1 DERIVING RECURSIVE ENSEMBLE AVERAGE LIGHT TRANSPORT

The step from Eq. (12) to Eq. (14) in the main document is not immediately obvious. It does not require any deep insights, but a thorough reproduction of the intermediate steps, while lengthy, provides extra intuition for our method. We provide this here in a slightly more general form that should apply to any quantity that can be estimated with a recursive Monte Carlo estimator.

1.1 Expectations over realizations of a Gaussian Process

We will often want to compute expectations “over realizations of a Gaussian Process”, that is, functional integrals of the form

$$\langle LF_{GP}(\mu, k) \rangle = \int_{GP(\mu, k)} LF \, d\gamma_{\mu, k}(f), \quad (1)$$

where \( f \) is a realization of the Gaussian process \( GP(\mu, k) \), \( L \) an operator acting on \( f \), and \( \gamma_{\mu, k}(f) \) the classical Wiener measure of \( f \) with respect to \( GP(\mu, k) \), i.e. the probability density of sampling \( f \sim GP(\mu, k) \). If the operator \( L \) is linear in \( f \), we can resolve this in the following straightforward fashion

$$\int_{GP(\mu, k)} LF \, d\gamma_{\mu, k}(f) = LF \int_{GP(\mu, k)} \, d\gamma_{\mu, k}(f) = L\mu, \quad (2)$$

but many operators, in particular light transport, which we study in this paper, are not linear in \( f \), and the above simplification does not hold. In the following, we will drop the mean \( \mu \) and covariance kernel \( k \) for notational convenience. A rigorous treatment of functional integrals of this form is out of the scope of this work. Instead, we provide this here in a slightly more general form that should apply to any quantity that permits extra intuition for our method. We provide this here in a thorough reproduction of the intermediate steps, while lengthy, provides extra intuition for our method. We provide this here in a slightly more general form that should apply to any quantity that can be estimated with a recursive Monte Carlo estimator.

1.1.1 Conditioned Expectations. Additionally, we can write conditional expectations as usual:

$$\langle LF_{GP}(f_A = a) \rangle = \langle L[f_{PA}\cdot a] \rangle_{GP_{PA}}, \quad (6)$$

1.1.2 Expectations of delta functions. We can write an expectation containing a delta function over some subset of the domain \( A \) as

$$\langle \delta(f(A) - a) LF_{GP} \rangle = \int_{GP} \delta(f(A) - a) LF \, df(f) = \int_{GP_A} \delta(f_A - a) LF \, df(f | f_A) \, df_A(f_A) \quad (7)$$

where \( A \) is a subdomain of the domain of \( GP \), \( A^c \) its complement, \( f_A \) and \( f_{A^c} \) realizations of \( GP \) over \( A \) and \( A^c \) respectively and \( \{ f_A, f_{A^c} \} \) the “concatenation” of the individual realizations. The first equality simply decomposes \( f \) into \( f_A \) and \( f_{A^c} \). Going to the second line, we apply the law of conditional expectations. That is, we can sample \( f \) “all at once” or first sample a part of \( f \) (\( f_A \)) and then the rest (\( f_{A^c} \)) conditioned on the part that we sampled first. The last line simply says that we don’t have to restrict the inner process to \( A \) and can instead “resample” over the whole domain. For the part of the domain covered by the “conditioning region” \( A \), we will simply get back the same values that we conditioned on.
inner integral. We can then apply the definition of the delta function and drop the outer integral, replacing the integration variable \( f_A \) with \( a \) and evaluating the measure \( y_A(f_A) \) at \( a \).

1.1.3 Expectations of indicator functions. We can write an expectation containing an indicator function over some subset of the domain \( A \) as

\[
\langle 1(f(A) > 0) \rangle_{\text{GP}} = \int_A 1(f(A) > 0) Lf \, dy(f) = \int_{A^c} \mathcal{L}f \, dy(f) = \int_{A^c} \mathcal{L}f \, dy(f) = \int_{A^c} \mathcal{L}f \, dy(f) = \langle (\mathcal{L}f)_{\text{GP}}(f_A) \rangle_{\text{GP}^*}
\]

where \( GP^* \) is the process restricted to all positive realizations over the index set \( A \). Note that \( A \) can be a manifold in the domain of the GP (e.g., a 1D line segment in a 3D domain).

