Generating Three-Dimensional Building Models from Two-Dimensional Architectural Plans

Rick Lewis

Master's Project
under the direction of Carlo Séquin

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley

May 13, 1996

Abstract

There are many useful applications of three dimensional polyhedral building models, such as visualization, light, sound, energy, or fire simulations, and cost analysis. Unfortunately, construction of polyhedral building models using currently available modeling software is a tedious process that involves considerable human effort, and often results in topologically incorrect or incomplete models. This paper describes a system called the Building Model Generator (BMG), that creates correct 3D polyhedral building models from architectural floor plans, while requiring minimal user interaction.

Two-dimensional architectural floor plans are a standard format for expressing the detailed architectural design of a building. These plans are abstracted orthographic projections of the structure of each floor onto two dimensions, with symbolic annotations indicating special entities such as doors and windows that are not visible from the top-down orthographic view. These plans are easier to create than the corresponding 3D model of the building because they involve drawing in two dimensions rather than three, and the sheer number of geometric entities drawn is much lower. While these plans may contain small inconsistencies, most can be solved algorithmically in two dimensions.

With the BMG system, input floor plans from AutoCAD are passed through filters that perform topological correction and semantic analysis. Disjoint and overlapping entities are corrected, symbols indicating doors and windows are located, and edges are grouped into contours representing each labeled space on the plan. The result is then extruded to form the correct polyhedral model of the floor. A method is provided for stacking and assembling individual floors, and incorporating other structural elements such as staircases, into a composite building model with correct topology. Results are compatible with the Berkeley WALKTHRU system for interactive visualization of polyhedral environments, and the NIST CFAST fire simulator for buildings. The system was used to generate a new model of the new Soda Hall computer science building at Berkeley, using floor plans drawn by architects.

Contents

List of Figures

1 Introduction

The field of architecture has integrated computers mostly in a limited capacity as drafting tools. Architects use basically two-dimensional CAD tools such as AutoCAD [2] to draw plan views of individual floors or selected elevations. Typically, computerized three-dimensional models are not part of the iterative design process. It is easy to understand why; three dimensional modeling tools are not prepared to take specific advantage of the typical properties and characteristics of three-dimensional buildings, and thus the architect is required to synthesize the three-dimensional model manually, either through drawing or through assembly of building components (which were drawn manually). Both approaches are too tedious and error-prone for architects to embrace routinely during the early phases of the design process.

Systems have now been developed that allow users to view three dimensional models of buildings [1, 11] in a simulated walkthrough environment where the user can navigate through the building and observe a reasonably detailed and accurate approximation of what an occupant of the actual physical building would see. The advantage of the three-dimensional viewing capability is clear: users can observe the building from the occupant's perspective long before ground is broken, and while there is still a chance to make substantial changes at reasonable costs.

Despite its usefulness, most architects have not integrated the use of interactive walkthroughs into the iterative, formative stages of the building design process. It is laborious and difficult to synthesize a topologically correct three-dimensional model of a building with the help of today's CAD tools. The daunting task of constructing a three-dimensional building model requires that all facets of walls, doors, windows, and interior objects be drawn correctly in three-dimensional space on a two-dimensional screen, typically with a two-dimensional pointing device. Also, since three-dimensional walkthrough environments may incorporate radiosity-based illumination or use sophisticated visibility computations to minimize rendering time and increase frame rate, it is important that the three-dimensional model be free of inconsistencies such as non-planar, overlapping, intersecting, improperly oriented, or non-closed polygons. Finally, after a building model is developed, subsequent changes at a conceptual level are expensive; for example, if the interior height of the walls on a given floor is modified, every wall polygon must be redrawn in order to reflect the change, and the coordinates of the ceiling polygons must be adjusted as well. Construction of the polyhedral model of Soda Hall at Berkeley ended up consuming about two person-years of effort. Clearly, the amount of human resources required to design consistent three-dimensional models must be reduced by orders of magnitude before use of such models will become an integral part of architectural design.

There are other important uses for a fast 3-D building model generator: the resulting models can be used for simulations such as energy, lighting, acoustics, and fire. When changes to a model are required in response to the unsatisfactory outcome of an analysis or simulation, it is advantageous that the model can be regenerated quickly after simple two dimensional changes to the floor plan(s), or the adjustment of global parameters such as ceiling height. We have also found that automatic generation of the model meshes nicely with adapting the building model to new visualization systems; since room boundaries and portals are known explicitly at the time of extrusion, this information can be put directly into a format suitable to the visualization system, eliminating the need for a three-dimensional visibility preprocessing system to locate spaces and portals heuristically.

In this research we are interested in bridging the gap between two and three dimensions in the building-model generation process. Specifically, our goal was to develop a system that would accomplish the following: given existing floor plans in some computer-drawn form (such as AutoCAD), automatically construct the corresponding three-dimensional polyhedral model of the floor, and assemble a consistent model of the entire building that is appropriate for viewing with the Berkeley WALKTHRU system. Several distinct components comprise the solution to this problem: a conversion tool that converts floor plans drawn in AutoCAD into a suitable format for our purposes, a floor plan correction system that fixes topological errors, a floor plan analyzer that determines the semantic meaning of the geometry on the floor plan, an extrusion system that automatically constructs the three-dimensional representation of the floor based on the floor plan and some default parameters, a 3-D staircase generation tool, and a composition system that combines separate floors and staircases of the building into a single, correct 3D model.

2 Related Work

Several previous and ongoing projects at Berkeley, as well as a few commercially available software packages, preceded and motivated the development of the Building Model Generator.

2.1 Berkeley WALKTHRU

The "Walkthru" group at UC Berkeley developed the Berkeley Building Walkthrough (WALKTHRU) system for interactive visualization of densely occluded polyhedral environments. Interactive frame rates are achieved through visibility precomputation [19] and level-of-detail management [10]. The building model used to develop and demonstrate the system was produced by iteratively applying "clean-up" algorithms and user interaction to convert a grossly inadequate 3D AutoCAD wire-frame model of Soda Hall (then in the design stages) into a complete polyhedral model with correct face intersections and orientations [12]. The Berkeley UniGrafix [7] format was used to describe the geometry of the building, because of its compatibility with the modeling and rendering tools available within the group. The interior of the building, including furniture and light fixtures, was modeled by hand, through instancing of 3D models of those objects. In all, the creation of the detailed Soda Hall model required two person-years of effort. It became clear that better modeling systems were needed.

2.2 Architectural CAD Packages

Over the past few years, several building design programs have become available commercially. These packages generally fall into two broad categories: simple extrusion-based packages, and packages that provide pre-modeled building components that the user assembles to form a 3D building model. Both classes have deficiencies that created the need for the work described in this paper.

The commercially available extrusion-based packages, such as *Home Design 3D* [8] and *3D Virtual Reality Room Planner* [6], require the user to draw a simplistic floor layout schematic within a proprietary editing environment, which is then extruded into a 3D model. There is no capability to extrude directly from a pre-drawn, detailed, possibly inconsistent architectural plan. Capacity is small (tens of rooms) when reasonable performance is required, so large buildings are out of the question. Walls may be infinitely thin (represented by two oppositely-oriented, coincident polygons). In general, this type of system is targeted at home computer users who wish to experiment with interior design. Since the symbolic floor plan editors are limited in flexibility, rarely does the symbolic plan correspond precisely to the real architectural floor plans for the building.

More sophisticated architectural design tools, such as ArchiCAD, provide detailed building components that the user pieces together in 3D space to form a building model. While less laborious than drawing the entire 3D model in a drafting tool, the problems that arise when the user is required to position objects in 3D still apply. Since the user manipulates a 2D pointing device (i.e. a mouse) and receives visual feedback from a 2D computer screen, it is difficult to properly position entities in 3D space and end up with a consistent, solid building model.

Both classes of systems described above fail to meet our needs in two major areas: first, significant human effort is required. Second, there is no guarantee that resulting models are consistent and error-free, as required by the sophisticated applications we are interested in, such as interactive walkthrough and fire simulation. Some applications may also require richer semantics (such as room adjacency information) that these systems do not provide.

In addition, these tools ignore the fact that the majority of computer-drawn architectural floor plans are created the AutoCAD program. Although these plans are usually inconsistent and incomplete, some sophisticated correction and analysis can be performed by computer, resulting in clean 2D floor plans that almost completely specify the 3D geometry of the building. This information is wasted when the user is required to redraw the floor plan in a different format, or worse yet, required to draw the 3D geometry of the entire building.

2.3 WALKEDIT 3D Editing Tool

The WALKTHRU Editor (WALKEDIT) [4] was developed in order to facilitate the manipulation of objects within the virtual environment of the building. It employs an new object manipulation technique known as "Object Associations", which uses knowledge about the behavior of objects to reduce object manipulation within 3D virtual environments to a two-dimensional problem. For example, a picture frame can be selected and moved around; as it moves, it remains flat against the wall, therefore reducing movement to two degrees of freedom. This technique drastically reduces the amount of time required to properly position interior objects within the building model. Standard editing functions such as Copy, Paste, and Delete are also included.

These features combine to form a powerful tool for quickly modeling the interior of the polyhedral building model. A natural complement to this tool is a system that similarly facilitates the construction of the building model itself. Reducing user interaction to a two-dimensional level, rather than a 3D environment, shifts much of the burden of 3D modeling to the computer, freeing the user to be more creative and more productive.

2.4 Staircase Generator Tool

The staircase generator tool "StairMaster" [13] is an early demonstration of the paradigm we wished to follow with the Building Model Generator: the user specifies the layout of the model in two-dimensions using a floor plan, and the computer automatically generates the corresponding 3D model using the floor plan and some additional parameters (such as height). In the case of StairMaster, adjacent, coplanar polygons are used to specify the floor plan of a staircase. The StairMaster program reads this plan and computes proper locations for railings and support beams. The user specifies the lowest step and the direction of ascent. StairMaster then automatically creates a polyhedral model of the staircase and displays it in a viewing window. Real-time, interactive modifications of the staircase can be made by moving sliders on a user interface form that specify step height, railing height, and other parameters. The ease of creating a fairly detailed staircase model, by just drawing polygons in 2D and then changing a few slider values, demonstrated the validity of the paradigm. The staircase generator tool has been adapted for use with the Building Model Generator system described in this paper, and is described in a later section.

2.5 Symbolic Floor Plan Extruder Tool

The symbolic floor plan extruder tool [14] tool is a further demonstration of the concept of generating 3D geometry automatically from 2D plans. This tool uses very simple symbolic plan-view schematics drawn with any 2D editor. In this symbolic representation, rooms, corridors, and other spaces are all polygons in two dimensions. Figure 1 shows the symbolic schematic for the fifth floor of Soda Hall.

The tool analyzes adjacencies between rooms, automatically places doors and windows at reasonable locations on the schematic according to some simple rules, and constructs the three-dimensional geometry for the building, that corresponds to the two-dimensional plans. The resulting three-dimensional building representation is free of errors and inconsistencies because the impact of human error is minimized; that is, the geometry is generated algorithmically instead of drawn by a human. Resulting models are also viewable with the Berkeley WALKTHRU system.

With this tool, users can use a two-dimensional editor to quickly sketch out polygons representing the floor plan, and within minutes walk through the building that would result. This is

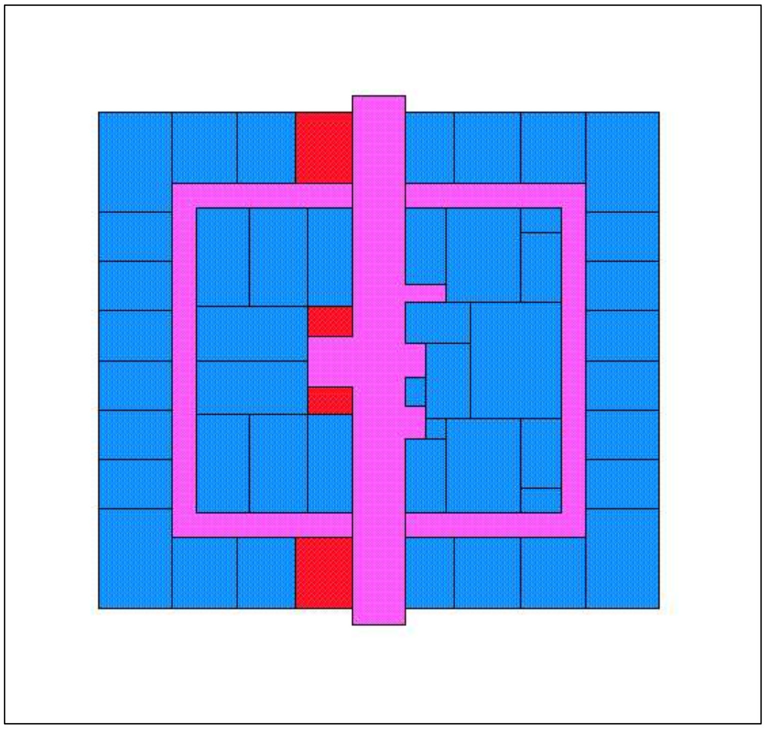


Figure 1: Symbolic plan of Soda Hall, floor 5.

potentially a very powerful addition to the set of computer-based design tools that architects use, since it allows building visualization, with reasonable accuracy, before final and formal floor plans are drafted using a program like AutoCAD, and without having to do any drawing in 3D. Real, detailed floor plans take many hours to draw correctly with a program such as AutoCAD; walls must be drawn with correctly offset inner and outer contours, standard symbols for doors and windows must be correctly drawn and properly placed into gaps in the wall contours, etc. The ability to use simpler, symbolic plans that are easy to sketch makes it possible to include building visualization in the early design stages when the architect is merely thinking conceptually about the form and function of the building.

We developed some tools specifically for fast, straightforward design of these symbolic floor plans. Also, we developed a conversion process that allows symbolic floor plans drawn in widely available drawing programs, such as Canvas and Form-Z, to be used with the symbolic extruder tool. For example, symbolic schematic floor plans for Soda hall were drawn using the Canvas drawing program for Macintosh, the plans were converted to the Berkeley UniGrafix format, and then used with the symbolic extruder tool to model a simplified version of Soda Hall. One floor from this model is shown below (Figure 2).

While the use of symbolic plans simplifies the problem, it also constrains the user to a lower level of detail and less realism in the final building model, compared to what would be possible using detailed architectural plans.

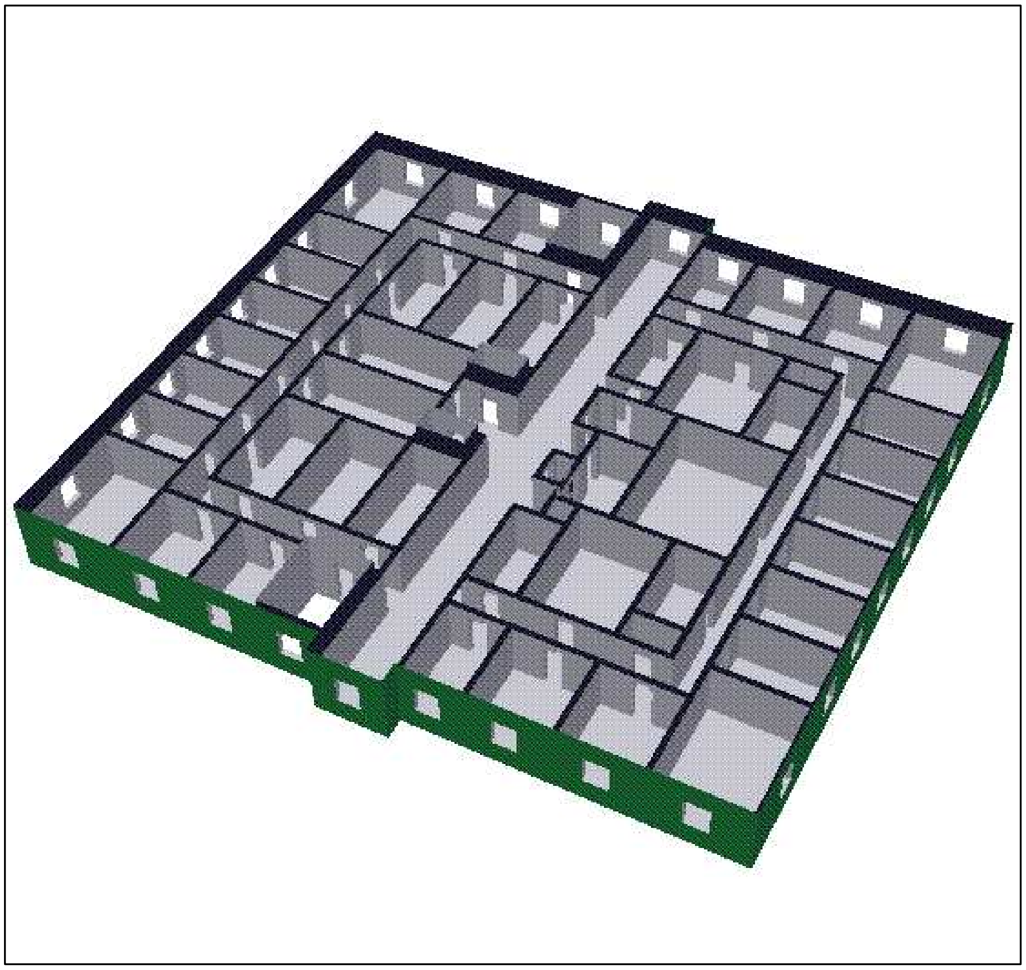


Figure 2: Approximated model of Soda Hall 5th floor, extruded from symbolic floor plan.

3 Overview of the Building Model Generator

The natural next step beyond tools that analyze and extrude symbolic floor plans was to create a system that could use real architectural plans (which are widely used and available) as input and produce polyhedral building models that conform precisely to those plans. This extension brings with it a great amount of additional complexity, because the input floor plans are drawn independently of the extrusion tool. The CAD tools that are typically used to drawn these plans, such as AutoCAD, allow the user total freedom in what they draw; therefore, the resulting plans are usually full of errors, inconsistencies, and ambiguities. There are problems both at the geometric and semantic levels. The Building Model Generator (BMG) system described in this paper corrects many of these problems, and attaches appropriate semantic meaning to the geometry. BMG then analyzes the floor plan and finds closed contours for each of the spaces. A sophisticated extrusion system then creates a complete 3D solid model of the floor. Multiple floors can be stacked into a full building model.

The BMG system is a pipeline of correction, analysis, extrusion, viewing, and editing tools, which are combined into a single integrated program. Figure 3 shows the stages in the process, and the following subsections give an overview of the main actions performed in each step.

3.1 Floor Plan Conversion from DXF to UNIGRAFIX

Floor plans used as input to the BMG system are described in the widely-used DXF (Drawing eXchange Format) [3] geometry description format. Each floor plan may contain many different types of entities, including lines, polylines, polygons, and text. The *conversion* phase of the BMG system converts the useful geometric entities concentrated on a few select layers in the DXF file into UniGrafix format. The subsequent BMG subsystem components operate on UniGrafix-format geometry.

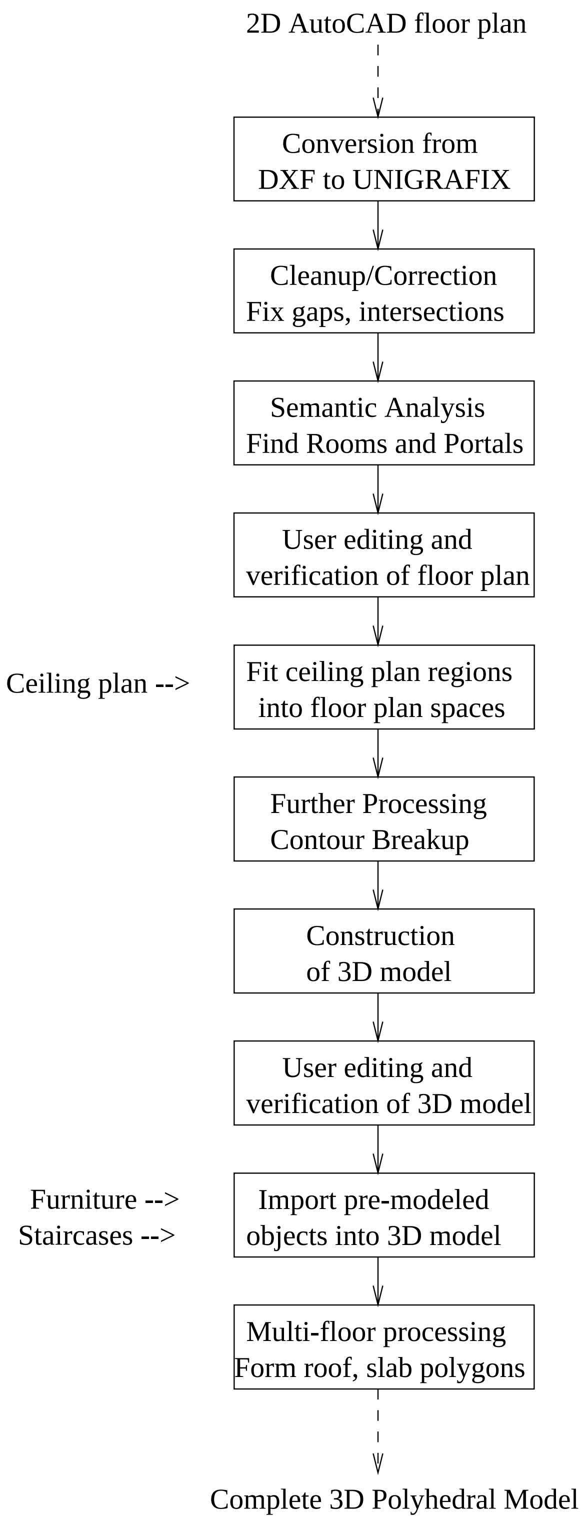
3.2 Floor Plan Correction

The input floor plans typically have errors and inconsistencies that preclude directly extruding the two-dimensional entities into a three-dimensional model. At the geometric level, the most common errors manifest themselves in the form of connectivity problems: entities meant to be adjacent may overlap or may be separated by a gap. Such problems make it difficult to create the desired semantic associations between entities. Also, these problems lead to holes, coincident polygons, and incorrect visibility and radiosity computations in the resulting 3D model. The *correction* subsystem eliminates most of these errors automatically.

3.3 Semantic Analysis of Floor Plan

Semantic analysis of the floor plans, performed by the *analysis* subsystem of BMG, is required for two reasons. First, it is necessary to locate symbolic entities in the floor plan and generate the corresponding literal equivalent for each; for example, each door symbol consists of a line or rectangle (representing the door), an arc (representing the swing of the door) and two line segments (representing the door jamb on each side of the door opening). For extrusion purposes, BMG must create the two edges that close each of the space contours on either side of the door opening.

Second, the geometric entities must be grouped together into entities with a specific semantic meaning. For our purposes, the edges representing walls should be grouped into a single contour specifying the space or room boundary. This is necessary in order to provide the user with the ability to control the properties of the extruded model on a per-room basis. Also, single contours representing each enclosed space enable straightforward determination of proper polygon orientation in the extruded model, and provide the compartmentalization needed for efficient visibility analysis and fire simulation.



2D AutoCAD floor plan
Conversion from DXF to UNIGRAFIX
Cleanup/Correction Fix gaps, intersections
Semantic Analysis Find Rooms and Portals
User editing and verification of floor plan
Ceiling plan -->
Fit ceiling plan regions into floor plan spaces
Further Processing Contour Breakup
Construction of 3D model
User editing and verification of 3D model
Furniture -->
Staircases -->
Import pre-modeled objects into 3D model
Multi-floor processing Form roof, slab polygons
Complete 3D Polyhedral Model

Figure 3: The BMG system.

