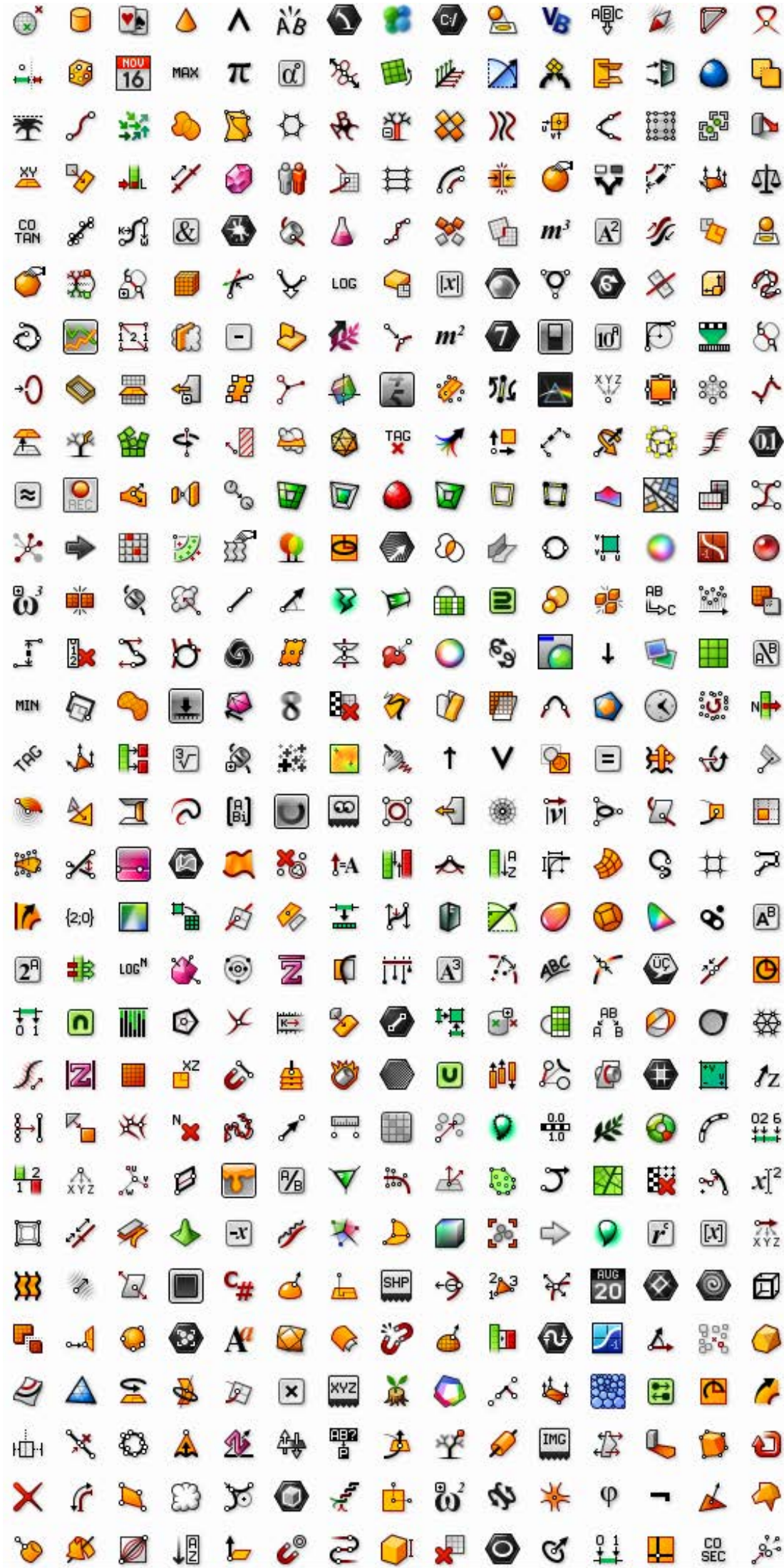


FOUNDATIONS

THE GRASSHOPPER PRIMER THIRD EDITION



I.0

Preface

WELCOME

You have just opened the third edition of the Grasshopper Primer. This primer was originally written by Andrew O. Payne of Lift Architects for Rhino4 and Grasshopper version 0.6.0007 which, at the time of its release, was a giant upgrade to the already robust Grasshopper platform. We now find ourselves at another critical shift in Grasshopper development, so a much needed update to the existing primer was in order. We are thrilled to add this updated primer to the many amazing contributions put forth by Grasshopper community members. With an already excellent foundation from which to build, our team at Mode Lab went to work designing and developing the look and feel of the third edition. This revision provides a comprehensive guide to the most current Grasshopper build, version 0.90076, highlighting what we feel are some of the most exciting feature updates. The revised text, graphics, and working examples are intended to teach visual programming to the absolute beginner, as well as provide a quick introduction to Generative Design workflows for the seasoned veteran. It is our goal that this primer will serve as a field guide to new and existing users looking to navigate the ins and outs of using Grasshopper in their creative practice.

FOUNDATIONS

This primer, titled Foundations, introduces you to the fundamental concepts and essential skill-building workflows to effectively use Grasshopper. Foundations is the first volume in a forthcoming collection of Grasshopper primers (to be released in the coming months). This first volume tracks through an introduction to the Grasshopper user interface, the anatomy of a Grasshopper definition, and three chapters dealing with parametric concepts and case study examples. An appendix closes out this volume, providing you with resources for continued exploration.

We hope that at the very least this primer will inspire you to begin exploring the many opportunities of programming with Grasshopper. We wish you the best of luck as you embark on this journey.

Gil Akos
Mode Lab
<http://modelab.is/>

Ronnie Parsons
Mode Lab
<http://modelab.is/>

A special thanks to David Rutten for the endless inspiration and invaluable work pioneering Grasshopper. We would also like to thank Andrew O. Payne for providing the assets from which this work initiated. Lastly, many thanks to Bob McNeel and everyone at Robert McNeel & Associates for their generous support over the years.

REQUIRED SOFTWARE

Rhino5

Rhino 5.0 is the market leader in industrial design modeling software. Highly complicated shapes can be directly modeled or acquired through 3D digitizers. With its powerful NURBS based engine Rhino 5.0 can create, edit, analyze, and translate curves, surfaces, and solids. There are no limits on complexity, degree, or size.

<http://www.rhino3d.com/download/rhino/5/latest>

Grasshopper

For designers who are exploring new shapes using generative algorithms, Grasshopper is a graphical algorithm editor tightly integrated with Rhino's 3D modeling tools. Unlike RhinoScript or Python, Grasshopper requires no knowledge of the abstract syntax of scripting, but still allows designers to build form generators from the simple to the awe inspiring.

<http://www.grasshopper3d.com/page/download-1>

FORUMS

The Grasshopper forum is very active and offers a wonderful resource for posting questions/answers and finding help on just about anything.

The forum has categories for general discussion, errors & bugs, samples & examples, and FAQ.

<http://www.grasshopper3d.com/forum>

The Common Questions section of the Grasshopper site contains answers to many questions you may have, as well as helpful links:

<http://www.grasshopper3d.com/notes/index/allNotes>

McNeel Forums

For more general questions pertaining to Rhino3D be sure to check out the McNeel Forum powered by Discourse.

<http://discourse.mcneel.com/>

GENERAL REFERENCES

Essential Mathematics

Essential Mathematics uses Grasshopper to introduce design professionals to foundation mathematical concepts that are necessary for effective development of computational methods for 3D modeling and computer graphics. Written by Rajaa Issa.

<http://www.rhino3d.com/download/rhino/5.0/EssentialMathematicsThirdEdition/>

Wolfram MathWorld

MathWorld is an online mathematics resource., assembled by Eric W. Weisstein with assistance from thousands of contributors. Since its contents first appeared online in 1995, MathWorld has emerged as a nexus of mathematical information in both the mathematics and educational communities. Its entries are extensively referenced in journals and books spanning all educational levels.

<http://mathworld.wolfram.com/>

THINGS TO REMEMBER

- Grasshopper is a graphical algorithm editor that is integrated with Rhino3D's modeling tools.
- Algorithms are step by step procedures designed to perform an operation
- You use Grasshopper to design algorithms that then automate tasks in Rhino3D.
- An easy way to get started if you are unclear how to perform a specific operation in Grasshopper would be to try manually and incrementally creating an algorithm using Rhino commands.

I.1

Contents

[I.] Introduction

I.0 Preface	2	
I.1 Contents	4	
I.2 Grasshopper - an Overview	8	→ http://www.grasshopper3d.com
I.3 Grasshopper in Action	10	

[F.] Foundations

F.0 Hello Grasshopper	14	
F.0.0 Installing & Launching Grasshopper	16	
F.0.0.0 Downloading	16	→ http://www.grasshopper3d.com/page/download-1
F.0.0.1 Installing	16	
F.0.0.2 Launching	17	
F.0.1 The Grasshopper UI	18	
F.0.1.0 The Windows Title BAR	18	
F.0.1.1 Main Menu Bar	19	
F.0.1.2 File Browser Control	19	
F.0.1.3 Component Palettes	21	
F.0.1.4 The Canvas	22	
F.0.1.5 Grouping	22	
F.0.1.6 Widgets	23	
F.0.1.7 Using the Search Feature	24	
F.0.1.8 The Find Feature	24	
F.0.1.9 Using The Radial Menu	25	
F.0.1.10 The Canvas Toolbar	26	
F.0.2 Talking to Rhino	28	
F.0.2.0 Viewport Feedback	28	
F.0.2.1 Live Wires	29	
F.0.2.2 Gumball Widget	29	
F.0.2.3 Baking Geometry	30	
F.0.2.4 Units & Tolerances	30	
F.0.2.5 Remote Control Panel	31	
F.0.2.6 File Management	32	
F.0.2.7 Templates	32	
F.1 Anatomy of a Grasshopper Definition	34	
F.1.0 Grasshopper Object Types	36	
F.1.0.0 Parameters	36	
F.1.0.1 Components	36	
F.1.0.2 Object Colors	37	

F.1.1 Grasshopper Component Parts	38	
F.1.1.0 Label vs Icon Display	38	
F.1.1.1 Component Help	40	
F.1.1.2 Tool Tips	40	
F.1.1.3 Context Popup Menus	41	
F.1.1.4 Zoomable User Interface	42	
F.1.2 Data Types	43	
F.1.2.0 Persistent Data	43	
F.1.2.1 Volatile Data	44	
F.1.2.2 Input Parameters	45	
F.1.3 Wiring Components	48	
F.1.3.0 Connection Management	48	
F.1.3.1 Fancy Wires	49	
F.1.3.2 Wire Display	49	
F.1.4 The Grasshopper Definition	50	→ f.1.4_the grasshopper definition.gh
F.1.4.0 Program Flow	50	
F.1.4.1 The Logical Graph	50	
F.2 Building Blocks of Algorithms	52	
F.2.0 Points, Planes & Vectors	54	
F.2.0.0 Points	55	
F.2.0.1 Vectors	55	
F.2.0.2 Planes	55	
F.2.1 Working with Attractors	56	
F.2.1.0 Attractor definition	57	→ f.2.1.0_attractor definition.gh
F.2.2 Mathematics, Expressions & Conditionals	60	→ f.2.2_operators and conditionals.gh
F.2.2.0 The Math Tab	60	
F.2.2.1 Operators	61	
F.2.2.2 Conditional Operators	61	
F.2.2.3 Trigonometry Components	62	→ f.2.2.3_trigonometry components.gh
F.2.2.4 Expressions	65	→ f.2.2.4_expressions.gh
F.2.3 Domains & Color	68	→ f.2.3_domains and color.gh
F.2.4 Booleans and Logical Operators	72	→ f.2.4_booleans and logical operators.gh
F.2.4.0 Booleans	72	
F.2.4.1 Logical Operators	72	
F.3 Designing with Lists	74	
F.3.0 Curve Geometry	76	
F.3.0.0 NURBS Curves	76	
F.3.0.1 Grasshopper Spline Components	77	→ f.3.0.1_grasshopper spline components.gh

F.3.1 What is a List?	79	
F.3.2 Data Stream Matching	80	→ f.3.2_data matching.gh
F.3.3 Creating Lists	84	→ f.3.3_list creation.gh
F.3.3.0 Manual List Creation	84	
F.3.3.1 Range	84	
F.3.3.2 Series	86	
F.3.3.3 Random	86	
F.3.4 List Visualization	87	→ f.3.4_list visualization.gh
F.3.4.0 The Point List component	87	
F.3.4.1 Text Tags	87	
F.3.4.2 Color	88	
F.3.5 List Management	89	→ f.3.5_list management.gh
F.3.5.0 List Length	89	
F.3.5.1 List Item	89	
F.3.5.2 Reverse List	89	
F.3.5.3 Shift List	90	
F.3.5.4 Insert Items	90	
F.3.5.5 Weave	91	
F.3.5.6 Cull Pattern	91	
F.3.6 Working with Lists	92	→ f.3.6_working with lists.gh
F.4 Designing with Data Trees	96	
F.4.0 Surface Geometry	98	
F.4.0.0 NURBS Surfaces	98	
F.4.0.1 Projecting, Mapping & Morphing	100	
F.4.0.2 Morphing Definition	101	→ f.4.0.2_morphing definition.gh
F.4.1 What is a Data Tree?	104	
F.4.1.0 Data Tree Visualization	105	→ f.4.1.0_data tree visualization.gh
F.4.3 Working with Data Trees	107	→ f.4.3_working with data trees.gh
F.4.3.0 Flatten Tree	107	
F.4.3.1 Graft Tree	107	
F.4.3.2 Simplify Tree	108	
F.4.3.3 Flip Matrix	108	
F.4.3.4 The Path Mapper	109	
F.4.3.5 Weaving Definition	110	→ f.4.3.5_weaving definition.gh
F.4.3.6 Rail Intersect Definition	116	→ f.4.3.6_rail intersect definition.gh
[A.] Appendix		
A.0 Index	124	

A.1 Resources	136
A.2 Notes	140
A.3 About this Primer	142

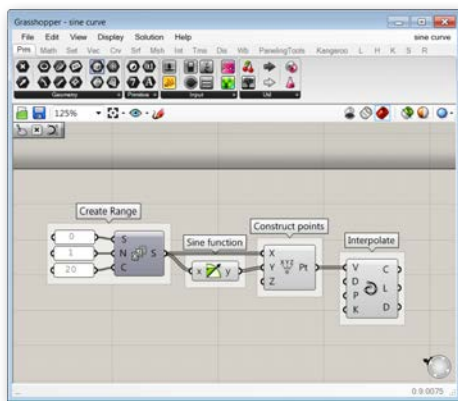
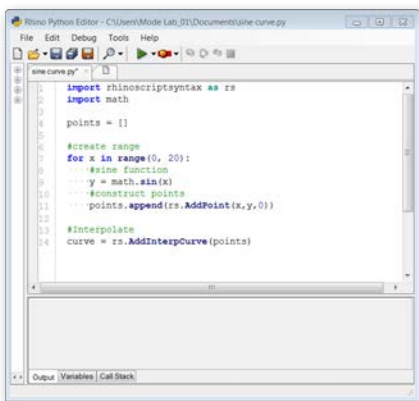
1.2 Grasshopper - an Overview

Grasshopper is a visual programming editor developed by David Rutten at Robert McNeel & Associates. As a plug-in for Rhino3D, Grasshopper is integrated with the robust and versatile modeling environment used by creative professionals across a diverse range of fields, including architecture, engineering, product design, and more. In tandem, Grasshopper and Rhino offer us the opportunity to define precise parametric control over models, the capability to explore generative design workflows, and a platform to develop higher-level programming logic – all within an intuitive, graphical interface.

The origins of Grasshopper can be traced to the functionality of Rhino3d Version 4's "Record History" button. This built-in feature enabled users to store modeling procedures implicitly in the background as you go. If you lofted four curves with the recording on and then edited the control points of one of these curves, the surface geometry would update. Back in 2008, David posed the question: "what if you could have more explicit control over this history?" and the precursor to Grasshopper, Explicit History, was born. This exposed the history tree to editing in detail and empowered the user to develop logical sequences beyond the existing capabilities of Rhino3D's built in features. Six years later, Grasshopper is now a robust visual programming editor that can be extended by suites of externally developed add-ons. Furthermore, it has fundamentally altered the workflows of professionals across multiple industries and fostered an active global community of users.

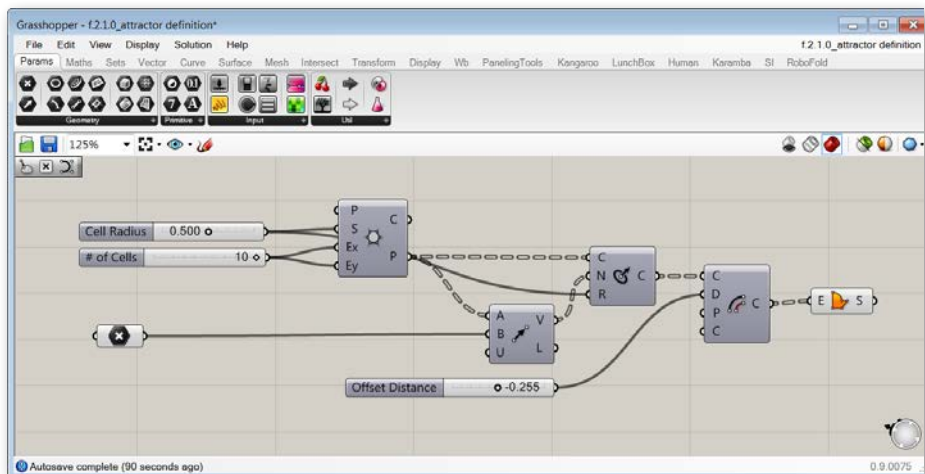
This primer focuses on Foundations, offering the core knowledge you need to dive into regular use of Grasshopper and several on-ramps to how you might go further within your own creative practice. Before diving into the descriptions, diagrams, and examples supplied hereafter, let's discuss what visual programming is, the basics of the Grasshopper interface and terminology, as well as the "live" characteristics of the viewport feedback and user experience.

Visual Programming is a paradigm of computer programming within which the user manipulates logic elements graphically instead of textually. Some of the most well-known textual programming languages such as C#, Visual Basic, Processing – and more close to home for Rhino – Python and Rhinoscript require us to write code that is bound by language-specific syntax. In contrast, visual programming allows us to connect functional blocks into a sequence of actions where the only "syntax" required is that the inputs of the blocks receive the data of the appropriate type, and ideally, that is organized according to the desired result – see the sections on Data Stream Matching and Designing with Data Trees. This characteristic of visual programming avoids the barrier to entry commonly found in trying to learn a new language, even a spoken one, as well as foregrounds the interface, which for designers locates Grasshopper within more familiar territory.



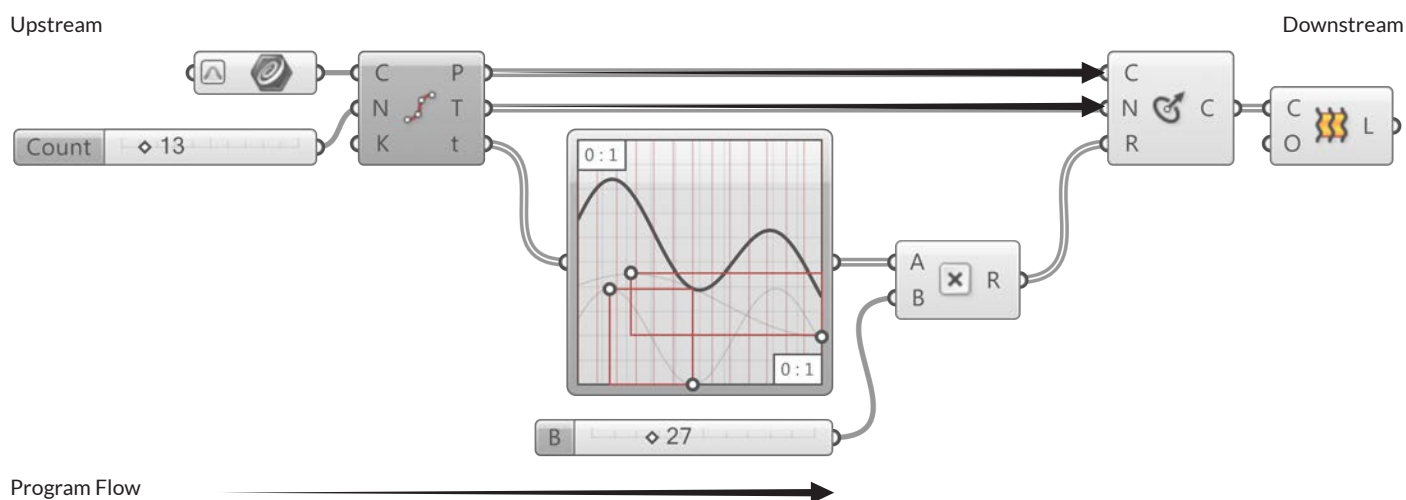
This image show the process for drawing a sine curve in python and in Grasshopper.

To access Grasshopper and its visual programming capabilities, we need to download and install the program from the Grasshopper3D.com website. Once installed, we can open the plug-in by typing “Grasshopper” into the Rhino Command Line. The first time we do so in a new session of Rhino, we will be presented with the Grasshopper loading prompt followed by the Grasshopper editor window. We can now add functional blocks called “components” to the “canvas,” connect them with “wires,” and save the entire “definition” in the .ghx file format.



A Grasshopper definition, made up of components connected with wires on the canvas

Once we’ve started to develop a Grasshopper definition and created “slider” objects within our canvas to control geometry, we may naturally intuit the connection we’ve made between this input object to what we see in Rhino’s viewports. This connection is essentially live – if we adjust the grip on the slider, we will see the consequences in that, within our definition an input somewhere has changed and the program must be solved again to recompute a solution and display the update. To our benefit when getting started with using Grasshopper, the geometry preview we see is a lightweight representation of the solution and it automatically updates. It is important to take note this connection now as when your definitions become more complex, adeptly managing the flow of data, the status of the “solver,” and what is previewed in the Rhino viewport will prevent many unwanted headaches.



As you begin first exploring Grasshopper or further building your skills, you have joined the global Grasshopper community, one that is full of active members from many fields, with diverse experience levels. The forum at Grasshopper3D.com is a useful resource for posing questions, sharing findings, and gaining knowledge. This is a community that we have held dear as we’ve written this primer and watched Grasshopper develop over the years. Welcome!

I.3 Grasshopper in Action



The Cloud
grasshopper3d.com/profile/0etxzm0mi4mb



Random Ornament
<http://patharc.com/>



lamellae lamp
<http://patharc.com/>



n_lamp
grasshopper3d.com/profile/NathanScheidt981



King Rack
grasshopper3d.com/profile/BobThe



Performative Foliage
<http://livecomponents-ny.com/>



Shellstar Pavilion
<http://matsysdesign.com/>



Stalasso Table
<http://patharc.com/>



Masisa Lab
<http://www.gt2p.com>



tesselated floorscape
<http://www.issstudio.com/>



Flat Hex
<http://www.gt2p.com>



cloud[S]cape
www.dfabstudio.com



Exhibition Wall
grasshopper3d.com/profile/leocao



Voironoi Tree
grasshopper3d.com/profile/KirillBrosalin



Parametric Bracelet
grasshopper3d.com/profile/Alex289



Generative Jewelry
<http://cargocollective.com/rafaelmorais>



Plis/Replis
<http://livecomponents-ny.com/>



Chrysalis III
<http://matsysdesign.com/>



bend lamp
<http://patharc.com>



Parametric Bar
<http://urdirlab.blogspot.com/>



Soft Voronoi Shelf
<http://www.gt2p.com/>



clove lamp
<http://patharc.com>



Tarrugao Collection
<http://www.gt2p.com/>



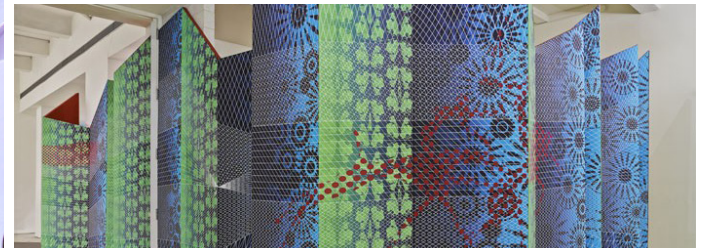
Opening Chronometry
livecomponents-ny.com/



NU:S 3D Printed Shoes
<http://www.arturotedeschi.com/>



minima(maxima)
www.grasshopper3d.com/profile/JohnBrockway



ZigZag
<http://www.issstudio.com/>



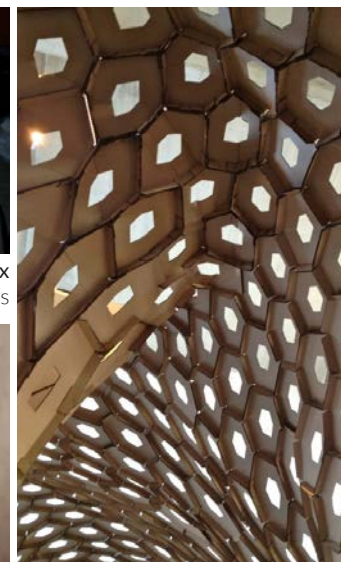
bayou-luminescence
<http://patharc.com/>



Divider
<http://www.arturotedeschi.com/>



Jewelry Lightbox
www.grasshopper3d.com/profile/jovanovicmilos



Catalyst Hexshell
<http://matsysdesign.com/>



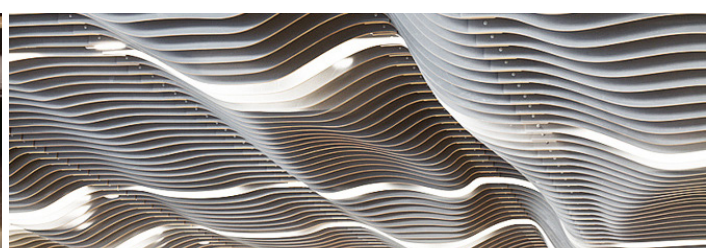
Shhh the hope keeper
<http://www.gt2p.com/>



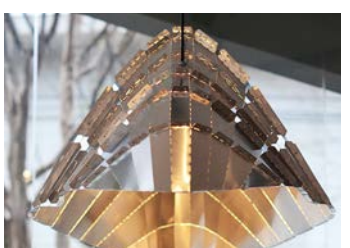
Image credit
www.creditlink.com



Bartop
[grasshopper3d.com/profile/MattHutchinson](http://www.grasshopper3d.com/profile/MattHutchinson)



Co Ceiling
<http://www.gt2p.com/>



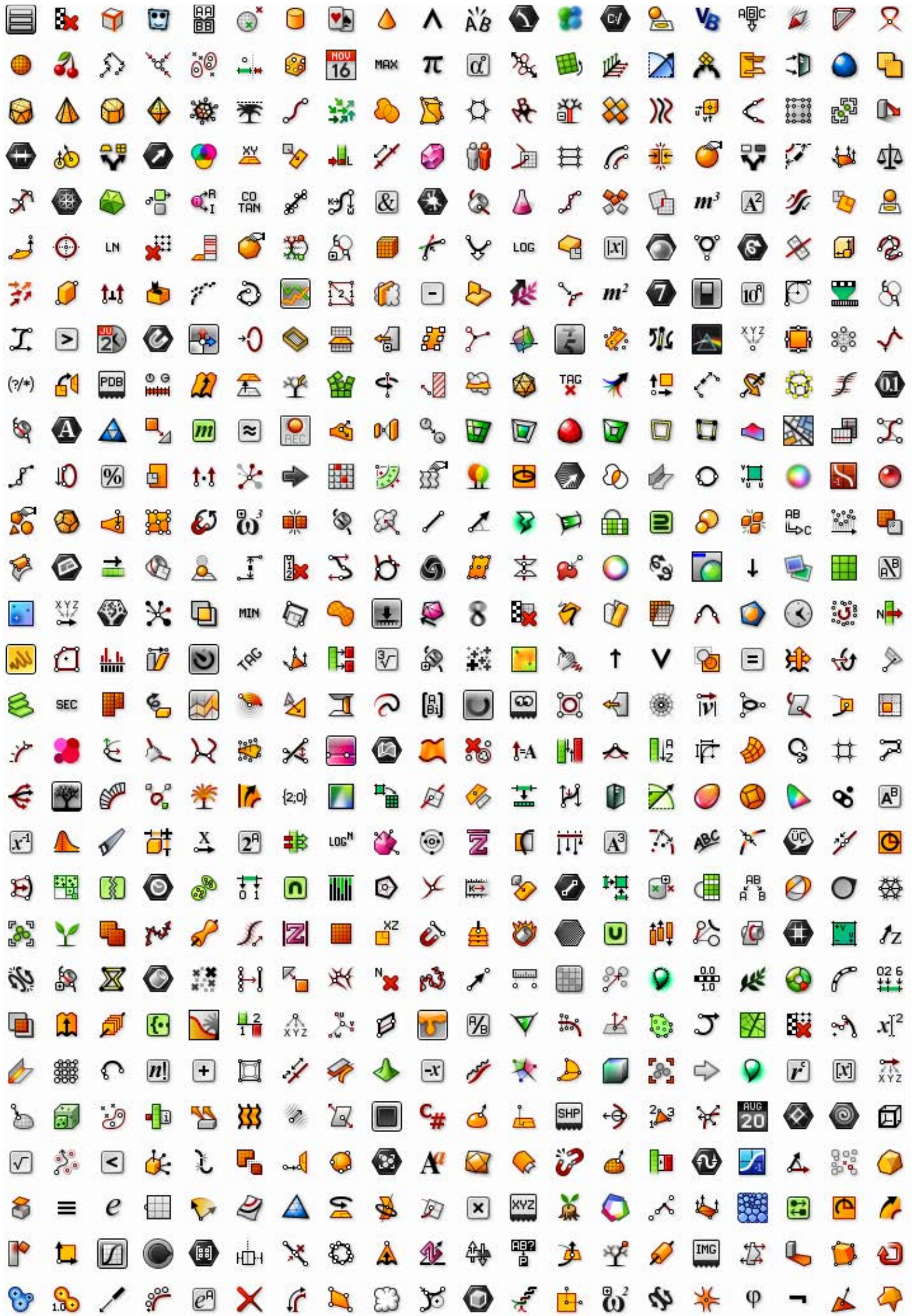
Vilu Light
<http://www.gt2p.com/>

[F.] FOUNDATIONS

A strong foundation is built to last. This volume of the Primer introduces the key concepts and features of parametric modeling with Grasshopper.

F.0 HELLO GRASSHOPPER

Grasshopper is a graphical algorithm editor that is integrated with Rhino3D's modeling tools. You use Grasshopper to design algorithms that then automate tasks in Rhino3D.



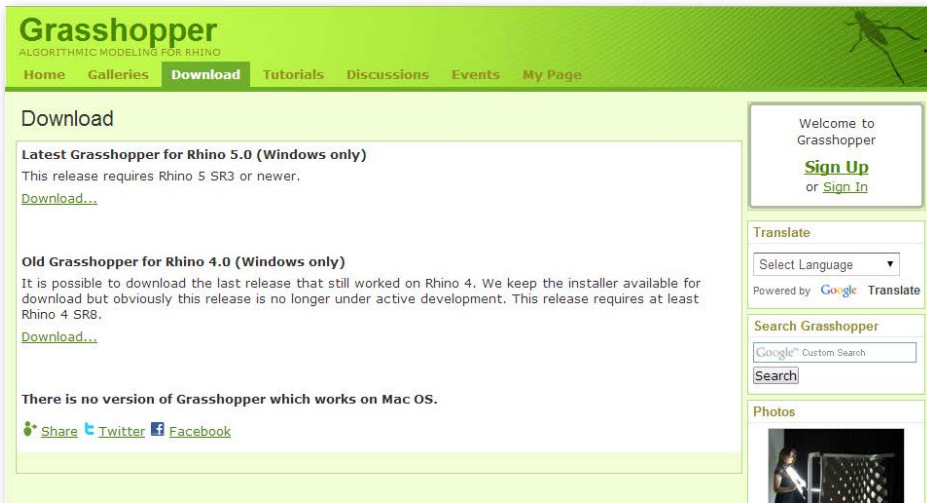
F.0.0

Installing & Launching Grasshopper

The Grasshopper plugin is updated frequently so be sure to update to the latest build. Note that there is currently no version of Grasshopper for Mac.

F.0.0.0 DOWNLOADING

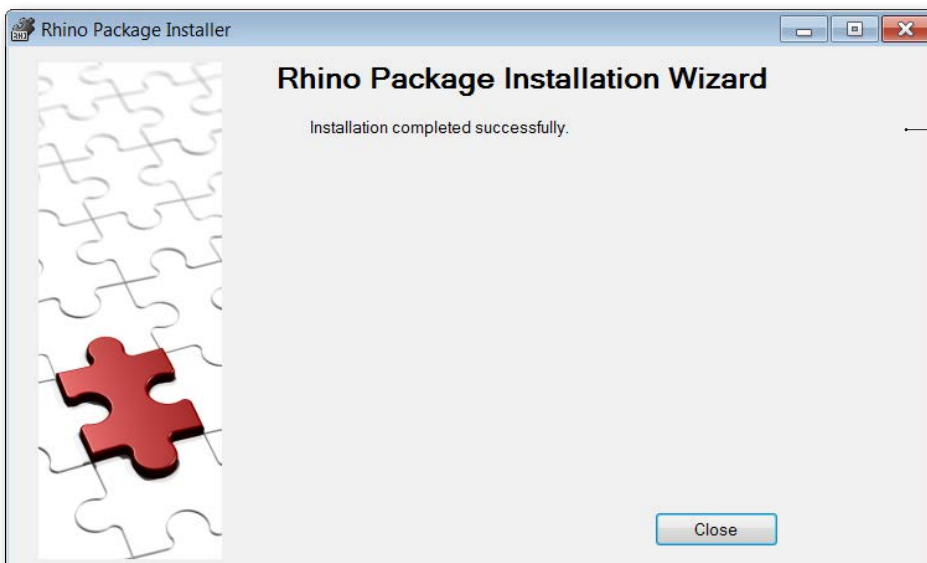
To download the Grasshopper plug-in, visit the Grasshopper web site. Click on the Download tab at the top of the page, and when prompted on the next screen, enter your email address. Now, right click on the download link, and choose Save Target As from the menu. Select a location on your hard drive (note: the file cannot be loaded over a network connection, so the file must be saved locally to your computer's hard drive) and save the executable file to that address.



<http://www.grasshopper3d.com/page/download-1>

F.0.0.1 INSTALLING

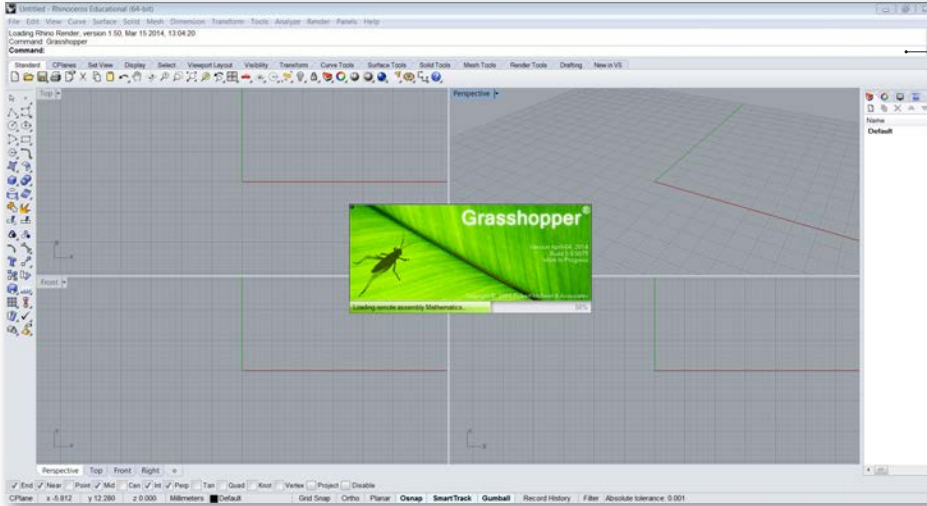
Select Run from the download dialog box follow the installer instructions. (note: you must have Rhino 5 already installed on your computer for the plug-in to install properly).



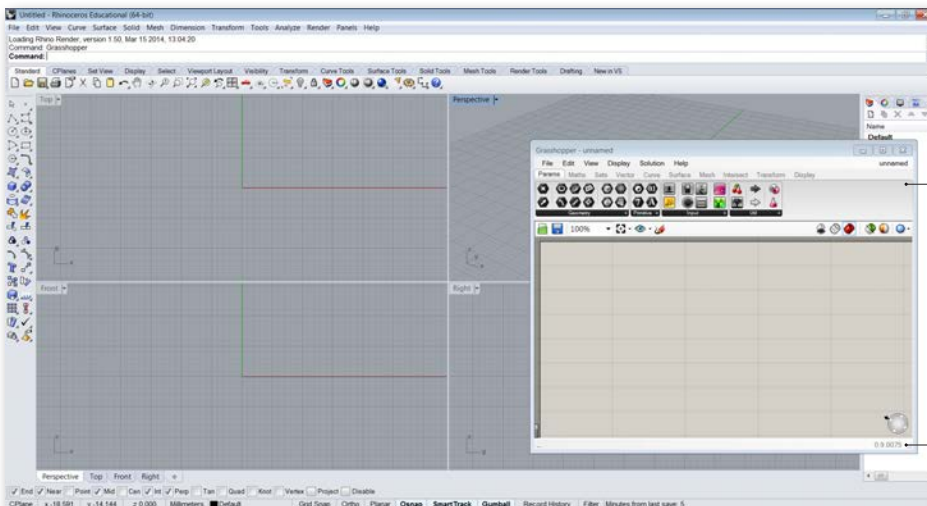
Follow the steps in the Installation wizard

F.0.0.2 LAUNCHING

To Launch Grasshopper, type Grasshopper into the Rhino Command line. When you launch Grasshopper, the first thing you will see is a new window floating in front of Rhino. Within this window you can create node-based programs to automate various types of functionality in Rhino. Best practice is to arrange the windows so that they do not overlap and Grasshopper does not obstruct the Rhino viewports.

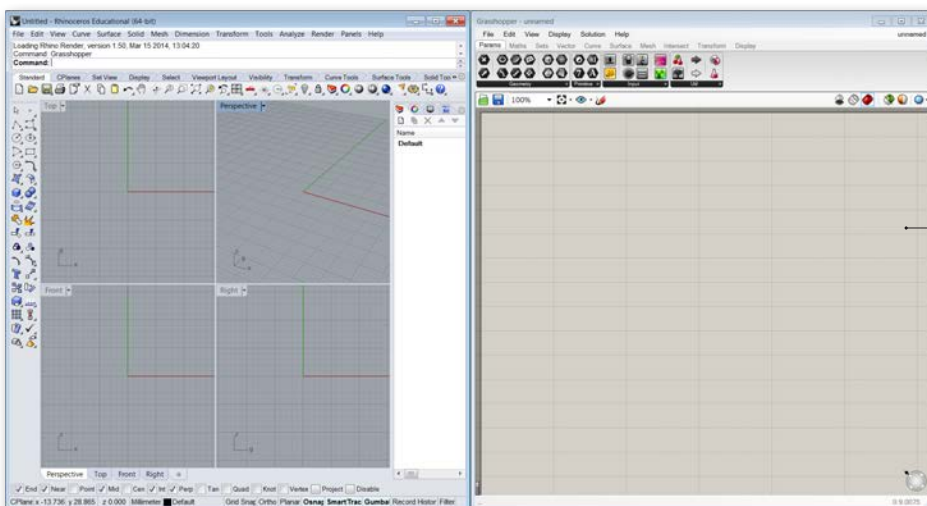


Type "Grasshopper" into the Rhino command line to launch the Grasshopper plugin



The Grasshopper window floats on top of the Rhino viewports

Grasshopper displays the version number at the bottom of the window.



Split the screen so that Grasshopper does not obstruct the Rhino Viewports. You can do this by dragging each window to opposite sides of the screen, or by holding the Windows key and pressing the left or right arrows.

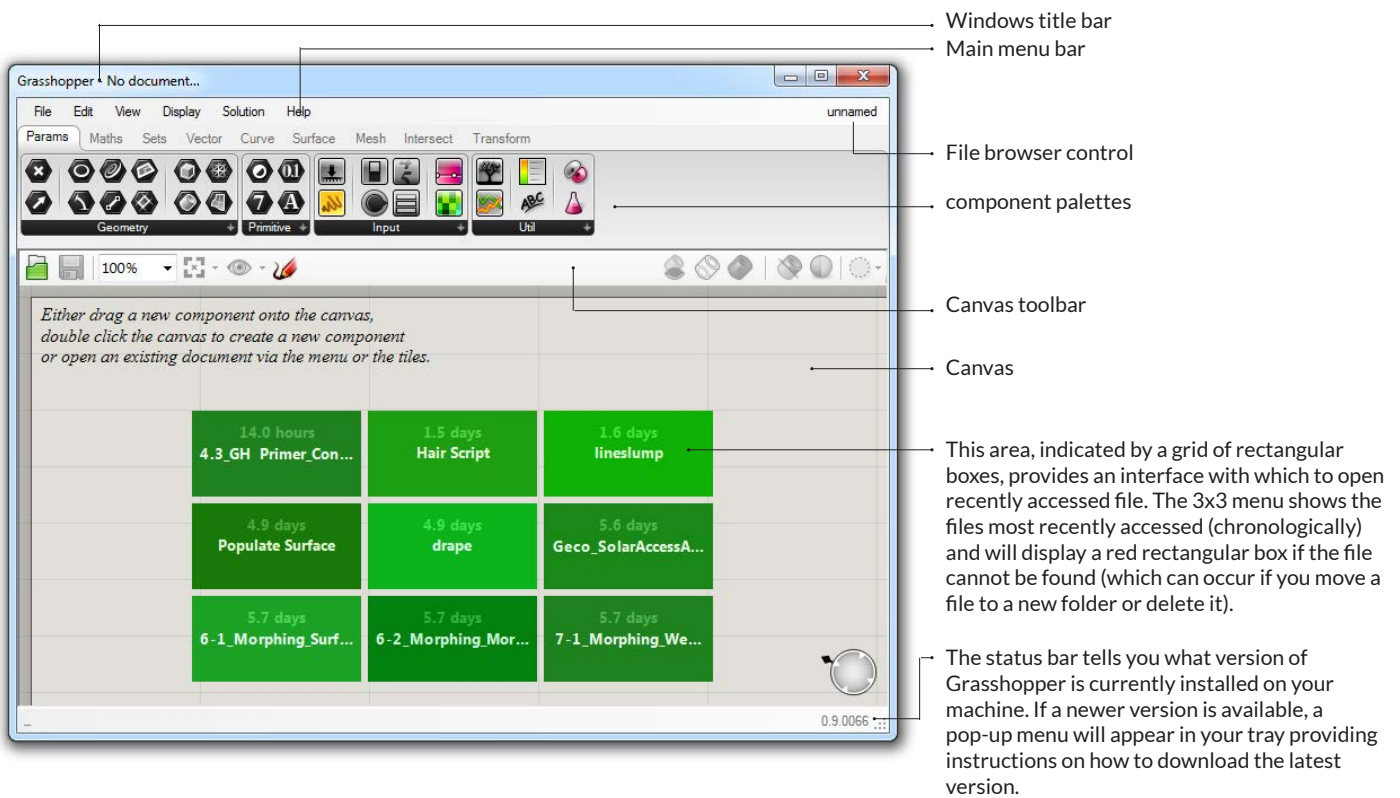
F.0.1

The Grasshopper UI

Grasshopper’s visual “plug-and-play” style gives designers the ability to combine creative problem solving with novel rule systems through the use of a fluid graphical interface.

Let’s start by exploring Grasshopper’s user interface UI. Grasshopper is a visual programming application where you are able to create programs, called definitions or documents, by dragging components onto the main editing window (called the canvas). The outputs to these components are connected to the inputs of subsequent components — creating a graph of information which can be read from left to right. Let’s get started with the basics.

Assuming you’ve already installed the Grasshopper plugin (see F.0.0), type the word “Grasshopper” in the Rhino command prompt to display the Grasshopper Editor. The Grasshopper interface contains a number of elements, most of which will be very familiar to Rhino users. Let’s look at a few features of the interface.



F.0.1.0 THE WINDOWS TITLE BAR

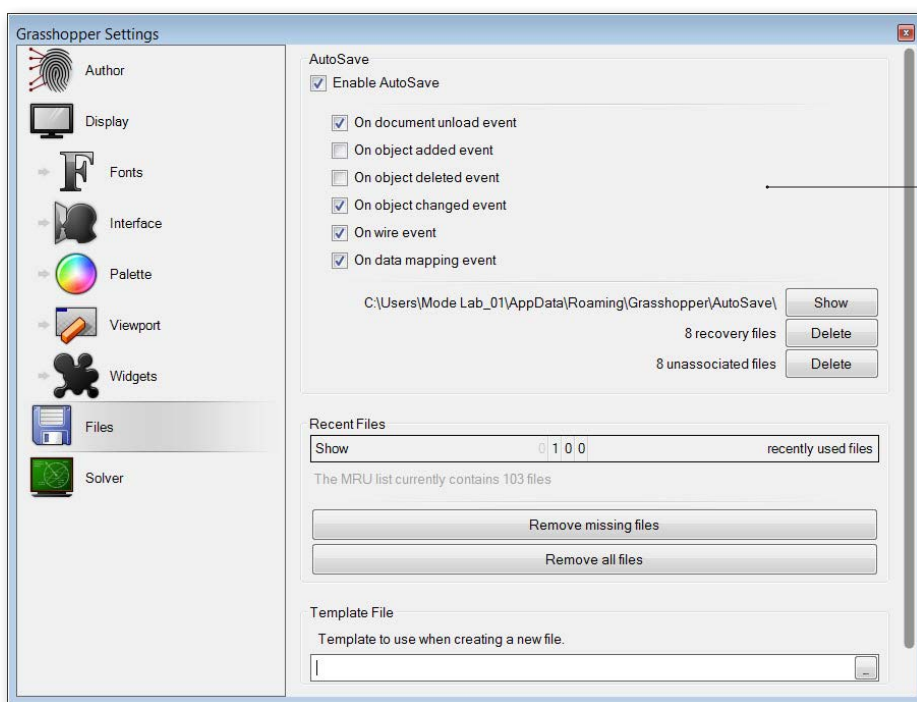
The Editor Window title bar behaves differently from most other dialogs in Microsoft Windows. If the window is not minimized or maximized, double clicking the title bar will collapse the dialog into a minimized bar on your screen. This is a great way to switch between the plug-in and Rhino because it minimizes the Editor without moving it to the bottom of the screen or behind other windows. Note that if you close the Editor, the Grasshopper geometry preview in the Rhino viewport will disappear, but the file won't actually be closed. The next time you run the “Grasshopper” command in the Rhino dialog box, the window will come back in the same state with the same files loaded. This is because once it is launched from the command prompt, your session of Grasshopper stays active until that instance of Rhino is closed.

