# Synthesis of Tiled Patterns using Factor Graphs

YI-TING YEH, KATHERINE BREEDEN, LINGFENG YANG, MATTHEW FISHER and PAT HANRAHAN
Stanford University

Patterns with pleasing structure are common in art, video games, and virtual worlds. We describe a method for synthesizing new patterns of tiles on a regular grid that are similar in appearance to a set of example patterns. Exemplars are used both to specify valid tile arrangements and to emphasize multi-tile structures. We model a pattern as a probabilistic graphical model called a *factor graph*. Factors represent the hard logical constraints between tiles, the soft statistical relationships that determine style, and the local dependencies between tiles at neighboring sites. We describe a simple method for learning factor functions from a small exemplar. We then synthesize new patterns through a stochastic search method that is inspired by MC-SAT. Efficient synthesis is challenging because of the combination of hard and soft constraints. Our synthesis algorithm, called BLOCKSS, scales linearly with the number of tiles and the hardness of the problem. We use our technique to model building facades, cities, and decorative patterns.

Categories and Subject Descriptors:

General Terms: Constrained Synthesis, Procedural Modeling

Additional Key Words and Phrases: Factor Graphs, Slice Sampling, Graphical Models

**ACM Reference Format:**

Yeh, Y. -T., Breeden, K., Yang, L., Fisher M., and Hanrahan, P. 2012. Synthesis of tiled patterns using factor graphs. ACM Trans. Graph. XX, XX, Article XXX (May 2012), 13 pages.
DOI = 10.1145/xxxxxxxx.xxxxxxx
http://doi.acm.org/10.1145/XXXXXX.XXXXXX

## 1. INTRODUCTION

Historically, many cultures have produced artwork which relies on the creative repetition of a relatively small palette of motifs. For instance, in the Native American tradition, pottery, basketry, and weavings often contain repeated canonical elements. Mosaics, embroidery, and other types of visual arts also share this characteristic. Today, many interesting patterns may be synthesized by rearranging a small number of discrete blocks in new and pleasing ways. For instance, architectural elements such as windows and doors are easily modeled in this manner. Grid based patterns are a popular way to represent virtual worlds where tiles can have semantic as well as visual effects, such as in the tile-based sandbox game Minecraft.

While tiles can be used to make the modeling process more efficient, generating these tile sets and patterns can be time consuming and expensive. We focus on an example-based technique which uses existing content to generate novel patterns. An example driven approach for this type of problem is favorable because the only input to the method is artist-rendered examples. Thus, an artist's time can be spent designing interesting tile sets and a few exemplars from which a comparatively vast variety of models can be generated.

In this paper, we use examples to generate novel patterns from a common collection of discrete tiles $\mathbb{T}$; patterns are simply arrangements of these tiles over a regular grid. The grid is modeled as a collection of random variables $X_i$, each representing a tile site $i$ which takes its value from the tile set $\mathbb{T}$. Therefore, a pattern $O$ of size $N$ tiles is given by:

$$O = \{\mathbf{x} = x_1, x_2, ..., x_N; x_i \in \mathbb{T}\} \quad (1)$$

where the array of tiles, $\mathbf{x}$, is referred to as an *assignment* of tiles: one tile per grid location. The set of possible output patterns, $\Sigma$, is exponentially large; that is, $|\Sigma| = |\mathbb{T}|^N$. However, not all $O \in \Sigma$ are equally desirable. We would prefer that $O$ be similar in appearance to the input patterns—a notion which is highly subjective, but for which we attempt to provide an objective basis by taking a probabilistic view.

Our contributions include the following:

(1) A formulation of the problem of tile-based pattern synthesis in terms of factor graphs. This formulation allows for the extension of model synthesis [Merrell 2007] to encompass larger (i.e., multi-tile) regions, and to encode statistical properties in addition to the logical constraints necessary for the synthesis of good tile based patterns.
(2) A method for synthesizing patterns according to constraints encoded by the factor graph. This synthesis method, which we call BLOCKSS, is an extension of MC-SAT [Poon and Domingos 2006]. BLOCKSS uses multi-tile ("block") updates to improve performance and better preserve local arrangements found in the exemplars.

We show how our method can be further refined by the use of simple user-directed constraints. We give a comparison between our method and techniques from texture synthesis, demonstrate the performance characteristics of BLOCKSS, and provide results generated using a variety of tile sets. These results show how our algorithm can produce diverse patterns containing novel variations, even when they are synthesized using the same exemplars and constraints.

As outlined in Figure 1, this work can be divided into two parts: analysis and synthesis. The analysis phase takes as input one or more example patterns and the tile set from which they are comprised. These are used to infer edge-matching rules and multi-tile features, and to assemble factors representing these relationships. The synthesis phase combines any user defined constraints with factors learned during analysis to build a factor graph representation of

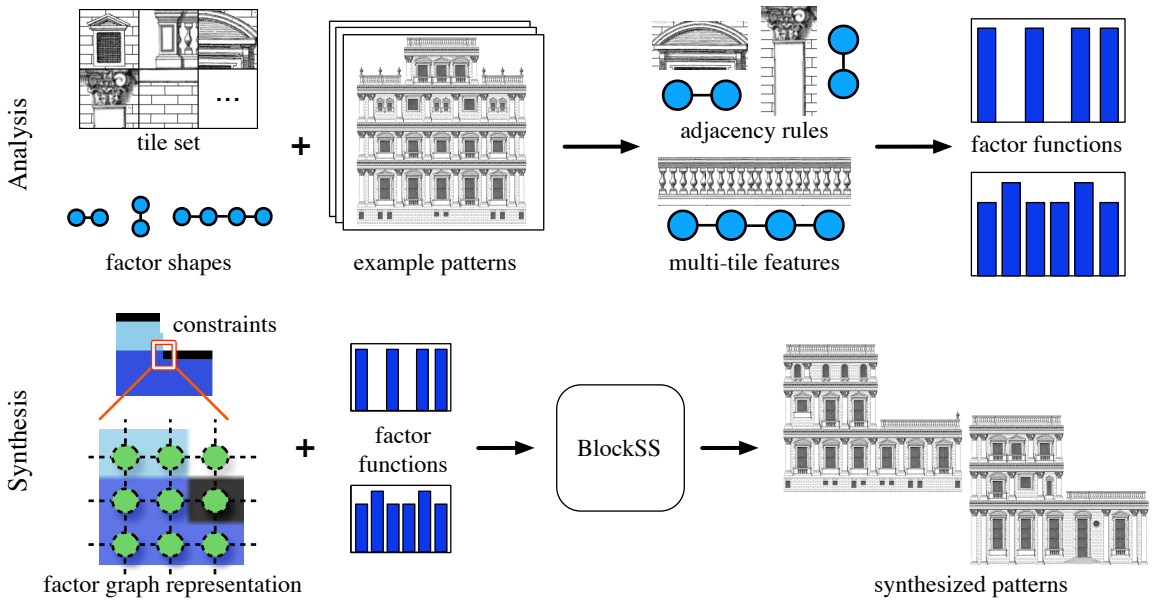



Fig. 1: An algorithm to generate tile-based patterns from factor graphs. Given a set of grid-structured exemplars and constraints, it learns a corresponding probability distribution in order to synthesize new arrangements.

the probability distribution over the space of possible output patterns. Then BLOCKSS is used to generate new patterns that resemble the examples. However, while BLOCKSS is inspired by Markov Chain Monte Carlo methods, it does not produce samples according to the distribution.

## 2. RELATED WORK

**Tile-based patterns.** Tile-based methods have been widely used in many areas of computer graphics, such as modeling and animation [Lagae et al. 2008]. One application of tiling is the use of Wang tiles for texture synthesis in Cohen et al. [2003]. They use a scanline algorithm to generate aperiodic patterns by matching abutting edge colors of Wang tiles. While patterns generated from Wang tiles are seamless, they cannot capture features encompassing more than two tiles.

**Procedural modeling.** Creating novel tile-based patterns is a 2-dimensional procedural modeling problem. Techniques for performing procedural modeling tend to fall into two categories: grammar based methods and example based methods. Grammar based methods are a highly active area of research, having been deployed in applications such as the automatic generation of trees [Prusinkiewicz and Lindenmayer 1996], decorative patterns [Wong et al. 1998] and buildings [Parish and Müller 2001; Müller et al. 2006]. Ways of combining example based modeling with grammar based methods have been recently explored; Bokeloh et al. [2010] derive symmetry rules from input 3D models in order to derive the shape grammar, addressing the difficulty of formulating effective grammars. We focus purely on example based synthesis. We believe that the modeling barrier for casual users is alleviated by exemplar guided synthesis, in which the creation of example patterns facilitates a visual exploration of the design space.

