Automation in Construction 72 (2016) 388–396

Contents lists available at ScienceDirect

Automation in Construction

journal homepage: www.elsevier.com/locate/autcon

Irregular architectural layout synthesis with graphical inputs

Hao Hua

[a]Key Laboratory of Urban and Architectural Heritage Conservation (Southeast University), Ministry of Education, China
[a]School of Architecture, Southeast University, 2 Sipailou, Nanjing 210096, China

ARTICLE INFO

Article history:
Received 3 November 2015
Received in revised form 19 September 2016
Accepted 23 September 2016
Available online 30 September 2016

Keywords:
Automated layout design
Pattern
Topology
Graph matching

ABSTRACT

This paper presents a method for automatically constructing irregular floor plans. Given image patterns, the program produces a variety of layouts that satisfy users' geometric and topological requirements. The program extracts irregular regions from images via statistical region merging. It employs subgraph matching and simulated annealing to construct topologically feasible layouts. Methods for measuring similarities between the desired templates and irregular rooms are also reported. Applications using raster images and vector graphics are tested. Both applications generate feasible floor plans whose appearances resemble the input patterns. The results indicate that layout automation is valid without an orthogonal framework in the plane. Irregular layout synthesis allows architects to instantly implement visual preferences in early design stages.

1. Introduction

Digital tools for architects come in two forms: The first is the various types of CAD software, from the early Sketchpad [1] to the most recent BIM (Building Information Modeling; many of the recent CAD software including Autodesk Revit falls into this category) solutions. Such software demands enormous amounts of user input but provides no automation from the computer. The other involves more experimental approaches that automate the process of design. Such software needs few specifications (typically, a list of the square footages of the rooms and their adjacencies) as the user input but requires complex automated reasoning. This work is related to both approaches. On the one hand, it invites the designer to provide graphical input; on the other hand, it creates a set of plausible layouts that exhibit the patterns in the given image. As such, this tool is an alternative to manual drafting for schematic floor plan design.

Layout automation shows promise for supporting the activities of design. But it has rarely, if ever, been taken up in practice. Two reasons may explain this. First, current tools only allow architects to input numerical/logical specifications, instead of high-level preferences. However, many architects are primarily concerned with the layout pattern. For instance, as shown in Fig. 1, architect Frank Wright created three plans, each with a distinct pattern, for the same design program (often represented by a graph such as in Fig. 1; refer to [2] for a deeper investigation). This research follows this line by letting the designer feed an image with arbitrary patterns into the automated tool. Second, most automation approaches have been confined to right-angle polygons. In contrast, this work focuses on layouts with complex shapes.

Before the 1970s, groundbreaking researchers ascribed historical significance to the optimal locations of facilities. Using the principles of linear and non-linear programming, these pioneers modeled the floor plans of rectilinear rooms and defined constraints and objectives of floor plans. Armour and Buffa [3], Whitehead and Eldars [4], Seehof et al. [5], and others explored the optimal allocations of grid cells; more precisely, they assigned labels (e.g., living room and kitchen) to all cells in a grid and enumerated the assignments so as to minimize the cost function. In one distinct approach, researchers read rooms as "floating rectangles" and searched for their optimal positions. Krejcirik [6] first took this approach, while Flemming and Chien [7] followed up on this with their SEED system. Liggett [8] reviewed more works of layout synthesis in detail. Most early works were intended for industry applications and featured well-defined criteria; however, they did not raise sufficient interest from architects.

More recently, researchers have adopted sophisticated optimization techniques to generate more complex and appealing layouts. In doing so, they have widely employed evolutionary algorithms. Rosenman [9] represented grid-based rooms with edge vectors and encoded potential layouts as sets of symbols called "genotypes". The genetic algorithm evolves populations of genotypes (whose phenotypes are the candidate layouts) until it discovers high fitness layouts. The fitness is deemed to be a measure of the rooms' geometric qualities and adjacency.

E-mail address: whitegreen@163.com.

http://dx.doi.org/10.1016/j.autcon.2016.09.009
0926-5805/

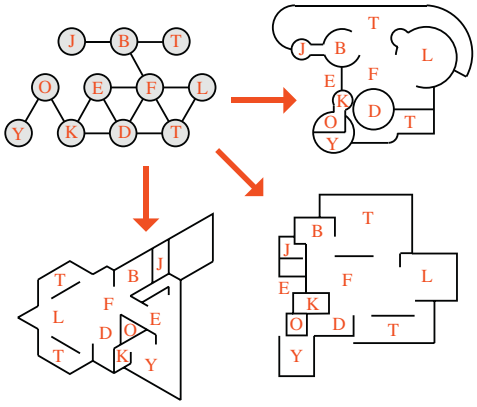


Fig. 1. Three manually designed floor plans implemented from the same design program (presented by a graph).

Rodrigues et al. [10] incorporated a design program with constraints and objectives of floor plans into their fitness function. Robust optimization is carried out by an evolutionary algorithm enhanced by local searches. Michalek and Papalambros [11] proposed a general method of interactive optimization; they modeled a layout as a group of related units whose locations and dimensions are optimized by defined objectives. During the optimization process, participants can modify the objectives, constraints, and units. Michalek and Papalambros used both local optimization (sequential quadratic programming) and global optimization (simulated annealing) to produce feasible plans. Merrell et al.'s residential layouts optimization [12] is a practical application of layout automation. Wall segments represent right-angle rooms (including concaves). A metropolis algorithm (similar to simulated annealing) is employed to minimize the cost function of the room dimensions and the accessibility.

One strategy for expanding single story buildings into multi-story ones involves stacking single-story floors on top of each other with various constraints on vertical circulation (with consistent stair and elevator positions across all levels). For examples, see [12,13]. Loose constraints on vertical circulation usually contribute to rich varieties [14]. Another strategy is directly organizing 3D units from the outset [15–18].

There are two recent trends. First, local heuristics accelerate the global optimization. For instance, the stochastic hill climbing of [10] is a local technique in the search processes; while the wall sliding and room swapping of [12] are local movements in the entire layout. Self-organization [18,19] is another distinct method. A plan is regarded as a multi-agent system in which elements (such as a room) interact with each other. The resulting solution is a natural consequence of enormous interactions between local behaviors [18]. Such self-organizing approaches are competitive with traditional optimization paradigms.

