Programming Form: Algorithmic Explorations of Space

by

Joshua Zabel

B. ENVD. (University of Colorado, Boulder) 2001
M. ARCH (University of California, Berkeley) 2005

A thesis submitted in partial satisfaction of the requirements for the degree of

Master of Architecture

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Roddy Creedon, Chair
Professor Lisa Iwamoto
Professor Anthony Burke
Professor Carlo Sequin

Spring 2005

# Contents


Acknowledgements i

1. Introduction
    i. Digital Morphogenesis 1
    ii. Programming Form 2
    iii. Algorithms 2
    iv. Intelligent Agents 3
    v. Human-Machine 4

2. Cases
    i. Geometry - Serpentine Pavilion 7
    ii. Versioning - Embryologic House 8
    iii. Form as Information - Dynaform 9
    iv. Form as Diagram - Yokohama Port Terminal 10
    v. Algorithm - Hybrid Tower and other projects 11

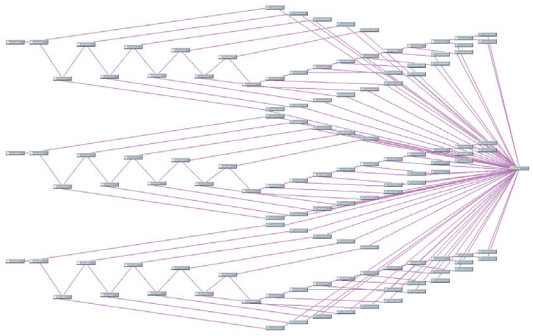
3. Project
    i. Site 12
    ii. Scripts 16
    iii. Data Model 43
    iv. Mapping Form to Data 49
    v. Structure 54
    vi. Skin 62
    vii. Building 65

4. Conclusion 78

Notes 81

Bibliography 83

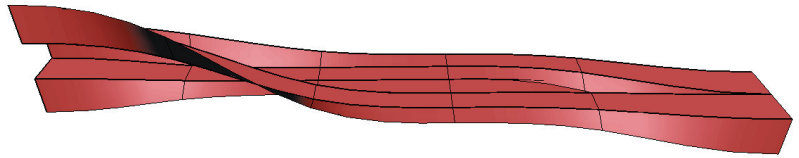
Appendix A: MEL scripts
    i. General tools 85
    ii. Automation 110
    iii. Relationships 116
    iv. Behavioral 120

Thankyou()
{
	my Family

	Alexis
	Anthony
	Carlo
	Erik
	Jess
	Leo
	Lisa
	Padma
	Roddy
	Stanley
}

# I. Introduction



## i. "Digital Morphogenesis"

"In contemporary architectural design, digital media is increasingly being used not just as a representational tool for visualization but as a generative tool for the derivation of form and its transformation - the digital morphogenesis." [1]. Further, the paradigm which "digital morphogenesis" represents is redefining the way that architects and designers visualize and think about problems. The tools and techniques that have defined architecture in the past have transformed into a new language, a new morphology - that of the electronic, the informational, the digital. It has become critical for architects and designers to understand the nature of this environment and engage a new set of tools and new way of thinking in order to successfully design and build within it.

ii. Programming Form

The software being used by designers is a consequence of programming and the programs themselves are embedded with languages as tools which enable their own modification. With programming as a medium for the coding of ideas, this design project explores the spatial applications of this scripting environment and it's implications as a technique for designing a building. Central to this research are the questions of how we encode intelligent decisions and how, as designers, we mediate between the role the computer plays and the role we play in the design process.

iii. Algorithms

Algorithms are a method of solving a problem using a step-by-step set of instructions. They have a long historical legacy as part of an architectural design methodology but computers are providing a much more powerful platform for their role. Architecture has always involved systems of rational decision making, and synthetic constraints to frame thoughts. The Golden Mean is a proportional set of rules (essentially an algorithm) which guided the composition of the Acropolis.[2] Le Corbusier developed his own sets of rules and codes which describe their approach to architectural design.[3] As digital means of exploring design infiltrate architecture, the language in which these systems of rationalizing design decisions are being expressed is the algorithm. The result is a synergy between the human-based desire and computer-based computational power which is enabled by this language.

## iv. Intelligent Agents

Given the presence of a new set of tools and a new set of techniques the focus shifts to what architects are trying to do with them, explicitly or not. Berkel and Bos state: "... techniques form the bridge between abstract thought and concrete production. This is a two-way bridge: techniques also form thought." [4] In other words, technique is the language of how we express our ideas. If coding is a central technique of the use of existing design software, then, in the act of changing that design environment towards architectural ends, we are inscribing an architectural intelligence onto that very environment.

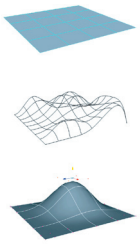
As computers become increasingly powerful, we have shifted the emphasis of their use from automation (the loom) towards intelligent behavior (AI, Bayesian networks, expert systems, etc). The nature of that intelligence is subject of much debate: Do computers only act intelligently, or are they capable of being autonomously intelligent in their own right? [5] If computers are capable of either, it will inevitably have an enormous impact on the way things are made, buildings in particular. Those impacts are the subject of this design project, and hopefully, address what is at stake for architectural design. And while we are at it, ask the question, in what ways must the emerging architect mediate between their own intelligence and the intelligence they encode in the process of designing? How does this change the responsibility architects have to society?

v. Human - Machine

While working on this project there was a constant effort to codify some sort of intelligence, which in effect passes a level of decision making off to the computer. For example, the `createPoolRoof()` MEL script [6] is capable of generating as many viable versions of a structure as one can invent combinations of variables to describe it's parameters (that is, millions of versions). At that juncture, when there are countless versions to choose from, a decision must be made about which ones are most viable and should be developed in the next stage of design. So the question to the scripting-enabled designer is whether or not the information which measures the performance of a version of the structural system is encoded in the script. It is at this critical moment that the designer must, in some way, be able to express desire in code. The measure of performance is expressed in an algorithm. Here, the reigns are released and the computer has, to some degree, taken control of the design by virtue of the fact that performance or desire are being expressed in machine language. In the case of this example with the roof structure system, that value was not encoded. Rather, the decision was left to my intuition about which combination of parameters best suited the goals of the program.

There is an increasing effort by programmers, artists, and architects to digitally replicate biological models.[7] This may come in the form of a genetic algorithm, a swarm, or just a simple set of instructions that mediate the evolution of a system or make a decision. By designing architecture in terms of digital representations and virtual models, the rules, codes, and limitations of the method become practiced to the point of internalization. At the moment of appropriation of the rules, assumption of their existence, and advancement of design based on that

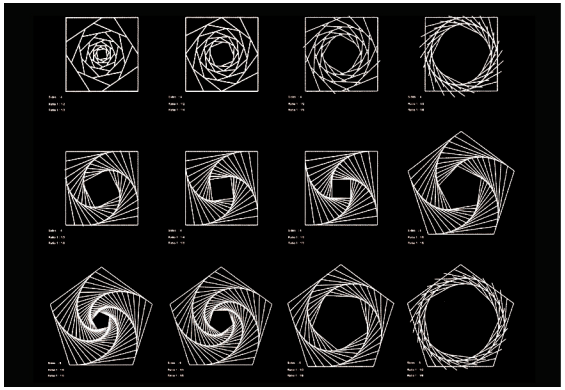
assumption, the machine has taken a step towards a biological model (autonomous intelligence?) and the designer has taken an equal step towards mechanization.

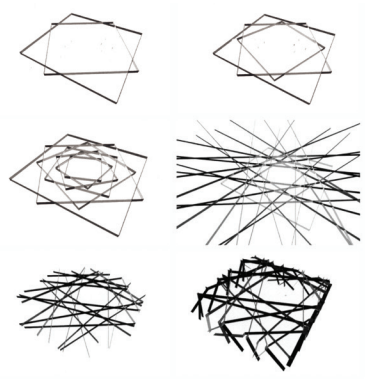


# 2. Case Studies / Precedents

The following cases are examined as historical precedents for my own research and exploration in the design of the Aquatic facility. For each I briefly describe the project, or the part of the design process which is of particular relevance and influence. I also explain why it is of interest and how it will relate to my project.

These projects define, in part, a context for a type of architecture that has begun to emerge from contemporary digital means of design. I have drawn from these cases a set of tools which are the form work for my own process and I have categorized them as Algorithm (Serpentine Pavilion), Versioning (Embryologic House), Form as Information (Dynaform), Form as Diagram (Yokohama), and Repetition & Growth (Hybrid Tower). Furthermore, each project, with the exception of the Hybrid Tower, has a unique and equally digitally-heavy means of fabrication. In some cases the fabrication information is intrisically tied to the very processes which were used to generate the form. Although my project is not taken to such a level of construction, the implications for deriving fabrication information from the model via scripting is an interesting potential.

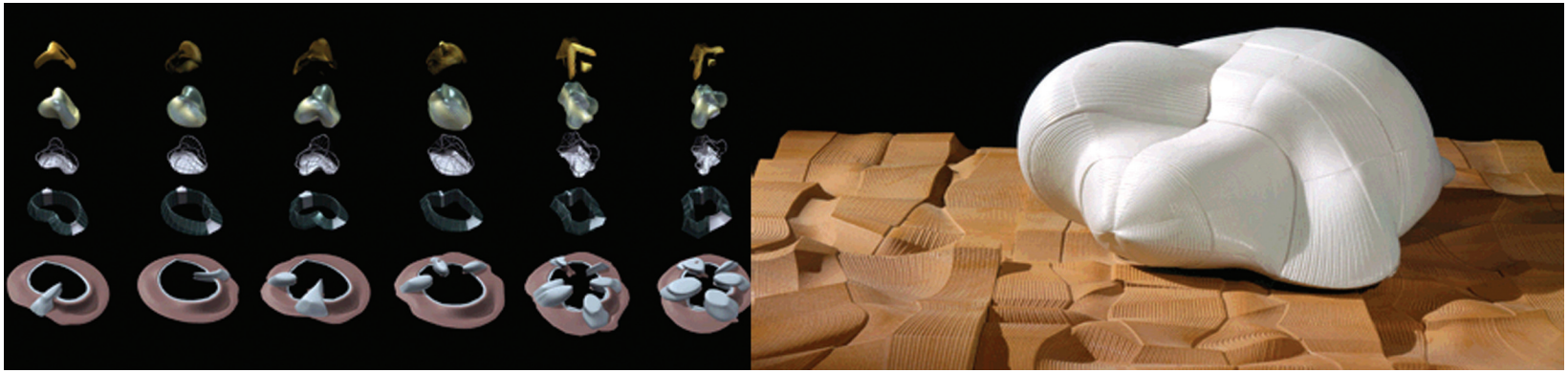




i. *Geometry*

The Serpentine Pavilion - Toyo Ito and ARUP - 2002

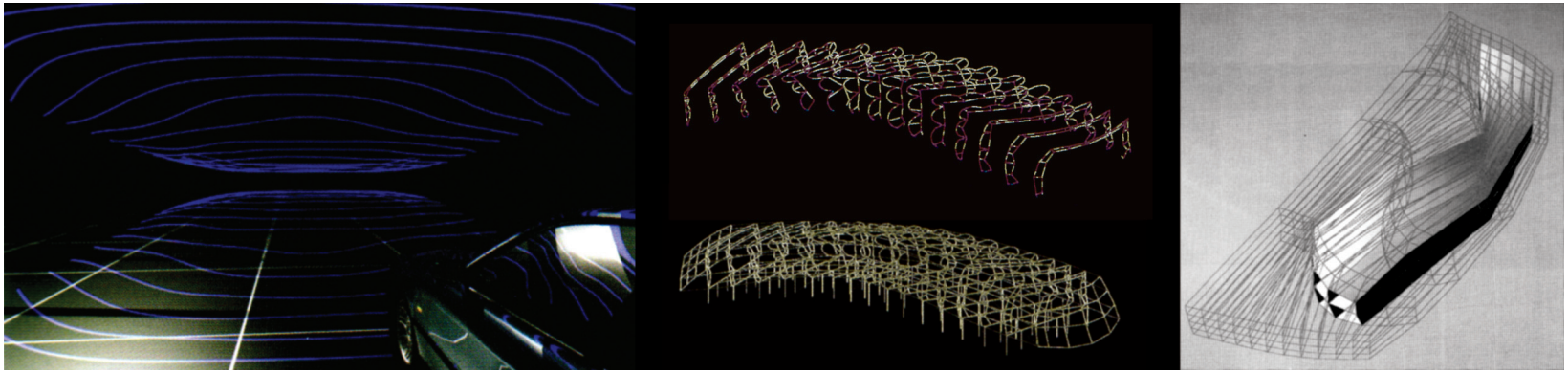
A proportionate rule system was used to construct the geometry for the Serpentine Pavilion. Feedback techniques and algorithms were employed to generate a rotating square pattern which becomes the framework for the roof and folds down to become the walls. The algorithm was developed by Daniel Bosia of Arup. It is designed to encode a system of rules which are parameterized by ratios, scale, and proportion. Different versions of the pattern were generated by the algorithm and the performance of the result was judged by the priorities of the program. Those priorities involved transcending the traditional notion of slab on columns and a play between positive and negative.[1] In the end we have a fanciful pavilion roof that does not seem to have any apparent effect on the behavior of it's users, nor can the users effect the pavilion. It is such effects which, I hope, algorithmic techniques can eventually achieve.



ii. *Versioning*

Embryologic House - Greg Lynn - 2000

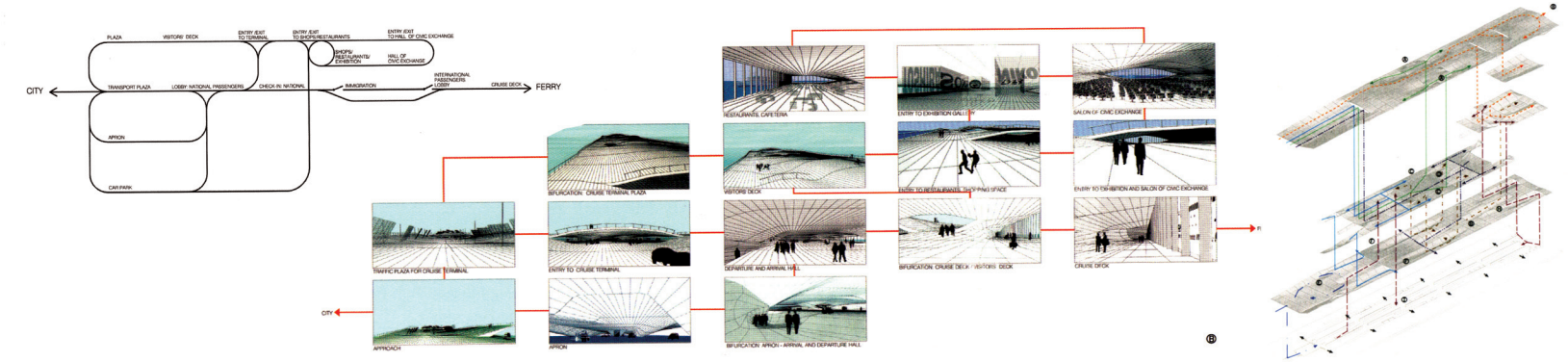
The shift from traditional means of industrial fabrication towards the trend of customization, in large part prompted Lynn's design of the Embryologic House: "mass-customizable individual house designs produced by differentiation achieved through parametric variation in non-linear dynamic processes." [2] Beginning with a primitive form, like an egg, rules were developed to break down the symmetry and set up links and connections between components. Thus, the interactions between all of the components are maintained while limitless variation can be achieved by mutating the form. The mutations are reactions to sunlight and environmental data, adjustments of posture to suit any surface, and opening or closing of architectonic apertures to suit programmatic requirements.[3]



iii. *Form as Information*

Dynaform - Bernhard Franken and ABB Architekten - 2001

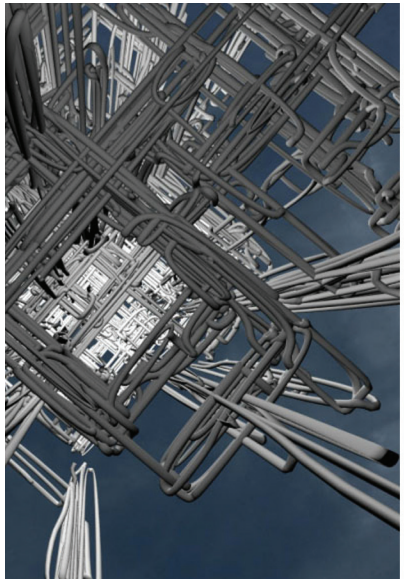
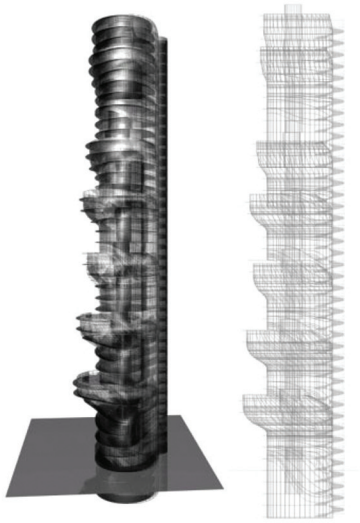
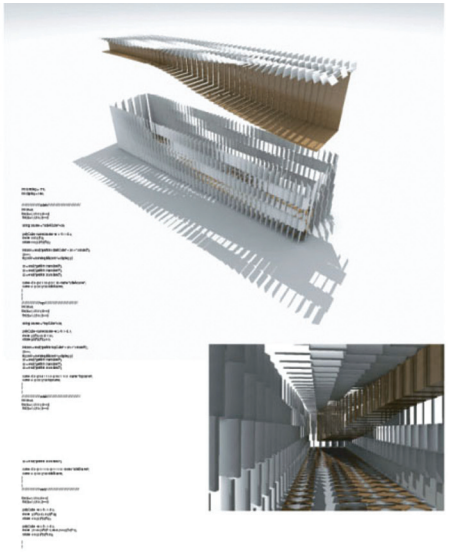
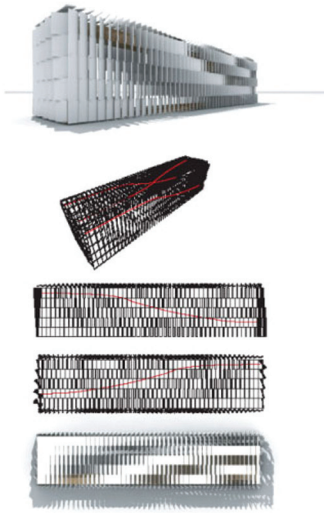
The master geometry for the Dynaform is the result of digital experiments in which laws governing the deformation of a basic structure are enacted based on contextual conditions, force functions simulated from physical site conditions, and extrapolated from the nature of sound travel and motion. Effects of those forces are perceived in the final form. "The information becomes form through a process of interaction between the designer and the computer. The force-field simulation is thus not only a method of generating the design, but is also used for its capacity to produce the spatial coding of information." [4] Once the final master geometry was established, "derivatives" from that model were taken to generate all forms of necessary information about the building including rendered images, section drawings, structural models, etc.




iv. *Form as Diagram*

Yokohama International Port Terminal - Foreign Office Architects - 2000

In the conceptual phases of the design of the Yokohama Terminal, when a decision was made that the building would be a warped surface, such a surface was associated with every segment of the flow diagram for the building's program (known as the "no-return diagram"), and a surface bifurcation to every bifurcation in the flow diagram. So a system of relating the form and diagram was established and provided some basic rules for seeking a preliminary version of the final form of the building. From there, the next question was about structure. The obvious solution was to support the surface with columns but, that would resort to old ideas about architecture being deployed on the diagram after the fact. Rather a system of structural system of folding planes was sought out which supported the idea embedded in the original diagram. As described later, the way in which the idea about the building's form meets the diagram is a pivotal moment in the design of the pool facility and is the focus for much of the effort in the project. There are also parallels in the order of operations from diagram to form to structural system.[5]

v. *Repetition and Growth*

Hybrid Tower and other projects - Kostas Terzidis

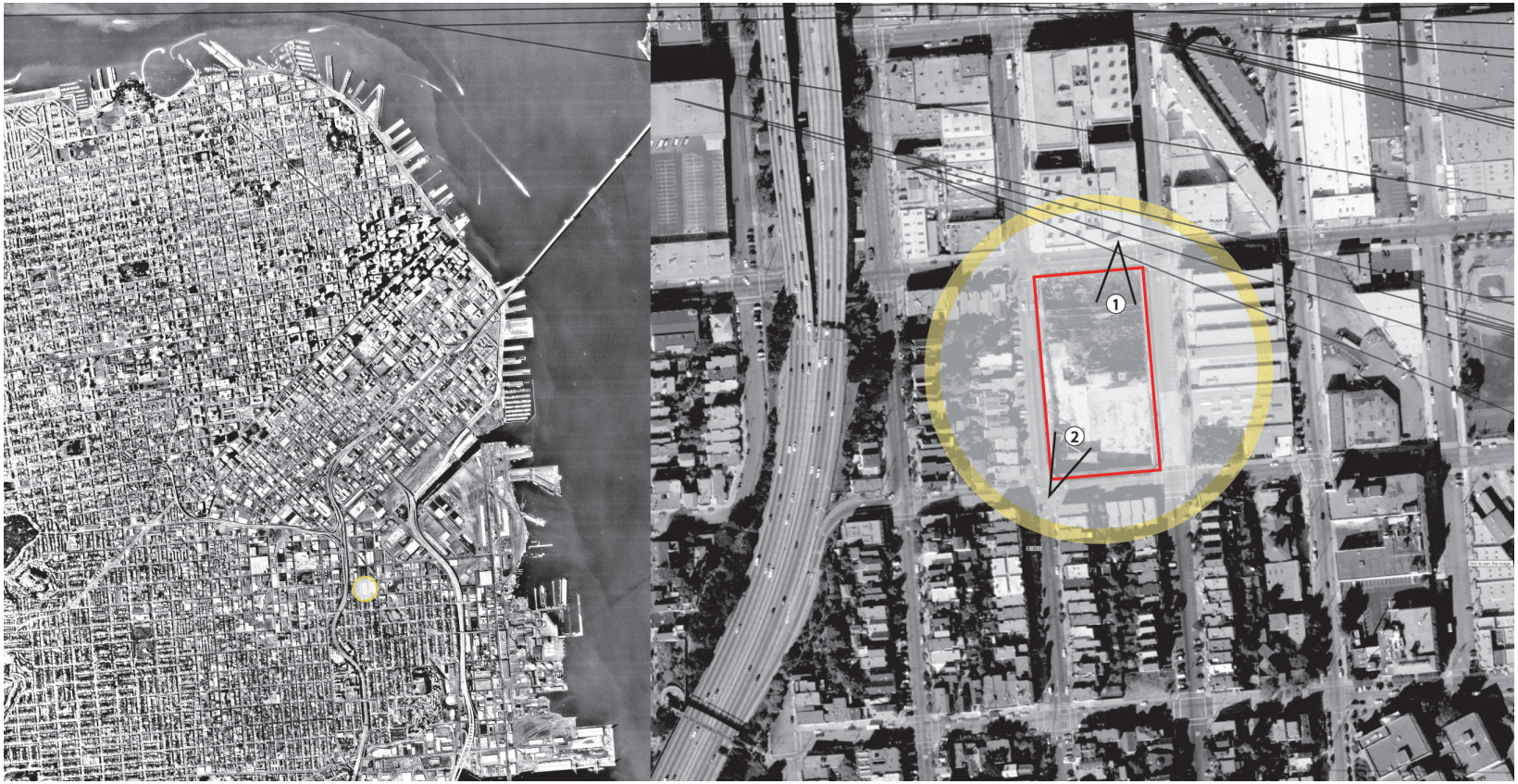
The Hybrid Tower uses an algorithm to morph a cylinder with a deformed NURBS surface in six steps. The resulting hybrid components form the tower in nine-story sections. In this example an algorithm is employed as a form-finding function which automates the transition from one shape to another to become the building components. Of particular note is the use of a MEL script to encode this effect on a Maya model. The script is entitled "Hetero-morphic Algorithm". Terzidis is also involved in projects which take on recursive processes, boolean algebra, and iteration as not only generative vehicles of ideas but as core principles for the definition of architectural form. [6]

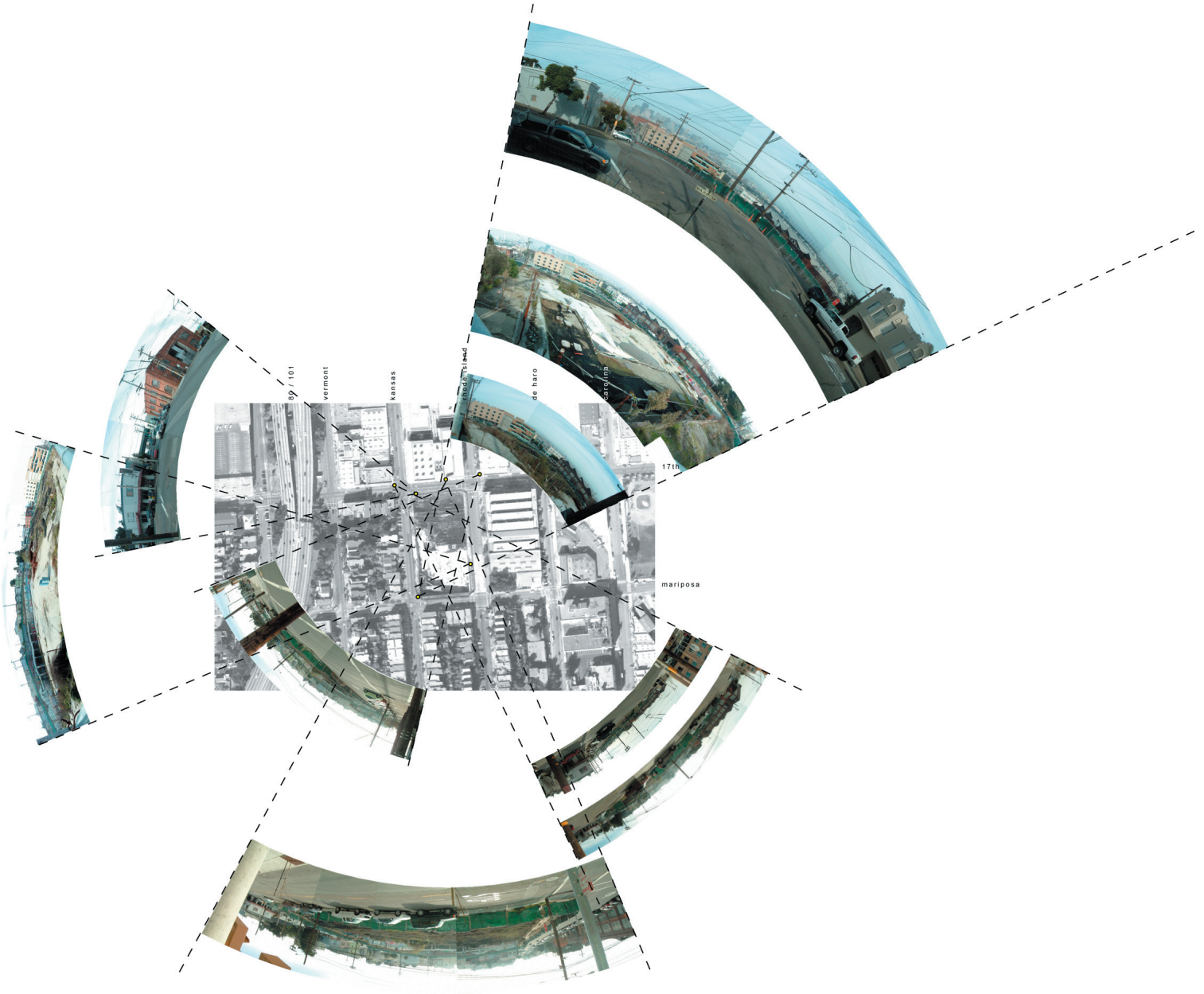
# Project

i. Site

The project is an aquatic recreation facility (swimming pool, gym, & amenities) in Potrero Hill, the form of which will be sought through a process of digital modeling and scripting. Thus, the project takes on two sites - one is the environment of Maya. The context for this site is Maya's Application Programming Interface (API)[1] which is controlled in part by the Maya Embedded Language (MEL)[2]. The result of the project as it relates to this environment is a set of scripts and plug-ins which are written as architectural-design-specific tools for Maya. Some of the scripts are more general and have the potential for re-application to different design processes, others are much more project-specific and are written for a particular task. In all, these scripts are as much the intended product of the project as the building itself.

The second (physical) site is a vacant lot in Potrero Hill between 17th & Mariposa St. and Kansas & Rhode Island St. The lot lies 3 blocks east of Highway 101 and 2 blocks west of Jackson Playground. It is bordered to


1
2


17th
mariposa

the south and west by 2 and 3 family residential housing and to the north and east by heavy industrial buildings, some of which have been recently appropriated by small businesses. The lot itself is about 300' x 200'. There is still the remains of the foundation of the recycling center which previously occupied the lot. Topographically the site is about 30' feet higher on the south west corner than on the north east. The higher corner affords an unblocked view of downtown San Francisco.

The physical site, and program (a swimming pool) serve as a test bed for the exploration of programming and scripting as a viable medium for the design of architectural form. The site and program are chosen for familiarity. The greater concentration of this project is on the process, the software and the coding involved. A swimming pool in Potrero Hill is simply a test bed for these ideas about coding and process.

ii. Scripts

Thirty-two individual MEL scripts were written as extensions of Maya's toolset. These scripts extend Maya's capabilities towards architectural goals, automate the construction of relationships between parts, and describe behavioral characteristics of the model which govern the formation of the proposed building. They are not only intrinsic to the project's formation but are tied also to it's representation in the way that they modify and extract information from the model in the form of a renderable scene, section cut, or a 3D-print-ready STL file. The scripts are divided into four types: General tools, Automation, Relationships, and Behavioral. The four categories are arranged in order of complexity of the script and, incidentally, in order of specificity to this particular project. Each category

is described below, followed by an outline of the scripts' inputs, and effects. Appendix A contains the complete code.

References to the scripts in the project description that follows refers to this categorization using the notation G|01 for General 01 which refers to the `annotateWithoutLeader()` script. Or R|03 for the `driveCVs()` script in the Relationships category, and so on.

1. General tools

One of the primary methods of customizing Maya's interface is the expansion of the basic tool set. Although this is the most basic form of script writing, these are the foundations for building a modeling environment which is more capable of supporting architectural intentions and sets up some basic functions which get used extensively in the progression of the project.

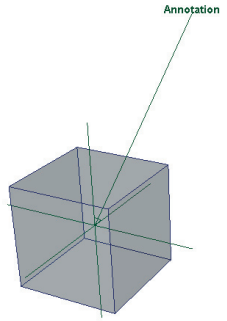
G|01. `annotateWithoutLeader()`

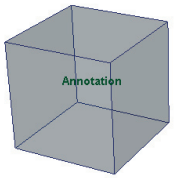
Input:

- Selected Maya object (transform node)
- Text field describing the annotation

Effect:

Attaches an annotation to the selected object. The existing Annotation tool attaches a Locator and Leader pointing from the annotation to the object. For neatness, this modification eliminates both Locator and Leader and attaches the annotation at the object's pivot.

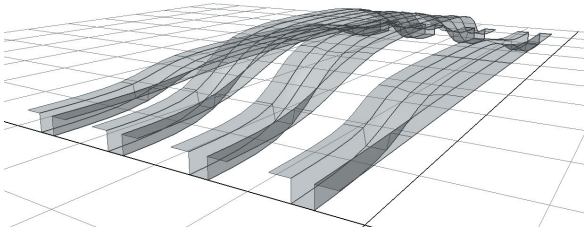


G|02. `buildTwistedSurface()`

Input:

- Number of ribbons
- Twisting data or twisting logic. This entails, for each ribbon, the degree of the twist, and a translation or an algorithm which describes such twisting and translation.



