An Introduction to Computer Visi👁n

or: An Unofficial Companion Guide to the Georgia Institute of Technology's **CS6476**: *Computer Vision*

George Kudrayvtsev

Last Updated: March 27, 2019

List of Algorithms

Multiple Views

> *There are things known and there are things unknown, and in between are the doors of perception.*
>
> — Aldous Huxley, *The Doors of Perception*

In this chapter we'll be discussing the idea of ***n*-views**: what can we learn when given multiple different images of the same scene? Mostly, we'll discuss $n = 2$ and create mappings from one image to another. We discussed this topic briefly in chapter 6 when talking about Stereo Geometry; we'll return to that later in this chapter as well, but we'll also be discussing other scenarios in which we create relationships from one image to another.

7.1 Image-to-Image Projections

There are many types of transformations; we open by discussing their mathematical differences.

Recall our discussion of perspective projection in 3D (see Equation 6.1): we had a 4×4 matrix and could model all of our various transformations (translation, rotation, etc.) under this framework by using Homogeneous Coordinates. This framework is convenient for a lot of reasons, and it enables us to chain transformations by continually multiplying. We can similarly adapt this model here with a 3×3 projective transformation matrix.

Translation This is our simplest transformation. We've seen this before: $\mathbf{x}' = \mathbf{x} + \mathbf{t}$. In our projective transformation model,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

This transformation preserves all of the original properties: lengths, angles, and orientations all remain unchanged. Importantly, as well, *lines remain lines* under a translation transformation.

Euclidean Also called a **rigid body transformation**, this is the 2D version of the rotation

matrix we saw in (6.7) combined with a translation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

This transformation preserves the same properties as translation aside from orientation: lengths, angles, and lines are all unchanged.

Similarity Under a **similarity transformation**, we add scaling to our rigid body transformation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a\cos\theta & -a\sin\theta & t_x \\ a\sin\theta & a\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Affine An **affine transformation** is one we've yet to examine but is important in computer vision (and graphics). It allows 6 degrees of freedom (adding **shearing** to the aforementioned translation, rotation, and scaling), and it enables us to map any 3 points to any other 3 points while the others follow the transformation.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

It preserves a very different set of properties relative to the others we've seen: parallel lines, ratios of areas, and lines (as in, lines stay lines) are preserved.

Combining all of these transformations gives us a **general projective transformation**, also called a **homography**. It allows 8 degrees of freedom:[1]

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \simeq \begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

It's important to be aware of how many points we need to determine the existence of all of these transformations. Determining a translation only requires a single point correspondence between two images: there are two unknowns (t_x, t_y), and one correspondence gives us this relationship. Similarly, for a homography, we need (at least) 4 correspondence points to determine the transformation.

7.2 The Power of Homographies

Why do homogeneous coordinates make sense? Let's return to our understanding of the 3D to 2D projection through an image plane to some camera center. In Figure 7.1, we can see

[1] The last element is a 1 for the same reason as it was in Method 2: Inhomogeneous Solution: because the homogeneous projective matrix is invariant under scale, we can scale it all by this last element and it will have no effect. This would give us a different w than before, but that doesn't matter once we convert back to our new (x', y').

such a projection. The point on the image plane at $(x, y, 1)$ is just the intersection of the ray with the image plane (in blue); that means *any* point on the ray projects onto the image plane at that point.

The ray is just the scaled intersection point, (sx, sy, s), and that aligns with our understanding of how homogeneous represent this model.

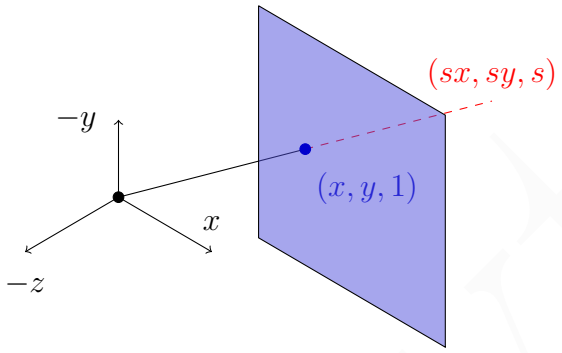


Figure 7.1: Each point on the image plane at $(x, y, 1)$ is represented by a ray in space, (sx, sy, s). All of the points on the ray are equivalent: $(x, y, 1) \simeq (sx, sy, s)$.

So how can we determine how a particular point in one projective plane maps onto another projective plane? This is demonstrated in Figure 7.2: we cast a ray through each pixel in the first projective plane and draw where that ray intersects the other projective plane.

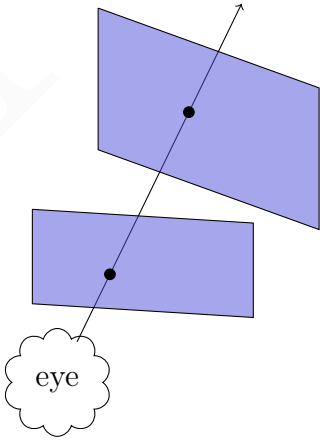


Figure 7.2: We can project a ray from the eye through each pixel in one projection plane and see the corresponding pixel in the other projection plane.

Notice that where this ray hits the world is actually irrelevant, now! We are no longer working with a 3D reprojection onto a 2D plane; instead, we can think about this as a 2D **image warp** (or just a transformation) from one plane to another. This basic principle is how we create **image mosaics** (colloquially, **panoramas**). To reiterate, homographies allow us to map projected points from one plane to another.

7.2.1 Creating Panoramas

Panoramas work because the camera center doesn't change as we rotate the camera; this means we don't need any knowledge about the 3D scene we're recording. A panorama can be stitched from multiple images via the following steps:

- Take a sequence of images from the same position, rotating the camera about its optical center.
- Compute a transformation between the first image and the second. This gives us a mapping between the images, showing how a particular pixel moved as we rotated, and which new pixels were introduced.
- Transform the second image to overlap with the first according to that transformation.
- Blend the images together, stitching them into one.