**Model synthesis.** Merrell [2007] described a method for the synthesis of new models from an example. Given a set of grid-aligned model pieces used in the exemplar, the algorithm enumerates all the immediately adjacent combinations of model pieces observed in the example model. Merrell's algorithm requires that all adjacent combinations of model pieces in the synthesized models also must be observed in the example model. The problem of synthesis then becomes one of constraint satisfaction, where the binary relation encoded by adjacent pieces is the constraint which must be satisfied during synthesis. Model synthesis is performed using a backtracking constraint satisfaction algorithm that outputs models satisfying the constraints. We extend this approach by allowing higher-order adjacency relations involving more than two tiles, and using these relationships as soft constraints.

**Markov random fields in texture synthesis.** Many probabilistic graphical models have been proposed to represent distributions of random variables. In computer graphics, one of the most familiar of these tools is Markov random fields (MRF), which have been widely applied to the problem of texture synthesis.

There are two types of approaches to texture synthesis using MRF models: parametric [Zhu et al. 1998] and non-parametric [Efros and Leung 1999]. These methods capture the statistics of pixels within a given neighborhood. Parametric methods perform well on textures that obey Gaussian statistics, while nonparametric methods are better suited to more complex textures. During synthesis, the pixel value at a location is conditioned on this neighborhood. This relationship is modeled by finding similar neighborhoods in the example texture. Many variations of the basic non-parametric approaches have been developed to improve the synthesis process, such as incorporating vector quantization to high dimensional interpolation [Wei and Levoy 2000] and coherent neighborhood search [Ashikhmin 2001]. Current methods in texture synthesis are overviewed by Wei et al. [2009].

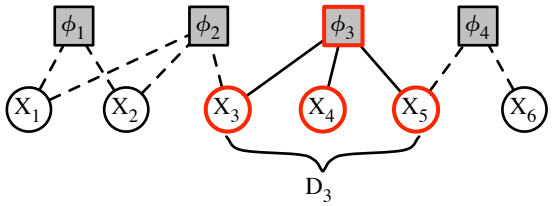
Recently, there are many techniques that formulate texture synthesis as an optimization problem. The Graphcut texture synthesis technique is based on determining the optimal seams between the

candidate patch and the output image [Kwatra et al. 2003]. Belief propagation has been commonly used to find the maximum probability assignment to a MRF. It has been applied in reconstructing images from a set of non-overlapping image patches [Cho et al. 2010]. In general, optimization is well-suited for producing a single good pattern. However, because we aim to synthesize tiled patterns exhibiting variations, we utilize a search method that is inspired by MC-SAT, a MCMC method, to generate a number of patterns that resemble example patterns.

While the two domains are closely related, naively applying texture synthesis algorithms to tile-based patterns often produces undesirable results; examples are given in Section 8. The explanation lies in fundamental differences between tile-based patterns and textures: pixel values are numeric and tiles are symbolic. The distance between neighborhoods of pixels can be computed using a simple Euclidean metric; it is also possible to produce sensible results when interpolating their values. Although tiles are comprised of pixels, they behave more like discrete symbols whose neighborhoods are harder to compare and whose values are much more difficult to interpolate. The prevention of seams between mismatched tiles is also of critical importance. Even one invalidly placed tile can create a pattern whose overall appearance is undesirable. Such constraints cannot be enforced by existing texture synthesis algorithms.

## 3. LOCAL RELATIONSHIPS AND FACTOR GRAPHS

Factor graphs are a class of probabilistic graphical model [Kschischang et al. 2001; Loeliger 2004]. To gain a better intuition of what is meant by a *factor*, begin by considering one node in the graphical model, which represents a single random variable. The dependencies between this random variable and other random variables is represented by factor nodes $\Phi$, which connect to the set of related random variables via edges $E$:




As shown, edges appear between each factor $\phi_j$ and each random variable whose value that factor depends on. This set of random variables is called the *scope* of $\phi_j$ and is denoted by $D_j$. Factors are functions that take an assignment $\mathbf{d}_j$ of $D_j$ and return a nonnegative real number. The *neighborhood* of a random variable consists of the random variable nodes that are reachable through one factor node. For example, the neighborhood of $X_5$ consists of the random variable nodes $X_3$, $X_4$ (through $\phi_3$), and $X_6$ (through $\phi_4$). This is because these nodes are reachable through exactly one factor node. Individual factors and the edges connecting them to variable sites form subgraphs which, together, form a *factor graph* representing the entire distribution.

Formally, a factor graph is a bipartite graph $G = ((\Phi, X), E)$. It is composed of a set of factor nodes $\Phi$ connected to a set of variable nodes $X$ via edges $E$. The full distribution over $X$ is given as a product of these factors, where each factor specifies dependencies over the variables of its scope.

$$P(X = \mathbf{x}) = \frac{1}{Z} \prod_j \phi_j(\mathbf{d}_j) \qquad (2)$$

where $Z$ is a normalization constant, and $\mathbf{d}_j$ is the subset of $\mathbf{x}$ which lies within the scope $D_j$ of the factor $\phi_j$. Invalid assignments are easily identified, as their joint probability will be zero. This occurs when even a single factor returns a zero value.

### 3.1 Properties of factor graphs

Here we present a few of the important properties of factor graphs.

**Logical structures.** Factors may be used to represent any logical predicate. This is accomplished using a *binary* factor which has a value 1 for certain variable assignments and 0 otherwise. Each factor function can represent any logical relationship among its arguments. The product of multiple factors is the "logical-and" of the relationships they encode.

**Statistical variability.** Factors can also capture relationships between variables that are more complex than logical predicates. For example, we may want to favor some assignments over others. To express their relative desirability, one can use factors which return larger values for preferred variable assignments, and smaller values for those which are less desirable. Thus, factor graphs can combine logical and statistical relationships.

**Locality.** Given the values in its neighborhood, the value of a single random variable is independent of all other random variables. There is a natural symmetry between this notion of graph locality and the notion of spatial locality that is key to graphics applications such as texture, model, or pattern synthesis. Because many useful constraints in these domains have spatially local effects, they are readily represented as factors having graphically local effects. However, factors are not restricted to expressing spatially local relationships.

## 4. PROBLEM OVERVIEW

Our method can be divided into two phases. The goal of the first phase, *analysis*, is to construct appropriate factor functions. During the second phase, *synthesis*, we construct the factor graph associated with output patterns with prescribed size, factor functions, and optional constraints.

**Analysis.** Input to the analysis phase is the tile set, example patterns, and the shapes of the factor functions whose values are to be determined. We build factors by analyzing local information in the exemplars. In particular, the exemplars provide us with data that will be used to prevent the synthesis of patterns containing seams, which occur when tiles whose borders do not match are placed adjacently. They also provide us with examples of multi-tile features.

Factor functions are referenced by their scope and their location in the tile grid. Recall that the scope of a factor is the collection of tile sites it acts upon. We introduce a new parameter, $s$, to describe the shape of this scope: how are the affected tile sites arranged spatially with respect to one another? Each of these shapes is referenced at some tile site $i$, an index which can be thought of as an "anchor" pinning the factor shape to a particular tile site in the synthesized image (see Figure 2). The various factor shapes $s$ can be thought of as stencils defining regions of affected tile sites; these shapes may overlap as they repeat over the image domain at each tile site $i$. When referring to some factor with shape $s$ anchored at tile site $i$, we refer to the factor as $\phi_{s,i}$, having scope $D_{s,i}$.

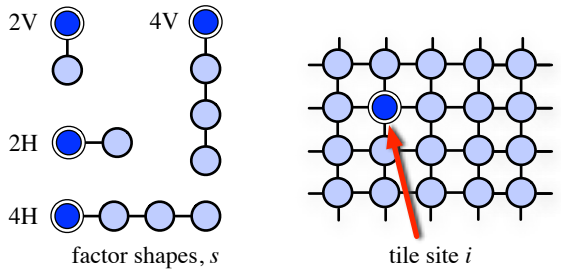



Fig. 2: Examples of factor shapes (left) anchored at a single tile site indicated in the factor graph (right).

**Synthesis.** While patterns may be synthesized directly from the distribution comprised of these factor functions alone, it is also often helpful for the user to specify additional constraints. Constrained synthesis may be accomplished by a process we call *zoning*. Zoning is performed in one of two ways. The first restricts tile values to a subset of $\mathbb{T}$ following user-defined areas. During the synthesis of a facade, this method might be used to specify the shape of the exterior. The second type of zoning restricts the factor functions in a user-defined area to those learned from a subset of the exemplars. In the synthesis of a city pattern, this method might be used to indicate that part of the pattern is synthesized from an urban exemplar while another part is synthesized from a suburban exemplar. We can express these constraints by conditioning the factor function on a zoning variable at each site. The method we use to synthesize patterns that satisfy the constraints encoded by this factor graph is introduced in Section 5.2.

## 5. BUILDING FACTOR FUNCTIONS

Because our probability distribution encodes a combination of hard and soft constraints, we design factor functions according to the maximum entropy world model in probabilistic logic [Paskin 2002], Assignments violating hard constraints have zero probability, while soft constraints encode preferences. We therefore design two corresponding types of factor functions: the first strictly enforces seamlessness between adjacent tiles, and the second favors assignments observed in the exemplars in order to better capture their appearance.

