross product of view vector and up vector
- *Dollying* – moves camera position along view vector (dollying in and out)
- Once camera attributes are set, objects are *projected* from 3D onto the 2D image plane
- Camera attributes determine which rays of light (that bounced off objects) will enter the camera and contribute to the rendered image

Actor Geometry: Actor Location and Orientation

- Rotations take place around the origin of the actor
- They are applied as a camera azimuth, elevation and roll, *in that order* – remember, order counts!
- VTK uses this orientation vector-based approach since it is very natural to manipulate objects in this fashion



Camera Manipulation

- Nuisance to manipulate the camera by changing all those parameters
- Usually its easier to specify camera movements with respect to the camera's *focal point*, the position in space which the camera is pointing at
- Consider taking a portrait (physical analogy):
  - Move around the person
  - Move forward and backward w.r.t. to person
  - Move camera up and down
  - Rotate camera while standing still

Model Coordinate System

- Coordinate system used to define an object or actor
- Coordinate system will be a natural choice
  - Example: A football might be described using a cylindrical coordinate system
  - What coordinate system might we use for a planet?
- System choice of person who created the object
- Units are application-dependent: inches, meters, cubits, etc.

Object Representations

- List of vertices: v_1, v_2, …, v_n, each given as (x_i, y_i, z_i)
- List of edges: (v_1, v_3), (v_4, v_7), …, (v_i, v_j),…
- List of faces: (e_1, e_3, e_4), (e_2, e_5, e_8), …OR
- List of faces: (v_1, v_3, v_5), (v_6, v_7, v_9), …
- When a vertex's position is changed due to transformation, all edges and polygons that include the vertex are consequently changed
- If we apply the same transformations to all vertices, the entire polygonal mesh moves as a unit, which is what we want

World Coordinate System

- 3D space in which our actors are positioned
- Each actor's model coordinate system has some position and orientation inside the world space
- Many model coordinate systems, only one world coordinate system
- Each actor rotates, scales and translates itself into the world coordinate system
- Lights and cameras are specified with respect to the world coordinate system
- Does a camera have its own coordinate system?

World Coordinate System

- Example:
  - Specify each of our bodies with a cylindrical coordinate system with the head as the origin
  - We position ourselves in the room (the world coordinate system) by giving the position of our heads w.r.t. the origin of the room (perhaps some corner)

