Mixture Trees for Modeling and Fast Conditional Sampling with Applications in Vision and Graphics

Frank Dellaert, Vivek Kwatra, and Sang Min Oh
College of Computing, Georgia Institute of Technology, Atlanta, GA
{*dellaert,kwatra,sangmin*}@*cc.gatech.edu*

Abstract

We introduce mixture trees, a tree-based data-structure for modeling joint probability densities using a greedy hierarchical density estimation scheme. We show that the mixture tree models data efficiently at multiple resolutions, and present fast conditional sampling as one of many possible applications. In particular, the development of this data-structure was spurred by a multi-target tracking application, where memory-based motion modeling calls for fast conditional sampling from large empirical densities. However, it is also suited to applications such as texture synthesis, where conditional densities play a central role. Results will be presented for both these applications.

1 Introduction

We introduce *mixture trees*, a novel multi-resolution approach to density estimation and modeling from large data sets. They are similar to bumptrees [9], multi-resolution kd-trees [8] and tree-based estimators [12] in that they can be pre-built for a large set of training instances, and can efficiently implement a number of machine-learning related operations. However, in addition, mixture trees are (a) built such that one can make an conditional cut through the tree and obtain a good conditional density estimate of the dataset at arbitrary resolution, and (b) as a result, support fast real-time conditional sampling from the underlying model.

As a result, mixture trees are well suited for applications where one needs to sample from conditional densities. This capability is central to many image processing applications, such as image restoration, compression, and texture classification [10]. In particular, one area where one needs *fast* conditional sampling is texture synthesis, in which many successful methods have taken a conditional Markov random field approach [2, 17]. Here pixel values in a synthesized image are inferred from neighboring pixels, based on a conditional density estimated from a training texture patch. Another application is modeling behavior, e.g. in [5] we need to model the motion of tracked honeybees in response to the relative location of nearby bees. Using mixture trees, these models could be learned from large data sets and then used in a particle filter, where sampling from a conditional motion model is a key computation.

Among the best-known hierarchical data structures commonly used in machine learning are *kd-trees*, originally developed to speed up nearest neighbor and range search by recursively partitioning the search space [1]. It was shown in [8] that kd-trees adorned with multi-resolution cached statistics can enable very fast locally weighted regression and other instance based learning algorithms. These *mrkd-trees* can also be used for clustering and EM-based mixture modeling [7]. Omohundro [9] showed that kd-trees and other common hierarchical data structures, such as oct-trees and ball-trees, could all be viewed as instances of bumptrees. A *bumptree* is a general tree-based structure where each node is associated with a function on the space, with the constraint that each interior node's function must be larger than the functions in the nodes below it. Other data structures that are suited for either high-dimensional data or spaces with non-Euclidean metrics are vantage-point trees [18] and the nearly identical metric trees [15].

Unlike the data structures above, the *mixture trees* introduced in this paper are built with the additional goal of approximating the underlying density of the data at *every* level of the tree, including at internal nodes. Each node in the tree stores the parameters of a parametric density model, hereafter called a *component*. **The basic idea is illustrated in Figure 1**, where 1000 samples drawn from a three-class mixture distribution are fit by a mixture tree with naive Bayes components. Two invariants are maintained throughout the entire tree:

Invariant 1: Each node's component optimally approximates the density of the data stored in its own subtree.

Invariant 2: Every internal node's children represent an optimal mixture density estimate of all the data points stored in the (parent) node subtree.

These invariants are established while recursively building the tree, as is explained below in Section 2. In Figure 1

