WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

AI-Based Automated House Floor Plan Generation System Using Graph-Constrained GAN and Computational Geometry

[1]A. L. Narayana Reddy, [1]B. Nikhila, [1]B. Prasanna Kumar, [1]D. Jaswanth, [2]Dr. S. Akila Agnes

[1]Students, Department of Computer Science and Engineering (CSE), GMR Deemed to be University
[2]Associate Professor, Department of Computer Science and Engineering (CSE), GMR Deemed to be University

ABSTRACT:

Designing a residential floor plan is slow and depends heavily on expert experience, while deep-learning generators usually output raster masks that cannot be measured, costed or visualised directly. This paper presents an AI-based automated house floor plan generation system that combines a pretrained graph-constrained generative model, HouseGAN++, with computational geometry and rule-based validation. From a plot and a list of required rooms, the system builds a room-relationship graph, generates several candidate layouts, repairs their geometry, scores them and selects a few distinct layouts. The chosen layout is stored as a single canonical description from which a furnished 2D plan, an interactive 3D model, a cost estimate and a PDF drawing are produced consistently. A sample run on a 40 ft × 60 ft plot shows that the system produces valid, measurable and furnished plans. The work shows that generative models become practical design tools when they are wrapped in graph, geometry and rule-based validation stages.

Index Terms *- Floor plan generation, HouseGAN++, Graph-conditioned GAN, Computational geometry, Binary Space Partitioning, Vastu Shastra, 3D visualisation, Bill of Quantities.*

I. Introduction

The floor plan is one of the earliest and most consequential decisions in a house-building project. It fixes the number, size and position of the rooms, decides how occupants move between them, determines how daylight and ventilation reach each space, and finally sets the quantity of material that has to be bought and assembled. Traditionally an architect or draftsperson iterates on a drawing with the client while balancing plot dimensions, family preferences, local rules and budget. The process is slow, depends on individual experience, and usually yields only a handful of alternatives.

Deep generative models can learn the regularities of human-designed layouts from large datasets and sample new ones automatically. Models such as House-GAN++ [1] generate rooms conditioned on a graph that states which rooms must be adjacent. The output of such a network, however, is a set of low-resolution raster masks. Masks are not drawings: they have no measurable walls, rooms may overlap or leave gaps, a room may be impossible to reach from the entrance, and nothing guarantees that the result fits an irregular plot. Between a generated image and a plan a builder can use lies a substantial engineering gap.

This work addresses that gap. Given the dimensions of a plot (or a polygonal boundary, or an uploaded survey sketch) and a list of required rooms, the system produces several valid, measurable and furnished floor plans, an interactive 3D model of the selected plan, a Vastu Shastra compliance report, an itemised BOQ cost estimate and a print-ready vector PDF. A pretrained generative model supplies variety, while computational geometry, graph theory and rule-based validation supply correctness. All outputs are derived from one authoritative plan description, so the 2D drawing, the 3D model and the cost estimate cannot disagree with one another.

The main contributions of this paper are: a hybrid layout generator that combines a pretrained graph-conditioned GAN with a deterministic fallback solver; geometry-repair and validation stages that turn generated masks into valid, measurable plans; and a single canonical plan from which the 2D drawing, 3D model, cost estimate and PDF are derived.

The rest of the paper is arranged as follows. Section II reviews related work and states the research gap. Section III describes the methodology, covering the workflow, system architecture, working procedure, models and outputs. Section IV presents the results and discussion, and Section V gives the conclusion.

II. Literature Review

Early computational approaches treated floor-plan design as an optimisation problem. Merrell et al. [2] sampled a room programme from a probabilistic model learned from real buildings and arranged the rooms by stochastic optimisation against architectural guidelines. Such methods give clean rectilinear layouts and respect explicit rules, but their quality depends on hand-crafted cost terms and the search must be repeated for every new set of requirements. Rule-based space partitioning, which recursively divides a region into sub-regions, is similarly deterministic and fast [3], [4], and is the conceptual ancestor of the fallback solver used here.

