Computer-Generated Residential Building Layouts

Paul Merrell Eric Schkufza Vladlen Koltun

Stanford University *

Figure 1: *Computer-generated building layout. An architectural program, illustrated by a bubble diagram (left), generated by a Bayesian network trained on real-world data. A set of floor plans (middle), optimized for the architectural program. A 3D model (right), generated from the floor plans and decorated in cottage style.*

Abstract

We present a method for automated generation of building layouts for computer graphics applications. Our approach is motivated by the layout design process developed in architecture. Given a set of high-level requirements, an architectural program is synthesized using a Bayesian network trained on real-world data. The architectural program is realized in a set of floor plans, obtained through stochastic optimization. The floor plans are used to construct a complete three-dimensional building with internal structure. We demonstrate a variety of computer-generated buildings produced by the presented approach.

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

Keywords: procedural modeling, architectural modeling, computer-aided architectural design, spatial allocation, data-driven 3D modeling

1 Introduction

Buildings with interiors are increasingly common in interactive computer graphics applications. Modern computer games feature sprawling residential areas with buildings that can be entered and explored. Social virtual worlds demand building models with cohesive internal layouts. Such models are commonly created by hand, using modeling software such as Google SketchUp or Autodesk 3ds Max.

This paper presents a method for automated generation of buildings with interiors for computer graphics applications. Our focus is on the building layout: the internal organization of spaces within the building. The external appearance of the building emerges out of this layout, and can be customized in a variety of decorative styles.

We specifically focus on the generation of residences, which are widespread in computer games and networked virtual worlds. Residential building layouts are less codified than the highly regular layouts often encountered in schools, hospitals, and office buildings. Their design is thus particularly challenging, since objectives are less precisely defined and harder to operationalize. Residential layouts are commonly designed in an iterative trial-and-error process that requires significant expertise (Section 3).

Our approach to computer-generated building layout is motivated by a methodology for building layout design commonly encountered in real-world architectural practice. The input to our tool is a concise (and possibly incomplete) list of high-level requirements, such as the number of bedrooms, number of bathrooms, and approximate square footage. These requirements are expanded into a full architectural program, containing a list of rooms, their adjacencies, and their desired sizes (Figure 1, left). This architectural program is generated by a Bayesian network trained on real-world data. A set of floor plans that realizes the architectural program is then obtained through stochastic optimization (Figure 1, middle). The floor plans can be used to construct a three-dimensional model of the building (Figure 1, right). Taken together, the approach provides a complete pipeline for computer-generated building layouts.

Since our method is designed for computer graphics applications, we focus on qualitative visual similarity to real-world residential layouts. We have found that the visual appearance of building lay-

*e-mail:{pmerrell,eschkufz,vladlen}@cs.stanford.edu

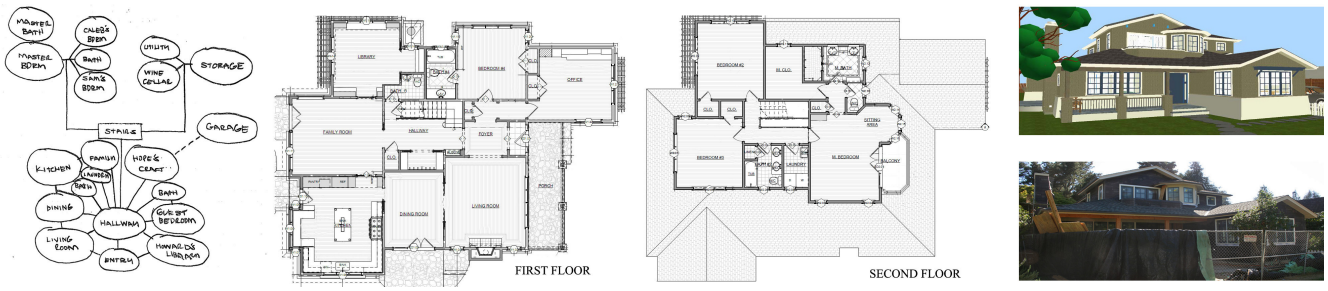


Figure 2: *Artifacts from an architecture practice, illustrating the design of a real-world residence. The client's requirements are refined into an architectural program, illustrated by a bubble diagram (left). The architectural program is used to generate a set of floor plans (middle). The floor plans are used to create a three-dimensional visualization (right, top). Ultimately, construction of the physical building begins (right, bottom). Materials from Topos Architects, reproduced with permission.*

outs arises out of complex considerations of human comfort and social relationships. Despite decades of architectural research, these considerations have resisted complete formalization. This motivates our decision to employ data-driven techniques for automated generation of visually plausible building layouts.

In summary, this paper makes a number of contributions that have not been previously demonstrated:

- Data-driven generation of architectural programs from high-level requirements.
- Fully automated generation of detailed multi-story floor plans from architectural programs.
- An end-to-end approach to automated generation of building layouts from high-level requirements.

2 Related Work

The layout of architectural spaces in the plane is known as the spatial allocation problem. Traditionally, automated spatial allocation aimed to assist architects during the conceptual design process, and focused on producing arrangements of rectangles in the plane, or on the allocation of grid cells. March and Steadman [1971] and Shaviv [1987] review the first 30 years of spatial allocation algorithms. Many classical approaches attempt to exhaustively enumerate all possible arrangements with a specified number of rooms [Galle 1981]. The exponential growth in the number of possible arrangements makes this approach infeasible as the size of the problem increases. Other approaches attempt to find a good arrangement using greedy local search over possible partitions of a regular grid [Shaviv and Gali 1974]. The specific problem of laying out a set of rectangles with given adjacencies in the plane admits elegant graph-theoretic formulations [Lai and Leinwand 1988]. To date, the application of spatial allocation algorithms has been limited to highly codified and regular architectural instances, such as warehouses, hospitals, and schools [Kalay 2004, p. 241].

In light of significant challenges with automated spatial allocation, researchers have explored algorithms that locally tune an initial layout proposed by an architect. Schwarz et al. [1994] developed such a system, inspired by VLSI layout algorithms [Sarrafzadeh and Lee 1993]. Specifically, given a list of rooms and their adjacencies, as well as their rough arrangement in the plane, this arrangement is numerically optimized for desirable criteria. However, the complete specification of rooms and adjacencies, as well as the initial layout, are left to the architect. A similar class of optimization problems, also operating on collections of rectangles in the plane, is known to admit a convex optimization formulation [Boyd and Vandenberghe 2004]. In this case as well, the optimization merely tunes an existing arrangement that must be created manually. A review of optimization techniques for facilities layout is provided by Liggett [2000].

