Improved Fast Gauss Transform

Based on work by Changjiang Yang (2003),
Vikas Raykar (2005), and Vlad Morariu (2007)

Fast Gauss Transform (FGT)

- Originally proposed by Greengard and Strain (1991) to efficiently evaluate the weighted sum of Gaussians:

$$G(y_j) = \sum_{i=1}^{N} q_i e^{-\|y_j - x_i\|^2 / h^2}, \quad j = 1, \ldots, M$$

- FGT is an important variant of Fast Multipole Method (Greengard & Rokhlin 1987)

Fast Gauss Transform (cont'd)

- The weighted sum of Gaussians

$$G(y_j) = \sum_{i=1}^{N} q_i \, e^{-\|y_j - x_i\|^2/h^2}, \quad j = 1, \ldots, M.$$

Targets Sources

is equivalent to the matrix-vector product:

$$\begin{bmatrix} G(y_1) \\ G(y_2) \\ \vdots \\ G(y_M) \end{bmatrix} = \begin{bmatrix} e^{-\|x_1 - y_1\|^2/h^2} & e^{-\|x_2 - y_1\|^2/h^2} & \cdots & e^{-\|x_N - y_1\|^2/h^2} \\ e^{-\|x_1 - y_2\|^2/h^2} & e^{-\|x_2 - y_2\|^2/h^2} & \cdots & e^{-\|x_N - y_2\|^2/h^2} \\ \vdots & \vdots & \ddots & \vdots \\ e^{-\|x_1 - y_M\|^2/h^2} & e^{-\|x_2 - y_M\|^2/h^2} & \cdots & e^{-\|x_N - y_M\|^2/h^2} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_N \end{bmatrix}$$

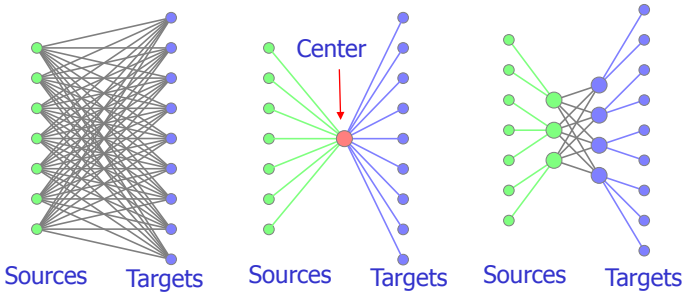
- Direct evaluation of the sum of Gaussians requires $O(N^2)$ operations.
- Fast Gauss transform reduces the cost to $O(N \log N)$ operations.

Fast Gauss Transform (cont'd)

- Three key components of the FGT:
  - The factorization or separation of variables
  - Space subdivision scheme
  - Error bound analysis

Factorization of Gaussian

- Factorization is one of the key parts of the FGT.
- Factorization breaks the entanglements between the sources and targets.
- The contributions of the sources are accumulated at the centers, then distributed to each target.



Factorization in FGT: Hermite Expansion

- The Gaussian kernel is factorized into Hermite functions

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

where *Hermite* function $h_n(x)$ is defined by

[redacted]

- Exchange the order of the summations

$$G(y_j) = \sum_{i=1}^{N} q_i \sum_{n=0}^{p-1} \frac{1}{n!} \left(\frac{x_i - x_*}{h}\right)^n h_n\left(\frac{y_j - x_*}{h}\right) + \epsilon(p),$$

$$= \sum_{n=1}^{p-1} \left[\frac{1}{n!} \sum_{i=1}^{N} q_i \left(\frac{x_i - x_*}{h}\right)^n\right] h_n\left(\frac{y_j - x_*}{h}\right) + \epsilon(p)$$

a.k.a. Moment, A_n

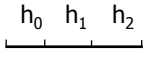
The FGT Algorithm

- Step 0: Subdivide the space into uniform boxes.
- Step 1: Assign sources and targets into boxes.
- Step 2: For each pair of source and target boxes, compute the interactions using one of the four possible ways:
  - $N_B \leq N_F$, $M_C \leq M_L$: Direct evaluation
  - $N_B \leq N_F$, $M_C > M_L$: Taylor expansion
  - $N_B > N_F$, $M_C \leq M_L$: Hermite expansion
  - $N_B > N_F$, $M_C > M_L$: Hermite expansion --> Taylor expansion
- Step 3: Loop through boxes evaluating Taylor series for boxes with more than M_L targets.

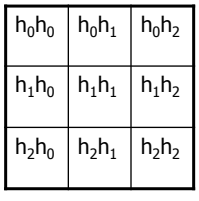
N_B: # of sources in Box B
M_C: # of targets in Box C
N_F: cutoff of Box B
M_L: cutoff of Box C

Too Many Terms in Higher Dimensions

- The higher dimensional Hermite expansion is the Kronecker product of d univariate Hermite expansions.
- Total number of terms is $O(p^d)$, p is the number of truncation terms.
- The number of operations in one factorization is $O(p^d)$.



d=1



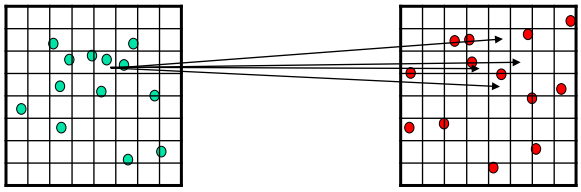
d=2

d=3

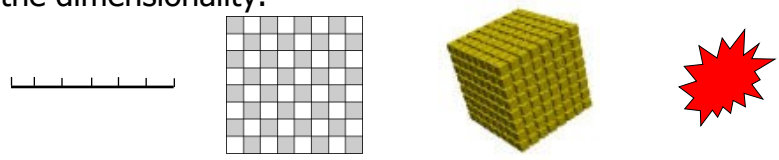
d>3

Too Many Boxes in Higher Dimensions

- The FGT subdivides the space into uniform boxes and assigns the source points and target points into boxes.
- For each box the FGT maintain a neighbor list.



