Manifold Path Guiding for Importance Sampling Specular Chains Supplemental Document

ZHIMIN FAN*, State Key Lab for Novel Software Technology, Nanjing University, China
PENGPEI HONG*†, University of Utah, United States of America
JIE GUO‡, State Key Lab for Novel Software Technology, Nanjing University, China
CHANGQING ZOU, Zhejiang Lab and State Key Lab of CAD&CG, Zhejiang University, China
YANWEN GUO‡, State Key Lab for Novel Software Technology, Nanjing University, China
LING-QI YAN, University of California, Santa Barbara, United States of America

ACM Reference Format:
Zhimin Fan, Pengpei Hong, Jie Guo, Changqing Zou, Yanwen Guo, and Ling-Qi Yan. 2023. Manifold Path Guiding for Importance Sampling Specular Chains Supplemental Document. *ACM Trans. Graph.* 42, 6, Article 257 (December 2023), 7 pages. https://doi.org/10.1145/3618360

1 SAMPLING VMF DISTRIBUTIONS

Numerically stable sampling of the vMF distribution

$$v(\boldsymbol{\omega}_D; \boldsymbol{\mu}, \kappa) = \frac{\kappa}{4\pi \sinh(\kappa)} e^{\kappa \boldsymbol{\mu} \cdot \boldsymbol{\omega}_D} \tag{1}$$

is provided by [Jakob 2012]:

$$\boldsymbol{\omega} = (\sqrt{1 - W^2} \cos(\xi_1), \sqrt{1 - W^2} \sin(\xi_1), W)^T \tag{2}$$

where W can be sampled using the inversion method:

$$F_W^{-1}(\xi_2) = \frac{\log(e^{-\kappa} + 2\xi_2 \sinh(\kappa))}{\kappa}. \tag{3}$$

ξ_1, ξ_2 are two independent, uniformly random variables in $[0, 1)$.

2 RECONSTRUCTING DISTRIBUTIONS FROM PHOTONS

Photons generated by a typical photon mapper can be used to generate the initial distribution for seed chains. In our experiment, we treat photon samples in the same manner as sub-path samples. This is well-studied in the context of guided path sampling [Jensen 1995; Vorba et al. 2014; Zhu et al. 2021]. The only difference lies in the evaluation of weights. Here, we directly use their flux as the weight:

$$w(\mathbf{x}_D, \overline{\mathbf{x}}_S^*, \mathbf{x}_L) = \Phi(\mathbf{x}_D, \overline{\mathbf{x}}_S^*, \mathbf{x}_L). \tag{4}$$

3 PSEUDO-CODES

In this section, we present several pseudo-codes to explain our implementation details further. We will release the source code upon acceptance.

3.1 Interfaces

To begin with, we list the interfaces of our spatial hierarchy and chain distribution (including the discrete decisions and directional sampling).

```
class ChainDistrbution:
 # Build a chain distribution from a set of subpath samples
 def init(subpaths):
  return ...

 # Sample the length of the chain
 def sample_n() -> int:
  return ...

 # Query the PMF for the given length
 def pmf_n(n) -> float:
  return ...

 # Sample the chain type for a given length
 def sample_tau(n) -> int:
  return ...

 # Sample the direction omega_D for a given type
 def sample_dir(tau) -> vec3:
  return ...

class SpatialHierarchy:
 # Build a spatial hierarchy from a set of subpath samples
 def init(subpaths):
  return ...

 # Query the corresponding ChainDistribution object for a
 ↪  given configuration (xD, xL).
 def query(xD, xL) -> ChainDistribution:
  return ...
```

*Joint first authors.
†The work was done at Nanjing University.
‡Corresponding authors.

Authors' addresses: Zhimin Fan, zhiminfan2002@gmail.com, State Key Lab for Novel Software Technology, Nanjing University, Nanjing, China; Pengpei Hong, hpommpy@gmail.com, University of Utah, Salt Lake City, United States of America; Jie Guo, guojie@nju.edu.cn, State Key Lab for Novel Software Technology, Nanjing University, Nanjing, China; Changqing Zou, changqing.zou@zju.edu.cn, Zhejiang Lab and State Key Lab of CAD&CG, Zhejiang University, Hangzhou, China; Yanwen Guo, ywguo@nju.edu.cn, State Key Lab for Novel Software Technology, Nanjing University, Nanjing, China; Ling-Qi Yan, lingqi@cs.ucsb.edu, University of California, Santa Barbara, Santa Barbara, United States of America.

