Image Stitching

Neal Bayya and Alexey Didenkov

April 3, 2019

1 Introduction

Stitching images is one of the oldest and widely-used applications of computer vision. The demand for image stitching originates from the photogrammetry and film photography communities, where it was used to ease methods for manually computing distances and eliminate the need for expensive wide-angle cameras. While these earlier approaches relied on rigorous technology such as *ground-control points* or special *rotating cameras*, advancements on the computer vision side of things made these methods more convenient with time. Modern image stitching algorithms are presently used for combining satellite imagery and creating panoramas on digital cameras. Today, we will discus how to represent, align, and then combine similarities between images.

2 Motion Models

Image stitching fundamentally relies on finding and relating similarities between images. With that in mind, we need to have a good way to describe these relations in order to find them in the first place. Specifically, we must find a mathematical model that relates pixel coordinates of one image to that of another.

2.1 Planar Perspective

The simplest approach to aligning images is with 2D rotation and translation. This works fine for small motions such as when stabilizing videos, but otherwise creates visible seams. We can do better.

Recall that each camera has intrinsic parameters that can be arranged in a 3×3 matrix $\mathbf{K}$. Combined with extrinsic 3D rotation and translation parameters, we can form a 3×4 *projection matrix* that maps homogeneous 3D world coordinates to screen coordinates:

$$\mathbf{P} = \mathbf{K}[\boldsymbol{R}|\mathbf{t}], \quad \mathbf{K} = \begin{bmatrix} \mathrm{f_x} & 0 & \mathrm{c_x} \\ 0 & \mathrm{f_y} & \mathrm{c}_y \\ 0 & 0 & 1 \end{bmatrix}$$

If we extend this $\mathbf{P}$ matrix to a 4×4 $\tilde{\mathbf{P}}$, we can invert it and perform the reverse operation. Then, mapping a pixel in one image to 3D coordinates and then to another image becomes:

$$\tilde{\boldsymbol{x}}_1 \sim \tilde{\boldsymbol{P}}_1 \tilde{\boldsymbol{P}}_0^{-1} \tilde{\boldsymbol{x}}_0 = \boldsymbol{M}_{10} \tilde{\boldsymbol{x}}_0$$

In the condition that the real-world points **all lie on a plane**, we can ignore the last row and column of the 4×4 matrix $\boldsymbol{M}_{10}$, reducing it to a 3×3 $\tilde{\boldsymbol{H}}_{10}$:

$$\tilde{\boldsymbol{x}}_1 \sim \tilde{\boldsymbol{H}}_{10} \tilde{\boldsymbol{x}}_0$$

This looks awfully familiar to the 2D **perspective transform** as covered in the transformation lecture:

$$k \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

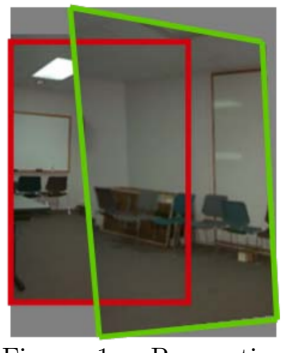

Figure 1: Perspective Transform

Although this transformation has 9 parameters, it is invariant to scale — an extra parameter k is discarded afterwards. Thus, the matrix has 8 degrees of freedom — the x- and y-coordinates of each of the four corners.

The actual h-parameters are found by solving the least squares fitting problem through robust methods such as RANSAC. Though this has been done on raw pixel values in the past, modern approaches typically find transformations based on pairing up *feature descriptors*, such as SIFT points.

Although the assumption that all points lie on a plane makes this model non-versatile, it can be used in situations such as when taking **pictures of a whiteboard** from multiple positions and angles.

2.2 Rotational

Often, such as when taking **panoramas**, the camera stays in the same location while only rotating. This is equivalent to assuming that all points are infinitely far away from the camera, i.e. lying on the *plane at infinity*. This invites us to use the reduced $\tilde{\boldsymbol{H}}_{10}$ matrix from the previous section, the upper-left 3×3 sub-matrix of

$$\boldsymbol{M}_{10} = \tilde{\boldsymbol{P}}_1 \tilde{\boldsymbol{P}}_0^{-1}$$

Since the camera does not move laterally, the extrinsic translation parameters are all zeroes. This reduces the projective matrix to $\mathbf{P} = \mathbf{KR}$ and lets us write out $\tilde{\boldsymbol{H}}_{10}$ only using 3×3 matrices:

$$\tilde{\boldsymbol{H}}_{10} = \boldsymbol{K}_1 \boldsymbol{R}_1 \boldsymbol{R}_0^{-1} \boldsymbol{K}_0^{-1} = \boldsymbol{K}_1 \boldsymbol{R}_{10} \boldsymbol{K}_0^{-1}$$

