EUROGRAPHICS 2015 / O. Sorkine-Hornung and M. Wimmer
(Guest Editors)

Volume 34 (2015), Number 2

Generating Design Suggestions under Tight Constraints with Gradient-based Probabilistic Programming

Daniel Ritchie Sharon Lin Noah D. Goodman Pat Hanrahan

Stanford University

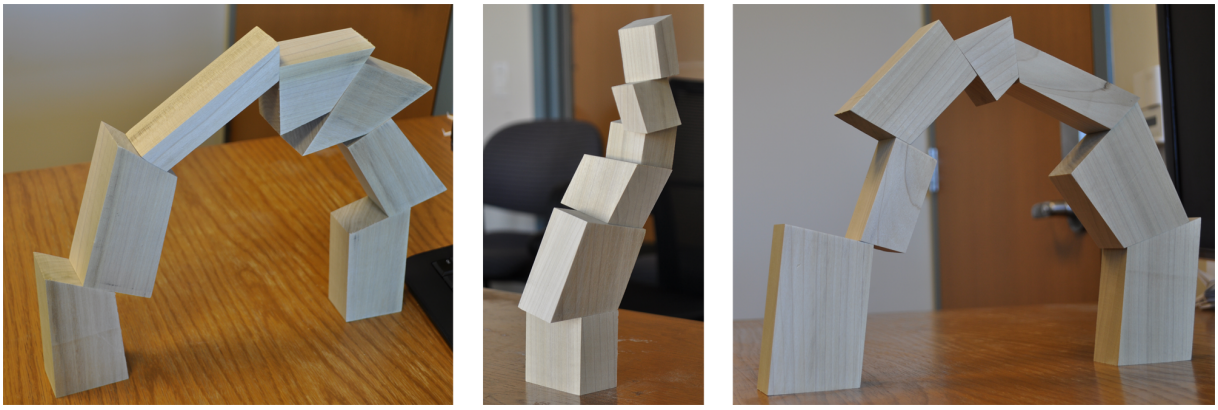

Figure 1: *Physical realizations of stable structures generated by our system. To create these structures, we write programs that generate random structures (e.g. a random tower or a randomly-perturbed arch), constrain the output of the program to be near static equilibrium, and then sample from the constrained output space using Hamiltonian Monte Carlo.*

Abstract

We present a system for generating suggestions from highly-constrained, continuous design spaces. We formulate suggestion as sampling from a probability distribution; constraints are represented as factors that concentrate probability mass around sub-manifolds of the design space. These sampling problems are intractable using typical random walk MCMC techniques, so we adopt Hamiltonian Monte Carlo (HMC), a gradient-based MCMC method. We implement HMC in a high-performance probabilistic programming language, and we evaluate its ability to efficiently generate suggestions for two different, highly-constrained example applications: vector art coloring and designing stable stacking structures.

1. Introduction

Considering multiple possibilities is critical in design. Exposure to different examples facilitates creativity—for instance, prototyping multiple alternatives can lead to better-quality final designs [KDK14, DGK*10]. Exploring the whole space of creative options seems to help people avoid fixation and overcome their unconscious biases [JS91]. Computation can assist with this exploration by generating suggestions: given a model of the design space, computers can synthesize examples that their users might never have thought of independently.

In computer graphics, probabilistic inference has become popular for computer-aided suggestion in domains as diverse as color selection and furniture layout [LRFH13, YYW*12]. In this framework, the user provides a model of the design space by expressing her preferences as soft constraints,

or *factors*. These factors are combined into a probability distribution, and sampling from this distribution using Markov Chain Monte Carlo (MCMC) provides an efficient suggestion-generation mechanism. Working with inference can be made easier through *probabilistic programming*. A probabilistic program defines a random process (e.g. constructing a random scene); inference then amounts to reasoning about the space of possible executions of that process under constraints [GMR*08]. Inference algorithms only need access to the elementary random choices made by the program, allowing them to be used in virtually any design application domain.

Real design applications feature a range of constraints, from vague preferences that loosely shape the design space ("Make this object a reddish color") to strict requirements that eliminate entire regions of the space as undesirable ("This container must hold one liter of liquid, up to manufacturing tolerance"). But the tighter these constraints, the more ill-conditioned the underlying probability distribution becomes. As we will show, the random walk MCMC methods typically used for design suggestion break down when faced with tight constraints, especially in high-dimensional design spaces.

To work around this problem, developers can implement complex, application-specific MCMC algorithms that exploit knowledge of constraint structure [JM12, SW14]. This strategy does not scale, however, as it requires new algorithms be developed for each new application. General-purpose solutions would be preferrable, especially for use with probabilistic programming.

In this paper, we take a step toward solving this problem by adopting a different sampling algorithm: Hamiltonian Monte Carlo (HMC). HMC is used in Bayesian statistics to train predictive models with many parameters [Nea10]. It excels when the posterior distribution of the parameters given training data causes some parameters to become highly correlated—the same statistical problem as design variables being strongly coupled by tight constraints. Its performance comes from using the *gradient* of the probability distribution to take less-random walks through the state space. This gradient can be computed automatically, making HMC a general-purpose, application-agonistic tool. HMC operates on continuous design domains (i.e subsets of $\mathbb{R}^n$). This property makes it a tool well-suited to graphics applications, since they often feature many continuous quantities (positions, directions, dimensions, colors, etc.)

To evaluate the usefulness of HMC for design suggestion tasks, we implemented the algorithm in Quicksand, an open-source probabilistic programming language embedded in the Terra language for high-performance computing [Rit14, DHA*13]. We then use our implementation to generate suggestions for two different example applications: vector art coloring and designing stacking structures. These applications employ several challenging and generally-useful constraints, such as physical stability and symmetry. We compare the performance of HMC to classical random walk MCMC on these two examples, demonstrating that HMC provides both qualitatively and quantitatively better design space exploration in the presence of tight constraints.

Our contributions are:

1. The introduction of Hamiltonian Monte Carlo to handle tight constraints in probabilistic design suggestion.
2. An efficient implementation of HMC in a general-purpose probabilistic programming language.
3. An evaluation of this implementation through two representative example applications: vector art coloring and designing stacking structures.

We view HMC as one new tool in a toolbox that needs to grow in order to make probabilistic computational design efficient and easy to use.

2. Background and Related Work

2.1. Design Space Exploration

Design space exploration in computer graphics can be traced back at least as far as the seminal work on Design Galleries by Marks and colleagues [MAB*97]. Exploration can be divided into two phases: generating suggestions and navigating between those suggestions. Our work focuses on generating suggestions; other researchers have examined the navigation problem [BYMW13, UIM12].

Researchers have experimented with different algorithmic frameworks for generating design suggestions, including genetic algorithms [XZCOC12], nonlinear manifold exploration [YYPM11], and probabilistic inference [JTRS12b, TLL*11, MSL*11]. Our system uses probabilistic inference, and the particular inference algorithm it relies on, Hamiltonian Monte Carlo, shares some mathematical similarities with manifold exploration methods.

Design domains can contain discrete variables, continuous variables, or some combination of both. Several existing design suggestion methods operate on purely discrete design spaces, including shape generation by part combination [KCKK12, JTRS12a] and tiled pattern synthesis [YBY*13]. In contrast, our work focuses on continuous design spaces, which are often used to model quantities such as positions, directions, sizes, and colors. In the mixed discrete/continuous regime, an important subclass of design spaces are those where discrete choices dictate the structure of a continuous parameter set [YYW*12, FRS*12]. The techniques presented in this paper can be also applied to the continuous subsets of these domains, for a fixed setting of the discrete choices.

