

Viewing

Part II (The Synthetic Camera)

Bin Sheng ©

10/6/2016

Brownie camera courtesy of http://www.geh.org/fm/brownie2/htmlsrc/mE13000034_ful.html

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

Bin Sheng © 10/6/2016

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.)
- CS123 uses a simpler, slightly less powerful model than the one used in the book
 - Omit tilt of view/film plane (i.e., no oblique projections), focal distance (blurring)



- We 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 for Perspective Projection

- Our camera is related to a pinhole. Look through the pinhole and see a volume of space. Rays of light reflect off objects and converge to the pinhole to let you see the scene on a film plane or wall behind the pinhole. The scene will be inverted.
- The pinhole is where our camera position will be ("center of projection"), and the volume we see will be our "view volume"
 Pinhole Camera
- In our camera, projectors intersect a plane, usually in between scene and pinhole, projecting the scene onto that plane
- Lastly, in synthetic camera, projection is mapped to some form of viewing medium (e.g., screen)
- For practical rendering, also have front and back clipping planes



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)
- Also parallel view volumes, e.g., for orthographic projections. These don't simulate eye or camera



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



Parallel volume projectors



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
 - Corresponds to Window Manager's client area
 - Note: window (aka Imaging Rectangle) in computer graphics means a 2D clip rectangle on a 2D world coordinate drawing, and viewport is a 2D integer coordinate region of screen space to which clipped window contents are mapped – it is the client area of a Window Manager'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



CS337 | INTRODUCTION TO COMPUTER GRAPHICS Constructing the View Volume (1/2)

- 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 (from where it's looking) 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
 - 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)



Position

Constructing the View Volume (2/2)

- *3) Position* of the camera (from where it's looking) it is the center of projection, behind the film plane
- *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 **clipping** everything outside of them avoids problem of having far-away details map onto same pixel, i.e., "**sampling error**" (much more in image processing lecture)

(Optional) *Focal length*: objects at focal length are sharp, objects closer/farther are blurry



1) Position (1/1)

- Where is the camera located with respect to the origin, in the world coordinate system?
- 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 index finger up along the +y axis, and point your middle finger towards the +z axis
 - If you're looking at a screen, the axis will be positive coming towards you





Courtesy of http://viz.aset.psu.edu/gho/sem_notes/3d_fundamentals/gifs/left_right_hand.gif

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

- Orientation is specified by a direction to look in (equivalently, a point in 3D space to look at) and a vector defining the rotation of the camera about this direction
- These correspond to the *look* and *up* vectors
 - Note: glm::lookAt() sets up our viewing space by taking three values: vectors of an eye position, a point to look at, and an *up* vector
- > In diagram below, camera is positioned at origin of WCS , but that isn't typical



- 2 & 3) Orientation: Look and Up vectors
- Look Vector
 - Direction the camera is pointing
 - Three degrees of freedom; can be any vector in 3space
- Up Vector
 - Determines how camera is rotated about *look*
 - For example, 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*



The Camera Coordinate Space (1/2)

- The equivalent of x, y and z WCS 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 *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 (2/2)

- 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



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 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



Object moves to left to simulate a camera moving to right

- 4) Aspect Ratio (1/1)
- Analogous to dimensions of film in camera
- Ratio of width to height of viewing window
- Viewport's aspect ratio usually defined by device being used
 - 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 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
 - Note: 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)



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 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)



CS337 | INTRODUCTION TO COMPUTER GRAPHICS 5) Viewing Angle (2/2)

- 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



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

- 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"

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 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

- 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; 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 (mathematical justification in Viewing III)

Games and Clipping 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 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).

Bin Sheng © 10/6/2016

Fence is

opaque

Games and Clipping 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 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 closer



Courtesy of http://www.ato micgamer.com/s creenshots/gam e-1552/10965-800.jpg



Courtesy of http://image s.tweaktown. com/news/1 /4/14981_07 .jpg

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



Courtesy of http://3d-pic.3ddl.net/uploads/allimg/110617/13-11061G05J20-Ljpg

CS337 | INTRODUCTION TO COMPUTER GRAPHICS The Parallel View Volume (1/2)

- Up until now the specifications for a perspective view volume
- Also need parallel view volumes (e.g., for measurement in CAD, architecture)
- 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/cuboid (all opposite edges parallel)



The Parallel View Volume (2/2)

- Objects appear same size no matter how far away since projectors are all parallel: uniform foreshortening based on angle of film plane to projectors, not depth-dependent perspective (non-uniform) foreshortening
- Benefits of parallel view volume
 - Easier clipping because of simpler plane equation
 - Easier depth comparison for Visible Surface
 Determination (solving the obscuration problem)
 - Really easy to project a 3D scene to 2D projection/film plane because no perspective foreshortening



Capabilities of the Generalized Camera (1/2)

- In a more generalized camera the viewing window doesn't have to be centered about the position of the camera and *look* (look direction is d in diagram)
- Nor does it have to be perpendicular to *look*



- > 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)

• We can do oblique projections (cavalier, cabinet parallel; perspective oblique) where *look*/projectors and film plane aren't perpendicular:

Non-oblique view volume:



Oblique view volume:

view camera: **look** vector (tilt of lens) is at an angle to the film plane, which remains parallel to face of interest, preserving its proportions



- Our model of a camera is not all-encompassing
- There are some capabilities that we have omitted for the sake of simplicity, e.g., focus, oblique projections
- Our film is centered around the camera position and always perpendicular to *look*

Bin Sheng © 10/6/2016

Next Task…

- 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
- > P.S.: Get ready to use our linear algebra transformations!