Cameras and Projections

COSC342

Lecture 6
15 March 2015

So What's This All About?

- The basic idea of cameras
- Pinhole cameras and lenses
- Projection matrices
- Rendering surfaces in cameras

Cameras and Projections

- Cameras project a 3D world onto a 2D image
- We will use (x, y, z) for 3D points, and (u, v) in 2D
- Input will be a 4-vector (homogeneous 3D point)
- Output will be a 3-vector (homogeneous 2D point)

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \mathrm{P} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- What form does P have?

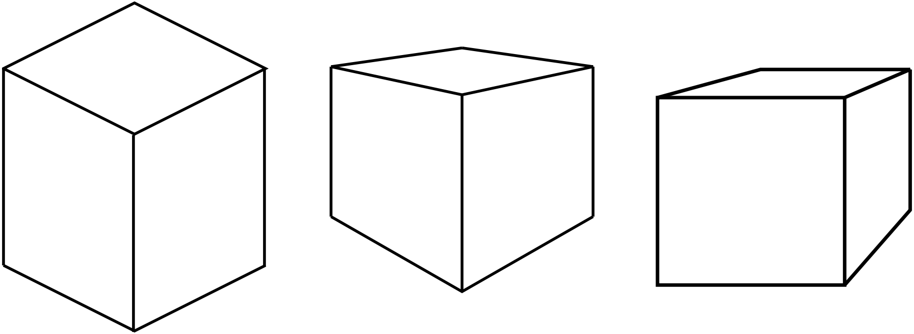
Orthographic Projection

- Simple way to go from 3D to 2D – delete one dimension
- Discarding the Z value projects onto the X-Y plane

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

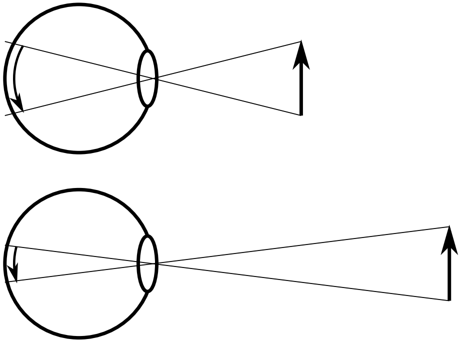
- This isn't how our eyes or most cameras work

Which Cubes are Drawn Correctly?



The Eye as a Camera

- The eye has a narrow opening (the pupil) with a lens
- This focuses light onto the retina where it is received
- This arrangement means that distant objects seem smaller



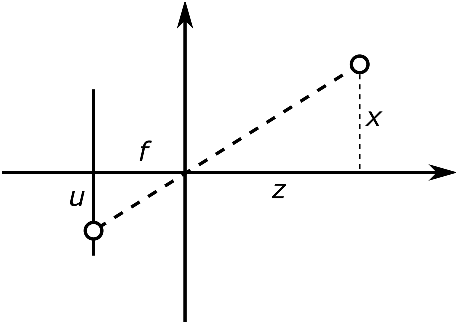
A Simple Camera Model

- The pinhole camera is a simple but useful model
- There is a central point of projection (the pinhole)
- Given a point in the world:
  - Cast a ray (line) from the point through the central point
  - Intersect this with an imaging plane
  - This intersection is the image of the world point
- This is a reasonable model for the eye and most cameras
  - The role of the lens is to let a large hole act like a pinhole
  - This lets enough light in to make an image with real sensors

The Pinhole Camera

- The distance from the pinhole to the image plane is the focal length, f
- By similar triangles, a 3D world point (x, y, z) projects to

$$u = \frac{-fx}{z} \qquad v = \frac{-fy}{z}$$



The Pinhole Camera

- We can avoid the sign change by putting the image plane in front of the camera centre
- This isn't practical for real cameras, but is mathematically equivalent

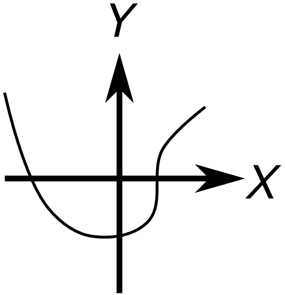
$$u = \frac{fx}{z} \qquad v = \frac{fy}{z}$$

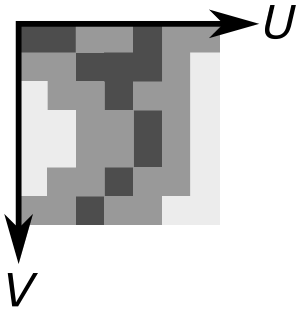
- We can express this as a projection matrix

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Co-ordinate Frames

- This assumes that the camera centre is at the origin
- The camera faces along the positive Z-axis
- The U axis runs from left to right in the image
- For a right-handed system, the V axis runs top-to-bottom
- This is different to our usual axes for $X - Y$ plots





