AUTODESK **UNIVERSITY** 2014

Dynamo Hero: Using Revit Scripting Tools to Optimize Real-World Projects

Michael Hudson – Flanagan Lawrence, Head of BIM
Andrea Vannini – Flanagan Lawrence, Architectural Scripting Specialist

AB6495

This class will present how real-life architectural projects have used the Dynamo visual programming extension to optimize complex design problems within Revit software. The class will give an introduction to optimization and rationalization algorithms, and the attendees will use the Dynamo extension and Python programming language scripts to create façades that adapt to varying design requirements. Attendees will also learn how to generalize the workflow so to apply it to other design problems concerning rationalization and optimization.

Learning Objectives

At the end of this class, you will be able to:

- Understand new ways to approach complex design problems
- Understand the principles of the Dynamo extensions' visual programming interface
- Learn how to manage basics of Python scripting
- Learn how to set up a simple optimization algorithm for any specific problem using the Dynamo extension plus Revit software

About the Speaker

Michael is a British Architect and University Lecturer. A graduate of The University of Nottingham, he has 12 years of experience with BIM software. He is currently an Associate at Flanagan Lawrence in London, UK and leads their BIM team. During his career, Michael’s expertise in 3D design has seen him work on schemes ranging in size from temporary installations to a 13km^2 Masterplan. Michael also leads a design studio at Leicester De Montfort University.

Mikehudsonarchi@gmail.com

www.uk.linkedin.com/in/mikehudsonarchi

@MikeHudsonArchi

About the Co-speaker

Andrea Vannini is an expert in computational design. He received his MArch in Rome (Italy) at La Sapienza University and his MSc in Adaptive Architecture and Computation at University College London. His thesis was in the field of parametric modelling and optimization using artificial intelligence algorithms. He worked at Flanagan Lawrence as an architect, implementing BIM and parametric design in many projects, working together with the BIM manager.

andrea.vannini.arc@gmail.com

http://www.linkedin.com/pub/andrea-vannini/32/3a8/633

Introduction

Why Dynamo is a Big Deal for Everyone

For many years the British architectural scene has had two personalities. There was the sculptural poetic and academic scene, which focused on the iconic one-off buildings and there was everyone else. Advances in computer aided design tools only widened the visual disparity between these two groups. While one group was quick to test and push the latest technologies to create ever more audacious forms, the majority of the profession did the same thing that they always had, but in CAD. So although programs like Grasshopper (GH) and Generative Components (GC) enabled designers to create improbable geometries, they often still needed 5 different software packages just to import and export models. With such as disjointed process, components often didn't fit without the assistance of a hammer by the time they had reached site. Conversely early BIM software was focused towards the commercial sphere of construction. Essentially, it was better if it was square. Autodesk's visual programming tool – Dynamo bridges the gap between these two disparate ideologies.

In 2007 the global recession had a profound effect on the construction industry in the UK. The projects that weren't cancelled needed to be 15 or 20% more cost efficient. BIM – although not by any means new – was seen as a mechanism for these savings to be made. The UK government mandated that by 2016, all publicly funded projects would need to be produced in a collaborative (Level 2) BIM way. Essentially the biggest client in the UK still wanted it to look great, but they didn't want to pay for it anymore. This posed a problem for the entirety of the architectural profession; how can we deliver a standardized set of information that doesn't limit us? We at Flanagan Lawrence believe that Dynamo is the solution.

This class shall discuss how Dynamo has been effectively used in our practice to enhance the design process. We shall present a variety of real design problems and explain how solutions were developed using Dynamo. We will also explain the principles of Pythonscript as an important tool to enhance the capabilities of the Dynamo tools.

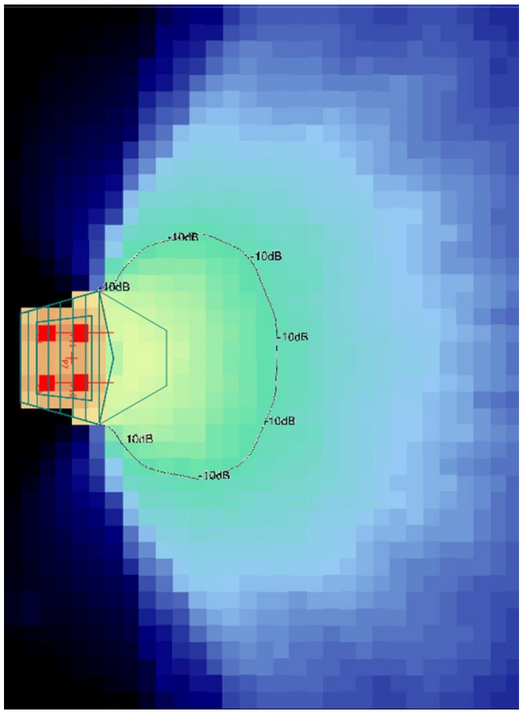
How do can we approach complex design problems?

Flanagan Lawrence

Flanagan Lawrence is an award-winning, design-led architectural practice based in London, UK. The practice has an impressive collective expertise across a broad range of sectors and building typology, including large-scale commercial projects and high-end residential schemes, as well as cultural, hotel and leisure, education, infrastructure, logistics, business parks and major master planning projects both in the UK and internationally.

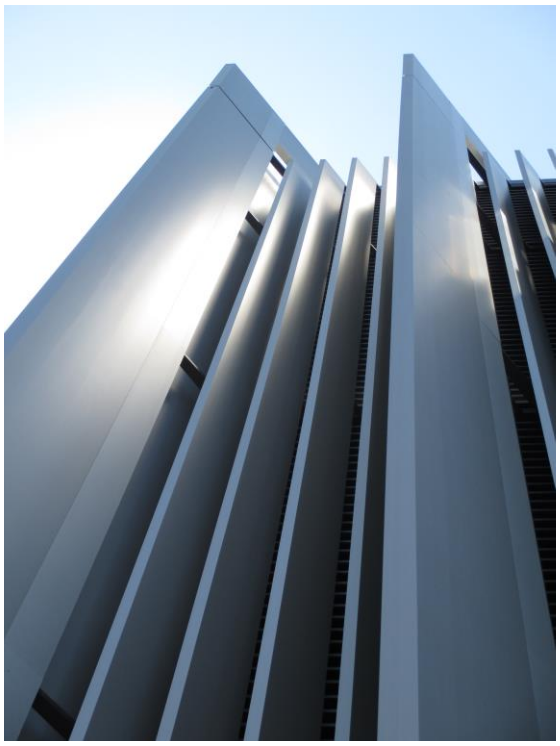
We strive to apply BIM processes to all of our projects; using Autodesk Revit as our primary authoring platform. However no two projects are ever the same, so often we have had to use multiple platforms to develop, coordinate, publish and construct our designs. Below are some examples of the projects which we had documented in Revit, but wanted to have a more integrated workflow.

Soundforms



Challenges Acoustic optimized geometry and digital fabrication

New Fetter Lane



New Fetter Lane – Complex geometry and collaborative workflows

Grosvenor Crescent

Large datasets (including point clouds) and existing structures

Why do Architects need coding skills?

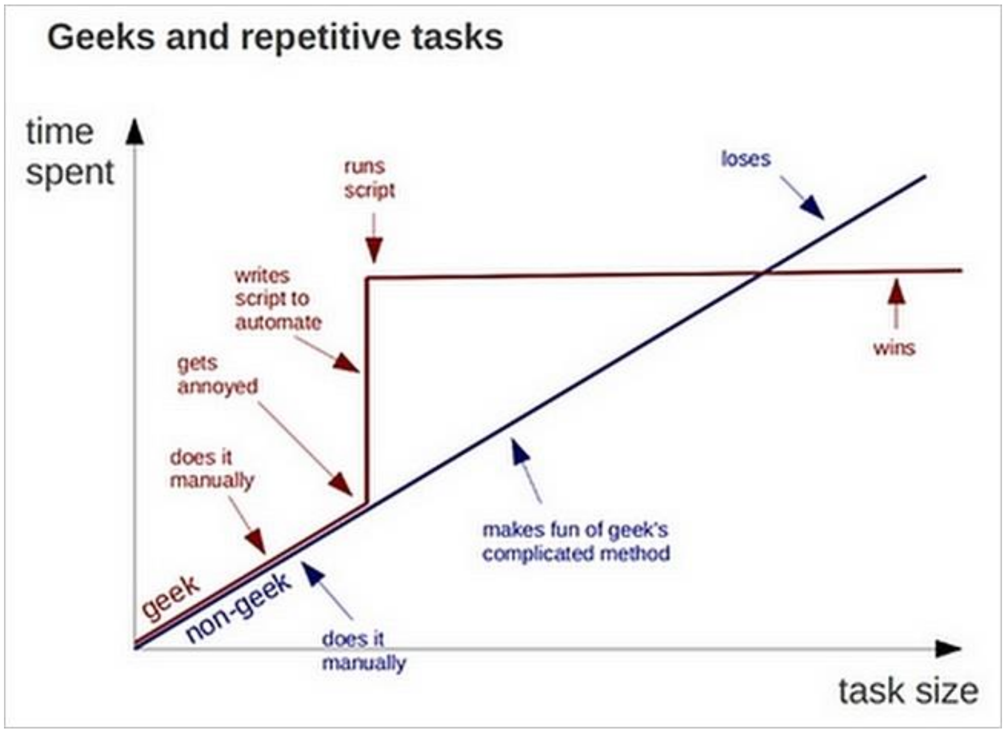


Image by Bruno 'btco' Olivera, Google

- We do lots of repetitive tasks
- Inputting and controlling data
- Design Optimization
- Internet of everything
- Crowd sourced design

A Closer look at Dynamo

Getting Started

How does Visual Programming Language (VPL) work?

Nodes

Connections

Lists

Why is it better than Grasshopper (GH)?

Stand-alone – And free so anyone can have a go without it needing to purchase expensive host.

It looks great - Everyone in our office has been really impressed with the stability of 0.7 and general tidiness of the UI.

Code blocks – Designscript - Find examples in CodeBlocksForDummies.dyn.

ASM kernel

The two way street – genuine BIM - GUID integrity

It is much easier to include external libraries such as .dll (windows extension) with the zero-touch plugin. IN GH everything needs to be 'wrapped' within the definition of a node class

Side-by-side comparison - acoustic shells

Dynamo way = Revit + Dynamo

GH Way = Revit > Humming Bird > XLS > Grasshopper (in Rhino) > XLS > Whitefeet > Revit

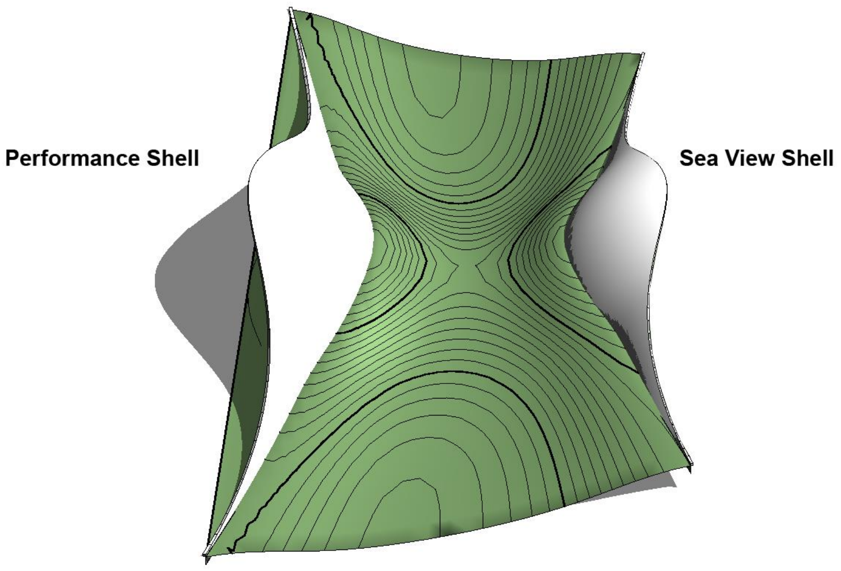
Lastly it feels good to be part of something BIG. It's great that Autodesk continue to allow this to be an open source project. It makes us feel like the work we are doing is for the good of the design community

It's worth noting that Rhynamo is also being developed by Case at the moment

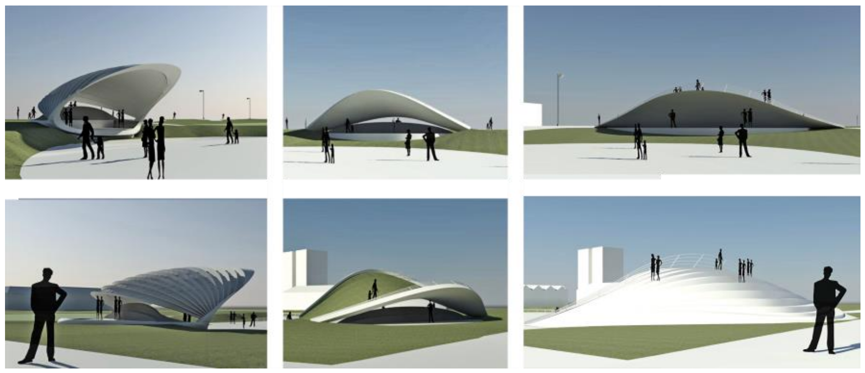
Acoustic Optimization

In 2013 Flanagan Lawrence won a competition to reimagine a 19th Century bandstand for the seaside town of Littlehampton, UK. The budget was £100,000 so you would be forgiven for thinking that this was not an ideal BIM project.

The brief was to interact with the main promenade but also provide a stage which would not need any amplification. Our expertise of acoustic architecture made it apparent that the best approach was to provide not one but two enclosures. A smaller shell would face the sea for seating, while a larger shell would face away from the noise of the sea for the performances.

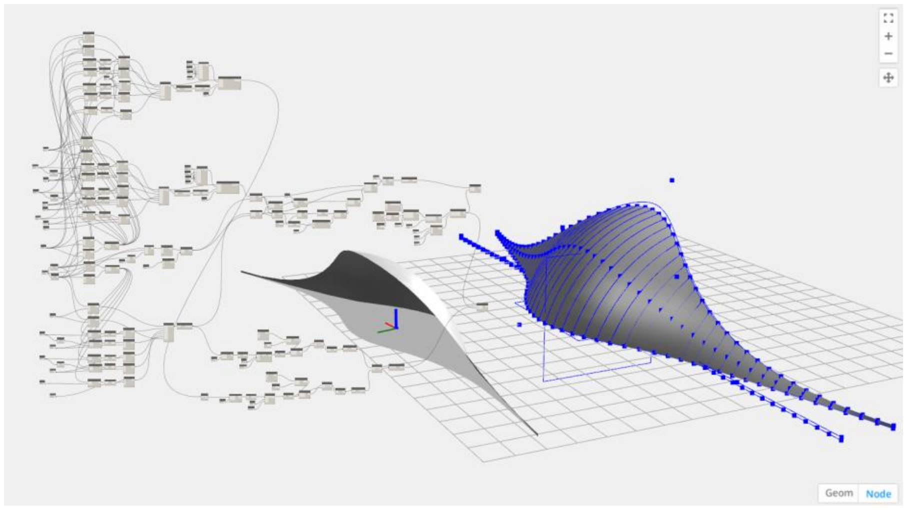


The local council loved the concept and we won the competition. However we also created a problem; proposing two structures effectively halved the already tight budget. Recognizing that from the outset that we would most likely need to use the absolute minimum material we began to explore forms. In previous projects we had wasted significant amounts of time on repetitive import/export procedures from GH to Revit. For this project to profitable for us, we had to use only Dynamo, which at the time was version 0.6

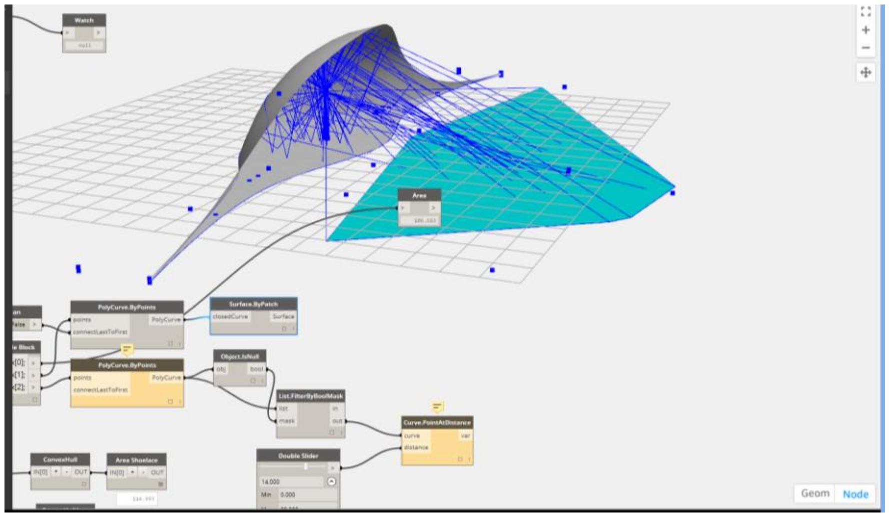


We went through many options with the premise that the forms would be a lightweight timber structure. However the client was concerned about vandalism, and decided that we should use concrete. Unsure if such a design was even possible we started to explore the stresses that might be within a half shell with a structural engineer. We also exported our model as an FBX for our in-house CGI artist.

