Design Procedures: A Computational Framework for Parametric Design and Complex Shapes in Architecture

by

Carlos Roberto Barrios Hernandez

Master of Architecture
School of Architecture
Pratt Institute, 1997

Bachelor of Architecture
Facultad de Arquitectura y Arte
Universidad de Los Andes
Merida,Venezuela,1993

Submitted to the Department of Architecture in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Architecture: Design and Computation at the Massachusetts Institute of Technology

June 2006

Signature of the Author [illegible]
Department of Architecture
May 1, 2006

Certified by: [illegible]
William J. Mitchell
Professor of Architecture and Media Arts and Sciences
Thesis Supervisor

Accepted by: [illegible]
Yung Ho Chang
Professor of Architecture
Chair, Committee on Graduate Students
Department of Architecture

Dissertation Committee

William J. Mitchell, Chair
Professor of Architecture and Media Arts and Sciences
Department of Architecture
Massachusetts Institute of Technology

Terry W. Knight, PhD
Professor of Design and Computation
Department of Architecture
Massachusetts Institute of Technology

Larry Sass, PhD
Cecil and Ida Green Career Development Professor
Department of Architecture
Massachusetts Institute of Technology

James Gips, PhD
Professor of Computer Science
Department of Computer Science
Boston College

Abstract

Design Procedures: A computational framework for Parametric Design and Complex Shapes in Architecture

By *Carlos Roberto Barrios Hernandez*

Submitted to the Department of Architecture on May 1, 2006, in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Architecture: Design and Computation

Through the use of computational generative procedures in the early stages of design, it is common to generate shapes of complex nature that could only be produced by the combined forces of human imagination and computer power. However the more complex the shapes are, the more difficult it becomes to establish a discourse that embodies the geometrical and spatial properties, as well as the formal attributes of a given shape. Furthermore, it has become problematic to differentiate between one complex shape and another, resulting in some abstract, cumbersome, and sometimes obscure explanation about how the shape came into being. In some cases, designers recur to complex expressions of mathematical nature that, even though they are precise descriptions of the form, do not offer any clear way to refer to them unless a person is trained in the language of mathematics.

Design Procedures proposes a way of looking at designs as a procedural enterprise where complex shapes are the result of computational process in a step by step basis. Design Procedures in combination with appropriate descriptions of spatial attributes, can offer some light in the dialog of irregular non-Euclidean forms and their properties. This thesis presents the application of Design Procedures to three case studies: 1) The generation of the columns of the Sagrada Familia; 2) The description of non regular shapes in the rod symmetry groups, in particular of double twisted geometries; and 3) The application to a computer program for the generation of non-Euclidean complex shapes for high-rise buildings.

Thesis Supervisor: William J. Mitchell, Professor of Architecture and Media Arts and Sciences.

Biography

Carlos Roberto was born in Caracas and educated as an Architect and Engineer in Universidad de los Andes in Merida, and received his Master of Architecture from Pratt Institute in New York. Before coming to MIT he practiced architecture in Caracas and New York City and taught Design in the School of Architecture of Universidad de los Andes where he held a position as Associate Professor.

Carlos is an outdoor sports enthusiast and a flight fanatic. He is a hang-glider pilot and paraglider instructor. Occasionally he also practices mountain climbing, scuba diving, and flights sailplanes.

Dedication

To my dear wife Rosita who believed in me

To my sons David and little Ricardo, for giving me more joy, delight, happiness, and pleasure, than free time to complete this work.

To my mother Carmen.

To my friends.

To all who thought that I was wasting my time here at MIT.

Acknowledgements

Bill Mitchell, who gave me enough freedom to find my own path

Larry Sass, friend and mentor who showed me to strive for excellence and serve others beyond the call of duty

Terry Knight, for teaching me the joy of learning

Cynthia Wilkes, for your diligence and patience

George Stiny for showing me that computation can be as serious as fun

Jim Gips for helping me clarify my ideas

Patrick Winston, for his lifelong lessons

Ike Colbert and Blanche Staton for their unconditional support

Mark Burry, Jordi Fauli and Jordi Cuso who allowed me access to the work of Antonio Gaudi

Kenfield Griffith, who help me with writing code

The students in the PhD forum, for challenging my ideas

The students in the Digital Mockups class for their belief and hard work.

Contents

Introduction

Introduction

With the increasing demand of flexible tools for Computer Aided Design (CAD), Parametric Modeling is becoming a mainstream of *Computer Aided Architectural Design* (CAAD) software, in order to make variations in the design process less difficult. This is traditionally called *Parametric Design*. Until recently, parametric design was understood as highly sophisticated and expensive software made exclusively for manufacturing in aerospace, shipping and automobile industries. However, designer's demands for flexibility to make changes without deleting or redrawing in a computer has pushed the incorporation of parametric modeling as standard tools in traditional CAD programs.

Variations in design are a fundamental part of the design process in the search for solutions to design problems. Design variations support improvement of design which in turn improves the quality of designed artifacts. Designers constantly go back and forth between different alternatives in the universe of possible solutions, working in a particular part at a given time, or looking back at the whole from a broader perspective. This is a continuous and iterative search process of variations of a design idea, and it is very likely to revisit a previously abandoned solution to rework it. As a result, designers demand flexible tools that allow variations in the design process until a solution is established for further development.

In this context Design Procedures (DP) is presented as a methodology that enhances the design capability of a parametric model to perform design variations. By using shapes as parameters and thinking of parametric design as a general procedure.

Consequently, a parametric model becomes a flexible tool allowing changes at the topological and geometrical levels.

The thesis starts by presenting axiomatic definitions of Design Procedures and parametric design. Chapter 1 introduces the idea of Design Procedures which is followed by a brief overview of traditional parametric models accompanied by examples presented on chapter 2. Chapters 3 presents important definitions of symmetry and elaborates on the description of rod symmetry groups. Chapters 4, 5 and 6 present three case studies on the application of Design Procedures. The first case is on the use of Design Procedures as an analytical tool and a computational design generative system. The second case discusses how Design Procedures are used to make descriptions of complex shapes. The third case study is on the generation of non-Euclidean geometrical forms for twisted high-rise buildings. Design Procedures (DP) are defined as a systematic methodology to overcome the limitations of traditional parametric models.

Chapter 1

Design Procedures

Chapter summary

This chapter introduces idea of looking at design from a computational perspective as a procedural enterprise. In computation are like recipes that contain a set of instructions that describe the steps necessary to complete a task. Analogously a Design procedure is a set of axioms or instructions that are carried in a systematic way to generate a design. This chapter describes the important aspects of design procedures and provides brief axioms that define design as a procedural pursuit.

Procedure

A procedure is a description of the steps necessary to accomplish a task. In computation, procedures are used to perform specific functions.

Procedures allow abstraction and encapsulation of complexity and reusability

In abstraction procedures usually have three parts: The name of the procedure, the arguments of the procedure, and the description / steps of the procedure.

The name is a handler that is used to call the procedure

The arguments are used to contain the parameters that will be used to perform the procedure. It is like the ingredients in a recipe

The description of the procedure is the set of instructions that are performed to make the procedure. It is the recipe itself. The user has access to the name and knows what it does through a description. When a procedure is called, the user hands over the values of the parameters which replace the arguments on the procedure. The procedure makes the necessary calculations and returns the answer to the user. In most cases this portion of the procedure is hidden from the user which is called encapsulation.

Parameters

Parameter is a term that has many definitions depending on the use. The term parameter comes from mathematics and it refers to a factor that controls the values of other factors with respect to a linear relation.

In computation, a parameter is the argument or series of arguments of a function with takes values as inputs. A parameter is also the placeholder for the value of a variable. They are used in the arguments of the procedure

In design a parameter is a non geometrical entity that can hold a value to control geometrical components or relations between geometrical components.

Parameters are used to substitute specificity for generality.

Geometrical Modeling as Procedures

In computational geometry, geometrical models are constructed in very specific ways. Since they are attached to specific data structures that will hold/contain the information, they must be constructed in specific ways which can be encapsulated as procedures.

There are procedures to construct primitive objects like points, lines, polygons and solids, and there are also procedures to construct more complex objects: Boolean operations.

Procedures to make geometry can have parameters that are use as inputs, therefore making the procedure a parametric procedure.

Design as Procedure

In some cases, design can be described as a step by step process, where some things can occur over and over again. This can be interpreted as a procedural way of making design

In computational, the geometry that describes a design can be also generated as the result of a script (procedure) or recipe that when followed step by step yields the same results.

Procedures to make designs can have parameters that will take different

Shapes as Parameters

In a procedure a parameter takes the form of a variable whose value can be altered to obtain different results. In a design procedure, numbers, relations, shapes and operations are treated as parameters.

If a shape is seen not as an explicit representation, but as the result of a procedure, any geometrical component used to generate it is an input of the procedure. A cube can be modeled as the result of the following procedure: A square shape that is extruded along an axis. The square as the initial shape is a parameter of the extrusion procedure. If the initial shape is substituted for another shape, like a circle, then the extrusion procedure generates a cylinder.

Operations as Parameters

In addition to having numerical values as parameters, there can also be operations as parameters. Operations will take the form of Boolean that will apply or not depending on the local conditions found. The operations can also take the form of a rule where they are the result of IF-THEN statements. If a particular condition is found and a rule is triggered, the result is a Boolean of the form YES/NO that will either activate or inactivate a particular.

Design Procedures

A design procedure is a set of instructions that performs actions to create a design. The design procedure carries instructions in a systematic order where all geometrical components that represent a design are parameterized.

In a design procedure, shapes, numbers, variables and operations are parameters.

Chapter 2

Parametric Design

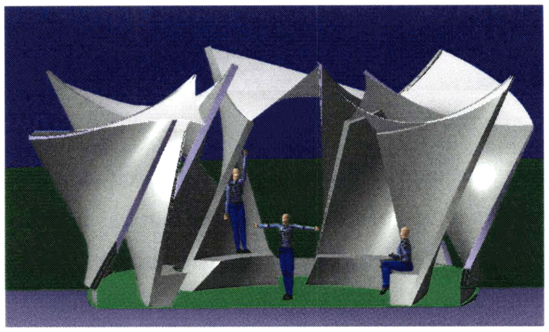
Chapter summary

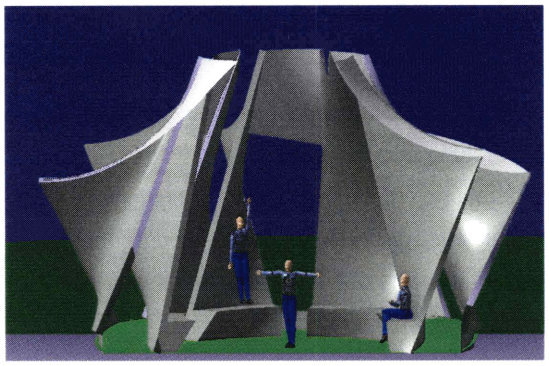
This chapter introduces the fundamental concepts of parametric design, starting on the premise that design is variation. Presents examples of classes of parametric models accompanied by illustrations of parametric models and discusses advantages of parametric design.

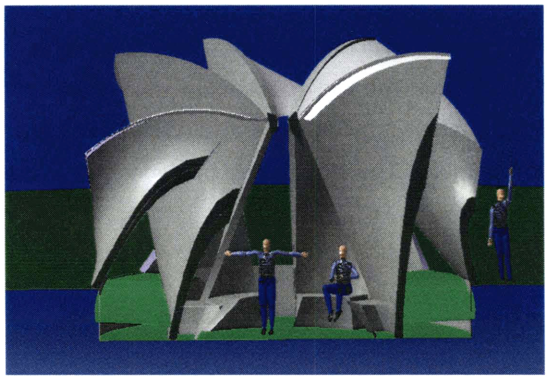
Variations in Design

Design as a process contemplates the search for solutions where there is no predetermined set of alternatives to choose from. As fundamental part of the design process, design variations allow for the search of better solutions to design problems. Designers constantly go back and forth between different alternatives in the universe of possible results, working in particular solutions at a given time, or looking at the whole design for synthesis.

In this iterative and continuous process it is not only possible, but very likely to revisit a previously abandoned solution to be reworked under a different set of criteria or tested under a different set of constraints. As a result a new design solution can emerge. Variations in design offer framework for improvement of current design solutions, through optimization and change, which in turn improves the quality of design artifacts. In this context, the elaboration of computer models that offer flexibility to allow variations of a design idea, and to be adapted to changing conditions during the design process, has become a field of its own domain, which is known as *Parametric Design.*







2.1 DESIGN VARIATIONS

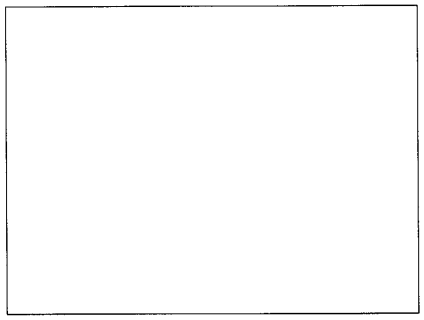
Here shown 3 design variations from the design

Parametric Models

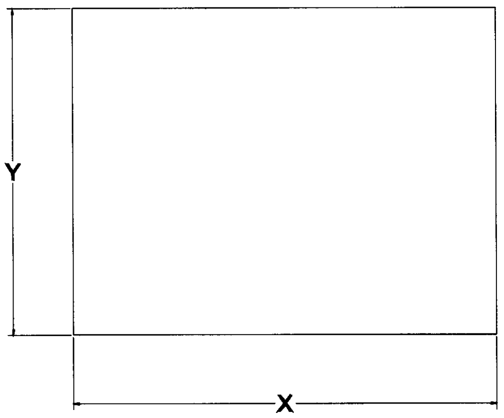
Geometric models can have two kinds of representations: *Explicit* and *Parametric.* An explicit model is a type of geometric model that has fixed attributes, therefore in order to perform any kinds of transformations of the model, it is necessary to erase and redraw the geometrical components. Variations can only be performed if a particular shape is literally substituted for a new shape.

On the other hand, a parametric model is characterized by having attributes that allow variations without erasing and redrawing any of its geometrical components. In this case, variations are carried out by changing the values of the parameters, allowing these variations to be propagated through the dependent attributes. Therefore, to carry out changes it is not necessary to erase and redraw.

A *Parametric Model* (PM) is a geometrical representation of a design that has some attributes or properties that can vary (parameters) and other attributes that are fixed. The attributes are controlled by a non-geometrical component that is called a parameter. The attribute will be dependent of the value of the parameter.



2.2 Explicit model of a rectangular shape. To perform variations it is necessary to erase and redraw a new rectangle.



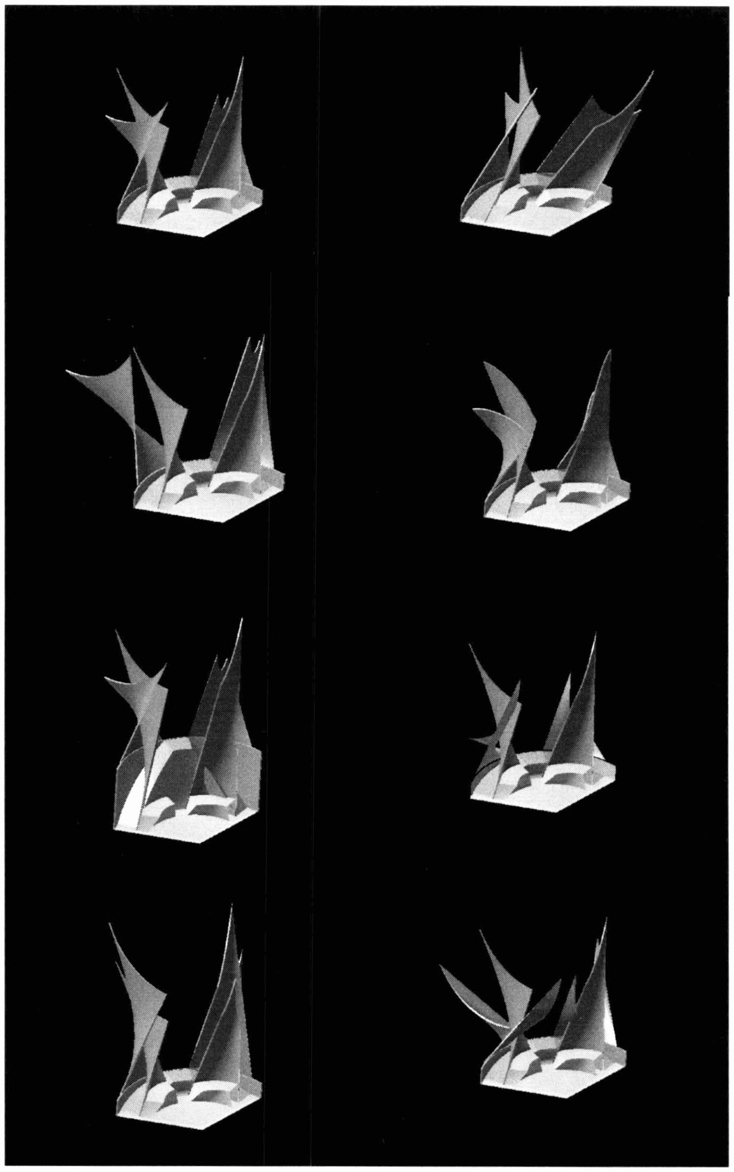
2.3 Parametric model of a rectangular shape. Note how the length and height attributes are parameterized by the **X** and **Y** parameters. The size of the rectangular shape can be altered by changing the values of the **X** and **Y** parameters, therefore to perform variations we use the parameters.

Parametric Design

Parametric Design (PD) is the process of designing with Parametric Models in a setting and/or environment where variations are effortless, thus replacing singularity with multiplicity in the design process. A Parametric Model (PM) is a three-dimensional computer representation of a design, constructed with geometrical sets of shapes that have some attributes (properties) that are fixed and others that can vary. The variable attributes are also called parameters and the fixed attributes are said to be constrained. The designer changes the parameters in the PM to search for different alternative solutions to the problem at hand, and the PM responds to the changes by adapting or reconfiguring to the new values of the parameters.

Each of the design variations obtained is called a design instance since it represents a definite value or sets of values of the parameters at a specific point in the design. Figure 1.4 shows different instances of design based on variations of the parameters of the original parametric model. The instances can be organized in a matrix format that shows how design variations occur.

Parametric Design implies the use of declared parameters to define a form. This requires rigorous thought in order to build a geometrical model embedded in a very sophisticated structure appropriate for the needs of the designer. Therefore the designer must anticipate which kinds of variations he might want to explore in order to determine the kinds of transformations the PM must allow. This is a very difficult task because of the unpredictability nature of the design process.



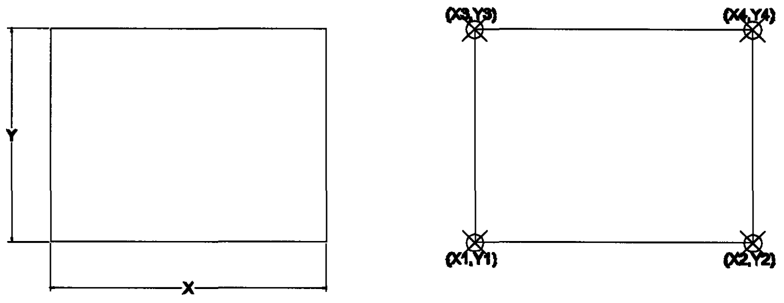
2.4 Parametric model showing variations. Each of the variations is called an instance of the parametric model.

Parameterization of Geometrical Models

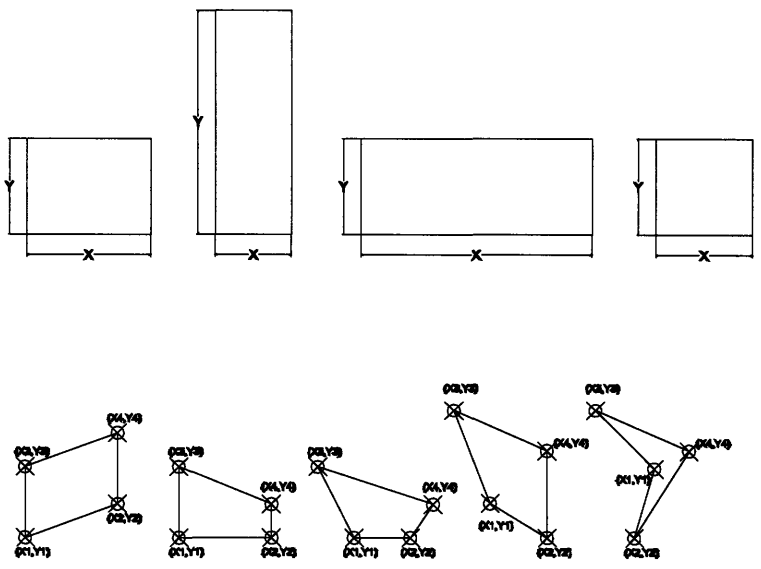
Parameterization is the process of assigning parametric attributes to a geometrical model which will determine how the geometrical components will vary. In other words is the process by which an explicit model is transformed into a parametric model.

Depending on the behavior desired by the designer any geometrical shape can be parameterized in different ways, therefore, a geometrical model can be subject to more than one parameterization schemas creating different ways to perform design transformations. Figure 2.5 shows two parameterization schemata for a rectangular shape. The two parametric schemas will let the designer create different instances based on the transformations that the parametric model allows. Figure 2.6 shows how the two parametric model schemas generate different design instances based on the kinds of transformations that the parametric model allows.

All instances created by one parametric model form a *family* of designs, despite the fact that one particular instance from a parametric model can be exactly the same as another instance from another parametric mode.



2.5 Parameterization Schemata of two rectangular shapes. The first schema shows a rectangular shape with *length* and *width* as parameterized attributes. The second schema shows the same rectangular shape with the position of the enc points as parameterized attributes.



2.6 Families of designs created by each of the parametric schemas.

Constraints

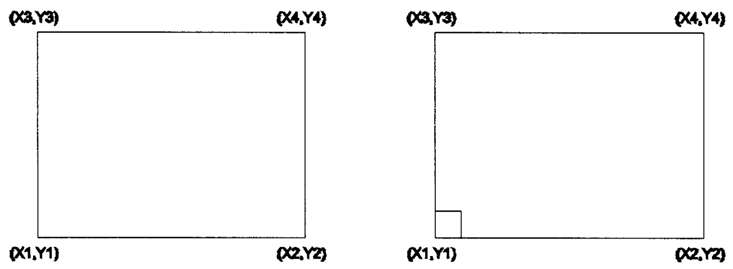
A constraint is a non-geometrical entity that limits the behavior of an attribute. Like the parameter, a constraint will control the behavior of an attribute or group of attributes. However the constraint will limit the scope of action of a parameter or group of parameters putting restrictions on the possible parametric variations.

Constraints can be of two types: geometrical and dimensional. Geometrical constraints will build relations between two geometrical entities. A geometrical constraint will limit the scope of action of one geometrical entity with respect to another geometrical entity. Some geometrical constraints include relations such as parallelism and perpendicularity, while others might have specific localized conditions such as the midpoint of a line[1].

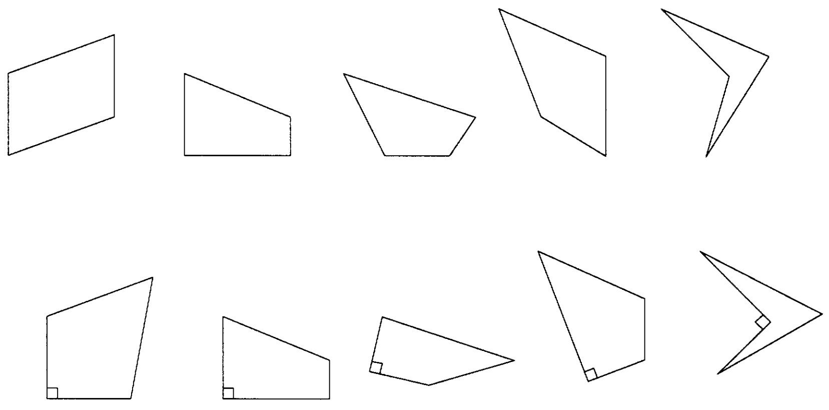
Dimensional constraints are defined as parameterized attributes with a fixed value. They are analogous to static variables, which can only take one value. Dimensional constraints form relations built between one geometrical component and another non-geometrical entity. The constraint will set the attribute to a fixed value, which can only change if the constraint is changed or removed.

A constraint model will only allow variations based on the scope of actions allowed by the constraints. In case of conflicting values between parameters and constraints, the constraints supersede any parameters that the model has. Constraints are also unidirectional, which means that constraints control the values of the parameters and not vice versa. A constraint is the parent-object (controlling entity) and the constrained attribute is the child (controlled entity).

[1] The midpoint of a line will always remain at the same relative distance of the two extremes regardless of the length of the line.



2.7 Two rectangular shapes showing a parameterization schema by endpoints. The figure on the left is unconstrained, while the figure on the right has a perpendicular constrained in one corner. This constraint will force the two edges meeting at that point to be perpendicular to each other.

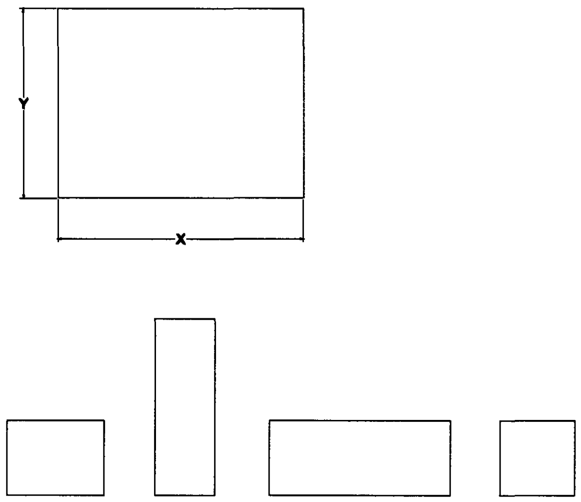


2.8. Family of designs produced by the parametric models. Note how in the second case the constraint acts to keep both edges perpendicular to each other, while the unconstrained model has the freedom to take on any variation, including all kinds of angles.

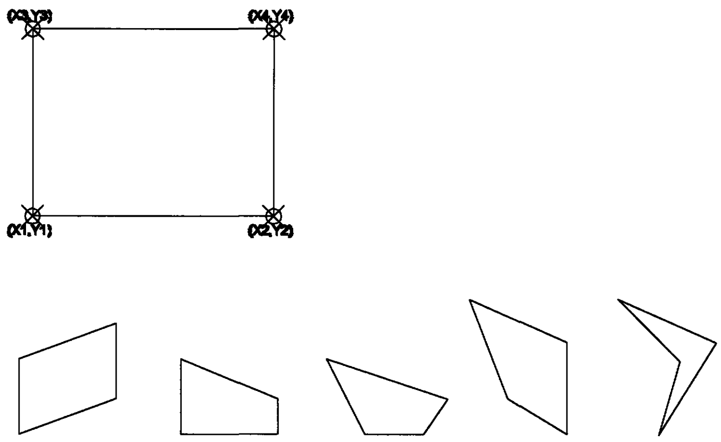
Degrees of freedom of a Parametric Model

Degree of freedom refers to a specific combination of the parameterized components, the existing constraints, and the kinds of transformations that they both allow. A parametric model with more parameterized attributes is said to have more degrees of freedom. As a consequence it will allow a larger family of designs. A parametric model with a high number of constraints will have less degrees of freedom limiting the instances that can be produced. In simple terms, adding parameters increases the degrees of freedom, while adding constraints reduces them.

A parametric model with a large number of parameterized attributes needs a higher and more complex structure to control the number of parameterized components. Therefore a specific transformation will require a large number of operations to be carried at a specific instance.



2.9 A parameterization of a rectangular shape by length and height attributes. This parametric model has two (2) degrees of freedom, allowing rapid regeneration of different design instances. The tradeoff is that all the instances are rectangular shapes.

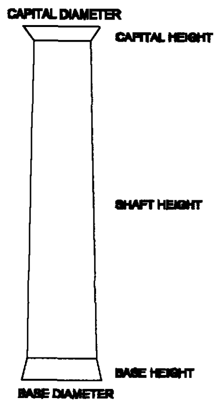


2.10 A parameterized rectangular shape by the coordinate endpoints has eight (8) degrees of freedom, allowing more variations on the designs. However this model requires a larger data structure to store and manipulate the eight variables controlling the coordinates of the endpoints.

