Pacific Graphics (2013)
B. Levy, X. Tong, and K. Yin (Editors)

Fast Random Sampling of Triangular Meshes

M. Šik[1] and J. Křivánek[1]

[1]Charles University in Prague, Czech Republic

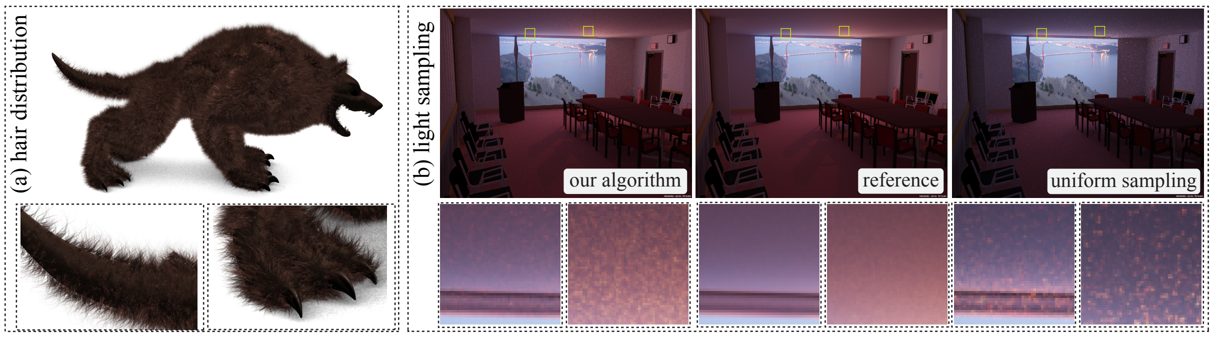

Figure 1: Our fast mesh sampling algorithm can place up to 78 million random sample points per second on a triangle mesh. We can use the algorithm, for example, to interactively distribute hair roots on a surface (a) or for sampling illumination from a complex luminaire, such as a projected HDR image, where uniform sampling produces a noisy image (b).

Abstract

We present a simple and fast algorithm for generating randomly distributed points on a triangle mesh with probability density specified by a two-dimensional texture. Efficiency is achieved by resampling the density texture on an adaptively subdivided version of the input mesh. This allows us to generate the samples up to 40× faster than the rejection sampling algorithm, the fastest existing alternative. We demonstrate the algorithm in two applications: fast placement of hair roots on a surface and sampling of illumination from a complex luminaire. Part of our mesh sampling procedure is a new general acceleration technique for drawing samples from a 1D discrete probability distribution whose utility extends beyond the mesh sampling problem.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Surfaces and object representations I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

Surface sampling has a number of uses in computer graphics including remeshing, point-based graphics, and texturing. In this paper, we consider applications such as Monte Carlo rendering that require the generated samples to be random and follow a specified probability density. Previous work on this problem focuses on the quality of point distributions [BWWM10, CCS12]. However, such approaches are limited in performance, especially for non-uniform densities, which precludes their use in interactive applications and in efficient Monte Carlo rendering.

We propose a simple and fast algorithm for generating randomly distributed points on a triangular mesh with probability density specified by a two-dimensional image texture mapped on the surface. Efficiency is achieved by resampling the density texture on an adaptively subdivided input mesh. Doing so allows us to achieve a speedup of up to 40 times compared to the rejection sampling algorithm, the fastest existing alternative. Since efficiency is our primary concern, we do not ensure regularity of the generated samples by stratification or by enforcing the Poisson-disk property. We demonstrate our algorithm in two applications: fast placement of hair roots on a surface and sampling of illumination from a complex luminaire in Monte Carlo rendering (Fig. 1).

Important for the efficiency of our approach is a new general acceleration technique for generating samples from a 1D discrete probability distribution that uses a lookup table to speed up the search in the cumulative distribution function (CDF). This technique achieves a $2\times$ to $14\times$ speedup over the traditional approach based on a binary search in the CDF [PH10]. Thanks to its generality, the utility of this technique goes beyond mesh sampling, as we demonstrate on the problem of sampling illumination from a high-resolution high dynamic range (HDR) environment map.

2. Related Work

A number of papers have addressed generation of point distributions (see [LD08, EMP*12] for an overview), however, relatively little research has focused on random sampling of surfaces embedded in 3D space. Surface sampling has been studied in different contexts such as remeshing [Tur92, QM06], point-based graphics [GP07], texturing [LD05], realistic rendering [JB02], or non-photorealistic rendering [Mei96]. The goal of these techniques, however, is not to generate unbiased samples from a prescribed probability density. In addition, their performance is usually limited.