0730-0301/2023/12-ART257 $15.00
https://doi.org/10.1145/3618360

3.2 Main algorithm

We present our main algorithmic framework in this subsection, which is a class instantiated for each pair of separators generated in a regular path tracing loop with standard emitter sampling.

```
spatial_hierarchy = make_spatial_hierachy(None)

class ManifoldPathGuiding:
 # Object of this class is created for each pair of separators
 def init(xD, xL, num_external_vertices):
  self.xD = xD
  self.xL = xL
  # Query the spatial hierarchy only once
  self.guiding_distr = spatial_hierarchy.query(xD, xL)
  # Reuse the depth and Russian Roulette setting in the underlying path tracer
  self.ctx = (MAX_DEPTH - num_external_vertices, RR_DEPTH - num_external_vertices, RR_PROB)
  # Only perform guiding if spatial hierarchy is already built
  self.fraction = GUIDE_FRAC if spatial_hierarchy.valid() else 0


 # Sample the number of bounce
 def sample_length():
  # One-sample MIS (Eq. 12)
  if rand() < self.fraction:
   return self.guiding_distr.sample_n()
  else:
   return self.uniform_sample_length(n, self.ctx)


 # Evaluate the PMF of specified number of bounce
 def pmf_length(n):
  # Mixture density (Eq. 12)
  return lerp(self.uniform_pmf_length(n, ctx),
     self.guiding_distr.pmf_n(), self.fraction)


 # Sample a seed chain without historical information
 def uniform_sample_seed(n):
  # Initialization strategy (Sec. 5.4, Fig. 17)
  if INIT_STRATEGY == "surface":
    # Uniformly select a specular surface point
    x1 = self.sample_surface()
  elif INIT_STRATEGY == "direction":
    # Uniformly pick a direction.
    dir = warp.square_to_uniform_hemisphere(sampler.next_2d())
    x1 = scene.intersect(xD, dir)
  elif INIT_STRATEGY == "photon":
    # Supplemental Sec. 2
    dir = sample_photon_distribution()

  # xD is needed for the Fresnel term querying
  seed_chain = [self.xD, x1]
  # Ray tracing loop (Eq. 5)
  for i in range(n - 1):
   # Sample scattering type proportional to the Fresnel term
   scatter_type = sample_scatter_type(seed_chain[-2:])
   # Collect vertices
   new_vertex = scene.intersect(seed_chain[-2:], scatter_type)
   seed_chain.append(new_vertex)
  return seed_chain


 # Sample a seed chain using our distribution
 def guided_sample_seed(n):
  tau = self.guiding_distr.sample_tau(n)
  dir = self.guiding_distr.sample_dir(tau)
  # Eq. 5
  return collect_vertices(tau, dir)


 # Sample a seed chain
 def sample_seed(n):
  # One-sample MIS (Eq. 12)
  if rand() < self.fraction:
   return self.guided_sample_seed(n)
  else:
   return self.uniform_sample_seed(n)


 # Sample an admissible chain (may be diverged)
 def sample_solution(n):
  seed_chain = self.sample_seed(n)
  return self.manifold_walk(seed_chain)


 # Reciprocal probability estimation (Eq. 14)
 def bernoulli(n, x):
  ans = 1
  # Prevent infinite loop due to numerical issues
  MAX_ITER = 1e6
  # Repeat trials until the same solution is founded
  # Compare both the type and the direction here
  while self.sample_solution(n) != x and ans <= MAX_ITER:
   ans += 1
  if ans > MAX_ITER:
    # Failed, discard this solution
   return None
  return ans


 # Entry point
 def specular_chain_sampling():
  # Sample the length first
  n = sample_length()
  # PMF is factored out and evaluate analytically (Eq. 14)
  pmf_n = pmf_length(n)
  # Sample an admissible chain
  ans = sample_solution(n)
  if ans.valid() == False:
    # Diverged, return zero throughput
    return 0
  # Estimate the reciprocal_probability p(xS* | xD, xL, n)
  inv_pdf = bernoulli(n, ans)
```

```
  return ans * inv_pdf / pmf_n  # (Eq. 14)


def render_one_iteration(spp, is_training):
 img, subpath_samples = render_blocks(spp)
 # Fitting distribution
 if is_training:
  spatial_hierarchy =
  ↪  make_spatial_hierarchy(subpath_samples)
```

