A Computational Approach to the Design of Free Form Diagrid Structures

by

Jessica Nicole Sundberg

B.S.A.D. Department of Architecture, S.B. Sloan School of Management
Massachusetts Institute of Technology, 2008

SUBMITTED TO THE DEPARTMENT OF CIVIL AND ENVIRONMENTAL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING IN CIVIL AND ENVIRONMENTAL ENGINEERING
AT THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2009

Signature of the author _____
Department of Civil & Environmental Engineering
May 15, 2009

Certified by _____
Jerome J. Connor
Professor of Civil & Environmental Engineering
Thesis Supervisor

Accepted by _____
Daniele Veneziano
Chairman, Departmental Committee for Graduate Students

A Computational Approach to the Design of Free Form Diagrid Structures

by

Jessica Nicole Sundberg

Submitted to the Department of Civil and Environmental Engineering on May 15,2009, in partial fulfillment of the requirements for the degree of Master of Engineering In Civil And Environmental Engineering

ABSTRACT

In order to satisfy the ever-increasing complexity of modern architectural design of tall towers, diagrid structural systems are becoming more relevant. To deal with irregular geometries, more sophisticated computational models are necessary.

This study establishes a method for integrated design of diagrid structures for complex building geometries. The process is outlined through an architectural method and a structural method with automated intermediary systems in between. The iterative process is then evaluated and recommendations are made for the development of the proposed method. Based on the design example, general guidelines for a general computational approach to the design of more free form structures are to be generated.

Thesis Supervisor: Jerome Connor
Title: Professor of Civil and Environmental Engineering

ACKNOWLEDGEMENTS

This thesis would not have been possible without the contribution of many people.

First, I'd like to thank Simon Kim and Skylar Tibbits for their troubleshooting help and guidance throughout their RhinoScripting seminar during MIT's Independent Activities Period. Without their instruction, much of the coding behind this thesis would not have been possible.

I am thankful to my advisor, Jerome Connor, for letting me pursue my interests in both Architecture and Structural Engineering through this somewhat unconventional thesis. His insight and lessons throughout the year have been invaluable. Secondly, I'd like to thank Simon Laflamme and Todd Radford for their assistance with some structural software inquiries.

I would like to thank my parents and my sister, Katrina, for their continuous support throughout both my undergraduate and graduate years at the Institute. Their encouragement to always pursue my life goals has brought me to where I am today and will continue to inspire me into tomorrow.

Finally, I'd like to thank all of this year's Course 1 MEng students—especially the HPS students. This year has been, by far, my most enjoyable year at MIT, and I look forward to continuing these friendships beyond graduation. Rose, Lauren, and Isabel—thank you for making my year all the more enjoyable and the long hours at work bearable.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

With the advancement of technology, the world of architecture is progressing at a rapid pace. As the design process becomes increasingly digital, firms handle their data in different ways. While representing geometry in three dimensions has become easier, firms have the opportunity to increase complexity of design to break barriers of traditional forms. With more complex forms, other challenges arise. First, it becomes more difficult to describe a design with the traditional 2D documents customarily provided by the architect. Secondly, coordination of disciplines along the chain of construction—from engineer through contractor—becomes increasingly more challenging. The digital capabilities vary by firm and by discipline. More specifically, the way building data is managed and provided is not necessarily standardized, creating an immense problem for project coordination. However, the software advances to date have opened the door to new possibilities for a more integrated design approach. In order to appreciate these advancements, it is beneficial to take a closer look at a more typical design process.

1.1 The Typical Process

The standard design process before integrated design or software advances was chaotic. The disarray of the organization of disciplines reflects the quality of the coordination of data. Figure 1 below represents the linear design process that lacks an integrated approach or iterations that span more than one discipline.

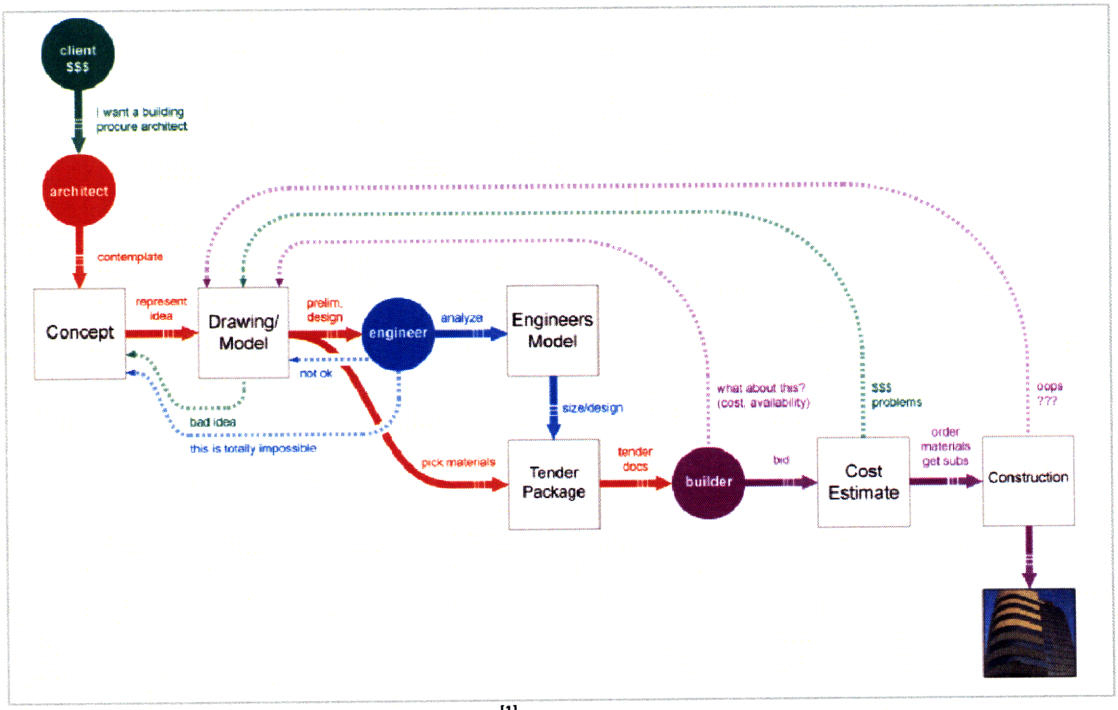


Figure 1 Representation of standard design process[1]

Figure 2 represents the chaotic process for the design of Zaha Hadid's Phaeno Science center with the structural engineering firm AKT of the UK—as analyzed in a case study through the Harvard Graduate School of Design[1]. This approach attempts to fit a very atypical project into a traditional process method model.

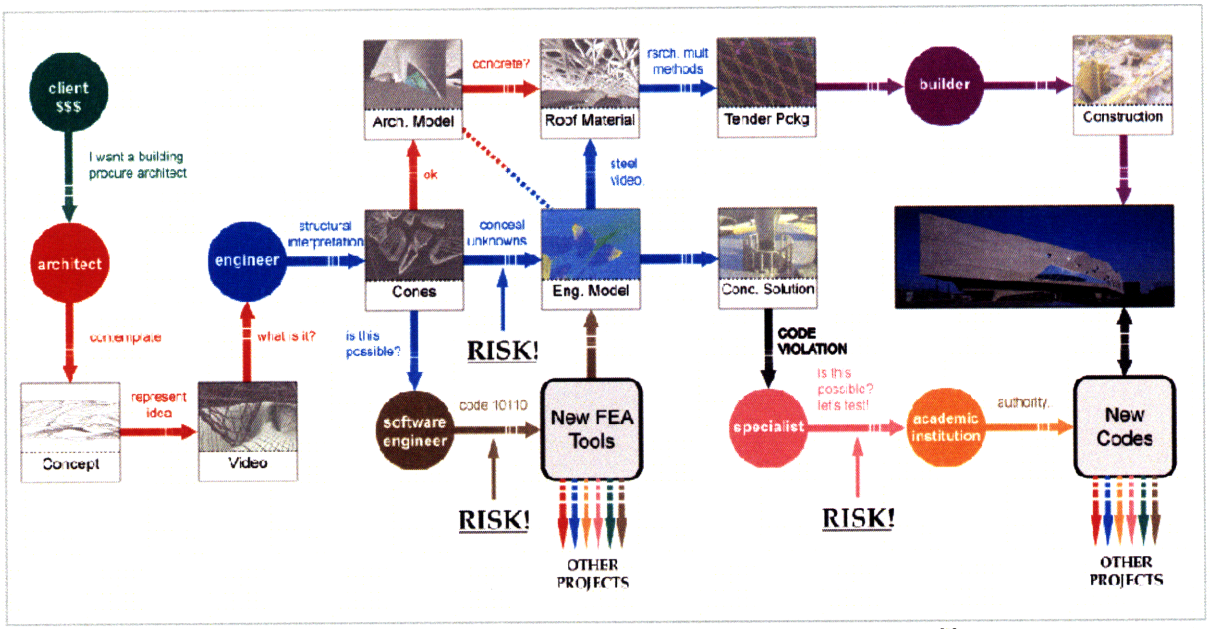


Figure 2 Representation of design process for Phaeno Science Center using contextual technology[1]

Since the completion of the Phaeno Science Center, software has been written which begins to coordinate the system development for buildings as conducted by multiple disciplines.

1.2 Building Information Modeling (BIM) and its limitations

Building Information Modeling, better known as BIM, is a method of modeling building data that is used from the architectural design stages through construction. The method was established due to the strong desire to better coordinate data and improve on the standard processes used in design projects.

The method represents a way of generating and managing data. The models generated with the specialized BIM software, such as Autodesk's Revit, contains information on the building's site location, basic geometry, and quantities, properties, and relationships of building elements. The models can be passed around between disciplines and used to manipulate in real-time through discipline-specialized design software packages.

Before BIM, each party did not always deal with data in the same fashion. Architects would have surface models of their buildings while engineers would have CAD models of elements representing columns, beams, and surfaces. Construction managers would value yet another kind of information than

the architect or the engineer, and clearly, not all parties spoke the same data 'language.' BIM is a first attempt to consolidate all of the necessary information into one useable model.

Although the model is standardized so that all parties participating in the design and construction process can deal with the data appropriately, all parties must still design independent of one another. The model must be passed along from discipline to discipline and although changes of the information will update, the structural design of elements will require the structural engineer to have a model of the final architectural model. Similarly, the contractors will need a final model from the structural engineer before performing the tasks required of them, and so on and so forth.

Ultimately, the attempt of BIM to advance integrated design through parametric modeling has proved successful, yet it is only an elementary step.

1.3 What is Parametric Modeling

One positive feature of the BIM system is that it allows for parametric modeling. By understanding parametric modeling, predictions for the future of integrated design can be made.

Parametric modeling is a very broad term. Possibly the most overused word in architecture these days, parametric modeling really just represents an approach to design where all attributes of a design are given parameters that can be changed. The real value arises when those parameters can be altered without adding work for the individual designer.

Computer programs such as Digital Project and Grasshopper for Rhinoceros 4.0 deal with parametric data. The software recognizes elements as having certain arrays of properties. Also, the elements have relations to one another that can likewise be defined in parameterized ways. Altogether, changes can be made to the model which update a certain parameter or number of parameters. Once a change is made, the model will adjust itself based on the relations and rules that the user has embedded in the system.

For example, in a typical CAD drawing, if a node was to be moved, the elements coming into that node would remain in their original place unless also moved to the new endpoint location. With a parametric model, that nodal relationship can be defined and instead, all elements would automatically move when the endpoint is controlled.

The model's ability to update itself while maintaining certain qualities allows the user to make extreme change at unheard of speeds compared to a decade ago. With this increased speed, the designer now has the ability to refine the design with greater ease and compare schemes more efficiently. Also,

the inherent data and attributes of the digital elements has the potential to be useful in analysis and evaluation further along in the design process.

1.4 Where the field is heading

BIM is only the beginning. It is possible and becoming more popular for engineers to take digital models from designers and to manipulate them for structural design. Engineers can now take parametric models from an architect, and through computer programming, specialize their design software to handle the input material. This streamlines the design process from the former BIM model.

An even more streamline approach would exist if the original design software that the architect is using has the ability to intelligently design a structural system. To start, this study is not implying the extinction of the structural engineer as an entity in the design process. It is merely looking at the process at face value and suggesting where shortcuts *could* be made.

By customizing software to have inherent structural knowledge, design iterations can be made by the architect without seeking an unnecessary response from the engineer. The collaboration of the architect and engineer can be completely integrated—as long as the structural system is predetermined and the particular code from which it is generated is written. As long as all ingredients are readied, the entire iterative design process can run smoothly and be controlled by decisions and opinions of the leading architect. Furthermore, this will have implications for the liability and responsibility for the safety of the design.

1.5 Intent

The following implementation attempts to take a structural system of particular interest and, starting at the initial architectural concept design level, iteratively design the structural system within the available software. Using coding tools such as Rhinoscript and VisualBasic, the preliminary framework for a more automated system was created. Furthermore, the limitations of this particular model will be discussed along with the limitations of the computing concept itself.

A preliminary presentation of an iterative design model was given a few months ago to a panel of architects, software engineers, and artists. The feedback on the initial intent, to create a piece of software that takes an input of form and outputs a structure, was surprising. The architects wanted a tool that they could learn from; they did not favor such a black box. It was for this reason that the more iterative, design-oriented decision making loop was created.

This approach is meant to be a rough model of what is possible with further software development to integrate architectural design and structural design software. However simplified the software is, the process still has the ability to handle whatever complex surface geometry the drawing software, Rhinoceros 4.0, can create.

The following text is organized according to the order of the iterative computational process. First, some background information on the structural advantages of the diagrid system is presented. Following that, there are details regarding the way that the architectural computer program handles geometric data in its inherent coordinate systems. The subsequent chapters outline the process of automation on both the architectural side and structural side. After implementation, the limitations and opportunities of this approach will be discussed, and further applications of this idea on other architectural works will be addressed.

2 Why a Diagrid

The diagrid structural system was chosen as the starting point for this entire inquiry. Many buildings, mainly tall structures, have been built using the diagrid. Norman Foster's Hearst Tower in New York, his 30 St Mary Axe (the Swiss Re Tower) in London, and KPF's design for the World Product Center in New York City on the following page are all examples of tall buildings utilizing this structural system. The main characteristic of these buildings to note is their relatively simple geometry. Each building is rather regular in plan which is either square-like or circle.

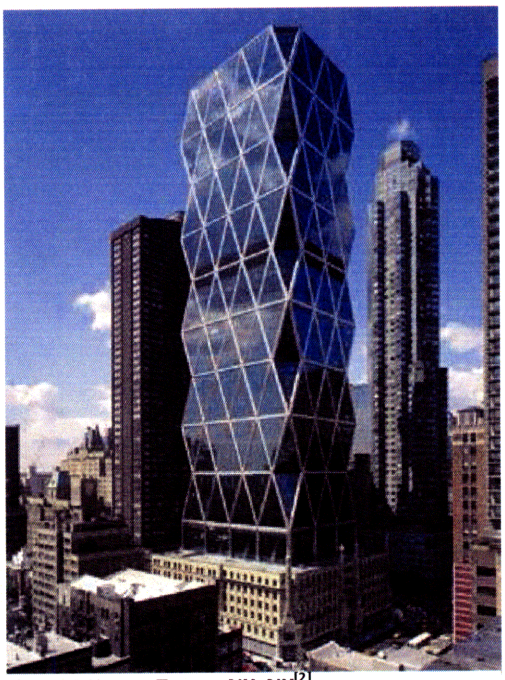


Figure 3 Hearst Tower, NY, NY[2]

2.1 Structural Advantages

In a diagrid structural system, most vertical and lateral loads of the building are taken by the exterior diagrid elements (ignoring the core). This allows for column-free floor plans which are ideal in tower design to provide the optimal rentable space—large column-free areas. The greatest advantage of the diagrid structure is its angled elements. These allow it to take both the vertical and lateral loads to improve stiffness of the structure. A stiffness-based approach to design is most fitting when considering a diagrid structure.

Figure 4 Swiss Re Tower, London, England[3]

When considering a basic motion based design, the stiffness of the system can be calculated based on the angles of the members[5]. In a regular diagrid system, the stiffness would be defined as the following in the horizontal direction:

$$K_H \cong \frac{AE}{h} \sin\theta \cos^2\theta$$

Likewise, the stiffness in the vertical direction is the following:

$$K_V \cong \frac{AE}{h}\sin^3\theta$$

Comparing these values to the stiffness of the equivalent rigid frame or braced frame structure, the diagrid provides relatively the same stiffness as the braced frame structure while eliminating the vertical members and thus reducing the amount of material.

While eliminating material, the exclusion of the vertical members also allows for a more interesting geometric pattern for the structure. The diagrid can be used to fit a more interesting, irregular surface.

Figure 5 Image Credit KPF World Product Center, NY, NY[4]

System	Lateral Stiffness
Rigid Frame, Pinned	$K \cong \frac{12EI}{h^3}$
Braced Frame	$K \cong \frac{AE}{h}$
Diagrid	$K \cong \frac{AE}{h}$

Table 1 Lateral stiffness of common structural types[5]

2.2 Versatility of Form

If the angles of the diagrid are reworked and not necessarily uniform, the diagrid has the potential to be fit to a variety of forms—forms that are not as regular as the diagrid structures that exist today.

As previously stated, architecture is heading in a direction where form becomes more complex, and the digital tools available are not the only limitations on complex designs. Somehow, they must be built, and somehow, they must stand up. A less regular diagrid—one of varied angles in [x,y,z]-space—could be used to fit more complex tower geometries while still providing lateral stiffness and a vertical load carrying system for the building.

Although this system seems best for more geometrically complex architecture, it creates a complex mathematical problem for the structural design. Having geometry that is extremely irregular immediately places the design into the realm of computing. This problem becomes ideal for solving using parametric modeling. The question of what is the best way to model and design a free form building with a diagrid system prompted the extensive investigation that follows.

3 Methodology

The following chapter follows the method and implementation of the automation process. The design iteration cycle in Figure 6 begins with the development of the tower geometry in the Rhinoceros 4.0 software package. From there, the script is run and, based on user input, the desired diagrid is drawn on the surface. That geometry, along with wind directionality data is exported and run through an intermediary software. Next, the data is opened in the structural analysis package, and the simulation is run. The output is then forwarded to the intermediary software which determines whether the goals were met. If so, a final image of the geometry is rendered and the process is complete. If the results do not compare favorably, then the user is prompted to make a decision to either change the geometry or density of the diagrid. From there, the data cycles through again until the output satisfies the deflection limits, and the process is complete.

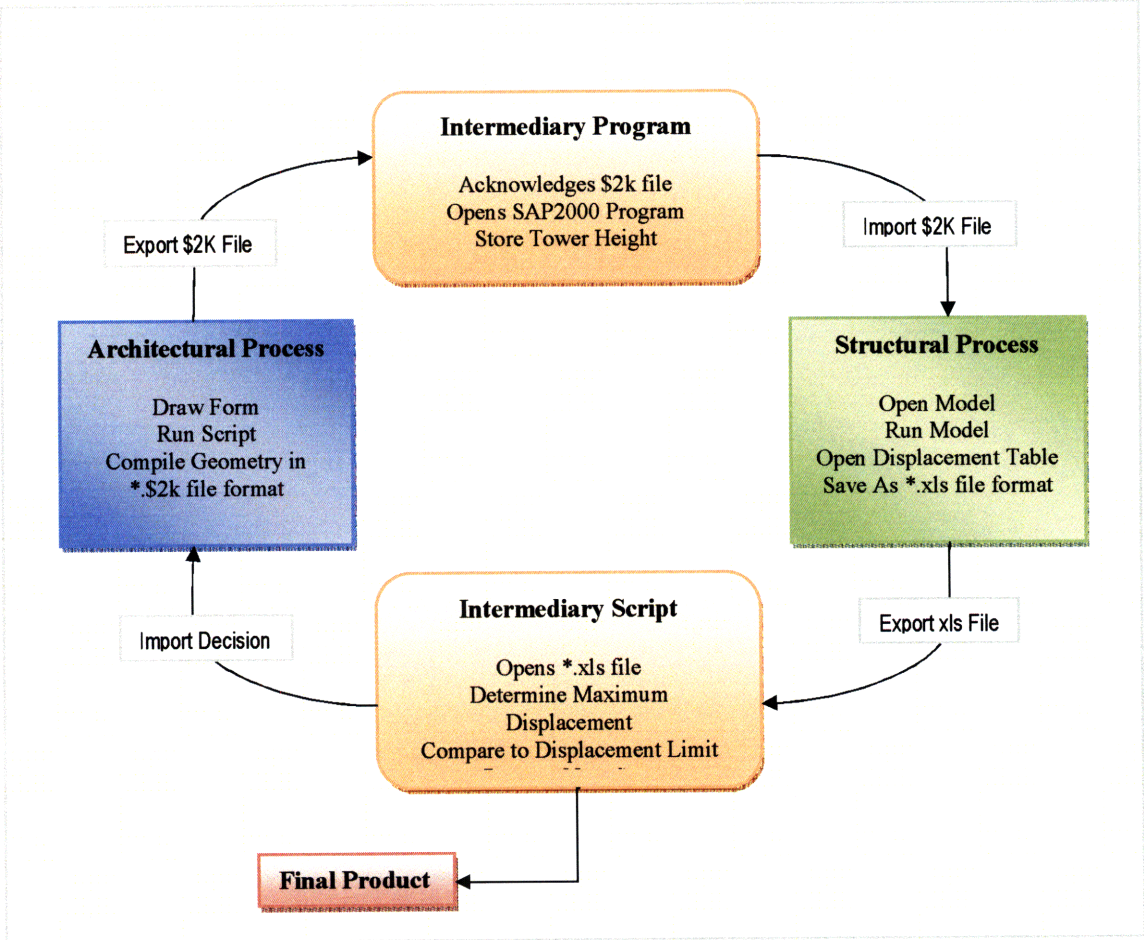


Figure 6 Representation of the Integrated Design Computation

3.1 Methods of Automation

In order to accomplish the automation for the desired process cycle above, multiple approaches to computer programming must be taken. The depth of scripting, or the extent to which programming plays a role, has multiple levels as seen in Figure 7. Design automation can be used in standard Microsoft Office products mainly using the VisualBasic language, specialized as VBA, through macros. Design programs such as Photoshop allow for certain automation of actions for tools or data file manipulation processes. Other programs such as Rhinoceros 4.0, Maya, 3DStudioMax and Catia allow for integration with the previous programs along with their own internal coding opportunities.

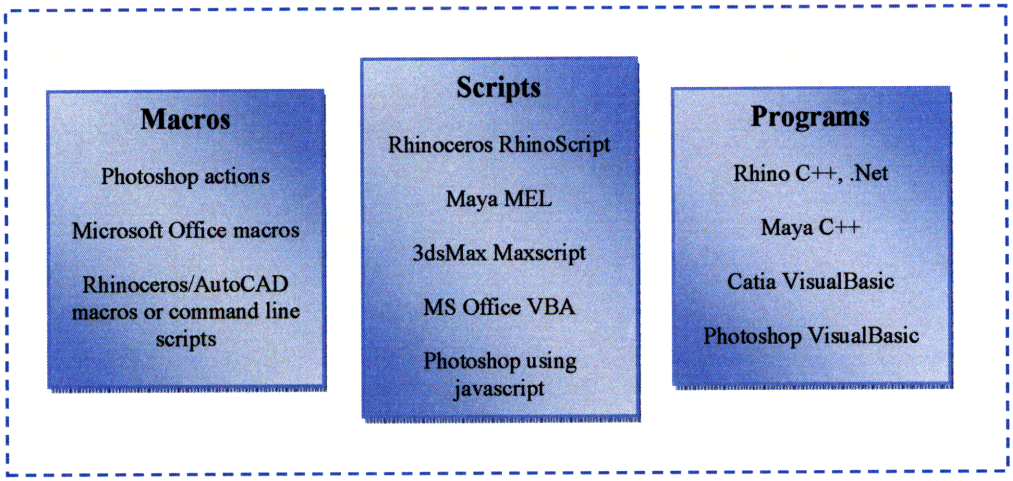


Figure 7 Methods of Automation

To accomplish the automation of the desired integrated design process for the diagrid, Rhinoceros RhinoScript, Microsoft Office VBA, and the Java programming language will be most appropriate. The Java and VisualBasic can handle the translation between programs as well as the initiation of the structural model within SAP2000. In this study, the majority of data manipulation was done within RhinoScript due to resource limitations and time restrictions. VisualBasic use within Excel will also be outlined as part of the iterative system.

3.2 Architectural Input

In order to apply a diagrid to the tower, the code recognizes only a specific type of surface which a certain directionality. In order to satisfy the geometrical constraints, the information that the architectural model provides must be translated in two ways (see Figure 8).

The Rhinoscript code can treat surfaces as having their own coordinate system, a system based off of [u,v] coordinates that move along the surface, rather than the Cartesian [x,y,z]-coordinates on which the model sits. In some cases, it is easier to perform operations along the surface using these coordinates rather than their [x,y,z] counterparts. With relatively few lines of code, the [u,v] coordinates can be determined for each point along the surface and the data stored similarly to the corresponding [x,y,z] coordinate. Both [u,v] and [x,y,z] coordinates were essential to the development of this script.