- The number of the boxes increases exponentially with the dimensionality.

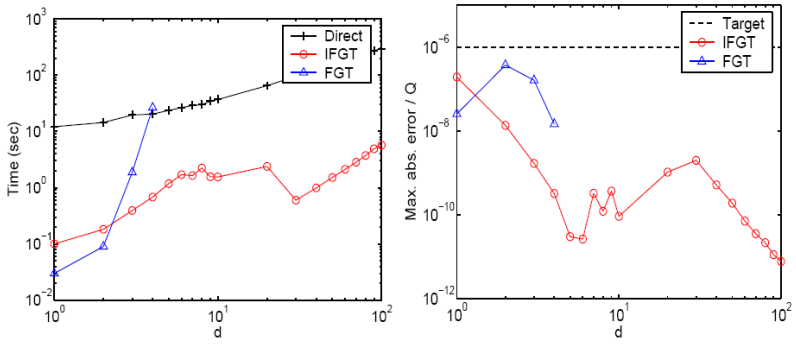


FGT in Higher Dimensions

- The FGT was originally designed to solve the problems in mathematical physics (heat equation, vortex methods, etc), where the dimension is up to 3.
- The higher dimensional Hermite expansion is the product of univariate Hermite expansion along each dimension. Total number of terms is $O(p^d)$.
- The space subdivision scheme in the original FGT is uniform boxes. The number of boxes grows exponentially with dimension. Most boxes are empty.
- The exponential dependence on the dimension makes the FGT extremely inefficient in higher dimensions.

Performance of the FGT degrades exponentially with increasing dimensionality, which makes it impractical for dimensions greater than three.

The *improved fast Gauss transform* or IFGT can handle higher dimensions.



Improved Fast Gauss Transform

- Multivariate Taylor expansion
- Multivariate Horner's Rule
- Space subdivision based on *k*-center algorithm
- Error bound analysis

New factorization

Separate out i and j

For any point $x_* \in \mathbf{R}^d$

$$\begin{aligned} G(y_j) &= \sum_{i=1}^{N} q_i e^{-\|y_j - x_i\|^2/h^2} \\ &= \sum_{i=1}^{N} q_i e^{-\|(y_j - x_*) - (x_i - x_*)\|^2/h^2}, \\ &= \sum_{i=1}^{N} q_i e^{-\|x_i - x_*\|^2/h^2} e^{-\|y_j - x_*\|^2/h^2} e^{2(y_j - x_*)\cdot(x_i - x_*)/h^2}. \end{aligned}$$

Crux of the algorithm

Separate this entanglement via the Taylor's series expansion of the exponentials.

Multivariate Taylor Expansions

- The Taylor expansion of the Gaussian function:

$$e^{-\|y_j - x_i\|^2/h^2} = e^{-\|y_j - x_*\|^2/h^2} e^{-\|x_i - x_*\|^2/h^2} e^{2(y_j - x_*)\cdot(x_i - x_*)/h^2},$$

- The first two terms can be computed independently.
- The Taylor expansion of the last term is:

$$e^{2(y_j - x_*)\cdot(x_i - x_*)/h^2} = \sum_{\alpha \geq 0} \frac{2^{|\alpha|}}{\alpha!} \left(\frac{x_i - x_*}{h}\right)^{\alpha} \left(\frac{y_j - x_*}{h}\right)^{\alpha}.$$

where $\alpha=(\alpha_1, \cdots, \alpha_d)$ is multi-index.

- The multivariate Taylor expansion about center x_*:

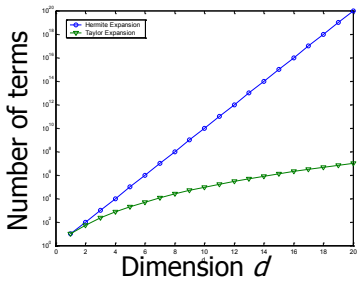
[redacted]

where coefficients C_α are given by

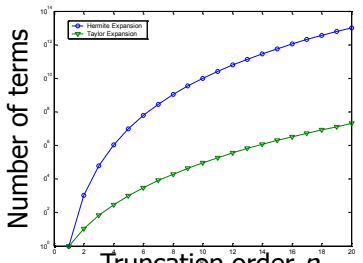
[redacted]

Reduction from Taylor Expansions

- The idea of Taylor expansion of the factored Gaussian kernel is first introduced in the course CMSC878R.
- The number of terms in multivariate Hermite expansion is $O(p^d)$.
- The number of terms in multivariate Taylor expansion is [redacted], asymptotically $O(d^p)$, a big reduction for large d.

