ANS MC2015 - Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method · Nashville, Tennessee · April 19–23, 2015, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2015)

ACCELERATED PATH GENERATION AND VISUALIZATION FOR NUMERICAL INTEGRATION OF FEYNMAN PATH INTEGRALS FOR RADIATIVE TRANSFER

Paul Kilgo and Jerry Tessendorf
School of Computing
Clemson University
100 McAdams Hall, Clemson, S.C. 29634
pkilgo@clemson.edu; jtessen@clemson.edu

ABSTRACT

Previous work in framing radiative transfer in terms of Feynman path integrals (FPI) is extended. The path sampling algorithm is extended with a constant time target function for root finding. Benchmarking for root finding schemes is reported. Additional analysis of the path weighting scheme is given. Utilization for the Metropolis algorithm is discussed. A log-normal fit for convergence of the integral sum is found. A visualization tool for attributed path data is presented.

Key Words: Radiative transfer, Feynman Path Integrals, Metropolis algorithm, Visualization

1 BACKGROUND

Volumetric rendering is ubiquitous in the visual effects industry. It is used to realistically render lighting effects, scattering media, and other natural phenomena. An approach which is quite common in volume renderers is based upon what is known as the *single scattering* approximation of the radiative transfer equation, originating from the theory of radiative transfer. Images produced via single scattering renderers are often sufficient for many purposes, although greater physical accuracy is often required for some desired effects. For example, a single scatter volume renderer would not pick up on the interactions from the media which do not lie on the straight paths from the image plane to the scattering locus to the light source.

To consider parts of the media which do not lie on such a path, we have to use a *multiple scattering* technique. Such techniques are more physically accurate as they assume multiple scattering events can occur along a sampling path. There are a few techniques to approximate multiple scattering in volume rendering.

One such rendering scheme is volumetric photon mapping, first posed by Jensen and Christensen [1]. More recent work in photon mapping involves a change to the photon map where the radiance is estimated along a beam, as opposed to a point [2, 3].

Instant radiosity [4] is another means of approximating multiple scattering. In brief, rays are drawn from the primary light source to find the point of first reflection, where a virtual point light

(VPL) is placed. The scene is then rendered with each contribution of every light added together to produce a final image. This technique is similar to another method often used in production where lights are placed on the interior of participating media to produce a glow characteristic of multiple scattering methods, while still only using a single scatter renderer.

While these techniques do produce nice images, there has not been a multiple scattering solution which seeks to be as physically accurate as currently possible. This void motivated the formulation of the radiative transfer equation in terms of Feynman path integrals [5]. One can then pose the volume rendering problem in these terms, but the solution will require the numerical integration of a path integral. In summary, this requires the generation of possibly hundreds of millions of paths, a weight computation, summation of the weights, and showing that the summation converges.

The Feynman path integral was introduced as an unconventional alternative to compute the probability amplitude of a quantum mechanical particle instead of solving a wave equation. The path integral formulates this probability amplitude as a sum of the contributions of all paths a quantum mechanical particle could have taken. Feynman and Hibbs offer an introduction to path integrals in [6].

Path integrals have wide applications and are useful for solving problems in physics, engineering, and chemistry. The use of Monte Carlo methods with path integrals is popular [7, 8]. Generally one can find many instances of these so-called "path integral Monte Carlo" methods in the domain of physics. A large portion of this type of method involves selecting random variates over a distribution of paths, hence the Monte Carlo element of the computation. Often, this is not enough to achieve convergence in a reasonable amount of time, so many involve the use of the Metropolis algorithm [9].

Using the Metropolis algorithm phrases the problem of path generation as a Markov process where the next path drawn from the distribution is a function of the previously accepted path. In many cases for paths, this is a less expensive operation compared to selecting a totally random path, which would often involve a costly random search depending on the application. A general overview of the benefits and caveats of using the Metropolis algorithm is given in [10]. An overview of Monte Carlo methods and the Metropolis algorithm as it relates to path integrals is given in [11].

This article is a continuation of the research described in [5] in which the numerical computation of the path integral is described using a novel framework involving Frenet-Serret curves. We present new developments in a theoretical speedup for the described sampling algorithm, new analysis done on the weighting kernel relating to using the Metropolis algorithm, and finally a tool developed for the analysis of path data emitted by the sampling algorithm.

