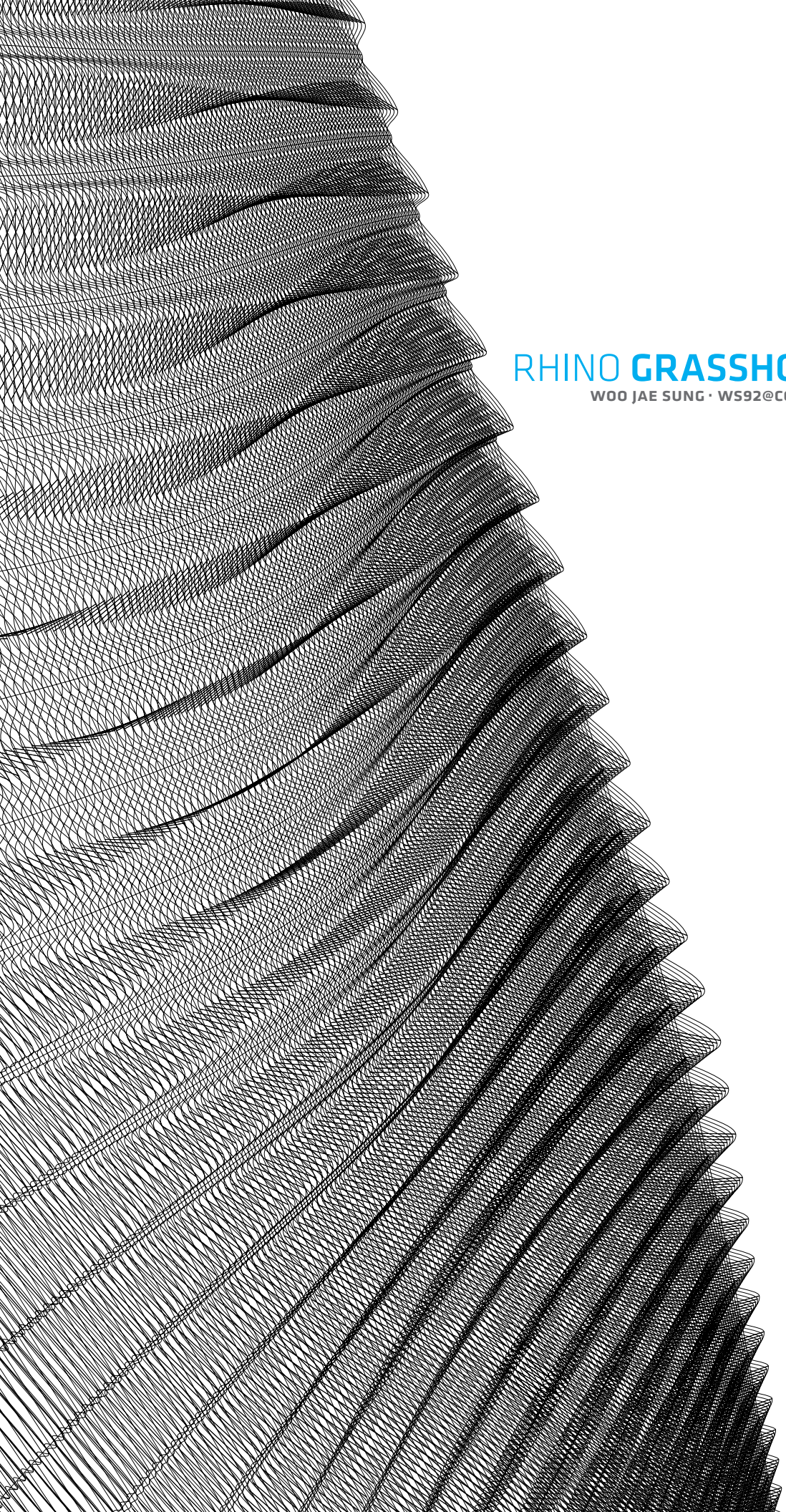
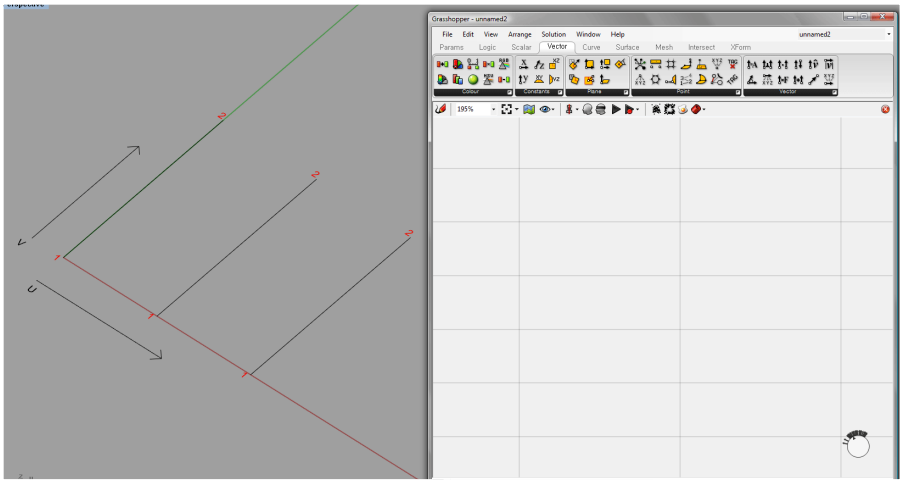
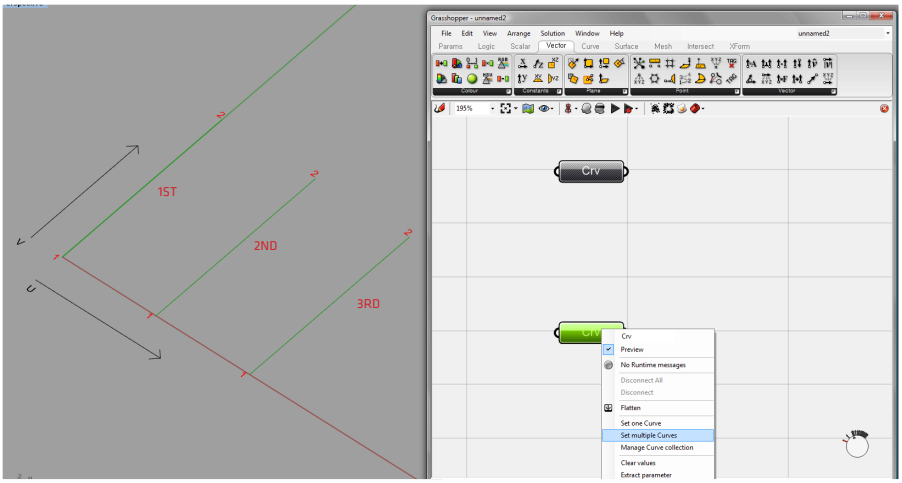


RHINO **GRASSHOPPER** TUTORIAL

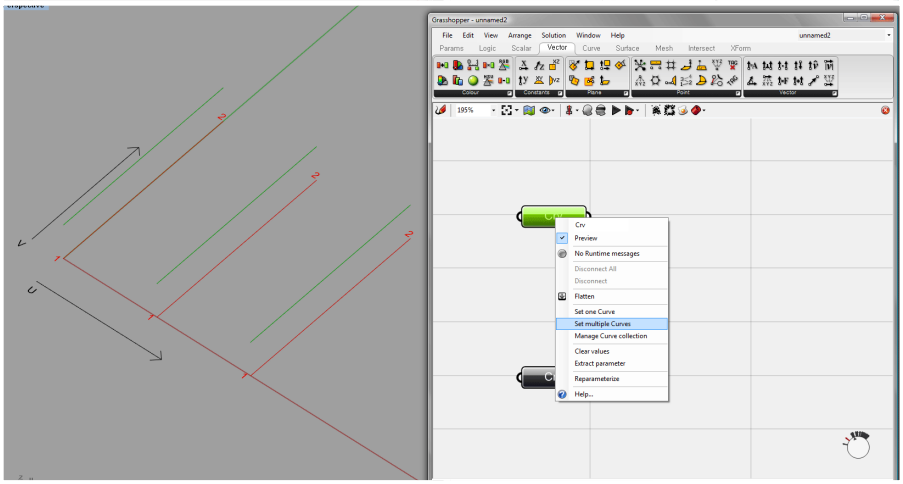
WOO JAE SUNG · WS92@CORNELL.EDU · WWW.WOOJSUNG.COM



