Raster-to-Vector: Revisiting Floorplan Transformation

Chen Liu
Washington University in St. Louis
chenliu@wustl.edu

Jiajun Wu
Massachusetts Institute of Technology
jiajunwu@mit.edu

Pushmeet Kohli†
DeepMind
pushmeet@google.com

Yasutaka Furukawa*
Simon Fraser University
furukawa@sfu.ca

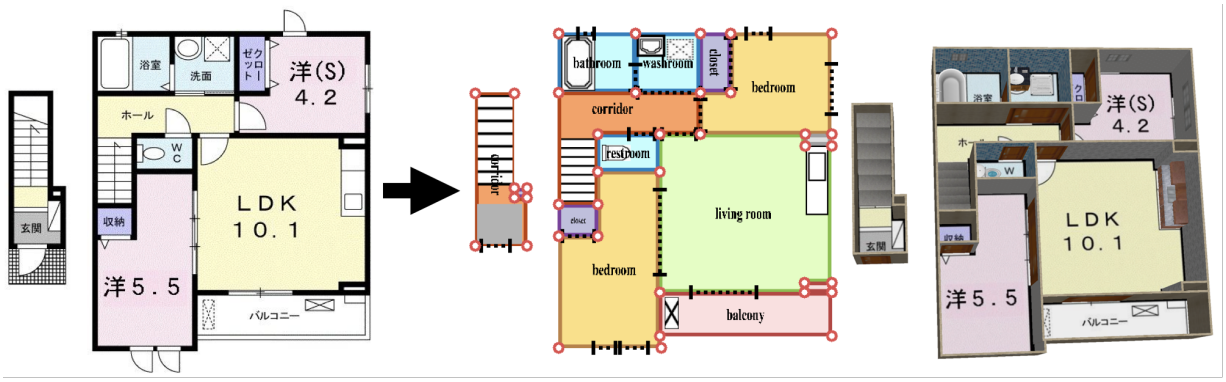

Figure 1: This paper makes a breakthrough in the problem of converting raster floorplan images to vector-graphics representations. From left to right, an input floorplan image, reconstructed vector-graphics representation visualized by our custom renderer, and a popup 3D model.

Abstract

This paper addresses the problem of converting a rasterized floorplan image into a vector-graphics representation. Unlike existing approaches that rely on a sequence of low-level image processing heuristics, we adopt a learning-based approach. A neural architecture first transforms a rasterized image to a set of junctions that represent low-level geometric and semantic information (e.g., wall corners or door end-points). Integer programming is then formulated to aggregate junctions into a set of simple primitives (e.g., wall lines, door lines, or icon boxes) to produce a vectorized floorplan, while ensuring a topologically and geometrically consistent result. Our algorithm significantly outperforms existing methods and achieves around 90% precision and recall, getting to the range of production-ready performance. The vector representation allows 3D model popup for better indoor scene visualization, direct model manipulation for architectural remodeling, and further computational applications such as data analysis. Our system is efficient: we have converted hundred thousand production-level floorplan images into the vector representation and generated 3D popup models.

1. Introduction

Architectural floorplans play a crucial role in designing, understanding, or remodeling indoor spaces. Their drawings are very effective in conveying geometric and semantic information of a scene. For instance, we can quickly identify room extents, the locations of doors, or object arrangements (geometry). We can also recognize the types of rooms, doors, or objects easily through texts or icon styles (semantics).

While professional architects or designers draw floorplans in a vector-graphics representation using software such as AutoCAD [1], HomeStyler [3], or Sketchup [5], the final use of an artwork is often just visualization for clients (e.g., home buyers or renters). As a result, floorplans are rasterized to print or digital media for publication. This process discards all the structured geometric and semantic information, limiting human post-processing or further computing capabilities such as model analysis, synthesis, or modification.

*At Washington University in St. Louis at the time of the project.
†At Microsoft Research Redmond at the time of the project.

Recovering the lost information from a rasterized floorplan image is a surprisingly hard task and has been a long-standing open problem. The problem poses two fundamental challenges. First, floorplan structure must satisfy high-level geometric and semantic constraints. For instance, walls corresponding to an external boundary or certain rooms must form a closed 1D loop. Second, this high-level model structure varies across examples (*e.g*., different houses have different numbers of bedrooms). Computer vision has recently witnessed dramatic progresses on similar high-level model prediction tasks, such as human hand or body pose estimation. In those problems, however, the model structure is fixed: Human hands have five fingers. In our problem, both the model structure as well as parameters need to be inferred.

