Computer Graphics Lecture 05: Camera and Viewing

DR. ELVIS S. LIU

SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

SPRING 2018

Image Formation (3D on a 2D Display)

- We have talked about 3D points and how they may be manipulated in space using matrices
- Those 3D points might be the vertices of a cube, which we might rotate using our Euler angle rotation matrices to create an animation of a spinning cube
- They might form more sophisticated objects still, and be acted upon by complex compound matrix transformations
- Regardless of this, for any modelled graphic we must ultimately create a 2D image of the object in order to display it

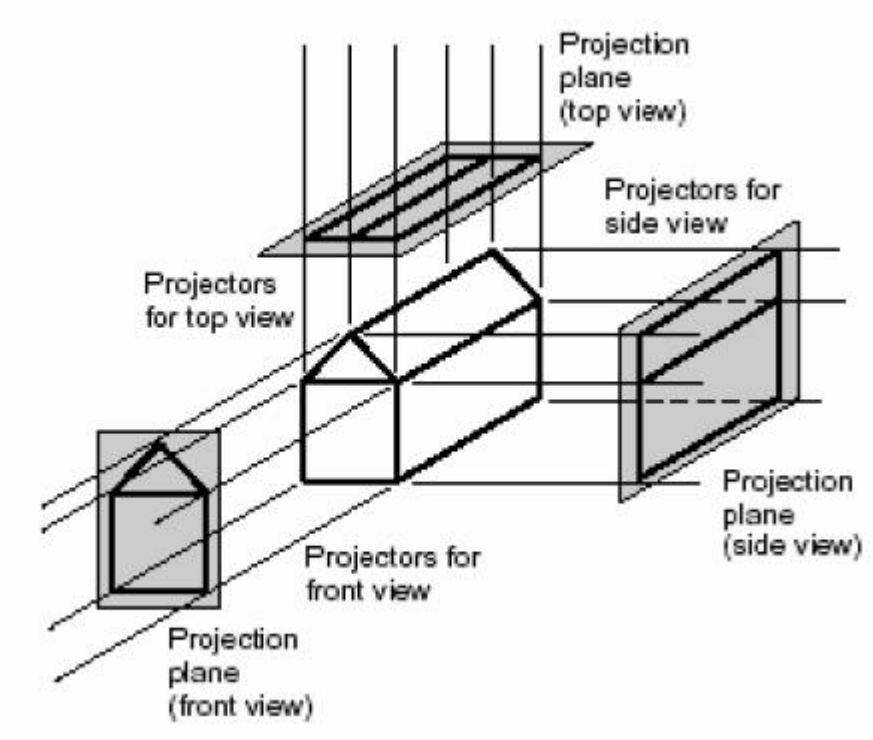
Projection

LECTURE 05: CAMERA AND VIEWING

Turning 3D to 2D

- Moving from a higher dimension (3D) to a lower dimension (2D) is achieved via a projection operation
- a lossy operation that too can be expressed in our 4 ×−4 matrix framework acting upon homogeneous 3D points
- Common types of projection are perspective projection and orthographic projection

Orthographic Projection

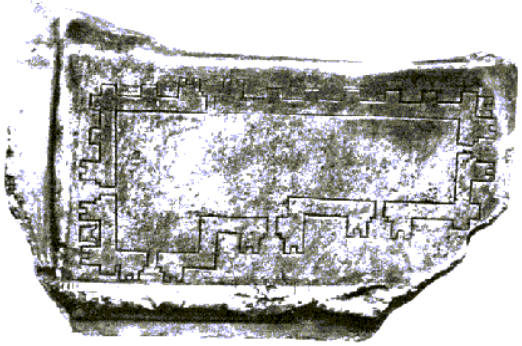


Drawing as Projection

- Painting based on mythical tale as told by Pliny the Elder: Corinthian man traces shadow of departing lover – a projection
- Detail from **The Invention of Drawing (1830)** – Karl Friedrich Schinkle

Early Forms of Projection

- Plan view (parallel, specifically orthographic, projection) from Mesopotamia (2150 BC): Earliest known technical drawing in existence
- Greek vase from the late 6th century BC: Shows signs of attempts at perspective foreshortening!
  - Note relative sizes of thighs and lower legs of minotaur

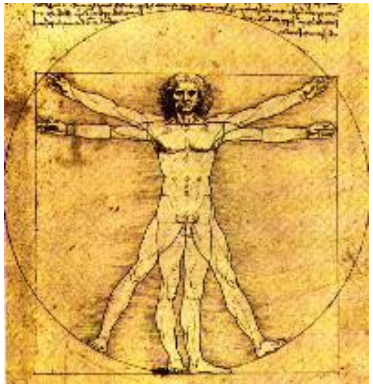


Early Forms of Projection (cont.)

- Ancient Egyptian Art
  - Multiple Viewpoints
  - Parallel Projection (no attempt to depict perspective foreshortening)
- Note how depiction of body implies a front view but the feet and head imply side view (early cubism!)

The Renaissance Changed Many Things

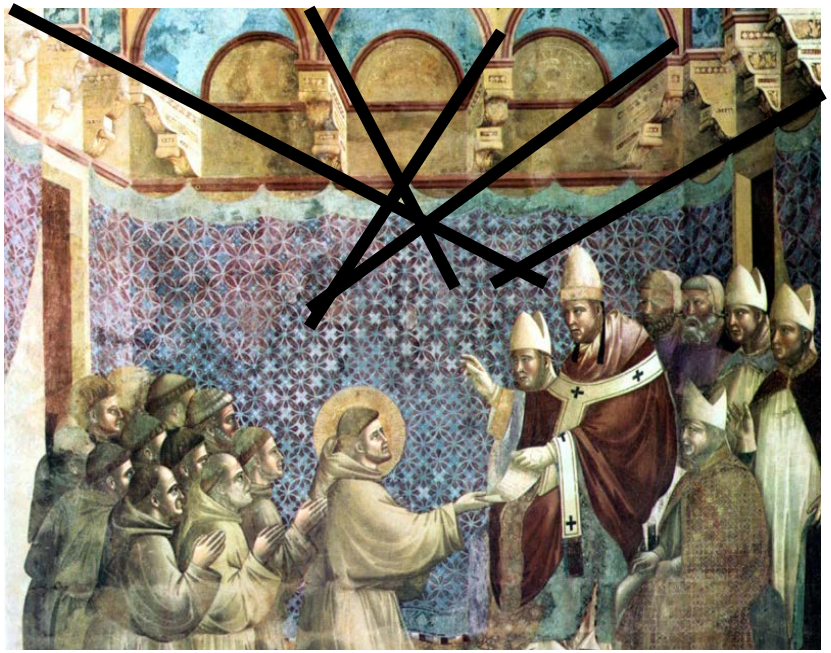
- Starting in the 13th century (AD): New emphasis on importance of individual viewpoint, world interpretation, power of observation (particularly of nature: astronomy, anatomy, etc.)
  - Masaccio, Donatello, DaVinci
- Universe as clockwork: rebuilding the universe more systemically and mechanically
  - Tycho Brahe and Rudolph II in Prague (detail of clockwork), c. 1855
  - Copernicus, Kepler, Galileo, Newton…: from earth-centric to heliocentric model of the (mechanistic) universe whose laws can be discovered and understood