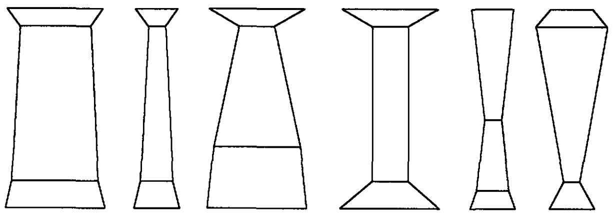
Parametric Variations

Parametric Variations (PV), also known as variational geometry, or constrained–based models, is a kind of PM based on the declarative nature of the parameters to construct shapes. The designer creates a geometrical model, which attributes are then parameterized and constrained, based on the desired behavior, thus creating a *parameterized modeling schema*. A parametric modeling schema shows which attributes of a geometrical model are parameterized and how the designer can change the values of the parameters.

The idea behind a PV model is that the geometrical components are controlled by means of changing the values of the parameters or constraints without changing the topology (number of components and their relations). The parametric modeling schema creates the *master model*, which is the starting point for parametric variations of the designs. Every time that the designer changes a parameter in the master model, a *design instance* is created. The collection of design instances generates a *family* of design solutions based on changes done to the parameterized components. The main characteristic of a PV is that the model allows transformations of the geometry without erasing and redrawing, in a closed contained system.



2.11 A parametric model of a column with a description of the parametric attributes. This kind of mode is called a parametric variation or variational geometry.

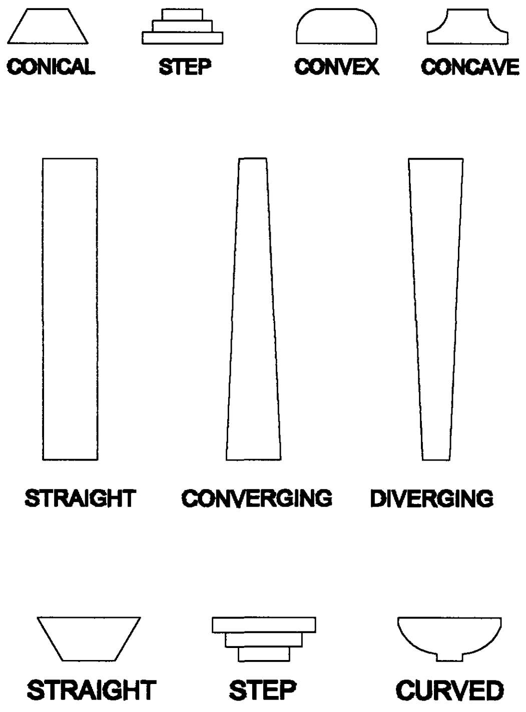


2.12 Family of design instances based on parametric variations. Here showing six (6) instances of design based on parametric variations.

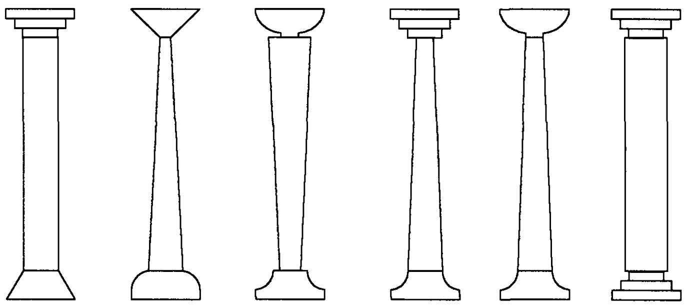
Parametric Combinations

Parametric Combinations (PC) are composed by a series of parameterized geometrical shapes that are put together to create more complex structures for design exploration. Also known as associative geometry models, or relational models, PC offers another degree of complexity beyond the parameterization of the geometrical components, which is done by constructing sets of relations between the parameters and the shapes.

By combining components in different ways, a variety of designs solutions are achieved from the initial vocabulary and the rules of combination. The strategy is to sub-divide a design into several components and derive specific ways in which the components will be combined. For example, a column can be divided into three components: base, shaft and capital; where each of the components has different possibilities of instantiation. A column design is the addition of the three components in an orderly way, first the base, then the shaft and finally the capital. A family of designs is obtained by combining the different components according to the rules of combination. If the number of components is fixed, the family of designs is limited to the number of possible valid combinations.



2.13 A parametric combination model of a column showing components for the base, shaft and capital.

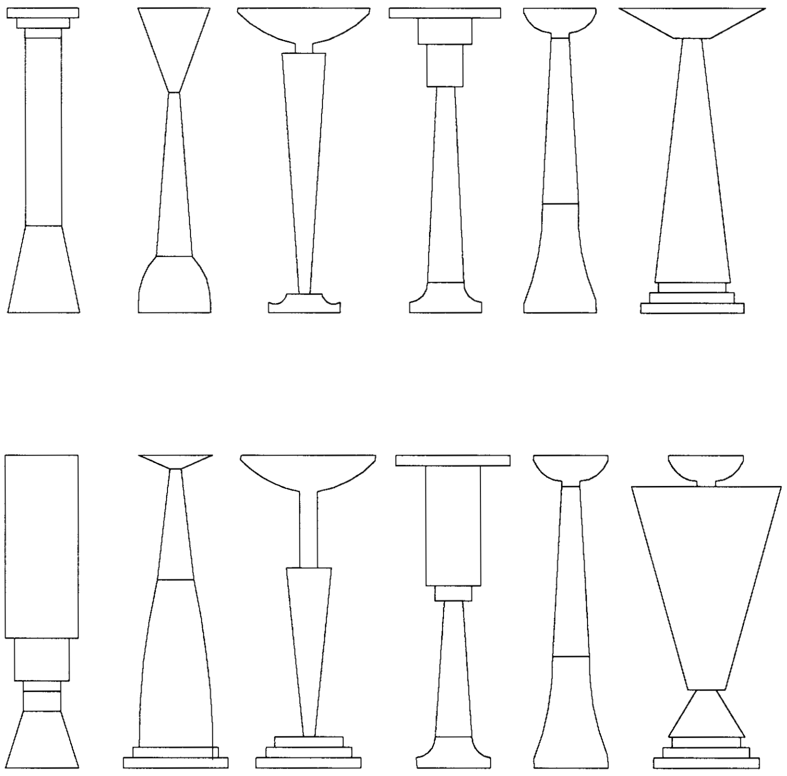


2.14 Family of designs of the column based on the parametric combination of the components.

Parametric Hybrids

Parametric Hybrid Models are formed from a combination of both parametric variations and parametric combinations. Hybrid models take the benefits of both types of parametric models and can be very robust for design exploration. However, in the real world they require a very solid data structure and demand a lot of computational power.

They are very difficult to construct and the chances of producing a collapse of the model increases proportionally to the complexity of the model. In some occasions is better to construct and design in two models in parallel, one for variations and another for combinations, instead of a hybrid model. Nevertheless, a hybrid model can proved usefulness in cases where simple parametric models are sufficient for design exploration.



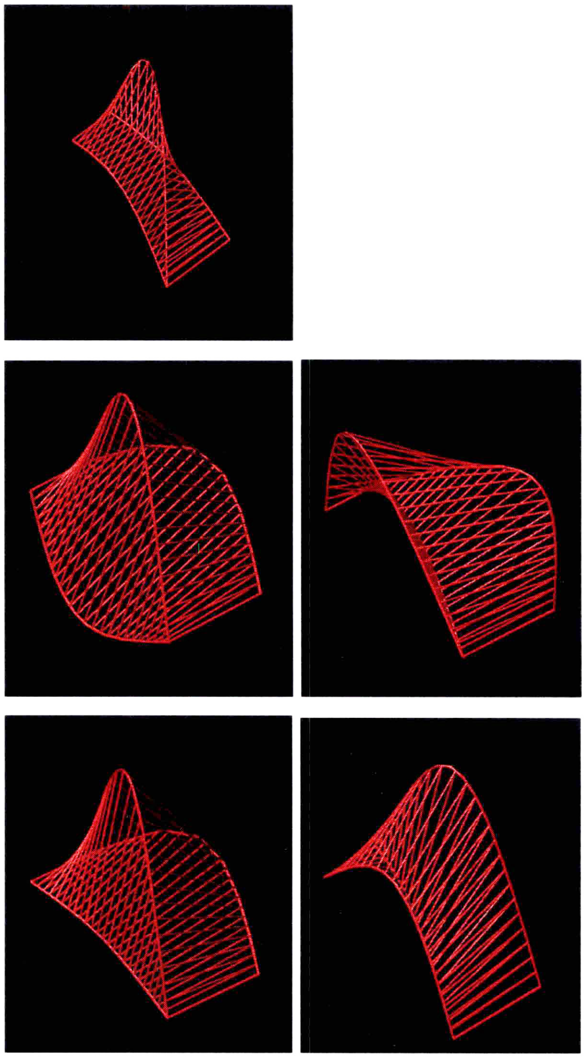
2.15 Family of designs of the column based on the parametric hybrid model. The number of possible instances increases in proportion to the number of combinations and the parameterization of the components. This produces some unexpected and interesting results.

Variants Programming

The very first approach to parametric design was through automation of repetitive tasks and the inclusion of variables to make different designs: a script. This piece of computer code, also know as programmed construct, contains parameters that are edited to obtain a family of variants of the same shape. With the use of simple programming modules, the designer can create routines containing repetitive tasks or simple procedures that can be used to build large model in small steps.

Scripting also allows incorporating interaction with the user, in most cases by asking the user to input the values of the parameters once the program is called, but before is executed. With a little knowledge of programming, the user can create its own small routines and use the procedures built in the program to generate the geometrical model by encoding a simple set of instructions that will be carried in orderly manner. The main limitation is that there is no interactive editing on the model once the routine is called and the procedure has started. It will not allow editing once the model is created. The only way to change the model is to delete the model and rerun the program to create a new model with different parameters. Its advantage is that allows the creation of a vocabulary of models that can be reused in design.

Although variants programming is well established as a computational procedure and most current software applications have the computer power to handle the scripts, architectural offices are still relying on the modeling skills of the CAD operators, therefore the models are literally made by hand using a mouse and the palette offered by the CAD program. This attitude is considered by some people as a waste of computing power, like getting a Ferrari and attaching it to a mule to ride in it. Nevertheless, some architects are recognizing the benefits of variants programming to be incorporated in the design process. For it to be successful it requires two important changes of paradigm: 1) Architects and designers must engage in the knowledge of computer programming, until new more user friendly programming interfaces and languages are develop, which means architects must study and learn how to program a machine; and 2) Architects and designers must define very well the problem they want to solve before it can be implemented in some form of computational format. For the computing point of view, a problem well stated is a problem half-coded.



2.16 Design instances from a variants programming model. The model in written in a script format that takes inputs from the designer and generates an instance of the model based on the input values.

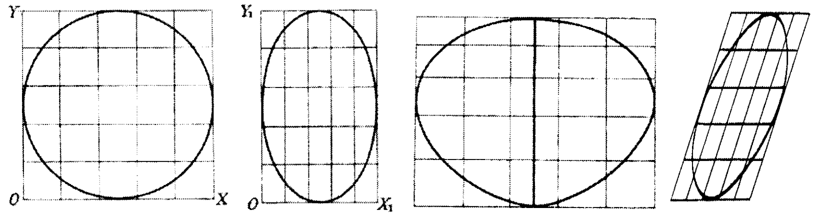
Darcy Thompson and the comparison of related forms

According to Darcy Thompson[2], the study of form can be done in two ways: descriptively, using verbal language; and analytically, using mathematics. For instance, an apple, an orange and a ball, are verbal descriptions of a spherical shape. However, such descriptions are limited and ambiguous. Thompson argues that analytical descriptions, founded in mathematical language, can provide precise definitions of form. For example, the formula $X^2+Y^2+Z^2=r^2$ is the mathematical description of a sphere, were the sum of the squares of three variables equals the square of the radius, and the variables indicate the location of the center of the sphere in the Cartesian space.

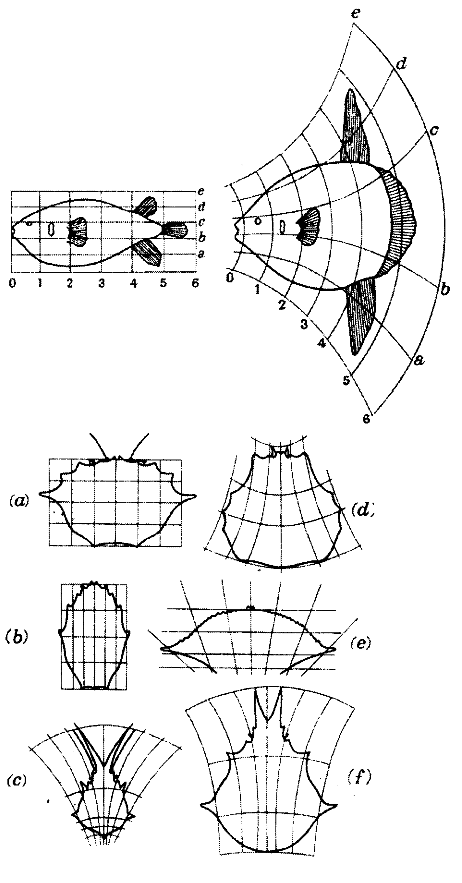
The method of Cartesian transformations is rooted in analytical descriptions of form. Its predecessor is the method of coordinates, which was used to translate curves into numbers and then into written data. The data could be used to recreate the curves by reversing the process, from data into numbers and then into curves (Thompson, 1917). This was mostly used by cartographers when making maps at different scales.

Instead of using one mathematical description of a particular form, Thompson's method of Cartesian transformations is concerned with the study of form by comparison. Darcy Thompson believed that one form can be easily understood by recognizing it as a permutation or deformation of another form[3]. These kinds of deformations are carried out systematically and recorded in a step-by-step mode. Thus two or more forms can be compared by means of the set of grids. For the morphologist who studies the form and structure of organisms, Cartesian transformations provide a visual framework to understand how forms are related and what kinds of transformations occur in between.

[2] Darcy Thompson, On growth and form.
[3] This method is founded in the mathematical *Theory of Transformations*, where two groups are clearly distinguished: substitution groups and transformation groups, where former one is discontinuous and the latter is continuous. It is the latter group that the theory of comparison of related forms presented by Darcy Thompson is founded on.



2.17 Transformations of a circle based on the deformation of the underlying Cartesian grid. The grid provides a framework to perform transformations while it serves as an analytical description of the current instance. (Image reproduced from On Growth and Form)



2.18 Examples of the application of the Cartesian transformations applied to show how living organisms can be compared. (Images reproduced from On Growth and Form)

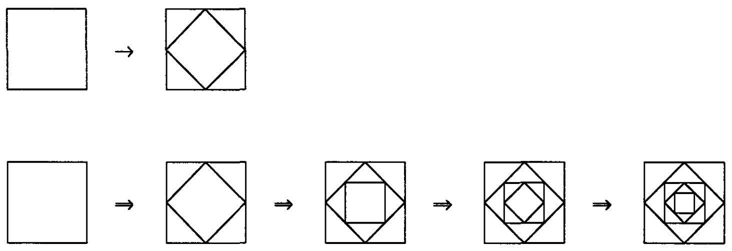
Transformations of Shape Grammars

Shape grammars are algorithmic systems for analyzing and creating designs directly through computations with shapes. Shape grammars are powerful to the study of form from the strict visual point of view. A very interesting aspect of shape grammars, which concerns stylistic change and transformations in design, deserves special attention. Any design generated by the grammar is considered to be in the language of the grammar; likewise any design that is not generated by the shape grammar is not considered in the language of the grammar. Each shape grammar generates designs in a language and provides different understanding of the theory of languages of designs (Knight, 1994).

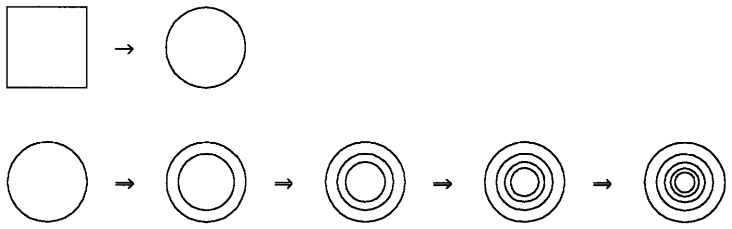
Terry Knight's model is based on the premise that a shape grammar will generate a unique language of designs; therefore, in order to determine whether two designs are from different languages, it is necessary to compare the shape grammar of each design. Comparing the shape grammars can be used to analyze how languages of designs have evolved through history and to study the transformations of a particular design language. According to Knight, the design language can be transformed by transforming the shape grammar, instead of transforming the designs. This is accomplished by three operations performed on the shape grammar: rule deletion, rule addition and rule change. A shape grammar will generate a body of designs in a language. If a rule is deleted from the shape grammar, then the body of designs will be reduced. Likewise, when a rule is added to the shape grammar, the body of designs will be expanded based on the new added rule[4]. If a rule is changed, then the transformed shape grammar will generate different designs than the original shape grammar. In any of the three cases, the corpus of design will change whereas the new shape g grammar will generate more, fewer or different designs than the original shape grammar[5].

[4] It is important to clarify that rule addition does not necessarily mean that the design will expand, or that a rule deletion will decrease the corpus of designs. A rule added to the shape grammar might limit or expand the corpus (body?) of design. A rule that is deleted from the grammar can also expand or limit the corpus (body?) of design. The main argument is that designs will be transformed as a result of transformations in the shape grammar.

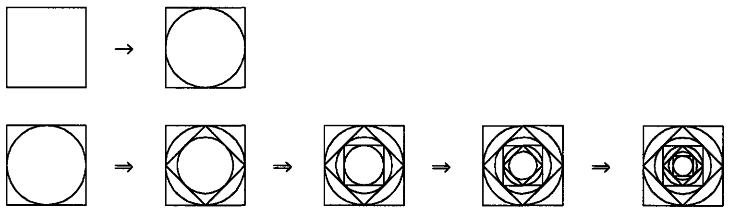
[5] It is important to note that the rule change can be considered as be performing a rule deletion and rule addition at the same time.



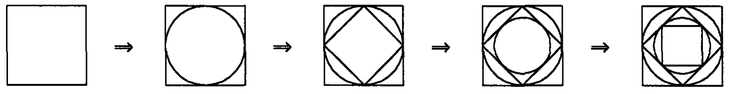
2.19 A shape grammar and d derivation.



2.20 Transformation of the rule and the corresponding derivation resulting in a new design language.



2.21 Another transformation of the rule and the corresponding derivation which creates new design language



2.22 A derivation of the alternative application of the rules of the first two shape grammars. Note how the final design corresponds to the third step on the previous derivation. Although both designs are identical, they are of different languages since they were generated from two different shape grammars.

Chapter 3

Symmetry

Chapter summary

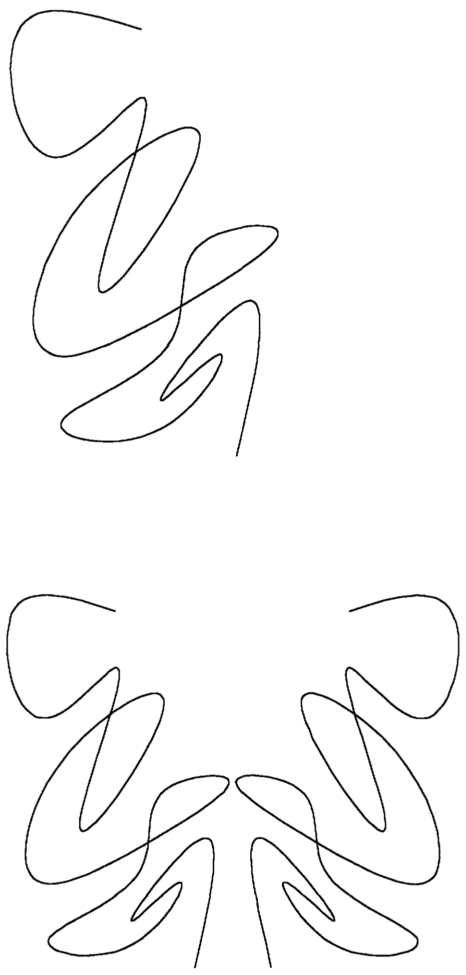
This chapter introduces the concepts of symmetry shows how it is applied to the single rod groups. Starts by presenting the fundamental concepts of the basic symmetry groups and shows how they can be used to build descriptions for complex shapes with high levels of symmetry.

Symmetry

Symmetry as a concept is understood in more than one way. Symmetry is recognized as a property of an object being well proportioned and balanced where its parts are somewhat corresponding in shape and size. It is also understood as a feature of living organisms where there is evident correspondence of constituents arranged in two opposite sides of an imaginary line[6]. Yet there is a third way in which symmetry is defined is the result of an arrangement in which regular patterns seem to emerge[7]. All of the former general notions of understanding symmetry are right in their own views, but having more than one definition for something only leads to confusion and ambiguity. Two fundamental concepts emerge from the former definitions of symmetry: the idea of relative equality and the idea of regularity. We will come back to these two concepts later.

In mathematics symmetry is defined as a characteristic of geometrical shapes such that when a transformation[8] is performed on the shape it does not appear to change. If a square is rotated 90 degrees with respect to its center, it will appear that no rotation took place. As a result, when a regular shape is subject of a Euclidean transformation, it looks the same visually, apparently it did not change. Therefore symmetry is associated with some notion of regularity where change is not evident.

[6] This is known as bilateral symmetry, where similar parts are arranged in two sides of a median axis or median plane. This is more evident in biological specimens and living organisms
[7] The idea of harmony and symmetry is best described in music, where rhythm creates patterns
[8] By transformations we mean Euclidean transformations



3.1 An object with no symmetry shown with a bilateral symmetry composition. Bilateral symmetry is the most basic class of symmetry.

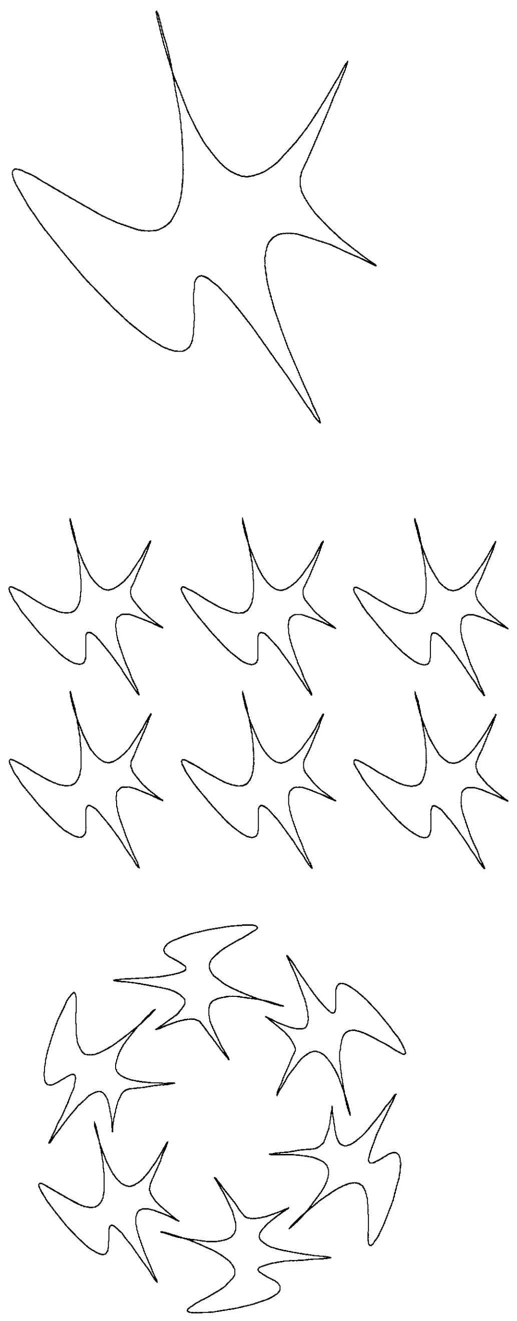
Relative Equality and Regularity as the basis of symmetry

According to Shubnikov and Koptsik[9], *relative equality* is the first notion necessary to better understand symmetry. For example, the right hand is relatively equal to the left hand. They are equal in the sense of their constituents, since both have 5 fingers. However a right hand and a left hand can not be superimposed to make a perfect match without a reflection over a mirror plane. The hands are relative to each other with respect to the mirror plane, thus the equality of both hands dependent on the mirror plane. Both hands are *equally related* through its constituents, the five fingers, and their relative position with respect to an imaginary mirror plane.

Regularity is the second fundamental concept of symmetry. By having the same number of constituents both left and right hands are considered regular, at least at the topological level. Other way of interpreting regularity is to divide a shape into equal parts without any remainders. A square that is divided into four smaller squares shows that the smaller squares are as regular as the larger square. .Furthermore, the idea of regularity is enhanced if the little squares are recursively subdivided into smaller squares. The predictable nature of the created pattern is the most important notion of regularity, which is fundamental to the understanding of symmetry.

Equality and regularity can be found in different ways. A regular shape that can be subdivided into parts shows evidence of emergent patterns that are either regular and or relatively equal. If a square is subdivided into smaller squares regular patterns emerge. If the square is subdivided into triangles, then the triangles are relatively equal to each other with respect to Euclidean transformations. This essential idea of relative equality and regularity is the basis for symmetry or a theory of symmetry.

[9] "Symmetry in Science and Art"



3.2 A non-symmetrical object used to generate two kinds of symmetrical compositions. Symmetrical compositions are generated by translation and rotation. All objects in the composition share the two main characteristics of symmetry: relative equality and regularity.

Finite Symmetry

The previous chapter introduced the idea of relative equality and regularity as the basis of symmetry. Relative equality is the man characteristic of shapes that belong to the *finite symmetry* group. Finite symmetry is based on rotation and reflection transformations and is characterized for having a central point or central axis from which the transformations occur.

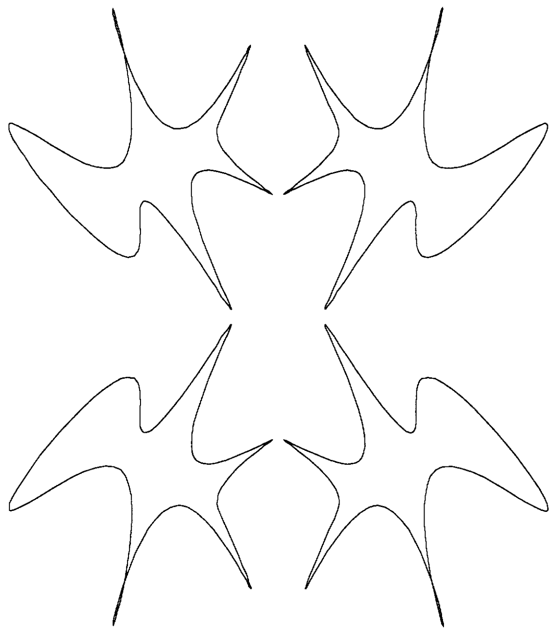
The most basic case of finite symmetry is found on the bilateral symmetry where two shapes are relatively equal to each other by means of the mirror plane. Mirror symmetry is denoted with the letter **M**, therefore any composition which has one mirror plane is denotes as having symmetry **M**.

A classical example of bilateral symmetry is found on the figures of the *Rorschach Inkblot Test*[10]. Another common example is a painting game where figures are created by randomly pouring ink on paper and folding it in half. When the paper is unfolded a figure with a perfect symmetry axis appears.

Bilateral symmetry is also present in most living organisms, in particular those capable of motion. The limbs are present in equal numbers and sizes to both sides of an imaginary central axis. Crabs, insects, mammals, etc, have all bilateral symmetry.

Some cases might include more than one mirror. Kaleidoscopes are perfect examples of this kind of symmetries, where a figure is created with multiple mirrors. In this case, the particular object is said to have more than a single mirror

[10] Named after Swiss psychiatrist Norman Rorschach, the Rorschach inkblot test is a projective test practiced to measure emotional and intellectual functioning of an individual. Beyond the controversy of the validity of the test or its results, what is important to the scope of our work is that the inkblots are perfect examples that illustrates bilateral symmetry.



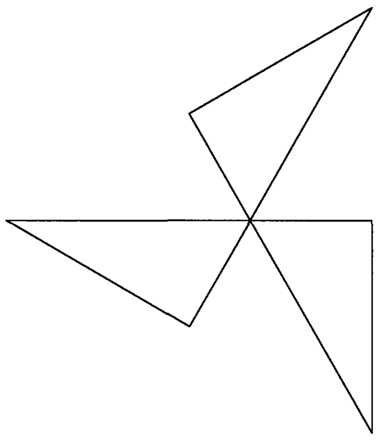
3.3 Composition s with a finite symmetry. In this case the composition has two planes of symmetry, one vertical and one horizontal.

