Hierarchical Co-Segmentation of Building Facades

Anđelo Martinović and Luc Van Gool
ESAT-PSI/VISICS, KU Leuven
andelo.martinovic@esat.kuleuven.be, luc.vangool@esat.kuleuven.be

Abstract

We introduce a new system for automatic discovery of high-level structural representations of building facades. Under the assumption that each facade can be represented as a hierarchy of rectilinear subdivisions, our goal is to find the optimal direction of splitting, along with the number and positions of the split lines at each level of the tree. Unlike previous approaches, where each facade is analysed in isolation, we propose a joint analysis of a set of facade images. Initially, a co-segmentation approach is used to produce consistent decompositions across all facade images. Afterwards, a clustering step identifies semantically similar segments. Each cluster of similar segments is then used as the input for the joint segmentation in the next level of the hierarchy. We show that our approach produces consistent hierarchical segmentations on two different facade datasets. Furthermore, we argue that the discovered hierarchies capture essential structural information, which is demonstrated on the tasks of facade retrieval and virtual facade synthesis.

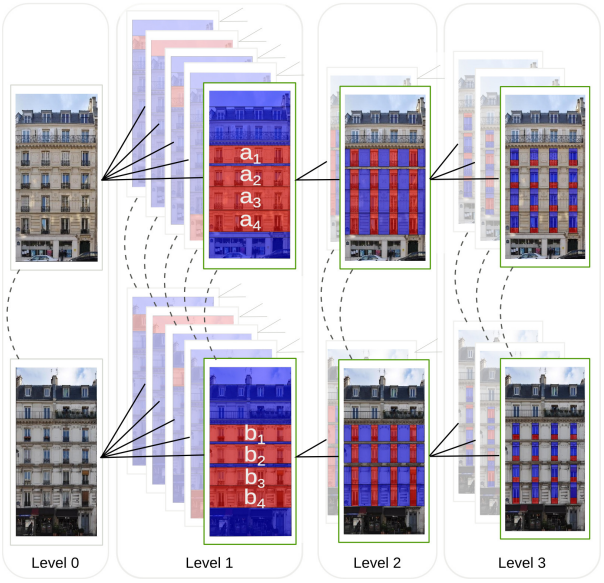


Figure 1. (Best viewed in color) An example hierarchical joint segmentation of two facade images. Solid lines represent the hierarchical decomposition. Dashed lines indicate which segments are used in a joint segmentation. For example, one joint segmentation is performed for the initial facades, and one for segments $\{a_i \cup b_i\}$.

1. Introduction

In the field of 3D city modeling, accurate reconstructions of buildings are essential when creating realistic models of urban spaces. Simple plane fitting and texturing is typically insufficient for an immersive, realistic user experience as errors often show up during unrestricted user movement through the reconstructed urban areas. Due to the fact that facades are the most prominent parts of buildings, modeling their structure has received considerable attention in the research community, resulting a wide spectrum of approaches tackling the problem of facade structure understanding.

On one side of this spectrum, model-based techniques encapsulate prior knowledge about the building structure as a set of rules, most commonly using shape grammars for architecture, popularised by [10]. These methods constrain the final reconstruction to be an instance of the predefined model, which creates accurate and visually pleasing results [18]. However, their downside is the reliance on a style-specific model which may not be readily available when dealing with different architectural styles.

In contrast, model-free techniques for facade parsing are typically based on supervised semantic segmentation approaches [8, 3, 2]. Although these methods provide impressive performance when pixel accuracy is concerned, their output is limited to a flat, 2-dimensional labeling of the input image. However, as many other man-made objects, facades exhibit a strong hierarchical structure, which cannot be captured by a flat segmentation approach.

Therefore, some authors have proposed to *learn* facade structure from data. Muller et al. [11] estimates a regular grid of facade elements and converts it into a shape grammar. Shen et al. [15] extends this approach by modeling multiple interlaced grids. Other facade structure learning

This work was supported by the KU Leuven Research Fund and the European Research Council (ERC) under the project VarCity (#273940).

methods either depend on user interaction [12, 1, 7] or predefined abstractions of the input facades in form of labeled boxes [23] or pixelwise annotations [9, 21].

In this work, we propose to merge the gap between the supervised facade parsing techniques, which produce imperfect labelings, and the structured learning approaches that require clean data to work. Due to the existence of noise in the data, we argue that the structure cannot be reliably estimated from a single facade image, and thus propose a joint optimization of a set of facade images. We create consistent hierarchies by performing a joint segmentation of facades and their parts recursively, see Fig. 1 for an illustration. The joint segmentation at each level is performed using a modified linear programming technique originally introduced in [6] for 3D shape segmentation.

The two approaches most similar to ours are the works of Shen et al. [15] and Van Kaick et al. [19]. In the first approach, a hierarchy is created for each facade independently, resulting in less stable hierarchies with no correspondence across images. In the second approach, coanalysis is performed on a set of 3D shapes, but with the key difference that the hierarchies are first created independently for each shape and subsequently merged. Furthermore, they are limited to an analysis of binary trees, while our approach allows us to create n-ary trees from the outset.

