Viewing II

The Synthetic Camera

Modeling and Rendering

- Modeling
  - Create models
  - Apply materials to models
  - Place models around scene
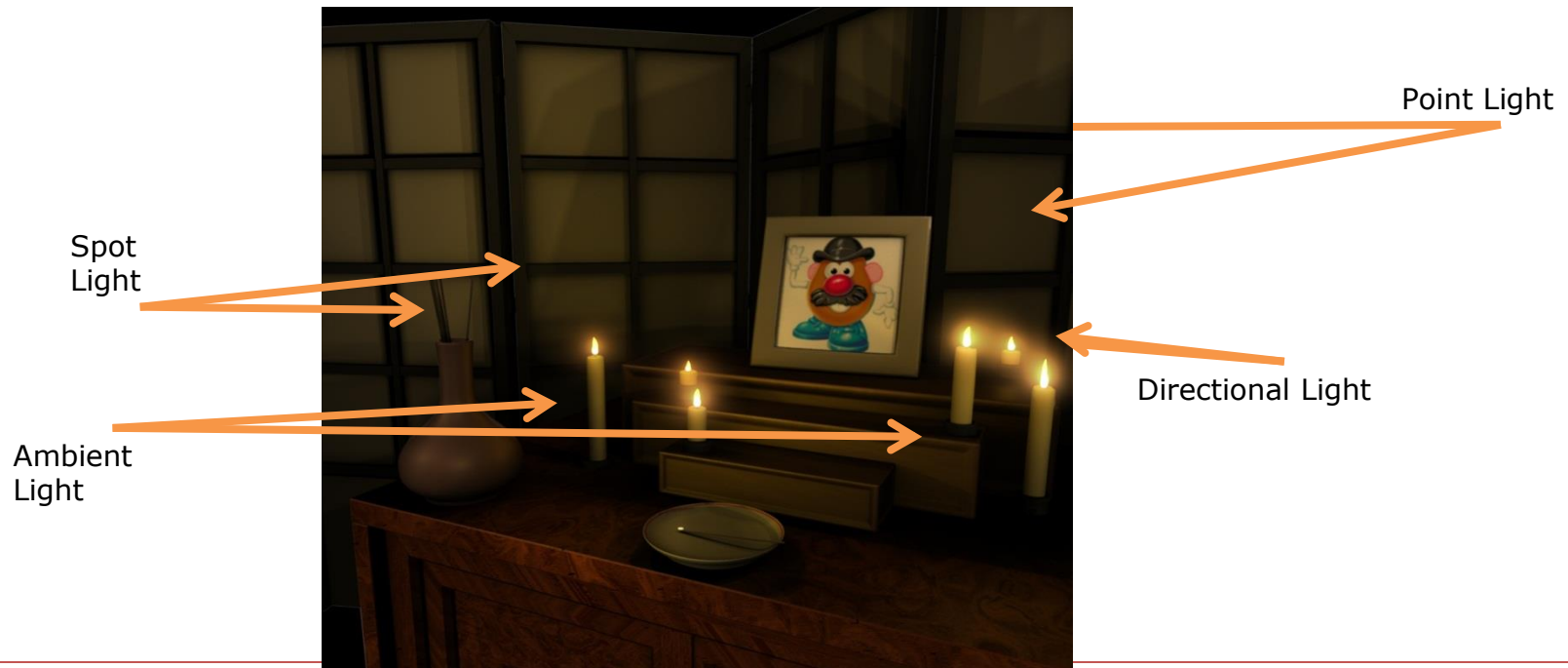 - Place lights in scene

- Place the camera

- **Rendering**

  Take "picture" with camera

  This involves:

  - Projecting the objects onto the image plane

  - Finding the right color for each pixel (Visible Surface Determination, lighting, etc)

CS128 lighting assignment by Patrick Doran, Spring 2009

Point Light

Spot Light

Directional Light

Ambient Light

2

The camera and the scene

▸ What does a camera do?

  ▸ Takes in a 3D scene

 ▸ Places (i.e., projects) the scene onto a 2D medium (a roll of film or a digital pixel array)
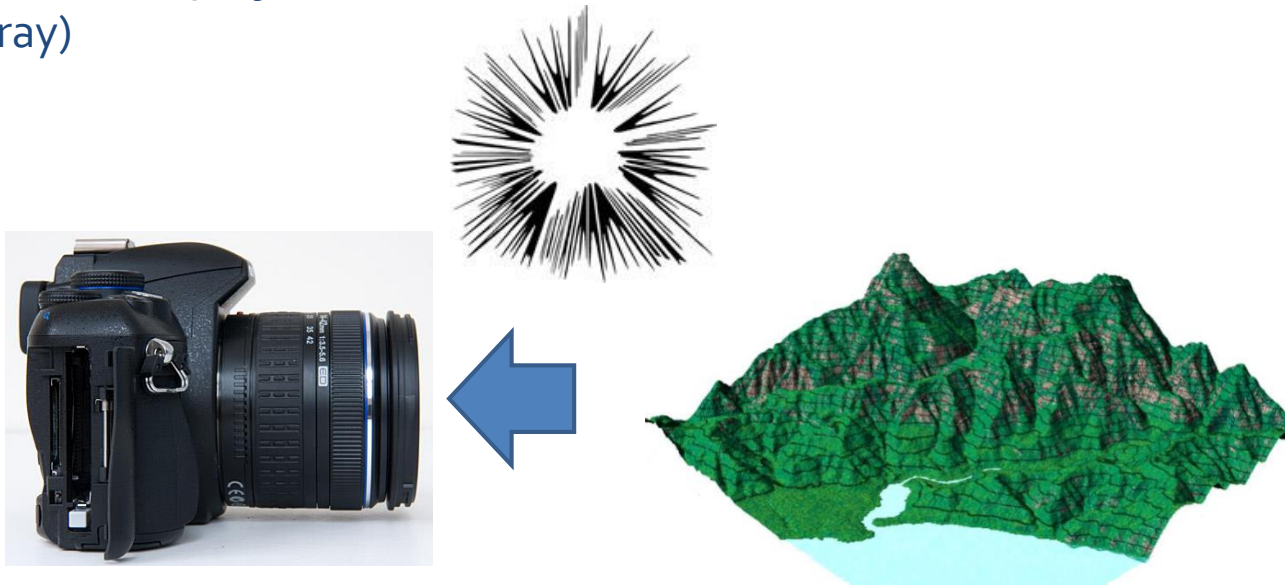
▸ The synthetic camera is programmer's model for specifying how a 3D scene is projected onto screen

3D Viewing: The Synthetic Camera

▸ General synthetic camera: each package has its own but they are all (nearly) equivalent