1.2 Application to light transport

With all of the puzzle pieces in place, we can apply them to Eq. (12) from the main paper, letting \( Lf \equiv L_{f_i}(x, \omega) \), we are particularly interested in the inner integral

\[
\int_0^\infty \left[ \int_\mathbb{R} \rho(x_i) \int_{\text{GP}^*} \delta^L(x_i, n) L_{f_i}(x_i, \omega_i) \, dy(f) \right] \, d\omega_i \, dt,
\]

and pull it out for more compact derivations, treating \( n \) and \( t \) as free variables. We first apply Eq. (7)

\[
\langle \delta^L(f(x_i)) > 0 \rangle = \delta^L(f(x_i) - n) \cdot L_{f_i}(x_i, \omega_i) = y_{x_i}(n) \langle L_{f_i}(x_i, \omega_i) \rangle_{\xi = f(x_i) = n, \xi = \nabla f(x_i) = n}
\]

and then Eq. (8) and arrive at

\[
y_{x_i}(n) \langle L_{f_i}(x_i, \omega_i) \rangle_{\xi = f(x_i) = n, \xi = \nabla f(x_i) = n}
\]

Since \( f(x_i) \) is already conditioned on \( \xi = f(x_i) = 0, \nabla f(x_i) = n \), we can add these to the inner condition without changing the expected value.

\[
y_{x_i}(n) \langle L_{f_i}(x_i, \omega_i) \rangle_{\xi = 0, \xi = \nabla f(x_i) = n}
\]

where \( \xi' = f(x_i) = 0, \nabla f(x_i) = n \). Plugging this back into Eq. (9) and expanding the outer ensemble average into its integral form, we get Eq. (14) from the main text.

1.3 Deriving the GPIS density in the Renewal and Renewal+ models

In Section 5 of the main document, we use the GPIS density to make connections to microfacet and participating media theory. We briefly want to give the full derivation that connects Eq. (14) and Eq. (17).

Recall that for the Renewal and Renewal+ models, we only condition on values at path vertices, not values along path segments. That means that \( R' \) does not contain \( f_{(0, x_i)} \) and hence does not depend on the integration variable in the innermost integral. This allows us to pull the recursive ensemble average out of the inner integral in Eq. (14) of the main text as

\[
\langle L_{r_i}(x^2, \omega) \rangle_{\xi} \approx \int_0^\infty \int_\mathbb{R} \rho(x_i) y_{x_i}(0, n | \xi) \langle L_{r_i}(x^2, \omega) \rangle_{\xi = f_{(0, x_i)} \wedge \xi = x_i} \, d\omega_i \, dt
\]

And finally, defining \( \Gamma(t, n | \xi) \) as the single realizaton containing an indicator function over some subset of the domain \( A \), we get Eq. (17) in the main text.

2 GAUSSIAN PROCESS DETAILS

2.1 Kernel Functions

We give analytic forms of a set of kernels and their spectral densities. Note that our method is not restricted to only these kernels since we support any smooth kernel. We pick these kernels in particular because they cover a range of different fundamental properties, such as non-locality and non-positiveness, which have the potential to strongly affect the results of our method. We give their most simple forms and denote the common parameters standard deviation and length scale with \( \sigma \) and \( f \), respectively. Deriving the spectral density, and especially sampling from it, is often non-trivial and depends on the number of dimensions. We give spectral densities for \( d = 3 \) where we can, and \( d = 2 \) otherwise. While theoretically, one does not have to sample basis functions proportional to the spectral density (one could choose a set using some proposal distribution and re-weight, like in any other Monte Carlo estimator), sampling proportional to the spectral density drastically reduces the number of basis functions required. For a visual overview of the listed kernels, see Fig. 3.

