Digital Image Processing

8. PATCH-MATCH
Correspondence Problem

Given two or more images of the same 3D scene, taken from different points of view, where differences are due to:

- movement of the camera (different point of view)
- N different cameras photographing at the same time
- elapse of time (different times)
- movement of objects in the photos relative to the camera(s).

?= “which parts of one image correspond to which parts of another image?”

The task of finding a set of points in one image which can be identified as the same points in another image.
Typical Application of correspondence

Image Stitching - the process of combining multiple images with overlapping fields of view to produce a segmented panorama or high-resolution image.

- Panorama Creation
- "Image Stabilization" feature in camcorders which use frame-rate image alignment.
- High resolution photo mosaics in digital maps and satellite photos.
- Medical Imaging.
- Multiple image super-resolution.
- Object Insertion
- Video Stitching

Alcatraz Island, seen in an example of a panorama created by image stitching.
Structural Image Editing

Synthesize complex texture and image structures that resembles input imagery

Image retargeting

Image reshuffling

Image completion
The Idea ...

Algorithm for rapidly finding correspondences between image patches.

Elements to be matched are image patches of fixed size

Task: what is the corresponding patch in a second image?
Previous approaches

Search similar patches elsewhere in the image.  
Fill in the missing region with their content.
The Classical PatchMatch

- The core **PatchMatch** algorithm
- Quickly finds correspondences between small square regions (or patches) of an image.

**Applications:**
- object Insertion or Object Removal
- reshuffling or moving contents of images
- retargeting or changing aspect ratios of images
- optical flow estimation
- Stereo correspondence
- Vision (Non-local means de-noising, Image forensics, Object detection)
- Videos: Tapestries, Patch-match in 3D – Temporal super-resolution

- Randomized algorithm that has a rapid convergence in practice
PatchMatch – 3 Steps Algorithm

(a) Initialization

(b) Propagation

(c) Search
Given images A and B, reconstruct image A based on correspondent points/features in B.

The result will be in an offset f.

How do we build the final image?

- Given images A and B, for each overlapping patch in **image A**
  - **We find the nearest neighbor patch** in **image B** by Approximating the nearest neighbors patch correspondences between 2 images.
  - This is done by computing the offset of all candidates to be the nearest neighbors.
  - The offset to the nearest neighbor patch in image B is assigned to be the offset is our new constricted image in the current iteration.
  - The iterations repeat until convergence.
Naïve approach

Sample every possible patch to find best match!

Complexity: \( O(mM^2) \)
Search Space

- Patch offsets vs Patches

Patch $p$ with 25 dimensions vs. Patch $p$ offset search with 2 dimensions (x and y displacements)
Key Idea #1

In searches for similar patches:

Random guesses for correspondence are likely to be wrong most of the time...
Law of large numbers

But...In a sufficiently large region, a few lucky guesses will be almost the correct correspondence.

(Law of large numbers: a non-trivial fraction of a large field of random offset assignments are likely to be good guesses).
Large numbers of guesses

M number of total pixels

Probability of correct random guess: 1/M

Probability of incorrect random guess: 1 - 1/M

Probability of all pixels with incorrect guess: \((1 - 1/M)^M\) [approximately 0.37]

⇒ Probability of at least 1 pixel with correct guess: 1 - \((1 - 1/M)^M\)

⇒ Probability of at least 1 pixel with good enough guess: 1 - \((1 - C/M)^M\)

Large number of random sampling will yield some good guesses!
(a) Initialization

Based on key Idea #1, begin with a random initial guess correspondences for every patch from image A on image B:

Each pixel is given a random patch offset as initialization.
Key Idea #1 Usage

- Initialize f with random values

Visualization of f:

Saturation = magnitude of match offset
(gray is matching patch in B at same pixel location as match patch in A)

Hue = direction of offset
offset X = red-cyan axis
offset Y = blue-yellow axis

(Magnitude)

(Angle)

When initializing with random offsets, we use independent uniform samples across the full range of image. This algorithm avoids using an initial guess from the previous level of the pyramid because in this way the algorithm can avoid being trapped in local minima.

Image credit: [Barnes et al. 2009]
Key Idea #2 – Spatial Coherence

- Because we use natural images there is a high coherence of nearest neighbors.
- Neighboring pixels have coherent matches!
- Nearest neighbor of patch at (x,y) should be a strong hint for where to find nearest neighbor of patch at (x+1,y)
Key Idea #2

Once we do find a good guess for a completing patch, it’s likely that many nearby patches have similar correspondences.
Finding the nearest neighbor (NNF Estimate)

- Ignoring boundary conditions, each MxN image consists of MxN patches, when each patch is defined by its center pixel with hue and saturation parameters.