2. Our approach

We start with a set of N facade images $\mathbf{F}$ and their noisy semantic segmentations (labelings) $\mathbf{L}$. We obtain the latter by running the first two layers of [8], which provide labeling results with the highest pixel accuracy, without introducing any explicit architectural knowledge. Other approaches for semantic segmentation could also be used, such as [3, 2]. Our main assumption is that the facades follow the Manhattan-world assumption, and can be decomposed by recursive splitting in the vertical and horizontal direction. This is a common assumption used by a large number of previous works in urban modeling [22, 15, 3, 18] which does not preclude the existence of non-rectangular elements (e.g. round windows) since they can still be represented with a bounding box. We define a *scope* z as an axis-aligned bounding box which contains a non-empty area of an image and its corresponding labeling. The initial set of images is thus converted to a set of N scopes $\mathbf{Z} = \{z_i\}$, each scope completely covering one facade image.

At every step of the hierarchy, we want to find the optimal segmentations of all scopes, such that the created segments are consistent across scopes. Due to the Manhattan assumption, we can restrict our search by considering only segments generated by splitting the scope in one of the main splitting directions. A valid k-way segmentation of a scope thus consists of k adjacent segments separated by $k-1$ splitting lines. A brute-force approach to finding the consistent segmentations would be to consider every possible combination of split lines in each scope, and selecting the combination which maximizes some predefined consistency score. Since this would be too computationally expensive, we reduce the dimensionality of the problem by limiting the number of allowed split line positions. This is done by first generating an oversegmentation of each scope into a large number of smaller segments, or *slices*, and constraining each segment to be a superset of contiguous slices (see Fig. 2). This idea is similar to using superpixels [13] in general image segmentation, or patches in shape segmentation [6]. The optimal subsets of segments for each scope are then selected by a modified co-segmentation approach of [6], detailed in Sec. 4. Then, a hierarchical decomposition is created with a recursive approach detailed in Sec. 5. Similar segments across scopes are discovered in a graph clustering step. All segments in one cluster are used as the input to the joint segmentation stage in the next level of the hierarchy. The process continues until the produced clusters contain uniform elements (e.g. wall regions) or elements too small for subdivision. In Sec. 6 we show that the resulting hierarchies can be used for structural facade retrieval and sampling of virtual facades. Our contributions are as follows: (1) A novel approach for creating consistent hierarchical decompositions of building facades. To the best of our knowledge, we are the first to use a co-segmentation approach in this context; (2) A graph clustering approach for automatic discovery of semantically similar elements across images; (3) A new tree distance measure for comparing n-ary trees based on sequence matching.

3. Initial segmentation

In order to generate an oversegmentation of a scope, we define a support function for placing a split line at each position in the scope:

$$\Upsilon(z) = \Upsilon_{IG}(\mathbf{F}_z) \cdot \Upsilon_{IC}(\mathbf{F}_z) \cdot \Upsilon_{LB}(\mathbf{L}_z) \cdot \Upsilon_{LC}(\mathbf{L}_z) \quad (1)$$

This function aggregates the data support from both the original image and the noisy labeling through the following four factors, normalized to the interval $[0, 1]$:

Image gradient support Υ_{IG} [15, 3] promotes placing of horizontal (vertical) split lines where horizontal (vertical) edges or gradients are prominent, and vertical (horizontal) edges are rare.

Image content support Υ_{IC} [3] proposes split line positions based on the inverse of the normalized cut between the two created parts of the image.

Label border support Υ_{LB} uses the semantic information from the labeling to penalize lines which split facade elements such as windows, doors and balconies.

Label content support Υ_{LC}: same as Υ_{IC}, but defined over the labeled image.

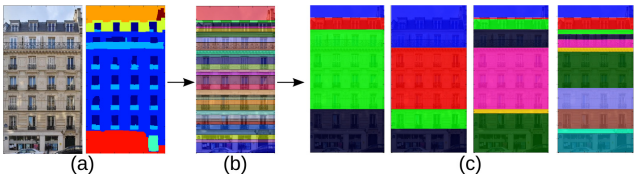


Figure 2. A single image-labeling pair (a) is oversegmented into a large number of slices (b). Randomized segmentations (c) with a varying number of segments create the initial pool of segments.

The next step is to create an oversegmentation of a scope into a predefined number of slices. We are looking for at most K split lines (30 in our experiments), corresponding to peaks in the data support function, which split the scope into $K+1$ slices.

A peak in a vector is defined as a position where the vector has a higher value than its neighbors, and is preceded by a value lower than a threshold τ. By setting τ to a very low value, we initially detect a large number of peaks, most of them affected by the noise in the support function. However, we can smooth the support function by convolving it with a Gaussian window, thus reducing the total amount of detected peaks. The peak detection problem is now posed as the search for the best Gaussian window which produces a number of peaks as close as possible to K.

