Probability Trees

Michael D. McCool and Peter K. Harwood

{mmccool,pkharwoo}@cgl.uwaterloo.ca
Computer Graphics Laboratory
Department of Computer Science, University of Waterloo,
Waterloo, Ontario, Canada N2L 3G1

Abstract

A k-D tree representation of probability distributions is generalized to support generation of samples from conditional distributions. An interpretation of the approach as a piecewise linear warping function is provided which permits a priori stratified sample generation. The representation is related to higher-order spline estimators and representations via projection. An application in glyph-based volume visualization is presented.

Keywords: sampling, antialiasing, Monte Carlo integration.

1 Introduction

Random quantities have numerous uses in computer graphics; many, but not all, are related to Monte Carlo integration in antialiasing [2, 3, 6, 9, 14, 20] and global illumination [2, 3, 9, 11, 23]. Generally speaking, whenever a difficult integral needs to be evaluated or a parameter estimated, a Monte Carlo approach can be used—although as a last resort when an analytic approach is infeasible.

The most basic Monte Carlo integration approach chooses uniform random samples $\{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N\}$ over some domain Ω and estimates the integral of some function f over that domain by averaging:

$$\eta_0 = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i) \approx \int_\Omega f(\mathbf{x})\, d\mathbf{x}$$

While very general, the main difficulty with this approach is that basic Monte Carlo integration has terrible convergence; the estimate η_0 is a normally distributed random variable with a variance that decreases only by $O(1/N)$. On the other hand, Monte Carlo integration works well even when the integrand is high-dimensional, defined over a non-Euclidean domain, or against an alternative measure. This flexibility is the technique's main advantage.

Several techniques have been proposed to speed convergence:

Stratified sampling generates samples distributed "evenly" over the domain Ω by binning, so the histogram of the actual sample distribution converges more rapidly to a uniform value.

Importance sampling generates samples non-uniformly, replacing some dominant factor of f with a different measure $\mu(\mathbf{x})$ over Ω.

Weighted estimators use information about the variance and spatial distribution of the samples to improve the convergence of the estimate.

Intelligent application of these techniques is essential in any practical Monte-Carlo based system.

This paper analyses and refines a basic tool that can be used when stratified samples need to be drawn from a multidimensional probability distribution: the k-D tree. This approach was first introduced in [11] where it was used for sequential stratification and antialiasing. We further the approach in this paper by illuminating the relationship of the k-D tree representation to warping, conditional distributions, and higher-order spline probability representation and estimation.

The connection to warping is useful because it allows the generation of stratified sampling patterns without having to store extra information in the tree. Conditional distributions are useful in that they permit knowledge of one variable to be used to refine the probability distributions of other variables, and in general permit one representation to be used in multiple ways. Finally, our projection-based interpretation of higher-order representations connects k-D tree and spline representations of probability distributions and suggests some new approaches to probability representation.

2 Overview

This paper is organized as follows. In Section 3 a short summary of the uses of probability distributions and random sampling in computer graphics will be presented. In addition, previous approaches to the modelling, manipulation, and generation of samples from arbitrary multidimensional probability distributions will be reviewed.

The k-D tree data structure will be motivated and presented in Sections 4–8. The main new algorithm presented is ***unification***, the generation of a random sample from a conditioned probability distribution. This algorithm will be presented in conjunction with the warping interpretation of the k-D tree approach which permits a priori stratification. Finally, we will present a connection between higher-order spline representations and marginal distributions.

3 Background

In this section, we first review applications of randomization and attempts to model and estimate probability distributions. The focus will be on applications that involve Monte Carlo integration and attempt to improve its convergence by generating specific sampling patterns.

3.1 Monte Carlo Integration

In the basic Monte Carlo integration technique, random samples of a function f are taken and averaged. Monte Carlo integration can be applied to rendering [1, 2, 3, 8], as the global illumination problem can be cast as a single integral equation [11]. Antialiasing and motion blur are other important rendering problems that require integration [3, 16].

The variance of the result in the basic Monte Carlo approach decreases with $O(1/N)$, which is very slow relative to other numerical integration techniques; the standard deviation of the error decreases only by $O(1/\sqrt{N})$. Convergence can be improved by stratification, importance sampling, and weighted estimators [9]—all techniques that modify the sampling pattern and/or the estimator.

3.1.1 Stratification

Stratification distributes samples evenly across the domain of integration using binning, in an attempt to avoid "clumping" of samples. Secondary stratification of marginal sample distributions and decorrelation of the samples can also improve convergence [17].

In order to perform stratification, some knowledge of how past samples were placed is required, either explicitly (in the form of stored samples) or implicitly, e.g. parametric estimation of the sample density, or a priori selection of bins. Stratification has limited usefulness in more than two dimensions [9, 19]. At best, it can improve the convergence of the variance to $O(N^{-2})$ in smooth regions of the integrand.