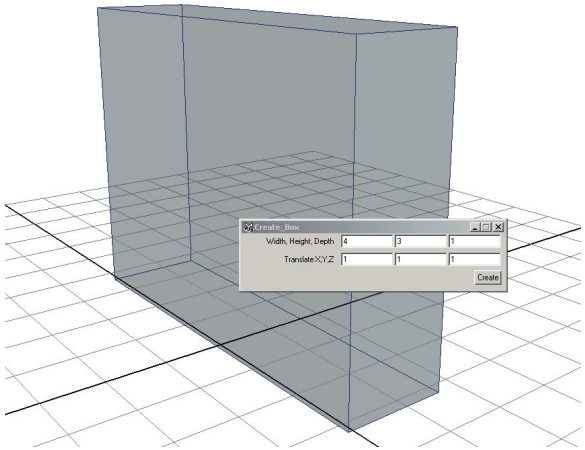
Effect:

Generates a NURBS surface divided into ribbons, the ends of which are transformed according to the inputs. This surface serves as starting point for the building's form and is later used with no initial inputs and is transformed by the data model.

G|03. `createBox()`

Input:

- Length, width, and height
- Translation, X, Y, Z

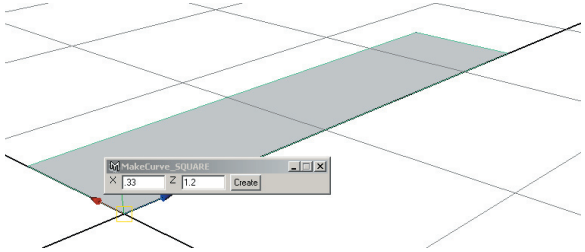



Effect:

Creates a polygon cube (Maya primitive) with given length, width, and height, and given translation from origin. Normal creation of a polygon cube in Maya is limited to width, height, and depth but does not include

translation.

G|04. `createCleanSquare()`



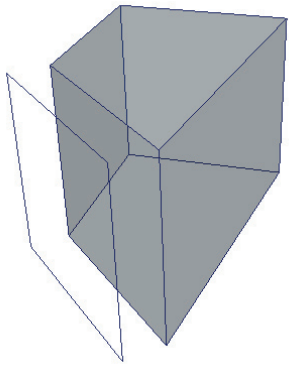

Input:

- Length, width

Effect:

Built in creation of a NURBS square creates a group of four individual, disconnected, linear segments. This modification of the Create NURBS Square command creates a single, continuous, linear NURBS curve. This functions more cleanly as a profile for extrusion along a path.

G|05. `createCurveFromPolygonVerts()`



Input:

- Selected polygon vertices

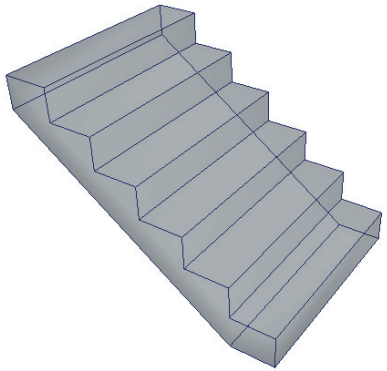
Effect:

Creates a closed, linear NURBS curve using the selected polygon vertices as control vertices for the curve.

G|06. `createStairs()`

Input:

- Rise, run
- Stair width
- Number of steps



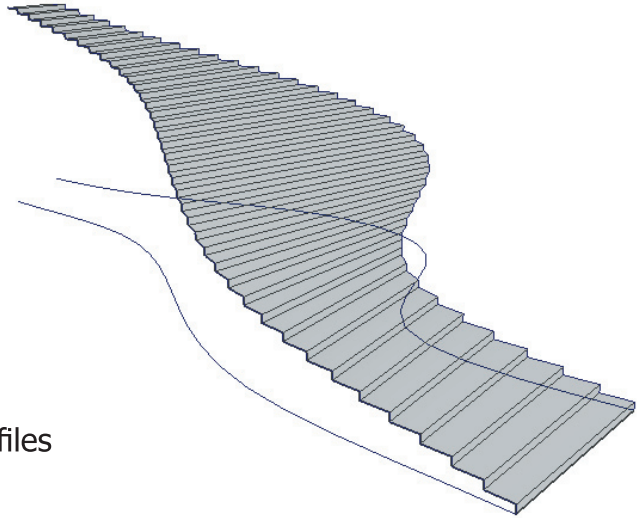
Effect:

Creates a staircase by generating a closed curve using the given rise and run and number of steps, creating a plane from the curve, and extruding the plane the given width.

G|07. `createStairsFromCurves()`

Input:

- Height
- Number of steps
- Selected two NURBS curves



Effect:

Rise and run are calculated according to the desired height and step count. Creates a staircase by lofting a surface between two stair profiles which are generated along the selected input curves.

G|08. `extractIsoparms()`

Input:

- Selected NURBS surfaces

Effect:

Rebuilds the selected surfaces and duplicates isoparms at U values of 0.0, 0.25, 0.5, 0.75, and 1.0. The duplicated surface curves are grouped in order with the prefix "isoparmGroup".

G|09. `extrudePolyline()`

Input:

- Height
- Selected NURBS curves

Effect:

Planarizes the selected curves and extrudes the resultant surfaces by the given height. This could be thought of as the converse of the createCurve-FromPolygonVerts script.

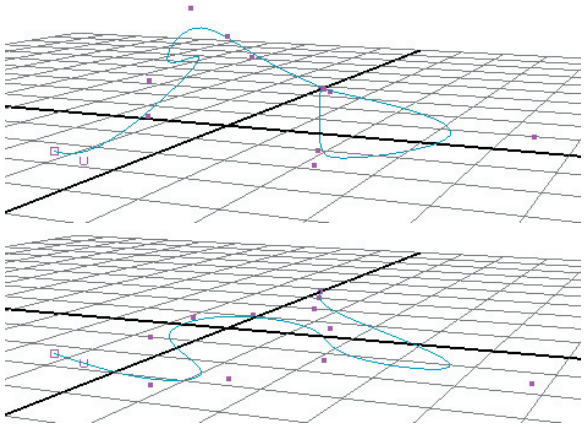
G|10. `flattenCVs()`

Input:

- Axis
- Selected CVs of a NURBS curve

Effect:

Flattens the selected curve control vertices along the given axis. The CVs are effectively planarized at their geometrical center.



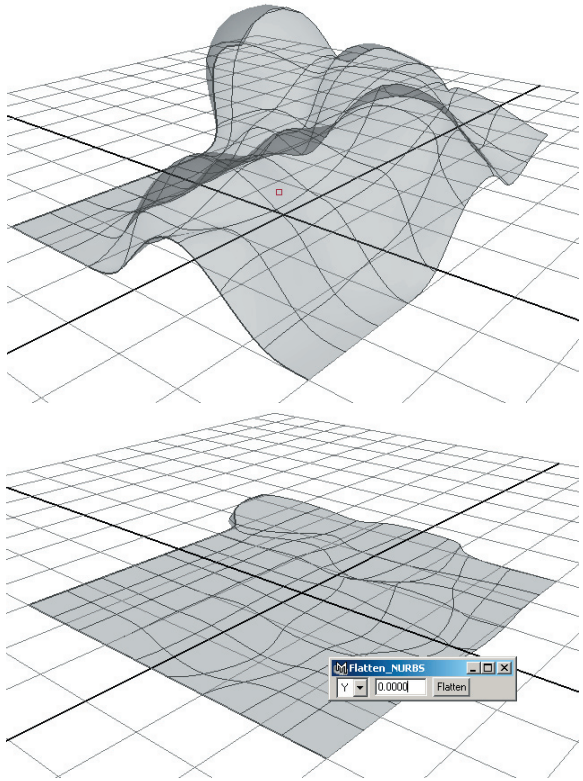
G|11. `flattenNURBS()`

Input:

- Axis
- Selected NURBS surface

Effect:

Flattens the selected surface along the given axis.



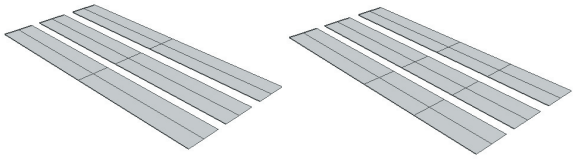

G|12. `insertIsoparmAt()`

Input:

- Location in U or V along surface
- Selected NURBS surfaces

Effect:

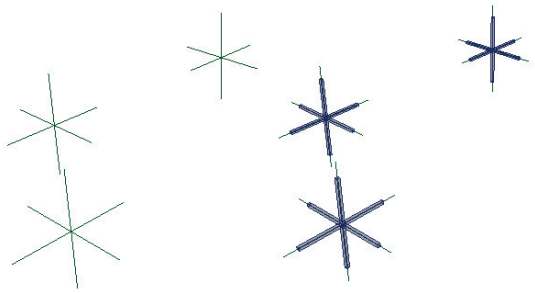
Inserts an isoparm on the selected surfaces at the given location in the U or V axis.



G|13. `makeLocatorsRenderable()`

Input:

- Size, thickness
- Selected locators for rendering



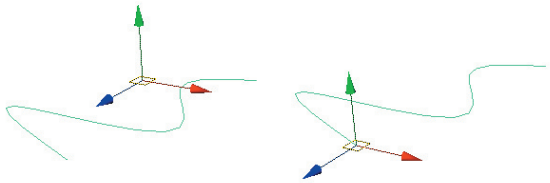
Effect:

In Maya, locators are one-dimensional point objects which don't appear when rendering the scene. makeLocatorsRenderable creates a solid geometry which resembles the screen appearance of a locator and attaches their positions. The new geometry is visible when the scene is rendered.

G|14. `moveCurvePivot()`

Input:

- Selected CV of a NURBS curve

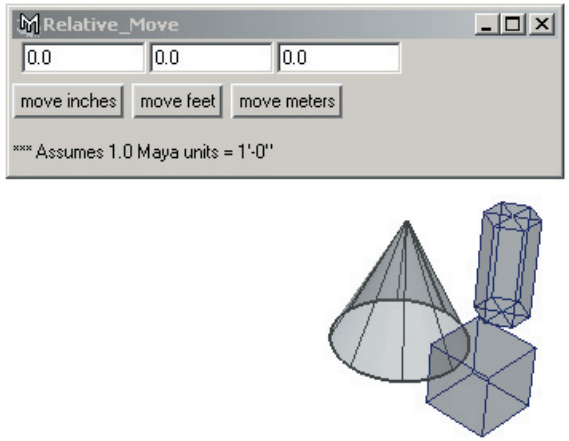


Effect:

It is sometimes necessary to snap a curve's endpoint to the endpoint of another curve, the vertex of a polygon, etc. This often entails moving the

curve's pivot to a CV at the endpoint. However, it is impossible to view a curve's CVs and move the pivot simultaneously. moveCurvePivot resolves this by allowing the user to simply specify the desired CV for the new pivot position.

G|15. `relativeMove()`




Input:

- Translate X, Y, Z
- Units (inches, feet, meters)
- Selected object (Maya transform node)

Effect:

Translates the selected objects the given number of units. Optional movement in inches, feet, or meters can be specified assuming the model is set up where one unit = one foot. This is useful when modeling or moving something that has information in meters when the rest of the model is in English units (for example, a 50 meter swimming pool in a 200 foot wide site).

G|16. `showHiddenChildren()`

Input:

- Selected object (Maya transform node)

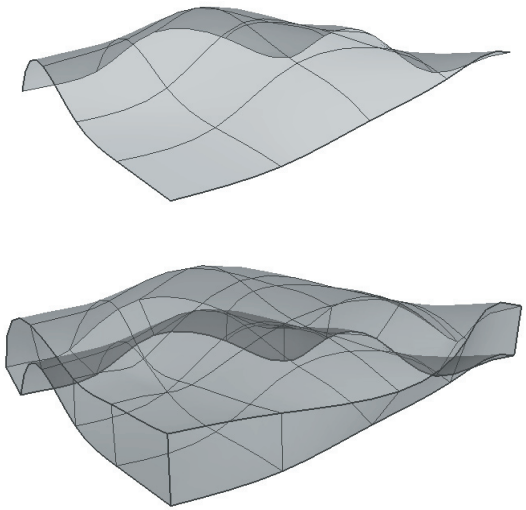
Effect:

If hidden, unhides the object and all of it's children. The given unhide command in Maya will unhide the selected object but, if they are hidden, it's children remain hidden.

G|17. `thickenSurfaces()`

Input:

- Thickness
- Selected NURBS surface



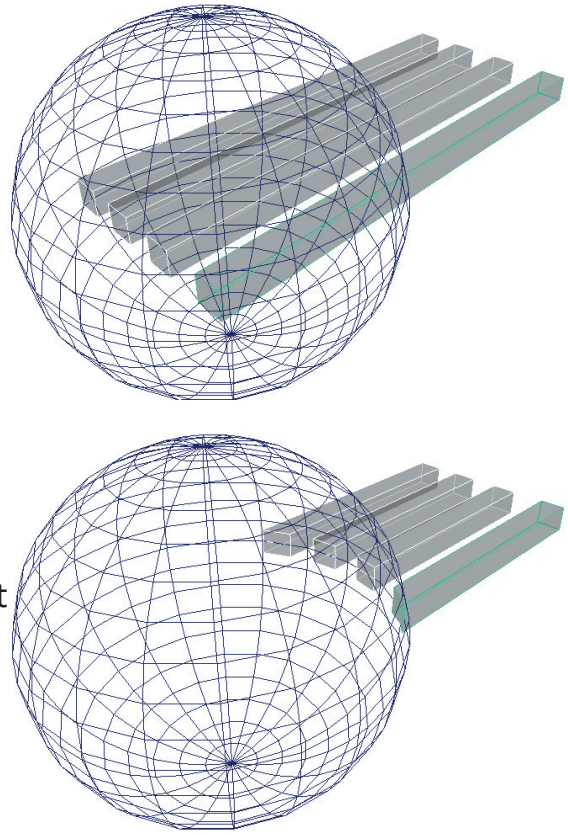
Effect:

Offsets the selected NURBS surface by the given thickness, and lofts surfaces around all four edges of the two. This effectively thickens the surface, often in preparation for some form of fabrication (rapid prototyping). An extension of this script connects all of the surfaces and converts it to a solid polygon which is typically more useful for fabrication techniques.

2. Automation

In some ways this type of script is resorting to the type of operation originally intended for loom: the identically repeated execution of a task. It is

arguable that all of the scripts would fall under the category of Automation since they are all, in some way, automating a set of operations. These however, are written specifically for the repetition of an often simple operation which would otherwise be a tedious task. The more powerful scripts in this category actually transform the operation according to an algorithm, variable, geometry, or specific user input.



A|01. `booleanMulti()`

Input:

- Selection of polygon objects
- Selected polygon knife

Effect:

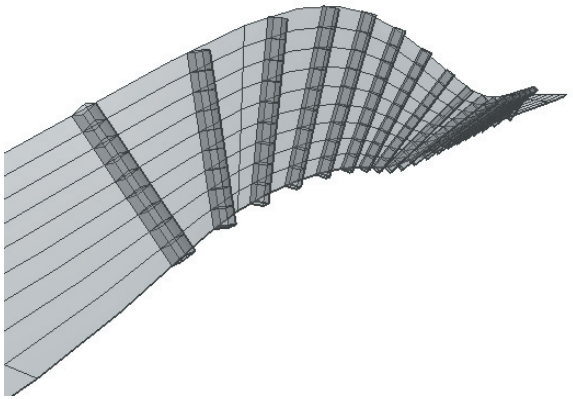
Boolean operations in Maya must be done one pair of objects at a time. booleanMulti (actually a misnomer: should read booleanSubtractMulti) takes the selected objects and repeatedly subtracts the last selected object (the knife) from each of them.

A|02. `createMullions()`

Input:

- Mullion count

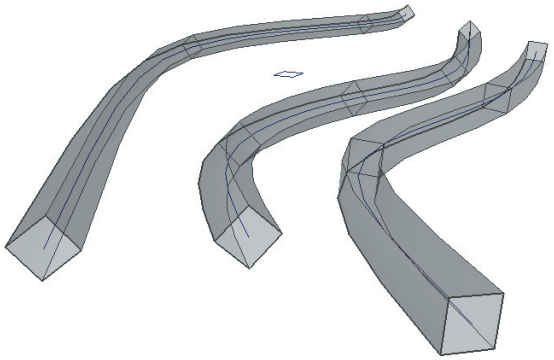
- Start at (greater than zero), end at (less than count)
- Profile
- Selected NURBS surface



Effect:

Subdivides the selected surface into the number of specified mullion segments and lofts the specified profile along the isoparms. In effect, a NURBS surface is created as the fenestration and its structural members are generated by the script.

A|03. `extrudeProfileOnSelectedCurves()`



Input:

- Profile NURBS curve
- Selected NURBS curves

Effect:

Extrudes the specified profile along each of the selected NURBS curves.

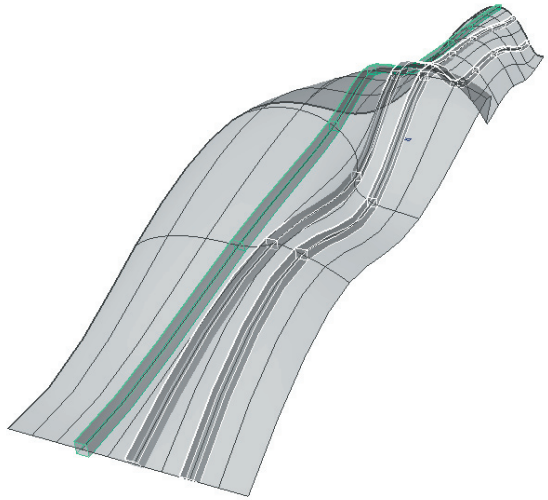
A|04. `extrudeProfileOnSelectedIsoparms()`

Input:

- Profile NURBS curve
- Selected isoparms of NURBS surfaces

Effect:

Similar to extrudeProfileOnSelectedCurves(), but operates on NURBS isoparms instead.



A|05. `fixIsoparms()`

Input:

- selected NURBS surfaces

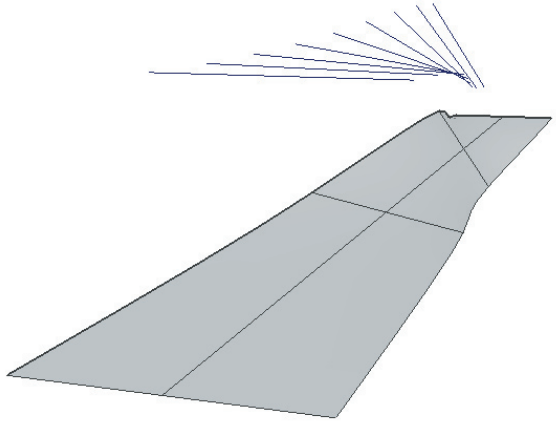
Effect:

Inserts isoparms at a specific location on each selected surface. (Note: this script was highly task specific and was written as a one-off use correction of a geometrical error.)

A|06. `generateLinesFromStrips()`

Input:

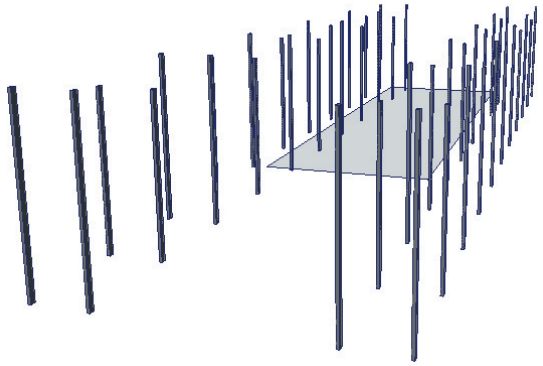
- Starting point
- Increment and count
- Selected NURBS surfaces



Effect:

Duplicates a sequence of isoparms specified by starting point, increment, and count along the selected NURBS surfaces.

A|07. `makeColumns()`



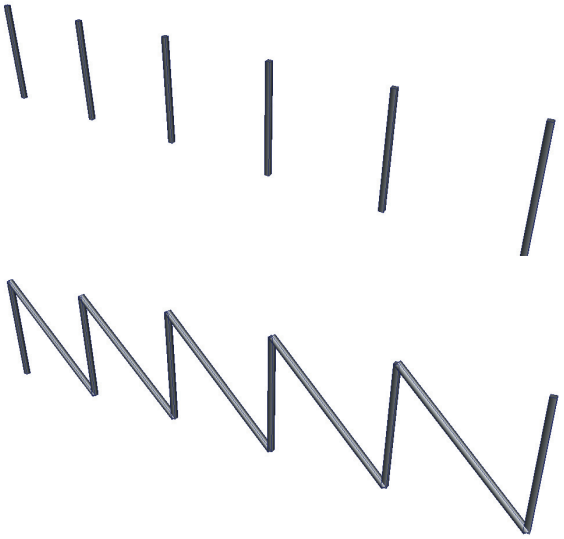
Input:

- Profile NURBS curve
- Selected NURBS surface

Effect:

Creates a matrix of columns underneath the selected surface using the specified curve as the column profile.

A|08. `sawToothLines()`



Input:

- Selected sequence of (approximately) vertical (Y-axis) NURBS curves

Effect:

Iterates through the selected vertical lines and joins them to form a saw-tooth profile intended to act as bracing for a truss.

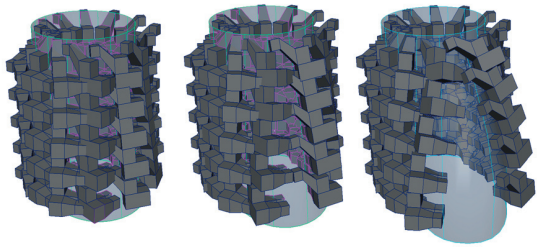
3. Relationships

One of the most powerful aspects of Maya is the ability to set up relation-

ships between objects' attributes. There is a fairly robust core set of relationships that are built into the software and can be applied easily to any transform node in Maya. Some of the basic relationships are called constraints which, for example, cause one object to maintain the same orientation as another (like the front wheels of a car), or cause one object to constantly remain oriented towards another (like a compass needle pointing north). The connection editor allows even more complex relationships to be established. Using this dialogue, relationships between any attribute of an object and any attribute of another object can be established. For example the rotation of an object can be wired to the scale of another; or the color attributes of a material can be wired to the location attributes of an object. Even more complex relationships can be established using the Expression Editor. Similar to the connection editor, relationships between any two attributes of any two objects can be established, but using expressions, functions and algorithms can describe these connections in much more complex ways.

The following scripts employ one, or a combination of these relationship-establishing techniques to "wire" the attributes of objects in the scene and help build a much more complex 3D model.

R|01. `attachToSurfaceWithGeoConstraint()`

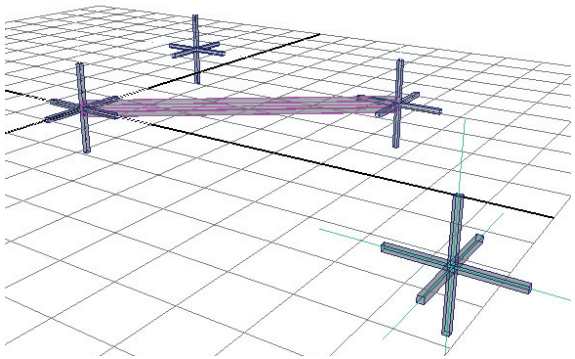


Input:

- NURBS surface base shape to receive units
- Unit geometry for duplication and application to surface

Effect:

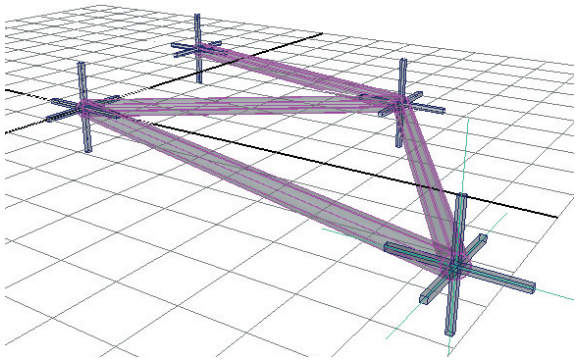
The unit geometry is point-constrained to the CVs of the specified surface. In order to maintain orientation, a geometry constraint is also applied between each unit and the surface. An extension of this script wires the locations of the units to the CVs and when the base surface is deformed, the units maintain their surface relationship with it.



R|02. `createLinearNodeConnection()`

Input:

- Profile NURBS curve
- Existing curve group and existing surface group
- Two selected transform nodes (ideally locators)



Effect:

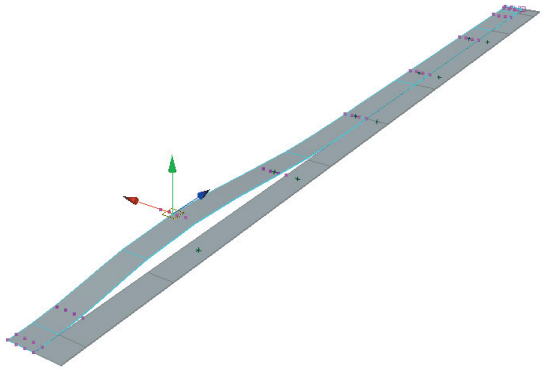
Creates a linear curve spanning between the selected objects and extrudes the profile curve along it. The position of the ends of the curve are wired

to the position of the locators the connection remains when the locators are moved. This script can be used repeatedly to establish a network of connections between a field of nodes. The spatial configuration of the network can be modified but will maintain the same topology of connections.

R|03. `driveCVs()`

Input:

- Selected transform node (ideally a locator)
- Four selected CVs of a NURBS surface



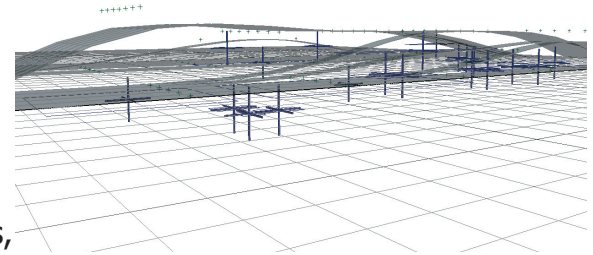
Effect:

Wires the Y value of the selected object to the Y values of the four selected CVs of the surface. Thus, when the locator is moved in the Y direction, the specified CVs of the surface follow accordingly. ** This script is used in conjunction with the following expression.

Expression: `nodeHeightByProximity()`

Input:

- N/A



Effect:

When used in tandem with the driveCVs() script and a network of locators,

this expression detects the proximity of a locator to each of the driving locators in the driveCV system. If a network locator is near enough, the Y value of a driver locator is changed, thus, deforming a the NURBS surface to which that locator is connected.

4. Behavioral

The next level of scripting takes these tools to the next level where their purpose is to cause objects in the scene to be generated, modified, or connected based on desired behaviors described by an algorithm, environmental variables, material properties, a simulated force system, etc. These behavioral scripts often transcend the sum of a set of Maya functions because they begin to describe systems of information that relate to phenomena which can't easily be described only in terms of the scripting language.

B|01. `createPoolRoof()`

Input:

- Roof load in PSF
- Steel tensile stress in PSI

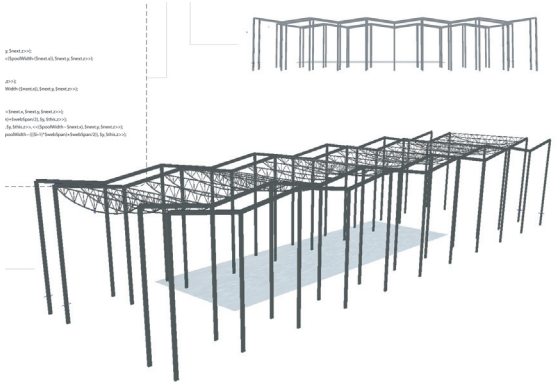
- Number of Spans

- Number of trusses per column

- Vertical fluctuation at each column set

- Truss webbing spans

- Spatial positioning of locators to indicate pool length, width, and depth, deck width on both sides, and truss depth.



Effect:

Used in conjunction with the expression updateTrussSystem() this script generates a dynamic structural system for a roof with trusses and steel members sized appropriately to minimize steel cost. The overall size of the structure is given by the position of the locators. Truss depth and steel member size are inversely proportional according to structural calculations. The truss depth can be adjusted and member sizing updated in real-time. At mid-semester this script was instrumental in developing the project further. Although the emphasis shifted away from manufacturing toward parametrics, the roof frame that this script generated was hybridized with two other schemes to form the final building structure.

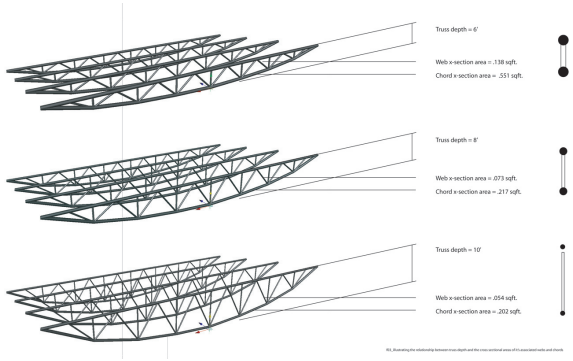
expression: `updateTrussSystem()`

Input:

- N/A

Effect:

Applied after executing the createPoolRoof() script, this expression updates truss parameters when the original locators are adjusted. In particular, if the locator describing the truss depth is moved, the trusses update their arcs, and the truss members update their cross-sectional area.




B|02. `panelizeSurface()`

Input:

- Booleans indicating whether or not to create beams, holes, or hydro elements in the slab

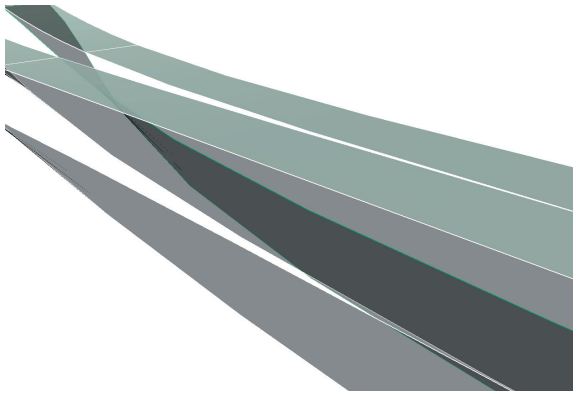
- Slab count, start at, and end at

- Slab sizing parameters: thickness, gab between slabs, web depth, web offset from edge

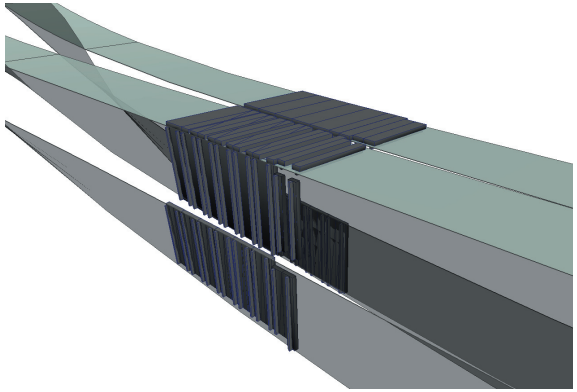
- Hole sizing parameters: offset from web, offset from beam, light node effect

- System of "light nodes" - locators with spheres associated with them to indicate porosity in the slabs

- Selected NURBS surface



Effect:



Using the selected surface as a guide, geometry representing double-T precast concrete slabs are generated in sequence along the surface. The slab geometry is generated entirely by the script, thus, everything about it's sizing is parameterized. The holes are calculated according to the positioning of light nodes. If a light node is in proximity to a slab element, the distance to the light node is calculated, and a hole opens in the slab, sized based on the distance. In effect, the holes in the slabs are a ghost of the locations of the locators and their associated spheres. Ideally, this script would be used in conjunction with an expression (not described in this paper) which caused the hole configuration in the slabs to update in real-time as the positioning of the light nodes is changed.