We can repeat this process as needed for every image in the sequence and get our panorama. We can interpret our mosaic naturally in 3D: each image is reprojection onto a common plane, and the mosaic is formed on that plane. This is demonstrated in Figure 7.3.[2]

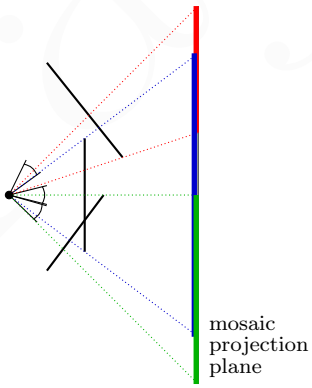


Figure 7.3: Reprojection of a series of images from the same camera center onto a "mosaic plane."

That's all well and good, but what about the math? That's what we're really interested in, right? ☺ Well, the transformation of a pixel from one projection plane to another (along a

[2] Because we're reprojecting onto a plane, we're limited by a 180° field of view; if we want a panorama of our entire surroundings, we'd need to map our mosaic of images on a different surface (say, a cylinder). The 180° limitation can be thought of as having a single camera with a *really* wide lens; obviously, it still can't see what's "behind" it.

ray, as we already showed in Figure 7.2) is just a homography:

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{7.1}$$

$$\mathbf{p}' = \mathbf{H}\mathbf{p} \tag{7.2}$$

How do we solve it? Well, there's the boring way and the cool way.

The Boring Way: Inhomogeneous Solution We can set this up as a system of linear equations with a vector $\mathbf{h}$ of 8 unknowns: $\mathbf{A}\mathbf{h} = \mathbf{b}$. Given at least 4 corresponding points between the images (the more the better),[3] we can solve for $\mathbf{h}$ using the least-squares method as described in Appendix A: $\min \|\mathbf{A}\mathbf{h} - \mathbf{b}\|^2$.

The Cool Way: Déjà Vu We've actually already seen this sort of problem before: recall the trick we used in Method 1: Singular Value Decomposition for solving the camera calibration matrix! We can apply the same principles here, finding the eigenvector of with the smallest eigenvalue in the SVD.

We create a similar system of equations as in (6.12), except in 2D. Then, we can rearrange that into an ugly matrix multiplication like in (6.17) and pull the smallest eigenvector from the SVD.

7.2.2 Homographies and 3D Planes

In this section we'll demonstrate how the 8 degrees of freedom in a homography allow us to create mappings between planes. Recall, first, the camera calibration matrix and its effect on world-space coordinates:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \simeq \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Suppose, though, that *all* of the points in the world were lying on a plane. A plane is represented by a normal vector and a point on the plane, combining into the equation: $d = ax + by + cz$. We can, of course, rearrange things to solve for z: $z = \frac{d+ax+by}{-c}$, and then plug that into our transformation!

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \simeq \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \end{bmatrix} \begin{bmatrix} x \\ y \\ {}^{d-ax-by}/_{c} \\ 1 \end{bmatrix}$$

[3] For now, we assume an existing set of correctly-mapped pixels between the images; these could be mapped by a human under forced labor, like a graduate student. Later, in the chapter on Feature Recognition, we'll see ways of identifying correspondence points automatically.

Of course, this effects the overall camera matrix. The effect of the 3rd column (which is always multiplied by x, y, and a constant) can be spread to the other columns. This gives us:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \simeq \begin{bmatrix} m'_{00} & m'_{01} & 0 & m'_{03} \\ m'_{10} & m'_{11} & 0 & m'_{13} \\ m'_{20} & m'_{21} & 0 & m'_{23} \end{bmatrix} \begin{bmatrix} x \\ y \\ {}^{d-ax-by}/_{c} \\ 1 \end{bmatrix}$$

But now the camera matrix is a 3×3 homography! This demonstrates how homographies allow us to transform (or warp) between arbitrary planes.

7.2.3 Image Rectification

Since we can transform between arbitrary planes using homographies, we can actually apply that to images to see what they would look like from a different perspective!

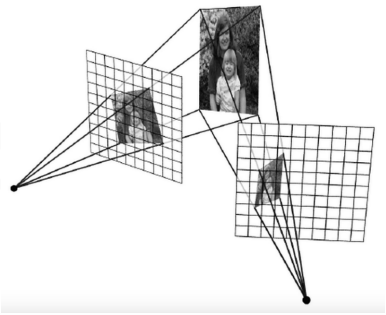
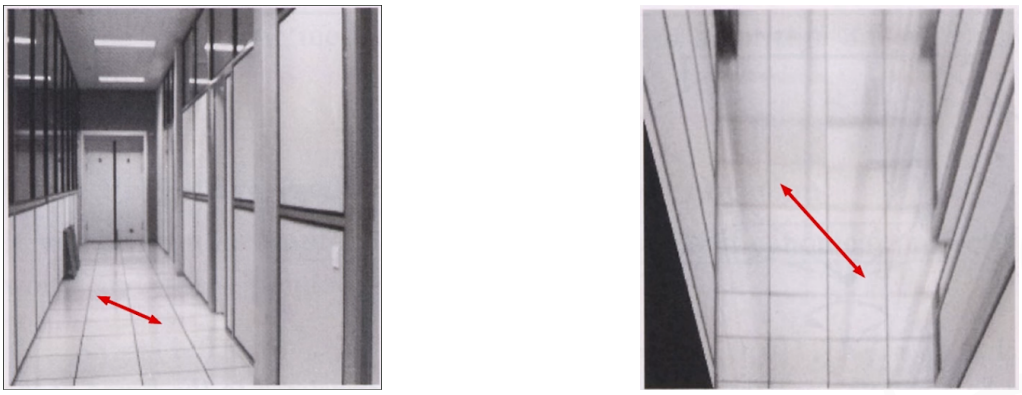


Figure 7.4: Mapping a plane (in this case, containing an image) onto two other planes (which, in this case, are the projection planes from two different cameras).

