Metropolis Procedural Modeling

Johannes Agricola

March 8, 2013

Abstract

Many procedural modeling schemes suffer from the weakness of the inability to be easily controlled. This report focuses on the presentation of an algorithm by Talton, Lou, Lesser and Duke that employs a Markov chain Monte Carlo based algorithm to force models generated from a procedural description to follow another specification. This specification may be of a higher level than the underlying description and is therefore able to exert control over all levels of abstractions that can be computed from the model description. It will be shown that the resulting algorithm is only restricted by the availability of a grammar and a function that evaluates a generated model in reference to the high-level specification and therefore applicable to a multitude of scenarios.

1 Introduction

Procedural modeling, the generation of models structured by algorithms, offers distinctive advantages to modern graphical systems. They possess the ability to model scenes of high detail and complex structure without the need for human intervention. Due to the nature of the algorithmic description, this can be achieved with very compact definitions.

Therein, however, lies a bothersome weakness of this approach: A small change in the generation procedure tends to lead to a perturbation that may unforseeably alter the generated models. Controlling this is thus tied to a considerable effort for the designer and therefore effectively cancels out the advantages of the approach.

Instead of trying to control the model by expensively changing the algorithmic specification, one can think of providing another high level specification that is evaluated during generation. This can then supply a measure to the generator how well it is doing in matching the requirements and thus guide to the desired outcome without interfering with the structural definition.

Talton et al. [4] suggest such a method for parametric, stochastic, conditional, context-free grammars as procedural definition and any high-level input specification that is able to provide a measure to the models generated from the aforementioned grammar. This specification may cover all properties of the generated models, or words, including shapes or complex analytical parameters. They achieve this with an additional function which returns the similarity between the input specification and the generated model. Given this function, the proposed algorithm performs a maximization of the posterior over the space of all words derived from the grammar.

Intuition already suggests that the space of concern is of complex structure and thus the choice of an algorithm has to be considered carefully. Markov chain Monte Carlo (MCMC) as formulated by Metropolis et al. [3] and Hastings [2] however possesses the ability to perform maximation on such a space. One exception to that is its transdimensional structure: The MCMC algorithm requires the space to have fixed dimension, while the space of all derivations from a grammar has subspaces of varying dimensionality.

Talton et al. therefore choose to make use of Green's MCMC generalization Reversible Jump Markov chain Monte Carlo (RJMCMC) [1]. In it he models the choice between the different subspaces as a bayesian inference problem and thus allows to create an MCMC variant that is able to jump between samples of different dimension.

2 Grammars

Before examining the proposed algorithm, a formal introduction to the employed grammars is inevitable, which shall be motivated here first. The context-free grammars used in the main paper of this report are also and especially classified as parametric, stochastic and conditional. They are parametric in the sense that each terminal can be assigned several *descriptive parameters* that provide an additional level of detail. In turn, *formal parameters* may be assigned to non-terminals that govern the productions which will be applied during succeeding inference steps. In particular the start symbol or *axiom* may supply parameters to the initial non-terminal which then propagate during inference.

In contrast to arithmetically derived parameters, they may also be sampled from a probability distribution, hence probabilistic grammar. Extending this characteristic is that the probability with which each production should be used during inference has to be defined. This is especially necessary as it is required for RJMCMC that the probability for each word derived from the grammar has to be well-defined.

A parametric, stochastic, conditional, context-free grammar is a tuple

$$G = (V, T, \Sigma, \omega, P)$$

where V is the set of *variables* (or nonterminals), T is the set of *terminals*, Σ is the set of *formal parameters*. $\omega \in ((V \cup T) \times \mathbb{R}^*)^+$ is the axiom consisting of terminals and initial variables and their parameters. At last

$$P \subseteq \overbrace{(V \times \Sigma^* \times \mathcal{B} \times \mathcal{E})}^{\text{predecessor } \psi \in} \times \overbrace{((V \cup T) \times \mathcal{E}^*)^*}^{\text{successor } \chi \in} \tag{1}$$

is the set of productions. $\mathcal{E}$ is the set of all valid arithmetic expressions with parameters from Σ. $\mathcal{B}$ is the set of boolean expressions constructed with elements from $\mathcal{E}$, which is the set of all arithmetic expressions that can be formulated from the formal parameters.

The meaning behind this is that the predecessor ψ, defined by a non-terminal (V) and its formal parameters (Σ^*) can be replaced under the condition from $\mathcal{B}$ and with probability derived from elements of $\mathcal{E}$ by a successor χ. In practice, the value derived from $\mathcal{E}$ may have to be normalized before that. The successor itself again consists of terminals, non-terminals and formal or descriptive parameters.

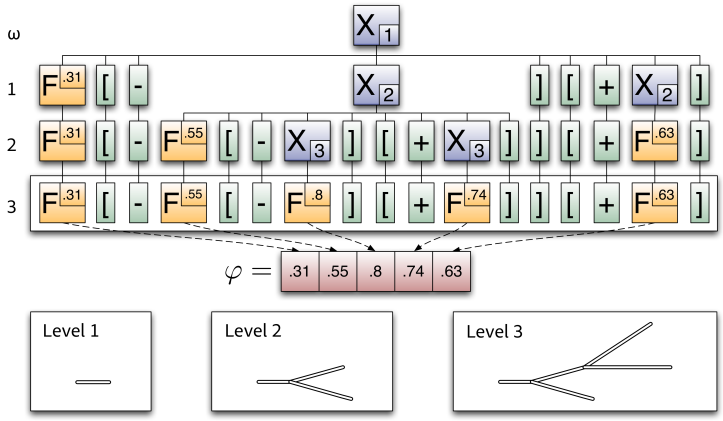

Figure 1: Example derivation tree for the example grammar (From [4]).