Vitruvian Man, 1490, study of man's proportions
Image credit: wikipedia

Early Attempts at Perspective

- Attempts to represent 3D space more realistically
- Earlier works invoke a sense of 3D space but not systematically
  - Parallel lines converge, but no single vanishing point (where all parallel lines converge)



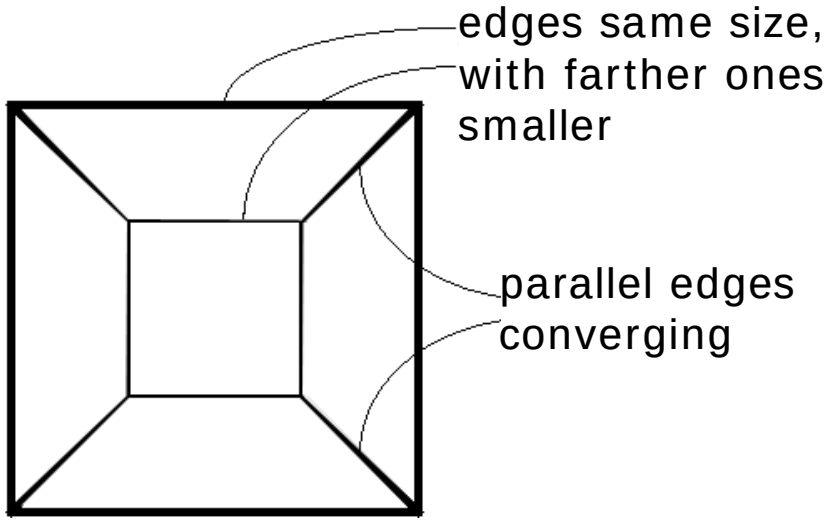
Perspective Viewing

Forced Perspective Art

- http://www.youtube.com/watch?v=uzNVo8NbpPI

Rules of Linear Perspective

- Driving ideas behind linear perspective:
  - Parallel lines converge (in 1, 2, or 3 axes) to vanishing point
  - Objects farther away are more foreshortened (i.e., smaller) than closer ones
- Example: perspective cube

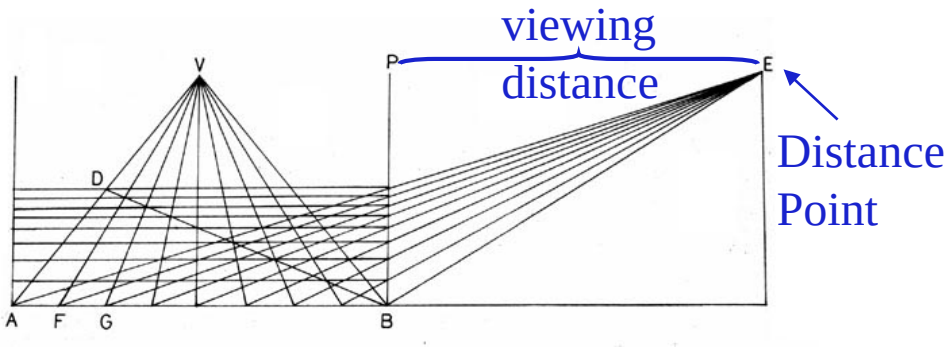


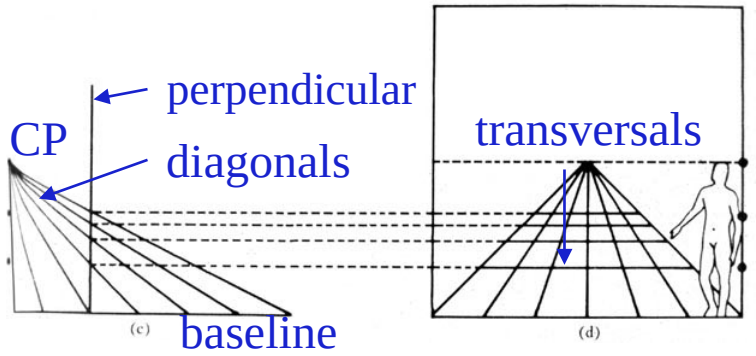
Vanishing Point

- Objects in the distance appear smaller that those close-by in perspective projection
- One consequence is that parallel lines in the real world do not map to parallel lines in an image under perspective projection
- Consider parallel train tracks (tracks separated by a constant distance) running into the distance
  - That distance of separation appears to shrink the further away the tracks are from the viewer
  - Eventually the tracks appear to converge at a vanishing point some way in to the distance.

