8-9-2013

Approximate nearest neighbors for recognition of foreground and background in images and video

Josh Allmann

Approximate Nearest Neighbors for Recognition of Foreground and Background in Images and Video

by

Josh Allmann

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master Science in Computer Science

Supervised By
Dr. Roger Gaborski

Department of Computer Science
B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

August 9, 2013

The project "Approximate Nearest Neighbors for Recognition of Foreground and Background in Images and Video" has been examined and approved by the following examination committee:

Roger Gaborski
Committee Chair

Mark Wambach
Reader

Roxanne Canosa
Observer

Abstract

Problems in image matching, saliency detection in images, and background detection in video are studied. Algorithms based on approximate nearest-neighbor matching are proposed to solve problems in these related domains. Image patches are quantized into features using a special Walsh-Hadamard transform, and put into a propagation-assisted kd-tree for indexing and search. Image saliency and background-detection algorithms are then derived by looking at patch similarity over time and space.

Acknowledgements

I would like to thank Dr. Roger Gaborski for his gentle reminders to finish this thesis, as well as his patience and willingness to work with all my crazy ideas, my parents for their more insistent reminders to finish this, and Rebecca for making an ultimatum out of it. I would never have gotten this far otherwise. And to everybody who has expressed support and encouragement while I've been working on this thesis, thank you.

Contents

1 Introduction

In this thesis, we study the related problems of image matching, saliency and background detection, and develop algorithms for each that follow a common approximate nearest-neighbor (ANN) approach. We will use the term "saliency" to refer to foreground detection in still images; that is, the problem of discriminating important objects from a single image. We will refer to "background detection" as the saliency counterpart for video; background detection is usually simplified to the problem of finding moving objects in a video, although it is often complicated by intermittent foreground motion, continuous background motion, shadows, jitter, or artifacts from the camera itself such as automatic color balancing.

1.1 This Work

The connection between saliency and background detection investigated here is that algorithms in either domain can be predicated on eliminating redundancy [1]. In saliency, this redundancy is spatial: distinct areas of the image can be inferred to contain the foreground, and hence a salient object. In video, the redundancy is temporal: areas that do not exhibit motion over time should be considered background. Of course, these assumptions are not always satisfiable, but the question is whether an algorithm can be designed, using similar premises, that is robust enough to handle a variety of realistic scenarios. Section 4 attempts to answer that question for saliency, and Section 5 for background detection.

To identify redundancy, a neighborhood-based approach is taken, comparing image patches. Depending on the problem domain, patches can be over a sliding window, partially overlapping, or block-based. To quantize these patches into tractable features, the 2-D Walsh-Hadamard transform is applied using a fast algorithm described in Section 2. Some data structure is needed to index and search these features. The kd-tree is chosen for its space-partitioning characteristics, which yields a nearest-neighbor approximation at each level of the tree. The accuracy of the kd-tree search algorithm is improved by a propagation step, which increases the quantity and accuracy of neighbors. Section 3 explains the kd-tree and its propagation algorithm.

[1] Eliminating redundancy is by no means the only way to do saliency or background detection, but it is the approach we are taking here.

1.2 General Background

A correspondence between a target image T and a source image S is the problem of finding the best match for some feature-set of T in S. One of the most influential correspondence algorithms of the last few years is PatchMatch [4]. The key contribution of PatchMatch is its propagation step that takes advantage of the similarity between neighboring patches. If a patch $p_{x,y}$ finds a good match at offset (a, b), then it is likely that $p_{x,y+1}$ will also have a good match at $(a, b + 1)$, and so on. While the original algorithm operates directly on RGB patches, the generalized version [5] is capable of using arbitrary image descriptors such as SIFT.

The applications for nearest-neighbor and correspondence algorithms are vast, even within the confines of computer vision: image matching, object recognition and categorization, content-based image retrieval, saliency, background detection, foregroud subtraction (inpainting), retargeting, error concealment, video stabilization, and more. To narrow the scope of discussion, we will only look at work that is relevant to saliency and background detection.

A common operation for image-based correspondence algorithms is hole-filling – reconstructing a missing (or damaged) region in the image, sometimes for the purposes of foreground subtraction – inferring the background in the presence of an active foreground. While hole-filling for PatchMatch and its image-based derivatives have commonly been done via the coherency/completeness method of [51], it is extremely slow [35], and alternatives tailored for video exist [18, 40]. Several algorithms make use of hole-filling to initialize the background in an active scene [43, 3], which is further elaborated in Section 5.

For video, existing optical flow algorithms do a good job at computing small offsets, which has promise for background detection when the flows are coalesced into a global displacement vector [6]. Chen et al [10] found that ANN based motion fields generally approximate the ground-truth and can be used to bootstrap a guess for optical flow, subject to refinement. For more complicated motion, feature-based image descriptors such as SIFT [31] are generally fast enough to be useful in real-time, which has been put in good use by Granados et al [18] to accommodate videos with heavy camera motion and object transformations.

Features need to be indexed for fast retrieval. Tree-based algorithms

are very popular for this, notably the kd-tree [13]. For high-dimensional trees, kd-tree has a runtime comparable to a linear search, since its accuracy depends on backtracking to check nodes along branch boundaries. This has prompted a large body of literature on improvements to tree-based search, the most recent (in the domain of image nearest-neighbors) being Propagation-Assisted KD-Trees [20] and TreeCANN [38]. Both have shown large speedups by propagating neighboring matches in the manner of PatchMatch. There are also more exotic types of trees used for ANN matching, among them TSVQ (vector quantization) [50], vp-trees (spherical partitioning), and ball trees. An overview of the latter two and comparisons to other NN algorithms is given by Kumar, et al [29].

Another method of feature lookup is with a probabilistic algorithm such as locality-sensitive hashing (LSH) [24], which has the advantage of good performance with high-dimensional features, however adjacent patches do not benefit from spatial coherence in the matching. Coherency-Sensitive Hashing (CSH) [27] fixes this by replacing the random-search step of PatchMatch with a LSH query, while retaining the neighborhood search to further propagate good matches.

2 Transforms using Gray Code Kernels

Pixels and patches need to be transformed into distinguishing features. For this we use the Walsh-Hadamard transform (WHT) which de-correlates a signal from its basis vectors, compacting them into the first few low-frequency coefficients (with frequency being defined as the number of zero crossings in the basis; see an illustration of the WHT basis vectors in Figure 2). There are numerous other transform-based operations which have been used for saliency and backgroud detection – the Fourier transform is the cornerstone of Hou and Zhang's spectral-residual saliency [23], while Olonetsky and Avidan use PCA as the feature quantization method in TreeCANN [38], and Reddy et al compare smoothness of DCT patches to estimate the background [43], and the WHT is prominent in [20, 27, 3].

Note that these methods are closely related – given enough samples, PCA basis vectors tend to resemble those of the DCT, while the DCT is simply composed of the real part of the Fourier transform, and WHT bases can be approximated from the DCT by setting positive values to 1, and

negative values to -1. Here we concentrate on the WHT due to its efficient implementation, as presented here.

2.1 Algorithm

The Gray Code Kernels (GCK) are a family of kernels that was shown by Ben-Artzi, et al [8] to be computable with just two operations per sample, per basis. We make use of the Walsh-Hadamard (WH) kernels that were derived in the original paper. The 1-D GCK kernels are defined in [8] recursively:

$$V_s^{(0)} = s, \tag{1}$$

$$V_s^{(k)} = \{[v_s^{(k-1)} \alpha_k v_s^{(k-1)}]\} \; st. \; v_s^{(k-1)} \in V_s^{(k-1)}, \; \alpha_k \in \{+1, \text{-}1\} \tag{2}$$

where $\alpha_k v$ indicates the multiplication of the vector v by α_k and [...] indicates concatenation. For our purposes, $s = [1]$ which generates the WH basis set in V.

Now, the sequence $\alpha = (\alpha_1 ... \alpha_k)$ uniquely defines a kernel $v \in V_s^{(k)}$, referred to as the α-index. Two kernels $v_+, \; v_-$ are α-related iff the hamming distance of their α-indices is one. Hence the two kernels have the form $(\alpha_1 ... \alpha_{r-1}, +1, ..., \alpha_k)$ and $(\alpha_1 ... \alpha_{r-1}, -1, ... \alpha_k)$. Define their sum v_p and difference v_m:

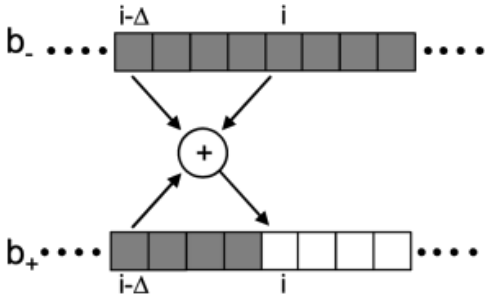


Figure 1: Calculating b_+ given b_- (from [8])

$$
\begin{aligned}
v_p &= v_+ + v_- \\
v_m &= v_+ - v_- \\
&\ldots \\
[0_\Delta v_p] &= [v_m 0_\Delta] \qquad (3) \\
v_p(i-\Delta) &= v_m(i) \qquad (4) \\
&\ldots \\
v_+(i) &= +v_+(i-\Delta) + v_-(i) + v_-(i-\Delta) \\
v_-(i) &= -v_-(i-\Delta) + v_+(i) - v_+(i-\Delta) \\
&\ldots \\
b_+(i) &= \sum_j x(j) v_+(i-j) \qquad (5) \\
b_-(i) &= \sum_j x(j) v_-(i-j) \\
b_+(i) &= +b_+(i-\Delta) + b_-(i) + b_-(i-\Delta) \qquad (6) \\
b_-(i) &= -b_-(i-\Delta) + b_+(i) - b_+(i-\Delta)
\end{aligned}
$$

where $\Delta = 2^{r-1}|s|$ is the length of the common prefix vector between v_+ and v_-, $|s|$ the length of s and 0_Δ a zero-vector of length Δ. b_0 denotes the first result of convolving an input signal with v_0; from there, subsequent results $b_1, b_2 ... b_{|s|}$ can be calculated using only previous b-results (6), which follows from the linearity of convolution (5). For a complete derivation of all these equations, see [8].

Index	v	α-index	α-binary	α-decimal
0	s s s s s s s s	+ + +	0 0 0	0
1	s s s s -s -s -s -s	+ + -	0 0 1	1
2	s s -s -s -s -s s s	+ - -	0 1 1	3
3	s s -s -s s s -s -s	+ - +	0 1 0	2
4	s -s -s s s -s -s s	- - +	1 1 0	6
5	s -s -s s -s s s -s	- - -	1 1 1	7
6	s -s s -s -s s -s s	- + -	1 0 1	5
7	s -s s -s s -s s -s	- + +	1 0 0	4

Table 1: Mapping kernels $v \in V^{(3)}$ to various representations

2.2 α-Terms

The definitions from Section 2.1 lead to interesting properties that can be used to our advantage. First is the observation that the α-index uniquely defines a kernel, and it is the α-relation that determines the GCK filtering procedure. Because of that, there is no need to explicitly generate the basis set V for WH kernels. To exploit this, construct the binary representation of α, where + maps to 0, and - to 1:

The key relationship in Table 1 is that α-decimal is simply the Gray code representation of the index value (first column of Table 1). From there, the GCK computation of the relevant α-terms can proceed with simple bit-twiddling operations as found in Section 2.4.

2.3 b-Terms

Recall the term $v_p(i - \Delta) = v_m(i)$ from Equation (4). In the signals sense, the system $y(t) = x(t - \tau)$ indicates a time delay – that is, the output at $t = \tau$ is the input at $t = 0$ [26]. In practice, that means the current position $t = i$ gets its result after Δ iterations: it induces a dependency on the future. Another way to see this is by re-writing Equation (3) as $[v_m 0_\Delta] = [0_\Delta v_p]$: now, to set $v_m(t = i)$, we would have to "look ahead" in v_p by Δ. While we could also "shift" our calculations and treat $t = i - \Delta$, it makes the implementation more difficult to reason about. So instead of delaying the signal, advance it:

$$
\begin{aligned}
b_p(i-\Delta) &= b_m(i) \\
b_p(i) &= b_m(i+\Delta) \\
b_+(i) + b_-(i) &= b_+(i+\Delta) - b_-(i+\Delta) \\
b_+(i) &= b_+(i+\Delta) - b_-(i+\Delta) - b_-(i) \\
b_-(i) &= b_+(i+\Delta) - b_-(i+\Delta) - b_+(i)
\end{aligned}
$$

This allows us to use WH results previously computed Δ iterations ago, with the element at $t = i$ being computed in just two subtractions per pixel, per base. [2].

2.4 Implementation

From our derivation of the α-terms and the b-terms, the implementation becomes clear. The 1-D case is straightforward as shown in Algorithm 1 which computes $b(0)$, $b(1)$, ..., $b(w)$ in sequency order for an output of length w, following Table 1.