Large collections of real plans changed the field. Wu et al. [5] introduced the RPLAN dataset of human-designed residential plans together with a data-driven method that predicts room placement conditioned on the boundary and then refines it geometrically. Graph2Plan [6] conditions generation on both the building boundary and a user-specified layout graph, encoded by a graph neural network and a convolutional network, so that users can edit the graph to steer the result. House-GAN [7] proposed a relational GAN whose generator and discriminator follow the input adjacency graph, with information flowing between connected rooms through convolutional message passing based on Conv-MPN [8]. House-GAN++ [1] extends this idea by combining a graph-constrained relational GAN with a conditional GAN so that a previous layout can act as the constraint for the next generation step; an official implementation and pretrained checkpoint are available [9].

Generated or scanned plans are raster images, whereas construction, costing and 3D modelling need vector data. Liu et al. [10] converted raster plans to vector graphics using deep junction and primitive detection reconciled by integer programming. The present system faces the related task of turning generated room masks into exact polygons but uses classical computational-geometry operations instead of a second neural network, which keeps the result explainable and verifiable.

The reviewed generative models concentrate on producing plausible room arrangements. By themselves they do not guarantee output that is metrically exact, free of overlaps, fully reachable, valid on an irregular plot, furnished, visualised in 3D or costed. The research gap is therefore the absence of an end-to-end pipeline around a pretrained generator that turns its output into a validated, constructible and costed plan, with optional support for culturally specific preferences such as Vastu Shastra. Table 1 summarises the main works and their relation to this paper.

Table 1: Techniques in related work and their relevance to the proposed system

Work	**Core technique**	**Relevance to this work**
Merrell et al. [2]	Probabilistic room programme and stochastic layout optimisation	Shows the value of explicit architectural rules; slow search per query
Wu et al. (RPLAN) [5]	Dataset of real plans; learned room placement plus refinement	Source of the data behind the pretrained checkpoint
Graph2Plan [6]	Layout graph and boundary encoded by GNN and CNN; editable graph	Supports the graph-first design of the room-relationship stage
House-GAN [7]	Relational GAN with graph-structured generator and discriminator	Introduces room-wise mask generation conditioned on a graph
House-GAN++ [1]	Graph-constrained relational GAN combined with conditional GAN	Pretrained generator used as the AI layout engine
Conv-MPN [8]	Convolutional message passing on feature maps	Basis of the message-passing blocks inside the generator
Liu et al. [10]	Raster-to-vector floor-plan conversion	Motivates vector realisation of raster output

III. Methodology

3.1 Workflow architecture

The complete workflow is shown in Fig. 1. It starts with user input and processing of the plot and requirements, followed by room-graph generation, AI layout generation, geometry repair, validation and scoring, ranking and selection of the best layouts, and the final floor-plan specification, from which the 2D plan, 3D model, cost estimate and PDF are produced.

