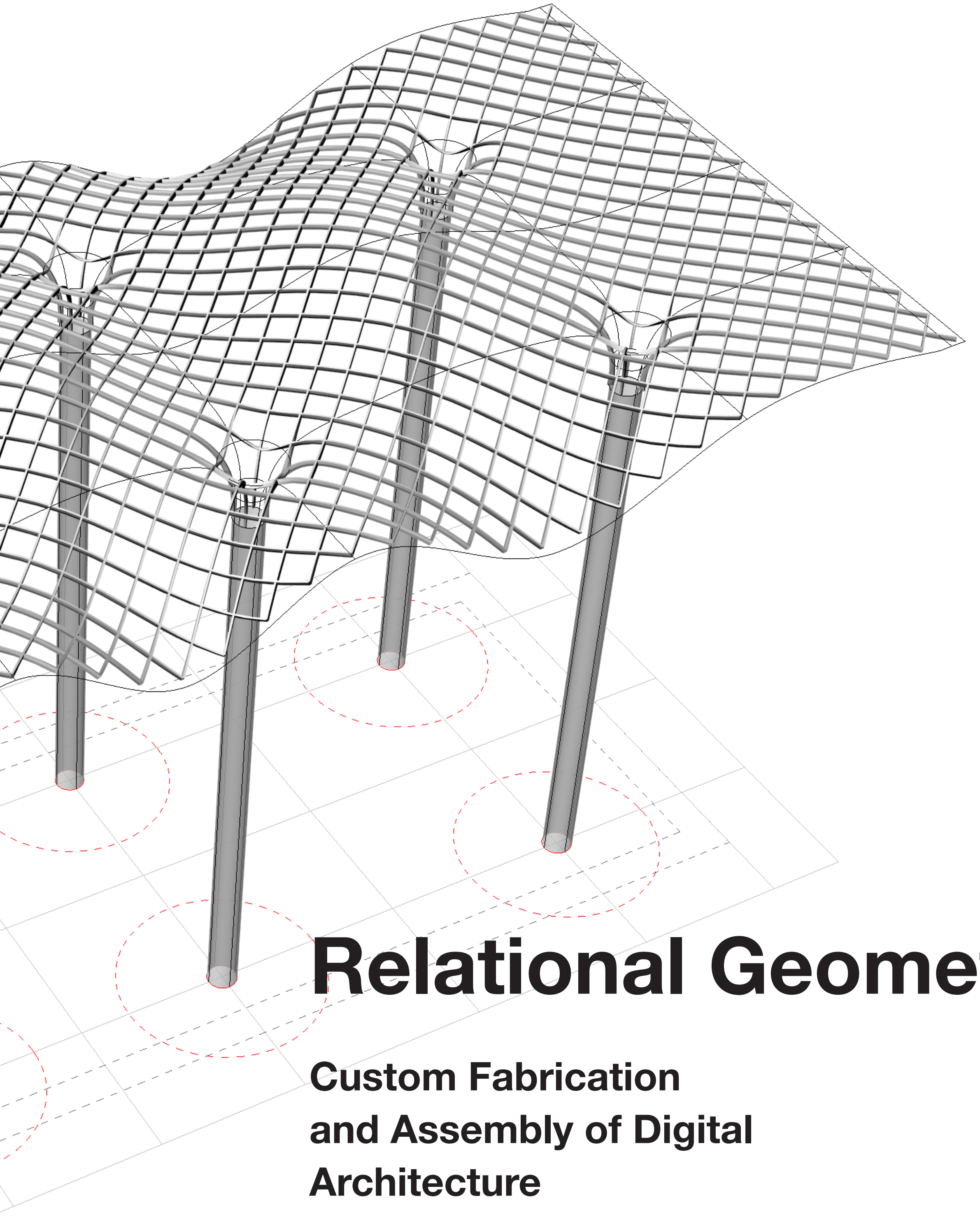


Relational Geometries

Custom Fabrication and Assembly of Digital Architecture

Robert Beson
Byera Hadley Traveling Scholarship 2010

Acknowledgements

I would like to take this space to thank the people and organisations that made the travel and research possible. Firstly, I am grateful to the Byera Hadley Travelling Scholarship and the New South Wales Architects Registration Board. Without both of which, none of this would have been possible. The same is true of Fabian Scheurer and the office of Design to Production, Switzerland. Extremely gracious with his time and knowledge, Fabian enabled an invaluable learning experience. I owe a debt of gratitude to Sandra Kajia-O'Grady and Anthony Burke, who provided the support of the University of Technology, Sydney. I would also like thank Damien Butler, Sarah Benton, and Gabriele Ulacco who provided their support and recommendations throughout the project.

Contents

Relational Geometries

Custom Fabrication and Assembly of Digital Architecture

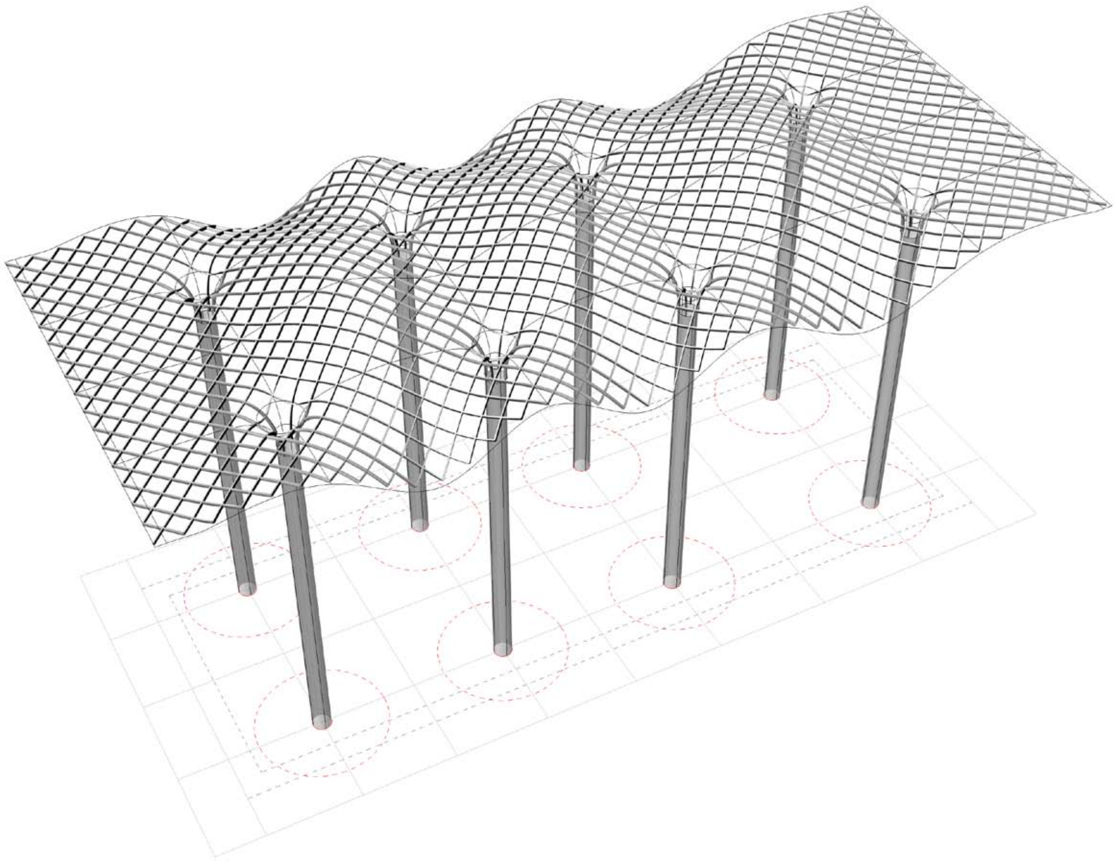
Digital design techniques are commonly taught in schools of Architecture, and at UTS we are invested in computer controlled fabrication equipment, such as laser cutters, milling machines and rapid prototypers. However, while there are many architects using digital tools, and fabricators using computer controlled machinery, there remains a gap between the designers of the projects and the fabricators who build them. This research aims to fill that gap and will provide architects with the expertise to better realise complex architectural designs. For large scale, complex projects our procurement methods are changing, and although architects are experts at traditional methods of construction, an inadequate knowledge of the digital fabrication and assembly process prevents the delivery of these projects.

With the use of digital design techniques, contemporary architectural projects are increasing in size and complexity. A number of current projects, on which I have worked, both in the academy and the profession, are characterised by an extreme number of prefabricated components each slightly varied. Delivering complex architecture of this sort requires a combination of specialist parametric software as well as custom-built, scripted tools. One of the best firms in the world at realising complex geometries is Design to Production. Based in Zurich, Switzerland, they have a strong history of working with architects around the world to help them rationalise, fabricate and assemble complex architectural projects.

Through an eight-week internship working with Design to Production, the investigation researches and develops techniques and planning processes for the custom fabrication and assembly of complex architectural projects. The focus is not to "illustrate architecture within the computer, but to extract architecture again from the computer in order to build artefacts, which cannot be realised with conventional practice."

The method for this project involved travelling to Zurich to work with Design to Production for eight-weeks. The internship was supplemented with excursions to a number of their recently completed projects in Switzerland and Germany including the EPFL Learning Center , the Hungerburg Funicular Station , and the Mercedes Benz Museum . While current publications on these buildings are helpful, working directly with Design to Production has enabled me to develop a toolset and production methodology immediately applicable to the education and production of architecture.

Gerode Project

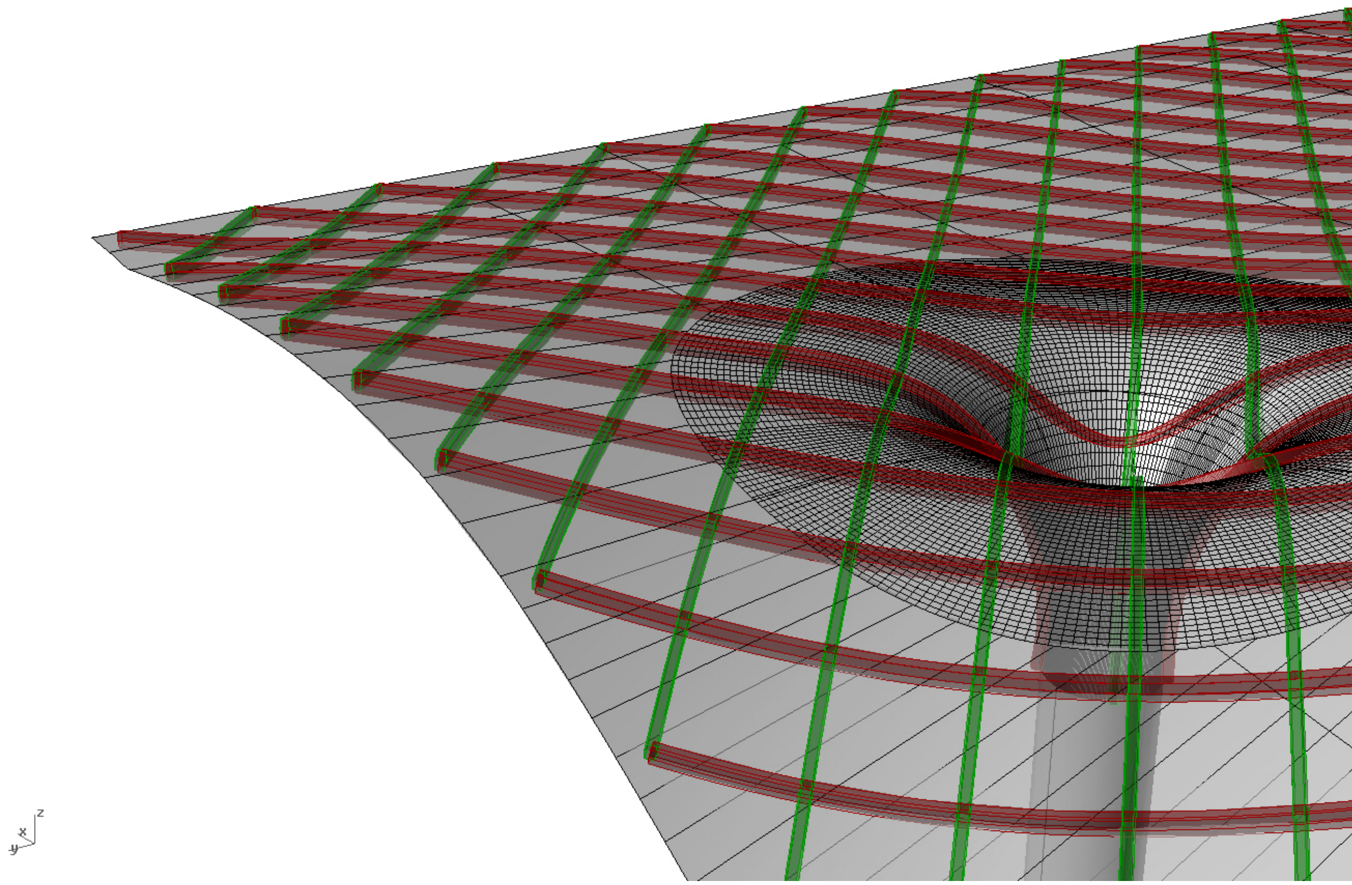


The Gerode project is a roof structure covering an old Romanesque church in the Swiss countryside.

It is to be support on eight columns at a height of 20m from the ground. It is a research project in collaboration with the Swiss timber company Blumer Lehmann. The research focuses on innovative techniques for fabricating and assembling doubly curved timber structures.

The project when finished will showcase the technical ability and collaboration between Blumer Lehmann and Design to Production.

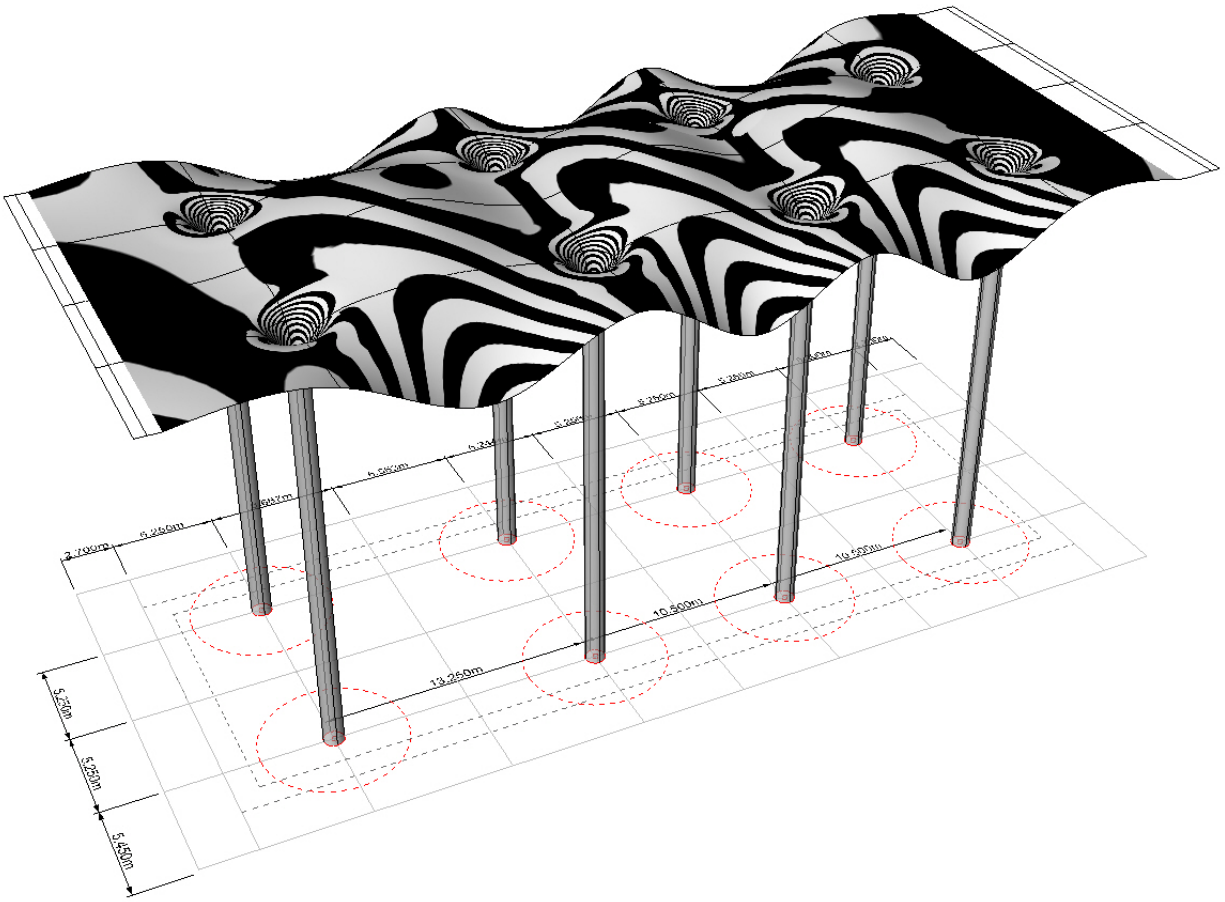
My role consisted of designing a number of surface



Reference surface and resulting girders

options for the roof and deploying scripts which resolve the surface into girders for construction. This is the first step in moving from a definition surface to the construction model.