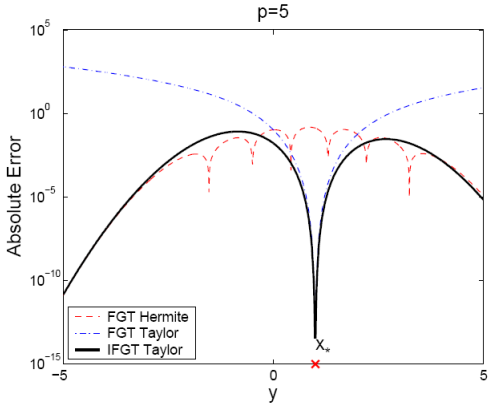


Fix p = 10, vary d = 1:20



Fix d = 10, vary p = 1:20

Global nature of the new expansion



The absolute value of the actual error between the one dimensional gaussian ($e^{-(x_i-y)/h^2}$) and different series approximations. The Gaussian was centered at $x_i = 0$. All the series were expanded about $x_* = 1.0$. $p = 5$ terms were retained in the series approximation.

Improved Fast Gauss Transform

- Multivariate Taylor expansion
- Multivariate Horner's Rule
- Space subdivision based on *k*-center algorithm
- Error bound analysis

Monomial Orders

- Let $\alpha=(\alpha_1, \cdots, \alpha_n)$, $\beta=(\beta_1, \cdots, \beta_n)$, then three standard monomial orders:
  - *Lexicographic order*, or "*dictionary*" order:
    - $\alpha \prec_{lex} \beta$ iff the leftmost nonzero entry in $\alpha - \beta$ is negative.
  - *Graded lexicographic order* (*Veronese map*, a.k.a, *polynomial embedding*):
    - $\alpha \prec_{grlex} \beta$ iff $\sum_{1 \le i \le n} \alpha_i < \sum_{1 \le i \le n} \beta_i$ or ($\sum_{1 \le i \le n} \alpha_i = \sum_{1 \le i \le n} \beta_i$ and $\alpha \prec_{lex} \beta$).
  - *Graded reverse lexicographic order*:
    - $\alpha \prec_{grevlex} \beta$ iff $\sum_{1 \le i \le n} \alpha_i < \sum_{1 \le i \le n} \beta_i$ or ($\sum_{1 \le i \le n} \alpha_i = \sum_{1 \le i \le n} \beta_i$ and the rightmost nonzero entry in $\alpha - \beta$ is positive).
- Example:
  - Let $f(x,y,z) = xy^5z^2 + x^2y^3z^3 + x^3$, then
  - w.r.t. lex f is: $f(x,y,z) = x^3 + x^2y^3z^3 + xy^5z^2$;
  - w.r.t. grlex f is: $f(x,y,z) = x^2y^3z^3 + xy^5z^2 + x^3$;
  - w.r.t. grevlex f is: $f(x,y,z) = xy^5z^2 + x^2y^3z^3 + x^3$.

The Horner's Rule

- The Horner's rule (W.G.Horner 1819) is to *recursively* evaluate the polynomial $p(x) = a_n x^n + \cdots + a_1 x + a_0$ as:

  $$p(x) = ((\cdots(a_n x + a_{n-1})x+\cdots)x + a_0.$$

  It costs n multiplications and n additions, no extra storage.
- We evaluate the multivariate polynomial *iteratively* using the graded lexicographic order. It costs n multiplications and n additions, and n storage.

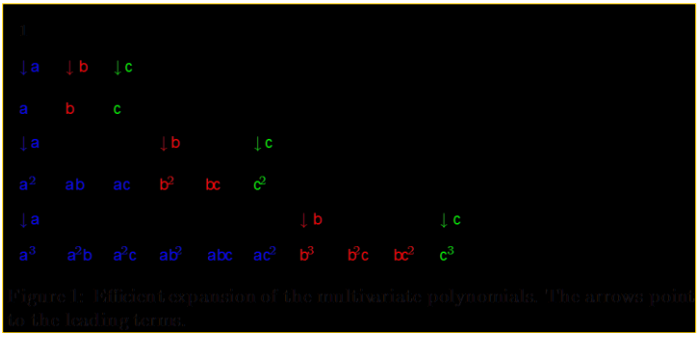
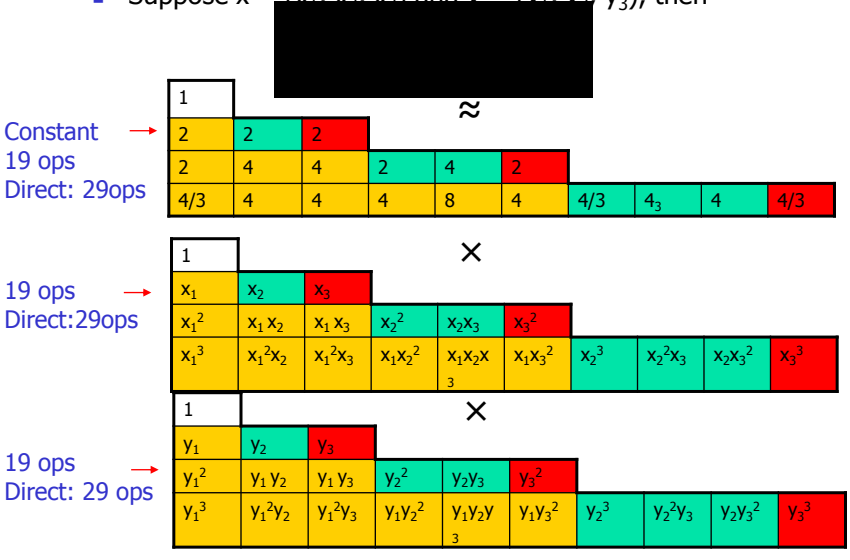


Figure 1: Efficient expansion of the multivariate polynomials. The arrows point to the leading terms.

