# Fast Mean Shift by Compact Density Representation

Daniel Freedman and Pavel Kisilev
Hewlett-Packard Laboratories
Haifa, Israel
daniel.freedman@hp.com

## Abstract

*The Mean Shift procedure is a well established clustering technique that is widely used in imaging applications such as image and video segmentation, denoising, object tracking, texture classification, and others. However, the Mean Shift procedure has relatively high time complexity which is superlinear in the number of data points. In this paper we present a novel fast Mean Shift procedure which is based on the random sampling of the Kernel Density Estimate (KDE). We show theoretically that the resulting reduced KDE is close to the complete data KDE, to within a given accuracy. Moreover, we prove that the time complexity of the proposed fast Mean Shift procedure based on the reduced KDE is considerably lower than that of the original Mean Shift; the typical gain is of several orders for big data sets. Experiments show that image and video segmentation results of the proposed fast Mean Shift method are similar to those based on the standard Mean shift procedure. We also present a new application of the Fast Mean Shift method to the efficient construction of graph hierarchies for images; the resulting structure is potentially useful for solving computer vision problems which can be posed as graph problems, including stereo, semi-automatic segmentation, and optical flow.*

## 1. Introduction

Kernel density estimation and the Mean Shift clustering procedure are well accepted techniques in the field of computer vision, see for example [10, 5] and references therein. Mean Shift is widely used in many imaging applications such as image and video segmentation [5, 20], denoising [3], object tracking [7], and texture classification [11], *inter alia*. Roughly speaking, the Mean Shift procedure consists of two steps: (i) the construction of a probability density which reflects the underlying distribution of points in some feature space, and (ii) the mapping of each point to the mode (maximum) of the density which is closest to it.

One of the main difficulties in applying Mean Shift based clustering to big data sets is its computational complexity which is superlinear in the number of data points. There are several existing techniques which have been developed to increase the speed of Mean Shift. DeMenthon and Megret [8] use an iterative, scale-space like approach to Mean-Shift, with increasing bandwidth. Yang *et al.* [23] use the Fast Gauss Transform to speed up the sum in the Mean Shift iteration. Guo *et al.* [13] decompose the Mean Shift sum into a number of local subsets. Paris and Durand [17] use the separability of the multidimensional Gaussian kernel to perform d separate one-dimensional convolutions. Wang *et al.* [21] use a clever data structure, the dual tree, to speed up Mean Shift. Also somewhat related is the paper by Vevaldi and Soatto on "Quick Shift" [19].

In this paper, we introduce a new technique, which deals directly with the description or space complexity of the Kernel Density Estimate, which is linear in the number of data points. The main focus of this paper is a novel fast Mean Shift procedure which is based on the computation of a pared down Kernel Density Estimate, using sampling techniques. We show theoretically that the resulting reduced KDE is close to the complete data KDE, to within a given accuracy. The time complexity of the proposed fast Mean Shift procedure based on the reduced KDE is considerably lower than that of the original Mean Shift; the typical gain is of several orders for big data sets. We verify this large gain experimentally in a number of segmentation experiments.

The method for constructing the more compactly represented KDE is easy to implement; furthermore, the new technique is orthogonal to the existing techniques for speeding up Mean Shift, and in most cases, can be implemented along side these older methods for even better performance. Note that while there has been some work in this regard (compact KDE representation) in the past, it mainly involves techniques which rely heavily on neural networks and self-organizing maps [18, 22, 12]. Neural networks lead to a high implementation complexity (as well as other issues) which we wish to avoid in this work.

The remainder of the paper is organized as follows. In Section 2 we first briefly review the KDE framework and the

978-1-4244-3991-1/09/$25.00 ©2009 IEEE 

Mean Shift algorithm for mode finding. Then we propose a sampling method for constructing the compact representation of KDE. Next, based on this compact KDE, we define our Fast Mean Shift procedure, and analyze its computational complexity as compared to the standard Mean Shift. We also provide a rule for optimal bandwidth selection for the Fast Mean Shift procedure. In Section 3, we demonstrate the performance of the proposed method on the tasks of image and video segmentation, and compare results with standard Mean Shift. In Section 4, we present a new application of the Fast Mean Shift method for constructing multiscale graph hierarchies for images. Section 5 concludes.

## 2. Fast Mean Shift

In this section, we present the Fast Mean Shift algorithm. After a brief review of the standard Mean Shift algorithm, we move to a general discussion of compact representations for the kernel density estimate, and prove that a sampling-based scheme gives us such a representation. Given this compact representation, we then propose a fast technique for Mean Shift, and show that its complexity is considerably lower than that of the standard Mean Shift. We then examine a soft clustering variant of this algorithm, and conclude with a discussion of the optimal choice of bandwidth.

