Learning Shape Placements by Example

Paul Guerrero*
KAUST, Vienna University of Technology

Stefan Jeschke†
IST Austria

Michael Wimmer‡
Vienna University of Technology

Peter Wonka§
KAUST

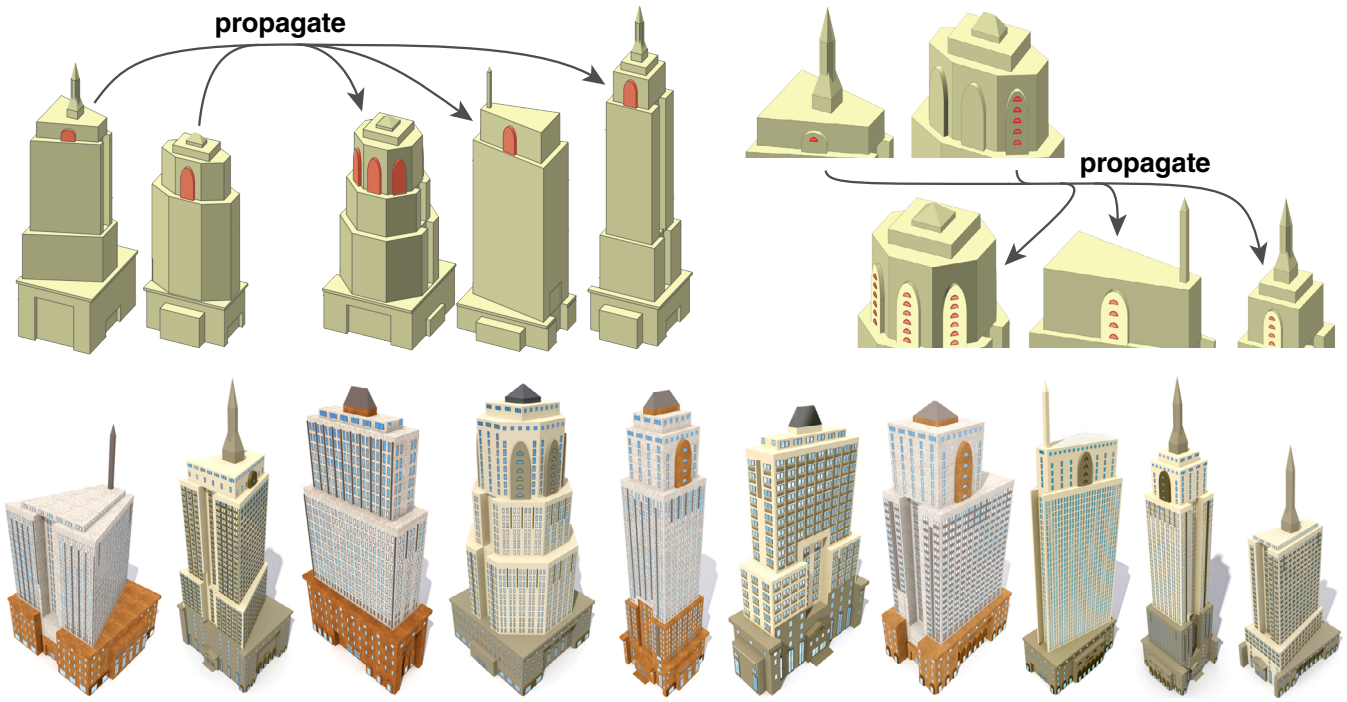


Figure 1: *New York-style skyscrapers generated by our method: Individual shape placement operations are learned during an example modeling session. Two of these operations are shown in the top row. New skyscraper variations like the ones in the bottom row can be generated from the learned model without additional user input.*

Abstract

We present a method to learn and propagate *shape placements* in 2D polygonal scenes from a few examples provided by a user. The placement of a shape is modeled as an oriented bounding box. Simple geometric relationships between this bounding box and nearby scene polygons define a *feature set* for the placement. The feature sets of all example placements are then used to learn a probabilistic model over all possible placements and scenes. With this model, we can generate a new set of placements with similar geometric relationships in any given scene. We introduce extensions that enable propagation and generation of shapes in 3D scenes, as well as the application of a learned modeling *session* to large scenes without additional user interaction. These concepts allow us to generate complex scenes with thousands of objects with relatively little user interaction.

*paul@cg.tuwien.ac.at
†sjeschke@ist.ac.at
‡wimmer@cg.tuwien.ac.at
§pwonka@gmail.com

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems;

Keywords: modeling by example, complex model generation

1 Introduction

Recently, the amount of detail that is expected in polygonal scenes in computer graphics has increased significantly, suggesting the need for powerful tools for model creation and manipulation. Existing systems mostly employ *rule-based methods* or formulate *optimization problems*, where an expert user encodes domain-specific knowledge as a large set of rules, constraints and/or objective functions. A disadvantage of these tools is that they are very different from the commercial modeling software that most modelers are proficient with. To resolve this discrepancy, *example-based modeling methods* have recently gained interest, since they do not require a complex pre-encoding of domain knowledge, but mostly work with familiar modeling operations. One typical option is to provide one or several complete scenes as examples and to produce "similar" scenes as output. Unfortunately, the degrees of freedom grow exponentially with example scene complexity—a typical case of the "curse of dimensionality". This means that impractically many example scenes are required to disambiguate object relations, which makes user control fairly limited.

The key problem to tackle is an efficient way to encode what valid

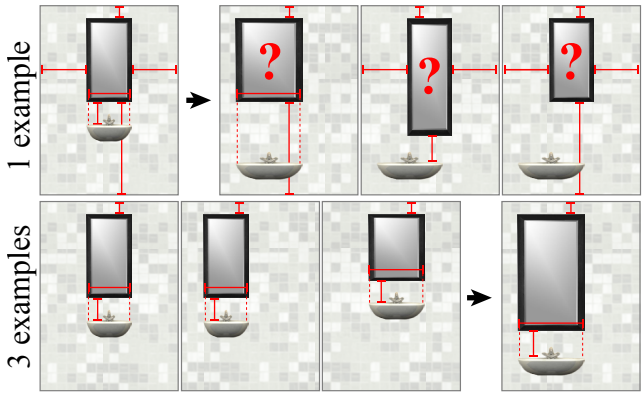


Figure 2: *A single example of a mirror placement leaves room for interpretation of the user's modeling intention, as is illustrated in the top row. The red lines indicate the matched relations. Should the mirror match the width of the sink? Should its vertical position match the sink or the wall? Each interpretation results in a different placement. Providing multiple examples like the ones in the bottom row disambiguates the user's intention.*

and desirable object relations are, as intended by a user. To reach this goal, we believe that it is more promising to provide examples not as complete scenes but at the level of individual editing operations. In this paper, we propose such a method. Specifically, a model that captures the placement of a shape is learned from a few given examples, illustrated in Figure 1. Our input is a two-dimensional scene, composed of a set of simple polygons with attached *labels* that describe scene semantics like "door" or "window". A user can now define any number of locations, orientations, widths and heights of a new shape, called *example placements*. By analyzing the shape's neighborhood, a model of the intended placement is learned without requiring further user input. With this model, we can then automatically identify any number of qualitatively "similar" placements in any given scene such that the modeling intent of the user is met. In the case of city modeling, such a user's intent may be *"20 cm below each window"* on a building façade, or something less precise like *"close to a house but not close to a street"* in a city layout. In such contexts, a single example placement is often too ambiguous to clearly reveal the user's intention, as illustrated in Figure 2. Therefore, one main point of this work is to analyze *multiple* input placements of a shape to resolve these ambiguities and thus efficiently narrow down the user's particular intent. Furthermore, we support varying shape rotation and per-axis scaling, which is not the case in existing frameworks. Such a general approach is novel to the best of our knowledge.

Technically, our model of the user's supposed intent is built on a kernel regression method for which we introduce a new distance metric between shape placements. It is specifically designed to work for different numbers and types of scene elements in the example shape's neighborhood. We prove that the kernel based on this distance metric fulfills the validity requirements for kernel regression. We also show how the 2D placement method can be extended to 3D scenes using extrusion and Boolean operators. The application of our method in different domains is presented, including placing elements in city layouts, on building façades, and on interior floorplans, demonstrating the capabilities of our method where previous methods fail. By recording edit operations, introducing random variations and applying them to larger scenes, our example-based shape-placement method can be used to generate both complex and unique models in large quantities.

2 Related Work

Most closely related to the work described here are analysis-and-edit approaches and scene modeling-by-example methods. Similar to this paper, these existing approaches exploit relationships between objects or object parts to help designers populate or modify complex models or scenes in a meaningful way.

One line of research concentrates on analyzing and editing individual objects [Gal et al. 2009; Zheng et al. 2011; Bokeloh et al. 2012]. The general idea is to derive geometric relationships between different object parts, such as one-dimensional feature lines [Gal et al. 2009] or other meaningful components [Zheng et al. 2011; Bokeloh et al. 2012]. Local edit operations are then propagated to similar object parts using these relationships. If a model database with numerous examples within an object class is available, new models can be synthesized as combinations of object parts found in this database [Funkhouser et al. 2004; Chaudhuri et al. 2011; Kalogerakis et al. 2012]. Here the challenge is to automatically identify model parts and their mutual relations, which are learned from the database [Chaudhuri et al. 2011; Kalogerakis et al. 2012]. Conceptually, the aforementioned body of work aims at finding sets of existing model parts that can be modified and recombined to generate reasonable new variants of objects. The model parts are placed either manually or using predefined placement points. In contrast, in this work, we learn *where* to place new shapes. Additionally, we can learn shape placements anywhere in a scene; the shapes need not be connected to each other.