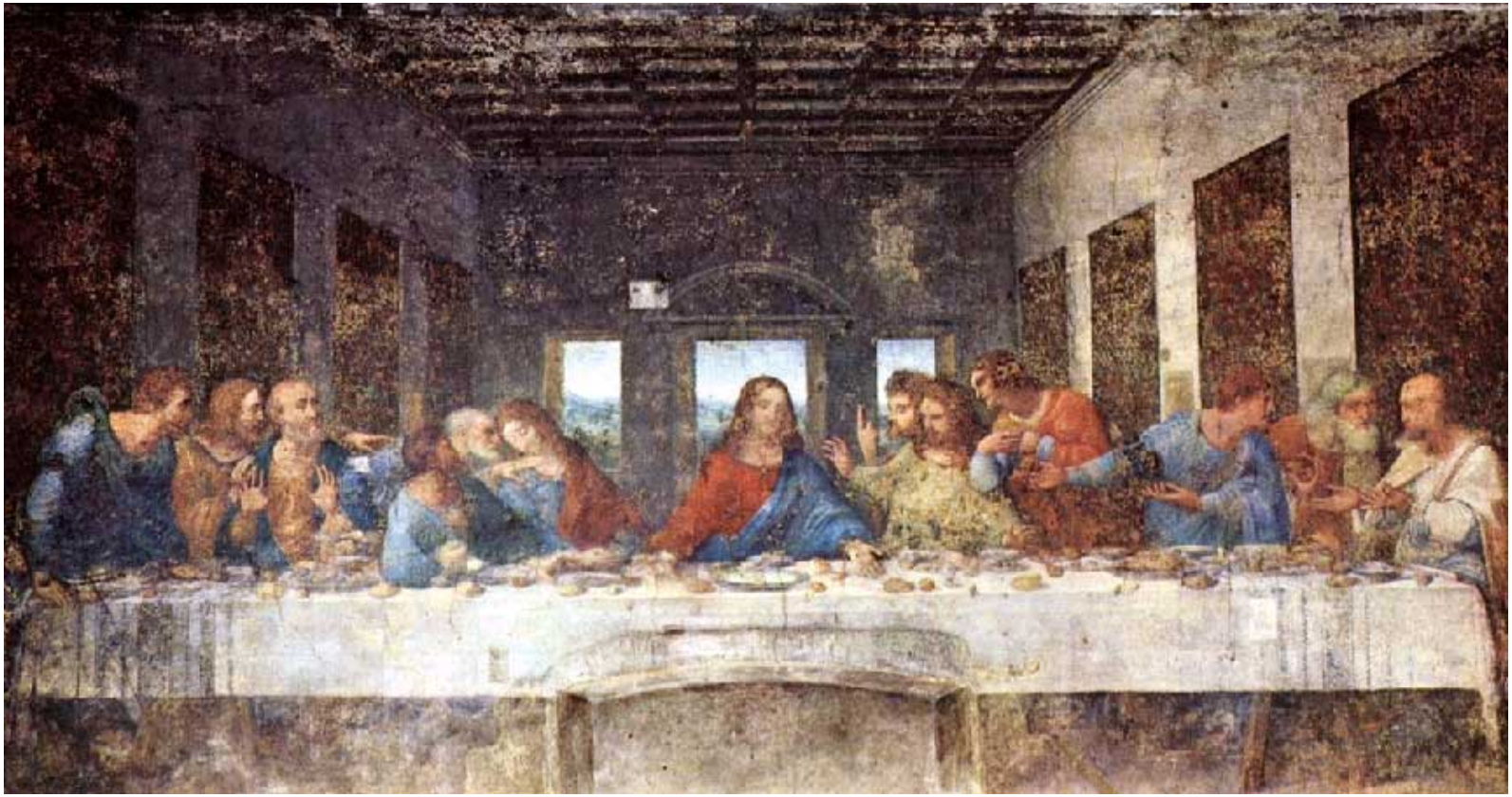
Vanishing Points in Renaissance Paintings

- Both Da Vinci and Alberti created accurate geometric ways of incorporating linear perspective into a drawing using the concept vanishing points



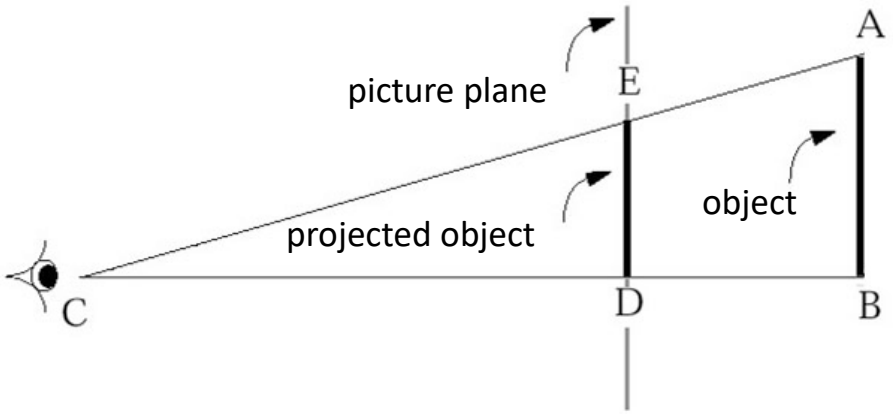


Leonardo's Vanishing Point



Triangles and Geometry

- Perspective projection of a 3D object to create a 3D image
- Light rays travel in straight lines from points on the object AB to a eye/camera focal point C, passing through a planar surface representing the image ED to be rendered.
- Easy to project object onto an image plane based on
  - Height of object |AB|
  - Distance from eye to object |CB|
  - Distance from eye to picture plan |CD|
  - Relationship |CB|/|CD| = |AB|/|ED|

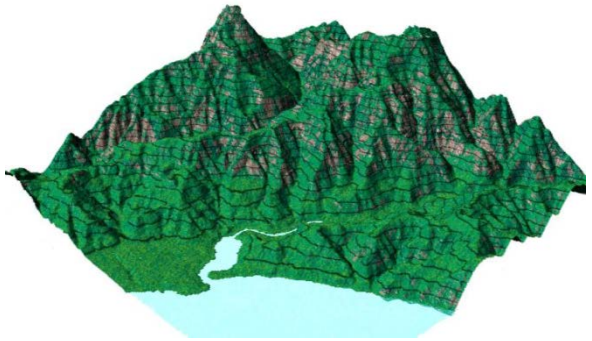


Camera

LECTURE 05: CAMERA AND VIEWING

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 such as a roll of film or a digital pixel array



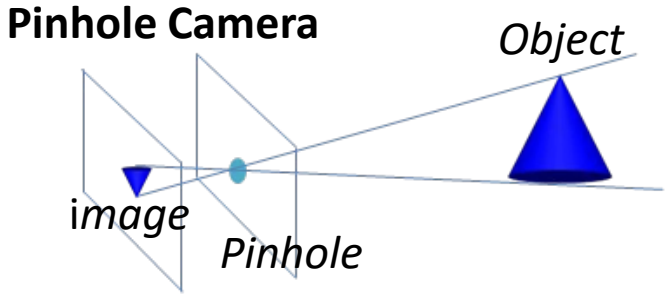
- The synthetic camera is a programmer's model for specifying how a 3D scene is projected onto the screen

3D Viewing: The Synthetic Camera

- General synthetic camera: each package has its own but they are all (nearly) equivalent, with the following parameters/degrees of freedom:
  - Camera position
  - Orientation
  - Field of view (angle of view, e.g., wide, narrow/telephoto, 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 or parallel projection (camera in scene with objects or infinite distance away, resp.)

