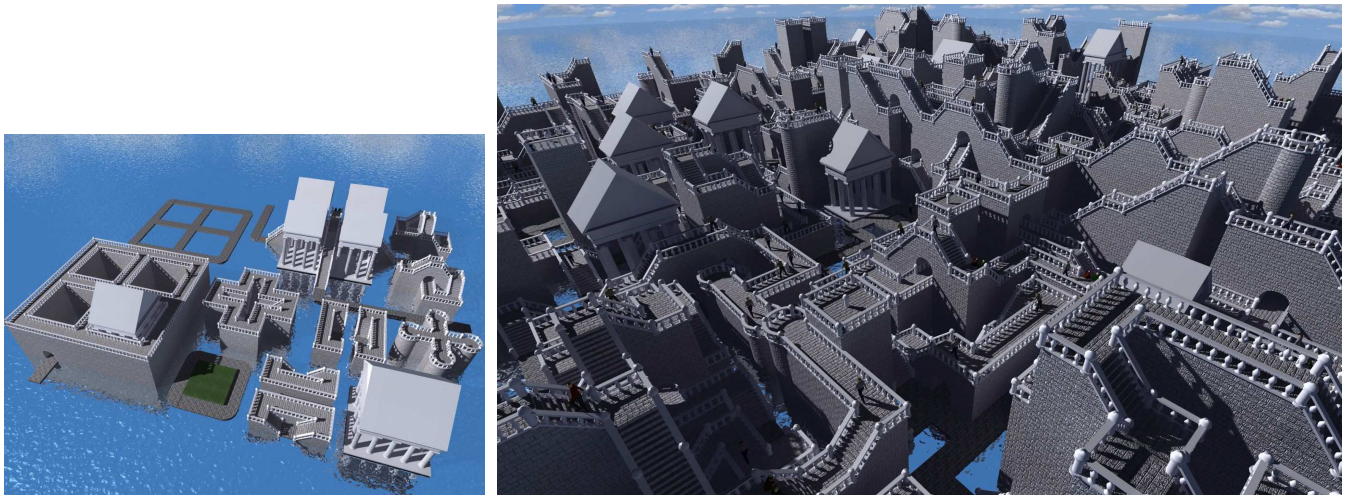
# Example-Based Model Synthesis

Paul Merrell*
University of North Carolina at Chapel Hill



(a) (b)

**Figure 1:** *From the example model (a), a larger model (b) is automatically created using model synthesis.*

## Abstract

*Model synthesis* is a new approach to 3D modeling which automatically generates large models that resemble a small example model provided by the user. Model synthesis extends the 2D texture synthesis problem into higher dimensions and can be used to model many different objects and environments. The user only needs to provide an appropriate example model and does not need to provide any other instructions about how to generate the model. Model synthesis can be used to create symmetric models, models that change over time, and models that fit soft constraints. There are two important differences between our method and existing texture synthesis algorithms. The first is the use of a global search to find potential conflicts before adding new material to the model. The second difference is that we divide the problem of generating a large model into smaller subproblems which are easier to solve.

**CR Categories:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—;

**Keywords:** procedural modeling, texture synthesis

## 1 Introduction

Many of the most visually exciting environments such as vast landscapes and cityscapes are quite large and quite intricate. Due to their size and complexity, the task of modeling these environments is often a long and tedious process. This difficult task could be eliminated if there were a suitable method to model these types of environments automatically.

Example-based techniques are widely used for texture synthesis. In texture synthesis, the user inputs an example texture and the algorithm outputs a more extensive texture that resembles the example texture. A similar approach could be used to model large 3D environments. The user inputs a small example model and then a computer algorithm outputs a larger model that resembles the example model. This process is illustrated in Figure 1. Figure 1(a) shows an example model and Figure 1(b) shows the new synthesized model generated by computer. The new model looks similar to the example model, but is larger and more complex. Model synthesis requires that it be possible to break apart the example model into small building blocks that can be arranged together on a 3D grid. Because model synthesis accepts many different example models, it is a general-purpose procedural modeling tool.

3D artists need considerable creative freedom to be able to create models that look both realistic and artistically interesting. This is why it is important for them to have general-purpose modeling tools. A procedural modeling technique that can only model a specific type of object often has limited value, because as soon as a significantly different object is needed, the technique must be reprogrammed. We expect that 3D artists will find that creating a new example model is often easier than adjusting an existing procedural modeling technique to suit their needs.

### 1.1 Related Work

Model synthesis is closely related to texture synthesis. Recently, many techniques have been developed for synthesizing texture. One group of methods could be categorized as global methods which attempt to match the large-scale stochastic properties of the example

*e-mail: pmerrell@cs.unc.edu

texture onto the new texture [Heeger and Bergen 1995; Portilla and Simoncelli 2000]. In a separate category are local region growing methods which synthesize the texture pixel by pixel or patch by patch [Efros and Leung 1999; Wei and Levoy 2000; Efros and Freeman 2001]. Model synthesis more closely resembles the local methods.

Model synthesis could be thought of as a generalization of texture synthesis into three or more dimensions. Previous extensions of texture synthesis used time as the third dimension [Doretto et al. 2003; Wei and Levoy 2000; Kwatra et al. 2003]. Texture synthesis has also been used to create 3D geometric texture on the surface of a given model [Bhat et al. 2004].

