



# C# in GH - Function Composability

Tokyo AEC Industry Dev Group - 06/13/2020

Ping-Hsiang Chen

## Compose - ability / -possibility

In computer science, “composability” is a system design principle that deals with the inter-relationships of elements. “Function composition” is a mechanism which allows the result of one argument passed to another one to form a more complex one. The final outcome is the accumulative result of the entire process.

This formation concept resonates with the geometric system typically employed in architectural design and modelling procedure.

In this session, the emphasis will be placed on how geometric functions in Grasshopper C# can be composed and have interchangeability to another one. We will also investigate how to create shape grammar based on the principle introduced.

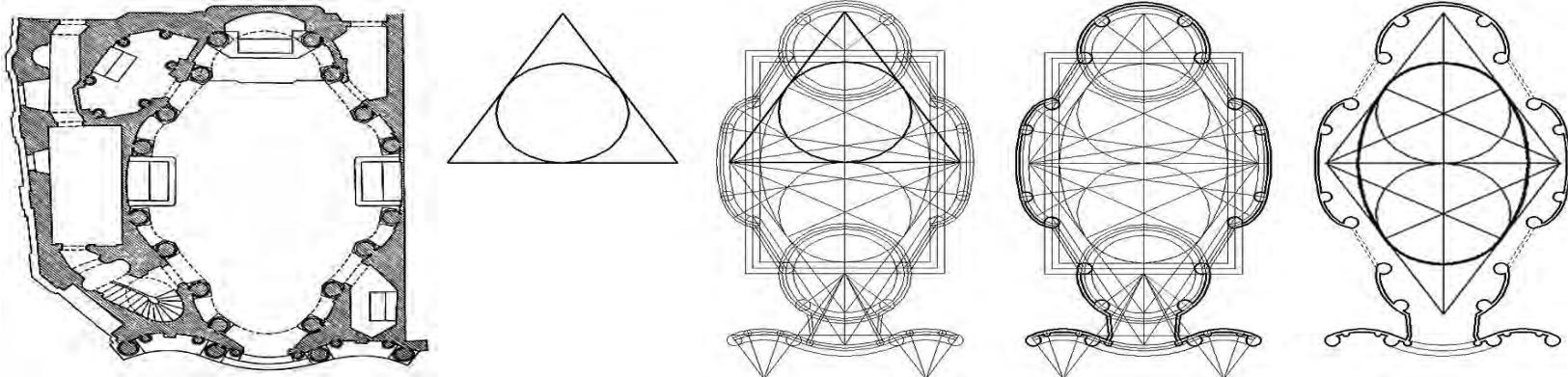


image source: Google

<https://digital.kenyon.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=1061&context=perejournal>



# Self-Organization in Biological Systems

Scott Camazine Jean-Louis Deneubourg Nigel R. Franks  
James Sneyd Guy Theraulaz Eric Bonabeau



PRINCETON STUDIES IN COMPLEXITY

## Topics

1. Function Handling in RhinoCommon
2. NodeInCode Function & Use Cases
3. Grasshopper Data Structure
4. Geometric Composition

## Related Topics

1. Emerging Behaviour
2. L-System
3. Shape Grammar

## Software

Rhino 6.0 & plug-ins

(<https://drive.google.com/open?id=16SBwSGzBacYTckU6d1tvKlzk2k6R7crT>)



# Overview

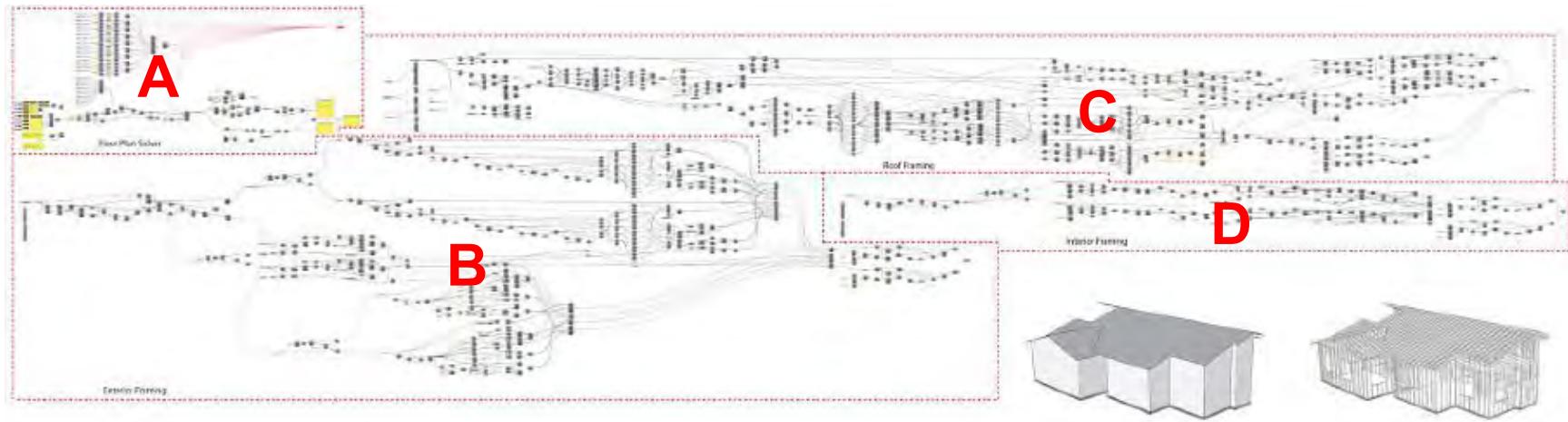


image source: Google

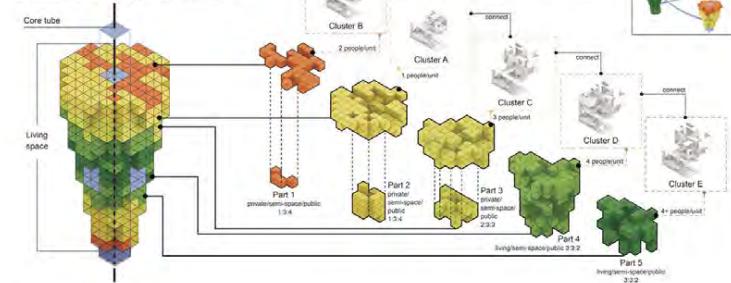
## UNDERGROUND X-Community 3

Workshop 2, Group 4

Group members  
Daoyuan CHEN, Pengcheng ZHAI, Zheng CHEN, Zheng CHEN, Kaijie YU, Yize LIU  
Tutor  
Ping-Hsiang Chen, Peng Qin

### Community Explosion Diagram

An example by Type 3  
The relationship between clusters, parts and communities



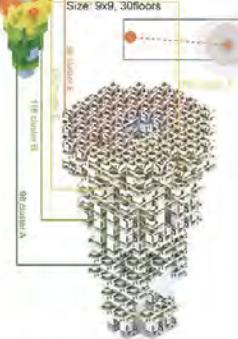
### Community Size

The distance between the community and the resource point control the size of the community. The closer the resource points are, the larger the size of the community, which means the community has more clusters and more layers.

In other types, communities that depend on the same resource point will be interconnected to form a group.

### Community Type

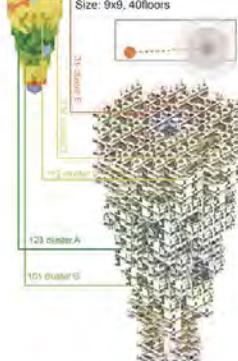
Type 1  
Size: 9x9, 30floors



Type 2  
Size: 12x12, 30floors

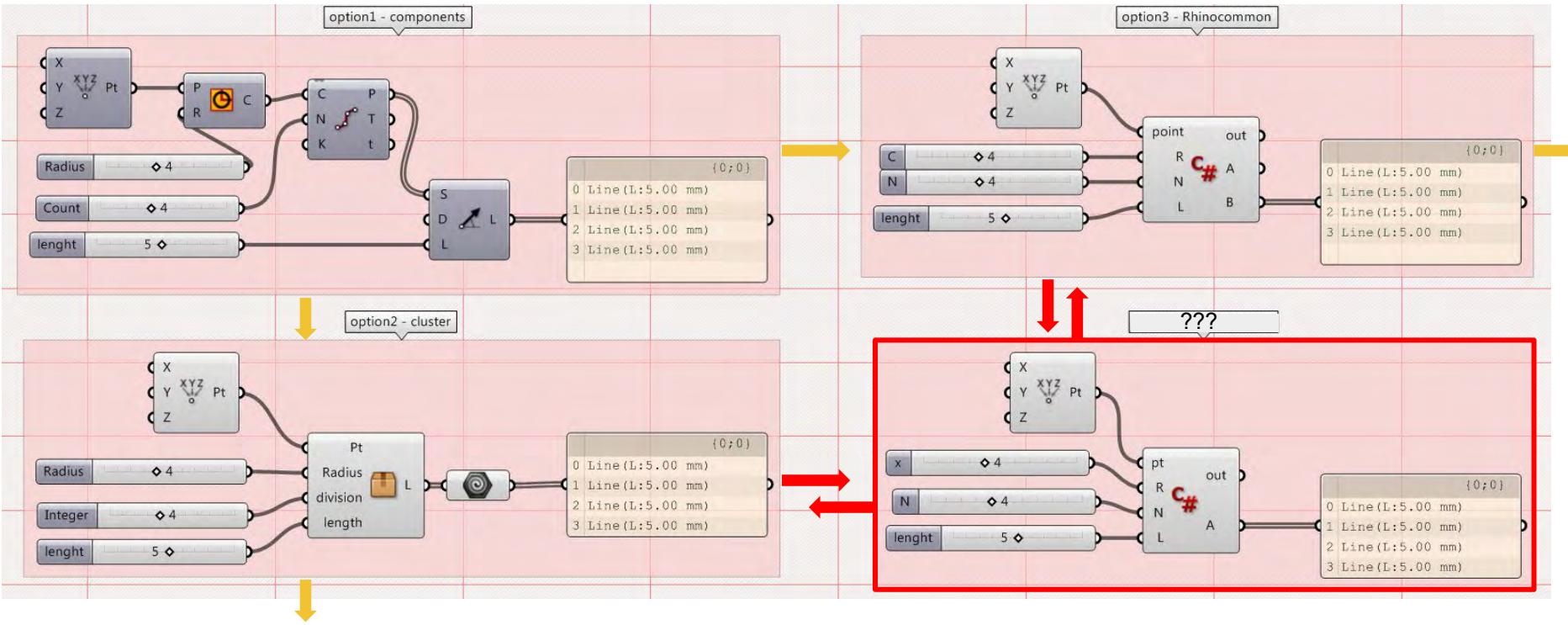


Type 3  
Size: 9x9, 40floors

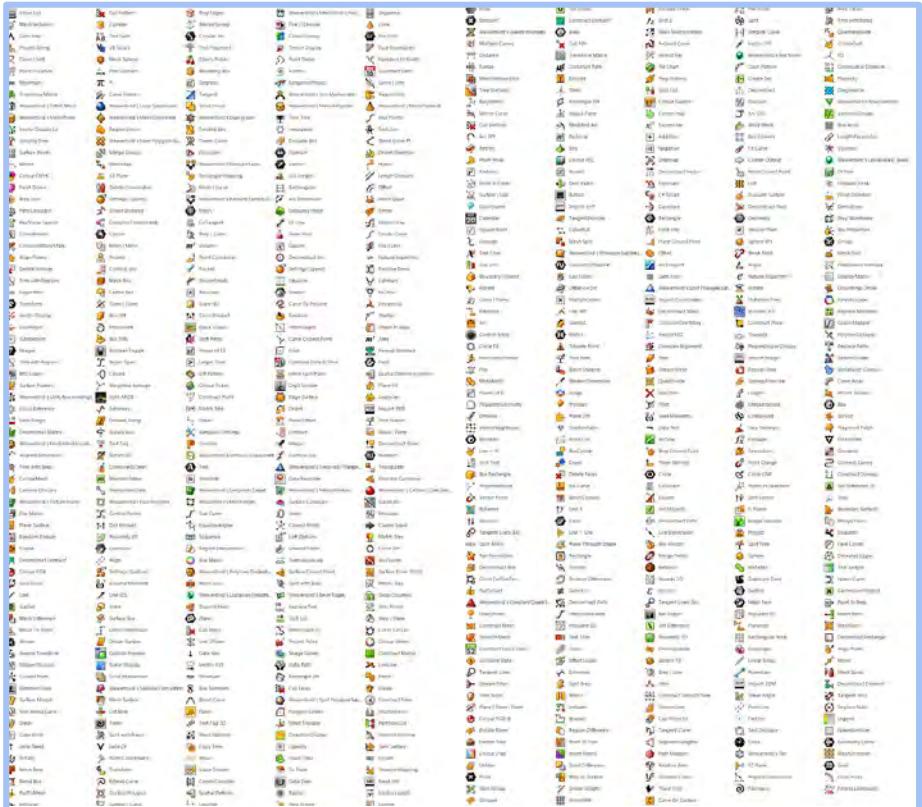


# Overview

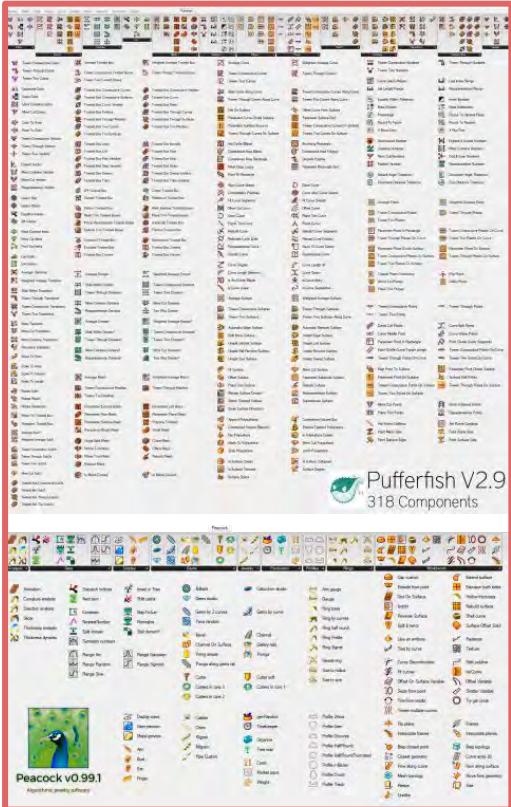
- Function composition
- Code re-usability
- Function extension
- Iterative capacity



# Overview



Grasshopper functionality

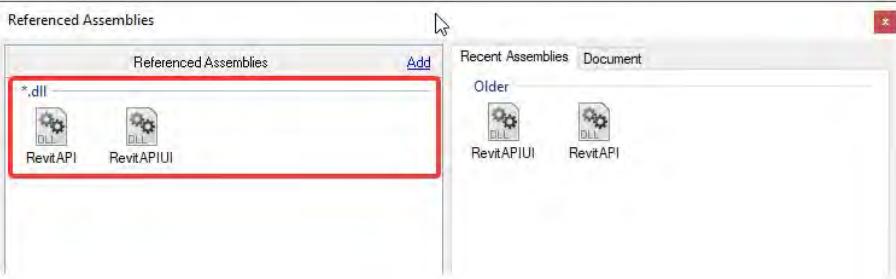


Plug-in functionality

image source: Google

# Overview - function extension

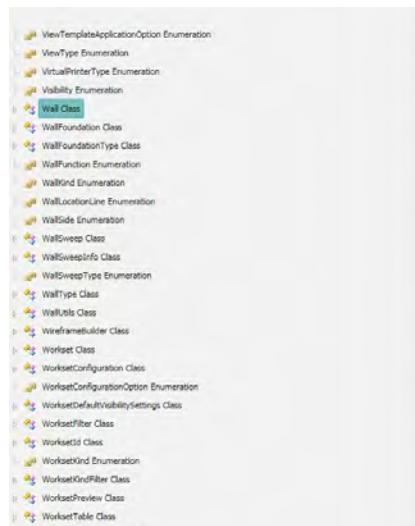
## Assembly management



## Namespace management

```
using RIR = RhinoInside.Revit;
using DB = Autodesk.Revit.DB;
using UI = Autodesk.Revit.UI;
```

## Syntax management



The wall object represents all the different kinds of walls in Revit.