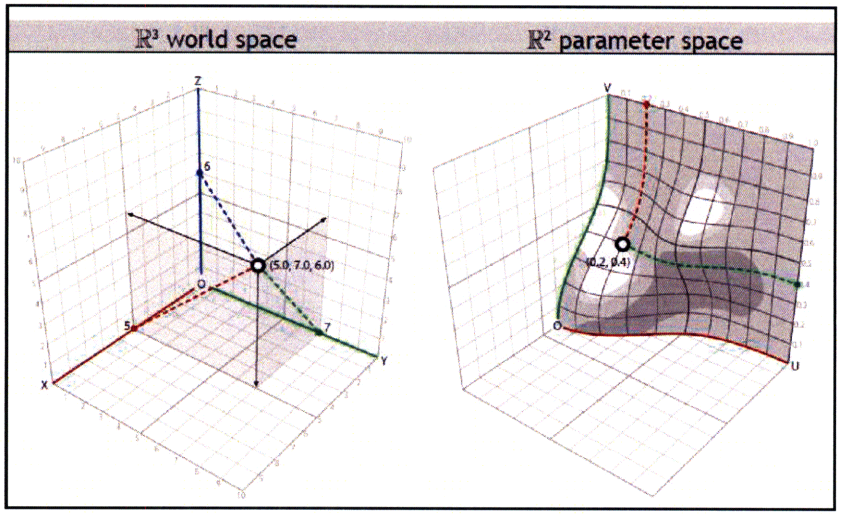


Figure 8 Cartesian versus [u,v] coordinate space in Rhinoceros 4.0[6]

In order to take advantage of these different coordinate systems within the script, the surface must be drawn in a particular fashion. The script will only work if the surface's v-axis is closed in on itself—meaning that the v-axis is the line that traces the plan of the building. This line is segmented into equal intervals based on the user input for the number of diagrid modules which will be demonstrated. For more complex geometry, this axis may change shape as one moves higher on the building surface causing the intervals to change in size, but the polyline at a constant v-value will always be defined as having the same number of intervals. The directionality of the v-axis does not matter; it can be clockwise or counterclockwise. It is crucial that the u-axis be in the direction in which the tower is reaching. Because the script is based off of conditional statements and loops along both u and v, it is critical that these dimensions are not confused.

The language of RhinoScript is derived from VisualBasic. The basic syntax is the same; however, RhinoScript requires less diligence when assigning variables and other object definitions. In general, RhinoScript is a watered-down version of its parent coding language, yet its simplicity ensures ease in comprehending what is required.

4 Architecture of the Code

The pseudo-code to the right (Figure 9) outlines the basic functions that are called through the main method of the addDiagrid script—the step of the process that applies a user defined set of lines that represent the diagrid of interest. This organization is the foundation for the systematic process that will be discussed in detail in the following sections. The code consists of a Main function. Within the main function, there are necessary basic user inputs along with methods that call specialized functions that are not inherent to RhinoScripting. These functions are written outside of the Main Sub method. The location where variables are defined is important and affects the accessibility of a variable at other points in the code.

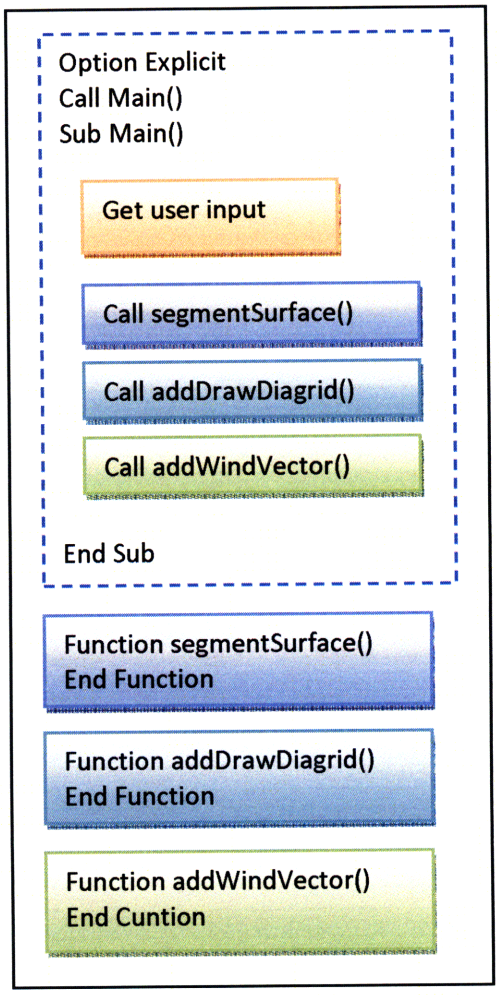
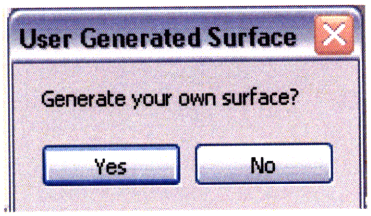


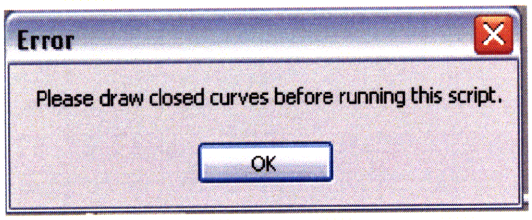
Figure 9 Pseudocode development

An example that follows will be developed in parallel with the development of the coding process in this thesis. Before any geometry is processed, the code prompts the user for certain inputs that prove the user is satisfactorily prepared to continue on with the script.

The main purpose of the script that has been created is to run an algorithm on a user-generated surface. After selecting to run the code, the user is prompted with the following:



The user has the option to select the default, which is yes, or no to generate his or her own surface. If No is selected, a second window will appear recommending the user draw original geometry before running the script.

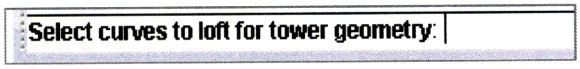


For help with this, there are also other scripts that can be developed and run that will generate pre-determined geometry. One such code was designed to generate a tower in the shape of a hyperbolic paraboloid which also serves to generate a plain cylindrical tower if no such tapering along the height is selected (see Appendix). The creation of the user-generated surface and the specifics behind this particular script are laid out in the following sections.

4.1 Embedded Geometry

To start, five closed curves have been drawn at heights of 0, 75, 150, 225, and 300 meters. These curves can be of any shape, but they will define the topology of the surface at the particular points of interest along the building's height. They must all be drawn in the same direction, either clockwise or counterclockwise because their directionality will affect the success of the subsequent lofting function. Hence, they ultimately determine the design of the building. The bottom curve must be entirely on the xy-plane, but the curve that defines the top of the building need not fall on a horizontal plane. The code will truncate what is in excess of the final floor of the building, assuming each story to be 3.5 meters in height; however, this will be described in further detail in a subsequent section. That way, the building is interpreted to have a consistent diagrid for the fully inhabitable floors below the cutoff point.

Once all of the curves that define the design are drawn on the xy-plane as in Figure 10, the user must manually move these curves to their proper height along the tower geometry. Note here that a curve must fall on the plane $z = 0$ and also dictate the top of the tower limits. Once these curves are in place, the script may be opened and its run sequence initiated. The user is prompted with the following in the Command line:



The curves must be selected in rising order—meaning the z = 0 curve is selected first and so on and so forth upwards along the tower. The order of selection is critical since the surface geometry must be formed with specific directionality in its [u,v] system. Once all curves have been selected, the entry is finalized by pressing the "Enter" key. Immediately, the script will loft the curves and draw in the surface that fits them all. The result of running the first segment of the script is shown in Figure 11 below.

From here on out, the program will consider this surface the geometry of choice for all operations. If the user is dissatisfied, he or she can opt to draw the surface on their own by manually lofting the curves *before* choosing to run the script. That way, the final geometry can be analyzed more thoroughly from all angles before choosing to run a script that will add numerous elements—lines and surfaces—to the drawing coordinate system. Once a manual surface has been chosen, the user should delete this surface or move it to another layer before running the script since the script first asks for the user to select the necessary curves to define what is to become the lofted surface.

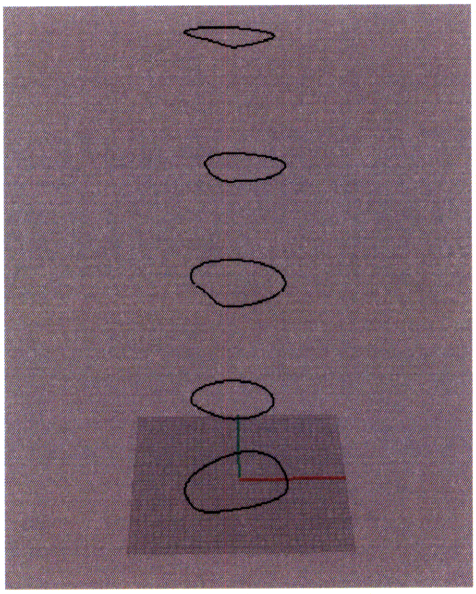


Figure 10 Design Example user-generated curves

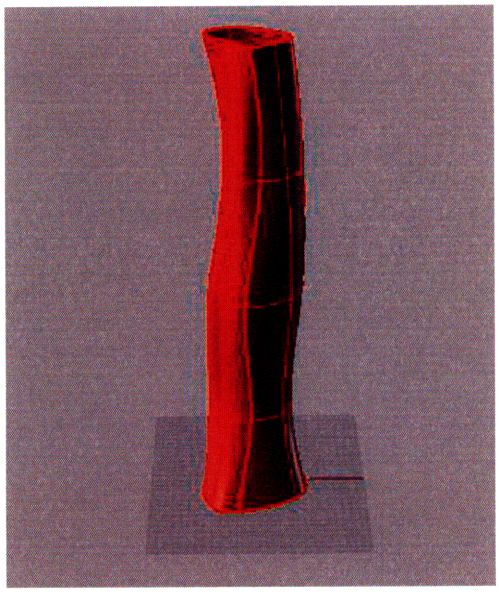


Figure 11 Design Example loft from curves

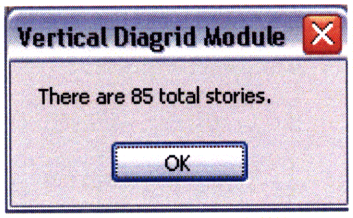
4.2 Modular Algorithms

Before the user is prompted for any more information or decisions, the script will analyze the surface and determine how many stories of height 3.5 meters are within the building. Modular arithmetic is used to discard the remainder at the top—which signifies a height that is less than the pre-determined allowable floor heights. If the 3.5 meter assumption is to be changed, the user can go back into the script and change this number [lines 54, 55, 106]. Note that in some cases, the number has been multiplied by ten in order to complete the modular arithmetic. Now, assuming that each floor height is 3.5 meters, the script will calculate the total number of full stories and output that number as part of the next prompt to the user.

Once the surface is generated by lofting the user-drawn curves and the unseen calculations are run, the script will round off the height of the tower to the closest number of stories without going over. Modular arithmetic is performed, and the height in excess of intervals of 3.5 meters will be disregarded. This is the first of a series of modular arithmetic operations in order to simplify the height of the tower to discrete, user-defined segments. The variable *intStories* is defined here to be the number of floors that are available within the buildings maximum height, *intHeight*. In this calculation, the variable *intRem* stores the remainder from the modular arithmetic. Note, the calculation has been multiplied by 10/10 so that the modular arithmetic is possible since modular operations require whole numbers.

$$intRem = ((10*intHeight)\ Mod\ 35)/10$$
$$intStories = ((intHeight - intRem)/(7/2))$$

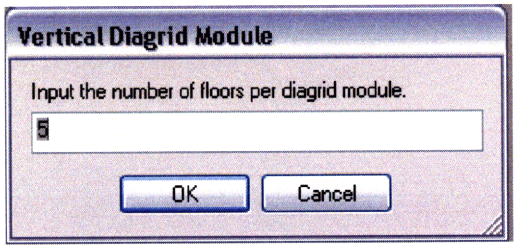
First the script will let the user know how many total stories there are in the building so that he or she may make a more educated decision about the number of floors in each vertical diagrid module.



The variable *intStories* stores the calculated number of stories and can be used to pass a numerical value into text as shown above; the variable *intStories* will appear to the user in numeric form within the first dialog box. In this example, a tower of height 300 meters will have a final story count of 85 stories, and

the prompt will read as seen above. This shows that the modular arithmetic reduces what would be 85.7143 stories to 85 in order to work with reasonable real life values.

After clicking O.K., the user will then input the number of stories per diagrid module. The script will ask:



This chosen input number will reflect the number of floors that one diagrid triangulation will span across vertically. Figure 12 demonstrates the difference between diagrid modules of two, three, and four floors respectively. (2 modules = 1 diamond)

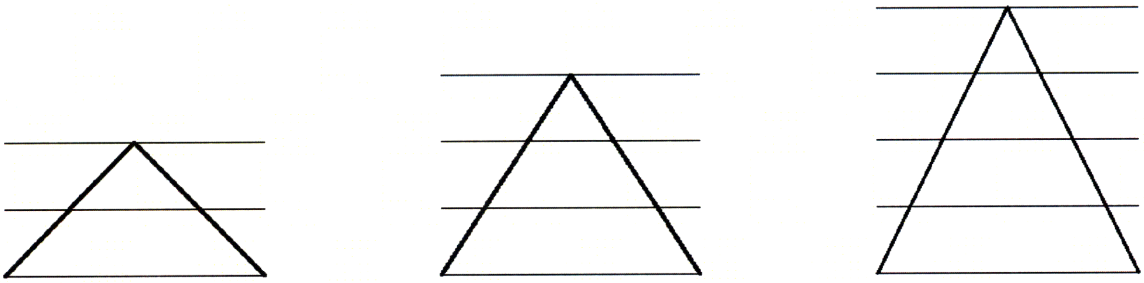


Figure 12 Diagrid Module Variation for 2, 3 or 4 stories

The code specifies the number of total stories in the current prompt so that the user has the necessary information to select a reasonable number of floors for the input. For example, the division can be selected so that the end appearance serves a specific aesthetic purpose for the building.

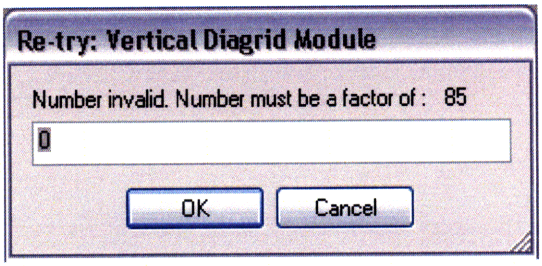
If the user inputs a number that is not a factor of the total number of stories, the script will perform an operation to see how far that number deviates from a factor of the total number of stories. The script performs modular arithmetic on the input variable *intGridZ*. The operation loop is based off of the function *intStories* Mod *intGridZ*. This line of code returns the remainder of floors that do not fit in a

diagrid module at the top of the tower. The loop from this code can be found in the Appendix in lines 59-66.

First, the script tests to see if the number of stories is exactly divisible by the number of floors that the user inputs for their choice of diagrid module. If this is true, the script will determine the total number of vertical divisions of the tower by dividing the number of stories by the number of floors per diagrid module.

If it is false, the script next checks to see if the excess floors are significant enough to redefine the structural module. An arbitrary test was designed to see if the number of excess floors exceeds 50% of the chosen module. If the remainder is less than half of the diagrid module choice, the script will just disregard these floors and assume a smaller structural system can be implemented for those top few floors.

If the remainder is greater than half of the diagrid module input from the user, the script responds with an error message that gives the user another opportunity to choose a second choice for factor of diagrid division. It will return:

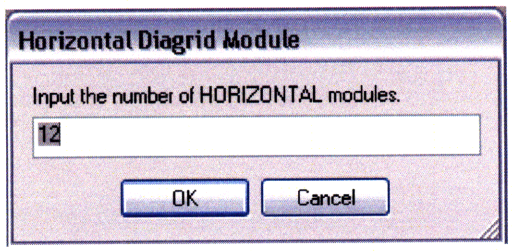


This gives the user a second opportunity to select an appropriate module for the diagrid. Here, the script assumes that the user will perform the necessary math to make sure this calculation is sufficient; otherwise, the script fails and aborts. Hopefully, an exact factor of the total number of stories will be chosen.

For the building with a height of 300 meters, the script will first prompt the user with the dialog boxes on pages 22 and 23. There are three possible outcomes. i) If the user inputs "4," the code will skip to the first else statement to find that the remainder is one which is less than fifty percent of four. This will be an acceptable diagrid module, and the remaining single story (one quarter of four) at the top will be discarded from the system. ii) If the user inputs "8," then the code will skip to the first Else statement, and the remainder will be five. The five stories is not less than the fifty percent allowable remainder (five

is roughly sixty-two percent of 8). Therefore, the user will get the error message asking them to re-enter a value that is an acceptable factor of the total number of stories. iii) If the user had input "5," the modular division will work out cleanly, and there will be seventeen total modules with vertical reach of five stories along the tower. For this example, the diagrid will span five floors in the vertical direction. In comparison, the Hearst Tower in New York City has four floors per diagrid module with 9 total modules for the building[2] while the Swiss Re Tower has two floors for every one of its over 20 modules[3].

There is one more geometric decision that the user must make before letting the script do all of the work. Now that the tower is divided in the vertical direction, it must also be divided in the *horizontal* direction. This number will determine the number of modules that will go *around* the building. The user will be prompted with:



Capital letters are used to avoid confusion with the previous question, and the script will store the answer as variable *intGrid*. For the example tower, there will be 12 total diagrid modules in the horizontal direction. Within the code, the user input is multiplied by two and stored as a new variable in order to make the triangulation operations that appear later on in the code possible. There are far fewer limitations on the horizontal input as compared to the vertical module input; the number must be an integer. By definition within the code, the variable type can only accept an integer as input. However, the designer or user must act with proper judgment when selecting a module.

It is critical that the user understand the implications of one of the two choices of divisions of the diagrid on the other. The compound solution of the two inputs determines the angle(s) of the diagrid system, and consequently, the stiffness of the system as described in Section 2. Disregarding the structural implications for now, the user input process for the final geometry is complete.

4.3 Describing the Surface in a [u,v] Coordinate System

Because the geometry is irregular, the surface is allowed to deviate from simple geometry. The utilization of the [u,v] coordinate system will segment in the two directions of the surface, which—after

following the steps above—will be in the customary horizontal and vertical directions. The surface will be segmented along its u-axis into *intGrid* intervals. No matter how curved the tower gets, the surface will always have an axis of length *intGrid* intervals around the circumference of the surface when sliced in plan. All intervals will be the same for a single z-value height, but because the curve of the tower may vary, the distance between each node on the axis will change depending on the height of the isocurve along the tower.

In order to segment in the vertical direction, additional operations must be completed. With a surface that curves in multiple directions, simply dividing the curve by an equal number of intervals of equal length X will not guarantee equal floor heights. Figure 13 shows an exaggeration of a curve that shows how this approach can fail. On the right, the surface is simply divided into the same number of segments that the user had input for the module and floor heights, yet the floor-to-floor heights are not consistent due to the curve. Instead of using intervals along the curve that are equal, the intervals in the v-direction must be different, as seen on the left, to satisfy equal floor heights. Therefore, the usual ease of scripting with the [u,v] system must be bypassed using a more calculated approach.

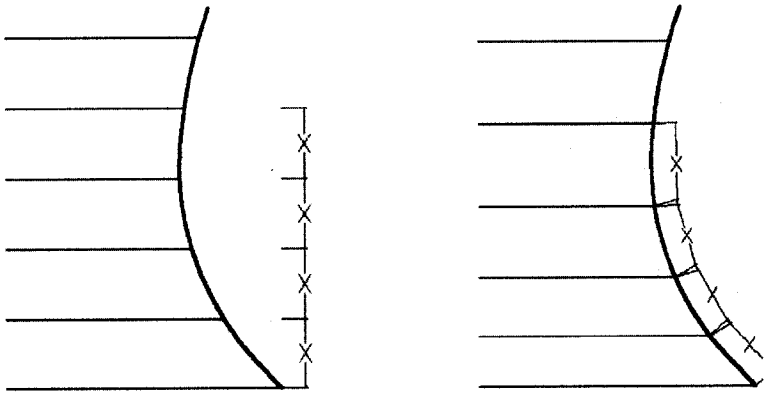


Figure 13 Comparison of methods for assigning floor-to-floor heights

Instead, the script must take into account the curve of the surface in the vertical direction and segment the surface within the [u,v] coordinate system based on the [x,y,z] coordinate system. Now, the z-coordinate is the defining value as the script loops to find the closest point on the surface with the same Cartesian z-value. Outside of the Main function of the AddDiagrid script, the function *addDrawDiagrid()* starts by creating an array of points that starts on the plane z=0 and extends up the tower until i=*intV*. In this loop, *dblR* refers to the maximum value of the absolute value of the y value for each coordinate point that defines the bounding box of the surface. More simply, this guarantees that the array of points does not fall anywhere within the drawn surface. The variable *i* counts the number of

diagrid modules from the zero or first floor through the full height of the tower. By multiplying by 3.5 and *intGZ*—which represents the number of floors per diagrid module within this external function—the z-coordinate for each division point is found. After each of these Cartesian coordinates are found, the closest point on the surface is found under the constraint that it also has the same z-value. The column of points that is used to evaluate the surface isocurve can be seen in Figure 14 on the right. The isocurve, or curve along that same xy-plane, is found.

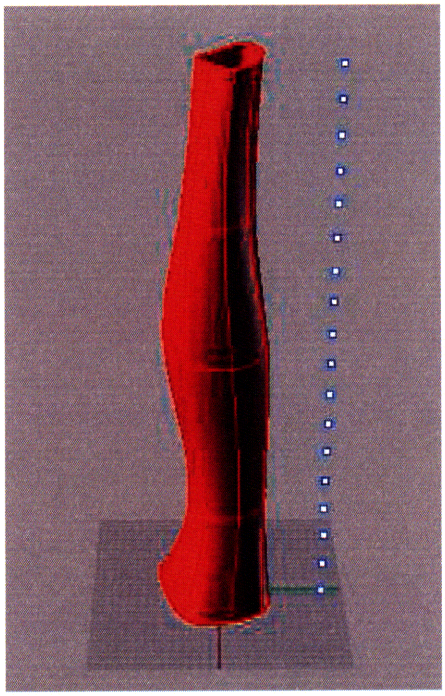


Figure 14 Design Example with floor height points

Next, that curve is divided into *intG* segments. This segmentation ensures that at each diagrid module height, the proper diagrid segmentation occurs in the horizontal direction. This step may seem redundant since the horizontal segmentation can be done using the simple [u,v] system, but it is the only way to ensure that all of the new coordinate system intervals fall on the same height isocurve. The whole segmentation process is carried out by two For loops within the external function. The code can be found within the *segmentSurface()* method in lines 104-121.

Now that the segmentation is complete, the surface is divided in the following way. The array *arrDivPts*(i,j) now holds all coordinates for points within the surface's [u,v] system, now represented in the code as [i,j] and pictured in Figure 15. Note that the u axis runs along the height of the tower while the v-axis is along the circumference.

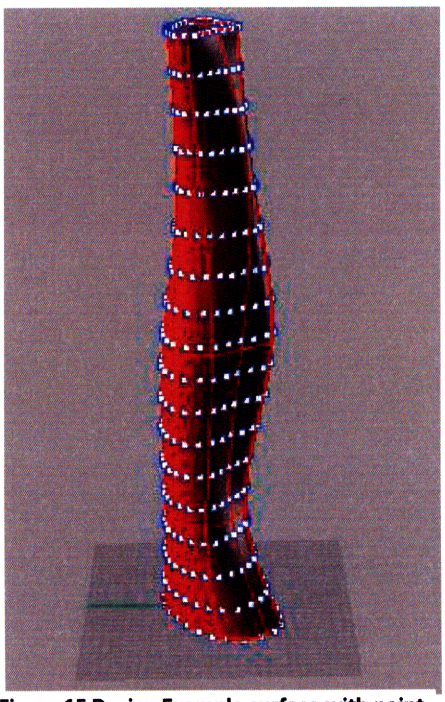


Figure 15 Design Example surface with point drawn at every [u,v] point

4.4 Drawing the Diagrid

Now that the coordinate system has been determined for the surface, the next step is to apply the diagrid. The surface coordinate system has twice the number of intervals as divisions in the horizontal direction and an equal number of intervals as diagrid modules in the vertical direction. That way, the diagrid can be drawn on the surface using the method pictured in Figure 16 on the right. The horizontal intervals are represented by the lowercase letter L and the vertical intervals are represented by the letter K.

