An Improved Fast Gauss Transform with Applications

Ramani Duraiswami
Perceptual Interfaces and Reality Laboratory
Institute for Advanced Computer Studies
University of Maryland, College Park
http://www.umiacs.umd.edu/~ramani

Joint work with Changjiang Yang, Nail A. Gumerov and Larry S. Davis

Fast Multipole Methods

- Originally proposed by Rokhlin and Greengard (1987) to efficiently evaluate sums of monopoles:
- FMM accelerates matrix vector products (sums) of the type
  - X source point set
  - Y evaluation point set
  - Φ some function
- Original functions Φ for which FMM was developed were long-ranged and singular at the source point
- FMM relies on "separation of variables" to achieve speed

$$\mathbf{v} = \mathbf{\Phi u},$$

$$\mathbf{\Phi} = \begin{pmatrix} \Phi(\mathbf{y}_1,\mathbf{x}_1) & \Phi(\mathbf{y}_1,\mathbf{x}_2) & \dots & \Phi(\mathbf{y}_1,\mathbf{x}_N) \\ \Phi(\mathbf{y}_2,\mathbf{x}_1) & \Phi(\mathbf{y}_2,\mathbf{x}_2) & \dots & \Phi(\mathbf{y}_2,\mathbf{x}_N) \\ \dots & \dots & \dots & \dots \\ \Phi(\mathbf{y}_M,\mathbf{x}_1) & \Phi(\mathbf{y}_M,\mathbf{x}_2) & \dots & \Phi(\mathbf{y}_M,\mathbf{x}_N) \end{pmatrix}.$$

$$X = \{\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_N\}, \quad \mathbf{x}_i \in \mathsf{R}^d, \quad i = 1,\dots,N,$$
$$Y = \{\mathbf{y}_1,\mathbf{y}_2,\dots,\mathbf{y}_M\}, \quad \mathbf{y}_j \in \mathsf{R}^d, \quad j = 1,\dots,M.$$

$$v_j = \sum_{i=1}^{N} u_i \Phi(\mathbf{y}_j,\mathbf{x}_i), \quad j = 1,\dots,M.$$

Factorization

$$\forall \mathbf{x}_i, \mathbf{y}_j \in \Omega \subset \mathbb{R}^d :$$

$$\Phi(\mathbf{y}_j, \mathbf{x}_i) = \sum_{m=0}^{\infty} a_m(\mathbf{x}_i - \mathbf{x}_*) f_m(\mathbf{y}_j - \mathbf{x}_*) = \sum_{m=0}^{p-1} a_m(\mathbf{x}_i - \mathbf{x}_*) f_m(\mathbf{y}_j - \mathbf{x}_*) + Error(p, \mathbf{x}_i, \mathbf{y}_j)$$

Expansion center

Truncation number

Expansion coefficients

Basis functions

- Substitute in the product
- Rearrange summation order
- Inner sum does not depend on evaluation points

$$\begin{aligned} v_j &= \sum_{i=1}^{N} \Phi(\mathbf{y}_j, \mathbf{x}_i) u_i \\ &= \sum_{i=1}^{N} \left[\sum_{m=0}^{p-1} a_m(\mathbf{x}_i - \mathbf{x}_*) f_m(\mathbf{y}_j - \mathbf{x}_*) + Error(p, \mathbf{x}_i, \mathbf{y}_j) \right] u_i \\ &= \sum_{m=0}^{p-1} f_m(\mathbf{y}_j - \mathbf{x}_*) \sum_{i=1}^{N} a_m(\mathbf{x}_i - \mathbf{x}_*) u_i + \sum_{i=1}^{N} Error(p, \mathbf{x}_i, \mathbf{y}_j) u_i \\ &= \sum_{m=0}^{p-1} c_m f_m(\mathbf{y}_j - \mathbf{x}_*) + Error(N, p), \end{aligned}$$

Reduction of Complexity

Straightforward (nested loops):

```
for j = 1,...,M
    v_j = 0;
    for i = 1,...,N
        v_j = v_j + Φ(y_j, x_i) u_i;
    end;
end;
```

Complexity: *O*(*MN*)

If $p \ll \min(M,N)$ then complexity reduces!