Second, automation programs are usually limited to rectangles or right-angle polygons. A few exceptions exist; for example, [18,20] generated plans based on Voronoi tessellations. Menges [16] extended the 2D Voronoi diagrams into 3D structures. Doulgerakis [21] took a novel approach by defining a few splitting operators, which partition a rectangular block in certain sequences that correspond with particular layouts. Doulgerakis' genetic programming searches over the splitting sequences to find layouts of desirable quality.

In summary, an approach that can generate layouts of freeform shapes without imposing a particular data structure (e.g., a grid) has not yet been proposed, and previous automation tools do not deal with a designer's preferences of patterns. Graphical communication between the designer and the computer program, which subsequently requests the tools to recognize and produce unpredictable, irregular shapes, is of great importance. This research makes two contributions. First, it manipulates highly irregular shapes (non-orthogonal and non-convex) without a particular structure, such as a grid or Voronoi tessellation, in the plane. Second, it allows the designer to impose patterns from an arbitrary image onto the layout so that the final plan follows the input pattern.

2. Irregular layouts with graphical inputs

2.1. Method overview

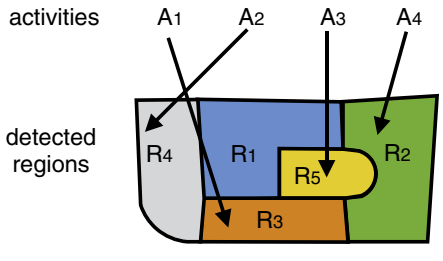
This research decomposes the layout into two parts: the individual rooms and the relationship between them. More precisely, this research focuses mainly on the evaluation of the irregular shape of each room and the adjacencies between each pair of rooms. When a client asks for direct access between a pair of rooms, the solution is to place them next to each other so that the two rooms share a common edge on which the architect can design the doorway in later stages. This work features the following elements, which will be further explained in Subsections 2.2 and 2.3.

I. **Geometric requirements of activities**. Traditionally, the client presents a design program with the activities and their configurations. Let A_i be the geometric requirement (an elementary requirement of a room is its size) of the ith room (activity). Set $A = \{A_1, A_2, \ldots A_N\}$ specifies all the rooms.

II. **Regions detected from image**. The essential procedure is associating each activity to one region, i.e., to create a map $A \rightarrow R$. This is equivalent to deriving an ordered set $R = \{R_1, R_2, \ldots R_N, R_{N+1} \ldots R_M\}$, where the first N regions are sorted according to the order of set A (Fig. 2).

III. **Cost function of topology**. A matrix P of $N \times N$ binary values represents the desired adjacencies between the activities. $P_{ij} = 1$ means that room R_i and room R_j must be connected, while $P_{ij} = 0$ means the connectivity between the two is disregarded. Next, a matrix Q of the actual adjacencies between the regions R is constructed, and the program examines how P differs from Q (Fig. 2). The cost function of the topology is given by

$$T(A \rightarrow R) = \sum_{ij} P_{ij}(1 - Q_{ij}) \tag{1}$$



	A1	A2	A3	A4
A1		1	0	1
A2			1	0
A3				1
A4				

Matrix **P**: desired adjacencies between the activities.

	R3	R4	R5	R2
R3		1	1	1
R4			0	0
R5				1
R2				

Matrix **Q**: adjacencies between the regions, the order corresponds to the order of the activities.

Fig. 2. Top: mapping activities to the regions. Bottom: adjacency matrix of the activities and adjacency matrix of the regions. Here, the latter differs from the former, which means that the layout does not satisfy the topology requirements.

Minimizing the cost means that the region pairs which are meant to be adjacent must be adjacent.

IV. **Cost function of geometry**. After labeling the regions with activities, the program evaluates each region by the geometric cost function $g(R_i, A_i)$. Subsequently, the geometric cost function for the whole layout is defined as $G(A \to R) = \sum_i g(R_i, A_i)$.

V. **Optimizer**. The cost functions drive the optimization process to produce geometrically and topologically feasible layouts.

2.2. Layout synthesis with raster images

2.2.1. Program summary

Inputs:

1. $A = \{A_1, A_2, \ldots A_N\}$, the square footage of each activity (room);
2. matrix P: the desirable adjacencies between the rooms;
3. an image.

Process:

1. detect regions $(R_1, R_2, \ldots R_M)$ from the image;
2. optimize the assignment $A \to R$, concerning both the adjacencies and the room shapes.

Outputs: the labeled regions as rooms (e.g., Fig. 3 left).

2.2.2. Algorithms

Region detection. Given a raster image, the program detects a set of regions according to the pixel colors, and subsequently the segmentation reflects the patterns in the image. Statistical region merging [22] subdivides the image into a set of disjunct regions $(R_1, R_2, \ldots R_M)$, as illustrated in Fig. 4. The algorithm contains control parameters associated with the number of resulting regions. Each detected region is further divided by a straight line into two parts. A list of candidates, instead of the best subdivision, is preserved for each region, as shown in Fig. 4. At a later stage, an activity can be mapped either to the original region or to one of its subdivided parts. The cost function for the subdivision is:

$$\frac{(c_1)^2}{a_1} + \frac{(c_2)^2}{a_2} \tag{2}$$

where c_1 and a_1 represent the perimeter and area of the first subdivided part, respectively; and c_2 and a_2 represent perimeter and area for the second.

Cost function of geometry. Geometric metrics of rectilinear rooms (e.g., aspect ratio) are not valid for highly irregular shapes. The program takes two steps to assess an irregular blob. First, it computes the depth (shortest distance to the boundary of the region) of each pixel in the region. Then the center of the region (analogous to the center of mass) is defined as:

$$c = \left(\frac{\sum d_i x_i}{\sum d_i}, \frac{\sum d_i y_i}{\sum d_i} \right) \tag{3}$$

where d_i denotes the depth of the ith pixel in the region; and x_i, y_i are the coordinates of the ith pixel. Second, the algorithm places a circle (or a Gaussian kernel) at the center of the region (Fig. 5). The area of the circle is equal to the desirable square footage of the corresponding activity. Then the irregular region can be evaluated by:

$$g(R_i, A_i) = \frac{|R_i \uparrow \Phi|}{|R_i| + |\Phi|} = \frac{|R_i \uparrow \Phi|}{|R_i| + A_i} \tag{4}$$

