# FRAMEWORK FOR A

# VISUAL ENERGY USE SYSTEM

A Thesis

by

CHRISTOPHER ERNEST MCDONALD

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2007

Major Subject: Architecture

# FRAMEWORK FOR A

# VISUAL ENERGY USE SYSTEM

A Thesis

by

CHRISTOPHER ERNEST MCDONALD

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

| | |
|---|---|
| Chair of Committee, | Charles Culp |
| Committee Members, | David Claridge |
| | Vinod Srinivasan |
| Head of Department, | Mark Clayton |

August 2007

Major Subject: Architecture

# ABSTRACT

Framework for a

Visual Energy Use System. (August 2007)

Christopher Ernest McDonald, B.S., Texas A&M University

Chair of Advisory Committee: Dr. Charles Culp

The goals of this research include developing and identifying software technologies, which facilitate the use of buildings described in Building Information Modeling (BIM) tools in both a simulation and visualization. The study focused on the development of a tool to fulfill the visualization needs of a Visual Energy Use System. To accomplish this, the study identified an open BIM file standard, the Industry Foundation Classes (IFC). The study also identified a video game based 3D virtual environment, the Doom 3 Engine. A tool developed during the study, IFCtoMAP, converts IFC data into the .MAP file format understood by the Doom 3 Engine. Finally, the study identified the IFCtoIDF utility, which translates IFC data into a format understood by the building energy simulation program EnergyPlus.

Data from the Building Information Modeling tool Revit Building exports to the .IFC file format, which in turn drives the two conversion utilities IFCtoMAP and IFCtoIDF. The output of the IFCtoIDF tool consists of an .IDF file that EnergyPlus uses to perform an energy simulation. The output of the IFCtoMAP tool consists of a .MAP file, which the Doom 3 game engine uses to display three dimensional first person perspective visualization.

The result of the study was the successful creation of an automated tool that converts building geometry found in .IFC files into the .MAP file format understood by Doom 3 game engine. This document details the methods employed by the IFCtoMAP software along with a brief discussion of the IFCtoIDF conversion utility.

*To my family for*
*all the loving support.*

# ACKNOWLEDGMENTS

I take this opportunity to thank the chair of my advisory committee, Dr. Charles Culp, for his time and assistance over the last few years. I am also grateful to my committee members, Dr. David Claridge and Dr. Vinod Srinivasan, for their assistance in creating this document. A special thanks to Yoshinobu Adachi of SECOM Ltd. for both the IFCsvr ActiveX object and answering questions about IFC. Finally, I would like to thank my parents for the unfaltering support they have given to me over the years.

# TABLE OF CONTENTS

Page

1 INTRODUCTION ..... 1

2 PURPOSE AND OBJECTIVES ..... 3

3 LITERATURE REVIEW ..... 5

3.1 Introduction to Computer Building Energy Simulation ..... 5
3.2 Developments in the Acquisition of Building Geometry ..... 6
3.3 Manual Simplification ..... 6
3.4 Semi-Automated Data Extraction ..... 7
3.5 Developments in Building Energy Simulation ..... 9
3.6 Fundamental Information Visualization Concepts ..... 12
3.7 Emerging Trends in Architectural Visualization ..... 15
3.8 Game Architecture ..... 16
3.9 Game Engine Genres ..... 16
3.10 Game Engine Availability ..... 17
3.11 Game Modding ..... 18
3.12 Summary ..... 19

4 PROPOSED RESEARCH METHODOLOGY ..... 20

5 SOFTWARE BACKGROUND ..... 21

5.1 Introduction ..... 21
5.2 Doom 3 Engine Considerations ..... 25
5.3 Developing Software with IFC ..... 27
5.4 Doom 3 Engine .MAP File Format ..... 29
5.5 Summary ..... 33

6 VISUALIZATION METHODS: IFCTOMAP ..... 34

6.1 Introduction ..... 34
6.2 Internal Data Structures ..... 40
6.3 IFC File Loading ..... 43
6.4 Conversion from IFC to .MAP ..... 61
6.5 Writing the .MAP File Format ..... 86
6.6 Summary ..... 89

Page

7 SIMULATION METHODS ..... 91

7.1 Introduction ..... 91
7.2 Energy Plus ..... 93
7.3 Summary ..... 97

8 RESULTS ..... 98

9 CONCLUSIONS AND RECOMMENDATIONS ..... 109

9.1 Further Work ..... 111

REFERENCES ..... 114

APPENDICES ..... 123

VITA ..... 141

# LIST OF FIGURES

Page

Figure 2.1 Framework for a Visual Energy Use System ....4

Figure 3.1 Information Visualization Pipeline[33] ....13

Figure 5.1 Example of Concave and Convex Shapes ....26

Figure 5.2 Example of Line Intersections of Concave and Convex Shapes ....27

Figure 5.3 brushDef3 Code Representation of a 128x128 Unit Cube ....31

Figure 5.4 Top Down XY Representation of a 128x128 Unit Cube in D3Editor ....32

Figure 5.5 brushDef Components of a Single Plane ....33

Figure 6.1 IFCtoMAP Design Intent ....36

Figure 6.2 IFC File Reading ....37

Figure 6.3 Methods for Geometric Transformation Left, .MAP File Writing Right ....38

Figure 6.4 Internal Data Structures ....38

Figure 6.5 Diagram of Major Component Classes ....39

Figure 6.6 Building Class Diagram ....40

Figure 6.7 `GeoObjects` Class Inheritance Diagram ....42

Figure 6.8 File Reading: IFCLoader Class Diagram ....44

Figure 6.9 IFCtoMAP Screenshot Showing the 'Select IFC Input File' Dialog ....44

Figure 6.10 User Feedback Dialog ....45

Figure 6.11 File Reading: IFCLoader Class Implementation ....47

Figure 6.12 IFCExplorer Tree View of a Common IfcWallStandardCase Entity ....49

Figure 6.13 Cross Product of Z and X Which Yields Y ....51

Figure 6.14 Example of Local Relative Placement ....54

Page

Figure 6.15 IFCExplorer Tree View of IfcProductDefinitionShape .....55

Figure 6.16 IFCExplorer Tree View of IfcRectangleProfileDef .....57

Figure 6.17 IFCExplorer Tree View of a Boundary Representation Object .....58

Figure 6.18 IFCExplorer Tree View of a IfcFurnishingElement Object .....61

Figure 6.19 Closed Polygon Loop Representation of a Wall .....63

Figure 6.20 Vector Projection of Point *P* onto Vector $\overrightarrow{V_L}$ .....65

Figure 6.21 Determining the Sign of Distance .....67

Figure 6.22 Constrained Triangulation Example.....70

Figure 6.23 Line Intersection Test .....72

Figure 6.24 Extruded CDT IfcArbitraryClosedProfileDef .....74

Figure 6.25 Example of Two Possible Hole Types .....75

Figure 6.26 Labeled Exterior Edge and Hole Points .....76

Figure 6.27 Example of Injecting a Hole into an Exterior Edge .....77

Figure 6.28 Minimizing Wall Points .....79

Figure 6.29 XY View of Minimized Wall Points .....80

Figure 6.30 Example IfcRelVoidElement(s) such as Doors and Windows.....80

Figure 6.31 Projection of Two Lowest Points of IfcRelVoidElement on to Wall.....81

Figure 6.32 Projection of $H_0$ and $H_1$ on Line Segment 2-3 .....81

Figure 6.33 Three Segments Created by Wall Opening. .....82

Figure 6.34 Left: Door Extrusions, Right: Window Extrusions .....83

Figure 6.35 Door in Red, Seen When Wall Width Is Thicker than Door Width.....84

Figure 6.36 Doors Rotate Clockwise About Point $H_0$ .....84

Page

Figure 6.37 Doors: Single on the Left, Double on the Right ..... 85

Figure 6.38 Simple Generic .MAP File ..... 86

Figure 6.39 Example `Light` Entity ..... 87

Figure 6.40 Example `func_rotatingdoor` Entity ..... 88

Figure 6.41 Example `func_static` Entity ..... 89

Figure 7.1 Three Representations of Langford Building B: Left Revit Building, ..... 95

Figure 8.1 Arial View of Architecture Building B on Campus ..... 98

Figure 8.2 Langford Building B Floor Plans ..... 100

Figure 8.3 Langford B Lobby: Virtual Representation on Top, Actual on Bottom ..... 102

Figure 8.4 Woodshop Office: Virtual Representation on Top, Actual on Bottom ..... 103

Figure 8.5 Auditorium: Virtual Representation on Top, Actual on Bottom ..... 104

Figure 8.6 Stairs: Actual on Left, Virtual Representation on Right ..... 105

Figure 8.7 Upstairs Hallway: Actual on Left, Virtual Representation on Right ..... 106

Figure 8.8 Round Off Errors ..... 107

Figure 8.9 Backwards Triangles ..... 108

Figure 9.1 Framework for a Visual Energy Use System ..... 110

# LIST OF TABLES

Page

Table 5.1 Doom 3 System Requirements .....25

Table 5.2 Doom 3 Engine Map Files .....29

Table 5.3 .MAP File Components .....30

3Table 6.1 Supported IFC Object Types/Building Elements .....48

Table 6.2 IFC Representation of IfcRectangularProfileDef .....62

Table 6.3 Intermediate Representation of IfcRectangularProfileDef .....62

Table 6.4 Example Constrained Triangulation Table of Points.....69

# 1 INTRODUCTION

According to the Annual Energy Review of 2004, the combined electrical energy requirements for cooling and heating in commercial buildings in the U.S. reached 948 trillion BTUs a year in 1999[1]. This study identified space conditioning as the single largest consumer of electricity in commercial buildings. Determining the amount of energy consumed becomes an important task for many applications, including building design and operational energy conservation. One method of determining the energy consumed by a building uses computer based energy simulation. Hui describes energy simulation as a means to "analyze the energy performance of a building to gain a better understanding of the relationship between design parameters and the energy use characteristics of the building"[2].

Determining the energy consumption of a building can be accomplished using a number of methods. The most common method uses computer based modeling to simulate the energy consumption. The most commonly used energy simulation programs have been those supported by the US government in the 1960s and 1970s. Kusuda explains that these programs have their roots in cold war studies into the "thermal environment in fallout shelters by an hour by hour simulation of heat and moisture transfer process between human occupants and shelter walls under limited ventilation conditions"[3]. These programs require a user to have advanced knowledge in the fields of building Heating Ventilation and Air Conditioning (HVAC) equipment and software programming to prepare and understand the output from a simulation.

---

This thesis follows the format of *Computer*.

According to Hong, simulation programs of the 1970s and 80s required mainframes to run the software which limited their use to "research laboratories and [were] rarely employed in building design practice because of the level of difficulty and high cost involved in their use"[4]. While there a need exists for building designers and engineers to improve their understanding of the buildings they design and operate, the complexity of simulation programs has continued to limit the user base. According to the Crawley, Donn, Hui, and Lam surveys; building designers and engineers want to improve building energy simulations by making use of existing Computer Aided Drafting (CAD) design tools, thereby reducing the complexity of simulation preparation and offering a more intuitive and visual correlation between the results and the built environment[2, 5-7].

As the computing resources for running the simulations have become economically accessible, so also have the resources for visualizing results and linking CAD tools to the simulation. This work will be a significant first step in the direction of bringing together CAD tools and energy simulation results in a three dimensional virtual environment[4, 8-10]. Although the underlying technologies for each aspect of this project have existed since 1999, the software developed for this thesis is unique for the Architecture, Engineering, and Construction (AEC) industry. To date, published reports of software technology developed for this thesis have not been found in any of the searches in the journals cited in the References section.

# 2 PURPOSE AND OBJECTIVES

The objectives of this research include identifying and developing software technology, which provides a link between a building's geometry from an object oriented CAD file with the calculation of a simplified energy simulation of the same building represented in the CAD file. The results will then be displayed visually in a spatially relevant virtual environment.

To accomplish these objectives, the author performed the following tasks:

1. The author selected an open CAD file standard to provide the geometry data for the visualization and simulation.
2. The author selected a video game based 3D virtual environment to display the building's geometry and the results of the energy simulation.
3. The author identified software simulation technology that calculates the energy consumption of a building using the same geometry as used in both the CAD file and the 3D virtual environment.
4. The author developed software that represents the technology framework for extracting selected information from the CAD files and transforming that information into a format usable by the 3D virtual environment and energy simulation.

This research will develop and identify the software requirements to link a simulation and a visualization together using a single CAD data source. Figure 2.1 illustrates the intended flow of information from the Building Information Model (BIM) software through the CAD file standard into both the 3D virtual environment

and the energy simulation software. The sum of building information model coupled with the visualization and energy simulation components represent the Framework for a Visual Energy Use System.

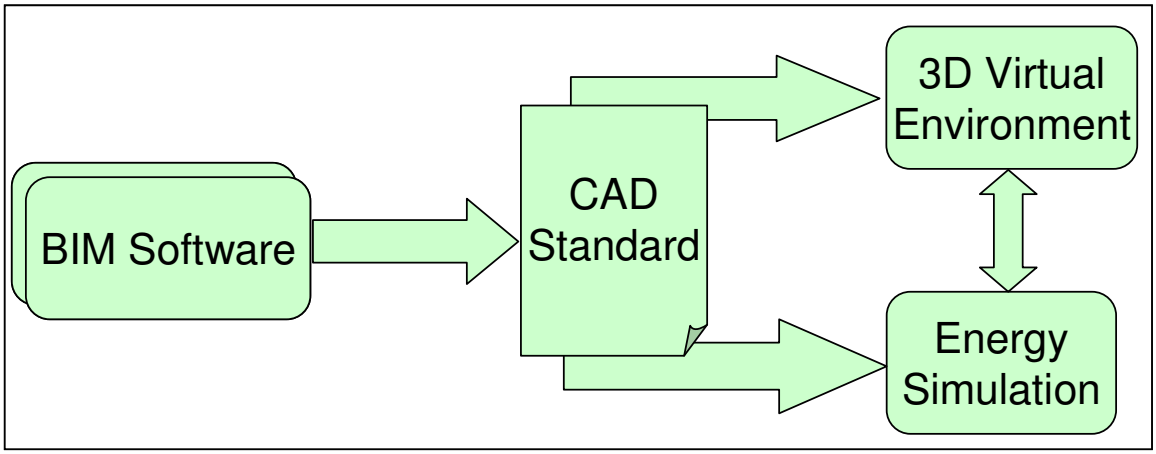


**Figure 2.1 Framework for a Visual Energy Use System**

# 3 LITERATURE REVIEW

The review of related literature included the following categories: 1) computer based building energy simulation; 2) fundamental information visualization concepts; and 3) emerging trends in architectural visualization. The primary sources for this literature survey included books, journals, conference proceedings, and similar publications from both the building industry and computer science communities.

## 3.1 Introduction to Computer Building Energy Simulation

Methods of evaluating energy consumption in buildings have evolved and have included simplified hand calculations, measuring the usage of each component of a system over time, and computer based modeling. The first method, simplified hand calculations, requires training and produces a broad range of estimated loads and consumption. However, Clarke states that "calculations are based on analytic formulations that embody many simplified assumptions," which render the results difficult to translate into real world operational changes to improve building efficiency[11].

The second method, measuring the usage of each component of a system over time, produces the most numerically accurate estimate of energy consumption; however, it also requires the most expense. Installing monitoring systems, and gathering and compiling data requires time and significant capital investment.

The third method, computer based modeling, simulates the energy consumption of a building. This method avoids the simplifying assumptions used in manual calculations and the time and expense of costly monitoring systems.

## 3.2 Developments in the Acquisition of Building Geometry

Multiple contemporary approaches to acquiring building geometry for energy simulations and visualizations include manual simplification of a building into abstract zones and scanning CAD files with proprietary software. According to Bazjanac, when using the current generation of energy simulation programs, up to 80% of the time can be required for the preparation of the building geometry and defining the building's zones[12]. Building geometry involves the physical layout and materials of the building as they relate to energy simulation. Waltz explains that building elements that partition zones such as walls, windows and doors (both exterior and interior) have importance for simulation purposes, whereas adiabatic walls within a single zone have no impact on an energy simulation[13]. For the purposes of this research, adiabatic walls and doors between zones will be taken into account, as they provide for spatial context within the visualization.

## 3.3 Manual Simplification

Bazjanac describes the first method of acquiring building geometry, manual simplification of a building geometry into abstract zones as the "standard practice in preparing energy simulation input typically [involving] repetitive manual operations that in essence amounts to duplication of existing data" [12]. This method of acquiring building geometry requires the user to create an artificial representation of the building and does not make use of available geometry information in CAD files. A number of applications available have graphical user interfaces to facilitate this process and reduce the

simulation preparation time for government sponsored simulation software such as DOE-2.0, BLAST, or Energy Plus (EP). One such program known as EP-Quick (EP-Q) uses simple templates for the shape and zone layout of a building to generate the input files for Energy Plus[14]. EP-Q, as a typical example of geometry input file generation tools, allows up to 24 different building spaces / zone layouts. These tools let the users quickly generate building geometry, but introduce a layer of abstraction between the resulting calculations and the actual building. This additional layer of abstraction requires the interpretation of calculated results, rather than visually aligning these results to the building being simulated.

### 3.4 Semi-Automated Data Extraction

The second method for acquiring building geometry does not require the re-creation of information, but rather uses existing CAD building geometry information to generate the input geometry description for an energy simulation program. Bazjanac explains that the level of effort required for the preparation of a building energy simulation input file can be reduced by a factor of four through the use of semi-automated tools[12]. One factor slowing the adoption of this method in general practice centers on the unique file format generated by each CAD software package. These unique file formats restrict the use of analysis tools and require re-creation or translation of the data when moving from one building software package to another. Chaisuparasmikul laments that it "is inevitable in the traditional design process to recreate the same building model as much as seven or eight times"[15]. In the past, translation programs convert data from one application's

unique file format to another. These interface programs add development cost and require commitment to a particular software package.

The International Alliance for Interoperability (IAI) has developed a building information model (BIM) known as the Industry Foundation Classes (IFC) standard for the exchange of building information. This includes building geometry and building systems[16]. The IAI formed in October 1995 when 12 U.S. based companies joined together to address the need for interoperability in the Architecture, Engineering, and Construction (AEC) industries. The first standard published in 1997 had limited support for some processes in the AEC community. Because of the release of the IFC2.x model, the Building Lifecycle Interoperable Software (BLIS) consortium formed to assist in the creation of software data exchange interoperability. In November 2002, the International Standards Organization (ISO) adopted the IFC 2.x standard as the publicly available specifications under the title ISO/PAS 16793[17]. The ISO-PAS designation represents and important milestone for IFC because it implies a level of maturity and stability of the model that justifies implementation by commercial companies. According to Chaisuparasmikul, in the United States, the General Services Administration (GSA) and other government agencies require the use of BIM solutions for work done at their facilities[15]. At present all major BIM-CAD software developers (e. g. Autodesk, Bentley, GraphiSoft) offer a minimum of IFC core module support, and a few energy simulation tools (e. g. Energy Plus) offer limited IFC interoperability[18-24].

## 3.5 Developments in Building Energy Simulation

Haberl explains, that the first generation of Computer based Building Energy Simulations (CBES) were developed in the mid-1960s when a group of mechanical engineers formed the Automated Procedures for Engineering Consultants, Inc (APEC)[25]. APEC created a CBES known as the APEC Heating and Cooling Peak Load Calculation (HCC) Program[26]. APEC designed HCC to calculate peak heating and cooling loads and air quantities for the sizing of building HVAC equipment. In 1967 a number of the APEC and American Society of Heating, Refrigeration and Air Conditioning Engineers (ASHRAE) members formed the ASHRAE Task Group on Energy Requirements, TGER[27]. TGER published procedures for determining heating and cooling loads for computerized energy calculations in 1969. These procedures included simulating dynamic heat transfer through building envelopes, and calculating psychometric properties and algorithms for the simulation of both primary and secondary HVAC system components.

Widespread use of computer based building energy performance simulation programs grew out of the significant increase in computational power and the threat of an Arab oil embargo in the early 1970s[28]. Early development of CBES received financial support, primarily from the government, in particular the United States Post Office Department, the Department of Energy, and the Department of Defense. The first such public domain energy analysis program became known as the Post Office Program. The Post Office Program later merged with the National Bureau of Standards Load Determination (NBSLD) program[29]. NBSLD, originally designed for the cooling load

analyses of a room for the design-cooling day with a clear sky condition, was also the first program to couple a space heat gain/heat loss with the cooling/heating capacity of a building's HVAC systems through the heating/cooling coils.

The program generated from the union of the Post Office program and NBSLD released by the National Aeronautics and Space Administration (NASA) known as NASA's Energy Cost Analysis Program, or NECAP[30]. In 1976, the California Energy Commission, along with the Energy Research and Development Administration (ERDA, which later became the Department of Energy), funded the collaboration between Lawrence Berkeley Laboratory, the Los Alamos Scientific Laboratory, the Argonne National Laboratory, and the Computational Consultants Bureau to improve NECAP. The improved version of NECAP released as CAL-ERDA in recognition of the funding organizations. At the same time, another variant existed based on NBSLD, known as Building Loads Analysis and System Thermodynamics (BLAST). The release of CAL-ERDA and BLAST marked the beginning of the second generation of CBES. Shortly after the ERDA became the U.S. Department of Energy in 1978, CAL-ERDA and CAL/CON, a variant of CAL-ERDA, were merged into DOE-1[31]. In 1979, releases of DOE-2 and BLAST 2.0 became available.

The second generation of computer based energy simulators in the 1980s saw an explosion of proprietary energy analysis programs tailored for use in large commercial and residential buildings[11]. First generation CBES required mainframes to run the software, whereas second generation programs were developed to run on the emerging workstation and micro Personal Computer (PC) technologies. However, most of these

programs were neither easy to use nor well documented. In addition, the programs were still very expensive. This resulted in many of these proprietary systems not surviving into the third generation.

With the advent of improved user interfaces for the PC such as Windows in the late 1980s, the third generation of computer based building energy simulation programs also achieved the Graphical User Interface, or GUI[11]. The Department of Energy developed a directory of over 200 tools which specialize in everything from day lighting calculations to whole building load and analysis tools[32].

This thesis work reviewed feedback from users of contemporary tools. In a survey of 241 architectural and engineering firms conducted in the United States in 1997, Donn reported a low usage of simulation in the design process[6]. The results in the United States were later confirmed by a survey of 584 firms in Singapore by Lam in 1999, which found that only 1.6% of architecture firms and 46.4% of engineering firms used performance-based energy and HVAC sizing simulations tools[7]. Of the firms that used the various software tools, none of the architects who responded had training regarding their use. Among the reasons for not using the software, those surveyed cited: the extensive data input requirements, the lack of CAD design tool integration, and the disconnection between results and the real buildings. Both surveys indicate that building designers and engineers want building energy simulations tools to use existing CAD design tools, to simplify simulation preparation, and have a more intuitive and visual correlation between the results and the built environment[2, 5-7].

When choosing a method for simulation, three options will be considered: (1) the creation of a new energy simulation calculation tool that would be linked to 3D virtual environment; (2) the creation of calculations from within the 3D virtual environment; and (3) the reuse of an existing simulation calculation tool such as EnergyPlus, the latest government sponsored energy simulation tool, based on DOE2 and BLAST.

## 3.6 Fundamental Information Visualization Concepts

Given building geometry and input to the energy calculation system, the next step in building energy simulation process involves visualizing the building and the results of the calculations. The method chosen to visualize the calculation results has the potential to greatly increase the user's ability to understand the simulation results or to greatly obscure understanding. For instance, if the result of a year's worth of hourly calculations, presented in a spreadsheet with 8760 rows and 20 or more columns, it would be difficult to understand the data. However, with graphed data, trends can be seen that often disappear in tabular form. Energy calculation results become significantly more valuable when not only the numerical values convey meaning, but also when the effects of those values on the overall performance of a building become highlighted.

Blazej describes the process of creating visualizations as a "visualisation pipeline"[33]; because just as water flows through a pipeline to its final destination, information flows from an author to an audience via a "visualisation pipeline" as seen in Figure 3.1. This "visualisation pipeline" occurs in three stages: the encoding of information by the author

using a symbolic map, the subsequent display, and finally the decoding of the information by the audience.

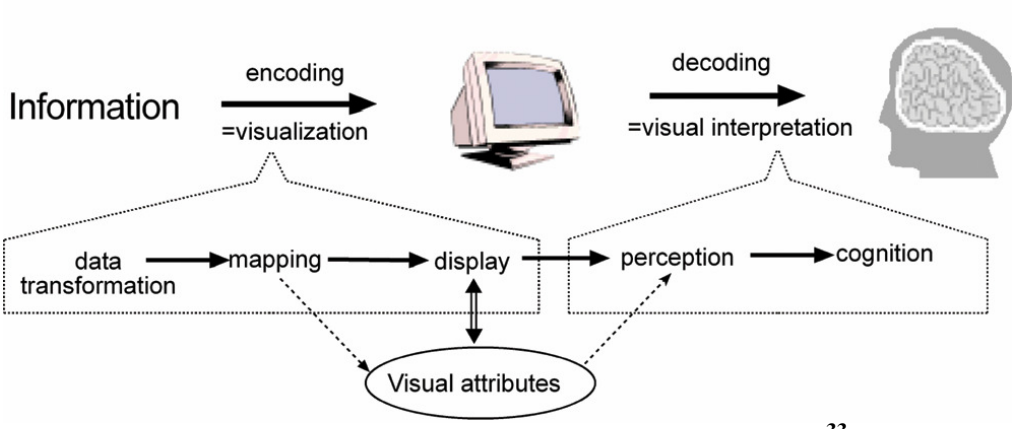


**Figure 3.1 Information Visualization Pipeline[33]**

When an author determines that a set of information could be better understood visually, the author must decide the best method to convey the intended meaning to the audience. Blazej explains that the "encoding and decoding [of information is] connected via visual attributes such as shape, position, and colour, and textual attributes such as text and symbols which themselves are represented by simple visual attributes" [33]. According to Blazej, the information to be conveyed must be transformed from a concept, a collection of numbers, or some other raw format into a symbolic map. This symbolic map contains a collection of visual attributes that give meaning to a visualization. Having chosen a symbolic map, the author then displays the encoded information to the audience.

Figure 3.1 combines the use of text, arrows, and pictorials to describe the flow of information from the left side of the graphic to the right. The text used in the graphic contains germane words and phases that convey key concepts. The concepts interconnected by bounding volumes and directional arrows. The use of arrows serves a dual role; the solid arrows describe information flow, and the dotted arrows describe a conceptual link between words. Now that the information has been encoded and displayed, the audience decodes the message.

The ultimate goal of encoding information visually is the successful decoding of the information by an audience. Wise explains that during the encoding stage, the author spatially transforms information into a visual representation revealing thematic patterns and relationships[34]. This encoding uses a symbolic map, assumed to be familiar to the audience, to translate the information into an illustration. As the audience perceives the information, they form a cognitive understanding as it passes through a personal preconceived symbolic map. The personal symbolic map, a variation of a universal symbolic map, is a specialized symbolic map defined by a group in society who share common information. For example, architects and engineers have a common set of terminology and symbols used in the exchange of information pertaining to their respective professions. When an audience understands the information being conveyed, then an author finds success.

## 3.7 Emerging Trends in Architectural Visualization

An emerging trend in the field of architecture uses real-time graphics engines for visualization. The use of video gaming technologies outside of the entertainment industry has been gaining momentum and credibility in the last seven years according to Zyda[35]. The first account found in this literature survey related to this trend reported that Richens in conjunction with the CADLAB at Cambridge in London used the Quake II game engine to demonstrate the use of existing gaming technology for remote collaborative design review in 1999[36]. The same year V. Miliano released "Unrealty," a commercial real estate demonstration program based on the Unreal game engine[37]. Both of these programs led a consumer focused visualization revolution. Before 1999, the use of visualizations existed with only those organizations that possessed resources to invest in virtual reality (VR) and "until right after the turn of the century, [these] high-end VR-systems outperformed the game systems by being capable of handling several orders of magnitude more polygons, textures, and fill rates"[38].

Lewis explains that there no longer exists a need for supercomputers in the creation of "realistic simulations and sophisticated graphics;" the more contemporary approach offers the means to "trade down from expensive gear to standard PCs running game software" [39]. Contemporary game systems have closed the performance gap between high-end VR and consumer graphics systems. The gaming industry's pursuit of realistic real-time graphic solutions drives the strides made in consumer graphics. The U.S. video and computer game industry generated record revenues of $9.4, $10.5, and $12.5 billion in 2001, 2005, and 2006, respectively[40, 41].

## 3.8 Game Architecture

Wynters and Wunsche both explain that gaming developers cannot recover the entire cost of game design on a single game title[42, 43]. Common practice in the game industry involves designing games in a modular fashion. Licensed game engines may be used in three or more major game titles. Wunsche divides games into three major components: game engines, game logic, and game art[43]. Game engines handle the input, output, and physics or interactions for the game world[39]. Game logic describes the particular application of the game engine and defines the game play and uses of the game engine and game art. For example, game logic might define a game as a vehicle racing game as opposed to a children's mystery game. However, both games use the same engine for the graphics and interaction. The engines track users, locations, and objects in complex 3D environments.

## 3.9 Game Engine Genres

Graphically based computer games come in almost as many different styles or genres as do game engines. Three very prominent genres include First Person Shooters (FPS), Real Time Strategy (RTS), and Role Playing Games (RPG)[33]. Further classifications could be made into single player games and multi-player games. However, most contemporary games have the ability to co-exist in both single and multi-player domains. The virtual environments of the FPS game genres involved viewing from the perspective of a character or agent. The user experiences the world as if seeing it through the character's eyes, and then in turn interacts with the environment using virtual

extremities. In RTS, the user experiences the virtual environment from a third person's perspective of a large environment. The user does not experience the environment as a character from within but rather as a controlling agent from without (e. g. controlling an army on a battlefield). The RPG genre is very similar to the RTS however; the user can only control one character within the environment.

The genre chosen for use in this work, the FPS style, conveys a sense of spatial presence not seen in the other two genres. This spatial presence allows the user to experience the virtual environment in as close a fashion as possible to actually being within the environment[43].

## 3.10 Game Engine Availability

Devmaster.net maintains a database of more than 230 game engines[44]. Game engines come in two major categories: open-source and closed source. Open-source game engines developed by a community of users where every aspect of the game engine, game logic, and game art can be customized. Closed source game engines allow a user to modify the game logic and the game art but not the engine itself. Open-source engines can be an inexpensive medium in which to develop and have a community of support. Many closed source games engines come from major game development companies that use the same engine in multiple games. Closed source engines generally offer more advanced options than open-source engines, offering unique features not seen elsewhere, such as code optimizations. Source code for closed source engines can only be obtained by those who license these engines.

Because both types of engines allow for the modification of game logic and game art, the author chose a closed source engine based on the online community support for modification and the author's modification experience. The chosen engine, the Doom 3 engine, developed by id Software and that receives its' named from the first released title, Doom 3, which used the engine[45]. The Doom 3 game engine released to the public in August 2004 and has been enhanced and re-released twice with Quake 4 (2006) and Prey (2006).

## 3.11 Game Modding

"Many popular game engines come with scripting languages that allow users to modify their behaviors, create new worlds for exploration, or even modify existing games into completely new ones", a process often referred to as *modding*[46]. "The customization of existing commercial games through the use of freely available development tools can provide an excellent means of creating applications … without requiring the time and money that is needed to create a game from scratch"[47]. Similar to other commercial games, Doom 3 has a number of built-in tools that the original game creators used in production which released with the games allow game modding communities to edit the game content.

Although a comparison of available game engines will be discussed in more detail later in this study, the Doom 3 game was chosen primarily for the built-in tools for modding and the size and activity level of the online modding community. The online Doom 3 modding community has approximately 10,000 registered users at Doom3World.org alone and the forums include more than 100,000 threads related to game modifications.

## 3.12 Summary

The proposed research focuses on creation of a software tool which brings CAD tools, energy simulation, and information visualization together into a virtual environment. Computer based energy simulation has matured over the past 50 years from mainframe resource intensive calculations that required highly trained individuals to much simpler software that can be run on a personal computer. However, the AEC community seeks an even more intuitive approach to energy simulation and the presentation of calculation results. This study directly addresses the needs as expressed by users of energy simulators in both the Lam and Donn surveys[6, 7].

# 4 PROPOSED RESEARCH METHODOLOGY

The aim of this research is to develop software technology for reusing buildings described in the IFC BIM standard in both a simulation and visualization. The following outlines the methodology:

1. Identify the geometric elements from IFC that will be used in the selected visualization engine.
2. Research and identify the modification requirements for the visualization engine.
3. Develop software that:
    a. Transforms building geometry in IFC files into the format required by the selected virtual environment.
    b. Simulates the energy consumption of the building.
4. Obtain drawings for a building that can be converted into the IFC file format. The selected building will be a demonstration for the virtual environment transformation and the energy simulation.
5. Analyze the technologies developed and identify areas of future research.

To meet the research goal, an on campus building was selected as a test case. The selected building is the Architecture B building located on main campus between the Bright building and the Architecture A and C Buildings. The Architecture B building currently serves multiple purposes as it contains: a woodshop, an auditorium, classrooms, and research space. The Architecture B building contains many elements of a common office building and will serve as a good example of the geometry that can be transformed for the visualization engine.

# 5 SOFTWARE BACKGROUND

## 5.1 Introduction

To meet the objectives of the proposed research, the author chose an open CAD file standard and a video game based 3D virtual environment. When choosing the CAD file standard, the author considered two important elements: first, the standard needed to contain information related to building geometry; and second, the standard needed to be capable of providing thermally relevant information about a building. It was clear that the CAD standard would need to be a Building Information Model (BIM). Ibrahim and Krawczyk describe Building Information Models as a specialized form of CAD that approaches buildings as objects and decomposes buildings into elements that contain both geometry and data associated with relevant properties of an object in a building[48]. Bazjanac defines a Building Information Model as "an instance of a populated data model of buildings that contains multidisciplinary data specific to a particular building which they describe unambiguously."[10]

The BIM-CAD file standard chosen for this project is the Industry Foundation Classes (IFC) developed by the International Alliance for Interoperability. The IFC standard first published in 1997, has been revised a number of times over the last ten years. According to Blazjanac and O'Donnell, the most current releases, IFC2x2 and later, have frozen the core kernel data, and extensions have been added which specifically target 'post-CAD' tools such as energy simulation[16, 23].

