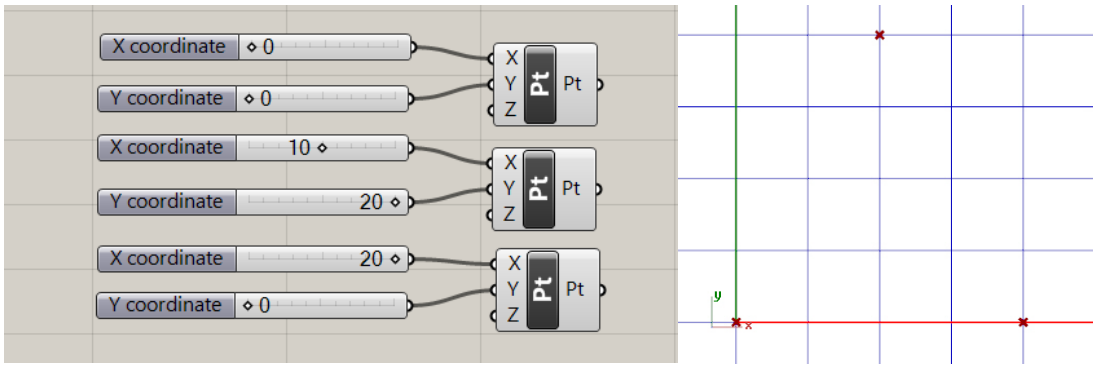
S206E057 -- Lecture 12, 10/13/2023, Grasshopper 2D – an overview

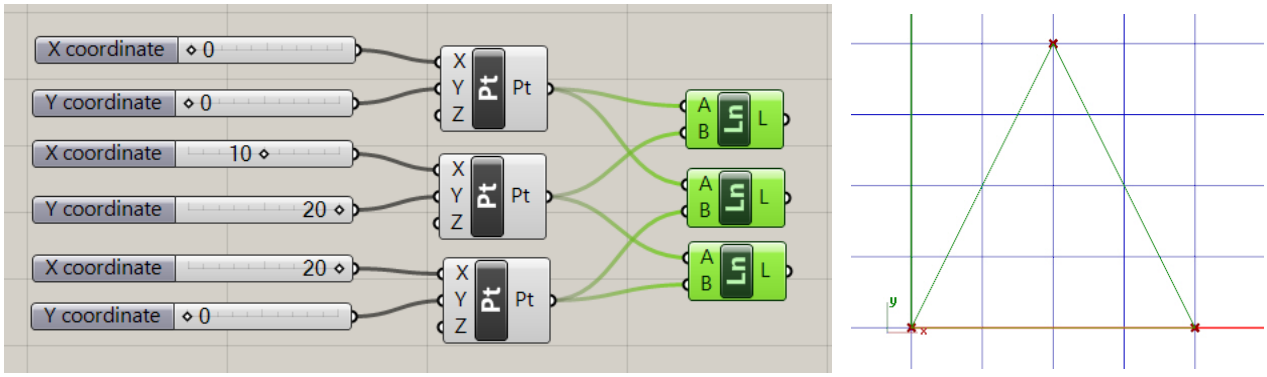
Grasshopper 2D exercises on recursive generation of 2D curves. Recursion is a programming and mathematical concept that is the process in which a function calls itself directly or indirectly. Commonly used in computer science and mathematics to solve problems that exhibit repetitive, self-similar structures. This lecture will quickly explore the concept.

Exercise: Create a triangle and put an inscribed circle inside the triangle.

1. Generate three components of "**Construct Point**". Each represents the corner point of the triangle and which could be adjusted by three set of number sliders. Their output points were lists of the x, y, z coordinates.