3.3 Spatial Neighboring

When comparing KNN and STree, it's important to note that KNN generates chain distribution objects on-the-fly, whereas STree pre-computes them when building the hierarchy.

KNN. We use KDTree in ANN library [Mount and Arya 2010] for the implementation of the KNN. The approximated searching in their library does not offer significant improvements in our test, so we use accurate searching instead.

```
class SpatialHierarchyKNN: SpatialHierarchy
 def init(subpaths):
  # Building 6D kdtree according to (xD, xL)
  self.kdtree = build_kdtree(subpaths)

 def query(xD, xL) -> ChainDistribution:
  # Query kdtree to get the nearest samples
  samples = self.kdtree.query(xD, xL)
  # Build a chain distribution on-the-fly
  return make_chain_distribution(samples)
```

STree. Our implementation of the 6D spatial adaptive binary tree generally follows [Müller 2019]. The major difference is that we build the spatial structure and splat the samples simultaneously. We refine the structure while keeping track of the set of sub-path samples attached to each (current) leaf node. When a leaf node is split, we realize spatial filtering by copying the leftmost $\lfloor \varepsilon k \rfloor$ samples of the right node to the left child and the rightmost $\lfloor \varepsilon k \rfloor$ samples of the left node to the right child, with $\varepsilon = 10\%$ being the spatial filtering threshold and k being the number of samples in the current node. In each node, when a sample after filtering is outside the bounding box of the node, and the distance to the bounding box is larger than 2ε times the length of the current extent, we discard it. After subdivision, the left and right subtrees are handled recursively using thread-level parallel for acceleration. In our implementation, the spatial structure is completely rebuilt in each iteration and does not adopt an incremental updating process.

3.4 Chain Distribution

Bounce and type sampling. It's a standard binning and normalizing process.

```
class ChainDistributionImpl: ChainDistrbution
 def init(subpaths):
  super.init(subpaths)
  self.distr_n = {}
  self.distr_tau = {}
  # Summation for each n and tau (Eq. 9 and Eq. 10)
  for sp in subpaths:
   # sp.weight is defined in Eq. 8
   self.distr_n[sp.n] += sp.weight
   self.distr_tau[sp.n][sp.tau] += sp.weight
  # Normalization
  self.distr_n = Distribution1D(self.distr_n)
  for key in self.distr_tau:
   self.distr_tau[key] =
   ↪  Distribution1D(self.distr_tau[key])


 def sample_n() -> int:
  return self.distr_n.sample()


 def pmf_n(n) -> float:
  return self.distr_n.pdf(n)


 def sample_tau(n) -> int:
  return self.distr_tau[n].sample()
```

KNN-based particle footprint. We choose to use particle footprints [Hey and Purgathofer 2002] with directional density estimation for their accuracy, as discussed in our validation of building blocks. Note that we cache a mapping from sub-path samples to their kernel radius, which leads to much faster evaluation in our test.

```
class ChainDistributionKNN: ChainDistributionImpl
 def init(subpaths):
  super().init(subpaths)
  self.distr_omega = {[] for tau in self.distr_tau}
  # Build a discrete distribution of samples for each
  ↪  chain type
  for sp in subpaths:
   self.distr_omega[sp.tau].append((sp.omega, sp.weight))
  self.distr_omega = Distribution1D(self.distr_omega)
  # Cache the kernel radius leads to about 10x faster in
  ↪  practice
  self.radius_cache = {}


 def sample_dir(tau) -> vec3:
  # Sample the kernel direction first
  omega = self.distr_omega.sample()
  if omega not in self.radius_cache:
   # Cache miss
   min_dis = inf
   # Estimating the kernel radius using nearest neighbor
   ↪  distance
   for omega_ in self.distr_omega:
    min_dis = min(min_dis, l2norm(omega - omega_))
   # Write to the cache
   self.radius_cache[omega] = min_dis
  radius = self.radius_cache[omega]
  # Sample the kernel (Eq. 11)
  return sample_vmf(omega, radius)
```