[2]The notation may be entirely un-intuitive, and indeed, in our implementation, the signs are flipped; $i + \Delta$ translates to $y[i] = x[i - delta]$

```
int* gck1d(int *data, int data_len, int kern_len, int bases)
{
    int i, j, w = data_len + kern_len - 1;
    int res[len*bases], *cur = res, *prev;
    int bits = log2(kern_len) - 1; //assumes kern_len = 2^k

    calculate_dc(data, cur, data_len, kern_len); // first kernel
    for (int i = 1; i < bases; i++) {
        int alpha = gray_code(i);
        int prefix = prefix_len(gray_code(i-1), alpha, bits);
        int delta = 1 << prefix;
        int sign = (alpha >> (bits - prefix)) & 1; // determine + or -
        prev = cur;
        cur += len;
        for (int j = 0; j < delta; j++) cur[j] = -prev[j]; // zero-pad
        for (; j < w; j++) {
            if (sign) cur[j] = prev[j - delta] - cur[j - delta] - prev[j];
            else cur[j] = cur[j - delta] - prev[j - delta] - prev[j];
        }
    }
    return res;
}
```

Algorithm 1: 1-D GCK

The 2-D case takes a bit more thought. First, an α-related ordering must be defined for 2-D kernels; we produce a path similar to Figure 2. Adjacent kernels (whether from top or left) are α-related, and are re-used (with a preference for kernels from the left). When re-using a result from the left, the GCK computation is run against the α-related result from the top (and vice versa). In this way, each kernel result can be computed with a single pass through the image.

The horizontal case is simply the 1-D transform, where $b(0)$, $b(1)$, ..., $b(w)$ is computed, and repeated for each row of length w. The vertical case attempts to operate on contiguous blocks as much as possible to preserve locality, so the work is transposed: $b(i)$ is processed each pixel of row i before moving to $b(i+1)$ at the next row, and so on.

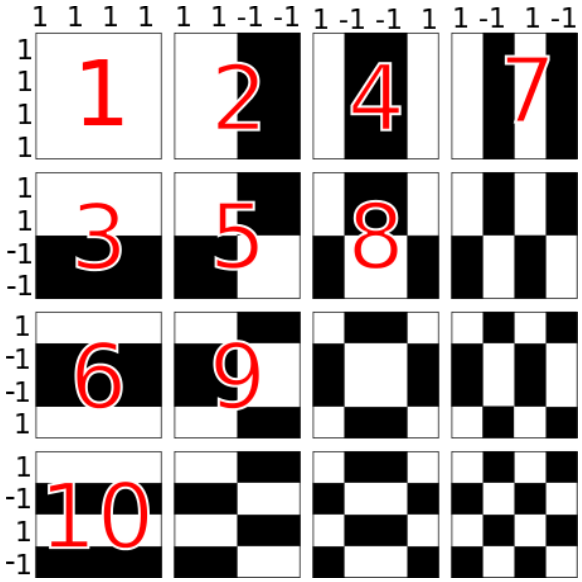


Figure 2: 2D traversal path. Adjacent kernels are α-related.

2.5 Analysis

Asymptotically, the GCK transform is $O(n)$, per pixel, with respect to the number of bases, while the sliding-window Fast Walsh-Hadamard Transform (FWHT) is $O(nlog_2(n))$[3], per pixel, with respect to the kernel size [46]. This is a crucial distinction: the run-time for GCK is the same no matter how large the kernel is; the asymptotic bottleneck of the FWHT has been completely eliminated. Additionally, there is no opportunity to compute a partial set of coefficients with the FWHT; all n coefficients will be generated, whether they are needed or not. Runtime for GCK depends directly on how many coefficients are needed, which is a major win.

As seen from Algorithms 1, 2, 3, the GCK implementation for WH kernels consists of simple arithmetic and bit-twiddling operatons, in a single pass over the signal (for each base). Since the core of the main loop is just two subtractions, GCK is amenable to vectorization, will speed up further. From tests, each base after the first takes 12ms to compute for an 1080p image on an Intel i7-2620M at 2.7GHz, regardless of kernel size. This amortizes to 37 cycles per pixel, which could be lowered substantially, perhaps to under 10.

[3]The FWHT construction closely follows that of the Fast Fourier Transform

2.6 Supplemental Algorithms

```
int gray_code(int a)
{
    return (a >> 1) ^ a;
}
```

Algorithm 2: Decimal to Gray Code conversion

```
int prefix_len(int a, int b, int bits)
{
    int count = 0, mask = 1 << bits;
    while (mask && (a & mask) == (b & mask)) {
        count += 1;
        mask >>= 1;
    }
    return count;
}
```

Algorithm 3: Length of bitwise prefix between two integers, of length *bits*

3 Propagation-Assisted KD-Tree

Once features have been generated, they need to be organized for fast retrieval. For the problem of nearest-neighbor matching, kd-trees have long been popular ([31, 37, 20, 38, 47]). kd-trees have a shortcoming that its matches are approximate, bactracking to improve accuracy, but a higher accuracy leads to linear-time search efficiency. This problem becomes especially acute with high-dimensional features [13].

Many methods have been developed to improve kd-tree search accuracy: Arya, et al backtrack in priority order depending on distance to the query [2], Beis and Lowe prune branches if the distance to a branch node is greater than the current-best match [7], Silpa-Anan and Hartley build multiple randomized kd-trees and select the best result [47], and Muja and Lowe introduce k-means trees [37].

More recently, there has been interest in propagation-based kd-tree matching due to the success of PatchMatch. Propagation-based methods take advantage of query dependency: if a good match for a patch $p_{x,y}$ is found at offset (a, b) then perhaps $p_{x+1,y+1}$ will have a good match at $(a + 1, b + 1)$.

TreeCANN, proposed by Olonetsky and Avidan [38], uses a kd-tree to find k-nearest matches on a sparse grid using PCA features and a search-cutoff factor of $1 + e, e \approx 3$. The propagation step is aided by a new construction involving the integral image. Altogether, the TreeCANN algorithm typically falls within <3% of the ground truth for a fraction of the computational time.

The object of our study, however, is the Propagation-Assisted KD-Tree as proposed by He and Sun [20], which was chosen for its simplicity and promising performance. WHT-based features are used, paired with a straightforward propagation step.

3.1 Algorithm

The kd-tree is a type of space-partitioning binary tree where nodes have tuples of dimension d [4]. The construction is follows that of a standard binary tree – for a value v at axis a ($0 \leq a < d$) where $v = Tuple[a]$: if $v \leq p$ for some pivot $p = Node\text{-}tuple[a]$, send $Tuple$ down the left branch; otherwise $Tuple$ goes right. The construction continues for the next axis at the next level. See Listing 4 a simplified algorithm. Axes need not be sequential (eg, the choice of a at one level does not imply $a + 1$ at the next level), rather the axis ordering is typically selected to maximize the balance of the tree. Axis ordering is further discussed in Section 3.4. The selection of the pivot p (and by extension, the selection of the node contents $Node\text{-}tuple$) is also extremely important to maintain balance and make tree-building performant. Pivot selection is discussed in Section 3.5. For our purposes, we also follow the lead of [20] and terminate each leaf with at most $m = 8$ tuples (m is configurable at compile-time). This automatically gives us $m - 1$ of the best match's nearest-neighbors for free, without any further refinement.

```
node* new_node(int** tuples, int nb_tuples, int depth)
{
    node *n = make_node();
    if (nb_tuples < m) {
        node->tuples    = tuples;
        node->nb_tuples = nb_tuples;
```

[4] The term 'tuple' is used rather than the more specific 'image patch' since the kd-tree operates on any d-dimensional feature vector

```
        return node;
    }
    int axis = depth % d;
    int median = find_median(tuples, nb_tuples, axis);
    n->pivot = tuples[median][axis];
    n->tuples = tuples + median;
    n->left = new_node(tuples, nb_points-median, depth+1);
    n->right = new_node(tuples+median+1, nb_tuples-median-1,
                        depth+1);
    return n;
}
```

Algorithm 4: Simplified tree-building. Note the find_median operation is destructive; it partitions tuples around the median tuple, at the given axis.

Querying is likewise similar: at each level, the query and the node's tuple are compared at the appropriate axis, and branching goes from there.

```
node* query_node(node* n, int* query, int depth)
{
    int axis = depth % d;
    if (!n->left && !n->right) return n;
    if (query[axis] == n->pivot &&
        tuple_matches(n->tuples[0], query))
        return n;
    if (n->left && query[axis] <= n->pivot)
        query_node(n->left, k, query, depth + 1);
    if (n->right && query[axis] > n->pivot)
        query_node(n->right, k, query, depth + 1);
}
```

Algorithm 5: Simplified search.

3.2 Differences from original Propagation-Assisted KD-Tree

While this work is very similar to [20], there are several differences:

- The tree is 16-dimensional ($d = 16$) composed of tuples constructed from the Walsh-Hadamard (WHT) transform in RGB space, truncated

to 2-9-5 coefficients, rather than 16-4-4 in YUV space as was done in the original paper. Changing the colorspace and reducing dimensionality decreases error by 4% and runtime by 23% on the VidPairs data set. Section 3.6 further discusses the colorspace and dimensionality selection.

- Given a target patch $P_B(x, y)$, rather than than propagating $P_A(x, y)$ from the left and top adjacencies previously found at $P_A(x-1, y)$ and $P_A(x, y-1)$ instead directly propagate the best k matches from each location at $P_A(x-1, y)$ and $P_A(x, y-1)$ (rather than the adjacencies) since this seems to lead to much more accurate results. This may be partly because all comparisons are done in WHT space, which is not completely reflective of the true RGB distance between two patches. [20] proposes re-ranking to address this problem, but re-ranking was not implemented here. It is also possible there is an error in this implementation leading to degraded results with the original algorithm.

3.3 Test Setup

Testing was done on 126 pairs of JPEGs of the VidPairs data set [28]. Each pair of images is taken publicly available movie trailers, several frames apart. For each pixel in the destination image, patches from the source image are used to determine the nearest neighbor – the best pixel in S to use for reconstructing T. The accuracy of the result is computed by $\sum |I_r - I_{gt}|$: taking the SAD of the reconstructed image and the ground-truth best match.

```
image* reconstruct_image(image* src, image* dst) {
    image *rec = new_image(dst->width, dst->height);
    for (int dst_y = 0; dst_y < dst->height; dst_y++) {
        for (int dst_x = 0; dst_x < dst->width; dst_x++) {
            int src_x, src_y;
            tuple *patch = get_patch(dst, dst_x, dst_y);
            tuple *match = query(src, patch);
            get_coords(src, patch, \&src_x, \&src_y);
            rec[dst_y][dst_x] = src[src_y][src_x];
        }
    }
    return rec;
```

```
}
```

Algorithm 6: Simplified test procedure

3.4 Coefficient Selection

After the WHT operation, the coefficients are in planar order; this must be transformed to an interleaved order (eg, $AAABBBCCC \rightarrow ABCABCABC$) before insertion. The 2-D WHT coefficients are selected in approximately sequency (frequency-increasing) order to take advantage of WHT energy compaction in the first few frequencies. This resembles the entropy-coding stage for image and video codecs where transformed coefficients are selected in a zig-zag order (see Figure 1).

After interleaving, the axis ordering for insertion from the d coefficients needs to be determined. To reduce run-time, [20] randomly samples tuples and generates an ordering going from the dimension with the largest spread (difference between min/max values) to the lowest. This random sampling is not implemented here; instead a full traversal is done. While sampling is simple enough to implement, tree balance checks haven't been implemented, which are necessary in order to verify the sampling. In fact, sampling might be entirely avoidable by simply visiting the coefficients in sequency order, which would further reduce tree-building time. However, this hypothesis still needs to be tested – again, via tree balance checking.

3.5 Pivot Selection

Quickselect is the linear-time median selection algorithm used to find an appropriate pivot value for each inner node as the tree is built. First an initial guess at the median is made (using the median-of-3 method [11]), and partitions elements around that guess, refining the guess at each iteration until convergence. A non-recursive implementation from [12] is used, modified to accommodate d-dimensional tuples, pivoting tuples around a given axis. Another convenient property of the quickselect algorithm is that it approximately sorts numbers around the median, wth values growing closer to the median towards the middle. This helps amortize the cost of repeated sub-selections, which occur on each branch as the tree is built. There is a drawback to the quickselect algorithm – while it will find the median of an unordered list of numbers, if the median is duplicated in the list, there is no

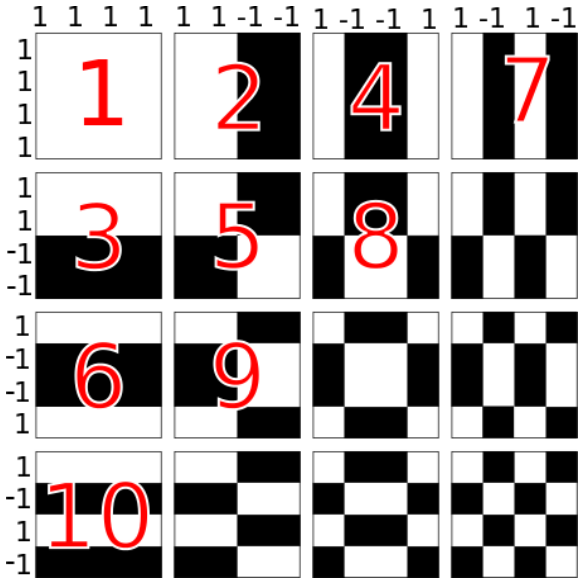


Figure 3: W-H basis vectors and coefficient selection after projection. Note the selections in order of increasing frequency. Frequency can be counted as the number of zero crossings from -1 to 1 (or from black to white)

guarantee the median will be pivoted around itself. Hence the median could be present in both partitions, leading to much lower query accuracy (and makes it impossible for an image to query itself). One solution (as was implemented here) is to take a second pass over the data and pivot around the median; this algorithm is described in [11] as the Hoare Partition. Quicksort suffers from a similar problem as described in [45], which can be mitigated by using a two-pivot variant. Something similar could be investigated here, which would avoid a second pass over the data.

