Towards Automatic 3D Reconstruction from 2D Floorplan Image

S. H. Or
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
{shor@cse.cuhk.edu.hk}

Abstract

Reconstruction of 3D model representation from a 2D image(s) is proven to be a difficult task. In this paper, we present a simple solution to the restricted case of generating model description of a building solely from its 2D floorplan. The motivation of this research is that i) a lot of existing buildings has only the floorplans remained, ii) In general a lot of efforts i.e human labor work, have to be made in order to build the 3D model. We proposed two approaches with different degree of automation: The first approach operates by an user analyzes and breaks down the floorplan into a number of building primitives. The coordinate information are then manually recorded and input to a program which will generates the 3D model descriptions i.e. a popular 3D model file such as *3D Studio* etc. Another approach applies image processing techniques in which the user selects and changes the lines in the floorplan into different color coded areas. These areas again represent different 3D primitives similar to the first approach. A raster to vector program then converts these primitives into 3D format. The latter approach involves only minimal human effort. Using our approaches, an ordinary user can produce a 3D model of a building in a matter of seconds. We present some of the results and discuss the relative merits of the two approaches. The presented algorithms are well suited for data generation in architectural walkthrough on old buildings and 3D games development.
keyword : 3D modeling, block-based approach, raster to vector conversion, color-coded blocks

1 Introduction

Given the rapid advances in modeling technologies in recent years, it is relatively easy for an average user to build a 3D model of a building using any popular modeling tools such as Maya or 3D Studio. However for buildings that exist years ago, what we got are only their floorplans, which is in general a 2D representation of the building under varying assumptions. A lot of such buildings are of historic as well as educational importance if one can navigate and examine its structure [1]. To "recover" such buildings in the virtual space, tedious modeling procedures are usually required.

A floorplan has the feature of abstracting a lot of spacial information in a 2D postulation. In general a floorplan consists of a number of lines of various widths postulating the top view of a building examined. The lines designate walls of different thickness. In actual buildings, especially those used for residential purpose, it happens that openings on the walls, i.e. windows, are needed. In order to cater for the above situation, a special signature, say a pair of thin lines is used to denote that an opening should be placed there. Depends on the building nature, usually a set of rules or signatures is used to represent the deviation from the normal simple wall cases.

With the aid of current modelling package, one have to trace the lines on the floorplan, which represents the walls and instruct the package to build the corresponding 3D representations. A typical way is to use a graphic-tablet-and-probe approach to manually trace the lines on the draw-

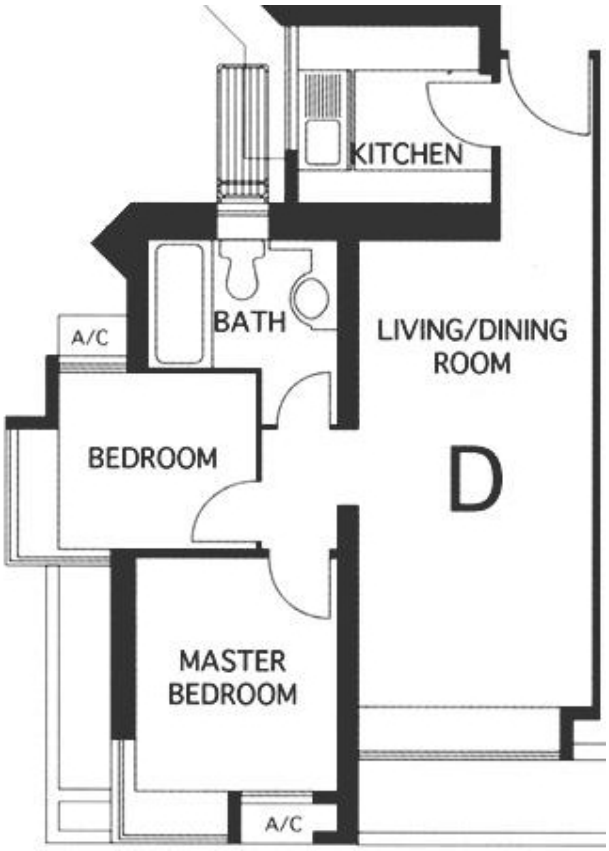

Figure 1: A typical floorplan of a building

ing so as to enter the coordinate information into the computer. During this process, the modeller has to take care of the various aspects of the plan by noting attributes, such as height, thickness, of the lines and tell the package such information. The result is that much time is needed for converting a floorplan into its 3D representation. Not to mention the more additional efforts to further refine the 3D models such as texturing and lighting. Another disadvantage is that a skilled personnel is required in order to convert the embedded spacial signature into corresponding 3D information.