Instead of studying the structure of object parts, modeling-by-example approaches focus on complex, three-dimensional scenes (mostly indoors) [Xu et al. 2002] by studying individual objects and their spatial, structural, semantic, and functional relationships. Related questions include how to efficiently retrieve relevant 3D objects from a database such that they fit in a given scene context with high probability [Fisher and Hanrahan 2010; Fisher et al. 2011]. New scenes can be synthesized after learning object relationships from a database of example scenes [Fisher et al. 2012; Yu et al. 2011; Yeh et al. 2012]. The proposed mathematical models include probabilistic models based on Bayesian networks and Gaussian mixtures [Fisher et al. 2012] and specialized probability distributions together with constraints encoded as factor graphs [Yeh et al. 2012]. The space spanned by these models is then appropriately sampled to derive plausible scenes. Alternatively, a cost function can be formulated whose optimization yields realistic furniture arrangements [Yu et al. 2011]. For some indoor scenes, synthesis can be based on special design guidelines [Merrell et al. 2011] that include functional and visual criteria apart from purely geometric relationships. While these methods have a broader scope than our work (e.g., they also need to pick the right type and number of objects), their placement models are limited compared with our approach. In Section 5, we compare our approach to one of the recent methods [Fisher et al. 2012]. Additionally, our approach to provide examples at the level of individual operations has some advantages compared with using complete scenes as examples. First, there is more user control: the modeler can decide which operations to apply next based on feedback from the previous operation. Second, each individual operation has far fewer degrees of freedom, which means that fewer examples are necessary to communicate the user's modeling intent clearly. Finally, the modeling session includes information that is not available in the final model, e.g., which individual operations were used to create the model? More information to work with means better results; that is, a better estimation of the user's intent.

Recently, Guerrero et al. [2014] proposed an example-based editing system that propagates an example object with a particular orientation (called a *pose*) to locations in a two-dimensional polygonal

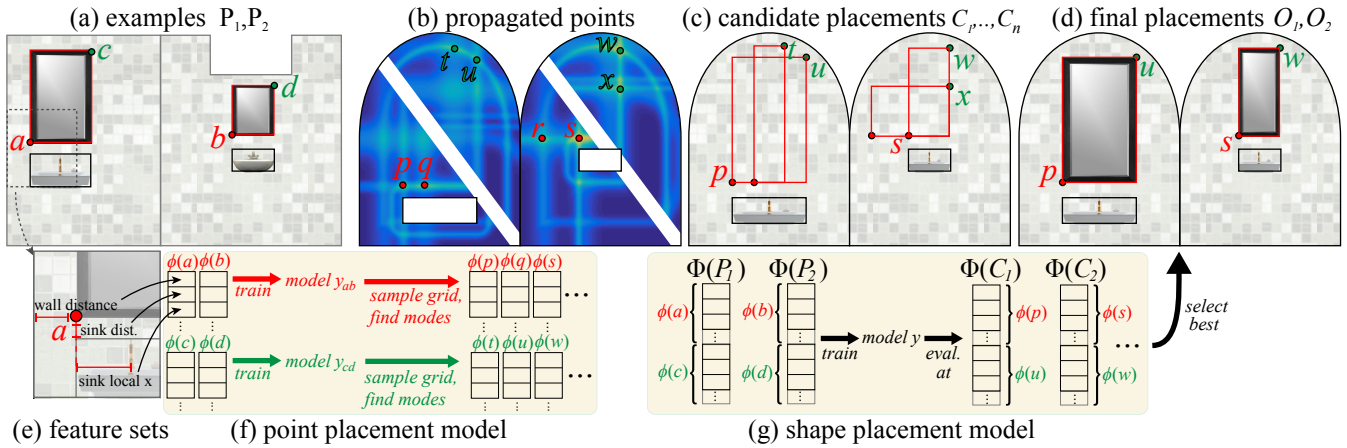


Figure 3: *Main steps of the proposed method: Given a set of example placements (outlined in red in (a)), our method sparsely samples the bounding box of each placement with two sample points (red and green dots; in our implementation we use 5 sample points). For each sample point, relationships to surrounding scene elements as shown in (e) are combined into a feature set. Based on this feature set, we train a statistical point-placement model (f) and extract promising sample-point positions on a dense grid, shown as heat map in (b). In this sub figure, the lower half of the heat map corresponds to the lower-left sample point, and the upper half to the upper-right sample point. The candidate placements in (c) are constructed from combinations of the extracted heat-map maxima and evaluated with a combined statistical shape-placement model in (g). Finally, the best candidate placements are selected as the final propagated shapes, shown in (d).*

scene that exhibit a similar set of geometric relations to the surrounding geometry. In that system, a user can provide only a *single example*, which is often not enough to uniquely specify the desired placement properties. As is illustrated in Figure 2, a single example is ambiguous when transferring a shape from a large rectangle to a small one: should the shape scale with the rectangle or retain its size? Should it retain the same distance to an edge of the rectangle or to its center? Specifying multiple examples in rectangles of different sizes disambiguates the placement. Furthermore, representing shapes by a *single point* limits the class of relations that can be captured. For example, a user may want to place a balcony on a facade above a door and in front of two windows. In this placement, different parts of the balcony are constrained by different relations: the bottom part of the balcony needs to be close to a door, while the sides need to be close to two different windows. Placements of this type cannot be captured when representing shapes with a single point. Finally, Guerrero et al. only considered strictly 2D layout problems.

In this paper, we use multiple placements to solve ambiguous cases that are inherent in any placement process. More importantly, we do not just spatially position objects; we adjust them in width, height and orientation to automatically fit a learned local neighborhood, which is not possible with existing approaches to the best of our knowledge. Finally, because our mathematical model is based on the more general concept of object relationship *functions*, it generally provides higher synthesis quality than previous work based on binary relations.

3 Placement Learning and Propagation

3.1 Overview

The input of our method is one or multiple *example scenes* $S_1, \ldots, S_n$, each composed of simple 2D polygons. All polygons have labels describing semantics, like "door" or "window". For a polygonal input shape to be propagated, the user provides a set of *example placements* $P_1, \ldots, P_n$, where a placement P is given as an oriented bounding box defined by five parameters: the center position, width, height and orientation. Given these example shape placements and a *query scene* S, our intuitive goal is to automatically find placements $O_1, \ldots, O_m$ in S that are "similar" to the example placements. The following sections provide four conceptual building blocks that formalize this process.

Placement descriptors In Section 3.2 we introduce the notion of a placement descriptor. To capture user intent, such as *"place a flower box 20 cm below a window"* on a building façade, we describe a placement using a *feature set* $\Phi(P, S)$, which is composed of simple geometric relationships between the placement P and nearby elements of the scene S, such as the distance to a polygon boundary or the coordinates with respect to the local frame of a polygon.

Placement similarity measure Having a formal placement description, the next key concept is an adequate measure of *similarity* between any two given shape placements. The challenge here is that two placement descriptors typically contain different numbers and types of features. Additionally, the number of features is usually large, but only some of them are relevant for determining the similarity. In Section 3.3, we introduce the *SL0 kernel* $k(\Phi(P_1, S_1), \Phi(P_2, S_2))$, defined on the placement's feature sets, which is well suited to address these challenges.

Statistical placement model As noted above, one main goal of this work is to exploit the information implicitly encoded in multiple input placements in order to resolve potential placement ambiguities. For this, we define the similarity of a placement to a *set of example placements* by a kernel regression model $y(\Phi(P, S))$ over the space of feature sets, using the similarity k as a kernel. This model is trained on all example placements $\Phi(P_i, S_i)$, as described in detail in Section 3.4.

Placement propagation method Finally, given the trained model y and a query scene S, we show how to identify placements with high similarity $y(\Phi(P, S))$ to the example placements in Section 3.5. Unfortunately, an exhaustive search of all possible placements P in S is prohibitively expensive. Instead, we evaluate y

only for a set of *candidate placements* C. To find these candidates, we first decompose the high-dimensional search space into several 2D subspaces and separately find point locations with high similarity in each subspace. Candidate placements C are then formed as combinations of three point locations coming from different subspaces. We output candidates with high similarity $y(\Phi(P,S))$ as output placements O_m.

Figure 3 illustrates the concepts described above. In Section 4, we present several application-dependent filters to select output placements, as well as extensions to allow the generation of 3D scenes from example modeling sessions. Finally, we show applications like mass modeling, interior modeling and façade modeling. We also provide comparisons to existing work and detailed quantitative evaluations in Section 5.

3.2 Placement Descriptors

In this section, we describe a placement by a *feature set*, based on the geometric relationship between the placement and the surrounding scene geometry. To make the notion of such a geometric relationship tractable, our first step is to decompose the scene geometry into simple parts, called *scene elements*, as detailed below. In addition, we start our exposition by describing geometric relationships between individual points and scene elements with so-called *relationship functions*, and then extend this concept to placements. The value of a relationship function will be called *feature tuple*, and the set of tuples for all nearby elements is the *feature set* of a placement.

Scene elements Similar to Guerrero et al. [2014], all scene polygons are decomposed into "smooth" segments, which are comprised of edges with no sharp angles, and sharp corners, which connect the smooth segments. This gives us elements of type *polygon*, *segment*, and *corner*, which are more formally defined in Appendix A.1.

Feature tuples A *relationship function* ψ expresses multiple geometric relationships between a point and a scene element numerically, such as the *boundary distance* and the *corner ratio*. They are more formally defined in Appendix A.2. The function ψ maps a point and a scene element to a tuple $(v_1, v_2, ..., v_{|}\psi|) \in \mathbb{R}^{|}\psi|$ of geometric relationship values v_i, called a *feature tuple*. Each element type has a specific set of $|\psi|$ relationships, defined in Appendix A.2. For example, for a point $\mathbf{p}$ and corner c, the feature tuple $\psi(\mathbf{p}, c) = (v_1, v_2)$ consists of the relationship values for the corner distance v_1 and corner ratio v_2. Note that unlike the relationship definition in Guerrero et al. [2014], we use points instead of poses, i.e., we discard directional information.

