PatchMatch

Notes

Uses a new randomised algorithm for quickly finding approximate nearest neighbour matches between image patches. It performs (20-100)x better than the previous state of the art - enabling interactive tools in image editing e.t.c. It also optimises the synthesis process by constraints - which offers the users a level of control that has not been available previously.

Key insight is that some good patch matches can be found via random sampling and that natural coherence in the imagery allows us to propagate such matches quickly to surrounding areas.

Image re-targeting: images resized to a new aspect ratio - good to have interactive which was achieved.

Image completion: erasing an unwanted portion of the image - good that the computer automatically synthesises a fill region that plausibly matches the erased area.

Image reshuffling: grabbing a portion of the image and moving it around - the computer automatically synthesises the remainder of the image to resemble the original while respecting the newly moved regions.

This paper improves performance which enables the interactivity.

Three key components of the algorithm that improves efficiency: 1. dimensionality of the offset space - populates sparsely patches with O(M). Does not use KD-trees and dimensionality reduction. Instead it searches through a 2D space of possible patch offset, achieving greater speed and memory efficiency.

2. Natural structure of image - ??
3. The law of large numbers - one random choice of a patch assignment is bad, but a large field of random assignments is likely a good choice.

Overall computes approximate NNF using incremental updates. It begins with a guess (either based on previous knowledge or randomly) and then it iterates in two processes: (1) propagation using coherence to gain good solutions, (2) random search ?

Runtime O(mM log M), memory usage O(M) where m is the number of pixels and M the number of patches.

Shorter Refined Version

PatchMatch uses a randomised algorithm for quickly finding approximate nearest neighbour fields (ANNF) between image patches.

While previous solutions use KD-Trees with dimensionality reduction, PatchMatch instead searches through a 2D space of possible patch offset. The initial step is choosing random patches followed by 4-5 iterations containing two processes: (1) propagation applying a statistic that can be used to examine the relation between two signals or data sets, known as coherence, (2) random search in the concentric neighbourhood to seek better matches by multiple scales of random offsets. The argument is that one random choice when assigning a patch is non-optimal, however applying a large field of random assignments is likely a good choice because it populates a larger domain.

The gains are particularly performance enabling interactive image editing and sparse memory usage, resulting in runtime O(mM log M) and memory usage O(M), where m is the number of pixels and M the number of patches. The downside is that PatchMatch depends on similar images because it searches in nearby regions in few iterations, which in turn also affects its accuracy.

Coherency Sensitive Hashing

Notes

Coherency Sensitive Hashing (CSH) is inspired by the techniques of Locality Sensitive Hashing (LSH) and PatchMatch by using the hashing scheme of LSH in replacement of the random search in PatchMatch and relying on image coherency to propagate good matches.

Indexing stage applies Walsh-Hadamard kernels.

Steps: 1. Take a random line defined by a patch `a` and divide it into bins of constant width `r` and shift the division by a random offset of `b`… 2. Project the patch onto the most significant 2D Walsh-Hadamard kernels and ordered by increasing frequency. 3. Assign it a hash value, being the index of the bin it falls into.

Search stage extend the candidate patches that are considered, instead of searching within a small region. It also combines the coherence based approach of PatchMatch along with the appearance based approach of LSH.

Choosing among the candidates is done in an approximate manner where the WH kernels rejection scheme for pattern matching is beneficial.

The benefits are better performance and a higher level of accuracy.

Shorter Refined Version

Coherency Sensitive Hashing (CSH) is inspired by the techniques of Locality Sensitive Hashing (LSH) and PatchMatch. CSH uses a likewise hashing scheme

of LSH and is built on two overall stages, indexing and searching.

The indexing stage applies 2D Walsh-Hadamard kernels in which the projections of each patch in image A and B are initially computed onto the WH kernels. Subsequently the hash tables are created by the following. First, it takes a random line defined by a patch and divides it into bins of a constant width while shifting the division by a random offset. Second, the patch is projected onto the most significant 2D Walsh-Hadamard kernels. Last, a hash value is assigned, being the index of the bin it falls into.

Applying hash tables has the benefit that similar patches are likely to be hashed into the same entry.

The searching stage starts by initialising an arbitrary candidate map of ANNF. This is followed by iterations through each patch in image A where: (1) the nearest neighbour candidates of image B are found using the current ANNF and the current hash table, (2) the current ANNF mapping is updated with approximate distances.

Selecting candidates follows the appearance-based techniques of LSH and as well as the coherence-based techniques of PatchMatch. Choosing among the candidates is done in an approximate manner where the WH kernels rejection scheme for pattern matching is applied beneficially.

CSH has 46% better performance than PatchMatch and a higher level of accuracy with error rates that are 22% lower in comparison.