Squared Exponential. This is probably the most widely used kernel in Gaussian process literature.

\[
k_{SE}(x, y) = \sigma^2 \exp\left(-\frac{\|x - y\|^2}{l^2}\right)
\]

\[
\rho_{SE}(\omega) = \frac{\sigma^2 e^{-\frac{1}{2} l^2 \omega^2}}{\sqrt{2\pi}}
\]
stable. It also induces the most “exponential” first-passage times and is a great choice when the aim is to represent close-to-classical participating media.

**Algorithm 1: Sample \( p^{SE} \)**

\[
\xi \sim U(0, 1) \\
r \leftarrow \sqrt{2 - \log(\xi)} \\
\phi \sim U(0, 2\pi) \\
\text{return } [\sin(\phi), \cos(\phi)]^\top \cdot r
\]

**Rational Quadratic.** The rational quadratic kernel is a superposition of infinitely many squared exponential kernels with different length scales. The weighting of these length scales is determined by the additional parameter \( a \).

\[
k^\text{RQ}_a(x, y) = \sigma^2 \left( 1 + \frac{||x - y||^2}{2a^2} \right)^{-\alpha} \\
p^\text{RQ}_a(\omega) = \frac{\sigma^2}{\Gamma(a)} \left[ \omega^2 + \frac{1}{\omega^2} \right]^{-\frac{a+1}{2}} K_\frac{\alpha}{2}(\frac{\sqrt{2|\omega|}}{\sqrt{2a}})
\]

In particular, \( \lim_{a \to \infty} k^\text{RQ}_a(x, y) = k^{SE}(x, y) \). For small \( a \), the realizations of the rational quadratic kernel have “fractal” properties.

**Algorithm 2: Sample \( p^\text{RQ}_a \)**

\[
\tau \sim \Gamma(a, I^{-2}) \\
\text{return } \text{Sample } p^{SE} \text{ with } l = \tau^{-2}
\]

**Periodic.** Periodic kernels produce periodic realizations. This strongly affects the light transport in the scene and is a very challenging case for our algorithm. Any limit to the memory will produce drastically different results. Additionally, this shows that the covariance matrix is not sparse in general.

\[
k^\text{Per}_\lambda(x, y) = \sigma^2 \exp\left( -\frac{2 \sin^2(\pi ||x - y|| / \lambda)}{\lambda^2} \right)
\]

**Locally Periodic.** To ease some of the difficulties that come with the periodic kernel, while still preserving its interesting properties locally, we make use of kernel composition to construct a “locally periodic” kernel.

\[
k^\text{LocPer}_\lambda(x, y) = k^\text{LocPer}_\lambda(x, y) \ast k^{SE}(x, y) \\
p^\text{LocPer}_\lambda(\omega) = (p^\text{LocPer}_\lambda \ast p^{SE})(\omega),
\]

where \( \ast \) denotes convolution. This kernel is “locally periodic” in the sense that in realizations new each “repetition” is allowed to deviate slightly from the previous one. The length scale of the squared exponential kernel controls how quickly disorder settles in.

**Thin-plate.** Williams and Rasmussen [2006] suggest the “thin-plate” covariance. It is motivated by the fact that posterior GPISes should not “return to zero” for locations far away from the conditioning points. Instead, it is a more reasonable assumption that the magnitude of the sampled realizations should increase as you move away from the surface sample.

\[
k^\text{TP}(x, y) = \sigma^2 (2||x - y||^2 + 3R||x - y||^2 + R^3) \\
p^\text{TP}(\omega) = \frac{3\sigma^2}{\pi^2a^6}
\]

Note that this covariance is only p.s.d. for \( ||x - y|| \leq R \), so \( R \) has to be chosen to be at least as large as the size of the domain.

3 **ALGORITHMS**

In this section, we discuss the rendering algorithms we use to visualize Gaussian process implicit surfaces in more detail. These algorithms are, essentially, direct Monte Carlo estimators of Eq. (8) and Eq. (14). In particular, we formulate them in such a way that we avoid ever having to evaluate the GPIS \( \Gamma(r, m) \) density directly.