Point placement descriptors We call the set of all feature tuples of a point $\mathbf{p}$ in scene S the *per-point feature set* $\phi(\mathbf{p}, S) = \{\psi(\mathbf{p}, e) | e \in S\}$, which is a descriptor of the point's relationship to all surrounding scene elements.

Shape placement descriptors Now we extend the placement descriptor from points to shapes. Recall that we defined the placement of a shape by position, orientation, width, and height of the shape's bounding box. To describe the shape's relationship to neighboring scene elements, we sample its bounding box with a fixed set of points (five in our implementation: the bounding box corners and center). The descriptor of a shape placement is then simply a combination of the descriptors of its sample points. More specifically, for each scene element e, we describe the relationships between placement and element with a *compound tuple* by concatenating the feature tuples $\psi(\mathbf{p}, e)$ of all sample points for that element in a defined order (e.g., point 1, 2, ...). The set of all compound feature tuples is the *per-shape feature set* $\Phi(P, S)$, called a *shape placement descriptor*. As will become clear in the next section, this definition allows us to conveniently compare shape placements.

3.3 Similarity Measures

In this section, we first develop a measure of similarity between individual feature tuples. Then we extend this measure to point placements and finally to shape placements.

Feature tuples Assume that we are given two points, each with a feature tuple that encodes the relationship to some neighboring scene element. We can compare the placement of the two points with respect to the surrounding scene elements with a similarity measure s between two feature tuples. Three main observations guide us in the design of s. First, such a comparison is only useful for scene elements with the same label and of the same type, such that the tuple size (i.e., the number of tuple elements) is identical and their entries are comparable. Otherwise, we will define s to be zero. Second, we are interested in whether two tuple values "agree" (given a certain tolerance range) or not. For example, when comparing two distances, it is irrelevant if they differ by 10 or 100 times the tolerance range. Finally, to successfully capture various user's intents, we use a large set of tuples. For a given input placement, this set of tuples typically over-represents the user's intent such that two similar placements agree on only a small subset of the features. Therefore, a single unmatched element should only have a limited influence on the overall similarity. With these observations, we define a *SL0 similarity measure* $s(\psi_1, \psi_2)$ [1] between two feature tuples ψ_1 and ψ_2 based on the *smoothed ℓ_0 (SL0) distance* [Oxvig et al. 2012; Cui et al. 2010] as

$$s(\psi_1, \psi_2) = \begin{cases} \sum_{i=1}^{|\psi_1|} \mathcal{G}(\psi_1^i - \psi_2^i | 0, \sigma), & \text{if } type(\psi_1) = type(\psi_2) \\ 0, & \text{otherwise,} \end{cases} \quad (1)$$

where $|\psi|$ is the tuple size of ψ and ψ^i indicates the i-th element in tuple ψ. $\mathcal{G}(x|\mu, \sigma)$ is a Gaussian with mean μ and variance σ (defined further below). The function $type(\psi)$ gives the type and label of the scene element used to compute the tuple ψ. The measure in Equation (1) counts the number of comparable relationship functions with similar values, which can intuitively be understood as accumulating "evidence" that the two relationships are similar.

Point placements Now we extend the above similarity definition from a single scene element to *multiple* elements per point. The feature sets ϕ_1 and ϕ_2 for any two points (see Section 3.2) will now be used to compute a placement similarity between these points. Of course, the tuples in the two feature sets generally differ because each point is surrounded by a different number of scene elements of different type. Therefore, we establish a *bijective mapping* $\mathbf{M} : \phi_1 \to \phi_2$ from tuples in ϕ_1 to tuples in ϕ_2. To match the size of the two feature sets, we fill the smaller one with *NIL* tuples, which will give zero similarity s. $\mathbf{M}$ tells us which scene elements to compare, so that we can apply $s(\psi_1, \psi_2)$ to the corresponding tuple pairs ψ_1 and ψ_2 and accumulate the results. This gives us a measure of placement similarity for two points:

$$k(\phi_1, \phi_2) = \sum_{\psi \in \phi_1} s(\psi, \mathbf{M}(\psi)). \quad (2)$$

[1] See the supplementary material for a proof that this similarity can be used as a valid kernel for regression.

Given the above setup, our goal is to find a mapping $\mathbf{M}$ that maximizes $k(\phi_1, \phi_2)$, i.e., $\mathbf{M} = \arg\max_{\mathbf{M}} k(\phi_1, \phi_2)$. In other words, we seek an assignment between scene elements that maximizes our similarity measure. This assignment problem is well studied in graph theory and we solve it efficiently using the *Hungarian algorithm* [Kuhn 1955]. We further improve the efficiency of the assignment by using the fact that s is zero for elements of different type. Consequently, we only compute the assignment for tuple sets with the same element type. Note that since $k(\phi_1, \phi_2)$ is a sum over SL0 similarities $s(\psi_1, \psi_2)$, $k(\phi_1, \phi_2)$ is again a SL0 similarity. Maximizing $k(\phi_1, \phi_2)$ therefore corresponds to finding the mapping $\mathbf{M}$ with the maximum SL0 similarity. The resulting value of $k(\phi_1, \phi_2)$ is a good measurement of how similar two point placements are with respect to the scene elements surrounding each point.

Shape placements Finally, we extend the placement similarity k from single points to placed shapes. This extension is a straightforward replacement of the point placement descriptors in (2) with the shape placement descriptors Φ introduced in Section 3.2:

$$k(\Phi_1, \Phi_2) = \sum_{\psi \in \Phi_1} s(\psi, \mathbf{M}(\psi)), \tag{3}$$

where ψ are now compound tuples. This naturally extends our similarity measure from individual points to placed shapes, where all above arguments still hold.

3.4 The Statistical Placement Model

To take multiple example placements into account, we need to evaluate how well a candidate placement corresponds to a *set* of given example placements, based on the similarity k between two placements. Since we use feature sets as placement descriptors, we need to operate in *feature space*, i.e., the space of all possible per-shape feature sets.

The input is a set of per-shape feature sets $\mathbf{Q} = \{\Phi_1, \ldots, \Phi_n\}$ corresponding to the example placements, and a feature set $\mathbf{x}$ corresponding to a new candidate placement. While we could create a measure based on the similarities of $\mathbf{x}$ to the elements of $\mathbf{Q}$ (for example using kernel density estimation), this would give preference to placements that are provided more often by a user. Instead, the user assigns a *target value* t ($0 \leq t \leq 1$) to all $\Phi \in \mathbf{Q}$, which are then used to predict the target value of $\mathbf{x}$. Note that t also gives users some freedom to specify preferences for particular placements. Consequently, given $\mathbf{Q}$ and $\mathbf{t} = (t_1, \ldots, t_n)^{\mathrm{T}}$, we have to learn a function $y_{\mathbf{Q},\mathbf{t}}(\mathbf{x})$, in short $y(\mathbf{x})$. If no target values are given, we set all $t_i = 1$.

In machine learning, kernel regression is a method for learning functions that has several advantages for our setting. First, it uses no fixed set of basis functions that need to completely cover the space of possible functions; instead, it adapts to the given data points. Second, it can be expressed using a kernel function quantifying the similarity between two input elements. The kernel function can be chosen freely under some validity constraints and can therefore be applied directly to our feature sets. The disadvantage of kernel methods is that the evaluation can be slow because all input elements are used to represent the target function. Fortunately, this is no problem in our case as the number of example placements is comparably low.

More precisely, we follow the definition of Bishop [2006] of kernel regression, where the target function is calculated as

$$y(\mathbf{x}) = \mathbf{k}(\mathbf{x})(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{t}. \tag{4}$$

Here, λ is a regularization factor to avoid overfitting, and $\mathbf{k}(\mathbf{x}) = (k(\Phi_1, \mathbf{x}), \ldots, k(\Phi_n, \mathbf{x}))$, where $k(\mathbf{x}, \mathbf{y})$ is the kernel function defined in Equation (3). Similarly, $\mathbf{K}$ is the gram matrix of the kernel

$$K_{ij} = k(\Phi_i, \Phi_j). \tag{5}$$

We choose a kernel size σ proportional to the size of the placement. A more detailed discussion is provided in Section 5. We prove in the supplementary material that this is a valid kernel function. Note also that the methodology described in this section can be applied to per-point feature sets, and we will use this fact to simplify shape placement propagation in Section 3.5.

3.5 Shape Placement Propagation

Propagating a shape in a scene S can be described as finding five-parameter transformations (2D translation, 2D scaling, and rotation) that lead to "good" placements. Given the learned model $y(\mathbf{x})$, the quality of any given candidate placement P can be computed from its feature set $\mathbf{x} = \Phi(P, S)$. Unfortunately, exhaustively searching the five-dimensional space of possible placements to find good ones seems computationally impractical. Instead, our idea is to take advantage of the special problem structure to extract an approximate set of local maxima of $y(\mathbf{x})$.

Point propagation First, we decompose the space of placements into five two-dimensional subspaces, one (spatial) subspace for each bounding-box sample point (the bounding box center and corners, see Section 3.3). For each such sample point $\mathbf{p}_i, i \in \{1, .., 5\}$, we learn an individual statistical model $y_i(\mathbf{x})$ from all input placements using the methodology described in Section 3.4. Then, each $y_i(\mathbf{x})$ is sampled on a dense grid spanning the given spatial domain, shown as heat map in Figure 4(b). In this grid, we extract point sets $\hat{\mathbf{P}}_i$ at local maxima by non-maximum suppression with a small fixed radius [Kitchen and Rosenfeld 1982]. Intuitively, these maxima represent propagations of the sample point $\mathbf{p}_i$ to positions that are locally optimal in the statistical model y_i. To improve computational efficiency, we discard positions at local maxima below 10% of the global maximum.