  ▸ Camera position

  ▸ Orientation

  ▸ Field of view (angle of view, wide, narrow, normal…)

  ▸ Depth of field (near distance, far distance)

  ▸ Focal distance

  ▸ Tilt of view/ film plane (if not perpendicular to viewing direction, produces oblique projections)

  ▸ Perspective of parallel projection (Camera near objects or infinite distance away

▸ EPL607 uses a simpler slightly less powerful model than the one used in the book

 ▸ Omit tilt of view/film plane, focal distance (blurring)

Cameras in the rendering process
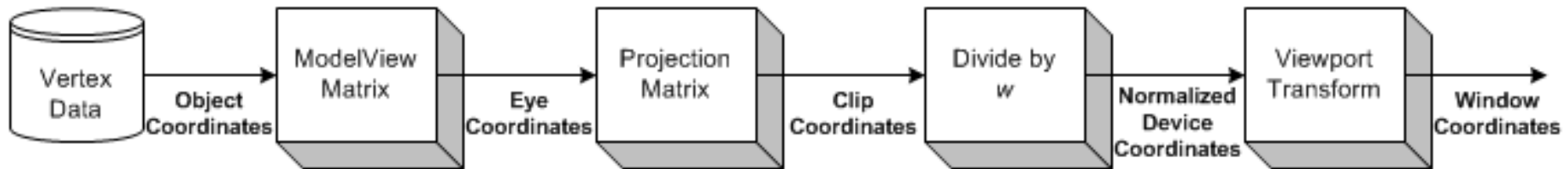
▶ We will talk about the coordinate system involved with cameras, the view volume specifications, projecting on to the film plane, and transforming into the viewport

▶ Let's first define view volume and viewport

The pinhole model

▸ You can think of camera as a pinhole. As you look through pinhole you see a certain volume of space

▸ Rays of light reflect off objects and converge to pinhole to let you see the scene

▸ Rays of light can be considered **projectors**, that converge towards the pinhole as center of projection

▸ The pinhole is where our camera position will be and the volume we see will be our "view volume"

▸ Projectors intersect a plane, usually in between scene and pinhole, to get projected onto that plane

▸ Lastly, in synthetic camera the projection is mapped to some form of viewing medium (screen)

Object

Pinhole

Image proj
on image plane
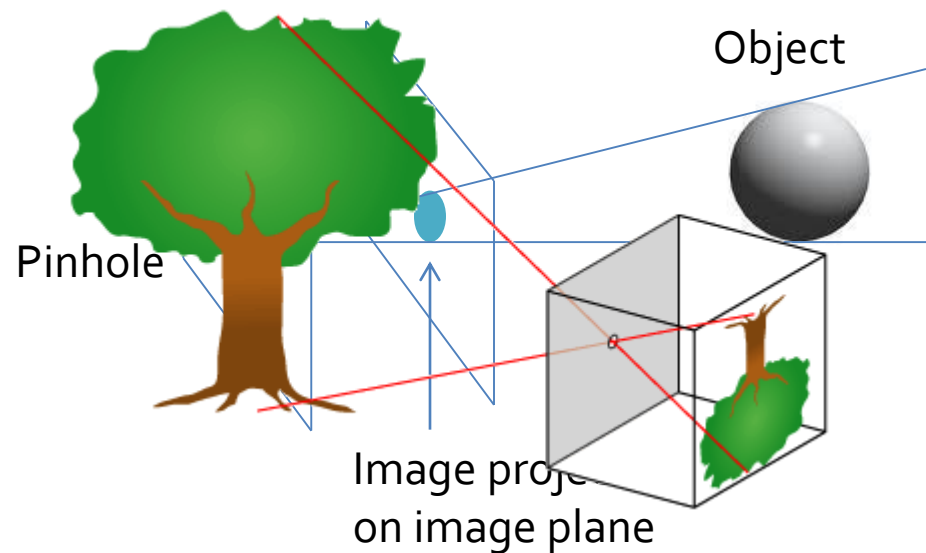
View Volumes (the focus of today's lecture)

▶ A view volume contains everything the camera sees

▶ Conical – Approximates what eyes see, expensive math when clipping objects against cone's surface (simultaneous quadratics)

▶ Can approximate this using a rectangular frustum view volume

- ▶ Simultaneous linear equations for easy clipping of objects against sides (stay tuned for clipping lecture)

▶ Also parallel view volumes, for orthographic projections. These don't simulate eye or camera

eye

conical perspective view volume (eye's is much wider)

frustum approximation, view volume

synthetic camera

View volume
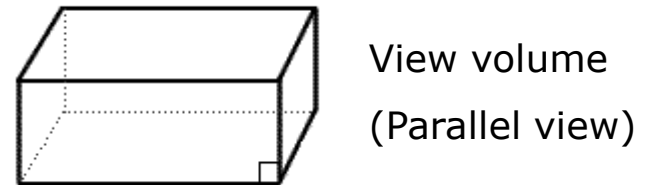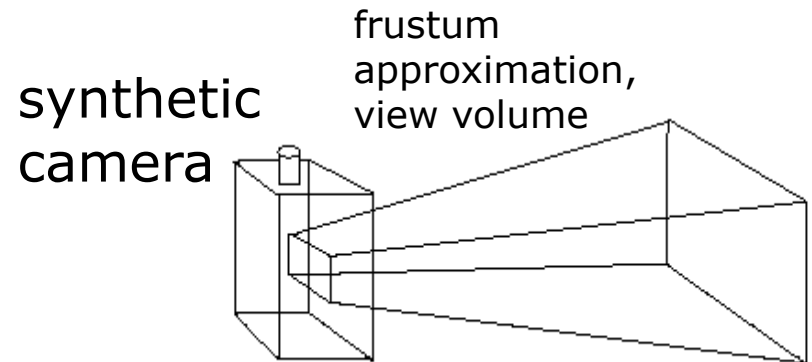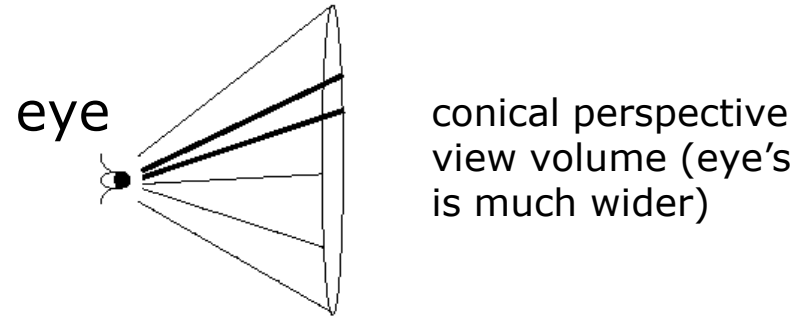
(Parallel view)

