Chapter 7

Geometric gestures

Architect: Kohn Pederson Fox Associates (KPF)

by Onur Yüce Gün

Parametric modeling enables designers to build complex designs with precise control. Through connecting discrete parts with hierarchical relationships, designs can be driven, updated and modified via the use of numerical, textual and logical values: parameters. The design model is no longer a fixed entity. It becomes malleable, granting us the opportunity to explore, test and evaluate design variations.

One of the greatest benefits of parametric modeling over conventional CAD modeling is the propagation graph, which enables simultaneous manipulation of parts across a design. This encourages the designer to think across a range of interconnected design ideas and enables discovering or establishing relational rules within design parts. One single action triggers a chain of reactions within the built system. When the logic of parametric modeling systems is combined with contemporary free-from modeling, form-finding and design exploration are vastly enhanced.

Unbounded and playful exploration in design is how we discover new ideas. More computational power and fewer geometric limitations simply mean a larger ground for innovation. However, the realities of design-construction practice eventually require more geometric and cost control. Once set into a parametric model, geometrical relationships, connections and limitations can be harnessed towards these practical ends.

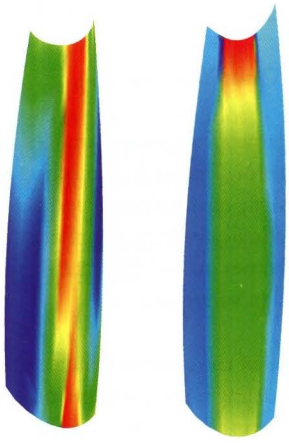
When geometry is incorporated early into the design process, the well-known strategy of post-design rationalization becomes pre-rationalization: geometry and structure become form-making ideas in their own right. Through using such tools, designers gain insight and clarity.

7.1 Geometrical fluidity: White Magnolia Tower

Kohn Pedersen Fox Associates' 68-storey White Magnolia Tower was designed in 2003 as the landmark building of the Luwan district of Shanghai. The digital model of the tower was built in Rhinoceros® 3.0, using non-uniform rational B-Spline (NURB) modeling techniques. The initial design was an exercise in sculpture using neither pre-rationalization nor parametric modeling techniques.

The original model of the tower comprised three identical surfaces that were extended in a slightly different fashion at the top of the building (Figure 7.1). A similar approach applied to the canopy development at the tower base.

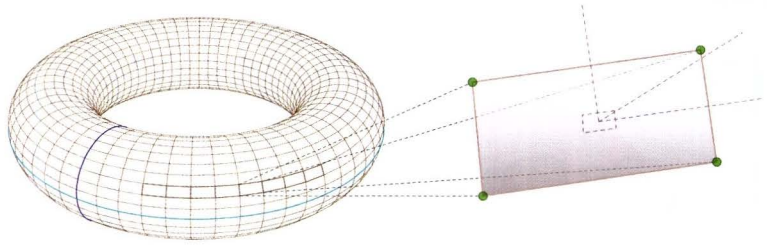
7.1: 3D print of the tower.
Source: Robert Whitlock and KPF.



7.2: Surface curvature properties before (left) and after (right) geometrical rationalization.
Source: Onur Yüce Gün and KPF.

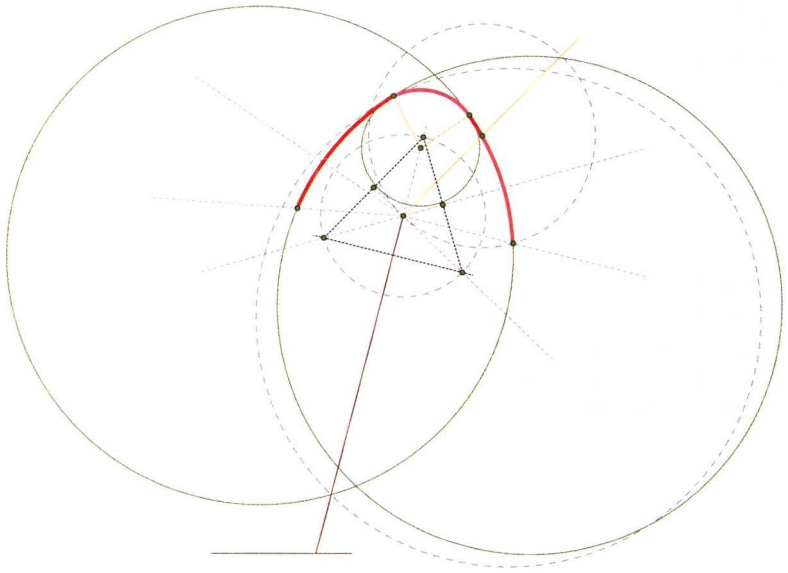
Driven mainly by form-making considerations, the designers made no attempt to control the surface curvature in the original digital model. The complex result had varying and irregular curvature values across the surface. Practical curtain wall design rewards regularities of almost any kind: curvature, planar faceting or common edge lengths (Figure 7.2). Smooth variation in curvature enables a more regular and cost-effective panelization. Flat panels still retain their historical advantages over warped panels, including production time and cost, durability and maintenance.

The design development studies of the White Magnolia Tower centred on the idea of generation and use of parametrically controlled torus patches. A torus, or a rectilinear torus patch, which is a cutout from the surface of a torus, can be subdivided into flat quadrilaterals. These quadrilaterals can be interpreted as flat panels for curtain-wall construction (Figure 7.1).

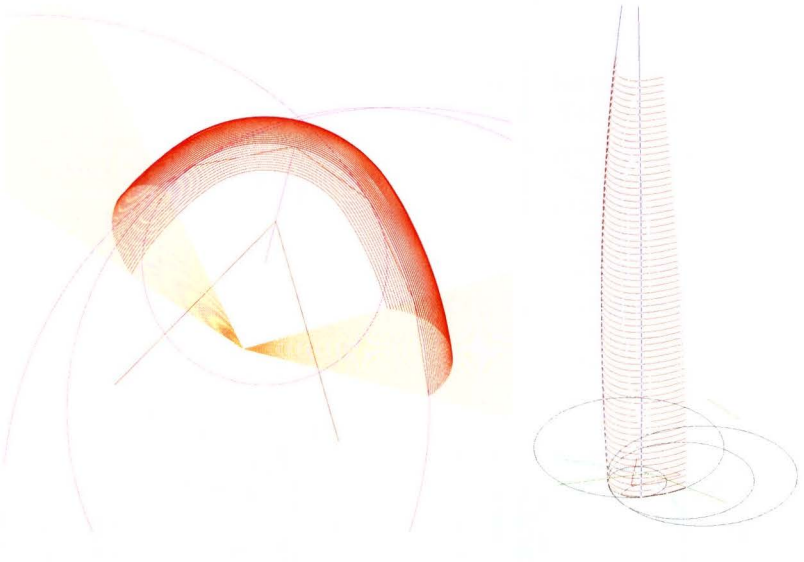


7.3: A torus surface can be panelized with flat panels. The panels common to a horizontal row are the same size.
Source: ©2010 Onur Yüce Gün.

A parametrically controlled set-out generated the floor slab perimeters for each floor of the building. A set of circles with tangential dependencies defined a series of co-tangential arcs forming in to a composed curve (Figure 7.4). At each floor, the composite curve representing the slab perimeter lines was then trimmed from both ends with trimming lines. These lines rotate a small amount (0.44°) in successive floors, yielding 30° of twist overall across the 68 storeys (Figure 7.5). Regardless of the twisted cut on the edges of the overall surface, the shape of the surface remains the same.



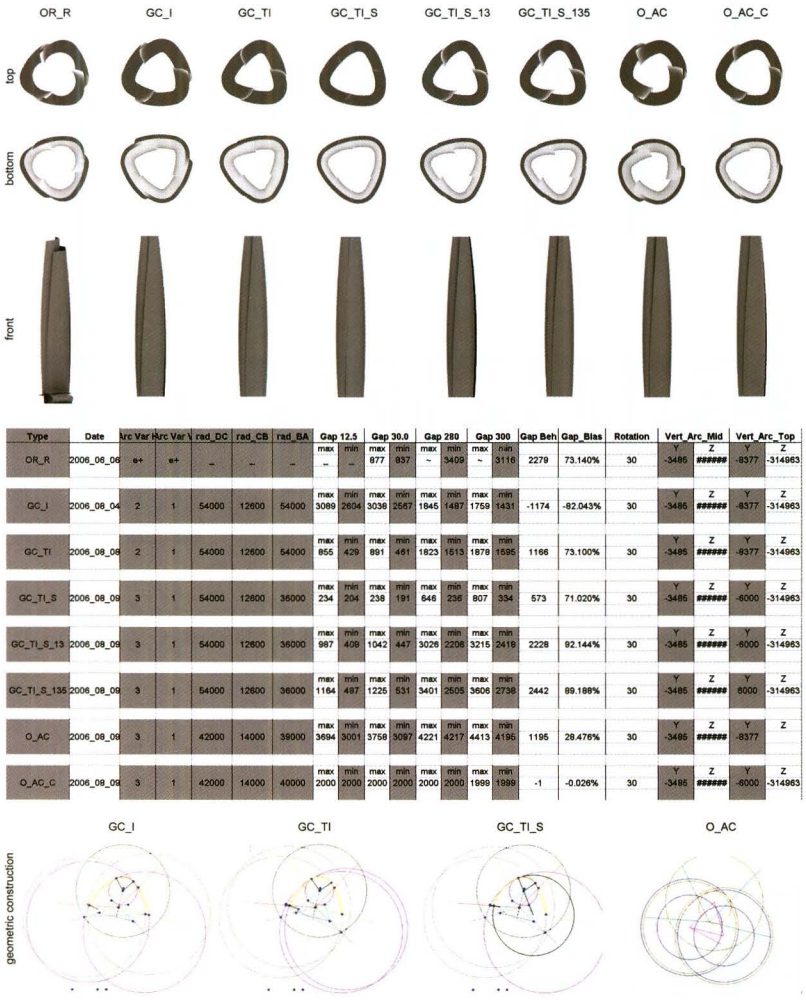
7.4: An underlying diagram of tangential circles create a continuous arc composite with smooth transitions.
Source: Onur Yüce Gün and KPF.



7.5: The composite base curve (in Figure 7.4) is trimmed by trimming lines (in orange) as it is carried upward to generate all floor slab perimeters. Note the twisting effect created by rotating the trimming lines.
Source: Onur Yüce Gün and KPF.

The composite slab perimeter curve scales as it moves along a vertical arc. The manipulation of both the vertical arc and the composite slab perimeter curve defines the overall form of the building, controlling the amount of tapering and the maximum width in the middle of the building. When swept along one of the base arcs, the vertical arc creates a torus patch. Since the base comprises three co-tangential arcs, the resulting geometry is a compound surface of three torus patches. However, the transitions between these patches are smooth since the composite curve arcs have tangential continuity.

The parametric model of the White Magnolia Tower was developed in Bentley's GenerativeComponents®. This model can be driven by both global variables, and by editing associative dependencies between the underlying geometries, which dynamically update in connection to any change in a geometric part. Once running, designers used the parametric model to generate variations of the tower for further evaluation (Figure 7.6).



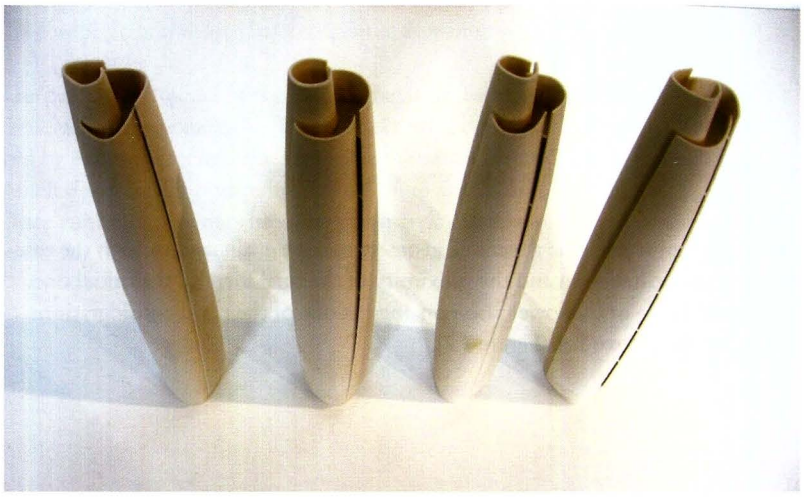
Type	Date	Arc Var	Arc Var V	rad_DC	rad_CB	rad_BA	Gap 12.5 max	Gap 12.5 min	Gap 30.0 max	Gap 30.0 min	Gap 280 max	Gap 280 min	Gap 300 max	Gap 300 min	Gap Beh	Gap_Bias	Rotation	Vert_Arc_Mid Y	Vert_Arc_Mid Z	Vert_Arc_Top Y	Vert_Arc_Top Z
OR_R	2006_06_06	e+	e+	–	–	–	–	–	877	837	~	3409	~	3116	2279	73.140%	30	-3485	######	-8377	-314963
GC_I	2006_08_04	2	1	54000	12600	54000	3089	2604	3038	2567	1845	1487	1759	1431	-1174	-82.043%	30	-3485	######	-8377	-314963
GC_TI	2006_08_08	2	1	54000	12600	54000	855	429	891	461	1823	1513	1878	1595	1166	73.100%	30	-3485	######	-8377	-314963
GC_TI_S	2006_08_09	3	1	54000	12600	36000	234	204	238	191	646	236	807	334	573	71.020%	30	-3485	######	-6000	-314963
GC_TI_S_13	2006_08_09	3	1	54000	12600	36000	987	408	1042	447	3026	2208	3215	2418	2228	92.144%	30	-3485	######	-6000	-314963
GC_TI_S_135	2006_08_09	3	1	54000	12600	36000	1164	487	1225	531	3401	2505	3606	2738	2442	89.188%	30	-3485	######	6000	-314963
O_AC	2006_08_09	3	1	42000	14000	39000	3694	3001	3758	3097	4221	4217	4413	4196	1195	28.476%	30	-3485	######	-8377	Z
O_AC_C	2006_08_09	3	1	42000	14000	40000	2000	2000	2000	2000	2000	2000	1999	1999	-1	-0.026%	30	-3485	######	-6000	-314963

7.6: Various towers as a product of the parametric model.
Source: Onur Yüce Gün and KPF.

During design studies the generated geometries were evaluated on both the ease and cost of construction and the proximity of the final form to the initial one. The visual shape of the tower also remained as one of the main considerations during the design studies (Figure 7.7).