This process is called **image rectification** and enables some really cool applications. We can do things like rectify slanted views by restoring parallel lines or measure grids that are warped by perspective like in Figure 7.5.



(a) The original image. **(b)** After unwarping.

Figure 7.5: Applying unwarping to an image in order to measure the floor length.

How do we do this image warping and unwarping process? Suppose we're given a source image, $f(x, y)$ and a transformation function $T(x, y)$ that relates each point in the source image plane to another plane. How do we build the transformed image, $g(x', y')$? There are two approaches, one of which is incorrect.

Forward Warping

The naïve approach is to just pump every pixel through the transformation function and copy that intensity into the resulting (x', y'). That is, for each pixel $(x, y) \in f), (x', y') = T(x, y)$.

Unfortunately, though our images are discretized into pixels, our transformation function may not be. This means that a particular $(x', y') = T(x, y)$ may not correspond to an individual pixel! Thus, we'd need to distribute the color from the original pixel around the pixels *near* the transformed coordinate. This is known as **splatting**, and, as it sounds, doesn't result in very clean transformed images.

Inverse Warping

Instead of mapping from the source to the destination, we instead look at every pixel in the destination image and figure out where it came from in the source. Thus, we *start* with (x', y'), then find $(x, y) = T^{-1}(x', y')$.

We still have the problem of non-discrete locations, but can come up with a better solution. Now we know the neighboring pixels of our source location, and can perform **interpolation** on their intensities to get a better approximation of the intensity that belongs at the destination location.

The simplest approach would be **nearest neighbor interpolation**, which just takes the intensity of the closest pixel.

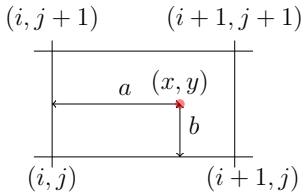


Figure 7.6: Finding the values for bilinear interpolation of (x, y).

This doesn't give great results. A better approach that is known as **bilinear interpolation**, which weighs the neighboring intensities based on the distance of the location (x, y) to each of its neighbors. Given a location, we can find its distances from a discrete pixel at (i, j): $a = x - i, b = y - j$. Then we calculate the final intensity in the destination image at (x', y'):

$$\begin{aligned} g(x', y') = &(1-a)(1-b) && f[i, j] \\ &+ a(1-b) && f[i+1, j] \\ &+ ab && f[i+1, j+1] \\ &+ (1-a)b && f[i, j+1] \end{aligned}$$

This calculation is demonstrated visually in Figure 7.6. There are more clever ways of interpolating, such as **bicubic interpolation** which uses **cubic splines**, but bilinear interpolation is mathematically simpler and still gives great results.

7.3 Projective Geometry

In this section we'll be getting much deeper into the mathematics behind **projective geometry**; specifically, we'll explore the duality of points and lines and the power of homogeneous coordinates (again!). We'll use this to build a mathematical basis *(no pun intended)* for approaching n-views in the subsequent section.

Recall, first, the geometric significance of homogeneous coordinates. In Figure 7.1 we saw that a point on the image is a ray in space, and every point along the ray is projected onto that same point in the image; we defined this as being *projectively similar*.

7.3.1 Alternative Interpretations of Lines

Homogeneous coordinates are also useful as representations of lines. Recall the standard form of the equation of a line in 2D:

$$ax + by + c = 0$$

Here, a, b, and c are all integers, and $a > 0$. From this, we can imagine a "compact" representation of the line that only uses its constants: $\begin{bmatrix} 2 & 3 & -12 \end{bmatrix}$. If we want the original equation back, we can dot this vector with $\begin{bmatrix} x & y & 1 \end{bmatrix}$:

$$\begin{bmatrix} a & b & c \end{bmatrix} \cdot \begin{bmatrix} x & y & 1 \end{bmatrix} = 0 \tag{7.3}$$

Now recall also an alternative definition of the dot product: the angle between the two vectors, θ, scaled by their magnitudes:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta$$

This indicates that the vector $\begin{bmatrix} a & b & c \end{bmatrix}$ is perpendicular to every possible point (x, y) in the $z = 1$ plane... or the ray passing through the origin and the point $(x, y, 1)$.

There's another interpretation, as well. Suppose we find a vector perpendicular to the line, as in Figure 7.7. The slope of the normal line is the reciprocal of the initial slope: $m' = -1/m$, and it passes through the origin $(0, 0)$.

In addition to this normal, we can define the line by also including a minimum distance from the origin, d. A line can be uniquely identified by this distance and normal vector; we can represent it as $\begin{bmatrix} n_x, n_y, d \end{bmatrix}$, where $\hat{\mathbf{n}} = \begin{bmatrix} n_x, n_y \end{bmatrix}$, and $d = c/\sqrt{a^2+b^2}$.[4]

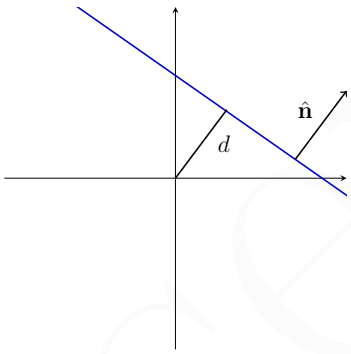


Figure 7.7: An alternative interpretation of a line.

7.3.2 Interpreting 2D Lines as 3D Points

First, we will demonstrate the relationship between lines in a 2D plane and points in 3D space under projective geometry.