View Volumes and Projectors

▸ Given our view volume we need to start thinking about how to project scene contained in volume to film plane

▸ **Projectors**: Lines that essentially map points in scene to points on film plane

▸ **Parallel Volumes:** Parallel Projectors, no matter how far away an object is, as long as it is in the view volume it will appear as same size, (using our simple camera model, these projectors are also parallel to look vector)

▸ **Perspective Volumes**: Projectors converge on eye point = center of projection, like rays of light converging to your eye
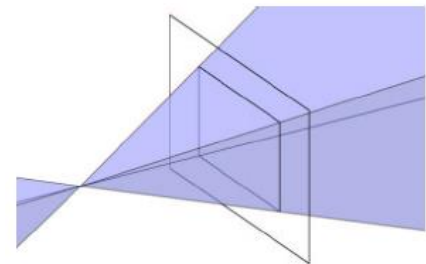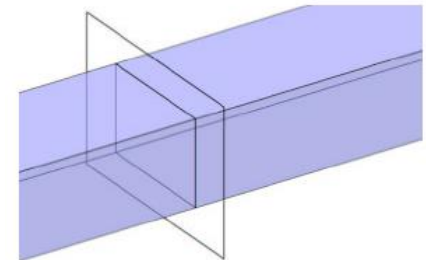
The film plane

▸ Film plane is a plane in world space – 3D scene is projected onto a rectangle (the film) on that plane using some projection transformation and from there onto the viewport on screen

▸ Film for our camera model will be perpendicular to and centered around the camera's look vector, and will match dimensions of our view volume
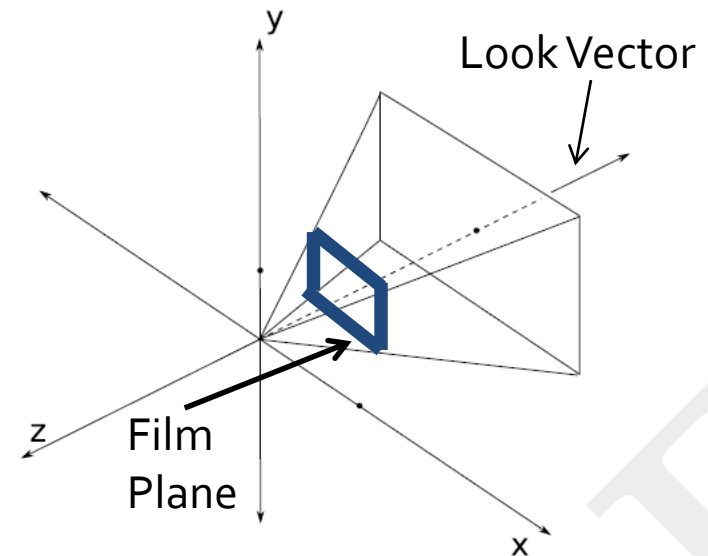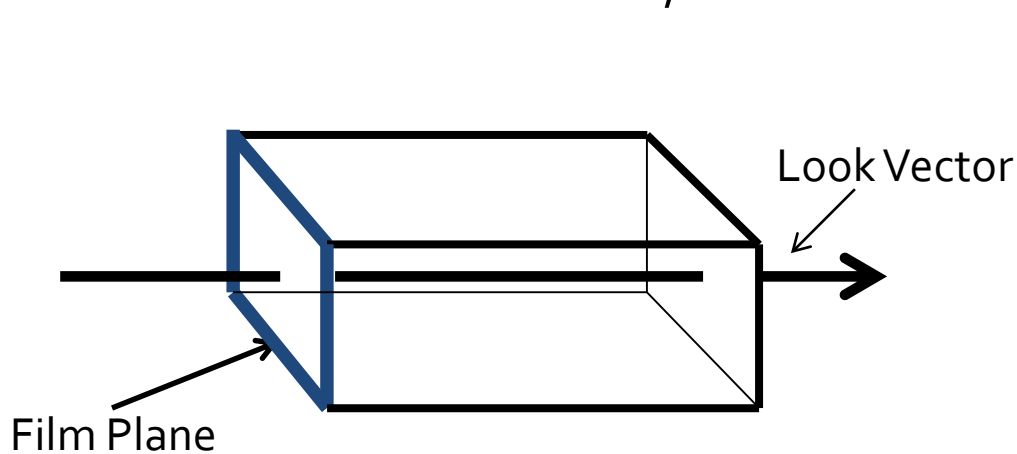
▸ Actual location of film plane along look vector doesn't matter as long as it is in front of scene

Προοπτική προβολή (απλή περίπτωση)

- In order to get a natural looking image we need the perspective

- For a simple arrangement it is easy to find the projection
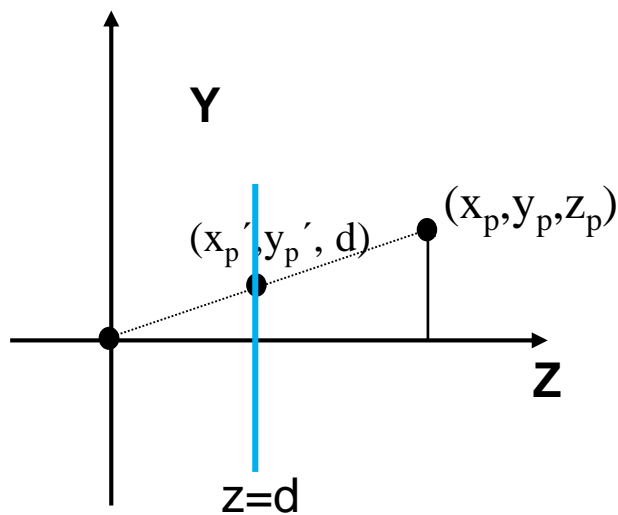
$$\frac{y_p{}'}{d} = \frac{y_p}{z_p} \qquad \Rightarrow \qquad y_p{}' = d\,\frac{y_p}{z_p}$$

$$\frac{x_p{}'}{d} = \frac{x_p}{z_p} \qquad \Rightarrow \qquad x_p{}' = d\,\frac{x_p}{z_p}$$

Y

$(x_p{}', y_p{}', d)$

(x_p, y_p, z_p)

Z

z=d

The viewport

▸ Viewport is the rectangular area of screen where a scene is rendered

  ▸ may or may not fill Window Manager's window

  ▸ note: *window (aka. Imaging Rectangle – Spike)* in computer graphics used to means a 2D clip rectangle on a 2D world coordinate drawing, and *viewport* is 2D integer coordinate region of screen space to which clipped window contents are mapped.