The example provided by Talton et al. for this is a simple branching model. With $V = \{X\}$, $T = \{F, +, -, [,]\}$, $\Sigma = \{l\}$, $\omega = X(1)$ the production rules are

$$P = \begin{Bmatrix} X(l) : l \leq 3 & \xrightarrow{1-l/3} & F(\mathcal{N}(l, 0.2))[-X(l+1)][+X(l+1)], \\ X(l) : l \leq 3 & \xrightarrow{l/3} & F(\mathcal{N}(l, 0.2)) \end{Bmatrix},$$

where $F\left(\mathcal{N}\left(\mu, \sigma^2\right)\right)$ means that the parameter of F should be sampled from a normal distribution with mean μ and variance σ^2.

An example derivation for this grammar can be seen in Figure 1. A starting branch is first split into two branches, one of which is finalized and the other is split once more. Each of the terminal symbols which are interpreted as branches has one descriptive parameter, namely its length. This example will also be used throughout this document to illustrate the behaviour of the proposed algorithm.

Each descriptive parameter φ_t from the set of all descriptive parameters of one model φ is drawn from its own distribution Φ_t. φ and the whole derivation tree τ constitute a derivation $\delta = (\tau, \varphi)$, which is an element of the space of all possible derivations $\Delta(G)$ from a Grammar G.

2.1 Probabilistic Analysis of Inference

The definitions established previously explicitly specify probabilities for choices that occur during traditional inference. Thus they allow the use of Monte Carlo methods on languages specified by grammars of this type. The next step is to enable the algorithm to walk on the space of all possible derivations $\Delta(G)$ of a Grammar G. For this it is necessary to evaluate how a specific step influences the quality of the model.

Therefore the objective at this point is to find a measure that compares an element $\delta \in \Delta(G)$ to a specification I. This is given by the posterior

$$p\left(\delta | I\right) \propto L\left(I | \delta\right) \cdot \pi\left(\delta\right). \tag{2}$$

It is not necessary to know the exact value of p for a specific model as it is only going to be used to compare models to one another.

In Eq. 2, $\pi(\delta)$ is the model prior, the probability that the model δ has been drawn from $\Delta(G)$. This is proportional to the probability to choose the necessary productions and the probability to draw the respective descriptive parameters:

$$\pi(\delta) \propto \prod_{s \in \delta} P(s|\text{parent}(s)) \cdot \prod_{t \in \delta_{\Rightarrow}} \Phi_t(\varphi_t)$$

where $P(s|\text{parent}(s))$ is the probability of producing the symbol s from the parent in the derivation tree and $\delta_{\Rightarrow}$ are the leafs of δ.

Eq. 2 also requires a definition of $L(I|\delta)$, the likelihood of an input I given a model δ. Such a likelihood function has to be specified for each model characteristic which is supposed to be controlled.

3 Markov chain Monte Carlo

Markov chain Monte Carlo was originally used to compute the state of matter by Metropolis et al. [3]. A system of discs in a box which were assigned a potential was allowed to perform several *diffusion* steps. After each step the cumulative energy of the system was evaluated to search for the lowest energy state, so steps which lower the energy were used as a basis for future steps. However, to avoid local minima, even steps that result in a higher energy were accepted with a probability that was lower the higher the energy increment became.

The method has since evolved and is used in a variety of applications. Fundamental for this is the generalization provided by Hastings [2]. A Markov chain, the basis for this formulation, is a sequence of random variables $X_1, X_2, \ldots, X_S$ where the distribution of each of these variables depends only on its predecessor,

$$P(X_i = x_i | X_{i-1} = x_{i-1}) = P(X_i = x_i | X_{i-1} = x_{i-1}, X_{i-2} = x_{i-2}, \ldots).$$

Further $P(X_i = j | X_{i-1} = i) = b_{ij}$ is called the transition matrix of the Markov chain. If the chain is constructed such that a probability distribution $p = (p_1, \ldots, p_S)$ exists with $p = pB$, it is called its *stationary distribution*.

This allows to easily estimate a function like

$$I = \sum_{x=1}^{S} f(x) \cdot p_x,$$

which in turn may already be an estimate for $\int f(x)p(x)\mathrm{d}x$, a base problem for Monte Carlo methods, by simulating the Markov chain in N steps. For $N \to \infty$ this is

$$I \approx \frac{1}{N} \cdot \sum_{i=1}^{N} f(X_i).$$

Of higher importance here is that it also allows to perform maximization by $\max p(x) \approx \text{argmax}_i\, p(X_i)$ — a Markov chain formalization of Metropolis' method.

The *Metropolis-Hastings* algorithm makes it easy to fulfill the assumption of the existence of a stationary distribution. Instead of working directly on the

distribution p which is to be sampled, it takes another distribution, the proposal distribution with density function $q(\cdot|\cdot)$. For a diffusion step from x_i to x_{i+1}, a tentative sample $x^* \sim q(\cdot|x_i)$ is drawn. The sample is then accepted as x_{i+1} with a probability

$$\alpha_{x_i \to x^*} = \min\left\{1, \frac{p(x^*)}{p(x_i)} \frac{q(x_i|x^*)}{q(x^*|x_i)}\right\}. \tag{3}$$

Otherwise another sample is drawn. The performance of the algorithm will now obviously depend on the difficulty to sample from q. Additionally it will most likely improve with fewer differences between p and q, as this facilitates fewer moves that have largely different probabilities of occurring when sampling from q versus sampling from p.

3.1 Reversible Jump Markov chain Monte Carlo

At this point it is important to note that to evaluate the next step, only comparisons to the previous step were made. This relative nature of the algorithm allows on one hand to work with a non-normalized p. In fact p can be any function that maps the state space of the Markov chain into the positive real numbers $p : \chi \to \mathbb{R}^+$. On the other hand the algorithm ceases to work as soon as there is no meaning behind comparing two values from p. This is for example the case, once the number of parameters for p, the dimension of the model, changes. In the previously prepared space of derivations $\Delta(G)$ this is the case. Every word derived from G may have different length and thus a different amount of parameters.

To eliminate this issue, Green proposed additional *jump* moves which constitute the Reversible Jump MCMC [1] (RJMCMC) algorithm. In contrast to the already established diffusion moves, they are attentively constructed in a way that they allow moving between samples of different dimension. In our case this would, at the simplest level, allow to merge two terminals into one or split one terminal into two whilst minimizing parameter changes.