The definition surface acts as the critical interface between the design, engineering, and construction teams. It is usually the object of most concern to the architect, and that which is engineered. It is important that accurate and well-modeled surfaces are used as primary and secondary structural systems as well as facade are all designed in relation to this single reference. Problems with the reference or its coordination between consultants will reverberate throughout the construction.

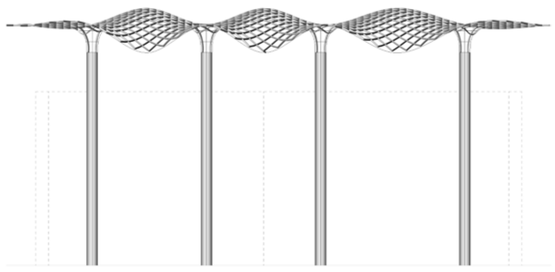
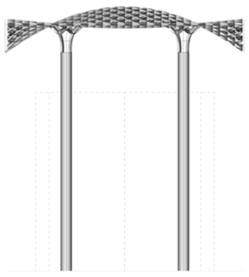
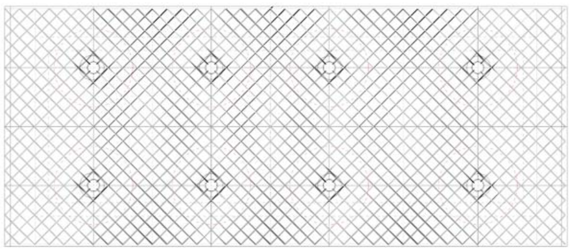
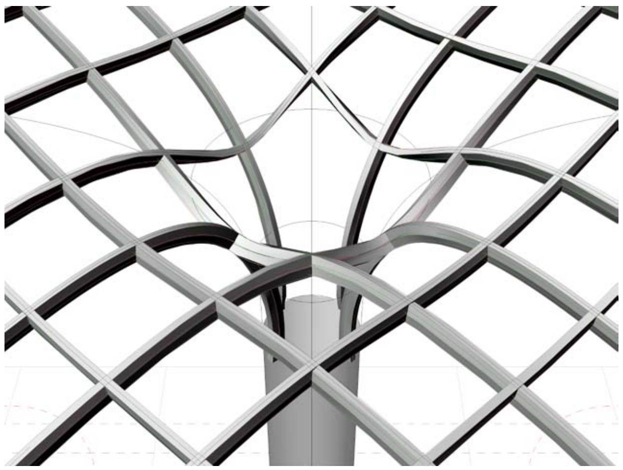


Surface curvature analysis

Strategies

Accurately modeling the reference surface is of the utmost importance. The reference surface must be composed of a single NURBS surface, not a poly-surface.

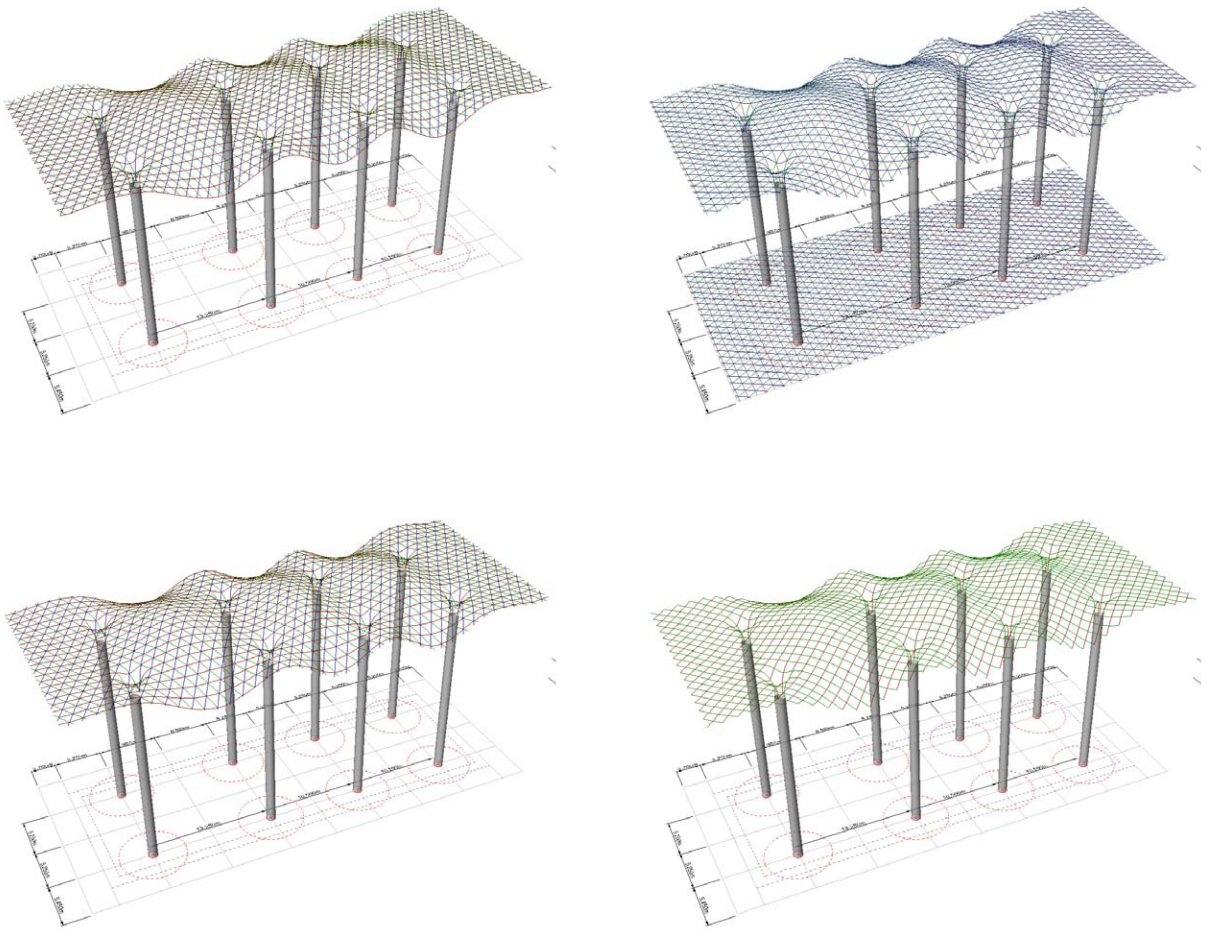
As a NURBS surface is a defined mathematically, it allows for accurate extraction of sub-elements such as curves and points. In this case, we will be creating a series of girder curves based on the surface. A facetted, poly-surface would result in facetted curves, resulting in facetted timber beams. We need to maintain the resolution

of our surfaces and curves as far as possible as we move information downstream.

Further, its curvature must be continuous. Any slight variations in the surface curvature will disrupt the creation of the girders.

Keeping in mind the structural action and construction needs, we must take care to orient the grid with the stems. As the roof surface funnels down to the stems, it is easy to introduce discontinuities between the attachment of the girders and the stems.


Options for timber girder construction

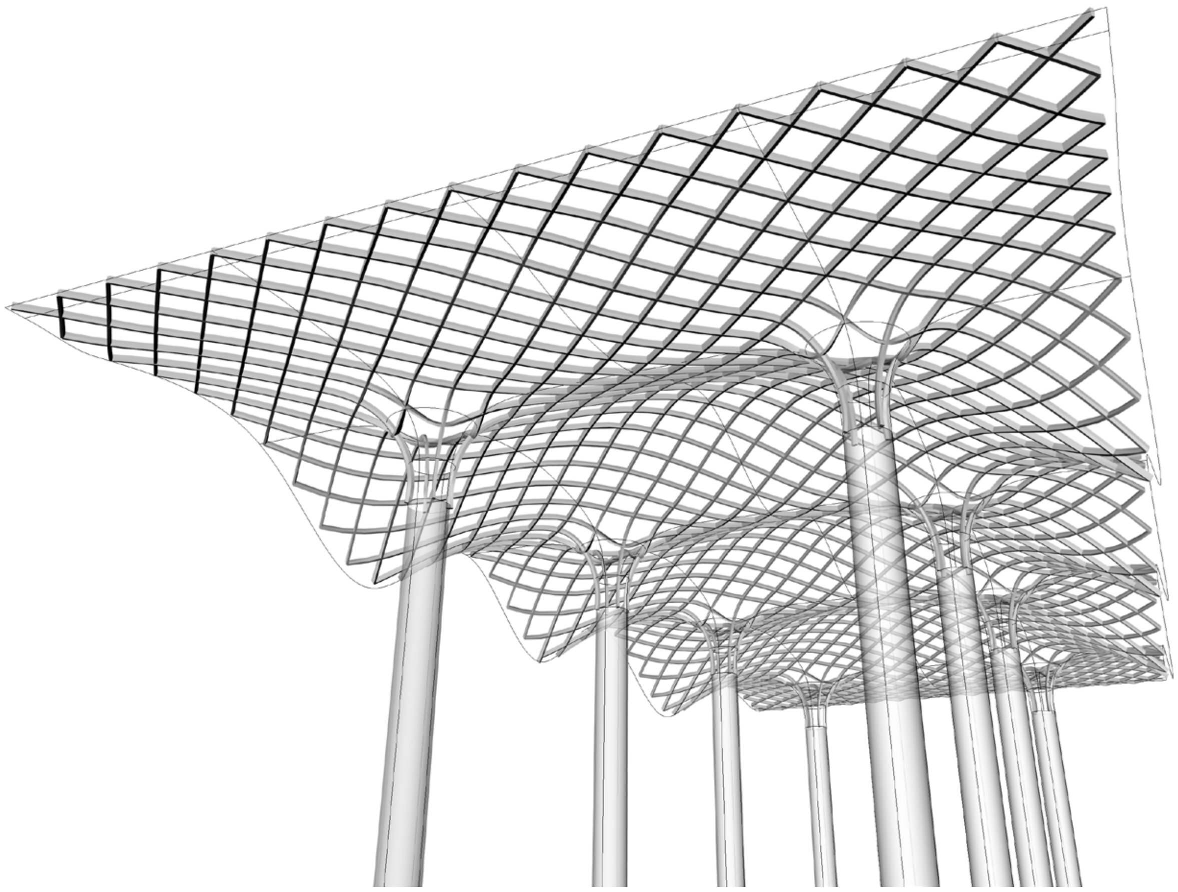
Process

A grid is defined for the structure. The grid will be used later to project the girders onto a reference surface.

The intersections of the grid mark the centers for the location of the stems.

The reference surface is modeled according to design sensibility, the boundary of the roof, the grid, and the stems. It is used, in combination with the girder axes, to produce the girders.

The simplest way to create the girders on the surface is



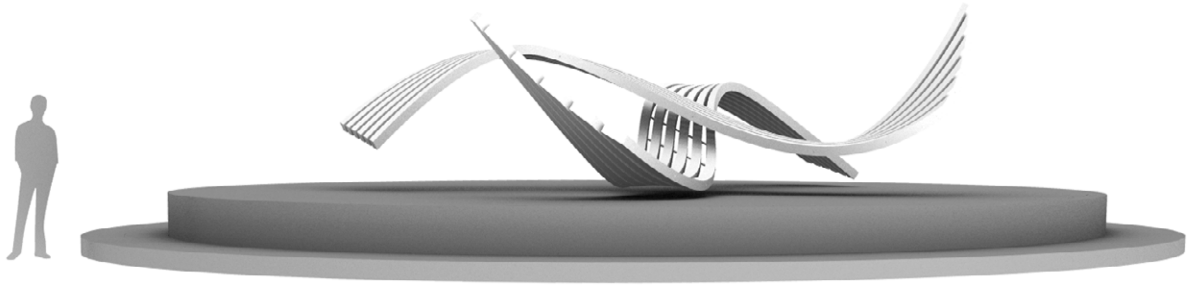
first to define them as a grid on the ground plane, which is then projected onto the reference surface. The projected curves will be used to create the girder volumes.

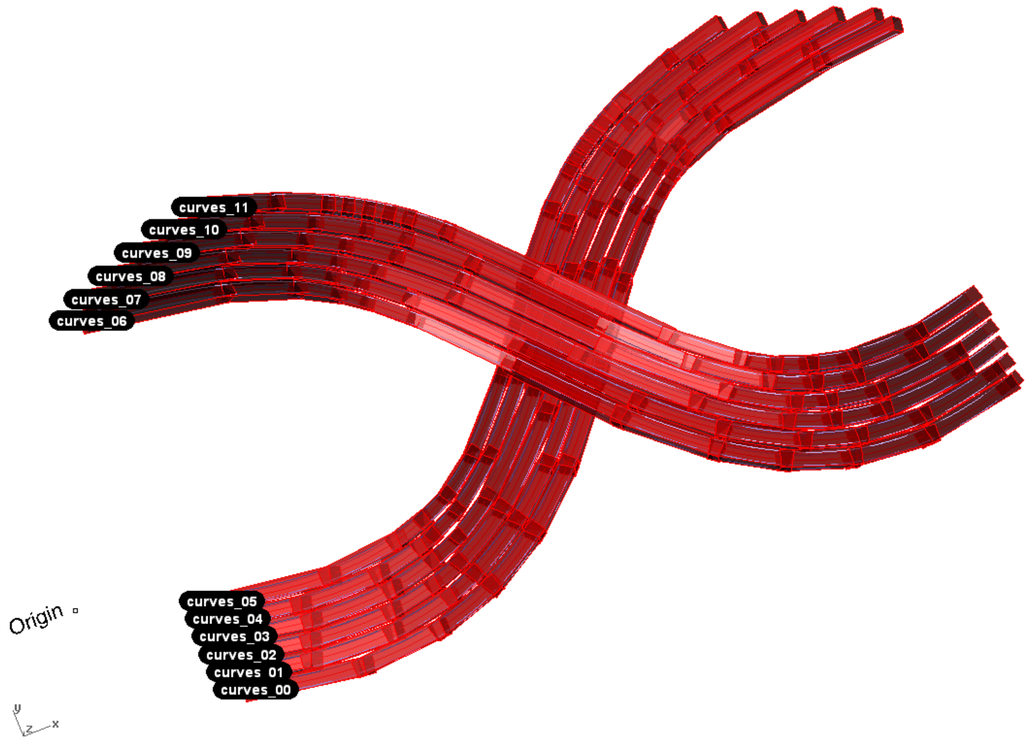
At this stage, a number of options are tested according to construction, structural and aesthetic constraints. With the chosen timber construction process, each timber girder can only intersect with one other girder at a time.

The project paused at this stage while evaluating the structure against the two chosen designs (shown).

The next steps in the process of taking the design to production would be to define the joint locations, segmentisation of the girders, and export of segments for CNC fabrication.

Kreisel Waldstatt



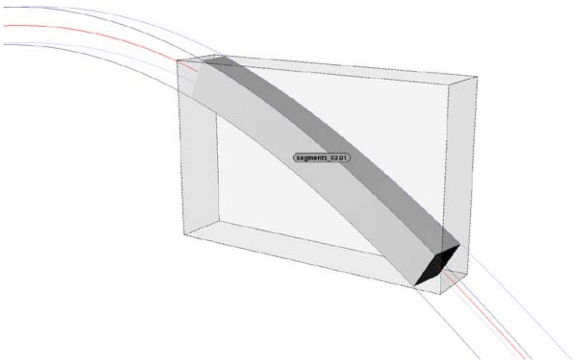


Structure organised into optimum blank sizes for milling

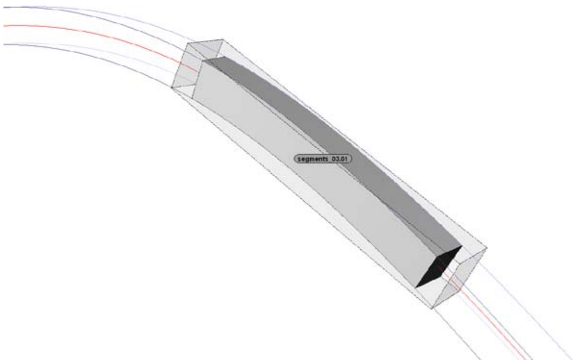
The Kreisel Waldstatt is a public-art project at a round-about in Switzerland.