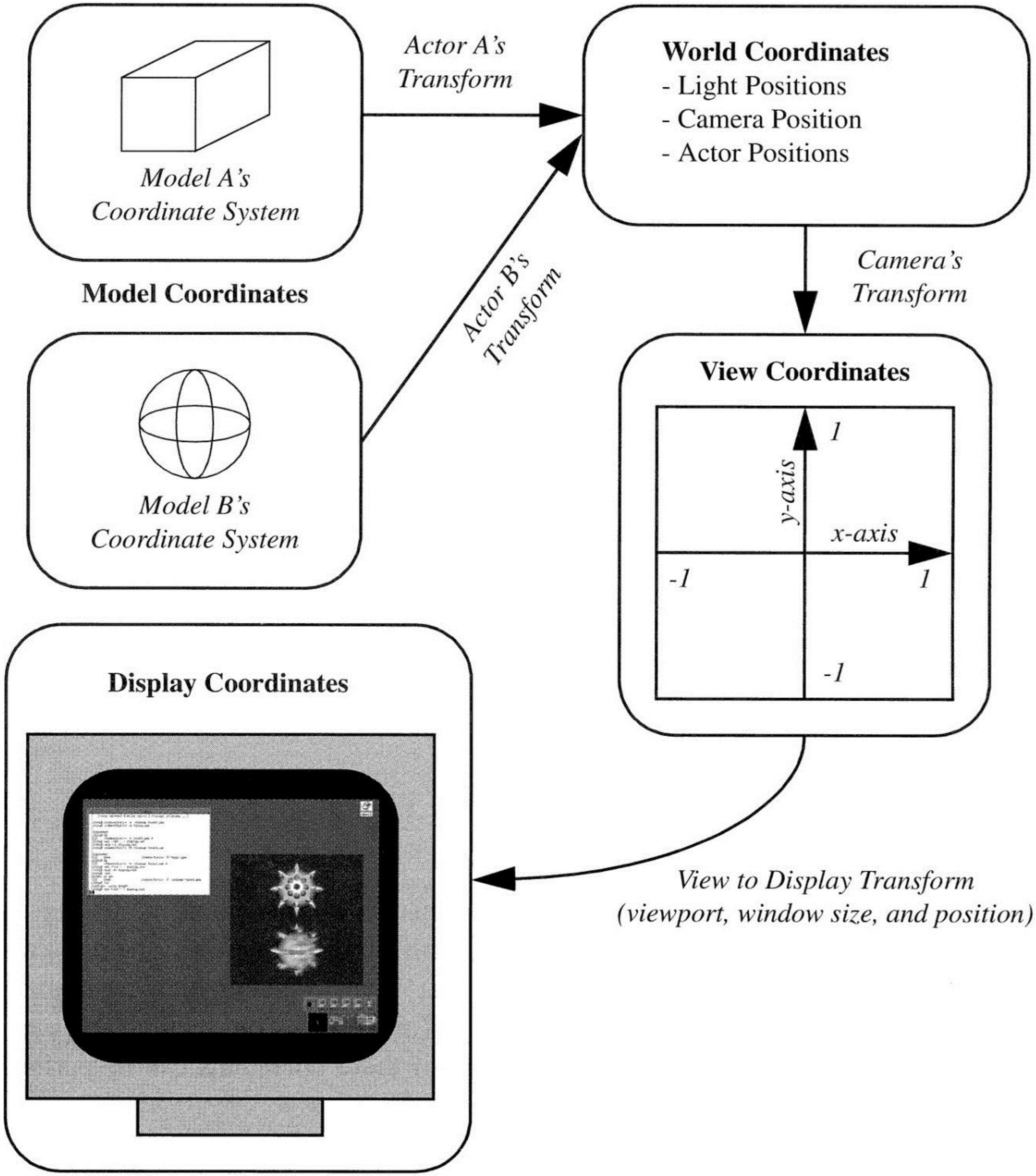
View Coordinate System

- Represents what is visible to the camera
- Given by (x,y,z) values
- x, y in [-1, 1]
- z is some depth > 0
- x, y give location of some object in the image plane
- z give distance of object from camera
- A matrix is used to convert from world coordinates into view coordinates (i.e., projection!)
- Perspective effect can be accommodated by this matrix

Display Coordinate System

- x, y are pixel values on screen
- z is still the depth
- What are restrictions on x and y?
- Window size helps determine valid range for x, y
- Display can be divided into multiple *viewports*, each of which has its own coordinate system
- Must select which viewport is used for rendering

Coordinate Systems



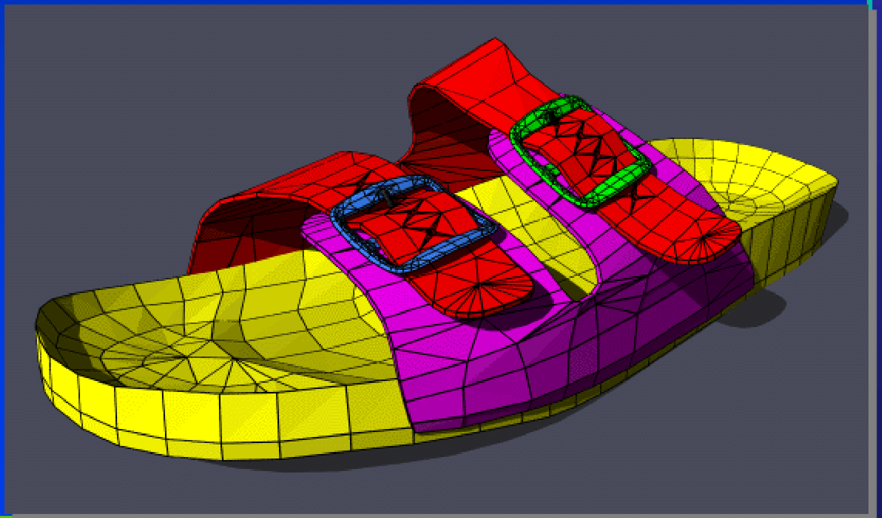
Coordinate Systems (Computer Graphics Pipeline)