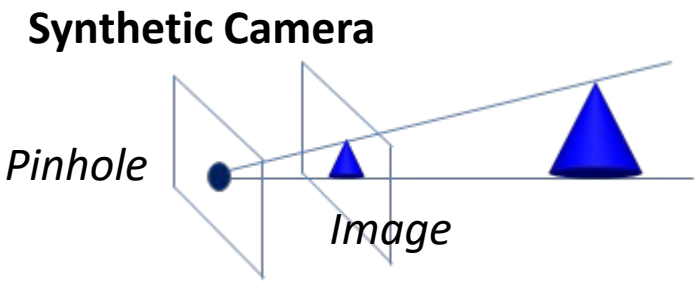
Pinhole Model for Perspective Projection – Geometric Optics

- Pinhole camera optically projects a volume of space (the scene) in front of it
  - Rays of light reflect off objects and those visible from the camera's Point of View (i.e., those (parts of objects) that lie inside its view volume and are visible from the PoV) converge to the pinhole and then project the scene on a film plane or wall behind the pinhole
  - The scene will be inverted.
- Pinhole is our camera position ("center of projection"), and view volume is what we see



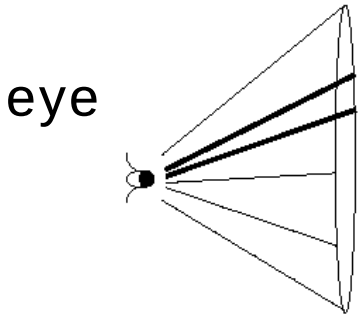
Synthetic Model for Perspective Projection – Geometric Optics

- In synthetic camera, projectors intersect a plane, usually in between scene and pinhole, projecting the scene onto that plane (no inversion)
- Also, in this camera, projection is mapped to some form of viewing medium (e.g., screen)
- For practical rendering, add front and back clipping planes



View Volumes

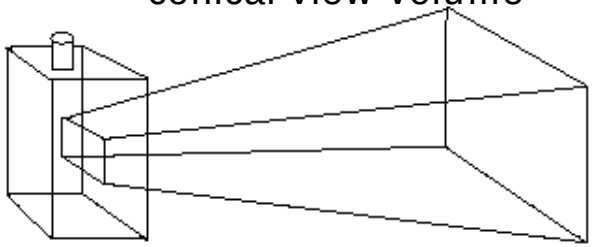
- A view volume contains everything the camera sees
- Conical – Approximates what eyes see, expensive math when clipping objects against cone's surface (simultaneous linear (for projector) and quadratics (for cone) equations)
- Frustum – Can approximate conical view volume with rectangular pyramid
  - Simultaneous linear equations for easy clipping of objects against sides (stay tuned for clipping lecture)
- Parallel – Doesn't simulate eye or camera (e.g., for orthographic projections).

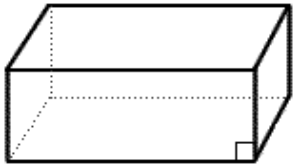


Conical perspective view volume (eye's is much wider, e.g., ≥180 degrees, esp. for motion!)

synthetic camera

Frustum: approximation to conical view volume



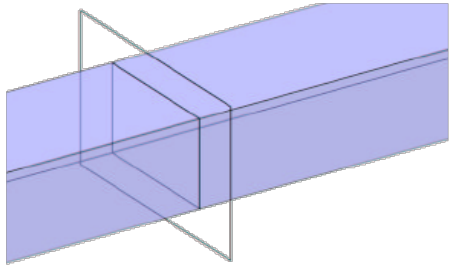


View volume

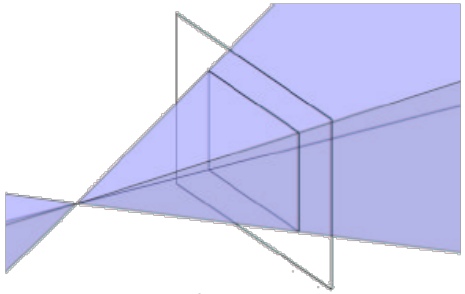
(Parallel view)

View Volumes and Projectors

- Concerns 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 the look vector, the direction in which the camera is looking)
- Perspective Volumes: Projectors emanate from eye point = centre of projection, inverse of rays of light converging to your eye (see Dürer woodcut)



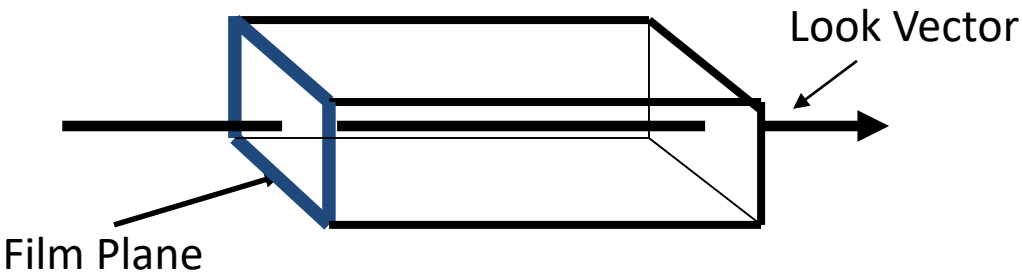
Parallel volume projectors

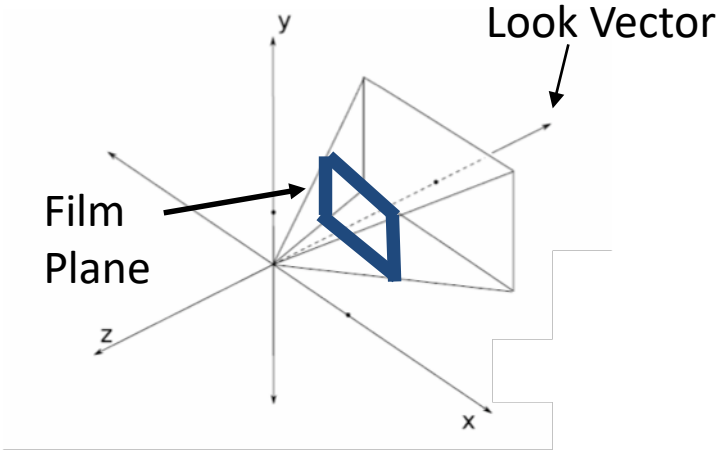


Perspective volume projectors

The Film Plane

- Film plane is 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 via window -> viewport mapping (Transformations lecture)
- Film for our camera model will be perpendicular to and centred around the camera's look vector and will match dimensions of our view volume