### Examples

```
public void GetInfo.Wall(Wall wall)
{
    string message = "Wall";
    // Get wall AnalyticalModel type
    AnalyticalModel model = wall.GetAnalyticalModel();
    if (null != model)
    {
        // For some situations (such as architecture wall or in building version),
        // this property may return null, but if it doesn't return null, it should
        // be AnalyticalModelWall
        message += "\nWall AnalyticalModel type is " + model;
    }

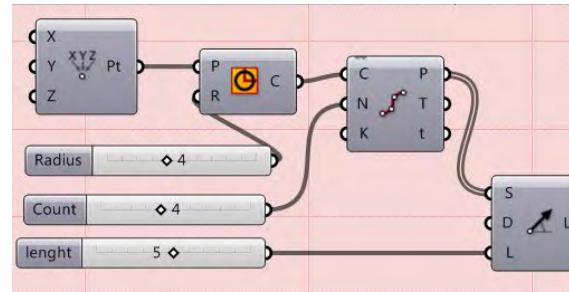
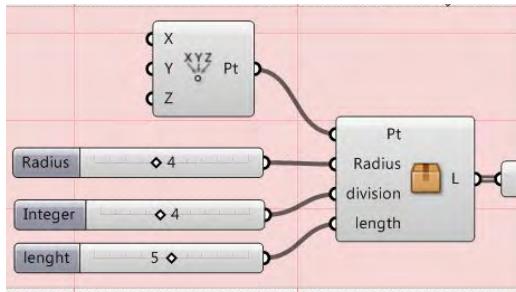
    wall.Flip();
    message += "\nIf wall Flipped " + wall.Flipped;
    // Get curve start point
    message += "\nWall orientation point is " + wall.Orientation.X + " "
              + wall.Orientation.Y + " " + wall.Orientation.Z + "/";
    // Get wall StructuralUsage
    message += "\nWall StructuralUsage is " + wall.StructuralUsage;
    // Get wall type name
    message += "\nWall type name is " + wall.WallType.Name;
    // Get wall width
    message += "\nWall width is " + wall.Width;

    TaskDialog.Show("Revit",message);
}
```

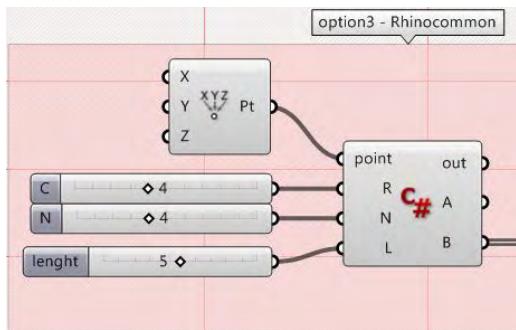


## Interoperability Issue

## Cluster vs Rhinocommon



GH geometry / data type  
principle



Base and Derived class

Curve  
ArcCurve  
CurveProxy  
LineCurve  
NurbsCurve  
PolyCurve  
PolylineCurve

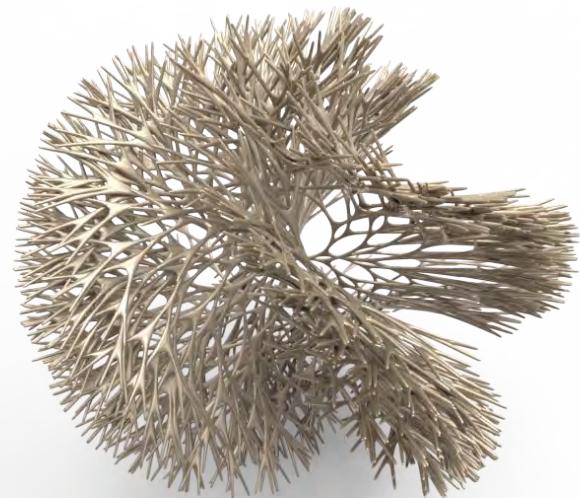
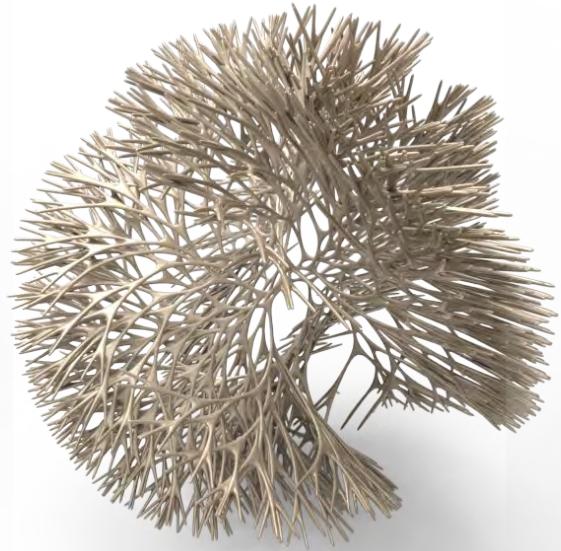
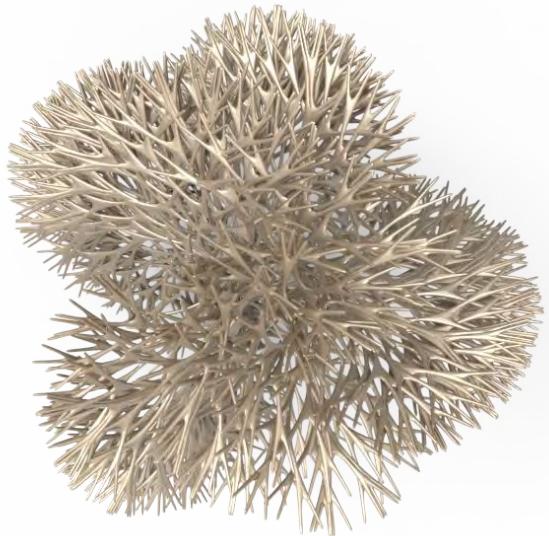
Overload methods

Point3d(Point3d)  
Point3d(Point3f)  
Point3d(Point4d)  
Point3d(Vector3d)  
Point3d(Double, Double, Double)

Ex. PointAt(t)

(Explicitly define)

## Iterative Capability



# NodeInCode

RhinoCommon API

- RhinoCommon API
- Namespaces
- Rhino.NodeInCode
  - ComponentFunctionInfo Class
  - Components Class
  - NodeInCodeTable Class

## Rhino.NodeInCode Namespace

[Missing <summary> documentation for "N:Rhino.NodeInCode"]

### ▪ Classes

Class	Description
 ComponentFunctionInfo	Defines the base class for a function representing a component. This class is abstract.
 Components	Provides access to all Grasshopper runtime components.
 NodeInCodeTable	Permits rapid access to references to all Grasshopper functions.

[https://developer.rhino3d.com/api/RhinoCommon/html/N\\_Rhino\\_NodeInCode.htm](https://developer.rhino3d.com/api/RhinoCommon/html/N_Rhino_NodeInCode.htm)

## Reference

- <https://developer.rhino3d.com/guides/rhinopython/ghpython-call-components/>
- <https://stevebaer.wordpress.com/2013/12/11/ghpython-node-in-code/>
- <https://discourse.mcneel.com/t/using-the-nodeincode-namespace/49735>
- <https://discourse.mcneel.com/t/new-version-of-ghpythonlib-components/34619>

# NodeInCode

December 11, 2013 / stevebaer

## ghPython – New component and parallel modules

Just in time for Christmas... ghPython 0.6.0.3 was released this week and it has two new features that I'm really excited about.

[Download ghPython...](#)

### A little background

David Rutten was visiting the McNeel Seattle office in November to discuss future work on Grasshopper and Rhino. When David is in town it always gives me the chance to brainstorm with him and try to solve some of the features that users ask for. Two features that we commonly hear about are "how can I do what X component does, but through RhinoCommon/code?" and "how can I improve performance on my computer with many CPUs?"

Out of these chats came the two major new features in ghPython 0.6.0.3; the ability to call components from python and an easy way to do this using multiple threads. ghPython 0.6.0.3 ships with a new package (ghpythonlib) that supports these two new features.

### Components As Functions (node-in-code)

There is a module in ghpythonlib called components which attempts to make every component available in python in the form of an easy to call function. Here's a sample to help paint the picture.

```
1 import ghpythonlib.components as ghcomp
2
3 # call Voronoi component with input points
4 curves = ghcomp.Voronoi(points)
5 # call Area component with curves from Voronoi
6 centroids = ghcomp.Area(curves).centroid
```



Do you want live notifications when people reply to your posts? [Enable Notifications](#).

Using Rhino remotely during COVID-19 • See what's in the Rhino 7 WIP

nodeincode



+ New Topic

33 results for nodeincode

Sort by Relevance ▾



#### Using the NodeInCode namespace

Rhino Developer ghpthon rhinocommon ghpythonlib grasshopper nodeincode Nov '17 - ...ib-components/34619 ghpthonlib.components wrapper library , which is easier to use in Python. As some noticed some time ago, we have been working on **NodeInCode** functionality. The goal was to make this functionality more understandable - with warnings that could give useful suggestions, stable - also when use...



#### Rhino.NodeInCode namespace?

Serengeti (Rhino 7 WIP) Developer rhino grasshopper python Nov '16 - Not sure if this is a dream or I missed some announcement. 😊 Is this what I think it is?



#### NodeInCode DivideCurve strange result

Grasshopper Developer Apr '18 - ...free strange results, am I doing something wrong? Canvas%20at%2017%3B02%3B35 private void RunScript(ref object A, ref object B, ref object C) { Rhino.NodeInCode ComponentFunctionInfo cfi = Rhino.NodeInCode Components.FindComponent("DivideCurve"); string[] warnings, object[] result = cfi.Evaluate(new object[]{});



#### Asynchronous issues with NodeInCode

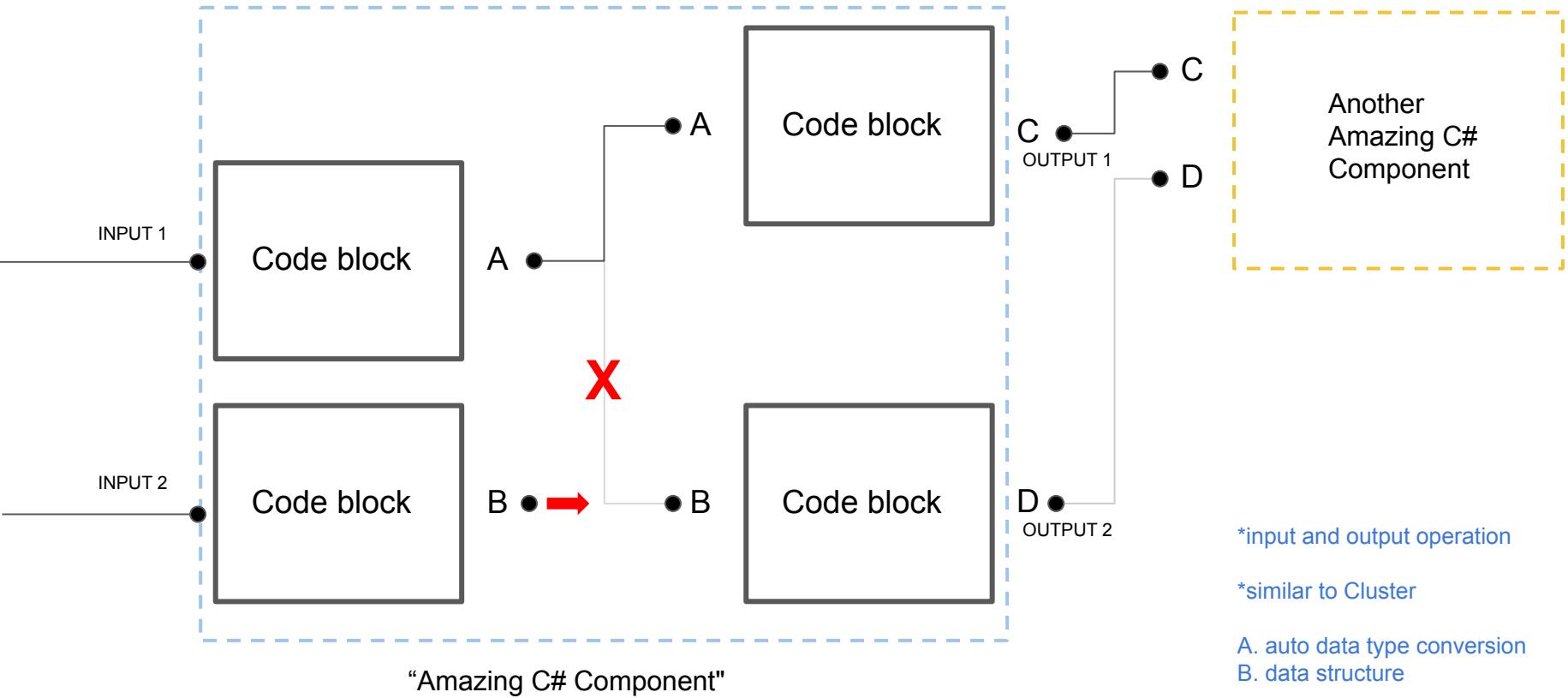
Scripting Apr '18 - Great Guillio! I have a question, what about cascading using **NodeInCode**? that is, create a definition (connected nodes) from code. Should we use asynchronous patterns? what considerations can you add for this? Thanks!



#### Rhinocommon NodeInCode doesn't work without starting grasshopper

Scripting Dec '18 - Hey @piac i fiddled around with **nodeInCode** today and i cant recreate this: <https://discourse.mcneel.com/using-the-nodeincode-namespace/49735/8> Using the **NodeInCode** namespace Do you need gras...

## What is NodeInCode?



# Create a custom function using Rhinocommon default syntax

## Curve.PointAt Method

Evaluates point at a curve parameter.

**Namespace:** Rhino.Geometry  
**Assembly:** RhinoCommon (in RhinoCommon.dll)

### Syntax

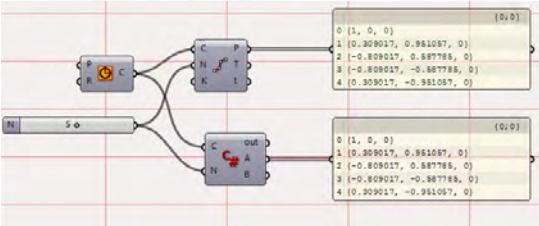
C#

```
public Point3d PointAt(  
    double t  
)
```

### Parameters

t

Type: System.Double  
Evaluation parameter.



## Curve.DivideByCount Method (Int32, Boolean, Point3d[])

Divide the curve into a number of equal-length segments.

**Namespace:** Rhino.Geometry  
**Assembly:** RhinoCommon (in RhinoCommon.dll)

### Syntax

C#

```
public double[] DivideByCount(  
    int segmentCount,  
    bool includeEnds,  
    out Point3d[] points  
)
```

### Parameters

segmentCount

Type: System.Int32  
Segment count. Note that the number of division points may differ from the segment count.

includeEnds

Type: System.Boolean  
If true, then the point at the start of the first division segment is returned.

points

Type: Rhino.Geometry.Point3d[]  
A list of division points. If the function returns successfully, this point-array will be filled in.

### Return Value

Type: Double[]

Array containing division curve parameters on success, null on failure.

# Create a custom function using NodeInCode

## Method 1

### Components.FindComponent Method

Finds a component given its full name.

**Namespace:** Rhino.NodeInCode

**Assembly:** RhinoCommon (in RhinoCommon.dll)

#### Syntax

C#

```
public static ComponentFunctionInfo FindComponent(  
    string fullName  
)
```

VB

#### Parameters

*fullName*

Type: System.String

The name, including its library name and a period if it is made by a third-party.

#### Return Value

Type: ComponentFunctionInfo

[Missing <returns> documentation for "M:Rhino.NodeInCode.Components.FindComponent(System.String)"]

### ComponentFunctionInfo.Evaluate Method

Evaluates the component with a set of arguments. There needs to be an argument for each input param, and each output param gives an entry in the output array.

**Namespace:** Rhino.NodeInCode

**Assembly:** RhinoCommon (in RhinoCommon.dll)

#### Syntax

C#

```
public abstract Object[] Evaluate(  
    IEnumerable args,  
    bool keepTree,  
    out string[] warnings  
)
```

VB

#### Parameters

*args*

Type: System.Collections.IEnumerable

The arguments list. Each item is assigned to each input param, in order.

*keepTree*

Type: System.Boolean

A value indicating whether trees should be considered valid inputs, and should be returned. In this case, output variables are not simplified to common types.

*warnings*

Type: System.String[]

A possible list of warnings, or null.

#### Return Value

Type: Object[]

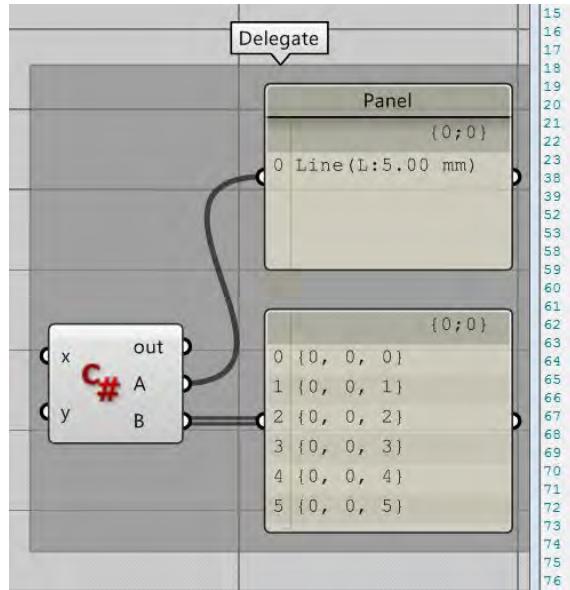
An array of objects, each representing an output result.