Candidate placements We now derive a set of potentially good shape placements. Recall that a placement has five degrees of freedom. Therefore, it is uniquely defined by a *triplet* of three locally optimal point positions ($\hat{\mathbf{p}}_i \in \hat{\mathbf{P}}_i, \hat{\mathbf{p}}_k \in \hat{\mathbf{P}}_k, \hat{\mathbf{p}}_l \in \hat{\mathbf{P}}_l$) with $i \neq k \neq l$, except for triplets with collinear points (which leave too many degrees of freedom). We call each such triplet a *candidate placement*. Figure 4 (c),(d) shows an example of candidate placements on a façade.

Candidate rating Given the statistical model y learned from the example placements (Section 3.4), we now evaluate the generated candidate placements using their full per-shape descriptors (based on the compound tuples) $\mathbf{x}$. Then $y(\mathbf{x})$ gives us the similarity between a candidate placement and the given example set. Note that the combined statistical model y better encodes the intuition behind a placement compared with simply taking the sum of the target values y_i of all five sample points. This is illustrated in Figure 4(c) and (d). One reason is that only placements that are consistent across all sample points get a high score. In the given example, this ensures that each placement's lower left and upper left corners have relations to the same window, which avoids many unsound balcony placements.

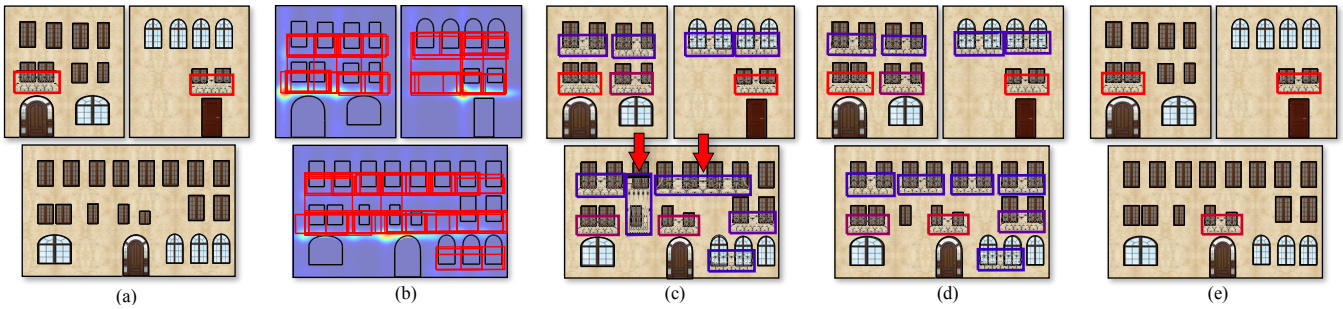


Figure 4: *Candidate placements and ranking: Given are two façades with one red example balcony placement each, shown in (a). The candidate placements in (b) are identified using individual sample points; the quality score for the lower left corner is shown as heat map. In (c) the ranking of these candidates is based on the sum of all individual sample-point scores, with the highest score in red and the lowest score in blue. One can observe that this results in undesirable placements, which are marked by red arrows. The proposed combined model used in (d) includes sufficient information about the entire placement's intuition, and the combined score gives better results. A user can adjust a score threshold so that only the placements most similar to the examples remain, shown in (e).*

Geometric deformations The statistical model $y(\mathbf{x})$ considers geometric relations to neighboring scene elements, but not geometric deformations of the placed shapes themselves. We penalize such deformations based on the change of aspect ratio and area of the shape. More precisely, let w_e,h_e,A_e be the average width, height and area, respectively, of all example-placement bounding boxes. Furthermore, let w, h, A be the width, height and area, respectively, of a particular candidate placement $\mathbf{x}$. The tolerance to deviations in these measures is then modeled as Gaussians with user-specified variance σ_{wh} and σ_A, which modify $y(\mathbf{x})$ to get $y'(\mathbf{x})$:

$$y'(\mathbf{x}) = y(\mathbf{x})\mathcal{G}(\log\left(\frac{w}{h}\frac{h_e}{w_e}\right)|0,\sigma_{wh})\mathcal{G}(\log\left(\frac{A}{A_e}\right)|0,\sigma_A). \quad (6)$$

Note that we employ only this basic form of shape matching since our focus lies on matching the relations between shapes and neighboring scene elements, rather than the geometric properties of the placed shapes themselves.

Overlapping shapes In our model, we focus on learning relevant relationships of a single shape to the surrounding geometry, but we do not consider the relationships between candidate placements of the same propagation. Therefore, a set of candidates with high scores may contain overlapping shapes, which may not be desirable in some applications. As a simple solution, we optionally remove placements that overlap with a "better" candidate placement with respect to y'.

Finally, we are left with a list of reasonably good candidate placements ordered by y'. Depending on the application, we provide visualizations (like the color coding in Figure 4) and selection tools to let a user select final placements. As a simple example, a slider might be used as a selection tool to threshold placement quality y'.

4 Applications

The proposed method can be used directly for *2D modeling applications*, like shape placement on a façade given as a set of labeled polygons, or shape placement in parcels. A large scene can be populated by providing few example placements. In this section, we introduce several extensions to improve modeling efficiency for certain types of applications and to implement several steps in a *3D city modeling pipeline* using our method, including building placement in parcels, mass modeling, façade modeling, and furniture placement. We also show how to record a modeling session in an *operations tree* that can be applied to new scenes to automatically create complex and varied models. Results for these applications will be shown in Section 5.

4.1 Extensions

Polygon hierarchy Complex layouts such as façades or street plans can often be naturally divided into several disjoint areas where the elements are placed independently; for example, if there is little influence between the layouts of adjacent parcels. Similar to Guerrero et al. [2014], we create a hierarchy of polygons to generalize this notion to arbitrary polygonal scenes. A polygon A_p is parent of a polygon A_c if it contains A_c or if the polygons intersect and A_p has a larger area. The parent polygon of a placed shape P is defined similarly, while the parent of a point $\mathbf{p}$ is defined as the smallest polygon containing the point. When computing the feature set Φ or ϕ, only relationships to elements that have the same parent as P or $\mathbf{p}$ are considered. Additionally, each polygon has a main orientation and we optionally constrain all placements to share the orientation of their parent. Finally, we can filter out placed shapes that intersect polygons with labels that were not intersected by any example placement.

Distance importance Elements close to a placed shape are typically considered to be more important than elements farther away. To take this into account, we can add a weight to the tuple similarity defined in Equation (1), based on the distance to a scene element:

$$s'(\psi_1,\psi_2) = w_1 w_2\, s(\psi_1,\psi_2), \quad (7)$$

where $w_i = \mathcal{G}(d_i|0,\sigma)$ is a Gaussian of the distance d_i between the point or placement and the scene element of tuple ψ_i. The standard deviation $\sqrt{\sigma}$ of this Gaussian is set to a multiple of the placement's bounding box diagonal (2 in our case).

Shape arrays Instead of stretching a shape to fit the placement bounding box, we can repeat the shape with a fixed size inside the bounding box, resulting in a 1D or 2D array of shapes (e.g., windows). The supplementary material provides details.

3D extensions Since our method operates on two-dimensional domains, we cannot apply it directly to three-dimensional city models. Instead, we apply it to parametrized planar surfaces in a 3D scene. Each planar surface forms a separate 2D scene, and 3D shapes placed on this surface are represented by the outline of their 2D projection onto this surface. To allow for mass modeling, we introduce extrusions of 2D shapes, including a simple roof generator, and Boolean operations on 3D shapes. For example, a building footprint might be extruded to form a house, which is merged in a Boolean union with the extruded footprint of a garage. We refer to the supplementary material for more details of these extensions.

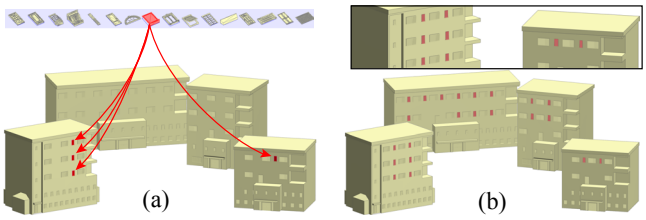


Figure 5: *Typical operations performed in our editor: In a placement operation (a), an element is dragged from the element repository (light blue) to surfaces in the scene and optionally resized. This is repeated for every example element. (b) shows results of the propagation to several buildings.*

Operations tree Each modeling operation is recorded in an *operations tree*. A recorded modeling session can then be applied to a different scene without further user interaction. Examples include applying a specific style to a given basic building shape (an example might be a saloon or a run-down hotel) or creating a small, diverse city section from a given set of parcels. Each node in this tree is an operation that is limited in scope to the output of its parent operation. Operations applied to the entire scene are root nodes. The output of a node can be split randomly into multiple disjoint sets, and different operations may be applied to each set, creating *alternative branches* in the operations tree and increasing variation in the resulting scene. Nodes in the tree are operations such as shape placement, extrusion, Boolean operations, deletion and selection, where elements with a given label are selected, for example. A more detailed description is provided in the supplementary material.

4.2 Workflow

We created a prototype application that implements our method and the extensions described above. This application handles a number of different scenarios. In particular, it handles several steps of an urban modeling pipeline: object placements in parcels, mass modeling of buildings, façade generation, and interior modeling.