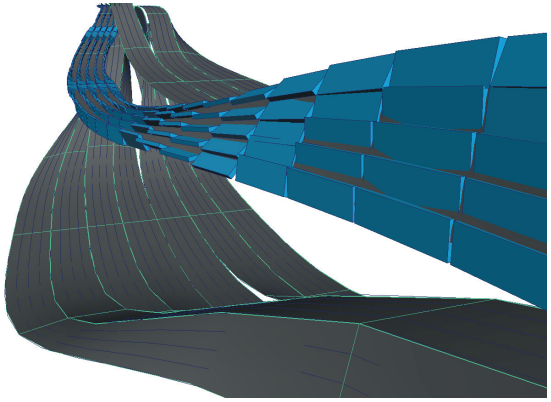
B|03. `snapshots()`

Input:

- Polygon geometry for "base unit"
- Start number and count
- Bank threshold
- Selected NURBS curves

Effect:

Animates the polygon geometry along the selected curves using the indicated bank threshold. When used carefully in combination with the curvature of a series of approximately parallel curves, the bank can cause controlled openings in the system of units. The relationship between the swarm of units and the curves is kept after the script is run so that modification of the base unit or the curves automatically updates the configuration of the system. This script, like `createPoolRoof()` was hybridized with two other ideas about the building structure to form the final structural scheme.



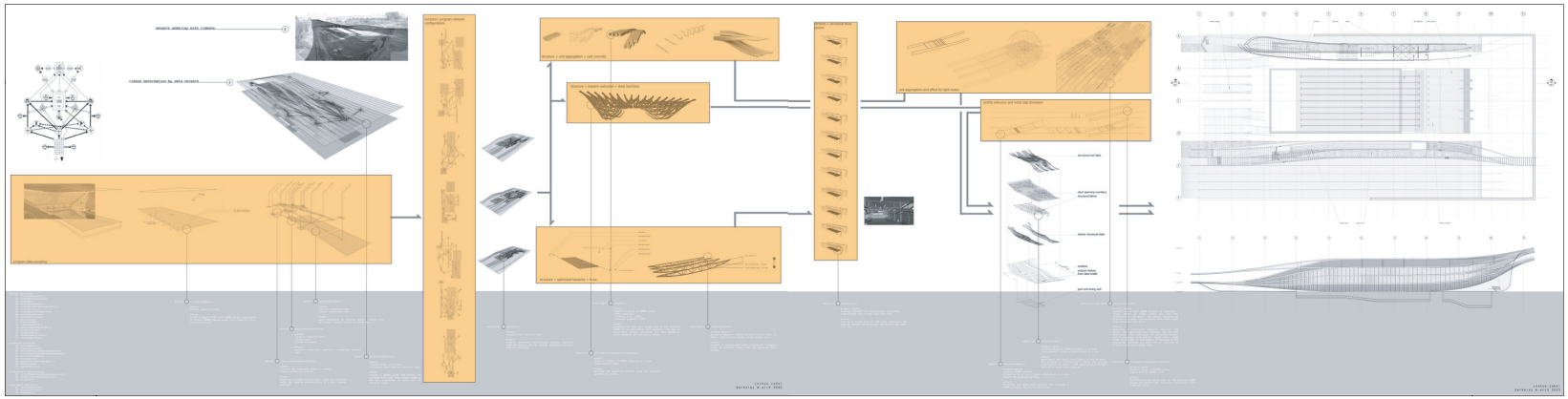
Process

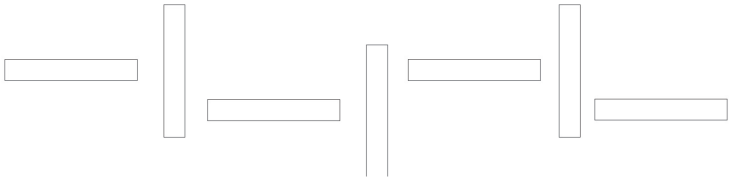
The scope and level of detail of the 3D model are an effect of the desired products the model is intended to produce. In this case, an architectural thesis project in an academic setting, where the output is digital renderings, diagrams, and 3D prototypes. Similarly, the scripts which are built to support this output are written with the intention of achieving the same goals - in fact, some of the scripts are written specifically for the translation of the model to the output. To refer again to some of the original aspirations of this project and the relationship of this process to manufacturing, I argue that the potential for the model and it's accompanying scripts can be extrapolated to perform a similar task if charged with the job of producing, say, construction documents, or digital fabrication information.

Programmatic information about the pool, as a type, is derived from precedents of swimming pool design and general architectural knowledge about the swimming pool type (Graphic Standards). That information is embedded into a spatial data network model which contains information about each program element of the building. The site is configured as a series of bands, or ribbons, continuous with the site topography which the model is connected to. An initial form is chosen from versions of this model generated by modifying the configuration of the network and its input data. Next, several structural ideas are explored, and hybridized to form a virandeel truss-like system which is a result of the cross-sectional twisting of the ribbons. Long structural steel spans are extruded along isoparms of the NURBS which define the ribbons. Several possibilities for the building envelope are also explored which become a pre-cast double-T concrete slab spanning between the long steel spans. Openings in the concrete units are defined by a light-node system and algorithm. The nodes are configured according to data from

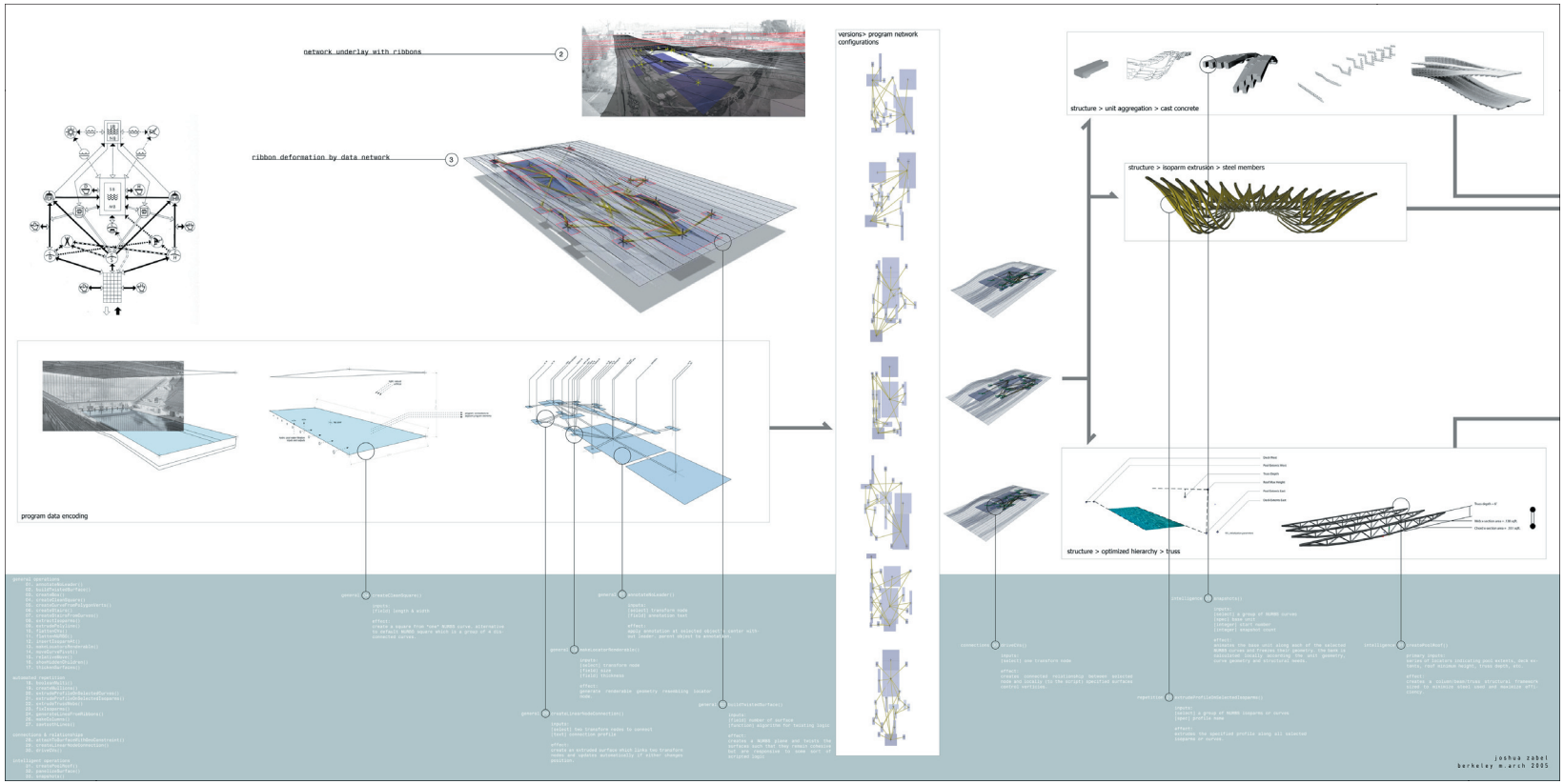
the original network model.

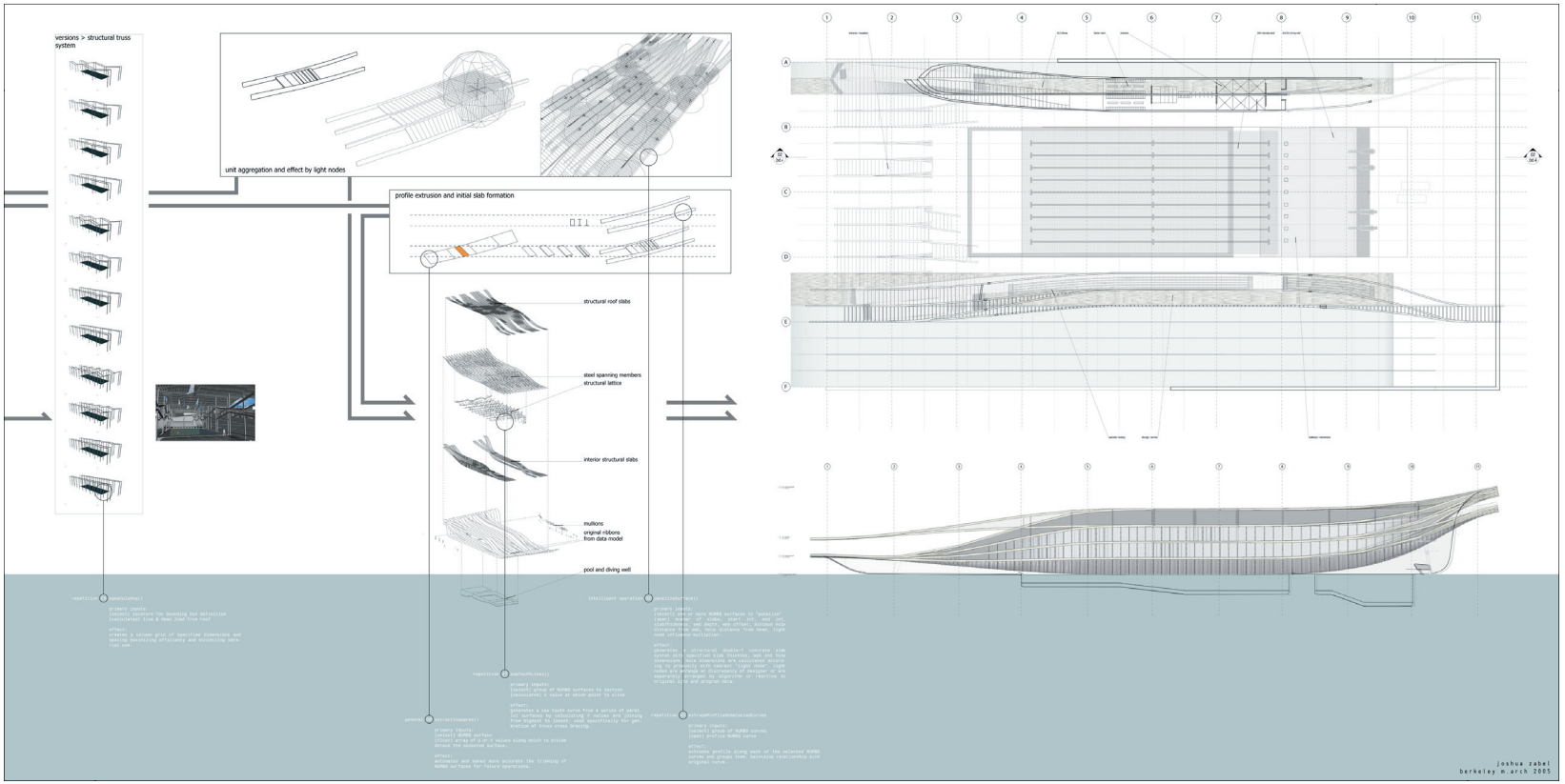
The following sections outline the design process and describe the ways in which the above scripts have been instantiated. Four stages are outlined starting with a data model, mapping form to the data, followed by structural and skin solutions.





The images preceeding, below, and on the following page graphically outline the process of coding and versioning. A horizontal box contains a sequence of operations which define a coded operation and a vertical box contains a selection of the versions that code produced.

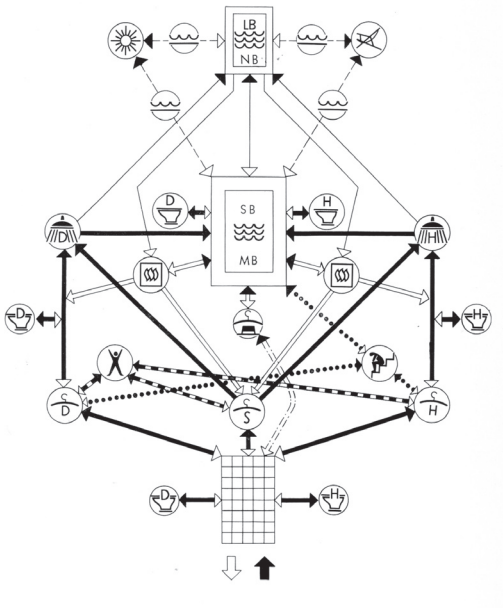


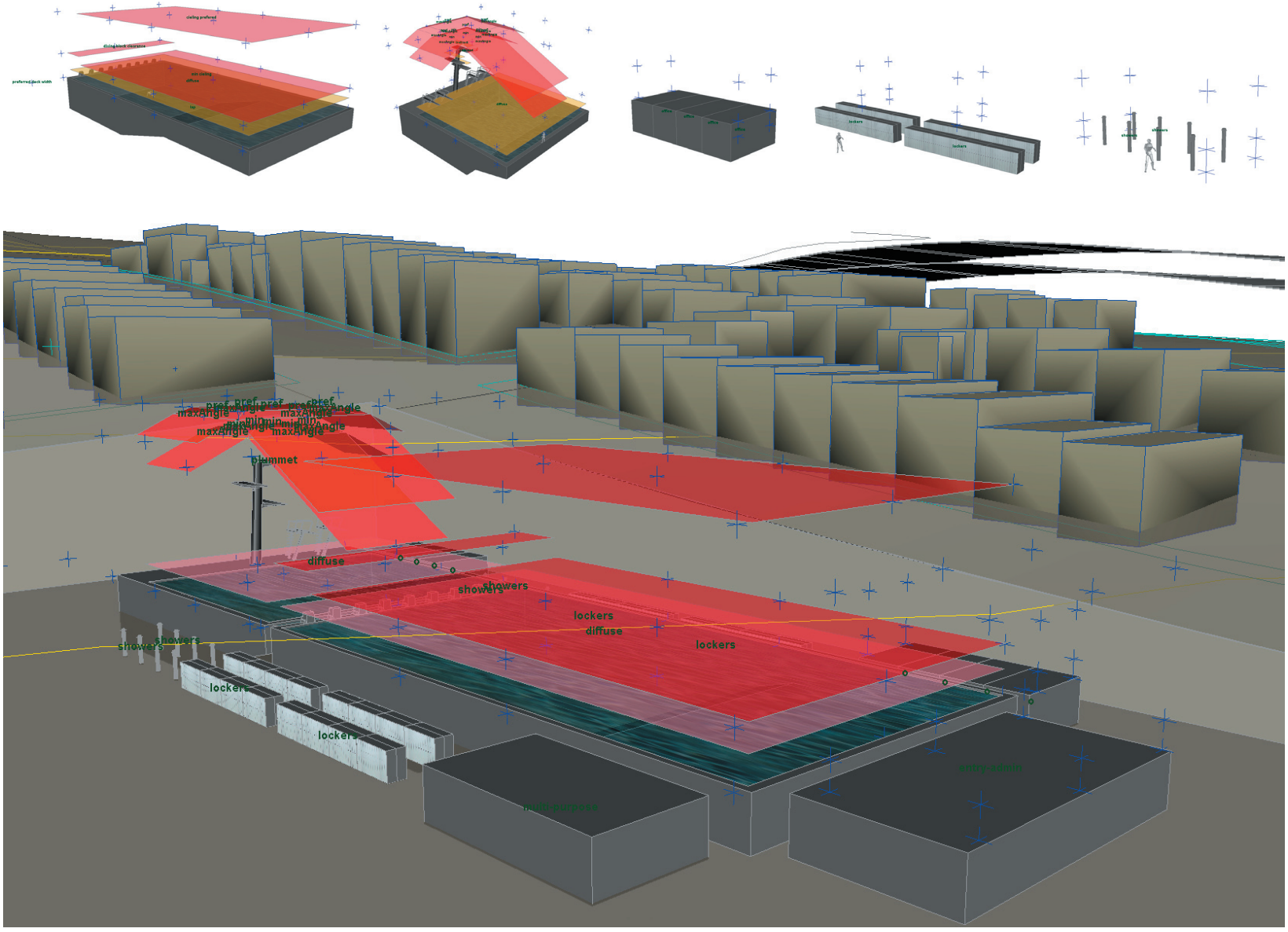

versions > structural truss system
unit aggregation and effect by light nodes
profile extrusion and initial slab formation
structural roof slabs
steel spanning members
structural lattice
interior structural slabs
mullions
original ribbons
from data model
pool and diving well
joshua zabel
berkeley m.arch 2005

iii. Data Model

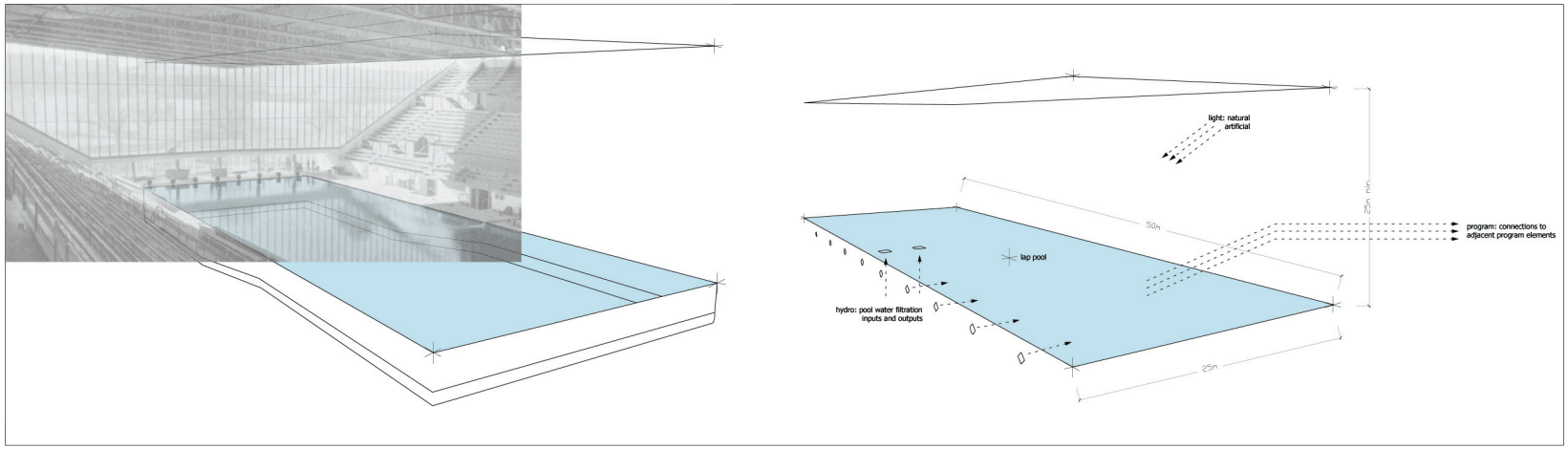
As a way to understand the program for the swimming pool type and the site a 3D networked data model was built. The nodes in the network correspond to program elements and have attributes associated with them to describe their spatial requirements (length, width, height), light needs (quantity, quality, natural, artificial, etc.), water requirements (potable, chlorinated, sewage, etc.), electrical, and so on.

The original topology of the network is derived from a program diagram used to layout the circulation of the Raumzuordnung und Wegefuhrung im Hallenbad pool [1]. The digital model for this project is constructed and is operated by scripts for establishing and keeping spatial relationships between its nodes.

