Procedural Generation of a 3D model of MIT campus

Research project report

By Andreas F. Wehowsky

Graphics Lab at MIT

May 2001

Contents

Introduction

Overview

This report is to document my work in a research project in the spring of 2001 at the Computer Graphics Lab at MIT with the title *Procedural Generation of a 3D Model of MIT Campus* and Seth Teller as advisor. I worked in a group together with Todd Drury and Keyuan Xu. The goal of this project is to find a way to procedurally generate a 3D model of the MIT campus utilizing already existing information and tools and by designing and implementing new tools. The ge neration of the model is based on database information at MIT, namely MIT floor plans and the base map of MIT. Given these files and additional annotation files, our goal is to make a fully automated model generation tool with a minimum of user interaction.

My part of the project has mainly been to understand how a powerful tool (BMG) to extrude 2d architectural floor plans to 3d models works, how it be used for our purpose and how to restructure it. I have examined how a batch version of BMG can be implemented, since we are interested in both an interactive and a batch version of BMG. Also my work has been to make a high level design of the 3d campus model generation system.

Possession of a complete 3d model of MIT campus leads the way to many applicatio ns. Examples are virtual tours online, 3d navigation at MIT for Internet browsers and even augmented reality if the floor plans contain plumbing and electrical wires information. The model can be generated using batch programs running anytime. Since the De partment of Facilities at MIT provides the floor plans in AutoCAD and update them frequently, fetching the floor plans and generating a 3d model of the campus will always yield an up-to-date version, reflecting even the latest changes in buildings.

A gene rated model will be empty of furnishing but even that can be handled, using an automated furnishing program developed at MIT [3]. This will not be discussed further in this report.

Limitations

As this project is being written in the beginning of May, my final results cannot be represented because I will continue to work on the project over the summer. This report is a documentation of a snapshot of the project in the beginning of May 2001.

Structure of report

The following chapter, Design, gives an overview of the model generation system, describes the tools needed for the generation and the steps in the pipelined processing of data.

The Implementation section describes the tools that I have developed for this project. The following section, BMG, focuses on my work with BMG. Then the results of our group are described, and finally a conclusion is given.

Appendix A describes how to use the programs, including how to check programs out from the repository, make them and execute them.

Design

This chapter focuses on the design process, going into details on existing tools, what information is provided, was is not and how it can be obtained. Some of the tools were not designed to work with MIT floor plans, so we had to look into ways to modify these tools. The processing pipeline of the generation system and the dataflow is described. Ideas how to run the system in parallel are outlined and a design of an error handling system is described, which would help automate the generation process.

Overview of the model generation system

The Department of Facilities (DOF) at MIT maintains a database consisting of architectural two-dimensional floor plans in AutoCAD format of every floor in every building at MIT. That is currently more than 50 buildings and more than 800 floor plans. These floor plans are updated frequently and therefore mirror the actual floor plans at MIT. Also DOF provides a two dimensional base map of the whole MIT campus, showing building contours and their corresponding numbers.

To procedurally generate the 3D model of MIT campus, one first has to fetch all floor plans of the base map from the DOF database. Then, using the BMG extrusion tool being described below, use the 2d floor plans as a source to generate 3d floor plans. The 3d floor plans must be stacked on top of each other to generate buildings and finally each building is transformed to fit to the base map. This pipeline process requires some extra information that must be provided to generate correct models. This will be described below.

Tools provided

BMG

The powerful tool to generate 3D building models given 2D floor plans, BMG [1], which stands for Building Model Generation, is to be used for the MIT campus model generation. BMG was developed in 1996 at Berkeley, and has to be modified to make it work with MIT floor plans, which are different than Berkeley floor plans. Since I used a great fraction of my time working with BMG, I devoted a section in this project report describing what I did to be able to use BMG.

Acad2Ug

This tool converts AutoCAD files in ASCII format (DXF files) to UG files. UG, Unigrafix, is a graphics model file format originating from Berkeley that we are using, because BMG uses that file format. Acad2Ug can perfectly convert Berkeley floor plans from DXF to UG, but produces errors when converting MIT floor plans. Todd Drury from the group has modified Acad2UG, so it now works with both MIT and Berkeley floor plans.

