Generating a Three-Dimensional Campus Model

by

Todd Drury

May 10, 2001

Advanced Undergraduate Project
6.199
Spring 2001

Abstract

The goal of this project was to design a system to generate a three-dimensional model of the MIT campus. In order to achieve this goal, we attempted to augment an existing tool that generates three-dimensional building models. We wanted this tool to be able to build models of multiple MIT buildings, place them in correct relative positions, and do this with no user interaction. In order to do this, the system required information about the campus that exists in the form of 2D DXF floorplans for every MIT floor, and a 2D DXF basemap file that contains the location and orientation of every MIT building on campus. Although we did not achieve these goals, we did make progress towards them. This paper outlines the project in more detail, describes my contributions towards accomplishing these goals, and summarizes our successes and failures.

1. Introduction

1.1. Motivation

When people attempt to navigate a place they do not know well, they will typically look at a map to find out where things are and how to get to them. If they are smart, they will even bring the map along with them. However, despite this preparation, they still frequently end up lost. When asked how they could get lost when they have a perfectly good map telling them where they are and where everything else is, they usually respond, "Because the map doesn't look anything like what I'm actually seeing." They are absolutely correct. There is a very limited amount of information that can be presented in a simple two-dimensional map. Looking at a map of New York City cannot possibly prepare anyone for what the city actually looks like. If a map were a three-dimensional pop-up book, on the other hand, it could really help the poor, lost person find their way. This pop-up book effect is what we hoped to achieve by creating a three-dimensional computer model of the MIT campus. However, helping people find their way and helping virtual tourists try to get a better idea of the campus's appearance are far from the only uses for this technology.

Because our model will be generated by extruding building floorplans into three-dimensional models, the information in the model will not be limited to how buildings appear and are positioned relative to each other but also will extend to similar information about rooms in a building's interior. The model, therefore, will contain an absolute wealth of information about the campus. While this project is only concerned with making the base model, it could be extended to such applications as planning escape routes. Also, the model could be made more realistic if additional information were provided. Texture swatches could make walls and floors extremely realistic. Room specifications (arrangement of furniture, etc.) could also be added to make the model paint an almost fully realistic picture of campus.

1.2. BMG

The BMG (Building Model Generator) is a tool developed by Rick Lewis at the University of California-Berkeley. (A full description of the project can be found at http://www.cs.berkeley.edu/~rickl/report.ps.gz.) This tool provides the starting point for our project. It essentially takes an input of building floorplans and outputs a well-formed three-dimensional model of the building. The entire process is much more complicated than this, going through a number of stages and requiring a large amount of supplemental information and user information to complete its task, but its ability to properly extrude the walls of a building from two to three dimensions makes it the perfect starting point for this project.

Here is an outline of how it works. First, it receives a floorplan in DXF format, and converts it to its own native file format, Unigraphics. Next, it runs various correction algorithms on the plan, in order to transform the file into a condition from which a well-formed model can be generated (i.e. making sure walls meet but do not overlap at corners). Once it reaches this state, the real extrusion work begins. The floor is divided into rooms, and the walls are extruded. Various other objects (such as windows, door, and stairways) still need to be dealt with specially, but, in a general sense, this is basically how BMG works. The final result is a set of Unigraphics files that, viewed together, present a complete and correct model of the building.

1.3. File Formats

1.3.1. DXF

DXF (Drawing eXchange Format) is a standard language for representing architectural floorplans. Fortunately, it is an ASCII format and is thus technically human readable. The basic organizational features of a DXF file are its layers. A layer is just a grouping of other objects in the file and acts to provide some structure for the file, as well as helping categorize certain building features. For example, MIT floorplans have different layers for interior walls, exterior walls, room labels, etc. Beyond these layers, however, the structure of a DXF file is very difficult to understand. It consists entirely of group code/group value pairs. In these pairs, the group code is an integer that describes what the following group value means. For example, within a vertex object, a group code of “10” indicates that the next value in the file will be the vertex’s x coordinate. This format is unpleasant to read, but is extremely easy to parse and has become the industry standard.