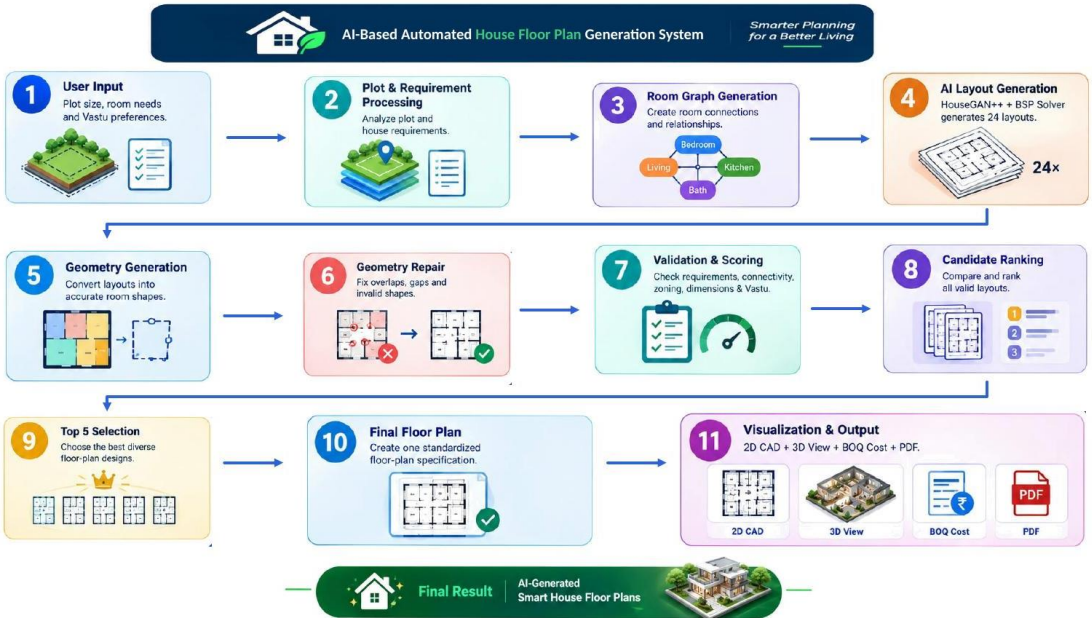


Fig. 1: workflow architecture of the AI-based floor plan generation system (eleven steps from user input to final output)

3.2 System architecture

The platform follows the three-tier architecture of Fig. 2, in which each tier has its own job. The three services communicate through REST-style JSON interfaces [11]. The frontend (React 18 with Vite) renders the interface, the interactive 2D canvas and the 3D viewer and keeps session state in a Zustand store. The backend (Node.js and Express [12] with MongoDB [13]) authenticates users, stores projects, plots, generations and plan versions, and forwards generation, export and costing requests to the AI service. The AI and geometry service (Python with FastAPI [14]) hosts the PyTorch model and all geometric computation. Separating the tiers means that the AI component can be improved or scaled without touching the user interface.

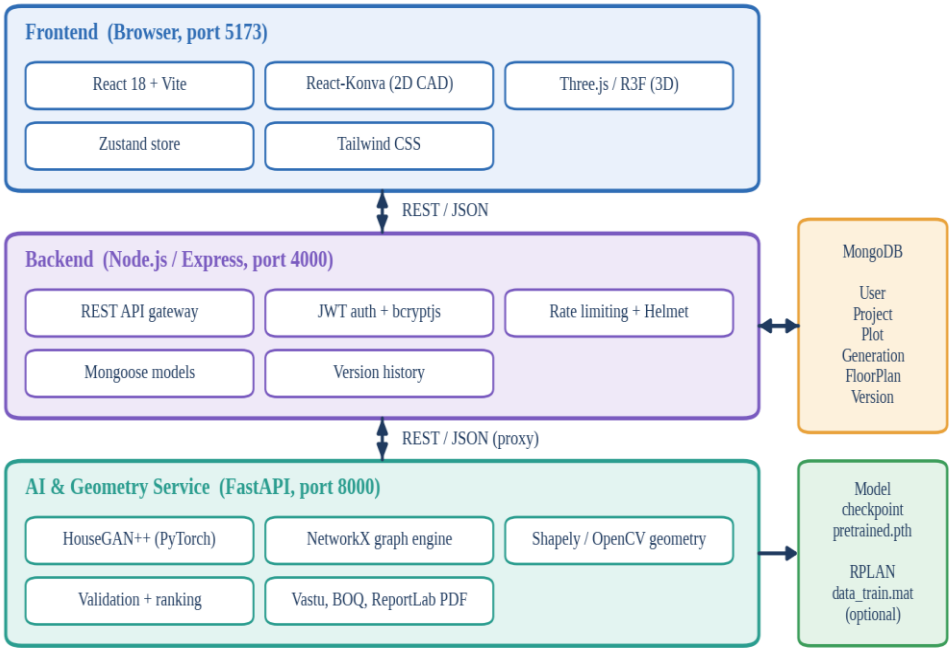


Fig. 2: three-tier architecture of the system (frontend, backend gateway and AI/geometry service)

Inside the AI service the data flow is: requirements and plot → room graph → hybrid generator (HouseGAN++ or BSP solver) → geometry repair → validation and scoring → diversity ranker (Top 5) → canonical floor plan → 2D, 3D, cost and PDF outputs. Access control uses password hashing, JSON Web Tokens [15] and request rate limiting.

3.3 Working procedure

1. Plot and requirement input: the plot is entered as width and length, as a polygon, or as an uploaded survey image, which is processed with standard image-processing techniques [16] to recover the buildable boundary; the user also states the rooms required and may switch the Vastu preference on.

