Building a Mosaic from Non-Overlapping Images

Benjamin Choi
Carnegie Mellon University
5000 Forbes Ave
15213 Pittsburgh, PA
benchoi@cmu.edu

Abstract

Most methods for stitching images of different parts of the same scene into a mosaic require that they overlap so that a correspondence can be determined. However, in practice, images are sometimes taken that nearly, but do not, overlap. We implement a method that extrapolates outside of image boundaries based on its content in order to align images, fill in the gaps between them, and produce a single mosaic.

1. Introduction

To construct a panorama from photos of the same scene, some overlap is usually required so that adjacent images can be aligned (using methods such as matching keypoints found in both images), and so that the resulting mosaic will not contain holes. In practice, images may sometimes be taken that do not overlap.

To stitch such images together, it is necessary to infer based on the content that is available in the images what to expect beyond their boundaries. Humans are able to do this well by observing patterns such as lines. The algorithm we implement takes advantage of the self-similarity of images by finding patches within the image that are analogous to incomplete patches on image boundaries, and uses this to extrapolate images to larger sizes.

Extrapolation from different sides of a gap creates new content that can then be aligned and used to fill in the gap, allowing a sequence of non-overlapping images to be stitched into a continuous mosaic.

2. Related Work

Methods of aligning and stitching images into a mosaic often are based on detecting keypoints in both images [8] and determining correspondances. However, this is not applicable to images wich do not overlap.

To stitch images that are immediately adjacent but do not overlap, Pomeranz et al [10] use a form of gradient-based extrapolation from each boundary pixel in their automated jigsaw solver. This is similar to what we need to do, but we need to account for gaps in between images, and we also need a way of painting inside those gaps.

To paint in gaps between images, one method is to extend contours from the boundary of the region being inpainted [2]. Another approach, which works better when the gaps involved are large, is to transplant content directly from similar images in a large dataset [6]. Exemplar-based image inpainting [3] works well for filling in large gaps, but is not directly transferable to our problem since relative alignments of the images would first have to be determined. Efros and Leung's texture synthesis [5] method is useful for image extrapolation, Poleg and Peleg [9] use a similar algorithm to extrapolate each image, then align the results. We implement an algorithm very similar to [9] for this work.

Implementing this algorithm requires searching for patches in the image similar to a given partial patch. Doing this naively by sliding a window over the entire image is computationally extremely expensive. There are a number of approaches to quickly find an approximate match. [1] is a fast approach that employs randomness for this specific problem. Patches of pixels can be generalized as high-dimensional points in feature space, and the problem reduced to finding nearest neighbors. Since kd-trees have reduced effectiveness in high-dimensional spaces, some approaches [7] first use PCA for dimensionality reduction before performing the nearest-neighbor search. Methods such as locality-sensitive hashing [4] are also able to find exact nearest neighbors in logarithmic time.

3. Algorithm

The algorithm consists of 5 steps:

1. A Gaussian pyramid is constructed
2. Extrapolation is performed from each border pixel and the results blended to create an extrapolated outer rim

Figure 1: Original images

3. The resulting image is upscaled and this process is repeated for each level of the Gaussian pyramid
4. The resulting extrapolated images are aligned
5. The aligned images are blended

3.1. Constructing a Gaussian pyramid

First, a Gaussian pyramid is constructed by iteratively applying a Gaussian blur filter and downsampling to half size.

3.2. Extrapolation from border pixels

Take the image at the lowest level of the Gaussian pyramid to be P, a $w \times h$ array of RGB pixels.

For each border pixel at location (x, y), let Ω represent the set of offsets (i, j) within a square patch of radius r (and side length $k = 2r + 1$) such that $(x + i, y + j)$ lies within the image.

$$\Omega = \{(i, j) \in \mathbb{Z} \times \mathbb{Z}, (-r \leq i, j \leq r), (x + i, y + j) \in I\}$$

A patch location (x', y') is then sought within the image at that level of the Gaussian pyramid such that $(x \pm r, y \pm r)$ is entirely within image bounds and the following cost is minimized:

$$\frac{\sum_{(i,j)\in\Omega} d(P_{x+i,y+j}, P_{x'+i,y'+j})}{|\Omega|} \tag{1}$$

where d is a pixel similarity function between RGB pixels equivalent to the Euclidean distance in RGB space.

This can be done by brute force search, but this is computationally expensive. For a fast approximation, we compute the average pixel for each $k \times k$ patch in the image (a single RGB pixel) and place it into a kd-tree. To find an approximate matching patch, we look up the m nearest neighbors in the kd-tree to the pixel which is the average of pixels in $P_{x+i,y+j}$ for $(i, j) \in \Omega$. We exhaustively search these m nearest neighbors to find the best match using equation 1.