### 2.1. Review of Mean Shift

In this section, we review the ordinary Mean Shift procedure. We begin by briefly defining the Kernel Density Estimate (KDE). Our data is a set of points $\{x_i\}_{i=1}^n$, sometimes referred to as feature vectors, living in a Euclidean space: $x_i \in \mathbb{R}^d$. In computer vision applications, there is generally one such vector per pixel (or voxel in the three-dimensional case); the vector may be colour, colour augmented with position, texture, and so on. The Kernel Density Estimate (KDE) of this data is then taken to be

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} K_{\mathbf{H}}(x - x_i)$$

This is an estimate of the probability density underlying the points, where the function $K_{\mathbf{H}}(x - x_i)$ is essentially a bump centered at $x_i$. More specifically, we take $K_{\mathbf{H}}(z) = |\mathbf{H}|^{-1/2}K(\mathbf{H}^{-1/2}z)$, where the *kernel* $K$ is itself a probability density with zero mean, identity covariance, and satisfying $\lim_{\|x\|\to\infty} \|x\|^d K(x) = 0$. Common choices for $K$ include Gaussian, uniform, and (multidimensional) Epanechnikov kernels. In many cases of interest, $\mathbf{H}$ will be diagonal; in general, though, this need not be the case.

The Mean Shift algorithm is essentially a hill-climbing algorithm; that is, starting at any point $x$, the Mean Shift algorithm is an efficient iteration scheme which brings $x$ up the hill of the KDE, finally stopping at the local maximum (or mode) of the KDE in whose basin of attraction $x$ lies. To specify the Mean Shift iteration formally, let us simplify the form of the kernel, and take the radially symmetric form $K(x) = ck(\|z\|^2)$, where $k$ is a one-dimensional *profile*, such as the one-dimensional Gaussian, uniform, or Epanechnikov profiles, and $c$ is a normalization. Denoting $g = k'$, then the Mean Shift iteration is given by

$$x \leftarrow \frac{\sum_{i=1}^n x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^n g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)} \equiv M(x) \qquad (1)$$

Iterating an infinite number of times is guaranteed to bring $x$ to the mode in whose basin of attraction it lies. The advantage of this procedure over ordinary gradient ascent is the lack of need to set a time-step parameter (note no such parameter is present in the above expression); practically, the mean-shift tends to converge in a very small number of steps, typically around 5.

In order to use the Mean Shift algorithm for segmentation or clustering on the set $\{x_i\}_{i=1}^n$, one may run the iterations starting at each data point. Denoting by $M^1(x) = M(x)$, $M^2(x) = M(M(x))$ and so on, we map each point $x_i$ to $M^\infty(x_i)$ (though recall that typically, $M^5(x_i)$ will do the job). Since there are a much smaller number of modes than there are points, this is a form of segmentation or clustering, which works very well in practice in a number of applications [20, 6, 14].

### 2.2. A More Compact KDE: the Problem

Without explicitly computing the complexity of Mean Shift clustering (we put off this exercise until Section 2.5), we note that the running time will be superlinear in $n$, the number of data points. If we are concerned with large images coming from today's cameras, $n$ can be easily be in the range $10^7$. Worse yet, we may be interested in using Mean Shift on video or on 3D imagery, such as CT or MRI images; in this case, $n$ of the order of $10^8$ or higher is easily conceivable. It is therefore worthwhile to consider a faster Mean Shift technique. One of the main speed bottlenecks is the description complexity of the KDE, which is $O(n)$; we thus begin by discussing the problem of finding a more compact description for the KDE.

We wish to find a KDE which is close to $f(x)$, but which uses many fewer points to generate it. That is, we wish to solve the following problem:

$$\min_{\{\hat{x}_j\}_{j=1}^m, \hat{\mathbf{H}}} D(\hat{f}(\cdot), f(\cdot)) \text{ subject to } \hat{f}(x) = \frac{1}{m}\sum_{j=1}^{m} K_{\hat{\mathbf{H}}}(x - \hat{x}_j)$$

where $D$ is a distance measure between probability densities and $m$ is a fixed number of points, with $m \ll n$. This optimization seeks to find the data points $\hat{x}_j$ underlying a second KDE $\hat{f}$ which well-approximates the original KDE $f$, but using many fewer points.

There are several natural choices for the distance measure $D$, ranging from $L_p$ type distances to more information theoretic measures such as the Kullback-Leibler divergence. In all such cases, the resulting optimization problem is hard to solve globally; this is due the fact that the $\hat{x}_j$ appear within the kernel $K$, leading a non-convex optimization, for which global solutions cannot generally be found efficiently. However, $f$ and $\hat{f}$ are more than functions; they are both densities, and this fact gives us a means of finding an efficient solution to the compact representation problem.