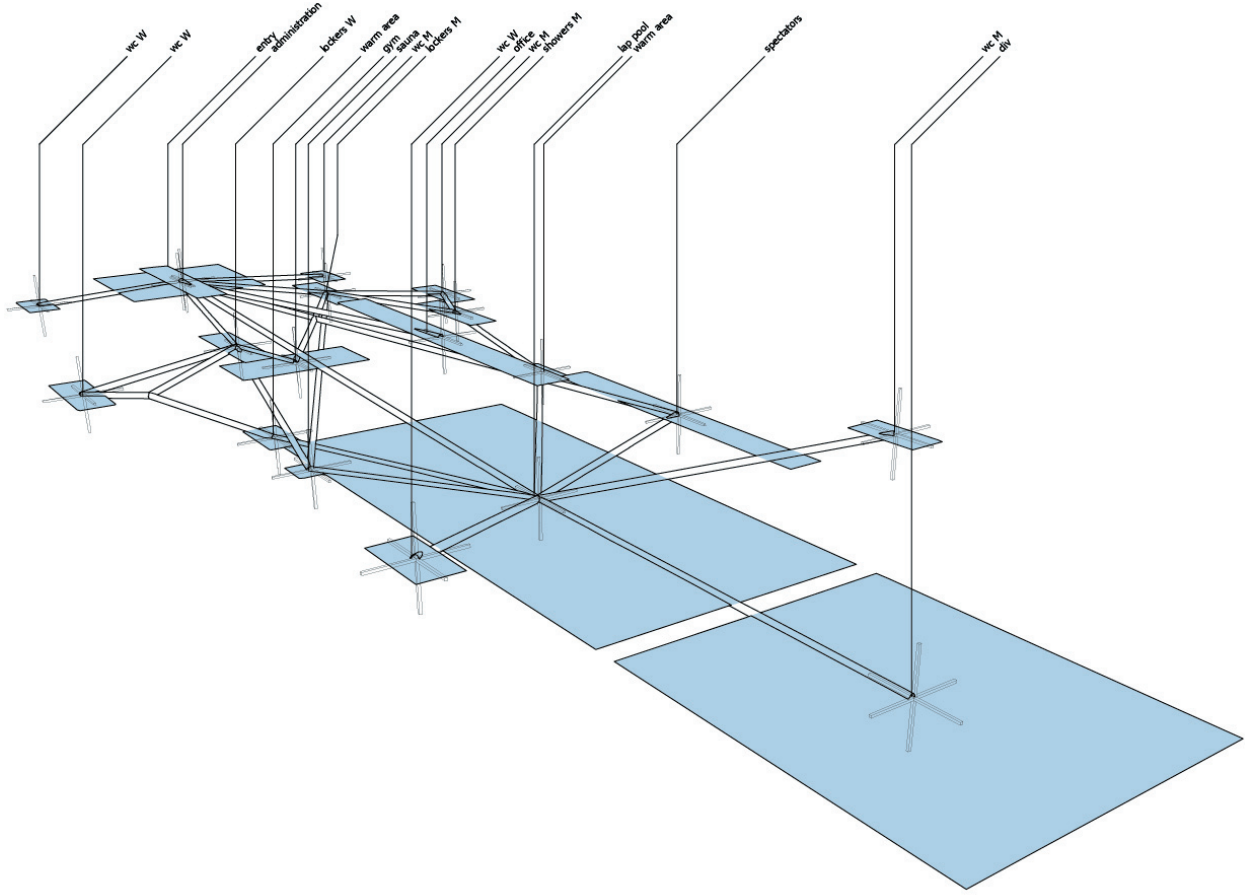




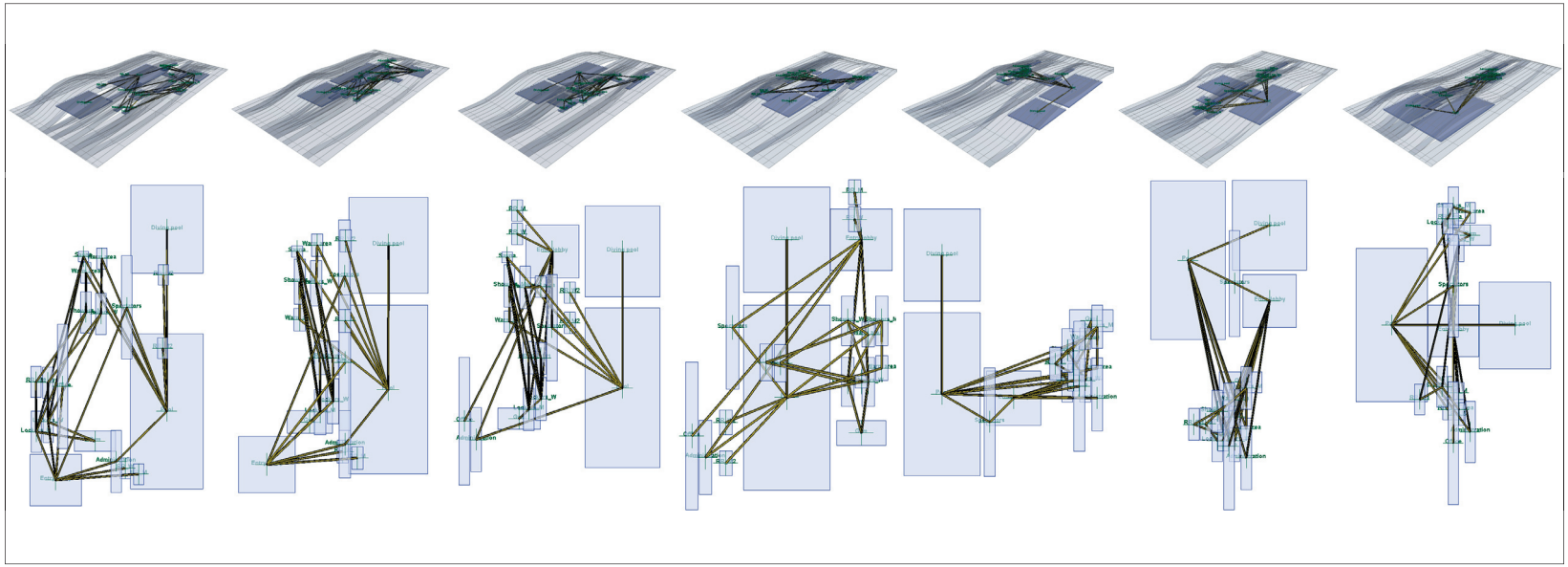

Original 3D data model




Data model node for the lap pool



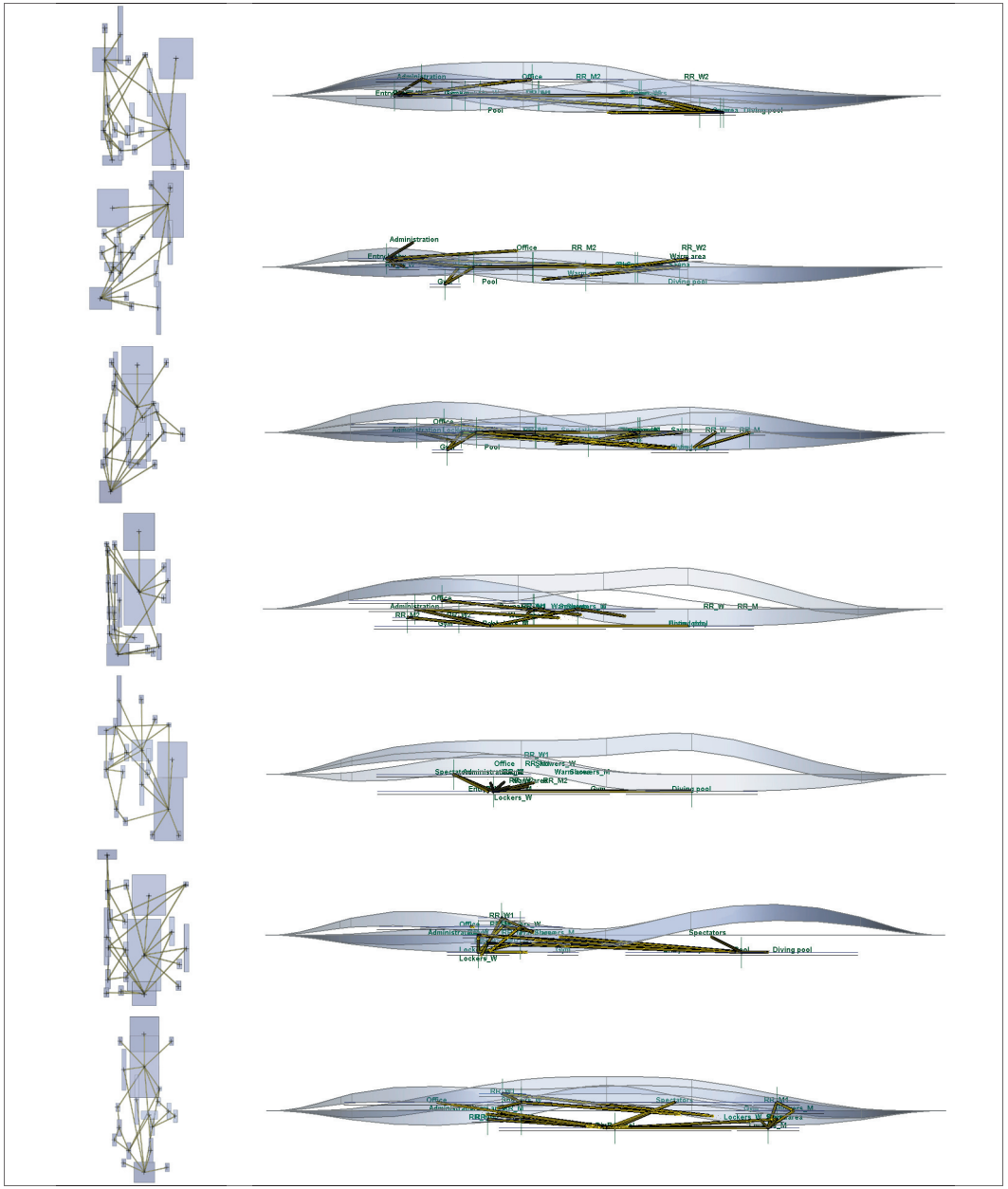

Complete data network



Network configurations

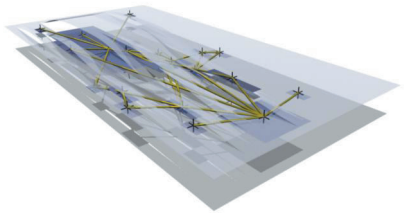
Network

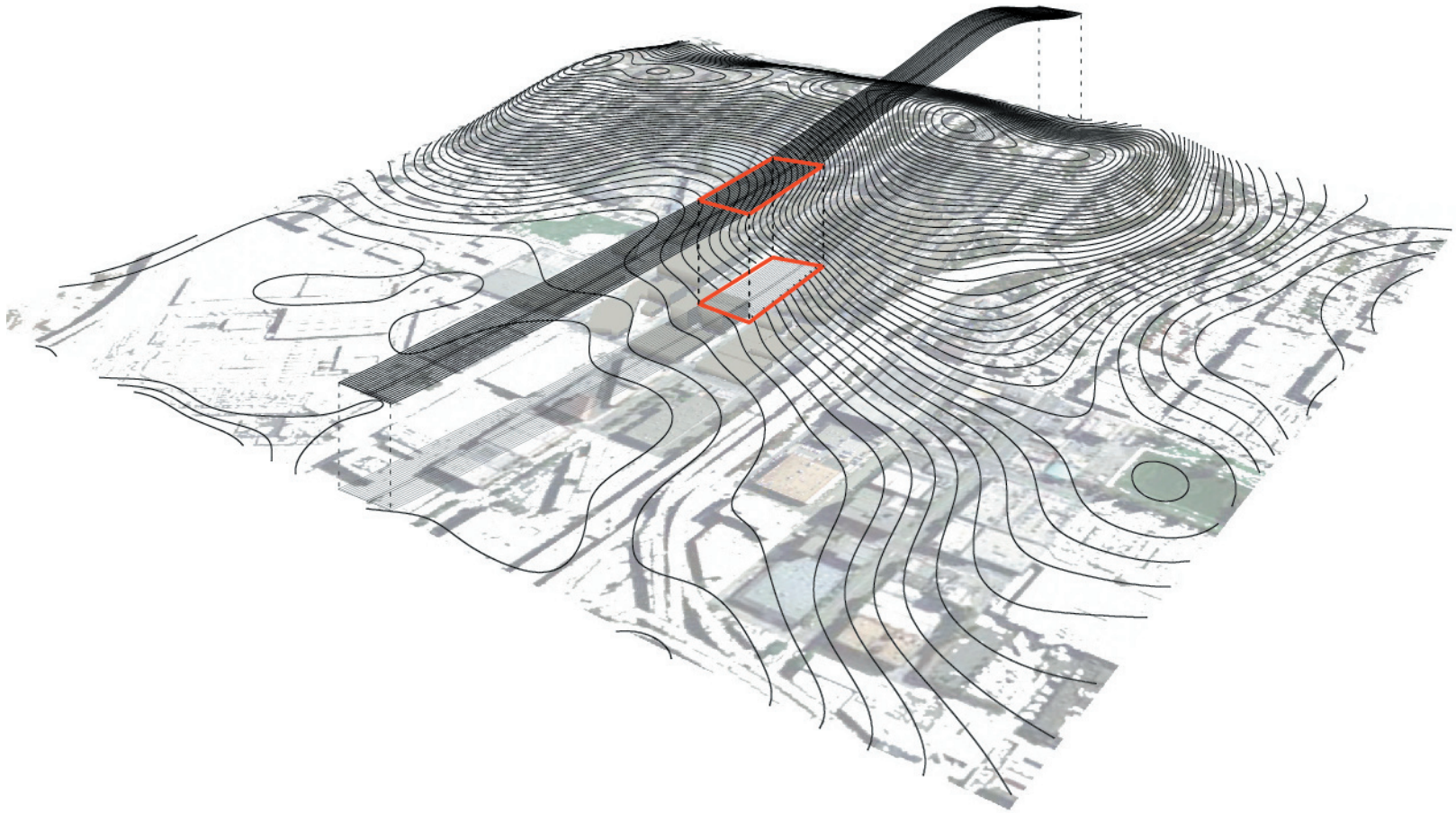
configurations



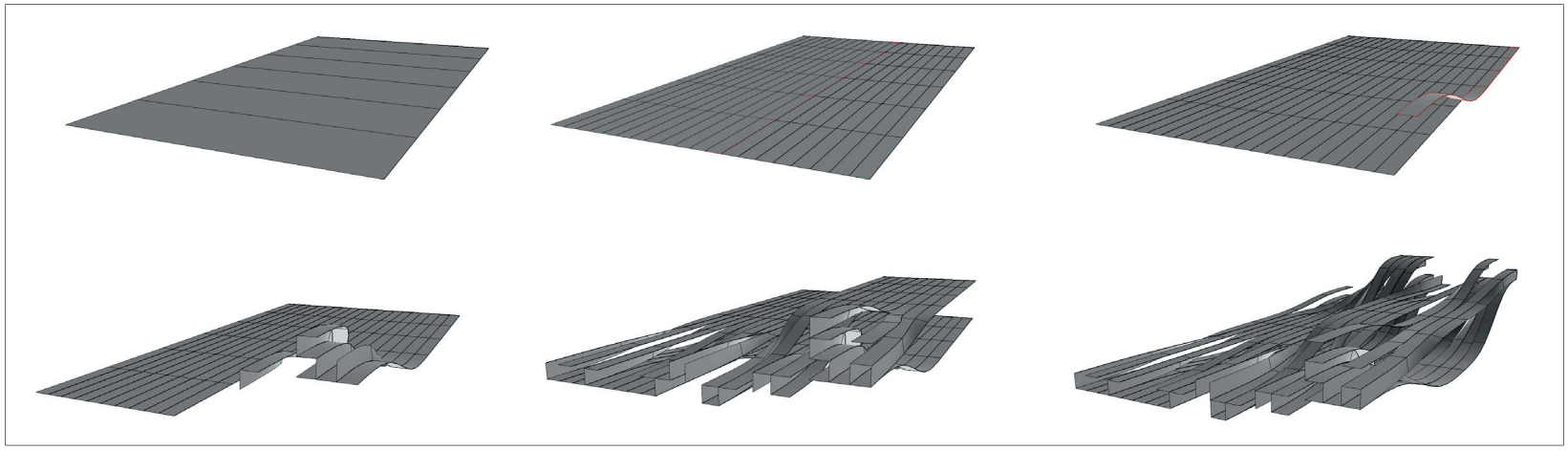
iv. Mapping Form to Data

The steep slope of the site is one of the primary considerations in how form gets mapped to the network diagram. The site is divided into a sequence of ribbons - long strips extending beyond the limits of the lot. The network model is overlaid on the site with the ribbons and their geometry is wired to the configuration of the network diagram. Information about spatial requirements of program elements embedded in the data model effect the heights of the ribbons to accommodate.

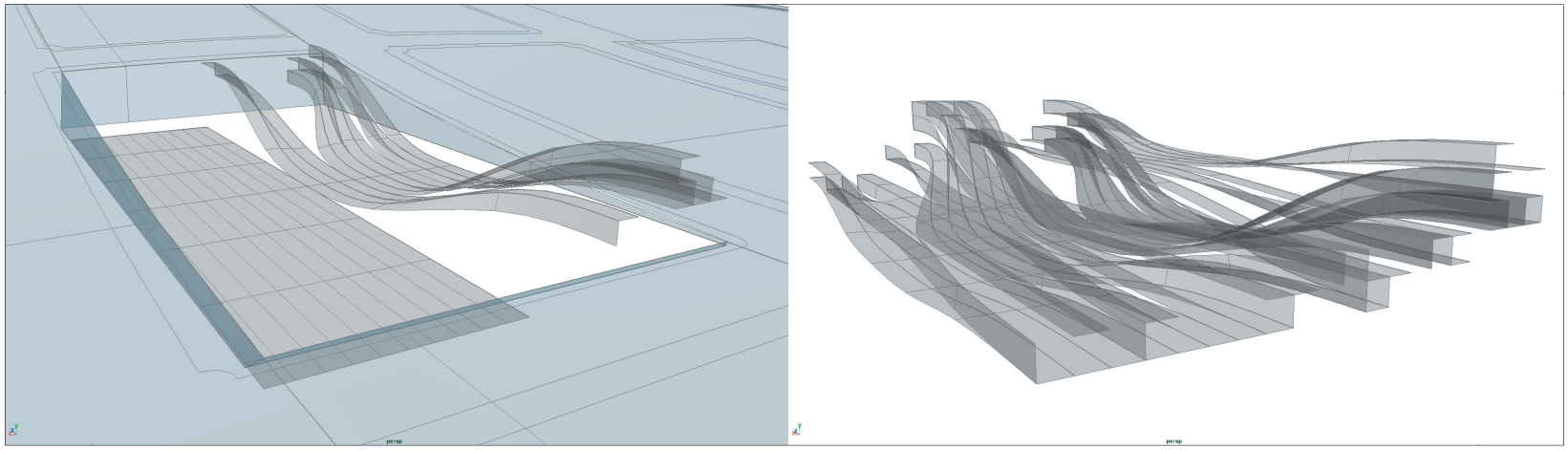




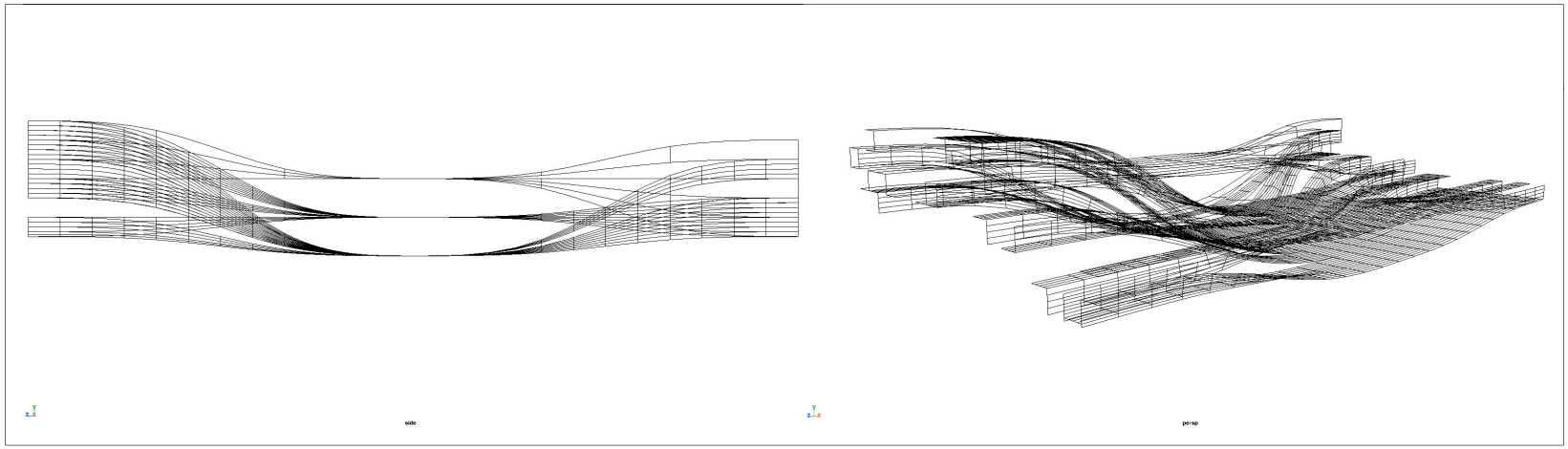
Above: the site configured as a series of north-south oriented ribbons stretching into the fabric of the city. The site block is in red.



Reconfiguration of the ribbons

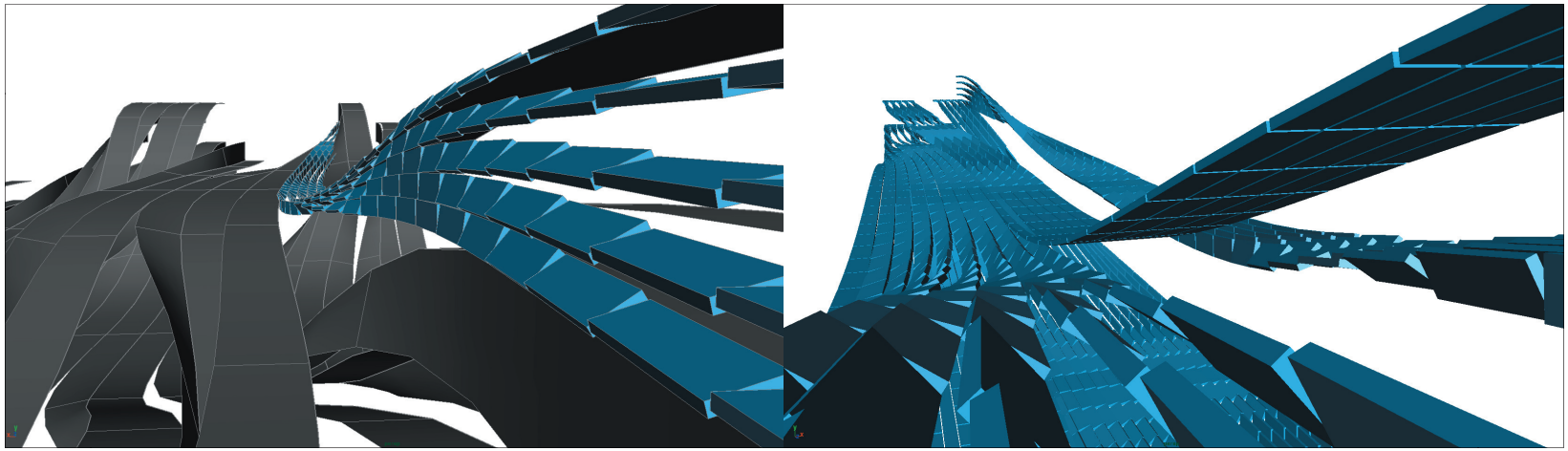


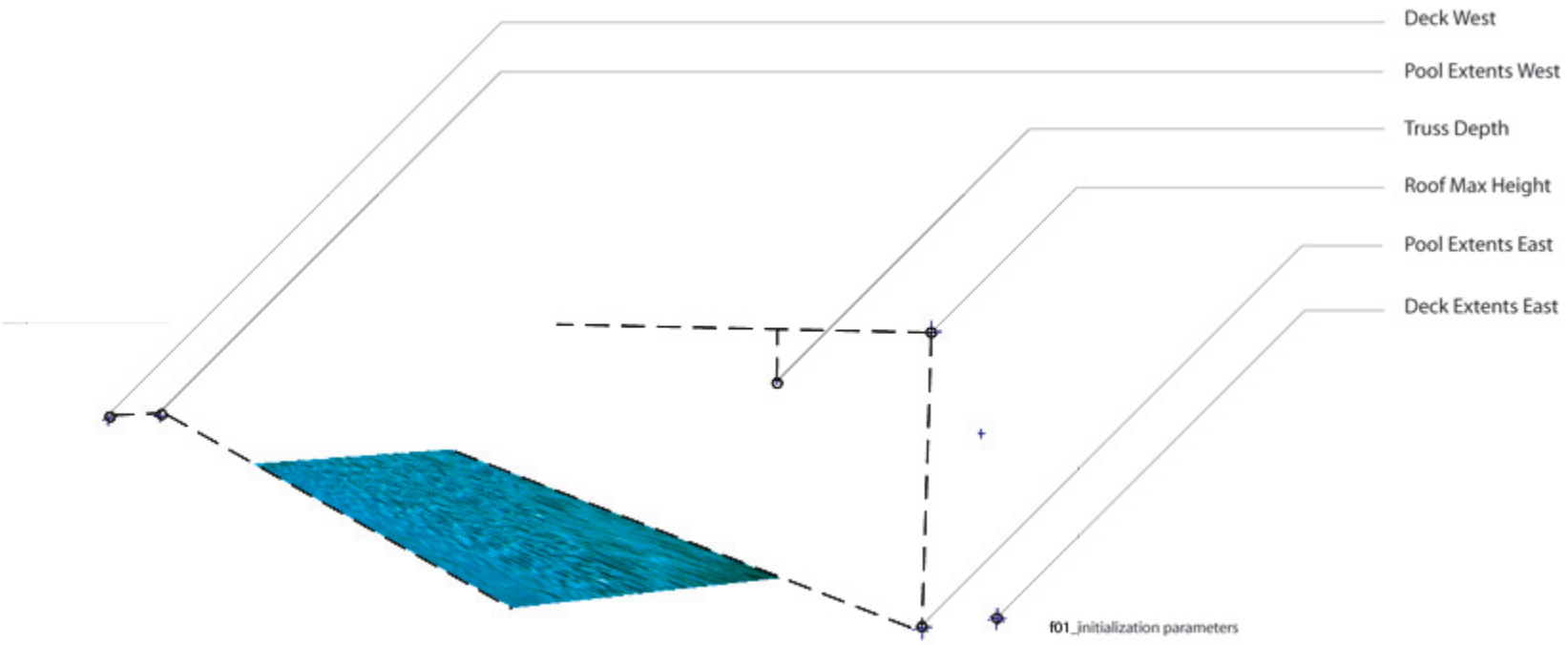
Scripted ribbon reconfiguration on site



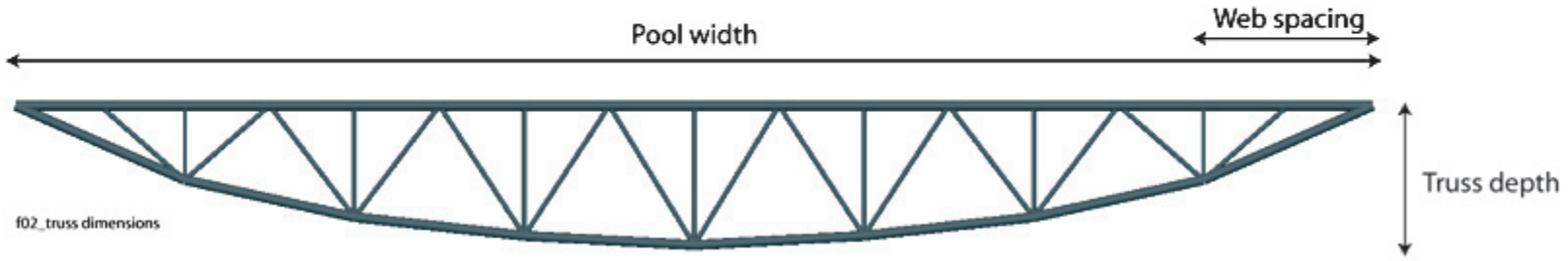

Scripted ribbon reconfiguration on site

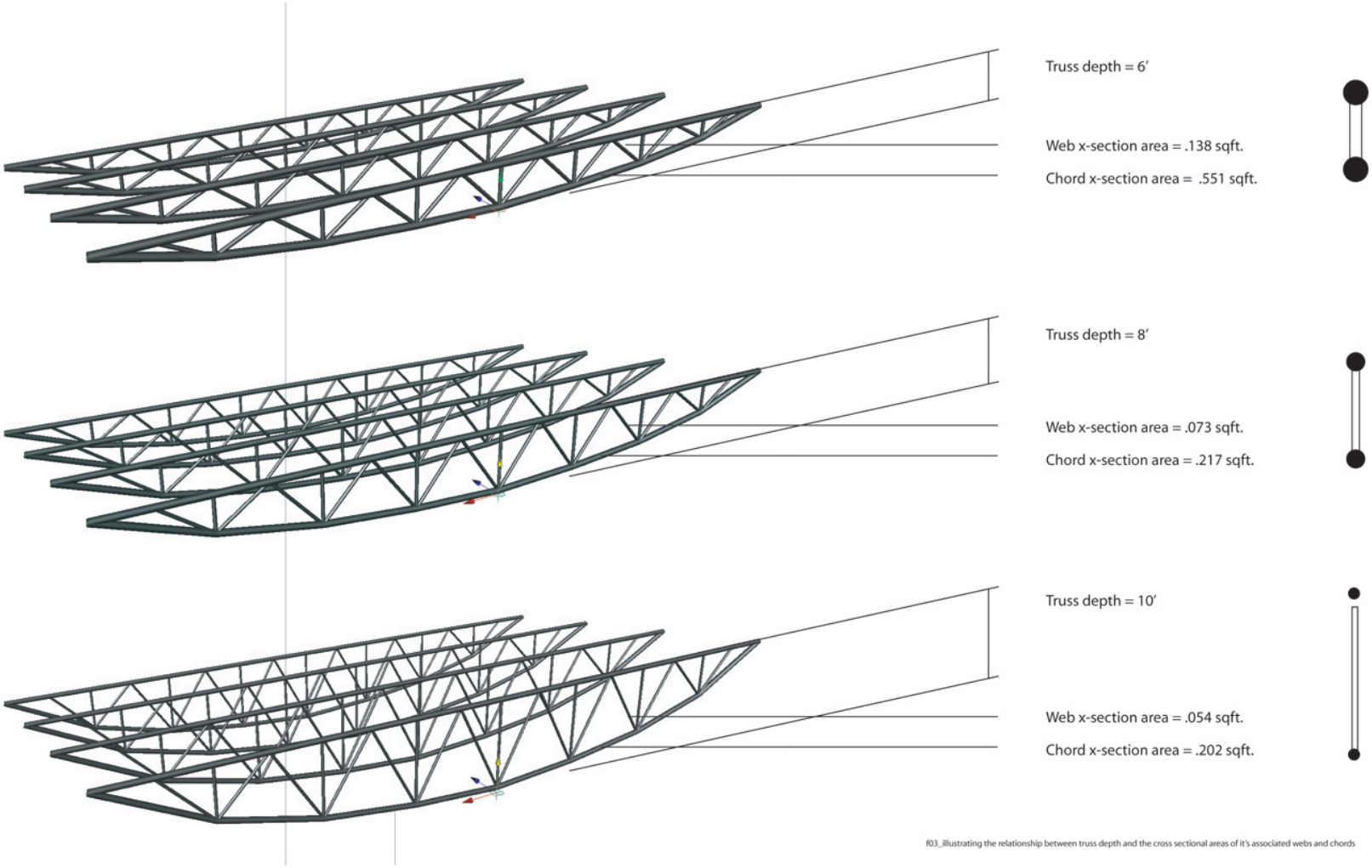
v. Structure





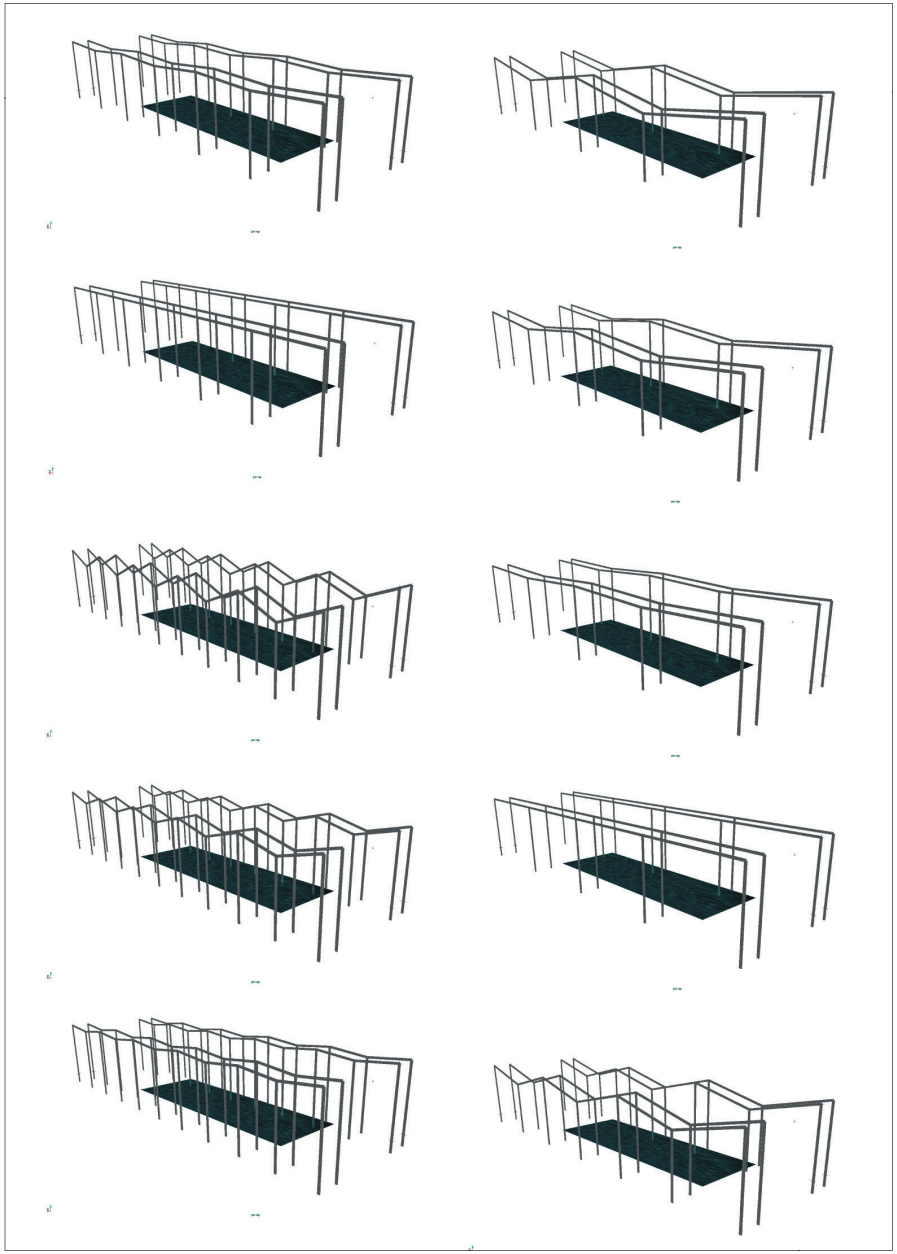

Implementation of `createPoolRoof()` script





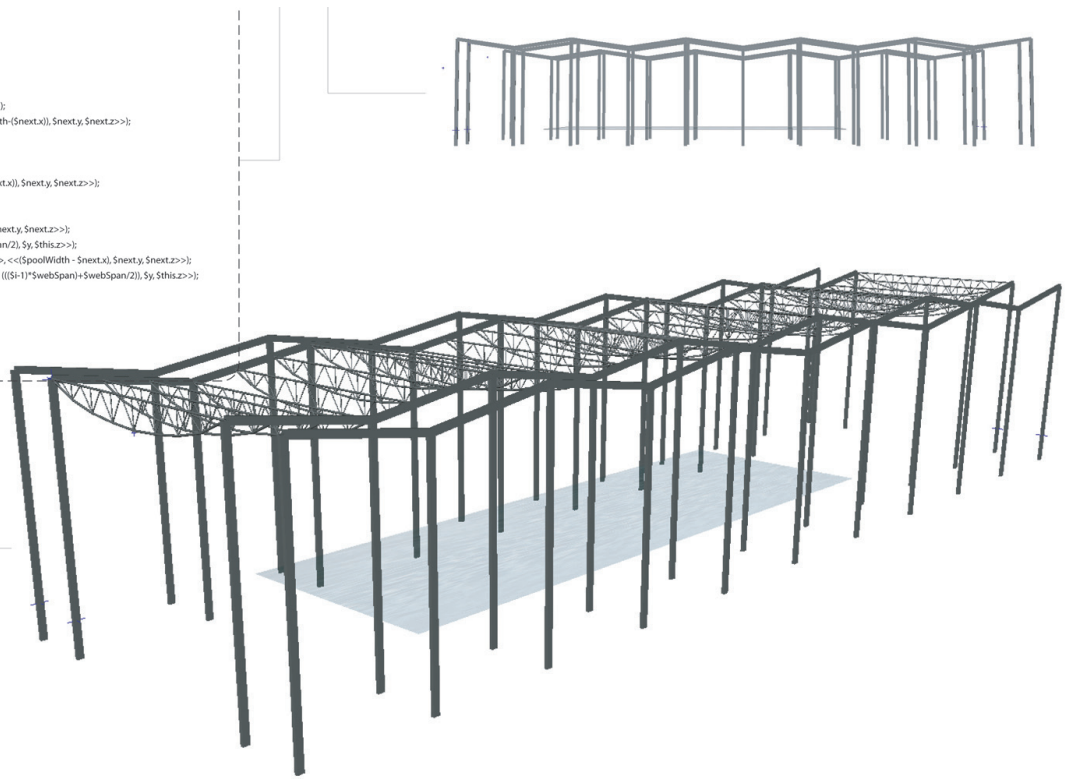

f03_illustrating the relationship between truss depth and the cross sectional areas of it's associated webs and chords

Implementation of `createPoolRoof()` script

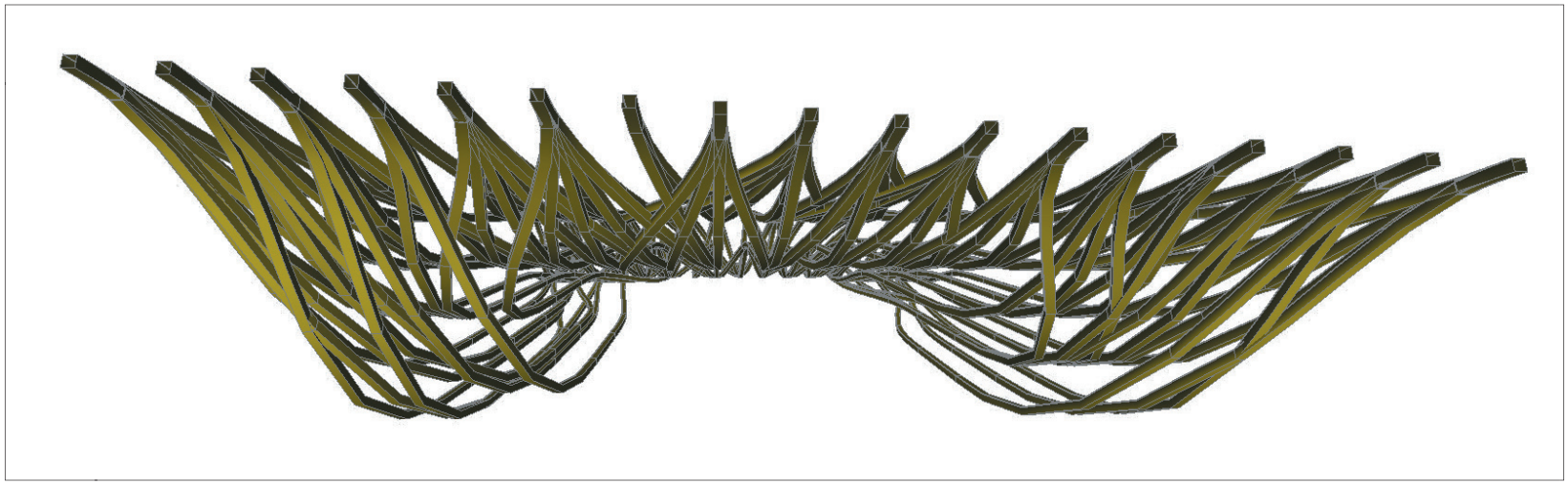


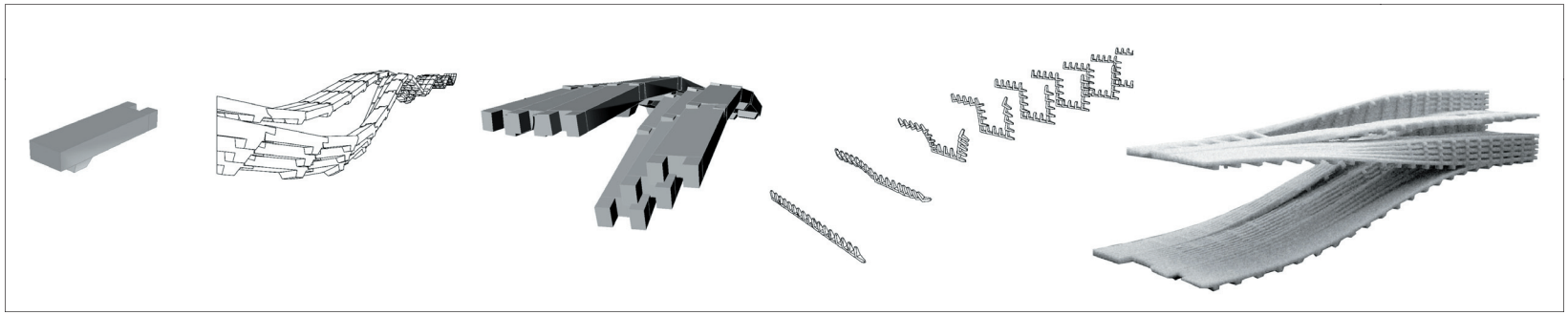
Versioning of `createPool-Roof()` script

```
        for($i=1; $i<=$nWebSpans/2; $i++)
        {
                vector $next = <<($i*$webSpan), ($y-($trussDepth*($c-1)/$c)), $z>>;
                $c += $c;

                // LOWER CHORD
                buildTrussMember(($name+"_web_lowerChord_"+$i+"a"), <<$this.x, $this.y, $this.z>>, <<$next.x, $next.y, $next.z>>);
                buildTrussMember(($name+"_web_lowerChord_"+$i+"b"), <<($poolWidth-($this.x)), $this.y, $this.z>>, <<($poolWidth-($next.x)), $next.y, $next.z>>);

                // VERTICALS
                buildTrussMember(($name+"_web_vertical_"+$i+"a"), <<$next.x, $y, $this.z>>, <<$next.x, $next.y, $next.z>>);
                buildTrussMember(($name+"_web_vertical_"+$i+"b"), <<($poolWidth-($next.x)), $y, $this.z>>, <<($poolWidth-($next.x)), $next.y, $next.z>>);

                // DIAGONALS
                buildTrussMember(($name+"_web_diagonal_"+$i+"a"), <<(($i*$webSpan)-$webSpan/2), $y, $this.z>>, <<$next.x, $next.y, $next.z>>);
                buildTrussMember(($name+"_web_diagonal_"+$i+"b"), <<$this.x, $this.y, $this.z>>, <<((($i-1)*$webSpan)+$webSpan/2), $y, $this.z>>);
                buildTrussMember(($name+"_web_diagonal_"+$i+"c"), <<($poolWidth - (($i*$webSpan)-$webSpan/2)), $y, $this.z>>, <<($poolWidth - $next.x), $next.y, $next.z>>);
                buildTrussMember(($name+"_web_diagonal_"+$i+"d"), <<($poolWidth - $this.x), $this.y, $this.z>>, <<($poolWidth - (((i-1)*$webSpan)+$webSpan/2)), $y, $this.z>>);

                $this = <<($next.x), ($next.y), ($next.z)>>;
        }

        group -n ("trussGroup_"+$name) `getAllObjectsStartingWith($name)`;
}

proc buildTrussMember(string $name, vector $p1, vector $p2)
{
        string $profile;
        if(gmatch($name, "*Chord*"))
                $profile = "chordProfile";
        else
                $profile = "webProfile";

        curve  -d 1     -p ($p1.x) ($p1.y) ($p1.z)
                        -p ($p2.x) ($p2.y) ($p2.z)
                -k 0 -k 1 -n $name;

        extrude -ch true -rn false -po 0 -et 2 -ucp 1 -fpt 1 -upn 1
                -rotation 0 -scale 1 -rsp 1
                -n ($name + "_loft")
                $profile $name;
}

proc string[] getAllObjectsStartingWith(string $prefix)
{
        string $selected[] = (`ls -type "transform" ($prefix + "*")`);
        return $selected;
}

createRoof();
```