3.1 **Global realization sampling via weight-space GPs**

We first describe a method to directly compute Eq. (9). Recall that there, we require realizations that span the whole space we want to trace paths through at once, and sampling from high dimensional multivariate Gaussian distribution quickly becomes expensive, as discussed in Section 3. If we limit ourselves to stationary kernels, we can apply the weight space decomposition to the Gaussian process instead [Wilson et al. 2020]. Here we approximate realizations as a weighted sum of a finite number \( M \) of basis functions, e.g. random Fourier features \( \phi_i(x) = \sqrt{2m} \cos(\theta_i)^T x + \tau_i \) where \( \theta_i \) are chosen according to the spectral density \( S(\cdot) = \int k(r) \frac{1}{4\pi} (2\pi r)^{d/2} \, dr \) of the covariance function, with \( \int_{d/2-1} \) being a Bessel function of order \( d/2 - 1 \). Additionally, \( \tau_i \sim \text{Uniform}(0, 2\pi) \) and \( d \) is the dimensionality of the domain. Sampling a realization is then equivalent to sampling the weights for the basis function, and for an unconditioned process, the weights are independently Gaussian distributed.

The realization can then be evaluated at any set of points \( X \) in \( \mathcal{O}(N) \) time as

\[
f(X) \approx \sum_{i=1}^M w_i \phi_i(X), \quad w_i \sim \mathcal{N}(0, 1).
\]
This lets us draw correlated samples at a cost only dependent on the chosen number of basis functions instead of on the number of correlated samples.

### 3.1.1 Practical considerations
In particular, once we have sampled weights for the basis functions, we can treat the resulting linear combination like any other implicit function and trace against it by finding the first zero crossing along each ray. Unfortunately, we can’t use a fast root-finding method like sphere tracing because the Lipschitz constant of our basis grows linearly with the number of basis functions, forcing sphere tracing to take very small steps. Instead, we rely on the slower but more general method of root-finding via affine arithmetic [Sharp and Jacobson 2022]. Because the basis functions use a limited number of operations, implementing them in affine arithmetic is relatively straightforward. When we have found an intersection, we can analytically compute the normal at that point and do shading as for any other implicit surface. We illustrate this in Fig. 1. Note that because the basis functions use non-linear operations (cosines), the bounds produced by affine arithmetic are not tight. Nonetheless, we are guaranteed not to miss intersections. Weight-space GPIS rendering makes it practical to get results equivalent to the Global model, assuming that the number of basis functions is chosen to be large enough. This means that this method is mainly useful for verifying more general ones, such as the function-space algorithms we will discuss next, in simplified scenarios. There are several practical concerns when using weight-space GPISs for GPIS rendering. The most obvious one is that the standard weight-space formulation can only support stationary covariance kernels. The mean can still vary arbitrarily, allowing us to model interesting scenes, but we cannot vary properties such as the roughness of surface-type GPIS or include surface-type and volume-type GPIS in the same scene. Choosing the correct number of basis functions is critical, and the exact number depends on the covariance kernel. When choosing a low number of basis functions, the underlying kernel is approximated poorly, which leads to significantly different realizations (see Fig. 2). Empirically, we observe that kernels with longer-than-exponential tails, such as the rational quadratic kernel, require a much larger number of basis functions than, for example, the squared exponential kernel. Finally, one has to be able to sample from the spectral density of the chosen covariance kernel. This is not always trivial to do, and we provide procedures for the stationary kernels used in this work in Section 2.1 of the supplemental.

### 3.2 Practical function-space sampling strategies
In Section 4.2 of the main text we discussed our function-space sampling strategy and left off with two practical issues to solve in an implementation: Determining scene bounds along rays and limiting the number of correlated values that have to be sampled jointly. Here we go into more detail on how we handle both of these issues.

