Cameras

- Capture images
  - a measuring device
- Digital cameras
  - fill in memory with color-sample information
  - CCD (Charge-Coupled Device) instead of film
  - film also has finite resolution (graininess)
    - depends on speed (ISO 100, 200, …, 6400, …)
    - size (35mm, IMAX etc)

1

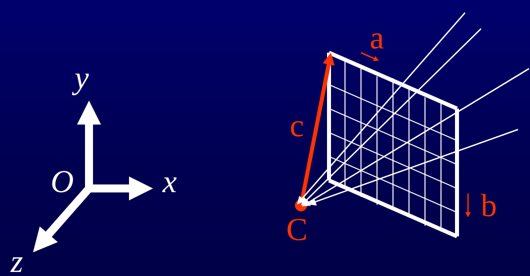
Importance of camera models

- Understanding cameras allows:
  - Using photographs of real world for modeling and rendering
  - Rendering 3D scenes, which is equivalent to taking pictures of the virtual world

2

Planar pinhole camera model

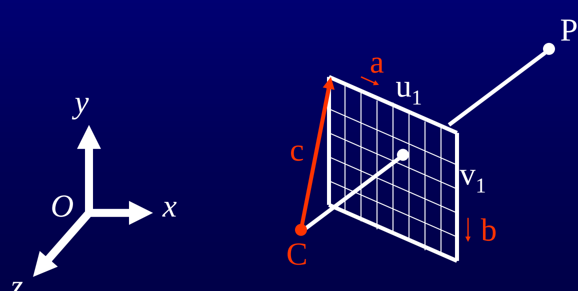
- Pinhole C
  - also called center of projection
  - point of convergence of all incoming rays



3

Planar pinhole camera model

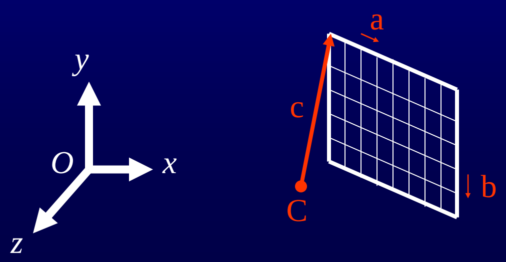
- Image plane
  - plane where intersecting incoming rays create the color samples (pixels) of the image
  - defined by non-parallel vectors *a* and *b*



4

Planar pinhole camera model

- Point C and vectors a, b, c define a general planar pinhole camera



5

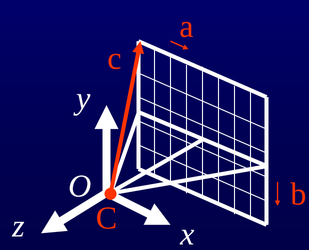
Special pinhole camera model

- OK to assume that
  - vectors a and b are perpendicular
  - square pixels (a and b same length)
  - C projects in the center of the image plane

6

Constructor

- PHC(float *hfov*, int *w*, int *h*)
  - *hfov* is the horizontal field of view [degrees]
  - *w* is the width of the image [pixels]
  - *h* is the height of the image [pixels]



$$\bar{a} = (1,0,0)$$

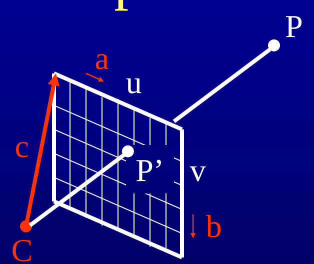
$$\bar{b} = (0,-1,0)$$

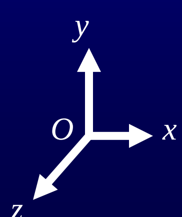
$$\dot{C} = (0,0,0)$$

$$\bar{c} = \left(-\frac{w}{2}, \frac{h}{2}, -\frac{w}{2\tan(hfov/2)} \right)$$

7

Projection of points





- w <= 0, no projection
- w > 0, valid projection

$$\dot{P} = \dot{C} + (\bar{a}u + \bar{b}v + \bar{c})w$$

$$\begin{bmatrix} \bar{a} & \bar{b} & \bar{c} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} w = \dot{P} - \dot{C}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} w = \begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{bmatrix}^{-1} \begin{bmatrix} P_x - C_x \\ P_y - C_y \\ P_z - C_z \end{bmatrix}$$

