A Development Model of Cross Platform Mobile Reconstructing Software for 3D Structures from 2D Vector Floor Plans

Toyin Johnson[1], Gilbert George[2], and Amit Mitshra[3]
[1-3]Department of Computer Science, BAZE University, Abuja, Nigeria
Corresponding author: gilbert.george@bazeuniversity.edu.ng

ABSTRACT: This research focuses on the analysis and development of a cross-platform application built using React for the autonomous reconstruction of 3D building models from 2D vector floor plans. The system interprets both the geometric and semantic content of the plans. We present a simple and robust method for transforming architectural drawings into 3D building representations. Since these plans are originally designed for human interpretation, they pose challenges for automated 3D model generation. To address this, we propose a fully automated plan-cleaning algorithm that enables the accurate construction of 3D models from floor plans. The effectiveness of our approach is demonstrated through experiments that convert floor designs into detailed 3D models with high efficiency.

KEYWORDS: Android, Architectural plans, Computer- Aided-Design ,3D model creation, Vector Image Conversion, three js.

I. INTRODUCTION

2D architectural floor plans are common and extensively utilized method for architects to convey their designs. A true 2D vector architecture floor layout is shown in Fig. 1. 3D building models are simple to use and offer a wide range of applications. Building usefulness and safety can be greatly improved by using 3D building models for fire simulations [1]. The building becomes more realistic through rendering on 3D building models. Another quickly rising trend is the use of virtual 3D surroundings for navigation. Nevertheless, manually building a 3D model of a collection of floor plans is difficult and takes time and effort [2]. Therefore, it is crucial that 2D floor layouts are automatically converted into 3D models. In real life, a building is typically made up of a variety of useful places, including rooms, hallways, etc. Walls and apertures such as doors and windows divide functional spaces. Functional areas are shown by loops that are made up of wall lines and apertures, matching the 2D architectural floor plan. Hence, there are three key activities involved in converting a 2D architectural floor plan to a 3D model:

(a). identifying structural elements such walls, doors, and windows [1], [3];
(b). loop searching to obtain spatial data [1];
(c). extrusion to produce final 3D models.

We presume that a typical architectural floor plan has the following features because vector architectural floor plans from real world come in a variety of styles: The graphic consists of lines, arcs, and text. In our preprocessing, POLYLINEs and other sorts of visual elements are first automatically transformed into lines and arcs. It is impossible to isolate structural elements like walls and openings like doors. They must connect to, be close to, or contain structural components.