This paper proposes a novel solution to the problem of raster-to-vector floorplan transformation. Existing techniques typically rely on a sequence of low-level image processing heuristics [14]. Our approach integrates a deep network and an integer programming through two novel intermediate floorplan representations. First, a deep network extracts low-level geometric and semantic information into a set of junctions (*i.e*., points with types) as the first representation. For example, a wall structure is represented as a set of junctions of four different types, depending on the degree of its connections. Note that we assume Manhattan world and restrict candidate lines or boxes to be axis-aligned. Second, an integer programming (IP) aggregates junctions into a set of simple primitives (*e.g*., walls as lines and icons as boxes). IP enforces higher-level constraints among primitives, for instance, a room as a chain of wall primitives forming a closed loop. This ensures that a simple post-processing step can produce a complete vector representation,

We have manually annotated 870 floorplan images for training as well as for quantitative evaluation. The proposed algorithm significantly outperforms existing methods with around 90% precision and recall, getting closer to production-ready performance. We believe that this paper makes progress in understanding floorplans with a new approach achieving a significant performance boost, a large-scale benchmark for evaluating future algorithms, and a large corpus of vector floorplan data and popup 3D models, opening potentials for a new family of Computer Vision research.

2. Related work

Raster to vector conversion of floorplan images has a long history in Pattern Recognition. Existing approaches generally rely on a sequence of low-level image processing heuristics. The first critical step is the separation of the textual data from the graphical ones. Textual data (the dimensions, room labels, *etc*.) can be separated by finding connected components, removing those that are either too small, too large, or too far away from other texts, and identifying the bounding boxes (*i.e*., OCR process) [21]. For graphical data, lines are the fundamental primitives to be extracted first. Morphological operations, Hough transform, or image vectorization techniques are used to extract lines, normalize line width, or group them into thin/medium/thick lines [6, 11]. Detected lines are used to identify walls, where existing approaches require various heuristics, such as convex hull approximation, polygonal approximation, edge-linking to overcome gaps, or the color analysis along lines [18, 6]. Doors and windows exist in walls, and are detected by either looking at the openings on walls [6] or symbols spotting techniques [18]. Alternatively, image recognition techniques such as a bag of visual words can be used [11]. Finally, rooms can be extracted by decomposing images [18] or finding the connected components surrounded by the walls, assuming that rooms are always separated by walls [11]. Existing systems often consist of several steps with various heuristics in each processing unit. Their level of performance falls far behind that of human annotators or what is required for production. This paper significantly improves results in the literature with a fundamentally new approach consisting of two computational machineries, deep representation learning and integer programming.

One major problem of current floorplan conversion research is the lack of a representative large-scale dataset. Heras *et al*. [12] combined multiple existing datasets to acquire vector-graphics representation groundtruth for 122 floorplans. Liu *et al*. [17] provided a dataset which contains 215 floorplans and associated images for the floorplan-image alignment problem. However, their dataset does not fit our purpose due to the small quantity and the limited variations in style. In contrast, we have annotated 870 images and tested our system on 100,000 unannotated images.

Besides the raster-to-vector transformation problem, the Computer Vision community has tackled various problems involving floorplan images. Photograph to floorplan alignment has been an active research topic in recent years, with applications ranging from image localization [19, 10], navigation [22, 10], to real-estate content creation [17]. Pointcloud to floorplan alignment has also been studied for building-scale 3D scanning [23]. Indoor 3D reconstruction from images [13], depth-images [9], or laser-scanned pointclouds [20, 8, 16, 24] are also closely related to floorplan modeling. However, the critical difference is that these 3D reconstruction methods recover a surface representation (*i.e*., a wall as two surfaces instead of one thickened plane), and do not explicitly model the wall structure as a skeleton for instance. The surface-based representation suffices for rendering, but cannot allow standard floorplan editing operation, such as moving walls to change room sizes, or thinning walls for a new style. The goal of this paper is to recover a vector representation of a floorplan as a CAD model, which enables human post-processing or further computing applications.