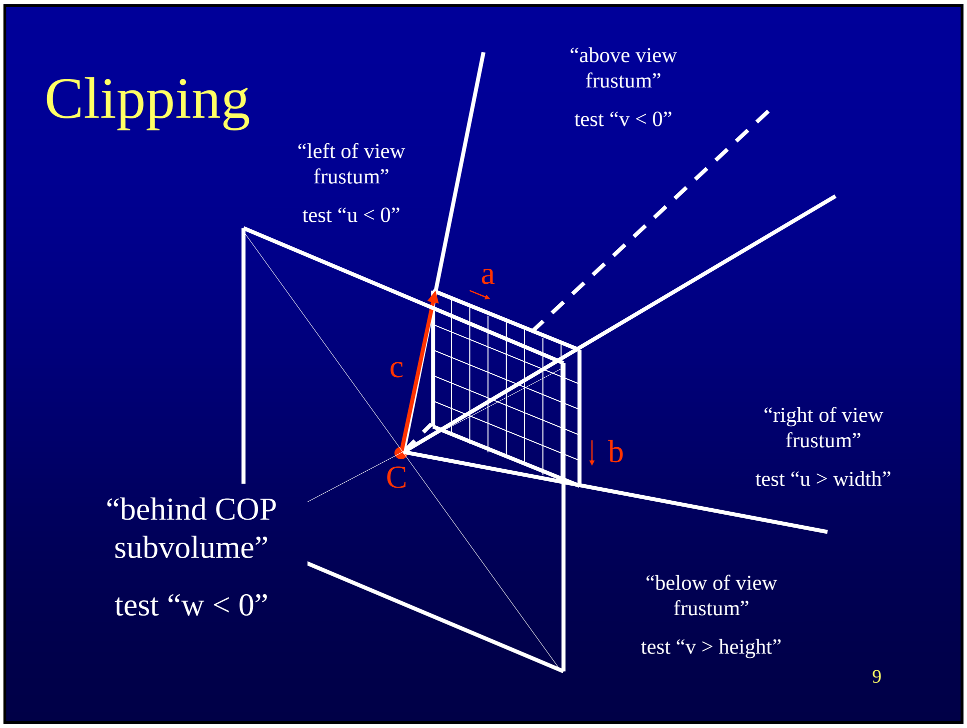
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} w = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \end{bmatrix}$$

$$w = q_2$$

$$u = \frac{q_0}{w}$$

$$v = \frac{q_1}{w}$$

8



Pixel coordinates

- Pixel (*u*, *v*) has center at (.5f+(float)*u*, .5f+(float)*v*)
- The image stretches from 0.0f to (float)*w*, and from 0.0f to (float)*h*
- A row has *w* pixels: pixel 0, 1, …, *w*-1
- A column has *h* pixels: pixel 0, 1, …, *h*-1
- An image point (*uf*, *vf*) – *uf* and *vf* are floats – belongs to pixel ((int) *uf*, (int) *vf*)

10

Other camera methods

- Access
  - Get view direction & focal length
  - Get ray & pixel center
  - Get horizontal / vertical field of view
  - Get principal point (pixel coordinates of COP projection onto image plane)
- Navigation
  - Translation left-right, up-down, forward-backward
  - Rotation left-right (pan, yaw), up-down (tilt, pitch), sideways (roll)
  - Revolve horizontally around point *P*, *theta* degrees
  - Revolve vertically around point *P*, *theta* degrees
- Positioning
  - Place camera such that it looks at point *P*, from distance *d*, and has up vector *up*
- Internal parameters change
  - Zoom in-out (change of field of view)
  - Change of resolution
  - Cropping/extensions
- View interpolation
  - Give PHC_0 and PHC_1, create *N* cameras that smoothly change the view from PHC_0 to PHC_1

11

Access

- Get view direction
  - $vd = (axb)$.UnitVector()
- Get focal length
  - $f = vd*c$
- Get ray for pixel (u, v) -- integers
  - $ray(u, v) = a*(u+0.5f)+b*(v+0.5f)+c$
- Get ray for pixel image point (uf, vf) -- floats
  - $ray(uf, vf) = a*uf + b*vf + c$
- Get pixel center -- integers
  - $P(u, v) = C + ray(u, v)$
- Get horizontal field of view
  - $hfov = 2*\text{atan}(w/2*a.\text{Length}()/f)$ // assumes *C* projects at $w/2$
- Get principal point (image coordinates of *C* projection)
  - $PP_u = -c*a.\text{UnitVector}()/a.\text{Length}()$
  - $PP_v = -c*b.\text{UnitVector}()/b.\text{Length}()$

