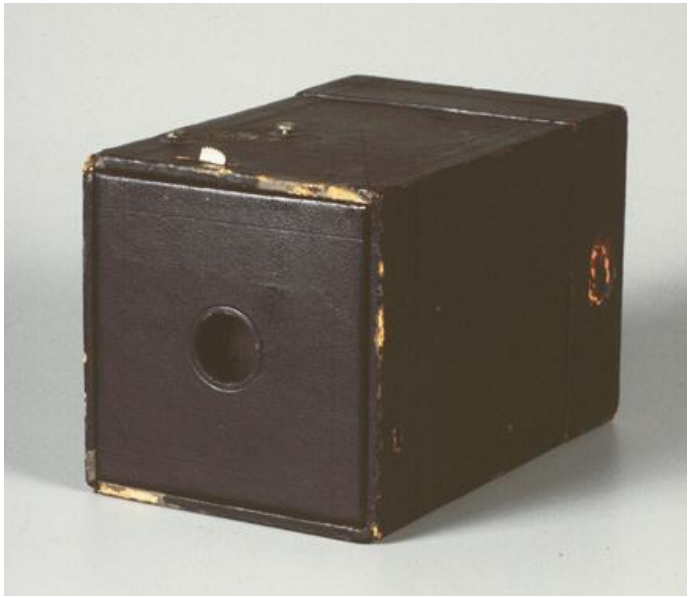


Viewing

Part II (The Synthetic Camera)

The camera and the scene

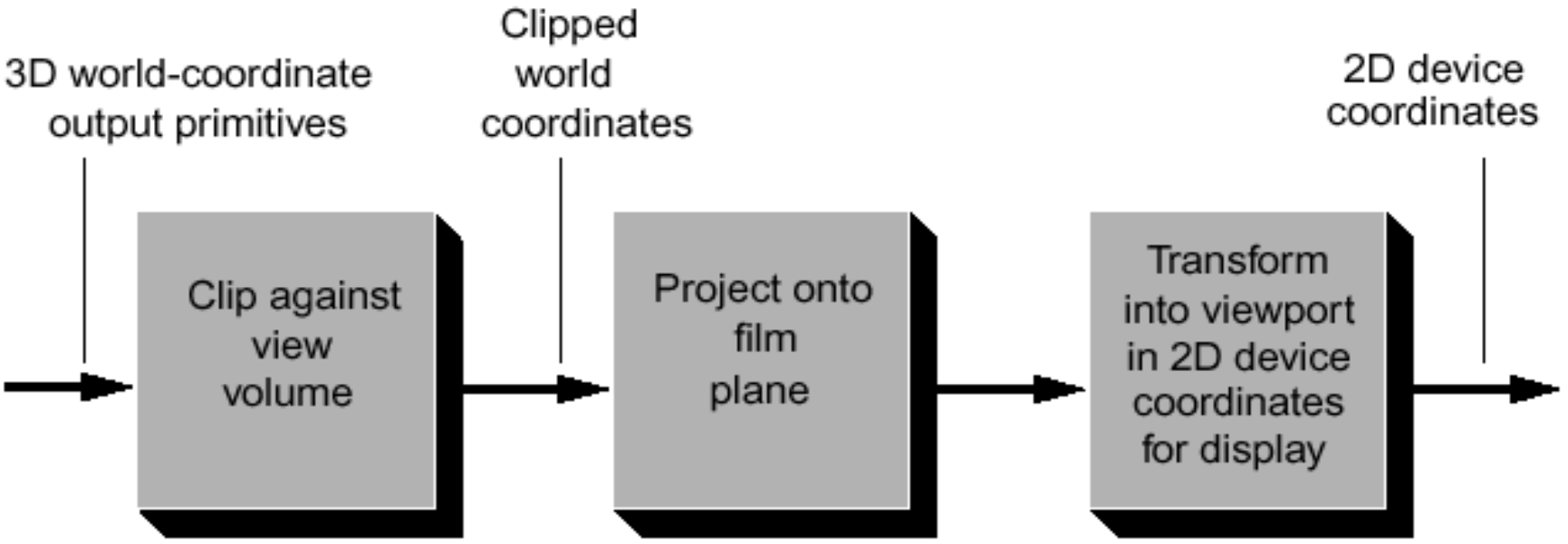
- What does a camera do?
  - Takes in a 3D scene
  - Places (i.e., projects) the scene onto a 2D medium (a roll of film or a digital pixel array)

- 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/Focal distance (near distance, far 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, resp.)
- CS123 uses a simpler slightly less powerful model than the one used in the book
  - Omit tilt of view/film plane, focal distance (blurring)