We solve this problem using binary search, setting the initial lower and upper bound on the Gaussian window size to $\gamma_l = 1$ and $\gamma_u = |\Upsilon(z)|$ respectively. In each step, we convolve the support function with the Gaussian window of size $\gamma_m = (\gamma_u + \gamma_l)/2$. If we detect more peaks than $K-1$, the search is continued by setting $\gamma_l = \gamma_m$. If the number of peaks is smaller, we set $\gamma_u = \gamma_m$. The algorithm finishes when $\gamma_l = \gamma_u$ or the number of generated peaks is equal to K. The result of this step is a set of $K+1$ slices $\mathbf{C}_z$, and K splitting lines $\mathbf{l}_z$, for each scope z. The support function evaluated at the splitting line positions $\Upsilon(z, l_j)$ gives us the strength of each split line, which we use to group the slices into segments.

3.1. Segment proposals

Similar to [6], from a set of slices $\mathbf{C}_z$, we generate many proposal segmentations by varying the number of target segments k from 1 to 20, and running 250 rounds of randomized segmentations for each k. In each round, we perform k-medoid clustering of slices, following the EM pattern. We initialize the algorithm by uniformly sampling k slices as cluster centers. In the M-step, every slice $c_i \in \mathbf{C_z}$ is assigned to the closest cluster center c_m, based on the distance between two slices $\delta(c_1, c_2)$. We define this distance as the maximum of all split line strengths Υ between two slices, which penalizes the creation of segments which span strong split lines. If there is another cluster center c'_m between c and c_m, the distance $\delta(c, c_m)$ is set to infinity. This forces the segmentation to contain only contiguous clusters of slices. In the E-step, the medoids are estimated from the cluster members, by minimizing the sum of distances between elements in one cluster.

Typically, many segments generated in this fashion will appear in more than one randomized segmentation. Segments that are generated the most often are the ones most useful to us, being less sensitive to randomization and the selected number of target segments. Therefore, we weight each unique segment s by the number of times it appears over all randomized segmentations of a single scope, i.e. the frequency of appearance is used as the fitness score w_s of the segment. This simple weighting scheme proved to be sufficient for the task, as we did not observe any improvement by using the more elaborate weighting scheme from [6]. Finally, to reduce the total amount of segments for the subsequent optimization procedure, for each scope we retain $n = 100$ segments with the highest weight as the set of proposals I_z, making sure that it contains at least one complete segmentation of the scope.

We represent each segment s with a vector $\mathbf{h}(s)$ which is a concatenation of two types of features:
Label features $\mathbf{h}_l(s)$. Histogram of labels from the entire segment and from each of its $2x2$ subdivisions. The resulting feature vector captures the coarse distribution of labels.
Image features $\mathbf{h}_i(s)$. Histogram of visual words. Dense SIFT features are extracted from all images, followed by K-means clustering into a codebook of 256 visual words.

Finally, to measure the dissimilarity between two segments, we introduce a distance measure based on the histogram intersection between the feature vectors:

$$d(s, s') = 1 - \frac{\sum_i min(\mathbf{h}_i(s), \mathbf{h}_i(s'))}{\sum_i \mathbf{h}_i(s')} \qquad (2)$$

4. Co-segmentation

The purpose of the co-segmentation step is to find the best subset of segments for each scope, such that they are salient in each scope and consistent across scopes. We follow the same basic algorithm which was introduced for joint segmentation of 3D shapes [6]. In this section, we summarize the basic algorithm, with emphasis on the main differences introduced in our work.

4.1. Pairwise co-segmentation

Given two scopes z_1 and z_2 and their corresponding sets of proposal segments I_1 and I_2, the pairwise co-segmentation searches for the best valid subsets of segments $S_1 \subseteq I_1$ and $S_2 \subseteq I_2$, by maximizing both the quality of individual segmentations, and the consistency between them. A subset of segments is considered valid only if the selected segments cover the entire scope without overlapping. The consistency between two scopes is modeled through two

many-to-one mappings $M_{ij} \subset S_i \times S_j$, from segments in S_1 to segments in S_2, and vice versa. The many-to-one mappings allow us to match scopes with different amount of corresponding parts. Thus, each segment in one scope will be mapped to at most one segment in the other scope. The maximization can be written as

$$\max_{S_1,S_2,M_{12},M_{21}} \sum_{s \in S_1 \cup S_2} r_s w_s + \lambda \sum_{(s,s') \in \{M_{12},M_{21}\}} r_s w_{(s,s')} \quad (3)$$

where the parameter λ (0.1 in our experiments) weighs the relative importance of the segmentation (left) and consistency scores (right). The segmentation score is a normalized sum of segment weights w_s, defined in Sec. 3.1. The normalization factor is the relative size of the segment s in the scope z: $r_s = \text{area}(s)/\text{area}(z)$.

The consistency term is a normalized sum of similarity weights between all segment pairs (s, s') induced by each of the two mappings. The similarity is determined based on the distance measure d between two segments (Sec. 3.1):

$$w_{(s,s')} = exp\left(-\frac{d^2(s,s')}{2\sigma^2}\right) \quad (4)$$

In our experiments, σ is set to half the maximum distance between all pairs of most similar segments.