1

2

# Create a custom function using NodeInCode

## Method 2



```
15  using Rhino.NodeInCode;
16
17
18  /// <summary>
19  /// This class will be instantiated on demand by the Script component.
20  /// </summary>
21  public class Script_Instance : GH_ScriptInstance
22  {
23      Utility Functions
24
25      Members
26
27      private void RunScript(object x, object y, ref object A, ref object B)
28      {
29
30          Func<object,object,object,object> com1 = (Func<object,object,object,object>) Components.FindComponent("LineSDL").Delegate;
31          object ot1 = com1(Point3d.Origin, Vector3d.ZAxis, 5);
32
33          //A = com1.DynamicInvoke(Point3d.Origin, Vector3d.ZAxis, 5);
34
35          A = ot1;
36
37
38          Func<object,object,object,object> com2 = (Func<object,object,object,object>) Components.FindComponent("DivideCurve").Delegate;
39          //IList<object> ot2 = com2(ot1, 5, false) as IList<object>;
40
41          IList<object> ot2 = com2.DynamicInvoke(ot1, 5, false) as IList<object>;
42
43          B = ot2[0];
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
```

ComponentFunctionInfo.Delegate Property

Returns a delegate that can be directly invoked using a list of arguments. This flattens trees.

**Namespace:** Rhino.NodeInCode  
**Assembly:** RhinoCommon (in RhinoCommon.dll)

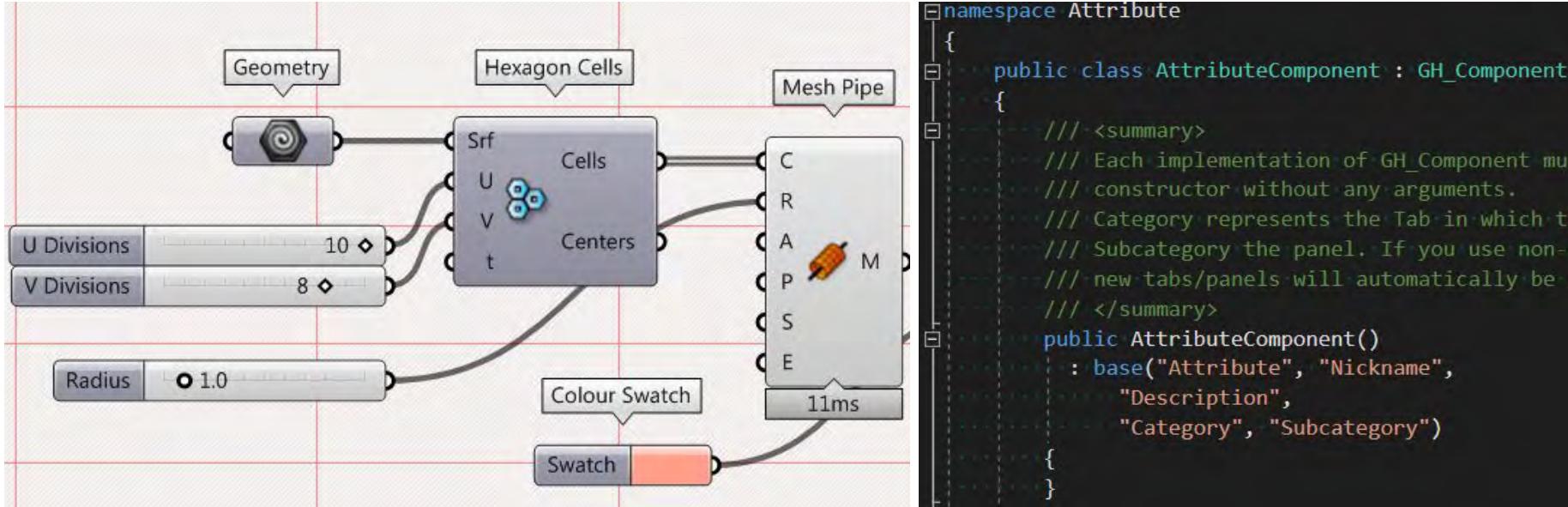
### Syntax

C#

VB

```
public Delegate Delegate { get; }
```

## Step1. Name Inspection



# NodeInCodeTable.GetDynamicMemberNames Method

Returns all additional names in the table.

**Namespace:** [Rhino.NodeInCode](#)

**Assembly:** RhinoCommon (in RhinoCommon.dll)

## ◀ Syntax

C#

VB

```
public override IEnumerable<string> GetDynamicMemberNames()
```

### Return Value

Type: [IEnumerable<String>](#)

[Missing <returns> documentation for "M:Rhino.NodeInCode.NodeInCodeTable.GetDynamicMemberNames"]

## Search for available component names

The screenshot shows the Grasshopper interface with a search results panel on the left and a code editor panel on the right.

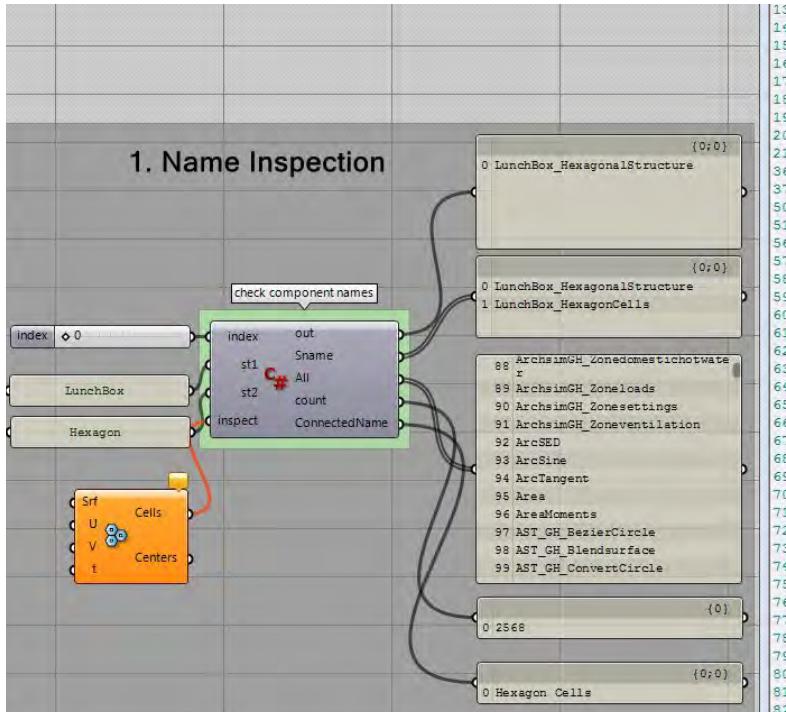
**Left Panel (Search Results):**

- A search bar at the top contains the text "C# All".
- The results list includes:
  - 2370 Starling\_slSphere
  - 2371 Starling\_slStarMesh
  - 2372 Starling\_slTopoDisk
  - 2373 Starling\_slTopoSphere
  - 2374 Starling\_slTutte (highlighted in red)
  - 2375 StreamFilter
  - 2376 StreamGate
  - 2377 Stretch
  - 2378 SubCurve
  - 2379 SubList
  - 2380 SubSet
  - 2381 Substrate
  - 2382 Subtraction
  - 2383 SumSurface
  - 2384 Surface|Curve
  - 2385 Surface|Line
  - 2386 SurfaceBox

**Right Panel (Code Editor):**

```
C# Script component: C#
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4
5  using Rhino;
6  using Rhino.Geometry;
7
8  using Grasshopper;
9  using Grasshopper.Kernel;
10 using Grasshopper.Kernel.Data;
11 using Grasshopper.Kernel.Types;
12
13 using Rhino.NodeInCode;
14
15 /// <summary>
16 /// This class will be instantiated on demand by the Script component.
17 /// </summary>
18 public class Script_Instance : GH_ScriptInstance
19 {
20     public Utility_Functions
21     {
22         public Members
23         {
24             public void RunScript(ref object All)
25             {
26                 var names = Components.NodeInCodeFunctions.GetDynamicMemberNames();
27
28                 All = names;
29             }
30         }
31     }
32
33     // <Custom additional code>
34
35     // </Custom additional code>
36 }
```

# Use name search / filter and reflection



```
13  using System.Linq;
14  using Rhino.NodeInCode;
15
16  /// <summary>
17  /// This class will be instantiated on demand by the Script component.
18  /// </summary>
19  public class Script_Instance : GH_ScriptInstance
20  {
21      Utilities_Functions
22
23      Members
24
25      /**
26      * RunScript(int index, string st1, string st2, object inspect, ref object Sname, ref object All, ref
27      * 
28      var names = Rhino.NodeInCode.Components.NodeInCodeFunctions.GetDynamicMemberNames();
29
30      var iname = names.Where(n => n.ToLower().Contains(st1.ToLower()) && n.ToLower().Contains(st2.ToLower()));
31
32      List<string> connected = new List<string>();
33
34      Component.Params.Input[3].Sources.ToList().ForEach(n =>
35      {
36          connected.Add(n.Attributes.GetTopLevel.DocObject.Name);
37      });
38
39
40      if(iname.Count() > index){
41          Print(iname.ElementAt(index));
42      }else{
43          Print("set smaller index");
44      }
45
46      Sname = iname;
47      All = names;
48      count = names.Count();
49      ConnectedName = connected;
50
51  }
```

# ComponentFunctionInfo.Evaluate Method

Evaluates the component with a set of arguments. There needs to be an argument for each input param, and each output param gives an entry in the output array.

**Namespace:** Rhino.NodeInCode

**Assembly:** RhinoCommon (in RhinoCommon.dll)

## ◀ Syntax

C#

VB

```
public abstract Object[] Evaluate(  
    IEnumerable args,  
    bool keepTree,  
    out string[] warnings  
)
```

### Parameters

*args*

Type: System.Collections.IEnumerable

The arguments list. Each item is assigned to each input param, in order.

*keepTree*

Type: System.Boolean

A value indicating whether trees should be considered valid inputs, and should be returned. In this case, output variables are not simplified to common types.

*warnings*

Type: System.String[]

A possible list of warnings, or null.

### Return Value

Type: Object[]

An array of objects, each representing an output result.

## Step2. NodeInCode syntax

The screenshot shows the Grasshopper interface with a C# NodeInCode component setup. On the left, the Script Editor window displays the C# script code. On the right, the main canvas shows the node connections.

**Script Editor Content:**

```
C# Script component: C# Script
1  using System;
2
3  /// <summary>
4  /// This class will be instantiated on demand by the Script component.
5  /// </summary>
6  public class Script_Instance : GH_ScriptInstance
7  {
8      public Utility functions;
9
10     public Members
11     {
12         public void RunScript(Curve C, int N, ref object A, ref object output_1, ref object output_2, ref object output_3)
13         {
14             Rhino.NodeInCode.ComponentFunctionInfo d = Rhino.NodeInCode.Components.FindComponent("DivideCurve");
15
16             if(d == null) return;
17
18             string[] w;
19             object[] div = d.Evaluate(new object[]{C, N}, false, out w);
20
21             A = div;
22             output_1 = div[0];
23             output_2 = div[1];
24             output_3 = div[2];
25
26         }
27
28         // <Custom additional code>
29
30         // </Custom additional code>
31     }
32 }
```

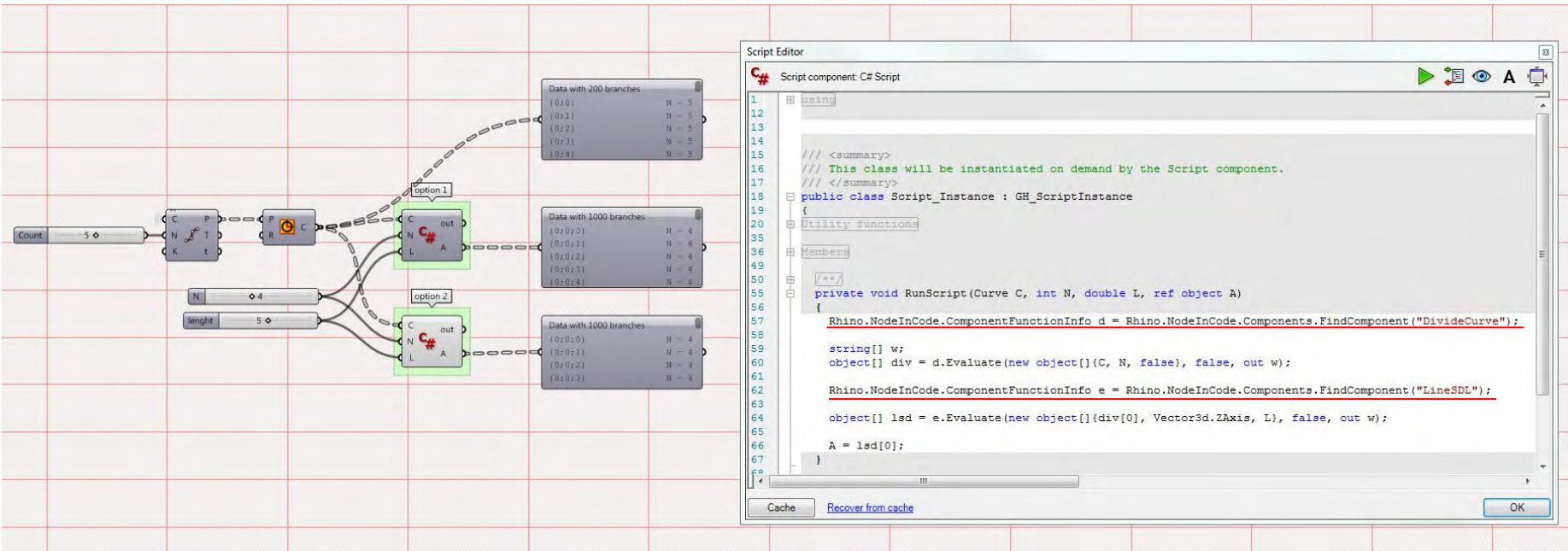
**Grasshopper Canvas (Connections):**

- A **Point R** node is connected to the **P** input of a **Component** node.
- The **N** input of the **Component** node is connected to a **Number** node set to 5.
- The **Component** node has three outputs:
  - out** connects to a **Curve C** node.
  - C# output\_1** connects to a **List** component.
  - C# output\_2** connects to another **List** component.
  - C# output\_3** connects to a third **List** component.
- The **Curve C** node outputs to a **Component** node, which then outputs to the **out** port of the main **Component** node.
- The three **List** components output to **Curve C** nodes, which then output to the **C# output\_1**, **C# output\_2**, and **C# output\_3** ports of the main **Component** node.