This was originally supplemented with an example from step function analysis. Performing such an analysis task with the Metropolis-Hastings algorithm requires a preselection of the amount of steps which the model to be fitted should have while the number of steps may be varying in general. The jump move resolves this by switching between models from the more generic class of all step function models.

For now let us assume that we have a countable set of models $\{\mathcal{M}_m\}$. k_m denotes the amount of parameters of $\mathcal{M}_m$. An instance of these parameters is $x_m \in \chi_m \subseteq \mathbb{R}^{k_m}$ with χ_m being the parameter space of $\mathcal{M}_m$. A specific sample is then identified by (m, x_m). The algorithm then operates on the domain $\chi = \bigcup_m \{m\} \times \chi_m$.

The goal at this point is to model jump moves within the formalism of the Metropolis-Hastings algorithm. Therefore it has to satisfy ergodicity. Ergodicity requires that it is possible to reach any state of the chain within a finite number of expected steps. The way chosen by Green to do that is to ensure reversibility of each jump move: For each jump from a model $(m, x_m) \in \chi$ to $(n, x_n) \in \chi$ a move from (n, x_n) to (m, x_m) has to be possible. By this mechanism it is impossible that the chain gets trapped at particular dimensionalities.

Secondly it is necessary to map x_m to x_n. As the models $\mathcal{M}_m$ and $\mathcal{M}_n$ have different amounts of parameters which in addition may very well also differ in their semantics, it is necessary to think about how to map the parameters from the model before the jump move to the new model afterwards. In general, this is done by the *dimension matching function*. The idea behind this function is to find a common parameter space which includes both model parameter spaces as subspaces and then perform the map in this space:

$$f_{m\to n} : \chi_m \times \mathcal{U}_{m,n} \to \chi_n \times \mathcal{U}_{n,m}$$

with $\mathcal{U}_{m,n} \subseteq \mathbb{R}^{r_m}$, $\mathcal{U}_{n,m} \subseteq \mathbb{R}^{r_n}$ and to ensure the same dimension

$$k_n + r_n = k_m + r_m.$$

Formally, for a jump from $\mathcal{M}_m$ to $\mathcal{M}_n$, the parameters x_m are supplemented with paramters $u_m \sim q_{m\to n}(\cdot|m, x_n)$. The dimension matching function then gives

$$(x_n, u_n) = f_{m\to n}(x_m, u_m).$$

Finally, as with the diffusion move, the accepance probability has to be defined. A state $(x_n^*, u_n^*) = f_{n\to m}\,(x_m, u_m)$ is accepted as next state with the probability

$$\alpha_{m\to n} = \min\left\{1, \frac{p(n, x_n^*)}{p(m, x_m)} \cdot \frac{j(m|n)}{j(n|m)} \cdot \frac{q_{n\to m}(u_n^*|n, x_n^*)}{q_{m\to n}(u_m|m, x_m)} \left|\det \frac{\partial f_{m\to n}(x_n, u_n)}{\partial(x_n, u_n)}\right|\right\}. \tag{4}$$

The new factor $j(n|m)$ is the probability of performing a jump from $\mathcal{M}_m$ to $\mathcal{M}_n$.

This concludes the introduction of RJMCMC. In the next section its application to Grammars as presented by Talton et al. will be shown.

4 RJMCMC applied to Grammars

For an application to Grammars let us again assume the existence of a Grammar G, the input specification I and the corresponding likelihood function $L(I|\delta)$. At first the initial state of the Markov chain is chosen by deriving one word from G, according to the rules laid down in section 2, and thereby obtaining $(\tau, \varphi) =: \delta$. Initially $\delta \sim \pi(\cdot)$. From this point, the Markov chain is started. For every new state, it has to be decided whether to perform a diffusion or jump move.

Diffusion. In diffusion moves, the derivation tree τ is left untouched, only the leaf node parameters φ are allowed to vary. The changes to φ are applied by randomly selecting one parameter φ_t and then generate a proposal state δ' by resampling $\varphi_t' \sim q(\delta|\delta') = q(\delta) = \Phi_t$. An example for such a move is depicted in Figure 2. Putting this into Eq. 3 with Eq. 2 gives the acceptance probability

$$\alpha_{\delta\to\delta'} = \min\left\{1, \frac{p(\delta'|I)}{p(\delta|I)}\frac{q(\delta|\delta')}{q(\delta'|\delta)}\right\} = \min\left\{1, \frac{p(\delta'|I)\Phi_t(\varphi_t)}{p(\delta|I)\Phi_t(\varphi_t')}\right\} = \min\left\{1, \frac{L(I|\delta')}{L(I|\delta)}\right\}$$

for a diffusion move.

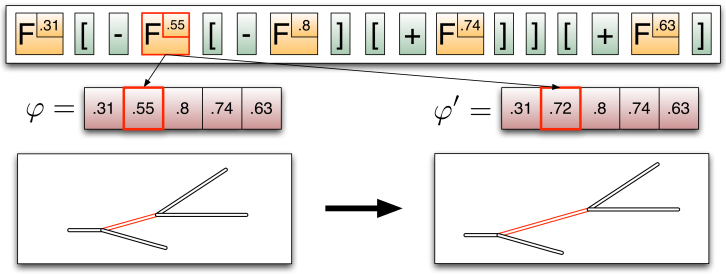

Figure 2: Example diffusion move: One descriptive parameter that represents a branch length has been selected and resampled which resulted in a longer branch (From [4]).

Jump. The jump move on the other hand tries to hold on to φ while modifying τ. This is again done by randomly selecting any nonterminal $v \in \tau$ and subsequently rederive the subtree rooted in v, τ_v, to generate τ'. Figure 3 shows how this may look like. This has to be modeled in a reversible fashion.