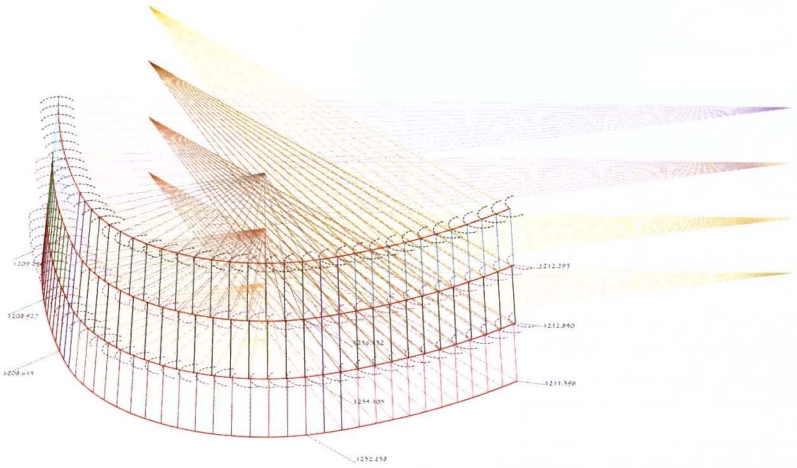
7.7: Preliminary renderings of tower variations. Note the differences regarding the gap between surfaces, and the varying visual sharpness created by different curvature values.
Source: Onur Yüce Gün and KPF.

Three-dimensional prints were used to compare and contrast the visual qualities of alternatives (Figure 7.8).



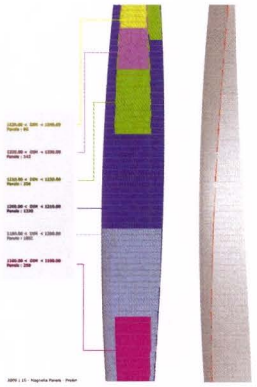
7.8: 3D prints help the designers understand the qualities of the tower form.
Source: Onur Yüce Gün and KPF.

In the next phase of the study, a script developed in McNeel's Rhinoceros® helped automate the tower panelization. The panel placement works as follows. The start point of the slab perimeter line is the centre of a circle, whose radius is equal to desired panel width. This circle intersects the slab perimeter line at a point, which determines the second base point for the panel. The next panel uses this second point of the first panel. The second point becomes the centre of the second intersecting circle, which determines the second point of the second panel. And the routine keeps creating the panels until it reaches the end of the slab floor perimeter line, and then the next floor is processed (Figure 7.9).



7.9: Each panel is created in reference to the previous one and to local geometrical guides.
Source: Onur Yüce Gün and KPF.

Once all the panels were created, they were grouped by their size and colour coded for a quick visualization of the number of panel types. A curtain wall construction tolerance of 10mm determined the boundaries between groups of like panels. With this technique, the tower can be panelized with six different panel types (Figure 7.10).



7.10: Panels grouped and colour coded with a 10mm tolerance in size.
Source: Onur Yüce Gün and KPF.

Computational design methodologies developed for the White Magnolia Tower influenced KPF's ongoing studies for numerous towers, which, at the time of writing, were either under construction or confirmed for construction around the world. For example, geometrical models and construction documentation of the CSCEC Tower in Pudong, Shanghai and F3–F5 Towers in Songdo, South Korea, each extend the studies done for the White Magnolia Tower.

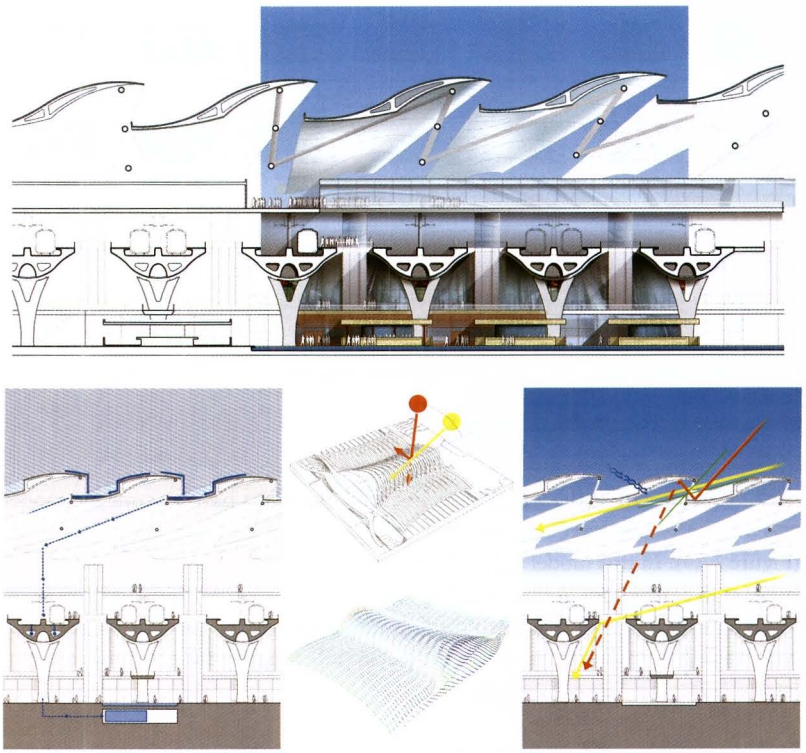
Design studies of the White Magnolia Tower kept the overall form appealing and interesting while achieving practical curtain wall construction. As the KPF (New York) Computational Geometry Group, we exhibited in several events and exhibitions, including the SIGGRAPH 2008 Design Computation Gallery in Los Angeles. During the preparation of this exhibit, we explored additional experimental structural façade patterns (Figure 7.11).

7.11: Fiber-façade: An interpretation by the KPF (NY) Computational Geometry Group as shown in the SIGGRAPH 2008 Design Computation Gallery.
Source: Onur Yüce Gün and KPF.

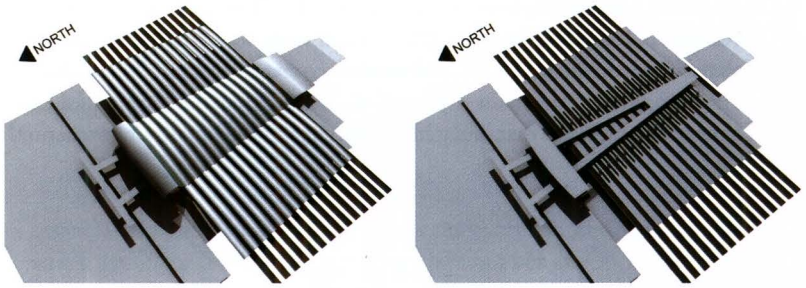
7.2 Designing with bits: Nanjing South Station

Kohn Pedersen Fox New York entered a competition for the Nanjing South Station, which was planned as part of China's high-speed and regular service railroad system. The station is sited in a shallow valley and is bisected through its centre by a "green corridor" connecting the area's major parks. Inside the station the green corridor takes the form of an intermodal hall, around which the arrival hall is located, and on top of which runs the station's platforms and departure lounges. Above the elevated departure lounges, a large sweeping roof protects passengers from rain, sun and wind (Figure 7.12).

The conceptual non-parametric CAD model, prepared as the first 3D model of the station, reveals the initial design intentions around massing and geometric organization (Figure 7.13). Large canopies cover 500m-long platforms lying between the 15 train-tracks aligned on an east–west axis. However, the tracks themselves are not covered in order to admit sunlight onto the platforms. The canopies transform into arced stripes to define the intermodal hall in the middle of the building. Additional canopies connected to the middle of the station on the north and on the south accentuate the entrances.

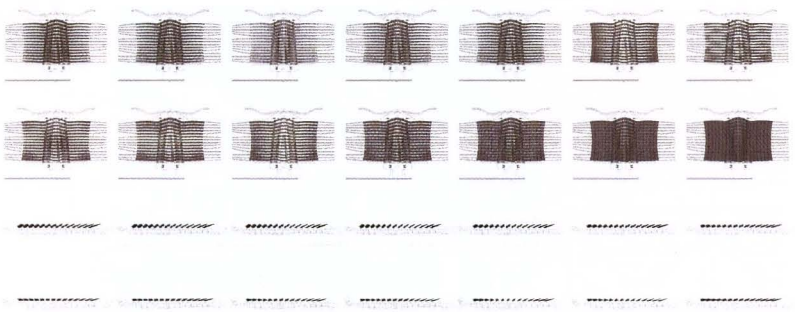


7.12: Section drawings prepared by the design team revealing the intentions about the performance of the sweeping roof. Note the scale of the roof surfaces in comparison to the trains and human figures.
Source: Nicholas J. Wallin and KPF.



7.13: The first non-parametric digital model of Nanjing South Station prepared by the design team, showing the sweeping roof on the left and the train tracks on the right.
Source: David Malott and KPF.

The team intended an organic and fluid form. Conventional non-parametric modeling requires that the very properties to be explored must be decided at the outset. Parametric modeling allows such decisions to be deferred to the end.

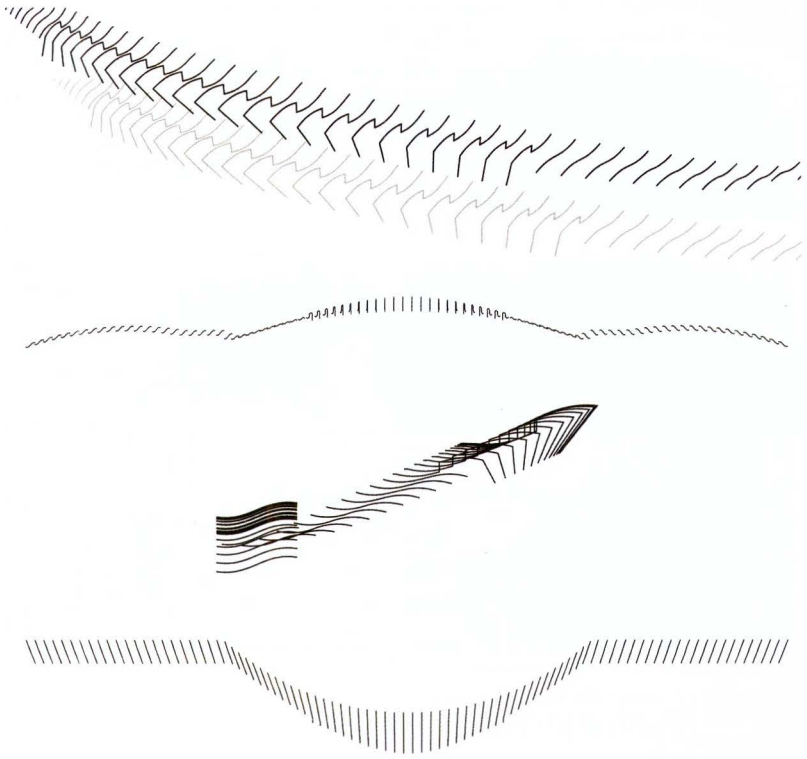


7.14: A parametric model enables the design team to generate and discuss various formal configurations.
Source: Onur Yüce Gün, Stelios Dritsas and KPF.

With references from the non-parametric CAD model, a basic parametric model was built in GenerativeComponents® to explore more formal organizations (Figure 7.14). In this model a global perimeter surface hosts all individual canopy surfaces as continuous forms. Simple parameters update the width and height of these surfaces enabling quick exploration. At this stage, the specific parametric relationships were less important than the overall form. Precise control came later.

In this more advanced modeling phase, a generative S-shaped section plays the main role in defining the characteristics of the surfaces. Although individually simple, under composition and parametric control the S-curve creates a range of different formal conditions (Figure 7.15). These include the steepness and depth of surface, and the amount of projection towards the side. A splitting function divides the S-curve at the higher portion of the roof, tearing an extra opening for more sun exposure where necessary (Figure 7.16). The split basically occurs right in the middle of the S-curve: while one half is elevated, the other remains in its place. The split ends are then tied with a vertical connector. The ends of the S-curves connect to the main structural elements below in a similar fashion. The connection angles are manipulated in reference to the underlying structural elements.

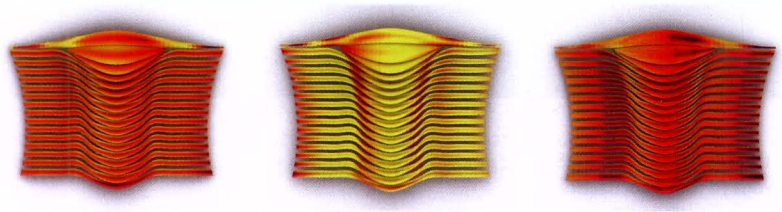
Various configurations, deformations and transformations of simple S-curves, driven by global rule sets and internal parameters for local adaptations, define the characteristics of the surfaces. While driving and determining the design form, these curves remain invisible. The resulting design form affects reflection from and penetration to the station of direct sunlight, as well as water drainage. Solar insolation in each season is affected in a similar manner (Figure 7.17). The final design configuration is a result of these rule-sets imposed on the S-curve system, rather than being a "hand-crafted" geometry.



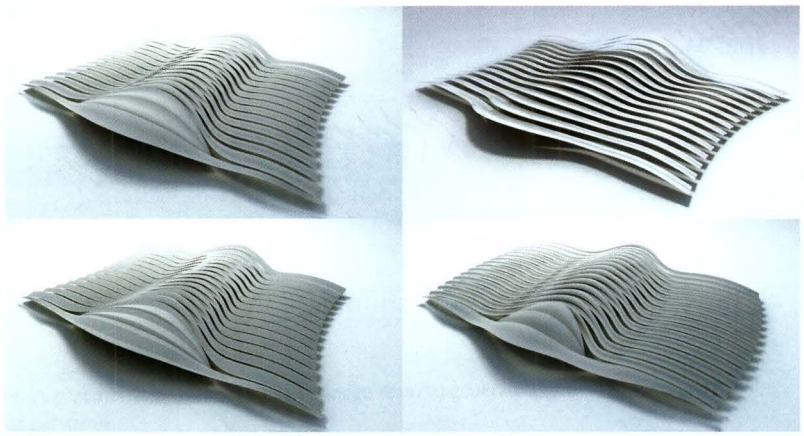
7.15: Simple S-curves and their power of generating variations.
Source: Onur Yüce Gün and KPF.

7.16: A split function creates a gap right in the middle of the S-curve, which is then connected with a linear member.
Source: Onur Yüce Gün and KPF.

While the flexibility and freedom in the exploration phase helps discovery of different formal organizations (Figure 7.18), the limitations and constraints defined in the parametric system help develop precision and higher control over the geometry in later phases.



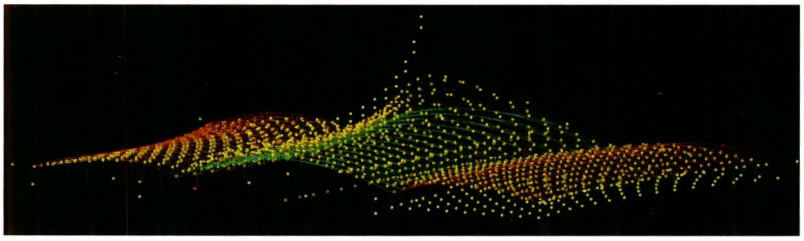
7.17: Solar insolation simulations done for spring, summer and fall give ideas about overall solar exposure of the roof surface.
Source: Onur Yüce Gün, Stelios Dritsas, Mirco Becker and KPF.