2 Our Approach

In examining the layout of a floorplan, a line segment on a floorplan denotes a block oriented vertically in the 3D space. A simple approach is therefore to register the coordinates of the four points defining each line on a floorplan and generate the 3D representation using a normal in y-axis. By recursively processing all the lines in a floorplan, we should be able to generate the 3D description of the whole building.

This approach suffers from the following drawbacks. Firstly there are various line signatures inside a typical floorplan which designate different constructs in 3D space, for example a windowed block. Decomposing a floorplan solely into vertical blocks cannot faithfully reproduce the correct 3D representation. In addition, each vertical block generated typically will have faces that never be seen, for example, the base of a vertical block as well as part of the faces in a L-shaped lines which are being occluded by its orthogonally aligned neighbor. These faces will affect both the memory required during program execution and the rendering performance. Finally these redundant faces also create visual artifacts on most modern rendering engines.

2.1 Block Based Approach

It is inevitable that the vertical block based approach mentioned in previous section, in spite of the number of drawbacks, is feasible and easy to start with. We extend the idea of building blocks and come up with an approach which has the simplicity advantage but avoiding the drawbacks mentioned.

By examining extensively the different line signatures in a floorplan as in fig.1, we derive several basic building blocks encountered in a usual building. In our example, the target is the interior of a flat. We emphasize here what we presented is a framework and in practice more basic building blocks can be designed given the appropriate application arena. Moreover we believed that the blocks we are going to present include most of the elements required in a floorplan.

In the following discussions, we assumed that a floorplan denotes the top view of the building, which is the x-z plane and the vertical axis is the y-axis.

- The simplest elementary block is the vertical block, we termed it "*Box*". A *Box* can be fully described by 6 parameters(fig. 2):

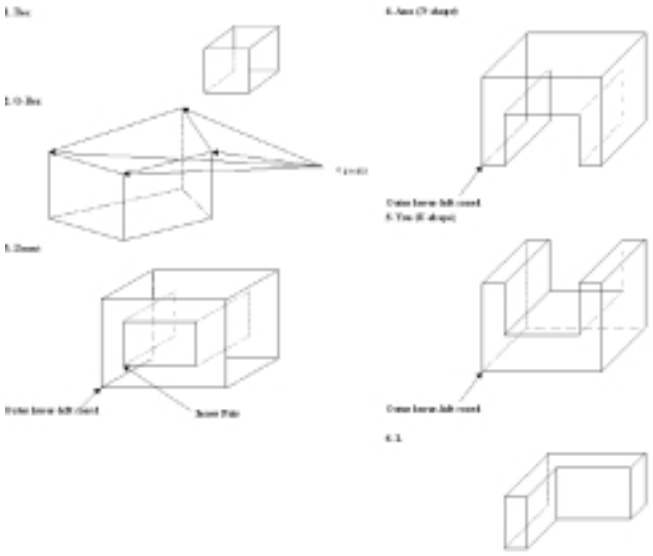

Figure 2: Various building blocks

The position of the reference point(3),
The dimensions, or size of the box in three axis(3).

- To cater for boxes rotated at an angle about the y-axis, we add a generalized box notion and termed it "*G-Box*". A *G-Box* is fully represented by 10 parameters (fig. 2):

  X-, & Z-coordinates for the four vertices(8), lower and upper Y-coordinates.

- Windows, or related structures, are popular in man-made building. To represent a window, we introduce the "*donut*" block. A *donut* is a block with an opening in its interior and is modeled by 10 parameters(fig. 2):

  The outer lower left coordinates(3) of the whole block,
  The dimensions of the whole block(3),
  The coordinates of the lower left corner of inner block(2),
  The dimensions of inner block(2) {See fig.2}.