where Φ denotes the set of pixels inside the circle. Symbol $||$ means cardinality, so $|\Phi|$ is equal to the number of pixels inside the circle. $\uparrow$ is the alternative denial, e.g., the **NAND** operation between two 2D regions. The cost function favors compact shapes for a better accommodation of inhabitant activities. An alternative is to use a Gaussian kernel instead of a rigid circle (Fig. 5), in which case the cost function becomes:

$$g(R_i, A_i) = \frac{A_i + \sum_j h(j) k \exp\left(-\frac{||v_j - c||^2}{2\sigma^2}\right)}{|R_i| + A_i}, \quad h = \begin{cases} -1, v_j \in region \\ 1, otherwise \end{cases} \tag{5}$$

where v_j denotes the position of the jth pixel in the plane. $||v_j - c||$ represents the Euclidean distance between the jth pixel and the center of the region. The choices of the constants k and σ must be subject to the equation:

$$\sum_j k \exp\left(-\frac{||v_j - c||^2}{2\sigma^2}\right) = A_i \tag{6}$$

Cost function of topology. By checking whether a pair of regions shares a common edge (which must be wide enough for a doorway), the actual adjacencies Q between the regions (and the subdivided ones) are constructed. Hence, the cost function $\sum_{ij} P_{ij}(1 - Q_{ij})$ can give the topological error of the whole layout.

Optimizer. The critical procedure is associating each activity with a region, i.e., searching for the optimal map $A \to R$. Notice that an activity could be mapped either to an original region or to its subdivided component. The cost function is as follows:

$$F(A \to R) = W_t \sum_{ij} P_{ij}(1 - Q_{ij}) + W_g \sum_i g(R_i, A_i) \tag{7}$$

W_t and W_g are the weights for the adjacency cost and for the geometric cost, respectively. Nevertheless, the program only accepts the layouts whose topology error is zero at the end of optimization. Simulated annealing is used for optimization. For each iteration, there are two kinds of moves: changing the assignment of one

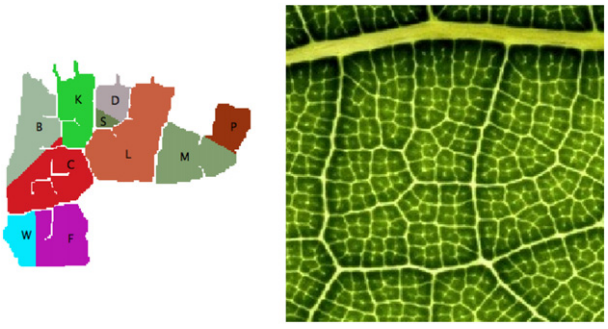


Fig. 3. An irregular layout computed for the design specifications in Table 1. Left: the synthesized plan. Right: the input image. Whenever two rooms are required to be adjacent, the program places the pair closely together. These schematic plans do not specify doorways.

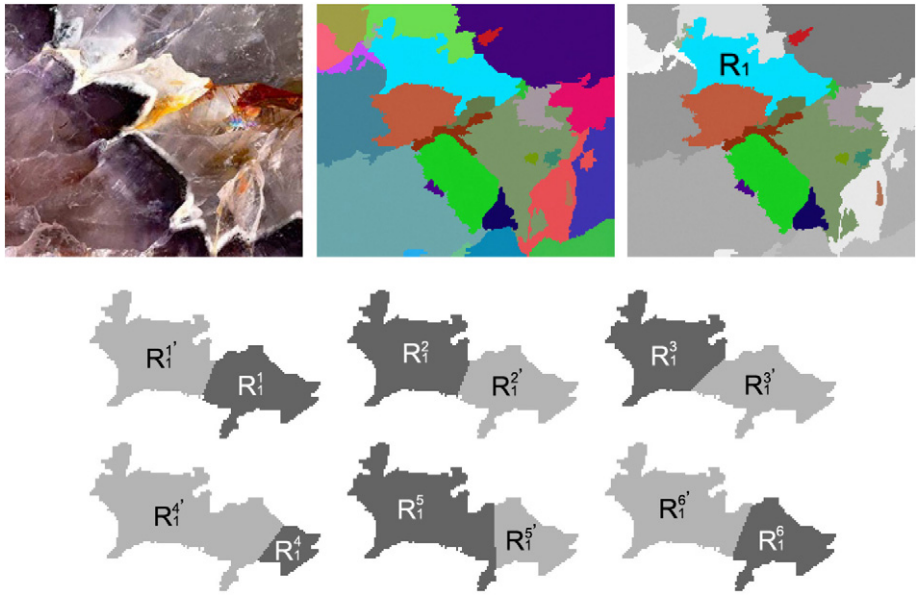


Fig. 4. Region detection and subdivision. Top left: the input raster image. Top middle: detected regions. Top right: removal of regions touching the boundary (gray). The program subdivides the detected regions. For instance, region R_1 is divided into two parts, and a list of the six best subdivisions is preserved for the region.

activity (to an unlabeled region) or swapping two assignments. The annealing scheme is planned as:

$$p = \begin{cases} 1 & \text{if } e' < e \\ \frac{\exp(e-e')}{ab^t} & \text{otherwise} \end{cases} \quad (8)$$

where p: the probability of adopting a new map $A \rightarrow R$; e: the error of current map; e': the error of the new map; and t: the current number of iteration. The term ab^t (a and b are constants) represents the "temperature" of the annealing process.

2.3. Layout synthesis with parametric vector graphics

2.3.1. Program summary

Inputs:

1. $A = \{A_1, A_2, \ldots A_N\}$, the required dimension (width and depth) of each activity (rooms);
2. matrix P: the desirable adjacencies between the rooms;
3. a set of parametric 2D components $V = \{v_1, v_2, \ldots v_L\}$ as wall segments (e.g. Fig. 7 (a)).

Process:

1. detect regions $(R_1, R_2, \ldots R_M)$ from the vector graphics;
2. compute a valid assignment $A \rightarrow R$ satisfying adjacency requirements;
3. optimize the shapes of labeled regions.

Outputs: the labeled regions as rooms(e.g. Fig. 7 (e)).

