Supplementary Material for

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

1 Details of the Extensions

In the following we provide more detailed information for some of the extensions introduced in Section 4 of our paper.

Shape Arrays allow propagating arrays of shapes that adapt the number of elements to the surrounding geometry with a single placement propagation. The choice between scaling or repeating shapes to fill the bounding box of a placement can be specified separately for the width and height dimensions. To implement shape arrays, we specify a minimum spacing between shapes and find the maximum number of shapes that can be placed along the width and height dimensions while respecting this minimum spacing. The spacing is then adjusted so that there is no space before the first and after the last shape in both dimensions.

Labels can be given to each scene element. Edges and Corners get the label of the polygon they are part of by default, but can be given individual labels as well. As described in Section 3.3 of our paper, tuples corresponding to scene elements with different labels have zero similarity s.

Planar Surfaces on 3D objects are either created and parametrized automatically as part of the mass modeling operations described below, or the surfaces and their parameterization are defined by the user. All elements in the scene are placed on a surface, e.g., a house is placed on the parcel surface, and a window is placed on the facade surface. On such a surface, the elements are represented as two-dimensional polygons, corresponding to the projection of the elements onto the surface. Each surface forms a separate scene, thus relations are only computed inside a surface, not between surfaces. The parameterization of a surface is a two-dimensional euclidean coordinate system that has the same scale as the global coordinate system of the three-dimensional scene (i.e., it is an isometry of the x-y coordinate plane in the global coordinate system).

Extrusions New 3D elements may be created through the extrusion of polygons. For example, the footprint of a building may be extruded to create the basis for a building model. Planar surfaces are automatically found and parametrized on the extruded mesh. Polygons are always extruded in a direction normal to their surface with a height specified interactively by the user. An optional random multiplier can be applied to the height to add variation to the resulting models. Straight boundary segments of the polygon correspond to planar surfaces on the extruded 3D element. Finally, a simple roof or a flat cap can be added to the bottom or top side for a more realistic appearance.

Boolean Operations To construct more interesting 3D shapes, we can combine several elements using Boolean operations such as unions or subtractions. This allows the creation of more complex building mass models with variation introduced by the placement of the individual elements. Boolean operations are applied in one of two ways: Either by merging or subtracting one or several elements with/from their surface, or by or merging intersecting elements on the same surface.

Operations Tree Each modeling operation performed by the user during an example modeling session is recorded in an *operations tree*. This tree can then be applied to a different scene without additional user interaction to create varied and complex models. Operations that may be nodes of this tree are:

Placement, which finds new placements in a given input scene from the bounding boxes of a set of example elements. The information that is recorded in the tree consists of the mesh that is to be placed and the kernel regression model trained on the example placements, as well as the five models for the sample points described in Section 3.5 of our paper. Each trained model consists of the feature sets Φ_i or ϕ_i of the example placements or sample points, and the adjusted target values $\mathbf{t}'$ that result from training the models.

Extrusion, which extrudes a given set of polygons to create new meshes and surfaces. The height of the extrusion is stored in the operations tree, as well as flags indicating which type of cap should be added to the extruded model (flat, roof or none).

Boolean Operation, which merges or subtracts an element from its own surface, or subtracts/merges all elements on the same surface that intersect the given element. No parameters need to be stored for this operation.

Deletion, which deletes elements. Again, no parameters are needed.

Selection, which selects elements by label. Only the label is stored in the operations tree.

The tree is defined by connecting the input elements of child operations to the output elements of their parent. The scope of each operation is then limited to its input elements. Each operation can optionally specify if the output should be split randomly into a given number of disjoint sets. This effectively creates alternative branches in the tree, representing alternative chains of operations to be performed on different subsets of the operation's output elements. Creating a tree with alternative branches increases the variation of the resulting scene, since the possible chains of operations from the root to the leaf of the tree increase exponentially with the number of branching nodes along the chain.

2 Validity of the SL0 Kernel

In this section we show that the SL0 kernel defined in Equation (2) and (3) is valid:

$$k_{\mathrm{SL0}}(\Phi_1, \Phi_2) = \sum_{\psi \in \Phi_1} s(\psi, \mathbf{M}(\psi)), \tag{1}$$

where we denote the kernel with k_{SL0} to avoid confusion with other kernels.

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

To prove this is a valid kernel, we first show that, given a set of example feature sets $\mathbf{Q}$, the kernel evaluated between an example set $\Phi \in \mathbf{Q}$ and any other feature set is equivalent to the SL0 similarity in a high-dimensional vector space:

$$k_{\mathrm{SL0}}(\Phi, \mathbf{x}) = s(\Phi', \mathbf{x}') \tag{2}$$

for suitable vectors Φ' and $\mathbf{x}'$. Then we proceed to show that the SL0 similarity is a valid kernel.

2.1 The SL0 Kernel is Equivalent to an SL0 Similarity in a Suitable Vector Space

The example feature sets $\Phi \in \mathbf{Q}$ do not form a vector space, since they generally do not have the same number or types of tuples. To get the sets into a common vector space, we perform a series of expansions of the feature sets. We first give a short overview of these steps and continue with a more detailed proof.

First, we expand the example feature sets in $\mathbf{Q}$ to have the same number and types of tuples and give them a specific order, resulting in the expanded feature sets $\Phi^* \in \mathbf{Q}^*$. Any given feature set $\mathbf{x}$ can then be expanded as well to have the same number and type of tuples as the feature set in $\mathbf{Q}^*$. Through the assignment M, we can perform a second expansion of feature sets Φ^* and $\mathbf{x}^*$ that results in a consistent ordering of the tuples in all feature sets. Since all feature sets now have the same number and type of tuples that are consistently ordered, we can simply append their tuples to get vectors in a common vector space. The SL0 similarity on these vectors is equivalent to the SL0 kernel on the original feature sets.

First Expansion We first identify all of the scene elements that correspond to any of the tuples in the example feature set $\mathbf{Q}$ and denote this set with $\mathbf{E}$. Then we expand the feature sets Φ to include tuples for all elements in $\mathbf{E}$. Added tuples are filled with *NIL* values that always result in zero SL0 similarity. The expanded feature sets now all have the same number and types of tuples. Thus, we can order the tuples by elements $e_1 ... e_{|\mathbf{E}|} \in \mathbf{E}$, giving us ordered sets of tuples Φ^*.

Since our kernel is a sum over the SL0 similarities of individual tuples, and $\mathbf{M}_{\mathbf{x},\Phi^*} : \Phi^* \to \mathbf{x}$ maximizes this sum, it is easy to see that the result of the kernel remains the same when replacing example feature sets Φ with the expanded feature sets Φ^*:

$$\begin{aligned} k_{\mathrm{SL0}}(\Phi, \mathbf{x}) &= k_{\mathrm{SL0}}(\Phi^*, \mathbf{x}) & (3)\\ &= \sum_{i=1...|\mathbf{E}|} s(\psi_i, \mathbf{M}_{\mathbf{x},\Phi^*}(\psi_i)), & (4)\end{aligned}$$

where ψ_i are the tuples of the ordered set $\Phi^* = \{\psi_1, \ldots, \psi_{|\mathbf{E}|}\}$. Here we explicitly show the dependence of $\mathbf{M}$ on the feature sets Φ^* and $\mathbf{x}$. This expression is equivalent to a sum of SL0 similarities over corresponding tuples in Φ^* and an ordered feature set $\mathbf{x}^*_{\Phi^*} = \{\chi_1, \ldots, \chi_{|\mathbf{E}|}\}$:

$$\begin{aligned} k_{\mathrm{SL0}}(\Phi, \mathbf{x}) &= \sum_{i=1...|\mathbf{E}|} s(\psi_i, \mathbf{M}_{\mathbf{x},\Phi^*}(\psi_i)) & (5)\\ &= \sum_{i=1...|\mathbf{E}|} s(\psi_i, \chi_i), & (6)\end{aligned}$$

where $\chi_i = \mathbf{M}_{\mathbf{x},\Phi^*}(\psi_i)$ is the feature tuple of $\mathbf{x}$ assigned to ψ_i. The ordered feature set $\mathbf{x}^*_{\Phi^*}$ is the expanded version of feature set $\mathbf{x}$, which now has the same number and type of tuples as any feature set in $\mathbf{Q}^*$.