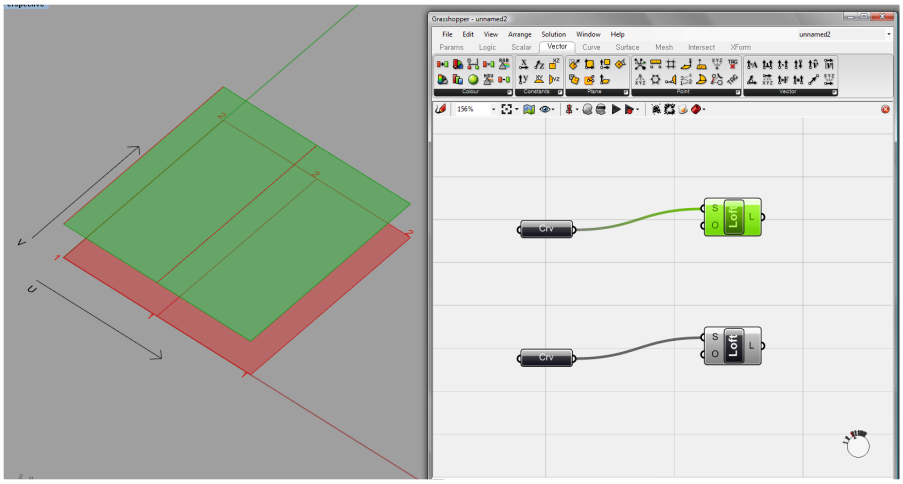
+ **Draw three lines [Rhino input]**
  - Note line direction



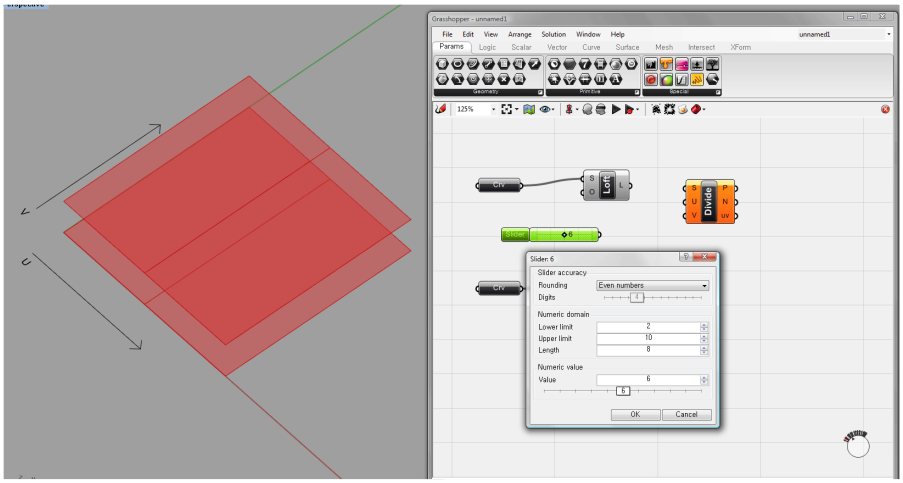
+ **Crv object [GH object]**
+ **Connect Rhino & GH objects**
  - RMB, select 'set multiple curves'
  - Note line picking order



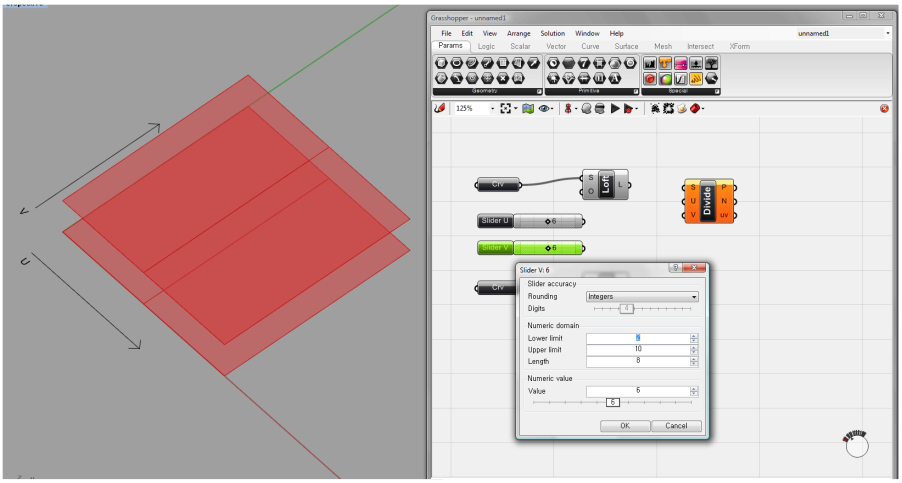
+ **Do the same thing for upper layer**



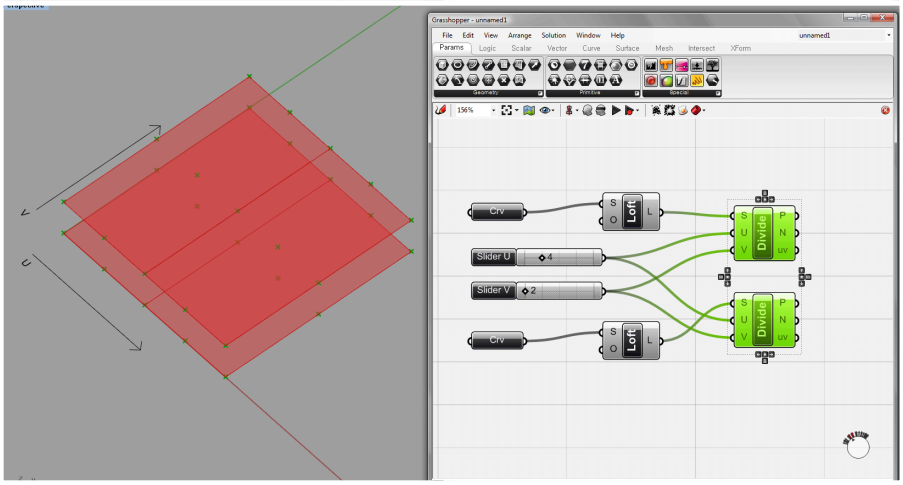
+ **Loft object**
  - Default loft option



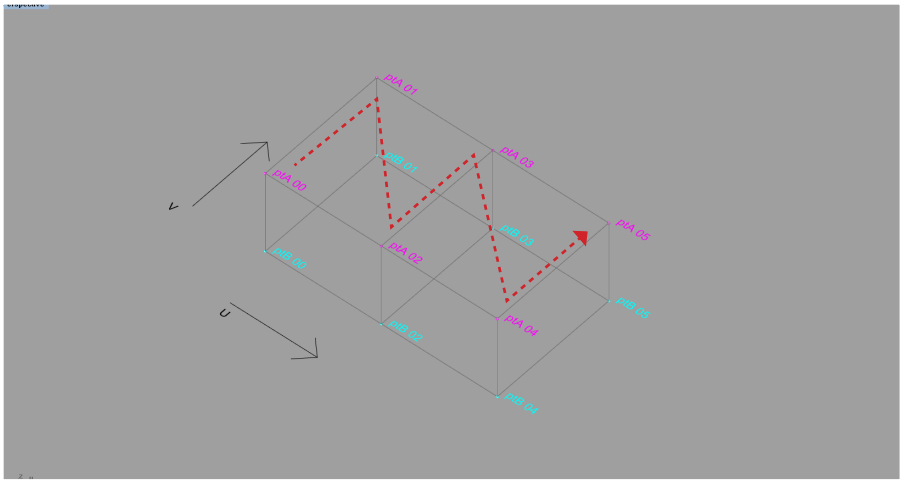
+ **Divide object for surface division**



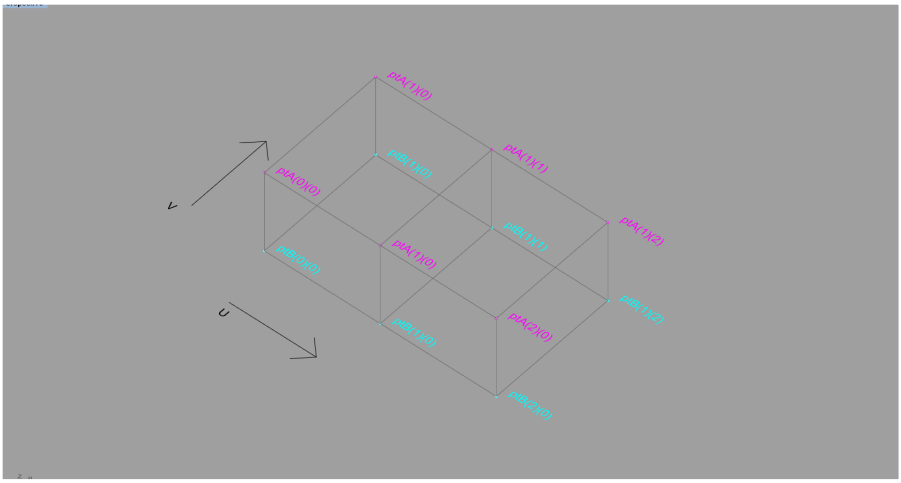
+ **Number slider setting**
  - ‘Even numbers’ for U direction
  - ‘Integer’ for V direction