F.0.1.1 MAIN MENU BAR

The title bar is similar to typical Windows menus, except for the file browser control on the right (see next section). The File menu provides typical functions (eg. New File, Open, Save, etc.) in addition to a few utility tools which let you export images of your current Grasshopper document (see Export Quick Image and Export Hi-Res Image). You can control different aspects of the user interface using the View and Display menus, while the Solution menu lets you manage different attributes about how the solver computes the graph solution.

It is worth noting that many application settings can be controlled through the Preferences dialog box found under the File menu. The Author section allows you to set personal metadata which will be stored with each Grasshopper document while the Display section gives you fine grain control over the look and feel of the interface. The Files section lets you specify things like how often and where to store automatically saved file (in case the application is inadvertently closed or crashes). Finally, the Solver section lets you manage core and third-party plugins which can extend functionality.

Note: Be careful when using shortcuts since they are handled by the active window which could either be Rhino, the Grasshopper canvas or any other window inside Rhino. It is quite easy to use a shortcut command, only to realize that you had the wrong active window selected and accidentally invoked the wrong command.



The Preferences dialog allows you to set many of Grasshopper's application settings.

F.0.1.2 FILE BROWSER CONTROL

The File Browser allows you to quickly switch between different loaded files by selecting them through this drop-down list. Accessing your open files through the File Browser drop-down list enables you to quickly copy and paste items from open definitions. Just click on the active file name in the browser control window and a cascading list of all open files will be displayed (along with a small thumbnail picture of each open definition) for easy access. You can also hit Alt+Tab to quickly switch between any open Grasshopper documents.

Of course, you can go through the standard Open File dialog to load any Grasshopper document, although you can also drag and drop any Grasshopper file onto the canvas to load a particular definition.

Grasshopper is a plug-in that works "on-top" of Rhino and as such has its own file types. The default file type is a binary data file, saved with an extension of .gh. The other file type is known as a Grasshopper XML file, which uses the extension .ghx. The XML (Extensible Markup Language) file type uses tags to define objects and object attributes (much like an .HTML document) but uses custom tags to define objects and the data within each object. Because XML files are formatted as text documents, you could open up any Grasshopper XML file in a text editor like NotePad to see the coding that is going on behind the scenes.

Grasshopper has several different methods by which it can open a file, and you will need to specify which option you would like to use when using this method.

Open File: As the name suggests, this file option will simply open any definition that you drag and drop onto the canvas.

Insert File: You can use this option to insert an existing file into the current document as loose components.

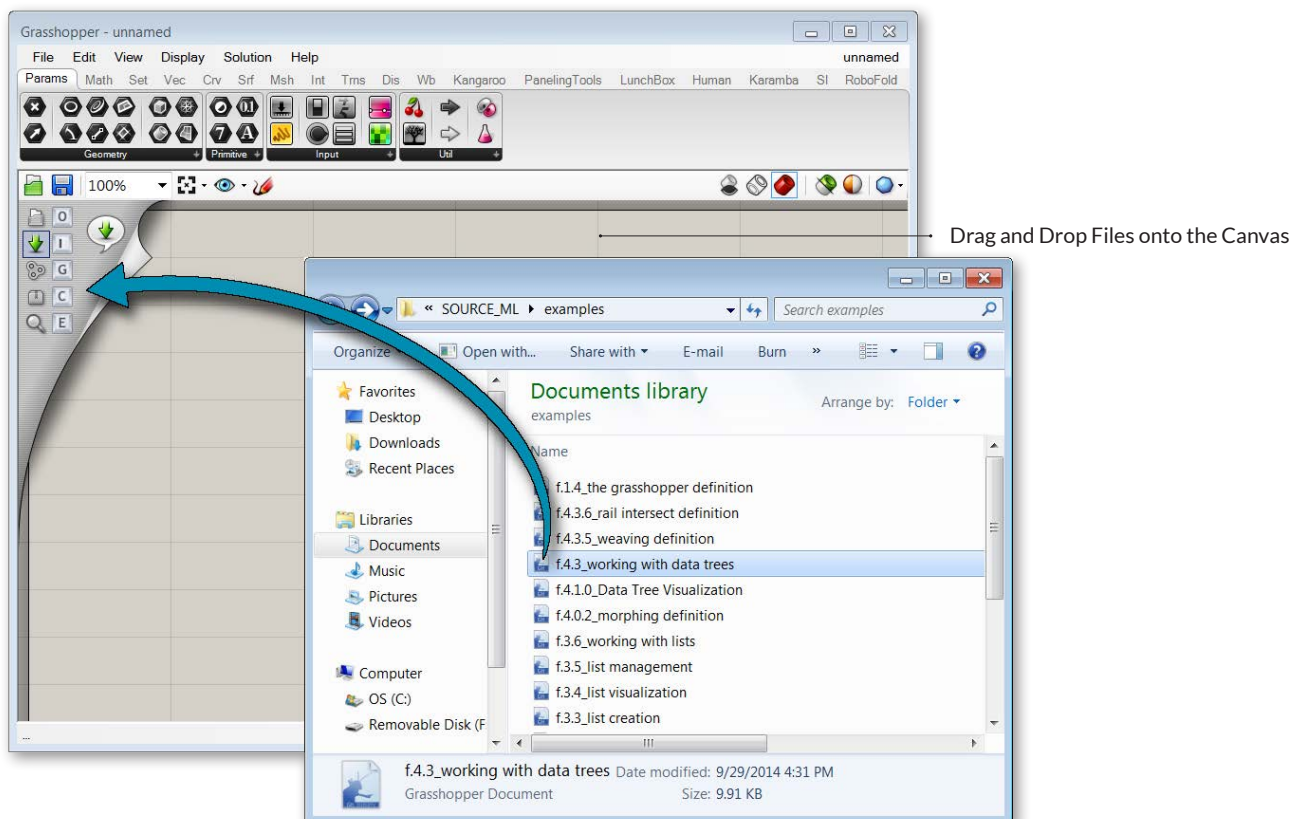
Group File: This method will insert a file into an existing document, but will group all of the objects together.

Cluster File: Similar to the group function, this method will insert a file into an existing document, but will create a cluster object for the entire group of objects.

Examine File: Allows you to open a file in a locked state, meaning you can look around a particular file but you can't make any changes to the definition.

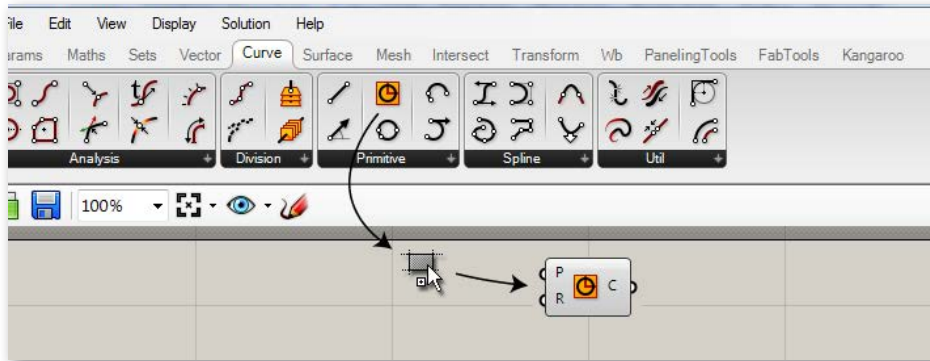
Grasshopper also has an Autosave feature which will be triggered periodically based on specific user actions. A list of Autosave preferences can be found under the File menu on the Main Menu Bar. When the active instance of Rhino is closed, a pop-up dialog box will appear asking whether or not you want to save any Grasshopper files that were open when Rhino was shut down.

Note: Autosave only works if the file has already been saved at least once



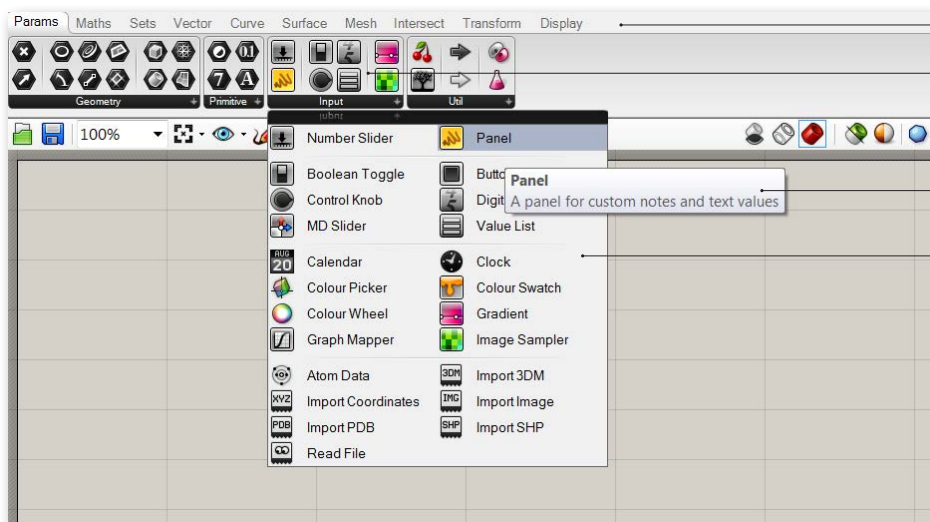
F.0.1.3 COMPONENT PALETTES

This area organizes components into categories and sub-categories. Categories are displayed as tabs, and subcategories are displayed as drop-down panels. All components belong to a certain category. These categories have been labeled to help you find the specific component that you are looking for (e.g. “Params” for all primitive data types or “Curves” for all curve related tools). To add a component to the canvas, you can either click on the objects in the drop-down menu or you can drag the component directly from the menu onto the canvas.



Drag + Drop a component from the palette to add a component to the canvas.

Since there can be many more components in each sub-category than will fit into the palette, a limited number of icons are displayed on each panel. The height of the component palette and the width of the Grasshopper window can be adjusted to display more or fewer components per sub-category. To see a menu of all of the components in a given sub-category, simply click on the black bar at the bottom of each sub-category panel. This will open a dropdown menu which provides access to all components in that sub-category.



Category tab - click the black bar to open the sub-category panel menu

Sub-category panel

Hover your mouse over a component for a short description

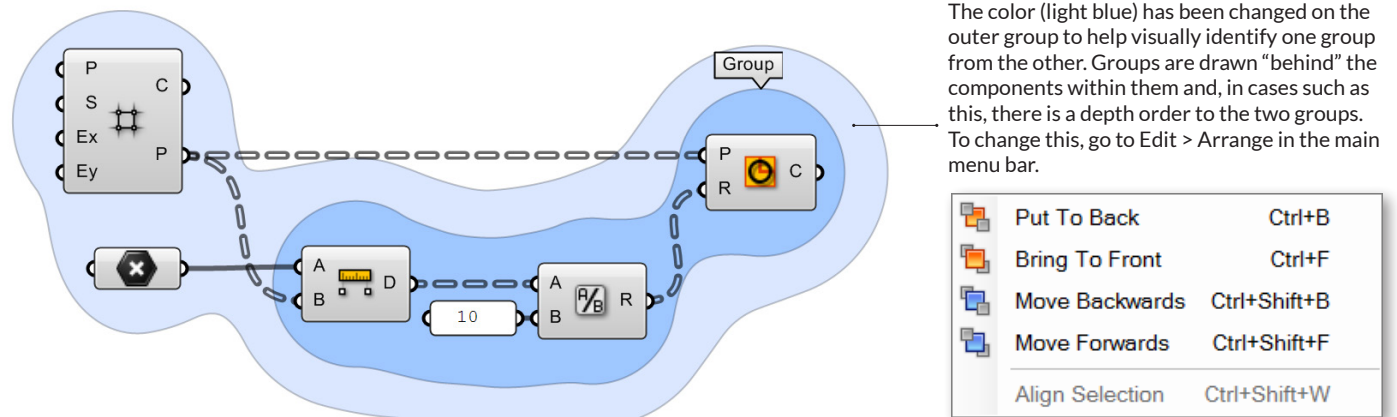
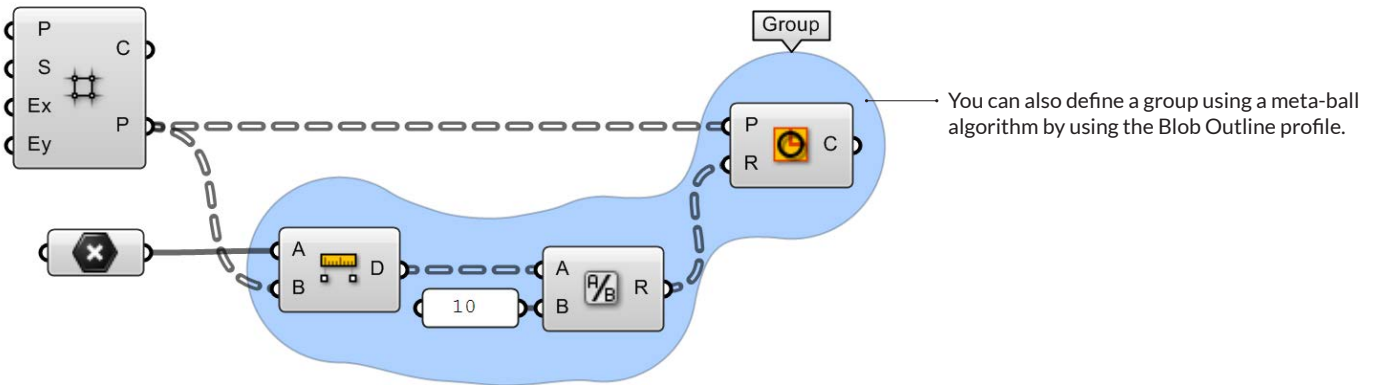
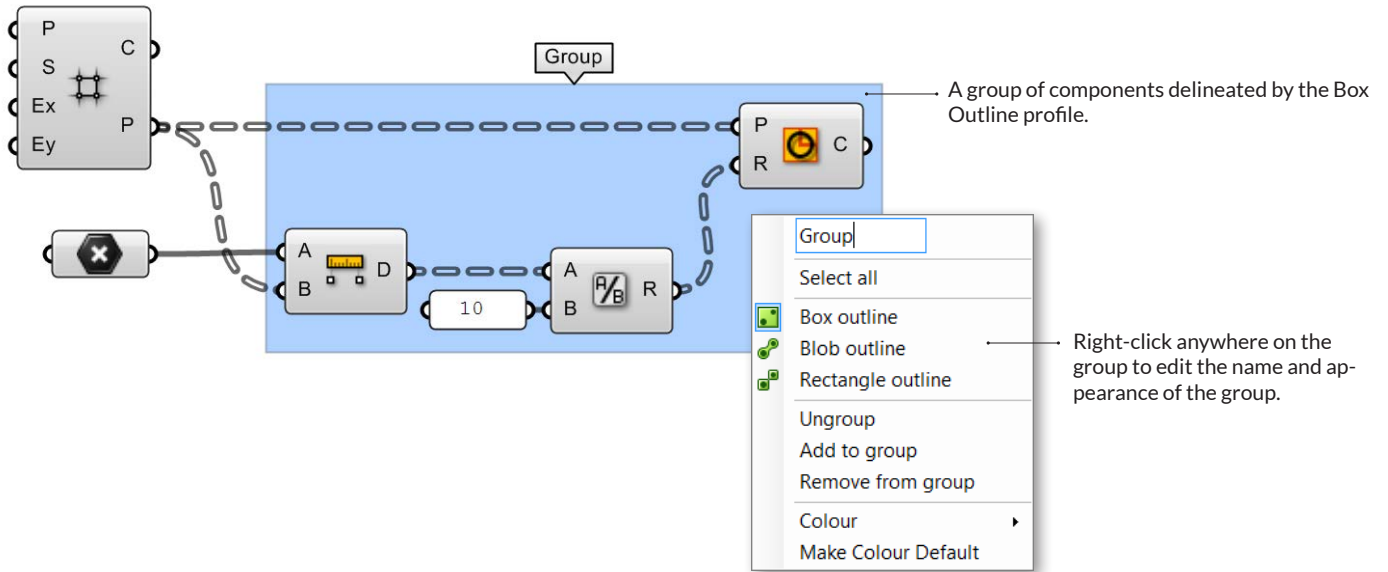
Drop-down menu

F.0.1.4 THE CANVAS

The canvas is the primary workspace for creating Grasshopper definitions. It is here where you interact with the elements of your visual program. You can start working in the canvas by placing components and connecting wires.

F.0.1.5 GROUPING

Grouping components together on the canvas can be especially useful for readability and comprehensibility. Grouping allows you the ability to quickly select and move multiple components around the canvas. You can create a group by typing Ctrl+G with the desired components selected. An alternate method can be found by using the "Group Selection" button under the Edit Menu on the Main Menu Bar. Custom parameters for group color, transparency, name, and outline type can be defined by right-clicking on any group object.

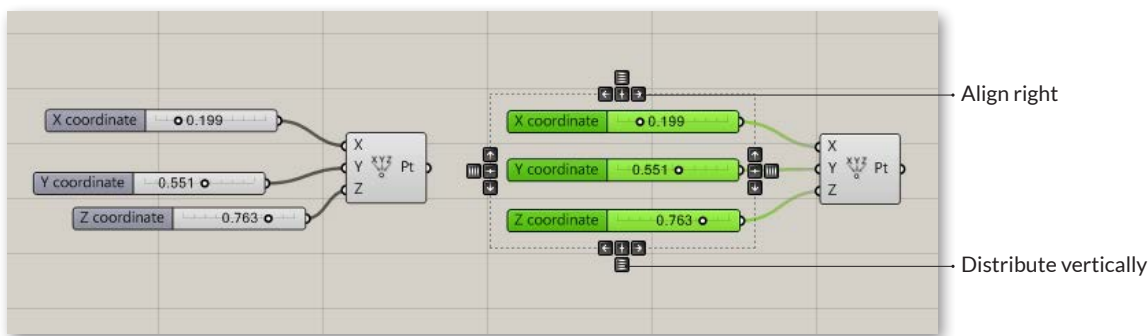


F.0.1.6 WIDGETS

There are a few widgets that are available in Grasshopper that can help you perform useful actions. You can toggle any of these widgets on/off under the Display menu of the Main Menu bar. Below we'll look at a few of the most frequently used widgets.

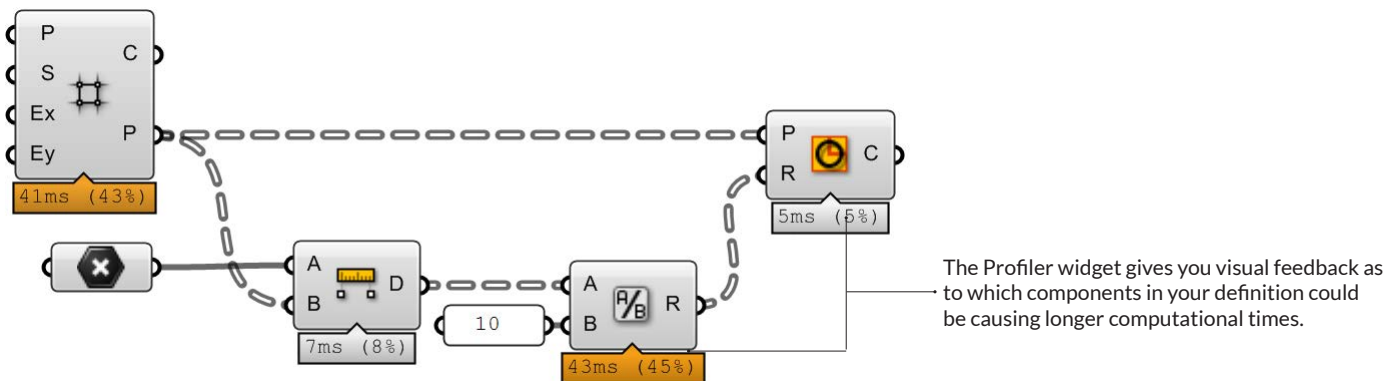
The Align Widget

One useful UI widget which can help you keep your canvas clean is the Align widget. You can access the Align widget by selecting multiple components at the same time and clicking on one of the options found in the dashed outline that surrounds your selected components. You can align left, vertical center, right, or top, horizontal center, bottom, or distribute components equally through this interface. When first starting out, you may find that these tools sometimes get in the way (it is possible to make the mistake of collapsing several components on top of each other). However, with a little practice these tools can be invaluable as you begin to structure graphs which are readable and comprehensible.



The Profiler Widget

The profiler lists worst-case runtimes for parameters and components, allowing you to track down bottlenecks in networks and to compare different components in terms of performance. Note that this widget is turned off by default.



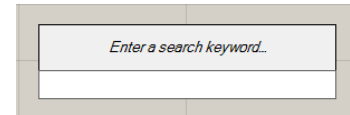
The Markov Widget

This widget uses Markov chains to 'predict' which component you may want to use next based on your behavior in the past. A Markov chain is a process that consists of a finite number of states (or levels) and some known probabilities. It can take some time for this widget to become accustomed to a particular user, but over time you should begin to notice that this widget will begin to suggest components that you may want to use next.

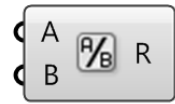
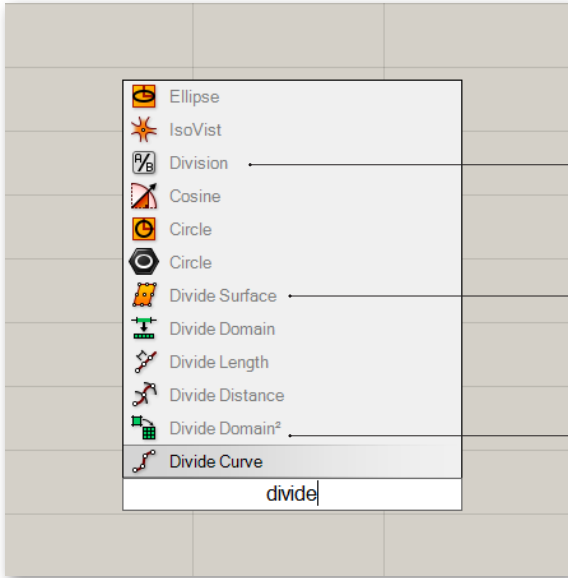
The Markov Widget can suggest up to five possible components depending on your recent activity. You can right-click on the Markov widget (the default location is the bottom left-hand corner of the canvas) to dock it into one of the other corners of the canvas or to hide it completely.

F.0.1.7 USING THE SEARCH FEATURE

Although a lot of thought has gone into the placement of each component on the component panel to make it intuitive for new users, people sometimes find it difficult to locate a specific component that might be buried deep inside one of the category panels. Fortunately, you can also find components by name, by double-clicking on any empty space on the canvas. This will invoke a pop-up search box. Simply type in the name of the component you are looking for and you will see a list of parameters or components that match your request.



Double-click anywhere on the canvas to invoke a key word search for a particular component found in the Component Panels.

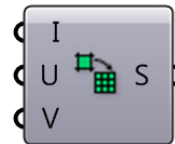


A search for "divide" lists a variety of components.

Division operator component



Divide Surface component

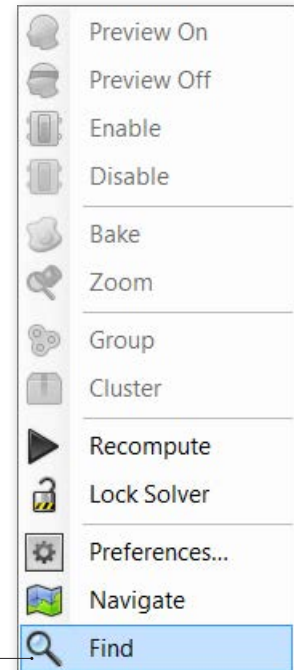


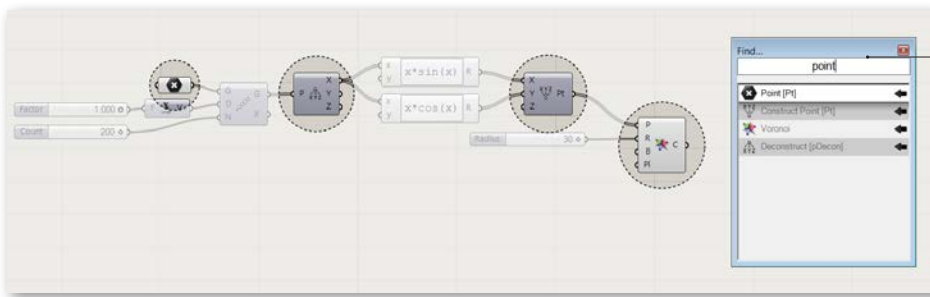
Divide Domain² component

F.0.1.8 THE FIND FEATURE

There are literally hundreds (if not thousands) of Grasshopper components which are available to you and it can be daunting as a beginner to know where to look to find a specific component within the Component Palettes. The quick solution is to double-click anywhere on the canvas to launch a search query for the component you are looking for. However, what if we need to find a particular component already placed on our canvas? No need to worry. By right-clicking anywhere on the canvas or pressing the F3 key, you can invoke the Find feature. Start by typing in the name of the component that you are looking for.

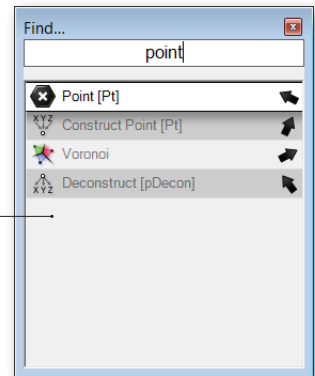
The Find feature employs the use of some very sophisticated algorithms which search not only for any instances of a component's name within a definition (a component's name is the title of the component found under the Component Panel which we as users cannot change), but also any unique signatures which we may have designated for a particular component (known as nicknames). The Find feature can also search for any component type on the canvas or search through text panel, scribble, and group content. Once the Find feature has found a match, it will automatically grey out the rest of the definition and draw a dashed line around the highlighted component. If multiple matches are found, a list of components matching your search query will be displayed in the Find dialog box and hovering over an item in the list will turn that particular component on the canvas green.





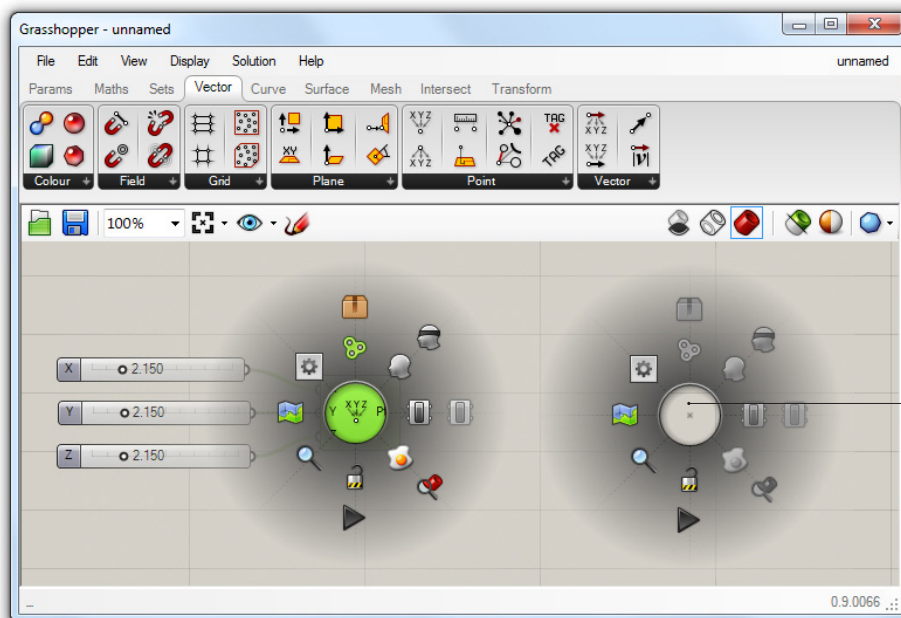
The Find feature can be quite helpful to locate a particular component on the canvas. Right-click anywhere on the canvas to launch the Find dialog box.

A small arrow will also be displayed next to each item in the list which points to its corresponding component on the canvas. Try moving the Find dialog box around on the canvas and watch the arrows rotate in to keep track of their components. Clicking on the Find result will try to place the component (on the canvas) next to the Find dialog box.



F.0.1.9 USING THE RADIAL MENU

As you become more proficient in using the Grasshopper interface, you'll begin to find ways to expedite your workflow. Using shortcuts is one way to do this, however there is another feature which can allow you to quickly access a number of useful tools – the radial UI menu. You can invoke the radial menu by hitting the space bar (while your mouse is over the canvas or a component) or by clicking your middle mouse button. The radial menu will enable different tools depending on whether you invoke the menu by clicking directly on top of a component, or just anywhere on the canvas. In the image below, you see the radial menu has more features available when clicking on top of a selected component versus just clicking anywhere else on the canvas. This menu can dramatically increase the speed at which you create Grasshopper documents.



The Radial UI menu allows you to quickly access frequently used menu items.

F.0.1.10 THE CANVAS TOOLBAR

The canvas toolbar provides quick access to a number of frequently used Grasshopper features. All of the tools are available through the menu as well, and you can hide the toolbar if you like. The toolbar can be re-enabled from the View tab on the Main Menu Bar.

Save File: A shortcut to save the current Grasshopper File.

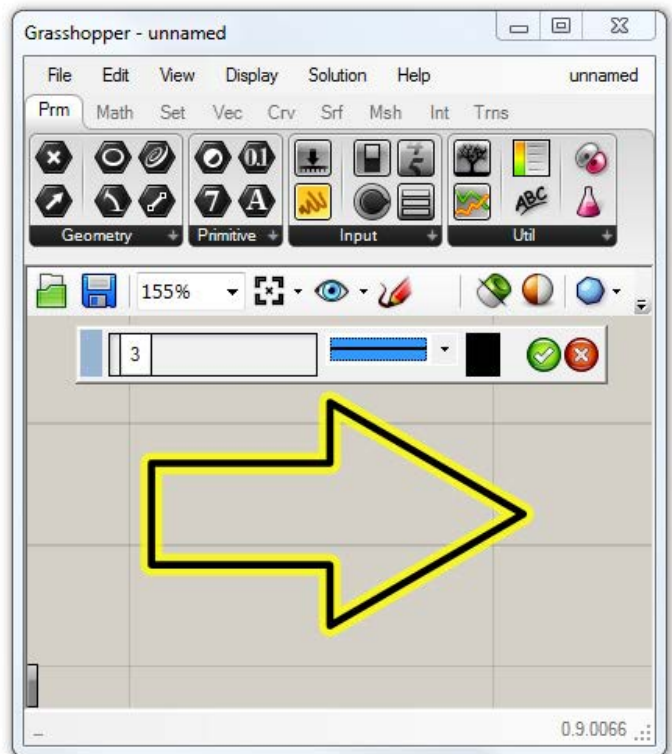
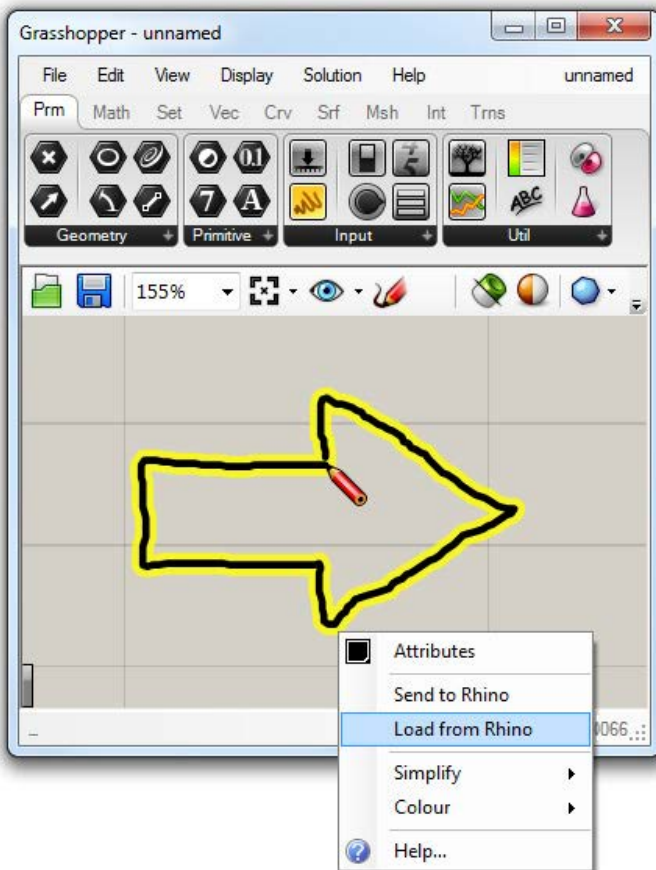
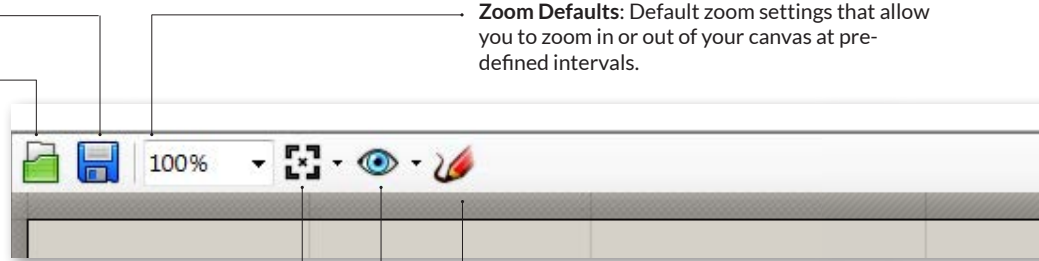
Open File: A shortcut to open a Grasshopper File.

Zoom Defaults: Default zoom settings that allow you to zoom in or out of your canvas at pre-defined intervals.

Zoom Extents: Zoom to the extents of your definition. Click on the arrow next to the Zoom Extents icon to select one of the sub-menu items to zoom to a particular region within your definition.

Named Views: This feature exposes a menu allowing you to store or recall any view area in your definition.

The Sketch Tool: The sketch tool works similarly to the pencil tool set found in Adobe Photoshop with a few added features.



The sketch tool allows changes to the line weight, line type, and color. By right-clicking on the selected sketch object you can choose to simplify your line to create a smoother effect. Right-click on your sketch object and select "Load from Rhino". When prompted, select any 2D shape in your Rhino scene. Once you have selected your referenced shape, hit Enter, and your previous sketch line will be reconfigured to your Rhino reference shape.

Note: Your sketch object may have moved from its original location once you have loaded a shape from Rhino. Grasshopper places your sketch object relative to the origin of the canvas (upper left hand corner) and the world xy plane origin in Rhino.

Preview Settings: If a Grasshopper component generates some form of geometry, then a preview of this geometry will be visible in the viewport by default. You can disable the preview on a per-object basis by right-clicking each component and de-activating the preview feature, or globally change the preview state by using one of these three buttons.



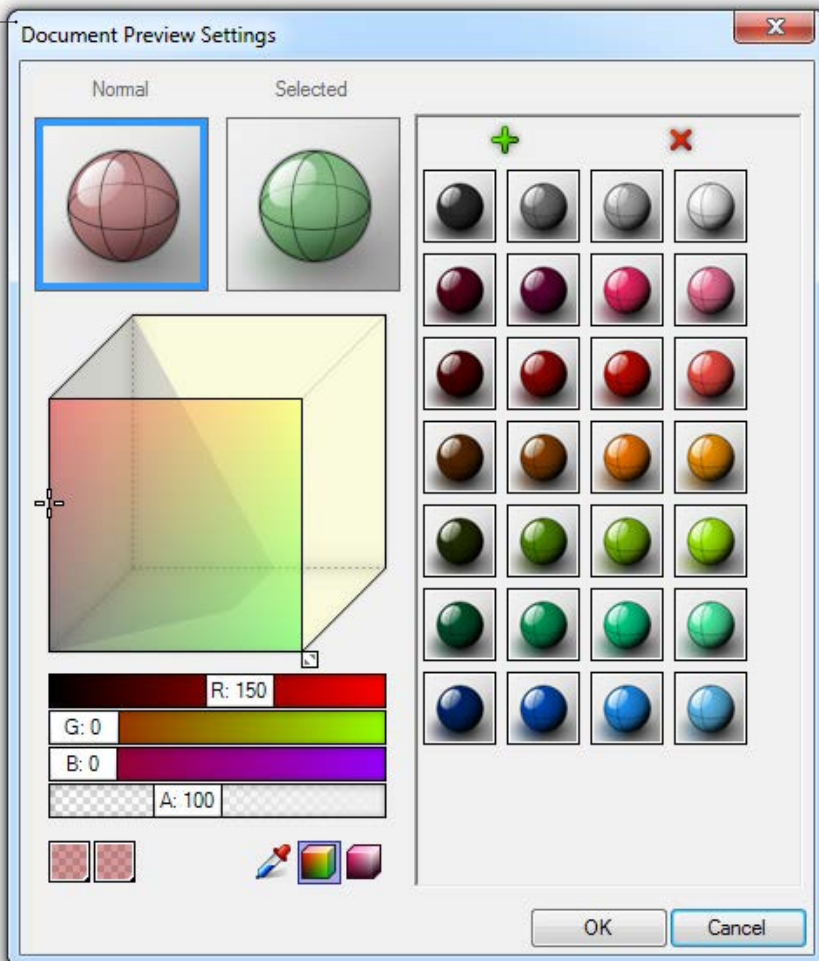
Wire-frame preview
 Turn off preview
 Shaded preview (default)

Preview Mesh Quality: For optimization purposes, these settings allow you to control the quality of the mesh/surface display of the geometry rendered in Rhino. Higher quality settings will cause longer calculation times, whereas lower settings will display less accurate preview geometry.

It should be noted that the geometry still maintains a high-degree of resolution when baked into the Rhino document - these settings merely effect the display performance and quality.

Preview Selected Objects: With this button toggled, Grasshopper will only display geometry that is part of selected components, even if those components have a preview=off state.

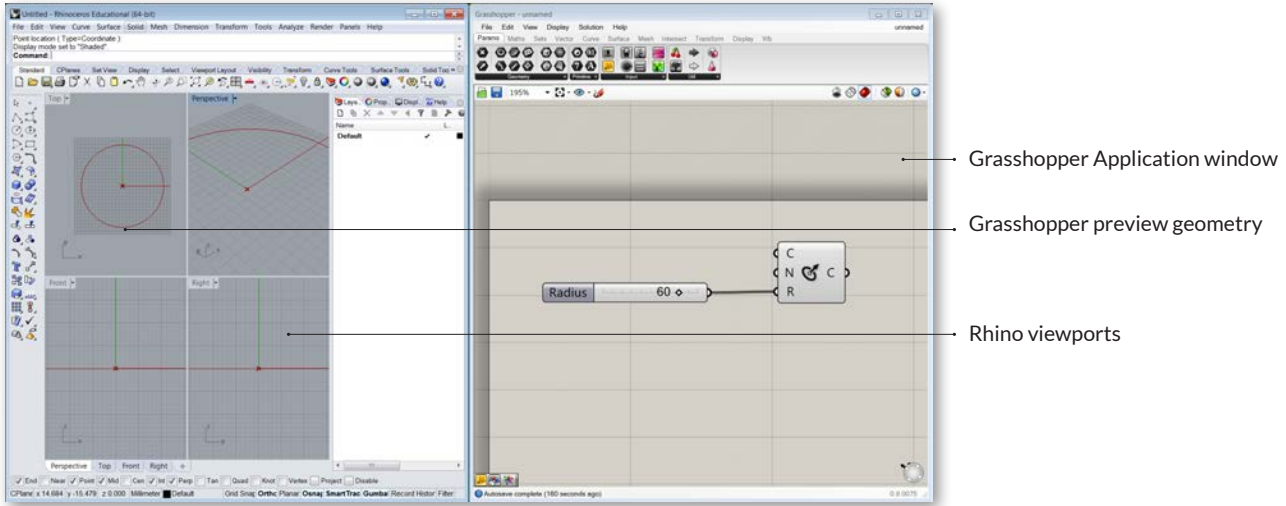
Document Preview Settings: Grasshopper has a default color scheme for selected (semi-transparent green) and unselected (semi-transparent red) geometry. It is possible to override this color scheme with the Document Preview Settings dialog.



F.0.2

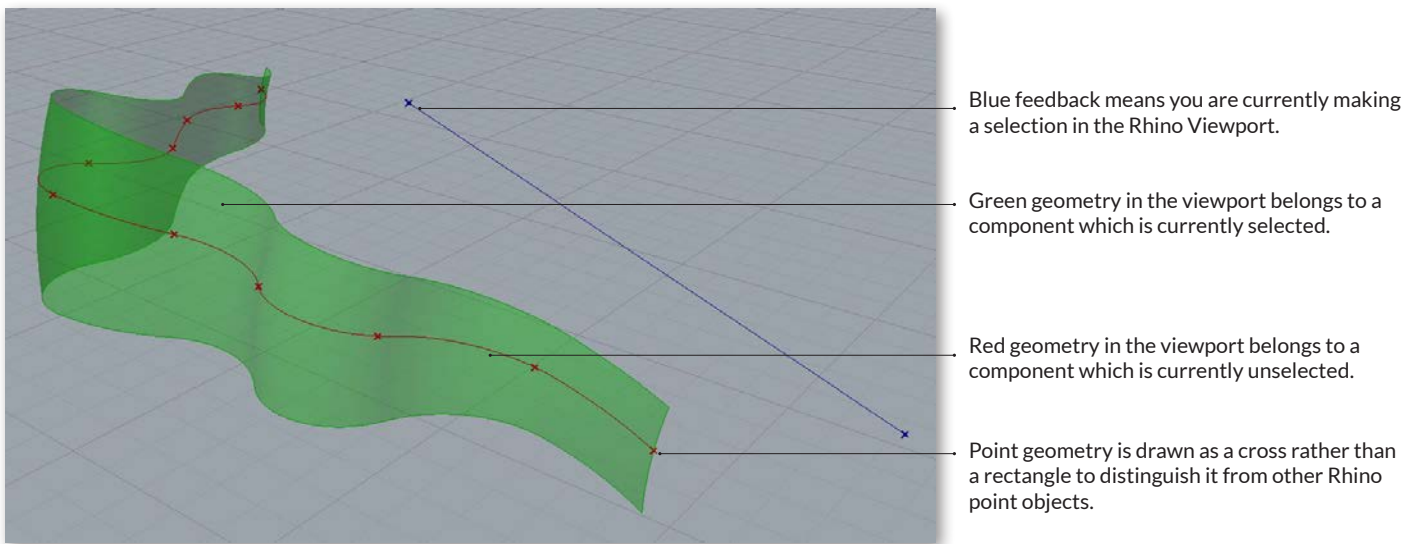
Talking to Rhino

Unlike a Rhino document, a Grasshopper definition does not contain any actual objects or geometry. Instead, a Grasshopper definition represents a set of rules & instructions for how Rhino can automate tasks.



F.0.2.0 VIEWPORT FEEDBACK

All geometry that is generated using the various Grasshopper components will show up (by default) in the Rhino viewport. This preview is just an Open GL approximation of the actual geometry, and as such you will not be able to select the geometry in the Rhino viewport (you must first bake it into the scene). You can turn the geometry preview on/off by right-clicking on a component and selecting the Preview toggle. The geometry in the viewport is color coded to provide visual feedback. The image below outlines the default color scheme.



Blue feedback means you are currently making a selection in the Rhino Viewport.

Green geometry in the viewport belongs to a component which is currently selected.

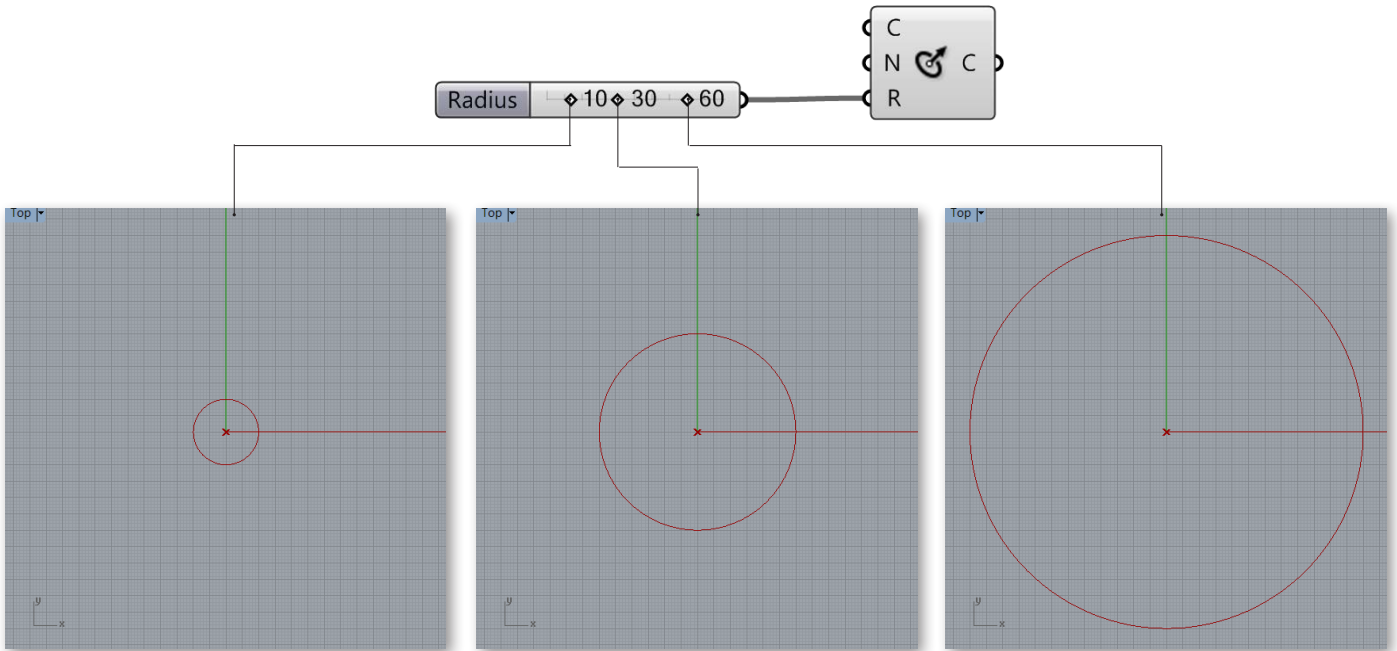
Red geometry in the viewport belongs to a component which is currently unselected.

Point geometry is drawn as a cross rather than a rectangle to distinguish it from other Rhino point objects.

Note: This is the default color scheme, which can be modified using the Document Preview Settings tool on the canvas toolbar.