Wang tiles are small blocks of texture that can be arranged together on a grid to create larger textures and have been used in texture synthesis [Cohen et al. 2003]. The 3D counterpart of a Wang tile is a Wang cube [II and Kari 1996]. Wang cubes have been used to model asteroid fields [Sibley et al. 2004] and to render volume data [Lu et al. 2004]. Synthesizing a model is not difficult after a Wang cube set has been found. However, in order to form an acceptable Wang cube set many different parts of the model must be stitched together without creating seams where they meet. This can be difficult to achieve on some models.

Context-based surface completion [Sharf et al. 2004] is designed to complete models where sections of the model are missing surface information. Surface completion fills in any missing sections with surfaces that resemble the rest of the model. The rest of the model effectively acts as the example. Another tool [Funkhouser et al. 2004] can be used to stitch together parts of many different example models to create a new model. These two methods are useful for modeling individual objects, but model synthesis is better suited to modeling large-scale structures.

Procedural modeling techniques that model specific objects or environments such as plants [Měch and Prusinkiewicz 1996; Prusinkiewicz et al. 2001], terrain [Musgrave et al. 1989], and buildings [Muller et al. 2006; Legakis et al. 2001] have been extensively studied [Ebert et al. 1998], but these methods require that many rules for generating the models be specified and are only able to model a small class of objects. In contrast, model synthesis is a general-purpose procedural modeling tool. While the example model must satisfy a few requirements, any example model that does satisfy them can be used regardless of the type of object.

## 2 The Consistency Problem

To describe the models, we use a set of predefined *model pieces*. Model pieces are the building blocks of a model. The pieces are arranged together in space to form models. Figure 2(a) demonstrates how a model of a pillar can be constructed using four distinct model pieces. Every position in a 3D lattice is assigned an integer value corresponding to the model piece that occupies the space. In this case, there are four different model pieces numbered 0 through 3.

The model pieces are constructed manually. The example model provided by the user is assembled from the model pieces. It must be possible to decompose the example model into a few unique pieces.

A model will not look plausible if it is pieced together haphazardly. For example, Figure 2(b) shows what happens when the pieces are arranged together without any rules. The result is a entirely unsatisfactory. Since empty space pieces are allowed to be below the other pieces 1 through 3, some of the columns in the model are levitating. To avoid generating nonsensical models like this, a set of rules must be established to ensure that the pieces all fit together correctly and

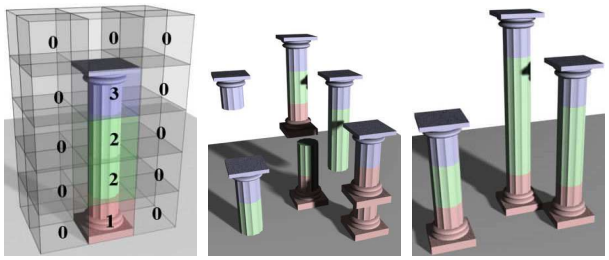



**Figure 2:** *(a) A model composed of four model pieces, (b) An Inconsistent Model, (c) A Consistent Model*

seamlessly. One rule is that a model piece may only be beneath another model piece if it was beneath that piece in the example model. Similarly, a model piece may only be behind another piece if it was behind that piece in the example model. If a model obeys all these rules, then it is *consistent*. More precisely, a model $M$ is consistent with an example model $E$ if all the model pieces that are adjacent to one another in $M$ are found adjacent to one another in $E$ along the same direction. Consistency is tested in six directions which are the positive and negative $x$, $y$, and $z$ directions.

## 3 Method

Model synthesis is performed on a three-dimensional lattice which consists of a set of vertices $V$ connected by a set of edges. Each vertex is occupied by a model piece. Each model piece at a particular position in space will be represented by a label assigned to a particular vertex. If there are $N$ possible model pieces, then there are $N$ possible labels. The problem of model synthesis is how to assign a label to each vertex without violating the rules of consistency. A labeled vertex $c = (v, k)$ is a vertex $v \in V$ that has been assigned a label $k \in [1 \dots N]$. A model $M$ is defined as a set of labeled vertices. Two labeled vertices may not occupy the same space. A model that is unfinished or incomplete will have some vertices that are missing labels. If a model contains one labeled vertex for each vertex in $V$, then it is *complete*.

The transition function $T$ is a Boolean function that controls how the vertices are labeled. Let $c_1$ and $c_2$ be two adjacent labeled vertices. If $c_1$ and $c_2$ are allowed to be next to one another, $T(c_1, c_2)$ is equal to one. Otherwise, it is zero. For non-adjacent vertices, $T(c_1, c_2)$ is defined to be one. A complete model $M$ is consistent if for any two labeled vertices $c_1, c_2 \in M, T(c_1, c_2) = 1$. An incomplete model $M$ is consistent if there exists a complete consistent model $M'$ such that $M \subset M'$.

