Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2010)
M. Otaduy and Z. Popovic (Editors)

A Bayesian Interactive Optimization Approach to Procedural Animation Design

Eric Brochu Tyson Brochu Nando de Freitas

University of British Columbia

Abstract

The computer graphics and animation fields are filled with applications that require the setting of tricky parameters. In many cases, the models are complex and the parameters unintuitive for non-experts. In this paper, we present an optimization method for setting parameters of a procedural fluid animation system by showing the user examples of different parametrized animations and asking for feedback. Our method employs the Bayesian technique of bringing in "prior" belief based on previous runs of the system and/or expert knowledge, to assist users in finding good parameter settings in as few steps as possible. To do this, we introduce novel extensions to Bayesian optimization, which permit effective learning for parameter-based procedural animation applications. We show that even when users are trying to find a variety of different target animations, the system can learn and improve. We demonstrate the effectiveness of our method compared to related active learning methods. We also present a working application for assisting animators in the challenging task of designing curl-based velocity fields, even with minimal domain knowledge other than identifying when a simulation "looks right".

Categories and Subject Descriptors (according to ACM CCS): Learning [I.2.6]: Parameter Learning.—User Interfaces [H.5.2]: Interaction Styles.—Three-Dimensional Graphics and Realism [I.3.7]: Animation.—

1 Introduction

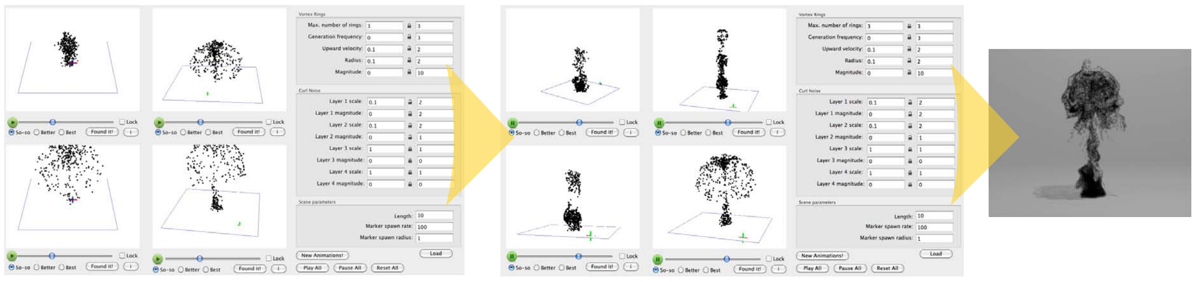
Procedural methods for generating animation have long been used by visual effects and games studios due to their efficiency and artist controllability. However, this control comes with a cost: a set of often unintuitive parameters confronts the user of a procedural animation system. The desired end result is often identifiable by the user, but these parameters must be tuned in a tedious trial-and-error process.

For example, realistic animation of smoke can be achieved by driving a particle system through a simple combination of vortex rings and curl noise [BHN07]. However even these two relatively simple procedural methods are influenced by several parameters: The velocity, radius and magnitude of the vortex rings, and the length scale and magnitude of the curl noise. Adding more procedural "flow primitives", such as uniform and vortical flows, sources and sinks [WH91], turbulent wind [SF93], vortex particles [SRF05], and vortex filaments [AN05] can produce a wider variety of animations, but each of these primitives carries its own set of associated parameters. These parameters can interact in subtle and non-intuitive ways, and small adjustments to certain settings may result in non-uniform changes in the appearance.

Brochu *et al.* [BGdF07, BdFG07] propose a Bayesian optimization technique to assist artists with parameter tuning for bidirectional reflectance distribution functions (*BRDF*s). In their iterative scheme, the algorithm selects two sets of parameters and generates example images from them. The user selects the preferred image and the algorithm incorporates this feedback to learn a model of the user's *valuation* function over the domain of parameter values. Given this valuation function, the algorithm is able to select parameters to generate simulations that are likely to be closer to the ones wanted by the artist. The process is repeated until the user is satisfied with the results.

During the development of a procedural smoke animation system, we found ourselves with a parameterized system with 12 continuous parameters. Setting these was a challenge for the developers, let alone other users, so we looked to adapt [BdFG07]. In the process, though, we found that the model as presented was unsuitable for our procedural animation. In particular, we identified several limitations:

DOI: 10.2312/SCA/SCA10/103-112



Vortex Rings

Max. number of rings:	3	🔒	3
Generation frequency:	0	🔒	3
Upward velocity:	0.1	🔒	2
Radius:	0.1	🔒	2
Magnitude:	0	🔒	10

Curl Noise

Layer 1 scale:	0.1	🔒	2
Layer 1 magnitude:	0	🔒	2
Layer 2 scale:	0.1	🔒	2
Layer 2 magnitude:	0	🔒	1
Layer 3 scale:	1	🔒	1
Layer 3 magnitude:	0	🔒	0
Layer 4 scale:	1	🔒	1
Layer 4 magnitude:	0	🔒	0

Scene parameters

Length:	10
Marker spawn rate:	100
Marker spawn radius:	1

Vortex Rings

Max. number of rings:	3	🔒	3
Generation frequency:	0	🔒	3
Upward velocity:	0.1	🔒	2
Radius:	0.1	🔒	2
Magnitude:	0	🔒	10

Curl Noise

Layer 1 scale:	0.1	🔒	2
Layer 1 magnitude:	0	🔒	2
Layer 2 scale:	0.1	🔒	2
Layer 2 magnitude:	0	🔒	1
Layer 3 scale:	1	🔒	1
Layer 3 magnitude:	0	🔒	0
Layer 4 scale:	1	🔒	1
Layer 4 magnitude:	0	🔒	0

Scene parameters

Length:	10
Marker spawn rate:	100
Marker spawn radius:	1

Figure 1: *The animation gallery in action. An animator is searching the space of possible parameters to find an animation. He or she provides preferences over a gallery of animations created procedurally from different parameter settings. The system uses these preferences to automatically learn a model of what the user is searching for and to propose a new set of animations. The new gallery automatically balances the exploration-exploitation tradeoff between finding improvements on animations the user likes and exploring new settings. It also incorporates user constraints about what parameters or ranges to search. When acceptable parameters are found, a higher-quality animation can be generated offline. As the tool is used, it automatically learns the properties of the parameter space that generate good animations — the more it is used, the better able it is to guide new users.*

- The settings of the kernel hyperparameters, which control the smoothness of the objective function on each of the parameters, has a very large impact on the quality of the optimization. [BdFG07] has an expert set these, but our best attempts to do so were simply not good enough. However, approaches in the literature for automatically setting these values require far more data than we have available.
- Many regions of the parameter space (combinations of parameter ranges) never produce good animations. However, under the zero-mean Bayesian optimization model, until the user supplies evidence, parameter settings in these regions are no less likely to be selected than ones that are frequently chosen by users. Particularly with the relatively large number of parameters we use, this requires users to repeatedly view and rate uninteresting settings until evidence is accrued, instead of first focussing on the most promising areas.
- The earlier model does not offer a way of incorporating user expertise. After even a small amount of time spent using the system, users had developed a good enough semantic understanding of certain parameters that they wanted to either set parameters to a specific value, or restrict the optimization to a user-defined region.
- The pairwise interface, in which two instances are shown side-by-side, seemed inefficient to users who felt they would rather select animations from a larger gallery of instances.