Ug2Iv

Since we don’t have a standalone UG file viewer, UG files can be converted to the Open Inventor file format (.iv) and be viewed with *ivview* for example.

Tools needed

A couple of tools had to be developed to support the automatic generation of the 3d model.

1. Program to fetch files via URLs (ReadURL)

2. Program to extract room numbers and types from MIT floor plans (getrooms)
3. A batch program developed by Todd Drury that assembles the campus model by calling the other tools and retrieving building transforms from the base map and floor plans. Please read his report for the details.

URL fetching tool
To be able to fetch floor plan files, an Internet fetch file program is needed. Java makes that task very easy, since all networking capabilities are natural parts of Java.

Room information filter
Room information in form of types and numbers is needed, both for extrusion purposes using BMG and for other purposes. Also the automatic furnishing system [3] takes advantage of room information when furnishing rooms.

For MIT floor plans in the DXF format, room information (types and numbers) exists in the layer A-AREA-IDEN. The convention used is that one object represents the type of the room and two other objects represent the room number. The first object holds the floor number; the second object holds the last two digits of the room number. These two numbers concatenated make up the room number. It seems (it has not been confirmed at this time) that the room information is in canonical order in all MIT floor plan DXF files. This makes it fairly simple to retrieve room information.

Information needed

Transform information is needed for *each* floor plan to be able to generate the campus model. The floor plans are in a different coordinate system than the base map. All z-values are zero (they are two dimensional). They are also rotated, so just translating the floor plans would not be adequate. A 3x4 transform matrix must be provided with the correct scale, rotation and translation.

How can that information be obtained?

Most of the floor plans have a north arrow in the DXF drawing. The rotation, which is only neede d about the z axis pointing up from the base map, can be found by comparing the north arrow on the DXF floor plans with the known north direction of the base map. The scale and translation can be found by comparing the building contours of the floor plans and their corresponding contours on the base map. This is possible, because a building number is provided for every building contour on the base map. See the report of Todd Drury and Keyuan Xu for details. The translation along the z-axis, in other words, every floor plan height above the ground level or mean sea level, is also needed. It turns out that this information only lives in spreadsheets that have to be parsed.
For floor plan extrusion purposes the door heights and window heights must be known. This information does not exist officially according to Greg Knight at DOF. Default values will be used.

The plan (which is not yet implemented) is that once the transform information is known, it should be stored at certain URLs for future use. This would make it much simpler to perform the procedural generation of the model.

The processing pipeline

The processing pipeline that was discussed briefly in the overview in the beginning in this chapter, is the processing of taking the raw data in form of floor plans as input, and using the tools provided produce a complete 3d model as output.

Overview of the pipeline

```
1. from MIT campus base map in the DXF format, extract vertices, faces and
names of buildings and store them in a UG file.
2. for each building
   a. for each floor
      1. download DXF floor using the URL fetching tool.
      2. convert it to UG using Acad2UG.
      3. find transform (rotation: north arrow, trans/scale: compare to base map
      UG file and floor height information file)
      4. use BMG to extract floor to 3D
      5. transform BMG output file to fit to base map
      6. save UG file
   b. update building information file
3. assemble all UG files for all floors from all buildings with the base map
```

Figure 1. Outline of the pipeline.

On the next page, Figure 2 outlines the different elements and the data flow in the pipeline.