Several recent papers have addressed surface sampling with stratification [NS04] or blue-noise properties [BWWM10, CCS12]. However, the quality of the resulting distribution comes at the cost of relatively low sampling performance. This is an important limitation in applications where sampling speed is critical, such as sampling of complex luminaires in Monte-Carlo rendering or interactive placement of hair roots. Another important limitation of these methods is that they generate uniform samples without the possibility of controlling the sample density.

Fast unbiased sampling of meshes is discussed in few papers [OFCD02, CCS12] but these techniques are limited to uniform distributions. Several existing methods [LRR05, SSKKC10] can be used for fast sampling of non-uniform distributions, however they can not be straightforwardly expanded to mesh sampling. A widely known approach to generalize such techniques to generating samples with any given density is rejection sampling [PH10, p. 671]. Yet, its performance suffers greatly for highly varying densities . Our algorithm overcomes this problem and consistently outperforms any of the existing alternatives by a large margin, however in its current state is unable to ensure high-quality distribution of samples (such as blue-noise distribution) and therefore can not be used for some applications such as mesh reconstruction or NPR.

3. Mesh Sampling

In this section we give a detailed description of our new mesh sampling algorithm.

3.1. Problem Definition

Given a set of n triangles $\{T_i\}_{i=1}^n$, $T_i \subset \mathbb{R}^3$, $\mathcal{T} = \cup_{i=1}^n T_i$, a density function $f : [0,1]^2 \to \mathbb{R}_0^+$ represented by an image texture, and a mapping $m : \mathcal{T} \to [0,1]^2$ that maps points on $\mathcal{T}$ onto the texture domain, we want to draw samples from a distribution with the probability density function (PDF) $p : \mathcal{T} \to \mathbb{R}_0^+$ (w.r.t. the surface area measure) given by:

$$p(\mathbf{x}) = \frac{f(m(\mathbf{x}))}{\int_{\mathcal{T}} f(m(\mathbf{x}'))\mathrm{d}A(\mathbf{x}')} \tag{1}$$

3.2. Possible Approaches

A straightforward solution to the above problem is to use *rejection sampling*:
1) Pick a triangle T_i proportionately to its surface area,
2) Propose a sample $\mathbf{x}$ from a uniform distribution on T_i,
3) Generate a random number ξ from uniform distribution $U(0,M)$, with $M = \sup\{f(\mathbf{u}) \mid \mathbf{u} \in [0,1]^2\}$,
4) Accept $\mathbf{x}$ if $\xi < f(m(\mathbf{x}))$, otherwise go to step 1.
This approach suffers from the usual disadvantages of rejection sampling. Efficiency degrades rapidly for non-uniform density f, and the number of random numbers used to generate one sample cannot be bounded prior to the calculation.

Another alternative would be to carry out the sampling in the $[0,1]^2$ texture space, where the density function f is defined by an image that can be efficiently sampled [PH10, p. 671], and then transform the samples to the triangles using the inverse of the m mapping. However, generality of this approach is compromised by the fact that m is often not invertible (e.g. for a tiled texture) and usually not area-preserving.

3.3. Algorithm Overview

Our approach does not suffer from the performance and generality issues of the aforementioned alternatives. Consider a simple case, where the density texture f is constant over the area of any one triangle. Our sampling problem would then reduce to choosing a triangle from a suitable discrete distribution and drawing the sample from a uniform distribution on the triangle. The main idea of our approach is to map the general problem to this simplified case by subdividing the triangles of the input mesh until the density texture f can be considered constant over the sub-triangle area, and resampling the density texture on the subdivided triangles. Doing so avoids problems due to the incompatibility of the density texture and triangle mesh domains, and greatly simplifies the sampling algorithm. This idea is similar to the Mesh Colors approach for texturing 3D meshes [YKH10], though the purpose is different.