1.3.2. Unigraphics

The Unigraphics file format is a three-dimensional modeling language developed at the University of California-Berkeley and is the native file format of the BMG. Unfortunately, it never gained a large following and, therefore, lacks commercial viewing tools. It is an extremely simple and straightforward language that is easily human readable. Beyond basic geometric entities (lines, etc.), Unigraphics also permits groups of these entities to be defined as objects and later instantiated with transforms applied to them. This ability allows files to be written in an organized and comprehensible fashion.

1.3.3. Open Inventor

The Open Inventor file format is a three-dimensional modeling language (like Unigraphics) that has become extremely widespread and popular. Since it is a subset of VRML, Inventor files can be viewed over the internet by most web browsers with only trivial file modification. (The file header must be changed to indicate that the file is VRML instead of Inventor.) Inventor files are also highly structured, and require the scene to be described in a fully specified scene graph. In this project, the Inventor file format is used only for viewing convenience.

1.4. MIT Floorplan Conventions

All MIT floorplans were written according to some conventions. As mentioned above, the main way DXF files tend to be organized is by their layers. All MIT floorplans have layers named “A-WALL-CORE,” “A-WALL,” “A-AREA-IDEN,” and “0.” Some floors have additional layers, but they do not contain information necessary for this project and, since not all floors have them, are not really conventions. The “A-WALL-CORE” layer contains all of the structural walls and any other load-bearing structures (pillars, etc.). The “A-WALL” layer contains all other walls. The “A-AREA-IDEN” layer contains all of the labels for rooms and other labeled objects (stairs, elevators, etc.). These labels are stored as “TEXT” objects and are linked to whatever object that they label only by their location. The “0” layer contains reference information about the floorplan, including an object named “PLANS-NORTH-ARROW,” which is a compass rose pointing in the direction of global north. Unfortunately, these four layers contain all of the structural information for a floor, and there are not separate layers for things like doors and windows (which are included in the A-WALL and A-WALL-CORE layers, respectively).

2. Project Goals

2.1. Running BMG on MIT Floorplans

The most essential goal of this project was to make BMG able to extrude MIT floorplans. This goal was actually divided into two parts -- making BMG work and making it work on MIT floorplans. The first goal arose because BMG was written several years ago and was not maintained. Therefore, it suffered a great deal of code rot in the interim. In fact, since there were not detailed build and directory structure instructions, it took several weeks of work before we were even able to coax BMG into compiling.

Once BMG worked, the second phase of the goal was making it work on MIT floorplans. BMG is extremely dependent on the structure of the input floorplans, and the structure of MIT plans varies greatly from that of Berkeley ones. Specifically, MIT plans are not nearly as strictly structured as the Berkeley ones are. If the MIT plans were equally structured and merely structured differently, this goal could be achieved quite trivially. Unfortunately, they are not, and BMG is dependent on having very structured input files.

2.2. Running BMG in Batch Mode

BMG is also dependent on its user interface. It constantly asks the user to check its work and decide what to do next. The user interface is also used to gather information about the plans at run time. We hoped to fully automate this process because the MIT campus contains thousands of floors, which makes it is infeasible to have a user monitor the interactive processing of each floor (as this process would probably take several hours). Unfortunately, several features of BMG made achieving this goal rather difficult. While the extrusion of a file is a fairly orderly procedure, BMG occasionally detects problems in the process that it cannot solve. Currently, it would simply ask the user what to do about these problems, but we needed to make it smarter and know how to handle these problems itself. Also, we needed to make sure that all of the peripheral data that the program might need would be available at the start of the process, since there would no longer be an opportunity for a user to provide this data at runtime. Finally, and possibly the most problematic, BMG was written exclusively as an interactive program. Therefore, the authors did not concern themselves if some functional work was technically done by the UI or if the functional part needed to interact with the user itself. They also were not worried by the fact that they created an almost impenetrable web of functional dependencies that made it extremely difficult to remove any part of the program from the rest, much less one as large as the UI.

2.3. Positioning Buildings in Correct Relative Positions

BMG was also designed to extrude single buildings. It was our job to try to place these buildings in correct position, relative to each other. To do this, we needed to find a transform that would move each building from its own local coordinate system to some campus-wide system. This goal had the advantage of being more or less separated from the BMG itself and, as such, had the potential to be written as an independent subsystem without dependencies on BMG code.