However, in antialiasing applications stratified sampling patterns also shape the correlation function of the aliasing noise, shifting its energy to higher frequencies and thus making it less noticeable to the human visual system [6]. This fact, together with the improvement of convergence, has inspired several algorithms for generating sampling patterns with desirable spatial characteristics [13, 14, 16, 17]. The theory of discrepancy analysis can be used to identify sampling patterns that are optimal relative to specific analytic image models [7, 18].

3.1.2 Importance Sampling

Importance sampling uses an estimate of the magnitude and/or variance of the integrand to guide sampling to more interesting parts of the integrand.

An *importance function* defines the probability distribution of the sampling pattern. The true mean can be correctly estimated from importance-distributed sampling by dividing the samples of the integrand by the value of the importance function at each sample point. It has been shown that using an importance function which is a dominating factor of the integrand best improves convergence.

There are many analytic techniques for generating non-uniform samples [4, 5, 12] distributed according to specific probability distributions. However, if the importance function is not known in advance, or is too difficult to integrate, invert, and/or bound analytically, a general technique based on approximation is useful.

An approximation to an importance function does not have to be very accurate or smooth to improve convergence, as long as the approximation can be bounded and the actual value of the approximate distribution can be accurately computed. However, it is more convenient if the importance function *is* in fact a good approximation to some factor of the integrand, as it is then unnecessary to divide out the importance function and multiply by a more accurate value for that factor.

Algorithms have been developed that permit the generation of stratified sampling patterns that can be refined [11, 20] and/or be simultaneously distributed according to an importance function [14]. This generalized form of stratification tries to improve convergence by choosing a sequence of samples whose histogram converges more rapidly to the target distribution than arbitrary samples drawn from the target distribution.

Finally, if samples are generated non-uniformly, they must be combined somehow without introducing bias. One technique that also improves convergence for $n < 4$ is weighted averaging, which uses the n-dimensional volume of the points closest to a particular sample to weight that sample. It has been shown that if the samples are generated independently and the volumes of the corresponding d-dimensional Voronoi cells are used as weights, then the convergence of the variance will be $O(1/N^{4/d})$—which is a great improvement in 1D, good in 2D, minor in 3D, and equal to or worse than crude Monte Carlo in higher dimensions. It is unclear just how stratification, importance sampling, and weighted reconstruction interact, however; the derivation in [25] depends explicitly on the properties of completely independent samples.

3.2 Representation of Probability Distributions

In the discussion of Monte Carlo sampling techniques above, two prior works present representations of n-dimensional probability distributions from which non-uniform sampling patterns can be generated: the k-D tree data structure in [11] and the tensor product B-spline model in [21].

The k-D tree data structure in [11] was presented in the context of hierarchical stratification and antialiasing. A k-D tree is a binary tree whose internal nodes subdivide space one dimension at a time, using axis-aligned splitting planes. This is illustrated in Figure 1 for the case of a two-dimensional domain.

Sequential stratification starts with a single cell and inserts a random sample into it. When another sample is needed, the cell is split. The old sample ends up in one half of the split cell, so the new sample

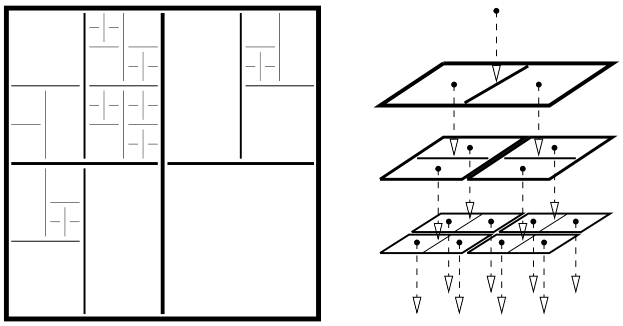

Figure 1: *A k-D tree adaptively subdivides space into cells by splitting along one dimension at a time. The representation in memory uses a binary tree data structure.*

is drawn from the other, empty subcell. This process is repeated recursively, using a probalistic selection strategy that results in a breadth-first subdivision of the tree and uniform or adaptive distribution of samples. Once all samples have been drawn, the integral can be estimated by forming a piecewise constant approximation to the sampled function using the value of the sample in each cell. This amounts to weighting each sample by the size of the cell in which it lies.

This technique requires the storage of samples, but could be used to incrementally drive a sampling pattern to a desired importance distribution, while keeping samples from becoming too clumped together. It also automatically provides a weighted averaging scheme, called hierarchical integration, for recombining samples and improving convergence for $n < 4$.

The k-D tree idea was refined in [20], where the idea of using a per-node sample counter rather than stored samples to control stratification was introduced. This approach can still drive a sampling pattern quickly towards a desired global probability distribution, but requires less storage.