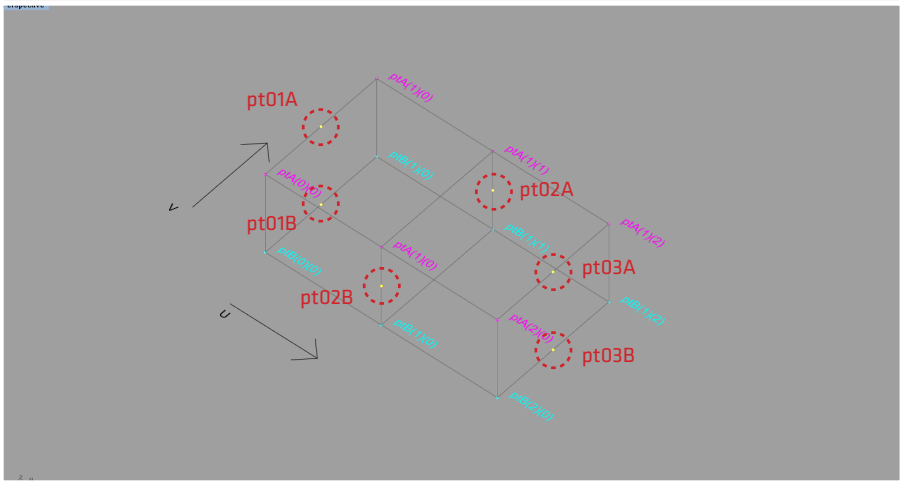
+ **Connect Divide object to surfaces & sliders**



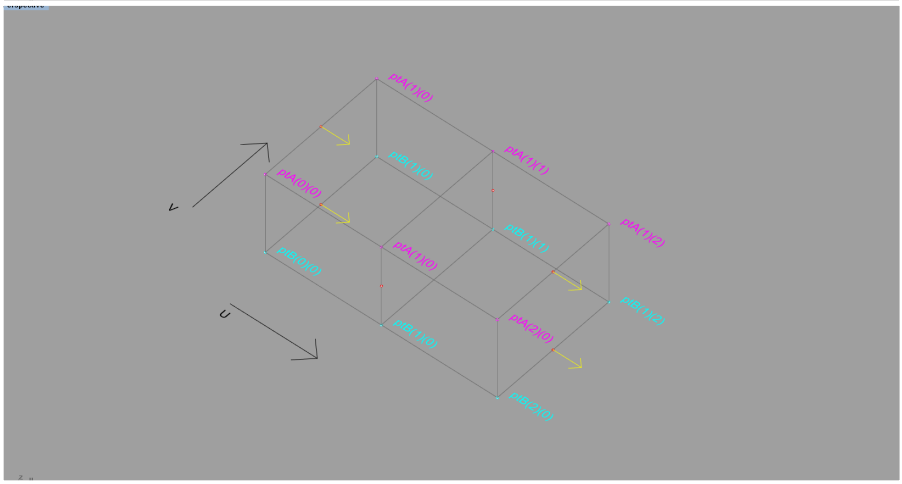
+ **By default, points generated through ‘Divide Srf’ object are to be ordered in zigzag fashion.**



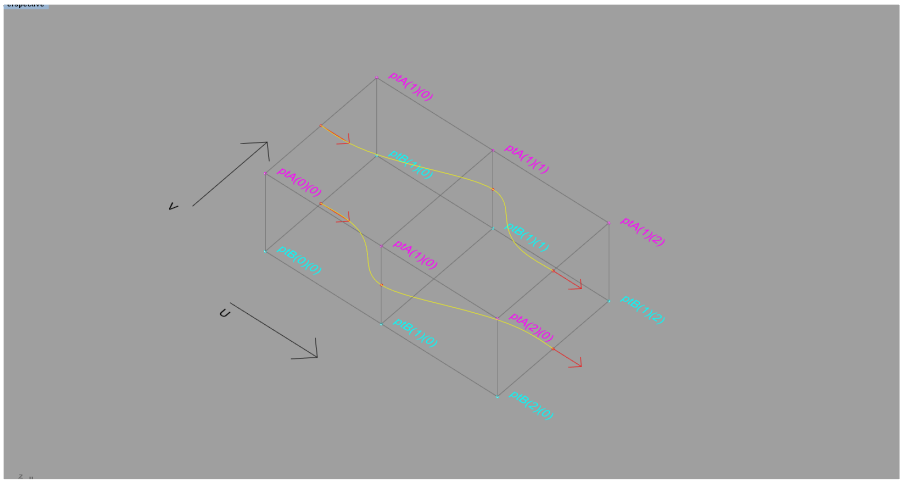
+ **For better control, point ordering method should be changed into 2 x 2 array or list.**



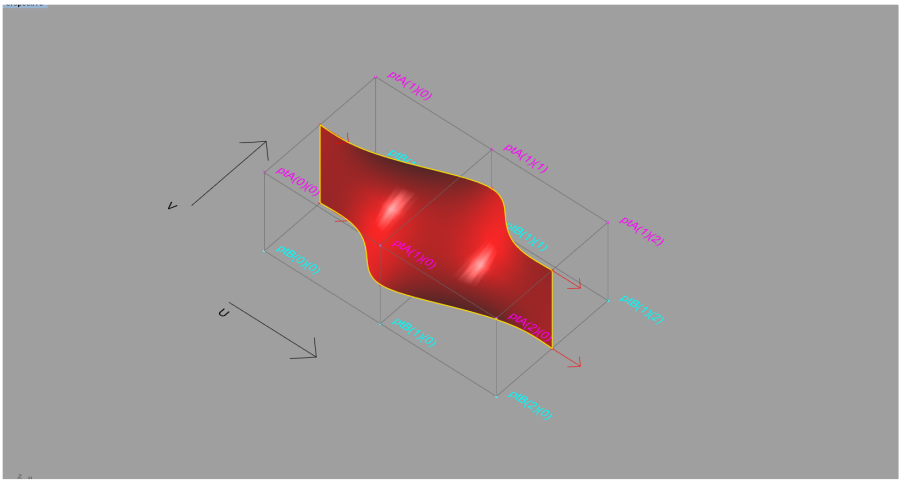
+ **Get mid points**
  - pt01A = (ptA(0)(0) + ptA(1)(0)) / 2
  - pt02A = (ptA(1)(1) + ptB(1)(1)) / 2
  - pt03A = (ptA(2)(0) + ptA(1)(2)) / 2
  - pt01B = (ptB(0)(0) + ptB(1)(0)) / 2
  - pt02B = (ptA(1)(0) + ptB(1)(0)) / 2
  - pt03B = (ptB(2)(0) + ptB(1)(2)) / 2



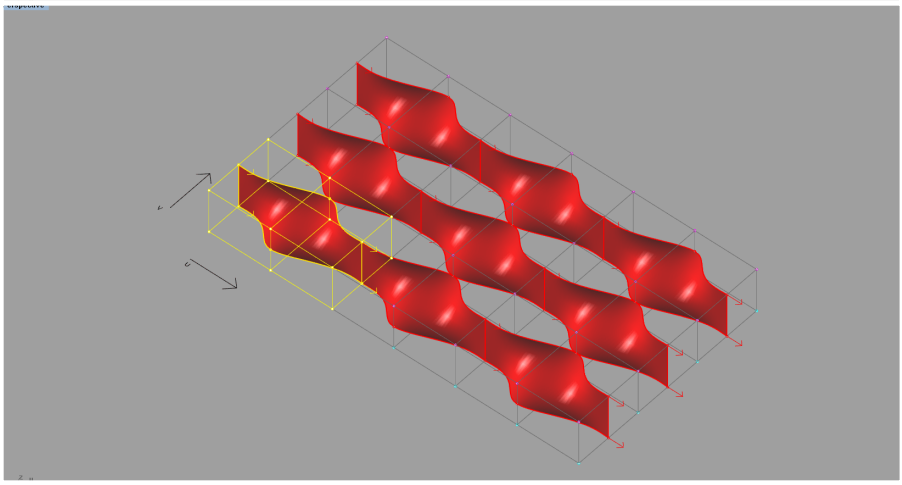
+ **Set direction vectors**



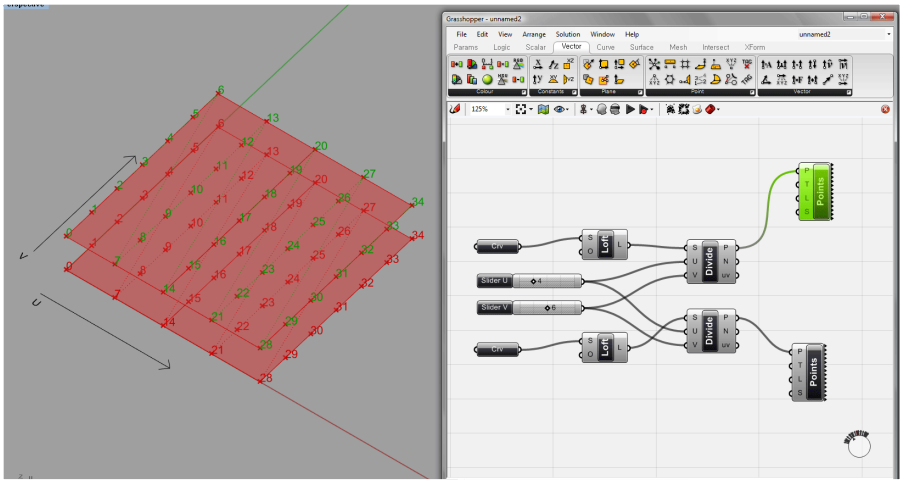
+ Draw two interpolate curves.