Once we were confident that the structure was achievable we rebuilt the shells to the engineer's geometry.

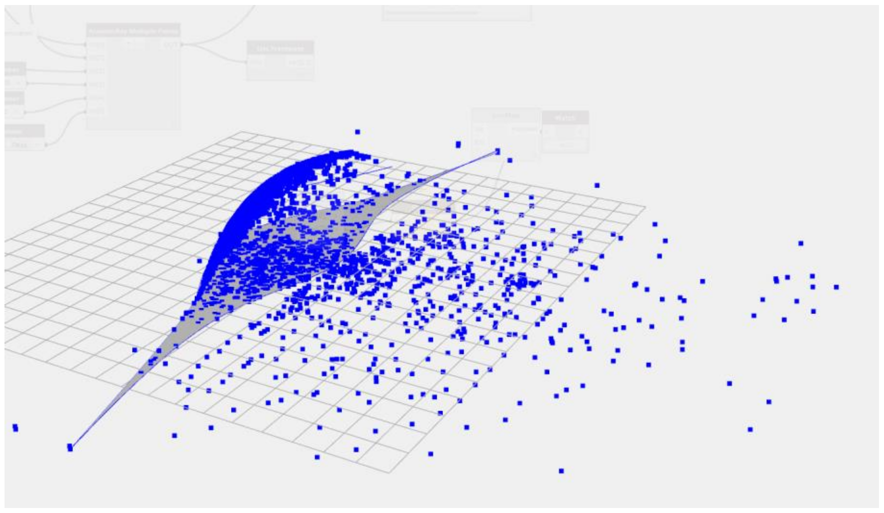


First we rebuilt the shell to much higher accuracy as we knew that that minor variations could have a huge effect on the acoustic performance. Initially we looked to optimize the form for a single performer on the center of the stage.

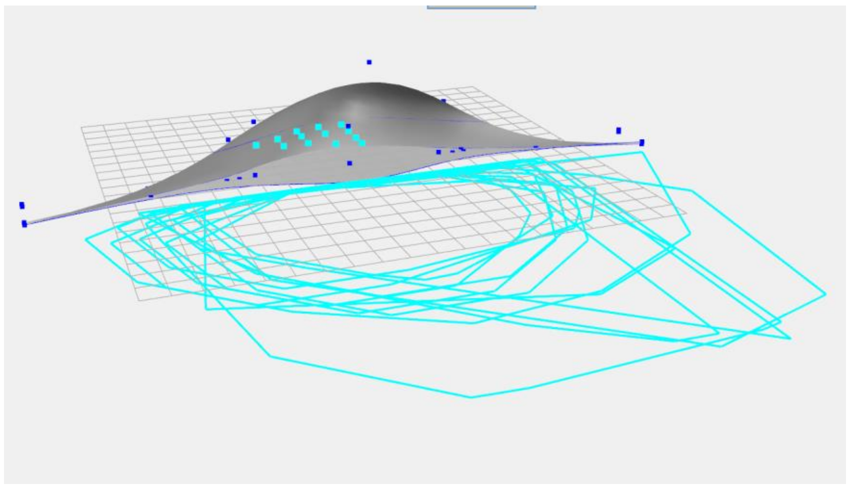


We used the ray trace tool and changed its setting so that it particle acted like sound to allow us to understand the how the shell performed. The light blue area represented the area which would receive a good sound without too large an interval between direct and bounced (reverb) sound

We then looked to optimize the form for multiple performers. As the shell was mirrored, we only needed to consider performers on half the stage.

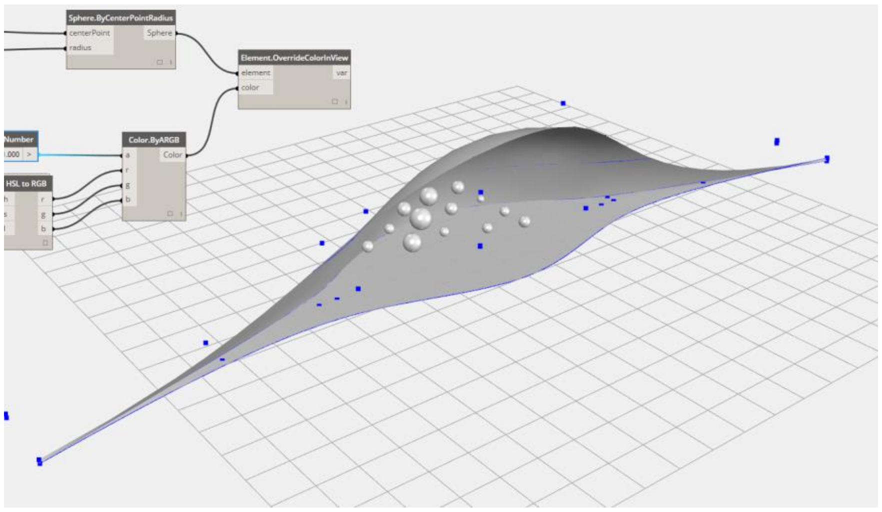


As you can see the volume of das that we were starting to produce was making it hard for us to analyze

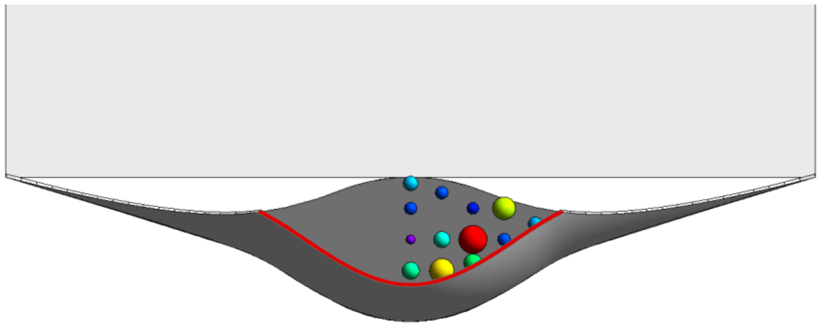


The light blue area represented the area which would receive a good sound.

We continued to tweak the form to allow it to cater for a variety of performance sizes

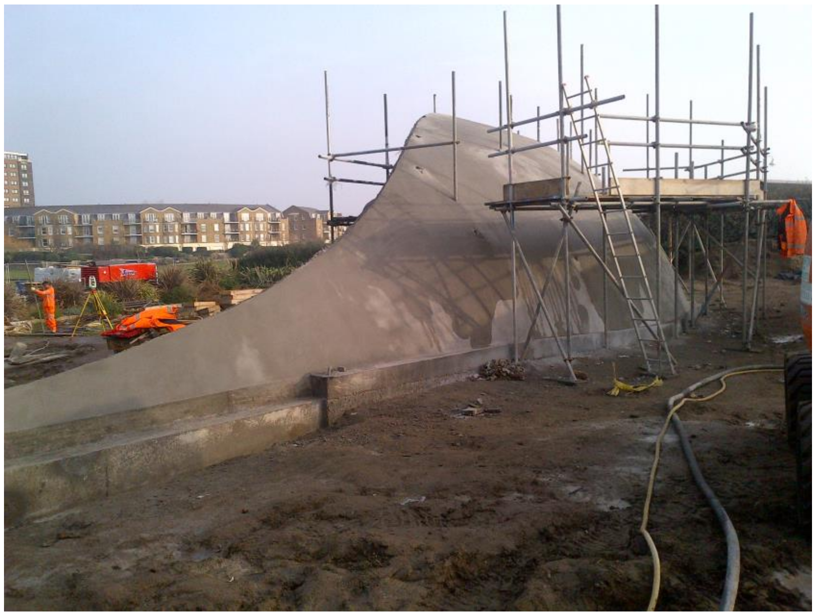


Eventually be found a compromise that created hot spots for acoustic performers (large spheres) and quitter locations (small spheres) for louder instruments. The data was issued to the client as color coded guidance diagrams to help them arrange performances.

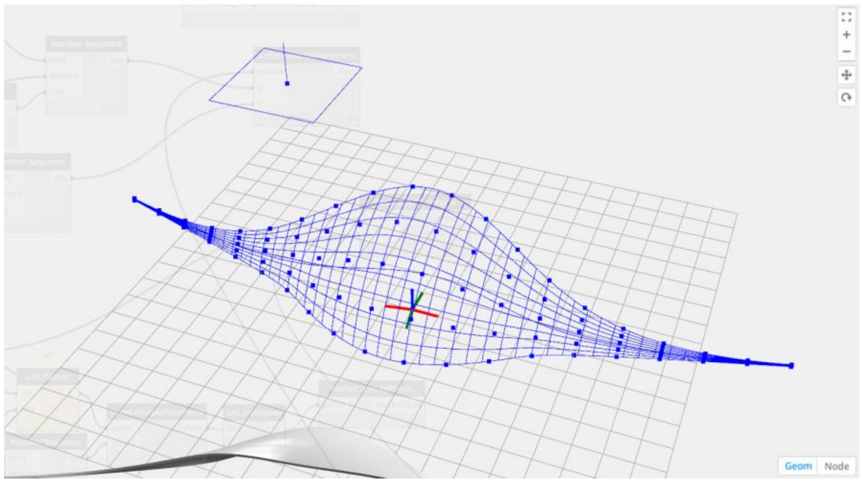


Fabrication example

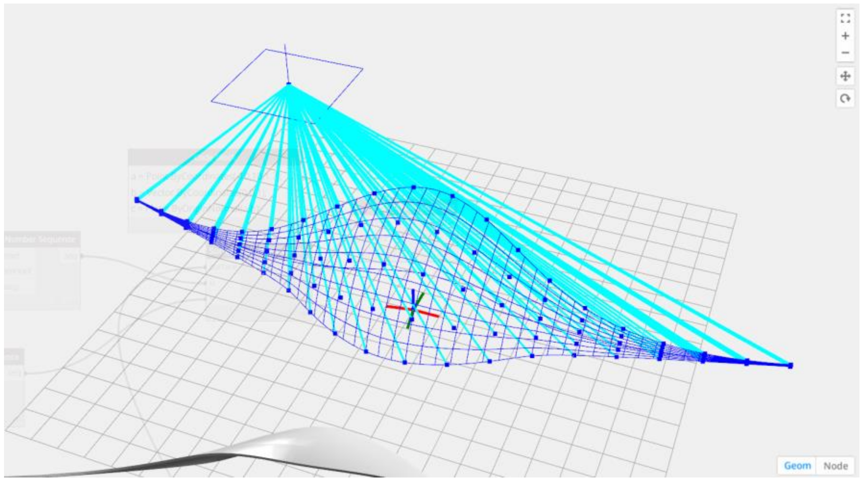
Accurately setting out a double-curved sprayed concrete structure was the perfect example of how Dynamo helped us to combine a quite inaccurate sprayed concrete technique with the very tight tolerances required for acoustics ,.



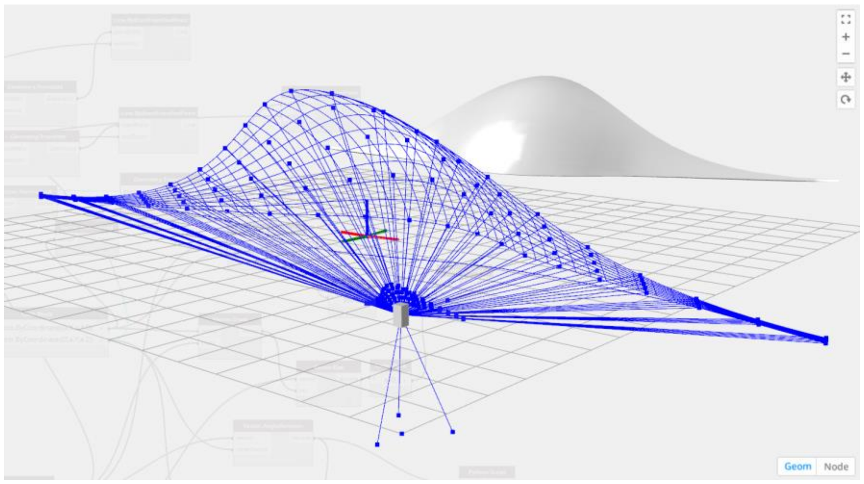
We devised a method which aligned the intersections of our temporary scaffold structure with a laser. Looking at the inner surface as an example we first agreed the location of a laser pointer which would be used to set out the surface.



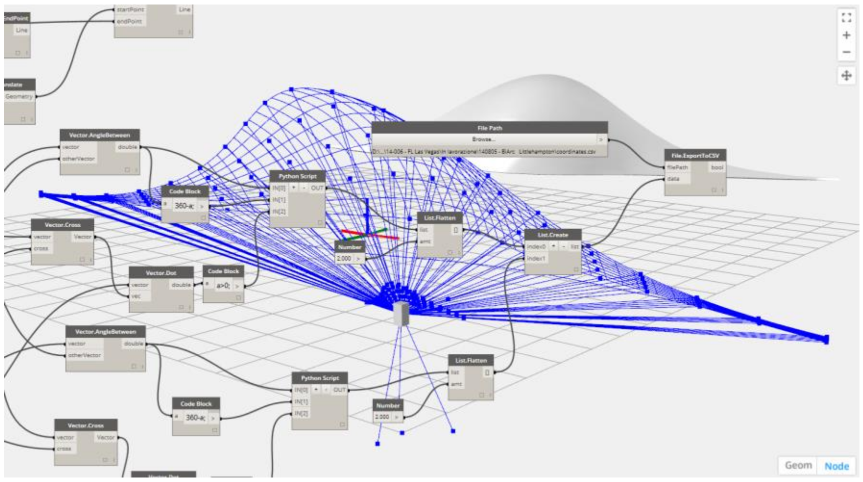
Finding an appropriate location for the laser pointer was complicated, as we needed to find a location which was least likely to be obstructed by the scaffold guides. We reused the ray trace tool to establish a location which could 'see' the entire surface.



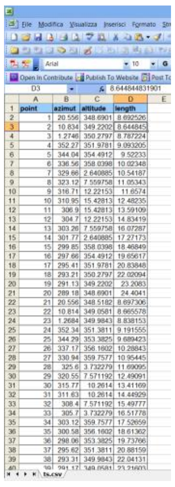
We then realized that the laser guides normally stand of tripods for stability, so we estimate it's height of above the datum. This was parameterized in case it needed to be changed



We then added PythonScript nodes to accurately output the vectors of each intersection for the laser guide operator

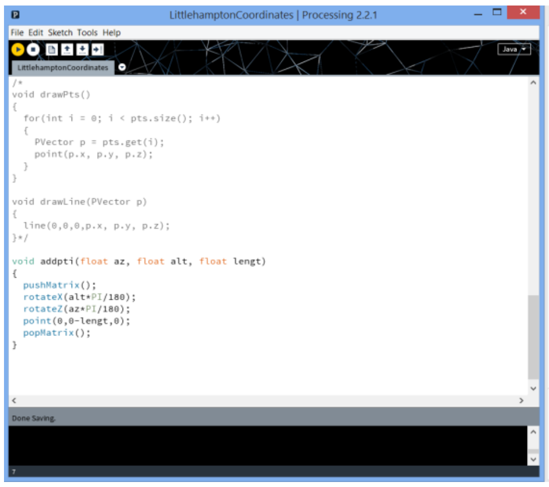


The data was exported in an excel chart (left). To validate the coordinates we imported the data into Processing, which created this array.