With these shortcomings in mind, we extended the Bayesian optimization approach with several new features, which we incorporate into a novel application for assisting animators (Figure 1). The interface allows users to view four animations simultaneously, and to manually set ranges of parameters (including setting them to a single value). Our primary improvement, however, is a new hierarchical Bayesian method for incorporating data from previous users of the system to adaptively tune model hyperparameters. These hyperparameters control aspects such as the relative importance of changing settings of the different parameters, and the belief, before evidence is added, that an area of the space is unlikely to generate good animations. The problem of learning hyperparameters is usually difficult or impossible for small data sets, such as the ones generated in a single user session. We get around this by treating the problem as one of tracking the distributions of the hyperparameters across user sessions. As different users employ the system to find different animations, the system automatically improves and adapts.

In Section 2, we present the Bayesian optimization model. In Section 3, we introduce novel extensions in which the hyperparameters and mean function of the model are automatically learned between runs, with data from different users constantly added. In Section 4, we employ this learning model in an animation design gallery application that allows artists and non-expert users to find desired animation parameters without any knowledge of the underlying model. We present experimental results in Section 5, on synthetic functions and real users, and offer conclusions in Section 6.

1.1 Related work

In the computer graphics literature, the work of Talton *et al.* [TGY*09] is probably most similar to ours. They introduce a collaborative system which uses data from a body of users to learn spatially-varying parameters, though their work is still quite distinct from ours. Their model is based on density estimation in high-dimensional spaces, whereas we are interested in optimizing individual user valuation. Their approach is also intended as a novel interface to aid users who are unfamiliar with the system, while our approach is intended to work in conjunction with more traditional slider manipulation approaches to finding parameters. The *Design*

Gallery [MAB*97] interface for animation and the gallery navigation interface for reflectance functions [NDM06] are other artist-assistance tools with some similarity to ours. They both use non-adaptive heuristics to find the set of input parameters to be used in the generation of the display. We depart from this heuristic treatment and instead present a principled probabilistic decision making approach to model the design process, which can be studied using the existing tools of machine learning and optimization, and adapted to a variety of design scenarios. Psychoperceptual preference elicitation has been previously done in graphics in the context of image-based rendering [KYJF04], as well as evaluation of tone-mapping operators for high dynamic range (HDR) images [LCTS05].

2 Bayesian optimization for preference galleries

The process of tweaking parameters to find a result that looks "right" can be seen as akin to sampling a perceptual objective function and searching the parameter space to find the best result. This process is, in essence, solving an optimization problem. Our objective function is the psychoperceptual process underlying judgment — how well a realization fits what the user has in mind. In the econometrics domain, this is often called the *valuation* function.

In the case of a person rating the suitability of an animation (a procedurally-generated fluid animation in our case), each sample of valuation involves creating an animation with the given parameters and asking a human to provide feedback, which is interpreted as the function response. This is a very expensive class of functions to evaluate! Furthermore, it is in general impossible to even sample the function directly and get a consistent response from users. Asking humans to rate an animation on a numerical scale has built-in problems — not only will scales vary from user to user, but human evaluation is subject to phenomena such as *drift*, where the scale varies over time, *anchoring*, in which early experiences dominate the scale [PBJ93]. Some of these problems can be alleviated by asking users to select binary preferences between instances [McF01], but this comes with its own costs and limitations. In our work, we propose the use of more flexible preference galleries to obtain feedback from the animators. The interface will be described in Section 4.2. To present the models and algorithms, which is the focus of the current section, it suffices to say that this interface enables us to gather the user's preferences in the form of discrete choices.

More formally, the dataset gathered using the gallery approach consists of ranked pairs $\mathcal{D} = \{\mathbf{r}_k \succ \mathbf{c}_k;\ \ k = 1,\ldots,M\}$, where the symbol $\succ$ indicates that the user prefers $\mathbf{r}$ to $\mathbf{c}$. Assuming that the user has already provided some feedback, we use $\mathbf{x}_{1:N} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, $\mathbf{x}_i \in \mathcal{X} \subseteq \mathbb{R}^d$, to denote the N elements in the training data. Each element $\mathbf{x}_n$ is a specific parameter choice, so that $\mathbf{r}_k$ and $\mathbf{c}_k$ correspond to two elements of $\mathbf{x}_{1:N}$. Our goal is to predict the next item $\mathbf{x}$ which we believe will have the highest user valuation $u(\mathbf{x})$ in as few comparisons as possible.

Our probabilistic model for specifying preferences in terms of the latent valuation function $u(\cdot)$ is the one advocated by [CG05]. It is a classical model for relating binary observations to a continuous latent function, which is known as the Thurstone-Mosteller law of comparative judgment [Thu27, Mos51]. For completeness, we review it briefly here. We model the valuation functions $u(\cdot)$ for $\mathbf{r}$ and $\mathbf{c}$ as $u(\mathbf{r}_k) = f(\mathbf{r}_k) + e_{rk}$, $u(\mathbf{c}_k) = f(\mathbf{c}_k) + e_{ck}$ where the noise terms are Gaussian: $e_{rk}, e_{ck} \sim \mathcal{N}(0, \sigma^2)$. Under this utility model, the likelihood that item $\mathbf{r}$ is preferred to item $\mathbf{c}$ is given by:

$$P(\mathbf{r}_k \succ \mathbf{c}_k) = P(u(\mathbf{r}_k) > u(\mathbf{c}_k)) = \Phi(d_k)\,,$$

where $d_k = \frac{f(\mathbf{r}_k) - f(\mathbf{c}_k)}{\sqrt{2}\sigma}$, and Φ is the cumulative function of the standard Normal distribution.