In choosing the video game based 3D virtual environment (VE), the author examined a number of important considerations: (1) the VE must be accessible to those who will use it; (2) the availability and quality of documentation; (3) the size and nature of the user community; and finally (4) the ease of extensibility of the environment.

For the technology developed by this research to be disseminated, the VE must be accessible on commonly available computer hardware and software. These requirements led to the selection of a game engine that could be run on common and popular PC hardware.

The accessibility factor being satisfied, the ability to extend the game content and functionality became the focus of the VE search. This ability requires quality documentation and knowledgeable user communities. The three largest user communities found on the web were for Doom 3, Torque and Unreal. When examining these communities, the extensibility and ease of content creation for the game engines became apparent.

Each of these game engines have 'world building' tools that allow the game designers to create and organize content within the VE. When creating content for playable VEs, called maps, Meigs explains that there exists a number of approaches ranging from one extreme, a self written standalone editor, to the other, using the tools that accompany a commercially licensed engine technology[49]. Because writing a standalone editor for one of these engines could not be created within the time constraints, the author chose a commercially available world building tool. The author

reviewed the world building tools UnrealEd (Unreal 2 Engine), and D3Radiant (Doom 3 Engine).

UnrealEd, the world building tool for the Unreal 2 game engine uses the principle of Constructive Solid Geometry (CSG)[50]. When creating a map, the world in UnrealEd begins as a giant cubic mass. To create space for a player to inhabit, the space must be subtracted from the mass. Fristch explains that objects created by CSG form as a logical combination of simple forms such as cuboids, pyramids, and spheres[51]. After performing Boolean subtractions, spaces form in which solids can be placed back into the world to form structures such as walls, stairs, and other objects. The input map files for the unreal engine, .unr files, consists of binary packages containing map geometry and texture references in addition to compiled UnrealScript code, sounds, textures, and music[52]. These files can not be read by people, a factor considered to unnecessarily complicate this study.

The D3Radiant tool takes the opposite approach to building a world. The map begins as a void, and the elements added create the boundaries of space. The D3Radiant tool uses ASCII files to store the map data in sets of bounding planes, which define the contours of solid objects. At run time, the map compiles and the engine converts the bounding plane sets into a respective set of polygons for solid convex primitive objects. The resulting set of optimized faces does not include the removed hidden or redundant faces created by adjacent objects. This face set when converted into a binary space partition tree (commonly called a "BSP tree") representation can be used for both collision detection purposes and efficient visibility calculations. Each of the primitive

objects, known as `BrushDef3`, are created from the intersection of planes and therefore cannot contain concave features. This requires that all objects generated from primitives to be convex. ASCII input files for this engine can be generated outside of the D3Radiant tool. Another noted benefit of this engine is the forward compatibility with ASCII .map files created for previous engines, such as Quake 3, developed by the same company. The forward compatibility leads to an extended life of the work developed and adds value to the work produced using this format.

While other game engines and map editing programs to choose from, these options represent two large online game modding communities. Lewis and Jacobs state that "while neither id Software nor Epic Games is in the business of supporting research, their user communities can provide active sources of help and information for game-using researchers"[39].

Between the UnrealEd and D3Radiant, D3Radiant best met the needs of this research because the input files, being ASCII, can be generated outside the D3Radiant tool. The size and activity level of the Doom 3 modding community was also considered. Therefore, the Doom 3 game was chosen as the virtual environment for this research work. Table 5.1 details the computer hardware specifications for this project. A detailed minimum hardware requirements list for running the Doom 3 game has been added in the Appendices as Doom 3 System Requirements.

**Table 5.1 Doom 3 System Requirements**

| | Minimum Requirements | Test System |
|---|---|---|
| CPU | Intel P4 1.5ghz/AMD Athalon 1500 | AMD64 3500+ |
| Graphics | Nvidia GeForce3/ATI Radeon 8500 | Nvidia 6600GT |
| RAM | 384MB | 2 GB |
| Hard Drive | 2.8GB | 120GB |
| Operating System | Windows 2000 | Windows XP SP2 |
| Software | DirectX 9b | DirectX 9c |

## 5.2 Doom 3 Engine Considerations

After selecting the Doom 3 engine for this project, an effort was made to understand and outline the known limitations of the engine. The following summarize the restrictions as they relate to this thesis.

There exists a significant and noticeable visual performance degradation appears (lower than 20fps) when more than 600K+ polygons become visible at any given time. This limitation requires the use of visibility culling of polygons, VisPortals, to decrease the total number of visible polygons. However, VisPortals do not cull imported objects from art packages (3D Max, Maya, Lightwave, etc.). After some testing, it was determined that importing full 3D models of buildings from art packages was not a viable solution for the visualization.

The following discussion illustrates the Doom 3 engine's use of `Brushdef3` primitives, which require all objects to be constructed from a set of convex primitives. Figure 5.1 shows two 2D objects, a green cross and a yellow pentagon. The red dot in the center of both figures below represents the origin. The yellow pentagon is a convex

shape and the green cross is concave. When described in the brushDef3 format, the Doom 3 Engine renders the closest set of intersecting lines. Given this rendering logic, the pentagon maintains the original shape as seen in Figure 5.2. On the other hand, the green cross becomes the left shape in Figure 5.2 because only the intersection of the closest lines to the origin (in red), as derived from the original shape of the cross, are rendered. The cross shape, when defined in this manner, yields a square and the original shape is not preserved. In order to preserve the original shape, the cross must be converted into a set of convex shapes such as triangles.

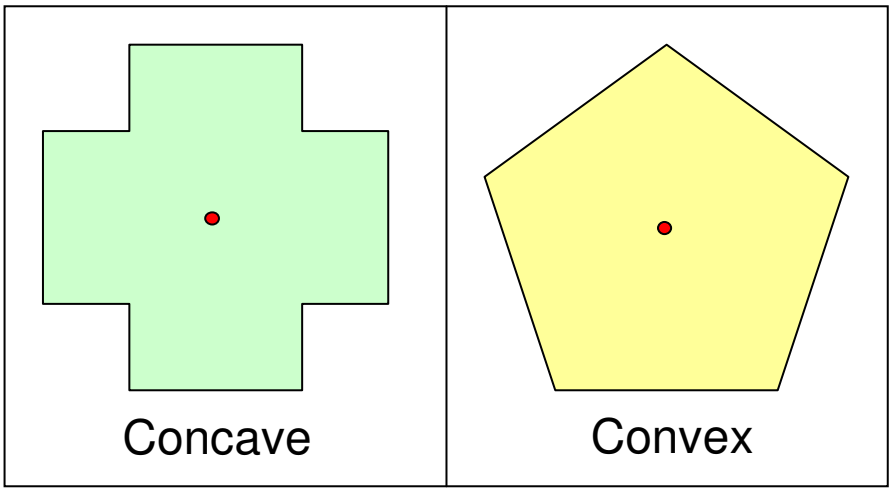


**Figure 5.1 Example of Concave and Convex Shapes**

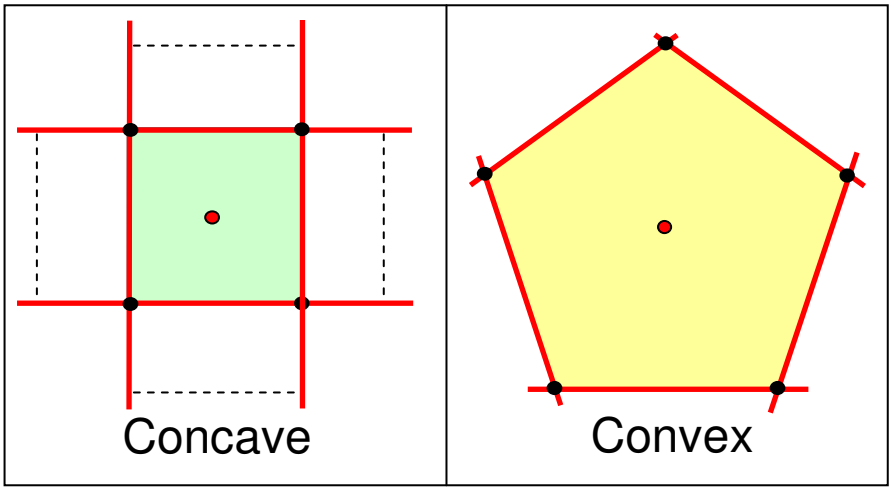


**Figure 5.2 Example of Line Intersections of Concave and Convex Shapes**

## 5.3 Developing Software with IFC

The method for acquiring building geometry chosen for this research is a semi-automated approach. The automation extracts select elements of a building's geometry from IFC and transforms that information into a representation that is understood by Doom 3 engine. Autodesk, the software developer of Revit, is a member of the IAI that published the IFC standard. It is among the major companies adopting import/export functions for IFC. For these reasons, Revit Building was selected as the BIM file creation tool and was used to generate all of the test-case IFC files.

To expedite the addition of .IFC file support for applications, the IAI has added a list of companies offering auxiliary tools, such as application programming interfaces (API) or toolboxes. Although the specification and format of the IFC model file have been published for a number of years and the AEC community holds many aspirations for

BIM interoperability, mature and inexpensive auxiliary tools for IFC implementers are still not common. Fu attributes the limited availability of auxiliary tools to the gaps in the specification and file formats between different versions of IFC [53]. These file-based toolboxes provide developers with an easy way to read, modify, add and represent the data in an IFC model. One such API toolbox is the freeware IFCsvr released by the SECOM CO.,LTD. Intelligent Systems Laboratory of Japan[54]. When selecting a toolbox, this researcher contacted a number of toolbox developers and the only Yoshinobu Adachi of SECOM CO., Ltd[55] responded.

Adachi offered an Excel spreadsheet developed in 2000 that converted extruded rectangular profile geometry from IFC2.0 to Quake 3 file format as an example of where to start learning about IFC[56]. Adachi then sent the source code for IFCExplorer, an IFC file viewing program that uses the IFCsvr300 toolbox[57].

The IFCExplorer source code offered a number of clear examples of how to use the IFCsvr toolbox. The IFCsvr is an ActiveX component for the handling of Industry Foundation Classes data. The IFCsvr API (Application Program Interface) is written in C# .NET and was used to allow IFCtoMAP to read IFC information from test files that were generated in Autodesk Revit 9. However, the IFCsvr provides not only a programming interface to IFC data for importing but also for exporting, searching, creating, and modifying. By using the IFCsvr to provide the input mechanism, IFCtoMAP can seamlessly support input and output functions for various implementations of .ifc including STEP Part21, BLIS-XML, in addition to IFC2x3 Final

and ifcXML. IFC2x3 Final is the latest revision of IFC, released in December 2005, and at this time is still the most current version of IFC.

## 5.4 Doom 3 Engine .MAP File Format

The Doom 3 game engine (D3E) uses a file with the extension .map to generate playable levels (virtual environments) for the game. Unlike other engines, D3E uses the .map file in both the level editor as well as in the game with no need to compile the editor .map file into a different format for the engine. The .map files are raw ASCII linking files used to create three more files by using the `dmap` command from a D3E command prompt. The three files created are mapfilename.cm, mapfilename.proc, and mapfilename.aas. Table 5.2 explains the purpose of these files.

**Table 5.2 Doom 3 Engine Map Files**

| *File Extension* | *Description* |
|---|---|
| mapfilename.map | Defines all the entities, brushes and patches in the map. Used to generate .cm, .proc, and .aas files by using the dmap command. |
| mapfilename.cm | Defines the collision geometry used by the physics system for collision detection. |
| mapfilename.proc | Contains all the pre-processed geometry as visible triangles, batched as surfaces, in addition to pre-calculated shadow volumes if light and brushes do not move. |
| mapfilename.aas | Contains the area awareness data for the Artificial Intelligence to navigate through the level. |

Within the D3E .map files there is a structure that allows the engine to define the virtual environment. This structure defines all entities, brushes and patches in the virtual environment. Entities are objects, which can be dynamically controlled using game logic or .map level scripts and are often composed with brushes, patches or models.

**Table 5.3 .MAP File Components**

| ***.map Object*** | ***Description*** |
|---|---|
| Entities | Functional objects in a map such as lights, monsters, items, light switches, doors. They can be composed of brushes, patches or models. These objects are dynamically controlled by either game logic or level scripts. |
| Brushes | Convex solids that form the basic geometry of a map. Objects which do not require a model are produced with brushes, such as floors, ceilings, walls, steps, beams, columns, etc. Additionally, brushes can be used to define volumes within a map with unique properties such as ladders or force fields. |
| Patches | Rounded or curved surfaces within a map which cannot be created by brushes and are defined by Bezier curves. |
| Model | Objects which are too complex to be defined by either brushes or patches. These objects are models in external 3d modeling applications and imported as .ASE or .LWO files formats. |

The .map file format has not changed radically since the release in 2004. The .map files created using the method described here can be used in all games released using the D3E. In order to understand the conversion from IFC CAD information into the .map format used in the D3E, a brief explanation of the .map file format follows.

The .map files hold entities in an ASCII format. The only required element of an entity is the `classname` variable. The `classname` variable links the entities in the .map file to functions within the game logic code. The only entity that must be present is

`worldspawn`. The `worldspawn` entity is composed of primitive elements defined by brushes and patches, or brushDef3 and patchDef2 as they are known in the engine. As described in Table 5.3, patchDef2 patches define smooth curved surfaces governed by Bezier curves.

**Code:**

```
1  Version 2
2  // entity 0
3  {
4  "classname" "worldspawn"
5  //Primative 0
6  brushDef3
7  {
8   ( 0 0 -1 -64 ) ( ( 0.125 0 0 ) ( 0 0.125 0 ) ) "_emptyname" 0 0 0
9   ( 0 0 1 -64 ) ( ( 0.125 0 0 ) ( 0 0.125 0 ) ) "_emptyname" 0 0 0
10  ( 0 -1 0 -64 ) ( ( 0.125 0 0 ) ( 0 0.125 0 ) ) "_emptyname" 0 0 0
11  ( 1 0 0 -64 ) ( ( 0.125 0 0 ) ( 0 0.125 0 ) ) "_emptyname" 0 0 0
12  ( 0 1 0 -64 ) ( ( 0.125 0 0 ) ( 0 0.125 0 ) ) "_emptyname" 0 0 0
13  ( -1 0 0 -64 ) ( ( 0.125 0 0 ) ( 0 0.125 0 ) ) "_emptyname" 0 0 0
14 }
15 }
```

**Figure 5.3 brushDef3 Code Representation of a 128x128 Unit Cube**

To better understand how primitives such as a brushDef3 object are represented in the D3 engine, a discussion follows describing the creation of a 128X128 unit cube centered on the coordinates (0, 0, 0). Figure 5.4 is a screenshot from the built-in D3E .map editor, known as D3editor, showing a 128X128 unit cube that is centered on the coordinates (0, 0, 0) as seen from a top down XY direction, with Z representing the up direction.

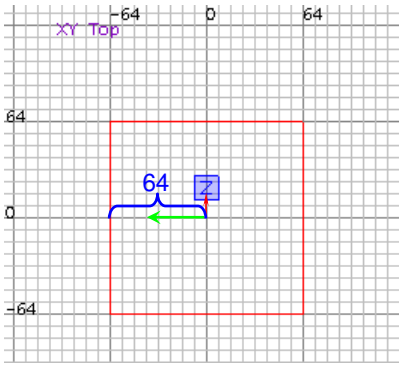


**Figure 5.4 Top Down XY Representation of a 128x128 Unit Cube in D3Editor**

The cube seen in Figure 5.4 is the same cube described in the code in Figure 5.3. Looking more closely at this code, one can see that line 6 defines a brushDef3 as the type of brush being used and line 7 starts the syntax of the object. The brushDef3 are primitive objects composed of planes. In Figure 5.3, each of the lines 8-13 that follows the brushDef3 statement inside the curly braces ("{" and "}") is a plane. Each plane is defined by three sets of numbers and a string as seen in Figure 5.5. The first set of four numbers labeled as A and B in Figure 5.5 relate to the orientation and position of the plane. The first three numbers in this first set, labeled as A, are a unit vector in the direction of the plane's normal. The fourth number in the first set, labeled as B, is the distance from the origin to the closest point on the plane along the normal defined by the first three numbers. By convention this number is negative if the distance from the origin to the closest point on the plane is in the direction of the normal of the plane and positive if the distance from the origin to the closest point on the plane is in the opposite direction of the normal to the plane.

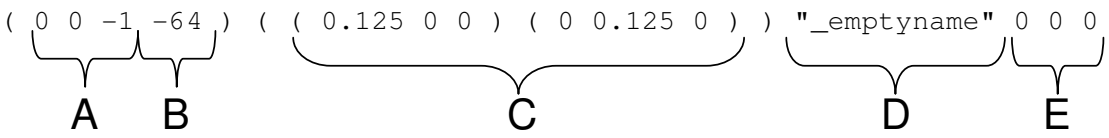


**Figure 5.5 brushDef Components of a Single Plane**

The second sets of numbers, labeled as C, relate to the size and orientation of the texture, or color information file, on the plane. Label C is not dynamically altered at the moment, as a library of known tileable textures and sizes are stored and the visualization is limited to a small texture pallet. The "_emptyname" string following the first two sets of numbers represents the relative path to the texture for this plane from the Doom 3 base folder. After contacting id Software, searching the community knowledge base and consulting the documentation, no clear answer as to the use of the last three numbers, labeled as E, has been determined. For the purposes of this discussion, the last numbers will always be zeros.

## 5.5 Summary

In meeting the technical challenges involved in this thesis work, software tools and technologies were identified based on information gathered during the literature review. Those tools include a Building Information Model, the Industry Foundation Classes; an API toolbox for interacting with IFC data, IFCsvr and, finally, a video game based 3D virtual environment, the Doom 3 game engine. Each of these technologies have implementation requirements as detailed above. The following sections discuss the visualization methods and the benefits and limitations of the technologies.

# 6 VISUALIZATION METHODS: IFCtoMAP

## 6.1 Introduction

The technology developed by this thesis work, which brings together BIM and visualization, is appropriately named IFCtoMAP as it converts a building's IFC information contained in CAD BIM files into D3E .map file format which may then be visually toured in the Doom 3 game engine. According to Terzidis, scientific visualization has been used to present a clear and faithful representation of aspects of the physical world that are impossible to perceive [58]. The purpose of this research was to develop a framework for the encoding of building geometry and energy information for architects and engineers into a visualization engine. In order to encode the information a symbolic map is required that will be quickly identified and understood by architects and engineers.

Computer Aided Drafting (CAD), and in particular Building Information Modeling (BIM), gives us such an initial symbolic map. Both professions share information about building design using BIM drawings. The design intent of IFCtoMAP is to allow architects and engineers the ability to define a building's geometry in Building Information Model (BIM) software, and in the test case this is Autodesk Revit. Then using the BIM model elements, in conjunction with additional building related inputs, generate a building energy simulation. The results will become the subject of the visualization. A Visual Energy Use System is composed of two components: the visualization and the simulation tools, both of which use the same data source. IFCtoMAP represents the geometric conversion utility that transforms selected IFC geometric data into a format understood by the Doom 3 game engine. Figure 6.1 shows a diagram of the IFCtoMAP program consisting of IFC file reading methods, geometry transformation methods, .map file writing methods, and internal data structures. The details of Figure 6.1 are seen in Figure 6.2, Figure 6.3 and Figure 6.4.

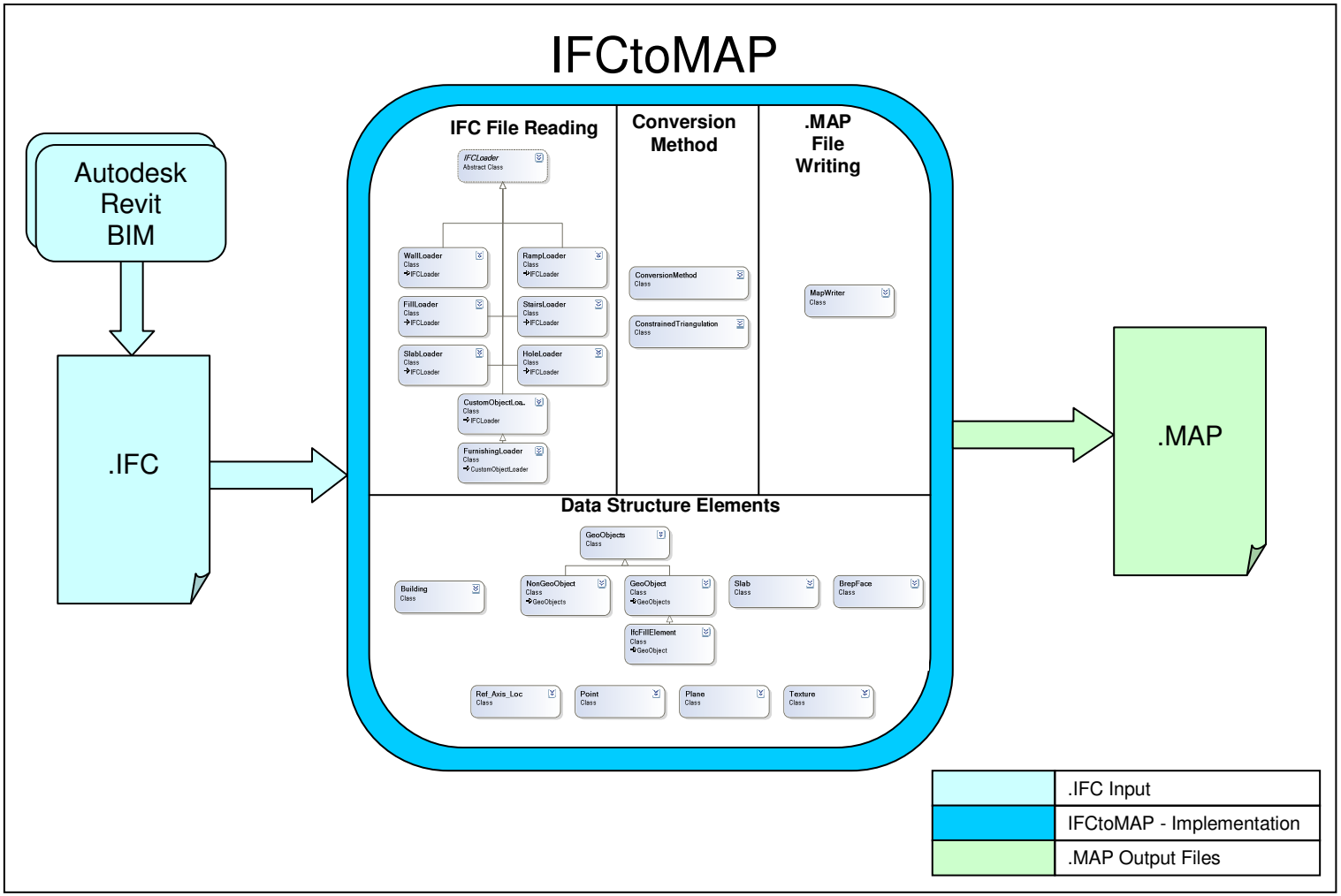


**Figure 6.1 IFCtoMAP Design Intent**

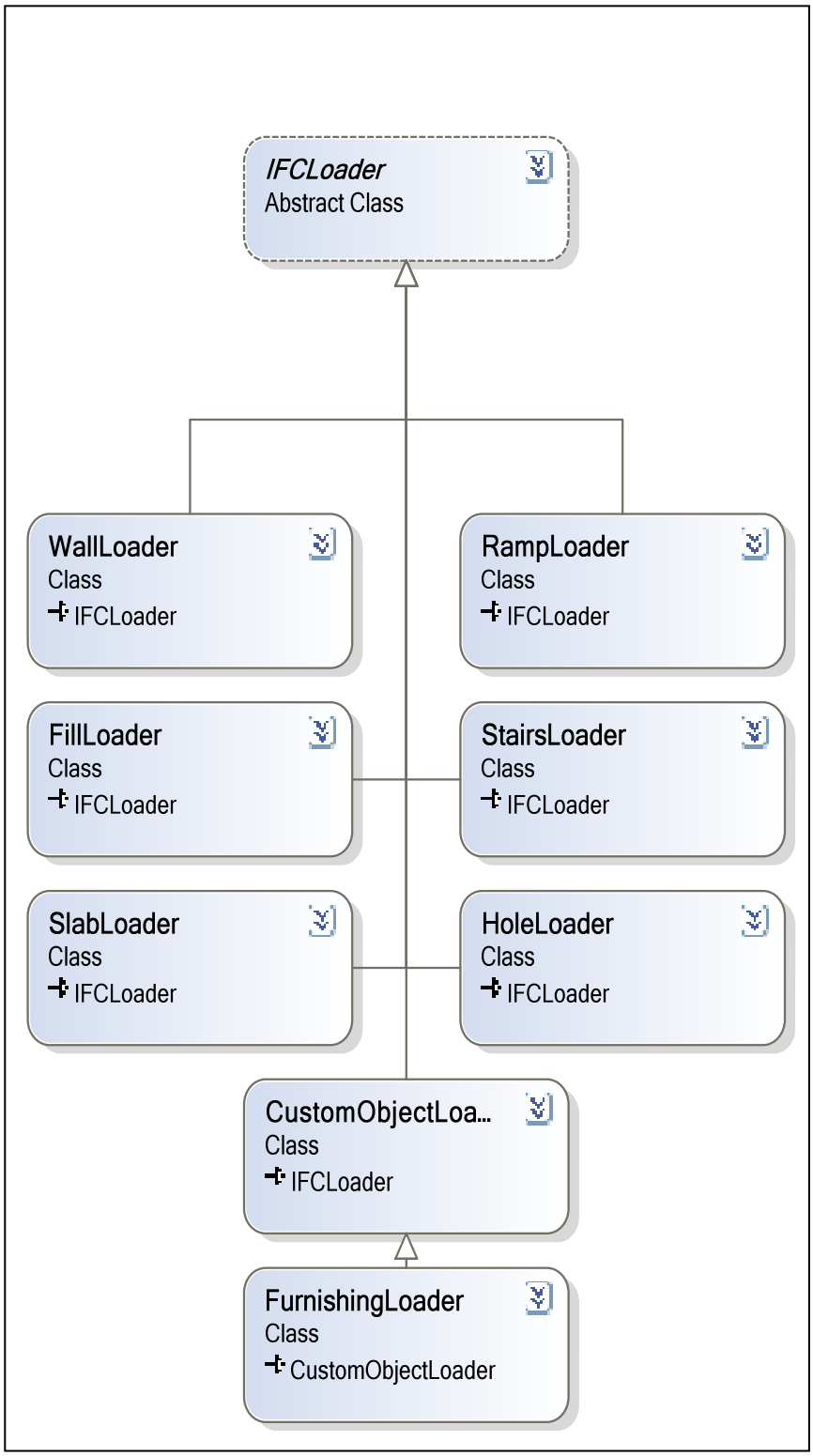


**Figure 6.2 IFC File Reading**

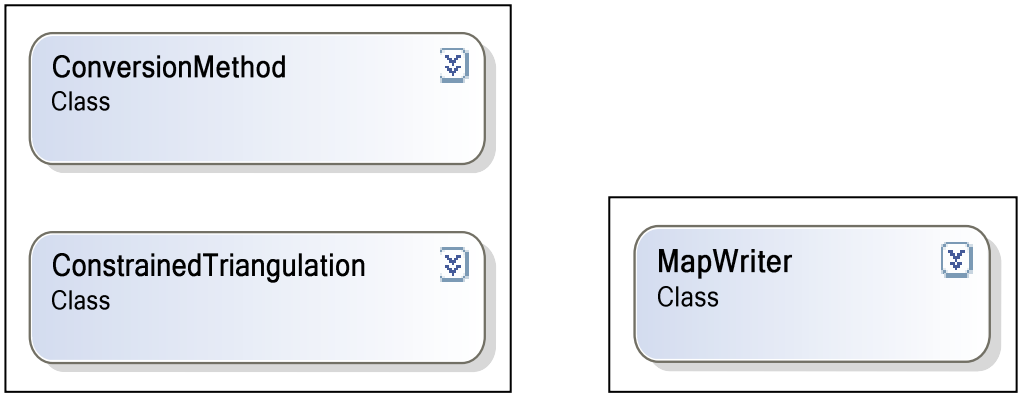


**Figure 6.3 Methods for Geometric Transformation Left, .MAP File Writing Right**

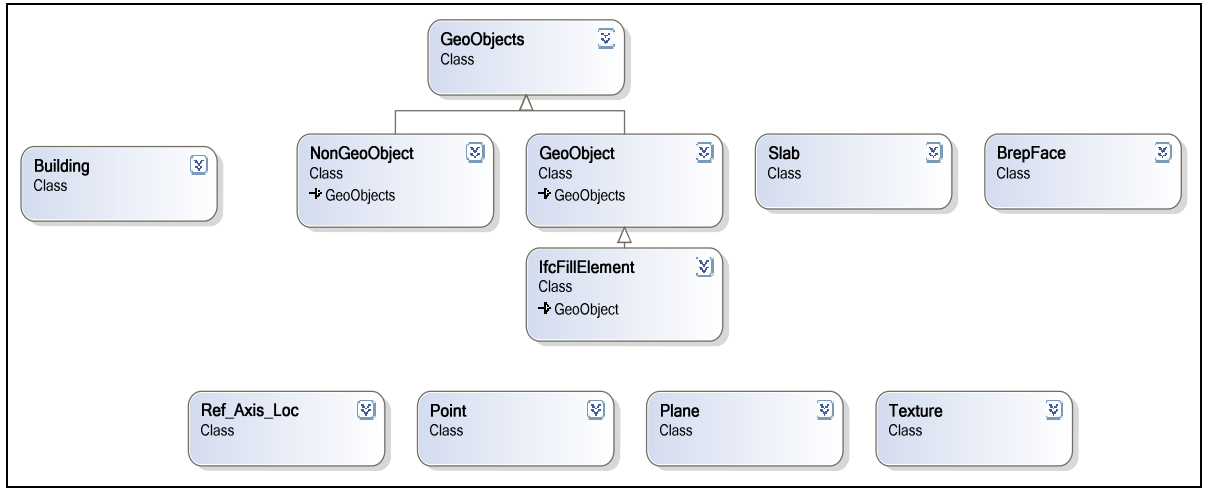


**Figure 6.4 Internal Data Structures**

The following sections detail the method used to develop the IFCtoMAP program, specifically how information found in the .IFC test files, generated by Revit, are used to create primitive brushes in the Doom 3 game engine. The IFCtoMAP component of the VEUS system is likewise broken into four major components: internal data structures, IFC file reading, geometry transformations, and .map file writing. IFC file reading uses the IFCsvr ActiveX object to traverse input files and extract specific elements from the data structure, which becomes the input to the geometry transformation. When the calculations finish, the output of the geometric transformations become .map file format compatible and written as such. The four important components mentioned above can be seen in Figure 6.5.

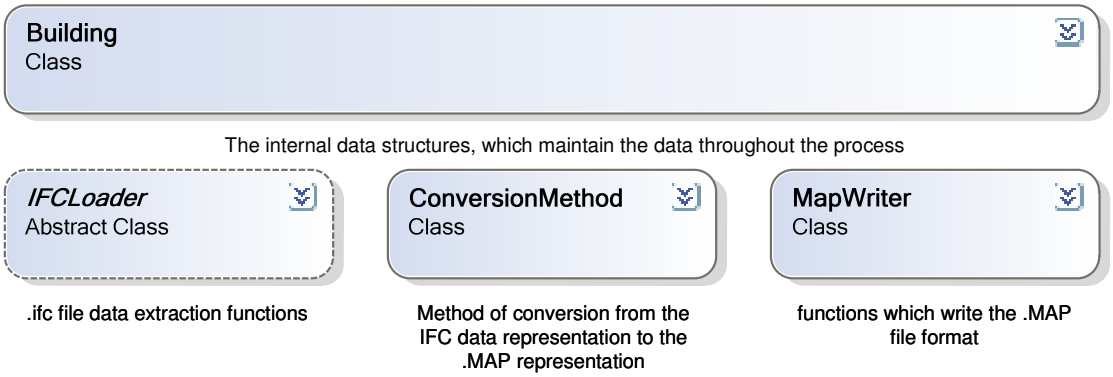


**Figure 6.5 Diagram of Major Component Classes**

## 6.2 Internal Data Structures

After extracting information from IFC and before the data is in a form understood by the .MAP file format, there was need for an internal data structure that could hold both sets of information. This subsection discusses initially how the internal data is structured in the `Building` class, followed by a discussion of the other three major component classes. A number of class objects when composed form the data structures of this work. The top most level of this data structure was the class called `Building`. The `Building` class can be seen in the class diagram labeled as Figure 6.6.

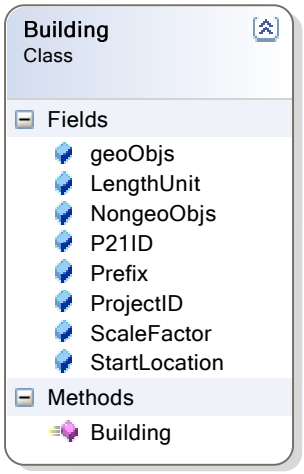


**Figure 6.6 Building Class Diagram**

As the name implies this class maintains all the information related to a single building. Among the data stored in the `Building` object are the following:

- The building's unique `P21ID` number

- The building's unique `ProjectID` number
- The user's `StartLocation` within the map
- The metric unit of length prefix called `Prefix`, which is null if the units of measure are imperial
- The unit of length called `LengthUnit`, which could be either imperial or metric
- The `ScaleFactor` for converting the units of measure from the IFC file to the .map scale
- A list of geometric objects from IFC which will be converted into .map primatives called `geoObjs`
- A list of objects from IFC that will not be converted into .map primatives but rather will be linked to 3D art models called `NongeoObjs`

The two elements in the `Building` class most often referenced are the two object lists: `geoObjs` and `NongeoObjs`. These two lists contain different object classes that inherit attributes from a common class, `GeoObjects`, as seen in Figure 6.7.