More recently, Michalek et al. [2002] generate layouts for rectangular single-story apartments by searching over the space of connectivity relationships between rooms. This search is performed using an evolutionary algorithm that does not take into account real-world data or any user requirements. The approach is primarily intended for generating a variety of candidate layouts that can be refined by an architect. Arvin and House [2002] apply physically based modeling to layout optimization, representing rooms and adjacencies as a mass-spring system. This heuristic has only been demonstrated on collections of rectangles and is sensitive to the initial conditions of the system.

Harada et al. [1995] introduced shape grammars to computer graphics and developed a system for interactive manipulation of architectural layouts. Shape grammars were subsequently applied to procedural generation of building façades [Müller et al. 2006]. Structural feasibility analysis has been introduced in the context of masonry buildings [Whiting et al. 2009]. Techniques were developed for texturing architectural models [Legakis et al. 2001; Lefebvre et al. 2010], and for creating building exteriors from photographs and sketches [Müller et al. 2007; Chen et al. 2008]. Advanced geometric representations and algorithms were developed for generating architectural freeform surfaces [Pottmann et al. 2007; Pottmann et al. 2008]. However, none of these techniques produce internal building layouts from high-level specifications.

Given a complete floor plan, a variety of approaches exist for extruding it into a 3D building model [Yin et al. 2009]. Automated generation of realistic floor plans is, however, an open problem. Two heuristic approaches for generating building layouts have been proposed in the computer graphics literature. Hahn et al. [2006] generate grid-like internal layouts through random splitting with axis-aligned planes. Martin [2005] outlines an iterative approach to building layout generation, but results are only demonstrated for arrangements of six rectangles. A later poster [Martin 2006] demonstrates a 9-room layout.

In summary, no approach has been proposed that can generate realistic architectural programs from sparse requirements, and no previous work generates detailed building layouts from high-level user specifications. Our work contributes a data-driven approach to automated generation of architectural programs, based on probabilistic graphical models. This enables an end-to-end pipeline for computer-generated building layouts, patterned on the layout design process employed in real-world architecture practices.

3 Building Layout Design

A number of formalisms have been developed in architectural theory that aim to capture the architectural design process, or particular architectural styles [Mitchell 1990]. These models have primarily been used to derive schematic geometric arrangements, rather than detailed floor plans. Formalisms such as shape grammars have so far not yielded models able to produce complete building layouts, akin to ones created by architects in practice [Stiny 2006]. The underlying difficulty is that real-world building layout design does not deal exclusively with geometric shapes and their arrangements. A central role in building layout is played by the function of individual spaces within the building, and the functional relationships between spaces [Hillier and Hanson 1989]. In practice, building layout design relies on a deep understanding of human comfort, needs, habits, and social relationships.

Numerous guidelines have been proposed for the building layout process [Alexander et al. 1977; Susanka 2001; Jacobson et al. 2005], and a few are near-universal in practice. One is the privacy gradient, which suggests placing common areas, such as the living room, closer to the entrance, while private spaces, such as bedrooms, should be farther away. Another concerns room shapes, which should be largely convex and avoid deep recesses, due to the instinctive discomfort sometimes triggered by limited visibility in concave spaces. On the whole, however, the proposed rules of thumb have proved too numerous and ill-specified to be successfully modeled by a hand-designed rule-based system.

Our approach is to apply machine learning techniques to infer aspects of building layout design from data. In order to derive the methods presented in this paper, we have studied the building layout process as it is carried out by residential architects in practice. The balance of this section summarizes this process, which serves as the model for our approach. The presented summary is distilled from interviews and on-site observations at three residential architecture practices in a large suburban area, as well as from published references [Wertheimer 2009; Kalay 2004; Séquin and Kalay 1998]. While there is great variability in the design methods of different architects, this summary presents some significant commonalities.

The first challenge in the process is to expand the incomplete and high-level requirements given by the client into a detailed specification for the residence. " 'I want a three bedroom house for under $300,000' is a typical initial problem statement" [Kalay 2004, p. 206]. From these initial requirements, the architect produces a list of rooms and their adjacencies. An adjacency indicates direct access, such as a door or an open wall. At this stage, the architect often sketches a number of bubble diagrams, in which rooms are represented by ellipses or rounded rectangles, and adjacencies are represented by edges connecting the rooms (Figure 2, left).

Through prototyping with bubble diagrams, the list of rooms and their relationships is progressively refined. The architect toggles between floors, and specifications for one floor are not finalized until the other floors are pinned down. This is a time-consuming iterative process that is often considered to be among the most creative aspects of architectural design. It culminates with an architectural program, a complete list of internal spaces on each floor, their adjacencies, and their rough sizes. Multi-story spaces, such as stairwells and atria, are indicated as such.

After the architectural program is vetted by the client, the architect creates a schematic plan, or concept sketch. This is a rough planar layout of the spaces on each floor, such that adjacent spaces are next to each other, and the spaces have roughly the desired sizes. This stage involves significant trial-and-error, and some architects liken it to "assembling a puzzle." In the final stage of the process, the schematic plan is refined into a detailed floor plan for each floor. At this stage, wall segments are pinned down and doors, windows, and open walls are precisely specified (Figure 2, middle).

Feature	Domain
Total Square Footage	$\mathbb{Z}$
Footprint	$\mathbb{Z} \times \mathbb{Z}$
Room	{bed, bath, . . . }
Per-room Area	$\mathbb{Z}$
Per-room Aspect Ratio	$\mathbb{Z} \times \mathbb{Z}$
Room to Room Adjacency	{true, false}
Room to Room Adjacency Type	{open-wall, door}

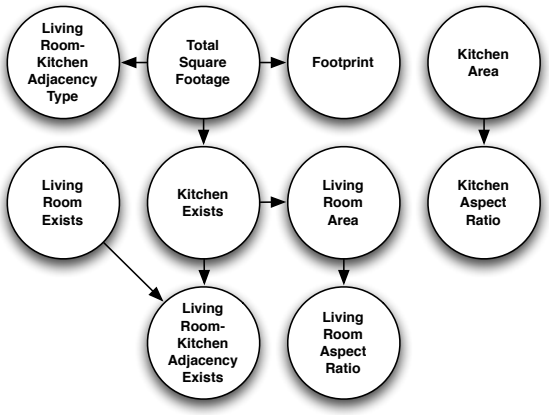


Figure 3: *Representing a distribution of architectural programs with a Bayesian network. The table (top) summarizes the types of features that were extracted from real-world data. An example Bayesian network, trained on a hypothetical corpus of mountain cabins, is illustrated below. Note that this is a didactic example: networks trained on real-world architectural programs are much larger.*

While the external appearance of the house is considered during the layout design process, practicing residential architects often regard the exterior style to be largely independent of the internal building layout. Floor plan design is commonly governed by considerations of comfort and accessibility. Exterior trim, as well as distinctive windows and entrances, are applied to customize the house in styles such as "American Craftsman" or "Colonial Revival."