Our algorithm works in two stages. The preprocessing stage, executed once for a given mesh and density texture, subdivides the input triangles and creates a piece-wise constant PDF on the sub-triangles that can be efficiently sampled. The sampling stage then draws samples from this PDF. In order to make the approach practical, we propose a memory-efficient representation for the subdivided triangles

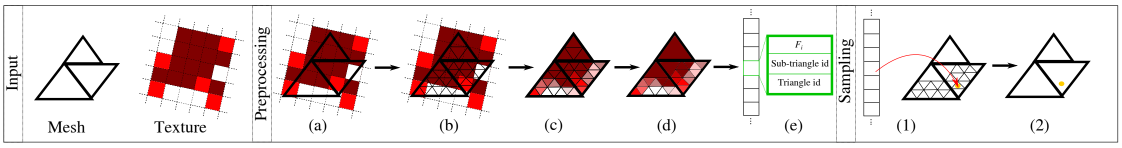

Figure 2: In the preprocess, we subdivide each triangle (a) until the sub-triangle size matches the density texture resolution (b), and subsequently compute a sub-triangle probability P_i by resampling the density texture on the subdivided triangle (c). To speed up sampling and reduce memory cost, we merge sub-triangles with same sampling probabilities (d). Finally, we compute a cumulative distribution function (CDF) F as $F_i = \sum_{j=1}^{i} P_j$ and for each sub-triangle i store F_i, sub-triangle index and parent triangle index (e). In the sampling stage, we randomly select a sub-triangle using the computed CDF F and place a sample on it (1). We use the stored sub-triangle index to calculate sample location at the parent triangle (2).

and an accelerated procedure for generating surface samples from this representation. The rest of this section and Section 4 describe the technical details of our solution. For full algorithm overview see Fig. 2.

3.4. Stage 1: Preprocessing

Pseudocode of the preprocessing stage is given in Fig. 3. In this stage, we subdivide the input triangles and effectively replace the desired sampling PDF (Eq. 1) by its approximation p' that is constant over the area of each sub-triangle. The PDF value for sub-triangle D_i is determined by taking a (bilinearly filtered) sample f_i of the density texture at the sub-triangle's barycenter. For $\mathbf{x} \in D_i$, the approximate PDF is defined as

$$p'(\mathbf{x}) = \frac{f_i}{\sum_{j=1}^{n_s} f_j\,|D_j|}, \qquad (2)$$

where n_s is the total number of sub-triangles and $|D_j|$ denotes the surface area of sub-triangle D_j.

```
For each mesh triangle:
1. Determine subdivision level for the triangle.
2. Subdivide the triangle up to the determined level.
3. Compute sampling probability $P$ for each sub-triangle.
4. Merge sub-triangles with same sampling probabilities.
5. For each sub-triangle, store probability $P$, parent triangle
   index, and sub-triangle index.
```

Figure 3: The preprocessing stage of our algorithm.

To make the above PDF approximation accurate, our goal is to subdivide the input triangles until the density texture can be safely considered constant inside each sub-triangle. To achieve this goal, each triangle is subdivided so that its sub-triangles are smaller than a texel of the density texture mapped on the triangle. Because the m mapping is affine over each input triangle (as per linearly interpolated per-vertex texture coordinates), the subdivision depth can be determined directly from the number of texels mapped on the triangle. Subsequently, we recursively replace sub-triangles with the same PDF value by their parent, so that the total number of resulting sub-triangles is minimized.

Sampling of the piecewise constant PDF p' involves picking a sub-triangle with probability $P_i = \int_{D_i} p'(\mathbf{x})\mathrm{d}A(\mathbf{x}) = f_i\,|D_i| / \sum_{j=1}^{n_s} f_j\,|D_j|$, generating a sample in the sub-triangle, and mapping it to the original mesh triangle. For this purpose, the preprocessing stage calculates and stores the cumulative distribution function (CDF) $F_i = \sum_{j=1}^{i} P_j$, and, for each sub-triangle, a reference to the parent input triangle, as well as the barycentric coordinates of the sub-triangle inside the parent triangle. We take advantage of the fact that the subdivision scheme is the same for all triangles, so a unique sub-triangle index is sufficient to determine its barycentric coordinates (see Fig. 4). A separate pre-computed table accessed by the sub-triangle index stores the actual sub-triangle barycentric coordinates. This significantly reduces memory consumption, since for each sub-triangle we only need to store its CDF value F_i and two indices (parent triangle index and sub-triangle index). The size of the precomputed data structure which stores sub-triangle barycentric coordinates is negligible.

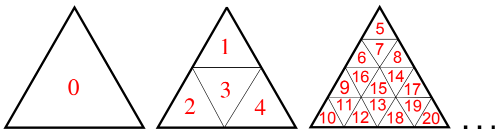

Figure 4: Index of a sub-triangle uniquely determines its position inside the parent triangle.