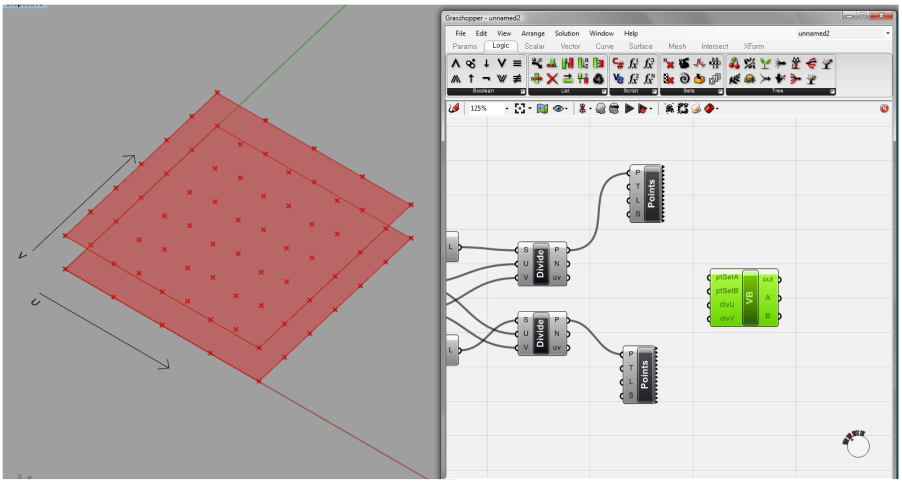
+ Loft



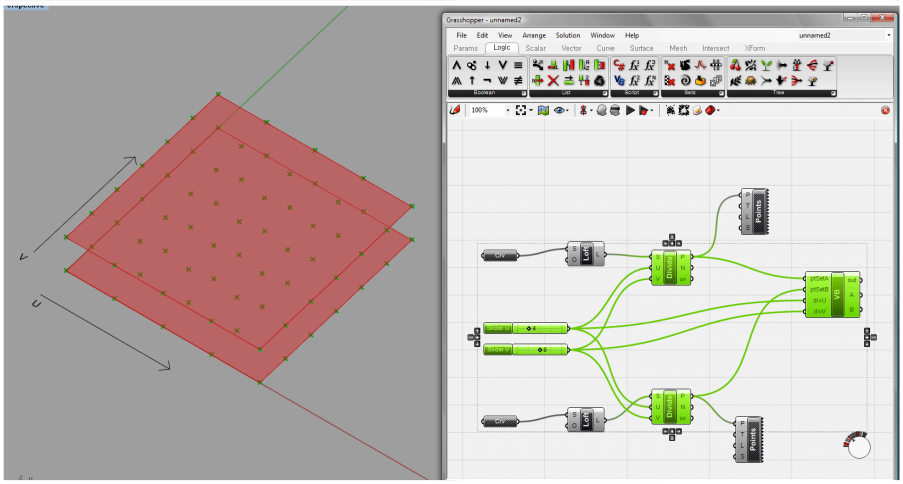
+ Component expansion



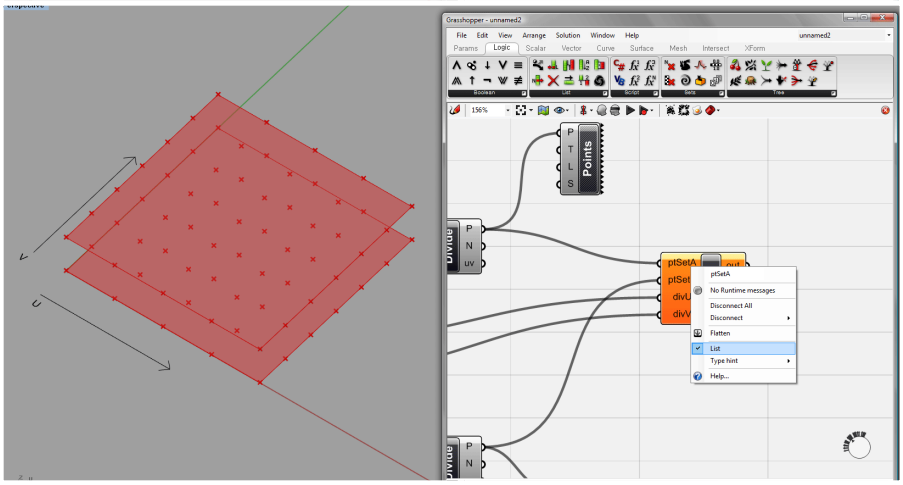
+ **Point List object attached to see point order on surfaces (zigzag order)**



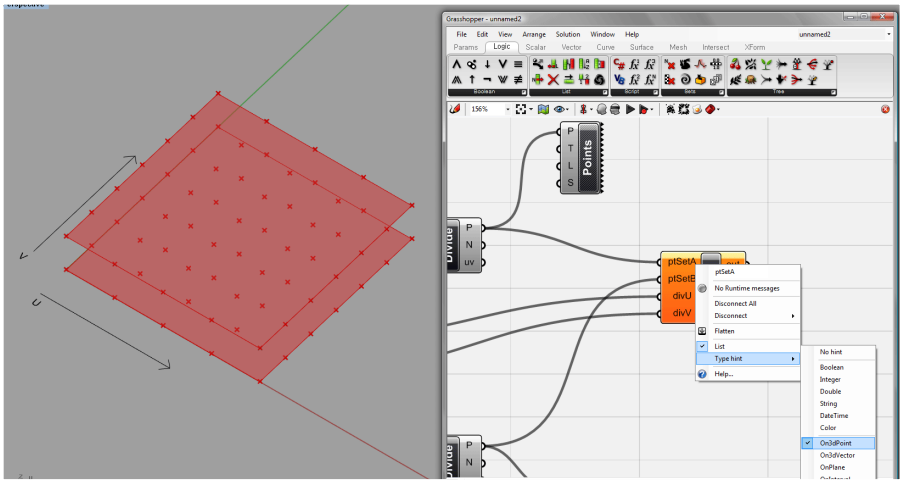
+ **Setting VB component**
  - Four input parameters
  - ptSetA : points set A from upper surface
  - ptSetB : points set B from lower surface
  - divU : U direction division factor
  - divV : V direction division factor



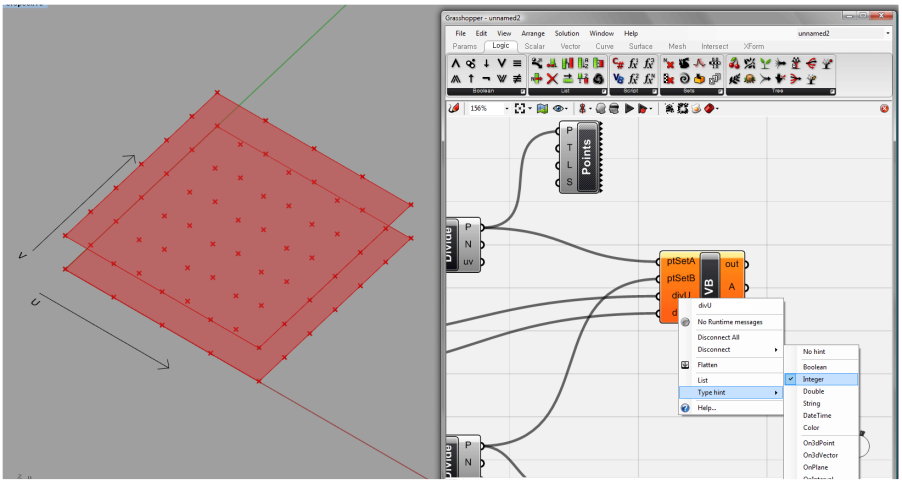
+ **Get connected**



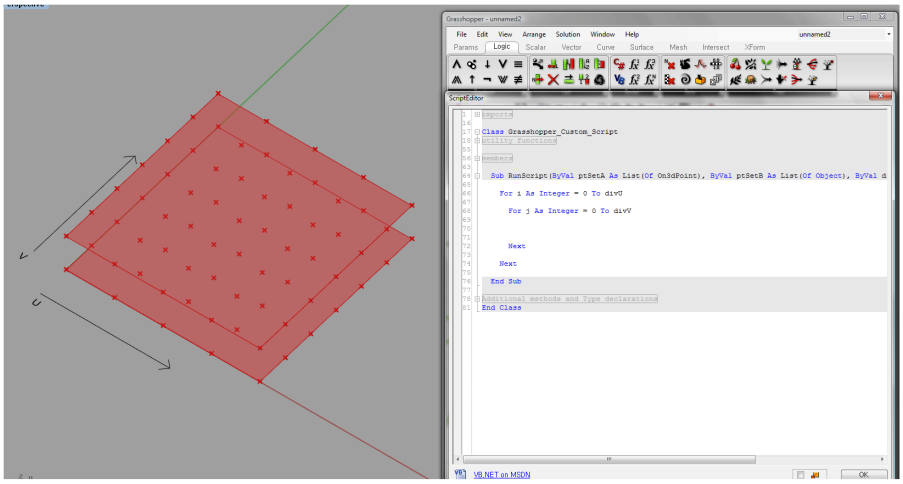
+ **Set input parameters : ptSetA & B**
  - Check 'List' - multiple point input