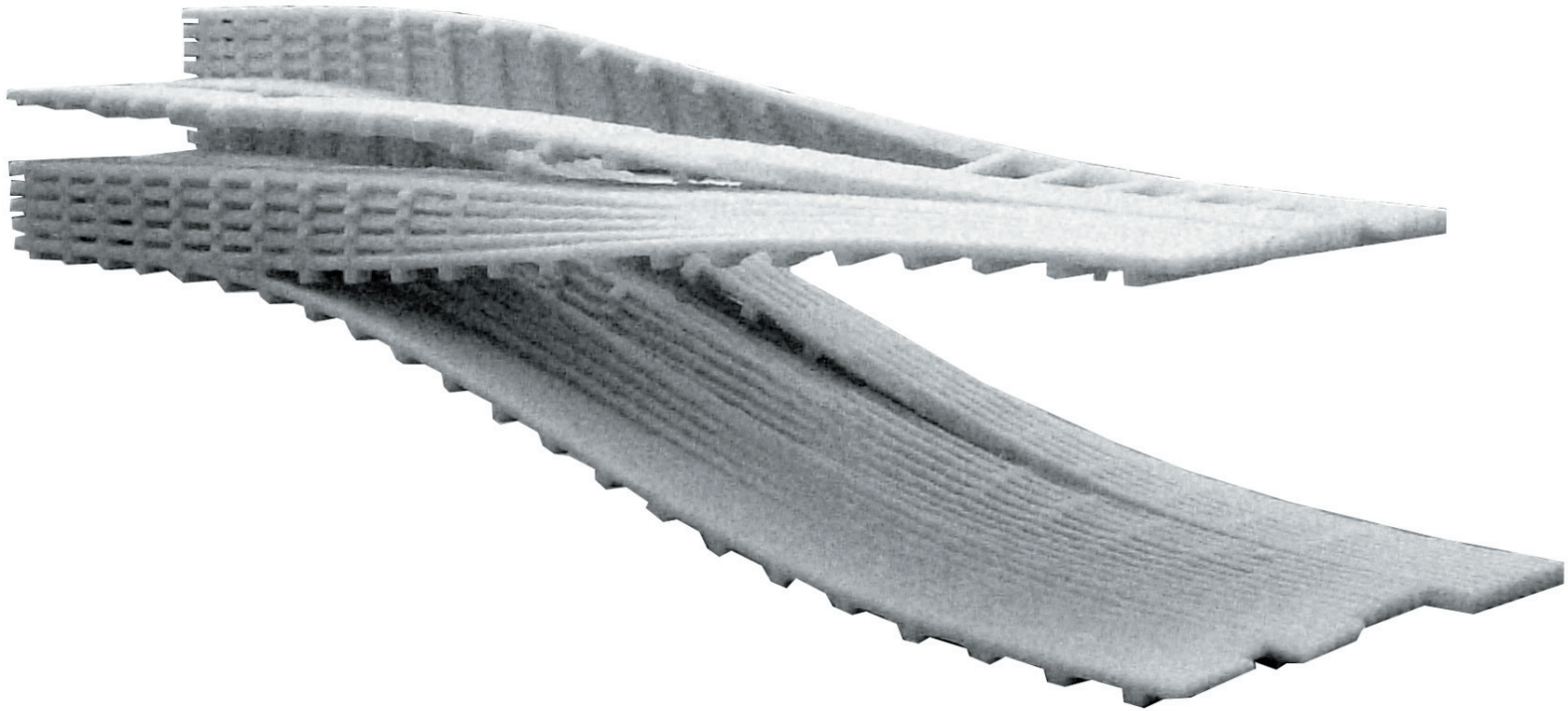


Final implementation of `createPoolRoof()` script



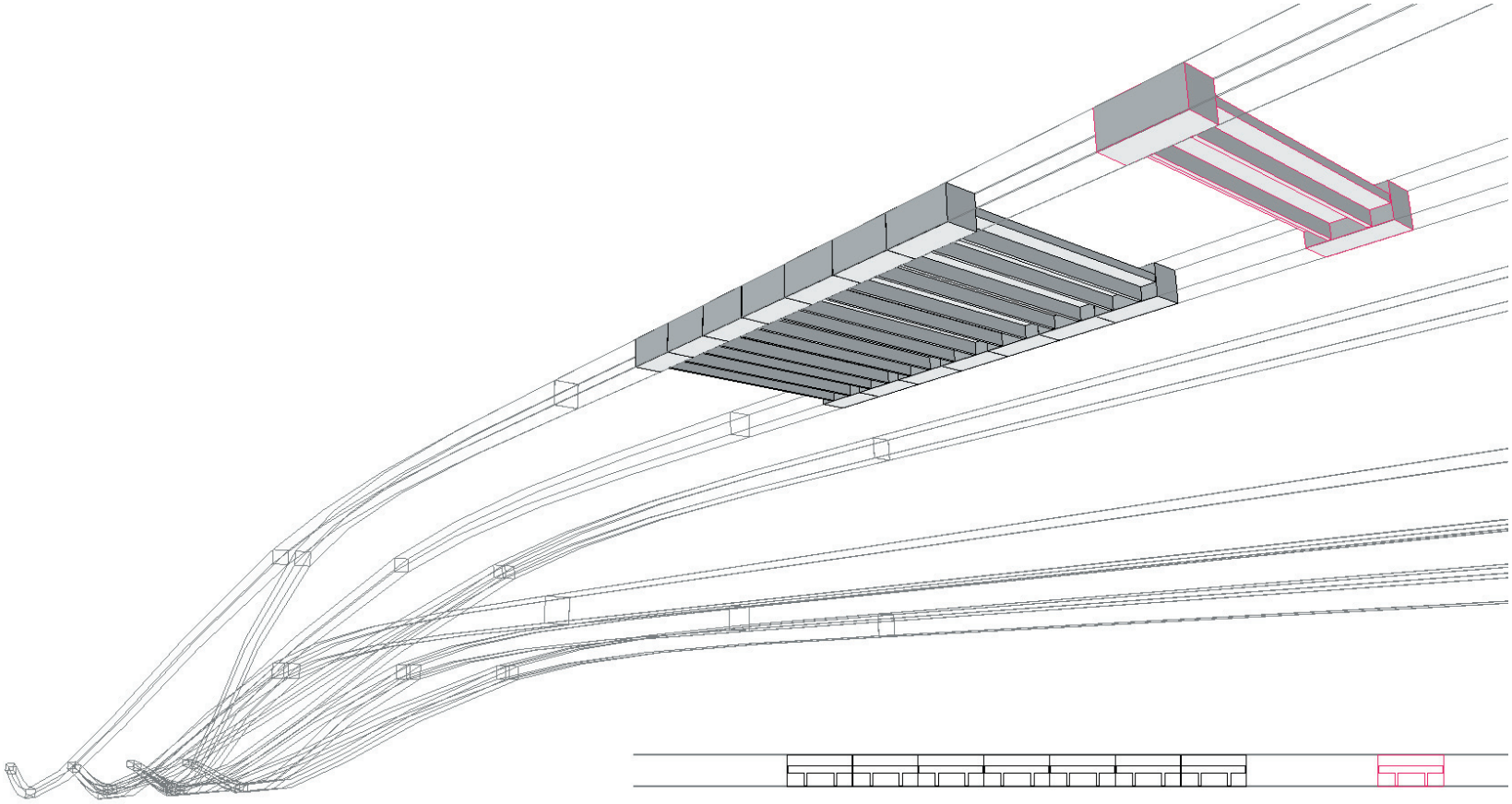


Structural unit aggregation: cast concrete

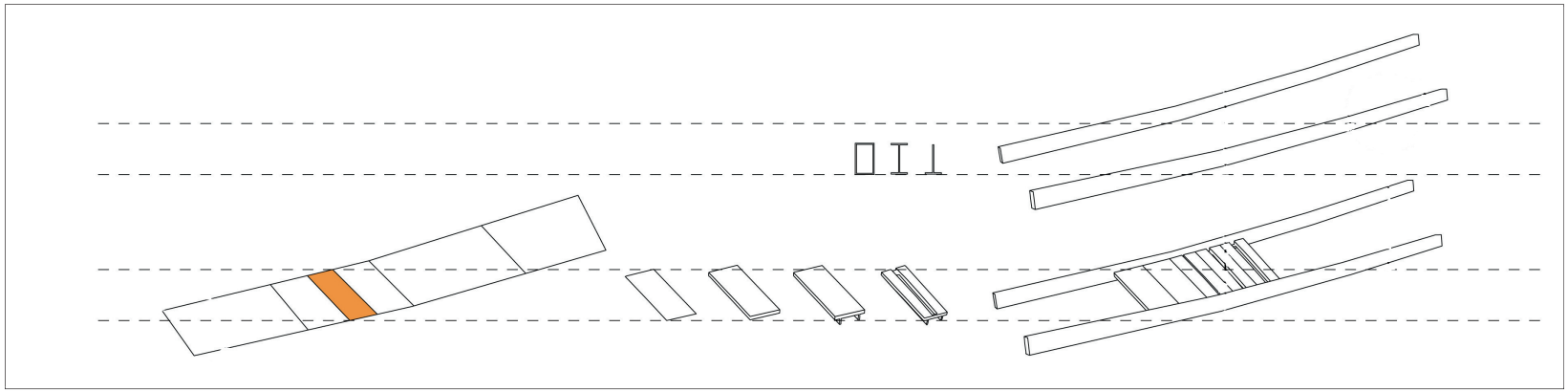


Stereolithograph prototype model of structural unit aggregation

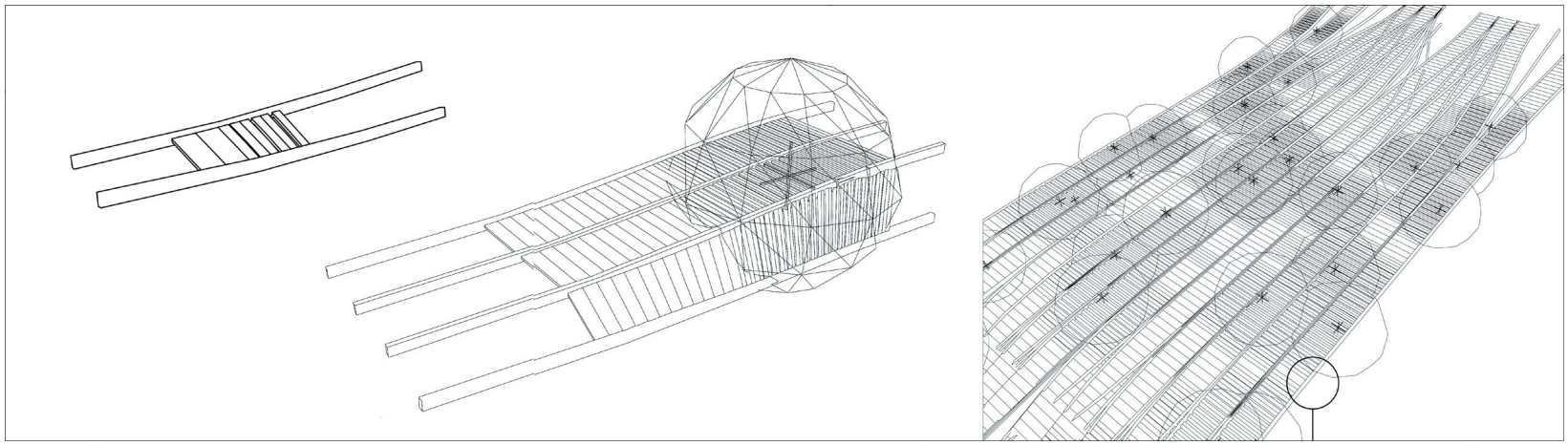
vi. Skin



Concrete double-T unit sketch

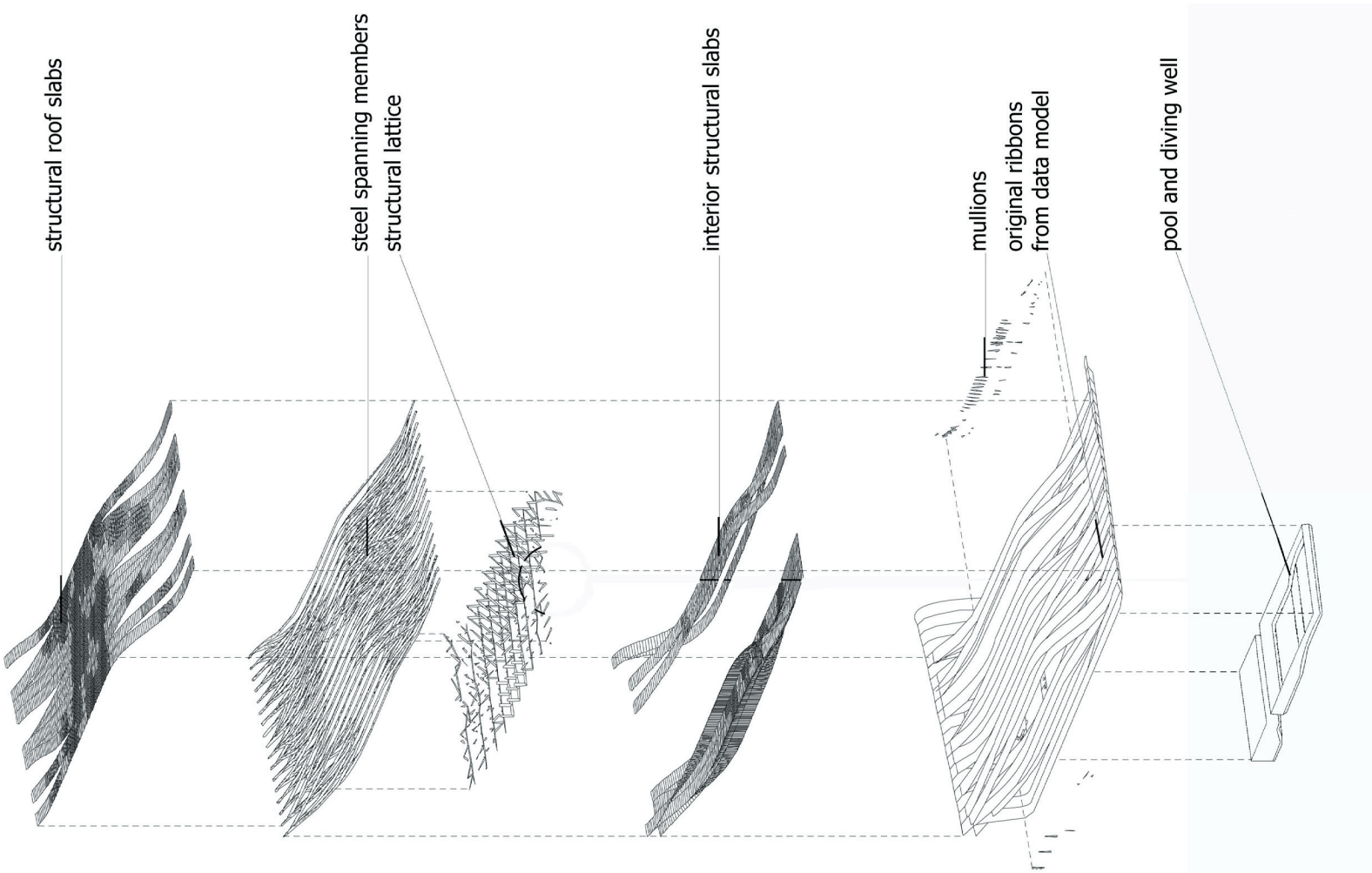


Profile extrusion and initial slab formation

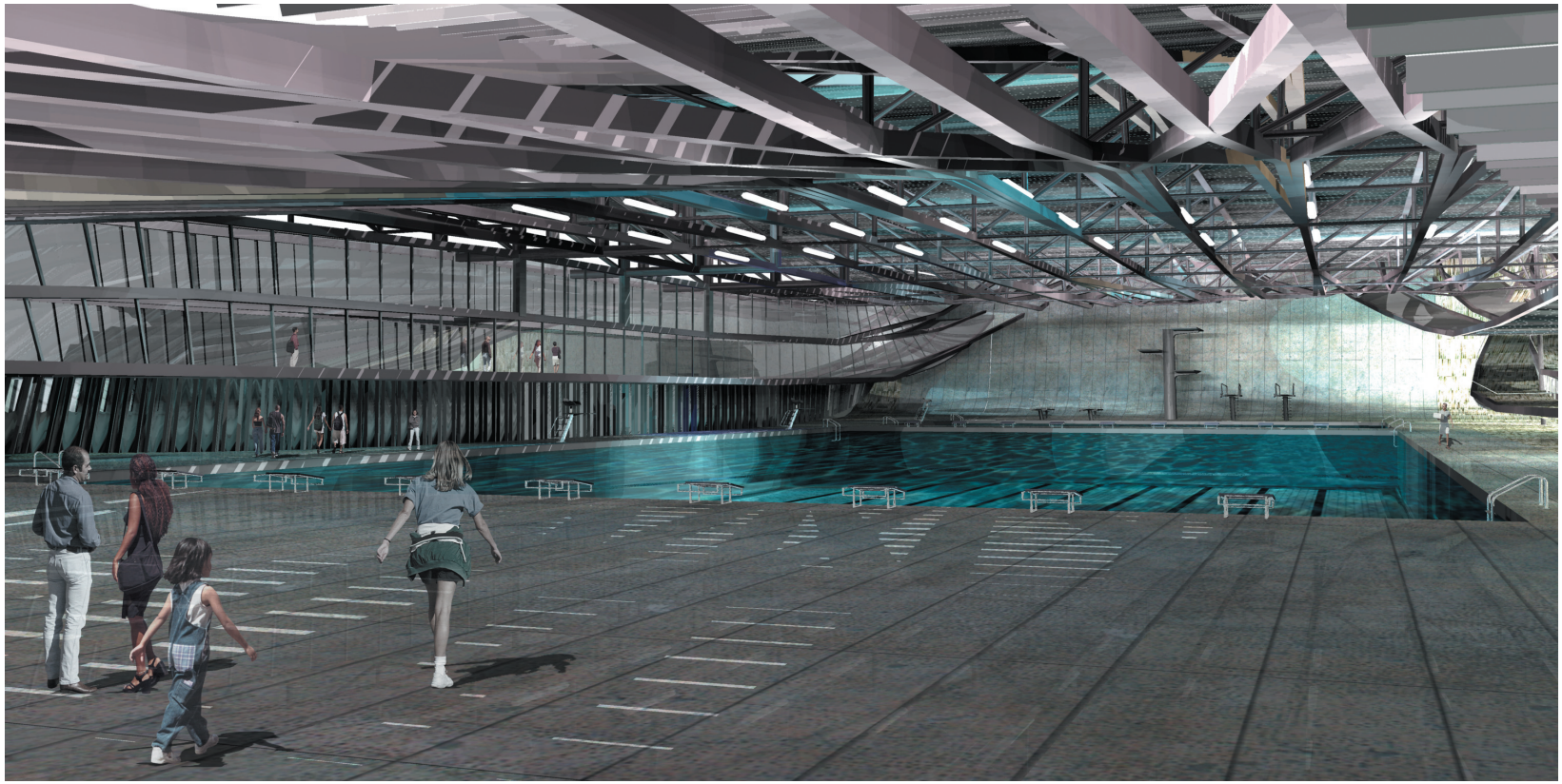


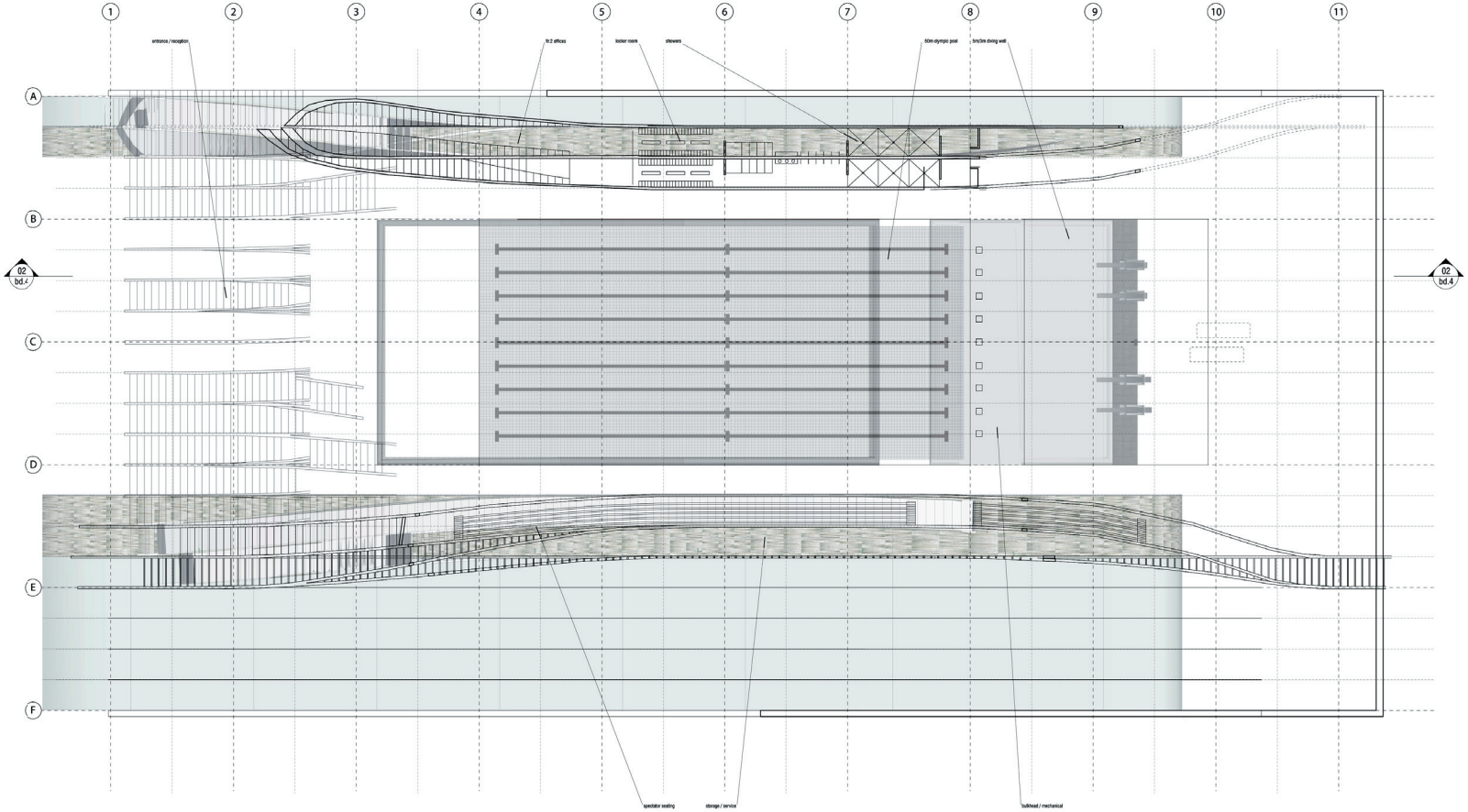
Unit aggregation and effect by light nodes

vii. Building

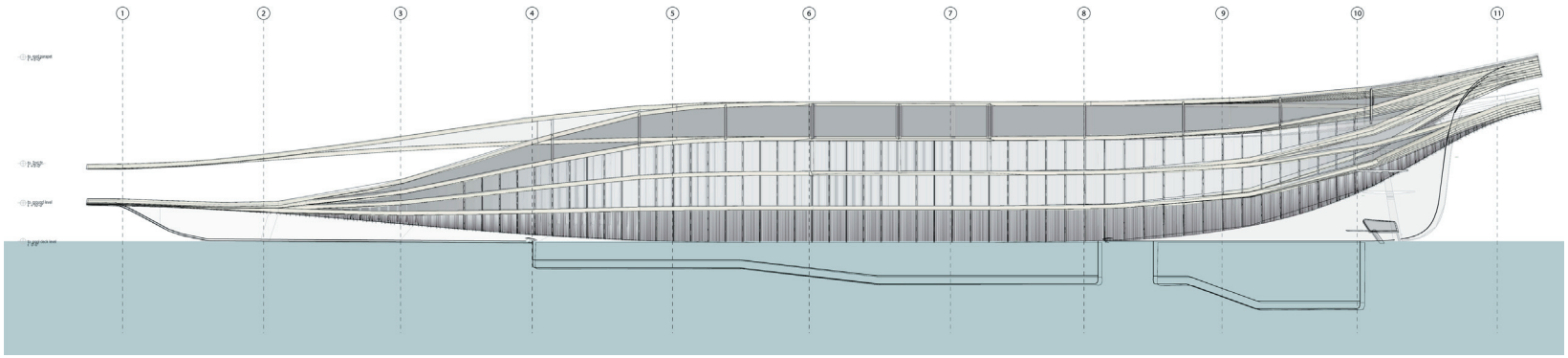



Exploded axo of final proposition

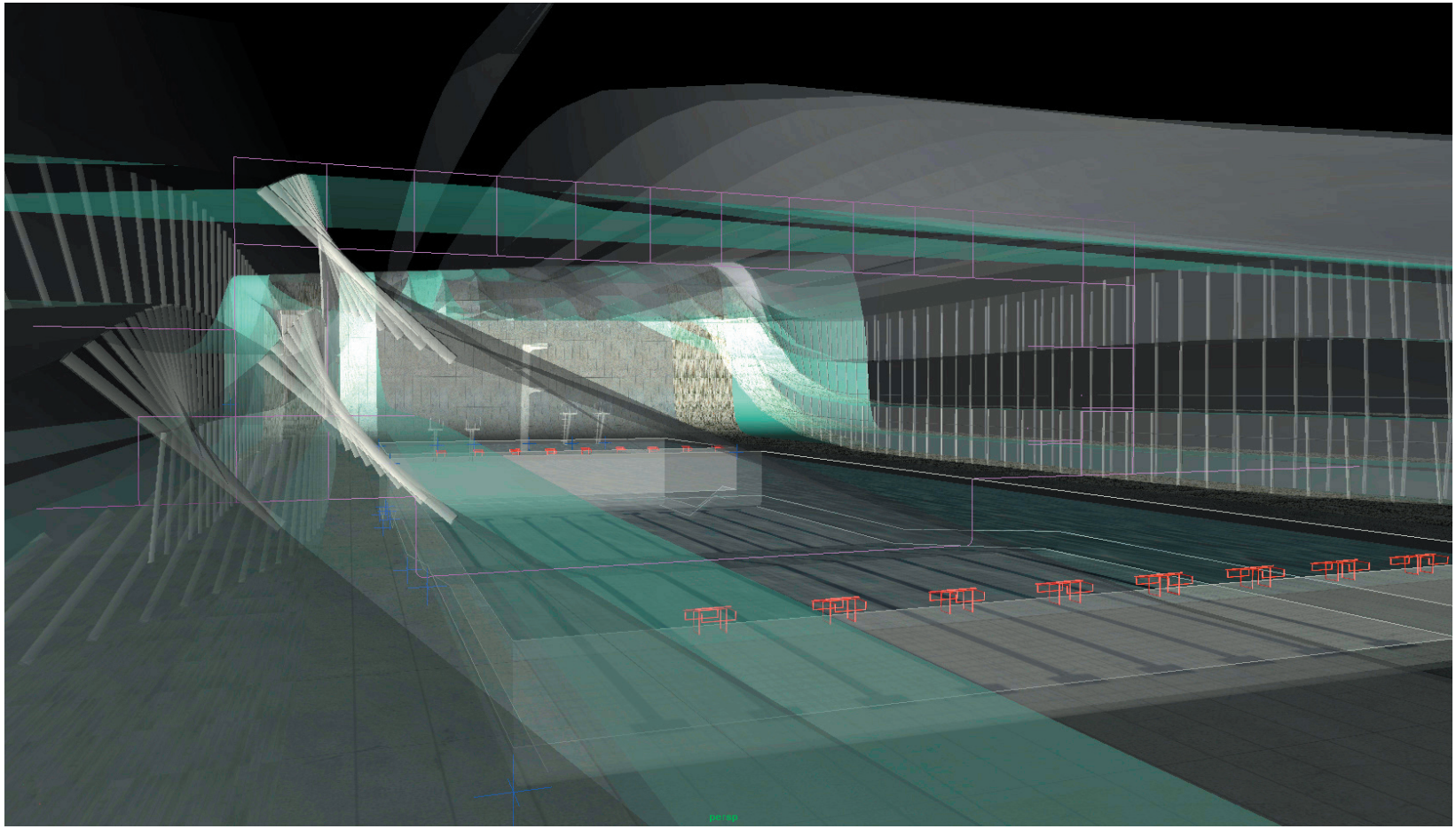


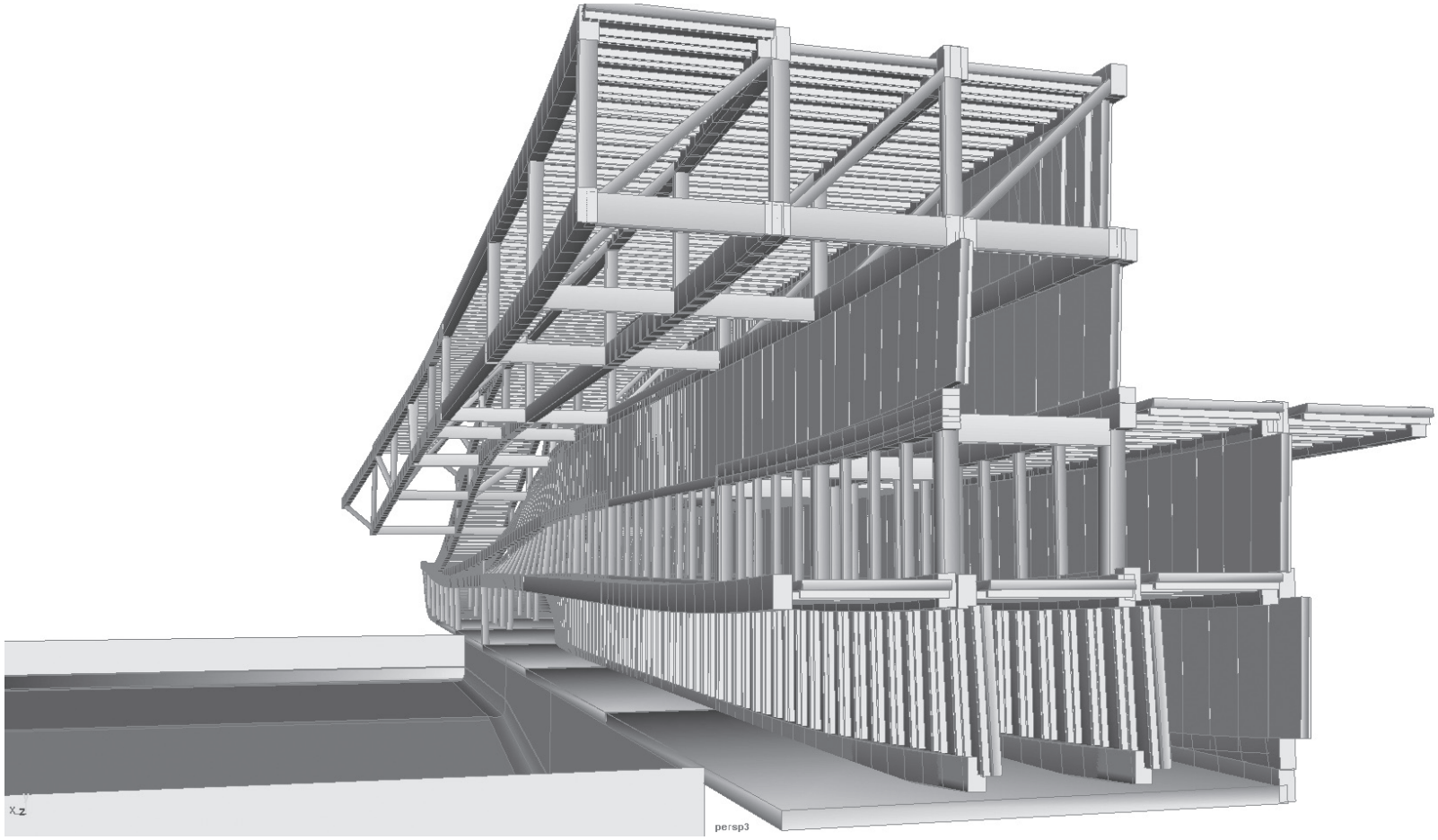

1
2
3
4
5
6
7
8
9
10
11
A
B
C
D
E
F
entrance / reception
fit 2 offices
locker room
showers
50m olympic pool
5m/3m diving well
02
bd.4
spectator seating
storage / service
bulkhead / mechanical