3.5. Stage 2: Generating Point Samples

Given the data computed in the preprocessing stage, generating a random sample is straightforward. First, we choose a sub-triangle D from the precomputed probability distribution (Section 4 provides details). We then draw a sample from a uniform distribution on the selected sub-triangle using a standard approach [PH10, p. 670] as follows: Given two random numbers ξ_1 and ξ_2 from the uniform distribution $U(0,1)$, the barycentric coordinates of the generated sample are $u_D = 1 - \sqrt{\xi_1}$ and $v_D = \xi_2 \cdot \sqrt{\xi_1}$. Finally, we map the sample to the barycentric coordinates (u,v) w.r.t. the parent mesh triangle:

$$\begin{aligned} u &= u_D \cdot u_1 + v_D \cdot u_2 + (1 - u_D - v_D) \cdot u_3 \\ v &= u_D \cdot v_1 + v_D \cdot v_2 + (1 - u_D - v_D) \cdot v_3 \end{aligned}$$

where u_i and v_i are the barycentric coordinates of the vertices of sub-triangle D inside its parent mesh triangle. We obtain

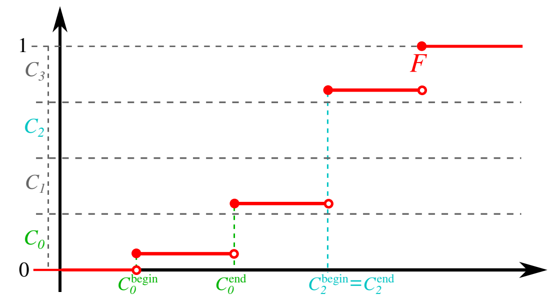

Figure 5: A lookup table T built over the codomain of the cumulative distribution function F.

u_i and v_i by a lookup in the pre-computed sub-triangle position data structure using the sub-triangle index.

4. Fast Sampling from a Discrete 1D Distribution

The first step of the procedure that generates a sample on the mesh involves drawing a sub-triangle from a 1D discrete distribution given by the CDF $F = [F_1, \ldots F_{n_s}]$. The standard way to implement this is to use interval bisection to find sub-triangle $D_i = \arg\min_i \{F_i > \xi\}$, where ξ is a random number from $U(0,1)$ [PH10, p. 647]. Thanks to its logarithmic running time, this bisection algorithm is usually considered efficient for practical purposes. However, the number of sub-triangles can be large in our case and the logarithmic search for generating each sample may incur a significant overhead.

We substantially improve the efficiency of sampling from a 1D discrete distribution by means of a lookup table T over the codomain of F (i.e. the $[0,1]$ interval) created in the preprocessing stage (a similar method can be found in [CRW09]). For each table cell $C_k = [C_k^{\min}, C_k^{\max}]$ we compute two indices C_k^{begin} and C_k^{end}, as illustrated in Fig. 5.

$$\begin{aligned} C_k^{\text{begin}} &= \arg\min_i \{F_i \geq C_k^{\min}\} \\ C_k^{\text{end}} &= \arg\min_i \{F_i \geq C_k^{\max}\} \end{aligned}$$

When drawing an element from the distribution (sub-triangle in our case), we first generate a random number ξ from $U(0,1)$ as before, then we look up, in constant time, the table cell C_k for which $\xi \in C_k$, and finally we find the element using the bisection algorithm limited to the domain $[C_k^{\text{begin}}, C_k^{\text{end}}]$. Since $C_k^{\text{end}} = C_{k+1}^{\text{begin}}$ for every k, we can store only one index per each cell.

Both the table resolution and the characteristic of the probability distribution influence the performance of this approach. In our tests, the speedup due to our table-based search compared to the usual bisection varied between 2 and 14. Please note that this table-based search is a general procedure for accelerated sampling from a 1D discrete distribution and may be used in other applications, such as sampling HDR environment maps as shown in the results section.

5. Results

We test our sampling algorithm on 4 cores with 8 threads of a 3.07 GHz Intel Core i7-950 PC with 6 GB RAM running Windows 7 64bit. We use density textures with a resolution of 1024×1024 unless noted otherwise. To make full use of the four available CPU cores, both the triangle subdivision and sampling are parallelized. Fig. 6 shows sample distributions generated by our algorithm for the three example models we use in our tests.