In terms of the overall research agenda, the round-about project picks up the fabrication process where we left it with the last project. The doubly-curved beams are defined and need to be organised for a digital fabrication process.

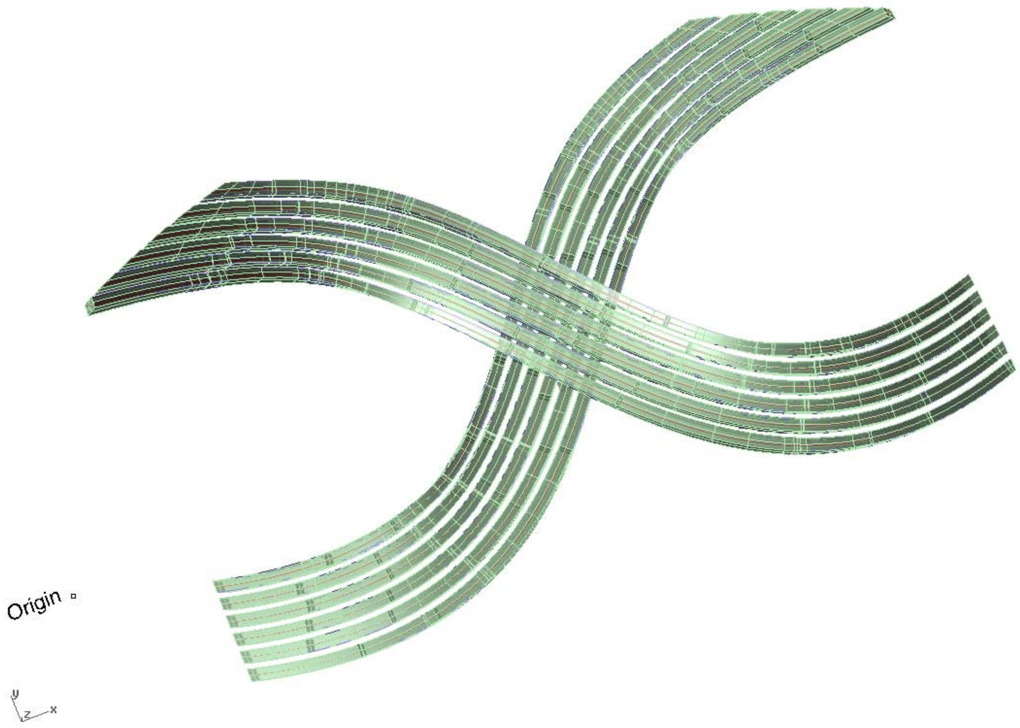
It consists of a series of doubly-curved timber beams each approximately 12m long. The structure is to be built from standard timber blanks measuring 12m x 250mm x 250mm.



Segment sitting within unoptimised blank



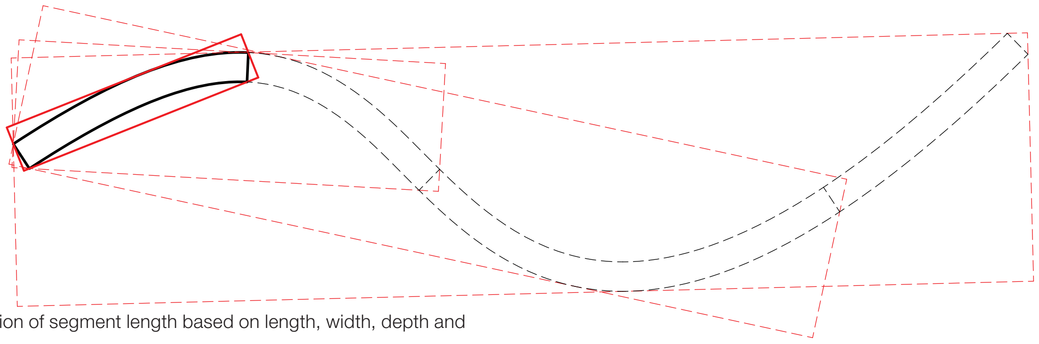
Optimum blank based on minimum bounding box



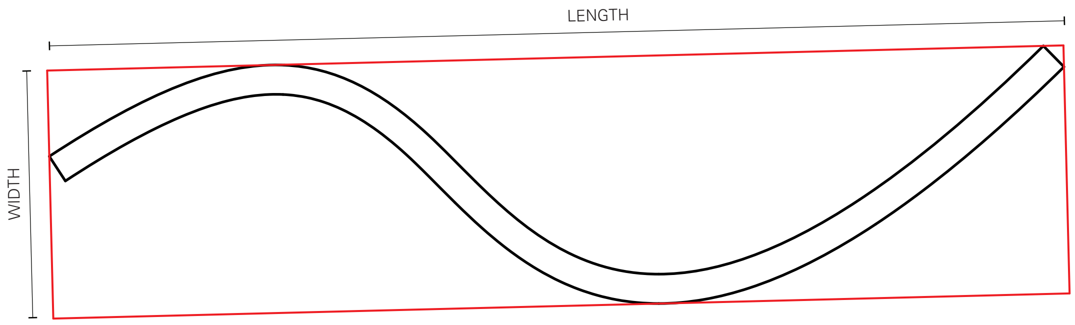
Finished digital model constructed from segments to be milled

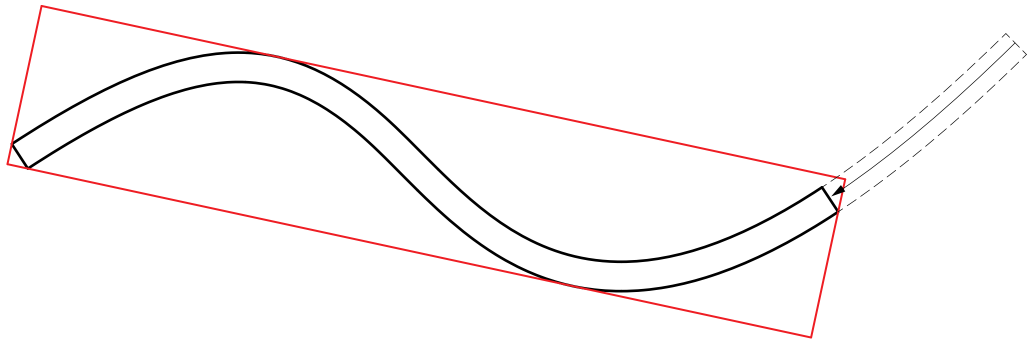
Custom curved blanks could also be used, however, they cost approximately ten times the price of the standard timber. The beams will have to be segmented to fit within the standard blanks, which will then be machined back into the doubly-curved segment on a 5-axis timber milling machine.

The goal for this project was to find the minimum blank that could accommodate each segment individually, while still fulfilling all of the fabrication, structural, and construction constrains. to optimize the blanks for minimum wastage and then prepare and export the fabrication data.



Iterative reduction of segment length based on length, width, depth and fiiber direction

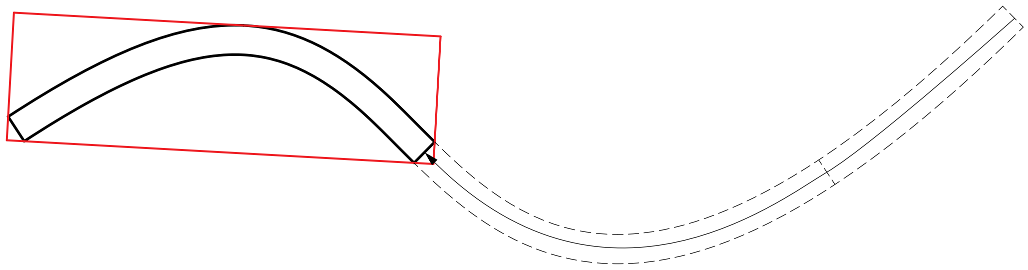


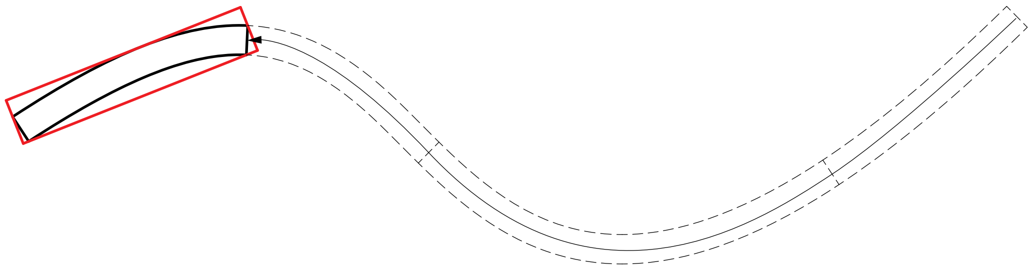


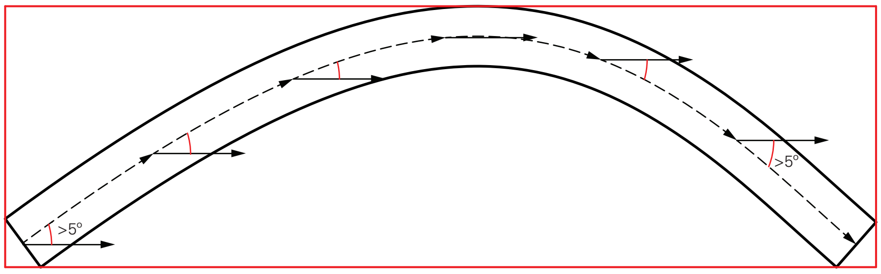
Strategies

Our strategy is to achieve the longest blank possible with the minimum waste, while working within the constraints of manufacturing, structural integrity, and assembly.

Starting with some predefined solid geometry, the goal is to optimize the data before exporting for fabrication. To accomplish this, I developed an iterative, brute-force, algorithm that starts with the maximum potential length and checks it against the constraining parameters - in



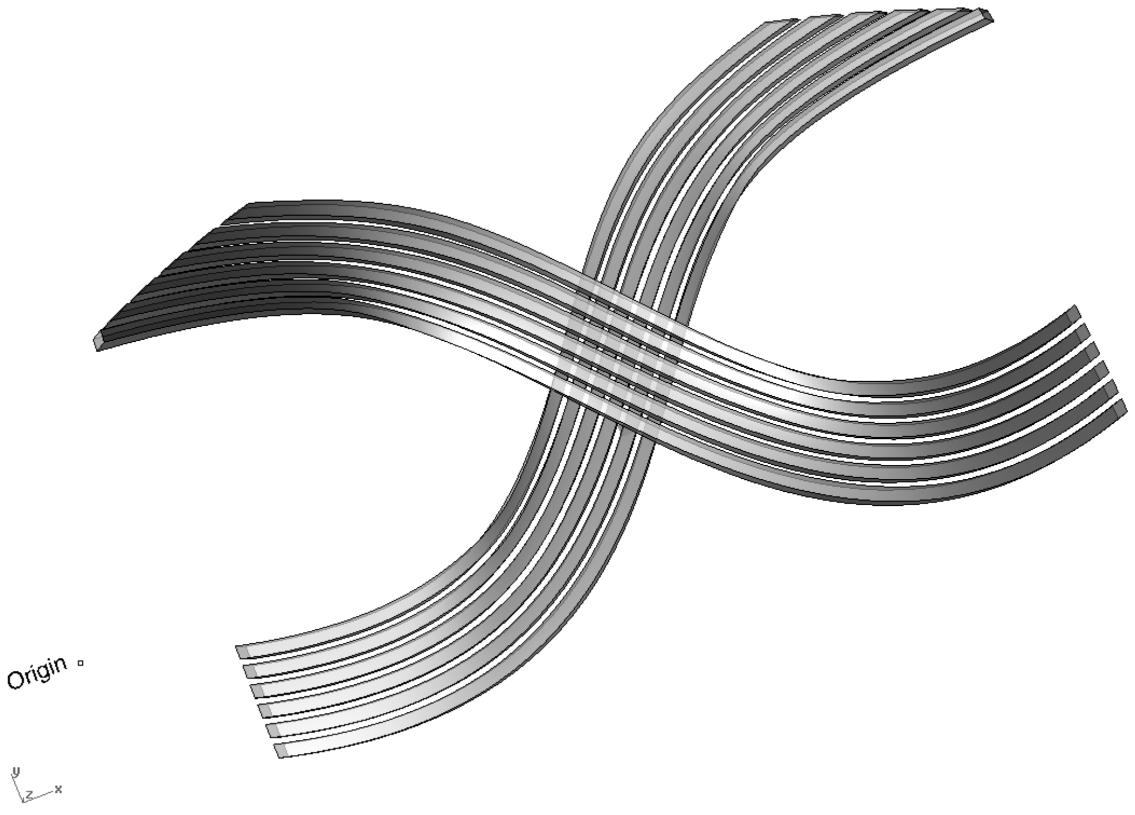




Fiber diection of segment versus blank

this case, the width and depth of the blank, and the fiber direction of the timber. If the constraints are not satisfied, then then the algorithm steps back along the timber beam and tries a shorter blank size. It repeats the process until a solution is reached for the particular segment. It then starts again from the end of the current segment, working its way along the beam. It does this for every beam until all the segments are accounted for. This will produce large, efficient blanks for straighter portions of the beams where curvature is low and shorter blanks where curvature is high.

The most constraining factor here is the fiber direction. We have to minimise the difference in fiber direction to preserve structural action of the beam. The difference between the fiber direction of the blank and the finished segment cannot be greater than 5 degrees.

