Chapter 11

Lecture 11: Cameras, lenses, and calibration

Tuesday, Feb. 22, 2010 MIT EECS course 6.869, Bill Freeman and Antonio Torralba
Slide numbers refer to the file "lecture11pinholesLenses.key"

slides 2-25, see notes for slides 77-100 of previous lecture

slides 26 - 33 Now it's time you were told where images come from–today we'll talk about cameras. Why did we wait so long? We prefered to let you dive in with manipulating images. Now that you have image manipulation experience under your belt, we can talk about cameras and how they render the world.

I spent considerable amount of time as a young kid trying to figure out why you don't see an image when you hold up a white sheet of paper. Does anyone have a succinct explanation for it?

These images, showing how light bounces around, try to tell that story. We do see images on the blank sheet of paper when we hold it up. The problem is that we see very many of them, all superimposed on each other, and everything gets washed out.

You can see a more coherent image on a white sheet of paper when you hold it up if you remove all the extraneous images that are landing on it, and just let one shine in. Let's put our white sheet of paper in a box, and let's open up only one tiny hole in the wall opposite the blank sheet of paper. This will let in the rays shown here, and create an image, falling on the white sheet.

slide 34 show pinhole camera room demo. show pinhole camera poster board.

slides 35-45 Description of problem set 7. See problem set write-up.

slides 46-47 One of the many things we want to do with vision is to tell where things are, or how big they are. To do that, we'll need to know how the position of 3d points in the world relate to where the 2d positions of those points in the image. This relationship is called the projection model.

It's pretty easy to work out the projection model for a pinhole camera. Here's the relevant picture. Point "O" represents the pinhole, and we set that to be the origin of our coordinate system. The x and y coordinate basis vectors are i and j, respectively. The k unit vector forms a right handed coordinate system and points toward the camera, as it's drawn here.

Let z be the distance from the object to the camera, and the focal length, f, be the distance from the sensor plane to the pinhole. Then, by similar triangles, we have that $\frac{y}{z} = \frac{y'}{f}$. Dividing both sides by

f and dropping the z' coordinate of the sensor plane (the same for every point in the image), leads the perspective projection equations,

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{fx}{z} \\ \frac{fy}{z} \end{pmatrix} \tag{11.1}$$

$$\tag{11.2}$$

slide 48 Oftentime for convenience, we draw the sensor plane in front of the focal point. That just flips and scales the image, but makes it easier to see how points are imaged.

slide 49 Let's examine some properties of projection. A point in the 3d world gets mapped to... a point in the 2-d world. That's an easy one. Lines go to... lines, except in degenerate cases. What are the degenerate cases? Any line that goes through the focal point.

Planes are mapped to either to the whole image, or to half-planes. Although a plane passing through the focal point gets mapped to a line.

slide 50-52

Let's see what a line maps to. A line passing through the point $\begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}$ parallel to the vector $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$ has a parameterized form given by

$$x(t) = x_0 + at \tag{11.3}$$
$$y(t) = y_0 + bt \tag{11.4}$$
$$z(t) = z_0 + ct \tag{11.5}$$

Let's put the points on that line into the perspective projection equation. Doing that, and letting $t \to \infty$ reveals a family of lines that all get mapped to the same point at $t = \infty$ (for $c \neq 0$): all the lines parallel to the vector $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$.

This is called a "vanishing point". Here we can see a picture of that in cross section. All lines in any family of parallel lines converge to the same vanishing point. Of course, artists are aware of this property of parallel lines. Here you can see that each of these two families of parallel lines on the building converge to a vanishing point.

You can also show that parallel lines in the same plane give collinear vanishing points. We call this the horizon.

slides 54, 55

Here's a question: what do you get if you photograph a brick wall, placed fronto-parallel to the camera? Do the parallel lines all converge to two different points on the right and left hand sides of the image? As we described the imaging configuration, this is the degenerate case when $c = 0$ in the parameterized equations of the lines. We then solve the equations as shown in this slide.

slides 56-58

Perspective projection is simple but has that pesky $\frac{1}{z}$ makes the transformation be non-linear. There are two commonly used linear approximations that are used. One is orthographic projection–we just drop the z coordinate. A very large focal length camera approximates the orthographic projection.

Another, similar, linear model is called "weak perspective". Here, the x and y coordinates are scaled by the focal length divided by an average or proxy z value, z_0. This can model a set of points under perspective imaging conditions, provided they are near enough to each other in depth.

slides 59

Linearization is one way to handle the annoying division by z. A different way that people use is a simplification in the mathematical notation. We define what are called "homogeneous coordinates". The trick is to add one more coordinate to each vector. To go from a 2-d vector to its homogenous image coordinate, we introduce a 1 in the 3rd coordinate. To take a 3d vector into its homogeneous scene coordinate, we add a 1 as a 4th coordinate.

Homogenous coordinates are always defined only up to an unknown scale factor. To convert from homogeneous coordinates, the rule is to divide the other coordinates by the last coordinate (the top two by the third, or the top three by the fourth).

slide 60 Why make up these crazy rules? There are many benefits to using this representation for geometry (representing 2-d lines and line intersections is particularly simple), but we introduce them here to let us write perspective projection as a matrix multiplication. (equation on slide). Note that it all works out: our scene point is a 4-d vector that becomes a 3d vector in homogenous coordinates. When we convert that to Euclidean coordinates, we get the 2d vector that is the perspective projection of our original point. The matrix we multiplied the original vector by is called the projection matrix.

slide 61 Let's examine this matrix. Does its overall scaling matter? No, because homogenous coordinates are only defined up to an arbitrary scale factor, and their normalization when you convert back to Euclidean coordinates removes the effect of any overall scale factor in the projection matrix.

slide 62 Orthographic projection can also be written simply in homogeneous coordinates (although we didn't have trouble writing that in Euclidean coordinates).

slide 64 Many times, we have measured the positions, in pixels, of some feature points or the boundary of some object in the world, and we want to know what those pixel positions tell us about where those features or object may be in the real world.