### 2.3. A More Compact KDE via Sampling

Our solution is very simple: we choose the samples $\hat{x}_j$ by sampling from the distribution given by $f(\cdot)$. Before discussing the logic of this approach, note that such sampling is quite feasible in general, and may be implemented as a three-step procedure:

**Theorem 1** *For each $j = 1, \ldots, m$, suppose we construct $\hat{x}_j$ as follows:*

1. *choose a random integer $r_j \in \{1, \ldots, n\}$;*
2. *choose a random sample $\delta_j$ from $K(\cdot)$;*
3. *set $\hat{x}_j = x_{r_j} + \mathbf{H}^{1/2}\delta_j$.*

*Then $\hat{x}_j$ is a proper sample of $f$.*

**Proof:** See [9].

As long as one can sample easily from $K(\cdot)$, as is the case for the Gaussian, uniform, and Epanechnikov kernels, then the procedure is very simple.

Now, the main question becomes: if we construct a KDE $\hat{f}$ based on the random samples $\hat{x}_j$, will it be close to the true KDE $f$? Of course, the KDE $\hat{f}$ itself will be a random variable (i.e. a random function), and thus any result we prove will be of a probabilistic nature. In fact, we have the following result, which guarantees that $f$ and $\hat{f}$ are close in expectation, in an $L_2$ sense:

**Theorem 2** *Let $f$ be KDE with $n$ points, and let $\hat{f}$ be a KDE constructed by sampling $m$ times from $f$, as above, and assume a diagonal bandwidth matrix $\hat{\mathbf{H}} = \hat{h}^2\mathbf{I}$. Let the expected squared $L_2$ distance between the two densities be given by $J = E[\int (f(x) - \hat{f}(x))^2 dx]$. Then*

$$J \leq 4A\hat{h} + A^2\hat{h}^2V + \frac{B}{m\hat{h}^d} + \frac{ABV}{m\hat{h}^{d-1}} \qquad (2)$$

*where $A, B, V$ are constants which do not depend on $\hat{h}$ or $m$.*

**Proof:** See [9].

The meaning of this theorem is straightforward: the two KDEs $f$ and $\hat{f}$ will be close (in expectation) if $m$ is large enough, and if the bandwidth $\hat{h}$ is chosen properly[1] as a function of $m$. This is precisely what we want in a compact representation: the description complexity of the KDE has been reduced from $O(n)$ to $O(m)$, but we generate close to the same density, in the expected $L_2$ sense.

We turn to the issue of bandwidth selection in Section 2.7; in the next section, we turn to the central question of how to use our reduced representation KDE in Mean Shift Clustering.

### 2.4. Fast Mean Shift

How can we incorporate our more compact KDE into the Mean Shift clustering algorithm? We use the following three step procedure:

1. **Sampling:** Take $m$ samples of the density $f$ to yield $\{\hat{x}_j\}_{j=1}^m$. Form the new density $\hat{f}(x) = \sum_{j=1}^m K_{\hat{h}}(x, \hat{x}_j)$.
2. **Mean Shift:** Perform Mean Shift on each of the $m$ samples: $\hat{x}_j \rightarrow \hat{M}^\infty(\hat{x}_j)$. Here, $\hat{M}$ indicates that we use $\hat{f}$ (rather than $f$) for the Mean Shift.
3. **Map Backwards:** For each $x_i$, find the closest new sample $\hat{x}_{j^*}$. Then $x_i \rightarrow \hat{M}^\infty(\hat{x}_{j^*})$.

In Step 1, we construct the reduced KDE, and in Step 2, we perform Mean-Shift on this smaller KDE. Given these modes in the reduced KDE, the question is how to map backwards to the original data. We deal with in Step 3, by mapping from each point in the original data ($x_i$) to the closest point in the reduced data ($\hat{x}_{j^*}$), and from there to the mode to which that point flows ($\hat{M}^\infty(\hat{x}_{j^*})$).

The key speed-up occurs in the second step; instead of using all $n$ samples to compute the Mean Shift, we can use the reduced set of $m$ samples. In the next section, we will quantify the precise theoretical speed-up that this entails; for the moment, note that in a naive implementation it can lead to a speed-up of $n/m$, which can in pratice be greater than 100, and often considerably larger than that.