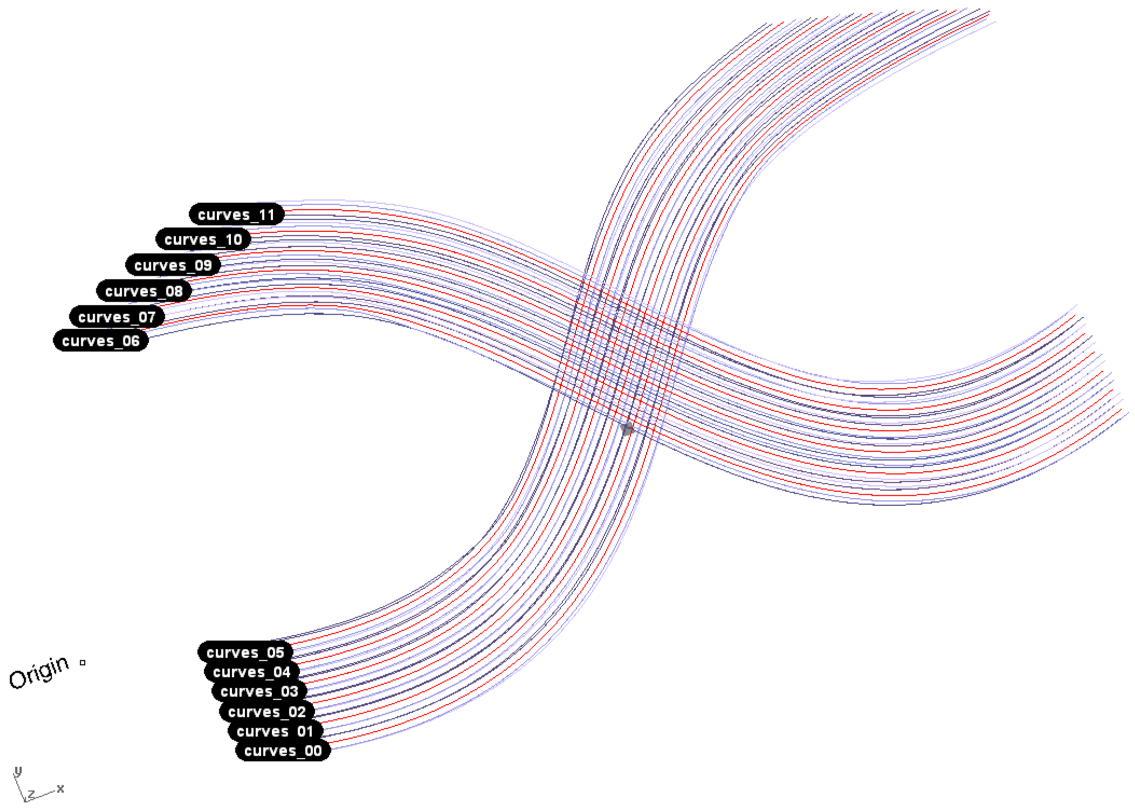


Original modeled volumes

Process

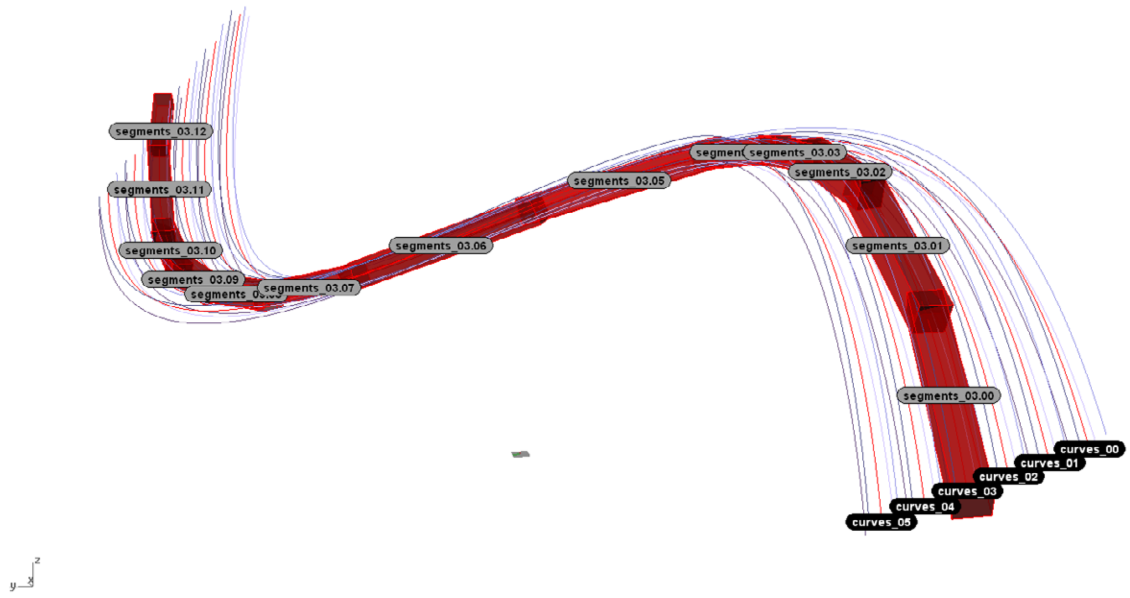
Firstly, global parameters are defined:

1. Labeling, layer names, colours and file naming conventions;
2. the maximum blank length, width, and height;
3. the minimum blank length, width, and height;
4. the excess length, width and height of the blank needed for milling;
5. the maximum permitted deviation of the segment's fiber direction from the blank's z-axis.



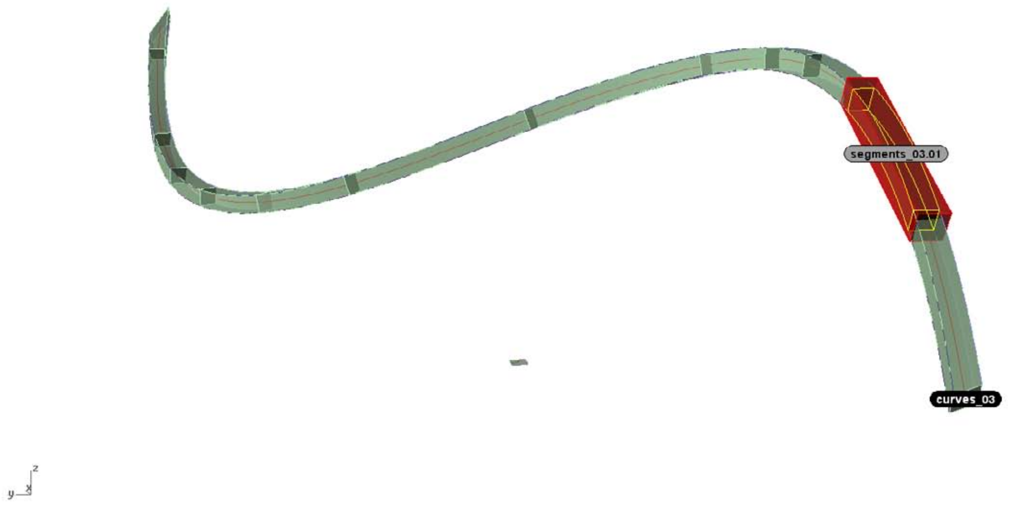
Extraction of edge-curves and centerline

Secondly, the center-lines and edge-curves are extracted from the Rhino beam geometry. All subsequent operation will always be conducted on the beam centerline. The centerline and associated edge-curves are put on appropriate layers, labeled and grouped.

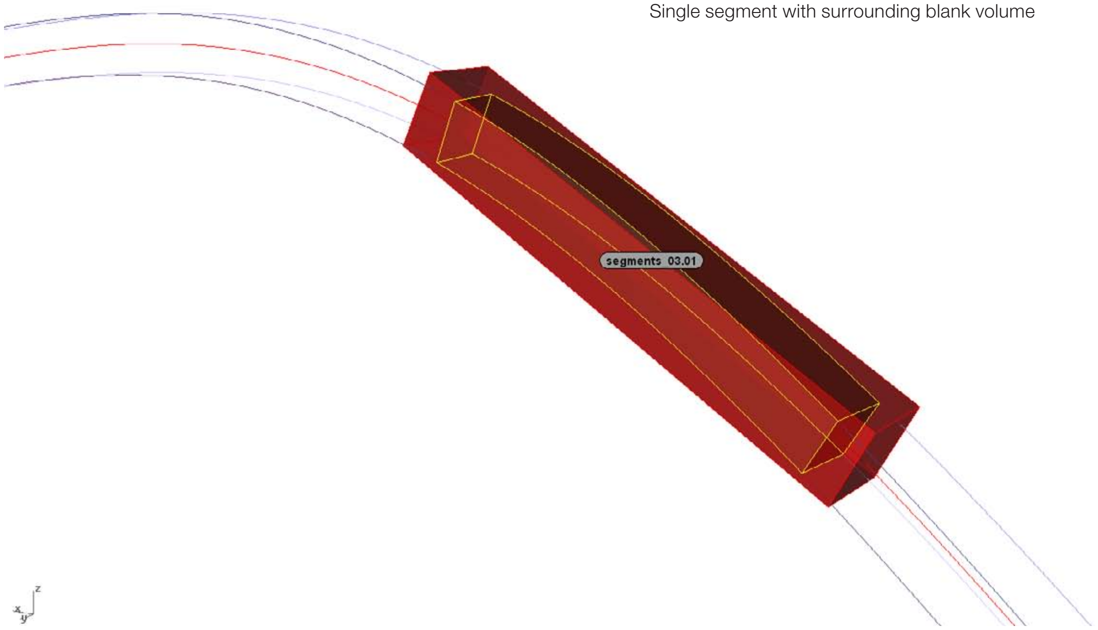


Segmentisation of timber beam

Segmentize Curves: The body of the script uses a iterative, brute-force method to test all possible bounding boxes for the current segment. The script finds the minimum bounding box for the current segment and tests it against the constraining parameters. For example, it tests whether the current blank's width is less than the maximum width of 250mm and greater than the minimum width of 100mm. If, this or any other of the constraining conditions are not met, then it reduces the length of the segment and tries again. It repeats the process, always with the minimum bounding box, until a valid option is found. Once found, it uses the minimum bounding box to create the blank volume.



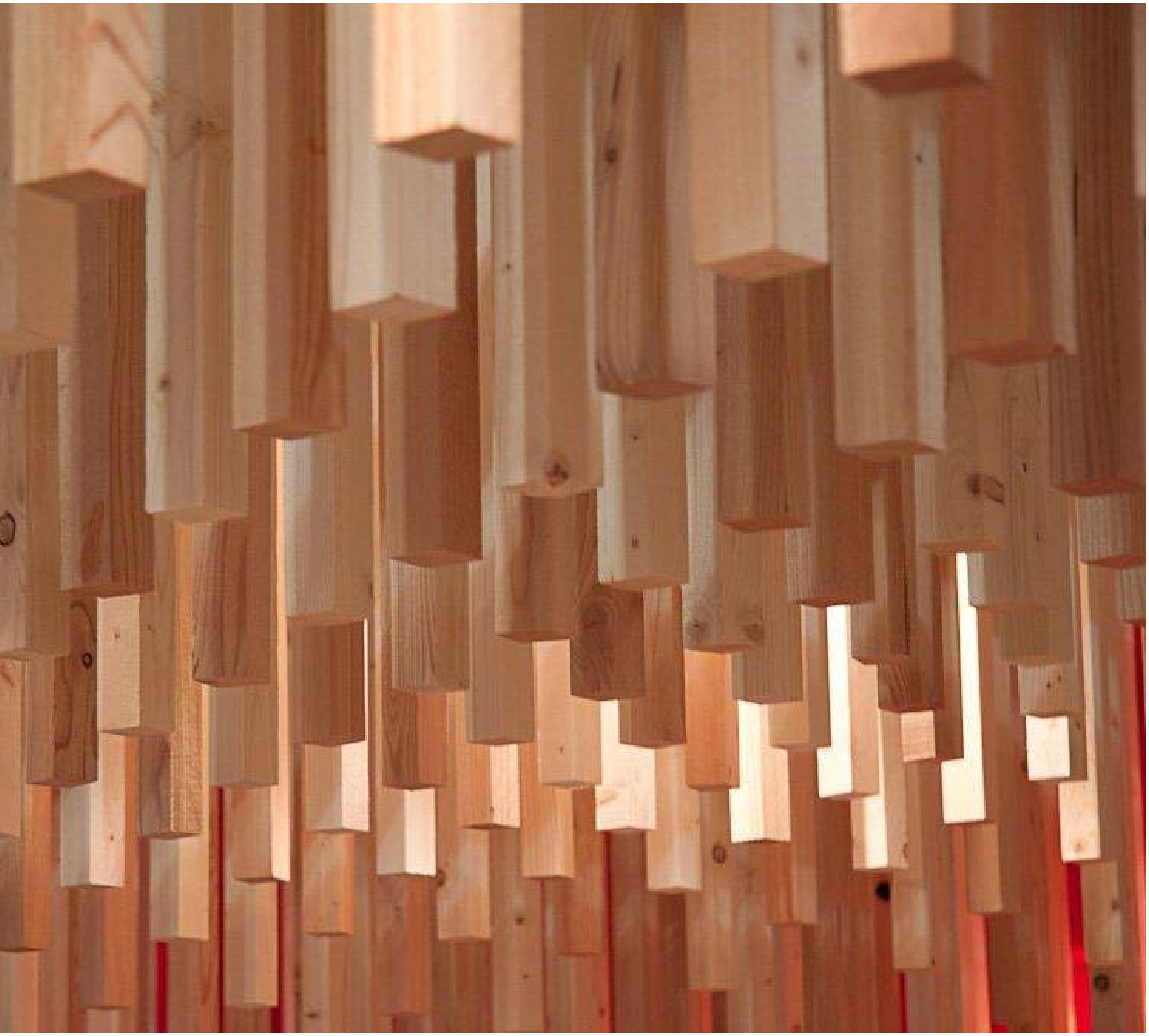
Single segment with surrounding blank volume



Build Segment Volumes: this small script uses the centerline and edgecurves to build the final volume for each discretised segment of the beam.

The final script exports all the fabrication data to an excel file.

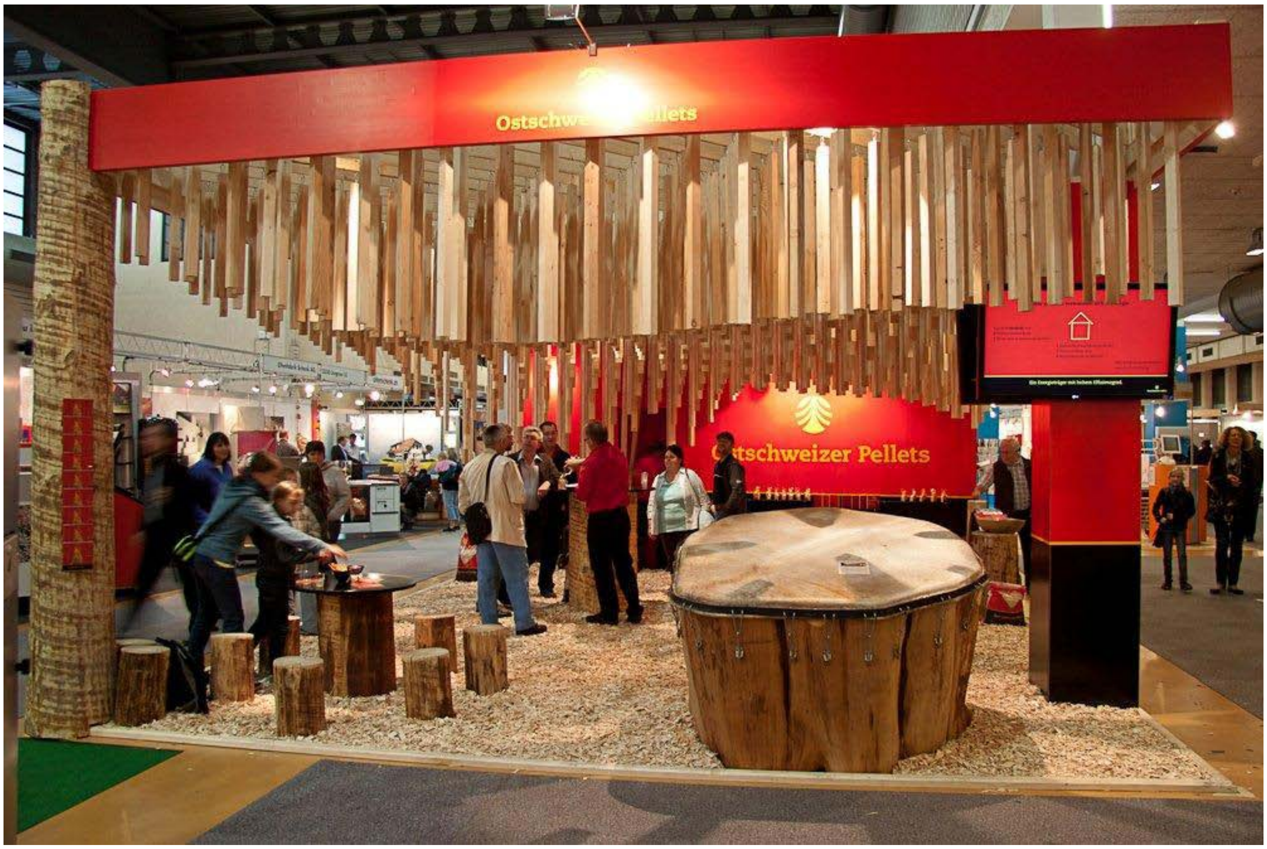
Beniwood Pavilion



The Beniwood Pavilion is a temporary pavilion designed and built for a timber trade show.

Although a separate project, the Beniwood Pavillion explores the final part of the design to production process – exporting the information for fabrication.

From an initial sketch design (given), the brief consisted of two parts: first, to build a parametric Grasshopper model in Rhino for the client to experiment before finalising the design; and second, to produce fabrication and assembly data for the finalized design.



Easy to design

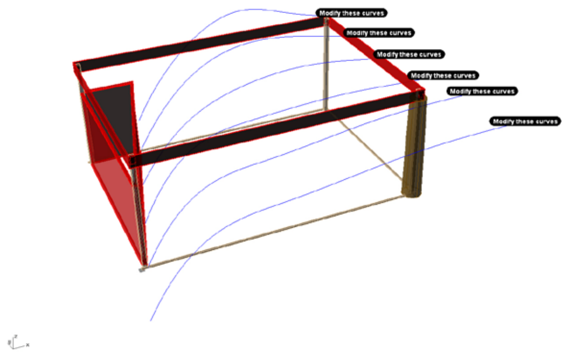
In producing the parametric model for the client, the goal was to incorporate all of the design drivers into the simplest model. The model needed to have sufficient control to allow experimenting with the design while being easy for a non-specialist to manipulate.

