Improved Fast Gauss Transform and Efficient Kernel Density Estimation

Changjiang Yang, Ramani Duraiswami, Nail A. Gumerov and Larry Davis
Perceptual Interfaces and Reality Laboratory
University of Maryland, College Park, MD 20742, USA
{yangcj,ramani,gumerov,lsd}@umiacs.umd.edu

Abstract

Evaluating sums of multivariate Gaussians is a common computational task in computer vision and pattern recognition, including in the general and powerful kernel density estimation technique. The quadratic computational complexity of the summation is a significant barrier to the scalability of this algorithm to practical applications. The fast Gauss transform (FGT) has successfully accelerated the kernel density estimation to linear running time for low-dimensional problems. Unfortunately, the cost of a direct extension of the FGT to higher-dimensional problems grows exponentially with dimension, making it impractical for dimensions above 3. We develop an improved fast Gauss transform to efficiently estimate sums of Gaussians in higher dimensions, where a new multivariate expansion scheme and an adaptive space subdivision technique dramatically improve the performance. The improved FGT has been applied to the mean shift algorithm achieving linear computational complexity. Experimental results demonstrate the efficiency and effectiveness of our algorithm.

1 Introduction

In most computer vision and pattern recognition applications, the feature space is complex, noisy and rarely can be described by the common parametric models [7], since the forms of the underlying density functions are in general unknown. In particular, data in high-dimensional feature space becomes more sparse and scattered, making it much more difficult to fit them with a single high-dimensional density function. By contrast, without the assumption that the form of the underlying densities are known, nonparametric density estimation techniques [22, 20] have been widely used to analyze arbitrarily structured feature spaces.

The most widely studied and used nonparametric technique is kernel density estimation (KDE), first introduced by Rosenblatt [22], then discussed in detail by Parzen [20] and Cacoullos [3]. In this technique the density function is estimated by a sum of kernel functions (typically Gaussians) centered at the data points. A bandwidth associated with the kernel function is chosen to control the smoothness of the estimated densities. In general, more data points allow a narrower bandwidth and a better density estimate.

Many approaches in computer vision and pattern recognition use kernel density estimation, including support vector machines [23], M-estimation [18], normalized cut [24] and mean shift analysis [5]. With enough samples, the kernel density estimates provably converge to any arbitrary density function. On the other hand, the number of samples needed may be very large and much greater than would be required for parametric models. Moreover, the demand for large number of samples grows rapidly with the dimension of the feature space. Given N source data points, the direct evaluation of densities at M target points takes $O(MN)$ time. The large dataset also leads to severe requirements for computational time and/or storage.

Various methods have been proposed to make the process of kernel density estimation more efficient. The existing approaches can be roughly divided into two categories. One is based on the k-nearest neighbor searching, where spatial data structures and/or branch and bound are employed to achieve the computational saving [21, 6, 10, 19]. One is based on the fast Fourier transform (FFT) for evaluating density estimates on gridded data which, however, are unavailable for most applications [25]. Recently the fast multipole method (FMM) and fast Gauss transform (FGT) have been used to reduce the computational time of kernel density estimation to $O(M + N)$ time, where the data are not necessarily on grids [15, 8].

As faster computers and better video cameras become cheaper, the collection of sufficient data is becoming possible, which results in a steady increase in the size of the dataset and the number of the features. Unfortunately the existing approaches including the fast Gauss transform suffer from the curse of dimensionality. The complexity of computation and storage of the FGT grows exponentially with dimension. In this paper, we proposed an improved fast Gauss transform (IFGT) to efficiently evaluate the sum of Gaussians in higher dimensions. By higher dimensions,

we mean dimensions up to ten. Such high dimensional spaces are commonly used in many applications such as in video sequence analysis and eigenspace based approaches. We also show how the IFGT can be applied to the kernel density estimation. Specifically the mean shift algorithm [11, 4, 5] is chosen as a case study for the IFGT. The mean shift algorithm is based on the KDE and recently rediscovered as a robust clustering method. However, the mean shift algorithm suffers from the quadratic computational complexity, especially in higher dimensions. The proposed IFGT successfully reduced the computational complexity into linear time.

2 FMM and FGT

The fast Gauss transform introduced by Greengard and Strain [15, 26] is an important variant of the more general fast multipole method [13, 16]. Originally the FMM was developed for the fast summation of potential fields generated by a large number of sources, such as those arising in gravitational or electrostatic potential problems in two or three dimensions. Thereafter, this method was extended to other potential problems, such as those arising in the solution of the Helmholtz and Maxwell equations, those in chemistry and interpolation of scattered data [16].