3.4 User Editing of Floor Plan and Semantic Information

Contours corresponding to the rooms and portals on the floor plan are formed automatically; however, it is possible that the intended results are not obtained through this process. In these cases, user interaction is required. The user may define new edges in order to form closed contours for rooms and portals that were not automatically found during the analysis phase. After adding new geometry, the user may have BMG attempt to automatically form the additional contours; optionally, the user may specify the entire boundary of an additional contour by hand. Also, undesired contours may be deleted. This editing phase provides a way for the user to verify the results of the correction and analysis phases, before continuing on to create the three dimensional floor model.

3.5 Reflected Ceiling Plan Correction

A reflected ceiling plan (RCP) may optionally be used to partition rooms that will contain several regions of different ceiling heights. Each such region is indicated by a diagonal specifying the region's typically rectangular boundaries. These descriptions contain similar numerical inaccuracies and connectivity problems (with respect to the room contours that contain them) as all other geometric entities; therefore, the RCP correction phase intelligently snaps the boundaries of special ceiling areas to nearby corners in the room. Regions very close to one another are expanded or shrunk to create the most plausible adjacency. Rooms containing one or more RCP regions are partitioned according to the region boundaries, resulting in spaces that will have no physical walls between them, but whose ceiling heights will be separately addressable.

3.6 Classifying Doors and Windows

The user may assign a user-defined type to each portal located by BMG. With each portal type, the user can associate a prototype model that will be inserted into the corresponding portal opening in the 3D model. BMG automatically sizes the height of the portal opening in the 3D model to fit the inserted prototype. This allows the user to interactively change and observe the size and/or style of window and door frames easily, by simply changing the inserted portal prototype.

3.7 Pre-extrusion Processing of Space Contours

Several additional processing steps are required in the pre-extrusion stage in order to support the generation and editing of the 3D model. First, concave regions, such as L-shaped rooms and long, winding corridors, are broken up into collections of adjacent, convex regions. These convex regions provide a straightforward path to generating cells to be used in visibility computations for interactive walkthrough of the 3D model.

The edges between such adjacent regions are marked specially and are treated differently than regular wall edges during extrusion. After extrusion, the user may select each space and set its ceiling height. Spaces that were created by breaking up a single concave space into multiple convex regions should not have walls between them; however, if two such adjacent spaces have different ceiling heights, there should be a piece of wall generated that closes off the opening created by the two adjacent spaces of different heights. The pre-extrusion processing phase enables this by locating and marking the edges of adjacency, and giving each of the spaces that share an edge a pointer to the other space, thus recording the pairs of spaces that have this type of adjacency.

3.8 Generation of Three Dimensional Floor Model

In general, the 3D polyhedral model of each floor is created by traversing the 2D contour for each region, and generating a vertical, inward-facing rectangular polygon for each edge. Floors and ceilings are respectively upward and downward-facing polygons, at appropriate heights, whose 2D projection is identical to the corresponding 2D space contour. Where each space is adjacent to a portal, a suitable hole is formed in the extruded walls. Properties such as wall height, portal opening size, and portal elevation are specified by the user prior to extrusion, and can also be changed later.

3.9 User Editing of Three Dimensional Model

After the 3D model for the floor has been created, the user is afforded a number of ways to edit the result. Spaces can be edited at three levels of granularity: the entire collection of spaces on the floor, all spaces of a given type (office, corridor, etc.), or any individual space. Space properties that can be edited include: name, type, wall height, and colors. Portals can also be edited at three levels of granularity: the entire collection of portals on the floor, all portals of a given type, or any individual portal. The user may change the height (Z value) of the bottom and top of the portal opening, and the portal type. If the type of a portal is changed and there is a prototype model associated with the newly chosen type, the opening is sized automatically in Z to fit the prototype. Prototypes that do not fit the portal opening in X or Y are not allowed. When changes are applied, the BMG system updates the model as needed and shows the results immediately in the model view window.

3.10 Incorporation of Detail into Three Dimensional Model

The user may wish to add specialized models of particular components such as staircases, elevator cars, and non-rectilinear structures to the 3D model of the floor. To this end, the *component insertion* feature of BMG allows the user to specify the location and orientation with which such component models may be inserted into the building. Each object to be incorporated must be described in the UniGrafix language. To include such an externally designed component in the 3D model created by BMG, the user may specify an insertion point (a point corresponding to the origin of the inserted component's coordinate system) by clicking on a vertex in the 3D model. Then the user must specify the rotation by clicking on a second vertex. With this information, BMG determines the correct translation and rotation, and inserts an instance of the component into the 3D floor model.

3.11 Combination of Multiple Floor Models into a Single Building

The BMG user can form a complete building model by stacking individual floor models atop one another. However, holes may exist in the exterior of the resulting building model because the exterior walls of adjacent floors may not have an identical 2D projection. For example, if the first floor is significantly larger than the second, stacking them up results in a hole where the second floor does not entirely cover the first. BMG analyzes the 2D projections of adjacent floors and automatically creates suitable floor or ceiling polygons to close off these openings, in order to preserve the solid model characteristics of the building.

4 Description of Input Floor Plans

Floor plans used as input to BMG (Figure 4) are assumed to have some minimal semantic information attached to them, in the form of *layers*. Using layers is the standard method of classifying and grouping geometry in AutoCAD, and in DXF format in general. The geometry in the Soda hall floor plans is grouped into specific layers for walls, doors, windows, and other more specialized elements.

In addition, numbered rooms and spaces in the building are indicated by room number labels on the floor plan, placed near the center of the boundary of each space. This label proves useful in locating the closed contours for these spaces.

While this modest amount of semantic information can be assumed, correct geometry can not. In fact, input floor plans typically suffer from many errors and ambiguities. At the geometric level, line segments representing contiguous walls in a room may not meet at their endpoints, which would result in a hole in the wall if extruded as-is. The ramifications of such errors are typically not considered when the floor plan is drawn: edges meant to form a closed contour overlap or are disjoint, door and window symbols do not fit properly in their openings, and there are both superfluous and missing geometric entities.

Geometry that is intended to have the same semantic meaning (wall edges, for example) is often drawn using different primitives, resulting in entities of different types. For example, a room with four walls may be drawn as a collection of single line *segments*, or as a *polyline*, or as a *polygon* (closed face).

There is also a lack of sufficient semantic grouping of geometric elements. For extrusion and editing purposes, it is useful for all of the line segments that comprise the boundary of a particular space (room, corridor, etc.) to be grouped together into a single closed contour;

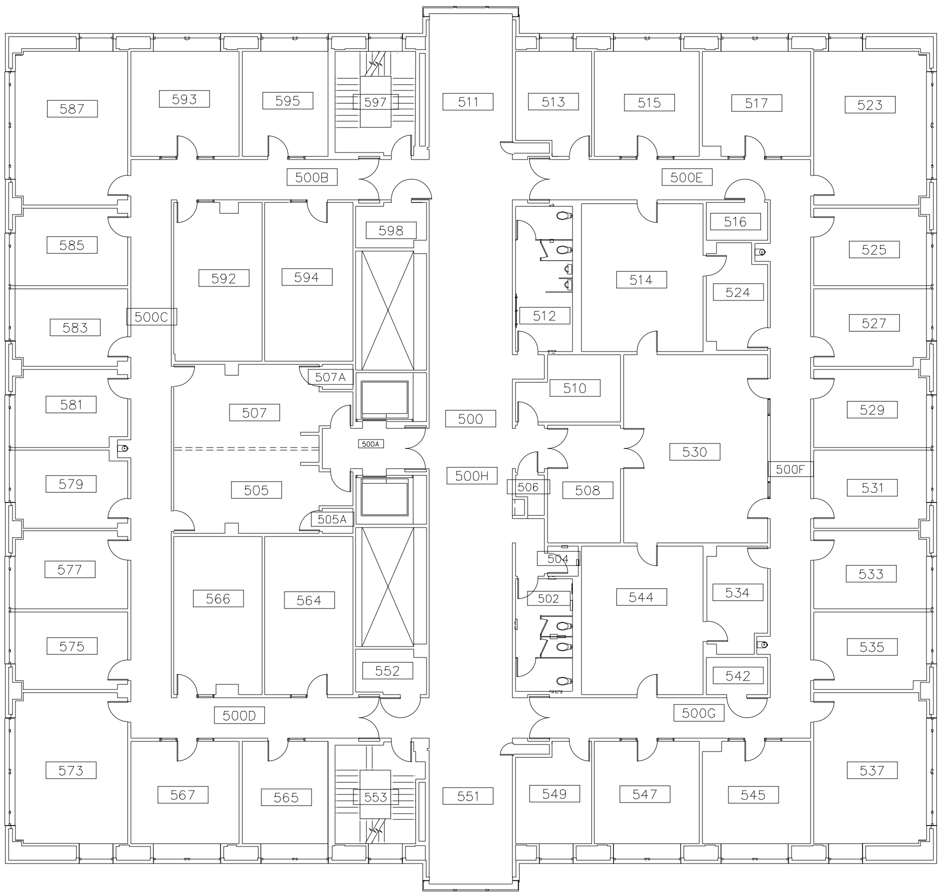


Figure 4: Floor plan for Soda Hall, floor 5.

however, no such grouping typically exists in the input floor plans. A single room is just a set of (possibly overlapping and/or disjoint) segments and polylines, which in turn may extend to represent wall segments of adjacent rooms.

Given the typical condition of most input floor plans, some correction and analysis is required in order to eliminate superfluous entities, ensure correct topology, and obtain meaningful groupings of simple geometric entities.

5 Conversion of Floor Plans from AutoCAD to UniGrafix

The floor plans for Soda Hall were created by Anshen and Allen, Architects, in San Fransisco, using the AutoCAD program as a drafting aide. The persons creating these plans did not know the plans would be used some time later for extrusion of a 3D model.

AutoCAD saves drawings using a proprietary binary file format, which is not suitable for our purposes. We chose to convert the floor plans to the Berkeley UniGrafix format because it is text-based, simple, and compatible with a large library of graphics software used by our group. The process of converting from AutoCAD to UniGrafix involves exporting the floor plan from AutoCAD into a DXF file, and then converting the DXF-format file to UniGrafix format. We chose to export the AutoCAD drawings in DXF format over other available formats because our library of tools already included a program that converts DXF-format files into UniGrafix [12].

5.1 Layer Selection Within the AutoCAD Program

Typical floor plans contain more information than is required for our purposes, such as wiring and plumbing, special interior objects not related to extrusion such as sinks and water fountains, and visual aids for the architects, such as broad areas of hatch pattern that indicate space usage. From within AutoCAD, we determined each layer's purpose by turning layers on and off and observing the resulting changes to the display. The layers relevant to 3D model generation were noted for future use. Examples of relevant layers include: wall geometry layers, door geometry layers, window geometry layers, and layers containing room labels identifying each space.

The entire drawing is then exported in DXF format for use in the next step of the conversion process.

5.2 The *acad2ug* DXF-format to UniGrafix-format Converter

The *acad2ug* [12] program accepts DXF-format files as input and converts desired layers into the corresponding UniGrafix-format geometry file. Layer names in DXF translate to color names in UniGrafix. The most common geometric entities translate as follows:

DXF	UNIGRAFIX	Example use
LINE	Wire	Wall, window sill, door jamb
POLYLINE	Wire	Wall
3DFACE	Face	Room label, door
VERTEX	Vertex	N/A
TEXT	Wires	Room number

Any relative coordinates in the DXF file are translated into absolute, world coordinates in the UniGrafix file. Since not all of the layers in the DXF file are needed, the user creates a file that specifies the layer names of all entities for which conversion is desired. Entities not in any of the specified layers are ignored. The following layers in the Soda Hall plans have been converted for our purposes:

Layer	Meaning
`AN-WL-IN`	Interior walls (inside the building)
`AN-WL-EX`	Exterior walls (outside the building)
`AN-DR-DR`	Door
`AN-DR-JB`	Door jamb
`AN-WN-SL`	Window sill
`AN-WN-JB`	Window jamb
`ANRTX-NT`	Reflected ceiling plan diagonals
`AN-VP-ST`	Stairs
`DN-RM-NO`	Room label geometry
`DN-RM-NM`	Room identities associated with room labels

(Reflected ceiling plans will be discussed in section 7.4, and stairs will be discussed in section 10.)

The resulting UniGrafix file must be further processed before correction and analysis. Special filters are used to eliminate illegal entity names (for example, the hyphens in the layer names are not allowed in UniGrafix color names, and are replaced with underscores), to convert faces into wires, and to subdivide wires into single-segment lines (UniGrafix wires with only one edge). This conversion and subdivision is required because faces and multiple-edge wires imply a semantic association between edges that was not necessarily intended by the creator of the original AutoCAD file. For example, the contour formed by all of the walls in a given office may have been drawn as a single 3DFACE (UniGrafix face), or as a single POLYLINE (UniGrafix wire), or as a combination of several LINE (UniGrafix wire with one segment) and POLYLINE entities. Since the visual result in the typical AutoCAD wire-frame display of any of these combinations is identical, no special association should be assumed between the segments of a single entity; that is, the final contour that is formed from a collection of wires should be no different from the contour formed by a face or a single closed wire. So all wall geometry is decomposed into the most atomic element, the single edge. The resulting homogeneous collection of edges is used later by a contour-forming tool that walks along connected edges and forms a closed contour for each space.

5.3 Specification of Semantic Meaning of Individual Layers

As the last step in preparing a floor plan for the correction phase, the semantic meaning of the individual layers must be specified. The correction and semantic analysis tools rely upon a knowledge of the semantic meaning of each layer of geometry in the file. Each layer is grouped as such because the entities in each layer are of a type that conforms to a certain behavior. For example, wall edges (in the layers AN-WL-IN or AN-WL-EX) should be used to form room contours, whereas edges in the room label layer (DN-RM-NO) are symbolic and exist only so that the identity of the surrounding room is known. A file is created by the user that specifies the semantic meaning of each layer:

```
wall-geometry:  AN-WL-IN
                AN-WL-EX
door:           AN-DR-DR
door-jamb:      AN-DR-JB
window-jamb:    AN-WN-JB
window:         AN-WN-GL
room-label:     DN-RM-NO
```

This file is referred to by the subsequent correction and analysis tools. In its current form, BMG requires exactly these six separate semantic groupings. Since we want BMG to be extensible for use beyond the Soda Hall plans, it is important to note that not all floor plans will have exactly the same layer names and groupings of geometry. Of course, the brute-force option of properly labeling and grouping floor plan geometry from within the drafting program would solve this problem, but this option may be undesirably labor-intensive, especially for large buildings. Should it be desired to use floor plans that do not contain all of these six groupings, additional floor plan processing would be required. It is reasonable to require absolutely that wall segments be grouped into a layer, and this is typically done for all computer-drawn floor plans [17]. To sort the remaining geometry into the five other semantic groups (*door, door-jamb, window, window-jamb, room-label*), a preprocessing system would be implemented that intelligently examines each geometric entity and attempts to classify it into one of the groups. For example, *door-jamb* geometry could be found by searching for matching pairs of wall-width edges that are door-width apart. Therefore, the additional generality to accommodate floor plan sets from other projects, done in a different style that does not exactly conform to the above six semantic groupings, can be expected to require a small, project-specific preprocessing system.

6 Topological Correction of Floor Plans

Usually, a very small difference between the endpoints of two edges in a contour is an error; typically, what was intended by the draftsperson was two edges that intersect at their endpoints. This error occurs frequently when room contours are drawn as a collection of single line segments (each of which can be independently positioned with the potential for error) rather than as a single closed polygon wherein two consecutive edges are guaranteed to share the same vertex. This type of error manifests itself in two ways: gaps between edges, and overlapping edges. Such errors need to be corrected in order to permit the formation of closed contours that consist of contiguous edges. Closed contours are required for extrusion purposes, since extruding a contour that is not closed will result in incorrect connectivity between polygons in the resulting 3D model. Therefore, several algorithms are applied that correct gaps and improper intersections.

6.1 Coercing Vertices to a Grid

The first and most efficient algorithm used to fix gaps and improper intersections snaps each of the vertices to a grid of specific resolution. We could merge all vertices within a maximum distance of one another, but a simple vertex merge is inappropriate for this purpose because such an algorithm might choose a bad position for the final vertex. This can disrupt the axial alignment of the walls (Figure 5b).

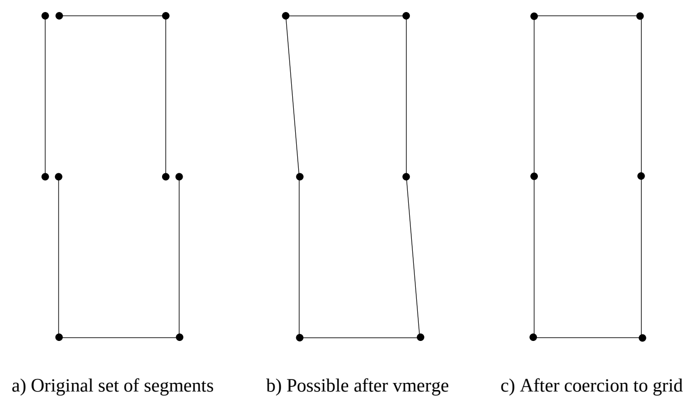


Figure 5: Advantage of coercing to grid vs. vertex merging.

Suppose the innermost walls of a row of interior office spaces are all to be placed 20 feet from the outside of the building. If some vertices are in error and stray from this axis, we would certainly want those erroneous vertices to be moved when correction takes place, instead of moving the correct vertex from the other edge endpoint to the incorrect location, as the simple vertex merge might do.

The solution to this problem is to decide on a uniform grid with a specific spacing between grid points. It is likely that the original drawing was also drawn on an integer grid. The new grid is intelligently placed such that most vertices are already on the grid. Each vertex is snapped to the closest grid point.

BMG produces a good value for grid spacing automatically. Frequently-occurring point-to-point distances between vertices are recorded. The smallest few of these common distances is indicative of the unit of resolution the floor plan creator intended. The greatest common denominator of these distances is used as the grid spacing. For example, if the two most common distances between vertices are 1.25 in. and 0.50 in., the resulting grid spacing is 0.25 in.

Given the grid spacing, it is trivial to snap vertices to that grid. By applying the coerce-to-grid algorithm, most small (less than half the grid size) gaps and overlaps are fixed. The effectiveness of this approach is evident when it is applied to the Soda Hall floor plans. The algorithm produces a grid which roughly two-thirds of the vertices already lie upon, and snaps the other third of the vertices.

Figure 6 illustrates that the maximum possible distance between any vertex and the nearest grid point is equal $\sqrt{2} * s/2$, where s is the grid spacing. Figure 7 is a histogram showing the distribution of vertex deviation from the 0.25-inch grid for the 6th floor, Soda Hall floor plan. In all, 1265 of the total 1725 vertices already coincide with a grid point, and all other vertices are within 25% of the maximum deviation possible, suggesting that the automatically-produced grid was quite appropriate.

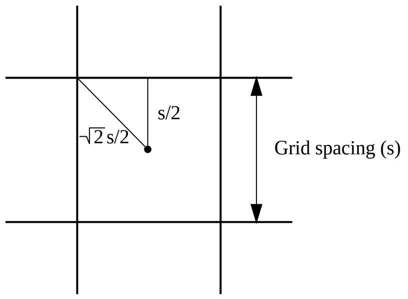


Figure 6: Maximum vertex deviation from grid.

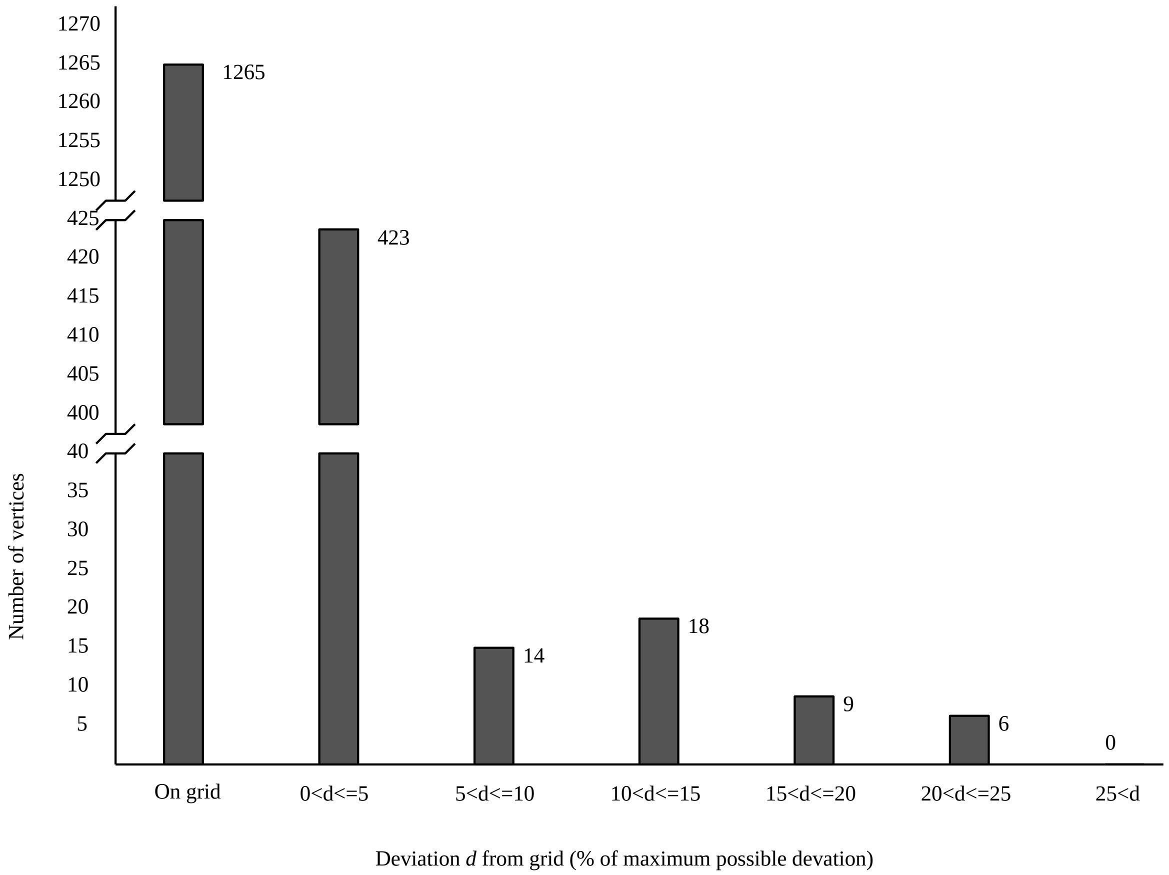


Figure 7: Histogram showing vertex deviation from generated grid.

6.2 Correcting Gaps

Gaps that are larger than half the grid spacing are not corrected by the coerce-to-grid algorithm. Typically, such gaps (Figure 8a and Figure 8b) arise when the floor plan creator draws line segments that are slightly too short, causing a gap in the contour. The intended contour (Figure 8c) must be produced by applying a gap-closing algorithm to the disjoint edges.