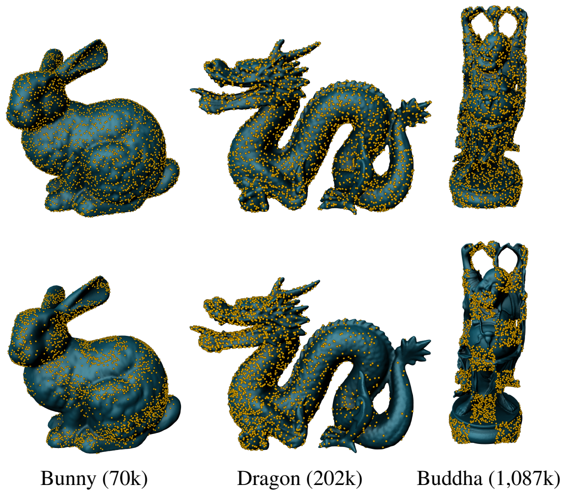

Figure 6: Three different models with 7,000 point samples distributed uniformly (top row) and according to simple density textures (bottom row). Triangle counts for each model are listed in parentheses.

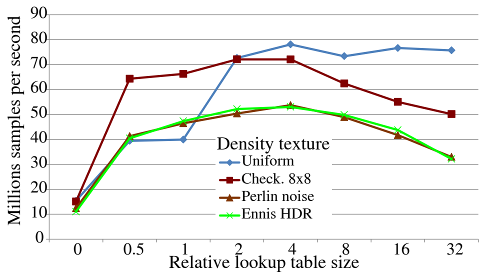

Figure 7: Sampling rate of our algorithm for different lookup table sizes (reported as a ratio of the CDF size).

5.1. Lookup Table Performance

We start the evaluation by investigating the influence of the lookup table size on sampling performance as shown in Fig. 7. We have used the Bunny model and four different density textures for this test: uniform, checkerboard, Perlin noise, and a HDR environment map. The highest sampling rate is achieved when the number of cells is about four times higher than the CDF domain size (i.e. the number of sub-triangles). We use this setting for generating the results in the remainder of this section.

As an entirely general technique, the table-based CDF search that we use to speed up our mesh sampling algorithm can be used for any other applications that require sampling from a discrete probability distribution. As an exam-

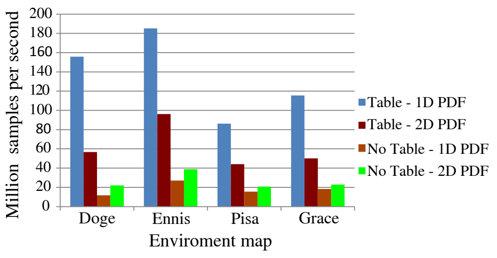

Figure 8: Performance of the environment map sampling with and without our table-based CDF search.

(a) Ennis (6144 × 3072) (b) Grace (3072 × 1536) (c) Pisa (4096 × 2048) (d) Doge (4096 × 2048)

Figure 9: HDR environment maps used in our tests. The resolution of each environment map is listed in parentheses.

ple, Fig. 8 compares sampling from the environment maps shown in Fig. 9 with and without the lookup table. In the tested cases the use of the table led to a speedup between 5 to 14 when considering the environment map as a 1D probability distribution, and between 2 to 3 for a 2D distribution.

5.2. Sampling Performance and Memory Consumption

We compare the performance and memory consumption of our mesh sampling algorithm to rejection sampling (as described in Section 3.2) because it is the fastest existing alternative. We consider two flavors of our algorithm – with and without the lookup table optimization described in Section 4 – and perform the test for a number of different density textures: uniform, checkerboard and 2-dimensional Perlin noise textures with different frequencies, and two HDR environment maps from Fig. 9 converted to a 1024×1024 resolution.

Fig. 10 plots the performance of the compared algorithms. With a uniform density texture, rejection sampling never rejects any samples, therefore the faster sampling rate of our algorithm is only due to the lookup table. Performance of rejection sampling significantly deteriorates for non-uniform density textures (up to 40 times for the tested HDR textures and would be even worse for a texture with greater dynamic range), while our algorithm's sampling rate drops at most by 35% compared to the sampling rate with a uniform texture.