Directional quad-tree. Recall that we also compare our directional density estimation method with SDTree. This part follows PPG, and please refer to [Müller et al. 2017] for more algorithmic details.

4 DETAILED RESULTS

We provide full results comparing different building blocks, spatial filtering strategies, and choice of spatial neighboring size in Fig. 1 and Fig. 2. We also validate the effect of bounce sampling, type sampling, and directional sampling in Fig. 3.

5 DISCUSSIONS

Extension to paths with multiple chains. Our sampling strategy for specular chains can be generalized to multiple separators. In general, the separators and specular chains are sampled incrementally: we first importance sample specular chains between the first and second separator. Then, after the third separator is determined, we sample specular chains between the second and third separator, and so on. Note that our method only focuses on sampling specular chains, and for the importance sampling of a complete path in path space, it is necessary to also consider the importance sampling of separators. We leave this for future work.

Exploration of chains with large roughness. Note that we still use the direction from one separator to the first specular vertex to represent a chain of a particular type. This way, we actually model the marginal of chains in glossy cases, which slightly increases variance. However, conventional path guiding [Ruppert et al. 2020; Vorba et al. 2019] or those designed for glossy cases [Li et al. 2022; Loubet et al. 2020] would be more suitable for this case.

Temporal stability. The video demonstrates the strong temporal stability of our method, which is achieved due to its regular Monte Carlo nature. Unlike MCMC approaches, our method effectively avoids the occurrence of blotchy artifacts and temporal instability.

However, it is important to acknowledge that there are still inherent limitations of temporal stability in our method. The training process may not encompass all possible solutions for every configuration. In such cases, a small region of space may exhibit higher variance compared to its surroundings. This arises from the inherent limitations of path guiding. To address this issue, when rendering animations, employing the temporal distribution reuse would be beneficial.

Box spatial filtering. Here, we explain why we do not adopt the *box filtering* [Müller 2019] for the spatial hierarchy. For box filtering, theoretically, a single sample will contribute to at least 2^d leaf nodes, where d represents the dimension (6 in our method). As a result, if we start with K sub-path samples, we could end up with at least $64K$ samples after applying box filtering, which is relatively unbearable. In contrast, our method increases the total number of samples by at most 2ε per layer. Thus, if there are h layers in the spatial structure and K original samples, the total number of samples after filtering will never exceed $(1+2\varepsilon)^h K$. Actually, in all our test scenes, the total number of samples after filtering is generally around twice the size of the unfiltered sample set.

Continuous admissible chain spaces. In the context of general surface representations, scenarios involving a continuous 1D subspace of admissible chains can be constructed. For instance, consider a cylinder that has reflective properties on the inside, where a light source and a camera are positioned at the centers of the cylinder's two caps [Wang et al. 2020]. However, as Zeltner et al. [2020] discussed in their paper, it is important to note that such cases have limited relevance when it comes to rendering natural scenes. This is because even a slight perturbation in the surface geometry would disrupt the symmetries required to create a 1D solution subspace. Following prior works [Walter et al. 2009; Wang et al. 2020; Zeltner et al. 2020], we disregard this particular corner case.

REFERENCES

Heinrich Hey and Werner Purgathofer. 2002. Importance Sampling with Hemispherical Particle Footprints. In *Proceedings of the 18th Spring Conference on Computer Graphics* (Budmerice, Slovakia) *(SCCG '02)*. Association for Computing Machinery, New York, NY, USA, 107–114. https://doi.org/10.1145/584458.584476

Wenzel Jakob. 2012. *Numerically stable sampling of the von Mises Fisher distribution on S^2 (and other tricks)*. Technical Report. Cornell University, Department of Computer Science.