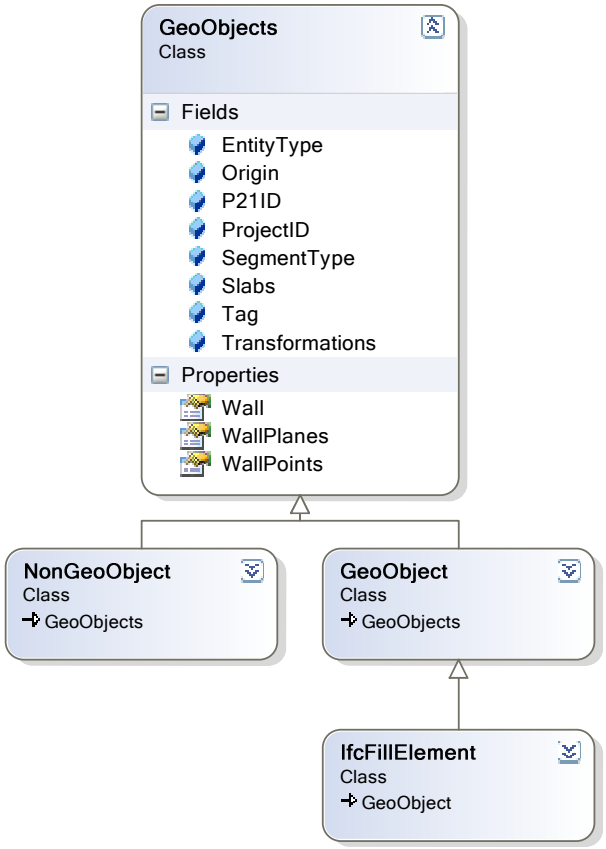


**Figure 6.7 `GeoObjects` Class Inheritance Diagram**

All `GeoObjects` contain the following:

- An `EntityType` delcaration, which holds the IFC entity name. (e. g. IfcSlab, IfcWallStandardCase, or IfcStairFlight)
- The `Origin` which defines the first point of an object and is used as the pivot point for doors and the center of rotation for `NongeoObjs`
- The building's unique `P21ID` number
- The building's unique `ProjectID` number

- A `SegmentType`, which defines the IfcShapeRepresentation of an object
- A list called `Slabs`, which contains:
    - The object's unique ID number which is called a `Tag`
    - A set of transformations, which when applied to the points contained within a single slab in the list `Slabs`, translate and rotate the points into the correct location relative to other objects in the building

The `NongeoObj` class contains objects whose geometric representation will not be extracted from IFC by IFCtoMAP. Rather, `NongeoObj` objects act as a object locator and file link to art assets generated in 3D modeling packages. `GeoObject`, on the other hand, contains data structures for holding IFC geometric representations for IfcRectangleProfileDef, IfcArbitraryClosedProfileDef and B-Rep - Boundary Representation objects. IfcFillElement inherits properties from `GeoObject` and is used to store data related to openings such as doorways and windows.

## 6.3 IFC File Loading

The first important class is the IFCLoader class, which can be seen in the class diagram labeled Figure 6.8. This class is an abstract class that contains a number of methods used to extract data from the IFC file format. IFCLoader is a generic loader class that contains methods for traversing an IFC file using the IFCsvr object. The IFCLoader class uses IFCsvr methods to identify the units of measure in an IFC document, determine the geometric representation of building elements, extract the

specific 3D geometric data for those elements and finally extract the transformation to be applied to those elements in order to place them in 3D space.

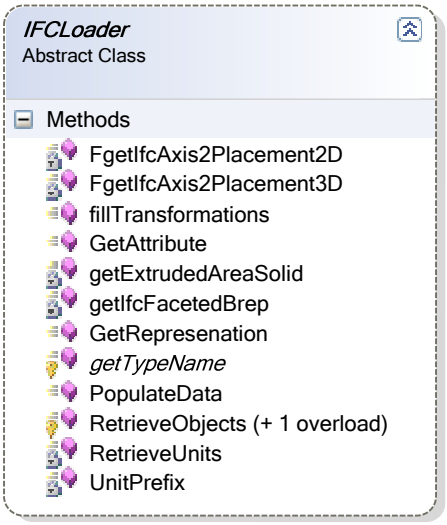


**Figure 6.8 File Reading: IFCLoader Class Diagram**

The static public method `PopulateData` is called by the sucessful selection of a .IFC file in the dialog generated by the GUI button labeled ‘Browse’ next to the ‘Select IFC Input File’ as seen in Figure 6.9 .


**Figure 6.9 IFCtoMAP Screenshot Showing the ‘Select IFC Input File’ Dialog**

The `PopulateData` function is passed a file name, a set of building elements to be extracted and a pointer to a textbox object. The file name includes the file path of the IFC file to be loaded. The pointer to a textbox points to an element in the GUI used to send text feedback about file loading progress, such as the number of IFC elements found in a particular file, as seen in Figure 6.10. The textbox is also used to display messages when errors occur during the file loading and parsing functions.

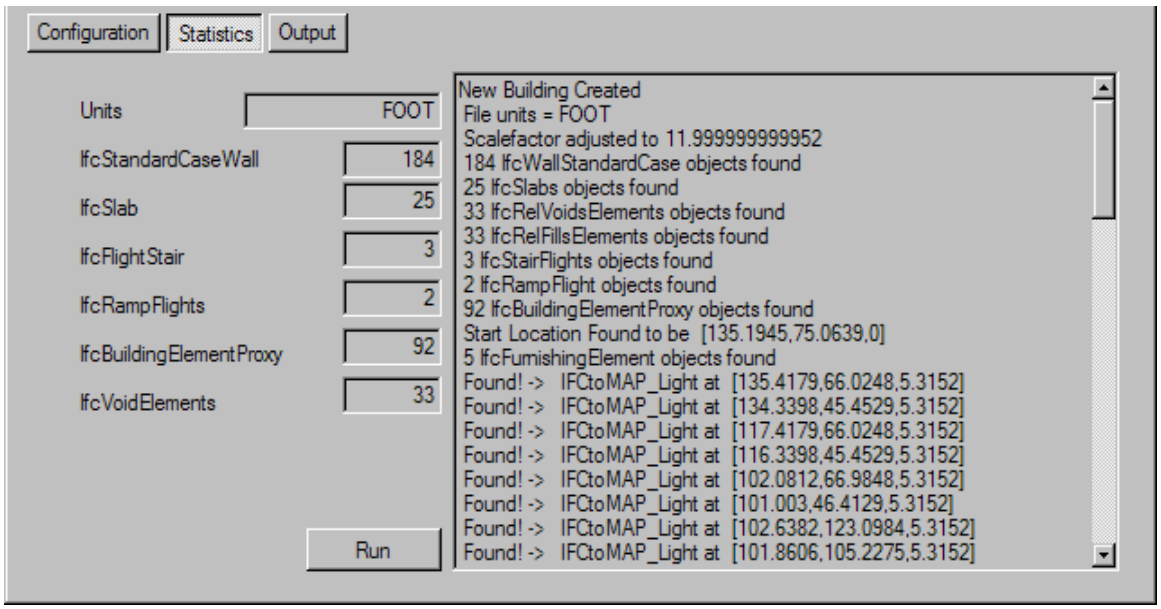


**Figure 6.10 User Feedback Dialog**

Figure 6.10 shows how the `PopulateData` method progresses. First, a new `Building` object is created. Then, the units of measure and a numeric scale factor are extracted and generated, followed by a succession of counted building elements.

Because the IFCLoader is an abstract class, there are no instances of the object. Figure 5.8 shows eight classes that inheret the methods from ICFLoader and are called by the static method PopulateData. Theses classes are generally named after the IFC building elements that they load.

One implementation of the IFCLoader, the protected virtual function `RetrieveObjects` accepts an `IFCsvr.Design` object, which contains the input file data, and a `Building` object in which to put data. Using the IFCsvr function `FindObject` all elements in the design can be found when provided a string with the object type name which coresponding to an IFC object type. The supported IFC object types / building elements are seen in Table 6.1.

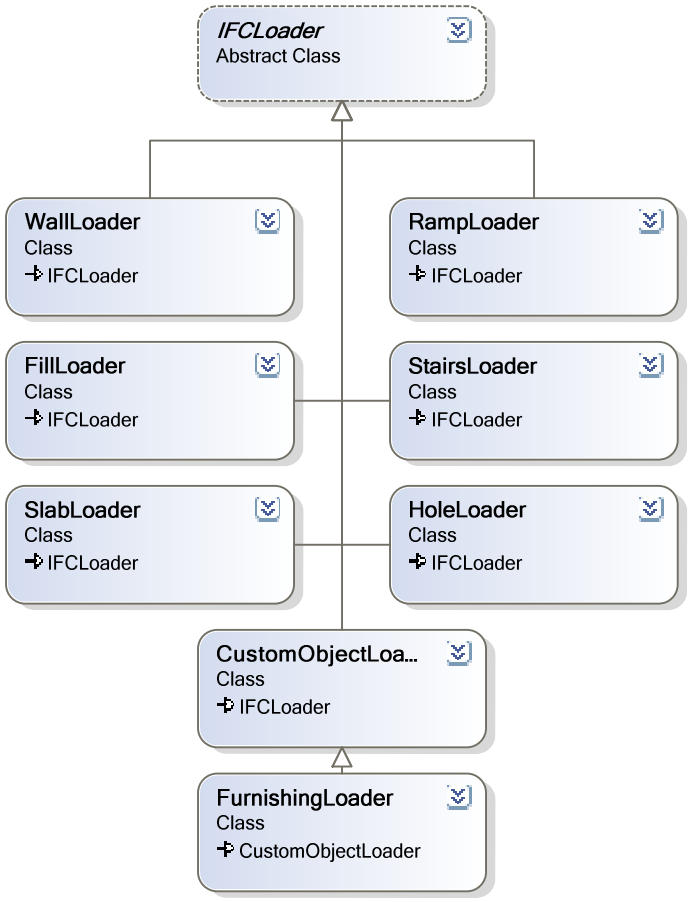


**Figure 6.11 File Reading: IFCLoader Class Implementation**

**Table 6.1 Supported IFC Object Types/Building Elements**

| *IFC Objects* | *Description* |
|---|---|
| IfcWalls | Used for all other occurrences of wall, particularly for walls with changing thickness along the wall path, or walls with a non-rectangular cross sections. |
| IfcWallsStandardCase | Used for all occurrences of walls that have a non-changing thickness along the wall path and where the thickness parameter can be fully described by a material layer set. These walls are always represented geometrically by a SweptSolid geometry, if a 3D geometric representation is assigned. |
| IfcSlab | A slab is a component of the construction that normally encloses a space vertically. The slab may provide the lower support (floor) or upper construction (roof slab) in any space in a building. A special type of slab is the landing, described as a floor section to which one or more stair flights or ramp flights connect. |
| IfcRamp | A vertical passageway which provides a human circulation link between one floor level and another floor level at a different elevation. It may include a landing as an intermediate floor slab. |
| IfcRampFlight | Inclined slab segment, normally providing a human circulation link between two landings, floors or slabs at different elevations. |
| IfcStair | A vertical passageway allowing occupants to walk (step) from one floor level to another floor level at a different elevation. It may include a landing as an intermediate floor slab. |
| IfcStairFlight | Assembly of building components in a single "run" of stair steps (not interrupted by a landing). The stair steps and any stringers are included in this object. A winder is regarded as part of a stair flight. |
| IfcWindows | A window consists of a lining and one or several panels. Properties concerning the lining and panel(s) are defined by the IfcWindowLiningProperties and the IfcWindowPanelProperties. |
| IfcRelVoidsElement | Objectified Relationship between an building element and one opening element that creates a void in the element. This relationship implies a Boolean Operation of subtraction for the geometric bodies of Element and Opening Element. |
| IfcRelFillsElement | Objectified relationship between an opening element and an building element that fills (or partially fills) the opening element. |
| IfcBuildingElementProxy | The IfcBuildingElementProxy is a proxy definition that provides the same functionality as an IfcBuildingElement, but without having a defined meaning of the special type of building element it represents. The building element proxy should be used to exchange special types of building elements, for which the current IFC Release does not yet provide a semantic definition. |
| IfcFurnishingElement | Generalization of all furniture related objects. |
| IfcConversionBasedUnit | A conversion based unit is a unit that is defined based on a measure with unit. |

The above list of supported elements may appear to be redundant as it includes both IfcStair and IfcStairFlight. This is not so however, as IFC specifies IfcStair as a single uninterrupted flight moving from one level of a building to the next, and IfcStairFlight could consist of two or more flights of stairs moving between levels of a building that may include a landing.

The IFCLoader function `RetrieveObjectProperties` accepts a set of IFCsvr `Entities` and a `Building` object. The result of the `FindObject` function, an `Entities` object contains a list of all records in the design of a particular type (e.g. IfcWall). `RetrieveObjectProperties` then loops through the list of entities, and for each entity creates a new `geoObject` and retrieves the requisite `ProjectID`, `P21ID`, `EntityType`, `SegmentType`, `Name`, `slabs`, `Transformations`, and `Tag`.

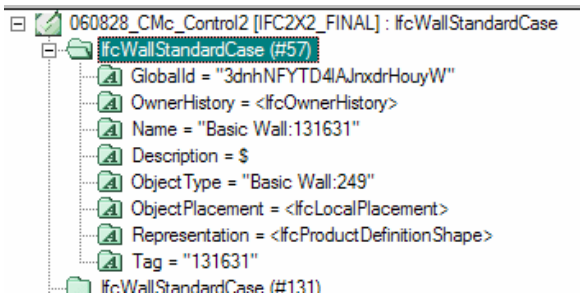


**Figure 6.12 IFCExplorer Tree View of a Common IfcWallStandardCase Entity**

The geometric representations defined in IFC Releases 2.0 and 2.x match closely to the ISO 10303-42:1994 STEP geometric definition. Objects, stored in IFC with a geometric representation have two attributes: *ObjectPlacement* and *Representation* as

seen in Figure 6.12. Both object placement and a geometric representation exist, and the `RetrieveObjectProperties` function then parses through each entity and calls the `fillTransformations` and `GetRepresenation`.

The `fillTransformations` function traverses the ObjectPlacement tree and extracts the location information from local to world origin. The IFC IfcLocalPlacement specifies the location of an object based on a coordinate space from two vectors, `Axis` and `RefDirection`, and an origin point, `Location`. The XYZ axis placement is calculated from `Axis` and `RefDirection`. `Axis` represents the direction of the local Z-axis as a 3D unit vector. If the `Axis` value is omitted, then it can be assumed to be [0, 0, 1] or the positive $Z_{axis}$. `RefDirection` represents a vector within the positive XZ plane; many times this is the $X_{axis}$ though it is not necessarily orthogonal to the `Axis`. If the `RefDirection` value is omitted, then it can be assumed to be [1, 0, 0] or the positive $X_{axis}$. Following the right hand rule, the cross product of these two unit vectors can be calculated by using the Equation 6.1 to create the third unit vector in 3d space orthogonal to the XZ plane, the $Y_{axis}$. This is shown graphically in Figure 6.13.

$$\overrightarrow{Y_{axis}} = \overrightarrow{Axis} \times \overrightarrow{RefDirection}$$

**Equation 6.1**

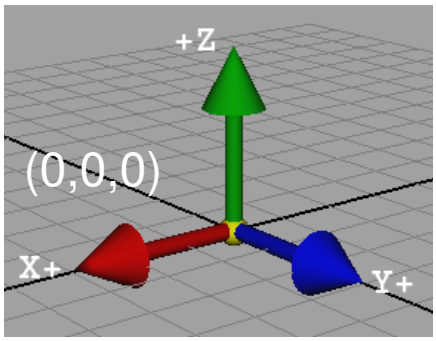


**Figure 6.13 Cross Product of Z and X Which Yields Y**

Because `RefDirection` is not necessarily the $X_{axis}$ it is calculated by crossing the newly created $Y_{axis}$ with the $Z_{axis}$ as shown in Equation 6.2.

$$\overrightarrow{X_{axis}} = \overrightarrow{Y_{Axis}} \times \overrightarrow{Axis}$$

**Equation 6.2**

The object's axis origin then is translated to the `Location` point, which becomes the new origin. In Figure 6.13 the point is not translated and remains at the position (0, 0, 0). As mentioned above, the IFC definition describes the position of objects as a tree of related transformations. A coordinate system matrix, $M_{CoordinateSystem}$, is built from each of the transformations as seen in Equation 6.3. The top three left rows contain the coordinate system axes listed in order, vertically. The `Location` point also known as the translation point, T, appears vertically in the last column. By convention, the bottom row contains the values (0, 0, 0, 1).

$$M_{CoordinateSystem} = \begin{bmatrix} X_{Axis_X} & X_{Axis_Y} & X_{Axis_Z} & T_x \\ Y_{Axis_X} & Y_{Axis_Y} & Y_{Axis_Z} & T_y \\ Z_{Axis_X} & Z_{Axis_Y} & Z_{Axis_Z} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**Equation 6.3**

In the following example a global coordinate system is built from the following values; `Axis` = [0, 0, 1]; `refDirection` = [1, 0, 0]; and Location = [0, 0, 0], then, from Equation 6.1 the Y-axis would be [0, 1, 0]. When composed into Equation 6.3 the result would be Equation 6.4.

$$\mathrm{M}_{Global} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**Equation 6.4**

Continuing the example, if there were a new local coordinate system into which an object were transformed with the following coordinate system built from the values, `Axis` = [1, 0, 0], `refDirection` = [0, 0, -1], and Location = [-7, 1, 6]. Then, using Equation 6.1 the Y-axis would be [0, 1, 0]. When composed into Equation 6.3 the result would be Equation 6.5.

$$M_{Local} = \begin{bmatrix} 0 & 0 & -1 & -7 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**Equation 6.5**

When transforming an object from the most local coordinate system into the most global coordinate system a matrix multiplication takes place. Each coordinate matrix is multiplied by the next more global coordinate system until a single transformation matrix exists. This matrix can be applied once to all points in an object to transfer them from the local coordinate system into the global coordinate system. This can be seen in Equation 6.6.

$$M_{TotalTransform} = M_{Local} * M_0 \ldots M_n * M_{Global}$$

**Equation 6.6**

When multiplying the matrix $M_{local}$ and $M_{global}$ in this example, the result is a -90 degree rotation about the Y-axis and a translation from the coordinates [0, 0, 0] to the location [-7, 1, 6] in the global coordinate system. This can be seen in Figure 6.14. Once all coordinate transformations have been composed into a single transformation $M_{TotalTransform}$ as seen in Equation 6.6, the $M_{TotalTransform}$ can be applied to all points in an object, resulting in the proper placement of the object relative to other objects in the IFC file.

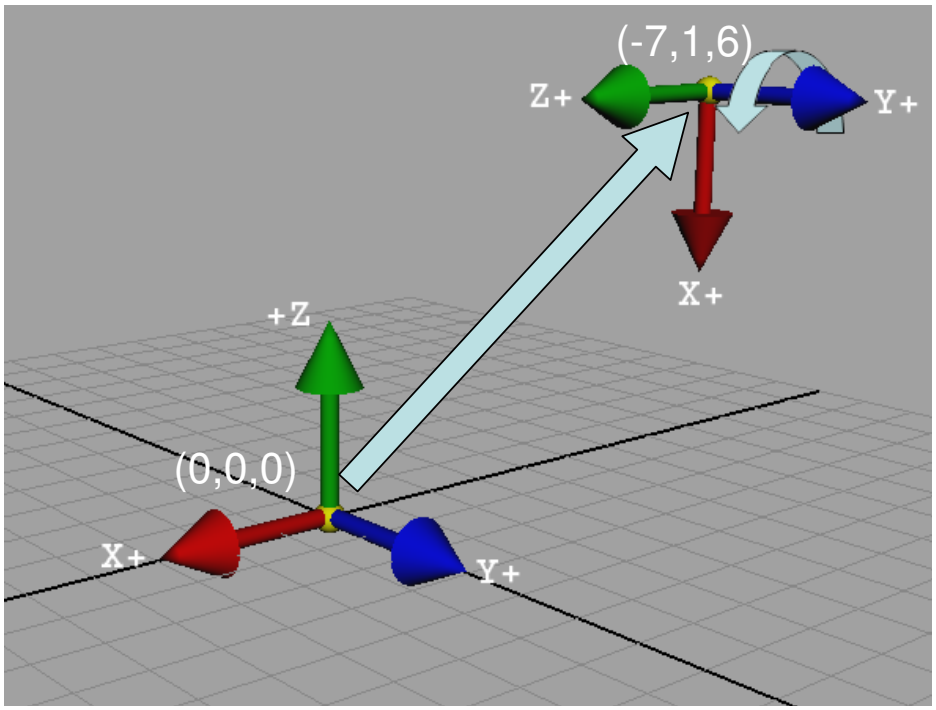


**Figure 6.14 Example of Local Relative Placement**

The next function relates to extracting an object's geometric representation. Similar to the method used to traverse the .ifc file in the `fillTransformations` function, the `GetRepresenation` also traverses the data structure to extract information related to an object's geometric representation. The geometric representation is found within an entity called IfcProductDefinitionShape by IFC. Figure 6.15 shows an example of an IfcProductDefinitionShape that contains two IfcShapeRepresentation(s). The first IfcShapeRepresentation is of type *Curve2D*; the second is of type *SweptSolid*. For the purposes of this discussion, the two dimensional representation seen as Curve2D representations of building elements are ignored in favor of the second SweptSolid 3D representation of the same elements. SweptSolid in this context represents an extruded

two-dimensional shape. The IFCLoader function `GetRepresenation` distinguishes between the following three 3D representation types:

- SweptSolid
- Clipping
- B-rep

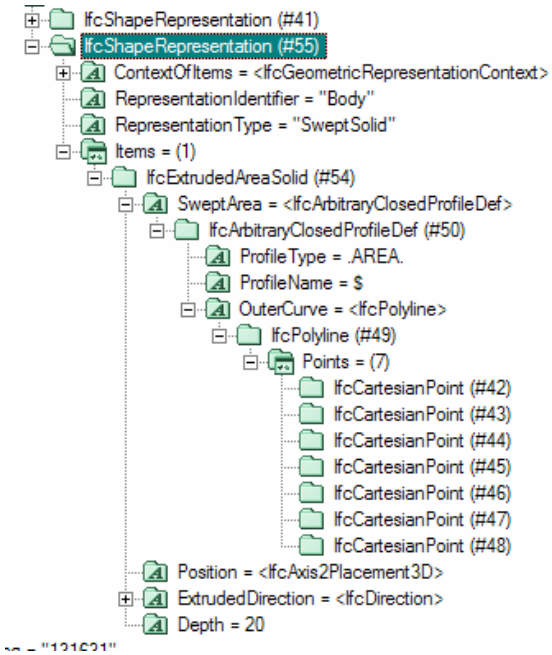


**Figure 6.15 IFCExplorer Tree View of IfcProductDefinitionShape**

SweptSolid objects consist of four elements: Position, ExtrudedDirection, Depth, and SweptArea. The position property is the most local coordinate system / transformation a SweptSolid can have and is added to the list of coordinate transformations kept in the `GeoObject`. The ExtrudedDirection, as it claims, is a vector in the direction of the extrusion. Commonly, walls extrude up, positive Z,

from the level of a building (e. g. First Level/Floor), and floors or slabs are extruded down, negative Z, from the level of a building. The Depth is the length of the extrusion in the direction of ExtrudedDirection. The last element of the SweptSolid is the SweptArea. There are more than four different possible values for SweptArea, ranging from complex parametric curved shapes to simple parametric rectangular shapes. The only two values of SweptArea supported by IFCtoMAP are IfcArbitraryClosedProfileDef and IfcRectangleProfileDef.

The first of the two SweptArea(s), IfcArbitraryClosedProfileDef consists of an IfcPolyline. Examining the example above in Figure 6.15, one can see an IfcPolyline consisting of a set of seven IfcCartesianPoints. The convention used by the IfcArbitraryClosedProfileDef is that the first point and the last point are the same. For simplicity, the IFCtoMAP internal data structures follow the same convention. The second of the two SweptArea(s), IfcRectangleProfileDef seen in Figure 6.16, consists of an additional IfcAxis2Placement2D position, an X-dimension and a Y-dimension. It is important to note that the IfcRectangleProfileDef definition of Position is an additional transformation added into the list of transformations in `GeoObject`(s).

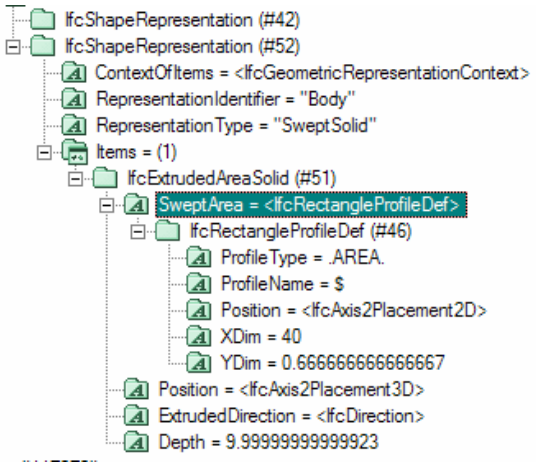


**Figure 6.16 IFCExplorer Tree View of IfcRectangleProfileDef**

The second type of 3D representation supported by the IFCLoader function `GetRepresenation` is Clipping. A Clipping geometric representation of an object is often the boolean addition, subtraction or difference between an extruded SweptArea and a second SweptArea or a plane. To simplify the .ifc reading process, the boolean additions, subtractions, and difference calculations are not performed. The `GetRepresenation` function only extracts the extruded SweptArea portion of the object and ignores the clipping elements. This results in a reuse of the SweptArea code while allowing for the addition of full support for Clipping object types to be added in the future.

The third geometric representation supported by `GetRepresenation` is Boundary Representation, referred to as B-rep, objects. The implementation of B-rep objects recognized by `GetRepresenation` is defined as a set of IfcClosedShell(s). Each of the IfcClosedShell consists of one or more IfcFace(s). In turn each IfcFace is

composed of four or more IfcFaceOuterBound ‘polygons,’ and each of the ‘polygons’ in an IfcFaceOuterBound is composed of three or more IfcCartesianPoints, as seen in Figure 6.17. To maintain the data integrity in the IFCtoMAP, internal data structures mimic the IFC data structures in that B-Rep objects are stored as a linked list of related IfcClosedShell(s) linked to a list of polygon faces consisting of points.

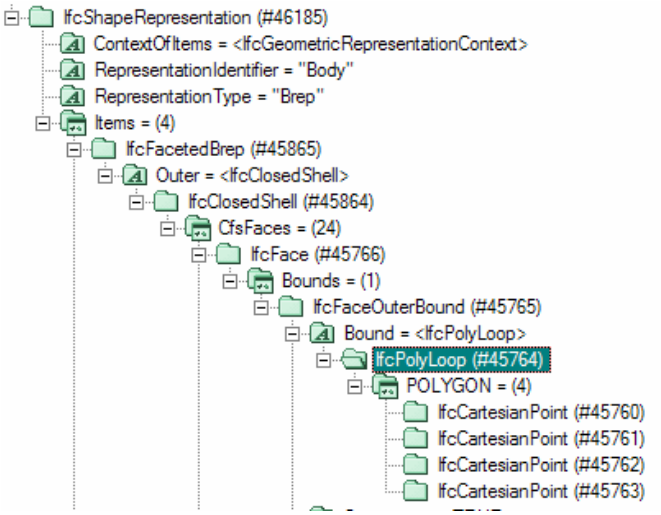


**Figure 6.17 IFCExplorer Tree View of a Boundary Representation Object**

In addition to extracting objects with geometry or `geoObjects` from IFC, IFCLoader and its child classes also extract what have been termed as `NongeoObjects`. The term `NongeoObjects` comes from the fact that the IFCtoMAP program uses information about those objects from IFC without extracting the objects’ geometry. The only loader that creates `NongeoObjects` is CustomObjectLoader and its child class IfcFurnatureLoader. These are special cases that

share the IFCLoader functions for determing position – rotation and displacement transformations, but do not use the geometry from IFC.

The `NongeoObjects` class objects provide a way to use Revit and IFC for Doom 3 specific .map information within the bounds of the IFC conventions and are not an officially recognized method of sharing IFC information.

The method of creating `NongeoObjects` was decided as a result of having chosen Revit as the IFC data provider. Although Revit provides IFC2x2 Final support for basic building features such as walls, windows, doors, floors and ceilings, Revit does not allow users to specifiy new IFC export options such as IfcPropertySet(s). IfcPropertySet is the IFC's way of expanding the data dictionary for custom information exchange. An example of an IfcPropertySet object would be exporting the color of a wall from Revit. If the IFC standard did not have a definition of how to color walls, other IFC tools would not necessarily implement the wall color and would ignore the Revit IfcPropertySet related to wall color. IfcPropertySet allows for IFC software developers to encode and exchange non-standard information in .IFC files even if the IFC standard does not currently support the feature. Often, what begins as an IfcPropertySet may eventually be incorporated into the IFC standard and receive an IFC name and definition. In the example above, if the wall color was eventually encorporated, it may be given the name IfcWallStandardCaseColor or something similar.

Revit does implement all IFC building elements. All objects in Revit are members of Families. Frequently used Families, such as walls, windows, doors, floors and ceilings have support for export to IFC. Revit users are allowed to create custom Families, based

on the frequently used Families or may create a Family from scratch. Revit does not support linking a custom Family to a specific IFC element. For example, the start location of viewer inside the .map file format requires a special entity classname called `info_player_start`. In order to place this entity, a custom Family was created inside of Revit with the prefix IFCtoMAP. When exported to IFC, the custom Family is exported as an IfcBuildingProxyObject. The CustomObjectLoader parses all of the IfcBuildingProxyObject(s) and whichever have the name with prefix IFCtoMAP are handled differently than those objects which are fully supported by IFC.

The FurnishingLoader inherits properties of the CustomObjectLoader. As previously discussed, .map uses models generated in 3D art packages to describe complete objects that fill the space in the virtual environment with elements such as furniture. The FurnishingLoader class parses all IfcFurnishingElement(s) and those with a name string with the 'IFCtoMAP' prefix, similar to the example seen in Figure 6.18, are read into the internal data structures as a `NongeoObject`. FurnishingLoader extracts both the ObjectPlacement and ObjectType data. The ObjectPlacement is the same transformation and displacement information used by all IFCLoader classes. The ObjectType, however, specifies a relative path within the Doom 3 game structure and points to a 3D model that can be linked in the .map file. As an example, Figure 6.18 shows an IfcFurnishingElement object, a chair, which when linked to the game would be located in a file on the hard drive as: `C:\Doom 3\base\models\IFCtoMAP\Chair.lwo`. Note that the geometric representation stored in IFC is not converted into primatives because most of these objects contain 3D

shapes with concave parts and are better defined as models from 3D art packages, in this example a Lightwave .LWO file. Breaking up a 3D shape into a set of convex 3D shapes is outside the scope of this project.

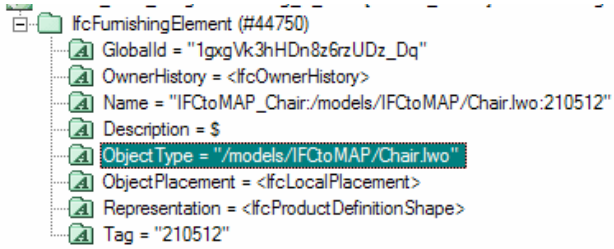


**Figure 6.18 IFCExplorer Tree View of a IfcFurnishingElement Object**

## 6.4 Conversion from IFC to .MAP

Having loaded the IFC file into the internal data structures, the next task is to convert the geometric representions into a format understood by the Doom 3 game engine. This section details the conversion method from the three supported IFC geometric representations IfcRectangularProfileDef, IfcArbitraryClosedProfileDef, and Boundary Representation into a geometric representation suitable for writing to the .map brushDef3 format.

The first IFC geometric representation is IfcRectangularProfileDef. One might recall from Section 6.3 that IfcRectangularProfileDef(s) consists of three basic components: an X-dimension, a Y-Dimension and an ExtrusionDepth. Those components will be referred to as X-dim, Y-dim and Depth respectively as seen in Table 6.2.

**Table 6.2 IFC Representation of IfcRectangularProfileDef**

| Name | Value |
|---|---|
| XDim | 40 |
| YDim | 0.66 |
| Depth | 9.99 |

In the example in Table 6.2, we will assume for this discussion that the IFC file uses feet as the units of measure. The IfcRectangularProfileDef in IFC is .66 feet or 8 inches wide, 40 feet long and about 10 feet tall. The parametric form of the IfcRectangularProfileDef is then converted into an internal representation as a set of four co-planer counterclockwise ordered points, using the formulas seen in Table 6.3.

**Table 6.3 Intermediate Representation of IfcRectangularProfileDef**

| Point | X | Y | X | Y |
|---|---|---|---|---|
| 1a | 0 | .5 * YDim | 0 | -0.33 |
| 2 | 0 | -0.5 * YDim | 0 | 0.33 |
| 3 | XDim | -0.5 * YDim | 40 | 0.33 |
| 4 | XDim | .5 * YDim | 40 | -0.33 |
| 1b | 0 | .5 * YDim | 0 | -0.33 |

It is important to note that the first point, 1a, and the last point, 1b, are the same in Table 6.3. This and ordering the points in a counterclockwise manner are both conventions of IFC. These conventions allow the internal data structures to maintain uniformity that facilitates reuse of code functionality. IFC defines

IfcRectangularProfileDef as being positioned with the left-most points centered about the x-axis. When plotted, the points result in a closed polygon loop as seen in Figure 6.19.