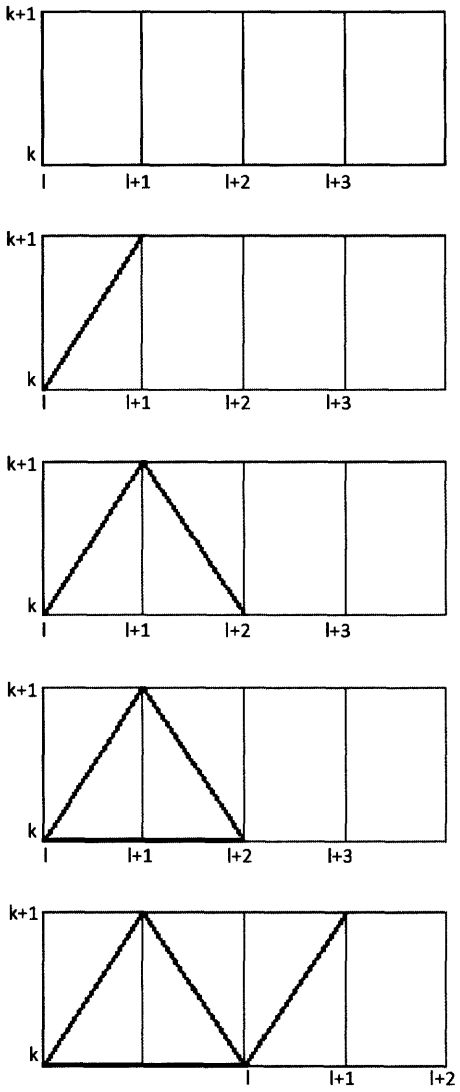


Figure 16 Step increment for addDrawDiagrid function

The best way to perform the repeated function of adding polylines between particular points is to create a looping function. The intervals of k represent each vertical diagrid module. For k=0 to *intV*—the number of modules total—the loop is performed to draw each story's diagrid modules. For each story, a separate loop is performed along the variable l that represents the horizontal surface coordinates. First, a line is drawn from the point (k,l) to (k+1,l+1). Next, a line is drawn from (k+1,l+1) to (k, l+2). Finally, a line is drawn from (k,l+2) to (k,l) to close the triangle. The loop steps the variable l by 2 so that the start of the next triangle begins at the bottom corner of the previous triangle—as seen on the right. This is why the desired horizontal modules are multiplied by two in the preceding code. This intricate looping segment of code was written with specific loops for special cases such as when the loop must close the final triangle on a vertical module by returning to the l=0 point and the top of the highest module which is only a series of lines on xy-plane.

4.4.1 Approximation of the Architectural Surface by Structural Elements

Finally, after looping over all stories, the script will terminate, and the user will be left with a surface with a series of diagrid lines applied to it as seen in Figure 17. Since the endpoints of the linear segments are determined by coordinate points from what may be a curvilinear surface, the lines to not fall entirely on the surface. Most likely, the only two points of the line that *do* fall on the surface are the

endpoints due to geometry. Therefore, the more diagrid modules there are in the vertical and horizontal direction, the greater density of modules which leads to a better approximation of the surface form itself.

The lines of the diagrid are stored on a new layer called, "Diagrid Lines." If, after the script is drawn, the user is dissatisfied with the outcome, he or she has the ability to go select all objects on the "Diagrid Lines" layer and delete them to start from scratch with different variable inputs. Besides the original automation of the diagrid drawing process, this cuts down on the tedious actions required to select intricate geometry that may overlap other drawn elements.

The application of the diagrid lines to the surface marks the final step of the pure-geometry motivated scripting. The details of the exportation of this data along with any added load vectors can be found in the following sections.

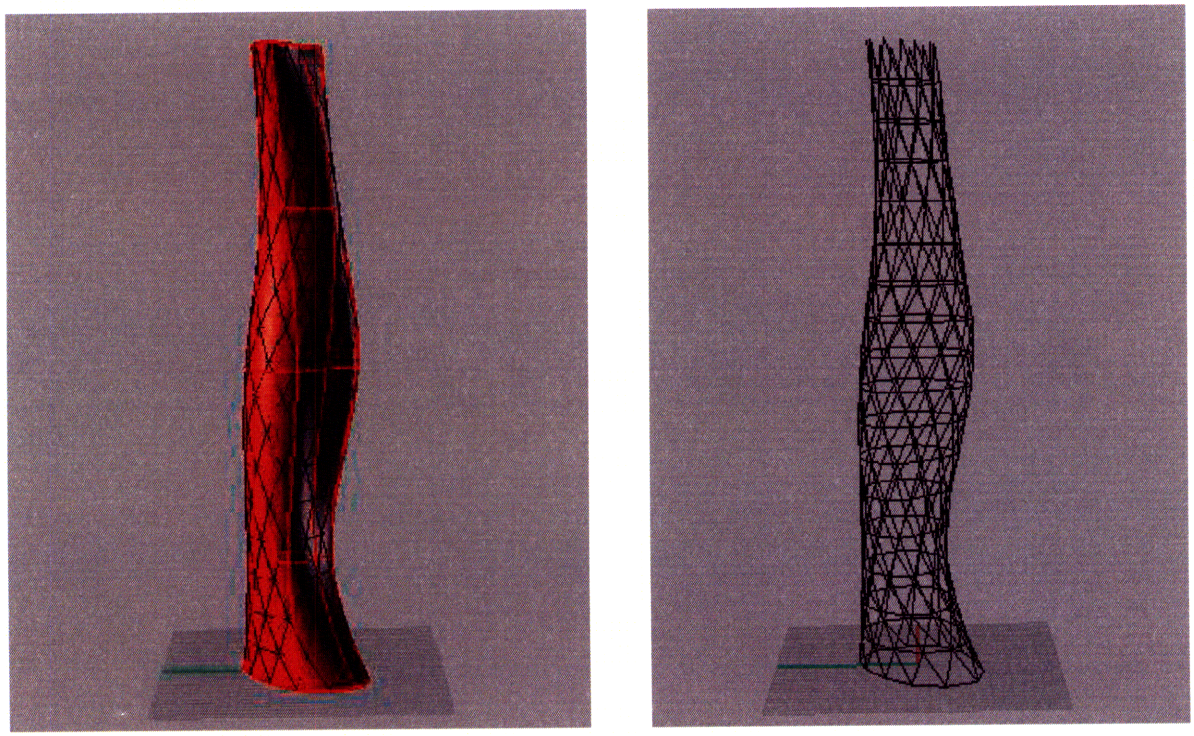
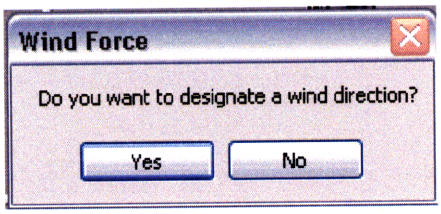


Figure 17 Design Example resulting diagrid with and without architectural surface

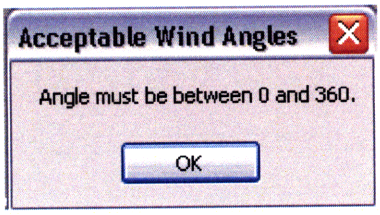
4.5 Coding the Wind Load Vectors

It may be useful to define load vectors for wind forces on the tower structure within Rhinoceros 4.0 before exporting the data through an intermediary to a structural analysis package. Assuming that the surface model is drawn with the positive y-direction representing north and the negative y-direction representing South, the user can designate a direction for the wind using the script options. Ultimately, the wind force vector will only be applied at the nodes of the diagrid for simplification.

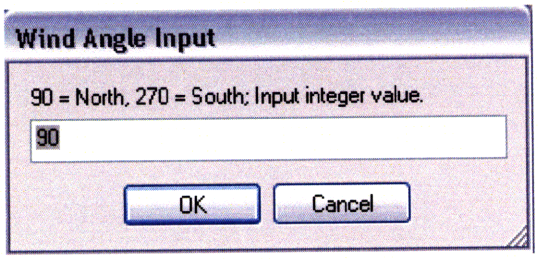
After the diagrid is applied as seen above, the script runs another function which begins by asking the user:



This is the load for which the program will run a structural analysis. If the user chooses to not designate a wind direction, a conditional statement immediately throws the script out of the function. If the user chooses to draw in wind vectors, he or she will be prompted to input an angle for the desired direction of the wind. The script follows up with:



and prompt on the following page to obtain the value:



The angle that the user has will then be used to determine a unit vector for the wind force. In Polar coordinates, the radius is assumed to be one for a unit vector while the angle is exactly what the user designated. In order to convert from Polar to Cartesian coordinates, the following definitions were used.

$$x = r*\cos(\theta)$$
$$y = r*\sin(\theta)$$

Also for simplicity, the user has entered the angle in degrees; however, the sine and cosine functions in the coding language operate on radians. The methods *Rhino.ToDegrees()* and *Rhino.ToRadians()* help simplify the transitions between units.

When applying the wind force, only one side of the building will feel the windward, lateral force. In order to select only the nodes that are on the windward side, the vector normal to the surface at each node was compared to the directional vector of the wind. The only surface normal vectors that were considered were at the vertices of the diagrid members. Another loop was used to expedite this selection.

Within each loop, the dot product was found between the wind vector and the surface normal vector. Using the definition of a Vector Dot Product,

$$A \cdot B = |A||B|*\cos(\theta)$$

The angle between the two vectors was found. Since this study is limited to relatively slight angles that differentiate from vertical along the tower, it was assumed that the contribution of the z-component on the surface normal vector would be insignificant. Finally, to determine if the local vertex of the diagrid system was facing in the direction of the oncoming wind, the angle was compared to ninety degrees. If the angle was less than ninety degrees, then it could be assumed that the local surface would have no direct wind force. Angles of more than ninety degrees signify that the particular local vertex would be exposed to the wind force.

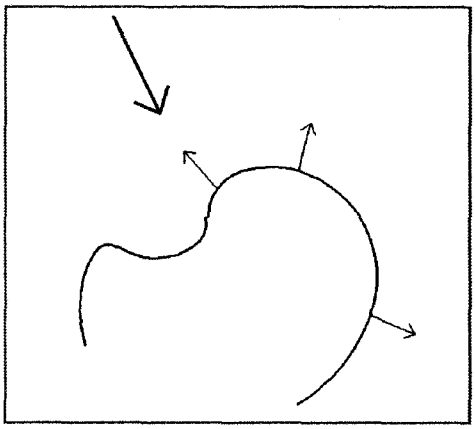


Figure 18 Wind force compared to surface normal vectors

As the vertices are split into these two categories, the wind vector is applied, scaled, and drawn at each vertex of the latter set. For visualization purposes, the wind force vectors are set in their own layer of a unique color to stand out from the rest of the model. Also, the scaling helps to get a sense for the increasing lateral force that correlates to the increase of altitude. The scale factor can be adjusted for accuracy by returning to the code, and it would be possible in future studies to incorporate an algorithm that determines a more accurate, scientifically-based wind load along the varying vertical height.

Similar to the diagrid line elements from the previous section, the wind vectors are drawn on their own layer called, "Wind Vectors." The layer color is a light blue which helps to visually distinguish the wind vectors from other line elements in the drawing field. Along with the scaling of the vectors for different wind intensities, this helps to give the user a better visual understanding of the chosen wind properties.

The following Figure 19 and Figure 20 highlight the visualization of the progress made thus far on the model. At this point, the RhinoScripting side of the first cycle is done. All of the data that is necessary for transfer to a structural engineering package is present in the model; however, the way that this information is exported is critical for easy access on the other end.

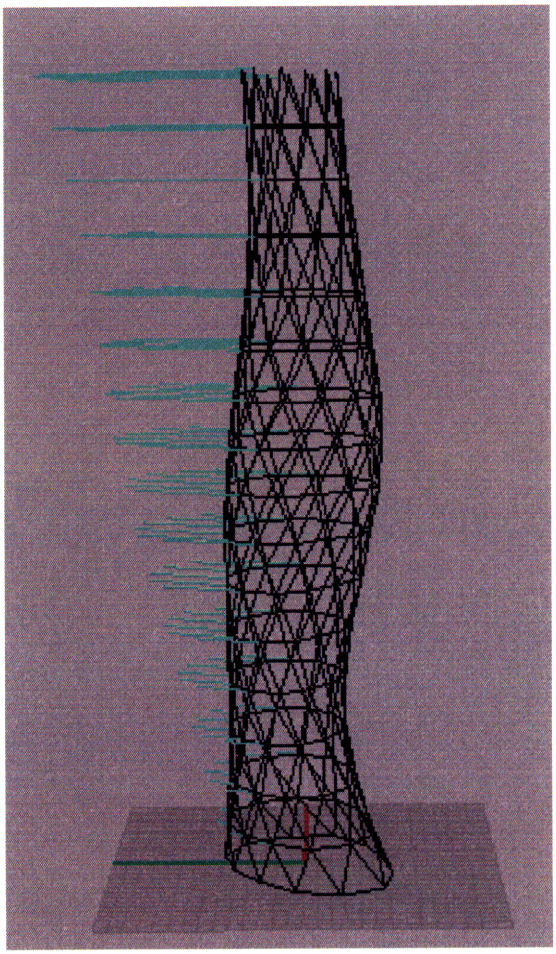


Figure 19 Design Example final architectural process output, elevation view

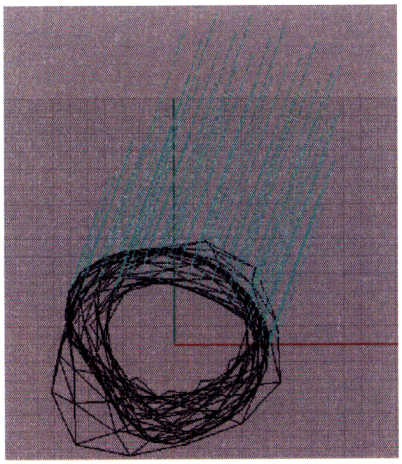


Figure 20 Design Example final architectural output, plan view

4.6 Critical User Input

For the model to reach the state pictured in the image on the previous page, the user or designer has had to contribute certain required inputs. Although the curve inputs require greater study and investigation in order to arrive at a desirable design, the other inputs are simple parameters. The purpose of the program itself is to allow the user to easily vary these parameters and quickly redraw multiple variations before moving on to export the final architectural data. Table 2 summarizes the necessary inputs for the DrawDiagrid program along with their limitations or restrictions.

Input	Variable Name	Description	Limitations
one or more closed curves	*strGeo, becomes strShape*	Used as input to generate the tower geometry by lofting the curves, a Rhino function	Curves must all be in same direction (clockwise or counterclockwise)
number of floors per diagrid module	*intGridZ*	The number of building stories that the diagrid will span vertically	Division must work out to be a close factor of the total height of the building
number of horizontal diagrid modules	*intGrid*	The number of diagrid modules that will surround the building	None, but limited by aesthetics and eventual structural considerations. If too many, diagrid system less effective laterally.
wind angle	*dblWind*	Angle between 0 and 360 degrees with 90 degrees representing due North and 270 representing due South	Angle input should be between 0 and 360 degrees, otherwise program will still convert to radians

Table 2 Documented user input variables and their restrictions

4.7 Limitations

The architectural script, however powerful and noteworthy, still has significant limitations. The curved geometry cannot be too extreme, there is no embedded variability for the floor-to-floor height, and the modular arithmetic crops the height of the original design.

The design example, what would be cause for alarm for a conventional structural engineer, shows conservative irregularity of form. This coding exercise is equipped to handle more intense geometry such as the model in the image below; however, the script runs into trouble when certain extreme concavities and convexities appear. The method for determining the coordinate system on the surface is based on an array of points that are created adjacent to the surface tower geometry. The *SurfaceClosestPoint()* method finds the corresponding closest point on the surface at the shared height level as in Figure 14. This approach is no longer valid if the geometry experiences extreme bubbles or protrusions as in Figure 21. For example, the dashed line below goes through the surface at three different points on the plane z=n. Therefore, the diagrid cannot be fit to this surface because the commands will create a discontinuity and the script will fail. In this case, the diagrid system is not best to accurately approximate the form, and another structural approach should be taken.

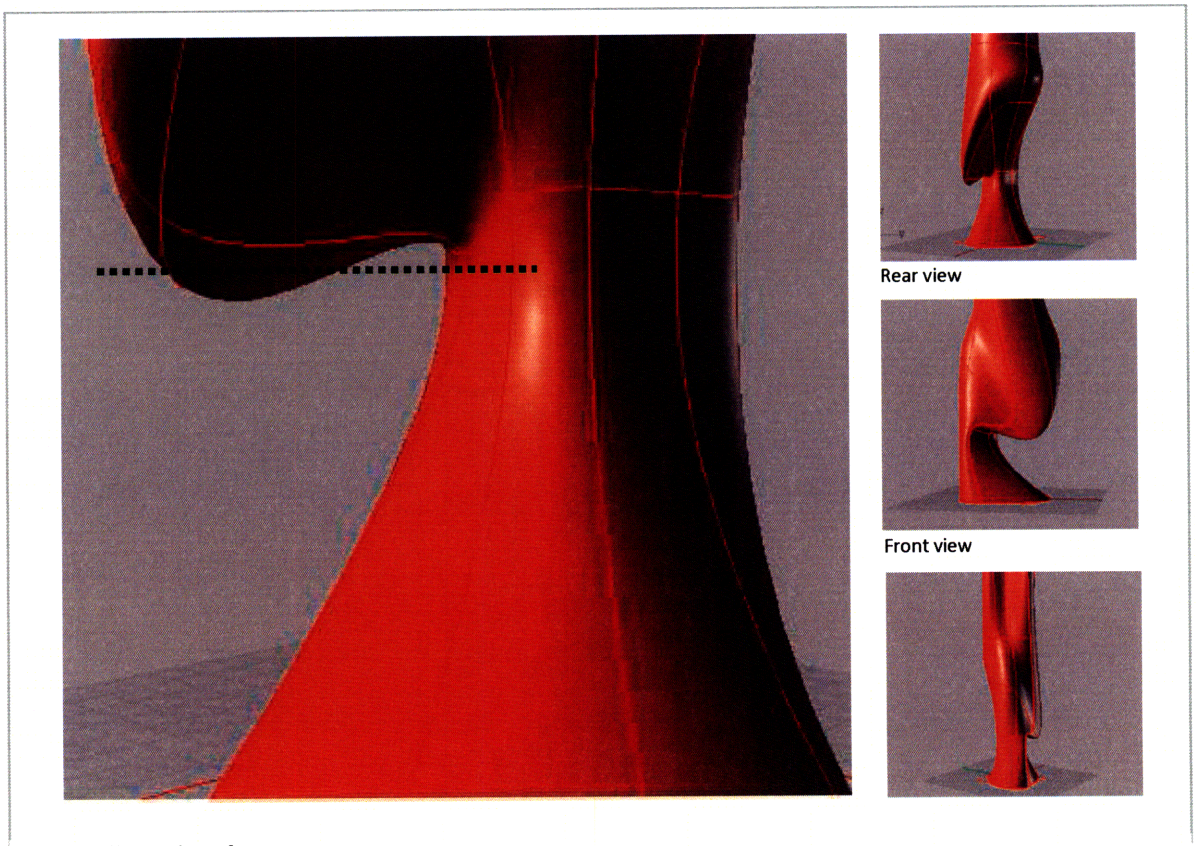


Figure 21 Illustration of extreme curvature concerns

Secondly, the script does not currently allow for any variation in floor-to-floor height. Similar to the DrawDiagrid script of concern, the Hearst Tower also has little modulation in floor-to-floor heights to keep with the very regular structural system (see Figure 22). Realistically, multiuse towers are typically occupied by retail, commercial, residential, and/or hotel programs. A designer would want control over these heights along the building. To overcome this limitation, the user input could be expanded to allow for different program type floors. The user should be able to allocate how many floors of each program type should occupy the tower, which would just demand more detailed algorithmic design for the modular mathematical script. The program could even store typical floor heights and let the user select how to organize them vertically, and subsequently, the code could fit a diagrid to that design. Lower level retail floors could be designated by 4m ceiling heights while the more intimate office, hotel, and residential settings could dictate 3.5 or 3 meter heights. Yet, the script was simplified to allow only one floor height in order to limit the over-use of user input. The more the user has to input, the closer the process comes to a tedious design modeling iterative process.

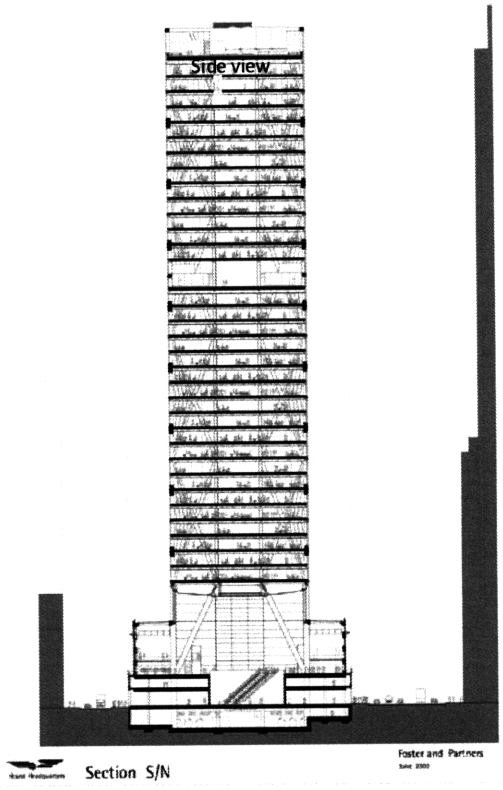


Figure 22 Hearst Tower section drawing[7]

Finally, a large portion of the script focuses on the modular calculations that are essential to the simplification and translation of surface models to building systems. These calculations truncate the height of the tower based on the default floor-to-floor height. From an architectural perspective, this could be detrimental to the design. Although it is possible to design a separate structural system for this portion of the building, the structural analysis and modeling procedures embedded in the integrated process would fail to account for this part of the building. Generally, the assumption of the truncation is acceptable, but as computational methods become more integrated and acceptable in structural design, a more accurate solution must be developed. Inclusion of all building elements is critical for more sophisticated structural design models.

After considering these limitations, the archtiectural process is now complete.

5 Intermediary Translation of Data

Rhinoceros, as well as other similar design tools, use the VisualBasic coding language to deal with complex data. When developing code, this becomes extremely convenient. When writing in RhinoScript, the author of the code has the ability to link with other file type programs to retrieve and export data. Most RhinoScript use for design is done by manipulating input data; however, with disciplines becoming ever more merged, coders have taken advantage of the ability to export data to programs like Microsoft Excel and work within *.txt files.

The structural engineering package used in this study, SAP2000, can import data in many forms—*.dxf files, *.xls files, and *.$2k files (similar to *.txt). It is simple to import an architectural model via the *.dxf format, but without additional user input, there are no forces applied to the structure, load scenarios chosen, or fixity assigned to any joints. In order to accomplish a fluid transition between software with limited manual input, another approach must be taken. The *.$2k file imports load cases and analysis options along with material data, geometric data, and model imaging data.

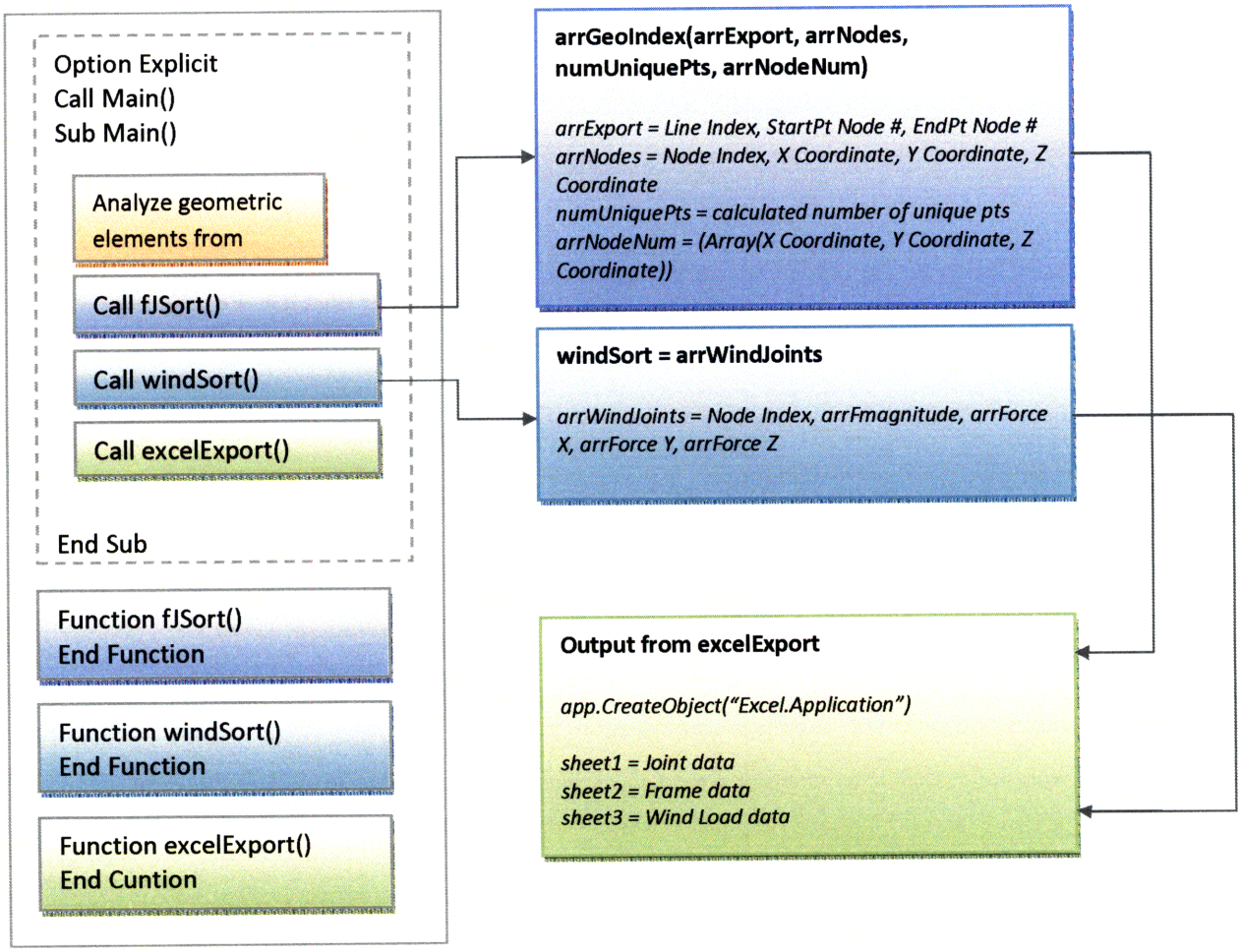


Figure 23 Pseudocode of organization methods and their corresponding outputs highlighted

Since the *.$2K file is a text file under a different file extension, visual basic can be used to translate RhinoScript data and write to an *.$2k file. An *.$2k can be parsed into an Excel spreadsheet for data translation and then translated back into the original text format or it can be used directly as a text file. Since RhinoScript can write directly to text files, this can be done in one pass; however, an intermediary Microsoft Excel file was used to simplify and help visualize the process. The pseudocode of the script that was written within RhinoScript is pictured in Figure 23. The most important consideration was to get the data in the form that the *.$2k file was organized. After that, this study will be limited to describing how further development of this segment of code could allow for full automation.