Plan

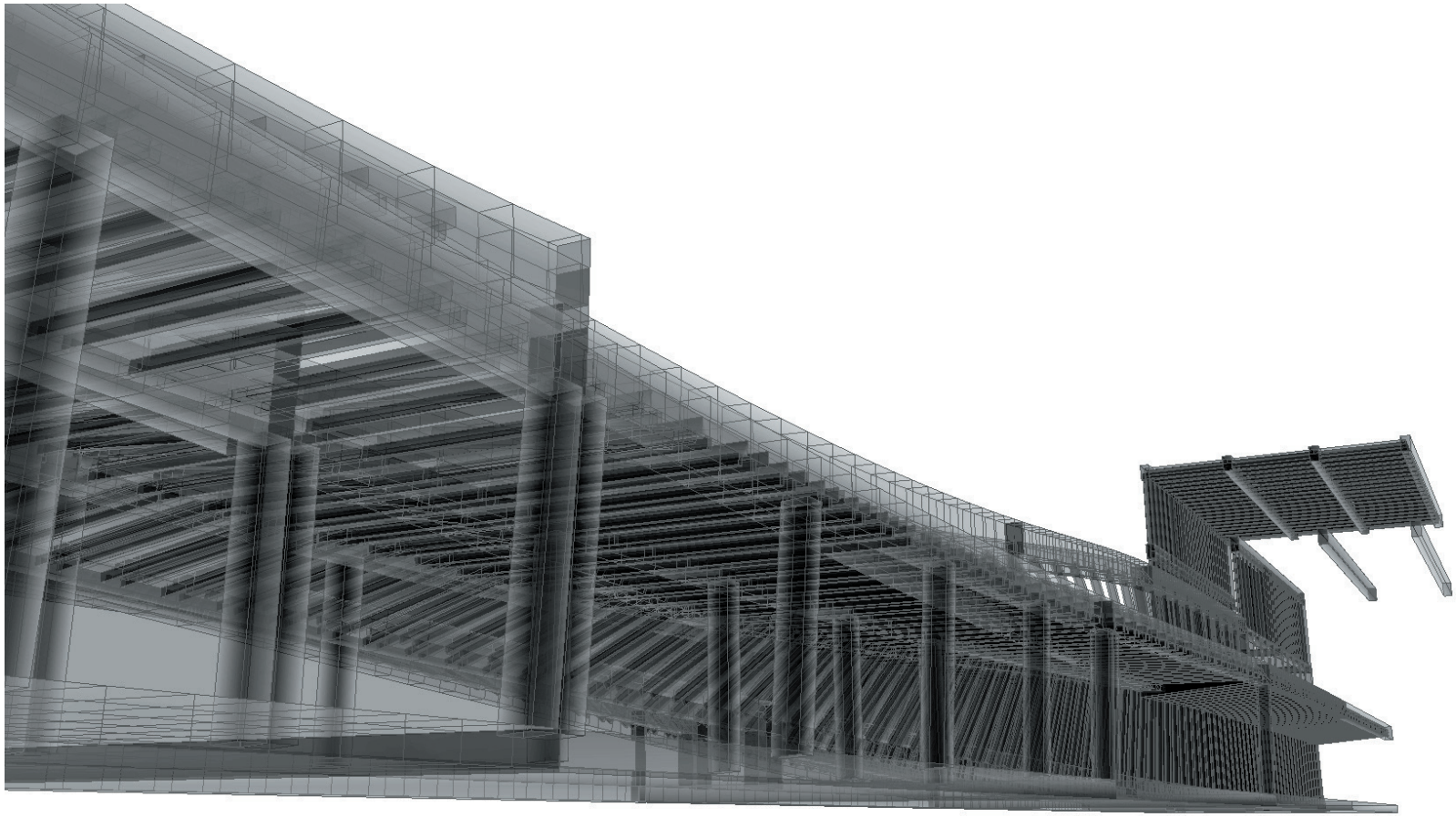



Section

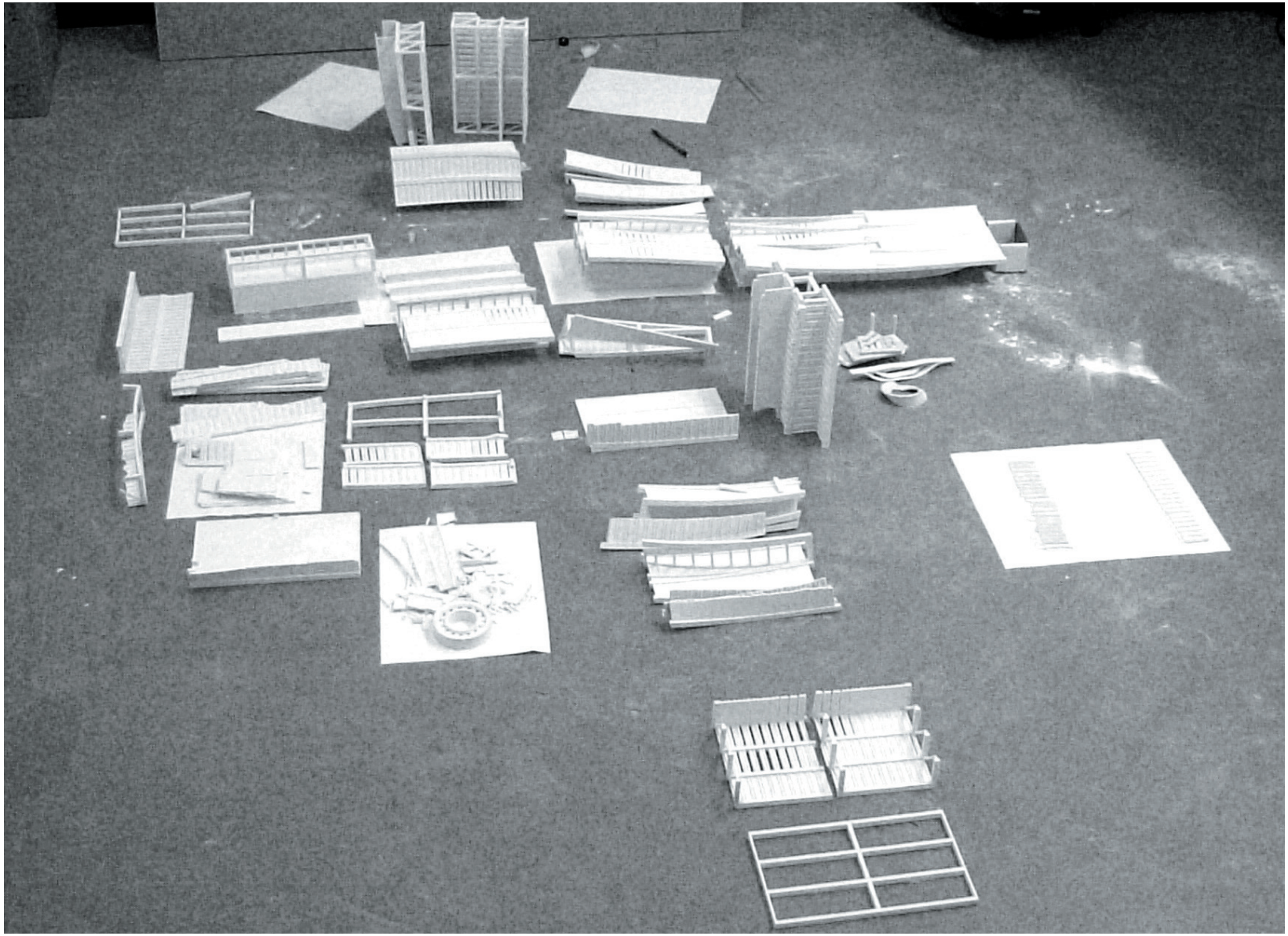




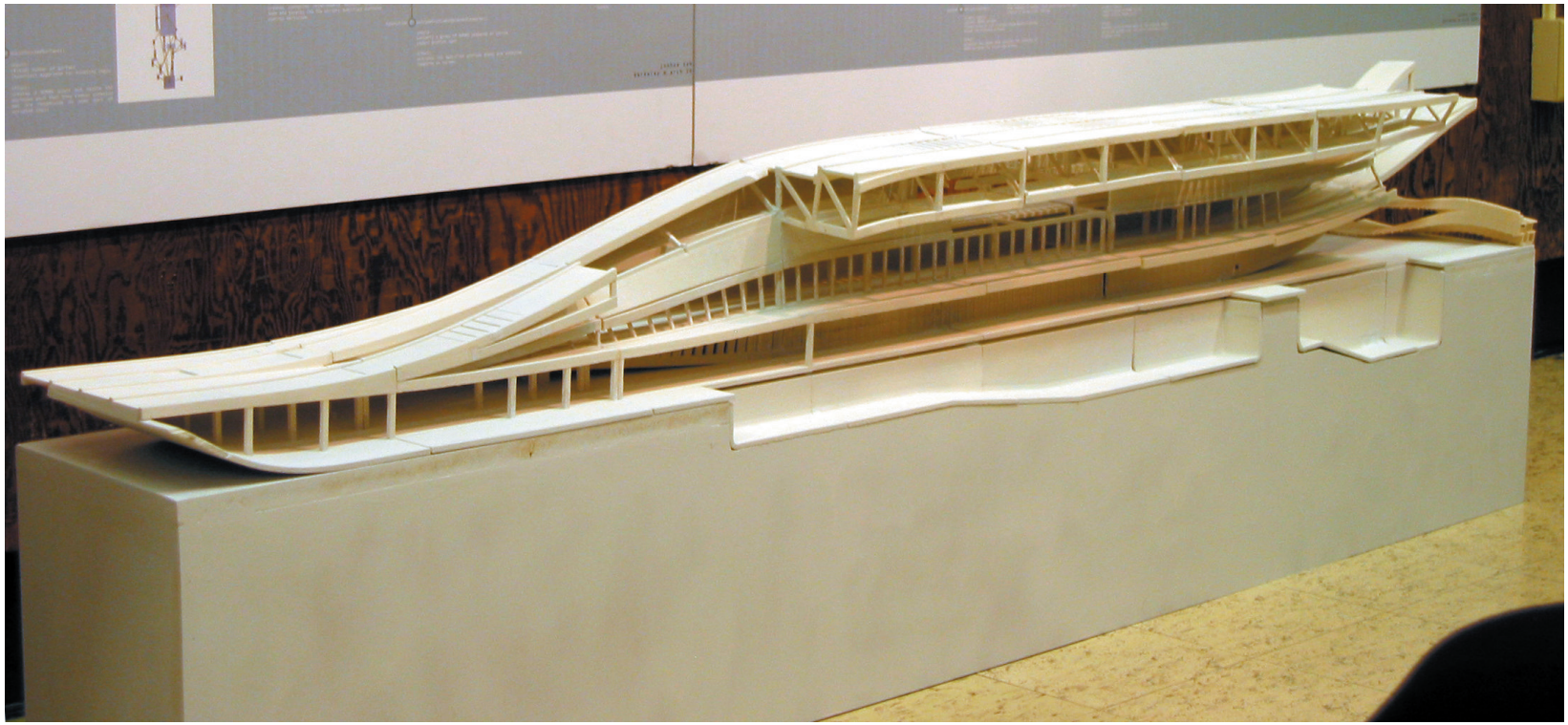

Mesh model for final STL print



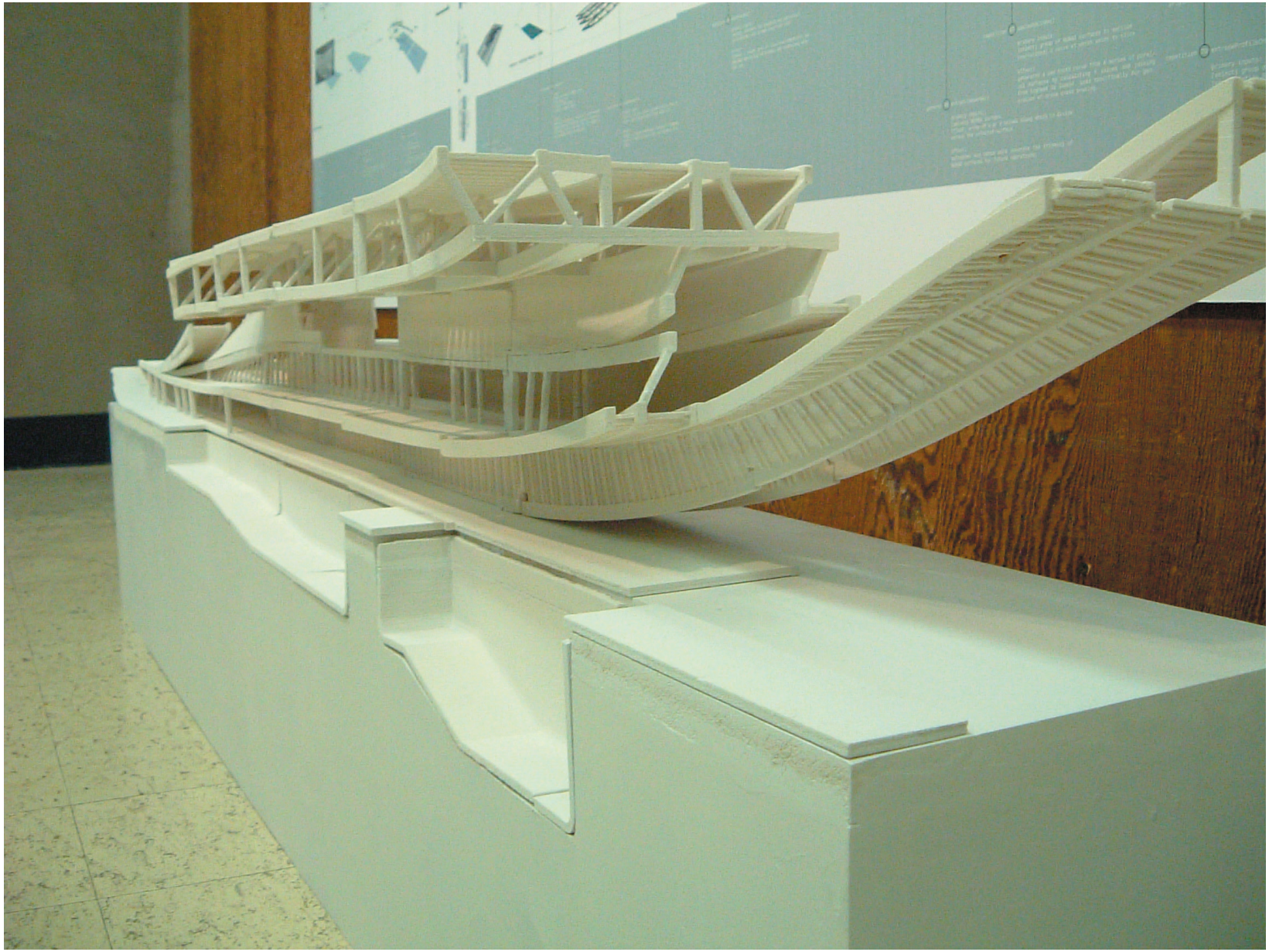
Mesh model for final STL print

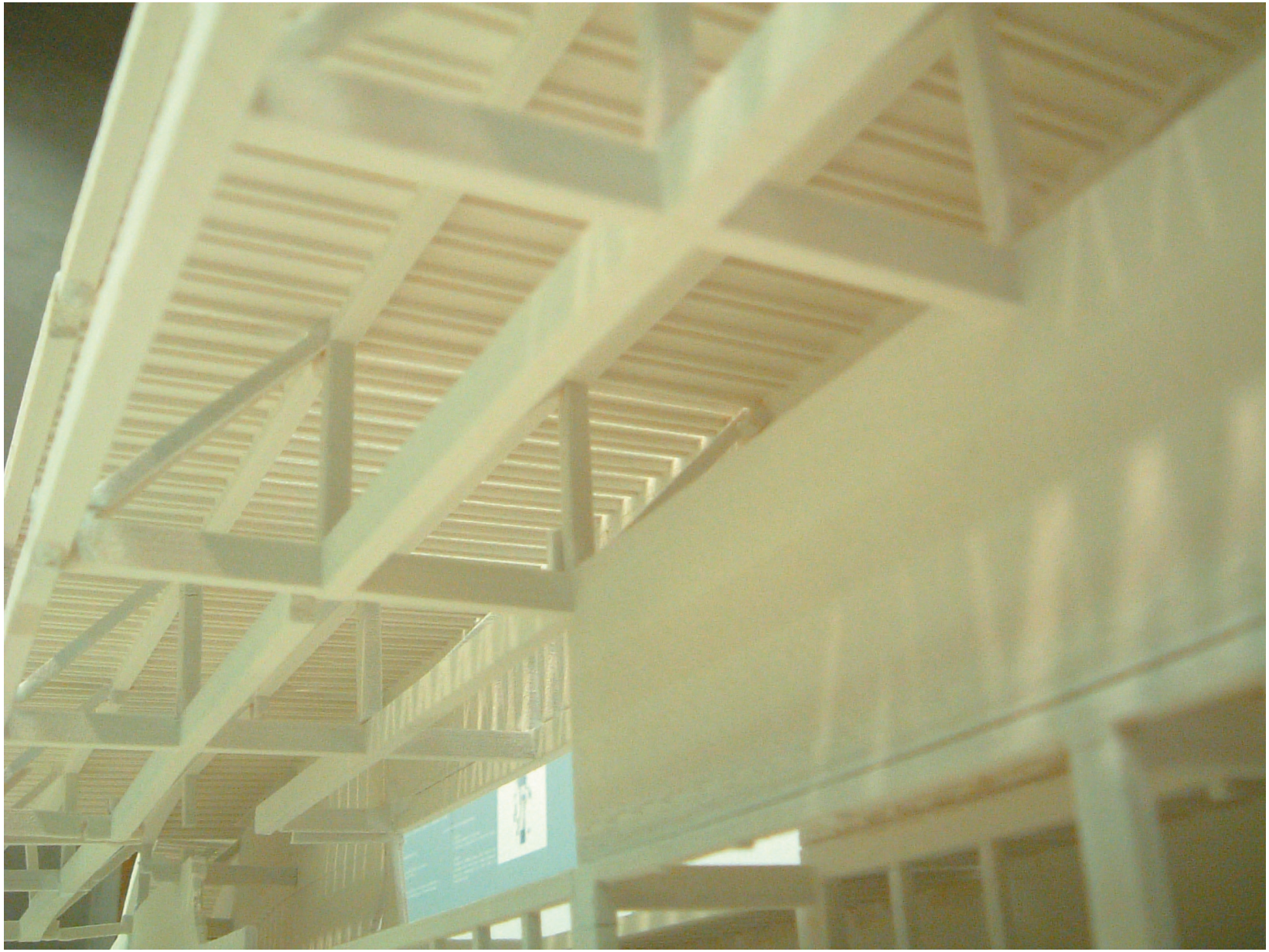


STL assembly



Final STL model





Aftermath, 05.01.2005

# Conclusion

In order for computers to be creative, they must be able to not only represent objects, events, and relationships, but also understand their meaning. [1] Historically, the idea about the role of computers in design is as assistants with tedious number crunching work. Evaluation of meaning and value judgement is then done by the human. [2] But, as computer power increases and systems for artificial intelligence become more sophisticated, it is beginning to seem feasible for computers to make those judgements. The line between which tasks are done independently between the designer and computer gets blurred.

Computers are quite capable of problem solving as long as the environment and performance measures are clearly defined. The problem with design is how to define that environment and performance, if it is even possible. [3] Some have tried to define this environment and establish measures of performance. In A Pattern Language, Christopher Alexander established a rational method for designing buildings that is clear enough to be programmed. However, the success of the measure of performance remains arguable. [4]

In general, the push seems to be for computers to perform more like humans. One finds nothing surprising in searching for the emergence of some sort of intelligence in computer behavior. Nothing surprising, insofar as

artificial intelligence is still the sum of a set of procedures and algorithms, and when such a system is understood completely, the performance is predictable and machine-like. There is no surprise. However, as designers and programmers allow computers to continue to influence their work, there is an equal push in the other direction: a push from computers causing humans to perform more like machines. As the language of machines and algorithms continues to infiltrate our design intentions, and the values of human judgement and performance gets inscribed and encoded as programs, the hybridization results in a new breed of designer, feeding off of the synergy of the human-machine interaction.

Although the scope of the final product of this project is a set of Maya scripts and a design proposition, I hope to extrapolate the implications that this process has for the actual fabrication of buildings and their constituent elements. Alexander begins his *Notes on the Synthesis of Form* with the statement "The ultimate object of design is form." Ultimately, and ideally, the processes described by this project are about the production of objects and their formation. Part of the original intentions of this project were to address these issues explicitly and to explore the realm of human-computer interaction at the level of how digital information gets translated to fabrication machines, and ultimately form. My own desire to more deeply understand the languages within that translation caused the focus to shift entirely towards the programming realm. Methods of computer aided manufacturing, however, remain extremely pertinent to this exploration and in another thesis project those issues would be addressed and would bolster the proposition given here.

At the final review for this project, Neil Denari recognized a relationship in the final proposition between a rational, clinical approach to design and something that is about an idea of site and flow. While the focus of my presentation was on the process and scripting, he raised a point about "desire" as somewhere to look for other

means to argue for the project. Agendas and plans of action that are larger will then come forth as arguments, and they are debatable because they are rooted in desire. These greater ideas are what propositions of architecture are really about.

# Endnotes

1 - Introduction

1 Branko Kolarevic, *Architecture in the Digital Age: Design and Manufacturing.* (New York: Spon Press, 2002), 13.
2 Hans Walser, *The Golden Section*. (USA: The Mathematical Association of America, 2001).
3 Le Corbusier
4 Ben van Berkel and Caroline Bos, *MOVE*. (Amsterdam: UN Studio & Goose Press, 1999). 15.
5 Stuart Russel and Peter Norvig. *Artificial Intelligence a Modern Approach*. (Upper Saddle River: NJ, Prentice Hall, 2003).
6 MEL - Maya Embedded Language: An interpreted scripting language for quick access and control of Maya's functions.
7 Neil Leach. *Digital Tectonics*. (Great Britain: Wiley-Academy, 2004). 71.

2 - Precedents

1 Neil Leach. *Digital Tectonics*. (Great Britain: Wiley-Academy, 2004). 129-135.
2 Branko Kolarevic, *Architecture in the Digital Age: Design and Manufacturing.* (New York: Spon Press, 2002), 53.
3 Benjamin H. Bratton, "The Premise of Recombinant Architecture: One." *Architectura e cultura digitale*, April 6, 2003, https://www.autistici.org/mailman/public/rekombinant/2003-April/002787.html
4 Branko Kolarevic, *Architecture in the Digital Age: Design and Manufacturing* (New York, Spon Press, 2002), 125. For more information see: Gernot Brauer (ed), A. von Iersel (trans), *Architecture as Brand Communication: Dynaform + Cube*. (Berlin: Birkhauser, 2002)
5 Albert Ferre, Tomoko Sakamoto, and Michael Kubo (ed), *The Yokohama Project, Foreign Office Architects* (Barcelona: Actar, 2002). 10-13.
6 Kostas Terzidis, "Algorithmic Architecture." 2004, http://www.bol.ucla.edu/~kostas/algorithmicArchitecture.html

3 - Project

1 Von Dietrich Fabian, Moderne Schwimmstatten der Welt. (Carl Schunemann Verlang Bremen, 1963). 41.

4 - Conclusion

1 Kostas Terzidis, *Expressive Form: A Conceptual Approach to Computational Design*. (London: Spon Press, 2004).
2 Christopher Yessios, "Syntactic Structures for Site Planning", in W.F.E.Preiser (ed.) Environmental

Design Research, Proceedings of the EDRA 4 Conference. (Stroudsbourg: Dowden, Hutchinson and Ross, 1973).
3 Kostas Terzidis, Expressive Form: A Conceptual Approach to Computational Design. (London: Spon Press, 2004).
4 Christopher Alexander, Notes on the Synthesis of Form. (Cambridge, MA: Harvard University Press, 1966).

# Bibliography


*A+A Architectureanimation* 2002, Collegi d'Arquitectes de Catalunya.

*Bits and Spaces 2001*, Birkhauser, Basel, Boston, Berlin.

Bratton, Benjamin H. "The Premise of Recombinant Architecture: One." *Architectura e cultura digitale*, April 6, 2003, https://www.autistici.org/mailman/public/rekombinant/2003-April/002787.html

Brauer, G. (Editor), von Iersel, A. (Translator). *Architecture as Brand Communication: Dynaform + Cube*. Berlin: Birkhauser, 2002.

Alexander, Christopher. *Notes on the Synthesis of Form*, (Cambridge, MA: Harvard University Press, 1966).

Boyer, C.M. *Cybercities*. New York, NY: Princeton Architectural Press, 1996.

DeLanda, M. *A Thousand Years of Nonlinear History*. New York, NY: Swerve, 2000.

Dollens, D. *Digital to Analog*, New Mexico: SITES Books, 2001.

Estevez, A. *Genetic Architectures*, New Mexico: Sites Books, 2003.

Fabian, Von Dietrich, *Moderne Schwimmstatten der Welt*. Carl Schunemann Verlang Bremen, 1963.

Ferre, Albert, Tomoko Sakamoto and Michael Kubo, eds. *The Yokohama Project, Foreign Office Architects*. Barcelona: Actar, 2002.

Galofaro, L. Digital Eisenman: An Office of the Electronic Era, Switzerland: Birkhauser, 1999.

Horan, T. *Digital Places, Building our City of Bits*. Washington: Urban Land Institute, 2000.

Johnson, S. *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*, New York, NY: Scribner, 2001.

Kelly, Kevin *Out of Control*, Cambridge, MA: Basic Books, 1994.

Kolarevic, Branko. *Architecture in the Digital Age: Design and Manufacturing*, New York: Spon Press, 2003.

Kwinter, S. *Architectures of Time: Toward a Theory of the Event in Modernist Culture*, Massachusetts: MIT Press, 2001.

Leach, Neil. *Digital Tectonics*. Great Britain: Wiley-Academy, 2004.

Lindsey, B., *Digital Gehry: Material Resistance Digial Construction*, Switzerland: Birkhauser, 2001.

Liu, Y. *Developing Digital Architecture*. Boston: Birkhauser, 2003.

McCullough, M. *Digital Ground, Architecture, Perfasive Computing, and Environmental Knowing*, Massachusetts: MIT Press, 2004.

Mitchell, W. *The Logic of Architecture: Design, Computation, and Cognition*, Cambridge, Massachusetts: MIT Press, 1990.

Russell, Stuart and Peter Norvig. *Artificial Intelligence a Modern Approach*, Upper Saddle River, NJ: Prentice Hall, 2003.

SHoP/Sharples Holden Pasquarelli (ed). *Versioning: Evolutionary Techniques in Architecture* Great Britain: Wiley-Academy, 2002.

Terzidis, Kostas. "Algorithmic Architecture." 2004, http://www.bol.ucla.edu/~kostas/algorithmicArchitecture.html

Terzidis, Kostas. *Expressive Form: A Conceptual Approach to Computational Design*. London: Spon Press, 2004.

van Berkel, B. & Bos, C. *MOVE*, Amsterdam: UN Studio & Goose Press, 1999.

Yu, L. *Number-Based Design Reasoning Systems*, Technische Universiteit Delft: Publikatieburo Bouwkunde, 1994.