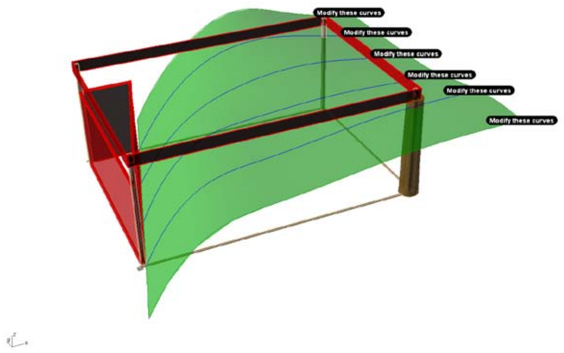
Simple to fabricate

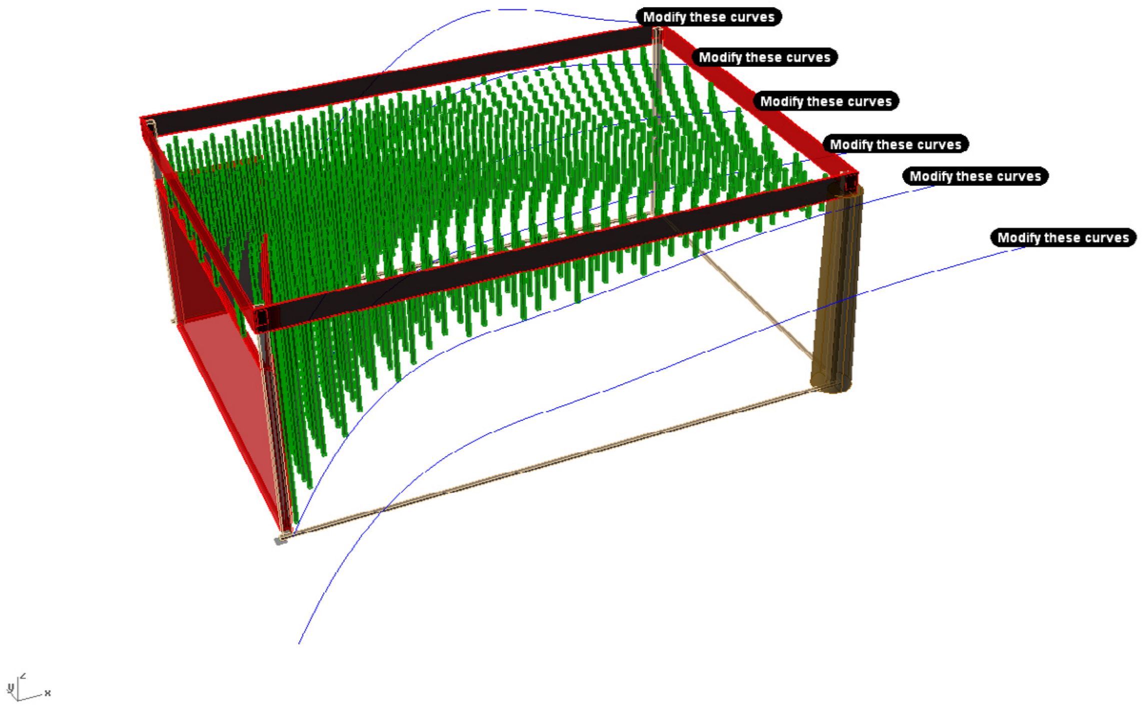
The machine used to fabricate the timber for the pavilion was only available for one day in the coming month. This necessitated a simple and quick fabrication process. To facilitate this, a single timber profile was chosen. The single profile can be fed into the saw machine and cut to the programmed lengths.

Quick to install and remove.

A hanging mechanism was chosen to allow for rapid installation and removal. A simple numbering system was







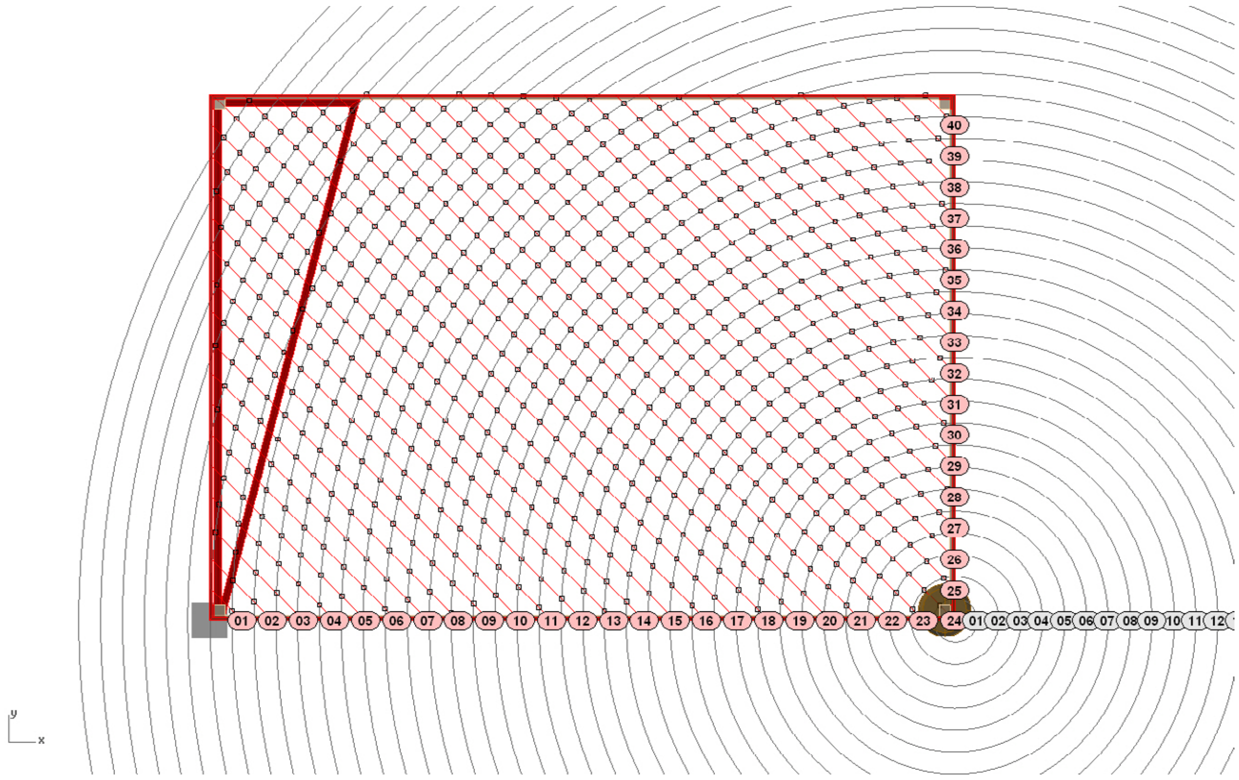
devised to assist assembly on site.

Assembly information also needs to be provided so that the installer can quickly find the required piece, locate where it is to be installed and orient it in the right position. Many things can slow this down, one of the most troublesome, but also easily preventable, is having too much information present. One need the absolute minimal

Parametric Grasshopper model

From the client-supplied sketch model, a reference surface was constructed using six curves. The curves allow simple modification of the reference surface while minimizing complexity.

The vertical timber members form a circular pattern based on the growth rings of a tree. They are to hang from beams cutting across the pavilion. In order to place them, the intersections are found between the beam



curves and a series of offset circles.

A line is drawn between each intersection and its projection on the reference surface.

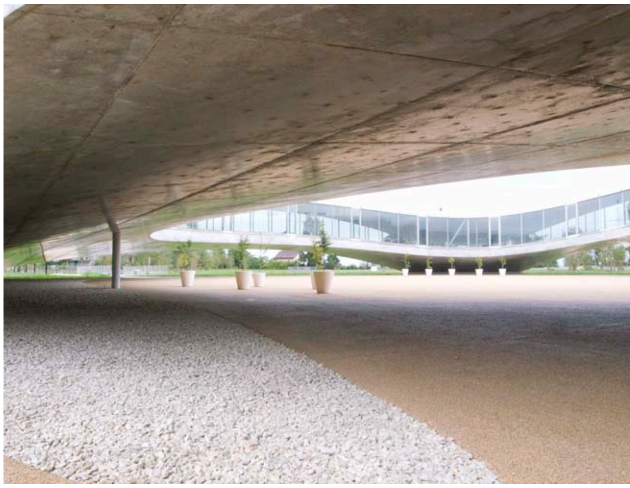
Along this line, a profile is extruded, showing the current design. When ready, this can "baked" out into standard Rhino geometry for the fabrication and assembly scripting.

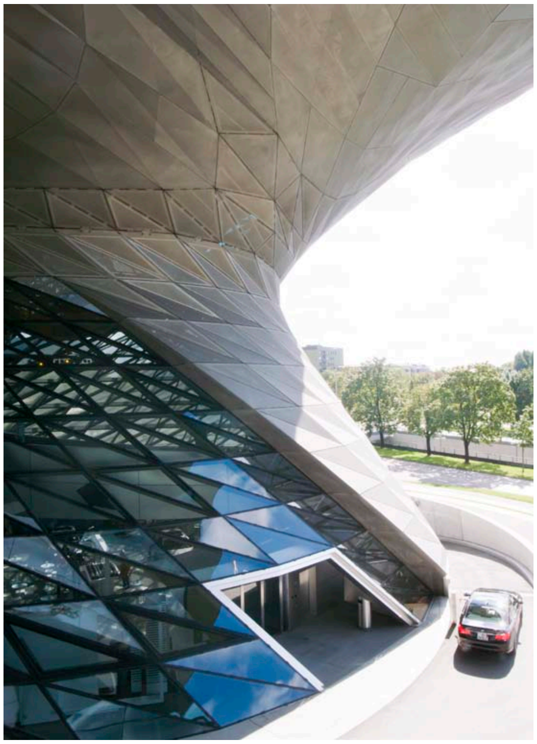
Scripted Output

Once the design was finalised, the geometry was taken from Grasshopper and the output was scripted: the timber boards were labelled according to the beam and circle intersection. The name and length of each board was then exported to an excel spreadsheet for fabrication.

Field Trips

Switzerland is perfectly positioned for weekend visits to most of Western Europe. There are a number of buildings throughout Europe on which Design to Production have collaborated. There are also innumerable architectural projects in between. Part of the research during the two months involved seeing as many recent buildings, which relied on advanced fabrication and assembly, as possible. Although the trip involved visits to six countries in Europe, I shall focus on eight field trips that inform the construction of complex architectural projects: Blumer Lehman timber in Gossau, Switzerland; the ETH fabrication workshop, Zentrum Paul Klee in Bern, Mercedes Benz Museum, Stuttgardt, the EPFL Learning Centre in Lausanne; Munich's Olympic Park and BMW Welt; and the Maaxi in Rome.

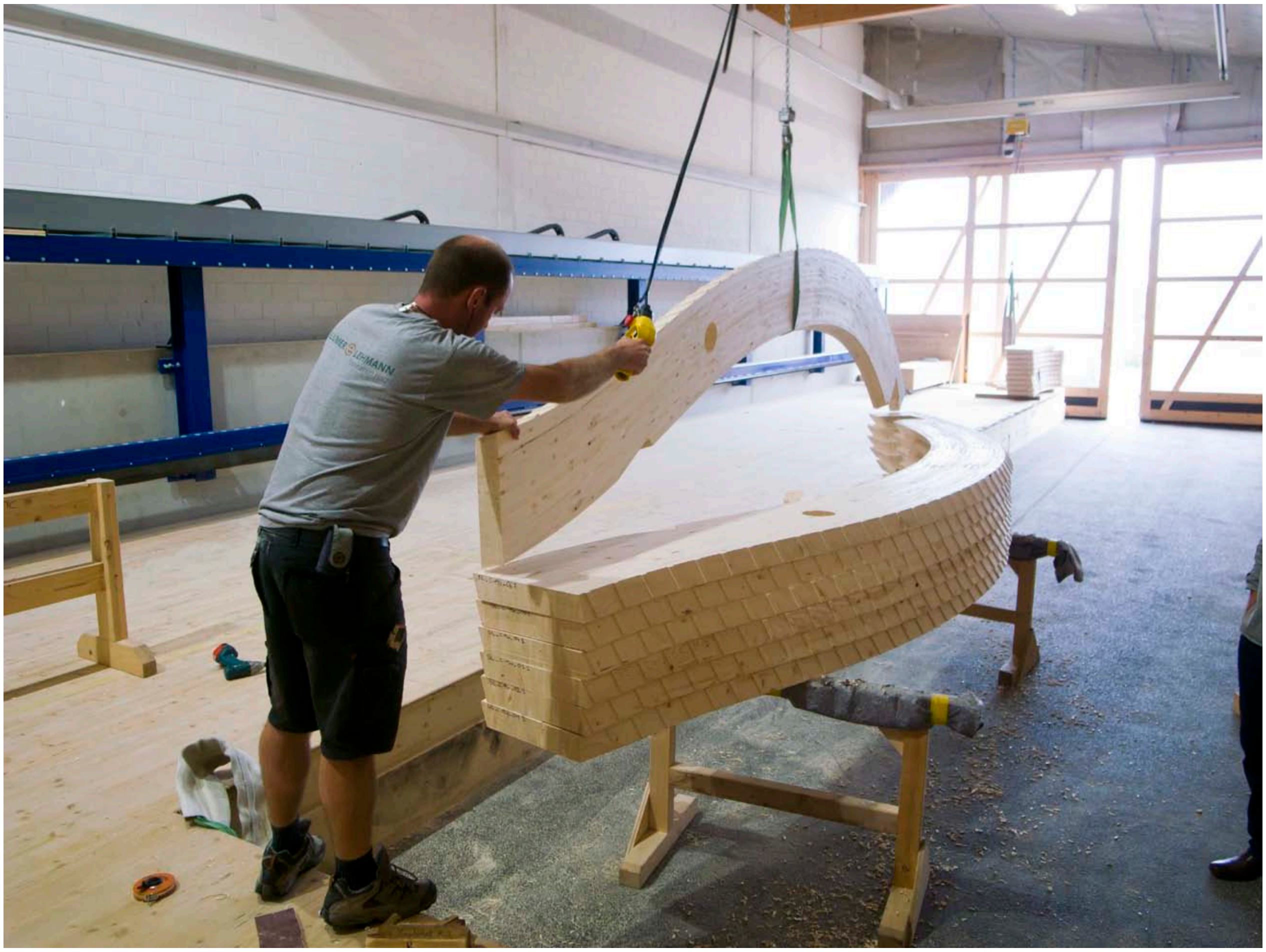




Blumer Lehmann

Located in Gossau, in north-east Switzerland, Blumer Lehmann is a timber processor and fabricator. The factory touches all aspects of timber production, including drying, processing and 5-axis machining of complex timber elements. They accomplish this through a relationship with an engineering firm and Design to Production, built up over a number of projects. Most importantly, they have worked on Shigeru Ban's Golf Club in South Korea and the Kilden Kristiansand Facade in Norway.

Together, the trio have created an effective workflow of knowledge sharing that allows them to complete the most complex of architectural projects. In particular, Blumer Lehmann and Design to Production work very closely together from the beginning of the fabrication. D2P has intimate knowledge of Blumer Lehmann's tools and manufacturing constraints, which enables them to optimise their manufacturing processes.