4 Data-driven Architectural Programming

The first stage of our building layout pipeline expands a sparse set of high-level requirements – such as the desired number of bedrooms, bathrooms, and floors – into a complete architectural program. The architectural program specifies all the rooms in the building, each room's desired area and aspect ratio, and all adjacencies between rooms.

Real-world architectural programs have significant semantic structure. For example, a kitchen is much more likely to be adjacent to a living room than to a bedroom. As another example, the presence of three or more bedrooms increases the likelihood of a separate dining room. Such relationships are numerous and are often implicit in architects' domain expertise. It is not clear how they can be represented with a hand-specified set of rules, or with an ad-hoc optimization approach.

A data-driven technique is therefore more appropriate for capturing semantic relationships in architectural programs. A natural approach would be to encode the existence probability of any room type and any adjacency between rooms of specific types. We could then sample a set of rooms and a set of adjacencies between them. However, this approach does not take into account conditional dependencies between multiple rooms and adjacencies, and can generate unrealistic architectural programs. For example, the frequent occurrence in the data of a bedroom-bathroom adjacency and a kitchen-bathroom adjacency could lead to architectural programs being generated with kitchen-bathroom-bedroom paths, which have very low likelihood.

To learn structured relationships among numerous features in architectural programs, we use probabilistic graphical models [Koller and Friedman 2009]. Specifically, we train a Bayesian network on a corpus of real-world programs. The Bayesian network compactly represents a probability distribution over the space of architectural programs. Once the Bayesian network is trained, we can sample from this distribution. Crucially, we can also fix the values of any subset of features – such as number of bedrooms and bathrooms, total square footage, and areas of specific rooms – and generate complete architectural programs conditioned on those values. Any subset of variables in the Bayesian network can be used as a set of high-level requirements.

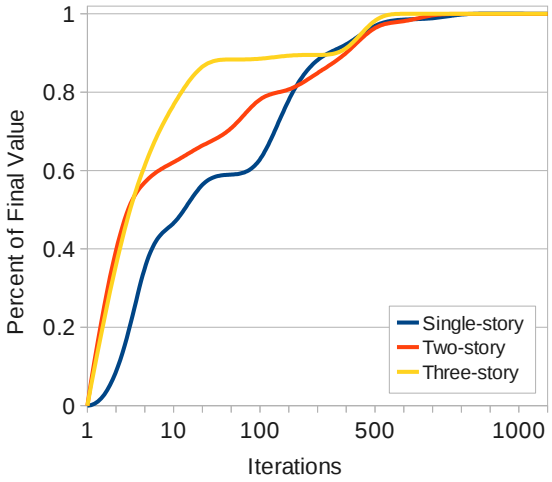
4.1 Bayesian Networks

For the purpose of this work, we manually encoded 120 architectural programs from an extensive catalogue of residential layouts [Wood 2007]. Figure 3 (top) summarizes the attributes that were recorded for each instance. Globally, we recorded total square footage and bounding footprint. On a per-room basis, we recorded type, square footage, and aspect ratio of bounding rectangle. On an inter-room basis, we recorded whether rooms are adjacent, and if so, whether the adjacency is open-wall or mediated by a door. Continuous attributes, such as room area, were quantized to simplify parameter estimation in structure learning (Section 4.2).

The space of architectural programs encoded in this fashion is extremely large. Inferring a probability distribution over this space, even given hundreds of exemplars, is a challenging task. Fortunately, our data is highly structured: there is a strong statistical relationship between different features. Room type, for instance, is a strong predictor of the room's square footage and aspect ratio. We leverage this structure to make the inference tractable using Bayesian networks.

To illustrate the application of Bayesian networks to architectural programming, Figure 3 (bottom) shows a Bayesian network for representing the underlying distribution of a hypothetical corpus of mountain cabins. We assume that the corpus consists entirely of cabins containing a living room and an optional kitchen. All the types of features present in Bayesian networks used in this work are illustrated in the example.

The node `Total Square Footage` encodes the distribution over square footages observed in the corpus. The node `Footprint` encodes the distribution over bounding footprints. The incoming edge from `Square Footage` indicates a conditional dependence on its distribution: large footprints are more likely given large square footage. The same is true of the nodes `Kitchen Exists` and `Living Room-Kitchen Adjacency Type`: kitchens are more likely given larger square footages, and a door (rather than an open wall) between the rooms is more likely given a square footage that is larger still. Although the example network contains a single existence node for each of the living room and kitchen, Bayesian networks for more complex



Network	Nodes	Edges	Training time (s)
Single-story	42	33	1452
Two-story	82	61	4407
Three-story	120	92	8293

Figure 4: *Bayesian network structure learning. Top: Log-likelihood score as a function of training iterations, for each network. Bottom: Total training times.*

corpuses will have multiple existence nodes for some room types, such as bedrooms. Finally, the two incoming edges to the `Living Room-Kitchen Adjacency Exists` node suggest a functional dependence on the existence of both a living room and a kitchen. The remainder of the example network is structured similarly.

Given a trained Bayesian network and a set of high-level requirements, we can automatically generate architectural programs that are compatible with the requirements. The requirements are used to fix the values of corresponding variables in the network. Values for the other variables are generated by sampling from the network. The distributions on the unobserved variables, which are induced by the values of the observed variables, can be obtained using the junction tree algorithm [Lauritzen and Spiegelhalter 1988].

4.2 Structure Learning

Given a set of variables and a corpus of training data, we need to construct a Bayesian network on the variables, such that the structure and parameters of the network maximize the posterior probability of the structure given the data. Structure learning is NP-hard and exact algorithms are super-exponential in the number of variables. We use a standard local search heuristic for structure search, which explores the space of possible structures by adding, removing, or flipping a single edge in the network at a time. The algorithm attempts to maximize the Bayesian score

$$\log p(D, S^h) = \log p(D|S^h) + \log p(S^h),$$

where D is the training data and S^h is a model structure. The prior $p(S^h)$ is taken to be uniform. The marginal likelihood $p(D|S^h)$ is approximated using the Bayesian information criterion. Further details on this procedure are provided by Heckerman [1999].