 ▸ Pixel coordinates for viewport are most commonly referred to using the (u, v) coordinate system

Constructing the view volume (1/2)

▸ We need to know six things about our synthetic camera model in order to take a picture using our perspective view **frustum**

Up vector

Look vector
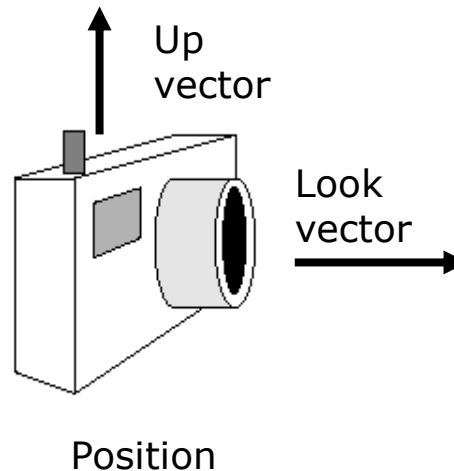
Position

1) *Position* of camera (from where it's looking)

2) *Look vector* specifies direction camera is pointing

3) Camera's *Orientation* is determined by *Look vector* and angle through which the camera is rotated about that vector, i.e., the direction of *Up vector*

Constructing the view volume (2/2)
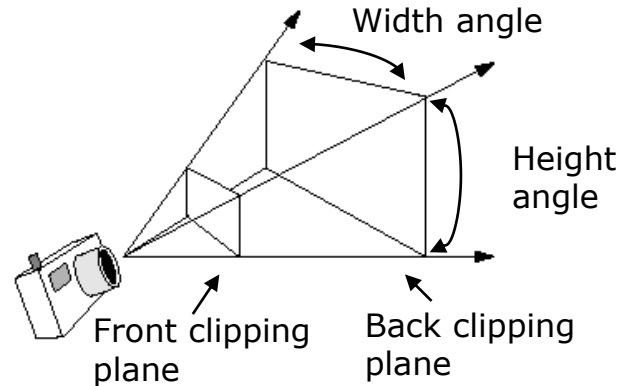
4)   *Aspect ratio* of the electronic "film:" ratio of width to height

5)   *Height angle:* determines how much of the scene we will fit into our view volume; larger height angles fit more of the scene into the view volume (width angle determined by height angle and aspect ratio)

   ▸   the greater the angle, the greater the amount of perspective distortion

6)   *Front* and *back clipping planes:* limit extent of camera's view by rendering (parts of) objects lying between them and throwing away everything outside of them

Optional: Focal length, objects at length are sharp, objects closer/farther blurry

# 1) Position (1/1)

▶ Where is the camera located with respect to the origin?

▶ For our camera in 3D space we use a right-handed coordinate system

  ▸ Open your right hand, align your palm and thumb with the $+x$ axis, point your index finger up along the $+y$ axis, and point your middle finger towards the $+z$ axis

  ▸ If you're looking at a screen the z axis will be positive coming towards you
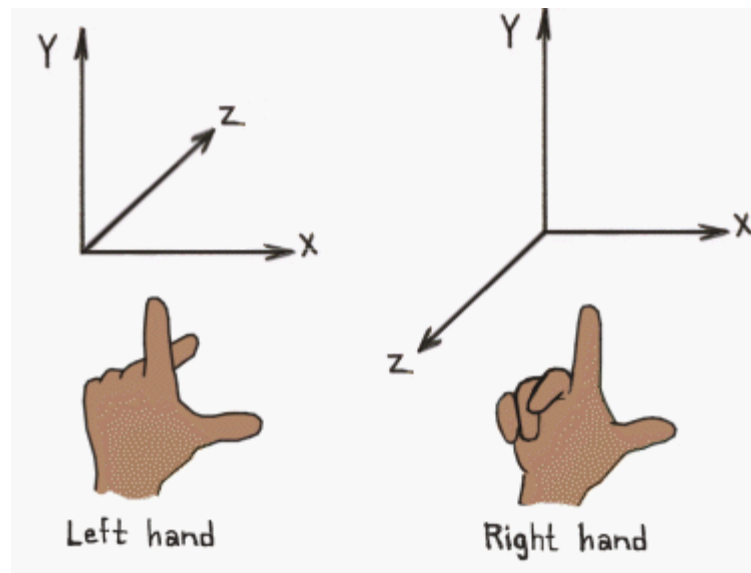
2 & 3) Orientation: Look and Up vectors (1/2)

‣ Orientation is specified by a point in 3D space to look at (or a direction to look in) and an angle of rotation about this direction

‣ These correspond to the look vector and up vectors
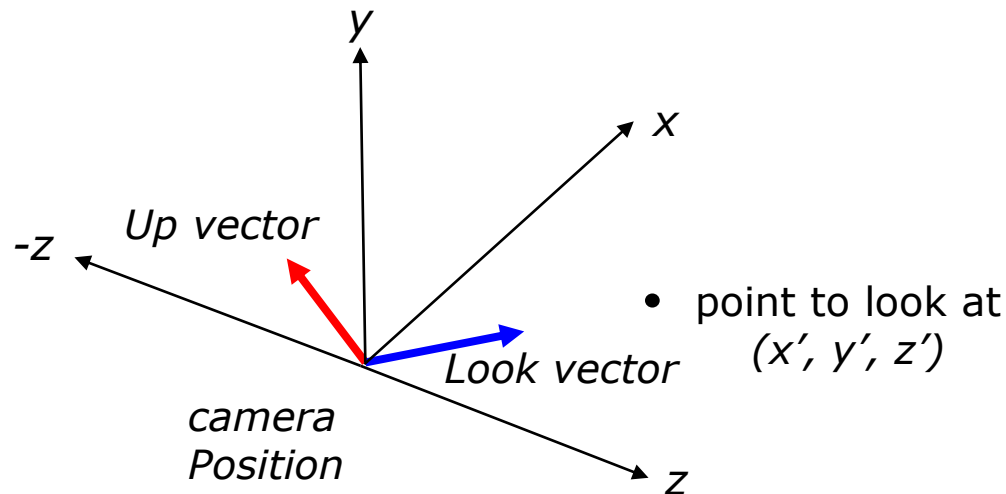
2 & 3) Orientation: Look and Up vectors (2/2)

▸ *Look Vector*

- ▸ Direction the camera is pointing
- ▸ Three degrees of freedom; can be any vector in 3-space

▸ *Up Vector*

- ▸ determines how camera is rotated around *Look vector*
- ▸ for example, holding camera horizontally or vertically
- ▸ *Up vector* must not be parallel to *Look vector* but it doesn't have to be perpendicular either— actual orientation will be defined by part of vector perpendicular to look vector, lying in plane with *Look* as normal