In [21] tensor-product splines are proposed for the representation of probability distributions. If the distribution is represented by

$$P(\mathbf{x}) = \sum_i P_i B_i(\mathbf{x})$$

where the B_i are spline basis functions, then an unbiased estimator of the probability distribution that generated a sequence of samples $\mathbf{x}_i$ is given by

$$P_i = \frac{1}{N} \sum_j B_i(\mathbf{x}_j).$$

Generation of samples distributed according to a B-spline basis function can be done using the urn interpretation of B-splines, which for a uniform B-spline basis function amounts to the sum of n uniform random variables. Generating a sample from the complete spline representation interprets $P(\mathbf{x})$ as a mixture: choose some i based on the discrete probabilities P_i, then distribute the sample according to B_i. This sample generation technique is *not* based on warping, and so it is not possible to stratify samples a priori, although it would be wise to at least drive the actual selection of the i's to match the discrete distribution given by the P_i's.

In Section 8 we will show how the k-D tree and B-spline approaches are related via projection, and will give the estimation technique above a simple geometric interpretation.

3.3 Other Applications

Two other applications of multidimensional sampling patterns are worth mentioning in light of the algorithm and data structure presented in this paper: glyph distribution for volume visualization [22] and dithering.

We give an example of the first application at the end of this paper. Any algorithm which can generate stratified samples distributed according to an importance function can be used for dithering, although producing dithering patterns for real printers requires attention to clustering, a problem not addressed by the data structure given here.

4 Functional k-D Trees

The probability tree data structure is built around a k-D tree representation of a real-valued importance function. This representation has the desirable properties of simplicity, space and time efficiency, freedom from dependence on a fixed number of dimensions, and hierarchical adaptability. We obtain this result by sacrificing smoothness: a piecewise constant approximation is used.

A k-D tree subdivides a rectangular k-dimensional domain into k-dimensional rectangular cells at its leaves. By specifying a constant approximation to a function over each cell, we can approximate the function over the entire domain. To adaptively subdivide, an upper bound on the approximation error is needed; further subdivision can be performed if the error of the piecewise constant approximation exceeds a threshold.

The restriction of the k-D tree representation to a bounded, rectangular domain can be removed by introducing an appropriate nonlinear coordinate space mapping. The Jacobian of this mapping can be built into the probability distribution as needed. As long as the mapping is a homeomorphism, the built-in flexibility afforded by the hierarchical representation can adapt to the distortion.

There are several variations on this theme. The splitting plane can always be in the center of the cell being split, or can be movable. The dimensions being split can cycle in a strict order, or the dimension being split can be specified in each internal node.

Here we consider only the simplest case: the splitting plane is always in the center of the cell being split, and the dimensions being split cycle in strict order. In addition, we assume that the approximation to the function in our leaf cells is simply a constant, so the overall approximation is piecewise constant.

There are obvious disadvantages to these simplifications. The volume of a cell can only decrease by a factor of two at each level of the tree, and so adapting the hierarchy to a function with a lot of detail in a concentrated area will require many levels in the tree—the approximation can only converge linearly. In addition, one can easily construct functions for which the dimension rotation rule will construct unnecessary subdivisions, for example $f : x, y \mapsto \sin^2(x)$. On the other hand, the subdivision produced *will* be adaptive.

Constant approximation also has the very important advantage that since the basis functions don't overlap, we don't have to search an exponential number of leaves to find all the coefficients affecting a point.

A k-D tree approximation can easily be constructed bottom-up from a regular grid of samples of the function; given a maximum depth all the leaf cells are the same size. In addition, the storage per node in the tree is very small. In fact, at a minimum we simply have to distinguish an internal node from a leaf. To handle probability and a constant functional approximation, we will have to add a single floating-point or fixed-point number to each node and leaf.

Consider briefly how a specific leaf cell containing the spatial position $\mathbf{x} = (x_1, x_2, \ldots, x_k)$ in a k-D tree is found. As we descend the tree, the central dividing plane of each node is compared with each coordinate in turn. If x_i is less than the position of the splitting plane for dimension i, then $\mathbf{x}$ must lie in the lower cell of a node that splits that dimension; otherwise, $\mathbf{x}$ lies in the upper cell. In other words, the leaf cell containing $\mathbf{x}$ is found by a *rotating binary search.*

5 Non-uniform Generation

One approach to generating samples drawn from a non-uniform probability distribution uses the inverse of the cumulative distribution function. Alternatively, we can use a hierarchical approximation to a probability distribution, with analytic inversion over a simple approximation as the base case. Both of these approaches can be generalized to an arbitrary number of dimensions.

5.1 Cumulative Distributions

To generate samples distributed according to an arbitrary probability density function $f(x)$, the inverse $F^{-1}(y)$ of the cumulative distribution function $F(x)$ can be used, with $F(x)$ defined using

$$F(x) = \int_{-\infty}^{x} f(\xi)\, d\xi.$$

If y is a random variable over $[0,1]$, then $x = F^{-1}(y)$ will be distributed according to $f(x)$.