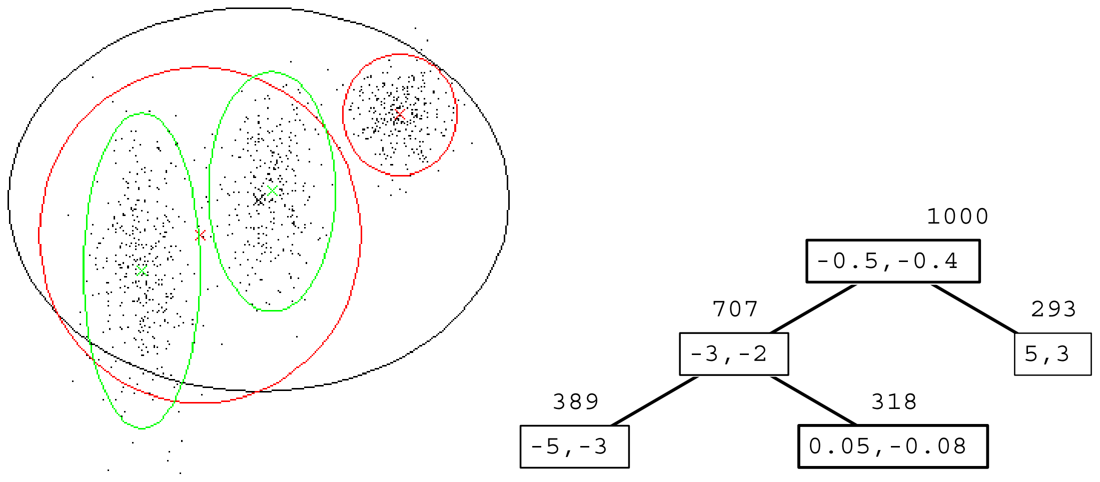

Figure 1. 2D representation of a naive mixture tree, fit to 1000 samples drawn from mixture of 3 Gaussians. The structure of the tree is shown on the left : the root models all 1000 samples by a naive Bayes component with mean (-0.5,-0.4). The two nodes below the root model subsets of size 707 and 293, respectively. The largest of those two nodes is split up once more, and the three leaves of the tree recover the three mixture components, which had weights of 0.4, 0.3, and 0.3.

we used naive Bayes densities for the mixture tree components, but any type of component density can be used. To date we have used spherical normal densities, naive Bayes (diagonal multivariate normal) densities, and probabilistic PCA densities [14, 11] as the mixture components.

Even when the tree is grown until every leaf contains a single sample, one can arbitrarily *cut* the tree and obtain a good approximation to the underlying density, where a cut is defined as pruning subtrees below a certain level. This is in contrast to kd-tree based data structures, where the data is partitioned in a manner that is not related to any approximation principle, but rather for computational convenience. In section 2 it will become clear how mixture trees are fundamentally different from kd-trees in this respect.

Mixture trees are thus similar to general mixtures, but maintain the mixture components in a hierarchical data structure. This is similar to tree-structured vector quantization (TSVQ) [3], which has been used in texture synthesis [17] to speed up conditional sampling. However, we generalize TSVQ in that any parametric density type can be used as the mixture component. When using probabilistic PCA components [14, 11], mixture trees model the data in a manner similar to many other approaches that tessellate the space with subspace approximations to the data [4, 13]. PPCA mixture trees offer the additional advantage of being able to cut the tree arbitrarily to recover a different subspace tessellation. This also sets it apart from the bottom up hierarchical mixtures described in [16]. Finally, mixture trees are different from the "mixtures of trees" in [6], which are standard mixtures using a tree-structured Bayes network.

2 Building Mixture Trees

Algorithm 1 Building a mixture tree

```
given: sample set ($X_0$), minimum number of samples in a node ($rmin$), branching factor ($k$), and chosen parametric form for component densities ($P(x|\theta)$)
do
1. estimate the root component $\theta_0$ using all samples $X_0$
2. starting with the root, recursively create all nodes n:
    given: node component $\theta_n$ with assigned samples $X_n$
    do
    a) if $|X_n| < rmin$ then the node is declared a leaf: $n := Leaf[\theta_n, X_n]$
    b) else, using EM, estimate the mixture $\sum_{i=1}^{k} \pi_i P(x|\theta_i)$ from the samples $X_n$
    c) create $k$ children $c_i$ and redistribute the samples $X_n$ among them: for each sample $x_{nj}$, randomly choose a child $c_i$ according to the class posteriors
        $$P(i|x_{nj}, \{\pi_i, \theta_i\}) = \frac{\pi_i P(x_{nj}|\theta_i)}{\sum_{i'=1}^{k} \pi_{i'} P(x_{nj}|\theta_{i'})} \quad (1)$$
    d) create $n$ as an internal node with children $c_i$, $n := Node[\theta_n, \{\pi_i, c_i\}]$
    e) recurse on each of the children $c_i$
```

Several greedy algorithms can be considered for building mixture trees. We implemented a top-down method, given as Algorithm 1. First the root node component is estimated, satisfying invariant 1 for the root. Then, we recursively estimate the mixtures at each internal node using expectation-maximization (EM), satisfying invariant 2. The recursion bottoms out when a node contains less than a minimum amount of data points.