ETH Fabrication Workshop

The Gramazio and Kohler chair at the ETH, Zurich, "examine(s) the changes in architectural production requirements that result from introducing digital manufacturing techniques." They are particularly interested in the results of combining data and material and the implications this has on architecture. "The possibility of directly fabricating building components described on the computer expands not only the spectrum of possibilities for construction, but, by the direct implementation of material and production logic into the design process, it establishes a unique architectural expression and a new aesthetic." (http://www.dfab.arch.ethz.ch/web/e/about/index.html)

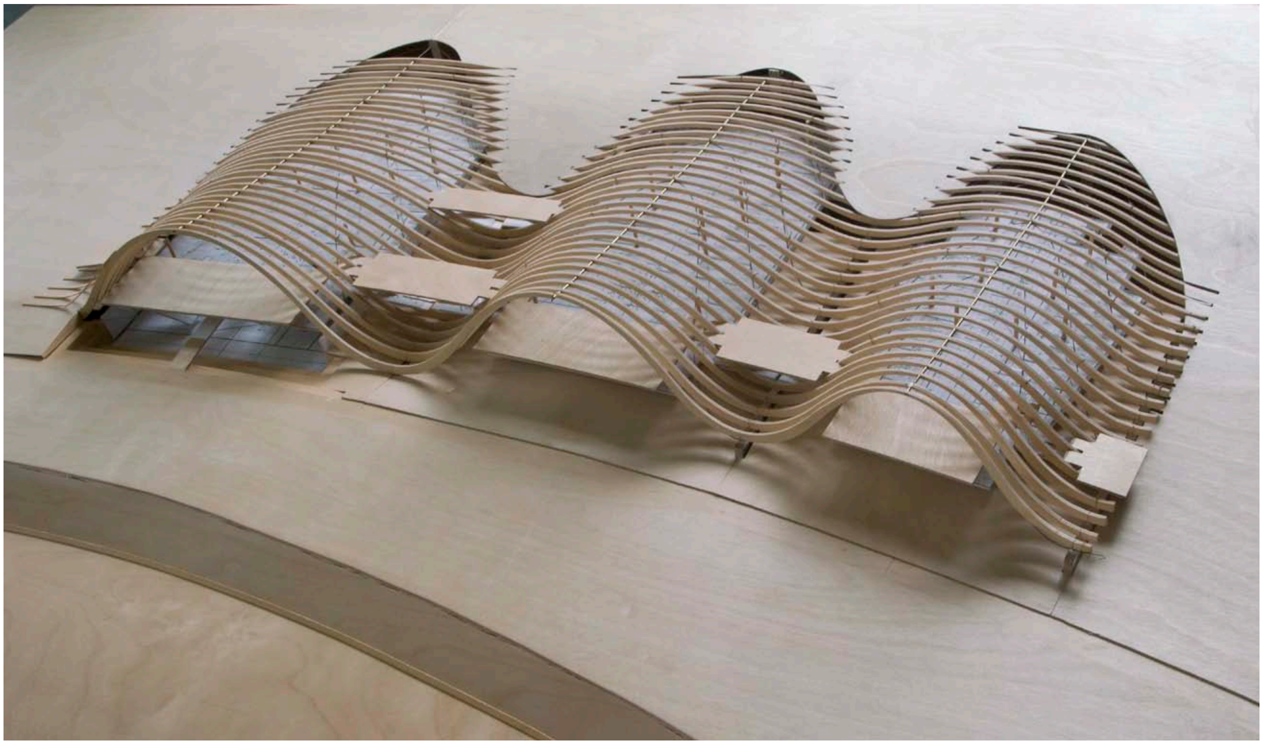
In 2005, Gramazio and Kohler installed a flexible construction facility based on a Kuka robot. The robot travels on a seven meter-long linear axis and has a reach of three meters – able to fabricate and position building parts on a 1:1 scale.

Previously, their research focussed on using the robot to very accurately position simple building elements, such as bricks. They have recently shifted their focus to using the robot to fabricate complex building elements that are simple to assemble by hand.

DeWALT

Zentrum Paul Klee

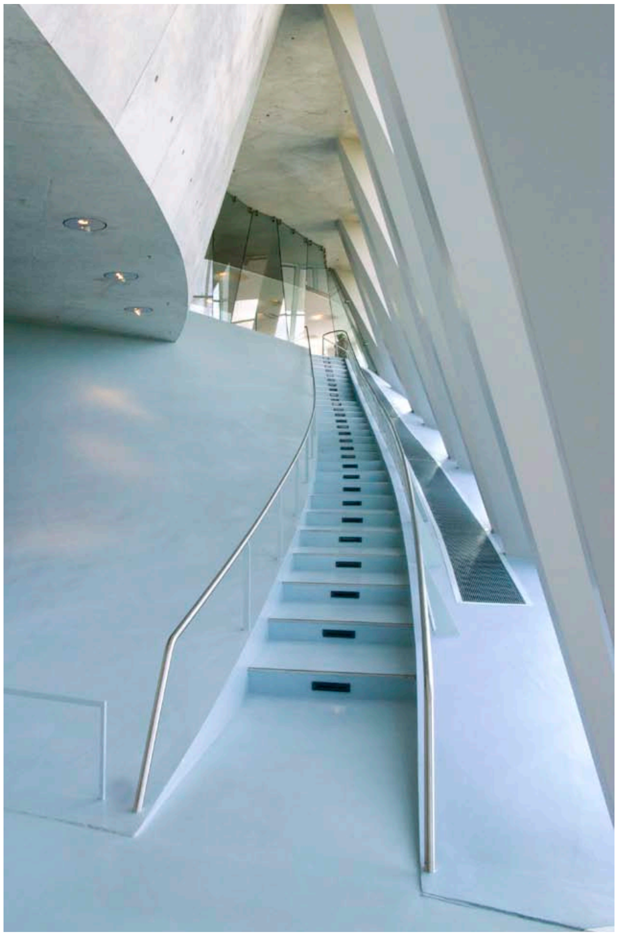
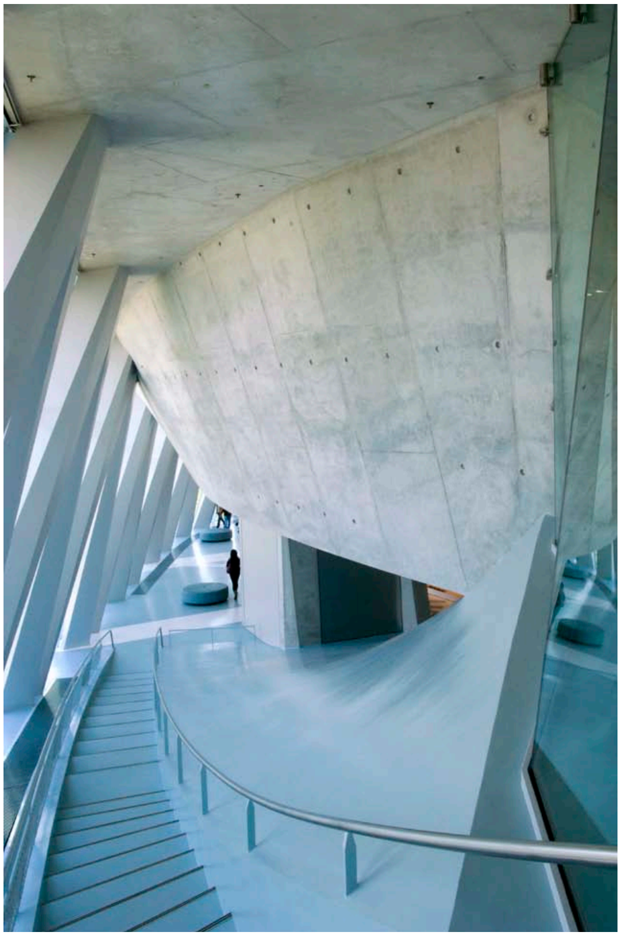
The Zentrum Paul Klee is a museum of art in Bern, Switzerland, dedicated to the work of Paul Klee. Designed by Renzo Piano, it was constructed in 2004. The building's form is a series of waves based on concentric circles. In order for the architect to test alternative solutions, Design to Production developed a parametric model of the steel structure. The model was used to produce the 2-dimensional plans of the curved I-beams for the steel contractor.



Mercedes Benz Museum

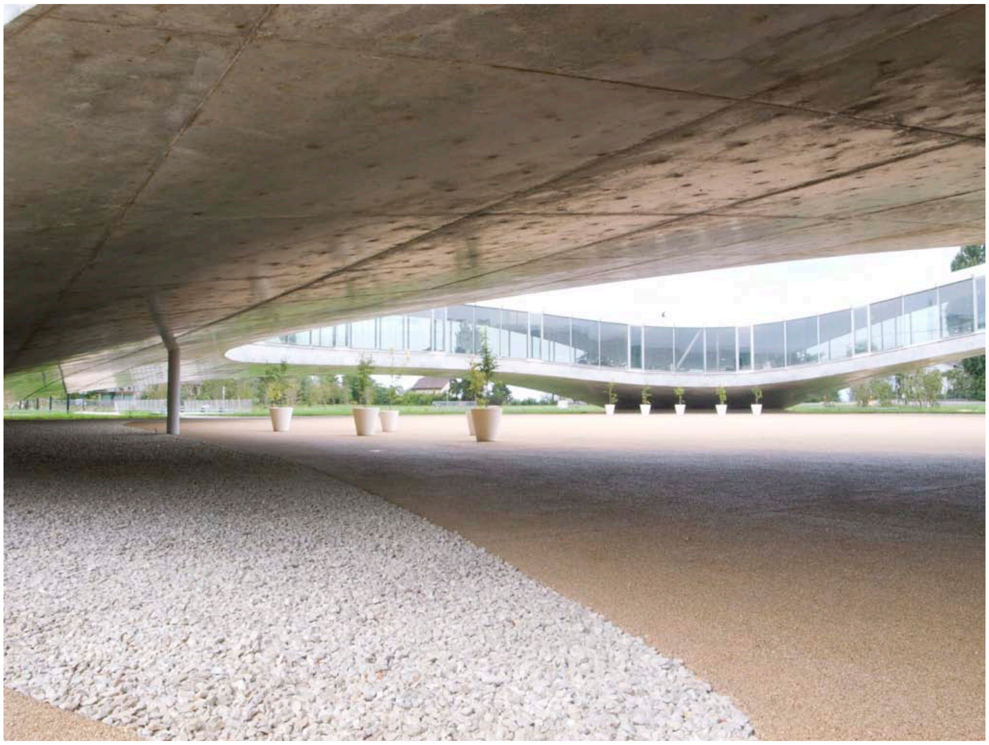
Finished in 2005, UN Studio's Mercedes Benz museum in Stuttgart is composed of two intertwining ramps that spiral through the building. While they create a unique spatial experience for the visitor, they are very difficult to describe with traditional architectural drawings. Further, the complex geometry of the smooth, fair-faced concrete surfaces were beyond the scope of existing formwork systems and manual planning methods.

Design to Production constructed a parametric model of the whole building to coordinate all the subsequent planning steps of the numerous trades involved. The master geometry was used to generate thousands of plans during the building process. They also developed a method for producing doubly curved formwork from planar boards. The formwork panels were all prefabricated on a CNC router and bent to the desired shape in-situ during construction.

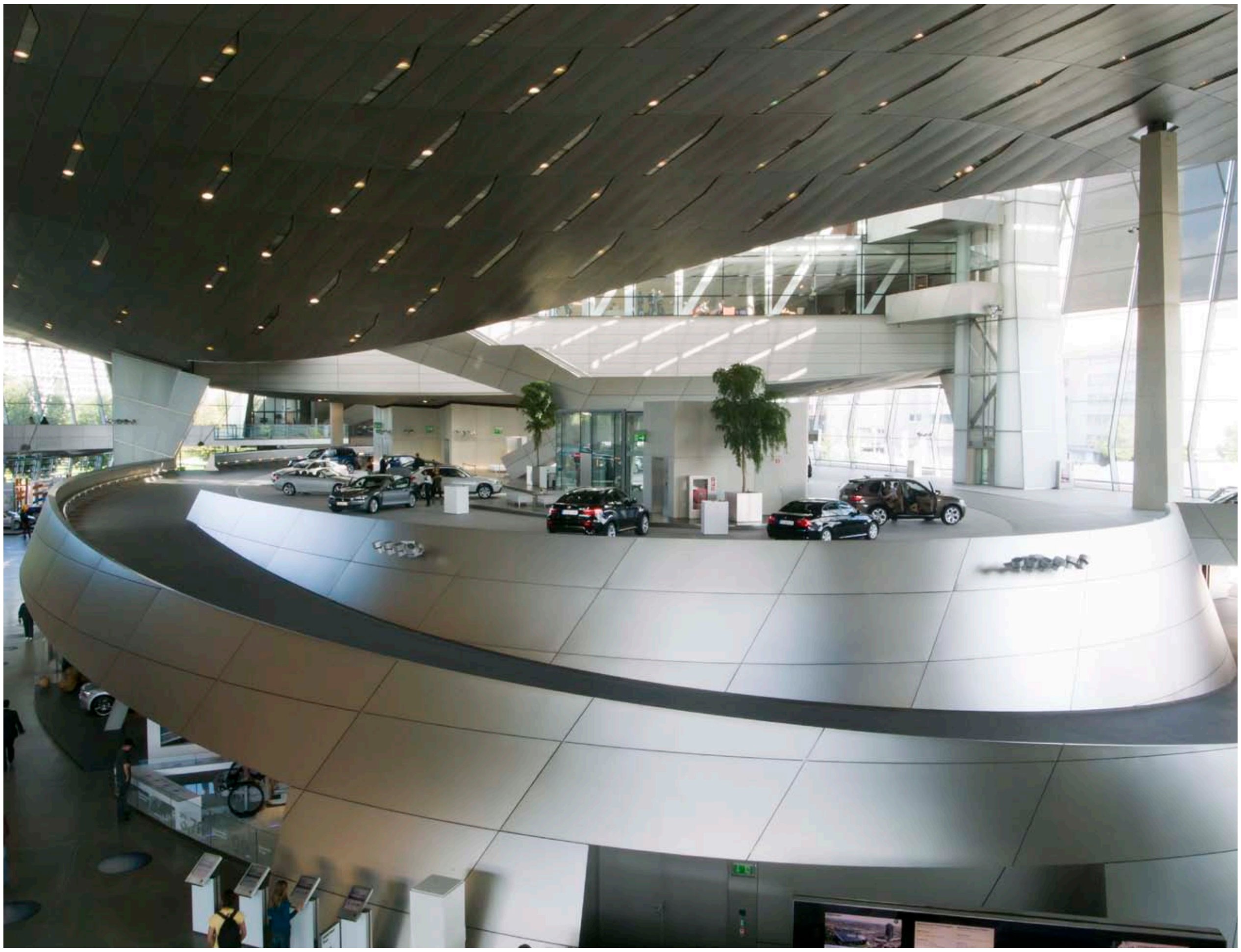



EPFL Learning Center

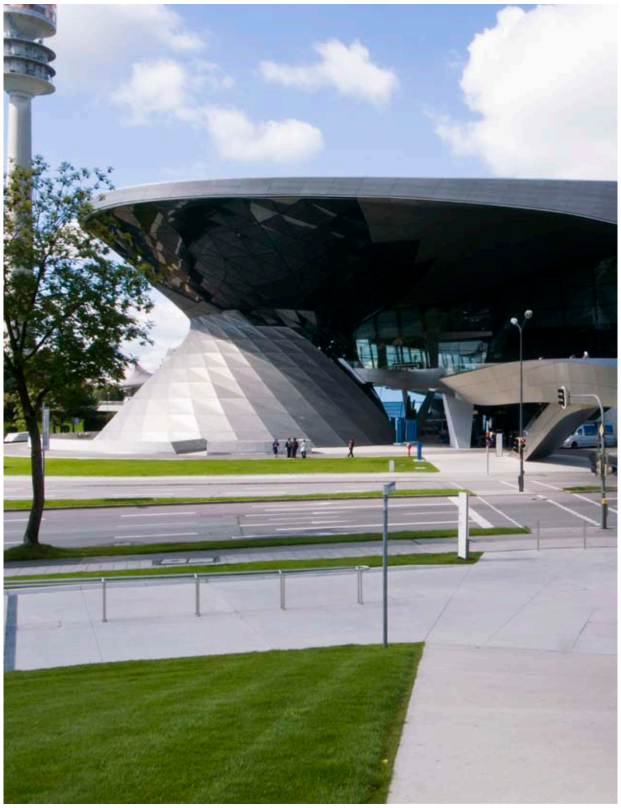
SANAA's Learning Center at the EPFL in Lausanne is defined by smooth, doubly curved concrete slab measuring 7,500 sqm. Design to Production automated the planning process by using the slab surface to produce the detailed plans for all 1,500 formwork tables as well the machine data for the CNC-cutting of 10,000 individual cleats.