5.1 The Shell of the *.$2k File

The basic *.$2k file that SAP2000 will import and export is a text file with a series of tables. A basic file example can be found in the Appendix which shows the breakdown of necessary tables. Most of these inputs are default settings, and only a few of the tables vary according to the model geometry and load scenarios. The tables of interest are the following along with a brief description and the corresponding fragment of code.

Analysis Case Definitions, Defines the load cases of consideration. This is where new load cases can be initiated, in addition to the default DEAD and MODAL load scenarios.

```
TABLE:  "ANALYSIS CASE DEFINITIONS"
   Case=DEAD    Type=LinStatic    InitialCond=Zero    RunCase=Yes
   Case=MODAL    Type=LinModal    InitialCond=Zero    RunCase=Yes
   Case=wind    Type=LinStatic    InitialCond=Zero    RunCase=Yes
```

Case – Static 1 – Load Assignments, Here, the loads defined under Analysis Case Definitions are given weighted factors for the analysis simulation.

```
TABLE:  "AUTO WIND - ASCE7-05"
LoadCase=wind ExposeFrom=Diaphragms Angle=0 WindwardCp=0.8 LeewardCp=0.5 ASCECase=1 E1=0    E2=0
UserZ=No WindSpeed=70  Exposure=B    I=1  Kzt=1  GustFactor=0.85 Kd=0.85 ExpWidth="From diaphs"
LoadCase=wind-2  ExposeFrom=Diaphragms  Angle=90  WindwardCp=0.8  LeewardCp=0.5  ASCECase=1  E1=0
E2=0 UserZ=No WindSpeed=70 Exposure=B I=1 Kzt=1 GustFactor=0.85 Kd=0.85 ExpWidth="From diaphs"
```

Connectivity – Frame, Defines each frame element (diagonal column elements) by its joints, or endpoints. Also defines the curvature or linearity of the element.

```
TABLE:  "CONNECTIVITY - FRAME"
   Frame=1    JointI=1    JointJ=2    IsCurved=No
   Frame=2    JointI=3    JointJ=4    IsCurved=No    ...    (count frame elements until nth frame)
   Frame=n    JointI=x    JointJ=y    IsCurved=No
```

Coordinate Systems, Defines the type of coordinate system along with the origin.

```
TABLE:  "COORDINATE SYSTEMS"
   Name=GLOBAL   Type=Cartesian   X=0   Y=0   Z=0   AboutZ=0   AboutY=0   AboutX=0
```

Frame Loads – Point, Defines the frame loads by location on frame, frame number, load case, load type, and direction. For this preliminary iterative design study, only wind loads on joints were considered for simplification, so the frame loads could be ignored. Still, below is a sample of what this excerpt would look like.

```
TABLE:  "FRAME LOADS - POINT"
   Frame=n   LoadCase=truck   CoordSys=GLOBAL   Type=Force   Dir=Z   DistType=RelDist   RelDist=0.75
   AbsDist=2.625 Force=-10000
```

Frame Output Station Assignments, Defines the output stations for indicated for all frames, in this case frame variables x and y.(?????)

```
TABLE:  "FRAME OUTPUT STATION ASSIGNMENTS"
   Frame=x   StationType=MinNumSta   MinNumSta=3   AddAtElmInt=Yes   AddAtPtLoad=Yes
   Frame=y   StationType=MaxStaSpcg   MaxStaSpcg=0.5   AddAtElmInt=Yes   AddAtPtLoad=Yes
```

Frame Section Assignments, This table defines the type of section per frame element. The section type is defined in the next table.

```
TABLE:  "FRAME SECTION ASSIGNMENTS"
   Frame=n   AutoSelect=N.A.   AnalSect=FSEC1   MatProp=Default
```

As the design process develops, the script can be adapted to chose frame sections corresponding to the size needed to sustain the loads at different locations along the tower. For example, smaller sections could be assigned to frames that are higher up on the tower and larger sections to frames on the bottom. The script that formulates the current frame number assignments could also be used to organize frames into height groupings and then parse the data into cross-sectional assignment groupings. The actual section declaration follows in the next table.

Frame Section Properties 01 – General, Assigns section names along with properties including material, shape of cross section, thicknesses, as well as other default material values.

As iterations improve on sizing the cross section of the diagonal bracing elements, this table will be required to be updated for the new steel cross section properties. When the architect gives feedback about

their satisfaction or dissatisfaction with the size of the elements, he or she can limit the code to certain desirable preset sections.

```
TABLE:  "FRAME SECTION PROPERTIES 01 - GENERAL"
   SectionName=FSEC1 Material=A992Fy50 Shape=Box/Tube t3=0.1524 t2=0.1016 tf=0.00635   tw=0.00635
   Area=0.00306451 TorsConst=1.01854328602056E-05 I33=9.77059914155833E-06
   I22=5.14002453175833E-06 AS2=0.00193548_ AS3=0.00129032 S33=1.28223085847222E-04
   S22=1.01181585270833E-04  Z33=0.00015618920375   Z22=0.00011726992675   R33=5.64650967334938E-
   02  R22=4.09545420868536E-02  Color=White  FromFile=No AMod=1 A2Mod=1 A3Mod=1 _ JMod=1 I2Mod=1
   I3Mod=1 MMod=1 WMod=1  Notes="Added 1/25/2009 3:12:53 PM"
```

Joint Coordinates, Organizes the joints by number. Each joint is declared as a coordinate type along with its corresponding Cartesian coordinates, X(or R), Y, and Z.

```
TABLE:  "JOINT COORDINATES"
   Joint=1    CoordSys=GLOBAL    CoordType=Cartesian    XorR=9.33986640718973    Y=2.49728729643643
   Z=17.4892373221086   SpecialJt=No
   Joint=2    CoordSys=GLOBAL    CoordType=Cartesian    XorR=7.50973957815568    Y=4.32741422762975
   Z=13.9892373221086   SpecialJt=No
   Joint=3
```

Joint Loads – Force, Numbers the joints that are receiving the specific load case, and defines the load (force or moment) on each joint.

```
TABLE:  "JOINT LOADS - FORCE"
   Joint=n   LoadCase=wind   CoordSys=GLOBAL   F1=x   F2=y   F3=0   M1=0   M2=0   M3=0
```

In the completed study, this table would be used to define the input wind force on each joint. The force in the z-direction will be zero since the wind is assumed to be acting completely in the lateral direction on the façade. The vectors drawn by applying the input data during the architectural process can represent the magnitude of the wind force using appropriate data and thus, can be used to determine the values for the force on the joint in the x and y direction.

Joint Restraint Assignments, Defines the fixity constraints on the named joints. The variables U {1,2,3} define the displacement rules in the x,y,and z directions while the variable R {1,2,3} define the rotation restrictions. Yes means there *is* a restriction, and No means there is no limitation.

```
TABLE:  "JOINT RESTRAINT ASSIGNMENTS"
   Joint=1  U1=Yes   U2=Yes   U3=Yes   R1=Yes   R2=Yes   R3=Yes
   Joint=i   U1=Yes   U2=Yes   U3=Yes   R1=Yes   R2=Yes   R3=Yes
   Joint=j   U1=No   U2=No   U3=No   R1=No   R2=No   R3=No
```

A script can be written to determine the joints with their z-coordinate equal to zero. The U and R values for these joints can be declared in an array to be Yes while all other joint restraint variables can be set to

No. When storing the restriction data, the joints with z-coordinate equal to zero will appear as pictured above in rows 1 to i, assuming full fixity at the base, while all other joints will have the same conditions as Joint=j.

Load Case Definitions, Defines the load cases by design type (dead or live). Self weight multipliers are declared.

```
TABLE:  "LOAD CASE DEFINITIONS"
   LoadCase=DEAD   DesignType=DEAD   SelfWtMult=1
   LoadCase=wind   DesignType=WIND   SelfWtMult=0   AutoLoad=ASCE7-05
   LoadCase=windcustom   DesignType=WIND   SelfWtMult=0
```

The preset wind load cases is for ASCE7-05. In order to simulate a wind force that is realistic on joints and not surfaces, a new load case can be defined here. The wind load would be a live load with self weight multiplier of 1. This new load must generate a table similar to the Joint Loads – Force table shown above. This new table will follow the Auto Wave 3 – Wave Characteristics – General Table (see Appendix) in the *.$2k file, and it can carry over the wind angle assignment from the architectural user input variable declaration.

The rest of the tables in the lengthy file determine default properties of the standard SAP2000 files. These excess tables can be used to form a shell file in which new model information can be inserted. The ultimate organization of the model *.$2k file can be found in the Appendix. The following sections describe how the RhinoScript data can be manipulated into usable data to insert into the *.$2k file.

5.2 Member Geometry

A single script can be written to translate the geometry of the architectural model into manageable data for insertion into the *.$2k file. The organization of the element data is essential for proper translation to the structural model.

To start, all frames or lines of the diagrid must be indexed. In the end, the line definition requires an index for the joint associated with the endpoints of each line. A second list, the compiled list of endpoints, must represent all unique vertices or joints. Since these two lists go hand-in-hand, so the development of the code that follows will discuss both interchangeably.

5.2.1 Indexing the Geometric Information

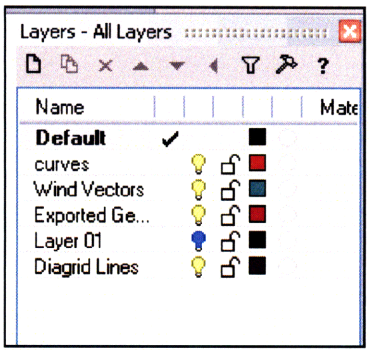


Figure 24 Rhinoceros 4.0 Layer Manager

While running the *addDrawDiagrid()* function at the end of the AddDiagrid script, the drawing functions were performed within a new layer called "Diagrid Lines." Now that these lines are isolated from the other elements of the model, the contents of the entire layer can be selected as an object. Each line is represented by an identifier variable that is then stored in an array of all diagrid lines. First, the code *fJSort()* goes through each line and determines the coordinate points of the line's start and end points. These points are stored in association with the particular line. Since each joint needs to have its own identifier without overlaps or multiples, the end points must be indexed in a different way and reduced to unique joints. To do this, an algorithm checks each line's endpoints in the following fashion. The single first line in the array of all lines is observed. Since it is the first line selected, its endpoints will both be unique. These endpoints are stored as the first and second elements in the final array of unique end points, *arrNodeNum*. After being stored in the array *arrNodeNum*, the endpoints must be re-identified or re-associated with the original line in their new index form.

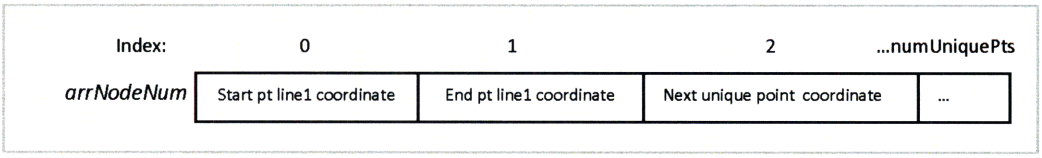


Figure 25 Array visualization of *arrNodeNum* variable

After the first line is complete, the algorithm cycles through each line to check if that line's start point or end point is unique. The check for uniqueness is done by comparing each point to the existing nodes in the array *arrNodeNum*. If both are unique (i.e., they do not share coordinates with the previously stored joints in *arrNodeNum*), the endpoints are stored at the end of the array of unique endpoints and the line is appropriately re-linked with its newly indexed endpoints. If only one is unique, the point in common will automatically find its pair in the array *arrNodeNum* and associate that array index with its original position within the line's array. The other endpoint, the one that was found to be unique, will create a new position for itself at the end of *arrNodeNum* and update its reference within the line array. If both the start and end point are unique, they will both be stored in *arrNodeNum* and their line's data updated. This continues until all lines on the layer "Diagrid Lines" has been indexed.

Now, *arrNodeNum* holds all of the unique nodes or joints in the model. The array *arrInfo* compiles within the loop above to store the new line data. This array will eventually be used to insert its contents into an Excel file which will handle the final data compilation. Each element of the array *arrInfo*

is a line represented by another array that holds the line number q, that line's start point's index in *arrNodeNum*, and that line's end point's index in *arrNodeNum*.

Finally, a new array, *arrNodes*, will hold the information about every joint element while it indexes each array element in *arrNodeNum*. The array arrNodeNum must be reduced from a multidimensional array to a two-dimensional array that will store an index number and a coordinate point in each slot. Remember, the array *arrNodeNum* is an array containing arrays that represent the coordinate points for each of the unique joints. The array *arrNodes* is generated by looping on the total number of arrays within *arrNodeNum* (or, the number of unique joints). With each loop, the index number is stored along with the point of the corresponding joint. Once the loop terminates, the indexing of geometric elements has completed.

Conveniently, the script is able to assume that a line's endpoints need not be checked against one another since a line must have length and, therefore, their coordinates cannot be identical. This saves some computational time; however, checking each point against all other previous unique nodes does slow the script down.

If time permits, the joints that lie on the plane z = 0 must all be assigned to an array that can be imported into Excel in the final data compilation process. This array will be used to identify the joint locations that must get restrained to full fixity as a foundation condition.

5.2.2 Storing the Data

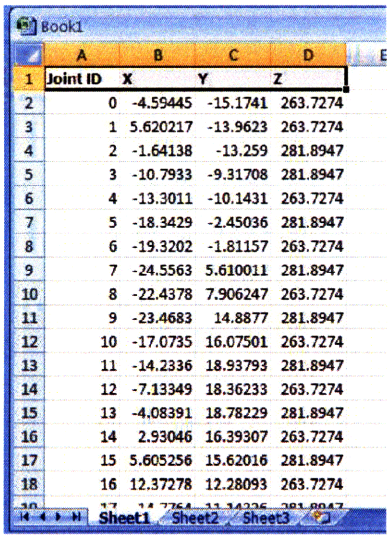
Now that the frame and joint data has been indexed, the data must be put in a form that SAP can recognize. There are two options. RhinoScript can converse and move data to either text or Excel files. The data can either i) be stored in columns in Excel and then inserted into an *.xls file that has been converted from the shell *.$2k text file or ii) be inserted directly into a text file which requires pre-scripting the default inputs to be imported along with the model-specific data. This study will focus on the first approach since it is easier to program intricate loops and arrays to assign information in a cellular spreadsheet rather than a blank text file.

After associating the correct joints and frames together in the section above, the DiagridExport script also knows to export the array elements and store them in a spreadsheet. The code initializes this function, *excelExport()*, by creating an object via the Excel Application. This creates a new spreadsheet which is identified by a variable *app*. The spreadsheet can be easily navigated through RhinoScripting. It is feasible to change the current worksheet, make the window visible or hidden during data storage, and even change the font definitions. Since Excel is based on the VisualBasic language, it is possible to access the methods within the Excel program by calling them within the RhinoScript code.

```
Dim app
  Set app = CreateObject("Excel.Application")
  ' Make it visible
```

```
app.Visible = True
' Add a new workbook
Dim wb,ws
Set wb = app.workbooks.add
'set current sheet
Set ws = wb.Worksheets(2)
ws.Activate
```

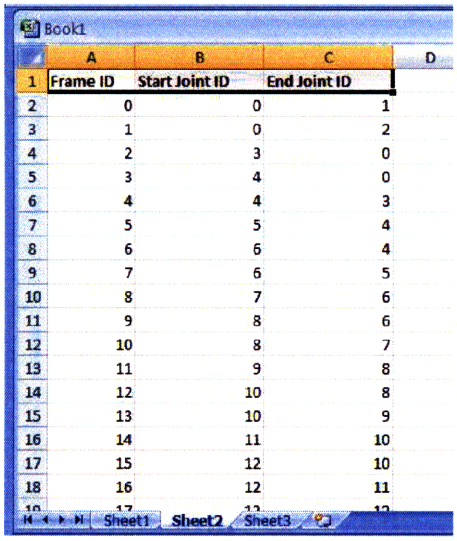
The *excelExport()* function requires the two sorted arrays from Section 5.2.1, *arrInfo* and *arrNodes*. It takes the line or joint information from *arrInfo* and stores it in the second worksheet, and then it stores the node list in the first worksheet. Upon completion, the script automatically keeps the Excel window open so that the user can continue processing information manually. Figure 26 and Figure 27 below show the nodal information for the design example as it appears in the Excel window.



	A	B	C	D
1	Joint ID	X	Y	Z
2	0	-4.59445	-15.1741	263.7274
3	1	5.620217	-13.9623	263.7274
4	2	-1.64138	-13.259	281.8947
5	3	-10.7933	-9.31708	281.8947
6	4	-13.3011	-10.1431	263.7274
7	5	-18.3429	-2.45036	281.8947
8	6	-19.3202	-1.81157	263.7274
9	7	-24.5563	5.610011	281.8947
10	8	-22.4378	7.906247	263.7274
11	9	-23.4683	14.8877	281.8947
12	10	-17.0735	16.07501	263.7274
13	11	-14.2336	18.93793	281.8947
14	12	-7.13349	18.36233	263.7274
15	13	-4.08391	18.78229	281.8947
16	14	2.93046	16.39307	263.7274
17	15	5.605256	15.62016	281.8947
18	16	12.37278	12.28093	263.7274

Sheet1 Sheet2 Sheet3

Figure 26 Design Example joint index Excel sheet 1



	A	B	C	D
1	Frame ID	Start Joint ID	End Joint ID	
2	0	0	1	
3	1	0	2	
4	2	3	0	
5	3	4	0	
6	4	4	3	
7	5	5	4	
8	6	6	4	
9	7	6	5	
10	8	7	6	
11	9	8	6	
12	10	8	7	
13	11	9	8	
14	12	10	8	
15	13	10	9	
16	14	11	10	
17	15	12	10	
18	16	12	11	

Sheet1 Sheet2 Sheet3

Figure 27 Design Example frame index Excel sheet 2

5.3 Applying Wind Loading on Desired Nodes

Similar to the way the joint and frame information was parsed above, the wind load vectors can be organized in a way that is useful for data combination. The wind angle, affected joints, and vector magnitude at each joint are the necessary pieces of information summarize the load data. The full RhinoScript code can be found in the Appendix in the DiagridExport program under the *windSort()* and *excelExport()* functions.

5.3.1 Indexing the Load Attributes

In order to compile the data, the pieces must be arranged with proper association. First, the angle of the wind vectors must be stored as an accessible variable within the new sorting function *windSort()*. Just as the DiagridExport method used the independent layer of "Diagrid Lines," the *windSort()* function within DiagridExport uses the homogeneous layer "WindVectors" to manage initial selection criteria. Since the wind force vectors must be associated with joints, much of the same process stands as before. Now, however, the head of each vector is compared with the final list of joints from the main Sub method of DiagridExport. When a match is found, that joint index is stored in an array with the corresponding vector's magnitude. The coordinate points of the joints that intersect with force vectors are necessary only to determine the height of the joint. The height information is then passed down the function to calculate the magnitude of the wind force as it changes with altitude.

At this point, the magnitude of the vector drawn using Rhino can also be replaced with a new force vector. The original script only calls for a constant scalar that adjusts with height, yet adjustment can be made here for a more accurate load case, rather than in the original DrawDiagrid script. The change can be made within the *windSort()* Function.

Along with the force magnitude from above, it is also necessary to know the angle of the wind force vector to determine the contributions of the force in the x, y, and z-directions. Since the wind is a lateral force, the z direction can be set to a default value of zero. Once the angle of the vectors are recalculated by sampling one wind vector, all of the force magnitudes can be determined for each joint along with their x and y components and stored in *arrForce*. This information is stored within the same data array as the Joint number, *arrWindJoints*, so that it can be accessed by Excel to overlay on a single sheet. This data will be used in the future to satisfy the inputs for "Joint Loads – Forces" and "Case – Static 1 – Load Assignments," in the shell document.

5.3.2 Sorting the Data

After the wind vector data is arranged properly, the same Excel file as created in Section 5.2.2 is used to store this fresh data. The same Rhino-to-Excel scripting method applies, except now the data is stored in sheet 3, as to not overwrite the existing joint and frame data. In short, all of the information for

each wind vector including the joint index, the total magnitude of the particular wind force, and the force components in the x, y, and z-directions is stored in sheet 3 as seen in Figure 28. The complete data set is now ready for compiling with the framework of the SAP2000 *.$2k file.

Book1

	A	B	C	D	E	F
1	Joint ID	Force magnitude	Fx	Fy	Fz	
2	9	2818.946931	964.1366	2648.944	0	
3	11	2818.946931	964.1366	2648.944	0	
4	13	2818.946931	964.1366	2648.944	0	
5	15	2818.946931	964.1366	2648.944	0	
6	17	2818.946931	964.1366	2648.944	0	
7	19	2818.946931	964.1366	2648.944	0	
8	10	2637.274467	902.001	2478.227	0	
9	12	2637.274467	902.001	2478.227	0	
10	14	2637.274467	902.001	2478.227	0	
11	16	2637.274467	902.001	2478.227	0	
12	18	2637.274467	902.001	2478.227	0	
13	28	2453.401022	839.1126	2305.443	0	
14	29	2453.401022	839.1126	2305.443	0	
15	30	2453.401022	839.1126	2305.443	0	
16	31	2453.401022	839.1126	2305.443	0	
17	32	2453.401022	839.1126	2305.443	0	
18	33	2453.401022	839.1126	2305.443	0	
19	41	2261.020207	772.2145	2124.664	0	

Sheet1 Sheet2 Sheet3

Figure 28 Design Example applied load data Excel sheet3

5.4 Excel to *.$2k

After all of the architectural data organization and exportation to Excel has occurred, the data can be sorted internally within Excel to fit in the structural model's shell file. Working within the spreadsheet allows for easy manipulation of data than line-based text files. For now, the coding must move to be internal to Excel rather than Rhinoceros. Before being ready for iteration, the structural model shell file must be converted into an Excel file.

5.4.1 Preparing the Shell Excel File

The file extension for the structural model shell file must be changed from *.$2k to *.txt. This is an easy manual adjustment. After this change, it becomes accessible in Microsoft Excel. Within Excel, the file can be opened using the Data: (From Text) function. This opens a dialogue window where the shell file, now a .txt, can be found. From here, its contents are formatted by selecting Delimited, Tab and Space, and General for the default units. Finally, there is a dialogue box that gives the user the option to place the chart on the current sheet in a selectable location or another sheet. The file is now ready to be manipulated within Excel to fill in the fields that customize the basic file to a particular building.

5.4.2 Automation Opportunity with VBA

Microsoft Office's VBA platform allows for developers to program certain macros and elements into the various Office products. A parent Excel file can be created to control the automation of the aggregation of the data. Once the shell file is an Excel document, the VBA of the parent file can interface between the shell document and the data set compiled in Section 5.3. The parent file will have to control the placement of variables as well as account for repetitions of standard or default inputs according to the required number of joint or frame definitions. It should also be configured to store information from past iterations such as cross section choice, wind vector directions, and displacement outcomes as a process log. It is possible to manually copy and paste values between the sheets, yet repeating that process manually for multiple design iterations would quickly become tedious. Although full automation of this part of the design process was not achieved, the powerful intrinsic indexing ability of Excel spreadsheets coupled with the simplistic VBA logic suggest that this parent approach is the most promising.

5.4.3 Output

The design example's final output from Excel is a spreadsheet that is based on the *.$2k shell file printout that appears in the Appendix.

The final step in this intermediary process is to reverse-translate this Excel document back into a *.txt file. This is done by using the "Save As" function and selecting "Other Formats" and "*.txt." After naming and saving the file, the WordPad document can be opened and re-saved as a text based, *.$2k file, called model.$2k. Renaming this file concludes the semi-automated intermediary process, and the user can now move on to importing the model with applied loads into SAP2000.

6 Structural Evaluation with SAP2000

The second half of the integrated design approach involves structural analysis of the generated forms. For this, the SAP2000 software was chosen due to its versatility when handling imported data. The ultimate goal is to run a simple analysis on the building model with a basic lateral wind load. The building model will be represented by joints and frames with certain restraints on the ground-level joints and wind forces on certain local joints. With a predefined steel cross section set at an initial value when arranging data within Excel, the building's overall stiffness will be selected (to be a default option from the program). Keeping with the Performance Based Design approach, the fundamental structural performance will eventually be judged according to the resulting max displacement. This displacement should be no larger than around L/300 or L/500, where L is the overall height of the tower.