The first concern at this point is the dimension matching function. In preparation for a formulation of this function, a transformation of the parameters φ is introduced. Instead of looking at $\varphi \sim \Phi$, Talton et al. make use of the fact that sampling from any distribution Φ is equivalent to sampling from a set of uniform distributions and the application of a transformation. So without loss of generality we replace $\varphi \in \mathbb{R}^q$ with $\widetilde{\varphi} \in [0,1]^w$, $w \geq q$ for the remainder of this section.

With now all parameters within $[0,1]$, the formulation of a dimension matching function is straightforward. As the aforementioned transformations are defined by the model, for a jump with increasing number of elements in $\widetilde{\varphi}$, additional values u are sampled from $U_{[0,1]}$

$$\widetilde{\varphi}' = f_{m\to n}(\widetilde{\varphi}, u) = (\widetilde{\varphi}, u),$$

for decreasing number of elements, they are simply dropped

$$(\widetilde{\varphi}', u') = f_{m\to n}(\widetilde{\varphi}) = \widetilde{\varphi}.$$

With careful selection of the parameters to be dropped and the insertion positions this is easy to implement while modifying φ only as necessary. The reversibility of this is obvious, as

$$f_{n\to m}\left(f_{m\to n}\left(\widetilde{\varphi}\right)\right) = \widetilde{\varphi}.$$

The jacobian, which will be required for the acceptance probability, also simplifies nicely to

$$\left|\det \frac{\partial f_{\delta\to\delta'}(\widetilde{\varphi}, u)}{\partial(\widetilde{\varphi}, u)}\right| = \left\{\begin{array}{l} \left|\det \frac{\partial(\widetilde{\varphi}')}{\partial(\widetilde{\varphi}, u)}\right| \text{ for decreasing dimension} \\ \left|\det \frac{\partial(\widetilde{\varphi}', u')}{\partial(\widetilde{\varphi})}\right| \text{ for increasing dimension} \end{array}\right\} = 1.$$

To compute the acceptance probability, the probability of selecting a specific jump $j(\cdot|\cdot)$ is required. If the jump replaces the subtree τ_v by τ'_v, this is given

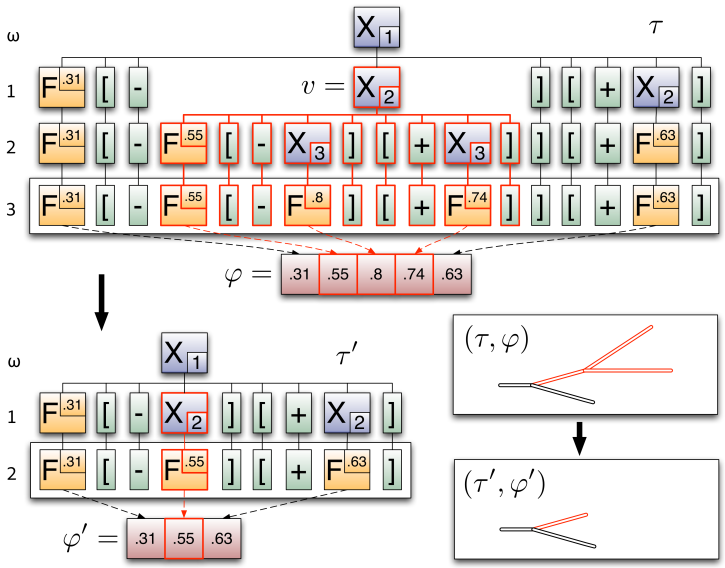

Figure 3: Example jump move: One terminal v from the original branch example (see Figure 1) has been selected. Rederivation of the subtree rooted at it leads to a completely different model with reduced dimension. (From [4]).

by

$$j\left(\tau'_v|\tau_v\right) = q_\tau(v) \cdot \prod_{s \in \tau'_v} P(s|\text{parent}(s))$$

with the probability of selecting the node v as subtree root $q_\tau(v)$.

The acceptance probability can then be derived from Eq. 4 as

$$\begin{aligned}
\alpha_{\delta \to \delta'} &= \min\left\{1, \frac{p(\delta'|I)}{p(\delta|I)} \cdot \frac{j(\tau_v|\tau'_v)}{j(\tau'_v|\tau_v)} \cdot \frac{U_{[0,1]}(u')}{1} \cdot \left|\det \frac{\partial f_{n\to m}(\widetilde{\varphi}', u')}{\partial(\widetilde{\varphi}, u)}\right|\right\} \\
&= \min\left\{1, \frac{p(\delta'|I)}{p(\delta|I)} \cdot \frac{j(\tau_v|\tau'_v)}{j(\tau'_v|\tau_v)} \cdot \frac{U_{[0,1]}(u')}{1}\right\} \\
&= \min\left\{1, \frac{p(\delta'|I)}{p(\delta|I)} \cdot \frac{j(\tau_v|\tau'_v)}{j(\tau'_v|\tau_v)}\right\} \qquad \because U_{[0,1]}(u') = 1 \\
&= \min\left\{1, \frac{p(\delta'|I)}{p(\delta|I)} \cdot \frac{q_{\tau'}(v) \cdot \prod_{s\in\tau_v} P(s|\text{parent}(s))}{q_\tau(v) \cdot \prod_{s\in\tau'_v} P(s|\text{parent}(s))}\right\} \\
&= \min\left\{1, \frac{L(I|\delta')\pi(\delta')}{L(I|\delta)\pi(\delta)} \cdot \frac{q_{\tau'}(v) \cdot \prod_{s\in\tau_v} P(s|\text{parent}(s))}{q_\tau(v) \cdot \prod_{s\in\tau'_v} P(s|\text{parent}(s))}\right\} \\
&= \min\left\{1, \frac{L(I|\delta')}{L(I|\delta)} \cdot \frac{q_{\tau'}(v)}{q_\tau(v)} \cdot \frac{\prod_{t\in\tau'_v} \Phi_t(\varphi_t)}{\prod_{t\in\tau_v} \Phi_t(\varphi'_t)}\right\} \qquad (5)
\end{aligned}$$