Since $f(x)$ is non-negative, $F(x)$ is always monotonic, and the inverse exists. Actually, since $f(x)$ could be locally zero, $F(x)$ might not be strictly monotonic. That happens not to matter; we can easily resolve the ambiguity in the evaluation of the inverse because samples are never generated over regions with probability zero.

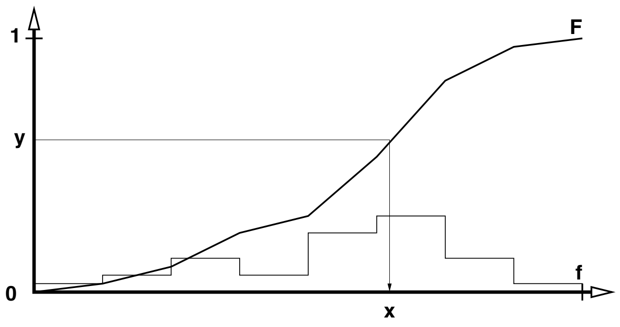

Figure 2: *The inverse of the cumulative distribution function can be used to generate samples distributed according to a given probability density. Here the probability density is piecewise constant, so the cumulative distribution is piecewise linear.*

Assuming we have a representation of $y = F(x)$, since F is monotonic given y we can find x using binary search as in Figure 2.

To generalize to a higher number of dimensions, we need to generate a univariate *marginal* cumulative distribution function by integrating out all but one parameter. Once we have inverted one dimension, we invert the resulting conditional cumulative distribution over the remaining dimensions. It does not matter in which order we eliminate variables [5, 4].

We can generalize this structure; the processes of marginalization and conditioning can be interchanged, and performed over regions rather than entire dimensions, leading to a *hierarchical cumulative distribution* which can be represented in a k-D tree.

5.2 Hierarchical Representation

Construction of a *probability tree* approximation to a probability distribution defined over a rectangular region R proceeds recursively as follows:

1. Base case: If an adequate approximation of the given probability density f over R is possible, create the approximation and return it as a leaf node.
2. Otherwise, create an internal node that splits dimension $(i+1) \bmod k$, if the last dimension split was i.
3. Using the splitting plane, subdivide the domain R into two non-overlapping subdomains P_0 and P_1.
4. Compute the marginal probabilities p_j for each subregion P_j by integrating the given probability density f over P_j.
5. Store the value $p_0/(p_0+p_1)$ at the node. This is the probability that a random sample point will appear in region P_0, given that it appears in region R, or $p_0 = \Pr(\mathbf{x} \in P_0 | \mathbf{x} \in R)$.
6. Recursively generate representations of the conditional probability densities $f(\mathbf{x}|\mathbf{x} \in P_j)$ for each subregion.

Once the tree is generated, to generate a random sample first choose k uniform random numbers $y_i \in [0,1]$, then descend the tree starting from the root. At a node splitting dimension i, descend the upper subtree if $y_i > p_0$, and the lower subtree otherwise. At each step, renormalize y_i to the total relative probability of the subtree. At a leaf, analytically invert the local approximation to the probability density. If the approximation is constant, the inversion is simply a linear interpolation.

A formalization of the descent algorithm is given in Figure 4. This descent algorithm can be interpreted as a multidimensional, rotating binary search

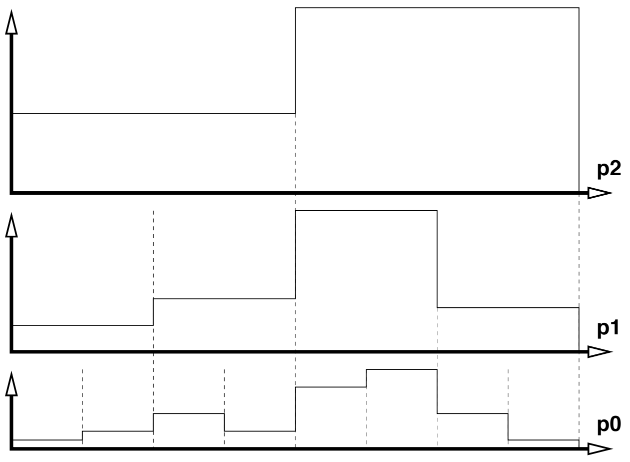

Figure 3: *A hierarchical representation of a probability distribution integrates over subregions.*

that inverts the (hierarchical) cumulative distribution function. The search region is contracted one dimension at a time by conditioning the probability distribution to smaller and smaller regions. At the leaves, we again invert the analytic form of the approximation to the probability distribution.

6 Unification

The unification algorithm presented in Figure 5 takes a k-D tree representation of a probability distribution over k dimensions and interlaces a rotating binary search for position with a rotating binary search to invert the probability distribution.