Point Symmetry

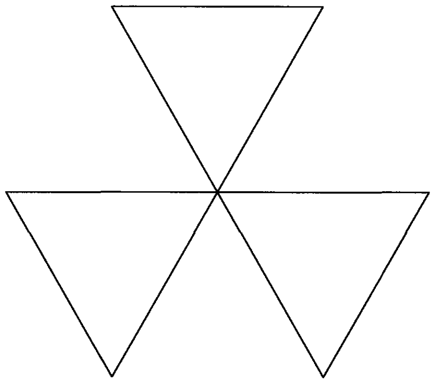
Point symmetry is the second class of the finite symmetry group. In point symmetry the relative equality occurs by means of rotations, reflections or both. Objects rotate with respect to a fixed point and/or are reflected from mirror lines that go through the same point. Therefore point symmetry's main characteristic is the presence of an identifiable central point.

Let us take a closer look at figures with rotations only. Rotations can occur at any interval between 0 and 360 degrees. The number of rotations is multiple of 360 denoted by the letter **N**, and indicates how many rotations occur before the figure can be aligned with itself. A design with point symmetry is a design that can be rotated 360/N degrees in order to be aligned with itself. Each rotation of 360/N degrees makes the figure identical to the original design. Since this can be done **N** times before the figure completes a full rotation, the symmetry of the figure is **N**. The number corresponding to **N** is calculated by dividing 360 by the number of degrees that will have to be rotated to obtain the same figure. For example, design that has symmetry N=3 will have 3 rotations of 120 degrees corresponding to 360/3. A design that has symmetry N=5 will have 5 rotations of 72 degrees. Likewise a design with a symmetry of N=4 will have four rotations of 90 degrees to align the shape with itself. This kind of symmetry is known as cyclic symmetry and it can also be denoted as "C_N" where the subscript N indicates again the number of rotations. Cyclic symmetry has a sense of direction; therefore rotations can occur either clockwise or counterclockwise.

When rotations are combined with reflections, another class of symmetry emerges, which unlike the cyclic symmetry, it has no sense of direction. Again, the presence of a mirror plane is denoted with the letter **M**. A square for example, has four rotations, but at the same time it has four reflections, one vertical and one horizontal, and two along its diagonals. This particular example has a symmetry which is denoted by the formula **NxM** or **4M**, where **N** indicates the number of rotations and **M** indicates the presence of mirror planes. Other notations call this symmetry group dihedral or $\mathbf{D_N}$, therefore an object with **4M** symmetry is exactly the same as a $\mathbf{D_4}$.



3.4 Composition with cyclic point symmetry. Note the rotation effect produced by the form and relative position of the three triangles.



3.5 Composition with dihedral point symmetry. Mirror planes eliminate the rotational effect.

Axial Symmetry

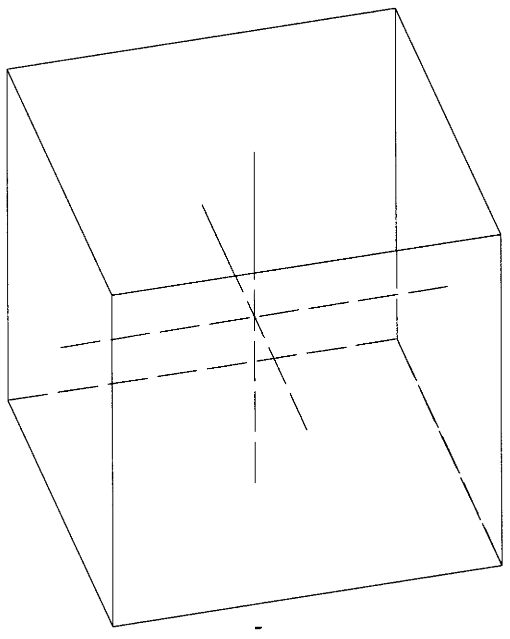
Axial symmetry is a class of symmetry that groups objects in 3D space. It is also known as *3D Symmetry* or *3D Point Symmetry*, since this class includes all 3D objects that have a singular point or singular axis. It is the three dimensional analogous to the 2D point symmetry, therefore the only transformations possible are rotations and reflections.

In most three dimensional shapes, rotations occur with respect to an axis, denoted as **A**. Analogous to the 2D point symmetry, in three dimensions **N** indicates the number of rotations that occur before the shape is aligned with its original position.

While rotations take place with respect to an axis, reflections in 3D will occur with respect to a plane. Mirror planes are of two types:

1) Mirror planes that contain the axis **A**, denoted with **M**; and
2) Mirror planes perpendicular to the axis **A**, denoted with lowercase **m**. This mirror plane is also called transverse mirror plane.

While there are some kinds of shapes that have more than one axis, like regular polyhedrons and some kinds of crystals, they will not be included in the following discussions.



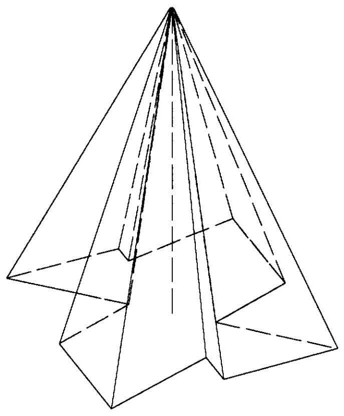
3.6 Three dimensional object showing axes. In 3D, rotations occur with respect to these axes, while reflections are performed through the mirror planes that contain them. The cube has 3 axes and twelve mirror planes. This gives the cube symmetry of 48, which means that 48 transformations will align the cube with itself.

Shapes with axial cyclic symmetry: A∗N

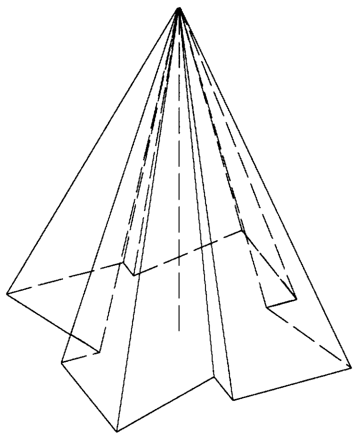
Shapes with axial rotational symmetry that will only remain the same under rotational transformations along the rod axis belong to this first class. Any other kind of transformation will change the visual appearance of the shape. In this category we find regular pyramids which base is a closed polygon with cyclic symmetry on the plane.

The notation **A*N** indicates that the shape has a single main axis from which rotations occur and the number of rotations. It is assumed that no reflections exist. Other kinds of shapes that belong to this category are regular pyramids with alternating open faces.

A characteristic of axial cyclic shapes is that they have a direction of rotation with respect to the main axis; therefore the rotation can be clockwise or counterclockwise.



3.7 Shape with symmetry A*2



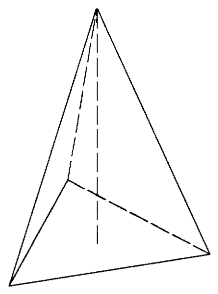
3.8 Shape with symmetry A*4

Shapes with axial dihedral symmetry: A*N*M

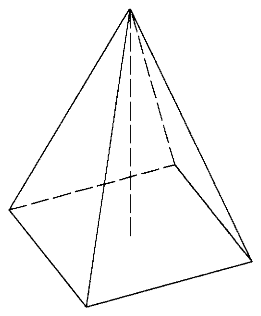
In this second group we find shapes that have axial rotation as well as a mirror symmetry plane that contains the axis. Regular pyramids made from dihedral 2D shapes and the like belong to this class. The symmetry planes pass through the central axis, making both sides relatively equal with respect to the mirror plane.

The notation **A*N*M** indicates that the shape has a single main axis from which rotations occur. **N** indicates the number of rotations before the shape completes a full 360 degree rotation and **M** denotes that there are mirror planes that pass through the main axis in number equal to **N**.

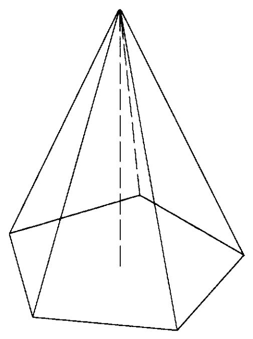
In this case the shape will remain unchanged under axial rotations, as well as reflections through any of the mirror planes.



3.9 Shape with symmetry A*3*M



3.10 Shape with symmetry A*4*M



3.11 Shape with symmetry A*5*M

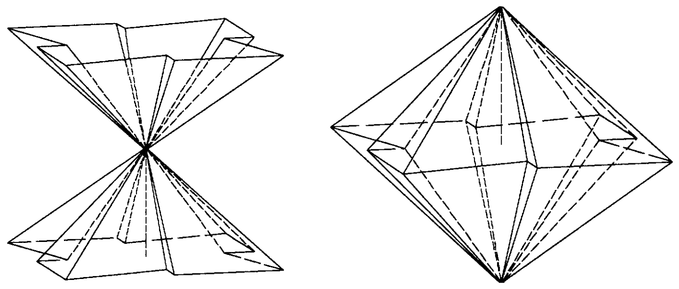
Shapes with top-down symmetry: A*N*m

This third class denotes shapes with axial cyclic symmetry that have no distinction on both ends of the main axis. In this case there are two kinds of transformations that will result in similar shapes: rotations with respect to the main axis, and reflections on a mirror plane that is normal to the main axis. This plane is also known as a transverse plane.

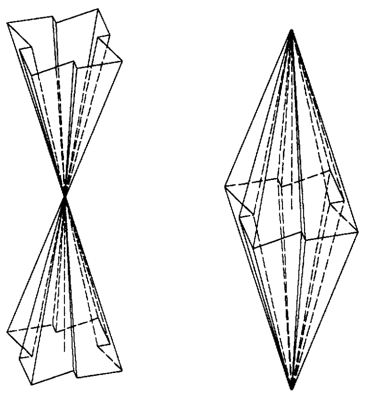
The notation **A*N*m** indicates that the shape has a single main axis from which rotations occur. **N** indicates the number of rotations that occur before the shape becomes aligned with itself, and **m** denotes the presence of a mirror plane normal to the main axis, which makes the top and bottom of the shape relatively equal. These kinds of shapes are said to have a cyclic and top-down symmetry, and they have a rotational direction.

These kinds of shapes are commonly found in special machinery. As well as some kinds of turbines and the like, which all belong to this class of symmetry. Any shape that has cyclic axial symmetry which is mirrored on a transverse plane will result in a shape with top-down symmetry.

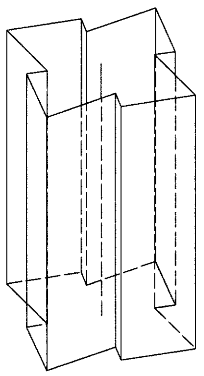
Other options are cyclic 2D shapes that are extruded along the main axis. In this case the transverse plane is more implicit and less evident to see.



3.12 Shape with symmetry A*2*m



3.13 Shape with symmetry A*4*m



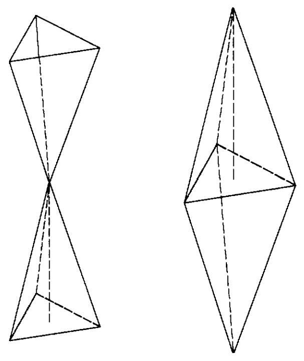
3.14 Shape with symmetry A*4*m

Shapes with top-down finite dihedral symmetry: A*N*M*m^f

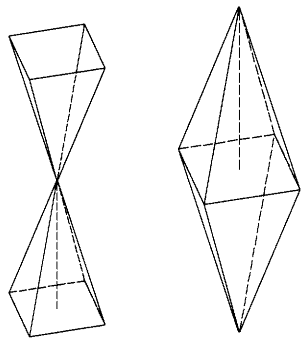
This class denotes shapes with axial dihedral symmetry with a transverse mirror plane. Three kinds of transformations will result in similar shapes: rotations with respect to the main axis, reflections on axial mirror planes, and reflection with respect to the transverse plane.

The notation **A*N**M*m^f** indicates that the shape has a single main axis from which rotations occur. **N** indicates the number of rotations that will make the shape aligned with itself, as well as the number of axial reflections. **M** indicates the presence of axial mirror planes which will make the shape with dihedral symmetry. **m** denotes the presence of a mirror plane normal to the main axis, which makes the top and bottom of the shape relatively equal. The superscript **f** indicates that the shape has a finite top down symmetry, which is distinct from the kinds of shapes that will go endlessly along the main axis.

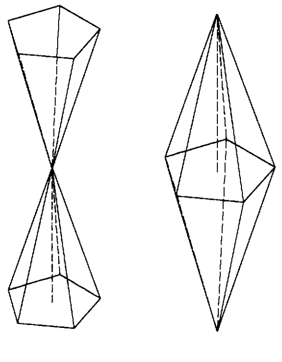
By-pyramids and by-cones are shapes that belong to this class, however there is no distinction if the shapes a joined by their base or their top vertex.



3.15 Shape with symmetry A*3*M*m



3.16 Shape with symmetry A*4*M*m



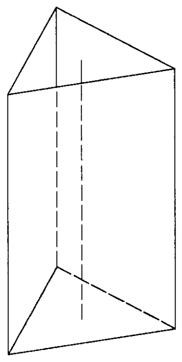
3.17 Shape with symmetry A*5*M*m

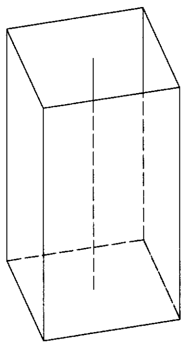
Shapes with top-down infinite dihedral symmetry: A*N*M*m^i

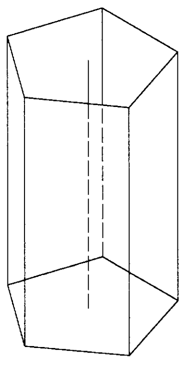
This class denotes shapes with axial dihedral symmetry with a transverse mirror plane; however they have a different flavor to them. Similar to the previous class, there are three kinds of transformations will result in similar shapes: rotations with respect to the main axis, reflections on axial mirror planes, and reflection with respect to the transverse plane.

The notation **A*N**M*m^i** indicates that the shape has a single main axis from which rotations occur. **N** indicates the number of rotations that will make the shape aligned with itself, as well as the number of axial reflections. **M** indicates the presence of axial mirror planes which will make the shape with dihedral symmetry. **m** denotes the presence of a mirror plane normal to the main axis, which makes the top and bottom of the shape relatively equal. The superscript **i** indicates that the shape has an infinite top down symmetry, which means that the shape could be extended infinitely along the main axis without changing its symmetry.

Regular prisms and 2D dihedral extrusions are shapes that belong to this class.


3.18 Shape with symmetry A*3*M*m


3.19 Shape with symmetry A*4*M*m

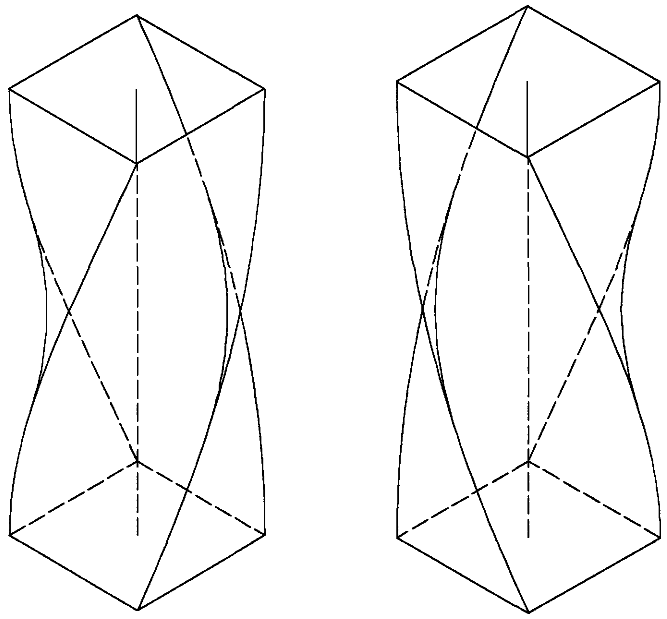

3.20 Shape with symmetry A*5*M*m

Shapes with continuous single rotation along the rod axis (twist): A*N*M*R

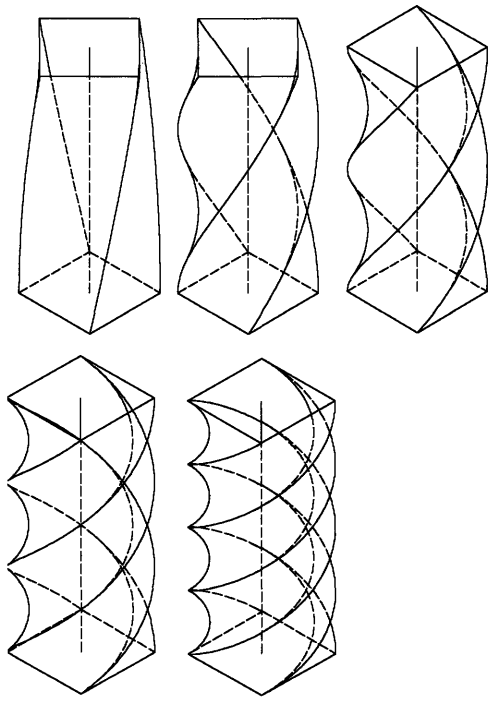
These shapes are generated by the continuous translation and rotation along the main axis. Twisted shapes belong to this class and two kinds of transformations will result in the same shape: rotations with respect to the main axis and a 180 degree rotation of the main axis.

The notation **A*N*M*R** indicates that the shape has a single main axis **A** from which rotations occur. **N** indicates the number of rotations that will make the shape will be self aligned. **M** indicates the presence of axial mirror planes which will only apply to the initial shape, since any displacements along the axis **A** will result in a rotation. **R** indicates the total rotation, which can be expressed as a parameter or a fixed number. There is no top-down symmetry trough a transversal mirror plane, but a rotation of 180 degrees of the shape and the axis will align the shape with itself.

Twisted shapes belong to this class, which makes them have a direction with respect to the main axis. They can twist in two directions: clockwise or counterclockwise.



3.21 Shape with symmetry A*4*M*R(90) here shown two directions of rotation: clockwise and counterclockwise



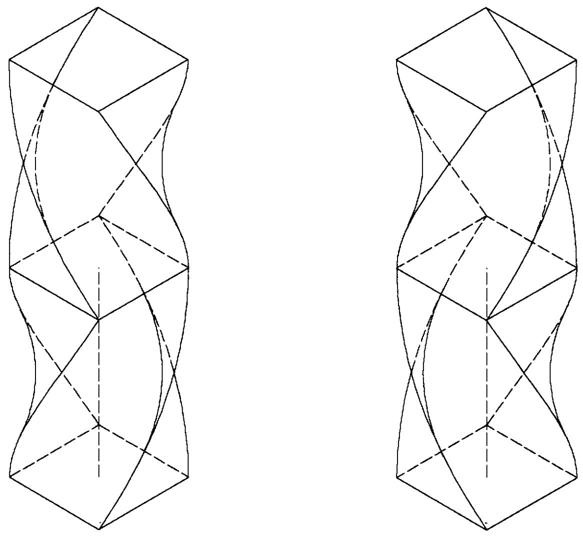
3.22 Shapes with symmetry A*4*M*R

Shapes with continuous single rotation along the rod axis (twist): A*N*M*m*R

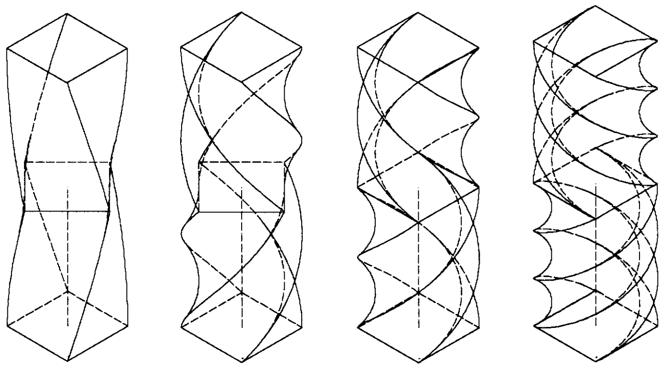
Theses shapes are generated by the continuous translation and rotation along the main axis, with an additional transverse mirror plane.

The notation **A*N*M*m*R** indicates that the shape has a single main axis **A** from which rotations occur. **N** indicates the number of rotations that will make the shape will be self aligned. **M** indicates the presence of axial mirror planes which will only apply to the initial shape, since any displacements along the axis **A** will result in a rotation. An **m** indicates that there is a top-down symmetry along a transverse mirror plane located at the middle point of the main axis. **R** indicates the total rotation, which can be expressed as a parameter or a fixed number.

Twisted shapes belong to this class, which makes them have a direction with respect to the main axis. They can twist in two directions: clockwise or counterclockwise.



3.23 Shape with symmetry A*4*M*m*R(90) here shown in



3.24 Shapes with symmetry A*4*M*m*R

Shapes with continuous double rotation along the rod axis (double twist): A*N*M*RR

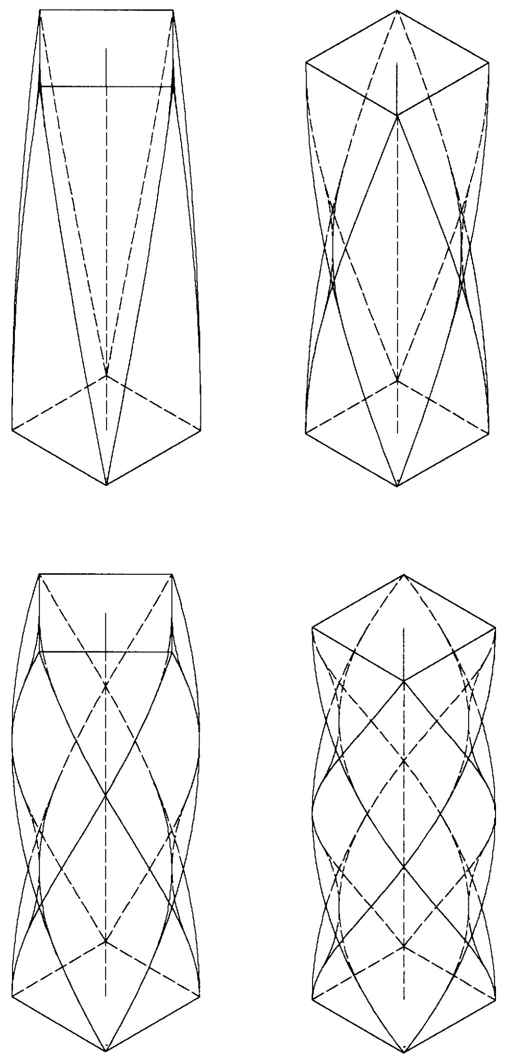
This is an interesting class of shapes since they can have more than one description. The symmetry description of the resulting shape can be different from the description of how the shape is generated. In this case I am inclined to give preference to the way the shape is generated over the description of the resulting shape.

Just like the generation process of the single rotation shapes, double rotated shapes are the result of a shape that is translated along the main axis and at the same time is rotated with respect to the main axis. However one of the shapes is rotated in a positive direction and the other shape is rotated in the negative direction; creating a clockwise shape and a counterclockwise shape.

The notation **A*N*M*RR** indicates that the shape has a single main axis **A** from which rotations occur. **N** indicates the number of rotations that will make the shape will be self aligned. **M** indicates the presence of axial mirror planes if the initial shape has dihedral symmetry. **RR** indicates that there are two rotations one positive and one negative. One important distinction that will be presented later is that two shapes can involve Boolean operations.

There is no top-down symmetry trough a transversal mirror plane. However, if the rotation and counter-rotation go through more than one cycle where the two shapes are aligned, the resulting shape will have emergent top-down symmetry. This will make a distinction on the description of the shape, which never had a transverse mirror plane **m** to begin with.

Another interesting feature of these kinds of shapes is that a **left-right** symmetry will emerge as a result of the equal values of the two opposite rotations, just as if **M** planes existed before.



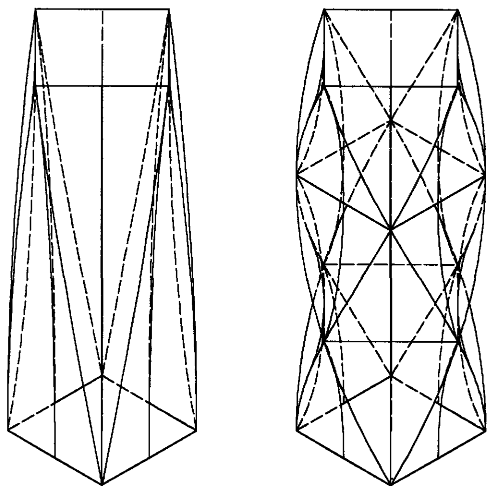
3.25 Shapes with symmetry A*4*M*RR

Shapes with continuous double rotation and Boolean addition: $A^*N^*M^*RR_{(ba)}$

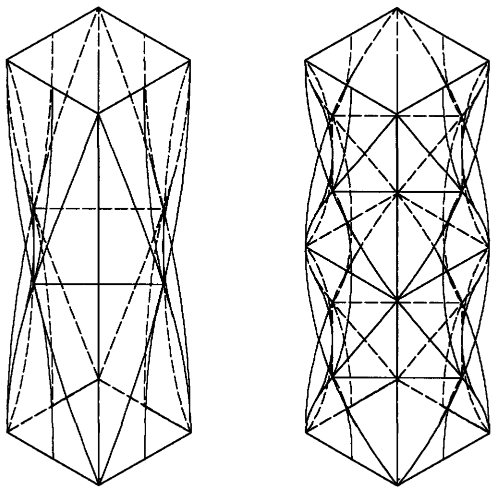
In computational geometry, when two shapes that overlap are involved, there are three Boolean operations that can be performed. A Boolean addition is the first of these three operations. In this particular sub-class, two shapes with opposing rotations will be added to form a final shape.

The notation **A*N*M*RR** indicates that the shape has a single main axis **A** from which rotations occur. **N** indicates the number of rotations that will make the shape will be self aligned. **M** indicates the presence of axial mirror planes if the initial shape has dihedral symmetry. **RR** indicates that there are two rotations one positive and one negative, both of equal magnitude. The **(ba)** subscript will indicate that a Boolean addition will complete the operation.

Some emerging properties could include dihedral symmetry planes **M,** and transverse planes **m** if rotations align more than once.



3.26 Shapes with symmetry A*4*M*RR



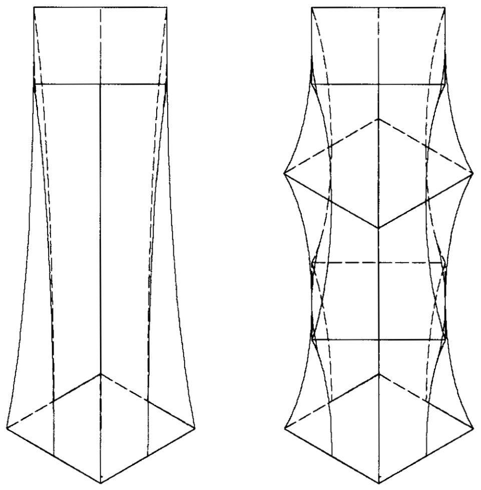
3.27 Shapes with symmetry A*4*M*RR and Boolean addition.
Note the emergent to-down symmetry.

Shapes with continuous double rotation and Boolean intersection: A*N*M*RR$_{(bi)}$

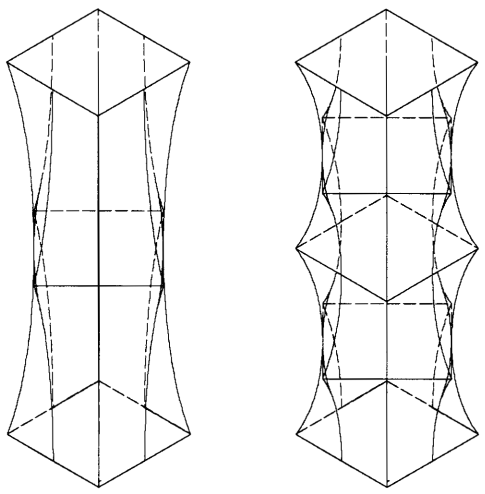
This particular subclass is similar to the previous but instead of a Boolean addition, it involves a Boolean intersection. A Boolean intersection is the resulting shape that contains only the common area of the two opposite rotating shapes.

