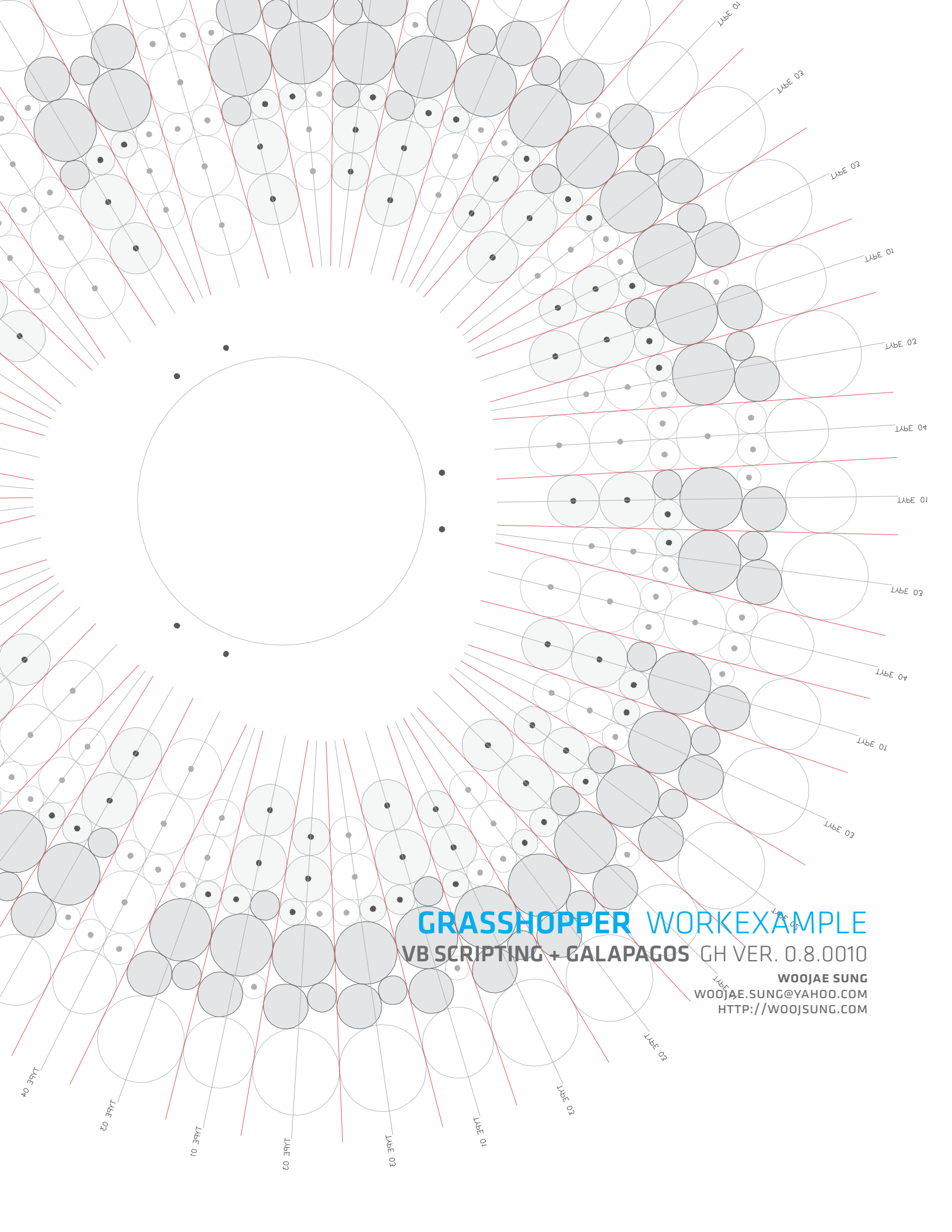


GRASSHOPPER WORKEXAMPLE

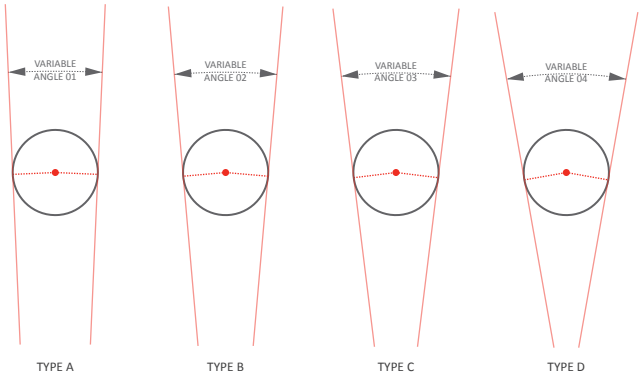
VB SCRIPTING + GALAPAGOS GH VER. 0.8.0010