4.1.1 Integer programming formulation

The maximization problem from Eq. 3 can be reformulated as an integer program [6]. For for every segment $s \in I_i$, an indicator variable x_s is introduced, and defined to be $x_s = 1$ when the segment is selected, and 0 otherwise. Additionally, for every pair of segments $(s, s') \in I_i \times I_j$, the indicator variable $y_{(s,s')}$ is defined to be 1 when this pair is selected in the mapping M_{ij}. The objective function from Eq. 3 is then reformulated as follows:

$$\max \sum_{i \in \{1,2\}} \mathbf{x}_i^T \mathbf{w}_i^{seg} + \lambda \sum_{ij \in \{12,21\}} \mathbf{y}_{ij}^T \mathbf{w}_{ij}^{cor} \quad (5)$$

where $\mathbf{x}_i$ and $\mathbf{w}_i^{seg}$ represent all segment indicators in I_i and their normalized weights. Likewise, $\mathbf{y}_{ij}$ is a binary vector of all pair indicators in $I_i \times I_j$, and $\mathbf{w}_{ij}^{cor}$ are their normalized similarity weights. The first set of constraints in the integer program states that the selected segments must cover the entire scope z_i, without overlapping:

$$\sum_{s \in cover(c)} x_s = 1 \quad \forall c \in \mathbf{C}_{z_i} \quad (6)$$

where $cover(c)$ is the set of all segments that contain slice c. Secondly, each segment of I_i can map to at most one segment in I_j, which itself has to be selected:

$$\sum_{s' \in I_j} y_{(s,s')} \leq x_s \quad \forall s \in I_i \quad (7)$$

$$y_{(s,s')} \leq x_{s'} \quad \forall (s, s') \in I_i \times I_j \quad (8)$$

The integer problem (IP) is obtained by adding the integrality constraints on the x and y variables. Note that our program contains $2n + 4n^2$ integer variables, unlike [6], where adjacency constraints create $2n^4$ additional variables. In our experiments, using the adjacency term did not result in any noticeable improvement. Another difference is that we solve the IP by relaxing the integrality constraints only on the y variables, resulting in a mixed-integer linear program (MILP) with $2n$ integrality constraints. This approach gives us a tighter bound on the IP solution than we would obtain by relaxing all variables. Although the worst-case complexity of this MILP is $O(2^{2n})$, in practice the constraints (7) and (8) allow for a quick convergence of branch-and-cut methods, such as the MOSEK MILP solver in the CVX software package [5]. For $n = 100$, the optimal solution is usually reached within a few seconds on an 8-core machine. The fractional y variables are subsequently rounded to the closest integer, respecting the constraints.

Segment filtering. After the optimization in Eq. 5 has been performed for every pair of scopes, we introduce an additional filtering step. For each scope, we keep only the segments that are selected in at least one of the pairwise optimizations. By discarding the remaining segments, we reduce the computational burden in the subsequent stages.

4.2. Multiway co-segmentation

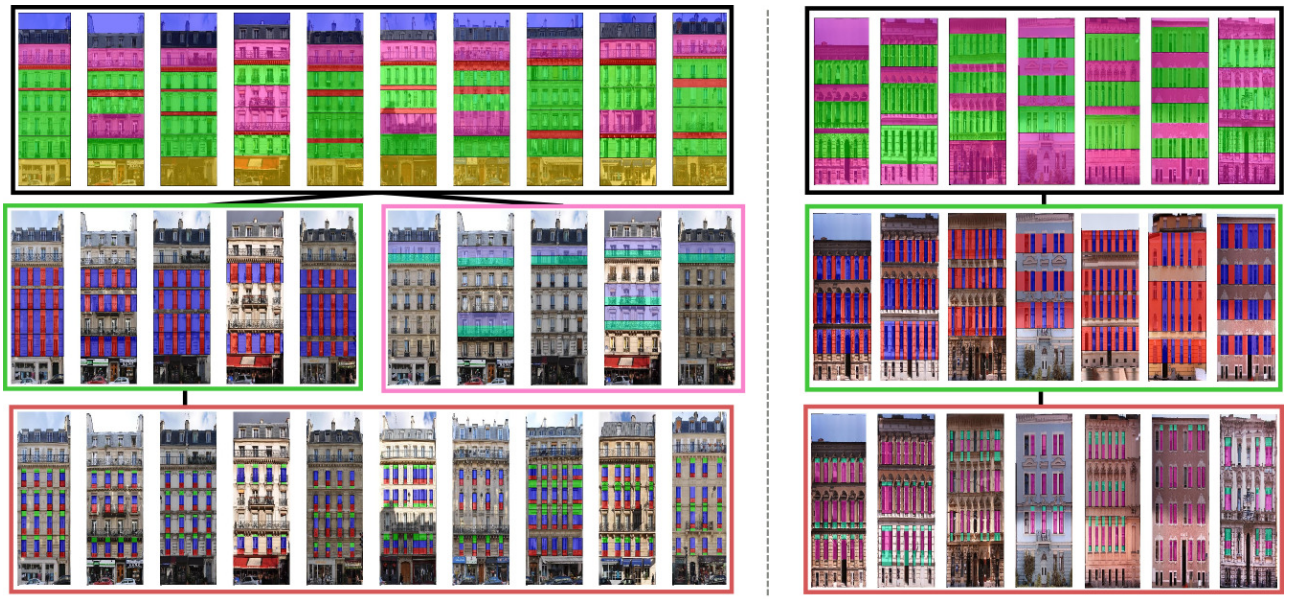
A joint segmentation of all scopes is performed by a generalization of Eq. 5 to N scopes:

$$\max \sum_{i=1}^{N} \mathbf{x}_i^T \mathbf{w}_i^{seg} + \frac{\lambda}{N-1} \sum_{\substack{i=1 \\ i \neq j}}^{N} \sum_{j=1}^{N} \mathbf{y}_{ij}^T \mathbf{w}_{ij}^{cor} \quad (9)$$

Note that the segment filtering step reduces the size of $\mathbf{w}$ vectors compared to Eq. 5. The resulting optimization is again solved with CVX, but due to its higher complexity, we constrain the maximum run-time of the solver to 30 minutes. In our experience, this was enough to reach a solution with a sufficiently small optimality gap. An approximate block-coordinate procedure as in [6] could be employed to increase the speed of optimization, but with no optimality guarantees.

5. Hierarchical co-segmentation

The co-segmentation step results in a flat segmentation of each scope, with mappings between corresponding segments in different scopes. The next step is to create a hierarchical decomposition of each facade. The first step towards this goal is finding subsets of semantically identical elements, which can be segmented jointly in the next step of the hierarchy.



(a) (b)

Figure 3. (Best viewed in color) Hierarchical joint segmentation of facade images: (a) ECP dataset. (b) Gruenderzeit dataset. Each cluster of similar elements in one level of the hierarchy is represented with the same overlay color, which corresponds to the border color in the next level. Due to space restrictions, only a subset of the hierarchy is shown.

5.1. Segment clustering

We represent the segments selected in all scopes by the joint segmentation into one directed, weighted assignment graph $G = (V, E)$. Each node in the set V corresponds to a selected segment s_i, i.e. a segment for which $x_i = 1$. E is a set of directed edges, where node v_i is connected to v_j if the value of the corresponding y_{ij} variable is equal to 1. The weight of each edge e_{ij} is defined in Eq. 4.

Our key observation is that the groups of similar elements will form dense clusters in the graph G. By discovering these clusters, we will also find self-similarities in the input scopes, a feature not modeled by the joint segmentation itself. Thus, our goal is to determine the groups of segments which correspond to each other, within and across scopes. To this end, we run spectral clustering [20] on the graph G. We calculate the normalized graph Laplacian as in [16], and use its eigenvalue decomposition to find the number of clusters κ. Based on the eigengap heuristic, we sort the eigenvalues λ_i in ascending order, and pick κ as $argmax_i(\lambda_{i+1} - \lambda_i)$.

There is a possibility that after the clustering step, two neighboring segments in a scope are assigned to the same cluster, e.g. two wall parts next to each other. In these cases, we simplify the final segmentation by merging those segments into one. However, this must not be done indiscriminately, since there are cases when we expect neighboring segments of the same class (e.g. two floors). We merge two neighboring segments only if their potential merger has a small distance d to the remaining segments in the cluster.

5.2. Hierarchy creation

Initially, N segmentation trees are created, each containing a single root node, corresponding to the whole facade. After performing the co-segmentation and clustering, every tree is augmented with κ children nodes, one for each cluster. In Fig. 3 the trees are merged and the same-cluster segments overlaid with the same color. These segments now become new sets of scopes for the next level of joint segmentation, performed recursively on each cluster. The recursion stops when either the average scope size in the direction of splitting is smaller than a predefined size, or the scopes in the set are uniform in appearance.

The direction of splitting for each node in the hierarchy is determined adaptively. We perform the joint segmentation in both directions, and select the one which gives a more consistent joint segmentation, based on the similarity between a pair of scopes z_i and z_j [6]:

$$w(z_i, z_j) = \mathbf{y}_{ij}^T \mathbf{w}_{ij}^{cor} + \mathbf{y}_{ji}^T \mathbf{w}_{ji}^{cor}, w \in [0, 2] \qquad (10)$$

5.3. Segment synchronization

As we go deeper in the hierarchy, the number of scopes to be jointly segmented increases dramatically (e.g. 20 facades result in ~80 floors and ~400 windows). We can reduce the computational burden for the joint segmentation steps further down in the hierarchy by making the following observation: within one cluster, two scopes with a common parent node are more alike than scopes originating from different parents. Therefore, instead of considering each of

these scopes separately, we perform segment synchronization. First, for each set Ψ of same-cluster scopes originating from the same node, we average their data support functions:

$$\boldsymbol{\Upsilon}^{avg} = \frac{1}{|\Psi|} \sum_{s \in \Psi} \boldsymbol{\Upsilon}(s) \tag{11}$$

and create a *representative* scope by averaging the feature vectors of all scopes in Ψ. This scope replaces all scopes in Ψ during the joint segmentation. Afterwards, the discovered segment borders are back-projected to the original scopes.