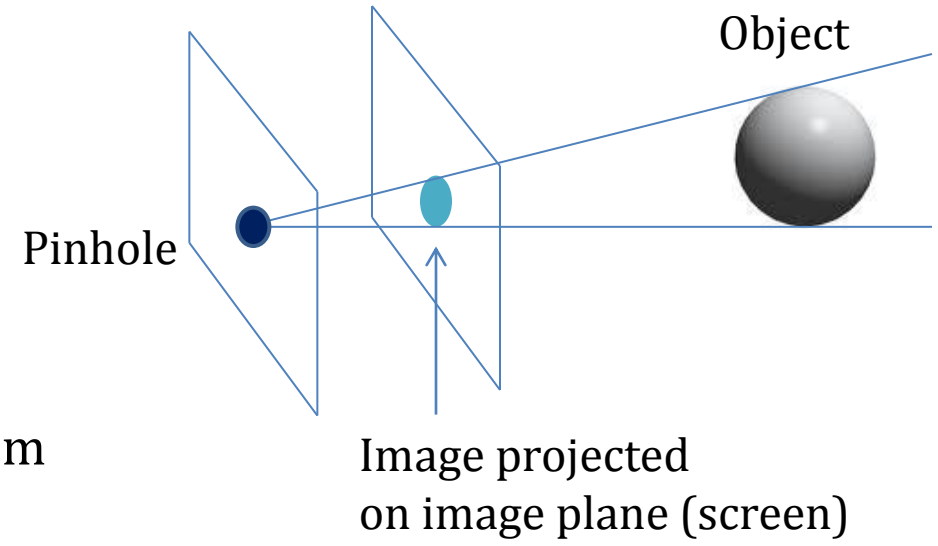
Cameras in Rendering Process



- Will detail coordinate systems for camera, i.e., view volume specification, projecting onto film plane, and transforming into viewport
- Let's first define viewport and view volume

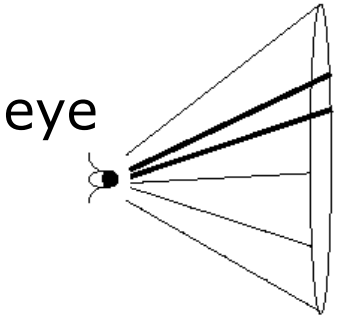
The pinhole model

- You can think of camera as a pinhole. As you look through pinhole see a certain volume of space
- Rays of light reflect off objects and converge to pinhole to let you see the scene
- The pinhole is where our camera position will be ("center of projection") and 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 projection is mapped to some form of viewing medium (screen)



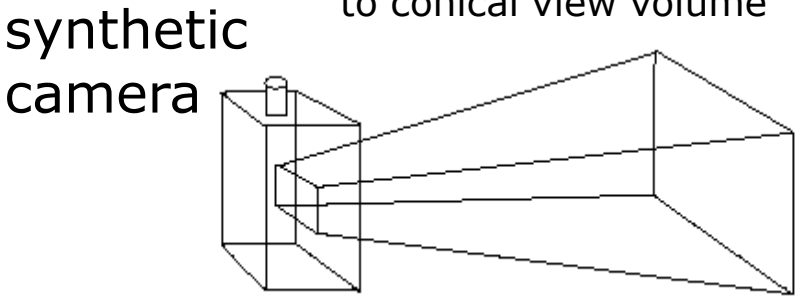
View Volumes (focus of today's lecture)

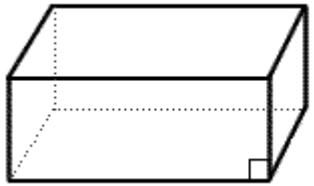
- A view volume contains everything the camera sees
- Conical – Approximates what eyes see, expensive math when clipping objects against cone's surface (simultaneous linear and quadratics)
- Can approximate this using a rectangular frustum view volume
  - Simultaneous linear equations for easy clipping of objects against sides (stay tuned for clipping lecture next time)
- Also parallel view volumes, e.g., for orthographic projections. These don't simulate eye or camera



conical perspective view volume (eye's is much wider, e.g., >= 180 degrees, esp. for motion!)

Frustum: approximation to conical view volume





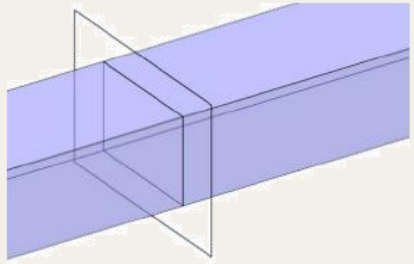
View volume

(Parallel view)

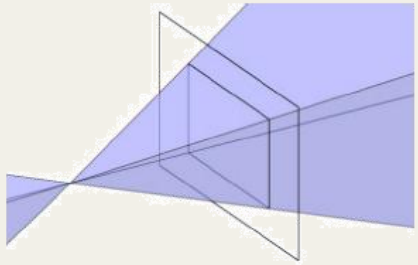
View Volumes and Projectors

- Given our view volume 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)