In order to estimate the posterior distribution of the latent function $p(\mathbf{f}|\mathcal{D})$, which is the optimal Bayesian estimate of the unknown function $f(\cdot)$, we also need to specify a prior on $f(\cdot)$. [BdFG07] assign a nonparametric Gaussian process prior to the unknown mean valuation: $f(\cdot) \sim GP(0, K(\cdot,\cdot))$. That is, at the N training points:

$$p(\mathbf{f}) = |2\pi\mathbf{K}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\mathbf{f}^T\mathbf{K}^{-1}\mathbf{f}\right),$$

where $\mathbf{f} = \{f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_N)\}$ and the symmetric positive definite covariance $\mathbf{K}$ has entries (kernels) $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = \exp(\theta\|\mathbf{x}_i - \mathbf{x}_j\|)$. With reference to the hyperparameter θ, their paper states: "*Initially we learned these parameters via maximum likelihood, but soon realized that this was unsound due to the scarcity of data. To remedy this, we elected to use subjective priors using simple heuristics, such as expected dataset spread.*" In this respect, we depart significantly from their approach. Here, we will assume a hierarchical nonparametric prior on $f(\cdot)$. The hyperparameters of this prior will be learned by taking into consideration multiple animation trials. This construction will be discussed in detail in Section 3.

Aside from the prior construction and the use of preference galleries, we follow the approach of [BdFG07] for Bayesian optimization. We therefore refer the interested reader to that paper for details and focus on a high-level description of the approach. The first step is to compute the *posterior distribution* via Bayes rule:

$$p(\mathbf{f}|\mathcal{D}) \propto p(\mathbf{f}) \prod_{k=1}^{M} p(d_k|\mathbf{f}). \tag{1}$$

The posterior is not analytical, but can approximated using Laplace's method, expectation propagation or Monte Carlo [RW06]. After experimenting with Laplace's method [CG05], we ended up choosing a simpler alternative, which we found to be more numerically efficient, stable, easier to code, and which delivers similar performance in our preference gallery setting: We simply approximate $p(\mathbf{f}|\mathcal{D})$ with a Dirac-delta mass at its mode $\mathbf{f}^{MAP}$. The mode is easily found by calculating the derivative of the log-posterior of equation (1).

For any possible instance $\mathbf{x}_{n+1}$, the *predictive distribution* $p(f(\mathbf{x}_{n+1})|\mathcal{D})$ can be easily obtained using the matrix inversion lemma [RW06]. This distribution has the following mean and variance:

$$\begin{aligned}\mu(\mathbf{x}_{n+1}) &= \mathbf{k}^T\mathbf{K}^{-1}\mathbf{f}^{MAP}\\ s^2(\mathbf{x}_{n+1}) &= k(\mathbf{x}_{n+1},\mathbf{x}_{n+1})-\mathbf{k}^T\mathbf{K}^{-1}\mathbf{k},\end{aligned}$$

where $\mathbf{k}^T=[k(\mathbf{x}_{n+1},\mathbf{x}_1)\cdots k(\mathbf{x}_{n+1},\mathbf{x}_N)]$. As we will see in Section 3, our prior will give rise to a slightly different predictive distribution.

The last step of the Bayesian optimization approach is to use a statistical measure of improvement to decide where to sample next. That is, after computing the predictive distribution, we need to decide what simulations to show the animator. The predictive distribution will enable us to balance the tradeoff of exploiting and exploring in this process. When exploring, we choose points where the predicted variance is large. When exploiting, we choose points where the predicted mean is large (high valuation). This tradeoff between exploration and exploitation is balanced with a classical measure of expected improvement [JSW98]. Let $\mathbf{x}^+$ denote the point of the highest estimate of the predictive distribution thus far. That is, $\mu(\mathbf{x}^+)$ is the highest valuation for the data provided by the individual. For simplicity, we will use $\mathbf{x}$ to indicate $\mathbf{x}_{n+1}$. [JSW98] define the improvement over the current best point as $I(\mathbf{x})=\max\{0,\mu(\mathbf{x})-\mu(\mathbf{x}^+)\}$. This results in an expected improvement of

$$EI(\mathbf{x})=\begin{cases}(\mu(\mathbf{x}^+)-\mu(\mathbf{x}))\Phi(d)+s(\mathbf{x})\phi(d) & \text{if } s>0\\ 0 & \text{if } s=0\end{cases}$$

where ϕ is the standard Normal density and $d=\frac{\mu(\mathbf{x}^+)-\mu(\mathbf{x})}{s(\mathbf{x})}$.

To find the point at which to sample, we still need to maximize the constrained objective $EI(\mathbf{x})$. *Unlike the original unknown objective function, $EI(\cdot)$ can be cheaply sampled*, so we can use conventional optimizers. We use *DIRECT* [JPS93], a deterministic, derivative-free optimizer. The overall approach is summarized in Algorithm 1.

Algorithm 1 Bayesian optimization for animation galleries

```
1: Let $n = 0$ and $NG$ be the number of instances in the gallery
   interface, and choose an initial set of parameters, $\mathbf{x}_{1:NG}$.
2: repeat
3:    Generate gallery of animation instances from the parameters.
4:    Record $k$ user preferences $\{\mathbf{r}_{1:k},\mathbf{c}_{1:k}\}$ from the set
      $\{\mathbf{x}_{n+1:n+NG}\}$ and add to $\mathcal{D}$.
5:    Compute the predictive distribution $\mu(\cdot), s^2(\cdot)$.
6:    Let $n = n + NG$.
7:    Compute a new set of $NG$ parameters $\{\mathbf{x}_{n+1:n+NG}\}$ by iter-
      atively maximizing the expected improvement function.
8: until Animator is satisfied
```

3 Learning from multiple sessions

Unlike traditional approaches to Bayesian optimization, we take into account the fact that the process of optimization is repeated many times either by the same animator or by different animators. Our central hypothesis is that the interests of animators are regular. That is, there are only so many kinds of smoke that people care about when producing animations. The standard zero-mean Gaussian process (GP) prior fails to capture this. In our approach, on the other hand, we use the results of previous sessions (optimizations) to learn the hyperparameters of a semi-parametric prior for the current trial. This is crucial to both reduce the number of user interactions and to scale the method to higher dimensions.

In the remainder of this paper, we use *iteration* to refer to a single cycle of Bayesian optimization — maximizing the EI to find a label or preference candidates, getting labels of preferences, and updating the model. *Session* refers to an entire run of the optimization, from zero data until a termination condition is reached. *Instance* refers to an animation generated from a specific set of parameters.