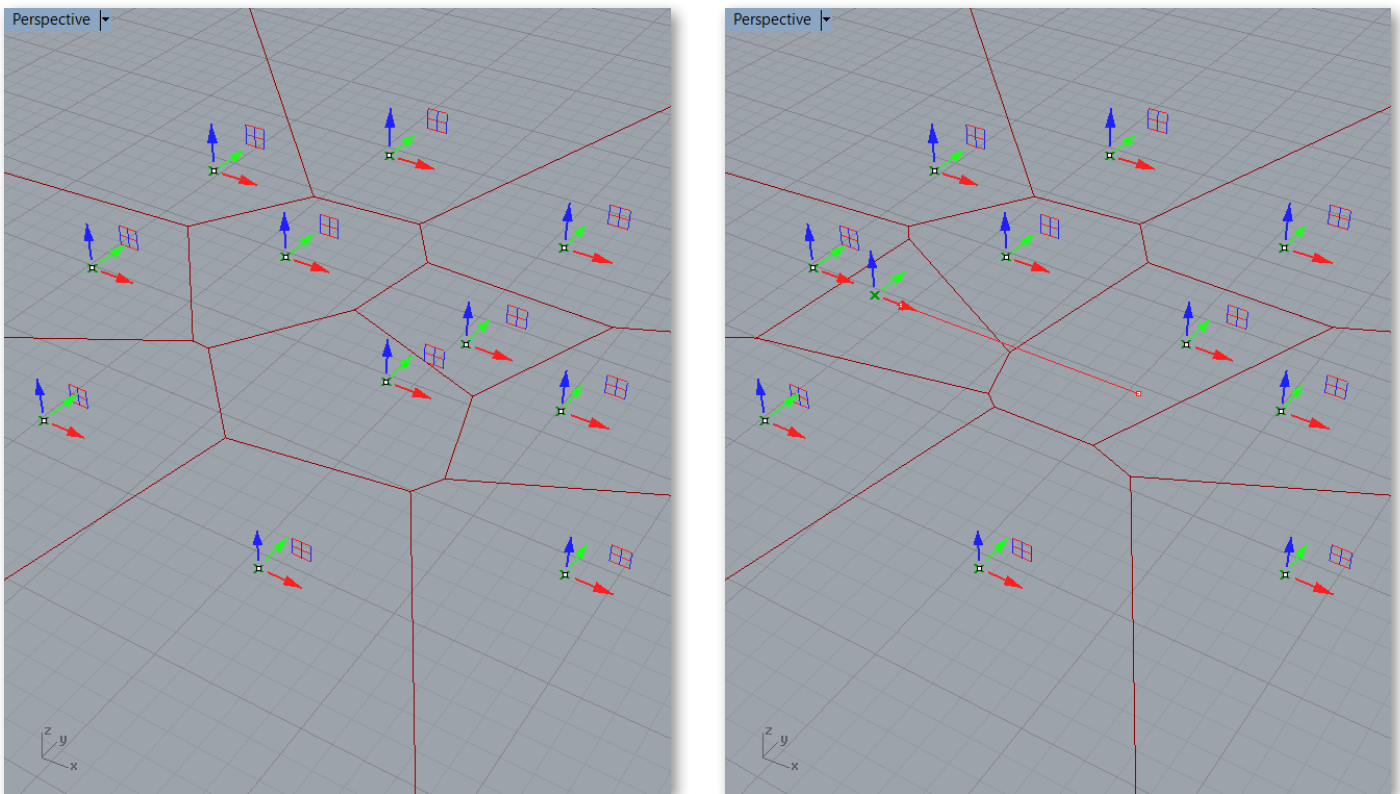
F.0.2.1 LIVE WIRES

Grasshopper is a dynamic environment. Changes that are made are live and their preview display is updated in the Rhino viewport.



F.0.2.2 GUMBALL WIDGET

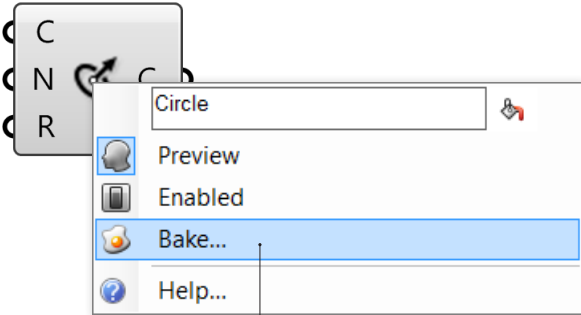
When storing geometry as internalized in a Grasshopper parameter, the gumball allows you to interface with that geometry in the Rhino viewport. This update is live and updates will occur as you manipulate the gumball. In contrast, geometry referenced from Rhino directly will continue to exist in the Rhino document and updates from Grasshopper will happen only after any changes occur (as opposed to during).



F.0.2.3 BAKING GEOMETRY

In order to work with (select, edit, transform, etc.) geometry in Rhino that was created in Grasshopper, you must “bake” it. Baking instantiates new geometry into the Rhino document based on the current state of the Grasshopper graph. It will no longer be responsive to the changes in your definition.

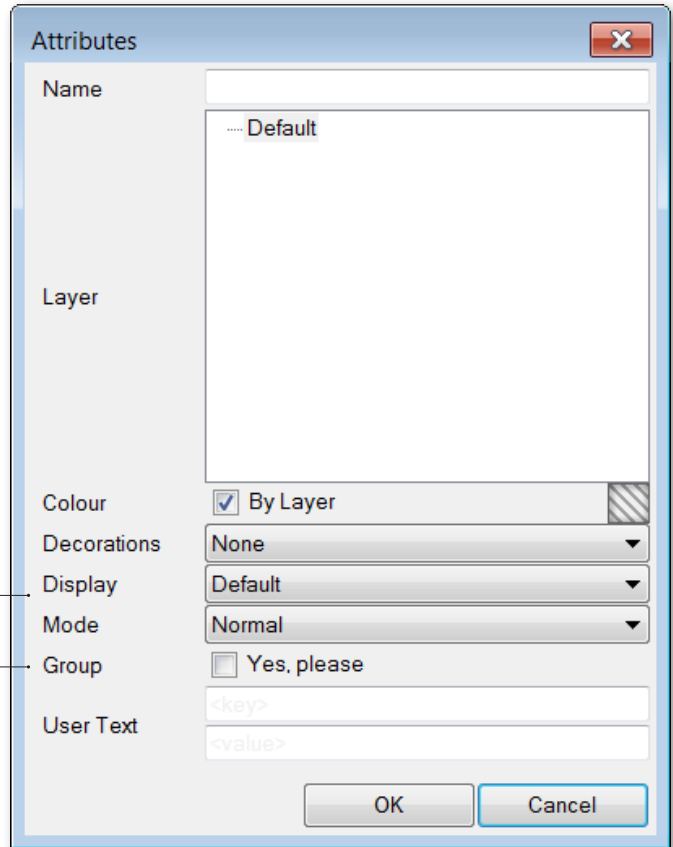
Press the INSERT key to bake all selected components into the currently active Rhino layer



Bake by right-clicking a component and selecting Bake.

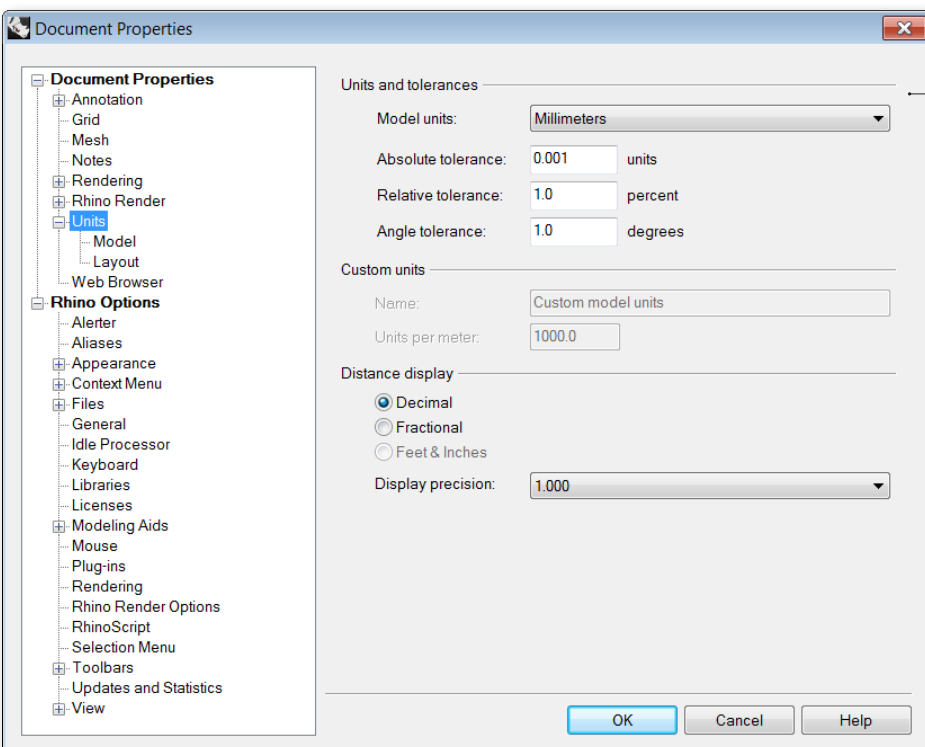
A dialog will appear that allows you to select onto which Rhino layer the geometry will bake.

Grouping your baked geometry is a convenient way to manage the instantiated Rhino geometry, particularly if you are creating many objects with Grasshopper.



F.0.2.4 UNITS & TOLERANCES

Grasshopper inherits units and tolerances from Rhino. To change the units, type Document Properties in the Rhino command line to access the Document Properties menu. Select Units to change the units and tolerances.

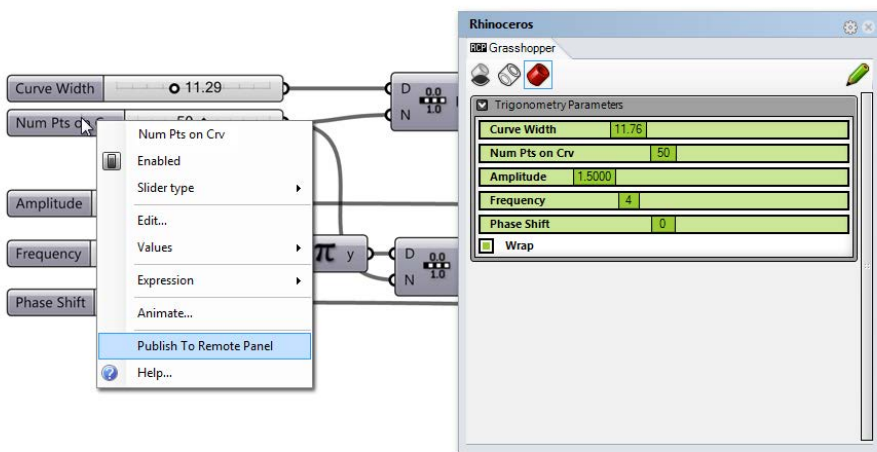


Change the units and tolerances in the Rhino Document Properties menu

F.0.2.5 REMOTE CONTROL PANEL

Once you get the hang of it, Grasshopper is an incredibly powerful and flexible tool which allows you to explore design iterations using a graphic interface. However, if you're working with a single screen then you may have already noticed that the Grasshopper editor takes up a lot of screen real-estate. Other than constantly zooming in and out and moving windows around your screen, there really isn't an elegant solution to this problem. That is...until the release of the Remote Control Panel!

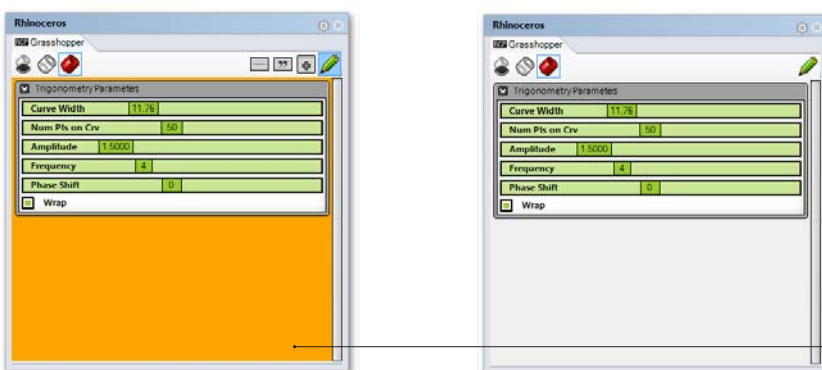
The Remote Control Panel (RCP) provides a minimal interface to control your definition without taking up a substantial portion of your screen. The RCP can be instantiated by clicking on the toggle under the View menu of the Main Menu bar. By default, the RCP is blank – meaning it doesn't contain any information about your current Grasshopper document. To populate the RCP with UI elements like sliders, toggles, and buttons, simply right click on the element and click Publish To Remote Panel. This will create a new group and create a synchronized UI element in the RCP. Changing the value of the element in the RCP will also update the value in the graph, as well as modify any geometry in the viewport which might be dependant on this parameter. You can publish multiple elements and populate a complete interface which can be used to control your file without having the clutter of the visual graph showing up on top of the Rhino viewport.



In order to get a UI element (eg. slider, toggle, button, etc.) to show up in the Remote Control Panel, we have to first publish it.

Note: The RCP will inherit the UI elements name and use it as the label. It is good practice to update your sliders and toggles with comprehensible and meaningful names. This will translate directly to your RCP making it easier to use.

The RCP UI can also be customized – allowing you to control where objects appear in the interface, the names and colors of different groups. To modify the layout of the RCP you first have to switch from Working Mode (the default RCP view) to Edit Mode. You can enter the Editing Mode by clicking on the green pencil in the upper right hand corner of the RCP. Once in Editing Mode, you can create new UI groups, rearrange elements within groups, add labels, change colors and more. To delete a UI element, simply drag the element outside the border of the RCP. You cannot change the individual values of the parameters if you are in Editing Mode. Instead, you will have to click on the green pencil icon to switch back to the standard Working Mode.

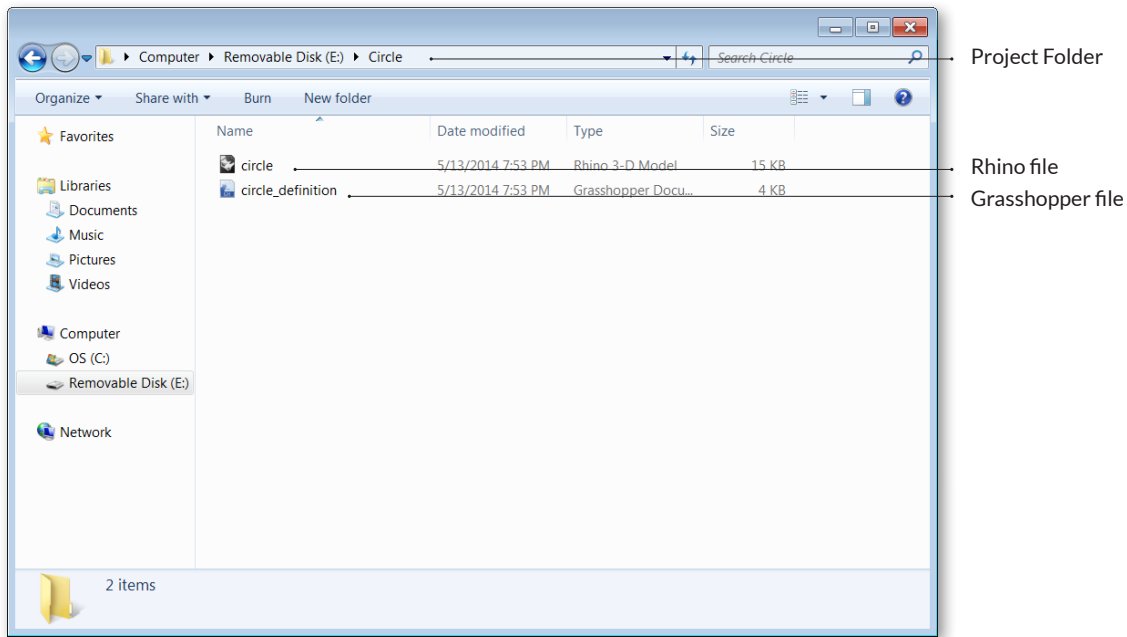


The Remote Control Panel has two modes: Edit Mode (left) which allows you to reorganize the look and feel of the RCP, and Working Mode where you can modify the actual values of the UI elements.

The Remote Control Panel in Edit Mode has an orange background.

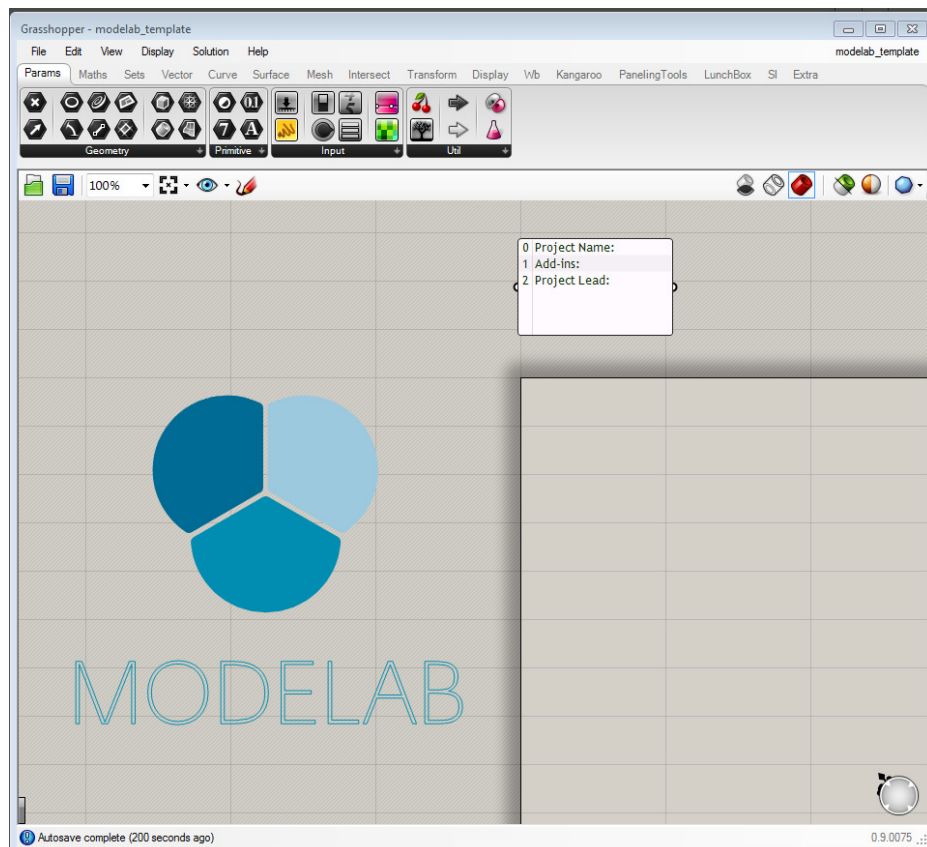
F.0.2.6 FILE MANAGEMENT

If your Grasshopper file references geometry from Rhino, you must open that same file for the definition to work. Keep your files organized by storing the Grasshopper and Rhino files in the same folder, and giving them related names.

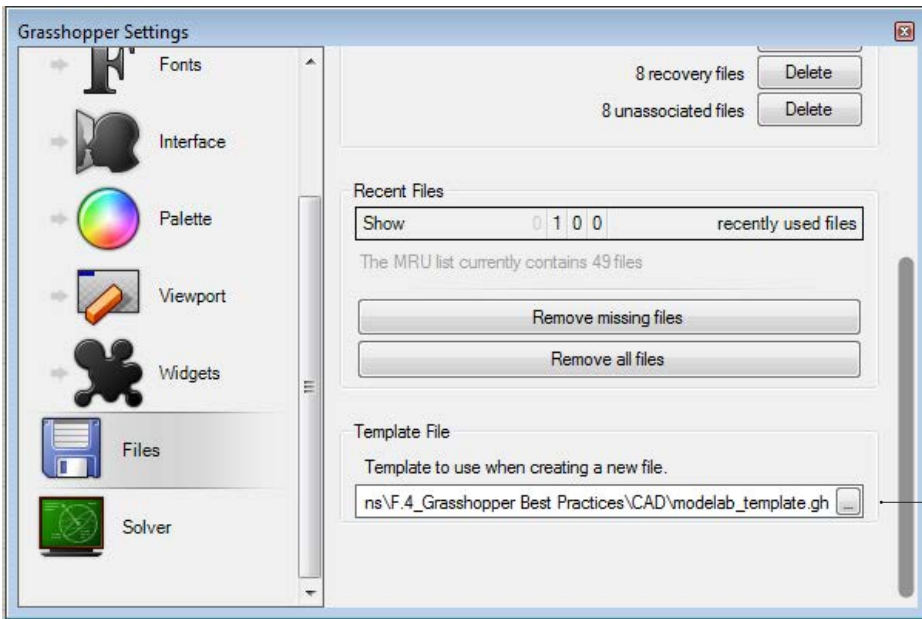


F.0.2.7 TEMPLATES

Creating and specifying a template file in your Grasshopper preferences is convenient way to set up every new Grasshopper definition you create. The template can include Grasshopper components as well as panels and sketch objects for labeling.



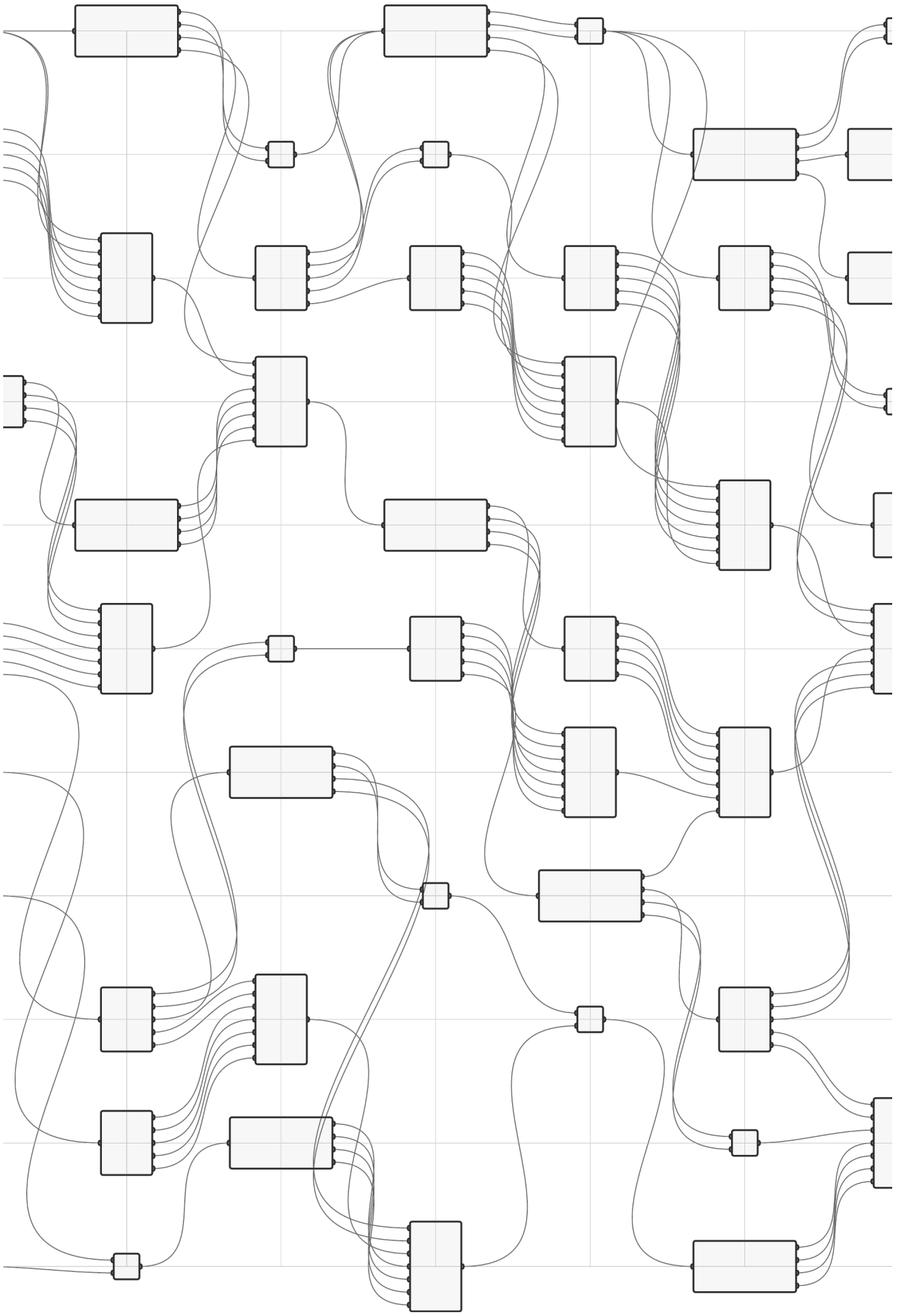
Create a template file and save it



In File/Preferences, load the file you just created under Template File. Your template will now be used each time you create a new file.

F.1 ANATOMY OF A GRASSHOPPER DEFINITION

Grasshopper allows you to create visual programs called definitions. These definitions are made up of nodes connected by wires. The following chapter introduces Grasshopper objects and how to interact with them to start building definitions.



F.1.0 Grasshopper Object Types

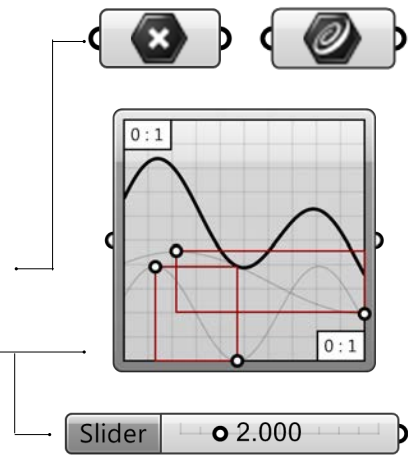
Grasshopper consists of two primary types of user objects: parameters and components. Parameters store data, whereas components perform actions that result in data. The most basic way to understand Grasshopper is to remember that we will use data to define the inputs of actions (which will result in new data that we can continue to use).

F.1.0.0 PARAMETERS

Parameters store the data - numbers, colors, geometry, and more - that we send through the graph in our definition. Parameters are container objects which are usually shown as small rectangular boxes with a single input and single output. We also know that these are parameters because of the shape of their icon. All parameter objects have a hexagonal border around their icon.

Geometry parameters can reference geometry from Rhino, or inherit geometry from other components. The point and curve objects are both geometry parameters.

Input parameters are dynamic interface objects that allow you to interact with your definition. The number slider and the graph mapper are both input parameters.



F.1.0.1 COMPONENTS

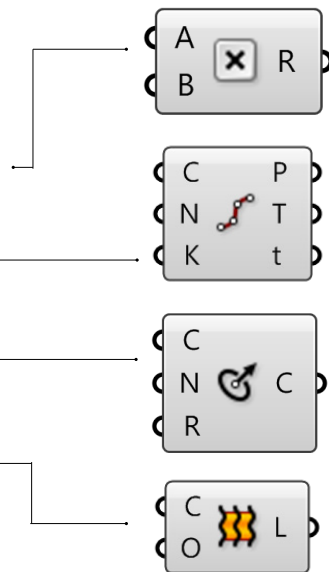
Components perform actions based on the inputs they receive. There are many types of components for different tasks.

The multiplication component is an operator that calculates the product of two numbers.

The Divide component operates on geometry, dividing a curve into equal segments.

The Circle CNR component constructs a circle geometry from input data; a center point, normal vector, and radius.

The Loft component constructs a surface by lofting curves.



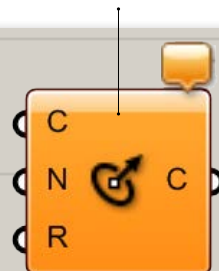
A parameter with no warnings or errors



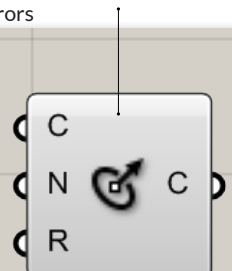
A parameter with warnings



A component with warnings



A component with no warnings or errors



F.1.0.2 OBJECT COLORS

We can glean some information about the state of each object based on their color. Let's take a look at Grasshopper's default color coding system.

A parameter which contains neither warnings nor errors is shown in light gray. This color object indicates that everything is working properly with this parameter.

A parameter which contains warnings is displayed as an orange box. Any object which fails to collect data is considered suspect in a Grasshopper definition since it is not contributing to the solution.. Therefore, all parameters (when freshly added) are orange, to indicate they do not contain any data and have thus no functional effect on the outcome of the solution. By default, parameters and components that are orange also have a small balloon at the upper right hand corner of the object. If you hover your mouse over this balloon, it will reveal information about why the component is giving you a warning. Once a parameter inherits or defines data, it will become grey and the balloon will disappear.

A component is always a more involved object, since we have to understand and then coordinate what its inputs and outputs are. Like parameters, a component with warnings is displayed as orange. Remember, warnings aren't necessarily bad, it usually just means that Grasshopper is alerting you to a potential problem in your definition.

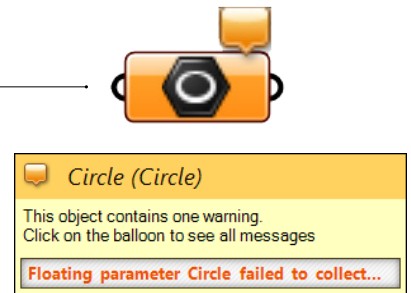
A component which contains neither warnings nor errors is shown in light gray.

A component whose preview has been disabled is shown in a slightly darker gray. There are two ways to disable a component's preview. First, simply right-click on the component and toggle the preview button. To disable the preview for multiple components at the same time, first select the desired components and then toggle the disable preview icon (blindfolded man) by right clicking anywhere on the canvas.

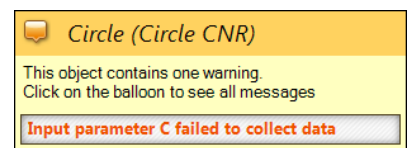
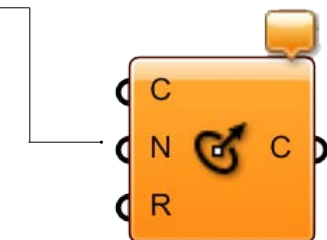
A component that has been disabled is shown in a dull gray. To disable a component you may right-click on the component and toggle the disable button, or you may select the desired components, right click anywhere on the canvas and select Disable. Disabled components stop sending data to downstream components.

A component which has been selected will be shown in a light green color. If the selected component has generated some geometry within the Rhino scene, this will also turn green to give you some visual feedback.

A component which contains at least 1 error is displayed in red. The error can come either from the component itself or from one of its inputs or outputs.

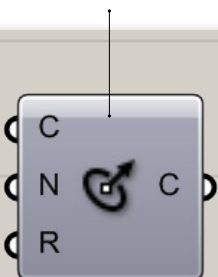


If you hover your mouse over the balloon in the upper right hand corner of the parameter, a tooltip will explain the warning.

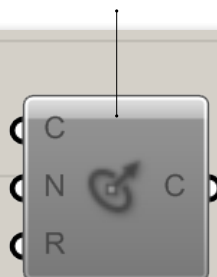


This component contains a warning because it does not have enough information to create a circle.

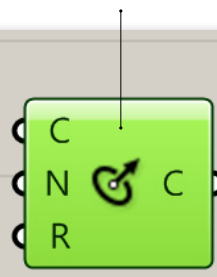
A component with preview disabled



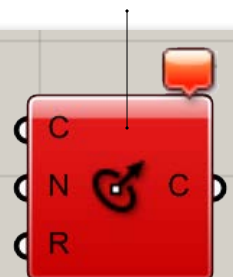
A component that has been disabled



A selected component



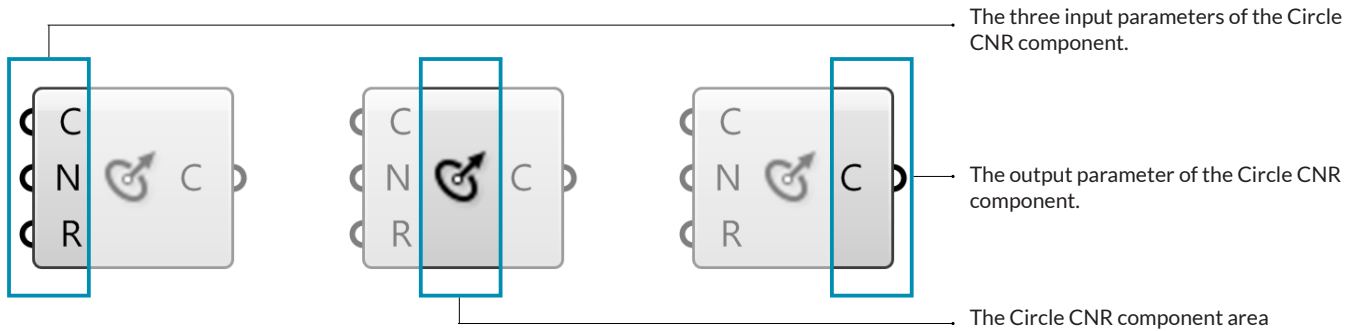
A component with an error



F.1.1

Grasshopper Component Parts

Components are the objects you place on the canvas and connect together with Wires to form a visual program. Components can represent Rhino Geometry or operations like Math Functions. Components have inputs and outputs.



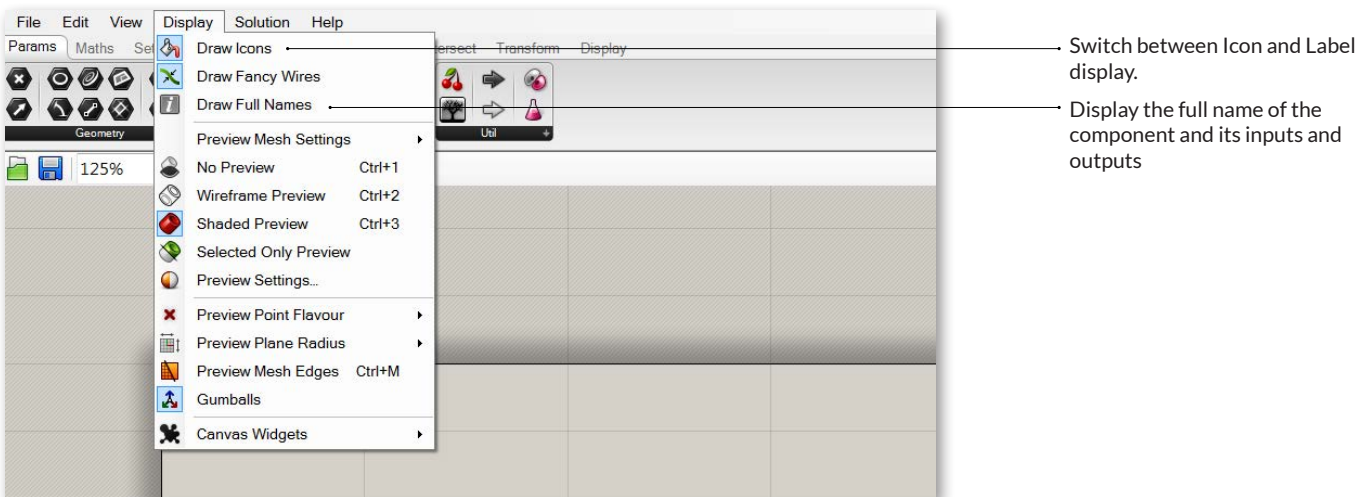
A component requires data in order to perform its actions, and it usually comes up with a result. That is why most components have a set of nested parameters, referred to as Inputs and Outputs, respectively. Input parameters are positioned along the left side, output parameters along the right side.

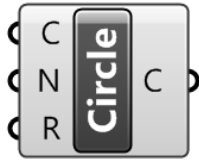
There are a few Grasshopper components that have inputs but no outputs, or vice versa. When a component doesn't have inputs or outputs, it will have a jagged edge.



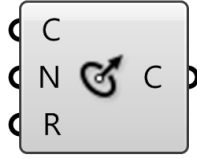
F.1.1.0 LABEL VS ICON DISPLAY

Every Grasshopper object has a unique icon. These icons are displayed in the center area of the object and correspond to the icons displayed in the component palettes. Objects can also be displayed with text labels. To switch between icon and label display, Select "Draw Icons" from the display menu. You can also select "Draw Full Names" to display the full name of each object as well as its inputs and outputs.

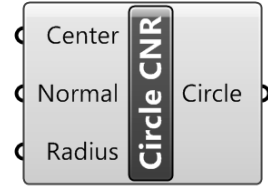




The Circle CNR component in Label Display

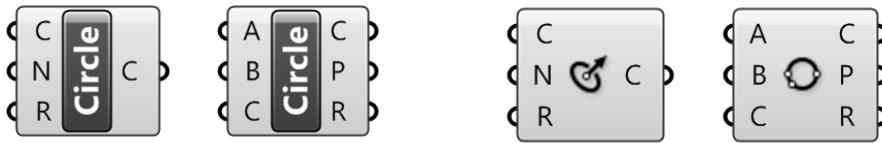


The Circle CNR component in Icon Display



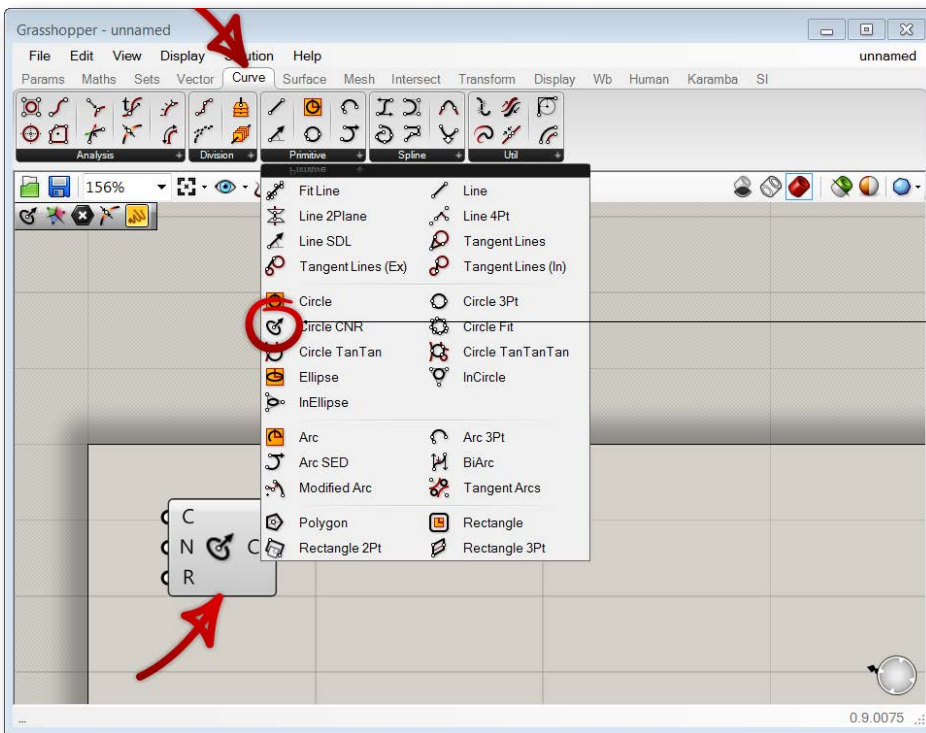
The Circle CNR component with full names displayed

We recommend using icon display to familiarize yourself with the component icons so you can quickly locate them in the palettes. This will also enable you to understand definitions at a glance. Text labels can be confusing because different components may share the same label.



Circle CNR and Circle 3pt have the same label, but different icons.

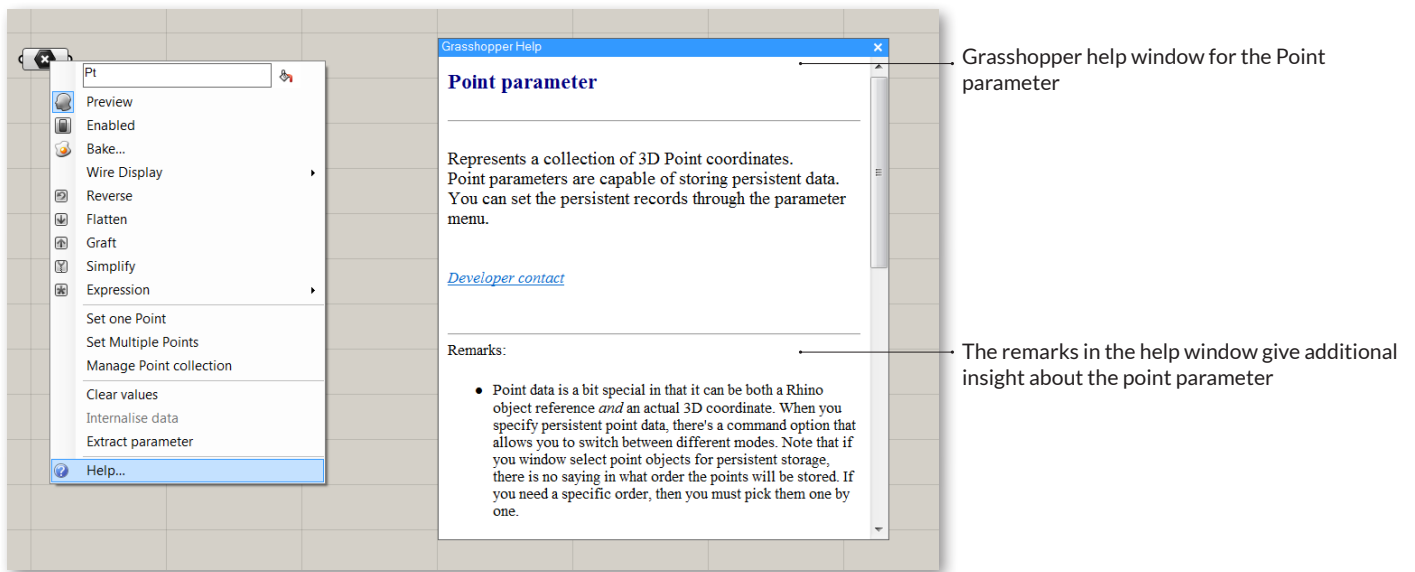
One feature that can help you familiarize yourself with the location of components in the palettes is holding down Ctrl + Alt and clicking on an existing component on the canvas. This will reveal its location in the palette.



Hold Cntrl + Alt and click on a component on the canvas to reveal its location in the palettes

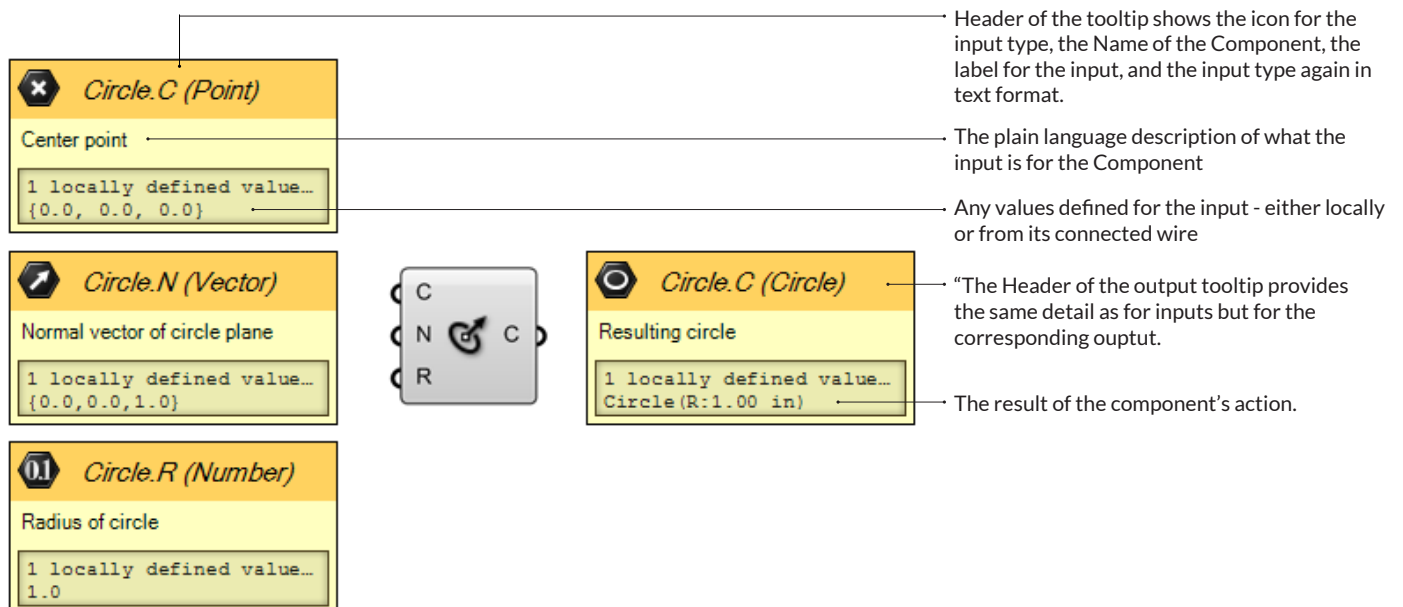
F.1.1.1 COMPONENT HELP

Right clicking an object and selecting “Help” from the drop-down menu will open a Grasshopper help window. The help window contains a more detailed description of the object, a list of inputs and outputs, as well as remarks.