- Door structures are also frequently seen in a floorplan. We introduce an "*Ann*" block (See fig.2) to model this. *Ann* block has fewer parameters (8) than *donut*:

  The lower left coordinates(3) of the whole block,
  The dimensions of the whole block(3),
  The dimensions of inner block(2).

- A simple extension of *Ann* is the "*You*" block, which is basically an inverted *Ann* (See fig.2):

- A combination of *Box* which is popular in floorplan construct is the *L*-shaped block (fig.2). An *L*-block is basically a subtraction of a smaller box from a larger one. We use 8 parameters for this block:

  The lower left coordinates(3) of the whole block,
  The dimensions of the whole block(3),
  The position of inner corner of L(2).

- As in *G-Box*, we have the generalized versions of all the above blocks which need more parameters to specify. As these are trivial extensions, we will not detail their implementations here.

The operations in utilizing the above building blocks to parameterize a floorplan involves treating the line complex of the whole floorplan as a combinations of the block types and decomposing them and recording the parameters. The detail algorithm works as follow:

Block Based Algorithm for building the 3D representation from a floorplan

1. Overlay a uniform grid on the floorplan image. The spacing of the grid can conveniently be chosen the same as the scale noted on the plan. In this case, the recovered 3D representation will thus have exactly the same scale as in human world.

2. Select a point on the grid as the origin. Manually record the coordinates of vertices of all lines in the plan.

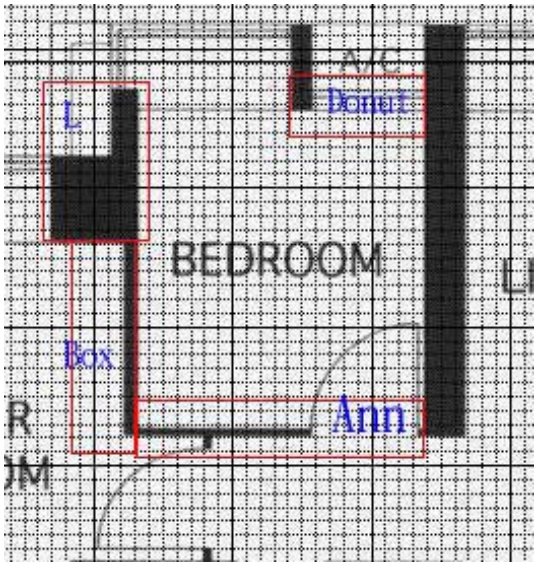

Figure 3: Breakdown of a portion of floorplan into blocks

3. Based on the different line signatures in the plan, map a suitable block type to the lines.

4. All the block information are then entered into a program which will generate the corresponding 3D data file.

5. The user can then perform an interactive walkthrough on the resulting data file so as to refine the model. This iterative refinement process can be performed on the data file generated by the program in step 4. Additonal touch can also be done by various more sophisticated package such as *3D Studio* or *Maya*.

A typical result of applying the algorithm to a floorplan is shown in fig.3. In this figure, a *L*-block is first setted for the upper left lines. Subjected to this choice, the lower left block can be chosen as another *L* or a *Box*. But since the line on the lower right has a door on it, which called for an *Ann* there, thus a single *Box* is selected. For the upper right portion, here we found a line on which there should be an opening on it (the air conditioner 's position). thus a *donut* is chosen. As can be seen in this example, the setting of blocks' choices can be varied during the application of the algorithm and there is no unique solution to it. The best result can only be achieved through experiments and the understanding of the relative merits of the blocks.

2.2 Color-coded Polylines Approach

The block-based approach provides a fast method to recover the 3D representation from a floorplan image. The drawback is that it still requires a labor intensive stage of collecting coordinates of the lines in the plan. To further reduce the time needed, we devise a more automated approach. In this approach, the different signatures in a floorplan are coded as various unique colors. The algorithm is shown as below.

Color-coded Polylines algorithm: Convert a floorplan image into 3D representation

1. The floorplan image, which is assumed to be in gray scale, is properly thresholded into a binary image so that only the lines representing the construct are left. It is also better to remove any textual patterns reside.