The amount of training data required for structure learning grows rapidly with the number of variables in the network. For this rea-

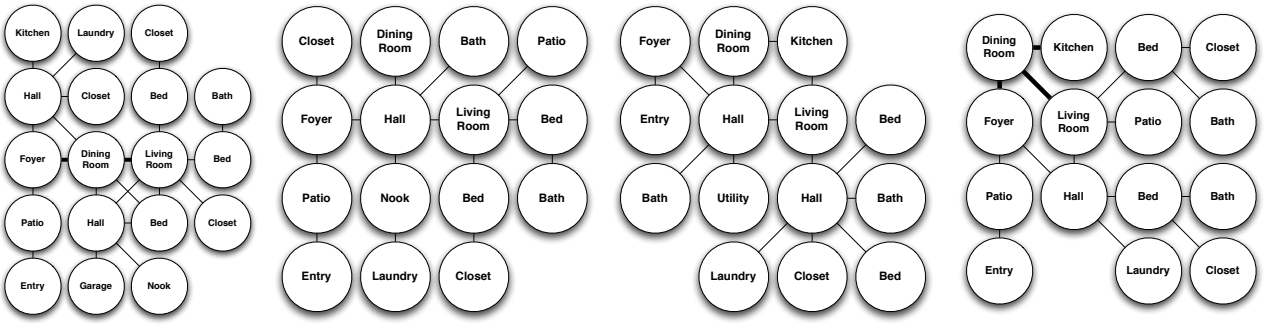


Figure 5: *Bubble diagrams that visualize architectural programs generated by a Bayesian network after 10 structure learning iterations (a), 100 iterations (b), and 1000 iterations (c). The diagram in (a) features a low-likelihood clique between a dining room, living room, bedroom, and hallway. The diagram in (b) is missing a kitchen. The diagram in (c) features plausible connectivity and reflects the privacy gradient. An architectural program designed by an architect is visualized in (d) for comparison. Bold edges indicate open-wall connections.*

son, we use known properties of architectural programs to minimize the number of variables exposed to structure learning. Specifically, to simplify the learning of the dependence between room areas and room aspect ratios, we omit the aspect ratio variables during structure learning and add them to the learned network in an automatic post-processing step. In the case of the cabin example, the nodes `Living Room Aspect Ratio` and `Kitchen Aspect Ratio` would have been omitted from training, and connected respectively to `Living Room Area` and `Kitchen Area` after the rest of the structure is learned.

Furthermore, the structure of residential architectural programs varies significantly with the number of floors. Singe-story cottages, for example, have very different programs from three-floor mansions. While it is possible to train a single network to encode this variability using hidden variables, the amount of required training data is prohibitive. For this reason, we train separate networks for single-story, two-story, and three-story residences. The size of the training set for each network is 40, 50, and 30 real-world architectural programs, respectively.

Figure 4 (top) illustrates the training of each network. Percent of final log-likelihood score is plotted as a function of structure learning iterations. Figure 4 (bottom) shows corresponding training times. Performance was measured on an Intel Pentium D clocked at 2.8 GHz. Our implementation uses the open-source Probabilistic Networks Library [Eruhimov et al. 2003]. Figure 5 visualizes architectural programs generated by a network being trained on the corpus of single-story residences. As the training progresses, the network learns the structure of the data and generates more realistic architectural programs.

5 Floor Plan Optimization

Once an architectural program is generated, it is turned into a building layout: a detailed floor plan for each floor. These floor plans must realize the program and feature well-formed internal and external shapes. We compute these floor plans by optimizing over the space of possible building layouts. Different floors are optimized together to ensure mutual coherence.

A space of floor plans is typically parameterized by the horizontal and vertical coordinates of the rectilinear segments that form the shape of the plan [Schwarz et al. 1994; Sarrafzadeh and Lee 1993]. Since the number of segments is not constant across floor plans that conform to a given architectural program, the space we want to optimize over has varying dimensionality. Thus global optimization algorithms like Covariance Matrix Adaptation – which have recently been applied to a number of highly multimodal optimization problems in computer graphics [Wampler and Popović 2009] – cannot be used. We have successfully experimented with Reversible jump Markov chain Monte Carlo [Green 1995] for optimizing over the space of layouts. However, the detailed balance condition and the associated dimension matching functions complicate both the implementation and the exposition. In practice, we have found the simple Metropolis algorithm [Press et al. 2007], which has been widely used for the related problem of VLSI layout, to be sufficiently effective. Unlike greedy techniques, the Metropolis algorithm can accept moves that increase the cost function, in order to escape from local modes.

Specifically, define a Boltzmann-like objective function

$$f(\mathbf{x}) = \exp(-\beta C(\mathbf{x})),$$

where $\mathbf{x}$ is a given building layout, C is the cost function defined in Section 5.2, and β is a constant. At each iteration of the algorithm, a new building layout $\mathbf{x}^\star$ is proposed and is accepted with probability

$$\begin{aligned} \alpha(\mathbf{x}^\star|\mathbf{x}) &= \min\left(1, \frac{f(\mathbf{x}^\star)}{f(\mathbf{x})}\right) \\ &= \min\left(1, \exp\left(\beta(C(\mathbf{x}) - C(\mathbf{x}^\star))\right)\right). \end{aligned}$$

The optimization begins from a canonical high-cost configuration, in which equally-sized rectangular rooms are packed on a grid on each floor. Although we have found the algorithm to perform well with a set value of β, annealing β over time facilitates a more rapid exploration of the space [White 1984].

The floor plan optimization process is visualized in Figure 6. The illustrated optimization took 35 seconds on an Intel Core i7 clocked at 3.2GHz. For the purpose of illustration, the floor plans in Figures 6 and 8 were augmented with windows and other architectural elements, as described in Section 6.

5.1 Proposal Moves

In order to effectively explore the space of layouts, the proposal moves $\mathbf{x} \rightarrow \mathbf{x}^\star$ must contain both local adjustments that tune the existing layout and global reconfigurations that significantly alter the layout. We have found the following simple set of moves to be particularly effective for this purpose. It is easy to show that any

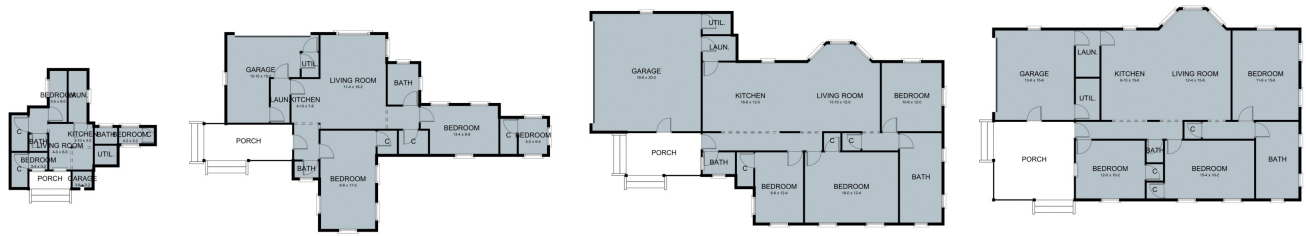


Figure 6: *Floor plan optimization. As the optimization proceeds, interior and exterior shapes become more cohesive and the architectural program is realized.*

building layout can be reached from any other layout using these moves, as long as the layouts have the same number of rooms. Our notation is as follows. A floor plan $\mathbf{x}$ is defined as a set of n rooms. Each room is a rectilinear polygon enclosed by *wall segments*. A maximal contiguous set of collinear wall segments is said to form a *wall*.