### 5.1 Direct adjacency dependencies

To ensure edge matching between tiles, every synthesis using our method includes two factor shapes: one for vertical pairs of tile sites, $\phi_{2V}$, and one for horizontal pairs, $\phi_{2H}$. The shapes are illustrated in Figure 2. These logical factors yield a 1 when the tiles adjoin seamlessly and 0 when their intersection creates a seam. In other words, seams are treated as logical inconsistencies.

Points in the support of a distribution given by the product of logical factors are also the solutions to the constraint satisfiability problem encoded by edge-matching constraints. Each valid pattern has a uniform, positive probability; each invalid pattern has zero probability.

### 5.2 Higher order dependencies

While $\phi_{2V}$ and $\phi_{2H}$ are sufficient to yield seamless patterns, we would also like our method to give preference to patterns containing common multi-tile arrangements found in the exemplars. To do this, we also include factors with scopes containing multiple tile sites. We call these *higher-order* factors. There are two issues: selection of the factor shapes and selection of the factor functions.

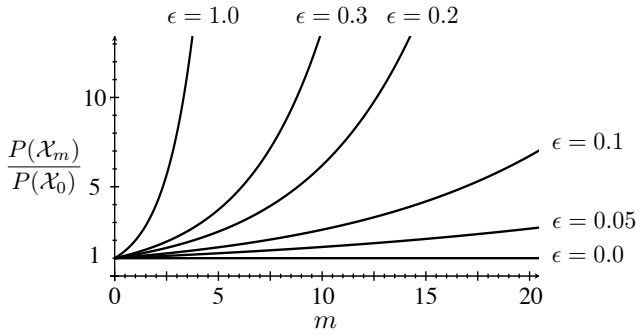



Fig. 3: Plot of the probability ratio of patterns $\mathcal{X}_m$ satisfying $m$ higher-order factors to patterns $\mathcal{X}_0$ satisfying zero higher-order factors.

Factor shapes should reflect the shapes of the structures we want to preserve. In the facade examples in this paper, we use the factor shapes 4V and 4H shown in Figure 2. The 4V configuration is used to capture the fact that columns are aligned between floors. The 4H configuration preserves the horizontal spacing of windows. In other patterns, different factor shapes may better capture the structure of the pattern. In city layouts (see Figure 15), square 3x3-tile factors are used to capture the structure of city blocks and roads.

We use a simple step function for these factor functions. Given a factor shape $s$, $\phi_s$ is equal to 1 if the assignment has not been observed in the exemplar, and $1 + \epsilon$, $\epsilon > 0$ if the assignment has been observed. This parameter controls the relative likelihood of patterns replicating multi-tile assignments versus patterns that do not.

Figure 3 demonstrates the effect of $\epsilon$ on the relative probability of patterns satisfying some number of higher order factors compared with patterns that do not. Let $\mathcal{X}_m$ denote a pattern that satisfies $m$ higher-order factors. We see that if $\epsilon = 0$, all valid patterns have the same probability. If $\epsilon > 0$, the probability of a pattern increases exponentially with the number of higher-order factors satisfied. This effect is more pronounced as $\epsilon$ increases. As $\epsilon \to \infty$, higher-order factors become hard constraints. The examples in this paper are generated using $\epsilon = 0.1$. We find this gives a reasonable balance between producing novel variations and preserving the appearance of the exemplars. The trade-offs between quality and performance for various values of $\epsilon$ are further discussed in Section 8.

Note that, for each output pattern size, it is possible to model the product of factor functions as an exponential family distribution and to then learn the corresponding parameters using maximum likelihood. However, in the use cases we consider the number of exemplars is insufficient for the application of such methods. We leave the problem of learning factor functions from sparse data to future work.

## 6. SYNTHESIS METHOD — BLOCKSS

Given a tile set, exemplars, factor shapes, and zoning constraints as input, we can derive the factor graph associated with the output patterns. The probability distribution encoded by this factor graph is given by:

$$P(X = \mathbf{x}) = \frac{1}{Z} \prod_s \prod_i \phi_{s,i}(\mathbf{d}_{s,i}). \tag{3}$$

Our synthesis method, which we call BLOCKSS, is inspired by a product slice sampler called MC-SAT. BLOCKSS interleaves two steps:

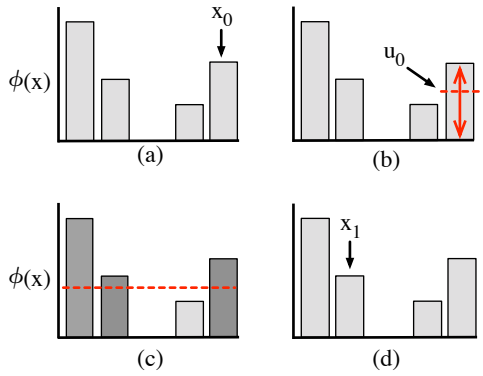



Fig. 4: Stages of slice sampling. (a) The starting state, $x_0$. (b) An auxiliary sample, $u_0$, is taken uniformly from $[0, \phi(x_0)]$. (c) The support of this slice is shown in dark gray. (d) The next state, $x_1$, is chosen from this support. Note the algorithm is able to move between disconnected regions of support.

(1) Form *slices* by sampling auxiliary variables based on the current state.

(2) Generate the next state, in which each factor score exceeds the corresponding auxiliary variable value sampled in (1).

When a uniform sampler is used to find the set of solutions satisfying slice thresholds (defined below), the Markov chain generated by MC-SAT will satisfy ergodicity and the detailed balance condition [Poon and Domingos 2006]. However, we found that implementing a perfectly uniform sampler is both too costly and unnecessary for generating good quality patterns. Instead, we employ a blocking scheme in step (2) as an efficient, but biased, substitute. Therefore, BLOCKSS does not exactly sample from the distribution shown in Eq (3). Before giving a detailed description of our synthesis method, we begin with an overview of univariate slice sampling.

## 6.1 Slice Sampling

In slice sampling [Neal 2000], the space $X$ is augmented with an additional *auxiliary variable*, $U$. The resulting joint distribution is then over $(X, U)$. Provided the auxiliary variable is formulated properly, the marginal distribution of $X$ in $P(X, U)$ is the original distribution of interest, $P(X)$. Typically, the MCMC transition functions in slice sampling consist of Gibbs-style updates alternating between sampling from $P(X|U)$ and from $P(U|X)$. The benefit is that the scale of changes made in $X$ when sampling from $P(X|U)$ adapts to the distribution.

In Figure 4, we visualize how slice sampling works for one auxiliary variable. To sample $u$ given some $x_0 \in X$, begin by taking a sample $u_0$ uniformly from $[0, \phi(x_0)]$. This defines a threshold for the corresponding factor, $\phi(x)$. Next, obtain a new sample $x_1$ given $u_0$. This is done by sampling uniformly from the set $A = \{x : \phi(x) > u_0\}$. This subset of $X$ is known as the *slice*. As shown in Figure 4 (c), a low threshold will increase $|A|$, because there will be many assignments whose scores exceed the threshold. Conversely, high threshold values make $|A|$ small.

Slice samplers can be specialized to handle the distribution of a product of factors, i.e., $P(X) \propto \prod_{j=1}^{n} \phi_j(\mathbf{x})$, with multiple auxiliary variables, $\{u_1, u_2, \ldots u_n\}$, one for each factor. This is called the *product slice sampler* [Edwards and Sokal 1988; Mira and Tierney 1997]. As in the one-dimensional case, each $u_i$ defines a threshold for its corresponding factor and is uniformly sampled.

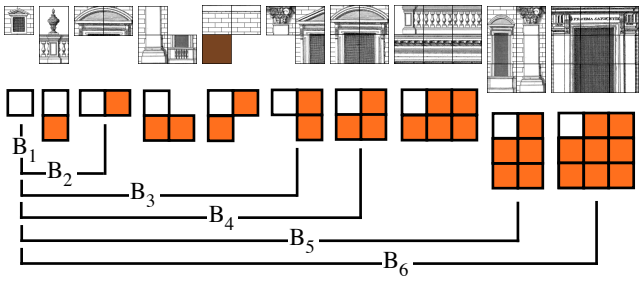



Fig. 5: Tiled examples (top) demonstrate the underlying shape of each block (middle). The reference tile is indicated in white. Blocks are grouped into schemes B1, B2, ... B6 (bottom).

## 6.2 Auxiliary Variable Sampling in BLOCKSS

As in product slice sampling, BLOCKSS employs one auxiliary variable $u_j$ for each factor $\phi_j$. At every iteration, we sample the slice threshold for each $u_j$ uniformly from 0 to the current score of its corresponding factor. In our setting of tiled patterns, the slice corresponds to valid tilings whose per-factor likelihood scores are each greater than the value of the corresponding auxiliary variable sample (Algorithm 1).