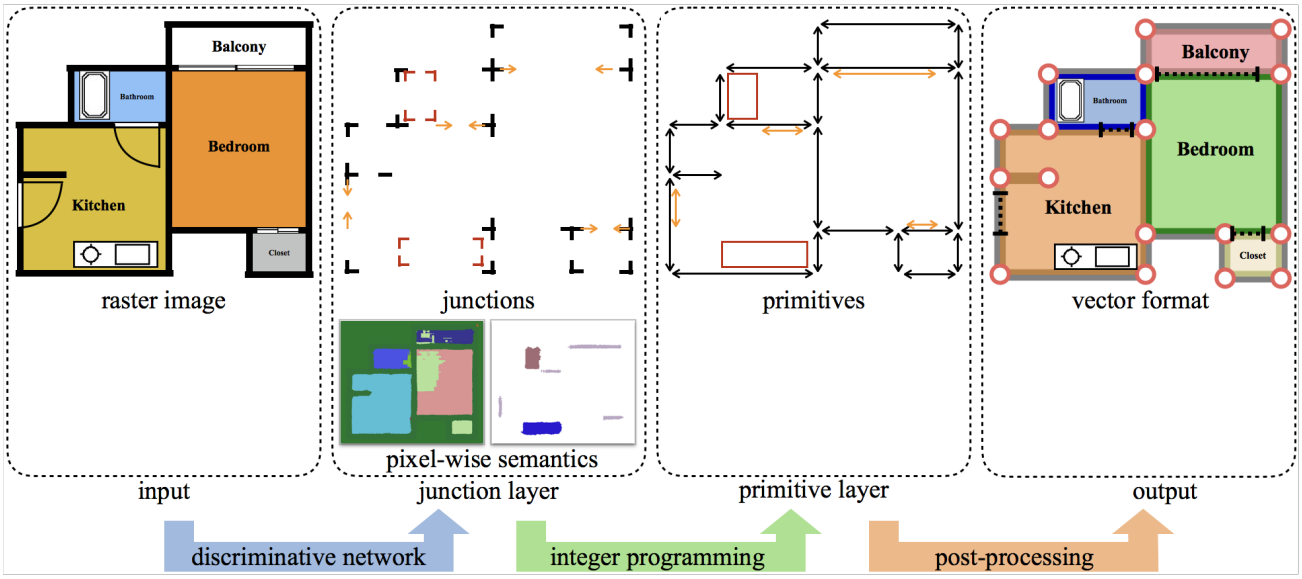

Figure 2: Our approach converts a floorplan image through two intermediate representation layers. A neural architecture first converts a floorplan image into a junction layer, where data is represented by a set of junctions (*i.e.*, points with types) or per-pixel semantic classification scores. An integer programming aggregates junctions into a set of primitives (*i.e.*, lines or boxes), while ensuring a topologically and geometrically consistent result. A simple post-processing can be used to produce the final vector format.

3. Stratified floorplan representation

Instead of directly solving for the vector representation with high-level structure, we take a stratified approach and convert a floorplan image through two intermediate floorplan representations (See Fig. 2). This stratified approach effectively integrates the use of a Convolutional Neural Network (CNN) and Integer Programming (IP), where the network extracts low-level geometric and semantic information as junctions, and IP aggregates junctions into primitives while enforcing high-level structural constraints.

In the intermediate layers, the floorplan data are represented by three factors: walls, openings (doors or windows), or icons. Note that we do not have a factor corresponding to rooms, which will come out in the final post-processing step. Door and window symbols are usually indistinguishable, and are modeled as the same "openings". In our system, we treat an opening as a window if one side of the opening is outside in the final vector format, except for the main door which sits next to the entrance. This section explains how we encode those factors through our intermediate layers, namely the junction layer and the primitive layer.

3.1. Junction layer

The first layer encodes factors as a set of junctions (*i.e.*, points with types).

- Wall structure is represented by a set of junctions where wall segments meet. There are four wall junction types, I-, L-, T-, and X-shaped, depending on the degrees of incident wall segments (*c.f.* Fig. 3). Considering orientations, there are in total 13 $(= 4 + 4 + 4 + 1)$ types.
- An opening is a line represented by opening junctions at its two end-points. Considering rotational variations, there are 4 types of opening junctions.
- An icon is represented by an axis aligned bounding box, and hence, icon junctions consist of four types: top-left, top-right, bottom-left, or bottom-right.