Break-down of the time our algorithm spends on sampling reveals that $2.5\% - 51.2\%$ is spent on selecting a sub-triangle by searching the CDF, while random number generation takes $16.8\% - 33.5\%$, and the rest (i.e. generating a sample position in a sub-triangle and transforming it to the parent triangle) takes $32\% - 64\%$ of the time. The fact that substantial fraction of the time is spent on random number generation suggests that our algorithm does not leave much space for performance improvement other than micro-optimization.

Preprocess [ms]	Bunny	Dragon	Buddha
Rejection	6	11	67
Our w/out table	$65 - 68$	$67 - 69$	$68 - 71$
Our w/ table	$67 - 70$	$68 - 72$	$71 - 74$
Memory [MB]	Bunny	Dragon	Buddha
Rejection	4.3	4.8	8.1
Our w/out table	$2.8 - 17.8$	$4.3 - 17.8$	$14.4 - 17.9$
Our w/ table	$3.9 - 23.7$	$5.5 - 24.1$	$21 - 25$

Table 1: Preprocessing time in milliseconds (top) and memory consumption in megabytes (bottom) for rejection sampling and our algorithm (w/out and w/ lookup table). Same models and textures as in Fig. 10 were used. The lowest values were measured for the checkerboard 2×2 texture and the highest for the HDR textures.

Table 1 top shows the time spent on the mesh and density texture preprocessing. Rejection sampling has the shortest preprocessing time since it only creates the CDF based on triangle areas. Our algorithm needs to subdivide triangles, calculate their probabilities based on the density texture, create the CDF, and build the lookup table. The table construction time is negligible ($3 - 4\%$) compared to the rest of the preprocessing. Since triangle count for the Buddha model (1,087k) is higher than the texel count of the density texture (1,024k), only few triangles are subdivided and the preprocessing time for our algorithm is only slightly higher than for rejection sampling.

Memory consumption of rejection sampling and our algorithm is shown in Table 1 bottom. Even though our algorithm has higher memory cost in most cases, it is still insignificant with respect to the usual computer's memory size.

In summary, our mesh sampling algorithm offers a speedup between 3 and 40 compared to rejection sampling and the usage of the lookup table improves the performance up to 7 times.

5.3. Applications

Hair generation. Our algorithm can be used for fast placement of hair roots on a surface. The high performance of our algorithm is important especially when hair is procedurally generated at render-time as is the case in our implementation. Fig. 1 (a) shows a model with 6 million individual hairs placed by our sampling algorithm. The complete generation of hair including hair root placement took only 1.6 seconds. Our algorithm can also be easily extended to maintain coherence during animation, by simply running the preprocessing stage for only one frame of the animation.

Monte Carlo rendering with complex light sources. We can utilize our mesh sampling algorithm for sampling complex light sources represented by a triangle mesh with emission defined by an HDR texture as shown in Fig. 1 (b), where

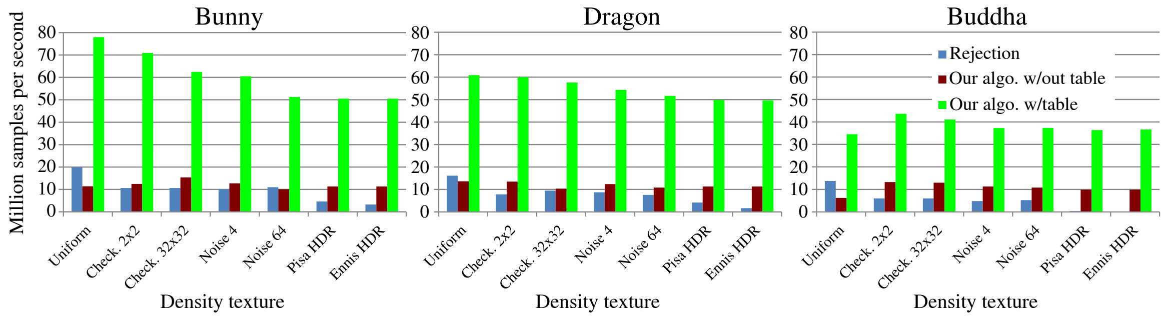

Figure 10: Sampling rate (in million samples per second) for rejection sampling and our algorithm (w/out and w/ lookup table).

an HDR image projected on the wall illuminates the scene. The images in the figure were rendered using path tracing in the same time (one hour), except for the reference image which took 16 hours to compute. Uniform light source sampling produces a highly noisy image and fails to reproduce the correct reddish tone of the illumination from a HDR map (which is due to tiny high-intensity street lights that uniform sampling almost never samples). Light source sampling using our algorithm, on the other hand, reduces the noise and reproduces the correct tint of the image early in the progressive computation. Rejection sampling is not a viable option in this case, since only 1 in about 2500 samples is accepted.