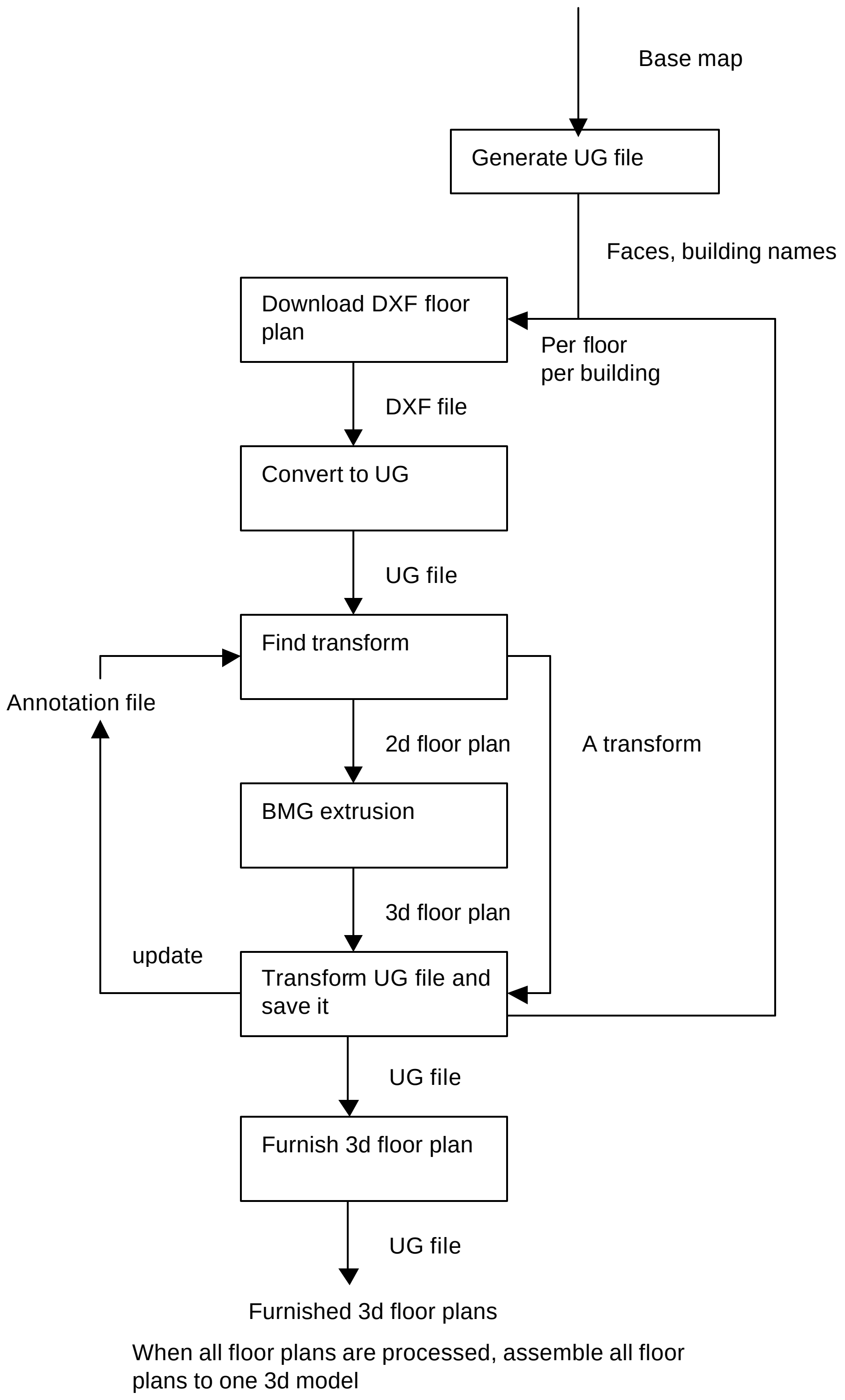


Figure 2. Diagram of the processing pipeline and data flow

Error handling

Different errors can potentially occur during the processing. Here is an outline of how errors can be handled. For some errors, the processing can continue, for other fatal errors the processing must be interrupted and the errors corrected.

Below is shown an extension to the procedure described above that handles errors.
The DOF log file contains information about errors that only DOF can fix.

```
1. extract building contours and numbers from the base map
If error: write missing or ambiguous identities to a log file
2. for each building
    a. get floor plan transform information from a URL. (only from found buildings)
       If error: building information has not been recorded. Use brute force method
       to obtain information (described below).
    b. for each floor
       1. download DXF floor (if error report to DOF-log file: URL problems)
       2. convert to UG (if error report to DOF-log file: file format error)
       3. find transform (rotation: north arrow, trans/scale: compare to base map UG file
       and building info file)
       3.a. If there is no north arrow, get orientation information from building info file.
       3.b. If building info file has no transform information or floor number and elevation
       information report error.
       4. use BMG to extract floor to 3D
       If errors write them to a log file. These problems have to be solved by the group.
       5. transform BMG output file to fit to the base map
       6. save the transformed UG file
    c. update building information file if necessary
3. assemble all UG files for all floors from all building with the base map
```

Figure 3. An outline of the processing pipeline (see Figure 2) that includes error handling.