**Determining scene bounds.** One simple and practical solution to define scene bounds is to prescribe them a priori. For example, we can simply intersect the ray with a box or a sphere that we define as the limits of the scene, resulting in only having to sample a realization over a finite ray segment. A more principled option is to consider that many scenes are naturally finite in extent. In our case, this occurs when $\mu(x) \to \infty$ for large $x$, e.g. when the mean surface is modeled using an SDF. Then, for large $x$, $P(f(x) \leq 0) \to 0$, i.e. the probability of sampling a value that we see as “inside” a GPIS goes towards 0. We can define the bounds of the scene as the region for which $P(f(x) \leq 0) > \epsilon$ for some small epsilon. That is, we discard regions of space for which we are very unlikely to sample a zero crossing. Evaluating this “point-wise occupancy” is cheap as it is just the CDF of a mono-variate normal distribution. We can intersect the ray with the level set $\{ x \mid P(f(x) \leq 0) = \epsilon \}$ using ray marching. Accuracy here is not critical as long as we are conservative. This gives us a finite ray segment to work with as long as our scene is finite. The only case this does not cover is if we have an infinite scene. The most common example of this would be a scene filled with an infinite volume-type GPIS. Here we will have to rely fully on the second level of our strategy.

**Progressive ray segment sampling.** There are two cases where we can’t simply distribute sample points along a ray segment: We have an infinitely large scene, or the ray segment is long compared to the length scale of the covariance kernel (in which case the $O(n^2)$ scaling of function-space sampling would make sampling a sufficiently dense set of points too slow). In practice, we found that taking anything more than 256 correlated samples at a time significantly impacts render times. To overcome this, we apply the same strategy we applied in Section 4 to path segments to ray segments. That is, we sample realizations $f(x_i, x_{i+1}) \sim \text{GP}(x_i, x_{i+1})|\zeta^{\prime}\alpha^\prime$ with $t_0 = 0$ and $t_{i+1} = t_i + n \cdot \Delta t$, where $\Delta t$ is a step size chosen based on the covariance kernel, and $\zeta^{\prime}$ the condition based on the chosen memory model. We repeat this until $I(f(x_i, x_{i+1})) = 0$, i.e. $f(x_i, x_{i+1})$ contains a zero crossing. We then proceed as before, sampling a normal followed by an outgoing direction. This process is illustrated in Fig. 8.
3.3 Thoughts on next-event estimation

Our function-space sampling approach enables us to employ next-event estimation to a much greater degree than the global weight-space sampling approach. When determining a global realization before tracing a path, we fix not only the realization’s values but also its gradients and, hence, normals. Then, during tracing, there is exactly one possible normal at each intersection point. This means that, when using a mirror micro-BSDF, we cannot perform next-event estimation since, with the normal at the intersection point being deterministic, the outgoing ray direction is completely determined. In function-space sampling, the normal at an intersection is not determined ahead of time. Instead, we sample it from a distribution conditioned on the realization values we saw along the incoming ray. This realization only fixes the directional derivative of the GP at the intersection point but leaves us two additional degrees of freedom to choose a gradient and, hence, a normal. Assuming non-degenerate covariance kernels (i.e. volume-type GPISes, that do not assume perfect correlation along any axis), the resulting gradient distribution is non-degenerate and assigns some density to the whole hemisphere of normals facing the ray. Hence, even when using a mirror micro-BSDF, we can always find a normal such that the reflected ray points into the desired NEE direction. Computing the density of this distribution for a given normal is not trivial. The conditioned distribution of gradients is a 3D multivariate Gaussian distribution (no matter how complex the conditioning is). To compute the (unnormalized) density of sampling a given normal, we simply need to integrate over all possible gradients that normalize to that normal as

\[ p(n) = \int_0^\infty f(n + t) \, dt, \quad (27) \]

where \( f \) is the pdf of the conditioned gradient distribution at the intersection point. Our preliminary investigation has shown that the integral in Eq. (27) is tractable analytically, and we should be able to compute the normalization constant to turn \( p(n) \) into a true pdf. This would then allow us to evaluate the conditioned visible normal distribution, enabling (single scattering) NEE for mirror micro-BSDFs. We leave this for future work but still apply classical NEE after sampling a normal when the micro-BSDF is diffuse.

3.4 Implementation

We implemented the sampling strategies described in this section in the Tungsten renderer [Bitterli 2018] with a focus on correctness over performance. A GPIS is treated as a participating medium that allows for sampling free-flight distances and computing transmittance, and provides a phase function at scattering locations. Light and camera positions are uncorrelated from the GPIS and we often place uncorrelated, non-GPIS surfaces in the scene alongside the GPIS we are investigating. We could, of course, represent these as uncorrelated GPIS as well, but that would unnecessarily increase rendering times without aiding in the validation or understanding of our method. We use standard data structures such as OpenVDB grids to store volumetric scene data (variance, length scale, and mean). For the mean, we also support using a mesh directly and computing SDFs on the fly.