Fig. 1 illustrates the process of synchronization: floors a_i are synchronized: they originate from the same facade, so they are segmented in the same way. However, their hierarchies are allowed to differ. As can be seen in the next level of the hierarchy, window tiles in floors a_i are synchronized, but different from the synchronized tiles of floors b_i. This allows us to model local differences, while still correctly capturing the global correspondence.

6. Results

In this section we show some qualitative results of the joint hierarchical segmentation, and evaluate the approach on the task of facade retrieval. Finally, we show how virtual facade layouts can be generated from the induced hierarchy.

6.1. Experimental setup

The main evaluation of our approach is performed on the well-established ECP facades dataset [18], containing 104 images of buildings in Paris. Since all facades in this dataset follow the same Haussmannian architectural style, it is an ideal candidate for our joint segmentation approach. We deal with 7 semantic labels in this dataset, namely $\{window, wall, balcony, door, roof, sky, shop\}$. We also test our approach on a subset of [14], consisting of 30 images in Gruenderzeit style, annotated with a smaller set of labels: $\{window, wall, door\}$. Since we use the output of a supervised facade parsing approach [8] as the input to our approach, we are limited to the analysis of the test set. To cover the entire ECP dataset, we repeat our experiments 5 times, in each fold using different 20 images as the test set, and average the results.

6.2. Hierarchical co-segmentation

In Fig. 3 we visualize the results of our approach on a subset of ECP and Gruenderzeit dataset, respectively. Due to space limitations, here we show only some nodes in the hierarchy, while the full results can be found in the supplementary material.

The first row of Fig. 3 (a) shows the consistent first-level segmentation of the Parisian facades. The coloring corresponds to the different clusters discovered in the data. Our system has automatically detected 6 clusters of similar elements, roughly corresponding to the regions of sky, roof and shop, ledges (red) and two types of floors: regular (green) and floor with running balcony (purple). The consistent segmentation reveals that running balconies usually appear in the second and fifth floor, which is one of the distinguishing properties of Haussmannian architecture. We can also see that the floors with running balconies are split differently than the regular floors in the next level of the hierarchy.

In the Gruenderzeit dataset, we can solely separate floors from the wall regions, since there are only three semantic labels in the annotations. Even in this case, the hierarchical segmentation produces reasonable results, splitting floors into window tiles, which are further subdivided into window and wall regions.

6.3. Facade retrieval

In facade retrieval, a query facade is presented to the system. The query is then compared to a set of known facades based on a pre-defined distance measure. These facades are then re-ranked based on their distance to the query facade, and top K ranked facades are returned as output.

In this section, we demonstrate that our hierarchical representation of facade structures can be used for retrieval of structurally similar facades, rather than those similar in local appearance. We follow the protocol for facade comparison introduced in [21] for the ECP dataset. The gold standard distance δ_{GT} between two facades is defined as the total number of architectural changes: number of floors, number of window columns, position of the running balconies and doors. For each facade in the set, all other facades are re-ranked in the ascending order of δ_{GT}, and the one with the smallest distance is kept as the ground-truth nearest neighbor. Once the ground truth ranking is established, various retrieval methods are evaluated using the Cumulative Match Characteristic (CMC). This measure counts the percentage of correctly retrieved results (gold distance nearest neighbors) in the top-K ranking. When K is equal to the dataset size, all facades are retrieved, resulting in CMC value of 1 for any method.

We test the retrieval results with several different methods, and show the results in Fig. 4. As the first baseline for facade comparison, we create the histograms of semantic labels in each image, and use the histogram intersection measure as the distance δ_L. Second, we use the histograms of dense SIFT features coded into a vocabulary of 256 visual words, to obtain the distance δ_S. A combined distance δ_{LS} is calculated by concatenating the two aforementioned histograms. Additionally, we evaluate the distance $\delta_J = 2 - w(z_i, z_j)$, measuring the dissimilarity of scopes in the first-level joint segmentation step (Eq. 10).

In order to test our hierarchical method, we introduce a new measure for tree distance δ_T. The distance is de-

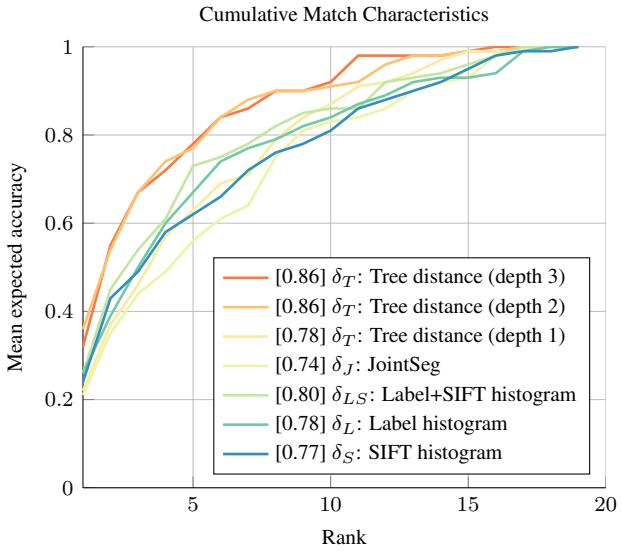


Figure 4. Cumulative Match Characteristics (CMC) for different retrieval methods on the ECP dataset, averaged over 5 folds. Normalized area under curve is shown in square brackets.