Modernist architects often define unique shapes of wall segments and create novel overlapping and intersections(Fig. 6). To reflect this scenario, in the second application the user defines or selects the components of the partition walls, e.g., quadrilaterals, crosses, and T shapes (Fig. 7 (a)), and then the program computes their positions and orientations to yield a layout that meets the dimension and topology requirements. Let v_i be the parameters specifying the ith segment. The set $V = \{v_1, v_2, \ldots v_L\}$ represents all the segments lying in the plane. A set of regions (enclosed with the segments) R can be

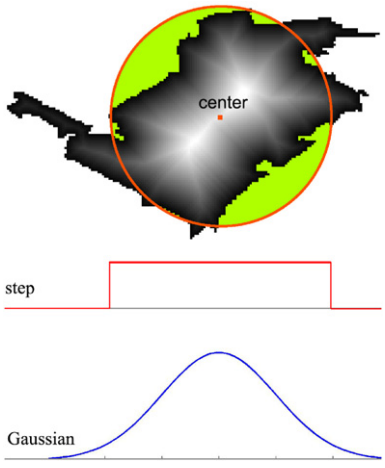


Fig. 5. Evaluation of irregular region. Either the circle (step function) or the Gaussian kernel is employed.

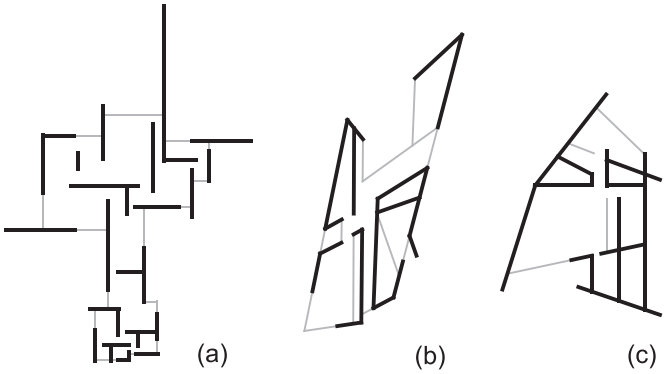


Fig. 6. Three manually designed floor plans featuring novel compositions of wall segments. Architects: (a) Ludwig Mies van der Rohe, (b) Zaha Hadid Architects, (c) Daniel Libeskind.

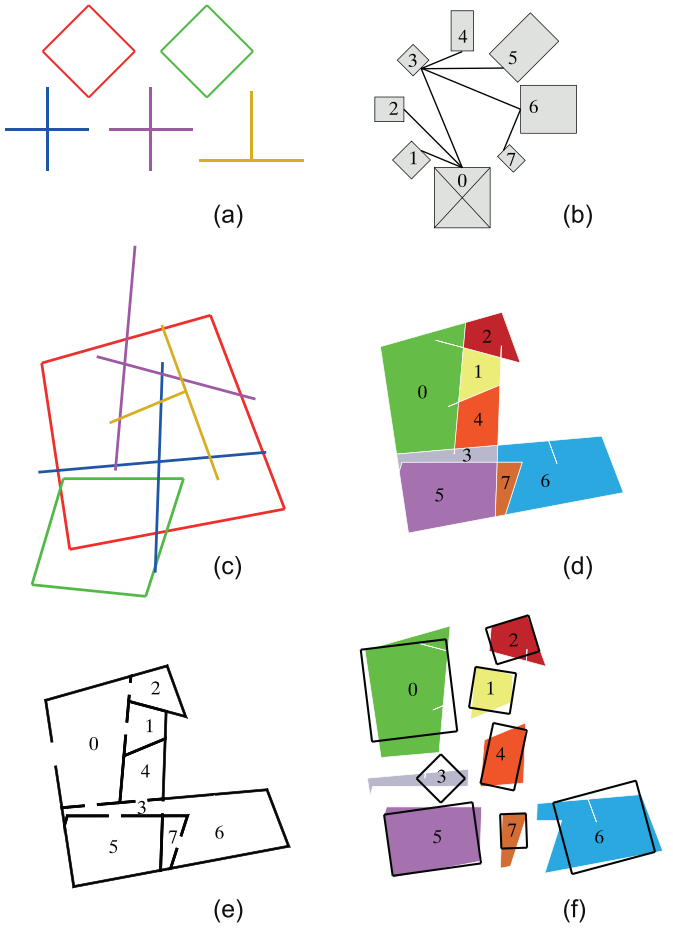


Fig. 7. The elements of the layout: (a) the 2D components defined by the designer, (b) the given dimensions of each room and the adjacencies, (c) the solution represented by segments, (d) the solution represented by polygons (labeled), (e) the refined layout, and (f) the differences between the template rectangles and the polygons.

detected. The procedure for optimizing over V and over $A \rightarrow R$ is carried out as follows:

Algorithm 1.

Initialize segments V.
DO {
  move segments V.
  detect regions R.
  map activities to regions[1] ($A \rightarrow R$), **CONTINUE** if fails.
  optimize the geometries of labeled regions [2]:
    initialize temperature
    FOR N times {
      move segments V
      accept or reject the move according to the probability (10)
      decease temperature
    }
} **WHILE** (geometry optimization fails)

2.3.2. Algorithms

Region detection. Much of the literature on computational geometry (e.g. [23]) discusses the detection of enclosed polygons from a set of line segments. The detected polygons R are regarded as a function of the segments V.

Constructing topology feasible layouts. The subgraph matching algorithm finds a valid mapping from activities A to polygons R. Topology satisfaction means finding a subgraph of Q (actual adjacencies of the detected polygons) that is identical to P (desirable adjacencies of the predefined activities); i.e., which makes the value of cost function (1) zero. Gold and Rangarajan's algorithm [24] is simple and efficient for subgraph matching:

Algorithm 2.

Inputs:
Q, $M \times M$ matrix of binary entries: the adjacencies of detected polygons. $M > N$;
P, $N \times N$ matrix of binary entries: the desirable adjacencies between predefined activities.
Outputs:
$t_j \in Z, 1 \le j \le N$: the index of the corresponding node in Q for the jth node in P.

Initialize A ($M+1 \times N+1$ matrix of float entries).
$\beta \leftarrow \beta_0$
WHILE (iteration$< I_1$) **DO** {
  For $i \leftarrow 1$ to M, $j \leftarrow 1$ to N
    $\Delta_{ij} \leftarrow \sum_{kl}(P_{ik}Q_{jl})A_{ij} + \lambda_{ij}$
  For $i \leftarrow 1$ to M, $j \leftarrow 1$ to N
    $A_{ij} \leftarrow exp(\beta\Delta_{ij})$
  WHILE (iteration$< I_2$) **DO** {
    For $i \leftarrow 1$ to M+1, $j \leftarrow 1$ to N
      $A_{ij} \leftarrow \frac{A_{ij}}{\sum_j A_{ij}}$ //normalize rows
    For $i \leftarrow 1$ to M, $j \leftarrow 1$ to N+1
      $A_{ij} \leftarrow \frac{A_{ij}}{\sum_i A_{ij}}$ //normalize columns
  }
  $\beta \leftarrow \beta\eta$
}
For $j \leftarrow 1$ to N
  $t_j \leftarrow \underset{i}{\text{argmax}}(A_{ij})$
Verify the found subgraph by checking whether $P_{ij} = Q_{(a_i)(a_j)}$ for all i, j.