An Example of Taylor Expansion

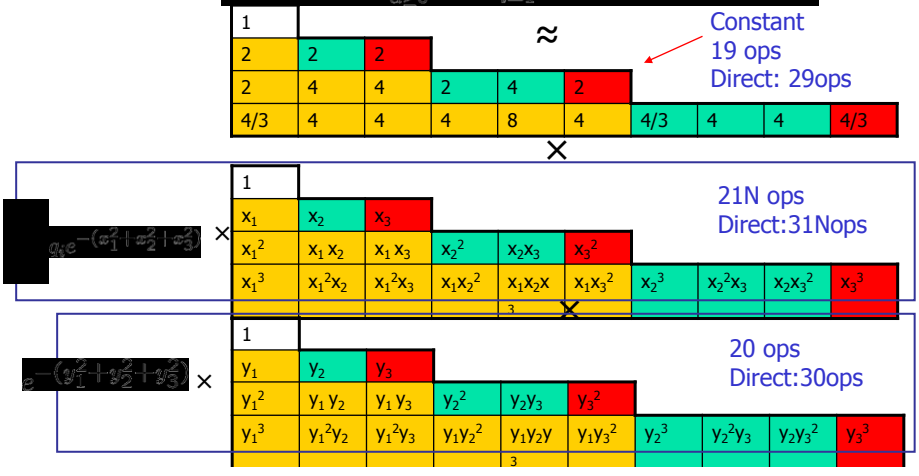
- Suppose $x = (x_1, x_2, x_3)$ and $y = (y_1, y_2, y_3)$, then



An Example of Taylor Expansion (Cont'd)

$$G(y) = \sum_{i=1}^{N} q_i e^{-\|x_i - y\|^2} = \sum_{i=1}^{N} q_i e^{-\|x_i\|^2} e^{-\|y\|^2} \sum_{\alpha \geq 0} \frac{2^{|\alpha|}}{\alpha!} x_i^\alpha y^\alpha$$

$$= e^{-\|y\|^2} \sum_{\alpha \geq 0} \frac{2^{|\alpha|}}{\alpha!} y^\alpha \sum_{i=1}^{N} q_i e^{-\|x_i\|^2} x_i^\alpha$$



Improved Fast Gauss Transform

- Multivariate Taylor expansion
- Multivariate Horner's Rule
- Space subdivision based on *k*-center algorithm
- Error bound analysis

Space Subdivision Scheme

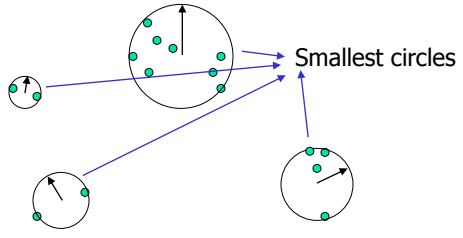
- The space subdivision scheme in the original FGT is uniform boxes. The number of boxes grows exponentially with the dimensionality.
- The desirable space subdivision should adaptively fit the density of the points.
- The cell should be as compact as possible.
- The algorithm is better to be a progressive one, that is the current space subdivision is from the previous one.
- Based on the above considerations, we model the task as *k*-center problem.

k-center Algorithm

- The *k*-center problem is defined to seek the "best" partition of a set of points into clusters (Gonzalez 1985, Hochbaum and Shmoys 1985, Feder and Greene 1988).

  Given a set of points and a predefined number *k*, *k*-center clustering is to find a partition $S = S_1 \cup S_2 \cup \cdots \cup S_k$ that minimizes $\max_{1 \le i \le k} radius(S_i)$, where $radius(S_i)$ is the radius of the smallest disk that covers all points in S_i.

- The *k*-center problem is NP-hard but there exists a simple 2-approximation algorithm.

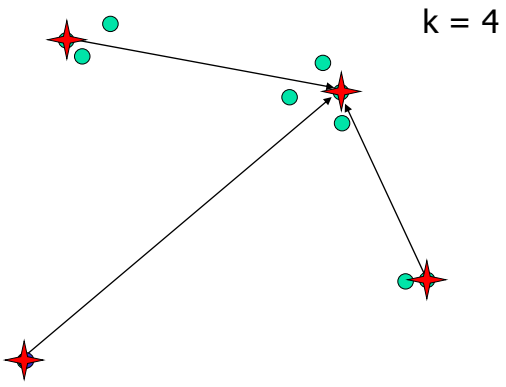


Farthest-Point Algorithm

- The farthest-point algorithm (a.k.a. *k*-center algorithm) is a 2-approximation of the optimal solution (Gonzales 1985).
- The total running time is $O(kn)$, n is the number of points. It can be reduced to $O(n \log k)$ using a slightly more complicated algorithm (Feder and Greene 1988).