Since all feature sets are now ordered and have the same number and types of tuples, they can be thought of as vectors in vector space of dimension $N = \sum_{i=1..|\mathbf{E}|} n_{e_i}$, where n_{e_i} is the tuple size for element $e_i \in \mathbf{E}$. Then,

$$\begin{aligned} k_{\mathrm{SL0}}(\Phi, \mathbf{x}) &= \sum_{i=1...|\mathbf{E}|} s(\psi_i, \chi_i) & (7)\\ &= s(\Phi^*, \mathbf{x}^*_{\Phi^*}), & (8)\end{aligned}$$

which follows from the definition of the SL0 similarity in Equation (1) of our paper.

Second Expansion Remember that our goal is to find a single feature vector for Φ and a single feature vector for $\mathbf{x}$ in a high-dimensional vector space that have an SL0 similarity equivalent to the SL0 kernel of Φ and $\mathbf{x}$. We now have two feature vectors Φ^* and $\mathbf{x}^*_{\Phi^*}$; however, $\mathbf{x}^*_{\Phi^*}$ still depends on Φ^*, due to the dependence of the assignment $\mathbf{M}_{\mathbf{x},\Phi^*}$ on Φ^*. To find a single vector for $\mathbf{x}$, we have to remove the dependence of $\mathbf{x}^*_{\Phi^*}$ on Φ^*. For this purpose, we introduce a second expansion of the feature vectors $\mathbf{x}^*_{\Phi^*}$ and Φ^*.

Specifically, we concatenate the vectors $\mathbf{x}^*_{\Phi^*_j}$ $j = 1 \ldots |\mathbf{Q}|$, such that

$$\mathbf{x}' = \left[(\mathbf{x}^*_{\Phi^*_1})^{\mathrm{T}}, (\mathbf{x}^*_{\Phi^*_2})^{\mathrm{T}}, \ldots, (\mathbf{x}^*_{\Phi^*_{|\mathbf{Q}|}})^{\mathrm{T}}\right]^{\mathrm{T}}, \tag{9}$$

where square brackets denote concatenation. Feature vector $\Phi^*_j \in \mathbf{Q}^*$ is expanded to get Φ'_j as follows:

$$\Phi'_j = \left[O_1, \ldots, I_j, \ldots, O_{|\mathbf{Q}|}\right]^{\mathrm{T}} \Phi^*_j, \tag{10}$$

where O_i denotes a *NIL* matrix and I_i the identity matrix, both square matrices of size $N \times N$. This simply puts Φ^*_j at the same position as $\mathbf{x}^*_{\Phi^*_j}$ in vector $\mathbf{x}'$ and fills the remaining entries with *NIL*. Then,

$$\begin{aligned} k_{\mathrm{SL0}}(\Phi_j, \mathbf{x}) &= s(\Phi^*, \mathbf{x}^*_{\Phi^*}) & (11)\\ &= \sum_{k \neq j} s(O_k \Phi^*_j, \mathbf{x}^*_{\Phi^*_k}) + s(I_j \Phi^*_j, \mathbf{x}^*_{\Phi^*_j}) & (12)\\ &= s(\Phi'_j, \mathbf{x}') & (13)\end{aligned}$$

for all $j \in \{1 \ldots |\mathbf{Q}|\}$, since $O_k \Phi^*_j$ only contains *NIL* elements and $s(O_k \Phi^*_j, \mathbf{x}^*_{\Phi^*_k})$ is therefore 0, q.e.d.

2.2 The SL0 Similarity is a Valid Kernel

Next, we prove that the SL0 similarity of two vectors is a valid kernel in the sense defined by Bishop [2006]. To do so, we show that the SL0 similarity can be constructed from simpler kernels that are known to be valid. The Gaussian kernel

$$k_{\mathcal{G}}(x, y) = \mathcal{G}(x - y | 0, \sigma) \tag{14}$$

is known to be valid (see [Bishop 2006], Section 6.2). Then by repeatedly applying the kernel construction rule (see [Bishop 2006], Equation 6.21)

$$k(\mathbf{x}, \mathbf{y}) = K_a(x_a, y_a) + K_b(x_b, y_b), \tag{15}$$

which states that the left-hand side is a valid kernel if the kernels on the right-hand side are valid, we see that k_{SL0} is a valid kernel, q.e.d.

In the next section, we give a different proof of the kernel validity, which is based on its dual representation as a linear basis function model, and allows an interesting visualization of these basis functions.

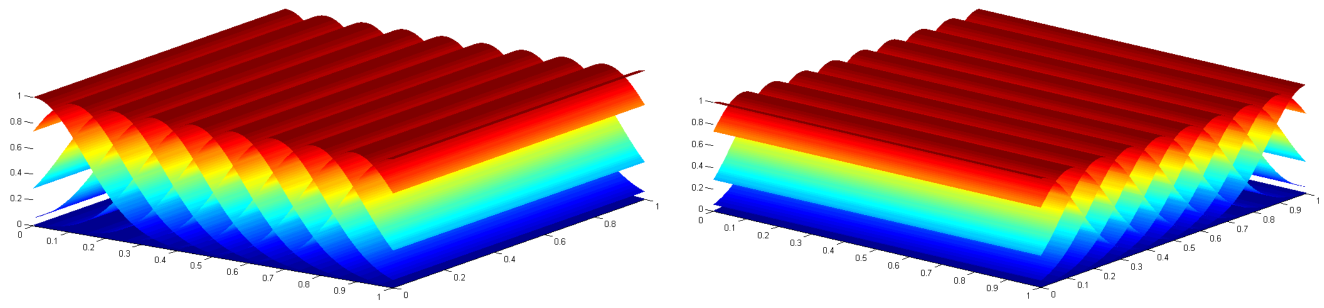

Figure 1: *Basis function for regression. We show all basis functions for a 2-dimensional feature space. The left image shows all basis functions that are non-constant in the first dimension, the right image shows the basis functions that are non-constant in the second dimension.*

2.3 Dual Representation of the SL0 Kernel

In this section, we show that linear regression with SL0 similarity as kernel (and thus, as shown before, also with the SL0 kernel) is the equivalent dual representation of a linear regression model that uses a regularly spaced set of extruded 1D gaussian basis functions (see Figure 1) as the number of basis functions approaches infinity. The model is defined as:

$$y(\mathbf{x},\mathbf{w}) = \sum_{j=1}^{M} w_j \phi_j(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}) \tag{16}$$

where M is the number of basis functions and the basis functions ϕ_j have the shape described above:

$$\phi_j(\mathbf{x}) = \mathcal{G}(x_{k(j)}|\mu_j, \sigma) \tag{17}$$

where $k(j)$ is the non-constant dimension of basis function j. We assume a constant variance σ, so the basis functions can be parametrized by their mean and the index of their non-constant dimension k. To enable the use of a fixed, finite set of basis functions, we restrict the domain of our model to a hyper-rectangle in feature space, defined by a minimum and maximum value in each dimension. Along each of the dimensions, we place a fixed number L_k of basis functions, for a total of $M = \sum_k L_k$ basis functions in our model. For dimension k we place basis functions that are non-constant in that dimension in regular intervals $\mu_{k1} \ \ldots \ \mu_{kL}$ between the minimum and the maximum value of that dimension. Equation 16 then becomes:

$$y(\mathbf{x},\mathbf{w}) = \sum_{k=1}^{D} \sum_{l=1}^{L_k} w_{kl} \, \mathcal{G}(x_k|\mu_{kl}, \sigma) \tag{18}$$