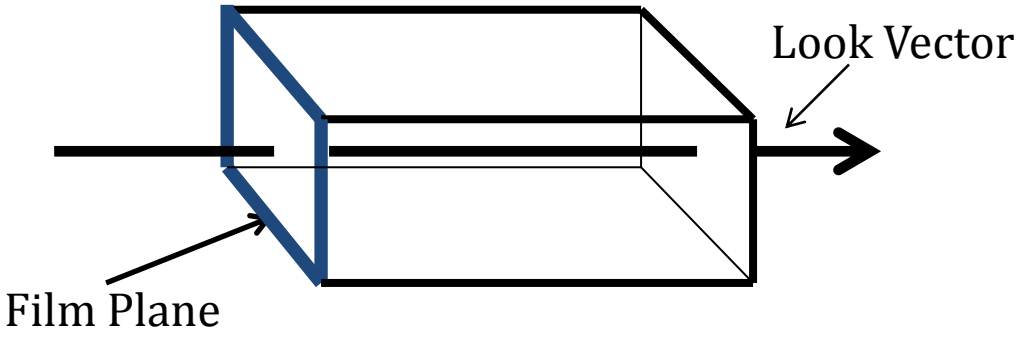


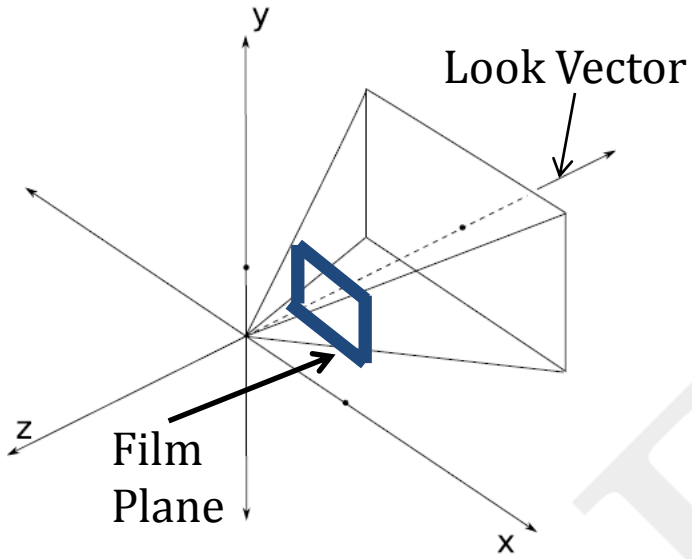
- **Perspective Volumes**: Projectors converge on eye point = center of projection, like rays of light converging to your eye



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

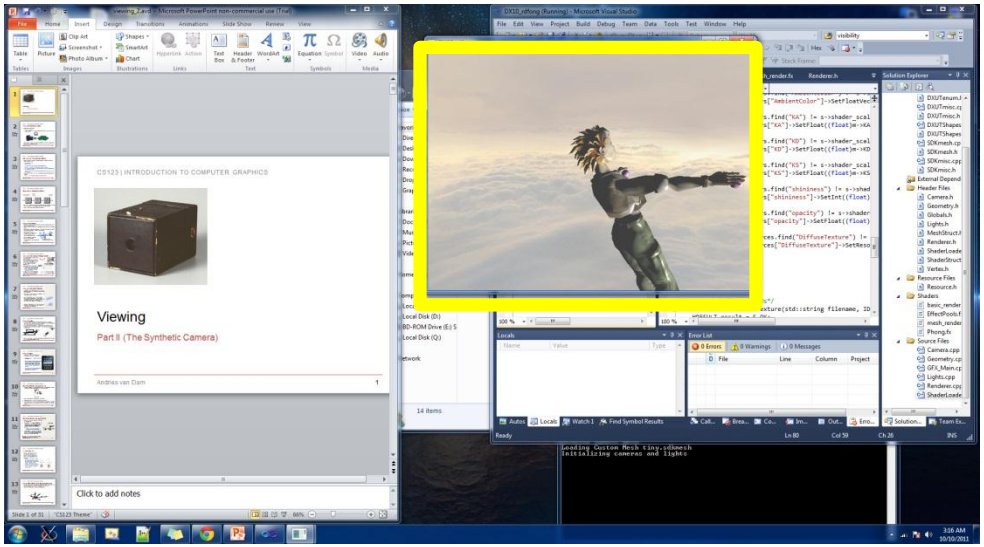




- Actual location of film plane along look vector doesn't matter as long as it is between eye/COP and scene

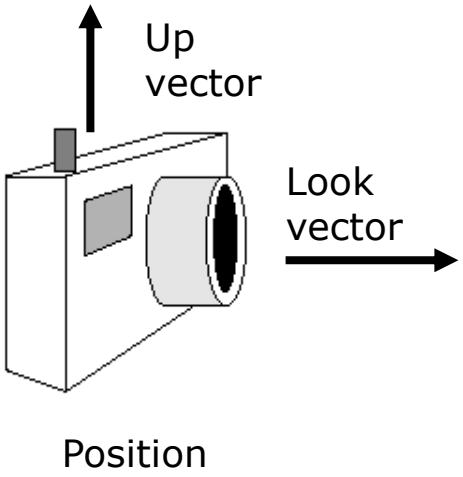
The viewport

- 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)* in computer graphics used to mean 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)

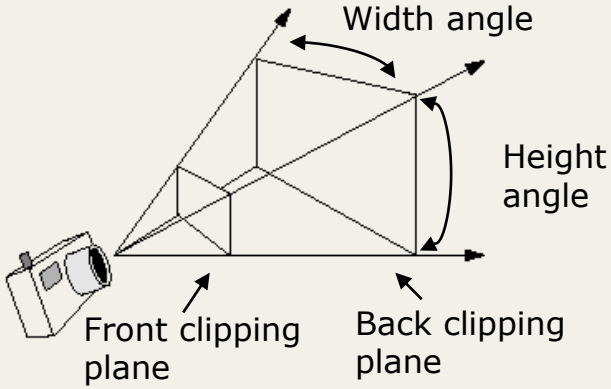
- We need to know six things about our synthetic camera model in order to take a picture using our perspective view **frustum**



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)

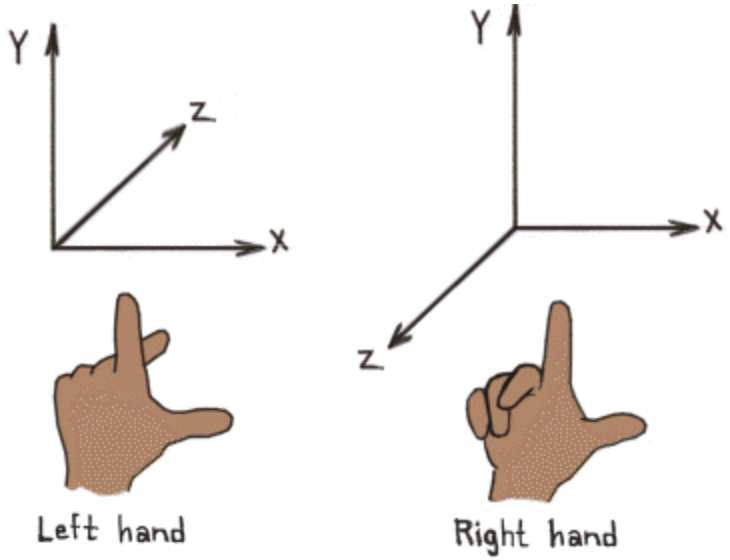
4) *Aspect ratio* of the electronic "film:" ratio of width to height
5) *Height angle:* determines how much of the scene we will fit into our view volume; larger height angles fit more of the scene into the view volume (width angle determined by height angle and aspect ratio)
   - the greater the angle, 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