Probability distributions over design spaces are typically complex, and researchers have explored techniques

to make sampling from them more tractable. Parallel tempering, which assists samplers when probability mass is concentrated around multiple modes, is one notable example [TLL*11,MSL*11,LRFH13]. In contrast, the techniques we present help when probability mass is concentrated along thin manifolds. The two methods can be used in concert if a design space exhibits both multi-modality and manifold structure.

2.2. HMC Applications

HMC has been applied in other areas that require searching through complex design spaces. It has found use in trajectory optimization for robot motion planning [ZRD*13]. It has also been applied in 3D printing for estimating and correcting material shrinkage during the printing process [HZSD]. Both of these efforts are concerned with optimization problems: they attempt to find the best possible solution in a large design space. In contrast, we seek to explore large sets of possibilites in design spaces.

HMC also been applied to probabilistic programs. One such effort implements HMC in the Church programming language by viewing the gradient computation as a non-standard interpretation of the program [WGSS11]. The Stan inference system also uses a variant of HMC to perform inference in user-programmable generative models [Sta14]. For experimenting with graphics applications, we chose to implement HMC in Quicksand [Rit14]. Quicksand generates high-performance, low-level code (whereas Church is a high-level, functional language) and is a general-purpose programming language (whereas Stan uses a statistical modeling domain-specific language)—these properties make it easier to efficiently express graphics programs.

3. The Problem: Tight Constraints

To illustrate the problem posed by tight constraints, we examine a token example application, constraining the position of a 2D point, that evokes the kind of constraints that can arise in spatial layout tasks.

Suppose we constrain the position of a point (x,y) with the following energy penalty:

$$\texttt{softeq}(y^4 - y^2 + x^2 - 0.25, 0, \sigma) \tag{1}$$

Here, the 'soft equality' function $\texttt{softeq}(z,\mu,\sigma)$ is an alias for the log of the normal distribution with mean μ and variance σ^2 evaluated at z (i.e. $\log\mathcal{N}(z,\mu,\sigma)$). In this case, it penalizes points that fall too far from the 0-isocontour of the function $y^4 - y^2 + x^2 - 0.25$. The bandwidth σ controls the tightness of this factor, or how aggressively it applies its penalty. The top left of Figure 2 shows the probability density $\pi(x,y)$ that results from setting $\sigma = 0.1$.

To sample from such a distribution, Markov Chain Monte Carlo—in particular, the Metropolis-Hastings algorithm (MH) [MRR*53]—is often the method of choice. MH

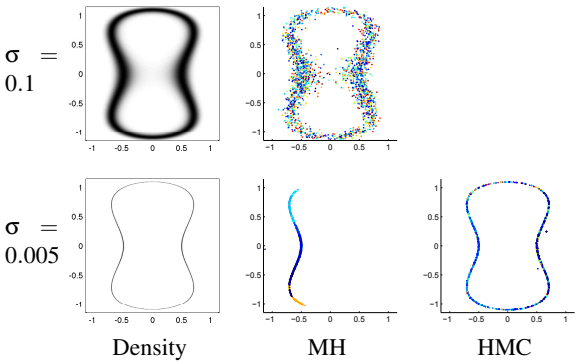

Figure 2: *Tight constraints in action on a simple 2D example. Top left: The probability density of Equation 1 with $\sigma = 0.1$. Top middle: Samples drawn from this density using MH. Bottom left: The probability density of Equation 1 with $\sigma = 0.005$. Bottom middle: Samples drawn from this density using MH. Bottom right: Samples drawn from this density using HMC. HMC fully explores the distribution when constraints are tight, while MH does not. Samples are colored by time to illustrate the dynamics of the two algorithms.*

works by taking some initial current state (x_0,y_0), *proposing* a new state $(\tilde{x}_0,\tilde{y}_0)$, and then *accepting* that state if its probability did not decrease by too much. If the proposed state is accepted, it becomes the new current state (x_1,y_1), and the process repeats. A simple, popular choice of proposal strategy is to construct $(\tilde{x}_0,\tilde{y}_0)$ by choosing one of x_0 or y_0 at random and making a small random perturbation to it. The top middle of Figure 2 shows 2000 samples drawn from $\pi(x,y)$ using MH. The samples form a good approximation to the true distribution.

The same does not hold when the constraint is tightened by decreasing σ to 0.005. The bottom left of Figure 2 shows the new probability density $\pi'(x,y)$; the narrow ridges of high probability reflect the tightened constraint. Running MH for the same number of samples on this new distribution gives the result in the bottom middle of Figure 2. While MH finds its way to a high-probability region (the phase of sampling statisticians call *burn-in*), it does not fully explore the distribution. Most random perturbations push the point (x,y) off the narrow, high probability ridges, so the perturbation size must be made very small. We also color sample points by time; the spatially-contiguous regions of constant color illustrate the sampler's slow progress. This is a two-dimensional example chosen for ease of visualization; we could exploit this low-dimensionality and our knowledge of the distribution's symmetry to do better via brute force. Unfortunately, this isn't possible for high-dimensional distributions with unknown shape, where the variable-coupling problem becomes even worse for MH [Nea10].

We can quantify MH's poor performance using *autocorrelation*, a measure of how similar successive samples are to

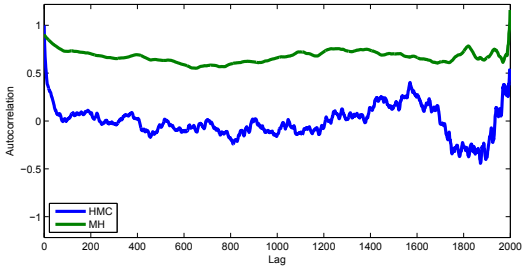

Figure 3: *Autocorrelation plots for the samples show in the bottom row of Figure 2. HMC oscillates around zero (the ideal value), while MH never approaches this target.*

one another. This is a standard test for assessing MCMC performance for Bayesian statistics [KCGN98]. The green line in Figure 3 shows the autocorrelation plot of the MH sampling trace, which is far from the ideal value of zero: since the sampler is stuck in the same part of the state space, many samples are similar, so autocorrelation remains high.

Hamiltonian Monte Carlo performs both qualitatively and quantitatively better on this example. The bottom right of Figure 2 shows samples drawn from $\pi'(x,y)$ by HMC given the same computational budget as MH. Visually, these samples represent the distribution much better, and autocorrelation quickly drops to near-zero (Figure 3, blue line).

The way HMC works can be explained by physical analogy. If we invert the probability density landscape in the bottom left of Figure 2, the thin ridges become narrow valleys. Imagine placing a ball in one of these valleys and rolling it in some random direction. It would roll up and down the surrounding walls, but it would also make progress down the length of the valley. This is the core process underlying Hamiltonian Monte Carlo: it runs a simulation of frictionless Hamiltonian dynamics using the negative log probability $-\log\pi'(x,y)$ as its potential energy.

In the next section, we describe the Hamiltonian Monte Carlo algorithm and our implementation of it in more detail. We then evaluate our system on two different design suggestion tasks: coloring vector art, and designing stable stacking structures (Section 5).

4. Hamiltonian Monte Carlo

As shown in the previous section, using MH can result in a sampler that moves very slowly across the state space—producing highly-correlated samples—when multiple variables are strongly coupled by tight constraints.