Henrik Wann Jensen. 1995. Importance Driven Path Tracing using the Photon Map. In *Rendering Techniques '95*, Patrick M. Hanrahan and Werner Purgathofer (Eds.). Springer Vienna, Vienna, 326–335.

He Li, Beibei Wang, Changhe Tu, Kun Xu, Nicolas Holzschuch, and Ling-Qi Yan. 2022. Unbiased Caustics Rendering Guided by Representative Specular Paths. In *SIGGRAPH Asia 2022 Conference Papers* (Daegu, Republic of Korea) *(SA '22)*. Association for Computing Machinery, New York, NY, USA, Article 41, 8 pages. https://doi.org/10.1145/3550469.3555381

Guillaume Loubet, Tizian Zeltner, Nicolas Holzschuch, and Wenzel Jakob. 2020. Slope-Space Integrals for Specular next Event Estimation. *ACM Trans. Graph.* 39, 6, Article 239 (nov 2020), 13 pages. https://doi.org/10.1145/3414685.3417811

David Mount and Sunil Arya. 2010. ANN: A Library for Approximate Nearest Neighbor Searching. https://www.cs.umd.edu/ mount/ANN/.

Thomas Müller. 2019. "Practical Path Guiding" in Production. In *ACM SIGGRAPH Courses: Path Guiding in Production, Chapter 10* (Los Angeles, California). ACM, New York, NY, USA, 18:35–18:48. https://doi.org/10.1145/3305366.3328091

Thomas Müller, Markus Gross, and Jan Novák. 2017. Practical Path Guiding for Efficient Light-Transport Simulation. *Computer Graphics Forum* 36 (07 2017), 91–100. https://doi.org/10.1111/cgf.13227

Lukas Ruppert, Sebastian Herholz, and Hendrik P. A. Lensch. 2020. Robust Fitting of Parallax-Aware Mixtures for Path Guiding. *ACM Trans. Graph.* 39, 4, Article 147 (aug 2020), 15 pages. https://doi.org/10.1145/3386569.3392421

Jiří Vorba, Johannes Hanika, Sebastian Herholz, Thomas Müller, Jaroslav Křivánek, and Alexander Keller. 2019. Path Guiding in Production. , Article 18 (2019), 77 pages. https://doi.org/10.1145/3305366.3328091

Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. 2014. On-Line Learning of Parametric Mixture Models for Light Transport Simulation. *ACM Trans. Graph.* 33, 4, Article 101 (jul 2014), 11 pages. https://doi.org/10.1145/2601097.2601203

Bruce Walter, Shuang Zhao, Nicolas Holzschuch, and Kavita Bala. 2009. Single Scattering in Refractive Media with Triangle Mesh Boundaries. *ACM Trans. Graph.* 28, 3, Article 92 (jul 2009), 8 pages. https://doi.org/10.1145/1531326.1531398

Beibei Wang, Miloš Hašan, and Ling-Qi Yan. 2020. Path Cuts: Efficient Rendering of Pure Specular Light Transport. *ACM Trans. Graph.* 39, 6, Article 238 (nov 2020), 12 pages. https://doi.org/10.1145/3414685.3417792

Tizian Zeltner, Iliyan Georgiev, and Wenzel Jakob. 2020. Specular Manifold Sampling for Rendering High-Frequency Caustics and Glints. *ACM Trans. Graph.* 39, 4, Article 149 (jul 2020), 15 pages. https://doi.org/10.1145/3386569.3392408

Shilin Zhu, Zexiang Xu, Tiancheng Sun, Alexandr Kuznetsov, Mark Meyer, Henrik Wann Jensen, Hao Su, and Ravi Ramamoorthi. 2021. Photon-Driven Neural Reconstruction for Path Guiding. *ACM Trans. Graph.* 41, 1, Article 7 (nov 2021), 15 pages. https://doi.org/10.1145/3476828