3.6 Colorspace and Dimensionality Selection

Proper selection of colorspace and coefficients is important for query accuracy and time spent in tree-building. RGB was found to be faster, more accurate and require fewer coefficients. The question is – why? Non-RGB colorspaces are common for for image processing operations, including nearest-neighbor search ([20], [27]). These colorspaces typically model luminosity separately from the color components (chrominance). Looking at Figure 5, luminosity has a significant impact on perceptual quality, even with minimal chroma information. Since high-frequency coefficients after the WHT tend

Configuration	Error (% of PM)	Time (ms)
HSV-4-4-16	1.270	4101
Yuv-25-1-1	0.984	4939
Lab-19-4-4	0.995	4990
Lab-16-4-4	0.996	4569
Lab-25-1-1	1.001	5301
Yuv-19-4-4	0.979	4812
Yuv-16-4-4	0.980	4435
Yuv-10-3-3	0.981	3505
RGB-8-8-8	0.953	4268
RGB-5-9-2	0.949	3399
PatchMatch	1.000	16544

Table 2: Average runtimes, as compared to PatchMatch. The configuration scheme first names the colorspace followed by the number of coefficients allocated to each color channel.

towards zero, extra low-frequency chroma terms significantly improve the quality of matching in addition to reducing run-time slightly (Figure 2).

RGB preserves the original colorspace, with information more equally spread among the its channels, leading to more consistent fluctuations in the transformed signal. In effect, other colorspaces are *too efficient* in compacting information in the luminosity plane: paired with the WHT, there is simply not enough information to diffuse through the tuple for matching. An interesting note is the optimal set of coefficients for RGB (2-9-5) mirrors the ratio commonly used in RGB to grayscale conversion.

To emphasize how perceptually compressed non-RGB color spaces are: Figure 5 compares a portion of the Lena image by doing a 8x8 non-overlapping WHT, quantizing each color channel to the specified number of coefficients (using the method of 3.4), then taking the inverse WHT. Figure 5a retains the most detail with just one coefficient for UV. The only visible degradation compared to Figure 5b is some minor chromatic aberration along the diagonal edge, and a slightly duller color overall. On the other hand, the 24 coefficients of 5c noticeably degrade the image – even though information is distributed more evenly among RGB color channels, which should minimize the impact of quantizing any one channel. This is even more apparent in the 16-coefficient case of Figure 5d. Strikingly, while Figure 5 is arranged in order of decreasing visual quality, reconstruction quality is increasing, with 5d

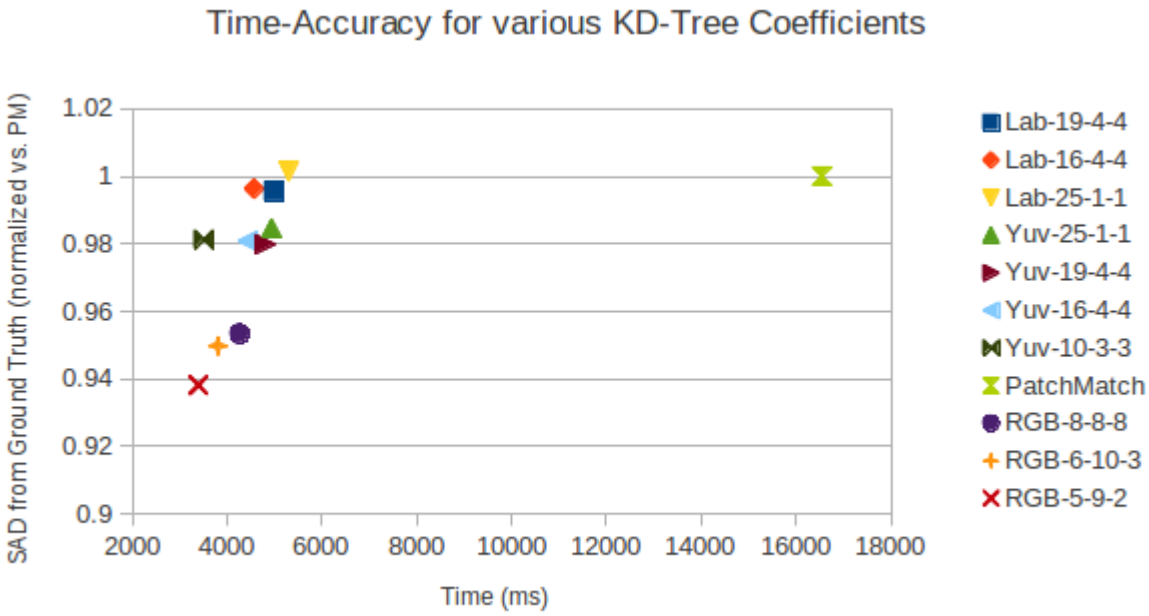


Figure 4: Average results for colorspace/coefficient combinations with VidPairs. HSV, as an outlier, is not graphed

providing by far the best parameterization for kd-tree matching (see Table 2).

3.7 PatchMatch Comparision

Note in Figure 6 there are two distinct groups of images as observed by the running time: for the kd-tree, there is a group around 3000ms and another around 4500ms, while PatchMatch has groups around 14000ms and 20000ms. This is due to variations in the data set; a few pairs of images in VidPairs are 1920x800 while most are 1920x1080, effectively a 35% increase in pixel count. Figure 7 shows the visual results of reconstruction for the propagation-assisted kd-tree, PatchMatch, and the ground truth best match. Both Figures 7a and 7b exhibit characteristic ringing around the edges, although PatchMatch's is more pronounced due to how it propagates adjacent pixels. Since this implementation of the kd-tree propagates $P_A(x-1,y)$ (or $P_A(x,y-1)$) rather than $P_A(x,y)$, there is visible banding around areas that are both uniform in color, and difficult to match (usually due to few approximations of that color in the source image). This

(a) YUV-14-1-1 (b) YUV-10-3-3 (c) RGB-8-8-8

(d) RGB-2-9-5

Figure 5: Visual comparison of quantization using various parameters, scaled 250%

occurs due to $P_B(x, y)$ 'borrowing' patches from previously matched neighbors. This is plainly visible in the background, where PatchMatch also has difficulty.

3.8 Implementation Notes

Tuples are stored in memory in raster order, and a separate array of pointers is built that points to the beginning of each tuple. The pointers are partitioned during tree-building; pointer swapping is much faster than swapping blocks of memory. Moreover, this method reduces the cache footprint of a single node (which only has to maintain pointers to its m tuples). This also has the result of speeding up tree traversal (and querying), although the final determination of the best candidate is a bit slower due to the need to access a different region of memory for each tuple in a leaf. This indirction reduces tree-building time by up to 18%.

While indirection makes queries against the tree a bit slower (at the final step; when finding the exact best match among m candidates), its impact during propagation is minimized by using previously found top and left

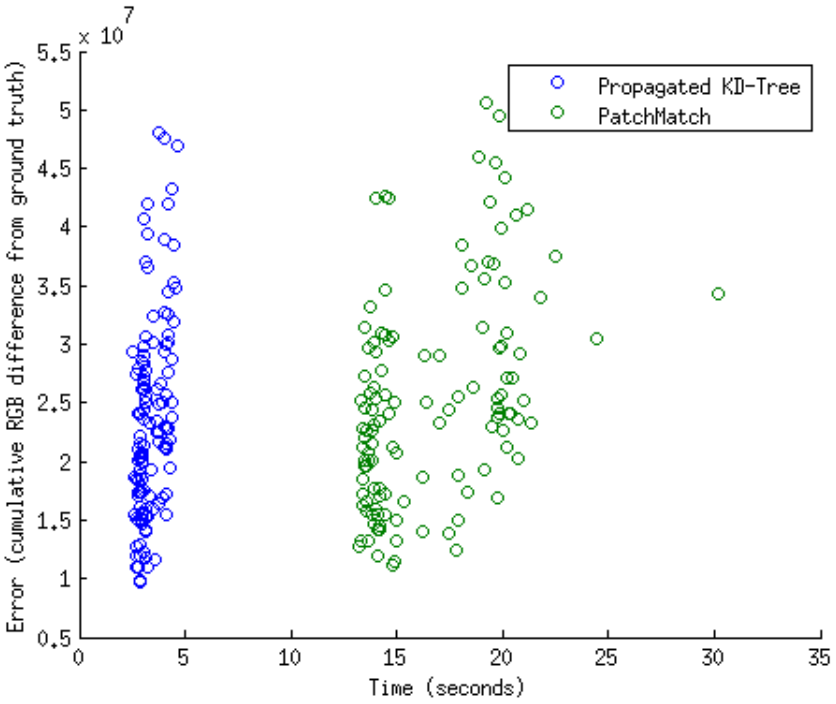
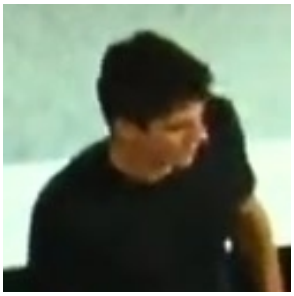


Figure 6: PatchMatch comparison for VidPairs tests

(a) KD-Tree (b) PatchMatch (c) Ground Truth



(d) Original

Figure 7: Visual comparison of PM, KD-Tree, Ground Truth and original

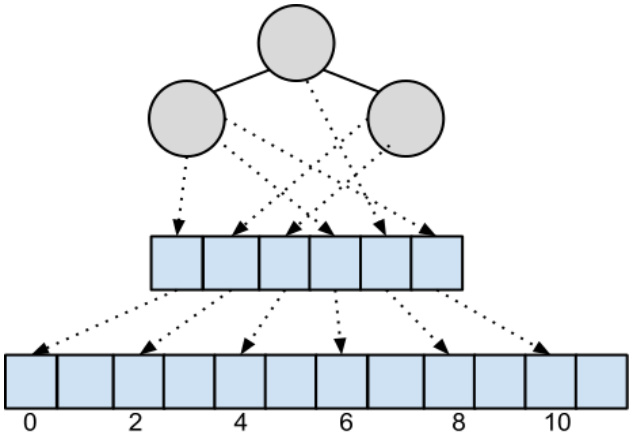


Figure 8: Layout of tuples in memory ($d = 2$, $m = 3$), with node values pointing to tuple indices.

tuples as a "guide" rather than checking against the m tuples in the node containing them. Hence, only a single row of pointers is needed as a history to track previous matches, with the current position in the history (position x) indicating the top (prior to being overwritten by the best match for the current query), and the left at position $x - 1$.

Another advantage to indirection is that it allows fast construction of partially-overlapping blocks. The results from the sliding-window WHT can be reused by simply skipping a few tuples to gain the desired amount of overlap. For example, an overlap of 50% for a kernel of size 8 would skip 4 tuples for every one that is incorporated into the final tree. This is used to good effect for saliency detection in Section 4. Indirection reduces the amount of copying necessary by simply setting pointers to the right place in the WHT data. This is useful in the saliency-detection step, where a minimum spacing between blocks is usually desirable.

By storing a pointer to the beginning of the tuple array, the exact coordinates of a tuple in the source image can be calculated given its offset from the beginning of all tuples. This can be given by:

```
coords find_xy(int *tuple, int *tuple_start) {
    w      = image_width * k
    offset = tuple - tuple_start
    x      = offset % w
    y      = offset / w
```

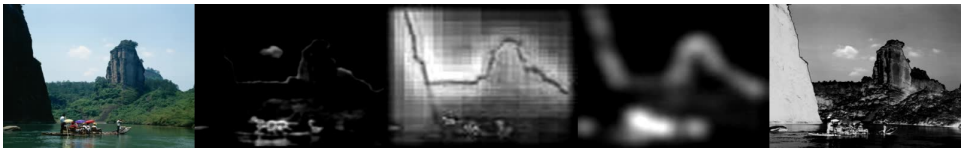


Figure 9: Saliency detection results on a particularly difficult image. Left to right: Original image, our approach, [25], [23], [1]

```
    return make_coordinates(x, y);
}
```

Algorithm 7: Finding x, y coordinates given tuple address

Calculating the original (x, y) coordinates of the tuple is useful primarily for determining its location in the source image. This information is used to reconstruct the target image using the pixel value at (x, y). Internally, a map also exists that points to the containing node for each (x, y) position. The map is assigned during tree-building, and can be used as an index into the tree (or node) during propagation, but is currently unused.