To generate a complex scene, a user models a small but representative example scene, where each performed operation is recorded in an operations tree. By default, the scope of each operation is limited to the output of the previous operation. However, the user may also click on any scene element to define the output of a previous operation as the current scope. This effectively chooses a particular parent node in the operations tree. Alternatively, the operation can be applied to the entire scene, which makes the operation a root node. After the operation is performed, the user may choose to split the output randomly into several disjoint groups. All subsequent operations can then be applied to these groups separately.

For a shape placement, the user typically provides 2-5 representative examples by dragging shapes from a shape repository to a planar surface. The examples are propagated to all active surfaces (defined by the parent node as described above, or the entire scene for root nodes), taking all elements on these surfaces into account, including previously propagated shapes. Note that for satisfactory results, these examples should all represent the same modeling intention (as illustrated in Figure 2). While it is possible to provide examples for several different modeling intentions in a single operation, like "*above a door OR right of a window*", such an operation would require more examples to clearly disambiguate each of the modeling intentions. In this case, the statistical model described in Section 3.4 would have maxima for each of the modeling intentions. However, there is no need to specify multiple intentions in a single operation, as they can be propagated separately in different consecutive operations, thus requiring fewer examples per operation. An example of a placement operation is shown in Figure 5, where four examples of a window placement are propagated to all surfaces in the scene. Figure 6 illustrates how combining shape placements with extrusions and Boolean operations allows for mass modeling.

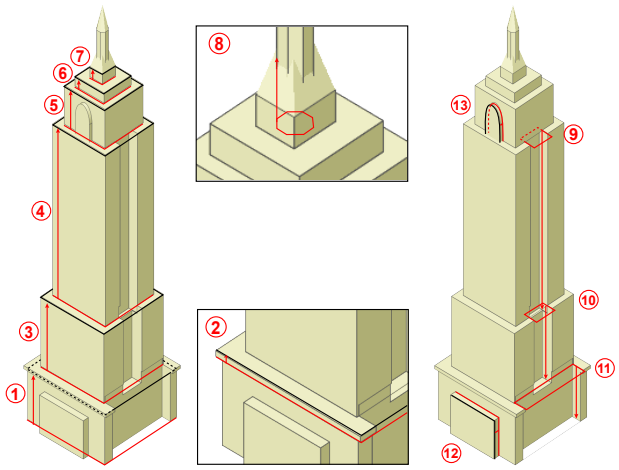


Figure 6: *Mass model creation of a skyscraper: The mass model shown above is created with several edit operations in the order indicated in the red circles. In each operation, the red shape is placed and extruded along the red arrow. In steps 1-8, 12 and 13, this is followed by a Boolean union with the parent surface. In steps 9-11, a Boolean subtraction is used instead. Steps 7 and 8 cap the extruded mesh with a roof.*

Since shape placements require an existing scene, modeling typically starts with a rough 2D layout of the desired scene, such as the parcel layout in a city scene or the basic room layout for interior scenes, but existing complex 3D scenes can also be used as starting points. After an example modeling session, the resulting operations tree can be applied to new 2D layouts or 3D scenes to generate new models without additional user interaction.

5 Results

Example scenes We demonstrate our method on four example scenes modeled in our Matlab® prototype implementation. The scenes show various steps of a city modeling pipeline. For each scene, we create an operations tree in a small example modeling session by propagating each operation using typically between 2 and 5 example placements. An optional quality score threshold can then manually be adjusted to balance a large number of placed objects (like the balconies in Figure 4d) versus similarity to the example placements (like in Figure 4e). The resulting placements are not manually changed in any way; in fact, there is no way to store manual changes in the operations tree. The operations tree is then applied to a much larger scene or to multiple different scenes to create the results described in this section. Although we did not explicitly time the modeling sessions, we give the approximate time needed to create each example scene (including creative tasks like experimenting with different shapes and layouts), as well as the number of required operations. A discussion of the computational complexity of our method follows thereafter.

Figure 7 shows a *large city scene*, where we populate the parcels in a street layout of Tempe, AZ, with three types of buildings. All parcels have an orientation that points towards the street-facing side and are labeled either 'house' or 'apartment building', corresponding to the type of building placed into the parcel. Three types of

Figure 7: *Given a set of labeled parcels, large city areas can be generated by our method without further user interaction. Three different types of houses (condominium building, two-story house and one-story house) were generated in three editing sessions. The resulting operations tree was applied to four city maps shown in the inset on the left. The left image is taken from the lower-right tile, the right image from the upper-left tile.*

houses are generated in the parcels: two types of family houses and one apartment building type. The example modeling session was done on a scene with 7 houses and 2–6 examples were provided for each operation. Note how the building shape and consequently the façade details are determined by the parcel size, creating varied house models. Other sources of variation are the random selection of alternative branches in the operations tree and the use of random extrusion heights, as described in Section 4. The example modeling session took approximately 3.5 hours: one hour for each of the two family houses and ~ 1.5 hours for the apartment building. A total of 135 operations (including selections) were used to create all three building types.

Figure 1 shows a scene with *New York-style skyscrapers*, where we apply an operations tree to coarse footprints of the skyscrapers. Nine skyscrapers were part of the example scene and between 2 and 8 examples were provided in each operation. Variation is generated in part through the skyscraper footprints and in part through random selection operations and extrusions with random height. Creating the example scene took around 4 hours and was split into four editing sessions: mass modeling (~ 1.5 hours), façades on the base floors of the skyscrapers (~ 1.5 hours), on the center floors (~ 0.5 hours) and on the top floors (~ 0.5 hours). All together, 146 operations were performed in these four editing sessions.

Figure 8 shows a subset of various *kitchen scenes* that we generated, where we applied an operations tree to single rooms. The starting point in this example is a polygon defining the shape of the room and labels indicating walls with a planned door. The example scene contained 9 kitchens and 2–6 examples were given in each operation. Note how the furniture adapts to the shape of each room, creating variation in the scenes. Additionally, alternative branches in the tree were used to place different window styles and furniture models. In total, 85 operations were performed in a modeling session that took around 3 hours.

Three *Japanese garden scenes* are shown in Figure 9. These were generated from 2D layouts of paths, lakes, stone gardens and pavilion footprints. One of the gardens was used as the example scene (marked in the figure), which was created with 18 operations in approximately one hour, with 2–6 examples given for each operation. Variation is generated mainly by adaption to the input layout, as well as alternative branches in the operations tree for pavilions with or without roofs.

Computational complexity The computational complexity of the method mainly depends on the number of example placements N_e, the number of points N_p in the position grid defined in Section 3.5, and the maximum number of elements N_f with the same label and type in the example feature sets. Since we use a kernel method, the feature sets of all grid points have to be compared to the feature sets of all example placements. For each pair of feature sets, one assignment problem has to be solved with a maximum size given by the number of elements with same type and label, as defined in Equations (2) and (3). The resulting computational complexity is $O(N_e \cdot N_p \cdot N_f^3)$. In practice, N_f does not depend on the scene size, since increasing the scene complexity usually only increases the number of independent surfaces or hierarchy polygons, like façades or parcels, while the number of mutually dependent elements inside these areas stays more or less constant. N_e is usually independent of the scene size as well; typically a user provides 2-10 example placements. This leaves the term N_p, which directly depends on the scene size, giving our method a linear time complexity in practice. In practice, operations typically take between half a second up to one minute. In the most demanding scenes, like the skyscraper facades with hundreds of elements on a single façade, a propagation can even take several minutes in our unoptimized Matlab prototype. These timings can certainly be improved in future work. For example, the relations between each sample grid entry and the surrounding geometry are computed directly in Matlab, taking every façade element into account. Considering only those elements that can actually have a significant contribution could already provide a large speedup.

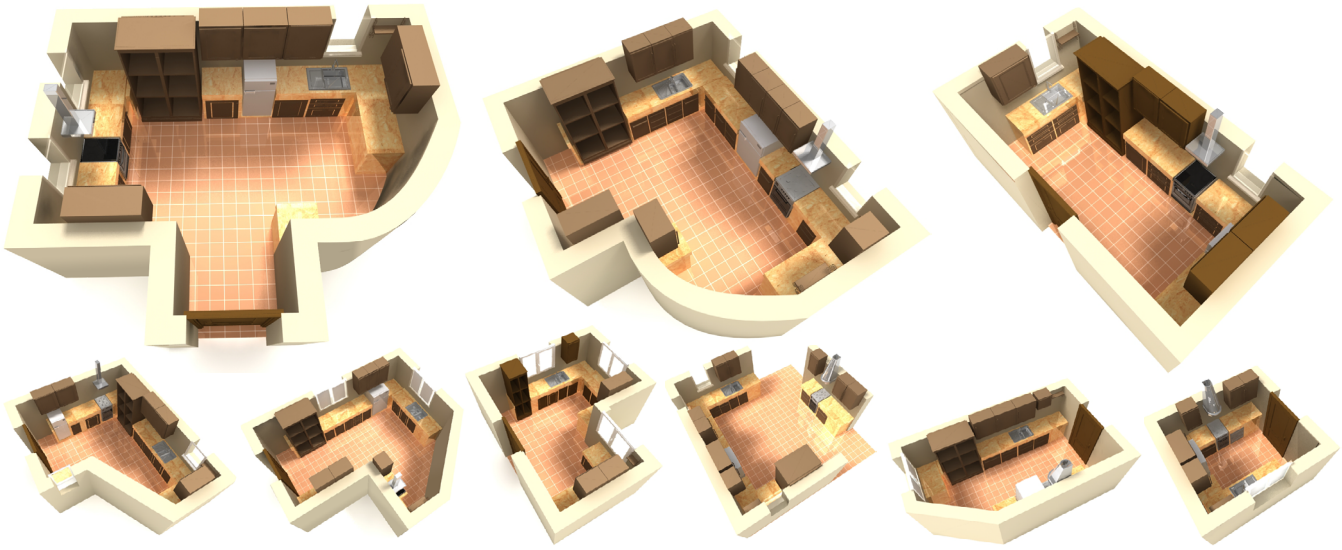


Figure 8: *Kitchen interiors generated by our method: Starting from a labeled room shape, we generate the 3D model of the kitchen, including furniture, walls and windows. Nine of the 17 generated kitchens are shown here. The supplementary material provides the full set.*