2. Different structural signatures in the resulting image are colored according to their codes, for example, a line segment which has an opening is selected and the whole segment is colored into blue.

3. A program read the colored image, and based on the different color, "parse" the image into several binary versions of the image. Each of these images contains only the portion which belongs to that particular construct, for example a windowed one.

4. Another program reads these images in turn, and converts the binary image into a vectored one, i.e. a polyline description of the outlines of the colored region.

5. A final program is then used to generate the 3D description i.e. the 3D formatted file, from the polyline files generated in previous stage. The program will take care of different constructs and their corresponding descriptions in 3D space.

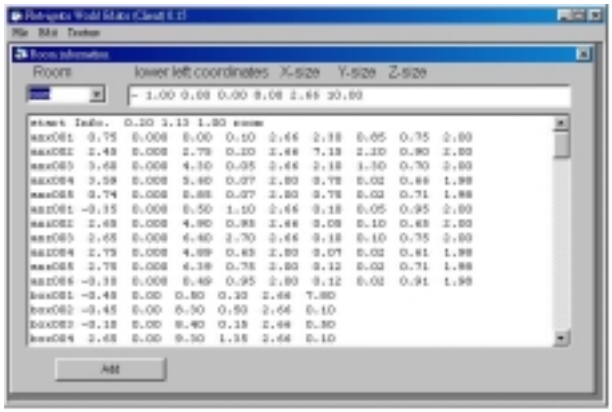

Figure 4: FlatEd – the program to build a block-based 3D file

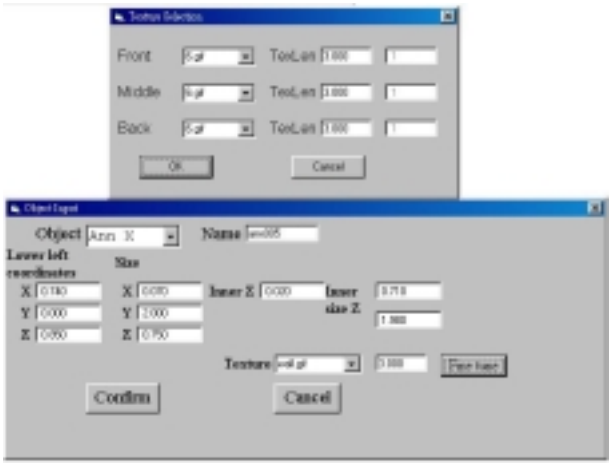

Figure 5: FlatEd allows the user to fine tune the generated models.

As one can readily observe, minimal human effort (stage 1) is involved in the above algorithm. The user only needed to i) properly control the thresholding process to minimize the unneeded polylines from noise in image, ii) Color the image according to the signature of different constructs.

The color-coded algorithm has various advantages over the block-based one. The most important one is that the user now no longer has to go through the tedious stage of measurement of vertex coordinates. This results in significant speedup of the coversion process. In addition, the usage of color code guaranteed more flexibility in defining one's floorplan requirement. Finally the color-coded algorithm does not require the user to be proficient in different floorplan constructs i.e. the breakdown process.

A little drawback of the color-coded algorithm is that the parameters of different constructs have to be fixed prior to processing, thus limiting the freedom of the user to modify the outlook of the generated 3D files. However, this problem can easily be solved by using more sophisticated modeling package such as *AutoCad* or *Maya*.

3 Implementation and Discussion

We implemented both approaches and tested the generated 3D files on *Crystal Space*, which is a free 3D engine written by Jorrit [6] and others. The engine allows interactive walkthrough of the generated file so that one can immediately inspect the model generated. In principle, given that the polygonal description of most 3D file formats are basically the same, it is relatively easy for one to modify the program to generate file in other 3D formats. We briefly describe below our implementation and discuss possible extensions.