Hamiltonian Monte Carlo (HMC) is a variant of MCMC that can efficiently explore highly-coupled, high-dimensional continuous distributions. It was originally developed for lattice field theory simulations in statistical physics [DKPR87], but has since seen increasing adoption in the Bayesian statistics community (see Neal [Nea10] for an excellent overview and survey).

HMC derives its name from Hamiltonian dynamics, which it uses to generate proposals. For this purpose, Hamiltonian dynamics specify the behavior of a frictionless, unit-mass particle with some position $\mathbf{x}$ and momentum $\mathbf{p}$. At a given point in time, the particle has kinetic energy $K(\mathbf{p}) = \mathbf{p}^T\mathbf{p}/2$ and potential energy $U(\mathbf{x}) = -\log\pi(\mathbf{x})$, the sum of which is called the Hamiltonian: $H(\mathbf{x},\mathbf{p}) = K(\mathbf{p}) + U(\mathbf{x})$.

Given a current state $\mathbf{x}$ from state space $\mathbf{X}$, HMC generates proposals as follows:

1. Sample a random momentum $\mathbf{p} \sim \mathcal{N}(\cdot, 0, \mathbb{I}^n)$.
2. Simulate the dynamics of the particle $(\mathbf{x},\mathbf{p})$ for L time steps, resulting in the particle $(\mathbf{x}',\mathbf{p}')$.
3. Accept the new particle with probability $\min[1, \exp(H(\mathbf{x},\mathbf{p}) - H(\mathbf{x}',\mathbf{p}'))]$.

Essentially, HMC performs a Metropolis Hastings propose + accept step on an augmented state space where the states are $(\mathbf{x},\mathbf{p}) \in \mathbb{R}^n \times \mathbb{R}^n$. Instead of walking through the state space with single steps in random directions, HMC follows long, multi-step paths along the energy landscape defined by $-\log\pi(\mathbf{x})$. When this landscape is defined by constraints, as in our applications, adhering to its contours corresponds to constraint satisfaction.

To perform Step 2 above, we must simulate the time evolution of our fictitious particle. This is governed by the differential equations:

$$\frac{d\mathbf{x}}{dt} = \nabla_{\mathbf{p}} H(\mathbf{x},\mathbf{p}) = \nabla K(\mathbf{p}) = \mathbf{p}$$
$$\frac{d\mathbf{p}}{dt} = -\nabla_{\mathbf{x}} H(\mathbf{x},\mathbf{p}) = -\nabla U(\mathbf{x}) = \nabla \log\pi(\mathbf{x})$$

which are numerically simulated using the discrete-time update rules

$$\mathbf{p}(t+\frac{\varepsilon}{2}) = \mathbf{p}(t) + \frac{\varepsilon}{2}\nabla\log\pi(\mathbf{x}(t))$$
$$\mathbf{x}(t+\varepsilon) = \mathbf{x}(t) + \varepsilon\mathbf{p}(t+\frac{\varepsilon}{2})$$
$$\mathbf{p}(t+\varepsilon) = \mathbf{p}(t+\frac{\varepsilon}{2}) + \frac{\varepsilon}{2}\nabla\log\pi(\mathbf{x}(t+\varepsilon))$$

known as the *leapfrog* integrator [LR04]. The leapfrog scheme has two critical properties that make it work for HMC proposals. First, it is a *sympletic* integrator (i.e. the map from $\mathbf{X}$ to $\mathbf{X}$ that it defines preserves volume). Second, it is *time-reversible*: if $\texttt{leapfrog}((\mathbf{x},\mathbf{p}),\varepsilon) = (\mathbf{x}',\mathbf{p}')$, then $\texttt{leapfrog}((\mathbf{x}',-\mathbf{p}'),\varepsilon) = (\mathbf{x},-\mathbf{p})$. In other words, flipping the direction of momentum and simulating 'backwards' returns the system to the state from which it started. These properties are key to proving that HMC satisfies the detailed balance condition and thus defines a valid sampler [Nea10].

Parameters HMC has two parameters: the number of leapfrog steps L and the simulation step size ε. The tighter the constraints used in a particular application, the smaller

ε must be to keep the Hamiltonian dynamics simulation numerically stable. Consequently, the number of steps L must increase for HMC proposals to make progress exploring the state space. We use a method proposed by Hoffman and Gelman [HG14] to automatically set ε and leave L as the only free parameter in the system. We found $L = 100$ to be sufficient for most of our experiments. There is also a variant of HMC that attempts to automatically, adaptively determine L as it traverses the state space [HG14].

Bounded variables Many design applications require variables with strict bounds (e.g. "this object must be between 10 and 50 cm long"). These can be incorporated into HMC via *variable transformation*: letting a variable x be unbounded, but transforming it such that the value $\tilde{x}$ exposed to the program is bounded. For a variable with both a lower bound l and an upper bound u, the typical transformation is logistic:

$$\tilde{x} = l + (u - l) \cdot \frac{1}{1 + \exp(-x)}$$

Similar transforms exist for one-sided bounds [Sta14].

Implementation We implemented Hamiltonian Monte Carlo in Quicksand, a probabilistic programming language embedded in Terra [Rit14, DHA*13]. We chose this implementation target because Terra is a low-level language that compiles to efficient machine code, which we believe to be important for achieving sufficient performance for graphics applications. We implement HMC as a custom MCMC kernel in Quicksand. Quicksand supports inference over arbitrary programs, including recursive programs and programs whose set of random choices may change based on control flow decisions. Since HMC operates on $\mathbb{R}^n$, we can only use HMC to explore parts of the execution space with a fixed set of random choices. HMC could be composed with other Quicksand MCMC kernels (such as LARJ-MCMC [YYW*12]) to perform inference on structure-changing programs.

Our system uses automatic differentiation (AD) to compute the gradients required by HMC, relieving the user of having to derive gradients manually. In particular, it uses *reverse-mode* AD, which computes the gradient in just two passes over the program, regardless of state space dimensionality [CFG*01, Spe80]. Efficient symbolic differentiation could also be used for some parts of the program and might further improve performance [Gue07].

5. Evaluation

We use our implementation to evaluate the usefulness of HMC for computational design by generating suggestions for two example applications: vector art coloring and building stacking structures. These are two unrelated application domains that both require tight constraints to eliminate undesirable regions of the design space.

We compare the statistical efficiency of HMC with MH and show that HMC's improved efficiency leads to qualitatively better suggestion results. For fair comparison, we initialize each algorithm by burning in for a fixed number of MH iterations. In all experiments, MH proposal bandwidths are automatically adapted to give $\sim 23\%$ acceptance, and HMC steps sizes are automatically adapted to give $\sim 65\%$ acceptance. Selecting a good target acceptance rate can be highly problem-specific, but there is theoretical evidence that these are good general settings for their respective algorithms [RGG97]. Each algorithm is allotted the same computational budget in terms of *program evaluations*. An HMC sampler with L leapfrog steps uses $2L$ as many evaluations to generate a sample as an MH sampler (the factor of 2 comes from the reverse-mode AD backwards pass). So if the HMC sampler runs for 1000 iterations using 100 leapfrog steps, MH is allowed to run for $(2 \cdot 100) \cdot 1000 = 200000$ iterations.

We also collect timing data to demonstrate that our implementation generates suggestions quickly enough for practical use. All timing information reported in the following experiments was collected on an Intel Core i7-3840QM machine with 16GB RAM running OSX 10.8.5.

Finally, source code for these examples is available on GitHub at `https://github.com/dritchie/graphics-hmc`.