**Component Data Outputs (List Components):**

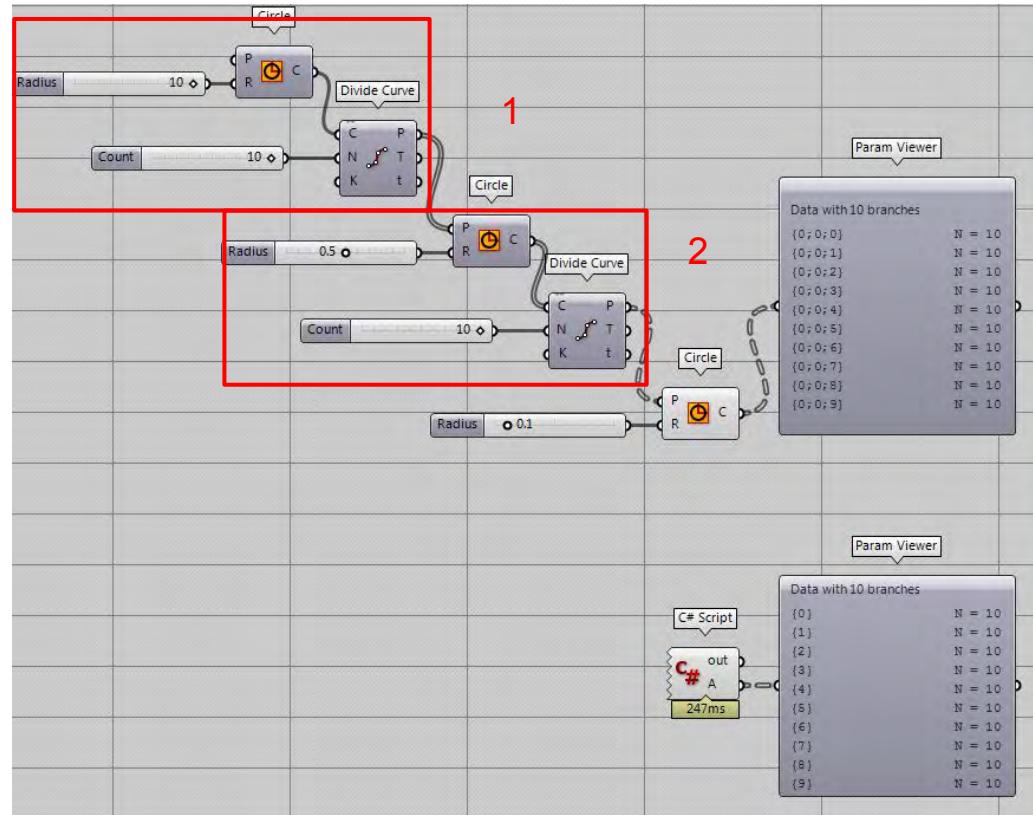
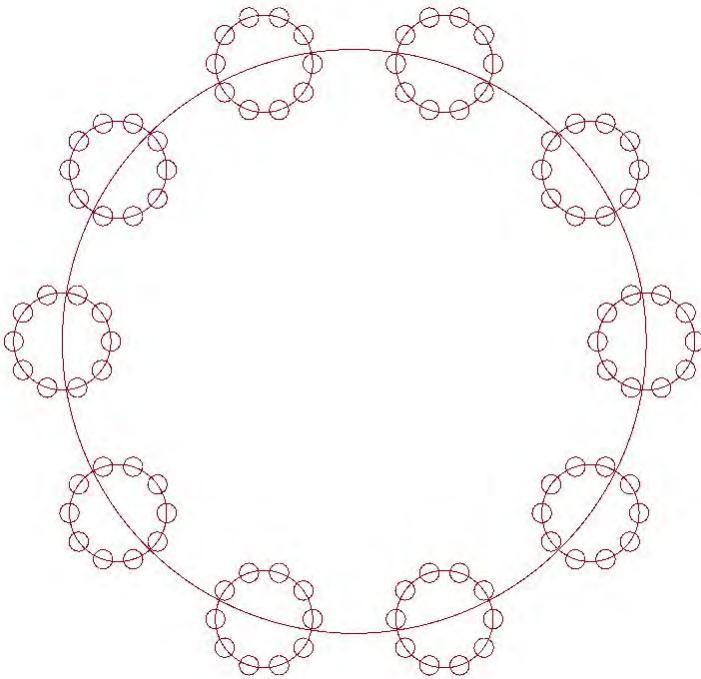
- C# output\_1**:
  - 0 {1, 0, 0}
  - 1 {0.309017, 0.951057, 0}
  - 2 {-0.809017, 0.587785, 0}
  - 3 {-0.809017, -0.587785, 0}
  - 4 {0.309017, -0.951057, 0}
- C# output\_2**:
  - 0 {1, 0, 0}
  - 1 {-0.951057, 0.309017, 0}
  - 2 {-0.587785, -0.809017, 0}
  - 3 {0.587785, -0.809017, 0}
  - 4 {0.951057, 0.309017, 0}
- C# output\_3**:
  - 0 0
  - 1 1.256637
  - 2 2.513274
  - 3 3.769911
  - 4 5.026548

## NodeInCode - multiple components

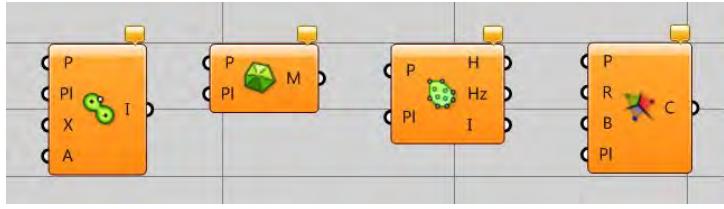


You can chain up a series of components together without considering much about data structure

## NodeInCode - multiple components



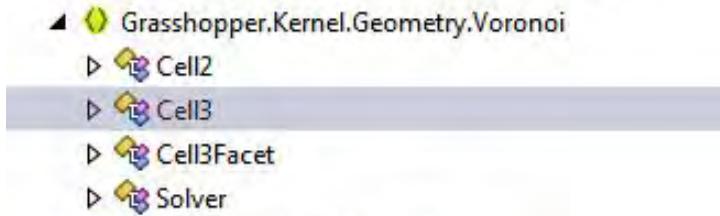
# NodeInCode - components without documentation



## Grasshopper API - Search

Voronoi  Sort by title

1. GH\_Canvas.DisplayVoronoiWarning Property
2. GH\_Canvas Properties
3. GH\_Canvas Class

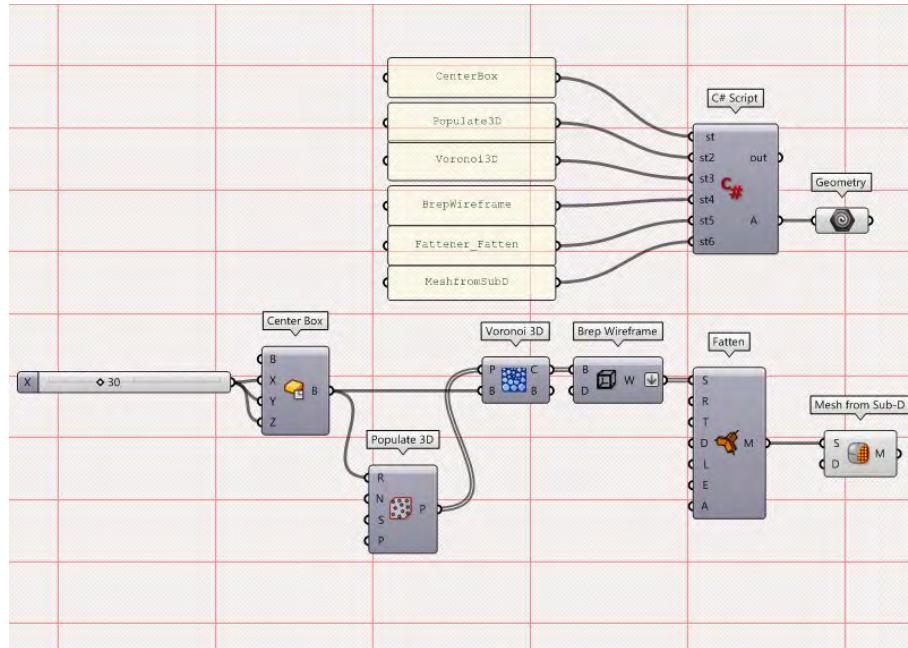
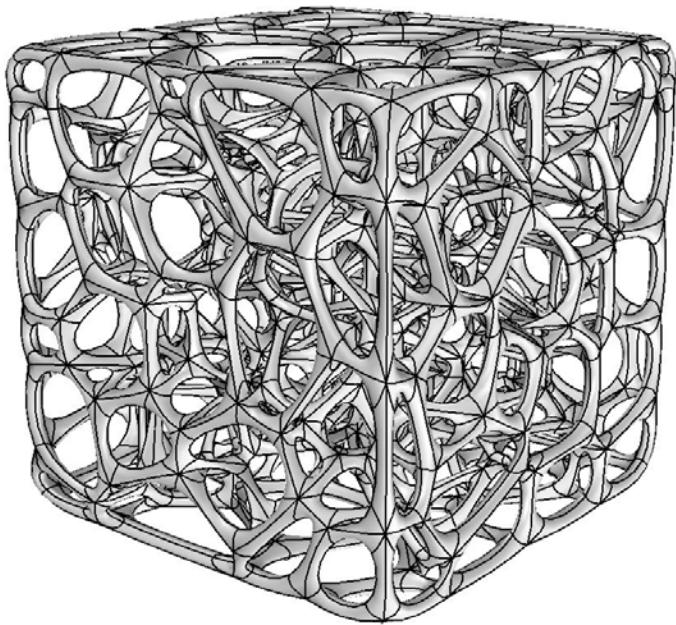


```
using Rhino;
using Rhino.Collections;
using Rhino.Geometry;
using Rhino.Runtime;
using System;
using System.Collections.Generic;
using System.Linq;

namespace Grasshopper.Kernel.Geometry.Voronoi
{
    public class Cell3
    {
        private List<Cell3Facet> m_facets;
        private Point3d m_center;
        private double m_tolerance;

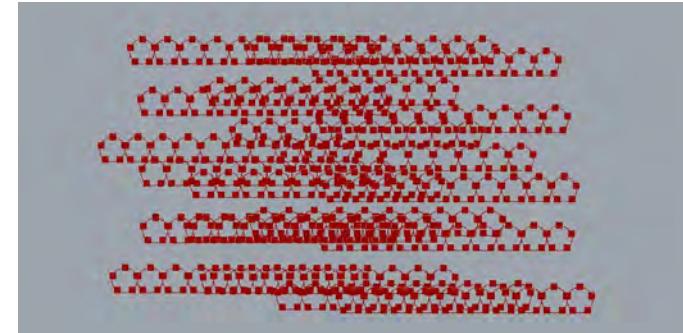
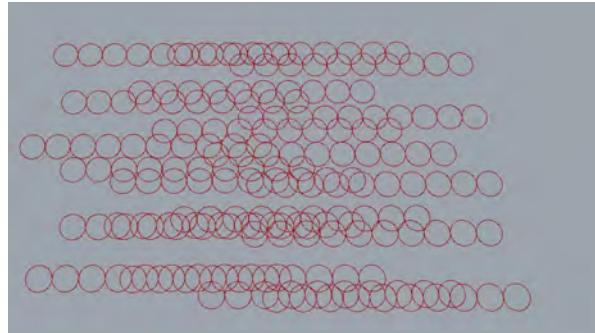
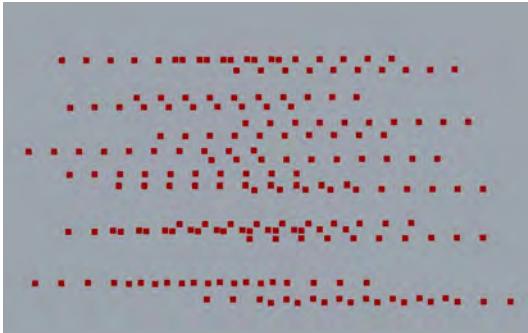
        public Cell3(Point3d center, Box box)
        {
            this.m_facets = new List<Cell3Facet>();
            if (((Box) ref box).get_Volume() == 0.0)
                throw new InvalidOperationException("Cell box must have a positive volume");
            this.m_center = center;
            Point3d[] corners = ((Box) ref box).GetCorners();
            this.m_facets.Add(new Cell3Facet((IEnumerable<Point3d>) new Point3d[5]
            {
                corners[3],
                corners[2],
                corners[1],
                corners[0],
                corners[3]
            }));
            this.m_facets.Add(new Cell3Facet((IEnumerable<Point3d>) new Point3d[5]
            {
                corners[0],
                corners[1],
                corners[5],
                corners[4],
                corners[0]
            }));
            this.m_facets.Add(new Cell3Facet((IEnumerable<Point3d>) new Point3d[5]
            {
```

Example - documentation is not available /  
3rd party function (Rhino7 WIP - using SubD component)

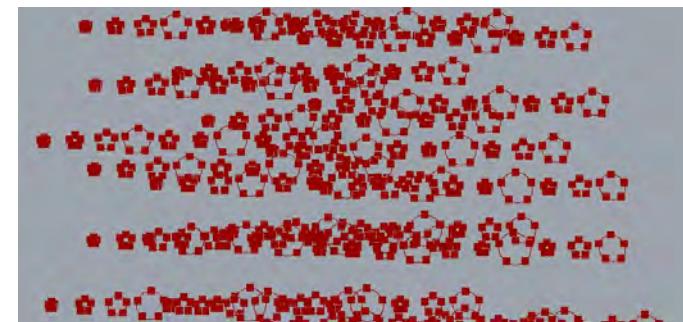
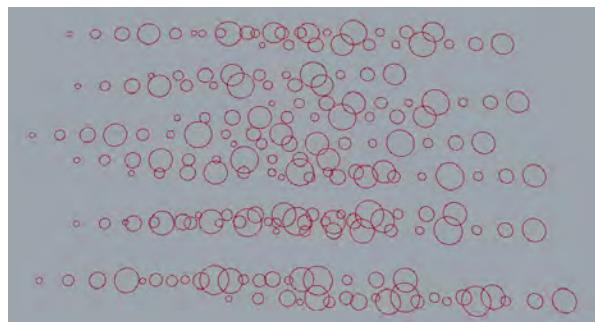
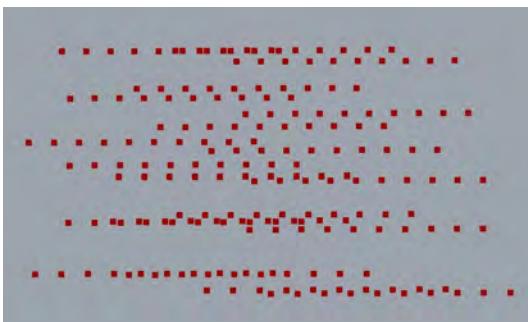


## NodeInCode - Data Flow Control

One to one / one to multiple (multiple to one)