WOOJAE SUNG
WOOJAE.SUNG@YAHOO.COM
HTTP://WOOJSUNG.COM

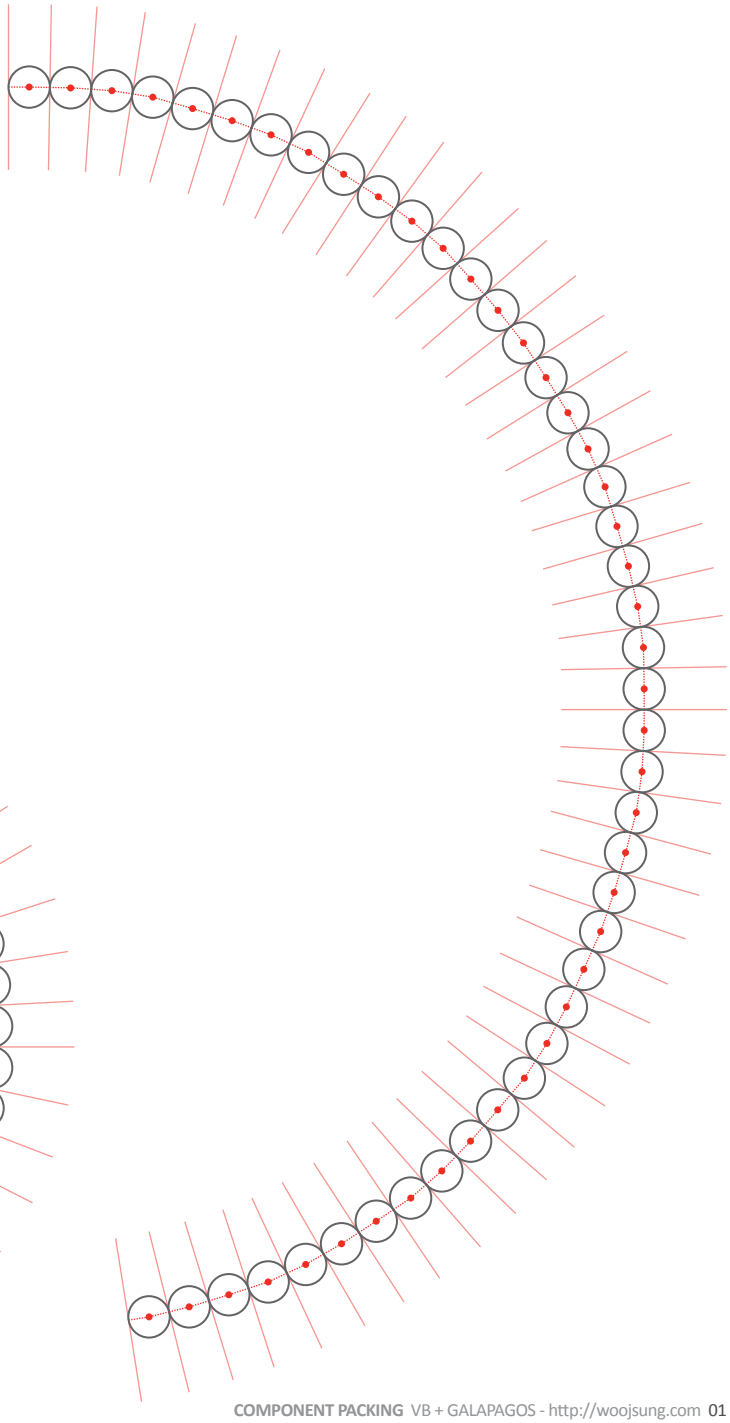
IDEA

The idea was to create four component types, and then array / pack them together. Each component has only one variable, the angle between two lines, which are represented by solid red lines tangent to a circle in the center. Each angle varies within a predefined range, for example, TYPE A's angle is in a range between 5 to 10 degrees, TYPE B's in between 7 to 12, etc.. In the second step we array them based on a certain order, which we can modify later. Then in the last step we want to minimize the sum of distance D1 and D2 so the array can be packed as tight as possible. Since we have four independent variables, it seems quite tough to get the optimum angle value for each component by moving number sliders. To solve the optimum value for each number sliders, we will use Galapagos, built-in algorithm solver in Grasshopper.