2.1 Fast Multipole Method

We briefly describe the FMM here. Consider the sum

$$v(y_j) = \sum_{i=1}^{N} u_i \phi_i(y_j), \quad j = 1, \ldots, M. \tag{1}$$

Direct evaluation requires $O(MN)$ operations. In the FMM, we assume that the functions ϕ_i can be expanded in multipole (singular) series and local (regular) series that are centered at locations x_* and y_* as follows:

$$\phi(y) = \sum_{n=0}^{p-1} b_n(x_*) S_n(y - x_*) + \epsilon(p), \tag{2}$$

$$\phi(y) = \sum_{n=0}^{p-1} a_n(y_*) R_n(y - y_*) + \epsilon(p), \tag{3}$$

where S_n and R_n respectively are multipole (singular) and local (regular) basis functions, x_* and y_* are expansion centers, a_n, b_n are the expansion coefficients, and ϵ is the error introduced by truncating a possibly infinite series after p terms. The operation reduction trick of the FMM relies on expressing the sum (1) using the series expansions (2) and (3). Then the reexpansion for (3) is:

$$v(y_j) = \sum_{i=1}^{N} u_i \phi_i(y_j) = \sum_{i=1}^{N} u_i \sum_{n=0}^{p-1} c_{ni} R_n(y_j - x_*), \tag{4}$$

for $j = 1, \ldots, M$. A similar expression can be obtained for (2). Consolidating the N series into one p term series, by rearranging the order of summations, we get

$$v(y_j) = \sum_{n=0}^{p-1} \left[\sum_{i=1}^{N} u_i c_{ni} \right] R_n(y_j - x_*) = \sum_{n=0}^{p-1} C_n R_n(y_j - x_*). \tag{5}$$

The single consolidated p term series (5) can be evaluated at all the M evaluation points. The total number of operations required is then $O(Mp + Np) \simeq O(Np)$ for $N \sim M$. The truncation number p depends on the desired accuracy alone, and is independent of M, N.

The functions ϕ_i in the FMM are not valid over the whole domain. So the singular expansions (2) are generated around clusters of sources. In a fine-to-coarse pass, the generated coefficients are translated into coarser level singular expansions through a tree data structure by "translation" operators. In a coarse-to-fine pass, the coefficients of the singular expansions at coarser level are converted via a sequence of translations to coefficients of regular expansions at finer levels, then evaluated at each evaluation point.

2.2 Fast Gauss Transform

The fast Gauss transform was introduced in [15] for efficient computation of the weighted sum of Gaussians

$$G(y_j) = \sum_{i=1}^{N} q_i \, e^{-\|y_j - x_i\|^2 / h^2} \tag{6}$$

where q_i are the weight coefficients, $\{x_i\}_{i=1,\ldots,N}$ are the centers of the Gaussians (called "sources"), h is the bandwidth parameter of the Gaussians. The sum of the Gaussians is evaluated at each of the "target" points $\{y_j\}_{j=1,\ldots,M}$. Direct evaluation of the sum at M target points due to N sources requires $O(MN)$ operations.

The original FGT directly applies the FMM idea by using the following expansions for the Gaussian:

$$e^{-\|y - x_i\|^2 / h^2} = \sum_{n=0}^{p-1} \frac{1}{n!} \left(\frac{x_i - x_*}{h} \right)^n h_n\left(\frac{y - x_*}{h} \right) + \epsilon(p), \tag{7}$$

$$e^{-\|y - x_i\|^2 / h^2} = \sum_{n=0}^{p-1} \frac{1}{n!} h_n\left(\frac{x_i - y_*}{h} \right) \left(\frac{y - y_*}{h} \right)^n + \epsilon(p), \tag{8}$$

where the Hermite functions $h_n(x)$ are defined by

$$h_n(x) = (-1)^n \frac{d^n}{dx^n} \left(e^{-x^2} \right).$$

The two expansions (7) and (8) are identical, except that the arguments of the Hermite functions and the monomials (Taylor series) are flipped. The first is used as the counterpart of the multipole expansion, while the second is used as the local expansion. The FGT then uses these

expansions and applies the FMM mechanism to achieve its speedup. Conversion of a Hermite series into a Taylor series is achieved via a translation operation. The error bound estimate given by Greengard and Strain [15] is incorrect, and a new and more complicated error bound estimate was presented in [1].