Any of $v, v1, v2, v3$
could be the up vector

Look Vector

Up vector

Look vector

Projection
of Up vector
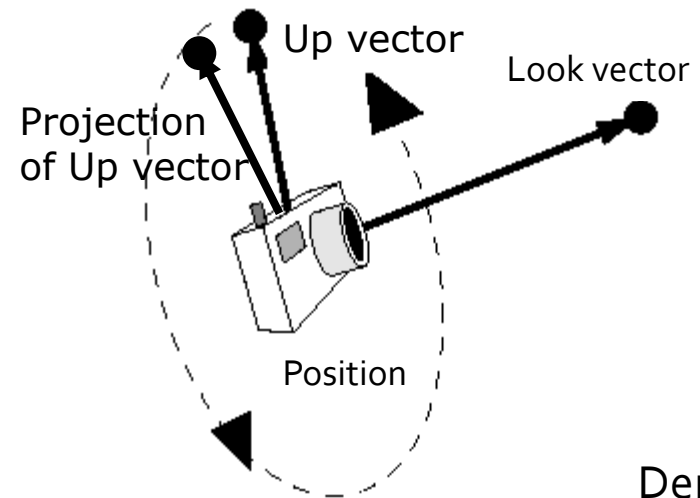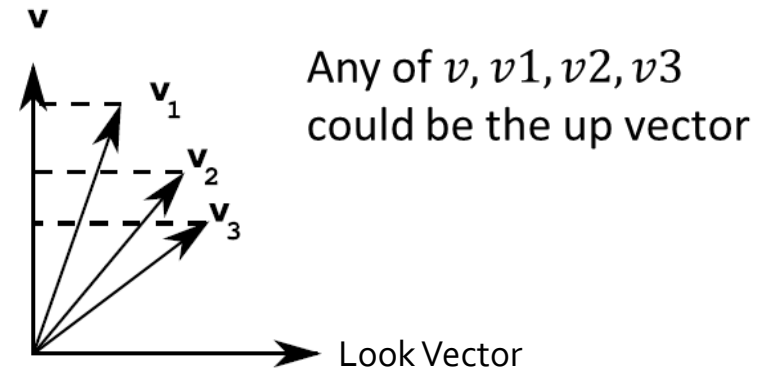
Position

Demo!

The camera coordinate space (1/2)

▸ The equivalent of x, y and z axes in camera space are unit vectors u, v and w (not to be confused with the homogenous coordinate, w)

  ▸ **Also a right handed coordinate system**

  ▸ w is a unit vector in the opposite direction of the look vector

  ▸ v is the part of the up vector perpendicular to the look vector, normalized to unit length

  ▸ u is the unit vector perpendicular to both v and w
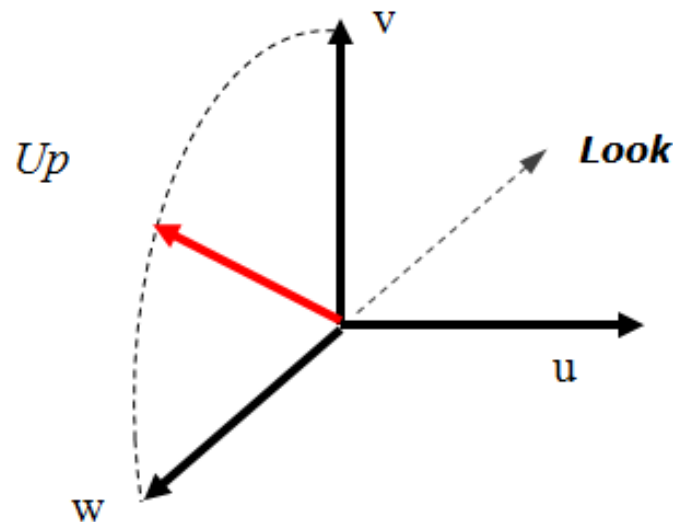
The camera coordinate space (2/2)

▸ There are three common transformations that use the camera space axes

▸ Roll:
  ▸ Rotating your camera around w

▸ Yaw:
  ▸ Rotating your camera around v

▸ Pitch:
  ▸ Rotating your camera around u
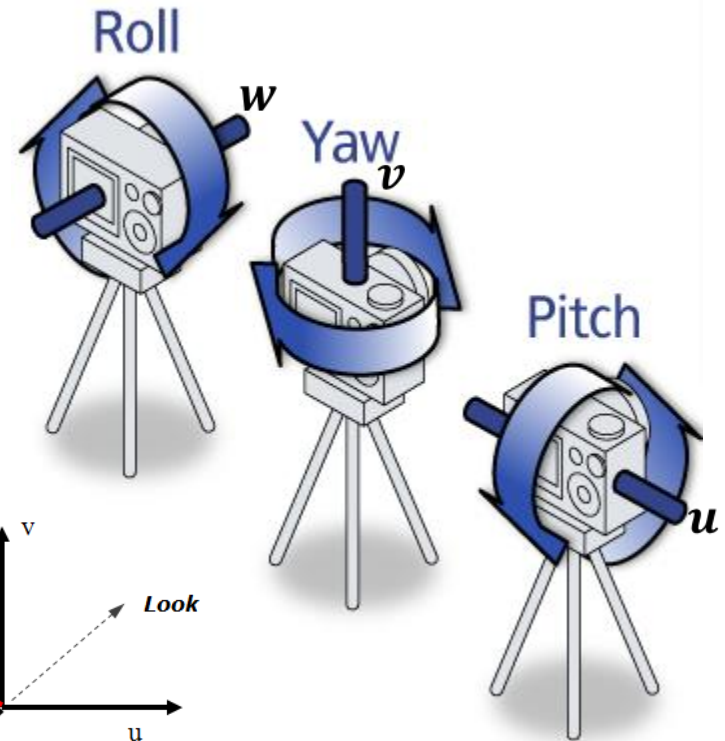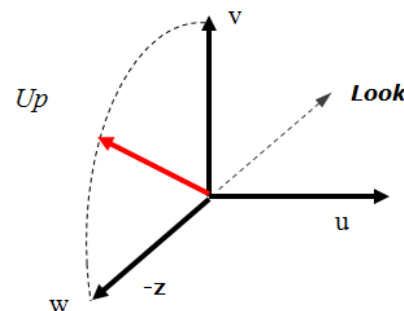
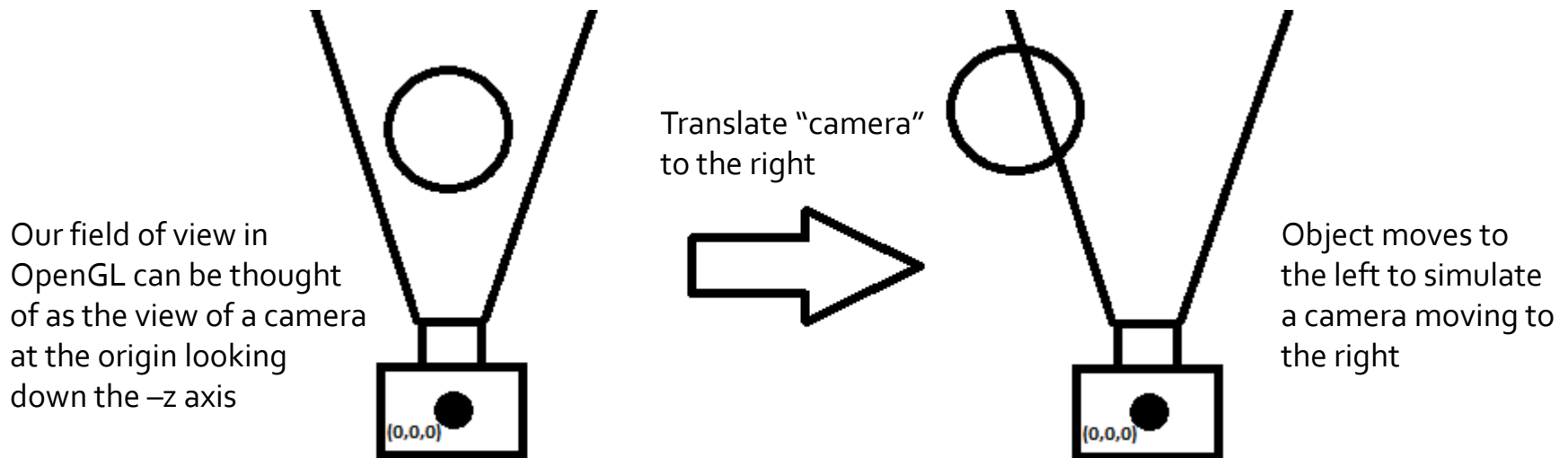If we send the camera to the origin and align the axes with the standard axes, we can use our rotation matrices to perform these transformations and then un-align and un-translate.