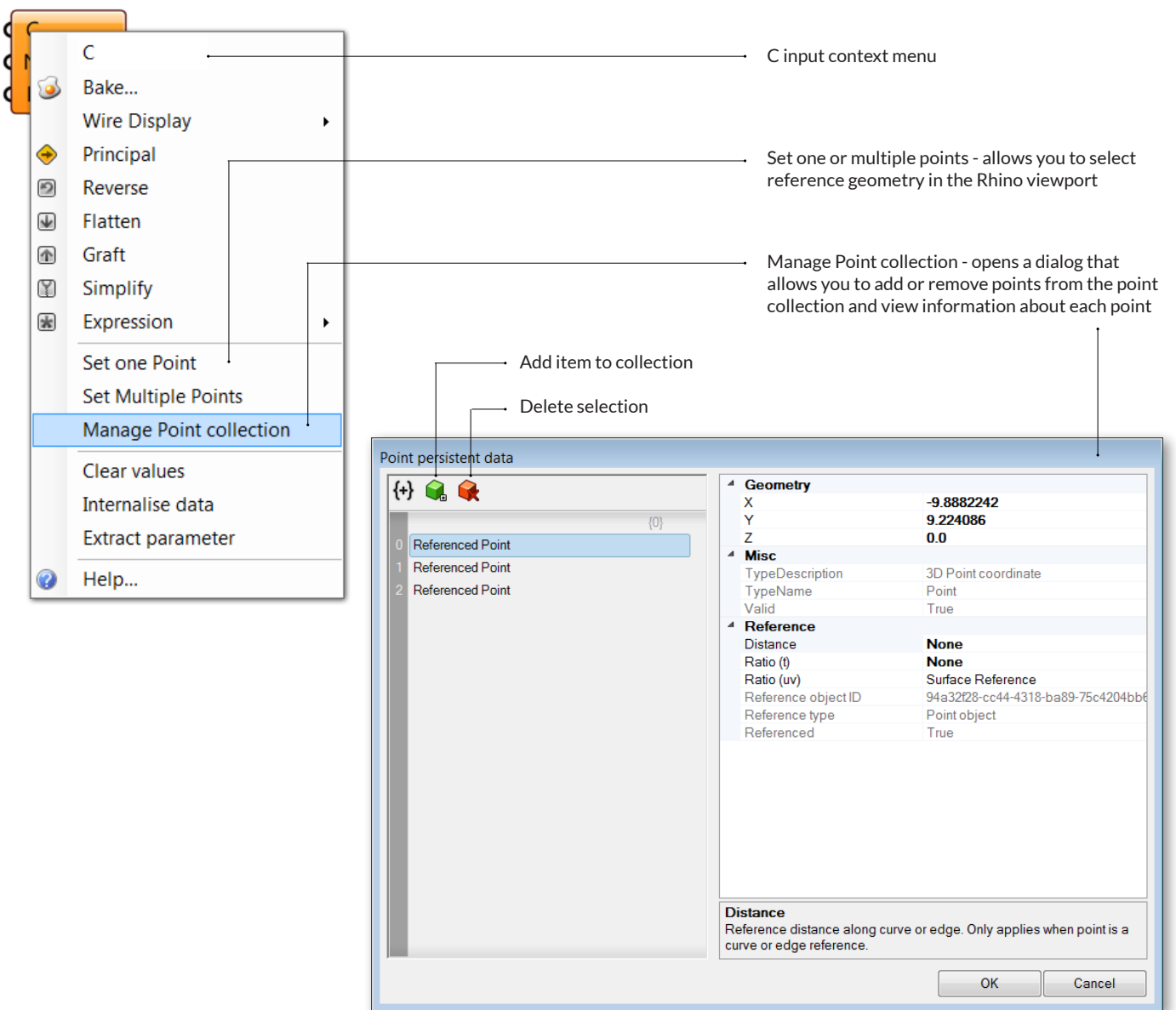
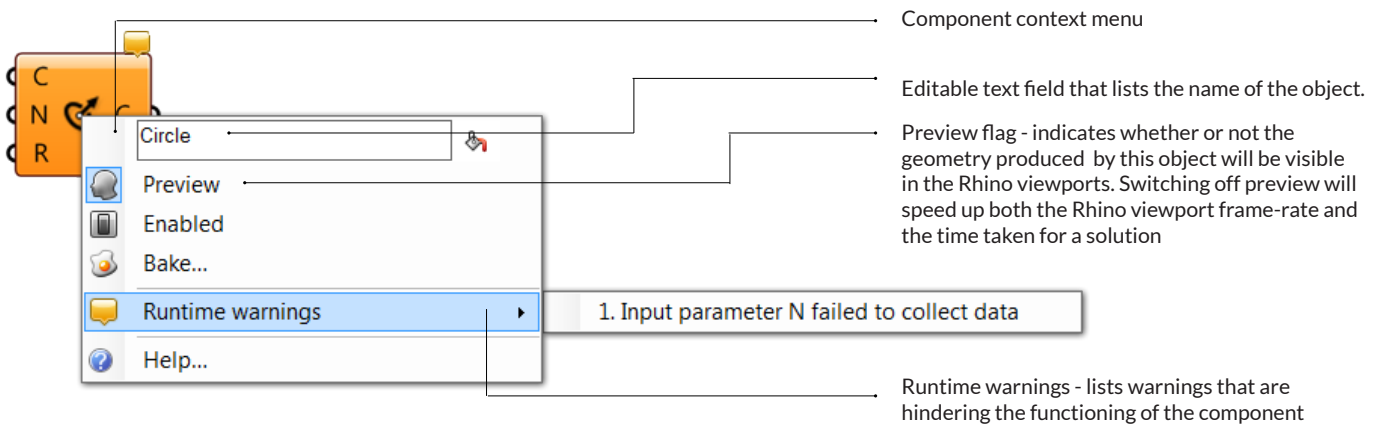
F.1.1.2 TOOL TIPS

Component inputs are expecting to receive certain types of data, for example a Component might indicate that you should connect a point or plane to its input. When you hover your mouse over the individual parts of a Component object, you'll see different tooltips that indicate the particular type of the sub-object currently under the mouse. Tooltips are quite informative since they tell you both the type and the data of individual parameters.



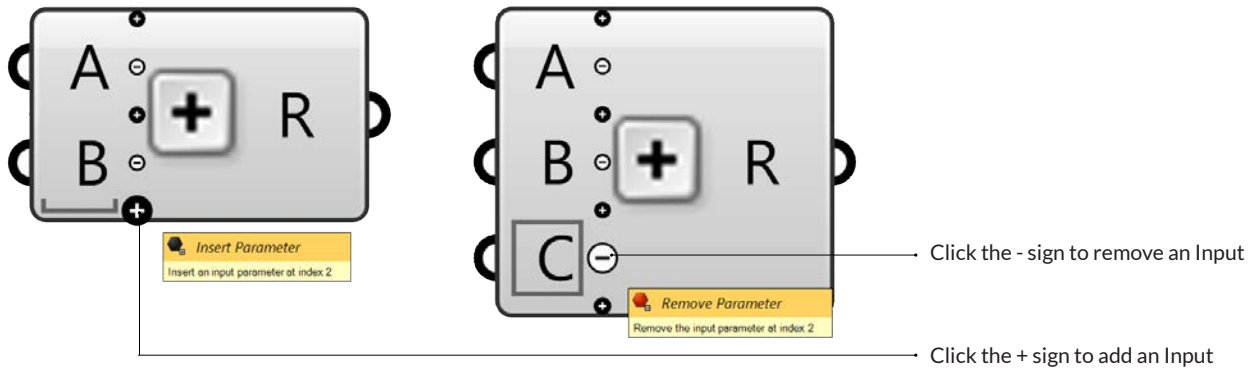
F.1.1.3 CONTEXT POPUP MENUS

All objects on the Canvas have their own context menus that expose their settings and details. You can access this context menu by right-clicking on the center area of each component. Inputs and outputs each have their own context menus which can be accessed by right-clicking them.

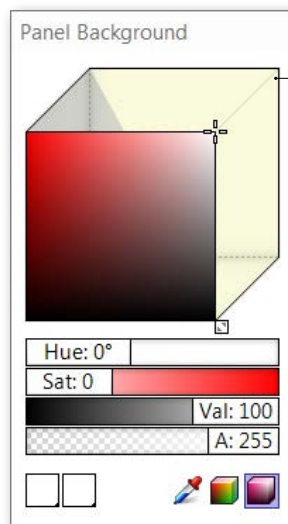
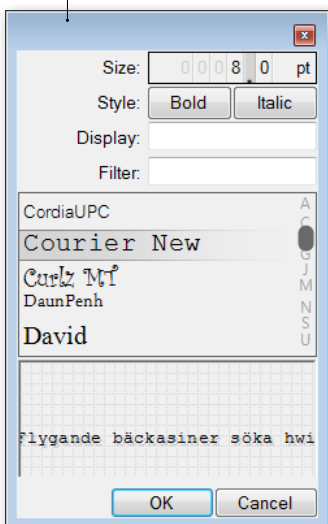
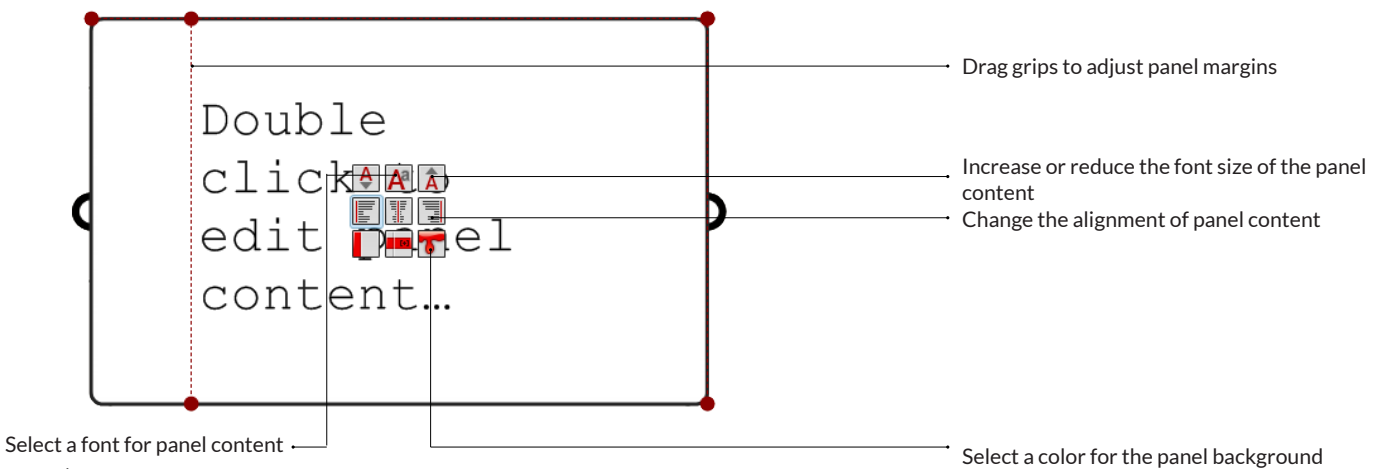


F.1.1.4 ZOOMABLE USER INTERFACE

Some components can be modified to increase the number of inputs or outputs through the Zoomable User Interface (ZUI). By zooming in on the component on the canvas, an additional set of options will appear which allows you add or remove Inputs or Outputs to that component. The Addition component allows you to add inputs, representing additional items for the addition operation.



The panel component also has a zoomable user interface. A Panel is like a Post-It™ sticker. It allows you to add little remarks or explanations to a Document. You can change the text through the menu or by double-clicking the panel surface. Panels can also receive and display data from elsewhere. If you plug an output into a Panel, you can see the contents of that parameter in real-time. All data in Grasshopper can be viewed in this way. When you zoom in on a panel, a menu appears allowing you to change the background, font, and other attributes. These options are also available when you right-click the panel



Note: You can set a new default color for your panels by right clicking the panel and selecting "Set Default Color"

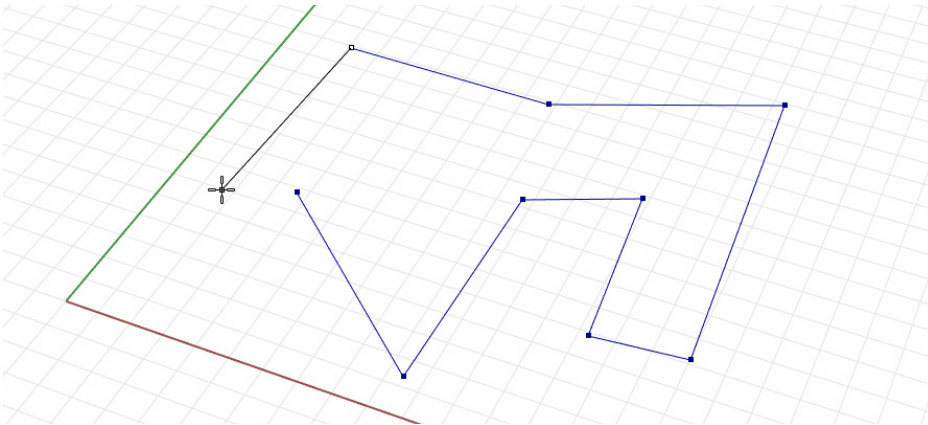
F.1.2 Data Types

Most parameters can store two different kinds of data: Volatile and Persistent. Volatile data is inherited from one or more sources and is destroyed (i.e. recollected) whenever a new solution starts. Persistent data is data which has been specifically set by the user.

F.1.2.0 PERSISTENT DATA

Persistent data is accessed through the menu, and depending on the kind of parameter has a different manager. A Point parameter for example allows you to set one or more points through its menu. But, let's back up a few steps and see how a Point Parameter behaves.

When you drag and drop a Point Parameter from the Params/Geometry Panel onto the canvas, the Parameter is orange, indicating it generated a warning. It's nothing serious, the warning is simply there to inform you that the parameter is empty (it contains no persistent records and it failed to collect volatile data) and thus has no effect on the outcome of the solution. The context menu of the Parameter offers two ways of setting persistent data: single and multiple. Right click on the parameter to set Multiple Points. Once you click on either of these menu items, the Grasshopper window will disappear and you will be asked to pick a point in one of the Rhino viewports.



Once you have defined all the points you can press Enter and they will become part of the Parameters persistent data record. This means the Parameter is now no longer empty and it turns from orange to grey. (Notice that the information balloon in the upper right corner also disappears as there are no more warnings). At this point you can use the points stored in this Parameter for any subsequent input in your definition.

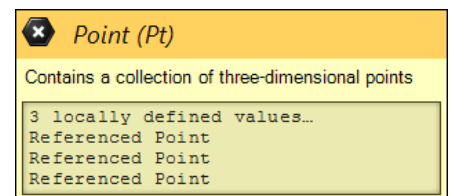
The exception here are outputs, which can neither store permanent records nor define a set of sources. Outputs are explicitly dependent upon the component of which they are a part.



The parameter is orange, indicating it contains no persistent records and it failed to collect volatile data) and thus has no effect on the outcome of the solution. Right click on any parameter to set its persistent data.



Once the parameter contains some persistent data, the component will turn from orange to grey.



The tooltip for the point parameter shows the persistent data (a collection of referenced points) that is stored

F.1.2.1 VOLATILE DATA

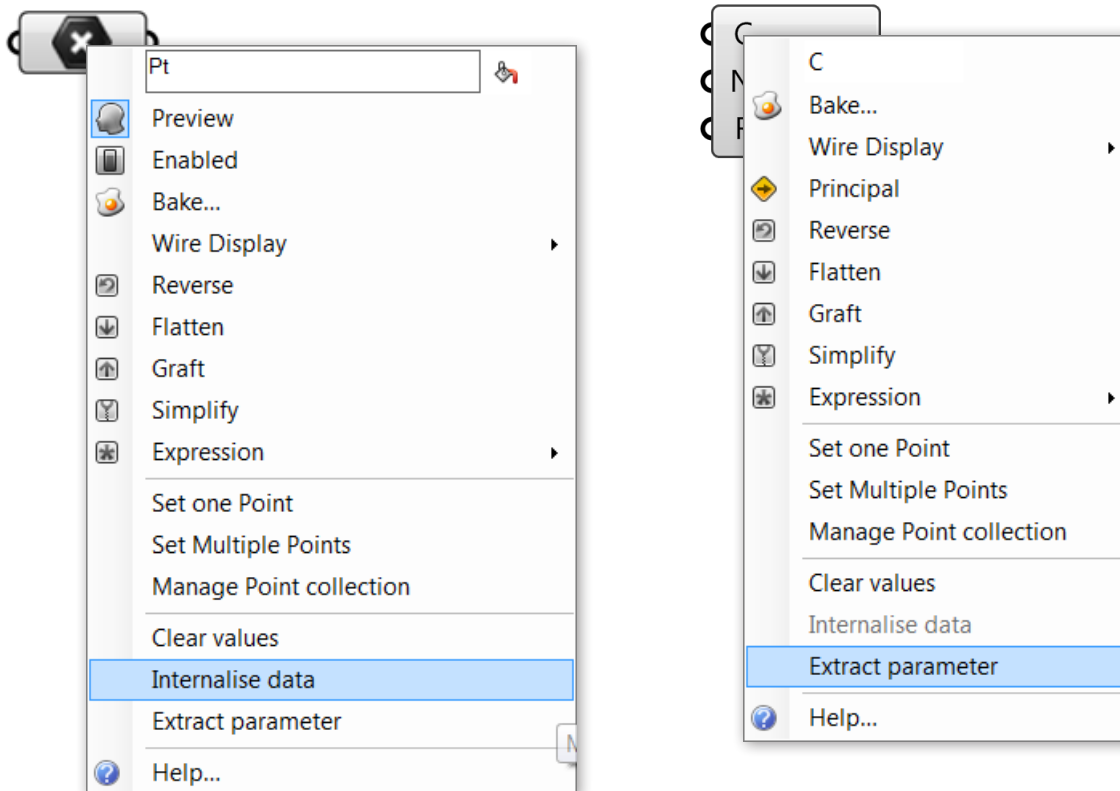
Volatile data, as the name suggests, is not permanent and will be destroyed each time the solution is expired. However, this will often trigger an event to rebuild the solution and update the scene. Generally speaking, most of the data generated 'on the fly' is considered volatile.

As previously stated, Grasshopper data is stored in Parameters (either in Volatile or Persistent form) and is used in various Components. When data is not stored in the permanent record set of a Parameter, it must be inherited from elsewhere. Every Parameter (except output parameters) defines where it gets its data from and most Parameters are not very particular. You can plug a number Parameter (which just means that it is a decimal number) into an integer source and it will take care of the conversion.

You can change the way data is inherited and stored in the context menu of a parameter or component input. To change store referenced Rhino geometry in the grasshopper definition itself, right click a parameter and select Internalise data from the menu. This is useful if you want your grasshopper definition to be independent from a Rhino file.

You can also Internalise data in a component input. Once you select Internalise data in the menu, any wires will disconnect from that input. The data has been changed from volatile to persistent, and will no longer update.

If you want the data to become volatile again, simply reconnect the wires to the input and the values will automatically be replaced. You can also right click the input and select Extract parameter. Grasshopper will create a parameter connected to the input with a wire that contains the data.



F.1.2.2 INPUT PARAMETERS

Grasshopper has a variety of Parameters that offer you the ability to interface with the data that is begin supplied to Component inputs and thereby control for changing the result of your definition. Because they Parameters that change with our input, they generate Volatile Data.

Number Slider

The number slider is the most important and widely used Input Parameter. It allows us to output a value between two given extremes by interacting with its grip with our mouse. Sliders can be used to specify a value and see the change to our deifnition that comes with adjusting the grip, but a slider should also be thought of as the means to identify successful ranges of our definition.

The diagram illustrates the interaction with a Grasshopper Slider component. On the left, a slider is shown with a value of 2.000. A context menu is open over it, with annotations explaining its options:

- Right click the slider component to change the name, type, and values**: Points to the context menu.
- Editable text field for the slider name**: Points to the 'Slider type' option in the menu.
- Select the type of number for the slider to use**: Points to the sub-menu containing 'Floating point', 'Integers', 'Odd numbers', and 'Even numbers'.
- Edit the range of values**: Points to the 'Values' option in the menu.

 Below the slider, two instructions are provided:

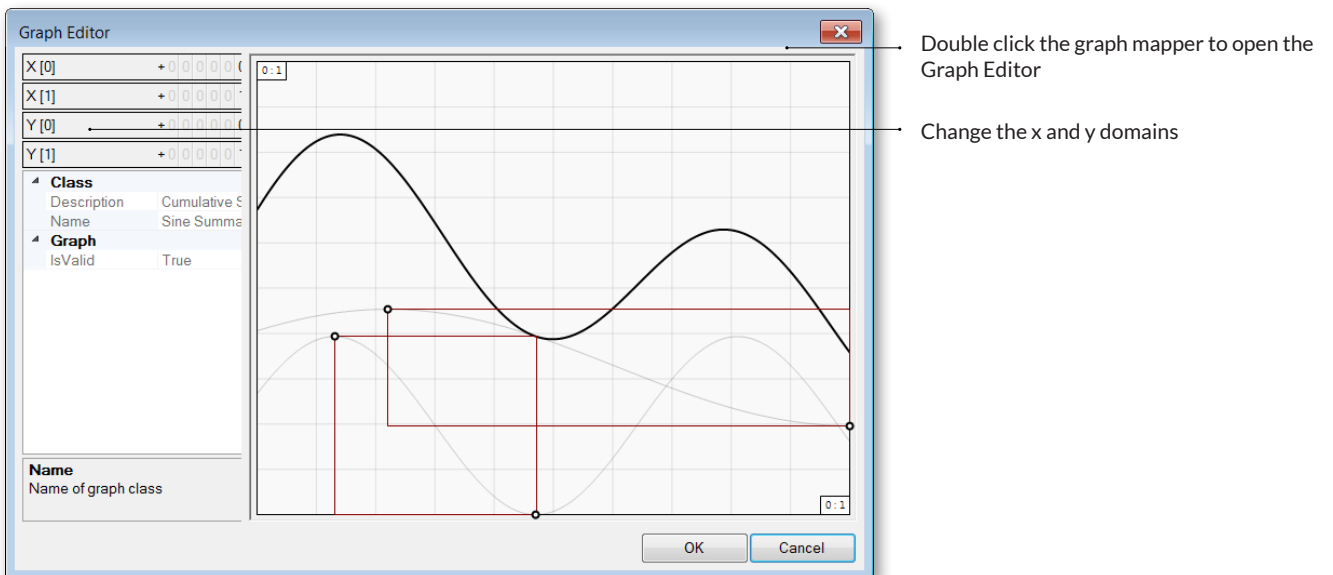
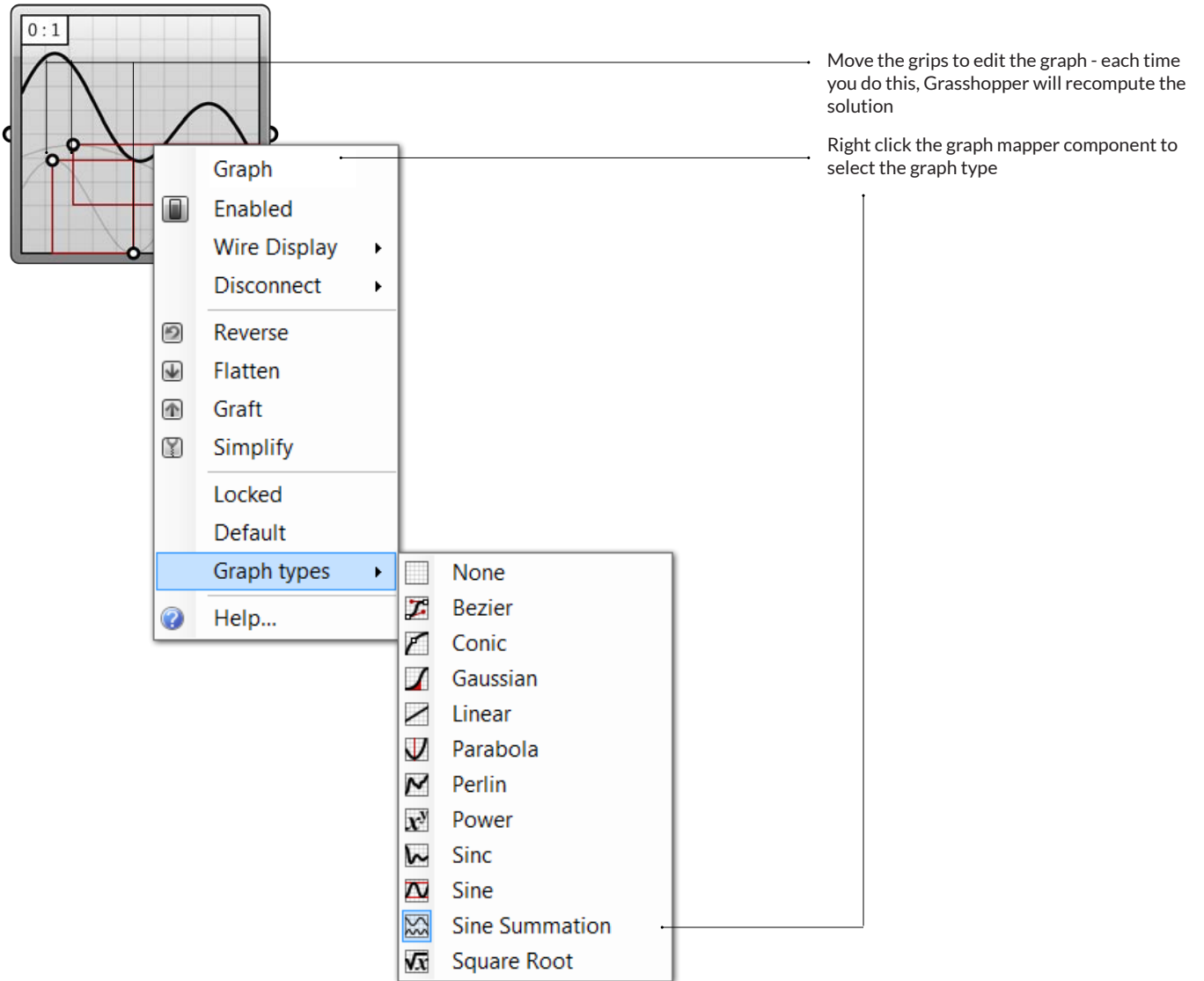
- Drag the slider grip to change the value - each time you do this, Grasshopper will recompute the solution**: Points to the slider's grip.
- Double click the name portion of the slider component to open the Slider Editor**: Points to the 'Slider' text on the component.

 On the right, the **Slider:** dialog box is shown, which allows for detailed configuration:

- Properties**: Includes fields for Name, Expression, and Grip Style (set to 'Shape & Text').
- Slider accuracy**: Features 'Rounding' buttons (R, N, E, O) and a 'Digits' slider set to 3.
- Numeric domain**: Contains input fields for Min (+0.0000000000000000), Max (+0.0000000000000005), and Range (0.0000000000000005).
- Numeric value**: Includes a text input field (0.0000000000000002) and a visual slider set to 2.
- Buttons**: 'OK' and 'Cancel' buttons are at the bottom.

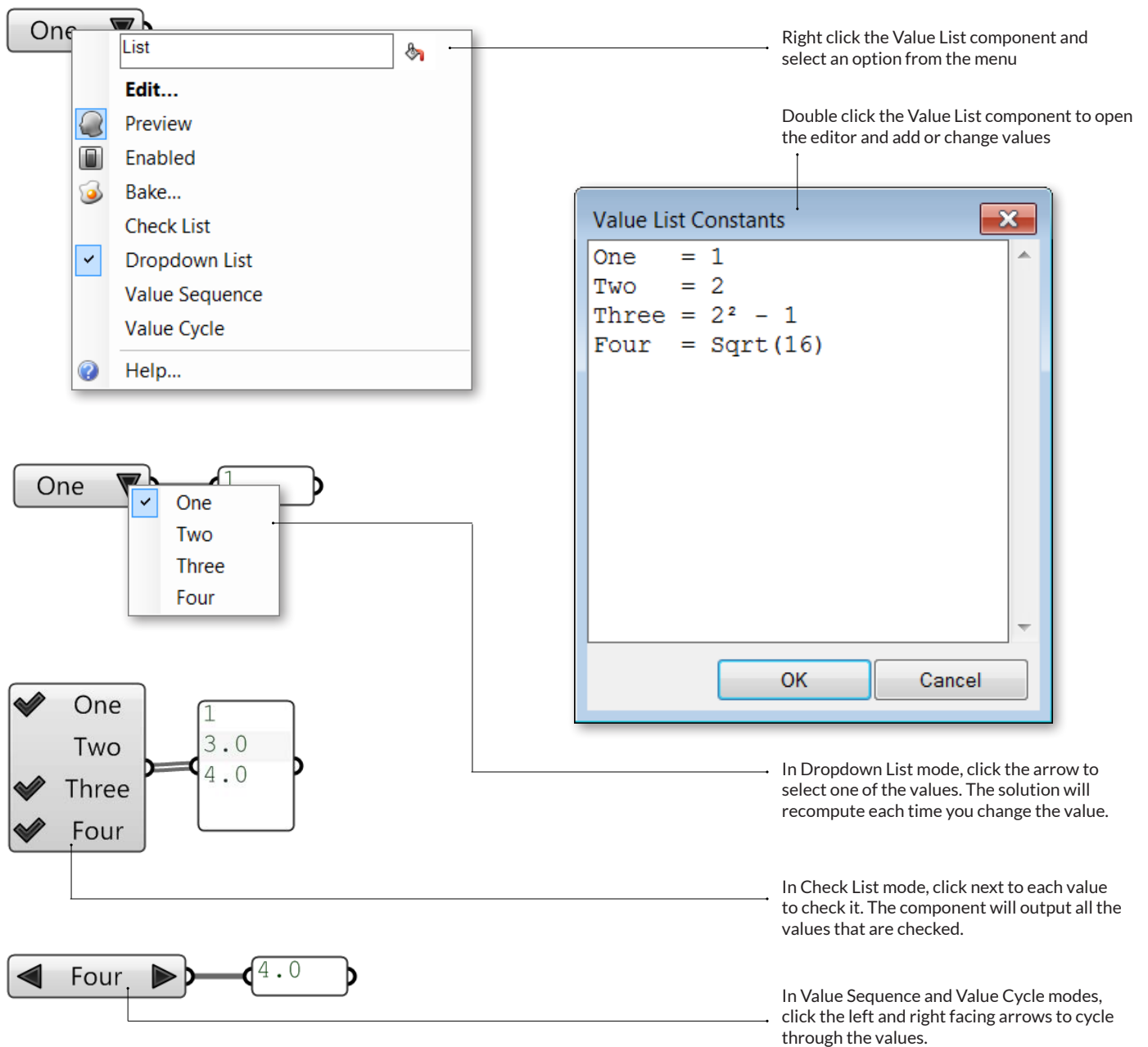
Graph mapper

The Graph Mapper is a two-dimensional interface with which we can modify numerical values by plotting the input along the Graph's X Axis and outputting the corresponding value along the Y Axis at the X value intersection of the Graph. It is extremely useful for modulating a set of values within an instinctive, grip-based interface.



Value List

The Value List stores a collection of values with corresponding list of Labels associated by way of an equal sign. It is particularly useful when you want to have a few options, labeled with meaning, that can supply specific output values.

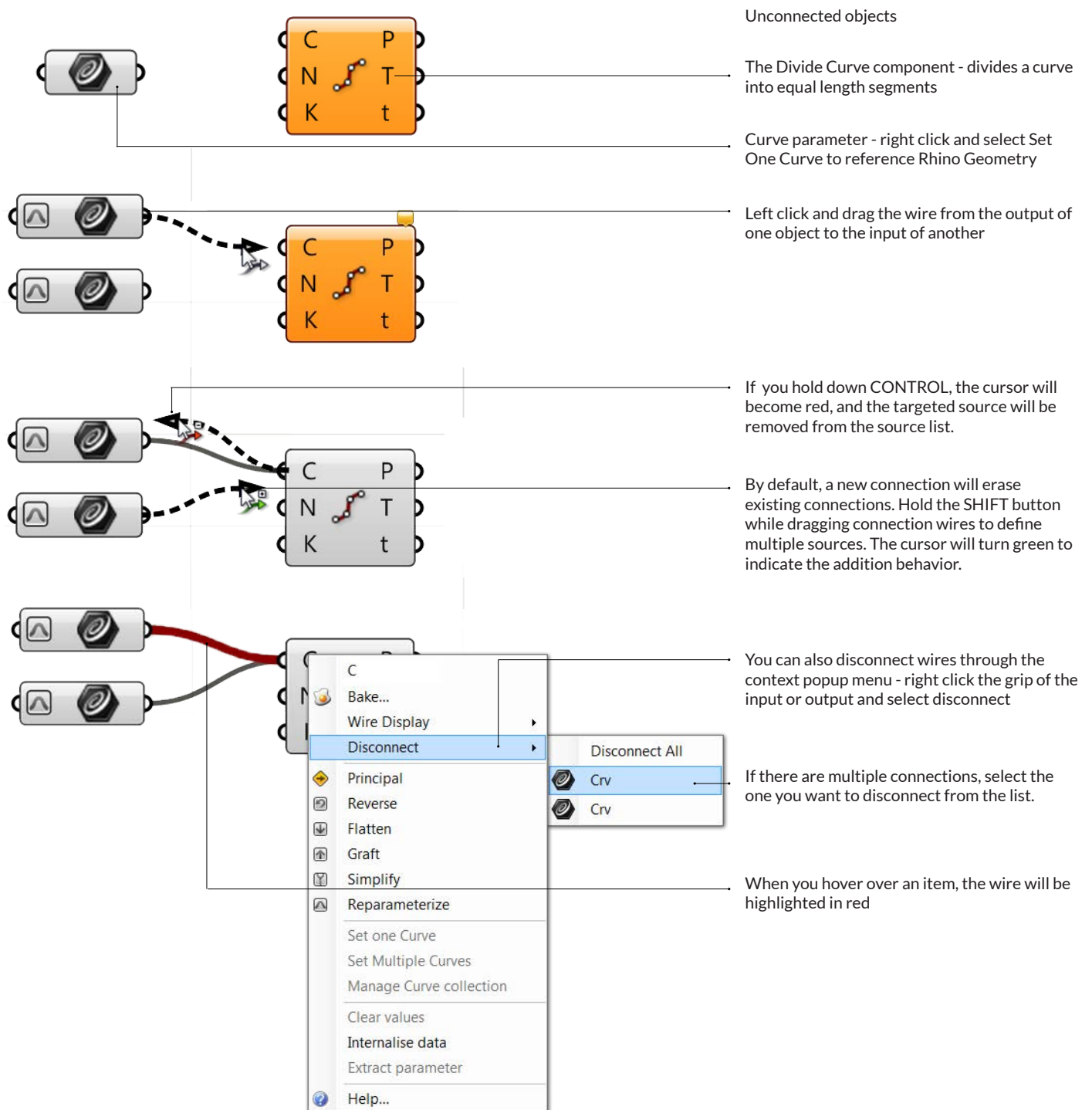


F.1.3 Wiring Components

When data is not stored in the permanent record set of a parameter, it must be inherited from elsewhere. Data is passed from one component to another through wires. You can think of them literally as electrical wires that carry pulses of data from one object to the next.

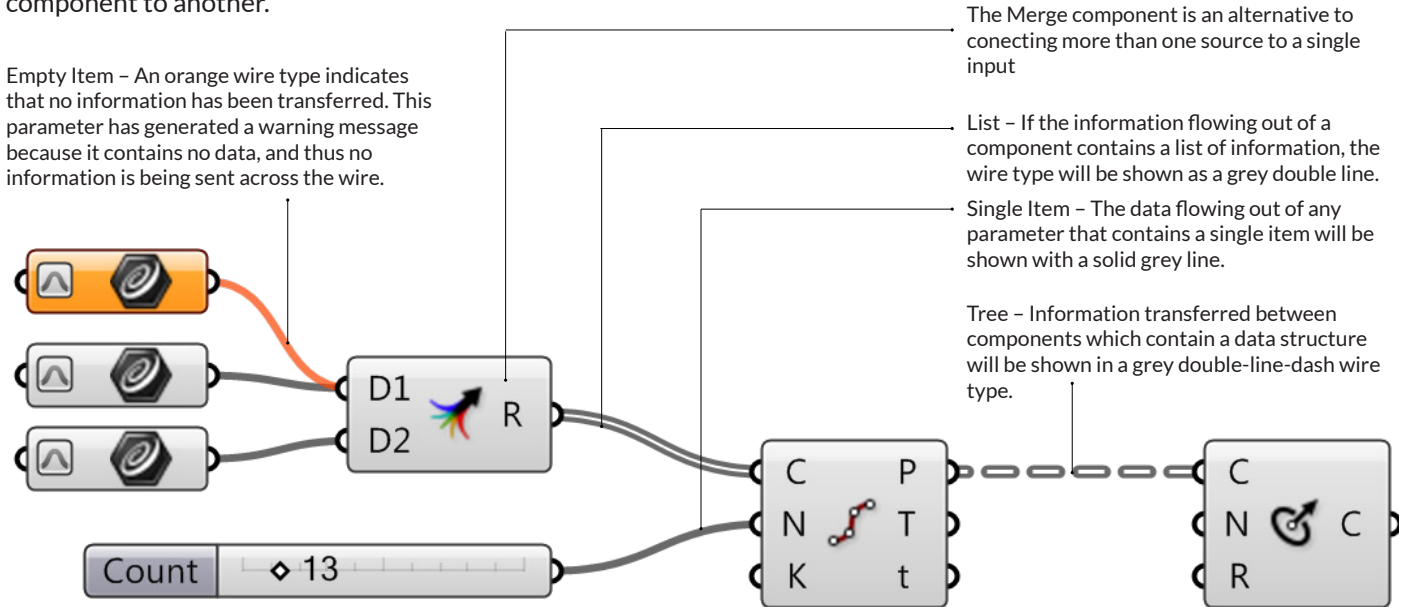
F.1.3.0 CONNECTION MANAGEMENT

To connect components, click and drag near the circle on the output side of an object. A connecting wire will be attached to the mouse. Once the mouse hovers over a potential target input, the wire will connect and become solid. This is not a permanent connection until you release the mouse button. It doesn't matter if we make the connections in a 'left to right' or 'right to left' manner.



F.1.3.1 FANCY WIRES

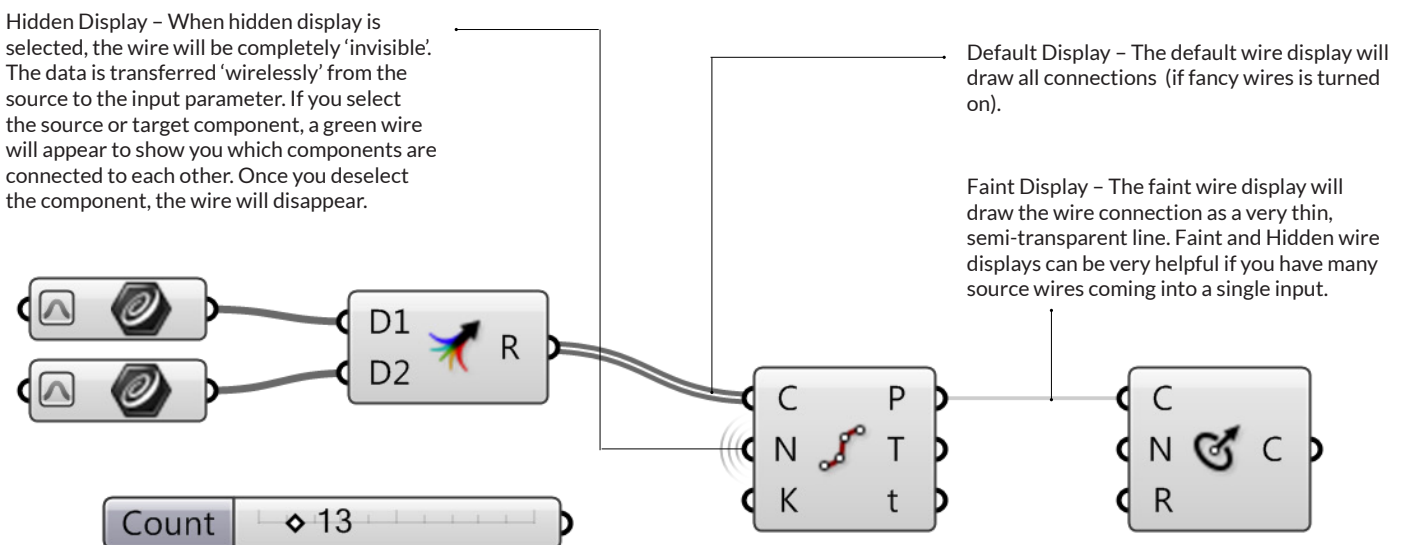
Wires represent the connections as well as the flow of data within the graph in our definition. Grasshopper can also give us visual clues as to what is flowing through the wires. The default setting for these so-called “fancy wires” is off, so you have to enable them before you can view the different types of line types for the connection wires. To do this, simply click on the View Tab on the Main Menu Bar and select the button labeled “Draw Fancy Wires.” Fancy wires can tell you a lot of information about what type of information is flowing from one component to another.



F.1.3.2 WIRE DISPLAY

If you have spent any great deal of time working on a single Grasshopper definition, you may have realized that the canvas can get cluttered with a nest of wires quite quickly. Fortunately, we have the ability to manage the wire displays for each input of a component.

There are three wire displays: Default Display, Faint Display, and Hidden Display. To change the wire display, simply right-click on any input on a component and select one of the views available under the Wire Display pop out menu.

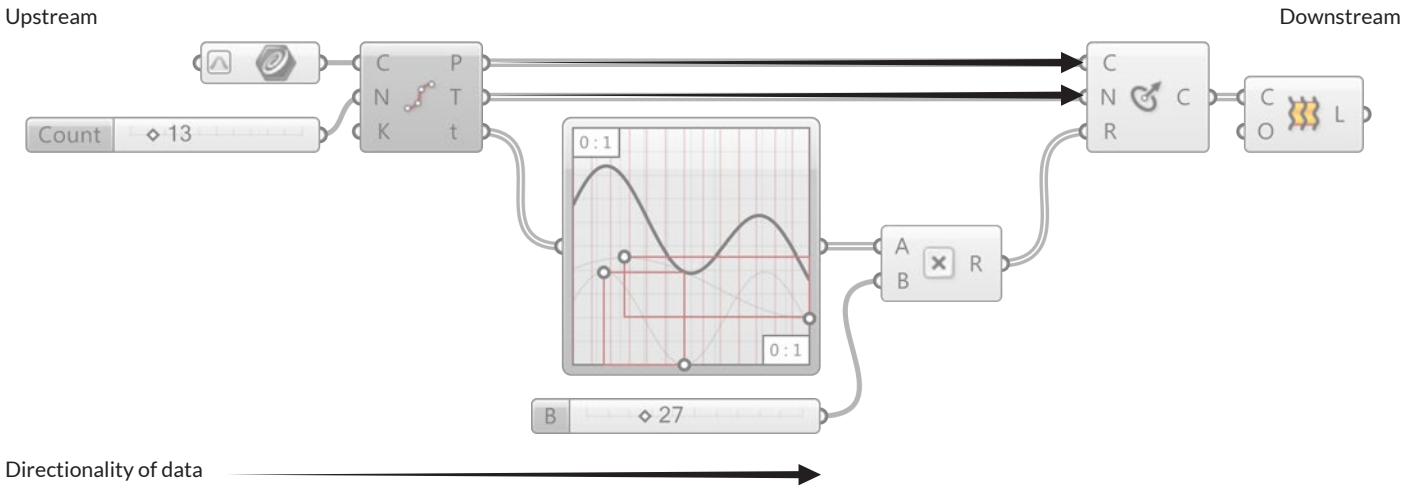


F.1.4 The Grasshopper Definition

Grasshopper Definitions have a Program Flow that represents where to start program execution, what to do in the middle and how to know when program execution is complete.