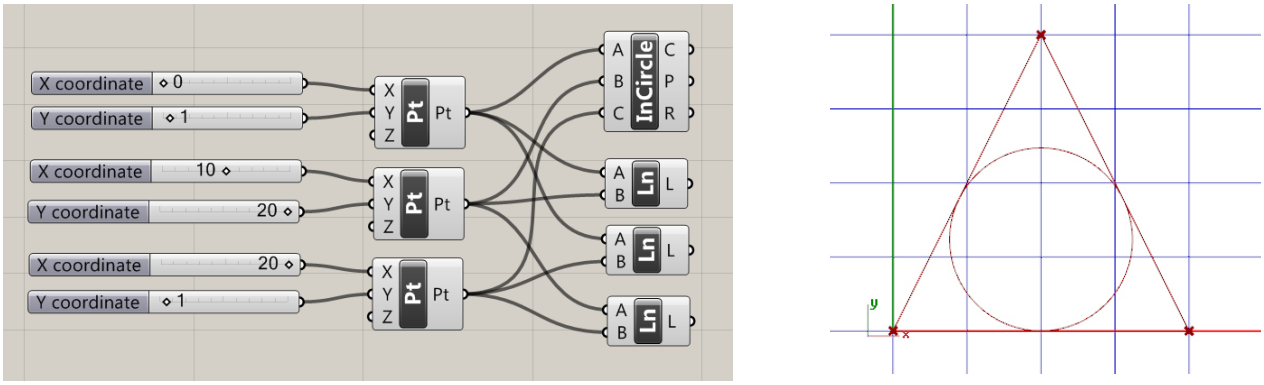


2. Generate a "**line** between two points" (type **ln**) with the input points from the combination of the three point components. At this moment, three lines are created temporary in Rhino.



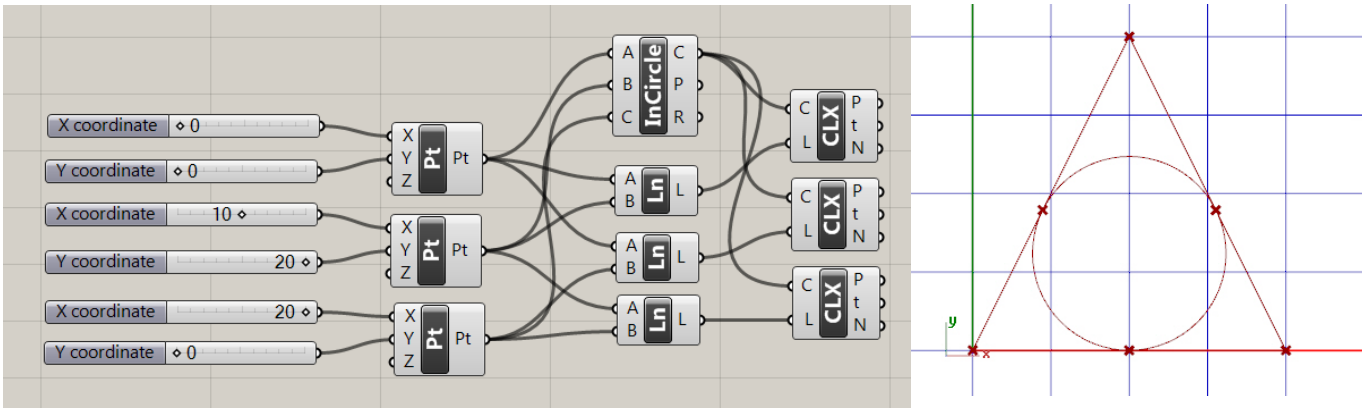
3. Use the function of "**InCircle**" which will create an incircle of a triangle, which is a circle inscribed inside a triangle.

   Note: **InCircle** will take the first, second, and third corner points of the triangle as corners A, B, C input point. As long as the component knows these three corner endpoints, it will create three outputs of a **C**ircle, a circle **P**lane, and the **R**adius of this circle. Of course, a circle is drawn inside the triangle in Rhino.

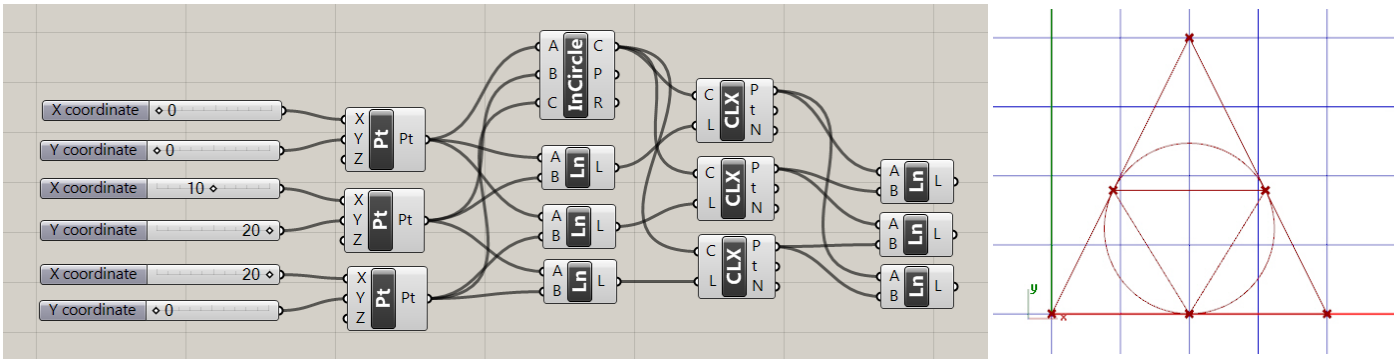


2nd Run: Put another triangle inside the circle and an inscribe circle of the triangle.