We adopt the following semi-parametric Gaussian process prior:

$$f(\cdot)\sim GP(\mathbf{m}(\cdot),K(\cdot,\cdot)). \tag{2}$$

In this model, the mean $\mathbf{m}(\cdot)$ is represented with radial basis functions (RBFs) with centers $\mathbf{c}$ and coefficients α:

$$\mathbf{m}(\mathbf{x})=\mathbf{B}(\mathbf{x},\mathbf{c})\alpha=\sum_{j=1}^{p}g(\|\mathbf{x}-\mathbf{c}_j\|)\alpha_j.$$

For the bases $g(\cdot)$, we adopt the typical choices: splines, Gaussians, multi-quadrics and so on. We refer the reader to, for example, [Bis06] for an introduction to RBF approximation.

In addition, we adopt a more sophisticated representation for the kernels known as automatic relevance determination (ARD) [RW06]. The actual expression is:

$$k(\mathbf{x}_i,\mathbf{x}_j)=\exp\left(\sum_{\ell}\frac{\|x_{i,\ell}-x_{j,\ell}\|}{\theta_\ell^2}\right)+\sigma^2\delta_{ij},$$

where ℓ is an index over the components of the vector $\mathbf{x}_i$ and δ_{ij} is the Kronecker-Delta function. Using an ARD kernel is critical, as it allows the smoothness to be determined independently along each dimension. Intuitively, if a particular θ_ℓ has a small value, the kernel becomes independent of the ℓ-th input, effectively removing it automatically. Hence, irrelevant dimensions are discarded.

We propose to learn the RBF parameters $(\mathbf{c},\alpha)$ as well as the kernel parameters (θ,σ) using data gathered from previous optimizations. Note that the conventional approach to setting these is to maximize the log-likelihood of the model within a specific optimization run. This method works well for many application of GPs, but unfortunately for our application, it is known to work poorly when the number of training data is small (exactly the situation we are in). The sparsity of data in the space can lead to the likelihood function becoming very flat on some dimensions, or even monotonically increasing to infinity. This can lead to low-quality models or ill-conditioned covariance matrices. Previous methods have dealt with this by initializing the data set with a large number of random samples from a Latin hypercube [SWN03], or having a system expert select values for the hyperparameters or hyperposteriors [Liz08], neither of which is suitable to our application.

Our insight is that *every time the application is used, a related model is trained*. While different runs might involve users with different simulation goals, if we record the final user data $\mathcal{D}$ of each run, this can be used to generate a distribution over the hyperparameters, which identifies some hyperparameter settings as more likely than others, even before a new user starts using the system. This distribution can be tracked across user sessions, so that as the system is used in different ways, the distribution will come to reflect that fact. It is important to emphasize that this does not directly affect the *actual* parameters of the animation — that is determined by the user's feedback.

We estimate the locations of the basis centers $\widehat{\mathbf{c}}$ by clustering the parameters of previous simulations using k-means. Intuitively, the resulting clusters capture the different types of smoke that animators are typically interested in. By conditioning on $\widehat{\mathbf{c}}$, the coefficients α can be obtained analytically using standard Bayesian conjugate analysis. Specifically, we place a Gaussian hyper-prior on them: $\alpha \sim \mathcal{N}(0, \delta^2)$ with regularizer δ^2. We use Bayes rule to combine this prior with the Gaussian model of equation (2) evaluated at the training data, to obtain the following estimate of α:

$$\widehat{\alpha} = \left(\mathbf{B}^T\mathbf{K}^{-1}\mathbf{B} + \delta^{-2}\mathbf{I}_p\right)^{-1}\mathbf{B}^T\mathbf{K}^{-1}\mathbf{f}^{MAP},$$

where $\mathbf{B}$ denotes an $N \times p$ matrix of bases functions evaluated at the N training data and p locations $\widehat{\mathbf{c}}$. By approximating the posterior of α with a Dirac-delta mass at $\widehat{\alpha}$, the mean and variance of the predictive distribution become:

$$\begin{aligned}\mu(\mathbf{x}) &= \mathbf{B}(\mathbf{x},\widehat{\mathbf{c}})\widehat{\alpha} + \mathbf{k}^T\mathbf{K}^{-1}\left(\mathbf{f}^{MAP} - \mathbf{B}^T\widehat{\alpha}\right)\\ s^2(\mathbf{x}) &= k(\mathbf{x},\mathbf{x}) - \mathbf{k}^T\mathbf{K}^{-1}\mathbf{k}.\end{aligned}$$

We now turn our attention to the process of inferring the kernel parameters θ and σ. In our user application, these have intuitive interpretations with regard to the impact on the user — θ is the relationship between distance in the parameter space and the valuation function, and σ is the noise or uncertainty associated with the user's ratings. We will refer to these as the model *hyperparameters*, with the generic symbol β.

We can model the sequence of optimizations conducted by the same animator or different animators using a dynamical state space model [DdFG01]. In this dynamic model, the hyperparameters β correspond to the unknown states. The state space model consists of an initial belief $p(\beta_0)$, a stochastic evolution process $p(\beta_t|\beta_{t-1})$ and an observation component $p(\mathbf{f}_t|\beta_t)$. To be precise, the observation $\mathbf{f}_t$ is in fact the maximum a posteriori estimate $\mathbf{f}^{MAP}$ derived from the user feedback as outlined in the previous section, but we drop the *MAP* superscript to simplify the notation. Our goal is then to compute the optimal filtering distribution $p(\beta_t|\mathbf{f}_{1:t})$ given that t runs of the application have taken place. This distribution is intractable, but may be approximated with a Monte Carlo histogram estimator with samples obtained using a *particle filter* [DdFG01], as illustrated in Algorithm 2.

Algorithm 2 Particle Filter for Hyperparameter Learning

```
1: For $i = 1,\ldots,N$ sample $\beta_0^{(i)} \sim p(\beta_0)$.
2: Let $t = 1$.
3: while True do
4:     For $i = 1,\ldots,N$ sample $\widetilde{\beta}_t^{(i)} \sim p(\beta_t|\beta_{t-1}^{(i)})$
5:     For $i = 1,\ldots,N$ evaluate importance weights $\widetilde{w}_t^{(i)} = p(\mathbf{f}_t|\widetilde{\beta}_t^{(i)})$ and normalize them.
6:     Resample with replacement $N$ particles $(\beta_{0:t}^{(i)}, i = 1,\ldots,N)$ from the set $(\widetilde{\beta}_{0:t}^{(i)}, i = 1,\ldots,N)$ according to the importance weights $\widetilde{w}_t^{(i)}$.
7:     t = t + 1
8: end while
```

We treat the hyperparameters as independent, and use a separate dynamic Gaussian diffusion model for each hyperparameter. (This is done for efficiency and interpretability — we could theoretically model the hyperparameters using a single particle filter if necessary.) The dynamic diffusion model allows the hyperparameters to converge to values that exhibit some random drift over time. We don't eliminate this drift intentionally as we believe it is useful to have a nonstationary model component in our application domain.