In the three-dimensional Cartesian case, $T$ is calculated from the example model $E$ according to the equations

$$T((v_1,k_1),(v_2,k_2)) = \begin{cases} T_x(k_1,k_2) & ,v_1 = v_2 + (1,0,0) \\ T_x(k_2,k_1) & ,v_1 = v_2 - (1,0,0) \\ T_y(k_1,k_2) & ,v_1 = v_2 + (0,1,0) \\ T_y(k_2,k_1) & ,v_1 = v_2 - (0,1,0) \\ T_z(k_1,k_2) & ,v_1 = v_2 + (0,0,1) \\ T_z(k_2,k_1) & ,v_1 = v_2 - (0,0,1) \\ 1 & ,\text{otherwise} \end{cases}$$

$$T_x(k_1,k_2) = \begin{cases} 1 & ,\exists v | (v,k_1),(v+(1,0,0),k_2) \in E \\ 0 & ,\text{otherwise} \end{cases}$$

$$T_y(k_1,k_2) = \begin{cases} 1 & ,\exists v | (v,k_1),(v+(0,1,0),k_2) \in E \\ 0 & ,\text{otherwise} \end{cases}$$

$$T_z(k_1, k_2) = \begin{cases} 1 & , \exists v | (v, k_1), (v + (0, 0, 1), k_2) \in E \\ 0 & , \text{otherwise} \end{cases} \quad (1)$$

### 3.1 Global Search for Conflicts

Our goal is to create a complete consistent model. We can start with an empty model and individually add labeled vertices to it until it is complete. However, if we are not careful it is likely that the model will become inconsistent as more labeled vertices are added to it. For some labeled vertices, it is relatively easy to see that the model will become inconsistent if they are added to it. We use a global search to find these types of labeled vertices and remove them from consideration. Once these labeled vertices are removed, we are left with a set of candidate labels that we are considering adding to the model $M$ and this set is called $C(M)$. $C(M)$ is updated every time the model changes.

A detailed explanation of how to calculate $C(M)$ is provided in Table 1 and a simple example of the calculation is provided in Figure 3. This calculation requires several iterations. $C_t(M)$ is the estimated value of $C(M)$ at iteration $t$. During each iteration, labeled vertices that do not belong in $C(M)$ are removed. After many iterations, $C_t(M)$ will be equal to the desired value of $C(M)$. At each vertex that has been assigned a label in $M$, all other labels are removed in Step 1 since each vertex may only have one label. Whenever $C_t(M)$ changes at a vertex, all of its neighbors must be checked to see if any of them are affected by the change. Let $u_t$ be the set of all vertices that have been changed, but whose neighbors have not been checked. In Step 3, a vertex $v$ is selected out of $u_t$ and then its neighbors are updated in Step 4. A labeled vertex only belongs in $C_t(M)$ if each of its neighbors has at least one possible label that is consistent with it. All labels that do not belong are removed in Step 4. All of the vertices that changed in Step 4 are added to the set $u_{t+1}$ in Step 5. This process then repeats itself. A new vertex is selected out of $u_t$, its neighbors are updated, and then any of its neighbors that have changed are added to $u_t$. This continues until all parts of the model that need to be checked have been checked, which occurs when $u_t$ is empty.

This method improves upon existing texture synthesis methods that only examine a local neighborhood before synthesizing texture. A local search may miss some conflicts that a global search would detect, since many model pieces have an influence far outside their local neighborhood.

Although a global search is better than a local search, it also is imperfect. This global search only eliminates labeled vertices that are relatively easy to eliminate. For some labeled vertices, it can be extraordinarily difficult to decide if the model will become inconsistent if they are added to it. Deciding whether or not a model is consistent is shown to be an NP-complete problem in the appendix. (This proof applies to both 2D textures and 3D models.) Deciding if a model with an additional labeled vertex is consistent is also NP-complete. This means that we must limit the size of the models that are being checked for consistency which is achieved by operating on small parts of the model separately which is discussed in Section 3.2. Creating a single consistent model is not a difficult problem. Most models contain some empty space. An entire model full of empty is consistent, but not very interesting. Even though this solution is not very useful by itself, it can be used as an initial solution that is improved step by step over time.

### 3.2 Synthesis

A detailed description of the model synthesis algorithm is given in Table 2. An example is shown in Figure 4. We begin with a trivial solution which normally consists of empty space over a ground

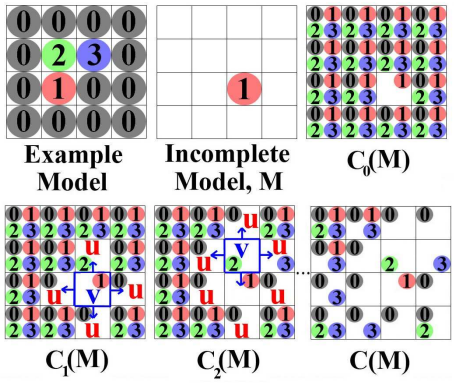



**Figure 3:** *A 2D example of the $C(M)$ Calculation*

| | |
|---|---|
| Step 1: | $C_0(M)$ is the set of all labeled vertices except those occupying the same space as another vertex in $M$: $C_0(M) = V \times [1 \dots N] - \{(v,k) \vert \exists i \neq k, (v,i) \in M\}$ |
| Step 2: | $u_0$ is the location of all vertices in $M$: $u_0 = \{v \vert (v,i) \in M\}$, $t = 0$ |
| Repeat | Steps 3-6 while $u_t \neq \emptyset$ |
| Step 3: | Select a vertex $v$ from $u_t$ |
| Step 4: | Remove all neighboring vertices that do not agree with any of the labeled vertices at $v$: $C_{t+1}(M) = C_t(M) - \{(v',k) \vert \nexists i, (v,i) \in C_t(M)$ and $T((v,i),(v',k)) = 1\}$ |
| Step 5: | Add all locations that changed into $u$ and remove $v$: $u_{t+1} = (u_t - v) \cup \{v' \vert \exists k, (v',k) \in C_t(M) - C_{t+1}(M)\}$ |
| Step 6: | Increment $t$ |