In the above pseudocode, I_1, I_2, β_0, η, and λ_{ij} are constants. Constant η controls how fast the learning rate β decreases. λ_{ij} are constant values near zero for symmetry breaking (again multiple isomorphic subgraphs). The algorithm is guaranteed to yield a valid subgraph if one exits.

Cost function of geometry. After meeting the topology requirements, the shape of each labeled polygon can be evaluated individually. Such evaluation requires two steps. First, the program overlaps the template rectangle A_i onto the polygon, and maximizes the overlapped area between the two (some results are depicted in Fig. 7 (f)). A gradient-based algorithm is employed to identify the location and the orientation of the template rectangle. Second, the program computes the geometric cost of the polygon by:

$$g(R_i, A_i) = W_1|a(R_i) - a(A_i)| - W_2 m(R_i, A_i) \tag{9}$$

where $m(R_i, A_i)$ denotes the maximum overlapping area between R_i and A_i, and $a()$ is the area function.

Geometry optimizer. The geometry optimizer employs a simulated annealing scheme to reduce the geometric cost $f(V) = \sum_i g(R_i, A_i)$. Slightly modifying the values of V, or in other words, slightly moving the segments of the layout, results in a proposal move. The optimizer temporarily filters out the moves that break the valid adjacencies established in the main loop. The annealing scheme is defined as

$$p = \begin{cases} 1 & if \quad f(V) > f(V') \\ \frac{\exp(f(V) - f(V'))}{a - b^t} & otherwise \end{cases} \tag{10}$$

where p: the probability of adopting a move; $f(V)$: the error of the current state of the segments V; $f(V')$: the error after a move; t: the current number of iteration; and a and b are constants.

3. Results and discussion

3.1. Tests with raster images

3.1.1. Geometry and topology validation

A residential program with ten rooms (Table 1) is tested. The program transforms input images into schematic layouts (Fig. 3).

Table 1
Residential program I. In addition, the table lists both the desired areas of the activities and the actual sizes of the generated rooms.

	F	W	B	C	K	L	D	M	S	P	Desirable size (m^2)	Actual size (m^2)
Foyer (F)				1							10	5.5
Bath (W)				1							4.4	2.5
Bedroom (B)				1							8.0	8.0
Corridor (C)	1	1	1		1	1					9.0	13.7
Kitchen (K)				1			1		1		10.0	11.0
Living (L)				1			1	1			19.0	25.0
Dining (D)					1	1					11.0	12.4
MBedroom (M)						1				1	10.5	6.7
Storage (S)					1						3.0	1.5
Porch (P)								1			4.8	3.5

The adjacencies between the resulting regions are consistent with the topological requirements. For example, Kitchen (K) is to be connected with Corridor (C), Dining Room (D), and Storage (S) (see Table 1); the floor plan meets these conditions (Fig. 3).

The geometric qualities of each room can be validated in two ways. First, formula (4) can measure the difference between the desired template and the actual room. Table 3 lists a series of results. In Table 1, the rooms' desired and actual sizes are compared; this shows that the rooms are close to expectations. Second, as aesthetic and subjective preferences are of great importance to architects, this topic is discussed in the following subsection.

3.1.2. Patterns and diversity

When a user feeds images of regular patterns into the program, the resultant layouts are, not surprisingly, regular (Fig. 8 (a)). In contrast, complex images lead to highly irregular layouts (Fig. 8 (b),(c),(d)). The resulting plans substantially exhibit the patterns of the given image. This feature enables designers to address visual preferences with images and to see and evaluate the results immediately, which leads to a loop of image selection-layout automation-evaluation.

The outputs of the program are much more diverse than those of rectilinear layouts. The diversity of this system can be interpreted as follows: Given the same design program, the generated plans vary a good deal with the input images. This diversity is essential to early design stages, though researchers have not yet addressed it. An architect's preferred layout at this stage is a rough prototype to which more details will be added in future design stages.

Table 2
Residential program II. In addition, the table lists both the desired areas of the activities and the actual sizes of the generated rooms.

	F	W	B	C	K	L	D	Size (m^2)
Foyer (F)				1				10
Bath (W)				1				4.4
Bedroom (B)				1				9.3
Corridor (C)	1	1	1		1	1		9.0
Kitchen (K)				1			1	10.0
Living (L)				1			1	19.0
Dining (D)					1	1		11.0

3.1.3. Implementation and performance

This program is implemented in Java. Typically, the input image resolutions are 600 × 400. The statistical region merging parameters regulate the number of regions to be detected. In addition, the algorithm crops the input image if there are too many regions. The program sets the resulting regions to 2–3 times more than the number of regions that are expected in the final plan.

Both pre-processing and post-processing can deal with image noise. Pre-processing (e.g., a Gaussian filter) is simply making the input image smoother. As a result, the detected regions will be smoother. Post-processing produces smooth edges by Gaussian blur and blob detection (Fig. 9).

The simulated annealing algorithm consists of about 1000 iterations. The annealing procedure takes a few seconds on an Intel Core i5 clocked at 2.4 GHz. Table 3 lists the iterations, the errors, and the runtimes of a series of tests. The results imply that one iteration takes roughly 1 ms.

3.2. Tests with vector graphics

3.2.1. Geometry and topology validation

The computer program yields a variety of plans represented by wall segments according to the given design program (Fig. 10). First, it guarantees that the connectivity between the generated rooms is identical with the desired connectivity. For instance, the graph in the first row of Fig. 10 demands that room 3 must be connected with rooms 0, 4, 5, and 6; this is the case in the four generated plans.

The geometric quality of each room are measured using the difference between the rectangular template and the generated shape, according to the cost function (9). A more comprehensive metric is $m(R_i, A_i)/A_i$; this is the ratio of the overlapping area (between the template rectangle and the actual room) to the template's area. Table 4 shows a series of results.

Table 3
Tests with raster images: runtime, and geometric error of each room. Room indices are identical to Table 1.