Running in parallel

In general, the processing of data for the model generation is intensive, since it requires lots of file input/output and computationally expensive algorithms. The whole process of procedurally generating the 3d model can be split up in a number of concurrent processes in several different ways.

A single computer can process each building or each floor. Or another process (computer) can download all the DXF floor plan files and convert them to the UG file format.

As the BMG floor plan extrusion is very computationally expensive that task may have to take place on its own workstation for each floor.

It is important to design flexible batch programs that allow for concurrent data processing. Scripts can be written to handle the execution of these batch programs on different workstations.

Implementation

The pr ograms for the model generation system are developed in C/C++ on the SGI Irix platform using the C++ compiler, except the URL fetch program, which is written in Java. This section focuses on the few programs that I have written, and only mentions the progr ams that were written by the other members of the group. Please see their reports for details. All programs can be checked out from the repository at the Graphics Lab at MIT. For details see Appendix A.

I have implemented three tools:

1. Annotation file (building and floor plan information file) parser
2. Room information filter (getrooms)
3. Vertex merging utility (ugvmerge)
4. URL fetch utility (ReadURL)

Annotation file parser

The annotation file parser is written in C++, and can parse building information files and store them in internal data structures. The program can function as a library being used by other programs, e.g. the program that transforms models. The program is not being used at the moment.

The format of the information files being parsed is explained in Appendix B.

Room information filter

The C++ program *getrooms* filter out room number, type and position information for a given MIT floor plan in the DXF format. BMG also has its own get-rooms utility, which unfortunately does not work with MIT floor plans.
The program parses the DXF file, locates the objects that contain room information and store them in a format suitable for BMG. BMG needs to know where the rooms are on a floor for extrusion. Each line in the output file specifies a room:

```
Room number / room type / x / y
```

Where x and y are the position of the room label (often located in the middle of the room) in floor plan coordinates. The name of the output file must be x.rooms, where x is the building and floor name.

Vertex merging utility

The C program *ugvmerge* merges multiple vertices that have the same position but are being used by different edges and faces. The different parts of the source code existed at different places.
I assembled the pieces, compiled and linked it.
To check it out from the repository run

```
cvs checkout walkthru/src/ugvmerge
```

URL fetch utility

The URL fetch utility, ReadURL.class, is written in Java and uses the network functions provided by the *net* package in Java to download files from the network. It opens a stream of bytes from a

specified URL using a buffered input stream reader and the openStream method in the URL class and saves the fetched file locally.
The ASCII file ReadURL.properties is used to specify from which server and what files shall be fetched. ReadURL first reads an URL base from the file ReadURL.properties (first line).
Then it continues to read one line at a time from that file. Each line specifies the filename of the file from the server that must be fetched. For each filename, the URL source is downloaded and saved locally with the same filename.

An example of ReadURL.properties is

```
http://floorplans.mit.edu/dxfs/
NE43.1.dxf
```

Which causes the file NE43.1.dxf to be downloaded from http://floorplans.mit.edu/dxfs/NE43.1.dxf and stored locally as NE43.1.dxf.
If any errors occurred during runtime, files will not be generated locally and an error message is displayed on the screen / console window.

To run the program type at the command prompt

```
java ReadURL
```

The utility can be checked out from *walkthru/URL*.

Tools written by group members

Todd Drury wrote the program Ugtrans, which is an implementation of the data processing outlined in Figure 1, except that it does not call BMG at the moment (May 2001).
Keyuan Xu wrote a program that can generate a pseudo 3d campus model without floor plans.

BMG

In this section, I give a full description of my work with BMG: from the start, where BMG could not eve n compile to the situation now in mid-May, where it is running and being able to generate 3d models of floor plans. For further details on how to get a working copy of BMG, see Appendix A. For BMG implementation details see [2].