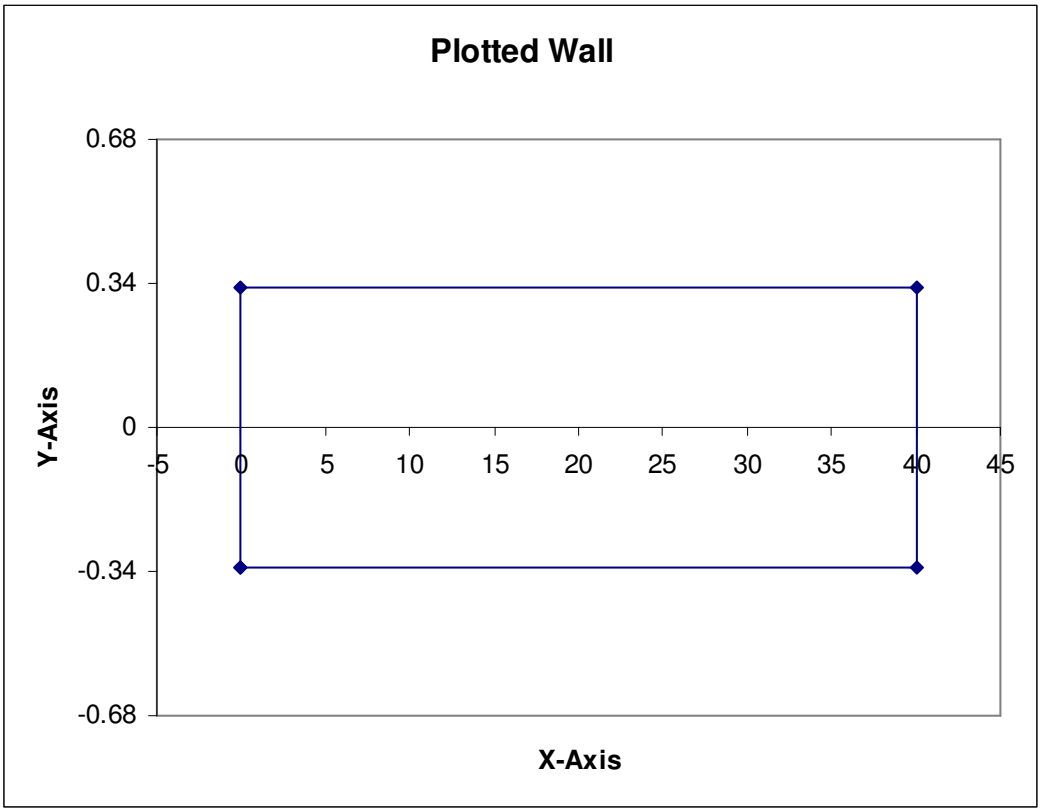


**Figure 6.19 Closed Polygon Loop Representation of a Wall**

Although Figure 6.19 is a two dimensional figure, each of the lines that define the rectangle are representative of planes that define the sides of a rectangular cuboid. The next step in order to move the IfcRectangularProfileDef from this internal intermediate representation into the .MAP file format is to generate a set of the plane unit normals and distances from the origin to the closet point on the plane.

To accomplish this, the program generates vectors using two points at a time from the set of points seen in Figure 6.19. Because the points are stored in a counterclockwise order, the algorithms in the IFCtoMAP program traverse the points in a counterclockwise direction. The order the points are traversed is important for maintaining the direction of the normals of the plane facing away from the center of the object and towards a viewer. The vectors are constructed using the cross product formula (see in Equation 6.7) where $P_0$ is the first point and $P_1$ is the second point in counterclockwise order around the shape.

$$\overrightarrow{V_N} = P_1 \times P_0$$

**Equation 6.7**

The result of the cross product, the vector $\overrightarrow{V_N}$, is then normalized into a unit vector. The magnitude $\left|\overrightarrow{V_N}\right|$ of a vector is $\sqrt{x^2 + y^2} = Length$. This equation is said to be normalized if $\sqrt{x^2 + y^2} = 1$ and thus $\left|\overrightarrow{V_N}\right|$ is considered a unit normal vector after normalization.

The next step is to find the shortest distance between the plane, described by the vector $\overrightarrow{V_N}$, and the origin. Distance computations are fundamental in computer graphics and computational geometry and there are well-known formulas for them. An illustration of the vector projection can be seen in Figure 6.20 where $\vec{w}$ is a vector between point $P_0$

and the origin *P*, and *P(b)* is the closest point between the origin and the plane defined by the vector $\overrightarrow{V_N}$ .

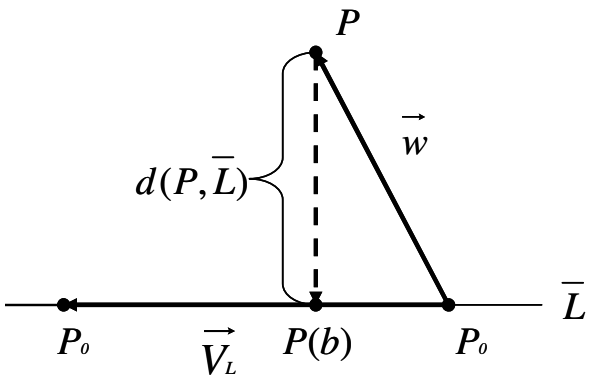


**Figure 6.20 Vector Projection of Point *P* onto Vector $\overrightarrow{V_L}$**

The distance $d(P,L)$ seen in Figure 6.20 can be calculated by first finding $u = proj_{\overrightarrow{V_L}}\vec{w}$ or the vector projection of the vector $\vec{w}$ onto $\overrightarrow{V_L}$ . $\vec{w}$, being the vector constructed from $P_0$ to the origin, *P*, using the Equation 6.8; and $\overrightarrow{V_L}$ being the vector constructed from $P_0$ to $P_1$, using the calculation seen in Equation 6.9. Having solved for $\overrightarrow{V_L}$ and $\vec{w}$, the shortest distance is then calculated using Equation 6.10.

$$\vec{w} = P_{(0,0)} - P_1$$

**Equation 6.8**

$$\overrightarrow{V_L} = P_1 - P_0$$

**Equation 6.9**

$$u = proj_{\overrightarrow{V_L}} \vec{w} = \frac{\overrightarrow{V_L} \bullet \vec{w}}{\left|V_L\right|^2}$$

**Equation 6.10**

$$d\left(P, \overrightarrow{L}\right) = \left|\overrightarrow{V_L} + \vec{u} * \vec{w}\right|$$

**Equation 6.11**

The result of the above calculations seen in Equation 6.7, $\overrightarrow{V_N}$, and Equation 6.8, $\vec{w}$, as used in Equation 6.10, produce the needed unit vector in the direction of the normal of a plane along with the distance from *P,* the origin, to *P(b)*. This unit normal vector represents the first three variables in the description of a plane within a brushDef3 object. The magnitude of the distance from plane to the origin is, however, a positive number as a result of the magnitude equation $\sqrt{x^2 + y^2} = Length$. Therefore the plane described by the normal exists either behind, or in front of the point of origin as *P(b)* or *P(b)'*. See Figure 6.21. The location of *P(b)* can be determined by evaluating the formula of a line, Equation 6.12, and solving for *d*. The sign of *d* then becomes the third number in the first set of four used in the brushDef3's description of a plane.

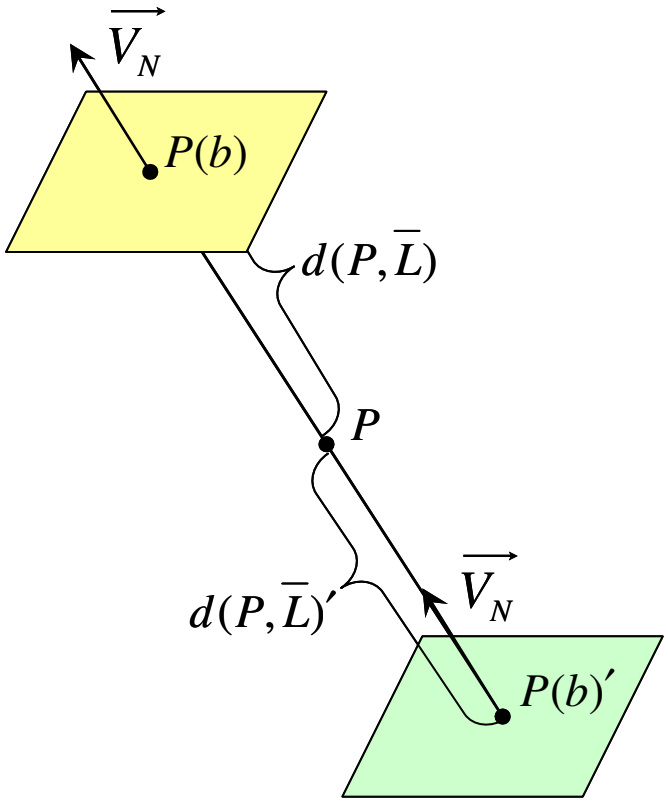


**Figure 6.21 Determining the Sign of Distance**

$$P = P(b) + d * \overrightarrow{V_N}$$

**Equation 6.12**

This method is then repeated for each set of points until the entire set of points have been traversed. Then the bottom and top of the IfcRectangularProfileDef are inserted as unit normal $\left|\overrightarrow{V_N}\right|$ [0, 0, -1] at distance 0 from the origin and unit normal $\left|\overrightarrow{V_N}\right|$ = [0, 0, 1] at a distance of the extrusion depth respectively. Note that many of the Figures in this subsection are two dimensional, but this does not change calculation equations because

the vector calculations can be extended to the third dimension by using (X, Y, Z) components.

An IfcRectangularProfileDef, by the nature of a rectangle, is inherently convex. IfcArbitraryClosedProfileDef objects define any arbitrary closed profile of points that are not necessarily convex. To ensure that the calculations above will work with IfcArbitraryClosedProfileDef objects, a set of pre-processing functions, contained within the `ConstrainTriangulation` class, when applied to the closed loop points, break the loop into a set of triangular loops. (To understand the necessity for this conversion to triangles, the reader should review section 5.2 Doom Considerations.) The purpose of the `ConstrainTriangulation` class is to eliminate the possibility of concave objects being sent to the MapWriter class by IfcArbitraryClosedProfileDef objects. The set of triangular loops, when viewed as a complete set, generates the same shape as the original IfcArbitraryClosedProfileDef.

The calculation method used is called a Constrained Delaunay Triangulation (CDT)[59]. The known constraints on the points are the counterclockwise order and their closed loop nature. Data points on the IfcArbitraryClosedProfileDef traverse as a set of three consecutive points. The algorithm used to triangulate the IfcArbitraryClosedProfileDef into a set of triangles appears in detail below. To illustrate the CDT used by the IFCtoMAP software, one can consider a concave shape such as the one seen numerically in and graphically in Figure 6.22. The Figures for this discussion are in the Appendices labeled as Constrained Triangulation Example Figures 1-14.

**Table 6.4 Example Constrained Triangulation Table of Points**

| Point # | X | Y | Z |
|---|---|---|---|
| 0 | 46.25 | 36.72 | 0 |
| 1 | 54.75 | 36.72 | 0 |
| 2 | 54.75 | 43.89 | 0 |
| 3 | 45.92 | 43.89 | 0 |
| 4 | 25.59 | 43.89 | 0 |
| 5 | 25.59 | 23.89 | 0 |
| 6 | 10.25 | 23.89 | 0 |
| 7 | 10.25 | 28.39 | 0 |
| 8 | 5.59 | 28.39 | 0 |
| 9 | 5.59 | 23.56 | 0 |
| 10 | 5.59 | 3.22 | 0 |
| 11 | 46.25 | 3.22 | 0 |
| 12 | 46.25 | 36.72 | 0 |

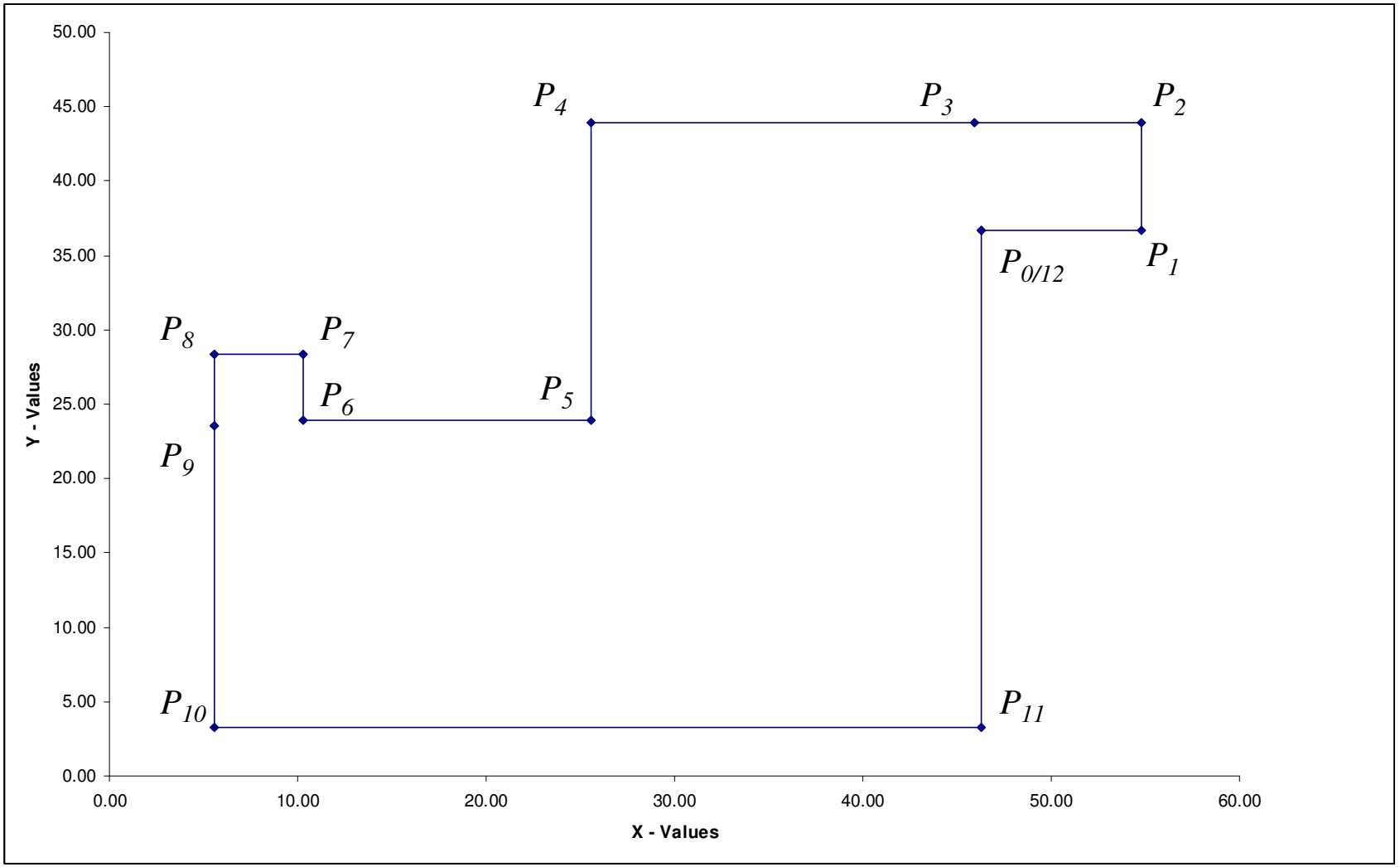


**Figure 6.22 Constrained Triangulation Example**

The shape seen in Figure 6.22 is the same shape seen in the Appendices as Figures 1-14. Note that point 0, $P_0$, and point 12, $P_{12}$, are the same and all points are counterclockwise in order, which follows the convention first seen in Table 6.3 Intermediate Representation of IfcRectangularProfileDef. The following are steps of the algorithm in triangulating Figure 6.22.

1. The first step selects a pivot point. The first pivot point chosen will be the first point in the set point $P_0$.
2. The next step creates a test edge between points $P_0$ and $P_2$. This results in a triangle consisting of points $P_0$, $P_1$, and $P_2$ as seen in Figure 1.
3. After this test edge has been created, checks are made that determine whether the points used to generate the test triangle are ordered in a counterclockwise manor. The sign of the determinate of a matrix representing the points, as in Equation 6.13, reveals the orientation of the points. If the determinate is positive the points are clockwise; if the result is negative the points are ordered counterclockwise. In this case the triangle is counterclockwise in orientation.

$$Det = \begin{vmatrix} P_{ox} & P_{oy} & 1 \\ P_{1x} & P_{1y} & 1 \\ P_{2x} & P_{2y} & 1 \end{vmatrix} = P_{0x} * P_{1y} + P_{1x} * P_{2y} + P_{2x} * P_{0y} - P_{0x} * P_{2y} - P_{1x} * P_{0y} - P_{2x} * P_{1y}$$

**Equation 6.13**

4. Next, test the edge created by $P_0$ and $P_2$ against all other edges in the IfcArbitraryClosedProfileDef and any triangles that have been removed from it

during the course of triangulation. The algorithm used to perform the intersection test between two line segments requires the creation of a vector *r* orthogonal to the original line segment, in this case $P_0$ and $P_2$, and the test line segment, $Tp_0$ and $Tp_2$, as seen in the Figure 6.23 and Equation 6.14.

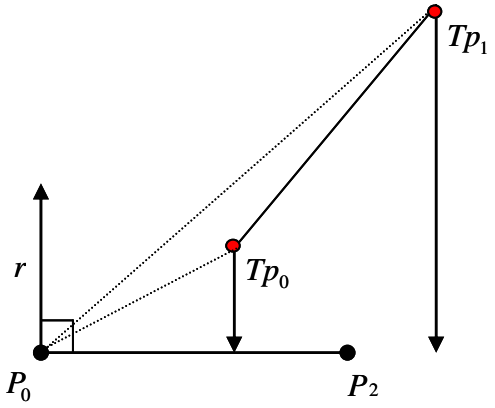


**Figure 6.23 Line Intersection Test**

$$\vec{r} = \left(P_0 - P_2\right)^{\perp}$$

**Equation 6.14**

The line segment intersection test checks to see if the interval created by Equation 6.15 spans 0. If the interval does span 0, then the lines intersect. Otherwise, they do not intersect. If the edge created by $P_0$ and $P_2$ does not intersect any of the object's other edges then create a new related IfcArbitraryClosedProfileDef object from points $P_0$, $P_1$, and $P_2$, and remove point $P_2$ from the original set of points.

$$\left( \frac{(Tp_1 - P_0) \bullet r}{|r|}, \frac{(Tp_0 - P_0) \bullet r}{|r|} \right)$$

**Equation 6.15**

5. After $P_2$ has been removed, the algorithm continues with the same result as seen in Constrained Triangulation Example 2-4.
6. However, in Constrained Triangulation Example 5 the test edge created by points $P_0$ and $P_6$ crosses the edge between $P_4$ and $P_5$, and the resulting test triangle is also clockwise in orientation. In this case, the algorithm will shift the pivot point from $P_0$ to $P_5$ and begin again as seen in Constrained Triangulation Example 6.
7. After changing the pivot point to $P_5$ in Constrained Triangulation Example 6, the algorithm again fails as the triangle created by $P_5$, $P_6$ and $P_7$ is not counterclockwise in orientation. In this case, the algorithm will again shift the pivot point from $P_5$ to $P_6$ and begin again as seen in Constrained Triangulation Example 6.
8. After the pivot has changed to $P_6$, the algorithm continues and breaks off four more triangles as seen in Constrained Triangulation Example 7-10.
9. In Constrained Triangulation Example 11, a test edge between $P_{0/12}$ to $P_6$ intersects with the edge created by $P_4$ and $P_5$.
10. The pivot point is then changed to $P_{11}$, and the edge created by $P_{11}$ and $P_5$ is tested, yielding another triangle as see in Constrained Triangulation Example 12.
11. Finally, an edged is created and tested between $P_{11}$ and $P_6$ as seen in Constrained Triangulation Example 13 which yields the last triangle.

12. The resulting shape seen in Constrained Triangulation Example 14 consists of 12 triangles which when seen as a set compose the original shape. When finally extruded, the resulting shape seen in Constrained Triangulation Example14 will look like the Figure 6.24.

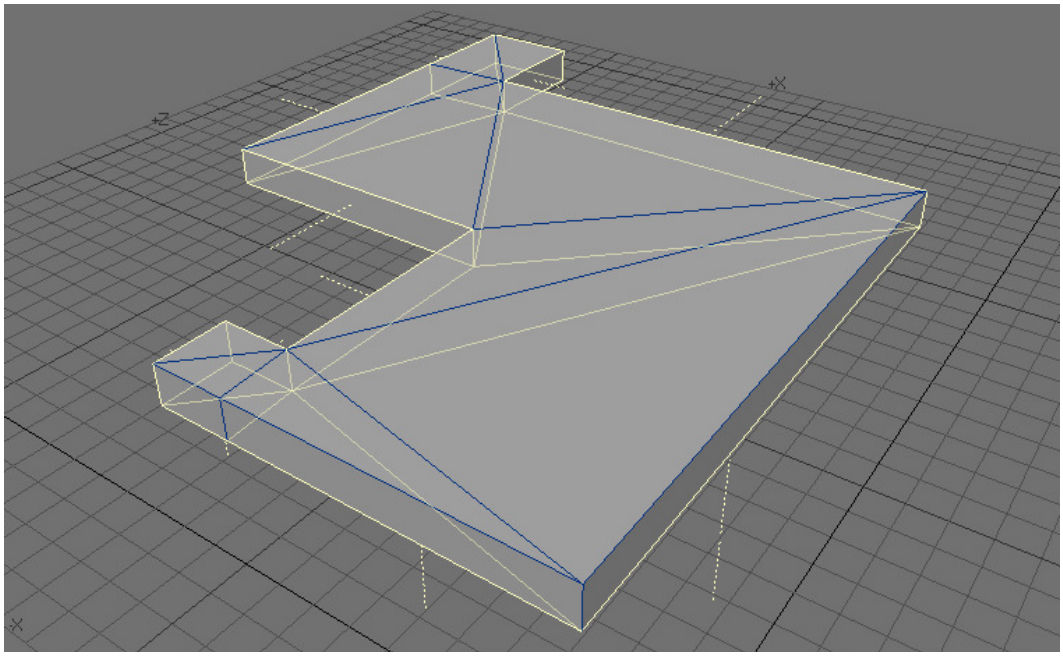

**Figure 6.24 Extruded CDT IfcArbitraryClosedProfileDef**

The implemented CDT algorithm discussed does not solve all possible cases of IfcArbitraryClosedProfileDef shapes, but does offer a solution that includes a number of shapes seen in building layouts.

Having covered both IfcRectangularProfileDef and IfcArbitraryClosedProfileDef shapes, one must also consider special cases. These include holes in floors, such as shafts, and openings in walls, such as doors and windows.

The first special cases considered are openings in floors; these are often seen in floors where there are stairwells or elevator shafts. In Figure 6.25, there are three closed polylines labeled as Exterior Edge, Hole A, and Hole B. The blue polyline labeled Exterior Edge represents an IfcArbitraryClosedProfileDef of either a floor or ceiling. Each of the other three shapes are holes, or IfcVoidElements, in the surface. Hole A contains two edges that intersect with the Exterior Edge. IFCtoMAP does not support holes similar to Hole A. IFCtoMAP does support all holes that exist within the bounds of an IfcArbitraryClosedProfileDef.

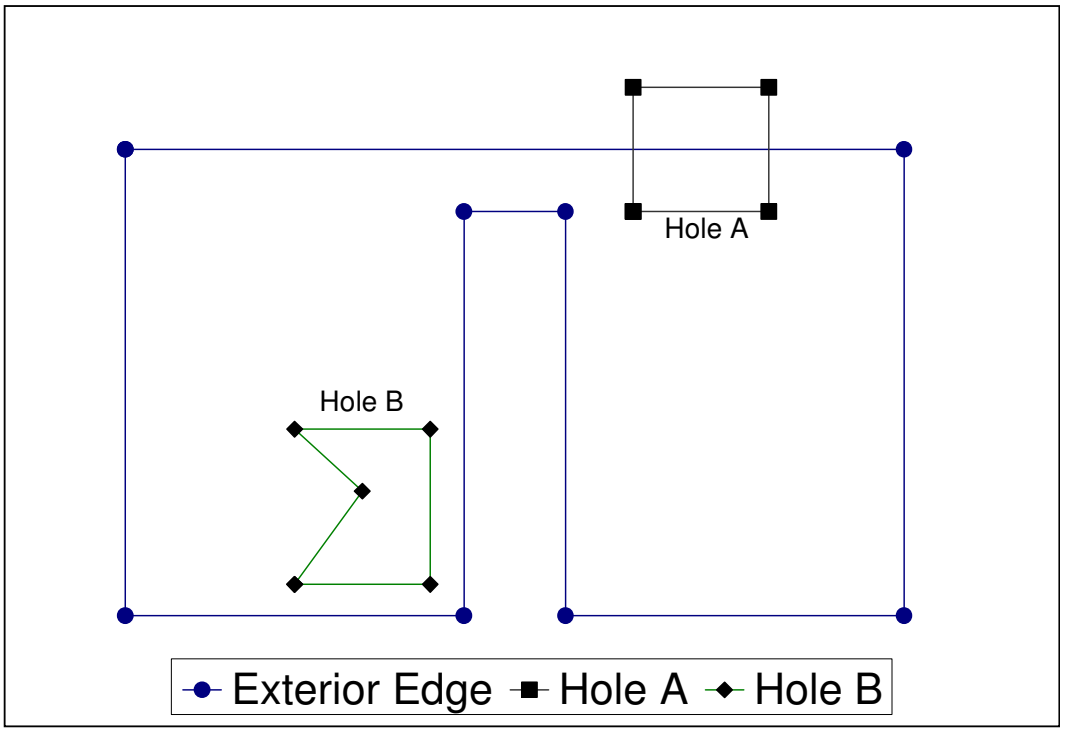


**Figure 6.25 Example of Two Possible Hole Types**

Figure 6.26 shows the names and locations of points on both the Exterior Edge and Hole B. The exterior edge points, 0-8, and the Hole B points, $B_0$-$B_5$, both follow the standard convention: first point and last point are the same, and the points organized in a counterclockwise manor as indicated by the arrows.

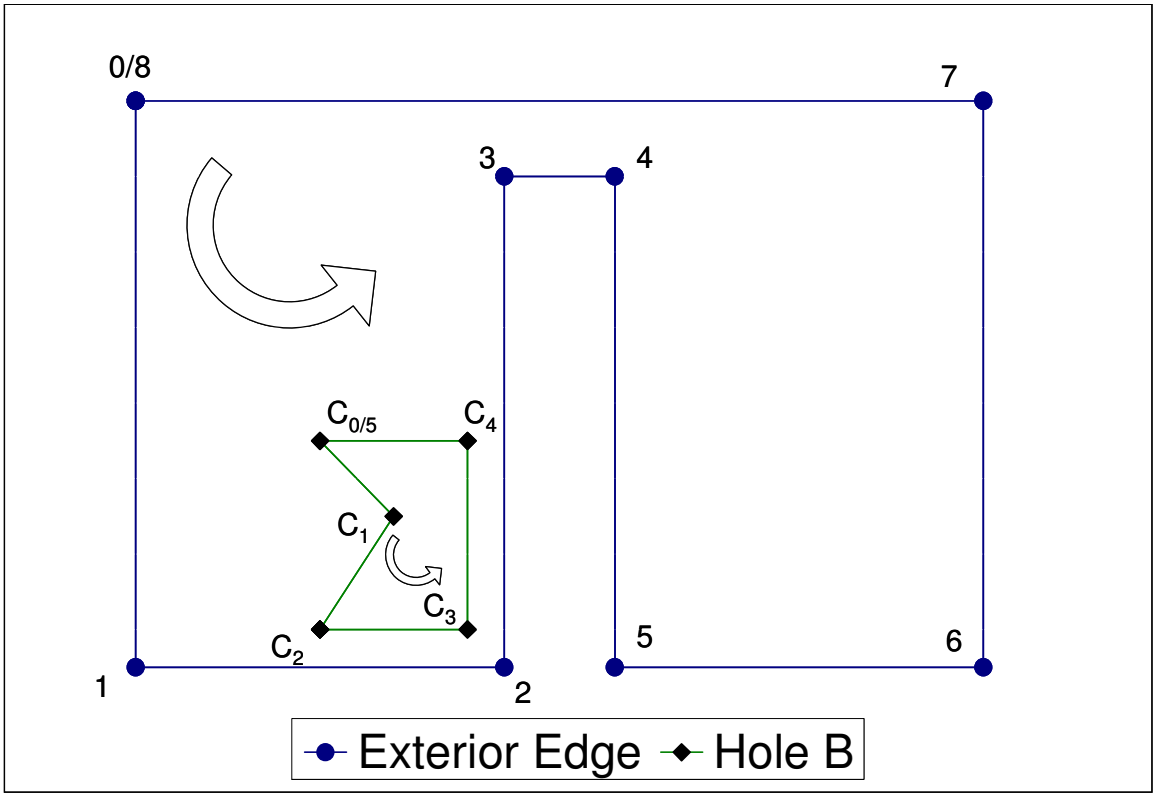


**Figure 6.26 Labeled Exterior Edge and Hole Points**

Because IfcArbitraryClosedProfileDef objects must triangulate, the solution selected for handling holes follows these steps:

1. Create a new edge between the first point, 1, on the Exterior and the first point on the Hole, $B_0$.

2. Check the new edge verses all other edges in both the set of exterior points and the set of hole points. If the new edge intersects any other edges, start over with the second point on the exterior. If the new edge does not intersect any other edges on the exterior or the hole, continue.
3. Reverse the orientation, that is the order, of the hole points from $B_0$, $B_1$, $B_2$, $B_3$, $B_4$, $B_5$ to $B_5$, $B_4$, $B_3$, $B_2$, $B_1$, $B_0$.
4. Then, inject those points into the list of exterior edge points as seen in Figure 6.27:

   Exterior Edge Points Before: 0, 1, 2, 3, 4, 5, 6, 7, 8

   Exterior Edge Points After: 0, $B_5$, $B_4$, $B_3$, $B_2$, $B_1$, $B_0$, 1, 2, 3, 4, 5, 6, 7, 8

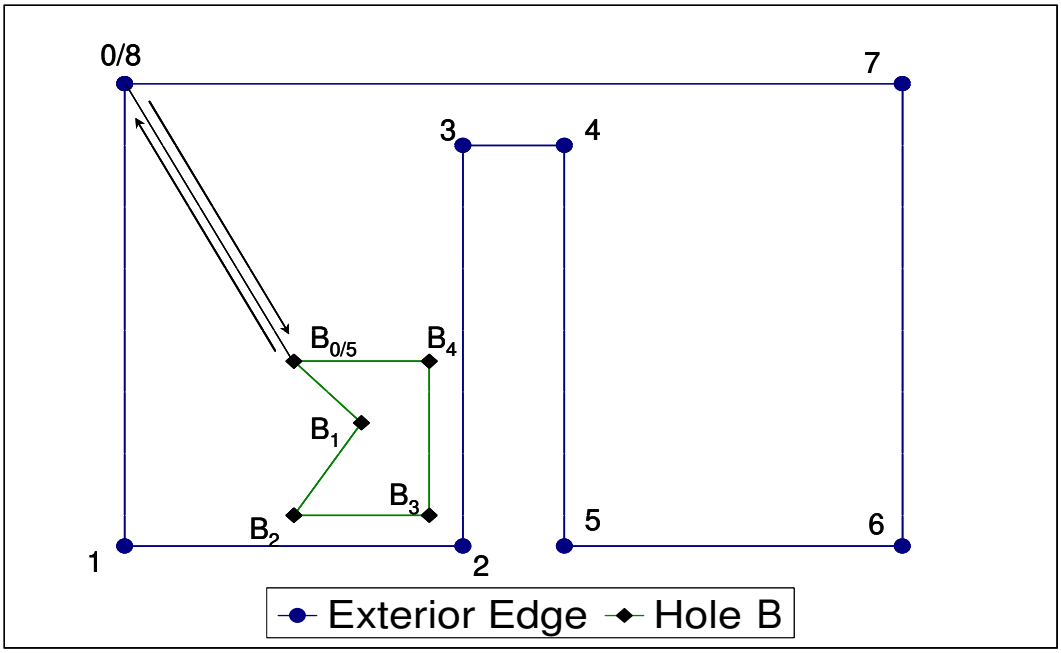


**Figure 6.27 Example of Injecting a Hole into an Exterior Edge**

The result is a single set of points defining an IfcArbitraryClosedProfileDef, which are counterclockwise in order with the same first and last points that can be sent to the CDT function for triangulation. This process of adding holes can be repeated for as many holes as there are in a shape. This solution is not optimal and can inject holes into exterior edges that cause the CDT function to fail. However, the solution works for simple shapes seen in typical office building geometry.

The next types of opening considered were openings in walls that consist of empty openings, doors and windows. When imported from IfcWalls, these are either IfcRectangularProfileDef or IfcArbitraryClosedProfileDef shapes. IfcRectangularProfileDef shapes have only four points that define the entire shape, where IfcArbitraryClosedProfileDef could have more. When creating an opening using an IfcRelVoidElements in a wall, the following method is used:

1. First the total number of points that define the wall is reduced to the smallest set of points possible, often four points as seen in Figure 6.28.