5.1. Vector Art Coloring

In vector illustrations, a significant portion of a design's visual impact comes from color choice. Designers must consider semantics as well as aesthetics to create plausible and harmonious colorings. For example, certain objects may be strongly associated with specific colors (e.g. sky to blue), regions that are part of the same material may need to have similar hues, and shading effects may dictate that some regions should be lighter than others.

Previous work has modeled the compatibility of color combinations and arrangements [OL06, OAH11, LRFH13]. For pattern images, Lin and colleagues introduce a coloring model composed of soft constraints learned from artist examples [LRFH13]. Their system uses MH, augmented with variable swaps and parallel tempering for faster exploration of multiple modes, to sample coloring suggestions from the learned model.

To compare the effectiveness of HMC to MH for coloring constrained vector art, we add tight semantic constraints to a simplified version of the coloring model by Lin and colleagues. Specifically, we add *Same-Chroma* constraints, which enforce that two regions should have the same chromatic content (i.e. the same color irrespective of lightness), and *Lightness-Relation* constraints, which enforce that one region should be brighter or darker than another. These constraints are much tighter than the soft constraints that comprise the base model, and thus they are likely to cause trouble

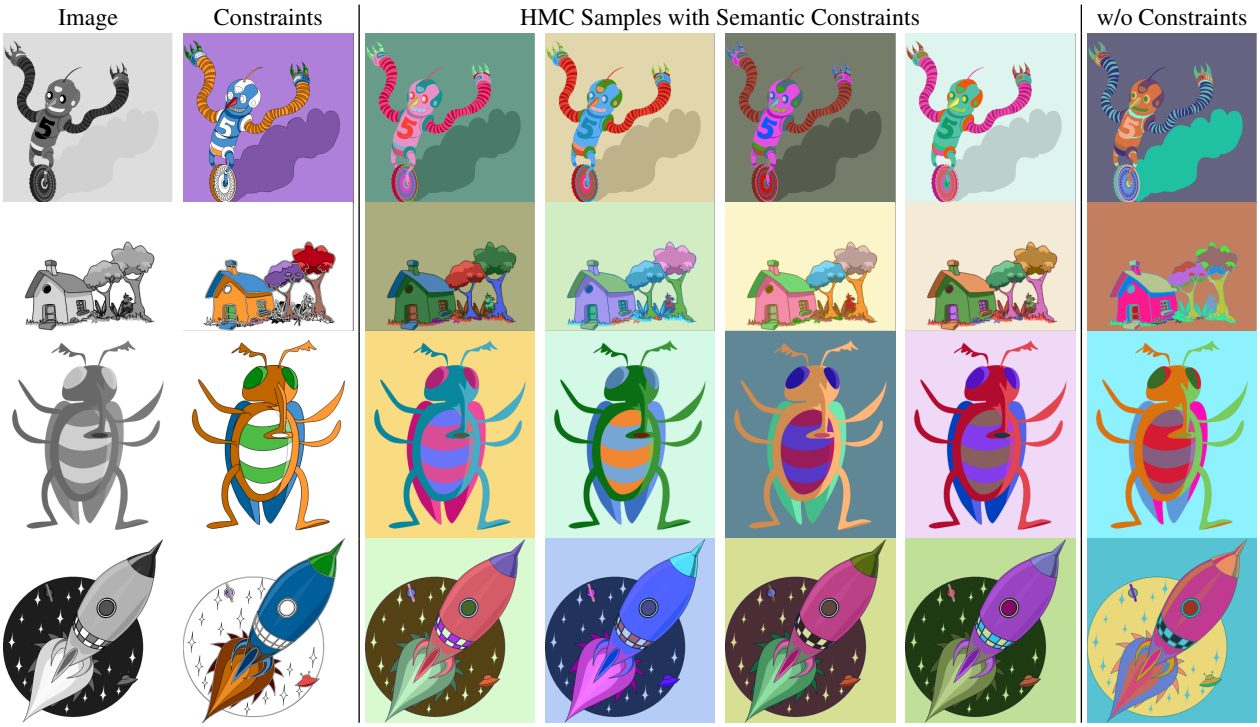

Figure 4: *Vector art colorings with and without semantic constraints. Image: The image template, which maps individually-recolorable regions to different grayscale levels. Constraints: Visualization of the applied constraints. Same-Chroma constraints over regions are visualized with the same hue. White regions have no hue constraints. Lightness-Relation constraints for regions of the same hue are visualized with darker or lighter shades.* Additional Lightness-Relation constraints are as follows*: Robot: eye centers lighter than helmet lights, helmet lights lighter than helmet and robot body, number "5" darker than body.* House*: sky lighter than roof and tree highlights, lineart darker than shadows.* Rocket*: lineart darker than space, stars lighter than middle flame, window darker than rocket body.*

for MH. Refer to the Appendix for the full specification of our coloring model.

Figure 4 shows examples of sampling from three vector art images using HMC under multiple Same-Chroma and Lightness-Relation constraints. The first and second columns of the figure show the vector art template and a visualization of the semantic constraints applied. Under these tight constraints, HMC is still able to sample a variety of different colorings.

In Figure 5, we compare the performance of HMC and MH under the same computational budget. We ran the HMC sampler for 1000 iterations using 100 leapfrog steps and the MH sampler for the equivalent of 1000 HMC iterations (200000 iterations). The first two columns show 'coverage maps' for the two sampling traces, where stronger blue regions indicate that more colors were sampled for that region and that the sampler is exploring the space better. To compute coverage, we discretize CIELAB space into 256 bins (4x8x8) and count the percentage of bins visited for each region. HMC consistently samples more colors than MH. The background in the Bug example has high coverage under MH because it does not participate in any semantic constraints. The third column shows autocorrelation plots for the runs, again demonstrating that the MH samples are more self-similar.

Timings for these examples are shown in Figure 6. We report the time consumed by the burn-in phase (the 'start-up cost' of the system), the time taken by sampling (when the system is generating useful suggestions) as well as the acceptance ratio of the HMC sampler. The HMC sampler draws around 60 new samples per second (number of samples drawn multiplied by acceptance rate and divided by sampling time). Rates such as these should be sufficient for use in an interactive coloring tool.

5.2. Stable Stacking Structures

People are fascinated with the stability of physical structures. The Leaning Tower of Pisa draws over a million visitors each year, games such as Hasbro's Jenga and Areaware's Balancing Blocks have enduring popularity, and balancing

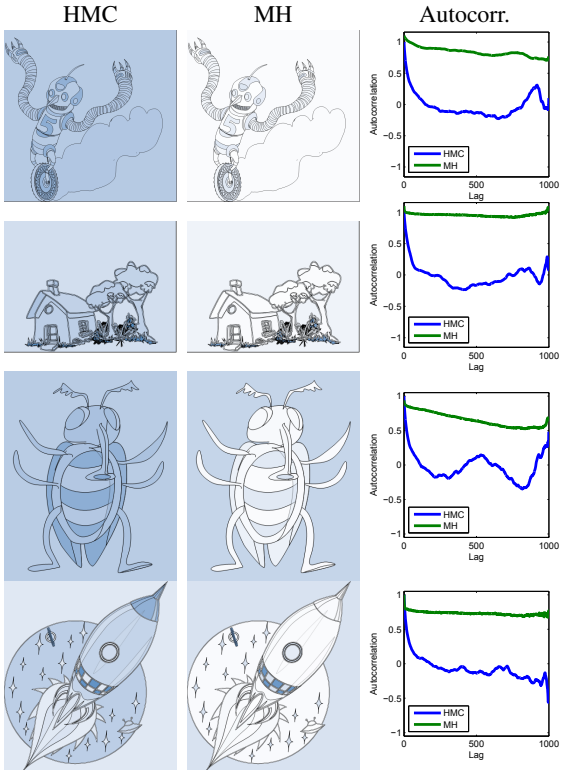

Figure 5: *The first two columns show coverage plots for HMC and MH sampling on the three image templates. Darker shades of blue indicate that more colors were sampled for the given region, while white indicates fewer colors sampled. Colors are counted by discretizing CIELAB space into 256 bins. The last column shows autocorrelation plots comparing HMC and MH.*