## 6.3 Blocked SampleSAT in BLOCKSS

To draw $\mathbf{x} \in A$, we employ a state space search algorithm inspired by SampleSAT [Wei et al. 2004], a hybrid method for the approximate uniform sampling of solutions to SAT problems. We extend the SampleSAT framework to update multiple random variables at each step (blocked updates). We call this BlockSampleSAT (Algorithm 2). Similar to each step of SampleSAT, it randomly selects WalkSAT and simulated annealing with probability $p_{walk}$ and $1 - p_{walk}$, and follows the use of a fixed temperature $T$ in the simulated annealing step. The differences lie in the blocked updates.

Each blocked update changes a set of tiles relative to a reference site $i$. First, the shape of this block is chosen uniformly at random from a predefined set. This defines the block as a set of tile sites in the neighborhood of site $i$. We then update all random variables in the block around site $i$ by copying in a block of the same shape from the exemplars, also chosen uniformly at random. This restricts the assignments of our replacement blocks to those which are found in the exemplars. Similar to the factor shapes discussed in section 4, block shapes contain a reference tile used to anchor them consistently relative to each reference site. Example blocks and the block shapes used in our updating scheme are shown in Figure 5.

**Efficiency.** To see why updating more than one tile at a time in SampleSAT is desirable, we first note a problem encountered when performing "single-site" updates, in which only the tile found at a single location is changed. These updates often cause synthesis to become trapped in local optima, where no single-site change improves the number of factors whose current configurations exceed their slice thresholds.

Blocked updates lessen this effect. Recall that in our formulation, a factor will have a score exceeding the slice threshold if its configuration comes from the exemplar, as such configurations maximize its score. Because each blocked update directly transfers configurations from the exemplar, factors with scopes inside the updated block will score above the slice threshold. This guarantees movement through the state space.

**Algorithm 1** Pattern synthesis algorithm BlockSS.

**Input:** factor graph $G$, replacement blocks $\mathbf{B}$, tile set $\mathbb{T}$.
**Output:** samples $O_1, O_2 \ldots O_R$.

Initialize pattern $O_0$ with tiles uniformly sampled from $\mathbb{T}$.
**for** n := 0 to $R-1$ **do**
  **for** $\phi_j$ in $\Phi$ **do**
    $u_j \leftarrow \mathcal{U}[0, \phi_j(\mathbf{d}_j)]$
  **end for**
  $O_{n+1} \leftarrow$ BlockSampleSAT(G, $\mathbf{u}$, $\mathbf{B}$, $O_n$)
**end for**
**return** $O_1, O_2, \ldots, O_R$

**Replicating statistical relationships.** Our formulation has two key properties: (1) the same factor function repeats over the pattern; (2) each factor function takes on two values — $\{0, 1\}$ or $\{1, 1+\epsilon\}$. This allows us to use the same factor functions to output patterns of different sizes.

However, overlapping factor functions with limited local scope cannot prevent the synthesis of patterns with longer range, undesirable tile arrangements. For instance, consider a facade exemplar containing a vertical arrangement of 4 tiles into a column feature. If these tile relationships were expressed using only 2V factors, synthesis might produce undesirable elongated columns of 6 or more tiles. However, because each 2V pair of tiles was validly placed, the probability of this less desirable pattern would be the same as a pattern containing 4 tile columns whose appearance more closely matched the exemplar.

In general, no matter the size of the factors, exemplars may contain features which extend beyond their local neighborhood. This issue is mitigated by performing randomized blocked updates, which help simulate exemplar appearance without drastically increasing the complexity of the problem. By duplicating higher-order structures from the exemplar (such as vertical arrangements of 4 or more tiles), the synthesized patterns might contain, for example, more 4-tile columns and fewer "extra long" columns. In general, blocking exploits the fact that locality in the synthesized pattern ought to correspond to locality in the exemplars; this is in accordance with previous work in other synthesis algorithms which attempt to preserve locality [Wei et al. 2009].

Blocking does not obviate the usefulness of higher-order factors. With only 2H/2V factors, the candidate replacement block needs only to satisfy constraints between directly adjacent tiles. When higher-order factors are used, the candidate block will need to satisfy constraints with scopes further into the pattern. Through this mechanism, higher-order factors introduce correlations between candidate blocks at longer ranges. This has the effect of making the synthesized patterns more consistently replicate the appearance of the exemplars.

In short, blocked updates improve the efficiency and quality of synthesis by taking advantage of the spatial locality of our application domain. Pseudocode for BLOCKSS is provided in Algorithms 1 and 2. Patterns synthesized in this paper all used $T = 0.25$, $p_{walk} = 0.6$ and $p_{flip} = 0.1$ where $T$ is the temperature used in the simulated annealing step, $p_{walk}$ and $p_{flip}$ are the probabilities of performing WalkSAT steps and random flips in the BlockSampleSAT algorithm.

## 7. QUANTIFYING EXEMPLAR APPEARANCE

When evaluating the quality of synthesized patterns, we will use histograms defined over local tile configurations in the exemplars.

**Algorithm 2** BlockSampleSAT(G, $\mathbf{u}$, $\mathbf{B}$, $O'$).

**Input:** factor graph $G$, auxiliary variables $\mathbf{u}$, replacement blocks $\mathbf{B}$, current pattern $O'$, maximum iteration $N$, sampleSAT parameters($p_{walk}$, $p_{flip}$, $T$).
**Output:** pattern $O$.

$O \leftarrow O'$
**for** t := 1 to $N$ **do**
  $u_{walk} \sim \mathcal{U}[0, 1]$;
  **if** $u_{walk} < p_{walk}$ **then**
    Pick a currently unsatisfied factor $\phi_j$ anchored at site $i$;
    $u_{flip} \sim \mathcal{U}[0, 1]$;
    **if** $u_{flip} < p_{flip}$ **then**
      Update $O$ with a random replacement block $b \in \mathbf{B}$ anchored at site $i$;
    **else**
      Randomly select a replacement block type that will cover the scope of $\phi_j$;
      Update $O$ with the assignment to that block type that maximizes the number of satisfied slice thresholds;
    **end if**
  **else**
    Let $c_{old}$ be the number of satisfied slice thresholds of $O$;
    Propose a new assignment $O^*$ by randomly pick a site $i$, update with a replacement block $b \in \mathbf{B}$ anchored at site $i$;
    Let $c_{new}$ be the number of satisfied slice thresholds of $O^*$;
    $u_{accept} \sim \mathcal{U}[0, 1]$;
    **if** $u_{accept} < \min(1, \exp((c_{new} - c_{old})/T))$ **then**
      $O \leftarrow O^*$
    **end if**
  **end if**
  **if** there are no remaining unsatisfied factors **then**
    **return** $O$;
  **end if**
**end for**
**return** $O$;

Let $\Sigma_s$ denote the set of all possible tile assignments to shape $s$, with particular tile configurations $\mathbf{c}_s \in \Sigma_s$. Then the histogram $f(\mathbf{c}_s)$ is calculated as

$$f(\mathbf{c}_s) = \frac{F(\mathbf{c}_s)}{\sum_{\mathbf{b}_s \in \Sigma_s} F(\mathbf{b}_s)} \tag{4}$$

where $F(\mathbf{c}_s)$ is the number of observations of $\mathbf{c}_s$ in the exemplars. We abbreviate these functions as $f_s$ and $F_s$, respectively.

Figure 6 illustrates $F_{2H}$ and $F_{2V}$ for a simple facade. Here, we can see that histograms of tile configurations are able to encode informative design metrics; for instance, the solid windowsill occurs half as often as the fenced windowsill (see the top two entries for $F_{2H}$).

Because desirable synthesized patterns will encode the same design metrics as the exemplars from which they are formed, their histograms should be analogous. We will quantify the similarity between two histograms using Kullback-Leibler (KL) divergence, which has also been used in the visual domain to compare images [Goldberger et al. 2003]. A small KL divergence between the histograms of exemplar and synthesized patterns will indicate a good replication of local tile configurations.