The rest of this article is organized as follows. Section 2 outlines the basic operation of the sampling algorithm in a precise formulation and describes the steps necessary for an $O(1)$ speedup per iteration in the root solver step. Section 3 describes the weighting kernel and presents analysis of its convergence and a direction for applying the Metropolis algorithm. Section 4 introduces a new general tool developed for analyzing attributed path data like that generated by the sampling algorithm. Finally, Section 5 gives concluding remarks.

2 ACCELERATED PATH GENERATION

We refer to previous work which introduces a discrete Frenet-Serret curve formulation and an algorithm for perturbing Frenet-Serret curves [5]. This section presents a precise formulation of the algorithm and an optimization with analysis. We start by defining the points along the Frenet-Serret curve (Eq. 1) and the orthonormal frames at each of those points (Eq. 2):

$$\vec{x}_i = \Delta s \hat{\mathbf{T}}_i + \vec{x}_{i-1} \tag{1}$$
$$F_i = U_i F_{i-1}. \tag{2}$$

We take F_i to be a matrix consisting of the tangent (written as $\hat{\mathbf{T}}_i$), normal, and binormal vectors written as the rows of the matrix. F_0 and $\vec{x}_0$ are initial orientation and position. We take M to be the number of points defined along the curve. κ_i and τ_i are respectively the curvature and torsion at the i^{th} discrete Frenet-Serret segment. Δs is the step length of the curve. The U_i matrices describe an incremental transformation between orthonormal bases along the length of the curve. An expression in terms of κ_i, τ_i, and Δs is given in [5]. The algorithm is given as

1. Choose three random indices $0 \leq i_0 < i_1 < i_2 < M$.
2. Let $\kappa_{i_1} \in [a, b]$ such that $0 \leq a < b$.
3. Solve a five-dimensional equation
$$G(Y) = 0 \tag{3}$$
where
$$Y = \begin{bmatrix} \kappa_{i_0} \\ \kappa_{i_2} \\ \tau_{i_0} \\ \tau_{i_1} \\ \tau_{i_2} \end{bmatrix} \qquad G(Y) = \begin{bmatrix} x_M - x'_M \\ y_M - y'_M \\ z_M - z'_M \\ \|\hat{\mathbf{T}}_M - \hat{\mathbf{T}}'_M\|_1 \\ \|\hat{\mathbf{T}}_M - \hat{\mathbf{T}}'_M\|_2^2 \end{bmatrix}. \tag{4}$$
In this case, x_M, y_M, and z_M are the original, unperturbed components of the endpoint coordinate of the curve; x'_M, y'_M, and z'_M are the perturbed components; and $\hat{\mathbf{T}}'_M$ is the perturbed end tangent vector. We use $\|\cdot\|_i$ as notation for the i^{th} norm. G is a $\mathbb{R}^5 \mapsto \mathbb{R}^5$ function optimized when the curve is valid.
4. If the optimization failed to converge, restore the curve to its original state.
A failure to converge can occur for many reasons, but often it is simply that the choice of indices or κ_{i_1} made it difficult for the root solver to resolve a solution.
5. Optionally reject the curve according to a Metropolis sampling algorithm.

A typical root solver might look something like Algorithm 1. The $solve()$ routine needs to calculate the endpoint and tangent of the input curve. A first instinct might be to use Eq. 1 and Eq. 2 explicitly, which takes $O(M)$ matrix multiplications and vector additions.

Algorithm 1 A typical root solver routine.

```
Require: $C = (\vec{x}_0, F_0, K, s)$
Ensure: $endpoint(C) = \vec{x}_{goal} \wedge endtangent(C) = \hat{\mathbf{T}}_{goal}$
  $K' \leftarrow perturb(K)$
  $C' \leftarrow (\vec{x}_0, F_0, K', s)$
  $Y \leftarrow select(K')$
  while $\neg solve(C', Y)$ do
    $Y \leftarrow iterate(Y)$
    $update(C', Y)$
  end while
  if $converged()$ then
    $update(C, Y)$
  end if
```