File Modifica Visualizza Inserisci Formato
Arial
Open In Contribute
Publish To Website
ts.csv

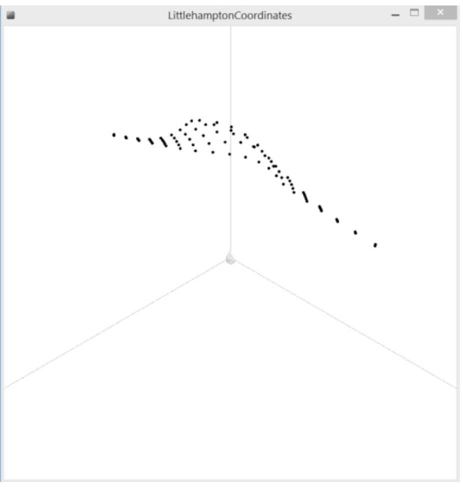
D3 8.644844831901

	A	B	C	D
1	point	azimut	altitude	length
2	1	20.556	348.6901	8.692526
3	2	10.834	349.2202	8.644845
4	3	1.2746	350.2797	8.787224
5	4	352.27	351.9781	9.093205
6	5	344.04	354.4912	9.52233
7	6	336.56	358.0398	10.02348
8	7	329.66	2.640885	10.54187
9	8	323.12	7.559758	11.05343
10	9	316.71	12.22153	11.6574
11	10	310.95	15.42813	12.48235
12	11	306.9	15.42813	13.59109
13	12	304.7	12.22153	14.83419
14	13	303.26	7.559758	16.07287
15	14	301.77	2.640885	17.27173
16	15	299.85	358.0398	18.46849
17	16	297.66	354.4912	19.65617
18	17	295.41	351.9781	20.83848
19	18	293.21	350.2797	22.02094
20	19	291.13	349.2202	23.2083
21	20	289.18	348.6901	24.4041
22	21	20.556	348.5182	8.697306
23	22	10.814	349.0581	8.665578
24	23	1.2684	349.9843	8.838153
25	24	352.34	351.3811	9.191555
26	25	344.29	353.3825	9.689423
27	26	337.17	356.1602	10.28843
28	27	330.94	359.7577	10.95445
29	28	325.6	3.732279	11.69095
30	29	320.55	7.571192	12.49091
31	30	315.77	10.2614	13.41169
32	31	311.63	10.2614	14.44929
33	32	308.4	7.571192	15.49777
34	33	305.7	3.732279	16.51778
35	34	303.12	359.7577	17.52659
36	35	300.58	356.1602	18.61362
37	36	298.06	353.3825	19.73766
38	37	295.62	351.3811	20.88159
39	38	293.31	349.9843	22.04131
40	39	291.17	349.0581	23.21603


LittlehamptonCoordinates | Processing 2.2.1
File Edit Sketch Tools Help
Java
LittlehamptonCoordinates

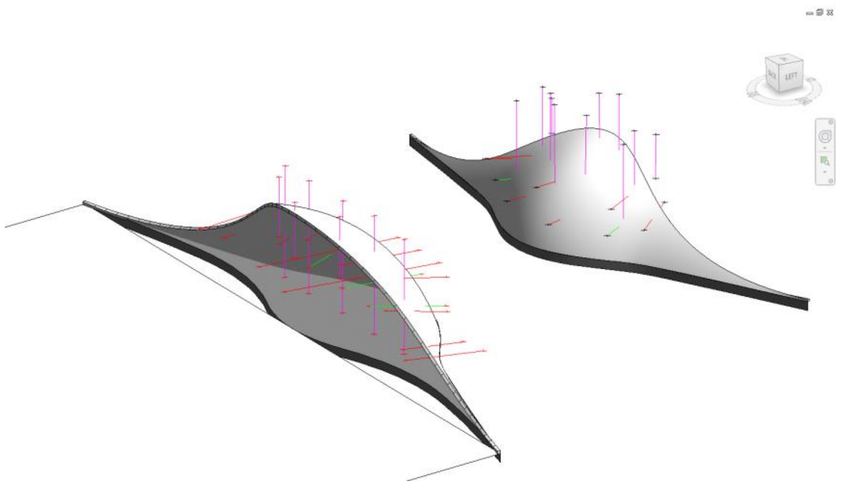
```
/*
void drawPts()
{
  for(int i = 0; i < pts.size(); i++)
  {
    PVector p = pts.get(i);
    point(p.x, p.y, p.z);
  }
}

void drawLine(PVector p)
{
  line(0,0,0,p.x, p.y, p.z);
}*/

void addpti(float az, float alt, float lengt)
{
  pushMatrix();
  rotateX(alt*PI/180);
  rotateZ(az*PI/180);
  point(0,0-lengt,0);
  popMatrix();
}
```

Done Saving.

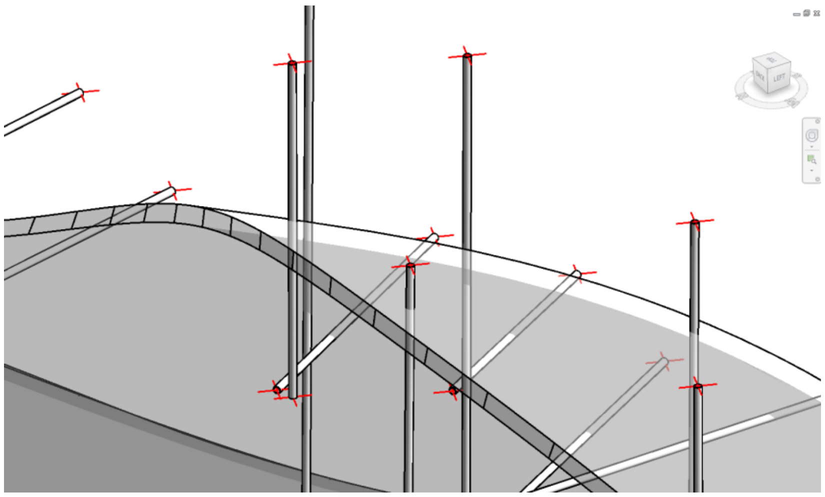


In an ideal world, we would have then used our rational setting out but in reality the steel scaffolding had already been erected by the time we had reached site.

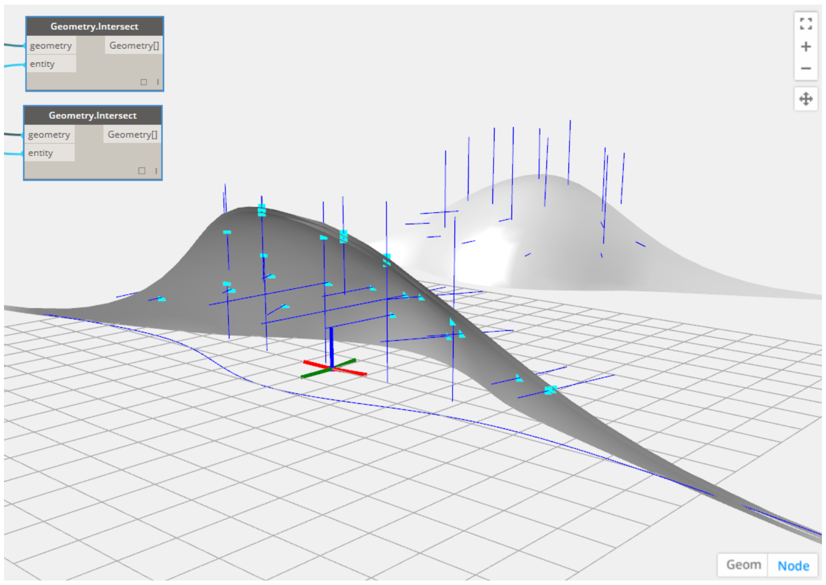
This was not a problem as we quickly surveyed the location of east bar and then rebuilt the vectors of each element inside Dynamo.



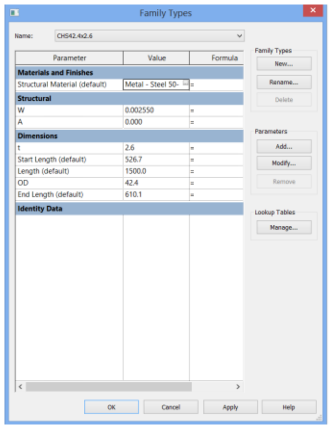
As you can see from the close-up, none of the steel poles were arranged to any particular alignment.



As discussed earlier, the inner and outer faces of the shells where not perpendicular, so added we nodes to identify the points at which the steel poles intersected the two faces.

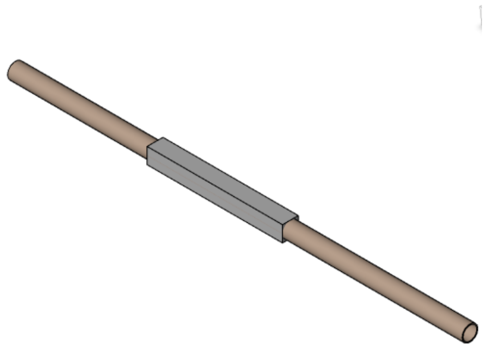


We then created a simple adaptive family, which would more accurately represent the points at which the site workers would need to mark the steel poles

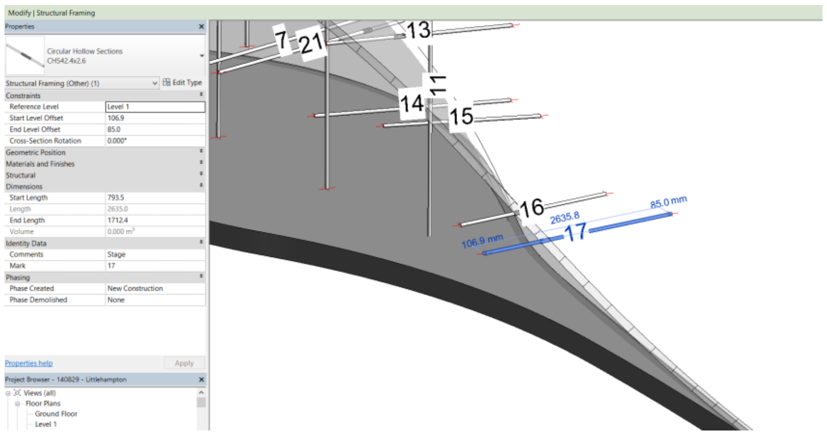


Name: CHS42.4x2.6

Parameter	Value	Formula
Materials and Finishes		
Structural Material (default)	Metal - Steel 50-	=
Structural		
W	0.002550	=
A	0.000	=
Dimensions		
t	2.6	=
Start Length (default)	526.7	=
Length (default)	1500.0	=
OD	42.4	=
End Length (default)	610.1	=
Identity Data		



The square central sections of the pole are driven by point elements with cord length instance parameters and each pole numbered.



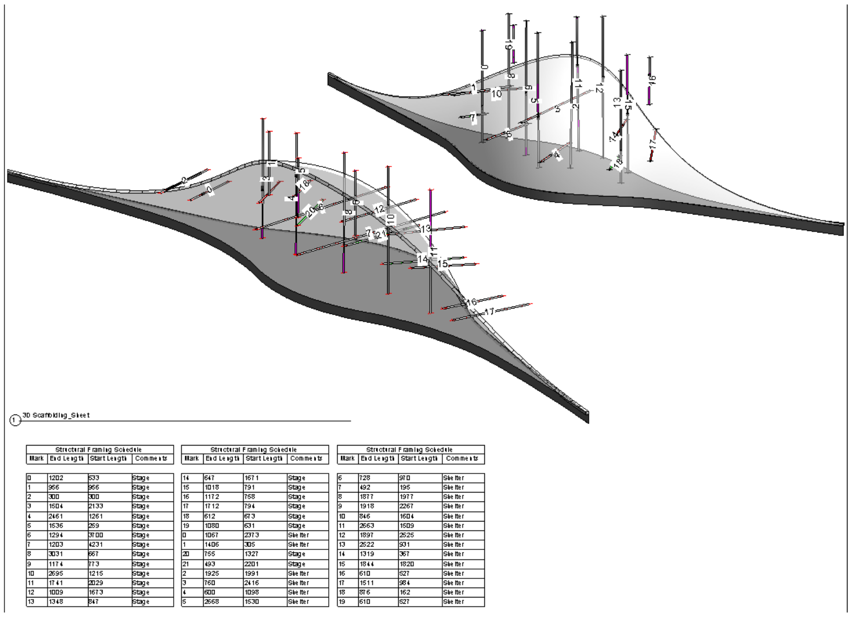
Circular Hollow Sections
CHS42.4x2.6

Structural Framing (Other) (1)

Constraints	
Reference Level	Level 1
Start Level Offset	106.9
End Level Offset	85.0
Cross-Section Rotation	0.000°
Geometric Position	
Materials and Finishes	
Structural	
Dimensions	
Start Length	793.5
Length	2635.0
End Length	1712.4
Volume	0.000 m³
Identity Data	
Comments	Stage
Mark	17
Phasing	
Phase Created	New Construction
Phase Demolished	None

Project Browser - 140829 - Littlehampton
- Views (all)
  - Floor Plans
    - Ground Floor
    - Level 1

Each instance was scheduled and drawings were issued to the site workers

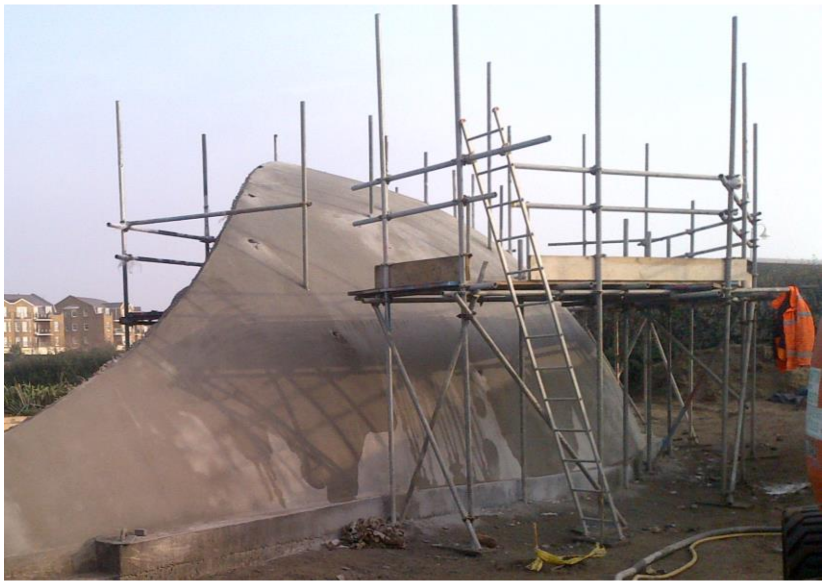


Structural Framing Schedule			
Mark	End Length	Start Length	Comments
0	1202	533	Stage
1	956	956	Stage
2	300	300	Stage
3	1504	2133	Stage
4	2461	1261	Stage
5	1536	259	Stage
6	1294	3700	Stage
7	1203	4231	Stage
8	3031	667	Stage
9	1174	773	Stage
10	2695	1215	Stage
11	1741	2029	Stage
12	1009	1673	Stage
13	1348	847	Stage

Structural Framing Schedule			
Mark	End Length	Start Length	Comments
14	647	1671	Stage
15	1018	791	Stage
16	1172	758	Stage
17	1712	794	Stage
18	612	673	Stage
19	1080	631	Stage
0	1067	2373	Shelter
1	1406	305	Shelter
20	755	1327	Stage
21	493	2201	Stage
2	1925	1991	Shelter
3	760	2416	Shelter
4	600	1098	Shelter
5	2668	1530	Shelter

Structural Framing Schedule			
Mark	End Length	Start Length	Comments
6	728	970	Shelter
7	492	195	Shelter
8	1877	1977	Shelter
9	1918	2267	Shelter
10	846	1604	Shelter
11	2663	1509	Shelter
12	1897	2525	Shelter
13	2522	931	Shelter
14	1319	367	Shelter
15	1844	1820	Shelter
16	610	527	Shelter
17	1511	984	Shelter
18	876	162	Shelter
19	610	527	Shelter

Image of the temporary steel poles penetrating the concrete form.



The holes were filled and the surface was smoothed

The stage opened for use in July 2014

Python Scripting

If Dynamo is so great, why do we need it?

Visual programming is good but more experienced programmers can build more effective definitions if they can also use text-based coding languages. As dynamo is relatively new, we found that there were situations when we had to code some extra functionality into the interface. Examples of these are:

- Customized nodes
- Parallel Processing
- Dynamic analysis
- External Datasets

So why did we use Python?

Firstly a node was available so we thought that we should try it!

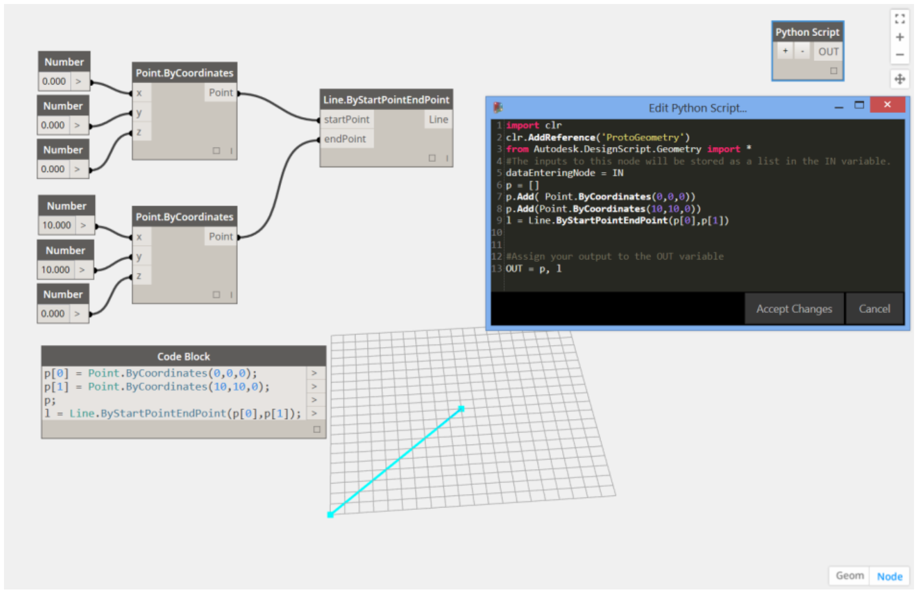
Python was originally created in 1991 so is very stable. Version 3.0 was released in 2008 and is backwards compatible so there is quite a substantial library available.