6.1 Running model

Now that the data has been processed by the architectural software side and fed through the intermediary sorting functions, the model is ready to be imported into SAP2000. After loading the software (manually, for this limited automated study), the model.$2k file can be opened through the SAP Import function under the File tab. Figure 29 below shows the design study model just after it was imported and opened.

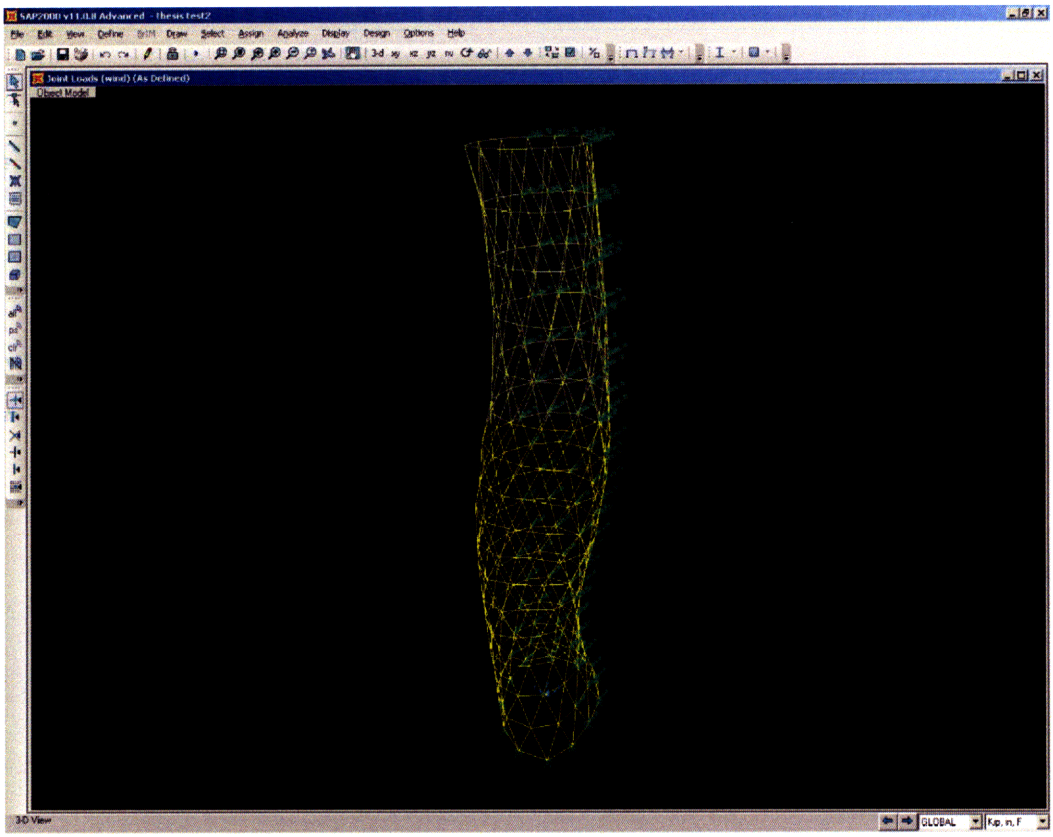


Figure 29 Design Example after imported into SAP2000

Since all of the load cases have been predefined, the user needs only to initiate the run sequence for the analysis. Select the analysis tab and then click "Run Analysis." The proper load cases should display in a dialogue box as in the screen capture on the following page. If the user wants to visualize the displacement that the tower is predicted to undergo, the resulting displacement drawing can be found under the display tab as "Show Deformed Shape."

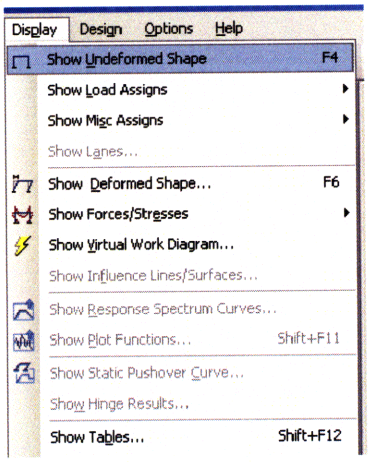


Figure 30 SAP display menu

6.2 SAP output

Most importantly, every time a model is run, SAP stores the run's numerical nodal displacements in a series of tables that can be found under the display tab. The evaluation of the preliminary structural design relies mainly on the numerical displacement outputs from the model. After selecting to view the tables, the user can opt to only view the displacement table for one particular load case by checking or deselecting the appropriate boxes.

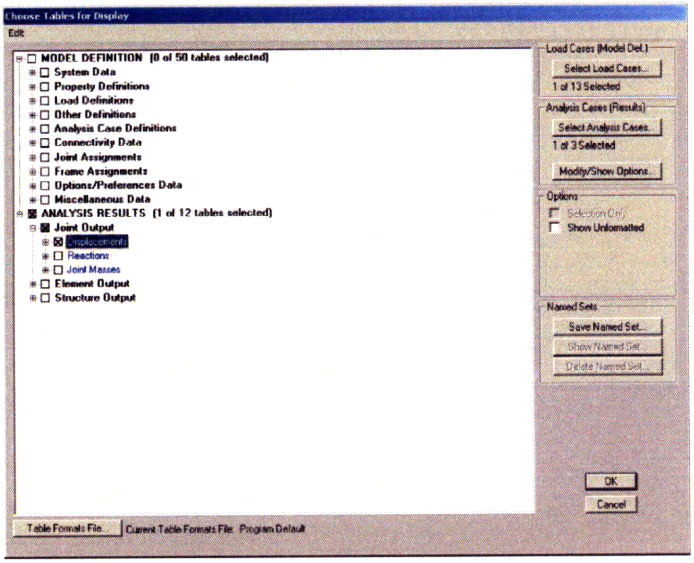
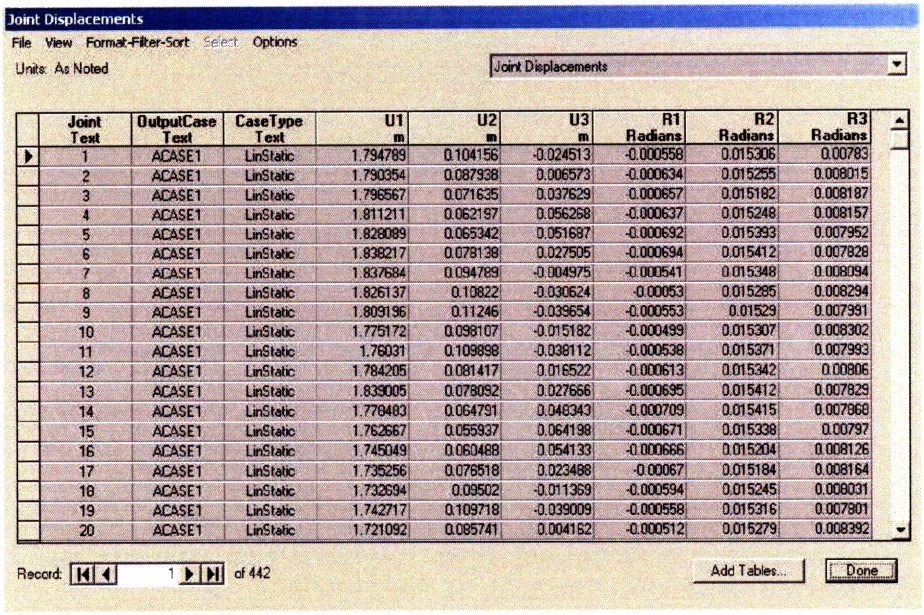


Figure 31 SAP displacement table selection window

The displacement data is stored in columns that hold the Joint index, the selected load output case and its type, the linear displacements in the x, y, and z-directions in meters, and the rotational motions around all three axes in radians. After this customized displacement table loads, the table's GUI has a button for the option to save and export the table as an Excel file. With limited automation, the user can manually select this option; however, with access to SAP's internal code workings, it would be possible to run a script that selects within the menu and the subsequent windows to view the desired table and then exports the data table as an Excel file—automatically.



Joint Text	OutputCase Text	CaseType Text	U1 m	U2 m	U3 m	R1 Radians	R2 Radians	R3 Radians
1	ACASE1	LinStatic	1.794789	0.104156	-0.024513	-0.000558	0.015306	0.00783
2	ACASE1	LinStatic	1.790354	0.087938	0.006573	-0.000634	0.015255	0.008015
3	ACASE1	LinStatic	1.796567	0.071635	0.037629	-0.000657	0.015182	0.008187
4	ACASE1	LinStatic	1.811211	0.062197	0.056268	-0.000637	0.015248	0.008157
5	ACASE1	LinStatic	1.828089	0.065342	0.051687	-0.000692	0.015393	0.007952
6	ACASE1	LinStatic	1.838217	0.078138	0.027505	-0.000694	0.015412	0.007828
7	ACASE1	LinStatic	1.837684	0.094789	-0.004975	-0.000541	0.015348	0.008094
8	ACASE1	LinStatic	1.826137	0.10822	-0.030624	-0.00053	0.015285	0.008294
9	ACASE1	LinStatic	1.809196	0.11246	-0.039654	-0.000553	0.01529	0.007991
10	ACASE1	LinStatic	1.775172	0.098107	-0.015182	-0.000499	0.015307	0.008302
11	ACASE1	LinStatic	1.76031	0.109898	-0.038112	-0.000538	0.015371	0.007993
12	ACASE1	LinStatic	1.784205	0.081417	0.016522	-0.000613	0.015342	0.00806
13	ACASE1	LinStatic	1.839005	0.078092	0.027666	-0.000695	0.015412	0.007829
14	ACASE1	LinStatic	1.778483	0.064791	0.048343	-0.000709	0.015415	0.007968
15	ACASE1	LinStatic	1.762667	0.055937	0.064198	-0.000671	0.015338	0.00797
16	ACASE1	LinStatic	1.745049	0.060488	0.054133	-0.000666	0.015204	0.008126
17	ACASE1	LinStatic	1.735256	0.076518	0.023488	-0.00067	0.015184	0.008164
18	ACASE1	LinStatic	1.732694	0.09502	-0.011369	-0.000594	0.015245	0.008031
19	ACASE1	LinStatic	1.742717	0.109718	-0.039009	-0.000558	0.015316	0.007801
20	ACASE1	LinStatic	1.721092	0.085741	0.004162	-0.000512	0.015279	0.008392

Figure 32 Joint displacement output table

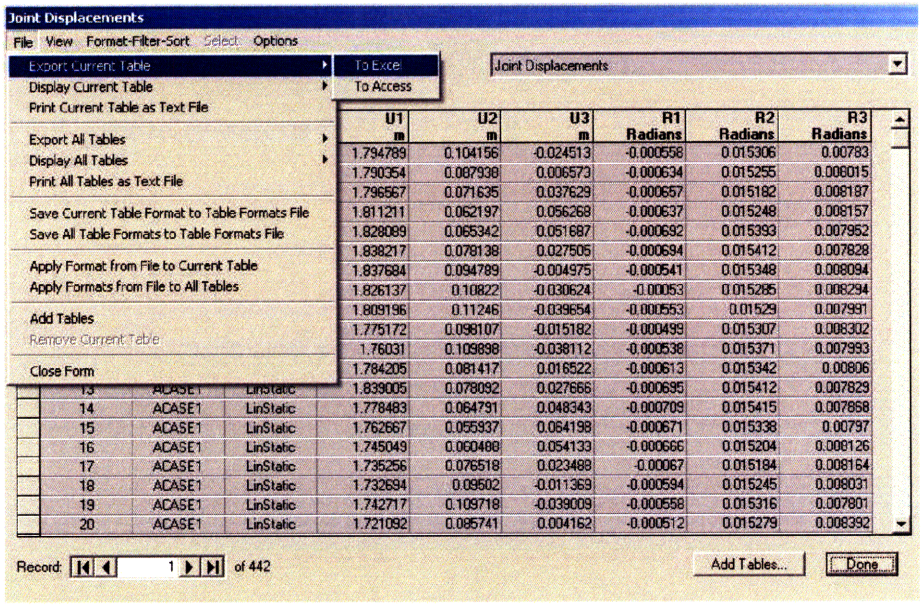


Figure 33 Joint displacement manual export to Excel

Without the software license permission and convenience for now, the manual option to export the spreadsheet will do. This sheet will be referred to as the filename deflections_1.xls, so that an integer count can be made to store the deflection data and increment each time the model iterates and passes through the structural analysis. With the collection of displacement data stored, the design study can progress to the final component—the evaluation of results and feedback loop interaction with the user.

6.3 Limitations

The current SAP analysis only considers the overall displacement of the structure. Moreover, it does not evaluate the system for member buckling control. Therefore, it does not check to see if buckling of the independent frame elements is dominant for the design above overall structural displacement. Secondly, the model only accounts for a single load case. In the future, the loading cases that the model runs can be expanded to include a more substantial range of reasonable building loads to further validate the outcome of the analysis. Furthermore, this process model was developed to initiate an investigation into the opportunities of an integrated computational design process, and much further development is necessary to provide results that are satisfactory for actual structural design.

7 Evaluation and Iteration

The final component of the computational design integration study is another collection of intermediary steps between the architectural and structural software packages. The logic behind what has the potential to become this script is simple. Now that the member displacements have been calculated based on the desired wind live load, they must be measured against the displacement limit criteria for tall buildings of L/300 to L/500. If the displacements at the top of the structure are found to be within the acceptable limits for tall buildings, the design iteration process will not have to cycle again, and the satisfactory model can exit the loop. If the displacements of the top joints are found to be greater than the allowable movement, then the user must make a decision. The options include: changing overall geometry and starting from scratch, changing the density of the diagrid, or increasing the building's overall stiffness by increasing the steel member cross sections. After coming to a decision, the user will initiate the model back into the Architectural Process or Intermediary Program, and the design cycle will repeat again to refine the design based on the user's choices.

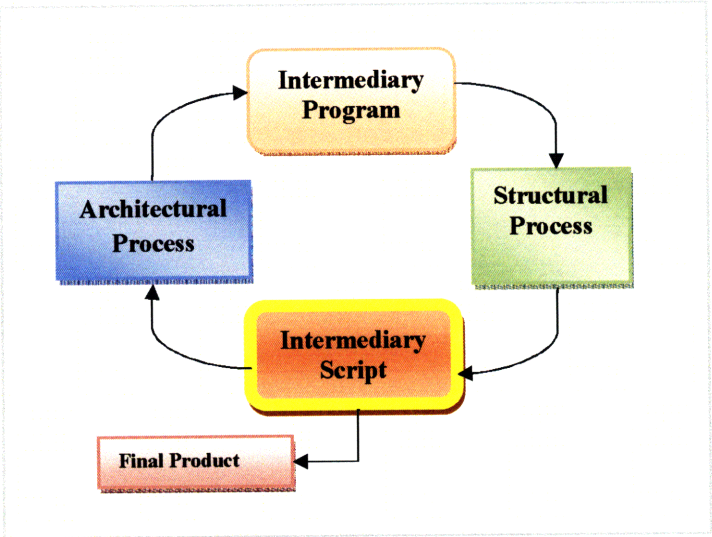


Figure 34 Final Stage Highlight within the Integrated Design Process

7.1 Evaluation through Excel interface

In greater detail, the exported spreadsheet of deflection data for the processed model will serve as a workspace for evaluating the building along the deflection criteria. As in the Excel algorithms in previous sections, an external macro within the intermediary parent excel file can be developed using Microsoft's VBA language that will first take action to load the spreadsheet displacements_1.xls that can be seen in Figure 35. The script should then initialize a sequence to organize, alter, and evaluate data within the document. Ignoring the torsional effects for this simplified study, or the columns of rotational motions, the script can focus on evaluating the linear displacements. The script can cycle through each joint and calculate the joint's total displacement. To find this magnitude, the parent script can add an additional column to the right of the sixth column, F, where the following function can be entered:

$$=\text{sqrt}((Di)^2+(Ei)^2+(Fi)^2)$$

This additional column can be seen in Figure 36. By summing the values' squares and then square-rooting the quantity, the absolute displacement is found for the corresponding joint i. Then, the script can have an algorithm that loops through each absolute displacement to find the maximum value and returns the value. Finally, the parent VBA script can compare the maximum displacement to the allowable displacement for the maximum height of the tower as derived from the original geometry's storage of joint coordinates. The result can be stored as a Boolean variable within the parent function that will dictate what the user is prompted with next.

	A	B	C	D	E	F	G	H	I	J
1	TABLE: Joint Displacements									
2	Joint	OutputCase	CaseType	U1	U2	U3	R1	R2	R3	
3	Text	Text	Text	m	m	m	Radians	Radians	Radians	
4	1	ACASE1	LinStatic	1.794789	0.104156	-0.024513	-0.000558	0.015306	0.00783	
5	2	ACASE1	LinStatic	1.790354	0.087938	0.006573	-0.000634	0.015255	0.008015	
6	3	ACASE1	LinStatic	1.796567	0.071635	0.037629	-0.000657	0.015182	0.008187	
7	4	ACASE1	LinStatic	1.811211	0.062197	0.056268	-0.000637	0.015248	0.008157	
8	5	ACASE1	LinStatic	1.828089	0.065342	0.051687	-0.000692	0.015393	0.007952	
9	6	ACASE1	LinStatic	1.838217	0.078138	0.027505	-0.000694	0.015412	0.007828	
10	7	ACASE1	LinStatic	1.837684	0.094789	-0.004975	-0.000541	0.015348	0.008094	
11	8	ACASE1	LinStatic	1.826137	0.10822	-0.030624	-0.00053	0.015285	0.008294	
12	9	ACASE1	LinStatic	1.809196	0.11246	-0.039654	-0.000553	0.01529	0.007991	
13	10	ACASE1	LinStatic	1.775172	0.098107	-0.015182	-0.000499	0.015307	0.008302	
14	11	ACASE1	LinStatic	1.76031	0.109898	-0.038112	-0.000538	0.015371	0.007993	
15	12	ACASE1	LinStatic	1.784205	0.081417	0.016522	-0.000613	0.015342	0.00806	
16	13	ACASE1	LinStatic	1.839005	0.078092	0.027666	-0.000695	0.015412	0.007829	
17	14	ACASE1	LinStatic	1.778483	0.064791	0.048343	-0.000709	0.015415	0.007868	
18	15	ACASE1	LinStatic	1.762667	0.055937	0.064198	-0.000671	0.015338	0.00797	
19	16	ACASE1	LinStatic	1.745049	0.060488	0.054133	-0.000666	0.015204	0.008126	
20	17	ACASE1	LinStatic	1.735256	0.076518	0.023488	-0.00067	0.015184	0.008164	

Figure 35 view of displacement_1.xls file

	A	B	C	D	E	F	G	H	I	J
1	TABLE: Joint Displacements									
2	Joint	OutputCase	CaseType	U1	U2	U3	Abs. U	R1	R2	R3
3	Text	Text	Text	m	m	m	m	Radians	Radians	Radians
4	1	ACASE1	LinStatic	1.794789	0.104156	-0.024513	1.7979758	-0.000558	0.015306	0.00783
5	2	ACASE1	LinStatic	1.790354	0.087938	0.006573	1.7925244	-0.000634	0.015255	0.008015
6	3	ACASE1	LinStatic	1.796567	0.071635	0.037629	1.7983883	-0.000657	0.015182	0.008187
7	4	ACASE1	LinStatic	1.811211	0.062197	0.056268	1.8131519	-0.000637	0.015248	0.008157
8	5	ACASE1	LinStatic	1.828089	0.065342	0.051687	1.8299865	-0.000692	0.015393	0.007952
9	6	ACASE1	LinStatic	1.838217	0.078138	0.027505	1.8400826	-0.000694	0.015412	0.007828
10	7	ACASE1	LinStatic	1.837684	0.094789	-0.004975	1.8401337	-0.000541	0.015348	0.008094
11	8	ACASE1	LinStatic	1.826137	0.10822	-0.030624	1.8295972	-0.00053	0.015285	0.008294
12	9	ACASE1	LinStatic	1.809196	0.11246	-0.039654	1.8131216	-0.000553	0.01529	0.007991
13	10	ACASE1	LinStatic	1.775172	0.098107	-0.015182	1.7779458	-0.000499	0.015307	0.008302
14	11	ACASE1	LinStatic	1.76031	0.109898	-0.038112	1.7641489	-0.000538	0.015371	0.007993
15	12	ACASE1	LinStatic	1.784205	0.081417	0.016522	1.7861381	-0.000613	0.015342	0.00806
16	13	ACASE1	LinStatic	1.839005	0.078092	0.027666	1.8408702	-0.000695	0.015412	0.007829
17	14	ACASE1	LinStatic	1.778483	0.064791	0.048343	1.7803193	-0.000709	0.015415	0.007868
18	15	ACASE1	LinStatic	1.762667	0.055937	0.064198	1.7647224	-0.000671	0.015338	0.00797
19	16	ACASE1	LinStatic	1.745049	0.060488	0.054133	1.7469359	-0.000666	0.015204	0.008126
20	17	ACASE1	LinStatic	1.735256	0.076518	0.023488	1.7371011	-0.00067	0.015184	0.008164

Figure 36 displacement_1.xls used to find absolute displacements

7.2 User Options, Interaction, and Feedback

This stage is possibly the most significant in the whole iterative process in regards to achieving the goals of this design study. After the structural model output is evaluated along displacement criteria, the user can face the following options as a function of the evaluation result.

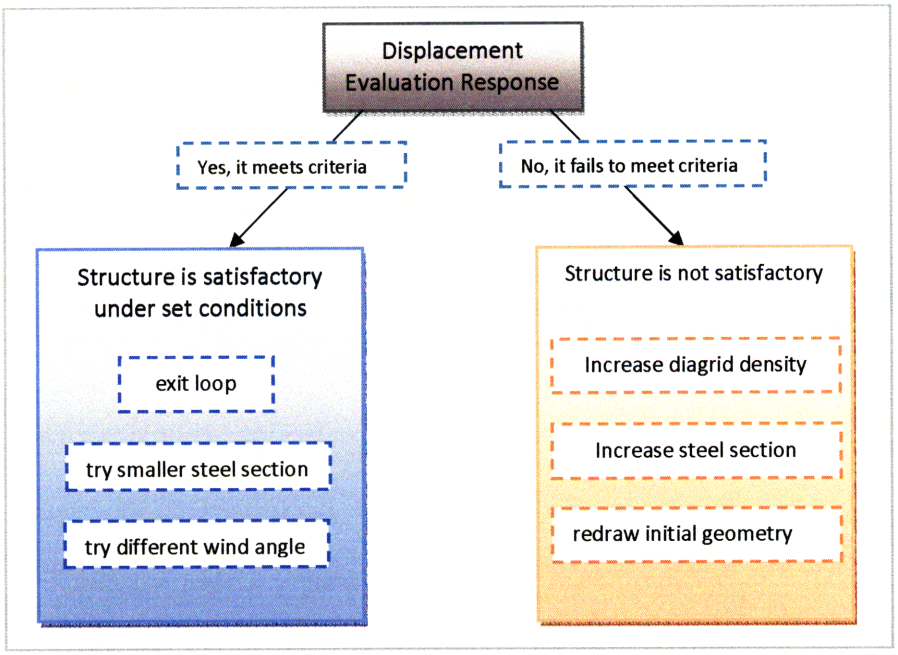


Figure 37 Flowchart of user options

7.2.1 Success Options

In the case of a success, defined as passing the deflection limit test, there are, but not limited to, three options for the user. He or she can choose to exit the design loop with a general design based on the original geometry and default member cross section. The user can refine the member cross section design further beyond this preliminary study through further iterations. Thirdly, the success can be validated through additional iterations by varying lateral wind force angles and confirming that the resulting displacement vectors are still within the acceptable limits.

When made available, the first choice should come with an interpretation of the structural results. Even though the displacements passed the comparison test, the model only represents a *single* wind load direction. Also, the final appearance was not rendered featuring the current cross section scheme which could be an unfavorable design. The quit-and-run option should not be encouraged since iteration is a valuable part of the design and learning process for this computational model.

The option for designating and testing a smaller steel section can be beneficial to the designer. The user can take the current steel cross section dimensions and apply them to the linear elements in the architectural model. That way, the architectural model will be a more accurate representation of how the

diagrid modules will appear. After evaluating the appearance of the current option, another algorithm could be run that calculates the difference in the volume of steel required for what would be the next iteration versus the current cross section solution.

With the concern brought up about the validity of the structural analysis when discussing the first option, another more promising choice would be to iterate and sample deflections for a variety of wind angles. This option could be addressed by providing warnings about asymmetrical or more extreme free form buildings that could have messy wind effects when perturbed from different angles.

These are only a few ways that a success can be handled within the software constraints. The real value in this study comes from the analysis and decisions that arise from structural evaluation outcomes that are failures.

7.2.2 Failure Options

When the displacement test fails, the user has the best opportunity to gain insight about structural analysis. If the displacement is greater than the limit value, the stiffness of the building can be increased to reduce the deflection at the top. To do this, the density of the diagrid can be increased, the steel member cross section can be increased, or ultimately, the initial geometry can be redrawn. By increasing the stiffness, all of these choices run the risk of bringing the stiffness within a range where the effects of other dynamic loadings, like ground motion, could cause damage to the structure.