Let us assume that our root solver needs to evaluate the endpoint and tangent a combined total of $O(P)$ times, therefore our current algorithm runs in $O(MP)$ time. We shall reduce this time to linear in M and P. In order to speed this up, we must figure out how to make $endpoint()$ and $endtangent()$ faster. Let us consider the case of finding the end tangent when only one segment, at index i_0, is being modified. Expanding Eq. 2, we get a chain of matrix multiplications,

$$F_M = \underbrace{U_M U_{M-1} ...}_{A_1} U_{i_0} \underbrace{...U_1 U_0}_{A_2} F_0. \tag{5}$$

Given that matrix multiplication is associative, we can precompute the annotated terms into A_1 and A_2 to save time. The idea easily extends to multiple points of perturbation:

$$F_M = \underbrace{U_M U_{M-1} ...}_{A_1} U_{i_2} \underbrace{...}_{A_2} U_{i_1} \underbrace{...}_{A_3} U_{i_0} \underbrace{...U_1 U_0}_{A_4} F_0. \tag{6}$$

For only an $O(M)$ penalty, we can precompute all of these matrices to make $endtangent()$ run in $O(1)$ within a loop iteration. However, $endpoint()$ still takes $O(M)$ time. To speed up $endpoint()$ is only slightly more difficult. At a desired point of perturbation i_0, there is a basis F_{i_0}. When this basis changes, the points $\vec{x}_{i_0+1}...\vec{x}_M$ undergo a rigid rotation according to the change in the underlying basis. Also, notice one can find a straight line vector, or summary vector, of the path by taking the difference $\vec{x}_{i_1} - \vec{x}_{i_0}$, calling it $\vec{v}_1$. Then, we notice by simply taking the sum of $\vec{x}_{i_0} + \Delta s \hat{\mathbf{T}}_{i_0} + \vec{v}_1$, we find the next point of perturbation.

We can find a summary vector between the points of perturbation. It then appears that we could rotate these summary vectors with their respective bases to find the next point of perturbation. Mathematically, this means we project $\vec{v}_1$ into F_{i_0}, $\vec{v}_{1p} = F_{i_0} \cdot \vec{v}_1$, and we remember these projections between iterations of the root solver. This allows us to project the vector out of the modified basis to find the updated position of the next point of perturbation: $\vec{v}_1' = inv(F_{i_0}') \cdot \vec{v}_{1p}$. In summary, we find the following $O(1)$ means of calculating the perturbed end basis and tangent:

$$F_M' = A_4 U_{i_2} A_3 U_{i_1} A_2 U_{i_0} A_1 F_0 \tag{7}$$

$$\begin{aligned} \vec{x}_M' &= \vec{x}_0 + \vec{v}_1 \\ &+ \hat{\mathbf{T}}_{i_0}' \Delta s + inv(F_{i_0}') \cdot \vec{v}_{2p} \\ &+ \hat{\mathbf{T}}_{i_1}' \Delta s + inv(F_{i_1}') \cdot \vec{v}_{3p} \\ &+ \hat{\mathbf{T}}_{i_2}' \Delta s + inv(F_{i_2}') \cdot \vec{v}_{4p}. \end{aligned} \tag{8}$$

Algorithm 2 shows the revised root solver routine. The relevant additions are the $O(M)$ operations of $premultiplyBases()$ and $presumVectors()$, which are implementations of the optimization discussed previously. Given that there are $O(P)$ function evaluations in the root solver, we have reduced the algorithm to being $O(M + P)$ overall. A preliminary estimate of the speedup by measuring the rate at which curves are generated over several randomized trials suggests it is close to an order of magnitude.

Algorithm 2 Revised root solver routine.

```
Require: $C = (\vec{x}_0, F_0, K, s)$
Ensure: $endpoint(C) = \vec{x}_{goal} \wedge endtangent(C) = \hat{\mathbf{T}}_{goal}$
  $K' \leftarrow perturb(K)$
  $C' \leftarrow (\vec{x}_0, F_0, K', s)$
  $A_1, A_2, A_3, A_4 \leftarrow premultiplyBases(C')$
  $\vec{v}_1, \vec{v}_2, \vec{v}_3, \vec{v}_4 \leftarrow presumVectors(C')$
  $Y \leftarrow select(K')$
  while $\neg solve(C', Y)$ do
    $Y \leftarrow iterate(Y)$
    $update(C', Y)$
  end while
  if $converged()$ then
    $update(C, Y)$
  end if
```