The key step in the algorithm is step 2.c: after a call to EM, the data is distributed *probabilistically* among the children, according to the class posterior estimates, as given by (1). This (a) implements a multi-resolution divide-and conquer strategy to estimate the density at a higher resolution, and (b) equally important, approximately ensures invariant 1 for the children c_i of node n. It can be proved that for an infinite number of samples that are actually distributed as $\sum_{i=1}^{k} \pi_i P(x|\theta_i)$, the samples assigned to child c_i in this way will be distributed according to $P(x|\theta_i)$, the i^{th} mixture component assigned to child c_i. When these assumptions are violated, the invariant holds only approximately.

Another key point to note is that, *after step 2.b, there are two estimates for the density of the sample X_n*, as we have:

$$X_n \sim P(x|\theta_n) \quad \text{and} \quad X_n \sim \sum_{i=1}^{k} \pi_i P(x|\theta_i)$$

In other words, the samples X_n assigned to the subtree rooted at node n and stored at the leaves of the subtree, are modeled in two ways at node n: as one component $P(x|\theta_n)$, the parameters θ_n of which are stored at the node n itself, *and* at a higher resolution as a mixture of the child components $P(x|\theta_i)$. This holds recursively: if the subtree is grown recursively below the children c_i, *every* cut of the subtree below n is an approximation of the empirical density of X_n, with the finest resolution obtained by the *leaves mixture*. The mixture weights of such a cut is obtained recursively: simply multiply all the mixture weights stored in the nodes on the path to the root (which has weight 1).

3 Fast Conditional Sampling

One application where mixture trees excel is fast conditional sampling (FCS) from a conditional density. If, for two subsets x and y of the variables, the component density $P(x, y)$ can be factored conveniently according to the chain rule $P(x, y) = P(x)P(y|x)$, then we can easily obtain the conditional density $P(y|\bar{x})$ for any given value $\bar{x}$ for x. In fact, under those assumptions *any* mixture can easily be converted to a conditional density, which is itself a mixture (in what follows we use the notational shortcut $P_i(.) = P(.|\theta_i)$):

$$P(y|\bar{x}) = \frac{\sum_i \pi_i P_i(\bar{x}) P_i(y|\bar{x})}{\sum_i \pi_i P_i(\bar{x})} = \sum_i \pi_i' P_i(y|\bar{x})$$

i.e., again a mixture with new, $\bar{x}$ specific components $P_i(y|\bar{x})$ and mixture weights π_i':

$$\pi_i' \triangleq \frac{\pi_i P_i(\bar{x})}{\sum_j \pi_j P_j(\bar{x})} \tag{2}$$

When factored component densities $P_i(x, y) = P_i(x)P_i(y)$ are used, the new components $P_i(y|\bar{x}) = P_i(y)$ are trivially obtained and do not depend on $\bar{x}$. This is the case for naive Bayes densities, where one can easily condition on an arbitrary subset of the variables.

Algorithm 2 Fast Conditional Sampling

```
given: mixture tree (rooted at node $n$), conditioning variables ($\bar{x}$), and a threshold $t$
do
1. for each child $c_i$ of $n$, compute mixture weight $\pi_i'$ (2)
2. choose $c_i$ with probability $\pi_i'$
3. if $\pi_i' < t$ or $c_i$ is a leaf, sample $y \sim P_i(y|\bar{x}, \theta_i)$
4. else $n := i$, goto step 1.
```

While the above holds for arbitrary mixtures, a *mixture tree* allows for very fast sampling from $P(y|\bar{x})$ by adaptively approximating the conditional density $P(y|\bar{x})$, increasing the resolution of the approximation where the value $\bar{x}$ of x induces a large probability mass over y. Indeed, the key computational shortcut that a mixture tree affords us, is that we can cut the tree arbitrarily to obtain the conditional mixture. In particular, we can approximate an entire subtree by its root component whenever π_i' falls below a threshold t. Algorithm 2 describes the pseudo-code for the FCS algorithm.