4 UNCERTAINTY QUANTIFICATION USING DOUBLE MONTE CARLO SAMPLING

We have a model \( M^\Phi(x) \) with uncertain parameters \( \Phi \sim p_\Phi \). We would like to compute the moments of the distribution of model outputs \( q_x(y_x = M^\Phi(x)) \). This problem is known as uncertainty quantification. If we have access to a deterministic method of evaluation \( M^\Phi(x) \), we can compute

\[
\mathbb{E}_{\Phi \sim p_\Phi} \left[ M^\Phi(x) \right] = \int M^\Phi(x) \, dp_\Phi(\Phi) 
\]

\[
\forall \Phi \sim p_\Phi \quad \left[ M^\Phi(x) \right] - \mathbb{E}_{\Phi \sim p_\Phi} \left[ M^\Phi(x) \right]^2 = \int M^\Phi(x)^2 \, dp_\Phi(\Phi) - \left( \int M^\Phi(x) \, dp_\Phi(\Phi) \right)^2
\]

and use Monte Carlo integration and sample variance to compute an estimate of the mean and variance, respectively as

\[
\mathbb{E}_{\Phi \sim p_\Phi} \left[ M^\Phi(x) \right]_N = \frac{1}{N} \sum_{i=0}^{N} M^\Phi(x) \quad \Phi_i \sim p_\Phi
\]

\[
\forall \Phi \sim p_\Phi \quad \left[ M^\Phi(x) \right]_N = \frac{1}{N} \sum_{i=0}^{N} M^\Phi(x)^2 - \frac{1}{N} \sum_{i=0}^{N} M^\Phi(x)^2 \quad \Phi_i \sim p_\Phi
\]

Note that due to the square in the second term and Jensen’s inequality, we need a relatively large number of samples to get an unbiased estimate of variance. In practice, this is not a big issue since we tend to have the samples available anyway if we want to compute a relatively converged estimate of the mean. That is, if we have enough samples to estimate the mean, we tend also to have enough samples to estimate variance.

Unfortunately, in many graphics applications like rendering, the model itself is often in the form of a complex integral equation, such as

\[
M^\Phi(x) = \int m^\Phi(x, y) \, dy
\]

and has to be estimated using Monte Carlo techniques. That is, we only have access to an estimate

\[
\overline{M^\Phi(x)}_N = \frac{1}{N} \sum_{i=0}^{N} m^\Phi(x, y_i) \quad y_i \sim p_y.
\]

This is not an issue when computing the mean of our model predictions. Since a one-sample Monte Carlo estimator is unbiased, that is

\[
M^\Phi(x) = \mathbb{E} \left[ \overline{M^\Phi(x)}_1 \right] = \mathbb{E}_{y \sim p_y} \left[ m^\Phi(x, y) \right] \frac{1}{p_y(y)}
\]

we can write

\[
\mathbb{E}_{\Phi \sim p_\Phi} \left[ M^\Phi(x) \right] = \mathbb{E}_{\Phi \sim p_\Phi} \left[ \mathbb{E}_{y \sim p_y} \left[ m^\Phi(x, y) \right] \frac{1}{p_y(y)} \right] \quad \Phi \sim p_\Phi
\]

\[
= \mathbb{E}_{\Phi \sim p_\Phi, y \sim p_y} \left[ m^\Phi(x, y) \right] \frac{1}{p_y(y)}
\]
and then use Monte Carlo integration to estimate the expectation as

\[ \mathbb{E}_{\Phi \sim p_{\Phi}, y \sim p_{y}} \left[ \frac{m_{\Phi}(x, y)}{p_{y}(y)} \right] \approx \frac{1}{N} \sum_{i=0}^{N} m_{\Phi_{i}}(x, y_{i}) \Phi_{i} \sim p_{\Phi}, y_{i} \sim p_{y} \]

(39)

Note that even though we are sampling two random variables now, we can still just average \( N \) independent evaluations of \( m_{\Phi_{i}}(x, y_{i}) \). This is one of the central benefits of Monte Carlo integration; its convergence does not depend on the dimensionality of the problem.