Increasing the density of the diagrid would affect the final appearance of the surface's structural system. Altering the number of floors per vertical diagrid module and/or the number of horizontal modules will change the angles in which the frame elements are aligned. Changing these angles will redistribute how the vertical or horizontal forces are being carried as a result of stiffness changes for vertical or lateral loading cases—as seen in Section 2.1. The vertical module density, horizontal module density, or density of both can be increased to try to reach the displacement goals.

Secondly, the cross section properties of the frame element also factor into the stiffness of the structural system. If the material strength, or modulus of elasticity, E, is held constant, increasing the cross sectional area can amplify the stiffness in both the vertical and horizontal directions.

Finally, if the user is unwilling to increase the density of either diagrid module or frame element size, a new tower surface can be designed altogether. The new design may raise the aspect ratio, which is the ratio of the tower's base dimension to its total height. Towers typically have an aspect ratio equal to or greater than 1:7, so the user can redraw geometry that has a larger base area in comparison to the prior design trial.

7.3 Design and Re-design Loop

7.3.1 Success Decisions

If the user opts to test a smaller steel section, he or she can test a reduction of the steel frame's cross section and measure its effect on the displacement of this updated model. The model can be put back through the design loop by bypassing the Architectural Process and passed into to the Architecture-to-Structure intermediary script. The parent VBA script can then update the tables "Frame Section Assignments" and "Frame Section Properties 01 – General" within the *.$2k translated Excel file. The new section data will have to be found in the section material library of the SAP software or the properties must be personally assigned. It would be useful to compile a spreadsheet file during development that contains all of the section material data sets that are inherent to SAP. The definition of the chosen cross section can be stored in the parent Excel function along with the iteration number so that future comparisons can be made between iteration choices using archived data. When dealing with an iterative process, it is helpful to have logs of data from past iterations in order to gauge progress.

After passing the displacement criteria test, the stored cross sectional properties can be used to apply the exact cross section shapes to the otherwise linear members in the architectural model. By applying the shape, the user can decide if the visual outcome is satisfactory. To apply the cross section shapes, additional scripting is necessary. Scripts must be written that extract the chosen cross section from its storage within the Excel document's log of material applied per iteration; next, that property set must be applied to the model by drawing surfaces within the architectural program for visualization.

Another second option that could be made available to the user would be to enter the Architectural Process after the diagrid is drawn and redefine the wind force angle. This way, the structural performance can be measured from multiple angles since the free form asymmetry implies that its behavior will vary. The RhinoScript file AddDiagrid will have to be altered and segmented in a way that would allow for re-entry with a pre-drawn diagrid model (hence, bypassing the original functions that draw the surface and diagrid geometry) and only apply the wind load vectors. After the model updates the wind vectors, the geometry and wind load data will have to be exported again using the DiagridExport script. Then, the model can pass through the intermediary script to rearrange the data again for SAP. Repeating this step for multiple angles will help to test the behavior of the free form structure when loaded from different angles. The irregular geometry of the free form shape allows for concavities and convexities that will affect the resulting displacement behavior differently when loaded at other angles. If the model continues to pass the displacement criteria test, the architectural design will be increasingly more validated.

Another adjustment worth exploring would be a reduction in the density of the diagrid module while, at the same time, increasing the area of the diagonal frame cross section. This option may take multiple iterations to get a sufficient end result by optimization of attributes in opposite directions. The

following sub-section details the method by which the diagrid module can be altered, and the same algorithm for the cross section applies as above except in this case, the next larger section will be used.

By iterating between these two choices, the user can refine the appearance of the structure and confirm its performance in multiple directions. Here, it is evident that the process of iteration is not exclusively beneficial to the failure scenarios; the user may gain an understanding of structural behavior by iterating and adjusting variables on a successful structural model.

7.3.2 Failure Decisions

In order to increase the density of the diagrid modules, the whole model must be put back through the geometric RhinoScript. The current script can be further developed to process an existing model by clearing all diagrid lines and wind vector lines to restart the drawing functions. The same surface can be re-evaluated, and the user can be directed by the same prompts to input the new values for the diagrid module. Once again, this is a scenario where the master Excel file could be used to log the previous iteration's diagrid module choices, and the most recent iteration's values can be returned within the prompt window that asks for that variable's new value. For instance, if the user wants to reduce the vertical diagrid module from 5 to 4 stories, the new dialogue box can read, "Input the number of floors per diagrid module. (previous: 5)," and the user can input 4. Likewise, the same alterations to the code can be made to extract the log of the previous horizontal module choices. The iteration process can continue as usual after the geometry is redrawn to the new specifications.

If the user chooses to increase the cross sectional area of the diagonal members, the automation process can behave in the same way as reducing the cross sectional area that is described above. The model can be put back through the design loop by bypassing the Architectural Process and passed into to the Architecture-to-Structure intermediary script. The master Excel file can access the log of material sections for the most recent iteration and find the next size increment from the material list of SAP recognized sections. The current material can be reset by the parent Excel function and updated within the *.$2k file, and then, the model can be processed normally in SAP.

When the user chooses to completely redefine the surface geometry, the script must only be updated to include a function that wipes all previous diagrid lines and wind vectors within the architectural model generated by RhinoScript. The script should move the surface model of the first iteration to a ghost layer. Then, the user can redraw a new geometry and maintain access to the previous model for comparison. The user can view the old model to ensure that the aspect ratio of the tower was improved or extreme geometry deviations were avoided that may have caused issues during past runs.

By changing the properties of the model in the ways described above, the user can gain an understanding of the effects of cross section or geometry changes on overall lateral displacement. The changes made to the variables will ultimately be representative of changes in the overall stiffness of the

structure which determines the displacement under the quasi-static lateral load. The choices outlined above can be used in coordination with each other to achieve different design goals. Further development of the script is necessary to provide for actual implementation of these options.

8 Model Conclusions

Currently, the process model serves as a starting point to develop integrated design methods through computation. It begins to automate the iterative process between architectural design and structural design that is otherwise accomplished by a repetitive dialogue between parties on both sides of the building design disciplines. It allows the user, or designer, to weigh options for building attributes while providing a framework for these options to be evaluated. Only then can the user comprehend the affect of possible combinations on the structural performance.

8.1 Opportunities for Improvement and Development

In order to achieve the valid re-entry steps discussed in Section 7.3, the current script collection must be altered and enhanced. Much like the iteration development described in that section, the intermediary processes that have been described in Section 5 require development of their corresponding scripts. The intent has been covered, yet the coding platform for these steps has not been scripted. The coding of translation between spreadsheet documents will be expedient after the hurdles from unfamiliarity with the language are passed. In addition to these coding exceptions, there are some attributes that can be analyzed before the cycle of iteration initiates in order to minimize the required number of iterations.

To avoid excessive iterations, an estimate for the required cross section can be found to dictate a more educated starting place. In order to provide a more accurate material selection at the start, a preliminary calculation can be done to optimize the floor stiffness for the maximum allowable displacement. An algorithm can be developed which will optimize the stiffness based on the building's first mode. After finding the required stiffness, a sample floor can be chosen from the building. The geometry of this floor can be used to calculate the cross sectional area necessary to achieve the desired value of stiffness as determined above. This cross section will become a more accurate starting place for the subsequent design iterations.

Likewise, this program can be expanded to account for a more randomized triangulation of diagrid modules. For this to occur, the script must enclose triangulated shapes on the surface which do not overlap and maintain certain relationships to one another. To avoid excessive customization of parts, the script can set basic sizes of frame elements and fit these particular elements in an optimized fashion to the surface coordinate system. An example of a possible randomized pattern can be seen to the right. This new capability could be more advantageous for users who do not like the regularized diagrid module system.

8.2 Ideal Direction and Possibilities for this Script

Ideally, this set of scripts can be collected into a program that will coordinate the iterative design process that has been outlined. After it has been assembled, the iterative method can be used by designers who wish to create tower geometries that have regularized diagrid skins. It can hopefully serve as a tool or starting place when pursuing discussions with structural engineers. Likewise, this method can be used as a template on which to model other similar tools for different structural systems. The iterative method and organization of data optimization can provide an outline for additional software development.

By providing a framework of educated options, the designer becomes empowered and better prepared to converse with the engineer to express their most critical and valuable desires. The software package not only enhances verbal communication, but digital data communication as well. The result is a more seamless and coordinated design process between architectural designer and structural engineer. Ultimately, a smoother design process could mean a better quality of design or end product.

Also, this iterative process would ideally be instantaneous as mentioned in Section 1.4. Preferably, the designer could be defining architectural geometry while there is real-time structural sampling. The intended structural system can be continuously tested within the defined geometry for performance. In this case, the diagrid process laid out in this thesis would be only one structural system that is a subset of a greater toolbox of possible applied structural systems.

9 Future of Integrated Design

With the expansion of technology and the launch of coordination between disciplines such as the BIM method, barriers are being broken down between fields in the building design chain. The coordination of data allows for improved communication as well as streamlined processes that were once impossible. Integrated design will most likely have a strong impact on the fabrication industry and expand well beyond the partnership pursued in this thesis of just engineers and architects.

9.1 Fabrication and Construction

Ideally, by storing all building information within spreadsheet and data arrays, this information could be useful to the other building disciplines such as construction and fabrication. As fabrication methods become increasingly digitized and controlled by computers, computerized data sets of building elements become more valuable. If compatible file formats are used or generated from existing data, the fabricators can easily use computer controlled machinery to assemble the desired elements. This process is known as Computer Aided Manufacturing, or CAM.

Prototyping and building fabrication processes can be easily accelerated through advanced computing in coordination with rapid prototyping devices such as 3D printers, laser cutters, and stereo lithography, or NC (Numerical Control)[8] tools such as plasma cutters, NC milling (metals, stone, plastics, and other solids), NC routing (aluminum, foam and wood), and water jet cutting (aluminum, steel, glass, stone, ceramics, and wood) for thicker materials. Most of these tools can function by a more modern CNC fashion, or Computer Numerical Control. Figure 38 above shows a water jet cutting through 6.5 inches of titanium; a water jet cutter is capable of cutting through solid metals of up to 18 inches (45 cm) thick with water pressures of up to 90,000 psi[10]. Elements on cut sheets can be laid out systematically in the most efficient way to optimize the use of material to limit waste, and the machine outputs no hazardous material, making the method more "green." This machine, if scaled appropriately, can be used in the fabrication of customized substantial building elements along with the other mentioned CAM methods.

Figure 38 Water jet cutting through solid titanium[9]

For example, a complex façade such as the one pictured in Figure 39 and Figure 42 of Peter Cook's Graz Museum (Kunsthaus Graz) in Graz, Austria can only be accurately manufactured by coordinating digital information with intelligent machinery[8]. Likewise, the joint connections and façade details for the Swiss Re tower by Foster and Partners required computer aided design to generate the building details and connections (Figure 40) that varied on every vertical level as seen in Figure 41. The same computer aided design tools that were used to generate the original form were used to devise a systematic way to realize the building diagrid elements.

Currently, these methods are used for what seem to be very customized projects; however, in order to realize designs of complex forms, these methods are used universally. Traditional construction

Figure 39 Image of Graz Museum from distance[11]

Figure 42 Image of Peter Cook's Graz Museum façade detail[8]

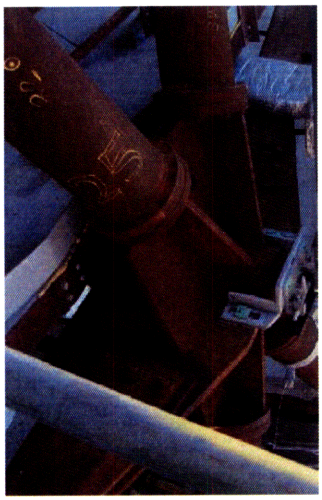


Figure 40 Swiss Re connection detail[12]

Figure 41 Image of Norman Foster's Swiss Re Tower under Construction[13]

techniques do not provide the necessary threshold of accuracy as these computer controlled methods. When building elements cease to be identical, the manpower associated with proper completion of the final product is limitless for what starts to be an inefficient approach. With digital fabrication, the time and energy associated with identical or customized elements is comparable, if not the same. Just as scripting can help prepare and transfer models for CAM, fabrication limitations can similarly be defined within design scripts to guarantee that each building element is within said limits. This embedded knowledge can help ensure constructability even before building data is passed through to contractors and other construction parties. Therefore, the push towards more complex or free form building designs warrants the advancement of construction methods towards more computerized processes. Thus, the way that the construction industry thinks of itself must be entirely reinvented.

9.2 Integration with other Coded Design Methods

In addition to enhancing the coordination with construction processes, integrated computer models also have the ability to access other building specific data to optimize the architecture along many performance measures. Eleftheria Fasoulaki discusses the incorporation of lighting optimization in generative building models in her thesis for the Master of Science in Architectural Studies degree at MIT[14]. She used a similar design algorithm, seen in Figure 43, between RhinoScript, data storage in Excel, and Ecotect, a lighting simulation program, to generate forms based on inputs of site data and

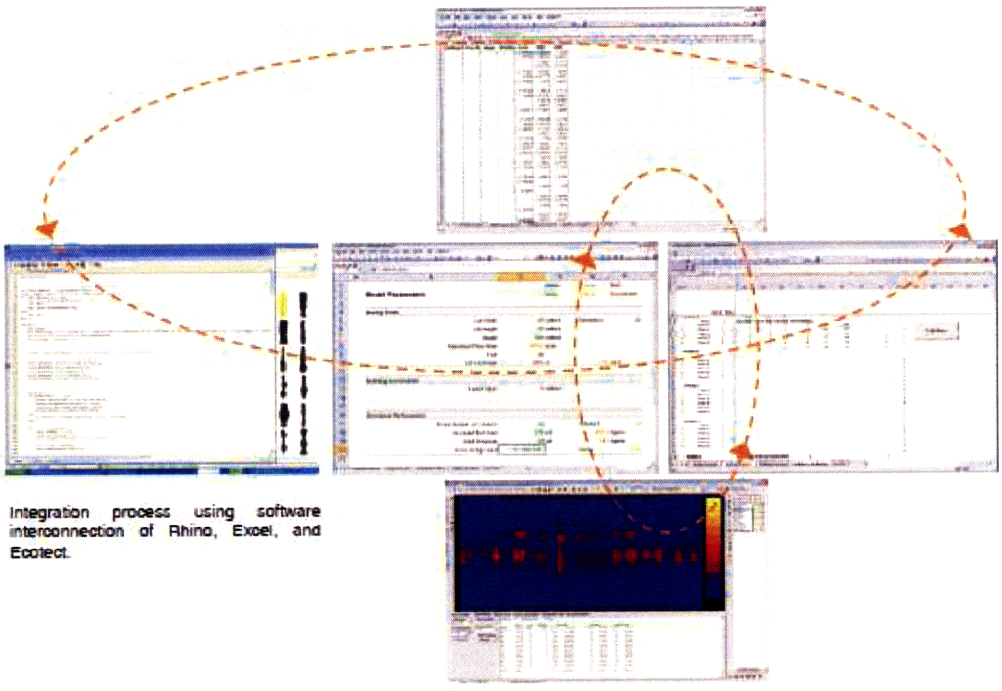


Figure 43 Representation of Fasoulaki's algorithm[14]

building geometry. Furthermore, she presents a model which lets the user weigh preferences for variables of basic structural, solar, and zoning definitions to give different resulting modes of optimization.

Similarly, the model outlined in this thesis for a more detailed structural design could be incorporated with other simulation data to optimize along multiple parameters. These parameters can be defined from MEP, developer, environmental engineering, or urban planning perspectives. At a base level, the parameters can be evaluated to guarantee fundamental limits have not been met. Then, weights can be assigned for other parameters such as daylighting, natural airflow, material use, site interaction or views, and/or seismic performance. For example, the diagrid module spacing can be optimized in two additional ways: i) for use of predetermined member lengths rather than strict user input for spacing, allowing for more regular sizing which could reduce material waste or simplify the fabrication process, and ii) to allow for unobstructed views on certain levels of the building in set directions.

If companies from all disciplines of building design moved away from use of proprietary software and towards the sharing of coordinated data, buildings can be optimized along multiple parameters from the start of the design process. Architects could then gain an intuition for the contribution of different requirements on building design. If a building is designed more intelligently, time and effort can be saved throughout the remainder of the project.

10 Design Implications and Limitations

Although the idea of linking structural design and architectural design together through imbedded scripted calculations is extremely powerful, the array of new tools, both available and possible, have the potential to limit design. When critiquing the scripted integrated design approach, the more overarching design implications must be considered.

10.1 What it Means for Architecture and Structural Engineering

The natural progression of architectural design flows through a problem statement, conceptual design approaches, iteration and refinement, and finally, a proposed design. The forms available to an architect are nearly limitless, while the chosen form may have specific, prescribed structural needs. Even though the overall structural typology may be decided, the intricate design of the structure may have a variety of solutions. It is here that the largest issues with imbedded scripting begin to unfold.

A script, as such outlined in this thesis, is written to perform a specific algorithm. That algorithm may have a variety of conditional networks that compare variables and make decisions based on those variables, yet all possible outcomes *must* be inherent in the script. Solutions that a computer arrives at may be unusual and therefore subject to evaluation and possible rejection through a human's judgment; however, a solution can never be entirely born from the computing tools. The computer cannot generate something that it has not been taught to create. Therefore, the tools available to a designer can only create that which has already been designed or coded.

This has serious implications for the prospects of innovative engineering and design. It seems as though we are in danger of taking a very innovative tool and destroying innovation itself. If these tools are relied on completely, we limit ourselves as designers—both structural and architectural—to the cards we hold in our hand. Design no longer evolves through discoveries and lessons learned on a project-to-project basis. Lessons are no longer carried over to act as a framework and foundation for subsequent design challenges. With these tools, we limit ourselves to the solutions that have been devised before, unless we innovate and expand the coding to allow for the new methods.

11 References

[1] Sundberg, J. et al., "Phaeno Science Center: Convention vs. Innovation," Harvard Graduate School of Design, In Search of the Engineer, Assignment 3, April 2009.

[2] http://archrecord.construction.com/projects/portfolio/archives/images/0608hearsTower_lg.jpg

[3] http://www.abbeyoffices.com/_assets/locationimages/Gherkin.jpg

[4] www.kpf.com

[5] Connor, J., "Diagrid Lecture," 1.571 Lecture, Fall 2008.

[6] McNeel, Robert. "RhinoScript[101], Primer Collection" pp. 48.

[7] http://www.e-architect.co.uk/new_york/jpgs/hearst_tower_section.jpg

[8] Sass, L., "Rapid Prototyping," MIT OpenCourseWare, 4.212 Lecture 4, Spring 2003.

[9] http://www.jetedge.com/content.cfm?fuseaction=dsp_applications_101

[10] Wikipedia waterjet definitions

[11] Wikipedia.com graz museum

[12] Sass, L., "Parametric Modeling," MIT OpenCourseWare, 4.212 Lecture 9, Spring 2003.

[13] Sass, L., "Introduction: History and Theory," MIT OpenCourseWare, 4.212 Lecture 1, Spring 2003.

[14] Fasoulaki, E., "Integrated Design: A Generative Multi-Performative Design Approach," MIT Thesis, June 2008.