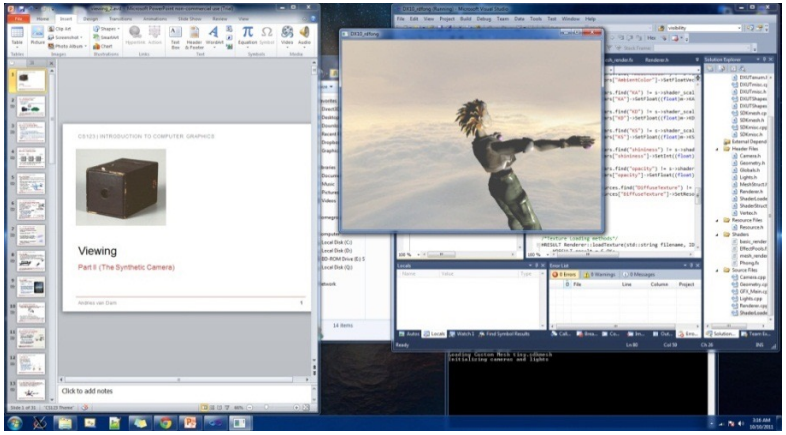




- Actual location of film plane along look vector doesn't matter as long as it is between eye / centre of projection and scene (identical projection for parallel, proportions preserved for perspective)

The Viewport

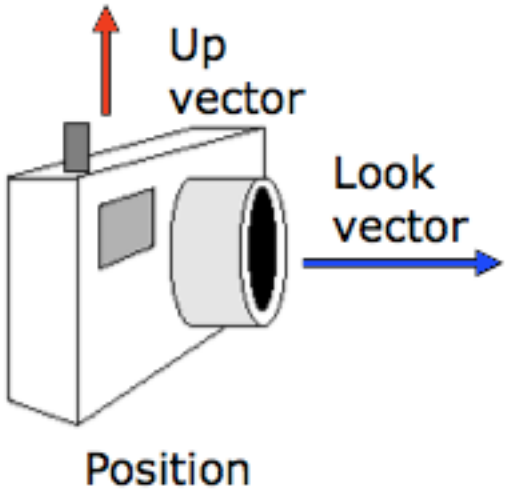
- Viewport is the rectangular area of screen where a scene is rendered
  - Corresponds to Window Manager's client area (the non-chrome part of the window)
  - Window (aka Imaging Rectangle) in computer graphics means a 2D clip rectangle on a 2D world coordinate drawing or 2D projection of the 3D scene on the "film plane", and viewport is a 2D integer coordinate region of screen space to which clipped window contents are mapped, typically either the whole screen or the client area of a WM's window
- Pixel coordinates for viewport are most commonly referred to using a (u,v) coordinate system
  - Unfortunately, that (u,v) nomenclature is also used for texture coordinates -- context disambiguates



Constructing the View Volume

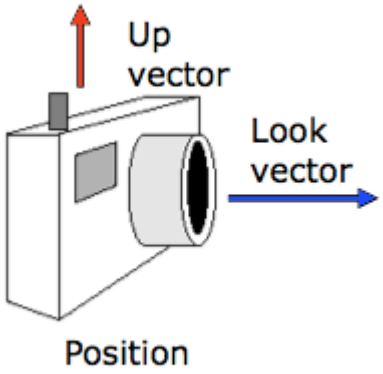
- We need to know six parameters about our synthetic camera model in order to take a picture using our perspective view **frustum:**

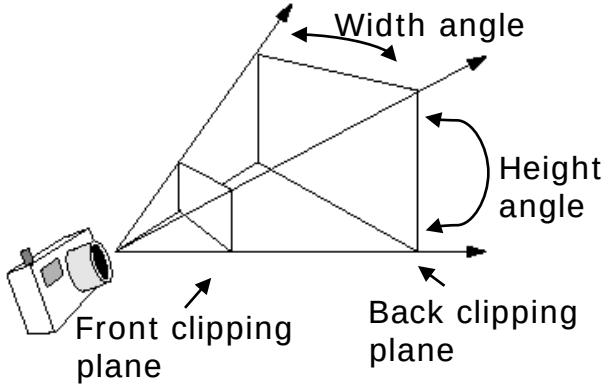
1) *Position* of the camera – it is the center of projection, behind the film plane
2) *Look vector* specifying direction camera is pointing. Note: ***look*** is not explicitly named or specified in OpenGL, but is derived using 'eye' and 'center' camera parameters (as in glm::lookAt()), which are the camera's position and the point it's looking at, respectively



Constructing the View Volume (cont.)

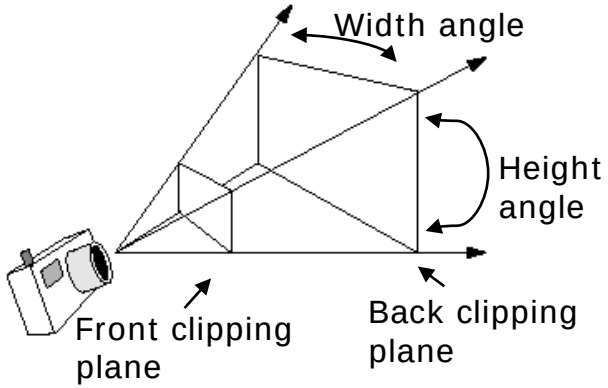
3) Camera's orientation determined by look and angle by which the camera is rotated about that vector, i.e., the direction of the up vector in World Coordinate System (WCS). It will be specified by an Up vector that needn't be perpendicular to Look, i.e., lie in the plane defined by Look as its normal
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