**Table 1:** *$C(M)$ Calculation*

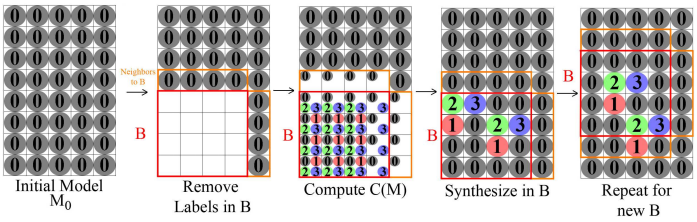



**Figure 4:** *An example illustrating the Model Synthesis Algorithm.*

| | |
|---|---|
| Step 1: | $M_0$ is a simple consistent model, $M = M_0$ |
| Repeat | Steps 2-5 until every part of the model has changed: |
| Step 2: | Choose a set of vertices $B$ to modify |
| Step 3: | Create a new model $M'$ without those vertices: $M' = M - \{(v,k) \vert v \in B\}$ |
| Step 4: | While $C(M') \neq \emptyset$ and $B \neq \emptyset$ Pick $(v,k)$ such that $v \in B$ and $(v,k) \in C(M')$ $M' = M' \cup \{(v,k)\}$ and $B = B - \{v\}$ |
| Step 5: | If $C(M') \neq \emptyset$ then $M = M'$ |

**Table 2:** *Model Synthesis Algorithm*

plane. Then we pick a section of the initial model to modify. Let $B$ be the set of vertices we have chosen to modify. Since the remaining unmodified part of the model is known to be consistent, the consistency only needs to be checked within the $B$ region. By only modifying the $B$ region, the size and complexity of the problem is lowered dramatically. Small parts of the model can be modified much more effectively. We create a new model $M'$ with all the old labels in $B$ removed. For each vertex in $B$, a new label is selected at random from our list of candidate labels $C(M')$ and assigned to the vertex. $C(M')$ can be calculated fairly quickly because the consistency only needs to be checked within the $B$ region. After all the vertices in $B$ have been labeled the changes are accepted in Step 5. However, since $C(M')$ is imperfect, there is a chance that a label will be assigned that causes the model to become inconsistent and $C(M')$ to become empty. When this happens, the new changes are rejected. These events are less likely to occur when the region to modify $B$ is small. After, the changes are accepted or rejected, a new section $B$ is selected and modified. Each vertex should have the chance to be modified a few times. Successive $B$ regions should overlap.

### 3.3 Resemblance

In addition to producing consistent models, another important goal of model synthesis is to create models that resemble the example model. To resemble the example model, each small region of the new model should approximately match other regions in the example. Let $\omega(v, M)$ be a small cubic region of the model $M$ centered on the vertex $v$ which has a width of $2w + 1$ pieces.

$$\omega(v, M) = \{(q, k)|(v + q, k) \in M \text{ and } q \in [-w \dots w]^3\} \quad (2)$$

The perceptual distance $d$ between two sets is defined as the number of model pieces that are not shared between the two sets which is:

$$d(P, Q) = |P| - |P \cap Q| \quad (3)$$

where $|P|$ is the cardinality of the set $P$.

Models that more closely resemble the example model will be produced more frequently, if the following modification is applied to the algorithm in Table 2 at Step 4. In Step 4, one label is chosen at the vertex $v$. This label should be chosen so that the perceptual distance between the new model and the example mode is minimized. Let $L(k)$ be the number of locations in the example model $E$ that closely resemble the model $M'$ with $(v, k)$ added to it, if $(v, k)$ is consistent

$$L'(k) = |\{v'|d(\omega(v', E), \omega(v, M' \cup \{(v, k)\})) \leq d_0\}| \quad (4)$$

$$L(k) = \begin{cases} L'(k) & , (v, k) \in C(M') \\ 0 & , (v, k) \notin C(M') \end{cases} \quad (5)$$

where $d_0$ is a threshold normally set at the minimum perceptual distance. In Step 4, the probability of picking the label $k$ will be assigned the value of:

$$P(k) = \frac{L(k)}{\sum_{i=1}^{N} L(i)}. \quad (6)$$

This will cause the new model to more closely resemble the example model.

## 4 Variants of Model Synthesis

### 4.1 Symmetric Models

Symmetric models can be synthesized using model synthesis. Many different kinds of symmetry can be described using a Boolean function of two labeled vertices called the symmetry function $S(c_1, c_2)$. $S(c_1, c_2)$ is similar to the transition function $T$. The transition function $T$ describes which labeled vertices are allowed to be in adjacent locations. The symmetry function $S$ describes which labeled vertices are allowed to be in symmetrical locations. When the two vertices $v_1$ and $v_2$ are in symmetrical locations, $S((v_1, k_1), (v_2, k_2))$ is equal to one only if the model piece corresponding to the label $k_1$ is symmetric to the model piece corresponding to the label $k_2$. When $v_1$ and $v_2$ are not in symmetric locations, $S((v_1, k_1), (v_2, k_2))$ is equal to one. In the same way that the transition function enforces consistency, the symmetry function enforces symmetry. If the transition function were replaced by the symmetry function, the algorithm would generate symmetric models which might not be consistent. We can create a new transition function $T'$ that combines the requirements of both the original transition function and the symmetry function, $T'(c_1, c_2) = S(c_1, c_2)T(c_1, c_2)$. By replacing the old transition function with the new one, the algorithm generates models that are both consistent and symmetric. Symmetric models can be created with only minor changes to the algorithms.