To show that this model is equivalent to liner regression with the SL0 kernel, we show that (see [Bishop 2006], Equation 6.10):

$$k_{\mathrm{SL0}}(\mathbf{x},\mathbf{y}) = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{y}) = \sum_{j=1}^{M} \phi_j(\mathbf{x}) \phi_j(\mathbf{y}) \tag{19}$$

as M approaches infinity. Using Equation 17, this becomes:

$$k_{\mathrm{SL0}}(\mathbf{x},\mathbf{y}) = \sum_{j=1}^{M} \mathcal{G}(x_{k(j)}|\mu_j, \rho) \, \mathcal{G}(y_{k(j)}|\mu_j, \rho) \tag{20}$$

where we use basis functions with variance ρ. Recall that the basis functions are defined on regular intervals μ_{kl} in each dimension of the feature space. We can therefore split the sum into an outer sum over all dimensions and an inner sum over all basis functions in that dimension.

$$k_{\mathrm{SL0}}(\mathbf{x},\mathbf{y}) = \sum_{k=1}^{D} \sum_{l=1}^{L_k} \mathcal{G}(x_k|\mu_{kl}, \rho) \, \mathcal{G}(y_k|\mu_{kl}, \rho) \tag{21}$$

Since the Gaussian function is symmetric around the origin, we can switch the argument with the parameter μ

$$k_{\mathrm{SL0}}(\mathbf{x},\mathbf{y}) = \sum_{k=1}^{D} \sum_{l=1}^{L_k} \mathcal{G}(\mu_{kl}|x_k, \rho) \, \mathcal{G}(\mu_{kl}|y_k, \rho) \tag{22}$$

The scaling of the Gaussians is unimportant. When scaling a Gaussian by a factor c, the result of the regression is the same (the corresponding entry in $\mathbf{w}$ is only scaled by the inverse c^{-1} of that factor). We can scale the Gaussians by the distance $\Delta\mu_k$ between two adjacent Gaussian centers. Note that $\Delta\mu_k$ only depends on the dimension since the centers of the Gaussians are spaced equally along each dimension.

$$k_{\mathrm{SL0}}(\mathbf{x},\mathbf{y}) = \sum_{k=1}^{D} \sum_{l=1}^{L_k} \mathcal{G}(\mu_{kl}|x_k, \rho) \, \mathcal{G}(\mu_{kl}|y_k, \rho) \, \Delta\mu_k \tag{23}$$

As L_k goes to infinity, the sum becomes an integral. Factoring out the inverse normalization factor $\frac{1}{\sqrt{2\pi\rho}}$ for a normal distribution from each Gaussian we get

$$k_{\mathrm{SL0}}(\mathbf{x},\mathbf{y}) = 2\pi\rho \sum_{k=1}^{D} \int \mathcal{N}(\mu_k|x_k, \rho) \, \mathcal{N}(\mu_k|y_k, \rho) \, d\mu_k \tag{24}$$

The integral of the product of two normal distributions is a zero-mean normal distribution of the distance between the centers:

$$k_{\mathrm{SL0}}(\mathbf{x},\mathbf{y}) = 2\pi\rho \sum_{k=1}^{D} \mathcal{N}(x_k - yk|0, 2\rho) \tag{25}$$

$$= \sqrt{\pi\rho} \sum_{k=1}^{D} \mathcal{G}(x_k - yk|0, 2\rho) \tag{26}$$

Which is equivalent to the definition of the SL0 kernel with $\sigma = 2\rho$ up to the constant scaling factor $\sqrt{\pi\rho}$.

3 Additional Results

In the following pages, we show the full set of models that we generated for the skyscrapers, city and kitchen scenes. Figures 2 and 3 show all generated skyscrapers, Figures 4 to 7 show additional views of the city scene and Figures 8 to 11 show all generated kitchens.

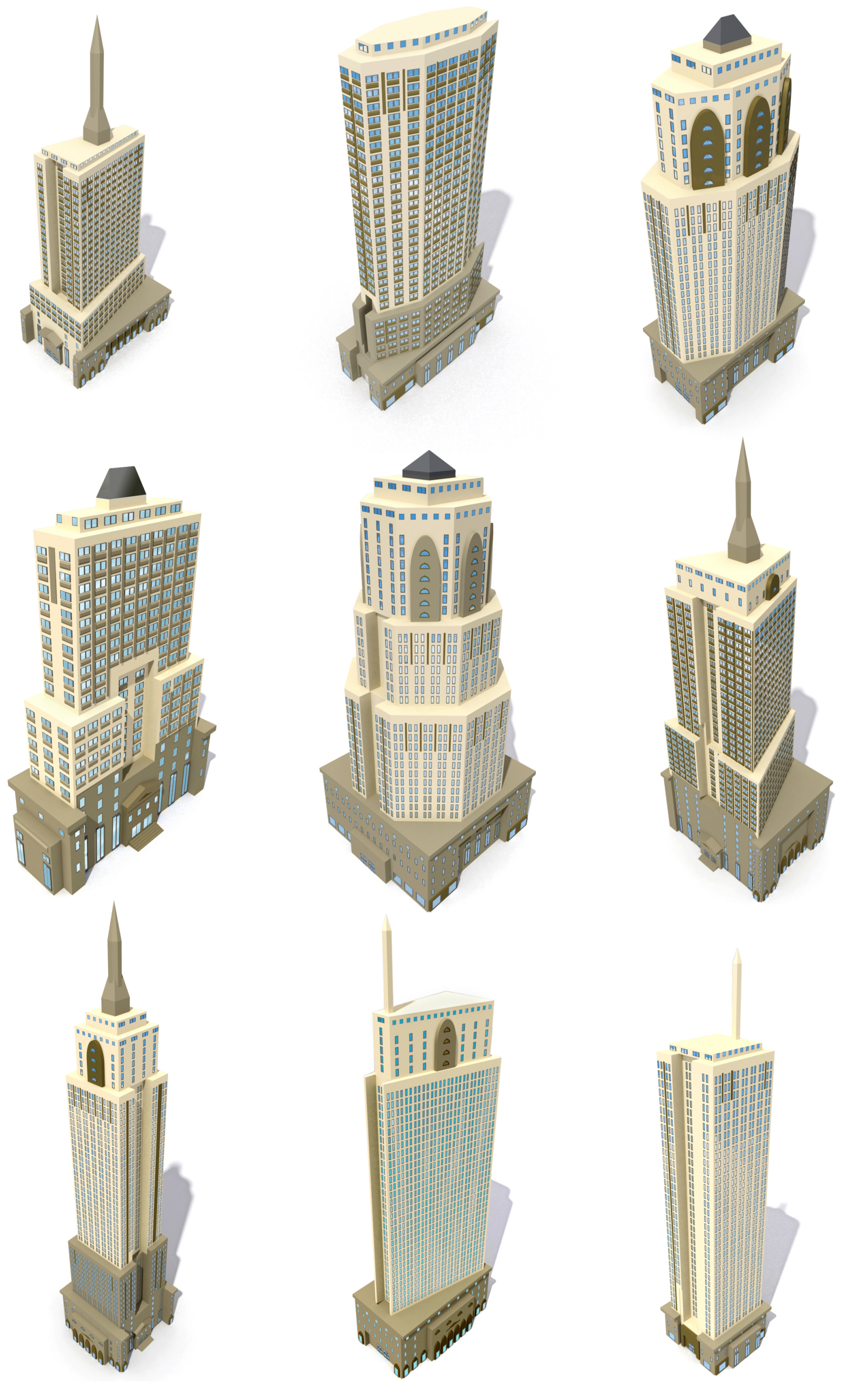

Figure 2: *Skyscrapers 1 to 9 generated by our method. Note how our method can be used to create varied mass models that guide the placement of facade elements to create interesting variations in the final models. A total of 146 operations were performed in the example modeling session.*