The python UI looks quite similar to Visual basic (.Net 4.0)

The syntax is very concise

Python Basics

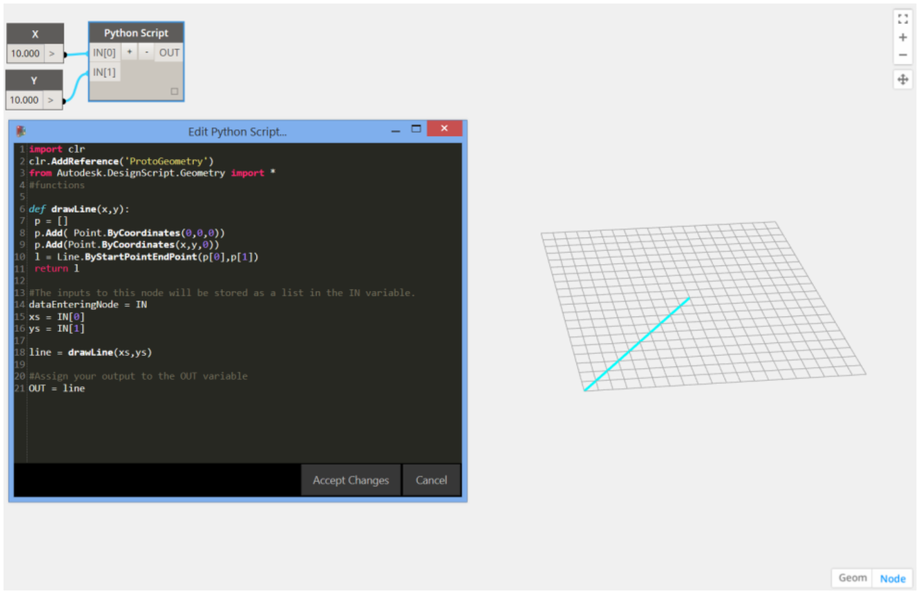
The first thing to understand is that using Python is not very difficult if you have already been using code blocks in Dynamo. The syntax of Designscript and Pythonscript are very similar. To demonstrate this we will use a simple example of drawing a line between two points.



```
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
#The inputs to this node will be stored as a list in the IN variable.
dataEnteringNode = IN
p = []
p.Add( Point.ByCoordinates(0,0,0))
p.Add(Point.ByCoordinates(10,10,0))
l = Line.ByStartPointEndPoint(p[0],p[1])


#Assign your output to the OUT variable
OUT = p, l
```

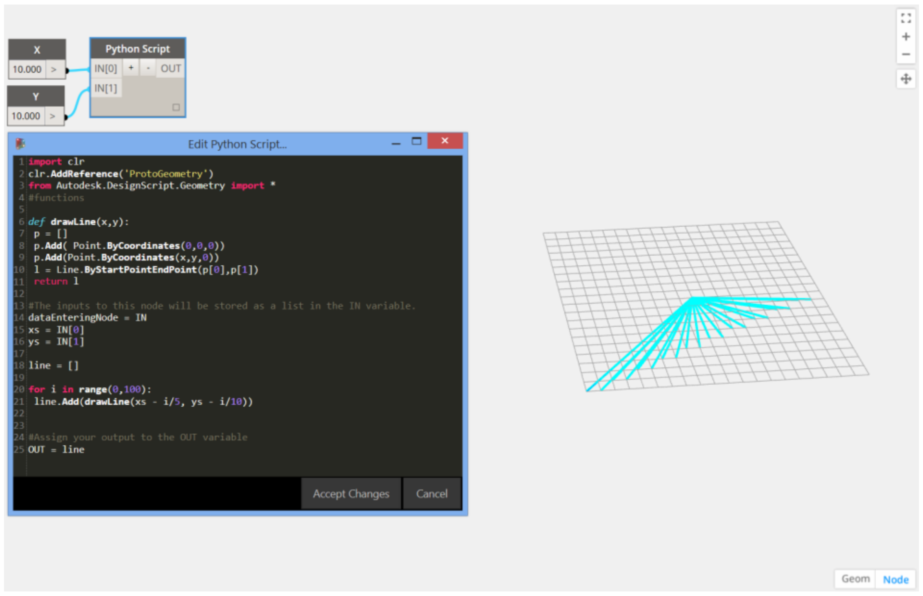
```
p[0] = Point.ByCoordinates(0,0,0);
p[1] = Point.ByCoordinates(10,10,0);
p;
l = Line.ByStartPointEndPoint(p[0],p[1]);
```

Sometimes when the script gets complicated it is necessary to build function and classes. Here the same example built with a function. In this scenario the function return values such as lines and points. I have left some parameters outside the function (the coordinates of the second point) but this can be any parameter we need.



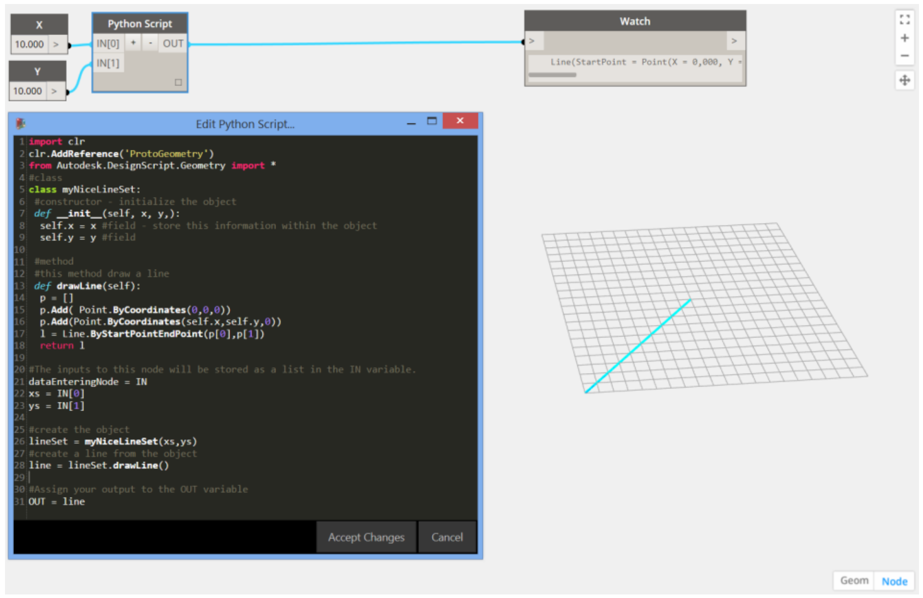
```
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
#functions

def drawLine(x,y):
 p = []
 p.Add( Point.ByCoordinates(0,0,0))
 p.Add(Point.ByCoordinates(x,y,0))
 l = Line.ByStartPointEndPoint(p[0],p[1])
 return l

#The inputs to this node will be stored as a list in the IN variable.
dataEnteringNode = IN
xs = IN[0]
ys = IN[1]

line = drawLine(xs,ys)

#Assign your output to the OUT variable
OUT = line
```

Functions are useful, for example when it is necessary to do repetitive tasks.



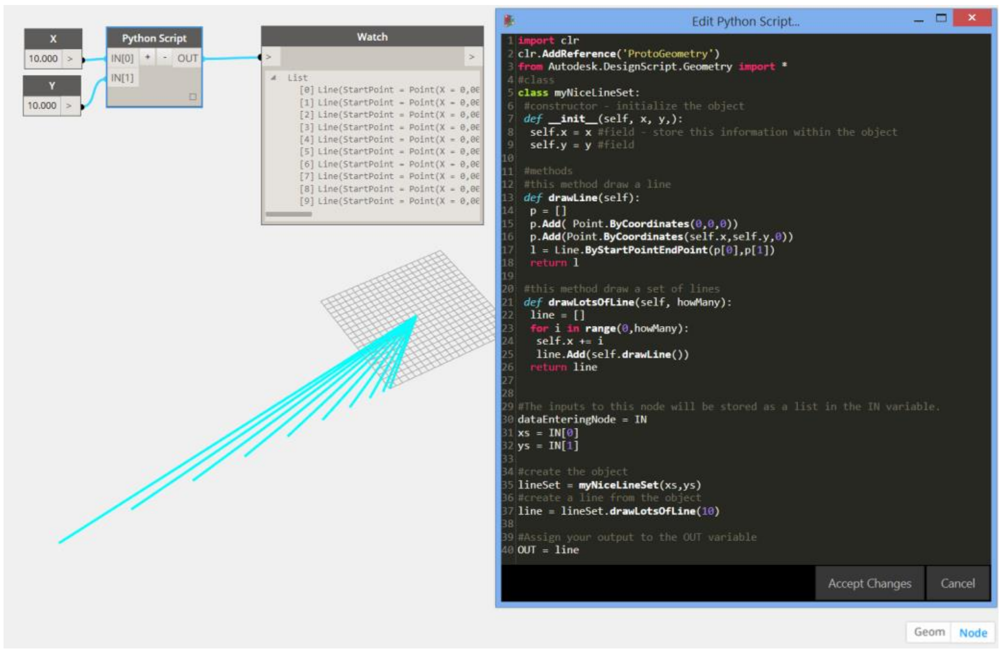
```python
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
#functions

def drawLine(x,y):
 p = []
 p.Add( Point.ByCoordinates(0,0,0))
 p.Add(Point.ByCoordinates(x,y,0))
 l = Line.ByStartPointEndPoint(p[0],p[1])
 return l

#The inputs to this node will be stored as a list in the IN variable.
dataEnteringNode = IN
xs = IN[0]
ys = IN[1]

line = []

for i in range(0,100):
 line.Add(drawLine(xs - i/5, ys - i/10))


#Assign your output to the OUT variable
OUT = line
```

Another important element is the classes, which allows the coder to create object which have many functions and can provide many methods. The information given to a class is called fields, the functions are the constructor, the one which initialize the class while the others are normally called methods.



```python
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
#class
class myNiceLineSet:
 #constructor - initialize the object
 def __init__(self, x, y,):
  self.x = x #field - store this information within the object
  self.y = y #field

 #method
 #this method draw a line
 def drawLine(self):
  p = []
  p.Add( Point.ByCoordinates(0,0,0))
  p.Add(Point.ByCoordinates(self.x,self.y,0))
  l = Line.ByStartPointEndPoint(p[0],p[1])
  return l

#The inputs to this node will be stored as a list in the IN variable.
dataEnteringNode = IN
xs = IN[0]
ys = IN[1]

#create the object
lineSet = myNiceLineSet(xs,ys)
#create a line from the object
line = lineSet.drawLine()

#Assign your output to the OUT variable
OUT = line
```

As you can see when the class is generated, the actual code in python script (between input and output) is very concise. Furthermore, this can be easily reused in other scripts by copying and pasting the class code without making changes. In Revit terms, we can say that the first case (hardcode) is the "model in place" object, the function is a "model in place" with some parameters while a class is a family.



```
List
  [0] Line(StartPoint = Point(X = 0,0
  [1] Line(StartPoint = Point(X = 0,0
  [2] Line(StartPoint = Point(X = 0,0
  [3] Line(StartPoint = Point(X = 0,0
  [4] Line(StartPoint = Point(X = 0,0
  [5] Line(StartPoint = Point(X = 0,0
  [6] Line(StartPoint = Point(X = 0,0
  [7] Line(StartPoint = Point(X = 0,0
  [8] Line(StartPoint = Point(X = 0,0
  [9] Line(StartPoint = Point(X = 0,0
```

```python
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
#class
class myNiceLineSet:
 #constructor - initialize the object
 def __init__(self, x, y,):
  self.x = x #field - store this information within the object
  self.y = y #field

 #methods
 #this method draw a line
 def drawLine(self):
  p = []
  p.Add( Point.ByCoordinates(0,0,0))
  p.Add(Point.ByCoordinates(self.x,self.y,0))
  l = Line.ByStartPointEndPoint(p[0],p[1])
  return l

 #this method draw a set of lines
 def drawLotsOfLine(self, howMany):
  line = []
  for i in range(0,howMany):
   self.x += i
   line.Add(self.drawLine())
  return line


#The inputs to this node will be stored as a list in the IN variable.
dataEnteringNode = IN
xs = IN[0]
ys = IN[1]

#create the object
lineSet = myNiceLineSet(xs,ys)
#create a line from the object
line = lineSet.drawLotsOfLine(10)

#Assign your output to the OUT variable
OUT = line
```

Gradient Descent/Hill Climbing

How do we optimize a general problem? There are an almost infinite number of optimization algorithms. One of the easiest to implement is the Hill-Climbing or Gradient Descent algorithm. Imagine yourself in the desert, blinded, and it is asked you to reach the top of a dune.

As you are blinded, there is a limitation on deciding the direction to the next highest dune. A process to do so might be:

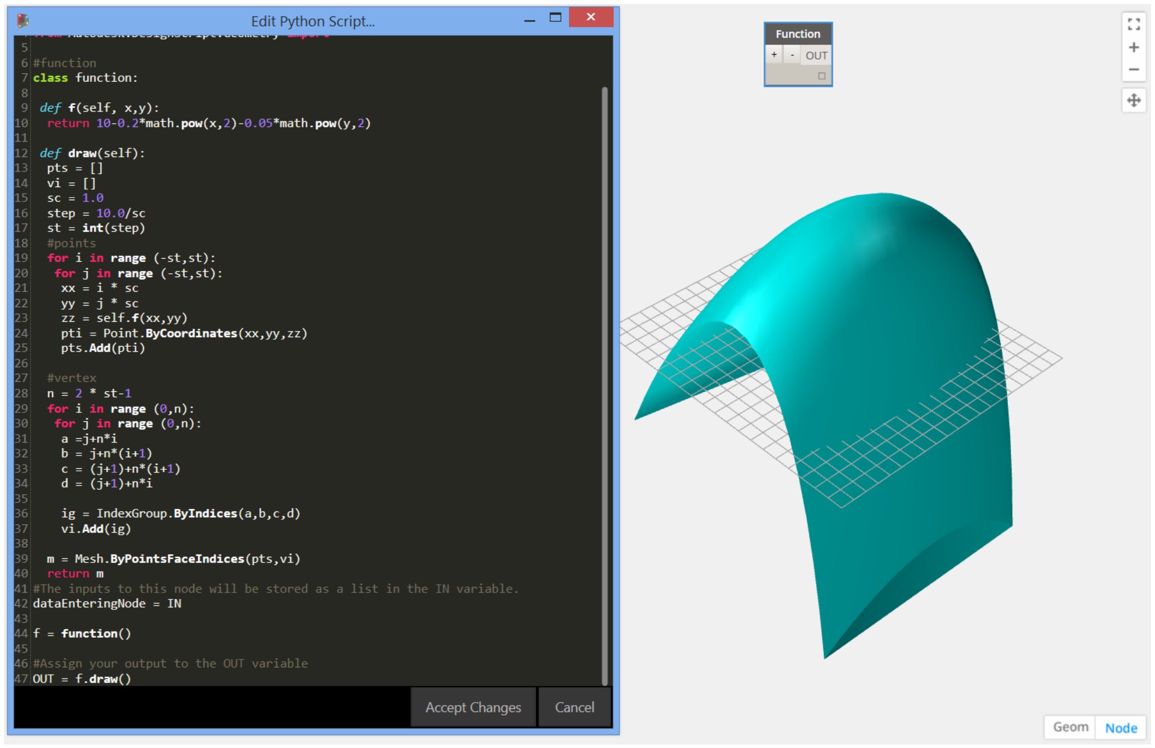
- Do a test step to one side (right of left)
    - Is it higher than the last step?
        - yes, move in this direction of a *portion of the step**
        - No, move in the opposite direction of a *portion of the step**
- Do a test step front or back
    - Is it higher than the last step ?
        - yes, move in this direction of a *portion of the step**
        - No, move in the opposite direction of a *portion of the step**
- Repeat

If you repeat this process enough times, you will reach the next highest dune. In mathematical terms, we are looking for the minimum or maximum of a mathematical function, trying to solve it numerically rather than analytically. This is because our function may be so complex that is impossible to solve it in analytical terms (for example a FEM model). Furthermore we have considered 2 parameters, x and y position; normally in our visual programming interface the parameters are more than 2, defining a higher dimensional space.