3. My Contribution

3.1. Viewing Unigraphics Files

Due to the lack of commercial viewers for Unigraphics, one of the first things that this project required was a way to view its resulting Unigraphics files in order to check its progress. We chose to convert the Unigraphics files into a more widely used format, rather than attempting to write our own viewer. We chose to use the Open Inventor language due to its widespread use and availability. My Unigraphics to Inventor converter (ug2iv) currently uses a three-step process. First it reads and parses the Unigraphics file and builds an internal data structure of the file. Next it calls routines that "flatten" the file. (This replaces all definitions and instantiations with their literal geometric equivalents.) Finally, it writes an Inventor file that is visually equivalent to the Unigraphics file. This process takes advantage of a pre-existing parser and scene builder for Unigraphics (UGread), as well as the Unigraphics scene graph flattener (UGflatten).

3.2. Converting DXF to Unigraphics

One of the first problems we faced once BMG became functional was trying to get BMG to even accept MIT floorplans as input. The problem is that the MIT plans are structured extremely differently from the Berkeley plans. The MIT plans were definitely never designed with this sort of automated analysis in mind. Despite these problems, we have made progress in this direction. The initial problem we faced was that the supplied version of acad2ug (DXF to Unigraphics conversion utility) did not properly convert MIT floorplans. The images that resulted from running acad2ug (viewed by being converted to Inventor) were either missing large numbers of building features or obscured by phantom faces.

The problem of missing building features almost had a trivial solution. The default mode for acad2ug was such that it chose to ignore all lines in the DXF file that did not have a specified width. Several MIT buildings had the majority of their interior walls specified in this way. This default mode could be changed with a simple command line parameter ("-2" for allow two-dimensional entities). However, it appeared that the code written for operation in this mode had never been fully tested (since this mode does not really apply to the Berkeley floorplans, which do not use “thin” lines) because running in this mode revealed a number of other errors in the conversion process. Most of these bugs were fixed rather easily, but it is possible that similar problems will surface again if new types of floorplans are converted, since the only bugs I found are the ones that cause errors for existing MIT plans.

The source of the phantom faces that were obscuring the floorplans was more difficult to discover. One feature of many of the MIT floorplans is a layer that included a single polyline that defined an outline of the entire building. This line does not have any actual structural meaning for the building; it is just intended to be a rough outline. This line, like all closed polylines, was interpreted by acad2ug as a face, making it a solid that obscured the entire floor. When the file was converted to Inventor, the face “grew” and covered even more area than the building occupied. This is because when Inventor renders a drawing, its algorithm for breaking faces into triangles assumes that the face is convex, which is not necessarily true of building outlines. Since this line has no real structural meaning for the building, its layer is turned off in the DXF file. That is, when the file is loaded by CAD software, the layer is not visible. It is simply there as extra data. Unfortunately, acad2ug does not check to see if layers are turned on or off when it converted the file. It instead asks the user to provide a file specifying which layers to include in the conversion process. If no file is specified, every layer is used and converted. Writing a layer specification file by hand is contrary to the goals of this project, and trying to write a single file that applies to the entire campus is a losing battle, due to a relative lack of standard layer names among files.

It turned out that someone had already begun to think about this problem, as evidenced by the existence of a program called getlayers. This was intended to be an accessory program that would extract the layer information from a DXF file and output a layer specification file that acad2ug could use. Getlayers was neither functional nor finished when I found it, but it did contain a sketch of the ideas I needed to finish and fix it. For example, it did note where and how a layer could be turned on or off in a DXF file. With the help of the layer specification files generated by getlayers, acad2ug could run on an MIT floorplan, and produce a visually correct Unigraphics file that BMG could load and begin to operate on.