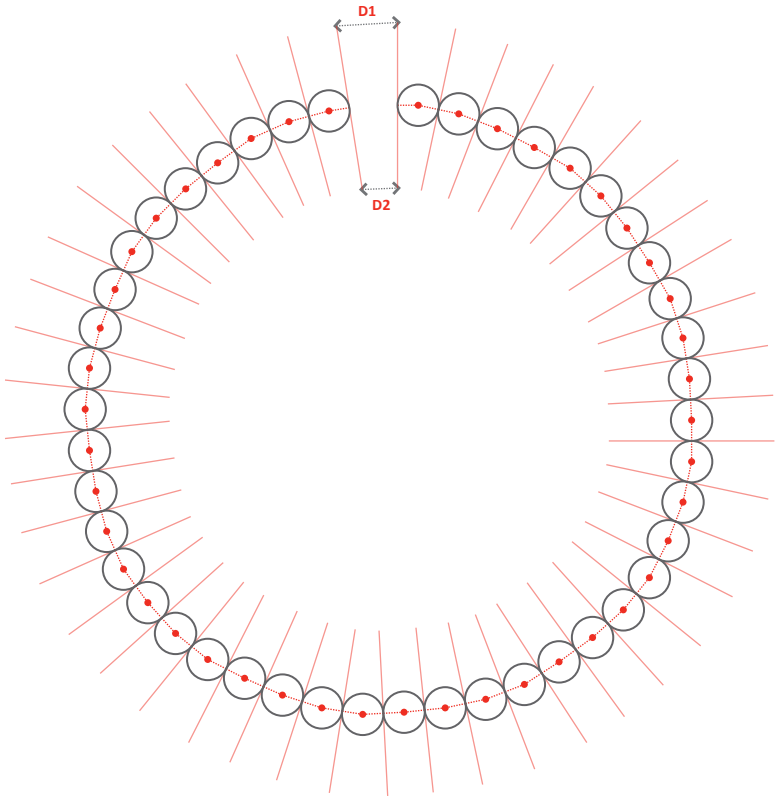
STEP01 COMPONENTS



STEP02 INITIAL ARRAY

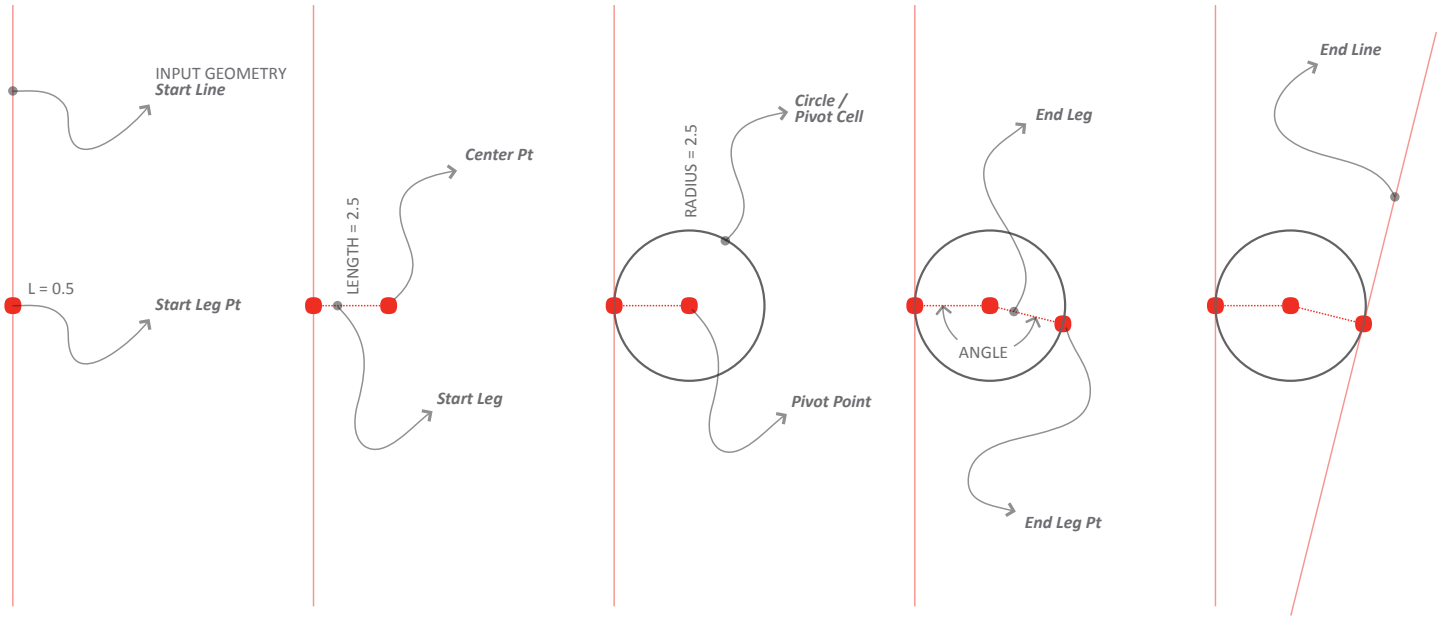


STEP03 PACKED



DEFINING A COMPONENT

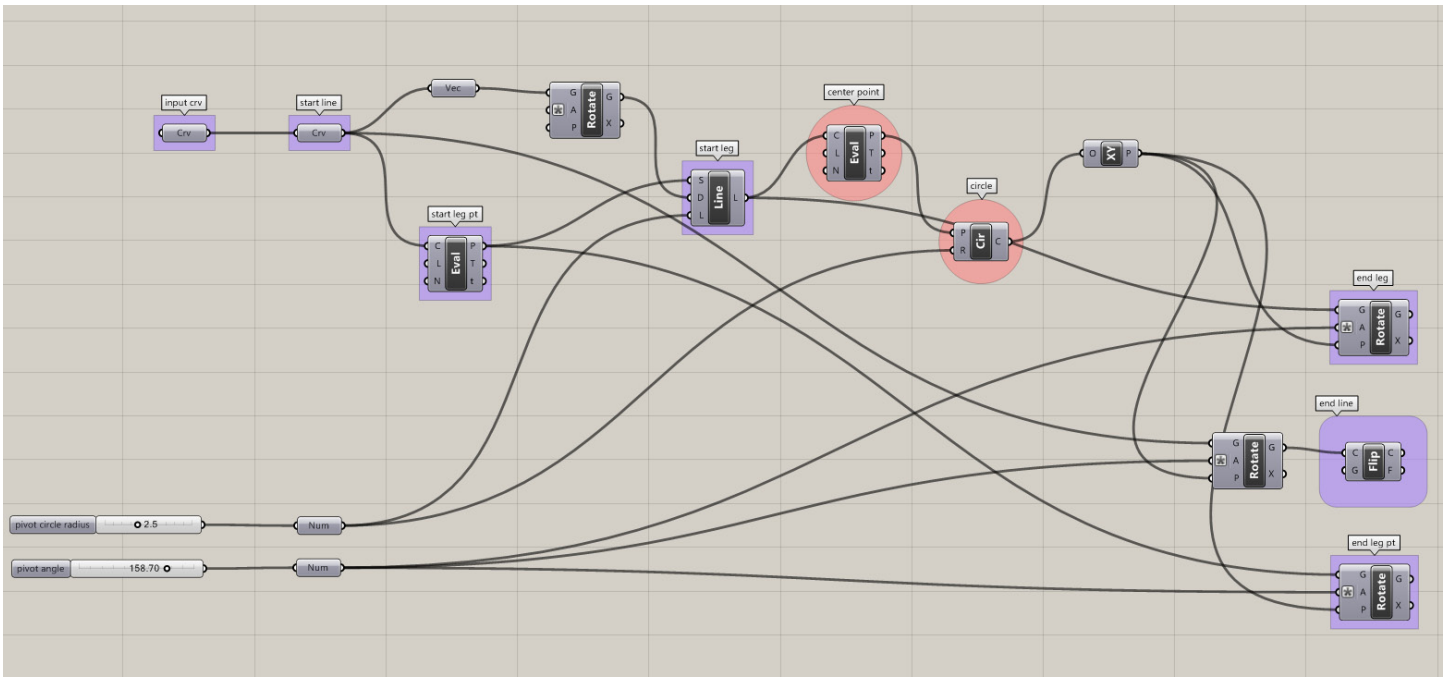
In the first place, we are going to define a component using only Grasshopper's built-in objects (then we will convert it to a custom VB scripting object). The first thing is to draw a line (we call it as Start Line) in Rhino in length of 20 and connect it to Grasshopper Curve Object. Then we get a mid point of the line to get a perpendicular line (we call it as Start Leg) in length of 2.5. Now we can get a tangential circle at the end of the perpendicular line. Then we rotate geometries including the Start Line and the Start Leg by a certain amount of angle to get the "End line" and "End Leg".



01 A CENTER POINT | 02 A PERPENDICULAR LINE | 03 A CIRCLE | 04 ROTATE BY ANGLE | 05 ROTATE BY ANGLE

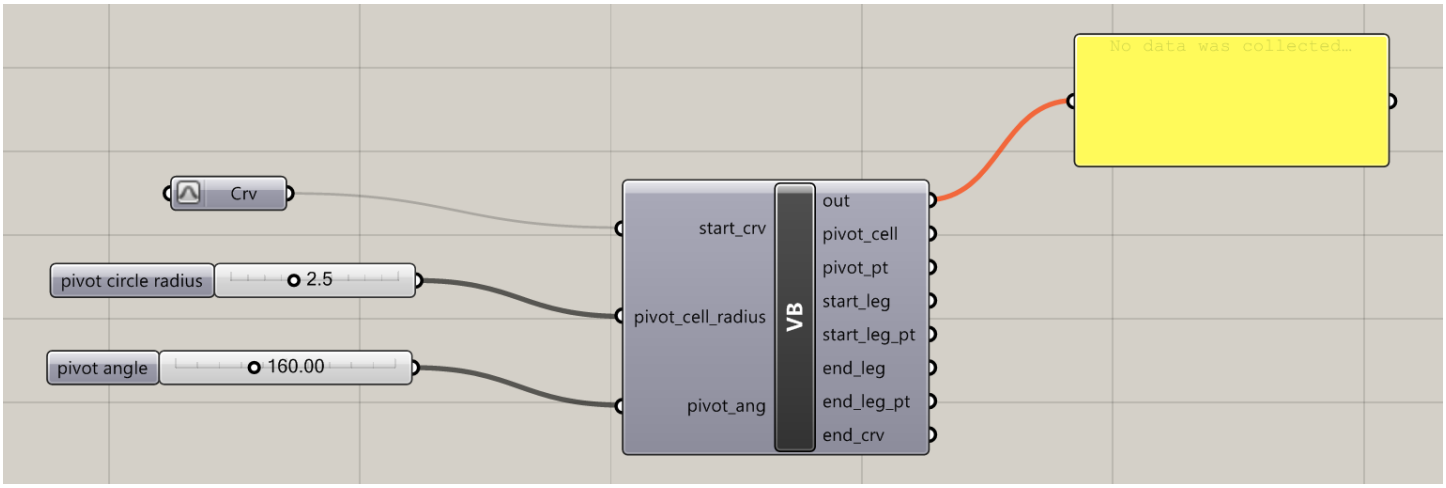
GH DEFINITION

Below is the screenshot of the definition (001 component GH objects.ghx).