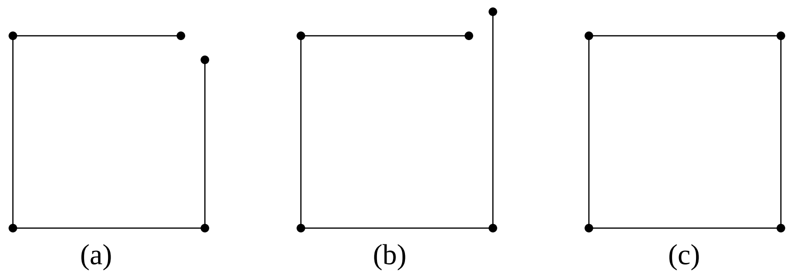


Figure 8: Typical contour gaps (a,b) and correct contour (c).

A fairly general approach can be used to fix most such gaps. Vertices that are referenced by only one edge are by definition indicative of disjoint edges; therefore, pairs of close vertices that meet this condition are located. If the edges that reference these vertices are parallel and collinear (Figure 9a), they are simply connected (Figure 9b). If parallel but not collinear (Figure 9c,e), they are connected perpendicularly (Figure 9d,f). If the two edges are perpendicular or nearly perpendicular, (Figure 9g), BMG computes the intersection of the lines that pass through the two edges and moves the two edge endpoints to the point of intersection (Figure 9h). Otherwise (Figure 9i), BMG intersects the line passing through one edge with the perpendicular to the other edge, and then two edges are created to close the gap (Figure 9j).

This approach does not correct all gaps; for example, it is possible for one edge to have a vertex belonging only to that edge, without another existing nearby vertex that has the same condition. Figure 10a shows an example situation where it is unclear to which neighboring vertex the highlighted vertex should be connected (recall: at this stage, there are no closed contours, the floor plan is just a large collection of edges). For such cases, an interactive method is provided for the user to specify how to close each gap. The user is allowed to specify what two endpoints are on opposite sides of the gap; then the general method for connecting the two endpoints is applied, resulting in the desired solution (Figure 10b).

6.3 Detecting and Fixing Incorrect Intersection

Edges that overlap (Figure 11a), instead of sharing endpoints, are equally as common as disjoint edges. The incorrect connectivity of these edges makes it impossible to form closed contours containing them. The edges must be made to share endpoints, such that appropriate connectivity will exist and a closed contour can be formed. This is accomplished by cutting each edge in an intersecting edge pair into two distinct edges (Figure 11b). The cut is made at the intersection point of the two original edges. Very small segments produced as a result of this

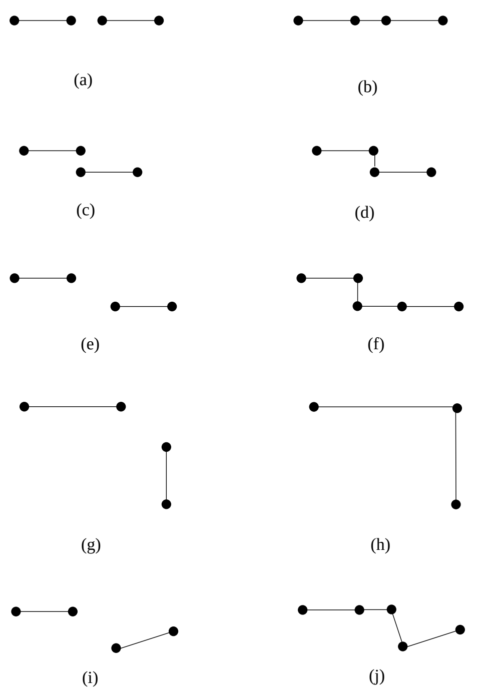


Figure 9: Correction of disjoint edges.

operation are assumed to be disposable if they do not share both endpoints with other edges, and therefore are discarded (Figure 11c).

7 Semantic Analysis of Floor Plans

Semantic analysis of the floor plans occurs in two stages. First, symbolic entities, such as door and window symbols, are converted to their literal equivalents. Second, closed contours defining the boundaries of individual spaces and portals are formed, and an adjacency graph is built by linking contours that share edges.

7.1 Converting Symbolic Geometry into Its Geometric Equivalent

Figure 12 shows two rooms as they appear before the semantic analysis phase. The door symbol positioned in the portal openings is one of the typical architectural symbols used for a door (Figure 13a). It is clearly not useful for extrusion purposes: that is, it is not a horizontal slice of the 3D doorway. BMG must replace each door symbol with two separate edges, one to close the room contour on each side of the doorway (Figure 13b). These edges serve two purposes: first, the edges will close otherwise non-closed contours defining the boundaries of the spaces on either side of the doorway. This makes it possible to then find these contours by means of a vertex-graph traversal that searches for closed loops with no internal sub-loops. Second, each

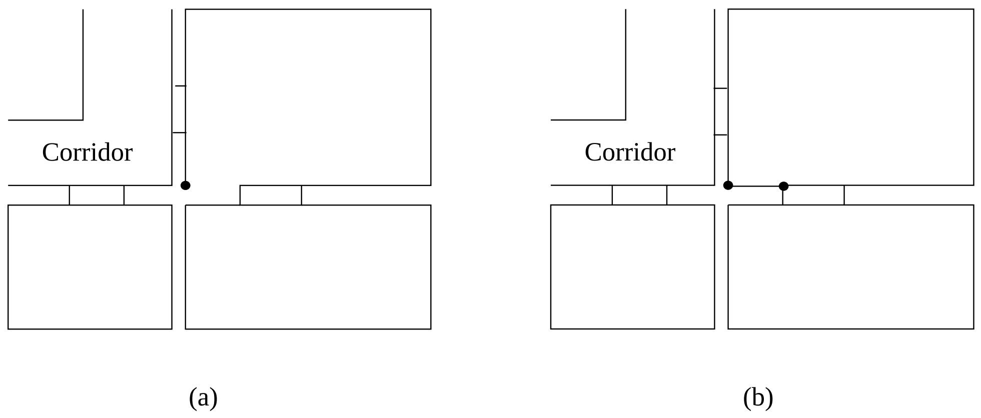


Figure 10: Example of ambiguous gap between floor plan edges.

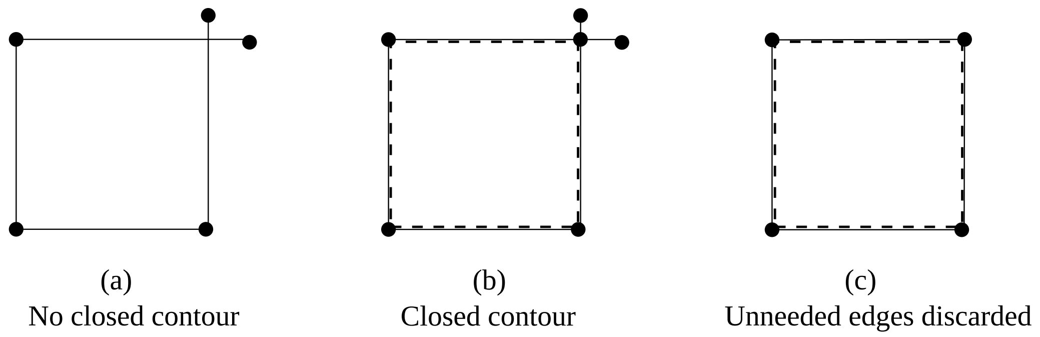


Figure 11: Correcting overlapping edges.

such edge will usually be extruded in the 3D model, in the form of a piece of wall extending from the top of the doorway up to the ceiling. The existence of each door is indicated by the existence of a geometric entity with a specific layer type. In its current implementation, BMG locates door symbols by searching for faces with layer type "DR-DR". In cases where the door symbol is composed differently (e.g. the door itself is a line segment, not a face), BMG could be modified to search for a different type of entity.

The two desired edges are formed using the following algorithm:

1. Locate a door symbol (look for a face with layer type "DR-DR").
2. Find nearest jamb edge to that door symbol.
3. Find opposite jamb edge (look for edge of same length as first jamb, door-length away from first jamb).
4. Connect the jambs with two specially-labeled portal edges.

Window symbols can be used as-is, since the appropriate contour-closing edges already exist as part of the window symbol itself.

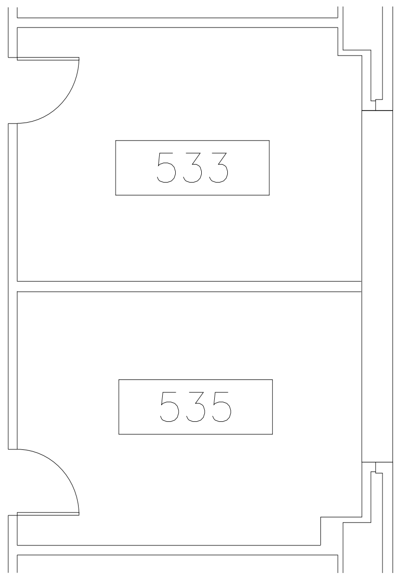


Figure 12: Door symbols as they appear on floor plan.

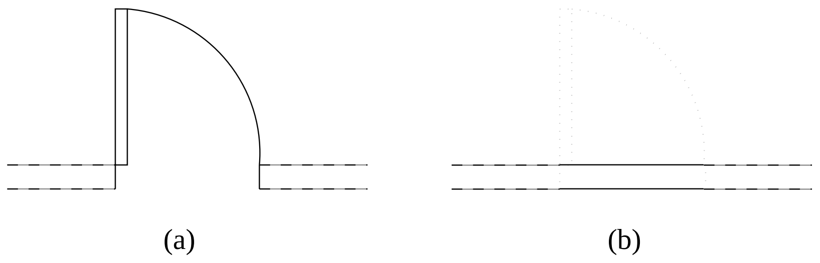


Figure 13: Door symbol before and after conversion.

7.2 Finding Spaces on the Floor Plan

As previously explained, it is desirable for extrusion and editing purposes to group boundary edges for each space into a single closed contour. This is accomplished by building a graph of floor plan vertices, intelligently choosing a starting vertex, and traversing the graph in an attempt to form a closed contour containing that vertex. The result is a set of closed contours for spaces and for portals, which are useful for extrusion into three dimensions.

7.2.1 The Vertex Graph

A non-directed graph of vertices and edges is built, that includes elements from all layers in the corrected floor plan. Each node in the graph consists of a 2D point, references to its neighbor nodes, and a status flag. A *neighbor node* of a given node is a vertex graph node that is directly connected to the given node by a single edge. Each node's neighbors are sorted by the slope of the line that contains the edge between the node and that neighbor. Merely searching the graph for closed loops would not necessarily result in the desired space contours, because most vertices are part of more than one closed contour. This situation is remedied by intelligently choosing the starting point and direction for a search, and by making appropriate subsequent decisions on how to proceed with the traversal at each node.

7.2.2 Choosing Starting Nodes for Vertex Graph Traversal

Searching the vertex graph for closed contours should result in counter-clockwise contours for interior spaces (such as rooms and corridors) and clockwise contours for exterior boundaries (i.e. the outside of the building). Therefore, when searching for interior-space contours, the first edge should be formed by picking a start vertex and second vertex such that the interior of the desired space is to the left of the directed edge (Figure 14). Choice of these first and second vertices is performed differently for known spaces, exterior boundaries, and any remaining unlabeled spaces.

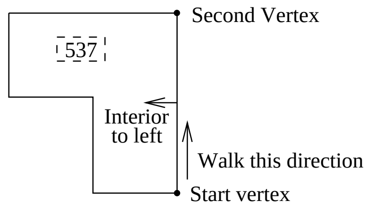


Figure 14: Direction of vertex graph traversal for interior spaces.

Rooms
The floor plan comes with room labels located near the center of numbered spaces on the floor plan. BMG looks for these labels and notes their position in space. To find closed contours for each room, the following algorithm is applied:

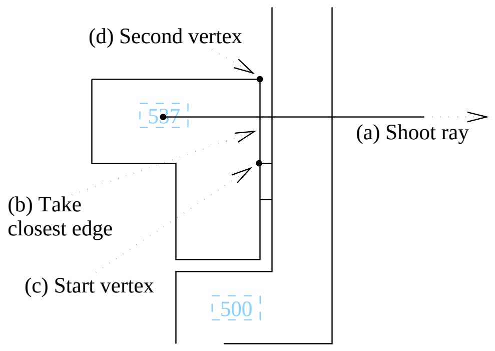


Figure 15: Finding starting edge and vertices for room contour search.

1. Shoot a ray from the label location in the +x direction (Figure 15a).
2. Find the closest intersected edge. This edge is assumed to be on the boundary of the desired space contour (Figure 15b).
3. Since we want to begin traversal in the +y direction (so that the interior of the room is always to the left of the current edge), set the first contour vertex equal to the edge endpoint with the lesser y value (Figure 15c).
4. Set the second contour vertex equal to the edge endpoint with the greater y value (Figure 15d).

There are conceivable situations where a space contour formed as a result of this algorithm will encompass more than one room label (Figure 16a). As a result, the entire space will be labeled with the identity of the room label that is used first as a seed for the contour search algorithm. In the case where exactly one contour per room label is required, the space contour could be partitioned into multiple adjacent contours placed reasonably about the room labels (Figure 16d), such that the union of the resulting regions is the original space. The following algorithm could be employed:

```
For each space contour S,
    1. Pick two room labels A and B enclosed by a single contour.
    2. Calculate the axis-aligned line L equidistant from A and B:
        (a) If labels are farther apart in x than in y, choose vertical (y-axis-aligned) line
            (Line 3 in Figure 16b, equidistant from labels 101 and 102).
        (b) else choose horizontal (x-axis-aligned) line (Line 5 in Figure 16c, equidistant
            from labels 102 and 103).
    3. If there is a space contour edge E parallel and close to line L (Edge 1 in Figure 16a
       for line 3 in Figure 16b, and edge 2 in Figure 16a for line 5 in Figure 16c), Move L
       so that it is collinear with E
    4. Partition the space contour S at line L, producing edges 4 and 6, respectively, in
       Figures 16b and 16c.
```

The algorithm is repeated until there is exactly one contour per room label. The resulting collection of contours is shown in Figure 16d. This algorithm attempts to break up the original contour in accordance with existing walls wherever possible. If the algorithm simply partitioned the space without taking existing wall edges into account, the result (Figure 16e) may not agree with the intuitive boundaries that seem to be implied by the existence of changes in wall direction.

In Figures 16a-c, the pair of room labels 101 and 102 is used first, followed by 102 and 103 (the remaining pair of 101 and 103 does not require a partitioning operation, since the first partitioning operation in Figure 16b separates labels 101 and 103). If the order of pair choice were reversed, such that labels 102 and 103 are used first, followed by 101 and 102, the results would be different (Figure 16f). Therefore, the user should be allowed to choose the order with which the pairs of room labels are chosen, so that the resulting spaces match the user's subjective interpretation of the intentions of the floor plan creator.

Exterior

It is not as straightforward to find exterior-boundary contours, since there are no room labels to use as seeds for finding the first and second vertices. The following algorithm is employed to find the starting edge:

1. Determine the X and Y value extents for the entire floor.
2. Let curY go from minY to maxY in increments equal to grid spacing.
   (a) Shoot ray from the seed point (minX - 10.0 , curY) in the +x direction (Figure 17a).
   (b) Find the closest intersected edge, if any.
   (c) If edge is unused (Figure 17b):
      i. Since we want to begin traversal in the +y direction (so that the interior of the contour is always to the right of the current edge, and the exterior empty space is on the left), set the first contour vertex equal to the edge endpoint with the lesser y value.
      ii. Set the second contour vertex equal to the edge endpoint with the greater y value.
   (d) else, edge is already used in contour (Figure 17c). Increment curY.

Note that edges marked (c) in figure 17 are already used because the contour found by the search originating from the start edge labeled (b) includes the entire exterior contour for that wing.

Most exterior contours can be found this way. In the unusual case that there are sufficient separate wings on a single floor plan such that no ray generated by the algorithm hits the exterior of a particular wing (Figure 18a), user interaction is required. The user provides the start edge by clicking on a single exterior edge (Figure 18b), and then the normal contour search is performed.

Remaining Contours

Once all room contours and exterior contours have been formed, interior spaces may remain that are not labeled with room numbers, such as elevator shafts and unassignable spaces used for heating, electricity, and plumbing. Since the vertices in labeled-room contours and exterior-boundary contours are marked as used, it is possible to simply search the remaining unused nodes in the vertex graph for closed contours. Vertices in the resulting contours are forced into counter-clockwise order because they are interior to the building, a fact that is guaranteed because the exterior contours were formed previously.

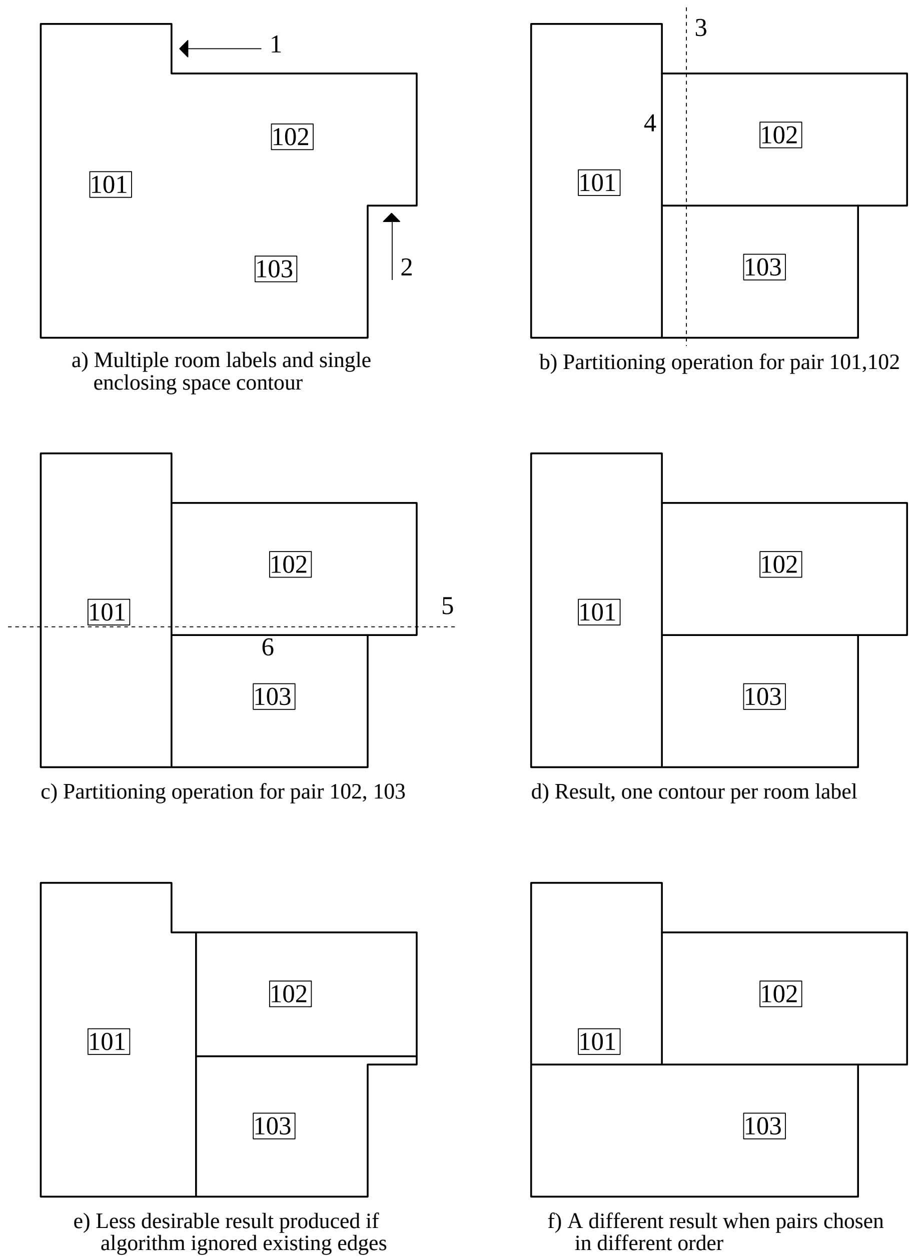

1
2
101
102
103
a) Multiple room labels and single enclosing space contour
3
4
b) Partitioning operation for pair 101,102
5
6
c) Partitioning operation for pair 102, 103
d) Result, one contour per room label
e) Less desirable result produced if algorithm ignored existing edges
f) A different result when pairs chosen in different order

Figure 16: Partitioning spaces to obtain one space per room label.

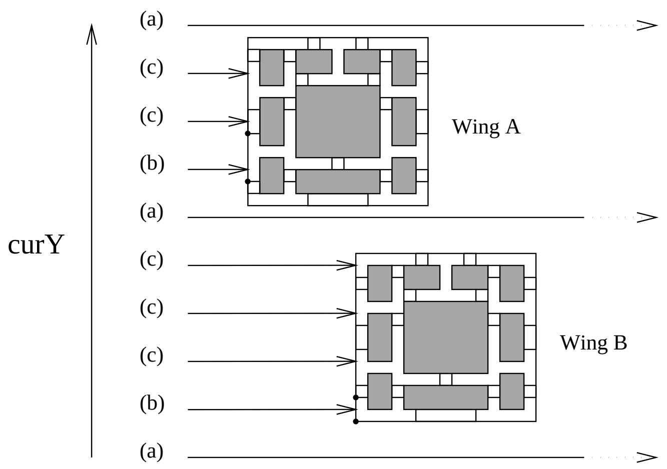


Figure 17: Finding starting edge for exterior contour search.

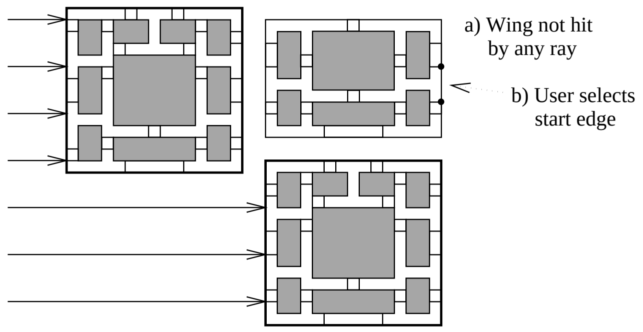


Figure 18: Exterior contour not automatically found.

7.2.3 Vertex Graph Traversal

The vertex graph is traversed until either a closed contour is formed, or all possible paths from the start vertex have been searched without success. The traversal backtracks where dead ends are reached. When the start node is reached, a closed contour has been formed, and the vertices in that contour are marked used so that they can not be incorporated into other closed contours.

The algorithm for the vertex graph search is as follows:

1. Start with contour containing two vertices (Figure 19a), given by algorithms in section 7.2.2, "Choosing Starting Nodes for Vertex Graph Traversal".
2. Set current node to last (second) contour vertex.
3. Begin algorithm: choose next neighbor node in search path: leftmost turn (Figure 19b, 19c, 19d, 19e).
4. If there are no unvisited neighbor nodes, remove current node from list of contour vertices and backtrack (Figure 19f) to previous node (go to 3). If back to second node in contour list, no solution (quit).
5. Add new node to list of contour vertices.
6. Is start node a neighbor of current node?
    (a) Yes: Closed contour found (Figure 19g). Mark contour nodes used. Done.
    (b) No: Go to 3.

The sorted angles of edge incidence at each vertex graph node guarantee traversal in the desired (clockwise or counter-clockwise) manner.