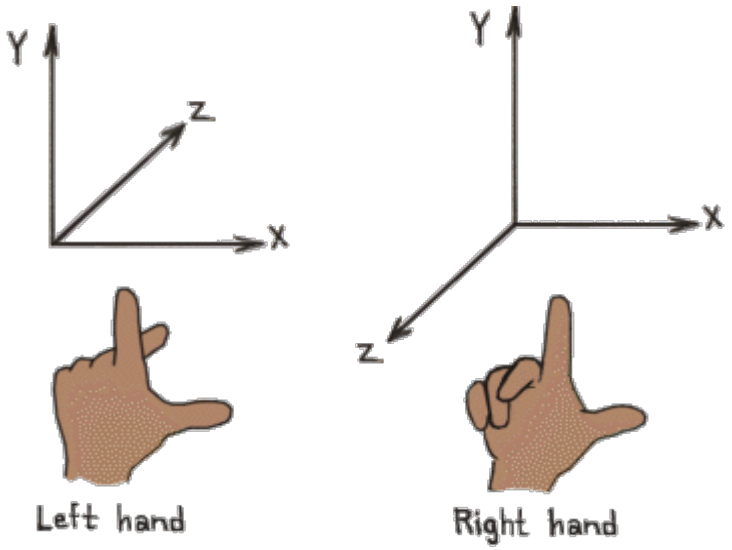
Constructing the View Volume (cont.)

6) Front and back clipping planes: limit extent of camera's view by rendering (parts of) objects lying between them and clipping everything outside of them – avoids problem of having far-away details map onto same pixel, another kind of alisiasing, i.e.,, sampling error
   - And why spend resources computing a rendering of details too small to resolve?!?



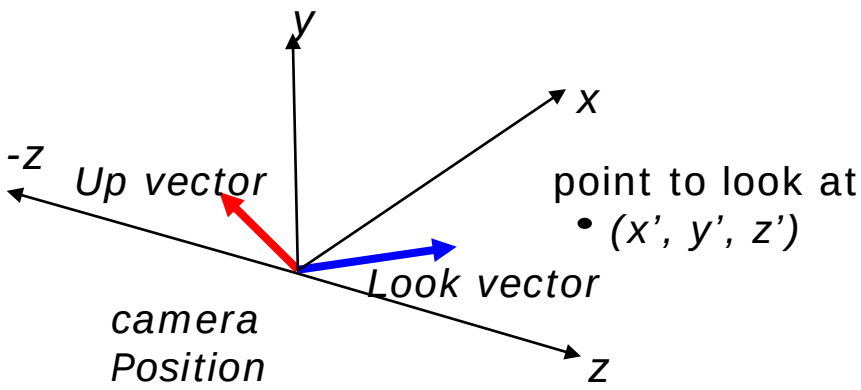
Camera Position

- Where is the camera located with respect to the origin, in the world coordinate system?
- For the 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 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



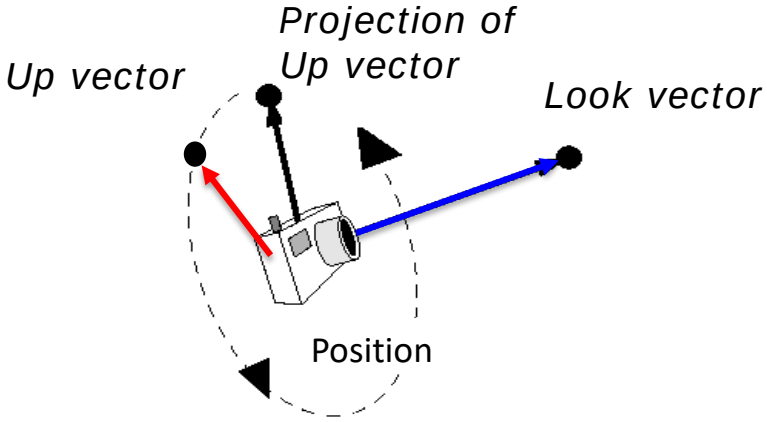
Orientation: Look and Up vectors

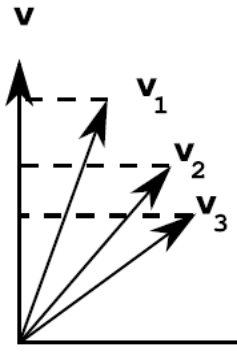
- Orientation is specified by a direction to look in (equivalently, a point in 3D space to look at) and a non-collinear vector Up from which we derive the rotation of the camera about Look
  - Note: glm::lookAt() sets up our viewing space by taking three values: vectors of an eye position, a point to look at, and an arbitrary non-colinfear Up vector
- In diagram, camera is positioned at origin of WCS , but that isn't typical



Orientation: Look and Up vectors (cont.)

- 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 about look
  - e.g., holding camera in portrait or landscape mode
  - Up must not be co-linear to look but it doesn't have to be perpendicular– actual orientation will be defined by the unit vector v perpendicular to look in the plane defined by look and up
    - easier to spec an arbitrary (non-collinear) vector than one perpendicular to look

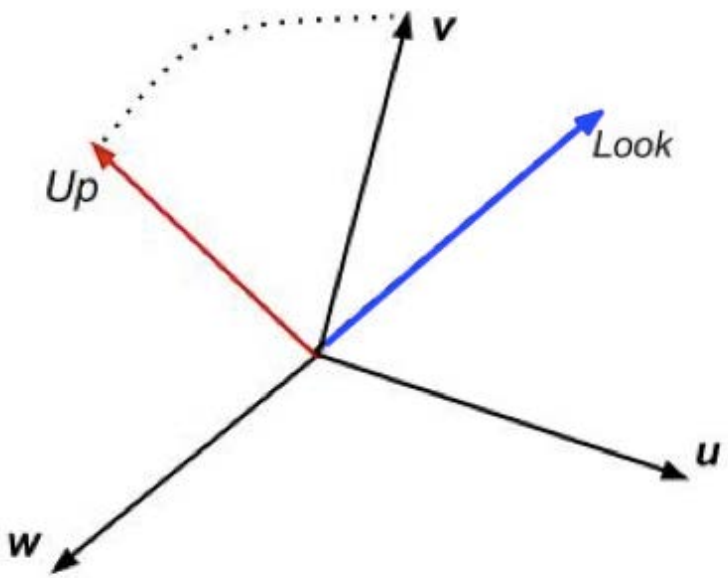




Any of v, v_1, v_2, v_3 could be the ***Up*** vector, with corresponding projections, which then must be normalized