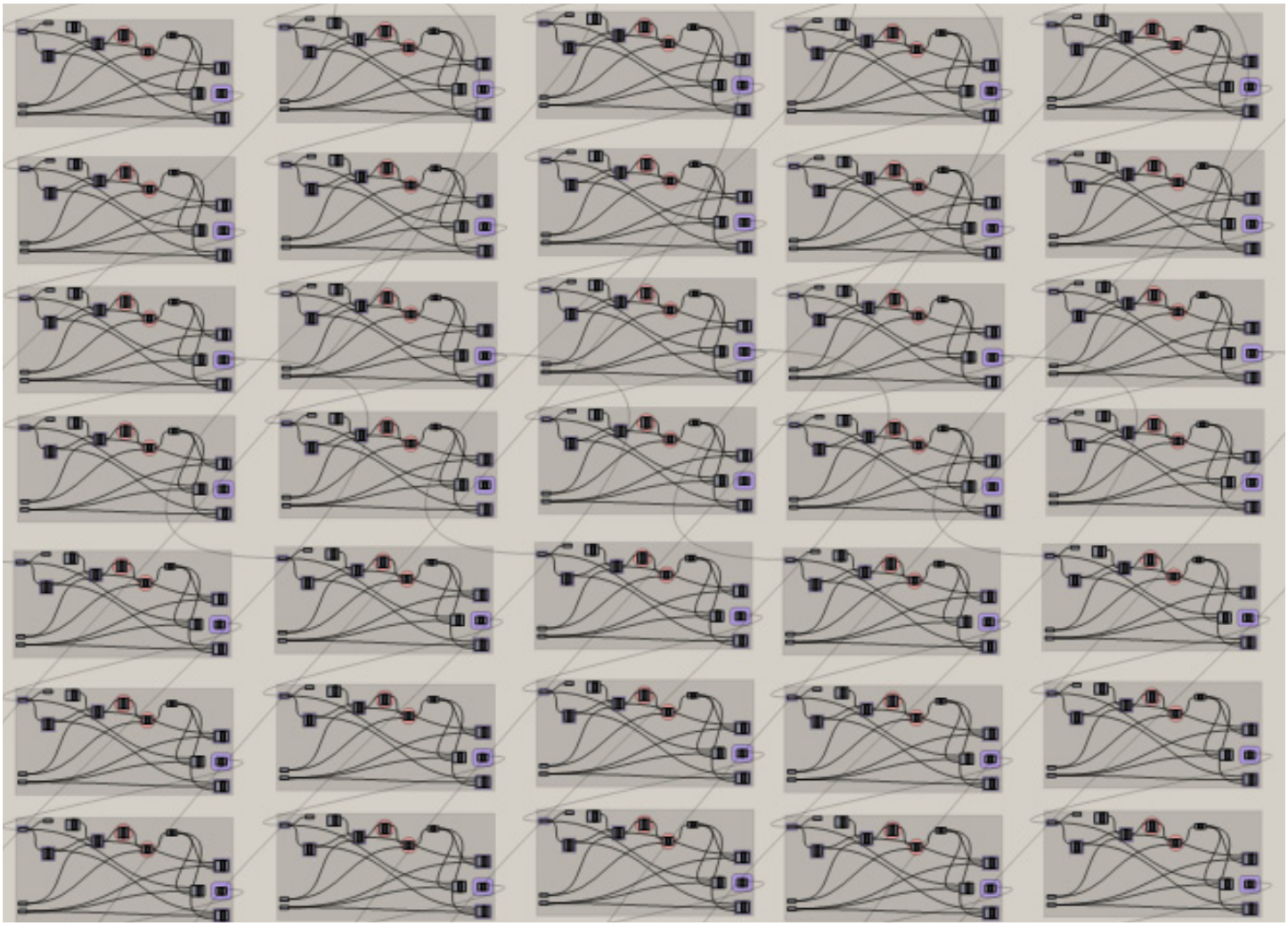


CUSTOM VB COMPONENT

As you can see from the screenshot below, the custom VB component has three inputs and seven outputs, which are pretty familiar with us. If not, refer to the component process diagram in the previous page. What it does is basically same thing with the GH definition in the previous page. We supply a curve as the initial input geometry, radius of a circle as fixed number (though it is from a number slider), and pivot angle as a variable. Then we get series of outputs such as End Curve, End Leg, etc.. (002 component VB.ghx)

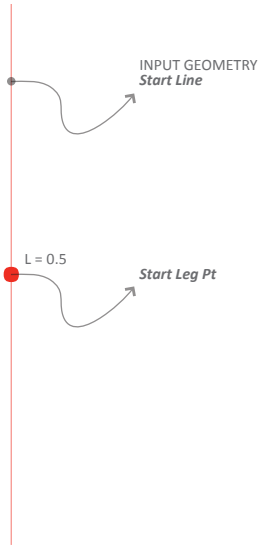


Why do we use VB component over GH Objects? If you don't bother with the length definition like below, you don't have to. However, you will find it much easier to do whatever you want to do as you get to know about VB Scripting better (Such a fantastic excuse!).



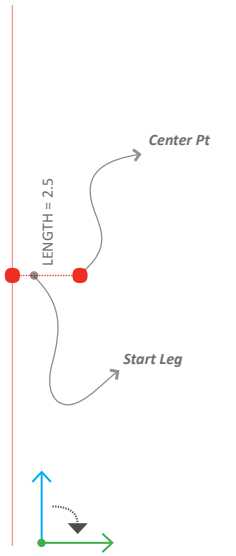
CODE REVIEW

Double click on the VB object, you will get a new window to edit VB script.