The notation **A*N*M*RR** indicates that the shape has a single main axis **A** from which rotations occur. **N** indicates the number of rotations that will make the shape will be self aligned. **M** indicates the presence of axial mirror planes if the initial shape has dihedral symmetry. **RR** indicates that there are two rotations one positive and one negative, both of equal magnitude. The **(bi)** subscript indicates a Boolean intersection that completes the operation.

Just like the shapes with Boolean additions, emerging properties could include dihedral symmetry planes **M,** and transverse planes **m** if rotations align more than once.



3.28 Shapes with symmetry A*4*M*RR and Boolean intersection.



3.29 Shapes with symmetry A*4*M*RR and Boolean intersection. Note the emergent top-down symmetry

Chapter 4

Gaudi Columns

Chapter summary

This chapter presents a case study on the columns of the Sagrada Familia and shows how Design Procedures are applied to the generation of the family of columns of the Sagrada Familia plus an infinite number of new designs.

The Sagrada Familia

Located in Barcelona, Spain, The Expiatory Temple of the Sagrada Familia was designed by the Catalonian Architect Antonio Gaudi between 1883 and 1926. Gaudi worked for 43 years in the temple and transformed what was to be a neo-gothic church into a masterpiece of architecture with no precedents.

During this period, Gaudi developed a unique language based on the application of simple rules to form complex geometry, and the use of three ruled surfaces: The helicoids, the hyperboloid and the hyperbolic paraboloid.

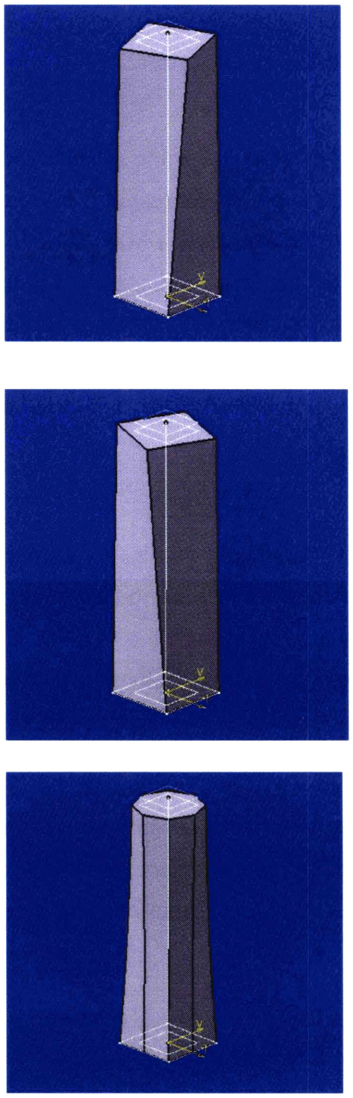
The singular character of Gaudi's architecture in the Sagrada Familia represents the synthesis of his observations of nature translated into geometrical abstractions

4.1 Started in 1883, Gaudi transformed the initial neo-gothic style of the original design into a larger design of complex nature which remains unfinished 123 years after.

The columns of the Sagrada Familia

Gaudi initially proposed a helicoidally shape for the columns, like the salomonic columns from the renaissance. However, he considered that the single twist was visually inappropriate, since it produced the perception of a weak column that could be squashed or deformed. The visual imperfection of the single twist column bothered Gaudi for a number of years, until he resolved to use a double rotated technique where two opposite twisting columns will cancel each other. This allowed the visual asymmetries of the single twisting column to disappear.

Gaudi's novel solution consisted in the use of two opposite rotations of the same shape, once clockwise and another counter-clockwise, and keeping the common parts of the two volumes, like in a Boolean intersection. This novel solution, which has no precedents in architecture, was used to generate the shapes of all the columns in the interior of the Temple. It, is the result of two continuous years of work and experiments of Gaudi's interpretation of the helicoidally growth present in trees and plants.



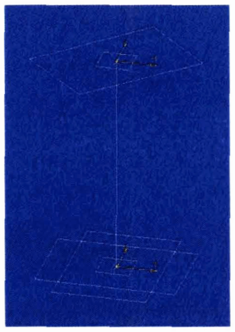
4.2 Generation method of the Sagrada Familia columns. Here shown the generation of the column of 4. A square shape is rotated in two opposite directions by 22.5 degrees. The Boolean intersection of the superimposed shapes produces the column of four

Design Procedure

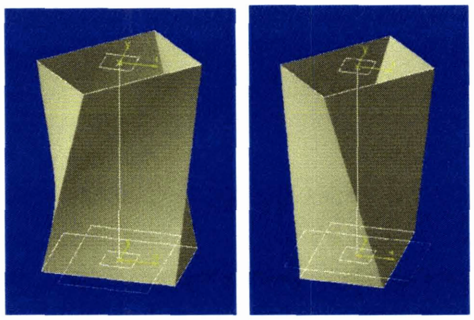
The work started by the reconstruction of the columns knots of the lateral nave of the Sagrada Familia. The rectangular knot was selected as the main model for the parametric exploration. The first challenge was to find a suitable modelling procedure that will yield an accurate representation of the knot. After a series of experiments, I found that using a bottom (initial) and top (final) shapes of the knot, and filling the space in-between with a surface fitting function, the resulting form will generate a shape that was visually equivalent to the original plaster model by Gaudi.

The first stage was to create the top and bottom figures in a wireframe model, which is called the parametric skeleton. A surface fitting function was applied to each pair of top-bottom shapes producing both the rotation and counter-rotation shapes. The two generated shapes were superimposed and used to perform the Boolean intersection that generates the original shape.

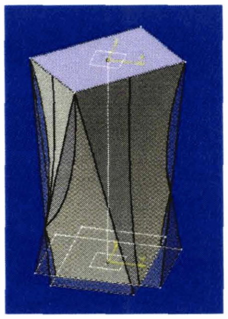
Although this procedure of blending between two pairs of shapes was not described by Gaudi, nor any other researchers and scholars, the resulting columns were not only geometrically accurate, but also visually correct when compared to the original Gaudi models.



4.3 Parametric Skeleton. Here showing two pairs of initial shapes, one for the rotation and the other for the counter-rotation.



4.4 Surface fitting functions create the rotation and counter-rotation shapes.



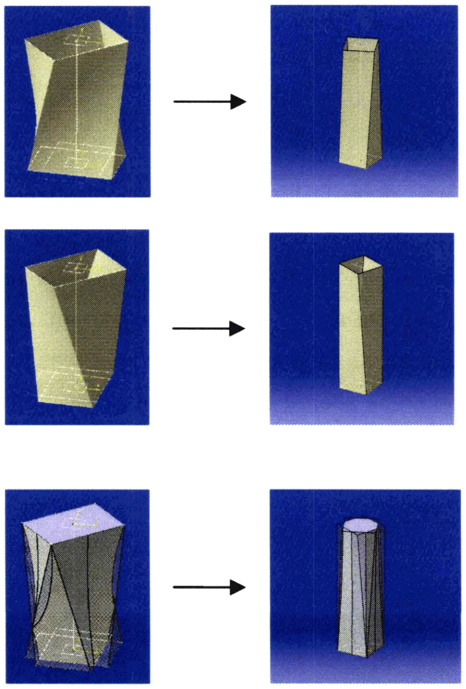
4.5 Boolean intersection of the two superimposed shapes generates the rectangular knot.

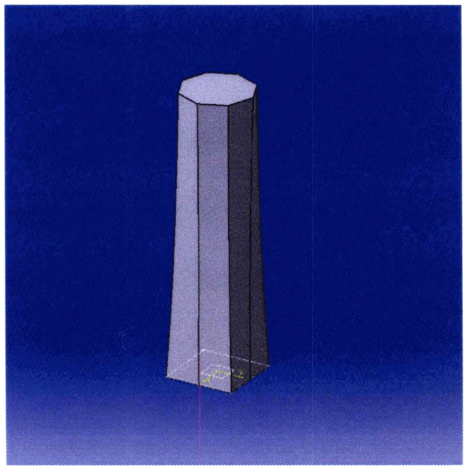
Parameterization of the column

In the parametric skeleton there are three types of geometrical components: the axis of the column represented by a line, two parallel planes where the top and bottom shapes will be located, and the top and bottom shapes. Each surface procedure is composed of two initial shapes one on the top and one for the bottom, for a total of 4 initial shapes. The parameterization schema only constrains the location of the initial shapes to the top and bottom planes. The planes must be normal to the axis line. The shapes are not constraint and are free to take any kind of geometrical and topological transformations.

The parametric model allows for variations on the values of the main axis as well as the dimensions of the initial shapes, and the rotation and counter-rotation angles corresponding to the opposite rotated shapes.

If the two shapes are squares and the angles are 22.5 degrees, the result will be a square column. If the two shapes are changed into rectangles, and the values of rotation and counter-rotation is 45 degrees, then the rectangular knot is generated.



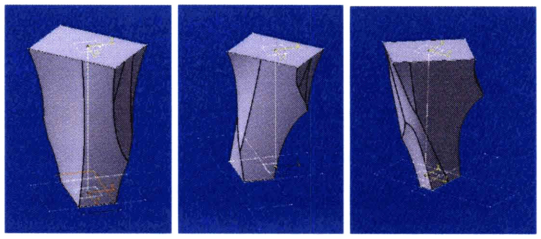


4.6 Transformation of the rectangular knot into the square column done by changing the initial shapes from rectangles to squares, and the angle of rotation, from 45 degrees to 22.5.

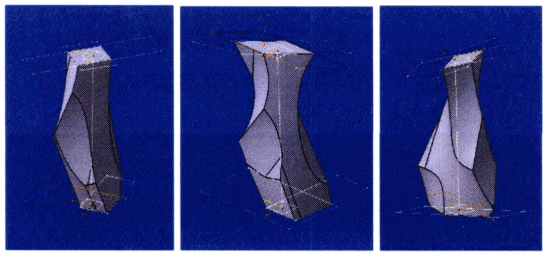
Transformations on the columns

The first set of transformations was done to the top and bottom shapes, starting with variations of the proportions of the lower rectangles, to variations on the angles and finally with variations on the four initial shapes. The height of the column, as well as the rest of the parameters remained unchanged through these set of operations. An important discovery was that the topology of the final column would be altered as a result of changing the parameters of the initial shapes, even though the topology of the all the geometrical components of the model remained unchanged.

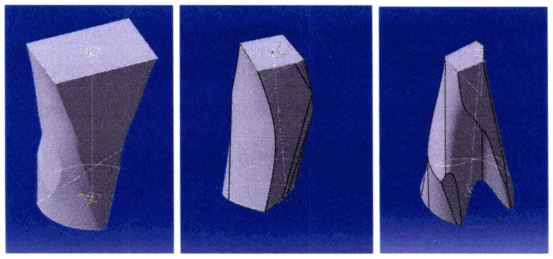
Different variations of the design obtained from the same parametric model. Other set of transformations included changes in the topology of the primitive shapes. The parametric model allowed topological changes and still maintained the integrity of the surface fitting procedures without breaking the model, or causing geometric problems.



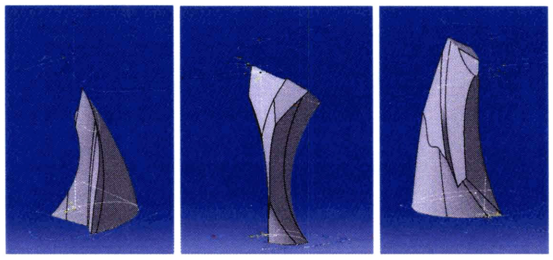
4.7 Parametric variations of the lower shapes



4.8 Parametric variations of lower and upper shapes



4.9 Topological transformations of shapes



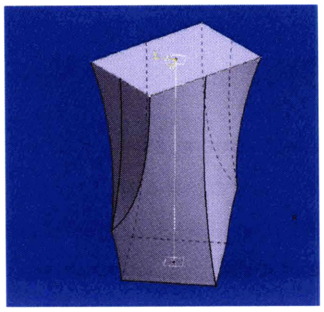
4.10 Topological transformations and displacement of the initial shapes

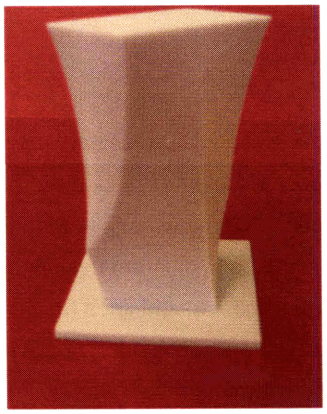
Comparison with original designs

A natural question for a generative procedure of any kind is to somehow evaluate the produced results. Performance driven processes is one of the most widely accepted methods of evaluations for designs when a specific set of criteria is to be met. Therefore the procedure might iterate between different solutions to find an optimal result. On the other hand, qualitative evaluation of designs is more complex.

Based on this premise, I propose two systems for evaluation of the designs generated by the parametric model. Each evaluation system is based on a general proviso for aesthetic evaluation. The first proviso determines if a design is in the language of the original designs, and the second proviso distinguishes which designs where made by Gaudi and which are new. Each new design is to be evaluated intrinsically with respect of the two evaluation provisos. This kind of evaluation tends to be objective in the sense that the criteria are determined before the instances of the model are generated, therefore there is no discrimination.

The first step was to determine if the procedure was accurate. For this a visual comparison between the original designs and the design procedure was done. Then measurements taken from both models were compared. As a result no visual discrepancies and no measurement differences were found when both models were compared.



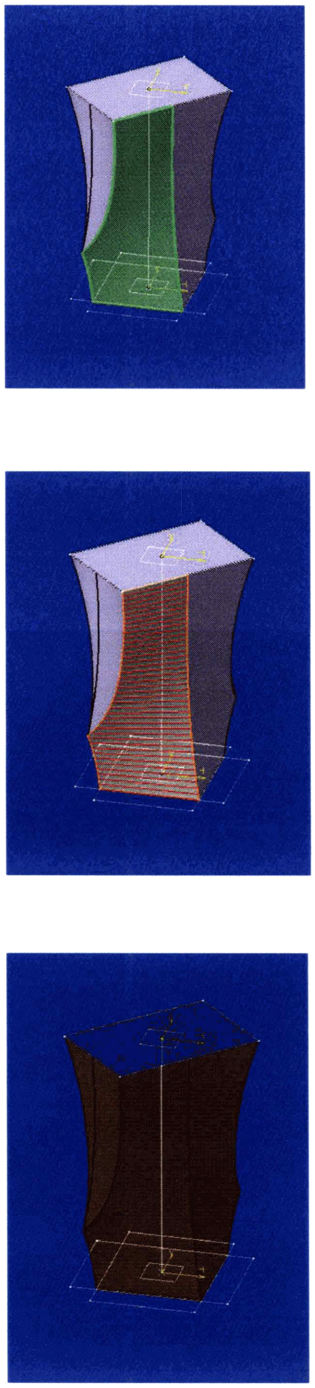


4.11 Visual comparison between the design procedure, the rapid prototype from the computer model and the original design by Gaudi. No visual or measured discrepancies were found between the compared models. This proved that the design procedure is accurate and valid.

Evaluation of the new designs

The evaluation consists in determining if a new design generated from the parametric model is in the language of Gaudi. We define that a shape is in the language of Gaudi if: 1) The shape is generated by the same procedures used by Gaudi; and 2) The generated shapes and the original from Gaudi produce similar results when analyzed. The parametric model is based on the same procedure that Gaudi used to generate the original columns therefore it can be assumed that all shapes generated by the parametric model are in the language.

The second test is to determine the nature of the resulting shapes and compare them with the original columns. The original columns made by Gaudi consisted in rotating shapes that produced ruled surfaces. It is widely know that Gaudi made extensive use of ruled surfaces in different designs. A ruled surface is defined by a curved shape formed with straight lines (ruling lines) and that has a non-positive Gaussian curvature when is analyzed. A sample of the generated shapes were tested to determine if they where made with ruled surfaces. All the generated shapes that where tested showed that they had surface continuity in each face of the shape , all the surfaces had ruling lines making them ruled surfaces such as the original Gaudinian designs, and finally the analysis of Gaussian curvature was either negative value or zero.

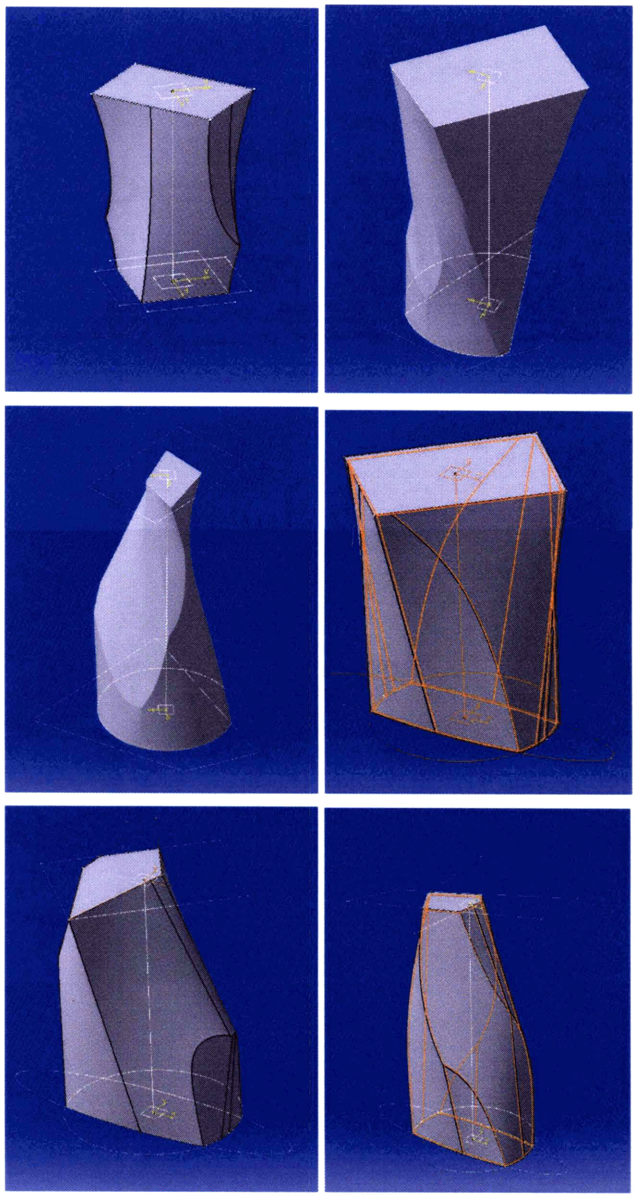


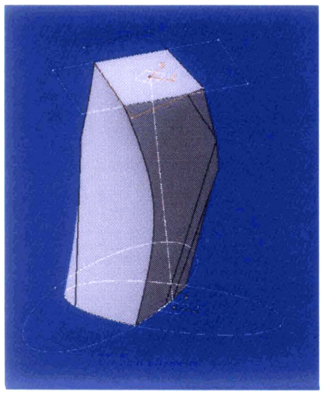
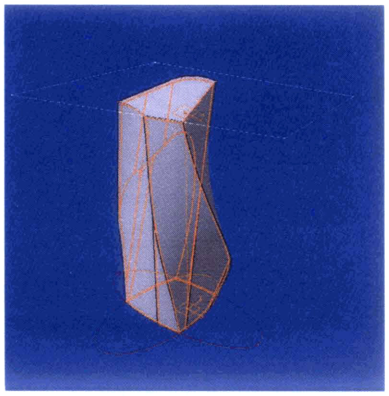
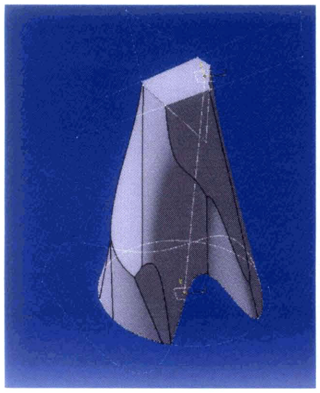
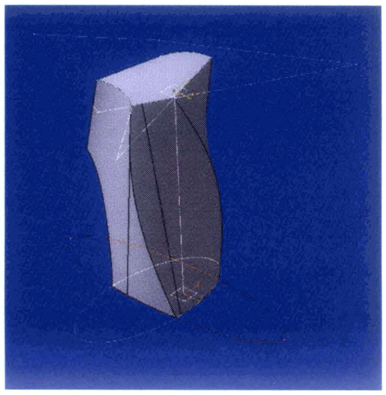
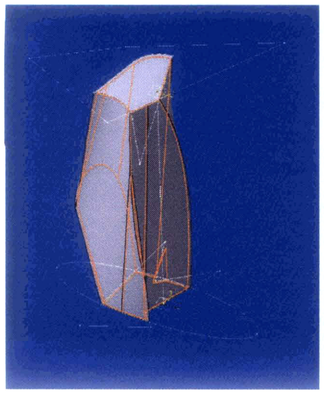
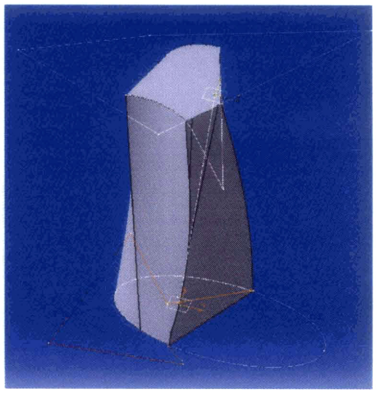
4.12 Evaluation of the rectangular knot showing the three steps for evaluation: 1) Surface continuity; 2) presence of ruling lines; and 3) Gaussian curvature analysis.

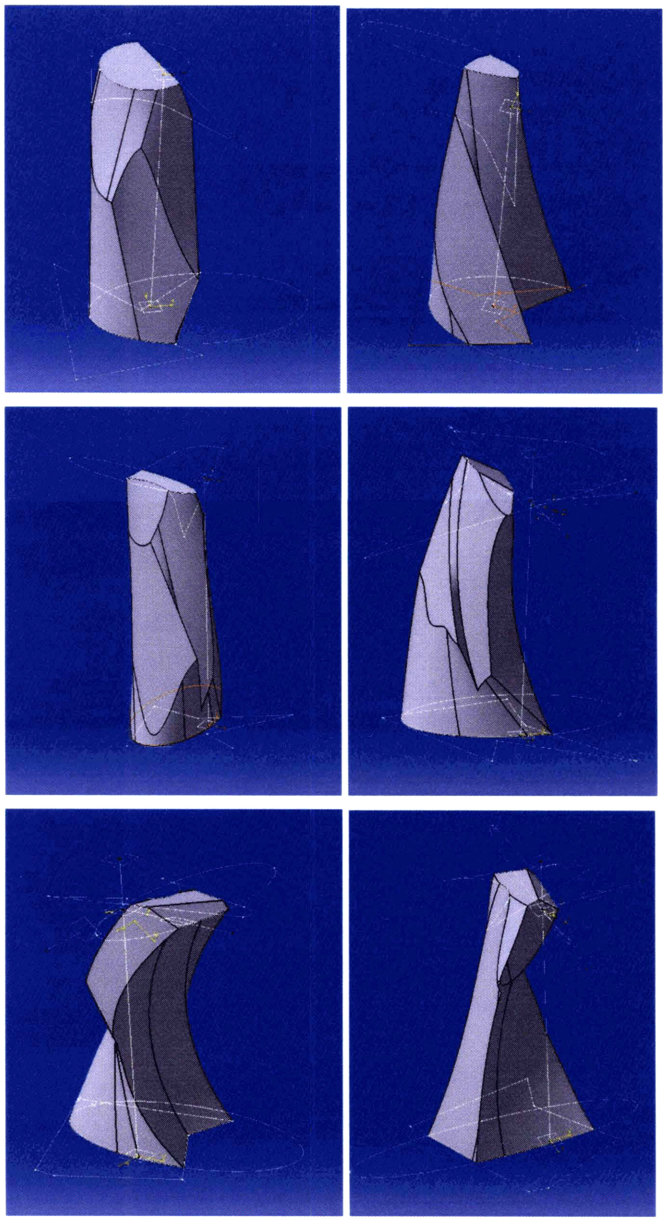
An infinite catalog of columns

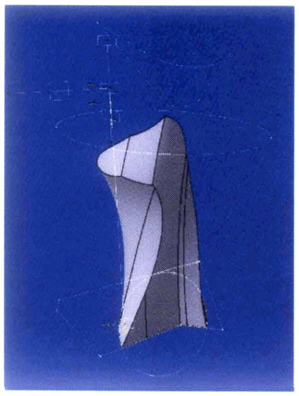
The design procedure allowed the generation of all the original design by Gaudi plus an infinite number of new designs. All design instances were directly generated by the design procedure with little effort. While the parametric variations produced interesting results, some of them very similar to the original design, the most unexpected yet stunning where produced when the initial shapes where changed.

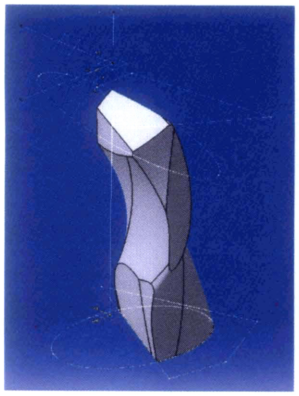
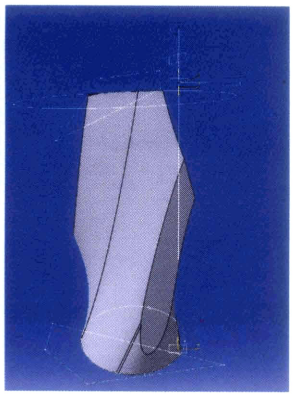
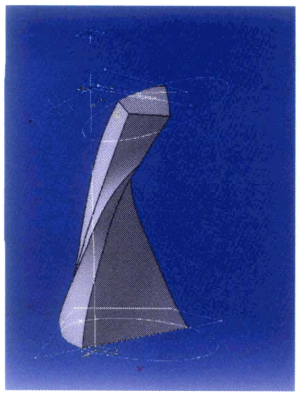
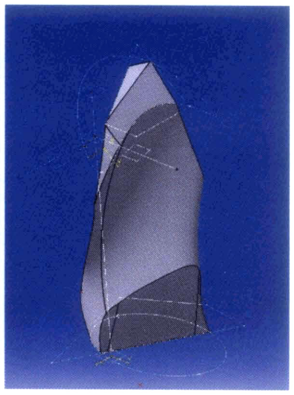
The initial shapes included closed regular and irregular polygons, polylines, splines, and any combination of them. Two important restrictions on the initial shapes must be considered: 1) the initial shapes must be closed shapes; and 2) the initial shapes must not be self intersecting shapes. Any contravention of the aforementioned descriptions will result in a non-compliance of the procedure, since the Boolean intersection requires two closed and well formed shapes. Since any closed shape can be used as one of the initial shapes, the possible number of designs that can be generated goes to infinity.