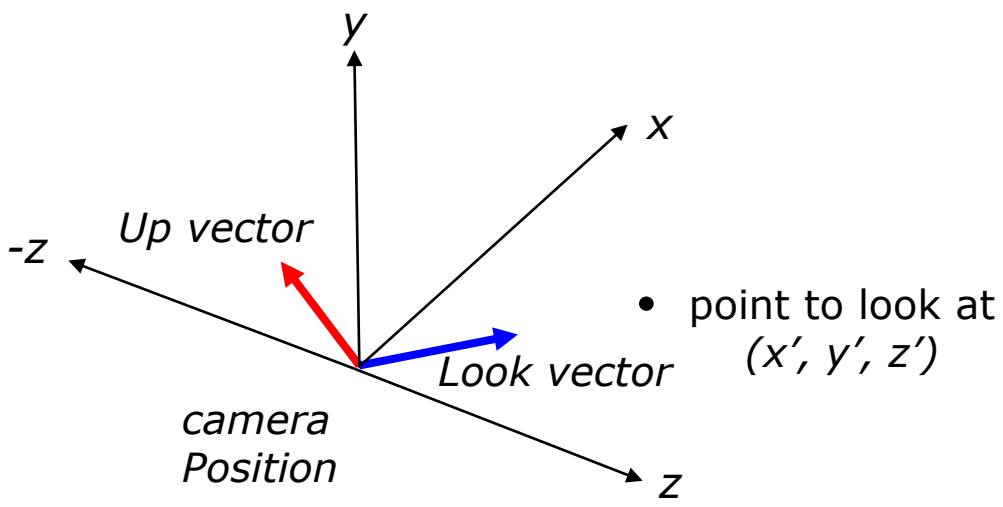
1) Position (1/1)

- 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



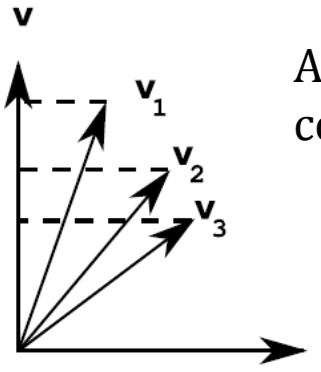
2 & 3) Orientation: Look and Up vectors (1/2)

- 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 vector

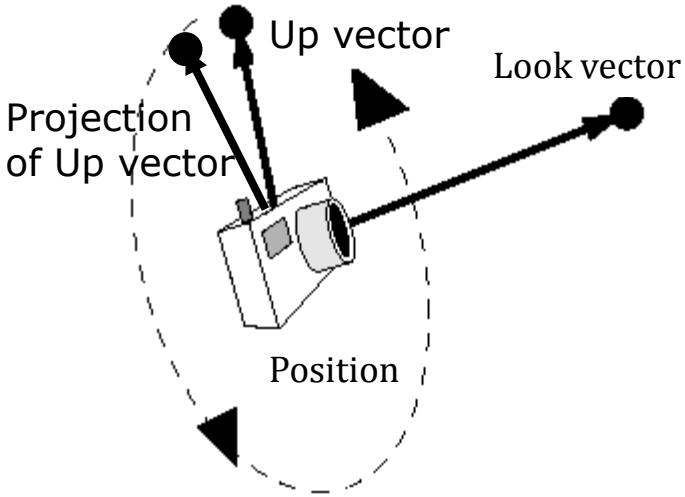


2 & 3) Orientation: Look and Up vectors (2/2)