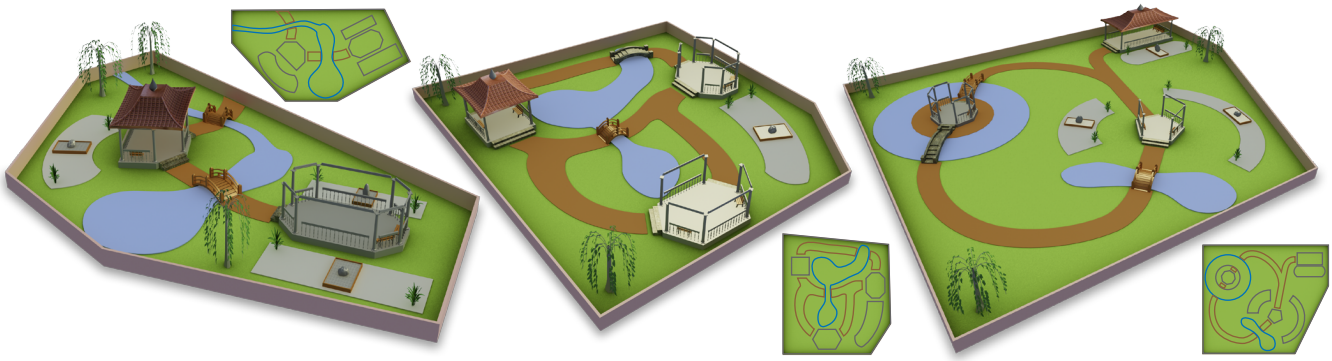


Figure 9: *Japanese garden scenes: In these scenes, a set of polygons with rather "organic" shapes was used as a starting point (see the small inlays). We populate the scenes with objects including pavilions (with handrails, columns and benches), bridges and trees in 18 operations. All example placements were performed on the right-most scene.*

Comparison to previous work Our method uses similar geometric relationships as Guerrero et al. [2014]. In the following, we discuss the main differences to this method. First, we propagate placements based on relationships at multiple points instead of poses based on relationships at a single center point. In our scenes, it is often necessary to adapt the size of an element to the surrounding geometry, for example, to adapt the width of a balcony to the width of the façade, as shown in Figure 10(a). The pose propagation of Guerrero et al. is rigid, and the relationships at a single center point do not provide enough information to adapt the width and height of a shape. Our method computes relationships at several points on a shape and adapts width and height of the shape to maximize the relationship similarity at each point. Second, we learn from multiple examples instead of a single one. This allows us to disambiguate placements by learning which relationships are important. Consider Figure 10(b), where a flower box is placed under a window that happens to be above a door. With a single example only, the relation to the door and to the window are equally important. Consequently, flower boxes are placed above all doors. With multiple examples, the user's intent is better captured and flower boxes are only placed below windows.

We additionally compare our work to the arrangement model used by Fisher et al. [2012] by adapting our method to use a Gaussian kernel and the x- and y-position relative to an object centroid as geometric relationships. In Figure 11, two mirrors are placed on a bathroom wall so they have the same width as the sink and end at the ceiling. If we take only the two relationships proposed by Fisher et al. into account, we obtain suboptimal placements since only the centroid of scene elements and not their shape is considered. Additionally, using a Gaussian kernel results in maxima that minimize the Euclidean distance to *all* of the features of an example placement (divided by a per-feature variance). However, this is often not desirable, as only some of the features are relevant for a given placement. For example, in Figure 11, the coordinates of the mirror relative the the centroid of the wall are not important, as well as the height of the mirror above the sink. In contrast, the SL0 kernel accumulates evidence and finds maxima that have the largest number of matched features, resulting in better placements.

Feature pooling In Section 3.3, we described how to compare two feature vectors of different sizes by finding an optimal assignment of features corresponding to scene elements. A more traditional approach to compare two vectors of different sizes is *feature pooling* [Boureau et al. 2010], where statistics over the feature val-

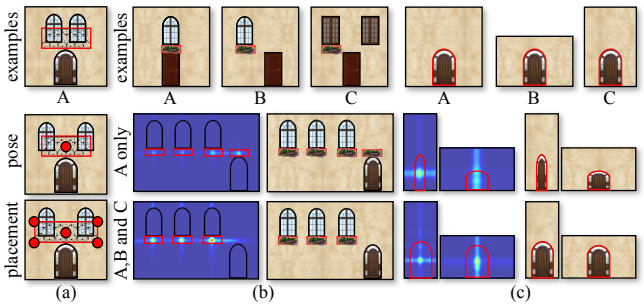


Figure 10: *Comparison of poses versus placements, as well as single examples versus multiple examples: In (a), we show that using multiple instad of a single example point allows adaption of the shape to the surrounding geometry (windows). In (b), the ambigous placement of the flower box in example A (above a door or below a window) can be resolved when using additional examples B and C. Providing multiple examples for doors on empty façades in (c) makes it clear that doors should maintain their size (bottom row) instead of adapting to the size of the façade (center row).*

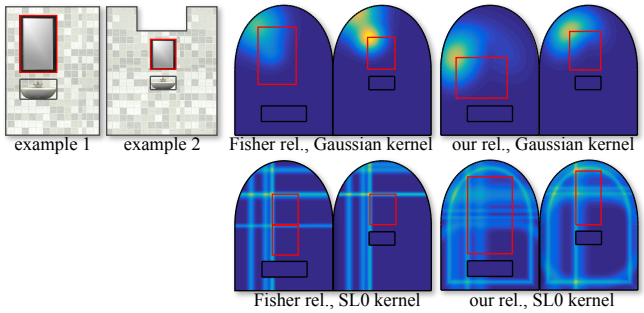


Figure 11: *Comparison to Fisher et al. [2012] for propagating the two mirror placements shown on the left. We show the propagated mirrors (red) and the prediction for the top left corner of the mirror as heat maps in the background for the two relationships proposed by Fisher et al. (center column) versus our relationships (right column) and for a Gaussian kernel (top row) versus the SL0 kernel (bottom row). Our method (bottom right) finds the best interpretation of example placements: 'Same width as the sink and ending at the ceiling.'*

ues are extracted. Figure 12 compares two feature-pooling methods to our approach. If we use only the relationships to the three closest elements, the implied assignment fails to correctly identify the relevant elements and results in bad placements, indicated by the numbered windows. Similarly, histograms are not usable since they also include the contributions of irrelevant elements (i.e., the windows on the first floor). Our approach finds the assignment that maximizes the similarity to the examples and thereby correctly identifies the relevant elements.

Deformation tolerance and overlap filter Although the focus of this paper is learning the relationships of shapes to surrounding scene elements, but not the properties of the shapes themselves, we provided a simple way to optionally penalize shape deformation in Section 3.5. The effect of the deformation tolerance is illustrated in Figure 13(a). Similarly, we focus on placing single shapes but no shape arrangements. In Section 3.5, we also provided a simple way to avoid overlap between those candidate placements presented to the user. Figure 13(b) shows the effects of this filter on a scene with three potentially good placements of the example balcony. Without the filter, all three placements are presented to the user. In contrast, the overlapping placements with a lower score are removed by the filter.

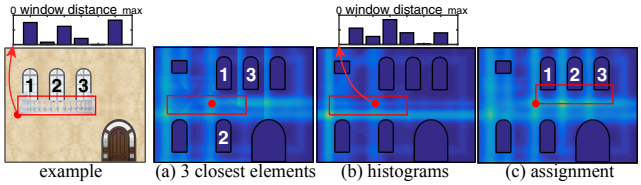


Figure 12: *Comparison to feature pooling: The balcony on the left is propagated using the three closest scene elements only (a), a histogram over feature values (b) and our scene element assignment (c). For the red point (correct position of the lower-left balcony corner), the implied assignment between windows is indicated by the numbers in (a) and (c) and the histogram of window distance values is shown in (b). Our approach compares the correct feature pairs and finds a good balcony placement (c).*

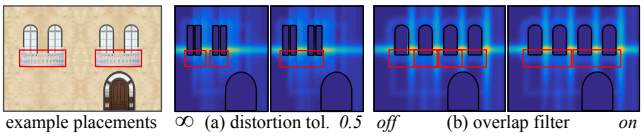
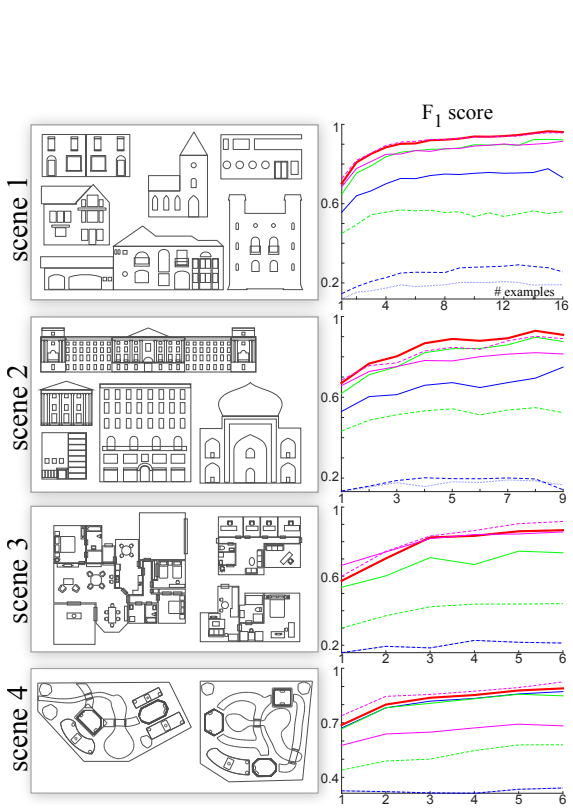