Factorized:

```
for m = 0,...,p − 1
    c_m = 0;
    for i = 1,...,N
        c_m = c_m + a_m(x_i − x_*) u_i;
    end;
end;

for j = 1,...,M
    v_j = 0;
    for m = 0,...,p − 1
        v_j = v_j + c_m f_m(y_j − x_*);
    end;
end;
```

Complexity: *O*(*pN*+*pM*)

- Remark: *O(N)* for fixed *p*.
- However, error grows with *N*
- For fixed error have to increase *p* with *N*
- For geometrically convergent series this introduces a factor of log *N*

Conventional FMM

- Function Φ is singular and a uniformly valid factorization is not available
- Construct patchwork-quilt of overlapping approximations
  - ❑ Local and Multipole Expansions
- Partition sum into a piece that is computed directly and piece that uses factorization.
- Tree data-structures used to reduce the cost of the piece that must be computed directly to that computed via factorization
  - ❑ Translation operators convert one representation to another
- Achieve *O(N)* complexity for fixed *p*
- Remarks
  - ❑ Building data structures is *O(N* log *N)*
  - ❑ For fixed error *p* could depend on *N* and make complexity *O(N* log *N)*

Fast Gauss Transform

- FMM was applied to evaluate sums of Gaussians by Greengard & Strain (1989, 1991)

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

Targets Sources

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

- Direct evaluation requires $O(N^2)$ operations.
- FGT reduces cost to $O(N \log N)$ operations.

Original FGT Factorization :Hermite Expansion

- Gaussian kernel factorized into Hermite and Taylor expansions

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

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

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

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

- Exchange order of summations

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

$$= \sum_{n=1}^{p-1} A_n h_n\left(\frac{y_j - x_*}{h}\right) + \epsilon(p)$$

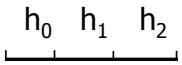
where A_n is defined by $\quad A_n = \frac{1}{n!}\sum_{i=1}^{N} q_i \left(\frac{x_i - x_*}{h}\right)^n$

FGT obtained by applying FMM framework

- Local and “far-field” expansion
- Translation of Hermite expansion to Taylor expansion
- Box data-structures
- Our goal to use the FGT for problems in computer vision and pattern recognition
- Problems not restricted to 1-3 dimensions
  - High dimensional “feature” spaces
- Need to use FGT in high dimensions
- FGT does not scale well with dimensionality

Hermite Expansion 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

h_0h_0	h_0h_1	h_0h_2
h_1h_0	h_1h_1	h_1h_2
h_2h_0	h_2h_1	h_2h_2

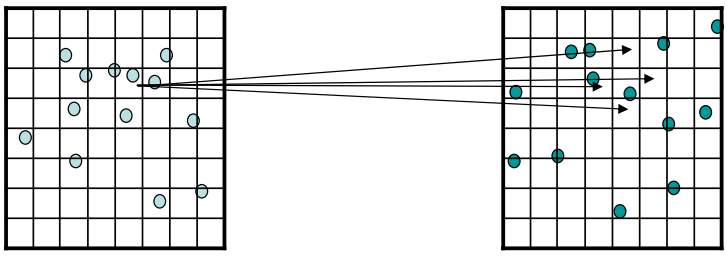
D=2

D=3

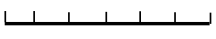
D>3

Space Subdivision in FGT

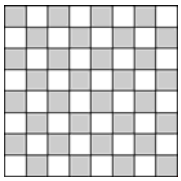
- The FGT subdivides the space into uniform boxes and assigns the source points and target points into boxes.
- For each box the FGT maintain a neighbor list.



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



D=1



D=2

D=3

D>3

FGT in Higher Dimensions

- 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 FGT was originally designed to solve the problems in mathematical physics (heat equation, vortex methods, etc), where the dimension is up to 3.
- The exponential dependence on the dimension makes the FGT extremely inefficient in higher dimensions.

