Vison: Image Primitives, Correspondence, and Two-View Geometry

Mark Presten and Ethan Hicks

April 2020

1 Correspondence

1.1 Visualization and Correspondence

Finding the correspondence between two images of the same object means that we most find points in both the of the two images that correspond to the same 3D point of an object in view. Thus, we first must understand some factors that go into viewing a scene. One factor is the source of light, which affects how we visualize different parts of an object and contributes to the *BRDF:* Bidirectional Reflectance Distribution Function. This function tells us how light is reflected off of a surface. However, many surfaces do not reflect light the same way as they have different characteristics. Lambertian objects, or matte objects, reflect light equally in all directions. Thus, the direction of the source of light does not matter. On the other side of the spectrum is Specular objects, or reflective objects. These objects look different depending on the viewing angle and source of light. Most common day objects are a mixture of Lambertian and Specular.

Stereo cues and monocular depth cues both play a role in how we interpret scenic views. Stereo works much better at short distances and monocular depth cues help with long distance viewing. As humans, we have the capability to switch between these two cues when viewing objects, which allows us to interpret the depth of a scene and how objects lie in relation to each other.

1.2 Dealing with Correspondence Problems

We can deal with the problems of correspondence though a number of hacks, as the system is under determined.

When dealing with the Specularity Problem, we can assume that the change between the two views, $h(x)$, is simple x summed with a constant, changing the non-linear transformation into $h(x) = x + d$.

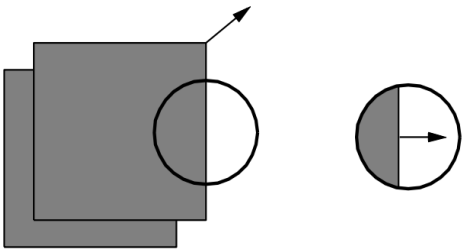
Matte objects, those that reflect light equally in all directions, do not change based on the viewing angle. When dealing with matte objects, we can assume the transformation between the two points where the image is being analyzed $h(x)$, exists as an affine transformation – that is it only preserves parallelism

and ratios between points and lines. Because we make the assumption of an affine transformation, we can define $h(x) = Ax + d$, where $A \in GL(3)$, meaning it has an inverse.

Since correspondence solves mapping parts of an image to another image, we must have a way to deal with depth discrepancies – or to simply eliminate objects that are too far away. This is solved through ambient occlusion.

1.3 Aperture Problem

The Aperture Problem arises when we are viewing a scene not in its entirety, but instead are viewing a select area through a small aperture. In the figure below, the object is moving up and to the right. However, when viewing through the aperture, the object looks as if it is moving laterally to the right, and we can not determine any vertical movement. This same illusion comes when viewing barber-pole through an aperture. In this case, the movement would seem to be solely vertical instead of lateral.



Visualization of Aperture Problem

The reason for this illusion is because of *Normal Flow*. *Normal* flow is defined as:

$$u_n \doteq \frac{\nabla I^T u}{\|\nabla I\|} \cdot \frac{\nabla I}{\|\nabla I\|} = -\frac{I_t}{\|\nabla I\|} \cdot \frac{\nabla I}{\|\nabla I\|} \tag{1}$$

Here, u_n represents is the motion of the object as viewed from the looking frame and ∇I is the image gradient. The vector u_n is not the actual motion of the object, but instead the perception of the object's motion. From the equation, we see that when solving for u_n in the context of the aperture problem, we can only recover motion in the lateral direction and NOT the vertical direction. This stems from the elimination of u, the objects motion.

2 Image Primitives

To begin to solve correspondence, we need a way to figure out how to find the details between the images so we can map the similarities to each other. Image primitives are a way to mathematically analyze images to figure out areas, points, lines, etc of interest so that we can reconstruct from the two views.