3.3. Building Space to Campus Space

Another important goal of the project was to be able to properly place buildings on campus. This is the role of UgTrans. UgTrans takes a Unigraphics basemap file that contains a map of the entire campus (The MIT basemap in contained in Appendix 1.) as input. It finds every building that is represented in the basemap, and finds the transform necessary to make the floorplans of that building line up with the basemap. To do this, it also needs each of the floorplans available in Unigraphics format. The floorplans then undergo a three-step process of rotation, scaling, and translation. After a floor's file is loaded, its compass rose (direction of north) is sought. The concept of north for the floorplan and the basemap are compared, and the floorplan is rotated into alignment with the basemap. Next, axis-aligned bounding boxes are found for the building in the basemap and in the floorplan, and the ratio of the sizes of these boxes is the scale factor necessary to make the floorplan the same size as the building on the basemap. Finally, the floorplan's scaled bounding box is recalculated and the difference between its lower left corner and that of the basemap's building's bounding box is calculated and used as the translation necessary to move the already properly sized and rotated floorplan into the right position relative to the basemap. At this point, the Z (elevation) translation should be found, but I was only able to approximate it because the height of each floor at MIT is not known. Therefore, I am limited to using a standard height (such as twelve feet) for each floor. Once this complete transform is found, a new Unigraphics file is output. This file defines a single object consisting of the entire original floorplan and instantiates that object once, applying the entire transform found above to it.

In order to demonstrate the ability of UgTrans to properly transform floorplans, I wrote a companion program that downloads every floorplan on the MIT DOF site, runs acad2ug on it, and calls UgTrans, followed by ug2iv, on the resulting file. The result is a huge set of Inventor files, one for every floor of every building on campus. In addition, the demo program outputs a "master" Inventor file, consisting entirely of Inventor file include statements (one for each floor) such that when this master file is loaded for viewing, it also loads every floor. Therefore, viewing this file presents the desired full-campus picture, without the annoyance of it being in a single large, unstructured file. A small sample output of this demo can be found in Appendix 2, and its block diagram appears in Appendix 4.

4. The Future of the Project

The next real challenge is to make BMG fully extrude a single MIT floor (A block diagram for the finished system is contained in Appendix 3.). The current sticking point preventing this goal from being achieved is the inability of BMG to divide the MIT floorplans into rooms. BMG performs this task by taking room labels and finding the contours that encloses them. To do this, BMG needs a list of room labels and their locations. This information can be provided by a room label specifications file or entered manually by the user through the graphical interface. Since one of the project goals is to completely remove user interaction from the process, using this interface is not a potential permanent solution. There are also utilities that, for a Berkeley floorplan, will find all the room labels and output an appropriate specification file. Unfortunately, these utilities cannot be trivially altered (as many others have been) to also conform to MIT conventions. The problem is that while Berkeley room labels are presented in the DXF file in an extremely organized manner, the MIT labels are in comparative disarray. While MIT conventions do contain a separate layer (A-AREA-IDEN) specifically for these room labels, the labels are encoded as text entities. This would not be particularly troublesome, but not all of the text really counts as room labels, and there is no meta-data associated with the text to describe its purpose. More troubling is the fact that individual room labels are often broken into multiple text objects, i.e. the label “220” might easily be made up of “2” and “20.” Having multiple labels placed in a room causes BMG to improperly divide and categorize the floor space and leads to incorrect extrusion.

Conquering this challenge is clearly not the end of the road, however. This is just the first in a series of problems that must be overcome due to the fact that BMG was designed to work only on highly structured floorplans (such as the Berkeley ones), and MIT plans are not nearly this organized. For example, BMG requires that various physical entities (doors, windows, etc.) be given their own layers in the plans. We will be faced with the challenge of either making MIT plans conform to this standard, or writing our own algorithms to recognize these objects without the aid of them being in their own layers.