2. Room graph construction: the requirements are compiled into a NetworkX graph [17] whose nodes are rooms and whose edges are required, preferred, access or forbidden relations (Fig. 3). A reachability test from the entrance checks that every room can be reached.

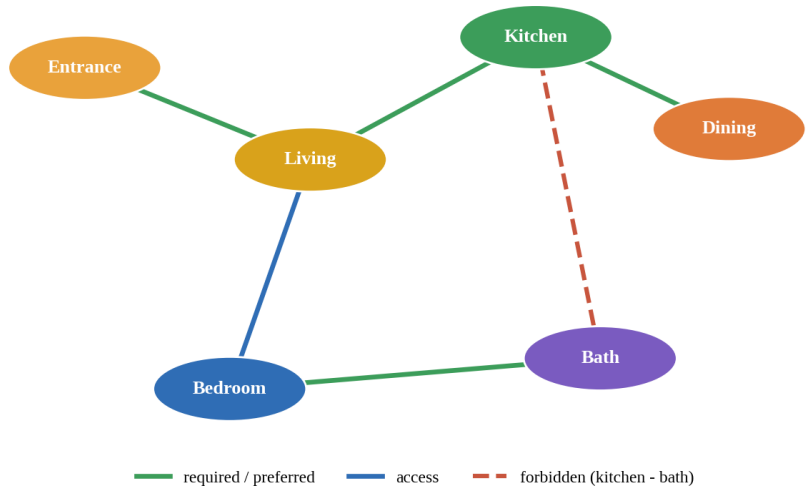


Fig. 3: example room-relationship graph with edge semantics

3. Candidate generation: several candidate layouts are produced per request. HouseGAN++ is used by default; for irregular plots, uploaded images, or machines without a checkpoint or GPU, a deterministic BSP solver is used instead.

4. Geometry repair: generated masks or cells are converted to exact polygons, simplified, aligned to the axes, cleared of overlaps and aligned along shared walls.

5. Validation and ranking: each candidate is scored on seven criteria and an optional Vastu criterion; degenerate candidates are removed and a diversity-aware selection keeps the best five distinct layouts.

6. Canonical plan and outputs: the chosen layout is stored as one canonical plan from which the furnished 2D drawing, the 3D scene, the cost estimate and the PDF are derived (Fig. 4). Each saved edit creates a new plan version.

3.4 Models and technologies used

HouseGAN++ generator. HouseGAN++ builds on generative adversarial networks [18], in which a generator is trained against a discriminator, and it is conditioned here on a room graph. It is a relational generator that produces one mask per room, with connected rooms exchanging information through convolutional message passing [8]. The pretrained checkpoint released by the authors [9], trained on RPLAN [5], is used for inference only and no retraining is performed. The model runs in PyTorch [19] on a GPU when available and on the CPU otherwise.

BSP constraint solver. The fallback solver recursively splits the buildable polygon into sub-regions in proportion to room area weights [3], [4]. Rooms are ordered by a functional zoning hierarchy in which public rooms come first, service rooms next and private rooms last. By construction the resulting cells do not overlap and exactly tile the region.

Geometry engine. The generated masks are converted to valid polygons [20], scaled to the plot dimensions, simplified and aligned to the horizontal and vertical axes using standard image and geometry operations [16]. Because every mask is decoded independently, rooms may overlap; the contested region is then assigned to the smaller room and removed from the larger one, and the procedure is repeated until no overlaps remain. Nearly colinear wall coordinates are finally merged so that shared walls coincide exactly.

Validation engine. Every candidate is scored on seven objective criteria: requirements, graph adjacency, circulation, geometry, zoning, proportions and plot utilisation. An optional Vastu criterion is added when selected. The criteria are combined by a weighted sum into an overall score on a 0–100 scale.

Diversity-aware ranking. Candidates with severe geometric defects are first removed. The remaining ones are sorted by score, and a greedy selection keeps the highest-scoring candidates whose layouts are sufficiently different from those already chosen, so that the user is offered five distinct designs rather than minor variants of one.

Vastu Shastra engine. When the Vastu option is on, the plot is divided into nine zones: eight compass sectors and a central open core. The engine checks that the kitchen lies in the south-east zone, that the master bedroom lies in the south-west zone and that the central core stays free of bathrooms and the

kitchen. These rules are treated as user-selectable preferences and not universal requirements, and an optimiser only suggests better placements without altering the generated plan.