Aside: The Camera as a model

▸ There are different ways we can model a camera

▸ In the generalized model we have a camera and a scene where both the camera and objects in the scene are free to be transformed independently

▸ In a more restricted model we have a camera that remains fixed in one position and orientation

  ▸ To transform the camera we actually apply the inverse transformation to objects in the scene

▸ This is the model OpenGL uses, note however that GLUT abstracts this concept away from the programmer (gluLookAt())

Translate "camera" to the right

Our field of view in OpenGL can be thought of as the view of a camera at the origin looking down the –z axis

Object moves to the left to simulate a camera moving to the right

(0,0,0)

(0,0,0)

4) Aspect Ratio (1/1)

▸ Analogous to dimensions of film used in a camera

▸ Is defined as ratio of width to height of your viewing window

▸ Viewport's aspect ratio is usually defined by the device being used

 ▸ A square viewing window has a ratio of 1:1

 ▸ NTSC TV is 4:3, HDTV is 16:9 or 16:10

▸ Aspect ratio of viewing window defines the dimensions of image that gets projected to that film plane, after which it is mapped to your viewport

 ▸ typically it's a good idea to have the same aspect ratio for both your viewing window and viewport, to avoid distortions/stretching

1:1
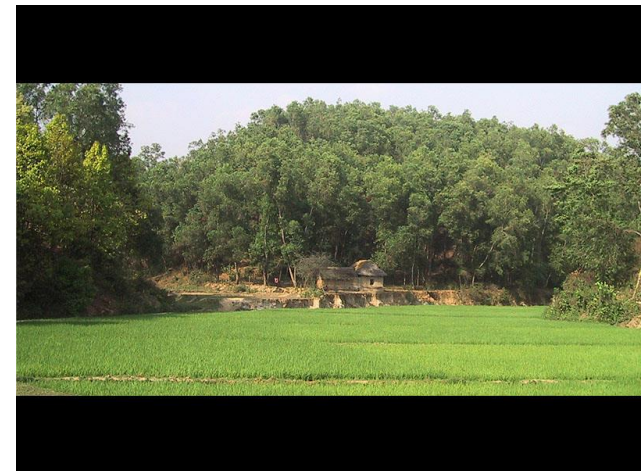
2:1

16:9

5) View Angle (1/2)

▸ Determines amount of perspective distortion in picture, from none (parallel projection) to a lot (wide-angle lens)

▸ In a frustum, two viewing angles: width and height angles

▸ Usually width angle is specified using the height angle and aspect ratio

▸ Choosing *View angle* analogous to photographer choosing a specific type of lens (e.g., a wide-angle or telephoto lens)

The *aspect ratio* is defined by width/height or width angle/height angle

Aspect Ratio $= \dfrac{w}{h}$
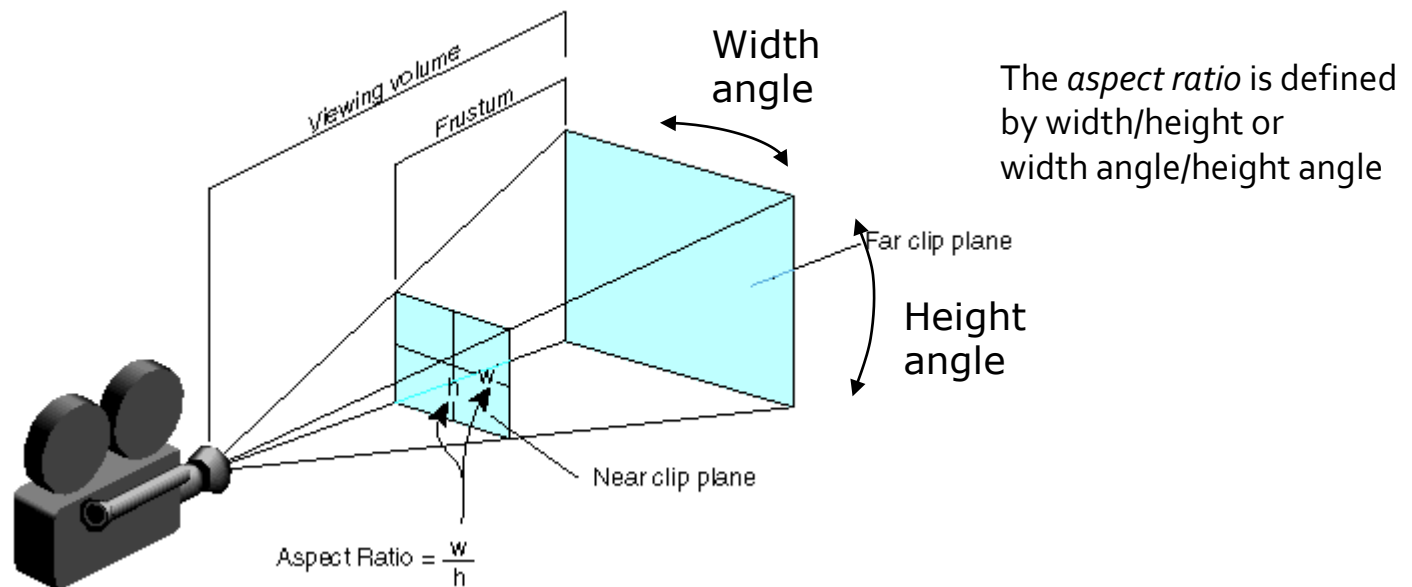
5) Viewing Angle (2/2)