12

Translations

$\overline{a}' = \overline{a}$

$\overline{b}' = \overline{b}$

$\overline{c}' = \overline{c}$

$left - right$

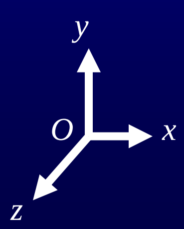
$\dot{C}' = \dot{C} + \overline{a}.UnitVector() * step$

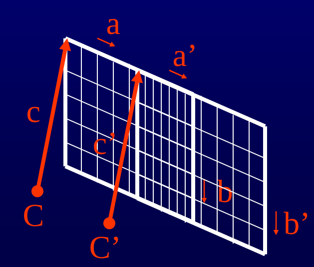
$up - down$

$\dot{C}' = \dot{C} - \overline{b}.UnitVector() * step$

$forward - backward$

$\dot{C}' = \dot{C} + (\overline{a} \times \overline{b}).UnitVector() * step$





Translation right

13

Rotations

Pan

$\overline{a}' = \overline{a}.RotateAbout(\dot{C}, \dot{C} - \overline{b}, \theta)$

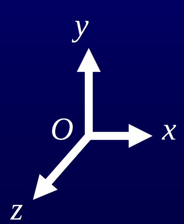
$\overline{b}' = \overline{b}.RotateAbout(\dot{C}, \dot{C} - \overline{b}, \theta)$

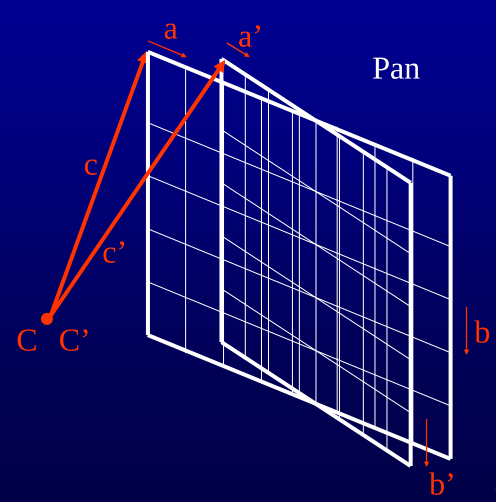
$\overline{c}' = \overline{c}.RotateAbout(\dot{C}, \dot{C} - \overline{b}, \theta)$

$\dot{C}' = \dot{C}$

$Tilt : RotateAbout(\dot{C}, \dot{C} + \overline{a}, \theta)$

$Roll : RotateAbout(\dot{C}, \dot{C} + \overline{a} \times \overline{b}, \theta)$





Pan

14

Camera positioning

- Place camera such that it looks at point *P* from distance *d*, has view direction *vd*, and *up* is a vector in the vertical plane of the camera

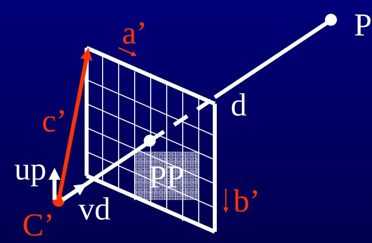
  // assumptions: rectangular pixels, up and vd are normalized

  *C’ = P – vd**d

  a’ = (*vd* x *up*).UnitVector()**a*.Length()

  b’ = (*vd* x *a’*).UnitVector()**b*.Length()