multiple to multiple



# Data Structure in GH

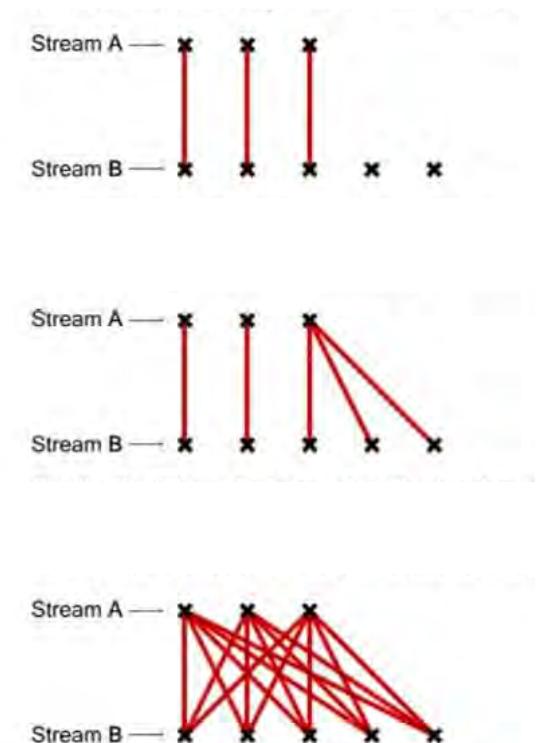
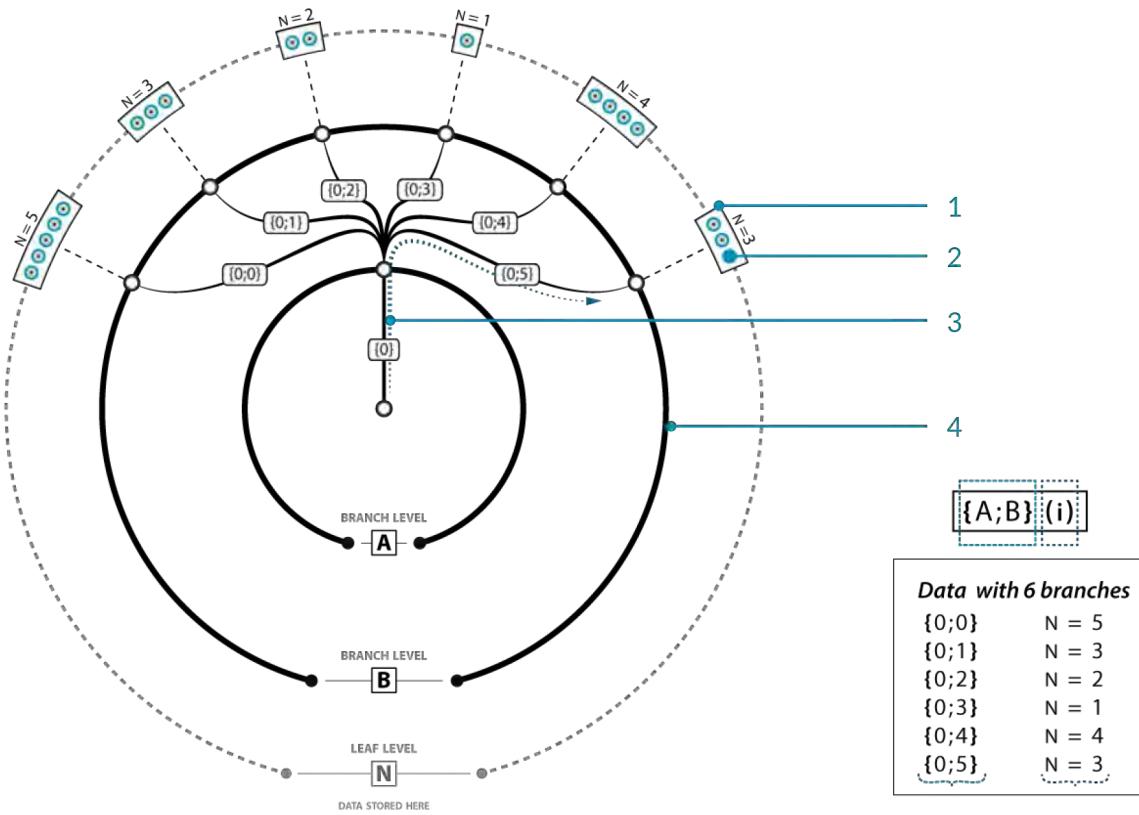
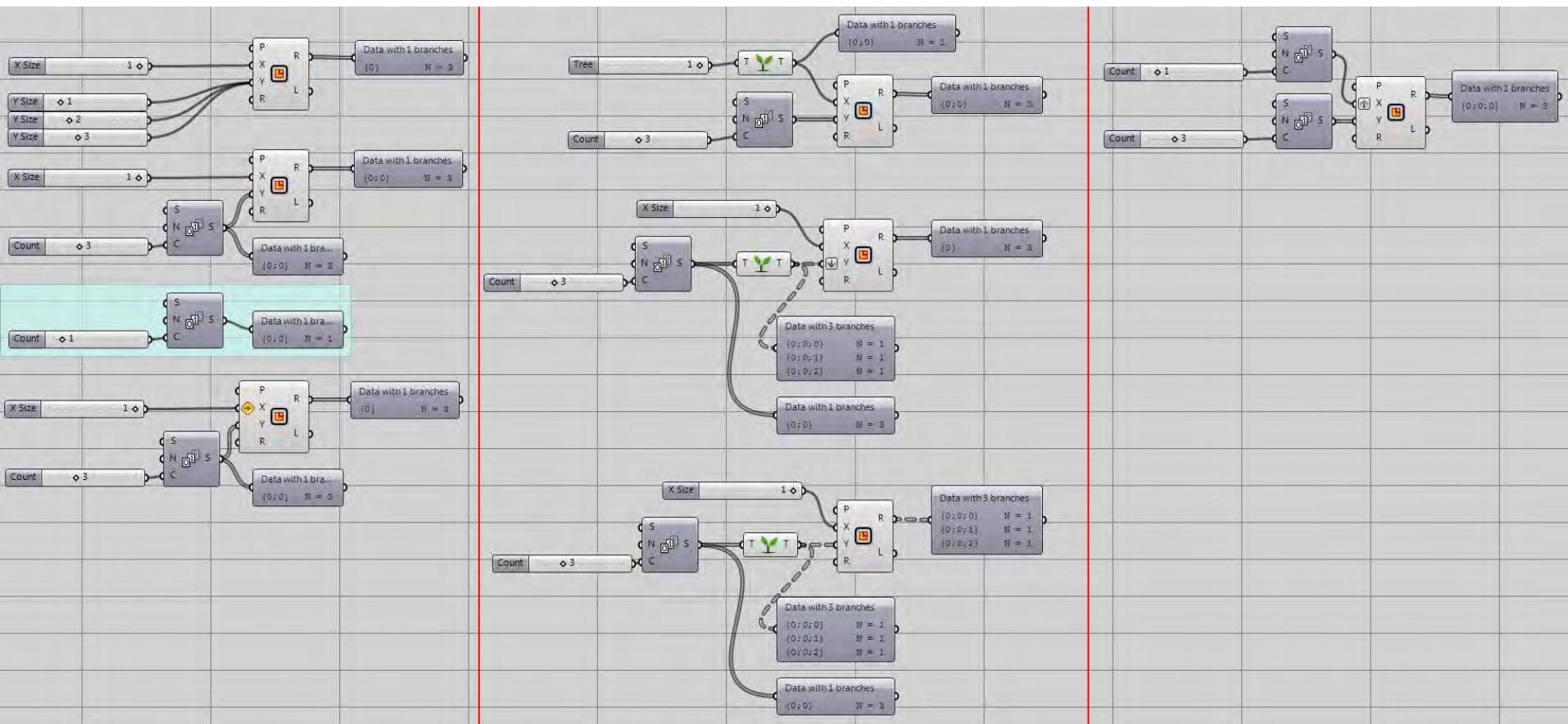
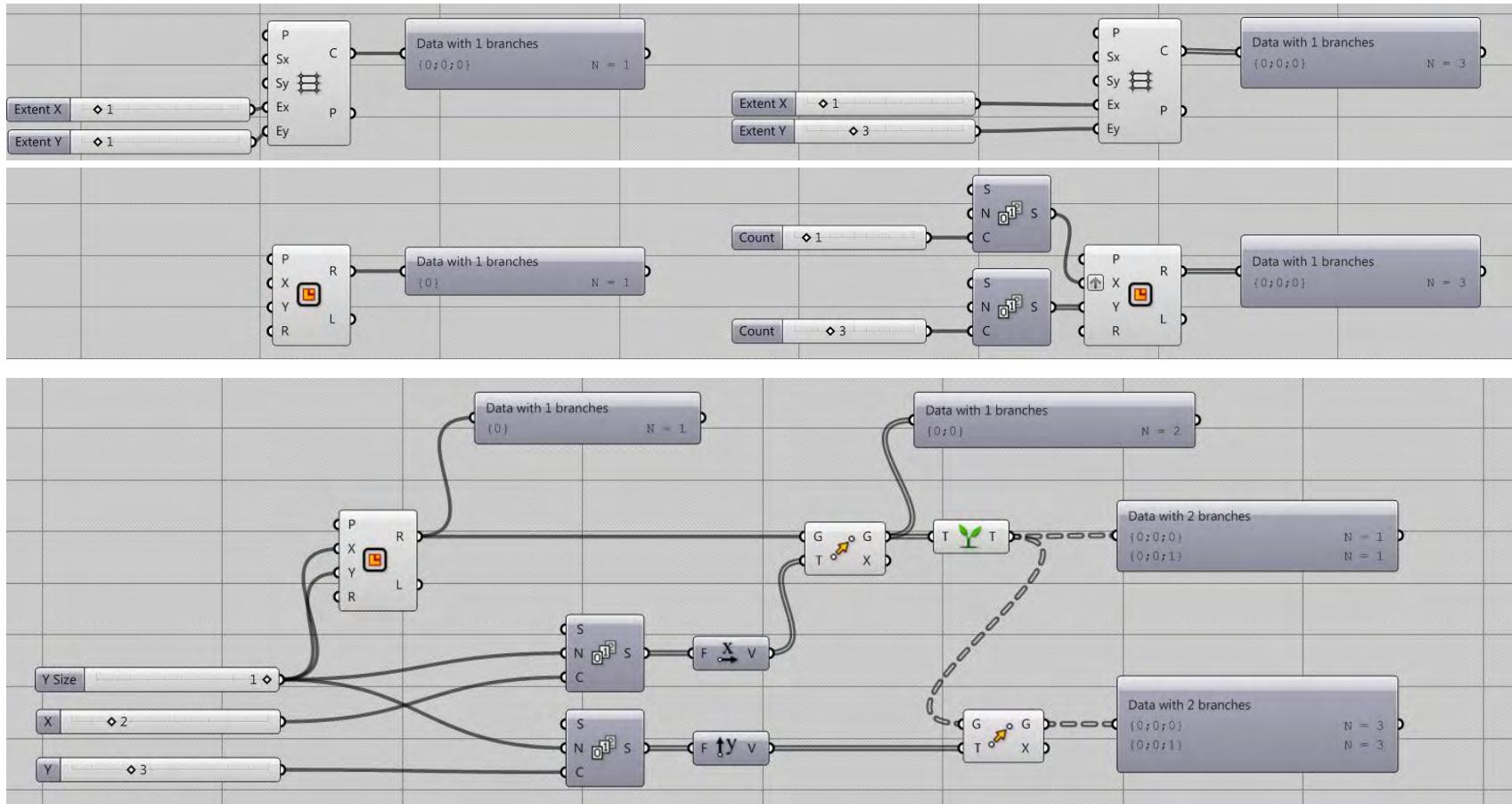
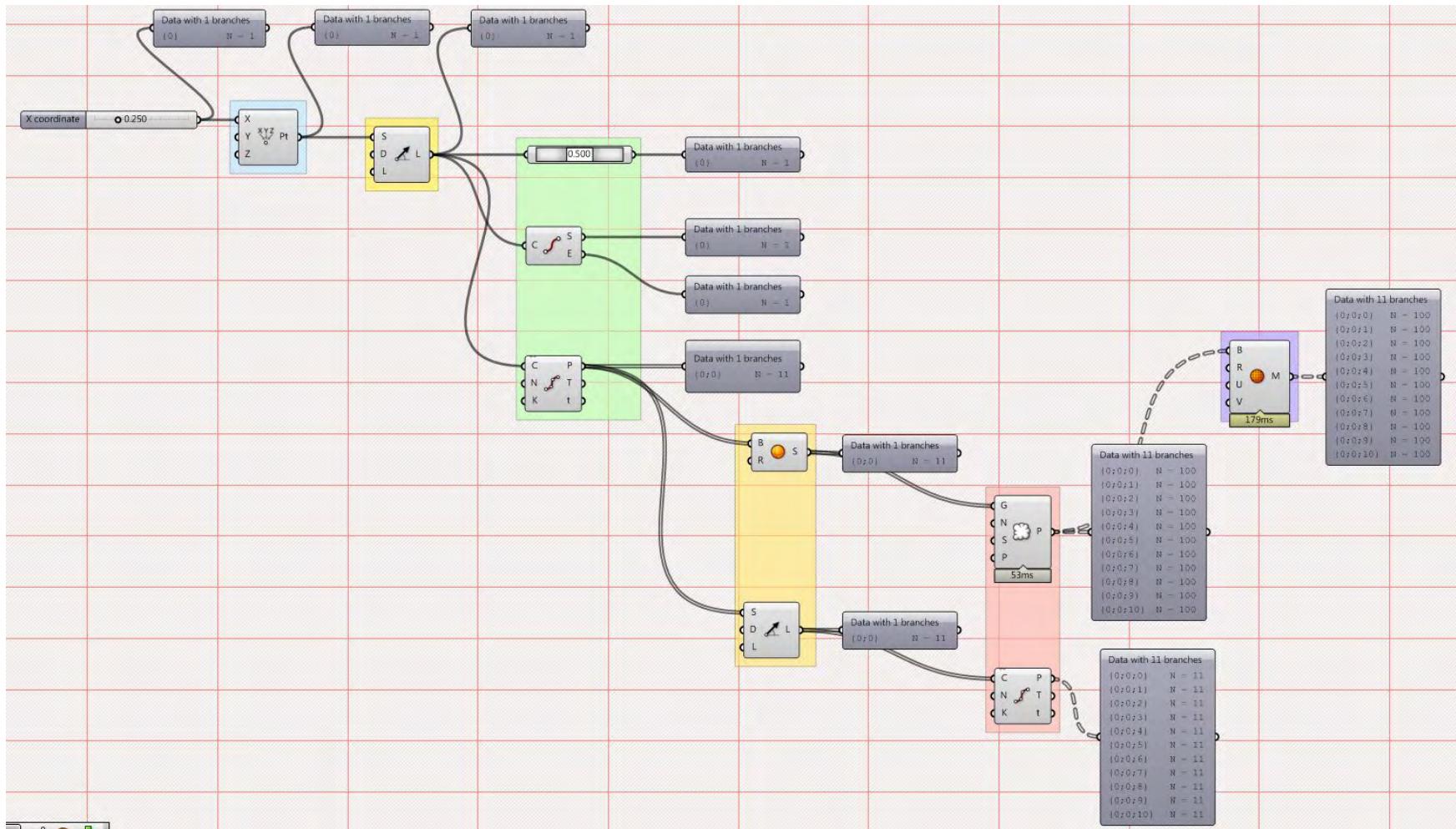


image source: Google



# Data Structure





## Curve.DivideByCount Method (Int32, Boolean, Point3d[])

Divide the curve into a number of equal-length segments.

**Namespace:** Rhino.Geometry

**Assembly:** RhinoCommon (in RhinoCommon.dll)

### ▲ Syntax

C#    VB

```
public double[] DivideByCount(
    int segmentCount,
    bool includeEnds,
    out Point3d[] points
)
```

Create new storage

## Curve.PointAt Method

Derived from  
Input geometry

Evaluates point at a curve parameter.

**Namespace:** Rhino.Geometry

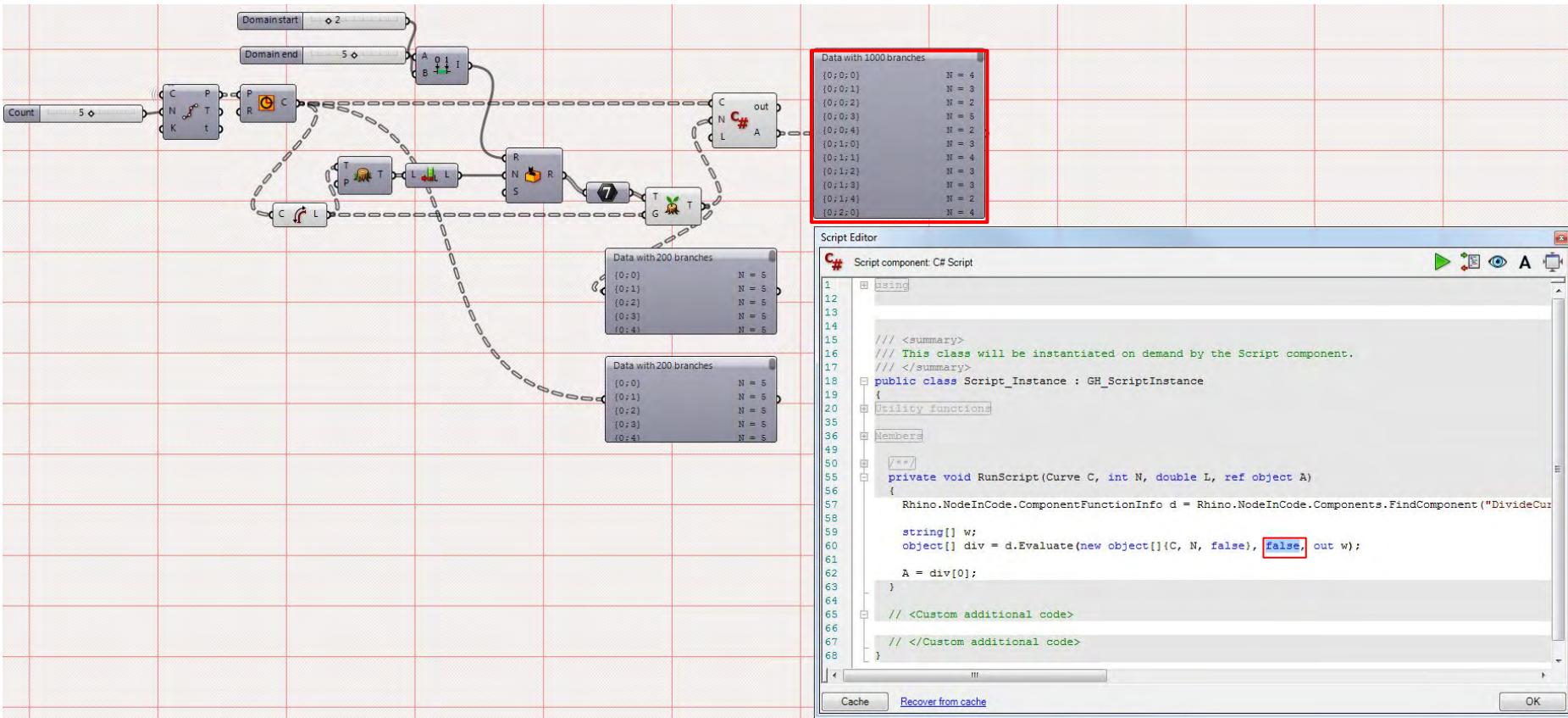
**Assembly:** RhinoCommon (in RhinoCommon.dll)