### 2.5. Complexity Analysis

The key aspect in the computation of complexity is the speed of the nearest neighbour search. Note that in each Mean Shift iteration, see Equation (1), one must compute the nearest neighbours out of the $n$ samples $x_i$ to the point in question, $x$. Suppose that we have a data structure which permits nearest neighbour queries in time $q(n)$, and requires a preprocessing time of $p(n)$ to compute; we will look at examples of such structures shortly, but for now, we will

[1] The question of how precisely to choose the bandwidth is discussed in Section 2.7. For the moment, it is sufficient to note that $\hat{h}$ should go to 0 as $m$ goes to $\infty$, but not too fast, i.e., in such as way that $m\hat{h}^d \rightarrow \infty$ as $m \rightarrow \infty$.

leave them as general. In this case, each Mean Shift iteration requires $O(q(n))$ time to compute; and assuming, as is the case in practice, that $O(1)$ iterations are required for convergence, then the cost of running the algorithm on all $n$ data points (i.e. the overall data reduction algorithm) is then

$$T_{orig}(n) = O(p(n) + nq(n))$$

How much faster is our proposed algorithm? Looking back at Section 2.4, we may break down the complexity for each step. Step 1, the sampling step, is $O(m)$. We have already computed the complexity of Step 2, the Mean Shift step; this is simply the above expression, but specified for $m$ rather than $n$ samples, i.e. $O(p(m)+mq(m))$. In Step 3, we must map backwards; that is, for each of the $n$ original samples, we must find the closest amongst the $m$ new samples. Using our data structure (whose preprocessing time we have already accounted for in Step 2), this requires $O(nq(m))$. The total is then

$$\begin{aligned} T_{reduce}(n, m) &= O(m + p(m) + mq(m) + nq(m)) \\ &= O(p(m) + nq(m)) \end{aligned}$$

Comparing this with the expression for $T_{orig}(n)$, and since $n = \Omega(m)$, we have clearly reduced the complexity. By how much depends on the precise details of the data structure used, and we now attempt to deal with this issue.

In the simplest case, we have no special data structure. Then $p(n) = 0$, and the query time is linear in the number of elements $q(n) = O(n)$. In this case, $T_{orig}(n) = O(n^2)$, while $T_{reduce}(m, n) = O(nm)$, so the speed-up is a factor of $n/m$. In practical cases of interest $n/m > 100$, so this is quite an impressive speed-up.

Now, we may use more complex data structures for search. Voronoi Diagrams are not very useful when the dimension in which the data lives is $d > 3$ or so, as the preprocessing time is $p(n) = O(n^{\lceil d/2 \rceil})$. Other popular data structures, such as kd-trees, also have space complexity exponential in $d$ [2]. There are a variety of approximate nearest-neighbour algorithms, such as those based on locality sensitive hashing [2], which can lead to more efficient searches, if we are willing to find neighbours which are close, but not the closest. Indeed, if we choose an approximation factor of $c \geq 1$ – that is, we find points whose distance from the query are within a factor of $c$ of the closest – then we will have $T_{orig}(n) = O(n^{1+1/c^2})$, and $T_{reduce}(n, m) = O(nm^{1/c^2})$, leading to a speed-up of $O((n/m)^{1/c^2})$; see for example [2]. Typically, we might take $c = 1.5$, leading to a speed-up of about 10 if $n/m \approx 100$. (Note, however, that such data structures often have excellent theoretical properties while being somewhat more difficult to implement in practice, see [2].)

## 2.6. Variant: Soft Segmentation or Cartoons

We propose the following variant to the Fast Mean Shift clustering algorithm, which leads effectively to a soft segmentation. Note that only the third step has changed; we reproduce the first two for clarity.

1. **Sampling:** Take $m$ samples of the density $f$ to yield $\{\hat{x}_j\}_{j=1}^m$. Form the new density $\hat{f}(x) = \sum_{j=1}^m K_{\hat{h}}(x, \hat{x}_j)$.

2. **Mean Shift:** Perform Mean Shift on each of the $m$ samples: $\hat{x}_j \to \hat{M}^\infty(\hat{x}_j)$. Here, $\hat{M}$ indicates that we use $\hat{f}$ (rather than $f$) for the Mean Shift.

3. **Weighted Map Backwards:** For each $x_i$ and each $\hat{x}_j$, compute a weight between them according to $w_{ij} \propto K_h(x_i, x_j)$, such that the weights sum to 1: $\sum_j w_{ij} = 1$. Then $x_i \to \sum_{j=1}^m w_{ij} \hat{M}^\infty(\hat{x}_j)$.

Such a procedure will yield, instead of a piecewise constant segmentation, a piecewise smooth or cartoon-like segmentation.[2] This is similar in spirit to the Mumford-Shah scheme [16], which seeks a piecewise smooth approximation to the original image. We show examples of both soft and hard segmentation in Section 3.