Let's implement this in Dynamo. If we generate a class containing a function which will be:

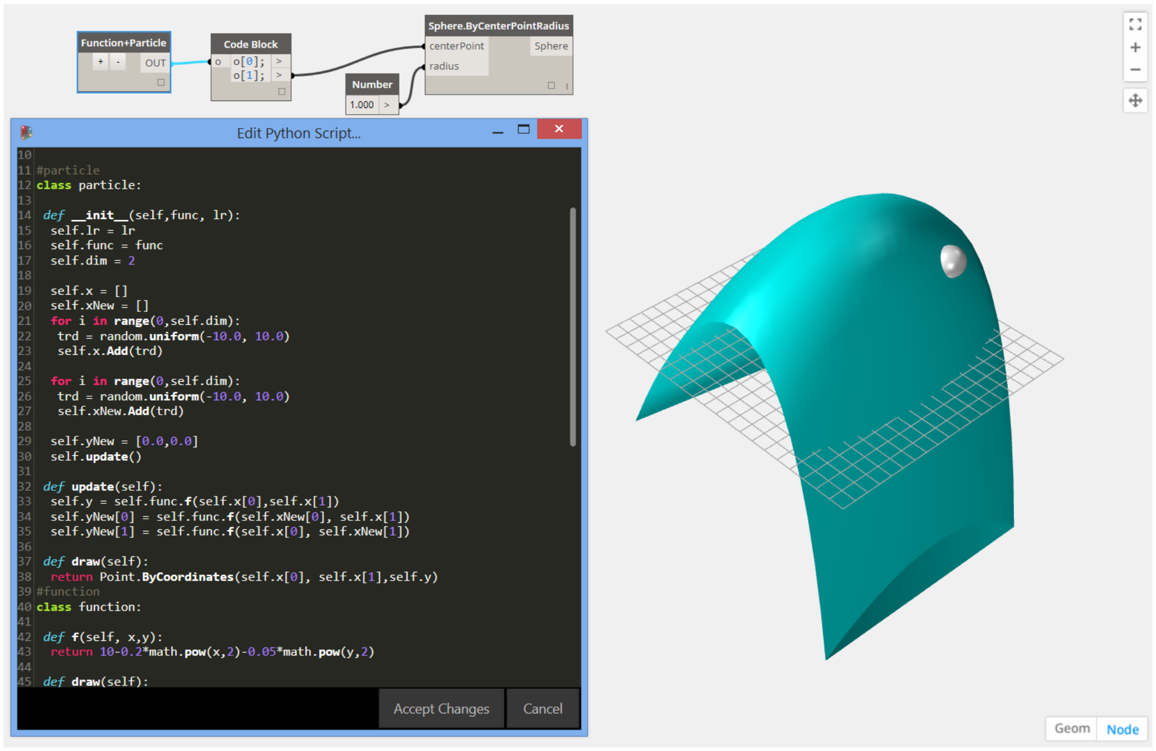
$$z = 10 - 0.2 * x^2 - 0.05 * y^2$$

Normally this function, which is called objective function or fitness landscape, is not known, but in this scenario is made explicit to understand the process. This can be more complex depending on the design.



```

#function
class function:

 def f(self, x,y):
  return 10-0.2*math.pow(x,2)-0.05*math.pow(y,2)

 def draw(self):
  pts = []
  vi = []
  sc = 1.0
  step = 10.0/sc
  st = int(step)
  #points
  for i in range (-st,st):
   for j in range (-st,st):
    xx = i * sc
    yy = j * sc
    zz = self.f(xx,yy)
    pti = Point.ByCoordinates(xx,yy,zz)
    pts.Add(pti)

  #vertex
  n = 2 * st-1
  for i in range (0,n):
   for j in range (0,n):
    a =j+n*i
    b = j+n*(i+1)
    c = (j+1)+n*(i+1)
    d = (j+1)+n*i

    ig = IndexGroup.ByIndices(a,b,c,d)
    vi.Add(ig)

  m = Mesh.ByPointsFaceIndices(pts,vi)
  return m
#The inputs to this node will be stored as a list in the IN variable.
dataEnteringNode = IN

f = function()

#Assign your output to the OUT variable
OUT = f.draw()
```

Now let's start climbing! First step is to generate yourself, which is done with the class Particle. This class defines the current position on the desert, which starts from a random point.

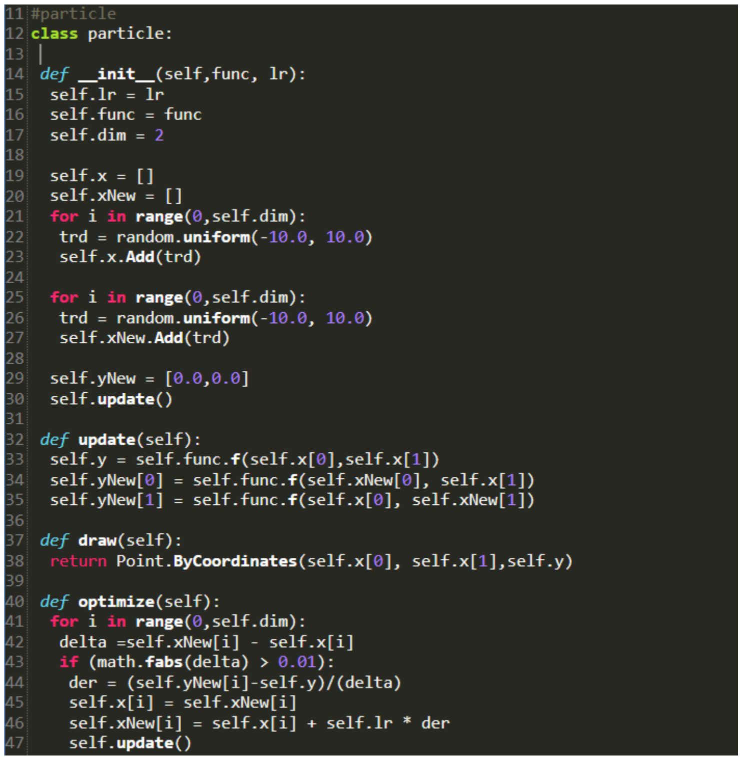


```
#particle
class particle:

 def __init__(self,func, lr):
  self.lr = lr
  self.func = func
  self.dim = 2

  self.x = []
  self.xNew = []
  for i in range(0,self.dim):
   trd = random.uniform(-10.0, 10.0)
   self.x.Add(trd)

  for i in range(0,self.dim):
   trd = random.uniform(-10.0, 10.0)
   self.xNew.Add(trd)

  self.yNew = [0.0,0.0]
  self.update()

 def update(self):
  self.y = self.func.f(self.x[0],self.x[1])
  self.yNew[0] = self.func.f(self.xNew[0], self.x[1])
  self.yNew[1] = self.func.f(self.x[0], self.xNew[1])

 def draw(self):
  return Point.ByCoordinates(self.x[0], self.x[1],self.y)
#function
class function:

 def f(self, x,y):
  return 10-0.2*math.pow(x,2)-0.05*math.pow(y,2)

 def draw(self):
```

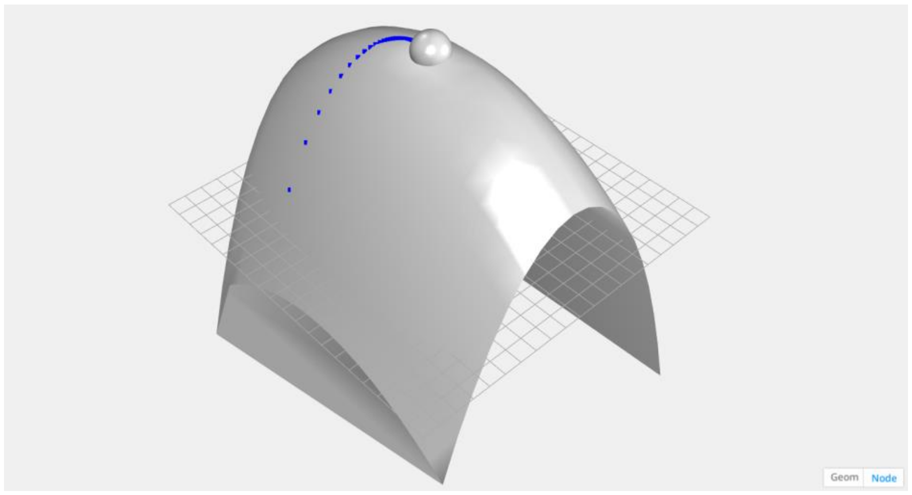
The class particle needs to be changed, first we need to store the previous and current position in the x[] and xNew [] arrays. Then we need to add a random position and a random initial step. The update method, the one which tells the z position of the particle depending on the function is the same while we need to add an optimize method. This involves some steps:

- For a given number of iterations:
  - calculate the length of the step done
  - if this is bigger than the tolerance of 0.01
    - calculate the first derivative (this is an improvement called Newton's Method. an actual step might be used as well).
    - update the previous step value with the new
    - update the new step value adding the derivative times the learning rate
    - update the y and yNew of the function

Another important parameter is the learning rate. This basically means that the step will not be used in full but by only a portion. It might be that the first step is so big that the process will not reach convergence.



```
#particle
class particle:
 
 def __init__(self,func, lr):
  self.lr = lr
  self.func = func
  self.dim = 2
 
  self.x = []
  self.xNew = []
  for i in range(0,self.dim):
   trd = random.uniform(-10.0, 10.0)
   self.x.Add(trd)
 
  for i in range(0,self.dim):
   trd = random.uniform(-10.0, 10.0)
   self.xNew.Add(trd)
 
  self.yNew = [0.0,0.0]
  self.update()
 
 def update(self):
  self.y = self.func.f(self.x[0],self.x[1])
  self.yNew[0] = self.func.f(self.xNew[0], self.x[1])
  self.yNew[1] = self.func.f(self.x[0], self.xNew[1])
 
 def draw(self):
  return Point.ByCoordinates(self.x[0], self.x[1],self.y)
 
 def optimize(self):
  for i in range(0,self.dim):
   delta =self.xNew[i] - self.x[i]
   if (math.fabs(delta) > 0.01):
    der = (self.yNew[i]-self.y)/(delta)
    self.x[i] = self.xNew[i]
    self.xNew[i] = self.x[i] + self.lr * der
    self.update()
```

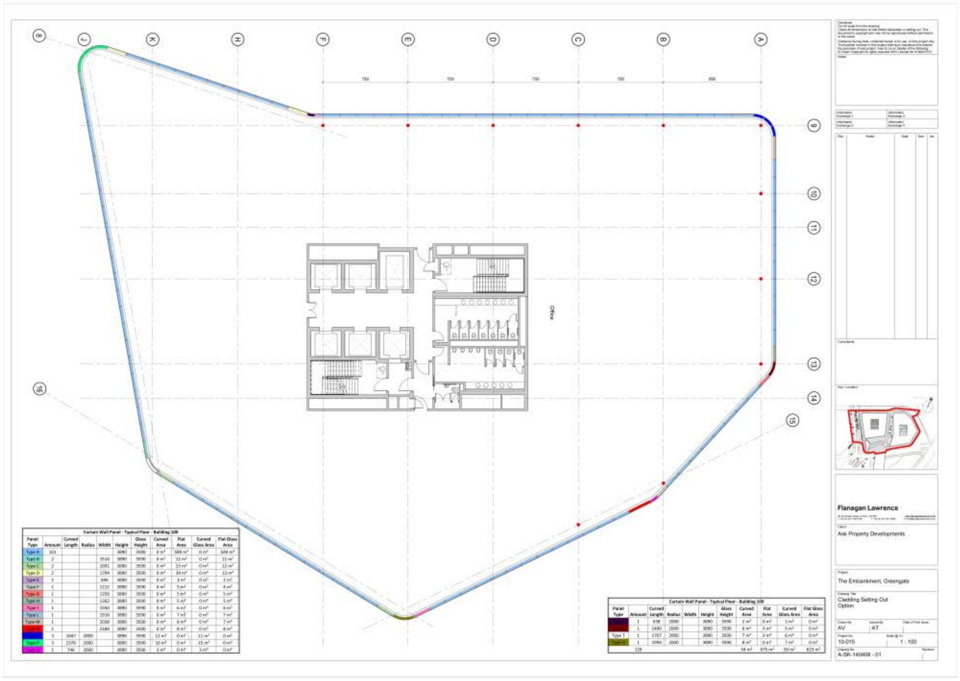


Simple optimization algorithm for any specific problem using the Dynamo extension plus Revit software

Façade optimization

101
100
101 EMBANKMENT
Q PARK
EMBANKMENT

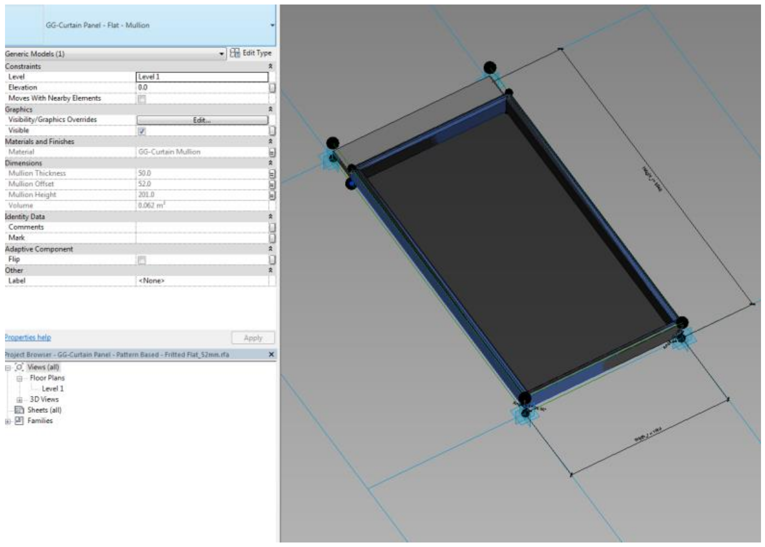
Building 100 was the second phase of a £60 million office development in Manchester, UK. The form of the two buildings evolved to maximize floor area and mitigate a windy microclimate. While building 101 has a fairly convention 4 sided plan, the combination of a listed heritage boundary and the wind modeling caused building 100 to have 7 facades.


Office
Flanagan Lawrence
Ask Property Developments
The Embankment, Greengate
Cladding Setting Out Option
AV
AT
10-015
1 : 100
A-SK-140408 - 01

Curtain Wall Panel - Typical Floor - Building 100

Panel Type	Amount	Curved Length	Radius	Width	Height	Glass Height	Curved Area	Flat Area	Curved Glass Area	Flat Glass Area
Type A	101				3880	3590	0 m²	588 m²	0 m²	544 m²
Type B	2			1514	3880	3590	0 m²	12 m²	0 m²	11 m²
Type C	2			1681	3880	3590	0 m²	13 m²	0 m²	12 m²
Type D	2			1784	3880	3590	0 m²	14 m²	0 m²	13 m²
Type E	1			696	3880	3590	0 m²	3 m²	0 m²	2 m²
Type F	1			1212	3880	3590	0 m²	5 m²	0 m²	4 m²
Type G	1			1256	3880	3590	0 m²	5 m²	0 m²	5 m²
Type H	1			1262	3880	3590	0 m²	5 m²	0 m²	5 m²
Type I	1			1560	3880	3590	0 m²	6 m²	0 m²	6 m²
Type L	1			1916	3880	3590	0 m²	7 m²	0 m²	7 m²
Type M	1			2038	3880	3590	0 m²	8 m²	0 m²	7 m²
Type N	1			2184	3880	3590	0 m²	8 m²	0 m²	8 m²
Type O	3	1047	2000		3880	3590	12 m²	0 m²	11 m²	0 m²
Type P	3	1376	2000		3880	3590	16 m²	0 m²	15 m²	0 m²
Type Q	1	746	2000		3880	3590	3 m²	0 m²	3 m²	0 m²
Type R	1	658	2000		3880	3590	2 m²	0 m²	2 m²	0 m²
Type S	1	1480	2000		3880	3590	6 m²	0 m²	5 m²	0 m²
Type T	1	1707	2000		3880	3590	7 m²	0 m²	6 m²	0 m²
Type U	1	1984	2000		3880	3590	8 m²	0 m²	7 m²	0 m²
	128						54 m²	675 m²	50 m²	623 m²

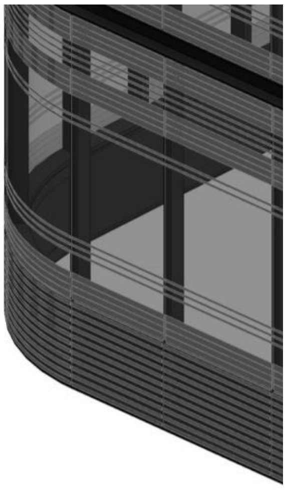
The design architect's façade concept was to employ a unitized system of double-glazed panels with black frit horizontal banding for solar protection. The corners would feature curved glass units with the same finish. We initially developed this design using Revit's conceptual massing tools with adaptive families nested into curtain panel based families to allow user to roughly chop and change design options without much concern of the final panel setting out.