### ▲ Syntax

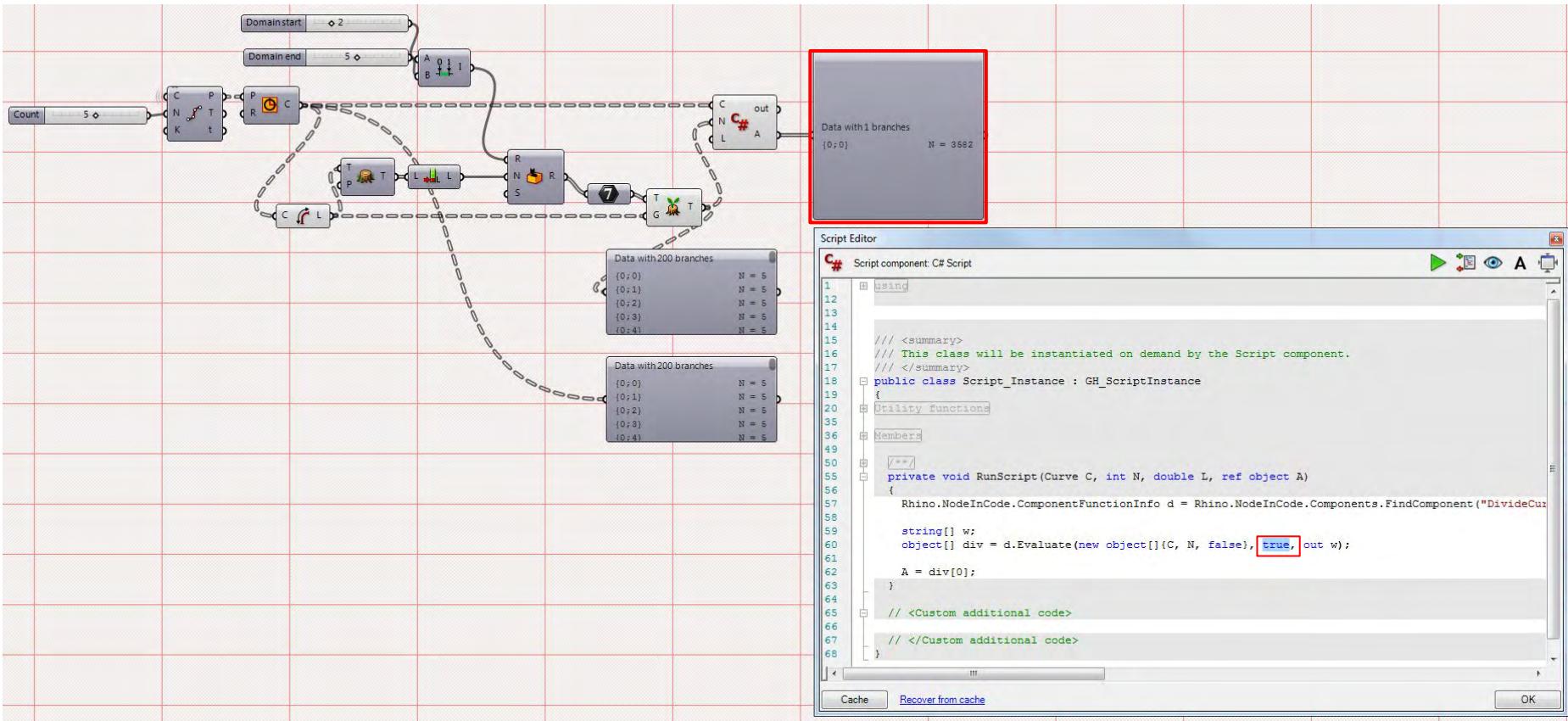
C#    VB

```
public Point3d PointAt(
    double t
)
```

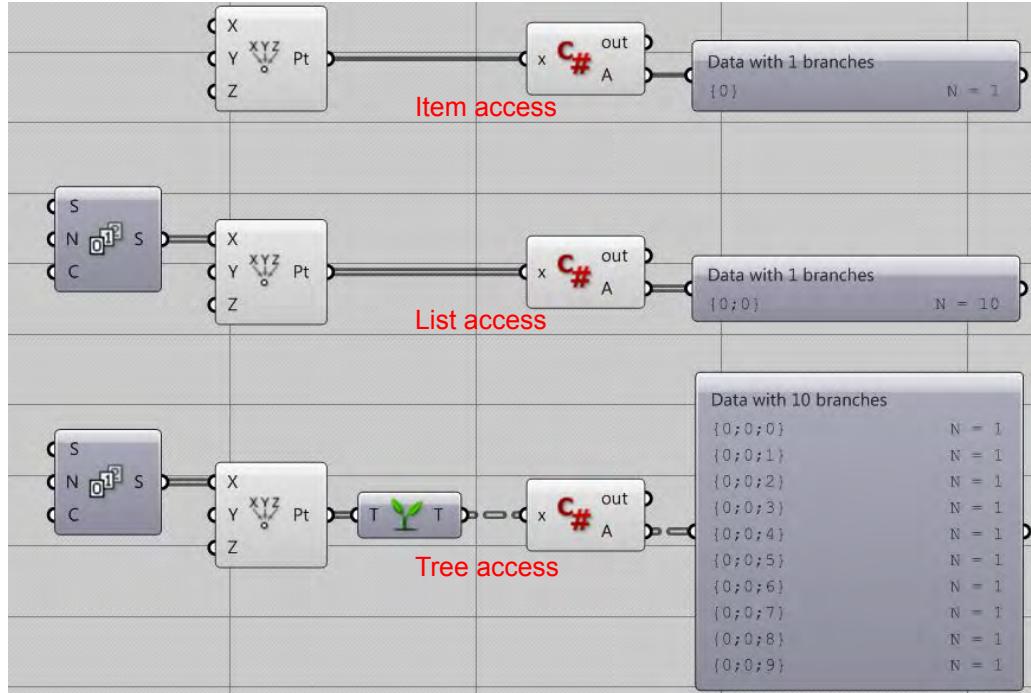
# Data Structure Control with NodeInCode



# Data Structure Control with NodeInCode



## Data Structure - data access



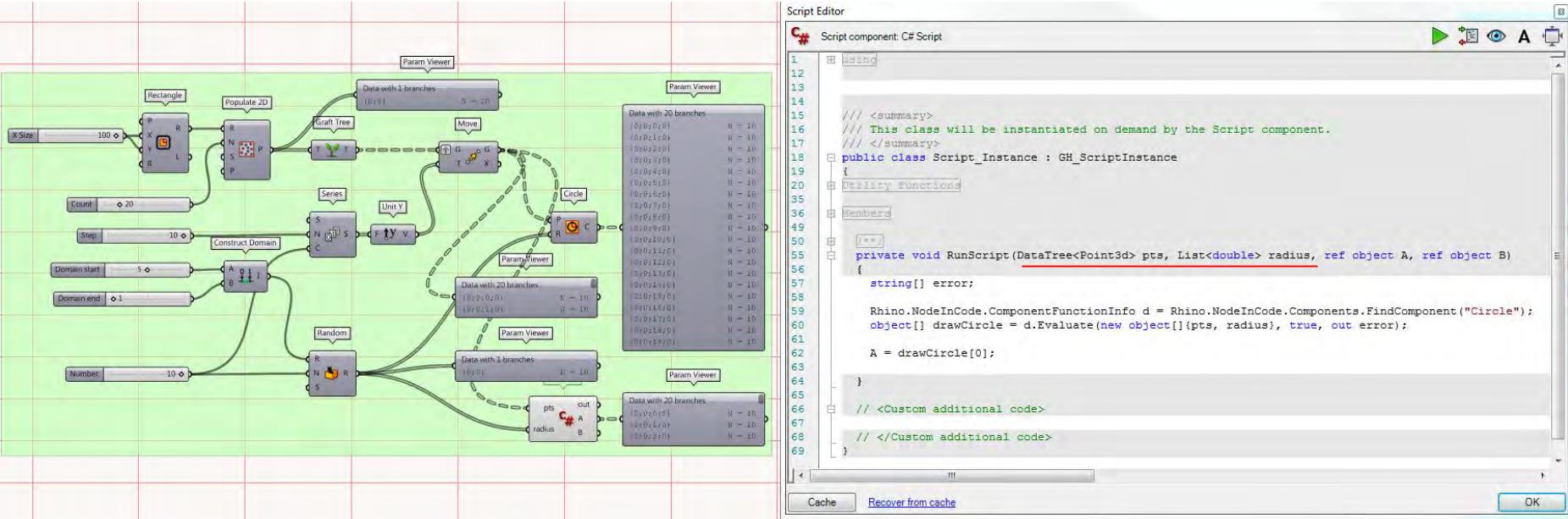
```
private void RunScript(Point3d x, ref object A)
{
    A = new Circle(x, 5);
}

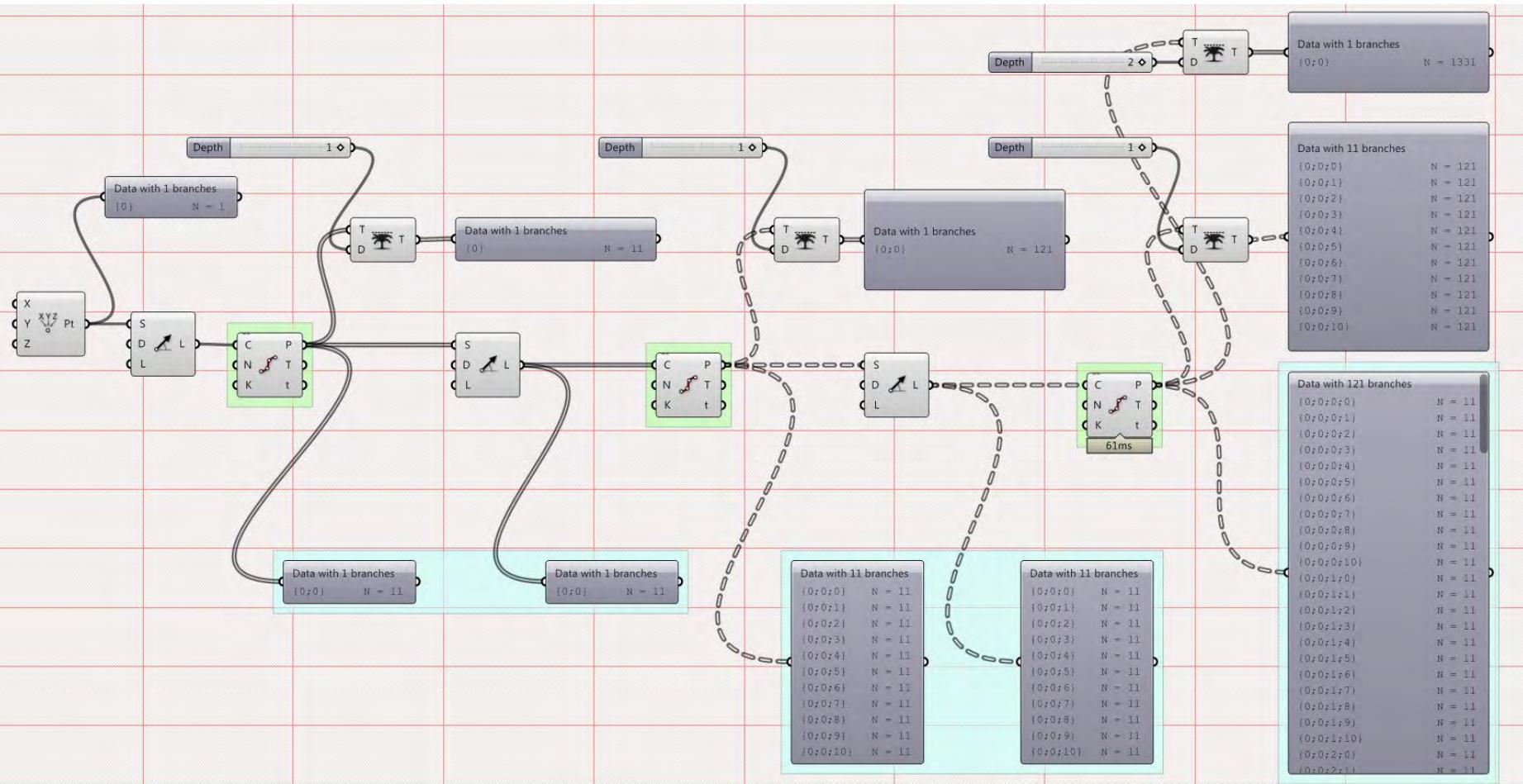
private void RunScript(List<Point3d> x, ref object A)
{
    var c = new List<Circle>();
    foreach (var pt in x){
        Circle cir = new Circle(pt, 5);
        c.Add(cir);
    }
    A = c;
}

private void RunScript(DataTree<Point3d> x, ref object A)
{
    DataTree<Circle> c = new DataTree<Circle>();
    foreach (var p in x.Paths){
        foreach (var pt in x.Branch(p)){
            Circle cir = new Circle(pt, 5);
            c.Add(cir, p);
        }
    }
    A = c;
}
```

# Data matching with multiple objects

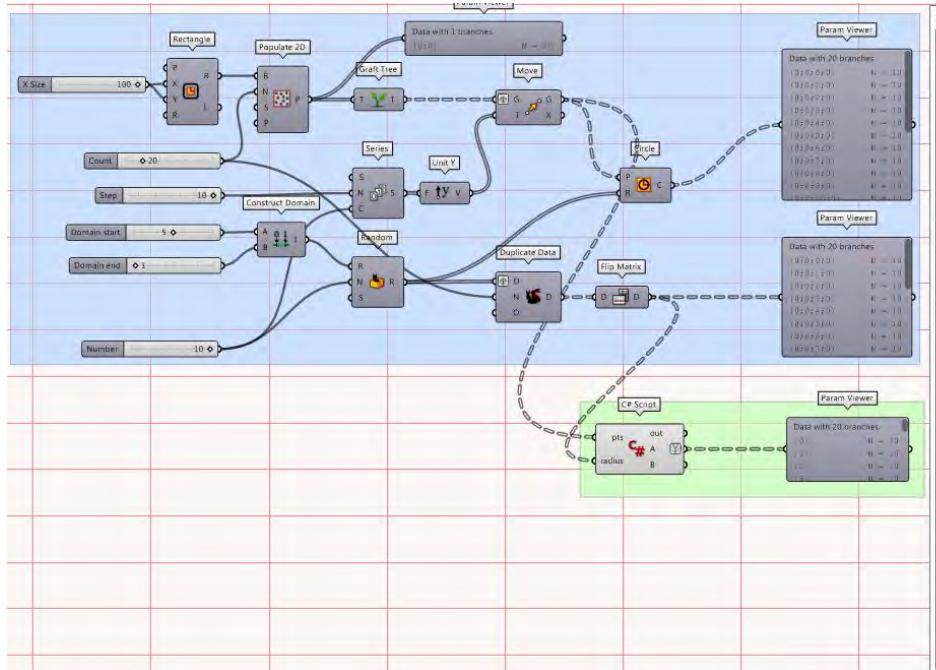
## Tree / List Access solution





# Data matching with multiple objects

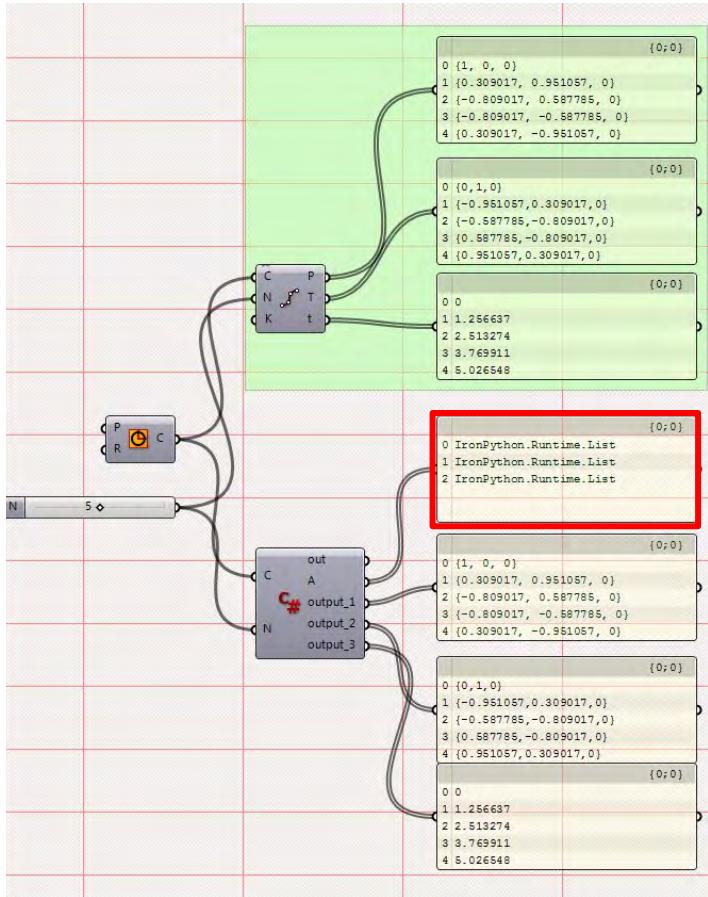
## Item Access solution



The screenshot shows the 'Script Editor' window with the following code:

```
1 //<summary>
2 //<summary> This class will be instantiated on demand by the Script component.
3 //</summary>
4 public class Script_Instance : GH_ScriptInstance
5 {
6     //<summary>
7     //<summary>
8     public void RunScript(Point3d pts, double radius, ref object A, ref object B)
9     {
10        Rhino.NodeInCode.ComponentFunctionInfo d = Rhino.NodeInCode.Components.FindComponent("Circle");
11
12        string[] error;
13        object[] drawCircle = d.Evaluate(new object[]{pts, radius}, false, out error);
14
15        Rhino.NodeInCode.ComponentFunctionInfo e = Rhino.NodeInCode.Components.FindComponent("TrimTree");
16
17        object[] output = e.Evaluate(new object[]{drawCircle[0]}, true, out error);
18
19        A = output[0];
20
21    }
22
23    //<Custom additional code>
24
25    //</Custom additional code>
26 }
```