### 4.2 Constrained Models

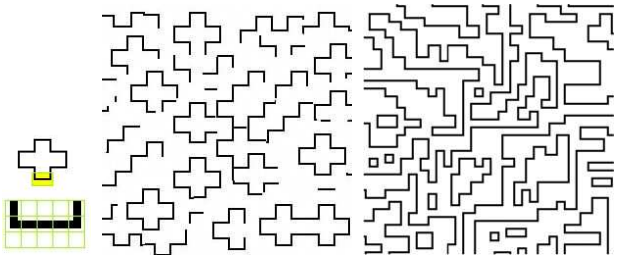
It is possible to add soft constraints to model synthesis, so that the models that are synthesized have certain desirable characteristics. For example, a model of a city could be constrained to be created in the shape of another object. Analogously, there are texture synthesis methods which are designed to create texture that fits some kind of constraint [Efros and Freeman 2001; Ashikhmin 2001; Kwatra et al. 2005].

One way a constraint can be added is based off of the Metropolis sampling algorithm. Let $f(M)$ be a function of the model $M$ that defines the constraint. We wish to find a model $M$ that maximizes $f(M)$. This can be done by making a minor change to our algorithm in Step 5. Originally, Step 5 would occur whenever $M'$ was consistent or, in other words, whenever $M'$ was consistent, Step 5 would occur with a probability of 1. If this is changed so that Step 5 occurs with a probability of $\min\left(1, \frac{f(M')}{f(M)}\right)$, then those models which better fit the constraint are more likely to be synthesized.

Another way to add a constraint is to modify Equation (6), so that the labels that better fit the constraint have a higher probability of being selected.

### 4.3 Higher-Dimensional Models

Model synthesis can be used to create four-dimensional and higher-dimensional objects. By adding a time dimension, it is possible to synthesize time-varying models. The process of synthesizing a time-varying model is almost identical to the process of synthesizing a stationary model. The user inputs a time-varying example model constructed out of time-varying model pieces, and the algorithm synthesizes a time-varying model that resembles the input. The same principles that applied in three dimensions apply in four dimensions. Temporal consistency is just as important as spatial consistency. In the same way that only some model pieces are allowed to be adjacent to one another in space, only some model pieces are allowed to be adjacent to one another in time. The extension into higher dimensions is fairly straightforward.



(a) (b) (c)

**Figure 5:** *(a) Example Texture, (b) Kwatra et al. 2005, (c) 2D Model Synthesis, Part of the example texture is magnified beneath the original to show the 4 × 4-pixel texture pieces.*

## 5 Results

A two-dimensional version of the algorithm was used to compare it with another texture synthesis algorithm. The example texture is shown in Figure 5(a). This texture was chosen because is it possible to decompose it into a few distinct texture pieces. 2D model synthesis can not be applied to an arbitrary texture as most other texture synthesis methods can, but for those textures that can be decomposed in this way, model synthesis is able to exploit the decomposition. The example model in Figure 5(a) is one continuous path with no dead ends. Model synthesis generates a consistent texture that is made up of only continuous paths as shown in Figure 5(c). The result from an existing texture synthesis algorithm is shown for comparison in Figure 5(b).

Figures 1 and 6 through 13 show a variety of models that were generated from example models, including cities with different architectural styles, plants, terrain, castles, and building interiors. The synthesized models are fairly large and would require a great amount of effort to model manually. Figure 12 shows models with different types of symmetry that are all based off the example model in Figure 1(a). Figure 13 shows models that are constrained to be in the shape of different symbols. The companion video shows a time-varying model of a city with moving cars synthesized from an example model using 4D model synthesis.

Model synthesis can also be used to light environments containing a large number of lights. This is done by including model pieces that have lights in them. In Figure 8, all of the street lights and car lights were generated using model synthesis.

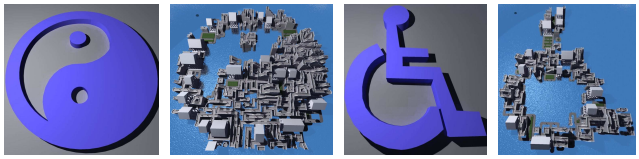
The computation time required to create models using model synthesis depends on the size of the $B$ region that is modified. For some types of models such as the city at night (Figure 8) and the landscape (Figure 9), the algorithm is successful even when the $B$ region is as large the entire new model. Consequently, the models in Figures 8 and 9 were created within a few seconds. The other models need smaller $B$ regions and more iterations to be successful. The models in Figures 1, 6, 7, 10, and 11 took between half a minute and half an hour to create.