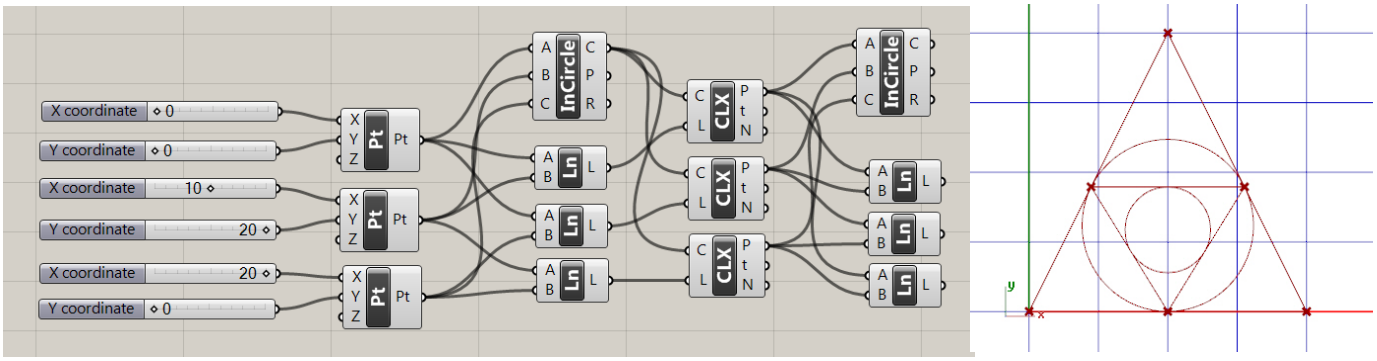
4. Use a "**curve | line**" component to solve interaction events for a curve and a line. It will take the "circle" output from the InCircle and "line" outputs from the lines as its input. It also will generate an inscribe point as intersection point, and parameter of the curve as output values. Then the three curve|line components will create three inscribe points as shown in the following pictures.



5. Use "**Line**" component to connect the three inscribe points on the triangle to generate a triangle inside the circle. See the following image.