2.1 Optical Flow and Feature Tracking

Optical Flow also relates to the aperture problem. Optical Flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by relative motion between an observer and a scene. To understand Optical Flow, we take an Image I and integrate over a patch of the image. We call this integration patch of the image $E_b(u)$ and is defined as follows:

$$E_b(u) = \sum_{W(x,y)} [\nabla I(x,y,t)u(x,y) + I_t(x,y,t)]^2 \tag{2}$$

From this equation, we can solve for the gradient of $E_b(u)$ and derive the equation:

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} u + \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix} = 0 \tag{3}$$

$$Gu + b = 0 \tag{4}$$

Let's say we have two images of the same scene - one taken from a perspective on the left and one taken from a perspective on the right. This equation, $Gu + b = 0$, allows us to understand how points in the left image move in the right image. Thus, we can correlate motion in these two images. However, as mentioned previously in the aperture and barber-pole problems, if the matrix G is not of full rank, this motion can be under-defined. If the $rank(G) = 0$, we have no information and thus are most likely looking at a blank wall. If the $rank(G) = 1$, we have the aperture problem and thus can find one direction of motion, but not the other. If the $rank(G) = 2$, or in other words if the rows of G are independent and we have no singularities, we have enough texture in our current viewing aperture and can interpret how the object in this scene moves with respect to another image. Thus, if G is of full rank, we have a good feature tracking candidate and can determine how this point/section moves with respect to another image in a different perspective of the same scene.

This method of calculating optical flow holds the assumption that the local flow is constant. Other regularization techniques including locally smoothing flow fields and integration along contours can help us move towards constant flow.

Identifying key Features in an Image

As mentioned prior, when the rank of G is full, we consider the the section to be a good feature candidate as it has enough texture for us to identify what direction the object is moving when in motion. When we have enough of these sections, we can start to visualize the 3D shape of the scene. Thus, this can be very helpful in 3D modeling as we can take pictures of an object and discover its 3D shape and derive how points in the image move together.

2.2 Point Feature extraction

To begin working to solve correspondence, we must identify where the two images are similar and where the two images are different, or on the same image to identify points of mathematical interest. Given that we know I as the transformation , we can determine if two points are either similar in both images or otherwise mathematically interesting by summing them across each x and y direction.

$$G = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

Since this is a symmetric matrix, we know that a solution to this equation exists when the rows and columns are linearly independent. Once the eigenvalues are found for this, we know the point is similar if its larger than some user-defined threshold τ. Doing this for all pixels in between the two images will give us all the points in common.

2.3 Region Based Similarity Matrix

Instead of working with points, what if we work with regions instead? We still create and define a G matrix, but the I values will change from pixels to our defined regions. To get a single value from the region – to then put into our G matrix – we have a number of options to work with:

1. Sum of Squared Differences: $f_{SSD}(I_1, I_2) = \sum_{x \in W} \|I_1 - I_2\|^2$

2. Normalized Cross Correlation $f_{NCC}(I_1, I_2) = \frac{\sum_W (I_1 - \bar{I})(I_2 - \bar{I_2})}{\sqrt{\sum_W (I_1 - \bar{I})^2 (I_2 - \bar{I_2})^2}}$

3. Sum of Absolute Differences $f_{SAD}(I_1, I_2) = \sum_{x \in W} |I_1 - I_2|$

The choice is up to the user and the correct option varies on the application. We can then construct our G matrix as:

$$G = \begin{bmatrix} f(I_1, I_1) & f(I_1, I_2) \\ f(I_1, I_2) & f(I_2, I_2) \end{bmatrix}$$

Doing this for all regions in the image will give us all the points the two images have in common. As an example, we can mark the points and visually demonstrate where the algorithm tells us which points are similar, shown below.

Detecting the similar points between the two views and marking them

2.4 Edge Detection and Line fitting

Since both pixels and region similarities come down to point solving, we are limited in both what we detect and what we can use our dection for. Often times, the point detections do not produce the results desired, which is why edge detection and line fitting come into play. We can take our image and compute the derivative of it, finding the gradients. From these gradients we can find the 'falling' angles and compare them to a user defined threshold. If they are larger than the threshold, we follow the edge until the threshold is no longer met. This is Canny edge detection and it allows us to find places of contrast within the image. Once the edges are found, we can use them to find where two images of the same scene overlap. This comes from the fact that edges are matte, and thus reflect light equally in all directions. As a transformation between two images of matte is in $GL(3)$, the parallelism and proportions of the edge are kept across the transformation, allowing us to solve the correspondence problems. All we need to do to solve the correspondence problem is to construct lines from the eigenvalues of the cross-correlation matrix of the edges. We can then map these lines across different views and reconstruct the environment.