At the beginning of each user session, we set the kernel hyperparameters to the particle filter means estimated from previous sessions. After each user session, we use the inferred function $\mathbf{f}$ to compute the fitness of each particle according to the following non-linear Gaussian observation model:

$$p(\mathbf{f}_t|\beta_t) = |2\pi\mathbf{K}(\beta)|^{-1/2}\exp\left(-\frac{1}{2}\mathbf{f}^T\mathbf{K}(\beta)^{-1}\mathbf{f}\right).$$

In this expression, we have emphasized the dependency of $\mathbf{K}$ on the kernel hyperparameters β for clarity.

Algorithm 2 shows the particle filter method of updating for hyperparameters β. This is a simple particle filter, where the importance sampling proposal is the transition prior $p(\beta_t|\beta_{t-1})$. If one has prior knowledge about the hyperparameters, it is possible to incorporate this into the design of more sophisticated proposal distributions.

4 A Bayesian design gallery for procedural animation

We apply the Bayesian optimization model as the learning engine for our procedural animation design tool. Our contribution to the design problem is a "gallery" approach in which users can view several animations generated from multiple parameters, and provide feedback in the form of real-valued ratings indicating how close an animation is to what they are looking for. In practice, the first few examples presented to the user will be points of high uncertainty, since little of the space is explored (that is, the model is very uncertain about the user's valuation criteria). Later galleries include more examples of high predicted valuation, as a model of the user's interest is learned. [BdFG07] presented an interface based on a pairwise gallery of two images, where the user indicates simple preference, and showed this was a superior approach to directly rating instances. However, other work in graphics and animation, such as the design gallery of [MAB*97] has used galleries of multiple instances. We introduce and study a gallery of multiple instances on which preferences can be defined, and investigate the effect of adopting a 2-panel or a 4-panel gallery.

4.1 Procedural animation

While the focus of this paper is on novel Bayesian optimization techniques for learning model parameters, we were motivated by a specific animation problem. As the gallery interface and later experiments use this problem, we present our procedural animation method here.

We produce smoke animation by driving a set of passive marker particles through a procedurally-generated velocity field. This velocity field is generated by taking the curl of a (vector-valued) potential function, which automatically ensures that the resulting velocity field is divergence-free, an important characteristic of fluid motion. There are two main components to this potential field, which we linearly combine: the contribution due to a set of vortex rings, and a spatially varying noise function.

The potential function of a vortex ring perpendicular to the y axis with center $\mathbf{h}$, radius r, at a point in space $\mathbf{z}$ is given by

$$\psi_v(\mathbf{z}) = \frac{1}{(r-d)^2 + 2rd} < \mathbf{z}_3, 0, -\mathbf{z}_1 >,$$

where $d = \|\mathbf{z} - \mathbf{h}\|$. The potential function associated with curl noise [BHN07] is a spatially and temporally continuous noise function $\psi_n(\mathbf{z}) = \mathbf{n}(\mathbf{z}, l)$, where l is the length scale of noise. The velocity field is then the curl of the linear combination of these two potential fields:

$$\mathbf{v}(\mathbf{z}) = \nabla \times (\Gamma\psi_v(\mathbf{z}) + \omega\psi_n(\mathbf{z})).$$

This simple model results in at least four parameters which must be tuned: the radius of a vortex ring, r, the length-scale of the noise function, l, and the relative strengths of each potential function, Γ and ω. Additionally, our examples use vortex rings which move upward with some velocity, and are generated at the origin with some frequency, resulting in two additional parameters per ring. We model a total of four distinct curl noise layers, for a total of 8 parameters, though the use of the curl noise layers is not required for all animations, and they can be disabled by setting both parameters to 0. Since this method is procedural, and not a simulation, the variety of animations capable of production is fundamentally limited, though still quite large.

4.2 Gallery

The gallery interface (Figure 1) is our user-facing parameter-optimization tool. Four animations are shown at a time, selected using Schonlau's method [SWJ98] of simulating updates to the GP by iteratively maximizing the EI and updating the covariance matrix for $\max EI$ (since in a GP, the covariance is independent of observations). At any stage, the user can set the parameters to a fixed value or change the range, which directly sets the bounds of the optimization of the EI function. This permits the user to set up useful workflows. For example, users can start with several free parameters and view examples until they find one similar to their target and fix most of the parameters, using the model to help set one or two "tricky" remaining values. Alternatively, the user can adjust parameters until they reach a point where they are frustrated with one or more and then use the system to help find it. In any case, the goal is not to remove or restrict the process of manually setting parameters, but to augment it.

Animations are generated using the procedural system described above, during a non-interactive "animation" phase. At each frame of the animation, the flow primitives are updated, and new ones are spawned if necessary. New particles are spawned at a source, advected according to the set of flow primitives, and all particle positions for a given frame are written to disk. The animation can then be previewed in an OpenGL window by streaming the particle data from disk. After each run of the application, the final data vector $\mathcal{D}_{1:n}$ for the run is logged and the RBF parameters and distributions of the hyperparameters $(\mathbf{c}, \alpha, \theta, \sigma)$ are updated (Section 3). The user has the option to skip this step if desired.

5 Experiments

The Bayesian optimization approach has been found to be very efficient for expensive objective functions [JSW98, Sas02, Liz08], which suggests it is suitable to our task. However, there are a number of other factors we wish to test, including the impact of learning the hyperparameters, the effect of using an RBF model as prior on the mean and the effectiveness of using preferences instead of ratings. We also need to confirm the results of our experiments using the gallery application and real users.

We have several specific aspects of our system which we wish to test. In the first set of experiments (Sections 5.1 and 5.2), we study the behaviour of our strategies for learning the kernel hyperparameters and mean function when applied to a known mathematical function. By not having a user in the loop and adopting functions whose optima are known, we are able to measure performance precisely. In Sections 5.3 and 5.4, we bring users into the loop with our animation gallery application (Section 4.2). While we know our methods work in a simulated environment, we want to make sure that some overlooked aspect of human psychology or human-computer interaction does not cause our assumptions to be violated for the application.

- In Section 5.1, we test the particle filtering algorithm. We use a standard test function for automatic testing, and show how even from a poor initial distribution, the algorithm converges to good hyperparameter settings, and that this greatly improves optimization performance.
- In Section 5.2, we use the same test function to study the effect of learning the GP mean. We show how an RBF model trained on increasing amounts of data improves optimization performance.
- In Section 5.3, we track the performance of users on a task, in which they were shown an expert-generated target animation and used variations of the interface to find it. In addition, we test the effect of learning the hyperparameters with humans in the loop, while keeping the GP mean fixed.
- Finally, in Section 5.4 users employ the application to find their own animations, both with and without an RBF mean function. We measure performance with a simple questionnaire.