Starting from scratch

Structure of BMG

It took quite some time to understand the folder structure of BMG, which is written in C for the Irix operating system. The source files were spread out in different directories, and there was no documentation of the implementation of BMG. BMG is a system, which generates files that can later be used with the *walkthru* system; BMG can be checked out by executing the command:

```
cvs checkout walkthru/BMG
```

Basically the root directory of BMG (walkthru/BMG) contains all kernel source files used for extrusion, i.e. all algorithms. One subdirectory, which is illogically named BMG, contains another set of C-files. These files originate from an old program, Animator, which has been modified and used as a GUI and UG file viewer for BMG. The table below gives a short summary of the most important files and directories. All directories listed below are by default subdirectories of *walkthru* .

BMG	Root directory. Contains kernel source files for BMG.
BMG/bin	Binary directory. All executables and scripts that BMG uses are located here.
BMG/BMG	Source files for the GUI and UG file viewer of BMG.
BMG/BMG/SODA6	A data directory with files necessary for generating a 3d model of the Soda Hall at Berkeley.
BMG/lib	BMG uses the FORMS library for the GUI. The source code is located in this directory.

The main program, *bmg*, is located and must be executed from the BMG/bin directory.

Compiler and linker issues

BMG is from 1996, and 5 years is a long time in the computer science world. Since the system architecture on the Irix platform has changed since 1996 and is not compatible with the old architecture, all programs had to be recompiled and linked. At the beginning BMG could not compile at all, mainly because of the evolution of compilers. In the meanwhile compilers have become stricter and do not allow the programmer the same freedom in terms of being specific with declarations (and other issues which will not be discussed here) as back then. Another problem was that different parts of BMG were being compiled with different compiler options. This can potentially cause a program to crash, even though it compiled and linked without errors.

Finally BMG compiled but it could not link properly without some important assistance from the project advisor. The problem here was mainly that it used some libraries that were compiled using the C compiler and not the C++ compiler, which was used to compile the main BMG program. This generally causes an “unresolved external symbol” error when linking, since the naming of functions and variables is different from C to C++. It was necessary to edit some source files manually, inserting the statement “#ifndef c_plusplus” and “extern C” to be able to compile C source files with the C++ compiler.

BMG could now compile, link and actually run, but there was still a long way to go to get a fully functional version. One major problem was that the make scripts for compiling and linking BMG were copying Unigrafix header files to different directories instead of specifying a unique include path. Different versions of the header files existed at different places, which caused bus errors when executing certain commands in the BMG user interface.

I got rid of all the duplicates and changed the make scripts and make files to avoid copying of files and using same compiler options, including the include paths.

BMG working with Berkeley floor plans

Now that BMG was able to run, it was time to test its capabilities. With BMG comes a complete data directory with sample floor plans from the computer science building, Soda Hall, at Berkeley and all necessary utility and configuration files. BMG has a GUI and three windows are opened when executing the program. From one window (Figure 4), commands can be executed, another window shows the status and activity of BMG (Figure 5) and the third window displays the current UG work file (Figure 6).

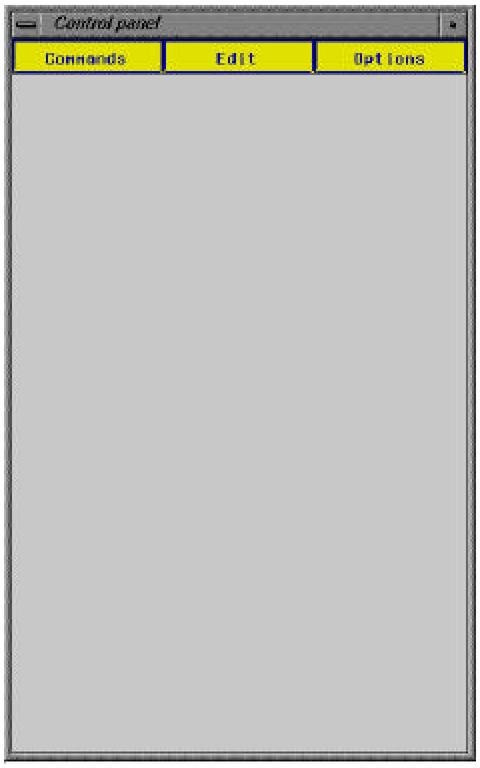


Figure 4. The main window of BMG.