Since GCK zero-pads data, we only use patches that fall within the image; so we end up with (width - 8 + 1)*(height - 8 + 1) patches. This leads to reconstruction artifacts along bottom-right borders, which are ignored during error calculation.

4 Saliency Detection using Nearest-Neighbors

In "Context-Aware Saliency Detection," Goferman et al. [16] proposes an algorithm to calculate the saliency of an object and its surrounding context by computing the distance (in color and space) to the nearest neighbors of an image patch. Salient objects would be distant from its neighbors, in similarity and space. A related approach is taken here, although details differ – primarily in the addition of a tempering step (and omitting the corresponding "attendance" formula from [16]). Additionally, in a follow-up paper, Goferman et al [15] laments the slowness of CPU-based approximate nearest-neighbor algorithms; fortunately, our implementation from Section 3 is quite fast, so it is re-used for nearest-neighbor search.

In review, since each kd-tree leaf contains up to 8 patches, a single query down the kd-tree yields up to 8 neighbors. By propagating the best two nodes from the top and left (found during earlier queries), up to 40 neighbors

can be compared – 8 for the current patch, and 16 each from the top and left. Comparing 40 patches is computationally expensive, but fewer patches lead to lower accuracy. Figure 12a shows results for 40-neighbor (propagated) and 8-neighbor (non-propagated) comparisons.

Section 4.1 describes the algorithm used, while Section 4.2 justifies the parametric choices made with regards to propagation and overlap. Quantitative comparisons to related work are made in Section 4.3, and weaknesses discussed in 4.4.

4.1 Algorithm

First we can begin by defining the distance formula, which incorpoates color and positional information from the kd-tree query:

$$d(p_i, p_j) = \frac{d_{color}(p_i, p_j)}{1 + c \cdot d_{position}(p_i, p_j)} \tag{7}$$

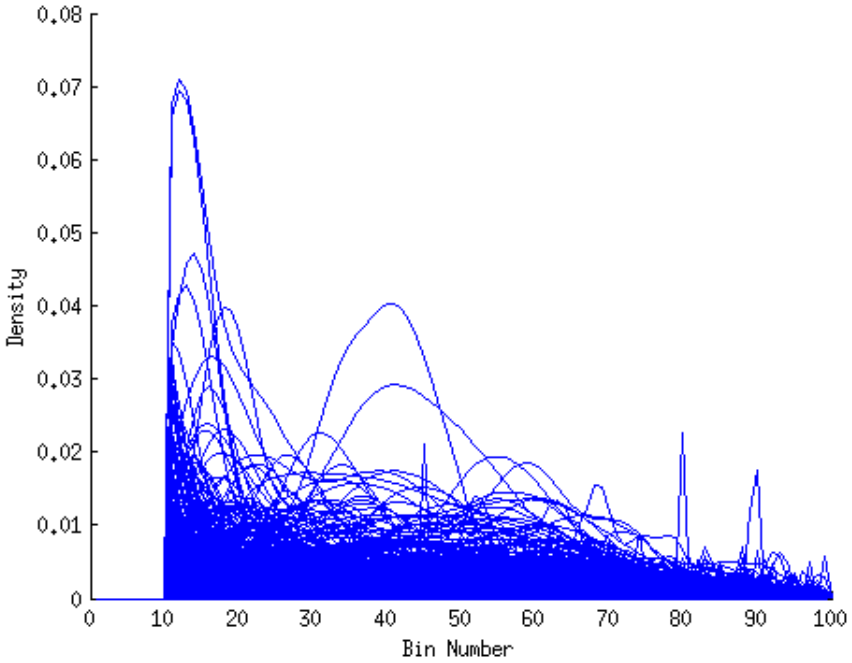
c is the dimensionality of the patch vector, here $c = 16$ following the kd-tree construction outlined in Section 3.1. d_{color} is the Euclidean distance (in quantized RGB-WHT space) between patches p_i and p_j, while $d_{position}$ indicates the Euclidean distance between the two patches.

We repeat the distance calculation for each patch p_i at several scales $r \in R = \{1, 0.8, 0.5, 0.25\}$, and take the result $\bar{S}_i$ to be the average of all scales for K neighbors, with a scaling factor:

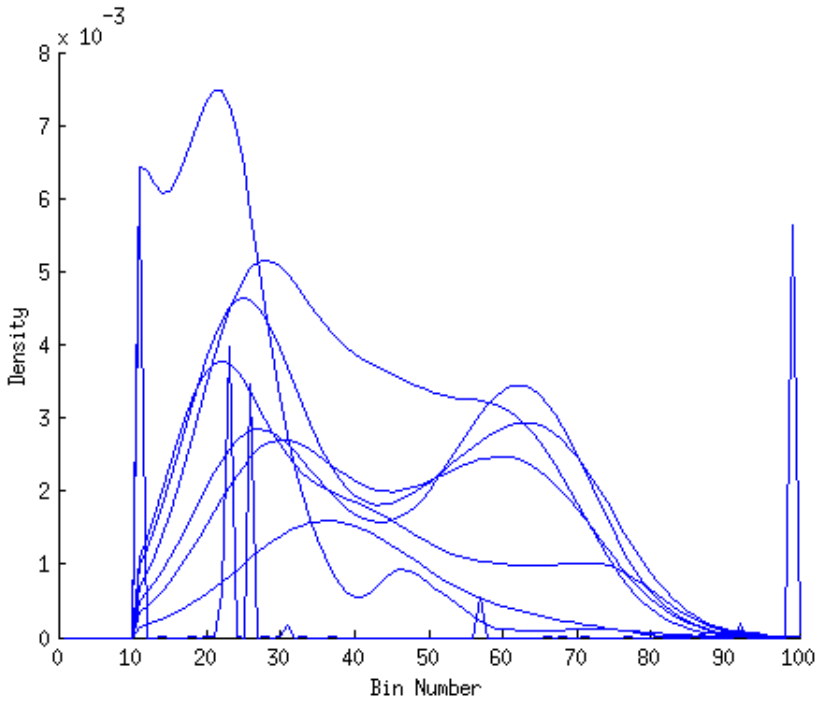
$$S_i^r = 1 - exp\left\{1 - \frac{1}{K}\sum_{k=1}^{K} d(p_i^r, p_k^r)\right\} \tag{8}$$

$$\bar{S}_i = \frac{1}{|R|}\sum_{r \in R}^{R} S_i^r \tag{9}$$

Usually by this point the map is noisy to various degrees (Figure 11b). To temper it, we apply an iterative thresholding function, and generate a normalized distance map $\hat{S}$ for each zero pixel to all nonzero pixels within a radius $r = 32$. A single threshold does not work well for all images across a dataset, and neither do typical adaptive methoods. Not only is the saliency map for an image typically multimodal (Figure 10b), in aggregate they tend towards uniformity (Figure 10a), making it difficult to use a fixed or adaptive threshold.



(a) Smoothed saliency histograms for all of Dataset 1, which tend towards uniformity.



(b) Smoothed saliency histograms for the first 10 images of Dataset 1, which are multi-modal.

Figure 10: Histograms of saliency maps

This tempering step is expensive – but without it, the result looks like Figure 11b, with an abundance of false positives. An alternative formulation is also considered, as shown in Figure 11e: the saliency map with a much faster, simplified tempering step. Simplified tempering takes the mean of 10 thresholds of $\bar{S}$ between 0 and 255. However, this is not of much help for reasons explained in Section 4.2. The full tempering map, $\hat{S}$, works as follows:

$$d_{thresh}(p_i, t) = \sum_{p_j \in \bar{S}(p) > t} d_{position}(p_i, p_j) \tag{10}$$

$$dist(p) = \frac{1}{|T|} \sum_{t \in T}^{T} d_{thresh}(p, t) \tag{11}$$

$$count(p) = \sum_{t \in T}^{T} |\bar{S}(p) > t| \tag{12}$$

$$\hat{S}_i = \frac{dist(p_i) \cdot count(p_i)}{r^2 \sqrt{2r^2}} \tag{13}$$

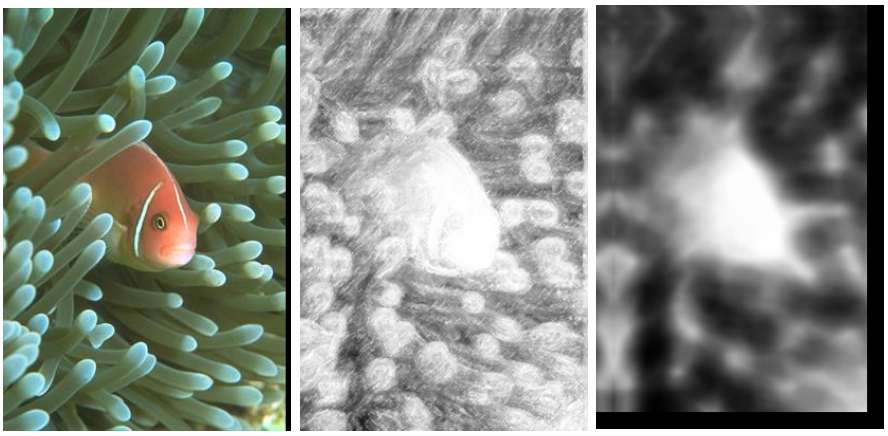
$$S = \bar{S} \cdot \hat{S} \tag{14}$$

where $\bar{S}(p)$ returns the set of pixels surrounding p within radius r, in $\bar{S}$. To justify the formulation of $\hat{S}$: $dist \cdot count$ rewards a pixel if its patch contains multiple nearby nonzero pixels. This helps to reduce the weight of sparse patches, and fills in dense blobs. The product is normalized by r^2 (maximum count) and $\sqrt{2r^2}$ (maximum L_2 distance to a nonzero pixel). The resulting distance map $\hat{S}$ is scaled to be between 0 and 1 [5].

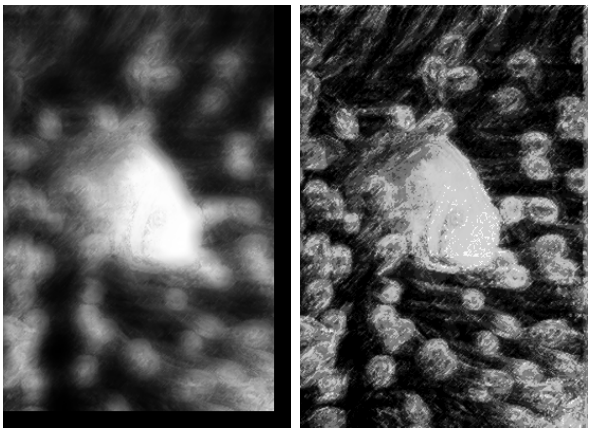
4.2 Propagation and Overlap

Complete (100%) overlap is the full sliding-window WHT, and 0% overlap is equivalent to the non-overlapping, block-based WHT. (In practice, "100%" overlap is actually $n - 1$ pixels, where n is the kernel size – here, $n = 8$.) Complete overlap leads to over-fitting from too many similar neighbors in a single leaf. Propagation helps mitigate this by increasing the search space, but from Figure 12a, simply decreasing the amount of overlap helps

[5] Technically normalization should place the results between 0 and 1 without further scaling, but an error was made – since the area is within a radius r, the maximum count is actually πr^2



(a) Original Image (b) $\bar{S}$ (c) $\hat{S}$



(d) Final Saliency Map ($\bar{S} \cdot \hat{S}$) (e) Saliency Map with Simplified Tempering

Figure 11: Sample image with various intermediate results. Final saliency map is at Figure 11d.

more. 75% overlap is functionally indistinguishable from 0% overlap (Figure 12b), in fact the optimal overlap appears to be around 25%. Here, the contributions of propagtion are small, albeit measurable – yielding a 3-4% improvement at most (Figure 12a).

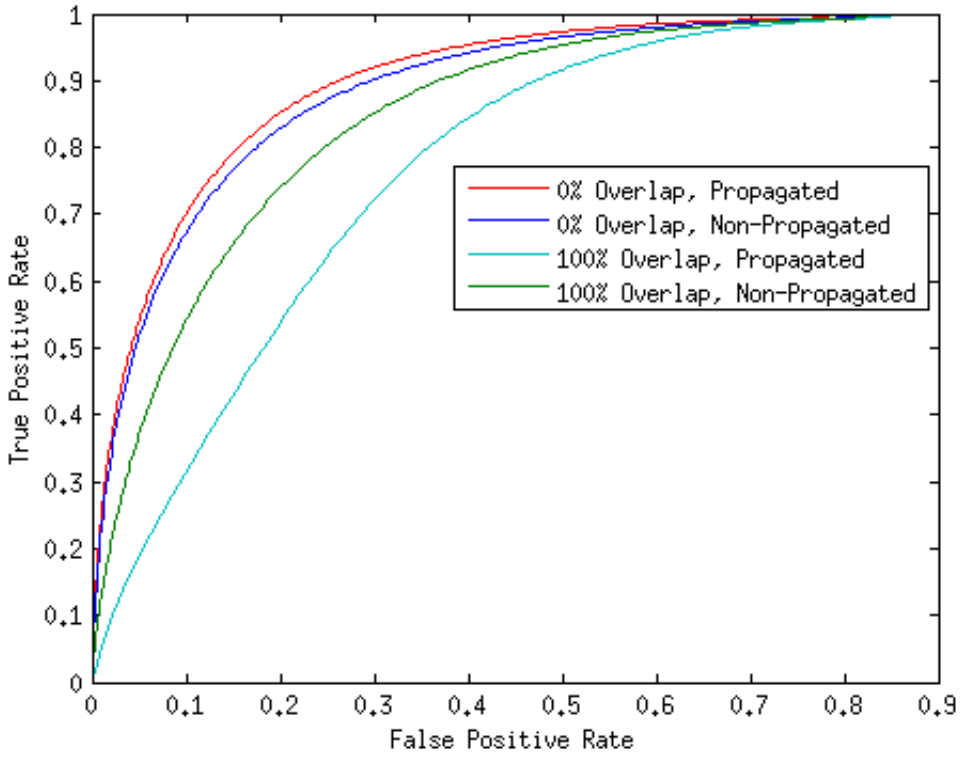
Note that the while the tree is built with partially overlapping patches, each pixel maintains its own patch during the saliency-detection stage.