## 2.7. The Optimal Bandwidth

Let us return to the expression for the $L_2$ distance between the original KDE $f$ and the reduced KDE $\hat{f}$ as given in Equation (2). Suppose we look at the asymptotic case, where $m$ is sufficiently large and $\hat{h}$ is sufficiently small; in this case, the upper bound on $J$ is well-approximated by

$$L = 4A\hat{h} + \frac{B}{m\hat{h}^d}$$

Now, we may try to find the bandwidth[3] which minimizes $L$, taking $m$ fixed; $L$ is convex in $\hat{h}$, and setting its derivative to 0 yields

$$\hat{h}^*(m) = C_1 m^{-\frac{1}{d+1}} \quad \text{and} \quad L^*(m) = C_2 m^{-\frac{1}{d+1}}$$

where $C_1$ and $C_2$ are (uninteresting) constants. Given the original bandwidth $h$, this gives us a simple way of choosing $\hat{h}$ for the reduced KDE; byy eliminating the constant $C_1$, we get

$$\hat{h} = (n/m)^{\frac{1}{d+1}} h$$

[2]Note that if the kernel has finite support, then for a particular choice of $h$ it is possible that all weights will be 0. Thus, soft segmentation using a kernel with infinite support, such as a Gaussian, is recommended.

[3]Note that our choice of bandwidth here is a scalar, yielding a diagonal covariance. For work which tries to find an adaptive non-diagonal bandwidth matrix from the data, see [4].

| | Parrot | Castle | Girl |
|---|---|---|---|
| Image Size | $844 \times 636$ | $960 \times 784$ | $551 \times 735$ |
| Sampling $m/n$ | 1024 | 1024 | 1024 |
| Speed-up Factor | 401 | 1160 | 1543 |

Table 1. Timing results for image segmentation.

# 3. Segmentation Experiments

## 3.1. Image Segmentation

In the image segmentation experiment, we compare the performance of the proposed fast Mean Shift method with standard Mean Shift on three images. The feature vectors are taken to be 5-dimensional, and are constructed from the three Lab colour vectors of the individual pixels in the image, and of the two corresponding spatial coordinates ($x$ and $y$). Table 1 summarizes the timing information, and the images themselves are shown in Figure 1. In each example, the sampling factor was taken to $n/m = 1024$. Our complexity analysis indicates that we ought to expect a speed-up factor of about $1024$; in fact, despite using fast nearest-neighbour queries (based on kd-trees, which despite the arguments in [2] tend to work well in low dimensions), the speed-up factor in two of the three cases exceeded this sampling factor, as shown in Table 1.

In Figure 1, we show the results of the standard Mean Shift procedure (second column), the Fast Mean Shift procedure (third column), and the soft segmentation variant of the Fast Mean Shift procedure (fourth column). Clearly, hard segmentation results obtained using the original and the fast Mean Shift methods look quite similar. As expected, the soft segmentation is kind a piecewise smooth cartoon of the original image, with small details removed from the original image. Therefore it may be suitable for noise reduction tasks.

## 3.2. Varying the Sampling Factor $n/m$

In order to understand the effect of varying the sampling factor on segmentation, we ran the fast Mean Shift algorithm with various sampling factors $n/m$ on the parrot image from the previous section. The images are shown in Figure 2. Note that the algorithm does relatively well even up to a sampling factor of $n/m = 4,096$, as shown in (d); as the image itself contains a bit more than $5 \times 10^5$ pixels, this corresponds to using only $m = 131$ randomly selected samples to construct the KDE, which is quite remarkable. The fast algorithm fails at $n/m = 16,384$; this is not surprising, as this corresponds to using only 32 randomly selected samples to construct the KDE. Indeed, at this high a subsampling rate, the algorithm produces quite different segmentations from one run to the next.

## 3.3. Video Segmentation

Figure 3 shows results of video segmentation on a $10$ frame sequence, for two consecutive frames from a video sequence (see upper row). In this experiment we compared the use of our fast Mean Shift method in two configurations. In the first one (the middle row), a frame by frame segmentation was performed. In the second configuration, data from a time window of 10 frames was combined into a single large data set, and used to perform the first two steps (the sampling, and the Mean Shift) in the algorithm of Section 2.4, using a subsampling factor of $1024$. (Note that in this example, it is possible to use a subsampling factor of $5000$, though the results are slightly less clean.) Then, the third step (the mapping backwards) from Section 2.4 was applied to each frame. The results of the "windowed" version look cleaner; for example, this version does not misclassify the color of the aircraft, as is the case in the frame-by-frame procedure. Another option is to use the time axis as an additional feature axis, which might be useful for applications such as target tracking. Clearly, the proposed fast method opens up some new possibilities for dealing with large spatio-temporal data sets.