## 6 Limitations

An important limitation is that the example models need to be manually decomposed into model pieces. There are some models that would be difficult to decompose in this way. Architectural objects are frequently composed of elements that repeat and are structured in a lattice. These types of objects are well-suited to model synthesis. Other objects may be more difficult. Terrain, trees, and other objects can be modeled using model synthesis, but the example models must be constructed carefully for it to work properly. If the example models are not constructed carefully, it is possible that the output will be so tightly constrained that model synthesis will simply reproduce the example model over and over. As long as the region to modify $B$ is kept reasonably small, the algorithm will not become stuck in a position where further change is impossible.

## 7 Conclusion

A new method for automatically synthesizing models using an example model has been presented. Model synthesis is able to model many different large, complex environments that would be difficult to create manually or from existing procedural modeling techniques. Model synthesis can be extended to generate models in motion, symmetric models, and models that fit constraints. We have shown how to perform a global search to avoid adding any model pieces that directly conflict with other model pieces and that by only modifying part of the model, it is much more likely that the modification will be successful. This allows us to generate large consistent models.



(a) Yin and Yang (b) Example 1a (c) Wheelchair (d) Example 1a

**Figure 13:** *Constrained Models*

## References


ASHIKHMIN, M. 2001. Synthesizing natural textures. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM Press, New York, NY, USA, 217–226.

BHAT, P., INGRAM, S., AND TURK, G. 2004. Geometric texture synthesis by example. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, ACM Press, New York, NY, USA, 41–44.

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

DORETTO, G., CHIUSO, A., SOATTO, S., AND WU, Y. 2003. Dynamic textures. *International Journal of Computer Vision 51*, 2 (February), 91–109.

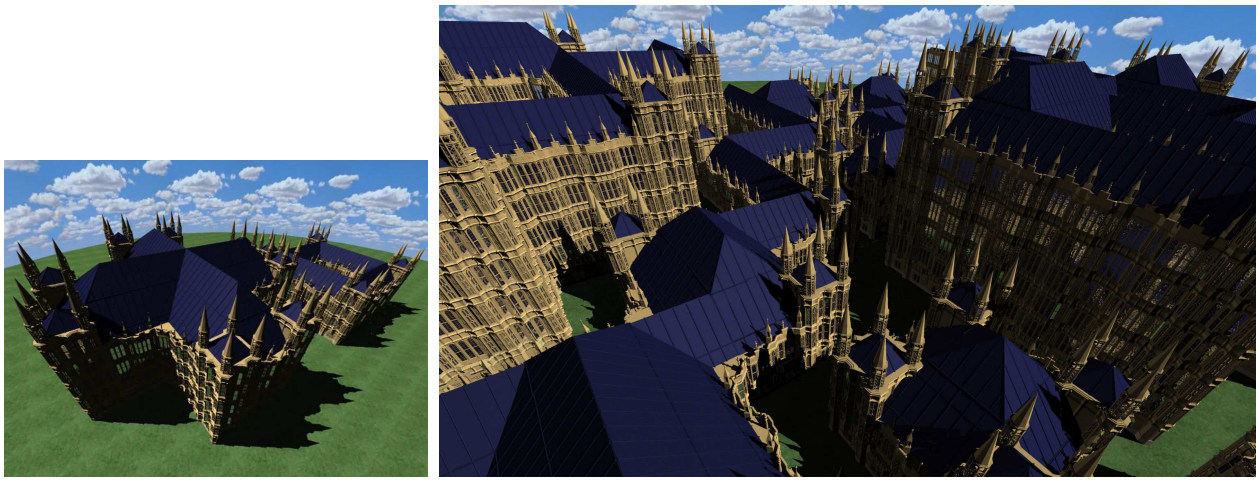
EBERT, D. S., MUSGRAVE, F. K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 1998. *Texturing and Modeling*. 2nd ed. Academic Press.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. *SIGGRAPH '01*, 341–346.

EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision*, 1033–1038.

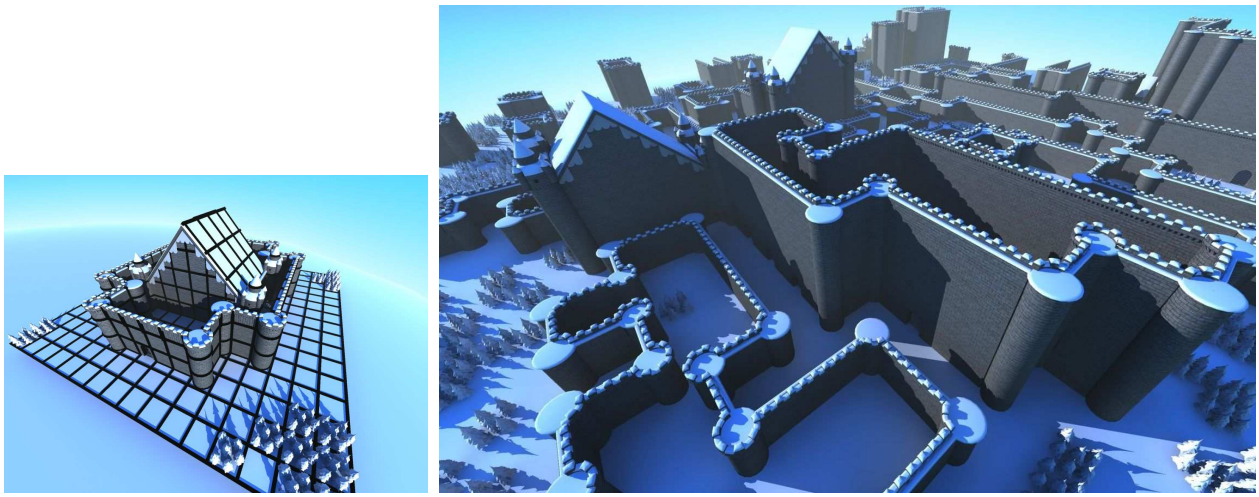
FUNKHOUSER, T., KAZHDAN, M., SHILANE, P., MIN, P., KIEFER, W., TAL, A., RUSINKIEWICZ, S., AND DOBKIN, D. 2004. Modeling by example. *SIGGRAPH '04*.