The extension to higher dimensions was done by treating the multivariate Gaussian as a product of univariate Gaussians, applying the series factorizations (7) and (8) to each dimension. For convenience's sake, we adopt the multi-index notation of the original FGT papers [15]. A multi-index $\alpha = (\alpha_1, \ldots, \alpha_d)$ is a d-tuple of nonnegative integers. For any multi-index $\alpha \in \mathbf{N}^d$ and any $x \in \mathbf{R}^d$, we have the monomial

$$x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d}.$$

The length and the factorial of α are defined as

$$|\alpha| = \alpha_1 + \alpha_2 + \ldots + \alpha_d, \qquad \alpha! = \alpha_1! \alpha_2! \cdots \alpha_d!.$$

The multidimensional Hermite functions are defined by

$$h_\alpha(x) = h_{\alpha_1}(x_1) h_{\alpha_2}(x_2) \cdots h_{\alpha_d}(x_d).$$

The sum (6) is then equal to the Hermite expansion about center x_*:

$$G(y_j) = \sum_{\alpha \geq 0} C_\alpha h_\alpha \left(\frac{y_j - x_*}{h} \right), \tag{9}$$

where the coefficients C_α are given by

$$C_\alpha = \frac{1}{\alpha!} \sum_{i=1}^{N} q_i \left(\frac{x_i - x_*}{h} \right)^\alpha. \tag{10}$$

The FGT in higher dimensions is then just an accumulation of the product of the Hermite expansions along each dimension. If we truncate each of the Hermite series after p terms (or equivalently order $p-1$), then each of the coefficients C_α is a d-dimensional matrix with p^d terms. The total computational complexity for a single Hermite expansion is $O((M+N)p^d)$. The factor $O(p^d)$ **grows exponentially** as the dimensionality d increases. Despite this defect in higher dimensions, the FGT is quite effective for two and three-dimensional problems, and has already achieved success in some physics, computer vision and pattern recognition problems [14, 8].

Another serious defect of the original FGT is the use of the box data structure. The original FGT subdivides the space into boxes using a uniform mesh. However, such a simple space subdivision scheme is not appropriate in higher dimensions, especially in applications where the data might be clustered on low dimensional manifolds. First of all, it may generate too many boxes (largely empty) in higher dimensions to store and manipulate. Suppose the unit box in 10 dimensional space is divided into tenths along each dimension, there are 10^{10} boxes which may cause trouble in storage and waste time on processing empty boxes. Secondly, and more importantly, having so many boxes makes it more difficult for searching nonempty neighbor boxes. Finally, and most importantly the worst property of this scheme is that the ratio of volume of the hypercube to that of the inscribed sphere grows exponentially with dimension. In other words, the points have a high probability of falling into the area inside the box and outside the sphere. The truncation error of the above Hermite expansions (7) and (8) are much larger near the boundary than near the expansion center, which will bring large truncation errors on most of the points.

In brief, the original FGT suffers from the following two defects that are the motivation behind this paper:

1. The exponential growth of complexity with dimensionality.
2. The use of the box data structure in the FGT is inefficient in higher dimensions.

3 Improved Fast Gauss Transform

3.1 A Different Factorization

The defects listed above can be thought as a result of applying the FMM methodology to the FGT blindly. As shown in section 2, the FMM was developed for singular potential functions whose forces are long-ranged and nonsmooth (at least locally), hence it is necessary to make use of the tree data structures, multipole expansions, local expansions and translation operators. In contrast, the Gaussian is far from singular — it is infinitely differentiable! There is no need to perform the multipole expansions which account for the far-field contributions. Instead we present a simple new factorization and space subdivision scheme for the FGT. The new approach is based on the fact that the Gaussian, especially in higher dimensions, decays so rapidly that the contributions outside of a certain radius can be safely ignored.

Assuming we have N sources $\{x_i\}$ centered at x_* and M target points $\{y_j\}$, we can rewrite the exponential term as

$$e^{-\|y_j - x_i\|^2/h^2} = e^{-\|\Delta y_j\|^2/h^2} \; e^{-\|\Delta x_i\|^2/h^2} \; e^{2\Delta y_j \cdot \Delta x_i / h^2}, \tag{11}$$

where $\Delta y_j = y_j - x_*, \Delta x_i = x_i - x_*$. In expression (11) the first two exponential terms can be evaluated individually at either the source points or the target points. The only problem left is to evaluate the last term where sources and targets are entangled. One way of breaking the entanglement is to expand it into the series

$$e^{2\Delta y_j \cdot \Delta x_i / h^2} = \sum_{n=0}^{\infty} \Phi_n(\Delta y_j) \Psi_n(\Delta x_i), \tag{12}$$

where Φ_n and Ψ_n are the expansion functions and will be defined in the next section. Denoting $\phi(\Delta y_j) = e^{-\|\Delta y_j\|^2/h^2}$, $\psi(\Delta x_i) = e^{-\|\Delta x_i\|^2/h^2}$, we can rewrite the sum (6) as

$$G(y_j) = \sum_{i=1}^{N} q_j \phi(\Delta y_j)\psi(\Delta x_i) \sum_{n=0}^{\infty} \Phi_n(\Delta y_j)\Psi_n(\Delta x_i). \tag{13}$$

If the infinite series (12) absolutely converges, we can truncate it after p terms so as to obtain a desired precision. Exchanging the summations in (13), we obtain

$$G(y_j) = \phi(\Delta y_j) \sum_{n=0}^{p-1} C_n \Phi_n(\Delta y_j) + \epsilon(p), \tag{14}$$

$$C_n = \sum_{i=1}^{N} q_i \psi(\Delta x_i)\Psi_n(\Delta x_i). \tag{15}$$

The factorization (14) is the basis of our algorithm. In the following sections, we will discuss how to implement it in an efficient way.