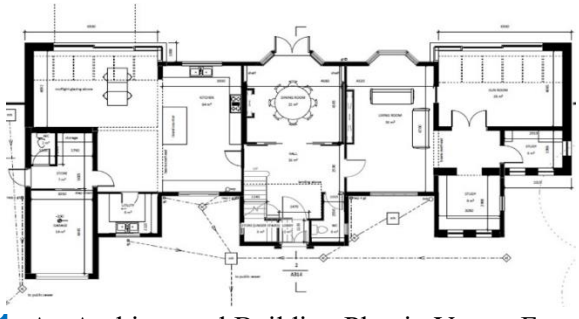


Fig. 1. An Architectural Building Plan in Vector Format

A highly automated method to create 3D models from 2D floor blueprints was put forth by Or et al. [4]. The system transforms a set of floor plan raster pictures into a set of vector images using them as input. Identification of door symbols is done using symbol recognition and windows, too. After completing Symbol recognition, the system creates 3D buildings and uploads them into the Genesis 3D game engine. By parsing floor layouts into connected segments, the technology creates 3D buildings by analyzing the relationships between these connected segments.
Dosch et al.'s [2] comprehensive technique for reconstructing 3D buildings from scanned 2D architectural plans. The system is broken down into two steps: 2D modeling and feature extraction. In the 2D modeling step, a powerful graphics recognition algorithm is described. To do this, the raster 2D image is divided into tiles, each of which is processed separately before being combined after vectorization. The system uses a skeleton-based approach for feature extraction, where extracted lines are represented as segments using a polygon approximation technique. They then suggest a 3D modeling technique that is utilized to match the rebuilt floors. Additionally, the system offers a user interface that is adaptable and capable of supporting human interaction.
A system that presents a four-phase construction process was reported by Horna et al. in [5]. Their technique begins by analyzing 2D edges to eliminate geometrical irregularities. The creation of topologies utilizing semantic data and extrusion for 3D building construction follow. The final model is built by superimposing the floor over the upper and lower ceilings, which are connected by stairways. Incidence and adjacency links between elements are expressed in the system's entire topological model. Additionally, the system associates semantic data with each volume in order to identify structures like rooms and walls. During extrusion to 3D, the system takes the semantic information into consideration. In the author's [3] employed 3DPlanNet Ensemble approaches that used rule-based heuristic methods to learn with just 30 floor plan photos. Using a vast amount of learning data, this method experimentally yielded object accuracy that was comparable to that of a prior study and a wall accuracy of more than 95%. Additionally, structural data was constructed as 3D models with layers segregated for each object, such as walls, doors, windows, and rooms, and 2D drawings without dimension information were transformed into ground truth sizes with an accuracy of 97% or higher.
A comprehensive system for automatic floor plan analysis was put forth by Ahmed et al. [1]. The system aids in the use and enhancement of current processing techniques, and it also introduces preprocessing techniques that enhance system performance. The separation of thick, medium, and thin lines, as well as the elimination of parts that are outside the convex hull of the outer walls, are some of the approaches used. In this paper [6], the goal of this project was to develop a framework for CNN (Convolution Neural Networks)-based floor plan analysis of complex buildings with diverse scales. The framework enables the proposed CNN model to analyze different size or high-resolution inputs, which is a hurdle for existing approaches, by breaking a floor plan into a set of normalized patches. Each patch's building objects were identified by the model, which then combined them into a single outcome by multiplying the relevant translation matrix. The discovered building objects were then vectorized in order to make them compatible with 3D modeling. In spite of the intricacy of the data used, our framework performed similarly to previous research in terms of detection rate (87.77%) and recognition accuracy (85.53%).

II. METHODOLOGY

Our 3D model development process starts with 2D vector versions of architectural blueprints. The application was developed using React JS which allows for a cross platform development. Plan cleaning is carried out by the Potrace library[7]. The library generates a smooth approximate vector representation of the pixelated image then aligns and stacks the cleaned floor plans in the building at the appropriate heights.