Figure 13 is interesting: the simplified tempering generates nearly identical ROCs as without – which is somewhat to be expected; while the magnitude of the intensities may change, on balance the relative differences stay the same (Figure 13a). Looking at the curves at each threshold (Figure 13b) tells a different story: the true positive rate (TPR) for $\bar{S}$ is extremely high – a symptom of over-fitting. Indeed, the false positive rate (FPR) for $\bar{S}$ is also convex: pixels are being labelled as salient indiscriminately. For simple tempering, the TPR/FPR has a more agreeable shape (in spite of being somewhat stair-shaped), but incorporating $\hat{S}$ into the final map S shows the best results, due to the fact it diffuses salient pixels from dense areas (increasing TPR) while simultaneously lowering the weight of sparse areas (decreasing FPR).

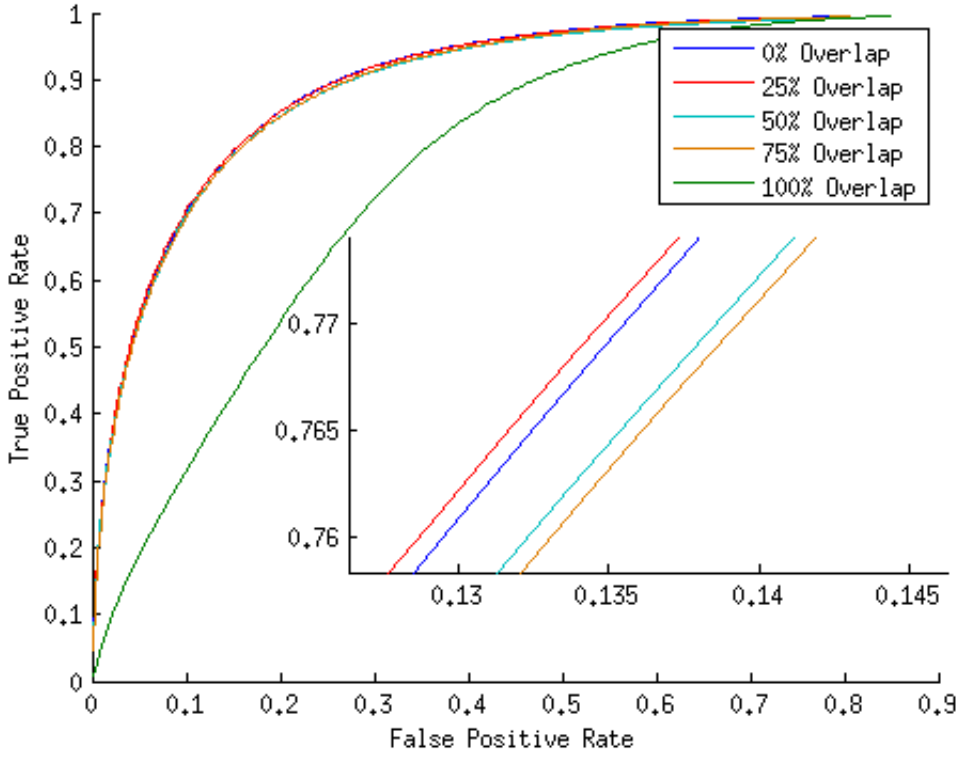
4.3 Related Work and Comparison

We compare against Itti's classical algorithm [25] which emulates low-level features of human vision, Hou's Spectral Residual [23], which suppresses redundant information by taking the difference of an image's Fourier log-spectrum and the spectrum's moving average, and Achanta's Frequency-Tuned Saliency [1] where the entire image is treated as a surround (approximated by the average color across the entire image), with the pixel of interest at the center. [25] and [23] operate on reduced-resolution versions of the image, so we scale the ground-truth down to accommodate when needed. Scaling also has the effect of making these algorithms much faster to run – we operate at full-resolution, with an extremely slow tempering step. [1] also operates at full resolution, which is naturally fast due to its simple construction.

Other saliency detection methods (which are not compared here) include the the bag-of-words approach taken by Parikh, et al to categorize co-occuring patch-based features [39]. Goferman, et al [16] determines saliency by patch distinctness (as described in Section 4), and Margolin, et al im-

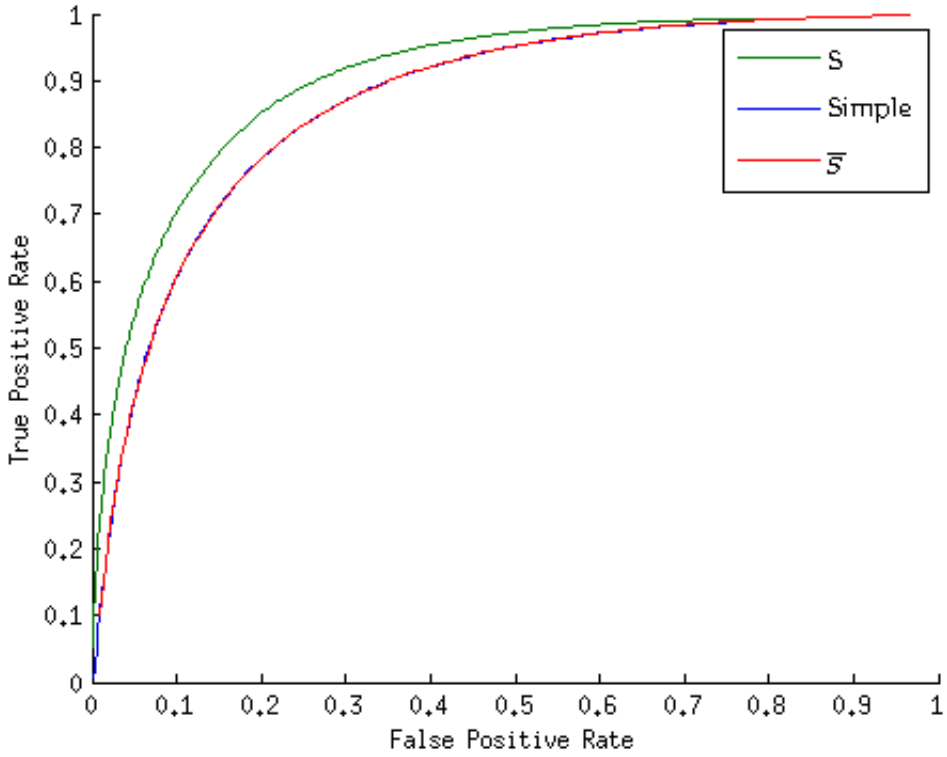


(a) Error curves for various parameters

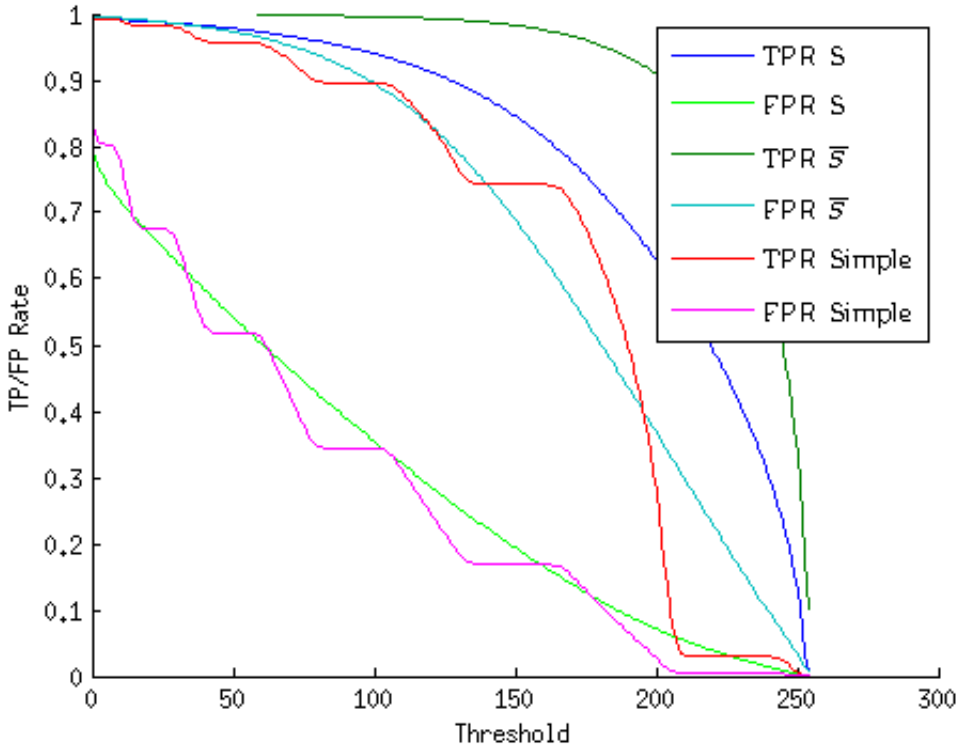


(b) Error curves for various overlaps (with propagation). Zoomed-in section shows the minimal difference between the first 4 modes (note the scale).

Figure 12: Error Curves for Propagation and Overlap, Dataset 1



(a) Error curves, with 25% overlap for each. "S" is the primary saliency map, "Simple" replaces $\hat{S}$ with the average of an iterated threshold (eg, Figure 11e), while $\bar{S}$ is equivalent to taking the non-thresholded result (eg, Figure 11b).



(b) True Positive/False Positive rates at each threshold value between 0 and 255.

Figure 13: Two perspectives on error rates, Dataset 1

proves on this by incorporating color in addition to patch (PCA) distance [34]. Liu, et al build a complex CRF-based algorithm that incorporates multi-scale contrast, a center-surround histogram and the color distribution [30].

ROC statistics are calculated following the method in [1]: The saliency map is thresholded from 0 to 255, and positive/negative statistics are gathered against the ground-truth mask. The values at each threshold are averaged across all images in the dataset. Algorithm 8 sketches this thresholding step, and each saliency algorithm tested is run against this for comparision. We also do not binarize the generated saliency maps (for any algorithm), since it is not needed for our purposes.

Implementations for [1] and [23] were obtained from the authors' websites, while [25] uses the code from [19].

Dataset 1 is composed of 1000 masks selected by [1], with the original images from the MRSA 5000-image salient object dataset [30]. Dataset 2 is composed of 62 images from [23] with well-outlined masks. In Dataset 1, the objects of interest are typically front-and-center while in Dataset 2, more background context is included, so salient features are less dominant. In Dataset 2 (Figure 14) the salient village is inconspicuous, while the lions blend into the background. On the other hand, Dataset 1's skateboarder and airplane are trivially distinguishable (to the human eye, at least). Our algorithm is run with up to 40 neighbors and 50% overlap between patches.

Results are competitive on Dataset 1. The poor performance on Dataset 2 is likely due to its use of difficult natural images. One interesting observation is that the paper which proposed the dataset tends to do well – [1] designated the masks for Dataset 1, while [23] dominates its own images in Dataset 2. Hence it is beneficial to compare results across a variety of datasets, both to prevent over-fitting to a particular image type, and to properly gauge efficacy claims.