An implementation of the described path generation algorithm was created using Python and SciPy [12]. The implementation includes the revised root solving algorithm and caches the computation of the U_i matrices where possible. A large portion of the computation is devoted to a root solving algorithm, and SciPy offers MINPACK's Powell hybrid method (*hybr*), MINPACK's Levenberg-Marquardt (*lm*), and several Newton methods. Among those are Broyden's good method (*broyden1*), Broyden's bad method (*broyden2*), Anderson mixing (*anderson*), Krylov

Method	Time (s)	Curves	CPCH	Evaluations
broyden1	541.92	338.30	2250.93 ± 4.81	-
excitingmixing	1297.75	0.11	0.32 ± 0.37	-
hybr	**66.79**	667.34	**36108.04** ± 877.22	**197476.51**
krylov	840.93	662.57	2855.83 ± 98.04	-
linearmixing	1202.42	0.17	0.51 ± 0.38	-
lm	121.22	**828.17**	24662.43 ± 511.32	382549.86

Table I. Results of the root solver performance benchmark.

approximation (*krylov*), Diagonal Broyden approximation (*diagbroyden*), and others (*linearmixing* and *excitingmixing*).

An experiment was designed to test the efficacy of each of these methods for this particular algorithm. For each method, 35 single core jobs were launched on a supercomputer generating 1000 curves. For these jobs, the time spent generating paths, the number of paths generated, and the number of function evaluations (if available) were recorded. Table I shows the acquired results. The means of the job time and number of curves generated are shown. Curves generated per core-hour (CPCH) is derived by measuring a simple rate (curves generated per unit hour) over each individual job and finding the mean. A confidence interval for CPCH is shown for $\alpha = 0.025$.

Of the methods available, many of them would diverge, suggesting that they are not a good fit for this situation; they are not included in the table. In terms of CPCH, Powell's hybrid method is a clear winner. Levenberg-Marquardt lags Powell's hybrid method in performance, but interestingly seems to converge more often.

3 INTEGRATION OVER PATHS

Once a path is generated we must be able to assign a weight to the path efficiently. In the derivation presented in [5], a weighting function is derived for a single node of a Frenet-Serret curve,

$$\omega_n = \exp(-c_n \Delta s) \int \frac{d^3\mathbf{p}}{(2\pi)^3} \exp(i\mathbf{p} \cdot \hat{\mathbf{N}}_n \kappa_n \Delta s + b_n \Delta s \tilde{Z}(|\mathbf{p}|)), \tag{9}$$

where the weight of the entire path is just the product over the nodes. In fact, an error was discovered in the original paper and several of the derived equations after being written in spherical coordinates are incorrect. Rewriting in spherical coordinates and taking $\mathbf{p} \cdot \hat{\mathbf{N}}_n = p \cos\theta$ yields

$$\omega_n = \exp(-c_n \Delta s) \int_0^\infty \frac{p}{2\pi^2} \exp(b_n \Delta s \tilde{Z}(p)) \frac{\sin(p\kappa_n \Delta s)}{\kappa_n \Delta s}\, dp. \tag{10}$$

We focus on the regularized form,

$$\omega_n = \exp(-c_n \Delta s) \int_0^\infty \frac{p}{2\pi^2} \exp(b_n \Delta s \tilde{Z}(p) - \frac{\epsilon^2}{2} p^2) \frac{\sin(p\kappa_n \Delta s)}{\kappa_n \Delta s}\, dp. \tag{11}$$

Many of the terms are parameters of the path discussed in Section 2. The terms are:

- Δs and κ_n which are the step size and the curvature of the Frenet-Serret curve,
- a_n, b_n and c_n, which are respectively absorption, backscatter, and $a_n + b_n$ at position n,
- $\tilde{Z}(p)$, the Fourier transform of the phase function,
- ϵ, a tunable parameter to make the integral more numerically feasible.