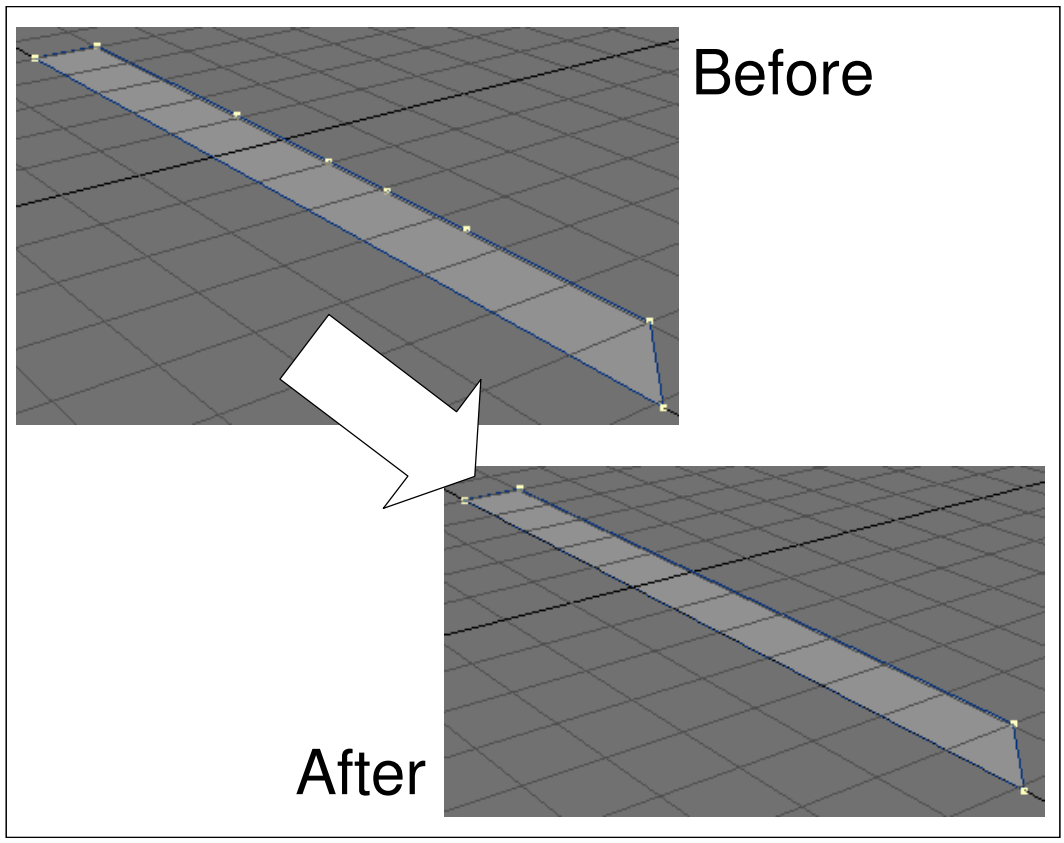


**Figure 6.28 Minimizing Wall Points**

This is done by traversing the set of wall points in a counter clockwise order and calculating the angle between three points at a time. If the angle between the points is equal to 0 or 180, then the middle of the three points is removed. Otherwise all points are kept, and the next three sets of points are checked. From an XY perspective, the wall looks as shown in Figure 6.29. In this example, the points would be listed as follows: 0, 1, 2, 3, 4.

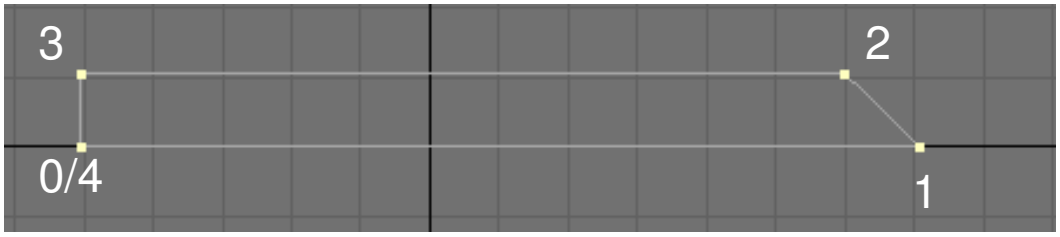


**Figure 6.29 XY View of Minimized Wall Points**

2. Figure 6.30 shows an example of IfcRelVoidElement(s), such as doors and windows which are defined in the XZ or YZ plane, where as IfcRectangularProfileDef and IfcArbitraryClosedProfileDef are most often defined in the XY plane. The next step is to identify the lowest two points on the IfcRelVoidElement object and project them into the XY plane. This is done by only using the XY components of the points and ignoring the Z component.

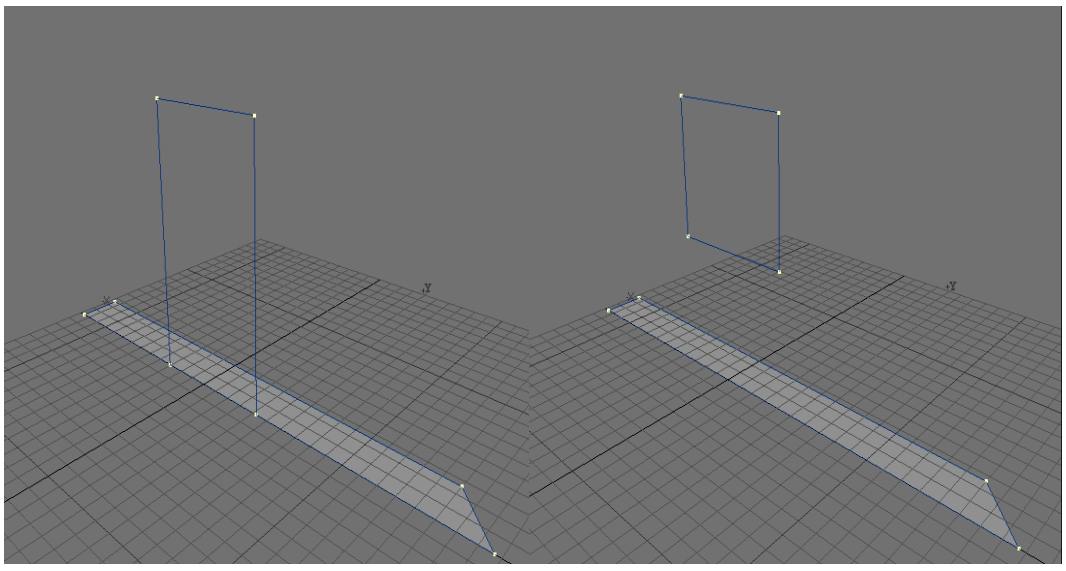

**Figure 6.30 Example IfcRelVoidElement(s) such as Doors and Windows**

3. The projections of each of the two lowest points, $H_0$ and $H_1$, when added into the set of points that define the wall, maintain the ordered counterclockwise orientation of points as seen in Figure 6.31. The resulting list of points: 0, $H_0$, $H_1$, 1, 2, 3, 4.

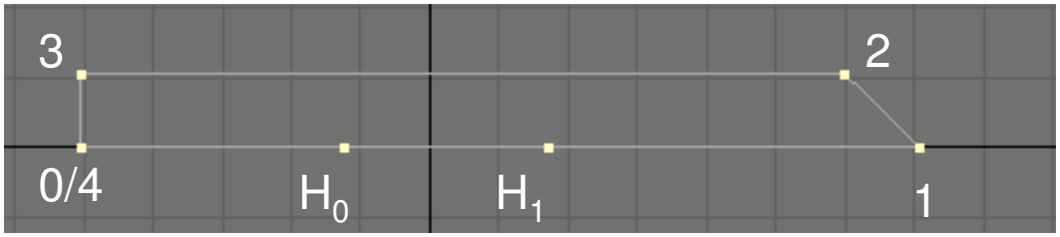


**Figure 6.31 Projection of Two Lowest Points of IfcRelVoidElement on to Wall**

4. The other side of the hole in the wall is then calculated by projecting both $H_0$ and $H_1$ onto the line segment created by points 2 and 3. The result is the creation of points $H_2$ and $H_3$. $H_2$ and $H_3$ when added into the point list, result in the following order 0, $H_0$, $H_1$, 1, 2, $H_3$, $H_2$ 3, 4. The result can be seen in Figure 6.32.

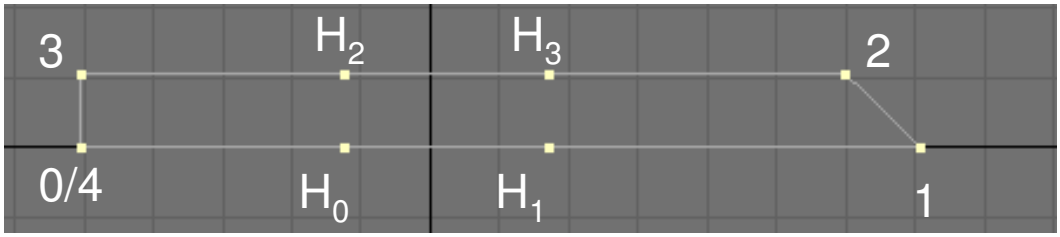


**Figure 6.32 Projection of $H_0$ and $H_1$ on Line Segment 2-3**

5. The shape then splits into three parts, as shown in Figure 6.33. The two segments consisting of points from the original shape and points from the IfcRelVoidElement shape extrude to the full height of the wall.

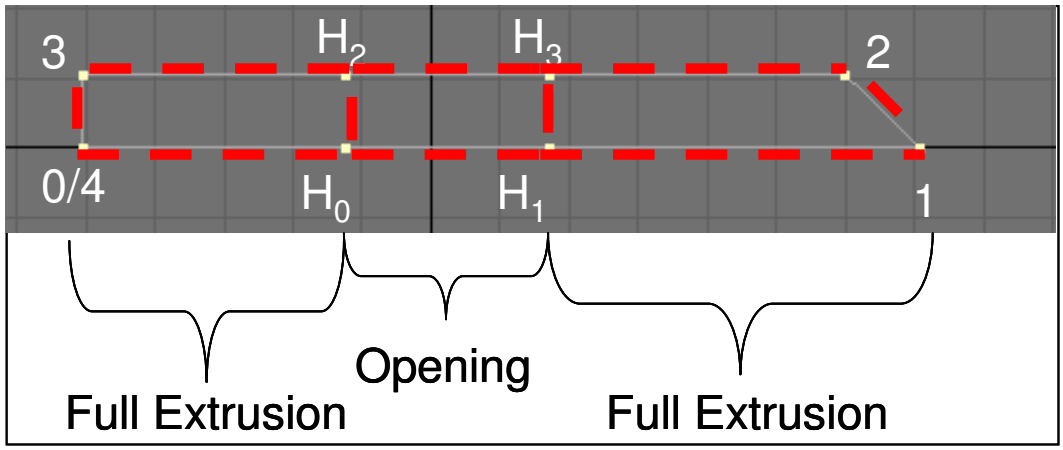


**Figure 6.33 Three Segments Created by Wall Opening.**

6. The opening shape, which contains only the points generated by the IfcRelVoidElement calculations, extrude differently based on the IfcRelFillElement associated with the opening. If the IfcRelFillElement is a door, then the opening extrudes from the top of the door to the full height of the wall, as shown on the left of Figure 6.34. Likewise, if the IfcRelFillElement is a window, then the opening points extrude from the bottom of the floor to the bottom of the window, and a second object extrudes from the top of the window to the full height of the wall. This appears as the shape on the right in Figure 6.34.

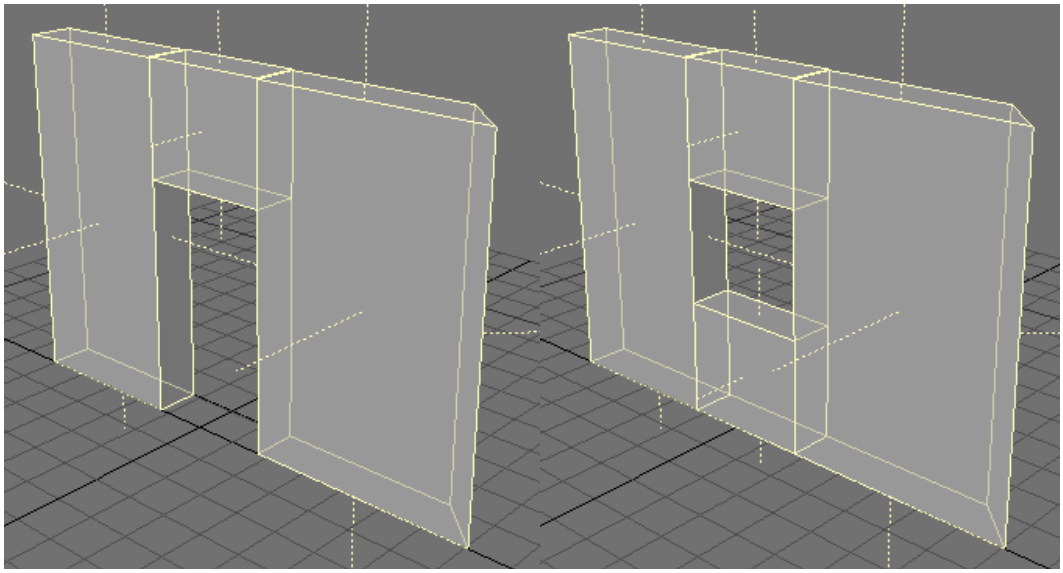

**Figure 6.34 Left: Door Extrusions, Right: Window Extrusions**

7. If the opening was intended for a window, a third extrusion is created using the same profile used for the top and bottom opening extrusions. This is done from the bottom of the window to the top, between the previous two extrusions. In order to differentiate the window from the wall a transparent glass texture is applied to the window surfaces.
8. If the opening were intended for a door, however, the width of the wall is taken into account. If the width of the wall is greater then the width of the door to be placed in the opening, as seen in Figure 6.35, the two points that correspond to the depth of the door points are added, $D_0$ and $D_1$. The resulting list of door points would be $H_0$, $H_1$, $D_1$, $D_0$, and H0. These points extrude to the height of the door.

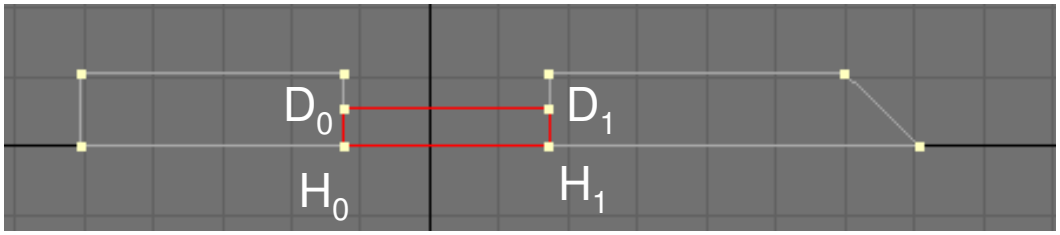


**Figure 6.35 Door in Red, Seen When Wall Width Is Thicker than Door Width**

9. When passed to the .map file writing function, doors are special entities known as `func_rotatingdoor`. Doors in Doom 3 slide open to the left and right. `func_rotatingdoor` is a custom object script written for Doom 3 by Bruce Worrall that allows doors to rotate about a pivot point [60].
10. The point sent as the pivot point is $H_0$. In the visualization, all doors rotate in a clockwise manner about their respective $H_0$ as seen from a top down XY perspective. For example, see Figure 6.36. Doors rotate clockwise about point $H_0$.

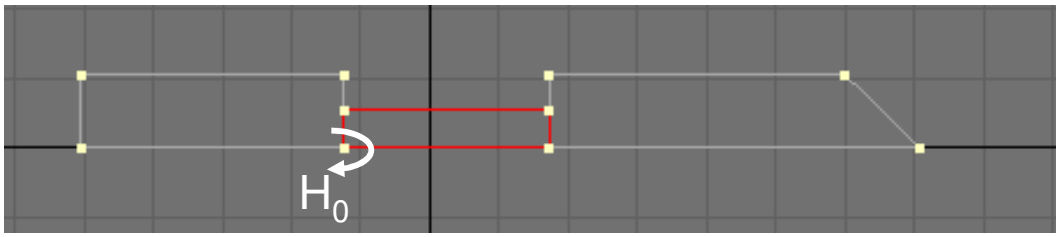


**Figure 6.36 Doors Rotate Clockwise About Point $H_0$**

When performing a conversion, it is important to note that a single method handles both single and double door. The result of using this method, when passed double doors, produces a single brush that spans the width of the double door opening. The large

double door opens and closes in the same fashion as the single doors, using the first point in the opening as a pivot point. The result is not an error and can be seen in Figure 6.37. The double doors are shown as very wide single doors. Algorithms for handling double doors were discussed; however, time limitations prevented them from being implemented.

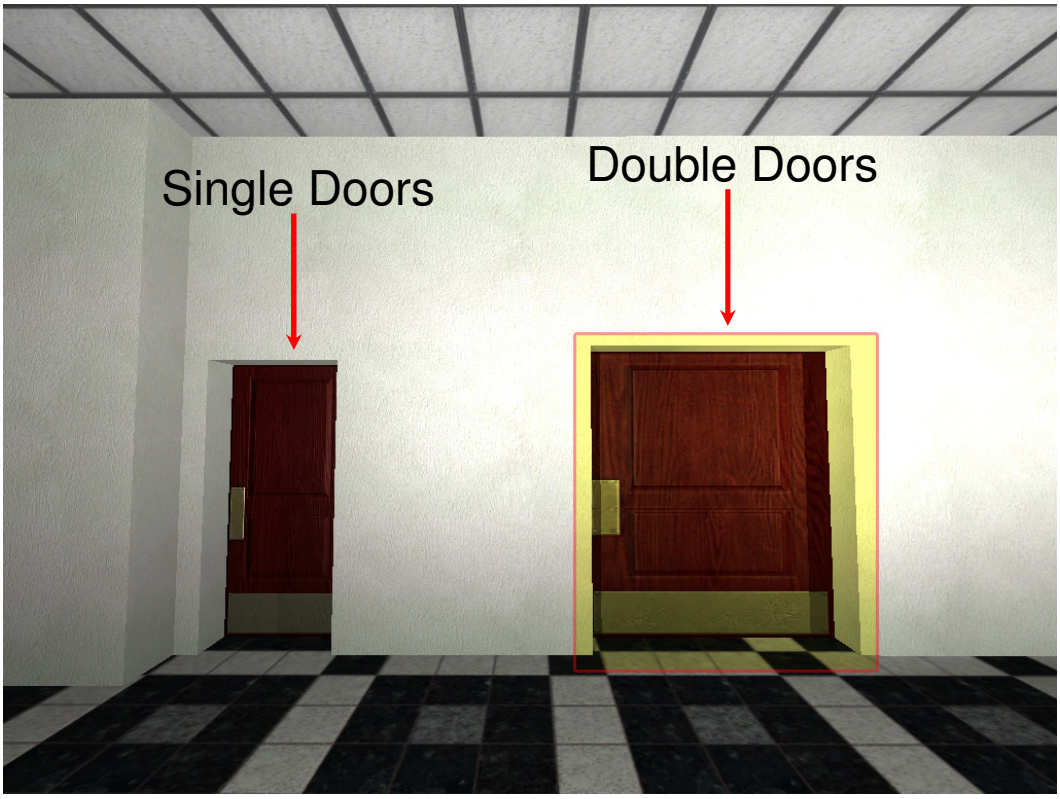


**Figure 6.37 Doors: Single on the Left, Double on the Right**

## 6.5 Writing the .MAP File Format

After all IFC building elements have been read from the .IFC file and converted into a format compatible with the .map file format, the .map file writing class traverses all `GeoObject`(s) and `NongeoObject`(s) and writes the result of the transformation function into .map file format. The simplest .map file that can be played is shown in Figure 6.38. .map files consist of a number of entities written together in a single ASCII file. The only two entities required to create a map that can by 'played' or walked around inside of are the `worldspawn` and `info_player_start` entities.

**Code:**

```
Version 2
// Example worldspawn entity
{
"classname" "worldspawn"
//Primitive 0
brushDef3 …
}
// Example Player Start Entity
{
"classname" "info_player_start"
"name" "info_player_start_1"
"origin" "1825.13 1013.36 0"
}
```

**Figure 6.38 Simple Generic .MAP File**

The `worldspawn` entity contains the definition of all brushes, brushDef3(s), which do not move (static brushes), in a map. These include all `geoObjects` such as walls,

windows, floors and ceilings. The `info_player_start` entity designates the start position of a player when the level is loaded. With these two entities, a map can be created and played. However, Doom 3 does not have ambient light so additional entities such as `Light`(s) are needed to add illumination to the space. `Light`(s), as shown in Figure 6.39, contain at a minimum, the `classname`, a unique object `name`, and finally the `origin` of where the light is located.

**Code:**

```
// Example Light entity
{
"classname" "light"
"name" "IFCtoMAP_Light 47303"
"origin" "1590.4 -73.286 71.7556"
}
```

**Figure 6.39 Example `Light` Entity**

As mentioned in the transformation above, Doors are written into the Doom 3 .map file format as special entities called `func_rotatingdoor`. Figure 6.40 shows an example of a `func_rotatingdoor`. `func_rotatingdoor` entities consist of the `classname`, a unique object `name`, the name of a `model`, an `origin`, an `open_angle` and finally a primitive `brushDef3`. The `model` entry has the same name as the object `name`, and this informs the Doom 3 of a `brushDef3` object defined within this entity. The `origin` is the rotational pivot point mentioned in the conversion above and labeled as the point $H_0$. The `open_angle` determines the direction of

rotation about the pivot point when a door is activated. Figure 6.40 shows an example of a door that rotates about the y-axis counterclockwise, -90 degrees when activated.

**Code:**

```
// Example Door entity
{
"classname" "func_rotatingdoor"
"name" "func_rotatingdoor_3871"
"model" "func_rotatingdoor_3871"
"origin" "1986.91 419.199 0"
"open_angle" "0 90 0"
// primitive 0
{
 brushDef3
 {
  ( 0 0 1 -90 )(( 0 0 -11 ) ( 0 0 -39 ) ) "texturename" 0 0 0
  ( 0 0 -1 0  )(( 0 0 -11 ) ( 0 0 -39 ) ) "texturename" 0 0 0
  ( 0 1 0 -0  )(( 0 0 -11 ) ( 0 0 -39 ) ) "texturename" 0 0 0
  ( 1 0 -0 -40)(( 0 0 -11 ) ( 0 0 -39 ) ) "texturename" 0 0 0
  ( 0 -1 0 -3 )(( 0 0 -11 ) ( 0 0 -39 ) ) "texturename" 0 0 0
  ( -1 0 0 0  )(( 0 0 -11 ) ( 0 0 -39 ) ) "texturename" 0 0 0
 }
}
}
```

**Figure 6.40 Example `func_rotatingdoor` Entity**

The only other objects to be transferred are `NongeoObject`(s). The special case `NongeoObject`(s), such as `Light`(s) and the `info_player_start` location, have been mentioned above. All other `NongeoObject`(s) are written generically as `func_static` entities in the .map format, as demonstrated in Figure 6.41. The `func_static` entities consist of the `classname`, a unique object `name`, an `origin`, an `open_angle` and a `model`. The model in the .map format is a string that holds a relative path to the location of a 3D model to be loaded by the game at runtime.

In the example in Figure 6.41, if the Doom 3 were run from the base folder installed in program files then the chair model might be located as follows:

```
C:\Doom 3\base\models\IFCtoMAP\Chair.lwo
```

**Code:**

```
// Example func_static entity - Chair
{
"classname" "func_static"
"name" "/models/IFCtoMAP/Chair.lwo 46199"
"origin" "1897.13 745.483 0"
"model" "/models/IFCtoMAP/Chair.lwo"
}
```

**Figure 6.41 Example `func_static` Entity**

## 6.6 Summary

The IFCtoMAP program reads a specific set of IFC building elements and converts the IFC representation into a format understood by the Doom 3 game engine. This conversion uses the IFCLoader classes, the ConversionMethod class, and the MapWriter class. Throughout the process, the data is stored in an instance of the Building class. The class structure of the program allows for the addition of new IFC building elements and IfcShapeRepresentation(s) by extending the current definitions.

When selecting additional IfcShapeRepresentation(s) to implement, one should consider the expected frequency of that shape representation occurring in IFC files. For example, adding basic support for B-rep objects in IFCtoMAP allowed for the reading of IfcStair and IfcStairFlight objects. However, the numbers of staircases seen in a building are relatively limited as compared to the number of holes in walls, such as those for doors and windows. Within the scope of the files tested during the creation of the IFCtoMAP program, the most commonly seen IfcShapeRepresentation(s) were IfcRectangularProfileDef and IfcArbitraryClosedProfileDef. Therefore, the author concluded time was best spent working on commonly seen IfcShapeRepresentation(s), such as IfcRectangularProfileDef and IfcArbitraryClosedProfileDef.

The above discussion is an overview of the methods used by the IFCtoMAP software to convert information from IFC into .MAP file format. For more detailed information related to the C# implementation, the IFCtoMAP program source code can be found in the Appendices in the Electronic File attachments as Source Code and Doxygen Documentation.

# 7 SIMULATION METHODS

## 7.1 Introduction

When choosing the method for simulation, three options were considered: (1) the creation of a new energy simulation calculation tool that would be linked to Doom 3 visualization; (2) the creation of calculations from within the Doom 3 game; and (3) the reuse of an existing simulation calculation tool such as EnergyPlus.

The first option was to create a new energy simulation calculation tool which could be externally linked to the game engine. There are two obstacles identified with this approach. First, the new simulation needed to use IFC data, however, there is only one program found in the literature survey which can write IFC HVAC data: MagiCAD. Second, capturing the output data from the energy calculations and reading them into the visualization was problematic. The Doom 3 game logic source code is available for download from the Id Software Developer website as a Software Development Kit (SDK). The Doom 3 game logic only allows for the reading and writing of a limited number of file types (.map, .script, .def, etc). With the author's limited knowledge of C++ and the limited documentation related to file reading and writing, the level of effort required to add file reading and writing routines for new file types into the Doom 3 game logic were considered to be out of the scope of this research. However, it is possible to rewrite the game logic and add support for reading new file types, such as the output from an energy simulation.

The second option for simulation involved integrating the energy calculations into the game. One way to do this would be to edit the game logic code from the SDK. Similar to adding new reading and writing methods discussed in the previous paragraph, the addition of an energy simulation into the Doom 3 game logic was outside the scope of this research. An alternative approach is to use Doom 3's on-the-fly script compiler, where a script written in the D3 scripting language compiles and run when the game runs. Similar to the `func_rotatingdoor` a new entity type created with additional properties such as an energy calculation. When investigated, the use of scripts became an issue when gathering data from the user. Doom 3 uses a custom .GUI or graphical user interface file to create and display interactive data from within the virtual environment. A number of factors limit the interactions; most importantly, the users can only interact with the GUIs by the use of a left click; text data entry is not possible. The data collection requirements for an energy simulation require input of multiple numeric variables that cannot reasonably be entered by left clicking alone.

The third option, reusing the simulation calculation tool Energy Plus, appeared to be promising as numerous journal articles and papers described tools such as IFCtoIDF, IFC interface to Energy Plus, and Energy Plus HVAC GUI that were intended to simplify the creation of simulation inputs by reusing data from IFC files[16, 21-23, 61-63]. Reports of the tools were similar to this:

> *In addition to its previously released IFCtoIDF utility that semi-automates the import of building geometry, the new IFC HVAC interface to EnergyPlus (released at the end of 2003) makes it possible to import*

*and export most of the data that define HVAC equipment and systems in a building directly from and to other IFC compatible software tools. This reduces the manual input of other data needed for successful simulation with EnergyPlus to a minimum… EnergyPlus is the first building energy performance simulation model able to import data directly from a Building Information Model (BIM) that describes a given building in IFC format. Its IFCtoIDF utility allows users to seamlessly acquire data that completely describe a given building geometry in a format needed for the simulation*[22].

## 7.2 Energy Plus

These reports created the expectation that a set of tools is currently available to facilitate the movement of geometric and mechanical HVAC data from IFC to an Input Data File (.IDF) file format used by Energy Plus. These reports were accurate in that data formatted according to the IFC specification was converted into an Energy Plus input file. However, the reports also stated that as of yet "no IFC files that contain diverse IFC2x2 based HVAC data can be generated by other tools at the moment…"[22].

Bazjanac described one of the IFC HVAC data generation tools, MagiCAD. To test MagiCAD for this research, a copy was obtained from Jani Suonvieri. The first task attempted was to load the Langford Building B .ifc file, which was exported from Revit. This was unsuccessful as the MagiCAD software crashed each time an attempt was made to load the Langford file. MagiCAD came with a sample IFC file that contained

data already created. Using Revit, an attempt was made to open the MagiCAD sample file. Likewise, Revit crashed when importing the MagiCAD file. After contacting both software developers, it is uncertain the exact cause of the incompatibility.

After investigating the one possible source of IFC HVAC data, attention was focused on moving building geometry into Energy Plus to generate a minimum a set of loads. During the investigation of Energy Plus, the author found that the building geometry conversion utility IFCtoIDF successfully converted data from IFC to IDF format. However, the IFCtoIDF utility did not convert all elements of the building's geometry from IFC into the IDF format.

Figure 7.1 shows a three dimensional view of the Langford B Building as it appears in Autodesk Revit Building including the walls, top, the floors and ceilings, center, and the combination of both, bottom. In the center of the same figure, one sees a screenshot of the .ifc file data as rendered by a freeware .ifc file viewer software called IFC Engine Viewer[64]. The Autodesk Revit .rvt file is attached in Appendix 0 These items accompany this thesis as separate files available for downloading as 070607_CMc_LangfordBuildingB.zip and 070607_CMc_ThesisMod.zip:

- The file 070607_CMc_LangfordBuildingB.zip contains the Langford Building B file data as follows:

  Langford Building B Revit File.

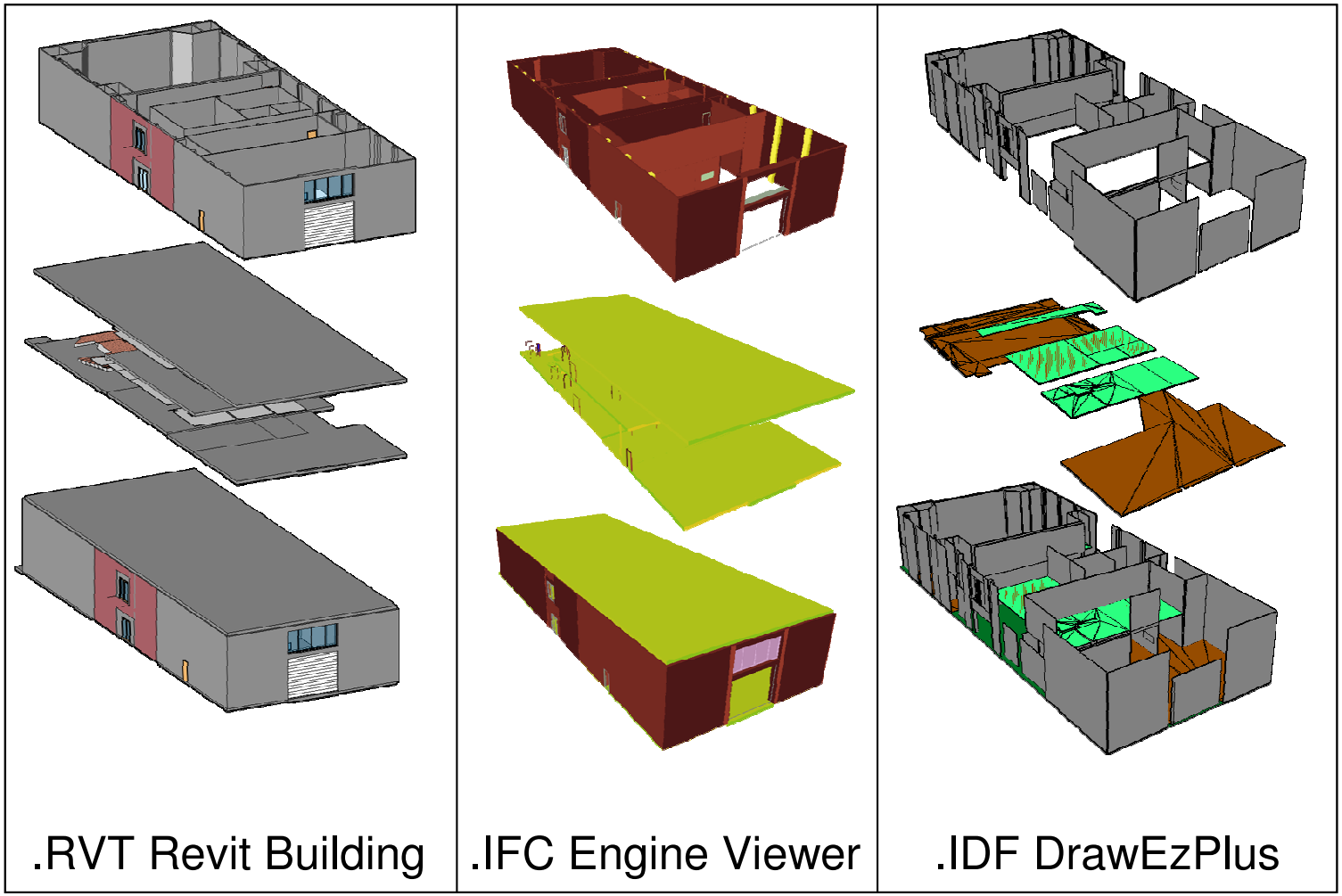


**Figure 7.1 Three Representations of Langford Building B: Left Revit Building, Center .ifc as seen in IFC Engine Viewer, Right .idf as seen in DrawEzPlus.**

The right side of Figure 7.1 shows the IDF output of the IFCtoIDF conversion of the same data set, Langford Building B. When looking at these three images, notice the IFC representation of the Langford Building appears almost indistinguishable from the Revit representation. The IDF output on the other hand is visibly missing elements seen in the other two files. Most notably the IDF file is missing the building roof, large swaths of the floor space in the center of the building, and many wall segments. These elements are missing as a result of the IFCtoIDF conversion process. Connected geometry is not necessary in order to perform an Energy Plus simulation. However, the simulation does require that all thermal zones be bounded by at least six surfaces as shown in the Appendices labeled as Langford Building B IFCtoIDF EnergyPlus Error File. The missing surfaces prevent the simulation from being performed.

The input data file for Energy Plus generated by the IFCtoIDF converter, attached as Appendix o, consists of more than ten thousand lines of ASCII text. Debugging this file would not be a trivial task, however it could be done. Debugging the Langford Building B IDF file could produce a successful simulation. For the purposes of this research it was sufficient to have verified that the IFCtoIDF conversion is still a semi-automated system that still requires the interaction of a user to generate a successful simulation.

## 7.3 Summary

Of the options explored none resulted in the successful creation of a simulation that made full use of existing BIM data from IFC. Each approach had limitations requiring an additional software development effort to make them truly automated. There is a lack of IFC mechanical equipment generation tools, forcing a user to create a simulation separate from the geometry creation. This results in a disconnect between the geometry and the simulation (mechanical information). The resolution of this situation has great practical benefits.