When $\mathbf{x}^\star = \operatorname{argmax}_{\mathbf{x}} f(\mathbf{x})$ is known, we can measure the *error*, which we define as the deviance of a selected set of parameters from $\mathbf{x}^\star$, projected onto a unit hypercube to account for the different scales of the parameters. When the max is known, we can also measure performance using the "gap" metric [HANZ06]

$$G = \frac{f(\mathbf{x}^{first}) - f(\mathbf{x}^+)}{f(\mathbf{x}^{first}) - f(\mathbf{x}^\star)},$$

where $f(\mathbf{x}^{first})$ is the value of the first function sample (or the case where multiple samples were used on the first iteration, the max of that set). G will therefore be a number between 0, indicating no improvement over the initial sample ($f(\mathbf{x}^+) = f(\mathbf{x}^{first})$), and 1, meaning that the incumbent is the maximum ($f(\mathbf{x}^+) = f(\mathbf{x}^\star)$).

5.1 Hyperparameter learning

In this section, we evaluate the performance of the particle filter (Algorithm 2) when optimizing a test function with known maxima: the *Shekel 5* function [She71], which is a very common test function in the field of global optimization. The function has 4 dimensions, 5 local maxima and 1 global maximum. Because of its dimensionality and fairly steep modes, it is difficult to optimize with naive techniques, but we can expect a well-designed general global optimizer to offer measurable improvement, even if it doesn't find the global maximum. This makes it ideal for study with the gap metric.

We fix the GP mean to zero and focus on studying the effect of learning only the ARD kernel width hyperparameters, $\beta = (\theta_{1:4})$. In this optimization setting, the observations are noise-free (we will examine the effect of learning the mean function in Section 5.2, and the interaction between noise and kernel width hyperparameters has been well-studied elsewhere [SWN03, Liz08]). We train the particle filters for 25 time steps t as shown in Algorithm 2. At each t, we gather an observation vector $\mathbf{f}_t$ by running 20 iterations of Bayesian optimization, using the mean of the predictive distribution $p(\beta_t|\mathbf{f}_{1:t-1})$. The simulation of $\mathbf{f}_t$ is not deterministic because of random initialization. The procedure continues as detailed in Algorithm 2.

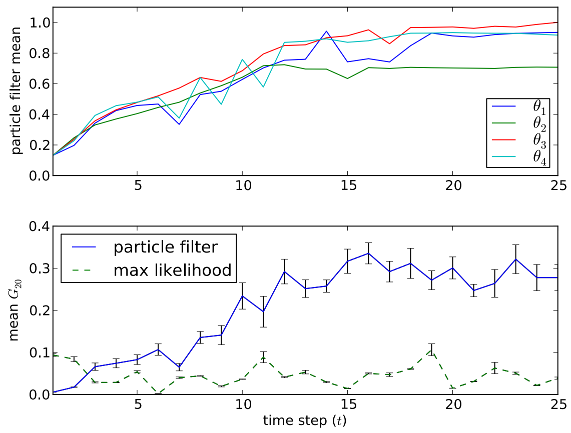


Figure 2: *Learning ARD kernel width hyperparameters using a particle filter. The upper subfigure shows the evolution of $\theta_{1:4}$ over 25 time steps. The lower subfigure shows the performance measure G corresponding to these hyperparameters. It also shows the same measure using hyperparameters learned by maximum likelihood.*

We store the 4 mean trajectories for 25 time steps, $\widehat{\theta}_{1:4,1:25}$, computed with the particle filter. For each of these trajectory values, we run 20 iterations of Bayesian optimization and record the mean and variance of G (G_{20}). We adopted 20 iterations because this is roughly the point at which users of the animation system start to quit if they do not see significant improvement. We repeated the experiment 10 times to obtain confidence estimates. The evolution of G_{20} is shown in the lower plot of Figure 2. For comparison, we also show G_{20} for hyperparameters learned by maximizing the likelihood directly.

The results show that the particle filter not only converges at a reasonably fast rate, but also leads to significant improvements in optimization performance.

5.2 Learning the mean function with RBFs

To test the impact of the mean function on the optimization, we again use the Shekel 5 test function. We fix the hyperparameters to the best values found with a particle filter. Here, we focus on studying the effect of varying dataset sizes to learn the RBF GP mean within this optimization task. The

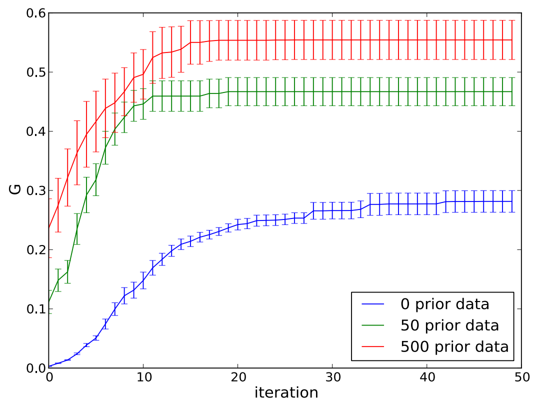


Figure 3: *Effect of adding more data to compute the RBF prior mean. We run Bayesian optimization to find the global maximum, with G as a measure of the optimization performance. Solid lines are the mean performance at each iteration and error bars show the variance. Clearly, learning the mean function improves optimization performance.*

hypothesis is that with more data one should learn a better GP mean function and hence do better at optimization.

To learn the α and $\mathbf{c}$ values of the RBF model, we sampled the test function using Latin hypercube sampling and additive Gaussian noise $\varepsilon \sim \mathcal{N}(0, 0.5)$. We then optimize the function iteratively by selecting the point of maximum EI, evaluating f, and updating the GP. We ran tests using 0, 50 and 500 data to train the RBF model. We set the size of the RBF expansion at $p = 25$ bases. We run the experiments for 50 iterations, which we've observed to be an upper bound on the most typical users would use the animation system. Figure 3 shows that an RBF model trained on 50 noisy samples offers striking improvement on the rate of optimization, and 500 samples even more.