While a_n and b_n can be any scalar field, here we focus on the case where they are constants. Smaller values of ϵ approach the theoretical value of the integral, but are also more sensitive to numerical noise. For $\tilde{Z}(p)$, we use the transformed Gaussian phase function presented in [13],

$$\tilde{Z}(p) = N_p \exp\left(\frac{-\mu p^2}{2}\right), \tag{12}$$

which is derived from the phase function,

$$P(\hat{\mathbf{n}}, \hat{\mathbf{n}}') = \frac{2N_p}{(2\pi\mu)^{3/2}} \exp\left(\frac{\hat{\mathbf{n}} \cdot \hat{\mathbf{n}}' - 1}{\mu}\right), \tag{13}$$

with the normalization constant

$$N_p = \frac{\sqrt{\pi\mu/2}}{1 - \exp(-2/\mu)}. \tag{14}$$

Cutting the integration bounds off at $10/\epsilon$ is sufficient for capturing the important parts of the integrand while avoiding numerical noise in the result. The bulk of the contribution is well within the bounds of integration, and anything beyond will likely only contribute numerical noise.

It is also helpful to understand how modifying the parameters of the integral affects the overall character of the weight function. Fig. 1 shows increasing μ widens the Gaussian phase function and therefore the weight function tails off over a longer period of time. Fig. 2 shows that ϵ corresponds

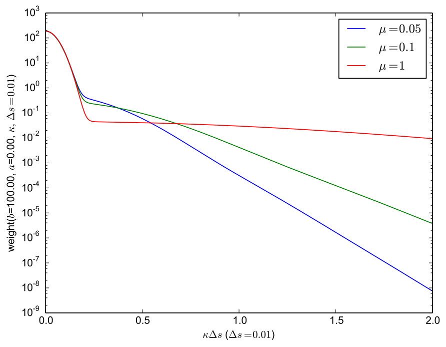


Figure 1. Weight function with varying μ.

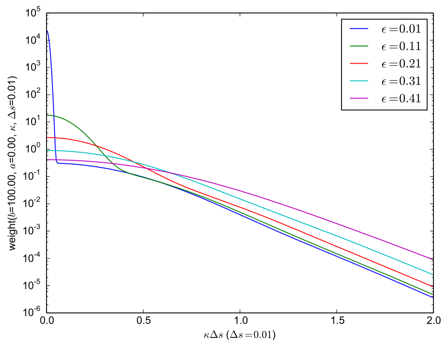


Figure 2. Weight function with varying ϵ.

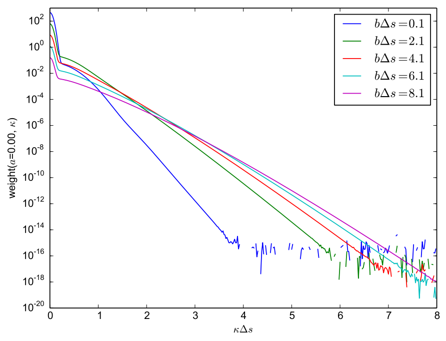


Figure 3. Weight function with varying $b\Delta s$.

to the width of the peak of the delta function and increases the maxima of the function. Fig. 3 shows increasing $b\Delta s$ causes the weight in general to decrease at a slower rate, though we are not typically interested in large values of $b\Delta s$. This plot also shows that the smoothness of the weight curve does tend to break down at a particular shelf.

It is expensive to evaluate the weight integral numerically, so an implementation would benefit from caching. The primary parameters in the weights are $b\Delta s$ and $\kappa\Delta s$. The parameters μ and ϵ should be frozen throughout a trial. We cache on $b\Delta s$ and $\kappa\Delta s$ as the function is smooth enough to be interpolated without loss of much accuracy.

3.1 Integration

A path's weight is $\Omega = \prod_i^M \omega_i$, where ω_i is the weight contributed by a single Frenet-Serret segment, defined by Eq. 11. We then apply Monte Carlo integration of the sequence of path weights generated, Ω_i, given as

$$F(N) = D\frac{1}{N}\sum_{i=1}^{N}\Omega_i \tag{15}$$

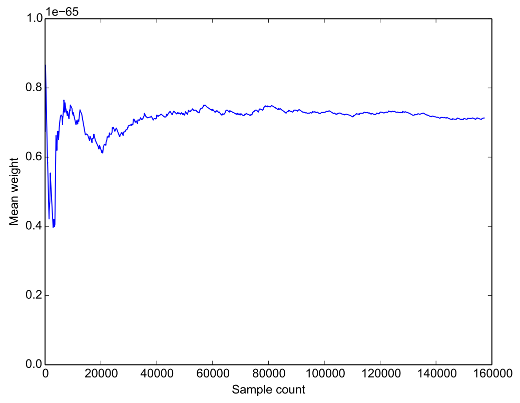


Figure 4. A plot of $1/N\sum_i^N \Omega_i$ for increasing N.