# 8 RESULTS

The aim of this research was to develop software technology for reusing buildings described in the IFC BIM standard in both a simulation and visualization. To test this goal, an on campus building was selected to use as a test case for the visualization. The selected building was the Architecture B Building located on main campus between the Bright Building and the Architecture A and C buildings, as shown in Figure 8.1. The Architecture B Building has a number of spaces serving multiple purposes including: a woodshop, an auditorium, classrooms, and research space.

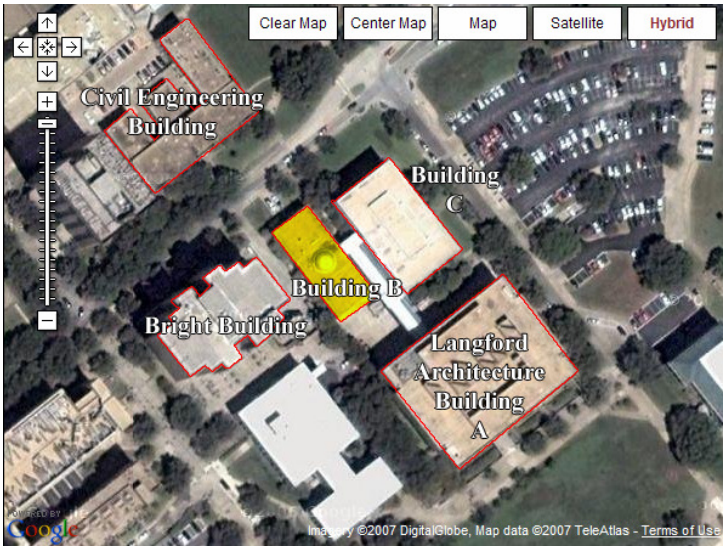


**Figure 8.1 Arial View of Architecture Building B on Campus**

The selected Architecture Building B contains many elements of a common office building and served as a good example of the typical geometry that can be transformed in the .map file format. What follows is a brief review of the supported IFC Building Elements as they exist within the Architecture B building and how they compared to the virtual building elements from the resulting .ifc file.

Figure 8.2 shows the floor plans of the Architecture Building B color coded to describe the space usage and labeled with numbered bubbles to represent camera positions of Figure 8.3-Figure 8.7. The camera positions are as follows:

1. The lobby from the perspective of the main double doors at the side entrance to the building
2. The woodshop office walls as seen from the within the woodshop
3. The auditorium from the perspective of a front corner
4. A first floor view of the stairwell that links the first to the second floor
5. The second floor hallways directly leading out from the stairwell

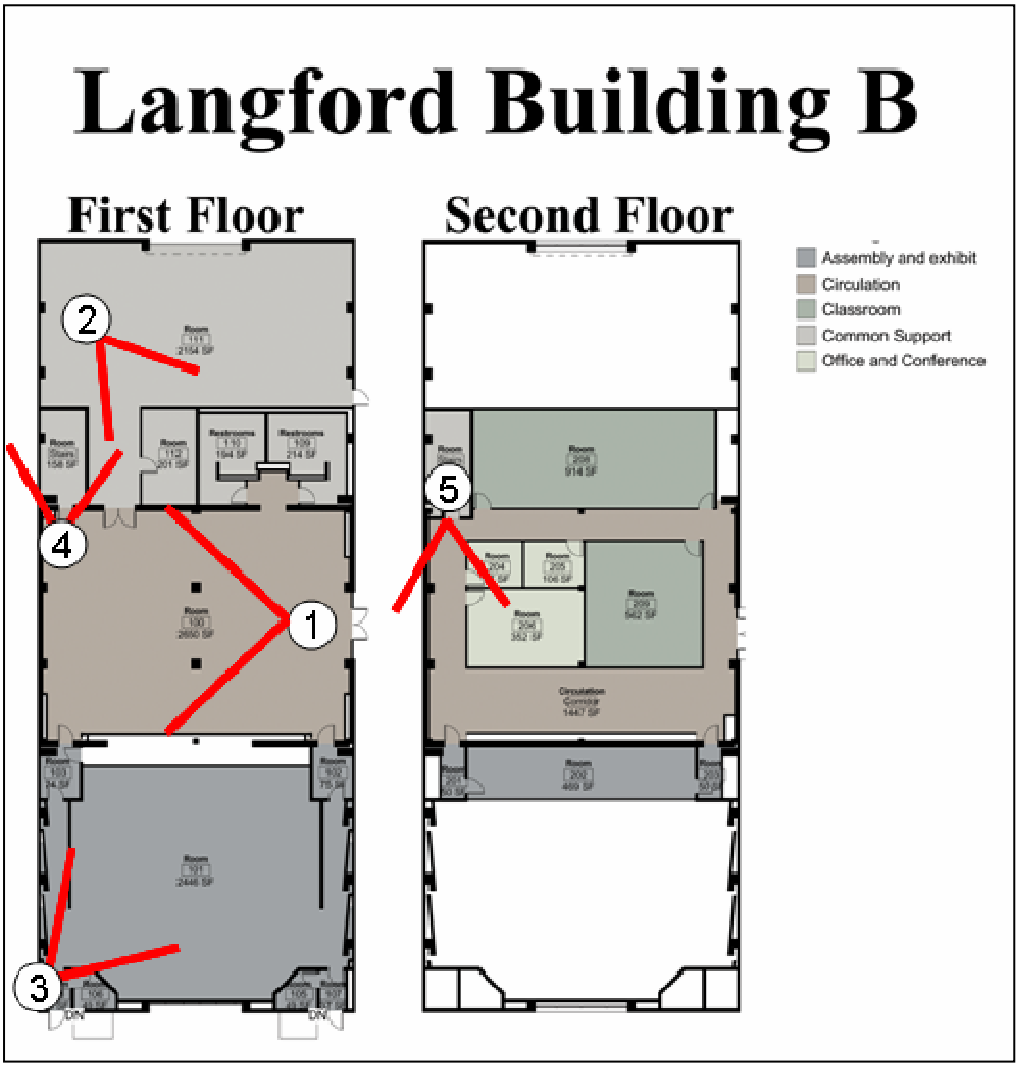


**Figure 8.2 Langford Building B Floor Plans**

The first two images in Figure 8.3 show the Langford Building B lobby from the perspective of the main double doors at the side entrance to the building and are labeled as camera positions 1 in Figure 8.2. Three distinct differences between the actual lobby and the virtual representation are the textures on the surfaces, the lighting, and the doors are set into the walls. The textures are the generic textures described earlier that IFCtoMAP applies to all similar surfaces. A marble tile texture is placed on all +Z facing

surfaces, particularly floors; a ceiling tile texture on all -Z facing surfaces, such as ceilings; and an off-white texture is placed on all ±X and ±Y surfaces, such as the walls and sides of floor slabs. Because Doom 3 does not have global illumination or ambient lighting, point lights were placed by hand to illuminate the space. Finally, the doors are set into the walls by the automated function discussed above that places doors in walls based on the first point describing the door and the depth of the wall.

Figure 8.4 shows the Langford Building B woodshop office walls as seen from the within the woodshop and is labeled as camera position 2 in Figure 8.2. The visual results created by the Doom 3 engine are quite similar to the actual environment. Similar to doors, the frames of the windows are not converted by the IFCtoMAP program, only the openings. In the case of windows, extruded shapes are generated to fill the window opening.

Figure 8.5 shows the Langford Building B auditorium from the corner of the room and is labeled as camera position 3 in Figure 8.2. The auditorium is a large open space with curved steps with seats. The Revit file represented the steps as rectangular slabs and they were imported into the visualization as such. It is important to note that there exists a perspective difference between the virtual representation and the actual representation that is caused by the difference in the field of view of the camera and the virtual user. The virtual user has between a 90-120 degree field of view where the camera image has 90 degrees or less in the field of view.

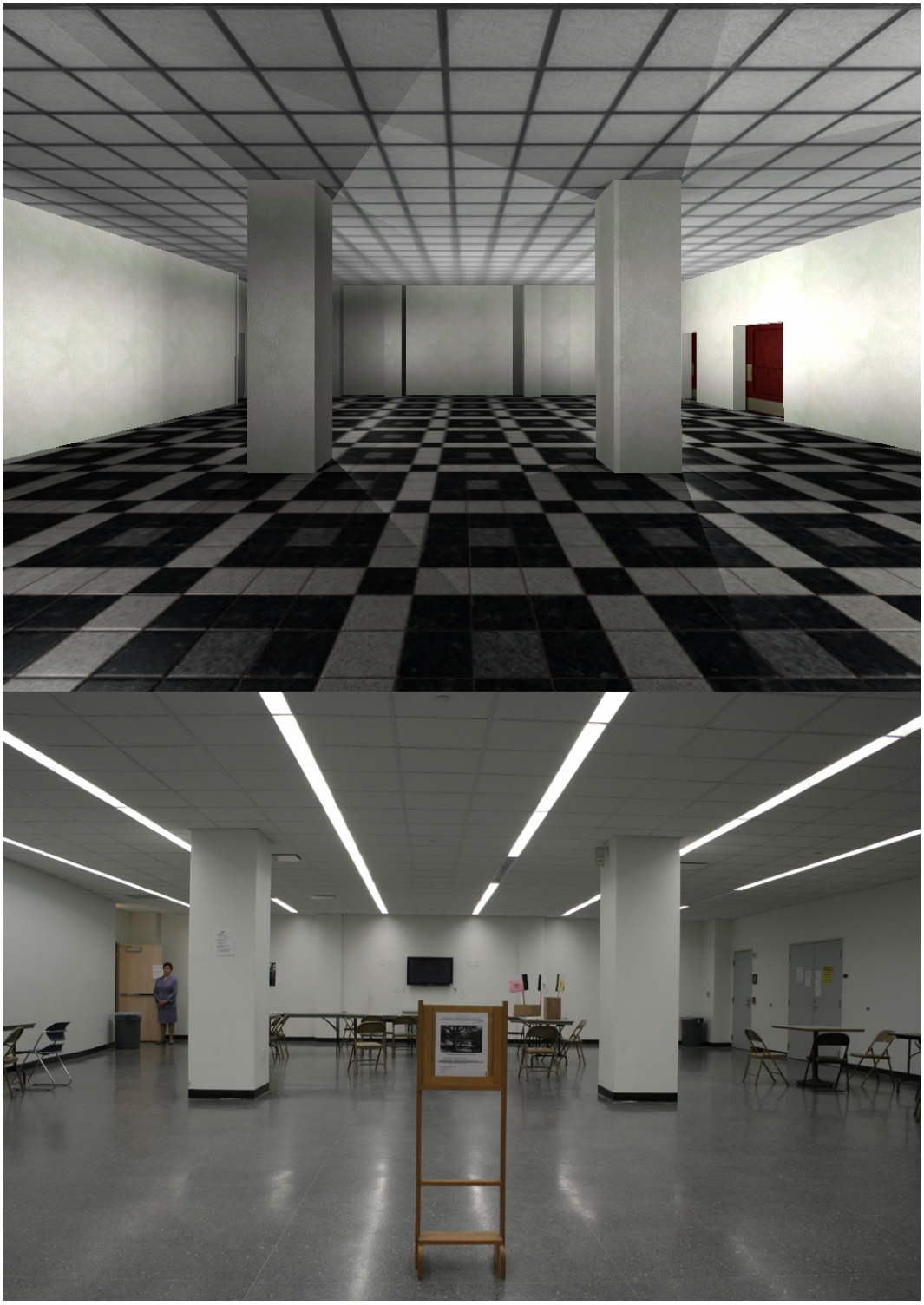

**Figure 8.3 Langford B Lobby: Virtual Representation on Top, Actual on Bottom**


**Figure 8.4 Woodshop Office: Virtual Representation on Top, Actual on Bottom**

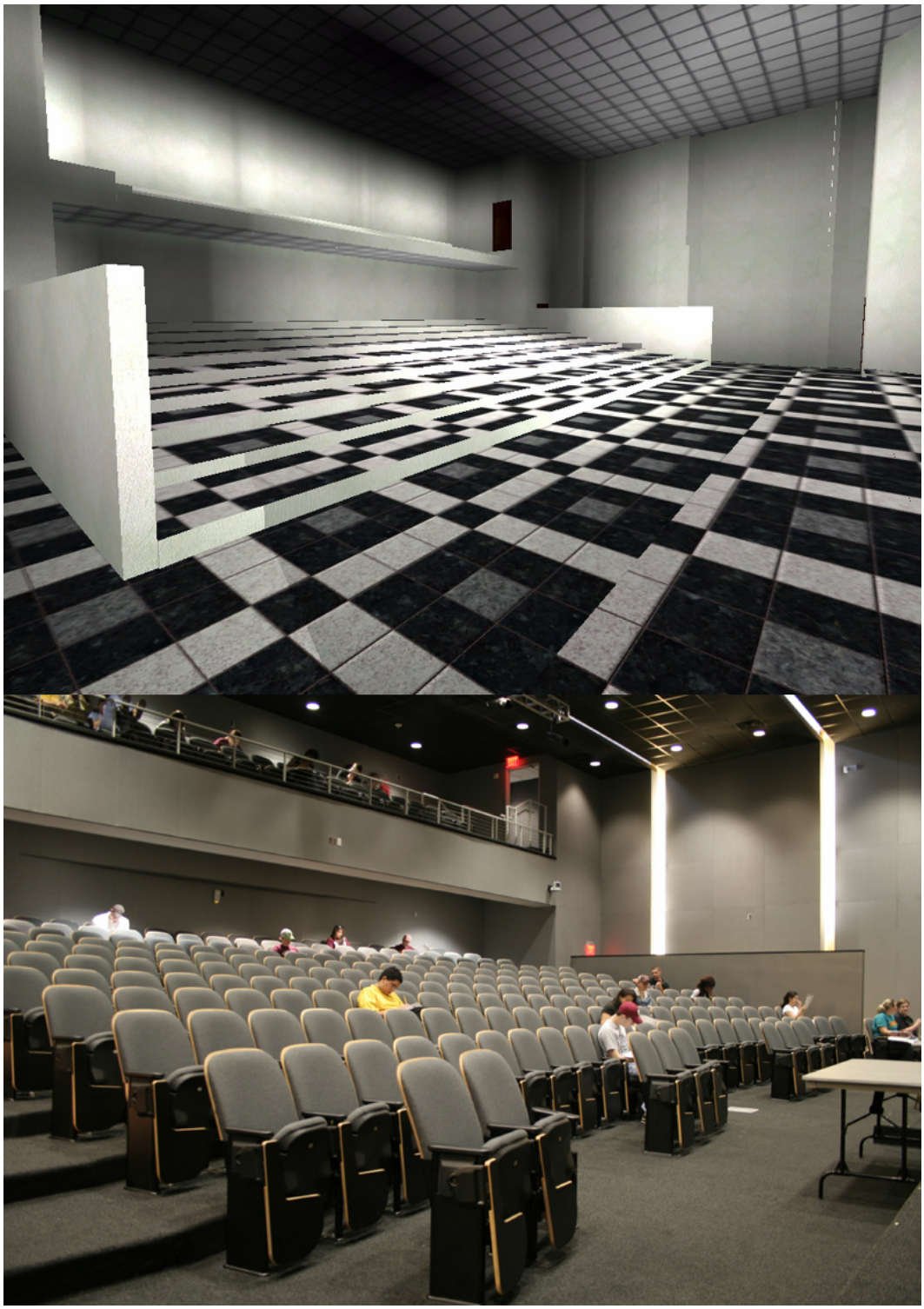

**Figure 8.5 Auditorium: Virtual Representation on Top, Actual on Bottom**


**Figure 8.6 Stairs: Actual on Left, Virtual Representation on Right**

There are distinct differences in the two images seen in Figure 8.6, labeled as camera position 4 in Figure 8.2. First, the number of stairs in the actual picture are not the same as the number of stairs in the right image. The reason for the discrepancy lies within the BIM model used. The BIM model has stairs placed in the stairwell that do not correspond to the actual number of stairs in the building. The second difference is the left half of the landing was not displayed in the Doom 3 model. This was investigated and landings are often composed of B-rep objects with concavities. These concavities can cause the B-rep objects to "disappear" in Doom 3. This same error can occur with any B-rep object that contains concavities.

**Figure 8.7 Upstairs Hallway: Actual on Left, Virtual Representation on Right**

With the exception of the perspective distortion seen in the virtual representation, the images in Figure 8.7 are a fairly accurate match showing the second floor of Langford Building B, labeled as camera position 5 in Figure 8.2. The field of view of a user within the Doom 3 game engine is set to 90 degrees by default, but can be changed to suit the user.

When transforming data from a point representation into a plane and distance based representation, there was a need to round off numbers to compare values. For example, when comparing .0000003 to 0, the result would be false. These numbers are not the same. However, in many instances .0000003 is so small that it can be considered to be 0 to reduce the calculation time. The results, as seen in Figure 8.8, are the creation of small jagged edges that come from the constrained triangulation calculations where a number is rounded off and the planes do not intersect exactly where expected.

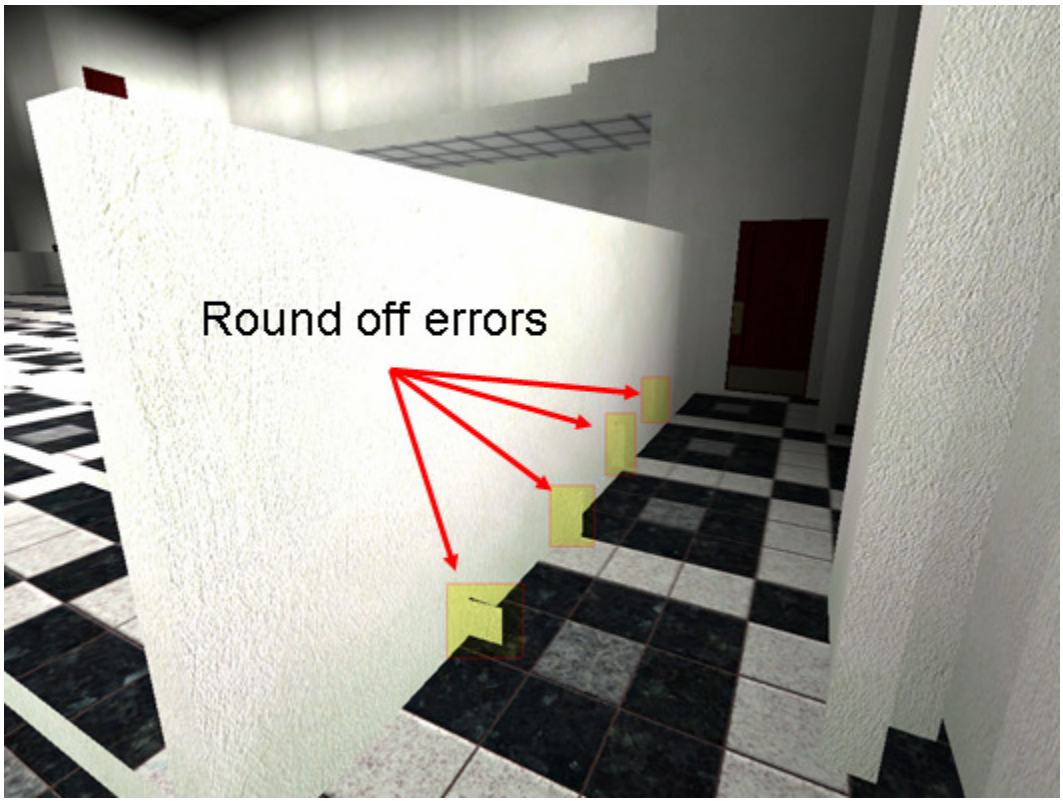


**Figure 8.8 Round Off Errors**

Figure 8.9 shows the result of a Doom 3 engine's failed attempt at optimizing geometry and flipping a triangle's surface normal in order to correct backwards triangles. However, these segments of wall did not contain backwards triangles and as a result the engine did not render them properly. These cases happen within the Doom 3 engine, and can be corrected by a manual adjustment of the brushes. No discernable pattern has been recognized as to why this happens.

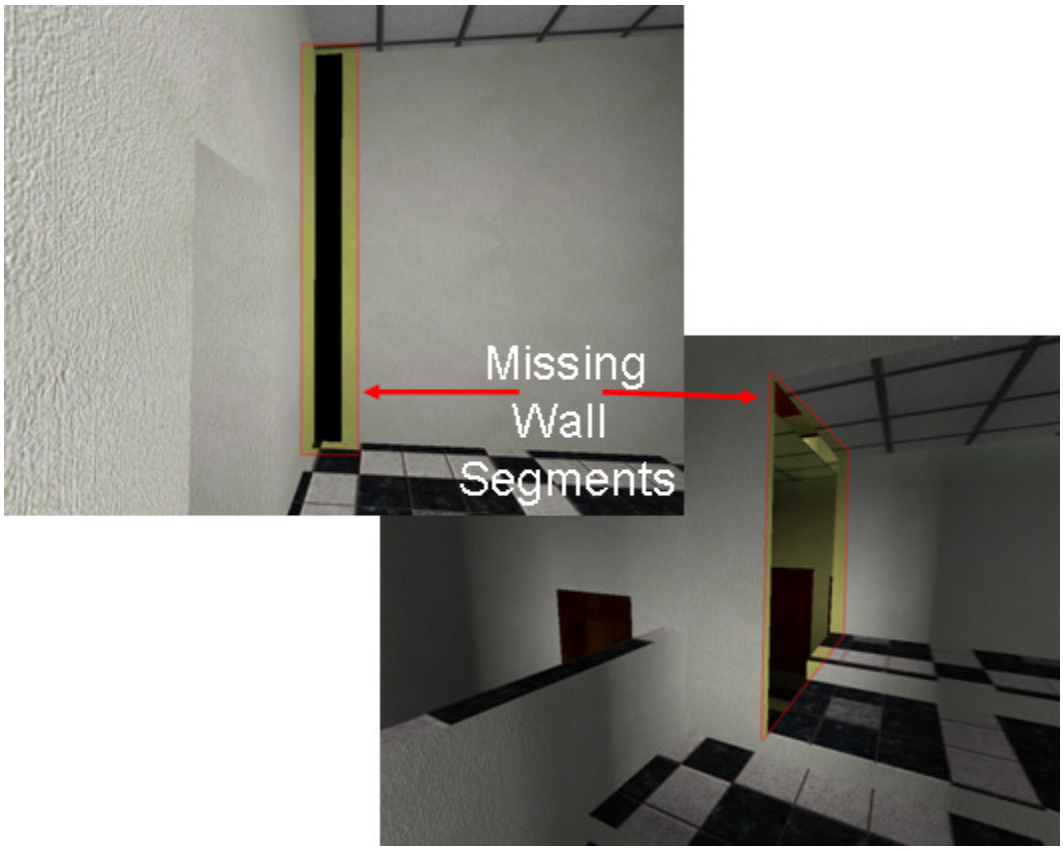


**Figure 8.9 Backwards Triangles**

# 9 CONCLUSIONS AND RECOMMENDATIONS

This document describes the process through which information from a Building Information Modeling tool such as Autodesk Revit Building can be moved through the Industry Foundation Classes into a visualization, such as Doom 3 and an energy simulation such as Energy Plus. A test building, the Langford Building B was used to demonstrate the process.

The objective of the research was to identify and develop software technologies that could use a building's geometry from a BIM file within both a visualization and the calculation of an energy simulation. The results of the energy calculations were intended to be displayed in a spatially relevant virtual environment. In order to achieve that objective the following tasks were undertaken:

- The Doom 3 game engine was selected as the game based 3D virtual environment to display the building's geometry.
- The IFC CAD file standard was selected and common building elements such as walls, windows, doors, floors and ceilings were selected and converted into a format recognizable by the Doom 3 game engine.
- The IFCtoMAP software was developed, which represents a technology framework for extracting select information from IFC files and transforming that information into a format usable by the Doom 3 game engine.
- IFCtoIDF was identified as a possible software solution that could facilitate the use of a building's geometry stored in an IFC file within Energy Plus.

The objectives of the research were successfully met. Technologies were developed and identified that could be used in a Visual Energy Use System by facilitating the use of Building Information Modeling tools in driving a visualization and a simulation as seen in Figure 9.1. Finally, the author proposes that with a change in game logic code, in a game such as Doom 3, the output of an energy simulation could be read into a game engine and displayed for a user. This objective was explored, but not fully achieved.

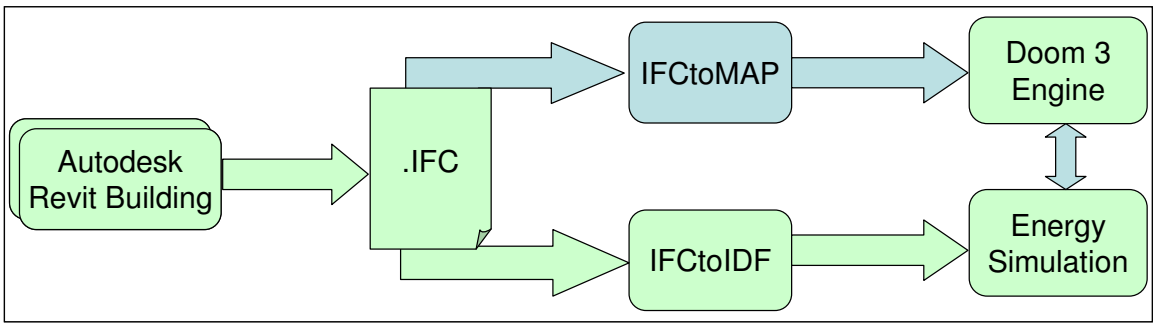


**Figure 9.1 Framework for a Visual Energy Use System**

The state of the art IFCtoIDF tool was identified as a potential tool for semi-automated conversion of IFC data into Energy Plus. The IFCtoIDF technology has IFC compatibility issues to work out. As of the writing of this document, IFCtoIDF had been removed from the Energy Plus installer for at least three releases (1 year). The IFCtoIDF conversion performed on the Langford Building B IFC file did not produce a simulation that could be immediately run. The errors generated by the conversion could be resolved by reviewing the ten thousand line IDF file and manually correcting the errors. This leads back to Bazjanac's claim that the level of effort required for the preparation of a

building energy simulation input file can be reduced by a factor of four through the use of semi-automated tools [12].

In addition to identifying the potential of IFCtoIDF, a new conversion utility was developed, IFCtoMAP. The IFCtoMAP conversion utility takes a building's geometry as found in an IFC file and converts the geometric representation into a format understood by the Doom 3 game engine. This conversion utility demonstrates the versatility of the data stored within an IFC file.

The IFCtoMAP component has many applications beyond energy simulation visualization. There were no reports within the literature survey of instances where BIM information could be directly converted into first person perspective visualization. Included among the applications of IFCtoMAP outside the scope of this research are: real estate pre-sales walkthroughs with long distance clients; the rapid creation of models for the build to suit industry; architectural pre-design walkthroughs; and the rapid creation of visualizations for virtual rehearsals of military tactical operations.

## 9.1 Further Work

This research is an initial study in the creation of a Visual Energy Use System. To further integrate visualizations and energy simulations into a single Visual Energy Use System, the following considerations warrant further study.

- When developing software, it is critical to have a well documented API and community of developers working with the same software. The longest delays when developing the software for this work were when learning to understand the IFC file

format. The documentation was vague and there were no free support communities found to assist in deciphering the documentation. Much of what was learned about IFC came from trial and error. The method most commonly used to understand the IFC was to export an IFC file from Revit, open it in IFCExplorer and review to see if the output was as expected. This method was done on single walls, walls connected to each other, floors, furniture, etc. Doom 3, though, was much simpler to learn because Doom3world.org has such an active community supporting anyone looking for help using the engine. With 150k+ threads of discussion, it was easy to find detailed answers to questions by using the built-in search function.

- From experience on this work, it does not appear that a first person perspective (FPP) game engine is the optimal approach to be taken when simulating a building's energy consumption. The same reasons that were used to justify choosing the first person perspective engine at the start of this thesis work are the same reasons this does not appear to be an optimal approach.
    - The user has a spatial sense of position within the building and those elements which make up the building. When seen from a first person perspective, the lack of detail in a space becomes apparent. The time spent creating visually pleasing art assets to hide or mask the lack of detail uses valuable time that could be spent more effectively on other aspects of the simulation.
    - Along the same lines, a FPP system puts the user in a single space within the building, and the user can easily lose perspective of the building as a whole. One

of the reasons for a building level simulation is to get a perspective of the entire building performance. This might be lost by a FPP system.

  - For these reasons, the author advises against using a FPP system in the future for this purpose.

- Given the above, a third person perspective or isometric perspective of a building is a more suitable approach. This researcher believes that a user may be more likely to accept a lower level of detail and more visual abstraction than a FPP system.
- From a development perspective, the need for fewer art assets and an overall lower level of detail is a distinct advantage over a first person perspective system.
- Based on the finding of this thesis research, the licensing of a third person perspective real time strategy game engine and the creation of custom game logic code may be a better approach to creating an energy consumption centric visualization. Using the game logic from an off the shelf game such as Doom limits the developer to using someone else's logic, which may be optimized for a first person shooter game but not for a FPP visualization.

# REFERENCES


1. EIA, "Annual Energy Review 2004," Department of Energy, Washington, D.C. DOE/EIA-0384 (2004), September 2005.
2. S. C. M. Hui, "Simulation Based Design Tools for Energy Efficient Buildings in Hong Kong," *Hong Kong Papers in Design and Development*, Department of Architecture, University of Hong Kong, 1998, pp. 40-46.
3. T. Kusuda, "Early History and Future Prospects of Building System Simulation," *Proc. Building Simulation*, vol. 1, 1999, pp. 3–15.
4. T. Hong, S. K. Chou, and T. Y. Bong, "Building Simulation: An Overview of Developments and Information Sources," *Building and Environment,* vol. 35, 2000, pp. 347-361.
5. D. B. Crawley, L. K. Lawrie, F. C. Winkelmann, W. F. Buhl, A. E. Erdem, C. O. Pedersen, R. J. Liesen, and D. E. Fisher, "The Next-Generation in Building Energy Simulation - A Glimpse of the Future," *Proc. of Building Simulation*, vol. 3, 1997, pp. 395-402.
6. M. R. Donn, "A Survey of Users of Thermal Simulation Programs," *Proc. Of IBPSA,* vol. 3, 1997, pp. 37-45.
7. K. P. Lam, N. H. Wong, and F. Henry, "Computer-Based Performance Simulation for Building Design and Evaluation: The Singapore Perspective," *Simulation & Gaming,* vol. 34, 2003, pp. 457-477.

8. Al-Sallal, K. A., and L. Degelman, "A Hypermedia Model for Supporting Energy Design in Buildings," *Proc. of ACADIA: Reconnecting*, 1994, pp. 39-49.

9. J. Whyte, N. Bouchlaghem, A. Thorpe, and R. McCaffer, "From CAD to Virtual Reality: Modeling Approaches, Data Exchange and Interactive 3D Building Design Tools," *Automation in Construction,* vol. 10, 2000, pp. 43-55.

10. V. Bazjanac, "Virtual Building Environments (VBE) - Applying Information Modeling to Buildings," *ECPPM 2004 – eWork and eBusiness in Architecture, Engineering and Construction,* Istanbul, Turkey, 2004, pp. 41-48.

11. J. A. Clarke, *Energy Simulation in Building Design*, 2nd ed. Oxford, Boston: Butterworth-Heinemann, 2001.

12. V. Bazjanac, "Acquisition of Building Geometry in the Simulation of Energy Performance," *Proc. Building Simulation Conference,* Rio de Janeiro, Brazil, vol. 1, 2001, pp. 305-311.

13. J. P. Waltz, *Computerized Building Energy Simulation Handbook*. New York: Marcel Dekker, Inc., 2000.

14. J. Glazer, [Software] *EP-Quick*, Glazer Software, 2005.

15. P. Chaisuparasmikul, "Bidirectional Interoperability Between CAD and Energy Performance Simulation Through Virtual Model System Framework," *Synthetic Landscapes Proc. of the 25th Annual Conference of the Association for Computer-Aided Design in Architecture*, Louisville, KY, 2006, pp. 232-250.

16. V. Bazjanac, "Improving Building Energy Performance Simulation with Software Interoperability," *Eighth International IBPSA Conference* Eindhoven, The Netherlands, vol. 1, 2003, pp. 87-92.

17. I. Kim and J. Seo, "Founding a Common Ground for the Emerging Industry Model Standard (IFC) and ISO Model Standard (STEP) for the Global Construction Industry," *Proc. of the World IT Conference for Design and Construction / INCITE*, Malaysia, 2004, pp. 535-542.

18. Autodesk Inc., *Autodesk Revit Building 9 Fully Certified for IFC Export*, San Rafael, CA, Autodesk Inc., 2006.

19. Bentley Architecture Inc., *Bentley Architecture Achieves IFC2x3 Certification*, Exton, PA, Bentley Architecture Inc., 2007.

20. Graphisoft, *IFC 2x Edition 2 Add-On Available for ArchiCAD,* Budapest, Hungary, Graphisoft, 2007.

21. E. Morrissey, J. O'Donnell, M. Keane, and V. Bazjanac, "Specification and Implementation of IFC Based Performance Metrics to Support Building Life Cycle Assessment of Hybrid Energy Systems," SimBuild, IBPSA-USA, National Conference Boulder, CO, 2004.

22. V. Bazjanac and T. Maile, "IFC HVAC interface to EnergyPlus - A Case of Expanded Interoperability for Energy Simulation," SimBuild, IBPSA-USA, National Conference Boulder, CO, 2004.

23. J. O'Donnell, E. Morrissey, M. Keane, and V. Bazjanac, "BuildingPI: A Future Tool for Building Life Cycle Analysis," SimBuild, IBPSA-USA, National Conference Boulder, CO, 2004.

24. J. Plume and J. Mitchell, "Collaborative Design Using a Shared IFC Building Model--Learning from Experience," *Automation in Construction,* vol. 16, pp. 28-36, 2007.

25. J. S. Haberl and S. Cho, "Literature Review of Uncertainty of Analysis Methods," Texas A&M University, College Station ESL-TR-04/11-1, November 2004.