3.5 Canonical floor plan and output generation

The central design decision is the canonical floor plan, a single JSON document from which every consumer reads (Fig. 4). It holds the plot, the rooms with their polygons, doors and windows, the walls, the furniture, the validation scores and generation metadata. Doors and windows are placed on the walls automatically, and furniture is placed by room type while keeping doorways clear.

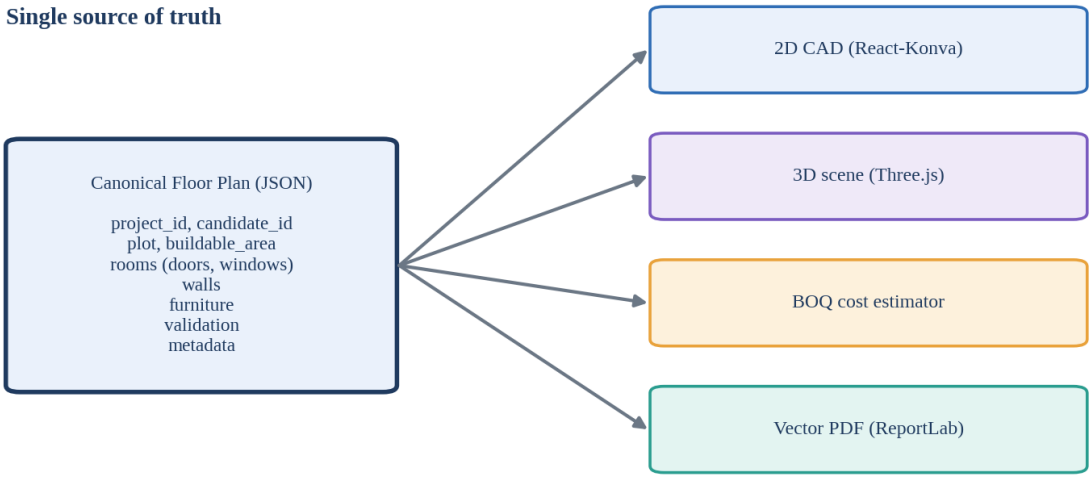


Fig. 4: the canonical floor plan as the single source of truth for all outputs

The 2D plan is drawn on an HTML5 canvas with React-Konva [21], [22] using architectural conventions such as wall thickness, door swing arcs, window symbols and dimension lines. The 3D viewer is built on Three.js through React Three Fiber [22], [23] and converts the same plan by a deterministic transformation, so that each room becomes a floor mesh and each wall an extruded block, shown either as a low cutaway dollhouse or at full height.

The Bill of Quantities estimator measures the plan itself instead of multiplying area by a rough rate: it uses the floor area, wall lengths, door and window counts and room types to build an itemised estimate of materials, additional costs and labour. A vector PDF drawing of the plan can also be exported.

3.6 Data sources

The system does not train a model of its own. The generator checkpoint was trained on the RPLAN dataset [5], a collection of human-designed residential layouts. Room requirements and plots are supplied by the user at run time.

3.7 Evaluation plan

Because the work is built around a pretrained model, it is evaluated by the validation score of the ranked candidates, by inspecting the geometry and outputs of a sample run, and by automated unit and integration tests.

IV. Results and Discussion

4.1 Results

Sample run. The sample run uses a plot of 40 ft × 60 ft with the road on the north side. The requirements are two bedrooms, a living room, a kitchen, a dining room, two bathrooms and a balcony.

Candidate ranking. After repair and validation the Top 5 layouts were selected (Fig. 5), and the highest-ranked layout scored 91.4 out of 100. In this layout every requested room is present, the required adjacencies hold and every room is reachable from the entrance.