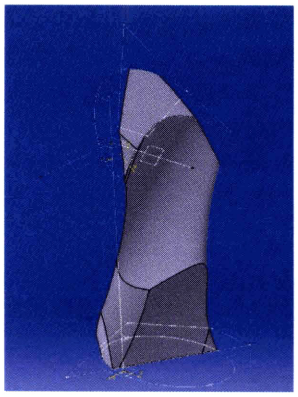
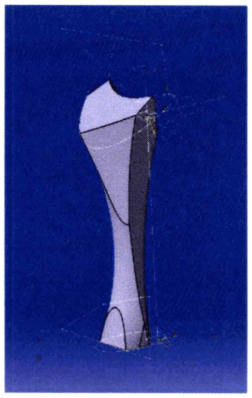
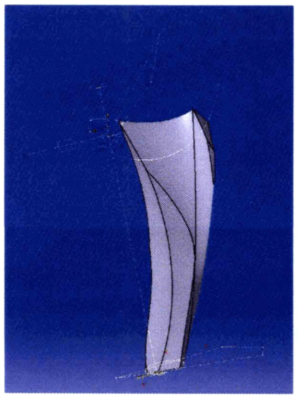


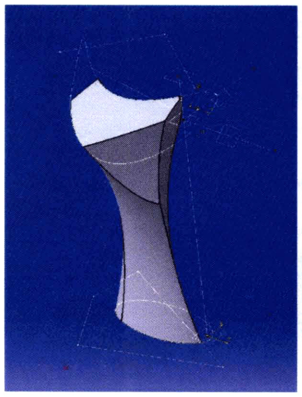
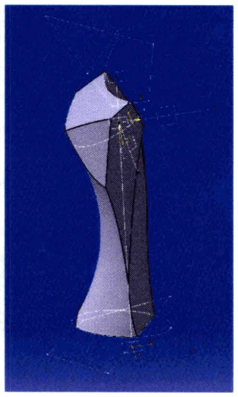







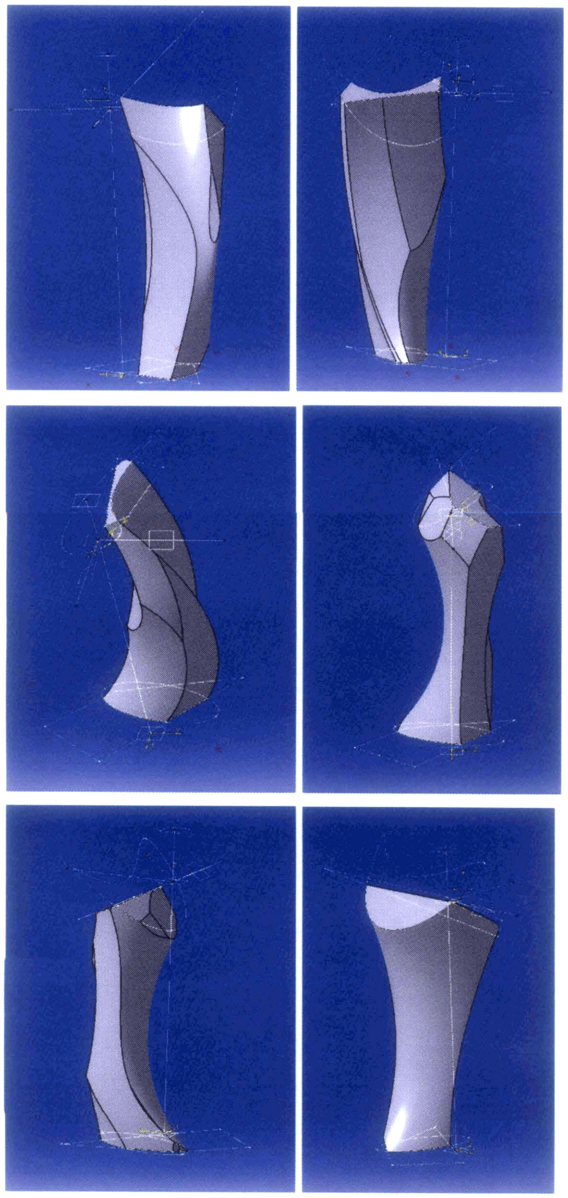


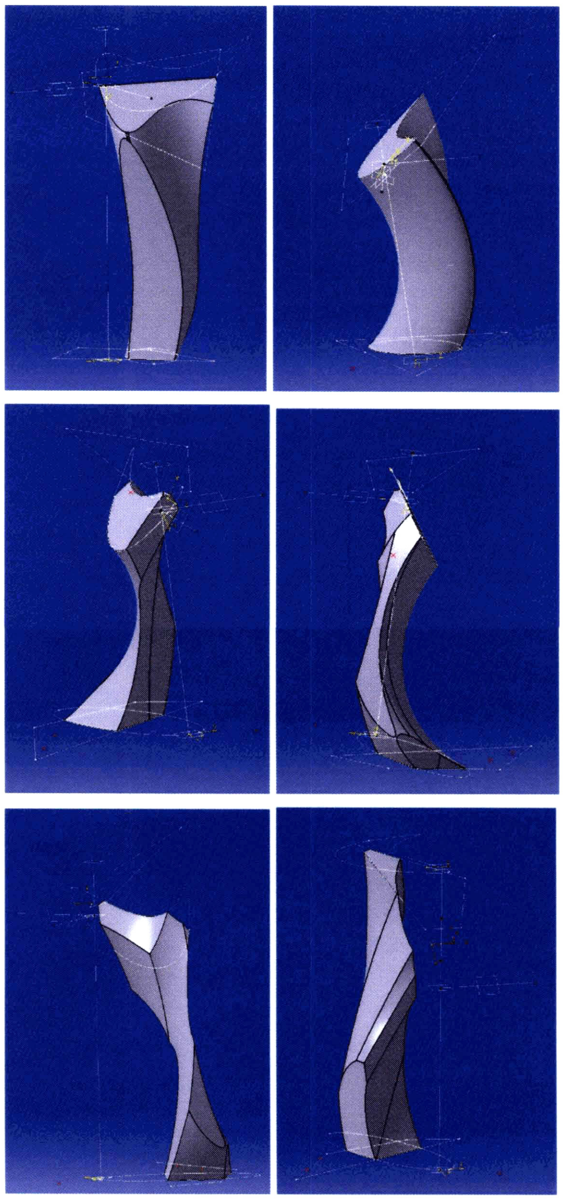


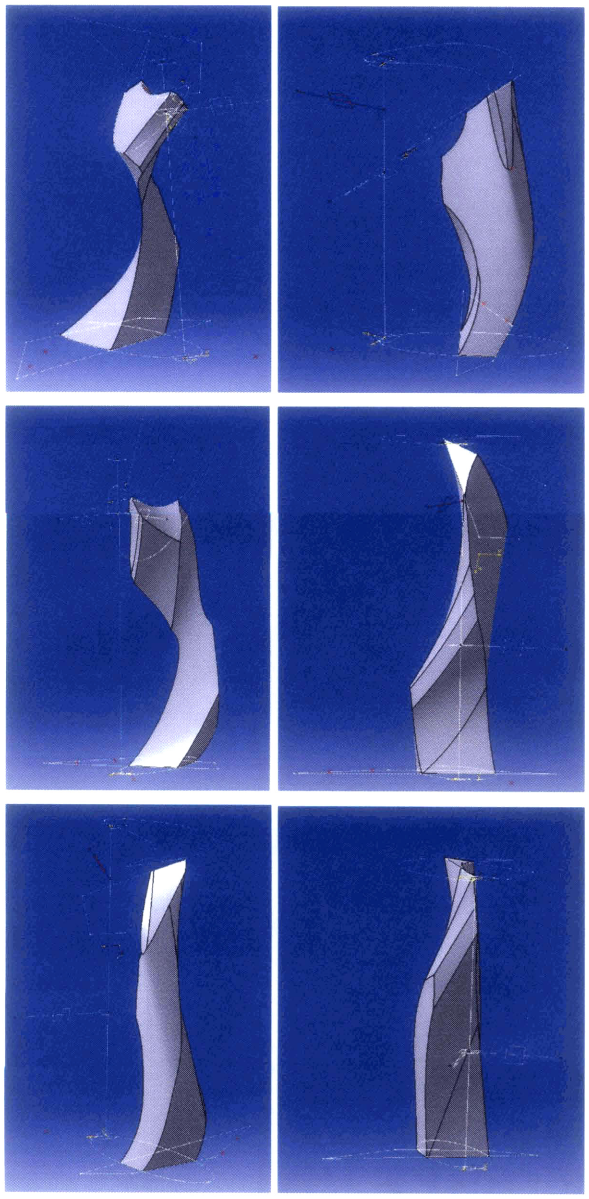





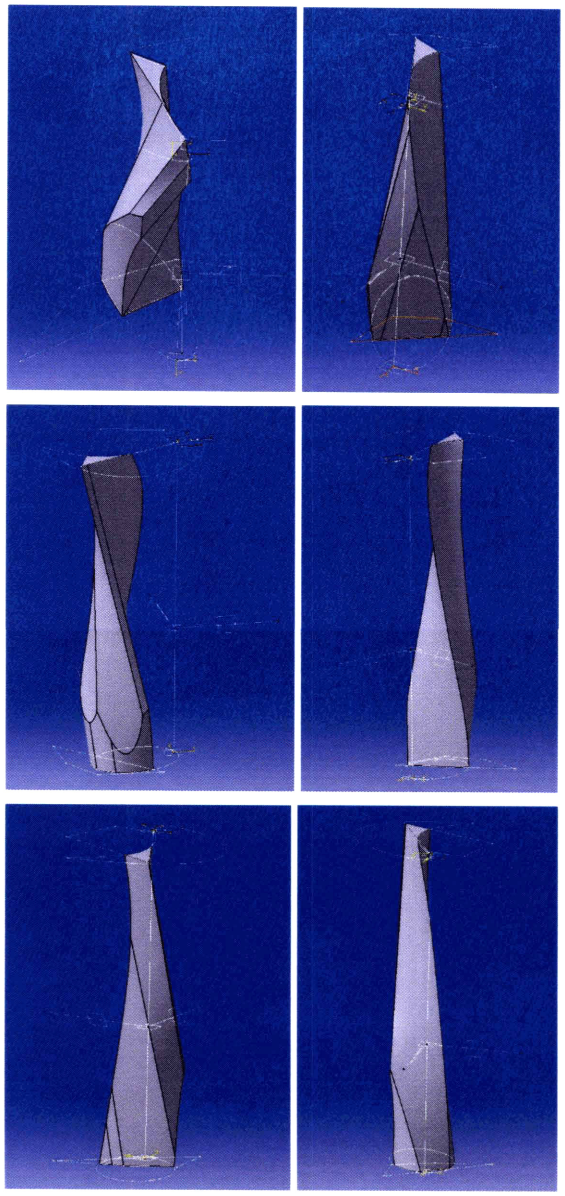




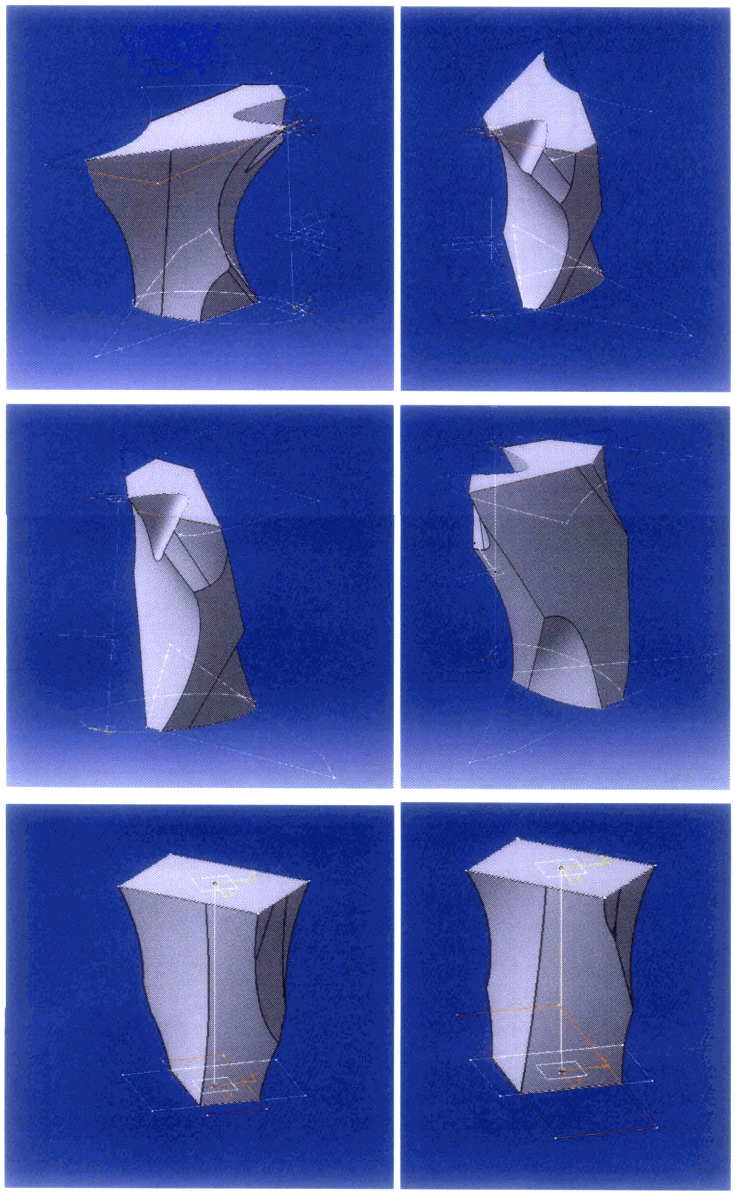



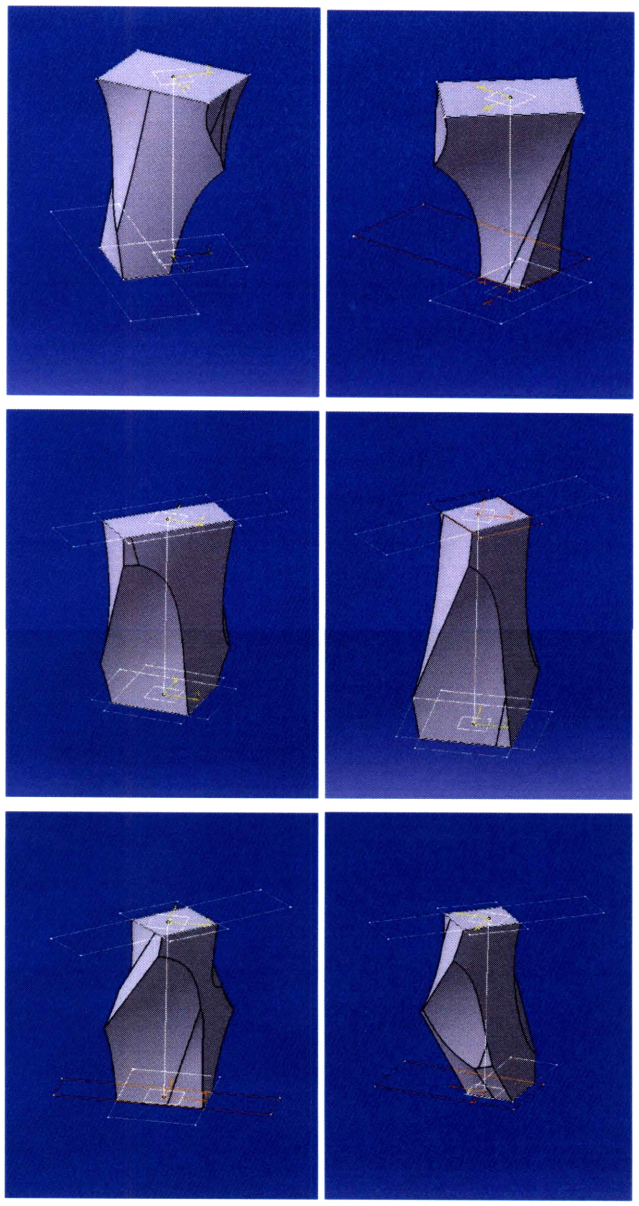


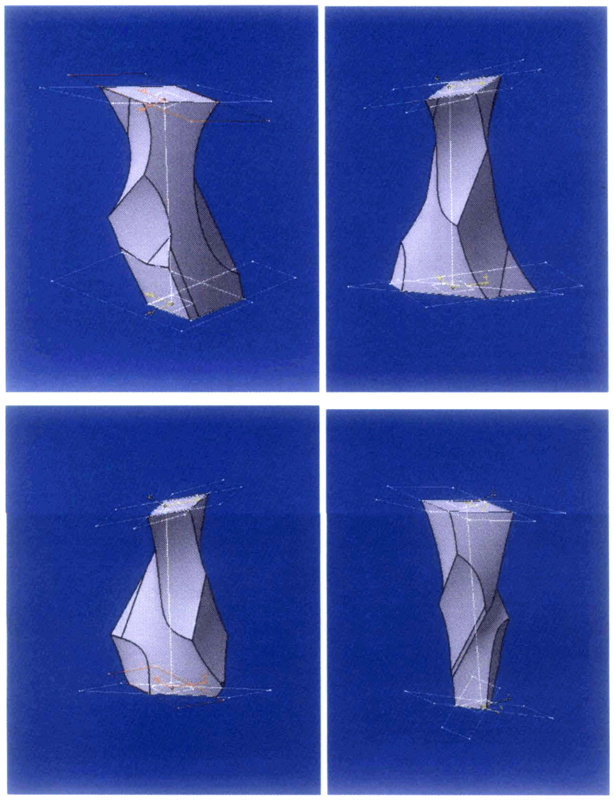












Chapter 5

Generative Symmetry

Chapter summary

This chapter expands on the knowledge of 3D symmetry group's descriptions and presents a design procedure to generate complex shapes based on single rod symmetry. The design procedure also generates a description of the symmetry group.

Ambiguity of 3D shapes with non regular symmetry

Current classifications of symmetry for 3D shapes have two major setbacks that make them ambiguous. Two or more different shapes can be classified under the same symmetry group, and consequently one shape can be classified in more than one symmetry group depending on how we look at it. This creates ambiguity. The basic problem is that current symmetry knowledge does not distinguish between shapes of single symmetry. It only says that all the shapes have symmetry of 1, therefore no kind of transformations will generate the same shape. However there are distinct features that are visually evident in two different shapes that according to the current conventions of symmetry are from the same symmetry group. This creates a particular problem when we are trying to differentiate between two non-symmetrical shapes.

I propose a model that expands on the knowledge of symmetry groups and provides a classification that distinguishes the most discrete features. When you have regular shapes it is easy to classify in a finite group of symmetry types and leave the ones that don't have any symmetrical features or that don't have any regularity, or that don't have any recognizable features that can be categorized as identical, to put them in a category simply called that they don't have symmetry or that their symmetry is just 1 (identity)

However if we look at symmetry not as the property of the resulting shape, but as the description of the generative process/procedure we should be able to make clear distinctions of two similar shapes., therefore the symmetry of a shape is not a property of the shape but a description of the generative process.

Description of 3D shapes with non regular symmetry

Here is a list of the descriptions of the symmetry groups in 3D:

A → denotes the presence of a single rod axis

N → denotes the symmetry level of the initial shape

M → denotes the presence or not of a mirror plane along the rod axis

m → denotes the presence of a transverse mirror plane normal to the axis

R^+ → denotes the presence or continuous axial rotation

R^- → Denotes the presence of a counter rotation or opposite rotation.

D → Denotes dilation symmetry along the rod axis

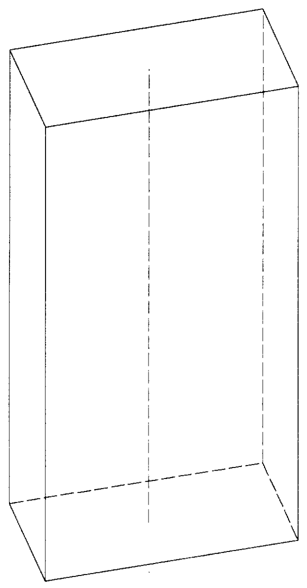
B(a) → Denotes Boolean addition

B(i) → Denotes Boolean intersection

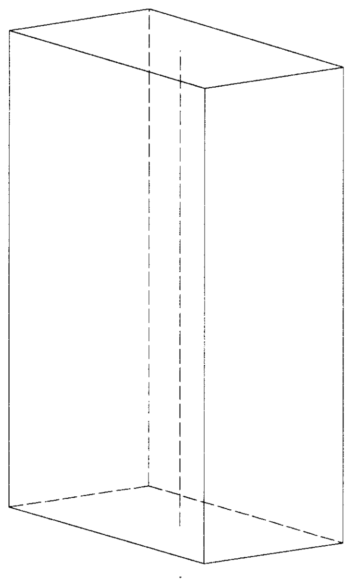
S(i) → Denotes that a shape was generated with a special procedure

Symmetry as generative procedure for design

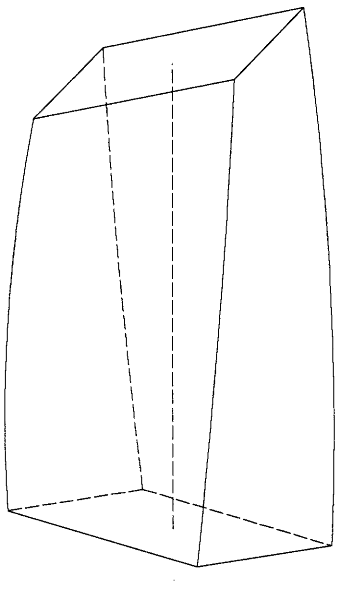
Descriptions can also be used as generative process for design. In the case of the 3D single rod symmetry groups, a description serves for the purpose of determining which operations will be used as parameters and will also indicate the value of the variables. Shapes used as inputs will remain as a separate procedure, therefore it will only be considered as an input. In case of discrepancy between two descriptions, the designer may use whichever he whishes; however I would give preference to the generating symmetry and not the description.

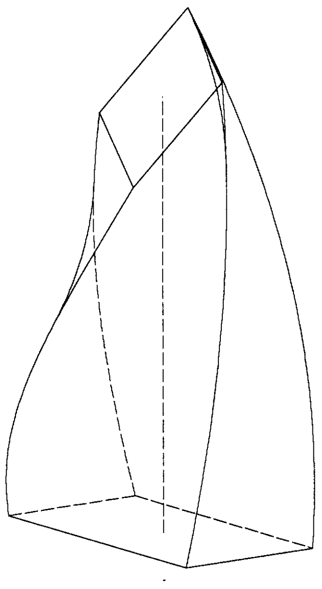


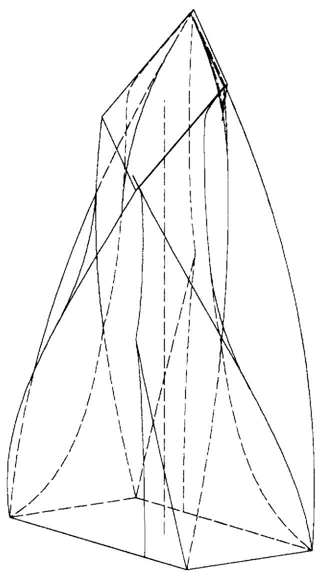
5.1 A2Mm



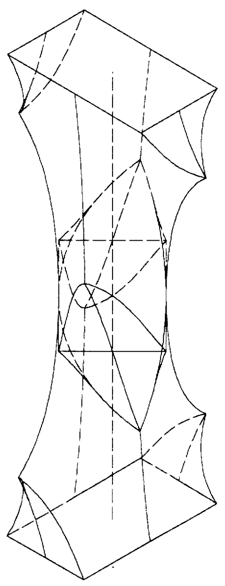
5.2 A2m


5.3 A2R45

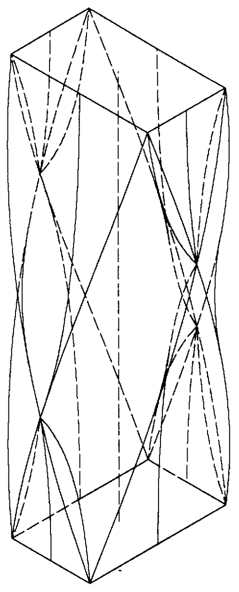

5.4 A2R90



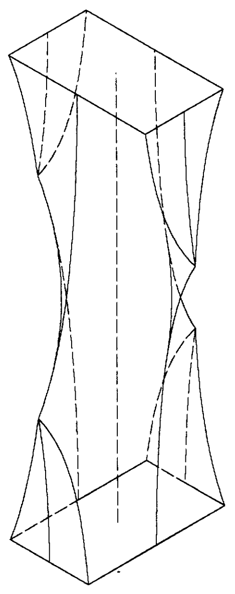
5.5 A2R90r90(ba)



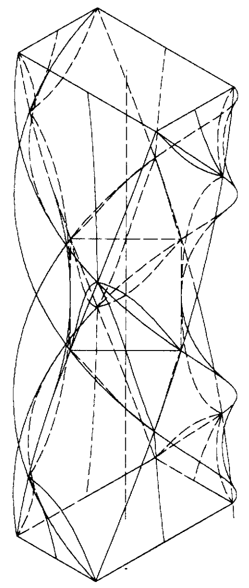
5.6 A2R90r270(bi)



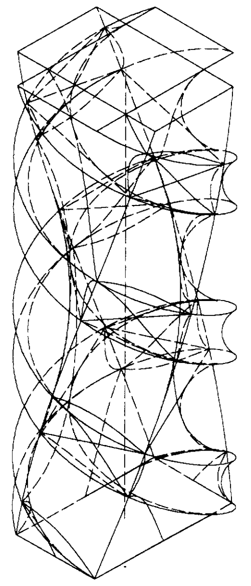
5.7 A2R90r90(ba)



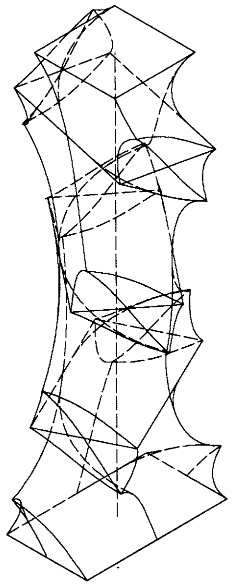
5.8 A2R90r90(bi)



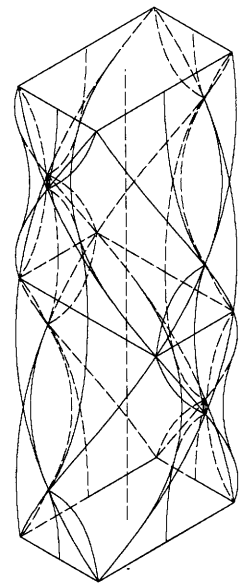
5.9 A2R90r270(ba)



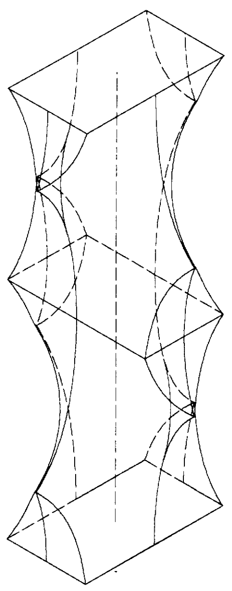
5.10 A2R90r540(ba)



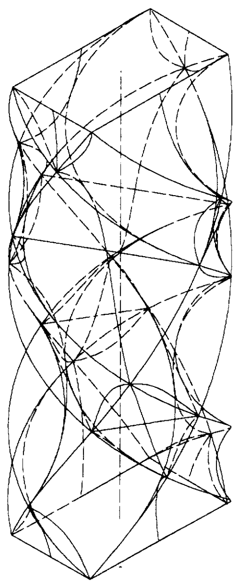
5.11 A2R90r270(bi)



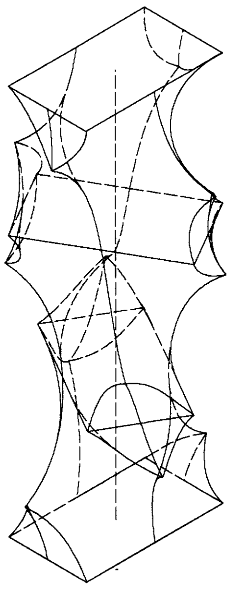
5.12 A2R180r180(ba)

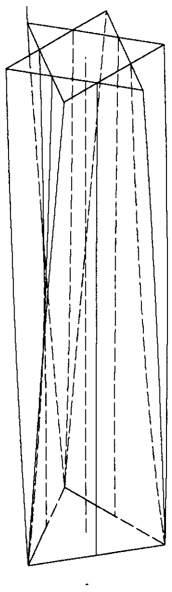


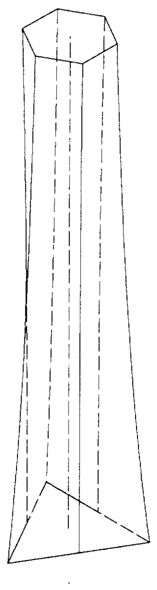
5.13 A2R180r180(bi)



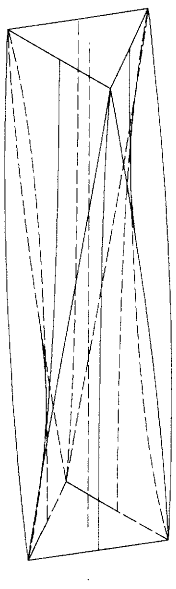
5.14 A2R90r360(ba)


5.15 A2R180r360(bi)

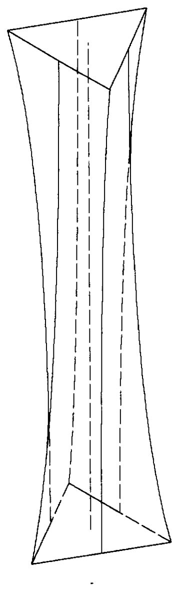

5.16 A3R30r30(ba)



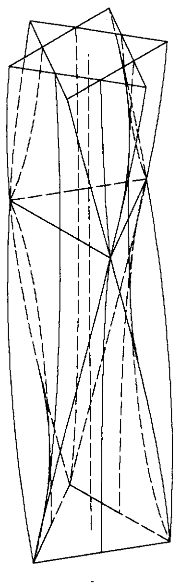
5.17 A3R30r30(bi)