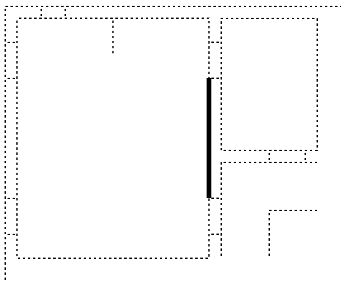
A *SPACE* node is created for each closed contour found by this algorithm. The SPACE node contains a list of edges defining the contour. SPACE nodes are labeled with a unique identifier and name. The name is derived from knowledge of the floor plan wherever possible; for example, those contours that were found by using room labels as seed points are given the name that was present in the room label itself (e.g. 537) and the type associated with that room label (e.g. OFFICE). In this process, the *layer* of the original line segment for each contour edge is remembered; this is necessary because, for example, an edge of type *interior-wall* should be extruded as a single rectangular wall piece, while an edge of type *window-sill* should be extruded in two pieces with a hole between them for the window portal. The list of SPACE nodes is later used as the basis for extrusion and editing.

Figure 20 shows the floor plan from Figure 4 after space contour formation.

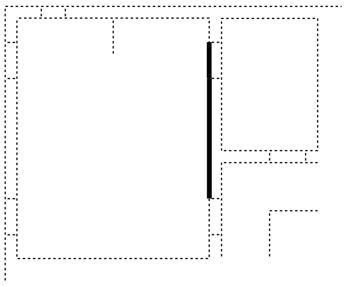
7.3 Finding Portals on the Floor Plan

The same contour-finding technique applied to spaces on the floor plan can also be applied to portals (doors and windows). In fact, these portals can be viewed as spaces themselves, since the outline of each portal region on the floor plan is precisely the area occupied by that portal (Figure 21).

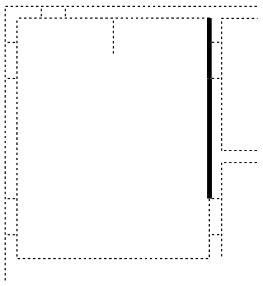
In the Soda Hall floor plans, door contours are composed of two layers, one layer for the jambs and one layer for the edges that span the width of the door opening itself. A door contour



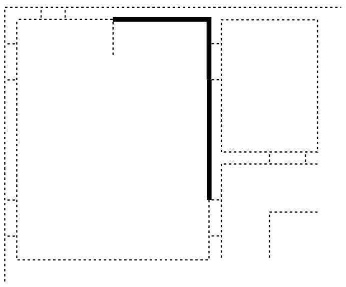
a) Start edge



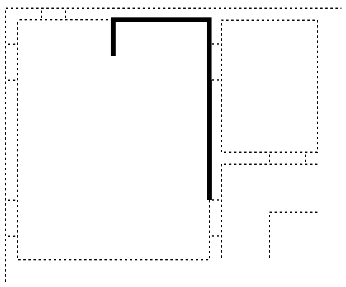
b) Proceed by choosing leftmost of two neighbor nodes



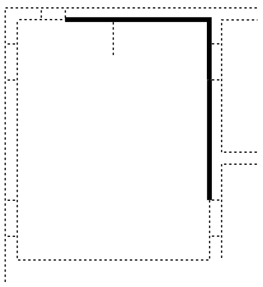
c) Proceed by choosing lef of two neighbor nodes



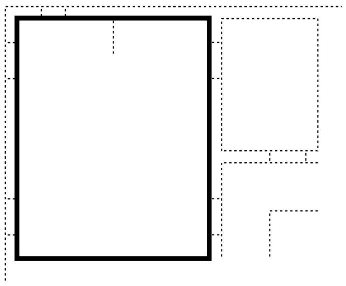
d) Proceed to only neighbor



e) Proceed by choosing leftmost of two neighbor nodes. Dead end.



f) Backtrack; proceed to other neighbor node



g) Result upon completion of search algorithm

Figure 19: Searching for interior contour.

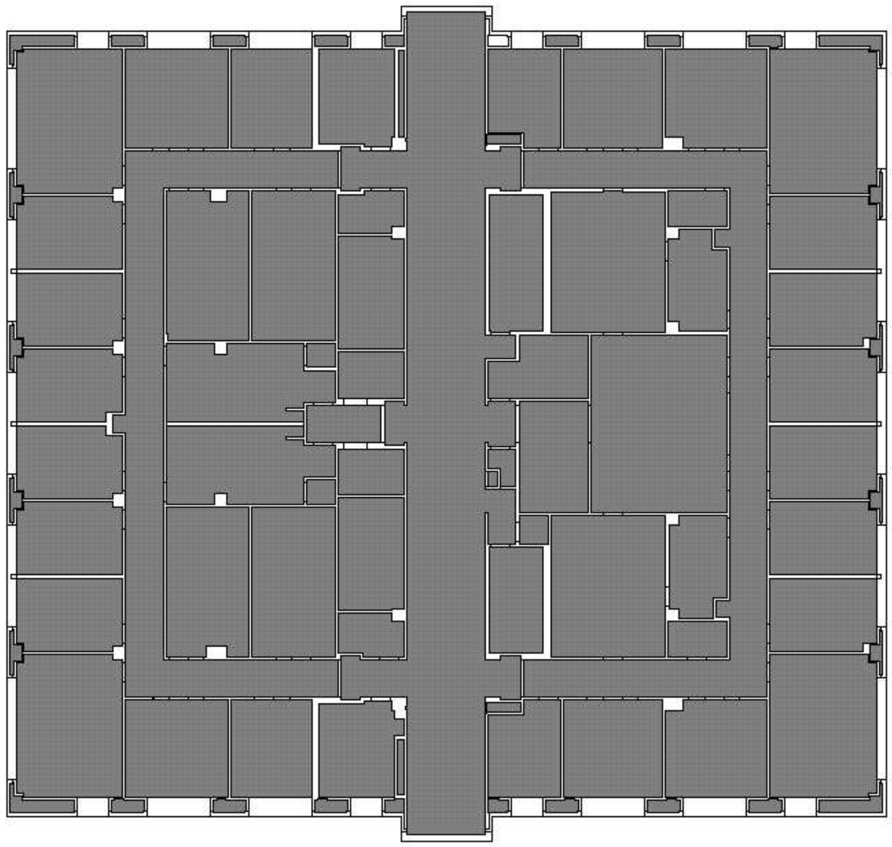


Figure 20: Spaces found on Soda Hall floor 5 plan.

consists of two jamb-layer edges and two door-width edges. Since the door openings were closed off with door-width edges during preparation for finding room contours, those edges can be used for finding closed door contours as well. Window contours are similarly constructed out of four edges, two each from the window-jamb layer and the window-sill layer. See section 17 for a discussion of floor plan design recommendations that would lead to easier location of portal contours.

To find doors, a vertex graph is built that includes only vertices in the door-jamb and door-width layers. Closed contours are found by starting with unused vertices and recursively searching for the shortest route to close the contour. The procedure is the same for windows, where the vertex graph contains only window-jamb and window-sill layers.

A PORTAL object is created for each portal contour found. Similar in its purpose to a SPACE object, the PORTAL object is a unit of data that includes information about the geometry of the portal contour, pointers to spaces that are connected by the portal, as well as other information that is utilized in the extrusion phase. A linked list of PORTAL objects is created.

Once the lists of SPACE and PORTAL objects are complete, analysis is performed to doubly link portals and the spaces they connect. Where a portal and a space share an edge, the portal

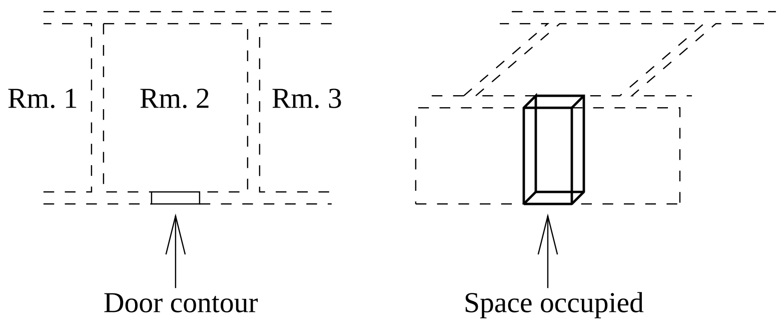


Figure 21: Portal contour and corresponding 3D geometry.

object is given a pointer to that space object, and that space object is given a pointer to the portal object. This connectivity information is useful in several ways. First, after extrusion during 3D editing, if a portal opening's dimensions are modified, the opening in each adjacent space's wall must also be modified. The link between the portal object and its adjacent spaces provides instant knowledge of what spaces need to be re-extruded to reflect the changes to the portal's geometry. In addition, the connectivity information is useful when the resulting model will be viewed using a walkthrough system that requires specification of such connectivity information in order to perform visibility computations that enhance rendering speed.

7.4 Using Reflected Ceiling Plans

Prior to extrusion, the user has the option to load a reflected ceiling plan (RCP), which specifies ceiling objects such as lights and sprinklers, and denotes regions of the ceiling that have different heights. This Reflected Ceiling Plan is an architect's standard means of specifying the various different ceiling heights for regions on the floor plan. The regions are denoted by diagonals that specify the exact location and dimension of the corresponding rectangular region. The user must be able to specify the height of each individual RCP region; therefore, the space must be partitioned according to the boundaries of the RCP regions it contains, resulting in several distinct spaces whose properties (including ceiling height) are individually addressable.

The input RCP consists of diagonals, or pairs of diagonal line segments grouped into a single layer. Figure 22 shows the reflected ceiling plan for the floor plan shown in Figure 4.

7.4.1 Identifying Ceiling Regions on RCP

Each RCP diagonal edge is checked to see if it is one of an intersecting pair of RCP edges that both define the same rectangle. Only one diagonal is necessary to define the rectangular region. For each such pair, one of the diagonals is discarded. For each of the remaining diagonals, an RCP-OBJECT is created, which contains the coordinates of the diagonal endpoints, the four edges that form the bounding box around the diagonal and thus define the region, and a pointer to the SPACE that encloses that region.

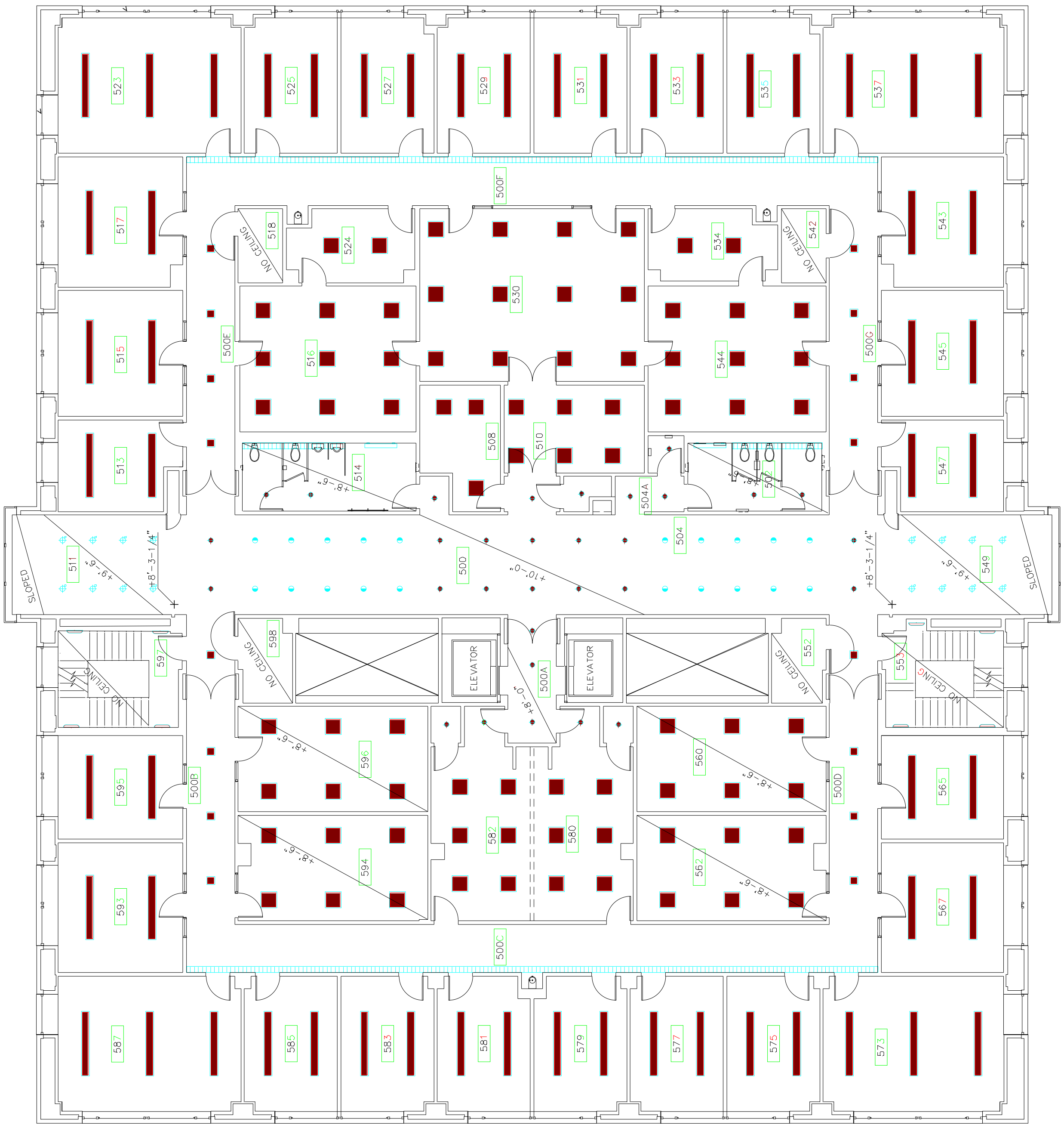
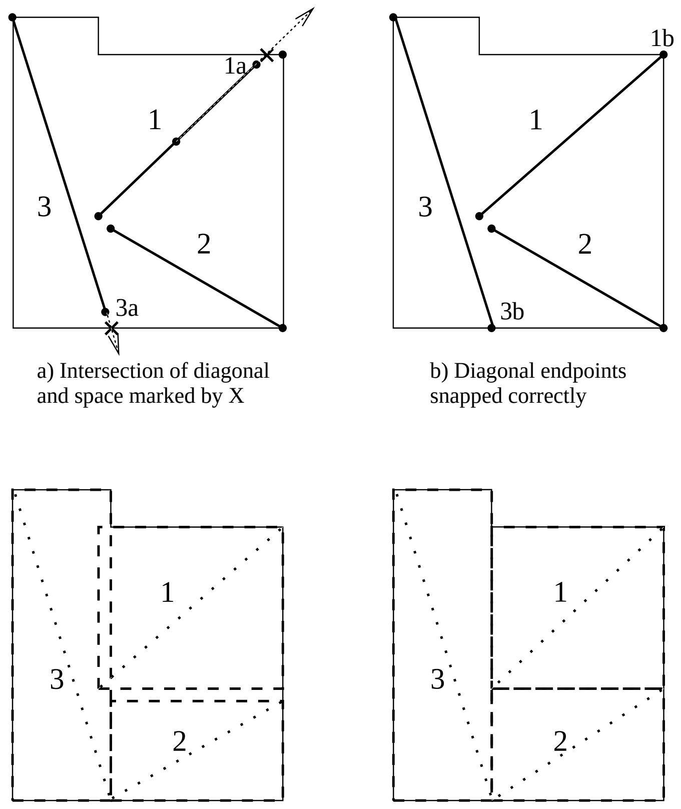


Figure 22: Reflected ceiling plan.

7.4.2 Correcting RCP Regions

Often, the boundary edges of each RCP region do not correctly match with the underlying space: there may be a small gap between a region and the wall it is intended to be adjacent to, or there may be overlap. These problems are corrected by snapping the RCP regions against walls they are very close to.



a) Intersection of diagonal and space marked by X

b) Diagonal endpoints snapped correctly

c) Regions 1 and 3 overlap; gap between regions 1 and 2

d) Region 1 shrunk to right; region 2 expanded upward

Figure 23: Fixing reflected ceiling plan.

The snapping of RCP regions works as follows:

```
For each endpoint of an RCP region diagonal,
    1. Extend the diagonal line beyond that endpoint.
    2. Find the first of the space's edges that is intersected.
    3. If the intersection point (Figure 23a- 1a and 3a) is close to the endpoint,
        (a) If the intersection point is near an existing space vertex (Figure 23a- 1a), move
            the endpoint to that vertex (Figure 23b- 1b).
        (b) else (Figure 23a- 3a), move the endpoint to the intersection point (Figure 23b-
            3b).
    4. else (intersection point is not close to endpoint),
        (a) If the endpoint itself is close to an existing space vertex, move the endpoint to
            that vertex.
        (b) else, snap the endpoint to the floor plan grid.
```

Just as the RCP regions may not line up with their underlying spaces, multiple RCP regions may not be properly adjacent to one another. This can result in very thin implicit regions between the specified regions. BMG shrinks or expands regions appropriately to eliminate these errors. The algorithm:

```
For each pair of RCP regions in the space (Figure 23c),
    For all pairs of edges (one each from each RCP region) that are close and parallel,
        1. Pick that edge E not confined by the boundary of the space, that is a member
           of the smaller of the two RCP regions.
        2. Move E (growing or shrinking the smaller RCP region), so that E is collinear
           with edge from larger RCP region (Figure 23d).
```

After correction of these RCP regions, the underlying space can be partitioned along the boundaries of the RCP regions. Note that areas on the underlying space that are not covered by an RCP region are implicitly regions themselves, which acquire default ceiling heights.

7.4.3 Breaking-up Spaces at RCP Region Boundaries

Spaces are partitioned along RCP region boundaries by performing boolean subtraction of polygons: each RCP region is subtracted from the underlying space in succession. The original space is then replaced by the group of new, smaller spaces that result from the partitioning operation. The non-geometric information associated with the original space, such as name and type, is transferred to each new space. Edge types, such as *interior-wall* or *window-sill* are transferred to the new space edges where applicable; of course, new edges that were created by the partitioning operation are given the special type *open* so that they are not extruded.

8 Construction of the Three-Dimensional Model

Construction of a consistent 3D model of the floor is possible once the floor plan has been corrected and analyzed. Figure 24 shows the 3D floor model for the fifth floor of Soda Hall, which corresponds to the floor plan shown in Figure 4.

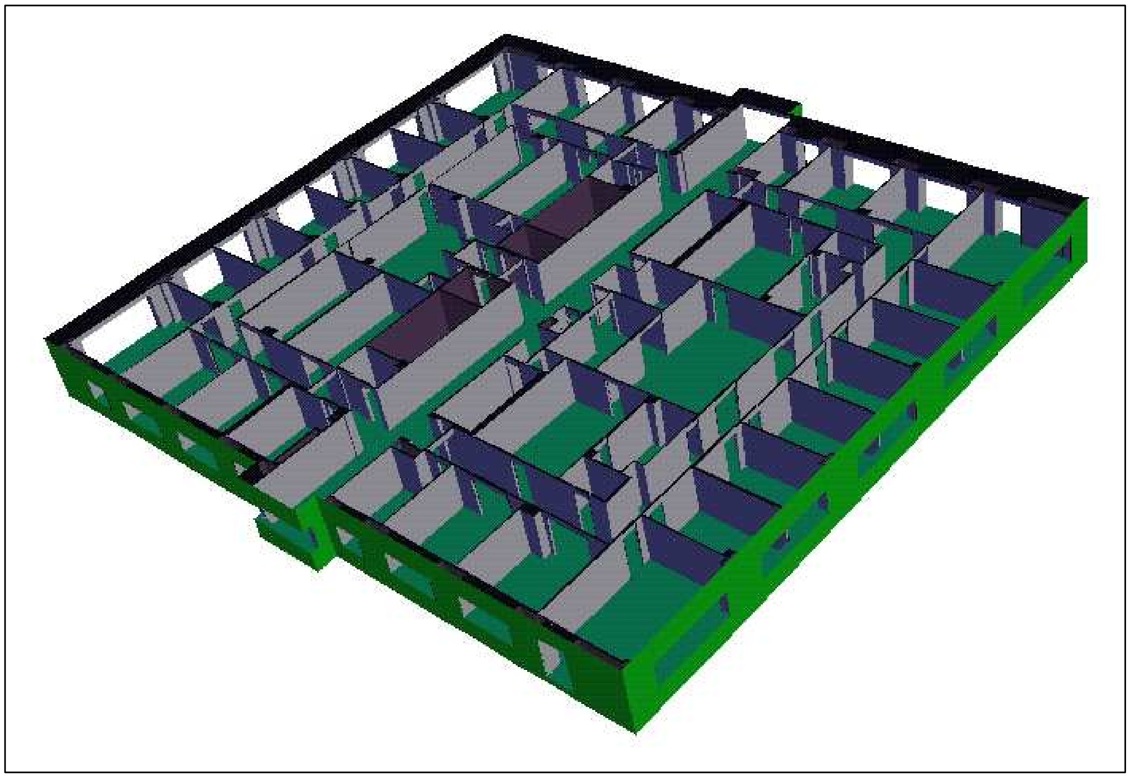


Figure 24: Generated 3D model, ceiling polygons excluded for viewing purposes.

The cleaned-up floor plan is extruded to form the walls of the 3D model. A floor and ceiling polygon is generated for each space. Doors and windows consist of holes in the walls of the connected spaces, plus polygons that connect the adjacent spaces. The processes for generating these components of the 3D floor model are described in detail in the sections below.

8.1 Construction Parameters

There are many properties of the resulting 3D model that are not directly specified by the 2D floor plan. The most essential of these is height information, which is absolutely necessary for constructing the 3D geometry. Clearly, height can not be extracted from a 2D drawing such as a floor plan; therefore, the user must specify the proper heights of ceilings, door openings, and window openings (Figure 25).

There exist many non-geometric parameters that the user may modify as well. In general, these

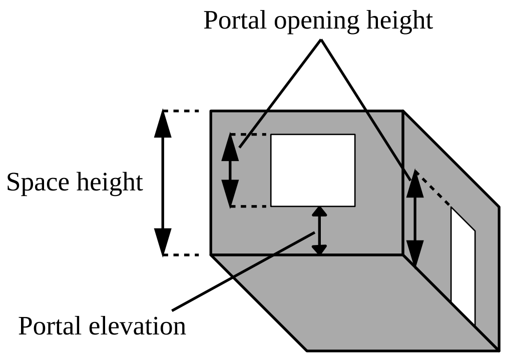


Figure 25: Heights specified by user.

parameters correspond to properties of the 3D model such as colors and textures.

8.1.1 Space Parameters

Space construction is affected by the following parameters, all of which can be specified and modified by the user:

Parameter	Type
Ceiling height	Numeric (float)
Wall color/texture	UG color name
Floor color/texture	UG color name
Ceiling color/texture	UG color name
Existence of ceiling	Boolean
Existence of floor	Boolean

The user may specify values for these parameters in a number of ways; first, default values can be specified, which are used for all spaces. Second, the user may edit space parameters while viewing the 2D floor plan, prior to generation of the 3D model. Finally, the user may edit a space's parameters after the 3D model has been generated; in this case, the BMG system regenerates the affected space in the 3D model. BMG uses reasonable default values if the user does not specify values for the parameters. Space editing is covered in section 9.1.

8.1.2 Portal Parameters

Since the height of each portal opening (Figure 26a), and the elevation of the base of each opening (Figure 26b), are not specified by the floor plan, the user must specify those parameters. In addition, the color and texture of the four polygons that connect the two spaces via the portal must be specified.

In summary, the parameters and their types are:

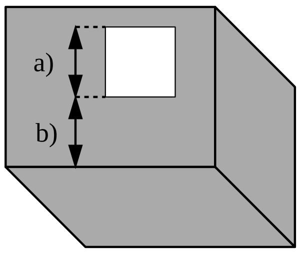


Figure 26: Portal parameters specified by user.