7.18: Variations derived from the source parametric model.
Source: Onur Yüce Gün and KPF.

The behaviours defined over the S-curves imposed constraints on geometric outcomes. This way, the model could be called pre-rationalized. With some anthropomorphic license, we can claim existence of a certain awareness in the model; it does not violate boundaries, either stopping or failing as it does. Most of the time it warns the user of impending failure. Thus there is some sort of intelligence, or at least part of the designers' intelligence, embedded in it.

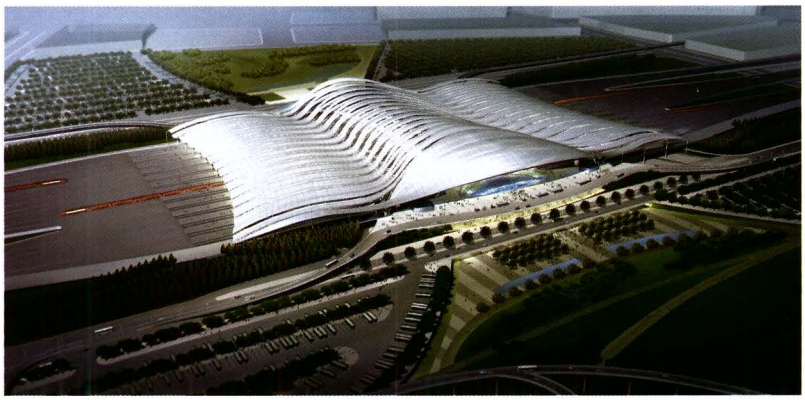
These efforts require custom tool-making, as the generic tools provided by CAD platforms are insufficient to resolve all the geometric requirements and intentions of even moderately complex building models (Figure 7.19). In this case, the GCScript language was used to construct arrays of nodes for references and generating geometric forms.



7.19: The Nanjing South Station model including data in various ranges, from the column layout to roof panelization.
Source: Onur Yüce Gün and KPF.

The parametric model helped generate the structural scheme via the creation of geometric placeholders – points, lines and curves. Files of these placeholders were passed to the structural consultant for analysis. The feedback from the consultants helped update the form towards greater structural efficiency.

The final form reveals the team's discovery of "gestural" form, especially when compared to the initial CAD model (Figure 7.20). Instead of a rapid fluctuation in geometry, the canopies now rise and break apart in the middle to celebrate the intermodal hall. The entrances on the north and south are highlighted by projections. The train station represents a unified form, enabled by relating parts through common geometry and logic. Any manipulation on the global form dynamically updates the form of each canopy.



7.20: The Nanjing South Station entry became one of the two competition finalists. Source: KPF.

A 1/400 scale model, approximately two meters long, was built in China using the digital documentation. This was a rehearsal of actual construction since all the structural members and surface pieces were prototyped. The model, when complete, gave ideas about the configuration of the ribs supporting the canopies and the qualities of the double curved surfaces comprising the overall form.

7.3 Alternative design thinking

The two distinct projects shown here have much in common. The designs for the White Magnolia Tower and the Nanjing Train Station cover a wide range of concerns. Their studies are neither purely technical nor purely aesthetic. On the contrary, the tools developed for each design empower form-finding under technical constraint. Designs today require so many inputs that they no longer are, or can be, one "master's sketch". Likewise, they cannot be a technician's product. The competitive profession of design demands that a firm's whole knowledge be used. Successful design is a complex process done in teams.

Parametric models can carry the needed design complexity. They can embed multifaceted design concerns into a relational digital model. In contemporary practice, the design model is a flexible entity that can be generated, manipulated and re-organized to produce elegant wholes comprising highly customizable and controllable interconnected parts.

Chapter 8

Patterns for parametric design

Abstraction is the hardest new skill for designers. Why? It involves thinking more like a computer scientist than a designer. But does it? Designers employ abstraction all the time as they organize projects and drawing sets. Removing unneeded detail helps keep focus on the issues to hand. Abstract representation enables progress on concrete issues such as circulation, light and structure.

Just as a designer would never specify a building (beyond a doghouse, of course) completely in a single drawing, a parametric modeler should never work in a single model. A complex model is made of (mostly reusable) parts.

Reusable, abstract parts are a keystone for professional practice. Over several years, my research group at Simon Fraser University has used design patterns to understand, explain and express the practice and craft of parametric design. In addition to the the patterns themselves, group members have written theses (Qian, 2004; Marques, 2007; Sheikholeslami, 2009; Qian, 2009) and publications (Qian and Woodbury, 2004; Woodbury et al., 2007; Qian et al., 2007, 2008). Ours is a shared enterprise; throughout this chapter, I use the first person plural to describe what we did together.

A pattern is a generic solution to a well-described problem. It includes both problem and solution, as well as other contextual information. Patterns have become a common device in explaining systems and design situations (Week, 2002; Tidwell, 2005; Evitts, 2000; van Duyne et al., 2002; Gamma et al., 1995). Authors express patterns in various ways. Here we adapt Tidwell's (2005) direct and self-explanatory style comprising *Title, What, Use When, Why, How* and *Examples*. The *Title* should be a brief and memorable name for the pattern. *What* uses an imperative voice describing how to put the pattern into action. *Use When* provides the context needed to recognize when the pattern might be applied. *Why* motivates the pattern and outlines the benefits that accrue to its use. *How* explains the pattern's mechanics. For us, a distinguishing feature of a

pattern is that it explains its mechanism, that is, all instances of the pattern have similar symbolic structure. Examples, which we call *Samples,* provide concrete instances of the necessarily abstract pattern descriptions.

Patterns use the imperative voice. They are normative, describing what should or might be done. They have ancient precursors. Throughout Western history at least, authors have codified practice through text. Vitruvius's *The Ten Books on Architecture* (Pollio, 1914) is the sole architectural text surviving whole from Roman times. From the Renaissance comes Palladio's (1742) *The Four Books of Architecture.* In the 19th Century, Ruskin's (1844) *Seven Lamps of Architecture* looked largely to long past works as the basis for practice. In the 20th Century, Alexander (1979) gave the common "pattern" a specific meaning as a "Pattern" – a formal, rhetorical device expressing design intent. To a computer scientist or linguist, it seems obvious that Alexander was influenced by the computational thinking of the time, particularly by Noam Chomsky's grammars. Alexander built a philosophy of architecture around his patterns. He used phrases such as a "Timeless Way of Building", "a process necessary for good" and "a quality without a name" to prescribe how people should use patterns in the world.

In the late 20th Century, software engineering discovered Alexander's work. In software, patterns became a tool to explain informal mid-level compositional ideas in computer programming (Gamma et al., 1995). The software engineers dropped all of the philosophy, leaving only the device itself. Their justification came from the world; they saw patterns as effective devices for achieving design goals. They grounded specific patterns in shared expertise within a group of authors and reviewers. Each design pattern systematically names, explains, and evaluates an important and recurring design in object-oriented systems. They intended patterns to help users choose design alternatives that make a system reusable and avoid alternatives that compromise reusability. The publication of Gamma et al.'s (1995) book tipped the concept of design patterns to worldwide popularity in the domain of software engineering and other fields.

We now understand that patterns are useful because they foster communication. Rather than having to explain a complex idea from scratch, a group of designers can just mention a pattern by name. Everyone will know, at least roughly, what is meant. Through such sharing patterns have become a popular vehicle for the collection and dissemination of practices and semi-formal ideas.

Our patterns aim to help designers learn and use propagation-based parametric modeling systems. We have largely focused on the GenerativeComponents® system as this allows us to access a large group of designers who are currently learning both the system and the computational concepts underlying propagation-based systems. While we expect that our results could generalize to other systems, at the time of writing we have done limited trials in CATIA® and SolidWorks®. Tsung-Hsien Wang and Ramesh Krishnamurti (2010) have implemented all of our patterns in Rhinoceros®.

We intend that our patterns capture these acts of authorship, above nodes but below designs. Patterns can aid learning. We have taught parametric modeling to several hundred professionals and graduate students. Over time we noticed that our instruction has increasingly focused on this tactical level. We now use patterns as explicit elements in teaching and learning.

This chapter presents 13 patterns for parametric design, explaining each in the abstract and through several samples. Complexity increases throughout – the earlier patterns are simple, the later ones more involved. They group into five categories. The first pattern is in a class of its own; it calls for CLEAR NAMES throughout a model. CONTROLLER, JIG, INCREMENT and REACTOR outline basic model structuring techniques. Paired together, POINT COLLECTION and PLACE HOLDER convey a key method for specifying and locating compound objects. PROJECTION, REPORTER and SELECTOR present ways to abstract information from a model. The final three patterns, MAPPING, RECURSION and GOAL SEEKER, comprise the inevitable residual category of useful (and somewhat complex) ideas.

8.1 The structure of design patterns

Alexander (1979) defines a pattern as a three-part construct: context, problem and solution. His patterns have a common format: a picture (demonstrating a typical example), an introductory paragraph (to set context), a headline (essence of the problem), a long section (body of the problem), a paragraph explaining the solution, and a diagram of the solution. Gamma et al. (1995) use a graphical notation to describe design patterns and provide multiple concrete examples. Tidwell's user interface (UI) patterns (Tidwell, 2005) have a clear and strong structure: name, diagram (usually made by example screenshots), what, use when, why, how and examples. Patterns can be presented both in a formal structure and as a set of flexible ideas. We build largely on software patterns (Gamma et al., 1995) and UI patterns (Tidwell, 2005) to develop a structure for parametric modeling design patterns as follows:

- **Name** is a noun phrase describing the pattern briefly and vividly.
- **Diagram** is a graphic representation of the pattern.
- **What** states a one-sentence description of the goal behind the pattern.
- **When** describes a scenario comprising a problem and a context.
- **Why** states the reasons to use this pattern.
- **How** explains how to adopt the pattern to solve the given problem.
- **Samples** illustrate the patterns with working code.
- **Related Patterns** show the connections among different patterns.

Of the eight pattern elements, samples are distinctive in our work in that they provide concrete, working code as pattern instances. We downplay the language aspects of patterns. Although many pattern authors aim for a complete pattern language that models a design's functional hierarchy, such comprehensiveness and authority proves itself elusive. In counterpoint, Week's (2002) short book of informally defined workplace patterns and Tidwell's (2005) extensive user interface pattern collection use simple categories of patterns and have achieved wide recognition with users and other experts.

8.2 Learning parametric modeling with patterns

Almost all computer manuals are example- and procedure-based. They take you through a series of worked examples, describing keystroke-by-keystroke what you must do to model the example. Some people learn well this way. If you do not, patterns may help. Through teaching parametric modeling to hundreds of people, we have developed a simple and effective three-step process. The first step is learning a minimal set of mechanical steps. You need to learn the basic interaction conventions of the modeler, a few modeling commands and succeed in making a very simple model. The second step is to make a model useful to you in your current work. Start with a sketch in any medium you wish; just make it quickly. Divide it into logical parts, so that each part can be modeled easily. We have found that good outside advice can really help you here. An experienced hand can clarify both the model and its division into parts. The third step is to model the parts and combine them into a whole. Here is where patterns shine. You will likely find that many of the parts resemble patterns (we derived the patterns largely from observing and interacting with designers as they worked). Copy and modify the pattern samples you think may be useful, combining each pattern sample into your model. The patterns will not make up the whole of your model, but the parts they do compose should be clear and clearly separate. This process helps you learn a powerful strategy you already know but in the new context of parametric modeling. Divide-and-conquer is a near-universal strategy in problem solving and design. It appears differently in each medium. In parametric modeling, patterns are one good manifestation.

8.3 Working with design patterns

We developed the design patterns in this book by working with and observing designers learning and using parametric modeling. Chapter 3 distills some of what we discovered into 14 classes of designer action. It would be no surprise if we argued that patterns may help in many of these, but such arguments are currently circular; we commit the *post hoc ergo propter hoc* fallacy when we use the same data to both form and verify theory. In the place of firm conclusions, I hypothesize that patterns can help design work and present several arguments supporting this hypothesis. In the rest of this section, I use the definite voice, presenting hypotheses as if they are established claims. The truth is that these are propositions to be tested by future research.

Four salient attributes of patterns is that they are explicit, partial (above nodes and below designs), problem-focused (shared problems) and abstract (generic).

Explicitness aids reflection. The acts of writing and reading patterns demands a mode of thought different from the flow of design. Like Schön's (1983) *reflection-in-action*, patterns provide a tool for advancing design skill. To write a pattern is to commit it to definite media for others to read in your absence. Patterns are a good tool for groups to build up a shared library of low-level modeling and design ideas. Though explicit, the samples in patterns are intended as throw-away code to be copied and modified at will. Since exact digital copies are freely available, samples cannot be ruined. Minimal work is lost in trying them out. Multiple samples for each pattern provide different roots from which to start. Pattern names are explicit handles for communication in design work.

That patterns are partial means they must be composed into designs. They provide parts with which to solve the "conquer" aspect of the divide-and-conquer strategy. By providing separate solutions to problem parts, they can help clarify the data flow through a model. Properly written, they are informal devices by which modules can be expressed in principle.

Patterns focus on solving problems. When well-written they state a problem and provide several clear solutions to it. They aid sketching by accelerating the creation of approximate models. They often combine geometric, mathematical and algorithmic insight. They demonstrate how to fuse these important and complementary skills.

Lastly patterns are abstract. To use them well evidences mastery of the "divide" part of divide-and-conquer. Using them at all helps develop the special form of divide-and-conquer demanded by parametric modeling.

8.4 Writing design patterns

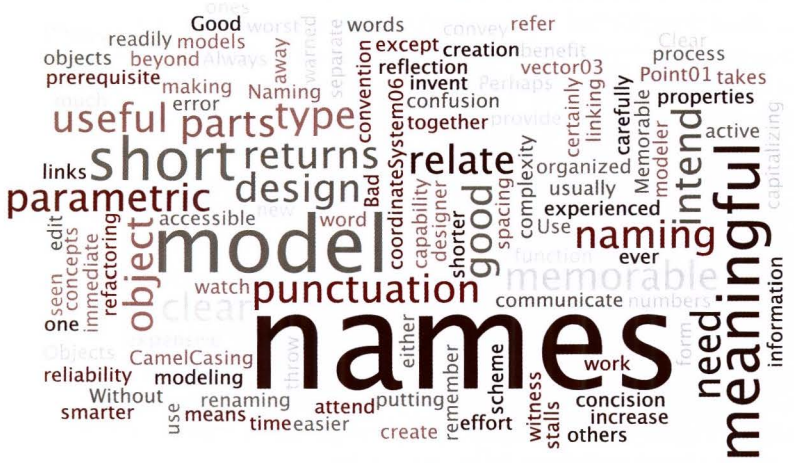
Our research shows that writing your own design patterns may aid reflection on and reuse of design ideas. Patterns take time and effort to write, and return clarity and simplicity later. They can amplify your professional skill. To write a pattern is to listen to yourself and your colleagues. Are you doing the same thing again and again in variations? Can you describe it in a phrase? Do you have sample code that you reuse? If you answer "yes" to each of these questions, think about writing a pattern. Stick to the eight pattern descriptors. As you start, focus mostly on Name, What, When and How. Collect a set of Sample files. Look at these together to discover what they share. Refactor the code in each to be consistent. At this point, you may have the beginnings of a useful pattern. In the slow periods of your work, reflect on the pattern. Refine it for clarity and simplicity. Use it in your work. If it is useful, refine it again. Make it public within your group. Make it easy to find: online is best. Others may be interested in what you have done. Share it widely if you possibly can.