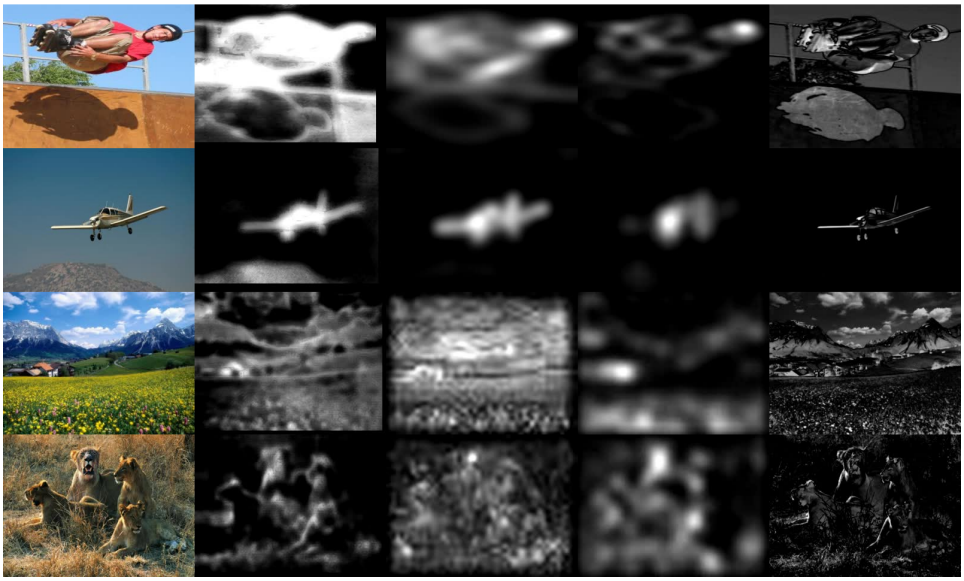
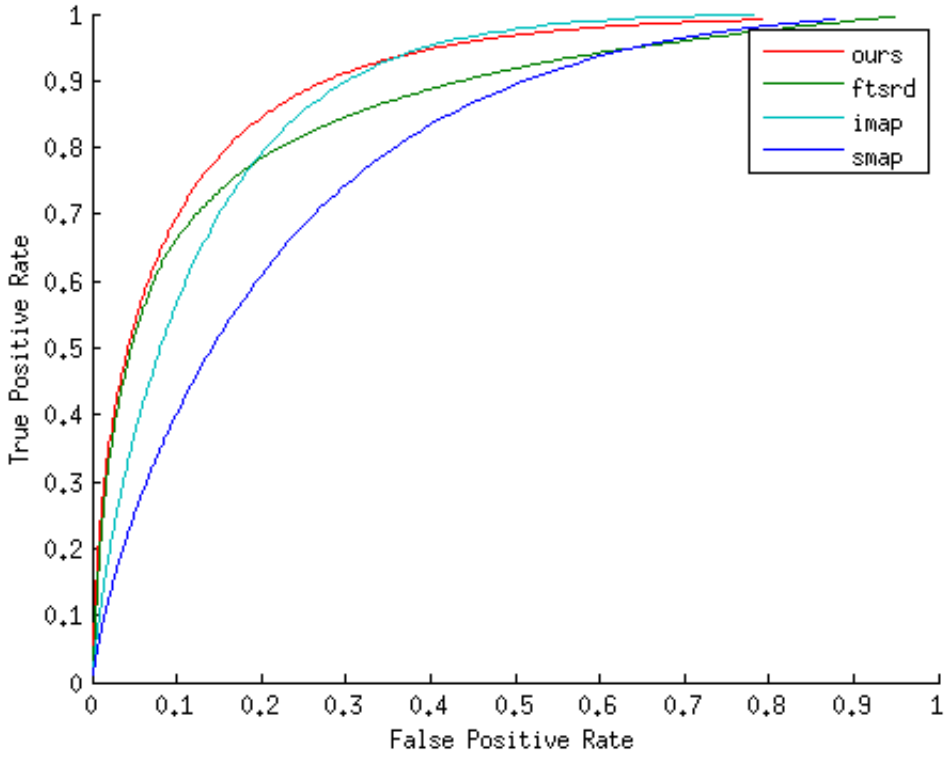


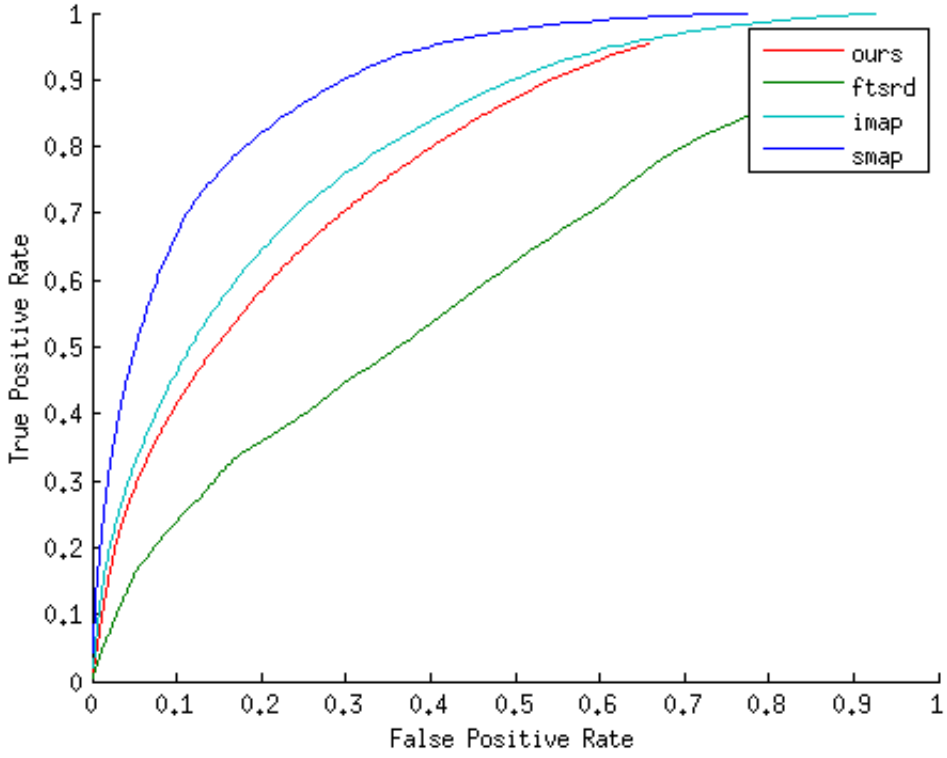
Figure 14: Dataset comparison. Top Half: Dataset 1, Bottom Half: Dataset 2. Left to Right: Original, Ours, Itti, et al [25], Hou et al [23], Achanta et al [1]

```
static void calc_roc(Image* img, Image* mask)
{
    int tp, fp, fn, tn, i;
    for (i = 1; i < 255; i++) {
        tp = fp = fn = tn = 0;
        Image *thresh = threshold(img, i);
        for (y = 0; y < img->height; y++) {
            for (x = 0; x < img->width; x++) {
                if (thresh->pixelAt(x, y) > 0 &&
                    mask->pixelAt(x, y) > 0)
                    tp++; // true positive
                else if (thresh->pixelAt(x, y) > 0)
                    fp++; // false positive
                else if (mask->pixelAt(x, y) > 0)
                    fn++; // false negative
                else
                    tn++;; // true negative
            }
        }
    }
}
```

Algorithm 8: Saliency Statistics



(a) Error curves for Dataset 1



(b) Error curves for Dataset 2

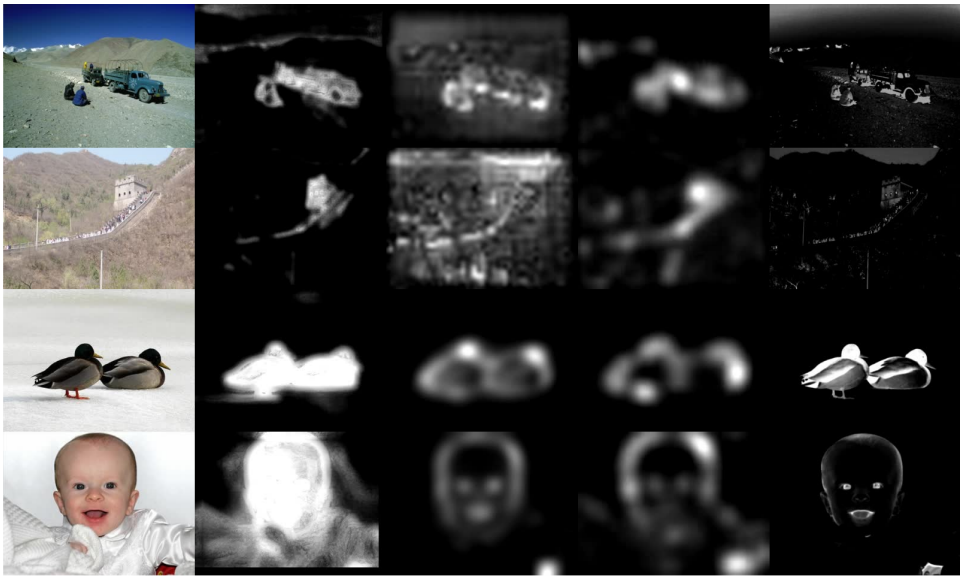


Figure 15: Strong results compared to other algorithms

4.4 Weakness

Images where the objects of interest blend into the background, or background objects have otherwise distracting colors and features. In short, we fare poorly on Dataset 2. For the top half of of Figure 16, the sky and landscaping are mistakenly detected, while in the bottom half, sun glare overwhelms salient objects. Moreover, the eagle is only outlined in the last row. Spectral Residual performs very well, and FTSRD suppresses background well, although it is still susceptible to glare.

5 Background Detection using Nearest-Neighbors

ANN can be used as a primitive operation for background detection in video. Here an algorithm is proposed which does a global ANN search for a query image against a background model. The error in reconstructing the query image from the background (in other words, the "closeness" of the ANN match) indicates the likelihood a pixel is foreground or background. The motivation for this global ANN search is to make the algorithm resilient to camera panning and jitter and simple object motion without bounding the search radius.

To minimize the search space for bacground detection, we take the "ex-

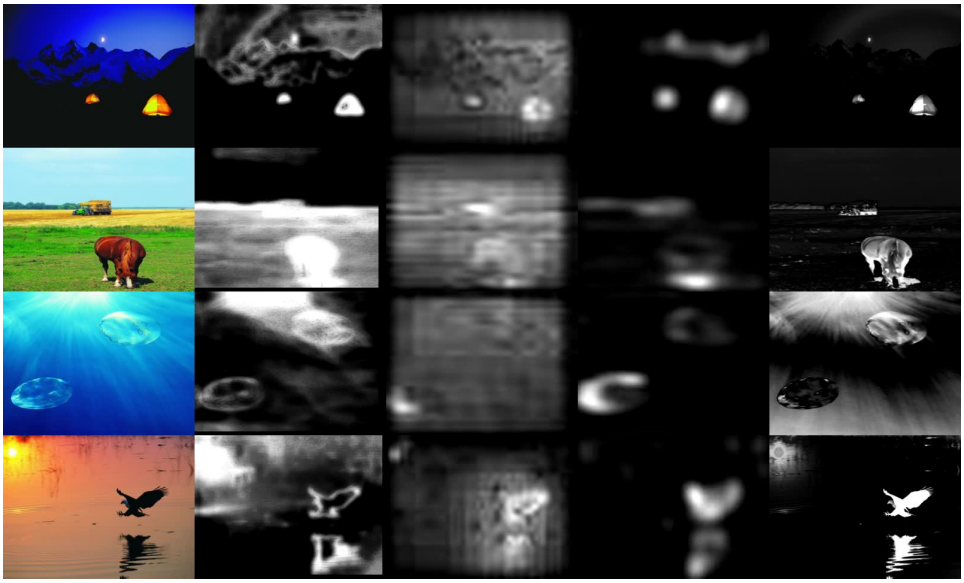


Figure 16: Poor Results compared to other algorithms

pected difference" image as the background model. Not only is this easier to query – most blocks will be (nearly) empty – the foreground should be especially prominent since foreground patches have no close nearest-neighbor in the background model. For this reason, only patch similarity is used for comparison; spatial distance between patches is ignored here.

Section 5.1 describes related work, Section 5.2 describes background-detection algorithm in detail, and Section 5.3 compares the results.

5.1 Related Work

Neighborhood-based methods for background detection are not new; [44] utilizes super-pixels (rather than blocks), which align better to object and neighborhood boundaries. [32] is a randomized algorithm which partitions patches into foreground/background "bags," approximating a super-pixel construction. In Rectgauss-TEX [42], coarse foreground segmentation is first done using a model composed of block-based color histograms, then the segmentation is further refined at the pixel level using Gaussian mixture models. In [21], the Local Binary Pattern statistic is used with a radial neighborhood-based histogram. [36] extends the ideas of [8] for efficient Walsh-Hadamard filtering, incorporating the temporal domain.

Some other algorithms use a purely block-based approach during the

backgroud initialization phase, to take advantage of energy-compacting transforms. After a period of observation, [43] replaces foreground regions with historical blocks using a DCT-based frequency coherence measure, while [3] does so using a Hadamard transform coupled with a gradient-based refinement step at block boundaries. These results struggle with high-frequency backgrounds (since they expect the frequency response to be smooth), leading to incorrect reconstructions of foreground blocks. Also, observation times may be long, leading to high latency before a background is initialized.

Foreground detection can also be represented as a saliency problem, which is the approach taken by Mahadevan et al [33] to subtract backgrounds in highly dynamic scenes.

State-of-the-art methods for background detection, however, are mostly be pixel-based while incorporating a diffusion step to propagate classification information to nearby pixels. PBAS [22] uses a feedback-based control scheme to continually adjust a background model and its segmentation thresholds. ViBE [6] holds a history for each pixel, expiring entries in the history at random. An update to ViBE [49] introduces a neighborhood-based element with morphological open/close operations as part of the thresholding step. As noted by [44], morphological operations may affect statistics, improving precision but hurting recall, likely because they tend to change object contours. PBAS and ViBE both make limited use of neighborhing pixel values through a diffusion step which propagates calculated information.