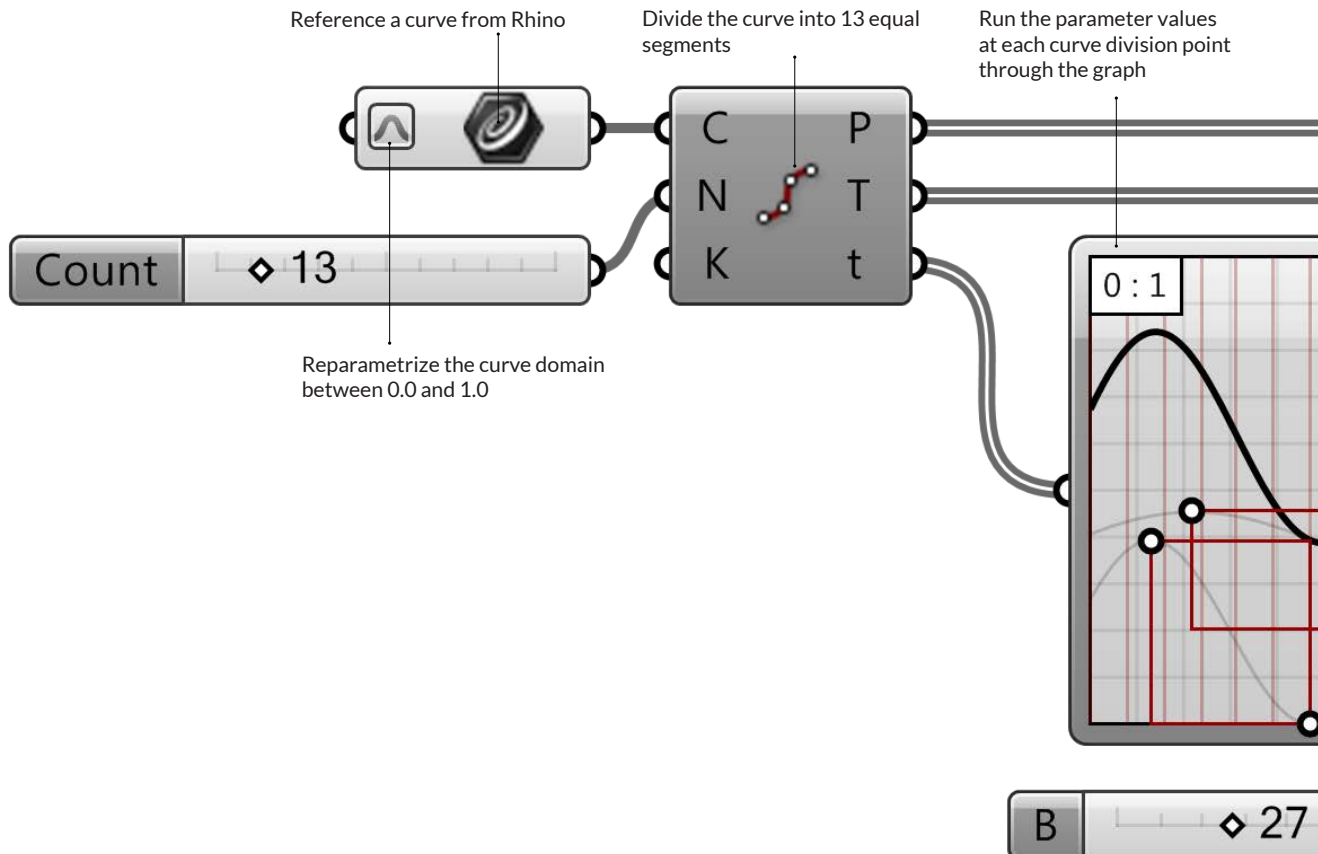
F.1.4.0 PROGRAM FLOW

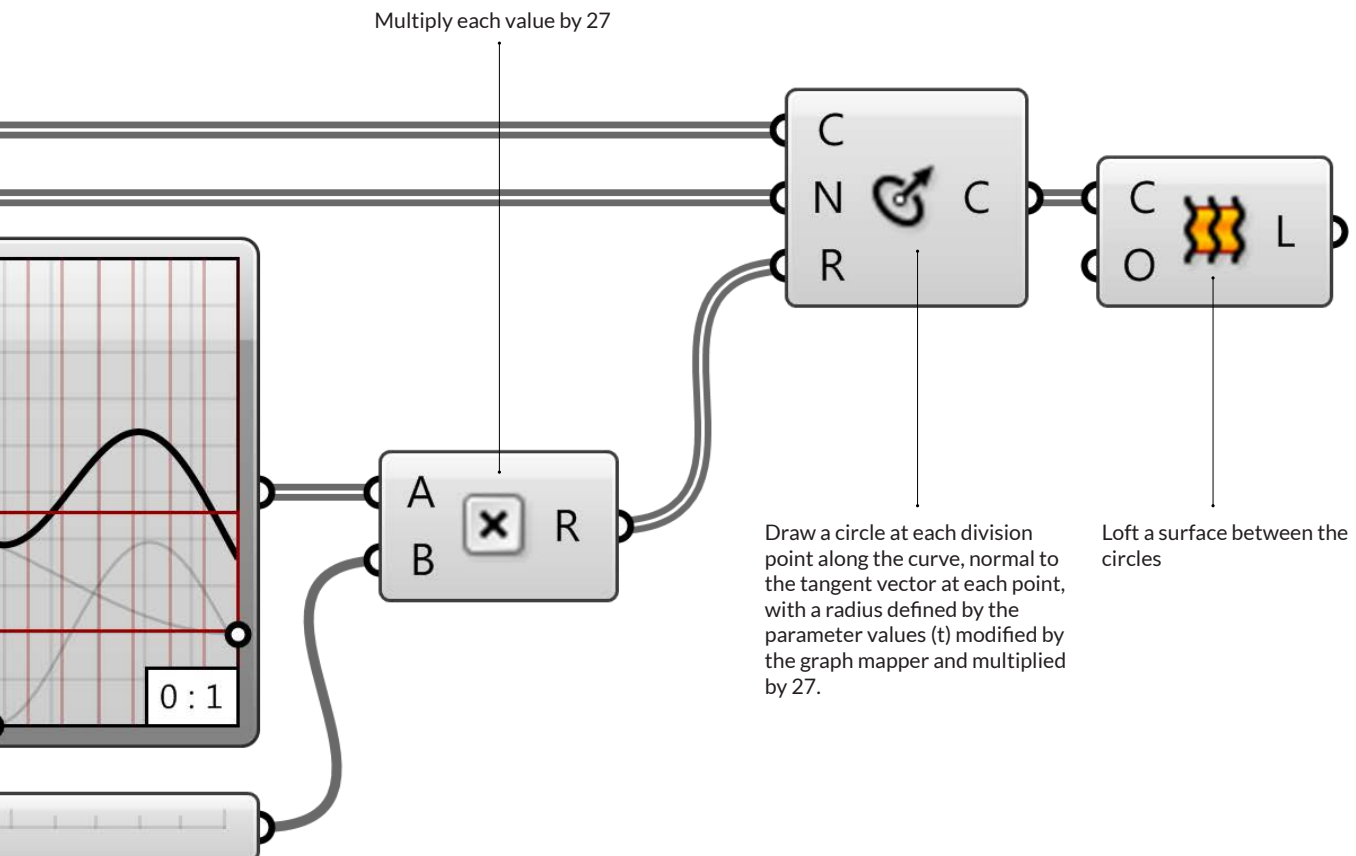
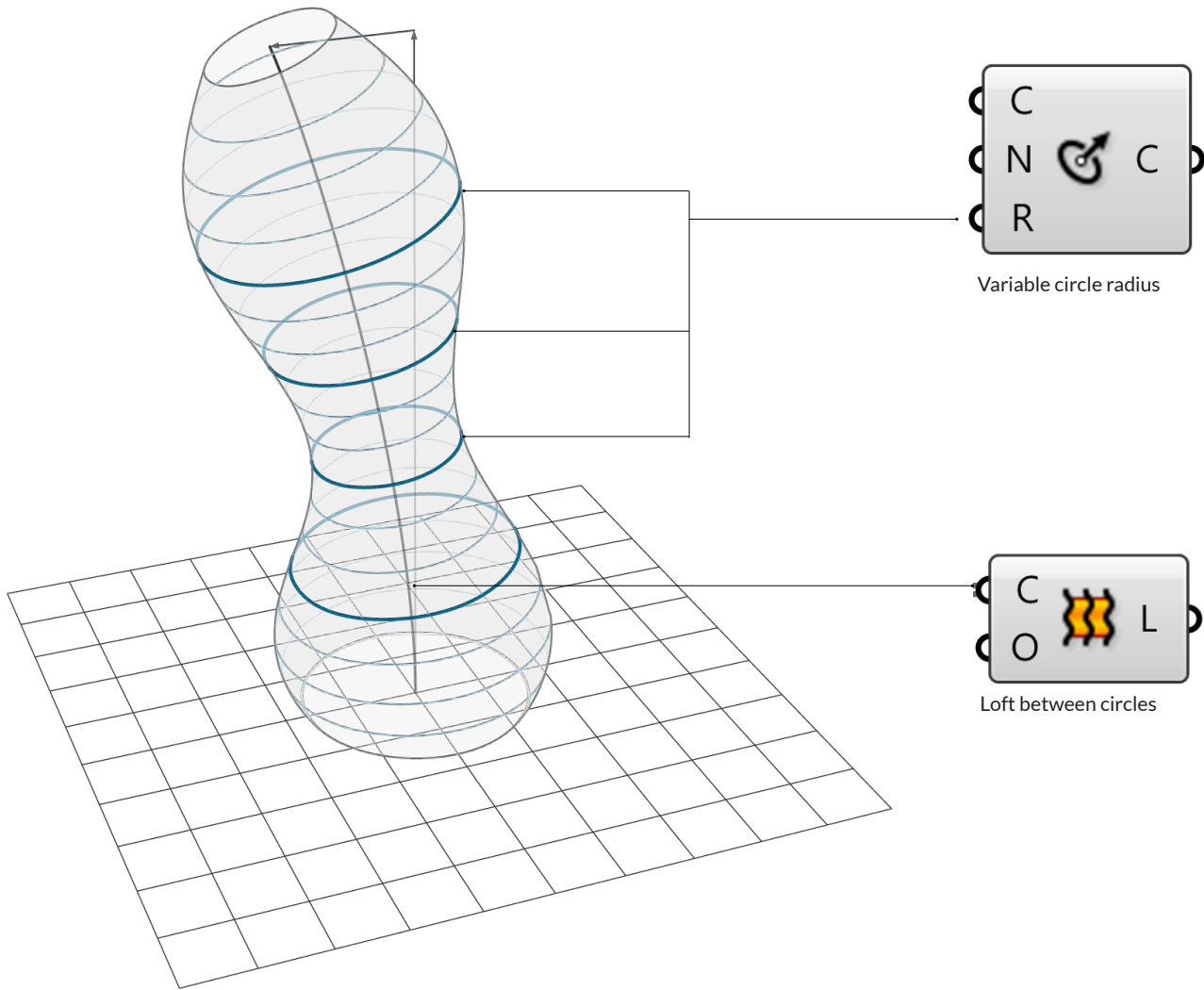
Grasshopper visual programs are executed from left to right. Reading the graph relative to the wired connections from upstream to downstream provides understanding about the Program Flow.



F.1.4.1 THE LOGICAL GRAPH

All of the objects and the wires connecting the objects represent the logical graph of our program. This graph reveals the flow of data, dependencies of any input to its wired output. Any time our graph changes, sometimes referred to as being "dirtied," every downstream connection and object will be updated.





F.2 BUILDING BLOCKS OF ALGORITHMS

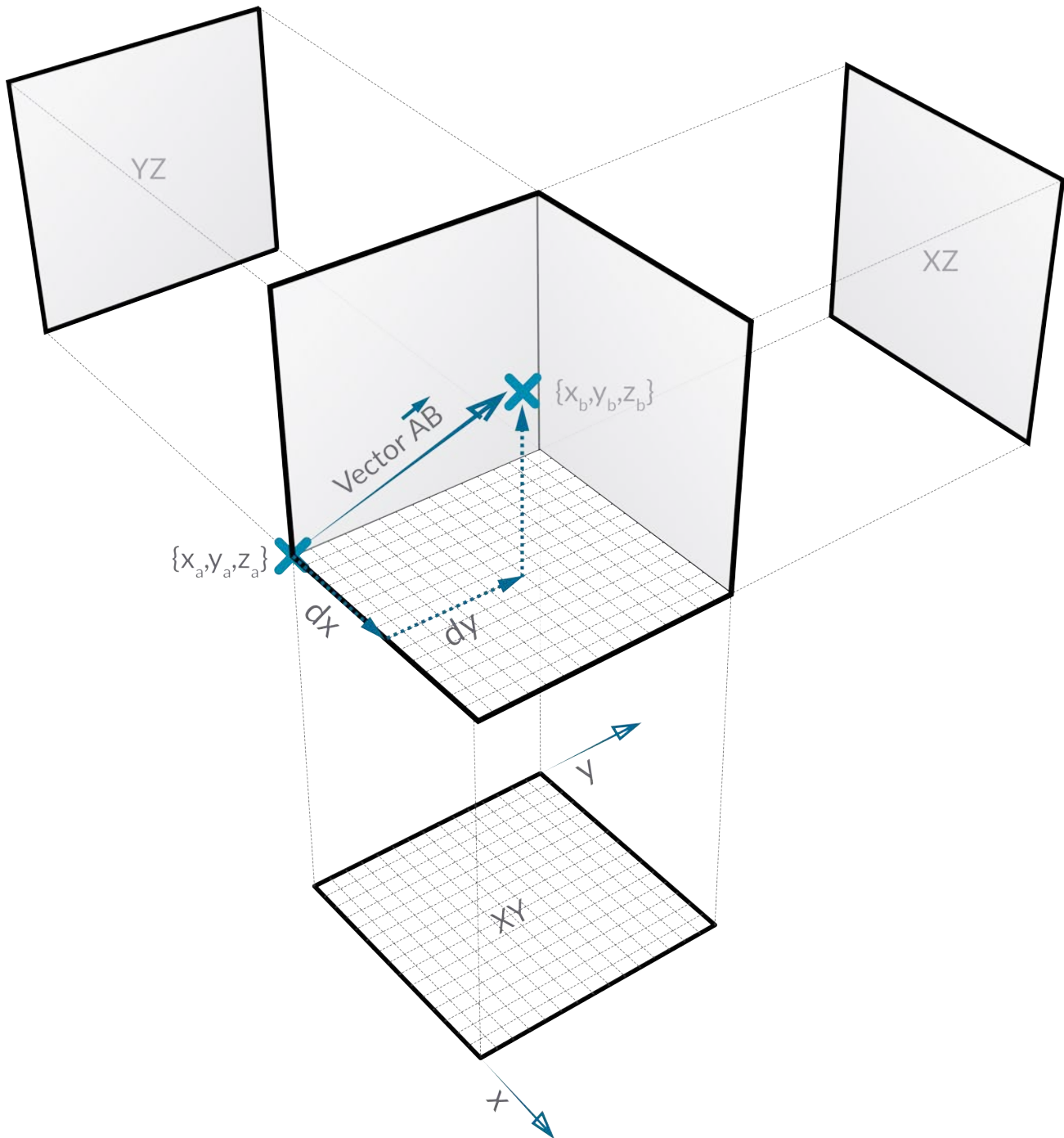
This chapter will introduce you to basic geometric and mathematical concepts and how they are implemented and manipulated in Grasshopper.



F.2.0

Points, Planes & Vectors

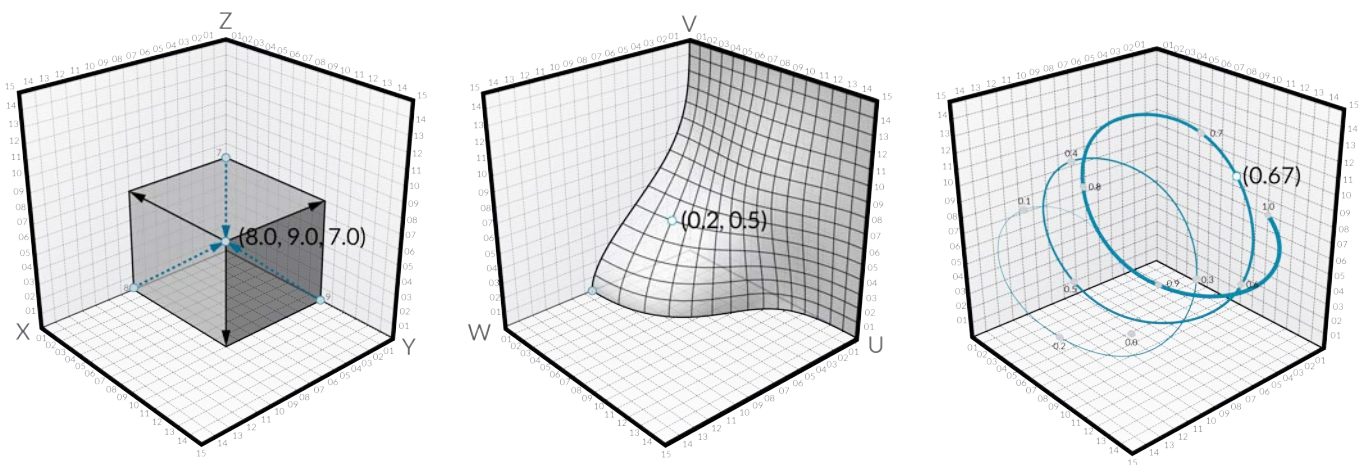
Everything begins with points. A point is nothing more than one or more values called coordinates. The number of coordinate values corresponds with the number of dimensions of the space in which it resides. Points, planes, and vectors are the base for creating and transforming geometry in Grasshopper.



F.2.0.0 POINTS

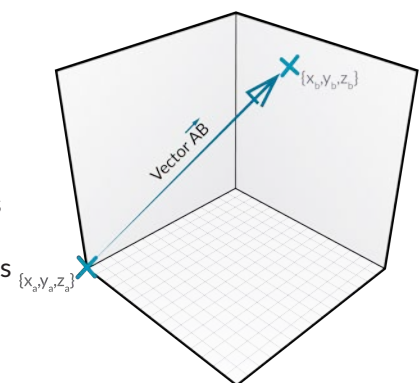
Points in 3D space have three coordinates, usually referred to as $[x,y,z]$. Points in 2D space have only two coordinates which are either called $[x,y]$ or $[u,v]$ depending on what kind of two dimensional space we're talking about. 2D parameter space is bound to a finite surface. It is still continuous, i.e. hypothetically there are an infinite amount of points on the surface, but the maximum distance between any of these points is very much limited. 2D parameter coordinates are only valid if they do not exceed a certain range. In the example drawing, the range has been set between 0.0 and 1.0 for both $[u]$ and $[v]$ directions, but it could be any finite domain. A point with coordinates $[1.5, 0.6]$ would be somewhere outside the surface and thus invalid. Since the surface which defines this particular parameter space resides in regular 3D world space, we can always translate a parametric coordinate into a 3D world coordinate. The point $[0.2, 0.5]$ on the surface for example is the same as point $[1.8, 2.0, 4.1]$ in world coordinates. Once we transform or deform the surface, the 3D coordinates which correspond with $[0.2, 0.5]$ will change.

If this is a hard concept to grasp, it might help to think of yourself and your position in space. We tend to use local coordinate systems to describe our whereabouts; "I'm sitting in the third seat on the seventh row in the movie theatre", "I'm in the back seat". If the car you're in is on the road, your position in global coordinates is changing all the time, even though you remain in the same back seat 'coordinate'.



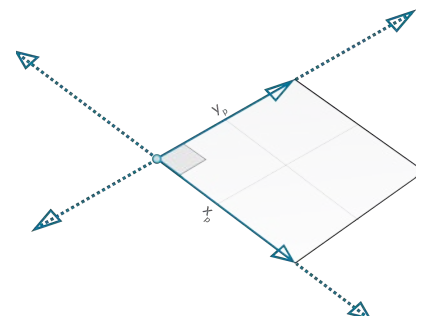
F.2.0.1 VECTORS

A vector is a geometric quantity describing Direction and Magnitude. Vectors are abstract; i.e. they represent a quantity, not a geometrical element. Vectors are indistinguishable from points. That is, they are both lists of three numbers so there's absolutely no way of telling whether a certain list represents a point or a vector. There is a practical difference though; points are absolute, vectors are relative. When we treat a list of three doubles as a point it represents a certain coordinate in space, when we treat it as a vector it represents a certain direction. A vector is an arrow in space which always starts at the world origin $(0,0,0,0)$ and ends at the specified coordinate.



F.2.0.2 PLANES

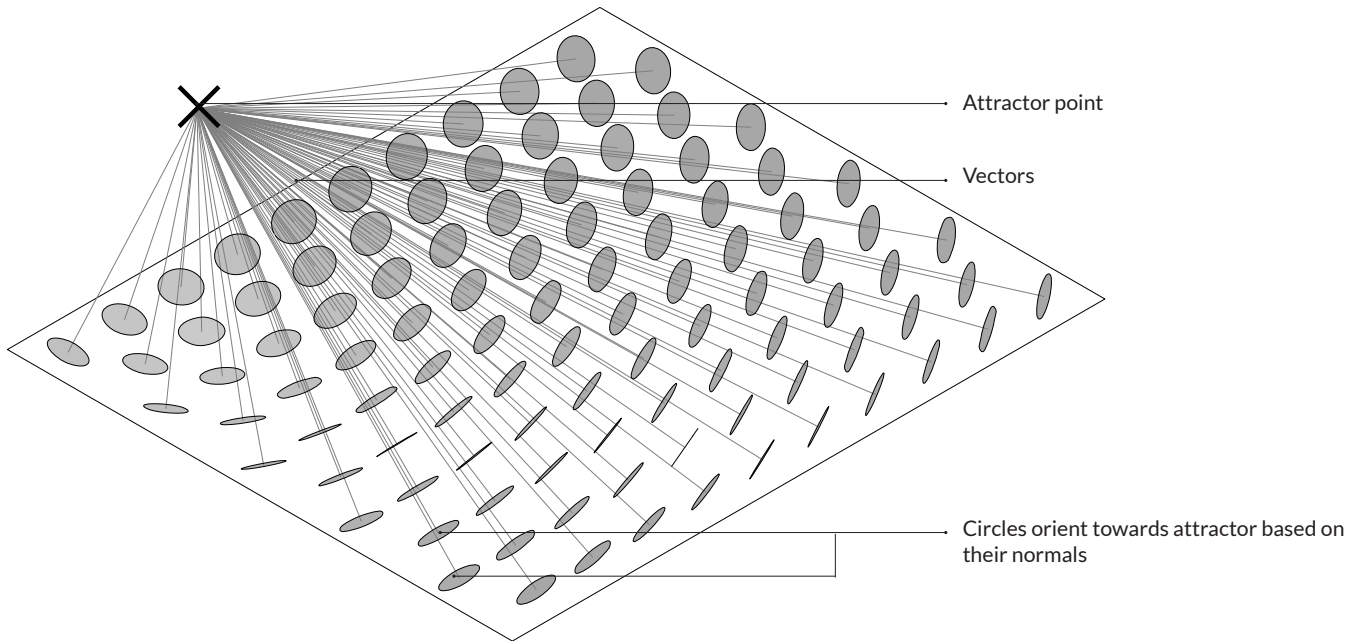
Planes are "Flat" and extend infinitely in two directions, defining a local coordinate system. Planes are not genuine objects in Rhino, they are used to define a coordinate system in 3D world space. In fact, it's best to think of planes as vectors, they are merely mathematical constructs.



F.2.1

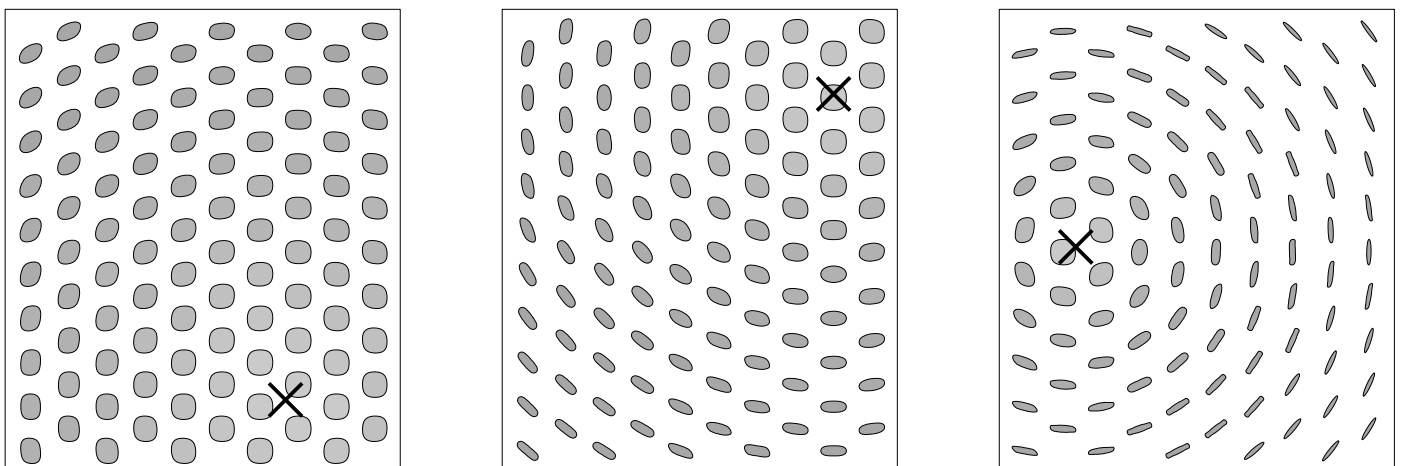
Working with Attractors

Attractors are points that act like virtual magnets - either attracting or repelling other objects. In Grasshopper, any geometry referenced from Rhino or created within Grasshopper can be used as an attractor. Attractors can influence any number of parameters of surrounding objects including scale, rotation, color, and position. These parameters are changed based on their relationship to the attractor geometry.



In the image above, vectors are drawn between an attractor point and the center point of each circle. These vectors are used to define the orientation of the circles so they are always facing the attractor point.

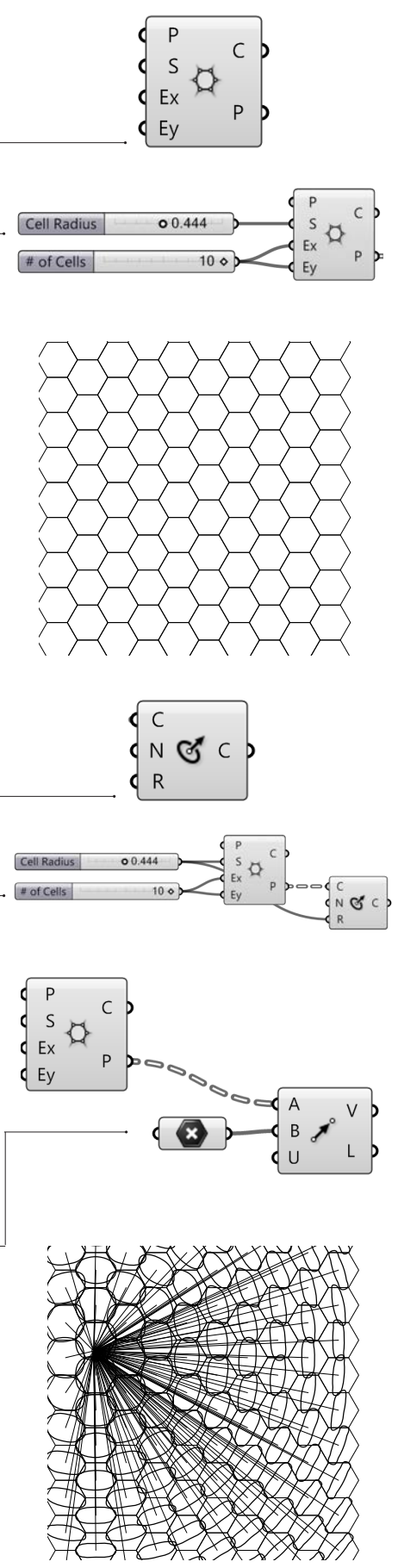
This same attractor could be used to change other parameters of the circles. For example, circles that are closest to the attractor could be scaled larger by using the length of each vector to scale the radius of each circle.



F.2.1.0 ATTRACTOR DEFINITION

In this example, we will use an attractor point to orient a grid of circles, based on the vectors between the center points of the circles and the attractor point. Each circle will orient such that it is normal to (facing) the attractor point.

01. Type Ctrl+N in Grasshopper to start a new definition
02. Vector/Grid/Hexagonal - Drag and drop the Hexagonal Grid component onto the canvas
03. Params/Input/Slider - Drag and drop two Numeric Sliders on the canvas
04. Double-click on the first slider and set the following:
 05. Name: Cell Radius
 06. Rounding: Floating Point
 07. Lower Limit: 0.000
 08. Upper Limit: 1.000
 09. Value: 0.500
10. Double-click on the second slider and set the following:
 11. Name: # of Cells
 12. Rounding: Integers
 13. Lower Limit: 0
 14. Upper Limit: 10
 15. Value: 10
16. Connect the Number Slider (Cell Radius) to the Size (S) input of the Hexagonal Grid component
17. Connect the Number Slider (# of Cells) to the Extent X (Ex) input and the Extent Y (Ey) input of the Hexagonal Grid component
18. Curve/Primitive/Circle CNR - Drag and drop a Circle CNR component onto the canvas
19. Connect the Points (P) output of the Hexagonal Grid to the Center (C) input of the Circle CNR component
20. Connect the Number Slider (Cell Radius) to the Radius (R) input of the Circle CNR component.
21. Vector/Vector/Vector 2Pt - Drag and Drop the component Vector 2Pt
22. Connect the Points output (P) of the Hexagonal Grid component to the Base Point (A) input of the Vector 2Pt component.
23. Params/Geometry/Point - Drag and Drop the Point component onto the canvas
24. Right-Click the Point component and select set one point. In the model space select where you would like the attractor point to be
25. Connect the Point component to the Tip Point (B) input of the Vector 2Pt component
26. Connect the Vector (V) output of the Vector 2Pt to the Normal (N) input of the Circle CNR component.
27. Curve/Util/Offset - Drag and Drop the Offset Component onto the canvas.
28. Params/Input/Slider - Drag and drop a Numeric Slider on the canvas
29. Double-click on the slider and set the following:
 30. Name: Offset Distance
 31. Rounding: Floating Point



32. Lower Limit: - 0.500

33. Upper Limit: 0.500

34. Value: -0.250

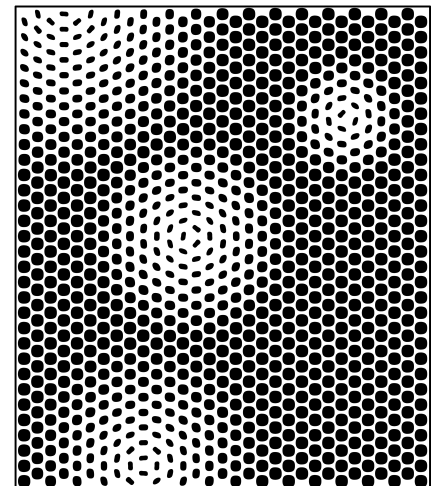
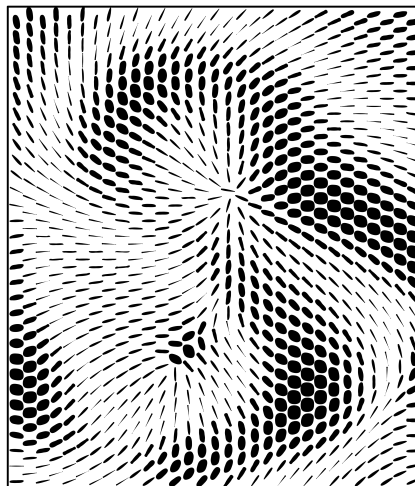
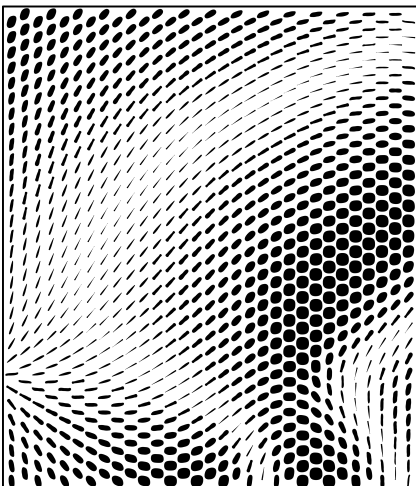
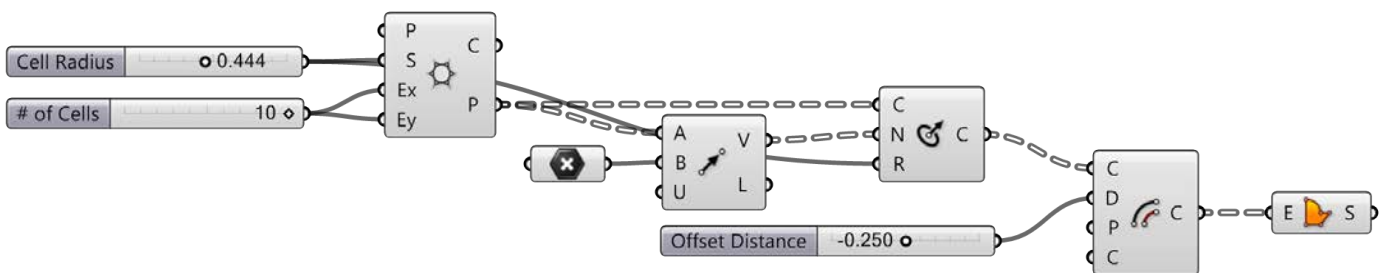
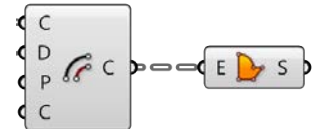
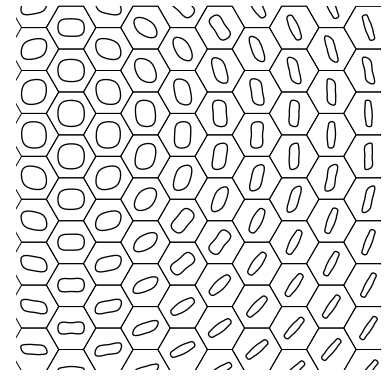
35. Connect the Number Slider (Offset Distance) to the Distance (D) input of the offset component

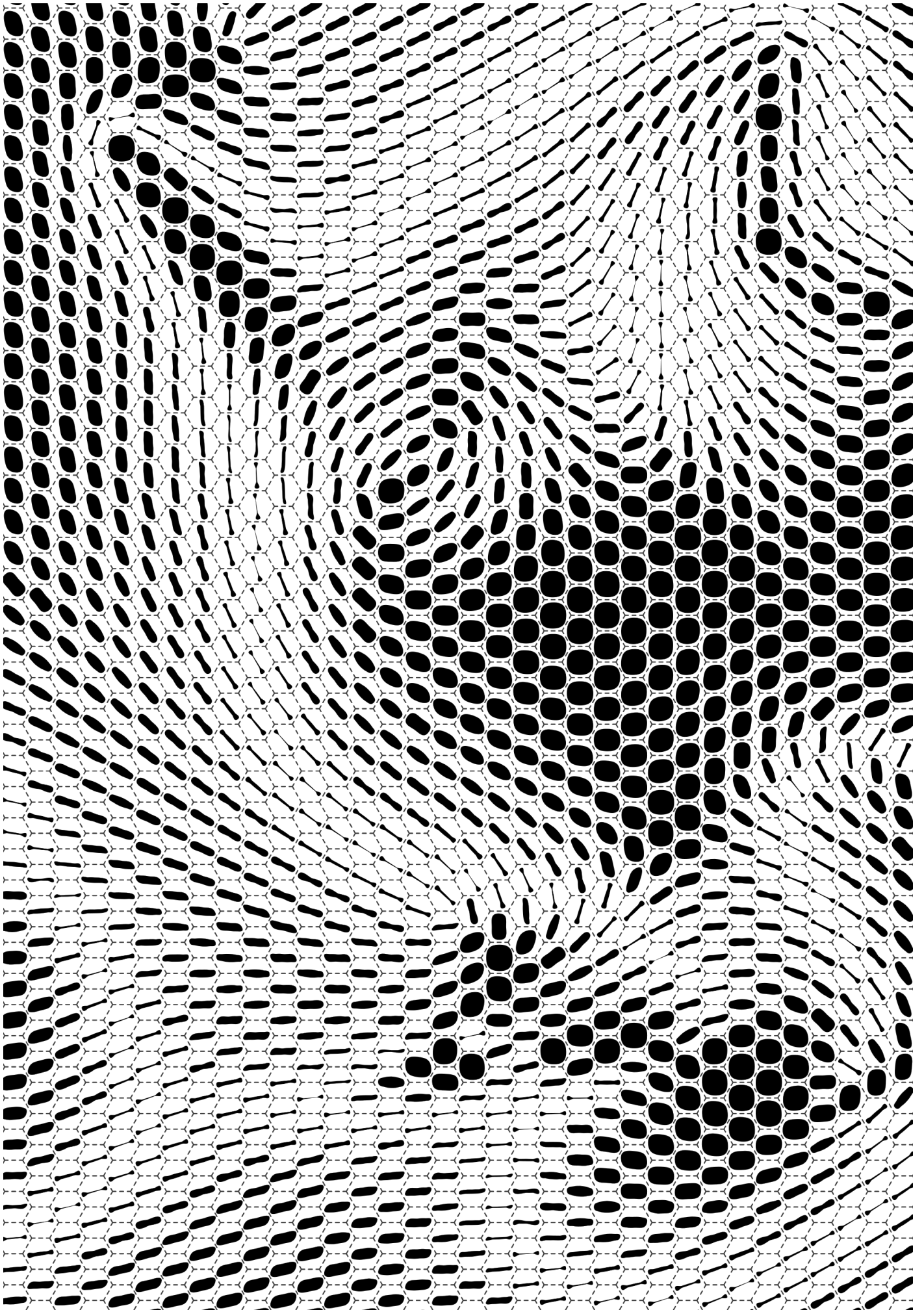
36. Surface/Freeform/Boundary Surfaces – Drag and drop Boundary Surfaces on to the canvas

37. Connect the Curves (C) output of the offset component to the Edges (E) input of the Boundary Surfaces.

38. Params/Geometry/Surface – Drag and Drop the Surface Component onto the canvas

39. Connect the Surfaces (S) output from the Boundary Surfaces to the Surfaces Parameter





F.2.2

Mathematics, Expressions & Conditionals

Knowing how to work with numeric information is an essential skill to master as you learn to use Grasshopper. Grasshopper contains many components to perform mathematical operations, evaluate conditions and manipulate sets of numbers.

In mathematics, numbers are organized by sets and there are two that you are probably familiar with:

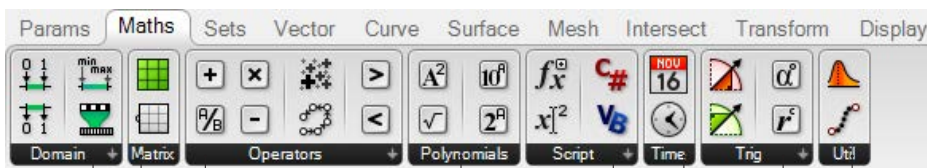
Integer Numbers: [..., -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, ...]

Real Numbers: [∞ , ..., -4.8, -3.6, -2.4, -1.2, 0.0, 1.234, e, 3.0, 4.0, ..., ∞]

While there are other types of number sets, these two interest us the most because Grasshopper uses these extensively. Although there are limitations in representing these sets exactly in a digital environment, we can approximate them to a high level of precision. In addition, it should be understood that the distinction between Integral types (integers) and Floating types (real numbers) corresponds to the distinction between discrete and continuous domains. In this chapter, we're going to explore different methods for working with and evaluating various sets of numbers.

F.2.2.0 THE MATH TAB

Most of the components that deal with mathematical operations and functions can be found under the following sub-categories of the Math tab:



Polynomials are one of the most important concepts in algebra and throughout mathematics and science. You can use the components found in this subcategory to compute factorials, logarithms, or to raise a number to the nth power.

Operators are used to perform mathematical operations such as Addition, Subtraction, Multiplication, etc. Conditional operators allow you to determine whether a set of numbers are larger than, less than, or similar to another set of numbers.

In mathematics, a matrix is an array of numbers organized in rows and columns. This subcategory contains a series of utility tools to construct and modify matrices.

Domains are used to define a range of values (formerly known as intervals) between two numbers. The components under the Domain tab allow you to create or decompose different domain types.

The utility subcategory is a 'grab bag' of useful components that can be used in various mathematical equations. Check here if you're trying to find the maximum or minimum values between two lists of numbers; or average a group of numbers.

The time subcategory has a number of components which allow you to construct instances of dates and times.

These components allow you to solve trigonometric functions such as Sine, Cosine, Tangent, etc.

The script subcategory contains single and multi-variable expressions as well as the VB.NET and C# scripting components.

F.2.2.1 OPERATORS

As was previously mentioned, Operators are a set of components that use algebraic functions with two numeric input values, which result in one output value.

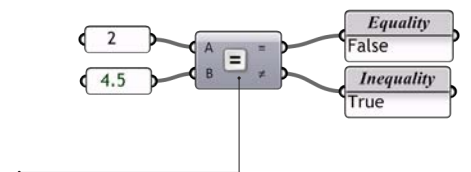
Most of the time, you will use the Math Operators to perform arithmetical actions on a set of numbers. However, these operators can also be used on various data types, including points and vectors.



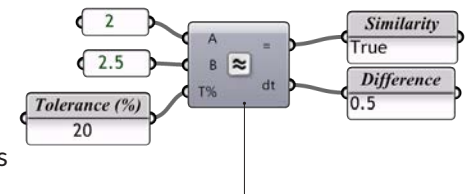
F.2.2.2 CONDITIONAL OPERATORS

Almost every programming language has a method for evaluating conditional statements. In most cases the programmer creates a piece of code to ask a simple question of “what if.” What if the area of a floor outline exceeds the programmatic requirements? Or, what if the curvature of my roof exceeds a realistic amount? These are important questions that represent a higher level of abstract thought. Computer programs have the ability to analyze “what if” questions and take actions depending on the answer to that question. Let’s take a look at a very simple conditional statement that a program might interpret: If the object is a curve, delete it. The piece of code first looks at an object and determines a single boolean value for whether or not it is a curve. There is no middle ground. The boolean value is True if the object is a curve, or False if the object is not a curve. The second part of the statement performs an action dependent on the outcome of the conditional statement; in this case, if the object is a curve then delete it. This conditional statement is called an If statement. There are four conditional operators (found under the Math/ Operators subcategory) that evaluate a condition and return a boolean value.

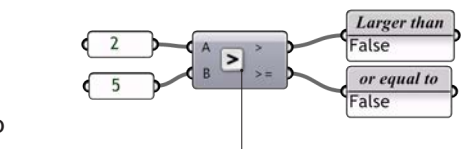
The Equality component takes two lists and compares the first item of List A and compares it to the first item of List B. If the two values are the same, then a True boolean value is created; conversely if the two values are not equal, then a False boolean value is created. The component cycles through the lists according to the set data matching algorithm (default is set to Longest List). There are two outputs for this component. The first returns a list of boolean values that shows which of the values in the list were equal to one another. The second output returns a list that shows which values were not equal to one another - or a list that is inverted from the first output.



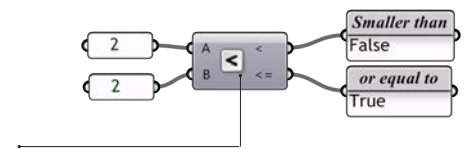
The Similarity component evaluates two lists of data and tests for similarity between two numbers. It is almost identical to the way the Equality component compares the two lists, with one exception: it has a percentage input that defines the ratio of list A that list B is allowed to deviate before inequality is assumed. The Similarity component also has an output that determines the absolute value distance between the two input lists.



The Larger Than component will take two lists of data and determine if the first item of List A is greater than the first item of List B. The two outputs allow you to determine if you would like to evaluate the two lists according to a greater than (>) or greater than and equal to (>=) condition.

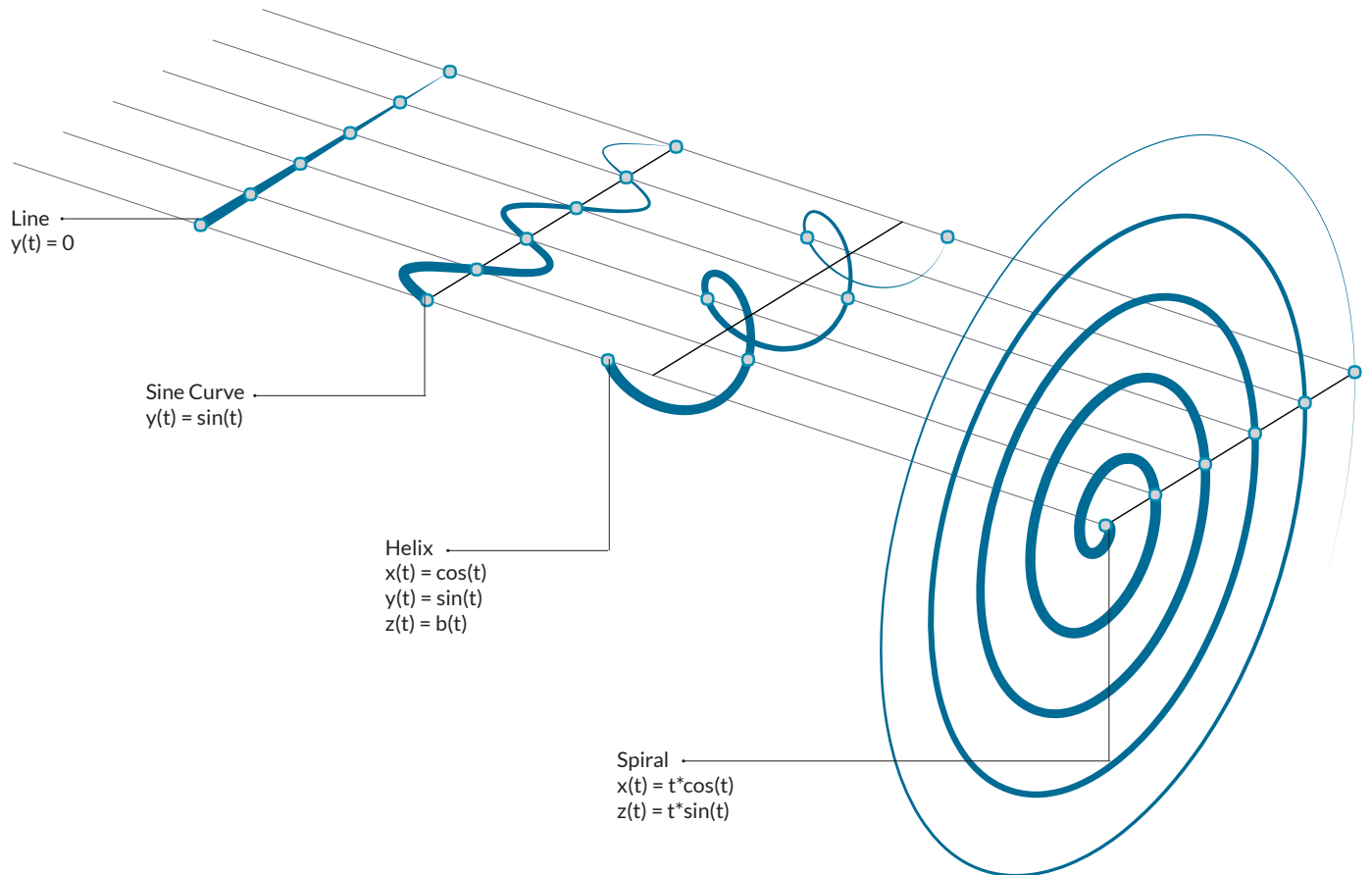


The Smaller Than component performs the opposite action of the Larger Than component. The Smaller Than component determines if list A is less than list B and returns a list of boolean values. Similarly, the two outputs let you determine if you would like to evaluate each list according to a less than (<) or less than and equal to (<=) condition.



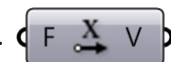
F.2.2.3 TRIGONOMETRY COMPONENTS

We have already shown that we can use an Expression (or Evaluate) component to evaluate conditional statements as well as compute algebraic equations. However, there other ways to calculate simple expressions using a few of the built in Trigonometry functions. We can use these functions to define periodic phenomena like sinusoidal wave forms such as ocean waves, sound waves, and light waves.



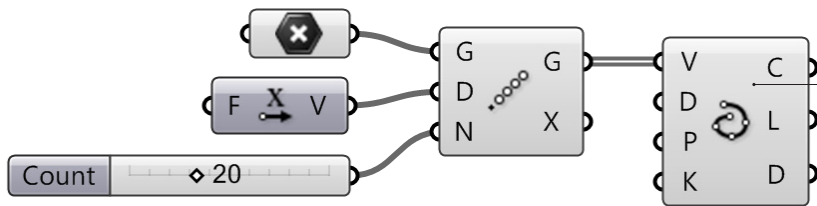
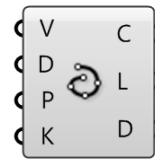
In this example, we will use Grasshopper to construct various trigonometric curves using trigonometry function components found in the Math tab:

01. Type Ctrl+N (in Grasshopper) to start a new definition
02. Params/Geometry/Point - Drag and drop a point parameter onto the canvas
03. Right click the Point parameter and click Set One Point - select a point in the Rhino viewport
04. Vector/Vector/Unit X - Drag and drop the Unit X component to the canvas
05. Params/Input/Number Slider - Drag and drop the Number Slider component onto the canvas
06. Double-click on the Number slider and set the following:
 - Rounding: Integer
 - Lower Limit: 10
 - Upper Limit: 40
 - Value: 20
07. Transform/Array/Linear Array - Drag and drop the Linear Array component onto the canvas



08. Connect the output of the Point parameter to the Geometry (G) input of the Linear Array component
09. Connect the Unit Vector (V) output of the Unit X component to the Direction (D) input of the Linear Array component
10. Connect the Number Slider output to the Count (N) input of the Linear Array Component
11. Curve/Spline/Interpolate – Drag and drop the Interpolate Curve component to the canvas
12. Connect the Geometry (G) output of the Linear Array component to the Vertices (V) input of the Interpolate Curve component

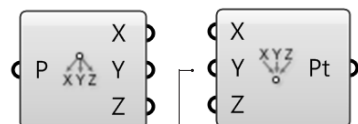
You should see a line of 20 points along the x axis in Rhino. Adjust the slider to change the number of points in the array.