# A. MEL Scripts

## i. General

### G|01. `annotateWithoutLeader()`

```
proc annotateWithoutLeader()
{
	string $text;
	string $result = `promptDialog
		-title "AnnotateNoLeader"
		-message "Text:"
		-button "OK" -button "Cancel"
		-defaultButton "OK" -cancelButton "Cancel"
		-dismissString "Cancel"`;
	if($result == "OK" && `promptDialog -q -text` != "")
	{
		$text = `promptDialog -query -text`;

		string $selected[] = `selectedNodes`;

		int $i=0;
		for($i=0; $i<size($selected); $i++)
		{
			string $node = $selected[$i];
			float $pos[] = `objectCenter $node`;
			annotate -text $text -p $pos[0] $pos[1] $pos[2] $node;
			setAttr "annotationShape1.displayArrow" 0;
			parent "annotation1" $node;
			rename "annotation1" ("annotation" + $i);
		}
	}
}
```

## G|02. buildTwistedSurface()

```
proc construct()
{
	string $planeID = "sourcePlane2";
	string $existingShader = "blinn1";
	int $i;

	nurbsPlane -n ($planeID +"1") -p 1 0 2 -ax 0 1 0 -w 2 -lr 2 -d 3 -u 20 -v 6 -ch 1;
		assignSG $existingShader ($planeID +"1");
	for($i=1; $i<20; $i++)
		detachSurface -n ($planeID +($i+1)) -ch 1 -rpo 1
			($planeID +$i +".u["+($i*0.05)+"]");

//	nurbsPlane -n ($planeID +"21") -p 1 0 2 -ax 0 1 0 -w 2 -lr 2 -d 3 -u 20 -v 6 -ch 1;
//		assignSG $existingShader ($planeID +"21");
//	for($i=1; $i<20; $i++)
//		detachSurface -n ($planeID +($i+21)) -ch 1 -rpo 1
//			($planeID +($i+20) +".u["+($i*0.05)+"]");

	xformGroup2($planeID);
	select -cl;
}

proc xformGroup2(string $planeID)
{
	int $i;
	for($i=1; $i<=20; $i++)
	{
		int $rot = ($i%5==2 ? -90 : ((($i-1)%5)%2)*90);
		float $x = ((($i-1)%5)*.05)-.1)*-1;
		float $y = -.2 + abs((($i-1)%5)-2))*.05;

		xformEnd($planeID, $i, $rot, $x, $y);
	}

/*	xformEnd($planeID, 1, 0, .1, -.1);
	xformEnd($planeID, 2, -90, .05, -.15);
```

```
	xformEnd($planeID, 3, 0, 0, -.2);
	xformEnd($planeID, 4, 90, -.05, -.15);
	xformEnd($planeID, 5, 0, -.1, -.1);

	xformEnd($planeID, 6, 0, .1, -.1);
	xformEnd($planeID, 7, -90, .05, -.15);
	xformEnd($planeID, 8, 0, 0, -.2);
	xformEnd($planeID, 9, 90, -.05, -.15);
	xformEnd($planeID, 10, 0, -.1, -.1);

	xformEnd($planeID, 11, 0, .1, -.1);
	xformEnd($planeID, 12, -90, .05, -.15);
	xformEnd($planeID, 13, 0, 0, -.2);
	xformEnd($planeID, 14, 90, -.05, -.15);
	xformEnd($planeID, 15, 0, -.1, -.1);

	xformEnd($planeID, 16, 0, .1, -.1);
	xformEnd($planeID, 17, -90, .05, -.15);
	xformEnd($planeID, 18, 0, 0, -.2);
	xformEnd($planeID, 19, 90, -.05, -.15);
	xformEnd($planeID, 20, 0, -.1, -.1);
*/
	xformSequence($planeID, 1, 20, 0, 0, -.3);

	groupSequence("redGroup", $planeID, 1, 20);
	move 0 .2 0 redGroup;
}

proc xformGroup1(string $planeID)
{
	xformEnd($planeID,  1, 0, 0, 0.3);
	xformEnd($planeID,  2, 90, -0.05,0.35);
	xformEnd($planeID,  3, 0, -0.1, 0.3);
	xformEnd($planeID,  4, 90, -0.15, 0.35);
	xformEnd($planeID,  5, 0, -0.2, 0.4);
	xformEnd($planeID,  6, 0, -0.3, 0.5);
	xformEnd($planeID,  7, -90, -0.35, 0.45);
	xformEnd($planeID,  8, 0, -0.5, 0.3);
	xformEnd($planeID,  9, 0, -0.3, 0.3);
```

```
xformEnd($planeID, 10, -90, -.35, 0.25);
xformEnd($planeID, 11, 0, -.2, .1);
xformEnd($planeID, 12, 90, -0.25, 0.15);
xformEnd($planeID, 13, 0, -.1, .2);
xformEnd($planeID, 14, -90, -0.15, 0.15);
xformEnd($planeID, 15, 0, -.2, .1);
xformEnd($planeID, 16, 0, -.1, .1);
xformEnd($planeID, 17, 0, 0, .1);
xformEnd($planeID, 18, 0, 0, .1);
xformEnd($planeID, 19, 0, 0, .1);
xformEnd($planeID, 20, 0, 0, .1);

xformEnd($planeID, 21, 90, -.05, -.05);
xformEnd($planeID, 22, 0, -.1, 0);
xformEnd($planeID, 23, 0, -.2, .1);
xformEnd($planeID, 24, 0, -.1, .2);
xformEnd($planeID, 25, -90, -.15, .15);
xformEnd($planeID, 26, 0, -.1, -.1);
xformEnd($planeID, 27, 0, -.2, -.2);
xformEnd($planeID, 28, 90, -.25, -.15);
xformEnd($planeID, 29, 0, -.1, -.2);
xformEnd($planeID, 30, 90, -.15, -.25);

xformEnd($planeID, 31, 0, -.2, 0);
xformEnd($planeID, 32, 90, -.15, .05);
xformEnd($planeID, 33, -90, -.15, .05);
xformEnd($planeID, 34, 0, -.1, 0);
xformEnd($planeID, 35, 0, -.1, -.1);
xformEnd($planeID, 36, 0, 0, -.1);
xformEnd($planeID, 37, 0, 0, 0);
xformEnd($planeID, 38, -90, -.05, -.05);
xformEnd($planeID, 39, 0, 0, 0);
xformEnd($planeID, 40, -90, .05, -.05);

groupSequence("group1", $planeID, 1, 20);
groupSequence("group2a", $planeID, 21, 30);
groupSequence("group2b", $planeID, 31, 40);

move -r 0 0.4 0 group2a;
```

```
	move -r 0 0.2 0 group2b;
}

proc xformEnd(string $planeID, int $stripNum, int $rot1, float $x1, float $y1)
{
	float $px = (float)(($stripNum-1)%20)/10 + 0.05;
	rotate -r -p $px 0 0 -os 0 0 $rot1 ($planeID + $stripNum + ".cv[0:3][6:8]");
	move -r $x1 $y1 0 ($planeID + $stripNum + ".cv[0:3][6:8]");
}

proc xformSequence(string $planeID, int $first, int $last, float $x, float $y, float $z)
{
	int $i;
	for($i=$first; $i<=$last; $i++)
		move -r $x $y $z ($planeID + $i + ".cv[0:3][5]");
}

proc groupSequence(string $name, string $root, int $a, int $b)
{
	group -n $name -r -em;
	xform -os -piv 0 0 0;

	int $i;
	for($i=$a; $i<=$b; $i++)
		parent ($root+$i) $name;
}

construct();
```

## G|03. createBox()

```
// Create box of specified length, width, and height

proc check(string $theName)
{
      string $theName;
      if (`window -exists $theName`)
            deleteUI $theName;
}

proc createBoxCmd()
{
      global float $whd[], $txyz[];
      string $bx[] = `polyCube -w $whd[0] -h $whd[1] -d $whd[2] -ax 0 1 0 -tx 1 -ch 1`;
      move $txyz[0] $txyz[1] $txyz[2] $bx[0];
      print(generateCompliment());
}

proc createBox()
{
      global float $whd[] = {1,1,1};
      global float $txyz[] = {0,0,0};
      int $winW=400, $winH=110;

      // Window setup
      string $winName = "Create_Box";
      check($winName);
      window -mxb 0 $winName;

      // Define components
      string $form = `formLayout - numberOfDivisions 100`;
      string $whdInputField = `floatFieldGrp -numberOfFields 3
            -v1 1 -v2 1 -v3 1
            -label "Width, Height, Depth"
            -cc "$whd = `floatFieldGrp -q -v whdInput`;" whdInput`;
      string $txyzInputField = `floatFieldGrp -numberOfFields 3
            -label "Translate X,Y,Z"
```

```
            -cc "$txyz = `floatFieldGrp -q -v txyzInput`;" txyzInput`;
      string $createButton = `button -label "Create"
            -command ("createBoxCmd();")`;

      int $margin = 5;
      formLayout -edit
            -attachForm $whdInputField "right" $margin
            -attachForm $txyzInputField "right" $margin
            -attachControl $txyzInputField "top" $margin $whdInputField
            -attachControl $createButton "top" $margin $txyzInputField
            -attachForm $createButton "right" $margin
            $form;

      window -e -widthHeight $winW $winH $winName;
      showWindow $winName;
}

proc string generateCompliment()
{
      string $comments[] = {
            "You are a scintillating wonder of piety!!!!!!!!!!!!!!!!!!!!!!!!!!!!",
            "You are a delightful model of victory!!!!!!!!!!!!!!!!!!!!!!!",
            "You are a jubilant personification of faith!!!!!!!!!!!!!!!!!!!!!!!",
            "You are a tireless instance of insight!!!!!!!!!!!!!!!!!!!!!!!",
            "You are clearly an attractive embodiment of courage!!!!!!!!!!!!!!!!!!!!!!!",
            "You are clearly a veritable realization of acumen!!!!!!!!!!!!!!!!!!!!!!!",
            "You effortlessly devise the most clever solution to every prob-
lem!!!!!!!!!!!!!!!!!!!!!!",
            "You are a fearless paradigm of a genius!!!!!!!!!!!!!!!!!!!!!!",
            "You are an outstanding veteran of piety!!!!!!!!!!!!!!!!!!!!!!!",
            "You are an extraordinary harbinger of wisdom!!!!!!!!!!!!!!!!!!!!!!!",
            "You are an undisputed instance of a friend!!!!!!!!!!!!!!!!!!!!!!"
            };

      return toupper($comments[(int)rand(size($comments))]);
}

createBox();
```

## G|04. createCleanSquare()

```
proc createCleanSquare()
{
	// Variables
	float $ccsw=1, $ccsh=1;
	int $winW=300, $winH=50;

	// Window setup
	string $winName = "MakeCurve_SQUARE";
	window -mxb 0 $winName;

	// Define components
	string $form = `formLayout - numberOfDivisions 100`;
	string $widthLabel = `text -label "X" -align "left"`;
	string $heightLabel = `text -label "Z" -align "left"`;
	string $widthField = `floatField -v $ccsw
		-cc "$ccsw = `floatField -q -v wfld`;" wfld`;
	string $heightField = `floatField -v $ccsh
		-cc "$ccsh = `floatField -q -v hfld`;" hfld`;
	string $goButton = `button -label "Create"
		-command ("createCleanSquareCmd($ccsw, $ccsh);")`;

	// Layout
	int $margin = 5;
	formLayout -edit
		-attachForm $widthLabel "left" $margin
		-attachControl $widthField "left" $margin $widthLabel
		-attachControl $heightLabel "left" $margin $widthField
		-attachControl $heightField "left" $margin $heightLabel
		-attachControl $goButton "left" $margin $heightField
		$form;

	showWindow $winName;
	window -e -widthHeight $winW $winH $winName;
}

proc createCleanSquareCmd(float $w, float $h)
```

```
{
	string $curve = `curve -d 1
		-p $w 0 $h
		-p $w 0 0
		-p 0 0 0
		-p 0 0 $h
		-k 0 -k 1 -k 2 -k 3`;

	closeCurve -rpo 1 $curve;
}

createCleanSquare();
```

## G|05. createCurveFromPolygonVerts()

```
proc createCurveFromPolygonCVs()
{
	string $selection[] = `ls -sl -fl`;

	vector $vec[];

	int $i;
	for($i=0; $i<size($selection); $i++)
	{
		float $pos[] = `pointPosition $selection[$i]`;
		$vec[$i] = <<$pos[0], $pos[1], $pos[2]>>;
	}

	print(createCurveFromPointData($vec));
}

proc string createCurveFromPointData(vector $p[])
{
	string $cmd = "curve -d 1 ";

	int $i;
	for($i=0; $i<size($p); $i++)
	{
		float $tmp[] = $p[$i];
		$cmd = $cmd +"-p "+ $tmp[0] +" "+ $tmp[1] +" "+ $tmp[2] +" ";
	}

	string $crv = `eval $cmd`;
	closeCurve -ch 1 -rpo 1 -p 0.1 $crv;
	select($crv);
	return($crv);
}
```

## G|06. createStairs()

```
proc generateStair()
{
	// INPUT
	float $rise = .475;
	float $run = .65;
	float $width = 3;
	int $nSteps = 6;

	// LOCAL
	int $i;
	int $nPoints = ($nSteps*2)+1;
	float $x, $y, $z;
	string $cmd = "curve -d 1 ";

	// add points to command to generate steps
	for($i=0; $i<$nPoints; $i++)
	{
		$x = 0;
		$y = $rise * (($i+1)/2);
		$z = $run *  (($i+0)/2);

		$cmd = $cmd +"-p "+ $x +" "+ $y +" "+ $z +" ";
	}

	// Close off the bottom
	$cmd = $cmd +"-p "+ $x +" "+ ($y-$rise) +" "+ $z +" ";
	$cmd = $cmd +"-p "+ 0 +" "+ 0 +" "+ $run +" ";

	// Add knot value flags to command
	for($i=0; $i<$nPoints+2; $i++)
		$cmd = $cmd +"-k "+ $i +" ";

	// Create curve
	string $curveName = `eval($cmd)`;
	closeCurve -ch 0 -ps 1 -rpo 1 -bb 0.5 -bki 0 -p 0.1 $curveName;
```

```
	nurbsToPolygonsPref -f 0 -pt 1 -pc 1;
	string $surfName[] = `planarSrf -ch 1 -d 3 -ko 0 -tol 0.00393701
				-rn 0 -po 1 -n "stair1" $curveName`;
	polyExtrudeFacet
		-ch 1 -kft 0
		-tx 0 -ty 0 -tz 0 -rx 0 -ry 0 -rz 0 -sx 1 -sy 1 -sz 1
		-ran 0 -divisions 1 -twist 0 -taper 1 -off 0 -w 0 -ws 0
		-ltz $width -ltx 0 -lty 0 -lrx 0 -lry 0 -lrz 0
		-lsx 1 -lsy 1 -lsz 1 -ldx 1 -ldy 0 -ldz 0
		-gx 0 -gy -1 -gz 0 -att 0 -mx 0 -my 0 -mz 0
		($surfName[0] + ".f[0]");

	delete $curveName;
	select -cl;
}

generateStair();
```

## G|07. createStairsFromCurves()

```
proc generateStairFromCurves()
{
	string $selection[] = `ls -sl`;

	float $height = 100;
	int $stepCount = 150;
	float $rise = $height/$stepCount;
	float $run = (float)1/$stepCount;

	float $xyz[], $y;
	string $cmd;

	rebuildCurve -ch 1 -rpo 1 -rt 0 -end 1 -kr 0
		-kcp 0 -kep 1 -kt 0 -s 0 -d 3 -tol 0.00393701
		$selection[0];
	rebuildCurve -ch 1 -rpo 1 -rt 0 -end 1 -kr 0
		-kcp 0 -kep 1 -kt 0 -s 0 -d 3 -tol 0.00393701
		$selection[1];

	$cmd = "curve -d 1 ";
	int $i;
	for($i=0; $i<$stepCount; $i++)
	{
		$xyz = `pointPosition -w ($selection[0] + ".u[" + $i*$run + "]")`;
		$y = $i * $rise;
		$cmd = $cmd +"-p "+ $xyz[0] +" "+ $y +" "+ $xyz[2] + " ";
		if($i != $stepCount-1)
			$cmd = $cmd +"-p "+ $xyz[0] +" "+ ($y+$rise) +" "+ $xyz[2] + " ";
	}
	for($i=0; $i<($stepCount*2)-1; $i++)
		$cmd = $cmd +"-k "+ $i +" ";
	string $curve1 = `eval($cmd)`;

	$cmd = "curve -d 1 ";
	for($i=0; $i<$stepCount; $i++)
	{
```

```
            $xyz = `pointPosition -w ($selection[1] + ".u[" + $i*$run + "]")`;
            $y = $i * $rise;
            $cmd = $cmd +"-p "+ $xyz[0] +" "+ $y +" "+ $xyz[2] + " ";
            if($i != $stepCount-1)
                  $cmd = $cmd +"-p "+ $xyz[0] +" "+ ($y+$rise) +" "+ $xyz[2] + " ";
      }
      for($i=0; $i<($stepCount*2)-1; $i++)
            $cmd = $cmd +"-k "+ $i +" ";
      string $curve2 = `eval($cmd)`;

      loft -ch 1 -u 1 -c 0 -ar 1 -d 3
            -ss 1 -rn 0 -po 0 -rsn true
            $curve1 $curve2;

}

generateStairFromCurves();
```

## G|08. extractIsoparms()

```
proc extractIsoparms()
{
	string $selectedSurfs[] = `selectedNodes`;

	int $i;
	for($i=0; $i<size($selectedSurfs); $i++)
	{
		string $rebuildName = (“rebuild” + $i);

		rebuildSurface -name $rebuildName
			-ch 1 -rpo 0 -rt 0 -end 1 -kr 0 -kcp 0
			-kc 1 -su 4 -du 3 -sv 7 -dv 3
			-tol 0.00393701 -fr 0  -dir 2
			$selectedSurfs[$i];

		duplicateCurve -name ($rebuildName + “isoparm1”) ($rebuildName + “.u[0]”);
		duplicateCurve -name ($rebuildName + “isoparm2”) ($rebuildName + “.u[.25]”);
		duplicateCurve -name ($rebuildName + “isoparm3”) ($rebuildName + “.u[.5]”);
		duplicateCurve -name ($rebuildName + “isoparm4”) ($rebuildName + “.u[.75]”);
		duplicateCurve -name ($rebuildName + “isoparm5”) ($rebuildName + “.u[1]”);

		group -name (“isoparmGroup” + $i)
			($rebuildName + “isoparm1”)
			($rebuildName + “isoparm2”)
			($rebuildName + “isoparm3”)
			($rebuildName + “isoparm4”)
			($rebuildName + “isoparm5”);

		delete $rebuildName;
	}
}

extractIsoparms();
```

## G|09. extrudePolyline()

```
proc test()
{
	string $selected[] = `ls -sl`;
	for($i=0; $i<size($selected); $i++)
	{
		planarSrf -n ($selected[$i] + "_" + $i) -ch 0 -d 1 -ko 0 -tol 0.01 -rn 0 -po 1
$selected[$i];
		polyExtrudeFacet -ch 1 -kft 0 -pvx 324.2387426 -pvy 358.7226447 -pvz 621.8901867
			-tx 0 -ty 0 -tz 0
			-rx 0 -ry 0 -rz 0
			-sx 1 -sy 1 -sz 1
			-ran 0 -divisions 1 -twist 0 -taper 1 -off 0
			-ltz -30 -ws 0 -ltx 0 -lty 0 -lrx 0 -lry 0 -lrz 0
			-lsx 1 -lsy 1 -lsz 1 -ldx 1 -ldy 0 -ldz 0 -w 0 -gx 0 -gy -1 -gz 0
			-att 0 -mx 0 -my 0 -mz 0 ($selected[$i]+"_"+$i+".f[0]");
	}
}

test();
```

## G|10. flattenCVs()

```
proc flattenCVs()
{
	string $selectedCVs[] = `ls -sl -fl`;

	int $i;
	for($i=0; $i<size($selectedCVs); $i++)
	{
		float $pos[] = `getAttr $selectedCVs[$i]`;
		setAttr $selectedCVs[$i] $pos[0] 0 $pos[2];
	}
}

flattenCVs();
```

## G|11. flattenNURBS()

```
// Flattens NURBS planes by moving CVs along $axis

proc check(string $theName)
{
      string $theName;
      if (`window -exists $theName`)
            deleteUI $theName;
}

proc flattenNURBSInterface()
{
      string $winName = "Flatten_NURBS";
      check($winName);
      window -widthHeight 210 50 $winName;

      string $form = `formLayout -numberOfDivisions 100`;
      string $axis = `optionMenu axisOption`;
            menuItem -label "Y";
            menuItem -label "X";
            menuItem -label "Z";
      string $val = `floatField -ann "Value" valFloat`;
      string $button = `button -label "Flatten"
            -command ("flattenNURBS(`optionMenu -q -v axisOption`, `floatField -q -v valFloat`);")`;

      int $margin = 5;
      formLayout -edit
            -attachForm $axis "left" $margin
            -attachControl $val "left" $margin $axis
            -attachControl $button "left" $margin $val
            $form;

      showWindow $winName;
}

proc flattenNURBS(string $axis, float $value)
{
```

```
	string $selection[] = `selectedNodes`;
	int $cvcnt[] = `getAttr ($selection[0] + ".spansUV")`;
	float $pos[];

	int $u, $v;
	for($u=0; $u<=$cvcnt[0]+2; $u++)
		for($v=0; $v<=$cvcnt[1]+2; $v++)
		{
			float $pos[] = `getAttr ($selection[0] + ".cv["+ $u +"]["+ $v +"]")`;
			float $newPos[] = createPositionArray($axis, $value, $pos);

			setAttr ($selection[0] + ".cv["+ $u +"]["+ $v +"]")
				$newPos[0] $newPos[1] $newPos[2];
		}
}

proc float[] createPositionArray(string $axis, float $value, float $pos[])
{
	if($axis == "x" || $axis == "X")
		return {$value, $pos[1], $pos[2]};
	else if($axis == "y" || $axis == "Y")
		return {$pos[0], $value, $pos[2]};
	else if($axis == "z" || $axis == "Z")
		return {$pos[0], $pos[1], $value};
	else
		return {0, 0, 0};
}

flattenNURBSInterface();
```

## G|12. insertIsoparmAt()

```
proc insertParms(float $loc)
{
	int $i;
	for($i=25; $i<=32; $i++)
	{
		insertKnotSurface -ch 1 -nk 1 -add 1 -ib 0 -rpo 1
			(“original_full_strip_rebuilt1detachedSurface” +$i +”.v[$loc]”);
	}
}

insertParms(0.401);
```

## G|13. makeLocatorsRenderable()

```
proc makeLocatorsRenderable()
{
	string $sel[] = `ls -sl`;
	float $size = 20;
	float $thickness = .5;

	int $i;
	for($i=0; $i<size($sel); $i++)
	{
		vector $xyz = `getAttr ($sel[$i] + ".translate")`;

		string $c1[] = `polyCube -w $thickness -h $thickness -d $size`;
		string $c2[] = `polyCube -w $thickness -h $size -d $thickness`;
		string $c3[] = `polyCube -w $size -h $thickness -d $thickness`;

		string $b1[] = `polyBoolOp -op 1 -ch 0 $c1[0] $c2[0]`;
		string $b2[] = `polyBoolOp -op 1 -ch 0 $b1[0] $c3[0]`;

		move ($xyz.x) ($xyz.y) ($xyz.z) $b2[0];
	}
}

makeLocatorsRenderable();
```

## G|14. moveCurvePivot()

```
proc moveCurvePivot()
{
	string $obj[] = `ls -sl -fl`;
	string $objType = `objectType $obj[0]`;

	// Selection Check
	if(size($obj)>1)
		error "Select *ONE* CV on a NURBS curve or surface!";

	else if($objType == "nurbsCurve" || $objType == "nurbsSurface")
	{
		string $shape[] = `listRelatives -ap $obj[0]`;
		string $nurbs[] = `listTransforms $shape[0]`;
		string $crv = $nurbs[0];

		float $xyz[] = `pointPosition -w $obj`;
		move -a $xyz[0] $xyz[1] $xyz[2] ($crv + ".scalePivot") ($crv + ".rotatePivot");

		select $crv;
	}

	else
		error "moveCurvePivot: Selection error!";
}

moveCurvePivot();
```

## G|15. relativeMove()

```
proc relativeMove(float $x, float $y, float $z, string $units)
{
      float $factor;

      if($units == "inches")
            $factor = 0.08333333;
      else if($units == "feet")
            $factor = 1.0;
      else if($units == "meters")
            $factor = 3.2808399;

      move -r ($x*$factor) ($y*$factor) ($z*$factor);
}

proc callMoveWindow()
{
      float $xyz[3] = {0, 0, 0};

      string $winName = "Relative_Move";
      window $winName;

      string $form = `formLayout -numberOfDivisions 100`;
      string $inputField = `floatFieldGrp -numberOfFields 3
            -cc "$xyz = `floatFieldGrp -q -v input`;" input`;
      string $inchOKbutton = `button -label "move inches"
            -command ("relativeMove($xyz[0], $xyz[1], $xyz[2], \"inches\");")`;
      string $feetOKbutton = `button -label "move feet"
            -command ("relativeMove($xyz[0], $xyz[1], $xyz[2], \"feet\");")`;
      string $meterOKbutton = `button -label "move meters"
            -command ("relativeMove($xyz[0], $xyz[1], $xyz[2], \"meters\");")`;
      string $warning = `text -label "*** Assumes 1.0 Maya units = 1'-0\""`;

      int $margin = 5;
      formLayout -edit
            -attachForm $inputField "left" $margin
            -attachControl $inchOKbutton "top" $margin $inputField
```

```
            -attachControl $feetOKbutton "left" $margin $inchOKbutton
            -attachControl $feetOKbutton "top" $margin $inputField
            -attachControl $meterOKbutton "left" $margin $feetOKbutton
            -attachControl $meterOKbutton "top" $margin $inputField
            -attachForm $warning "bottom" $margin
            $form;

      window -e -widthHeight 350 110 $winName;
      showWindow $winName;
}
```

## G|16. showHiddenChildren()

```
proc showHiddenChildren()
{
      string $sel[] = `ls -sl`;

      int $i;
      for($i=0; $i<size($sel); $i++)
      {
            showHidden -b $sel[$i];
      }
}

showHiddenChildren();
```

## G|17. thickenSurfaces()

```
// Thickens selected NURBS surface.

proc thickenSelectedSurfs()
{
	// $n => number to start naming things with
	// $thickness

	float $thickness = 0.4;
	int $n = 1;
	string $surface[] = `selectedNodes`;

	int $i;
	for($i=0; $i<size($surface); $i++)
	{
		int $uvcnt[] = `getAttr loftedSurface2attachedSurfaceShape1.spansUV`;

		offsetSurface  -ch on -m 0 -d $thickness -n ("surf"+($n+$i)+"_1") $surface[$i];
		loft -ch 1 -u 1 -c 0 -ar 1 -d 3 -ss 1 -rn 0 -po 0 -n ("lft"+($n+$i)+"_1")
			($surface[$i]+".u[0]") (("surf"+($n+$i)+"_1")+".u[0]");
		loft -ch 1 -u 1 -c 0 -ar 1 -d 3 -ss 1 -rn 0 -po 0 -n ("lft"+($n+$i)+"_2")
			($surface[$i]+".u["+$uvcnt[0]+"]") (("surf"+($n+$i)+"_1")+".u["+$uvcnt[0]+"]");
		loft -ch 1 -u 1 -c 0 -ar 1 -d 3 -ss 1 -rn 0 -po 0 -n ("lft"+($n+$i)+"_3")
			($surface[$i]+".v[0]") (("surf"+($n+$i)+"_1")+".v[0]");
		loft -ch 1 -u 1 -c 0 -ar 1 -d 3 -ss 1 -rn 0 -po 0 -n ("lft"+($n+$i)+"_4")
			($surface[$i]+".v["+$uvcnt[1]+"]") (("surf"+($n+$i)+"_1")+".v["+$uvcnt[1]+"]");

		group -n ("group"+($n+$i))
			$surface[$i]
			("surf"+($n+$i)+"_1")
			("lft"+($n+$i)+"_1")
			("lft"+($n+$i)+"_2")
			("lft"+($n+$i)+"_3")
			("lft"+($n+$i)+"_4");
	}
}
```

## ii. Automation

### A|01. booleanMulti()

```
proc booleanMulti()
{
	// Selection Breakdown
	string $selection[] = `ls -sl -fl`;
	string $knife = $selection[size($selection)-1];

	// Variables inside loop
	int $i;
	string $firstParent;
	string $newKnife[], $boolResult[];

	for($i=0; $i<size($selection)-1; $i++)
	{
		// Remember parent of each object
		$firstParent = firstParentOf($selection[$i]);

		// Make a copy of the knife
		$newKnife = `duplicate -rr $knife`;

		// Subtract and store name, parent with original parent
		$boolResult = `polyBoolOp -op 2 -ch 0 $selection[$i] $newKnife[0]`;
		parent $boolResult $firstParent;
	}
}

booleanMulti();
```

## A|02. createMullions()

```
proc createMullions()
{
	// Input Variables
	int $nMullions = 80;
	int $startAt = 20;
	int $endAt = 70;
	string $profileName = "mullion_profile";

	// Local Variables
	float $interval = 1/(float)$nMullions;

	// Selection breakdown
	string $selection[] = `ls -sl -fl`;

	// Variables set inside loop
	string $obj;
	string $grp;

	int $i, $j;
	for($i=0; $i<size($selection); $i++)
	{
		$obj = $selection[$i];
		$grp = `group -em -n "mullions1"`;

		for($j=$startAt; $j<$endAt; $j++)
		{
			string $mull[] = `extrude -ch true -rn false -po 0 -et 2 -ucp 1
				-fpt 1 -upn 1 -rotation 0 -scale 1 -rsp 1
				$profileName ($obj + ".v[" + ($j*$interval) + "]")`;
			parent $mull[0] $grp;
		}
	}

	select -cl;
}

createMullions();
```

## A|03. extrudeProfileOnSelectedCurves()

```
proc extrudeProfileOnSelectedCurves()
{
	string $profile = "structureDiags5x5";
	string $sel[] = `ls -sl -fl`;

	int $i;
	for($i=0; $i<size($sel); $i++)
	{
		nurbsToPolygonsPref -f 0 -pt 1 -pc 1;

		polyCloseBorder `extrude -ch 0 -rn false -po 1 -et 2 -ucp 1 -fpt 1
			-upn 1 -rotation 0 -scale 1 -rsp 1
			$profile $sel[$i]`;
	}
}
```

## A|04. extrudeProfileOnSelectedIsoparms()

```
proc extrudeProfileOnSelectedIsoparms()
{
	string $profile = "truss_span_profile1";
	string $sel[] = `ls -sl -fl`;

	int $i;
	for($i=0; $i<size($sel); $i++)
	{
		extrude -ch true -rn false -po 0 -et 2 -ucp 1 -fpt 1
			-upn 1 -rotation 0 -scale 1 -rsp 1
			$profile $sel[$i];
	}
}
```

## A|05. fixIsoparms()

```
proc test()
{
	string $s[] = `ls -sl`;

	int $i;
	for($i=0; $i<size($s); $i++)
	{
		insertKnotSurface -ch 1 -nk 1 -add 1 -ib 0 -d 0  -rpo 1
			-p 0.42992 -p 0.45995 -p 0.47998 $s[$i];
	}
}
```

## A|06. generateLinesFromStrips()

```
proc generateLinesFromStrips()
{
	string $sel[] = `ls -sl -fl`;

	float $startAt = 0.375;
	float $inc = 0.0625;
	int $count = 9;

	int $i, $j;
	for($i=0; $i<size($sel); $i++)
	{
		for($j=0; $j<$count; $j++)
		{
			float $v = $startAt + $j*$inc;
			duplicateCurve -ch 1 -rn 0 -local 0 ($sel[$i] +".v["+ $v +"]");
		}
	}
}
```

## A|07. makeColumns()

```
proc test()
{
	string $selected[];
	string $obj;
	float $epxyz[];

	$selected = `ls -sl`;
	$obj = $selected[0];
	$epxyz1 = `pointPosition ($obj + ".cv[0][3]")`;
	$epxyz2 = `pointPosition ($obj + ".cv[4][3]")`;
	$epxyz3 = `pointPosition ($obj + ".cv[14][3]")`;
	$epxyz4 = `pointPosition ($obj + ".cv[18][3]")`;

	string $c1 = `curve -d 1 -k 0 -k 1
		-p $epxyz1[0] $epxyz1[1] $epxyz1[2]
		-p $epxyz1[0] 0 $epxyz1[2]`;
	string $c2 = `curve -d 1 -k 0 -k 1
		-p $epxyz2[0] $epxyz2[1] $epxyz2[2]
		-p $epxyz2[0] 0 $epxyz2[2]`;
	string $c3 = `curve -d 1 -k 0 -k 1
		-p $epxyz3[0] $epxyz3[1] $epxyz3[2]
		-p $epxyz3[0] 0 $epxyz3[2]`;
	string $c4 = `curve -d 1 -k 0 -k 1
		-p $epxyz4[0] $epxyz4[1] $epxyz4[2]
		-p $epxyz4[0] 0 $epxyz4[2]`;

	extrude -ch true -rn false -po 0 -et 2 -ucp 1
		-fpt 1 -upn 1 -rotation 0 -scale 1 -rsp 1
		"column_profile_01"
		$c1 ;
	extrude -ch true -rn false -po 0 -et 2 -ucp 1
		-fpt 1 -upn 1 -rotation 0 -scale 1 -rsp 1
		"column_profile_01"
		$c2 ;
	extrude -ch true -rn false -po 0 -et 2 -ucp 1
		-fpt 1 -upn 1 -rotation 0 -scale 1 -rsp 1
		"column_profile_01"
```

```
            $c3 ;
      extrude -ch true -rn false -po 0 -et 2 -ucp 1
            -fpt 1 -upn 1 -rotation 0 -scale 1 -rsp 1
            "column_profile_01"
            $c4 ;

      select -cl;
}
```