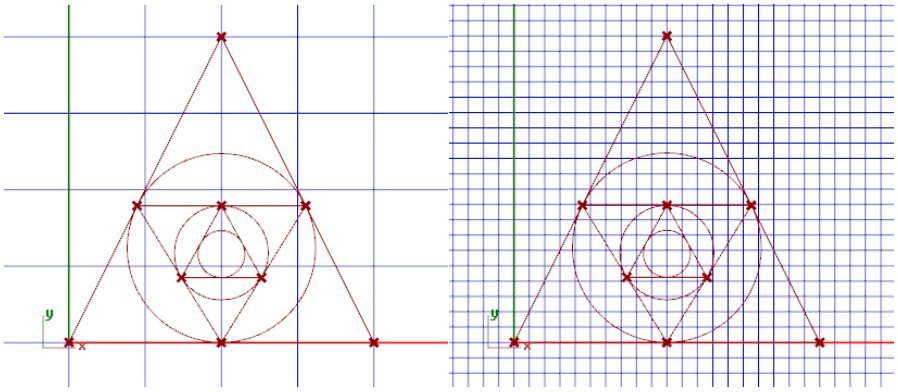


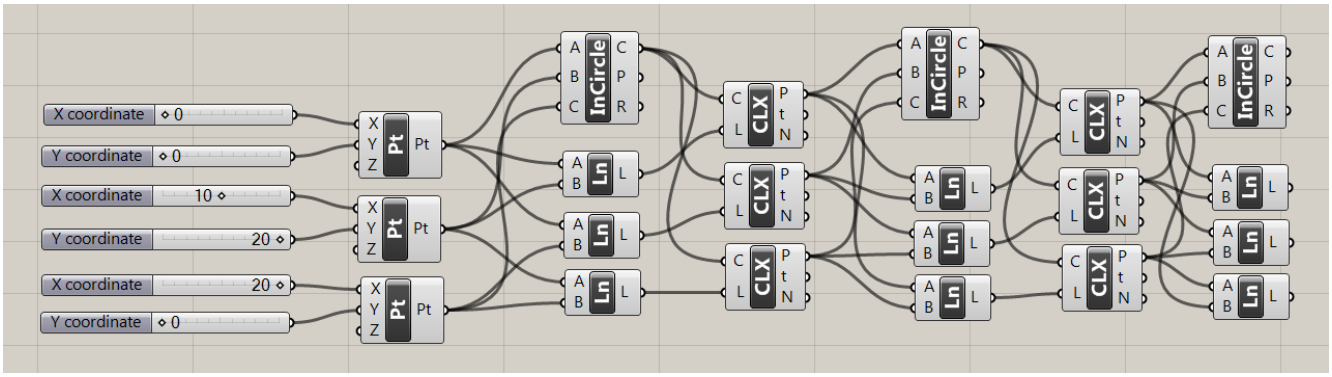
6. Use **InCircle**, again, for generating the inscribed circle inside the second triangle. This will complete the second run of execution and creation.



3rd Run:

Apply the same methods and procedures to execute the third run of generating a triangle inside an inscribed circle. In this third run of generating the third circle within the third triangle, results will have the following image composition. The generation could keep repeating again and again to create circles sitting inside of a triangle, which is the notion of recursive repetition.





Notions of recursive pattern:

Such repetitious procedures are the notion of repetition and recursion. Repetition is to apply the same steps over again. Recursion is the process of repeating items in a self-similar way. The classic example of recursion is Fibonacci sequence.

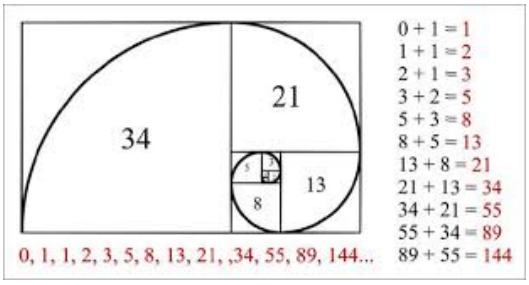
Fib(0) = 0 as base case 1,
Fib(1) = 1 as base case 2,
For all integers n>1, Fib(n) := Fib(n-1) + Fib (n-2).

$$F(n) = \begin{cases} f(n-1) + f(n-2), & for\ n > 1; \\ 1, & for\ n = 1; \\ 0, & for\ n = 0; \end{cases}$$

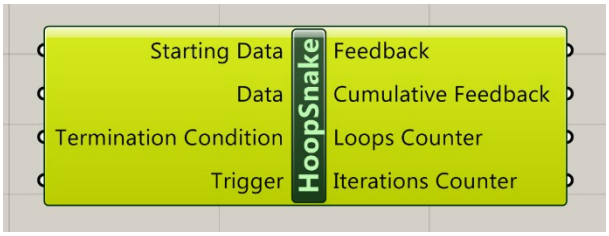
Each copy of Fibonacci can create two new copies, not just one. Thus, Fibonacci is said to be doubly recursive. In mathematics, the Fibonacci numbers or Fibonacci sequence are the numbers in the following integer sequence:
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144….

or formally written as…
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144…

Thus,
Fib (3) = Fib (1) + Fib (2)
1 + 1 = 2



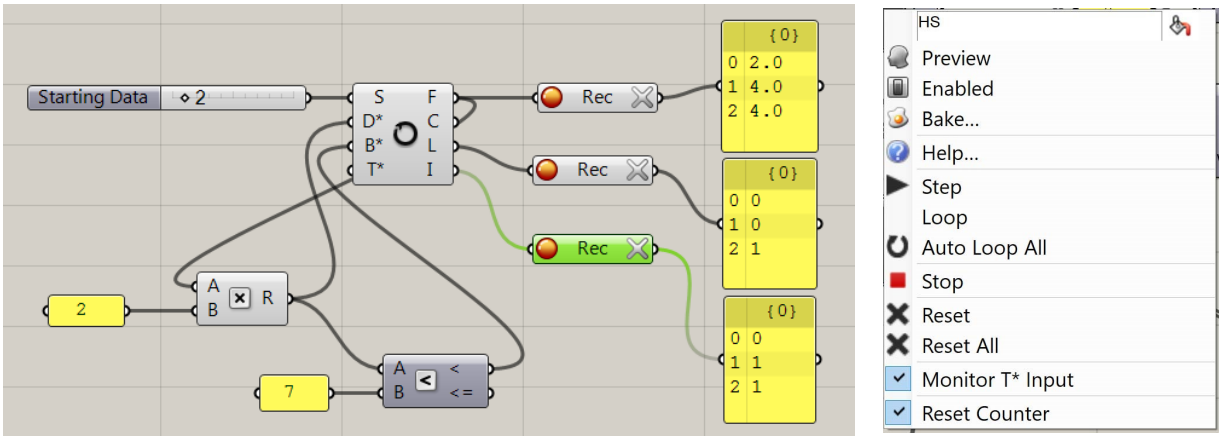
In this exercise, repetition could be implemented by recursion. A GH plug-in, HoopSnake, could be applied to do the job. HoopSnake is a component that creates a copy of the data it receives at its input upon user request and stores it locally. In theory, the component could send the output of a set of components back to its input in a looping fashion, which could open up some new possibilities for iterative process modeling. HoopSnake could be downloaded from www.food4rhino.com page. Information about HoopSnake is on: https://www.grasshopper3d.com/group/hoopsnake and http://yconst.com/software/hoopsnake/. Yet, it is an old file…