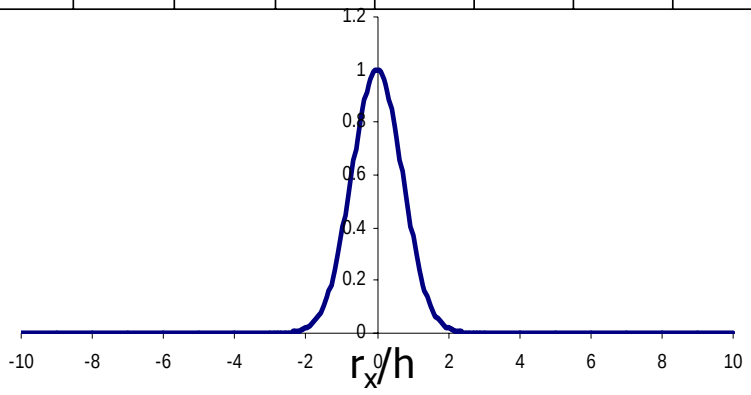
Improved Fast Gauss Transform

- Reconsider data structures and expansions needed
- Comparing Gaussians with conventional FMM Φ
  - Gaussian is not singular – it is infinitely differentiable!
  - Gaussians vanish exponetially quickly in the far-field
- Modified expansions
  - Local: Multivariate Taylor Expansions
  - Far field expansion is zero!
- Modified data structures
  - Data structures are not needed to separate domains of validity (expansions are valid throughout)
  - Rather need data structures to decide where to ignore the effect of the Gaussian and to decide center of Gaussian

Far Field Expansion is Zero

- The decay of the Gaussian kernel function is rapid.
  - ❑Effect of Gaussian outside certain range can be safely ignored
- Time consuming translation operators in original FGT can be safely removed!

r_x/h	0	1	2	3	4	5	6	7	8	9	10
$f(r_x/h)$	1	0.367879	0.018316	0.000123	1.13E-07	1.39E-11	2.32E-16	5.24E-22	1.6E-28	6.64E-36	3.72E-44



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 depend on x_i or y_j alone.
- 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_*:

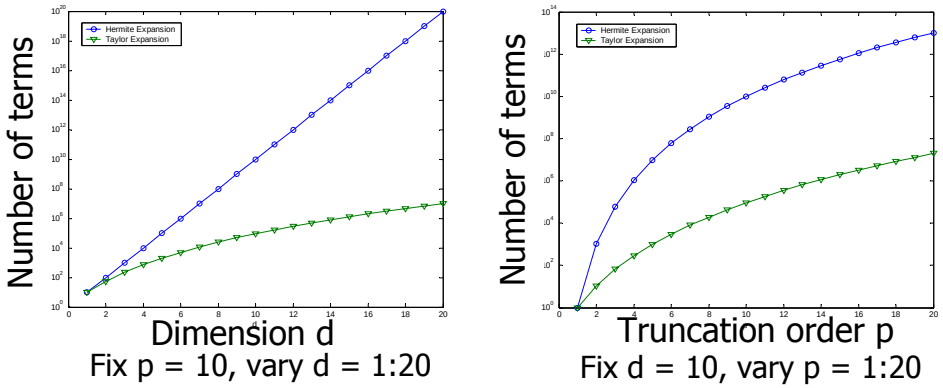
$$G(y_j) = \sum_{\alpha\geq 0} C_\alpha e^{-\|y_j-x_*\|^2/h^2} \left(\frac{y_j-x_*}{h}\right)^{\alpha},$$

- where coefficients C_α are given by

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

Modified Factorization: Taylor Expansions

- The number of terms in multivariate Taylor expansion is $\binom{p+d-1}{d}$, asymptotically $O(d^p)$
- Original expansion has $O(p^d)$ terms
- New expansion results in a big reduction for large d and moderate p



Space Subdivision Scheme

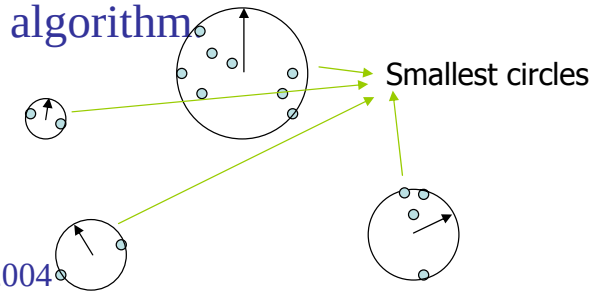
- The space subdivision scheme in the original FGT is uniform boxes. The number of boxes grows exponentially with the dimensionality.
- Need a data structure that
  - ❑ Allows ignoring the far-field
  - ❑ Assigns each point to a local expansion center
- The space subdivision should adaptively fit density of the points.
- The cell should be as compact as possible.
- The algorithm should be a progressive one,
  - ❑ Refined space subdivision obtained from previous one.
- Based on the above considerations, we develop a structure using the 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.