**Pitfalls of using histograms as factor functions.** Note that these histograms $f_s$ cannot be used as factor functions $\phi_{s,i}$. This is because histograms represent global statistics over the entire pattern,

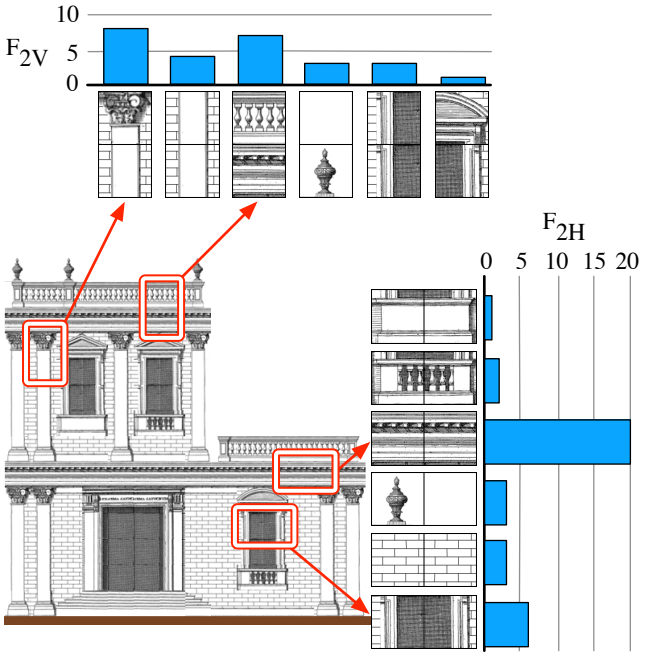



Fig. 6: Histogram of local configurations $F_{2V}$ and $F_{2H}$.

whereas factor functions express local probabilities. To illustrate this point of why local probabilities and global statistics are not equal, suppose we choose $\phi_{s,i} = f_s$. Then, in our formulation, the probability density of a given pattern would be proportional to the product of repeated factor functions, $\prod_i f_s$. The product raises the global statistics to a power that depends on the size of the pattern, incorrectly exaggerating the arrangement of highest local probability. In general, the global frequencies will not be equal to the local probabilities used to generate the pattern [Koller and Friedman 2009].

## 8. RESULTS

We judge the quality of synthesized patterns by two criteria:

(1) Seamless tile placement
(2) Preservation of higher-order statistics from the exemplars.

Patterns adhering to these constraints will contain global features such as sensible roofs and floor levels, as well as local multi-tile structures such as doors, windows, and other stylistic elements. Other methods are not able to reliably achieve these properties. To account for burn-in, we discard the first five patterns synthesized in each experiment.

### 8.1 Comparisons with other synthesis algorithms

**Texture synthesis.** We use tile-based architectural facades to compare the quality and performance of our results with previous synthesis algorithms. The tiles and facades used here originate from a book of architectural designs [Campbell 2006]. An important question is whether texture synthesis algorithms adapted to operate on tiles can also synthesize seamless patterns. In Figure 7, we show that they cannot. The exemplar is the same as in Figure 8. Traditionally, texture synthesis employs algorithms similar to those found in [Wei and Levoy 2000], in which a causal MRF neighborhood is used to generate the result in scanline order. The algorithm determines the final value of each site as a function of its local neighborhood, which in turn consists of previously assigned values. This approach can create neighborhoods in which it is impossible to place the next tile without violating logical constraints. Non-scanline neighborhood-based update methods may also fail to produce coherent solutions for the same reason. These patterns would all have probability zero under our formulation. Another alternative is a patch-based texture synthesis method called Image Quilting [Efros and Freeman 2001]. We see that results from this technique also exhibit seams and discontinuities. Moreover, it is not clear how to recover discrete tiles from these results, as the method operates on individual pixels.

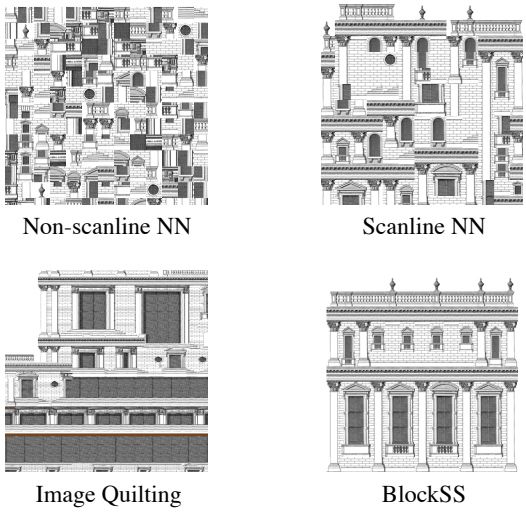



Fig. 7: Comparison with texture synthesis. Neither neighborhood-based nor image quilting methods synthesize logically consistent results, Neighborhood-based methods were implemented with a Hamming distance metric and cross-shaped non-causal and causal neighborhoods. The exemplar used for these patterns is shown in Figure 8.

**Model synthesis.** We compare the effectiveness of BLOCKSS using 2H/2V and 4H/4V factors with the model synthesis algorithm of Merrell [2007], which formulates the task as a logical constraint satisfaction problem (CSP). Figure 8 shows the qualitative difference between the two methods using the facade tile set. Unlike methods from texture synthesis, model synthesis produces seamless patterns. However, particularly when unconstrained (see Figure 8a and 8c), model synthesis often yields results containing undesirable artifacts such as repeated windows and elongated doors which were not present in the exemplar. Fixing the values of a few tile sites improves the performance of model synthesis. In Figure 8b and 8d, we show patterns produced by both methods when the floor and top left tiles are fixed. While the results for model synthesis are improved, they still cannot capture longer-range features.

**Pure constraint satisfaction.** The use of CSP solvers for model synthesis can be further developed. We show that with 2H/2V constraints, we can synthesize higher quality patterns than those produced by an off-the-shelf CSP solver. Furthermore, we demonstrate that including multi-tile factor functions improves quality as compared to only including 2H/2V factor functions.

We synthesized 75 L-shaped facades using three methods:

(1) A CSP solver with $\phi_{2H}$ and $\phi_{2V}$ as hard constraints.

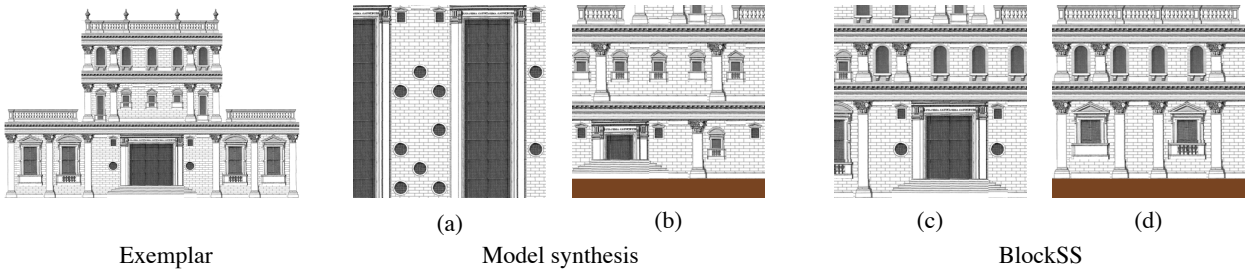



Fig. 8: Model synthesis (center) produces patterns that are seamless but contain strange multi-tile features. Our method (right) shows improved results. Fixing floor and top left tiles as in (b) and (d) improves the quality of results for both methods.

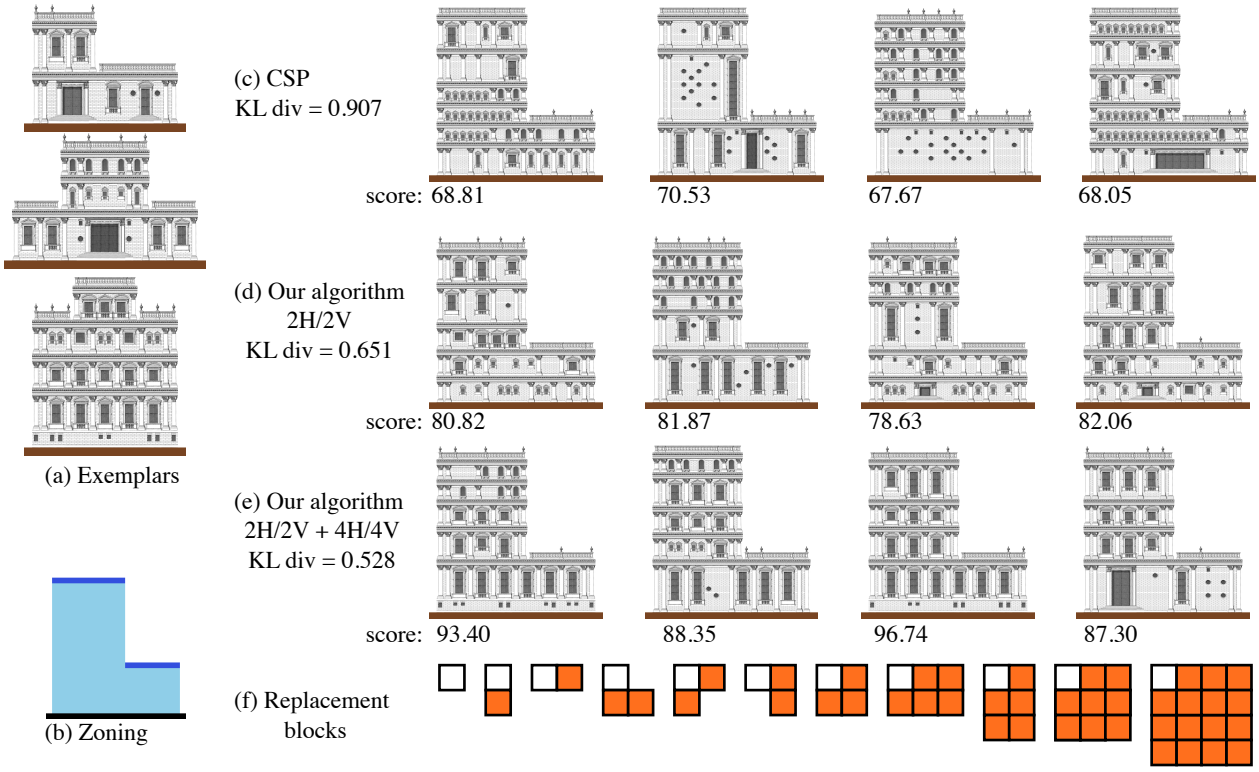



Fig. 9: Comparison with CSP. (a) Input exemplars. (b) Zoning constraints. (c) Results obtained using a general-purpose CSP solver with the 2H/2V logical factors as constraints. (d) Our algorithm with the same logical factors. (e) Our method with additional 4H/4V factors as soft constraints. (f) Replacement blocks used in (d) and (e). KL-divergence is calculated for a set of 75 results over $f_{4H}$ and $f_{4V}$; the results here are representative samples. The score below each facade is the unnormalized log probability under both hard 2H/2V and soft 4H/4V factors. The tiles were extracted from the book "Vitruvius Britannicus: The Classic of Eighteenth- Century British Architecture" ©Dover Publications.