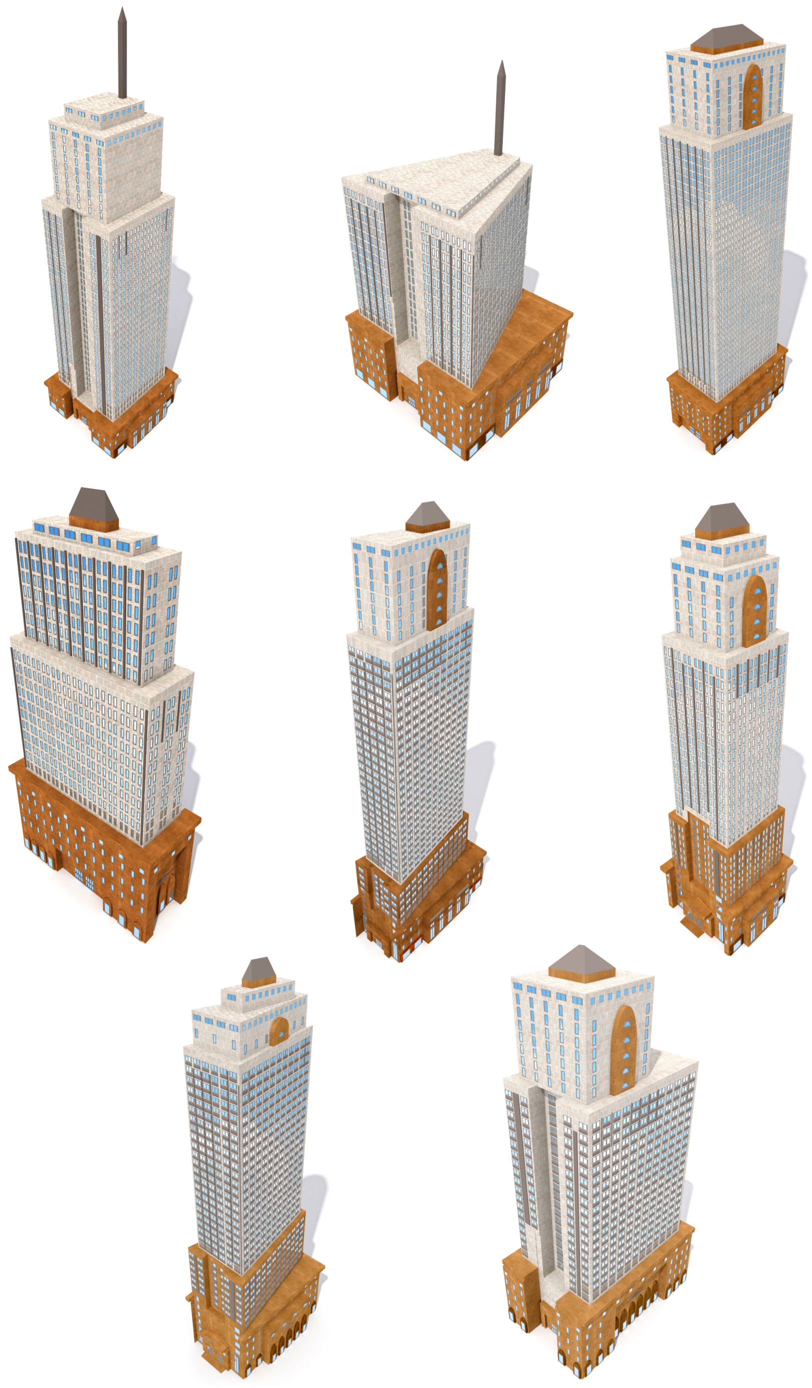

Figure 3: *Skyscrapers 10 to 17.*

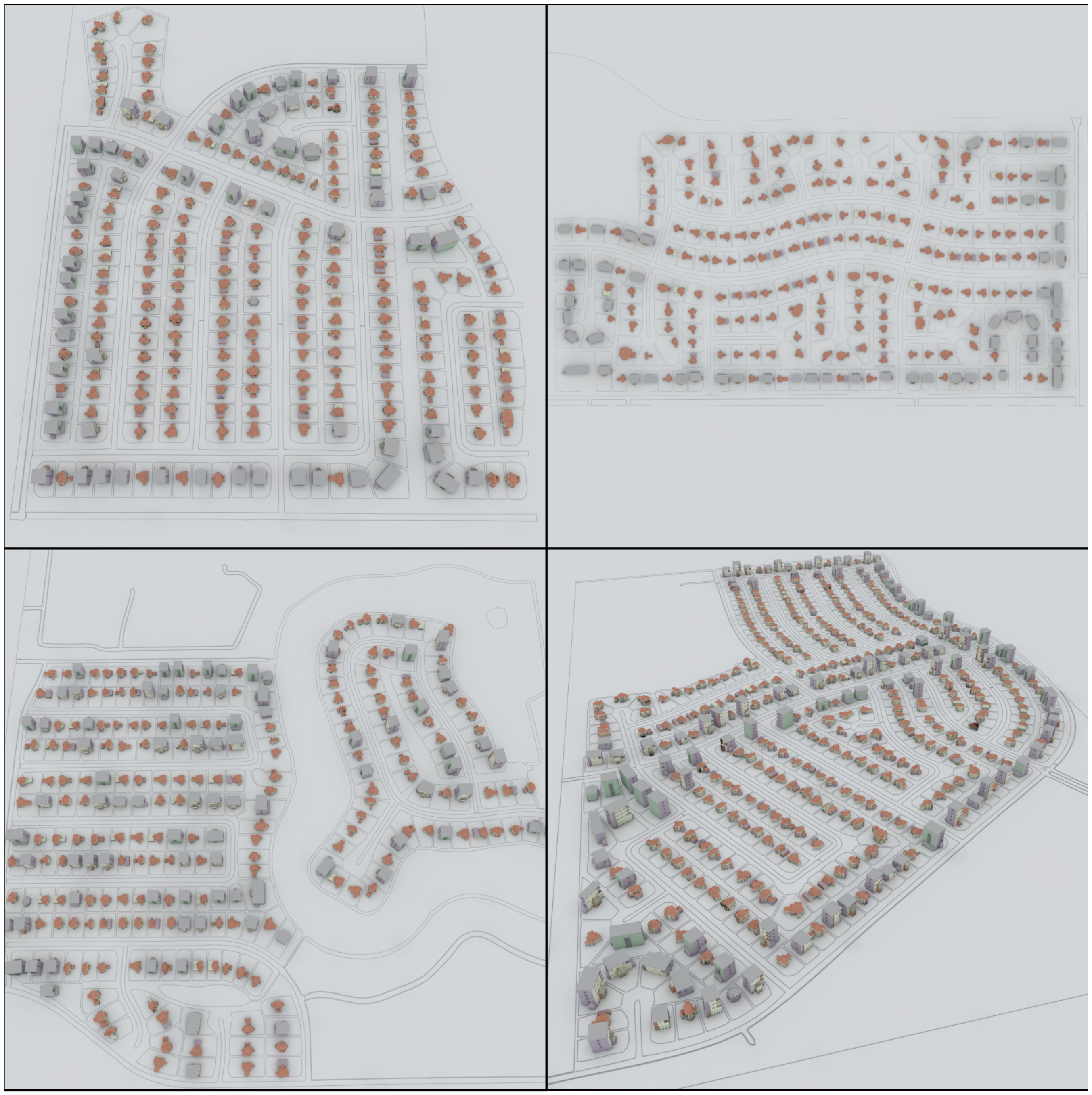

Figure 4: *Overview of the four tiles of the city scene generated by our method. Input for this scene was the parcel and street layout shown as lines on the ground plane. A total of* 1181 *houses