5.2 Algorithm

Firstly, the background model needs to be built. Here, the moving average of the first n frames of the image sequence is taken. n can be any number, but the dataset being used (`http://changedetection.net`ChangeDetection.net [17]) provides a training period for each of its image sequences, typically $100 < n < 1000$. Algorithms are not required to use all n frames for training, although results will not start tabulating until frame $n + 1$; see Section 5.3.

During the training step, first take the average of the first n images as

the background image I_b, and the average difference I_d:

$$I_b = 1/n \sum_{i=1}^{n} I_i \tag{15}$$

$$I_d = 1/n \sum_{i=1}^{n} |I_i - I_b| \tag{16}$$

For the testing step, perform the following procedure for each input image I_i:

$$I_q = WHT_{16}(|I_i - I_b|) \tag{17}$$

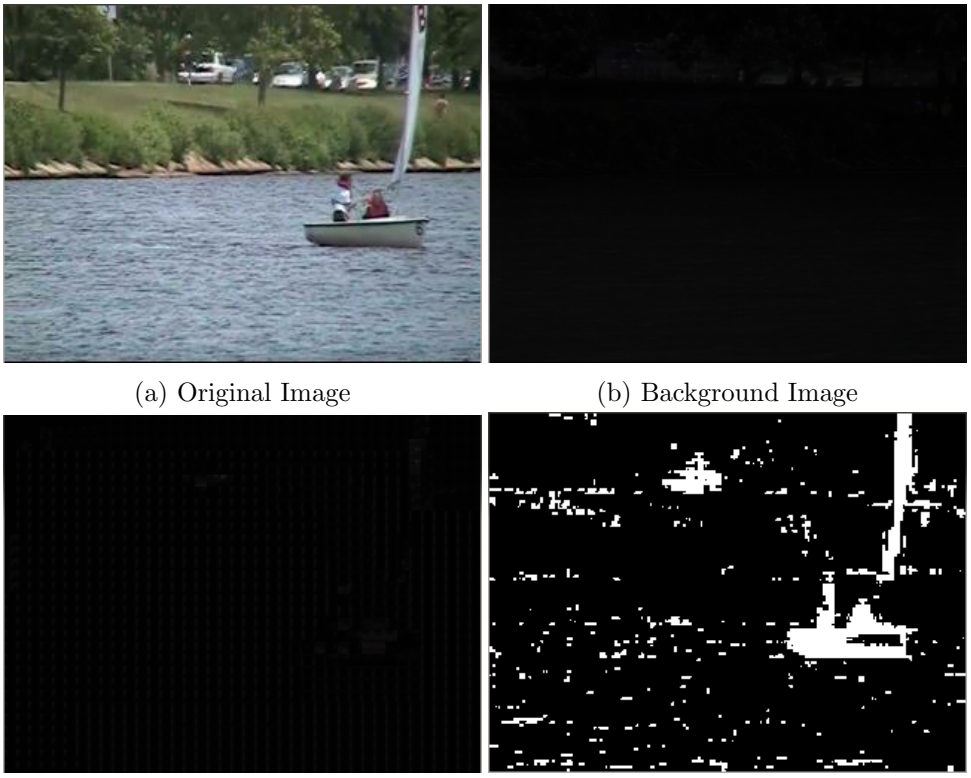
$$I_r = ANN(I_q, I_d) \tag{18}$$

$$I_m = |I_q - I_r| > 25 \tag{19}$$

Note that WHT_{16} indicates the block-based Walsh-Hadamard transform, quantized to 16 coefficients, and ANN performs a nearest-neighbor search for blocks in the query image I_q from the background model which is the "expected difference" indicated by I_d. Areas with a high reconstruction error are likely to be newly introduced foreground items, since the closest ANN match is comparatively distant. This reconstructed image I_r is compared against the original query I_q and thresholded by 25 to obtain a binary foreground mask I_m.

The WHT is performed using the GCK, while the ANN is based off the propagation-assisted KD-tree. Tuples within the kd-tree are composed of the first 16 WHT coefficients, extracted from a 8x8 sliding-window in RGB space. Using I_d for nearest neighbor search (as opposed to the actual background I_b) enables more accurate reconstruction by identifying contours of areas that are likely to see change from frame-to-frame. False positives are minimized this way; see Section 5.3.

While the kd-tree of the background is built using a sliding window WHT, search tuples are extracted from the query image with the block-based (eg, non-overlapping) WHT. Building the kd-tree out of a sliding window is an easy way to increase the search space at a minimal processing cost by building more "templates" for a given area, making the match resilient to jitter and translational object motion. The same does not hold true for the



(a) Original Image (b) Background Image

(c) Reconstructed Image (d) Foreground Mask

Figure 17: Image sequence involving a dynamic background (water).

query image, however; Table 3, exhibits variations on this basic idea, and explains why the approach chosen here is optimal.

Figure 17 shows the algorithm in various states, with a video that exhibits water shimmering (as part of the dynamic background categrory). This can be seen in the Figure 17b, which is the background model: the water area is slightly gray. However, the cars behind the tree exhibit a stronger response in the background, since they are transient, and this foreground activity gets captured during the background-training phase. Figure 17c shows the result after reconstruction; the boat and a moving car in the backgrund are visible (albeit slightly; it is difficult to tell from the reconstruction alone). Figure 17d is the final foreground mask. The car and the boat are clearly captured, but there is a good amount of noise from the water.

5.3 Comparison

Tests were run on the dataset from ChangeDetection.net [17], which incorporates image sequences for several categories, including: a baseline set of videos, and those with various characteristics: dynamic background (swaying foliage, shimmering water), camera jitter, intermittent object motion, thermal imagery, and shadows. Additional, the videos themselves have capture artifacts resulting from compression, sudden color shifts from white balance, etc. Altogether the dataset attempts to be representative of real-world scenarios without biasing algorithms too strongly. Altogether, the dataset has about 90,000 frames with manually annotated ground truth for each one. Results were computed using utilities provided on the ChangeDetection.net site, and compared against publicly available statistics for other algorithms, also from the site.

Two modes of tuple selection were tested, with permutations of two image modes, for a total of four modes. For the bkg image mode, the actual image is used – first the NN search is done with the average image I_b, while results depend on how well the query image $I_q = WHT_{16}(I_i)$ is reconstructed. For the diff image mode, $I_q = WHT_{16}(|I_i - I_b|)$ while the NN uses I_d, which is the default algorithm described in Section 5.2. In the block mode, I_q is broken up into non-overlapping query tuples for NN search (the default algorithm), while in dense mode, query tuples are composed from a sliding window over I_q.

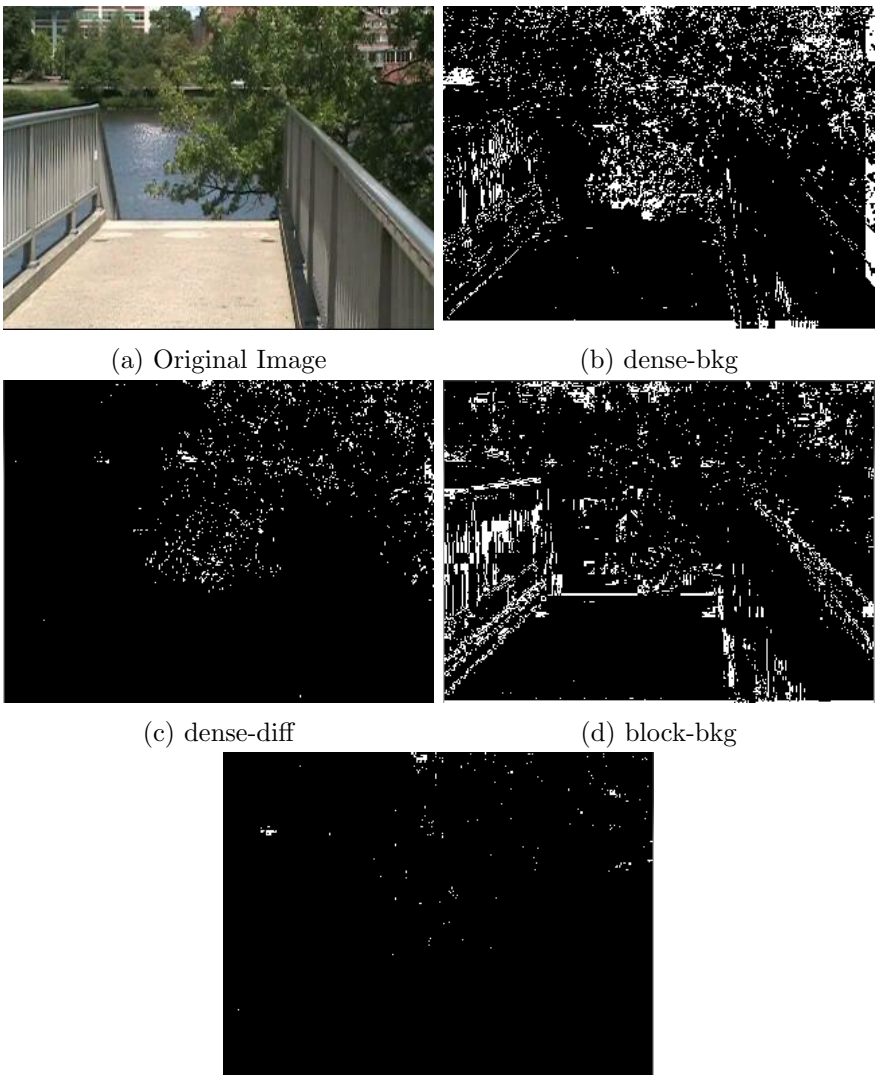
Block+diff has the best results. Due to the nature of the nearest-neighbor reconstruction, each reconstructed tuple is likely to have a non-zero error compared to the original. Dense tuples compound this error, degrading the overall results. When using the difference image, there is less relevant detail which could cause erroneous reconstructions. Figure 18 illustrates these shortcomings. Hence, the combination of block+diff works best for this algorithm, in 5 out of 7 metrics. The FNR and recall are the exceptions – since block+diff is a mechanism to reduce sensitivity of the algorithm, it tends to under-label more pixels than the other modes. However, this is compensated by greatly increased precision – the FMeasure suggests this is a sensible tradeoff. Figure 18 illustrates the foreground/background masks that are generated using each mode.

Shown in Table 4 are tests incorporating pixel-based methods, neighborhood-based methods, and "traditional" methods, taken from the 2012 workshop

Algorithm	Recall	Specificity	FPR	FNR	PWC	Precision	FMeasure
block-bkg	0.3179	0.9353	0.0647	0.0300	8.9900	0.2162	0.2124
dense-bkg	0.7416	0.8563	0.1437	0.0122	14.7091	0.2166	0.3055
block-diff	0.6168	0.9711	0.0289	0.0174	4.2247	0.6026	0.5728
dense-diff	0.6394	0.9573	0.0427	0.0161	5.4406	0.5062	0.5122

Table 3: Average result for various nearest-neighbor modes across all categories of the ChangeDetection.net dataset. *block* or *dense* indicates whether the query image was broken up into non-overlapping blocks for NN search, or conducted using a sliding window. *bkg* indicates whether the background model used for NN search was the background itself (bkg, I_b) or the background difference (diff, I_d).

- Recall= $\frac{TP}{(TP+FN)}$
- Specificity= $\frac{TN}{(TN+FP)}$
- FPR (False Positive Rate) = $\frac{FP}{(FP+TN)}$
- FNR (False Negative Rate) = $\frac{FN}{(TP+FN)}$
- PWC (Percentage of Wrong Classifications) = $\frac{100*(FN+FP)}{(TP+FN+FP+TN)}$
- Precision= $\frac{TP}{(TP+FP)}$
- F-Measure= $2 * \frac{recall*precision}{(recall+precision)}$



(a) Original Image (b) dense-bkg

(c) dense-diff (d) block-bkg

(e) block-diff

Figure 18: Masks for various modes from a dynamic background sequence. Black indicates a pixel labelled as background, white foreground.