Suppose some number n of the variables are fixed, i.e. have given values. The $k-n$ others are free, and we would like to generate random values for them according to the conditional probability distribution given by the fixed variables. For each of the free variables, uniform random numbers between 0 and 1 are generated to seed the algorithm. Then the unification algorithm descends the tree. For fixed dimensions, the choice of child cell is made by comparing the fixed coordinate with the position of the splitting plane—doing a rotating binary search on spatial position. For the free dimensions, the choice of child cell is made by comparing the value of the corresponding random seed number with the threshold stored in the cell—doing a rotating binary search to invert the hierarchical cumulative distribution function. At a leaf node, we invert the free variables analytically.

We also return the joint probability density at the generated point, since that value is stored at the leaf.

```
REAL
Generate (INT k; REAL y[k], VAR x[k],
          REAL lower[k], upper[k])
  i := 0;
  WHILE NOT node INSTANCEOF Leaf DO
    // Update cell bounds
    middle := (lower[i] + upper[i])/2.0;
    // Choose upper or lower branch and
    // renormalize
    IF node.f < y[i] THEN
      lower[i] := middle;
      y[i] := (y[i] - node.f)/(1.0 - node.f);
      node := node.upperTree;
    ELSE
      upper[i] := middle;
      y[i] := y[i] / node.f;
      node := node.lowerTree;
    ENDIF;
    // Rotate to next dimension
    i := (i+1) mod k;
  ENDWHILE;
  // Invert normalized free variables
  FOR i:=0 TO k DO
    x[i] := y[i]*(upper[i]-lower[i])+lower[i];
  ENDFOR;
  // Leaf cell contains function value
  RETURN node.f;
```

Figure 4: *Pseudocode that generates a sample from a hierarchical representation of a probability distribution.*

Note that this value is *not* a sample of the conditional density. In order to compute the actual conditional density we would need to first generate a sample of the marginal density by walking the appropriate subset of the tree and summing all cells that intersect the $k-n$ dimensional subspace defined by the given combination of fixed and free parameters. Dividing the joint density by the value of the marginal density would give the value of the conditional density. Once we knew the marginal density for a given set of fixed parameters we could reuse it for all conditional samples.

It is not immediately apparent that interlacing these two binary searches in this way is valid. However, given the way we have constructed the probability tree, each subtree represents a probability distribution conditioned over just its subdomain.

As a base case of an inductive proof of the validity of this algorithm, we know the analytic inversion at a leaf is valid. Assuming the inversion of each subtree is valid, at a node splitting a free dimension we deter-

```
REAL
Unify (INT k; REAL y[k], VAR x[k];
       BOOL fixed[k],
       REAL lower[k], upper[k])
  i := 0;
  WHILE NOT node INSTANCEOF Leaf DO
    middle := (lower[i] + upper[i])/2.0;
    IF fixed[i] THEN
      IF middle < x[i] THEN
        lower[i] := middle;
        node := node.upperTree;
      ELSE
        upper[i] := middle;
        node := node.lowerTree;
      ENDIF;
    ELSE
      IF node.f < y[i] THEN
        lower[i] := middle;
        y[i] := (y[i] - node.f)/(1.0 - node.f);
        node := node.upperTree;
      ELSE
        upper[i] := middle;
        y[i] := y[i] / node.f;
        node := node.lowerTree;
      ENDIF;
    ENDIF;
    i := (i+1) mod k;
  ENDWHILE;
  FOR i := 0 TO k DO
    IF fixed[i] THEN
      x[i] := y[i]*(upper[i]-lower[i])+lower[i];
    ENDIF;
  ENDFOR;
  RETURN node.f;
```

Figure 5: *Pseudocode for the unification algorithm.*

mine if the generated sample point lies in the upper or lower subregion, and pick the appropriate conditioned probability distribution using the marginal cumulative distribution. At a node splitting a fixed dimension, similar reasoning applies: we condition the probabilities of the subtrees on the knowledge we have about the fixed parameter. Given the non-overlapping nature of the subregions, one subregion will have conditional probability 1 and the other will have conditional probability 0.

Note that each step of the unification algorithm involves *only* a comparison, perhaps some trivial computation to update the position of the splitting plane, and possibly a renormalization. We show renormalization on the fly; in fact, renormalization can be built into the values stored at each node. Therefore, the algorithm is relatively fast, depending only on the depth of the tree. With precomputed normalization the algorithm is also symmetrical, it being no more expensive to make a decision with respect to a free variable than with respect to a fixed one.

The probability thresholds define a dual tree with movable splitting planes; we could imagine an equivalent representation of a probability tree where every cell at a given depth had the same probability, with the spatial position of the splitting plane adjusted rather than the probability threshold.

The simple rotating-dimension approach to generating k-D tree representations of distributions may create unnecessarily deep trees, and this will impact both the running time and the storage cost. It is possible to extend this algorithm to trees with m-way splits at each internal node, with splitting dimensions stored at nodes, and with movable planes. An interesting question when the last two options are combined is the generation of *optimal* representations of probability distributions.

7 Stratified Sampling

The k-D probability tree as described is a warping function from $[0,1]^k$ to the domain of the probability distribution. Stratified importance sampled patterns can be generated by warping a uniform stratified sampling pattern.