Example	$\|\mathbf{X}\|$	Burn-in	Sampling	Accept. ratio
Robot (MH)	57	0.12 s	12.66 s	0.24
Robot (HMC)	57	0.12 s	10.61 s	0.65
House (MH)	60	0.13 s	13.38 s	0.24
House (HMC)	60	0.13 s	10.67 s	0.65
Bug (MH)	30	0.06 s	6.08 s	0.24
Bug (HMC)	30	0.06 s	3.56 s	0.63
Rocket (MH)	60	0.12 s	14.53 s	0.23
Rocket (HMC)	60	0.12 s	10.61 s	0.63

Figure 6: *Timing data for the examples shown in Figure 4. $|\mathbf{X}|$ is the number of random choices made by a program.*

Figure 7: *Real-world inspiration for our stable stacking application. Left: Areaware's Balancing Blocks game. Right: Balacing rock sculpture.*

rock sculptures have become a form of performance art (Figure 7). In this section, we consider the computational design of stacking structures made of rigid blocks that remain stable despite their apparent precariousness.

Prior work has addressed the stability of design artifacts in domains such as truss structure design, 3D printing, and procedural building grammars [SHOW02,PWLSH13,WOD09]. These projects pose stability as an optimization problem: given an initial input object (e.g. a 3d model or procedural grammar derivation), seek toward a configuration of the object that is stable. In contrast, we wish to explore the variety of possible configurations of a given structure that will stand.

For a rigid structure to be stable, it must be in *static equilibrium*: the net force and net torque on every component must be zero. In general, these forces are not directly computable from the structure's geometry but are defined implicitly by this equilibrium condition. We can think of them as random variables whose values are tightly coupled by the equilibrium constraint. This suggests a simple generative model for creating stable structures: generate a random block structure, introduce latent variables representing forces between blocks, encourage equilibrium with a tight constraint, and sample from the resulting distribution using HMC. See the Appendix for the full specification of our statics model.

To keep our example application simple, we consider only convex, hexahedral blocks. While this simplification does not capture all the rich detail of stacking structures in the real world (e.g. the irregular convex polyhedra in Figure 7, left), it admits a wide range of stacking arrangements.

Figure 8 shows some examples of sampling from a random block stacking program using both HMC and MH. We

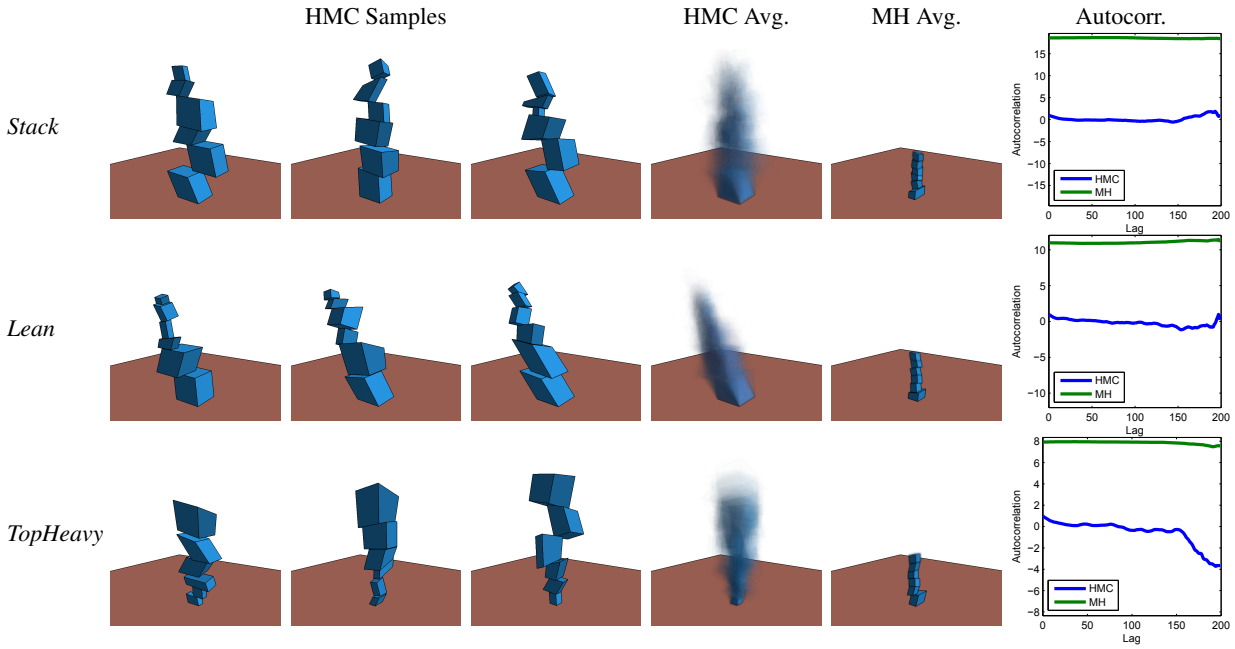

Figure 8: *Generating stable block stacks with different criteria. Top: A stack with no additional constraints. Middle: Encouraging the stack to lean in a particular direction. Bottom: Encouraging each block to be twice as large as the block below it. For each scenario, we show three HMC samples, the average of all samples generated by each method (200 for HMC, 400000 for MH), and a comparison of their autocorrelation curves.*

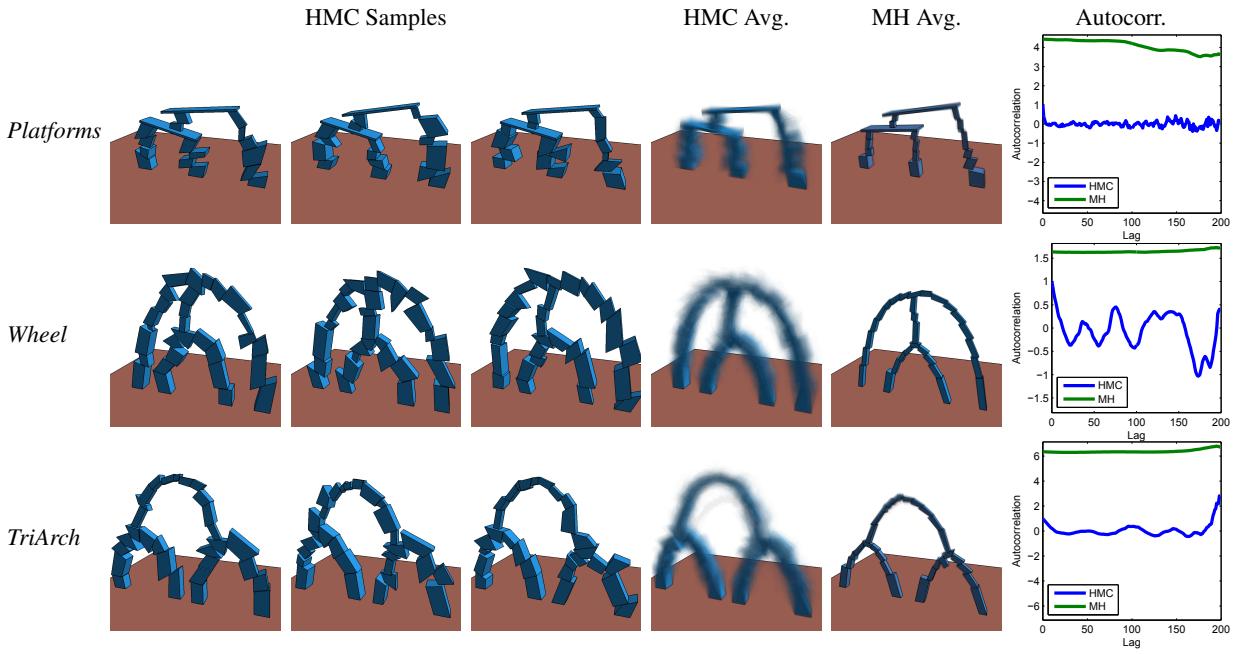

Figure 9: *Generating stacking structures with more complex, cyclical topologies.*