After the acceptance probabilities for the respective move have been calculated, a random value $k \sim U_{[0,1]}$ is sampled. If $k < \alpha$, the δ' is accepted as

Input : Grammar G, likelihood function $L(I|\delta)$,
target specification I, computational budget N.
Output: a MAP estimate of δ

sample $\delta_0 \sim \pi_G$
$\delta_{max} \leftarrow \delta_0$; $max \leftarrow p(\delta_0|I)$
for $n \leftarrow 1$ **to** N **do**
  switch `RandomMove()` **do**
    case *diffusion*
      $t \leftarrow$ `RandomTerminal()`
      $\delta'_n \leftarrow \delta_{n-1}$
      resample $\varphi'_t \sim \Phi_t$ from δ'_n
      $\alpha \leftarrow \min\left\{1, \frac{L\left(I|\delta'_n\right)}{L\left(I|\delta_{n-1}\right)}\right\}$
    case *jump*
      sample $v \sim q_{\delta_{n-1}}$
      $\tau'_v \leftarrow$ `Rederive(`τ_v`)`
      $\delta'_n \leftarrow$ `DimensionMatch(`τ'_v, τ_v`)`
      $\alpha \leftarrow \min\left\{1, \frac{L(I|\delta'_n)}{L(I|\delta_{n-1})} \cdot \frac{q_{\tau'}(v)}{q_\tau(v)} \cdot \frac{\prod_{t\in\tau'_v}\Phi_t(\varphi_t)}{\prod_{t\in\tau_v}\Phi_t(\varphi'_t)}\right\}$
  sample $k \sim [0,1]$
  if $k < \alpha$ **then**
    $\delta_n \leftarrow \delta'_n$
  else
    $\delta_n \leftarrow \delta_{n-1}$
  if $max < p(\delta_n|I)$ **then**
    $\delta_{max} \leftarrow \delta_n$, $max \leftarrow p(\delta_n|I)$
return δ_{max}

Algorithm 1: Metropolis Procedural Modeling

the next value for δ, the current position of the chain. Otherwise δ remains unchanged. Afterwards the next move is performed.

During this time the algorithm keeps track of the model δ_{max} with maximum $p\left(\delta_n|I\right)$. After the assigned computational budget has been used, δ_{max} contains the result. Algorithm 1 is the derived algorithm.

4.1 Optimizations

Talton et al. discuss several optimizations that can be applied to this algorithm. I will briefly sketch three of their suggestions.

Nonterminal selection. The initial idea of selecting the node at which a rederivation is started for a jump move uniformly, is subject to bias towards leaf nodes due to the exponential increase of nodes per tree level. Selecting those nodes results in only small changes in the tree per jump move. Prebiasing the selection towards the root node substantially increases mixing for jump moves.

Parallel tempering. They also make use of parallel tempering. The name stems again from the original problem used by Metropolis et al. and refers to running the chain (or system) at different temperatures which is equivalent to tuning the acceptance probabilities to accept more or less moves and running several chains at different tunings.

Delayed rejection. Another improvement is delaying rejection of tentative samples directly after a jump move. Such a sample may have skewed parameters as the ones from a previous model were mapped 1:1. Allowing the chain to perform an additional diffusion move may compensate for particularly bad maps.

5 Implementation

A modeling system that uses the presented approach has been created. Most of it is written in C++, with the exception of Grammars. These are written in a specialized language which is then translated to C++ and compiled into dynamically loadable libraries. This enables the usage of native data types and reduces the overhead of Grammar processing during run time compared to a purely interpretative approach.

Apart from the modeling core, grammars and likelihood functions for different model and input specification classes have been implemented. This section will enumerate these.

Figure 4 shows random derivations from these grammars. No input specification has been applied to show the natural results.

5.1 Grammars

Architecture. Talton et al. created a set of two grammars for architectural models. One models a single building by subdividing and shrinking of an initial block. This has been done to test the performance of the algorithm in models with irregularly divided mass. The second grammar's aim is to create a city of box-shaped buildings. In contrast to the previous grammar, the city is grown.

Trees. Several tree grammars have been implemented, to generate young oak, old oak, willow, acacia, conifer and poplar trees. The latter two are of the monopodial branching class, the others are sympodial. As in the example in the beginning, they are using a budget for each branch that is depleted and divided with each added element.

Mondrian art. All these grammars are contrasted with one which describes a subset of the grid-like composition paintings by Piet Mondrian.

5.2 Likelihood functions

Area and Volume based. For the tree and architecture grammars, a simple area and volume based likelihood function is used. The generated models are sampled to a voxel or pixel buffer I_δ, depending on the type of the input specification I, and compared to it. Assuming that each element of this buffer is