In Figure 6 we show an k-D tree approximation to a function, and both a stratified and unstratified importance function sampling patterns distributed over it, along with histograms of these sampling patterns. The relative error tolerance is relatively large in these examples (1/10) to show the structure of the approximation. The uniform stratified sampling pattern was generated by jittering on a regular grid, with full subcell coverage.

A visualization of the warp induced by our k-D tree data structure is shown in Figure 7. Observe what happens to the shaded square in this diagram; the warp is *not* continuous: it can shear cells, breaking them up into subcells. At each level of the tree, each subdomain is stretched linearly to map a specific proportion of the input domain to one side or another of the splitting plane.

8 Higher-Order Representations

As pointed out in [15, 10], B-spline basis functions are just projections of appropriate convex polytopes from higher dimensions. In particular, B-spline basis

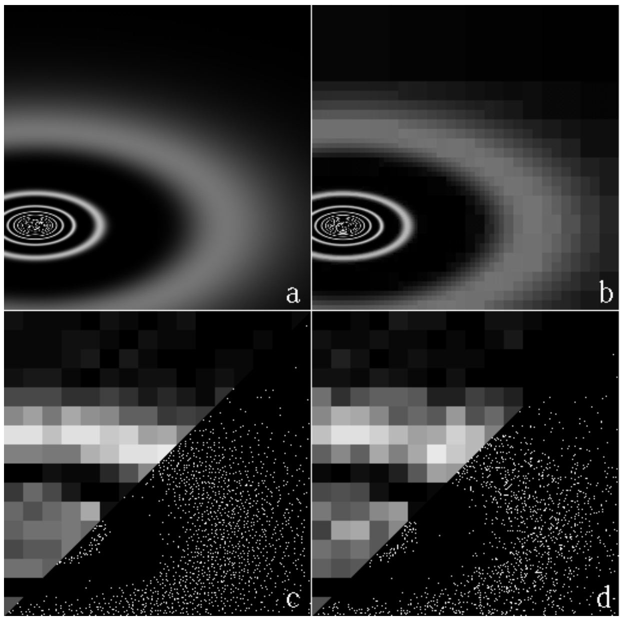

Figure 6: *Stratified sampling with a probability tree: a) the test function; b) the probability tree with* $\varepsilon = 0.1$*; c) a stratified importance weighted sampling pattern and histogram; d) a non-stratified importance weighted sampling pattern.*

functions can be represented as projections of simplices or hypercubes, and many of the familiar properties of B-splines—partition of unity, subdivision, recurrence—can be derived from geometric considerations.

This observation can be used to connect the B-spline representation in [21] with the k-D tree representation. Consider Figure 8. In this diagram, a piecewise linear probability distribution is generated by projecting a piecewise constant probability distribution along a skew axis. Each box in the constant approximation projects down onto a single B-spline basis function. The "ray" R pierces several cells of the k-D tree. If this ray were a sample and we needed to determine how to distribute its contribution among the cells in order to estimate a probability density, one approach would be to take the size of the intersection of the ray with each cell. But this size is just the value of the basis function B_i at the position of the ray, so we obtain the following estimator:

$$P_i = \frac{1}{N} \sum_j B_i(\mathbf{x}_j).$$

In general, the "ray" will not be one-dimensional, but the principle remains the same. The particular projection shown gives rise to uniform B-splines, but projections for tensor product splines, simplex splines, Bézier splines, and box splines can also be defined.

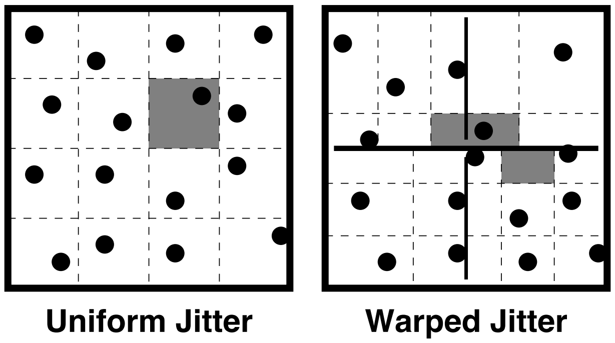

Figure 7: *Stratified sampling is accomplished by piecewise linear warping of a uniform stratified sampling pattern.*

This diagram immediately raises these new possiblilities, but also points out several flaws in both the k-D tree and spline representations of probability distributions. First of all, note how sample generation works: a sample is distributed across a cell, but then the extra dimension is thrown away when the sample is projected. It will be very hard to stratify samples in the "marginal distribution" without an auxiliary data structure; stratification within the k-D tree will help, but will not be ideal. Secondly, note that the k-D tree data structure is redundant; two of the cells that intersect the ray shown project onto exactly the same support. This redundancy can be removed by using a DAG rather than a tree, but that complicates subdivision, as now a search-and-merge step is required. Therefore, the k-D tree is probably not the best representation of a spline—although note how simple the representation of hierarchical spline basis functions are. Finally, note that evaluating the value of a density is more complex with a spline representation—the contributions of several basis functions need to be summed.