(2) BLOCKSS on the distribution with factors $\phi_{2H}$ and $\phi_{2V}$ as hard constraints.
(3) BLOCKSS on the distribution with factors $\phi_{2H}$, $\phi_{2V}$ (hard constraints), $\phi_{4H}$, and $\phi_{4V}$ (soft constraints with $\epsilon = 0.1$).

The effect of the choice of $\epsilon$ is outlined in Section 8.3. We found $\epsilon = 0.1$ to represent a tradeoff between quality and speed suitable for a semi-interactive tool, being able to consistently produce a pattern of reasonable quality every few seconds. The CSP solver used is a general-purpose solver called MINION [Gent et al. 2006]. In order to achieve a higher level of pattern variation, the solver was set to randomize the ordering of decision variables before each run. Zoning constraints were required to control the exterior shape of the synthesized facades.

In Figure 9, we selected representative results synthesized by each method. As expected, the overall appearance of the CSP results is similar to that of Merrell in Figure 8, and does not match that of the exemplars. In contrast, the results of BLOCKSS replicated the appearance of the exemplars while also introducing some novel variations. We also applied CSP with 4H/4V as hard constraints. However, we found that there was no satisfying assignment for the L-shaped facade under these constraints. The treatment of higher-

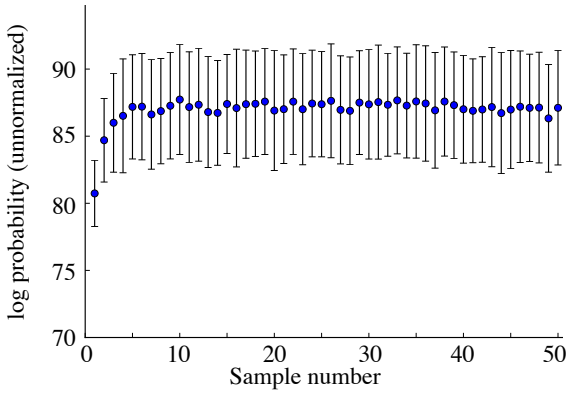



Fig. 10: The average and standard deviation of unnormalized log probability of samples over time using BLOCKSS. The plot was made by running the algorithm 115 times on the L-shaped facade with 2H/2V and 4H/4V factors.

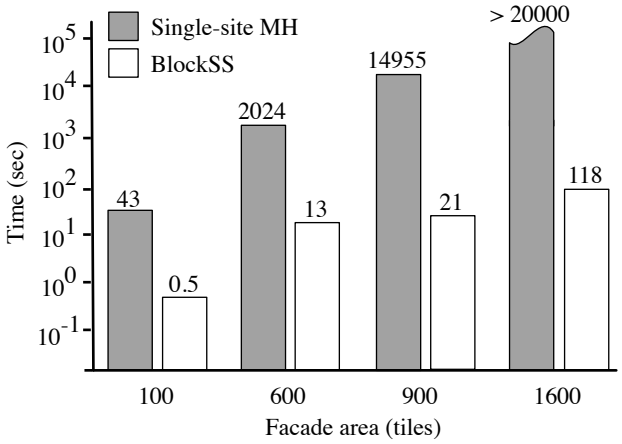



Fig. 11: Performance comparison of BLOCKSS versus single-site MH. MH scales poorly with pattern size.

order factors as soft constraints is what allows our method to avoid this pitfall.

**Quantitative measure of style preservation.** We would like to quantify both how well the output patterns match the exemplars statistically and the effect of including additional soft, higher-order constraints. To do this, we took the KL divergence of the output patterns with respect to the $f_{4H}$ and $f_{4V}$ distributions for the input facades according to Equation 4. Along with the KL divergence, we also included the value of the unnormalized log probability composed of both hard 2H/2V constraints and soft 4H/4V constraints according to the schemes outlined in sections 5.1 and 5.2. These values are included in Figure 9. The CSP-based results had a KL-divergence of 0.907. Our algorithm using just 2H/2V factors improved the KL-divergence to 0.651. This quantitative change is reflected qualitatively; the exemplar facade's appearance is better captured. Our algorithm run with additional 4H/4V higher-order factors attained the lowest KL-divergence (0.528), and produced facades that best captured the appearance of the exemplars. Finally, we see that the log probability of patterns synthesized with the higher-order factors is higher than those synthesized using only hard constraints. In Figure 10, we demonstrate how BLOCKSS mixes by running it 115 times on the distribution used in Figure 9(e). We plot the average and standard deviation of unnormalized log probability at each point in time, confirming that 5 samples are sufficient for burn-in.

## 8.2 Comparison with single-site Metropolis-Hastings

We compare the performance of BLOCKSS to single-site Metropolis-Hastings [Hastings 1970], a commonly used Markov Chain Monte Carlo technique. We synthesized 10 rectangular facades of various sizes from the exemplars shown in Figure 9. The probability distribution was composed of 2H/2V factors only. We measured the average time taken to synthesize a facade in the support of the distribution. Because single-site MH requires a positive probability distribution to converge, we assigned a small probability of $10^{-8}$ to invalid configurations; the results are shown in Figure 11. Our method achieves a speedup of over 90x for small facades (100 tiles). As the facade size increases, we achieve greater speedups (up to 7000x for the facade of 900 tiles). In the case of the 1600 tile facade, single-site MH does not converge in a reasonable amount of time, while our algorithm produces a result in 2 minutes. All timings were obtained using an Intel Xeon desktop clocked at 2.66 GHz with 8GB RAM.

The scalability of our algorithm determines its range of applicability. We summarize the effect of three variables that critically affect the performance of our algorithm: blocked updates, pattern size, and constraint hardness.

**Gains from blocked updates.** We measured the effect of blocking on performance by synthesizing 50 facades and calculating the average time to generate a facade in the support of the distribution using different sets of replacement blocks, which are shown in Figure 5. The distribution was encoded using 2H/2V logical factors. This test was repeated using various zoning constraints on the exterior facade shape. Results are given in Figure 12. In most cases, using larger blocks speeds up the sampling process. However, we also see that occasionally the logical constraints encoded by the zoning are easily satisfied even using a single-site block updating scheme, in which case blocking provides a smaller benefit (see the right-most plot in Figure 12).

**Scaling with pattern size.** Tiles are a convenient, effective, and popular way to represent large virtual worlds. We created a city tile set in a style similar to that found in "Sim City" type games. We synthesized city layouts of increasing size, and measured the time taken to produce a city layout in the support of that distribution. A 4-way crossroad was fixed at regular intervals of 20 tiles as constraints, acting as the seed for a road network. The results are given in Figure 13 (a). We find the runtime to increase linearly with respect to the size of the city for all of the factor schemes tested. For a distribution composed of 2H/2V factors, the largest city (170x170) takes about 5 minutes. On the other hand, it took over 2 hours for single-site MH to synthesize even a small 20x20 city under the same constraints. When we add higher-order factors (2x2, 4x4), performance degrades by a constant factor.