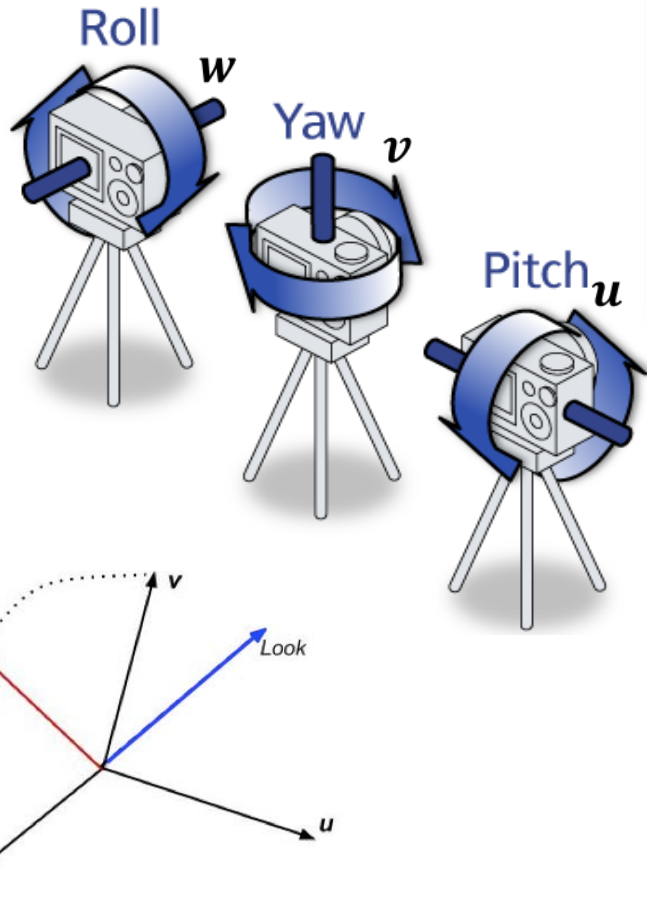
The Camera Coordinate Space

- The equivalent of x, y and z WCS axes in camera space are unit vectors u, v, w (u, v, not to be confused with texture axes, w is the homogenous coordinate,)
  - Also a right handed coordinate system
  - w is a unit vector in the opposite direction of look (i.e. look lies along the –-w axis)
  - v is the component of the up vector perpendicular to look, normalized to unit length
  - u is the unit vector perpendicular to both v and w



The Camera Coordinate Space (cont.)

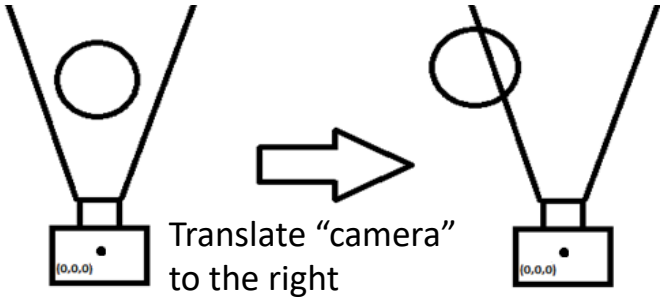
- Three common rotation transformations that use camera space axes, with camera in same position
- Roll:
  - Rotating your camera around w
- Yaw:
  - Rotating your camera around v
- Pitch:
  - Rotating your camera around u
- To do these, send camera to WCS origin and rotate to align its axes with the WCS axes, then use our rotation matrices to perform specified transformations, then un-rotate, and un-translate



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 this concept is abstracted away from the programmer with glm::lookAt(), in which a viewing matrix is created from up, camera position, and point it’s looking at

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



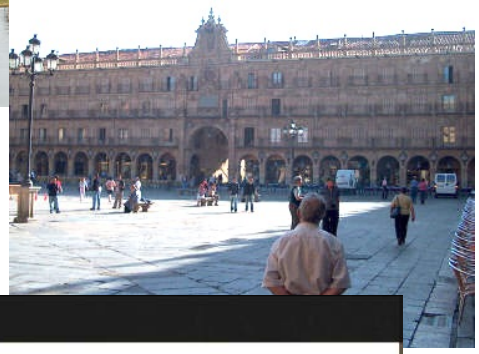
Translate “camera” to the right

Object moves to left to simulate a camera moving to right

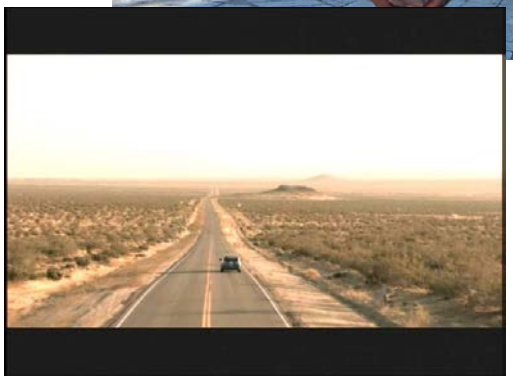
Aspect Ratio

- Analogous to dimensions of film in camera
- Ratio of width to height of viewing window
- Viewport's aspect ratio usually defined by screen being used , or by common ratio's, e.g., 16:9
  - Square viewing window has a ratio of 1:1
  - NTSC TV is 4:3, HDTV is 16:9 or 16:10

1:1



4:3

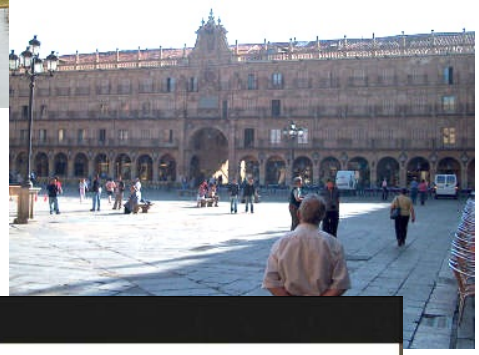


16:9

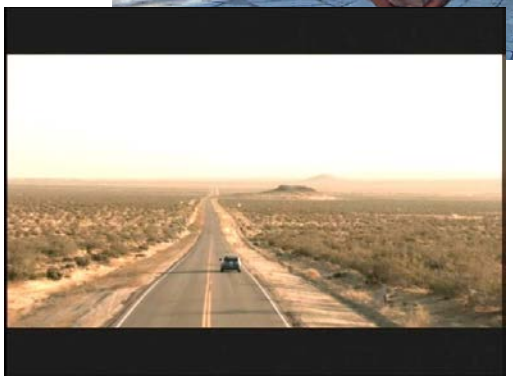
Aspect Ratio (cont.)