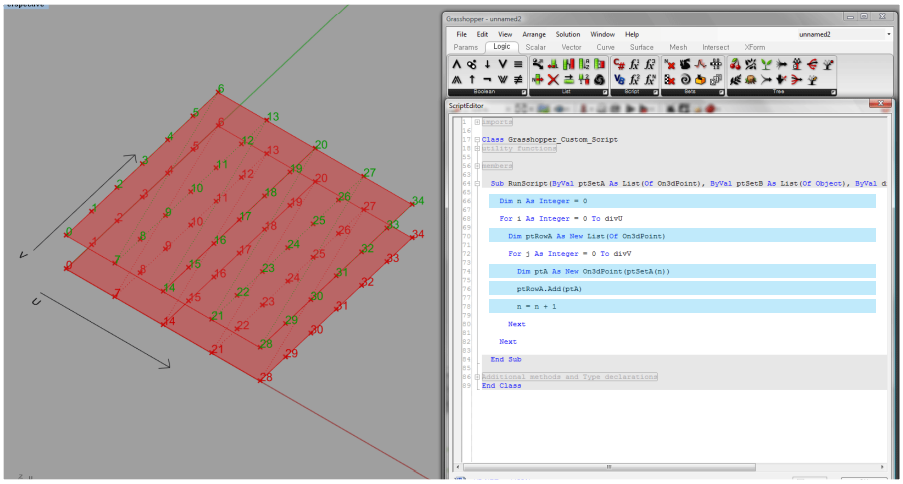
+ **Set input parameters : ptSetA & B**
  - Check ‘On3dPoint’ for data type hint



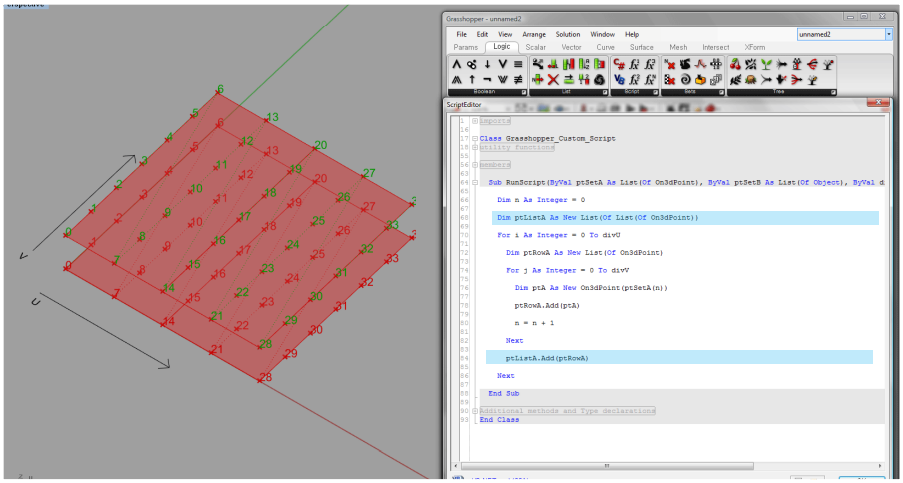
+ **Set input parameters : divU & V**
  - Check ‘Integer’



+ **Double ‘For ~ Next’**
  - To remap one dimensional linear point input data into two dimensional array or list, we will use double ‘for ~ next’ loop.

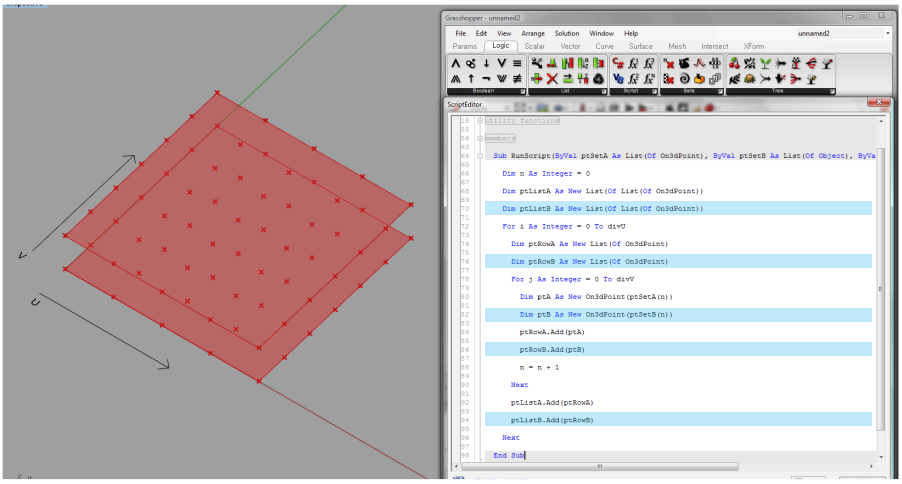


+ **Iteration for V direction(‘j’ direction, pt01 ~ pt06)**
  - Define ‘n’ as integer. (Overall index, 0 to point upper bound)
  - Define ‘ptRowA’ as list of points
  - Define ‘ptA’ as individual points
  - Assign the point ‘ptSetA(n)’, n th member of input point list, onto ‘ptA’, temporary address
  - Add the point on ‘ptRow’
  - Increase ‘n’ by 1



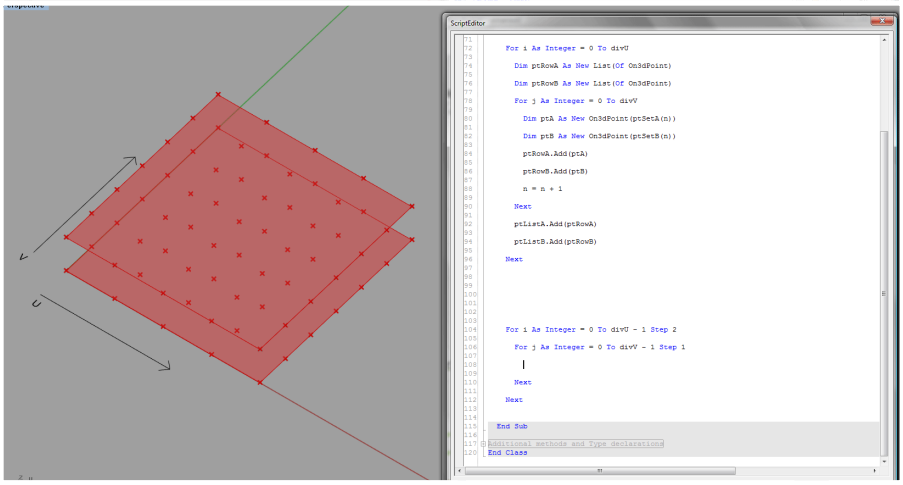
```
Imports
Class Grasshopper_Custom_Script
utility functions
members
 Sub RunScript(ByVal ptSetA As List(Of On3dPoint), ByVal ptSetB As List(Of Object), ByVal d
  Dim n As Integer = 0
  Dim ptListA As New List(Of List(Of On3dPoint))
  For i As Integer = 0 To divU
   Dim ptRowA As New List(Of On3dPoint)
   For j As Integer = 0 To divV
    Dim ptA As New On3dPoint(ptSetA(n))
    ptRowA.Add(ptA)
    n = n + 1
   Next
   ptListA.Add(ptRowA)
  Next
 End Sub
Additional methods and Type declarations
End Class
```

+ **Iteration for U direction('i' direction, pt07 ~ pt13 / pt14 ~ pt20 / ... / pt28 ~ pt34)**
  - Define 'ptListA' as list of list (not list of points)
  - Add 'ptRowA' to 'ptList'