- *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

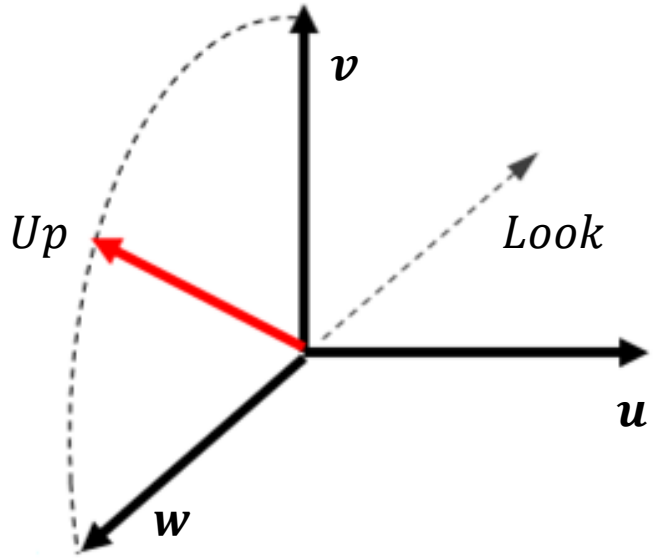


Any of $v, v1, v2, v3$ could be the up vector



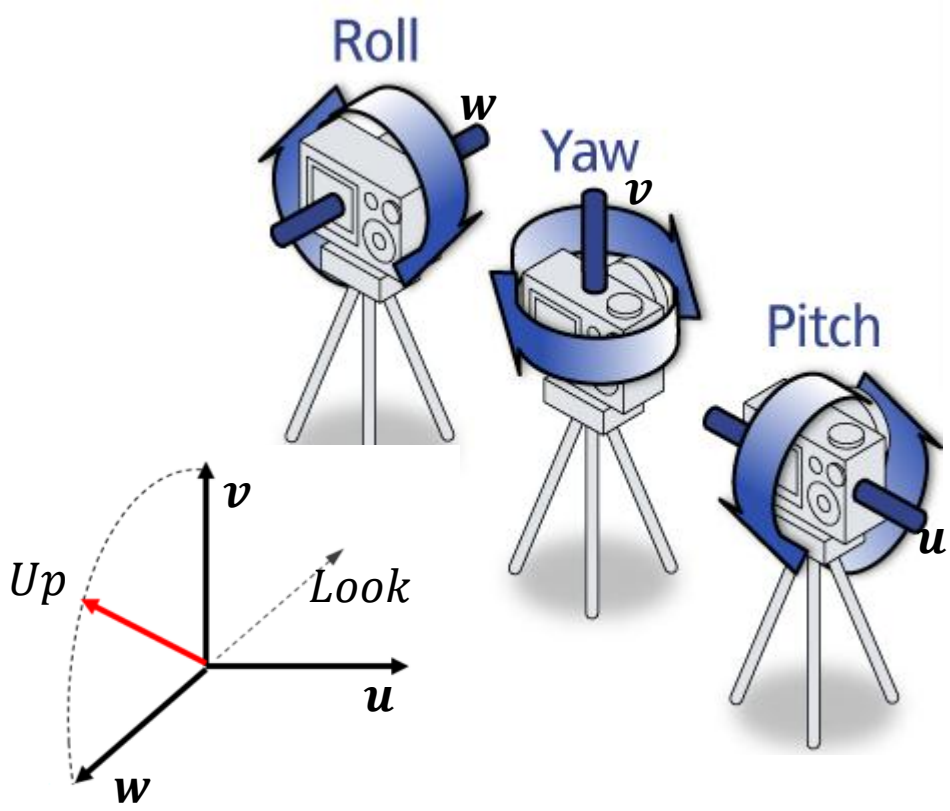
The camera coordinate space (1/2)

- The equivalent of ***x, y*** and ***z*** axes in camera space are unit vectors ***u, v*** and ***w*** (not to be confused with 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 $\boldsymbol{w}$
- Yaw:
  - Rotating your camera around $\boldsymbol{v}$
- Pitch:
  - Rotating your camera around $\boldsymbol{u}$

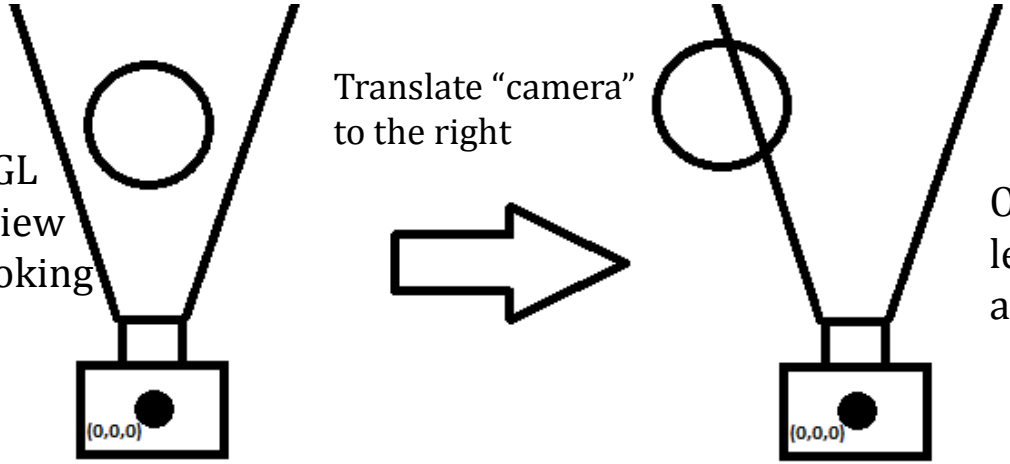


- Send camera to origin and align axes with the standard axes, then use our rotation matrices to perform these transformations and then un-align and un-translate.

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 inverse transformation to objects in scene
- This is the model OpenGL uses; note however that GLU abstracts this concept away from the programmer (gluLookAt())

Field of view in OpenGL can be thought of as view of camera at origin looking down –z axis