5.3 Gallery interface performance

The previous experiments show clearly that particle filters and RBF models for the GP mean result in significant improvements in Bayesian optimization of known mathematical functions. To test the performance of components of the system with human beings in the loop, we would like to simulate the task of an animator looking for a specific animation. The difficulty with measuring performance in these animation tasks is that we don't know the animator's precise intentions. That is, we don't know the objective function's global maximum. To overcome this difficulty, an expert generated a set of five distinct animations, for which the target parameters were known. Users were shown one of the target animations, picked at random, and asked to find the corresponding target parameters using different variations of the interface. When users found an animation they felt was "close enough", they could select it. Subsequently, the application terminated and logged the number of iterations, distinct animations viewed, and the error. A GP zero mean function was used so as to avoid giving unfair advantage to the target animations. We tested the following scenarios:

- **expert-set hyperparameters (expertHP)** Our work is an extension of [BdFG07], and so we wish to compare our work to that as directly as possible, even though the applications are different. To do this, we (a) set the kernel hyperparameters θ and σ to expert-selected values; (b) restricted the gallery to 2 windows, one generated from $\text{argmax}_{\mathbf{x}} EI(\mathbf{x})$ and one from $\text{argmax}_{\mathbf{x}} \mu(\mathbf{x})$.
- **particle filter hyperparameters (PFHP)** We compared the expert-set hyperparameter model to a pairwise model which is identical except that it uses hyperparameters learned with a particle filter trained on user sessions, as described in Section 3.
- **ratings** We repeated the rating experiment of [BdFG07] on our application to see if there was any undiscovered aspect of our problem that might make it easier to find the target by rating instances numerically. In this case, the user was shown a single animation at a time, corresponding to $\text{argmax}_{\mathbf{x}} EI(\mathbf{x})$ and asked to rate it with a "score" between 0 and 1.
- **4-gallery** Using the same hyperparameters as PFHP, for comparison purposes, we generated a gallery of 4 instances over which the user could enter preferences. Once all preferences were entered, they were added to the model, and a new set of gallery parameters selected.

As shown in Table 1, the hyperparameters learned by PFHP result in a significant improvement in both the number of iterations and animations viewed over the expert hyperparameters (*expertHP*), and result in a slightly lower error. This represents a given by substantial savings in human effort, *and* eliminates the risk involved in having a human expert attempt the difficult but crucial task of setting the hyperparameters.

The *4-gallery* approach requires viewing more animations than pairwise, but is the most accurate, while direct rating performs poorly, as predicted. (Of course, it should be no surprise that the 4-gallery involved more instances, as at each iteration, 4 animations are generated, rather than 2.) The pairwise preferences produced higher error, but required fewer total animation views. Anecdotally, users reported difficulty in using the pairwise preferences due to the limited feedback available. We suspect this lead to the very large standard deviations on the error. Based on these results, the case for using a gallery of several instances is significant, but not as clear-cut as the case for learning the hyperparameters and mean function. If animating is cheap and the cognitive effort of rating is low (as in our application), a large gallery offers lower error. If the goal is to minimize the total number of animations generated and rated, using pairs is preferable. Numerically rating individual animations, however, suffers in both accuracy and the number of instances. Clearly, if we want to make the best use of user time, preferences are the most suitable choice.

scenario	gallery size	parameters	sessions	iterations	animations	error
expertHP	2	4	20	12.43 ± 4.45	22.66 ± 7.35	0.44 ± 0.30
PFHP	2	4	20	8.45 ± 2.81	**14.44 ± 5.03**	0.36 ± 0.34
ratings	1	4	20	28.35 ± 5.13	28.35 ± 5.13	0.31 ± 0.26
4-gallery	4	4	30	**7.57 ± 4.67**	24.85 ± 15.48	**0.22 ± 0.17**
manual *(novice)*	1	12	3	35.33 ± 7.13	35.33 ± 7.13	2.02 ± 0.35
manual *(expert)*	1	12	5	28.40 ± 5.95	28.40 ± 5.95	**0.91 ± 0.30**
4-gallery + manual	4	12	20	**5.38 ± 2.63**	**19.14 ± 7.02**	1.23 ± 0.74

Table 1: *Results of experiments in which users were shown target animations and asked to find them using only specific methods.* **gallery size** *is the number of simultaneous animations instances in the interface used.* **parameters** *is the number of free parameters the user was trying to find.* **sessions** *is the total number of sessions the scenario was run to collect data.* **iterations**, **animations** *and* **error** *are the mean and standard deviation of the number of interface iterations, the total number of different animations viewed, and the divergence of the user's selected parameter values from the target animation, respectively.*

Next, we test whether our "real" application — in which users can restrict ranges and fix parameters in addition to using a gallery of machine-selected animations — is more effective than more traditional "parameter twiddling" applications:

- **manual** We implemented a single-window GUI with no machine learning: the user sets all parameters manually. We distinguished between novices, who have no real understanding of what the parameters mean, and an expert, who is very familiar with the procedural animation system, and who also has a commanding knowledge of fluid animation in general.
- **4-gallery + manual** This is the full interface, in which all parameters start off unrestricted, and the user can indicate preference, and can also restrict the range of any parameter, including setting it to a single value. This directly controls the bounds of the EI maximization, so the next set of parameters will be in the indicated ranges.

The hyperparameters are again learned with a particle filter, and the prior mean function is set to zero. We found that we had to discontinue the initial manual-tweaking experiment. Our first subjects became so frustrated with trying to set the parameters that they frequently expressed a desire to give up. The numbers show the results when the experiment was terminated. We include the final results by means of comparison. We also included a small number of runs for an expert user who was familiar with the manual interface. These are shown as *manual (expert)* in Table 1. We consider the numbers for the manual tool very unreliable — the significant result is that for non-experts, using the gallery made completing the task feasible at all! With that caveat, though, we find it interesting that the expert trials with the manual interface were only slightly better than the non-expert users using the gallery-plus-manual interface.

5.4 Discovery

To test the effect of learning the mean function, we designed a "discovery" experiment, where a user has an idea of what they want, but no ground truth. This also allows us to assess the impact of learning the mean function in a realistic setting. In this experiment, users (familiar with the system, but non-experts) are simply asked to have a rough idea in mind and use the system to find an animation they are satisfied with. In the first 15 trials, we used the zero-mean function. In the second 15, we learned the mean function using data from the first trials. Users were not told which mean function they were using. At the end of each session, users were asked to answer, using a subjective scale of 1–10, the following questions: (Q1) "how close is your selected animation to what you had in mind?"; and (Q2) "independent of how close it was to your target, how much did you like the selected animation?". The goal of these two questions is to measure the effect the mean function had on helping the user find an instance, and also to determine whether the trained mean might cause the user to favour instances that are appealing but not what they were looking for. This could happen, for instance, if the mean function had too much influence, biasing the search toward animations it was trained over exploration.

The results are shown in Table 2. The most dramatic difference is in the number of iterations required for users to find a target — from an average of 11.25 to just 6.5. Moreover, the higher responses to Q1 indicate that users were, indeed, finding what they were looking for, and were not just settling for instances suggested due to the mean function.