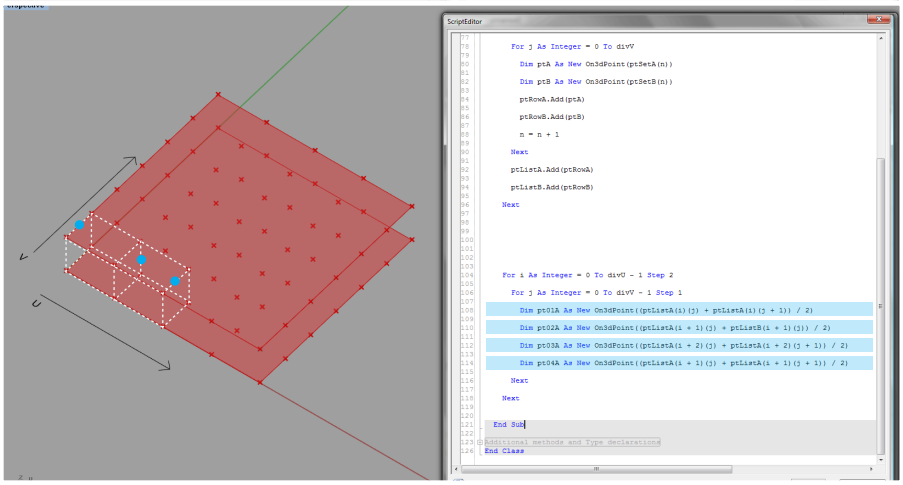
```
Utility functions
members
 Sub RunScript(ByVal ptSetA As List(Of On3dPoint), ByVal ptSetB As List(Of Object), ByVa
  Dim n As Integer = 0
  Dim ptListA As New List(Of List(Of On3dPoint))
  Dim ptListB As New List(Of List(Of On3dPoint))
  For i As Integer = 0 To divU
   Dim ptRowA As New List(Of On3dPoint)
   Dim ptRowB As New List(Of On3dPoint)
   For j As Integer = 0 To divV
    Dim ptA As New On3dPoint(ptSetA(n))
    Dim ptB As New On3dPoint(ptSetB(n))
    ptRowA.Add(ptA)
    ptRowB.Add(ptB)
    n = n + 1
   Next
   ptListA.Add(ptRowA)
   ptListB.Add(ptRowB)
  Next
 End Sub
```

+ **Duplicate for the lower surface points**
  - Duplicate codes for 'ptSetB'



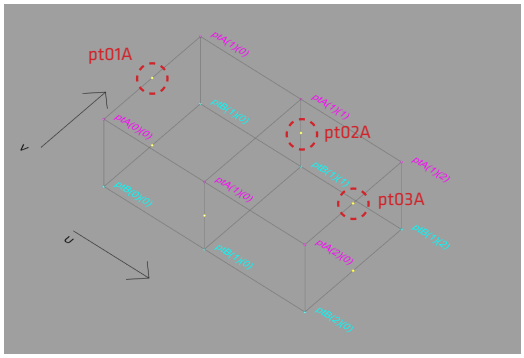
```
  For i As Integer = 0 To divU
   Dim ptRowA As New List(Of On3dPoint)
   Dim ptRowB As New List(Of On3dPoint)
   For j As Integer = 0 To divV
    Dim ptA As New On3dPoint(ptSetA(n))
    Dim ptB As New On3dPoint(ptSetB(n))
    ptRowA.Add(ptA)
    ptRowB.Add(ptB)
    n = n + 1
   Next
   ptListA.Add(ptRowA)
   ptListB.Add(ptRowB)
  Next

  For i As Integer = 0 To divU - 1 Step 2
   For j As Integer = 0 To divV - 1 Step 1

   Next
  Next
 End Sub
Additional methods and Type declarations
End Class
```

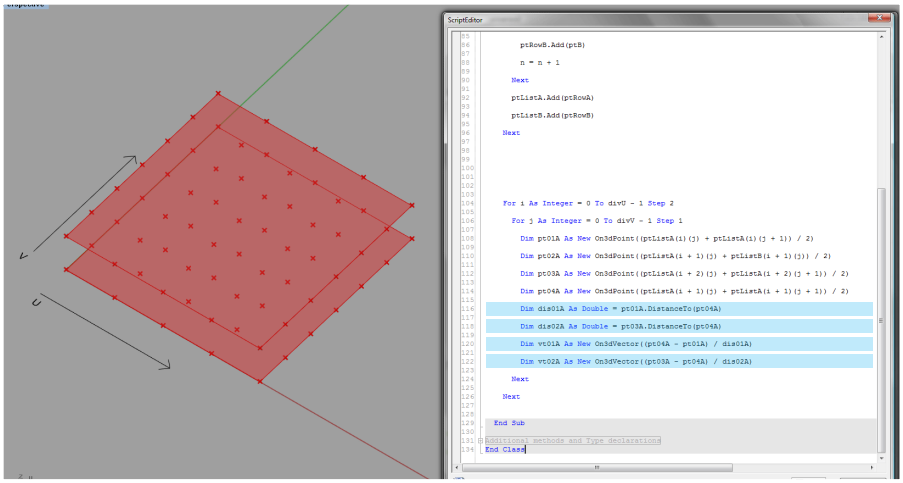
+ **Double 'For ~ Next' for point assigning**
  - Since our component is 2(U) by 1(V), set U direction step as 2



```
   For j As Integer = 0 To divV
    Dim ptA As New On3dPoint(ptSetA(n))
    Dim ptB As New On3dPoint(ptSetB(n))
    ptRowA.Add(ptA)
    ptRowB.Add(ptB)
    n = n + 1
   Next
   ptListA.Add(ptRowA)
   ptListB.Add(ptRowB)
  Next

  For i As Integer = 0 To divU - 1 Step 2
   For j As Integer = 0 To divV - 1 Step 1
    Dim pt01A As New On3dPoint((ptListA(i)(j) + ptListA(i)(j + 1)) / 2)
    Dim pt02A As New On3dPoint((ptListA(i + 1)(j) + ptListB(i + 1)(j)) / 2)
    Dim pt03A As New On3dPoint((ptListA(i + 2)(j) + ptListA(i + 2)(j + 1)) / 2)
    Dim pt04A As New On3dPoint((ptListA(i + 1)(j) + ptListA(i + 1)(j + 1)) / 2)
   Next
  Next
 End Sub
Additional methods and Type declarations
End Class
```

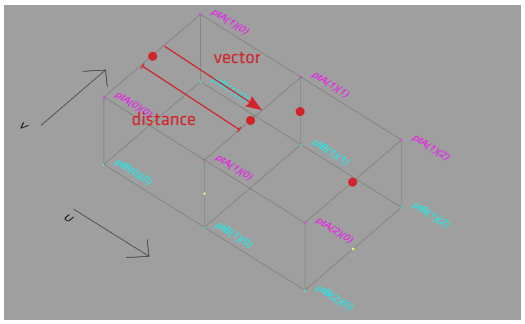
+ **Get mid points**
  - Note that 4th point is to get direction vectors

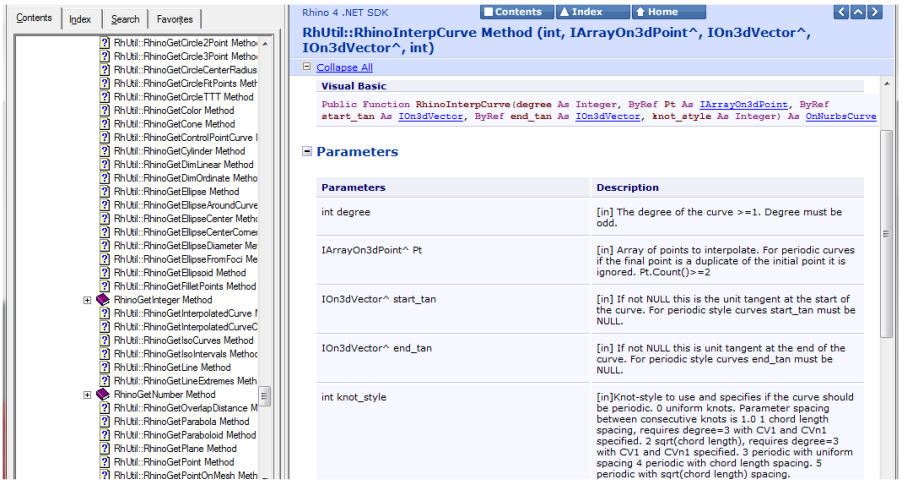




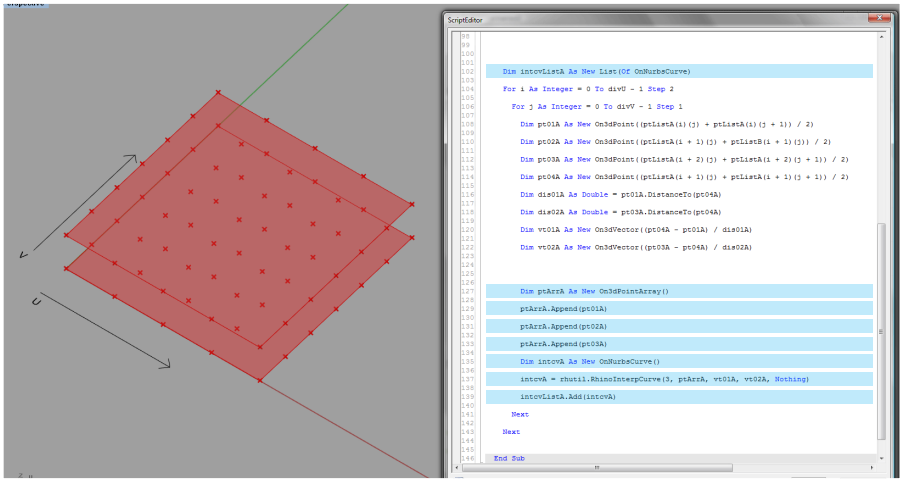
```
        ptRowB.Add(ptB)
        n = n + 1
      Next
      ptListA.Add(ptRowA)
      ptListB.Add(ptRowB)
    Next

    For i As Integer = 0 To divU - 1 Step 2
      For j As Integer = 0 To divV - 1 Step 1
        Dim pt01A As New On3dPoint((ptListA(i)(j) + ptListA(i)(j + 1)) / 2)
        Dim pt02A As New On3dPoint((ptListA(i + 1)(j) + ptListB(i + 1)(j)) / 2)
        Dim pt03A As New On3dPoint((ptListA(i + 2)(j) + ptListA(i + 2)(j + 1)) / 2)
        Dim pt04A As New On3dPoint((ptListA(i + 1)(j) + ptListA(i + 1)(j + 1)) / 2)
        Dim dis01A As Double = pt01A.DistanceTo(pt04A)
        Dim dis02A As Double = pt03A.DistanceTo(pt04A)
        Dim vt01A As New On3dVector((pt04A - pt01A) / dis01A)
        Dim vt02A As New On3dVector((pt03A - pt04A) / dis02A)
      Next
    Next

  End Sub
  Additional methods and Type declarations
End Class
```