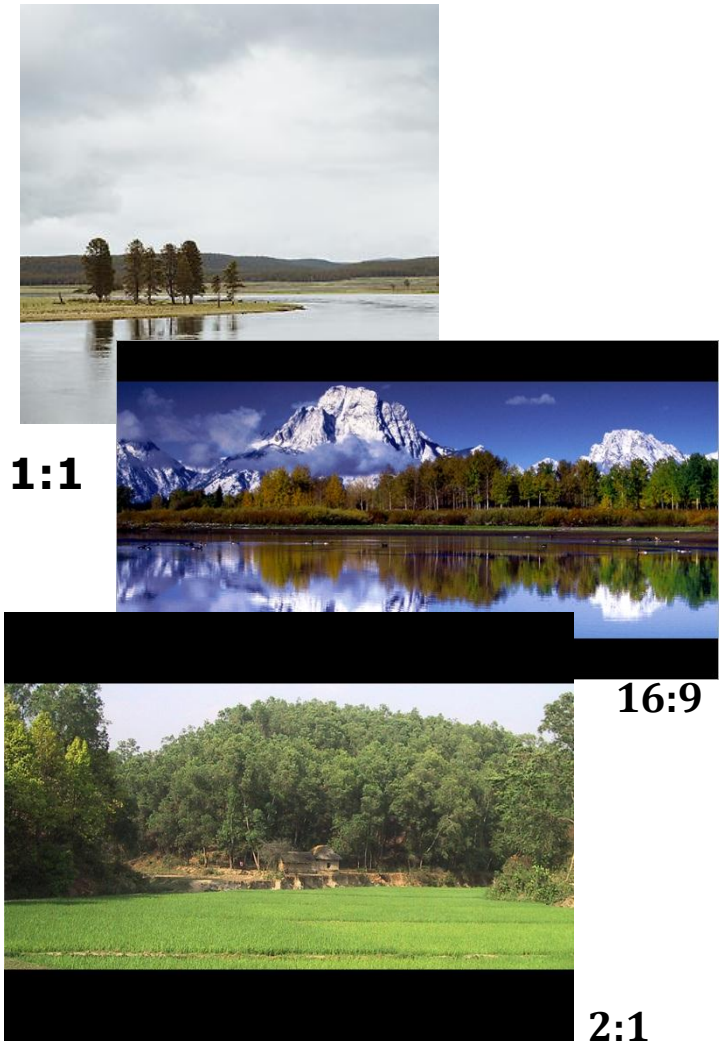


Translate "camera" to the right

Object moves to left to simulate a camera moving to right

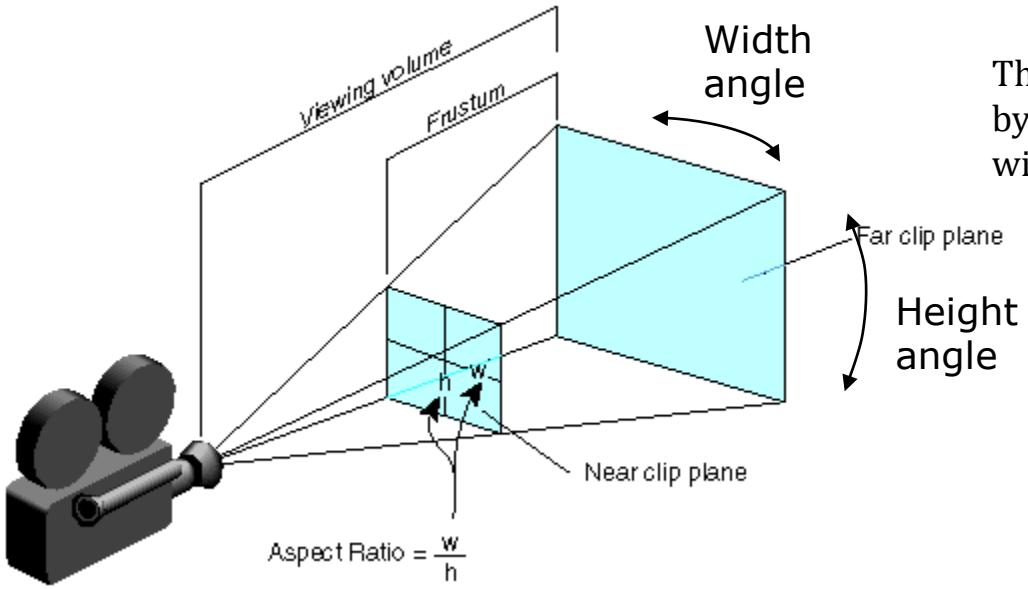
4) Aspect Ratio (1/1)

- 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 same aspect ratio for both your viewing window and viewport, to avoid distortions/ stretching



5) View Angle (1/2)

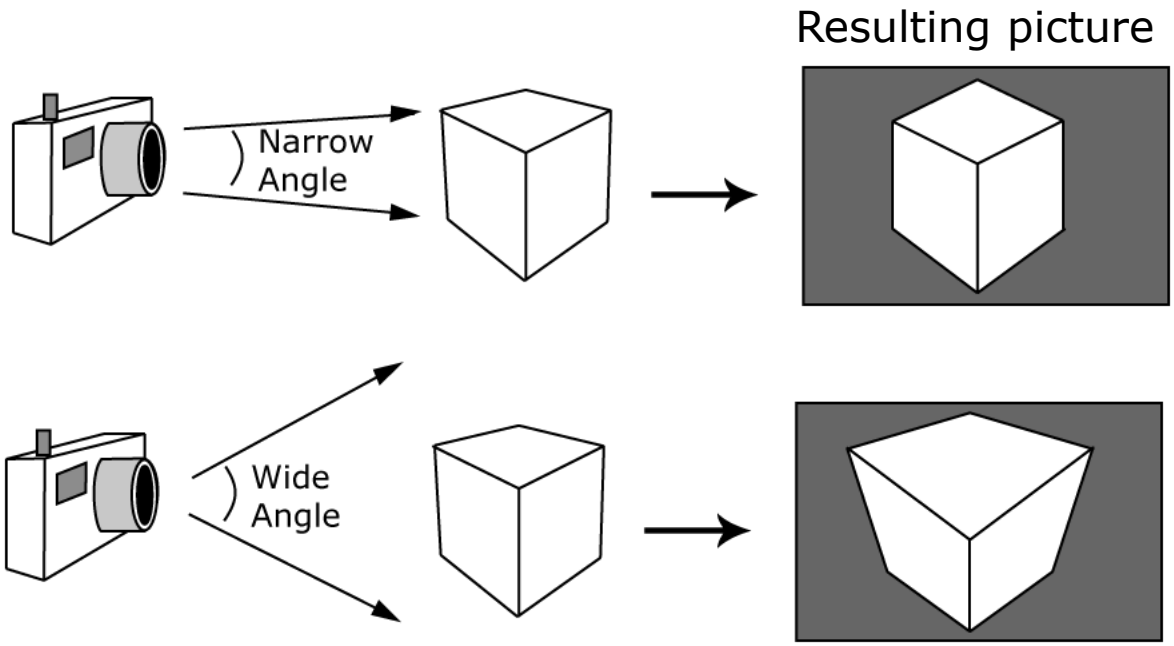
- Determines amount of perspective distortion in picture, from none (parallel projection) to a lot (wide-angle lens)
- In a frustum, two viewing angles: width and height angles
  - Usually width angle is specified using the height angle and aspect ratio
- Choosing *View angle* analogous to photographer 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