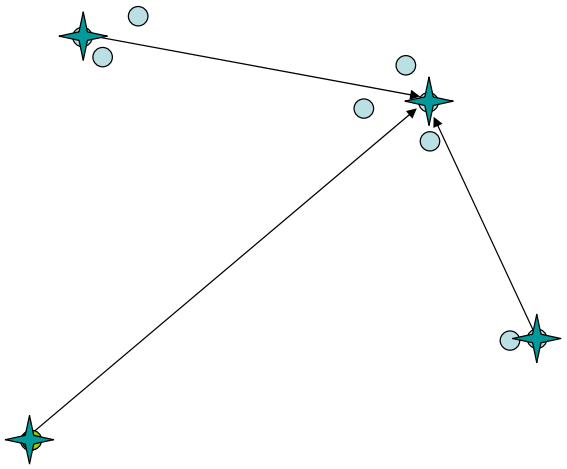
Farthest-Point Algorithm

- The farthest-point algorithm (a.k.a. *k*-center algorithm) is a 2-approximation to 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 $C = \{v_0, v_1, \ldots, v_{i-1}\}$: $d_i(v, C) = \min_{c \in C} \|v - c\|$.
   - From the points $V - C$ find a point v_i that is farthest away from the current center set C, i.e. $d_i(v_i, C) = \max_v \min_{c \in C} \|v - c\|$.
   - 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.

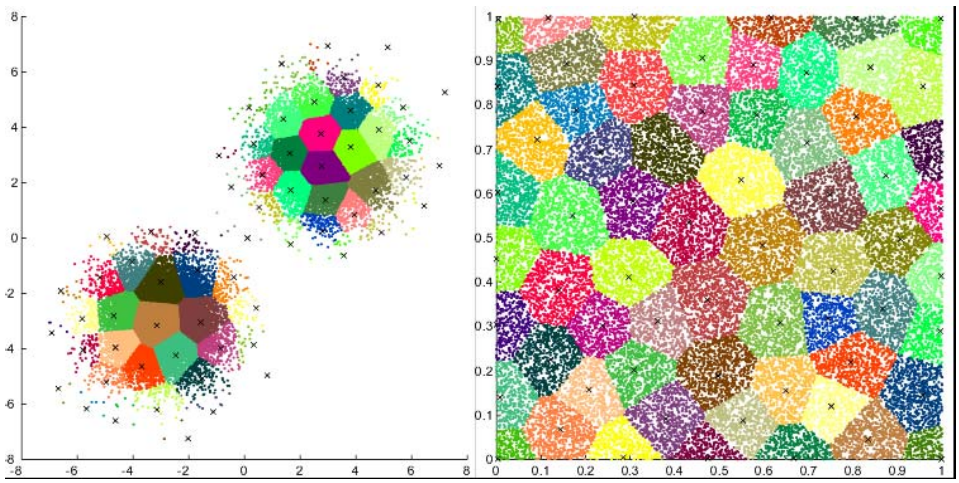
A Demo of *k*-center Algorithm

k = 4



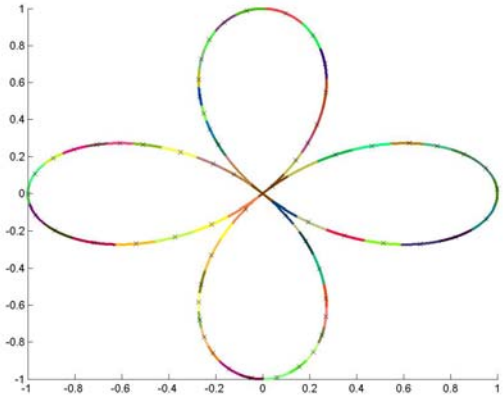
Results of k-center Algorithm

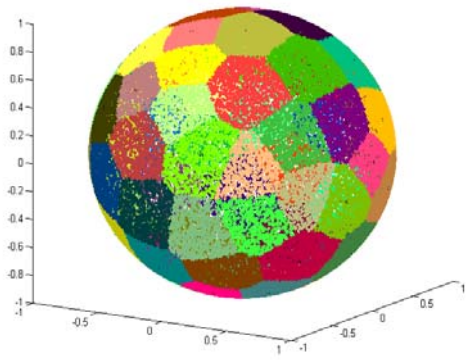
- The results of k-center algorithm. 40,000 points are divided into 64 clusters in 0.48 sec on a 900MHZ PIII PC.