- In the initial offset field saturation is the magnitude is and hue is the angle.

- The reconstruction is done by patch “voting” - each patch looks up its nearest neighbor’s patches values (hue and saturation), and these are averaged for all overlapping patches.

- The result is computed in the nearest neighbor field (NNF):
  
  - Nearest Neighbor Field is function $f: A \rightarrow \mathbb{R}^2$ (maps patches in A to patches in B)

  - Example: if patch b in B is NN of patch a in A, then $f(a) = b$
Propagation to improve NNF estimate

- The NNF estimate provides a “best-so-far” NN for each patch in A
  - NN patch: \( f(a) \)
  - NN distance = \( d(a,b) \) (where \( b=f(a) \))

- We try to improve NNF estimate by exploiting spatial coherence with left and top neighbor (candidate matches):
  - Let \( a=(x,y) \), then candidate matches for \( a \) are:
    - \( f(x-1, y) + (1,0) \) => Calculate \( d(a, f(x-1,y)+(1,0)) \)
    - \( f(x, y-1) + (0,1) \) => Calculate \( d(a, f(x,y-1)+(0,1)) \)

- If candidate patch is better match than \( f(a) \), then replace \( f(a) \) with candidate
  - Replace \( f(a) \) with candidate patch if \( d(a, f(x,y-1)+(0,1)) < d(a, f(a)) \)

- In next iteration, use bottom and right neighbors as candidates
Propagation

Alternate between

1. propagating good correspondences to neighboring patches

Each pixels checks if the offsets from neighboring patches give a better matching patch. If so, adopt neighbor’s patch offset.

$$\arg\min_{(x,y)}\{D(f(x,y)), D(f(x+1,y)), D(f(x,y+1))\}$$
Optimization: enrichment

- Propagation step propagates good matches across spatial dimensions of image
- Can also propagate good matches across space of matches itself
- The Idea: if \( f(a) = b \), and \( f(b) = c \), then \( c \) is a good candidate match for \( a \)
  - If you think of the NNF as a graph, then enrichment looks for nodes reachable in two steps
  - Note: assumes we’re searching for matches in the same image as the image we are trying to complete
Random Search – To avoid local min

- Propagation can cause PatchMatch to get stuck in local minima (where the distance no longer improve).
- So instead of systematically go on the graph to find the next candidates, we use another way to find the candidates, and sample a random sequence of candidates from an exponential distribution:
  - Let \( a=(x,y) \), then candidate matches for \( a \) are: \((x,y) + w\alpha^i R^i\)
  - \( w \) is maximum search radius (e.g. width of entire image)
  - \( \alpha \) is typically 1/2
  - \( R^i \) is uniform random offset in \([-1,1]x[-1,1]\)
  - Check all candidates where \( w\alpha^i \geq 1 \)
Iteratively improve results

2. Sampling the nearby image space to find even better correspondences

- Each pixel searches for better patch offsets within a concentric radius around the current offset.
- The search radius starts with the size of the image and is halved each time until it is 1.

Let $v_0 = f(x,y)$, we attempt to improve $f(x,y)$ by testing a sequence of candidate offsets at an exponentially decreasing distance from $v_0$

$$u_i = v_0 + w\alpha^i R^i$$
Algorithm

1. Initialize pixels with random patch offsets
2. Check if neighbors have better patch offsets
3. Search in concentric radius around the current offset for better patch offsets
4. Go to Step 2 until converge.

Complexity: O(m*MlogM)
Summary

- Although the initialization is completely random, a few lucky guesses propagate quickly to neighboring pixels in the propagation phase.
- Then the ransom sampling phase complements this greedy approach in order not to get stuck in a local minima, redirecting the sampling to other areas in the image.
Example: Reconstruct image A using patches from image B
1st pass
First Pass

Image A

Image B

Correspondence Vectors
(hue: angle, saturation: magnitude)

Reconstruction of image A using patches from image B
Fifth Pass (converged)

Image A

Image B

Correspondence Vectors (hue: angle, saturation: magnitude)

Reconstruction of image A using patches from image B
The whole process
Application Examples - Editing the same image using high level constraints

Image Completion/ Inpainting
(a) input  (b) hole and guides  (c) completion result
Object Manipulation

Building segment marked by user

Building scaled up, preserving texture

And retargeting (smaller/ bigger/ stretched and etc...)
Image retargeting (changing aspect ratio)

Original image (with user-provided search constraints)
Retargeted (without constraints)
Retargeted (with constraints)
Drag and drop decomposition of an image (reshuffling)
Other aspects

- Computationally Propagation step is inherently serial, but good parallel approximations exist
  - PatchMatch has been implemented efficiently on GPUs

- Data access caches well, but is unpredictable
Thank you!