4 Fast Approximate Nearest Neighbors

Another closely related application of mixture trees is fast approximate nearest neighbor search (FANN). Given a data point $\bar{z}$, we want to find the sample point z from the original dataset that is closest to $\bar{z}$. Often, it is easier to find a sample point that is not the closest yet close enough to $\bar{z}$. Mixture trees provide one such mechanism: we first find the mixture component that $\bar{z}$ is most likely to belong, and then search exhaustively for the closest point among the sample points contained in this component. The search for the most appropriate mixture component is computationally similar to the special case of fast conditional sampling where *all* variables are known. Specifically, we need to find the mixture component k that maximizes the mixture weight $\pi_k P(\bar{z}|\theta_k)$.

We can perform this search quickly by hierarchically refining the resolution at which to search for the best mixture component. Starting at the root, we recursively pick the component with the maximum mixture weight until either the mixture weight falls below the desired threshold or we reach a leaf node, where we perform the exhaustive search. Thus, mixture trees can be used as a hybrid parametric/memory-based model, where the data-driven cut through the tree adaptively shapes the approximation to be closer to the original data where the extra work is warranted.

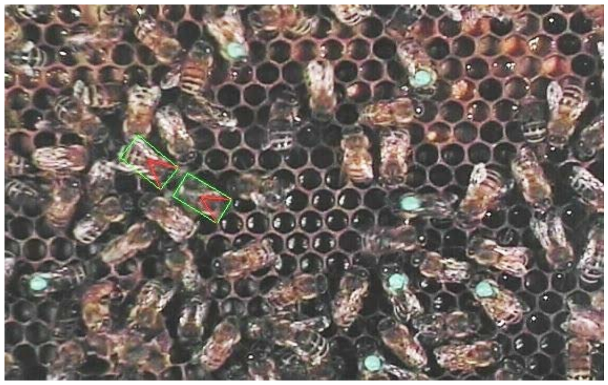

Figure 2. A dancer-follower pair of honeybees tracked in a video stream. The dancing bee (right) and the potential recruits (left) are being tracked. To track these bees effectively, a model for the follower bee motion relative to the dancer bee location can be built from the collected tracks.

5 Modeling Behavioral Data

We applied mixture trees to model the behavior of insects in order to more accurately track them as well as analyze their behavior. E.g. honeybees execute a complicated dance in order to communicate location and distance to a food source. Figure 2 shows a dancer-follower pair of bees being tracked in video. We obtained a database of 5567 pairs of relative dancer locations and the resulting follower motions (3 DOF each) using appearance based particle filters. The mixture tree for this data is shown in Figure 5. The first panel shows the original data, projected onto 2 dimensions, the relative XY position of the dancer bee. The center of the crescent-shaped distribution is the location of the follower bee (0,0). In the result shown, we used $rmin = 10$ and branching factor $k = 2$.

We used the honey bee dataset to evaluate fast conditional sampling, showing that we are able to sample from the distribution in real time. Figure 3 shows that FCS speeds up conditional sampling by orders of magnitude. We sampled y given 10000 randomly generated x points and repeated the experiments with seven different threshold values for π'_i (conditional mixture weights). The sampling rate obtained without any hierarchical information, i.e. using all 855 leaves of the mixture tree is 215 samples/sec on average. In contrast, when sampling using FCS, the average number of mixture components varies between 3.31 and 69.32, while the sampling rate is between 2538 and 83333 samples/sec. Hence, FCS is able to achieve a speed up of 11.92 to 368.73 times depending upon the desired accuracy of the distribution of samples.