show three interesting structures sampled by HMC, as well as the 'average' structure produced by both algorithms. We also compare the autocorrelation curves of the two sample traces, where autocorrelation is computed by reducing each structure to a vector of all block vertex positions. We ran the HMC sampler with 1000 leapfrog steps for 200 samples and the MH sampler for the equivalent of 200 HMC samples (400000 iterations).

The top row shows results from the stacking program. In the middle row, we add a factor to encourage the stack to lean in a particular direction by penalizing the distance of each block's center of mass to a target line. We also generate precarious-looking 'top-heavy' stacks by adding a factor that encourages each block's volume to be twice as large as that of the block below it (bottom row). HMC has little trouble exploring the complex probability landscape induced by the stability constraint, but MH struggles, seeking out a local maximum and barely deviating from it. MH generates small structures because we initialize the latent force variables to zero, so it can quickly minimize force and torque residuals by shrinking all blocks to the minimum size allowed by the program.

Figure 9 shows this same comparison with programs that generate more topologically-complex structures. HMC again successfully samples many interesting configurations of these structures, whereas MH again becomes stuck. The complex, cyclical contact relationships in these examples make the space of stable configurations more tightly constrained than in the examples of Figure 8. The supplemental video shows animations of some of these sample traces which better illustrate the dynamics of the different algorithms.

Since it uses softened constraints, HMC in general cannot guarantee that the structures it samples will be exactly in equilibrium, only that they will be close to it. Thus, these examples need more leapfrog steps (1000) because constraint bandwidths have to be kept very tight to keep the sampler sufficiently close to the static equilibrium manifold (see Appendix). We used a linear program solver to check whether each generated structure satisfies the equilibrium equations, and nearly all of them do.

Timing statistics for these examples are shown in Figure 10. In general, running time scales linearly with the complexity of block topology. The sampling rate is lower than in the coloring examples, since the complexity of the static equilibrium constraint necessitates taking more small leapfrog steps. Each step is also more expensive, since the statics model requires more computation and makes many more random choices.

To illustrate another use of tight constraints, Figure 11 shows three structures generated from a simple arch program and the *TriArch* program with an additional bilateral symmetry constraint. Adding this constraint takes very little extra effort in our system: the program simply reflects the structure about the symmetry plane and then applies a `softeq` factor to each symmetric pair of block vertices.

Example	$\lvert\mathbf{X}\rvert$	Burn-in	Sampling	Accept. ratio
Stack (MH)	118	30.57 s	65.22 s	0.23
Stack (HMC)	118	36.30 s	148.76 s	0.61
Lean (MH)	118	24.10 s	48.39 s	0.23
Lean (HMC)	118	27.90 s	118.86 s	0.62
TopHeavy (MH)	118	33.35 s	58.48 s	0.23
TopHeavy (HMC)	118	38.04 s	186.62 s	0.59
Platforms (MH)	330	62.52 s	118.75 s	0.25
Platforms (HMC)	330	62.54 s	279.9 s	0.59
Wheel (MH)	555	93.76 s	688.70 s	0.26
Wheel (HMC)	555	98.04 s	729.54 s	0.63
TriArch (MH)	555	94.90 s	191.25 s	0.26
TriArch (HMC)	555	95.5 s	700 s	0.62

Figure 10: *Timing data for the examples shown in Figures 8 and 9.* $\lvert\mathbf{X}\rvert$ *is the number of random choices made by a program.*

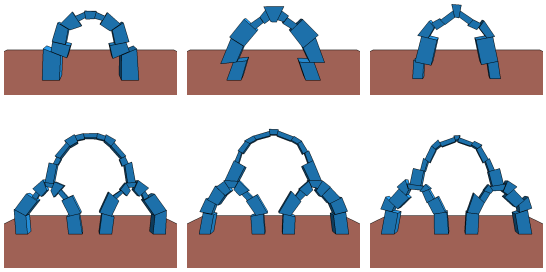

Figure 11: *Structures generated with an additional constraint encouraging bilateral symmetry.*

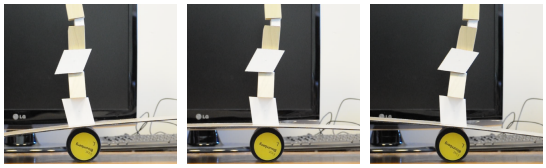

Figure 12: *Testing a block stack generated under the constraint that it be stable at up to* $\pm 10^\circ$ *tilts of the ground plane.*

To validate our statics model, we built physical protoypes of some structures generated by our system (Figure 1). In

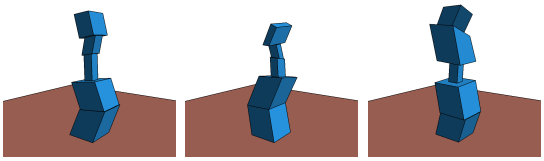

Figure 13: *Block stacks generated under the additional constraint that they be stable at every intermediate construction step.*

the program used to generate these examples, we restricted block shapes to be planar extrusions to allow us to easily cut them out of wood stock. All blocks shown were cut from 38 mm ($1\frac{1}{2}$ in) poplar, whose density and coefficient of friction we estimated as 425 kg/m^3 and 0.3, respectively [Ber10].

To increase the stability of a structure, we can enforce that it be stable under some class of perturbations, rather than at a single rest configuration. Figure 12 shows a block stack generated under the constraint that it stand under as much as $\pm 10^\circ$ tilts of its ground plane. To enforce this condition, we write a program that generates a random stack as before, then rotates the entire scene $\pm 10^\circ$ about one axis, applying a stability constraint at each rotation.

The generated structures shown thus far are not necessarily stable at every step of their construction, which can complicate the process of physically building them. We can mitigate this problem by applying a stability factor at intermediate phases of structure generation, as opposed to just one at the end. Figure 13 shows three example stacks generated this way. To generate more exciting structures under this constraint, one could write programs that use temporary scaffolding to hold the structure up, treat the presence and configuration of that scaffolding as random variables, and infer plausible build processes. This is a promising avenue for future work.

6. Discussion and Future Work

This paper introduced Hamiltonian Monte Carlo to probabilistic computational design. We implemented HMC in a high-performance probabilistic programming language, and we evaluated it on two example applications, showing that it can efficiently generate suggestions in highly-constrained scenarios.