The junction layer also has two types of per-pixel probability distribution maps over different semantic types. The first map distinguishes if a pixel belongs to a wall or a certain room type. The map is a probability distribution function (PDF) over 12 different classes, as there are 11 room types: *living-room*, *kitchen*, *bedroom*, *bathroom*, *restroom*, *washing-room*, *balcony*, *closet*, *corridor*, *pipe space*, or *outside*. The second map distinguishes if a pixel belongs to an opening, a certain icon type or *empty*. The map is a PDF over 10 different classes, as there are 8 icon types: *cooking counter*, *bathtub*, *toilet*, *washing basin*, *refrigerator*, *entrance mat*, *column*, or *stairs*. Note that a pixel can be both a bathroom (first map) and a bathtub icon (second map).

3.2. Primitive layer

The second layer encodes the three factors as low-level geometric primitives. A wall or an opening primitive is represented as a line, namely a valid pair of junctions. Similarly, an icon primitive is represented as a box, that is, a valid quadruple of junctions. Here, "validity" denotes if junctions

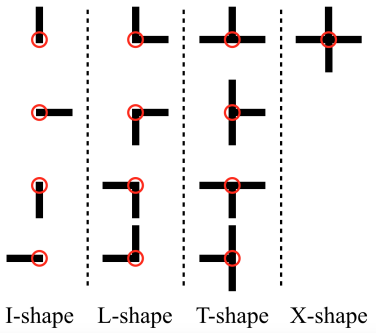

Figure 3: There are four wall junction types: I-, L-, T-, and X-shaped, depending on the degrees of incident wall segments. Considering orientations, we have in total 13 $(= 4 + 4 + 4 + 1)$ types.

can be connected given their orientations. Each primitive is also associated with some semantics information.

• A wall primitive is associated with room types at its both sides. The possible room types are the same as in the junction layer.
• An opening primitive, which is either a door or a window, does not have any semantics associated at this layer, as doors and windows are indistinguishable on our floorplan images.
• An icon primitive is associated with one of the icon types, which are the same as in the junction layer.

Primitives must satisfy a variety of constraints so that a simple post-processing can extract a floorplan vector representation with high-level structure. For example, a bedroom must be represented by a set of wall primitives that form a closed-loop and have consistent room type annotation on one side. The constraints are enforced in solving the Integer Programming as explained in Section 4.2.

4. Raster to vector conversion

Our system consists of three steps. First, we employ a CNN to convert a raster floorplan image into the first junction layer (*i.e.*, junction maps and per-pixel room-classification scores). Second, after generating primitive candidates from junctions based on simple heuristics, we use Integer Programming to select the right subset while enforcing high-level geometric and semantic constraints. Third, a simple post-processing is used to convert the floorplan data into the final vector-graphics representation. We now explain each step.

4.1. Junction layer conversion via a CNN

CNNs have been proven to be very powerful in extracting low-level information from images. Our junction and per-pixel classification representation allows straight-forward application of CNNs. We borrow the residual part of the detection network from [7], which modified ResNet-152 [15] to predict heatmaps at pixel-level. We drop their last deconvolution layer, and append three deconvolution layers in parallel, one for junction heatmap regression and two for per-pixel classifications. For junctions, there are at total $21 (= 13 + 4 + 4)$ different types, and one heatmap is regressed for each type, where pixelwise sigmoid cross entropy loss is applied. For classification tasks, we use pixelwise softmax cross entropy loss. We train three branches jointly, and the final loss is a weighted summation with larger weight, 20, only for junction heatmap regression. Both the input and output have resolution 256x256. Besides common data augmentation techniques like random cropping and color jittering, we also rotate the image with an angle randomly picked from 0°, 90°, 180°, and 270°. During inference, we threshold junction heatmaps with 0.4 (slightly lower than 0.5 to bring in more junction candidates for IP to choose), and apply non-maximum suppression to extract junctions.

While the network makes very good predictions of junction locations, it sometimes mis-classfies junction types (*e.g.*, mis-classifies a L-shaped junction as T-shaped). To make the detection robust, we allow one mistake in the estimation of the degree. For example, for each detected T-shaped junction, we hallucinate two L-shaped junctions and one X-shaped junction at the same location. The integer programming will enforce later that at most one of the junctions can be used. We found that mis-classification between I and L is rare, and perform the hallucination for all the other cases.