HEEGER, D. J., AND BERGEN, J. R. 1995. Pyramid-based texture analysis/synthesis. In *SIGGRAPH '95*, 229–238.



(a) Example Model (b) Synthesized Model

**Figure 6:** *Given two buildings (a), model synthesis produces a cluster of buildings (b).*



(a) Example Model (b) Synthesized Model

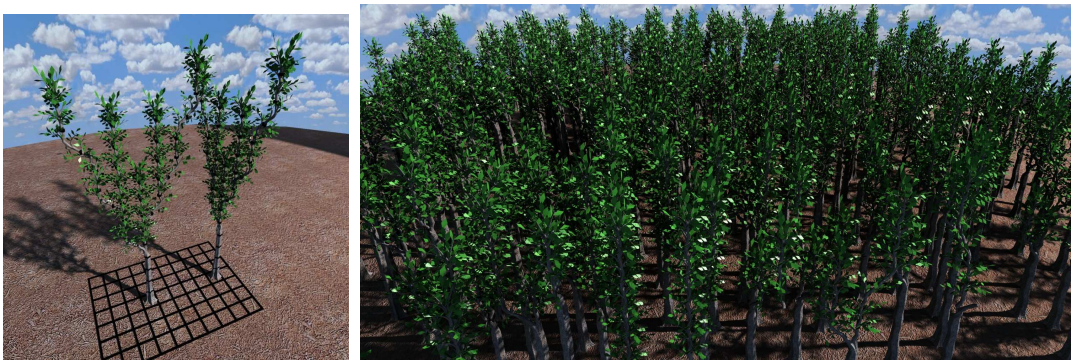
**Figure 7:** *Given a castle wall (a), model synthesis produces many fortifications (b). Grid lines are drawn in the example (a) to show how the model is divided into pieces.*

(a) Example Model (b) Synthesized Model

**Figure 8:** *Given a few buildings (a), model synthesis produces a city (b). Grid lines are drawn in the example model.*

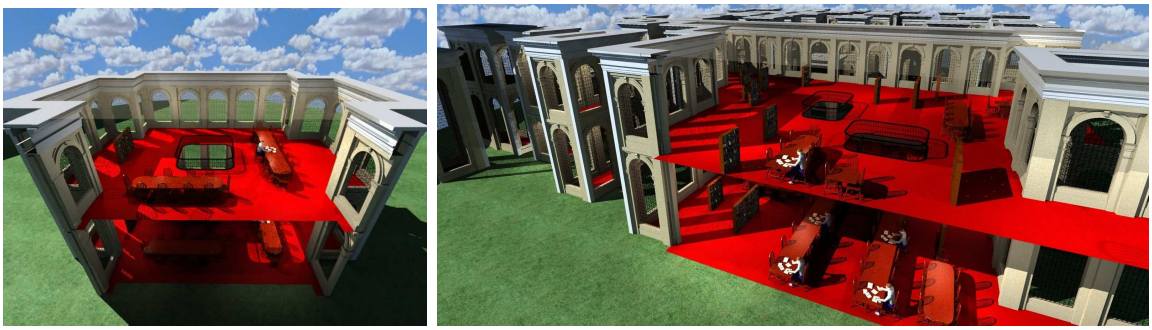
(a) Example Model (b) Synthesized Model

**Figure 9:** *Given a patch of land (a), model synthesis produces a landscape (b). Grid lines are drawn in the example model.*



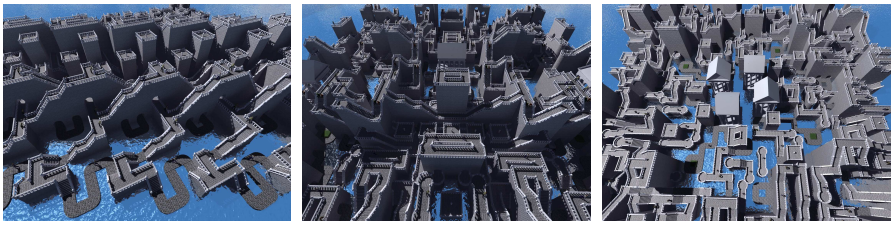
(a) Example Model (b) Synthesized Model

**Figure 10:** *Given two trees (a), model synthesis produces a forest (b). Grid lines are drawn in the example model.*



(a) Example Model (b) Synthesized Model

**Figure 11:** *Given the interior and exterior of a building (a), model synthesis produces several buildings (b). Sections of the buildings are cut away and the roof is made transparent to show the interior more clearly.*



(a) Translational Symmetry (b) Reflectional Symmetry (c) Rotational Symmetry

**Figure 12:** *Three symmetric models based off the example model from Figure 1(a).*

II, K. C., AND KARI, J. 1996. An aperiodic set of wang cubes. In *Symposium on Theoretical Aspects of Computer Science*, 137–146.