9 Glyph Distribution

As shown in [22], three-dimensional stratified importance-weighted sampling patterns can be useful for volume visualization. Glyphs—small, easily rendered shapes—are generated at random points according to a function that identifies the interesting parts of the volume. The shape, orientation, size, and/or colour of the glyph may then be modified

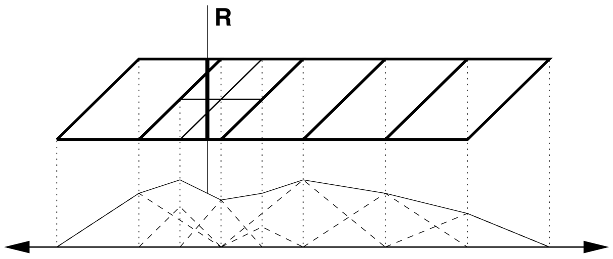

Figure 8: *A higher-order hierarchical spline probability density approximation can be viewed as the off-axis projection of a piecewise constant approximation.*

according to the values of the three-dimensional vector or scalar function being visualized. Stratification minimizes the number of collisions between glyphs. Both volume densities and isosurfaces of scalar functions can be visualized by picking the importance function appropriately.

In Figures 9, 10, and 11 we show the result of using our probability tree to represent a scalar volume (HIPIP, from the Chapel Hill test data set) and generate glyph distributions. We orient the glyphs according to the gradient of the scalar volume density, estimated according to the 3×3 operator given in [26].

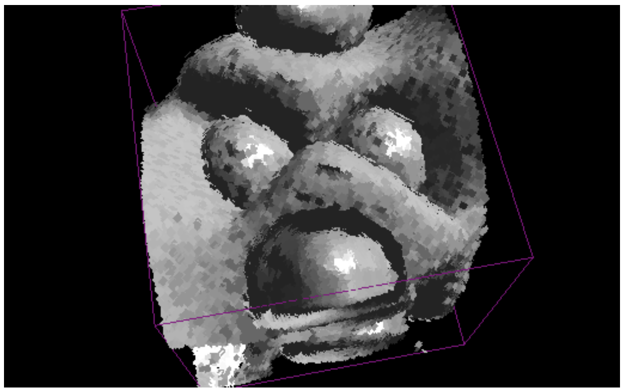

Figure 9: *A scalar volume density is visualized by using the scalar value to control the density and orientation of glyphs.*

Using the hierarchical Poisson disk algorithm in [14] to generate sampling patterns as in [22] has an advantage here: denser sampling patterns reuse old glyph positions, providing a smoother transition when the number of glyphs changes. The current algorithm does *not* have this property, unfortunately, but has the compensation that "glyph slices" can

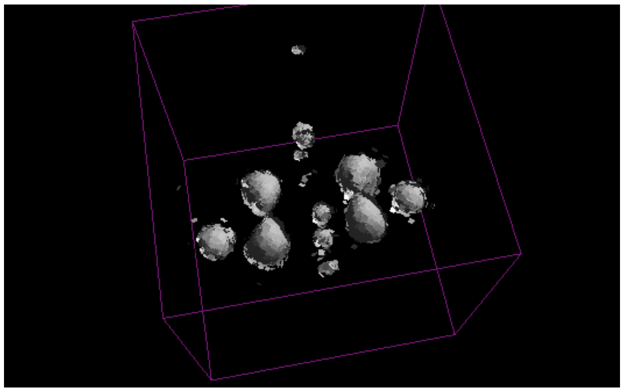

Figure 10: *A isosurface is visualized by using a derived importance function (here a Gaussian about the threshold value).*

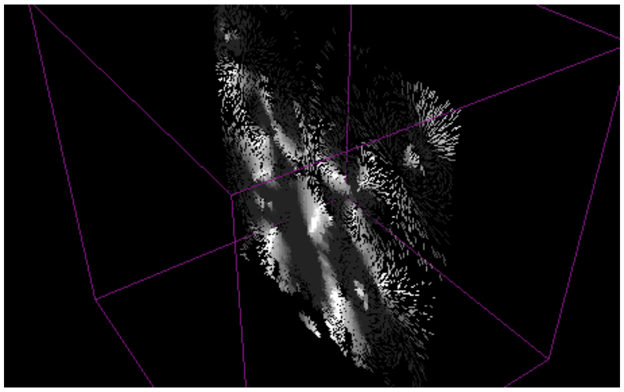

Figure 11: *A glyph slice can be extracted by fixing one dimension and using unification to distribute the glyphs in the remaining dimensions.*

be extracted by fixing one coordinate before unification, as in Figure 11. Also, no precomputed tables are required—other than the probability tree representation of the importance function. A probability tree representation of a scalar volume could also be used for direct volume rendering, after [24].