# IronPython to C#



```
public abstract Object[] Evaluate(
    IEnumerable args,
    bool keepTree,
    out string[] warnings
)
```

## Parameters

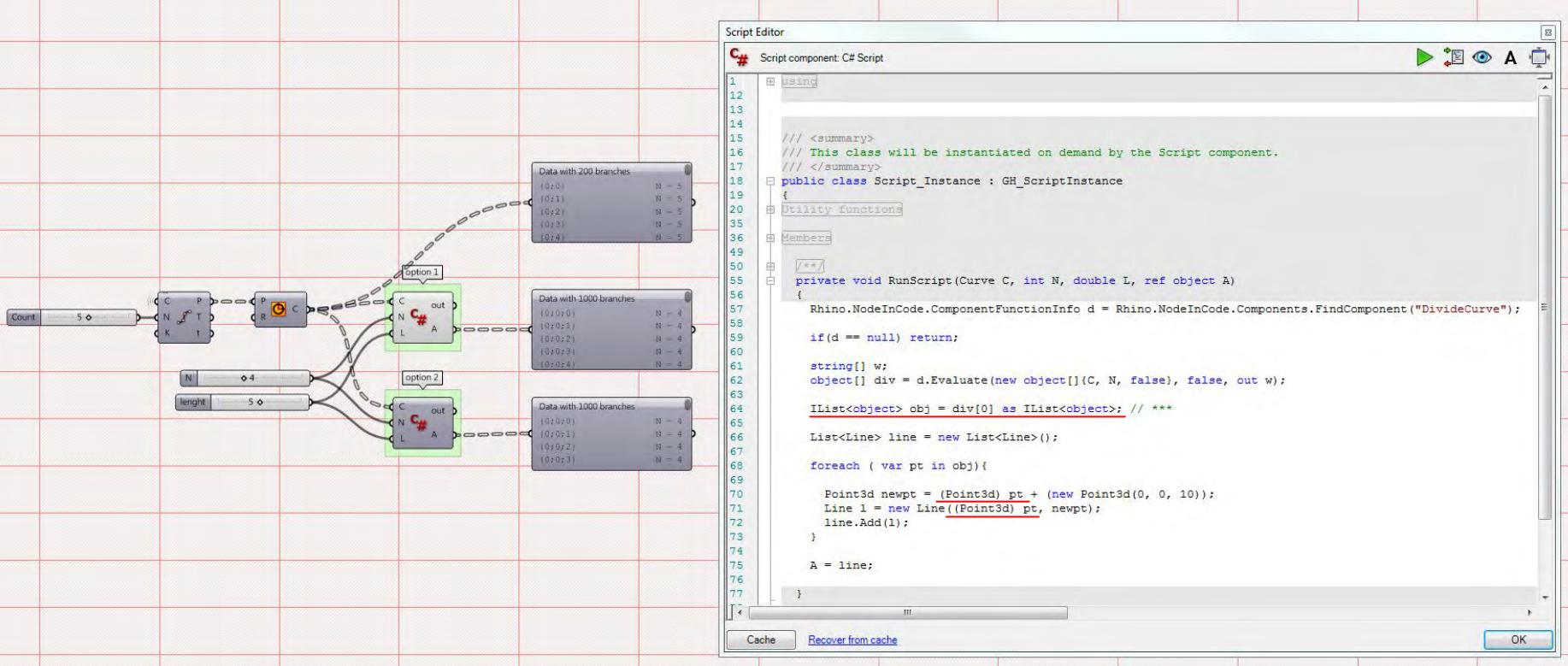
### args

Type: System.Collections.IEnumerable

The arguments list. Each item is assigned to each input param, in order.

IronPython.Runtime.List implements the following interfaces:  
IEnumerable<object> / ICollection<object> / IList<object>

# NodeInCode - work with List



The screenshot shows a Grasshopper interface with a workflow for dividing a curve into multiple segments. The workflow starts with a 'Count' input (5) which feeds into a 'P' component. The output of 'P' is connected to two parallel 'C' components. The first 'C' component has an 'N' input (4) and an 'L' input (5). Its output is labeled 'option 1'. The second 'C' component also has an 'N' input (4) and an 'L' input (5). Its output is labeled 'option 2'. Both 'option 1' and 'option 2' outputs feed into a 'Data with 200 branches' component. This component has a table showing 200 branches, each with N = 5. The second 'C' component's output also feeds into a 'Data with 1000 branches' component. This component has a table showing 1000 branches, each with N = 4.

**Script Editor (C# Script Component):**

```
1  //<summary>
2  /// This class will be instantiated on demand by the Script component.
3  ///</summary>
4  public class Script_Instance : GH_ScriptInstance
5  {
6  }
7
8  public void RunScript(Curve C, int N, double L, ref object A)
9  {
10    Rhino.NodeInCode.ComponentFunctionInfo d = Rhino.NodeInCode.Components.FindComponent("DivideCurve");
11
12    if(d == null) return;
13
14    string[] w;
15    object[] div = d.Evaluate(new object[]{C, N, false}, false, out w);
16
17    IList<object> obj = div[0] as IList<object>; // ***
18
19    List<Line> line = new List<Line>();
20
21    foreach (var pt in obj)
22    {
23
24      Point3d newpt = (Point3d) pt + (new Point3d(0, 0, 10));
25      Line l = new Line((Point3d) pt, newpt);
26      line.Add(l);
27    }
28
29    A = line;
30  }
31
32  public void Cache()
33  {
34  }
35
36  public void RecoverFromCache()
37  {
38  }
39
40  public void OnClose()
41  {
42  }
43
44  public void OnSave()
45  {
46  }
47
48  public void OnLoad()
49  {
50  }
51
52  public void OnDelete()
53  {
54  }
55
56  public void OnEdit()
57  {
58  }
59
60  public void OnCopy()
61  {
62  }
63
64  public void OnCut()
65  {
66  }
67
68  public void OnPaste()
69  {
70  }
71
72  public void OnSelect()
73  {
74  }
75
76  public void OnDeselect()
77  {
78  }
79
80  public void OnDoubleClick()
81  {
82  }
83
84  public void OnRightClick()
85  {
86  }
87
88  public void OnEnter()
89  {
90  }
91
92  public void OnLeave()
93  {
94  }
95
96  public void OnFocus()
97  {
98  }
99
100 public void OnBlur()
101 {
102 }
```

# NodeInCode - work with DataTree

The screenshot displays the NodeInCode interface, which consists of a workflow graph on the left and a Script Editor on the right.

**Workflow Graph:**

- The graph starts with a **Number** node (value 100) connected to a **Point** node (P).
- A **Loop** node (Count: 20, Step: 10) iterates over the workflow.
- Inside the loop:
  - A **Range** node (Domain start: 5, Domain end: 1) generates a range of values.
  - A **Switch** node (A: 0, B: 1, C: 2) branches the flow based on the value of A.
  - If A is 0, it goes through a **Point** node (R) and a **List** node (S) to a **Data with 1 branches** node (N = 0).
  - If A is 1, it goes through a **Point** node (R) and a **List** node (S) to a **Data with 1 branches** node (N = 1).
  - If A is 2, it goes through a **Point** node (R) and a **List** node (S) to a **Data with 20 branches** node (N = 20).
- After the loop, a **Point** node (R) and a **Number** node (10) are combined into a **Point** node (R).
- The final output is a **Text** node (X) with the value 5.

**Script Editor:**

```

C# Script component: C# Script

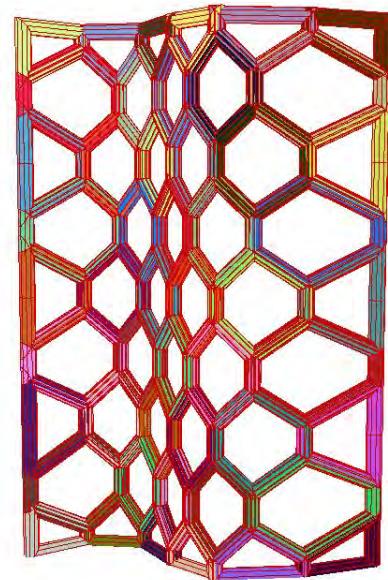
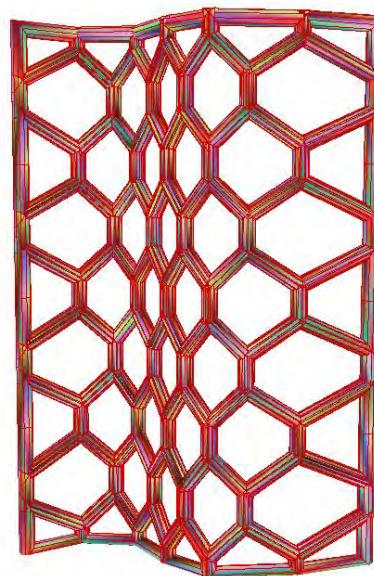
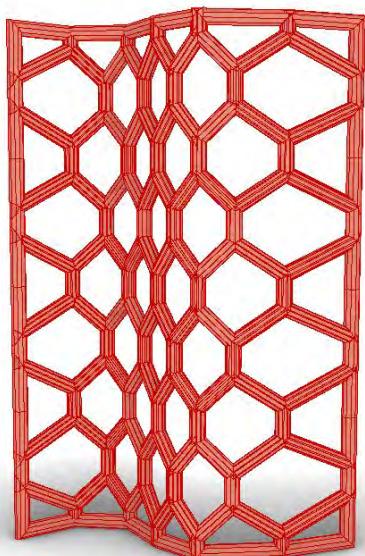
1  //<summary>
2  /// This class will be instantiated on demand by the Script component.
3  /// </summary>
4  public class Script_Instance : GH_ScriptInstance
5  {
6      // Utility functions
7
8      // Members
9
10     //<!--
11     private void RunScript(DataTree<Point3d> pts, List<double> radius, int division, ref object A, ref object
12     Rhino.NodeInCode.ComponentFunctionInfo d = Rhino.NodeInCode.Components.FindComponent("Circle");
13
14     string[] error;
15     object[] drawCircle = d.Evaluate(new object[]{pts, radius}, true, out error);
16
17     DataTree<object> tree = drawCircle[0] as DataTree<object>;
18
19     DataTree<Circle> circletree = new DataTree<Circle>();
20
21     foreach ( var p in tree.Paths){
22         foreach ( var c in tree.Branch(p)){
23             Circle ci = (Circle) c;
24
25             circletree.Add(ci, p);
26         }
27     }
28
29     A = circletree;
30
31     Rhino.NodeInCode.ComponentFunctionInfo e = Rhino.NodeInCode.Components.FindComponent("DivideCurve");
32     object[] result2 = e.Evaluate(new object[]{circletree, division}, true, out error);
33
34     B = result2[0];
35
36     //}
37 }
```

The script performs the following steps:
 

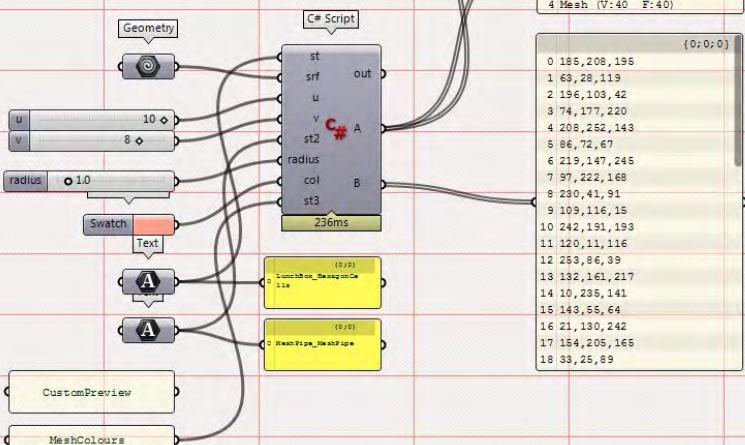
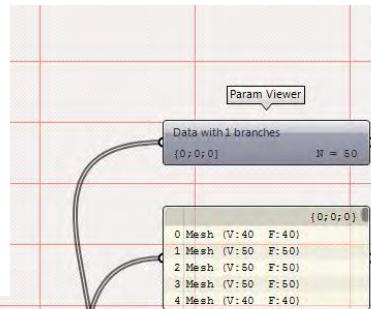
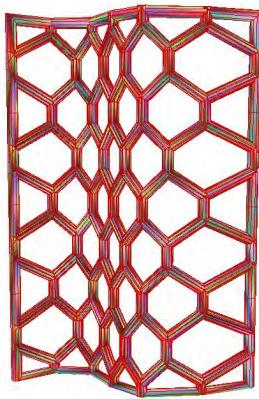
- It uses the `Circle` component to draw circles based on the input points and radius.
- It creates a **DataTree<Circle>** object to store the drawn circles.
- It iterates over the paths and branches of the input DataTree to add each circle to the tree.
- Finally, it returns the **circletree** and the result of the `DivideCurve` component.

Example - Using third party plug-in(s)

- Hybrid NodeInCode workflow / scenario



# NodeInCode method - data matching issue



Script Editor

C# Script component: C#

```
13  using Rhino.NodeInCode;
14  using System.Drawing;
15
16
17  /// <summary>
18  /// This class will be instantiated on demand by the Script component.
19  /// </summary>
20  public class Script_Instance : GH_ScriptInstance
21  {
22      Utility functions
23
24      Members
25
26      /* */
27
28      private void RunScript(string st, Surface srf, int u, int v, string st2, double radius, Color col, string st3, ref object[] output, ref string[] warning)
29      {
30          var com = Components.FindComponent(st);
31          var output1 = com.Evaluate(new object[]{srf,u,v}, false, out warning);
32
33          var com2 = Components.FindComponent(st2);
34          var output2 = com2.Evaluate(new object[]{output[0], radius}, false, out warning);
35
36          var com3 = Components.FindComponent(st3);
37
38          IList<object> convert = (IList<object>) output2[0];
39          int count = convert.Count;
40
41          List<object> final = new List<object>();
42          List<Color> co = new List<Color>();
43
44          for (int i = 0; i < count; i++)
45          {
46              var r = new Random(i);
47              Color c = Color.FromArgb(r.Next(0, 255), r.Next(0, 255), r.Next(0, 255));
48              co.Add(c);
49          }
50
51
52          var output3 = com3.Evaluate(new object[]{output2[0], co}, false, out warning);
53
54          A = output3[0];
55          B = co;
56
57      }
58  }
```

Cache

Recover from cache

OK

# Shape Grammar

Shape Grammar is computation theory that defines a formalism to represent visual and spatial thinking, and was introduced by George Stiny and James Gips in 1972. Shape grammars are descriptive and generative, combining a set of shape rules and transformation methods to create geometric patterns.

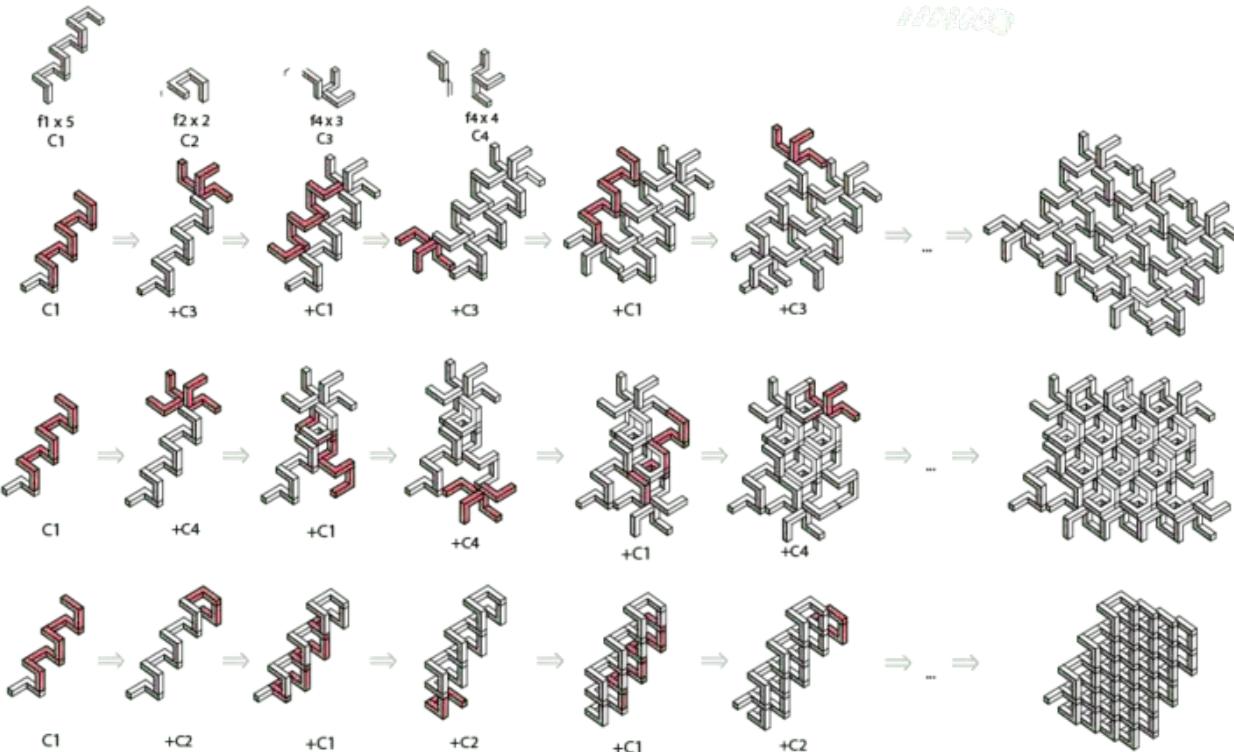


image source: Google

Shape Grammar Introduction  
<https://www.youtube.com/watch?v=faycjymzTzY>

MIT  
[http://www.mit.edu/~tknight/IJDC/page\\_introduction.htm#:~:text=A%20shape%20grammar%20is%20a,forms%20of%20the%20generated%20designs.](http://www.mit.edu/~tknight/IJDC/page_introduction.htm#:~:text=A%20shape%20grammar%20is%20a,forms%20of%20the%20generated%20designs.)