fined recursively between two induced hierarchical segmentations h_1 and h_2 as:

$$\delta_T(h_1, h_2) = \delta_N(N_1, N_2) + \sum_{(a_1,a_2)\in A} r(a_1, a_2)\delta_T(h_{a_1}, h_{a_2}) \quad (12)$$

where N_1 and N_2 are the root nodes of the hierarchies, and the set A contains all pairs of children nodes that belong to the same cluster. The relative size r of the children normalizes the sum on the right-hand side to 1. We evaluate our tree distance measure δ_T with varying depth of the hierarchy. For example, the tree distance in the first level of the hierarchy is simply δ_N.

Finally, we define the node distance δ_N between the roots of (sub)hierarchies. We want the distance to be sensitive to the relative order of child nodes. Therefore, we represent the parent node as a string, where each symbol corresponds to the cluster ID of the child node. We match these two sequences using the Smith-Waterman algorithm [17], a dynamic-programming approach traditionally used in bioinformatics to align DNA and protein sequences. We simply define the cost of assigning one element of the sequence to another, and the cost of skipping an element in the sequence (gap cost). We use a constant penalty for mismatched symbols, and size-dependent gap costs, i.e. smaller facade elements are more likely to be skipped.

The retrieval results are shown in Fig. 4. As expected, the combination of the label histograms and SIFT features δ_{LS} outperforms either of the stand-alone methods. On the other hand, using the joint segmentation distance δ_J results in poor performance, due to two main limitations. First, this distance models only the first level of the hierarchy, and does not capture finer structural differences. Second, many-to-one mappings disregard the relative frequency of elements, information which is not lost in histogram-based methods. For example, a perfect mapping can be found between a facade with 10 floors and a facade with 1 floor, resulting in a low δ_J. Our tree distance δ_T does not suffer from these issues. Even when using only one level of the hierarchy, where the only discovered elements are floors, our distance provides better discriminative power than δ_J, due to the ordering information. By using the second level of the hierarchy, we obtain even better results, due to the correct modeling of window tiles. We do not observe any significant improvement in retrieval by using the third level, where the hierarchy models only local differences. It is important to note, however, that using the deeper levels of the hierarchy does not degrade the results, even if similarities not captured by the ground truth are found.

6.4. Facade synthesis

The state-of-the-art methods for facade structure extraction [9, 21] have shown that procedural split grammars can be inferred from clean, ground-truth labelings. The grammars can subsequently be used to generate new facades by changing the parameters of the grammar. Similar to our approach, these methods split the facades in alternating vertical and horizontal directions, where each split is represented as one rule in the procedural grammar:

$$X^\alpha \rightarrow split(dir)\{r_1^\alpha : b_1^\alpha | r_2^\alpha : b_2^\alpha ... | r_n^\alpha : b_n^\alpha\} \quad (13)$$

where X^α represents the root node and dir the direction of splitting. With $\mathbf{b}^\alpha = \{b_i^\alpha\}$ and $\mathbf{r}^\alpha = \{r_i^\alpha\}$ we denote the set of children and the vector of their relative sizes.

However, both of the aforementioned methods create the structural decomposition of each facade separately, and then attempt to merge the inferred decompositions to obtain a joint grammar. On the other hand, our joint approach immediately creates consistent trees, albeit noisier due to the usage of imperfect input data. We create a procedural grammar from the ECP test set by transforming each of the 20 hierarchies into a set of procedural rules, which are then aggregated. In the next step, we perform the same production rule inference as in [21]. This process merges all similar rules which have the same form (but different size vectors) into one. The initial facades are now re-created by selecting the appropriate size vector $\mathbf{r}^\alpha$. New facades in the similar style can be obtained by fitting a multivariate Gaussian distribution to the set of size vectors in each rule, and then sampling from this distribution. Fig. 5 shows some virtual facade layouts sampled in this fashion. We export these layouts to CityEngine [4] and create 3D buildings by automatic placement of architectural elements from a 3D library at the positions defined by the layouts.

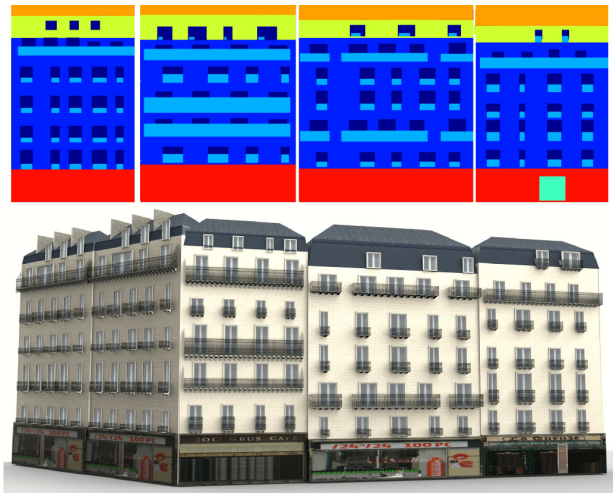


Figure 5. Synthesis of virtual facade layouts. The sampled layouts are represented as procedural split grammars and converted into 3D models with CityEngine.