6. Conclusion

We have presented a fast algorithm for generating random samples on triangular meshes with sample density defined by a two-dimensional texture. In our tests, the algorithm achieves a 3 to 40 speedup compared to the fastest available alternative – rejection sampling. The proposed algorithm is suitable for applications that require fast mesh sampling, such as the placement of roots of procedurally generated hair or sampling complex light sources for Monte Carlo rendering. Furthermore, we have described a lookup table for speeding up sampling from a 1D discrete distribution, whose utility extends beyond mesh sampling.

An important limitation of our algorithm is the lack of control of regularity of the generated samples. We believe some of the ideas presented here could be in the future employed for fast generation of non-uniform Poisson-disk distributions over triangular meshes.

Acknowledgements

This work was supported by the Czech Science Foundation grant P202-13-26189S and by the Charles University Grant Agency (project number 1362413).

References

[BWWM10] Bowers J., Wang R., Wei L.-Y., Maletz D.: Parallel Poisson disk sampling with spectrum analysis on surfaces. *ACM Trans. Graph. 29*, 6 (2010), 166:1–166:10. 1, 2

[CCS12] Corsini M., Cignoni P., Scopigno R.: Efficient and flexible sampling with blue noise properties of triangular meshes. *IEEE Transactions on Visualization and Computer Graphics 99*, RapidPosts (2012). 1, 2

[CRW09] Cline D., Razdan A., Wonka P.: A comparison of tabular PDF inversion methods. *Comput. Graph. Forum 28*, 1 (2009), 154–160. 4

[EMP*12] Ebeida M. S., Mitchell S. A., Patney A., Davidson A., Owens J. D.: A simple algorithm for maximal Poisson-disk sampling in high dimensions. *Computer Graphics Forum 31*, 2 (May 2012). 2

[GP07] Gross M., Pfister H.: *Point-based graphics*. Morgan Kaufmann, 2007. 2

[JB02] Jensen H. W., Buhler J.: A rapid hierarchical rendering technique for translucent materials. *ACM Trans. Graph. 21*, 3 (July 2002), 576–581. 2

[LD05] Lagae A., Dutré P.: A procedural object distribution function. *ACM Trans. Graph. 24*, 4 (Oct. 2005), 1442–1461. 2

[LD08] Lagae A., Dutré P.: A comparison of methods for generating Poisson disk distributions. *Computer Graphics Forum 27*, 1 (2008), 114–129. 2

[LRR05] Lawrence J., Rusinkiewicz S., Ramamoorthi R.: Adaptive numerical cumulative distribution functions for efficient importance sampling. In *Eurographics Symposium on Rendering* (June 2005). 2

[Mei96] Meier B. J.: Painterly rendering for animation. In *Proceedings of SIGGRAPH '96* (1996), ACM, pp. 477–484. 2

[NS04] Nehab D., Shilane P.: Stratified point sampling of 3D models. In *Eurographics Symposium on Point-Based Graphics* (June 2004), pp. 49–56. 2

[OFCD02] Osada R., Funkhouser T., Chazelle B., Dobkin D.: Shape distributions. *ACM Trans. Graph. 21*, 4 (Oct. 2002), 807–832. 2

[PH10] Pharr M., Humphreys G.: *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2010. 2, 3, 4

[QM06] Qu L., Meyer G. W.: Perceptually driven interactive geometry remeshing. In *Proceedings of the 2006 symposium on interactive 3D graphics and games* (2006), I3D '06, ACM, pp. 199–206. 2

[SSKKC10] Szécsi L., Szirmay-Kalos L., Kurt M., Csébfalvi B.: Adaptive sampling for environment mapping. In *Proceedings of the 26th Spring Conference on Computer Graphics* (2010), SCCG '10, pp. 69–76. 2

[Tur92] Turk G.: Re-tiling polygonal surfaces. In *Proceedings of SIGGRAPH '92* (1992), ACM, pp. 55–64. 2

[YKH10] Yuksel C., Keyser J., House D. H.: Mesh colors. *ACM Trans. Graph. 29*, 2 (Apr. 2010), 15:1–15:11. 2