12 Appendix

DrawDiagrid Script

```
Option Explicit

Call Main
Sub Main()

  'get user input surface
        Dim strChoice, intChoice
        Dim strShape
        Dim intStories
        Dim intGrid
        Dim dblRadius

        intChoice = Rhino.MessageBox("Generate your own surface?",4, "User Generated Surface")

        If intChoice = 6 Then
    'take user surface, divide by 3.5 and only consider whole floors
                'ask user the number of diagrid modules
                Dim strGeo, strSurf
                strGeo = Rhino.GetObjects("Select curves to loft for tower geometry",4)
                strShape = Rhino.AddLoftSrf(strGeo,,,0)

                'get highest point of surface
    Dim arrBBox
    arrBBox = Rhino.BoundingBox (strShape)
    Dim arrZPoints, arrYPoints
    Dim i,j
    Dim arrBBoxTemp
    Dim numOfBBox : numOfBBox = UBound (arrBBox)
    ReDim arrZPoints(numOfBBox)
    ReDim arrYPoints(numOfBBox)

    For i = 0 To numOfBBox
      arrBBoxTemp = arrBBox(i)
      arrZPoints(i) = arrBBoxTemp(2)
      arrYPoints(i) = arrBBoxTemp(1)
      Rhino.Print arrZPoints(i)
    Next

                Dim intHeight : intHeight = Rhino.Max(arrZPoints)
                dblRadius = Rhino.Max(arrYPoints)
                Dim intRem
                intRem = ((10*intHeight) Mod 35)/10
                intStories = ((intHeight - intRem)/(7/2))
                'translate number of stories to number of diagrid modules in the vertical direction
                Dim strNotice : strNotice = Rhino.MessageBox("There are " & CStr(intStories) & _
                        " total stories.", 0,"Vertical Diagrid Module")
                Dim intGridZ : intGridZ = Rhino.IntegerBox("Input the number of floors per diagrid module."
,5,"Vertical Diagrid Module")
                Dim intVertical 'the number of final diagrid modules vertically
                If (intStories Mod intGridZ = 0) Then
                        intVertical = (intStories/intGridZ)
                ElseIf ((intStories Mod intGridZ) < (0.5*intGridZ)) Then
                        intVertical = ((intStories - (intStories Mod intGridZ))/intGridZ)
                Else
                        intGridz = Rhino.IntegerBox("Number invalid. Number must be a factor of :   " _
                                & CStr(intStories),,"Re-try: Vertical Diagrid Module" )
                        intVertical = ((intStories - (intStories Mod intGridZ))/intGridZ)
                End If

                intGrid = Rhino.IntegerBox("Input the number of HORIZONTAL modules.", 12,_
                        "Horizontal Diagrid Module")
                intGrid = intGrid*2
        Else
                'fill in... if user doesn't want to draw surface
        Dim intChoiceNo
        intChoiceNo = Rhino.MessageBox("Please draw closed curves before running this script.", 0, "Error")
  End If

        Dim arrSegmentSurface
        arrSegmentSurface =  segmentSurface(strShape, intVertical, intGridZ, intGrid, dblRadius)
```

```
        'add new layer for diagrid lines, make current
        Rhino.AddLayer "Diagrid Lines", RGB(200,50,200)
        Rhino.CurrentLayer "Diagrid Lines"
        ''Draw diagrid On surface
        Call addDrawDiagrid(strShape, intVertical, intGridZ, intGrid, dblRadius, arrSegmentSurface)
        Rhino.CurrentLayer "Default"

        Call addWindVector(strShape, arrSegmentSurface, intVertical, intGrid)


  'give diagrid members piped dimension
 ' Dim arrCurve
        'arrCurve = Rhino.getObjects("select all curves ",4 )
        'drawTubesForEveryCurve(arrCurve)

End Sub


Function segmentSurface(strShape, intV, intGZ, intG, dblR)
        Dim i,j

        ''VERTICAL LINE of PTS
        'makes an invisible vertical line of points as tall as the tower, adjacent to the tower
        Dim arrCvPts ()
        Dim arrUVPtParam
        Dim arrLocation
        Dim arrPoint, strIso, arrPtIso
        Dim arrDivPts(200,200)

        Rhino.AddLayer "Column Points",RGB(50,100,200)
        Rhino.CurrentLayer "Column Points"
        For i = 0 To intV
                'ensures identical floor-to-floor height is maintianed
                arrPoint = Array(0,dblR*2,i*3.5*intGZ)

                'Rhino.addPoint(arrPoint)
                'find the point on the surface closest to the current cartesian pt at the same
                'Zvalue
                arrUVPtParam = Rhino.SurfaceClosestPoint(strShape(0), arrPoint)
                'extract the isocurve that goes through the point on surface
                strIso = Rhino.ExtractIsoCurve(strShape(0) , arrUVPtParam, 1)
                'divide the curve into the necessary segments for the horizontal diagrid modules
                arrPtIso = Rhino.DivideCurve(strIso(0), intG)
                'delete isocurves at each digrid module level
                Rhino.DeleteObjects(strIso)

                'store all values on the curve in an array to access later for coordinate system
                For j = 0 To UBound(arrPtIso)
                        arrDivPts(i,j) = arrPtIso(j)
                Next
        Next
        Rhino.CurrentLayer "Default"

        'return the array of surface coordinate points
        segmentSurface = arrDivPts

End Function


Function addDrawDiagrid(strShape, intV, intGZ, intG, dblR, segmentSurface)
        Dim k,l
        Dim arrLn(300,300)
        Dim strLn1, strLn2, strLn3, strLn4 'strPanel, strPanel2

        ' DIAGRID draw all triangle panels using points created
        'un-comment Panel lines to apply a surface to each triangle formed
        For k = 0 To intV
    For l = 0 To (intG-2) Step 2
                        If k Mod 2 = 0 And (k <= (intV-1)) Then
                                If (l >= (intG-3)) And (k < (intV-1)) Then

                                        strLn1 = Rhino.AddLine (segmentSurface(k,l),segmentSurface(k+1,l+1))
                                        strLn2 = Rhino.AddLine (segmentSurface(k+1,l+1),segmentSurface(k,0))
                                        strLn3 = Rhino.AddLine (segmentSurface(k,l), segmentSurface(k,0))
                                Else
                                        strLn1 = Rhino.AddLine (segmentSurface(k,l),segmentSurface(k+1,l+1))
                                        strLn2=Rhino.AddLine (segmentSurface(k+1,l+1),segmentSurface(k,l+2))
                                        strLn3 = Rhino.AddLine (segmentSurface(k,l),segmentSurface(k,l+2))

                                End If
```

```
                        ElseIf (k <= intV-1) And (k Mod 2 = 1)Then
                                If (l >= (intG-3)) And (k < (intV)) Then
                                        strLn1 = Rhino.AddLine(segmentSurface(k+1,l),segmentSurface(k,l+1))
                                        strLn2 = Rhino.AddLine(segmentSurface(k,l+1),segmentSurface(k+1,0))
                                        strLn3 = Rhino.AddLine(segmentSurface(k,l+1), segmentSurface(k,l))

                                Else
                                        strLn1 = Rhino.AddLine(segmentSurface(k+1,l), segmentSurface(k,l+1))
                                        strLn2= Rhino.AddLine(segmentSurface(k,l+1),segmentSurface(k+1,l+2))
                                        strLn3 = Rhino.AddLine(segmentSurface(k,l+1), segmentSurface(k,l+3))

                                End If

                                'top module, odd number of modules
                        ElseIf (k = (intV)) And (intV Mod 2 = 1) Then
                                print "k: doing this!"
                                If (l < (intG-3)) Then
                                strLn4 = Rhino.AddLine (segmentSurface(k,l+1), segmentSurface(k,l+3))

                                Else
                                        strLn4 = Rhino.AddLine (segmentSurface(k,l+1), segmentSurface(k,l))

                                End If

                                'top module, even number of modules
                        ElseIf (k= (intV)) And (intV Mod 2 = 0) Then
                                If (l < (intG-3)) Then
                                        strLn4 = Rhino.AddLine (segmentSurface(k,l),segmentSurface(k,l+2))

                                Else
                                        strLn4 = Rhino.AddLine (segmentSurface(k,l), segmentSurface(k,0))

                                End If

                        End If
                Next
                print "k :" & k
        Next

        'draws the horizontal lines between diagrid modules for top of tower
        Dim m, n
        m=intV
        For n=0 To (intG-2) Step 2
                If (intV Mod 2 = 1) Then
                        If (n < (intG-3)) Then
                                strLn4 = Rhino.AddLine (segmentSurface(m,n+1), segmentSurface(m,n+3))
                        Else
                                strLn4 = Rhino.AddLine (segmentSurface(m,n+1), segmentSurface(m,n))
                        End If
                Else
                        If (n < (intG-3)) Then
                                strLn4 = Rhino.AddLine (segmentSurface(m,n),segmentSurface(m,n+2))
                        Else
                                strLn4 = Rhino.AddLine (segmentSurface(m,n), segmentSurface(m,0))
                        End If
                End If
        Next

End Function


Function addWindVector(strShape, arrSegSrf, intV, intG)
        Dim intWindChoice
        'ask user if he/she wants to perform this function.  otherwise, tower geometry only
        intWindChoice = Rhino.MessageBox("Do you want to designate a wind direction?",4,_
                "Wind Force")
        If intWindChoice = 7 Then Exit Function

        'prompt user for wind angle, North = 90 degrees, South = 270 degrees.
        Rhino.MessageBox "Angle must be between 0 and 360.",4, "Acceptable Wind Angles"
        Dim dblWind : dblWind = Rhino.IntegerBox("90 = North, 270 = South; Input integer value.",_
                90,"Wind Angle Input")

        'convert angle input of unit degrees to radians
        dblWind = Rhino.ToRadians(dblWind)
        'create a vector that defines this wind direction
        Dim arrPoint1, arrPoint2
        arrPoint1 = Array(0,0,0)
        'determine arrPoint2 using properties of Polar coordinates, R=1
        'x=R*cos(theta), y=R*sin(theta)
        arrPoint2 = Array(cos(dblWind),sin(dblWind),0)
```

```
        Dim arrWind : arrWind = Rhino.VectorCreate(arrPoint1, arrPoint2)

        'find the plane that is defined by the normal through the origin
        Dim arrPlane
        arrPlane = Rhino.PlaneFromNormal(arrPoint1, arrWind)
        'move the plane far away from the tower model, BEHIND the wind vector.
        'could be used to project plane onto surface and obtain necessary locations by
        'intersection
        Rhino.MovePlane arrPlane, Array(100*cos(dblWind-(pi/2)), 100*sin(dblWind-(pi/2)),0)

        'define variables necessary for vector operations
        Dim i,j
        Dim arrPoint, arrParam, arrNormal
        Dim arrYes(200,200)
        Dim intDP, intA
        Dim arrNewPt, arrWindNew, dblA
        dblA = 5

        'create new layer and set current to draw vectors, color blue-ish
        Rhino.AddLayer "Wind Vectors",RGB(20,200,168)
        Rhino.CurrentLayer "Wind Vectors"

        'loop to apply wind vector only at valid nodes and not each coordinate block
        For i=1 To intV 'do not put wind load on ground level i=0
                If (i Mod 2 = 0) Then
                        For j = 0 To intG Step 2
                                arrPoint = arrSegSrf(i,j)
                                If IsArray(arrPoint)  Then
                                arrParam = Rhino.SurfaceClosestPoint(strShape(0), arrPoint)
                                arrNormal = Rhino.SurfaceNormal(strShape(0), arrParam)

                                intDP = VectorDotProduct(arrWind, arrNormal)
                                intA = Rhino.ACos(intDP)
                                intA = Rhino.ToDegrees(intA)
                                Rhino.Print intA
                                        If intA < 90 Then
                                        'arrYes(i,j) = 1
                                        arrWindNew = VectorScale(arrWind, dblA*i)
                                        arrNewPt = PointSubtract(arrPoint,arrWindNew)
                                        Rhino.AddLine arrNewPt,arrPoint
                                        Else
                                        arrYes(i,j) = 0
                                        End If
                                Else
                                End If
                        Next
                Else
                        'for odd numbered diagrid module levels
                        For j=1 to intG-1 Step 2
                                arrPoint = arrSegSrf(i,j)
                                If IsArray(arrPoint)  Then
                                        arrParam = Rhino.SurfaceClosestPoint(strShape(0), arrPoint)
                                        arrNormal = Rhino.SurfaceNormal(strShape(0), arrParam)

                                        intDP = VectorDotProduct(arrWind, arrNormal)
                                        intA = Rhino.ACos(intDP)
                                        intA = Rhino.ToDegrees(intA)
                                        Rhino.Print intA
                                        If intA < 90 Then
                                                'arrYes(i,j) = 1
                                                arrWindNew = VectorScale(arrWind, dblA*i)
                                                arrNewPt = PointSubtract(arrPoint,arrWindNew)
                                                Rhino.AddLine arrNewPt,arrPoint
                                        Else
                                                arrYes(i,j) = 0
                                        End If
                                Else
                                End If
                        Next
                End If
        Next
        Rhino.CurrentLayer "Default"

End Function
```

DiagridExport Script

```
Option Explicit
'Script written by <Jessica Sundberg>
'Script copyrighted by <Jessica Sundberg>
'Thesis Title: An Approach to the Design of Free Form Diagrid Structures

Call Main()
Sub Main()

        'creates an array of all frame identifiers
        Dim arrLines : arrLines = Rhino.ObjectsByLayer("Diagrid Lines")
        Rhino.AddLayer "Exported Geometry",RGB(200,0,50)
        'how many frames are there total
        Dim numArray : numArray = UBound(arrLines)


        Dim arrGeoIndex
        arrGeoIndex = fJSort(arrLines, numArray)

        Dim arrWindJs
        arrWindJs = windSort(arrGeoIndex(1), arrGeoIndex(2), arrGeoIndex(3))

        Call excelExport(arrGeoIndex(0), arrGeoIndex(1), numArray, arrGeoIndex(2), arrWindJs)

End Sub

Function fJSort(arrLines, numArray)

        Dim numUniquePts,j, arrEndPt(), arrStartPt()
        Dim arrNodeNum(), arrNew, arrNew2
        Dim numNode,i,y,a,b,q,r,s,m,z
        Dim pt1, pt2
        Dim arrInfo(), arrExport(1000,2)

        'store values for first frame element's two unique nodes
        For i=0 to numArray
                ReDim Preserve arrStartPt(i)
                ReDim Preserve arrEndPt(i)
                If Rhino.IsCurve(arrLines(i)) Then
                        arrStartPt(i)= Rhino.CurveStartPoint(arrLines(i))
                        arrEndPt(i)= Rhino.CurveEndPoint(arrLines(i))
                Else
                End If
        Next

        ReDim Preserve arrNodeNum(0)
        arrNodeNum(0) = arrStartPt(0)
        ReDim Preserve arrNodeNum(1)
        arrNodeNum(1) = arrEndPt(0)
        ReDim Preserve arrInfo(0)
        arrInfo(0) = Array(0,0,1)
        arrExport(1,0) = arrInfo(0)(0)
        arrExport(1,1) = arrInfo(0)(1)
        arrExport(1,2) = arrInfo(0)(2)
        Rhino.ObjectLayer arrLines(0),"Exported Geometry"

        numUniquePts = 2

        For q=1 To numArray'... but using smaller number to troubleshoot
                r=0
                s=0
                pt1=0
                pt2=0
                a=0
                b=0

                For j=0 To numUniquePts-1
                        'ReDim Preserve arrNodeNum(numUniquePts)

                        Dim arrDiff
                        'Rhino.Print Rhino.Pt2Str(arrStartPt(q))
                        'Rhino.Print Rhino.Pt2Str(arrNodeNum(j))
                        arrDiff = Rhino.VectorSubtract(arrStartPt(q),arrNodeNum(j))
                        If Rhino.IsVectorZero(arrDiff) Then
                                r=r+1
                                a=j
                        End If

                        Dim arrDiff2
```

```
                arrDiff2 = Rhino.VectorSubtract(arrEndPt(q),arrNodeNum(j))
                'Rhino.Print Rhino.Pt2Str(arrDiff2)
                If Rhino.IsVectorZero(arrDiff2) Then
                        s=s+1
                        b=j
                End If
        Next

        Rhino.Print CStr(r)
        Rhino.Print CStr(s)

        If r>0 Then
                pt1=a
        Else
                numUniquePts = numUniquePts + 1 'update x
                ReDim Preserve arrNodeNum(numUniquePts-1)

                arrNew= arrStartPt(q)
                arrNodeNum(numUniquePts-1)= arrNew
                'Rhino.AddSphere arrNodeNum(numUniquePts-1),10

                pt1=(numUniquePts-1)
        End If

        If s>0 Then
                pt2=b
        Else
                numUniquePts = numUniquePts + 1'update y
                ReDim Preserve arrNodeNum(numUniquePts-1)

                arrNew2= arrEndPt(q)
                arrNodeNum(numUniquePts-1)= arrNew2
                'Rhino.AddSphere arrNodeNum(numUniquePts-1),10

                pt2=(numUniquePts-1)
        End If

        'make array of (frame#,start coordinate,end coordinate)
        ReDim Preserve arrInfo(q)
        arrInfo(q) = Array(q,pt1,pt2)
        Rhino.Print "Step"& CStr(q)
        Rhino.Print "x : "& CStr(numUniquePts)
        'End If

        arrExport(q+1,0)= arrInfo(q)(0)
        arrExport(q+1,1)= arrInfo(q)(1)
        arrExport(q+1,2)= arrInfo(q)(2)

        Rhino.ObjectLayer arrLines(q),"Exported Geometry"
Next

Dim arrNodes(2000,3),intNode
For z=0 To numUniquePts-1
        intNode=z
        arrNodes(z+1,0) = intNode
        arrNodes(z+1,1) = arrNodeNum(z)(0)
        arrNodes(z+1,2) = arrNodeNum(z)(1)
        arrNodes(z+1,3) = arrNodeNum(z)(2)
Next

fJSort = Array(arrExport, arrNodes, numUniquePts, arrNodeNum)

End Function

Function excelExport(arrExport, arrNodes, numArray, numUniquePts, arrWindJs)
        Dim intJ : intJ = UBound(arrWindJs)
        'setup to export data
        ' Launch Excel
        Dim app
        Set app = CreateObject("Excel.Application")
        ' Make it visible
        app.Visible = True
        ' Add a new workbook
        Dim wb,ws,ws2
        Set wb = app.workbooks.add

        'make sheet of frame IDs
        Set ws = wb.Worksheets(2)
        ws.Activate
        'Declare a range object to hold our data
        Dim rng
```

```
        Set rng = wb.Activesheet.Range("A1").Resize(numArray+1,3)
        app.Columns(1).ColumnWidth = 10
        app.Columns(2).ColumnWidth = 15
        app.Columns(3).ColumnWidth = 15
        'Place titles on sheet
        rng.value = arrExport
        app.Cells(1, 1).Value = "Frame ID"
        app.Cells(1, 2).Value = "Start Joint ID"
        app.Cells(1, 3).Value = "End Joint ID"
        app.Range("A1:C1").Select
        app.Selection.Font.Bold = True

        'make sheet of joint ID coordinates
        Set ws = wb.Worksheets(1)
        ws.Activate
        Set rng = wb.Activesheet.Range("A1").Resize(numUniquePts+1,4)
        rng.value = arrNodes
        app.Cells(1,1).Value = "Joint ID"
        app.Cells(1,2).Value = "X"
        app.Cells(1,3).Value = "Y"
        app.Cells(1,4).Value = "Z"
        app.Range("A1:D1").Select
        app.Selection.Font.Bold = True

        Set ws = wb.Worksheets(3)
        ws.Activate
        Set rng = wb.Activesheet.Range("A2").Resize(intJ+1,5)
        app.Columns(1).ColumnWidth = 8
        app.Columns(2).ColumnWidth = 15
        app.Columns(3).ColumnWidth = 8
        app.Columns(4).ColumnWidth = 8
        app.Columns(5).ColumnWidth = 8

        rng.value = arrWindJs
        app.Cells(1,1).Value = "Joint ID"
        app.Cells(1,2).Value = "Force magnitude"
        app.Cells(1,3).Value = "Fx"
        app.Cells(1,4).Value = "Fy"
        app.Cells(1,5).Value = "Fz"
        app.Range("A1:E1").Select
        app.Selection.Font.Bold = True

        'Give the user control of Excel
        app.UserControl = True

End Function

Function windSort(arrNodes, numUniquePts, arrNodeNum)

        'define all wind vectors as array
        Dim arrVectors : arrVectors = Rhino.ObjectsByLayer("Wind Vectors")
        'calculate number of wind vectors
        Dim numWind : numWind = UBound(arrVectors)

        Dim arrStartVec(), arrEndVec(), arrWindJoints(1000,4)
        Dim k,i,j,zco
        Dim same, index, num, arrFmag(1000), arrForce(), mag

        For k=0 To numWind
                'define endpoint of vector
                ReDim Preserve arrStartVec(k)
                ReDim Preserve arrEndVec(k) 'endpoint at joint
                If Rhino.IsCurve(arrVectors(k)) Then
                        arrStartVec(k)= Rhino.CurveStartPoint(arrVectors(k))
                        arrEndVec(k)= Rhino.CurveEndPoint(arrVectors(k))
                End If
        Next

        'extract angle from vector
        Dim arrZero : arrZero = Array(1,0,0)
        arrZero  = Rhino.VectorUnitize(arrZero)
        Dim arrAngle : arrAngle = Rhino.VectorCreate(arrStartVec(1), arrEndVec(1))
        arrAngle = Rhino.VectorUnitize(arrAngle)
        'calculate angle using the dot product
        Dim dot : dot = Rhino.VectorDotProduct(arrZero, arrAngle)
        ' Force the dot product of the two input vectors to
        ' fall within the domain for inverse cosine
        If (dot < -1.0) Then
                dot = -1.0
        ElseIf (dot > 1.0) Then
                dot = 1.0
```

```
        End If
        Dim dblAngleRad : dblAngleRad = Rhino.ACos(dot)
        Dim dblAngle : dblAngle = Rhino.ToDegrees(Rhino.ACos(dot))
        Rhino.Print CStr(dblAngle)

        'search through all wind vectors
        For i=0 To numWind
                same = 0
                index = 0
                num=0
                arrFmag(i)=0
                zco=0

                'find joint index that matches wind vector location
                For j=0 To numUniquePts-1
                        Dim arrDiffs
                        arrDiffs = Rhino.VectorSubtract(arrEndVec(i),arrNodeNum(j))
                        If Rhino.IsVectorZero(arrDiffs) Then
                                same = same+1
                                index = j
                        End If
                Next

                'store the wind information in terms of global joint indexing
                If same>0 Then
                        num = index
                        'set variable equal to z coordinate of joint
                        zco = arrNodeNum(num)(2)
                        'solution for magnitude as a function of height
                        'in directon of vector, can be changed for more accurate fctn
                        arrFmag(i) = 10*zco
                        ReDim Preserve arrForce(i)
                        arrForce(i) = Array(arrFmag(i)*Cos(dblAngleRad), arrFmag(i)*Sin(dblAngleRad), 0)
                        'store the array
                        'ReDim Preserve arrWindJoints(i)
                        arrWindJoints(i,0)= num
                        arrWindJoints(i,1)= arrFmag(i)
                        arrWindJoints(i,2)= arrForce(i)(0)
                        arrWindJoints(i,3)= arrForce(i)(1)
                        arrWindJoints(i,4)= arrForce(i)(2)
                Else
                End If

        Next

        windSort = arrWindJoints

End Function
```

Draw Hyperbolic Paraboloid Script

```
Option Explicit

Call Main
Sub Main()

  'get user input surface
  Dim strChoice
  Dim strShape
  Dim intStories
  Dim intGrid
  Dim dblRadius
  strChoice = Rhino.GetString("Use your own surface? Y or N", "No",Array("Yes","No"))

  If strChoice = "Yes" Then
    'take user surface, divide by 3.5 and only consider whole floors
    'ask user the number of diagrid modules
    strShape = Rhino.GetObject("Select surface to apply diagrid.", 8)
    'get highest point of surface
    Dim arrBBox
    arrBBox = Rhino.BoundingBox (strShape)
    Dim arrZPoints, arrYPoints
    Dim i,j
    Dim arrBBoxTemp
    Dim numOfBBox : numOfBBox = UBound (arrBBox)
    ReDim arrZPoints(numOfBBox)
    ReDim arrYPoints(numOfBBox)

    For i = 0 To numOfBBox
      arrBBoxTemp = arrBBox(i)
      arrZPoints(i) = arrBBoxTemp(2)
      arrYPoints(i) = arrBBoxTemp(1)
      Rhino.Print arrZPoints(i)
    Next

    Dim intHeight : intHeight = Rhino.Max(arrZPoints)
    dblRadius = Rhino.Max(arrYPoints)
    intStories = ((intHeight - (intHeight Mod 3.5))/3.5)
    intGrid = Rhino.GetInteger("Input the number of diagrid modules", 8)

  Else
    'USER INPUT//NUMBER OF STORIES + DIAGRID modules
    'Dim strShape : strShape = Rhino.GetString("Select tower shape", ...
    intStories = Rhino.GetInteger("Input height of tower in stories",50,,300)
    dblRadius = Rhino.GetInteger("Input radius of the base of the tower", 10,,100)
    intGrid = Rhino.GetInteger("Input the number of diagrid modules", 8)
    Dim intMod : intMod = Rhino.GetInteger("Input the number of floors per diagrid module", 3)

    intGrid = intGrid*2
    'define final surface, to be updated by calling functions
    ''Draw surface of hyperbolic parabaloid
   strShape = addDrawHyperParab(intStories, intGrid, dblRadius)
  End If

  ''Draw diagrid On surface
  Call addDrawDiagrid(strShape, intStories, intGrid, dblRadius)

  'give diagrid members piped dimension
 ' Dim arrCurve
       'arrCurve = Rhino.getObjects("select all curves ",4 )
       'drawTubesForEveryCurve(arrCurve)

End Sub


Private Function addDrawHyperParab(ByVal intS, ByVal intG, ByVal dblR)
  'convert height and define translation vectors
  Dim intH : intH = intS*3.5
  Dim arrXvec : arrXvec = Array(1,0,0)
```

```
  Dim arrYvec : arrYvec = Array(0,1,0)
  Dim arrZvec : arrZvec = Array(0,0,1)

  'BASE CIRCLE  draw circle on origin
  Dim strBase
  Dim arrPlane
  arrPlane = Rhino.WorldXYPlane
  strBase = Rhino.AddCircle (arrPlane, dblR)

  'TOP CIRCLE  creates translation vector to get 'roof'
  Dim vecHeight
  vecHeight = Rhino.VectorScale(arrZvec, intH)
  'copies the circle to top of tower
  Dim strTop : strTop = Rhino.CopyObject(strBase , vecHeight)

  'CENTER CIRCLE  draws center circle
  Dim dblRr : dblRr = Rhino.GetInteger("Input radius of tower in meters at smallest
point",5,,300)
  Dim strWaist : strWaist = Rhino.AddCircle (arrPlane, dblRr)
  'moves circle to waist of tower
  Dim dblH : dblH = Rhino.GetInteger("At what percent of the height (0-100) does the skinniest
point occur?", 50,,100)
  'makes the translation vector for waist
  Dim intScl : intScl = (dblH/100)*intH
  Dim vecWaist : vecWaist = Rhino.VectorScale(arrZvec, intScl)
  strWaist = Rhino.MoveObject (strWaist, vecWaist)

  'SURFACE LOFT  lofts the curves to make surface shape
  Dim arrCvs : arrCvs = Array(strBase, strWaist, strTop)
  Dim arrLoft : arrLoft = Rhino.AddLoftSrf(arrCvs)

  addDrawHyperParab = arrLoft(0)
End Function
```