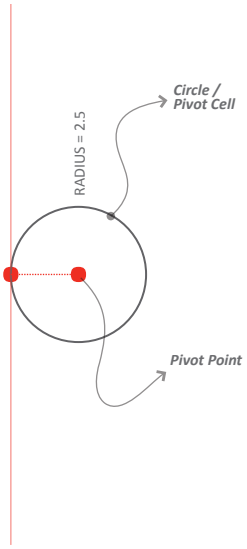

```
'defines start_leg_pt ////////////////////////////////////////////////////////////////////////////////////////////

Dim s_pt As New point3d(start_crv.PointAt(0.5))

start_leg_pt = s_pt
```

- Any line that starts with ' means that the line is just a note / comment. This does not do anything.
- The second line defines variable "s_pt" as a new point, and assign a mid point of the input curve ("start_crv") to it. Then assign "s_pt" to "start_leg_pt", the output of the VB object.



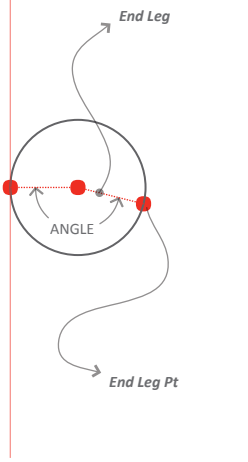

```
'defines start_leg ////////////////////////////////////////////////////////////////////////////////////////////

Dim start_crv_start As New Point3d(start_crv.PointAt(0.0))
Dim start_crv_end As New Point3d(start_crv.pointat(1.0))

Dim original_vector As New Vector3d(start_crv_start - start_crv_end)

Dim rotation_ang_start_leg As Double = math.PI / 2
Dim rotation_axis_start_leg As New Vector3d(0, 0, 1)

original_vector.Rotate(rotation_ang_start_leg, rotation_axis_start_leg)

Dim line_start As New line(start_leg_pt, original_vector, pivot_cell_radius)

start_leg = line_start
```

- Define two points variables one at the start and the other at the end of the input curve ("start_crv").
- Then defines a vector, "original_vector", using two points (We don't care about the actual length of the vector).
- We need to rotate the vector(blue arrow) 90 degrees clock wise to get a perpendicular vector (green arrow) in order to draw a perpendicular line, "Start Leg".
- To use 'rotate method', we first need to define rotation axis and angle. The angle should be 90 degrees (PI/2 in radians). And Z-direction vector will serve as a rotation axis.
- Apply 'rotate method' to the existing vector. Methods can be applied followed by dot connector. Rotate method consist of two input variables such as rotation angle and axis. For more information, visit http://www.rhino3d.com/5/rhinocommon/
- Then we define a line, "line_start", by three variables; start point of line segment, direction of line segment and length of line segment.
- Assign the line to "start_leg", the output of the VB object.

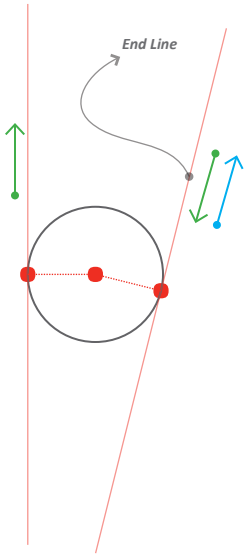



```
'defines pivot_pt ////////////////////////////////////////////////////////////////////////////////////////////

Dim p_pt As New point3d(line_start.PointAt(1.0))

pivot_pt = p_pt

'defines pivot_cell ////////////////////////////////////////////////////////////////////////////////////////////

Dim pivot_circle As New Circle(line_start.PointAt(1.0), pivot_cell_radius)

pivot_cell = pivot_circle
```

- Define “p_pt” as a new point object, then assign the end point of “start leg (line_start)” to it.
- Then we assign “p_pt” to ““pivot_pt”, the output of the VB object.
- Define “pivot_circle” as a new circle object, then assign a circle with given center(“pivot_pt”) and radius(“pivot_cell_radius”, the input of the VB object).
- Assign it to “pivot_cell”, the output of the VB object.



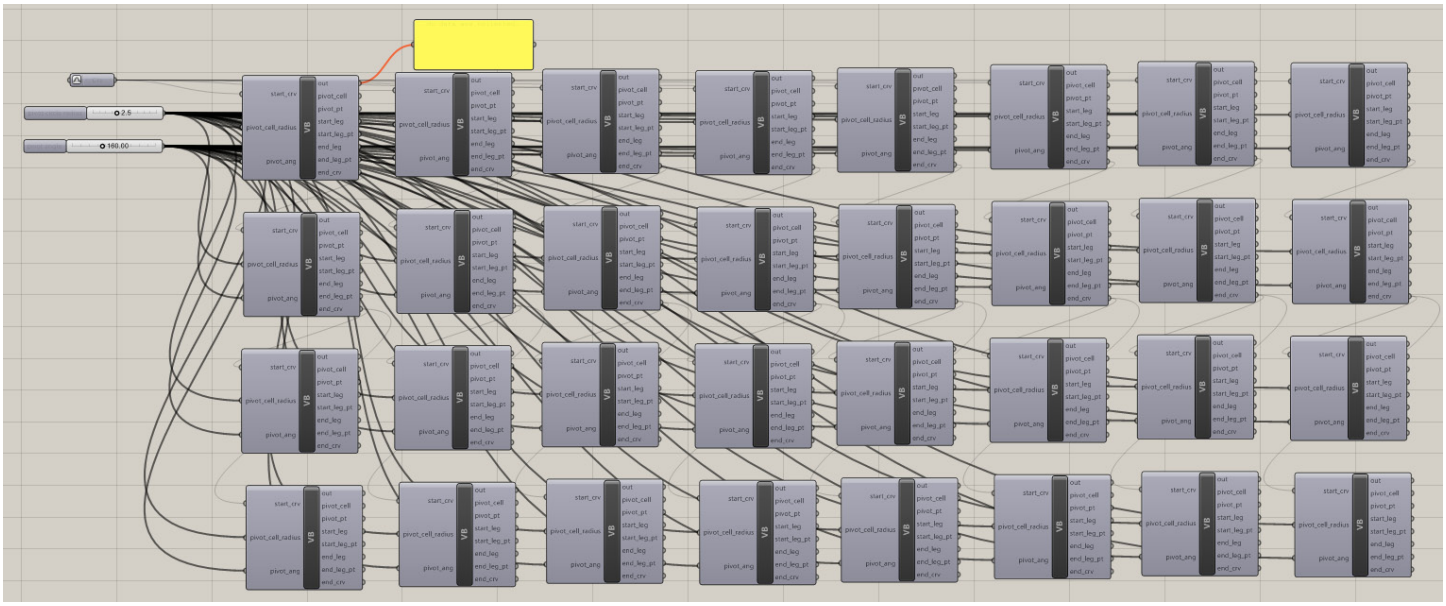

```
'defines end_leg ////////////////////////////////////////////////////////////////////////////////////////////

Dim rot As transform = transform.Rotation(pivot_ang * math.PI / 180, vector3d.ZAxis, line_start.PointAt(1.0))

Dim rotated_leg As line = line_start

rotated_leg.Transform(rot)

end_leg = rotated_leg

'defines end_leg_pt ////////////////////////////////////////////////////////////////////////////////////////////

s_pt.Transform(rot)

end_leg_pt = s_pt

'defines end_crv ////////////////////////////////////////////////////////////////////////////////////////////

Dim st_line As line = start_crv

st_line.Transform(rot)

Dim end_line_st_pt As New Point3d(st_line.PointAt(1.0))

Dim end_line_end_pt As New Point3d(st_line.PointAt(0.0))

Dim end_line As New line(end_line_st_pt, end_line_end_pt)

end_crv = end_line
```