10 Conclusions

A simple data structure has been presented that can represent a multidimensional probability distribution, and can generate samples drawn from that probability distribution or from various conditional distributions based on the original distribution. An interpretation as a warp permits space-efficient stratified sampling.

Acknowledgements

The Chapel Hill volume rendering test data set was used to test our glyph rendering algorithm. I would like to thank Takafumi Saito for seeking the first author out at SIGGRAPH '94 to provide a copy of [22], which indirectly inspired the current paper. I would like to thank NSERC and ITRC for providing funding in direct support of this research and also to the Computer Graphics Laboratory at the University of Waterloo. MathSci, SciDex, the SIGGRAPH online database, and the MultiText multi-bibliography search engine were used to obtain references.

References

[1] P. Blasi, B. Le Saec, and C. Schlick. An importance driven monte-carlo solution to the global illumination problem. *Fifth Eurographics Workshop on Rendering*, pp. 173–183, Darmstadt, Germany, June 1994.

[2] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. *SIGGRAPH '84 Conference Proceedings*, pp. 137–145, July 1984.

[3] R. L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72, January 1986.

[4] J. Dagpunar. *Principles of random variate generation.* Oxford University Press, New York, 1988.

[5] L. Devroye. *Non-uniform random variate generation.* Springer-Verlag, New York, 1986.

[6] M. A. Z. Dippé and E. H. Wold. Antialiasing through stochastic sampling. *SIGGRAPH '85 Conference Proceedings*, pp. 69–78, July 1985.

[7] D. P. Dobkin and D. P. Mitchell. Random-edge discrepancy of supersampling patterns. *Proceedings of Graphics Interface '93*, pp. 62–69, Toronto, Ontario, May 1993. Canadian Information Processing Society.

[8] P. Dutre and Y. D. Willems. Importance-driven monte carlo light tracing. *Fifth Eurographics Workshop on Rendering*, pp. 185–194, Darmstadt, Germany, June 1994.

[9] A. S. Glassner. *Principles of Digital Image Synthesis.* Morgan Kaufman, 1995.

[10] K. Hollig. Box splines. TR 640, Computer Sciences Department, University of Wisconsin, Madison, WI, April 1986.

[11] J. T. Kajiya. The rendering equation. *SIGGRAPH '86 Conference Proceedings*, pp. 143–150, August 1986.

[12] D. E. Knuth. *Seminumerical Algorithms.* Addison-Wesley, 1969.

[13] M. E. Lee, R. A. Redner, and S. P. Uselton. Statistically optimized sampling for distributed ray tracing. *SIGGRAPH '85 Conference Proceedings*, pp. 61–67, July 1985.

[14] M. D. McCool and E. Fiume. Hierarchical poisson disk sampling distributions. *Proceedings of Graphics Interface '92*, pp. 94–105, May 1992.

[15] M. D. McCool. Analytic antialiasing with prism splines. *SIGGRAPH '95 Conference Proceedings*, pp. 429–436, August 1995.

[16] D. P. Mitchell. Generating antialiased images at low sampling densities. *SIGGRAPH '87 Conference Proceedings*, pp. 65–72, July 1987.

[17] D. P. Mitchell. Spectrally optimal sampling for distributed ray tracing. *SIGGRAPH '91 Conference Proceedings*, pp. 157–164, July 1991.

[18] D. P. Mitchell. Ray tracing and irregularities of distribution. *Third Eurographics Workshop on Rendering*, pp. 61–69, May 1992.

[19] D. P. Mitchell. Consequences of stratified sampling in graphics. *SIGGRAPH '96 Conference Proceedings*, pp. 277–280, August 1996.

[20] J. Painter and K. Sloan. Antialiased ray tracing by adaptive progressive refinement. *SIGGRAPH '89 Conference Proceedings*, pp. 281–288, July 1989.

[21] R. A. Redner, M. E. Lee, and S. P. Uselton. Smooth B-spline illumination maps for bidirectional ray tracing. *ACM Transactions on Graphics*, 14(4), October 1995.

[22] T. Saito. Real-time previewing for volume visualization. *1994 Symposium on Volume Visualization*, pp. 99–106. ACM SIGGRAPH, October 1994.

[23] E. Veach and L. J. Guibas. Optimally combining sampling techniques for monte carlo rendering. *SIGGRAPH '95 Conference Proceedings*, pp. 419–428, August 1995.

[24] J. Wilhelms and A. Van Gelder. Multi-dimensional trees for controlled volume rendering and compression. *1994 Symposium on Volume Visualization*, pp. 27–34. ACM SIGGRAPH, October 1994.

[25] S. Yakowitz, J. E. Krimmel, and F. Szidarovszky. Weighted monte carlo integration. *SIAM J. Numer. Anal.*, 15(6), December 1978.

[26] S. W. Zucker and R. A. Hummel. A three dimensional edge operator. *Pattern Analysis and Machine Intelligence*, 3(3):324–331, May 1981.