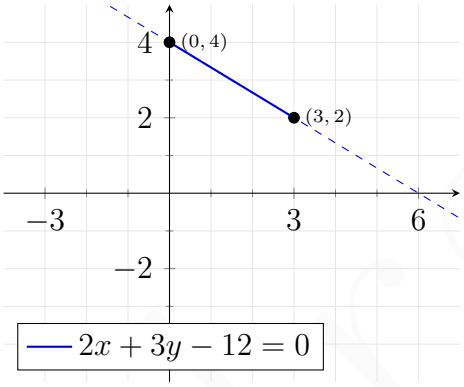


Figure 7.8: A line plotted on the standard xy-plane. Two points on the line and the segment connecting them are highlighted.

Suppose we plot some line in 2D, for example $2x + 3y - 12 = 0$ as in Figure 7.8. We can imagine a "compact" representation of the line by just using its constants: $\mathbf{m} = \begin{bmatrix} 2 & 3 & -12 \end{bmatrix}$, as we described above.

Now what if we drew this same line on a projection plane in 3D, like on the image plane at $z = 1$ in Figure 7.1? Then, suppose we visualize the rays that begin at the origin and pass through the two points on the line, $(0, 4, \mathbf{1})$ and $(3, 2, \mathbf{1})$ (notice the additional coordinate now that we're in 3-space).

The area between these rays is an infinite triangle into space, right? Overall, it's part of an entire plane! We have 3 points in space: the 2 points on the line and the origin, and three points define a plane. The intersection of that plane and the image plane *is the line* $2x + 3y - 12 = 0$, except now it's at $z = 1$. What we get is something that looks like Figure 7.9.

The most important and exciting part of this visualization is the orange normal vector. We can determine the normal by calculating the cross product between our two rays:

$$\begin{aligned} \begin{bmatrix} 0 & 4 & 1 \end{bmatrix} \times \begin{bmatrix} 3 & 2 & 1 \end{bmatrix} &= \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ 0 & 4 & 1 \\ 3 & 2 & 1 \end{vmatrix} \\ &= (4 \cdot 1 - 2 \cdot 1)\hat{\mathbf{i}} - (0 \cdot 1 - 3 \cdot 1)\hat{\mathbf{j}} + (0 \cdot 2 - 3 \cdot 4)\hat{\mathbf{k}} \\ &= \begin{bmatrix} 2 & 3 & -12 \end{bmatrix} \end{aligned}$$

[4] Wikipedia defines the minimum distance between a point and a line: see this link.

Look familiar...? That's **m**, the constants from the original line! **This means that we can represent any line on our image plane by a point in the world using its constants.** The point defines a normal vector for a plane passing through the origin that creates the line where it intersects the image plane.

Similarly, from this understanding that a line in 2-space can be represented by a vector in 3-space, we can relate points in 2-space to lines in 3-space. We've already shown this when we said that a point on the projective plane lies on a ray (or line) in world space that passes through the origin and that point.

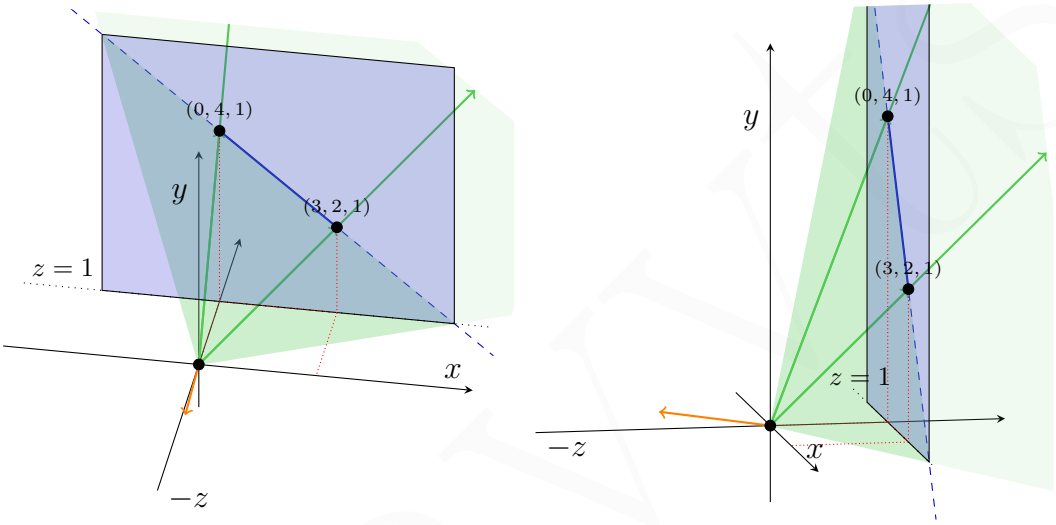


Figure 7.9: Two views of the same line from Figure 7.8 plotted on a projection plane (in blue) at $z = 1$. The green rays from the origin pass through the same points in the line, forming the green plane. As you can see, the intersection of the green plane with the blue plane create the line in question. Finally, the orange vector is the normal vector perpendicular to the green plane.

7.3.3 Interpreting 2D Points as 3D Lines

Suppose we want to find the intersection between two lines in 2-space. Turns out, it's simply their cross product! It (possibly) makes sense intuitively. If we have two lines in 2D (that are defined by a normal vector in 3D), their intersection is a point in the 2D plane, which is a ray in 3D.