- Aspect ratio of viewing window defines dimensions of the image that gets projected to film plane, after which it is mapped to viewport
  - Typically it's a good idea to have same aspect ratio for both viewing window and viewport, to avoid distortions/stretching
  - The black strips on the 16:9 image is a technique called letter boxing. It preserves the aspect ratio of the image when the screen can't accommodate it. This is in contrast to simply stretching the image which distorts the images (most notably, faces)

1:1



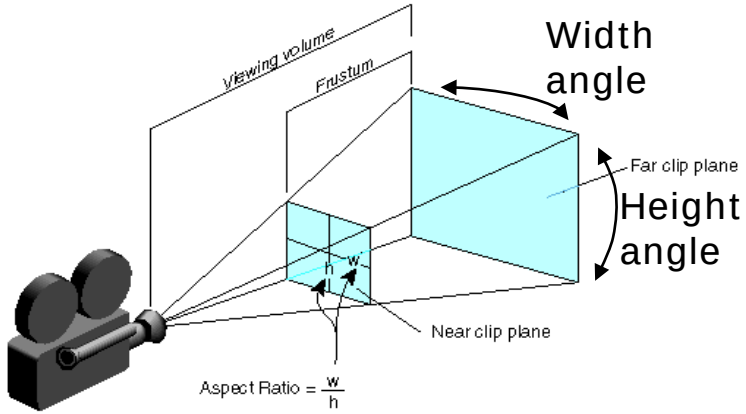
4:3



16:9

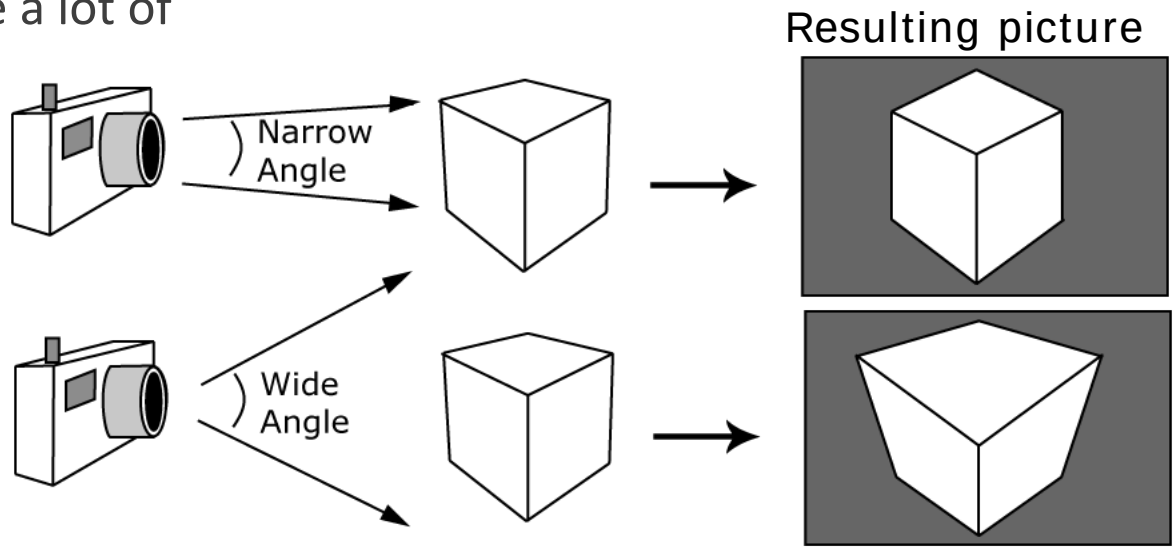
View Angle

- 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 height angle and aspect ratio
- Choosing view angle is analogous to photographer choosing a specific type of lens (e.g., a wide-angle or telephoto lens)



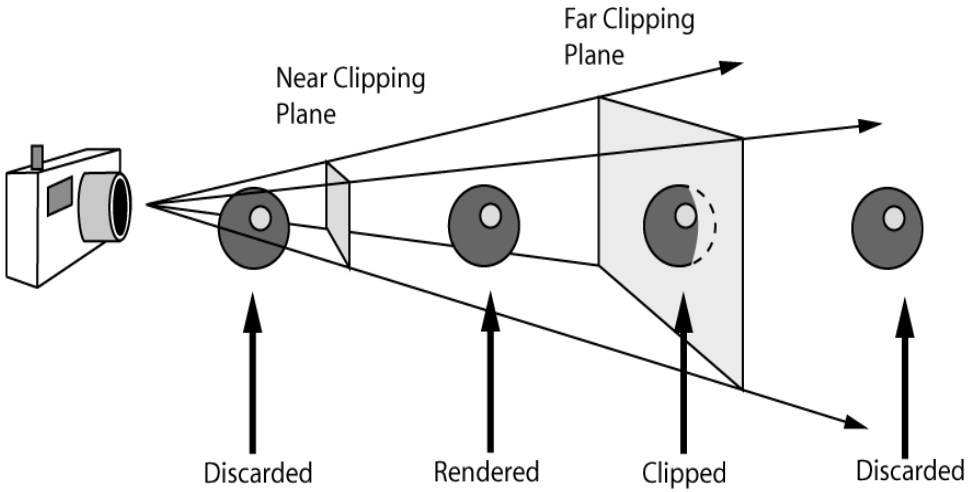
Viewing Angle (cont.)

- Telephoto 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



Near and Far Clipping Planes

- With what we have so far we can define four rays extending to infinity. These define 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 (also note that look and clipping planes are perpendicular)
- This volume (the frustum) defines what we can see in the scene
- Objects outside are discarded
- Objects intersecting faces of the volume are “clipped”



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 quite 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; different parts of an object may map onto same pixel (sampling error)
  - By discarding them we lose a small amount of detail but reclaim a lot of rendering time
  - Helps to declutter a scene
- These planes need to be properly placed, not too close to the camera, not too far

Games and Clipping Planes

- 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
- 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).

Fence is opaque

Fence is partially transparent

Screenshots from the game, *Okami*

Games and Clipping Planes (cont.)

- 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 the far clip plane.
- Old solution: add fog in the distance. A classic example, Turok: Dinosaur Hunter
- Modern solution (e.g. Stone Giant), dynamic level of detail: mesh detail increases when you get closer to it in the game
- Thanks to fast hardware and level of detail algorithms, we can push the far plane back now and fog is much less prevalent