- In this step, we will rotate “start line”, “start leg” and “start leg point” in a given angle, then assign them to the corresponding output variable: “end line”, “end leg”, and “end leg point”.
- Define a rotation matrix. This will be handy since we will need to repeat the exact same transformation couple of times. A rotation matrix consists of angle, axis, and center point of rotation. Supply corresponding value from previously defined variable.
- Define “rotated_leg” as a line and assign “line_start” to it. Then, rotate it by the rotation matrix.
- Do the same thing for leg point and end line.
- Unlike others, “end line / start line” is direction-sensitive. When we rotate “start line”, its direction will be reversed, so we need to flip the line.
- Get the start and end point of the rotated line, “st_line”, and make another line, “end_line”, out of the points. Note how we supply start and end points to reverse the line.

USING A SUBROUTINE (FUNCTION) IN VB COMPONENT

Now that we have a working custom object, we can copy and paste it multiple times to array all the components. However, copy and paste will cause the same problem that we had before. (003 subroutine component VB.ghx)



We can be smart to use a subroutine in VB scripting. The idea is to define a subroutine, which is kind of a custom function within VB script, and call it whenever we need it, just as we use "line" function.

```
line(start point, direction vector, distance)
```

Let's say that we want to make a subroutine called "FunctionA" which do something with two inputs and one output. Then one thing we should do is to define how it works, and the other is to call it and ask it to do whatever it supposes to do. The second part should be somewhere in VB scripting area, and the first part should be in "custom additional code" space (not sure how scripters call these). Also note that when we define a subroutine, "ByVal" means that it is an input variable, and "ByRef" means it is an output variable.

```
FunctionA (ByVal input1, ByVal input2, ByRef output1)
```

```
'<Custom additional code>
Sub FunctionA (ByVal input1, ByVal input2, ByRef output1)

        ......

End Sub
'</Custom additional code>
```

Let’s open up the script editor and copy from “Private Sub ~” to “End Sub” as is shown below.

```
38
39  Members
52
53  Utility functions
79
80    /**/
85    Private Sub RunScript(ByVal start_crv As Line, ByVal pivot_cell_radius As Double, ByVal pivot_ang As Double, ByRef pivot_cell As Object, ByR
86
87      'defines start_leg_pt  ////////////////////////////////////////////////////////////////////////////////////////////////
88
89      Dim s_pt As New point3d(start_crv.PointAt(0.5))
90
91      start_leg_pt = s_pt
92
93      'defines start_leg  ////////////////////////////////////////////////////////////////////////////////////////////////
94
95      Dim start_crv_start As New Point3d(start_crv.PointAt(0.0))
96      Dim start_crv_end As New Point3d(start_crv.pointat(1.0))
97
98      Dim original_vector As New Vector3d(start_crv_start - start_crv_end)
99
100     Dim rotation_ang_start_leg As Double = math.PI / 2
101     Dim rotation_axis_start_leg As New Vector3d(0, 0, 1)
102
103     original_vector.Rotate(rotation_ang_start_leg, rotation_axis_start_leg)
104
105     Dim line_start As New line(start_leg_pt, original_vector, pivot_cell_radius)
106
107     start_leg = line_start
108
109     'defines pivot_pt  ////////////////////////////////////////////////////////////////////////////////////////////////
110
111     Dim p_pt As New point3d(line_start.PointAt(1.0))
112
113     pivot_pt = p_pt
114
115     'defines pivot_cell  ////////////////////////////////////////////////////////////////////////////////////////////////
116
117     Dim pivot_circle As New Circle(line_start.PointAt(1.0), pivot_cell_radius)
118
119     pivot_cell = pivot_circle
120
121     'defines end_leg  ////////////////////////////////////////////////////////////////////////////////////////////////
122
123     Dim rot As transform = transform.Rotation(pivot_ang * math.PI / 180, vector3d.ZAxis, line_start.PointAt(1.0))
124
125     Dim rotated_leg As line = line_start
126
127     rotated_leg.Transform(rot)
128
129     end_leg = rotated_leg
130
131     'defines end_leg_pt  ////////////////////////////////////////////////////////////////////////////////////////////////
132
133     s_pt.Transform(rot)
134
135     end_leg_pt = s_pt
136
137     'defines end_crv  ////////////////////////////////////////////////////////////////////////////////////////////////
138
139     Dim st_line As line = start_crv
140
141     st_line.Transform(rot)
142
143     Dim end_line_st_pt As New Point3d(st_line.PointAt(1.0))
144
145     Dim end_line_end_pt As New Point3d(st_line.PointAt(0.0))
146
147     Dim end_line As New line(end_line_st_pt, end_line_end_pt)
148
149     end_crv = end_line
150
151
152   End Sub
```

Then paste it in between ‘<Custom additional code> and ‘</Custom additional code>