Let's consider an example. We'll use $2x+3y-12 = 0$ as before, and another line, $2x-y+4 = 0$. Their intersection lies at $(0, 4, 1)$; again, we're in the $z = 1$ plane.[5] What's the cross

[5] Finding this intersection is trivial via subtraction: $4y - 16 = 0 \longrightarrow y = 4, x = 0$.

product between $\begin{bmatrix}2 & 3 & -12\end{bmatrix}$ and $\begin{bmatrix}2 & -1 & 4\end{bmatrix}$ (let's call it $\mathbf{v}$)?

$$\begin{aligned}\mathbf{v} = \begin{bmatrix}2 & 3 & -12\end{bmatrix} \times \begin{bmatrix}2 & -1 & 4\end{bmatrix} &= \begin{vmatrix}\hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ 2 & 3 & -12 \\ 2 & -1 & 4\end{vmatrix} \\ &= (3\cdot 4 - (-1\cdot -12))\hat{\mathbf{i}} - (2\cdot 4 - (2\cdot -12))\hat{\mathbf{j}} + (2\cdot -1 - (2\cdot 3))\hat{\mathbf{k}} \\ &= \begin{bmatrix}0 & -32 & -8\end{bmatrix}\end{aligned}$$

Of course, we need to put this vector along the ray into our plane at $z = 1$, which would be where $\mathbf{v}_z = 1$:

$$\begin{aligned}\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|} &= \frac{\begin{bmatrix}0 & -32 & -8\end{bmatrix}}{\sqrt{(-32)^2 + (-8)^2}} \\ &= \begin{bmatrix}0 & -4/\sqrt{17} & -1/\sqrt{17}\end{bmatrix} \\ -\sqrt{17}\hat{\mathbf{v}} &= \begin{bmatrix}0 & 4 & 1\end{bmatrix}\end{aligned}$$

Convenient, isn't it? It's our point of intersection!

This leads us to **point-line duality**: given any formula, we can switch the meanings of points and lines to get another formula.

What if we wanted to know if, for example, a point $\mathbf{p}$ existed on a line $\mathbf{l}$? Well, the line is a normal vector in 3D defining a plane, and if the point in question existed on the line, it would be *on* that plane, and thus its ray would be perpendicular to the normal. So if $\mathbf{p}\cdot\mathbf{l} = \mathbf{p}^{\mathrm{T}}\mathbf{l} = 0$, then it's on the line!

Quick Maffs: Proving the Cross Product and Intersection Equivalence

I wasn't satisfied with my "proof by example" in Interpreting 2D Points as 3D Lines, so I ventured to make it more mathematically rigorous by keeping things generic.

We begin with two lines in 2-space and their vectors:

$$\begin{aligned} ax + by + c &= 0 & \mathbf{m} &= \begin{bmatrix}a & b & c\end{bmatrix} \\ dx + ey + f &= 0 & \mathbf{n} &= \begin{bmatrix}d & e & f\end{bmatrix}\end{aligned}$$

We can find their cross product as before:

$$\begin{aligned}\mathbf{v} = \begin{bmatrix}a & b & c\end{bmatrix} \times \begin{bmatrix}d & e & f\end{bmatrix} &= \begin{vmatrix}\hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ a & b & c \\ d & e & f\end{vmatrix} \\ &= (bf - ec)\hat{\mathbf{i}} - (af - dc)\hat{\mathbf{j}} + (ae - db)\hat{\mathbf{k}}\end{aligned}$$

This is a line in 3-space, but we need it to be at $z = 1$ on the projection plane, so

we divide by the z term, giving us:

$$\left(\frac{bf-ec}{ae-db}, \frac{dc+af}{ae-db}, 1\right)$$

Our *claim* was that this is the intersection of the lines. Let's prove that by solving the system of equations directly. We start with the system in matrix form:

$$\begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -c \\ -f \end{bmatrix}$$

This is the general form $\mathbf{Mx} = \mathbf{y}$, and we solve by multiplying both sides by $\mathbf{M}^{-1}$. Thankfully we can find the inverse of a 2×2 matrix fairly easily:[a]

$$\mathbf{M}^{-1} = \frac{1}{ad-bc}\begin{bmatrix} e & -b \\ -d & a \end{bmatrix}$$

What does this expand to in our system? Well,

$$\begin{aligned} \mathbf{M}^{-1}\mathbf{Mx} &= \mathbf{M}^{-1}\mathbf{y} \\ \begin{bmatrix} x \\ y \end{bmatrix} &= \frac{1}{ad-bc}\begin{bmatrix} e & -b \\ -d & a \end{bmatrix}\begin{bmatrix} -c \\ -f \end{bmatrix} \\ &= \frac{1}{ad-bc}\begin{bmatrix} -ec+bf \\ cd-af \end{bmatrix} \\ (x, y) &= \left(\frac{bf-ec}{ad-bc}, \frac{cd-af}{ad-bc}\right) \end{aligned}$$

Look familiar? Boom, done. ■

[a] Refer to this page.

7.3.4 Ideal Points and Lines

Our understanding of a point on the image plane as being the projection of every point on a ray in space (again, as shown in Figure 7.1 and other figures) relies on the fact that the ray does in fact intersect the image plane. When isn't this true? For rays parallel to the image plane! This would be any ray confined to an xy-plane outside of $z = 1$.

We call the points that form these rays **ideal points**. If you squint hard enough, you may be able to convince yourself that these rays intersection with the image plane at infinity, so we represent them as: $\mathbf{p} \simeq (x, y, 0)$; when we go back to non-homogeneous coordinates, we divide by zero, which makes x and y blow up infinitely large.

We can extend this notion to **ideal lines** as well: consider a normal vector $\mathbf{l} \simeq (a, b, 0)$. This is actually mathematically coherent: it defines a plane perpendicular to the image plane, and the resulting line goes through the origin of the *image plane*, which we call the **principle point**.

7.3.5 Duality in 3D

We can extend this notion of point-line duality into 3D as well. Recall the equation of a plane:

$$ax + by + cz + d = 0$$

where (a, b, c) is the normal of the plane, and $d = ax_0 + by_0 + cz_0$ for some point on the plane (x_0, y_0, z_0). Well, projective points in 3D have 4 coordinates, as we've already seen when we discussed rotation and translation in Extrinsic Camera Parameters: $\begin{bmatrix} wx & wy & wz & w \end{bmatrix}$. We use this to define planes as homogeneous coordinates in 3D! A plane N is defined by a 4-vector $\begin{bmatrix} a & b & c & d \end{bmatrix}$, and so $\mathbf{N} \cdot \mathbf{p} = 0$.