4.2. Primitive layer conversion via IP

Deep network makes very good predictions and simple heuristics suffice to extract primitive candidates (*i.e.*, walls, openings, and icons). Integer programming then finds the correct subset while enforcing various geometric and semantic constraints. With the small problem size, it takes around 2s to find the optimal solution to IP using Gurobi [2].

4.2.1 Primitive candidate generation

A wall primitive can be formed by two wall junctions if 1) they are axis-aligned with a tolerance of 10 pixels, and 2) their aligned orientation is compatible with the junction orientations. Similarly, two door junctions can form a door primitive if qualified. For icons, four axis-aligned junctions (top-left, top-right, bottom-right, and bottom-left) together form an icon primitive.

4.2.2 Integer programming

Integer Programming enforces geometric and semantic constraints among primitives to filter out spurious primitive candidates and guarantee properties of floorplan data, which must hold true. For instance, a bedroom must be surrounded by a set of walls forming a 1D loop, with a bedroom type associated with the correct side of each wall primitive.

Variable definition: We define indicator variables for junctions, primitives, and semantic types:

- $J_{wall}(j)$, $J_{open}(j)$, $J_{icon}(j)$ for junctions,
- $P_{wall}(p)$, $P_{open}(p)$, $P_{icon}(p)$ for primitives,
- $S^L_{wall}(p,s)$, $S^R_{wall}(p,s)$, $S_{icon}(p,s)$ for semantics.

j, p and s denote indexes for junctions, primitives and possible semantic types, respectively. For instance, $P_{open}(p)$ is an indicator variable encoding if the p_{th} opening primitive exists or not. Indicator variables for semantics employ one-hot encoding and have two indexes, a primitive index and a semantic type index. Lastly, a wall primitive is associated with two room types as semantics on its both sides. For instance, $S^L_{wall}(p,s)$ indicates if the p_{th} wall primitive has the s_{th} room type on the left hand side.

Objective function: The objective function for maximization is a linear combination of the junction and semantic indicator variables, where weights are defined as follows:

- The weight is 10 for all the junctions except for the hallucinated wall junctions, whose weight is set to -5 (See Section 4.1). In other words, we encourage the use of the primitives as much as possible, but discourage the use of the hallucinated ones.
- The weights for the semantic indicator variables are calculated based on the per-pixel classification scores in the junction layer. For an icon type indicator variable, the weight is simply the average icon type classification score inside the box. For a room type indicator variable associated with a wall primitive on one side, we use the average room type classification score in its neighborhood. A neighborhood is obtained by sweeping pixels on the wall primitives along its perpendicular direction (on the side of the room type variable). Each pixel is swept until it hit another wall primitive candidate.

We have not used primitive indicator variables in the objective function, as similar information has been already captured by the junction primitives.

Constraints: We enforce a variety of constraints either as linear equalities or linear inequalities.

- One-hot encoding constraints: When a wall primitive does not exist (i.e., $P_{wall}(p) = 0$), its wall semantic variables must be all zero. When it exists, one and only one semantic variable must be chosen. The same is true for icon primitives, yielding the following constraints.

$$P_{wall}(p) = \sum_s S^L_{wall}(p,s) = \sum_s S^R_{wall}(p,s),$$
$$P_{icon}(p) = \sum_s S_{icon}(p,s).$$

- Connectivity constraint: The degree (*i.e.*, the number of connections) of a junction must match the number of incident primitives that are chosen. This applies to walls, openings and icons, and here we only show the constraint for the walls, where the summation is over all the wall primitives connected with the wall junction:

$$\{\#\text{ degree}\} J_{wall}(j) = \sum P_{wall}(p).$$

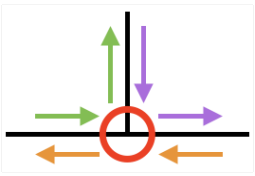

Figure 4: Loop constraints can be enforced locally at each junction. The room types must be the same for each pair of walls marked with the same color.