Finally, we evaluated the modeling power of mixture trees compared to other modeling techniques by comparing it with EM. First, we have chosen two synthetic Gaussian mixtures shown in the left panel of figures 4 and 6. From each distribution, 5000 samples are generated to which mixture trees and EM are applied to estimate the mixtures. The resulting mixtures are shown in the middle and right panels respectively. Additionally, we have measured Kullback-Leibler divergence (KLD) between the original and the estimated mixtures where each KLD is shown below the corresponding panels. As shown in figures 6 and 4, mixture tree estimates are comparable to EM-based estimates. All of the obtained KLD results are smaller than 0.1 (0.014,0.009,0.062,0.065), showing that mixture tree and EM are both model the data well. The results indicate that mixture trees model data effectively, with the additional advantage that they can be used for fast conditional sampling.

t	Mixture size	Sampling rate	Speed up
0.5%	69.32	2538	11.92
1%	49.84	3968	18.63
2%	32.18	6536	30.98
5%	16.02	13389	63.45
10%	8.64	28571	132.89
20%	4.97	47619	226.76
40%	3.31	83333	368.73

Figure 3. Regular conditional sampling vs. FCS: t is the threshold, mixture size is the average number of active mixture components for a given t, sampling rate is the average number of samples/sec, and speed up is the ratio of FCS sampling rate to regular-conditional-sampling rate.

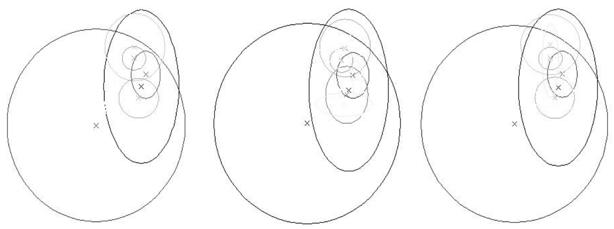

Figure 4. Left: 4D mixture projected into 2D; Middle: mixture tree 1 (leaves) which is built upon the samples from the original mixture 1 (10 classes, KLD=0.014416); Right: EM-based mixture 1 (10 classes, KLD=0.009088).

6 Texture Synthesis

We have also experimented with texture synthesis as an application of FCS and FANN. Given an input image of a texture, the goal of texture synthesis is to generate a potentially larger output image of the same texture. We use mixture trees to model the distribution of pixel *neighborhoods* in the input image. Each component of the mixture is modeled using PPCA. For synthesis, output neighborhoods are constructed by sampling from the mixture tree. We have experimented with two techniques for synthesis: (i) conditional sampling, given partial neighborhoods (uses FCS) and (ii) nearest neighbor search and blending (uses FANN). Both approaches are illustrated in Figure 7.

In (i), given a portion of the neighborhood, we synthesize the remaining portion by performing conditional sampling using our FCS algorithm. Starting with random noise, we synthesize the output neighborhood-by-neighborhood in scan-line order: for each neighborhood, we sample y given $\bar{x}$ and update the y-portion of the output (Figure 7a). We allow successive neighborhoods to overlap with each other to ensure consistency between them. Figure 7c shows the synthesis result using this approach.

In (ii), we synthesize the output texture by refining it iteratively, starting with random noise. During every iteration, we consider output neighborhoods in scan-line order and for each such neighborhood, find the closest input sample using FANN. The output is then updated by blending this input (neighborhood) sample with the output. We run multiple iterations – two to three are usually enough – of this process to obtain the final output texture. Figure 7d shows the synthesis result using this approach.

The synthesized textures in Figure 7 are 128×128 in size. They have been synthesized using neighborhoods as large as 21×21 pixels. Even though the resulting sample points reside in a very high dimensional space (1323 dimensions with three color channels), we are still able to synthesize our results fairly quickly – each iteration takes 11 seconds and 14 seconds using FCS and FANN respectively.

7 Conclusion

Mixture trees are elegant, easy to build, and very fast at run-time, especially when used in a conditional sampling context. They improve on mrkd-trees and generalize tree-structured vector-quantization. We have presented a recursive top-down algorithm that enforces the two mixture tree invariants by probabilistically re-distributing the data of a node among its children. The resulting tree models data accurately and can be cut arbitrarily to yield a good approximation to the underlying data density, which is the key property used in fast conditional sampling.