3.1 Block-based Approach

The parameters of every block appeared in the floorplan is recorded and entered into a program (fig. 4) which we called "*FlatEd*". This program is written in *Visual Basic* due to its support in various data entry widgets. The program also allows the user to fine tune the blocks such as texturing parameters (fig.5). *FlatEd* interfaces with a COM server which is written in *Python* [5]. The *Python* server will generate the needed world file i.e. 3D representation in *Crystal Space* format. A screen shot of one of the generated scenes is shown in fig. 7.

3.2 Color-coded Polylines Approach

The output after thresholding and colorization is shown in fig. 8. In this figure, a black line represents an ordinary wall, green blocks correspond to bay windows, a yellow block denotes the position of light sources, whereas blue one is the windowed portion. The fact that light sources are explicitly represented is solely an advantage arised from the

Figure 6: A typical floorplan converted into a collection of blocks.

Figure 7: View of fig. 6 with texture and lighting.

use of color code.

During stage 2, we need to convert the binary image into a vectored format, i.e. the polylines. We use Ras2Vec (Raster to vector conversion) program which is written by Davide [4]. The polylines generated is shown in fig. 9.

The conversion of different polylines into a single 3D file is performed by a program in *Python*

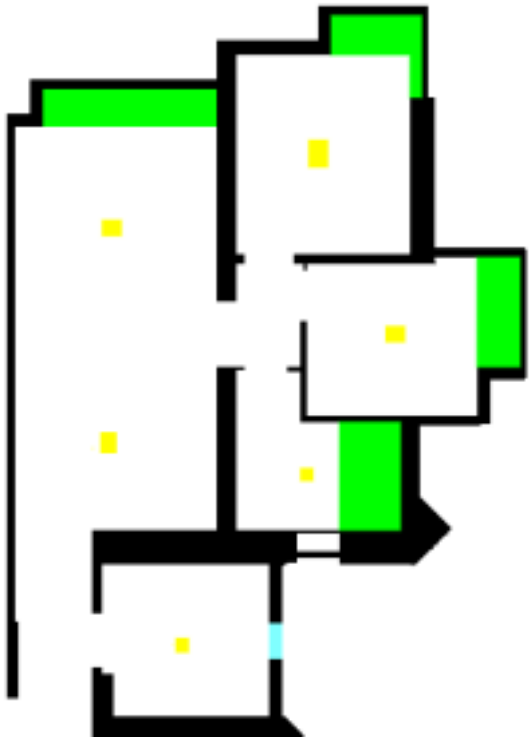

Figure 8: A floorplan in color coded format.

scripts. A point to note is that the polygons generated are in general required to be convex form, which is a basic requirement of most 3D rendering engines. However it is clear that sometimes the polygons generated from our approach are not necessarily convex. As noted in fig. 8, the bay window area on upper right is a concave one. We use the polygon decomposition algorithm by [3]. The algorithm will break down a complex, concave polygons into a number of convex polygons. We adopt the program written by Ofer [2] into a *Python* version which is integrated into our *Python COM* server mentioned in block-based approach. The resulting polygons are added to the 3D representation.

An application of the color-coded approach to floorplans representing larger area is shown in fig. 10. Fig. 10 is a rendered view of a development project in which several buildings as well as the roads and trees are placed using the different colors in the plan.

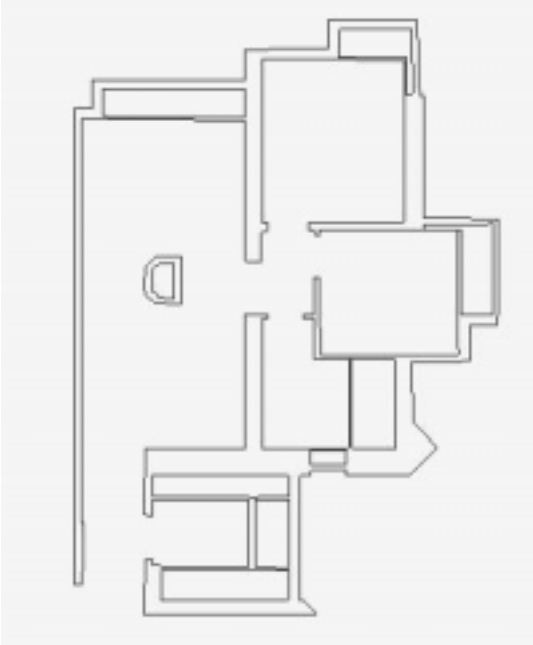

Figure 9: Floorplan converted to polyline by ras2Vec.