We have just connected the array of points with a curve

We have just created a line by connecting an array of points with a curve. Let's try using some of Grasshopper's Trigonometry components to alter this curve:

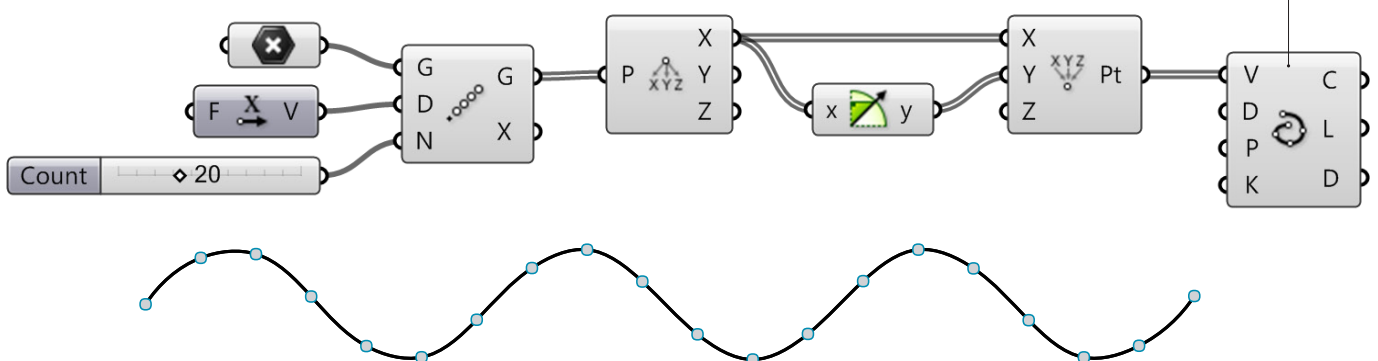
13. Vector/Point/Deconstruct – Drag and drop a Deconstruct component onto the canvas
14. Vector/Point/Construct Point - Drag and drop a Construct Point component onto the canvas
15. Maths/Trig/Sine - Drag and drop a sine component onto the canvas
16. Disconnect the wire from the Vertices (V) input of the Interpolate Curve component
17. Connect the Geometry (G) output of the Linear Array component to the Point (P) input of the Deconstruct component
18. Connect the Point X (X) output of the Deconstruct component to the X coordinate (X) input of the Construct Point Component
19. Connect a second wire from the Point X (X) output of the Deconstruct Component to the Value (x) input of the Sine component
20. Connect the Result (y) output of the Sine component to the Y coordinate (Y) input of the Construct Point component
21. Connect the Point (Pt) output of the Construct Point component to the Vertices (V) input of the Interpolate component



You can disconnect wires by holding down control and dragging, or by right-clicking the input and selecting Disconnect

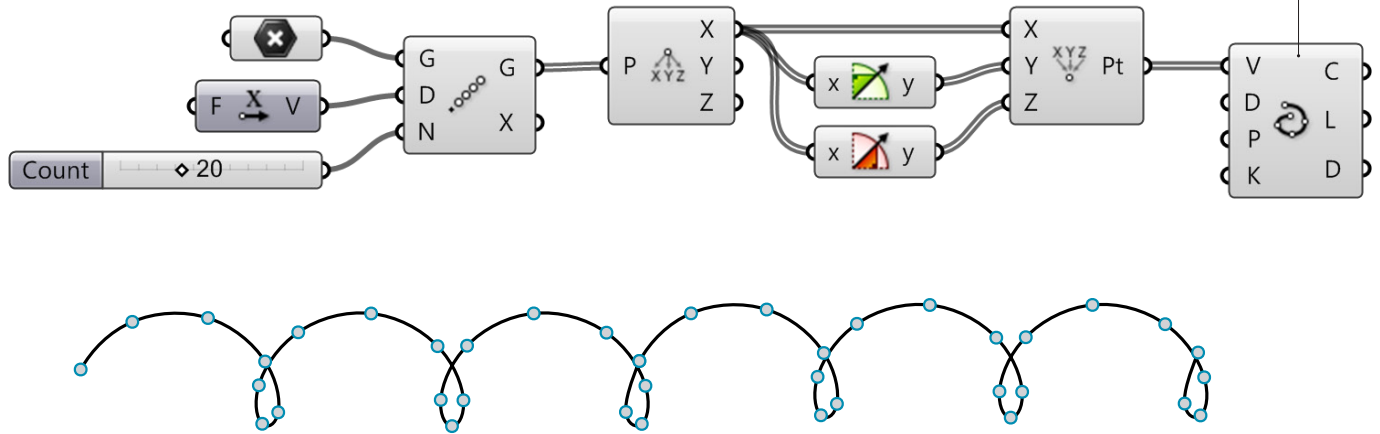
We have now reconstructed our points with the same X values, modifying the y values with a sine curve

You should now see a sine wave curve along the X axis in Rhino

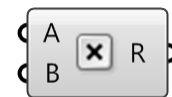


22. Maths/Trig/Cosine – Drag and drop a Cosine component to the canvas
23. Connect a third wire from the Point X (X) output of the Deconstruct Component to the Value (x) input of the Cosine component
24. Connect the Result (y) output of the Cosine component to the Z coordinate (Z) input of the Construct Point component

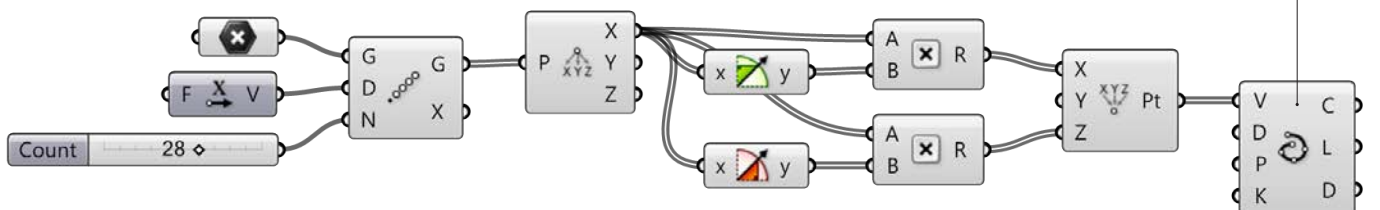
We have now created a 3D helix



25. Maths/Operators/Multiplication – Drag and drop two Multiplication components onto the canvas
26. Connect wires from the Point X (X) output of the Deconstruct component to the (A) input of each Multiplication component
27. Connect the Result (y) output of the Sine component to the (B) input of the first Multiplication component
28. Connect the Result (y) output of the Cosine component to the (B) input of the second Multiplication component
29. Disconnect the wire from the Y Coordinate (Y) input of the Construct Point component
30. Connect the Result (R) output of the first Multiplication component to the X Coordinate (X) input of the Construct Point component
31. Connect the Result (R) output of the second Multiplication component to the Z Coordinate (Z) input of the Construct Point component

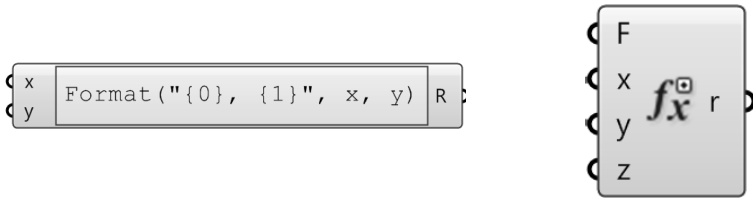


You should now see a spiral curve



F.2.2.4 EXPRESSIONS

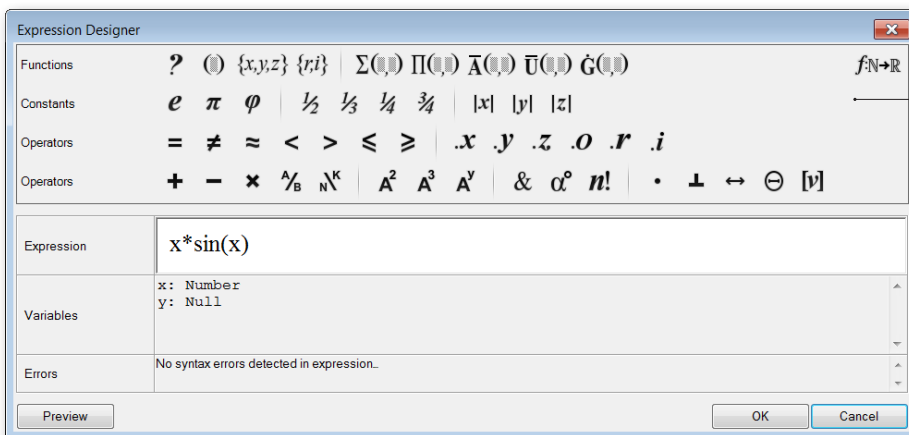
The Expression component (and its brother the Evaluate component) are very flexible tools; that is to say that they can be used for a variety of different applications. We can use an Expression (or Evaluate component) to solve mathematical algorithms and return numeric data as the output.



In the following example, we will look at mathematical spirals found in nature and how we can use a few Functions components to create similar patterns in Grasshopper. We will build on our trigonometric curves definition as a starting point.

01. Open your Trigonometric curves Grasshopper definition from the previous example
02. Delete the Sine, Cosine, Multiplication, and Interpolate components
03. Params/Input/Number Slider – Drag and drop a Number Slider onto the canvas
04. Double-click on the Number slider and set the following:
 Rounding: Float
 Lower Limit: 0.000
 Upper Limit: 1.000
 Value: 1.000
05. Connect the Number slider to the Factor (F) input of the Unit X component
06. Maths/Script/Expression – Drag two Expression components onto the canvas
07. Double-click the first Expression component to open the Expression Editor and change the expression to:
 $x * \sin(x)$
08. Double-click the second Expression component to open the Expression Editor and change the expression to:

This slider allows you to adjust the distance between the points in the array

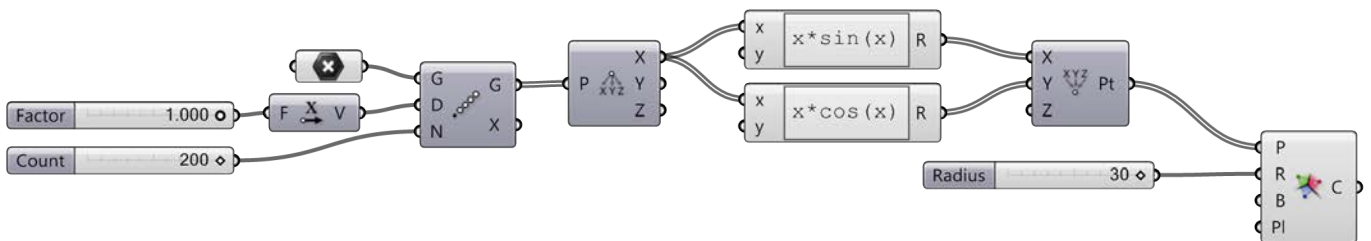
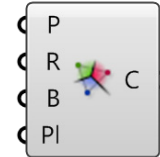


Double click the Expression component to open the Grasshopper Expression Editor

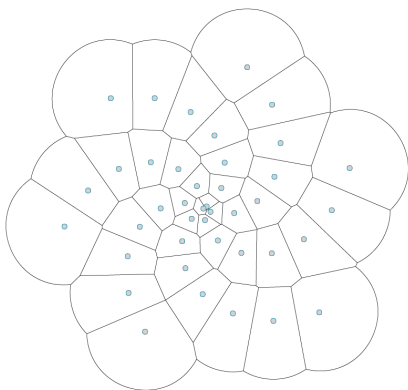
$$x \cdot \cos(x)$$

09. Connect two wires from the Point X (X) output of the Deconstruct component to the Variable x (x) input of each Expression component
10. Connect the Result (R) output of the first Expression component to the X coordinate (X) input of the Construct Point component
11. Connect the Result (R) output of the second Expression component to the Y coordinate (Y) input of the Construct Point component
12. Mesh/Triangulation/Voronoi – Drag and drop the Voronoi component onto the canvas
13. Params/Input/Number Slider – Drag and drop a Number Slider onto the canvas
14. Double-click on the Number slider and set the following:
 - Rounding: Integer
 - Lower Limit: 1
 - Upper Limit: 30
 - Value: 30
15. Connect the Number slider to the Radius (R) input of the Voronoi component
16. Connect the Point (Pt) output of the Construct Point component to the Points (P) input of the Voronoi component

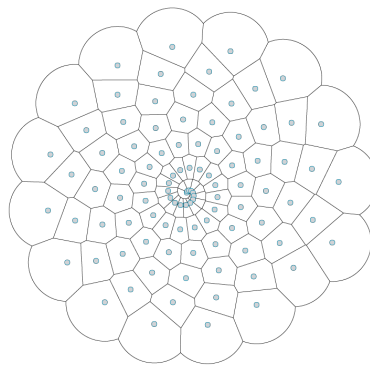
We have replaced the Trigonometry functions and multiplication operators with the expression component for a more efficient definition



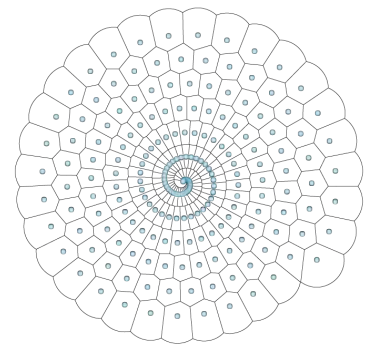
You can create different Voronoi patterns by manipulating the Factor, Count, and Radius sliders. Below are three examples:



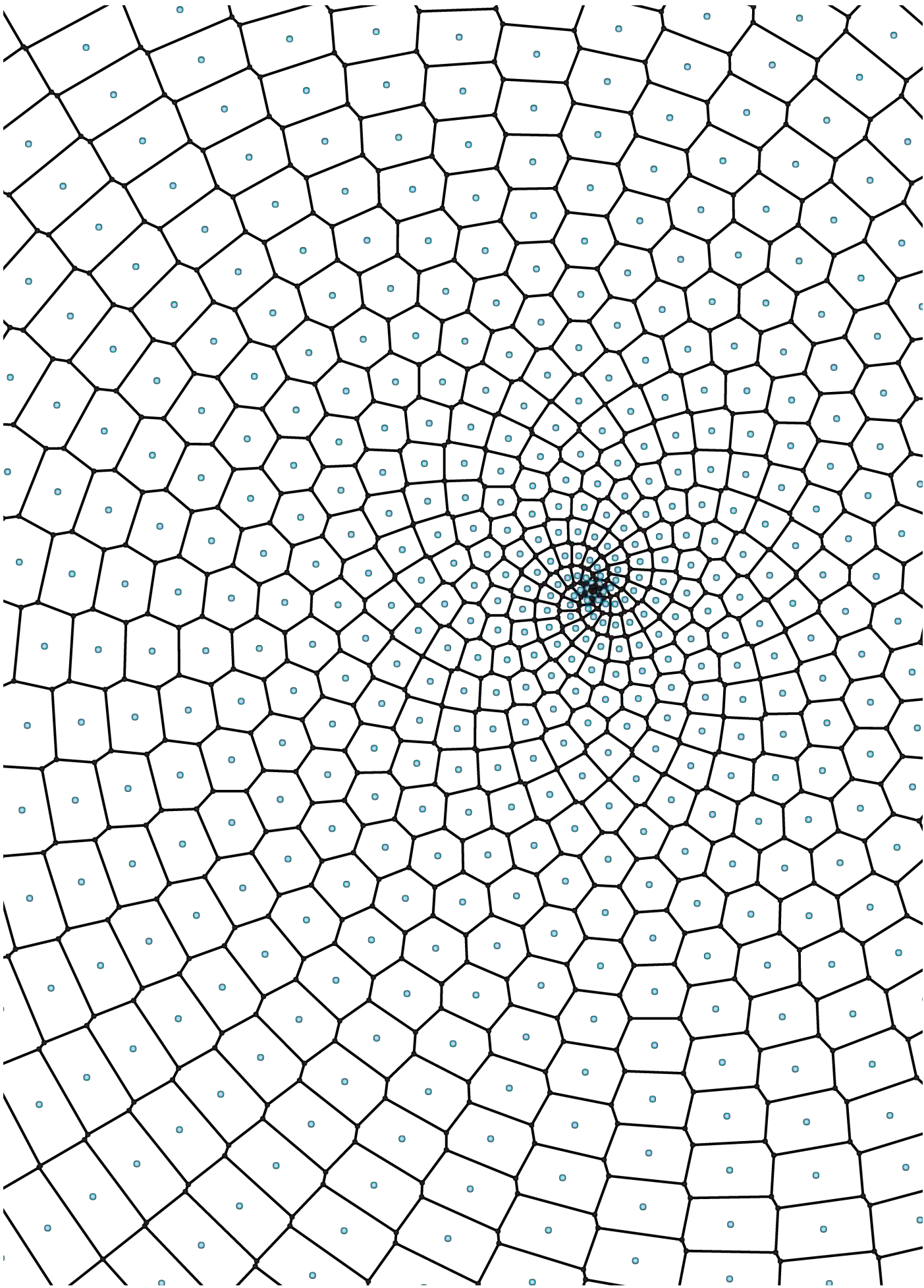
Factor = 1.000
Radius = 15



Factor = 0.400
Radius = 10



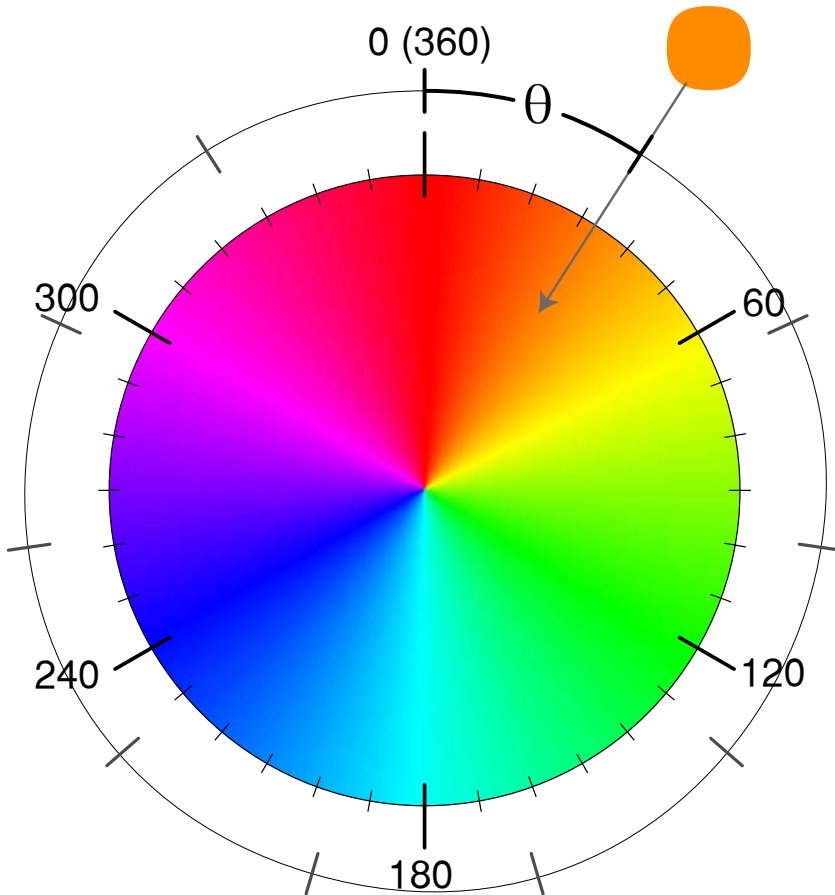
Factor = 0.200
Radius = 7



F.2.3

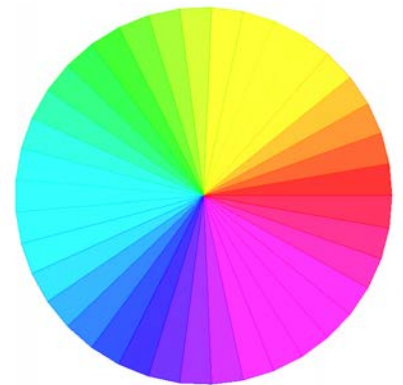
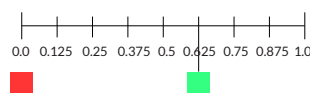
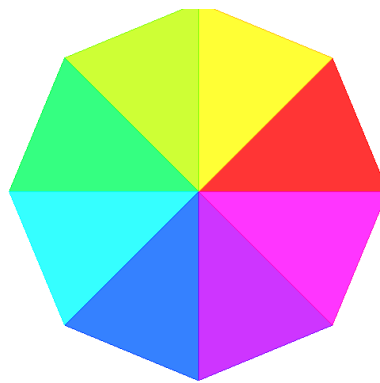
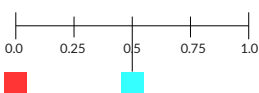
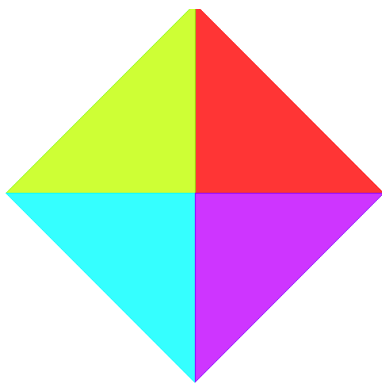
Domains & Color

The color wheel is a model for organizing colors based on their hue. In Grasshopper, colors can be defined by their hue value in a range of 0.0 to 1.0. Domains are used to define a range of all possible values between a set of numbers between a lower limit (A) and an upper limit (B).



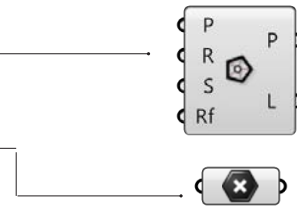
In the color wheel, hue corresponds to angle. Grasshopper has taken this 0-360 domain and remapped it between zero and one.

By dividing the Hue domain (0.0 to 1.0) by the number of segments desired, we can assign a hue value to each segment to create a color wheel.

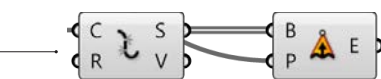
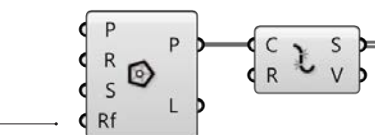
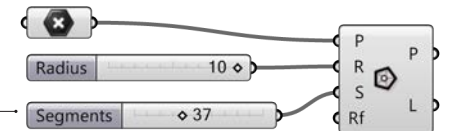


In this example, we will use Grasshopper's domain and color components to create a color wheel with a variable amount of segments.

01. Type Ctrl+N (in Grasshopper) to start a new definition
02. Curve/Primitive/Polygon – Drag and drop a polygon component onto the canvas
03. Params/Geometry/Point – Drag and drop a Point Parameter onto the canvas
04. Right-Click on the Point Component and select set one point
05. Set a point in the model space.
06. Connect the Point Parameter (Base Point) to the Plane (P) input of the polygon component
07. Params/Input/Number Sliders – Drag and drop two number sliders onto the canvas
08. Double-click on the first slider and set the following:
Rounding: Integers
Lower Limit: 0
Upper Limit: 10
Value: 10
09. Double-click on the second slider and set the following:
Rounding: Integers
Lower Limit: 0
Upper Limit: 100
Value: 37
10. Connect the Number Slider (Radius) to the Radius (R) input of the Polygon component
11. Connect the Number Slider (Segments) to the Segments (S) input of the Polygon component
12. Curve/Util/Explode – Drag and drop an Explode component onto the canvas.
13. Connect the Polygon (P) output of the Polygon component to the Curve (C) input of the Explode component
14. Surface/Freeform/Extrude Point – Drag and drop the Extrude Point component onto the canvas
15. Connect the Segments (S) output of the Explode component to the Base (B) input of the Extrude Point
16. Connect the Point Parameter (Base Point) to the Extrusion Tip (P)
17. Surface/Analysis/Deconstruct Brep – Drag and drop the Deconstruct Brep component on to the canvas
18. Connect the Extrusion (E) output of the Extrude Point component to the Deconstruct Brep (B) component
19. Maths/Domain/Divide Domain – Drag and drop the Divide Domain component
20. Connect the Number Slider (Segments) to the Count (C) input of the Divide Domain component
21. Math/Domain/Deconstruct Domain – Drag and drop the Deconstruct Domain component

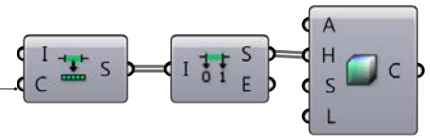


Note: When you connect a number slider to a component it will automatically change its name to the name of input that it is connecting to.

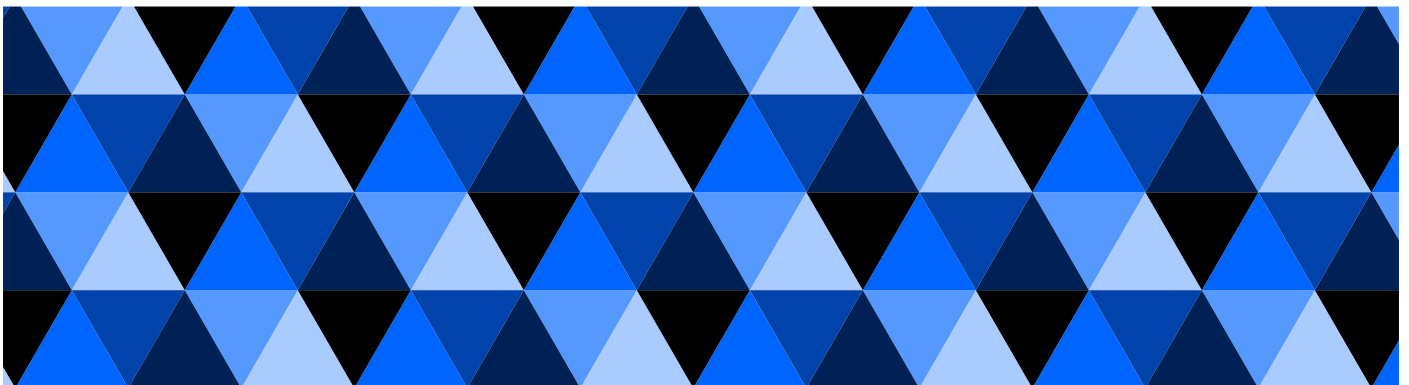
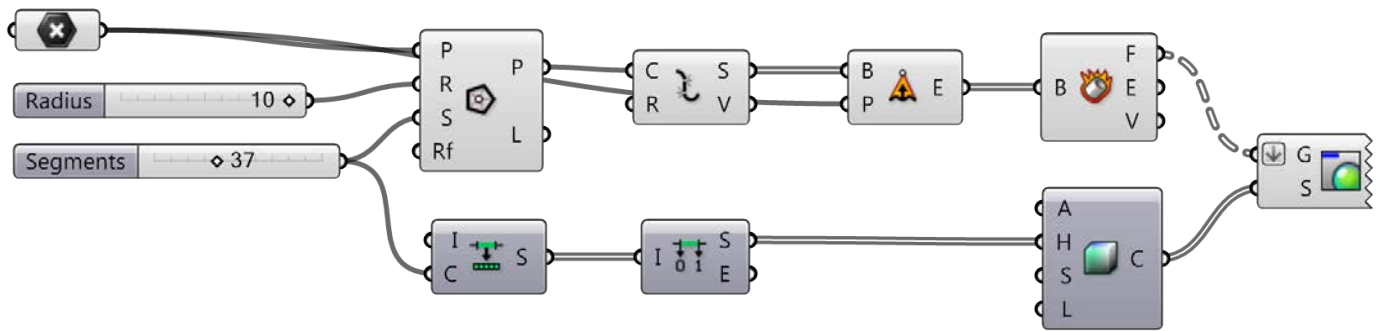


22. Connect the Segments (S) output of the Divide Domain component to the Domain (I) input of the Deconstruct Domain component
23. Display/Colour/Colour HSL – Drag and drop the Colour HSL component
24. Connect the Start (S) output of the Deconstruct Domain component to the Hue (H) input of the Colour HSL components
25. Display/Preview/Custom Preview – Drag and drop the Custom Preview component
26. Right click on the Geometry (G) input of the Custom Preview component and select Flatten
27. Connect the Faces (F) output of the Deconstruct Brep component to the Geometry (G) input of the Custom Preview
28. Connect the Colour (C) output of the Colour HSL component to the Shade (S) input of the Custom Preview Components

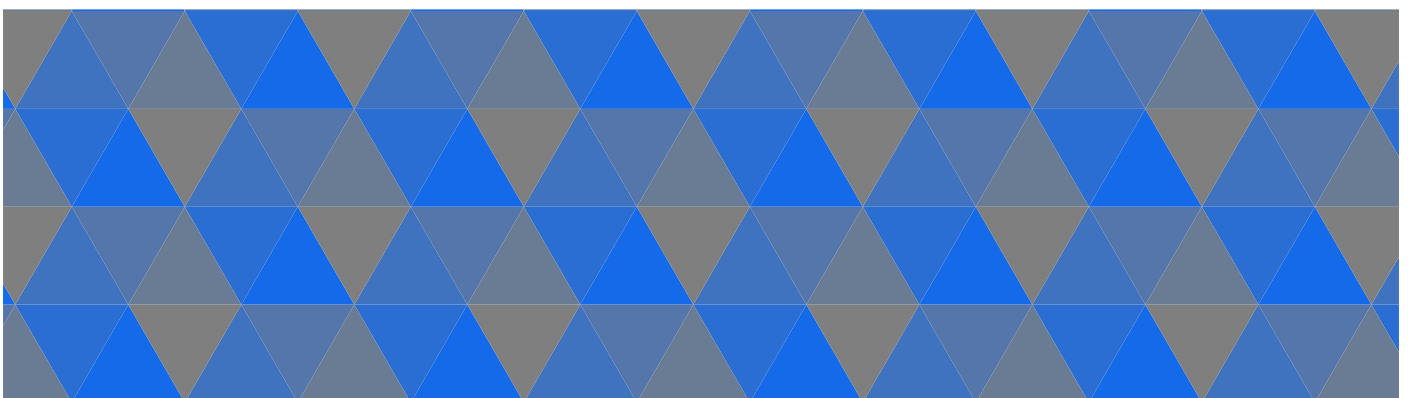
Note: The Base Domain (I) is automatically set between 0.0 -1.0 which is what we need for this exercise.

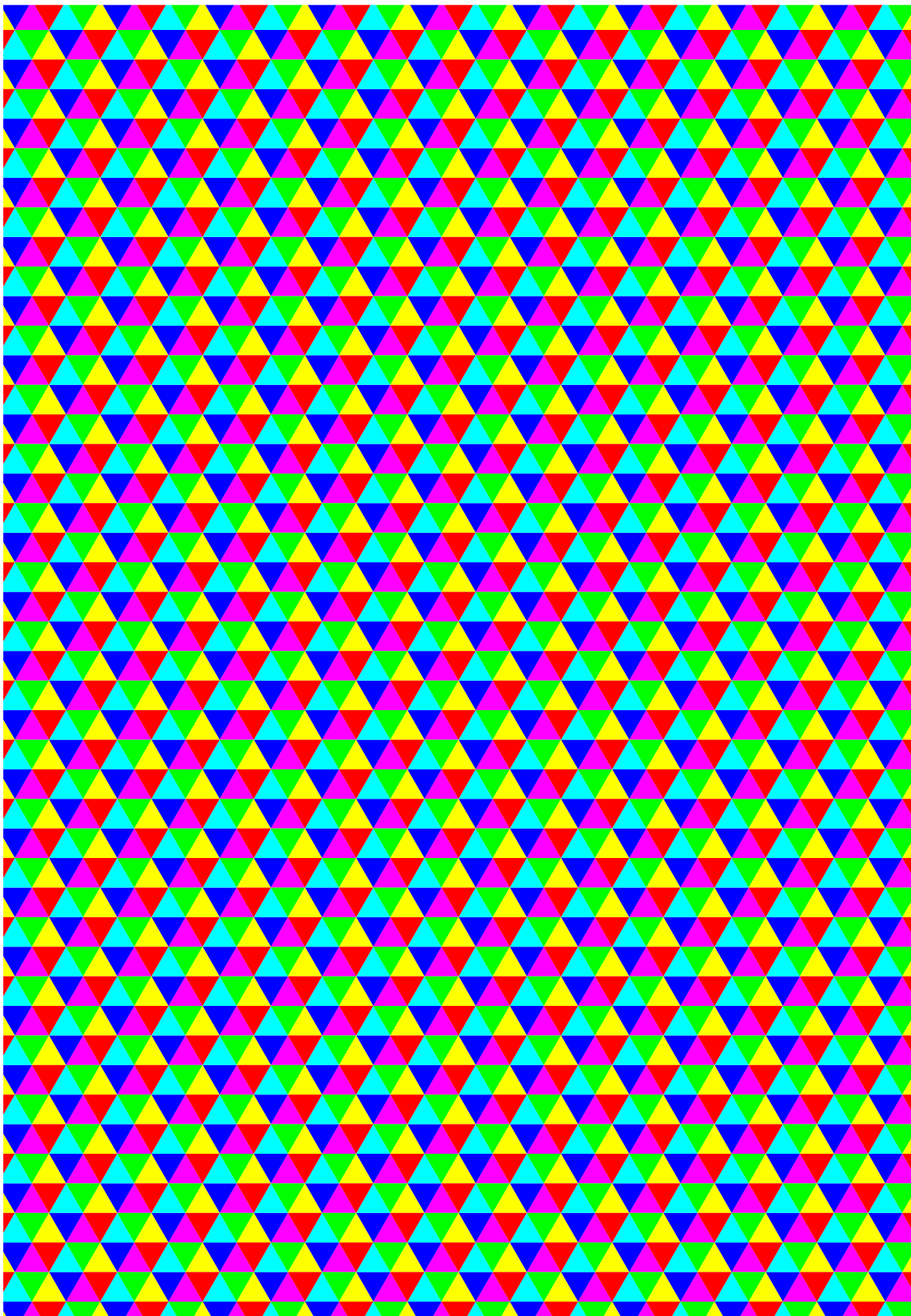


Note: See F.4 Designing with Data Trees for details about flattening



For different color effects, try connecting the Deconstruct Domain component to the saturation (S) or Luminance (L) inputs of the Colour HSL component.





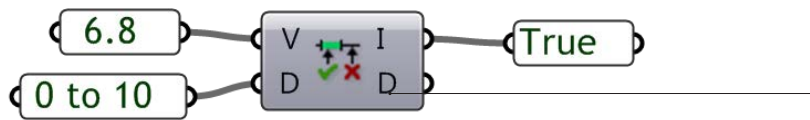
F.2.4 Booleans and Logical Operators

F.2.4.0 BOOLEANS

Numeric variables can store a whole range of different numbers. Boolean variables can only store two values referred to as Yes or No, True or False, 1 or 0. Obviously we never use booleans to perform calculations because of their limited range. We use booleans to evaluate conditions.

In Grasshopper, booleans can be used in several ways. The boolean parameter is a container for one or multiple boolean values, while the Boolean Toggle allows you to quickly change between single true and false values as inputs.

Grasshopper also has objects that test conditions and output boolean values. For example, the Includes component allows you to test a numeric value to see if it is included in a domain.



Boolean Parameter



Boolean Toggle - double click the boolean value to toggle between true and false

The Includes component is testing whether the number 6.8 is included in the domain from 0 to 10. It returns a boolean value of True.

F.2.4.1 LOGICAL OPERATORS

Logical operators mostly work on booleans and they are indeed very logical. As you will remember, booleans can only have two values. Boolean mathematics were developed by George Boole (1815-1864) and today they are at the very core of the entire digital industry. Boolean algebra provides us with tools to analyze, compare and describe sets of data. Although Boole originally defined six boolean operators we will only discuss three of them:

1. Not
2. And
3. Or

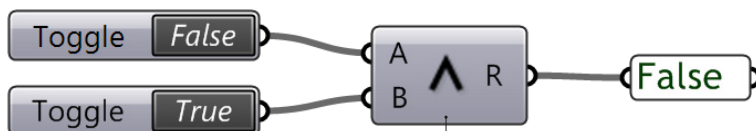
The Not operator is a bit of an oddity among operators, because it doesn't require two values. Instead, it simply inverts the one on the right. Imagine we have a script which checks for the existence of a bunch of Block definitions in Rhino. If a block definition does not exist, we want to inform the user and abort the script.



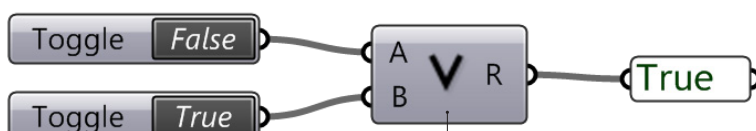
The Grasshopper Not operator (gate)

And and Or take two arguments on either side. The And operator requires both of them to be True in order for it to evaluate to True. The Or operator is more than happy with a single True value.

As you can see, the problem with Logical operators is not the theory, it's what happens when you need a lot of them to evaluate something. Stringing them together quickly results in convoluted code; not to mention operator precedence problems.



The Grasshopper And operator (gate)

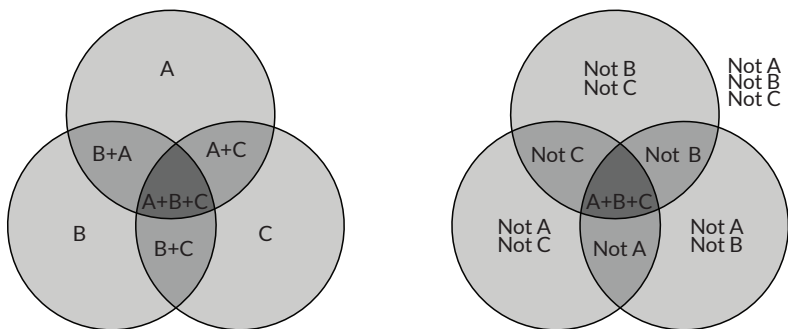


The Grasshopper Or operator (gate)

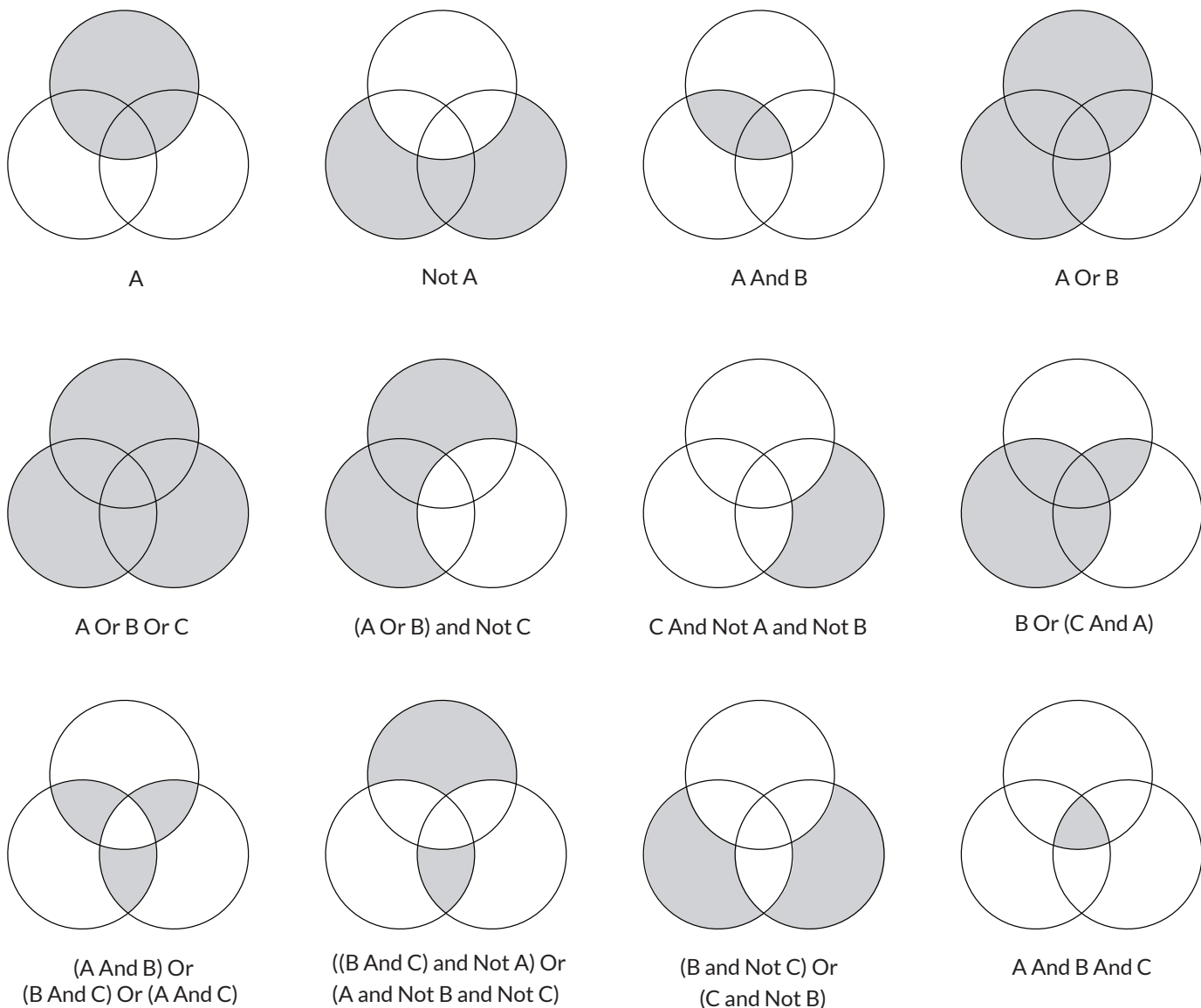
A	B	Result
True	True	True
True	False	False
False	True	False
False	False	False

A	B	Result
True	True	True
True	False	True
False	True	True
False	False	False

A good way to exercise your own boolean logic is to use Venn diagrams. A Venn diagram is a graphical representation of boolean sets, where every region contains a (sub)set of values that share a common property. The most famous one is the three-circle diagram:

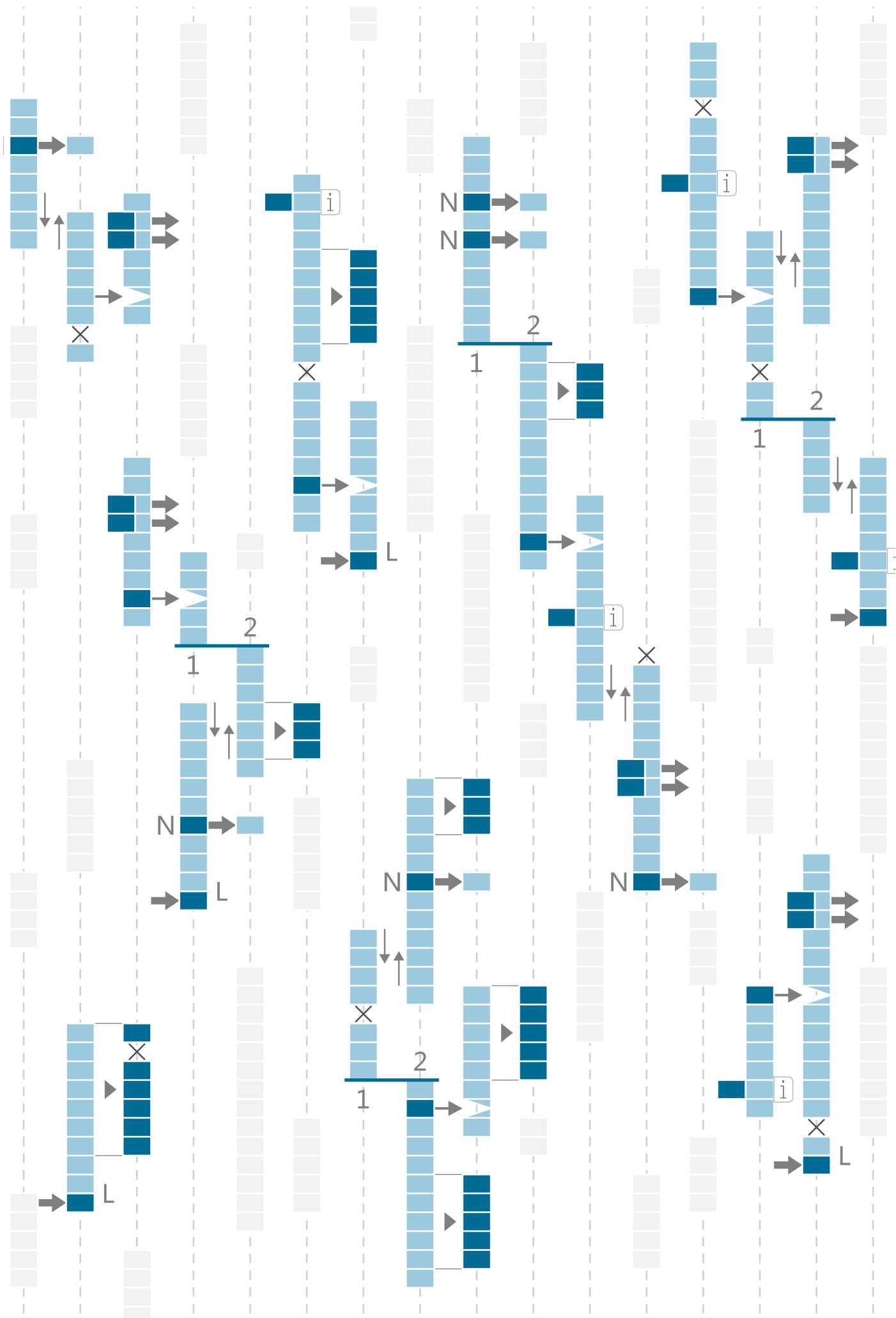


Every circular region contains all values that belong to a set; the top circle for example marks off set {A}. Every value inside that circle evaluates True for {A} and every value not in that circle evaluates False for {A}. By coloring the regions we can mimic boolean evaluation in programming code:



F.3 DESIGNING WITH LISTS

One of the most powerful features of Grasshopper is the ability to quickly build and manipulate lists of data. This chapter will explain how to create, manipulate, and visualize list data.

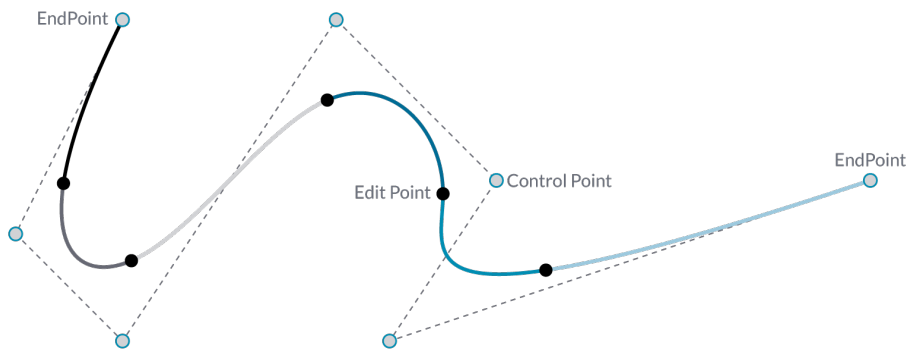
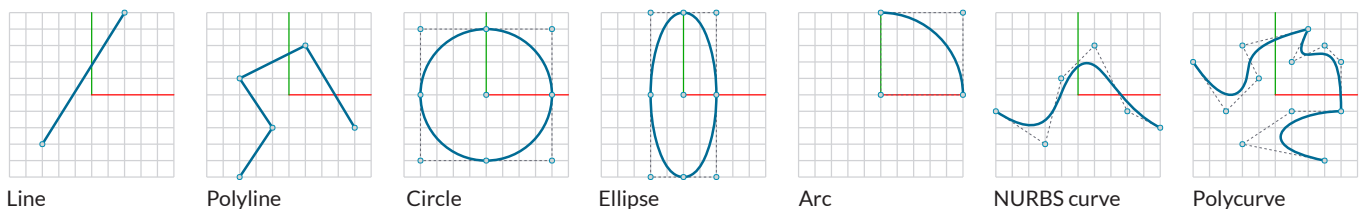


F.3.0

Curve Geometry

NURBS (non-uniform rational B-splines) are mathematical representations that can accurately model any shape from a simple 2D line, circle, arc, or box to the most complex 3D free-form organic surface or solid. Because of their flexibility and accuracy, NURBS models can be used in any process from illustration and animation to manufacturing.