Opportunities for future work are a more thorough, quantitative comparison between mixture trees and competing data structures, as well as research into automatic, model-selection-based criteria to stop growing the tree.

Acknowledgments This work was funded under NSF Award IIS-0219850, "ITR: Observing, Tracking and Modeling Social Multi-agent Systems". Tucker Balch provided the bee colonies in the Georgia Tech BORG lab, while Zia Khan obtained the bee tracking data. We would like to thank Alex Gray for a fruitful conversation about our ideas.

References

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[2] A. Efros and T. Leung. Texture synthesis by non-parametric sampling. In *Intl. Conf. on Computer Vision (ICCV)*, pages 1033–1038, 1999.

[3] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.

[4] N. Kambhatla and T. K. Leen. Fast non-linear dimension reduction. In *Advances in Neural Information Processing Systems (NIPS)*, pages 152–159, 1994.

[5] Z. Khan, T. Balch, and F. Dellaert. An MCMC-based particle filter for tracking multiple interacting targets. In *Eur. Conf. on Computer Vision (ECCV)*, 2004.

[6] M. Meila and M. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48, 2000.

[7] A. Moore. Very fast EM-based mixture model clustering using multiresolution kd-trees. In M. Kearns and D. Cohn, editors, *Advances in Neural Information Processing Systems (NIPS)*, pages 543–549. Morgan Kaufman, April 1999.

[8] A. Moore, J. Schneider, and K. Deng. Efficient locally weighted polynomial regression predictions. In *International Conference on Machine Learning (ICML)*. Morgan Kaufmann, 1997.

[9] S. M. Omohundro. Bumptrees for efficient function, constraint, and classification learning. In *Advances in Neural Information Processing Systems (NIPS)*. Morgan Kaufmann, 1991.

[10] K. Popat and R. W. Picard. Cluster based probability model and its application to image and texture processing. *IEEE Trans. Image Processing*, 6(2):268–284, 1997.

[11] S. Roweis. EM algorithms for PCA and SPCA. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[12] B. Stenger, A. Thayananthan, P. H. S. Torr, and R. Cipolla. Filtering using a tree-based estimator. In *Proc. 9th International Conference on Computer Vision*, volume II, pages 1063–1070, Nice, France, October 2003.

[13] M. Tipping and C. Bishop. Mixtures of probabilistic principal component analysers. *Neural Computation*, 11(2):443–482, 1999.

[14] M. Tipping and C. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society*, Series B 61(3):611–622, 1999.

[15] J. K. Uhlmann. Metric trees. *Applied Mathematics Letters*, 4(5), 1991.

[16] N. Vasconcelos and A. Lippman. Learning mixture hierarchies. In *NIPS*, 1998.

[17] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In K. Akeley, editor, *SIGGRAPH*, pages 479–488, 2000.

[18] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1993.