1. Model coordinates are transformed into
2. World coordinates, which are transformed into
3. View coordinates, which are transformed into
4. Display coordinates, which correspond to pixel positions on the screen

- Transformations from one coordinate system to another take place via *coordinate transformations*, which we'll look at now

Coordinate Transformations

- *Coordinate transformations* allow us to *translate, scale* and *rotate* our models in our virtual scene
- In Computer Graphics and Visualization, objects are often represented as meshes consisting of polygons, edges and vertices
- Two vertices define an edge
- Three or more edges define a polygon
- To transform an object, we apply the transformations to the vertices of the mesh



Coordinate Transformations

- Rather than represent 3D points using three coordinates (x,y,z), we will use four: (x,y,z,w)
- This approach is called *homogeneous coordinates*
- Transformations will be represented by (4 x 4) matrices
- Why not (3 x 3)?
- Because some transformations – including translation – cannot be represented by (3 x 3) matrices
- Most of the time $w = 1$, but there are special transformations for which $w \neq 1$

Coordinate Transformations: Translation

- Suppose we wish to translate the point (x,y,z) by the vector (t_x, t_y, t_z)
- This *translation transformation* can be described by the *translation matrix:*

$$T_T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Coordinate Transformations: Translation

- The new position is given by post-multiplying our point by the translation matrix:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- The new position of our point is (x', y', z')

Coordinate Transformations: Translation

- We can see that the matrix-vector multiplication is equivalent to the following formulas:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \\ z' &= z + t_z \end{aligned}$$

Coordinate Transformations: Scaling

- We can scale a mesh by applying the *scaling transformation* to each of its vertices:

$$T_S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Coordinate Transformations: Scaling

- When $s_x = s_y = s_z$, we call it *uniform scaling*
- Otherwise, we have *non-uniform scaling*

$$T_S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Suppose someone said to you that it makes no sense to apply scaling to vertices
- After all, how do you scale a 3D point, which has no width, height or depth?

Coordinate Transformations: Rotation

- We can rotate a vertex about one of the major axes by some angle θ using one of the *rotation matrices*:

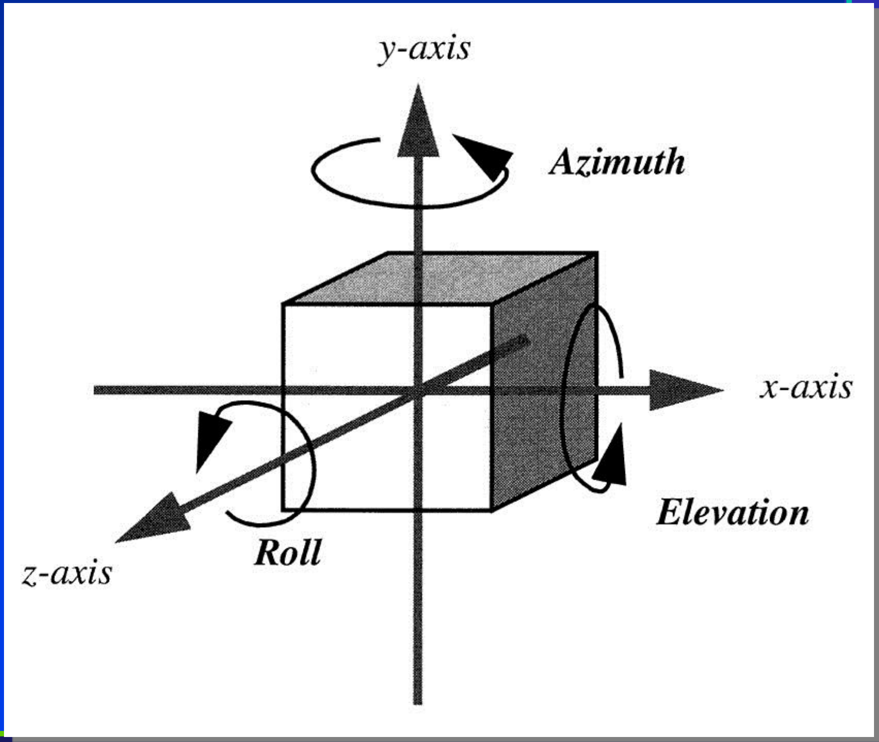
$$T_{R_x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{R_y} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{R_z} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Actor Geometry: Actor Location and Orientation