▸ Lenses made for distance shots often have a nearly parallel viewing angle and cause little perspective distortion, though they foreshorten depth

▸ Wide-angle lenses cause a lot of perspective distortion

Resulting picture
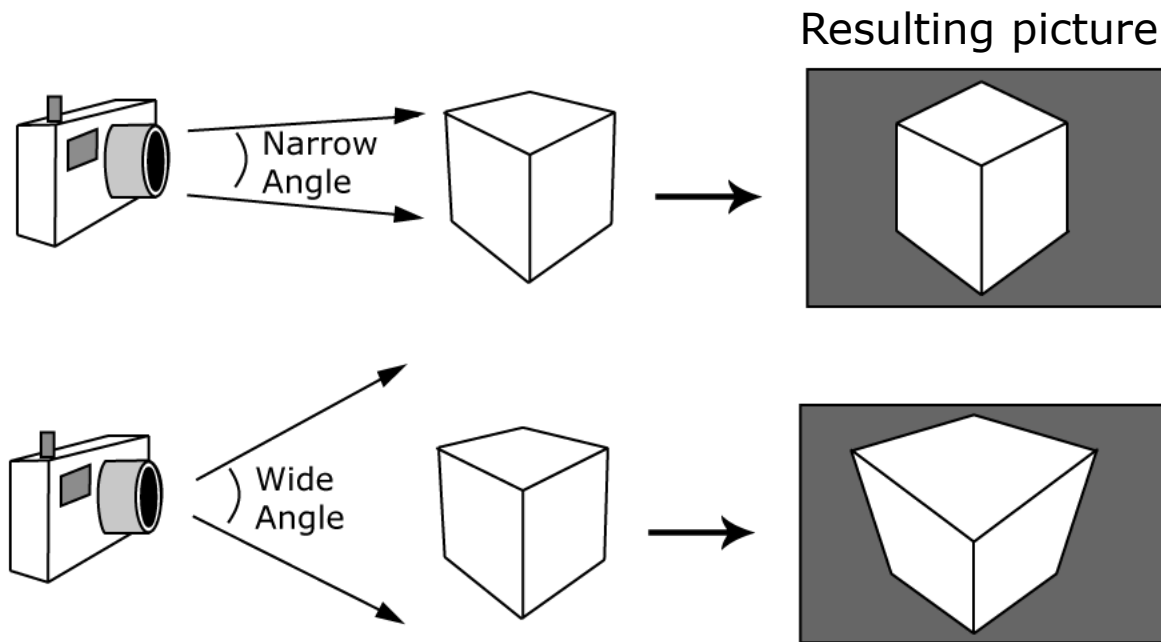
6) Near and Far Clipping Planes (1/2)

▸ With what we have so far we can define four rays extending to infinity defining the edges of our current view volume

▸ Now we need to bound front and back to make a finite volume -- can do this using the *near* and *far* **clipping planes**, defined by distances along the look vector (Also note that our look vector and clipping planes are **perpendicular**)
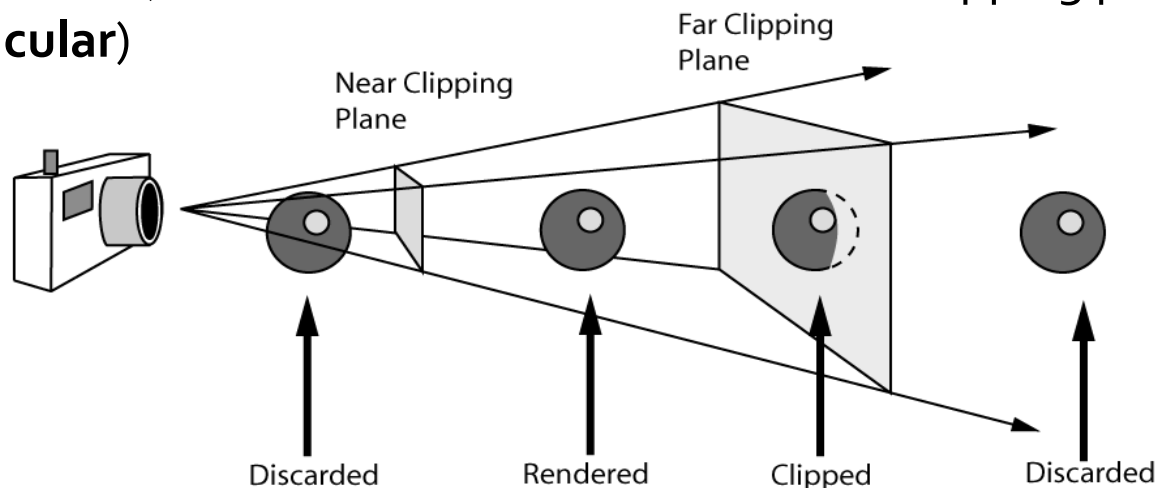
▸ This volume defines what we can see in the scene

▸ Objects outside are **discarded**

▸ Objects intersecting faces of the volume are "**clipped**"

6) Near and Far Clipping Planes (2/2)

▸ Reasons for *Front* (near) *clipping plane*:
  ▸ Usually don't want to draw things too close to camera
    ▸ would block view of rest of scene
    ▸ objects would be distorted
  ▸ Don't want to draw things behind camera
    ▸ wouldn't expect to see things behind camera
    ▸ in the case of perspective camera, if we were to draw things behind camera, they would appear upside-down and inside-out because of perspective transformation
▸ Reasons for *Back* (far) *clipping plane*:
  ▸ Don't want to draw objects too far away from camera
    ▸ distant objects may appear too small to be visually significant, but still take long time to render
    ▸ by discarding them we lose a small amount of detail but reclaim a lot of rendering time
    ▸ can also help to declutter a scene
▸ These planes need to be properly placed, not too close to the camera, not too far (mathematical justification next time)

Games and Planes (1/2)

▸ **Problem:** Sometimes in a game you can position the camera in the right spot that the front of an object gets clipped, letting you see inside of it.