*.$2k Shell File

```
File C:\Documents and Settings\jsundber\Desktop\sample tower 1.$2k was saved on 1/25/09 at
15:23:07

TABLE:  "ACTIVE DEGREES OF FREEDOM"
   UX=Yes   UY=Yes   UZ=Yes   RX=Yes   RY=Yes   RZ=Yes

TABLE:  "ANALYSIS CASE DEFINITIONS"
   Case=DEAD   Type=LinStatic   InitialCond=Zero   RunCase=Yes
   Case=MODAL   Type=LinModal   InitialCond=Zero   RunCase=Yes
   Case=wind   Type=LinStatic   InitialCond=Zero   RunCase=Yes
   Case=wind-2   Type=LinStatic   InitialCond=Zero   RunCase=Yes
   Case=wind-3   Type=LinStatic   InitialCond=Zero   RunCase=Yes
   Case=wind-4   Type=LinStatic   InitialCond=Zero   RunCase=Yes
   Case=wind-5   Type=LinStatic   InitialCond=Zero   RunCase=Yes
   Case=wind-6   Type=LinStatic   InitialCond=Zero   RunCase=Yes
   Case=wind-7   Type=LinStatic   InitialCond=Zero   RunCase=Yes
   Case=wind-8   Type=LinStatic   InitialCond=Zero   RunCase=Yes
   Case=wind-9   Type=LinStatic   InitialCond=Zero   RunCase=Yes
   Case=wind-10   Type=LinStatic   InitialCond=Zero   RunCase=Yes
   Case=wind-11   Type=LinStatic   InitialCond=Zero   RunCase=Yes
   Case=wind-12   Type=LinStatic   InitialCond=Zero   RunCase=Yes

TABLE:  "ANALYSIS OPTIONS"
   Solver=Advanced   Force32Bit=No   StiffCase=None   GeomMod=No

TABLE:  "AUTO WAVE 3 - WAVE CHARACTERISTICS - GENERAL"
   WaveChar=Default   WaveType="From Theory"   KinFactor=1   SWaterDepth=45   WaveHeight=18
WavePeriod=12   WaveTheory=Linear

TABLE:  "AUTO WIND - ASCE7-05"
   LoadCase=wind   ExposeFrom=Diaphragms   Angle=0   WindwardCp=0.8   LeewardCp=0.5   ASCECase=1
E1=0   E2=0   UserZ=No   WindSpeed=70   Exposure=B   I=1   Kzt=1   GustFactor=0.85   Kd=0.85
ExpWidth="From diaphs"
   LoadCase=wind-2   ExposeFrom=Diaphragms   Angle=90   WindwardCp=0.8   LeewardCp=0.5
ASCECase=1   E1=0   E2=0   UserZ=No   WindSpeed=70   Exposure=B   I=1   Kzt=1   GustFactor=0.85
Kd=0.85   ExpWidth="From diaphs"
   LoadCase=wind-3   ExposeFrom=Diaphragms   Angle=0   WindwardCp=0.8   LeewardCp=0.5
ASCECase=2   E1=0.15   E2=0   UserZ=No   WindSpeed=70   Exposure=B   I=1   Kzt=1
GustFactor=0.85   Kd=0.85   ExpWidth="From diaphs"
   LoadCase=wind-4   ExposeFrom=Diaphragms   Angle=0   WindwardCp=0.8   LeewardCp=0.5
ASCECase=2   E1=-0.15   E2=0   UserZ=No   WindSpeed=70   Exposure=B   I=1   Kzt=1
GustFactor=0.85   Kd=0.85   ExpWidth="From diaphs"
   LoadCase=wind-5   ExposeFrom=Diaphragms   Angle=90   WindwardCp=0.8   LeewardCp=0.5
ASCECase=2   E1=0.15   E2=0   UserZ=No   WindSpeed=70   Exposure=B   I=1   Kzt=1
GustFactor=0.85   Kd=0.85   ExpWidth="From diaphs"
   LoadCase=wind-6   ExposeFrom=Diaphragms   Angle=90   WindwardCp=0.8   LeewardCp=0.5
ASCECase=2   E1=-0.15   E2=0   UserZ=No   WindSpeed=70   Exposure=B   I=1   Kzt=1
GustFactor=0.85   Kd=0.85   ExpWidth="From diaphs"
   LoadCase=wind-7   ExposeFrom=Diaphragms   Angle=0   WindwardCp=0.8   LeewardCp=0.5
ASCECase=3   E1=0   E2=0   UserZ=No   WindSpeed=70   Exposure=B   I=1   Kzt=1   GustFactor=0.85
Kd=0.85   ExpWidth="From diaphs"
   LoadCase=wind-8   ExposeFrom=Diaphragms   Angle=90   WindwardCp=0.8   LeewardCp=0.5
ASCECase=3   E1=0   E2=0   UserZ=No   WindSpeed=70   Exposure=B   I=1   Kzt=1   GustFactor=0.85
Kd=0.85   ExpWidth="From diaphs"
   LoadCase=wind-9   ExposeFrom=Diaphragms   Angle=0   WindwardCp=0.8   LeewardCp=0.5
ASCECase=4   E1=0.15   E2=0.15   UserZ=No   WindSpeed=70   Exposure=B   I=1   Kzt=1
GustFactor=0.85   Kd=0.85   ExpWidth="From diaphs"
   LoadCase=wind-10   ExposeFrom=Diaphragms   Angle=0   WindwardCp=0.8   LeewardCp=0.5
ASCECase=4   E1=-0.15   E2=-0.15   UserZ=No   WindSpeed=70   Exposure=B   I=1   Kzt=1
GustFactor=0.85   Kd=0.85   ExpWidth="From diaphs"
   LoadCase=wind-11   ExposeFrom=Diaphragms   Angle=90   WindwardCp=0.8   LeewardCp=0.5
ASCECase=4   E1=0.15   E2=0.15   UserZ=No   WindSpeed=70   Exposure=B   I=1   Kzt=1
GustFactor=0.85   Kd=0.85   ExpWidth="From diaphs"
   LoadCase=wind-12   ExposeFrom=Diaphragms   Angle=90   WindwardCp=0.8   LeewardCp=0.5
ASCECase=4   E1=-0.15   E2=-0.15   UserZ=No   WindSpeed=70   Exposure=B   I=1   Kzt=1
GustFactor=0.85   Kd=0.85   ExpWidth="From diaphs"

TABLE:  "AUTO WIND EXPOSURE FOR HORIZONTAL DIAPHRAGMS"
```

```
   LoadCase=wind
   LoadCase=wind-2
   LoadCase=wind-3
   LoadCase=wind-4
   LoadCase=wind-5
   LoadCase=wind-6
   LoadCase=wind-7
   LoadCase=wind-8
   LoadCase=wind-9
   LoadCase=wind-10
   LoadCase=wind-11
   LoadCase=wind-12

TABLE:  "CASE - MODAL 1 - GENERAL"
   Case=MODAL    ModeType=Eigen    MaxNumModes=12    MinNumModes=1    EigenShift=0    EigenCutoff=0
EigenTol=0.000000001   AutoShift=Yes

TABLE:  "CASE - STATIC 1 - LOAD ASSIGNMENTS"
   Case=DEAD   LoadType="Load case"   LoadName=DEAD   LoadSF=1
   Case=wind   LoadType="Load case"   LoadName=wind   LoadSF=1
   Case=wind-2   LoadType="Load case"   LoadName=wind-2   LoadSF=1
   Case=wind-3   LoadType="Load case"   LoadName=wind-3   LoadSF=1
   Case=wind-4   LoadType="Load case"   LoadName=wind-4   LoadSF=1
   Case=wind-5   LoadType="Load case"   LoadName=wind-5   LoadSF=1
   Case=wind-6   LoadType="Load case"   LoadName=wind-6   LoadSF=1
   Case=wind-7   LoadType="Load case"   LoadName=wind-7   LoadSF=1
   Case=wind-8   LoadType="Load case"   LoadName=wind-8   LoadSF=1
   Case=wind-9   LoadType="Load case"   LoadName=wind-9   LoadSF=1
   Case=wind-10   LoadType="Load case"   LoadName=wind-10   LoadSF=1
   Case=wind-11   LoadType="Load case"   LoadName=wind-11   LoadSF=1
   Case=wind-12   LoadType="Load case"   LoadName=wind-12   LoadSF=1

TABLE:  "CONNECTIVITY - FRAME"
   Frame=1   JointI=1   JointJ=2   IsCurved=No
   Frame=2   JointI=3   JointJ=4   IsCurved=No
      ...


TABLE:  "COORDINATE SYSTEMS"
   Name=GLOBAL   Type=Cartesian   X=0   Y=0   Z=0   AboutZ=0   AboutY=0   AboutX=0

TABLE:  "DATABASE FORMAT TYPES"
   UnitsCurr=Yes   OverrideE=No

TABLE:  "FRAME AUTO MESH ASSIGNMENTS"
   Frame=1      AutoMesh=Yes      AtJoints=Yes      AtFrames=No      NumSegments=0      MaxLength=0
MaxDegrees=0
   Frame=2      AutoMesh=Yes      AtJoints=Yes      AtFrames=No      NumSegments=0      MaxLength=0
MaxDegrees=0
      ...


TABLE:  "FRAME DESIGN PROCEDURES"
   Frame=1   DesignProc="From Material"
   Frame=2   DesignProc="From Material"
      ...


TABLE:  "FRAME OUTPUT STATION ASSIGNMENTS"
   Frame=1   StationType=MinNumSta   MinNumSta=3   AddAtElmInt=Yes   AddAtPtLoad=Yes
   Frame=2   StationType=MinNumSta   MinNumSta=3   AddAtElmInt=Yes   AddAtPtLoad=Yes
      ...


TABLE:  "FRAME SECTION ASSIGNMENTS"
   Frame=1   AutoSelect=N.A.   AnalSect=FSEC1   MatProp=Default
   Frame=2   AutoSelect=N.A.   AnalSect=FSEC1   MatProp=Default
      ...


TABLE:  "FRAME SECTION PROPERTIES 01 - GENERAL"
```

```
   SectionName=FSEC1    Material=A992Fy50    Shape=Box/Tube    t3=0.1524    t2=0.1016    tf=0.00635
tw=0.00635       Area=0.00306451       TorsConst=1.01854328602056E-05       I33=9.77059914155833E-06
I22=5.14002453175833E-06   AS2=0.00193548
        AS3=0.00129032              S33=1.28223085847222E-04              S22=1.01181585270833E-04
Z33=0.00015618920375   Z22=0.00011726992675   R33=5.64650967334938E-02   R22=4.09545420868536E-02
Color=White   FromFile=No   AMod=1   A2Mod=1   A3Mod=1
        JMod=1   I2Mod=1   I3Mod=1   MMod=1   WMod=1   Notes="Added 1/25/2009 3:12:53 PM"

TABLE:  "FUNCTION - POWER SPECTRAL DENSITY - USER"
   Name=UNIFPSD   Frequency=0   Value=1
   Name=UNIFPSD   Frequency=1   Value=1

TABLE:  "FUNCTION - RESPONSE SPECTRUM - USER"
   Name=UNIFRS   Period=0   Accel=1   FuncDamp=0.05
   Name=UNIFRS   Period=1   Accel=1

TABLE:  "FUNCTION - STEADY STATE - USER"
   Name=UNIFSS   Frequency=0   Value=1
   Name=UNIFSS   Frequency=1   Value=1

TABLE:  "FUNCTION - TIME HISTORY - USER"
   Name=RAMPTH   Time=0   Value=0
   Name=RAMPTH   Time=1   Value=1
   Name=RAMPTH   Time=4   Value=1
   Name=UNIFTH   Time=0   Value=1
   Name=UNIFTH   Time=1   Value=1

TABLE:  "GRID LINES"
   CoordSys=GLOBAL      AxisDir=X      XRYZCoord=0      LineType=Primary      LineColor=Gray8Dark
Visible=Yes   BubbleLoc=End   AllVisible=Yes   BubbleSize=2.4384
   CoordSys=GLOBAL      AxisDir=Y      XRYZCoord=0      LineType=Primary      LineColor=Gray8Dark
Visible=Yes   BubbleLoc=End
   CoordSys=GLOBAL      AxisDir=Z      XRYZCoord=0      LineType=Primary      LineColor=Gray8Dark
Visible=Yes   BubbleLoc=End

TABLE:  "GROUPS 1 - DEFINITIONS"
   GroupName=ALL    Selection=Yes    SectionCut=Yes    Steel=Yes    Concrete=Yes    Aluminum=Yes
ColdFormed=Yes      Stage=Yes      Bridge=Yes      AutoSeismic=No      AutoWind=No      SelDesSteel=No
SelDesAlum=No   SelDesCold=No   MassWeight=Yes   Color=Red
   GroupName=DXFIN    Selection=Yes    SectionCut=Yes    Steel=Yes    Concrete=Yes    Aluminum=Yes
ColdFormed=Yes      Stage=Yes      Bridge=Yes      AutoSeismic=No      AutoWind=No      SelDesSteel=No
SelDesAlum=No   SelDesCold=No   MassWeight=Yes   Color=Black
   GroupName=DXFIN-1    Selection=Yes    SectionCut=Yes    Steel=Yes    Concrete=Yes    Aluminum=Yes
ColdFormed=Yes      Stage=Yes      Bridge=Yes      AutoSeismic=No      AutoWind=No      SelDesSteel=No
SelDesAlum=No   SelDesCold=No   MassWeight=Yes   Color=Black

TABLE:  "GROUPS 2 - ASSIGNMENTS"
   GroupName=DXFIN-1   ObjectType=Frame   ObjectLabel=1
   GroupName=DXFIN-1   ObjectType=Frame   ObjectLabel=2
       ...


TABLE:  "JOINT COORDINATES"
   Joint=1   CoordSys=GLOBAL   CoordType=Cartesian   XorR=9.33986640718973   Y=2.49728729643643
Z=17.4892373221086   SpecialJt=No
   Joint=2   CoordSys=GLOBAL   CoordType=Cartesian   XorR=7.50973957815568   Y=4.32741422762975
Z=13.9892373221086   SpecialJt=No
   Joint=3   CoordSys=GLOBAL   CoordType=Cartesian   XorR=5.00973936929888   Y=4.99728732929109
Z=17.4892373221086   SpecialJt=No
       ...


TABLE:  "JOINT LOADS - FORCE"
   Joint=1   LoadCase=wind-2   CoordSys=GLOBAL   F1=10   F2=0   F3=0   M1=0   M2=0   M3=0
   Joint=2   LoadCase=wind-2   CoordSys=GLOBAL   F1=10   F2=0   F3=0   M1=0   M2=0   M3=0
       ...


TABLE:  "JOINT PATTERN DEFINITIONS"
   Pattern=DEFAULT
```

```
TABLE:  "JOINT RESTRAINT ASSIGNMENTS"
   Joint=118   U1=Yes   U2=Yes   U3=Yes   R1=Yes   R2=Yes   R3=Yes
   Joint=120   U1=Yes   U2=Yes   U3=Yes   R1=Yes   R2=Yes   R3=Yes
   Joint=122   U1=Yes   U2=Yes   U3=Yes   R1=Yes   R2=Yes   R3=Yes
   Joint=124   U1=Yes   U2=Yes   U3=Yes   R1=Yes   R2=Yes   R3=Yes
   Joint=126   U1=Yes   U2=Yes   U3=Yes   R1=Yes   R2=Yes   R3=Yes
   Joint=128   U1=Yes   U2=Yes   U3=Yes   R1=Yes   R2=Yes   R3=Yes

TABLE:  "LOAD CASE DEFINITIONS"
   LoadCase=DEAD   DesignType=DEAD   SelfWtMult=1
   LoadCase=wind   DesignType=WIND   SelfWtMult=0   AutoLoad=ASCE7-05
   LoadCase=wind-2   DesignType=WIND   SelfWtMult=0   AutoLoad=ASCE7-05
   LoadCase=wind-3   DesignType=WIND   SelfWtMult=0   AutoLoad=ASCE7-05
   LoadCase=wind-4   DesignType=WIND   SelfWtMult=0   AutoLoad=ASCE7-05
   LoadCase=wind-5   DesignType=WIND   SelfWtMult=0   AutoLoad=ASCE7-05
   LoadCase=wind-6   DesignType=WIND   SelfWtMult=0   AutoLoad=ASCE7-05
   LoadCase=wind-7   DesignType=WIND   SelfWtMult=0   AutoLoad=ASCE7-05
   LoadCase=wind-8   DesignType=WIND   SelfWtMult=0   AutoLoad=ASCE7-05
   LoadCase=wind-9   DesignType=WIND   SelfWtMult=0   AutoLoad=ASCE7-05
   LoadCase=wind-10   DesignType=WIND   SelfWtMult=0   AutoLoad=ASCE7-05
   LoadCase=wind-11   DesignType=WIND   SelfWtMult=0   AutoLoad=ASCE7-05
   LoadCase=wind-12   DesignType=WIND   SelfWtMult=0   AutoLoad=ASCE7-05

TABLE:  "MASSES 1 - MASS SOURCE"
   MassFrom=Elements

TABLE:  "MATERIAL PROPERTIES 01 - GENERAL"
   Material=4000Psi      Type=Concrete      SymType=Isotropic      TempDepend=No      Color=White
Notes="Normalweight f'c = 4 ksi added 1/25/2009 3:03:56 PM"
   Material=A992Fy50      Type=Steel      SymType=Isotropic      TempDepend=No      Color=Yellow
Notes="ASTM A992 Fy=50 ksi added 1/25/2009 3:03:56 PM"

TABLE:  "MATERIAL PROPERTIES 02 - BASIC MECHANICAL PROPERTIES"
   Material=4000Psi      UnitWeight=23.563121614979      UnitMass=2.40276960611018
E1=24855578.2847654   G12=10356490.9519856   U12=0.2   A1=0.0000099
   Material=A992Fy50      UnitWeight=76.9728639422648      UnitMass=7.84904737995992
E1=199947978.795958   G12=76903068.767676   U12=0.3   A1=0.0000117

TABLE:  "MATERIAL PROPERTIES 03A - STEEL DATA"
   Material=A992Fy50      Fy=344737.894475789      Fu=448159.262818526      EffFy=379211.683923368
EffFu=492975.189100378      SSCurveOpt=Simple      SSHysType=Kinematic      SHard=0.015      SMax=0.11
SRup=0.17

TABLE:  "MATERIAL PROPERTIES 03B - CONCRETE DATA"
   Material=4000Psi   Fc=27579.0315580631   LtWtConc=No   SSCurveOpt=Simple   SSHysType=Kinematic
SFc=0.002   SCap=0.005   FAngle=0   DAngle=0

TABLE:  "MATERIAL PROPERTIES 06 - DAMPING PARAMETERS"
   Material=4000Psi   ModalRatio=0   VisMass=0   VisStiff=0   HysMass=0   HysStiff=0
   Material=A992Fy50   ModalRatio=0   VisMass=0   VisStiff=0   HysMass=0   HysStiff=0

TABLE:  "OPTIONS - COLORS - DISPLAY"
   DeviceType=Screen   Points=Yellow   LinesFrame=Yellow   LinesFrmExt=Yellow   LinesCable=Green
LinesTendon=Green      SpringLinks=Green      Restraints=Green      Releases=Green      Axes=Cyan
Text=Green   ShadowLines=Gray8Dark _
      GuideLines=Gray8Dark      Highlight=Red      Selection=White      AreaFillBot=Red
AreaFillTop=16744703      AreaFillSd=Red      AreaEdge=DarkRed      SolidF1=Red      SolidF2=Blue
SolidF3=Green   SolidF4=Yellow   SolidF5=White   SolidF6=Cyan _
      SolidEdge=DarkRed   Floor=Gray4   Background=Black   BGLowLeft=Black   BGLowRight=Black
BGUpRight=Black   Darkness=0.5
   DeviceType=Printer      Points=Gray8Dark      LinesFrame=Black      LinesFrmExt=Gray4
LinesCable=Black      LinesTendon=Black      SpringLinks=Gray8Dark      Restraints=Gray8Dark
Releases=Gray4   Axes=Black   Text=Black   ShadowLines=Gray4 _
      GuideLines=Gray4      Highlight=Black      Selection=Black      AreaFillBot=Gray4
AreaFillTop=Gray8Dark   AreaFillSd=Gray4   AreaEdge=Black   SolidF1=Gray1Light   SolidF2=Gray2
SolidF3=Gray3   SolidF4=Gray4   SolidF5=Gray5 _
      SolidF6=Gray6   SolidEdge=Black   Floor=Gray4   Background=White   BGLowLeft=White
BGLowRight=White   BGUpRight=White   Darkness=0.5
   DeviceType="Color Printer"      Points=Black      LinesFrame=7303023      LinesFrmExt=White
LinesCable=Green   LinesTendon=Green   SpringLinks=Green   Restraints=9408399   Releases=Green
Axes=Cyan   Text=Black   ShadowLines=Gray8Dark _
```

```
        GuideLines=10461087      Highlight=Red      Selection=10504778      AreaFillBot=16634568
AreaFillTop=14277119      AreaFillSd=16634568      AreaEdge=7303023      SolidF1=10122991
SolidF2=16756912   SolidF3=11599795   SolidF4=12713983 _
        SolidF5=White   SolidF6=16777128   SolidEdge=7303023   Floor=10461087   Background=White
BGLowLeft=White   BGLowRight=14671839   BGUpRight=White   Darkness=0.5

TABLE:  "OPTIONS - COLORS - OUTPUT"
   DeviceType=Screen     Contour1=13107400     Contour2=6553828     Contour3=Red     Contour4=16639
Contour5=Orange      Contour6=43775      Contour7=54527      Contour8=Yellow      Contour9=65408
Contour10=Green   Contour11=8453888   Contour12=Cyan _
        Contour13=16755200     Contour14=16733440     Contour15=Blue     Transpare=0.5     Ratio1=Cyan
Ratio2=Green     Ratio3=Yellow     Ratio4=Orange     Ratio5=Red     RatioNotD=Gray4     RatioNotC=Red
RatioVal1=0.5   RatioVal2=0.7   RatioVal3=0.9 _
        RatioVal4=1   DFillPos=Yellow   DFillNeg=Red   DFillRPos=Blue   DFillRNeg=Cyan
   DeviceType=Printer    Contour1=Black    Contour2=3158064    Contour3=4210752    Contour4=5263440
Contour5=6316128     Contour6=7368816     Contour7=Gray8Dark     Contour8=Gray7     Contour9=Gray6
Contour10=Gray5   Contour11=Gray4 _
        Contour12=Gray3    Contour13=Gray2    Contour14=Gray1Light    Contour15=White    Transpare=0
Ratio1=Gray2    Ratio2=Gray4    Ratio3=Gray8Dark    Ratio4=4210752    Ratio5=Black    RatioNotD=Gray4
RatioNotC=Black   RatioVal1=0.5 _
        RatioVal2=0.7     RatioVal3=0.9     RatioVal4=1     DFillPos=Gray8Dark     DFillNeg=Gray8Dark
DFillRPos=4210752   DFillRNeg=4210752
   DeviceType="Color  Printer"       Contour1=13107400       Contour2=6553828       Contour3=Red
Contour4=16639      Contour5=Orange      Contour6=43775      Contour7=54527      Contour8=Yellow
Contour9=65408   Contour10=Green   Contour11=8453888 _
        Contour12=Cyan   Contour13=16755200   Contour14=16733440   Contour15=Blue   Transpare=0.5
Ratio1=Cyan     Ratio2=Green     Ratio3=Yellow     Ratio4=Orange     Ratio5=Red     RatioNotD=Gray4
RatioNotC=Red   RatioVal1=0.5   RatioVal2=0.7 _
        RatioVal3=0.9      RatioVal4=1      DFillPos=Red      DFillNeg=Red      DFillRPos=Blue
DFillRNeg=Blue

TABLE:  "PREFERENCES - ALUMINUM DESIGN - AA-ASD 2000"
   THDesign=Envelopes         FrameType="Moment   Frame"         SRatioLimit=1         MaxIter=1
LatFact=1.33333333333333   UseLatFact=No   Bridge=No

TABLE:  "PREFERENCES - COLD FORMED DESIGN - AISI-ASD96"
   THDesign=Envelopes    FrameType="Braced Frame"    SRatioLimit=1    MaxIter=1    OmegaBS=1.67
OmegaBUS=1.67   OmegaBLTB=1.67   OmegaVS=1.67   OmegaVNS=1.5   OmegaT=1.67   OmegaC=1.8

TABLE:  "PREFERENCES - CONCRETE DESIGN - ACI 318-05/IBC2003"
   THDesign=Envelopes   NumCurves=24   NumPoints=11   MinEccen=Yes   PatLLF=0.75   UFLimit=0.95
SeisCat=D      PhiT=0.9      PhiCTied=0.65      PhiCSpiral=0.7      PhiV=0.75      PhiVSeismic=0.6
PhiVJoint=0.85

TABLE:  "PREFERENCES - DIMENSIONAL"
   MergeTol=0.001     FineGrid=0.25     Nudge=0.25     SelectTol=3     SnapTol=12     SLineThick=1
PLineThick=4   MaxFont=8   MinFont=3   AutoZoom=10   ShrinkFact=70   TextFileLen=240

TABLE:  "PREFERENCES - STEEL DESIGN - AISC-LRFD93"
   THDesign=Envelopes    FrameType="Moment Frame"    PatLLF=0.75    SRatioLimit=0.95    MaxIter=1
PhiB=0.9      PhiC=0.85      PhiT=0.9      PhiV=0.9      PhiCA=0.9      CheckDefl=No      DLRat=120
SDLAndLLRat=120   LLRat=360   TotalRat=240   NetRat=240

TABLE:  "PROGRAM CONTROL"
   ProgramName=SAP2000      Version=11.0.8      CurrUnits="KN,  m,  C"      SteelCode=AISC-LRFD93
ConcCode="ACI 318-05/IBC2003"   AlumCode="AA-ASD 2000"   ColdCode=AISI-ASD96   RegenHinge=Yes

TABLE:  "PROJECT INFORMATION"
   Item="Company Name"
   Item="Client Name"
   Item="Project Name"
   Item="Project Number"
   Item="Model Name"
   Item="Model Description"
   Item="Revision Number"
   Item="Frame Type"
   Item=Engineer
   Item=Checker
   Item=Supervisor
   Item="Issue Code"
   Item="Design Code"
```

```
TABLE:  "REBAR SIZES"
   RebarID=#2    Area=0.000032258    Diameter=0.00635
   RebarID=#3    Area=7.09675996154547E-05    Diameter=0.009525
   RebarID=#4    Area=1.29032001922727E-04    Diameter=0.0127
   RebarID=#5    Area=1.99999601538181E-04    Diameter=0.015875
   RebarID=#6    Area=2.83870398461819E-04    Diameter=0.01905
   RebarID=#7    Area=3.87096015381813E-04    Diameter=0.022225
   RebarID=#8    Area=5.09676413843632E-04    Diameter=0.0254
   RebarID=#9    Area=0.00064516    Diameter=2.86512005329132E-02
   RebarID=#10    Area=8.1935318769455E-04    Diameter=3.22579995155334E-02
   RebarID=#11    Area=1.00644956308365E-03    Diameter=3.58139991521835E-02
   RebarID=#14    Area=0.00145161    Diameter=4.30021989583969E-02
   RebarID=#18    Area=0.00258064    Diameter=5.73277992248535E-02
   RebarID=10M    Area=1.00000004162606E-04    Diameter=1.13000003604438E-02
   RebarID=15M    Area=2.00000008325212E-04    Diameter=1.60000002402959E-02
   RebarID=20M    Area=3.00000012487818E-04    Diameter=1.95000002928606E-02
   RebarID=25M    Area=5.00000020813031E-04    Diameter=2.52000011414055E-02
   RebarID=30M    Area=7.00000029138243E-04    Diameter=2.99000000675832E-02
   RebarID=35M    Area=1.00000004162606E-03    Diameter=3.57000012990997E-02
   RebarID=45M    Area=1.50000006243909E-03    Diameter=4.37000014192476E-02
   RebarID=55M    Area=2.50000010406515E-03    Diameter=0.056400002372922
   RebarID=6d    Area=2.83000004150781E-05    Diameter=6.00000009011096E-03
   RebarID=8d    Area=5.03000013308514E-05    Diameter=8.00000012014795E-03
   RebarID=10d    Area=7.85000032676458E-05    Diameter=1.00000001501849E-02
   RebarID=12d    Area=1.13000004703745E-04    Diameter=1.20000001802219E-02
   RebarID=14d    Area=1.54000006410413E-04    Diameter=1.40000002102589E-02
   RebarID=16d    Area=2.01000008366838E-04    Diameter=1.60000002402959E-02
   RebarID=20d    Area=3.14000013070583E-04    Diameter=2.00000003003699E-02
   RebarID=25d    Area=4.91000020438396E-04    Diameter=2.50000003754623E-02
   RebarID=26d    Area=5.31000022103439E-04    Diameter=2.60000003904808E-02
   RebarID=28d    Area=6.16000025641654E-04    Diameter=2.80000004205178E-02

TABLE:  "SOLID PROPERTY DEFINITIONS"
   SolidProp=SOLID1    Material=4000Psi    MatAngleA=0    MatAngleB=0    MatAngleC=0    InComp=Yes
Color=Magenta    Notes="Added 1/25/2009 3:04:03 PM"

END TABLE DATA
```