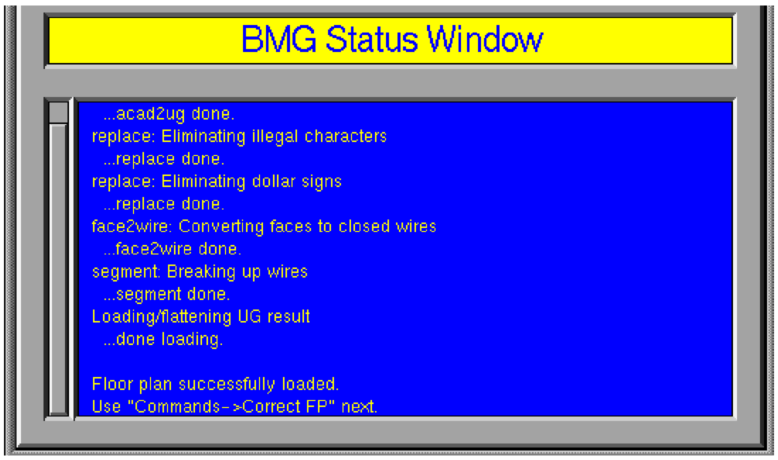
First I checked its basics UI commands as load, flatten and save UG file using very simple files. These functions worked fine.

Quickly a new set of problems appeared BMG uses a number of external tools in form of executable programs and Perl scripts. The executable programs also had to be recompiled and linked.
Todd Drury has fixed the ACAD to UG file converter. Another utility tool is the “ugvmerge” that merges duplicates of vertices that are being shared by edges and faces. The problem with ugvmerge was that no source code existed. Only fragments of the source code existed in libraries. By doing some research and with the help from our project advisor I brought the different pieces together to build a new version of ugvmerge.

Now BMG was able to load a Soda Hall floor plan, but it crashed when trying to correct the floor plan, which is the second step in the line of command calls in the BMG user interface. The pr oblem was located in the layer description file that gives each ACAD layer an internal semantic interpretation in BMG. The BMG layer category *window_jamb* had no specified ACAD layer. A quick look at the Berkeley ACAD files, showed that the window jamb laye r was AN_WN_JB.

After this correction, BMG was now capable of converting and correcting a Soda Hall floor plan. But somehow the graphics window showing the floor plan was filled with a cyan color making it impossible to see the floor plan. A small correction in the draw.c function in the BMG sub directory corrected that error.

Finally it was possible to generate a 3d model of a Soda Hall floor plan.



```
  ...acad2ug done.
replace: Eliminating illegal characters
  ...replace done.
replace: Eliminating dollar signs
  ...replace done.
face2wire: Converting faces to closed wires
  ...face2wire done.
segment: Breaking up wires
  ...segment done.
Loading/flattening UG result
  ...done loading.

Floor plan successfully loaded.
Use "Commands->Correct FP" next.
```

Figure 5. The status window of BMG.

MIT floor plans

Having BMG generate a 3d model of the Soda Hall shows the powerful capabilities of the program. It is now time to see if it can generate 3d models using MIT floor plans as input.

The problem of MIT floor plans

It has not been possible to successfully generate a 3d model of a MIT floor plan by the beginning of May. But we are close. The MIT floor plans are very different form the Berkeley floor plans and that has caused problems. Especially the fact that the MIT floor plans only have four use layers and BMG needs at least six layers to function. This will be described below.

Some of the utility scripts in the binary directory of BMG had to be modified, namely the two Perl scripts *replace* and *segment*. One problem that had to be solved was the generation of duplicates of UG-wires.

The Acad2ug tool had to be called with the –2 parameter, to avoid ignoring lines with a zero width. This modification implies that Berkeley floor plans are not converted correctly. To be able to convert Berkeley floor plans, the –2 parameter must be removed and BMG recompiled.

It is also necessary to generate AutoCAD layer information for each floor plan at MIT. The program *csblayers* extracts layer information from an AutoCAD DXF file and stores them in a file.

The MIT floor plans have different dimensions. They are often stretched and extruded floor plans can therefore not be stacked on top of each other to create a nice building without correcting, i.e. rescaling, the floors. The group has not yet addressed that problem.

The AutoCAD layers in MIT floor plans

The MIT floor plans generally have these layers.