BMW Welt

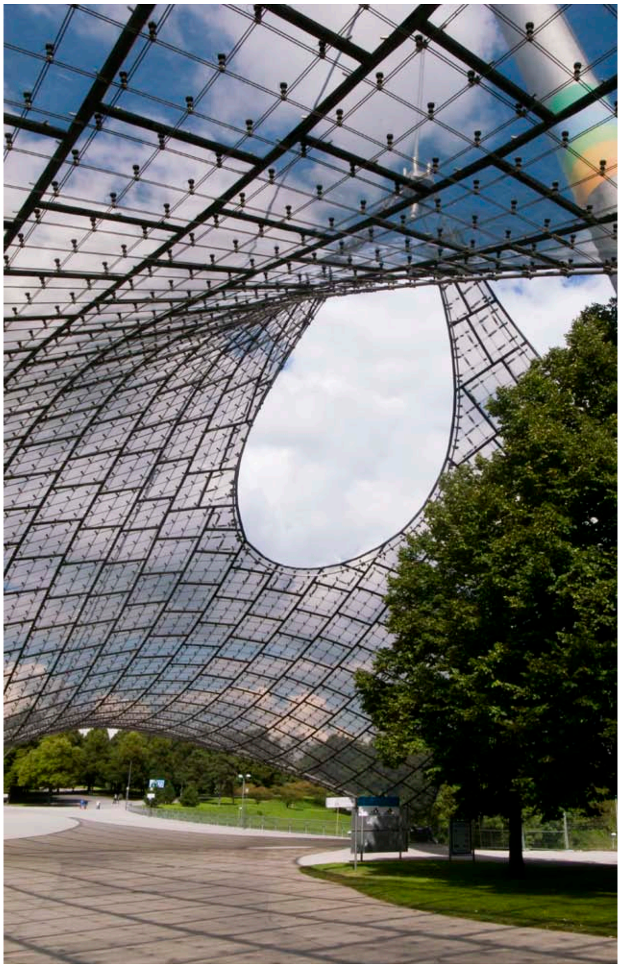
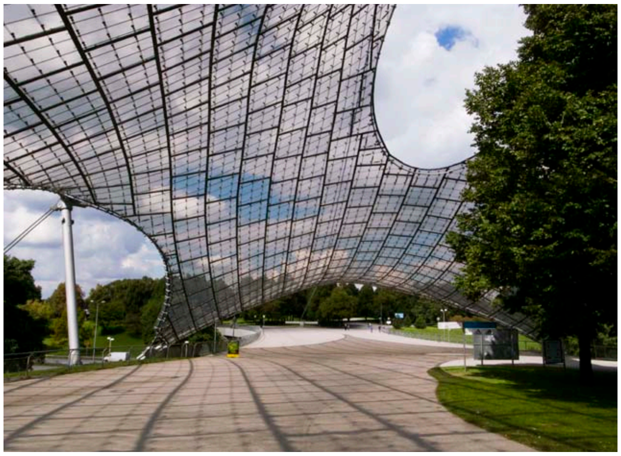
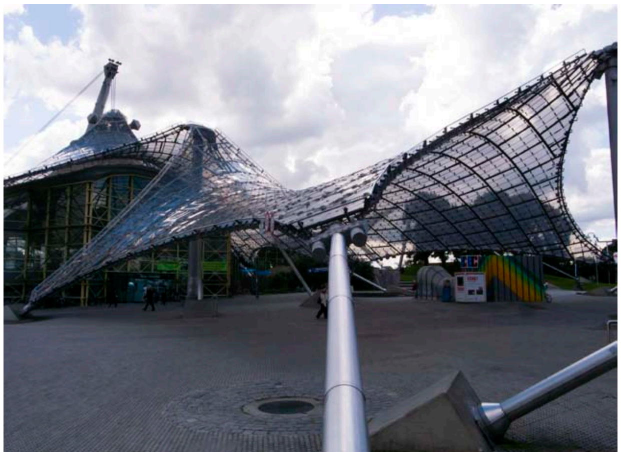


Designed by Coop Himmelb(l)au, the BMW Welt is a multi-function exhibition and car-pick up center in Munich. Constructed in 2007, the building relied on advanced in geometric modelling as well as structural and environmental analysis. Of particular interest is the integration of building services within the structure.

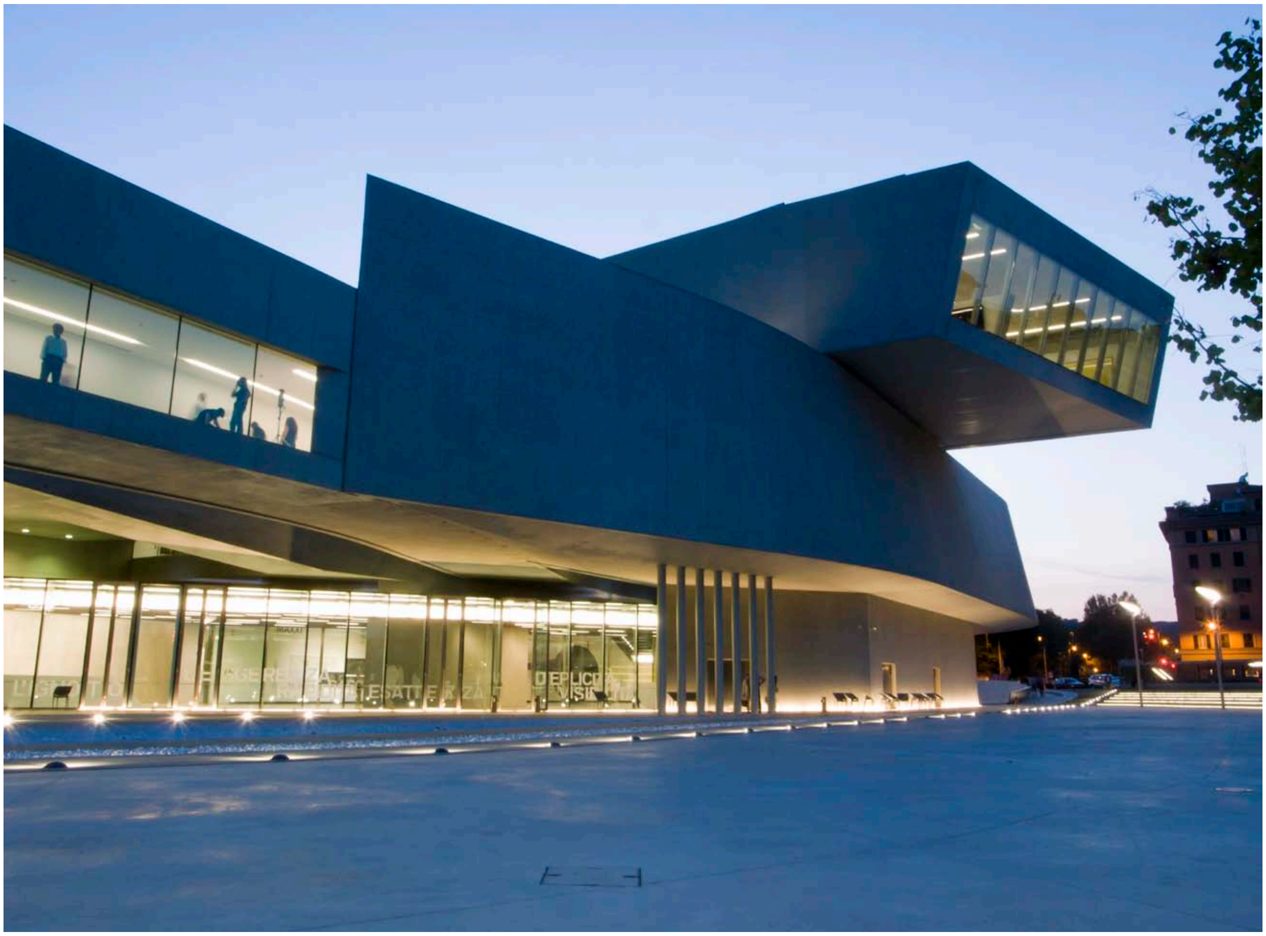


Munich Olympic Stadium

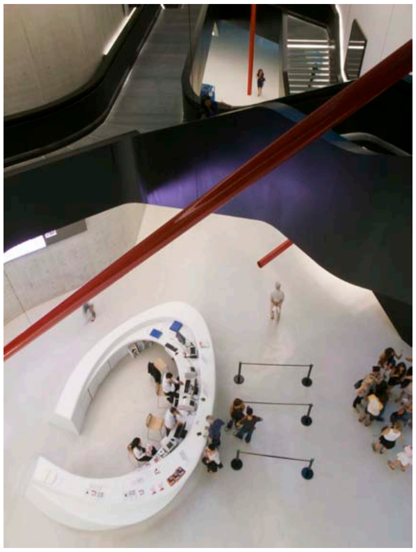
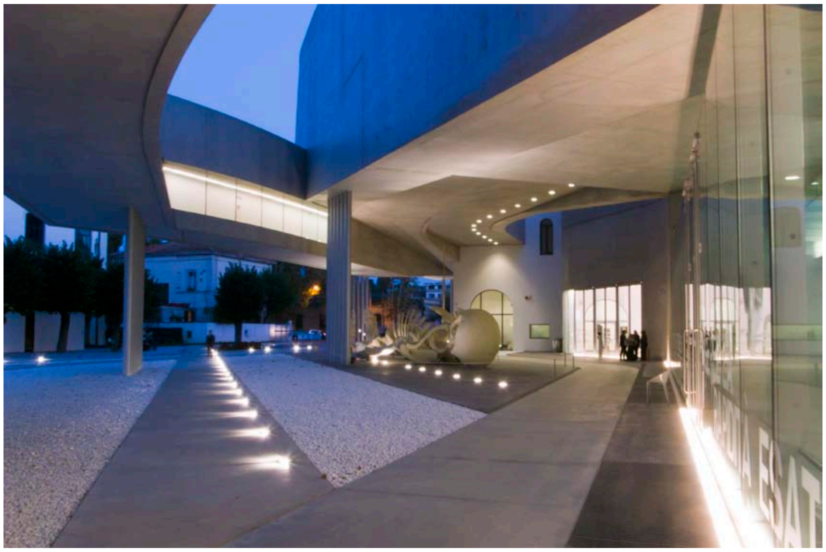
The Munich Olympic Stadium was built for the 1972 Munich Olympics. Designed by architect Gunther Behnisch and engineer Frei Otto, was at the time, and remains a revolutionary structure. Although digital processes were not used, the design and analysis extensively used analogue models to compute structural behaviour. Although four decades old, the analogue computing models of Frei Otto remain a relevant research topic.

Maxxi



Designed by Zaha Hadid and finished in 2009, the Maxxi is a museum of 21st century arts in Rome. The building is of interest, not so much for its digital planning and fabrication, but for its precise realisation of complex geometry.