## 8.3 Performance of BLOCKSS

**Scaling with constraint hardness.** The hardness of the constraints encoded by one exemplar can vary greatly. Recall that arbitrary boolean satisfiability problems (which are NP-complete) can be encoded by 2H/2V constraints [Merrell 2007]. Here, we characterize how performance is affected by intrinsic hardness. One

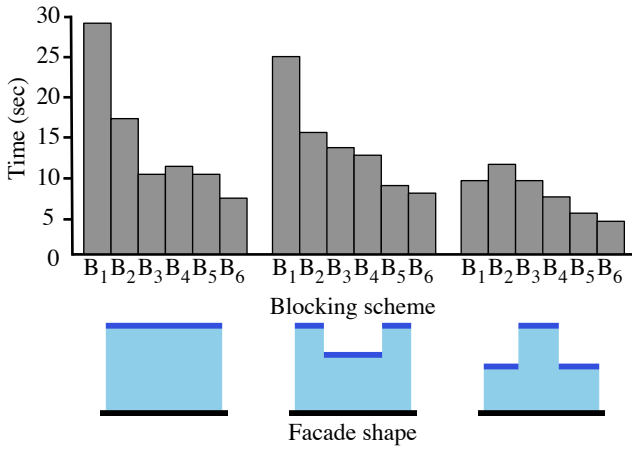



Fig. 12: Time to generate a single pattern under the blocking schemes shown in Figure 5 (average time taken over 50 patters, sized 20x20 tiles.) Higher-order blocking schemes increase performance; this effect is relatively insensitive to facade shape.

| $\epsilon$ | KL-div | avg. time (sec) | std. dev. time (sec) |
|---|---|---|---|
| 0.05 | 0.587 | 63.15 | 116.71 |
| 0.10 | 0.481 | 77.38 | 139.06 |
| 0.20 | 0.449 | 471.93 | 1458.80 |
| 0.30 | 0.453 | 236.52 | 378.42 |
| 0.40 | 0.389 | 419.04 | 441.47 |
| 0.50 | 0.358 | 1444.75 | 3004.11 |
| 0.60 | 0.301 | 452.94 | 935.73 |

Table I. : Trade-offs between quality and performance for various $\epsilon$ values.

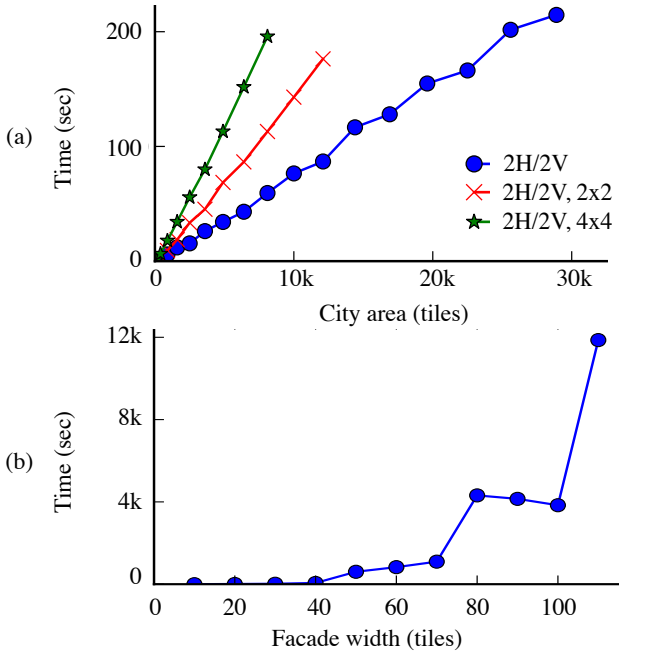



Fig. 13: (a) Time to produce a square city of increasing size; factor shapes are indicated in the legend. Performance scales linearly with area. (b) Hardness increases with facade width; this degrades performance significantly as compared to increasing area alone. Fixing one floor tile determines an entire row of the facade. As a consequence, the hardness, as the maximum size of the deterministic effects, is on the order of the width of the facade.

way to quantify hardness is to consider the range of deterministic constraints encoded by the exemplars.

Consider that for city layouts, placing any tile in the layout only determines a local set of tiles constant in size. This is true for a city of any size. In contrast, for our facade exemplars, a single floor tile will restrict all other tiles in that row. Consequently, as the width of a synthesized facade increases, so does the number of tiles spanned by this restriction. To evaluate the effect of hardness on performance, we synthesized square facades of increasing size. Results plotting synthesis time as a function of pattern width are given in Figure 13 (b). Unlike the results shown for the city patterns in Figure 13 (a), we can see that for facade patterns the hardness increases significantly with width. This is reflected in the increased runtime.

**Effect of $\epsilon$ on quality and performance.** We ran the L-shaped facade with zoning constraints as in Figure 9(b) using 2H/2V (hard factors) and 4H/4V (soft factors) on different $\epsilon$ values. For each value of $\epsilon$ we synthesized 40 patterns. To account for burn-in, we discard the first four patterns generated. In Table I, we relate $\epsilon$ to the KL-divergence and average time to generate a new pattern. We find that increasing $\epsilon$ increases quality (lower KL-divergence).

In general, larger $\epsilon$ increases the average and standard deviation of running time. However, the relation between $\epsilon$ and running time may not be strictly monotonic. In particular, we see that $\epsilon = 0.6$ results in shorter runtime than $\epsilon = 0.5$. While counterintuitive, this can be explained by how $\epsilon$ affects the state-space landscape and in turn the behavior of our algorithm. Because our algorithm depends on stochastic local search (SampleSAT) to find solutions that satisfy the slice threshold, it is possible for it to become trapped in local optima. Although higher $\epsilon$ decreases the number of solutions (as it "hardens" the soft constraints), it may simultaneously remove local optima, actually making the problem easier [Yokoo 1997].

## 8.4 Controlling and mixing style

Factors learned from an exemplar characterize its style. Combining factors from different exemplars allows the user to create hybrid styles. We demonstrate the composition of styles in Figure 14 using a decorative tile set taken from a catalog of Victorian frames and borders [Son and Co. 1976]. We use zoning to control the decorative border. One zone represents interior white space, another zone represents the outer border which must contain connecting tiles, and a transition zone between these allows for either blank or decorative tiles. Symmetry was achieved by synthesizing half of the frame and mirroring the result.

Figure 15 shows the ability of our algorithm to mix styles from different city layouts. Using the city tile set, we created two exemplars capturing different styles: suburban and urban. The urban city layout had a grid-like road structure and contained taller buildings. The suburban city layout had less structured roadways and contained mostly houses. We synthesized three layouts: one using the suburban exemplar, one using the urban exemplar, and one where zoning constraints were used to combine the two styles. The left portion of the city was zoned using the factors learned from the urban exemplar, and the right was zoned using the factors from the suburban exemplar. In all cases, the distribution was encoded using 2H/2V factors (hard) and square 3x3-tile factors (soft).