26. APEC, *HCC-Heating/Cooling Load Calculation Program*, Dayton, OH: Automated Procedures for Engineering Consultants, Inc., 1967.

27. H. Lau and J. M. Ayres, "Building Energy Analysis Programs," *11th Conference on Winter Simulation*, San Diego, CA, vol. 1, 1979, pp. 283-289.

28. J. M. Ayres and E. Stamper, "Historical Development of Building Energy Calculations," *ASHRAE Journal,* vol. 37, 1995, p. 8.

29. T. Kusuda, *NBSLD: Heating and Cooling Loads Calculation Program, National Bureau of Standards.* Washington D.C.: Dept. of Commerce, National Bureau of Standards, 1976.

30. R. H. Henninger, J. McNally, and R. Wallace, *NECAP - NASA's Energy - Cost Analysis Program, Part I - User's Manual, Part II - Engineering Manual*: National Aeronautics and Space Administration, 1975.

31. G. S. Leighton, H. D. Ross, M. Lokmanhekim, A. H. Rosenfeld, F. C. Winkelmann, and Z. O. Cumali, "DOE-1: A New State-of-the-art Computer Program for the Energy Utilization Analysis of Buildings," *International Symposium on the Use of Computers for Environmental Engineering Related to Buildings* Banff, Canada, 1978.

32. U. S. DOE, [Available Online] "Building Energy Software Tools Directory," available online *http://www.eere.energy.gov/buildings/tools_directory/*, 2005.

33. K. Blazej, W. Burkhard, G. John, and H. John, "Information Visualisation Utilising 3D Computer Game Engines Case Study: A Source Code Comprehension Tool," *Proc. of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human Interaction: Making CHI Natural* Auckland, New Zealand: ACM Press, vol. 94, 2005, pp. 53-60.

34. J. A. Wise, J. J. Thomas, K. Pennock, D. Lantrip, M. Pottier, A. Schur, and V. Crow, "Visualizing the Non-visual: Spatial Analysis and Interaction with Information from Text Documents," *Information Visualization*, 1995, pp. 51 – 58.

35. M. Zyda, "From Visual Simulation to Virtual Reality to Games," *Computer,* vol. 38, 2005, pp. 25-32.

36. P. Richens and M. Trinder, "Design Participation through the Internet," *Architectural Research Quarterly,* vol. 3, 1999, pp. 1-14.

37. V. Miliano, [Available Online] "Unrealty: Application of a 3D Game Engine to Enhance the Design, Visualization and Presentation of Commercial Real Estate," *5th International Conference on Virtual Systems and MultiMedia* Dundee, Scotland, UK: International Society on Virtual Systems and MultiMedia, http://www.unrealty.net/vsmm99/, 1999.

38. B. Stang, "Game Engines Features and Possibilities," Institute of Informatics and Mathematical Modeling at the Technical University of Denmark, 2003, p. 1-33.

39. M. Lewis and J. Jacobson, "Game Engines in Scientific Research," *Communications of the ACM,* vol. 45, 2002 pp. 27-31.

40. K. T. L. Tran, "U.S. Videogame Industry Posts Record Sales," *Wall Street Journal,* p. B.5, Feb. 7, 2002.

41. Reuters, "U.S. Game Sales Hit $13.5 Billion in 2006," *News.com*: CNET Networks, Inc., 2007.

42. E. L. Wynters, "3D Video Games: No Programming Required " *Journal of Computing Sciences in Colleges,* vol. 22, 2007, pp. 105-111.

43. B. C. Wunsche, B. Kot, A. Gits, R. Amor, J. Hosking, and J. Grundy, "A Framework for Game Engine Based Visualizations.," *Image and Vision Computing*, vol. 1, 2005, pp. 465-470.

44. Jelsoft Enterprises Ltd., [Available Online] Devmaster.net, "3D Engines Database," http://www.devmaster.net/, 2006.

45. id Software, [Software] "Doom 3 Engine," ver. 1.2. Mesquite, TX, 2004.

46. M. S. El-Nasr and B. K. Smith, "Learning Through Game Modding," *Computers in Entertainment,* vol. 4, 2006, p. 7.

47. F. Emmerson, "Exploring the Video Game as a Learning Tool," *ERCIM News,* vol. 57, 2004.

48. M. Ibrahim and R. Krawczyk, "The Level of Knowledge of CAD Objects within the Building Information Model," *Annual Conference of the Association for Computer Aided Design In Architecture*, Indianapolis, IN, 2003, pp. 173-177.

49. T. Meigs, *Ultimate Game Design: Building Game Worlds*. New York, NY: McGraw-Hill/Osborne, 2003.

50. J. X. Chen, *Guide to Graphics Software Tools*. New York, NY: Springer-Verlag Inc., 2002.

51. D. Fritsch and M. Kada, "Visualisation Using Game Engines," *Geo-Imagery Bridging Continents XXth ISPRS Congress*, Istanbul, Turkey, 2004, pp. 621-625.

52. BeyondUnreal, [Available Online] “Package Extension Catalog”, "http://wiki.beyondunreal.com/wiki/Package_Extension_Catalog," 2007.

53. C. Fu, G. Aouad, A. Lee, A. Mashall-Ponting, and S. Wu, "IFC Model Viewer to Support nD Model Application," *Automation in Construction,* vol. 15, 2006, pp. 178-185.

54. Y. Adachi, [Software, Available Online] "IFCsvr," ver. R300, SECOM CO., LTD. Intelligent Systems Laboratory, http://tech.groups.yahoo.com/group/ifcsvr-users/files/IFCsvrR300/ifcsvrr300_setup_1008_en.zip, 2006.

55. Y. Adachi, [Personal Communication, E-Mail] "IFCtoMAP Conversion Utility," C. McDonald, College Station, TX, 2006.

56. Y. Adachi, [Spreadsheet, Available Online] "IFCsvrR200Tk_MAPcnv_000428.xls," http://tech.groups.yahoo.com/group/ifcsvr-users/files/ToolKit/Ifc2QuakeMap/IFCsvrR200Tk_MAPcnv_000428.zip, 2000.

57. Y. Adachi, [Software, Available Online] "IFCExplorer," ver. 1.0.0.1, SECOM CO., LTD. Intelligent Systems Laboratory, http://tech.groups.yahoo.com/group/ifcsvr-users/files/IFCsvrR300/Sample/setup_ifcexplorer1.zip, 2006.

58. K. Terzidis and D. Campbell, "Data-Stream Driven Distributed Virtual Environments: Air Quality Management District Visualization," *IBPSA 1999* Kyoto, Japan, 1999, cód. A-01.

59. L. P. Chew, "Constrained Delaunay Triangulations," *Proc. of the Third Annual Symposium on Computational Geometry* Waterloo, Ontario, Canada: ACM Press, 1987, pp. 215-222.

60. B. Worrall, [Doom 3 Script, Available Online] "func_rotatingdoor.script," http://www.doom3world.org/phpbb2/viewtopic.php?f=65&t=15463, 2005.

61. B. O'Sullivan and M. Keane, "Specification of an IFC bases Intelligent Graphical User Interface to Support Building Energy Simulation," *9th IBPSA Conference* Montréal, Canada, 2005, pp. 875-882.

62. A. Karola, H. Lahtela, R. Hänninen, R. Hitchcock, Q. Chen, S. Dajka, and K. Hagström, "BSPRO COM-Server– Interoperability between Software Tools Using Industry Foundation Classes," *7th IBPSA Conference* Rio de Janeiro, Brazil, 2001, pp. 507-514.

63. H. S. He, A. Hammad, and P. Fazio, "Application of IT and International Standards to Improve Building Envelope Performance," *9th IBPSA Conference* Montreal, Canada, 2005, pp. 389-396.

64. P. Bonsma, [Software, Available Online] "IFC Engine Viewer," ver. 1.10 Beta, http://www.ifcbrowser.com/downloads/BETA/IFC%20Engine%20Viewer.exe, 2005.

# APPENDICES

## Electronic Files

These items accompany this thesis as separate files available for downloading as 070607_CMc_LangfordBuildingB.zip and 070607_CMc_ThesisMod.zip:

- The file 070607_CMc_LangfordBuildingB.zip contains the Langford Building B file data as follows:
    - Langford Building B Revit File: LangfordBuildinB.rvt
    - Langford Building B IFC File: LangfordBuildinB.ifc
    - Langford Building B IDF File: LangfordBuildinB.idf
        - This is the output of the IFCtoIDF Converter which has been passed through the EnergyPlus Version Translation programs from version 1.2 to v1.3 to v1.4 to v2.0. Attached as the electronic file
- The file 070607_CMc_ThesisMod.zip, contains the Mod directory for the Doom 3 game engine and has been tested using Doom 3 and Prey. To use unzip the file into the Doom 3 and place the converted .MAP file into the ‘map’ subdirectory.

Source Code & Doxygen Documentation for IFCtoMAP these items accompany this thesis as separate files available for downloading as: 070607_CMc_IFCtoMAP_SourceCode.zip and 070607_CMc_Doxygen.zip:

- 070607_CMc_IFCtoMAP_SourceCode.zip includes the solution directory for the IFCtoMAP Visual Studio 2005 project along with a compiled version of the software

in the bin\release directory. IFCtoMAP Source Code. This includes the C# source code used to compile the IFCtoMAP program. In order to compile the solution Visual Studio 2005 or newer with the C# compiler is needed. To use the solution in Visual Studio 2005 decompress the zip file and select IFCtoMAP.sln.

- 070607_CMc_Doxygen.zip includes an HTML directory containing the output of a Doxygen run on the IFCtoMAP source code. To begin viewing this documentation select the file index.htm.