were created from an example modeling session with 135 Operations (including selection operations).*

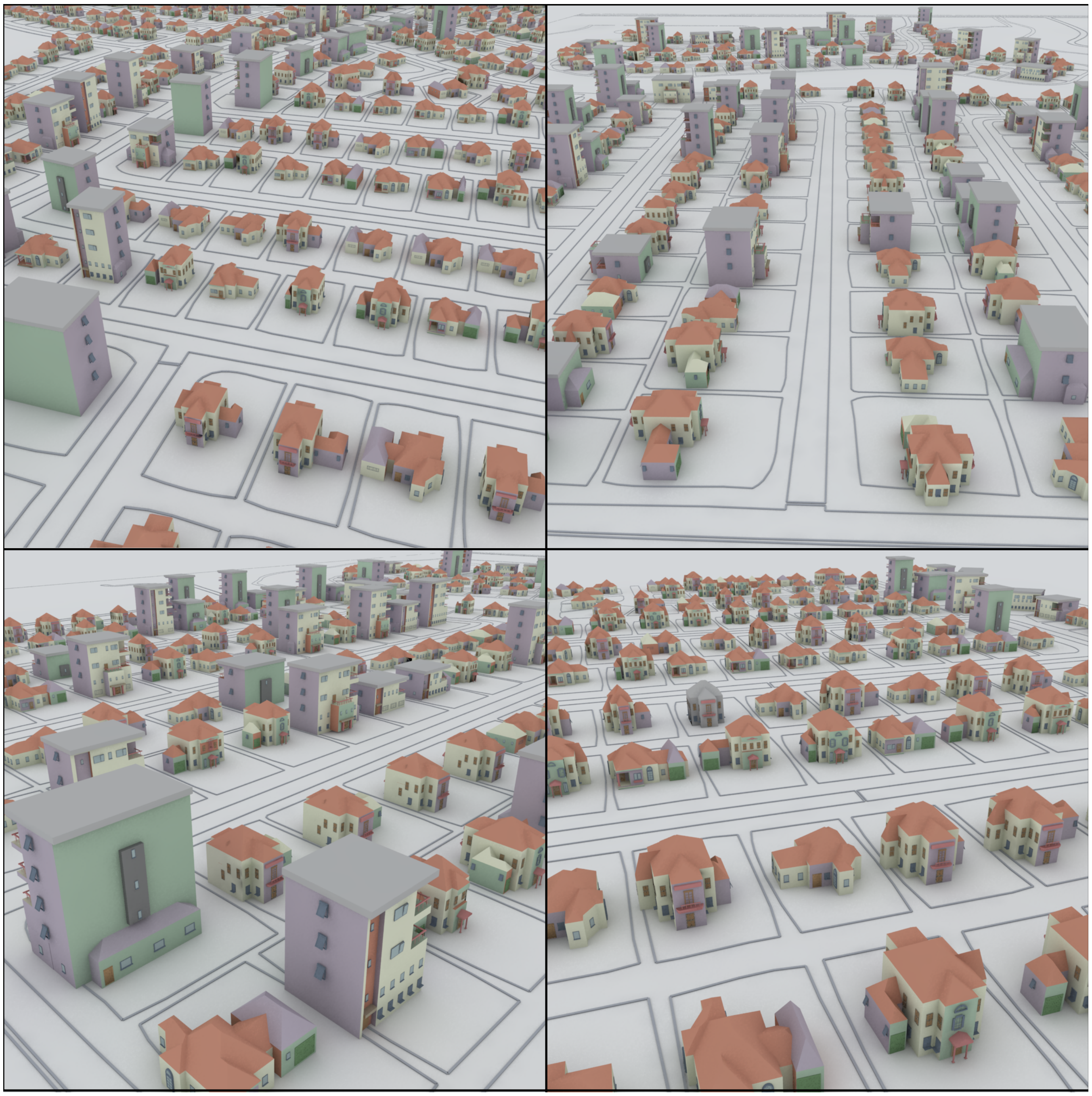

Figure 5: *Aerial views of the city scene. Note how the varying parcels shapes and additionally random parameters in the operations tree result in many varied models.*