Camera Co-ordinates

- Our projection puts the origin at the centre of the image, (c_u, c_v)
- We can move it to the top left corner by a shift
- In matrix form this makes our projection matrix

$$\begin{bmatrix} f & 0 & c_u & 0 \\ 0 & f & c_v & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- Often this is rewritten as

$$\begin{bmatrix} f & 0 & c_u \\ 0 & f & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \mathrm{K} \begin{bmatrix} \mathrm{I} & 0 \end{bmatrix}$$

where K is the *camera calibration matrix*

Transforming Cameras

- You can rotate and translate cameras
- It is easier to apply the inverse transform to the world
- E.g.: shifting the camera left 3 units = moving the scene right 3 units
- Often we rotate the camera by R and then shift by $\mathbf{t}$
- Equivalently, shift the points by $-\mathbf{t}$, then rotate them by R^T

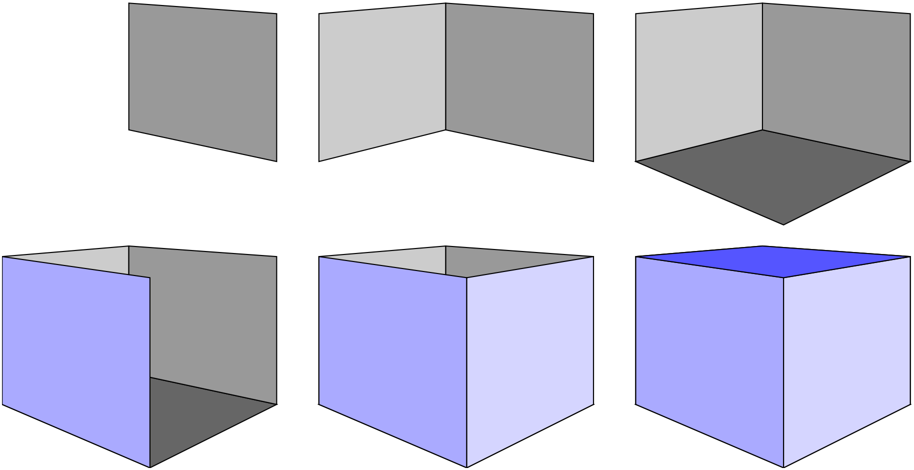
$$\begin{bmatrix} f & 0 & c_u \\ 0 & f & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{21} & r_{31} & 0 \\ r_{12} & r_{22} & r_{32} & 0 \\ r_{13} & r_{23} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \mathrm{K} \begin{bmatrix} \mathrm{I} & 0 \end{bmatrix} \begin{bmatrix} \mathrm{R}^\mathsf{T} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{I} & -\mathbf{t} \\ 0 & 1 \end{bmatrix} = \mathrm{K} \begin{bmatrix} \mathrm{R}^\mathsf{T} & -\mathrm{R}^\mathsf{T}\mathbf{t} \end{bmatrix}$$

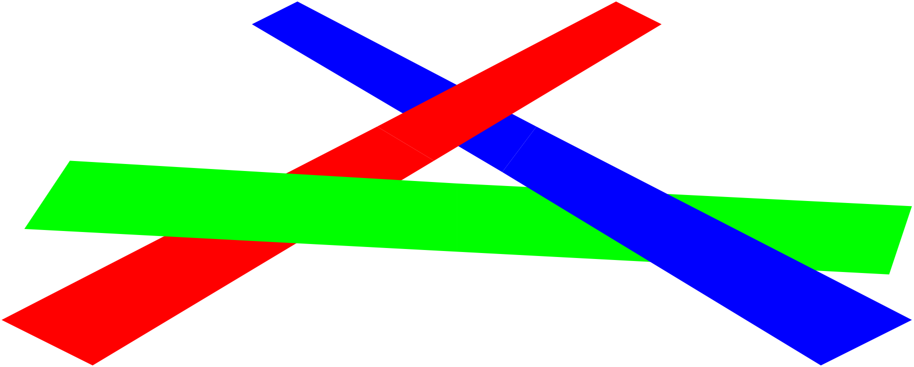
Rendering Scenes

- The camera model lets us project 3D points into the image
- Generally we want to draw surfaces such as triangles, planes, etc.
- We also need to deal with occlusion – some surfaces are hidden behind others
- A simple approach is the *Painter's Algorithm*
  - Order the surfaces by distance from camera
  - Draw the furthest surfaces first, and the nearest last

Painter's Algorithm



Painter's Algorithm



- Which surface should you draw first?

Z-Buffering

- The usual solution to this is the use of Z-buffers
- As well as the colour values, we record the depth (Z) at each pixel
- Only draw a new pixel if the new Z-value is less than the current one
  - If two surfaces are at the same depth, this is not deterministic
  - This leads to 'Z-fighting', and artefacts in the image
  - Because of limited precision, this can happen with surfaces which are close to each other but do not quite coincide
- You also need to be careful how you implement this
  - Depth and z values not quite the same thing
  - Linear interpolation across triangles is not quite right

Z-Buffering