3.2 Multivariate Taylor Expansions

The key issue to speed up the FGT is to reduce the factor p^d in the computational complexity. The factor p^d arises from the way that the multivariate Gaussian is treated as the product of univariate Gaussian functions and expanded along each dimension. To reduce this factor, we treat the dot product in (12) as a *scalar variable* and expand it via the Taylor expansion. The expansion functions Φ and Ψ are expressed as multivariate polynomials.

We denote by Π_n^d the space of all real polynomials in d variables of total degree less than or equal to n; its dimensionality is $r_{nd} = \binom{n+d}{d}$. To store, manipulate and evaluate the multivariate polynomials, we consider the monomial representation of polynomials. A polynomial $p \in \Pi_n^d$ can be written as

$$p(x) = \sum_{|\alpha| \le n} C_\alpha x^\alpha, \qquad C_\alpha \in \mathbf{R}. \tag{16}$$

It is computationally convenient and efficient to stack all the coefficients into a vector. To store all the r_{nd} coefficients C_α in a vector of length r_{nd}, we sort the coefficient terms according to *Graded lexicographic order*. "Graded" refers to the fact that the total degree $|\alpha|$ is the main criterion. Graded lexicographic ordering means that the multi-indices are arranged as

$$(0,0,\ldots,0),(1,0,\ldots,0),(0,1,\ldots,0),\ldots,(0,0,\ldots,1),$$
$$(2,0,\ldots,0),(1,1,\ldots,0),\ldots,(0,0,\ldots,2),\ldots,(0,0,\ldots,n).$$

The power of the dot product of two vectors x and y can be expanded into multivariate polynomial:

$$(x \cdot y)^n = \sum_{|\alpha|=n} \binom{n}{\alpha} x^\alpha y^\alpha, \tag{17}$$

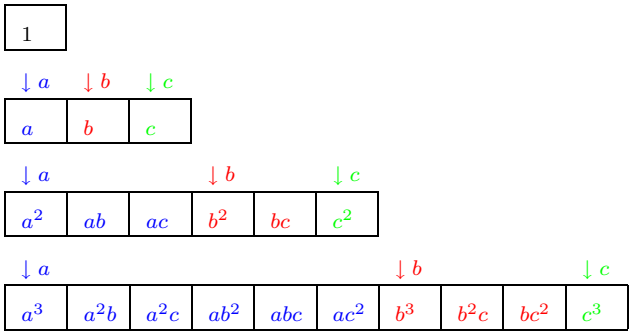

Figure 1: Efficient expansion of the multivariate polynomials. The arrows point to the leading terms.

where $\binom{n}{\alpha} = \frac{n!}{\alpha_1!\cdots\alpha_d!}$ are the multinomial coefficients. So we have the following multivariate Taylor expansion of the Gaussian functions:

$$e^{2x\cdot y} = \sum_{\alpha \ge 0} \frac{2^{|\alpha|}}{\alpha!} x^\alpha y^\alpha. \tag{18}$$

From Eqs.(11), (14) and (18), the weighted sum of Gaussians (6) can be expressed as a multivariate Taylor expansions about center x_*:

$$G(y_j) = \sum_{\alpha \ge 0} C_\alpha e^{-\|y_j - x_*\|^2/h^2} \left(\frac{y_j - x_*}{h}\right)^\alpha, \tag{19}$$

where the coefficients C_α are given by

$$C_\alpha = \frac{2^{|\alpha|}}{\alpha!} \sum_{i=1}^{N} q_i e^{-\|x_i - x_*\|^2/h^2} \left(\frac{x_i - x_*}{h}\right)^\alpha. \tag{20}$$

If we truncate the series after total degree $p-1$, the number of the terms $r_{p-1,d} = \binom{p+d-1}{d}$ is much less than p^d in higher dimensions (as shown in Table 1). For instance, when $d = 12$ and $p = 10$, the original FGT needs 10^{12} terms, the multivariate Taylor expansion needs only 293930. For $d \longrightarrow \infty$ and moderate p, the number of terms becomes $O(d^p)$, a substantial reduction.