8.5 CLEAR NAMES

Related Pattern • ALL OTHER PATTERNS

Some likely good names

MainBeam3
RoofPanel
SouthFacade
DesignSurface
PlaceHolder
Truss8
Purlin8_3
Foundation
RoseWindow
Pane3_7
ColumnA_7
Hypotenuse



What. Use clear, meaningful, short and memorable names for objects.

When. Always, except for work you intend to throw away.

Why. Objects have names. You use these to remember how you have organized a model, to refer to parts as you create and edit links, and to communicate to others. Clear, meaningful, short and memorable names are a prerequisite for making a model useful beyond its immediate creation.

Some usually bad names

Point02
BSplineA
foobar
aardvark
here
there
angle6
parabola
thingamabob
IansPlane
abc
AfDrAp

How. Good names are clear; they convey what you intend. They carry design meaning; usually they relate to form, function or location. They are as short as they need to be (and no shorter). A good and useful convention for concision is *CamelCasing*, putting words together with no spacing or linking punctuation and capitalizing each word (separate numbers with punctuation). Memorable names explain design concepts.

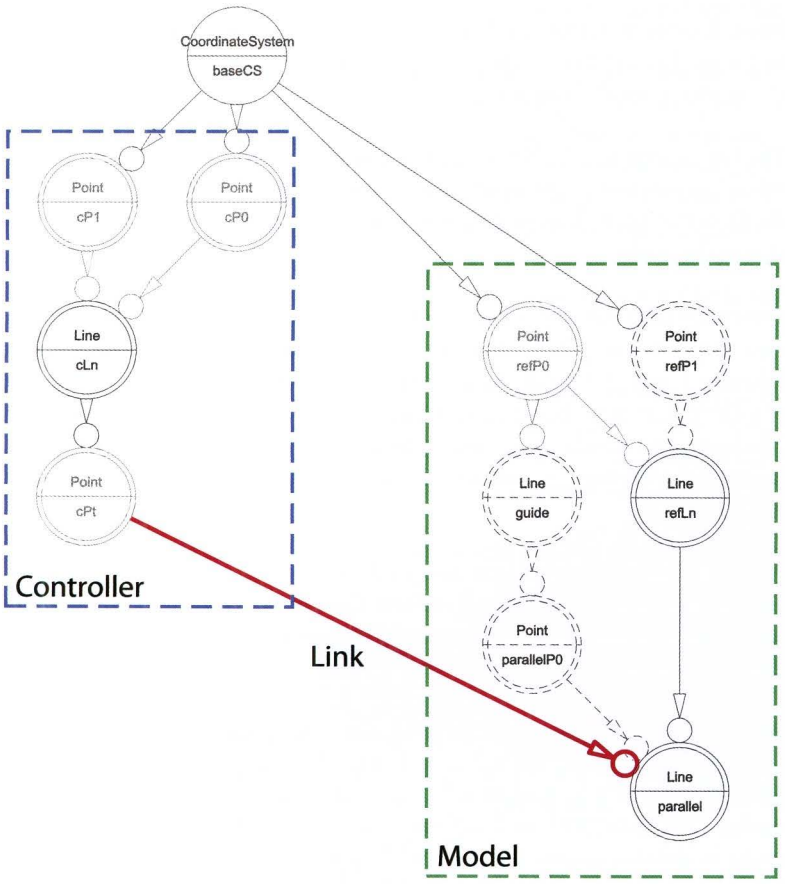
Bad names are easier to invent than good ones. Perhaps the worst naming scheme is by object type. "Point01," "vector03" and "coordinateSystem06" provide no new information; the type of an object is one of its properties.

Source for tag cloud:
www.wordle.net

Naming is active. If you watch an experienced parametric designer, you witness a process of naming, reflection and renaming. As model complexity increases, this expensive refactoring returns a benefit. In its absence, modeling stalls in confusion and error. Be warned. Unless you are smarter than any parametric modeler I have ever seen, you need to attend carefully to the names of model parts. It takes time and effort, but returns capability and reliability.

8.6 CONTROLLER

Related Pattern • JIG • POINT COLLECTION • REACTOR • REPORTER • SELECTOR • MAPPING



What. Control (a part of) a model through a simple separate model.

When. The essence of parametric modeling is the parameter – a variable that influences other parts of the design. Understanding how parameters affect a design is a crucial part of the modeling process. Use this pattern when you want to interact with your model in a clear and simple way, OR you want to convey to others how you intend a model to be changed. *Remember, in the future you may well be such another person if you have forgotten the model structure!*

Why. Isolating manipulations to a simple place away from the complex detail of a model means that you can change the model more easily. Using a logic for control that is different from the way the model is defined means that you can use the most appropriate interaction metaphor. Changing a collection of objects through a single interface simplifies the interaction task.

As models grow, so does the need for carefully considered CONTROLLERS. In particularly complex models, you may well design and implement a separate control panel that collects all of the CONTROLLERS into a single place in the interface.

How. A CONTROLLER can do either or both of the following: it can abstract an aspect of a model into a clear and simple device or it can transform an aspect of a model into a different form.

The key concept in a CONTROLLER is separation. You build a separate model whose outputs link to the inputs of your main model. The separate model is the CONTROLLER. It should express, simply and clearly, the way you intend to change the model.

CONTROLLERS can abstract or transform and they can do both at the same time. An abstracting CONTROLLER is a simple version of the main model that suppresses unneeded detail. Parameters on lines and curves are very simple cases of a CONTROLLER: they abstract a location on a curve into a single number. The layout of controls on a properly designed stovetop directly abstracts the layout of the burners. In contrast, the vast majority of stovetop controls fail to do this well.

A transforming CONTROLLER changes the way you interact with a model. For example, polar coordinates transform Cartesian coordinates into a different set of inputs. A rotating knob on a stovetop transforms the amount of energy delivered into an angle.

As one property changes in your model, one or more parts change; you can connect these changing properties to your model through a CONTROLLER. Then, you can simply change the CONTROLLER and see the result in your model. CONTROLLERS are thus independent – they have minimal connection to the model they control and are easily connected and disconnected as needed. This clear separation is the hallmark of a CONTROLLER: every well-designed CONTROLLER will have a symbolic model that shows only one or a scant few links between it and the model it controls.

CONTROLLER Samples

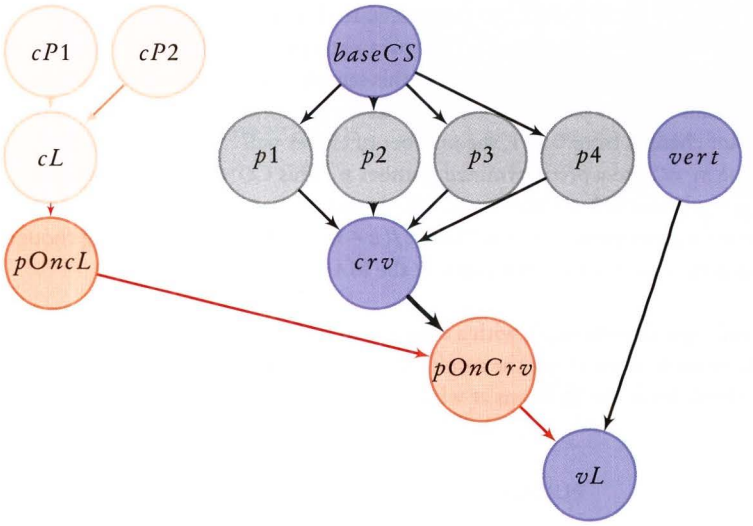
Vertical Line

When. Control the position of a vertical line on a curve with a CONTROLLER.

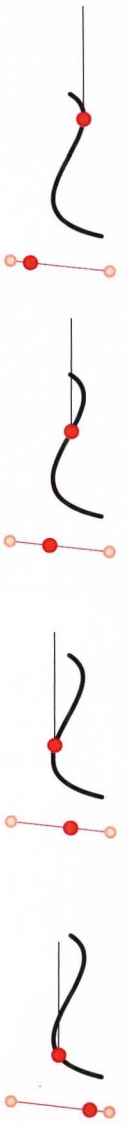
How. Curves and surfaces are complex objects. Their parametric structure is typically hidden from the interface – a point may move quickly in one part of a curve and slowly in another as its parameter changes. Further, curves and surfaces are usually part of a design. There may be many other objects around them that make it difficult to directly interact with their parametric points. Controlling a point on a curve through a parametric point on a line addresses both of these issues. You can see the relative parameterization of a curve point by examining its controlling point. Further, the controlling point can be in any position, near to or far from the model.

In this model, a single vertical line takes its position from a parametric point on the curve *pOnCrv*. The CONTROLLER is a line and a parametric point on it. Making the parameter of the point *pOnCrv* dependent on the point in the CONTROLLER transfers control from the curve to a straight line.

This is a very simple sample, but it demonstrates the essential idea of separation of control and model.



8.1: The CONTROLLER on the left joins to the main model on the right through only a single link. Such sparsity of connection is a hallmark of a CONTROLLER.



8.2: A simple CONTROLLER. The point on the line controls the point on the curve, which, in turn, is the base for a vertical line.

Line Length

When. Change the length of a vertical line with a slider.

How. This is a very simple sample of the CONTROLLER, but one that transforms a length in one direction to a length in another. Start with a vertical line and a horizontal CONTROLLER line. Connect the length of the vertical line to the parameter of the point on the CONTROLLER line. Moving the point in the CONTROLLER alters the length of the vertical line.

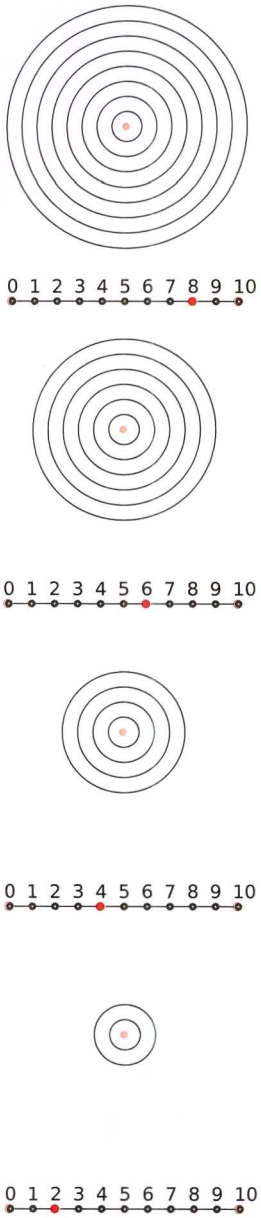
8.3: The previous sample maps the CONTROLLER'S motion along a line to the controlled points motion along a curve. This CONTROLLER is less direct: it maps location along a line to the height of the controlled line.

Multiple Circles

When. Change the number of concentric circles with a slider.

How. The quantities controlled can be continuous (a real number) or discrete (an integer or member of a sequence or set). In this sample, a slider controls the numbers of concentric circles. The parametic point on the slider connects to the creation method of the circle. In this sample, the number of the circles is determined by the parameter of the point on the slider. As the parameter of the point changes from 0 to 1, the number of circles will change from m (in this case $m = 0$) to a predetermined number n. This CONTROLLER requires a mapping between $0-1$ and $1-n$. The actual math is simple: the number of circles for a given parameter t is Floor$(t/(n-m))$. This idea of mapping though is so general that it has its own pattern: the MAPPING pattern.

Controlling a discrete result with a continuous slider creates visual dissonance: the slider seems smooth yet the result changes in steps. A typical solution is to mark the slider at the locations at which the number of discrete objects changes.



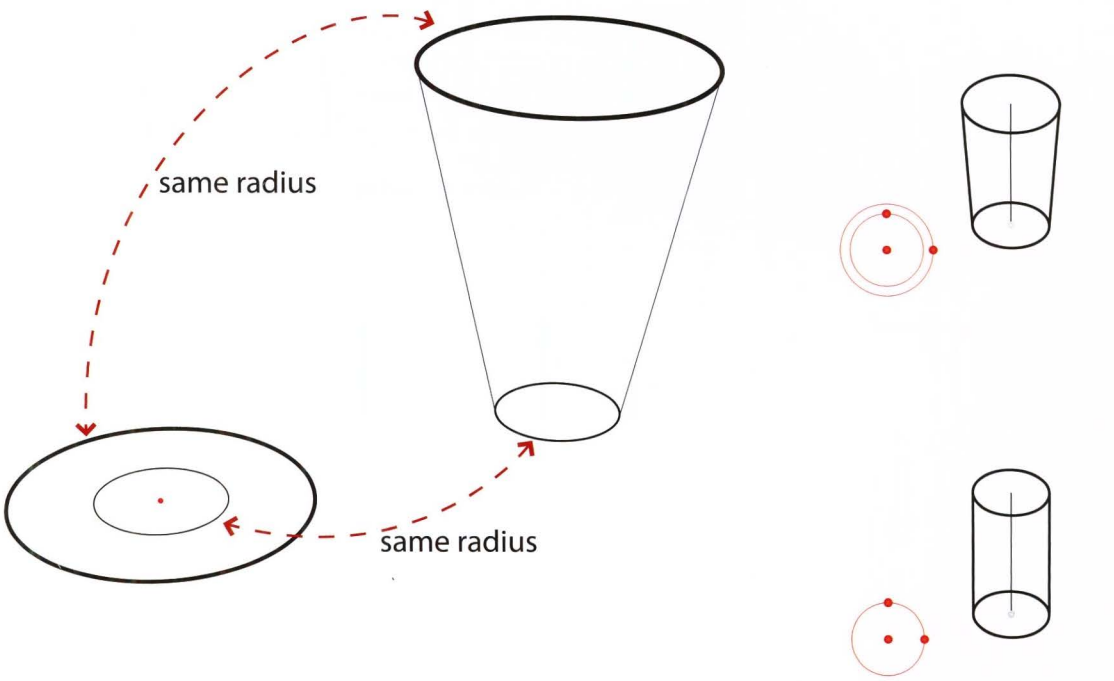
8.4: Control the number of circles in a model with a point on a line.