# 4. Application: Graph Hierarchies

In this section, we show how to use the Fast Mean Shift algorithm in order to construct a hierarchy of graphs corresponding to a particular image. Having a multiscale structure on image graphs can potentially be very useful, due to the number of computer vision problems which can be posed as graph problems, and solved using graph algorithms, such as graph cuts [15]. Examples of these problems include stereo, semi-automatic segmentation, optical flow, as well as many problems that can be posed as Maximum a Posteriori estimation over Markov Random Fields. A multiscale graph hierarchy can be used to help speed up the solution to these problems in the usual multiscale way: the problem is first solved on the coarsest scale; the solution to this problem is then used to initialize the next scale up; and so on. In addition to making the solution faster, such as approach can also lead in some instances to more accurate solutions. See [1] for an example of another sort of graph hierarchy, based on algebraic multigrid techniques.

## 4.1. A Graph Hierarchy

We define a continuous graph hierarchy using Mean Shift as follows. As before, our data consists of the set of feature vectors $\{x_i\}_{i=1}^n$, where in most applications of interest, each data point corresponds to a pixel in an image. Denote by $f_h(x)$ the KDE with bandwidth matrix $\mathbf{H} = h^2\mathbf{I}$, and if $x$ is a mode of $f_h$, let $B(x)$ be the basin of attraction of $x$, i.e. $B(x) = \{y \in \mathbb{R}^d : M^\infty(y) = x\}$. The graph corresponding to bandwidth $h$ is denoted by

(a) (b) (c) (d)

Figure 1. Image segmentation example: (a) the original image (b) the result of standard Mean Shift segmentation (c) the result of fast Mean Shift segmentation (d) the soft segmentation variant of the fast Mean Shift. Note similar hard segmentation results in (b) and (c); the proposed method is more than 1000 times faster than the original Mean Shift.

(a) (b) (c) (d) (e)

Figure 2. Effect of varying the sampling factor $n/m$. (a) $n/m = 64$ (b) $n/m = 256$ (c) $n/m = 1,024$ (d) $n/m = 4,096$ (e) $n/m = 16,384$.

$G_h = (V_h, E_h, W_h)$, where

$$V_h = \{x \in \mathbb{R}^d : x \text{ is a mode of } f_h(\cdot)\}$$
$$E_h = \{(u, v) : u, v \in V(h), B(u) \cap B(v) \neq \emptyset\}$$

and the edge weights are given by

$$W_h(u, v) = \int_{x \in B(u)} \int_{y \in B(v)} K_h(x, y) dx dy$$

The definition of the vertex set and edge set is quite natural in terms of Mean Shift: the vertex set is just the set of modes, and the edge set is the set of pairs of modes whose basins of attractions are adjacent in the feature space. The edge weights are also defined in a reasonable fashion, by aggregating the similarity between points in neighbouring basins of attraction, though many other definitions are possible as well. In practice, we will approximate the integral in the edge weight definition by its corresponding sum.

It is clear that this continuous graph hierarchy has the following desirable properties:

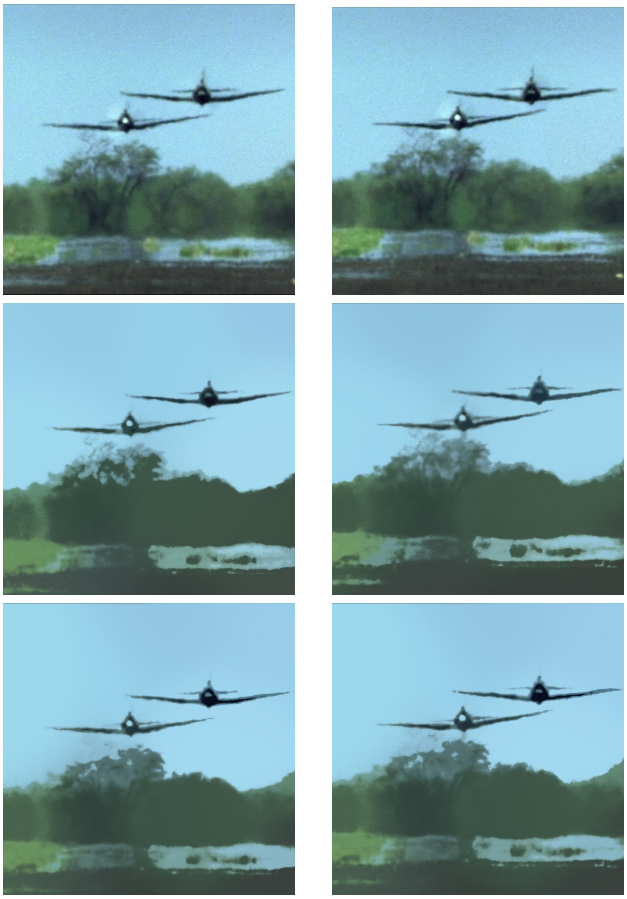


Figure 3. Video segmentation example. Upper row: two original frames from a video sequence. Middle row: the result of the frame-by-frame segmentation using the proposed method. Lower row: the result of the 10-frame window based segmentation using the proposed method.

**Theorem 3** *Given the graph hierarchy as defined above. Then the graph corresponding to* $h = 0$ *has the property that* $V_0 = \{x_i\}_{i=1}^n$*, while the graph corresponding to* $h = \infty$ *has the property that* $|V_\infty| = 1$.

**Proof:** See [9].

In other words, we move from a graph whose vertex set comprises the original pixels, to a graph with a single vertex. This is very much what we would expect from a multiscale hierarchy.

Now, to convert from a continuous graph hierarchy to a discrete one, we simply choose a fixed number of bandwidths, i.e. $G_\ell = G_{h_\ell}$, where the levels run from $\ell = 0$ (finest) to $\ell = L$ (coarsest). In particular, in $d$-dimensional space, it is natural to choose $h_\ell \approx 2^{\frac{1}{d}} h_{\ell-1}$, so that the *volume* of coverage doubles at each stage. In this case, we expect $L \approx \log n$; using a similar line of argument to that of Section 2.5, we may show that the computing this discrete graph hierarchy using ordinary Mean Shift requires $O(Ln^2) = O(n^2 \log n)$ time. (This assumes a naive nearest neighbour data structure, see Section 2.5 for a more in-depth discussion.)

To use the Fast Mean Shift algorithm, we make two observations. The first is that for the initial level ($\ell = 0$) of the hierarchy, we may use $m_0 \ll n$ samples; this is precisely what we have shown in Sections 2 and 3, from the theoretical and experimental points of view, respectively. The second observation is that for all subsequent levels, we may use even fewer samples, due to the increasing bandwidths of these levels. We have that $h_\ell \approx 2^{\frac{1}{d}} h_{\ell-1}$, and we may convert this statement about bandwidths to one about the number of samples to use at each level of the graph, using the result of Section 2.7 for bandwidth selection. In particular, if $m_\ell$ is the number of samples required at level $\ell$, this leads to $m_\ell \approx 2^{-\frac{d+1}{d}} m_{\ell-1}$, or for $d$ large, $m_\ell \approx \frac{1}{2} m_{\ell-1}$. Again, following the logic of Section 2.5, we have the complexity of the fast version of the discrete graph hierarchy as $O(\sum_{\ell=0}^{L} n 2^{-\ell} m_0) = O(n m_0)$.

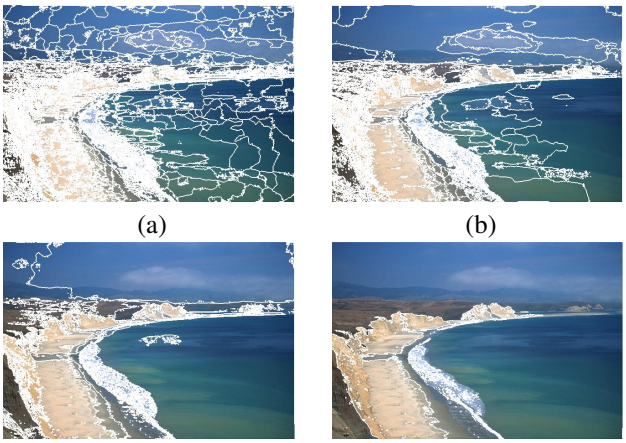
Note that the speed up factor in computing the graph hierarchy is even better than that of Mean Shift itself; it is $(n/m) \log n$, versus $n/m$ for the Mean Shift speed up. In cases of interest where $n \approx 10^7$ or $10^8$, the $\log n$ factor is not insubstantial (i.e. $20 - 25$).

### 4.2. An Example

In Figure 4, we show part of a graph hierarchy for an image of a seashore, computed using the fast technique described above. The four images we show are for bandwidths $h = 8, 16, 32$, and $64$. The number of vertices for these graphs are $266, 59, 11$, and $3$ respectively. We visualize the graphs by surrounding each contiguous region in white; note, however, that these contiguous regions are *not* the segments themselves, and often a segment (and hence a vertex) consists of multiple such contiguous regions. This is due to the fact that the graphs are not necessarily planar. This is true in our example, as our feature vector is colour (Lab) rather than texture, without the addition of position; thus, the sandy textured region looks as though it contains many nodes, whereas in fact it contains only a few non-contiguous nodes. Even with this effect, one can see the graph simplification as the bandwidth increases.