Assuming that $c_x = c_y = 0$ (pixel indexing starts from the optical center), $\mathbf{K}$ and $\mathbf{K^{-1}}$ become diagonal matrices. Further assuming that $f_x = f_y = f$ reduces the equation to

$$\begin{bmatrix} x_1 \\ y_1 \\ f_1 \end{bmatrix} \sim \boldsymbol{R}_{10} \begin{bmatrix} x_0 \\ y_0 \\ f_0 \end{bmatrix}, \quad \boldsymbol{R}_{10} = \boldsymbol{R}_1 \boldsymbol{R}_0^{-1}$$

If the focal length is constant or known, the tuning process becomes more stable as it no longer needs to find intrinsic parameters. This has the effect of reducing the degrees of freedom we have to fit on **down to 3**. In practice, tuning can be made even easier by using the small-angle approximation and finding step-wise perturbations instead.

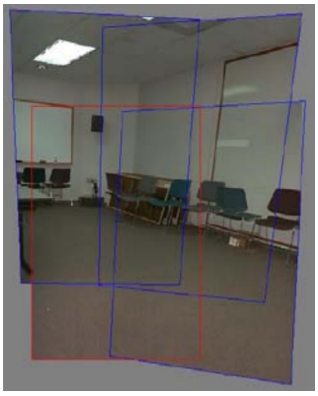

Figure 2: Rotation Model

2.2.1 Cylindrical Coordinates

Rather than depending on the motion models that we have seen insofar which have high degrees of freedom, we can warp the images in a cylindrical coordinate system and restrict our motion of alignment to translation. However, this will only work with a level camera or a known tilt angle. As you can see from Figure 3, the 3D cylindrical coordinates $(\sin\theta, h, \cos\theta)$ correspond to (x, y, f).

From this correspondence, *Szeliski and Shum 1997* derived the formula for the warped coordinates to be:

$$\begin{aligned} x' &= s\theta &&= s \tan^{-1} \frac{x}{f} \\ y' &= sh &&= s \frac{y}{\sqrt{x^2 + f^2}} \end{aligned}$$

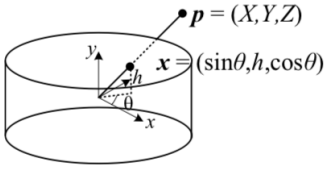

Figure 3: Cylindrical Warp

where s can be thought of as the radius of the cylinder. The selection of s actually dictates the resolution of the stitched output panorama resolution, and it is commonly set to f. The inverse mapping is given by

$$\begin{aligned} x &= f \tan\theta &&= f \tan \frac{x'}{s} \\ y &= h\sqrt{x^2 + f^2} &&= f \frac{y'}{s} \sec x's \end{aligned}$$

2.2.2 Spherical Coordinates

Images may also be projected in spherical coordinates if the camera motion includes a full sphere of of views. Figure 4 demonstrates that the spherical coordinates $(\sin\theta\cos\phi, \sin\phi, \cos\theta\cos\phi)$ corresponds to (x, y, f). The correspondence given by *Szeliski and Shum 1997* is

$$\begin{aligned} x' &= s\tan^{-1}\frac{x}{f} \\ y' &= s\tan^{-1}\frac{y}{\sqrt{x^2+f^2}} \end{aligned}$$

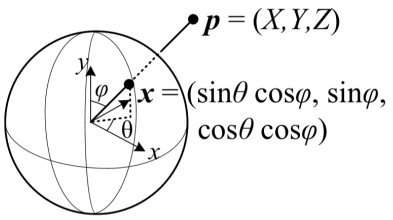

Figure 4: Spherical Warp

where s may again be thought of as the radius of the sphere. The inverse relationship is as follows:

$$\begin{aligned} x &= f\tan\frac{x'}{s} \\ y &= f\tan\frac{y'}{s}\sec\frac{x'}{s} \end{aligned}$$

In practice, spherical coordinates are used in image stitching when your camera is free to rotate around its vertical axis. Under the spherical coordinate system, images that are rotated at different angles are related with a simple translation (that is, when operating with x' and y'). Professional photographers often have camera systems which take images at specific intervals as the camera rotates around the vertical axis for precision.