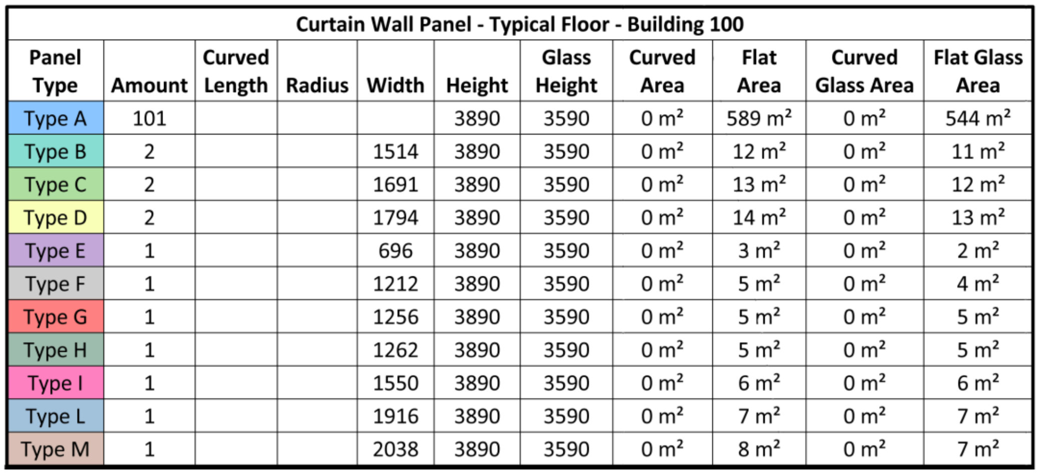
Constraints	
Level	Level 1
Elevation	0.0
Moves With Nearby Elements	☐
Graphics	
Visibility/Graphics Overrides	Edit...
Visible	✓
Materials and Finishes	
Material	GG-Curtain Mullion
Dimensions	
Mullion Thickness	50.0
Mullion Offset	52.0
Mullion Height	201.0
Volume	0.062 m³
Identity Data	
Comments	
Mark	
Adaptive Component	
Flip	☐
Other	
Label	<None>

Project Browser - GG-Curtain Panel - Pattern Based - Fritted Flat_52mm.rfa

- Views (all)
  - Floor Plans
    - Level 1
  - 3D Views
- Sheets (all)
- Families



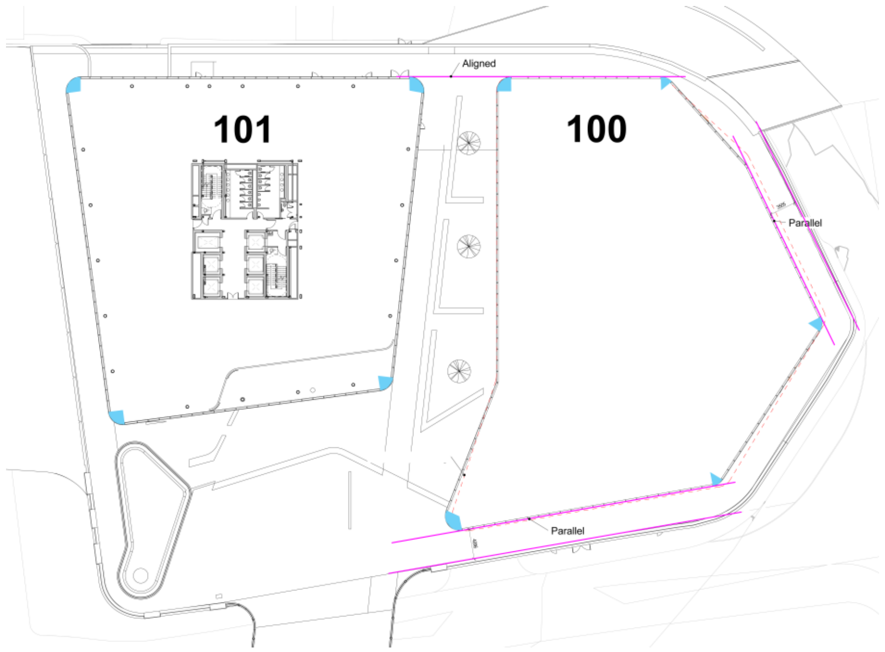
Once the design team had resolved their frit pattern design and general massing concept, we looked more closely at how to optimize the panel setting out. Initial analysis of the building revealed that if we used a standard 1500mm width unit, we would need 13 special width panels per floor. Clearly we needed to make the façade more cost effective for the concept to be viable.



Curtain Wall Panel - Typical Floor - Building 100

Panel Type	Amount	Curved Length	Radius	Width	Height	Glass Height	Curved Area	Flat Area	Curved Glass Area	Flat Glass Area
Type A	101				3890	3590	0 m²	589 m²	0 m²	544 m²
Type B	2			1514	3890	3590	0 m²	12 m²	0 m²	11 m²
Type C	2			1691	3890	3590	0 m²	13 m²	0 m²	12 m²
Type D	2			1794	3890	3590	0 m²	14 m²	0 m²	13 m²
Type E	1			696	3890	3590	0 m²	3 m²	0 m²	2 m²
Type F	1			1212	3890	3590	0 m²	5 m²	0 m²	4 m²
Type G	1			1256	3890	3590	0 m²	5 m²	0 m²	5 m²
Type H	1			1262	3890	3590	0 m²	5 m²	0 m²	5 m²
Type I	1			1550	3890	3590	0 m²	6 m²	0 m²	6 m²
Type L	1			1916	3890	3590	0 m²	7 m²	0 m²	7 m²
Type M	1			2038	3890	3590	0 m²	8 m²	0 m²	7 m²

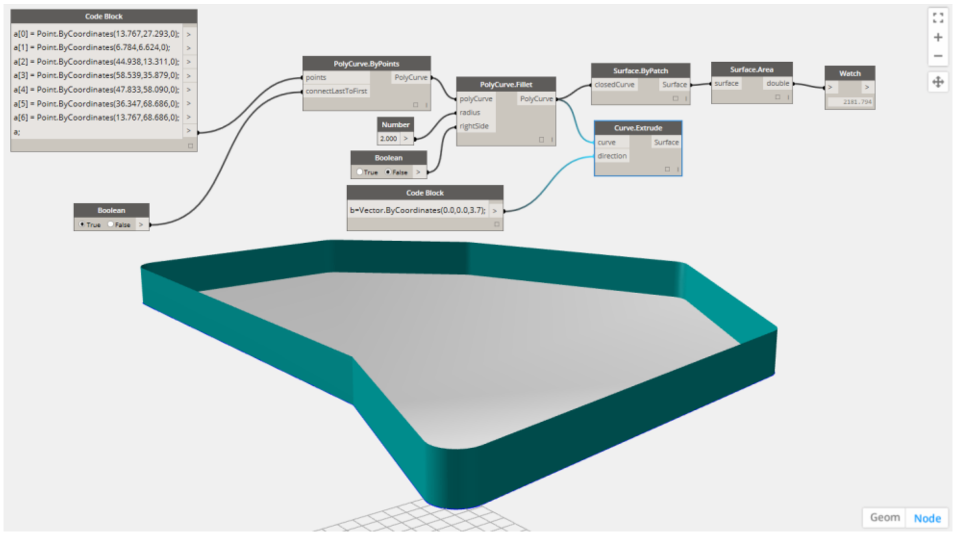
A quick appraisal of the number of possible solutions made us confident that a scripted approach was going to be more time effective than manual (hand drawn) iterations.



Before we could start optimizing, we needed establish the variables and constraints of the Building 100

1. 2 of the facades must remain parallel with the site boundary
2. The back façade must remain parallel with the rear of building 101
3. All the curved panel units on building 101 must have the same radius as building 100
4. Whatever we did the total office area of building 100 should not go below 165,000 ft^2

It was only once we were happy that we had established our constraints that we began our analysis within Dynamo. As all floors repeated we first built a simple surface to represent a level by extracting geometry of the original conceptual mass inside the Revit Project.

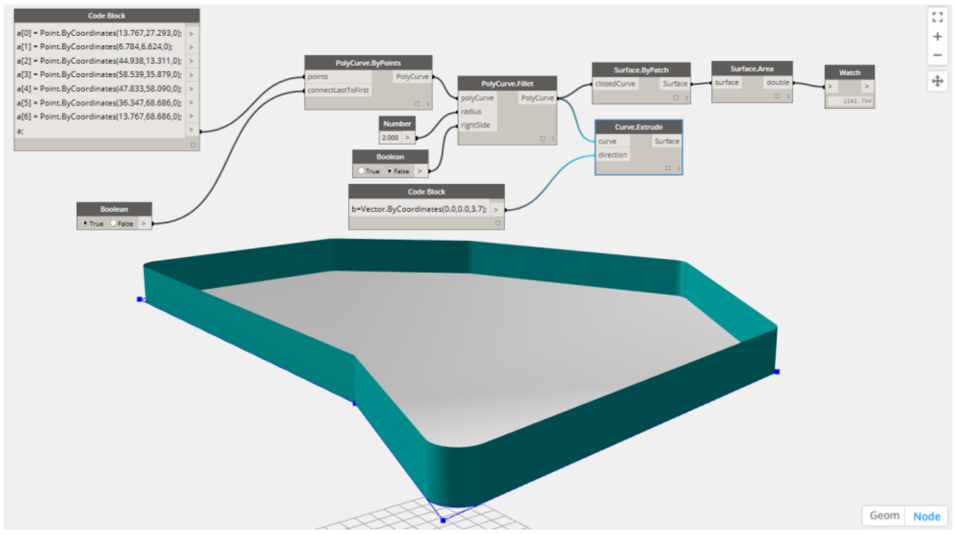


```
a[0] = Point.ByCoordinates(13.767,27.293,0);
a[1] = Point.ByCoordinates(6.784,6.624,0);
a[2] = Point.ByCoordinates(44.938,13.311,0);
a[3] = Point.ByCoordinates(58.539,35.879,0);
a[4] = Point.ByCoordinates(47.833,58.090,0);
a[5] = Point.ByCoordinates(36.347,68.686,0);
a[6] = Point.ByCoordinates(13.767,68.686,0);
a;
```

```
b=Vector.ByCoordinates(0.0,0.0,3.7);
```

Number: 2.000
Boolean: ● True ○ False
Boolean: ○ True ● False
Watch: 2181.794

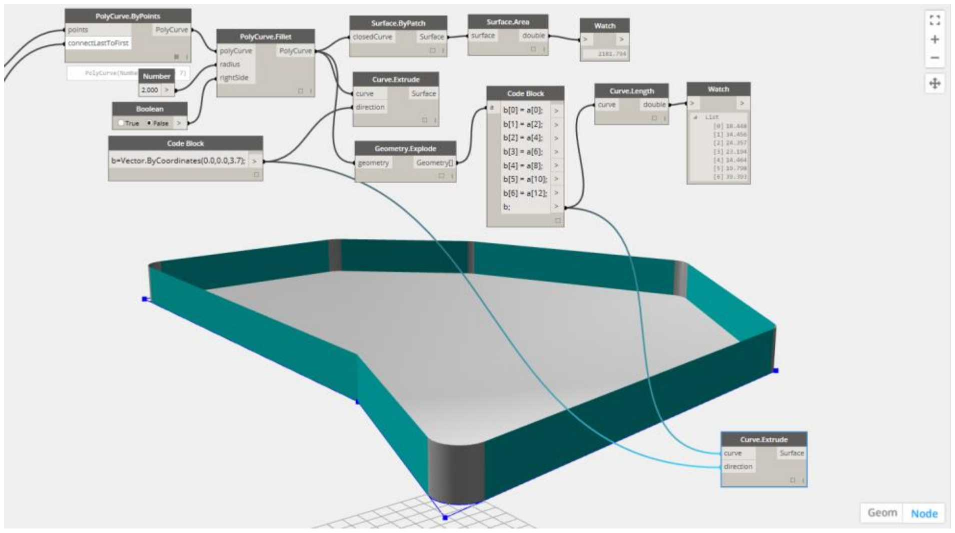
We then rationalized the perimeter into a series of vertices that surrounded the perimeter.



```
a[0] = Point.ByCoordinates(13.767,27.293,0);
a[1] = Point.ByCoordinates(6.784,6.624,0);
a[2] = Point.ByCoordinates(44.938,13.311,0);
a[3] = Point.ByCoordinates(58.539,35.879,0);
a[4] = Point.ByCoordinates(47.833,58.090,0);
a[5] = Point.ByCoordinates(36.347,68.686,0);
a[6] = Point.ByCoordinates(13.767,68.686,0);
a;
```

```
b=Vector.ByCoordinates(0.0,0.0,3.7);
```

We did this so that we would always have a start point from which all iterations would be compared. As good practice process it is always necessary to create a prototype of the code. In this scenario before start scripting in python a prototype of the code is made in Dynamo using its capabilities of visual programming. Then the visual script become a python code

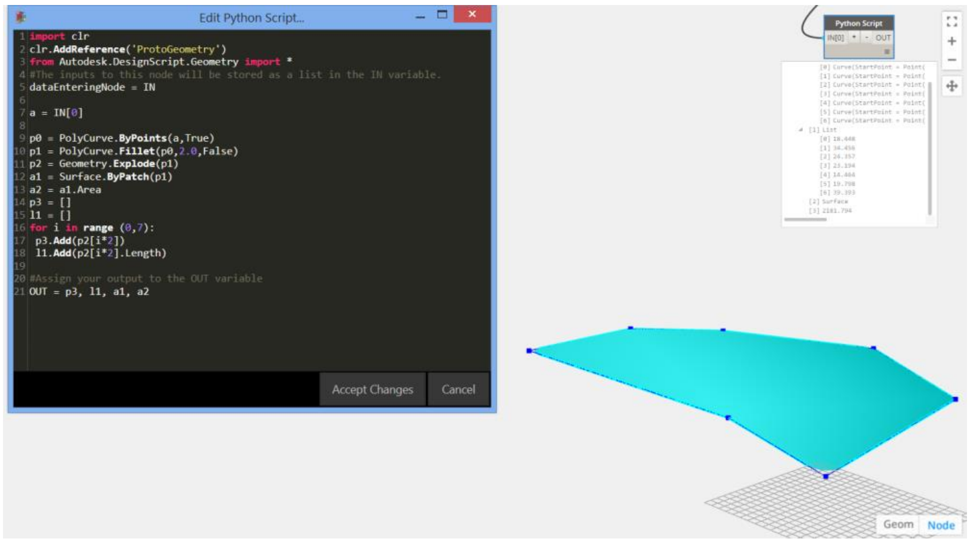


```
b=Vector.ByCoordinates(0.0,0.0,3.7);
```

```
b[0] = a[0];
b[1] = a[2];
b[2] = a[4];
b[3] = a[6];
b[4] = a[8];
b[5] = a[10];
b[6] = a[12];
b;
```

```
List
  [0] 18.448
  [1] 34.456
  [2] 24.357
  [3] 23.194
  [4] 14.464
  [5] 19.798
  [6] 39.393
```

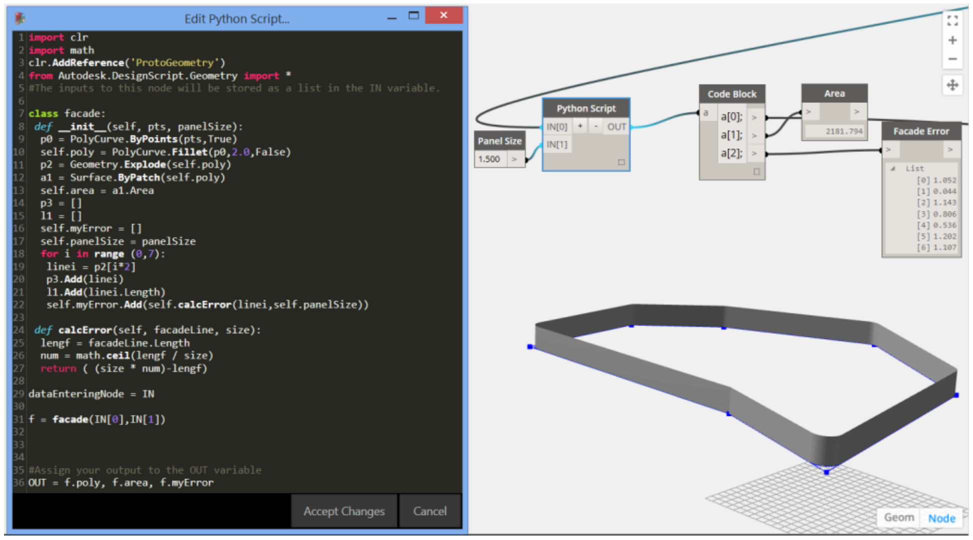
We then introduced a Pythonscript to allow us to run an optimization to find a solution with less panels width types. For reference the façade model was hardcoded as ‘model in place’.



```python
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
#The inputs to this node will be stored as a list in the IN variable.
dataEnteringNode = IN

a = IN[0]

p0 = PolyCurve.ByPoints(a,True)
p1 = PolyCurve.Fillet(p0,2.0,False)
p2 = Geometry.Explode(p1)
a1 = Surface.ByPatch(p1)
a2 = a1.Area
p3 = []
l1 = []
for i in range (0,7):
 p3.Add(p2[i*2])
 l1.Add(p2[i*2].Length)

#Assign your output to the OUT variable
OUT = p3, l1, a1, a2
```

```
    [0] Curve(StartPoint = Point(
    [1] Curve(StartPoint = Point(
    [2] Curve(StartPoint = Point(
    [3] Curve(StartPoint = Point(
    [4] Curve(StartPoint = Point(
    [5] Curve(StartPoint = Point(
    [6] Curve(StartPoint = Point(
  [1] List
    [0] 18.448
    [1] 34.456
    [2] 24.357
    [3] 23.194
    [4] 14.484
    [5] 19.798
    [6] 39.393
  [2] Surface
  [3] 2181.794
```

Initially we tested the suitability of the script by checking how much error (essential gaps) was generated when inputting our standard panel width is 1500mm into the original geometry

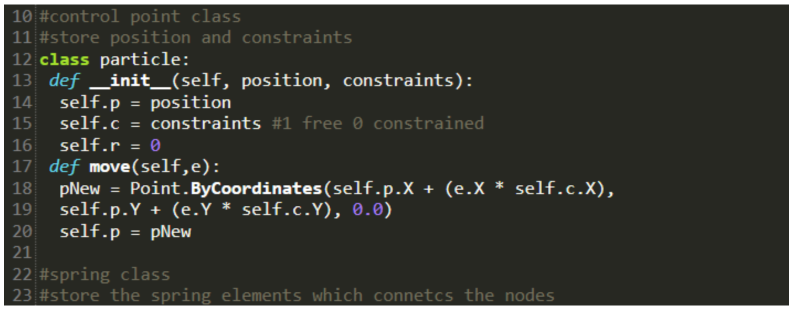


```python
import clr
import math
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
#The inputs to this node will be stored as a list in the IN variable.

class facade:
 def __init__(self, pts, panelSize):
  p0 = PolyCurve.ByPoints(pts,True)
  self.poly = PolyCurve.Fillet(p0,2.0,False)
  p2 = Geometry.Explode(self.poly)
  a1 = Surface.ByPatch(self.poly)
  self.area = a1.Area
  p3 = []
  l1 = []
  self.myError = []
  self.panelSize = panelSize
  for i in range (0,7):
   linei = p2[i*2]
   p3.Add(linei)
   l1.Add(linei.Length)
   self.myError.Add(self.calcError(linei,self.panelSize))

 def calcError(self, facadeLine, size):
  lengf = facadeLine.Length
  num = math.ceil(lengf / size)
  return ( (size * num)-lengf)

dataEnteringNode = IN

f = facade(IN[0],IN[1])



#Assign your output to the OUT variable
OUT = f.poly, f.area, f.myError
```

```
2181.794

List
  [0] 1.052
  [1] 0.044
  [2] 1.143
  [3] 0.806
  [4] 0.536
  [5] 1.202
  [6] 1.107
```

The class facade represents exactly what we have previously done with visual programming. Now that we have a class we can do an iterative procedure.