Cone Radii

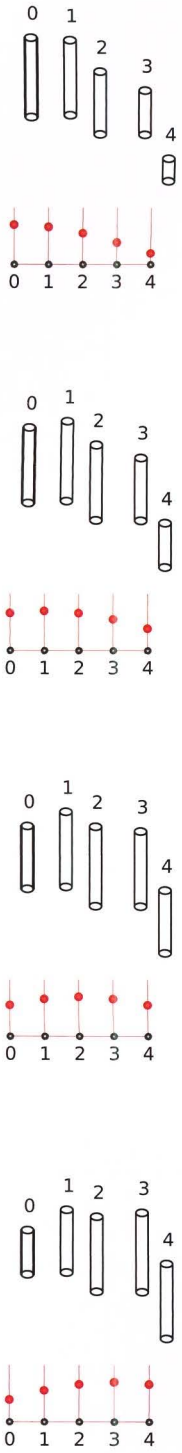
When. Change the radii of a cone with a pair of concentric circles.

How. A single CONTROLLER can control multiple aspects of a design. Of course, this alone poses a design problem. The CONTROLLER must visually cohere with the object being controlled. In this sample, the aim is to control the top and bottom radii of a cone. The CONTROLLER maps from concentric, coplanar circles to the cone's top and bottom surfaces. Its circles are controlled by points on their boundary. Two links, one the radius property of each of the CONTROLLER'S circles to the cone radii connect the CONTROLLER to the model. The CONTROLLER circles provide a visual reminder of the real objects being controlled: the top and bottom of the cone.

Most of the time, the relative size of the circles when compared with the cone suffices to distinguish the link between aspects of the CONTROLLER and model. Such geometric coincidence may fail to satisfy, for example, when viewing in perspective or when the two radii are very close. Other codes, such as colour (careful here!), text, line weight or graphic labels, might be useful.



8.5: The CONTROLLER'S circles visually map to the top and bottom of the cone.



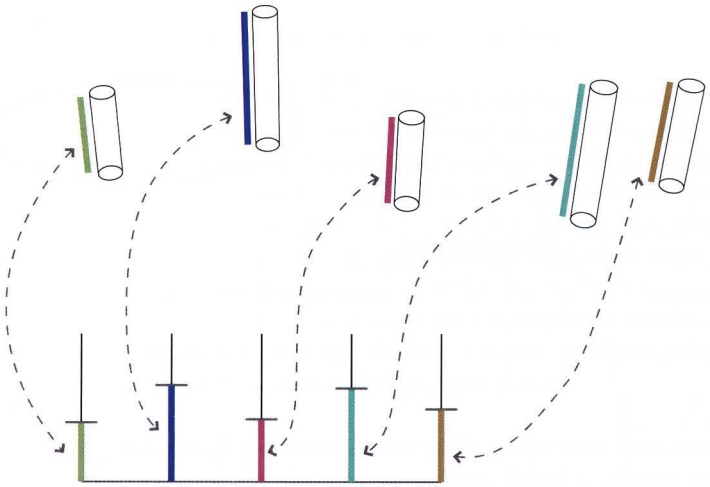
8.6: This CONTROLLER works like the familiar equalizer in sound systems.

Equalizer

When. Adjust the height of multiple cylinders with an equalizer.

How. In this sample, the CONTROLLER takes advantage of a roughly linear arrangement of objects in the model by using the well-known design for a sound equalizer. The equalizer is a row of sliders, each interactively independent of the others. This design puts the controlled dimensions into visual proximity and thus reveals their relative sizes, which might well be obscured in the model due to location, size and perspective effects.

This CONTROLLER misses an important aspect of the design of a physical sound equalizer. With such a device an operator can use his or her entire hand to simultaneously control several dimensions and to achieve a smooth curve across dimensions. The computer mouse, with its relentless one-thing-at-a-time design impoverishes the potential interactivity of the CONTROLLER. Some of this could be recovered by using a REACTOR or SELECTOR pattern as part of the CONTROLLER itself.



Parallel Lines

When. Adjust the length and position of a line parallel to a reference line.

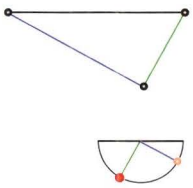
How. In this sample, a single CONTROLLER affects multiple parameters. It is the converse of the samples above, in which multiple independent controls form the CONTROLLER. A reference line establishes the direction and maximum length of the result line. The CONTROLLER comprises a single line carrying a parametric point. The line's direction and length determines the direction and length of a *guide* line that originates at one end of the reference line. A point on the guide line gets its parameter from the CONTROLLER and is the start point for the result. The result gets its direction from the reference and its length from the CONTROLLER.

If you move the CONTROLLER'S parametric point, both the result's length and its distance from the reference change. If you move the CONTROLLER line, the guideline moves to remain parallel.

This CONTROLLER combines several of its properties (line length, direction and parametric distance) to control multiple aspects of its result (distance, length and radial position). It does this by combining controls, for example, both length and distance of the result line are a function of parametric distance along the CONTROLLER. Sometimes, such *interdependence* is both intentional and beneficial. More often though, it can confuse: the result of the CONTROLLER becomes opaque with increasing complexity in its relation to the model. Most usability experts, for example, Don Norman (1988), are highly critical of such linked controls.

Be warned: good CONTROLLERS can be hard to write.

8.7: The point on the control line controls both distance between the two lines and length of the controlled line.



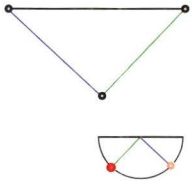
Right Triangle

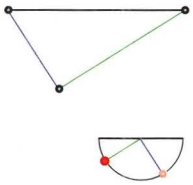
When. Create different right triangles with the same base.

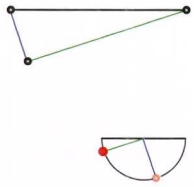
How. The right triangle is a fascinating and useful geometric object. Some of its instances have Pythagorean triples as dimensions; its hypoteneuse is the diameter of its circumscribed circle; it combines to form rectangles; and the sum of its two non-right angles is 90°. Each of these could be the base for a CONTROLLER design.

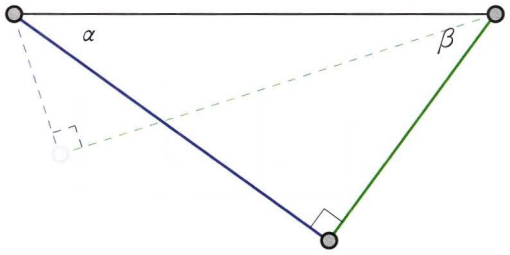
This CONTROLLER uses the last of the above features, by using a half-circle to express the 180° triangle angle sum. The half-circle's base gives the direction and length of the resulting triangle base. Rays between the circle centre and two points on the circle represent the direction of the sides of the triangle. If these two points can move freely on the half-circle, they specify an arbitrary triangle. Presume that the half-circle has a $0-1$ parameter domain. If the parameter t of one of these points is constrained to the domain $0.0-0.5$ and the other to the domain $t+0.5$, the generated triangle will always be right-angled. Further, all right-angled triangles can be reached.

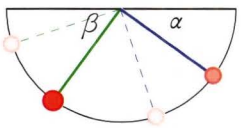
This CONTROLLER reveals that right-angled triangles are but two-parameter objects: the hypoteneuse length and one angle suffice to uniquely determine the triangle up to a rigid body motion. It does visually invert both the angle and side when compared with the result. In reading across both CONTROLLER and model, you encounter the angles and lengths in reverse order. Some visual coherence has been traded for geometric insight.











8.8: Controlling two angles fully determines a triangle if its base is known.

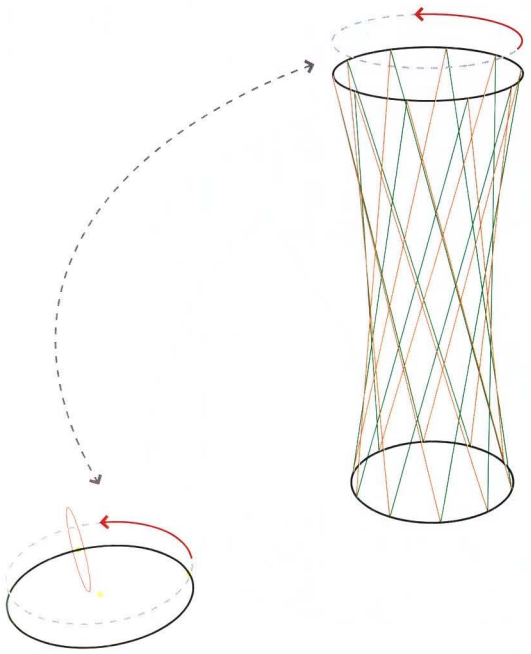
Hyperboloid of One Sheet

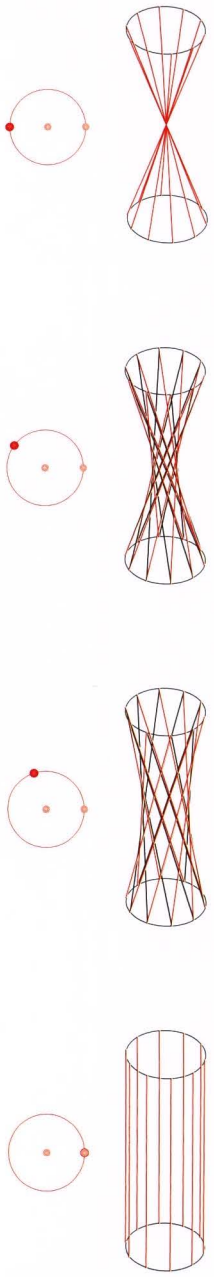
When. Abstract the geometry of a hyperboloid of one sheet to a plane.

How. A *hyperboloid of one sheet* is a ruled surface, that is, it can be formed from a sequence of straight lines. Further, it is doubly ruled: two such sequences can combine into a lattice, giving potential for structurally efficiency. Conceptually, a hyperboloid can be defined by twisting two parallel circles whose centres share a common line normal to the circles.

The hyperboloid's independent parameters are the radii of the two circles and the twist of one circle relative to the other. Starting with the CONTROLLER from the sample, add a twist control to one circle.

Of course, this CONTROLLER has limits. These range from $-180°$ to $180°$ exclusive. A twist of $180°$ turns the hyperboloid into a cone. Two surfaces with twist parameter a and $-a$ are geometrically the same but logically distinct. The difference is that the two sets of generating lines transpose. If one set carries information distinct from the other, the resulting design will differ as well.





8.9: A single point on a circle maps directly to the degree of twist in a hyperboloid of one sheet.

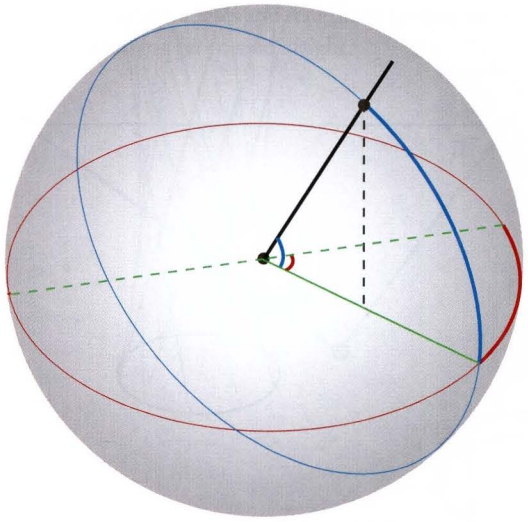
Azimuth Altitude

When. Control a direction by its *azimuth* and *altitude*.

How. The *azimuth* of a point with respect to a reference is the horizontal angle from a reference direction. The *altitude* is the vertical angle from the horizontal plane. As controls, azimuth and altitude are independent: they specify clearly separate changes to a point. Of course, azimuth and altitude relate to two of the dimensions of a *spherical* coordinate system (*azimuth*, *zenith* and *radius*), with $zenith = 90 - altitude$.

An azimuth–altitude CONTROLLER comprises two concentric circles with equal radii: one horizontal and one vertical. A point on the horizontal circle determines both azimuth (where $azimuth = t * 360°$) and the vertical plane on which the altitude circle lies. A point on the altitude circle gives the altitude. The CONTROLLER is easily programmed to report the angles it produces.

In this sample, the model is simple: a pyramid with apex controlled by a line of fixed length and direction given by the CONTROLLER. Four points make the base of the pyramid. The start point of the controlling line is the intersection of the base diagonals. The direction is that of the azimuth–altitude CONTROLLER and the distance from start to end is a predetermined value, set outside of the CONTROLLER, in the model at large.



8.10: A complex CONTROLLER comprising separate controls for azimuth and altitude.

8.7 JIG

PATTERN ALIAS • *Strut* • *Reference*

Related Pattern • CONTROLLER • POINT COLLECTION • PLACE HOLDER • MAPPING

Source: Amy Taylor

What. Build simple abstract frameworks to isolate structure and location from geometric detail.

When. Designers sketch. Carpenters build jigs. Parametric modelers make JIGS. These acts share intent; they abstract away inessential detail, leaving only a simple framework that can be easily changed. Design sketches express structure and form. Carpenters' jigs fix locations and tool paths in space. A parametric JIG mixes both of these traditions. Use this pattern when you want to quickly make and modify a simple version of your design and develop detail later.

Why. Most models contain many elements and a few controls. A JIG reduces the number of elements. It is an abstract model that reveals design structure and control behaviour without the distracting detail and slow interaction implied by a larger model. A JIG can be changed easily compared to a more complex model. Once developed, a JIG can be reused in other contexts, but only if it can be isolated from the rest of the model. JIGS are like abstracting controllers, but they are more specialized (they abstract a particular design). Further, JIGS typically describe the whole design and are embedded within the design rather than being separated from it. The design is built directly on top of the JIG.

How. A JIG should appear and behave as a simplified version of your intended design. A physical example is the strongback and stations used to build a small boat. The stations locate and support the hull when it is being constructed. Fairing, the process of making the hull smooth and continuous, can be done much more simply with a jig of stations than with a complete hull. JIGS are like construction lines in that they help locate elements. They are unlike such lines in that they are linked to the controls that enliven the parametric model.

JIGS typically connect to the model they control more richly than controllers, but still with a limited number of links. Most of these links should come from *sink* nodes. This is not a necessity – it is good programming style. Non-sink nodes capture the internal logic of the JIG. Connections from other than sink nodes run the risk of becoming invalidated when the JIG is refactored. In fact, if a sink node of a JIG is not used in the model it serves, it probably should not be there and can be deleted.