More Results of k-center Algorithm

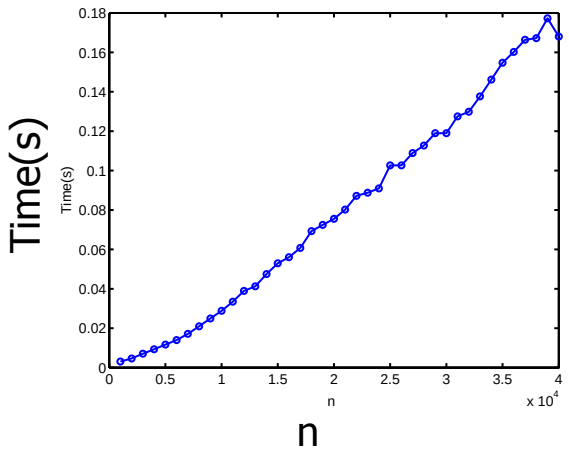
- The 40,000 points are on the manifolds.

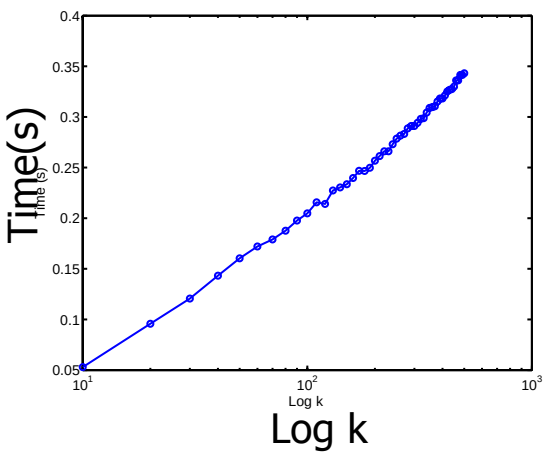




Properties of *k*-center Algorithm

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





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:
    - $\alpha \prec_{grlex} \beta$ 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: $f(x,y,z) = x^3 + x^2y^3z^3 + xy^5z^2$;
  - w.r.t. grlex: $f(x,y,z) = x^2y^3z^3 + xy^5z^2 + x^3$;
  - w.r.t. grevlex: $f(x,y,z) = xy^5z^2 + x^2y^3z^3 + x^3$.

Horner's Rule

- Horner's rule (Horner, 1819) *recursively* evaluates the polynomial $a_p x^p + \cdots + a_1 x + a_0$ as:

$$((\cdots(a_p x + a_{p-1})x + \cdots)x + a_0.$$

- costs p multiplications and p additions, no extra storage.
  Reduces complexity from $O(p^2)$ to $O(p)$
- We do this for the multivariate polynomial *iteratively* using the graded lexicographic order. Costs $C(p+d-1,d)$ operations and storage.