Sliding a wall. The first proposal move locally adjusts the floor plan by sliding a contiguous set of wall segments forward or backward. We randomly choose a wall and a *split point* along the wall. The wall is split into two collinear walls. One remains in place while the other is moved forward or backward a distance $d \sim \mathcal{N}(0, \sigma^2)$. (Figure 7, left.) The split point has a strictly positive probability of being one of the end-points of the wall, in which case the entire wall is moved without splitting. The splitting also prioritizes existing vertices, although there is a strictly positive probability of splitting an existing wall segment. Such splits are essential since they introduce concave corners. While architects prioritize convex spaces, concavities are necessary for realizing many architectural programs and are often found in real-world floor plans [Wood 2007].

A sequence of split moves increases the overall number of walls and can lead to highly irregular room shapes and building outlines. To balance this, we snap a moved wall to an existing one if, after a split move, the two walls lie within distance ε. (Figure 7, right.) We empirically set $\varepsilon = 1$ ft. and $\sigma = 2\varepsilon$. We similarly align the wall with walls on neighboring floors, to increase consistency across floors.

Swapping rooms. In addition to local adjustments, a successful application of the Metropolis algorithm requires proposal moves that significantly alter the layout, in order to more rapidly explore the space of layouts. We adopt the natural strategy of swapping the identities of two rooms in the existing layout. Two rooms are selected at random and their labels are interchanged. This generally results in a drastic change in the objective function value, triggering considerable modifications in the layout over the ensuing sequence of moves.

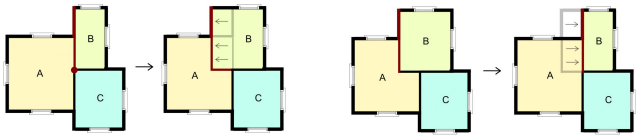


Figure 7: *A sliding move locally adjusts the layout.*

5.2 Cost Function

The optimization procedure attempts to minimize a cost function $C(\mathbf{x})$ that evaluates the quality of the layout. The cost function penalizes violations of the architectural program, ill-formed rooms and exterior outlines, and incompatibilities between floors. Specifically, the cost function is defined as

$$C(\mathbf{x}) = k_a C_a(\mathbf{x}) + k_d C_d(\mathbf{x}) + k_f C_f(\mathbf{x}) + k_s C_s(\mathbf{x}),$$

where k_a, k_d, k_f, and k_s are constants that specify the relative importance of each term. These terms are defined below. The effect of each term is demonstrated in Figure 8.

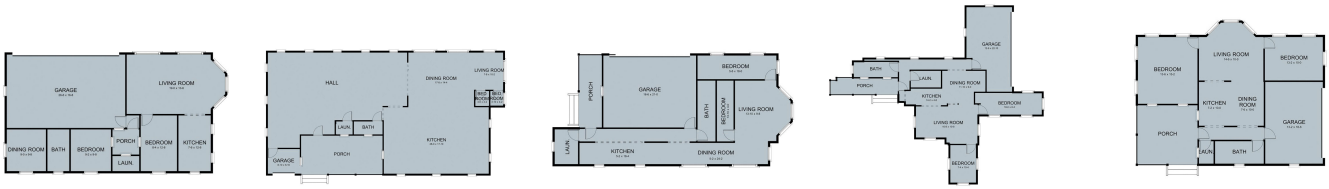
Accessibility. The architectural program specifies which pairs of rooms must be connected by a door or an open passage. These connections represent the circulation throughout the building and determine the privacy gradient. For a connection between two rooms to be possible, they must share a sufficiently long wall segment to accommodate a door. Furthermore, the entrance, patios, and garage must have access to the outside. We define the accessibility cost C_a as the number of missing connections plus the number of enclosed patios, entrances, and garages.

Dimensions. The architectural program also specifies the desired dimensions – in the form of area and aspect ratio – for each room. While it is possible to sample a single value for each room's area and aspect ratio from the Bayesian network, we have found it advantageous to retain the complete probability distribution produced for each of these values by the network. Thus, for each room, we have distributions over possible areas and aspect ratios. These distributions are used to penalize unlikely room dimensions. Specifically, let $\ell^i_{\text{a}}(\mathbf{x})$ be the log-likelihood of the observed area of room i in layout $\mathbf{x}$ and let $\ell^i_{\text{as}}(\mathbf{x})$ be the log-likelihood of the corresponding aspect ratio. The dimension cost is defined as

$$C_d(\mathbf{x}) = -\sum_{i=1}^{n} \left(\ell^i_{\text{a}}(\mathbf{x}) + \ell^i_{\text{as}}(\mathbf{x})\right).$$

Floors. Each floor must be contained within the envelope of the floor beneath [Wertheimer 2009]. Let $F^i_{\mathbf{x}}$ be the union of all rooms on the ith floor. Then $F^i_{\mathbf{x}} - F^{i-1}_{\mathbf{x}}$ is the unsupported region on the ith floor. The bottom floor $F^1_{\mathbf{x}}$ does not need to be supported, but it must fit within the building lot, denoted by $F^0_{\mathbf{x}}$. This enables the generation of buildings that fit in a pre-specified building lot. The floor compatibility cost $C_f(\mathbf{x})$ is defined as

$$C_f(\mathbf{x}) = \frac{\sum_{i=1}^{l} A(F^i_{\mathbf{x}} - F^{i-1}_{\mathbf{x}})}{\sum_{i=1}^{l} A(F^i_{\mathbf{x}})},$$



(a) accessibility term excluded (b) area term excluded (c) aspect ratio term excluded (d) shape term excluded (e) all terms included

Figure 8: *The cost function. (a,b,c,d) Ablation of individual cost terms and its effect on the floor plan optimization. (e) A floor plan optimized with the complete cost function.*

where l is the number of floors and $A(\cdot)$ is the area operator.

Shapes. Architects generally consider near-convex room shapes to be preferable to strongly concave ones [Alexander et al. 1977]. Near-convex rooms are viewed as more comfortable, while rooms that feature deep recesses often do not feel like cohesive spaces. We thus penalize large deviations from convexity with a shape cost $C_s(\mathbf{x})$. Specifically, given a shape S, we define a measure $M_c(S)$ as

$$M_c(S) = \frac{A(H(S)) - A(S)}{A(S)} + e(S),$$

where $H(S)$ is the convex hull of S and $e(S)$ is the number of edges along the outline of S. $e(S)$ functions as a regularization term that penalizes complexity even in near-convex shapes.

Hallways and stairways are often shaped differently from other rooms because they are meant to be traveled through, not lived in. They are not required to have the comforting qualities of inhabited rooms and should not be penalized for concavity. We define a hallway indicator h_i to be 1 if room i is a hallway or stairway and 0 otherwise. The overall shape cost for individual rooms is defined to be $\sum_{i=1}^{n}(1-h_i)M_c(R^i_{\mathbf{x}})$, where $R^i_{\mathbf{x}}$ is the ith room in the layout.