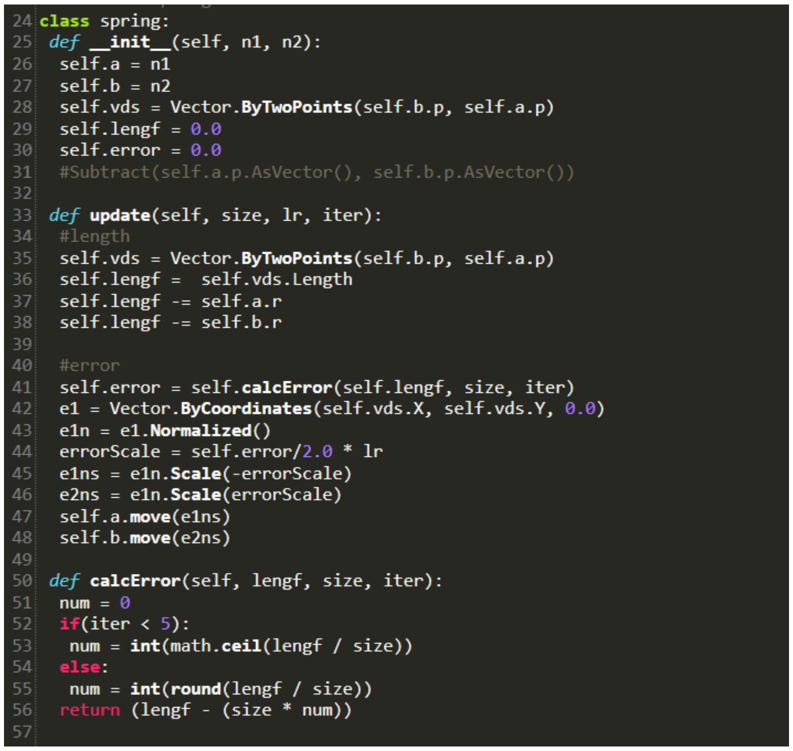
First we implement a class particle defining the position of the control points. A move method moves the control points if this is not constrained (value = 1). This class has also a field r which represents the distance between the control node and the intersection between the facade and the curved corner.



```
#control point class
#store position and constraints
class particle:
 def __init__(self, position, constraints):
  self.p = position
  self.c = constraints #1 free 0 constrained
  self.r = 0
 def move(self,e):
  pNew = Point.ByCoordinates(self.p.X + (e.X * self.c.X),
  self.p.Y + (e.Y * self.c.Y), 0.0)
  self.p = pNew

#spring class
#store the spring elements which connetcs the nodes
```

The spring is the edge connecting 2 particles. It has a vector for the direction (necessary to move the particles). The length and the error are initialized to 0.0 value. An update method is necessary to calculate the actual length between two points which is reduced depending on the fillet of the corner. This method does also the optimization; it calculates the error length which is the value which needs to be minimized, in other words our fitness landscape. Depending on the error and the learning rate, the length is moved closer to the optimal point by moving the nodes.

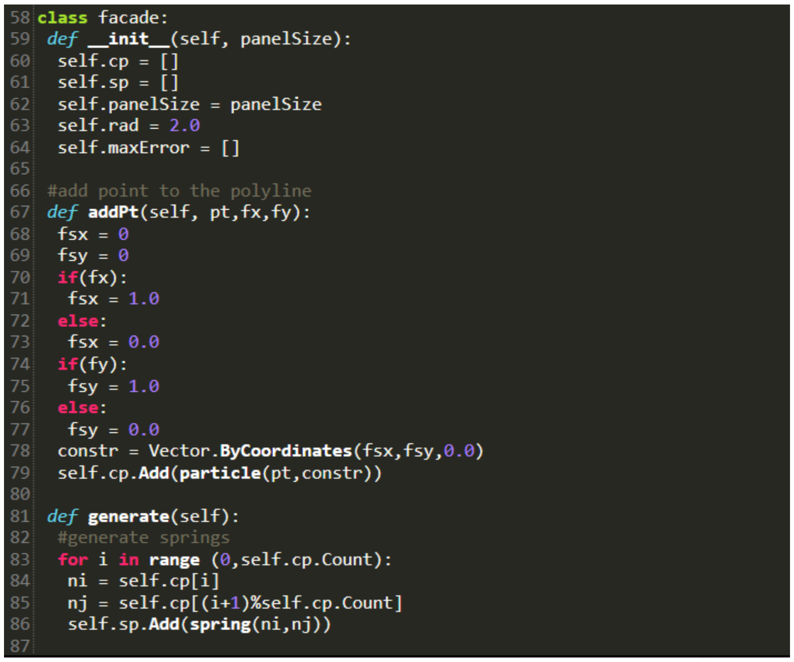
The calcError method calculates the error, our function, depending on the iteration (to make it more stable the round is different for the first 5 iterations) dividing the edge by the panel size, getting the actual size of the rationalized facade ad subtracting the actual length to this value.



```
class spring:
 def __init__(self, n1, n2):
  self.a = n1
  self.b = n2
  self.vds = Vector.ByTwoPoints(self.b.p, self.a.p)
  self.lengf = 0.0
  self.error = 0.0
  #Subtract(self.a.p.AsVector(), self.b.p.AsVector())

 def update(self, size, lr, iter):
  #length
  self.vds = Vector.ByTwoPoints(self.b.p, self.a.p)
  self.lengf =  self.vds.Length
  self.lengf -= self.a.r
  self.lengf -= self.b.r

  #error
  self.error = self.calcError(self.lengf, size, iter)
  e1 = Vector.ByCoordinates(self.vds.X, self.vds.Y, 0.0)
  e1n = e1.Normalized()
  errorScale = self.error/2.0 * lr
  e1ns = e1n.Scale(-errorScale)
  e2ns = e1n.Scale(errorScale)
  self.a.move(e1ns)
  self.b.move(e2ns)

 def calcError(self, lengf, size, iter):
  num = 0
  if(iter < 5):
   num = int(math.ceil(lengf / size))
  else:
   num = int(round(lengf / size))
  return (lengf - (size * num))

```

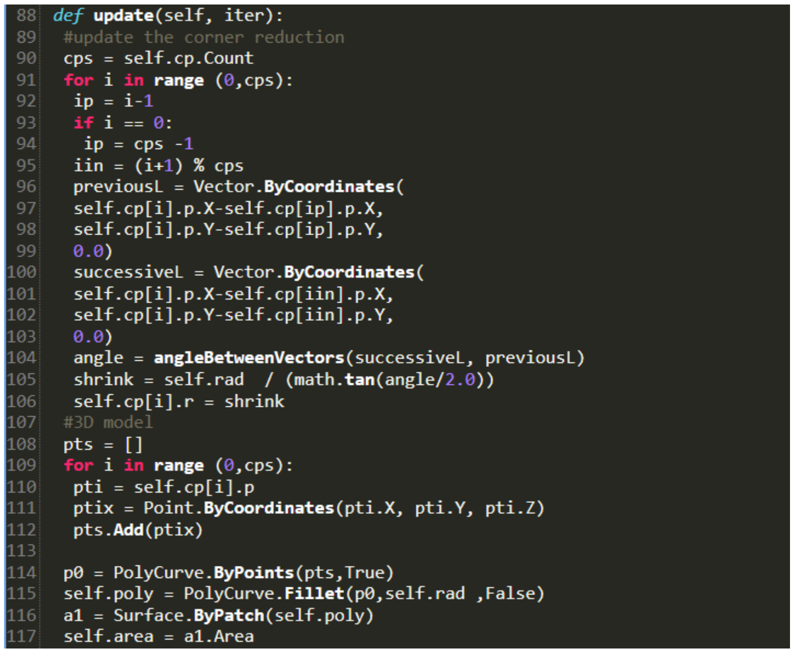
The class facade is pretty much the same as the one from the visual script. Spring and Particles are added as fields in order to perform the optimization.

In this case the initialization is more dynamic, the nodes are added sequentially using the addPt method, and before optimizing it a generate method generates spring and particles depending on the control points.



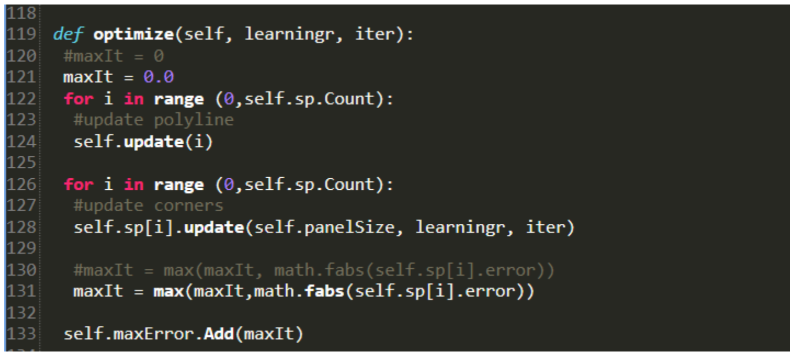
```
class facade:
 def __init__(self, panelSize):
  self.cp = []
  self.sp = []
  self.panelSize = panelSize
  self.rad = 2.0
  self.maxError = []

 #add point to the polyline
 def addPt(self, pt,fx,fy):
  fsx = 0
  fsy = 0
  if(fx):
   fsx = 1.0
  else:
   fsx = 0.0
  if(fy):
   fsy = 1.0
  else:
   fsy = 0.0
  constr = Vector.ByCoordinates(fsx,fsy,0.0)
  self.cp.Add(particle(pt,constr))

 def generate(self):
  #generate springs
  for i in range (0,self.cp.Count):
   ni = self.cp[i]
   nj = self.cp[(i+1)%self.cp.Count]
   self.sp.Add(spring(ni,nj))
```

Since the corners all needed to have the same radius we used and updated method to generate the corner reduction using trigonometry to establish the start points of each straight façade irrespective of intersection angle.



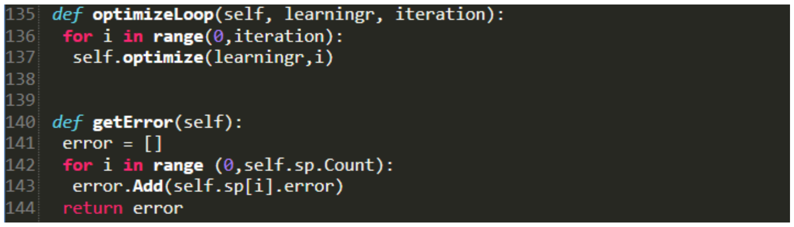
```
def update(self, iter):
 #update the corner reduction
 cps = self.cp.Count
 for i in range (0,cps):
  ip = i-1
  if i == 0:
   ip = cps -1
  iin = (i+1) % cps
  previousL = Vector.ByCoordinates(
  self.cp[i].p.X-self.cp[ip].p.X,
  self.cp[i].p.Y-self.cp[ip].p.Y,
  0.0)
  successiveL = Vector.ByCoordinates(
  self.cp[i].p.X-self.cp[iin].p.X,
  self.cp[i].p.Y-self.cp[iin].p.Y,
  0.0)
  angle = angleBetweenVectors(successiveL, previousL)
  shrink = self.rad  / (math.tan(angle/2.0))
  self.cp[i].r = shrink
 #3D model
 pts = []
 for i in range (0,cps):
  pti = self.cp[i].p
  ptix = Point.ByCoordinates(pti.X, pti.Y, pti.Z)
  pts.Add(ptix)

 p0 = PolyCurve.ByPoints(pts,True)
 self.poly = PolyCurve.Fillet(p0,self.rad ,False)
 a1 = Surface.ByPatch(self.poly)
 self.area = a1.Area
```

This method performs the optimization. First it updates the overall facade then optimize the single edge with a gradient descent algorithm developed in the spring class. The error is stored into an array in order to check the simulation convergence.



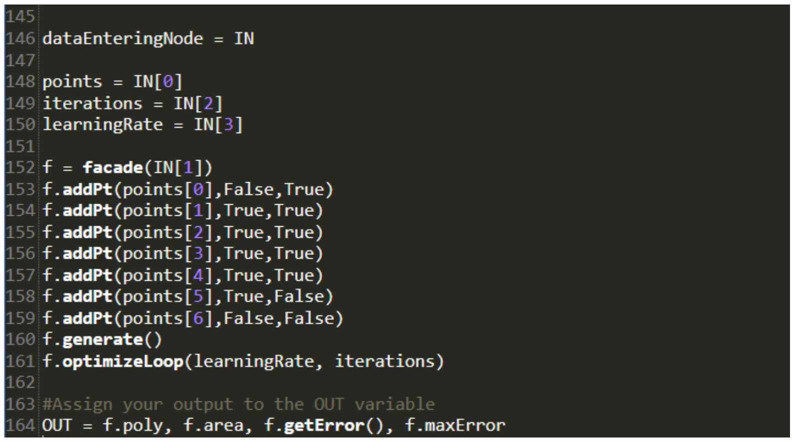
```

def optimize(self, learningr, iter):
 #maxIt = 0
 maxIt = 0.0
 for i in range (0,self.sp.Count):
  #update polyline
  self.update(i)

 for i in range (0,self.sp.Count):
  #update corners
  self.sp[i].update(self.panelSize, learningr, iter)

  #maxIt = max(maxIt, math.fabs(self.sp[i].error))
  maxIt = max(maxIt,math.fabs(self.sp[i].error))

 self.maxError.Add(maxIt)
```

Finally when we were confident that the script was producing stable results, we added code for it to run multiple iterations and output results



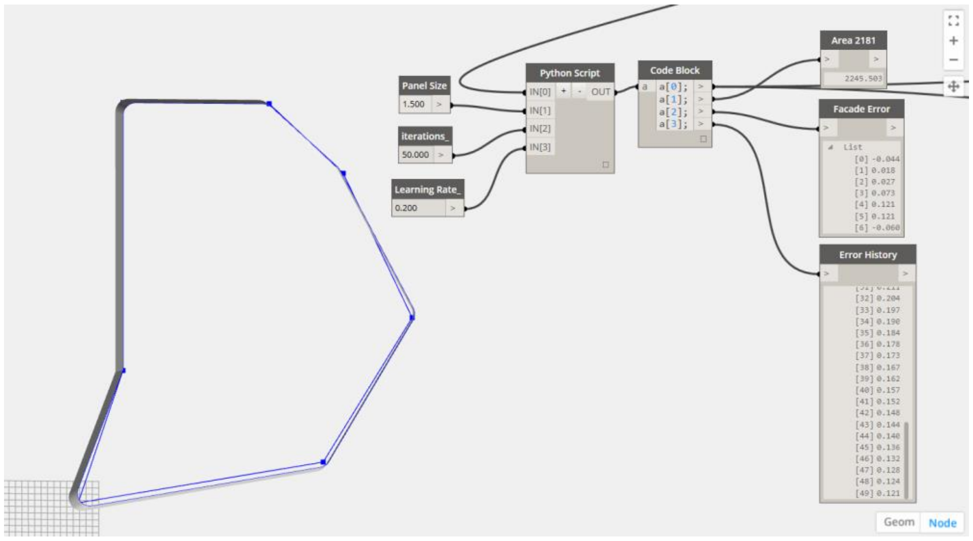
```
def optimizeLoop(self, learningr, iteration):
 for i in range(0,iteration):
  self.optimize(learningr,i)


def getError(self):
 error = []
 for i in range (0,self.sp.Count):
  error.Add(self.sp[i].error)
 return error
```

This is the actual main script. We input the points, iteration steps, and learning rate value. Then the facade is initialized (depending on the panel size) then each point is added, with its constraint. The facade is generated and optimized for a number of iterations). The results are in out variable.