RhinoScript Python

```
#==================================================================================
# 1.0 INTRODUCTION
#==================================================================================
# 1.1 Python overview
# 1.2 Rhino and Python
# 1.3 Resources and Documentation
# 1.4 External IDEs
# 1.5 Python Identifiers
# 1.6 Indentation
# 1.7 Multi-line statements
# 1.8 Comments
# 1.9 Assignment
# 1.10 Type Testing

#==================================================================================
# 2.0 TYPES
#==================================================================================
# 2.1 Numbers
# 2.2 Strings
# 2.3 Lists
# 2.4 Tuples
# 2.5 Dictionaries
# 2.6 Sets
# 2.7 Reference vs. Copies

#==================================================================================
# 3.0 SYNTAX
#==================================================================================
# 3.1 Statements
# 3.2 Printing: redirecting the output stream
# 3.3 If Test
# 3.4 While Loops
# 3.5 For Loops
```

```
#=================================================================================
# 4.0 FUNCTIONS
#=================================================================================
# 4.1 def Statement
# 4.2 Scope Rules
# 4.3 Global variable
# 4.4 Passing Arguments
# 4.5 Multiple Output

#=================================================================================
# 5.0 MODULES
#=================================================================================
# 5.1 Modules useful for Rhino
# 5.2 Package Imports
# 5.3 Private Data
# 5.4 __name__ and __main__

#=================================================================================
# 6.0 CLASSES
#=================================================================================


#=================================================================================
# 7.0 EXAMPLES
#=================================================================================
# 7.1 Project Setup
# 7.2 Project Parameters
# 7.3 Extracting EdgeCurves and CenterLine from existing geometry
# 7.4 Segmentizing curves according to manufacturing by iterative minimum-bounding box method
# 7.5 Reconstructing segment volumes
# 7.6 Storing data on objects
# 7.7 Exporting data to Excel
# 7.8 Project Documentation
# 7.9 Export geometry for manufacturing
```

```
#===================================================================================
# 1.0 INTRODUCTION
#===================================================================================

#------------------------------------------------------- 1.1 Python overview
‘’’
Python is a high-level, interpreted, interactive and object oriented-scripting language.

Python was developed by Guido van Rossum in the late eighties and early nineties at the
National Research Institute for Mathematics and Computer Science in the Netherlands.
‘’’

#------------------------------------------------------- # 1.2 Rhino and Python
‘’’
# IronPython  is a specific implementation of the python language that runs on top of .NET.
CPython is another common python implementation that people use that is written in C
while Jython is written in Java. I chose IronPython because of it’s incredible ability
to use all of the classes available in the .NET framework including all classes and functions
made available through RhinoCommon

Our python implementation includes a set of very similar functions that can be imported
and used in any python script for Rhino. This set of functions is known as the rhinoscript package.
Documentation on this package can be found at http://www.rhino3d.com/5/ironpython/index.html

Along with the RhinoScript style functions you will be able to use all of the classes in
the .NET Framework, including the classes available in RhinoCommon. As a matter of fact,
if you look at the source for the RhinoScript style functions, they are just python scripts
that use RhinoCommon. This allows you to do some pretty amazing things inside of a python script.
Many of the features that once could only be done in a .NET plug-in can now be done in a python
script.

To use python editor in Rhino, enter >> EditPythonScript
‘’’

#--------------------------------------------- # 1.3 Resources and Documentation
‘’’
Rhino Python
http://python.rhino3d.com/

RhinoScript Functions in RhinoPython
http://www.rhino3d.com/5/ironpython/index.html

RhinoCommon SDK documentation
http://www.rhino3d.com/5/rhinocommon/index.html

IronPython Cookbook
http://www.ironpython.info/index.php/Contents

Python Website
http://www.python.org/

Enthought Python Distribution
```

```
http://www.enthought.com/                    #Python + scientific, math, and graphing libraries
included

The Python Tutorial
http://docs.python.org/tutorial/

Python Docs
http://docs.python.org/

Python Tutorial Series
http://www.tutorialspoint.com/python/index.htm
‘’’

#--------------------------------------------------------- # 1.4 Extenal IDEs
‘’’
Eclipse
http://www.eclipse.org/

PyDev
http://pydev.org/

Tutorial for getting PyDev + Eclipse + Rhino working
http://python.rhino3d.com/entries/12-Configuring-Pydev-for-Rhino.Python
‘’’

#----------------------------------------------------- 1.5 Python Identifiers
‘’’
No punctuation characters
Python is case-sensitve

_identifier >> private
__identifier >> strongly private
__identifer__ >> python-defined special name

‘’’

#------------------------------------------------------------ 1.6 Indentation
‘’’
Blocks of code are defined by indentation, not brackets.
‘’’

#--------------------------------------------------- 1.7 Multi-line statements
‘’’
total = item_one + \
        item_two + \
        item_three

Statements within [], {}, () can drop lines

days = [‘Monday’, ‘Tuesday’, ‘Wednesday’,
        ‘Thursday’, ‘Friday’]
‘’’
```

```
#--------------------------------------------------------------- 1.8 Comments
‘’’
A single line comment begins with a #
A multi-line comment uses triple, single quotes
‘’’

#------------------------------------------------------------- 1.9 Assignment
‘’’
Everything in Python is an object.  Identifiers reference objects.
Pyhton always stores a reference to the object, never a copy of it unless you
specifically request it.
‘’’

#---------------------------------------------------------- 1.10 Type Testing
‘’’
Python does not care about type - objects are polymorphic.  The notion of type lives
with the object, not the variable.  Therefore, type testing is not a good idea in Python
because it limits your codes flexibility to working on just one type.
‘’’
```

```
#=================================================================================
# # 2.0 TYPES
#=================================================================================
‘’’
Python has 5 standard data types: Numbers, Strings, Tuples, Lists and Dictionaries
Every object is classified as immutable or not:
    Strings: immutable
    Numbers: immutable
    Tuples: immutable
    Lists: mutable
    Dictionaries: mutable

Other types include: sets and booleans.
‘’’

#--------------------------------------------------------------- # 2.1 Numbers
‘’’
Python supports four different number types:
- ints
- longs
- floats
- complex
‘’’

# Mixed types are converted up

a = 10              # int
b = 3.1415          # float

c = a + b
print c             # float

‘’’ On Variables
Variable are created when assigned, never ahead of time
Variables refer to objects

This means that you have to assign counters to zero before you can add them
and initialise lists to an empty list before appending them.
‘’’
#--------------------------------------------------------------- # 2.2 Strings
‘’’
Strings in Python are identified as a contiguous set of characters in between quotation marks.
‘’’

# Assignment
# Variable are created by assignment
aString = “I’m sorry, Dave.”
anotherString = “I’m afraid I can’t do that.”

# Print statement
print aString
```

```
# Print the length of a string
print len(aString)

# Print a string concatenated with a non-string
print ‘The number of characters in the above string is:’ + str( len(aString))

# Access the index of a string
print aString[0]
print aString[2]

# Access from the end by negative values
print aString[-1]
print aString[-2]

# Slice a string
print aString[4:9]

# Slice from the beginning to a value
print aString[:9]

# Slice from an value to the end
print aString[11:]

# Slice with a step
print aString[0:(len(aString)):2]
print aString[::3]
print aString[::-1]

# Make a top-level copy of the whole string
copyString = aString[:]
print copyString

# Concatenate strings
print aString + ‘ ‘ + anotherString

# Repetition on strings
print aString[:11] * 4 + ‘ ‘ + anotherString

# String Formatting
‘’’
import rhinoscriptsyntax as rs

curves = rs.GetObjects(‘select’)

for i in range(len(curves)):
    start = rs.CurveStartPoint(curves[i])
    end = rs.CurveEndPoint(curves[i])

    #rs.AddTextDot(i, start)                # Adds the number to each dot
    rs.AddTextDot(‘%04d’ %(i), start)      # Integer/Decimal number, formatted with padding of 4
    rs.AddTextDot(‘%f’ %(i), end)          # Float number
‘’’
```

```
#---------------------------------------------------------------- # 2.3 Lists
‘’’
Lists are positionally ordered collections of arbitrarily typed objects.
Accessed by offset.
Variable length, heterogenous and arbitrarily nestable.
They are mutable - unlike strings, they can be modified in-place.
They are similar to arrays in other languages.


A list contains items separated by commas and enclosed within square brackets [].
‘’’

aList = [3.14, ‘pi’, 314]                   # Can contain multiple data types
print ‘aList = ‘ + str(aList)
print ‘aList[1] = ‘ + str(aList[1]  )                          # Accessed by offset

anotherList = aList[1:]                      # Slicing returns a new list
print anotherList

thirdList = aList       # Concatenation returns a new list as well
thirdList[1] = ‘hi’
print thirdList
print aList

emptyList = []                                # Start with an empty list
emptyList.append(1)                           # Grow the list
emptyList.append(2)
emptyList.append([6, 1, 1])
emptyList.append([3, 4, 5])

print emptyList

emptyList.extend([6, 4, 5])                  # Extend the list
print str(emptyList) + ‘ extened’

emptyList.sort()                              # Sort the list
print str(emptyList) + ‘ sorted’

emptyList.reverse()                           # Reverse the list
print str(emptyList) + ‘ reversed’

emptyList.pop(2)                              # Shrink the list
print str(emptyList) + ‘ index 2 popped’


# 2.3.1  Bounds Checking
‘’’
You cannot reference items that do not exist
eg: emptyList[301] will return an error
‘’’
fifthList = [None] * 10
print fifthList
print ‘The length of the fifthList is: ‘,(len(fifthList))


# 2.3.2 Nested Lists
‘’’
Core data types in Python support arbitrary nesting
eg.
‘’’
pointList = [
             [‘origin’, 0, 0, 0],
             [‘point_1’, 2, 3, 5],
             [‘point_2’, 10, 40, -5],
             ]
print pointList
print pointList[1]
print pointList[2][3]

‘’’
When working with nested lists in Rhino, which is very common, it is important to keep
in mind the difference between Appending to the list and Extending the list.
‘’’
pointList.append([‘point_3’, 20, -11, 7])       # Append adds another list to the list
print pointList

pointList.extend([‘point_4’, 54, 93, -80])      # Extend merely adds the values to the end - ie.
flattens the list.
print pointList

pointList = pointList[:-4]
print pointList


# 2.3.3 An Iterable Object
‘’’
We can iterate through the values in a list
‘’’
for point in pointList:
    print ‘point  =  ‘ + str(point)

# 2.3.4 List Comprehensions
col1 = [row[1] for row in pointList]            # Collect the items in the second position in each
list.
print col1
```

```
#------------------------------------------------------------------ 2.4 Tuples
‘’’
A tuple is a sequence, like a list, but one that is immutable and hence cannot be changed.
Coded in () rather than []

Because tuples cannot be changed, they provide data integrity where lists do not.
‘’’

aTuple = (1,2,3,4)
print ‘Tuple length is: ‘, len(aTuple)
print ‘Tuple = ‘, aTuple


#------------------------------------------------------------ 2.5 Dictionaries
‘’’
Dictionaries are mappings that store objects, not by position, but by key.
Coded in curly brackets {}

Accessed by key, not offset
Unordered collection of arbitrary objects
Variable length, heterogenous, and abitrarily nestable
Of the category mutable mapping
‘’’
import rhinoscriptsyntax as rs

def SortedDictValues(adict):
    keys = adict.keys()
    keys.sort()
    return map(adict.get, keys)

# An origin and three points
origin = [0,0,0]

point_1 = [10,0,0]
point_2 = [0,30,0]
point_3 = [0,0,20]

# Distances between the origin and each point
length_1 = rs.Distance(origin, point_1)
length_2 = rs.Distance(origin, point_2)
length_3 = rs.Distance(origin, point_3)

# Tuples of points signifying lines
line_1 = [ [0,0,0], [10,0,0] ]
line_2 = [ [0,0,0], [0,30,0] ]
line_3 = [ [0,0,0], [0,0,20] ]

# A dictionary in which the ‘lines’ are associated with their lengths
dict = { length_1:line_1, length_2:line_2, length_3:line_3  }
print ‘Dictionary (keys:values) are: ‘, dict

# Return sorted keys and values in three steps
```

```
keys = dict.keys()                            # Get the keys
print ‘Keys = ‘ + str(keys)

keys.sort()                                   # Sort the keys
print ‘Sorted keys = ‘ + str(keys)

for k in keys:                                # Iterate through the sorted list of keys
    print k, ‘=>’, dict[k]

# Return sorted keys and values in one step
for key in sorted(dict):
    print key, ‘=>’, dict[key]

# Function returns the sorted values
sortedDict = SortedDictValues(dict)
print ‘Sorted Dictionary Values = ‘, sortedDict


#-------------------------------------------------------------------- 2.6 Sets
‘’’
Sets are collections of objects and support boolean operations as well as more exotic set
operations.
‘’’

# Two ways to make sets
x = set(‘abcde’)
y = set([‘b’, ‘c’, ‘l’, ‘m’, ‘n’])

print ‘x = ‘, x
print ‘y = ‘, y

# Set membership
if ‘e’ in x:
    print ‘True’
else:
    print ‘False’

# Union
print x | y

# Difference
print x - y

# Intersection
print x & y
#----------------------------------------------------- 2.7 Reference vs. Copies
‘’’
While working, you may be assigning multiple references to the same object.  Be aware that
changing a mutable object in place can affect other refernces down-stream.  If you don’t want
this to happen, you need to explicitly make copies of your objects.

Making copies
```

```
* Slice with empty limits        L[:]
* Dictionary copy method          D.copy()
* Some built-in functions        list(L)
‘’’

a = [1, 2, 3]
b = [a, 4, 5, 6]
c = [a[:], 4, 5, 6]

print ‘a =>’, a
print ‘b =>’, b
print ‘c =>’, c

a[0] = 99

print ‘a =>’, a
print ‘b =>’, b
print ‘c =>’, c




#import rhinoscriptsyntax as rs
```

```
#==================================================================================
# 3.0 SYNTAX
#==================================================================================

#------------------------------------------------------------- 3.1 Statements
'''
Printing                    print
Selecting                   if/elif/else
Sequence iterations         for/else
General Looping             while/else
Empty Placeholder           pass
Loop jumps                  break,        continue
Catching exceptions         try/except/finally
Triggering exceptions       raise
Module access               import,    from
Building functions          def, return, yield
Building objects            class
Namespaces                  global
Deleting references         del
Runnign code strings        exec
Debugging checks            assert
Context managers            with/as
'''

#---------------------------------- 3.2 Printing: redirecting the output stream
print '### 3.2 Printing ###'

log = open('log.txt', 'w')

a = 1
b = 2
c = 3

print >> log, a, b, c              # Saves all print statements to the log file
print a, b, c

print >> log, a+b
print >> log, a+b+c

log.close()                        # Close the log
print open('log.txt').read()      # Read its contents back in


#---------------------------------------------------------------- 3.3 If Test
print '### 3.3 If Test ###'
'''
if <test>:
    <statments1>
elif <test2>:       # Optional
    <statements2>
else:               # Optional
    <statments3>
```

```
'''

if a < b:
    print 'a is less than b'
elif a > b:
    print 'a is greater than b'
else:
    print 'a equals b'

#------------------------------------------------------------ 3.4 While Loops
print '### 3.4 While Loops ###'
'''
while <test>:
    <statements1>
else:
    <statements2>



while <test1>:
    <statements1>
    if <test2>: break        # Exit loop and skip else
    if <test3>: continue     # Go to top of loop
else:
    <statements2>
'''

a = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
while a:            # While a is not empty
    print a
    a = a[1:]       # Strip first character

a = 0
b = 10
while a < b:
    print a,
    a += 1

#-------------------------------------------------------------- 3.5 For Loops
print ''
print '### 3.5 For Loops ###'
'''
General python iterator
Iterable objects include strings, lists, tuples and...

An object is considered iterable if it is either a pysically stored sequence,
or an object that produces one result at a time in the context of an iteration tool
like a for-loop. ie. pysical and virtual sequences

Does not work with fractional steps!


for <target> in <object>:       # Assign object items to target
```

```
    <statements>                # Use target in repeated loop body
else:
    <statements>


for <target> in <object>:       # Assign object items to target
    <statements>
    if <test1>: break           # Exit loop
    if <test2>: continue        # Go to top
else:
    <statements>

‘’’

pointList = [[0,0,0],[10,0,0],[0,20,0],[0,0,30]]
for point in pointList:
    print point

# Counter loops
for x in range(10):
    print x,

print ‘’
for x in range(-10, 10):
    print x,

print ‘’
for x in range(-10, 10, 2):
    print x,

print ‘’
print ‘### Zip ###’
‘’’
points_1 = [[0,0,0],[10,0,0],[20,0,0],[30,0,0]]
points_2 = [[0,10,0],[10,10,0],[20,10,0],[30,10,0]]
points_3 = [[0,20,0],[10,20,0],[20,20,0],[30,20,0]]
points_4 = [[0,30,0],[10,30,0],[20,30,0],[30,30,0]]

pointList = [points_1, points_2, points_3, points_4]
print pointList

for points in pointList:
    rs.AddPolyline(points)

newPointList = zip(points_1, points_2, points_3, points_4)
print newPointList

for points in newPointList:
    lines = rs.AddPolyline(points)

‘’’
# Using a fractional step in a For-Loop
```

```
def frange(start, stop, step):
    rc = []
    x = start
    while( x<=stop ):
        rc.append(x)
        x+=step
    return rc

for i in frange(0,10, 0.5):
    print i,
#=================================================================================
# 4.0 FUNCTIONS
#=================================================================================

‘’’
‘def’ is an executable statement - it does not exist until Python reaches the code
and runs it.  For this reason, your functions have to come before they are called.

‘global’ inside the function declares module level variables that are to be assigned.

arguements, return values, and variables are not declared.

‘’’

#----------------------------------------------------------- 4.1 def Statement
‘’’
def <name> (arg1, arg2,... argN):
    “function_docstring”
    <statements>


def <name> (arg1, arg2,... argN):
    “function_docstring”
    <statements>
    return <value>
‘’’


#------------------------------------------------------------- 4.2 Scope Rules
‘’’
Names defined inside a def can only be seen by code within that def.
Names inside the def do not clash with variable outside the def.

* The enclosing module is global scope
* Global scope spans a single file only
    Names are partitioned into modules and you must import the module to use
    any names it defines.

* Each call to a function creates a new local scope.
* Assigned names are local unless declared global
* Name Resolution: LEGB (Local, Enclosing defs, Global, Built-ins)
‘’’


#--------------------------------------------------------- 4.3 Global variable
x = 10

def func_1():
    x = 20      # x is local here
    print x
func_1()
print x
```

```
def func_2():
    global x    # define x as global
    x = 20      # change global variable

func_2()
print x

#------------------------------------------------------- 4.4 Passing Arguments
‘’’
* Arguments are passed by automatically assigning objects to local names.
* Assigning to argument names inside a function doesnt affect the caller.
* Changing a mutable object argument in a function may impact the caller.
‘’’

def multiply(x, y):
    z = x * y
    return z

a = 5
b = 3

answer = multiply(a, b)
print a, ‘*’, b, ‘=’, answer


#---------------------------------------------------------- 4.5 Mutiple Output
‘’’
Python can return multiple values by returning a tuple and then assigning the results
back to the original argument names.
‘’’

def multi(x, y):
    mult = x * y
    add = x + y
    return mult, add

a = 5
b = 3

multiply = multi(a, b)
print a, ‘*’, b, ‘=’, multiply
#print a, ‘+’, b, ‘=’, addition
```

```
#==============================================================================
# 5.0 MODULES
#==============================================================================

‘’’
Each python file is a module (eg. module.py).  Modules import other modules to
use the names they define.

import    fetches a whole module
from      fetches particular names from a module
reload    reloads a modules code

Modules organize components into a system by serving as self-contained packages of
variables (ie. namespaces)

Imports happen only once.
If you change anything in a module that has already been imported, you need to ether
reload your module, or restart your session.
In the rhino python editor, go to Tools > Reset Script Engine



import aModule                    # imports the module as a whole
aModule.aFunction(argument)       # must qualify to get the names

import aModule as m               # imports the module and assigns a short name
m.aFunction(argument)             # qualify with short name

from aModule import aFunction     # copies out the variable
aFunction(argument)               # no need to qualify names

from aModule import *             # copies out all variables
aFunction(argument)
anotherFunction(arguemnt)

NB. Be careful with this last way of importing as it can corrupt namespaces.

‘’’
#-------------------------------------------------- 5.1 Module useful for Rhino
‘’’
<import rhinoscriptsyntax as rs>
Imports all of the RhinoPython functions.  They are the same as their RhinoScript
equivalents.

<import lib.d2pLib as dp>
Import the D2P library functions.

<import math>
Imports math functionality.

import xlwt
Imports the excel writer library.
```

```
‘’’


#------------------------------------------------------------ 5.2 Package Imports
‘’’
Imports can also name a directory path - known as a package import.
This turns the directory into a namespace.

For the directory:
C:\code\dir1\dir2\mod.py

import dir1.dir2.mod

NB. Each directory named in the import must contain a file named:
__init__.py

eg.

code\
    dir1\
        __init__.py
        dir2\
            __init__.py
            mod.py

The init file can contain Python code, or it can be empty.


Here is the current folder structure:
D2P\
    BLU_10_0000_Globals.py
    script_1.py            # These all import d2pLib as dp
    script_2.py
    script_N.py
    lib\
        __init__.py
        d2pLib.py            # This collects all the libraries into one namespace to make importing
easier
        python\
            __init__.py
            lib_1.py
            lib_2.py
            lib_N.py

‘’’

#--------------------------------------------------------------- 5.3 Private Data
‘’’
Python cannot strickly hide data and therefore is not really encapsulated.
However, there are two ways to hide data from imports:

1.    _x    Prefix the name with a single underscore
```

```
2.    Put __all__ at the top level of a module.  This will only allow those names
      listed in the all to be copied out.  Eg:

      __all__ = [‘functionName_1’, ‘functionName_2’, ‘functionName_3’, ‘functionName_N’]
‘’’


#--------------------------------------------------- 5.4 __name__ and __main__
‘’’
Each module has a built-in attribute called __name__ that is set automatically
as follows:

If the file is being run as a top-level program:
    __name__ is set to the string __main__ when it starts

If the file is being imported:
    __name__ is set to the modules name

This allows you to use your code as a library module of tools or an executable program.
```