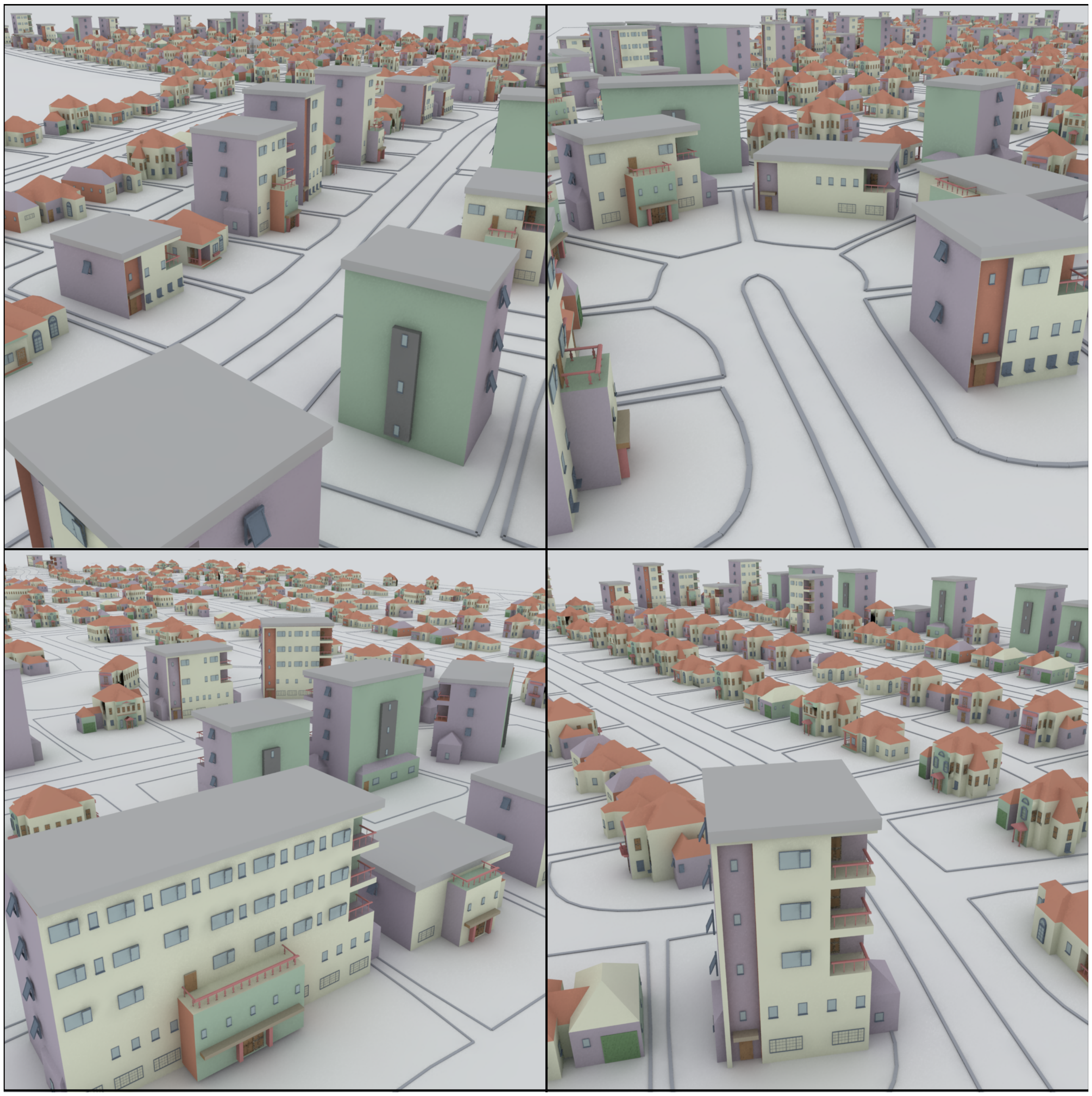

Figure 6: *A closer view on some parts of the city scene. Note how individual facade elements like windows and handrails adapt to the size of the facade, which in turn is influenced by the size of the parcel.*

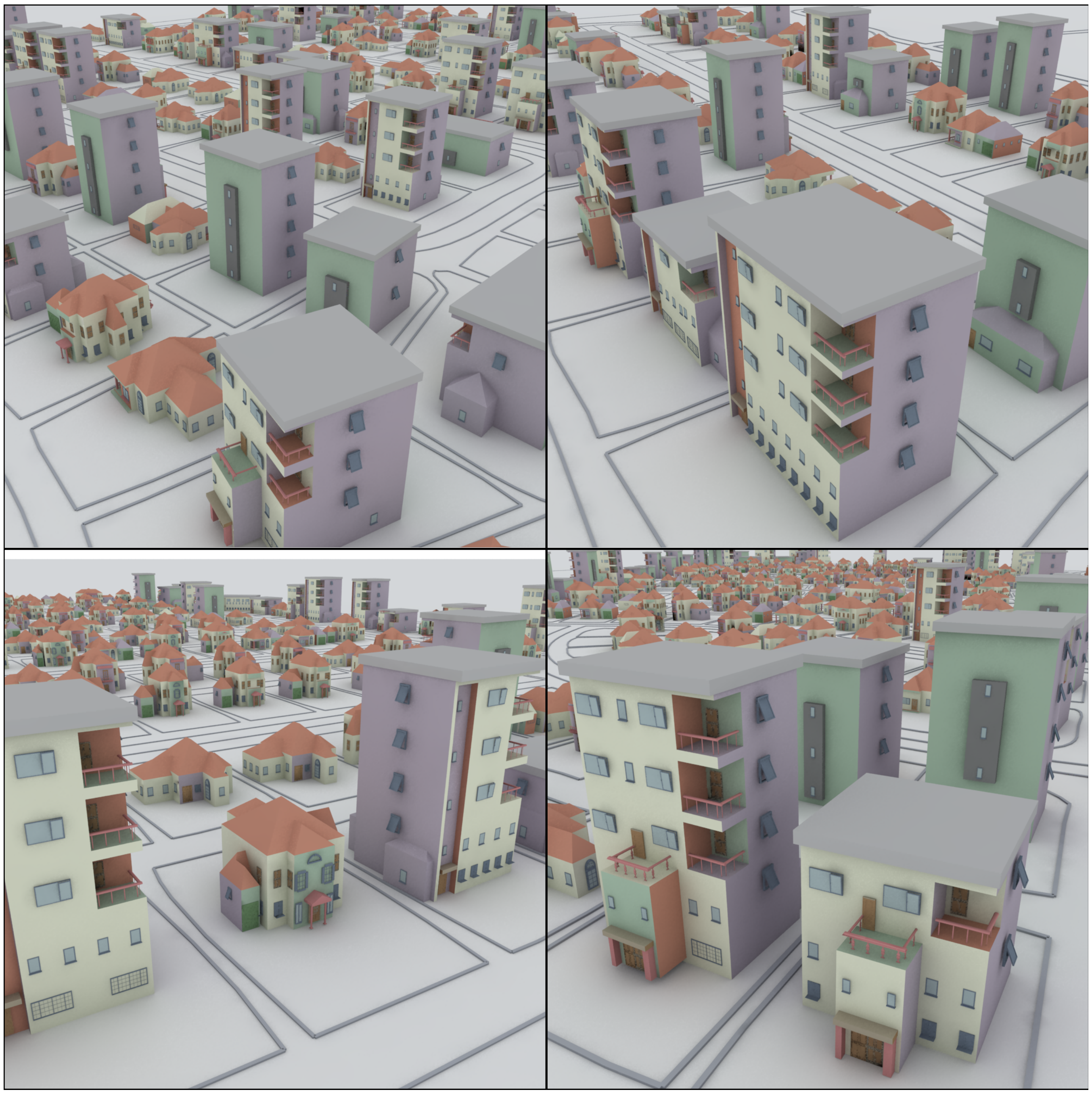

Figure 7: *Close-up view of buildings in the city scene. Note how the shape of the buildings guide the placement of elements to create interesting facade layouts.*