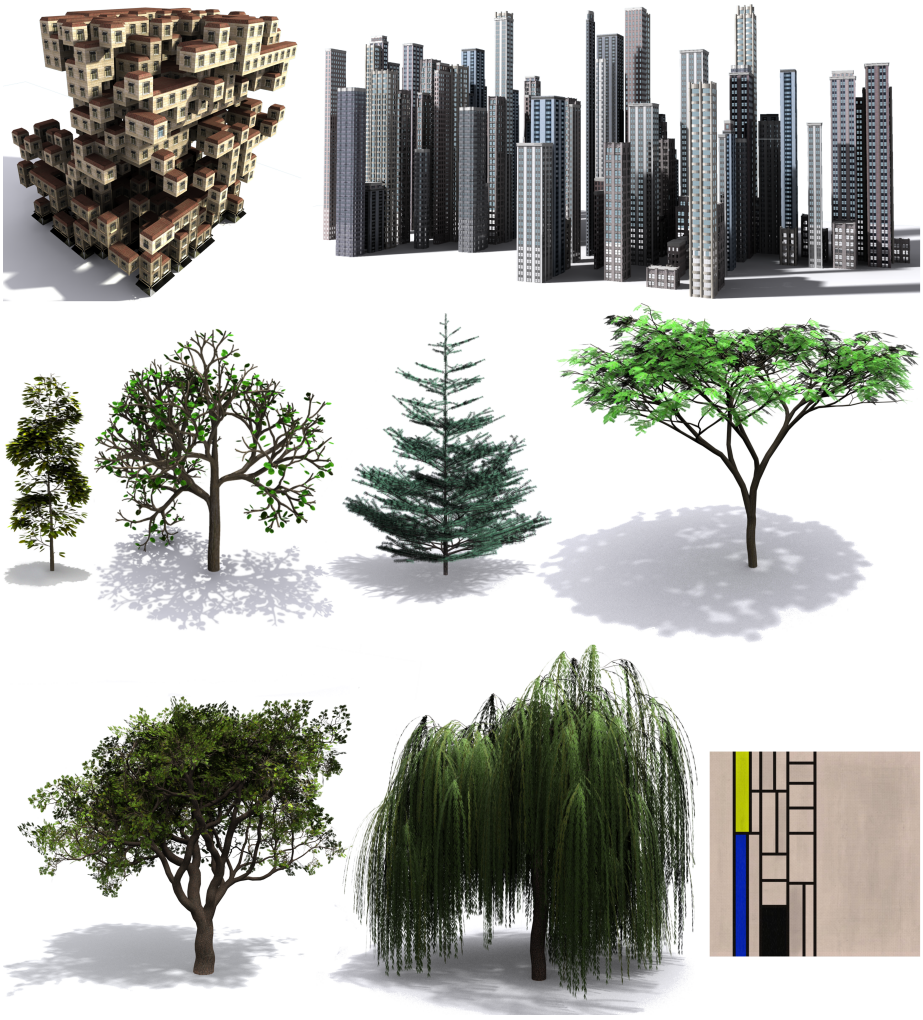

Figure 4: Models generated by the provided grammars that do not follow any input specification. The first row shows a single building and a city generated from the architecture grammars. The trees are a poplar, a young oak, a conifer, an acacia, an old oak and a willow. The rightmost in the bottom row is a mondrian-style picture. (From [4]).

independently identically Gaussian distributed allows to use least squares over the domain $\mathcal{D}$ of I and I_δ as a basis for the likelihood function:

$$\log L(I|\delta) \propto -\frac{1}{2\sigma^2} \sum_{x \in \mathcal{D}} ||I(x) - I_{\delta(x)}||^2.$$

with here $\sigma^2 \approx 0.22 \cdot \sqrt[\dim \mathcal{D}]{|\mathcal{D}|}$ being a tunable variance parameter.

Mondrian art. An additional likelihood function was created for mondrian paintings. Again in contrast to the previous function, it is based on an analysis of the aesthetic understanding of Piet Mondrian and of analytical form that can be computed without prior rendering.

6 Results

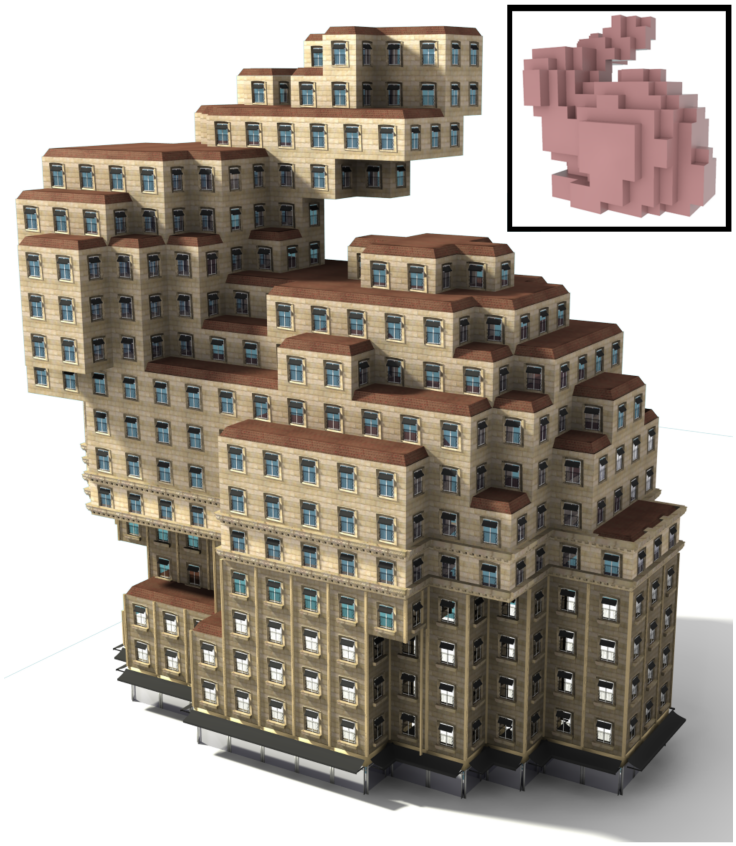

Figure 5: A bunny-shaped building derived from the building grammar (From [4]).

Figure 5 shows a building created by the previously introduced building grammar which has been moulded into the shape of a coarsely voxelized Stan-