5. Conclusion

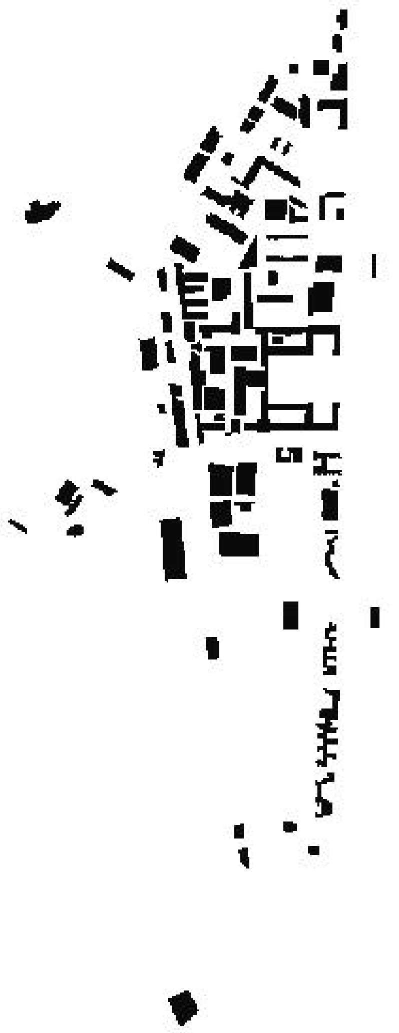
This project began with some rather ambitious goals. We wanted to take a system (BMG) that was non-functional due to years of neglect, resurrect it, and extend it. We hoped to make it more flexible, so it would not only be able to process plans from Berkeley but also plans from MIT (and, theoretically, almost anywhere else). We also hoped to make it run without the user interaction on which it currently depends. Finally, we hoped to be able to take the individual buildings generated by BMG and place them in correct relative positions to make a model of the entire campus. Although we were not able to accomplish all of these goals, we did make substantial progress toward all of them. BMG does function again, and it does begin (but not finish) to process MIT plans. We can also generate the transforms to place the buildings on campus. Hopefully, as the project continues, the remaining goals will also be met.

6. Acknowledgements

I would like to acknowledge the following people for their work on and assistance with this project:

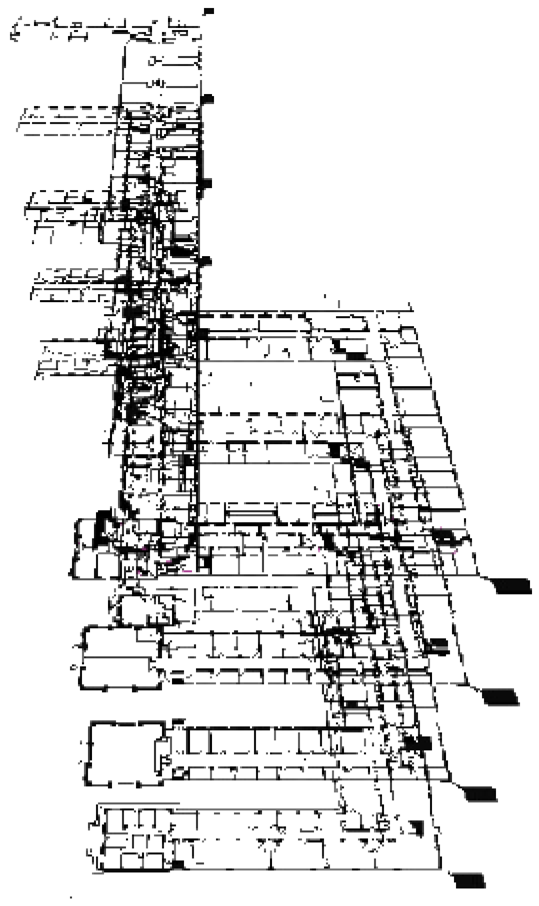
Supervisor:	Seth Teller
Co-workers:	Andreas Wehowsky
	Keyuan Xu
Berkeley BMG:	Rick Lewis
	Carlo Sequin
MIT DOF:	Greg Knight