Volume 1, Issue 1, 2025
https://doi.org/10.5281/zenodo.17537663

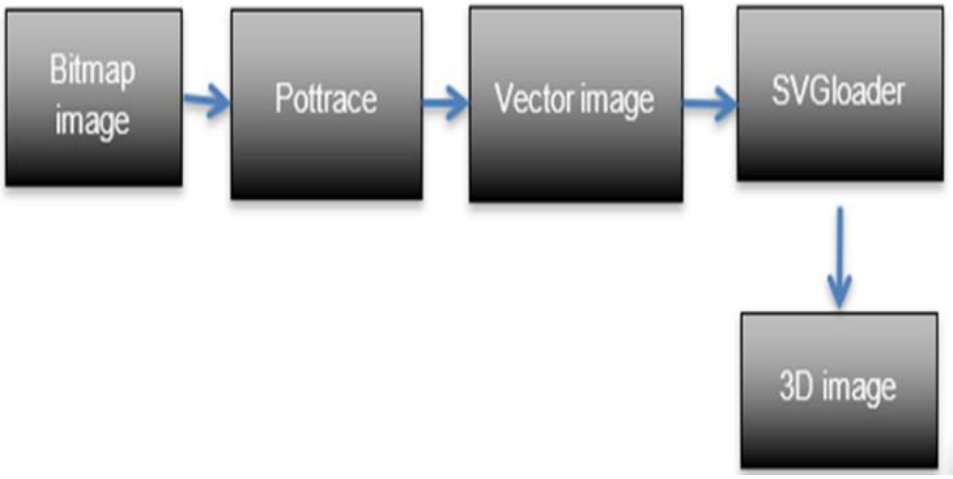


Fig. 2. Methodology for the Application

Then the image is then passed to another library called. Three.js [7], [8], we used specifically the SVGLoader class to read the generated SVG string. Once read a user defined function to extrude the path was used to create the 3-dimensional image. The conversion from JPG to SVG is seen in Fig.3.

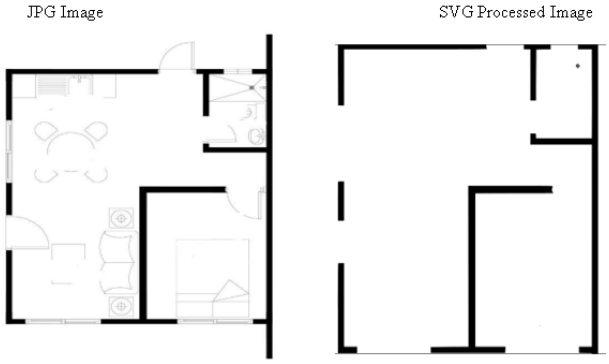
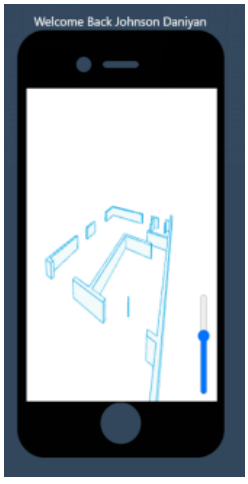


Fig. 3. The JPG to SVG Conversion using Pottrace



Volume 1, Issue 1, 2025
https://doi.org/10.5281/zenodo.17537663

Fig. 4. The Extrusion of the 3D Image from the SVG Image

A. Algorithm 1

1. Load the bitmap image into memory.
2. Convert the image to grayscale if it's not already in grayscale.
3. Apply a thresholding operation to convert the grayscale image into a binary image, where each pixel is either black or white based on a threshold value. This step helps in separating the foreground (object) from the background.
4. Initialize an empty set of curves or paths to store the vector representation of the image.
5. For each black pixel in the binary image:
    a. Start a new curve or path.
    b. Traverse the connected black pixels using a region-growing or connected-component labelling algorithm. This process helps in identifying the boundaries of the object.
    c. Store the coordinates of the boundary pixels as points in the current curve or path.
6. After traversing all the black pixels, you'll have a set of curves or paths that represent the object(s) in the image.
7. Apply curve simplification techniques, such as the Douglas-Peucker algorithm, to reduce the number of points in each curve. This step helps in smoothing out the curves and reducing the file size.
8. Optionally, apply curve optimization techniques to further refine the curves and remove unnecessary control points.
9. Output the resulting vector image in a suitable file format (e.g., SVG) that preserves the curve information.
10. End

B. Algorithm 2

1. Load the SVG file into memory.
2. Parse the SVG file to extract the necessary information, such as shapes, paths, and curves.
3. Determine the desired depth or height of the 3D representation. This will determine how much the SVG shapes will be extruded.
4. For each shape or path in the SVG:
    a. Convert the 2D shape or path into a set of 3D coordinates by adding a constant value for the extrusion depth.
    b. Create a set of triangular faces that connect the corresponding vertices of the 3D shape or path. You can use triangulation algorithms like Delaunay triangulation or ear clipping to generate the faces.
5. Optionally, apply smoothing or subdivision algorithms to refine the geometry and create smoother surfaces. Techniques like Catmull-Clark subdivision or Loop subdivision can be used.
6. If there are multiple shapes or paths in the SVG, you may need to merge or combine them into a single 3D object. This can be done by merging their vertices and faces.
7. Apply any additional transformations or operations you desire, such as scaling, rotating, or translating the 3D object.
8. Output the resulting 3D representation in a suitable file format, such as OBJ or STL, that can be used in 3D modelling or rendering software.
9. End

III. RESULTS AND DISCUSSION

With a collection of floor plans from various buildings, we display the findings of our system. On an android phone, we created our system entirely for. The development of the software was done on a laptop with an Intel Core i7 2.4 GHz processor and 8 GB memory. We installed our software on an Iphone 8 emulator phones with 4Gb memory.