- The modeling transformations we looked at earlier allow us to change the location and orientation of objects
- It's often useful to associate (i.e., store) an *orientation vector* (O_x, O_y, O_z) for each actor
- This vector implicitly defines the three rotation matrices



Coordinate Transformations

- Transformations can be *composed* by *right-multiplying* transformation matrices
- Example: a sequence ($S\ R_z\ T\ R_y$) would indicate:

1. A rotation about the Y axis, followed by
2. A translation, followed by
3. A rotation about the Z axis, followed by
4. A scaling

- So beware and remember: matrix multiplication is associative but it isn't commutative

Coordinate Transformations

- The above transformations can be applied to objects in the scene – these are referred to as the *modeling transformations*
- The camera (viewpoint) can also be transformed by the *viewing transformation*
- What transformation(s) might not make sense to apply to the viewpoint?
- *Projection transformation* is applied after modeling transformations to project the 3D actors onto the screen
- We won't study projection transformations in greater details in this course

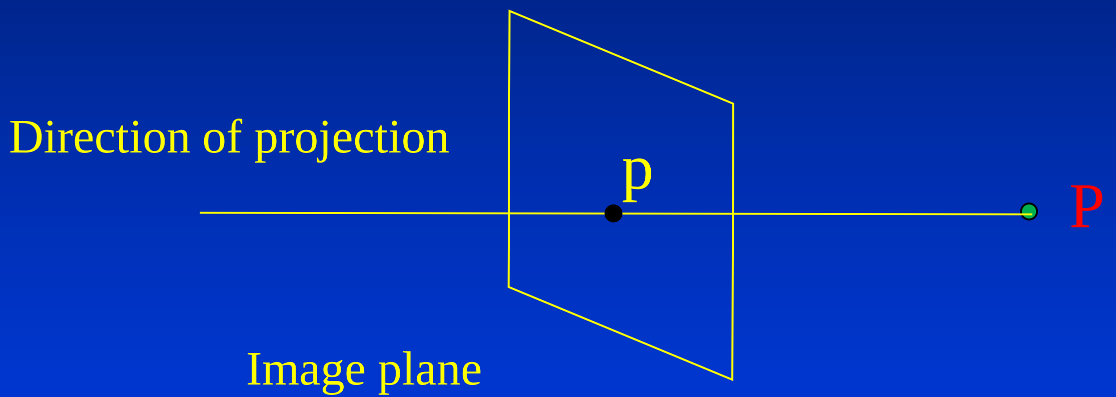
Camera Attributes

- *Projection* – method of projection determines how 3D objects are drawn on the *image plane*, or screen
- *Orthographic projection* – all rays of light are parallel to the projection vector
- 3D points are projected onto the screen along the same direction
- The perceived size of an object is not a function of its distance from the camera

Orthographic Camera Model

Infinite Projection matrix - last row is (0,0,0,1)

Good Approximations – object is far from the camera (relative to its size)