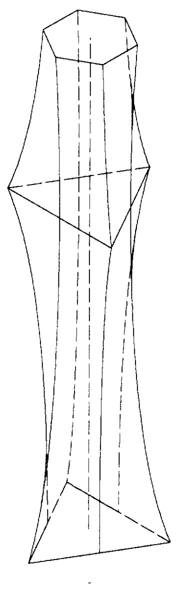
5.18 A3R60r60(ba)



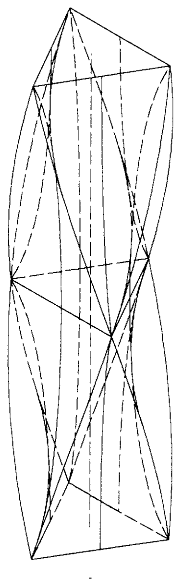
5.19 A3R60r60(bi)



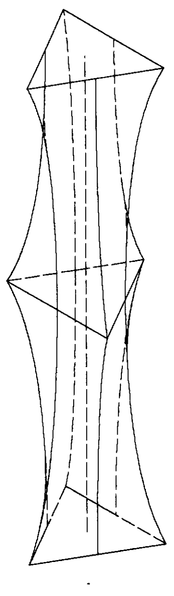
5.20 A3R90r90(ba)

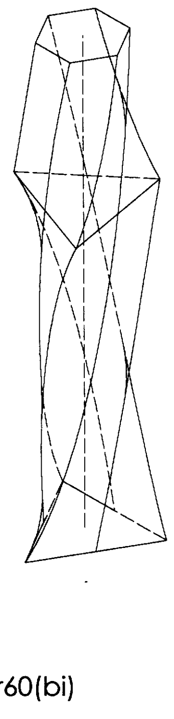


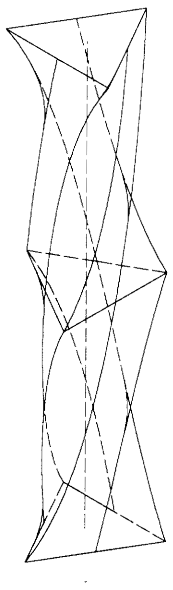
5.21 A3R90r90(bi)

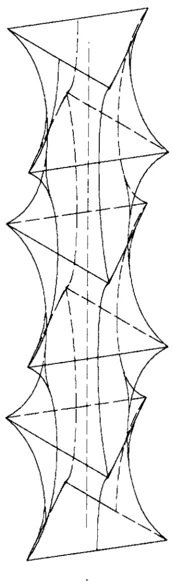


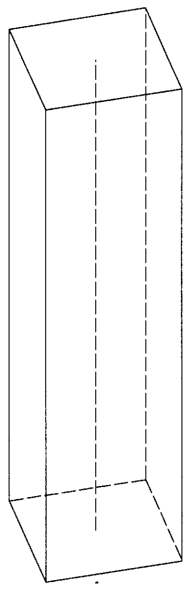
5.22 A3R120r120(ba)


5.23 A3R120r120(bi)

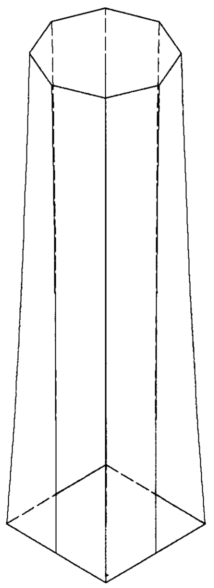

5.24 A3R240r60(bi)


5.25 A3R300r60(bi)

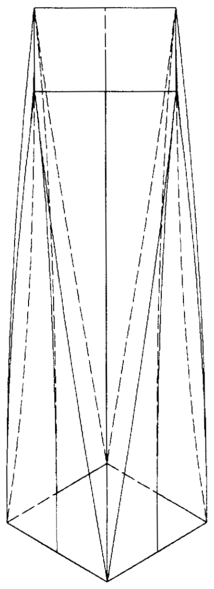

5.26 A3R300r300(bi)



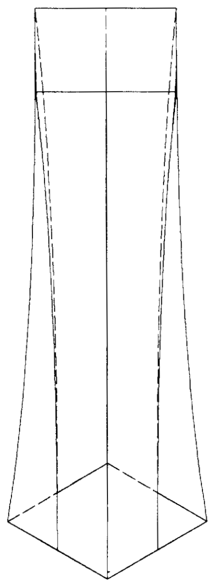
5.27 A4Mm



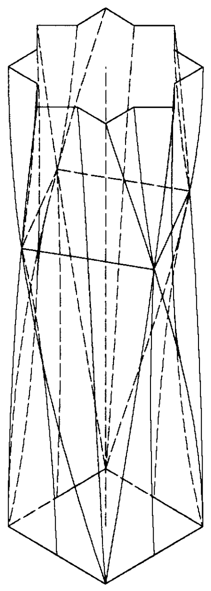
5.28 A4R23r23(bi)



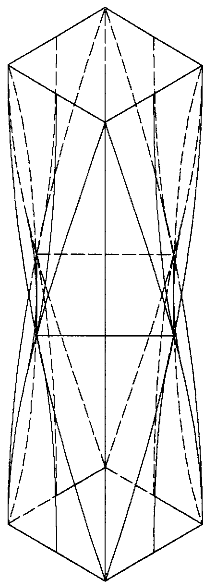
5.29 A4R45r45(ba)



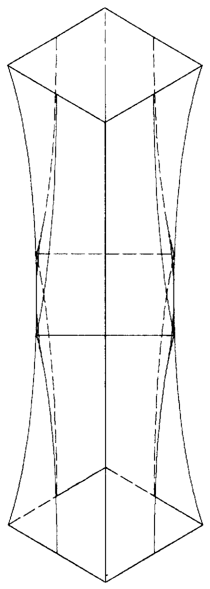
5.30 A4R45r45(bi)



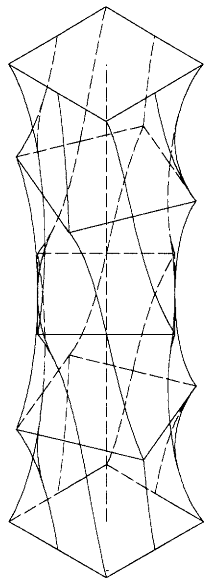
5.31 A4R45r90(ba)



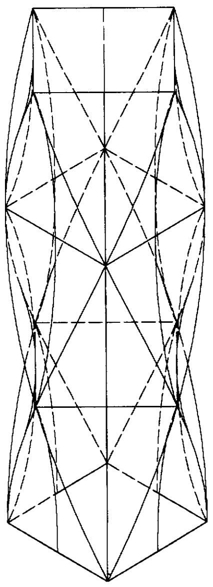
5.32 A4R90r90(ba)



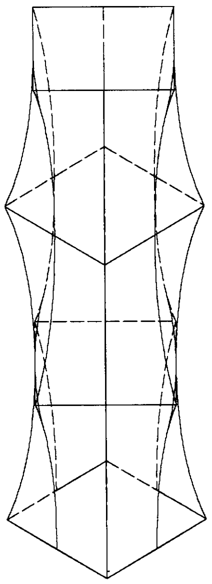
5.33 A4R90r90(bi)



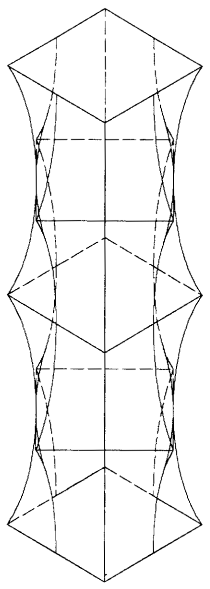
5.34 A4R90r270(bi)



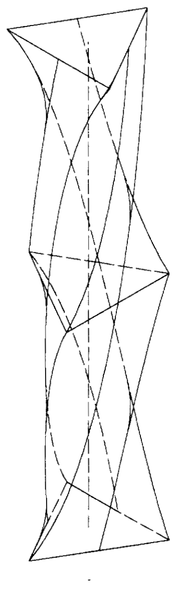
5.35 A4R135r135(ba)



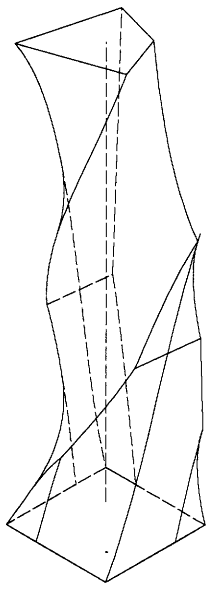
5.36 A4R135r135(bi)



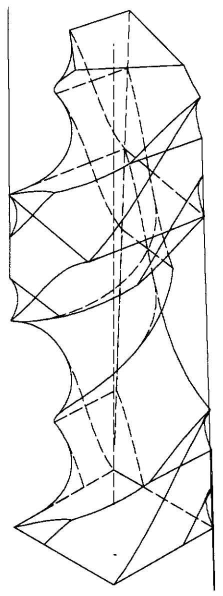
5.37 A4R180r180(bi)



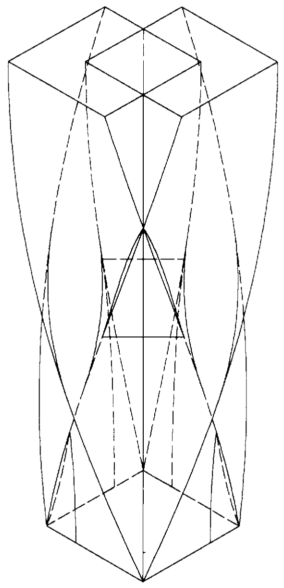
5.38 A3R300r60(bi)



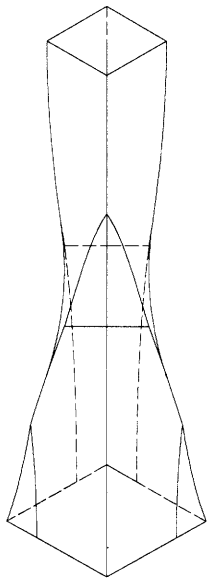
5.39 A(o)4R15r200(bi)



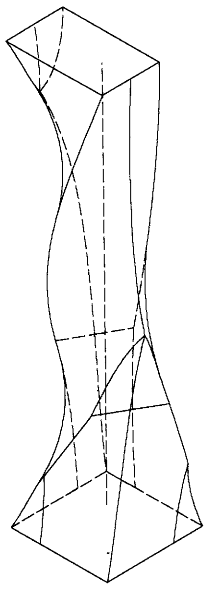
5.40 A(o)4R15r500(bi)



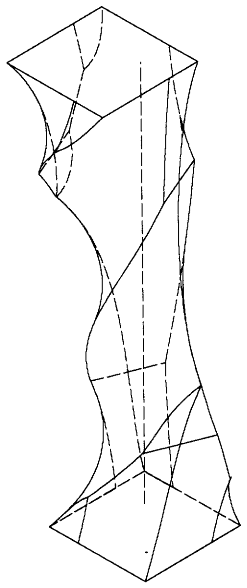
5.41 A(o)4R90r90(ba)



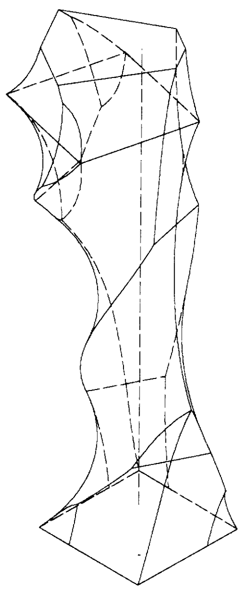
5.42 A(o)4R90r90(bi)



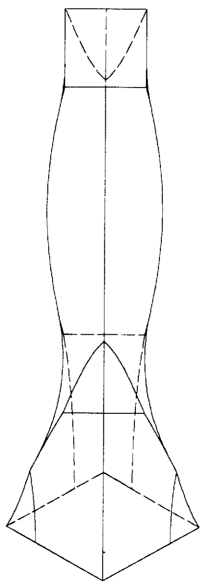
5.43 A(o)4R90r180(bi)



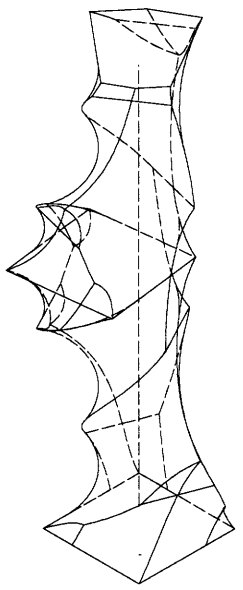
5.44 A(o)4R90r270(bi)



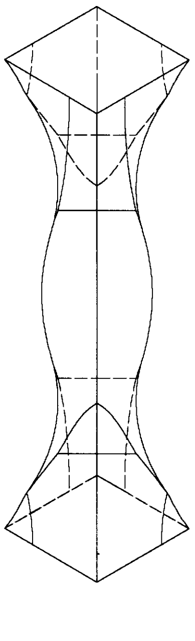
5.45 A(o)4R120r300(bi)



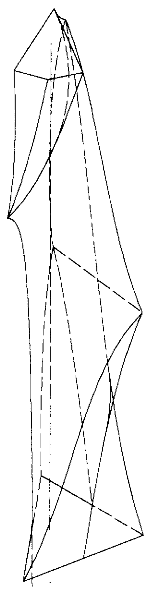
5.46 A(o)4R135r135(bi)



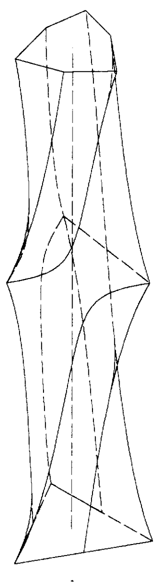
5.47 A(o)4R150r500(bi)



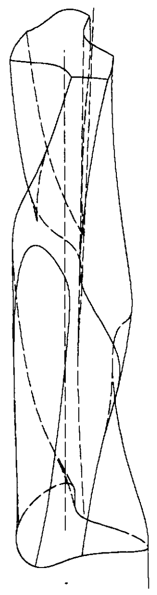
5.48 A(o)4R100r180(bi)



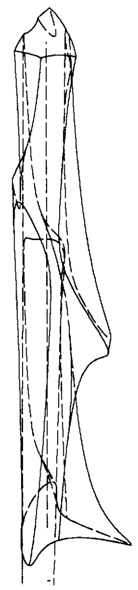
5.49 A(s1)R15r200(bi)



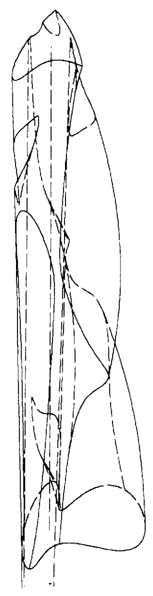
5.50 A(s2) R15r200(bi)



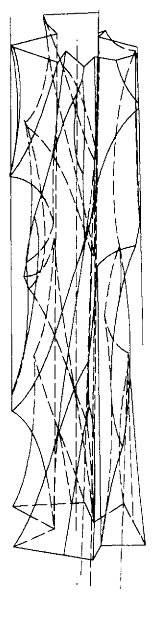
5.51 A(s3) R15r200(bi)



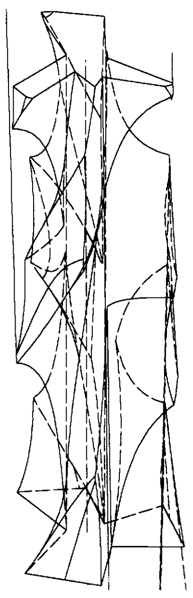
5.52 A(s4) R15r200(bi)



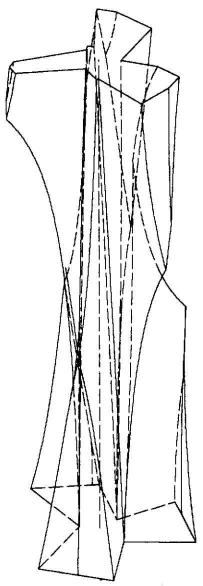
5.53 A(s5) R15r200(bi)



5.54 A(s6) R15r200(bi)

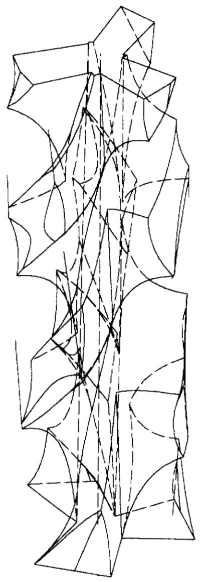


5.55 A(s7) R15r200(bi)

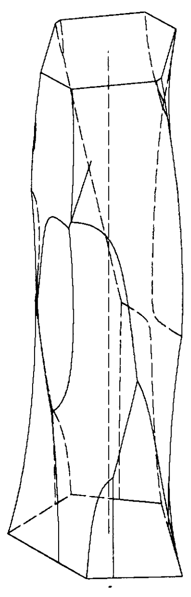


5.56 A(s7) R45r60(bi)

5.57 A(s7) R45r200(bi)


5.58 A(s7) R45r300(bi)

5.59 A(s7) R155r300(bi)



]

5.60 A(s8) R90r90(bi)

Chapter 6

Twisted Towers

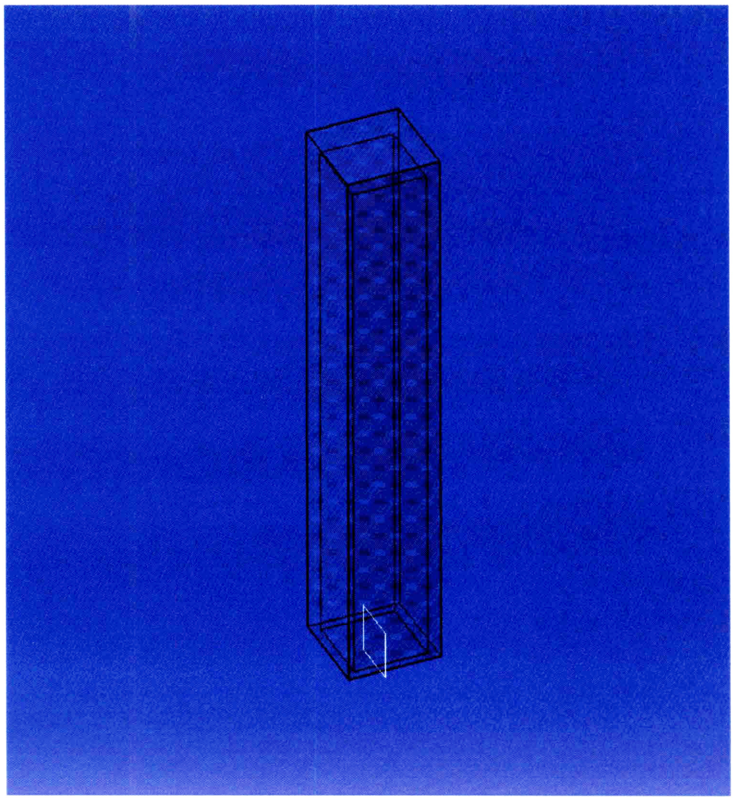
Chapter summary

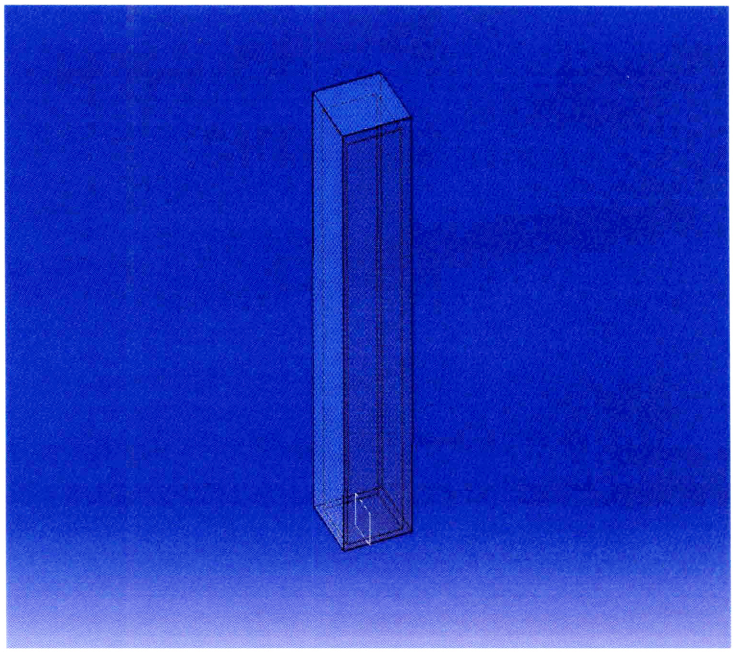
This chapter presents a case study on the generation of twisted shapes for high-rise buildings.

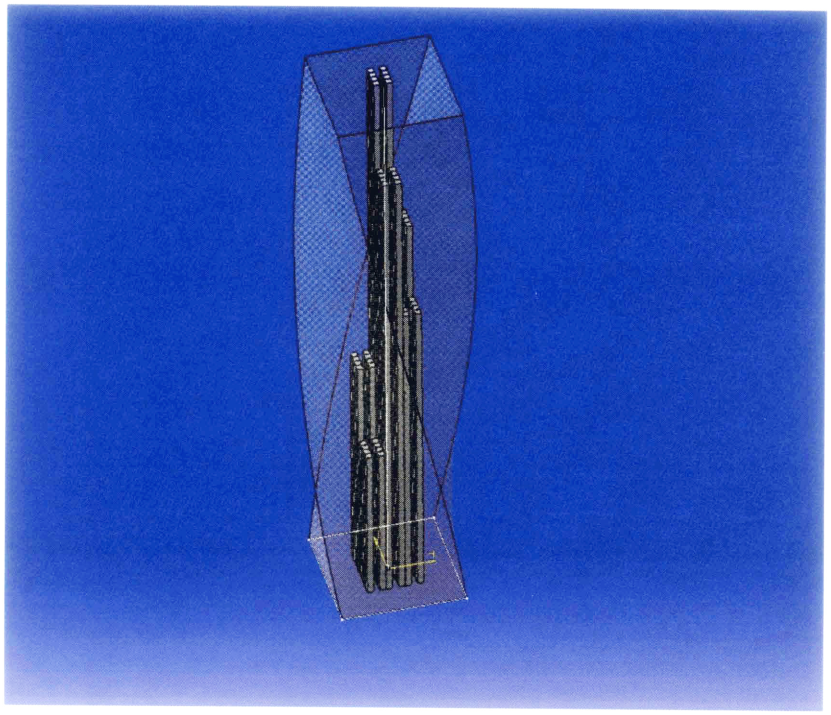
Twisted Towers

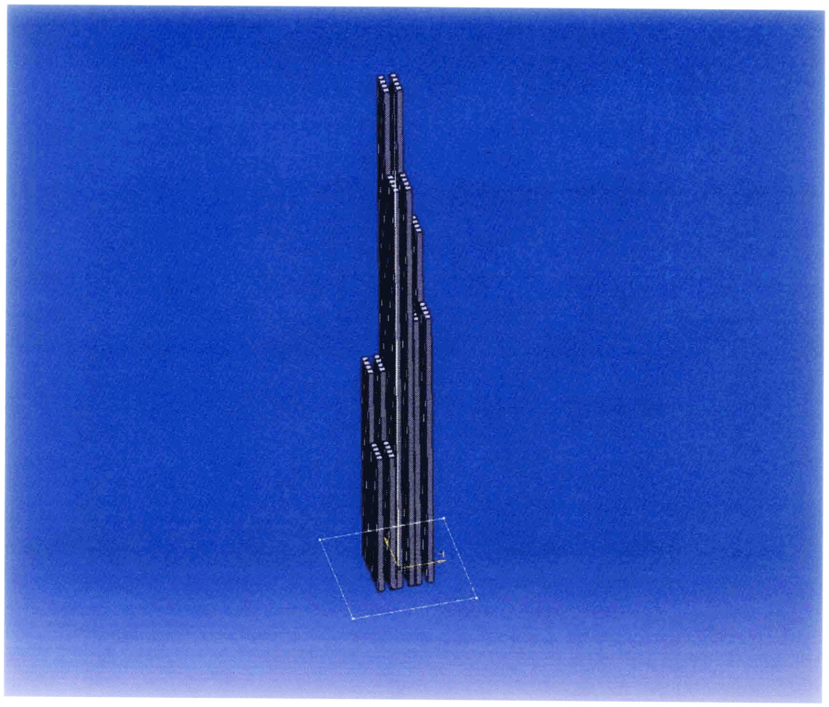
Twisted towers present a series of examples of the use of Design Procedures for the generation of twisted high-rise buildings.