Parameter	Type
Opening height	Numeric (float)
Base elevation of opening	Numeric (float)
Color/texture	UG color name

In addition to these parameters, a portal *type* may be associated with each portal. Each type maintains the same parameters listed above. If an individual portal has an associated type, the parameter values of the type are applied to that portal. Portal types are discussed in more detail in section 9.3.1.

Like space parameters, portal parameters default to some reasonable values, set by the user, which apply to all portals. Each portal may be selected and modified by the user in both the 2D floor plan and the 3D model view.

8.2 Generating Spaces

The BMG system iterates over the list of SPACE objects and creates the geometry for the 3D representation of each space. For each space, the list of edges that comprise the closed space contour is traversed. Information is maintained with each edge that dictates how that edge should be extruded in the 3D model.

Edges fall into three categories, each of which is extruded differently: a *wall* edge is unique to the space being extruded and is extruded into a rectangle. A *shared-portal* edge is a contour edge that is shared with an adjacent portal. A shared-portal edge is extruded into two separate rectangles; one from the bottom of the space to the bottom of the portal, and one from the top of the portal to the top of the space. A *shared-space* edge is shared with an adjacent space. A shared-space edge is not extruded, unless the ceiling height of the adjacent space is different from the space being extruded. All polygons created through edge extrusion are oriented toward the interior of the space being extruded; that is, the normal to each polygon passes through the room volume (Figure 27).

The following sections contain detailed descriptions of how each type of edge is handled during the extrusion process.

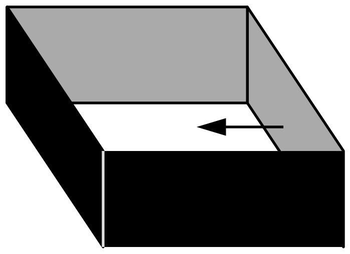


Figure 27: Orientation of wall polygons

8.2.1 Extruding *wall* edges

Each wall edge is extruded to form a rectangular wall polygon (Figure 28). The polygon extends in Z from the bottom of the space (Z_b in Figure 28) to the top of the space (Z_t in Figure 28). The Z values for these two parameters are obtained from the SPACE object currently being extruded. The value for Z_b is usually equal to the offset of the floor itself from the base elevation of that floor level (i.e. the height of the raised floor). Use of this offset allows accurate representation of buildings with raised floors.

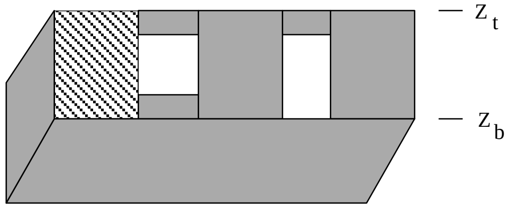


Figure 28: Wall edge extrusion.

8.2.2 Extruding *shared-portal* edges

A space contour edge that is shared by an adjacent PORTAL object is extruded into two rectangular pieces, forming a wall with a suitably sized hole for access through the portal (Figure 29). The lower polygon extends in Z from the bottom of the space ($Z_{b,space}$ in Figure 29) to the bottom of the portal opening ($Z_{b,portal}$ in Figure 29). In the case of doors, this polygon is not generated, since the $Z_{b,space}$ and $Z_{b,portal}$ values are the same. The upper polygon extends in Z from the top of the portal opening ($Z_{t,portal}$ in Figure 29) to the top of the space ($Z_{t,space}$ in Figure 29). The values for $Z_{b,space}$ and $Z_{t,space}$ are obtained from the SPACE object being extruded. The values for $Z_{b,portal}$ and $Z_{t,portal}$ are obtained from the PORTAL object that shares the edge being extruded.

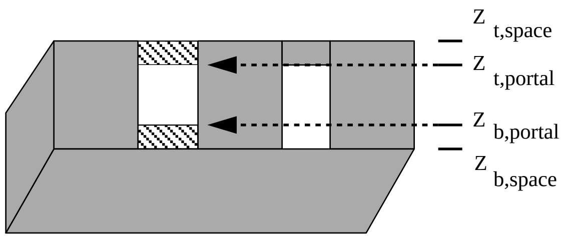


Figure 29: Portal edge extrusion.

8.2.3 Extruding *shared-space* edges

A space contour edge that is shared by another space is generally not extruded. This is because shared edges between spaces may only exist for two reasons: either the user partitioned a single space into two spaces manually, or BMG broke up a single concave space into two individual convex spaces. In both cases, there is no actual physical boundary between the two spaces. However, the edge is extruded if the ceiling heights of the two spaces that share the edge are different. In this case, a rectangular polygon is generated (Figure 30), which extends from the height of the shorter space (Z_s in Figure 30) to the height of the taller space (Z_t in Figure 30). The values for Z_s and Z_t are obtained from the two SPACE objects that share the edge.

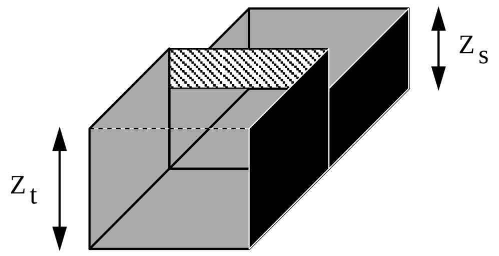


Figure 30: Extrusion at edge shared by spaces.

The following table summarizes extrusion of space contour edges:

Edge type	Construction	Parameters	Parameter Value Source
Wall	1 rectangle	Bottom of space	SPACE object
		Top of space	SPACE object
Shared-portal	2 rectangles	Bottom of space	SPACE object
		Top of space	SPACE object
		Bottom of portal	Adjacent PORTAL object
		Top of portal	Adjacent PORTAL object
Shared-space	0 or 1 rectangle	Top of space	SPACE object
		Top of adjacent space	Adjacent SPACE object

8.2.4 Creating Floors and Ceilings

Extruded spaces usually include a ceiling (downward-facing polygon) and floor (upward-facing polygon) (Figure 31). There are some spaces for which a floor and/or ceiling is unnecessary. For example, stairway and elevator shafts traverse multiple floors of the building, and therefore have neither a floor or ceiling except for the very bottom of the shaft (a floor polygon) and the very top of the shaft (a ceiling polygon).

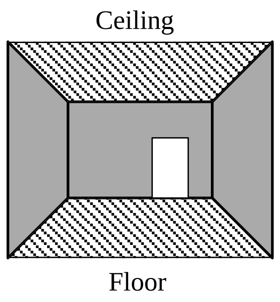


Figure 31: Ceiling and floor polygons.

Wall extrusion is affected by the existence of floor and ceiling polygons. If a space is marked with the flag specifying not to generate a floor, then the bottom of the space (and therefore the base of each wall in that space) will be set to the value of the base elevation for the entire 3D model of that floor level, instead of the raised floor offset. This is necessary so that shaft-type spaces on adjacent floor levels will exhibit proper connectivity by sharing edges, rather than being separated by an interstice equal to the raised floor offset.

The exterior of the building is generated in the same way as the interior spaces, with two exceptions. First, the exterior wall extends down to the base elevation of that level of the building, rather that the height of the raised interior floors. Second, the polygons are oriented to the outside of the building, rather than the interior.

In total, proper connectivity of the polygons generated for each space is guaranteed, since the floor plan cleanup and analysis produces only closed, non-self-intersecting contours. This demonstrates the advantage of the "cleanup and extrusion" approach: procedural generation of the polygons comprising the 3D model eliminates many of the errors that are introduced when polygons are drawn and positioned individually with a 3D CAD tool by a draftsperson.

8.3 Generating Portals

For each portal in the list of PORTAL objects, four polygons must be generated in order to properly connect the openings in the walls of the spaces connected by that portal (Figure 32). These polygons are created by extruding portal contour edges that are not shared with either of the adjacent spaces (Figure 32a,b), and by generating a *portal floor* (Figure 32c) and a *portal ceiling* (Figure 32d) polygon. Portal contour edges that are shared by adjacent spaces are not extruded, since the "hole" connecting the two spaces must remain open (Figure 32e).

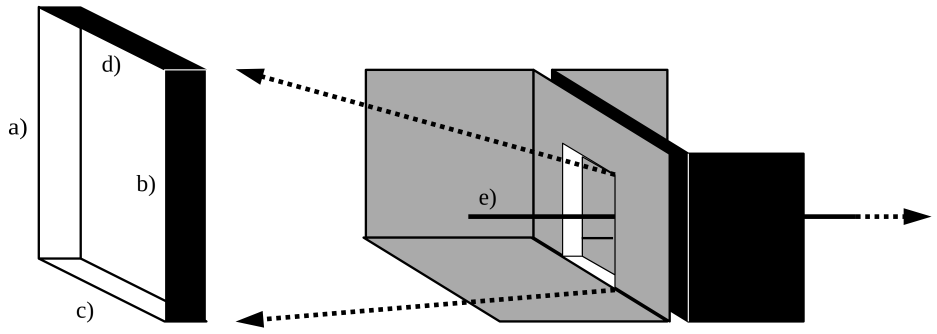


Figure 32: Portal generation.

8.3.1 Extruding *non-shared* Edges

Each non-shared edge (Figure 33a) is extruded to form a rectangular polygon (Figure 33b). The polygon extends in Z from the bottom of the portal opening (Z_b in Figure 33b) to the top of the portal opening (Z_t in Figure 33b). The Z values for these two parameters are maintained in the PORTAL object.

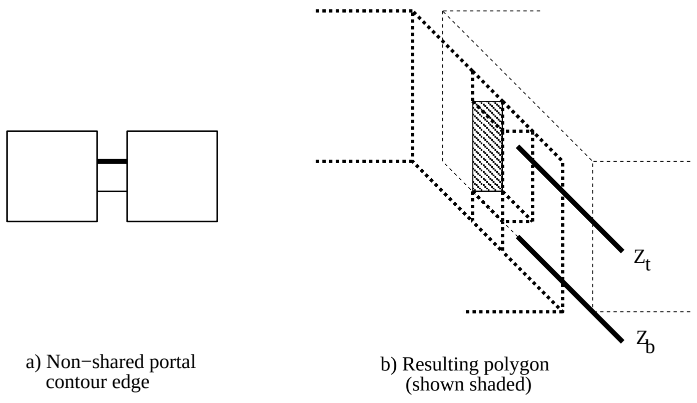

a) Non−shared portal contour edge
b) Resulting polygon (shown shaded)
Z_t
Z_b

Figure 33: Extrusion of *non-shared* portal edge.

8.3.2 Extruding *shared-portal* Edges

Similar to a space edge that is shared by two spaces, a portal edge that is shared by two portals (Figure 34a) must be extruded if the Z values of either the bottom or top of the two portals differs. The Z values for the top and bottom of each portal is maintained in each PORTAL object. In the case where the Z values for the bottom of the two portals differ, a rectangle is created that extends in Z from the bottom of the lower portal (Z_{low} in Figure 34b) to the bottom of the higher portal (Z_{high} in Figure 34b). In the case where the Z values for the top of the two portals differ, a rectangle is created that extends in Z from the top of the shorter portal (Z_{short} in Figure 34b) to the top of the taller portal (Z_{tall} in Figure 34b). The two cases may coexist, as demonstrated by Figure 34b.

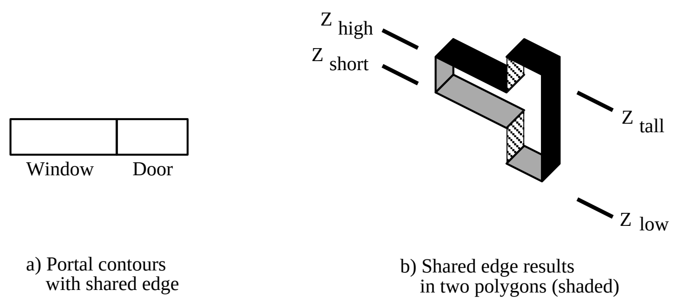

Window
Door
a) Portal contours with shared edge
Z high
Z short
Z tall
Z low
b) Shared edge results in two polygons (shaded)

Figure 34: Extrusion of *shared* portal edge.

The following table summarizes extrusion of portal contour edges:

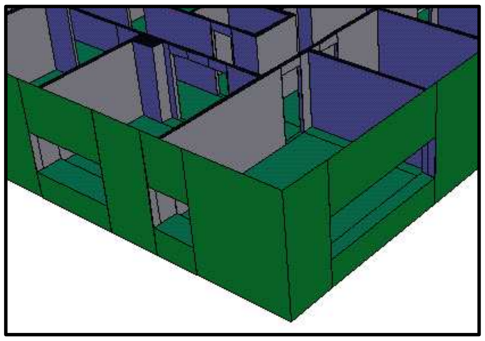
Edge type	Construction	Parameters	Parameter Value Source
Non-shared	1 rectangle	Bottom of portal	PORTAL object
		Top of portal	PORTAL object
Shared-portal	0, 1, or 2 rectangles	Bottom of portal	PORTAL object
		Top of portal	PORTAL object
		Bottom of adjacent portal	Adjacent PORTAL object
		Top of adjacent portal	Adjacent PORTAL object
Shared-space	Not extruded	N/A	N/A

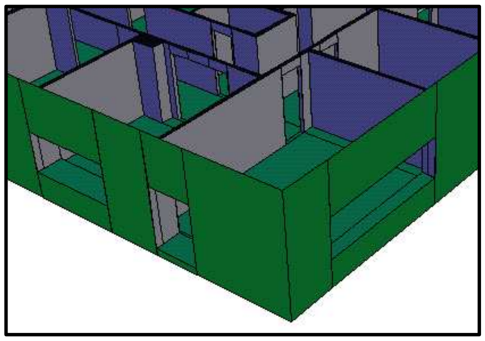
Polygons created by extruding portal contour edges are oriented toward the interior of the portal contour. Polygons created in the *shared-portal* case are oriented towards the interior of the portal with the lower bottom Z value (in the case where the portals differ in bottom Z value), or the portal with the higher top Z value (in the case where the portals differ in top Z value).

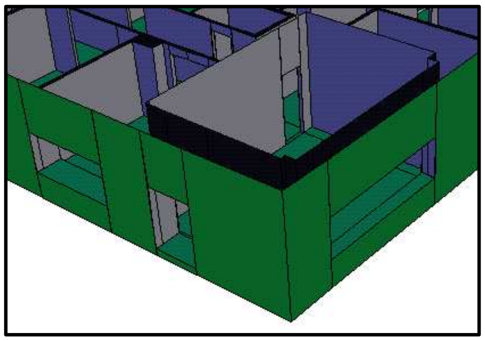
One floor polygon and one ceiling polygon is created for each portal contour. These polygons consist of the same two-dimensional contour as the portal contour itself; only the Z values differ, with the upward-facing floor polygon at the bottom of the portal opening, and the downward-facing ceiling polygon at the top of the portal opening.

9 Editing the 3D Floor Model

The BMG system generates the 3D model for each corrected floor plan using a set of reasonable default parameters, properties, and colors. The user may modify the 3D model after its construction. This feature is necessary so that certain rooms with different ceiling heights from the floor-wide default value can be properly constructed. Editing is also useful for changing the dimensions and frames of portal openings, and for changing colors and textures of the polygons comprising rooms, floors, and ceilings. Figure 35a shows a closeup view, with edges displayed for clarity, of some of the spaces and portals from the model in Figure 24. This set of spaces and portals is provided as a basis for the sections below, which describe editing features.


a) Original corner space


b) Portal opening size increased


c) Space height increased

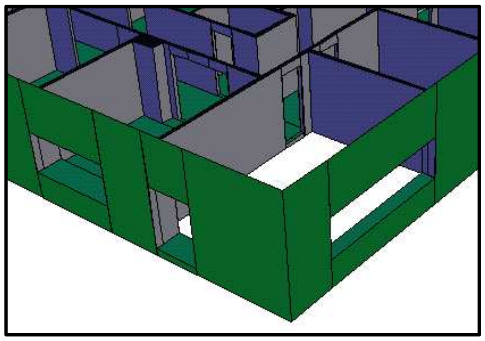

d) Make Floor set to FALSE

Figure 35: Closeup of generated model, with examples of 3D model editing features.

9.1 Editing Spaces

The user may modify spaces after generation of the 3D model. Clicking on the floor polygon, or a wall polygon, of the space the user wishes to edit causes the *space editing* form to pop up. Through this form, the user may change any of the parameters for that space, including height, existence of ceiling and floor, and colors and textures.

Figure 35c shows the corner room from Figure 35a after its ceiling height has been increased. Figure 35d shows the original space after *Make Floor* has been set to *FALSE*. Note how there is no floor polygon. Also note that the walls extend down to the base elevation of that floor level, as opposed to the height of the raised floor; a small step can be seen below the door leading from the room. This is because spaces without floors are used to form shafts (e.g. elevator and staircase) that connect multiple floor levels.

Changes to space parameters can be applied at various levels of granularity, to provide convenient editing for the user. Each space has a name, type, and floor associated with it; therefore, the changes made to a space may be applied at the following levels: the individual space, all spaces of a particular type, and all spaces on the current floor.

9.2 Editing Portals

Portals may be modified after generation of the 3D model. Clicking on any of the four polygons comprising a portal causes the *portal editing* form to pop up. Through this form, the user may change any of the parameters for that portal, including opening height, opening base elevation, and color and texture.

Figure 35b shows a portal from Figure 35a after the user has increased the opening size by 50 percent.

Changes to portals can be applied at three levels of granularity: the individual portal, all portals of a particular type, and all portals on the current floor. With this mechanism, the user can, for example, conveniently modify the size of all windows on the exterior of the building by simply applying changes to all portals of type "window".

9.3 Inserting Detailed Models into Portals

The 3D generation process described so far generates proper openings for each portal. In order to make the model more realistic, it is desirable to include details such as the frame and glass of a window, or the frame and jamb of a doorway. If the detailed component is pre-modeled, insertion of a properly translated and rotated instance of the component can be automated, since the floor plan analysis phase yields the location and orientation of the portals.

BMG allows the user to define portal types. Each type has associated with it the size of the portal opening in the 3D model, and an associated portal prototype model (Figure 36a). Portal prototype models include geometry for details such as door jambs and window frames and glass. BMG automatically inserts an instance of the defined prototype model into the portal opening for each portal with that portal type (Figure 36b). If the user changes the frame model for a given portal type, all portal openings with that type are resized automatically to accommodate the new frame model.

With this approach of associating detailed frame models with each portal type we sought to facilitate refinement of the model with components that were not generated through extrusion. The user need not position each model in 3D space. This is because, due to the semantic information extracted from the floor plan, the program knows where doors are, and can do proper positioning and sizing of the inserted door model automatically. The sections below describe how the portal model insertion system works.

9.3.1 Portal Types and Prototype Models

In either 2D floor plan or 3D model view, the user may select a portal and assign it a *portal type*. This type then dictates the size of that portal's opening in the 3D model. Each type may optionally have a portal *prototype model* associated with it. The user may change the parameters associated with each portal type definition. Changes made to the portal type

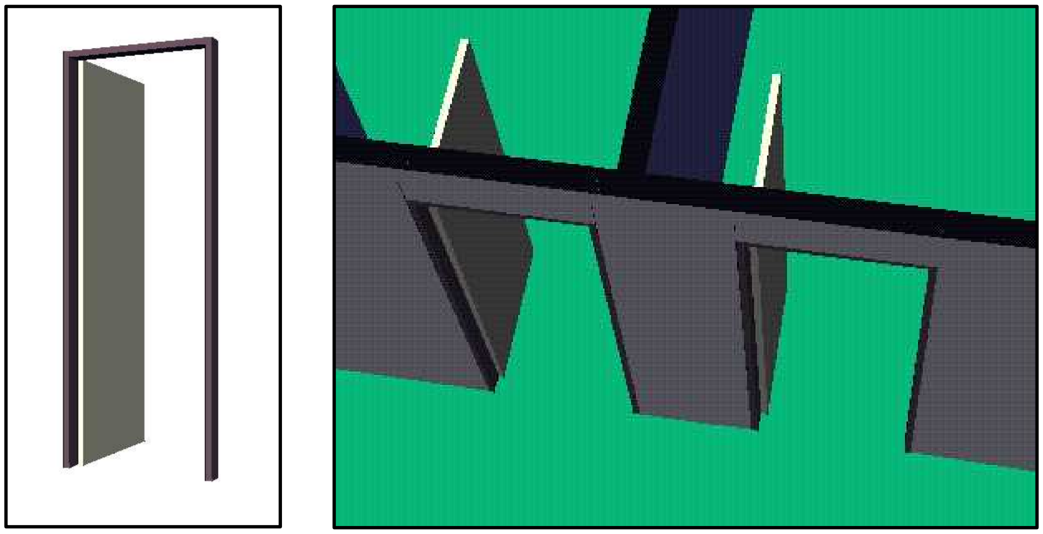


Figure 36: Portal prototype models and automatically generated instances.

definition are then applied to each portal of that type. For example, the user may define the portal type "OFFICE-DOOR", and specify that the portal prototype model for "OFFICE-DOOR" is described in the file "OFFICE-DOOR.ug". When a portal type is displayed for editing, BMG displays the prototype model in a window on the screen so the user may verify that it is the desired model.

When changes to a type definition are applied, BMG loads the geometry of the prototype from the file, determines the height of the opening required to accommodate that model, and sets the *opening height* field of the portal type definition to that value. The width and depth of each portal opening (i.e. the size in X and Y, respectively) is fixed by the floor plan; therefore, neither the width or depth is resized in response to an inserted model. It is the user's responsibility to create portal prototype models that fit each opening properly in width. Failure to insert models of proper width could result in coincident or intersecting faces, and could thus corrupt the integrity of the building model. However, the depth of an inserted model need not be equal to the depth of the portal opening, since it is acceptable for entities to protrude into the spaces on either side of the portal opening.

The user may then select various portals on the floor plan and give them the type "OFFICE-DOOR". When the 3D model is extruded, or if the type definition is changed after extrusion, the height of each portal opening of type "OFFICE-DOOR" in the extruded model is automatically adjusted to match the height needed to accommodate the prototype model, without any user interaction required.

9.3.2 Inserting Portal Prototype Models

Portal prototype models must be defined such that the origin of the portal model coordinate system (Figure 37a,b) corresponds to the appropriate corner vertex of the portal opening in the 3D model, into which the portal model will be inserted (Figure 38), and the x-axis of the prototype coordinate system points in the direction of the wall into which the prototype model is to be fitted. It is up to the user to ensure that the portal prototype model meets these conditions, since there is no way to algorithmically determine which vertex in the portal model is intended to correspond to the portal opening corner. The height of the opening required to accommodate the model is determined based on the Z extents of the model. BMG automatically resizes the affected portal in Z, to accommodate the inserted model.

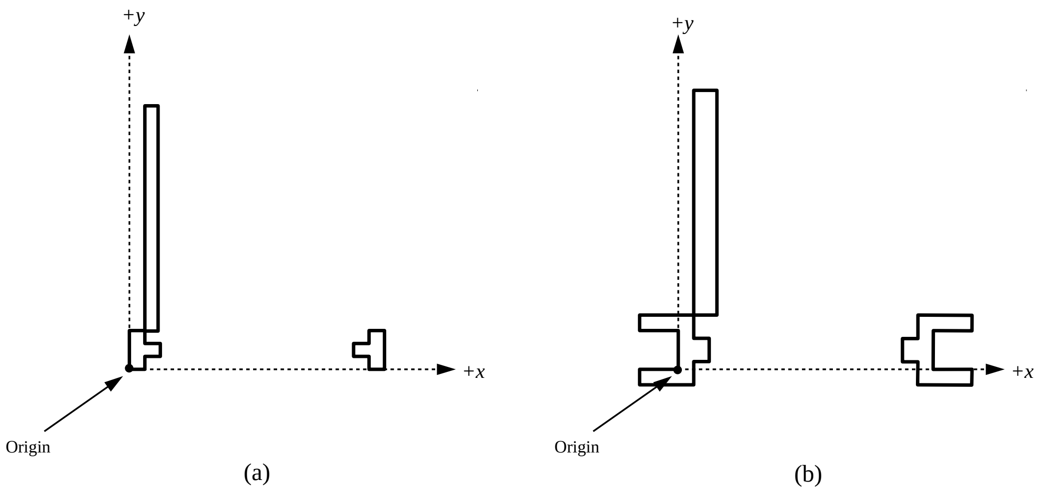


Figure 37: Portal prototype model, orthographic projection onto xy plane in local coordinate system.