The inverse mask Ω' is computed where

$$\Omega' = (i, j) \in \mathbb{Z} \times \mathbb{Z}, (-r \leq i, j \leq r), (i, j) \notin \Omega$$

Pixels $(x' + i, y' + j)$ for each $(i, j) \in \Omega'$ are then copied to locations $(x + i, y + j)$. At most such destination locations, pixels will be contributed from multiple patches, and the average of these pixels is taken in the result P', which is a $(w + 2r) \times (h + 2r)$ image.

Figure 2: Extrapolated image

Mask size for Gaussian blur	3
σ for Gaussian blur	1
Patch radius r	4
Patch side length k	9
Number of pyramid levels l	4

Table 1: Parameters used

3.3. Upscaling and repeating at each layer

The $(w+2r) \times (h+2r)$ image is upsampled to twice its size, and now has dimensions $(2w+4r) \times (2h+4r)$. The $2w \times 2h$ image from the matching pyramid level is transplanted to fit inside the upsampled extrapolated rim. The outer rim now has radius $2r$, and the inner r are replaced by performing the previous step for each boundary pixel (along the boundary of the $2w \times 2h$ image), but considering all $k \times k$ pixels now (not just the ones that lie inside the original image, but also the extrapolated pixels). The result is still a $(2w+4r) \times (2h+4r)$ image: the inner $2w \times 2h$ is the original image for that pyramid level, the rim of width r around it is an extrapolation, and the rim of r around that is a more blurred extrapolation (since it was not updated at this scale by extrapolating again).

At this point, the step 2 can be performed again to get an image of size $(2w+6r) \times (2h+6r)$, and then step 3 repeated. This can be continued until the top pyramid level is reached, and the result will contain the original image at the top pyramid level, but with an extrapolated rim of width $2^l r$, where l is the number of pyramid levels.

3.4. Aligning images

The resulting extrapolated images are aligned by searching for a displacement such that the sum of pairwise errors is minimized. The pairwise error between two images P and Q, of dimensions $w_P \times h_P$ and $w_Q \times h_Q$ respectively and offsets (a,b) and (c,d) respectively, is the mean distance between pixels that overlap (if there is no overlap, then the distance is ∞):

$$\frac{\sum_{(x,y)\in\Gamma} d(P_{x,y}, Q_{x+a-c,y+b-d})}{|\Gamma|}$$

where $\Gamma = \{(x,y) \in \mathbb{N} \times \mathbb{N}, 0 \le x < w_P, 0 \le y < h_P, 0 \le x+a-c < w_Q, 0 \le y+b-d < h_Q\}$

If the arrangement of the images is known, the search cost can be substantially reduced since a search can be done in the local vicinity of the initial offsets. For instance, it may be known that images P and Q are arranged horizontally, with P on the left. Then (a,b) can be initialized to $(0,0)$ and (c,d) initialized to $(w_P, 0)$, and a search performed in this vicinity. To further accelerate search, a multiscale search can be performed.

3.5. Blending aligned images

To mitigate the effect of visible seams being created where images overlap, linear blending is used. Each pixel that both images P and Q contribute to takes the value

$$(1-\alpha)p + \alpha q$$

where $\alpha = \frac{s_P}{s_P+s_Q}$, s_P is the distance to the closest pixel in P that does not overlap with Q, s_Q is the distance to the closest pixel in Q that does not overlap with P, and p and q are RGB pixels from P and Q respectively. The result is a linear blend wherever extrapolated images overlap.

4. Results

The algorithm, implemented in MATLAB, is able to extrapolate an image that is approximately 400×600 in about 2 minutes on an Intel Core i5 with a 4-level pyramid. When there are 3 such images to be aligned and mosaiced, and their ordering is known, alignment and mosaicing also takes about 2 minutes, using a 2-level search pyramid.

Figure 3: Mosaic of aligned images

5. Conclusions

We implement an algorithm that is able to extrapolate from images and align and stitch non-overlapping images into a mosaic. This has applications in creating photo panoramas.

References

[1] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics-TOG*, 28(3):24, 2009.

[2] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 417–424. ACM Press/Addison-Wesley Publishing Co., 2000.

[3] A. Criminisi, P. Pérez, and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *Image Processing, IEEE Transactions on*, 13(9):1200–1212, 2004.

[4] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[5] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1033–1038. IEEE, 1999.

[6] J. Hays and A. A. Efros. Scene completion using millions of photographs. *Communications of the ACM*, 51(10):87–94, 2008.

[7] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 327–340. ACM, 2001.

[8] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1615–1630, 2005.

[9] Y. Poleg and S. Peleg. Alignment and mosaicing of non-overlapping images. In *Computational Photography (ICCP), 2012 IEEE International Conference on*, pages 1–8. IEEE, 2012.

[10] D. Pomeranz, M. Shemesh, and O. Ben-Shahar. A fully automated greedy square jigsaw puzzle solver. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 9–16. IEEE, 2011.