To make a JIG, you need to understand the parametric behaviour you want and how the JIG will be used to define the complete model. A good JIG typically has relatively few geometric inputs (for example, points, lines, planes, coordinate systems) and each of these is carefully named. The small number of geometric inputs allows you to easily locate the JIG. The names are the primary means by which you will understand the JIG when you (or someone else) reuse the JIG in the future.

Use the internal structure of the JIG to capture intended logical behaviour. For example, if the depth of a truss is proportional to its span, a JIG might comprise a line and a variable whose value is proportional to the length of the line.

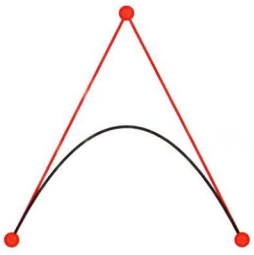
JIG Samples

Controlled Surface Variation

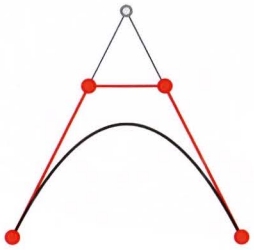
When. Make variations starting from a surface with a parabolic cross-section. Use a JIG to model these variations in a controlled way.

How. Low-order curves and surfaces are easy to model and often display visual regularity that is difficult to achieve with higher orders. An order 3 curve can be represented by a higher-order curve by locating the control points of the higher-order curve in precise relation to those of the lower-order curve. In the curve literature this is called *degree elevation.*

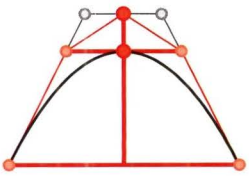
A symmetric JIG comprising an upright and a crossbar provides a simple set of parameters that support controlled surface variations starting from a parabolic curve (see (a) below). To generate the control points of an identical order 4 curve from those of an order 3 curve (see (b)), divide the two sides of the order 3 control polygon in the ratio of 2 : 1 and 1 : 2 respectively. The order 5 control polygon divides the three sides of the 4 in the ratios 3 : 1, 2 : 2 and 1 : 3. Initially locate and size the crossbar to give these ratios. Varying the ratios (d) produces symmetric curves that are visually close to the parabola. Restoring the crossbar settings to the above defaults restores the initial parabolic surface section. This allows the designer to vary the surface cross-section in comparison to a known, simple and potentially fabricatable form.



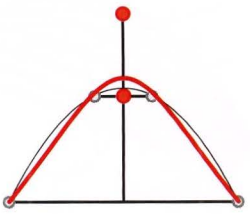
(a)



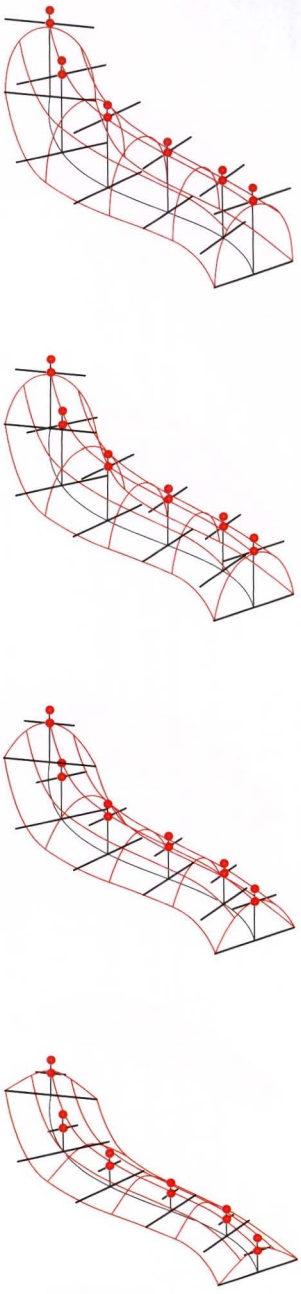
(b)



(c)

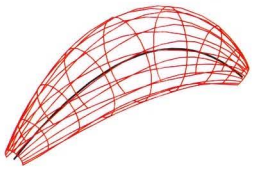


(d)

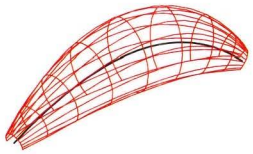


8.11: Two parameters give the overall height and width. Three more control variation away from the default parabola form: the proportional heights of the central points of the order 4 and 5 control polygons and the proportional width of the central points of the order 4 control polygon.

Tube

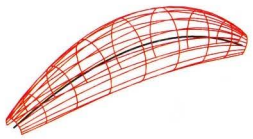


When. Use the local properties of a curve to determine the local radius and orientation of circular JIGS. Use the circles to define a tube. In turn use a curve as another JIG to apply a global form to the tube as a whole.



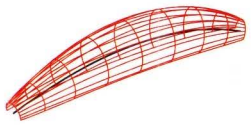
How. Start with a curve as the central path of the tube. See (a) below. It can be specified by four points with almost arbitrary x-, y- and z-coordinates. Along this curve, place a sequence of circles perpendicular to a global axis, here the y-axis. Evenly distribute the circles in parameter space, making the geometric spacing between circles vary along the curve. Each circle takes its radius from a property of its centrepoint, in this case the height above some external datum. In this sample, the radius is the absolute value of the centrepoint's z-value plus a small increment (to avoid the possibility of a zero or negative radius). Since these circles are the elements that construct the tube, they comprise a JIG.

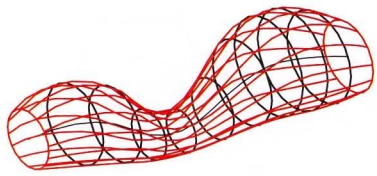
Now (b) JIG the JIG. Make a simple curve using a low-order B-Spline. Substitute it for the existing curve used to define the JIG. The tube now reflects the simple, strong geometry of the curve.



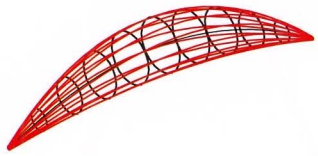
Make (c) arcs comprising those parts of the circle JIGS above the xy-plane.

Lastly (d), change the planes on which the circle JIGS lie to be perpendicular to the defining curve, resulting in a subtle, but significant change to the tube's form.

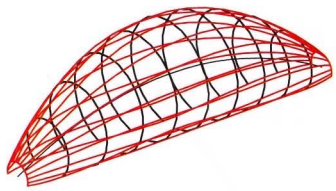




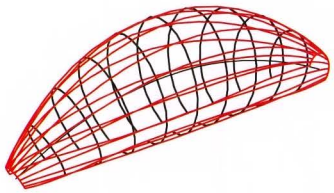
(a)



(b)



(c)



(d)

8.12: The control polygon for the JIG curve comprises three points only. In this sequence the middle point of the control polygon moves in all of the x-, y- and z-directions.

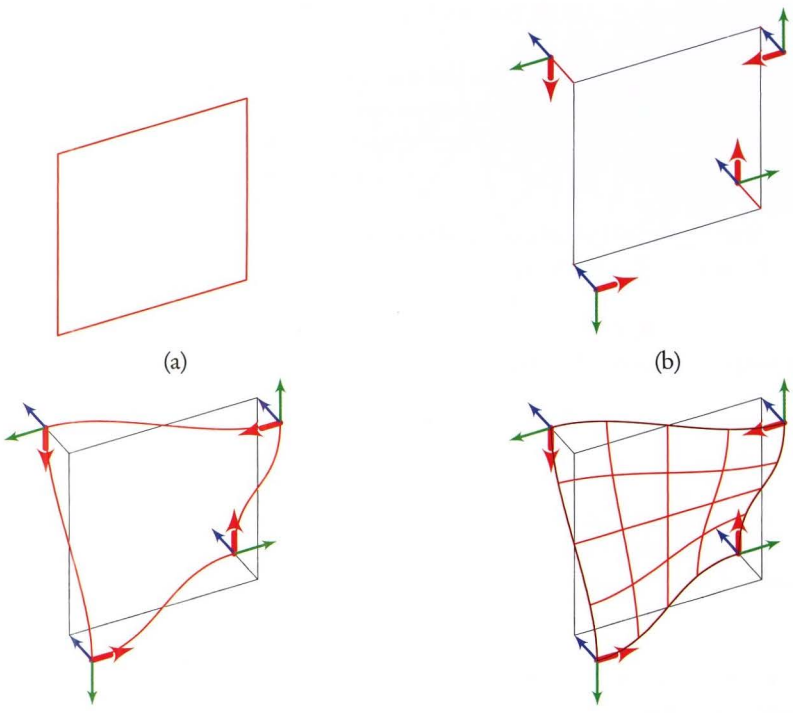
Sheet

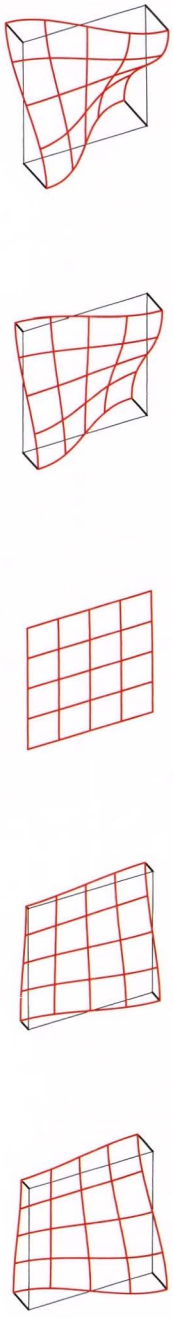
When. Simplify controls for a surface by relating them to a quadrilateral.

How. Standard surface controls can provide too much freedom. This sample reduces the control available to model a surface by ensuring corner tangency conditions. It still provides a wide range of visually logical variation.

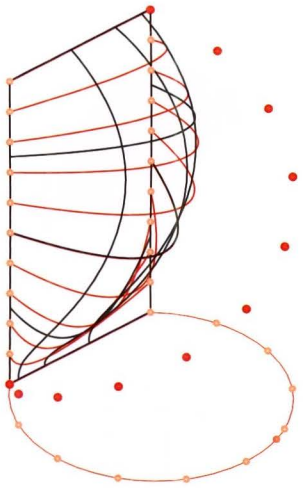
The JIG is hierarchical – it comprises JIGS built on JIGS, as shown below. The first JIG (a) is a quadrilateral, which may be planar or not. The second JIG (b) has two parts. The first comprises struts at each vertex, each perpendicular to the local plane of the quadrilateral (defined by the vertex and its predecessor and successor vertices). The second adds frames at the end of each strut, such that the x-axis of the frame aligns with the successor vertex and the y-axis with the predecessor vertex, but in the opposite direction (the quadrilateral has right-hand rule orientation, so the frame's y-axis has the same direction as the vector from the predecessor vertex to the vertex itself). The third JIG (c) comprises curves with end tangents defined by the x- and y-directions of the frames. The result (d) is the surface itself with the curves as its defining boundary.

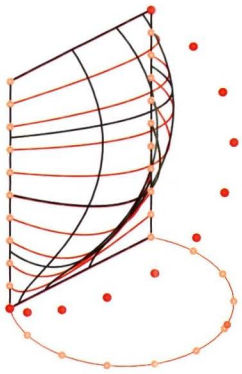
The controls for this JIG comprise the quadrilateral itself and the four strut lengths. Each enters the system at a different level of the JIG.

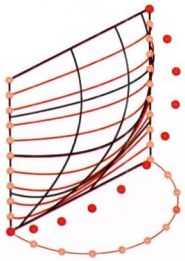




8.13: Four lengths, one for each of the corner struts, are sufficient to access a wide range of surface geometry.







8.14: Developed initially from the constant angle subtended from a chord on a circle, the JIG for this design opens a space of related designs.

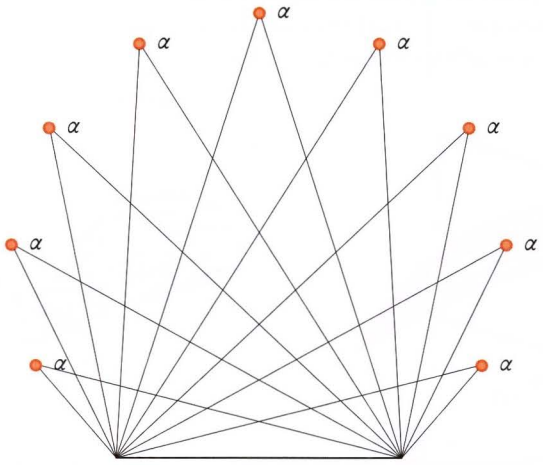
Scallop

When. Use the shape of a scallop as a point of departure in a search for form. The actual surface will be analogous to, but not a copy of, a scallop.

How. In plan, the geometry of a scallop is approximately that of a circular arc. The base of the scallop is a chord of the arc. Any point on the edge of the circle will subtend a constant angle with the base.

The idea is to "open" up the base of the scallop – to turn it from a line into a vertical rectangle. This JIG comprises a sequence of triangles on horizontal planes arrayed vertically from the base line. The apex of each triangle is the projection of a point on the circle onto the plane of the triangle. This JIG has three parameters: the angle subtended on the circle, the spacing of base points on the circle and the vertical spacing of the JIG elements. CONTROLLERS could be put on each of these to open a design space for the surface.

The generating triangles are actually modeled as order 2 B-Spline curves. This and the order of the surface itself give two additional controls. The resulting forms are far from the original scallop point of departure.



8.8 INCREMENT

Related Pattern • POINT COLLECTION • MAPPING

What. Drive change through a series of closely related values.

When. Parts may be similar in structure but vary in their inputs. Very often, input variations are gradual from part to part and parts in sequences or other arrangements are similar to their neighbours. Use this pattern when you are making collections of related parts.

Why. Being able to relate and edit parts through gradually changing inputs lends surety and control. As a form-making strategy, gradual change provides a background against which a strong figure can play.

How. Gradual change occurs in two forms. The first is the integers, stepping in units of one from low to high,

$$\ldots -1, 0, 1, 2, 3, \ldots$$

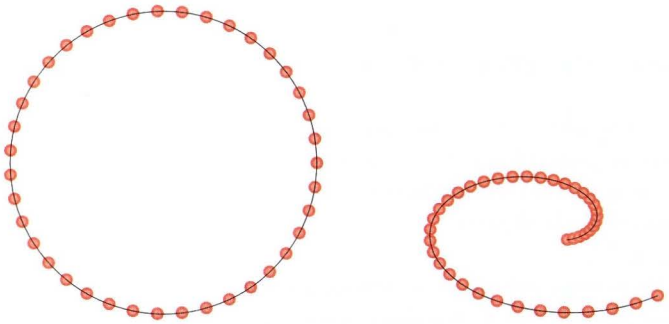
The second is the reals, varying continuously (infinitely divisible). Taken by themselves, the integers and reals can express only limited kinds of change. Functions transform sequences of integers and sampled reals into new sequences that may be dramatically different from the originals.