To install:
In GH, File > Special Folders > Components folder.
Save the gha file there.
Right-click the file > Properties > clean the "blocked" text

Restart Rhino and GH after the HoopSnake is copied to the Component Folder in GH.

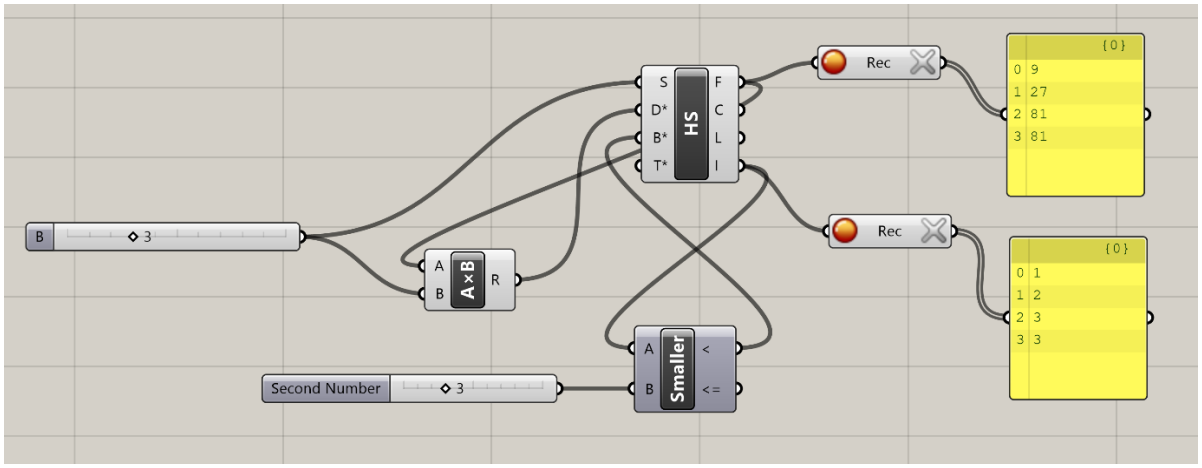
Notions of recursion in HoopSnake – A math example. Right click HP component, select Loop to start the recursion.



There is the component of "Data Recorder" applied, which is the parameter that stored the data generated each time and kept the data in Rhino as well. The data could be erased from clicking the X sign and record it again. When you click on the button, you actually turn the component off; and turn it on again to make it workable. Here are the short descriptions of a recursion example.