Algorithm	Recall	Specificity	FPR	FNR	PWC	Precision	FMeasure
Ours	0.6168	0.9711	0.0289	0.0175	4.2247	0.6026	0.5728
PBAS [22]	0.7840	0.9898	0.0102	0.2160	1.7693	0.8160	0.7532
ViBE [49]	0.6907	0.9928	0.0072	0.3093	2.1824	0.8318	0.7224
PSP [44]	0.8037	0.9830	0.0170	0.1963	2.3937	0.7512	0.7372
Rectgauss [42]	0.5156	0.9862	0.0138	0.4844	3.6842	0.7190	0.5221
GMM [48]	0.7108	0.9860	0.0140	0.2892	3.1037	0.7012	0.6624
Bayesian [41]	0.6018	0.9826	0.0174	0.3982	3.3879	0.7435	0.6272
Mahalanobis [9]	0.7607	0.9599	0.0401	0.2393	4.6631	0.6040	0.6259
Euclidean [9]	0.7048	0.9692	0.0308	0.2952	4.3465	0.6223	0.6111

Table 4: Algorithm comparision, average score across all categories of the ChangeDetection.net dataset.

on change detection [17]. PBAS [22], ViBE [49] and PSP [44] are the three top-performing algorithms. PSP (Probabilistic SuperPixels) [44] and Rectgauss [42] (block-based GMM) are neighborhood-based. PBAS and ViBE [6] are fundamentally pixel-based (with randomized diffusion/propagation steps.) Historical or "traditional" algorithms are also included for reference. Two are based on simple pixel-wise distance from a fixed background model, using the Euclidean and Mahalanobis metrics as described in [9]. The other two are statistical approaches, based off a Bayesian background model [41] and the canonical paper on Gaussian mixture models for background subtraction [48].

The results show our method near the bottom of the pack across all metrics. Interestingly, Rectgauss, the method most similar to ours, also struggles to be competitive, indicating that block-based neighborhood operations might not be ideal for this application. This is perhaps due to capturing excessive context, paired with an inflexible background model. There is little reason to choose our algorithm for background detection in its present state; most other algorithms are simpler, more accurate and more performant – for example, ViBE is simple enough to be embedded onto point-and-shoot cameras [6].

In Table 5, results are shown for all categories in the ChangeDetection.net dataset. Unsurprisingly, the baseline sequences exhibit the best results. Note that the dynamic background has a comparatively low value for PWC; looking at with low precision/recall values, it may be that the segmentation threshold is set too low for this particular category. However,

Category	Recall	Specificity	FPR	FNR	PWC	Precision	FMeasure
Baseline	0.7359	0.9969	0.0031	0.0104	1.2750	0.9146	0.8019
CJ	0.4491	0.9725	0.0275	0.0237	4.9068	0.4833	0.4488
DB	0.5760	0.9876	0.0124	0.0040	1.6159	0.4241	0.4616
IOM	0.6014	0.9030	0.0970	0.0369	11.6014	0.4635	0.4606
Shadow	0.6572	0.9805	0.0195	0.0171	3.4941	0.6164	0.5936
Thermal	0.6810	0.9861	0.0139	0.0126	2.4551	0.7135	0.6701

Table 5: Results for different categories. CJ=Camera Jitter, DB=Dynamic Background, IOM=Intermittent Object Motion, using the algorithm described in section 5.2.

poor results in other categories advise against lowering the threshold for the general case. This algorithm, as-is, is simply not competitive, especially when compared to better methods.

6 Conclusion

In this thesis, we described a fast implementation of an already-fast Walsh-Hadamard Transform algorithm, improvements to a propagation-assisted kd-tree, and then used the WHT and kd-tree as the cornerstone of algorithms for saliency and background detection. Saliency showed promising results for images with a strong focal point, at the expense of an extremely expensive tempering step. Background detection struggled to surpass even primitive operations such as Euclidean distance [6], and while it runs in real-time, state-of-the-art backgroud detection algorithms such as [6] are both much faster and more accurate.

6.1 Algorithmic Improvements

There are many ways this work could be improved. Possible ideas include using a different transform instead of the WHT, such as the DCT or PCA as used in [34] – TreeCANN found the PCA yields higher matching accuracy at the expense of more computation [38]. Higher-level features such as SIFT could also be considered, or non-uniform neighborhoods such as superpixels. Patch-based histograms have also been used with success in [42, 21]. How

[6] Ironically, Euclidean is itself the primary distance metric used in our background detection algorithm

to incorporate these high-level features in the propagation-assisted kd-tree is an open question.

For video, a more robust background model should be investigated, rather than assuming a static background model after a fixed training step, to mitigate "ghosting" after a foreground object is accidentally initialized into the background, or transitory changes in the entire image, such as when the camera applies automatic color balancing. The background could be re-created every so often, or adapted iteratively, to prevent ghosting. High-level, color-invariant features could handle transient changes across the entire image.

6.2 Implementation Improvements

Another issue is that of processing speed. Although most of the code was in C, it is otherwise unoptimized.

Saliency detection and correspondence (intra-frame kd-tree querying) have immediately sequential dependencies due to the propagation step from the left, which precludes an embarrassingly parallel ANN query for each patch using the GPU [14]. This may be mitigated by slicing each image into independent sections and processing those in parallel, but quality is likely to suffer due to the lack of propagation along section boundaries. If a GPU-based ANN search can spend more time on accuracy during the search proper, omitting propagation might be acceptable if parallelization also leads to a speed increase. For example, propagation is of limited benefit to saliency, especially if the number (k) of neighbors retained by the original match is higher (Figure 12a).

The most time-consuming part of saliency detection, the tempering step, can be done in parallel for each threshold level, and each saliency scale is independent. Video background detection can be parallelized straightforwardly, since each frame (after training) is independent of the other.

On a more granular level, much of the computation-heavy code can be easily vectorized: SAD, Euclidean and WHT calculations would all benefit. There are easy wins available during the kd-tree building phase: a uniqueness filter (through a hash table, Bloom filter or alike), which can be incorporated to improve fan-out (identical patches in the kd-tree are redundant). This could be done during the pre-processing step used to determine traversal order. Alternatively, the traversal step could be skipped altogether,

albeit with testing first, as described in Section 3.4.

References

[1] Radhakrishna Achanta, Sheila Hemami, Francisco Estrada, and Sabine Süsstrunk. Frequency-tuned Salient Region Detection. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR 2009)*, pages 1597 – 1604, 2009. For code and supplementary material, click on the url below.

[2] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. In *ACM-SIAM SYMPOSIUM ON DISCRETE ALGORITHMS*, pages 573–582, 1994.

[3] Davide Baltieri, Roberto Vezzani, and Rita Cucchiara. Fast background initialization with recursive hadamard transform. In *Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on*, pages 165–171, 29 2010-Sept. 1.

[4] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), August 2009.

[5] Connelly Barnes, Eli Shechtman, Dan B Goldman, and Adam Finkelstein. The generalized PatchMatch correspondence algorithm. In *European Conference on Computer Vision*, September 2010.

[6] Olivier Barnich and Marc Van Droogenbroeck. Vibe: A universal background subtraction algorithm for video sequences. *Image Processing, IEEE Transactions on*, 20(6):1709–1724, June 2011.

[7] Jeffrey S. Beis and David G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces, 1997.

[8] Gil Ben-Artzi, Hagit Hel-Or, and Yacov Hel-Or. The gray-code filter kernels. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(3):382–393, March 2007.

[9] Yannick Benezeth, Pierre-Marc Jodoin, Bruno Emile, Christophe Rosenberger, and Hlne Laurent. Comparative study of background subtraction algorithms. *Journal of Electronic Imaging*, 19(3):033003–033003–12, 2010.

[10] Zhuoyuan Chen, Hailin Jin, Zhe Lin, Scott Cohen, and Ying Wu. Large displacement optical flow from nearest neighbor fields. *Computer Vision and Pattern Recognition, 2013. CVPR 2013. IEEE Conference on*, page To Appear, Jun 2013.

[11] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[12] Nicolas Devillard. Fast Median Search: an ANSI C Implementation. `http://ndevilla.free.fr/median/median/index.html`, July 1998.

[13] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, September 1977.

[14] V. Garcia, E. Debreuve, and M. Barlaud. Fast k nearest neighbor search using gpu. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pages 1–6, 2008.

[15] Stas Goferman, Lihi Zelnik-Manor, and Ayellet Tal. Context-aware saliency detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(10):1915–1926, 2012.

[16] Stas Goferman, Lihi Zelnik-Manor, and Ayellet Tal. Context-aware saliency detection. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2376–2383, June 2010.

[17] Nil Goyette, Pierre-Marc Jodoin, Faith Porikli, Janusz Konrad, and Prakash Ishwar. Changedetection.net: A new change detection benchmark dataset. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 1–8, June.

[18] Miguel Granados, KwangIn Kim, James Tompkin, Jan Kautz, and Christian Theobalt. Background inpainting for videos with dynamic objects and a free-moving camera. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision ECCV 2012*, volume 7572 of *Lecture Notes in Computer Science*, pages 682–695. Springer Berlin Heidelberg, 2012.

[19] Jonathan Harel. A saliency implementation in matlab. `http://www.klab.caltech.edu/~harel/share/gbvs.php`, December 2009.

[20] Kaiming He and Jian Sun. Computing nearest-neighbor fields via propagation-assisted kd-trees. In *CVPR*, pages 111–118. IEEE, 2012.

[21] M. Heikkila and M. Pietikainen. A texture-based method for modeling the background and detecting moving objects. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(4):657–662, 2006.

[22] Martin Hofmann, Philipp Tiefenbacher, and Gerhard Rigoll. Background segmentation with feedback: The pixel-based adaptive segmenter. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 38–43, June.

[23] Xiaodi Hou and Liqing Zhang. Saliency detection: A spectral residual approach. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, 2007.

[24] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[25] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, Nov 1998.

[26] Don Johnson. *Fundamentals of Electrical Engineering 1*. Connexions, 2008.

[27] Simon Korman and Shai Avidan. Coherency sensitive hashing. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1607–1614, November 2011.

[28] Simon Korman and Shai Avidan. VidPairs data set. `http://www.eng.tau.ac.il/~simonk/CSH/`, November 2011.

[29] Neeraj Kumar, Li Zhang, and Shree Nayar. What is a good nearest neighbors algorithm for finding similar patches. In *in images?, in ECCV*, pages 364–378.

[30] Tie Liu, Jian Sun, Nan ning Zheng, Xiaoou Tang, and Heung yeung Shum. Learning to detect a salient object. In *in: Proceedings of IEEE Computer Society Conference on Computer and Vision Pattern Recognition (CVPR)*, pages 1–8. CVPR, 2007.

[31] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[32] Le Lu and Gregory D. Hager. Dynamic foreground/background extraction from images and videos using random patches. In *NIPS'06*, pages 929–936, 2006.

[33] Vijay Mahadevan and Nuno Vasconcelos. Background subtraction in highly dynamic scenes. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–6, June 2008.

[34] Ran Margolin, Ayellet Tal, and Lihi Zelnik-Manor. What makes a patch distinct? *Computer Vision and Pattern Recognition, 2013. CVPR 2013. IEEE Conference on*, page To Appear, Jun 2013.

[35] Yasuyuki Matsushita, Eyal Ofek, Weina Ge, Xiaoou Tang, and Heung-Yeung Shum. Full-frame video stabilization with motion inpainting. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(7):1150–1163, July 2006.

[36] Y. Moshe, H. Hel-Or, and Y. Hel-Or. Foreground detection using spatiotemporal projection kernels. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3210–3217, 2012.

[37] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[38] Igor Olonetsky and Shai Avidan. Treecann - k-d tree coherence approximate nearest neighbor algorithm. In *Proceedings of the 12th European conference on Computer Vision - Volume Part IV*, ECCV'12, pages 602–615, Berlin, Heidelberg, 2012. Springer-Verlag.

[39] Devi Parikh, C.Lawrence Zitnick, and Tsuhan Chen. Determining patch saliency using low-level context. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision ECCV 2008*, volume 5303 of *Lecture Notes in Computer Science*, pages 446–459. Springer Berlin Heidelberg, 2008.

[40] K.A. Patwardhan, G. Sapiro, and M. Bertalmio. Video inpainting of occluding and occluded objects. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, volume 2, pages II–69–72, September.

[41] Fatih Porikli and Oncel Tuzel. Bayesian background modeling for foreground detection. In *Proceedings of the third ACM international workshop on Video surveillance & sensor networks*, VSSN '05, pages 55–58, New York, NY, USA, 2005. ACM.

[42] Dora Rahi, Pier-Luc St-Onge, and Guillaume-Alexandre Bilodeau. Rectgauss-tex: Block-based background subtraction. Technical Report EPM-RT-2012-03, Ecole Polytechnique de Montreal, Mar 2012.

[43] V. Reddy, C. Sanderson, and B.C. Lovell. An efficient and robust sequential algorithm for background estimation in video surveillance. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 1109–1112, November.

[44] A. Schick, M. Bauml, and R. Stiefelhagen. Improving foreground segmentations with probabilistic superpixel markov random fields. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 27–31, 2012.

[45] Robert Sedgewick and Kevin Wayne. *Algorithms, 4th Edition*. Addison-Wesley, 2011.

[46] J.L. Shanks. Computation of the fast walsh-fourier transform. *Computers, IEEE Transactions on*, C-18(5):457–459, 1969.

[47] Chanop Silpa-Anan and Richard Hartley. Optimised kd-trees for fast image descriptor matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

[48] Chris Stauffer and W. E L Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2, pages –252 Vol. 2, 1999.

[49] M. Van Droogenbroeck and O. Paquot. Background subtraction: Experiments and improvements for ViBe. In *Change Detection Workshop (CDW)*, Providence, Rhode Island, June 2012.

[50] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 479–488, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[51] Yonatan Wexler, Eli Shechtman, and Michal Irani. Space-time completion of video. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(3):463–476, March.