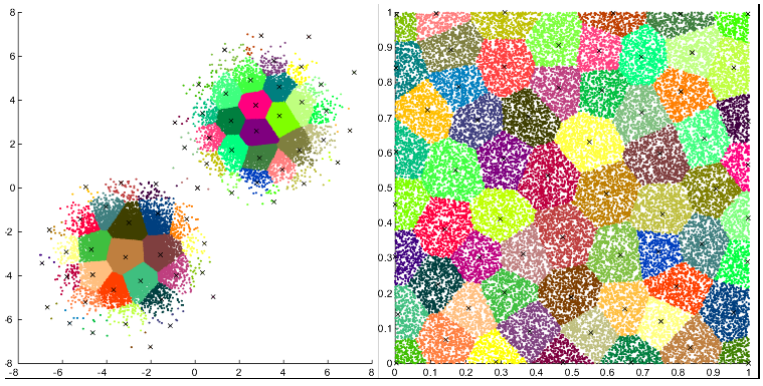
1. Initially randomly pick a point v_0 as the first center and add it to the center set C.
2. For $i = 1$ to $k - 1$ do
   - For every point $v \in V$, compute the distance from v to the current center set $C = \{v_0, v_1, \ldots, v_{i-1}\}$: $d(v, C) = \min_{c \in C} \|v - c\|$.
   - From the points $V - C$ find a point v_i that is farthest away from the current center set C, i.e. $d(v_i, C) = \max_v \min_{c \in C} \|v - c\|$.
   - Add v_i to the center set C.
3. Return the center set $C = \{v_0, v_1, \ldots, v_{k-1}\}$ as the solution to k-center problem.

A Demo of *k*-center Algorithm



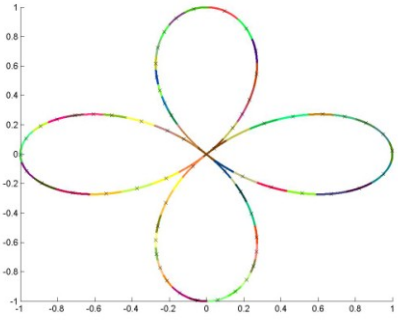
Results of k-center Algorithm

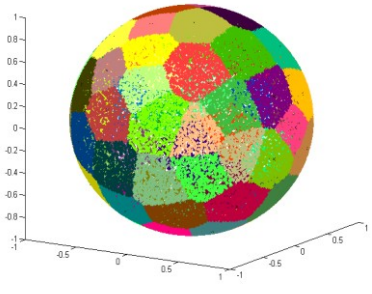
- The results of k-center algorithm. 40,000 points are divided into 64 clusters in 0.48 sec on a 900MHZ PIII PC.



More Results of k-center Algorithm

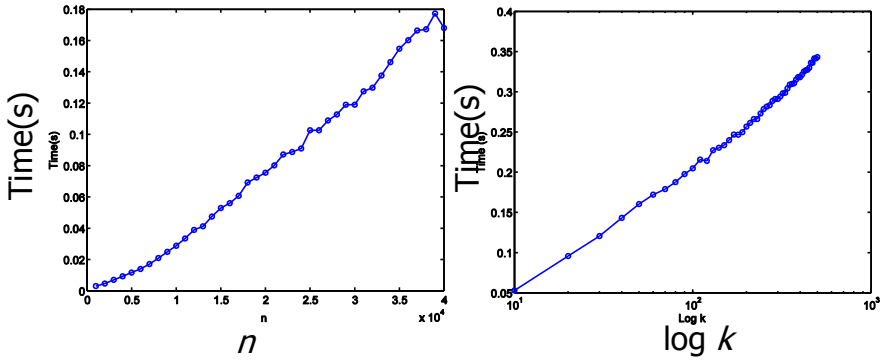
- The 40,000 points are on the manifolds.





Results of k-center Algorithm

- The computational complexity of k-center algorithm is $O(n \log k)$. Points are generated using uniform distribution. (*Left*) Number of points varies from 1000 to 40000 for 64 clusters; (*Right*) The number of clusters k varies from 10 to 500 for 40000 points.



Improved Fast Gauss Transform Algorithm

Control series truncation error

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

Step 2 Choose p sufficiently large such that the error estimate is less than the desired precision ϵ.

Step 3 For each cluster S_k with center c_k, compute the coefficients:

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

Step 4 Repeat for each target y_j, find its neighbor clusters whose centers lie within the range r_y. Then the sum of Gaussians can be evaluated by the expression:

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

Summarize the contributions from centers to targets

Complexities

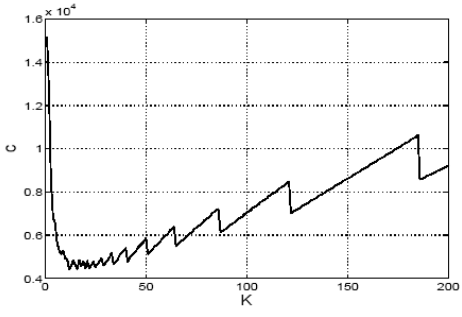
- FGT $O(p^d N) + O(p^d M) + O(dp^{d+1}(2n+1)^d min((\sqrt{2}rh)^{-d/2}, M)).$
  - Expansion and evaluation
  - Translation
- IFGT
  - Expansion and evaluation; data structures

The farthest point clustering has running time $\mathcal{O}(N \log K)$ (Feder and Greene, 1988). Computing the cluster coefficients C_α^k for all the clusters is of $\mathcal{O}(N r_{(p_{max}-1)d})$, where $r_{(p_{max}-1)d} = \binom{p_{max}+d-1}{d}$ is the total number of d-variate monomials of degree less than or equal to $p_{max} - 1$. Computing $\hat{G}(y_j)$ is of $\mathcal{O}(Mnr_{(p_{max}-1)d})$ where n if the maximum number of neighbor clusters (depends on the bandwidth h and the error ϵ) which influence the target. Hence the total computational complexity is

$$\mathcal{O}(N \log K + N r_{(p_{max}-1)d} + Mnr_{(p_{max}-1)d}).$$

Assuming $M = N$, the complexity is $\mathcal{O}(\left[\log K + (1+n)r_{(p_{max}-1)d}\right] N)$. The constant term depends on the dimensionality, the bandwidth, and the accuracy required. The number of terms $r_{(p_{max}-1)d}$ is asymptotically polynomial in d. For $d \to \infty$ and moderate p, the number of terms is approximately d^p.