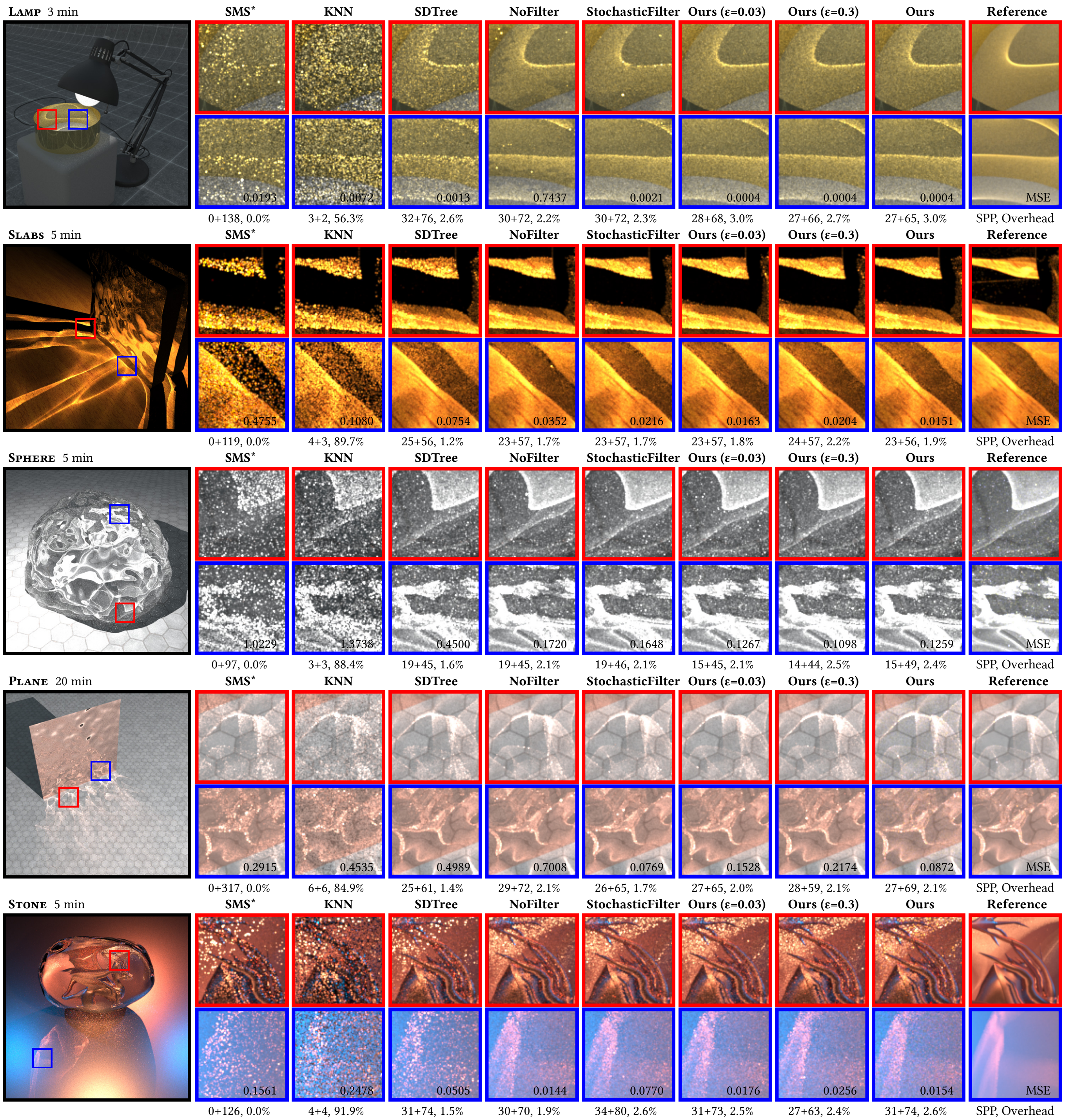

Fig. 1. **Choices of building blocks.** We perform equal-time comparisons of various neighbor searching methods and distribution representations.