Layer name	Description
A-WALL	Interior walls separating rooms, doors and windows
A-WALL-CORE	Structural walls including both exterior and interior walls
A-AREA -IDEN	Room number, type and position

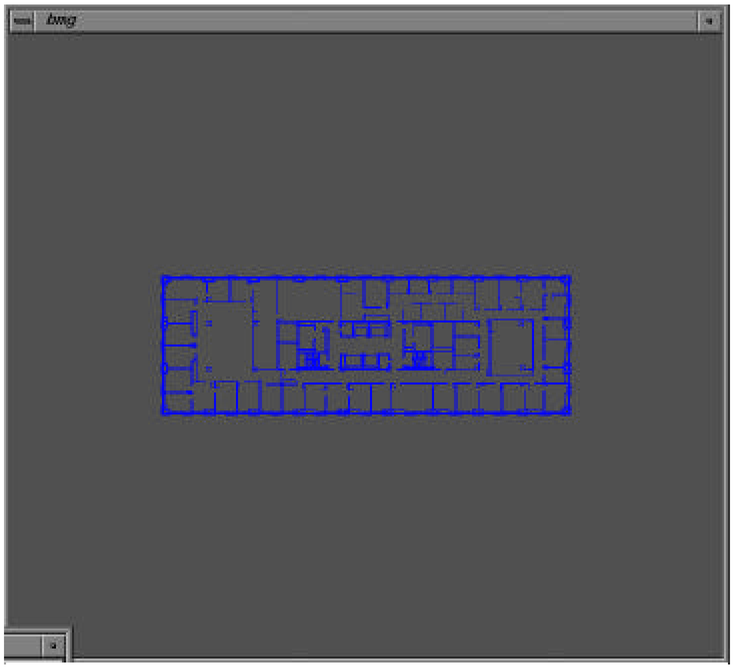


Figure 6. The current UG work file (in this case the second floor of NE43 at MIT) is displayed.

A problem with the MIT floor plans is that doors and windows are not in separate layers but exist in the A-WALL layer together with interior non-structural walls.
Right now we are working on methods to filter out the doors and windows and place them in separate layers.

BMG in batch mode
BMG was born with an integrated GUI that lets the user run different commands to generate the building models. This is fine if you want to see how BMG actually works step by step. But for our purpose a batch version is needed, since we want a fully automated model generation system.
In mid-May 2001 a batch version of BMG has not been implemented yet, since we still have to figure out exactly what other corrections are needed in order to make BMG generate 3d MIT building models. It seems to be a pretty straightforward programming task. I suggest starting with a full copy of BMG and get rid of all GUI function calls and interaction. Tarring BMG apart is a more difficult task, since all C-files share information with each other using the extern keyword. Data (and declaration) is decentralized, which makes it more confusing to figure out what data and what files are needed to perform a specific task.

Results

All tools except BMG are working. BMG is only working partly. We are able to call batch programs and procedurally generate a 3d mode l of the MIT campus without extruded floor plans. Some buildings are still not transformed correctly to the base map. The problem is that some buildings have wrong north arrows, which causes wrong rotations. The DOF has promised to fix these errors.

Conclusion

It has been very interesting to work with this project. It took a long time to get BMG to work, and I therefore only had little time to proceed and make BMG work with MIT floor plans. It still seems clear for us that the current obstacles can be handled and an almost correct 3d model of MIT campus can be generated. Almost correct means that in order to get correct floor plans, we have to wait for DOF to correct basic errors in the DXF files, such as stretched floor plans.

Other issues that have to be dealt with are the fact that MIT has domes and spheres forming buildings that have to be generated correct. In this case, BMG can be extended to handle these cases.

References

[1] Building Model Generation
Master Thesis by Rick Lewis
http://www.cs.berkeley.edu/~rickl/

[2] BMG tutorial
Rick Lewis, 5/5/01
http://www.aggiebear.com/bmg/
(Best viewed with Internet Explorer)

[3] Automatic Furniture Population of Large Architectural Models
Kari Anne Høier Kjølaas, MIT 2000
http://www.graphics.lcs.mit.edu/pub_theses.html

[4] The Walkthrough Project
http://www.cs.berkeley.edu/~sequin/PROJ/walkthru.html

Appendix

A. How to use the programs

This appendix explains how to get a working copy of BMG and how to run it. See [1] and [2] for details.
To get a working copy of BMG follow the steps below or check out README.BMG from walkthru/doc, and run "source README.BMG" which has the same effect as executing the commands listed below.