One of the benefits of the graded lexicographic order is that the expansion of multivariate polynomials can be computed efficiently. For a d-variate polynomial of order n, we can store all terms in a vector of length r_{nd}. Starting from the order zero term (constant 1), we take the following steps recursively. Assume we have already evaluated terms of order $k-1$. Then terms of order k can be obtained by multiplying each of the d variables with all the terms between the variable's *leading term* and the end, as shown in the Figure 1. The required storage is r_{nd} and the computations of the terms require $r_{nd} - 1$ multiplications.

3.3 Spatial Data Structures

As discussed above, we need to subdivide space into cells and collect the influence of the sources within each

Table 1: Number of terms in d-variate Taylor expansion truncated after order $p-1$.

$p\backslash d$	1	2	3	4	5	6	7	8	9	10	11	12
4	4	10	20	35	56	84	120	165	220	286	364	455
5	5	15	35	70	126	210	330	495	715	1001	1365	1820
6	6	21	56	126	252	462	792	1287	2002	3003	4368	6188
7	7	28	84	210	462	924	1716	3003	5005	8008	12376	18564
8	8	36	120	330	792	1716	3432	6435	11440	19448	31824	50388
9	9	45	165	495	1287	3003	6435	12870	24310	43758	75582	125970
10	10	55	220	715	2002	5005	11440	24310	48620	92378	167960	293930

cell. The influence on each target can be summarized from its neighboring cells that lie within a certain radius from the target. To efficiently subdivide the space, we need to devise a scheme that adaptively subdivides the space according to the distribution of points. It is also desirable to generate cells as compact as possible.

Based on the above considerations, we model the space subdivision task as a k-center problem, which is defined as follows: given a set of n points and a predefined number of the clusters k, find a partition of the points into clusters $S_1, \ldots, S_k$, and also the cluster centers $c_1, \ldots, c_k$, so as to minimize the cost function — the maximum radius of clusters:

$$\max_i \max_{v \in S_i} \|v - c_i\|.$$

The k-center problem is known to be NP-hard [2]. Gonzalez [12] proposed a very simple greedy algorithm, called *farthest-point clustering*. Initially pick an arbitrary point v_0 as the center of the first cluster and add it to the center set C. Then for $i = 1$ to k do the follows: in iteration i, for every point, compute its distance to the set C: $d_i(v, C) = \min_{c \in C} \|v - c\|$. Let v_i be a point that is farthest away from C, i.e., a point for which $d_i(v_i, C) = \max_v d_i(v, C)$. Add v_i to set C. Report the points $v_0, v_1, \ldots, v_{k-1}$ as the cluster centers. Each point is assigned to its nearest center.

Gonzalez [12] proved that farthest-point clustering is a 2-approximation algorithm which computes a partition with maximum radius at most twice the optimum. The proof uses no geometry beyond the triangle inequity, so it hold for any metric space. Hochbaum and Shmoys [17] proved that the factor 2 cannot be improved unless $P = NP$. The direct implementation of farthest-point clustering has running time $O(nk)$. Feder and Greene [9] give a two-phase algorithm with optimal running time $O(n \log k)$.

The predefined number of clusters k can be determined as follows: run the farthest-point algorithm until the maximum radius of clusters decreases to a given distance. In practice, the initial point has little influence on the final radius of the approximation, if number of the points n is sufficiently large. Figure 2 displays the results of farthest-point algorithm. In two dimensions, the algorithm leads to a Voronoi tessellation of the space. In three dimensions, the partition boundary resembles the surface of a crystal.

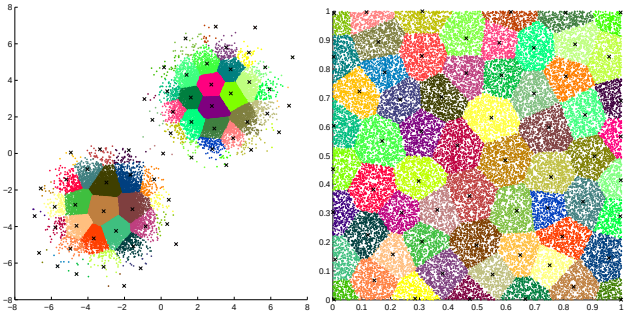

Figure 2: The farthest-point algorithm divides 40000 points into 64 clusters (with the centers indicated by the crosses) in 0.48 seconds on a 900MHZ PIII PC. Left: 2 normal distributions; Right: Uniform distribution.

3.4 The Algorithm

The improved fast Gauss transform consists of the following steps:

Step 1 Assign the N sources into K clusters using the farthest-point clustering algorithm such that the radius is less than $h\rho_x$.

Step 2 Choose p sufficiently large such that the error estimate (24) in appendix is less than the desired precision ϵ.