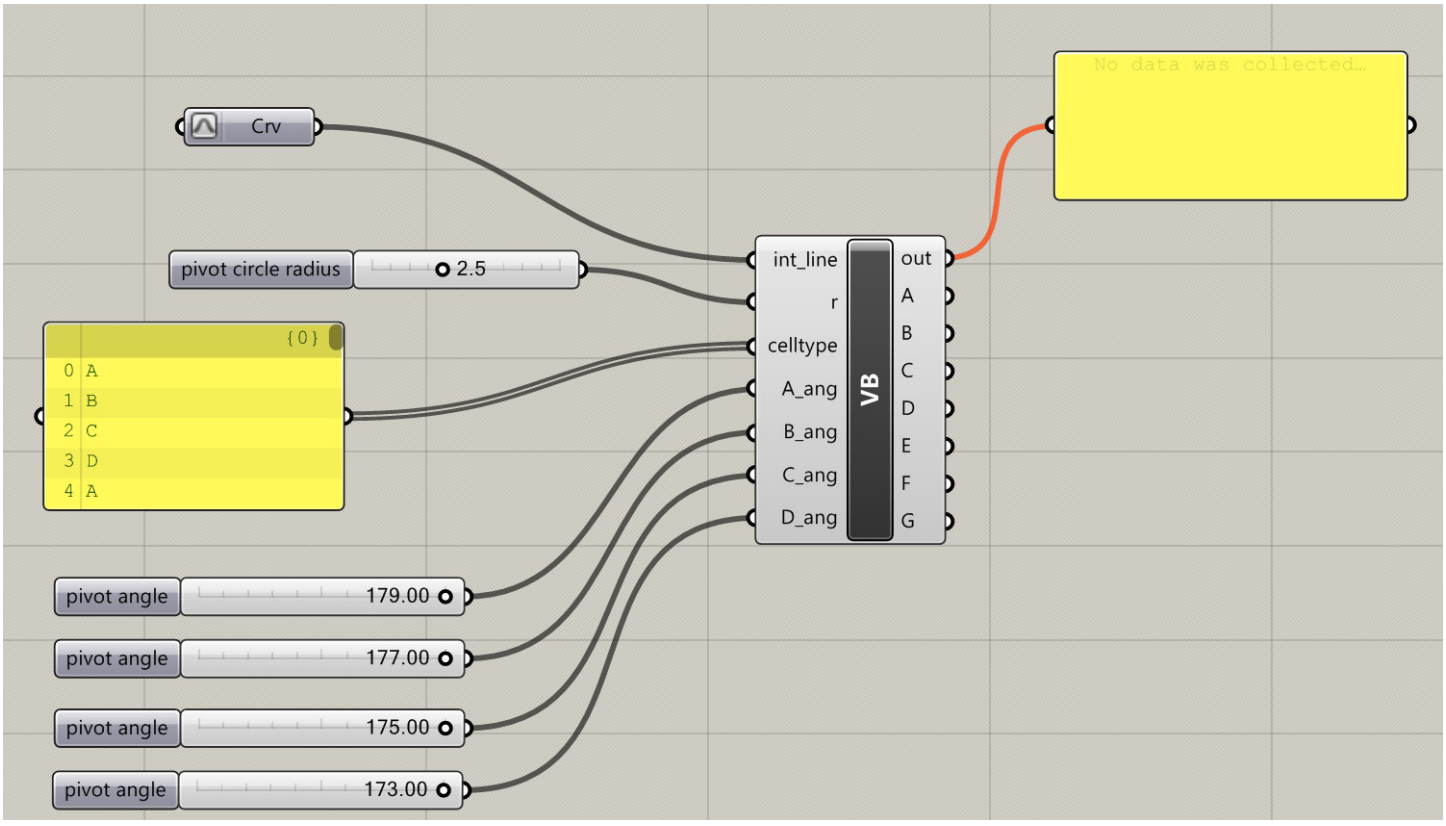
```
153
154   '<Custom additional code>
155
156   '</Custom additional code>
157
158 End Class
```

Then change the first part of the script as shown below. In this case, we call the subroutine as “component”.

```
182   End Sub
183
184   /**/
185
186   Private Sub RunScript(ByVal start_crv As Line, ByVal pivot_cell_radius As Double, ByVal pivot_ang As Double, ByRef pivot_cell As Object, ByR
187
188     'defines start_leg_pt  ////////////////////////////////////////////////////////////////////////////////////////////
189
190     Dim s_pt As New point3d(start_crv.PointAt(0.5))
191
```

```
182   End Sub
183
184   /**/
185
186   Sub component(ByVal start_crv As Line, ByVal pivot_cell_radius As Double, ByVal pivot_ang As Double, ByRef pivot_cell As Object, ByRef pivot
187
188     'defines start_leg_pt  ////////////////////////////////////////////////////////////////////////////////////////////
189
190     Dim s_pt As New point3d(start_crv.PointAt(0.5))
191
```

And we need to change inputs / outputs as shown below.
We have same inputs such as input curve, circle radius, and pivot angles. In the mean time, we also need an additional string input that defines component type (A/B/C/D).



CODE REVIEW

Double click on the VB component, you will get a new window to edit VB script.

```
'defines output parameters ////////////////////////////////////////////////////////////////////////////////

Dim pivot_cell_list As New List(Of circle)
Dim pivot_pt_list As New List(Of point3d)
Dim start_leg_list As New List(Of Line)
Dim start_leg_pt_list As New List(Of point3d)
Dim end_leg_list As New List(Of Line)
Dim end_leg_pt_list As New List(Of point3d)
Dim end_crv_list As New List(Of Line)
```

- The first part defines main outputs. Since we want to visualize all the components, we should define the variables in the form of list.

```
'defines output parameters of "component" subroutine ////////////////////////////////////////////////////////

Dim pivot_cell As circle
Dim pivot_pt As point3d
Dim start_leg As line
Dim start_leg_pt As point3d
Dim end_leg As line
Dim end_leg_pt As point3d
Dim end_crv As line
```

- Those variables are the output of "component" subroutine. If we do not define these before we call "component" subroutine, Grasshopper will spit out error messages. Unlike outputs, we don't have to worry about inputs of the subroutine, because they are already defined within the subroutine.

```
'defines rotation_ang ////////////////////////////////////////////////////////////////////////////////

Dim rotation_ang As Double
```

- This defines "rotation_angle". Why? Because this angle may vary depend on the type of component (A/B/C/D). So we first define the angle as an empty variable, then we assign specific value later based on component types.

```
'checks component types and computes output based on the types  ////////////////////////////////////////////////

For i As Integer = 0 To celltype.Count - 1

 If i = 0 Then

  If celltype(i) = "A" Then

   rotation_ang = A_ang

  Else If celltype(i) = "B" Then

   rotation_ang = B_ang

  Else If celltype(i) = "C" Then

   rotation_ang = C_ang

  Else If celltype(i) = "D" Then

   rotation_ang = D_ang

  End If

  component(int_line, r, rotation_ang, pivot_cell, pivot_pt, start_leg, start_leg_pt, end_leg, end_leg_pt, end_crv)
```