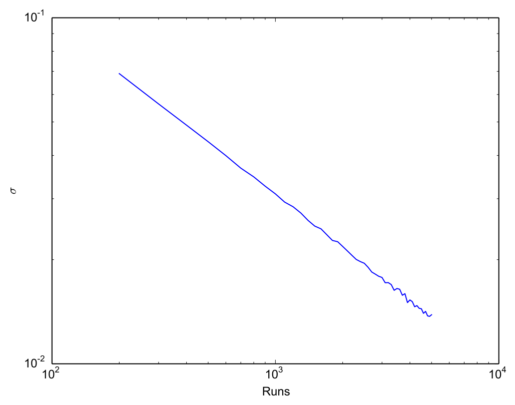


Figure 5. Log-normal parameter σ for increasing run samples.

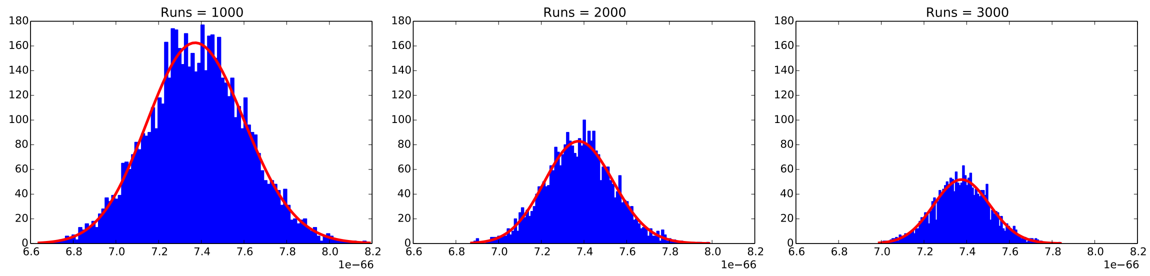
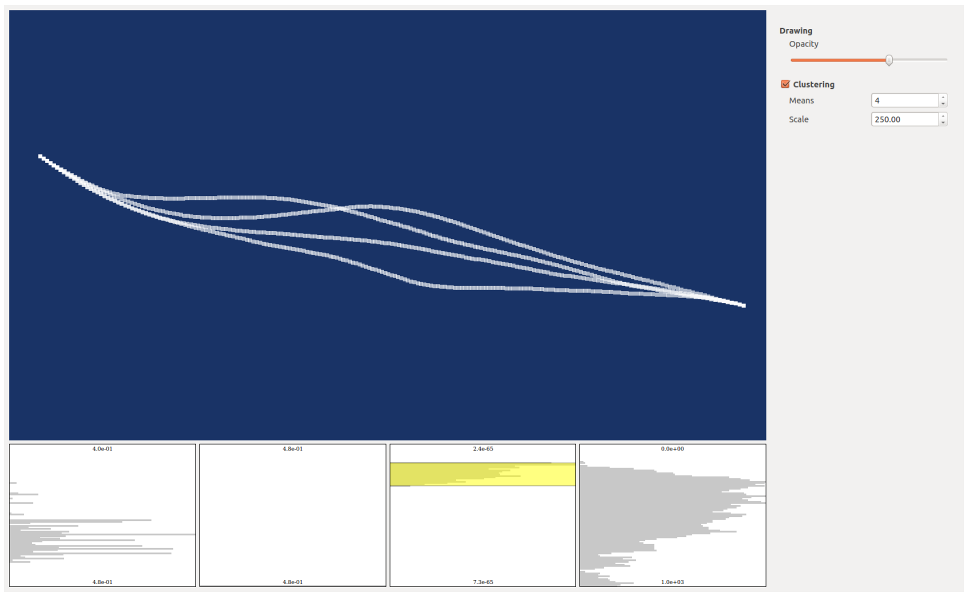


Figure 6. Log-normal fit with increasing run samples.

where D is a normalizing scalar factor. If we take $D = 1$ we can begin to get an idea of how large N will need to be in order for the integration to converge, and $F(N)$ produces a running average of the sequence Ω_i. Fig. 4 demonstrates the running average of the weighting function for increasing values of N when using a fixed input curve. The apparent convergence occurs after only tens of thousands of generated paths.