Step 3 For each cluster S_k with center c_k, compute the coefficients given by the expression (20):

$$C_\alpha^k = \frac{2^{|\alpha|}}{\alpha!} \sum_{x_i \in S_k} q_i e^{-\|x_i - c_k\|^2/h^2} \left(\frac{x_i - c_k}{h}\right)^\alpha.$$

Step 4 Repeat for each target y_j, find its neighbor clusters whose centers lie within the range $h\rho_y$. Then the sum of Gaussians (6) can be evaluated by the expression (19):

$$G(y_j) = \sum_{\|y_j - c_k\| \le h\rho_y} \sum_{|\alpha| < p} C_\alpha^k e^{-\|y_j - c_k\|^2/h^2} \left(\frac{y_j - c_k}{h}\right)^\alpha.$$

The amount of work required in step 1 is $O(NK)$ (for large K, we can use Feder and Greene's $O(N \log K)$ algorithm [9] instead). The amount of work required in step 3 is of $O(N\ r_{pd})$. The work required in step 4 is $O(Mn\ r_{pd})$, where n is the maximum number of the neighbor clusters for each target. For most nonparametric statistics, computer

vision and pattern recognition applications, the precision required is moderate, we can get small K and small r_{pd}. Since $n \leq K$, the improved fast Gauss transform achieves linear running time. The algorithm needs to store the K coefficients of size r_{pd}, so the storage complexity is reduced to $O(Kr_{pd})$.

4 Mean Shift Analysis with IFGT

Segmentation using the mean shift analysis is chosen as a case study for the IFGT. Mean shift is a clustering technique based on kernel density estimation, which is very effective and robust for the analysis of complex feature spaces. The mean shift procedure employing a Gaussian kernel converges to the stationary point following a smooth trajectory, which is theoretically important for convergence [5]. In practice, the quality of the results almost always improves when the Gaussian kernel is employed. Despite its superior performance, the Gaussian kernel is not as widely used in mean shift as it should be. In part this may be due to the high computational costs which we try to alleviate in the paper.

Given n data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in the d-dimensional space R^d, the *kernel density estimator* with kernel function $K(\mathbf{x})$ and a window bandwidth h, is given by [22, 20, 7]

$$\hat{f}_n(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right), \tag{21}$$

where the d-variate kernel $K(\mathbf{x})$ is nonnegative and integrates to one. The Gaussian kernel is a common choice.

The mean shift algorithm is a steepest ascent procedure which requires estimation of the density gradient:

$$\begin{aligned} \nabla \hat{f}_{h,K}(\mathbf{x}) &= \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{x}) g\left(\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2\right) \\ &= c_{k,g} \hat{f}_{h,G}(\mathbf{x}) \left[\frac{\sum_{i=1}^{n} \mathbf{x}_i g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)} - \mathbf{x}\right], \end{aligned} \tag{22}$$

where $g(x) = -k'_N(x) = \frac{1}{2} k_N(x)$ which can in turn be used as profile to define a Gaussian kernel $G(\mathbf{x})$. The kernel $K(\mathbf{x})$ is called the shadow of $G(\mathbf{x})$ [4]. Both have the same expression. $\hat{f}_{h,G}(\mathbf{x})$ is the density estimation with the kernel G. $c_{k,g}$ is the normalization coefficient. The last term is the *mean shift*

$$\mathbf{m}(\mathbf{x}) = \frac{\sum_{i=1}^{n} \mathbf{x}_i g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)} - \mathbf{x}, \tag{23}$$

which is proportional to the normalized density gradient and always points toward the steepest ascent direction of the function. The mean shift algorithm iteratively performs the following two steps till it reaches the stationary point:

- Computation of the mean shift vector $\mathbf{m}(\mathbf{x}^k)$.
- Updating the current position $\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{m}(\mathbf{x}^k)$.

The numerator in expression (23) is a weighted sum of Gaussians except that the weights are vectors. The denominator is a uniform weighted sum of Gaussians. So both can be evaluated by the improved fast Gauss transform as $d + 1$ independent weighted sums of Gaussians. The computation has been further reduced because they share the same space subdivisions and series expansions.

Table 2: Running times in milliseconds for direct evaluation, fast Gauss transform and improved fast Gauss transform in three dimensions.

Case	$N = M$	Direct	FGT	IFGT
1	100	2.9	5.5	4.6
2	200	11.4	13.0	12.5
3	400	46.1	37.0	21.1
4	800	184.2	121.8	33.2
5	1600	740.3	446.0	68.1
6	3200	2976.2	1693.8	132.8
7	6400	17421.4	6704.3	263.0
8	12800	68970.2	26138.6	580.2
9	25600	271517.9	103880.8	1422.0

5 Experimental Results

The first experiment compares the performance of our algorithm with the original fast Gauss transform. Since there is no practical fast Gauss transform in higher dimensions available, we only make comparisons in three dimensions. The sources and targets are uniformly distributed in a unit cube. The weights of the sources are uniformly distributed between 0 and 1. The bandwidth of the Gaussian is $h = 0.2$. We set the relative error bound to 2% which is reasonable for most kernel density estimation, because the estimated density function itself is an approximation. Table 2 reports the CPU times using direct evaluation, the original fast Gauss transform (FGT) and the improved fast Gauss transform (IFGT). All the algorithms are programmed in C++ and were run on a 900MHz PIII PC. We can find the running time of the IFGT grows linearly as the number of sources and targets increases, while the direct evaluation and the original FGT grows quadratically, though it is lower than the direct evaluation. The poor performance of the FGT in 3D is also reported in [8]. This is probably due to the fact that the number of boxes increases significantly by a uniform space subdivision in 3D, which makes the cost to compute the interactions between the boxes grow quadratically.