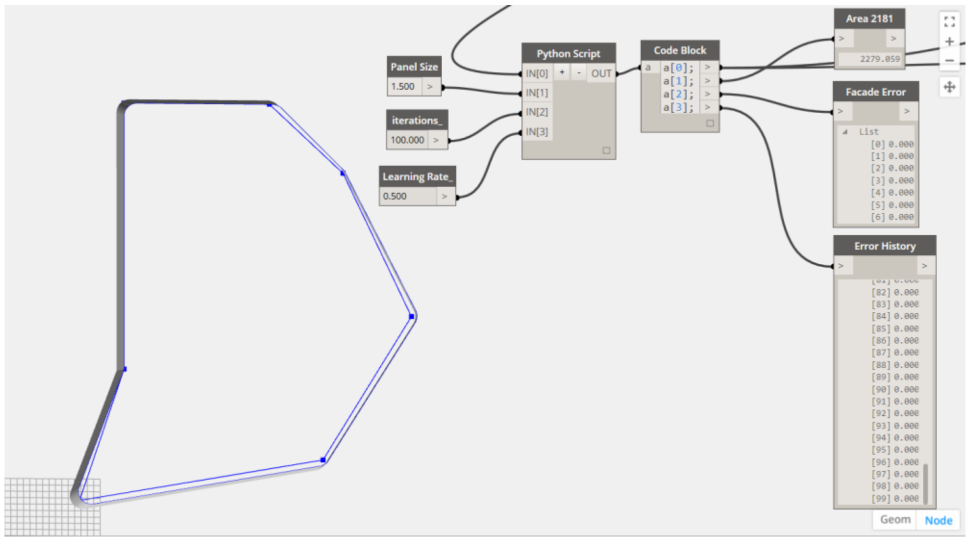
```

dataEnteringNode = IN

points = IN[0]
iterations = IN[2]
learningRate = IN[3]

f = facade(IN[1])
f.addPt(points[0],False,True)
f.addPt(points[1],True,True)
f.addPt(points[2],True,True)
f.addPt(points[3],True,True)
f.addPt(points[4],True,True)
f.addPt(points[5],True,False)
f.addPt(points[6],False,False)
f.generate()
f.optimizeLoop(learningRate, iterations)

#Assign your output to the OUT variable
OUT = f.poly, f.area, f.getError(), f.maxError
```

These images show the script created various alternative perimeter geometries (grey) which could be compared the original proposal (blue). As you can see from the first image, sometimes there are insufficient iterations to achieve the target values (59 iterations with a learning rate of 0.5)



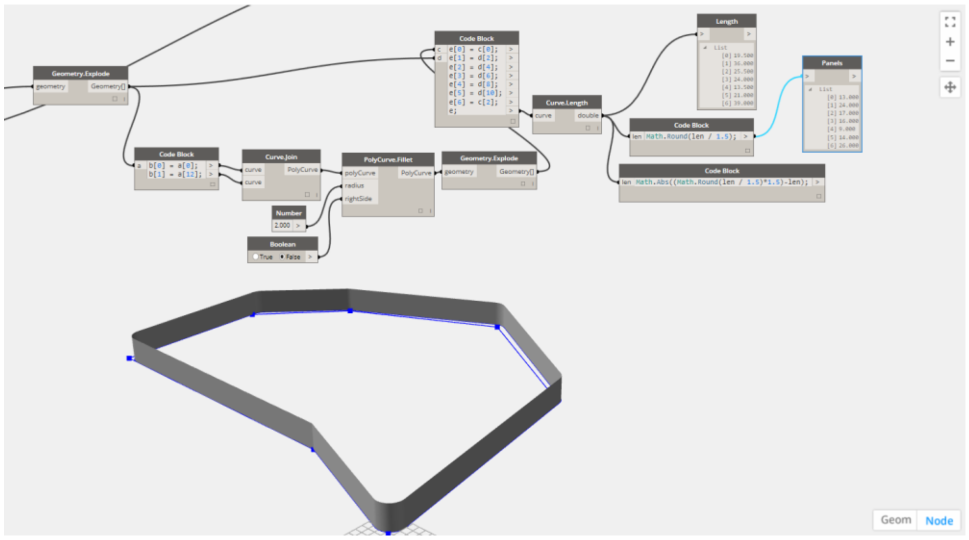
```
Panel Size        1.500
iterations_       50.000
Learning Rate_    0.200

Code Block
a   a[0];
    a[1];
    a[2];
    a[3];

Area 2181
2245.503

Facade Error
List
  [0] -0.044
  [1] 0.018
  [2] 0.027
  [3] 0.073
  [4] 0.121
  [5] 0.121
  [6] -0.060

Error History
[32] 0.204
[33] 0.197
[34] 0.190
[35] 0.184
[36] 0.178
[37] 0.173
[38] 0.167
[39] 0.162
[40] 0.157
[41] 0.152
[42] 0.148
[43] 0.144
[44] 0.140
[45] 0.136
[46] 0.132
[47] 0.128
[48] 0.124
[49] 0.121
```

However when the number of iterations was doubled to 100 (learning rate 0.5) the errors werer under 0.00049 m (within tolerance).



```
Panel Size        1.500
iterations_       100.000
Learning Rate_    0.500

Code Block
a   a[0];
    a[1];
    a[2];
    a[3];

Area 2181
2279.059

Facade Error
List
  [0] 0.000
  [1] 0.000
  [2] 0.000
  [3] 0.000
  [4] 0.000
  [5] 0.000
  [6] 0.000

Error History
[82] 0.000
[83] 0.000
[84] 0.000
[85] 0.000
[86] 0.000
[87] 0.000
[88] 0.000
[89] 0.000
[90] 0.000
[91] 0.000
[92] 0.000
[93] 0.000
[94] 0.000
[95] 0.000
[96] 0.000
[97] 0.000
[98] 0.000
[99] 0.000
```

The results



Code Block
```
b[0] = a[0];
b[1] = a[12];
```

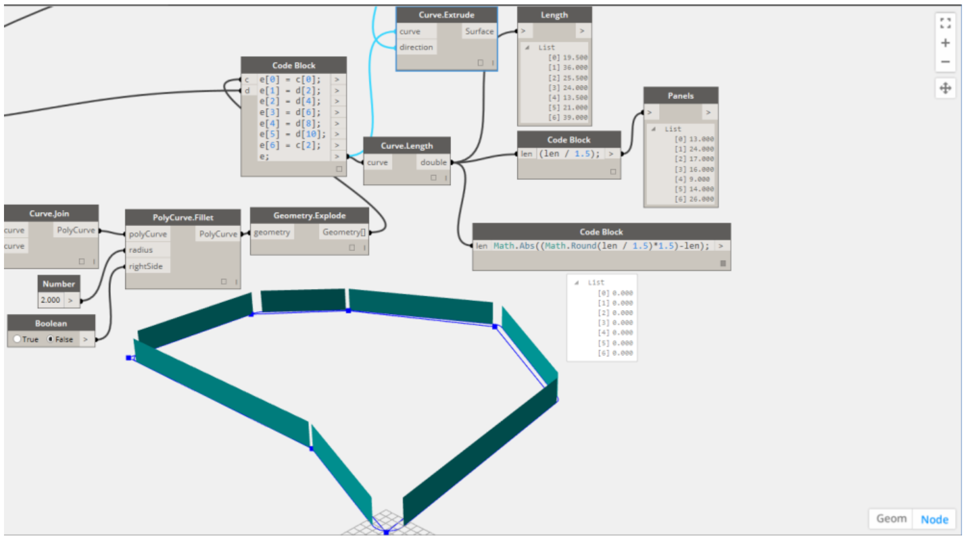
Code Block
```
e[0] = c[0];
e[1] = d[2];
e[2] = d[4];
e[3] = d[6];
e[4] = d[8];
e[5] = d[10];
e[6] = c[2];
e;
```

Code Block
```
Math.Round(len / 1.5);
```

Code Block
```
Math.Abs((Math.Round(len / 1.5)*1.5)-len);
```

Length
```
List
  [0] 19.500
  [1] 36.000
  [2] 25.500
  [3] 24.000
  [4] 13.500
  [5] 21.000
  [6] 39.000
```

Panels
```
List
  [0] 13.000
  [1] 24.000
  [2] 17.000
  [3] 16.000
  [4] 9.000
  [5] 14.000
  [6] 26.000
```



Code Block
```
e[0] = c[0];
e[1] = d[2];
e[2] = d[4];
e[3] = d[6];
e[4] = d[8];
e[5] = d[10];
e[6] = c[2];
e;
```

Code Block
```
(len / 1.5);
```

Code Block
```
Math.Abs((Math.Round(len / 1.5)*1.5)-len);
```

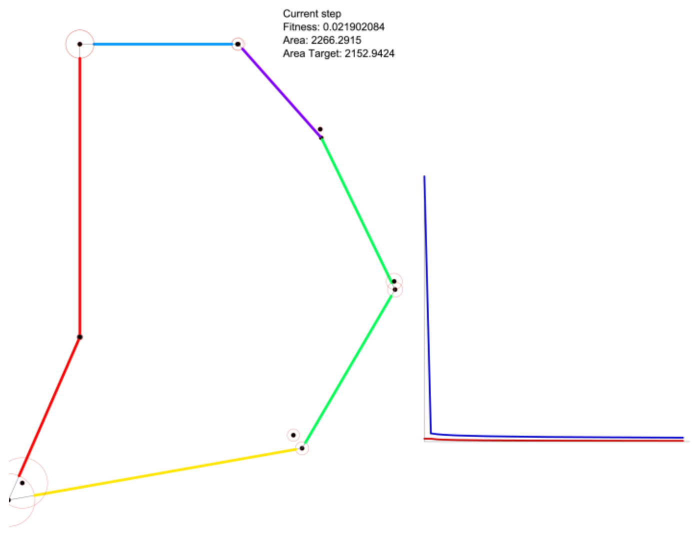
Length
```
List
  [0] 19.500
  [1] 36.000
  [2] 25.500
  [3] 24.000
  [4] 13.500
  [5] 21.000
  [6] 39.000
```

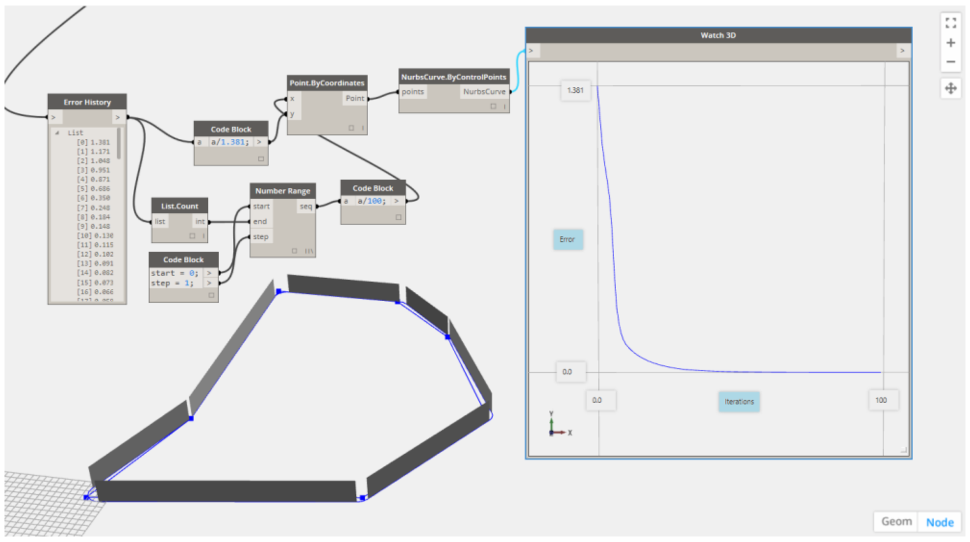
Panels
```
List
  [0] 13.000
  [1] 24.000
  [2] 17.000
  [3] 16.000
  [4] 9.000
  [5] 14.000
  [6] 26.000
```

```
List
  [0] 0.000
  [1] 0.000
  [2] 0.000
  [3] 0.000
  [4] 0.000
  [5] 0.000
  [6] 0.000
```

Live results

An excellent script prototyping tool called Processing inspired us to create a graphical tool for representing the calculations. It allows the user to save time by choosing to terminate the calculations if the curve converges on the two axes within a target tolerance.





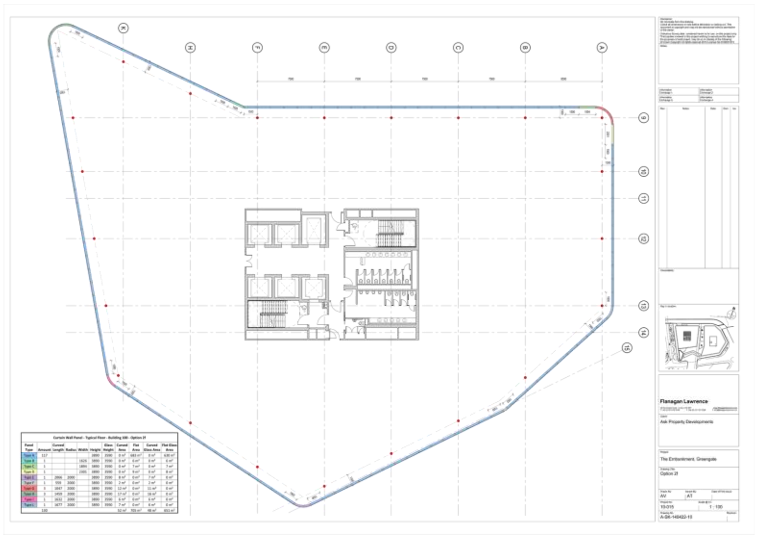
```
List
  [0] 1.381
  [1] 1.171
  [2] 1.048
  [3] 0.951
  [4] 0.871
  [5] 0.686
  [6] 0.350
  [7] 0.248
  [8] 0.184
  [9] 0.148
  [10] 0.130
  [11] 0.115
  [12] 0.102
  [13] 0.091
  [14] 0.082
  [15] 0.073
  [16] 0.066
```

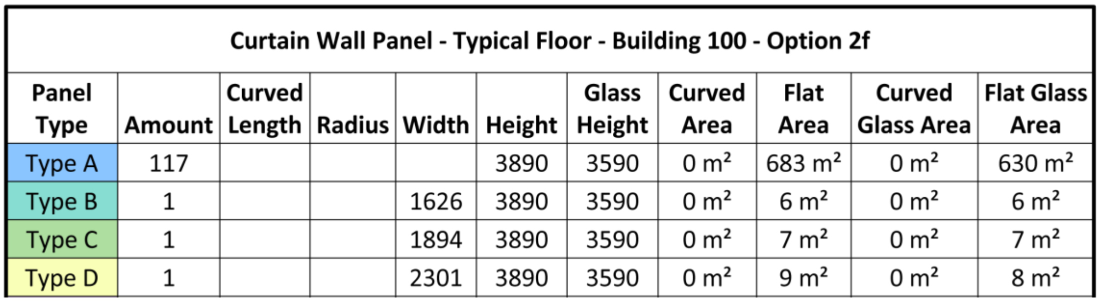
```
a/1.381;
```

```
a/100;
```

```
start = 0;
step = 1;
```

Overall we were able to reduce the number of special panels per floor from 13 to 3





Curtain Wall Panel - Typical Floor - Building 100 - Option 2f

Panel Type	Amount	Curved Length	Radius	Width	Height	Glass Height	Curved Area	Flat Area	Curved Glass Area	Flat Glass Area
Type A	117				3890	3590	0 m²	683 m²	0 m²	630 m²
Type B	1			1626	3890	3590	0 m²	6 m²	0 m²	6 m²
Type C	1			1894	3890	3590	0 m²	7 m²	0 m²	7 m²
Type D	1			2301	3890	3590	0 m²	9 m²	0 m²	8 m²

Conclusion

Lessons learnt

What are the limitations?

- Understanding the ASM kernel inside Revit
- Coding and math skills are necessary for the designer
- Coding interface not always intuitive such as visual programming
- A computational design expert in the team would be a beneficial (not necessarily easy to recruit)

Future opportunities

- Rhynamo
- Dynaworks
- Dynventor?
- Simulation tools similar to GH Galapagos

Session Feedback

We value your feedback – after the class is completed you will receive an email with a link to complete a class survey. You can respond to that email on your computer, mobile device or via the Survey stations and each day Autodesk will give away two free passes to next year's AU.