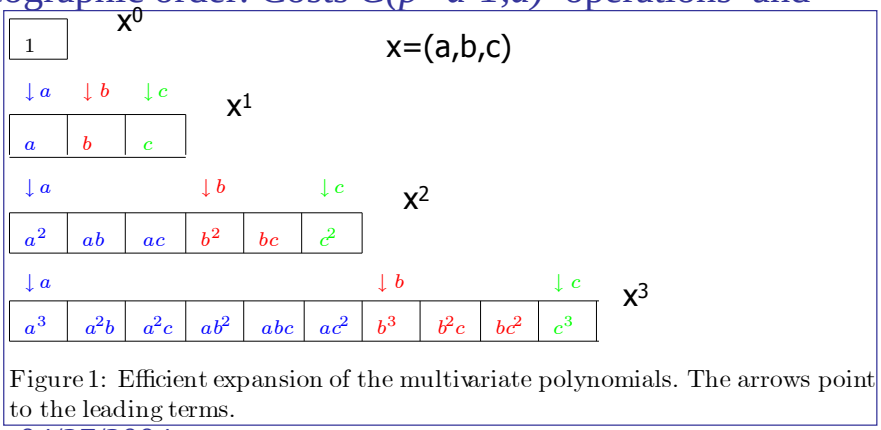
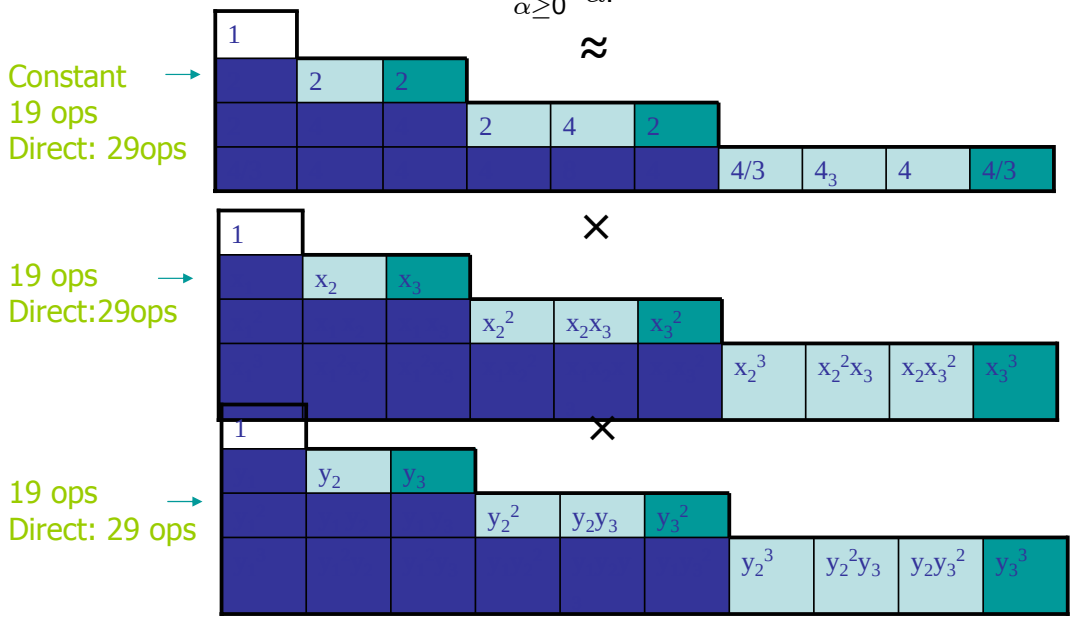


Figure 1: Efficient expansion of the multivariate polynomials. The arrows point to the leading terms.

An Example of Taylor Expansion

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

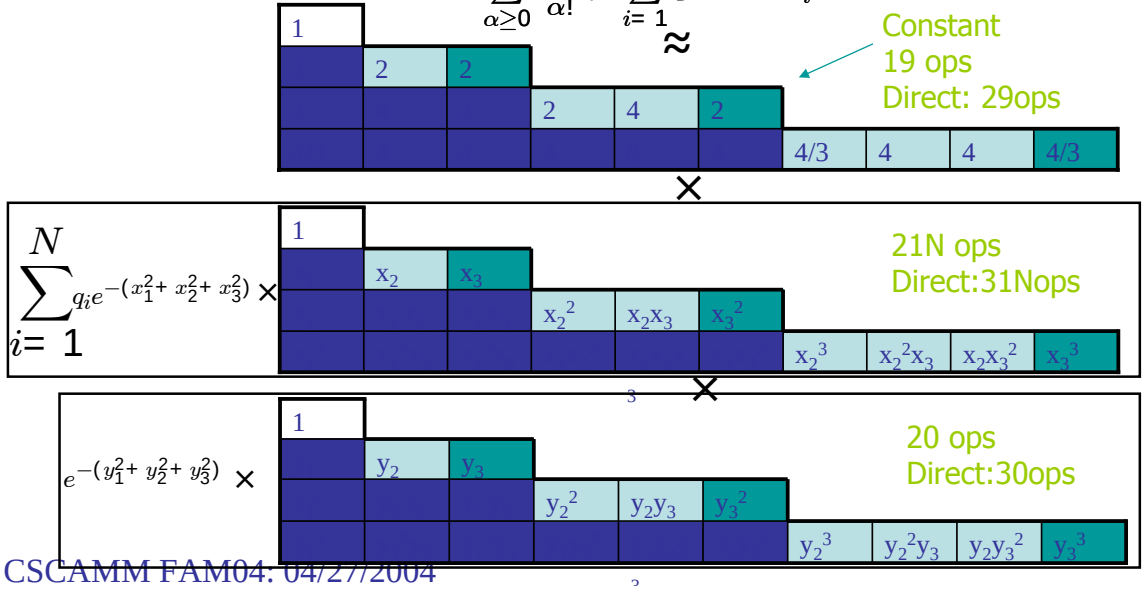
$$e^{2x\cdot y} = \sum_{\alpha\geq 0} \frac{2^{|\alpha|}}{\alpha!} x^\alpha y^\alpha \approx$$