7. Conclusion

We have introduced a system for higher-level understanding of building facades through a joint hierarchical decomposition approach. Unlike most previous facade structure learning approaches, which rely on user interaction or ground truth annotations, we show that facade structure can be induced even by using noisy inputs. Our key observation is that consistent hierarchies can be created by performing a joint segmentation approach on each level of the hierarchy. The joint segmentation allows us to produce stable, consistent segmentations across images, despite the noise present in the input data. Moreover, the induced hierarchies are a meaningful semantic representation of the building facade, which we demonstrate on the task of structural facade retrieval. We also convert the hierarchies into procedural grammars and use them to sample new facade designs, which respect the layout of the original facades.

In future work, we plan to further reduce the dependency on image labelings, and induce structure solely from images. Furthermore, feedback can be added in the hierarchy construction, to reduce the effect of error propagation to the lower levels of the hierarchy. Finally, we will investigate how to model different kinds of structural decompositions, such as repetition and symmetry.

References

[1] F. Bao, M. Schwarz, and P. Wonka. Procedural facade variations from a single layout. *ACM Trans.Graph.*, 32(1), 2013. 2

[2] A. Cohen, A. G. Schwing, and M. Pollefeys. Efficient structured parsing of facades using dynamic programming. In *CVPR*, 2014. 1, 2

[3] D. Dai, M. Prasad, G. Schmitt, and L. Van Gool. Learning domain knowledge for facade labeling. In *ECCV*, 2012. 1, 2

[4] Esri. CityEngine. http://www.esri.com/software/cityengine, 2013. 7

[5] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming. http://cvxr.com/cvx, Mar. 2014. 4

[6] Q. Huang, V. Koltun, and L. Guibas. Joint shape segmentation with linear programming. *ACM Trans. Graph.*, 30(6):125:1–125:12, 2011. 2, 3, 4, 5

[7] J. Lin, D. Cohen-Or, H. R. Zhang, C. Liang, A. Sharf, O. Deussen, and B. Chen. Structure-preserving retargeting of irregular 3d architecture. *ACM Trans.Graph.*, 30(6), 2011. 2

[8] A. Martinović, M. Mathias, J. Weissenberg, and L. Van Gool. A three-layered approach to facade parsing. In *ECCV*, 2012. 1, 2, 6

[9] A. Martinović and L. Van Gool. Bayesian grammar learning for inverse procedural modeling. In *CVPR*, 2013. 2, 7

[10] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural modeling of buildings. In *SIGGRAPH*, 2006. 1

[11] P. Müller, G. Zeng, P. Wonka, and L. J. V. Gool. Image-based procedural modeling of facades. *ACM Trans. Graph.*, 26(3):85, 2007. 1

[12] P. Musialski, M. Wimmer, and P. Wonka. Interactive coherence-based facade modeling. *Computer Graphics Forum*, 31(2pt3):661–670, 2012. 2

[13] X. Ren and J. Malik. Learning a classification model for segmentation. In *ICCV*, volume 1, pages 10–17, 2003. 2

[14] H. Riemenschneider, U. Krispel, W. Thaller, M. Donoser, S. Havemann, D. W. Fellner, and H. Bischof. Irregular lattices for complex shape grammar facade parsing. In *CVPR*, 2012. 6

[15] C.-H. Shen, S.-S. Huang, H. Fu, and S.-M. Hu. Adaptive partitioning of urban facades. *ACM Trans.Graph.*, 30(6):184, 2011. 1, 2

[16] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE TPAMI*, 22(8):888–905, Aug 2000. 5

[17] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981. 7

[18] O. Teboul, I. Kokkinos, L. Simon, P. Koutsourakis, and N. Paragios. Parsing facades with shape grammars and reinforcement learning. *IEEE TPAMI*, 35(7):1744–1756, 2013. 1, 2, 6

[19] O. van Kaick, K. Xu, H. Zhang, Y. Wang, S. Sun, A. Shamir, and D. Cohen-Or. Co-hierarchical analysis of shape structures. *ACM Trans.Graph.*, 32(4), 2013. 2

[20] U. von Luxburg. A tutorial on spectral clustering. *CoRR*, abs/0711.0189, 2007. 5

[21] J. Weissenberg, H. Riemenschneider, M. Prasad, and L. Van Gool. Is there a procdural logic to architecture? In *CVPR*, 2013. 2, 6, 7

[22] P. Wonka, M. Wimmer, F. X. Sillion, and W. Ribarsky. Instant architecture. *SIGGRAPH*, 22(3):669–677, 2003. 2

[23] H. Zhang, K. Xu, W. Jiang, J. Lin, D. Cohen-Or, and B. Chen. Layered analysis of irregular facades via symmetry maximization. *ACM Trans.Graph.*, 32(4):121:1–13, 2013. 2