  $c' = – PP_u*a' – PP_v*b' + vd*f$



15

Zooming

- Focal length changes: *f’ = f * zoom*

  C’ = C

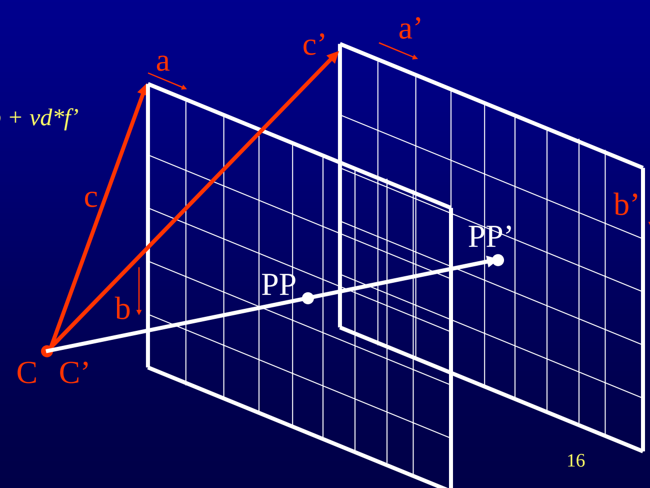
  a’ = a

  b’ = b

  $c' = - PP_u*a – PP_v*b + vd*f'$

C_PP = f

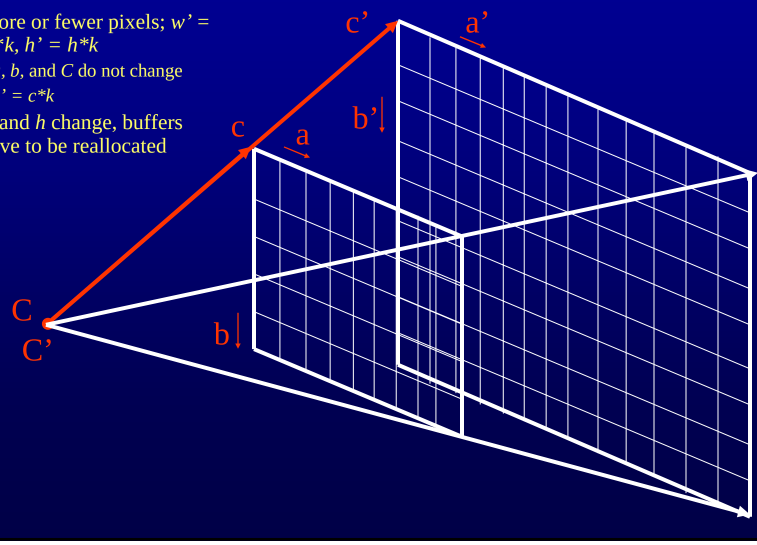
C’_PP’ = f’



16

Change of resolution

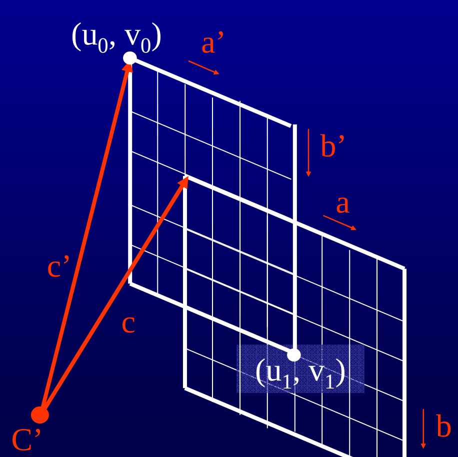
- More or fewer pixels; $w' = w*k$, $h' = h*k$
  - a, b, and C do not change
  - $c' = c*k$
- w and h change, buffers have to be reallocated



17

Cropping/extensions

- Set the image to rectangle (u_0, v_0, u_1, v_1)
  - $C' = C$
  - $a' = a$
  - $b' = b$
  - $c' = c + u_0*a + v_0*b$
  - $w' = u_1-u_0$
  - $h' = v_1-v_0$



18

View interpolation

- Given PHC_0 and PHC_1 create N intermediate cameras
  - *Assumption: PHC_0 and PHC_1 have the same internal parameters*

  $C_i = C_0 + (C_1\text{-}C_0)$*(float)$i$/(float)($N$-1)

  $vd_i = vd_0 + (vd_1\text{-}vd_0)$*(float)$i$/(float)($N$-1)

  $a_i = a_0 + (a_1\text{-}a_0)$*(float)$i$/(float)($N$-1)

  … (See camera positioning)

19

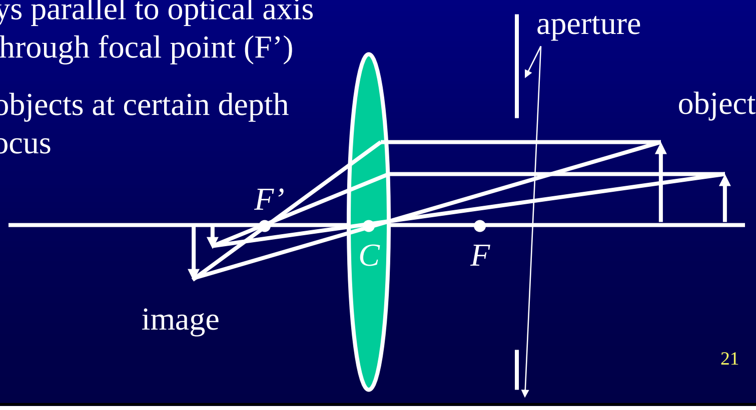
Real world camera models

- Aperture is finite
  - depth of field (only objects at a certain distance are in focus)
- Lens distortion
  - straight lines are curved in the image
  - barrel
  - pincushion

20

Depth of field

- Thin lenses
  - rays through lens center (C) do not change direction
  - rays parallel to optical axis go through focal point (F')
- Only objects at certain depth are in focus



21