Since curves are geometric objects, they possess a number of properties or characteristics which can be used to describe or analyze them. For example, every curve has a starting coordinate and every curve has an ending coordinate. When the distance between these two coordinates is zero, the curve is closed. Also, every curve has a number of control-points, if all these points are located in the same plane, the curve as a whole is planar. Some properties apply to the curve as a whole, while others only apply to specific points on the curve. For example, planarity is a global property while tangent vectors are a local property. Also, some properties only apply to some curve types. So far we've discussed some of Grasshopper's Primitive Curve Components such as: lines, circles, ellipses, and arcs.



F.3.0.0 NURBS CURVES

A NURBS curve is defined by degree, control points, and knots.

Degree: The degree is a positive whole number. This number is usually 1, 2, 3 or 5, but can be any positive whole number. The degree of the curve determines the range of influence the control points have on a curve; where the higher the degree, the larger the range. NURBS lines and polylines are usually degree 1, NURBS circles are degree 2, and most free-form curves are degree 3 or 5.

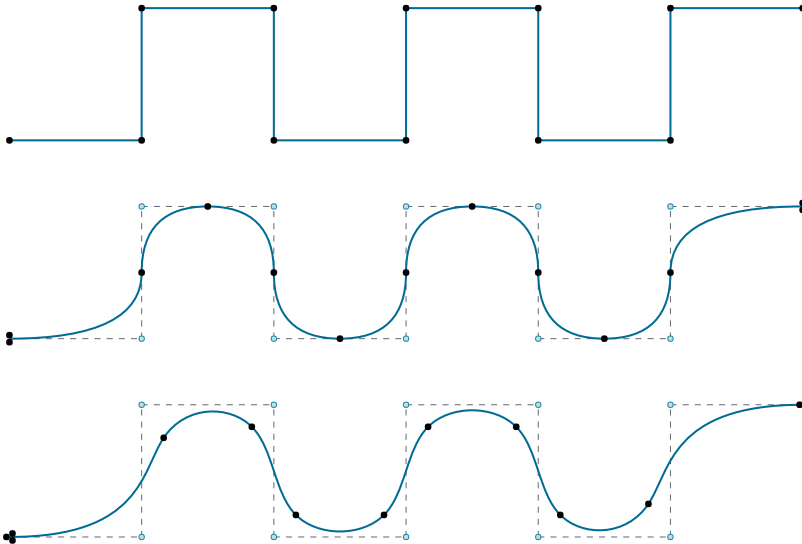
Control Points: The control points are a list of at least degree+1 points. One of the easiest ways to change the shape of a NURBS curve is to move its control points.

Weight: Control points have an associated number called a weight. Weights are usually positive numbers. When a curve's control points all have the same weight (usually 1), the curve is called non-rational, otherwise the curve is called rational. Most NURBS curves are non-rational. A few NURBS curves, such as circles and ellipses, are always rational.

Knots: Knots are a list of $(\text{degree}+N-1)$ numbers, where N is the number of control points.

Edit Points: Points on the curve evaluated at knot averages. Edit points are like control points except they are always located on the curve and moving one edit point generally changes the shape of the entire curve (moving one control point only changes the shape of the curve locally). Edit points are useful when you need a point on the interior of a curve to pass exactly through a certain location.

NURBS curve knots as a result of varying degree:



A D^1 NURBS curve behaves the same as a polyline. A D^1 curve has a knot for every control point.

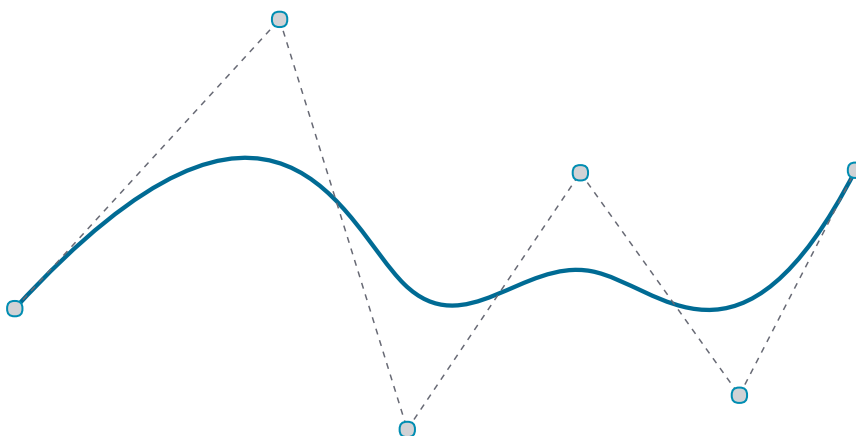
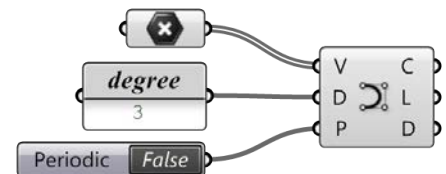
D^2 NURBS curves are typically only used to approximate arcs and circles. The spline intersects with the control polygon halfway each segment.

D^3 is the most common type of NURBS curve and is the default in Rhino. You are probably very familiar with the visual progression of the spline, even though the knots appear to be in odd locations.

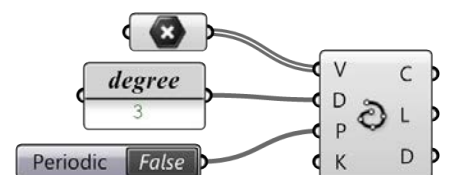
F.3.0.1 GRASSHOPPER SPLINE COMPONENTS

Grasshopper has a set of tools to express Rhino's more advanced curve types like nurbs curves and poly curves. These tools can be found in the Curve/Splines tab.

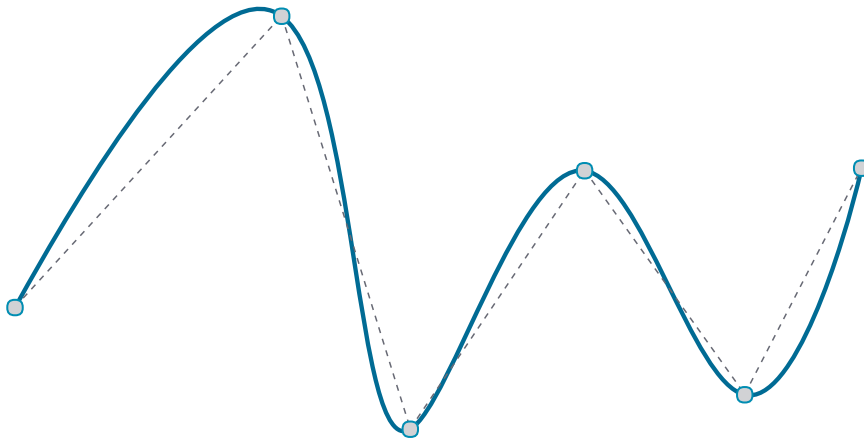
Nurbs Curve (Curve/Spline/Nurbs curve): The Nurbs Curve component constructs a NURBS curve from control points. The V input defines these points, which can be described implicitly by selecting points from within the Rhino scene, or by inheriting volatile data from other components. The Nurbs Curve-D input sets the degree of the curve.



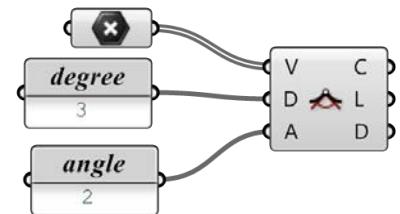
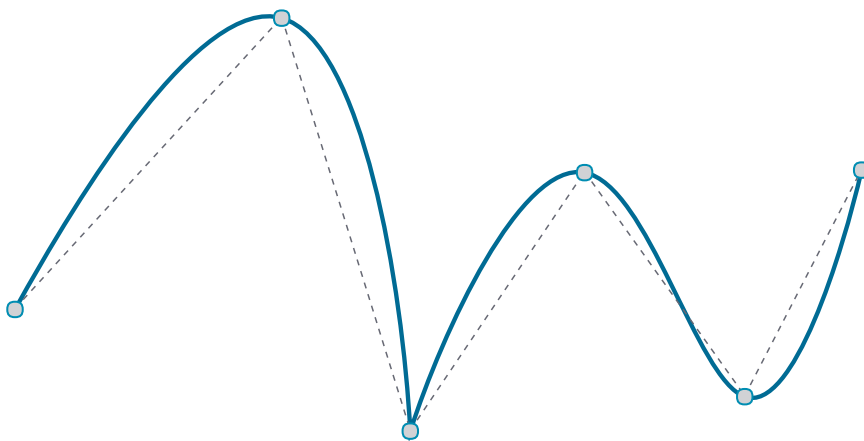
Interpolate Curve (Curve/Spline/Interpolate): Interpolated curves behave slightly differently than NURBS curves. The V-input is for the component is similar to the NURBS component, in that it asks for a specific set of points to create the curve. However, with the Interpolated Curve method, the resultant curve will actually pass through these points, regardless of the curve degree. In the NURBS curve component, we could only achieve this when the curve degree was set to one. Also, like the NURBS curve component, the D input defines the degree of the resultant curve. However, with this method, it only takes odd



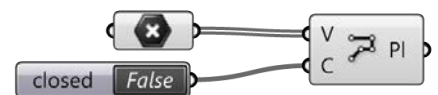
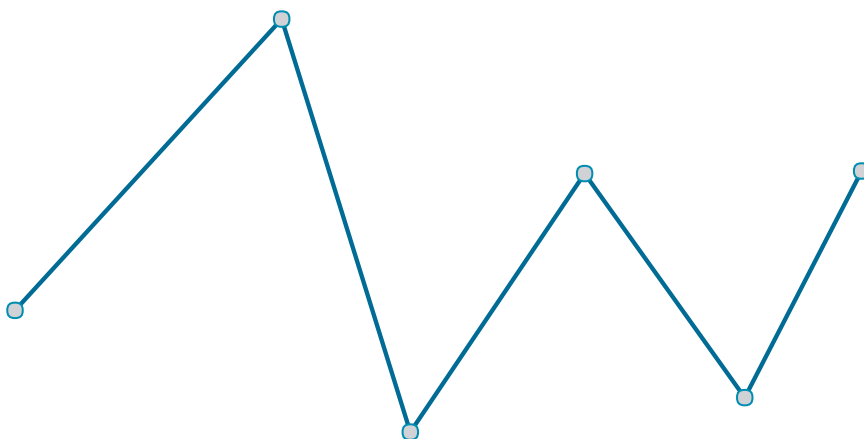
numbered values for the degree input. Again, the P-input determines if the curve is Periodic. You will begin to see a bit of a pattern in the outputs for many of the curve components, in that, the C, L, and D outputs generally specify the resultant curve, the length, and the curve domain respectively.



Kinky Curve (Curve/Spline/Kinky Curve): The kinky curve component allows you the ability to control a specific angle threshold, A, where the curve will transition from a kinked line, to a smooth, interpolated curve. It should be noted that the A-input requires an input in radians.



Polyline (Curve/Spline/Polyline): A polyline is a collection of line segments connecting two or more points, the resultant line will always pass through its control points; similar to an Interpolated Curve. Like the curve types mentioned above, the V-input of the Polyline component specifies a set of points that will define the boundaries of each line segment that make up the polyline. The C-input of the component defines whether or not the polyline is an open or closed curve. If the first point location does not coincide with the last point location, a line segment will be created to close the loop. The output for the Polyline component is different than that of the previous examples, in that the only resultant is the curve itself.

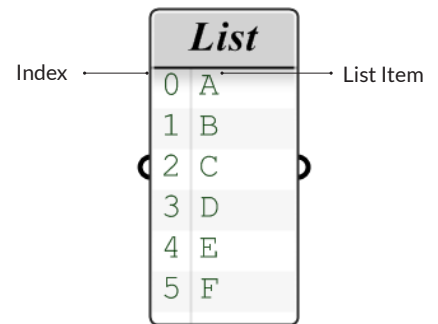
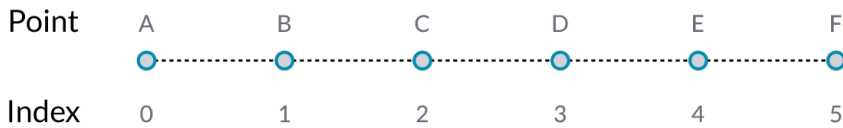


F.3.1 What is a List?

It's helpful to think of Grasshopper in terms of flow, since the graphical interface is designed to have data flow into and out of specific types of components. However, it is the data that define the information flowing in and out of the components. Understanding how to manipulate list data is critical to understanding the Grasshopper plug-in.

Grasshopper generally has two types of data: persistent and volatile. Even though the data types have different characteristics, typically Grasshopper stores this data in an array, a list of variables.

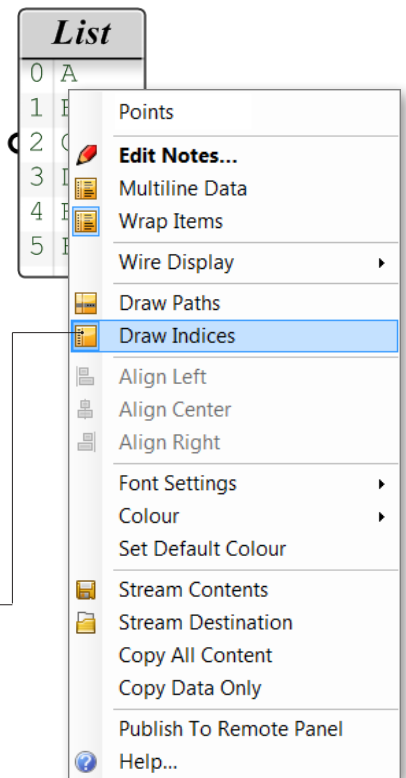
When storing data in a list, it's helpful to know the position of each item in that list so that we can begin to access or manipulate certain items. The position of an item in the list is called its index number.



The only thing that might seem odd at first is that the first index number of a list is always 0; not 1. So, when we talk about the first item of a list, we actually mean the item that corresponds to index number 0.

For example, if we were to count the number of fingers we have on our right hand, chances are that you would have counted from 1 to 5. However, if this list had been stored in an array, then our list would have counted from 0 to 4. Note, that we still have 5 items in the list; it's just that the array is using a zero-based counting system. The items being stored in the list don't just have to be numbers. They can be any data type that Grasshopper supports, such as points, curves, surfaces, meshes, etc.

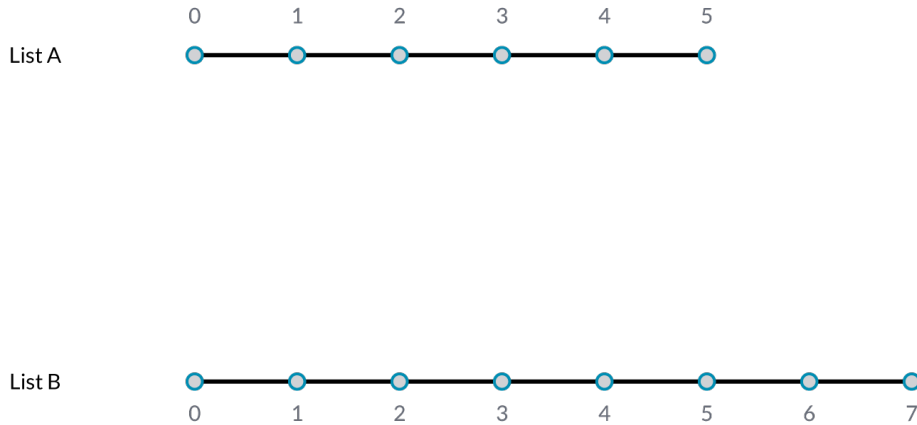
Often times the easiest way to take a look at the type of data stored in a list is to connect a Text Panel (Params/Input/Panel) to the output of a particular component. By default, the Text Panel automatically shows all index numbers to the left side of the panel and displays the data items on the right side of the panel. The index numbers will become a crucial element when we begin working with our lists. You can turn the index numbers on and off by right-clicking on the Text Panel and clicking on the "Draw Indices" item in the sub-menu. For now, let's leave the entry numbers turned on for all of our text panels.



F.3.2 Data Stream Matching

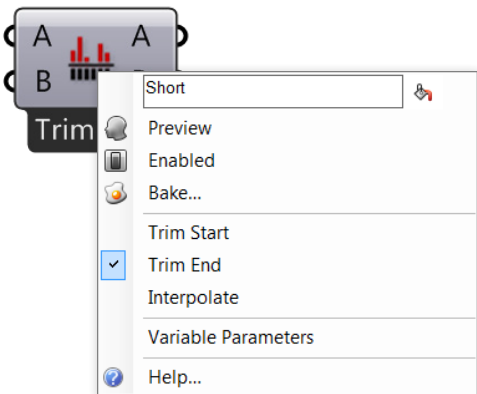
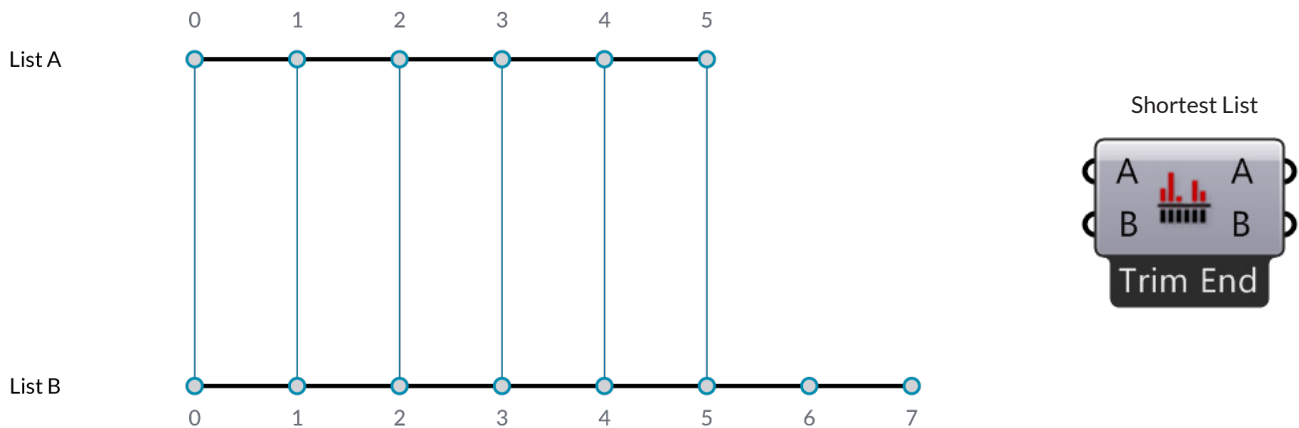
Data matching is a problem without a clean solution. It occurs when a component has access to differently sized inputs. Changing the data matching algorithm can lead to vastly different results.

Imagine a component which creates line segments between points. It will have two input parameters which both supply point coordinates (List A and List B):



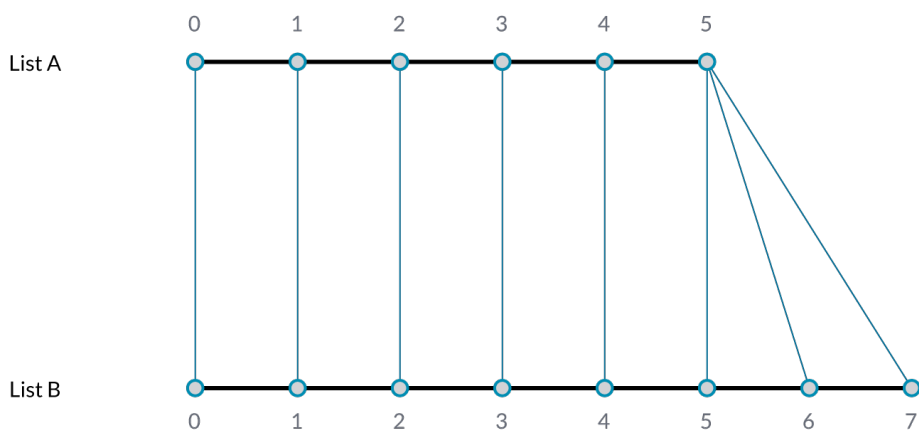
As you can see there are different ways in which we can draw lines between these sets of points. New to Grasshopper 0.9 are three components for data matching, found under the Sets/List panel: Shortest List, Longest List, and Cross Reference. These new components allow for greater flexibility within the three basic data matching algorithms. Right clicking each component allows you to select a data matching option from the menu.

The simplest way is to connect the inputs one-on-one until one of the streams runs dry. This is called the “Shortest List” algorithm:

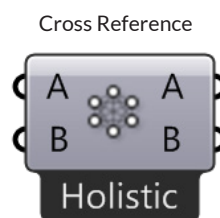
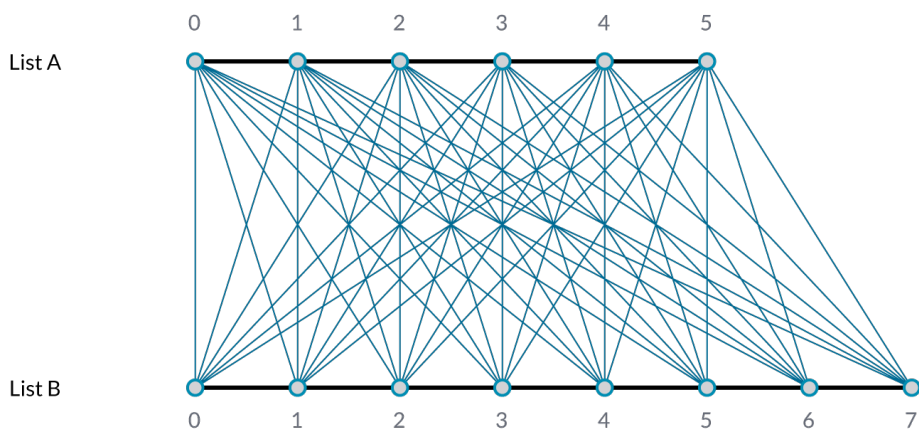


Select a matching algorithm option from the component menu by right-clicking the component

The “**Longest List**” algorithm keeps connecting inputs until all streams run dry. This is the default behavior for components:

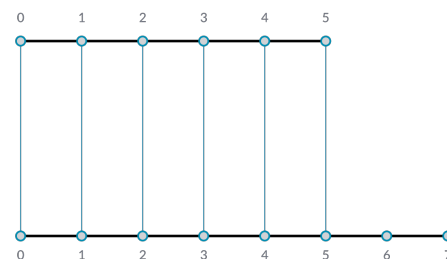
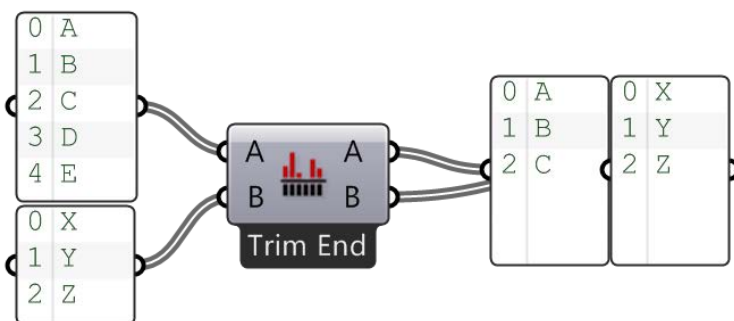


Finally, the “**Cross Reference**” method makes all possible connections:

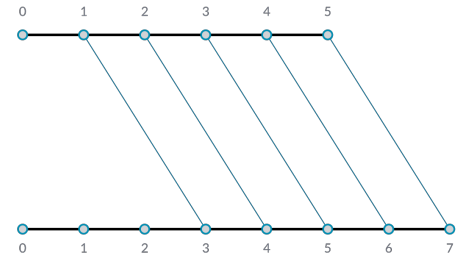
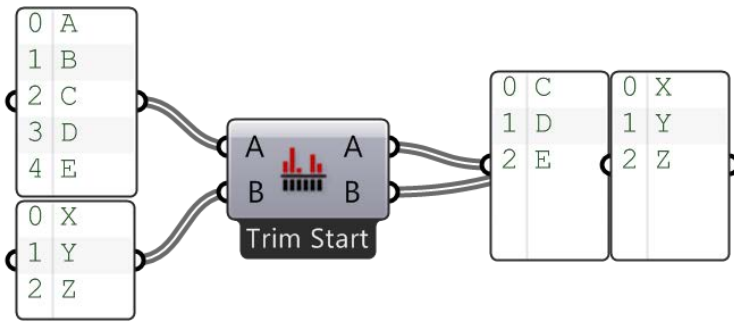


This is potentially dangerous since the amount of output can be humongous. The problem becomes more intricate as more input parameters are involved and when the volatile data inheritance starts to multiply data, but the logic remains the same.

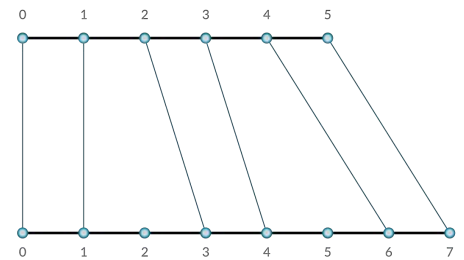
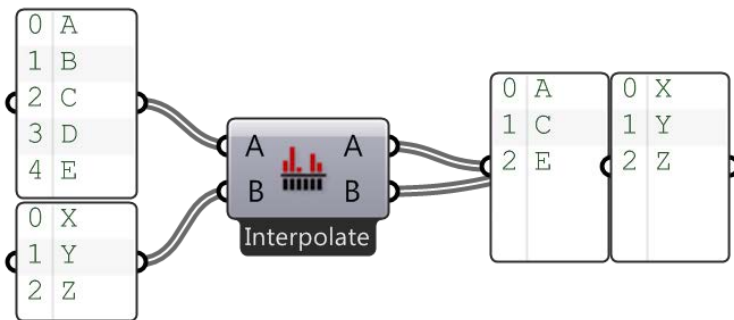
Let’s look more closely at the Shortest List component:



Here we have two input lists {A,B,C,D,E} and {X,Y,Z}. Using the Trim End option, the last two items in the first list are disregarded, so that the lists are of equal length.

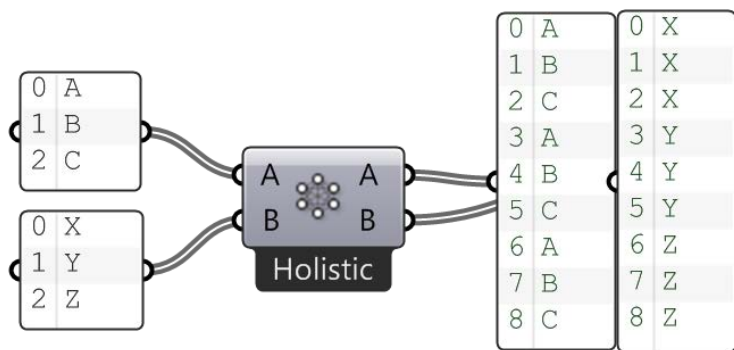


Using the Trim Start option, the first two items in the first list are disregarded, so that the lists are of equal length.



The Interpolate option skips the second and fourth items in the first list.

Now let's look at the Cross Reference component:



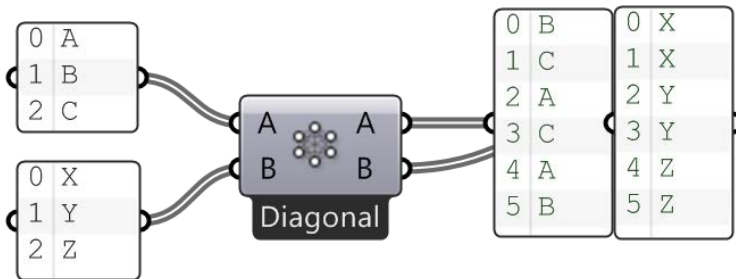
Here we have two input lists {A,B,C} and {X,Y,Z}. Normally Grasshopper would iterate over these lists and only consider the combinations {A,X}, {B,Y} and {C,Z}. There are however six more combinations that are not typically considered, to wit: {A,Y}, {A,Z}, {B,X}, {B,Z}, {C,X} and {C,Y}. As you can see the output of the Cross Reference component is such that all nine permutations are indeed present.

We can denote the behaviour of data cross referencing using a table. The rows represent the first list of items, the columns the second. If we create all possible permutations, the table will have a dot in every single cell, as every cell represents a unique combination of two source list indices.

	0	1	2	3
0	•	•	•	•
1	•	•	•	•
2	•	•	•	•
3	•	•	•	•
4	•	•	•	•
5	•	•	•	•

Sometimes however you don't want all possible permutations. Sometimes you wish to exclude certain areas because they would result in meaningless or invalid computations. A common exclusion principle is to ignore all cells that are on the diagonal of the table. The image above shows a 'holistic' matching, whereas the 'diagonal' option (available from the Cross Reference]component menu has gaps for {0,0}, {1,1}, {2,2} and {3,3}.

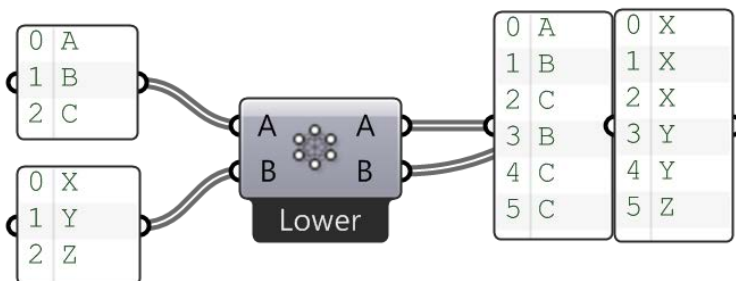
If we apply this to our {A,B,C}, {X,Y,Z} example, we should expect to not see the combinations for {A,X}, {B,Y} and {C,Z}:



	0	1	2	3
0		•	•	•
1	•		•	•
2	•	•		•
3	•	•	•	
4	•	•	•	•
5	•	•	•	•

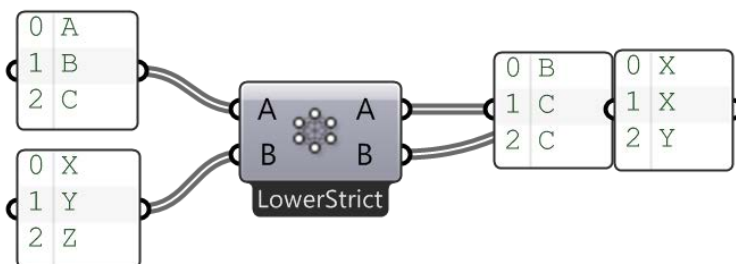
The rule that is applied to 'diagonal' matching is: "Skip all permutations where all items have the same list index". 'Coincident' matching is the same as 'diagonal' matching in the case of two input lists, but the rule is subtly different: "Skip all permutations where any two items have the same list index".

The four remaining matching algorithms are all variations on the same theme. 'Lower triangle' matching applies the rule: "Skip all permutations where the index of an item is less than the index of the item in the next list", resulting in an empty triangle but with items on the diagonal.



	0	1	2	3
0	•			
1	•	•		
2	•	•	•	
3	•	•	•	•
4	•	•	•	•
5	•	•	•	•

'Lower triangle (strict)' matching goes one step further and also eliminates the items on the diagonal:



	0	1	2	3
0				
1	•			
2	•	•		
3	•	•	•	
4	•	•	•	•
5	•	•	•	•

'Upper Triangle' and 'Upper Triangle (strict)' are mirror images of the previous two algorithms, resulting in empty triangles on the other side of the diagonal line.

F.3.3 Creating Lists

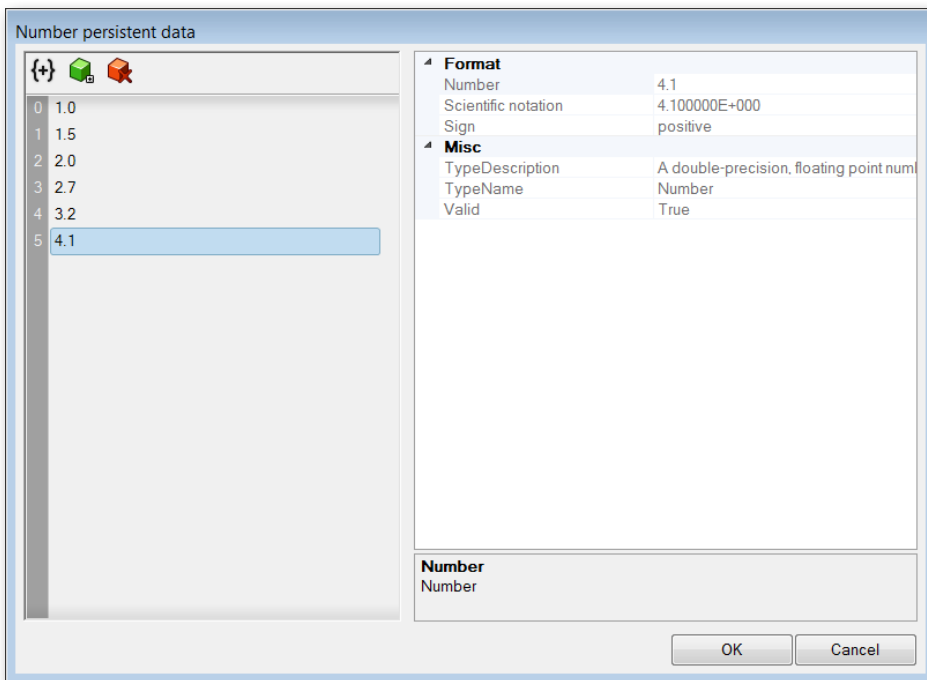
There are many different ways to generate lists in Grasshopper. Below, we'll look at a few different methods for generating lists and then look at how the data can be used to convey information in the viewport via a visualization.

F.3.3.0 MANUAL LIST CREATION

Perhaps the easiest way to create a list (and one of the most over-looked methods) is to manually type in a list of values into a parameter. Using this method puts added responsibility on the user because this method relies on direct user input (ie. persistent data) for the list creation. In order to change the list values, the user has to manually type in each individual value which can be difficult if the list has many entries. There are several ways to manually create a list. One way is to use a Number parameter. Right click the Number parameter and select "Manage Number Collection."

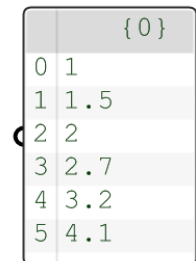


Right click the number component to open the Number Collection Manager



Click the Add Item icon to add a number to the list

Double click the number to change its value

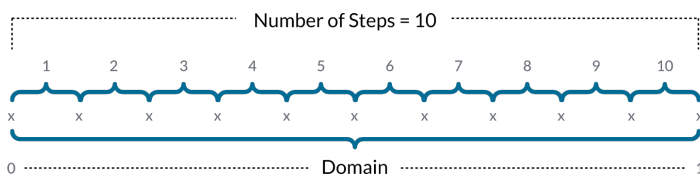


Another method is to manually enter the list items into a panel. Make sure that "Multiline Data" is deselected.

F.3.3.1 RANGE

The Range component, found under Sets/Sequence/Range, creates a list of evenly spaced numbers between a low and a high value called the Domain. A domain (also sometimes referred to as an interval) is every possible number between two numeric extremes.

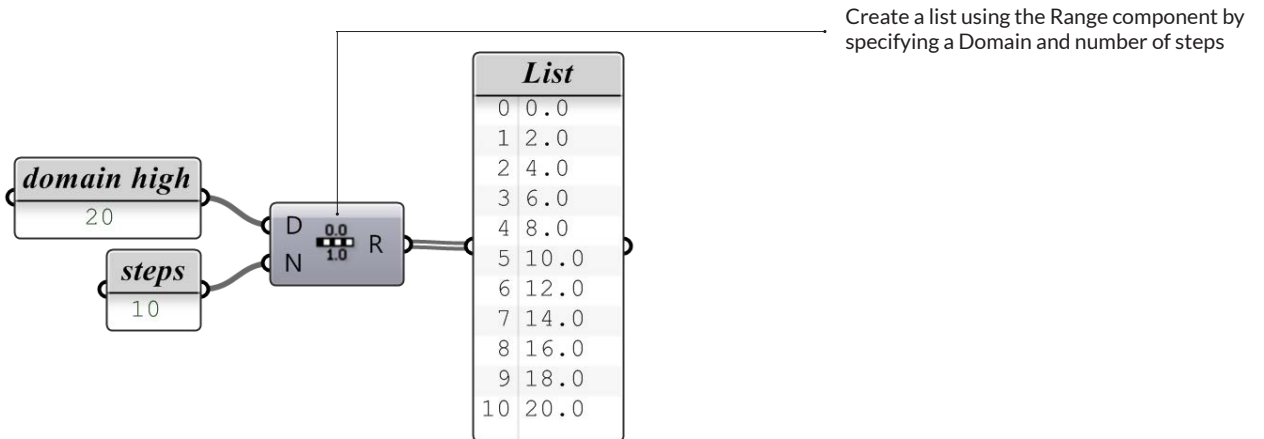
A Range component divides a numeric domain into even segments and returns a



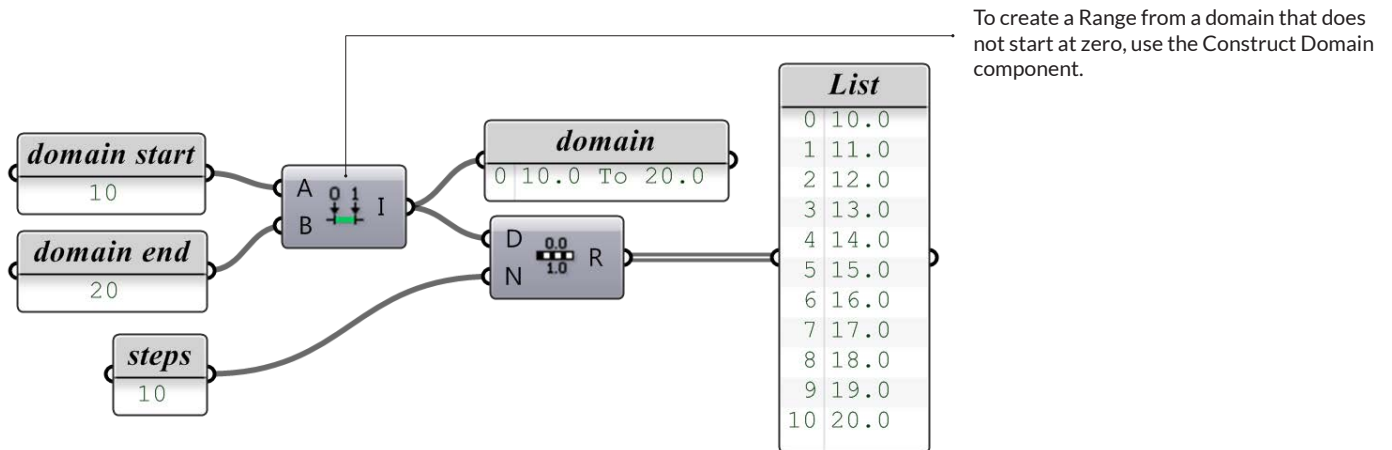
Total Number of Points = 11

A Range component divides a numeric domain into even segments and returns a list of values.

In the example below, the numeric domain has been defined as every possible number between 0 and 20. The Range component takes that domain and divides it up by the number of steps (in this case 10). So, we have 10 even spaced segments. The Range component returns a list of values. Because it keeps the first and the last values in the list, the output of a Range component is always one more than the number of steps. In the example above, we created 10 steps, so the Range component returns 11 values.

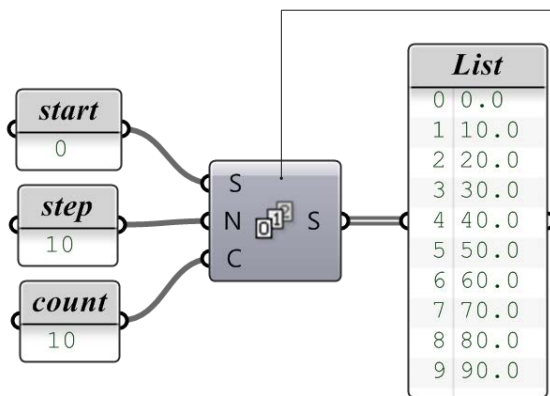


You may have noticed something a little quirky about the setup we just made. We know that a domain is always defined by two values (a high and low value). Yet, in our definition we simply connected a single value to the domain input. In order to avoid errors, Grasshopper makes an assumption that you are trying to define a domain between zero and some other number (our slider value). In order to create a range between two numbers that doesn't start at zero, we must use the Construct Domain component to specify the domain.



F.3.3.2 SERIES

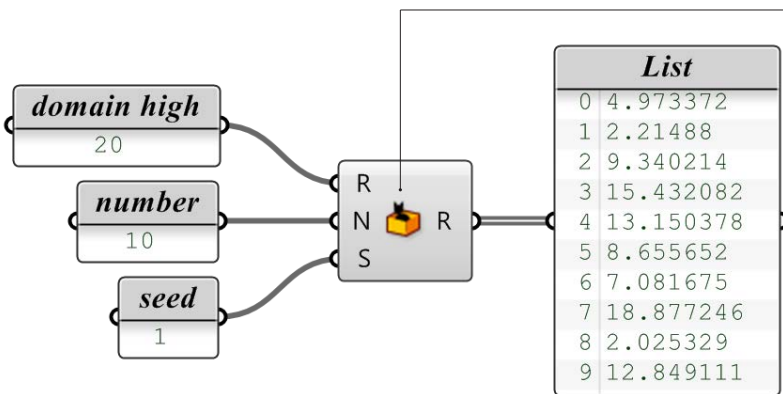
The Series component is similar to the Range component, in that, it also creates a list of numbers. However a Series component is different because it creates a set of discrete numbers based on a start value, step size, and the number of values in the series.



The Series component creates a list based on a start value, step value, and the number of values in the list.

F.3.3.3 RANDOM

The Random Component (Sets/Sequence/Random) can be used to generate a list of pseudo random numbers. They are referred to as “pseudo” random because the number sequence is unique but stable for each seed value. Thus, you can generate an entirely new set of random numbers by changing the seed value (S-input). The domain, as in the previous example, is a defined interval between two numeric extremes.

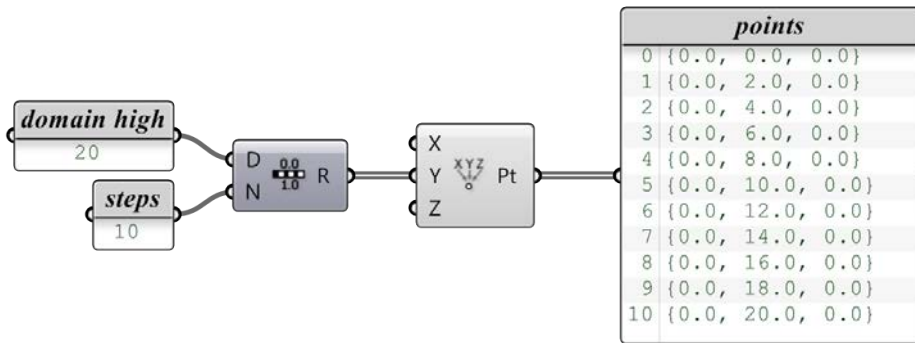


The Random component creates a pseudo-random list of values.

F.3.4 List Visualization

Understanding lists in Grasshopper can be difficult without being able to see the data flowing from one component to the next. There are several ways to visualize lists that can help to understand and manipulate data.

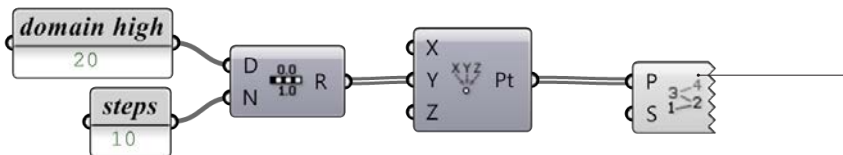
There are many different ways to visualize a list of data. The most common way is to create some geometry with the list of data. By connecting the R output of the Range component to the Y input of the Construct Point component, we can see an array of points in the Y direction.



Lets look at some components that can help us understand the data.

F.3.4.0 THE POINT LIST COMPONENT

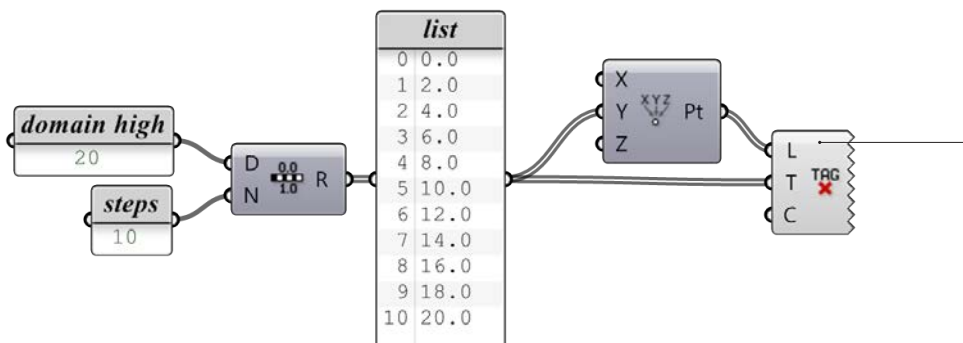
The Point List component is an extremely useful tool for visualizing the order of a set of points in a list. Essentially, the Point List component places the index item number next to the point geometry in the viewport. You can also specify whether or not you want to draw the number tags, the connection lines, or the size of the text tags.



You can visualize the order of a set of points using the Point List component.

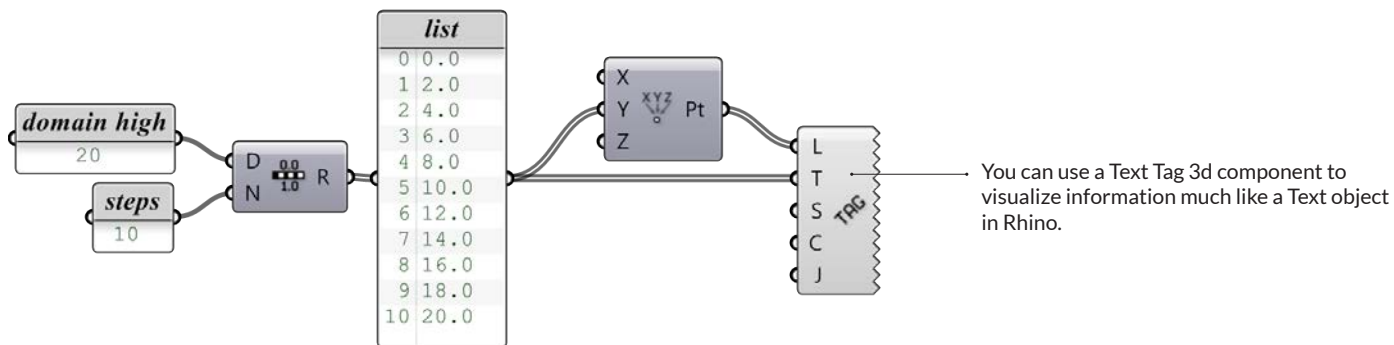
F.3.4.1 TEXT TAGS

The text tag component allows you to draw little strings (a string is a set of ASCII characters) in the viewport as feedback items. Text and location are specified as input parameters. When text tags are baked into the scene, they turn into Text Dots. The other interesting thing about Text Tags is that they are viewport independent - meaning the tags always face the camera (including perspective views) and they always remain the same size on the screen regardless of your zoom settings.