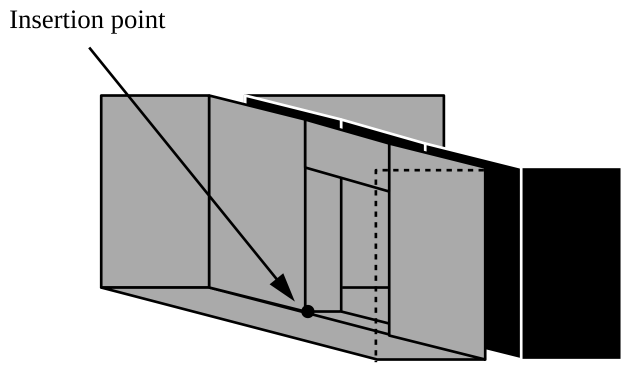


Figure 38: Portal opening, with insertion point for prototype model.

BMG determines the translation and rotation required for the prototype model to be inserted into each portal opening by analyzing the portal contour for each opening. The algorithms take advantage of the fact that portal contours are rectangular. The algorithm it outlined below:

```
For each portal contour (Figure 39),
    For each edge E from $P^0$ to $P^1$, with next edge N from $P^1$ to $P^2$,
        If the angle subtended by ray $P^1P^0$ and ray $P^1P^2$ (Figure 39a) is 90 degrees,
        and if the length of edge N is greater than the length edge E, and if edge N is
        adjacent to (shared by) a space, then
        1. $P^1_x$ is x translation
        2. $P^1_y$ is y translation
        3. Angle subtended by ray $P^1P^2$ and ray from $P^1$ in +x direction (Figure 39b)
           is rotation
```

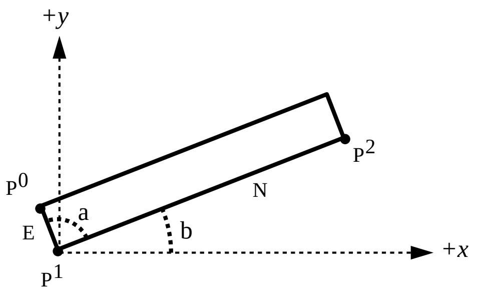


Figure 39: Determining translation and rotation for portal prototype model.

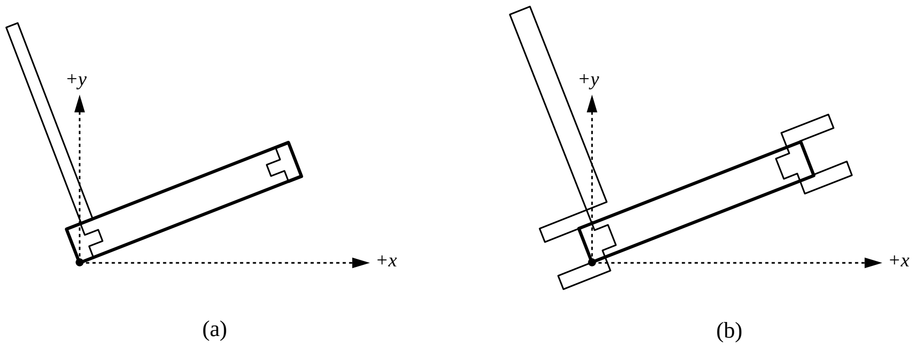


Figure 40: Orthographic view of portal prototype insertion.

Given the translation and rotation values, BMG creates an instance of the defined prototype model, translates it and rotates it, and inserts it into the portal opening (Figure 40a,b). This is done automatically for all portals with defined prototypes. If the user is unsatisfied with the inserted prototype, it can be selected and removed, and a new prototype can be inserted.

10 Modeling and Incorporating Staircases

A staircase consists of steps, plus step support structure, and railings. Since these properties of a staircase are well known, a 3D staircase model generator tool (StairMaster) [13] is provided, which allows the user to interactively specify parameters such as step height, incremental increase in height between adjacent steps, railing height, and others, and produces a polyhedral model of the staircase (Figure 41) that corresponds to the staircase plan (such as that shown in Figure 43) from the floor plan. This resulting model is then incorporated into the 3D model of the floor.

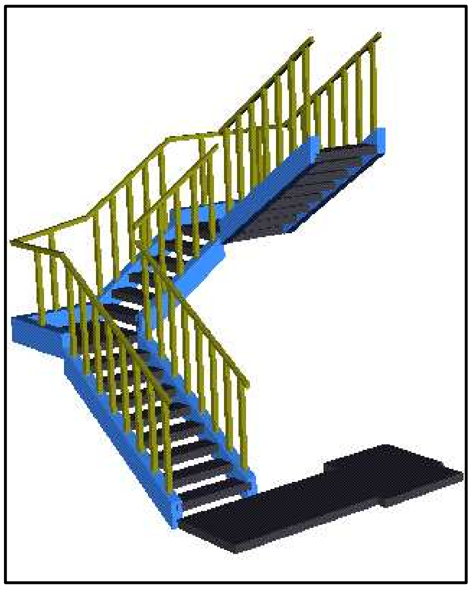


Figure 41: 3D model produced by staircase generator.

The staircase generator tool takes advantage of the fact that, much like a floor plan specifies the 3D geometry of the corresponding floor model, a 2D staircase plan specifies the geometry of the corresponding 3D staircase model. In both cases, all that is required from the user is height information; with staircases, this includes floor-to-floor height increment, step thickness, and railing height. User effort is drastically reduced by the staircase generator tool, since there is no drawing in 3D required to generate complex 3D staircase models.

10.1 Creating the Two-dimensional Staircase Plans

Architects usually draw staircases on floor plans as the 2D projection of that portion of the 3D staircase that is between two horizontal cut levels lying four feet above the respective floor and the floor below. The staircase plans may contain symbols that represent where the break at four feet occurs. Additionally, in the case of the Soda Hall plans (Figure 42), the geometry for steps is an incomplete collection of disjoint edges. As a result, the 2D representation of the staircase as it appears on the floor plan is not geometrically suitable for use as a basis for procedural generation of the 3D staircase model with StairMaster. This program requires every step of the stair to be drawn as a complete polygon. (Figure 43).

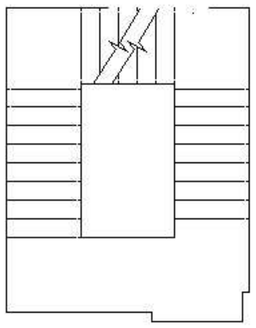


Figure 42: Original stair plan from Soda Hall floor 6.

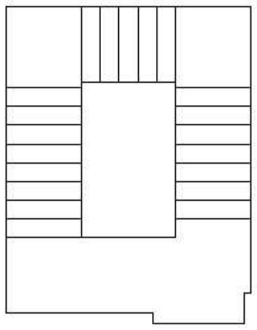


Figure 43: Completed stair plan.

In order to form a usable staircase plan, the user must complete the plan by drawing any step polygons not shown in the original symbolic version. For the Soda Hall plans, we simply drew the necessary polygons in AutoCAD, atop the incomplete symbolic plan. Then the staircase plan was exported in DXF format and converted to UniGrafix format.

10.2 Analyzing Staircase Plans

The staircase generator assumes that each polygon in the input file represents the outline on the XY plane of a step from the corresponding 3D model. Since there is no assumed or specified order to the steps, the staircase generator prompts the user to specify the lowest and second-lowest steps. This indicates with which step the staircase starts, and in which direction it ascends.

The staircase generator then orders the remaining steps, by searching at each step for another step polygon that is adjacent to that step polygon, where adjacency means there is at least one pair of edges, one edge from each step, that are coincident within some small epsilon perpendicular distance.

Determination of railing and support structure placement is made by following a set of rules as follows: if a step polygon edge is not shared by any other step polygon, a railing piece and support structure should be constructed at that edge. If an edge is shared, no railing or support structure should be created. If an edge is adjacent to, but not shared by, another step polygon edge, railing and support structure should be generated for the non-adjacent portion of the edge only.

Once the steps have been put into proper order, and railing and support structure placement determined, the 3D geometry for the staircase can be generated.

10.3 Creating the Three-dimensional Staircase Model

The staircase generator creates a 3D model of the staircase in UniGrafix format. Steps are right-rectangular extruded solids of a specific height (Figure 44a), placed at successively higher elevations as the staircase ascends. The height of each step solid is controlled by the *Step Height* slider, while the incremental elevation difference between steps (Figure 44b) is controlled by the *Step Separation* slider.

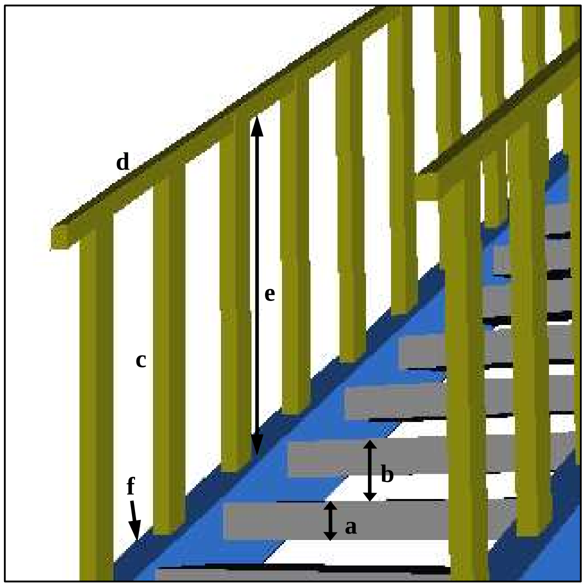


Figure 44: User-specified dimensions for staircase model.

Railings are composed of support posts (Figure 44c), which are right-rectangular extruded solids, and railing segments (Figure 44d), which are closed solids with rectangular cross section. The height of the support posts (Figure 44e) is controlled by the *Railing Height* slider, while the height or the rectangular cross-section of the railing segments is controlled by the *Railing Thickness* slider.

Runners, the support structure to hold the steps in place (Figure 44f), are closed solids with rectangular cross section. The height of the runners is controlled by the *Runner Height* slider.

All slider values may be entered precisely by the user in an input box when exact values are required. In addition, if the user specifies the separation between floors, the staircase generator automatically calculates the proper value for step separation.

The generated staircase models consist of hundreds of properly oriented polygons, none of which are drawn by the user in 3D space. As a result, this tool very significantly reduces modeling time for staircases. This is important because most buildings have multiple floor levels, and therefore have staircases. See [13] for a more detailed discussion of staircase model generation.

A completed staircase model is imported by the BMG system into the 3D floor model. All that is required for this task is a proper translation in Z to the appropriate elevation for that floor. Since the staircase plan is drawn in the same coordinate system as the whole floor, the X and Y values are correct.

11 Adding Complex Details to Building Model

Most buildings have some structural features that can not be modeled as simple extruded prismatic shapes. For example, the seventh floor of Soda Hall has several arches lining the exterior facade of the south side of the building (Figure 45). The basic arch element, which is repeated five times, is shown in figure 46. The arches are not specified geometrically on the floor plan, and can not be created through extrusion alone. So that such non-rectilinear structure may be included in the building model, the BMG system provides a mechanism for the user to import externally-modeled components into generated 3D floor models. The user interactively selects an *insertion point* in the 3D model, and then specifies the appropriate rotation by selecting an additional vertex in the model.

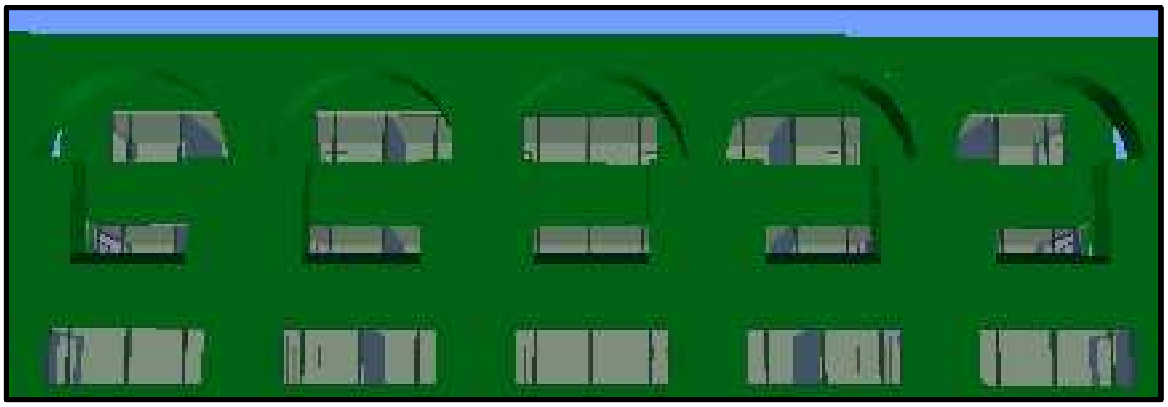


Figure 45: Arches on south side of Soda Hall floor 7.

BMG maintains a user-specified list of externally modeled components. As in the case of prototype portal models (section 9.3), these components must be modeled in a coordinate

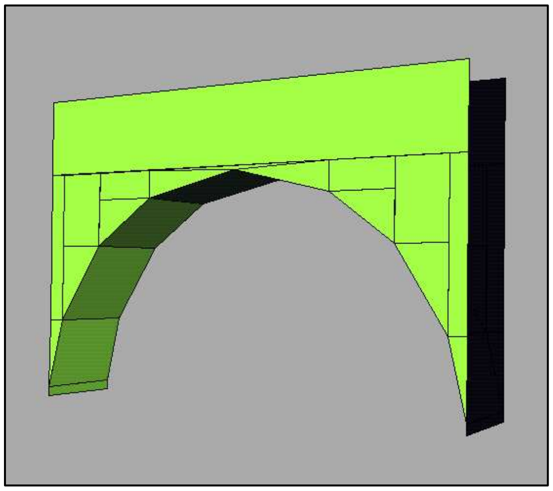


Figure 46: Soda Hall floor 7 arch component (edges displayed for clarity of construction).

system such that the insertion point in the floor model corresponds to the origin of the local coordinate system of the component. Given this constraint, the translation in X, Y, and Z is the same as the coordinate values of the insertion point selected by the user. The rotation is specified by a vector in the 3D floor model that corresponds to the positive Y axis in the component's local coordinate system. This user specifies this vector, which originates at the insertion point, and terminates at a second model vertex selected by the user.

This mechanism is important because it provides a necessary complement to the extrusion approach. While floor plan cleanup and intelligent extrusion suffices for the vast majority of building geometry (i.e., most walls and portals are rectangular), the mechanism for importing and properly positioning externally modeled geometry provides a very general way to include specific building details that can not be produced by extrusion.

12 Combining Finished Floor Models into a Building Model

After each individual floor has been modeled, the floors are stacked up to form the complete building model. This is done by combining the geometry for all modeled floors into a single UniGrafix file. Proper connectivity between vertically aligned exterior walls of adjacent floors is guaranteed, since the base elevation of the upper floor is equal to the top elevation of the lower floor, for all pairs of floors. However, holes may exist above or below floors whose exterior outlines are not aligned with adjacent floors above or below. For example, if the first floor of a two-story building is significantly larger than the second, stacking them up results in a hole where the second floor does not entirely cover the first (Figure 47a). Similarly, a hole will exist where the second floor has an overhang over the first floor (Figure 47b).

In order to produce the appropriate exterior roof or floor polygons to seal off these openings, BMG performs boolean subtraction on the exterior contours for adjacent floors. For each

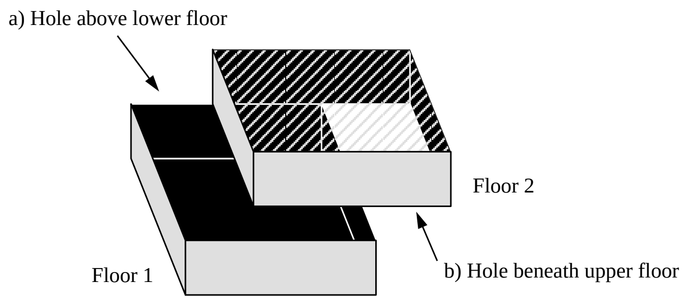


Figure 47: Holes in exterior surface when floors are stacked.

floor, there is a collection of 2D exterior contours, formed in the floor plan analysis phase. The necessary exterior polygons for a given floor are formed in the following manner: for roof polygons, the desired collection of 2D contours (Figure 48c) is obtained by subtracting the contours for the floor above (Figure 48b) from the contours for the current floor (Figure 48a). Then, upward-facing polygons with the resultant 2D contours are generated with height (Z values) equal to the top elevation of the current floor (Figure 48d).

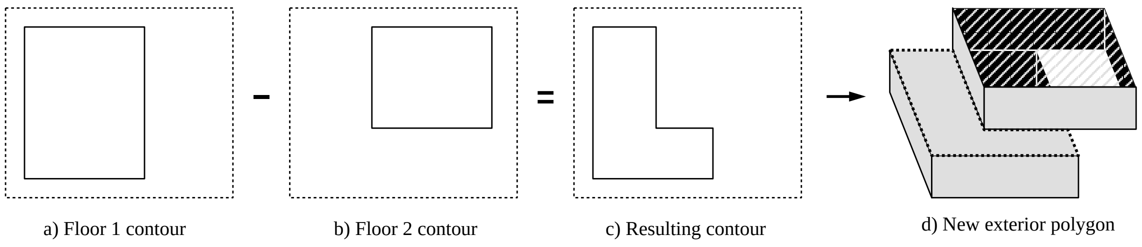


Figure 48: Determining polygons to close exterior hole above current floor.

For floor polygons, the desired collection of 2D contours (Figure 49c) is obtained by subtracting the contours for the floor below (Figure 49b) from the contours for the current floor (Figure 49a). Then, downward-facing polygons with the resultant 2D contours are generated with height equal to the base elevation for the current floor (Figure 49d).

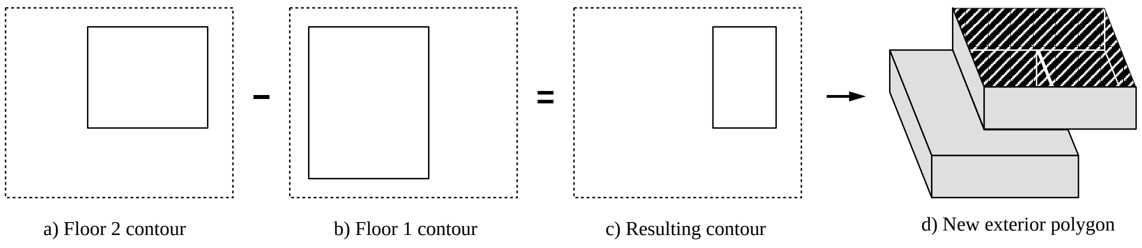


Figure 49: Determining polygons to close exterior hole below current floor.

The result after these operations is a consistent 3D building model with correct topology.

13 Implementation

The BMG system was built on the SGI IRIX platform, using C, C++, the Silicon Graphics GL graphics library [18], the Berkeley UniGrafix library, and the Forms user interface library [15]. The major code modules and the tasks the perform are:

1. Floor Plan Conversion
   (a) Convert to UniGrafix
   (b) Verify UG syntax
   (c) Break faces into wires
2. Floor Plan Correction
   (a) Coerce to grid
   (b) Close gaps
   (c) Correct intersections
3. Floor Plan Analysis
   (a) Find space contours
   (b) Find portal contours
   (c) Find room identity
   (d) Contour break-up
4. 3D Model Generation
   (a) Extrude spaces
   (b) Extrude portals
   (c) Generate floors and ceilings
5. 3D Model Editing
   (a) Space editing
   (b) Portal editing
   (c) Portal prototype insertion
   (d) Component insertion
   (e) Building integration

BMG uses the basic viewing mechanism of the *animator* program [9] previously developed in the WALKTHRU group at Berkeley. This viewing system displays UniGrafix geometry on SGI workstations, using calls to the GL library. It is used to display the floor plans at the various stages of correction and analysis, and three-dimensional building models at various stages of construction and refinement. The *Control* module is integrated with the viewer, and consists of a set of user interface forms and the routines that allow interaction between those forms and the BMG program.

The *Conversion* module accepts the DXF floor plan, converts it to UniGrafix, and filters out floor plan layers that are not needed for generation of the 3D model. The data structure produced by this module is a list of individual line segments, tagged with the appropriate layer name that specifies each segment's semantic meaning. The *Correction* module intelligently determines the grid spacing and snaps line segment vertices to the grid. Then it tests the line segments for improper intersection and corrects those errors. A non-directed graph of vertices is built, which is used to identify vertices that are referred to by only one edge, indicating a gap between edges. Such gaps are algorithmically corrected. Door and window symbols are replaced with extrudable line segments.

The *Analysis* module searches the vertex graph in order to find closed contours. Two lists of closed contours are constructed, one for spaces and one for portals. If a reflected ceiling plan is provided, affected space contours are divided up appropriately into multiple contours. Each pair of spaces that is connected by a portal is doubly linked, forming an adjacency graph of spaces. Then the *Model Generation* routines properly extrude the contours into a consistent 3D model. Each space in the list of spaces maintains a list of faces generated in extruding that space. In this way, a space can be easily deleted from the 3D model and regenerated with new parameters such as ceiling height or wall color.

The *Editing* module implements editing capabilities that are used at different stages of the building generation process. "Contour Verification" routines allow the user to specify how a non-closed contour should be completed. The "Space Editing" and "Portal Editing" routines allow the user to interactively modify the heights and other properties of spaces in portals in both 2D and 3D mode. When modifications are made in 3D model-view mode, the space or portal is regenerated in accordance with the new properties. The "Component Insertion" routines handle the translation, rotation, and instancing of pre-modeled components that the user inserts into the model. A list of inserted components is maintained. Each component in the list contains a list of the faces created for that component instance, so that the component can be easily removed from the model if the user is dissatisfied. Finally, the "Building Integration" routines combine multiple floor models together, suitably closing off exterior holes by comparing the exterior contours for each floor. This module also contains output routines to save the UniGrafix geometry of the model, and the cell and portal information for CFAST.

14 Building Models Generated with BMG

BMG has been used to generate building models for Soda Hall, the new Computer Science building at Berkeley, and the "Hex Tower", a tricky test floor plan by Carlo Séquin.