The second experiment is to examine the performance of IFGT in higher dimensions. We randomly generate the source and target points in a unit hypercube based on a uniform distribution. The weights of the sources are uniformly distributed between 0 and 1. The bandwidth is set to $h = 1$. The results are shown in Fig. 3. We compared the running

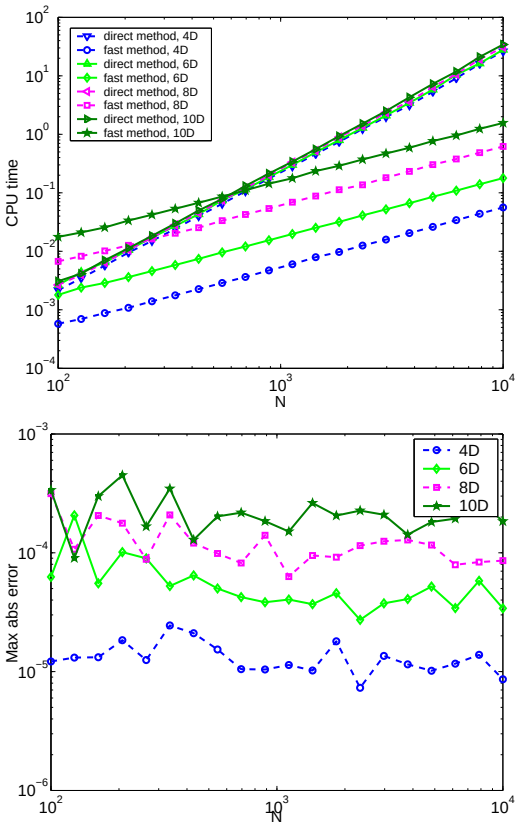

Figure 3: The running times in seconds (Top) and maximum absolute errors (Bottom) of the IFGT ($h = 1$) v.s. direct evaluation in dimensions $4, 6, 8, 10$.

time of the direct evaluation to the IFGT with $h = 1$ and $N = M = 100, \ldots, 10000$. The comparisons are performed in dimensions from 4 to 10 and results in dimensions 4, 6, 8, 10 are reported in Fig. 3. From the figure we notice that the running time of the direct evaluation grows quadratically with the size of points. The running time of the IFGT grows linearly with the size of the points. In 4, 6, 8, 10 dimensions, the IFGT takes 56ms, 406ms, 619 ms, 1568ms to evaluate the sums on 10000 points. The maximum relative absolute error as defined in [15] increases with the dimensionality but not with the number of points. The worst error occurs in dimension 10, and is below 10^{-3}. We can see that for a 10-D problem involving more than 700 Gaussians, the IFGT is faster than direct evaluation, while for a 4-D problem the IFGT is faster from almost the outset.

The third experiment is to apply the improved fast Gauss transform to the mean shift algorithm. We first transform the images to L*u*v* color space and normalize to a unit cube. Then we apply the mean shift algorithm with $h = 0.1$ to all the points in the 3D color space. After 5 iterations, the convergence points are grouped by a simple k-means algorithm [7]. We do not perform any postprocessing procedure as in [5]. The code is written in C++ with Matlab interfaces

Table 3: Image sizes *v.s.* the running time of the mean shift.

	House	*Cooking*	*Base Dive*	*Zebra*
Size	255x192	204x153	432x294	481x321
Time (s)	3.343	2.204	7.984	12.359

Figure 4: Segmentation results: (Right Column) The original images. (Left Column) Segmented images labelled with different colors. (Top Row) *House* image. (Second Row) *Cooking* image. (Third Row) *Base Dive* image. (Bottom Row) *Zebra* image.

and run on a 900MHz PIII PC. The results are shown in Fig. 4. The running time of the mean shift in seconds and the sizes of the images are shown in Table 3. The speed of our implementation is at least as fast as any reported. We find that the mean shift algorithm with the improved fast Gauss transform already achieves clear boundaries without any postprocessing. This is partly because that we apply the mean shift algorithm to all feature points without subsampling the feature space as in [5]. This leads to easily distinguishable valleys in our estimated densities. Another reason is that in our method the density evaluation at each target point has contributions from a much larger neighborhood because of the Gaussian kernel, which generates a smoother and better density estimate.