But for the variance, we have

\[ \forall_{\Phi \sim p_{\Phi}, y \sim p_{y}} \left[ \frac{1}{N} \sum_{i=0}^{N} m_{\Phi_{i}}(x, y_{i}) \right] \neq \forall_{\Phi \sim p_{\Phi}, y \sim p_{y}} \left[ \frac{m_{\Phi}(x, y)}{p_{y}(y)} \right] \]

(40)

Intuitively, this is because \( \forall \left[ \frac{M_{\Phi}(x)}{p_{y}(y)} \right] \neq 0 \) for \( N < \infty \) and we have, according to the law of total variance

\[ \forall_{\Phi \sim p_{\Phi}, y \sim p_{y}} \left[ M_{\Phi}(x) \right] = \mathbb{E}_{y_{1}, \ldots, y_{N} \sim p_{y}} \left[ \forall_{\Phi \sim p_{\Phi}} \left[ \frac{M_{\Phi}(x)}{y_{1}, \ldots, y_{N}} \right] \right] \]

+ \[ \forall_{y_{1}, \ldots, y_{N} \sim p_{y}} \left[ \mathbb{E}_{\Phi \sim p_{\Phi}} \left[ M_{\Phi}(x) \right] \right] \].

(41)

Here, we can see that for finite \( N \), the total variance is comprised of the expected value of the variance of our model due to model parameters and the variance due to the Monte Carlo estimation of our model.

We only recover the ground variance of the model due to the model parameters as \( N \to \infty \):

\[ \lim_{N \to \infty} \mathbb{E}_{y_{1}, \ldots, y_{N} \sim p_{y}} \left[ \forall_{\Phi \sim p_{\Phi}} \left[ \frac{M_{\Phi}(x)}{y_{1}, \ldots, y_{N}} \right] \right] = \forall_{\Phi \sim p_{\Phi}} \left[ M_{\Phi}(x) \right] \]

(42)

\[ \lim_{N \to \infty} \forall_{y_{1}, \ldots, y_{N} \sim p_{y}} \left[ \mathbb{E}_{\Phi \sim p_{\Phi}} \left[ M_{\Phi}(x) \right] \right] = 0 \]

(43)

Intuitively, now that means that if we want to compute the sample variance of our model due to the model parameters unaffected by variance due to Monte Carlo estimation of the model, we need to sample many \( y_{j} \sim p_{y} \) for each \( \Phi_{i} \sim p_{\Phi} \). Even if we re-use the same set of \( y_{s} \), we still need to evaluate \( M_{\Phi_{i}}(x) \) separately for each \( \Phi_{i} \) and compute sample variance as

\[ \forall_{\Phi \sim p_{\Phi}} \left[ \frac{M_{\Phi}(x)}{N,K} \right] = \frac{1}{N} \sum_{i=0}^{N} \left( \frac{1}{K} \sum_{j=0}^{K} \frac{m_{\Phi_{i}}(x, y_{j})}{p_{y}(y_{j})} \right)^{2} \]

\[- \left( \frac{1}{NK} \sum_{i=0}^{N} \sum_{j=0}^{K} \frac{m_{\Phi_{i}}(x, y_{j})}{p_{y}(y_{j})} \right)^{2} \Phi_{i} \sim p_{\Phi}, y_{j} \sim p_{y} \].

(44)

This now results in quadratic complexity \( O(NK) \) where \( N \) controls the bias due to the limited number of parameter samples and \( K \) controls the bias due to the additional variance in the Monte Carlo estimator. Hence, while it is certainly possible to do uncertainty quantification via Monte Carlo rendering of GPISes, it is not trivial to do so efficiently and we leave this for future work.
Fig. 3. We show an overview of some of the kernels that we implemented for our method. To give an intuition for the family of GPISes each kernel produces, we show samples from the prior (produced via both function-space and, where applicable, weight-space methods) and posterior occupancy.