Figure 50 shows the completed Soda Hall building generated by BMG, from the ground floor (floor 3) up. Figure 51 shows the fifth floor of Soda Hall, with components, portal frame models,

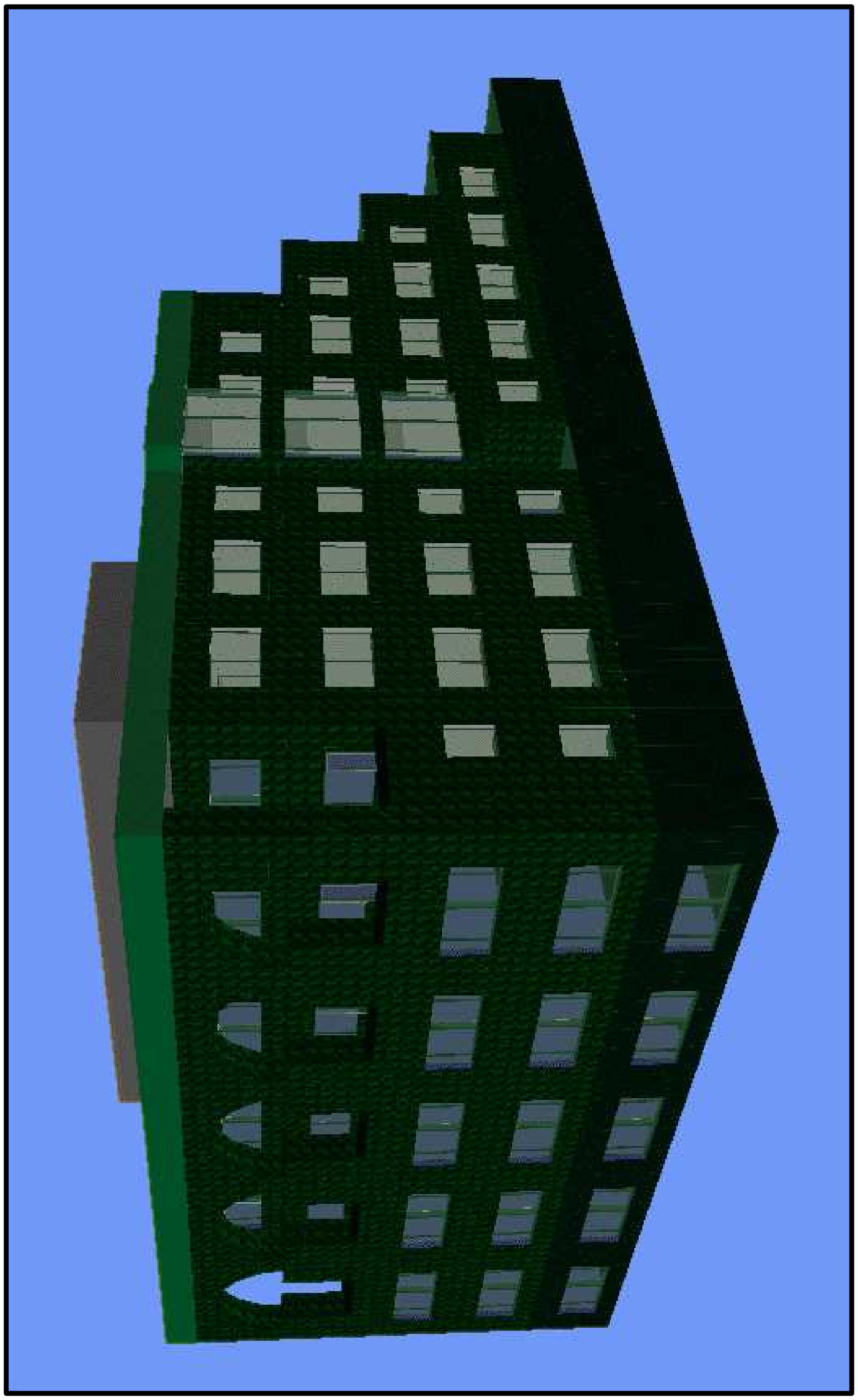


Figure 50: Completed Soda Hall model, ground level and above.

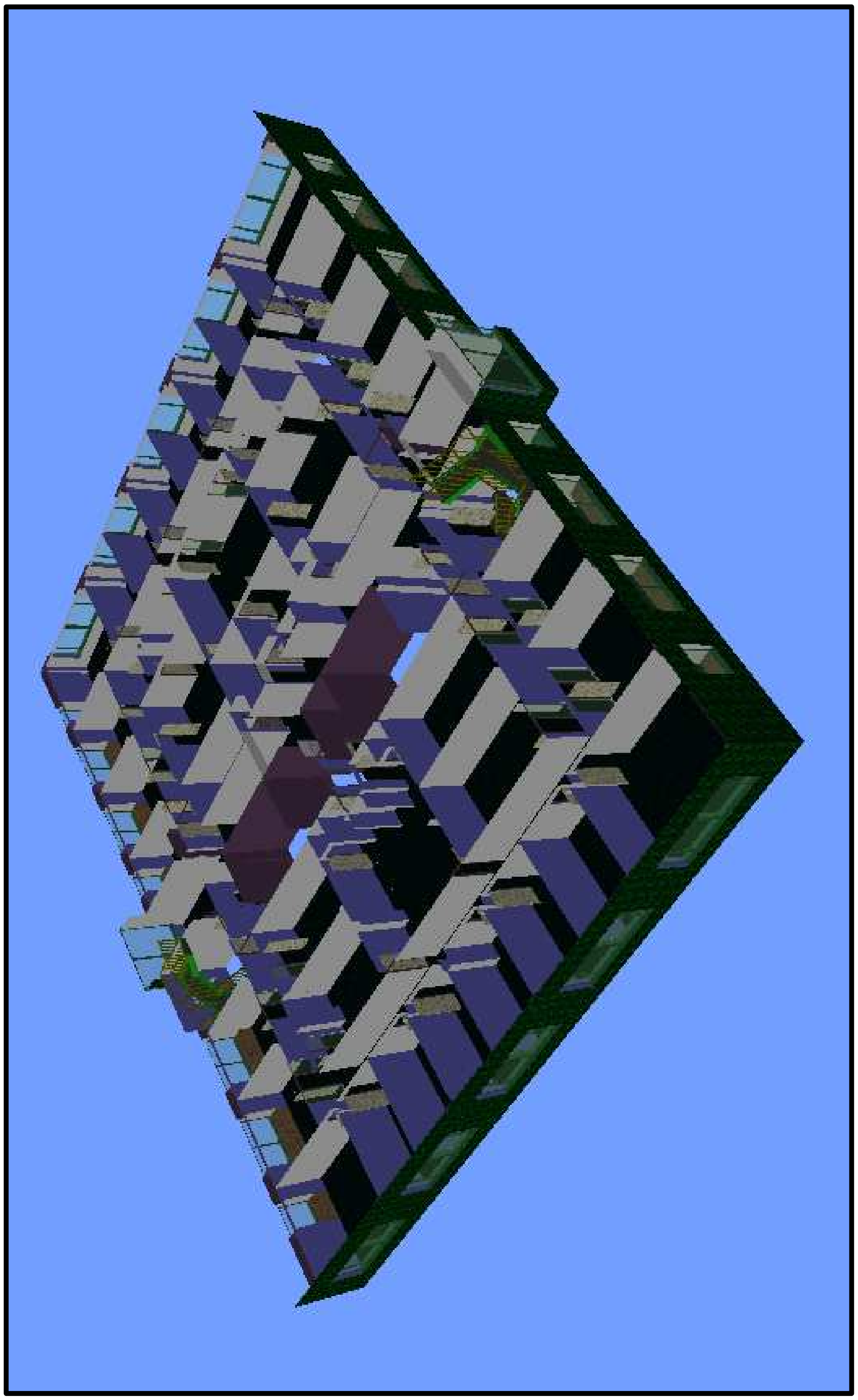


Figure 51: Extruded model of Soda Hall, floor 5.

and stairs. Refer to Figure 4 to see the original floor plan for this floor.

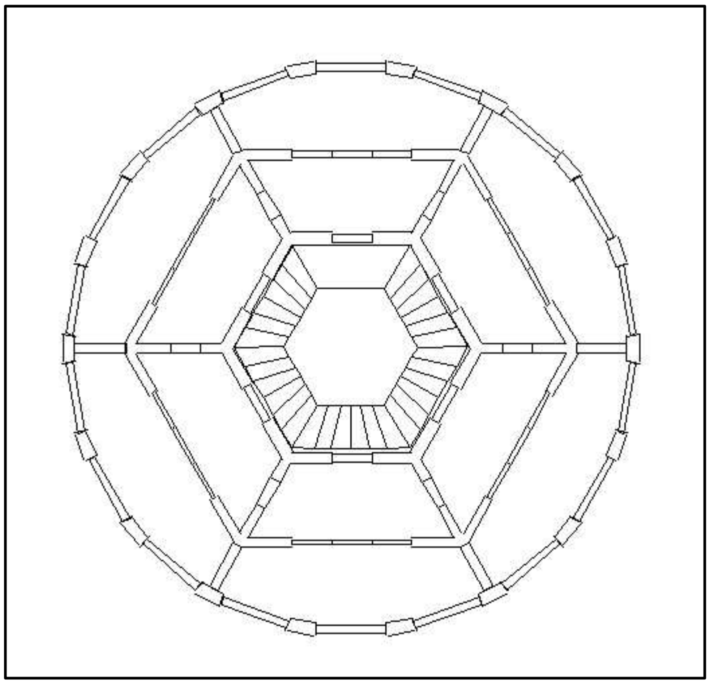


Figure 52: Floor plan for “Hex Tower” building.

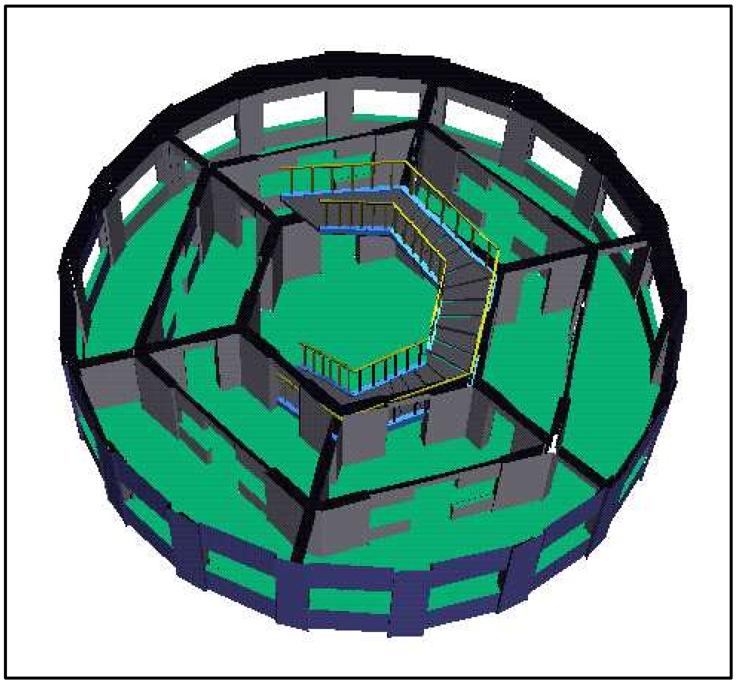


Figure 53: Generated single-floor 3D model for “Hex Tower” building.

Figure 52 shows the floor plan for the “Hex Tower”. The floor plan was drawn with the appro-

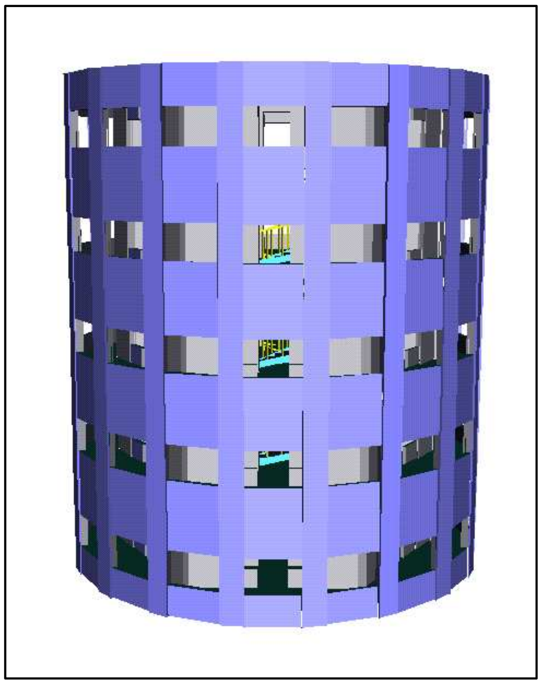


Figure 54: The "Hex Tower" building generated by BMG.

priate layers using the *Canvas* program for the Macintosh. This plan is an excellent illustration of the need for the geometric cleanup features of the BMG system. Minor modifications to the BMG system were required in order to generate the 3D model from the "Hex Tower" plan. First, door symbols were drawn will two jamb edges plus two *sill* edges; the sill edges correspond to the two edges BMG normally creates when processing door symbols (Section 7.1). Also, most edges on this plan do not exhibit proper connectivity. This is because only a small part of the floor plan was drawn explicitly; the rest was generated by using the imprecise copy and rotate features in Canvas. Figure 55a shows a portion of the "Hex Tower" plan. The two door jamb edges (Figure 55a-1) are poorly aligned with the wall to which they should be adjacent (Figure 55a-2). An additional cleanup step was added to BMG in response to this problem. A jamb-snapping algorithm was applied, which locates the nearest wall edge to each jamb edge, and properly translates and rotates the jamb edge so that it aligns properly with the wall. The translation takes place perpendicular to the wall edge, and the jamb edge is then rotated so that it is collinear with the wall edge. The plan as it is corrected by BMG is shown in Figure 55b.

Figure 53 shows a single generated floor of the "Hex Tower" building, and Figure 54 shows the entire building as generated during an interactive session with BMG in about 30 minutes.

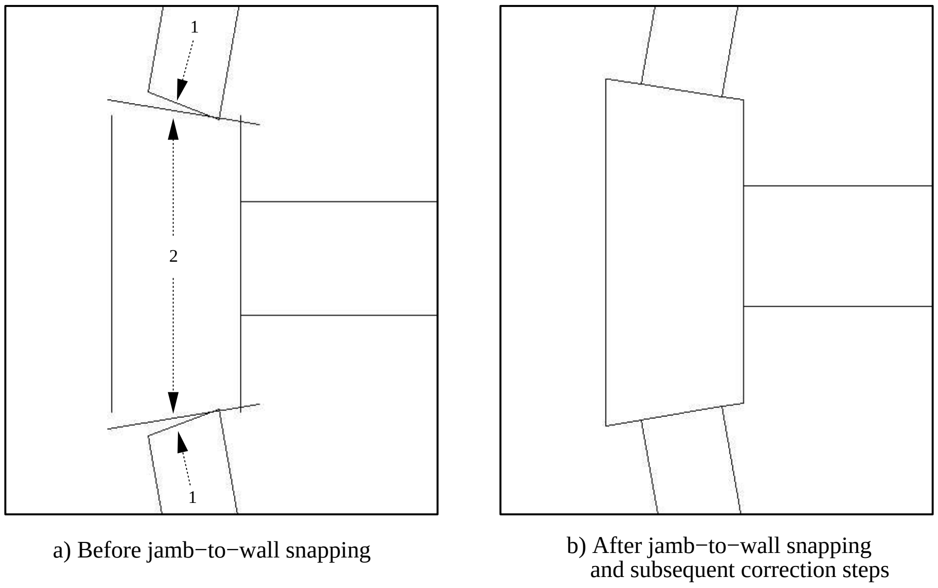

1
2
1
a) Before jamb−to−wall snapping
b) After jamb−to−wall snapping and subsequent correction steps

Figure 55: Portion of "Hex Tower" plan before and after jamb-to-wall snapping.

15 Using Generated Building Models with WALKTHRU

Completed building models are fully compatible with the Berkeley WALKTHRU visualization system [11]. The WALKTHRU system performs visibility preprocessing [19] and level-of-detail management [10] to reduce rendering time and increase frame rate, and provides an intuitive interface for user navigation through the virtual environment. Using the WALKTHRU system, it is possible for the user to obtain an impression of the building from the building occupant's perspective.

To use a building model with the WALKTHRU, the model must meet the criterion of containing only axis-aligned, convex occluding polygons; therefore, BMG produces models that meet this condition. Then, WALKTHRU utilities process the finished model and create a Berkeley WALKTHRU-compatible database, which can then be used with the interactive visualization system.

15.1 Breaking-up Concave Spaces to Form Rectangular Cells

The Berkeley WALKTHRU system utilizes the concept of visibility *cells* to speed rendering. In these systems, each cell is a volume bounded by occluders (opaque polygons), with *portals* that allow vision into the adjacent cell and possibly beyond. The computations performed in the Berkeley WALKTHRU system require that these cells consist of rectilinear, axis-aligned, and convex cells.

Since it was one of the goals for the BMG system to produce 3D models for the Berkeley

WALKTHRU system, BMG optionally breaks spaces into rectangular cells, as required by the visibility algorithms. Figure 56b shows the result of applying the breakup algorithm to the original set of space contours shown in Figure 56a.

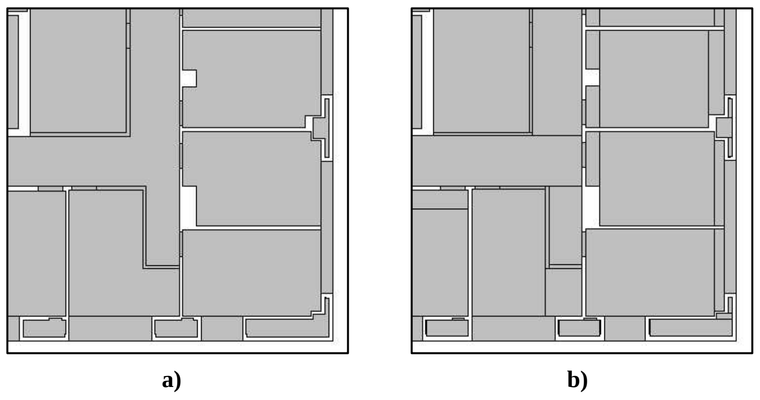


Figure 56: Spaces before and after breakup into convex components.

15.2 Creating the WALKTHRU Database

A suite of utilities is used to convert the 3D UniGrafix model of the building into the database format used by the WALKTHRU visualization system. This can be run as a batch process, and takes on the order of a few minutes.

In order to obtain the most realistic building model possible, the user may wish to populate the final model with objects such as furniture, that are not considered part of the building itself. For example, we have models for desks, chairs, lamps, and plants for Soda Hall. Including these objects in the model makes the building environment more realistic and interesting. These objects can either be added as part of the batch process that creates the database, or using the interactive WALKEDIT editor program [4].

15.3 Using the WALKTHRU Viewing System

The user may interactively navigate the completed building database. Figure 57 shows a view of the 6th and 7th floors of Soda Hall, created with the BMG system, from within the WALKTHRU environment.

In the WALKTHRU environment, the mouse buttons are used to specify forward or backward motion, and the lateral position of the mouse cursor controls the turning rate. Gravity is simulated; for example, the user can walk up or down stairs simply by moving forward. Interior objects can be picked up and moved around. Behaviors are associated with each object, such that each object can be manipulated with only two degrees of freedom; for example, a desk always rests on the floor, so movement is by default along the floor plane. Refer to [11] and [4] for more detailed discussions of the WALKTHRU and WALKEDIT environments.

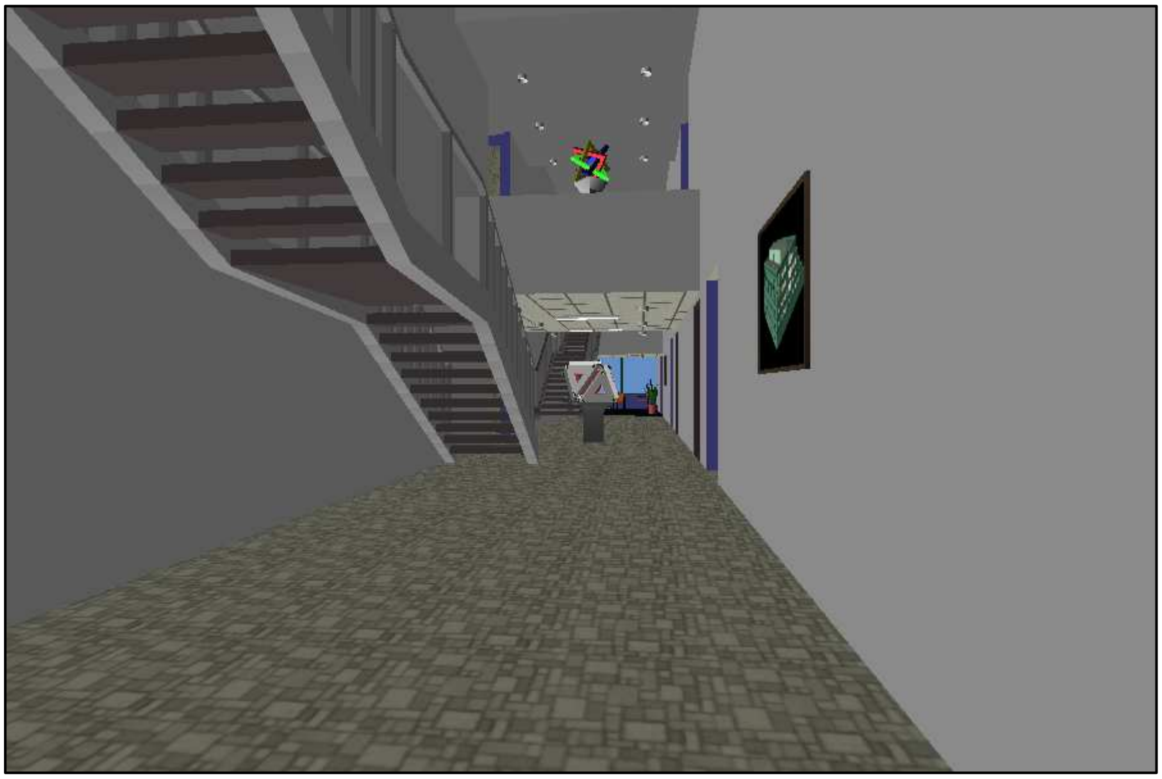


Figure 57: Soda Hall 6th floor corridor, with 7th floor above.

16 Using Generated Buildings with the NIST CFAST Fire Simulator

The NIST CFAST fire simulator [16] simulates the spread of fire through virtual environments. Building models generated with BMG can be exported in a format appropriate for use with CFAST. CFAST requires that the building environment be described as a collection of right-rectangular volumes, or "cells", and portals that connect those cells. The dimensions and locations of volumes and portals can be obtained in straightforward fashion from the data structures developed by BMG during the floor plan analysis phase. In particular, spaces on the floor plan are linked to adjacent portals, and portals are linked to adjacent spaces. The bounding volume of each space is determined by forming a bounding box around the 2D contour for each space, and extruding it to the height of that space in the 3D model. Portals are two-dimensional connections between volumes; the depth of the portal is not important in the current implementation of CFAST.

The current implementation of CFAST requires that volumes are rectilinear. This necessitates either breaking up or simplifying non-rectangular space contours. Currently, spaces are partitioned into rectangular pieces (Figure 58a); at each edge inserted to break up a space, a CFAST portal is created, which connects the two resulting CFAST volumes. This can result in an undesirably high number of volumes if there are "nooks" or protrusions in room walls; in Figure 58a, the two original spaces result in nine CFAST volumes. In the future, we intend to implement a simplification algorithm that will form bounding boxes around spaces (Figure 58b), partition spaces only if their bounding boxes overlap (Figure 58c), and expand resulting spaces (Figure 58d) such that they are adjacent along a zero-width boundary along which CFAST portals may exist.