Figure 13: *Effect of deformation tolerance and the overlap filter: Two examples of a balcony placement on the left are propagated to the two scenes in (a) and (b). In (a), the distortion tolerance is set to infinity, i.e., turned off (left), and to a moderate value of 0.5 (right), while in (b), the overlap filter is turned off (left) and on (right). Depending on the situation, the resulting list of candidates (red) may be more useful when applying one of these filters.*

Quantitative evaluation In Figure 14, we show detailed quantitative evaluations of our method and compare to previous work and the variations of our own method as described above. A ground truth was manually created for several shape placement operations in the scenes shown on the left (only parts are shown here; the complete scenes are provided as supplementary material). Exhaustive cross validation (or repeated random sub-sampling cross validation with a minimum of 25 splits if exhaustive cross-validation was not feasible) was conducted to compare the results of each operation to the ground truth. The tested training set size was varied from one to the number of ground truth placements. To simulate the user adjusting the score threshold for candidate placements, we chose the optimal threshold with respect to the ground truth in each operation (in all of the tested methods). The average F_1-score (i.e., the harmonic mean of precision and recall) of our method over all operations compared to previous methods is shown in the center of Figure 14. These evaluations support our conclusion that learning from multiple examples significantly increases the quality of propagated placements. In the table on the right, we show the average F_1-score over typical sets of 2-5 examples. The proposed method results in placements with significantly higher quality than the generated by existing methods. Since Guerrero et al. [2014] only use a single example and additionally cannot adapt the shape size, the upper bound for this method is given by the "single example" column of the table. Using feature pooling with the closest three elements is on average not much worse compared with our assignment approach. However, note that there are several scenes where this approach clearly provides sub-optimal results (e.g., the balconies and windows in scene 1, decorations A and B in scene 2, classroom chairs in scene 3 in Figure 14), for the reasons discussed earlier.

Relationship influence The influence of each relationship type on the score of a point is determined by the kernel. We show the most important relationships learned from two sample points of two example placements in Figure 15. Note that the wall light varies in size, but like the example placements, the mirror remains at the size of the sink. The effect of learning the examples is noticeable in the



edit operation	# placements	Fisher Gaussian	all Gaussian	Fisher SL0	histogram pooling	closest 3 pooling	single example	ours (no overlap filter)	ours (deform. filter)	ours	
doors	16	0.62	0.73	0.54	0.87	0.89	0.70	0.84	0.90	0.89	
flowerboxes	14	0.14	0.17	0.69	0.46	0.67	0.48	0.82	0.67	0.75	
balconies	24	0.03	0.02	0.65	0.36	0.74	0.59	0.84	0.84	0.89	
doorsigns	16	0.17	0.33	0.65	0.80	0.96	0.78	0.96	0.96	0.96	
windowframes	133	0.00	0.00	0.80	0.25	0.97	0.93	0.94	0.81	0.97	
door roofs	15	0.20	0.24	0.74	0.68	0.86	0.59	0.90	0.85	0.87	
windows	18	0.03	0.03	0.72	0.37	0.59	0.47	0.78	0.73	0.71	
scene1 average		0.17	0.22	0.68	0.54	0.81	0.65	0.87	0.82	0.86	F_1 scores
windows	8	0.29	0.39	0.26	0.80	0.85	0.58	0.70	0.75	0.82	
columns	9	0.38	0.37	0.55	0.58	0.71	0.55	0.73	0.68	0.73	
staircase	13	0.21	0.31	0.50	0.47	0.72	0.57	0.73	0.59	0.70	
balconies	19	0.04	0.03	0.45	0.49	0.76	0.53	0.71	0.68	0.77	
decorations A	32	0.00	0.00	0.80	0.31	0.72	0.60	0.82	0.73	0.88	
door frames	12	0.25	0.21	0.95	0.70	0.95	0.89	0.97	0.94	0.96	
decorations B	10	0.00	0.00	0.95	0.27	0.76	0.60	0.96	0.95	0.95	
scene 2 average		0.17	0.19	0.64	0.52	0.78	0.62	0.80	0.76	0.83	F_1 scores
beds	6	0.29	0.29	0.94	0.60	0.95	0.78	0.92	0.95	0.94	
night tables	7	0.62	0.62	0.82	0.70	0.81	0.74	0.90	0.85	0.82	
rugs	6	0.01	0.01	0.69	0.34	0.69	0.46	0.73	0.63	0.69	
desk chairs	12	0.11	0.11	0.75	0.44	0.75	0.37	0.77	0.79	0.75	
classroom chairs	7	0.00	0.00	0.83	0.01	0.21	0.34	0.86	0.83	0.83	
scene 3 average		0.21	0.21	0.81	0.42	0.68	0.54	0.84	0.81	0.81	F_1 scores
pavilionbeams	34	0.07	0.07	0.93	0.63	0.90	0.83	0.94	0.56	0.93	
stone area	6	0.50	0.50	0.81	0.67	0.78	0.55	0.82	0.63	0.81	
stones	5	0.71	0.71	0.74	0.75	0.79	0.71	0.82	0.69	0.82	
bridges	6	0.00	0.00	0.82	0.07	0.81	0.58	0.89	0.78	0.82	
scene 4 average		0.32	0.32	0.82	0.53	0.82	0.67	0.87	0.66	0.84	F_1 scores

Figure 14: *Quantitative evaluation of several shape placement operations by cross-validating against a manually generated ground truth and comparison with the results of previous methods. In the center, the average F_1-score over all edit operations is plotted as the number of examples selected from the ground truth increases. Each curve corresponds to the method with the same line style in the table. The table shows the average F_1-score for each edit operation over typical sets of 2-5 examples. Part of the scenes are shown on the left. See the supplementary material for the complete scenes.*

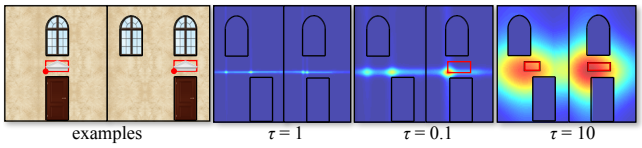


Figure 16: *Effects of changing the kernel size: In two examples, a small roof is placed between a door and a window (left). The score for the lower left edge point is shown as heat maps. The upper window is strongly misaligned, the lower window only slighty. With a small τ as the kernel size, both misalignments are too strong to be considered similar to the examples, while with a big kernel size, both misalignments are considered similar.*

low weight of the normalized bounding box width relationship to the lamp (second column, top row), compared to the same relationship in the bottom row.

Kernel size The size of the SL0 kernel is determined by the variance of the Gaussians in the SL0 similarity. Each relationship type defines a variance based on its range of values, e.g., a directional relationship requires a different variance than a relationship that computes an absolute distance. Relationships based on an absolute distance use the bounding box diagonal of the placed shape to normalize the variance. We can additionally multiply the variances with a constant factor τ to control the precision of the placements. Figure 16 shows the effect of changing τ. Lower values increase the precision and decrease the tolerance, while higher values do the opposite. A good value of τ is a tradeoff between precision and tolerance. We use $\tau = 1$ in all shown examples.

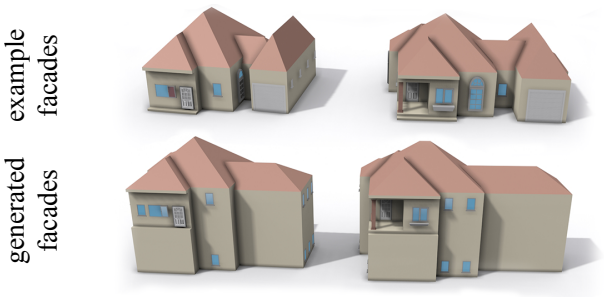


Figure 17: *Results of applying an operations tree learned from unrepresentative examples: The operations tree for a family house was learned on single-story buildings (top row). When applied to two-story buildings, there is ambiguity that results in unexpected element placements (bottom row). Providing addtional examples on two-story buildings would help to resolve the ambiguity.*

5.1 Limitations and Future Work

When an operations tree is created and example placements are learned, the example scenes should represent the types of scenes the operations tree is intended for. To illustrate this point, in Figure 17, an operations tree was generated from single-story buildings only. By applying this operations tree to buildings with two stories, too much unresolved ambiguity remains. For example, should the entrance be placed on the first or second floor, or should it cover both floors? Since our method mostly operates on a geometric level,

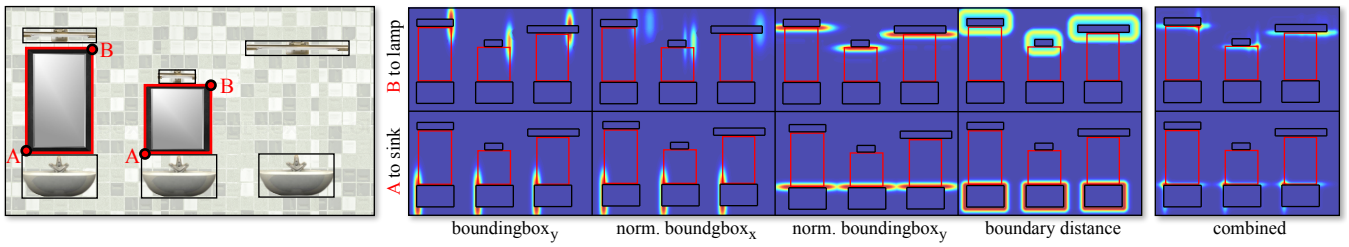


Figure 15: *Learned importance of individual relationships: A mirror placement between a sink of fixed size and a wall light of varying width is learned from two examples. Scores of individual relationships are shown as heat map on the right. For more information on the relationship types, see Appendix A.2.*

these questions could be resolved by providing additional examples.