- Mutual exclusion constraints: Two primitives cannot be chosen when they are spatially close, in particular, within 10 pixels. We find every pair of such primitives and enforce that the sum of their indicator variables is at most 1. The same constraint is also applied to wall junctions to ensure that two wall junctions are not close and hallucinated junctions are not selected simultaneously with the original one.
- Loop constraints: Bedroom, bathroom, restroom, balcony, closet, pipe-space, and the exterior boundary must form a closed loop (allowing some walls that stick out). It turns out that this high-level rule can be enforced by local constraints at every wall junction. We use a T-shaped wall junction to explain our idea in Fig. 4. Room types must be the same for a pair of walls with arrows of the same color in the figure.
- Opening constraints: An opening (a door or a window) must be on a wall. For each opening primitive, we find a list of wall primitives that contain the opening (parallel to the opening with distance smaller than 10 pixels) and enforce the following. Note that the right summation is over all the collected wall primitives.

$$P_{open}(p) \leq \sum P_{wall}(p).$$

4.3. Final data conversion

The IP output is close to the final representation with a few issues remaining. First, junctions are not well-aligned, because we allow some coordinate error when finding connections. The alignment issue can be simply fixed by averaging the junction coordinates along a straight line. The second issue is that doors are not sitting exactly on walls. To fix this issue, we move each door to align with its closest wall. The last issue is the missing of high-level room information, as room labels are currently associated with walls locally. To derive room information, we first find all the closed polygons formed by walls. If all the walls of a polygon share the same

room label, then polygon forms a room with that label. For a polygon with multiple labels, we further split the polygon by either horizontal lines or a vertical lines into sub-regions, and associate each sub-region with a room label. The detailed split algorithm is in the supplementary material.

5. Evaluations

The section explains 1) our data collection process; 2) evaluation metrics; 3) quantitative evaluations; and 4) qualitative evaluations.

5.1. Annotating a Floorplan Dataset

To fully exploit data-hungry neural architecture, we have built a large-scale dataset with groundtruth for vector-graphics floorplan conversion, based on the LIFULL HOME'S dataset [4] which contains 5 million floorplan raster images. To create the groundtruth, we randomly sample 1,000 floorplan images and ask human subjects to annotate the geometric and semantic information for each floorplan image. An annotator can draw a line representing either a wall or an opening, draw a rectangle and pick an object type for each object, or attach a room label at a specific location. We then convert the annotated information to our representation. The conversion is straightforward, and please refer to our supplementary material for details. We perform automatic checks to make sure that all the geometric and semantic constraints discussed in Section 4.2.2 are satisfied. After this verification, we manually go through all the annotations to correct the remaining mistakes. After dropping images with poor qualities, we collect 870 groundtruth floorplan images. As a benchmark, 770 of them are used for network training, while the remaining 100 examples are served as test images.

5.2. Metrics

We compute the precision and recall scores for wall-junctions, opening primitives, icon primitives, and rooms. For a prediction to be correct, its distance to a target must be the smallest compared to the other prediction, and less than a threshold. For wall junctions, we use Euclidean distance, and threshold τ_w. For opening primitives, the distance between a prediction and a target is the larger value of Euclidean distances between two pairs of corresponding endpoints, and the threshold is τ_o. For objects and rooms, we calculate Intersection Over Union (IOU), and use $1 - IOU$ as distance (with threshold τ_o and τ_r respectively). We set $\tau_w = \tau_d = 0.01 \max(W, H)$, $\tau_o = 0.5$ and $\tau_r = 0.3$, where W and H denote the image resolution.

5.3. Quantitative evaluations

We evaluate our results on our benchmark images using the metrics mentioned in Section 5.2 (See Table 1). We have implemented the algorithm in [6] as a baseline. We have not implemented OCR for detecting rooms labels in their algorithm, as we could not find a free open-source OCR system that handles the mix of Japanese and English. However, we believe that the use of OCR is comparable to our deep-network based solution. Their method also ignores objects, so we only evaluate the wall and door detection in their method and show results in Table 1. The table also shows the performance of our approach with various algorithmic features removed for an ablation study.

To evaluate the effectiveness of each constraint in IP, we disable one constraint each time and record the performance. Note that we have not evaluated the one-hot encoding constraints and the connectivity constraints, since they are essential. Table 1 shows that when full IP is performed, we improve precision consistently over the results without IP. The mutual exclusion constraints introduce the competition between conflicting primitives, and thus filter out many false-positives. This filtering process improves all precisions by a large margin, with only small sacrifice in recall for openings and objects. On the contrary, the recall for rooms increases due to the more accurate room shapes formed by walls. The loop constraints improve the precision for rooms by 2%. This improvement is critical in some cases to ensure the visual quality of conversion. The opening constraints ensure that no opening appears without a wall.