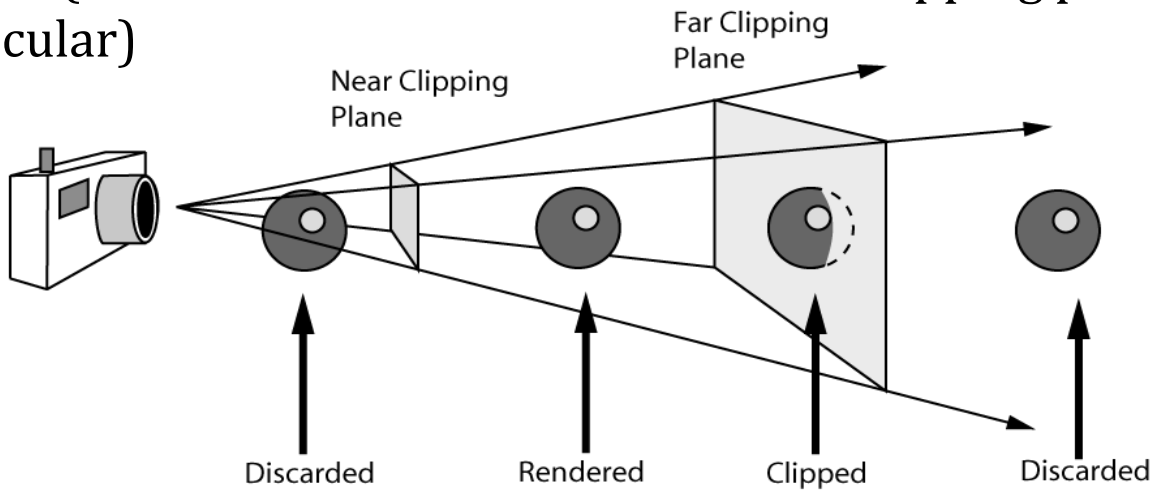
5) Viewing Angle (2/2)

- 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



6) Near and Far Clipping Planes (1/3)

- 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 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 (2/3)

- 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

6) Near and Far Clipping Planes (3/3)

- 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 later)

Games and Planes (1/2)

- 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

Fence is opaque

Fence is partially transparent

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 gets too close to it, allowing you to see the wolf behind it).

Games and Planes (2/2)

- 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 object coming inside far clip plane.
- Old solution, add fog in the distance. A classic example, *Turok: Dinosaur Hunter*
- Modern solution, dynamic level of detail: mesh detail increases as you get closer

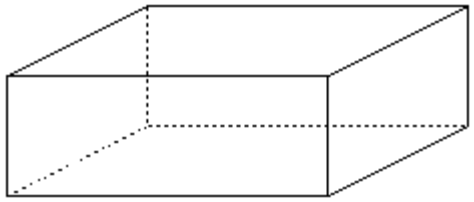
- Thanks to fast hardware and level of detail algorithms, we can push the far plane back now and fog is much less prevalent

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

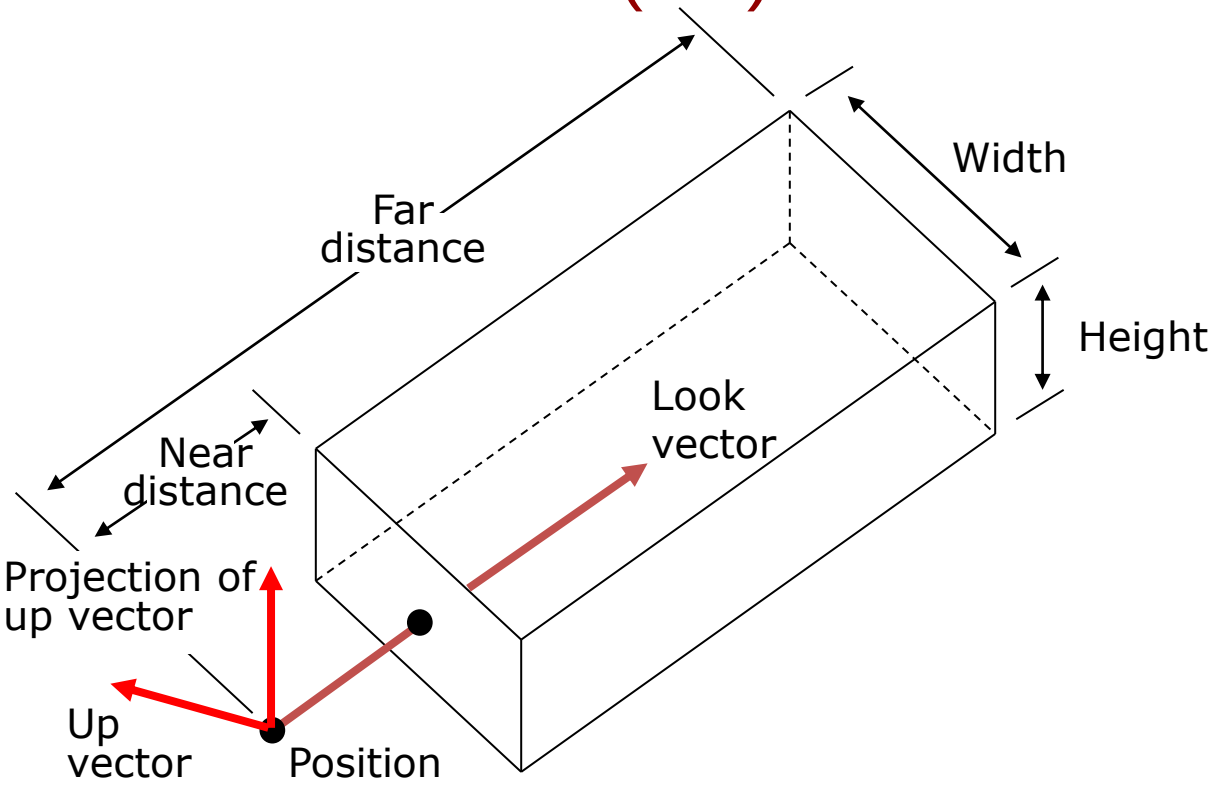
The Parallel View Volume (1/2)