Optimal number of centers



ınt $c = \log K + (1+n)r_{(p_{max}-1)d}$ as function K. $d = 2$, $h = 0.3$, and

Improved Fast Gauss Transform

- Multivariate Taylor expansion
- Multivariate Horner's Rule
- Space subdivision based on *k*-center algorithm
- Error bound analysis

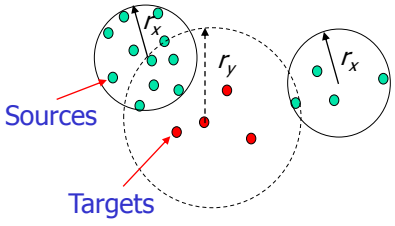
Error Bound of IFGT

- The total error from the series truncation and the cutoff outside of the neighborhood of targets is bounded by

$$|E(y)| \leq \sum |q_i| \left(\frac{2^p}{p!} \left(\frac{r_x}{h}\right)^p \left(\frac{r_y}{h}\right)^p + e^{-(r_y/h)^2} \right).$$

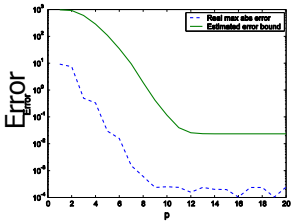
Truncation error

Cutoff error

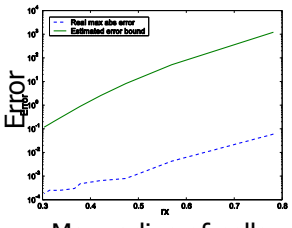


Error Bound Analysis

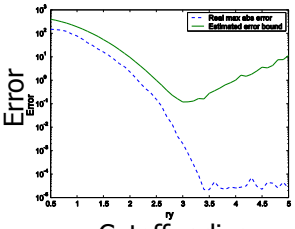
- Increase the number of truncation terms p, the error bound will decrease.
- With the progress of k-center algorithm, the radius of the source points r_x will decrease, until the error bound is less than a given precision.
- The error bound first decreases, then increases with respect to the cutoff radius r_y.



Truncation order p



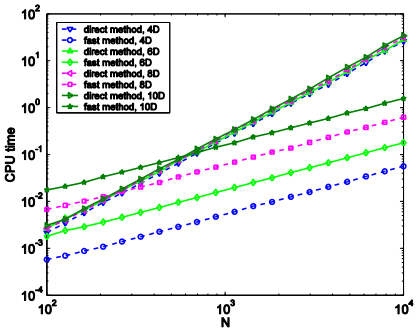
Max radius of cells

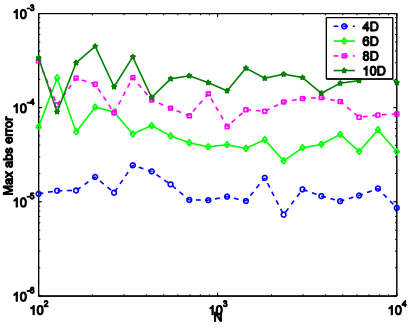


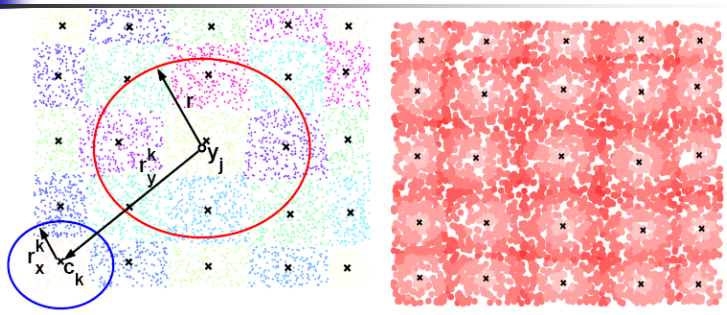
Cutoff radius

Experimental Result

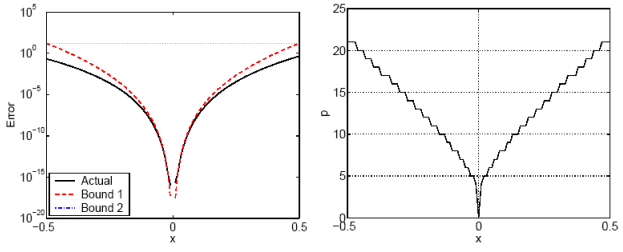
- The speedup of the fast Gauss transform in 4, 6, 8, 10 dimensions (h=1.0).







- Parameter selection is completely automatic.
- Tighter point-wise error bounds.
- Truncation number different for each point.