The homogeneity of the mean is important for this problem. Fig. 6 illustrates the character of convergence if we sum over slices of varying size across the whole data set and create a histogram of the result. What we find is that the means of the histograms line up with the convergence predicted by Fig. 4. A fit of the data places the mean of the log-normal distribution roughly in the same place. This suggests that if we choose any set of randomly generated curves and sum their weights, then they should roughly converge to the same mean.

The spread of the distribution is also of interest since it will gauge how close the calculation is to convergence. Fig. 5 shows the trend of the σ parameter for the resulting log-normal fits against increasing slice size. We see a decreasing trend as we might expect, but this offers insight to the accuracy of the prediction of the mean as a function of the slice size. For instance, there is an obvious difference between a slice size of 500 and 1000, but the difference is subtle between 2000 and 3000.



[✓ Clustering]
Means 4
Scale 250.00

Figure 7. A screenshot of CuE's user interface.

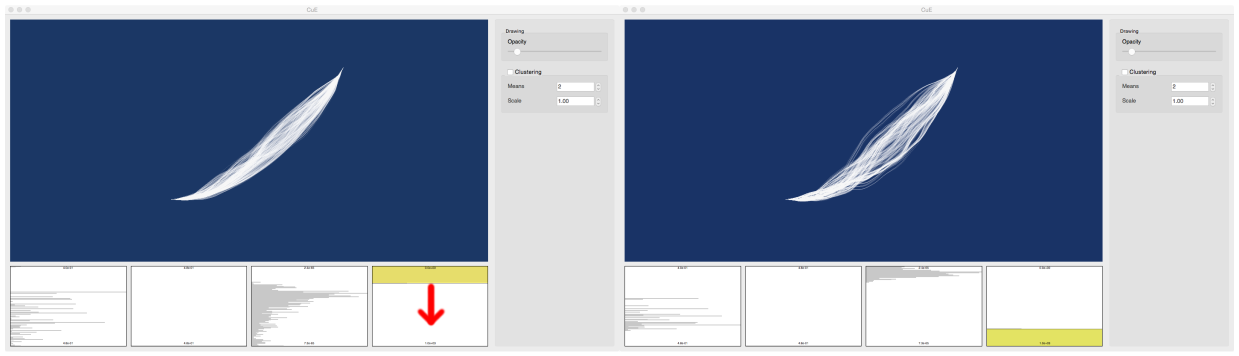
This fit is useful for the Metropolis algorithm. Let C_i be a given curve in the sampling sequence and Ω_i its weight; likewise C_c is a candidate curve and Ω_c its weight. Using the fit data we obtain a log-normal probability density function $p(x)$, and compute an acceptance probability $\alpha = min(1, p(\Omega_c)/p(\Omega_i))$. We let C_{i+1} be C_c with probability α, or C_i otherwise.

4 VISUALIZATION

The need to analyze the behavior of the developed sampling algorithm led to the development of a tool for the exploratory analysis of *attributed path data*. We use this as an abstract term to refer to data which are primarily a collection of paths, each of which may have a collection of numeric data values associated with it. The data produced by the algorithm are the three-dimensional point data of the curve itself, a weight, and a sequence number. A previous attempt to visualize the data was a non-interactive, on-line display of generated curves. A new tool named Curve Explorer, or CuE for short, was developed to explore a large batch of sampled curves off-line.

Fig. 7 shows a screenshot of CuE's user interface. The primary features are a viewport with full camera control, a set of widgets which allow the user to filter on the attributes of the path set, and controls to modify the display mode. The default view shows a simple plot of the path data in three-dimensional space. The user can use the filter widgets on the bottom of the screen to specify ranges to filter on. The filters can be moved or dismissed and the view updates accordingly. The filters show a histogram plot of the attributes which updates whenever the filter is adjusted.

The user can also adjust properties of the display of paths. To aid with over-plotting, the opacity of the graphics primitives can be adjusted. This allows the user to achieve a view of all of the path data to taste. The default view of the path data is a simple line plot. The user is also able to cluster the path data. In this mode, paths of M vertices are treated as $M \times 3$ vectors which are clustered



Means 2
Scale 1.00

Means 2
Scale 1.00

Figure 8. A possible interaction in CuE. Set a filter and sweep over the sequence number (rightmost box) and observe the evolution of path weights (second from right).

using k-means clustering. The cluster centroids are then plotted as a sequence of square sprites. The global error of the clustering operation is encoded into the size of the sprites.