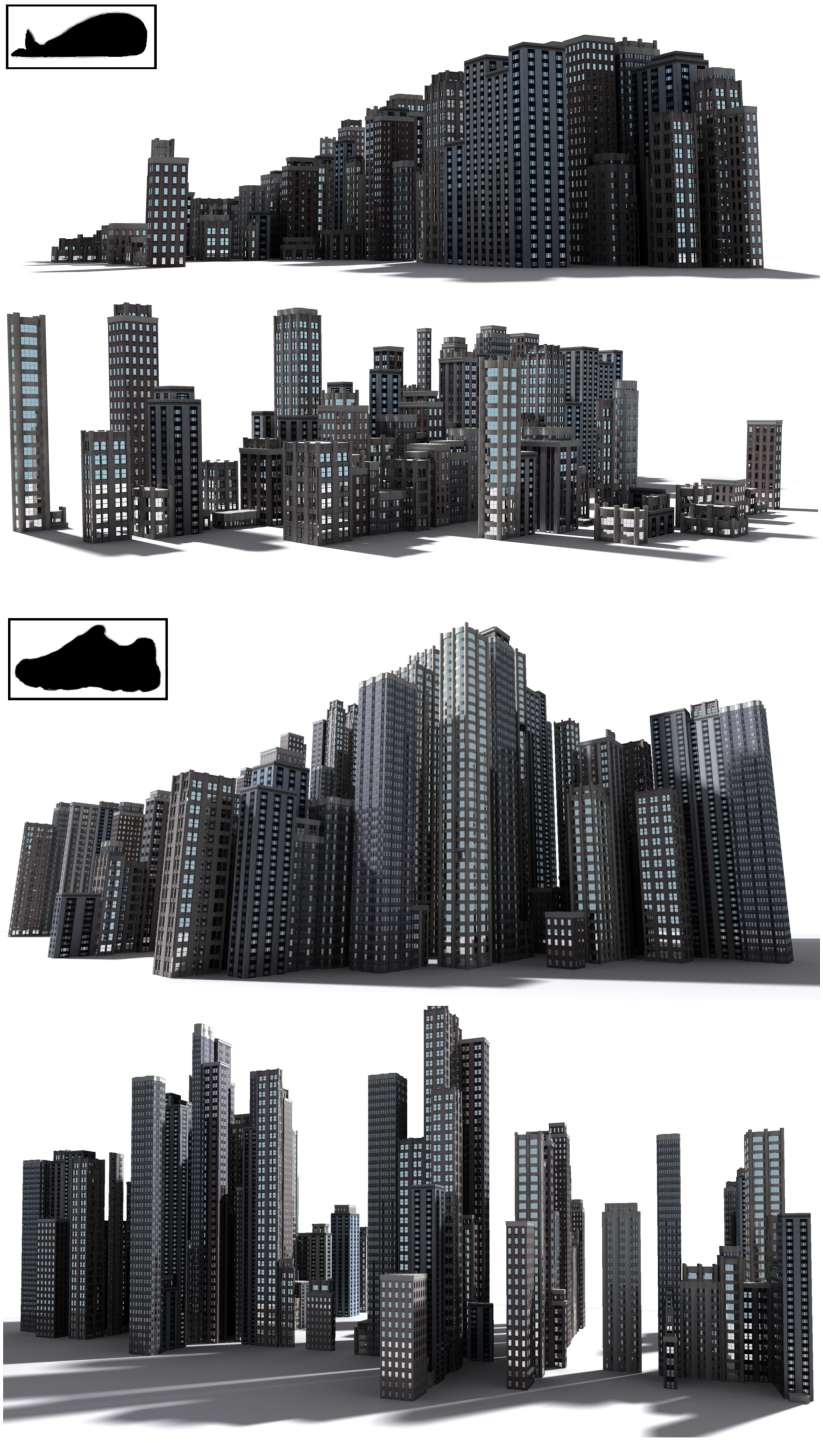

Figure 6: Two cities derived from the city grammar, rendered into a two dimensional image. The input specification for the top image was a whale, for the bottom image a shoe. For each model, an alternative viewpoint has been rendered. (From [4]).

ford Bunny. The easily recognizable strong resemblance shows the general applicability of Metropolis procedural modeling.

In Figure 6 two generated cities are depicted, one shaped as a whale, the other shaped as a shoe. This time, area-based input specifications have been used. From the viewpoint used in calculating the likelihood function, both cities adhere to the input specification. Differences near recesses can be attributed to the incapability of the grammar to generate these — in contrast to the building grammar, which was able to adequately model the bunny's ears. In addition to the matching viewpoint, another viewpoint is offered which shows the cities' random growth behaviour where it is not directly controlled.

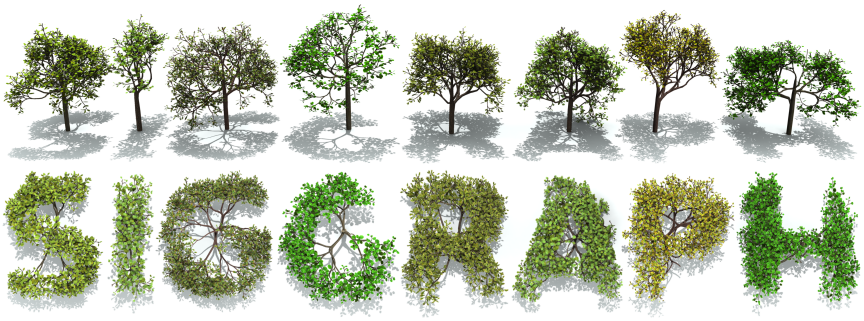

Figure 7: Young oak trees that have been shaped into a "SIGGRAPH" word. From the sides the trees look natural, the success of the operation can be seen in the shadow and from top. (From [4]).



Figure 8: An acacia and a willow tree that are shaped like the SIGGRAPH logo. (From [4]).

The Figures 7 and 8 show a very similar application to the tree grammars. The trees are controlled by the algorithm to grow so they spell "SIGGRAPH" or display the SIGGRAPH logo. Again, when viewed from the side, realistic trees can be observed. Only when the attention is moved to the shadow or the trees are viewed from the top, the intended structure becomes obvious.

A more notable example of a grammar's inability to comply to the requirements stipulated by the input specification than the city grammar examples is shown in Figure 9. While the grammar is perfectly able to generate a tree that adheres to the left input specification, the right input specification results in a seemingly unnatural and asymmetric centre branch that does not have many similarities with the other two.



Figure 9: Issues arising from incompatible input specification and grammar: The center branch in the right model seems unrealistic. (From [4]).

At last three images generated to resemble Mondrian pictures are shown in Figure 10.

Talton et al. identify the performance after some iterations have been progressed (see Figure 11). The vast improvements at first have reduced to very small steps. The impact on the performance by the optimizations is shown in Figure 12.

7 Conclusion

The algorithm for Metropolis procedural modeling by Talton et al. has been presented. As shown by the differences in the grammars employed for testing, it can be used in a wide range of applications. The results met the expectations as it was possible to control the models down to a very fine detail level as far as the grammars allowed it.