# L system (Lindenmayer-system)

## Example 1: Algae [edit]

Lindenmayer's original L-system for modelling the growth of algae.

**variables** : A B  
**constants** : none  
**axiom** : A  
**rules** : (A → AB), (B → A)

which produces:

n = 0 : A  
n = 1 : AB  
n = 2 : ABA  
n = 3 : ABAAB  
n = 4 : ABAABABA  
n = 5 : ABAABABAABAAB  
n = 6 : ABAABABAABAABAABAAB  
n = 7 : ABAABABAABAABAABAABAABAAB

## Example 1: Algae, explained [edit]

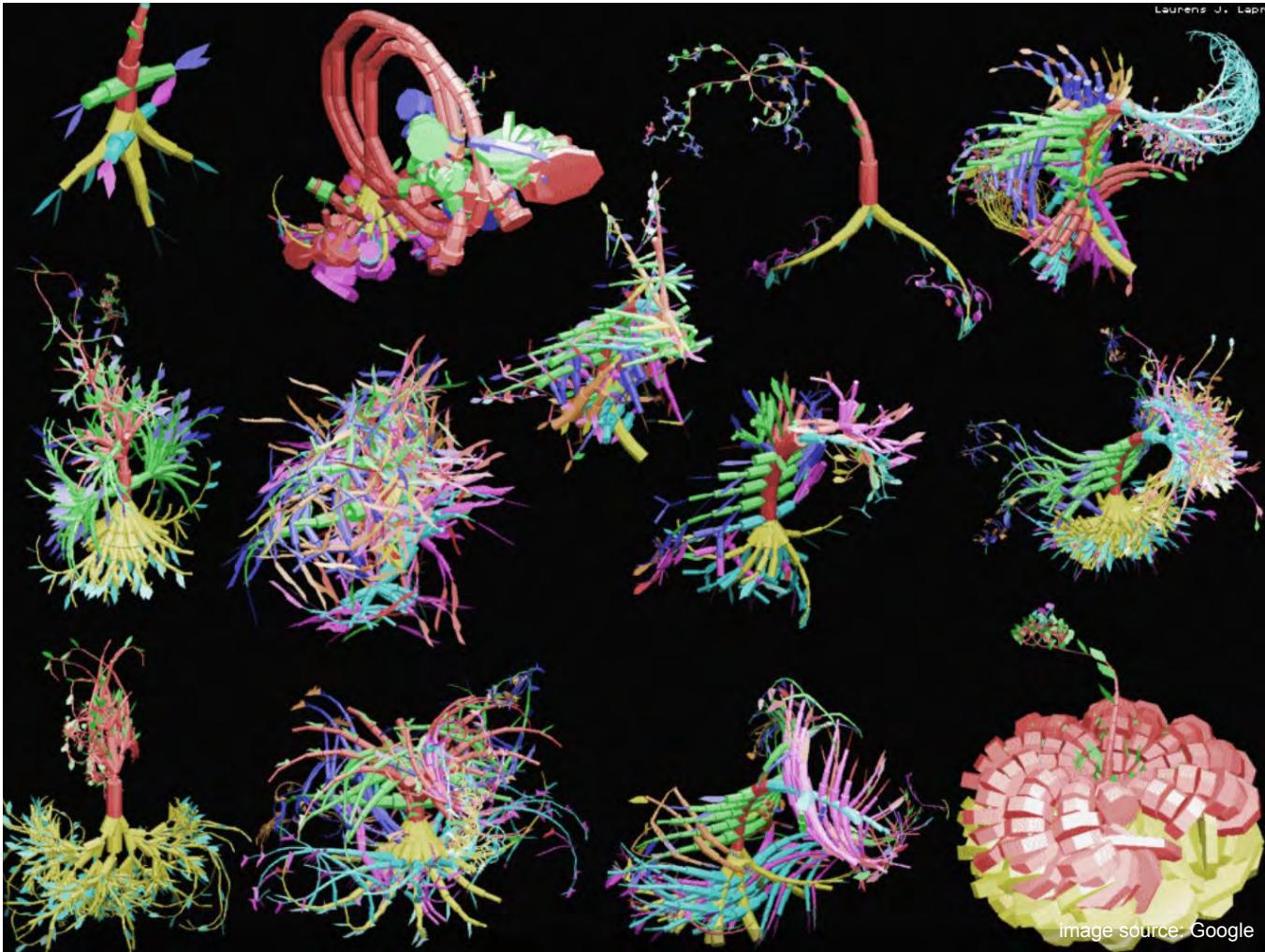
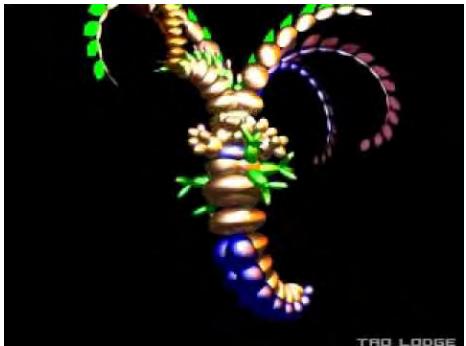
n=0:	A	start (axiom/initiator)
	/ \	
n=1:	A    B	the initial single A spawned into AB by rule (A → AB), rule (B → A) couldn't be applied
	/     \	
n=2:	A B    A	former string AB with all rules applied, A spawned into AB again, former B turned into A
	/        \	
n=3:	A B A    A B	note all A's producing a copy of themselves in the first place, then a B, which turns ...
	/       \ \	
n=4:	A B A A B    A B A	... into an A one generation later, starting to spawn/repeat/recuse then



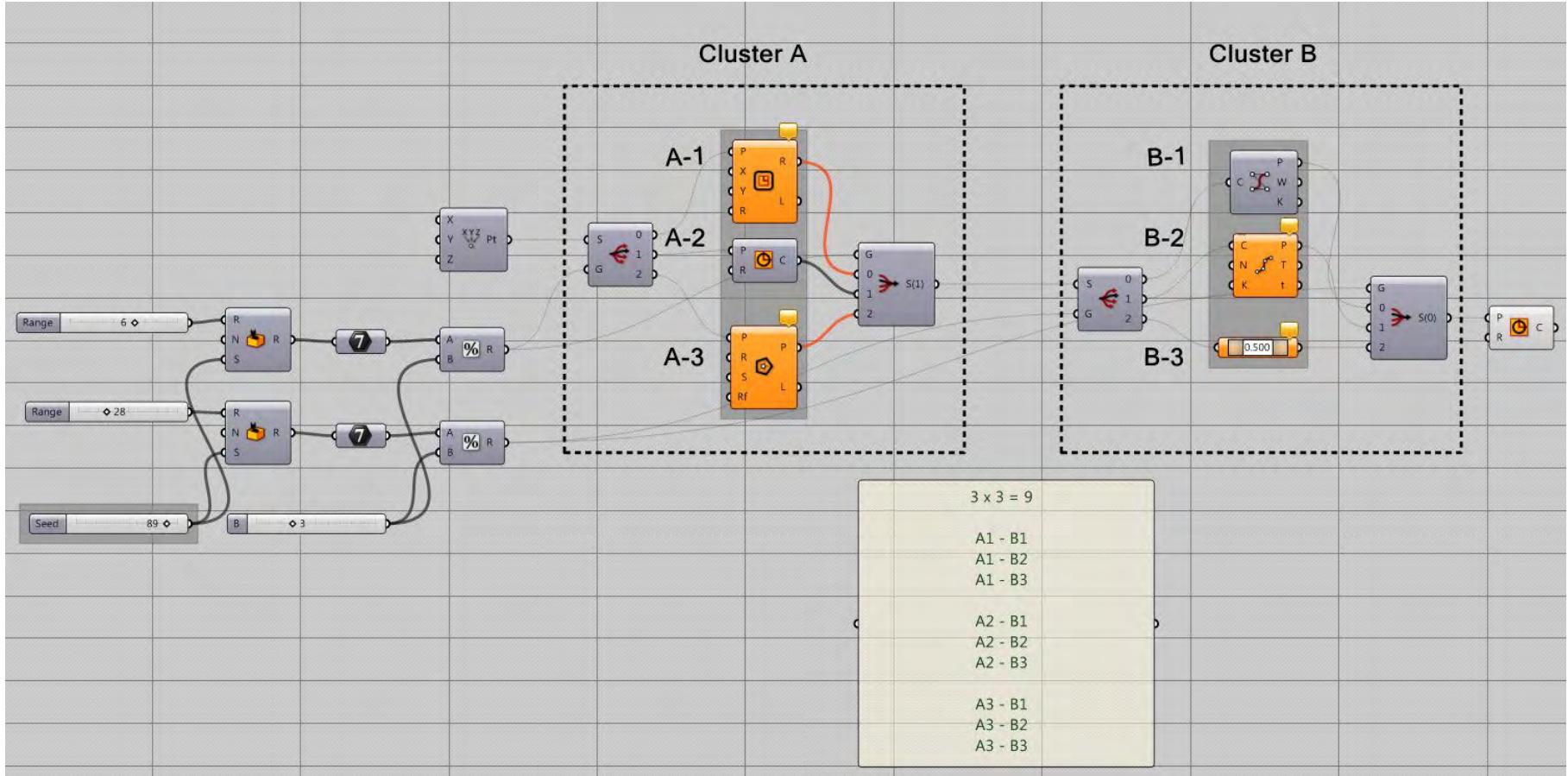
## L-parser

Lsystem parser allows mutations to take place during the growing of a form. These mutations can change the form slightly or dramatically. It allows a form to be created and then create a whole series of forms all clearly descendant from the original Lsystem.

<http://laurenslapre.nl/ViewLparser/ViewLparser.html>



## Example - random composition



# Shape Grammar based on iteration and combination



Circle + End Points  
Rectangle + Control Points  
Circle + End Points

Circle + End Points  
Rectangle + Control Points  
Polygon + End Points

Polygon + End Points  
Rectangle + Control Points  
Polygon + End Points

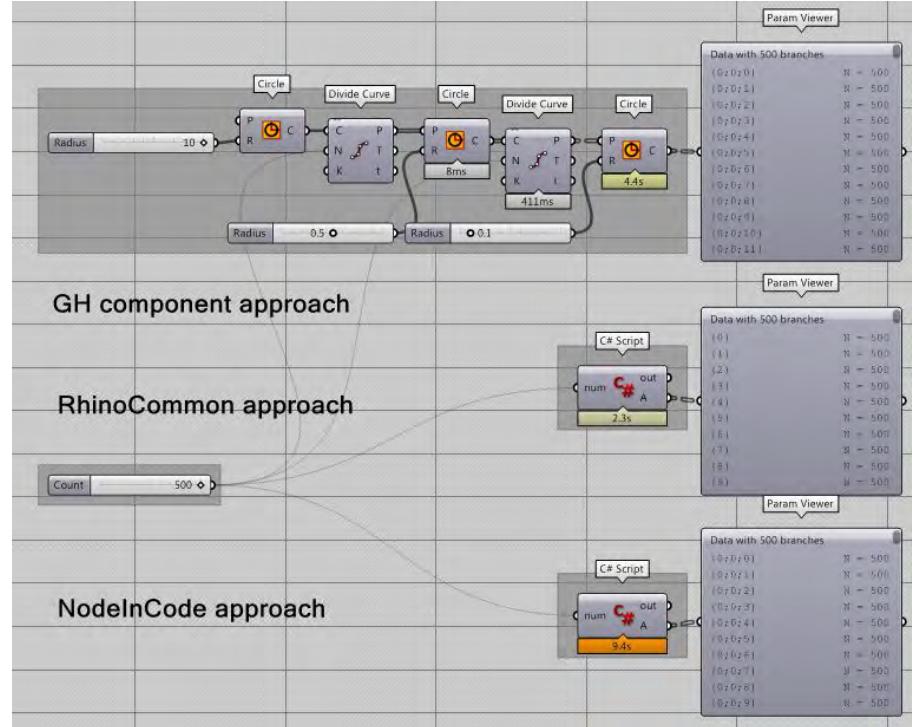
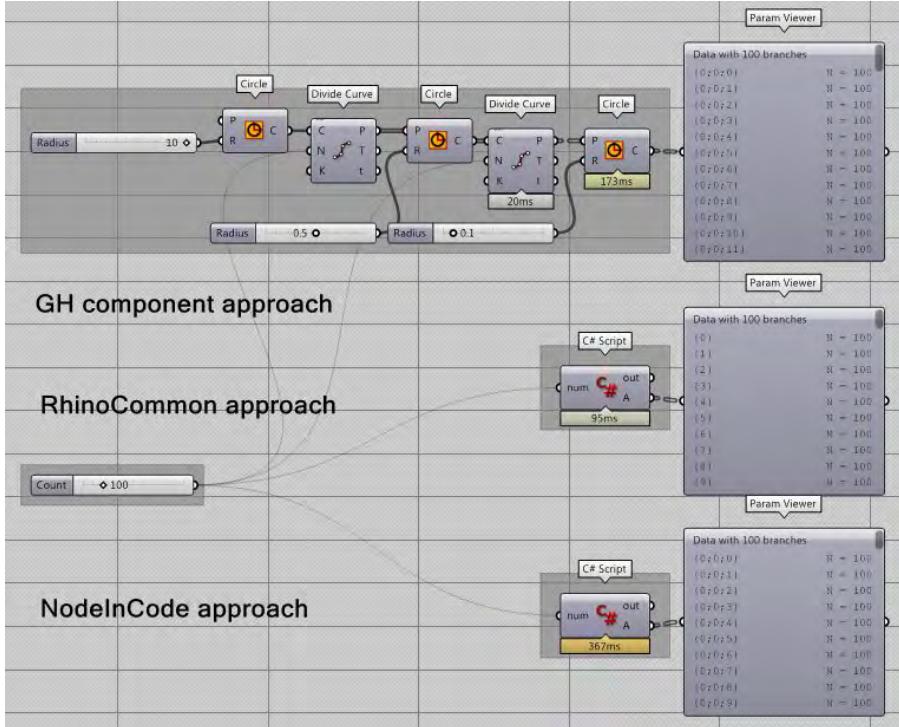


Rectangle + Control Points  
Polygon + Divide Curve  
Rectangle + Control Points

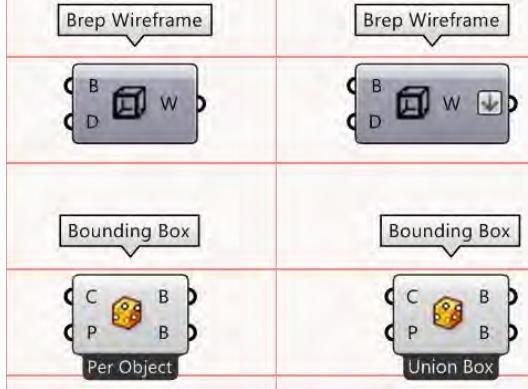


Polygon + Divide Curve  
Rectangle + Control Points  
Polygon + Divide Curve

# NodeInCode is very convenient, but any downside?



## NodeInCode - quick summary

Pros	Cons
<ul style="list-style-type: none"><li>* More intuitive and similar to component assembly principle in GH</li><li>* Easy to compose a series of Rhinocommon function together</li><li>* Easy access to plug-in functionality</li><li>* Easy to conduct interoperability with other platforms (if the function exists)</li><li>* Able to access components not documented in Rhinocommon</li></ul>	<ul style="list-style-type: none"><li>* Cannot access parameters in components</li></ul>  <ul style="list-style-type: none"><li>* Some components are not supported</li><li>* Not suitable for external use</li><li>* Performance</li></ul>

**Thank you very much!**

If you have any questions, please feel free to  
drop me an email: [ping-hsiang@dezact.org](mailto:ping-hsiang@dezact.org)