- Up until now we've been describing the specifications for a perspective view volume
- We also need to discuss the parallel view volume (as mentioned last time, 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

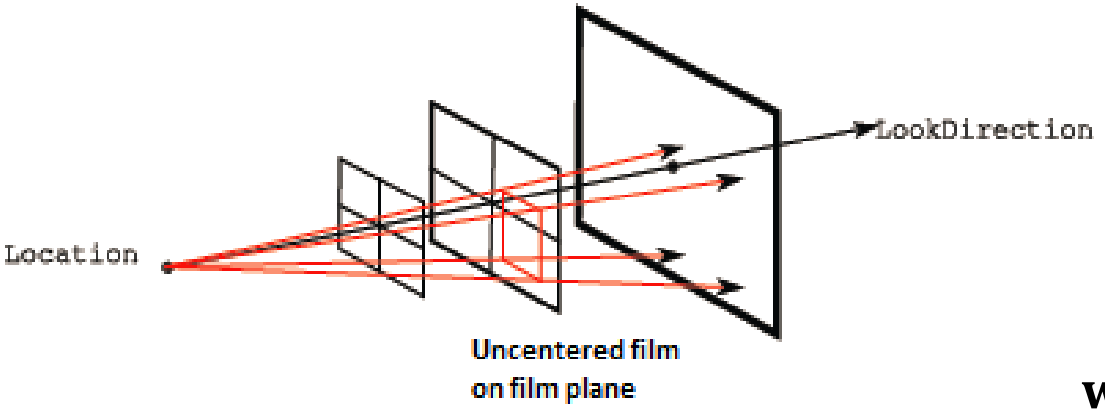
The Parallel View Volume (2/2)

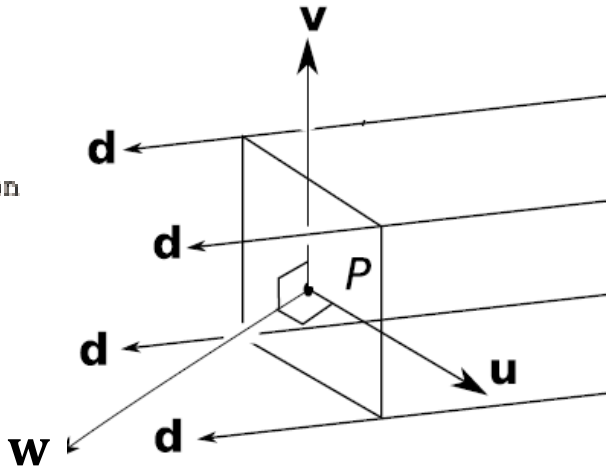


- Objects appear the same size no matter how far away they are since projectors are all parallel
- A benefit of the parallel view volume is that it's really easy to project a 3D scene to a 2D medium

Capabilities of the Generalized Camera (1/2)

- In a more generalized camera the viewing window doesn't have to be centered about the *Location+LookDirection*
- Nor does it have to be perpendicular to the LookDirection





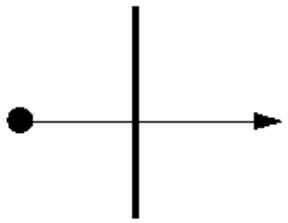
- This allows us to use a more flexible view as well as enable the use of more view types
- Using an uncentered film we can essentially choose which part of our original perspective projection to view

Capabilities of the Generalized Camera (2/2)

- Using a parallel view volume we can do oblique projections (cavalier, cabinet, perspective oblique) where the look vector/projectors and film plane aren't perpendicular:

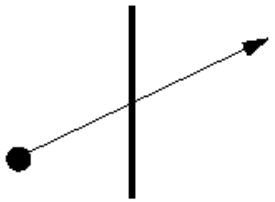
Non-oblique perspective view volume:

Look vector is perpendicular to film plane



Oblique perspective view volume:

Look vector is at an angle to the film plane



- Our model of a camera is not all encompassing
- There are some capabilities that we have omitted for the sake of simplicity
- Our film is centered around the camera position and always perpendicular to the look vector

Next time…

- We have now seen how to construct a perspective and parallel view volume and we mentioned how a scene is projected in these volumes onto the film plane
- But these view volumes can be located anywhere and positioned in any way depending on how our camera is specified
- How can we transition from knowing what the view volume looks like to actually rendering an image to the screen
- Next, describe canonical view volume and how we can use it for rendering

- PS: Get ready for some math!