I believe that the algorithm along with the created tools can and should be used in a number of other applications. Especially in research areas which are already using Markov chain Monte Carlo, Talton et al. bring the possibility to employ contest-free grammars.

They suggest the creation of tools to make working with grammar creation easier, as it took "several hundred man-hours to develop and refine" them. Also the extension of the algorithm to other procedural expressions, like context-free L-Systems are proposed.

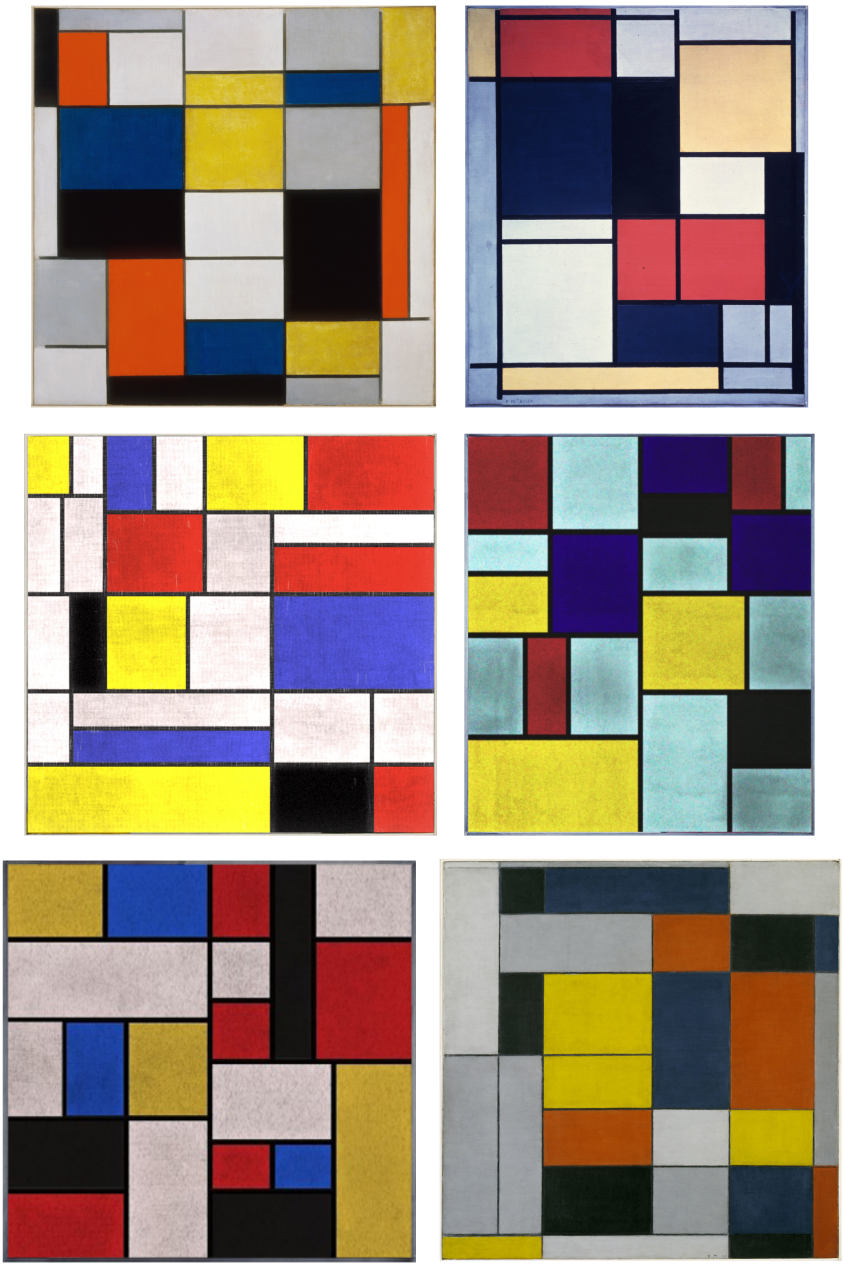

Figure 10: Results from Mondrian modeling. The first, second and last image are photographs of original paintings, the others are generated. (From [4]).

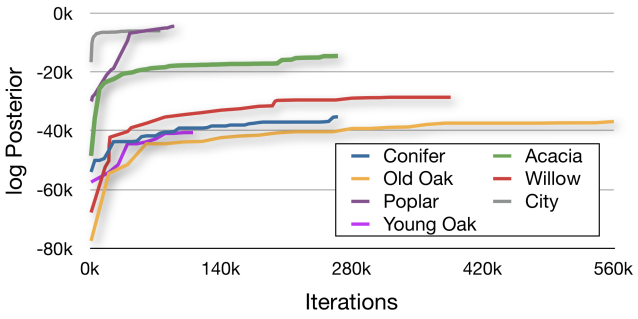

Figure 11: Performance of the algorithm for different tree grammars measured by the posterior value. (From [4]).

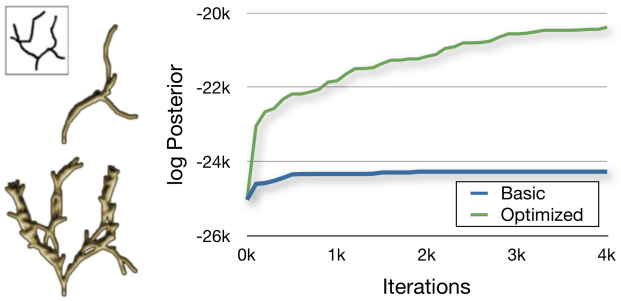

Figure 12: Algorithm performance with (bottom and green) and without (top and blue) the optimizations. (From [4]).

References

[1] Peter J. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82:711–732, 1995.

[2] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[3] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.*, 21:1087–1092, June 1953.

[4] Jerry O. Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Měch, and Vladlen Koltun. Metropolis procedural modeling. *ACM Trans. Graph.*, 30(2):11:1–11:14, April 2011.