F	W	B	C	K	L	D	M	S	P	Err sum	Iteration	Runtime(s)
0.39	0.4	0.66	0.66	0.63	0.56	0.98	0.42	1	0.27	5.97	840	1.01
1.63	0.64	0.56	0.35	0.91	0.69	0.58	0.44	1.16	0.35	7.3	840	1.17
0.99	0.59	0.64	0.71	0.81	0.26	0.78	0.74	0.27	1.48	7.25	840	0.90
0.71	1.14	0.36	0.38	0.31	0.45	0.43	0.54	0.8	0.57	5.7	840	1.15
0.6	0.93	0.64	0.48	0.88	0.57	0.32	0.63	0.81	1.17	7.03	840	0.87
0.49	0.94	0.78	0.66	0.68	0.47	0.85	0.58	0.46	0.45	6.36	1740	1.53
0.77	0.4	0.6	0.65	0.78	0.47	0.38	0.67	0.67	0.58	5.96	1740	1.15
0.3	0.56	0.3	0.32	0.37	0.47	0.33	0.79	0.87	0.79	5.1	1740	1.70
0.36	0.55	0.5	1	0.59	0.72	0.44	0.47	0.73	0.91	6.28	1740	1.23
0.79	0.35	0.69	0.58	0.58	0.47	0.44	0.47	0.87	0.57	5.8	1740	1.34

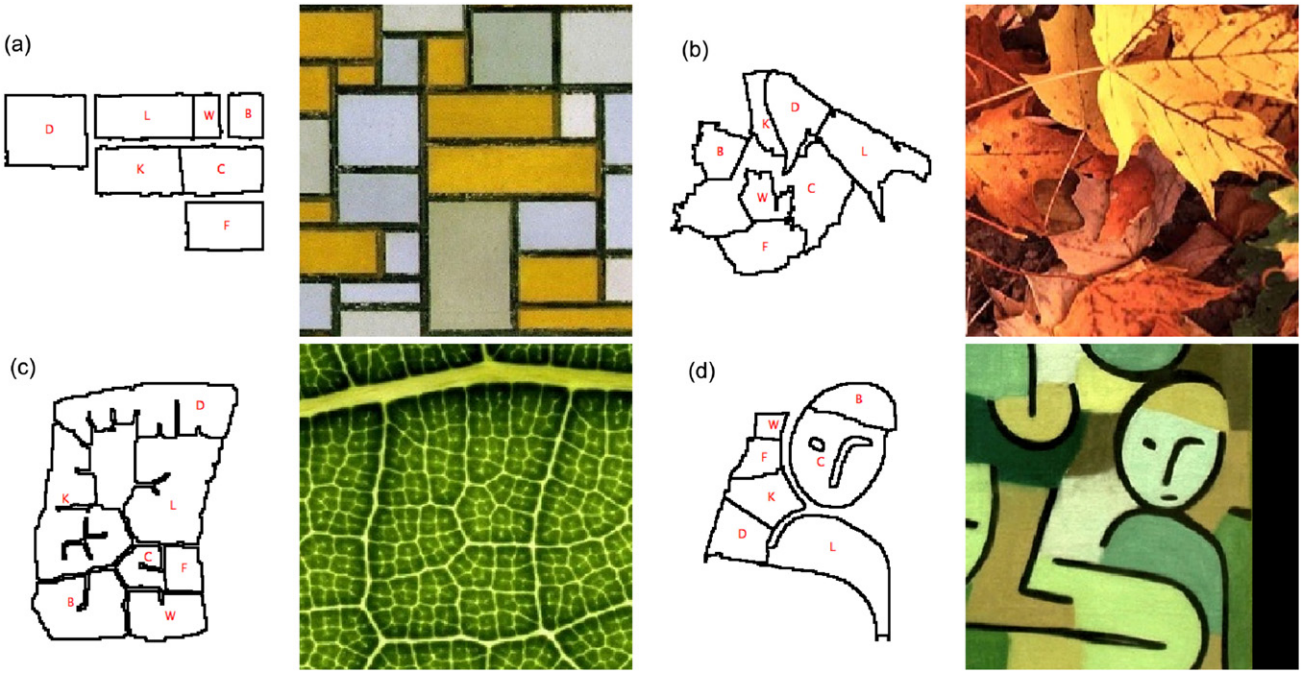


Fig. 8. The program turns a given image into a floor plan that satisfies user-defined geometric and topological requirements. The resulting plan reflects to a large extent the original patterns in the given image. Four pairs of results for the same design program (Table 2) are shown in the figure.

3.2.2. Patterns and diversity

The program allows designers to specify a set of parametric wall segments (or select a predefined set) at the beginning, so that the designers implicitly address their visual preferences for the resultant plans. This approach is analogous to architect Mies van der Rohe's composition of L and T shapes (Fig. 6 (a)). The program enables architects to define their visual vocabularies and immediately see the synthesized results. The generated layouts are highly irregular; the arrangement of the activities is somewhat unpredictable.

Fig. 10 shows that a design program and a set of wall segments can lead to a variety of floor plans. This complies with the consensus among architects that one design problem must lead to multiple solutions; otherwise, there is no freedom of design. The program provides a pool of candidates from which the architect can make further developments. They may also modify the program settings and conduct new searches. Thus, this automated tool involves the graphical communications (instead of mathematical ones) between the participants and the automation process.

3.2.3. Implementation and performance

The algorithm is coded in Java. Without multithreading, it takes dozens of seconds to find a valid layout on a computer with a 2.4 GHz Intel Core i5 processor. Table 4 shows a series of tests with runtimes and geometric errors. These results indicate that the search time increases with the number of rooms. The simulated annealing algorithm has a logarithmic time complexity. However, the probability of obtaining a better solution via a random move decreases as the number of rooms increases. As such, the actual efficiency is no better than that of quadratic complexity algorithms. The algorithm may fail with a very large number of rooms. Thus, future work could employ hierarchical structures to deal with larger design programs; for instance, one can first arrange the structure into zones and then place the rooms within these zones.

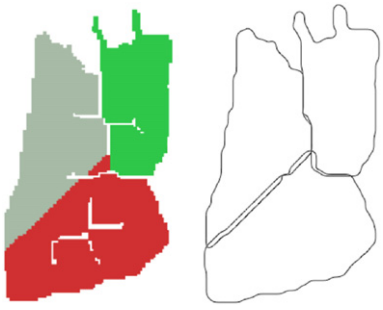


Fig. 9. Creation of smooth edges by Gaussian blur and blob detection. Left: original room shapes; right: smoothed edges.