5.4. Qualitative evaluations

Figure 5 shows an input floorplan image, the reconstructed vector representation visualized by our own renderer, and a popup 3D model for each example. In our rendering, a wall junction is highlighted as a disk with a red border, an opening primitive is shown with a black dashed line, an object primitive is shown as an icon with different styles, depending on the inferred semantics. We also change the background color of each room based on its type. The popup 3D model is generated by extruding wall primitive to a certain height, adding window or door textures at the location of opening primitives (an opening becomes a window, if it faces outside), and placing 3D objects models in the bounding box of icon primitives.

We have manually verified the correctness of the floorplan data, which are shown by the three numbers for the wall junctions, opening primitives, icon primitives, and rooms. For example, (57/59/60) means that there are 59 target predictions to make, where our algorithm makes 60 predictions and collected 57 correct ones. Verified by the quantitative evaluation, our algorithm is able to recover near complete floorplan models despite of varying styles (e.g., black and white, colored background, or textured background mixing Japanese, English, and Chinese characters). Please refer to the supplementary material for more results. We have run our algorithm on 100,000 raster images to recover their vector-

Figure 5: Floorplan vectorization results. From left to right, an input floorplan image, reconstructed vector-graphics representation visualized by our custom renderer, and the corresponding popup 3D model. We have manually checked the correctness of each junction and primitive estimation, whose statistics are shown under each example.

Method	Wall Junction		Opening		Icon		Room	
	Acc.	Recall	Acc.	Recall	Acc.	Recall	Acc.	Recall
Ahmed *et al.* [6]	74.9	57.5	61.3	48.7	N/A	N/A	N/A	N/A
Ours (without IP)	70.7	95.1	67.9	91.4	22.3	77.4	80.9	78.5
Ours (without mutual exclusion constraints)	92.8	91.7	68.5	91.1	22.0	76.2	82.8	87.5
Ours (without loop constraints)	94.2	91.5	91.9	90.2	84.3	75.0	82.5	88.2
Ours (without opening constraints)	94.6	91.7	91.7	90.1	84.0	74.8	84.3	88.3
Ours (with full IP)	94.7	91.7	91.9	90.2	84.0	74.6	84.5	88.4

Table 1: Quantitative evaluations based on our benchmark.

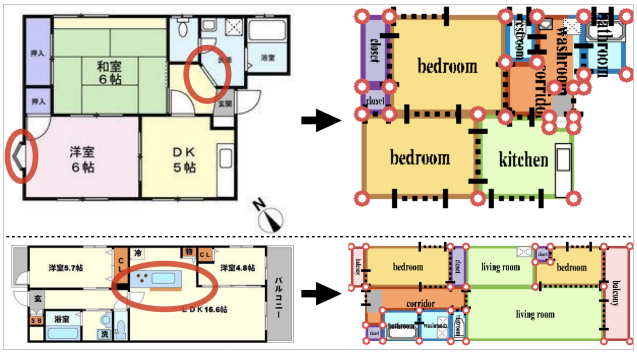

Figure 6: Typical failure cases. Relying on Manhattan assumption, our method is unable to detect walls which are neither horizontal nor vertical as shown in the first example. In the second example, our method puts a wall which does not exist.

Figure 7: Floorplan vectorization results on an image in Rent3D (top) and an image from the web (bottom).

graphics representation and 3D popup models. Though rare, there are some typical failure cases as shown in Fig. 6.

Lastly, we have evaluated the generalization capability of our system by processing floorplan images from other data sources. Figure 7 shows our results on two such examples: one from Rent3D database [17], and the other from Google image search with a search keyword "floorplan". Despite the fact that these two images have very distinctive styles from examples in our dataset, our system has successfully reconstructed most walls and openings. Due to dramatic differences in styles, semantic information (i.e., icon types and room types) is often mis-recognized. Again please refer to the supplementary material for more results.

6. Conclusion

We have presented a novel approach to the classical problem of converting a rasterized floorplan image into a vector-graphics representation. Instead of relying on low-level image processig heuristics, our approach invents two intermediate representations encoding both geometric and semantic information of a floorplan image. We employ a neural architecture to convert an input rasterized image into a set of junctions and per-pixel semantic classification scores as the first representation. Integer programming then aggregates junctions into line or box primitives as the second representation, while ensuring topologically and geometrically consistent result. We believe that the paper makes a key milestone in the literature with a novel approach achieving a significant performance boost, a large-scale benchmark for evaluation, and a large corpus of floorplan vector data and popup 3D models, opening potentials for a new family of big-data Computer Vision research.