## 5. Conclusions

We have presented a fast version of the Mean Shift algorithm, based on computing a pared down KDE using a sampling procedure. We have theoretically demonstrated the closeness of the pared down KDE to the original KDE, as well as the superior complexity of the fast Mean Shift algorithm compared to the standard Mean Shift. We have experimentally verified the algorithm's speed, and shown its potential utility in clustering large data sets, such as video,



(a) (b) (c) (d)

Figure 4. A visualization of the graph hierarchy. (a) $h = 8$, $|V| = 226$ (b) $h = 16$, $|V| = 59$ (c) $h = 32$, $|V| = 11$ (d) $h = 64$, $|V| = 3$. See accompanying discussion in text.

and in the computation of complex data structures such as the graph hierarchy presented. It is our hope that the algorithm can find immediate application in fast segmentation of existing large data sets, such as medical images.

## References


[1] S. Alpert, M. Galun, R. Basri, and A. Brandt. Image Segmentation by Probabilistic Bottom-Up Aggregation and Cue Integration. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8, 2007.

[2] A. Andoni and P. Indyk. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Communications of the ACM*, 51(1):117, 2008.

[3] D. Barash and D. Comaniciu. A common framework for nonlinear diffusion, adaptive smoothing, bilateral filtering and mean shift. *Image and Vision Computing*, 22(1):73–81, 2004.

[4] D. Comaniciu. An Algorithm for Data-Driven Bandwidth Selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 281–288, 2003.

[5] D. Comaniciu and P. Meer. Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 603–619, 2002.

[6] D. Comaniciu and V. Ramesh. Mean shift and optimal prediction for efficient object tracking. In *Proceedings of the International Conference on Image Processing*, volume 3, 2000.

[7] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-Based Object Tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 564–577, 2003.

[8] D. Dementhon. Spatio-temporal segmentation of video by hierarchical mean shift analysis. In *Center for Automat. Res., U. of Md, College Park*, 2002.

[9] D. Freedman and P. Kisilev. Fast Mean Shift. Technical report, Hewlett-Packard Laboratories, 2009.

[10] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, 1975.

[11] B. Georgescu, I. Shimshoni, and P. Meer. Mean shift based clustering in high dimensions: A texture classification example. In *International Conference on Computer Vision*, pages 456–463, 2003.

[12] I. Grabec. Self-organization of neurons described by the maximum-entropy principle. *Biological Cybernetics*, 63(5):403–409, 1990.

[13] H. Guo, P. Guo, and H. Lu. A Fast Mean Shift Procedure with New Iteration Strategy and Re-sampling. In *IEEE International Conference on Systems, Man and Cybernetics, 2006. SMC'06*, volume 3, 2006.

[14] K. Kim, K. Jung, and J. Kim. Texture-based approach for text detection in images using support vector machines and continuously adaptive mean shift algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(12):1631–1639, 2003.

[15] V. Kolmogorov and R. Zabih. What Energy Functions Can Be Minimized via Graph Cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 147–159, 2004.

[16] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Comm. Pure Appl. Math*, 42(5):577–685, 1989.

[17] S. Paris and F. Durand. A topological approach to hierarchical segmentation using mean shift. In *Proc. CVPR*, 2007.

[18] H. Traven. A neural network approach to statistical pattern classification bysemiparametric'estimation of probability density functions. *Neural Networks, IEEE Transactions on*, 2(3):366–377, 1991.

[19] A. Vedaldi and S. Soatto. Quick shift and kernel methods for mode seeking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2008.

[20] J. Wang, B. Thiesson, Y. Xu, and M. Cohen. Image and Video Segmentation by Anisotropic Kernel Mean Shift. *Lecture Notes in Computer Science*, pages 238–249, 2004.

[21] P. Wang, D. Lee, A. Gray, and J. Rehg. Fast mean shift with accurate and stable convergence. In *Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2007.

[22] J. Wu and C. Chan. A three-layer adaptive network for pattern density estimation and classification. *International Journal of Neural Systems*, 2(3):211–220, 1991.

[23] C. Yang, R. Duraiswami, N. Gumerov, and L. Davis. Improved fast Gauss transform and efficient kernel density estimation. In *Ninth IEEE International Conference on Computer Vision, 2003. Proceedings*, pages 664–671, 2003.