An Example of Taylor Expansion (Cont'd)

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

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



Improved Fast Gauss Transform

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:

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

Collect the contributions from sources to centers

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:

Control far field cutoff error

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

Summarize the contributions from centers to targets

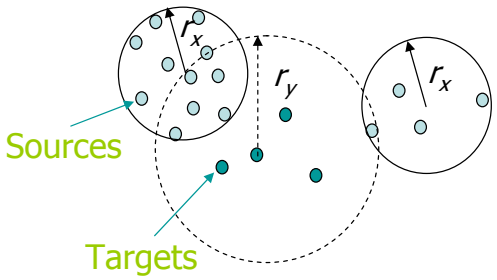
Error Bound of IFGT

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

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

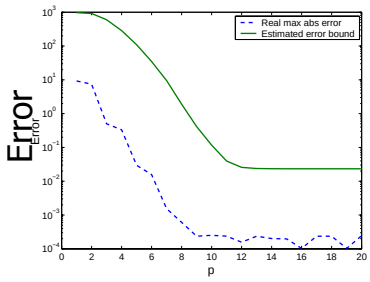
Truncation error

Cutoff error

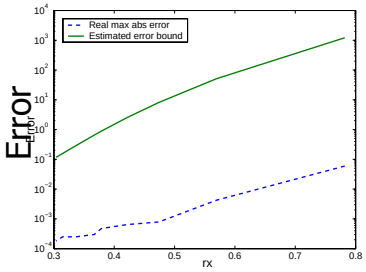


Error Bound Analysis

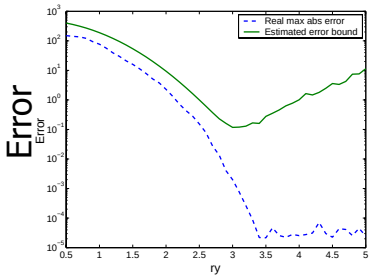
- Increasing number of truncation terms p, reduces error
- Increasing k in the k-center algorithm, radius of source point clusters r_x will decrease, until the error bound is less than a given precision.
- The error bound first decreases, then increases with respect to the cutoff radius r_y.



Truncation order p



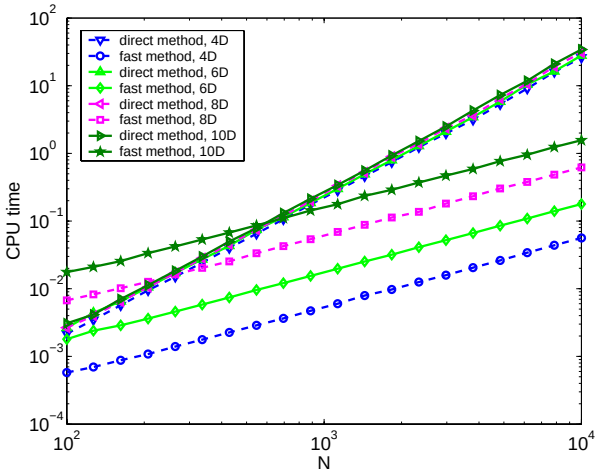
Max radius of cells



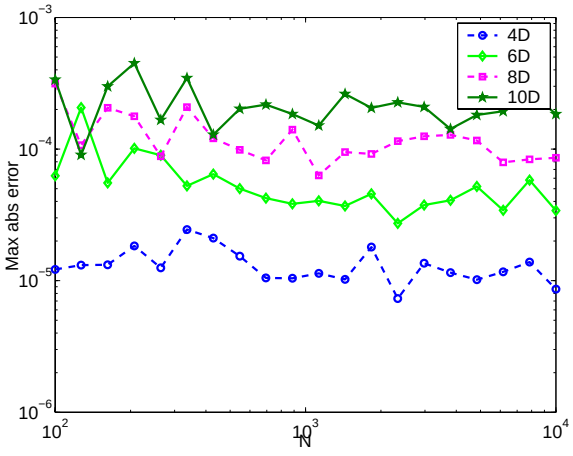
Cutoff radius

Experimental Result

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



N



N

Efficient Kernel Density Estimation

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.