+ **Get U directional module distances and directional vectors to unitize starting and ending vectors**



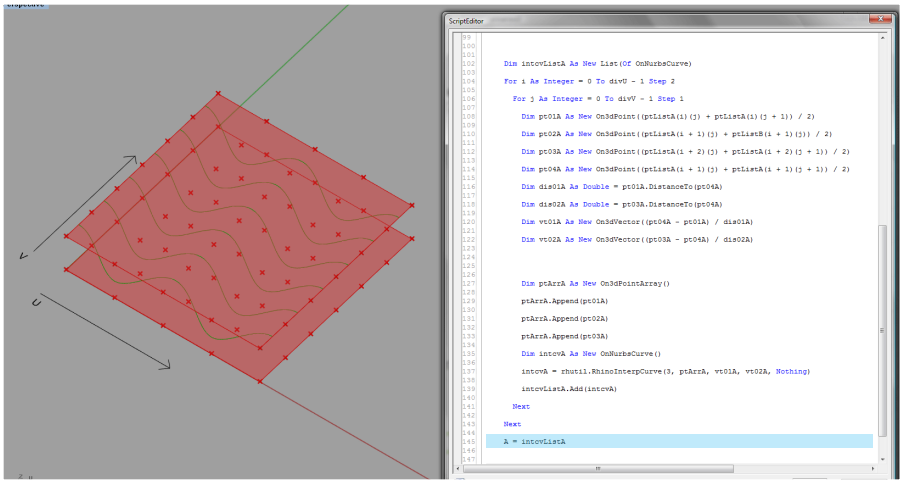


+ **Since we need to draw 'interpolate curves', look up Rhino .NET SDK help file.**
  - Rhino4DotNetPlugIns
+ **To draw interpolate curves, we should define point array first.**
  - In this case, the point array should contain three points defined in the previous step except point #4. Remember the 4 th point was just to get vectors.
  - Note that On3dPointArray is not same with Array of On3dPoint.



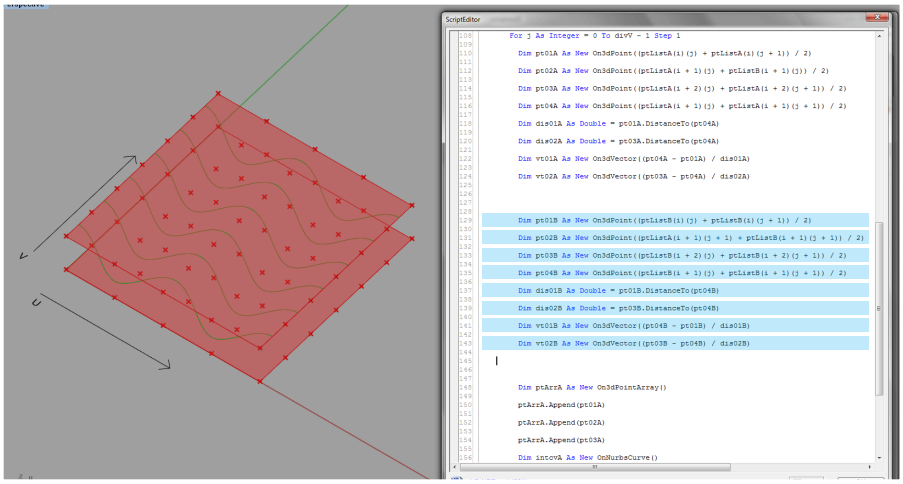
```
    Dim intcvListA As New List(Of OnNurbsCurve)
    For i As Integer = 0 To divU - 1 Step 2
      For j As Integer = 0 To divV - 1 Step 1
        Dim pt01A As New On3dPoint((ptListA(i)(j) + ptListA(i)(j + 1)) / 2)
        Dim pt02A As New On3dPoint((ptListA(i + 1)(j) + ptListB(i + 1)(j)) / 2)
        Dim pt03A As New On3dPoint((ptListA(i + 2)(j) + ptListA(i + 2)(j + 1)) / 2)
        Dim pt04A As New On3dPoint((ptListA(i + 1)(j) + ptListA(i + 1)(j + 1)) / 2)
        Dim dis01A As Double = pt01A.DistanceTo(pt04A)
        Dim dis02A As Double = pt03A.DistanceTo(pt04A)
        Dim vt01A As New On3dVector((pt04A - pt01A) / dis01A)
        Dim vt02A As New On3dVector((pt03A - pt04A) / dis02A)

        Dim ptArrA As New On3dPointArray()
        ptArrA.Append(pt01A)
        ptArrA.Append(pt02A)
        ptArrA.Append(pt03A)
        Dim intcvA As New OnNurbsCurve()
        intcvA = rhutil.RhinoInterpCurve(3, ptArrA, vt01A, vt02A, Nothing)
        intcvListA.Add(intcvA)
      Next
    Next
  End Sub
```

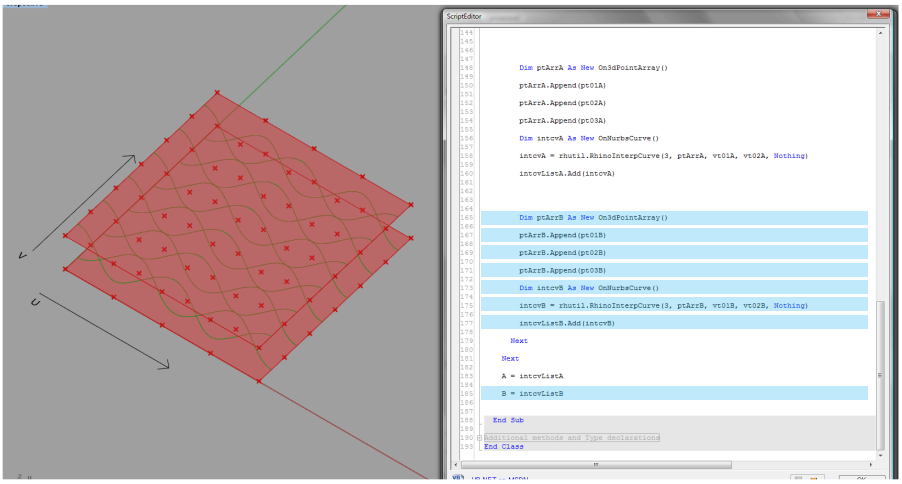
- Define new list of Nurbs curve
- Define On3dPointArray.
- Append three points to the Array
- Define interpolate curve as a nurbs curve
- Draw interpolate curve with argument (3, ptArrA, vt01A, vt02A, Nothing)
- Add the curve to interpolate curve list



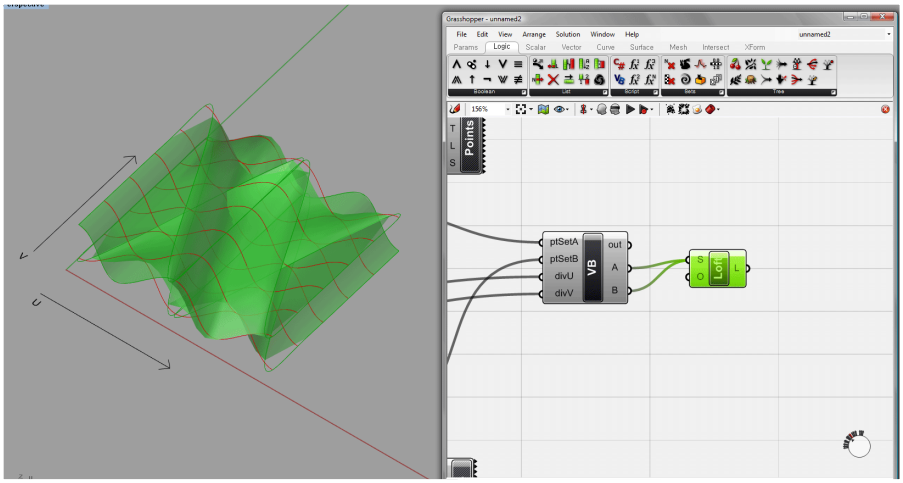
```
    Dim intcvListA As New List(Of OnNurbsCurve)
    For i As Integer = 0 To divU - 1 Step 2
      For j As Integer = 0 To divV - 1 Step 1
        Dim pt01A As New On3dPoint((ptListA(i)(j) + ptListA(i)(j + 1)) / 2)
        Dim pt02A As New On3dPoint((ptListA(i + 1)(j) + ptListB(i + 1)(j)) / 2)
        Dim pt03A As New On3dPoint((ptListA(i + 2)(j) + ptListA(i + 2)(j + 1)) / 2)
        Dim pt04A As New On3dPoint((ptListA(i + 1)(j) + ptListA(i + 1)(j + 1)) / 2)
        Dim dis01A As Double = pt01A.DistanceTo(pt04A)
        Dim dis02A As Double = pt03A.DistanceTo(pt04A)
        Dim vt01A As New On3dVector((pt04A - pt01A) / dis01A)
        Dim vt02A As New On3dVector((pt03A - pt04A) / dis02A)

        Dim ptArrA As New On3dPointArray()
        ptArrA.Append(pt01A)
        ptArrA.Append(pt02A)
        ptArrA.Append(pt03A)
        Dim intcvA As New OnNurbsCurve()
        intcvA = rhutil.RhinoInterpCurve(3, ptArrA, vt01A, vt02A, Nothing)
        intcvListA.Add(intcvA)
      Next
    Next
    A = intcvListA
```