6 Conclusions

The problem of setting parameters for a procedural animation system can be treated as one of optimization. However, as our results have shown, the success of this endeavour depends on the ability to learn the prior mean function and model hyperparameters. Learning hyperparameters in a single user session is impractical, but the nature of the task itself offers a solution: while each individual session with the application might involve different users looking for different animations, if we treat the hyperparameter-learning problem as one of state estimation in a hierarchical Bayesian model, we can use particle filters to make the process automatic.

While we were motivated by a specific animation task, we see no reason our model could not be extended to other domains, such as physical simulation, texture design and audio, where continual tuning is required to make something "feel

mean f'n	iterations	Q1	Q2
zero	11.25 ± 3.60	6.64 ± 1.76	7.00 ± 1.41
trained	$\mathbf{6.50 \pm 2.15}$	$\mathbf{7.42 \pm 1.10}$	$\mathbf{7.36 \pm 0.81}$

Table 2: *Comparison of users using the system with a zero function as the mean, and with a mean trained on previous user data. Users were asked to think of an animation and use the system to try to find it. The trained mean function offers improvement in all metrics, but especially in reducing the number of iterations of the system.*

right" according to an underlying psychoperceptual model. Particle filter hyperparameter learning is a novel and powerful solution to the problem of setting these difficult values in data-poor environments. By only updating the *distribution*, the model is guided toward good settings without being overly restricted. The continuous learning and updating ensures that the more the gallery application is used, the better it will become at meeting the needs of its users, even if those needs change over time.

Finally, user expertise is a valuable resource, and if a user knows the ranges or exact value of a parameter, or if they wish to use conventional "slider twiddling" as part of their workflow, it is important that our tool not interfere. We present a framework in which our techniques can be used in conjunction with existing methods.

References

[AN05] Angelidis A., Neyret F.: Simulation of smoke based on vortex filament primitives. In *ACM-SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)* (2005).

[BdFG07] Brochu E., de Freitas N., Ghosh A.: Active preference learning with discrete choice data. In *Advances in Neural Information Processing Systems 20* (2007).

[BGdF07] Brochu E., Ghosh A., de Freitas N.: Preference galleries for material design. In *ACM SIGGRAPH 2007 Posters* (2007), p. 105.

[BHN07] Bridson R., Houriham J., Nordenstam M.: Curl-noise for procedural fluid flow. *ACM Transactions on Graphics 26*, 3 (2007), 46.

[Bis06] Bishop C. M.: *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

[CG05] Chu W., Ghahramani Z.: Preference learning with Gaussian processes. In *Proc. 22nd International Conf. on Machine Learning* (2005).

[DdFG01] Doucet A., de Freitas N., Gordon N.: *Sequential Monte Carlo Methods in Practice*. Statistics for Engineering and Information Science. Springer, 2001.

[HANZ06] Huang D., Allen T. T., Notz W. I., Zheng N.: Global optimization of stochastic black-box systems via sequential Kriging meta-models. *J. of Global Optimization 34*, 3 (March 2006), 441–466.

[JPS93] Jones D. R., Perttunen C. D., Stuckman B. E.: Lipschitzian optimization without the Lipschitz constant. *J. Optimization Theory and Apps 79*, 1 (1993), 157–181.

[JSW98] Jones D. R., Schonlau M., Welch W. J.: Efficient global optimization of expensive black-box functions. *J. Global Optimization 13*, 4 (1998), 455–492.

[KYJF04] Kuang J., Yamaguchi H., Johnson G., Fairchild M.: Testing HDR image rendering algorithms. In *Proc. IS and T/SID 12th Color Imaging Conference* (2004).

[LCTS05] Ledda P., Chalmers A., Troscianko T., Seetzen H.: Evaluation of tone mapping operators using a high dynamic range display. *ACM Transactions on Graphics 24*, 3 (August 2005), 640–648.

[Liz08] Lizotte D.: *Practical Bayesian Optimization*. PhD thesis, University of Alberta, Edmonton, Alberta, Canada, 2008.

[MAB*97] Marks J., Andalman B., Beardsley P. A., Freeman W., Gibson S., Hodgins J., Kang T., Mirtich B., Pfister H., Ruml W., Ryall K., Seims J., Shieber S.: Design galleries: A general approach to setting parameters for computer graphics and animation. *Computer Graphics 31* (1997).

[McF01] McFadden D.: Economic choices. *The American Economic Review 91* (2001), 351–378.

[Mos51] Mosteller F.: Remarks on the method of paired comparisons: I. the least squares solution assuming equal standard deviations and equal correlations. *Psychometrika 16* (1951), 3–9.

[NDM06] Ngan A., Durand F., Matusik W.: Image-driven navigation of analytical BRDF models. In *Eurographics Symposium on Rendering* (2006), Akenine-Möller T., Heidrich W., (Eds.).

[PBJ93] Payne J. W., Bettman J. R., Johnson E. J.: *The Adaptive Decision Maker*. Cambridge University Press, 1993.

[RW06] Rasmussen C. E., Williams C. K. I.: *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, Massachusetts, 2006.

[Sas02] Sasena M. J.: *Flexibility and Efficiency Enhancement for Constrained Global Design Optimization with Kriging Approximations*. PhD thesis, University of Michigan, 2002.

[SF93] Stam J., Fiume E.: Turbulent wind fields for gaseous phenomena. In *SIGGRAPH '93* (1993), pp. 369–376.

[She71] Shekel J.: Test functions for multimodal search techniques. In *5th Princeton Conf. on Information Science and Systems* (1971).

[SRF05] Selle A., Rasmussen N., Fedkiw R.: A vortex particle method for smoke, water and explosions. *ACM Transactions on Graphics 24*, 3 (2005), 910–914.

[SWJ98] Schonlau M., Welch W. J., Jones D. R.: Global versus local search in constrained optimization of computer models. *Lecture Notes-Monograph Series 34* (1998), 11–25.

[SWN03] Santner T. J., Williams B., Notz W.: *The Design and Analysis of Computer Experiments*. Springer, 2003.

[TGY*09] Talton J. O., Gibson D., Yang L., Hanrahan P., Koltun V.: Exploratory modeling with collaborative design spaces. In *Proc. 2nd Annual ACM SIGGRAPH Conf. and Exhibition in Asia* (2009).

[Thu27] Thurstone L.: A law of comparative judgement. *Psychological Review 34* (1927), 273–286.

[WH91] Wejchert J., Haumann D.: Animation aerodynamics. In *SIGGRAPH '91* (New York, NY, USA, 1991), ACM, pp. 19–22.