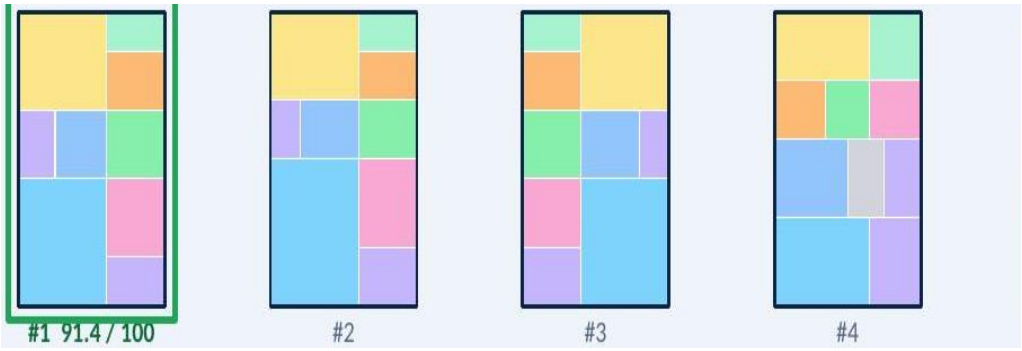
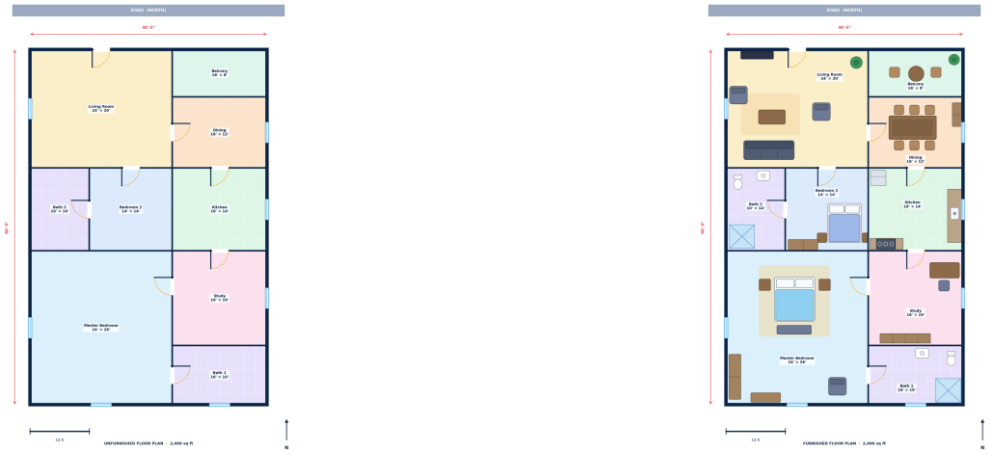


Fig. 5: ranked layouts for the 40 ft × 60 ft sample plot, #1 to #4 shown (illustrative); layout #1 scored 91.4 / 100

Two-dimensional plan. The chosen layout is first available as an unfurnished plan and is then furnished automatically (Fig. 6). The furnished plan shows furniture placed by room type, doors with swing arcs, windows, wall thickness, dimension lines and a scale bar. The user can pan, zoom, move and rotate furniture, and export the plan as a vector PDF.



(a) Unfurnished plan (b) Furnished plan

Fig. 6: chosen layout on the 40 ft × 60 ft plot, unfurnished and furnished (illustrative)

Three-dimensional model. The same plan data produce the dollhouse model of Fig. 7, so the 2D and 3D views always agree. The cutaway mode allows the interior to be inspected from above and the full mode shows the building at full height.



Fig. 7: interactive 3D dollhouse model in cutaway mode (illustrative render)

Bill of Quantities. The cost module derives the quantities directly from the measured plan geometry, so the estimate changes automatically when the plan changes.

Verification. Automated unit tests for the main modules and an end-to-end integration test from user input to the final plans pass in full.

4.2 Discussion

The results support the main objectives of the work. Generating several candidates and filtering them is more effective than trying to make one generation perfect, because the validator can cheaply discard defective ones. Using the graph both to condition the generator and to verify the result gives a consistent notion of a correct layout, and deriving all outputs from the canonical plan removes synchronisation errors between drawings and estimates. The same pipeline also handles polygonal plots and plot images through the BSP solver.

The limitations of the present work should be noted. The generator is the published pretrained checkpoint and is not fine-tuned, so the variety of layouts is bounded by that model. No quantitative comparison with other generators and no user study have been carried out yet, several figures are illustrative renderings, and only single-storey layouts are supported.