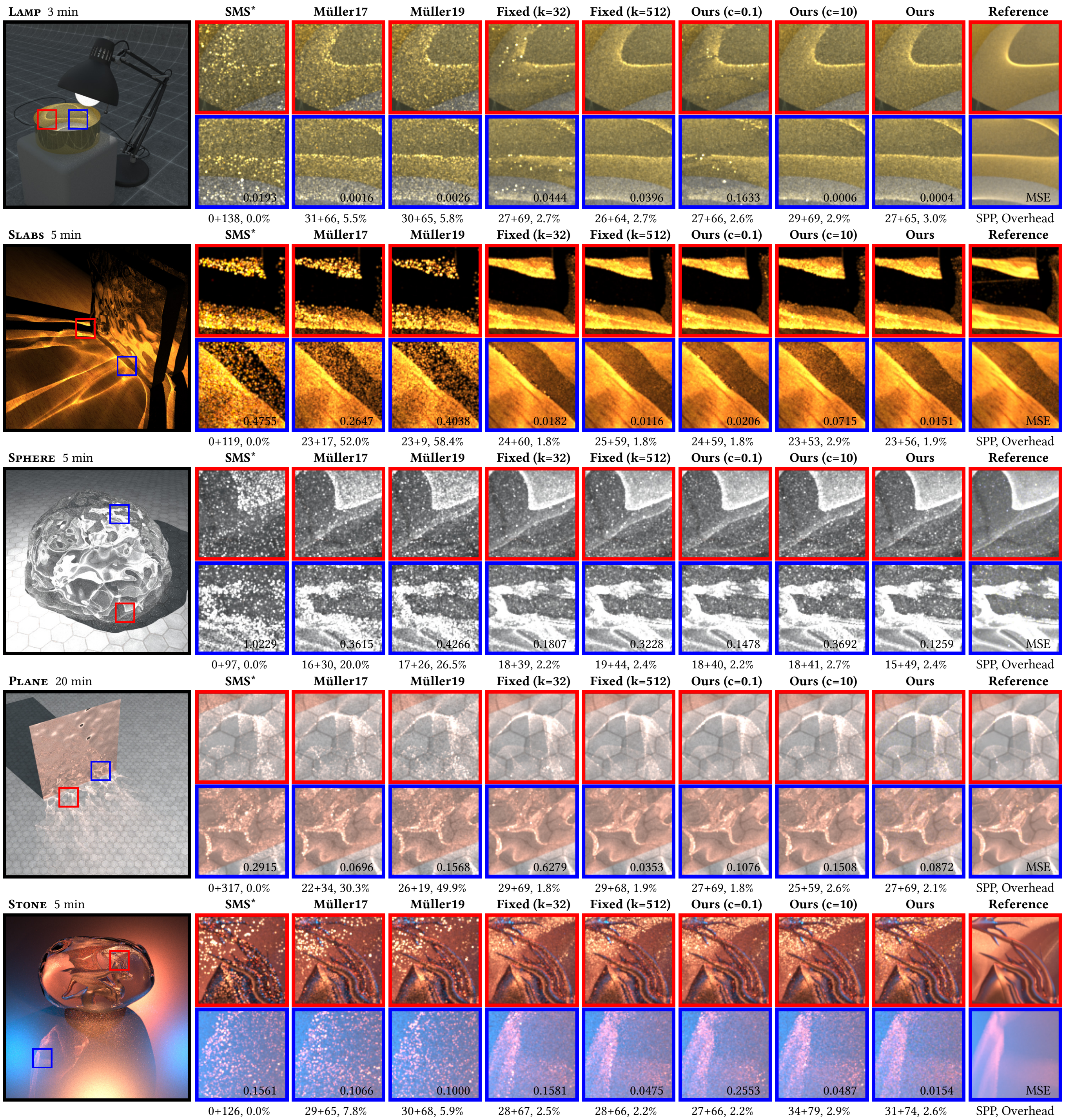

Fig. 2. **Equal-time comparison on various strategies for deciding spatial neighboring size.** Here, Müller17 and Müller19 stand for the formula proposed by [Müller et al. 2017] and [Müller 2019], respectively. Fixed means using a constant spatial neighboring size. We also include two variants of our automatic threshold $\sqrt{|\mathcal{S}|}$ by adding an extra coefficient c.