7.4 Applying Projective Geometry

With the vocabulary and understanding under our belt, we can now aim to apply this geometric understanding to multiple views of a scene. We'll be returning back to stereo correspondence, so before we begin, re-familiarize yourself with the terms and concepts we established previously when discussing Epipolar Geometry.

To motivate us again, this is our scenario:

> *Given two views of a scene, from two cameras that don't necessarily have parallel optical axes (à la Figure 6.6), what is the relationship between the location of a scene point in one image and its location in the other?*

We established the epipolar constraint, which stated that a point in one image lies on the epipolar line in the other image, and vice-versa. These two points, along with the baseline vector between the cameras, created an epipolar plane.

We need to turn these geometric concepts into algebra so that we can calculate them with our computers. To do that, we need to define our terms. We have a world point, $\mathbf{X}$, and two camera origins, $\mathbf{c}_1$ and $\mathbf{c_2}$. The rays from the world point to the cameras are $\mathbf{p}_1$ and $\mathbf{p}_2$, respectively, intersecting with their image planes at $\mathbf{x}_1$ and $\mathbf{x}_2$.

7.4.1 Essential Matrix

We will assume that we have *calibrated* cameras, so we know the transformation from one origin to the other: some translation $\mathbf{t}$ and rotation $\mathbf{R}$. This means that:

$$\mathbf{x}_2 = \mathbf{R}\mathbf{x}_1 + \mathbf{t}$$

Let's cross both sides of this equation by $\mathbf{t}$, which would give us a vector normal to the pixel ray and translation vector (a.k.a. the baseline):

$$\begin{aligned} \mathbf{t} \times \mathbf{x}_2 &= \mathbf{t} \times \mathbf{R}\mathbf{x}_1 + \mathbf{t} \times \mathbf{t} \\ &= \mathbf{t} \times \mathbf{R}\mathbf{x}_1 \end{aligned}$$

Now we dot both sides with $\mathbf{x}_2$, noticing that the left side is $\mathbf{0}$ because the angle between a vector and a vector intentionally made perpendicular to that vector will always be zero!

$$\begin{aligned}\mathbf{x}_2 \cdot (\mathbf{t} \times \mathbf{x}_2) &= \mathbf{x}_2 \cdot (\mathbf{t} \times \mathbf{R}\mathbf{x}_1) \\ \mathbf{0} &= \mathbf{x}_2 \cdot (\mathbf{t} \times \mathbf{R}\mathbf{x}_1)\end{aligned}$$

Continuing, suppose we say that $\mathbf{E} = [\mathbf{t}_\times]\mathbf{R}$.[6] This means:

$$\mathbf{x}_2^{\mathrm{T}}\mathbf{E}\mathbf{x}_1 = \mathbf{0}$$

which is a really compact expression relating the two points on the two image planes. $\mathbf{E}$ is called the **essential matrix**.

Notice that $\mathbf{E}\mathbf{x}_1$ evaluates to some vector, and that vector multiplied by $\mathbf{x}_2^{\mathrm{T}}$ (or dotted with $\mathbf{x}_2$) is zero. We established earlier that this relationship is part of the point-line duality in projective geometry, so this single *point*, $\mathbf{E}\mathbf{x}_2$, defines a *line* in the other $\mathbf{c}_1$ camera frame! Sound familiar? That's the epipolar line.

We've converted the epipolar constraint into an algebraic expression! Well what if our cameras are not calibrated? Theoretically, we should be able to determine the epipolar lines if we're given enough points correlated between the two images. This will lead us to the...

7.4.2 Fundamental Matrix

Let's talk about **weak calibration**. This is the idea that we know we have two cameras that properly behave as projective cameras should, but we don't know any details about their calibration parameters (such as focal length). What we're going to do is *estimate* the epipolar geometry from a (redundant) set of point correspondences across the uncalibrated cameras.

Recall the full calibration matrix from (6.11), compactly describing something like this:

$$\begin{bmatrix} wx_{im} \\ wy_{im} \\ w \end{bmatrix} = \mathbf{K}_{\text{int}}\mathbf{\Phi}_{\text{ext}} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

For convenience sake, we'll assume that there is no skew, s. This makes the intrinsic parameter matrix $\mathbf{K}$ invertible:

$$\mathbf{K} = \begin{bmatrix} -f/s & 0 & o_x \\ 0 & -f/s_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$

[6] This introduces a different notation for the cross product, expressing it as a matrix multiplication. This is explained in further detail in Linear Algebra Primer, in Cross Product as Matrix Multiplication.

Recall, though, that the extrinsic parameter matrix is what maps points from world space to points in the camera's coordinate frame (see **??**), meaning:

$$\mathbf{p}_{im} = \mathbf{K}_{\text{int}} \underbrace{\mathbf{\Phi}_{\text{ext}} \mathbf{p}_w}_{\mathbf{p}_c}$$
$$\mathbf{p}_{im} = \mathbf{K}_{\text{int}} \mathbf{p}_c$$

Since we said that the intrinsic matrix is invertible, that also means that:

$$\mathbf{p}_c = \mathbf{K}_{\text{int}}^{-1} \mathbf{p}_{im}$$

Which tells us that we can find a *ray* through the camera and the world (since it's a homogeneous point in 2-space, and recall point-line duality) corresponding to this point. Furthermore, for two cameras, we can say:

$$\mathbf{p}_{c,\text{left}} = \mathbf{K}_{\text{int,left}}^{-1} \mathbf{p}_{im,\text{left}}$$
$$\mathbf{p}_{c,\text{right}} = \mathbf{K}_{\text{int,right}}^{-1} \mathbf{p}_{im,\text{right}}$$

Now note that we don't *know* the values of $\mathbf{K}_{\text{int}}$ for either camera, since we're working in the uncalibrated case, but we *do* know that there *are* some parameters that would calibrate them. Furthermore, there is a well-defined relationship between the left and right points in the calibrated case that we defined previously using the essential matrix. Namely,

$$\mathbf{p}_{c,\text{right}}^{\text{T}} \mathbf{E} \mathbf{p}_{c,\text{left}} = \mathbf{0}$$