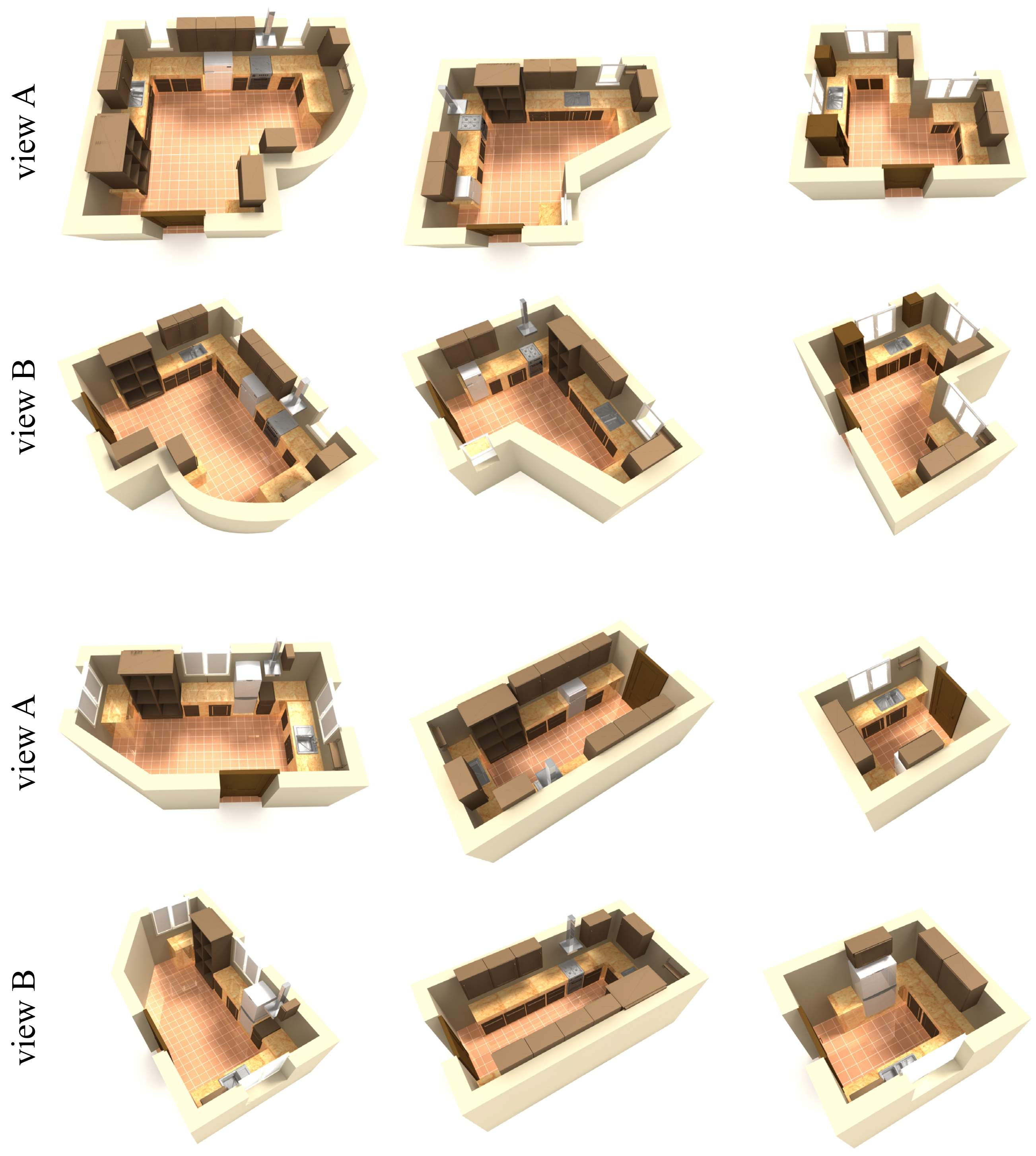

Figure 8: *Kitchens 1 to 6 generated by our method. We show two views of each kitchen (view A and view B). Note how the furniture placements adapt to the shape of the kitchen. Variation also comes from alternative branches in the operations tree, which are used for example to place different kinds of fridges, stoves and sinks, or to vary the width of the cupboards. A total of 85 operations were performed in the example modeling session.*

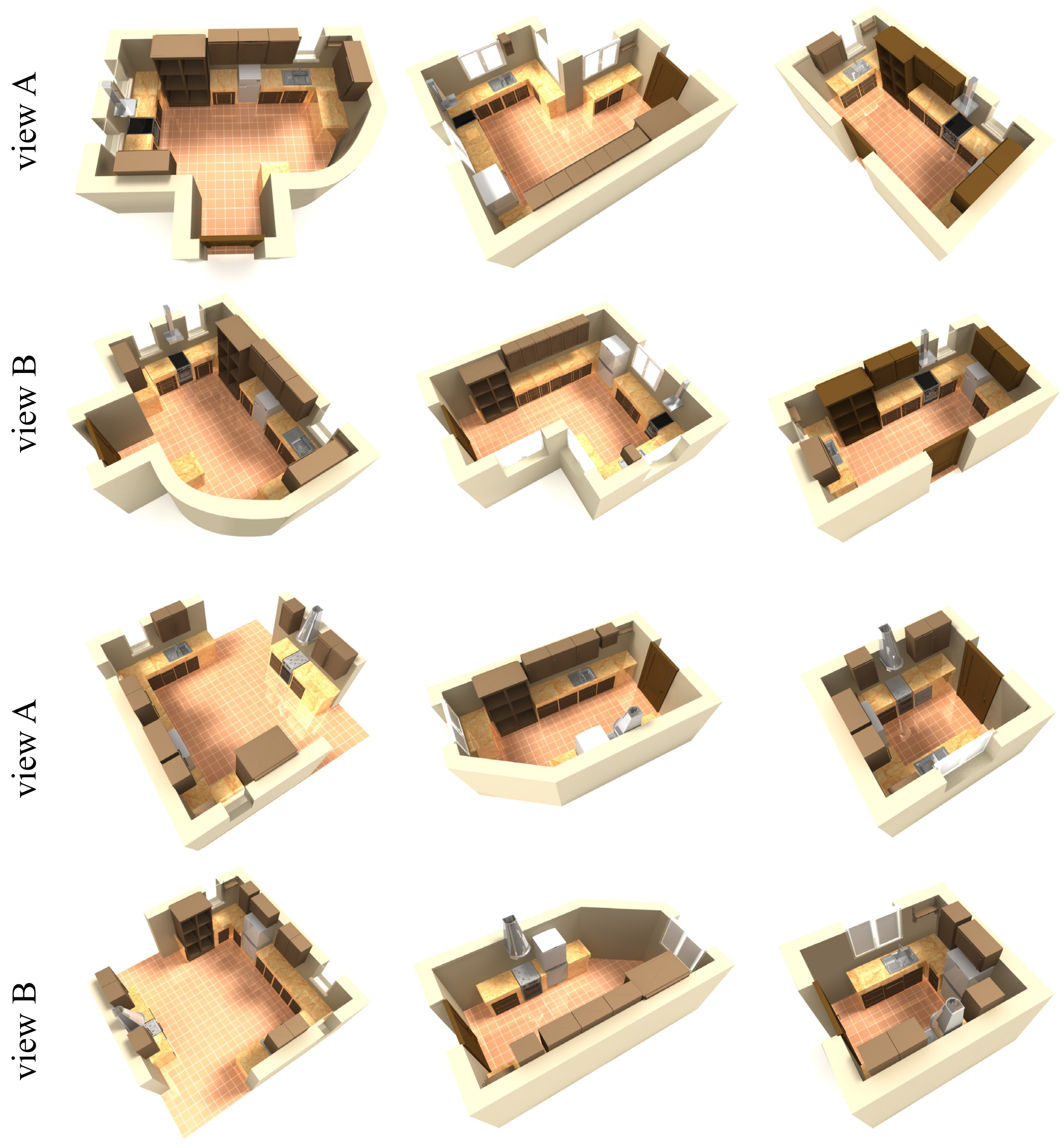

Figure 9: *Kitchens 7 to 12 generated by our method. We show two views of each kitchen (view A and view B).*