6 Conclusions

In this paper we present an improved fast Gauss transform to speed up the summations of Gaussians in higher dimensions. The success in acceleration of the FGT comes from two innovations: the use of the farthest-point algorithm to adaptively subdivide the high dimensional space, and the use of a new multivariate Taylor expansion we developed to dramatically reduce the computational and storage cost of the fast Gauss transform. The recursive computation of the multivariate Taylor expansion further reduces the computational cost and necessary storage.

The improved fast Gauss transform is applied to speed up the kernel density estimation which is the keystone for many applications in computer vision and pattern recognition. The general and powerful feature space analysis tool – the mean shift algorithm is chosen as a case study for the IFGT. Using the IFGT, we achieved a linear running time mean shift algorithm. Without using any heuristic vicinity information between the points, the mean shift based image segmentation achieve satisfactory results. In future work, we will study the capability of the IFGT in applications such as learning kernel classifiers (SVM), object tracking. We also plan to combine the IFGT with other techniques [27] to further improve the performance of mean shift algorithm.

Appendix: Error Bound of Improved FGT

The error due to the truncation of series (19) after order p and the cutoff error the satisfies the bound

$$|E(y)| \leq Q\left(\frac{2^p}{p!}\,\rho_x^p\rho_y^p + e^{-\rho_y^2}\right). \tag{24}$$

where $Q = \sum |q_j|$.

Acknowledgments

The support by the NSF under the awards 9987944, 0086075 and 0219681 is gratefully acknowledged.

References

[1] B. J. C. Baxter and G. Roussos. A new error estimate of the fast Gauss transform. *SIAM Journal on Scientific Computing*, 24(1):257–259, 2002.

[2] M. Bern and D. Eppstein. Approximation algorithms for geometric problems. In D. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 8, pages 296–345. PWS Publishing Company, Boston, 1997.

[3] T. Cacoullos. Estimation of a multivariate density. *Ann. Inst. Stat. Math.*, 18(2):179–189, 1966.

[4] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(8):790–799, 1995.

[5] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603 – 619, May 2002.

[6] L. Devroye and F. Machell. Data structures in kernel density estimation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 7(3):360–366, 1985.

[7] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, New York, 2nd edition, 2001.

[8] A. Elgammal, R. Duraiswami, and L. Davis. Efficient nonparametric adaptive color modeling using fast Gauss transform. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, Kauai, Hawaii, 2001.

[9] T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th ACM Symp. Theory of computing*, pages 434–444, Chicago, Illinois, 1988.

[10] K. Fukunaga and R. R. Hayes. The reduced Parzen classifier. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(4):423–425, 1989.

[11] K. Fukunaga and L. D. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Trans. Inform. Theory*, 21:32–40, 1975.

[12] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[13] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73(2):325–348, 1987.

[14] L. Greengard and J. Strain. A fast algorithm for the evaluation of heat potentials. *Comm. Pure Appl. Math.*, 43(8):949–963, 1990.

[15] L. Greengard and J. Strain. The fast Gauss transform. *SIAM J. Sci. Statist. Comput.*, 12(1):79–94, 1991.

[16] N. A. Gumerov, R. Duraiswami, and E. A. Borovikov. Data structures, optimal choice of parameters, and complexity results for generalized multilevel fast multipole methods in d dimensions. Technical Report UMIACS-TR-2003-28, UMIACS, University of Maryland, College Park, 2003.

[17] D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[18] P. Huber. *Robust Statistical Procedures*. SIAM, 2 edition, 1996.

[19] B. Jeon and D. A. Landgrebe. Fast Parzen density estimation using clustering-based branch and bound. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(9):950–954, 1994.

[20] E. Parzen. On estimation of a probability density function and mode. *Ann. Math. Stat.*, 33(3):1065–1076, 1962.

[21] J. G. Postaire and C. Vasseur. A fast algorithm for nonparametric probability density estimation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 4(6):663–666, 1982.

[22] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *Ann. Math. Stat.*, 27(3):832–837, 1956.

[23] B. Schlkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, Cambridge, MA, 2002.

[24] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.

[25] B. W. Silverman. Algorithm AS 176: Kernel density estimation using the fast Fourier transform. *Appl. Stat.*, 31(1):93–99, 1982.

[26] J. Strain. The fast Gauss transform with variable scales. *SIAM J. Sci. Statist. Comput.*, 12(5), 1991.

[27] C. Yang, R. Duraiswami, D. DeMenthon, and L. Davis. Mean-shift analysis using quasi-newton methods. In *Proc. Int'l Conf. Image Processing*, Barcelona, Spain, 2003.