Thus, via substitution, we can say that

$$\left(\mathbf{K}_{\text{int,right}}^{-1} \mathbf{p}_{im,\text{right}}\right)^{\text{T}} \mathbf{E} \left(\mathbf{K}_{\text{int,left}}^{-1} \mathbf{p}_{im,\text{left}}\right)^{\text{T}} = \mathbf{0}$$

and then use the properties of matrix multiplication[7] to rearrange this and get the **fundamental matrix**, $\mathbf{F}$:

$$\mathbf{p}_{im,\text{right}}^{\text{T}} \underbrace{\left(\left(\mathbf{K}_{\text{int,right}}^{-1}\right)^{\text{T}} \mathbf{E} \mathbf{K}_{\text{int,left}}^{-1}\right)}_{\mathbf{F}} \mathbf{p}_{im,\text{left}} = \mathbf{0}$$

This gives us a beautiful, simple expression relating the image points on the planes from both cameras:

$$\mathbf{p}_{im,\text{right}}^{\text{T}} \mathbf{F} \mathbf{p}_{im,\text{left}} = \mathbf{0}$$

Or, even more simply: $\mathbf{p}^{\text{T}} \mathbf{F} \mathbf{p}' = \mathbf{0}$. This is the **fundamental matrix constraint**, and given enough correspondences between $\mathbf{p} \to \mathbf{p}'$, we will be able to solve for $\mathbf{F}$. This matrix is very powerful in describing how the epipolar geometry works.

[7] Specifically, we use the fact that $(\mathbf{AB})^{\text{T}} = \mathbf{B}^{\text{T}} \mathbf{A}^{\text{T}}$, as shown here.

Properties of the Fundamental Matrix

Recall from Projective Geometry that when we have an $\mathbf{l}$ such that $\mathbf{p}^T\mathbf{l} = \mathbf{0}$, that $\mathbf{l}$ describes a line in the image plane. Well, the epipolar line in the p image associated with p' is defined by: $\mathbf{l} = \mathbf{F}\mathbf{p}'$; similarly, the epipolar line in the prime image is defined by: $\mathbf{l}' = \mathbf{F}^T\mathbf{p}$. This means that the fundamental matrix gives us the epipolar constraint between two images with some correspondence.

What if $\mathbf{p}'$ was on the epipolar line in the prime image for *every* point $\mathbf{p}$ in the original image? That occurs at the epipole! We can solve for that by setting $\mathbf{l} = \mathbf{0}$, meaning we can find the two epipoles via:

$$\mathbf{F}\mathbf{p}' = \mathbf{0}$$
$$\mathbf{F}^T\mathbf{p} = \mathbf{0}$$

Finally, the fundamental matrix is a 3×3 **singular** matrix. It's *singular* because it maps from homogeneous 2D points to a 1D family (which are points *or* lines under point-line duality), meaning it has a rank of 2. We will prove this in this aside and use it shortly.

The power of the fundamental matrix is that it relates the *pixel* coordinates between two views. We no longer need to know the intrinsic parameter matrix. With enough correspondence points, we can reconstruct the epipolar geometry with an *estimation* of the fundamental matrix, without knowing anything about the true intrinsic or extrinsic parameter matrix. In Figure 7.10, we can see this in action. The green lines are the estimated epipolar lines in both images derived from the green correspondence points, and we can see that points along a line in one image are also exactly along the corresponding epipolar line in the other image.

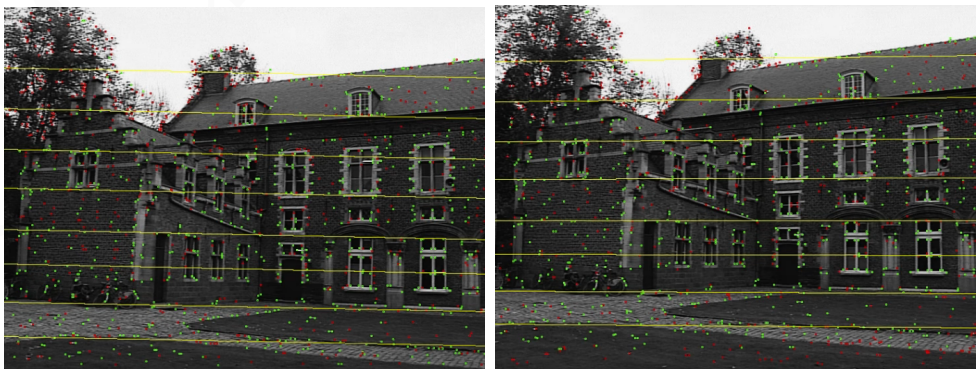


Figure 7.10: Estimation of epipolar lines given a set of correspondence points (in green).

Computing the Fundamental Matrix From Correspondences

Each point correspondence generates *one* constraint on $\mathbf{F}$. Specifically, we can say that:[8]

$$\underset{\mathbf{p}^\text{T}\mathbf{F}}{\begin{bmatrix} u' & v' & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}} \qquad \underset{\mathbf{p}'}{\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}} = \mathbf{0}$$

Multiplying this out, and generalizing it to n correspondences, gives us this massive system:

$$\begin{bmatrix} u'_1u_1 & u'_1v_1 & u'_1 & v'_1u_1 & v'_1v_1 & v'_1 & u_1 & v_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u'_nu_n & u'_nv_n & u'_n & v'_nu_n & v'_nv_n & v'_n & u_n & v_n & n \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = \mathbf{0}$$

We can solve this via the same two methods we've seen before: using the trick with singular value decomposition, or scaling to make $f_{33} = 1$ and using the least squares approximation. We explained these in full in Method 1: Singular Value Decomposition and Method 2: Inhomogeneous Solution, respectively.