There are also groups of rooms whose aggregate shape is important. If a group of rooms is connected by open walls, it can be experienced as a single interior space. Let $G^i_{\mathbf{x}}$ be such a group, for i ranging from 1 to g, where g is the number of such groups in the layout. The total shape cost is defined as

$$\begin{aligned} C_s(\mathbf{x}) &= k_r \textstyle\sum_{i=1}^{n}(1-h_i)M_c(R^i_{\mathbf{x}}) \\ &+ k_g \textstyle\sum_{i=1}^{g} M_c(G^i_{\mathbf{x}}) \\ &+ k_o \textstyle\sum_{i=1}^{l} e(F^i_{\mathbf{x}}), \end{aligned}$$

where k_r, k_g, and k_o are constants. The last term penalizes irregularity in the outline of each floor.

6 Generating 3D Models

Given a building layout, we can generate corresponding three-dimensional models, decorated in a variety of styles (Figure 9). Each available style is specified in a style template, listing the geometric and material properties of every building element: windows, doors, wall segments, gables, stairs, roofs, and patio poles and banisters. The style template also records the desired spacings of windows and banister poles, the roof angle, the presence of gables, and the length of roof overhangs. New styles can be introduced by adding new style templates.

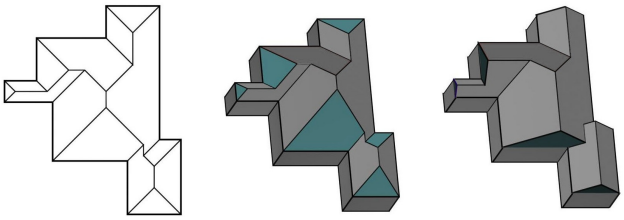
Passageways. Different types of passageways are used between different types of rooms. Ordinary doors are placed in private rooms such as bedrooms. French doors with windows are placed in semi-private rooms such as studies or patios. Wide passageways are placed between public rooms. Adjacent public rooms may be on different floors, such as a second-story hallway that overlooks a two-story living room; wherever a change in elevation occurs between adjacent rooms, a banister is placed instead of a passageway.

It is customary to place passageways near room corners. ("When [a door] is in the middle of the wall, it almost always creates a pattern of movement which breaks the room in two, destroys the center, and leaves no single area which is large enough to use" [Alexander et al. 1977].) We place the door in a location that minimizes the sum of the distances from the door to the nearest corner in each of the adjacent rooms.

Windows. Windows are laid out along exterior wall segments at regular intervals within each room. Different sizes and styles of windows can be used for different types of rooms. Available window types, along with their sizes and spacings, are listed in the style template. If a window is obstructed by a roof segment from a lower floor, the system defaults to a smaller window type, if available. If the obstruction is too great for any available window type, no window is placed. In the entrance room, the central window is replaced by the front door. Window boxes are handled similarly.

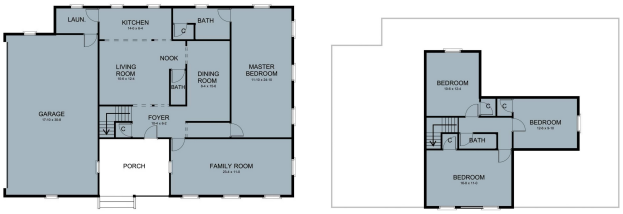
Staircases. Staircase spaces are specified in the building layout. The shape of a staircase determines how the steps are arranged. If the staircase space is narrow and rectangular, the steps are laid out linearly. If the staircase space is wide enough for two flights of stairs, a U-shaped staircase is used. If the staircase space is L-shaped, a 90°-turn is placed. Steps are made at least 36 inches

Figure 9: *3D models generated for the same building layout. From left to right: Cottage, Italianate, Tudor, and Craftsman.*



(a) Straight skeleton (b) Hipped roof (c) Gabled roof

Figure 10: *Roof construction. A straight skeleton is computed (a), yielding a hipped roof (b). If gables are desired, appropriate roof faces are projected onto the façade plane (c).*



first floor second floor

Figure 11: *Failure case. The accessibility term was not driven to zero before the optimization procedure terminated. As a result, the staircase is blocked on the second story.*

wide, with each step having a tread between 9 and 11 in. and a riser at most 8-1/4 in. [The American Institute of Architects 2007].

Roofs. Roofs are constructed using the straight skeleton algorithm [Aichholzer et al. 1995] (Figure 10(a)). This yields a hipped roof with no gables (Figure 10(b)). A gable is a triangular shape below the roof that becomes part of the façade. Gables are added if their existence is specified in the style template. For a given face of a hipped roof, its vertices are simply projected onto the façade plane to create a gable. If a roof face contains horizontal top edges (parallel to the ground plane), it is not converted into a gable, since this would distort other roof faces (Figure 10(c)).

7 Results

We first evaluate the individual components of the presented approach. Figures 4 and 5 demonstrate the performance of data-driven architectural programming, described in Section 4. Figure 12 focuses on the floor plan optimization. Figure 12 (left) shows a real-world building layout designed by a human architect, reproduced from a catalogue [Wood 2007]. To isolate the floor plan optimization step, we encoded the architectural program for this layout and used it as input to the algorithm described in Section 5; the result is shown in Figure 12 (right). Figure 9 demonstrates automatic generation of 3D models in different decorative styles for the same building layout (Section 6).

Figure 13 shows additional building layouts generated by our method. Real-world layouts designed by human architects [Wood 2007] are shown for comparison.

While the optimization procedure described in Section 5 can theoretically be made to converge to the global optimum [Geman and Geman 1984], it is not guaranteed in practice to yield a layout that minimizes all cost terms. Figure 11 presents a layout where the optimization terminated before the accessibility term was driven to zero. As a result, the staircase is blocked on the second story. Fortunately, such failures are easy to detect automatically. In our experiments they are virtually absent for single-story buildings, arise in roughly 1 out of 20 optimization instances for two-story buildings, and in 1 out of 5 three-story optimizations. The chief cause for these unsuccessful optimizations are multi-story spaces, such as staircases and two-story living rooms, which couple the optimization on multiple floors.

Figure 14 uses exploded views to visualize 3D models created with the presented approach [Niederauer et al. 2003]. While automated furniture placement techniques have been developed [Germer and Schwarz 2009], we placed furniture manually to illustrate the functions of the rooms. The three-story residences in our training set were designed for hilly neighborhoods, with a main entrance on the ground level, a lower level that is also accessible from the outside, and an upper level. The model shown in Figure 14(c) is thus situated on a hill by design. Figure 15 shows a collection of automatically generated residences with a variety of layout complexities and architectural styles. Figure 16 shows a computer-generated two-story Tudor, rendered in night-time conditions. All models shown in this paper took a few seconds to 7 minutes to generate by our single-threaded implementation.