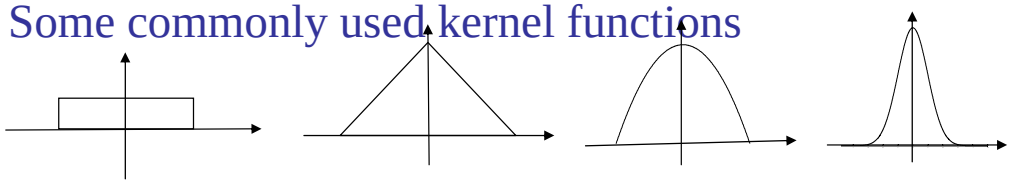
Kernel Density Estimation

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

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

Kernel function; Dimension; Bandwidth

- 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

$$K_N(\mathbf{x}) = (2\pi)^{-d/2} e^{-\frac{1}{2}\|\mathbf{x}\|^2}.$$

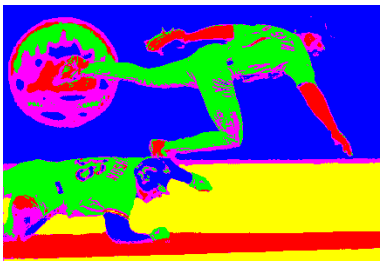
- The density estimate using the Gaussian kernel:

$$\hat{p}_n(\mathbf{x}) = c_N \sum_{i=1}^{N} e^{-\|\mathbf{x}-\mathbf{x}_i\|^2/h^2}$$

- 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

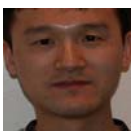
Direct evaluation:
more than 2 hours

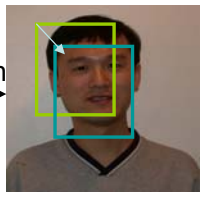

Size: 481X321
Time: 12.359 s

Direct evaluation:
more than 2 hours

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.



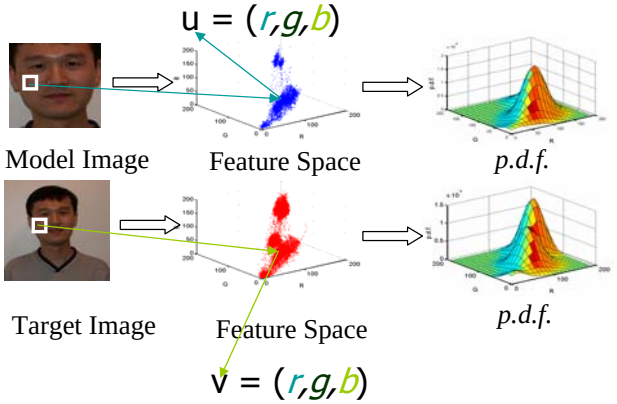


Model image

Target image

Image Representations

- Images are mapped into feature spaces.
- Feature spaces are described by the probabilistic density functions (*p.d.f.*).
- The *p.d.f.* is estimated using kernel density estimation:



$$\hat{p}_x(\mathbf{u}) = \frac{1}{N}\sum_{i=1}^{N} k(\|\frac{\mathbf{u}-\mathbf{u}_i}{h}\|^2)$$

$$\hat{p}_y(\mathbf{v}) = \frac{1}{M}\sum_{j=1}^{M} k(\|\frac{\mathbf{v}-\mathbf{v}_j}{h}\|^2)$$

- Accelerated using FGT. Details in Yang et al 2004.

Experimental results

Experimental results

Future Work

- Applications to classification via dimension reduction and FGT accelerated SVM
- Bandwidth selection
- FGT code FIGTREE (v 1.0) to be released shortly
  - Free for noncommercial use