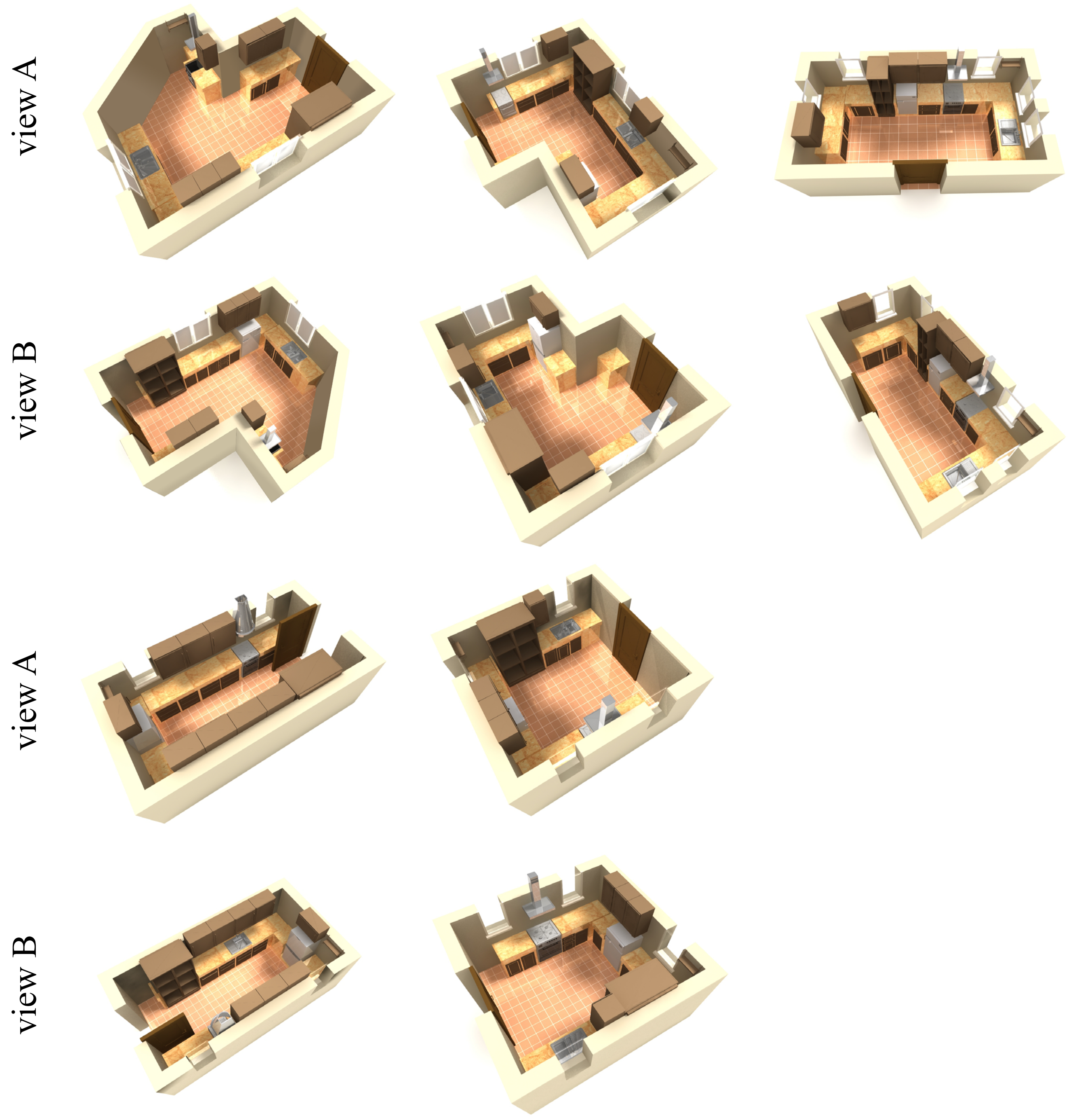

Figure 10: *Kitchens 13 to 17 generated by our method. We show two views of each kitchen (view A and view B).*

4 Scenes used in the Quantitative Evaluations

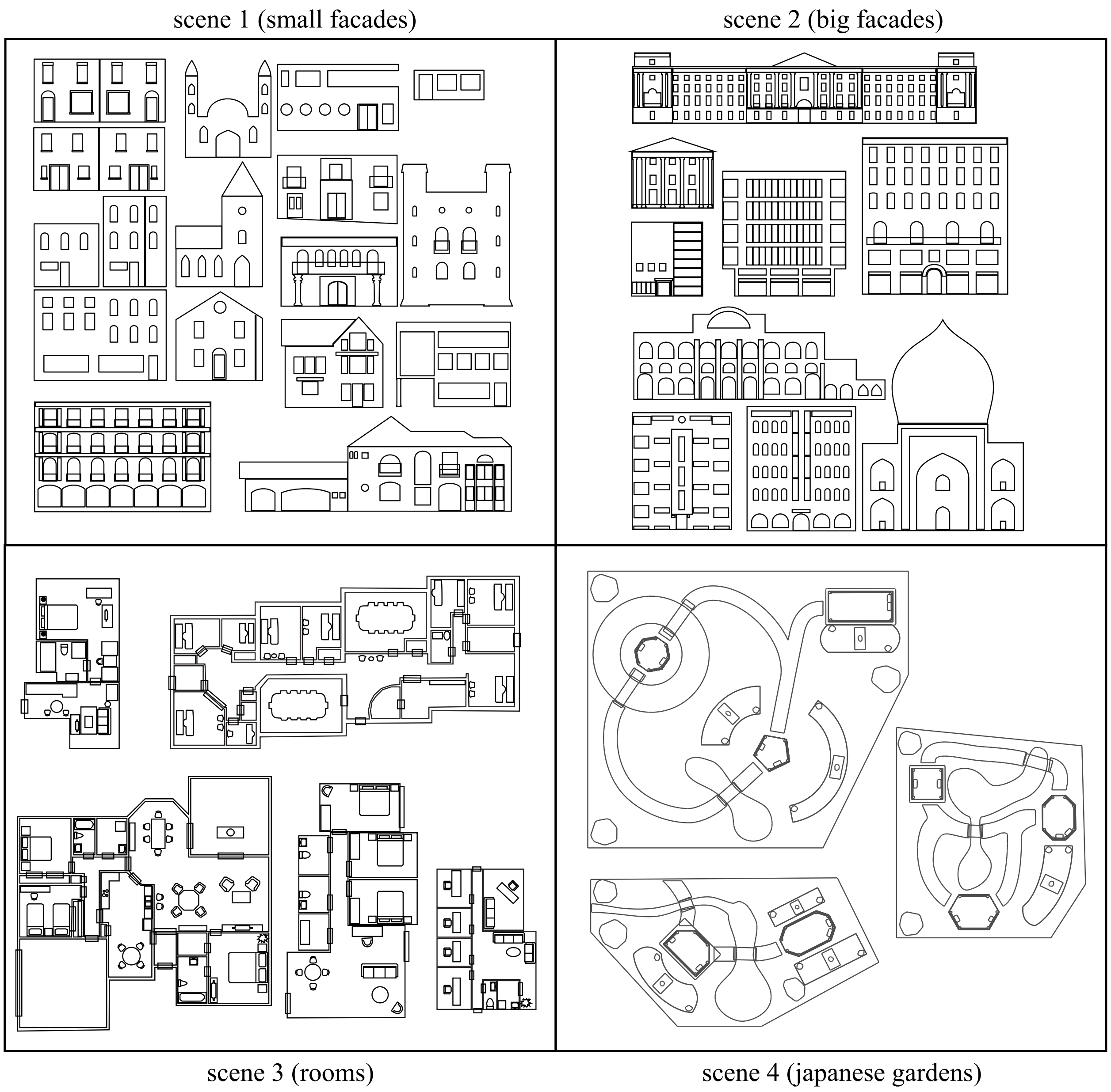

Figure 11: *The four scenes used in our evaluations. All operations were performed on variations of these scenes that represent different stages in their construction process. The first scene contains small- to medium sized facades, including family houses and small apartment buildings. Scene 2 contains larger facades that might be used to create landmark buildings or office buildings. Scene 3 contains several floor plans with different types rooms including offices, living rooms and kitchens. Finally, scene 4 shows three japanese gardens, with elements such as paths, lakes and stone gardens that have a more organic shape.*

References

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