4. Conclusions and future work

This research links layout automation with intensive graphical specifications from the designers. A framework and two applications that allow the designer to feed patterns from an image into the layout automation program have been presented. The participant is able to implement their personal preferences without deep knowledge of modeling and computing. Compared with previous work, most of which eliminated the involvement of the user in the automation process, the proposed method allows graphical input from the user. In addition, most previous work relied on rectilinear structures, while this work does not depend on any particular structure in the plane.

This layout automation program focuses on dimensions and adjacencies of rooms and does not take into account realistic building specifications, such as building materials, natural lighting, heating, and energy conservation. One can later analyze the generated floor plans with software such as EnergyPlus or Autodesk's Ecotect. However, a future development may be to incorporate more performance criteria in the automation. There are two ways to include more objectives. One strategy would be to develop a complete cost function (including a sophisticated simulation and assessment) to be minimized by the generative process. Examples are [11,15,16]. The other approach is a post-generation method; this would involve carrying out an additional optimization on a generated plan [25]. This work contributes to the framework of performance-based design [26].

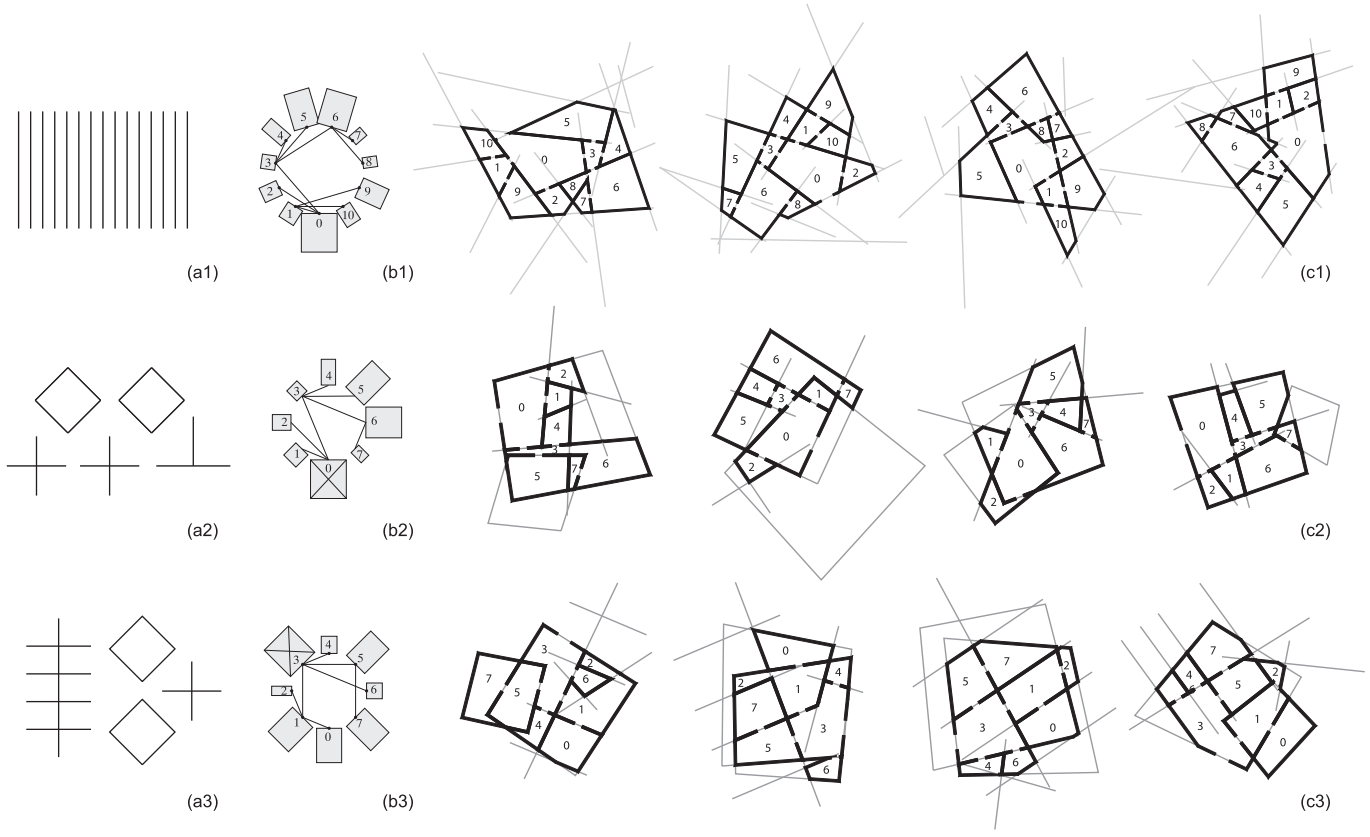


Fig. 10. Three groups of tests. For example, in the first group, (a1) depicts the user-defined segments; (b1) illustrates the dimension requirement and the topological requirement; and (c1) is the four resulting plans, with the rooms specified in (b1).

Because the synthetic data generated by the program provides a large set of candidate solutions in the early design stages, designers are able to select (either manually or via programming) the performative solution(s) via specific simulation/evaluation software in later design stages.

This work only considers single level buildings. However, it is possible to extend the program to multi-story buildings: First, the rooms can be aligned according to the stairs/elevators so that all rooms on distinct levels can be served. Second, the differences between the footprints of consecutive levels can be minimized.

This work can also be integrated with other applications for the built environment. For instance, a number of applications have been developed for extruding a complete floor plan into a 3D building model [27]. Creating a cohesive façade [28] for the 3D model is also practical for building design. The proposed tool would be helpful in correlating the façade components with the activities behind the façade. Another promising direction would be placing the generated buildings into an urban environment in a systematic way, such as with CityEngine [29]. This software allows the user to create the designs through manual or rule editing, while this research adds a new, yet more intuitive, method: specifying patterns with images.

Acknowledgments

I am grateful to Prof. Ludger Hovestadt at ETH Zürich and Prof. Li Biao at Southeast University Nanjing for their valuable advice. This work is supported by National Natural Science Foundation of China (51408123, 51478116, 51478101, 51538006).