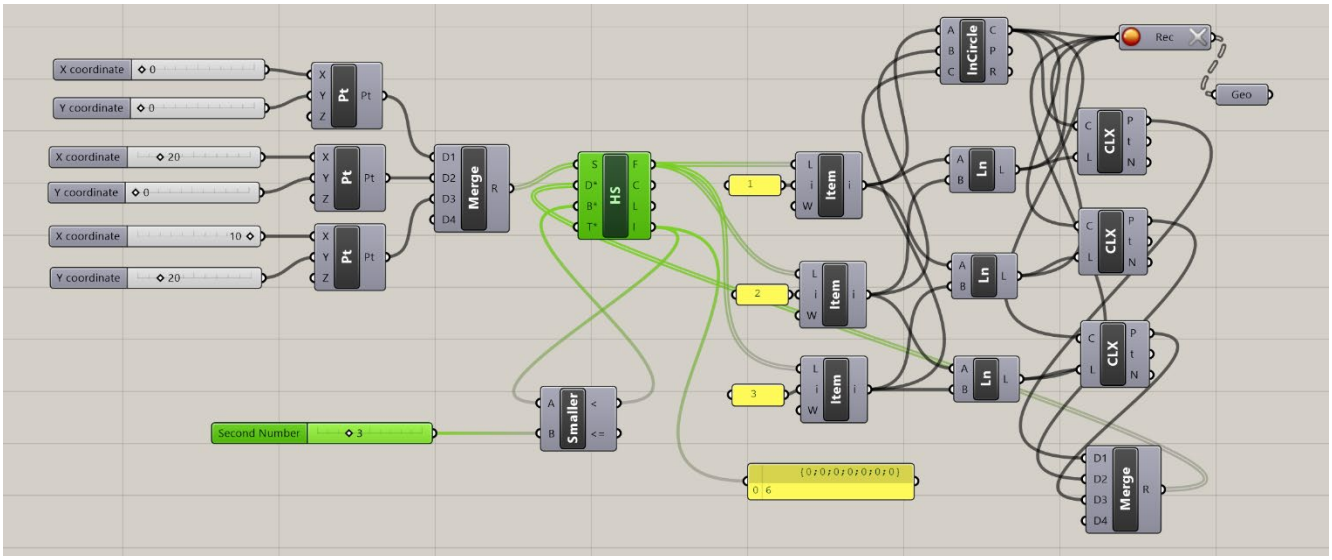
- **Initial - Index 0**: When the algorithm starts as initial run, S (Starting Data) accepts 2, HS algorithm checks the data in D (Data) for **F** output. Data in **D** is empty (null) at this point, so HS takes the original data of 2 and pass it to F (Feedback). Here, F is supposed to get a copy of the data from D for future actions, but, since D is null at this point, so **F** outputs 2 (shown in **index 0**) and calls the next iteration of **index 1**. But, the iteration has not been done, so the number of "I" is 0.
- **1st loop - Index 1:** The first iteration starts from F having the data of 2, pass it to the function of **AxB** for 4, and gives it to D, also checks if 4<7 is true for B (termination condition). If it is **true**, then 4 will be the data for F to start the next call for **index2**. At this end of **index 1**, F has the value of 4, L has 0 loop left, and **I** has 1 iteration done.
- **2nd loop - Index 2:** The second iteration starts with the value of 4 to **AxB,** 4x2=8, 8<7 is false, **B** (termination condition) stops the loop, and not pass the data to **F**. So, at the end of this second iteration (index 2), F still has the value of 4, **L** has 1 loop that has not completed, and "**I**" has one previous iteration done, so the value of iteration is still 1, iteration 2 isn't counted in **L**.

In this example, F provides input to outside components for generating outputs for D, which also provides data back to F. Thus, it is the recursive activity that repeats items in a self-similar way.
Here is another example of the second power or the third power of a number, which is terminated by the number of iteratipons. Here the root number is 3. It will run three times of multiplication. Results show on the first panel of the output from Feedback.



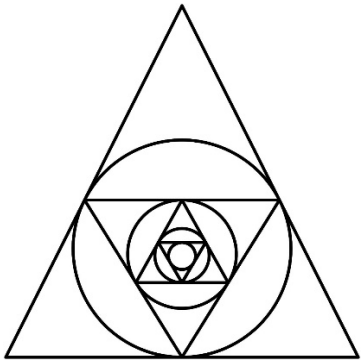
GH HoopSnake operation on the triangle and incircle

The following one is the recursion for incircles of triangles, which has the original triangle locates at (0,0), (20,0), (10,20), and loop three times to create three iterations. The final form has four triangles and four incircle.



Here, the three corner points of the first generation of the triangle will be **merged** (add D3 to increase inputs) into a big list as the input of the HoopSnake which does the major recursion work. The three lists of points are merged onto one list and pass into the HoopSnake. This list serves as the next generation of the corner points for creating the triangle shape and its incircle.

When HS starts loop function, it will take the list and separate three items into lists of point coordination, create a triangle, an incircle and the joint point between the incircle and the triangle will be merged into a big list. This long list will be the input to "D" data variable for next iteration. Before it starts the next operation, it will check the termination condition. If the desired loop is 3 and it already iterated three times, it will stop at the beginning of the fourth time. Results will have three runs (of three triangle and three circles) plus the original module with total of 4 each. Save the form through Geometry parameter and bake them to Rhino for further manipulations.



Future application – use HoopSnake to generate repeated objects or structural patterns.