In turn, an INCREMENT uses the output of a function to drive change in any of a variety of ways, limited only by imagination. Length, size, angle, orientation, distance, colour, transparency and surface texture can all be changed in orderly (and disorderly) ways through incrementing along sequences of integers or reals.

The samples in this pattern develop increasingly complex curves traced by a single point moving through space. Each successive sample increases both the number of parameters on which an increment applies and the complexity of the incrementing functions. Throughout each sample, the structure of the model remains constant; only the values of the parameters change.

Even a single point can demonstrate the basic structure of an increment. Start with a point in space, located as it must be with respect to a coordinate system.

The point can be thought of having either Cartesian (x, y, z) coordinates or cylindrical (r, θ, z), where r is the radius, θ is the azimuth angle and z is the height of the point. Use cylindrical coordinates and increment the azimuth angle θ to make the point trace out an arc. If the azimuth angle increments from 0° to 360° the arc becomes a circle. Increment the radius to turn the arc into a spiral.



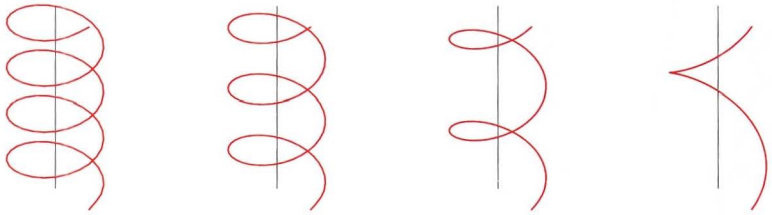
Incrementing the height of the point turns the arc into a helix and brings us to the first sample below.

INCREMENT Samples

Circular Helix

When. Move a point uniformly around a centre and upward in space.

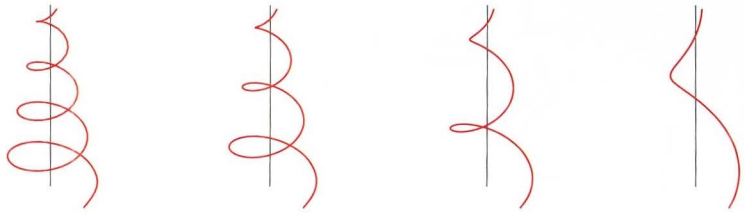
How. As a point moves around the circle, increment its height by a uniform amount. The result is to trace out a simple circular helix.



Conic Helix

When. Add a reducing radius increment to change a circular to a conic helix.

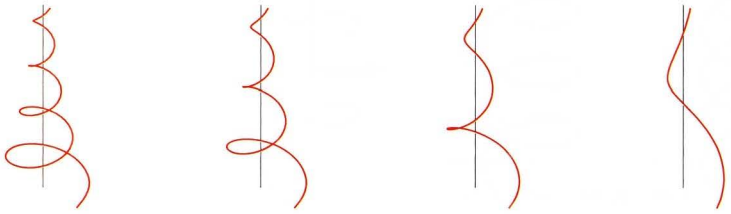
How. In addition to the two increments (angle and height) for a circular helix, reducing the radius incrementally from an initial value to a minimum value produces a conic helix, that is, a helix whose points lie on a cone.



Tapered Radius Spiral

When. Taper the radius of a conic helix to produce a spiral.

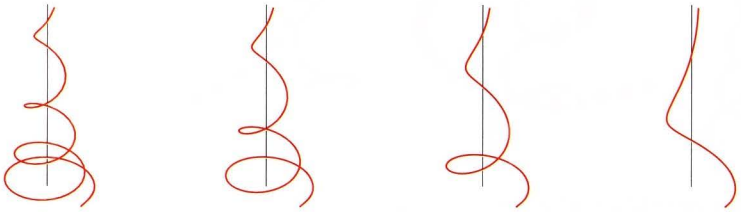
How. As a point on a conic helix moves upward its radius shrinks. The point can be imagined to have a parameter that is 1 at the helix base and 0 at the top. Squaring this parameter will still result in a series that goes from 1 to 0, but the series will taper across this interval. Mathematically, the curve changes from a helix to a spiral.



Tapered Height Spiral

When. Taper the height of a conic helix to produce a spiral.

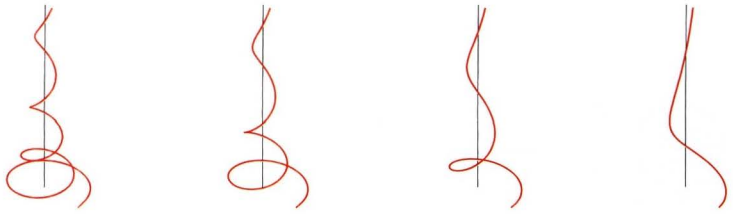
How. Instead of tapering the radius, taper the height with the same device, by squaring the parameter. In this case, the parameter is 0 at the helix base and 1 at the top. The helix, now a spiral, appears to have been differentially stretched from its base to its top.



Tapered Radius and Height Spiral

When. Combining increments yields unpredictable forms.

How. Combining both radius and height tapers can be done independently in the model. They do not affect each other computationally, but combine in the geometric result. They produce a spiral that would be hard to conceive of itself, but naturally emerges from the parameterization.

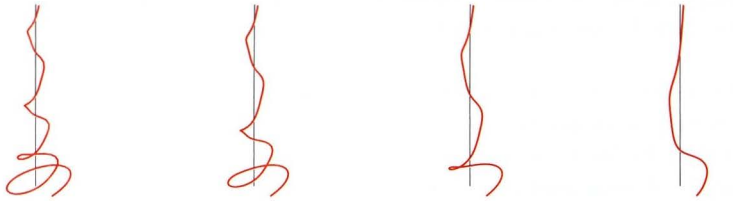


Elliptical Tapered Radius and Height Spiral

When. Change a circular spiral to an elliptical one.

How. In the prior samples, the radius, angle and height were independent in the model. In this sample, the radius becomes a function of the angle, by using a polar equation for the radius of an ellipse. If an ellipse has major axis of $r = 1$ and minor axis of $s = 0.5$, the radius as a function of θ is

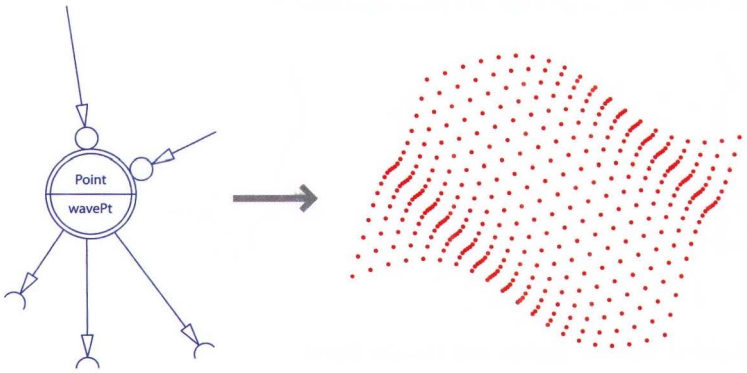
$$\frac{rs}{\sqrt{r^2\cos^2\theta + s^2\sin^2\theta}} = \frac{0.5}{\sqrt{\cos^2\theta + 0.25\sin^2\theta}}$$



8.9 POINT COLLECTION

PATTERN ALIAS • *Point Set* • *Point Grid*

Related Pattern • CONTROLLER • JIG • INCREMENT • PLACE HOLDER • PROJECTION • RECURSION



What. Organize collections of point-like objects to locate repeating elements.

When. Most designed artifacts have repeating elements. These may vary by both their absolute position and by their spatial relationships with nearby repeating elements. Use this pattern when you are able to think about the size and location of repeating elements in terms of a set of defining points.

Why. A collection of points organized to capture intended spatial relationships can greatly simplify the process of further model development. This saves time and effort in both modeling and reuse of a model in new contexts.

How. Point-like objects may be located in Euclidean space or parametric space, so a collection can be specified in either space. Euclidean space is the familiar space of everyday life. It can be represented through Cartesian, cylindrical or spherical coordinates. Most curves and surfaces (those defined internally by parametric equations) define a moving frame that gives locations on the curve or surface. Unlike those of Cartesian space, these parametric formulations may not preserve constant distance, either geometrically or along the defining object.

Use a collection of point-like objects as the input to define repeating elements. The logical structure of a collection is important – it provides the relationships through which points can be used to define objects. For instance, a collection structured as 2D array provides for each point P_{ij} easy access to the surrounding points, that is, p_{gh}, where $g \in \{i-1, i, i+1\}$ and $h \in \{j-1, j, j+1\}$. In comparision, a collection structured as a tree provides for each point P, easy access to parent(P) and children(P).

The following samples specify POINT COLLECTIONS with functions. A given parametric system will provide its own particular commands for organizing collections, for example, replication in GenerativeComponents. The uniform and universal function notation here allows comparison among the samples.

POINT COLLECTION Samples

Spiral

When. Place a sequence of points along a spiral.

How. A spiral is a curve that turns around an axis at a continously varying distance perpendicular to the axis. Spirals admit many parameterizations: this sample uses count, heightStep and radius. Count controls the number of points in the collection. HeightStep is the height increment between sequential points, not the entire height of the spiral. Radius decides the outer radius of the spiral: the distance from the first point of the spiral to the central axis. The function below generates a spiral.

The update method ByCylindricalCoords, which generates the actual spiral points, takes four arguments: a coordinate system, the point's distance from the origin, the point's angle of rotation from the x-axis, and the point's height above the xy-plane.

```
function spiral (CoordinateSystem cs,
                 int count,
                 double radius,
                 double heightStep)
  {
    Point spiralP = {};
    double radiusInt = 0.0;
    for (int i = 0; i < count; ++i)
      {
        spiralP[i] = new Point();
        radiusInternal=radius*(1—Pow(i/count,0.5));
        spiralP[i].ByCylindricalCoordinates(cs,
                                            radiusInternal,
                                            30.0*i,
                                            i*heightStep);
      }
    return spiralP;
};
```

8.15: Few can predict the form of the spiral from its parameters alone. In form-finding, designers typically iterate through cycles of coding and parameter play.

8.16: Several point collections, each along the positive arc of a parabola, each scaled by a real parameter. As this scale parameter increases, so does the slope of the parabola. The model limits the output range from zero to a set maximum value. The collections' input parameters are spaced so that each has an equal number of points.

Parabola

When. Arrange a sequence of points along a parabola.

How. Simple mathematical functions pervade the modeling act. Functions must be described mathematically to work at all, and it pays to use evocative names for their variables. For example, the parabola $y = kx^2$ scales the most simple parabola $y = x^2$ in the y-direction by the factor k.

Placing count points along the parabola yields both the POINT COLLECTION and its organization as a linear sequence. Algorithmically, a *for-loop* steps through the points, adding each at the end of the sequence in turn. Sampling at count equal intervals along the domain of the parabola function yields an unequally spaced collection of count points. Thus the function below generates a POINT COLLECTION as a sequence along the parabola.

```
function parabola(CoordinateSystem cs,
                  int count, double scale)
{
 Point pointOnParabola = {};
 for (int i=0; i < count+1; ++i){
  pointOnParabola[i] = new Point();
  pointOnParabola[i].ByCartesianCoords(cs,i,0.0,scale*i*i);
 }
 return pointOnParabola;
};
```

At the risk of repetition, this collection is a sequence – an array. Its members thus have indexes, that is, integers giving each member's position in the array. Members of pointOnParabola can thus be addressed as "pointOnParabola[i]", where i = 0...count—1.

Designers are often more interested in controlling the output range over which a function is used rather than its input domain. For example, to place a sequence of points along the part of a parabola below a given upper limit requires that the input to the function be scaled as in the following code.

```
function parabolaInRange(CoordinateSystem cs,
                         int count,
                         double scale, double range)
{
 Point pointOnParabola = {};
 double xStep = Sqrt(range/scale)/count;
 double x = 0.0;
 for (int i=0; i < count+1; ++i){
  x = i*xStep;
  pointOnParabola[i] = new Point();
  pointOnParabola[i].ByCartesianCoords(cs,x,0.0,scale*x*x);
 }
 return pointOnParabola;
};
```

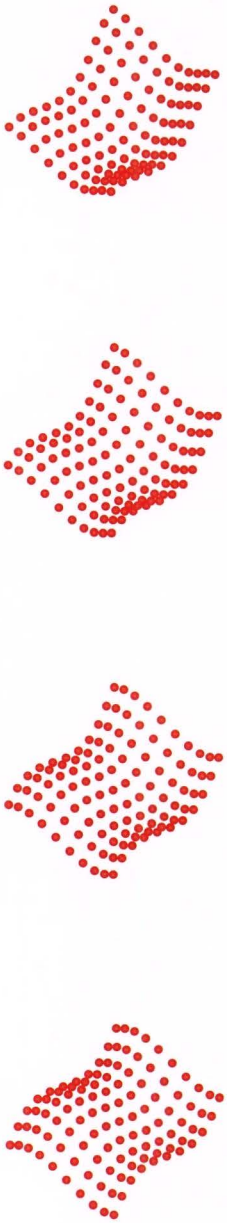
Waves

When. Simulate a waveform with a two-dimensional collection of points.

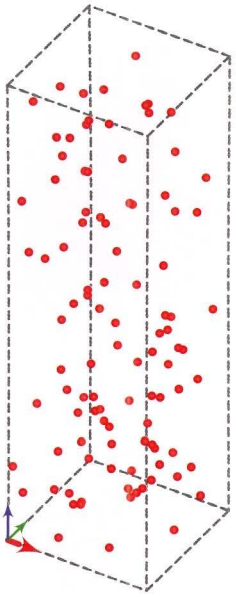
How. POINT COLLECTIONS in the previous two samples are one-dimensional. A two-dimensional collection can be organized as an *two-dimensional array*, an array of arrays. This sample demonstrates how to create such a two-dimensional collection. The generating function $f(x, y)$ is a sum of two sine functions, with two arguments taken respectively with domains along the x- and y-directions. The particular parameterization here comprises count the number of points in each direction (and the dimensions of the array), size the geometric extent of the collection in the x- and y-directions, amplitude the height of the wave function and startAngle the angle at which the sine curves starts.

A pair of nested *for-loops* makes the algorithm step through the points, row by row, defining each in turn. The structure of the algorithm maps directly to the structure of the collection!

```
function wave (CoordinateSystem cs,
               int count,
               double size,
               double amplitude,
               double angleStart)
{
 Point pt = {};
 double anglei = 0.0;
 double anglej = 0.0;
 double ordinate = 0.0;
 for (int i = 0; i<=count; ++i)
  {
   pt[i] = {};
   anglei = (i/count)*360 + angleStart;
   for (int j = 0; j <count; ++j)
    {
     pt[i][j] = new Point();
     anglej = (j/count)*360 + angleStart;
     ordinate = Sin(anglei) + Sin(anglej))*amplitude/2;
     pt[i][j].ByCartesianCoordinates(cs,
                                     (j/count*size),
                                     (i/count*size),
                                     ordinate);
    }
  }
 return pt;
};
```



8.17: The collection iterates through a complete sine curve in both parametric directions. The parameter angleStart has the greatest effect on the resulting form; it picks the place on the sine curve where the cycle begins.