These interactions support a few different useful tasks for analyzing sampled path data. For example, the user can turn on 1 or 2-means clustering, specify a thin filter over the sequence number and visually skim for a region where the sprites are largest, indicating a high amount of variance in that region. Also, the updating histograms are useful for skimming over the sequence number seeing where the paths have least contribution (Fig. 8).

5 CONCLUSION

This paper presents continuing research on the Monte Carlo integration of a formulation of the radiative transfer problem in terms of Feynman path integrals. We present a new means for optimizing the root solver to $O(1)$ per iteration. New analysis of the weighting kernel is shown. We find that convergence can be fit with a log-normal distribution and present a direction for applying Metropolis sampling. We also show a tool developed in the scope of this problem for the interactive analysis of attributed path data, or data sets which are primarily paths in nature each with associated numeric values.

Currently our sampling algorithm requires an initial input path. Future work will focus on developing candidate algorithms for generating arbitrary initial paths which satisfy the desired constraints. Additionally, analysis should be conducted on the effectiveness of using the Metropolis algorithm as a variance reduction technique for this problem.

6 ACKNOWLEDGMENTS

One author (PK) thanks Joshua A. Levine whose instruction made the development of CuE possible and for the clever idea of using k-means clustering to analyze path data.

7 REFERENCES

[1] H. W. Jensen and P. H. Christensen, “Efficient Simulation of Light Transport in Scenes with Participating Media Using Photon Maps,” *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’98, pp. 311–320, New York, NY, USA, 1998, ACM.

[2] W. Jarosz, D. Nowrouzezahrai, I. Sadeghi, and H. W. Jensen, “A Comprehensive Theory of Volumetric Radiance Estimation Using Photon Points and Beams,” *ACM Transactions on Graphics*, **30**, *1*, pp. 5:1 – 5:19 (2011).

[3] J. Novák, D. Nowrouzezahrai, C. Dachsbacher, and W. Jarosz, “Virtual Ray Lights for Rendering Scenes with Participating Media,” *ACM Transactions on Graphics*, **31**, *4*, pp. 60:1–60:11 (2012).

[4] A. Keller, “Instant Radiosity,” *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’97, pp. 49–56, New York, NY, USA, 1997, ACM Press/Addison-Wesley Publishing Co.

[5] J. Tessendorf, “Numerical Integration of the Feynman Path Integral for Radiative Transport,” *International Conference on Mathematics, Computational Methods and Reactor Physics*, Saratoga Springs, NY, May, 2009.

[6] R. P. Feynman and A. R. Hibbs, *Quantum Mechanics and Path Integrals*, Dover Publications, Mineola, New York, NY (2010).

[7] T. F. Miller and D. C. Clary, “Torsional Path Integral Monte Carlo Method for Calculating the Absolute Quantum Free Energy of Large Molecules,” *The Journal of Chemical Physics*, **119**, *1*, pp. 68–76 (2003).

[8] C. P. Herrero and R. Ramírez, “Structural and Thermodynamic Properties of Diamond: A Path-Integral Monte Carlo Study,” *Physical Review B*, **63**, *2*, pp. 024103 (2000).

[9] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, “Equation of state calculations by fast computing machines,” *The Journal of Chemical Physics*, **21**, *6*, pp. 1087–1092 (1953).

[10] M. H. Kalos and P. A. Whitlock, *Monte Carlo Methods*, volume I, John Wiley & Sons, Ltd. (1986).

[11] D. C. Khandekar, S. V. Lawande, and K. V. Bhagwat, *Path-Integral Methods and their Applications*, World Scientific Publishing Co. Pte. Ltd., P O Box 128, Farrer Road, Singapore 912805 (1993).

[12] E. Jones et al., “SciPy: Open source scientific tools for Python,” 2001–.

[13] J. Tessendorf, “Angular Smoothing and Spatial Diffusion from the Feynman Path Integral Representation of Radiative Transfer,” *Journal of Quantitative Spectroscopy and Radiative Transfer*, **112**, *4*, pp. 751 – 760 (2011).