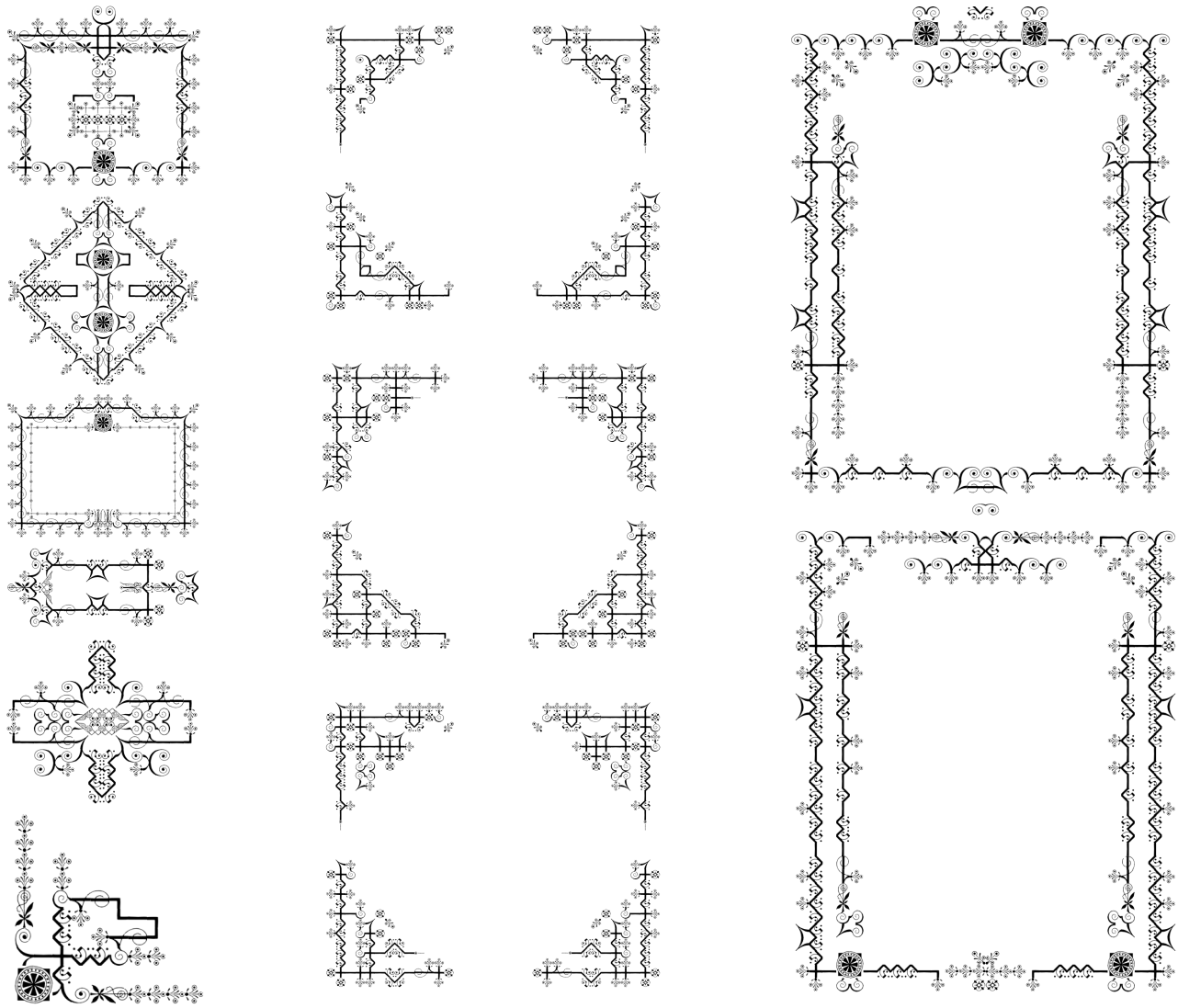


Fig. 14: Left: Exemplars. Middle, Right: Decorative patterns synthesized from these exemplars. Symmetry achieved through mirroring. For all the synthesized decorative patterns, factors are 2H/2V logical factors. Replacement blocks are the same as the ones shown in Figure 9 (e). The tiles were extracted from the book "Victorian Frames, Borders and Cuts: From the 1882 Type Catalog of George Bruces Son and Co." ©Dover Publications.

## 9. DISCUSSION AND FUTURE WORK

We have shown that a limited set of exemplars can be used to synthesize tile-based patterns that not only capture overall exemplar appearance but are guaranteed to do so without creating seams. We encode both hard and soft constraints using factor graphs.

As mentioned in Section 2, factor graphs and Markov random fields are two types of probabilistic graphical models. They can both be used to represent the same joint probability distribution. However, one factorization of the joint distribution might be better than the other. In our case, factor functions naturally represent constraints between tiles found in patterns such as facades and cityscapes.

Currently the higher-order factor shapes and weight parameter $\epsilon$ are determined manually. Automatically learning more sophisticated shapes and better-tuned weight parameters is a natural next step to better approximating the exemplar distribution and more faithfully capturing exemplar appearance. There is a large body of work on automatically learning the structure and parameters of graphical models [Koller and Friedman 2009].

There are many interesting avenues for exploring the use of factor graphs in pattern synthesis. One limitation of our method is its reliance on patterns which lie on a regular grid. However, this is not a restriction imposed by the factor graph representation: the structure of the factor graph depends only on the connectivity of abstract elements or symbols, and not on the appearance of tiles. Therefore, the factor graph representation is not limited to 2D grids; it could be extended to 3D grids or other topologies. Factor graphs can also be used to capture other types of semantic relationships, such as requiring all roads in a city be a minimum distance apart.

While these future directions pertain to pattern synthesis, factor graphs are not limited to this problem area. They provide a unified framework for the representation of probabilistic distributions by explicitly encoding dependencies using factors. The flexibility of this method makes it applicable to many areas of graphics; local dependencies between tiles easily map to those between more general spatial elements in images or models. For example, factor graphs could encode interior design principles for scene modeling applications. Similarly, factors could be used to capture the relationships between keyframes in animations or other types of temporal sequences in simulations.

## Acknowledgement

We are grateful to the reviewers for their helpful comments. We thank Dover Publications for granting us permission to use the images behind our tile sets. Support for this research was provided by a Stanford Graduate Fellowship, the Fannie and John Hertz Foundation, a National Science Foundation Graduate Research Fellowship, the ISTC-VC Intel Science and Technology Center for Visual Computing fund, and the DARPA SEEC grant HR0011- 11-C-0007-P00003.

## REFERENCES

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *Proceedings of the 2001 symposium on Interactive 3D graphics*. ACM, 217–226.

BOKELOH, M., WAND, M., AND SEIDEL, H.-P. 2010. A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graph. 29,* 4.

CAMPBELL, C. 2006. *Vitruvius Britannicus: The Classic of Eighteenth-Century British Architecture*. Dover Publications.

CHO, T. S., AVIDAN, S., AND FREEMAN, W. T. 2010. A probabilistic image jigsaw puzzle solver. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

COHEN, M., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. *ACM Transactions on Graphics 22,* 3, 287–294.

EDWARDS, R. AND SOKAL, A. 1988. Generalization of the Fortuin-Kasteleyn-Swendsen-Wang representation and Monte Carlo algorithm. *Physical review D 38,* 6, 2009–2012.

EFROS, A. AND FREEMAN, W. 2001. Image quilting for texture synthesis and transfer. In *ACM SIGGRAPH 2001*.

EFROS, A. A. AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. In *ICCV (2)*. 1033–1038.

GENT, I., JEFFERSON, C., AND MIGUEL, I. 2006. MINION: A fast, scalable, constraint solver. In *Proceeding of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29–September 1, 2006, Riva del Garda, Italy*. IOS Press, 98–102.

GOLDBERGER, J., GORDON, S., AND GREENSPAN, H. 2003. An efficient image similarity measure based on approximations of KL-divergence between two Gaussian mixtures. In *In Proc. ICCV*. 487–493.

HASTINGS, W. K. 1970. Monte carlo sampling methods using markov chains and their applications. *Biometrika 57,* 1, pp. 97–109.

KOLLER, D. AND FRIEDMAN, N. 2009. *Probabilistic graphical models*. MIT Press.

KSCHISCHANG, F. R., FREY, B. J., AND LOELIGER, H.-A. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory 47*, 498–519.

KWATRA, V., SCHDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003 22,* 3 (July), 277–286.

LAGAE, A., KAPLAN, C. S., FU, C.-W., OSTROMOUKHOV, V., AND DEUSSEN, O. 2008. Tile-based methods for interactive applications. In *ACM SIGGRAPH 2008 classes*. SIGGRAPH '08. ACM, New York, NY, USA, 93:1–93:267.

LOELIGER, H. A. 2004. An introduction to factor graphs. *Signal Processing Magazine, IEEE 21,* 1, 28–41.

MERRELL, P. 2007. Example-based model synthesis. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*. ACM, New York, NY, USA, 105–112.

MIRA, A. AND TIERNEY, L. 1997. On the use of auxiliary variables in Markov chain Monte Carlo sampling. In *Scandinavian Journal of Statistics*.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND GOOL, L. V. 2006. Procedural modeling of buildings. *ACM Trans. Graph. 25,* 3, 614–623.

NEAL, R. 2000. Slice sampling. *Annals of Statistics 31*, 705–767.

PARISH, Y. I. H. AND MÜLLER, P. 2001. Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM Press, New York, NY, USA, 301–308.

PASKIN, M. 2002. Maximum-Entropy Probabilistic Logics. Tech. rep., University of California at Berkeley.

POON, H. AND DOMINGOS, P. 2006. Sound and efficient inference with probabilistic and deterministic dependencies. In *AAAI'06: Proceedings of the 21st national conference on Artificial intelligence*. AAAI Press, 458–463.

PRUSINKIEWICZ, P. AND LINDENMAYER, A. 1996. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., New York, NY, USA.

SON, G. B. AND CO. 1976. *Victorian Frames, Borders and Cuts: From the 1882 Type Catalog of George Bruce's Son and Co.* Dover Publications.

WEI, L.-Y., LEFEBVRE, S., KWATRA, V., AND TURK, G. 2009. State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*. Eurographics Association.

WEI, L.-Y. AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *Siggraph 2000, Computer Graphics Proceedings*, K. Akeley, Ed. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 479–488.

WEI, W., ERENRICH, J., AND SELMAN, B. 2004. Towards efficient sampling: exploiting random walk strategies. In *AAAI'04: Proceedings of the 19th national conference on Artifical intelligence*. AAAI Press / The MIT Press, 670–676.

WONG, M. T., ZONGKER, D. E., AND SALESIN, D. H. 1998. Computer-generated floral ornament. *Computer Graphics 32,* Annual Conference Series, 423–434.

YOKOO, M. 1997. Why adding more constraints makes a problem easier for hill-climbing algorithms: Analyzing landscapes of csps. *Principles and Practice of Constraint Programming-CP97*, 356–370.

ZHU, S. C., WU, Y., AND MUMFORD, D. 1998. Filters, random fields and maximum entropy (frame) – towards a unified theory for texture modeling. *International journal of Computer Vision 27,* 2, 1–20.

(a)

Urban Suburban

(b)

(c)

(d)



Fig. 15: (a) Exemplars. (b) Synthesis from urban exemplar. (c) Synthesis from suburban exemplar. (d) Synthesis using factors from both exemplars. Zoning was used to create an urban area (lower left) and a suburban area (upper right). For all the synthesized city patterns, factors are 2H/2V logical factors and square 3x3 higher-order factors. Replacement blocks are the same as the ones shown in Figure 9 (e).