HMC relies on the gradient $\nabla \log \pi(\mathbf{x})$ to make proposals, so the probability π must be continuous and differentiable everywhere. This requirement limits the factors that can be used to define π. For exampe, `min` and `max` are useful for defining penalty functions but cannot be used directly, though they can often be approximated with relaxed versions. Graphics applications often feature other complex, highly-discontinuous functions, such as rendering and collision. These functions might also be similarly relaxed through *smooth interpretation*, a technique for automatically deriving a smooth, differentiable approximation of programs [CSL10].

In our evaluations, we validated the basic usefulness of HMC for design suggestion, but further studies are needed to understand how HMC can fully integrate into the probabilistic computational design ecosystem. For instance, HMC—like most core MCMC samplers—cannot reliably make large jumps between disconnected modes in the design space. How does it fare at mode-switching when coupled with parallel tempering? Does it enable more efficient dimension-jumping when used as the annealing kernel for LARJ-MCMC?

Ultimately, HMC is just one new tool in the toolbox of probabilistic design techniques. There are still many other problems that need general-purpose solutions. Complex hard constraints remain challenging; sufficiently tight soft constraints may be acceptable, as in the case of our stacking equilibrium examples, but this will not always be the case. An extension to HMC that explicitly adheres to a manifold may provide a good solution [BSU12]. Complex constraints on discrete variables are also difficult—integrating SAT solvers into MCMC methods may provide some traction here.

If we can solve these problems, then we can envision a future where any developer can use general-purpose probabilistic programming to build computational design tools with powerful, high-level creative controls. We believe that such systems can have a huge impact not only in computer graphics, but also in engineering and other disciplines where design plays a crucial role.

Acknowledgments

Support for this research was provided by Intel (ISTC-VC). This material is based on research sponsored by DARPA under agreement number FA8750-14-2-0009. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. The photographs in Figure 7 are provided by Flickr users *kowitz* and *bastique* under the Creative Commons CC BY-SA 2.0 license (`https://creativecommons.org/licenses/by-sa/2.0/`) The vector art images in Figures 4 and 5 are from `https://openclipart.org` and are in the public domain.

References

[Ber10] BERGMAN R.: *Wood Handbook âĂŞ Wood as an Engineering Material*. Forest Products Laboratory, 2010. 10

[BSU12] BRUBAKER M. A., SALZMANN M., URTASUN R.: A family of mcmc methods on implicitly defined manifolds. In *Proc. AISTATS 2012* (2012). 10

[BYMW13] BAO F., YAN D.-M., MITRA N. J., WONKA P.: Generating and exploring good building layouts. In *Proc. SIGGRAPH 2013* (2013), SIGGRAPH '13. 2

[CFG*01] CORLISS G., FAURE C., GRIEWANK A., HASCOËT L., NAUMANN U.: *Automatic Differentiation: From Simulation to Optimization*. Computer and Information Science. Springer, 2001. 5

[CSL10] CHAUDHURI S., SOLAR-LEZAMA A.: Smooth interpretation. In *Proc. PLDI 2010* (2010). 10

[DGK*10] DOW S. P., GLASSCO A., KASS J., SCHWARZ M., SCHWARTZ D. L., KLEMMER S. R.: Parallel prototyping leads to better design results, more divergence, and increased self-efficacy. *ACM Trans. Comput.-Hum. Interact. 17*, 4 (2010). 1

[DHA*13] DEVITO Z., HEGARTY J., AIKEN A., HANRAHAN P., VITEK J.: Terra: A multi-stage language for high-performance computing. In *Proc. PLDI 2013* (2013). 2, 5

[DKPR87] DUANE S., KENNEDY A., PENDLETON B. J., ROWETH D.: Hybrid monte carlo. *Physics Letters B 195*, 2 (1987), 216 – 222. 4

[FRS*12] FISHER M., RITCHIE D., SAVVA M., FUNKHOUSER T., HANRAHAN P.: Example-based synthesis of 3d object arrangements. In *Proc. SIGGRAPH Asia 2012* (2012), SA '12. 2

[GMR*08] GOODMAN N. D., MANSINGHKA V. K., ROY D. M., BONAWITZ K., TENENBAUM J. B.: Church: a language for generative models. In *Proc. of UAI 2008* (2008). 2

[Gue07] GUENTER B.: Efficient symbolic differentiation for graphics applications. In *Proc. SIGGRAPH '07'* (2007), SIGGRAPH '07. 5

[HG14] HOFFMAN M. D., GELMAN A.: The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research* (2014). 5

[HZSD] HUANG Q., ZHANG J., SABBAGHI A., DASGUPTA T.: Optimal offline compensation of shape shrinkage for 3d printing processes. *IIE Transactions on Quality and Reliability*. 3

[JM12] JAKOB W., MARSCHNER S.: Manifold exploration: a markov chain monte carlo technique for rendering scenes with difficult specular transport. In *Proc. SIGGRAPH 2012* (2012), SIGGRAPH '12. 2

[JS91] JANSSON D., SMITH S.: Design fixation. *Design studies 12*, 1 (1991). 1

[JTRS12a] JAIN A., THORMÄHLEN T., RITSCHEL T., SEIDEL H.-P.: Exploring shape variations by 3d-model decomposition and part-based recombination. *Comp. Graph. Forum 31*, 2pt3 (2012). 2

[JTRS12b] JAIN A., THORMÄHLEN T., RITSCHEL T., SEIDEL H.-P.: Material memex: Automatic material suggestions for 3d objects. In *Proc. SIGGRAPH Asia 2012* (2012), SA '12. 2

[KCGN98] KASS R. E., CARLIN B. P., GELMAN A., NEAL R. M.: Markov chain monte carlo in practice: A roundtable discussion. *The American Statistician 52*, 2 (1998). 4

[KCKK12] KALOGERAKIS E., CHAUDHURI S., KOLLER D., KOLTUN V.: A probabilistic model for component-based shape synthesis. In *Proc. SIGGRAPH 2012* (2012), SIGGRAPH '12. 2

[KDK14] KULKARNI C., DOW S. P., KLEMMER S. R.: Early and repeated exposure to examples improves creative work. In *Design Thinking Research*. Springer, 2014. 1

[LR04] LEIMKUHLER B., REICH S.: *Simulating Hamiltonian Dynamics*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2004. 4

[LRFH13] LIN S., RITCHIE D., FISHER M., HANRAHAN P.: Probabilistic color-by-numbers: Suggesting pattern colorizations using factor graphs. In *Proc. SIGGRAPH 2013* (2013), SIGGRAPH '13. 1, 3, 5, 12

[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. In *Proc. SIGGRAPH 1997* (1997), SIGGRAPH '97. 2

[MRR*53] METROPOLIS N., ROSENBLUTH A. W., ROSENBLUTH M. N., TELLER A. H., TELLER E.: Equation of State Calculations by Fast Computing Machines. *The Journal of Computational Physics 21* (June 1953). 3

[MSL*11] MERRELL P., SCHKUFZA E., LI Z., AGRAWALA M., KOLTUN V.: Interactive furniture layout using interior design guidelines. In *Proc. SIGGRAPH 2011* (2011), SIGGRAPH '11. 2, 3

[Nea10] NEAL R. M.: MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo* (2010). 2, 3, 4

[OAH11] O'DONOVAN P., AGARWALA A., HERTZMANN A.: Color Compatibility From Large Datasets. *ACM Transactions on Graphics* (2011). 5

[OL06] OU L.-C., LUO M. R.: A colour harmony model for two-colour combinations. *Color Research & Application* (2006). 5

[PWLSH13] PRÉVOST R., WHITING E., LEFEBVRE S., SORKINE-HORNUNG O.: Make It Stand: Balancing shapes for 3D fabrication. In *Proc. SIGGRAPH 2013* (2013). 7