The projection matrix tell us how things project from a camera-centered coordinate frame into coordinates in the focal plane, but coordinates that are in the same units as you measure things out in the world. We want to be able to go all the way, to relate world coordinate positions to camera pixel positions. Defining that relationship is called "camera calibration".

slide 65, 66

Here are two slides, from a past version of a future lecture, motivating why we might want to relate pixel values to world coordinates. You can measure pixel offsets of a person in an image and relate that to their height. You can measure spatial displacements in stereo images and relate that to the depth of the object.

slide 67, 68

First, let's remind ourselves how to account for translation and rotation, since as we move our camera around, it will, in general, have some arbitrary translation and rotation relative to the world coordinate frame.

First, regarding notation: we denote two coordinate systems by letters A and B. The left-superscript of a vector will indicate in which coordinate system the entries are referenced.

To transform a vector $\vec{p}$ from coordinate system A to B, we can first multiply $\vec{p}$ by a rotation matrix,

R, and then add in a translation vector, $\vec{t}$. We can use left-superscript B and left-subscript A for both R and $\vec{t}$ to indicate they take us from coordinate system A to B. The slide shows the transformation, in vector and matrix form.

Homogeneous coordinates let us write this a little more compactly, as shown in slide 68.

slide 69

Now we have the tools to relate world coordinates to camera pixel values. We can think of that transformation as involving two sets of parameters. First, those parameters that translate from a 3d coordinate system attached to the camera, with the origin at the pinhole (or lens aperture), are called the "intrinsic parameters", since they are intrinsic to the camera itself (at least, with its current focus settings). The "extrinsic parameters" describe the camera's rotation and translation relative to some world 3d coordinate system.

Let's first derive the intrinsic camera calibration parameters. Gradually building up in generality, we can first note that the perspective projection equation tells us how focal plane coordinates u and v relate to 3-d coodinates, x, y, and z (we're working in Euclidean coordinates here). [[equation in slide]]

slides 70, 71, 72

But the "pixels" are measured in arbitrary units, relative to the 3d coordinate units, so we need to use parameters α and β to describe that relationship. And the original of the pixel coordinates may not be centered on the optical axis, as the perspective projection equation assumes, so we should add an offset u_o and v_o to account for the origin of the pixel coordinates.

slides 73 Finally, there may be some skew in the pixel coordinate axes; they may not be orthogonal. If one axis is at an angle θ relative to the other, you can work out that the u and v axes (which we call primed up here in the small equation) then have this relationship to an unskewed set of u and v axes [[small equation on slide 73]].

Allowing for a possible skew angle θ between u and v then gives us this equation for the projection relation between camera-attached world coordinates, and focal plane pixel values: [[big equation on slide 73]].

(Note that if $\theta = 90°$ then this reduces to the equation of the previous slide).

slides 74 This was all in Euclidean coordinates. Using homogenous coordinates will let us re-write that (embellished) perspective projection equation as a matrix multiplication. Here is that matrix, and you can verify that it describes the same projection as the Euclidean version above.

So this matrix **K** lets us convert *from* the 3-d position in a coordinate frame attached to the camera *to* the focal plane pixel coordinates.

slides 75 Next is the easier part: we just need to convert from the camera-attached coordinate frame to world coordinates (centered in Greenwich, England or wherever you'd like to put your coordinate system origin and orientation). In the general case, that just involves specifying a rotation and translation vector to make that change of coordinates. Here we write it all together, using homogenous coordinates, as one matrix multiplication.

slides 76, 77 And we can then combine the intrinsic and extrinsic parameters to obtain a camera calibration matrix, relating 3d world coordinates to camera focal plane pixel indices.

slides 78 So how could you actually measure those parameters relating 3-d position to image pixel position?

You can put a calibration target into your image, and say that you know the 3-d position of each of

the corners of the black squares covering that box. Then you're going to measure the pixel position of each box corner, too.

slides 79-83

Going back to our summarizing results from before, we can put every measured corner, u_i and v_i in pixel coordinates, and $\vec{P}_i$ in world coordinates, in this equation that we know they must solve. We stack them all up, and rearrange to put the quantities we want to solve for in the column vector on the right.

We want to solve this equation, $Q\vec{m} = \vec{0}$ but want to avoid the trivial solution that $\vec{m} = 0$. The minimum eigenvector of the matrix Q^TQ gives us the solution we want, because it is the unit vector $\vec{x}$ that minimizes $\vec{x}^TQ^TQ\vec{x}$.

Once we have the entries $\vec{m}$ of the M matrix, that gives us the matrix M that lets us translate from world to pixel coordinates. We can also then solve explicitly for the intrinsic and extrinsic camera parameters, if we want to.

slides 84-96 For all this discussion of camera calibration and image projection models, we've been assuming that imaging happens through a pinhole. Why do we need lenses at all? To gather more light.

In their idealization, lenses have the magical property that all light rays leaving some point at a focussed depth that enter the lens aperture get redirected back to a point at the sensor plane. We'll show how you can shape a lens to do that. All you need is Snell's law, and then to use a spherical lens and the thin-lens approximation.

There's an interesting assumption being made when you use a lens to gather more light to image a surface. We're assuming that the Bi-directional Reflectance Distribution Function (BRDF) varies slowly over the cone of rays that intersect the lens and are focussed onto the camera sensor. Otherwise, it wouldn't make sense to average those rays together. Can you think of a situation where that doesn't hold? Photographing a mirror's surface is one such case.