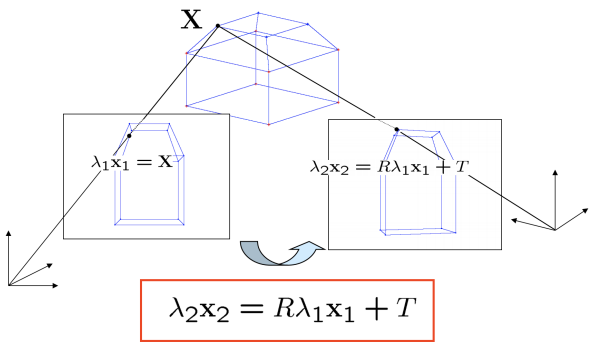
2.5 Epilogue on Image Primitives

Many of these algorithms are computationally expensive, Canny edge detection especially so. Furthermore, as mentioned above, stereo vision works best in the short distance, but images in real life can have large distances incorporated in the transformation from where the picture was taken. Given this, when dealing with real-life images we are often dealing with approximations everywhere to bring down the computational overhead and to line up the 'dirty' reality with the 'clean' mathematics that allow us to tell our computer to solve these transformations. One example of this lies with $h(x)$: often times, reality is not affine so we create an approximation to make this affine.

3 Two-View Geometry

Given two different views of the same scene, we want to recover the unknown camera displacement as well as the 3D scene structure. To do so, we use the pinhole camera model. In this model we define 3D points $X = [X, Y, Z, 1] \in \mathbb{R}^4$, image points $x = [x, y, 1]^T \in \mathbb{R}^3$, perspective projection $\lambda x = X$ where λ is depth, rigid body motion $\Pi = [R, T] \in \mathbb{R}^{3\times4}$, and rigid body motion along with perspective projection $\lambda x = \Pi X = [R, T]X$.

Using these equations and by modeling our camera as a pinhole camera, we can relate two different images of the same scene through $\lambda_2 x_2 = R\lambda_1 x_1 + T$. Lambdas here represent the depth, R represents the rigid body rotation, T represents translation between viewing points, and x corresponds to points in both images that correlate to the same 3D point. See the figure below for a visualization.



Rigid Body Motion - Two Views

3.1 3D Structure and Motion Recovery

The only knowns in the equation $\lambda_2 x_2 = R\lambda_1 x_1 + T$ are x_1 and x_2. The other variables, λ_1, λ_2, R, and T are unknown. To solve this, we use optimization techniques with the goal being to find rotation, translation, and depth such that

the re-projection error is minimized. From two separate viewing angles, we can have some 200 points for each image and would have to solve for the unknowns discussed above with this optimization in mind.

Through Epipolar Geometry we can algebraically eliminate depth from our relation equation and arrive at $x_2^T \hat{T} R x_1 = 0$. This thus eliminates two unknowns from our derived equation and allows for easier optimization with the same goal in mind that was mentioned above.

3.2 The Essential Matrix

The essential matrix is defined by certain properties of its SVD: $U, V \in SO(3)$ and

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Where $\sigma_1 \neq \sigma_2$. This exists as the definition of the essential matrix, and comes from $E = \hat{T}R$. We define this because it defines a null space: $x_2^T E x_1 = 0$. As x_1, x_2 are the camera positions, the Essential Matrix allows us to determine the position and orientation of the location of the two cameras that take the two images. The essential matrix is important in solving the reverse correspondence problem.

Due to the importance of the Essential Matrix, there are ways to estimate it so that we can recover the desired position and orientation. One such method exists as an optimization problem:

$$E = min_E \sum_{j=1}^{n} x_2^{jT} E x_1^j$$

Where the solution to this problem exists in 5 dimensions: 3 for rotation and two for translation.