Appendix 1:
The MIT Basemap

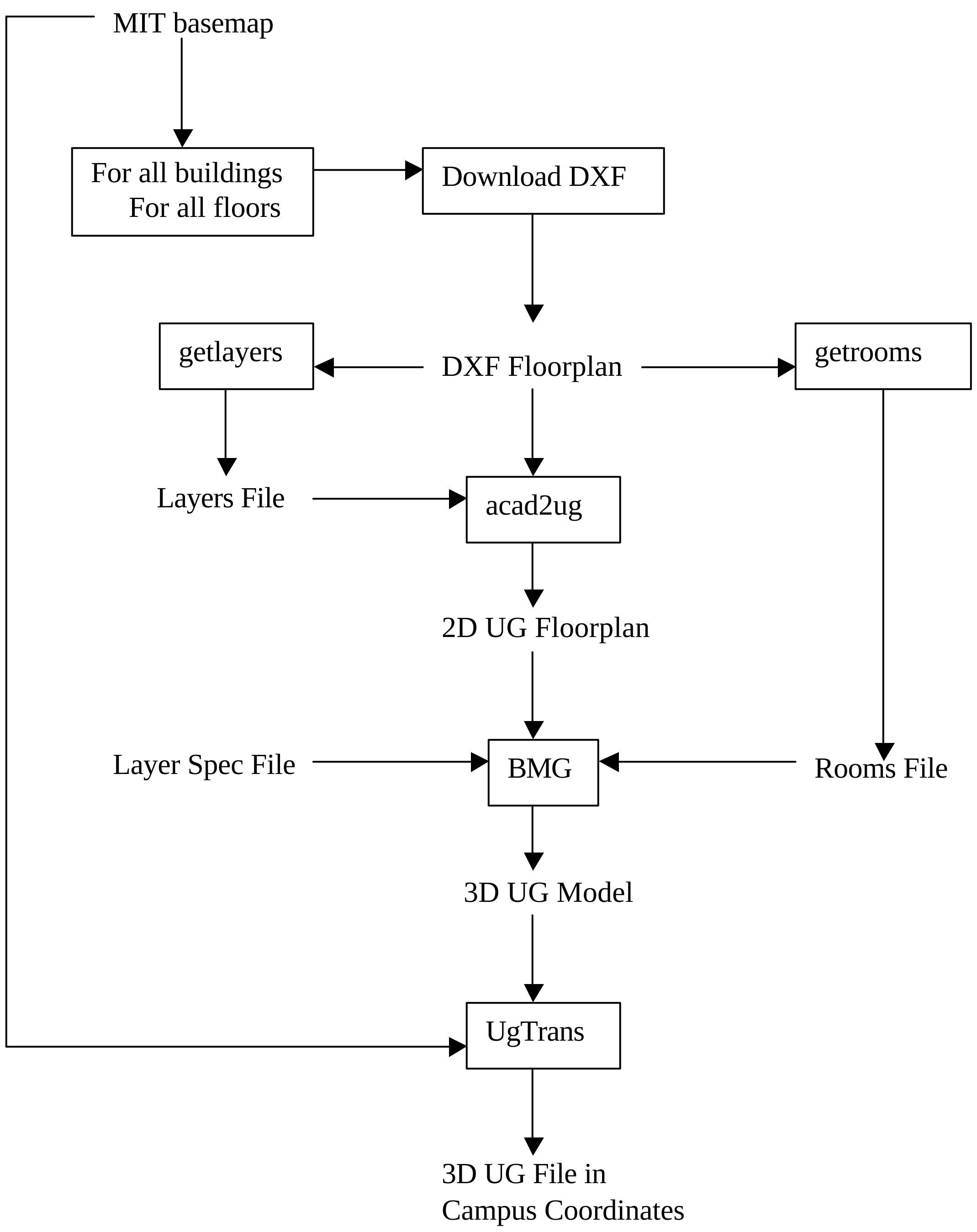


Appendix 2:
Results of Floor-Placement Demo

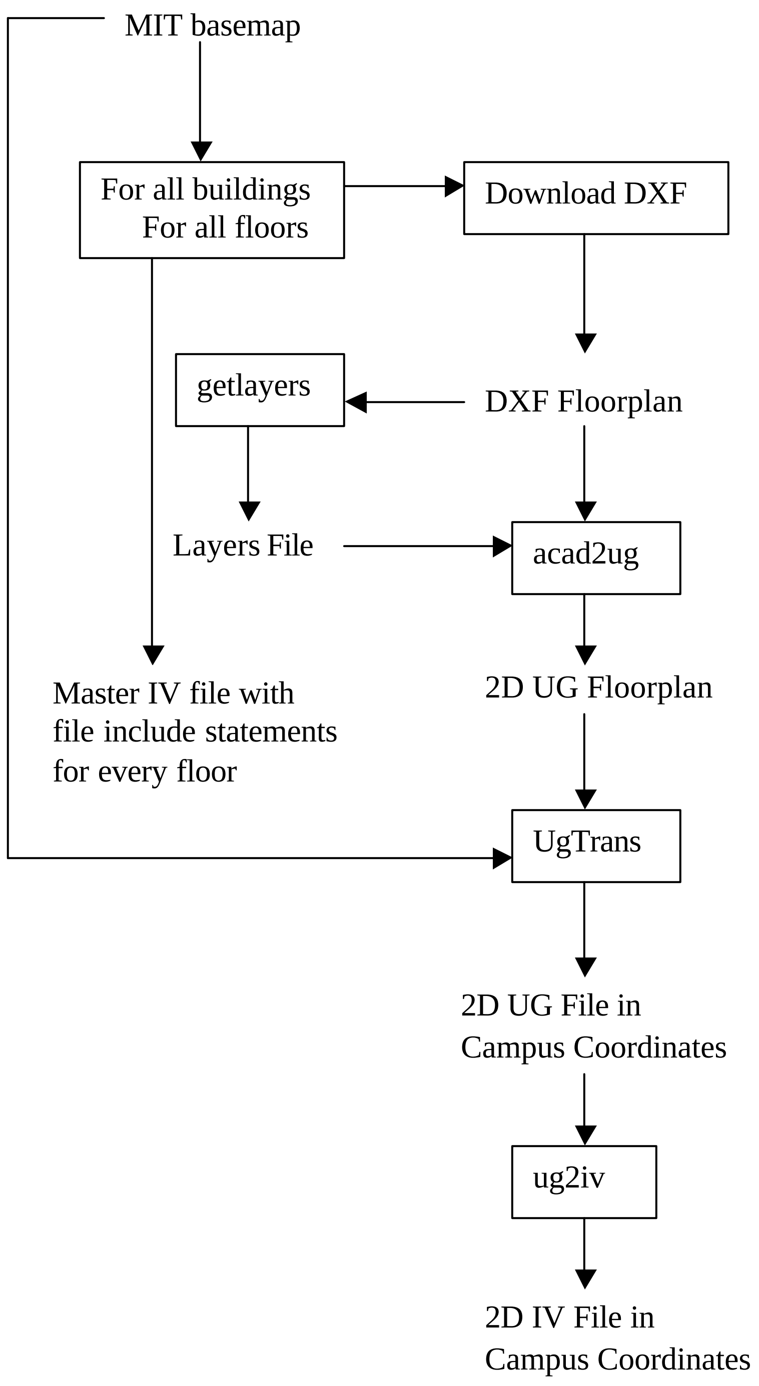
(Buildings 2 and 6 shown)



Appendix 3:
Block Diagram For the Model Generator



Appendix 4:
Block Diagram For the Building Placement Demo



Appendix 5:
Programs

ug2iv:
Location: /d9/projects/walkthru/src/ug/ug2iv
Function: Converts a Unigraphics file into an Inventor file
Usage: ug2iv infilename outfilename
Input: files are valid file names and infile contains a valid Unigraphics file
Dependencies: uglib

GetFiles:
Location: /d9/projects/walkthru/src/ug/GetFiles
Function: Downloads and converts to Unigraphics every MIT floorplan
Usage: GetFiles
Input: assumes there is a valid Unigraphics basemap file named MIT.ug.
Dependencies: uglib, acad2ug library

FloorPlacementDemo:
Location: /d9/projects/walkthru/src/ug/FloorPlacementDemo
Function: Downloads and converts to Inventor format every MIT floorplan
Usage: FloorPlacementDemo
Input: assumes there is a valid Unigraphics basemap file named MIT.ug.
Dependencies: uglib, acad2ug library

acad2ug:
Location: /d9/projects/walkthru/BMG/acad2ug/acad2ug
Function: Converts a DXF file to Unigraphics
Usage: acad2ug filename
Input: assumes filename.dxf and filename.ug are valid file names and filename.dxf is a valid DXF file.
Dependencies: acad2ug library

getlayers:
Location: /d9/projects/walkthru/BMG/acad2ug/getlayers
Function: Finds the layers in a DXF file and outputs them along with their color and whether they are turned on or off.
Usage: getlayers filename
Input: assumes filename is a valid DXF file
Dependencies: acad2ug library

getrooms:
Location: /d9/projects/walkthru/BMG/acad2ug/getrooms
Function: Finds the room labels in a DXF file and outputs them along with their location
Usage: getrooms filename
Input: assumes filename is a valid DXF file with MIT conventions
Dependencies: acad2ug library