3 Stitching

Our next task is to stitch the images together. Having found a relation between the coordinate domains of the two images, how exactly do we combine them into a single domain?

3.1 Composite Surface

When deciding which domain to project the stitched image onto, a simple answer would be to choose one that already exists. This means selecting a **reference** image and warping all the others around it in a **flat** panorama.

Now, imagine that you went outside and took some pictures of the night sky. If you try to combine these images naïvely you will notice weird distortions that get worse the further you go from the reference image. After all, the sky is a dome — attempting to flatten it out will severely stretch its edges.

Generally, this happens when the field of view exceeds about 90°. To fix this, one can project their panorama on a different composite surface, such as a **cylindrical**, **spherical**, or **cubic** map.

3.2 Sampling Issues

After putting pixel values through our mathematical models we get non-integer pixel locations. This means that we need to find a way to discretize them without creating visual artifacts. This can be done based on distance, or other methods such as higher-order (e.g. *cubic*) interpolators, *Gaussian averaging*, or *MIP mapping*. Also, a pre-filter is necessary to avoid aliasing in cases with resolution differences.

3.3 Pixel Selection

After placing all image points in a common transformation space, we end up with many overlapping areas and are tasked with combining information. A simple approach would be to *average* all images in areas of overlap, but this creates some problems:

- Presence of bright objects in certain images but not others can cause *exposure differences* — cameras typically darken the background for such images to compensate. Then, simple averaging will produce **seams** in places where such images have an edge.

- *Moving objects* will become translucent in what is known as **ghosting**.
- Mis-registration due to various alignment errors will cause **blurring**.

More complex weighting methods have been developed to deal with these problems:

- Ghosts can be removed by taking the *median* instead.
- Seams at image edges can be blended by weighing pixels more if they are closer to their original image's center. This is called **feathering**.
  - Raising distances to a higher power will make transitions sharper, much like in a *p-norm*. This lets us vary the amount of blurring we do, providing a trade-off between solving misalignment and seams.
  - It is favorable to blend more where images agree and less where they don't. Certain algorithms automatically vary the amount of blending based on how closely the overlapping regions align.
- *Minimum likelihood* can be used if retaining moving objects is desirable.
- A better quality blend can be achieved with *Laplacian pyramid blending*.

4 Global Alignment

So far, we have focused mainly on combining pairs of images. However, combinations typically use a large number of source images, and can be made better by taking them all into account before deciding how to stitch.

4.1 Bundle Adjustment

The simple solution is to keep adding images to the panorama in the order that they are given. As previously mentioned, in flat panoramas images will be distorted if the reference image is not chosen right. In 360° panoramas, accumulating errors can cause massive gaps or overlaps upon attempting to connect the ends.

The solution is to adjust alignment while minimizing errors for the collection as a whole, also called **bundle adjustment**. This is similar to tuning transformation parameters for pairs of images, but now it is done for all images at once.

Further improvements can be made — frequent features can be prioritized less to avoid being overweighted, modifying the loss to compute against 3D point positions can speed up the process by making matrices sparse.

4.2 Recognizing Panoramas

Sometimes, people take pictures of the same scene without realizing that a panorama could be created. In such cases, it would be useful to automatically detect panoramas from a collection of unsorted images, some of which may not belong in a panorama at all.

Similarly to bundle adjustment, overlaps between images are computed and minimized. The main difference is that edge connections can now be discarded if they exceed a certain threshold. Needless to say, this threshold is a fine balance between false positives due to similar structures and false negatives due to moving objects.

5 Other Errors

As previously mentioned, errors in circular panoramas can accumulate and result in large gaps or overlaps. Another solution to solving this is **gap closing**, an approach that relies on adjusting the focal length. We can update the focal length based on the amount of misregistration between the rotation matrices of the first and last image according to the following rule: $f' = f\,(1 - \theta_g/360)$. While this correction formula only works with a camera rotating along one dimension, there are extensions for arbitrary camera motions.

Even then, we might still find that our images look blurry or ghosted due to varying errors. **Parallax removal** aims to solve this by applying warping transformations to misaligned regions.