KWATRA, V., SCHDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. *SIGGRAPH '03*, 277–286.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *SIGGRAPH '05*.

LEGAKIS, J., DORSEY, J., AND GORTLER, S. 2001. Feature-based cellular texturing for architectural models. In *SIGGRAPH '01*, 309–316.

LU, A., EBERT, D. S., QIAO, W., KRAUS, M., AND MORA, B. 2004. Interactive Volume Illustration Using Wang Cubes. Tech. Rep. TR-ECE-04-05, Purdue University.

MĚCH, R., AND PRUSINKIEWICZ, P. 1996. Visual models of plants interacting with their environment. In *SIGGRAPH '96*, 397–410.

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

MUSGRAVE, F. K., KOLB, C. E., AND MACE, R. S. 1989. The synthesis and rendering of eroded fractal terrains. In *SIGGRAPH '89*, 41–50.

PORTILLA, J., AND SIMONCELLI, E. P. 2000. A parametric texture model based on joint statistics of complex wavelet coeffi - cients. *International Journal of Computer Vision 40*, 1, 49–70.

PRUSINKIEWICZ, P., MÜNDERMANN, L., KARWOWSKI, R., AND LANE, B. 2001. The use of positional information in the modeling of plants. In *SIGGRAPH '01*, 289–300.

SHARF, A., ALEXA, M., AND COHEN-OR, D. 2004. Context-based surface completion. *SIGGRAPH '04*, 878–887.

SIBLEY, P., MONTGOMERY, P., AND MARAI, G. E., 2004. Wang cubes for video synthesis and geometry placement. ACM SIGGRAPH 2004 Poster Compendium, August.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00*, 479–488.

## A Deciding if a Texture can be Completed is an NP-Complete Problem

We will show that deciding whether or not it is possible to fill in an incomplete 2D model or texture and create a complete consistent texture is an NP-complete problem. This problem is in NP since a solution can be guessed and then verified in polynomial time by checking the transition function at each vertex. To show that the problem is NP-hard, we reduce a known NP-complete problem the Planar 3-SAT problem to it. The Planar 3-SAT problem is to decide the satisfiability of a Boolean formula with three literals per clause that can be put into a planar graph. An example is shown in Figure 14(a). The literals are connected by wires into three-input OR gates. One of the three inputs must have a true value for the Boolean formula to be satisfied.

To reduce the problem to a texture completion problem, we construct a texture like that shown in Figure 14(c) which resembles the planar graph. Each literal and the wires coming out of the literal are enclosed by a group of texture pieces. The transition function

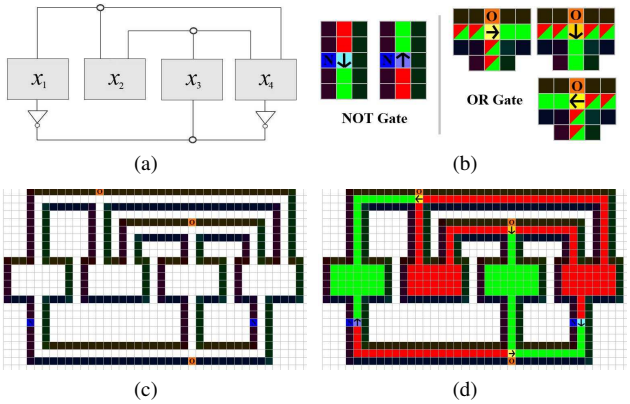



**Figure 14:** *(a) A Planar 3-SAT Problem* $(x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\sim x_1 \vee x_2 \vee \sim x_4)$*, (b) Possible configurations of a NOT and OR gate created from texture pieces, (c) An Equivalent texture synthesis problem, (d) A texture synthesis solution*

$T$ is carefully chosen so that all the texture inside the enclosure is one of two possible varieties which are the TRUE and FALSE texture pieces. The transition function is chosen so that the TRUE and FALSE texture can never touch one another. To be consistent, everything inside the enclosures must be completely TRUE or completely FALSE. The wires may have NOT gates attached to them and the wires meet at the OR gates.

By constructing the proper transition function, we can create NOT gates and OR gates. A NOT gate is created by placing a particular model piece on the wire. This piece always has one of two possible model pieces to the right of it which are shown as blue squares with arrows. The first of these pieces always has a true piece below it and a false piece above it. The second possible piece always has a false piece below it and a true piece above it. In both cases, the value on the wire is negated and so this functions as a NOT gate. An OR gate is created by placing a particular model piece where the three wires meet. This piece always has one of three possible model pieces below it which are shown with arrows. For each of these possible pieces, the TRUE texture piece must be found in the direction the arrow points. The other two directions may have TRUE or FALSE values. The texture can only be completed consistently if a TRUE value is found at one of the three incoming wires.

A solution to the Planar 3-SAT problem exists if and only if it is possible to complete the texture in a way that all the OR gates have at least one TRUE value and the values are negated at the NOT gates. The Planar 3-SAT problem is reduced to the texture completion problem in polynomial time. This result extends to three and higher dimensions, but not to the one-dimensional case.

## Acknowledgements

This work benefited from discussions with Jess Martin, Jeremy Wendt, Philippos Mordohai, and Benjamin Watson. Figure 5(b) was provided by Vivek Kwatra.