(a) (b)

Figure 1: (a) The actual residual (solid line) and the bound (dashed line) given by Equation 4 as a function of x. [$x_* = 0$, $y = 1.0$, $h = 0.5$, $r_x = 0.5$, $r_y = 1.0$, and $p = 10$]. The residual is minimum at $x = x_*$ and increases as x moves away from x_*. The dotted line shows the very pessimistic bound which is independent of x (Equation 15) used in the original IFGT. (b) The truncation number p required as the function of x so that the error is less than 10^{-6}.

Pointwise error bounds and translation

- Can be used to speed-up general FMM also

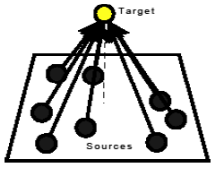
Some Applications

- Kernel density estimation
- Mean-shift based segmentation
- Object tracking
- Robot navigation

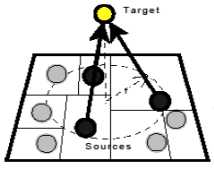
Auto Tuning (Morariu et al., 2008)

Tree data structure

- Using a tree data structure, we can compute Gauss Transform efficiently across bandwidths
- Must choose one of four evaluation methods

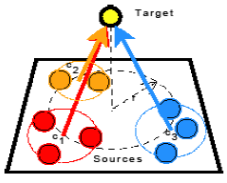


Direct: evaluate equation directly; works well for small N and M, and does not have high overhead cost

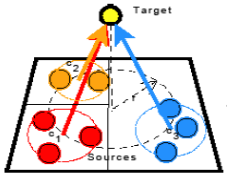


Direct+Tree: build tree directly on sources; works well for small bandwidths

Four algorithms and complexities



IFGT: works well for large N and M, and medium to large bandwidths



IFGT+Tree: build tree on cluster centers, works well for case where number of clusters K is large

Comparison of methods

$Cost_{\text{direct}}(d, N, M)$	$O(dMN)$
$Cost_{\text{direct+tree}}(d, N, M, n_s)$	$O(d(N + Mn_s)\log N)$
$Cost_{\text{ifgt}}(d, N, M, K, n_c, p_{\max})$	$O(dN\log K + (N + Mn_c)r_{(p_{\max}-1)d} + dMK)$
$Cost_{\text{ifgt+tree}}(d, N, M, K, n_c, p_{\max})$	$O((N + Mn_c)(d\log K + r_{(p_{\max}-1)d}))$

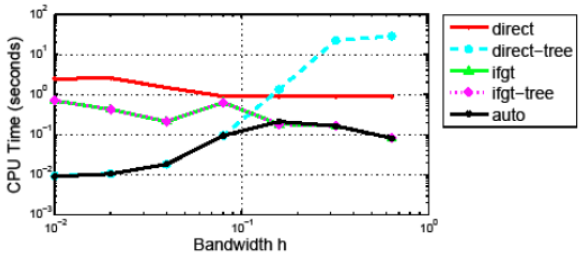
Automatic method selection

- n_s and n_c are the average number of sources and cluster centers, respectively, that are within the cut-off radius of a query target
- n_s and n_c can be approximated from sub-sampled dataset
- Given *d, N, M, K,* n_c, and n_s, we can estimate cost of each method

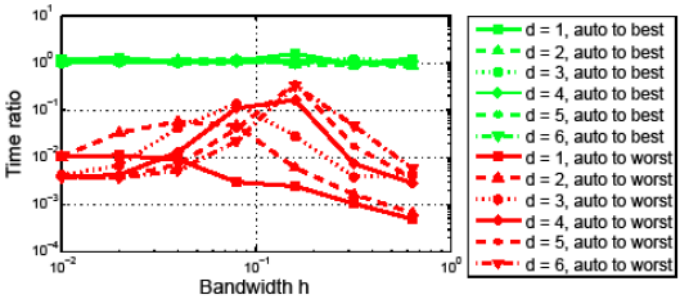
Summary of method selection approach:

1) Estimate n_s
2) Calculate $\text{Cost}_{\text{direct}}(d,N,M)$ and $\text{Cost}_{\text{direct+tree}}(d,N,M,n_s)$
3) Estimate highest K_{limit} for which *ifgt* and *ifgt+tree* could be faster
4) If $K_{limit} > 0$
   a) Compute IFGT parameters K and p_{max}
   b) Estimate n_c ; estimate $\text{Cost}_{\text{ifgt}}$ and $\text{Cost}_{\text{ifgt+tree}}$
5) Return arg $\min_i \text{Cost}_i$

Results

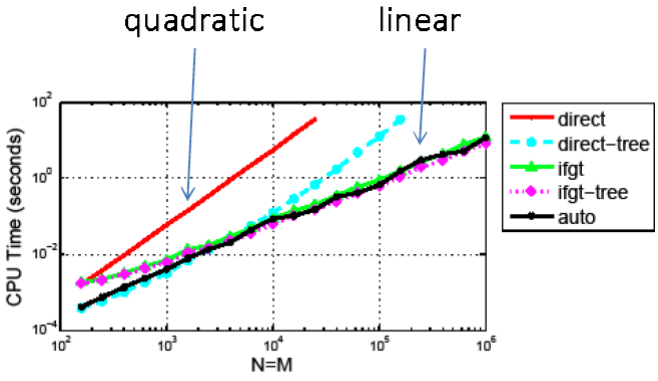


d=4, M=N=4000, ε=10^-4



M=N=4000, ε=10^-4

Results



d=3. h=.06. ε=10^-2

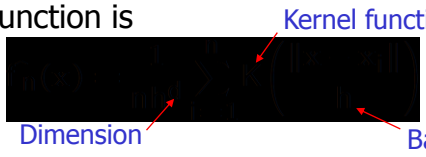
- Open source C/C++ and MATLAB bindings
- http://sourceforge.net/projects/figtree