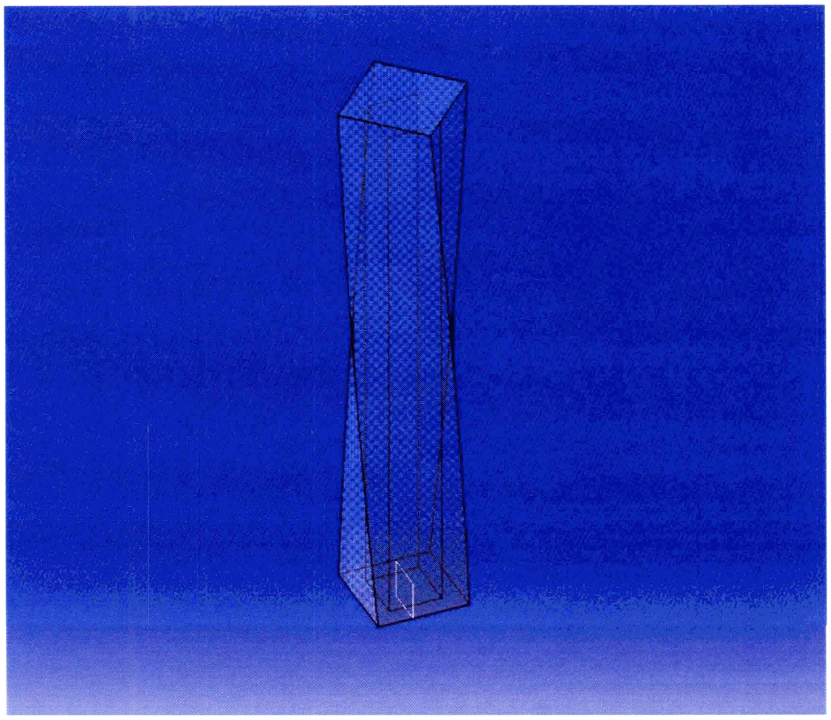
Here are

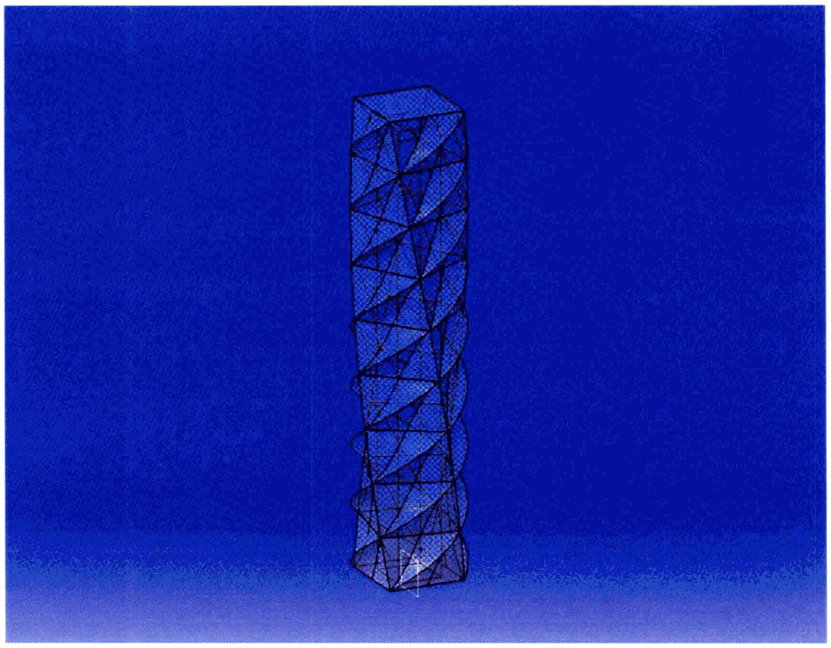


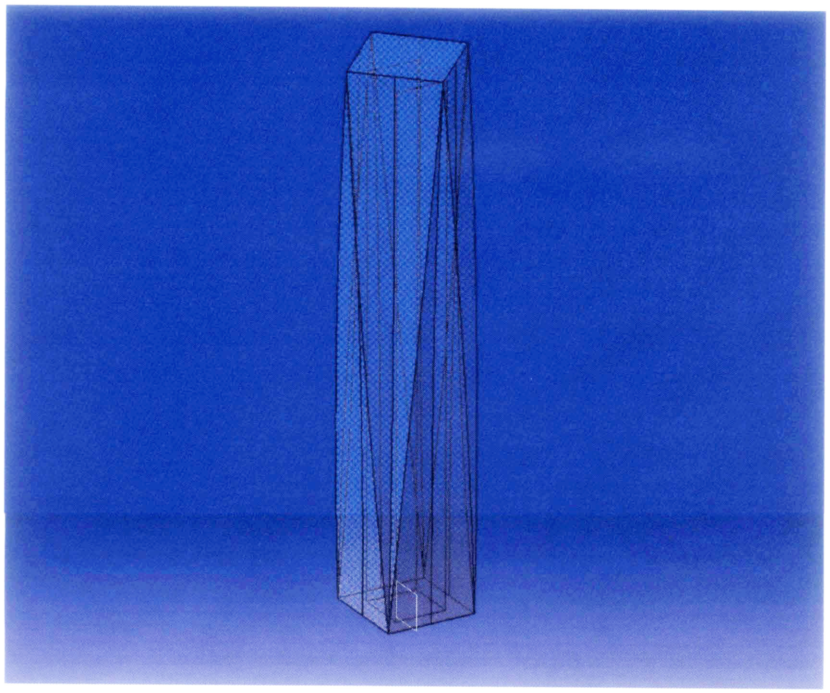


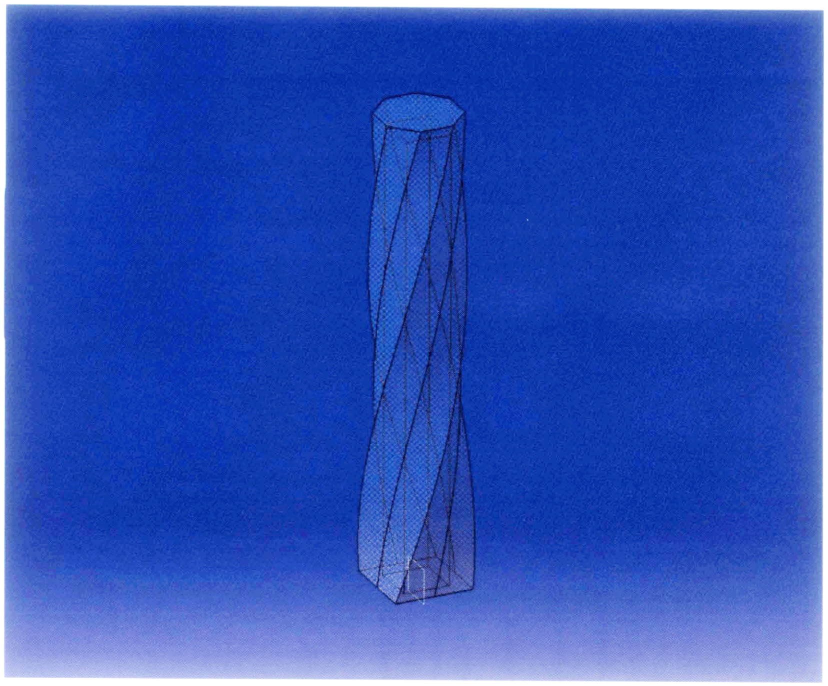


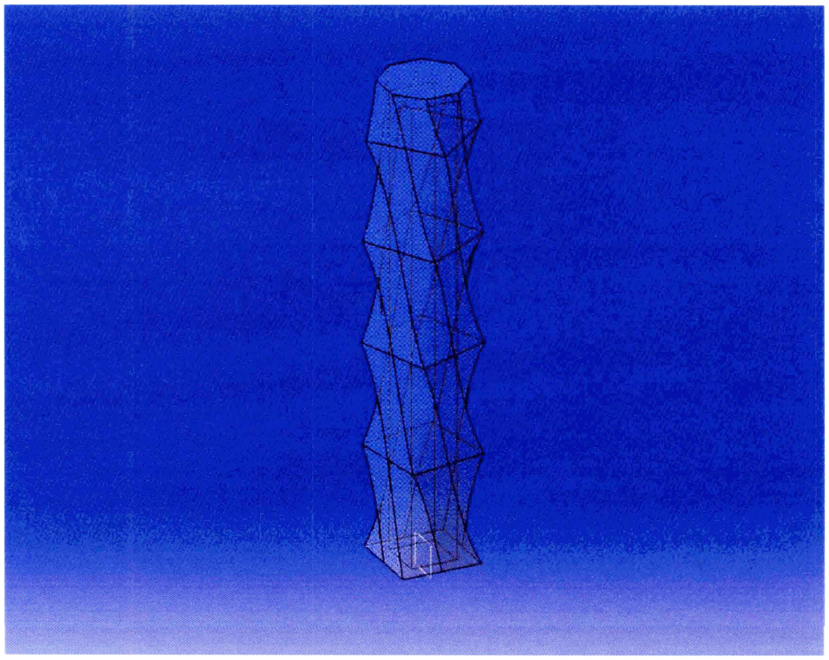


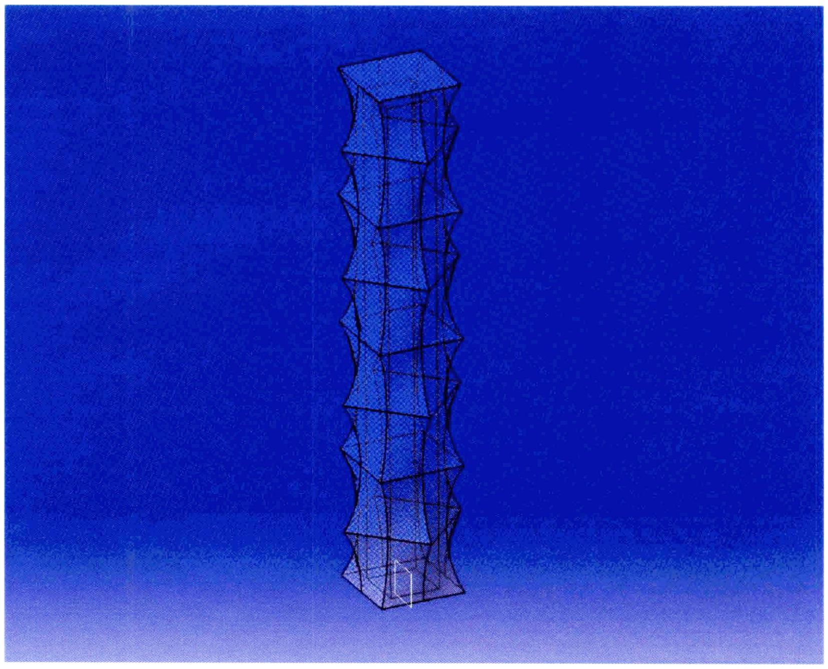


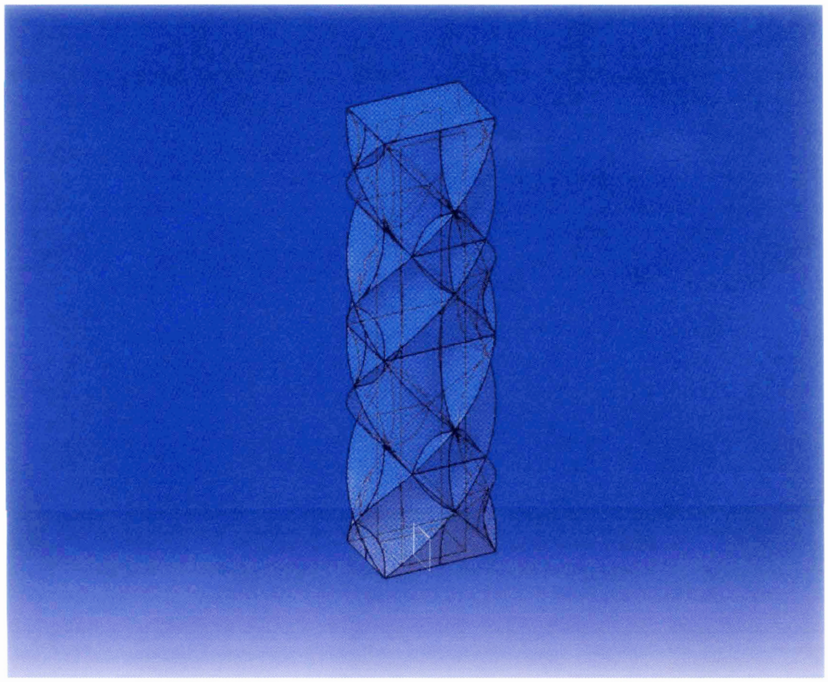


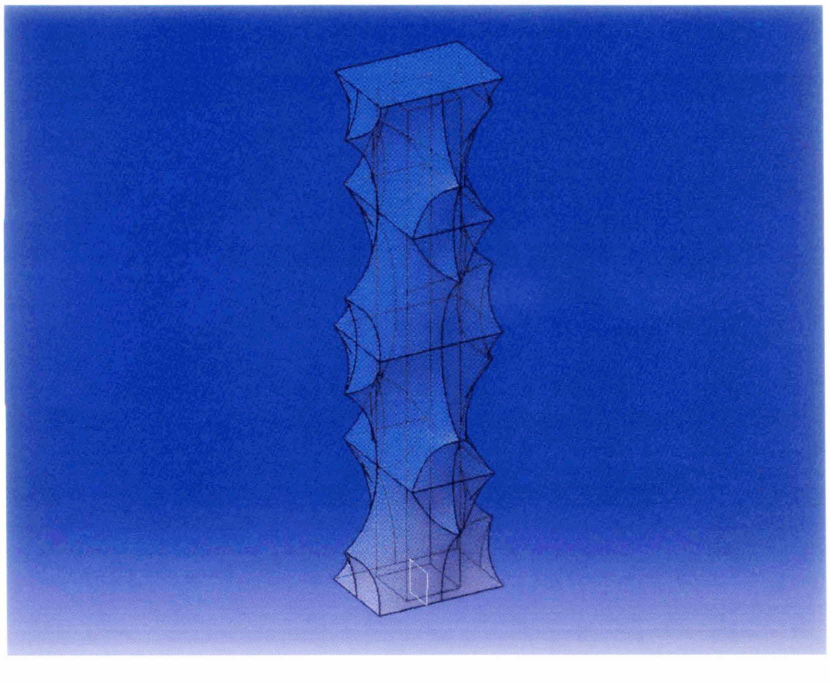


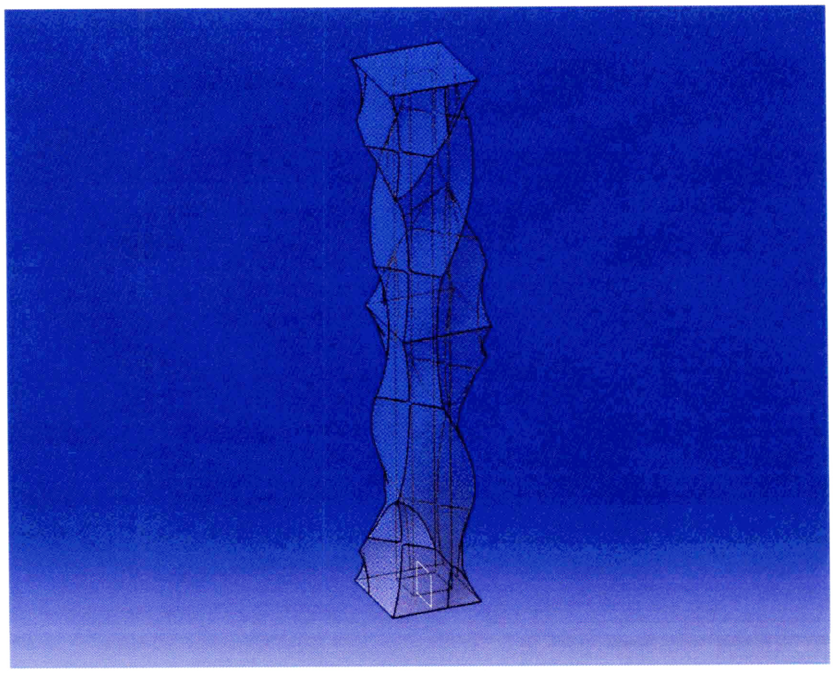


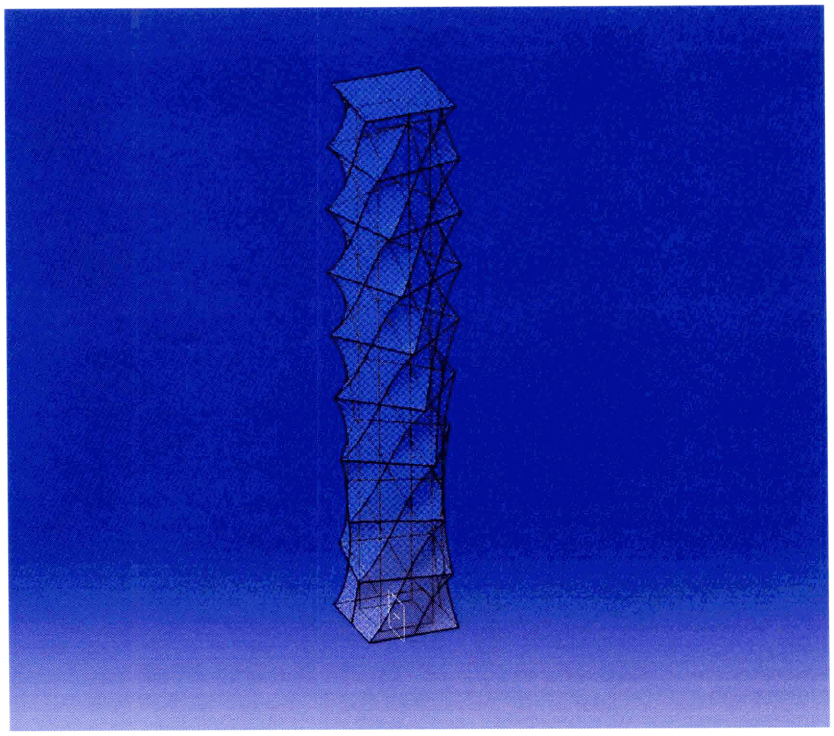


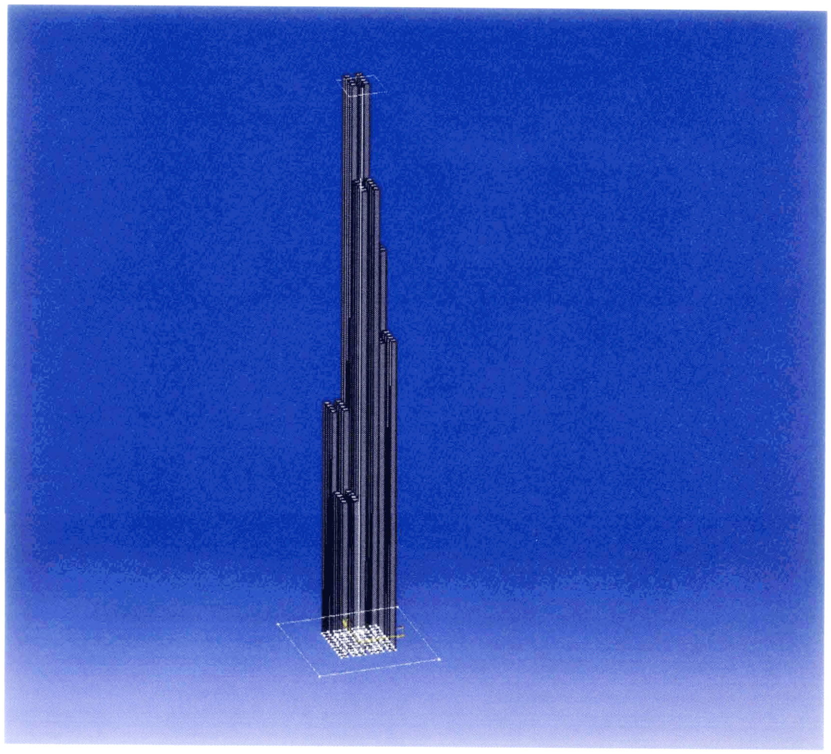


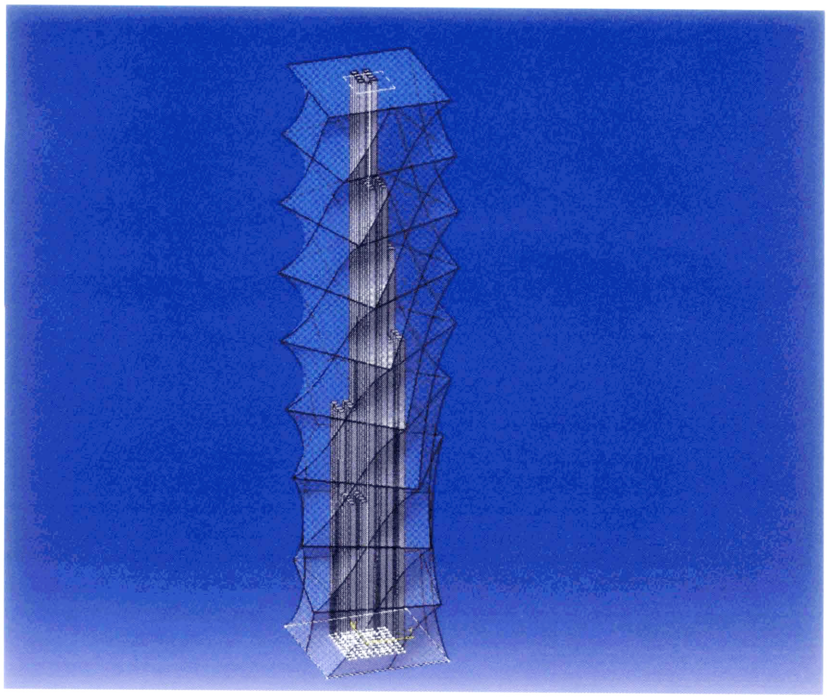


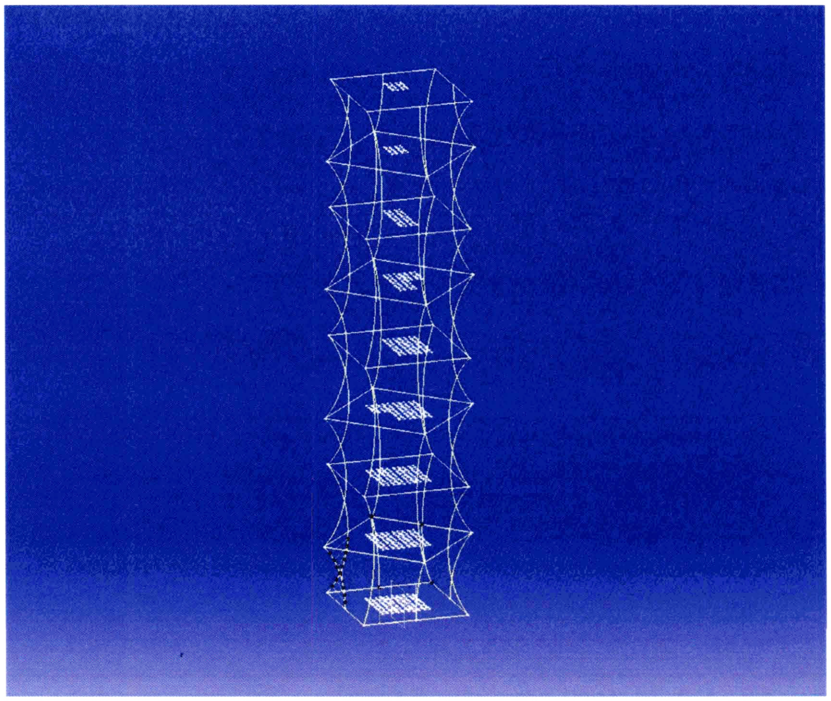


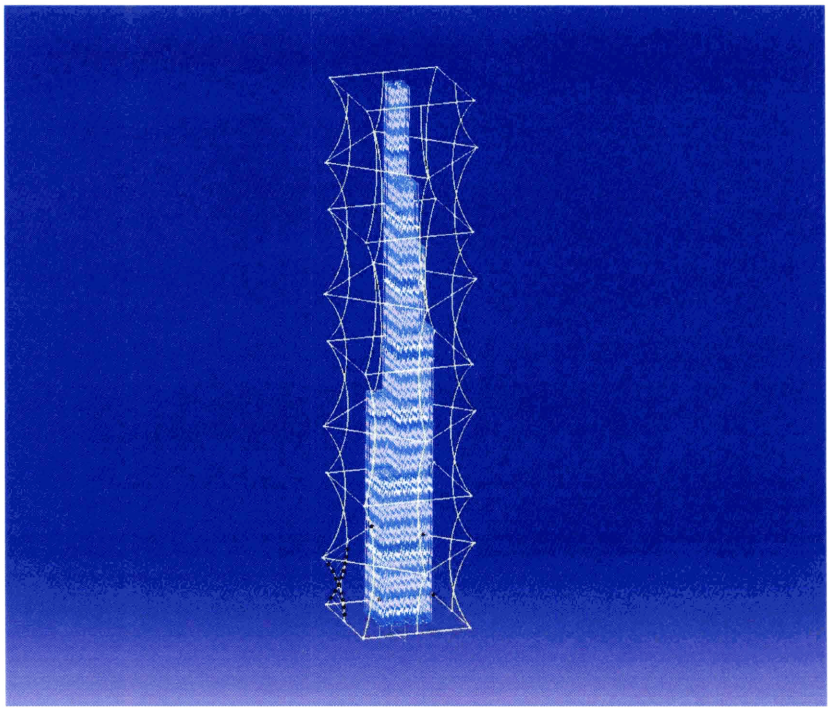












Conclusions

Conclusions

From a computation point of view, Design Procedures can be understood as a search-problem in a very large space of possible solutions. This task can be very expensive even with the most advanced search algorithms. On the other hand, a design procedure offers designers a powerful way to quickly generate parametric models that they can use for design exploration. Search for solutions in a large space of possibilities can be very provocative for a designer; another approach is to implement intermediate solutions where design procedures are constrained to produce certain designs only. These kinds are defined as *deterministic design procedures*.

Parametric models have the general purpose of providing a framework for high-level manipulation of geometrical components that perform transformations during the design process. Among the advantages of using those in design are:

1. The facility to perform changes in geometrical components without erasing a redrawing, allowing flexibility for design exploration and refinement.

2. Increased reusability of design solutions by encapsulation. Complex geometrical models can be placed into basic units that are treated as primitive entities.

3. Added rigor to design development, since a properly constrained parametric model allows some types of transformations, while restricting others.

4. Real time feedback when changes in the parametric model affect geometrical components or other parts of the design.

Design Procedures brings to the surface an important question concerning the validity of designs with respect to the design language. As previously mentioned, variations of a parametric model create instances which are grouped in a category named a family of designs. By simple analogy, a design procedure creates families of parametric models, in other words, families of families with a greater number of design instances. This matter calls for the evaluation of the parametric models as well as the instances.

Another important aspect to consider is the evaluation of the design instances. Evaluations can be one of three types: 1) Performance based; 2) Aesthetic; and 3) Compliance. In performance based, a design instance is evaluated with respect an ideal result, and the model is modified to optimize a solution with respect from the ideal one. Aesthetic evaluation will determine if an instance satisfies a set of values determined by the designer. Compliance asserts if a design instance fulfills a predetermined set of requirements. Any of the aforementioned criteria can be implemented in a design procedure for evaluation of the design instances. The evaluation can be interactive in real time or afterwards.

Design Procedures are inherently non-deterministic and boundless; therefore it is impossible to foresee all the potential results. This is the major assets that a generative system can offer a designer, in particular during the initial stages of design where multiple solutions are explored almost simultaneously. The most difficult task that remains to be solved is how to overcome the initial setup, which can a time consuming but worthwhile enterprise. Perhaps a careful and accurate analysis of the pre-conditions of setup would provide some solutions in this regard.

Design Procedures offers a novel solution to expand the universe for exploration of design instances, in particular as a model for generating parametric designs. Design procedures, which are based on a general course of action followed by a designer, is independent of the geometrical shapes and their representation. As a parametric models generation system, the possibilities for application of the design procedures are absolutely boundless.

References

References

1. Achten, H.H., *Generic representations : an approach for modeling procedural and declarative knowledge of building types in architectural design* in *Eindhoven University of Technology*. 1997.
2. Barrios, Carlos, *Parametric Gaudi.* in *SIGraDi.* 2004. Sao Leopoldo, Brazil.
3. Barrios, Carlos, *Thinking Parametric Design: Introducing Parametric Gaudi*, Design Studies, Vol 27 No 3, May 2006.
4. Burry, M., *Expiatory Church of the Sagrada Familia.* 1993, London: Phaidon Press Limited. 98.
5. Burry, M., *Rapid prototyping, CAD/CAM and human factors.* Automation in Construction, 2002. **11**(3): p. 313-333.
6. Duarte, J.P., *Customizing mass housing : a discursive grammar for Siza's Malagueira houses*, in *Department of Architecture*. 2001, Massachusetts Institute of Technology: Cambridge, MA. p. 536.
7. Gips, J.a.S., George, *Algorithmic aesthetics : computer models for criticism and design in the arts.* 1978. 220 p.
8. Gomez, J.e.a., *La Sagrada Familia : de Gaudi al CAD.* 1996, Barcelona: Edicions UPC, Universitat Politecnica de Catalunya. 166.
9. Gross, M.D., *Design as exploring constraints*, in *Massachusetts Institute of Technology, Dept. of Architecture.* 1986.
10. Knight, T.W., *Transformations of Languages of Designs.* Environment and Planning B: Planning and Design, 1983. **10**: p. (part 1) 125-128; (part 2) 129-154; (part 3) 155-177.
11. Lockwood, E.H. and Macmillan R.H., Geometric Symmetry. Cambridge University Press. 1978
12. Mitchell, W.J., *Computer-aided architectural design.* 1977, New York: Petrocelli/Charter. 573 p.
13. Mitchell, W.J., Kvan, Thomas, *The art of computer graphics programming : a structured introduction for architects and designers.* 1987: New York : Van Nostrand Reinhold. 572 p.
14. Mitchell, W.J., *The logic of architecture : design, computation, and cognition.* 1990, Cambridge, Mass.: MIT Press. 292 p.
15. Mitchell, W.J., *Roll Over Euclid: How Frank Gehry Designs and Builds*, in *Frank Gehry.* 2002.
16. Monedero, J., *Parametric design: a review and some experiences.* Automation in Construction, 2000. **9**(4): p. 369-377.
17. Oxman, R.E., *Expert System for Generation and Evaluation in Architectural Design*, in *Technion, Faculty of Architecture and Town Planning, Haifa.* 1988.
18. Sacks, R., C.M. Eastman, and G. Lee, *Parametric 3D modeling in building construction with examples from precast concrete.* Automation in Construction, 2004. **13**(3): p. 291-312.

19. Sass, L., *Reconstructing Palladio's villas : an analysis of Palladio's villa design and construction process.*, in *Department of Architecture*. 2000., Massachusetts Institute of Technology: Cambridge. p. 385.
20. Shubnikov A.V. & Koptsik V.A. *Symmetry in science and art*. Plenum Press, New York 1974.
21. Tang, T.-H., *Exploring the Roles of Exploring the roles of Sketches and Knowledge in the Design Process*, in *The University of Sydney, Faculty of Architecture*. 2001.
22. Thompson, D.A.W., *On growth and form*. An abridged ed. / edited by John Tyler Bonner. ed. 1992.: Cambridge Cambridge University Press. 345 p.
23. Wells, Mathew, *Skyscrapers: structure and design*. Lawrence King Publishing, London U.K., 2005
24. Whitehead, H., *Laws of Form*, in *Architecture in the Digital Age*, B. Kolarevic, Editor. 2004, Spoon Press: New York. p. 81-100.