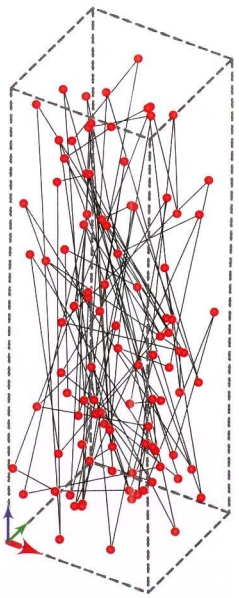


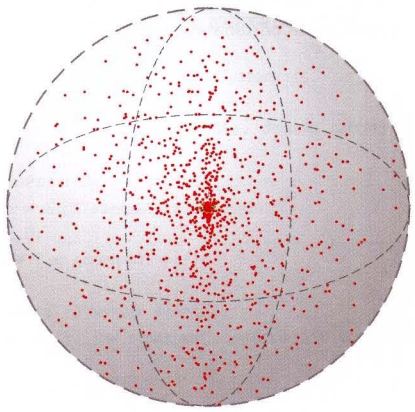
Point Cloud

When. Create a collection of random points to uniformly fill a volume.

How. The geometric and symbolic structures of collections need not be the same. Here, the symbolic structure is a sequence and there is no geometric structure, just randomness. This sample places uniformly-distributed random points within a rectangular bounding box. Its parameterization gives count, the number of points; lowerLeft, a frame defining the lower-left corner of the bounding box; and boundX, boundY and boundZ, reals that give the location of the upper-right corner of the bounding box. In this special case, the range of the function is a rigid body transformation of the domain. This means that the uniform distribution defined in the domain will persist into the range. Imagine using a random distribution in spherical coordinates. The points are random, but not uniformly distributed!

```
function cloud (CoordinateSystem lowerLeft,
                int count,
                double boundX,
                double boundY,
                double boundZ)
{
 Point randomP = {};
 for (int i = 0; i < count; ++i)
  {
   randomP[i] = new Point();
   randomP[i].ByCartesianCoords(lowerLeft,
                                Random(0.0,boundX),
                                Random(0.0,boundY),
                                Random(0.0,boundZ));
  }
 return randomP;
};
```





8.18: A random sequence likely has little utility in design (but designers always surprise us). This sample shows that symbolic and geometric structures may have any kind of relation, including the null relation of randomness.

Points on a Parametric Curve

When. Position a sequence of points along a parametric curve.

How. A parametric curve provides both a curve and a way to place points along it. A sequence of points along the curve at intervals of its parameter t yields a collection of points in which the symbolic successor of a collection point is the geometric successor of the corresponding curve point. The sequence can be made either by replication or an explicit function (shown here). Section 4.10 introduces replication as a convention of specifying a collection of values for a node property. The collection causes the system to generate an object for each item of the collection in nodes using the replicated property.

```
function pointOnCurve (Curve curve, int count)
{
 Point p = {};
 double tStep = 1/(count-1);
 for (int i = 0; i < count; i++)
  {
   p[i]=new Point();
   p[i].ByTParameter(curve, i*tStep);
  }
 return p;
};
```

8.19: A collection organized by its members' parametric position on a curve.

Points on a Parametric Surface

When. Array points on a parametric surface.

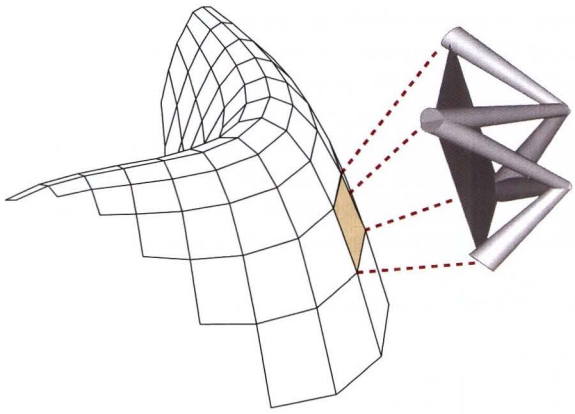
How. Analogous to a parametric curve, a parametric surface provides both a surface and locations on it through the parameters u and v. This gives a natural organization for the collection as an array of points, with neighbours in the array corresponding to neighbours on the surface. As with a curve, a surface can be generated by replication or by an explicit function.

```
function pointOnSurface (Surface surface, int uCount, int vCount)
{
 Point p = {};
 double uStep = 1/(uCount-1);
 double vStep = 1/(vCount-1);
 for (int i = 0; i < uCount; i++)
  {
   p[i]={};
   for (int j = 0; j < vCount; j++)
    {
     p[i][j]=new Point();
     p[i][j].ByUVParameters(surface, i*uStep, j*vStep);
    }
  }
 return p;
};
```

8.20: A collection organized by its members' parametric positions on a surface.

8.10 PLACE HOLDER

Related Pattern • JIG • POINT COLLECTION



What. Use proxy objects to organize complex inputs for collections.

When. Designs have parts. A single model may represent many variations of a part, for example different window designs. An effective modeling strategy copies the model, one copy for each part, and adjusts the model inputs to each copy. Typically a part has multiple inputs – customizing each one is a lot of work. Use this pattern when you are able to describe the multiple inputs to a model through a smaller number (preferably one) of abstract proxy objects.

Why. A very common scenario arrays a module across a target surface or along a set of curves. If this module requires point-like inputs themselves defined on the target, organizing these inputs is sure to be complex and error prone. If you can define the inputs to the complex module through a simple construct such as a polygon, it is often much easier to place the module. An arrangement of polygons on the goal surface creates proxies on which the module can be later (and easily) placed.

How. PLACE HOLDERS have two parts. First is the proxy: a simple object that carries the module inputs. For example, a rectangular module requires four input points, one for each corner. A four-sided polygon can act as a proxy for these points: each of the vertices of the polygon provides one of the points. The proxy simplifies the arguments provided to the module: instead of four points, use only one polygon. The second part relates the proxy object to the model. For example, a polygon proxy can be placed using a rectangular array of points: the ij^{th} polygon's vertices are the points $\dot{p}_{i,j}$, $\dot{p}_{i+1,j}$, $\dot{p}_{i+1,j+1}$ and $\dot{p}_{i,j+1}$. The code placing a generic object such as a polygon is more simple and reusable than the code for a specific module.

PLACE HOLDER Samples

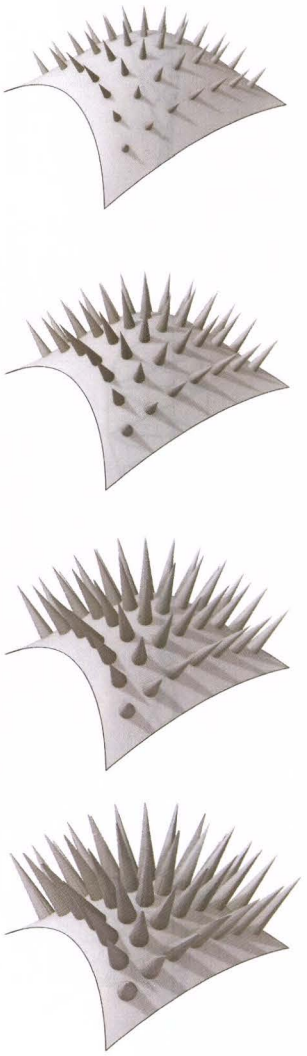
Hedgehog

When. Use a POINT COLLECTION as a PLACE HOLDER to locate and orient components (spines) that are perpendicular to a surface.

How. Every point on a surface defines a single frame comprising the surface normal and the vectors of principal curvature. This is sufficient information to place and size spine-like objects on the surface. The point provides location; the surface normal provides the direction for the spine; and the vectors of principal curvature provide information for further adapting the spine to context. Make a POINT COLLECTION structured by its u and v point-on-surface parameters. Instead of points, use frames – remember they have points inside them! Each of the frame points will serve as the base of a spine. Define two graph variables *count* and *height*. The POINT COLLECTION produces *count* frames in each parametric direction. At each of the frames, use the frame's z-direction and the parameter *height* to define a cone.

Kunsthaus Graz, Austria, by Peter Cook and Colin Fournier

Source: Anita Martinz



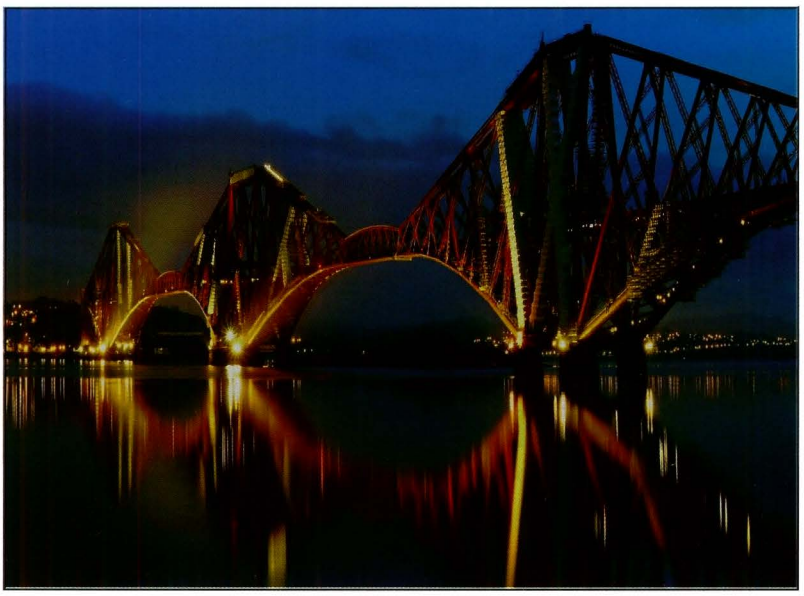
8.21: This simple PLACE HOLDER uses frame located on a surface to hold geometric information for placing cones.

8.22: A simple representation for the *Firth of Forth bridge* comprises three long and two short lines. These act as PLACE HOLDERS for more complex representations of the bridge segments.

Truss

When. Use lines as PLACE HOLDERS to locate the members of a truss.

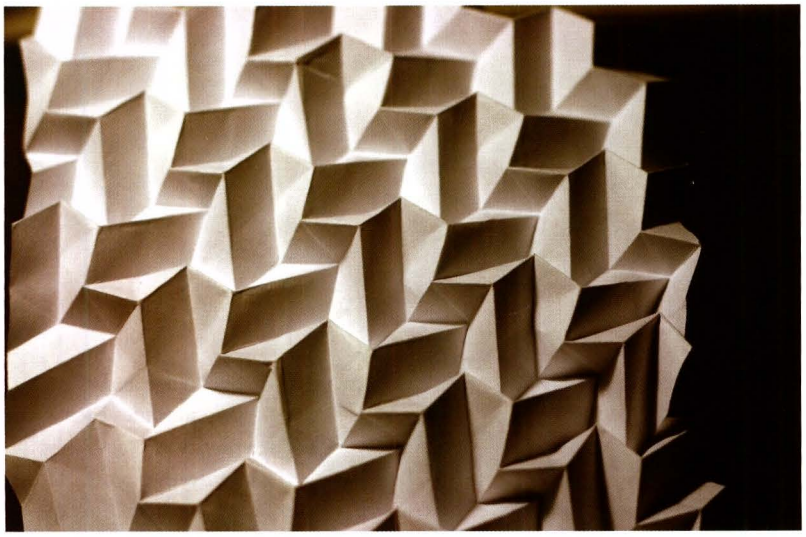
How. Each member of a truss might carry information such as the member section, material, moment of inertia and modulus of elasticity. In addition, the parametric model for truss members may be able to shape its ends depending on the context in which it is placed. Placing a truss member though requires only the baseline along which the member lies. First, develop a feature representing a truss member and requiring only a line as a geometric input. Second (and in a new model!), create an abstract truss comprising line segments to represent the truss members. Applying the truss member feature to these baseline PLACE HOLDERS, places the detailed truss members. Of course, this simplifies a real truss member PLACE HOLDER in which the truss member parametric model would need sufficient information to shape its sectional properties and details. Taking this next step would require that the PLACE HOLDERS become spatially more sophisticated and that the truss member feature use that new information to specify its details.



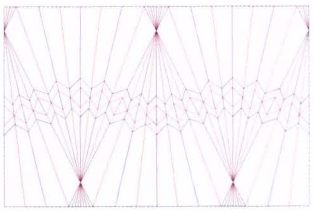
Firth of Forth bridge, Scotland
Source: Kenneth Barker

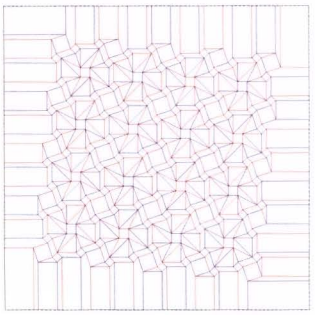
Paper Folding

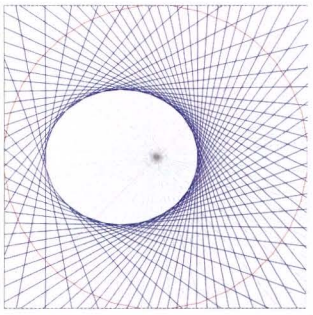
When. Use quadrilaterals as PLACE HOLDERS to simulate origami.



Source: Ray Schamp







8.23: Some origami folding patterns.
Source: Ray Schamp

How. Parametrically modeling folded paper is hard. The problem is physics – paper has actual dimensions and folds in it constrain the spatial configurations it can achieve. These real-world constraints inevitably imply that any model will require the solution of simultaneous equations, which propagation-based systems cannot do. (The GOAL SEEKER pattern gives a partial solution to this problem.) That said, design sketching is approximation and this sample shows a way to simulate a folded paper system, ceding from reality some dimensional variation in the individual panels.

In a folded structure, the pattern of folds can be thought of as separate from the size and location of the folded panels. Further, the folding pattern will belong to one of the 17 possible symmetry groups on the plane (each group represents one of the fundamentally different ways of arranging a collection of like motifs on the plane (Grünbaum and Shephard (1987, pp. 37-45); Weisstein (2009))). In each such group, there is a repeating module that imposes geometric conditions on where the paper edge must be to connect to the next module. The modeling task splits into three parts: the paper folds, ensuring geometric connection at the joints and arranging the resulting module across a surface.

The choice of module is key to clarity and simplicity. This sample comprises a collection of identical parallelograms (for symmetry *aficianados,* arranged in symmetry group *pmg* in crystallographic notation). It is much simpler though to combine parts of six parallelograms to form a module needing only simple