Select a working directory
```
set workdir=`pwd`
cd $workdir
```

Set root of CVS to point to the right place:
```
setenv CVSROOT /d9/projects
```

Set environment
```
setenv BASE $workdir/walkthru
setenv WALKTHRU $BASE
setenv UGBASE $BASE
setenv SYSTEM SYSTEM/E32M3
setenv LIBPATH_WALKTHRU $BASE/$SYSTEM
setenv MAKEPATH $WALKTHRU/makefiles
```

Check out necessary files and programs (the last two are optional)
```
cvs checkout walkthru/makefiles
cvs checkout walkthru/doc
cvs checkout walkthru/local/include
cvs checkout walkthru/src/{gtui,io,mat,system,togl,ug,ugvmerge}
cvs checkout walkthru/BMG
cvs checkout walkthru/URL
```

Build libraries
```
pushd walkthru/src
foreach i ( gtui io mat system togl ug ugvmerge)
  ( cd $i ; gmake )
end
popd
```

Build the FORMS library
```
pushd walkthru/BMG/lib/forms
rm -f FORMS/libforms.a
gmake
popd
```

Build BMG
```
pushd walkthru/BMG
source makem
```

Appendix A – Running BMG

This section gives an overview of how to run BMG and describes the first step of modeling a building in 3d.

To be able to run BMG, you must have compiled and build everything (explained above).
BMG needs a few data files for the extrusion. This section only describes the vital data files; the BMG tutorial [2] explains everything you need to know in details.
Assume that the data directory is walkthru/BMG/BMG/SODA6. The following inputs are required to model a building with BMG. Consult the examples to see representative contents.

File	**Comment**	**Example file in CVS directory BMG/SODA6**
DXF floor plan	The basis for the 3D model	soda6.dxf
Layer file	Lists the layers in the floor plan that acad2ug should convert to UG, i.e. the relevant layers	acad2ug.layers
Layer description file	Describes the semantic meaning of the converted layers	layers.desc
Color definition file	Provides colors to use for customizing the building appearance	soda6.coltex.ug
Portal prototype model definitions	The UG models of window and door inserts	GLDOOR6.pp.ug
3D component model definitions	The UG models of other building components to insert	disc_win.comp.ug
Reflected ceiling plan RCP layer file	Optional, used to partition spaces into multiple regions with different ceiling heights Lists the layers for acad2ug to extract from the RCP DXF file	soda6-rcp.dxf rcp.layers

In fact, only the first three items in the table: DXF floor plan, layer file and layer description file is needed to model a simple 3d floor plan with BMG. All of the above files already exist in the SODA6 directory if you did a fresh check out from the repository.

To run BMG, change directory to walkthru/BMG/bin and execute *bmg*.
The three windows (Figure 4,5,6) pop up on your screen. Initially the window with the title *bmg* is not showing a drawing.
The control panel window has three menu items. Right click with the mouse on a menu item to see the options. The menu items in the Command menu are used to generate the 3d model. The menu items in the Edit menu are used to manually modify and correct a model. The menu items in the Display menu are basically used to setup what is to be displayed.

To generate a model, you start selecting Command -> Convert and Load FP.
Then in the dialog box enter the binary and data directories relative to the current directory. If you started *bmg* from the binary directory, bin is ./ and data is ../BMG/SODA6.

After converting the floor plan, the model is generated by selecting one menu item in the Command menu at a time starting from the top. [2] explains exactly how to do this in details.

B. Annotation file format

The annotation file is in ASCII format and keeps information on building names, number of floors, total height, position on base map and transform information. The file format will likely be extended to hold information on door heights and windows heights for floors.

The format is:

```
BUILDING name
FLOORS number
TOTAL_HEIGHT height_in_feet
X  x-position of the building on base map
Y  y-position of the building on base map
Z  z-position of the building on base map
THETA rotation_angle_about_z_axis
SCALE_FACTOR x
(for each floor)
                F floor_number
                H floor_height_in_feet
BUILDING next
…
…
```