Kernel Density Estimation (KDE)

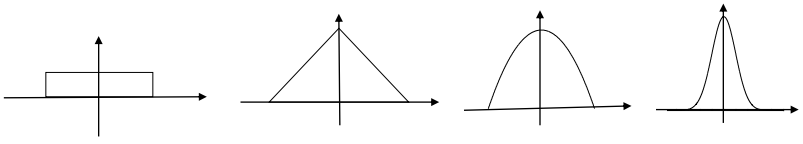
- Kernel density estimation (a.k.a Parzen method, Rosenblatt 1956, Parzen 1962) is an important nonparametric technique.
- KDE is the keystone of many algorithms:
  - Radial basis function networks
  - Support vector machines
  - Mean shift algorithm
  - Regularized particle filter
- The main drawback is the quadratic computational complexity. Very slow for large dataset.

Kernel Density Estimation

- Given a set of observations $\{x_1, \cdots, x_n\}$, an estimate of density function is



- Some commonly used kernel functions



- The computational requirement for large datasets is $O(N^2)$, for N points.

Efficient KDE and FGT

- In practice, the most widely used kernel is the Gaussian

- The density estimate using the Gaussian kernel:

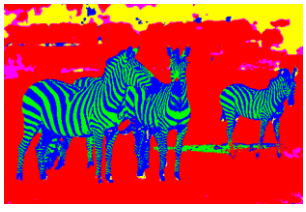
- Fast Gauss transform can reduce the cost to $O(N \log N)$ in low-dimensional spaces.
- Improved fast Gauss transform accelerates the KDE in both lower and higher dimensions.

Experimental Result

- Image segmentation results of the mean-shift algorithm with the Gaussian kernel.

Size: 432X294
Time: 7.984 s

Direct evaluation:
more than 2 hours



Size: 481X321
Time: 12.359 s

Direct evaluation:
more than 2 hours

Some Applications

- Kernel density estimation
- Mean-shift based segmentation
- Object tracking
- Robot Navigation

Object Tracking

- Goal of object tracking: find the moving objects between consecutive frames.
- A model image or template is given for tracking.
- Usually a feature space is used, such as pixel intensity, colors, edges, etc.
- Usually a similarity measure is used to measure the difference between the model image and current image.
- Temporal correlation assumption: the change between two consecutive frames is small.

Model image

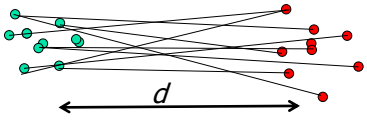
Target image

Proposed Similarity Measures

- Our similarity measure

[redacted]

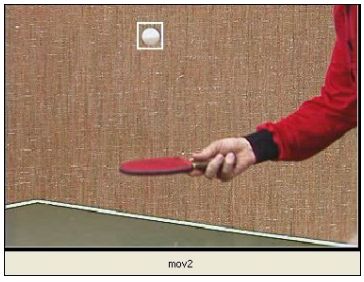
- Directly computed from the data points.
- Well scaled into higher dimensions.
- Ready to be speeded up by fast Gauss transform, if Gaussian kernels are employed.
- Interpreted as the expectation of the density estimations over the model and target images.



Experimental results

- Pure translation
  - Feature space: *RGB* + 2D spatial space
  - Average processing time per frame: 0.0169*s*.

Our method



Histogram tracking using Bhattacharyya distance

Experimental results

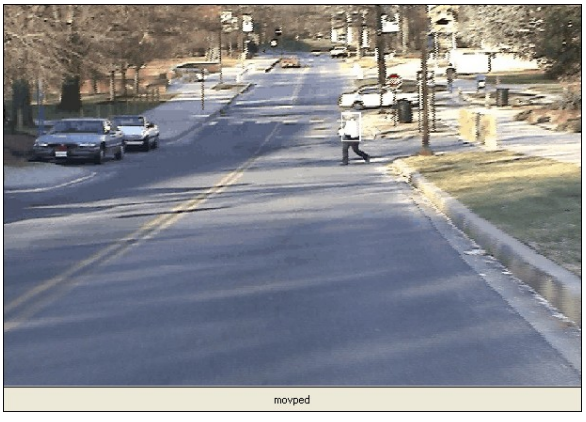
- Pure translation
  - Feature space: *RGB* + 2D gradient + 2D spatial space
  - Average processing time per frame: 0.0044*s*.

Experimental results

- Pure translation
  - Feature space: *RGB* + 2D spatial space
  - The similarity measure is robust and accurate to the small target, compared with the other methods, such as histogram.

Experimental Results

- Translation + Scaling
  - Feature space: RGB + 2D spatial space



Conclusions

- Improved fast Gauss transform is presented for higher dimensional spaces.
- Kernel density estimation is accelerated by the IFGT.
- A similarity measure in joint feature-spatial space is proposed for robust object tracking.
- With different motion models, we derived several tracking algorithms.
- The improved fast Gauss transform is applied to speed up the tracking algorithms to real-time performance.