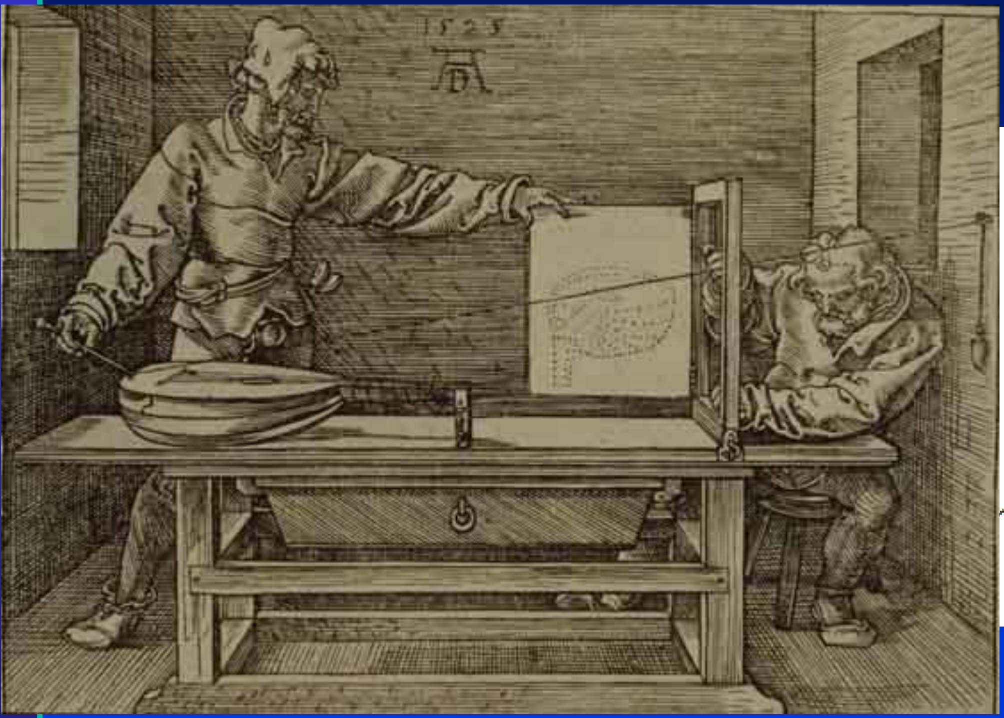


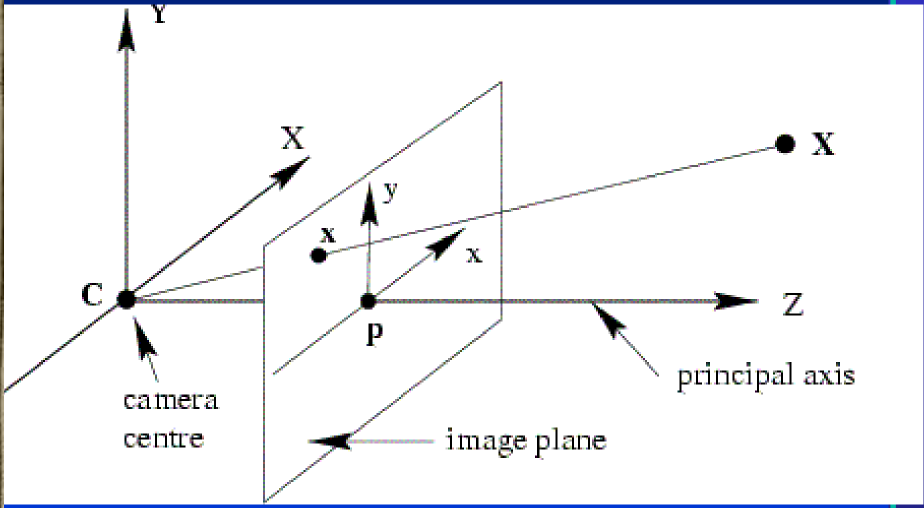
$$P_{orth} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera Attributes

- *Perspective projection* – all light rays travel through a central point, such as the viewpoint
- Objects appear smaller as their distances increase from the viewpoint, and vice versa
- This is what happens in real life
- Simulating perspective projection requires a *view angle*
- View angle and clipping planes define a *view frustum*, a truncated pyramid; one type of *viewing volume*
- In orthographic projection, we have a rectangular view volume instead because the light rays are parallel!!!

Projective Camera Model





$\mathbf{x} = P\mathbf{X} \quad P : 3\times 4$ **Projection matrix**

$\mathbf{x} = K[R \quad \mathbf{t}]\mathbf{X} \quad K : 3\times 3$ **Camera matrix (int. parameters)**

$R, \mathbf{t}$ **Rotation, translation (ext. parameters)**