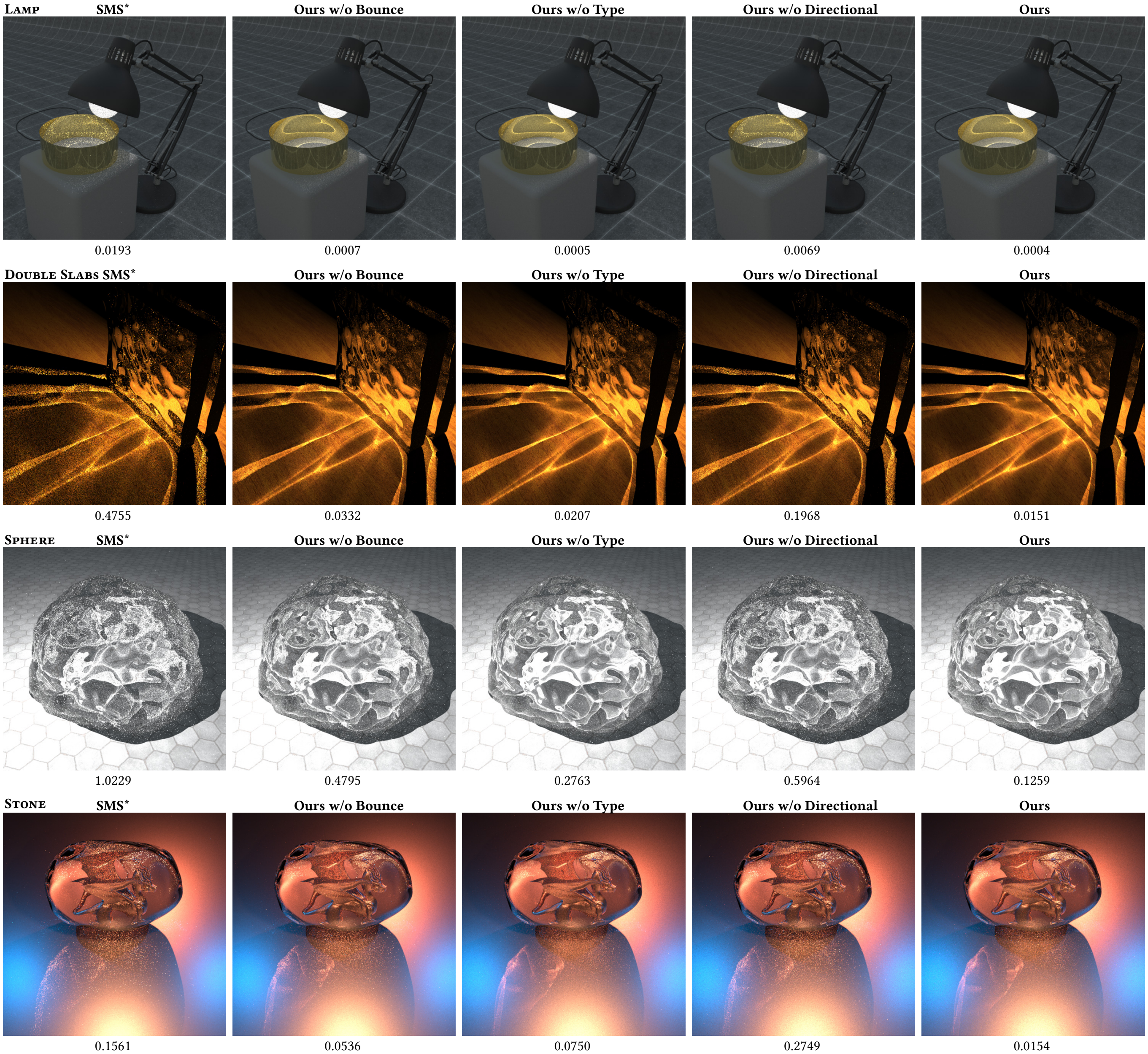

Fig. 3. **Ablation study.** We perform equal-time comparison (rendering time is the same as Fig. 2) to validate the effect of bounce sampling, type sampling, and directional sampling. All these parts are essential for efficient importance sampling of specular chains. Quantitative error in terms of MSE is reported.