▸ Video games use various techniques to avoid this glitch. One technique is to have **objects** that are very close to the **near clip plane fade out before** they get **cut off**, as can be seen below

Screenshots from the game, *Okami*

▸ This technique gives a clean look while solving the near clipping problem (the wooden fence fades out as the camera follows the running wolf).

Games and Planes (2/2)

▸ **Problem:** Ever played a video game and all of a sudden some **object pops up in the background** (e.g., a tree in a racing game)? That's an object coming inside far clip plane.

▸ The **old hack** to keep you from noticing the pop-up is to **add fog** in the distance. A classic example of this is from *Turok: Dinosaur Hunter (left)*

▸ Now all you notice is fog. This practically defeats the purpose of an outdoor environment! And you can *still* see pop-up from time to time.

▸ The **modern solution is dynamic level of detail**, mesh detail increases as you get closer to it, e.g. *Stone Giant (right)*

▸ Thanks to fast hardware and level of detail algorithms, we can push the far plane back now and fog is much less prevalent

Focal Length

▸ Some camera models take a *Focal length*

▸ *Focal Length* is a measure of ideal focusing range; approximates behavior of real camera lens

▸ Objects at distance equal to *Focal length* from camera are rendered in focus; objects closer or farther away than *Focal length* get blurred

▸ *Focal length* used in conjunction with clipping planes

▸ Only objects within view volume are rendered, whether blurred or not. Objects outside of view volume still get discarded


© davebutler3d@hotmail.com

The Parallel View Volume (1/2)

▸ Up until now we've been describing the specifications for a perspective view volume

▸ We also need to discuss the parallel view volume (used for orthogonal/metric views)

▸ What do we need to know this time?

▸ Everything we wanted for a perspective view volume except for width and height angles, replaced by just a width and height (also the width and height of our film on our film plane)

▸ A parallel view volume is a parallelepiped (all opposite edges parallel)
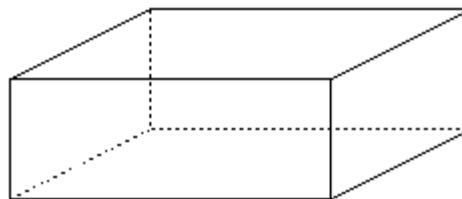
Rectangular Parallelpiped

The Parallel View Volume (2/2)
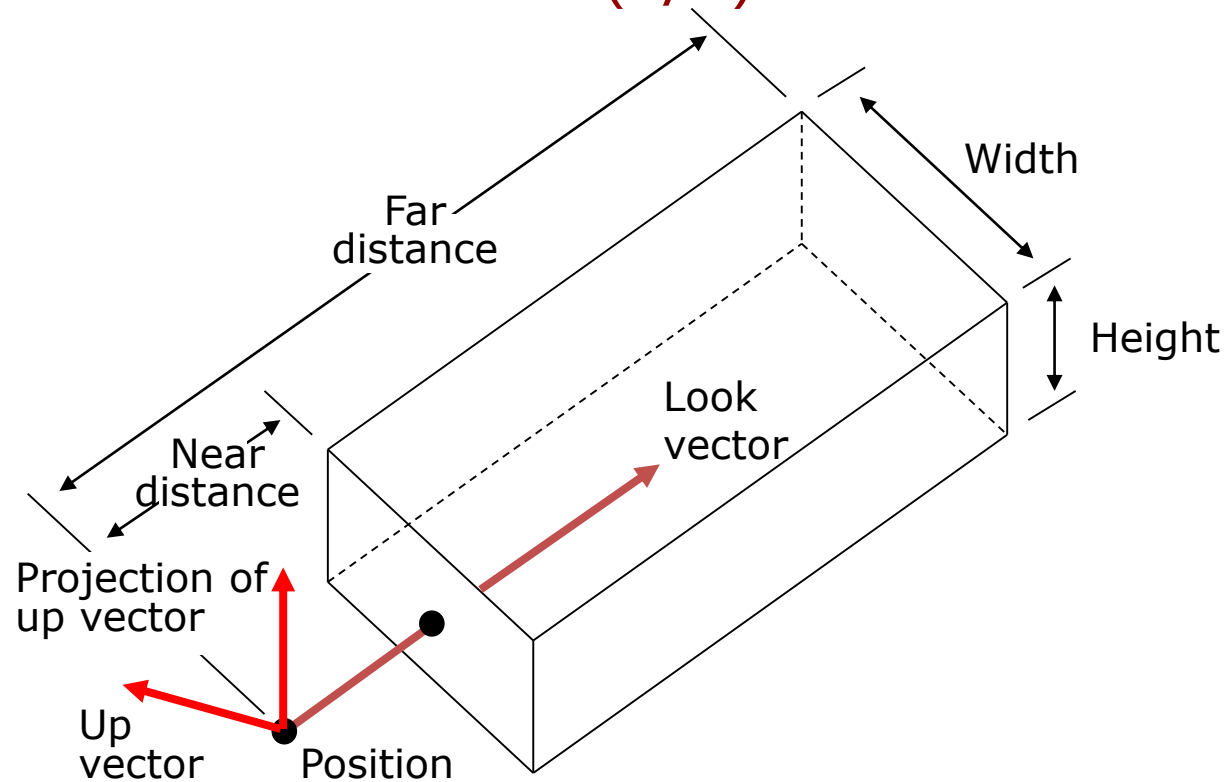
▸ Objects appear the same size no matter how far away they are since projectors are all parallel

▸ A benefit of the parallel view volume is that it's really easy to project a 3D scene to a 2D medium

http://www.cs.brown.edu/exploratories/freeSoftware/repository/edu/brown/cs/exploratories/applets/camera/parallel_camera_transformation_guide.html

Παράδειγμα

▸ Έστω ότι έχουμε ένα αντικείμενο όπως αυτό που βλέπουμε στο πιο κάτω δεξιά σχεδιάγραμμα, με βάση 2x3 μονάδες και ύψος 2 μονάδες. Αν υποθέσουμε ότι χρησιμοποιούμε προοπτική προβολή, ποιες πρέπει να είναι οι τιμές των παραμέτρων της κάμερας για να έχουμε την παρακάτω εικόνα;

εικόνα με 480x480 pixels