8 Discussion

This paper presents an end-to-end approach to automated generation of residential building layouts. The method combines machine learning and optimization techniques with architectural methodol-

real-world computer-generated

Figure 12: *A two-story layout designed by an architect (left) and an automatically generated layout for the same architectural program (right).*

Figure 13: *(a,b,d,e) Computer-generated building layouts. (c,f) Real-world layouts, reproduced from a catalogue, shown for comparison. The bottom row shows two-story layouts.*

ogy to produce visually plausible building layouts from high-level requirements.

Our building layout procedure is idealized and does not take into account the myriad of site-specific and client-specific factors that are considered by architects. In real-world architectural practice, layout design is affected by the local climate, the views from the site, and other environmental considerations. The client's personality also plays a role: if the client has a large collection of paintings, a good architect will make sure that sufficient wall space is available. Such considerations can be integrated into the presented approach by augmenting the objective function in Section 5.2. Another direction for future research is to apply the techniques developed in this work to other building types, such as apartment complexes and office buildings. We believe that globally coupled optimization of floor plans for all stories is not necessary in these cases, due to significant regularity between floors.

Another important direction is to enable interactive exploration of building layout designs. We believe that techniques presented in this work can lead to powerful tools for interactive creation of building layouts. An additional research avenue is to apply data-driven techniques to related layout problems, such as furniture placement and the layout of outdoor environments.

There are many other ways to extend the presented approach. One would be to add non-rectilinear or even curved wall segments, as well as variation in floor elevation and ceiling height [Susanka 2001]. A more explicit optimization of the building exterior could enable the synthesis of buildings with cohesive interiors as well as specific façade appearance [Müller et al. 2006]. Integration of structural stability considerations is another interesting possibility [Whiting et al. 2009]. These developments will further enhance the fidelity of computer-generated buildings.

Acknowledgments

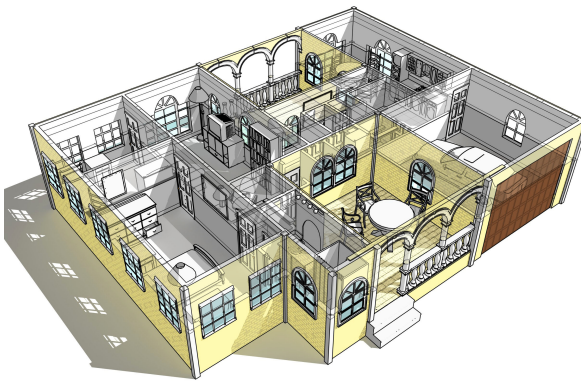
We are grateful to David Salesin, Marc Levoy, and Daniel Cohen-Or for their comments on drafts of this paper. John Haymaker and Carlo Séquin provided helpful references at early stages of the research. Lingfeng Yang, Jerry Talton, and Jared Duke assisted the project at various points. Katherine Breeden narrated the video. Peter Baltay and others at Topos Architects, David Solnick and Danielle Wyss at Solnick Architecture, and Jim McFall at McFall Architecture generously opened their practices to us and patiently answered our questions. This work was supported in part by NSF grants SES-0835601 and CCF-0641402.

References

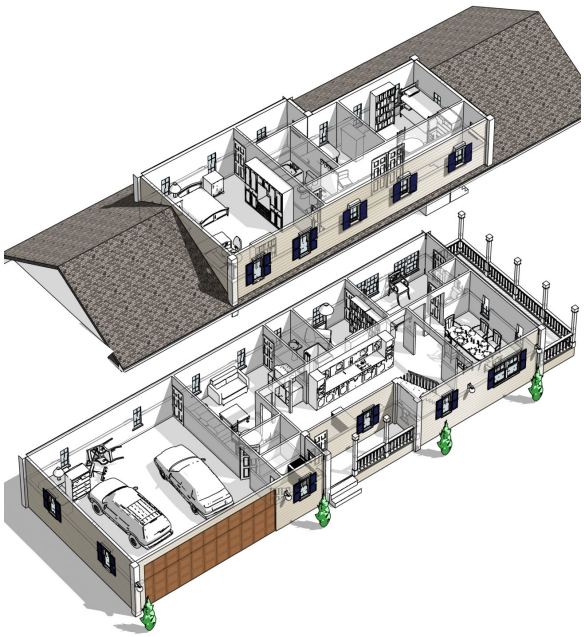
AICHHOLZER, O., AURENHAMMER, F., ALBERTS, D., AND GÄRTNER, B. 1995. A novel type of skeleton for polygons. *Journal of Universal Computer Science 1*, 12, 752–761.

ALEXANDER, C., ISHIKAWA, S., AND SILVERSTEIN, M. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press.

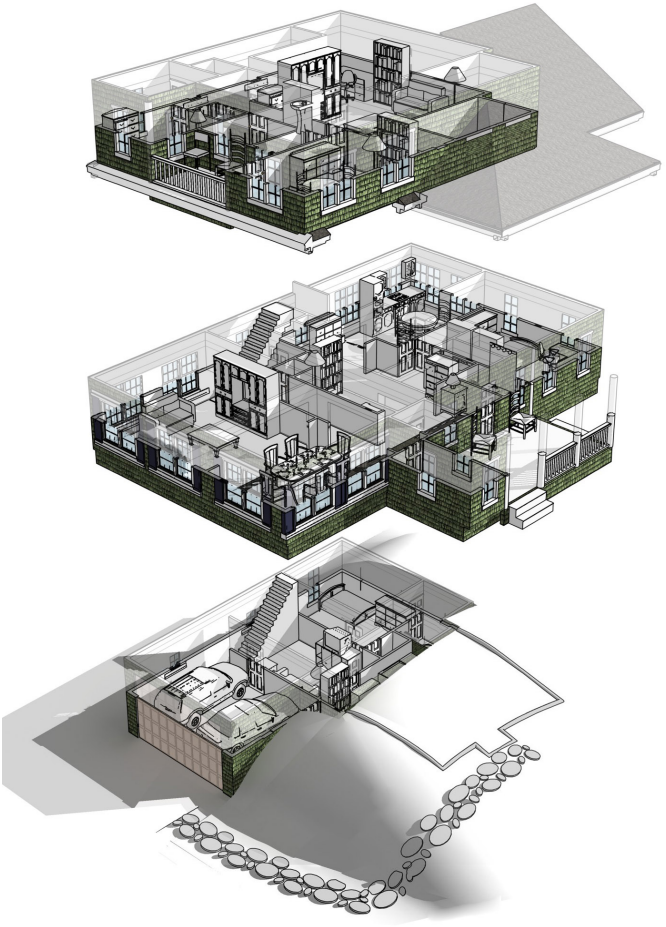
ARVIN, S. A., AND HOUSE, D. H. 2002. Modeling architectural design objectives in physically based space planning. *Automation in Construction 11*, 2, 213–225.