## Constrained Triangulation Example Figures 1-14

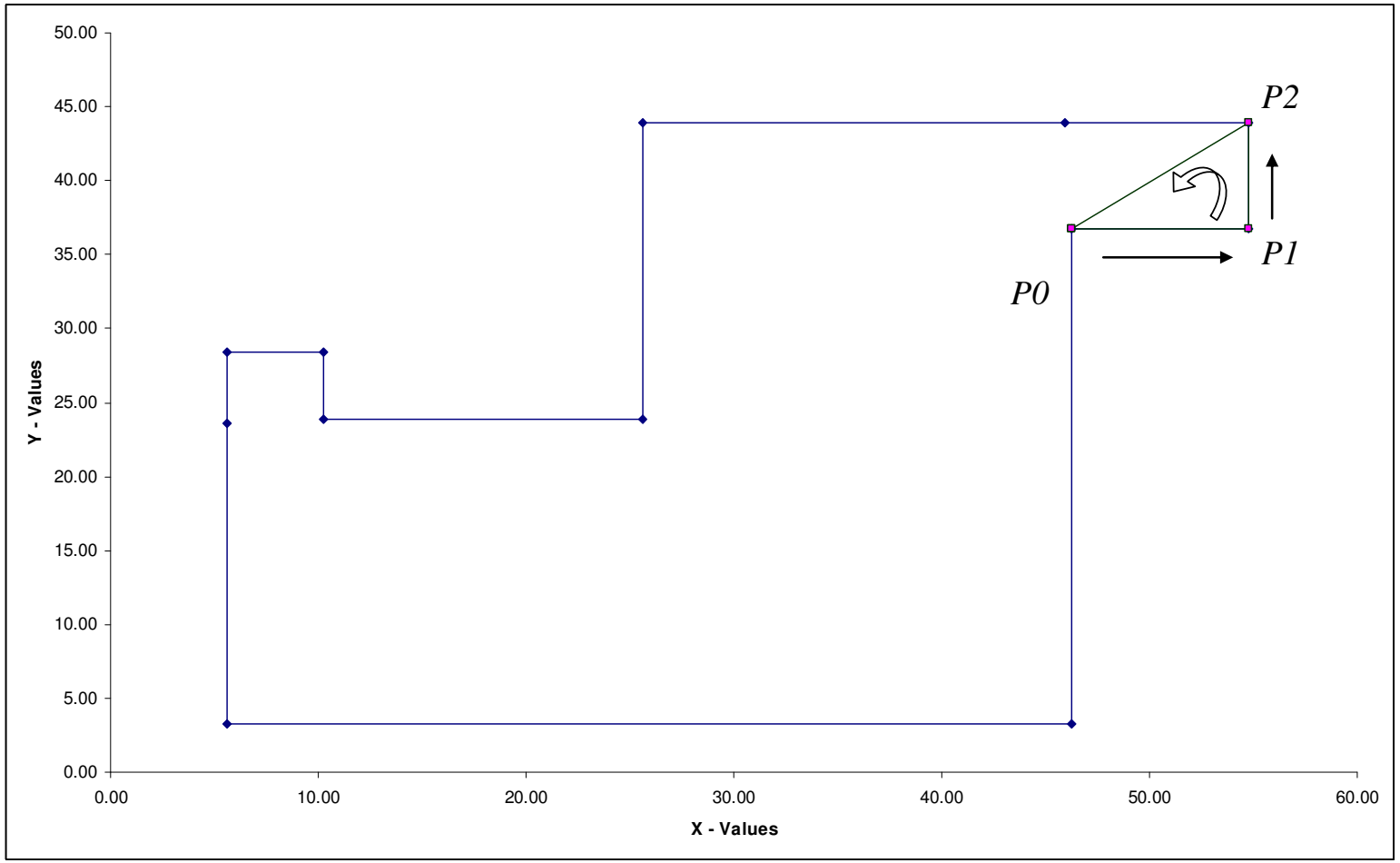


**Figure 1 Constrained Triangulation Example 1**

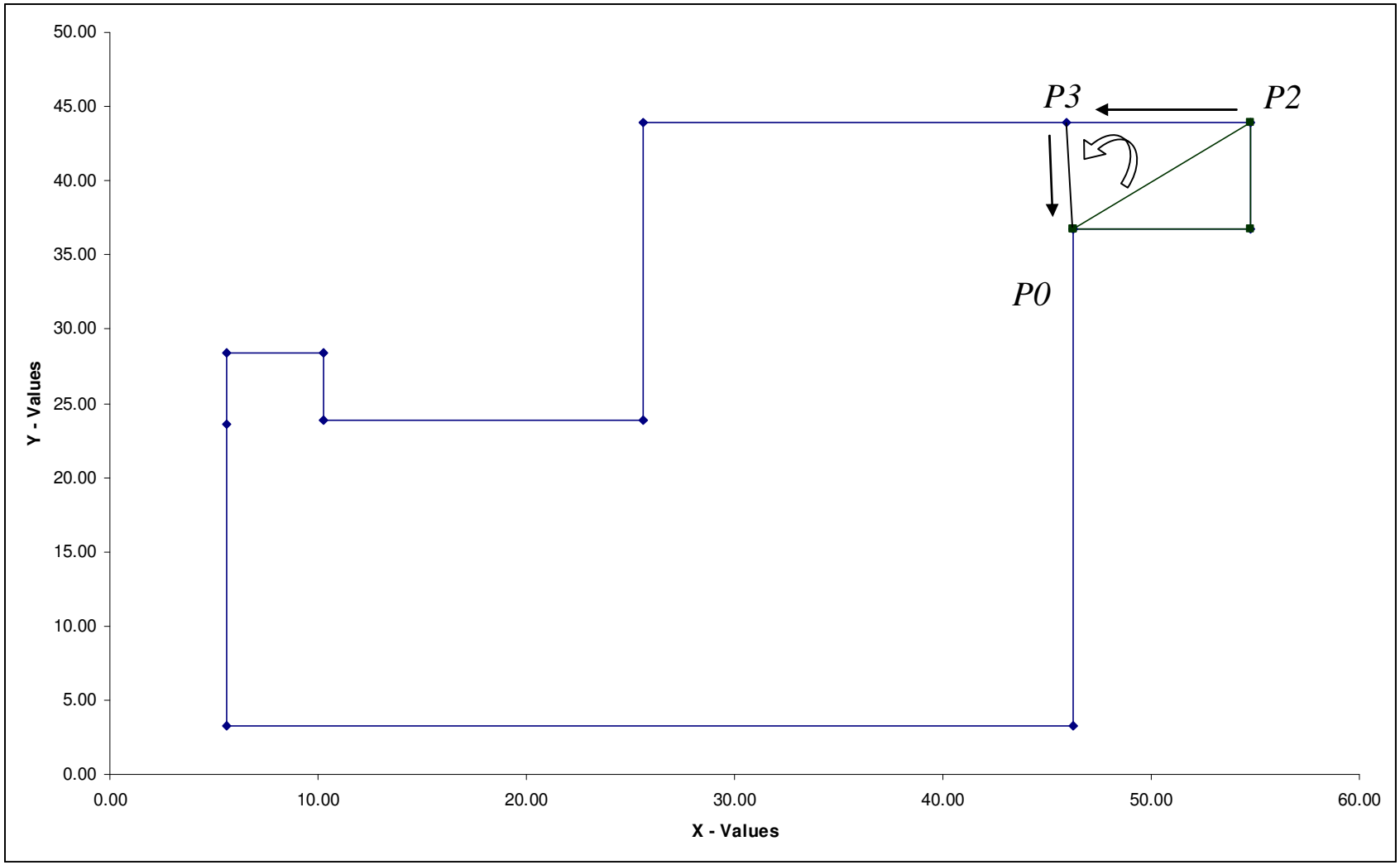


**Figure 2 Constrained Triangulation Example 2**

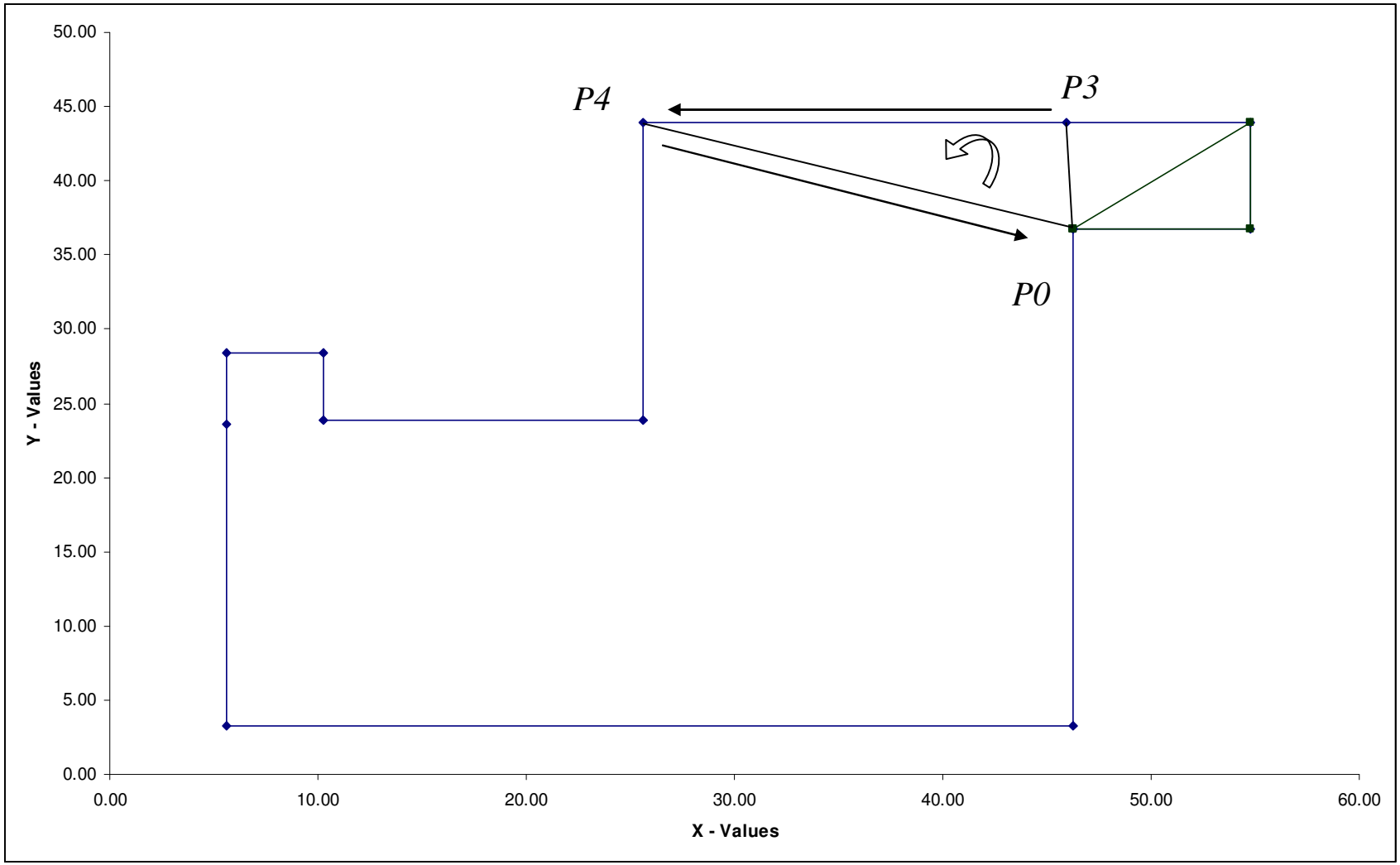


**Figure 3 Constrained Triangulation Example 3**

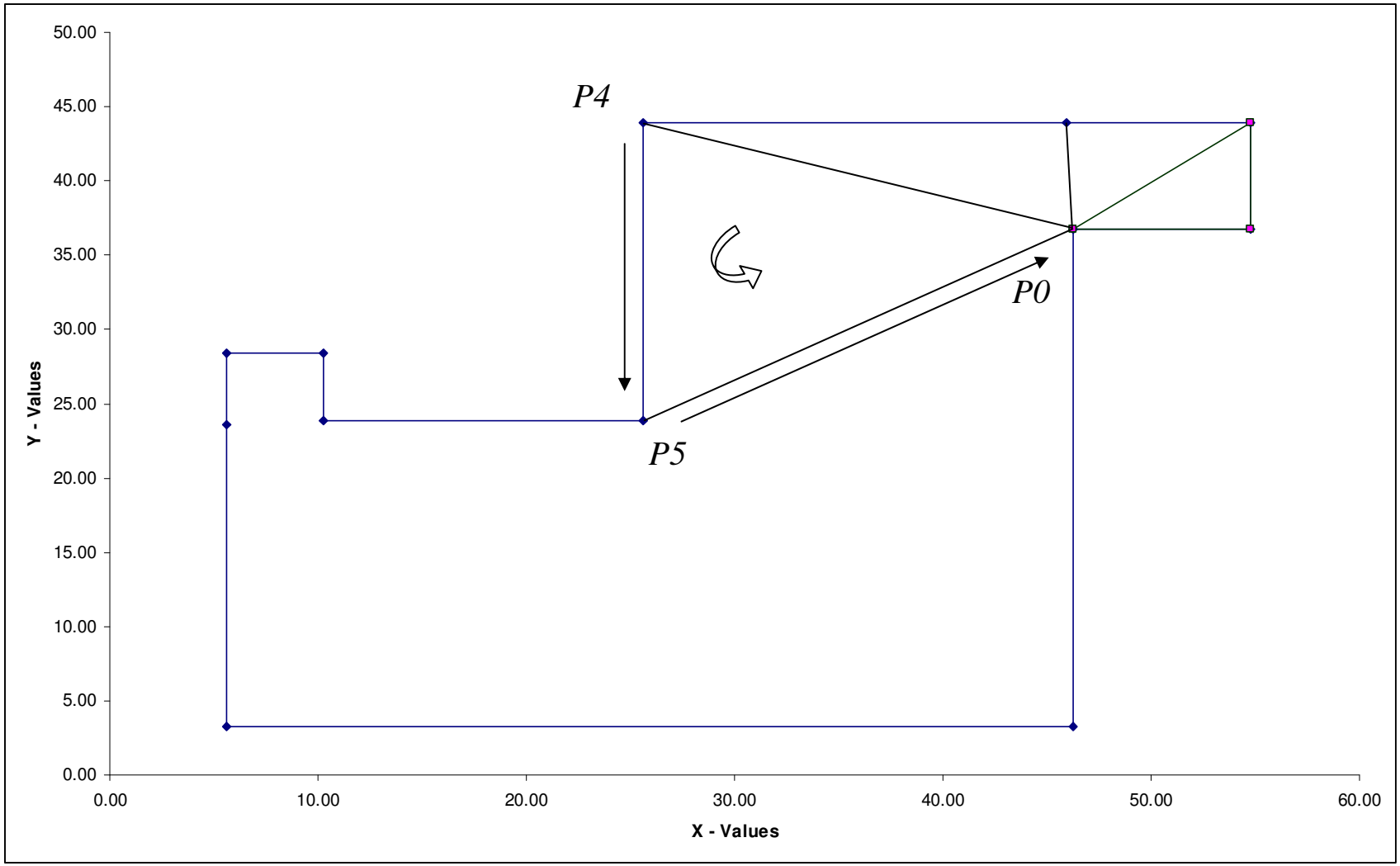


**Figure 4 Constrained Triangulation Example 4**

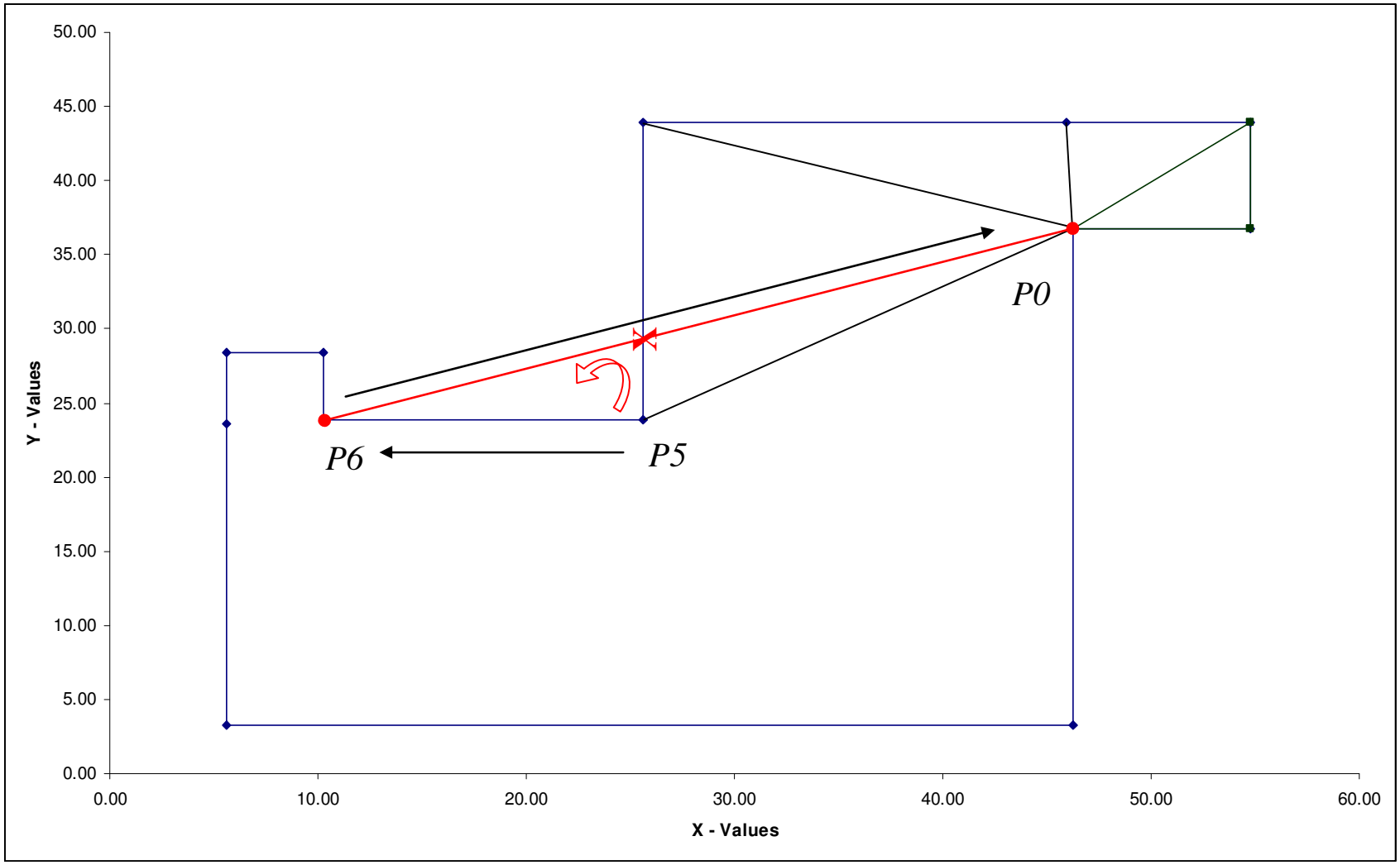


**Figure 5 Constrained Triangulation Example 5**

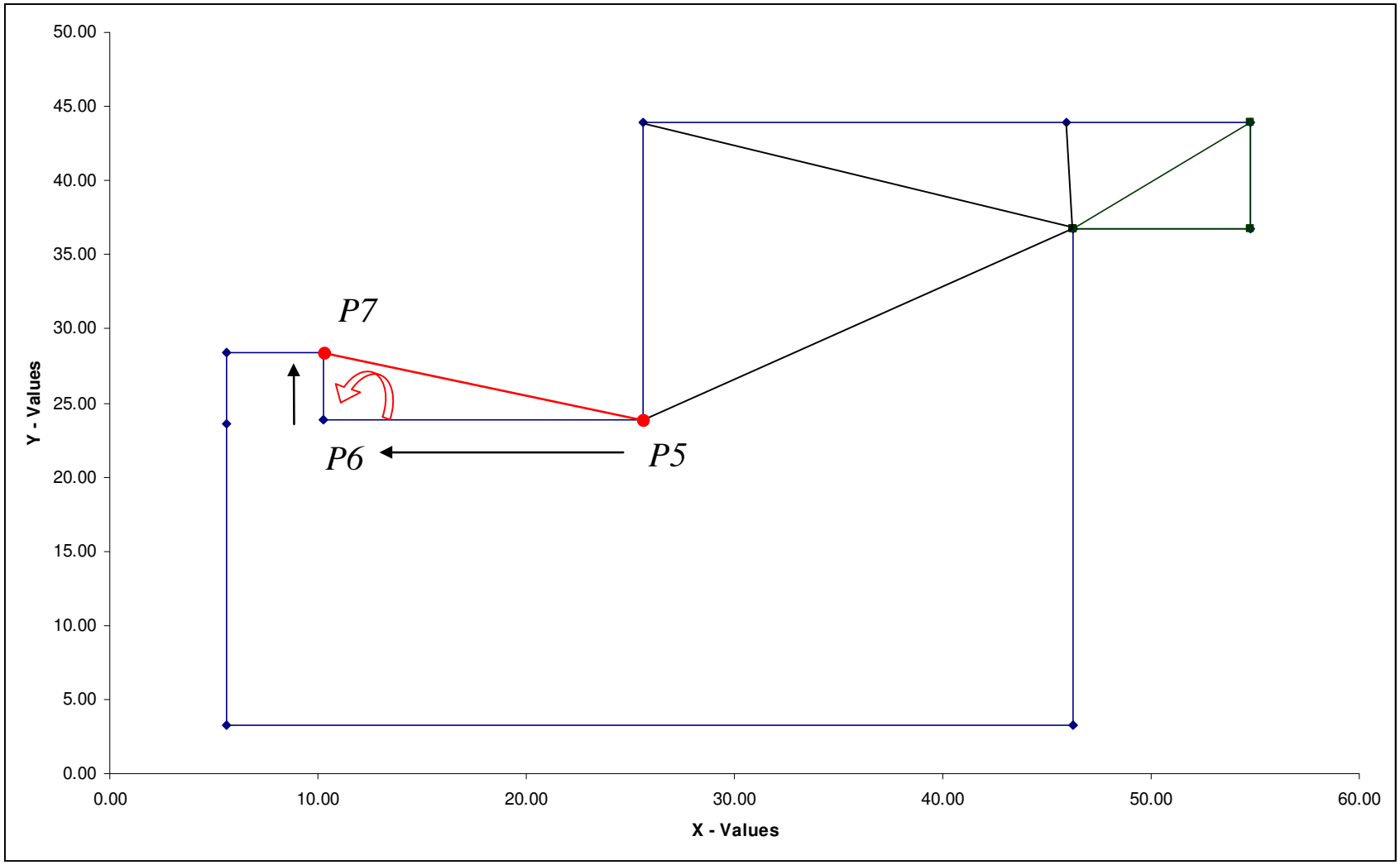


**Figure 6 Constrained Triangulation Example 6**

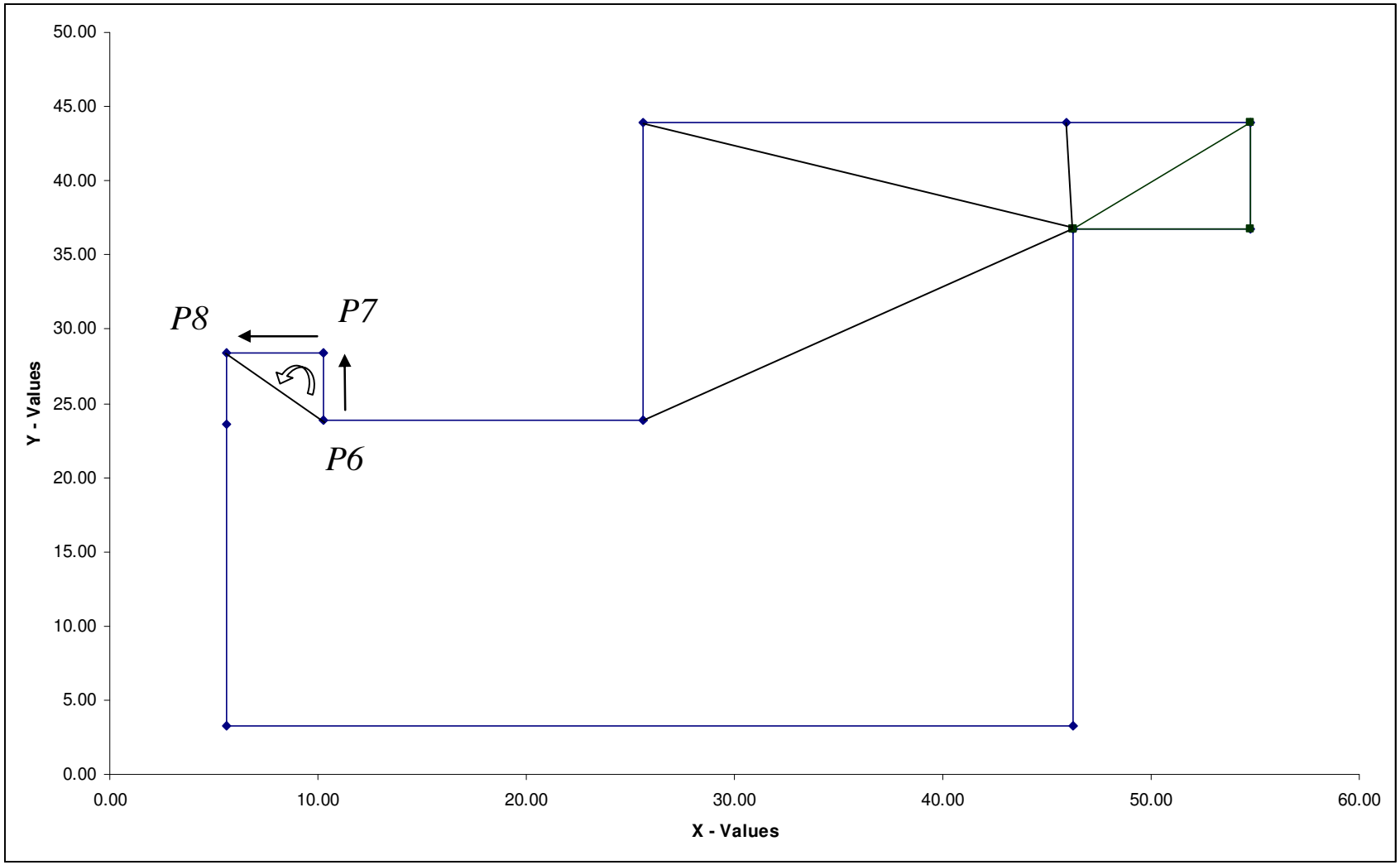


**Figure 7 Constrained Triangulation Example 7**

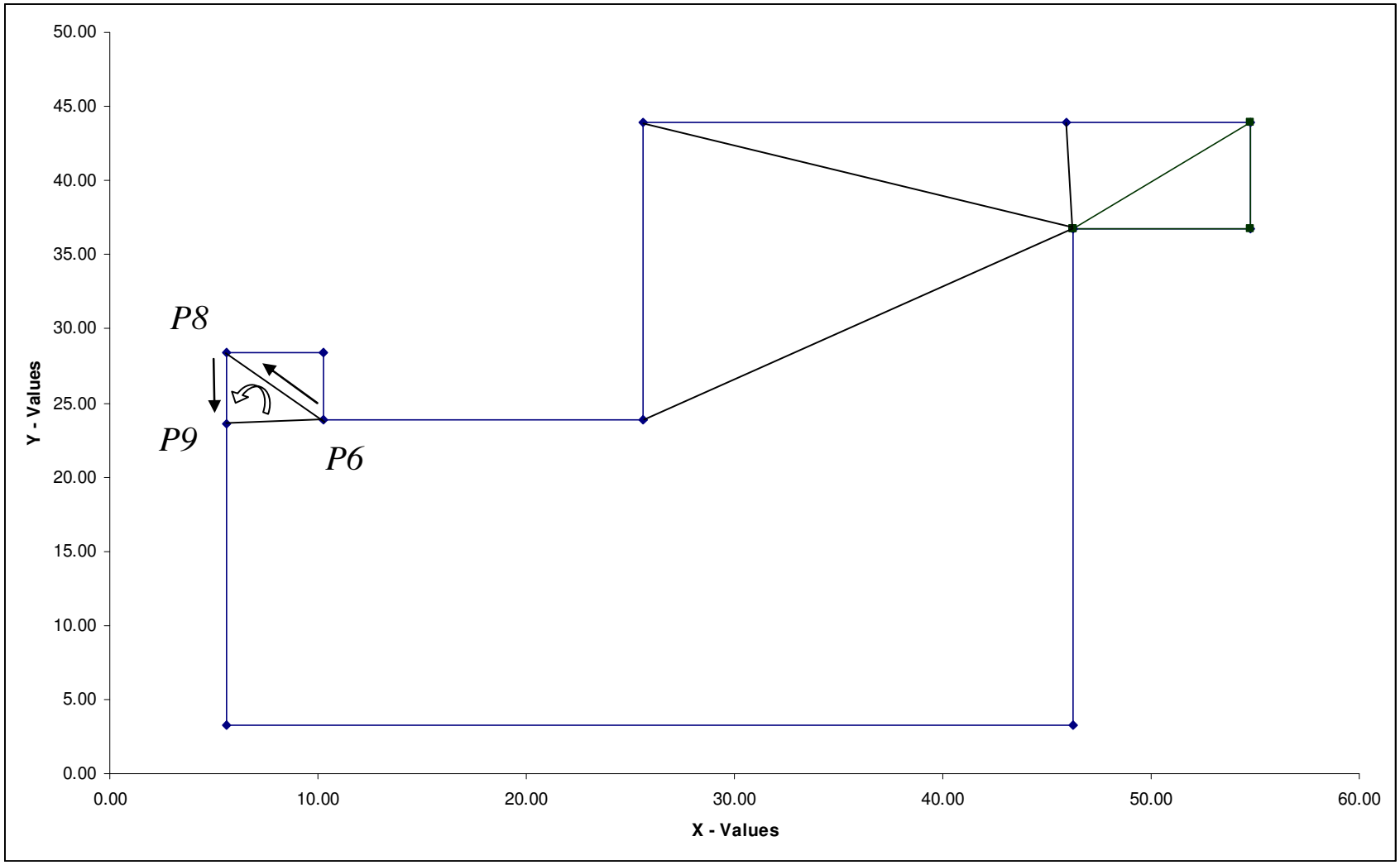


**Figure 8 Constrained Triangulation Example 8**

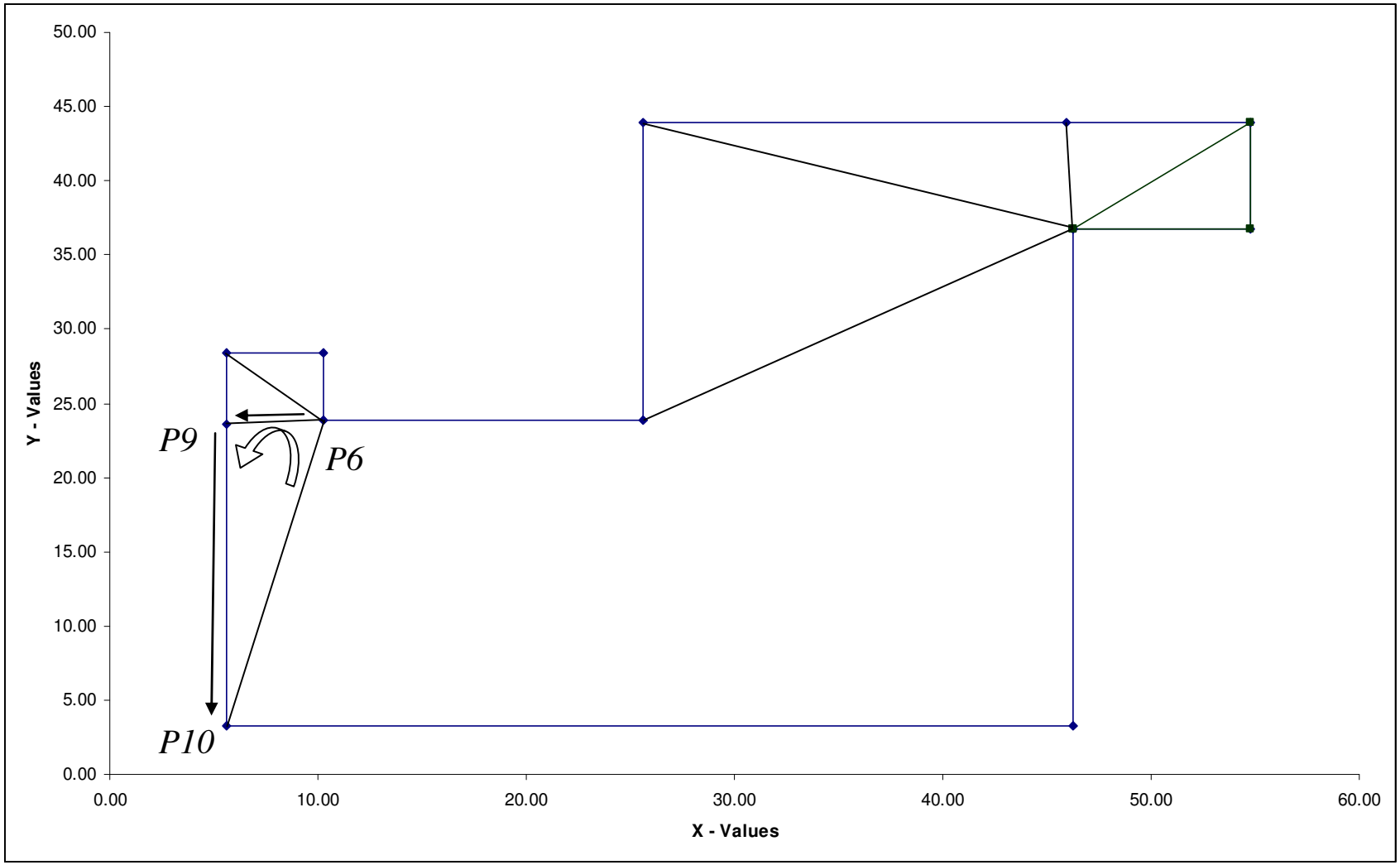


**Figure 9 Constrained Triangulation Example 9**

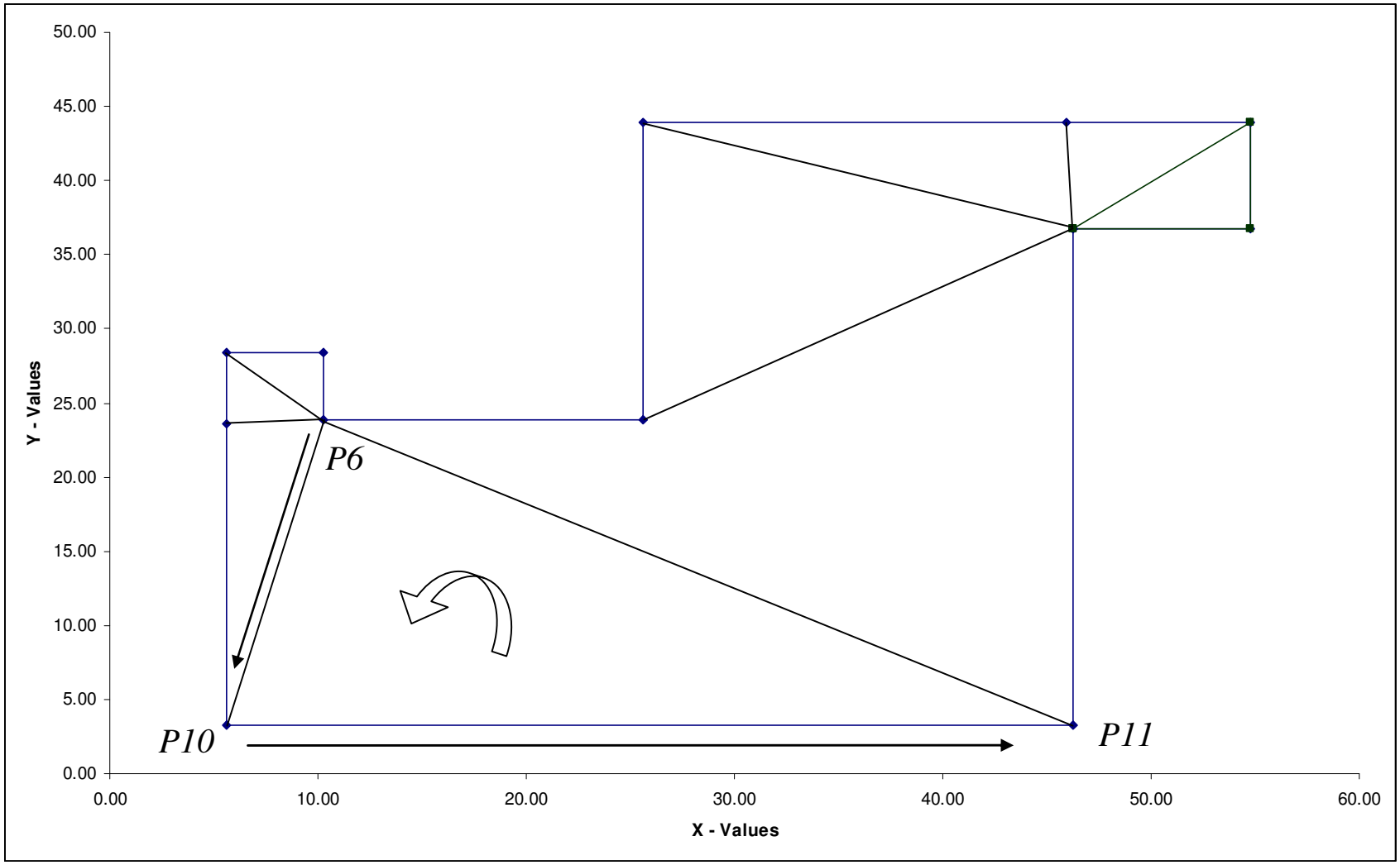


**Figure 10 Constrained Triangulation Example 10**

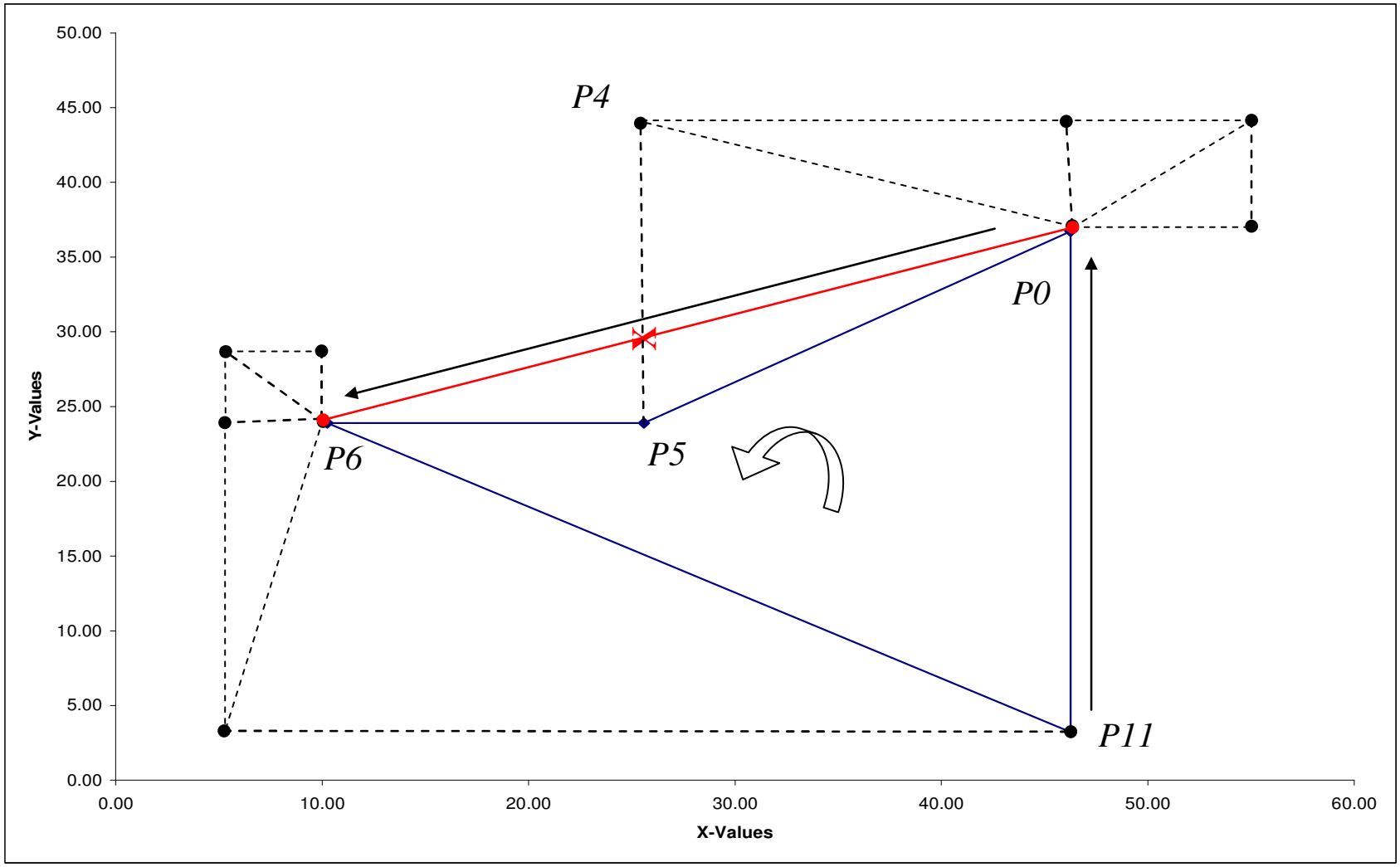


**Figure 11 Constrained Triangulation Example 11**

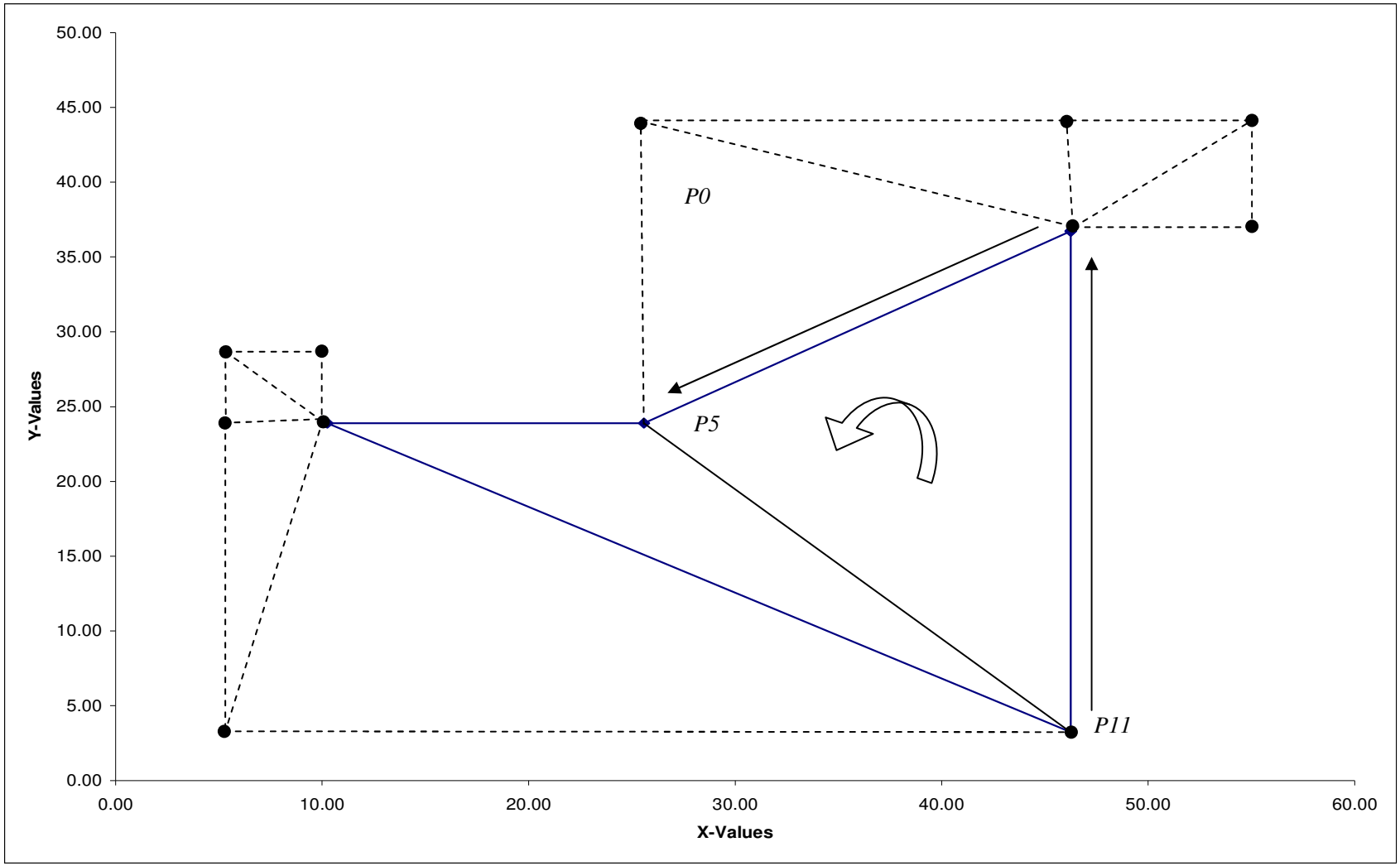


**Figure 12 Constrained Triangulation Example 12**

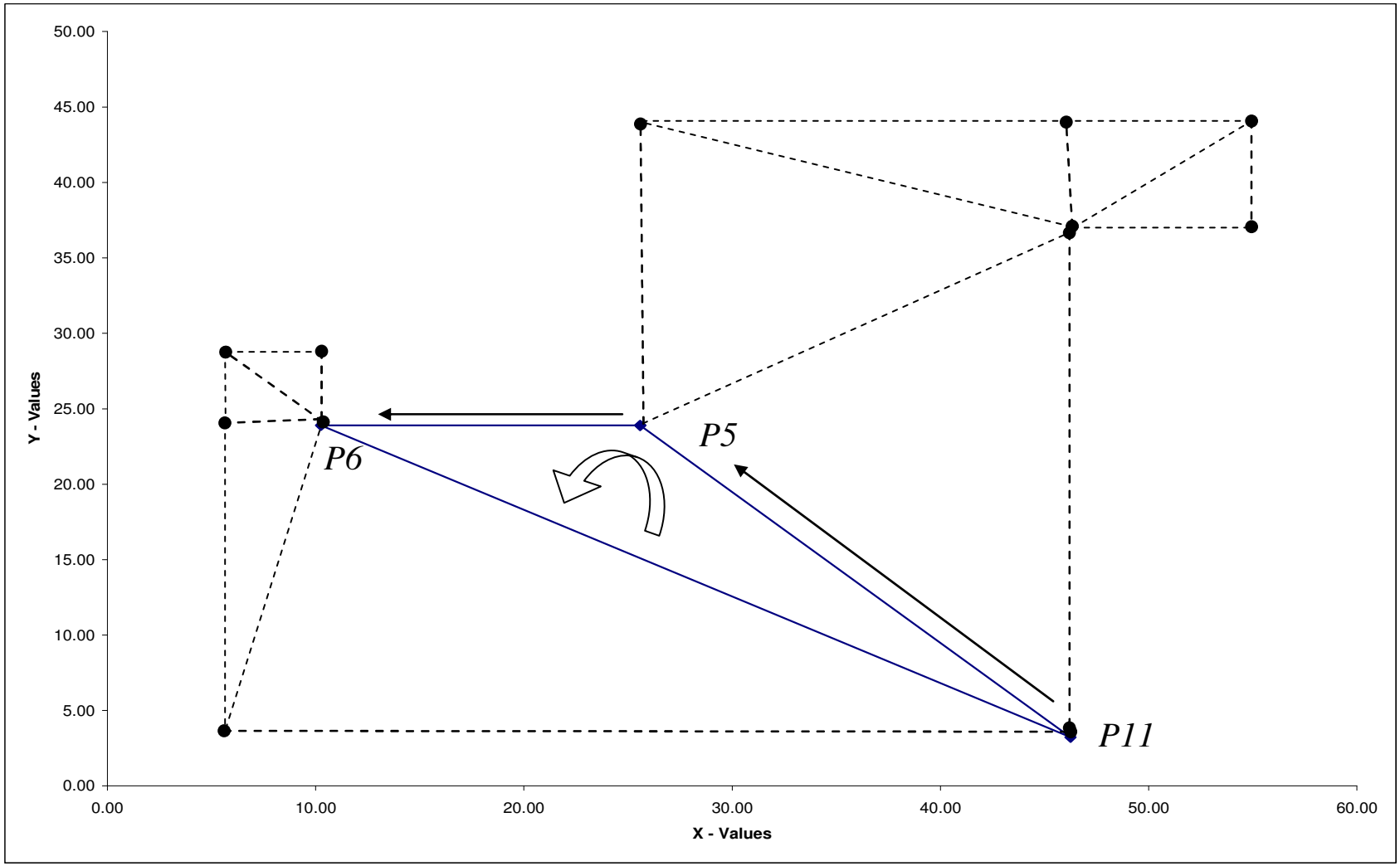


**Figure 13 Constrained Triangulation Example 13**

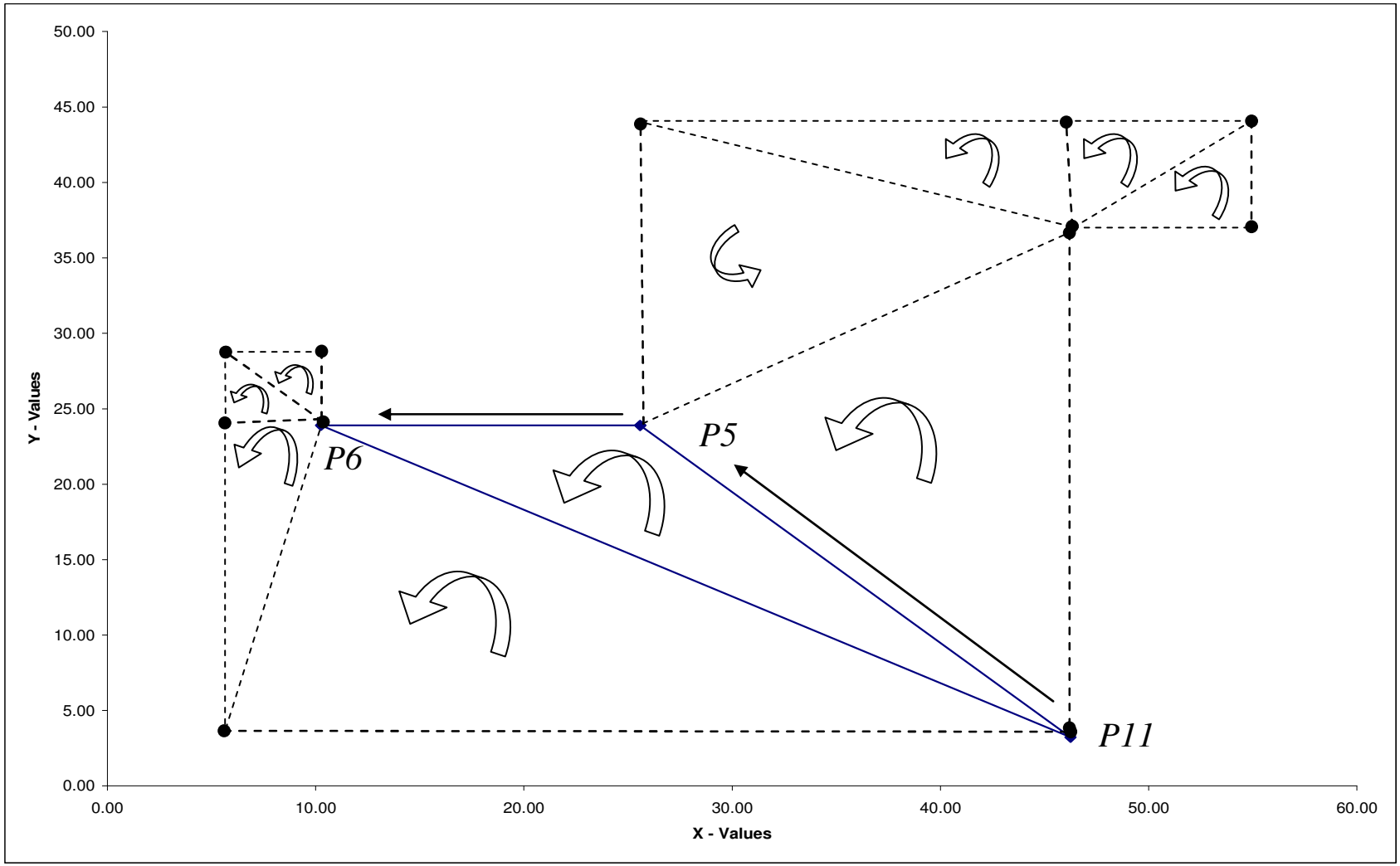


**Figure 14 Constrained Triangulation Example 14**

# Doom 3 System Requirements

- 3D Hardware Accelerator Card Required - 100% DirectX® 9.0b compatible 64MB Hardware Accelerated video card and the latest drivers*.
- English version of Microsoft® Windows® 2000/XP
- Pentium® IV 1.5 GHz or Athlon® XP 1500+ processor or higher
- 384MB RAM
- 8x Speed CD-ROM drive (1200KB/sec sustained transfer rate) and latest drivers
- 2.2GB of uncompressed free hard disk space (plus 400MB for Windows® swap file)
- 100% DirectX® 9.0b compatible 16-bit sound card and latest drivers
- 100% Windows® 2000/XP compatible mouse, keyboard and latest drivers
- DirectX® 9.0b (included)

MULTIPLAYER REQUIREMENTS:

- Internet (TCP/IP) and LAN (TCP/IP) play supported
- Internet play requires broadband connection and latest drivers
- LAN play requires network interface card and latest drivers

*Important Note: *Some 3D accelerator cards with the chipset listed here may not be compatible with the 3D accelerator features utilized by Doom 3. Please refer to your hardware manufacturer for 100% DirectX 9.0b compatibility. This product does not support Microsoft® Windows® 95/98/ME or NT.*

SUPPORTED CHIPSETS:

- ATI® Radeon(tm) 8500
- ATI® Radeon(tm) 9000
- ATI® Radeon(tm) 9200
- ATI® Radeon(tm) 9500
- ATI® Radeon(tm) 9600
- ATI® Radeon(tm) 9700
- ATI® Radeon(tm) 9800
- All nVidia® GeForce(tm) 3/Ti series
- All nVidia® GeForce(tm) 4MX series
- All nVidia® GeForce(tm) 4/Ti series
- All nVidia® GeForce(tm) FX series
- nVidia® GeForce(tm) 6800

# Langford Building B IFCtoIDF EnergyPlus Error File.

```
Program Version,EnergyPlus 2.0.0.025, 4/19/2007 3:11 PM,IDD_Version 2.0.0.025
   ** Warning ** GetSurfaces: Surfaces with interface to Ground found but no
"GroundTemperatures" were input.
   **   ~~~   ** Found first in surface=0HDUCC6BB75WLCEGCFDTDZ3SGAMKIGB1P8_OBRAYJHOD>FF0
   **   ~~~   ** Defaults, constant throughout the year of (18.0) will be used.
   ** Severe  ** GetSurfaceData: Zone has no surfaces, Zone=0HDUCC6BB75WLCEGCFDTDV2
   ** Warning ** GetSurfaceData:The total number of floors, walls, roofs and internal
mass surfaces in Zone 0HDUCC6BB75WLCEGCFDTDK
   **   ~~~   ** is < 6. This may cause an inaccurate zone heat balance calculation.
   ** Warning ** GetSurfaceData:The total number of floors, walls, roofs and internal
mass surfaces in Zone 0HDUCC6BB75WLCEGCFDTDQ
   **   ~~~   ** is < 6. This may cause an inaccurate zone heat balance calculation.
   ** Warning ** GetSurfaceData:The total number of floors, walls, roofs and internal
mass surfaces in Zone 37HSCZC0D1DBSPKQFCEYL1
   **   ~~~   ** is < 6. This may cause an inaccurate zone heat balance calculation.
   ** Warning ** GetSurfaceData:The total number of floors, walls, roofs and internal
mass surfaces in Zone 37HSCZC0D1DBSPKQFCEYL2
   **   ~~~   ** is < 6. This may cause an inaccurate zone heat balance calculation.
   ** Warning ** GetSurfaceData:The total number of floors, walls, roofs and internal
mass surfaces in Zone 37HSCZC0D1DBSPKQFCEYLF
   **   ~~~   ** is < 6. This may cause an inaccurate zone heat balance calculation.
   ** Warning ** GetSurfaceData:The total number of floors, walls, roofs and internal
mass surfaces in Zone 37HSCZC0D1DBSPKQFCEYL8
   **   ~~~   ** is < 6. This may cause an inaccurate zone heat balance calculation.
   ** Warning ** GetSurfaceData:The total number of floors, walls, roofs and internal
mass surfaces in Zone 37HSCZC0D1DBSPKQFCEYKR
   **   ~~~   ** is < 6. This may cause an inaccurate zone heat balance calculation.
   ** Warning ** GetSurfaceData:The total number of floors, walls, roofs and internal
mass surfaces in Zone 37HSCZC0D1DBSPKQFCEYJM
   **   ~~~   ** is < 6. This may cause an inaccurate zone heat balance calculation.
   **  Fatal  ** Fatal error discovered in GetSurfaceData, see previous messages
   ************* Fatal error -- final processing.  More error messages may appear.
   ************* Testing Individual Branch Integrity
   ************* All Branches passed integrity testing
   ************* Testing Individual Supply Air Path Integrity
   ************* All Supply Air Paths passed integrity testing
   ************* Testing Individual Return Air Path Integrity
   ************* All Return Air Paths passed integrity testing
   ************* No node connection errors were found.
   ************* EnergyPlus Terminated--Fatal Error Detected. 9 Warning; 1 Severe Errors;
Elapsed Time=00hr 00min  2.94sec
```

# VITA

Name: Christopher Ernest McDonald

Address: U.S. Department of State 2201 C Street NW Washington, DC 20520

Email Address: mcdonaldce@gmail.com

Education: B.S. Computer Engineering, Texas A&M University, 2005
M.S. Architecture, Texas A&M University, 2007