V. Conclusion

This paper presented an AI-based automated house floor plan generation system that bridges generative AI and constructible architectural geometry. By surrounding a pretrained HouseGAN++ generator with a room-graph compiler, a fallback solver, geometry repair, validation and diversity-aware ranking, the system turns raster masks into valid, measurable and furnished plans, with a synchronised 2D drawing, 3D model, cost estimate and PDF.

Future work includes multi-floor layouts, automated service routing, building-code checks, a quantitative comparison with other generators and a user study.

Acknowledgment

The authors would like to thank Dr. S. Akila Agnes, Associate Professor, Department of Computer Science and Engineering, GMR Deemed to be University, for her guidance and support throughout this project.

References

[1] N. Nauata, S. Hosseini, K.-H. Chang, H. Chu, C.-Y. Cheng, and Y. Furukawa, "House-GAN++: Generative adversarial layout refinement network towards intelligent computational agent for professional architects," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), 2021.

[2] P. Merrell, E. Schkufza, and V. Koltun, "Computer-generated residential building layouts," ACM Trans. Graph., vol. 29, no. 6, 2010.

[3] H. Fuchs, Z. M. Kedem, and B. F. Naylor, "On visible surface generation by a priori tree structures," in Proc. ACM SIGGRAPH, 1980.

[4] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, Computational Geometry: Algorithms and Applications, 3rd ed. Berlin, Germany: Springer, 2008.

[5] W. Wu, X.-M. Fu, R. Tang, Y. Wang, Y.-H. Qi, and L. Liu, "Data-driven interior plan generation for residential buildings," ACM Trans. Graph., vol. 38, no. 6, 2019.

[6] R. Hu, Z. Huang, Y. Tang, O. van Kaick, H. Zhang, and H. Huang, "Graph2Plan: Learning floorplan generation from layout graphs," ACM Trans. Graph., vol. 39, no. 4, 2020.

[7] N. Nauata, K.-H. Chang, C.-Y. Cheng, G. Mori, and Y. Furukawa, "House-GAN: Relational generative adversarial networks for graph-constrained house layout generation," in Proc. Eur. Conf. Comput. Vis. (ECCV), 2020.

[8] F. Zhang, N. Nauata, and Y. Furukawa, "Conv-MPN: Convolutional message passing neural network for structured outdoor architecture reconstruction," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), 2020.

[9] N. Nauata, "houseganpp: House-GAN++ official implementation and pretrained checkpoint," GitHub repository. [Online]. Available: https://github.com/ennauata/houseganpp

[10] C. Liu, J. Wu, P. Kohli, and Y. Furukawa, "Raster-to-vector: Revisiting floorplan transformation," in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), 2017.

[11] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Dept. Inf. Comput. Sci., Univ. California, Irvine, CA, USA, 2000.

[12] "Express: Fast, unopinionated, minimalist web framework for Node.js." [Online]. Available: https://expressjs.com

[13] MongoDB, Inc., "MongoDB manual." [Online]. Available: https://www.mongodb.com/docs/manual/

[14] S. Ramírez, "FastAPI framework, high performance, easy to learn, fast to code." [Online]. Available: https://fastapi.tiangolo.com

[15] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," Internet Engineering Task Force, RFC 7519, May 2015.

[16] G. Bradski, "The OpenCV library," Dr. Dobb's Journal of Software Tools, 2000.

[17] A. A. Hagberg, D. A. Schult, and P. J. Swart, “Exploring network structure, dynamics, and function using NetworkX,” in Proc. 7th Python Sci. Conf. (SciPy), 2008, pp. 11–15.

[18] I. Goodfellow et al., “Generative adversarial nets,” in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 27, 2014.

[19] A. Paszke et al., “PyTorch: An imperative style, high-performance deep learning library,” in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 32, 2019.

[20] S. Gillies et al., “Shapely: Manipulation and analysis of geometric objects,” 2007–. [Online]. Available: https://github.com/shapely/shapely

[21] “Konva.js: HTML5 2D canvas library.” [Online]. Available: https://konvajs.org

[22] Meta Platforms, Inc., “React: The library for web and native user interfaces.” [Online]. Available: https://react.dev

[23] “three.js: JavaScript 3D library.” [Online]. Available: https://threejs.org