- Now we multiply component based on component types supplied in the form of a list of strings (A/B/C/D).
- Do iteration for a certain amount of times depends on the length of characters list. That is what "For i As ~ " do.
- If this is the first path of the iteration(if i = 0 then), we use "int_line" (one that we made in Rhino and supplied as an input geometry of VB object) as our input geometry for the subroutine. Else, we use "end_crv" from the previous iteration as the input curve.
- Assign type specific rotation angle to variable "rotation_ang". A_ang / B_ang / C_ang / D_ang are input parameters that we can adjust in number sliders.
- Then finally we call the subroutine, "component". Note that we supply "int_line"(input geometry), "r"(radius of a circle) and "rotation_ang" and we get "pivot_cell", "start_leg", "start_leg_pt", "end_leg", "end_leg_pt", and "end_crv".
- As is mentioned, "end_crv" will be an input geometry for the next iteration.

```
 Else

  If celltype(i) = "A" Then

   rotation_ang = A_ang

  Else If celltype(i) = "B" Then

   rotation_ang = B_ang

  Else If celltype(i) = "C" Then

   rotation_ang = C_ang

  Else If celltype(i) = "D" Then

   rotation_ang = D_ang

  End If

  int_line = end_crv

  component(int_line, r, rotation_ang, pivot_cell, pivot_pt, start_leg, start_leg_pt, end_leg, end_leg_pt, end_crv)

 End If
```

- Note that "end_crv" in the current iteration is assigned as "int_line" for the next iteration.

```
  pivot_cell_list.Add(pivot_cell)
  pivot_pt_list.Add(pivot_pt)
  start_leg_list.Add(start_leg)
  start_leg_pt_list.Add(start_leg_pt)
  end_leg_list.Add(end_leg)
  end_leg_pt_list.Add(end_leg_pt)
  end_crv_list.Add(end_crv)

Next
```

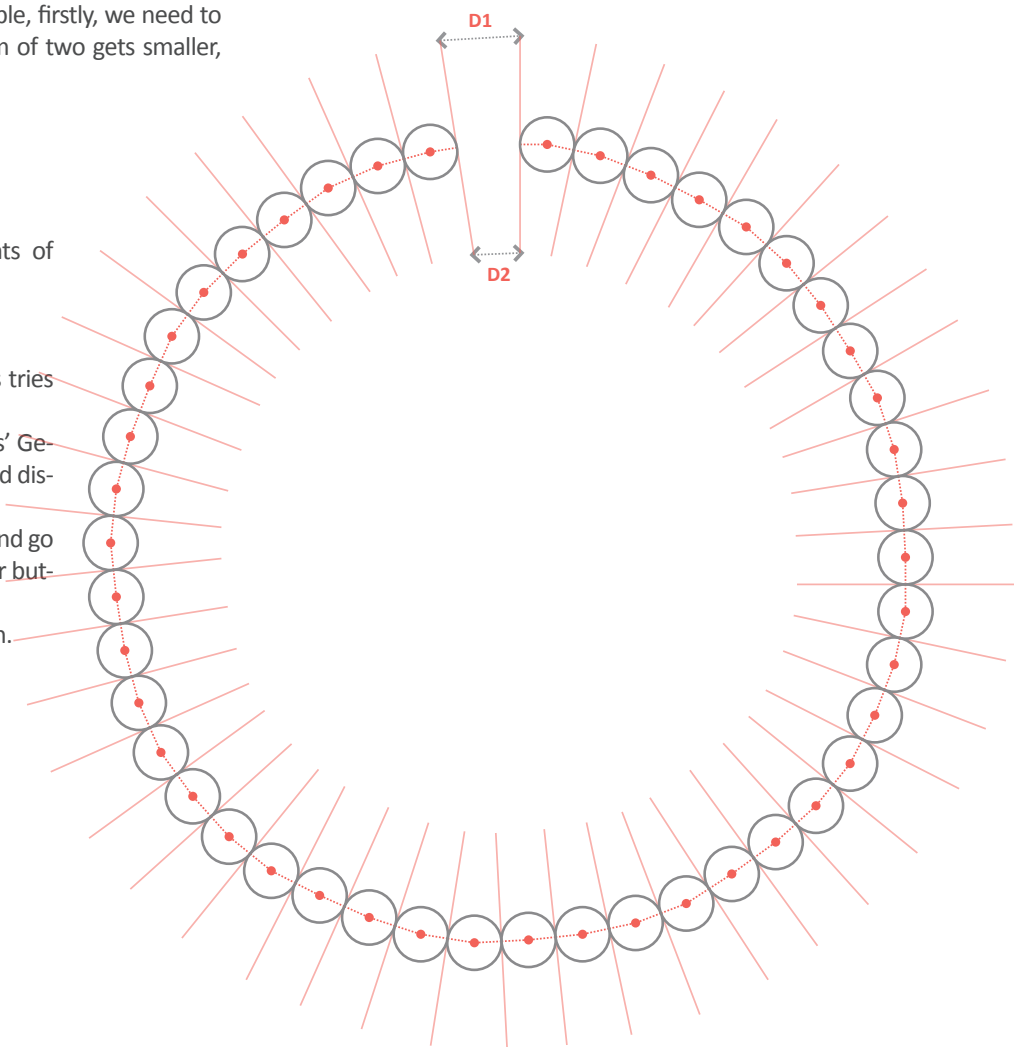
- Add every single output for the current iteration to the lists, before the next iteration.

```
'output ////////////////////////////////////////////////////////////////////////////////////////////////////////////

A = pivot_cell_list
B = pivot_pt_list
C = start_leg_list
D = start_leg_pt_list
E = end_leg_list
F = end_leg_pt_list
G = end_crv_list
```

- Put list variables to the corresponding output parameters so we can see the result in Rhino viewport.

OPTIMUM SOLUTION BY GALAPAGOS

To pack the components as tight as possible, firstly, we need to define two distance variables. As the sum of two gets smaller, the components will be packed tighter.
(004 galapagos.ghx)



- We get the first and last lines.
- Then we get start and end points of them
- Calculate distances, D1, D2.
- Add two values.
- Inverse the value, since Galapagos tries to find the maximum value.
- Connect angle sliders to Galapagos' Genome tab, and connect the inversed distance value to Fitness tab.
- Double click on Galapagos object and go to the solver tab and hit start solver button on top.
- Wait until it finished the calculation.