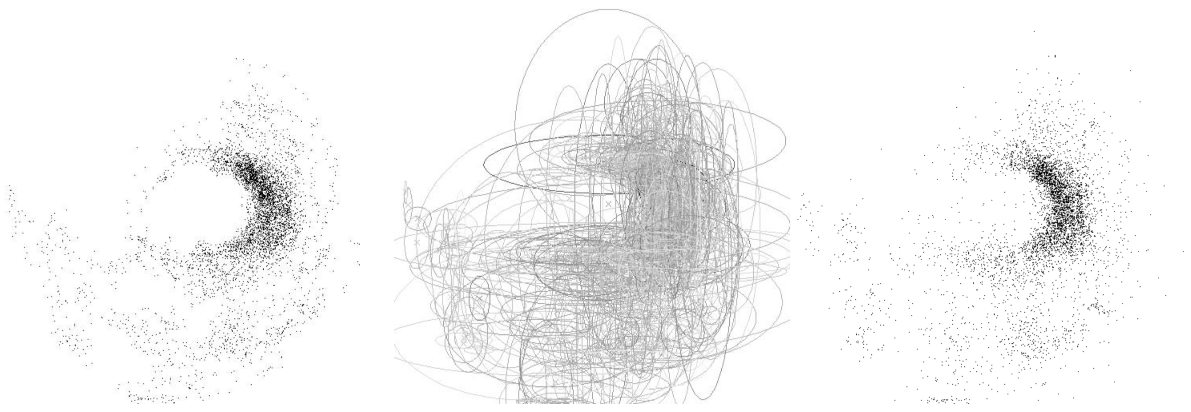

Figure 5. Mixture tree created from behavioral data collected from dancer-follower pairs of honeybees. Left: original 6D data projected in 2D, the relative XY location of the dancer bee; Middle: the leaves mixture. Darker covariance ellipses have more weight; Right: 5567 samples from the leaves, showing we are able to accurately model the original density.

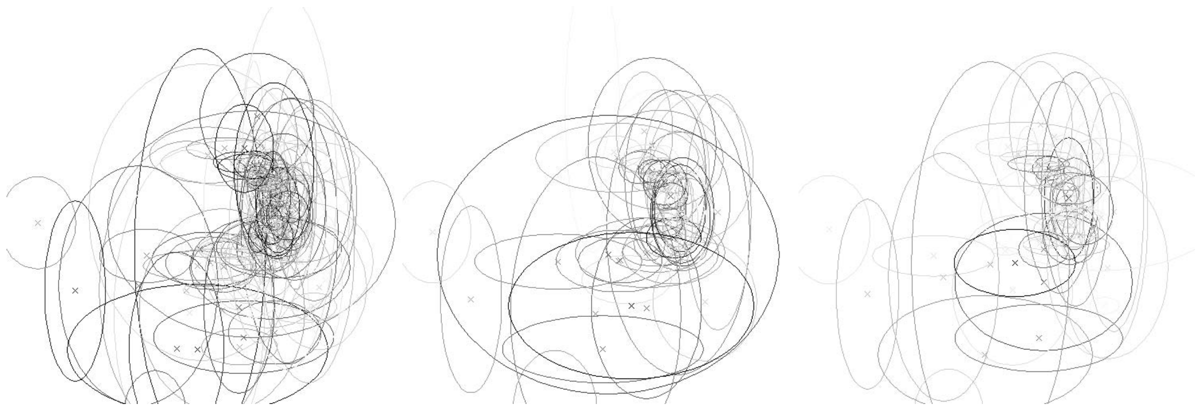

Figure 6. Evaluating the modeling power of mixture trees, see text. Left: Original mixture 2 (130 classes); Middle: Mixture tree 2 (leaves, 64 classes, KLD=0.061614); Right: EM-based mixture 2 (64 classes, KLD=0.065000).

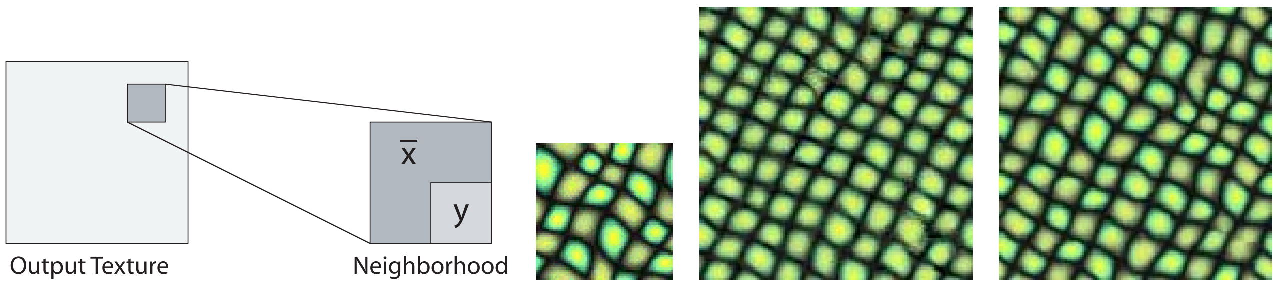

Figure 7. Texture Synthesis. (a) Synthesis using FCS: $\bar{x}$ and y are respectively the known and unknown portions of a neighborhood, (b) input texture, (c) texture synthesized using FCS, and (d) texture synthesized using FANN. See also text.