1-bed 2-bath, Italianate style



3-bed 3-bath, Cottage style



5-bed 5-bath, Craftsman style, on a hill

Figure 14: *Exploded views of 3D models produced by our method. Furniture placed manually for illustration.*

BOYD, S., AND VANDENBERGHE, L. 2004. *Convex Optimization*. Cambridge University Press.

CHEN, X., KANG, S. B., XU, Y.-Q., DORSEY, J., AND SHUM, H.-Y. 2008. Sketching reality: realistic interpretation of architectural designs. *ACM Transactions on Graphics 27*, 2.

ERUHIMOV, V., MURPHY, K., AND BRADSKI, G., 2003. Intel's open-source probabilistic networks library.

GALLE, P. 1981. An algorithm for exhaustive generation of building floor plans. *Communications of the ACM 24*, 12, 813–825.

GEMAN, S., AND GEMAN, D. 1984. Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence 6*, 721–741.

GERMER, T., AND SCHWARZ, M. 2009. Procedural arrangement of furniture for real-time walkthroughs. *Computer Graphics Forum 28*, 8, 2068–2078.

GREEN, P. J. 1995. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika 82*, 711–732.

HAHN, E., BOSE, P., AND WHITEHEAD, A. 2006. Persistent real-time building interior generation. In *Sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, ACM.

HARADA, M., WITKIN, A., AND BARAFF, D. 1995. Interactive physically-based manipulation of discrete/continuous models. In *Proc. SIGGRAPH*, ACM.

HECKERMAN, D. 1999. A tutorial on learning with Bayesian networks. In *Learning in Graphical Models*, M. I. Jordan, Ed. MIT Press.

HILLIER, B., AND HANSON, J. 1989. *The Social Logic of Space*. Cambridge University Press.

JACOBSON, M., SILVERSTEIN, M., AND WINSLOW, B. 2005. *Patterns of Home*. The Taunton Press.

Figure 15: *Computer-generated residences. Each building layout is unique.*

KALAY, Y. E. 2004. *Architecture's New Media: Principles, Theories, and Methods of Computer-Aided Design*. MIT Press.

KOLLER, D., AND FRIEDMAN, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

LAI, T.-T., AND LEINWAND, S. M. 1988. Algorithms for floorplan design via rectangular dualization. *IEEE Transactions on Computer-Aided Design 7*, 12, 1278–1289.

LAURITZEN, S. L., AND SPIEGELHALTER, D. J. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B 50*, 2, 157–224.

LEFEBVRE, S., HORNUS, S., AND LASRAM, A. 2010. By-example synthesis of architectural textures. *ACM Transactions on Graphics 29*, 4.

LEGAKIS, J., DORSEY, J., AND GORTLER, S. 2001. Feature-based cellular texturing for architectural models. In *Proc. SIGGRAPH*, ACM.

LIGGETT, R. S. 2000. Automated facilities layout: past, present and future. *Automation in Construction 9*, 197–215.

MARCH, L., AND STEADMAN, P. 1971. Spatial allocation procedures. In *The Geometry of Environment*. MIT Press, ch. 13, 303–317.

MARTIN, J., 2005. Algorithmic beauty of buildings methods for procedural building generation. Computer Science Honors Thesis, Trinity University.

MARTIN, J. 2006. Procedural house generation: a method for dynamically generating floor plans. In *Poster session, Symposium on Interactive 3D Graphics and Games*.

MICHALEK, J. J., CHOUDHARY, R., AND PAPALAMBROS, P. Y. 2002. Architectural layout design optimization. *Engineering Optimization 34*, 5, 461–484.

MITCHELL, W. J. 1990. *The Logic of Architecture: Design, Computation, and Cognition*. MIT Press.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND VAN GOOL, L. 2006. Procedural modeling of buildings. In *Proc. SIGGRAPH*, ACM.

MÜLLER, P., ZENG, G., WONKA, P., AND VAN GOOL, L. 2007. Image-based procedural modeling of facades. In *Proc. SIGGRAPH*, ACM.

NIEDERAUER, C., HOUSTON, M., AGRAWALA, M., AND HUMPHREYS, G. 2003. Non-invasive interactive visualization of dynamic architectural environments. In *Proc. Symposium on Interactive 3D graphics*, ACM, 55–58.

POTTMANN, H., LIU, Y., WALLNER, J., BOBENKO, A., AND WANG, W. 2007. Geometry of multi-layer freeform structures for architecture. In *Proc. SIGGRAPH*, ACM.

POTTMANN, H., SCHIFTNER, A., BO, P., SCHMIEDHOFER, H., WANG, W., BALDASSINI, N., AND WALLNER, J. 2008. Freeform surfaces from single curved panels. In *Proc. SIGGRAPH*, ACM.

PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., AND VETTERLING, W. T. 2007. *Numerical recipes: the art of scientific computing*, 3rd ed. Cambridge University Press.

SARRAFZADEH, M., AND LEE, D. T. 1993. *Algorithmic Aspects of VLSI Layout*. World Scientific.

SCHWARZ, A., BERRY, D. M., AND SHAVIV, E. 1994. Representing and solving the automated building design problem. *Computer-Aided Design 26*, 9, 689–698.

SÉQUIN, C. H., AND KALAY, Y. E. 1998. A suite of prototype CAD tools to support early phases of architectural design. *Automation in Construction 7*, 449–464.

SHAVIV, E., AND GALI, D. 1974. A model for space allocation in complex buildings. *Build International 7*, 6, 493–518.

SHAVIV, E. 1987. Generative and evaluative CAAD tools for spatial allocation problems. In *Computability of Design*, Y. E. Kalay, Ed. John Wiley & Sons, ch. 10, 191–212.

Figure 16: *Computer-generated house at night.*

STINY, G. 2006. *Shape: Talking about Seeing and Doing*. MIT Press.

SUSANKA, S. 2001. *The Not So Big House: A Blueprint for the Way We Really Live*. The Taunton Press.

THE AMERICAN INSTITUTE OF ARCHITECTS. 2007. *Architectural Graphic Standards, 11th ed.* John Wiley & Sons, Inc.

WAMPLER, K., AND POPOVIĆ, Z. 2009. Optimal gait and form for animal locomotion. In *Proc. SIGGRAPH*, ACM.

WERTHEIMER, L. 2009. *Schematic Design*. Kaplan Architecture Education.

WHITE, S. R. 1984. Concepts of scale in simulated annealing. *AIP Conference Proceedings 122*, 1, 261–270.

WHITING, E., OCHSENDORF, J., AND DURAND, F. 2009. Procedural modeling of structurally-sound masonry buildings. In *Proc. SIGGRAPH Asia*, ACM.

WOOD, H. 2007. *Essential House Plan Collection: 1500 Best Selling Home Plans*. Home Planners.

YIN, X., WONKA, P., AND RAZDAN, A. 2009. Generating 3d building models from architectural drawings: a survey. *IEEE Computer Graphics and Applications 29*, 1, 20–30.