[RGG97] ROBERTS G. O., GELMAN A., GILKS W. R.: Weak convergence and optimal scaling of random walk metropolis algorithms. *The Annals of Applied Probability 7*, 1 (1997). 5

[Rit14] RITCHIE D.: Quicksand: Low-level probabilistic programming in Terra. http://dritchie.github.io/quicksand, 2014. 2, 3, 5

[SHOW02] SMITH J., HODGINS J., OPPENHEIM I., WITKIN A.: Creating models of truss structures with optimization. In *Proc. SIGGRAPH 2002* (2002). 7

[Spe80] SPEELPENNING B.: *Compiling Fast Partial Derivatives of Functions Given by Algorithms*. PhD thesis, Champaign, IL, USA, 1980. 5

[Sta14] STAN DEVELOPMENT TEAM: *Stan Modeling Language Users Guide and Reference Manual, Version 2.2*, 2014. URL: http://mc-stan.org/. 3, 5

[SW14] SCHWARTZ M., WONKA P.: Procedural design of exterior lighting for buildings with complex constraints. *ACM Transactions on Graphics* (2014). 2

[TLL*11] TALTON J. O., LOU Y., LESSER S., DUKE J., MĚCH R., KOLTUN V.: Metropolis procedural modeling. *ACM Trans. Graph. 30*, 2 (2011). 2, 3

[UIM12] UMETANI N., IGARASHI T., MITRA N. J.: Guided exploration of physically valid shapes for furniture design. In *Proc. SIGGRAPH 2012* (2012), SIGGRAPH '12. 2, 12

[WGSS11] WINGATE D., GOODMAN N. D., STUHLMÜLLER A., SISKIND J. M.: Nonstandard interpretations of probabilistic programs for efficient inference. In *Proc. NIPS 2011* (2011). 3

[WOD09] WHITING E., OCHSENDORF J., DURAND F.: Procedural modeling of structurally-sound masonry buildings. In *Proc. SIGGRAPH Asia 2009* (2009). 7, 12

[XZCOC12] XU K., ZHANG H., COHEN-OR D., CHEN B.: Fit and diverse: Set evolution for inspiring 3d shape galleries. In *Proc. SIGGRAPH 2012* (2012), SIGGRAPH '12. 2

[YBY*13] YEH Y.-T., BREEDEN K., YANG L., FISHER M., HANRAHAN P.: Synthesis of tiled patterns using factor graphs. *ACM Trans. Graph. 32*, 1 (2013). 2

[YYPM11] YANG Y.-L., YANG Y.-J., POTTMANN H., MITRA N. J.: Shape space exploration of constrained meshes. In *Proc. SIGGRAPH Asia 2011* (2011), SA '11. 2

[YYW*12] YEH Y.-T., YANG L., WATSON M., GOODMAN N. D., HANRAHAN P.: Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. In *Proc. SIGGRAPH 2012* (2012), SIGGRAPH '12. 1, 2, 5

[ZRD*13] ZUCKER M., RATLIFF N., DRAGAN A., PIVTORAIKO M., KLINGENSMITH M., DELLIN C., BAGNELL J. A. D., SRINIVASA S.: Chomp: Covariant hamiltonian optimization for motion planning. *International Journal of Robotics Research* (May 2013). 3

Appendix

Color Compatibility Model

Our color compatibility model uses soft constraints simplified from the model by Lin and colleagues [LRFH13]. We include saturation, adjacent lightness difference, and adjacent perceptual difference constraints, since these factors had high learned weights in the original model. The perceptual difference constraints (implemented as distance in CIELAB color space) help distinguish image regions without excessive hue contrast. The lightness difference constraints help prevent equiluminant adjacent regions which can cause perceived "vibrations" and unstable-looking shapes. Constraints are parameterized differently if they are applied to foreground (FG) or background (BG) regions:

Saturation (BG): $\texttt{softeq}(\frac{\sqrt{a^2+b^2}}{\sqrt{a^2+b^2+L^2}}, 0.3, 1.0)$

Saturation (FG): $\texttt{softeq}(\frac{\sqrt{a^2+b^2}}{\sqrt{a^2+b^2+L^2}}, 0.7, 1.0)$

Lightness Diff (FG-BG): $\texttt{softeq}(\frac{|\Delta L|}{100}, 0.3, 0.4)$

Lightness Diff (FG-FG): $\texttt{softeq}(\frac{|\Delta L|}{100}, 0.2, 0.4)$

Perceptual Diff: $\texttt{softeq}(\frac{\sqrt{\Delta L^2+\Delta a^2+\Delta b^2}}{300}, 0.3, 0.2)$

where L, a, b are the color coordinates of a region in CIELAB color space, and ΔL, Δa, Δb are differences between the coordinates of adjacent regions. 100 is the maximum lightness value, and 300 is the maximum CIELAB distance. The rough shape of these constraints are based on those in the original model. Our model is a weighted sum of these factors for each region and each pair of adjacent regions. Saturation constraints are weighted by the region area, while pairwise adjacent constraints are weighted uniformly. As in the Lin et al. model, we represent and perform inference on color random variables in RGB space to ensure that all generated results use colors that can be visualized.

We also add constraints to enforce semantic properties where needed. We consider two types of additional constraints in our experiments:

Same-Chroma $\texttt{softeq}(\frac{\sqrt{\Delta a^2+\Delta b^2}}{282.9}, 0, \frac{\sigma}{282.9})$

Lightness-Relation $\texttt{softeq}(\frac{\Delta L}{100}, 0.15, \frac{\sigma}{100})$

where 282.9 is the maximum chroma difference. We set σ to 5 in our experiments. The Same-Chroma constraint dictates that two colors should have the same chromatic content (i.e. the same color irrespective of lightness). The Lightness-Relation constraint enforces a precise directional separation between the lightnesses of two colors; this constraint is useful for constraining colors to be shades of one another.

Block Statics Model

In our statics model, blocks are assembled into structures via *contacts*: wherever two blocks touch, some internal force distribution arises over the resulting rectangular contact region. We represent this distribution with forces at each of the contact region's four vertices:

- f_n: a compressive force normal to the contact region.
- f_{t1} and f_{t2}: two friction forces tangent to the contact region.

f_n is bounded to be non-negative, and f_{t1} and f_{t2} are bounded to be within $|s \cdot f_n|$, where s is the coefficient of static friction of the contact. We use $s = 0.5$ for all results presented in this paper unless noted otherwise. This statics model is essentially the same as that used by prior work on stable procedural buildings [WOD09]. It is important to note that because friction force directions are treated as free variables, this model is not strictly physically accurate; correct handling of frictional contacts in a statics context is still a challenging problem [UIM12].

Given this statics model, the process for generating a stable structure is straightforward. The user first writes a program that generates a random block structure, e.g. by iteratively stacking and perturbing random blocks. Our system then computes the net force $\bar{f}$ and net torque $\bar{\tau}$ on the center of mass of each block i and combines these 'residuals' into the following factor:

$$\sum_i \texttt{softeq}(||\bar{f}_i||, 0, \sigma_f) + \texttt{softeq}(||\bar{\tau}_i||, 0, \sigma_\tau)$$

The bandwidths σ_f and σ_τ must be set in an appropriately scale-invariant fashion, so that large structures are not penalized more than small ones. One option is to define them as percentages of the average external (i.e. due to gravity) force and torque acting on the structure. We find that 1% tolerance keeps an HMC sampler sufficiently close to the static equilibrium manifold while still allowing for exploration.