7. Acknowledgement

This research is partially supported by National Science Foundation under grant IIS 1540012 and IIS 1618685, Google Faculty Research Award, and Microsoft Azure Research Award. Jiajun Wu is supported by an Nvidia fellowship. We thank Nvidia for a generous GPU donation.

References

[1] Autocad. http://www.autodesk.com/products/autocad/overview. 1

[2] Gurobi. http://www.gurobi.com. 4

[3] Homestyler. http://www.homestyler.com. 1

[4] LIFULL HOME'S dataset. http://www.nii.ac.jp/dsc/idr/next/homes.html. 6

[5] Sketchup. https://www.sketchup.com. 1

[6] S. Ahmed, M. Liwicki, M. Weber, and A. Dengel. Improved automatic analysis of architectural floor plans. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 864–869. IEEE, 2011. 2, 6, 8

[7] A. Bulat and G. Tzimiropoulos. Human pose estimation via convolutional part heatmap regression. In *European Conference on Computer Vision*, pages 717–732. Springer, 2016. 4

[8] G. Chen, J. Kua, S. Shum, N. Naikal, M. Carlberg, and A. Zakhor. Indoor localization algorithms for a human-operated backpack system. In *3D Data Processing, Visualization, and Transmission*, page 3. Citeseer, 2010. 2

[9] S. Choi, Q.-Y. Zhou, and V. Koltun. Robust reconstruction of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5556–5565, 2015. 2

[10] H. Chu, D. Ki Kim, and T. Chen. You are here: Mimicking the human thinking process in reading floor-plans. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2210–2218, 2015. 2

[11] L.-P. de las Heras, S. Ahmed, M. Liwicki, E. Valveny, and G. Sánchez. Statistical segmentation and structural recognition for floor plan interpretation. *International Journal on Document Analysis and Recognition (IJDAR)*, 17(3):221–237, 2014. 2

[12] L.-P. de las Heras, O. Terrades, S. Robles, and G. Sánchez. Cvc-fp and sgt: a new database for structural floor plan analysis and its groundtruthing tool. *International Journal on Document Analysis and Recognition*, 2015. 2

[13] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski. Reconstructing building interiors from images. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 80–87. IEEE, 2009. 2

[14] L. Gimenez, J.-L. Hippolyte, S. Robert, F. Suard, and K. Zreik. Review: reconstruction of 3d building information models from 2d scanned plans. *Journal of Building Engineering*, 2:24–35, 2015. 2

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 4

[16] S. Ikehata, H. Yang, and Y. Furukawa. Structured indoor modeling. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1323–1331, 2015. 2

[17] C. Liu, A. G. Schwing, K. Kundu, R. Urtasun, and S. Fidler. Rent3d: Floor-plan priors for monocular layout estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3413–3421, 2015. 2, 8

[18] S. Macé, H. Locteau, E. Valveny, and S. Tabbone. A system to detect rooms in architectural floor plan images. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, pages 167–174. ACM, 2010. 2

[19] R. Martin-Brualla, Y. He, B. C. Russell, and S. M. Seitz. The 3d jigsaw puzzle: Mapping large indoor spaces. In *European Conference on Computer Vision*, pages 1–16. Springer, 2014. 2

[20] C. Mura, O. Mattausch, A. J. Villanueva, E. Gobbetti, and R. Pajarola. Automatic room detection and reconstruction in cluttered indoor environments with complex room layouts. *Computers & Graphics*, 44:20–32, 2014. 2

[21] K. Tombre, S. Tabbone, L. Pélissier, B. Lamiroy, and P. Dosch. Text/graphics separation revisited. In *International Workshop on Document Analysis Systems*, pages 200–211. Springer, 2002. 2

[22] S. Wang, S. Fidler, and R. Urtasun. Lost shopping! monocular localization in large indoor spaces. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2695–2703, 2015. 2

[23] E. Wijmans and Y. Furukawa. Exploiting 2d floorplan for building-scale panorama rgbd alignment. *arXiv preprint arXiv:1612.02859*, 2016. 2

[24] F. Yan, L. Nan, and P. Wonka. Block assembly for global registration of building scans. *ACM Transactions on Graphics (TOG)*, 35(6):237, 2016. 2