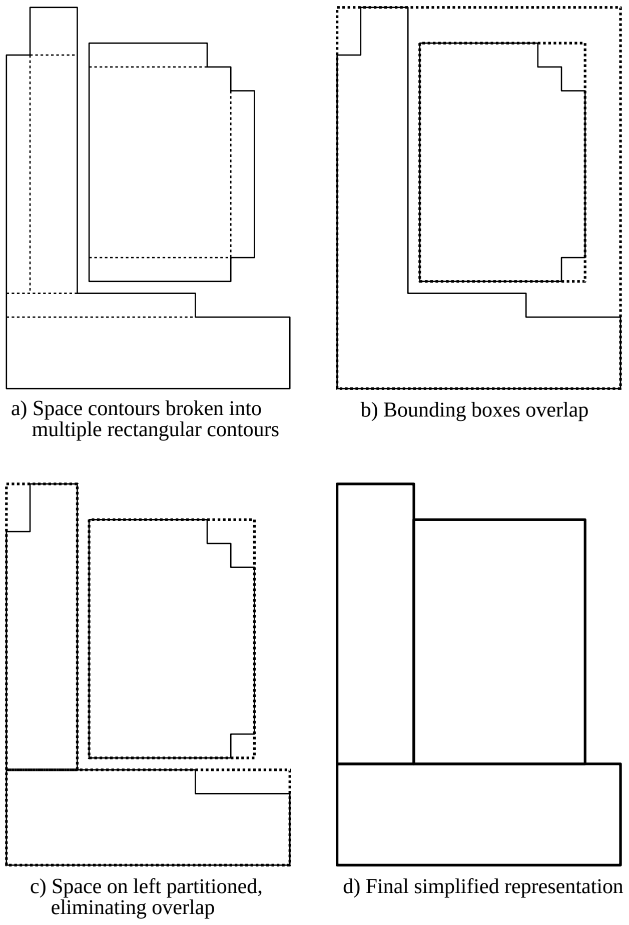

a) Space contours broken into multiple rectangular contours
b) Bounding boxes overlap
c) Space on left partitioned, eliminating overlap
d) Final simplified representation

Figure 58: Creating rectangular space contours required for rectilinear CFAST volumes.

After rectilinearization, the resulting input to CFAST is a list of cells and portals. Each portal definition contains the two-dimensional measurements of the portal opening (height and width). Each cell definition contains that cell's bounding volume dimensions, and pointers to the portals that connect that cell to its neighbor cells.

The CFAST system takes materials properties into account when performing the fire simulation. Each color or texture in the 3D model can be mapped to a specific material for CFAST. The CFAST system will be used to perform various simulations of the spread of fire throughout the Soda Hall building and other buildings modeled with the BMG system. The CFAST system is also currently being adapted to interactively display the progress of the fire throughout the virtual environment, using the WALKTHRU system as a display platform [5].

17 Recommendations for CAD Tool Builders and Users

A significant portion of the BMG system is devoted to cleaning up the geometry of input floor plans, and developing the semantic information necessary for extrusion and editing. Clearly, tools that assist the architect in producing clean, semantically complete floor plans would help to eliminate some of the required floor plan processing and correction.

There are some simple, common-sense guidelines that users of existing drafting tools would be well advised to follow when creating detailed floor plans. First, rooms should be drawn as a sequence of edges that share endpoints, rather than as a collection of independently-placed line segments. This eliminates the possibility of disjoint or improperly overlapping edges. It is also of benefit to the draftsperson who draws the floor plan; clearly, it is easier and more efficient to click and create the n vertices of a room contour, rather than creating and independently positioning n line segments. Second, symbols for doors and windows should be instances of pre-defined primitives, with all the necessary types of edges that comprise each symbol classified into the appropriate layers. If this rule is not followed, there is a need for a specific preprocessing step for each set of floor plans in order to classify each symbol edge into the appropriate group, and there is no guarantee that edges will be classified correctly when the operation is performed automatically. Finally, room labels should be provided, and positioned somewhere near the center of each space. Aside from providing some identity information for each space, these labels also facilitate the location of a closed contour for each space by providing the system with the knowledge of where the inside of the room is. With this knowledge, the system can look for the closest wall edge and perform a search to form a closed contour containing that wall edge.

A set of tools that would improve the geometric correctness and semantic completeness of detailed floor plans could start with an editor that assists the user in floor plan layout at the symbolic level (Figure 59a). Each room would be represented by a single closed contour, and rooms would not be allowed to overlap. Each room would have a specific identity, such that the geometry of the contour representing a particular room can be obtained by dereferencing that identity. Door and window symbols would be predefined primitives that can only be inserted along the boundary edge between two space contours. This simple representation would allow the user to quickly arrange the layout of the floor, calculate space usage, and examine connectivity between various rooms through their respective portals. Since the level of detail would be low, editing operations are simple and inexpensive.

The design could then move forward towards a more detailed representation, with thick (i.e. double-line) walls automatically produced by the tool (Figure 59b). Rooms should still be movable, subject to the same non-overlapping constraint plus an additional constraint that maintains some minimum wall thickness between rooms (Figure 59c). It should be possible to insert single wall pieces into existing rooms, dividing them (Figure 59d), with automatic fleshing-out into a thick separator wall by the tool (Figure 59e). Portal symbols would be fleshed out into contours representing the 2D projection of the physical connections between the spaces (Figure 59f). From this geometric representation of the floor plan, generation of a 3D model would be straightforward, because the exact geometry and semantic meaning of rooms and portals is known explicitly, and correct geometry is ensured because correctness is enforced throughout the process of creation of the plan. This representation could then be annotated with text and additional symbols, to create detailed construction documents.

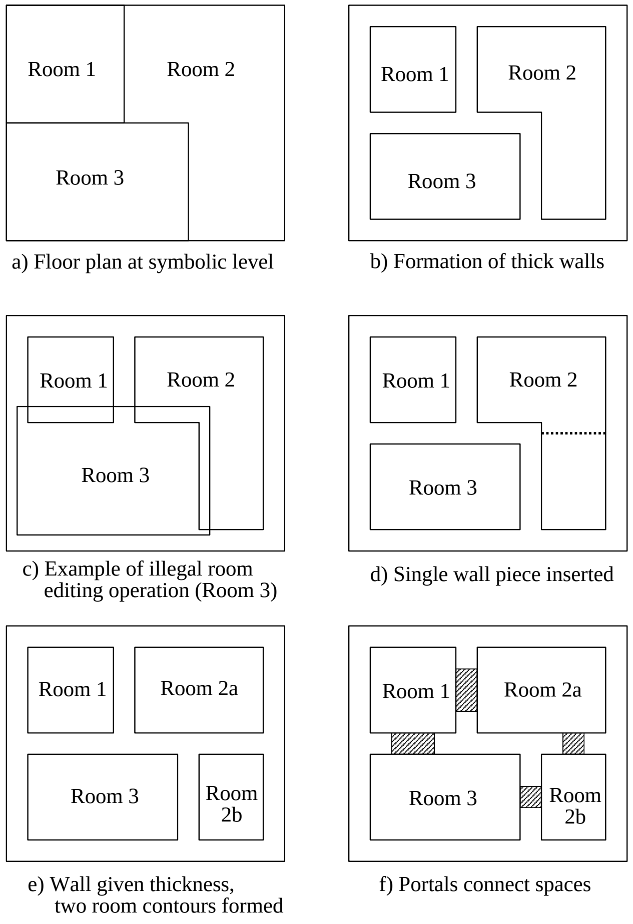

Room 1
Room 2
Room 3
a) Floor plan at symbolic level
Room 1
Room 2
Room 3
b) Formation of thick walls
Room 1
Room 2
Room 3
c) Example of illegal room editing operation (Room 3)
Room 1
Room 2
Room 3
d) Single wall piece inserted
Room 1
Room 2a
Room 3
Room 2b
e) Wall given thickness, two room contours formed
Room 1
Room 2a
Room 3
Room 2b
f) Portals connect spaces

Figure 59: Symbolic and simple geometric floor plan representations.

The advantage of this type of floor plan design system is that the user is prevented from creating entities with erroneous connectivity or unknown semantic meaning. Resulting floor plans can only consist of rooms and the portals that connect them, and the semantic meaning of each entity is precisely known. With clean geometry and sufficient semantic information, generation of 3D models from such plans would be much easier.

18 Future Work

We are pursuing several extensions to the BMG system, to couple it more closely with schematic layouts from earlier phases of building design, and to more seamlessly integrate BMG into the WALKTHRU. In addition, we are developing tools that are intended to improve the process and results of computer-assisted architectural design.

18.1 Extensions to BMG System

BMG will be augmented to allow arbitrarily sloping floors and ceilings, by maintaining height information at each space contour vertex, rather than at the level of an entire space. We also intend to provide the capability to go from symbolic schematic floor layout diagrams to real architectural plans automatically, therefore providing a smooth path from simple 2D layout sketches to 3D extruded models. This will be accomplished by enhancing the floor plan generation capabilities of the symbolic floor plan extrusion tool described in section 2.5, and integrating those capabilities with the BMG system. The result will be a complete extrusion system that can accept either schematic layouts or real architectural plans.

We plan to merge the BMG system with the WALKTHRU viewing system, thereby unifying our suite of building generation, viewing, and interactive editing tools into one program [5]. When this has been completed, the user will be able to load a floor plan, generate the 3D model, add furniture, navigate through the model, and interactively manipulate its contents, all from within the same executable program.

18.2 Additional Architectural Design Tools

The processes of building design, formal specification of completed designs, and implementation of designs (i.e. construction) are the three basic practices of the so-called "AEC" (Architecture, Engineering, and Construction) community. Currently, software to assist in these processes are is only mature at the level of formal design specification, or drafting, where detailed site plans and floor plans are drawn.

The design process itself is currently performed by architects without much help from computers. The result is that the design process is unnecessarily iterative. For example, the architect may decide on a building form, and sketch an entire floor layout, only to find after completion that the amount of assignable square footage on that floor is off by 50%. This is possible because the tools, whether paper and pencil or a simple polygon editor, do not assist the user in conforming to the specified building program. A floor layout tool that enforces requirements and constraints dictated by the building program would enable the architect to get the layout right with fewer iterations, and would free the architect to concentrate on the more creative and subjective aesthetic considerations when evaluating the design.

In the ideal design environment, the user should be able to move freely between different representations of the building design, such as initial functional bubble diagram, symbolic layout, detailed floor plan, and 3D building model. At each level of representation, it should be possible for automated agents to verify that the design conforms to the requirements of the building program. A mature suite of tools that assist in building design by performing this sort of automatic verification would increase productivity, and more importantly would result in buildings that actually implement the specifications put forth by the client.

19 Conclusion

A system that generates consistent 3D polyhedral building models from 2D architectural floor plans has been described. The system accepts floor plans in DXF format as input. The input floor plans are required to have certain primitives, such as wall edges and door symbols, grouped into properly labeled layers. Geometric correctness is not assumed. The input floor plans are corrected algorithmically by fixing gaps and improper intersections, a task which is tractable in two dimensions without requiring much user interaction. Door and window symbols are located and replaced with extrudable edges, and edges are grouped into a single closed contour for each space on the floor plan.

Space-to-space adjacencies are determined and an adjacency graph is built that is useful both in construction of the 3D model and as supplemental input to the CFAST fire simulator. The clean, analyzed floor plan is then extruded into a consistent 3D model, with properly oriented polygons, correct topology, and suitable holes for doors and windows. Heights, colors and textures can be edited at various levels of grouping. Convenient facilities are provided for inserting detailed door and window assemblies, with proper position and orientation determined automatically. Additional detailed components also can be inserted easily into the extruded model.

With the BMG system, it is easy to create a consistent polyhedral building model in a short period of time. When starting from reasonably structured floor plans, generated models correspond precisely to the specified building design. The user need not do any drawing in three dimensional space; therefore, most of the human effort and possibility of human error is eliminated. Since edge connectivity is corrected at the 2D floor plan level, the extrusion process yields a polyhedral model with correct topology. User interaction is minimal, and in general is only required when the user wishes to add detailed, externally-modeled components to the building model. Total conversion, cleanup and modeling time is on the order of hours per floor, rather than weeks or months, permitting more convenient creation and use of such models. Models generated with the BMG system can be viewed with the Berkeley WALKTHRU program, and are compatible with the NIST CFAST fire simulator.

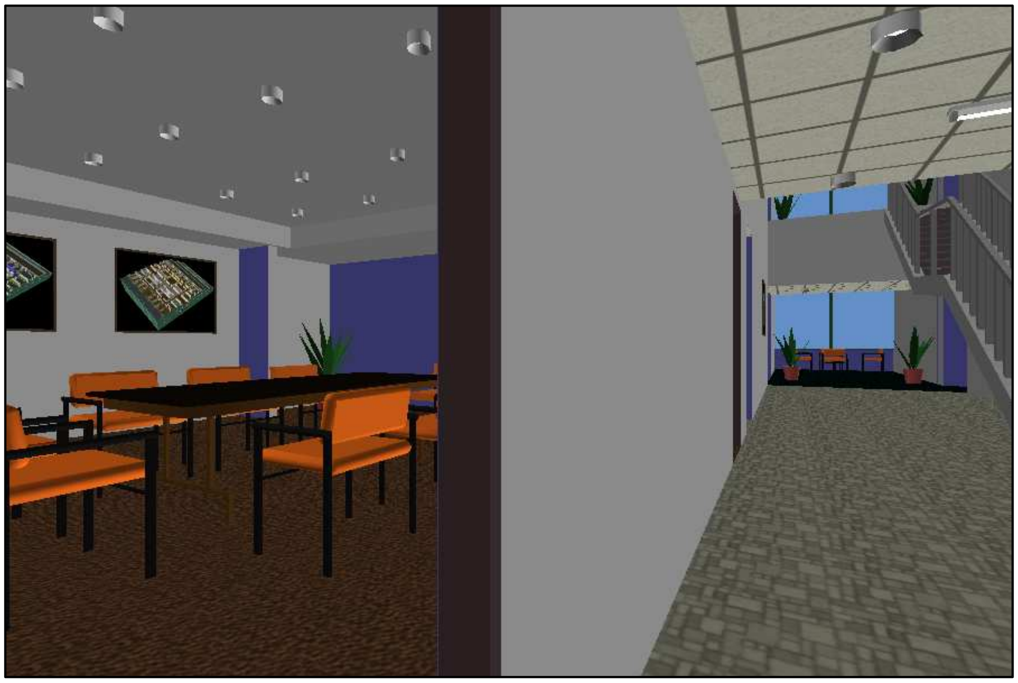


Figure 60: Interior view of Soda Hall model, sixth floor conference room and central corridor area.

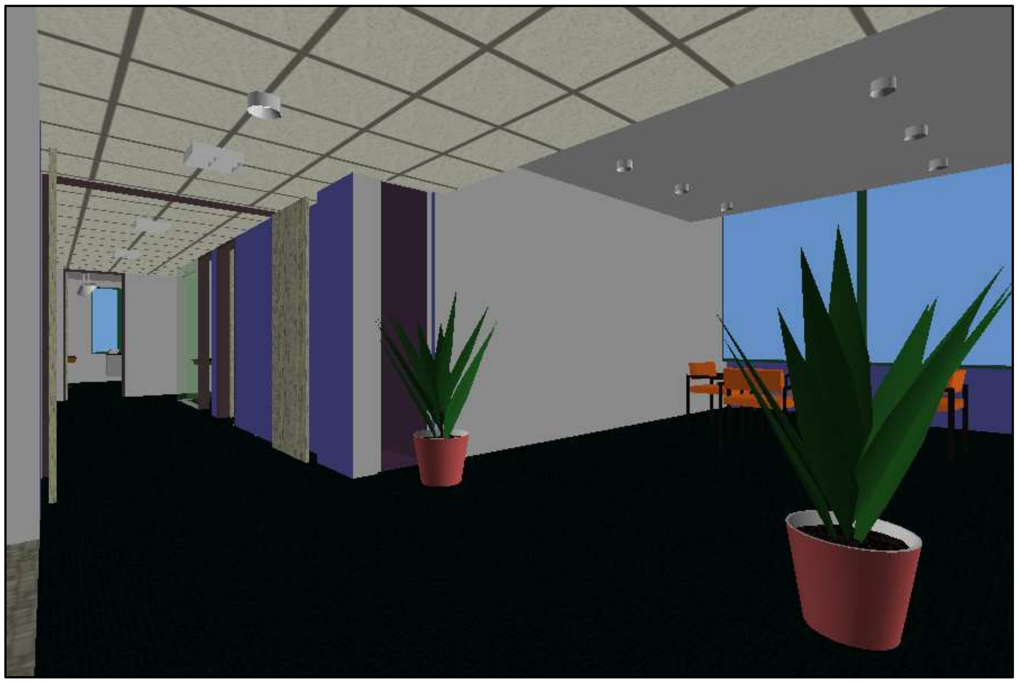


Figure 61: Corridor and discussion area, Soda Hall sixth floor.

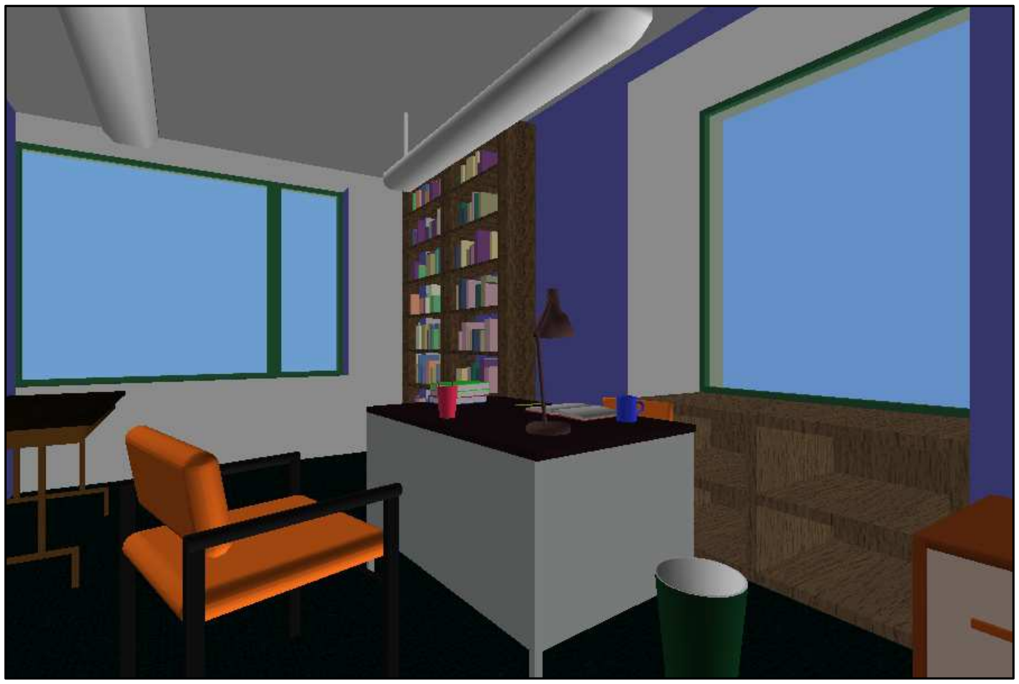


Figure 62: Office, Soda Hall sixth floor. Window model includes textured wood shelves below.

Acknowledgements

I would like to thank my advisor, Professor Carlo Séquin, for his thoughtful guidance and his great enthusiasm for this project. It has truly been a pleasure to be one of his students. I would also like to acknowledge the generosity of NEC Corporation and the National Institute of Standards and Technology; their support enabled me to concentrate all of my energies on this exciting research, and was instrumental in the completion of my Master's project. My sincere thanks to them for the help they have provided.

I am grateful for the friendship and assitance of my fellow graduate students and officemates: Rick Bukowski, Laura Downs, Randy Keller, Maryann Simmons, and Ajay Sreekanth. Many thanks to Rick, whose donation of Boolean polygon subtraction code was extremely helpful. Thanks to Maryann and Randy for being faithful late-night work companions in 537 Soda Hall (our "home away from home"), and for providing technical help and constructive criticism. Thanks to Terry Lessard-Smith and Bob Miller for all of their help in keeping everything running smoothly. Also, thanks to Don Sadler and Tony Rinella at Anshen and Allen for providing the Soda Hall floor plans.

Finally, thanks to my family and friends for being so patient and supportive (especially Jenn, Scott, and Pamela). Above all, I would like to say a special thank you to my parents, Dick and Karin Lewis. I credit them with enabling my success, personal fulfillment, and happiness. In my humble opinion, they are the greatest Mom and Dad on earth. Every day, I thank my lucky stars that they are my parents.

References

[1] Airey, John M., John H. Rohlf, and Frederick P. Brooks, Jr. "Towards image realism with interactive update rates in complex virtual building environments." *ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics*, 24, 2 (1990), 41-50.

[2] Autodesk, Inc. AutoCAD Release 12, Advanced Tools, 1994, 267-310.

[3] Autodesk, Inc. AutoCAD Release 12, User's Guide, 1994.

[4] Bukowski, Richard W. *The WALKTHRU Editor: Towards Realistic and Effective Interaction with Virtual Building Environments.* Master's Thesis, Computer Science Division (EECS), University of California, Berkeley, 1995.

[5] Bukowski, Richard W. Work in progress, 1996.

[6] Cosmi Software. 3D Virtual Reality Roomplanner, User's Guide, 1994.

[7] Couch, Gregory S. *Berkeley UNIGRAFIX 3.1 - Data Structure and Language.* Technical Report UCB/CSD-94-830, Computer Science Division (EECS), University of California, Berkeley, 1992.

[8] Expert Software. Home Design 3D, User's Guide, 1995.

[9] Funkhouser, Thomas A. "An Interactive UNIGRAFIX Editor," *Interactive Procedural Model Generation.* Technical Report UCB/CSD-91-637, Computer Science Division (EECS), University of California, Berkeley, 1991, 17-34.

[10] Funkhouser, Thomas A. *Database and Display Algorithms for Interactive Visualization of Architectural Models.* Ph.D. Thesis, Computer Science Division (EECS), University of California, Berkeley, 1993.

[11] T. A. Funkhouser, S. J. Teller, C. H. Sequin, and D. Khorramabadi, "UCB System for Interactive Visualization of Large Architectural Models," Presence, Spring 1996.

[12] Khorramabadi, Delnaz. *A Walk Through the Planned CS Building.* Master's Thesis, Computer Science Division (EECS), University of California, Berkeley, 1991.

[13] Lewis, Richard W. "StairMaster: An Interactive Staircase Designer," *Procedural Modeling.* Technical Report UCB/CSD-94-860, Computer Science Division (EECS), University of California, Berkeley, 1994, 27-36.

[14] Lewis, Richard W. "FBMG Floorplan and Building Model Generator," URL *http://www.cs.berkeley.edu/~rickl/fgen.html*, 1995.

[15] Overmars, Mark H. *Forms Library: A Graphical User Interface Toolkit for Silicon Graphics Workstations.* Department of Computer Science, Utrecht University, 1993.

[16] Richard D. Peacock, Glenn P. Forney, Paul Reneke, Rebecca Portier, and Walter W. Jones, "CFAST, the Consolidated Model of Fire Growth and Smoke Transport," NIST Technical Note 1299, National Institute of Standards and Technology, Gaithersburg, Maryland, 1993.

[17] Rinella, Tony. Personal communication, July, 1995.

[18] Silicon Graphics, Inc. Graphics Library Programming Guide, 1991.

[19] Teller, Seth J. *Visibility Computations in Densely Occluded Polyhedral Environments.* Ph.D. Thesis, Computer Science Division (EECS), University of California, Berkeley, 1992.