Volume 1, Issue 1, 2025
https://doi.org/10.5281/zenodo.17537663

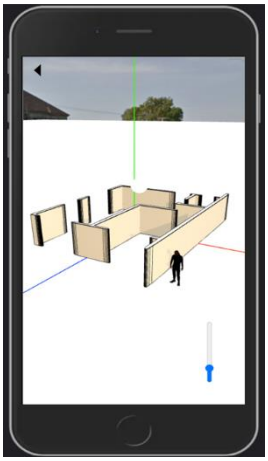


Fig. 5. Shows Application Running on an IPhone 8 Emulator

Building roofs were purposely left out of the data displayed below in order to better illustrate the results and highlight the intricacy of our models. Figure 5 depicts the 3D perspective of the vector image.

IV. CONCLUSION

We described a straightforward and reliable technique in this study for creating 3D models from a collection of 2D architecture floor plans. We demonstrated our algorithm, a key component of our system that enables us to produce 3D diagrams from floor plans before moving on to more generic operations like extrusion to build the 3D models. At the moment, our algorithm cannot develop 3D items like windows and doors which we hope to implement in the future. In the upcoming edition of our system, we also intend to provide support for textures. It is difficult to generate non-vertical complicated structures.

References

[1] J. Pandey and O. Sharma, “Fast and Robust Construction of 3D Architectural Models from 2D Plans.”

[2] P. Dosch, K. Tombre, C. Ah-Soon, and G. Masini, “A complete system for the analysis of architectural drawings,” *International Journal on Document Analysis and Recognition*, vol. 3, no. 2, pp. 102–116, 2000, doi: 10.1007/PL00010901/METRICS.

[3] S. Park and H. Kim, “3dplannet: Generating 3D models from 2d floor plan images using ensemble methods,” *Electronics (Switzerland)*, vol. 10, no. 22, Nov. 2021, doi: 10.3390/ELECTRONICS10222729.

[4] S. Or, K. Wong, Y. Yu, M. Chang, H. K.-P. Recognition, and undefined 2005, “Highly automatic approach to architectural floorplan image understanding & model generation,” *Citeseer*, Accessed: Jun. 03, 2023. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=bd3f155c73f4f176d96b90004b2e0ed55e104120

[5] S. Horna, G. Damiand, D. Meneveaux, Y. Bertrand, and S. Horna, “Building 3D indoor scenes topology from 2D architectural plans.” [Online]. Available: https://hal.science/hal-00337793

[6] H. Kim, S. Kim, and K. Yu, “Automatic Extraction of Indoor Spatial Information from Floor Plan Image: A Patch-Based Deep Learning Methodology Application on Large-Scale Complex Buildings,” *ISPRS Int J Geoinf*, vol. 10, no. 12, Dec. 2021, doi: 10.3390/IJGI10120828.

[7] A. Jain, C. Chen, T. Thormählen, D. Metaxas, and H. P. Seidel, “Multi-layer stencil creation from images,” *Computers and Graphics (Pergamon)*, vol. 48, pp. 11–22, Feb. 2015, doi: 10.1016/j.cag.2015.02.003.

[8] “Three.js Tutorial.” https://www.tutorialspoint.com/threejs/index.htm (accessed Jul. 04, 2023).
10.1109/IECON.2015.7392581.

https://doi.org/10.5281/zenodo.17537663