In future work, we would like to extend our method to include geometric relationships in the full 3D space. This would allow placing 3D shapes anywhere in the scene, not only on planar surfaces of objects. Also, we currently propagate point placements without considering their orientation relative to the surrounding scene elements. Guerrero et al. [2014] show that this relative orientation can be described by several interesting geometric relationships that would enrich our statistical model. Finally, extending our definition of placements to include more general deformations would allow a better adaption of the propagated shapes to the surrounding geometry, but would also require a new technique to find the maxima of our statistical model in the higher-dimensional space of placements.

6 Conclusion

We presented a method for placing shapes "by example" in 2D scenes. In contrast to previous example-based placement methods, which are limited to a single example and a single representative point, our method learns a model from multiple example placements and takes the width, height and orientation of the examples into account. This is made possible by using kernel regression to learn the user's intent, with a kernel defined on a similarity metric that can deal with heterogeneous feature sets. Quantitative evaluations show that learning from multiple examples significantly improves the quality of propagated placements.

We further generalize the 2D placement operation to 3D scenes using extrusion and Boolean operations, which makes it possible to apply the method for a wide range of modeling tasks. Through an operations tree that records operations carried out by the user, modeling sessions can be applied to different scenes. As an example, we have demonstrated how several steps in a city modeling pipeline can be carried out efficiently using our application prototype.

Acknowledgements

We would like to thank Khaled Abd El Gawad and Yoshihiro Kobayashi for rendering the scenes shown in Section 5, as well as Virginia Unkefer for proof-reading the paper.

This publication is based upon work supported by the KAUST Office of Competitive Research Funds (OCRF) under Award No. 62140401, the KAUST Visual Computing Center and the Austrian Science Fund (FWF) projects *DEEP PICTURES* (no. P24352-N23) and *Data-Driven Procedural Modeling of Interiors* (no. P24600-N23).

References

BISHOP, C. M. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

BOKELOH, M., WAND, M., SEIDEL, H.-P., AND KOLTUN, V. 2012. An algebraic model for parameterized shape editing. *ACM Trans. Graph. 31*, 4 (July), 78:1–78:10.

BOUREAU, Y.-L., PONCE, J., AND LECUN, Y. 2010. A theoretical analysis of feature pooling in visual recognition. In *27th International Conference on Machine Learning, Haifa, Israel*.

CHAUDHURI, S., KALOGERAKIS, E., GUIBAS, L., AND KOLTUN, V. 2011. Probabilistic reasoning for assembly-based 3d modeling. *ACM Trans. Graph. 30*, 4 (July), 35:1–35:10.

CUI, Z., ZHANG, H., AND LU, W. 2010. An improved smoothed l0-norm algorithm based on multiparameter approximation function. In *Communication Technology (ICCT), 2010 12th IEEE International Conference on*, 942–945.

FISHER, M., AND HANRAHAN, P. 2010. Context-based search for 3d models. *ACM Trans. Graph. 29*, 6 (Dec.), 182:1–182:10.

FISHER, M., SAVVA, M., AND HANRAHAN, P. 2011. Characterizing structural relationships in scenes using graph kernels. *ACM Trans. Graph. 30*, 4 (July), 34:1–34:12.

FISHER, M., RITCHIE, D., SAVVA, M., FUNKHOUSER, T., AND HANRAHAN, P. 2012. Example-based synthesis of 3d object arrangements. *ACM Trans. Graph. 31*, 6 (Nov.), 135:1–135:11.

FUNKHOUSER, T., KAZHDAN, M., SHILANE, P., MIN, P., KIEFER, W., TAL, A., RUSINKIEWICZ, S., AND DOBKIN, D. 2004. Modeling by example. *ACM Trans. Graph. 23*, 3, 652–663.

GAL, R., SORKINE, O., MITRA, N. J., AND COHEN-OR, D. 2009. iWIRES: an analyze-and-edit approach to shape manipulation. *ACM Trans. Graph. 28*, 3 (July), 33:1–33:10.

GUERRERO, P., JESCHKE, S., WIMMER, M., AND WONKA, P. 2014. Edit propagation using geometric relationship functions. *ACM Trans. Graph. 33*, 2 (Apr.), 15:1–15:15.

KALOGERAKIS, E., CHAUDHURI, S., KOLLER, D., AND KOLTUN, V. 2012. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph. 31*, 4 (July), 55:1–55:11.

KITCHEN, L., AND ROSENFELD, A. 1982. Gray-level corner detection. *Pattern Recognition Letters 1*, 2, 95 – 102.

KUHN, H. W. 1955. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly 2*, 1-2, 83–97.

MERRELL, P., SCHKUFZA, E., LI, Z., AGRAWALA, M., AND KOLTUN, V. 2011. Interactive furniture layout using interior design guidelines. *ACM Trans. Graph. 30*, 4 (July), 87:1–87:10.

OXVIG, C. S., PEDERSEN, P. S., ARILDSEN, T., AND LARSEN, T. 2012. Improving smoothed l0 norm in compressive sensing using adaptive parameter selection. *CoRR abs/1210.4277*.

XU, K., STEWART, J., AND FIUME, E. 2002. Constraint-Based Automatic Placement for Scene Composition. In *Graphics Interface*, 25–34.

YEH, Y.-T., YANG, L., WATSON, M., GOODMAN, N. D., AND HANRAHAN, P. 2012. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM TOG 31*, 4 (July), 56:1–56:11.

YU, L.-F., YEUNG, S.-K., TANG, C.-K., TERZOPOULOS, D., CHAN, T. F., AND OSHER, S. J. 2011. Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph. 30*, 4 (July), 86:1–86:12.

ZHENG, Y., FU, H., COHEN-OR, D., AU, O. K.-C., AND TAI, C.-L. 2011. Component-wise controllers for structure-preserving shape manipulation. *Computer Graphics Forum 30*, 2, 563–572.

A Scene Elements and Relationship Functions

A.1 Scene Elements

Relationship functions use the following scene elements:

- a **polygon** A, defined by a sequence of vertices $\mathbf{v}_k$, where $k \in \{1, \ldots, n_A\}$,
- a polygon **segment** $g_j \subseteq A$, defined as a subsequence of consecutive smooth vertices $g_j = \mathbf{v}_a, \ldots, \mathbf{v}_b$, where $a, b \in \{1, \ldots, n_A\}$. A vertex is called smooth if the dot product of its adjacent edges is above a threshold $(\frac{\mathbf{v}_{k+1}-\mathbf{v}_k}{||\mathbf{v}_{k+1}-\mathbf{v}_k||} \cdot \frac{\mathbf{v}_k-\mathbf{v}_{k-1}}{||\mathbf{v}_k-\mathbf{v}_{k-1}||} > \epsilon)$, otherwise it is called sharp,
- a **corner** c_j of a polygon, defined by a vertex $\mathbf{v}_{k_j}$ that is shared by two adjacent segments g_{j-1} and g_j. Corners correspond to sharp vertices.

A.2 Relationship Functions

We use three relationship functions, one for each element type. These functions compute tuples of geometric relationships. For *polygon* elements, we compute a 3-tuple, consisting of three values:

- The **boundary distance** is defined as the minimum distance between a point and a polygon boundary: $R_{bd}(\mathbf{p}, A) = \min_{\mathbf{x} \in b(A)} d(\mathbf{p}, \mathbf{x})$, where $b(A)$ is the boundary of A and d the Euclidean distance.
- The **bounding box coordinates** are defined as the x and y-components of the cartesian coordinates in the local coordinate frame of a polygon with origin at the bounding box minimum, the bounding box maximum and the bounding box center of the polygon, for a total of six relationship functions.
- The **normalized bounding box coordinates** are bounding box coordinates with origin at the bounding box minimum, normalized to the width and height of the bounding box.

For *segment* elements, we compute a 2-tuple:

- The **segment distance** $R_{gd}(\mathbf{p}, g)$ is the same as the boundary distance, but considers only a segment $g \in A$.
- The **segment arc length** is the normalized arc length between the location of minimum distance to a point p and the start of the segment. If $\mathbf{x} = \arg\min_{\mathbf{x} \in b(g_j)} d(\mathbf{p}, \mathbf{x})$, then $R_{gl} = \frac{t_x - t_a}{t_b - t_a}$, where t_x is the arc length at $\mathbf{x}$ and t_a, t_b the arc length at the start and the end of the segment, respectively.

For *corner* elements, we compute a 2-tuple:

- The **corner distance** is the distance between a point and a polygon corner: $R_{cd}(\mathbf{p}, c_j) = d(\mathbf{p}, \mathbf{v}_{k_j})$.
- The **corner ratio** is the angle that the direction from corner to point $\mathbf{p}$ makes with the start of the first polygon segment adjacent to the corner, normalized with the opening angle of the corner: $R_{cr}(\mathbf{p}, c_j) = \frac{\angle(\mathbf{p}-\mathbf{v}_{k_j}, \mathbf{v}_{k_j-1}-\mathbf{v}_{k_j})}{\angle(\mathbf{v}_{k_j+1}-\mathbf{v}_{k_j}, \mathbf{v}_{k_j-1}-\mathbf{v}_{k_j})}$, where $\angle(\mathbf{a}, \mathbf{b})$ is the angle between vectors: $\angle(\mathbf{a}, \mathbf{b}) = \arccos\left(\frac{\mathbf{a}}{||\mathbf{a}||} \cdot \frac{\mathbf{b}}{||\mathbf{b}||}\right)$.