Unfortunately, due to the fact that our point correspondences are estimations, this actually doesn't give amazing results. Why? Because we didn't pull rank on our matrix $\mathbf{F}$! It's a rank-2 matrix, as we demonstrate in this aside, but we didn't enforce that when solving/approximating our 3×3 matrix with correspondence points and assumed it was full rank.[9]

How can we enforce that? Well, first we solve for $\mathbf{F}$ as before via one of the two methods we described. Then, we take the SVD of *that* result, giving us: $\mathbf{F} = \mathbf{U}\mathbf{D}\mathbf{V}^\text{T}$.

The diagonal matrix is the singular values of $\mathbf{F}$, and we can enforce having only rank 2 by setting the last value (which is the smallest value, since we sort the diagonal in decreasing order by convention) to zero:

$$\mathbf{D} = \begin{bmatrix} r & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & t \end{bmatrix} \implies \hat{\mathbf{D}} = \begin{bmatrix} r & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Then we recreate a better $\hat{\mathbf{F}} = \mathbf{U}\hat{\mathbf{D}}\mathbf{V}^\text{T}$, which gives much more accurate results for our epipolar geometry.

[8] You may notice that this looks awfully similar to previous "solve given correspondences" problems we've done, such as in (6.17) when Calibrating Cameras.

[9] Get it? Because pulling rank means taking advantage of seniority to enforcing some rule or get a task done, and we didn't enforce the rank-2 constraint on $\mathbf{F}$? Heh, nice.

Proof: F is a Singular Matrix of Rank 2

We begin with two planes (the *left plane* and the *right plane*) that have some points $\mathbf{p}$ and $\mathbf{p}'$, respectively, mapping some world point $\mathbf{x}_\pi$. Suppose we know the exact homography that can make this translation: $\mathbf{p}' = \mathbf{H}_\pi \mathbf{p}$.

Then, let $\mathbf{l}'$ be the epipolar line in the right plane corresponding to $\mathbf{p}$. It passes through the epipole, $\mathbf{e}'$, as well as the correspondence point, $\mathbf{p}'$.

We know, thanks to point-line duality in projective geometry, that this line $\mathbf{l}'$ is cross product of $\mathbf{p}'$ and $\mathbf{e}'$, meaning:[a]

$$\begin{aligned}\mathbf{l}' &= \mathbf{e}' \times \mathbf{p}' \\ &= \mathbf{e}' \times \mathbf{H}_\pi \mathbf{p} \\ &= \left[\mathbf{e}'_\times\right] \mathbf{H}_\pi \mathbf{p}\end{aligned}$$

But since $\mathbf{l}'$ is the epipolar line for $\mathbf{p}$, we saw that this can be represented by the fundamental matrix: $\mathbf{l}' = \mathbf{F}\mathbf{p}$. Meaning:

$$\left[\mathbf{e}'_\times\right] \mathbf{H}_\pi = \mathbf{F}$$

That means the rank of $\mathbf{F}$ is the same as the rank of $[\mathbf{e}'_\times]$, which is **2**! *(just trust me on that part...)*

[a] As we showed in Interpreting 2D Points as 3D Lines, a line passing through two points can be defined by the cross product of those two points.

Fundamental Matrix Applications

There are many useful applications of the fundamental matrix:

- **Stereo Image Rectification**: Given two radically different views of the same object, we can use the fundamental matrix to rectify the views into a single plane! This makes epipolar lines the same across both images and reduces the dense correspondence search to a 1D scanline.
- **Photo Synth**: From a series of pictures from different views of a particular scene, we can recreate not only the various objects in the scene, but also map the locations from which those views were taken in a composite image, as demonstrated below in Figure 7.11.
- **3D Reconstruction**: Extending on the PhotoSynth concept, given enough different angles of a scene or structure, we can go so far as to recreate a *3D model* of that structure.

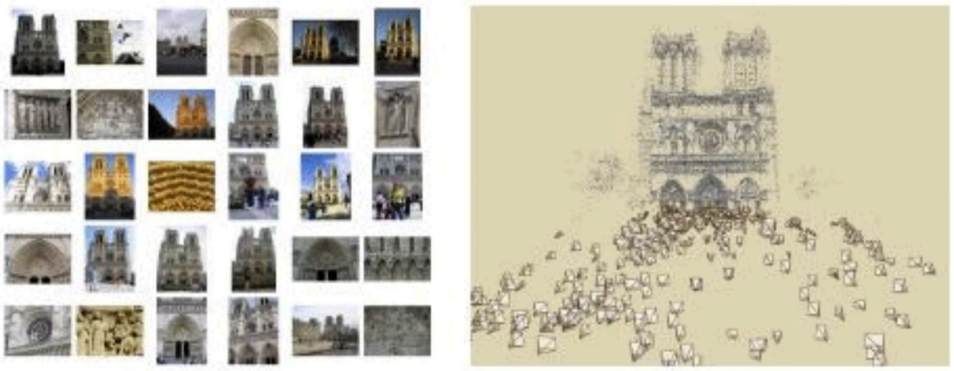


Figure 7.11: The Photosynth project (link). An assortment of photos of the Notre Dame (left) can be mapped to a bunch of locations relative to the object (right).

7.5 Summary

For 2 views into a scene (sorry, I guess we never did get into n-views ☺), there is a geometric relationship that defines the relationship between rays in one view and another view. This is Epipolar Geometry.

These relationships can be captured algebraically (and hence computed), with the essential matrix for calibrated cameras and the fundamental matrix for *un*calibrated cameras. We can find these relationships with enough point correspondences.

This is proper computer vision! We're no longer just processing images. Instead, we're getting "good stuff" about the scene: determining the actual 3D geometry from our 2D images.

Index of Terms

G

H

I

K

L

M

N

O

P

R

S

T

V

W

Y