+ **Set output as the list of interpolate curves**

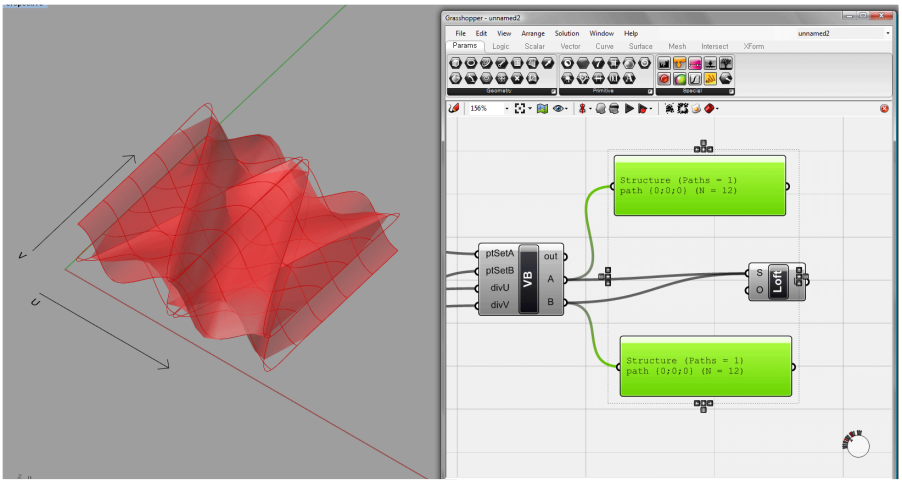




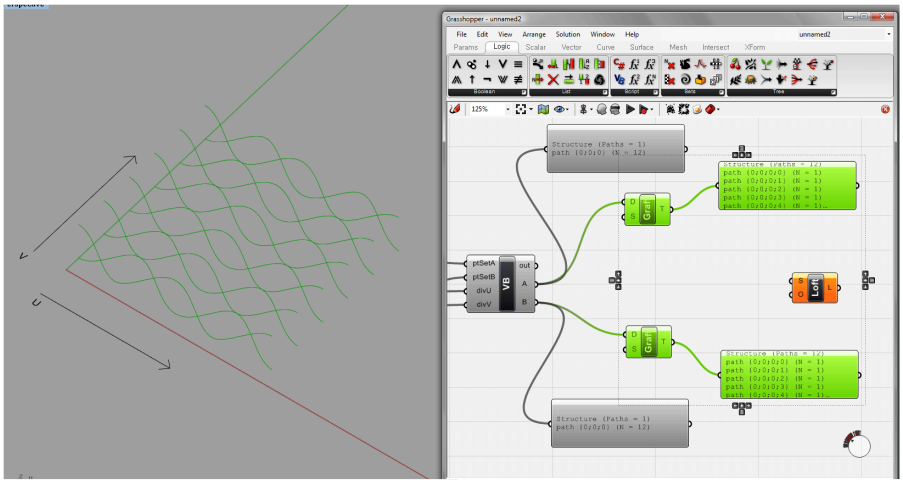
+ **Duplicate each codes for the lower surface**



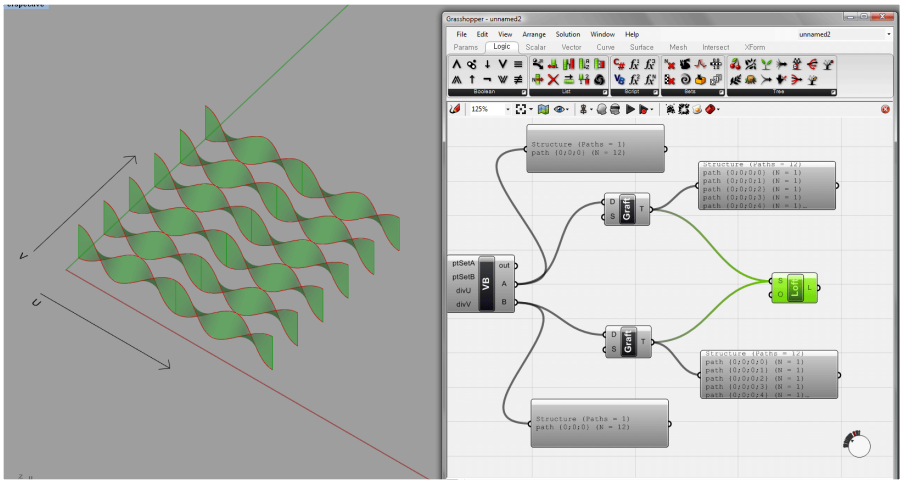
+ **Loft using two sets of interpolate curves**
  - Unexpected loft result



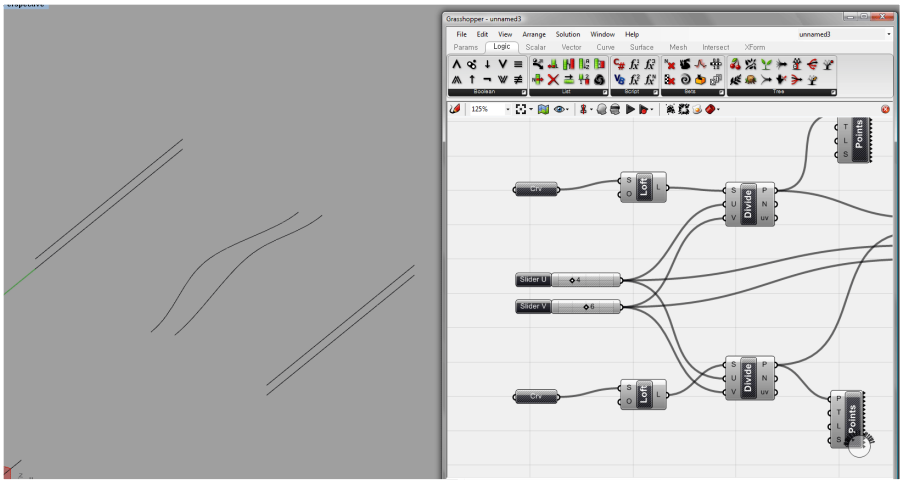
- Curves on the same data branch



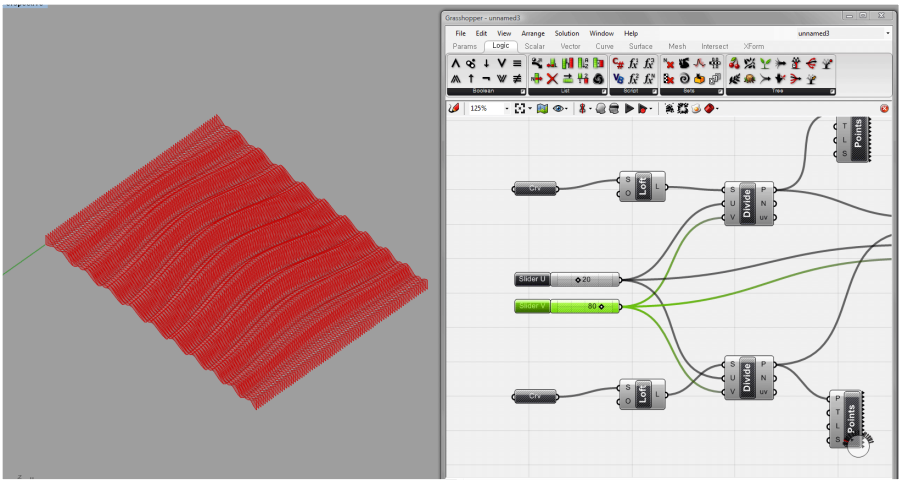
+ **Grafting**
  - Creates a new branch for every single data item.



+ **Lofting**



+ Modify input curves



+ Or control div numbers