## A|08. sawToothLines()

```
proc sawToothLines()
{
      string $sel[] = `ls -sl -fl`;

      int $i;
      for($i=0; $i<size($sel)-1; $i++)
      {
            float $p1[], $p2[];
            float $p1a[] = `pointPosition -w ($sel[$i] +".cv[0]")`;
            float $p1b[] = `pointPosition -w ($sel[$i] +".cv[3]")`;
            float $p2a[] = `pointPosition -w ($sel[$i+1] +".cv[0]")`;
            float $p2b[] = `pointPosition -w ($sel[$i+1] +".cv[3]")`;

            if($p1a[1] > $p1b[1]) $p1 = $p1a; else $p1 = $p1b;
            if($p2a[1] < $p2b[1]) $p2 = $p2a; else $p2 = $p2b;

            string $curve = `curve -d 1
                  -p $p1[0] $p1[1] $p1[2]
                  -p $p2[0] $p2[1] $p2[2]
                  -k 0 -k 1`;

      }
}
```

## R|01. attachToSurfaceWithGeoConstraint()

```
proc test()
{
	int $cvcnt[] = `getAttr baseShape.spansUV`;
	float $epxyz[];

	int $u, $v;
	for($u=0; $u<=$cvcnt[0]+2; $u++)
	{
		for($v=0; $v<=$cvcnt[1]+2; $v++)
		{
			int $unitNum = ($u*($cvcnt[0]+3) + $v) + 1;

			$epxyz = `pointPosition base.cv[$u][$v]`;
			duplicate -n ("unit"+ $unitNum) unit0;

			setAttr ("unit"+ $unitNum +".translateX") $epxyz[0];
			setAttr ("unit"+ $unitNum +".translateY") $epxyz[1];
			setAttr ("unit"+ $unitNum +".translateZ") $epxyz[2];

			geometryConstraint -weight 1 base ("unit"+ $unitNum);
		}
	}
}
```

## R|02. createLinearNodeConnection()

```
proc createLinearNodeConnection()
{
	string $profileCurve = "link_profile";
	string $curveGrp = "links|curves";
	string $surfGrp = "links|surfs";

	string $sel[] = `ls -sl`;

	string $curve = `curve -d 1 -p 0 0 0 -p 1 1 1 -k 0 -k 1 -n ("Link" +"_"+ $sel[0] +"_"+
$sel[1])`;
	string $curveShape[] = `listRelatives -s $curve`;

	connectAttr -f ($sel[0] + ".translate") ($curveShape[0] + ".controlPoints[0]");
	connectAttr -f ($sel[1] + ".translate") ($curveShape[0] + ".controlPoints[1]");

	string $surf[];
	if($profileCurve != "")
		$surf = `extrude -ch true -rn false -po 0 -et 2
			-ucp 1 -fpt 1 -upn 1 -rotation 0 -scale 1 -rsp 1
			$profileCurve $curve`;

	if($curveGrp != "")
		parent $curve $curveGrp;
	if($profileCurve != "")
		parent $surf[0] $surfGrp;
}

createLinearNodeConnection();
```

## R|03. driveCVs()

```
proc driveCVs()
{
	string $sel[] = `ls -sl -fl`;
	string $cv1 = $sel[1];
	string $cv2 = $sel[2];
	string $cv3 = $sel[3];
	string $cv4 = $sel[4];

	string $surfShape[] = `listRelatives -ap $sel[0]`;
	string $cp1[] = `listAttr $sel[1]`;
	string $cp2[] = `listAttr $sel[2]`;
	string $cp3[] = `listAttr $sel[3]`;
	string $cp4[] = `listAttr $sel[4]`;

	connectAttr -f ($sel[0] + ".translateY") ($surfShape[0] +"."+ $cp1[0] +".yValue");
	connectAttr -f ($sel[0] + ".translateY") ($surfShape[0] +"."+ $cp2[0] +".yValue");
	connectAttr -f ($sel[0] + ".translateY") ($surfShape[0] +"."+ $cp3[0] +".yValue");
	connectAttr -f ($sel[0] + ".translateY") ($surfShape[0] +"."+ $cp4[0] +".yValue");
}

driveCVs();
```

## Expression: `nodeHeightByProximity()`

```
proc float nodeHeightByProximity(vector $thisXYZ)
{
	string $locs[] = `listRelatives -c "nodes"`;
	int $i;
	for($i=0; $i<size($locs); $i++)
	{
		float $x = `getAttr ($locs[$i] +".translateX")`;
		float $y = `getAttr ($locs[$i] +".translateY")`;
		float $z = `getAttr ($locs[$i] +".translateZ")`;
		float $inf[] = `getAttr ($locs[$i] +".influence")`;
		vector $locXYZ = <<$x, $y, $z>>;
		vector $dist = abs($thisXYZ - $locXYZ);
		if($dist.x < ($inf[0]*5) && $dist.z < ($inf[2]*5))
			return $inf[1]*10;
	}
	return 0;
}

proc checkNodes()
{
	string $nodes[] = `listRelatives -c "driverGroup"`;
	int $i;
	for($i=0; $i<size($nodes); $i++)
	{
		float $x = `getAttr ($nodes[$i] +".translateX")`;
		float $y = `getAttr ($nodes[$i] +".translateY")`;
		float $z = `getAttr ($nodes[$i] +".translateZ")`;

		setAttr ($nodes[$i] +".translateY") (nodeHeightByProximity(<<$x, $y, $z>>));
	}
}

checkNodes();
```

## iv. Behavioral

### B|01. `createPoolRoof()`

```
proc createRoof()
{
	vector $poolExtents1 = `getAttr poolExtents1.translate`;
	vector $poolExtents2 = `getAttr poolExtents2.translate`;
	vector $deckW = `getAttr deckW.translate`;
	vector $deckE = `getAttr deckE.translate`;
	vector $roofHeightLocator = `getAttr roofHeight.translate`;
	vector $trussDepthLocator = `getAttr trussDepth.translate`;

	global int $nSpans, $nTrussSpansPerColumn, $nWebSpans;
	global float $spanFluctuation, $steelTensileStressPSI;
	float $roofLoadPSF;

		$roofLoadPSF = 30.0;
		$steelTensileStressPSI = 20000;
		$nSpans = 10;
		$nTrussSpansPerColumn = 3;
		$spanFluctuation = 6.0;
		$nWebSpans = 8;

	global float $roofHeight, $trussDepth, $poolWidth, $poolDepth;
	float $poolSurfaceArea, $deckWidthW, $deckWidthE;
```

```
	$roofHeight = ($roofHeightLocator.y);
	$trussDepth = ($roofHeight - $trussDepthLocator.y);
	$poolWidth = $poolExtents2.x - $poolExtents1.x;
	$poolDepth = $poolExtents2.z - $poolExtents1.z;

	$poolSurfaceArea = $poolWidth * $poolDepth;
	$deckWidthW = $deckW.x - $poolExtents2.x;
	$deckWidthE = $poolExtents1.x - $deckE.x;

global float $columnSpacing, $trussSpacing, $loadPerTruss, $maxTrussMoment;
global int $nTruss;
float $columnLocations[];
float $loadPerColumnPair, $maxTrussMoment;

	$columnSpacing = $poolDepth / $nSpans;
	$nTruss = $nSpans * $nTrussSpansPerColumn + 1;
	$trussSpacing = $columnSpacing / $nTrussSpansPerColumn;
	$loadPerColumnPair = $poolSurfaceArea*$roofLoadPSF / $nSpans;
	$loadPerTruss = $loadPerColumnPair / $nTrussSpansPerColumn;
	$maxTrussMoment = getMaxMoment($nWebSpans, $poolWidth, $loadPerTruss);

// ========================================================================================== //
print( "\n"
	+ "____LOADS_____________________________" + "\n"
	+ "Roof load (PSF)            " + $roofLoadPSF + "\n"
	+ "Steel Tensile Stress (PSI) " + $steelTensileStressPSI + "\n"
	+ "Load/column pair (lb)      " + $loadPerColumnPair + "\n"
	+ "Load/truss (lb)            " + $loadPerTruss + "\n"
	+ "Max truss moment (lb-ft)   " + $maxTrussMoment + "\n"
	+ "____DIMENSIONS_______________________" + "\n"
	+ "Roof height max (ft)       " + $roofHeight + "\n"
	+ "Pool Width (ft)            " + $poolWidth + "\n"
	+ "Pool Depth (ft)            " + $poolDepth + "\n"
	+ "____MEMBERS__________________________" + "\n"
	+ "Fluctuation (ft)           " + $spanFluctuation + "\n"
	+ "Column spans               " + $nSpans + "\n"
	+ "Column spacing (ft)        " + $columnSpacing + "\n"
	+ "Truss' per Column span     " + $nTrussSpansPerColumn + "\n"
	+ "Truss count                " + $nTruss + "\n"
```

```
            + "Truss spacing (ft)          " + $trussSpacing + "\n"
            + "Truss depth (ft)            " + $trussDepth + "\n"
            );

      // ========================================================================================== //
      for($i=0; $i<=$nSpans; $i++)
      {
            $columnLocations[$i] = $i*$columnSpacing;
            createColumn(("columnA"+$i), 0, $deckE.x, getColumnHeight($i), $columnLocations[$i]);
            createColumn(("columnB"+$i), 0, 0, getColumnHeight($i), $columnLocations[$i]);
            createColumn(("columnC"+$i), 0, $poolWidth, getColumnHeight($i), $columnLocations[$i]);
            createColumn(("columnD"+$i), 0, $deckW.x, getColumnHeight($i), $columnLocations[$i]);
      }

      createBeam("BeamA", $deckE.x);
      createBeam("BeamB", $poolExtents1.x);
      createBeam("BeamC", $poolExtents2.x);
      createBeam("BeamD", $deckW.x);

      sizeTrussMembers();
      for($i=0; $i<$nTruss; $i++)
            createTruss(("truss" + $i), getTrussHeight($i), ($i * $trussSpacing));

      group -n "Columns" `getAllObjectsStartingWith("column")`;
      group -n "Beam" `getAllObjectsStartingWith("Beam")`;
      group -n "Trusses" `getAllObjectsStartingWith("trussGroup_")`;

      select -cl;
}

proc sizeTrussMembers()
{
      global float $maxTrussMoment, $trussDepth, $steelTensileStressPSI;
      float $areaIN2 = $maxTrussMoment/$trussDepth/$steelTensileStressPSI;
      float $radiusIN = sqrt($areaIN2 / 3.1415927);

//    setAttr makeChordProfile.radius (0.1 * $radiusIN); // doesn't work if expression is already de-
fined ..
//    setAttr makeWebProfile.radius (0.1 * ($radiusIN*.5));
```

```
}

proc createColumn(string $name, float $load, float $x, float $y, float $z)
{
      // !!! size according to $load !!!

      global float $roofHeight;
      polyCube -n $name -w 1.0 -h $y -d 2.0;
      move $x ($y/2) $z $name;
}

proc createBeam(string $name, float $x)
{
      global float $roofHeight, $spanFluctuation, $columnSpacing, $poolDepth;
      global int $nSpans;

      for($i=0; $i<$nSpans; $i++)
            buildBeamMember(($name+"_seg"+$i), <<$x, getColumnHeight($i), $i*$columnSpacing>>, <<$x,
getColumnHeight($i+1), ($i+1)*$columnSpacing>>);
}

proc float getColumnHeight(int $i)
{
      global float $roofHeight, $spanFluctuation;
      if($i%2 == 0)           return $roofHeight;
      else              return ($roofHeight - $spanFluctuation);
}

proc float getTrussHeight(int $n)
{
      global float $roofHeight, $trussDepth, $spanFluctuation;
      global int $nTruss, $nTrussSpansPerColumn;
      float $stepSize = $spanFluctuation/$nTrussSpansPerColumn;
      return ($roofHeight - $stepSize * abs(($n+$nTrussSpansPerColumn)%($nTrussSpansPerColumn*2)-
$nTrussSpansPerColumn));
}

proc buildBeamMember(string $name, vector $p1, vector $p2)
{
```

```
	curve -d 1  -p ($p1.x) ($p1.y) ($p1.z)
		    -p ($p2.x) ($p2.y) ($p2.z)
		-k 0 -k 1 -n $name;

	extrude -ch true -rn false -po 0 -et 2 -ucp 1 -fpt 1 -upn 1
		-rotation 0 -scale 1 -rsp 1
		-n ($name + "_loft")
		"beamProfile" $name;
}

proc float getMaxMoment(float $nWebSpans, float $poolWidth, float $loadPerTruss)
{
	float $maxMoment = 0;
	float $loadIncrement = ($loadPerTruss/2) / $nWebSpans;
	for($i=0; $i<$nWebSpans/2; $i++)
		$maxMoment += (($poolWidth/$nWebSpans) * ($loadPerTruss - ($i*$loadIncrement)));
	return $maxMoment;
}

proc createTruss(string $name, float $y, float $z)
{
	global float $poolWidth, $maxTrussMoment, $trussDepth;
	global int $nWebSpans;
	float $webSpan = $poolWidth/$nWebSpans;

	buildTrussMember(($name+"_upperChord"), <<0, $y, $z>>, <<$poolWidth, $y, $z>>);

	vector $this = <<0.0, $y, $z>>;

	int $c = 2;
	for($i=1; $i<=$nWebSpans/2; $i++)
	{
		vector $next = <<($i*$webSpan), ($y-($trussDepth*($c-1)/$c)), $z>>;
		$c += $c;

		// LOWER CHORD
		buildTrussMember(($name+"_web_lowerChord_"+$i+"a"), <<$this.x, $this.y, $this.z>>,
<<$next.x, $next.y, $next.z>>);
		buildTrussMember(($name+"_web_lowerChord_"+$i+"b"), <<($poolWidth-($this.x)), $this.y,
```

```
$this.z>>, <<($poolWidth-($next.x)), $next.y, $next.z>>);

            // VERTICALS
            buildTrussMember(($name+"_web_vertical_"+$i+"a"), <<$next.x, $y, $this.z>>, <<$next.x,
$next.y, $next.z>>);
            buildTrussMember(($name+"_web_vertical_"+$i+"b"), <<($poolWidth-($next.x)), $y, $this.z>>,
<<($poolWidth-($next.x)), $next.y, $next.z>>);

            // DIAGONALS
            buildTrussMember(($name+"_web_diagonal_"+$i+"a"), <<(($i*$webSpan)-$webSpan/2), $y, $this.
z>>, <<$next.x, $next.y, $next.z>>);
            buildTrussMember(($name+"_web_diagonal_"+$i+"b"), <<$this.x, $this.y, $this.z>>, <<(($i-
1)*$webSpan)+$webSpan/2), $y, $this.z>>);
            buildTrussMember(($name+"_web_diagonal_"+$i+"c"), <<($poolWidth - (($i*$webSpan)-$web-
Span/2)), $y, $this.z>>, <<($poolWidth - $next.x), $next.y, $next.z>>);
            buildTrussMember(($name+"_web_diagonal_"+$i+"d"), <<($poolWidth - $this.x), $this.y,
$this.z>>, <<($poolWidth - (($i-1)*$webSpan)+$webSpan/2)), $y, $this.z>>);

            $this = <<($next.x), ($next.y), ($next.z)>>;
      }

      group -n ("trussGroup_"+$name) `getAllObjectsStartingWith($name)`;
}

proc buildTrussMember(string $name, vector $p1, vector $p2)
{
      string $profile;
      if(gmatch($name, "*Chord*"))
            $profile = "chordProfile";
      else
            $profile = "webProfile";

      curve -d 1  -p ($p1.x) ($p1.y) ($p1.z)
                  -p ($p2.x) ($p2.y) ($p2.z)
            -k 0 -k 1 -n $name;

      extrude -ch true -rn false -po 0 -et 2 -ucp 1 -fpt 1 -upn 1
            -rotation 0 -scale 1 -rsp 1
            -n ($name + "_loft")
```

```
            $profile $name;
}

proc string[] getAllObjectsStartingWith(string $prefix)
{
      string $selected[] = (`ls -type "transform" ($prefix + "*")`);
      return $selected;
}

createRoof();
```

## Expression: `updateTrussSystem()`

```
/*proc adjustTruss(string $name, float $y, float $z)
{
      dummyPoint.translateY = trussDepth.translateY;

      global float $roofHeight, $poolWidth, $maxTrussMoment;
      global int $nWebSpans;
      float $webSpan = $poolWidth/$nWebSpans;
      float $trussDepth = $roofHeight - (`getAttr trussDepth.translateY`);

      vector $this = <<0.0, $y, $z>>;
      vector $next;
      int $c = 2;

      for($i=1; $i<=$nWebSpans/2; $i++)
      {
            $next = <<($i*$webSpan), ($y - ($trussDepth*($c-1)/$c)), $z>>;
            $c += $c;

            // LOWER CHORD
            move -a ($this.x) ($this.y) ($this.z) ($name+"_web_lowerChord_"+$i+"a.cv[0]");
```

```
            move -a ($next.x) ($next.y) ($next.z) ($name+"_web_lowerChord_"+$i+"a.cv[1]");
            move -a ($poolWidth-($this.x)) ($this.y) ($this.z) ($name+"_web_lowerChord_"+$i+"b.
cv[0]");
            move -a ($poolWidth-($next.x)) ($next.y) ($next.z) ($name+"_web_lowerChord_"+$i+"b.
cv[1]");

            // VERTICALS
            move -a ($next.x) $y ($this.z) ($name+"_web_vertical_"+$i+"a.cv[0]");
            move -a ($next.x) ($next.y) ($next.z) ($name+"_web_vertical_"+$i+"a.cv[1]");
            move -a ($poolWidth-($next.x)) $y ($this.z) ($name+"_web_vertical_"+$i+"b.cv[0]");
            move -a ($poolWidth-($next.x)) ($next.y) ($next.z) ($name+"_web_vertical_"+$i+"b.cv[1]");

            // DIAGONALS
            move -a (($i*$webSpan)-$webSpan/2) $y ($this.z) ($name+"_web_diagonal_"+$i+"a.cv[0]");
            move -a ($next.x) ($next.y) ($next.z) ($name+"_web_diagonal_"+$i+"a.cv[1]");
            move -a ($this.x) ($this.y) ($this.z) ($name+"_web_diagonal_"+$i+"b.cv[0]");
            move -a ((($i-1)*$webSpan)+$webSpan/2) $y ($this.z) ($name+"_web_diagonal_"+$i+"b.cv[1]");
            move -a ($poolWidth - (($i*$webSpan)-$webSpan/2)) $y ($this.z)  ($name+"_web_diagonal_
"+$i+"c.cv[0]");
            move -a ($poolWidth - $next.x) ($next.y) ($next.z) ($name+"_web_diagonal_"+$i+"c.cv[1]");
            move -a ($poolWidth - $this.x) ($this.y) ($this.z) ($name+"_web_diagonal_"+$i+"d.cv[0]");
            move -a ($poolWidth - ((($i-1)*$webSpan)+$webSpan/2)) $y ($this.z) ($name+"_web_diagonal_
"+$i+"d.cv[1]");

            $this = <<($next.x), ($next.y), ($next.z)>>;
      }
}

proc adjustMembers()
{
      global float $roofHeight, $maxTrussMoment, $steelTensileStressPSI;
      float $trussDepth = ($roofHeight - (`getAttr trussDepth.translateY`));
      float $areaIN2 = $maxTrussMoment/$trussDepth/$steelTensileStressPSI;
      float $radiusIN = sqrt($areaIN2 / 3.1415927);

      makeChordProfile.radius = (0.2 * $radiusIN);           // should be 0.1
      makeWebProfile.radius = (0.2 * ($radiusIN*.5));
}
```

```
proc float getTrussHeight(int $n)
{
      global float $roofHeight, $trussDepth, $spanFluctuation;
      global int $nTruss, $nTrussSpansPerColumn;
      float $stepSize = $spanFluctuation/$nTrussSpansPerColumn;
      return ($roofHeight - $stepSize * abs(($n+$nTrussSpansPerColumn)%($nTrussSpansPerColumn*2)-
$nTrussSpansPerColumn));
}

proc runAdjust()
{
      global int $nTruss;
      global float $trussSpacing;

      for($i=0; $i<$nTruss; $i++)
            adjustTruss(("truss"+$i), getTrussHeight($i), ($trussSpacing * $i));

      adjustMembers();
}

runAdjust();*/
```

## B|02. panelizeSurface()

```
proc panelizeSurface()
{
	// Input Variables
	int $ch = 0;
	int $makeBeams = 0;
	int $makeHoles = 1;
	int $makeHydro = 0;

	int $nSlabs = 160;
//	int $startAt = 45;
//	int $endAt = 46;
	int $startAt = 40;
	int $endAt = 60;

	float $slabThickness = 0.5;		// in feet [0.2]
	float $slabGap = 0.0;		// feet [0.0]
	float $webDepth = 0.4;		// feet [0.5]
	float $webOffset = 0.4;		// relative [0.15]
	float $minHoleOffsetFromWeb = 0.25;	// factor (relative) [0.3]
	float $holeDistanceFromBeam = 0.75;	// feet [1.0]
	float $lightNodeEffect = 15.0;	// feet [15.0]
/*
	float $slabThickness = 0.40;		// in feet [0.2]
	float $slabGap = 0.0;		// feet [0.0]
	float $webDepth = 0.5;		// feet [0.5]
	float $webOffset = 0.15;		// relative [0.15]
	float $minHoleOffsetFromWeb = 0.25;		// factor (relative) [0.3]
	float $holeDistanceFromBeam = 0.75;	// feet [1.0]
	float $lightNodeEffect = 15.0;	// feet [15.0]
*/
	// Local Variables
	float $slabInc = 1/(float)$nSlabs;
	float $holeSizeRange = 0.499999 - $minHoleOffsetFromWeb;

	// Selection breakdown
	string $selection[] = `ls -sl -fl`;
```

```
	string $lightNodes[] = `listRelatives -c lightNodes`;

	// Variables inside loop
	string $panel[], $surfHistory[], $beams[], $beam1[], $beam2[], $holeKnife[];
	string $slabGrp, $slabSurface, $groupName, $beamGrp1, $beamGrp2;
	string $slabSurface;
	float $distToLightNode;
	int $i, $j;

	for($i=0; $i<size($selection); $i++)
	{
		$slabSurface = $selection[$i];

		if($makeBeams)
		{
			$surfHistory = `listHistory -pdo 1 -f 1 -bf $selection[$i]`;
			string $extrudedBeams[] = reduceToObjectsOfType($surfHistory, "extrude");
			$beam1 = `listConnections -sh 1 $extrudedBeams[0]`;
			$beam2 = `listConnections -sh 1 $extrudedBeams[1]`;

			$beamGrp1 = `group -em`;
			$beamGrp2 = `group -em`;
		}

		$groupName = `group -em -n "tSlabGroup1"`;
		$slabGrp = `group -em`;

		for($j=$startAt; $j<$endAt; $j++)
		{
			// SLAB
			$panel = createSlab($slabSurface, $j*$slabInc, ($j+1)*$slabInc, $slabThickness,
$ch);
			// WEBS
			createWebs($panel[0], $webOffset, $webDepth, $slabGap, $ch);
			// HOLES
			if($makeHoles) $panel = createHoles($panel, $slabSurface, $j, $slabInc, $lightNodes,
				$lightNodeEffect, $holeSizeRange, $minHoleOffsetFromWeb,
				$holeDistanceFromBeam, $ch);
			// HYDRO
```

```
                  if($makeHydro) createHydro();

                  parent $panel[0] $slabGrp;

                  // BEAMS
                  if($makeBeams)
                  {
                        $beams = createBeams($beam1[2], $beam2[2], (($j)*$slabInc)-1,
(($j+1)*$slabInc)-1, $ch);
                        parent $beams[0] $beamGrp1;
                        parent $beams[1] $beamGrp2;
                  }
            }

            if($makeBeams) parent $beamGrp1 $beamGrp2 $slabGrp $groupName;
            else parent $slabGrp $groupName;

            select -cl;
      }
}

proc string[] createSlab(string $surf, float $isoA, float $isoB, float $thickness, int $ch)
{
      // Face
      nurbsToPolygonsPref -f 0 -pt 1 -pc 1;
      string $result[] = `loft -ch 0 -u 1 -c 0 -ar 1 -d 1 -ss 1 -rn 0 -po 1 -rsn true
            ($surf + ".v["+ $isoA +"]") ($surf + ".v["+ $isoB +"]")`;

      // Thicken
      polyExtrudeFacet -ch $ch -kft 0
            -tx 0 -ty 0 -tz 0 -rx 0 -ry 0 -rz 0 -sx 1 -sy 1 -sz 1
            -ran 0 -divisions 1 -twist 0 -taper 1 -off 0
            -ltz $thickness -ws 0 -ltx 0 -lty 0 -lrx 0 -lry 0 -lrz 0 -lsx 1 -lsy 1 -lsz 1
            -ldx 1 -ldy 0 -ldz 0 -w 0 -gx 0 -gy -1 -gz 0 -att 0 -mx 0 -my 0 -mz 0
            ($result[0] + ".f[0]");

      return $result;
}
```

```
proc createWebs(string $surf, float $offset, float $depth, float $gap, int $ch)
{
	// Split faces and extrude webs
	polySplit -ch $ch -s 1 -ep 1  0.5 -ep 3 0.5 $surf;
	polySplit -ch $ch -s 1 -ep 12 0.6 -ep 3 0.4 $surf;
	polySplit -ch $ch -s 1 -ep 1 0.4 -ep 13 0.6 $surf;
	polySplit -ch $ch -s 1 -ep 15 $offset -ep 3 (1-$offset) $surf;
	polySplit -ch $ch -s 1 -ep 1 (1-$offset) -ep 19 $offset $surf;

	polyExtrudeFacet -ch $ch -kft 0
		-tx 0 -ty 0 -tz 0 -rx 0 -ry 0 -rz 0 -sx 1 -sy 1 -sz 1
		-ran 0 -divisions 1 -twist 0 -taper 1 -off 0 -ws 0
		-ltz $depth -ltx 0 -lty 0 -lrx 0 -lry 0 -lrz 0 -lsx 1 -lsy 1 -lsz 1 -ldx 1 -ldy 0 -ldz 0
		-w 0 -gx 0 -gy -1 -gz 0 -att 0 -mx 0 -my 0 -mz 0
		($surf + ".f[8:9]");

	// Tuck in edges of slab and cleanup
	move -r 0 0 (-1*$gap/2) ($surf + ".vtx[0:1]") ($surf + ".vtx[4:5]");
	move -r 0 0 ($gap/2)    ($surf + ".vtx[2:3]") ($surf + ".vtx[6:7]");
	delete ($surf + ".e[14]");
}

proc string[] createHoles(string $panel[], string $surf, int $j, float $slabInc, string $nodes[],
	float $effect, float $range, float $offWeb, float $offBeam, int $ch)
{
	float $distToNode = getDistToNearestLightNode($panel[0], $nodes);
	if($distToNode < $effect)
	{
		float $holeFactor = ($range * $distToNode/$effect) + $offWeb;

		$holeKnife = `loft -ch $ch -u 1 -c 0 -ar 1 -d 1 -ss 1 -rn 0 -po 1 -rsn true
			($surf + ".v["+ (($j*$slabInc)+($slabInc*$holeFactor)) +"]")
			($surf + ".v["+ ((($j+1)*$slabInc)-($slabInc*$holeFactor)) +"]")`;

		move -r -os -wd ($offBeam) 0 0
			($holeKnife[0] + ".vtx[1:2]");
		move -r -os -wd (-1*$offBeam) 0 0
			($holeKnife[0]+".vtx[0]") ($holeKnife[0]+".vtx[3]");
```

```
//          move -r -os -wd 0.5 .5 0
//                ($holeKnife[0] + ".vtx[1:2]");
//          move -r -os -wd 0.5 -0.5 0
//                ($holeKnife[0]+".vtx[0]") ($holeKnife[0]+".vtx[3]");

            move -0.05 -0.05 0 $holeKnife[0];
            polyExtrudeFacet -ch $ch -kft 0
                  -tx 0 -ty 0 -tz 0 -rx 0 -ry 0 -rz 0 -sx 1 -sy 1 -sz 1
                  -ran 0 -divisions 1 -twist 0 -taper 1 -off 0
                  -ltz (10+0.1) -ltx 0 -lty 0 -lrx 0 -lry 0 -lrz 0
                  -lsx 1 -lsy 1 -lsz 1 -ldx 1 -ldy 0 -ldz 0
                  -w 0 -ws 0 -gx 0 -gy -1 -gz 0 -att 0 -mx 0 -my 0 -mz 0
                  ($holeKnife[0] + ".f[0]");
            $panel = `polyBoolOp -op 2 -ch $ch $panel[0] $holeKnife`;
      }

      return $panel;
}

proc string[] createBeams(string $bA, string $bB, float $isoA, float $isoB, int $ch)
{
      // Loft shape
      string $sectA[] = `loft -ch $ch -u 1 -c 0 -ar 1 -d 3 -ss 1 -rn 0 -po 1 -rsn true
            ($bA + ".v["+ $isoA +"]") ($bA + ".v["+ $isoB +"]")`;
      polyCloseBorder -ch $ch $sectA[0];
      string $sectB[] = `loft -ch $ch -u 1 -c 0 -ar 1 -d 3 -ss 1 -rn 0 -po 1 -rsn true
            ($bB + ".v["+ $isoA +"]") ($bB + ".v["+ $isoB +"]")`;
      polyCloseBorder -ch $ch $sectB[0];

      string $beams[] = {$sectA[0], $sectB[0]};
      return $beams;
}

proc createHydro()
{
//first time:
//create circle

//second time
```

```
//duplicate first circle
// move duplicate to position
// loft from c1 to c2
// delete both circles

// new curve = dup isoparm[1]
// move curve to position
// loft isoparm[1] to curve
// delete curve
//attachSurface  -ch 1 -rpo 1 -kmk 1 -m 1 -bb 0.5 -bki 0 -p 0.1 "loftedSurface13attachedSurface1"
"loftedSurface14" ;
//rebuildSurface -ch 1 -rpo 1 -rt 0 -end 1 -kr 0 -kcp 1 -kc 0 -su 0 -du 0 -sv 0 -dv 0 -tol 0.00393701
-fr 0  -dir 2 "loftedSurface13attachedSurface1attachedSurface1";

// if in proximity of supply node
// create supply line
}

proc float getDistToNearestLightNode(string $obj, string $nodes[])
{
      float $n, $min = 100000000;
      int $i;
      for($i=0; $i<size($nodes); $i++)
      {
            float $dist = measureDistance(`objectCenter $nodes[$i]`, `objectCenter $obj`);
            if($dist < $min)
                  $min = $dist;
      }
      return $min;
}

proc float measureDistance(float $A[], float $B[])
{
      return sqrt (($B[0]-$A[0])*($B[0]-$A[0]) + ($B[1]-$A[1])*($B[1]-$A[1]) + ($B[2]-$A[2])*($B[2]-
$A[2]));
}

proc string[] reduceToObjectsOfType(string $objects[], string $type)
{
```

```
	string $list[];
	int $i;
	for($i=0; $i<size($objects); $i++)
		if(`objectType $objects[$i]` == $type)
			$list[size($list)] = $objects[$i];
	return $list;
}

panelizeSurface();
```

## B|03. snapshots()

```
proc makeSnapshots()
{
	cycleCheck -e off;

	string $baseUnit = "pCube";
	int $startNumber = 1;
	int $snapshotCount = 20;

	string $selectedCurves[] = `selectedNodes`;
	string $command;

	int $i;
	for($i=0; $i<size($selectedCurves); $i++)
	{
		duplicate -n ($baseUnit + ($startNumber + $i)) ($baseUnit + "0");

		pathAnimation -fractionMode true -follow true -followAxis z -upAxis y
			-worldUpType "vector" -worldUpVector 0 1 0 -inverseUp false
			-inverseFront false -bank true -bankScale 100 -bankThreshold 90
			-startTimeU 1 -endTimeU $snapshotCount ("pCube" + ($i+$startNumber))
$selectedCurves[$i];

		snapshot -increment 1 -constructionHistory 1
			-startTime 0 -endTime 80 -update animCurve;
	}

	group -n "baseUnits" -em;
	group -n "snapshots" -em;
	for($i=1; $i<=size($selectedCurves); $i++)
	{
		parent ($baseUnit + $i) "baseUnits";
		parent ("snapshot" + $i + "Group") "snapshots";
	}

	select -cl;
}
```