Figure 10: Another application of color-coded approach to floorplan.

3.3 Discussion

In this paper, we present two different approaches to generating 3D data from a floorplan image. Both approaches aimed at quickly generating a 3D model from a floorplan image, which is a first step in interactive 3D applications such as games and architectural walkthrough for ancient buildings. Usually for such applications, beside lowering the cost of model data generation, one would also want to modify as well as fine tune the results so that more eye-candies can be provided for final applications.

In our experiments, both approaches can generate 3D models which have high precision with respect to original floorplan. However from our experiences, the two approaches each have their own relative merits and drawbacks as well.

The block-based approach has the drawback of tedious labor in vertices information preparation. In addition, it requires the user to have quite in-depth knowledge on different block types so as to build a model of seamless joints between blocks. On the other hand, block based approach allows its user to perform more fine-tuning operations to the resulting models such as adding furnitures, or even adding special features on a particular polygon(fig. 11). An even more attractive feature is that it allow the user to easily dive into the generated data file to fine tune the model for special effect such as performance tuning by removing unused polygons. A typical floorplan as in fig. 8, which results in 637 polygons requires approximately 2 hours to generate the 3D model file.

For the color-coded approach, relatively little knowledge is required for a user to generate a 3D model. This opens up more avenues for this approach. One interesting application is an integrated environment where the user just has to draw with different colored pens and the program will build a model for interactive tour. The application should be well suited as a creativity tool for children, etc. In addition, the time used to build a 3D model is significantly reduced, for example, the 3D model is generated in half an hour, including the image processing time and color coding.

The drawbacks of the color-coded approach are that lesser fine-tune operations as well as flexibility in modeling is allowed. In addition, the generated file is difficult to edit. However we noted that the user can still edit the resulting data file in more sophisticated modeling packages.

In view of the goal of minimizing the time in construction of model from a floorplan, we con-

Figure 11: Block based approach allows easy modifications as well as analytical placement of other items.

cluded that we are successful in significantly reducing the time needed. As a simple benchmark of the performance of our algorithm, we noted that the floorplan presented in fig. 1 originally need a time of approximately two to three working days to build the resulting model using plain estimation and hand editing of data file, in contrast with the time reported above. One may argue that using sophisticated modeling package, one can significantly reduce the modeling time. But as the modeling package nowaday are written with generic purposes in mind, it would need quite extensive work in order to build the same model. Our algorithm takes advantages of the orthogonal feature of floorplan, and the redundant constructs in man-made building. The resulting algorithm can thus save more times in comparison to more general modeling approaches.

4 Conclusion

Two efficient approaches to construct a 3D model from a floorplan image are discussed. The first approach operates by fitting a set of building blocks to the floorplan image. The second approach used different colors to represent various constructs inside a floorplan image and then convert the raster image into a number of polylines. Each polyline is then used to generate a set of polygons which represent that particular block.

Figure 12: Some screen shots of the buildings created.

Relative merits and drawbacks of each algorithm are discussed and various results are presented.

5 Acknowledgement

The author would like to thank Francis Lee for suggesting the idea of blocks based approach and valuable helps throughout the project.

References

[1] Historical floorplan emporium. *http://www.b-ware.com/hive/fplans/index.htm.*

[2] Ofer Belinsky. Minimal convex polygon decomposition demo. *http://www.math.tau.ac.il/ sariel/TA/wcg98b-/convex_decomp/Default.html.*

[3] J.M. Keil. Decomposing a polygon into simpler components. *SIAM JC*, 14:799–817, 1985.

[4] Davide Libenzi. Raster to vector conversion program. *http://www.maticad.it/davide/.*

[5] Andy Robinson. Embedding python in visual basic / delphi / powerbuilder apps. *http://www.robanal.demon.co.uk/demos/pyvb/index.html.*

[6] Jorrit Tyberghein. Crystal space: a free 3d engine. *http://crystal.linuxgames.com/.*