Appendix

Source Code

```
'===================PARAMETERIZED BUILDINGS===================
'Draw the InitialShape of the building profile
'The initial shape is in 2D closed profile
'Run the script after your profile is drawn
'Script will prompt you to select your profile
'The script will prompt the user to input VALUES for the PARAMETERS
'The script will prompt the user to select the OPERATIONS to be used
'The script will prompt the user if HIGH ORDER OPERATIONS will be used and if
'   they are required, the script will ask the user to select which ones
'The script will run and generate the SOLUTIONS and the required OUTPUTS
'============================================================



'================================================================
=========
'PARAMETERS:
'================================================================
=========
'There are four(4) types of PARAMETERS used:
'  VARIABLES = that contain numerical VALUES for the PARAMETERS
'  SHAPES = are used as input PARAMETERS for CONSTRUCTION PROCEDURES
'  OPERATIONS = transformations on the SHAPES both geometrical and euclidean
'  HIGH ORDER OPERATIONS = complex OPERATIONS based on multiple applications of
'     different simultaneous OPERATIONS, RANDOMNESS and NON-LIENARITY
'================================================================
=========



'================================================================
=========
'Detailed description of the PARAMETERS
'PARAMETER TYPE is between brackets []
'================================================================
=========
'VARIABLES:
'BuildingBuildingHeight = [distance] total BuildingHeight of the building obtained by
multiplying
'     the FloorOffset by FloorNumber(measured in feet and inches)
'FloorOffset = [distance] floor to floor BuildingHeight (measured in feet and inches)
'     this is and input PARAMETER **INPUT**
'FloorNumber = [integer] total number of floors
'     this is an input PARAMETER **INPUT**
'RotationAngle = [angle] positive rotation of the floor plates (measured in degrees)
'     If TWIST==YES rotate SHAPE with RotationAngle
'CounterRotationAngle = [angle] negative rotation of the floor plates
'     (measured in degrees) apply only if boolean BoubleTwist PARAMETER==YES
'     !!If DoubleTwist==YES rotate SHAPE with CounterRotationAngle * -1!!
'VerticalScale = [factor] scale factor used for tapering
'     (number between 0 and 100)
'     this is an input PARAMETER **INPUT**
'     100 means the scale is the same at the top (no taper);
'     0 means full tapering until reach a point (pyramid)
'     50 means the to shape is half scale
'RotationOffset = [distance] distance between the center of the building and the
'     center of rotation; default value is ZERO 0
'     If the value is 0 then they are aligned
'     this PARAMETER requires two parts: 1) a boolean to check if the offset
'     exist; 2) the value of the offset by X and Y coordinates
'     this is an input PARAMETER **INPUT**
```

```
'CounterRotationOffset = [distance] distance between the center of the building
'     and the center of counter-rotation
'     just like the previous PARAMETER
'     (if the value is 0 then they are aligned)
'     This PARAMETER exists only if DoubleTwist==YES
'ScaleX = [factor] scale factor used for the 'x' axis only
'ScaleY = [factor] scale factor used for the 'y' axis only
'ScaleZ = [factor] scale factor used for the 'z' axis only
'Quadrant = [integer] vaule of 1 through 4 used to select the portion of the
'     InitialShape in case there is a LEFT-RIGHT or FRONT-BACK TopDownSymmetry
'CoreArea = [number] total area of the core
'FloorCoreArea(i) = [number] area of the core in each floor (i) indicates each
'     floor




'SHAPES:
'MainAxis = vertical line normal to the plane of the INITIAL SHAPE The length
'     of the line is the same as the BuildingBuildingHeight
'RotationAxis = {dependency PARAMETER} axis of the rotation operation
'     It is located at the RotationOffset distance from the MainAxis
'     If Helix==YES; then
'     ask for RotationOffset and assign to RotationAxis
'CounterRotationAxis = {dependency PARAMETER}
'     axis of counter-rotation. It is located and the CounterRotationOffset
'     distance from the MainAxis
'     If Helix==YES; then
'     ask for CounterRotationOffset and assign to CounterRotationAxis
'InitialShape = shape the user draws to be used as input for the generation of
'      the building designs, User draws a shape and the script assigns it to
'      the InitialShape variable
'DoubleShape = [boolean YES/NO] if DoubleShape==YES then select a second shape
'     and assign the shape to the Shape01 variable
'     prompt the user to select Shape01 and Shape02
'     InitialShape variable is not used
'     Shape01 and Shape02 can share variables like: building BuildingHeight, etc.
'     both shapes will have the same values for the PARAMETERS
'Shape01 = [shape] if DoubleShape==YES, select a shape as Shape01
'     Shape01 will use Translation, Rotation and VerticalScale as PARAMETERS
'Shape02 = [shape] if DoubleShape==YES, select a shape as Shape02
'     Shape02 will use Translation, CounterRotation and VerticalScale as its
'     PARAMETERS
'DifferentialShape: [boolean YES/NO] If DifferentialShape==YES, then each shape
'     of Shape01 and Shape02 will have theirs own internal PARAMETERS for their
'     own OPERATIONS;
'SPECIAL NOTE: If DoubleShape==YES, then each shape will have it's own set of
'     PARAMETERS. If DoubleShape==YES and DifferentialShape==YES, then each
'     Shape01 and Shape02 will have it's own values for it's own PARAMETERS
'LeftTopDownSymmetryShape = {dependency PARAMETER} left portion of the Shape used
in
'     case LeftRightTopDownSymmetry==YES, prompt user to select Left or Right portion
'RightTopDownSymmetryShape = {dependency PARAMETER} right portion of the Shape
used in
'     case LeftRightTopDownSymmetry==YES, prompt user to select Left or Right portion
'FrontTopDownSymmetryShape = {dependency PARAMETER} front portion of the Shape
used in
'     case FrontBackTopDownSymmetry==YES, prompt user to select Front or Back portion
'BackTopDownSymmetryShape = {dependency PARAMETER} front portion of the Shape
used in
```

```
'     case FrontBackTopDownSymmetry==YES, prompt user to select Front or Back portion
'Quadrant(L,R,F,B) = [string] to label which Quadrant is used in case that
'     LeftRightTopDownSymmetry==YES and FrontBackTopDownSymmetry==YES




'OPERATIONS:
'Translation: [distance] translation of the InitialShape along the MainAxis
'     this operation is applied recursively. The distance of translation is
'     determined by the FloorOffset and the number of translations is
'     determined by the FloorNumber
'Rotation: [Boolean YES/NO + RotationAngle] positive rotation of the InitialShape
'     with respect to the RotationAxis. If Rotation==YES The value of rotation
'     is determined by the RotationAngle.
'CounterRotation: [Boolean YES/NO + CounterRotationAngle] negative rotation of
'     the InitialShape with respect to the RotationAxis. If CounterRotation==YES
'     The value of the angle is determined by the CounterRotationAngle
'     If DoubleShape==NO use the same values of Rotation; else
'     If DoubleShap==YES, promtp the user for values;
'     If DifferentialShape==YES, prompt the user for values;
'Taper: [Boolean YES/NO + VerticalScale] if Taper==YES, apply VerticalScale
'     factor with respect to the MainAxis
'TopDownTopDownSymmetry: [Boolean YES/NO] If TopDownTopDownSymmetry==YES
then apply a top down
'     TopDownSymmetry
'LeftRightTopDownSymmetry: [Boolean YES/NO + Quadrant] If
LeftRightTopDownSymmetry==YES then
'     apply a left-right TopDownSymmetry
'FrontBackTopDownSymmetry: [Boolean YES/NO + Quadrant] If
FrontBackTopDownSymmetry==YES then
'     apply a front-back TopDownSymmetry
'QuadTopDownSymmetry = [Boolean YES/NO] if LeftRighTopDownSymmetry==YES and
FrontBackTopDownSymmetry==YES
'     then QuadTopDownSymmetry exists; prompt user to select which Quadrant to use
'     L=Left; R=Right; F=Front; B=Back; or if any combinations are needed to
'     generate the solutions(for example left and front, or all of them)
'Twist: [boolean YES/NO + RotationAngle] if Twist==YES, then apply Rotation
'     OPERATION with RotationAngle
'DoubleTwist: [Boolean YES/NO + RotationAngle and CounterRotationAngle] if
'     DoubleTwist==YES then apply CounterRotation; else apply Rotaion only
'Intersect: [Boolean YES/NO] apply only if DoubleTwist==YES. If intersect==YES,
'     then apply a boolean intersection between the RotationShape and the
'     CounterRotationShape; else apply a boolean addition and the two shapes
'     will be superimposed
'TransformX = [ScaleX] non-uniform scale only on the X axis
'TransformY = [ScaleY] non-uniform scale only on the Y axis
'TransformZ = [ScaleZ] non-uniform scale only on the Z axis (this will necessary
'     alter the number of floors and the total BuildingHeight of the building
'Helix = [Boolean YES/NO] if Helix==YES, use the RotationOffset; else
'     RotationAxis = MainAxis
'CounterHelix = [Boolean YES/NO] if CounterHelix==YES, and DoubleTwist==YES
'     use the CounterRotationOffset; else
'     CounterRotationAxis = MainAxis
```

```
'HIGH ORDER OPERATIONS:
'HighOrderOperations = [boolean YES/NO] promtp the user to ask if HIGH ORDER
'     OPERATIONS PARAMETERS will be used. If YES, ask which OPERATIONS, else
'     If NO then continue (by default OPERATIONS are linear)
'Linear: [boolean YES/NO] default operation where transformations are done using
'     simple recursion. If Linear==NO then select one of the non-linear options
'NonLinear: [STRING] several types described below. Select from a pull-down menu
'  Incremental: [factor] in each recursion the values of the parameters are
'     incremented by the incremental factor
'  SCurve: [factor] apply an factor to obtain an SCurve function
'  Growth: [factor]
'  Differential: [factor] growth by varying the factor according to a specific
'     rule, IE the edges should be bigger than the center
'  Random: [range] a random function applied where each itteration is assigned
'     a different value for a factor from a designated range. All the OPERATIONS
'     can have a random factor as a YES/NO boolean
'Glide: [boolean YES/NO] if Glide==YES apply a glide transformation. A glide
'     transformation is when one of the operations occurs alternatively as
'     with a possitive value and the next with a negative value
'DoubleGlide: [boolean YES/NO] simultaneous glide in two opposite directions
'DifferentialGlide: [boolean YES/NO] double glide with differential values
'     Glide01 will take the RotationAngle value and Glide02 will take the
'     CounterRotationValue




'OUTPUTS:
'FloorArea(i) = [number] calculate area of each floor
'     output each floor area in a spreadsheet by floor number
'TotalFloorArea = [number] calculate total area of all floors
'CoreArea(i) = [number] area of the core in each floor
'TotalCoreArea = [number] total area of the core
'FloorEfficiency(i) = [percent] percentage of the NetFloorArea on each floor
'     (((FloorArea-CoreArea)*100)/FloorArea)
'TotalEfficiency = [percent] total efficiency of the building
'     (((TotalFloorArea-TotalCoreArea)*100)/TotalFloorArea)
'AverageEfficiency = [percent] calculate the average of the efficiency by adding
'     all floors efficiencies and divide by number of floors
'FacadeArea(i) = [number] area of the facade in each floor
'TotalFacadeArea = [number] total area of the building facade
'TopDownSymmetryType = [string] the TopDownSymmetry type of the tower
'Procedure = a description of the procedure (geometry method statement) listing
'     all the variables, parameters used and their values




'PROTOTYPING OUTPUT
'ZCorp = output of an scaled STL file that will fit in the ZCorp bed
'LaserCutter = output of a cutsheet(s) of the floor plates for the laser cutter




'=================================================================
=========
'=========FUNCTIONS =========
'===================================================================
=========
'Description of required functions
'Function input is in parenthesis (parameters)
'Function output is after keyword returns
'Some functions only apply if the corresponding BOOLEAN PARAMETER is True
'
'TranslateShapeFunction (Shape, Distance)
'  takes a shape and translates the shape along the MainAxis
'  the amount of translation is the Distance
'  RETURNS: translated shape
'
'RotateShapeFunction (Shape, Angle)
'  takes a shape and rotates the shape with respect to the RotationAxis
'  the angle of rotation is the RotationAngle
'  RETURNS: the rotated shape
'
'CounterRotateShapeFunction (Shape, Angle) boolean restricted
'  Use only if BOOLEAN PARAMETER DoubleTwist==YES
'  takes a shape and rotates the shape with respect to the CounterRotationAxis
'  the angle of rotation is the CounterRotationAngle * -1 (negative rotation)
'  RETURNS: the rotated shape
'
'ScaleShapeFunction (Shape, ScaleFactor) boolean restricted
'  Use only if BOOLEAN PARAMETER Taper==YES
'  takes a shape and scales the shape with respect to the MainAxis
'  the scale factor is passed in from VerticalScale PARAMETER
'  RETURNS: the scaled shape




'===================================================================
=========
'GENERATIVE OPERATIONS WITH PARAMETERS==========
'===================================================================
=========
'EXTRUSSION(Shape, MainAxis, FloorNumber, FloorOffset)
'     TRANSLATE Shape along MainAxis by FloorOffset distance
'     recursively COPY Shape in a loop by FloorNumber
'
'TWIST(Shape, RotationAxis, RotationAngle)
'     If Twist==YES
'        ROTATE Shape with respect to RotationAxis by RotationAngle
'     else
'        do not rotate
'
'DOUBLE TWIST(Shape, CounterRotationAxis, CounterRotationAngle)
'     If DoubleTwist==YES
'        ROTATE shape with respect to CounterRotationAxis by
'        CounterRotationAngle * -1
'        (CounterRotationAngle is always possitive, but we must multiply by
'         negative 1 to rotate in the opposite direction)
'     else
'        do not apply operation
'
'TAPER(Shape, VerticalScale)
'     If Taper==YES
'        SCALE Shape by VerticalScale factor
'     else
'        do not apply operation
```

```
'
'TOP DOWN TopDownSymmetry()
'     If TopDownTopDownSymmetry==YES
'        Apply COPY Shape by 1/2 of FloorNumber and then
'        mirror all Shapes right at the middle
'
'LEFT RIGHT TopDownSymmetry(Shape, NorthSouthAxis)
'     If LeftRightTopDownSymmetry==YES
'        cut Shape by NorthSouthAxis
'        promtp user to select which side of the shape to keep
'        DELETE other side shape
'        MIRROR the selected side of the shape and continue
'        return Shapes (cut and mirror)
'     else
'        do not apply operation
'     ALTERNATIVE METHOD:
'     Once the shape is cut, generate both solutions with both sides
'
'FRONT BACK TopDownSymmetry(Shape, EastWestAxis)
'     If FrontBackTopDownSymmetry==YES
'        cut Shape by EastWestAxis
'        promtp user to select which side of the shape to keep
'        DELETE other side of the shape
'        MIRROR the selected side of the shape and continue
'        return Shapes (cut and mirror)
'     else
'        do not apply operation
'     ALTERNATIVE METHOD:
'     Once the shape is cut, generate both solutions with both sides
'
'ALL SIDES TopDownSymmetry(Shape, NorthSouthAxis, EastWestAxis, Quadrant)
'     If AllSidesSimmetry==YES
'        Prompt user to select Quadrant
'        If Quadrant not zero
'           cut shape with both axes to keep selected Quadrant
'        else
'      Cut shape with both quadrants and generate all four solutions
'
'HELIX(RotationAxis)
'  If RotationHelix==YES
'     Prompt user to input RotationOffset
'     Apply ROTATION using the RotationOffset
'  else
'     do not apply operation
'
'DOUBLE HELIX(Shape01, Shape02, RotationAxis, RotationAngle01, RotationAngle02)
'  If DoubleHelix==YES
'     Prompt user to input RotationOffset01
'     Prompt user to input RotationOffset02
'     Apply ROTATION to Shape01 using RotationOffset01 and RotationAngle01
'     Apply ROTATION to Shape02 using RotationOffset02 and RotationAngle02
'  else
'     do not apply operation
'
'COUNTER HELIX(Shape01, Shape02, RotationAxis, CounterRotationAxis, RotationAngle
'           CounterRotationAngle)
'  If CounterHelix==YES
'     Prompt user to input RotationOffset01
'     Prompt user to input RotationOffset02
'     Apply ROTATION to Shape01 using RotationOffset01 and RotationAngle
'     Apply NEGATIVE ROTATION to Shape02 using RotationOffset02 and
'     CounterRotationAngle
'  else
'     do not apply operation
'
'INTERSECTION (Shape01, Shape02)
'     CONDITION: both shapes must exist
'     If Shape01==True and Shape02==True;
'     or If DoubleTwist==YES
'     and Intersection==YES
'        apply boolean intersection to both shapes
'     else
'        apply boolean addition to both shapes
'     else
'        do not apply operation



'=====================================================================
========
'=====================================================================
========
'=====================================================================
========
'============================END OF
DEFINTIONS===============================
'=====================================================================
========
'=====================================================================
========
'=====================================================================
========




'=====================================================================
======
' = user has control of shape in beginning
'FLOOR OFFSET = vertical distance between floor plates
'NUMBER OF FLOORS = number of floor plates
'BUILDING BuildingHeight = (number of offsets * offset)
'ROTATION ANGLE = rotation of floor plates
'COUNTER ROTATION ANGLE = Negative rotation of the floor plates
'TIME STEPS = for every i then rotation (angle of rotation/i)
'SCALE FACTOR = for every j then (scale/j)
'TOP-DOWN TopDownSymmetry = vertical reflection (number of offets/2)
'LEFT-RIGHT REFLECTION: Reflection on a vertical plane
'GLIDE = Talk to Terry dude!!
'X-SCALE = Scale on the X axis
'Y-SCALE = Scale on the Y axis
'Z-SCALE = Scale on the Z axis


'============================
'====GLOBAL VARIABLES=========
'============================
Dim InitialShape
Dim FloorOffset
Dim BuildingHeight
Dim RotationAngle
dim CounterRotationAngle
dim DoubleRotation
dim DifferentialRotationAngle
dim CenterRotAngle
dim Taper
dim ScaleFactor
dim ScaleFactorX
dim ScaleFactorY
dim ScaleFactorZ
Dim FloorNumber
dim TopDownSymmetry
dim LeftRightSymmetry
dim FrontBackSymmetry
Dim CapBuilding

'========================================
'=======SET PARAMETER
VALUES========================================>PARAMETERS
'========================================
FloorOffset = 12
FloorNumber = 72
BuildingHeight = FloorOffset * FloorNumber
RotationAngle = 0
CounterRotationAngle = 0
CenterRotAngle = 2
ScaleFactor = 0 'percentage
CapBuilding = VbTrue
TopDownSymmetry = VbFalse
LeftRightSymmetry = VbFalse
FrontBackSymmetry = VbFalse
DoubleRotation = VbFalse
DifferentialRotationAngle = VbFalse
Taper = VbFalse

'Declare Operation Parameters
'Declare Top Down Symmetry
'Prompt User to get Top-Down Symmetry
dim TopDownSymmetryAnswer
TopDownSymmetryAnswer = Rhino.MessageBox ("TopDownSymmetry" , 4 , "Top Down
Symmetry")
if TopDownSymmetryAnswer = 6 then
   TopDownSymmetry = VbTrue
   'call TopDownSymmetry
end if

'Declere Left Right Symmetry
'Prompt User to get Left-Right Symmetry values
dim LeftRightSymmetryAnswer
LeftRightSymmetryAnswer = Rhino.MessageBox ("LeftRightSymmetry" , 4 , "Left Right
Symmetry")
if LeftRightSymmetryAnswer = 6 then
   LeftRightSymmetry = VbTrue
   'call LefRightSymmetry
end if

'Declere Front Back Symmetry
'Prompt User to get Front-Back Symmetry
dim FrontBackSymmetryAnswer
FrontBackSymmetryAnswer = Rhino.MessageBox ("FrontBackSymmetry" , 4 , "Front Back
Symmetry")
```

```
if FrontBackSymmetryAnswer = 6 then
   FrontBackSymmetry = VbTrue
   'call FrontBackSymmetry
end if

'Declare Rotation Angle
dim RotationAnswer
RotationAnswer = Rhino.MessageBox ("Rotation" , 4 , "Rotation")
if RotationAnswer = 6 then
   RotationAngle = Rhino.IntegerBox ("Enter Twisting Angle", 0 , "Rotation Angle")
   CounterRotationAngle = RotationAngle
end if

'Inquire about double Rotation
dim DoubleRotationAnswer
DoubleRotationAnswer = Rhino.MessageBox ("Double Rotation" , 4 , "Double Rotation")
if DoubleRotationAnswer = 6 then
   DoubleRotation = VbTrue
   dim DifferentialRotationAnswer
   DifferentialRotationAnswer =  Rhino.MessageBox ("Differential Rotation" , 4 , "Differential
Rotation")
   if DifferentialRotationAnswer = 6 then
     CounterRotationAngle = Rhino.IntegerBox ("Enter Counter-Twisting Angle", 0 , "Counter
Rotation Angle")
   end if
end if

'Declere Tapering
'Prompt User to get Taper value
dim TaperAnswer
TaperAnswer = Rhino.MessageBox ("Building Taper" , 4 , "Building Taper")
if TaperAnswer = 6 then
   Taper = VbTrue
   'call Taper
end if

'Prompt User to Enter Taper Value
dim TaperValueAnswer
if Taper = VbTrue then
   TaperValueAnswer = Rhino.IntegerBox ("Enter value", 0 , "Scale Factor")
   ScaleFactor = (TaperValueAnswer / 10)
end if

'*******************************************
'*******************************************
'==============BEGIN SUB - MAIN FUNCTION================
'*******************************************
sub rodTower()

Dim allShapes
Dim rotAngle
dim scale
dim cRotAngle

'Add one to size array correctly to include starting shape
rotAngle = RotationAngle
CounterRotationAngle = CounterRotationAngle * -1
cRotAngle = CounterRotationAngle
ScaleFactor = (100 - ScaleFactor)/100

'=======PROMPT USER TO DRAW SHAPE==========
'Get profile of building from designer
```

```
Rhino.MessageBox "Select your profile curve", 64, "TWIST AWAY"
InitialShape = Rhino.GetObject("Select profile")
Rhino.UnSelectAllObjects()
'If curve is not closed then exit program
'need a shape that represents a profile of a
'building
If IsNull(InitialShape)then exit sub
if not IsCurveClosed(InitialShape) then exit sub
If not IsCurve(InitialShape) then exit sub

'Check if designer wants vertical TopDownSymmetry
If TopDownSymmetry = VbTrue then
        FloorNumber = FloorNumber/2
end if

'call copy function
'make copies of shape and place in array
'scale shapes along with copy
'returns 1D array of copied shapess
allShapes = copyShape(InitialShape,FloorNumber,ScaleFactor)

'Build Rotations
'rotate each shape in array by the degree requested
for i = 0 to uBound(allShapes)
        call rotateShape(allShapes(i),rotAngle)
        rotAngle = rotAngle - RotationAngle
next

CounterAllShapes = copyShape(InitialShape,FloorNumber,ScaleFactor)
'Build CounterRotations
if DoubleRotation = VbTrue then
for i = 0 to uBound(CounterAllShapes)
        call rotateShape(CounterAllShapes(i),cRotAngle)
        cRotAngle = cRotAngle - CounterRotationAngle
next
end if
'Construct translations of floor plates
allShapes = verticalTranslate(allShapes,FloorOffset,TopDownSymmetry)
CounterAllShapes = verticalTranslate(CounterAllShapes,FloorOffset,TopDownSymmetry)

'Construct building skin
call constBuildSkin(allShapes,CapBuilding)
call constBuildSkin(CounterAllShapes,CapBuilding)

end sub
'==============END SUB



'============================================
'==========COPY and SCALE SHAPE FUNCTION================
'============================================
function copyShape(inputShape, howMany,scale)
        ReDim allShapes(howMany)
        Dim shapeCopy
        Dim boundBox
        Dim centerPt
        Dim diagonal

        'Put initial shape into array
        allShapes(0) = inputShape
```

```
          shapeCopy = inputShape
          for i = 1 to howMany
                    'Get approx center of shape by bounding box
                    boundBox = Rhino.BoundingBox(inputShape)
                    diagonal = Rhino.AddLine(boundBox(0), boundBox(2))
                    centerPt = Rhino.CurveMidPoint(diagonal)
                    'Delete construction line
                    Rhino.DeleteObject(diagonal)
                    'Rotate actual shape
                    allShapes(i) = Rhino.ScaleObject(shapeCopy, centerPt,
Array(scale,scale,scale), VbTrue)
                    shapeCopy = allShapes(i)
          next

          'Return array of new shapes
          copyShape = allShapes
end function
'==========END COPY


'============================================
'==========ROTATE SHAPE FUNCTION=============
'============================================
function rotateShape(shape, angle)
          Dim boundBox
          Dim centerPt
          dim diagonal
          dim AngleIncrement

          AngleIncrement = (angle / FloorNumber)
          'Get approx center of shape by bounding box
          boundBox = Rhino.BoundingBox(shape)
          diagonal = Rhino.AddLine(boundBox(0), boundBox(2))
          centerPt = Rhino.CurveMidPoint(diagonal)
          'Delete construction line
          Rhino.DeleteObject(diagonal)
          'Rotate actual shape
          shape = Rhino.RotateObject(shape, centerPt, AngleIncrement)

          'return shape to sub function
          rotateShape = shape

end function
'========END ROTATE



'============================================
'==========COUNTER-ROTATE SHAPE FUNCTION=============
'============================================
function counterRotateShape(shape, angle)
          dim boundBox
          dim centerPt
          dim diagonal

          'Get approx center of shape by bounding box
          boundBox = Rhino.BoundingBox(shape)
          diagonal = Rhino.AddLine(boundBox(0), boundBox(2))
          centerPt = Rhino.CurveMidPoint(diagonal)
          'Delete construction line
          Rhino.DeleteObject(diagonal)
          'Rotate actual shape
```

```
		shape = Rhino.RotateObject(shape, centerPt, angle)

		'return shape to sub function
		rotateShape = shape

end function
'========END COUNTER-ROTATE


'==========================================
'==========SCALE SHAPES FUNCTION=============
'==========================================
function scaleShape(shape, scaleFactor)


end function


'==========================================
'========TRANSLATE SHAPES FUNCTION===========
'==========================================
function verticalTranslate(shapes, distance, sym)


		Dim ORIGIN
		Dim transAmount
		Dim numShapes
		Dim bBox

		transAmount = distance

		'Create point of reference for translation
		ORIGIN = Array(0,0,0)

		'If TopDownSymmetry require copy shapes in descending order
		if sym = VbTrue then
		'Get true BuildingHeight of building
		'2*how many shapes
		numShapes = UBound(shapes) * 2
		Dim symShape

		'translate first set of shapes
		for i = 1 to numShapes/2
				Rhino.MoveObject shapes(i), ORIGIN, Array(0,0,transAmount)
				'add distane to get new Z value for i+1 shape translation
				transAmount = transAmount + distance
		next

		'mirror first set of shapes to create TopDownSymmetry
		for i = 0 to numShapes - 1

				if i = numShapes/2 then
								for j = 1 to UBound(shapes) - 2
								bBox = Rhino.BoundingBox(shapes(i-j))
								symShape = Rhino.CopyObject (shapes(i-j),
bBox(0), Array(bBox(0)(0),bBox(0)(1),bBox(0)(2)+distance*j*2))
								shapes =
Rhino.JoinArrays(shapes,Array(symShape))
								next
				end if
		next
		verticalTranslate = shapes
```

```
          '================
          'If no TopDownSymmetry required translate shapes normally
          '===============
          else
                    numShapes = UBound(shapes)


                    'iterate and translate each object
                    for i = 1 to numShapes
                              Rhino.MoveObject shapes(i), ORIGIN,
Array(0,0,transAmount)
                              'add distane to get new Z value for i+1 shape translation
                              transAmount = transAmount + distance
                              next

          verticalTranslate = shapes
          end if




end function
'==========END TRANSLATE


'============================================
'========CONSTRUCT ACTUAL TOWER FUNCTION============
'============================================
function constBuildSkin(shapes,cap)
          Dim Building
          'lofting shapes to make building skin
          Building = Rhino.AddLoftSrf (shapes)

          'Dim FloorPlates
          'FloorPlates = Rhino.AddPlanarSrf (shapes)

          'Cap building is requested
          if cap = VbTrue then
                    Rhino.Command "_SelPolysrf _Cap"
                    Rhino.UnSelectAllObjects()
                    Building = Rhino.LastObject()
          end if
          'Hide profile shapes
          Rhino.HideObjects(shapes)

end function
'=========END BUILDING SKINNING


'============================================
'========BUILD FLOOR PLATES FUNCTION============
'============================================
function makeFloorPlates(shapes)
          Dim floorSurface
          'making surface from each curve
          'floorSurface = Rhino.AddPlanarSrf  (shapes)

          'return SURFACE to sub function
          'rotateShape = shape

end function
```

```
'=========END BUILDING SKINNING


'===============================================
'=======RUN PROGRAM
'===============================================
rodTower
```