References

[1] I.E. Sutherland, Sketchpad: a man-machine graphical communication system, DAC '64 Proceedings of the SHARE Design Automation Workshop, ACM, New York, NY, USA, 1964, pp. 329–346. http://dx.doi.org/10.1145/800265.810742.
[2] C. Langenhan, M. Weber, M. Liwicki, F. Petzold, A. Dengel, Graph-based retrieval of building information models for supporting the early design stages, Adv. Eng. Inform. 26 (2013) 413–426.

Table 4
Tests with vector graphics: runtime, and geometric quality $(m(R_i, A_i)/A_i)$ of each room.

Room 1	2	3	4	5	6	7	8	8	10	11	Iteration	Runtime(s)
0.86	0.87	0.89	0.51	0.88	0.98	0.87	0.78				7424	19.1
0.87	0.67	0.94	0.77	0.85	0.89	0.88	0.88				3714	14.4
0.88	0.79	0.82	0.89	0.82	0.86	0.86	0.87				2317	1.3
0.87	0.88	0.90	0.85	0.88	0.91	0.88	0.69				14,897	21.4
0.80	0.75	0.87	0.87	0.88	0.95	0.87	0.99				13,433	25.9
0.86	0.83	0.80	0.91	0.89	0.90	0.87	0.79	0.78	0.86	0.81	6686	34.2
0.82	0.94	0.85	0.72	0.84	0.85	0.91	0.86	0.81	0.92	0.89	8112	38.7
0.84	0.88	0.83	0.77	0.90	0.84	0.93	0.87	0.92	0.87	0.94	3081	30.8
0.89	0.88	0.89	0.85	0.8	0.86	0.92	0.82	0.88	0.87	0.87	6471	31.6
0.84	0.85	0.80	0.81	0.85	0.89	0.89	0.77	0.87	0.89	0.76	8708	45.6

[3] G.C. Armour, E.S. Buffa, A heuristic algorithm and simulation approach to relative location of facilities, Management Science 9 (2) (1963) 294–309.
[4] B. Whitehead, M.Z. Eldars, The planning of single-storey layouts, Building Science 1 (2) (1965) 127–139.
[5] J.M. Seehof, W.O. Evans, J.W. Friederichs, J.J. Quigley, Automated facilities layout programs, ACM'66 Proceedings of the 1966 21st National Conference, ACM, New York, NY, USA, 1966, pp. 191–199. http://dx.doi.org/10.1145/800256.810696.
[6] M. Krejcirik, Computer-aided plant layout, Comput. Aided Des. 2 (1) (1969) 7–19.
[7] U. Flemming, S.F. Chien, Schematic layout design in SEED environment, J. Archit. Eng. 1 (4) (1995) 162–169.
[8] R.S. Liggett, Automated facilities layout: past, present and future, Autom. Constr. 9 (2) (2000) 197–215.
[9] M.A. Rosenman, The generation of form using an evolutionary approach, in: J.S. Gero, F. Sudweeks (Eds.), Artificial Intelligence in Design '96, Kluwer Academic, Kluwer Academic Publishers. 1996, pp. 643–662.
[10] E. Rodrigues, A.R. Gaspar, Á. Gomes, An evolutionary strategy enhanced with a local search technique for the space allocation problem in architecture, Part 1: methodology, Comput. Aided Des. 45 (5) (2013) 887–897.
[11] J. Michalek, P. Papalambros, Interactive design optimization of architectural layouts, Eng. Optim. 34 (5) (2002) 485–501.
[12] P. Merrell, E. Schkufza, V. Koltun, Computer-generated residential building layouts, ACM Trans. Graph. 29 (6) (2010) 485–501.
[13] E. Rodrigues, A.R. Gaspa, Á. Gomes, An approach to the multi-level space allocation problem in architecture using a hybrid evolution technique, Autom. Constr. 35 (2013) 482–498.
[14] H. Hua, A case-based design with 3D mesh models of architecture, Comput. Aided Des. 57 (2014) 54–60.
[15] O. Chouchoulas, Shape Evolution: An Algorithmic Method for Conceptual Architectural Design Combining Shape Grammars and Genetic Algorithms, University of Bath. 2003. (Ph.D. thesis)
[16] A. Menges, Biomimetic design processes in architecture: morphogenetic and evolutionary computational design, Bioinspir. Biomim. 7 (1). (2012).
[17] H. Hua, Decoupling grid and volume: a generative approach to architectural design, in: H. Achten, J. Pavliček, J. Hulín, D. Matějovská (Eds.), Proceedings of the 30Th ECAADe Conference-Volume 1, eCAADe, Prague, Czech Republic, 2012, pp. 311–317.
[18] M. Braach, Solutions you cannot draw, Archit. Des. 84 (5) (2014) 46–53.
[19] L. Hovestadt, Beyond the Grid-architectural and Information Technology, Birkhäuser Verlag. 2010.
[20] B. Dillenburger, M. Braach, L. Hovestadt, Building design as individual compromise between qualities and costs, in: T. Tidafi, T. Dorta (Eds.), Proceedings of the 13Th International CAADFutures Conference, CAADFutures, Les Presses de l'Université de Montréal. 2009, pp. 458–471.
[21] A. Doulgerakis, Genetic Programming + Unfolding Embryology in Automated Layout Planning, University College London. 2007. Master's thesis
[22] R. Nock, F. Nielsen, Statistical region merging, IEEE Trans. Pattern Anal. Mach. Intell. 26 (11) (2004) 1452–1458.
[23] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry, Springer. 2008.
[24] S. Gold, A. Rangarajan, A graduated assignment algorithm for graph matching, IEEE Trans. Pattern Anal. Mach. Intell. 18 (4) (1996) 377–388.
[25] E. Rodrigues, A.R. Gaspa, Á. Gomes, Improving thermal performance of automatically generated floor plans using a geometric variable sequential optimization procedure, Appl. Energy 132 (2014) 200–215.
[26] R. Oxman, Digital architecture as a challenge for design pedagogy: theory, knowledge, models and medium, Des. Stud. 29 (2008) 99–120.
[27] X. Yin, P. Wonka, A. Razdan, Generating 3D building models from architectural drawings: a survey., IEEE Comput. Graph. Appl. 29 (1) (2009) 20–30.
[28] P. Müller, G. Zeng, P. Wonka, L.V. Gool, Image-based procedural modeling of facades, ACM Trans. Graph. 26 (3). (2007)
[29] Y.I.H. Parish, P. Müller, Proceedings of the 28Th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '01, ACM, New York, NY, USA, 2001, pp. 301–308. http://dx.doi.org/10.1145/383259.383292.