You can visualize any string information in the viewport using the Text Tag component. In this setup, we have decided to display the value of each point on top of each point location. We could have assigned any text to display,

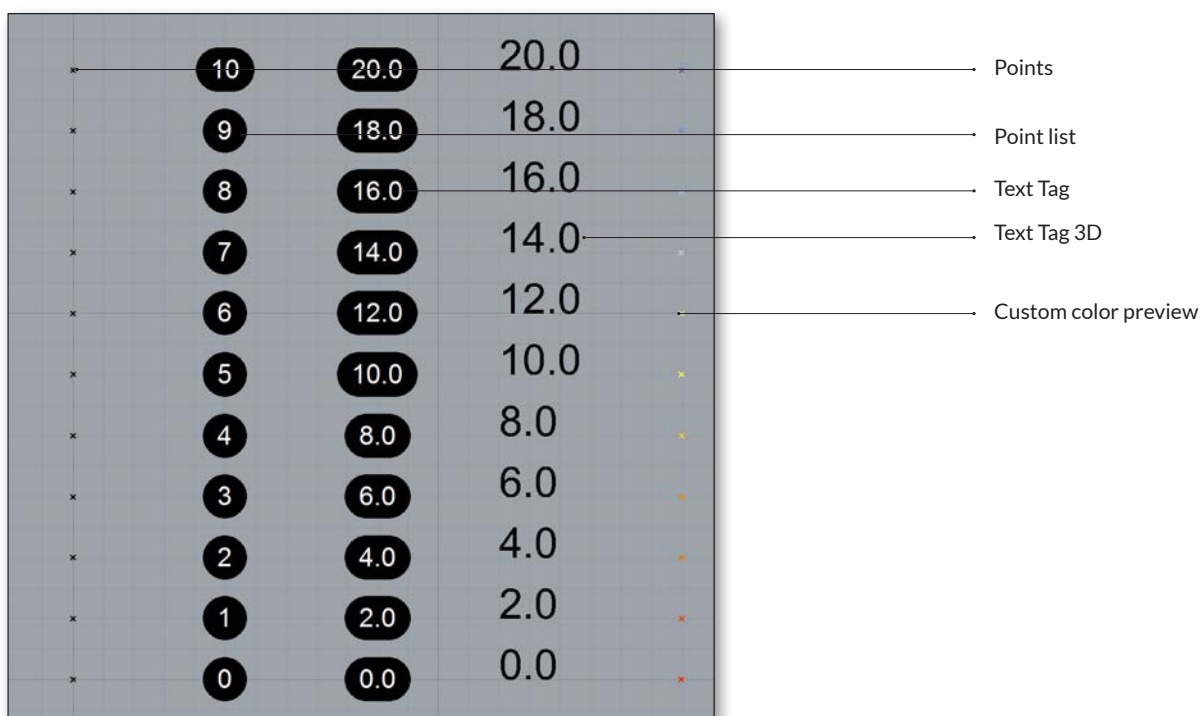
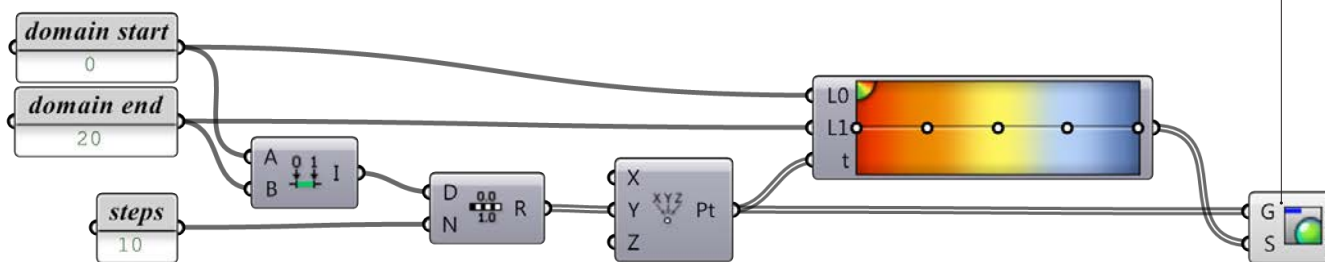
The Text Tag 3d component works very similarly to the Text Tag component. They differ, in that, when Text Tag 3d objects are baked into the scene, they become Text objects in Rhino. The scale of the Text Tag 3d font can also be controlled via an input (which is inaccessible in the Text Tag component).



F.3.4.2 COLOR

One of the other things we can do to visualize the list data is to assign color to the geometry. Grasshopper has limited 'rendering' capabilities, but we can control simple Open GL settings like color, specular color, transparency, etc. The L0 value represents the low end (left side) of the gradient, whereas the L1 value represents the upper end (right side). These values correspond to the start and end of our domain. The t-values are the elements in the list that will get mapped somewhere within the L0 and L1 range. The output of the gradient is a list of RGB color values which correspond to each point in our list. Right-click on the Gradient to set one of the gradient presets, or define your own using the color node points.

The Custom Preview component can be used to assign color values to any specific geometry.

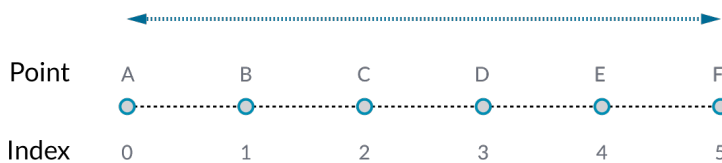
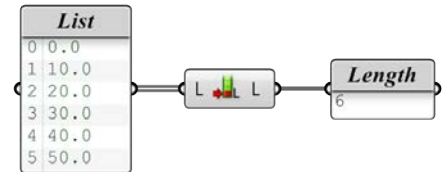


F.3.5 List Management

One of the most powerful features of Grasshopper is the ability to quickly build and manipulate various lists of data. We can store many different types of data in a list (numbers, points, vectors, curves, surfaces, breps, etc.) and there are a number of useful tools found under the Sets/List subcategory.

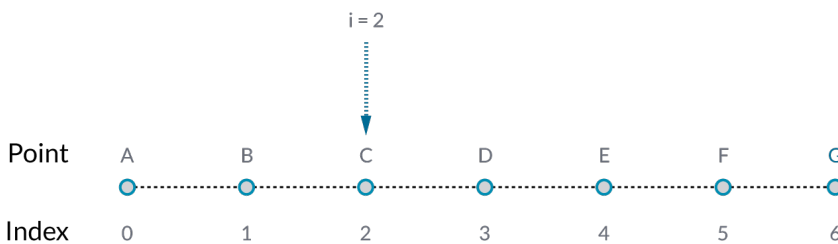
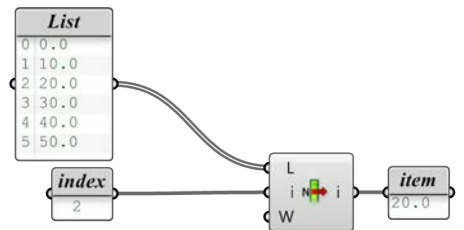
F.3.5.0 LIST LENGTH

The List Length component (Sets/List/List Length) essentially measures the length of the List. Because our lists always start at zero, the highest possible index in a list equals the length of the list minus one. In this example, we have connected our base List to the List Length-L input, showing that there are 6 values in the list.



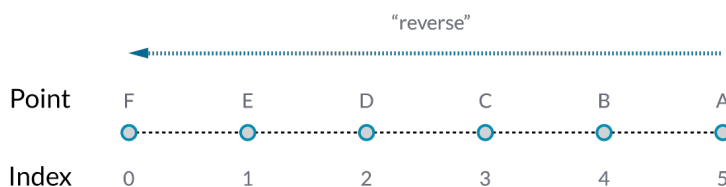
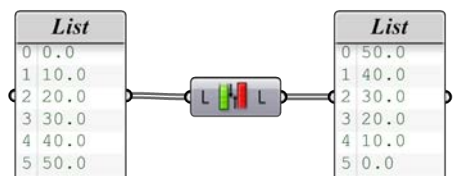
F.3.5.1 LIST ITEM

Our List is fed into a List Item component (Sets/List/List Item) in order to retrieve a specific data item from within a data set. When accessing individual items in a list, we have to specify the i-input; which corresponds to the index number we would like to retrieve. We can feed a single integer or a list of integers into the i-input depending on how many items we would like to retrieve. The L-input defines the base list which we will be analyzing. In this example, we have set the i-input to 5.0 so the List Item component returns the data item associated with the 5th entry number in our list.



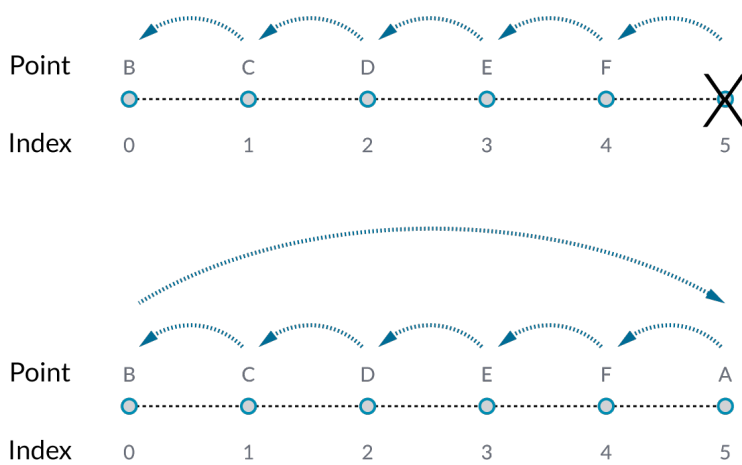
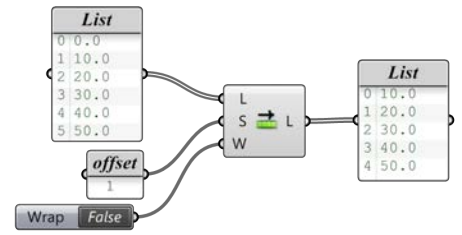
F.3.5.2 REVERSE LIST

We can invert the order of our list by using a Reverse List component (Sets/List/Reverse). If we input an ascending list of numbers from 0.0 to 9.0 into the Reverse List component; the output returns a descending list from 9.0 to 0.0.



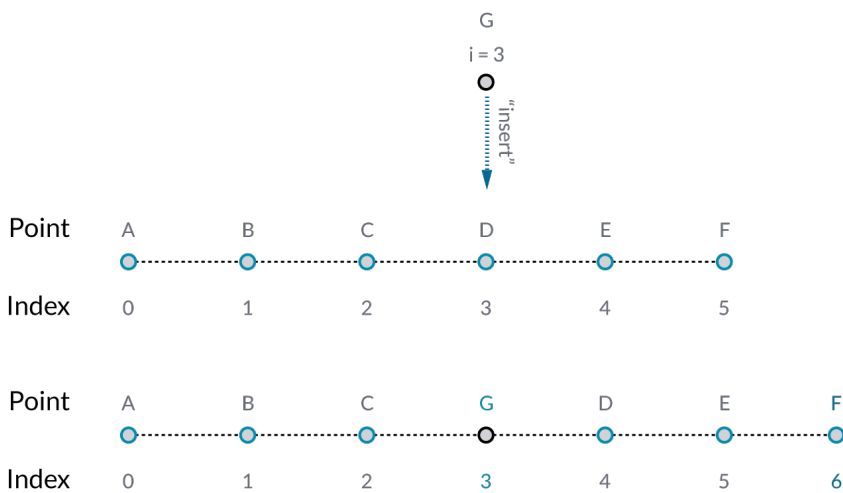
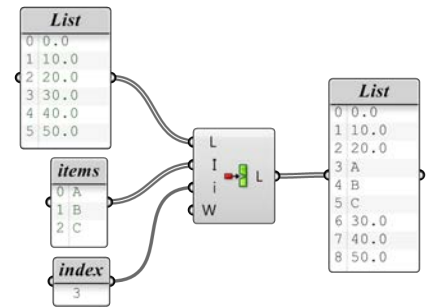
F.3.5.3 SHIFT LIST

The Shift List component (Sets/Sequence/Shift List) will either move the list up or down a number of increments depending on the value of the shift offset. We have connected the List output into the Shift-L input, while also connecting a number to the Shift-S input. If we set the offset to -1, all values of the list will move down by one entry number. Likewise, if we change the offset to +1, all values of the list will move up by one entry number. If Wrap input equals True, then items that fall off the ends are re-appended to the beginning or end of the list. In this example, we have a shift offset value set to +1, so that our list moves up by one entry number. Now, we have a decision to make on how we would like to treat the first value. If we set the Wrap value to False, the first entry will be shifted up and out of the list, essentially removing this value from the data set (so, the list length is one less than it was before). However, if we set the wrap value to True, the first entry will be moved to the bottom of the list



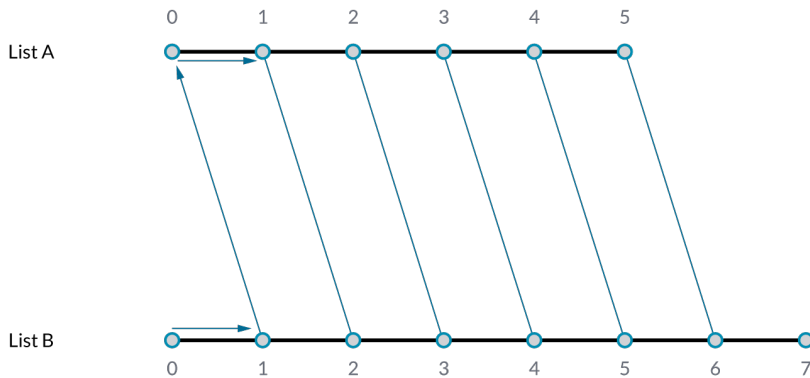
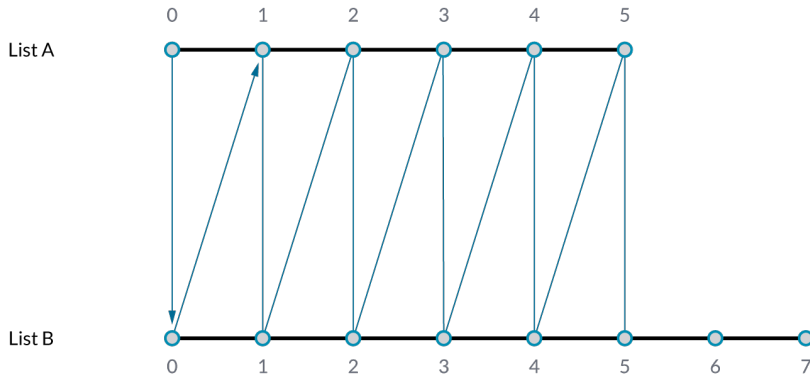
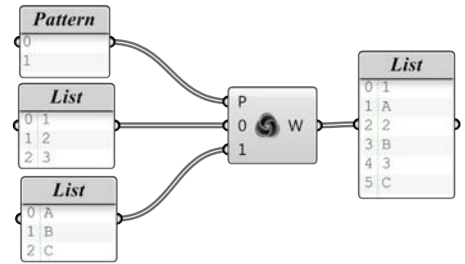
F.3.5.4 INSERT ITEMS

The Insert Items component (Sets/Lists/Insert Items) enables you to insert a collection of items into a list. In order for this to work properly, you need to know the items you want to insert and the index position for each new item. In the example below, we will insert the letters A, B, and C into index position three.



F.3.5.5 WEAVE

The Weave component (Sets/Lists/Weave) merges two or more lists together based on a specified weave pattern (P input). When the pattern and the streams do not match perfectly, this component can either insert nulls into the output streams or it can ignore streams which have already been depleted.



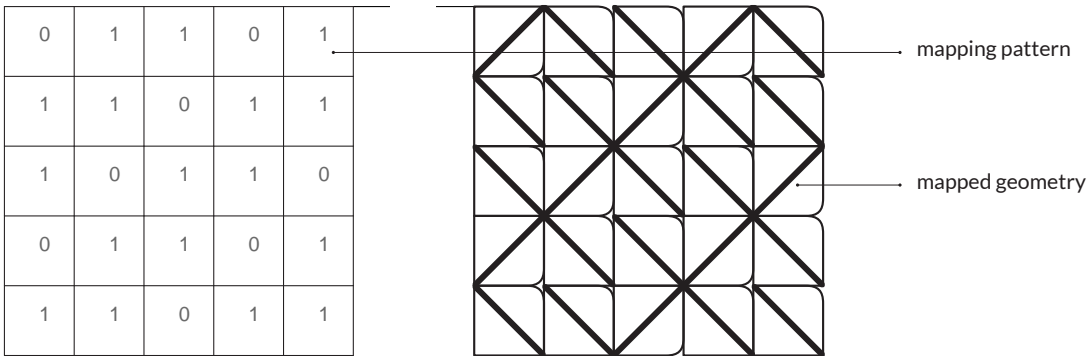
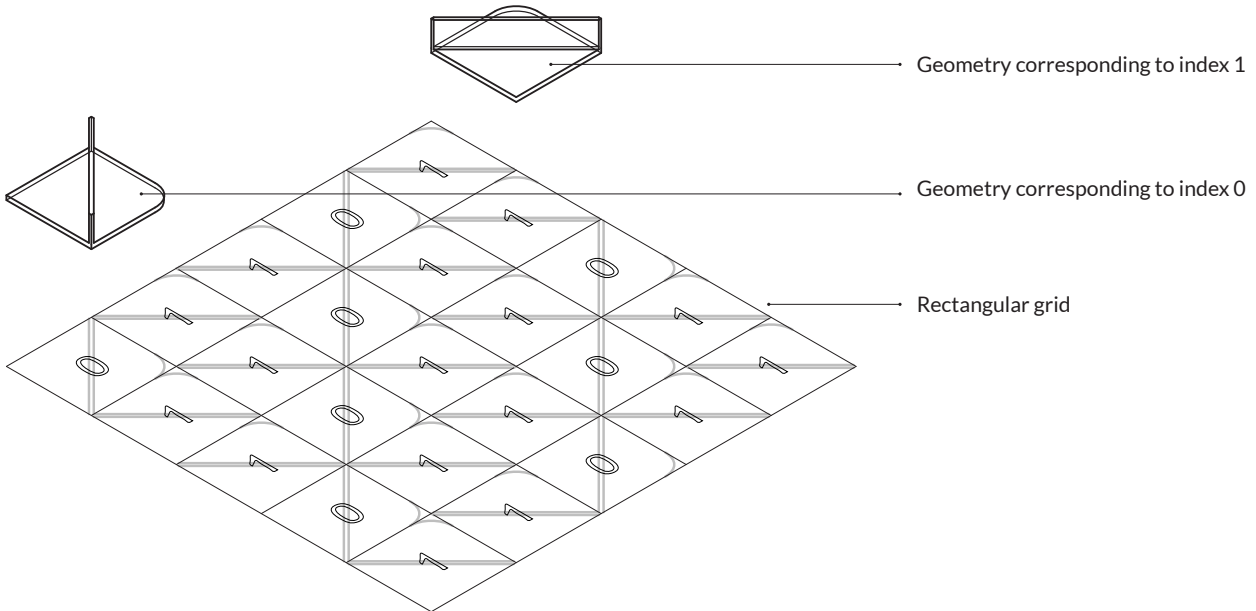
F.3.5.6 CULL PATTERN

The Cull component (Sets/Sequence/Cull Pattern) removes elements in a list using a repeating bit mask. The bit mask is defined as a list of Boolean (true or false) values. The bit mask is repeated until all elements in the data list have been evaluated.



F.3.6 Working with Lists

Lets take a look at an example using the components from the previous section. In this example, we are creating a tile pattern by mapping geometry to a rectangular grid. The pattern is created by using the List Item component to retrieve the desired tile from a list of geometry.



01. Start a Rhinoceros File
02. Create two equally sized squares
03. Create different geometries in each square
04. Start a new definition, type Ctrl+N (in Grasshopper)
05. Params/Geometry/Geometry - Drag and drop two Geometry parameters onto the canvas
06. Right-Click the first Geometry Parameter and select set one Geometry. Set the first Geometry that you are referencing
07. Right-Click the second Geometry Parameter and select set one Geometry. Set the second Geometry that you are referencing
08. Params/Geometry/Curve - Drag and drop two Curve parameters onto the canvas

For this exercise we will be referencing geometry in Rhino.

We created a simple surface with a tab. The tab is filleted to demonstrate the orientation and the base is filleted to distinguish the two geometries.

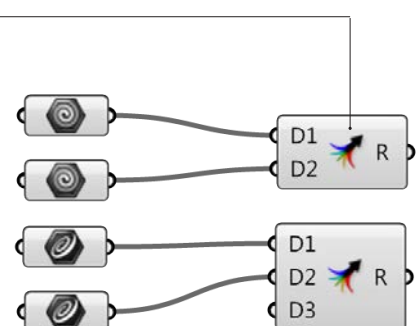
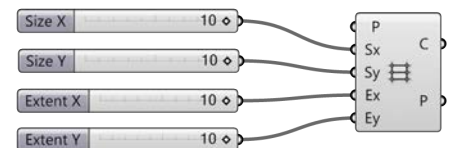
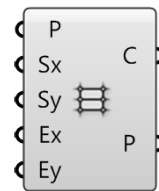


You can reference more geometries, but for simplicity we are using two.



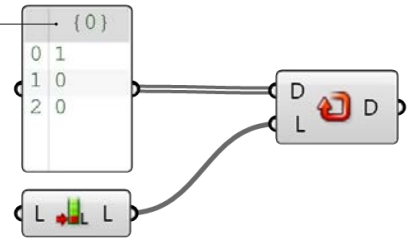
09. Right-Click the first Curve Parameter and select set one Curve. Set the first Square that you are referencing
10. Right-Click the second Curve Parameter and select set one Curve. Set the second Square that you are referencing
11. Vector/Grid/Rectangular – Drag and drop a Rectangular Grid component onto the canvas.
12. Params/Input/Slider - Drag and drop three Numeric Sliders on the canvas
13. Double-click on the first slider and set the following:
 - Rounding: Integers
 - Lower Limit: 0
 - Upper Limit: 10
 - Value: 10
14. Double-click on the second slider and set the following:
 - Rounding: Integers
 - Lower Limit: 0
 - Upper Limit: 10
 - Value: 10
15. Double-click on the third slider and set the following:
 - Name: Extents X & Y
 - Rounding: Integers
 - Lower Limit: 0
 - Upper Limit: 10
 - Value: 10
16. Connect the first Number Slider to the Size X (Sx) input of the Rectangular Grid component
17. Connect the second Number Slider to the Size Y (Sy) input of the Rectangular Grid component
18. Connect the third Number Slider to the Extent X (Ex) input and the Extent Y (Ey) input of the Rectangular Grid component
19. Sets/Tree/Merge – Drag and drop two Merge components onto the canvas
20. Connect the first Geometry parameter to Data Stream 1 (D1) input of the first Merge component
21. Connect the second Geometry parameter to Data Stream 2 (D2) input of the first Merge component
22. Connect the first curve parameter to Data Stream 1 (D1) input of the second Merge component
23. Connect the second curve parameter to Data Stream 1 (D2) input of the second Merge component
24. Right-click the Cells (C) output of the Rectangular Grid component and select Flatten
25. Sets/List/List Length – Drag and drop a List Length component onto the canvas
26. Connect the Cells (C) output of the Rectangular Grid component to the List (L) input of the List Length component
27. Sets/Sequence/Repeat Data – Drag and drop a Repeat Data component onto the canvas
28. Connect the Length (L) output of the List Length component to the Length

Be sure that the geometry and the square that you are referencing correspond.

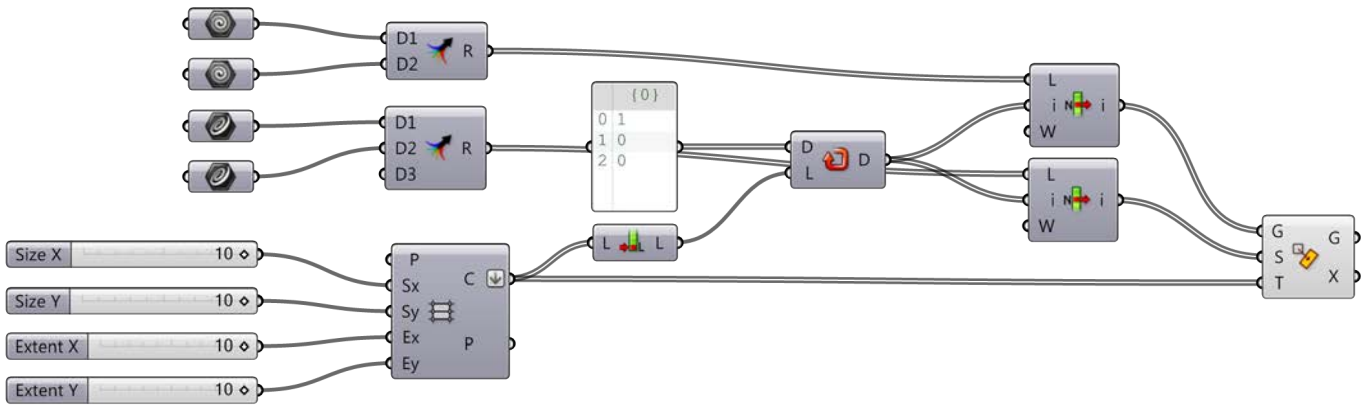


(L) input of the Repeat Data component

29. Params/Input/Panel – Drag and drop a panel onto the canvas
30. Double-click the panel. Deselect multiline data, wrap items, and special codes. Enter the following:
 - 1
 - 0
 - 0
31. Connect the panel to the Data (D) input of the Repeat Data component
32. Sets/List/List Item – Drag and drop two List Item components
33. Connect the Result (R) output of the first merge component to the List (L) input of the first List Item component.
34. Connect the Result (R) output of the second merge component to the List (L) input of the second List Item component.
35. Connect the Data (D) output of the Repeat data component to the Index (i) input of the first and second List Item components.
36. Transform/Affine/Rectangle Mapping – Drag and Drop the Rectangle Mapping component onto the canvas
37. Connect the Cells (C) output of the Rectangular Grid component to the Target (T) input of the Rectangular Mapping Component
38. Connect the items (I) output of the first List item component to the Geometry (G) input of the Rectangular Mapping Component
39. Connect the items (I) output of the second List item component to the Source (S) input of the Rectangular Mapping Component

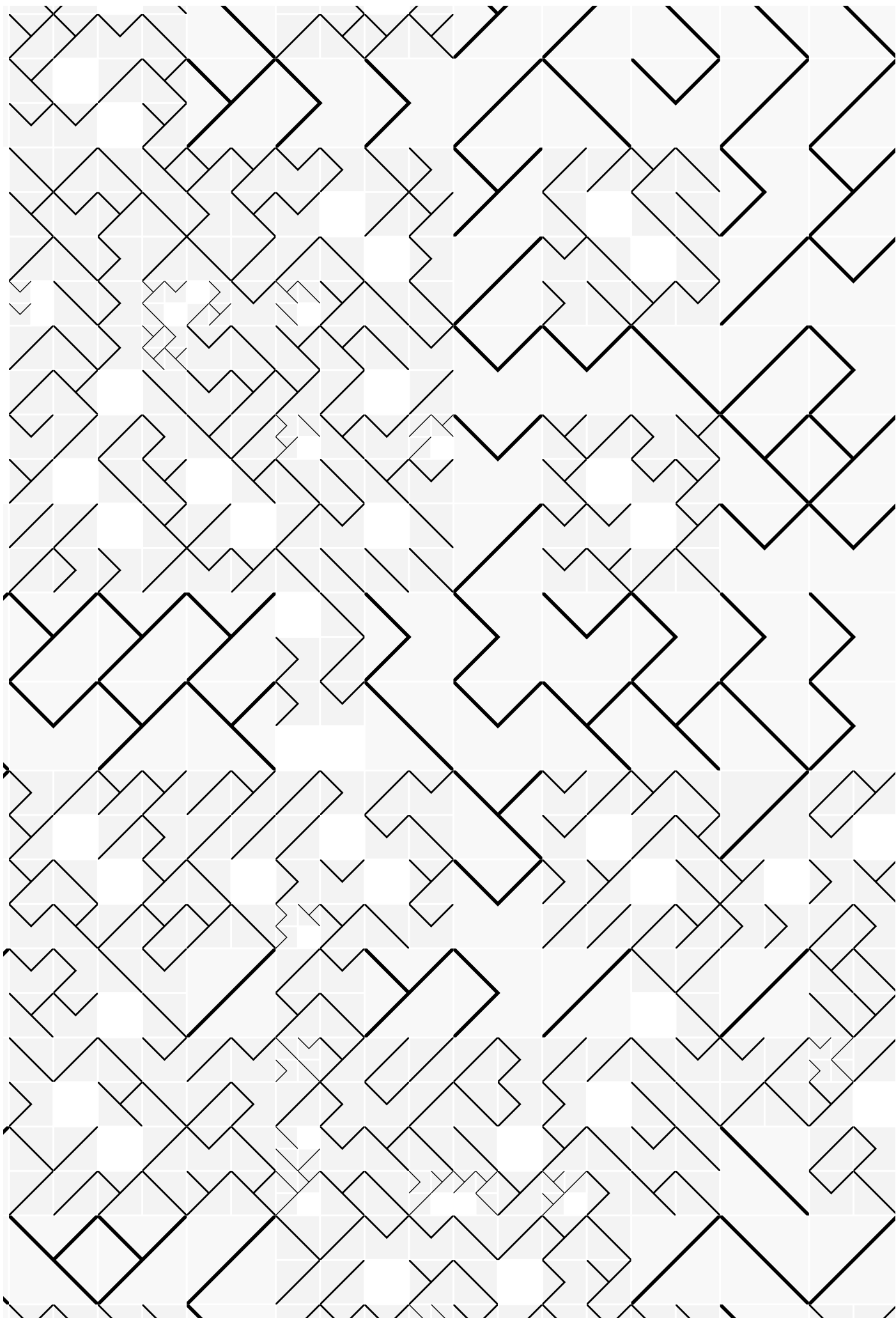


This is the pattern in which the geometries are being distributed. 0 is calling out the first referenced Geometry and 1 is calling out the second referenced Geometry. Changing the number sequence will change the pattern, as will changing the extents of the grid.



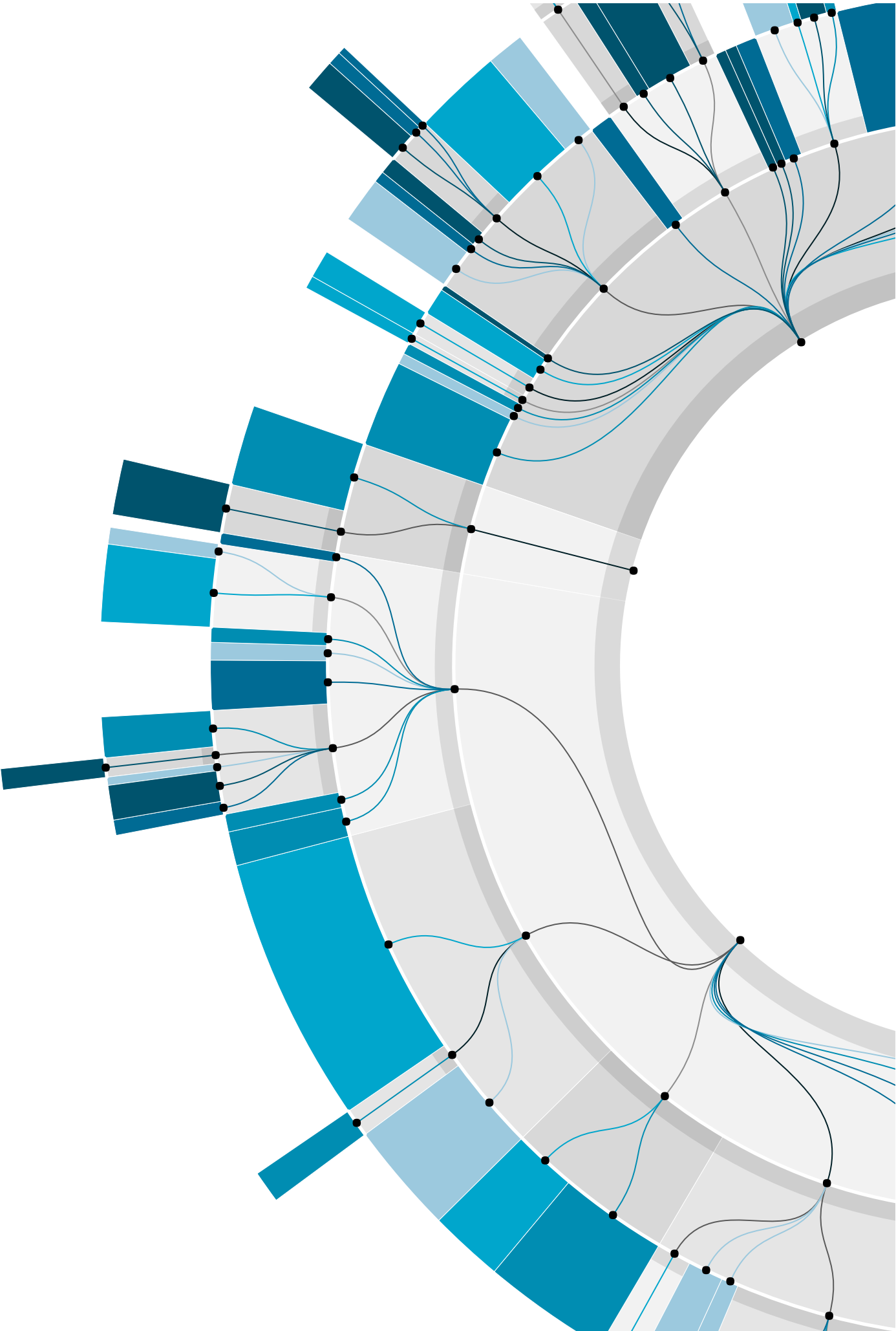
Changing the input geometry and the pattern will change the final tile pattern.

	#	A1,A2	B1,B2	C1,C2	HYBRID																									
	<table border="1"> <tr><td>0</td><td>1</td></tr> </table>	0	1																											
0	1																													
1,1,0...	<table border="1"> <tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr> </table>	0	1	1	0	1	1	1	0	1	1	1	0	1	1	0	0	1	1	0	1	1	1	0	1	1				B1,C2
0	1	1	0	1																										
1	1	0	1	1																										
1	0	1	1	0																										
0	1	1	0	1																										
1	1	0	1	1																										
1,0,0...	<table border="1"> <tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr> </table>	0	1	0	0	1	1	0	0	1	0	0	0	1	0	0	0	1	0	0	1	1	0	0	1	0				A2,C1
0	1	0	0	1																										
1	0	0	1	0																										
0	0	1	0	0																										
0	1	0	0	1																										
1	0	0	1	0																										



F.4 DESIGNING WITH DATA TREES

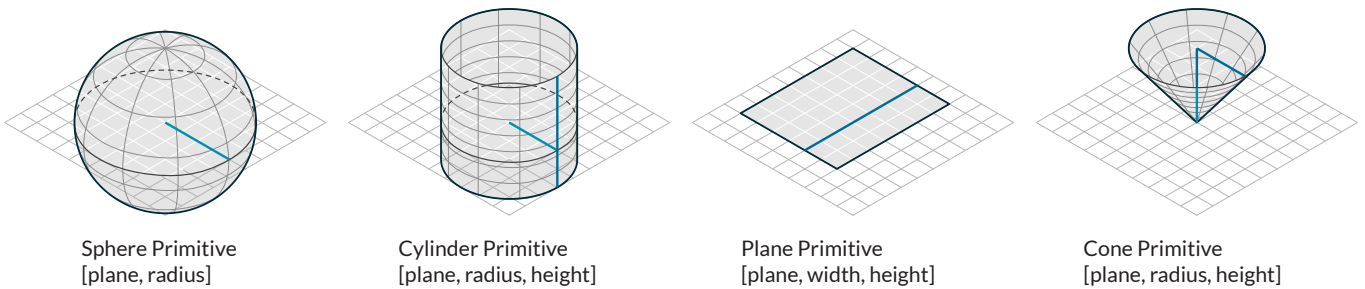
As your definitions increase in complexity, the amount of data flowing through also increases. In order to effectively use Grasshopper, it is important to understand how large quantities of data are stored, accessed, and manipulated.



F.4.0 Surface Geometry

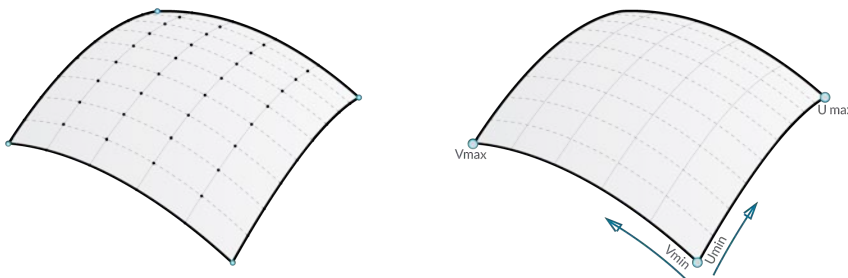
NURBS (non-uniform rational B-splines) are mathematical representations that can accurately model any shape from a simple 2D line, circle, arc, or box to the most complex 3D free-form organic surface or solid. Because of their flexibility and accuracy, NURBS models can be used in any process from illustration and animation to manufacturing.

Apart from a few primitive surface types such as spheres, cones, planes and cylinders, Rhino supports three kinds of freeform surface types, the most useful of which is the NURBS surface. Similar to curves, all possible surface shapes can be represented by a NURBS surface, and this is the default fall-back in Rhino. It is also by far the most useful surface definition and the one we will be focusing on.



F.4.0.0 NURBS SURFACES

NURBS surfaces are very similar to NURBS curves. The same algorithms are used to calculate shape, normals, tangents, curvatures and other properties, but there are some distinct differences. For example, curves have tangent vectors and normal planes, whereas surfaces have normal vectors and tangent planes. This means that curves lack orientation while surfaces lack direction. In the case of NURBS surfaces, there are in fact two directions implied by the geometry, because NURBS surfaces are rectangular grids of {u} and {v} curves. And even though these directions are often arbitrary, we end up using them anyway because they make life so much easier for us.

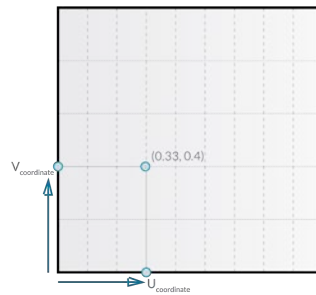
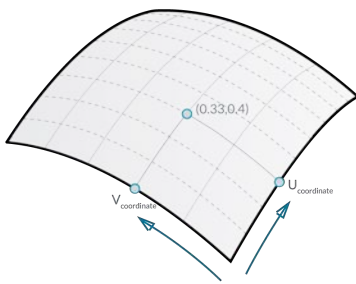


You can think of NURBS surfaces as a grid of NURBS curves that go in two directions. The shape of a NURBS surface is defined by a number of control points and the degree of that surface in the u and v directions. NURBS surfaces are efficient for storing and representing free-form surfaces with a high degree of accuracy.

Surface Domain

A surface domain is defined as the range of (u,v) parameters that evaluate into a 3-D point on that surface. The domain in each dimension (u or v) is usually described as two real numbers (u_min to u_max) and (v_min to v_max) Changing a surface domain is referred to as reparameterizing the surface.

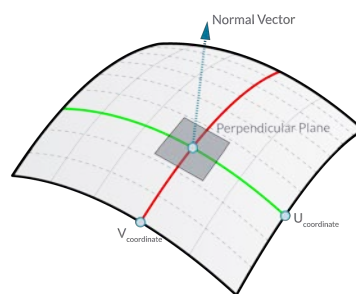
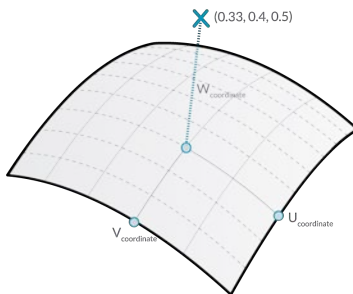
In Grasshopper, it is often useful to reparameterize NURBS surfaces so that the u and v domains both range from 0 to 1. This allows us to easily evaluate and operate on the surface.



Note: Evaluating parameters at equal intervals in the 2-D parameter rectangle does not necessarily translate into equal intervals in 3-D space.

Surface evaluation

Evaluating a surface at a parameter that is within the surface domain results in a point that is on the surface. Keep in mind that the middle of the domain (mid-u, mid-v) might not necessarily evaluate to the middle point of the 3D surface. Also, evaluating u- and v-values that are outside the surface domain will not give a useful result.



Normal Vectors and Tangent Planes

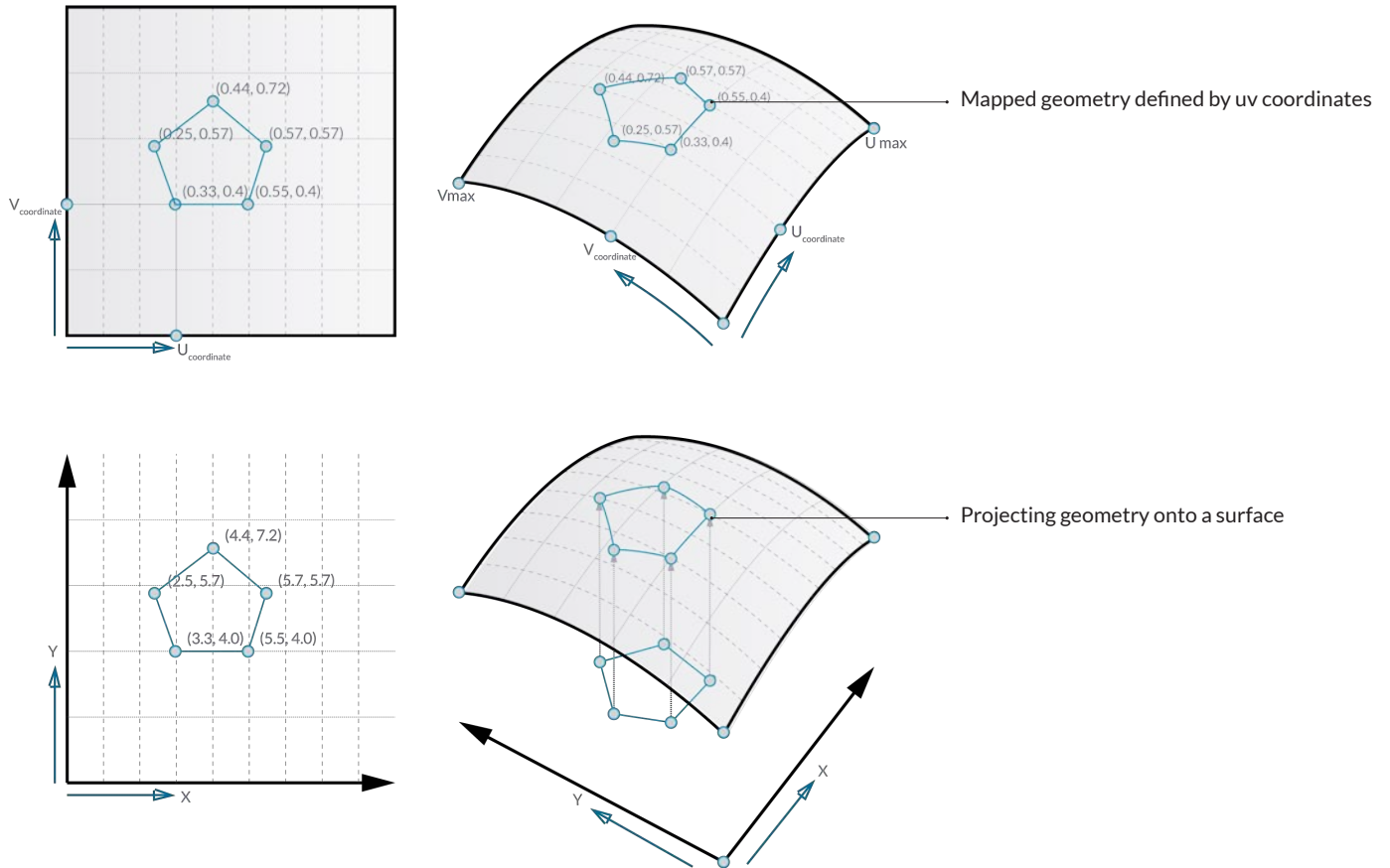
The tangent plane to a surface at a given point is the plane that touches the surface at that point. The z-direction of the tangent plane represents the normal direction of the surface at that point.

Grasshopper handles NURBS surfaces similarly to the way that Rhino does because it is built on the same core of operations needed to generate the surface. However, because Grasshopper is displaying the surface on top of the Rhino viewport (which is why you can't really select any of the geometry created through Grasshopper in the viewport until you bake the results into the scene) some of the mesh settings are slightly lower in order to keep the speed of the Grasshopper results fairly high. You may notice some faceting in your surface meshes, but this is to be expected and is only a result of Grasshopper's drawing settings. Any baked geometry will still use the higher mesh settings.

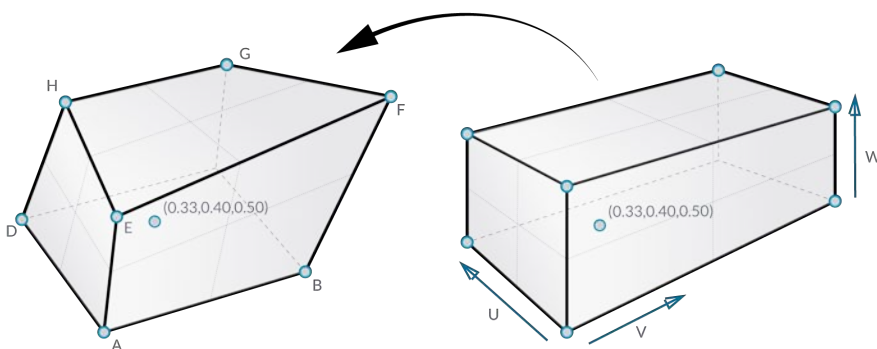
F.4.0.1 PROJECTING, MAPPING & MORPHING

In the previous section, we explained that NURBS surfaces contain their own coordinate space defined by u and v domains. This means that two dimensional geometry that is defined by x and y coordinates can be mapped onto the uv space of a surface. The geometry will stretch and change in response to the curvature of the surface. This is different from simply projecting 2d geometry onto a surface, where vectors are drawn from the 2d geometry in a specified direction until they intersect with the surface.

You can think of projection as geometry casting a shadow onto a surface, and mapping as geometry being stretched over a surface.



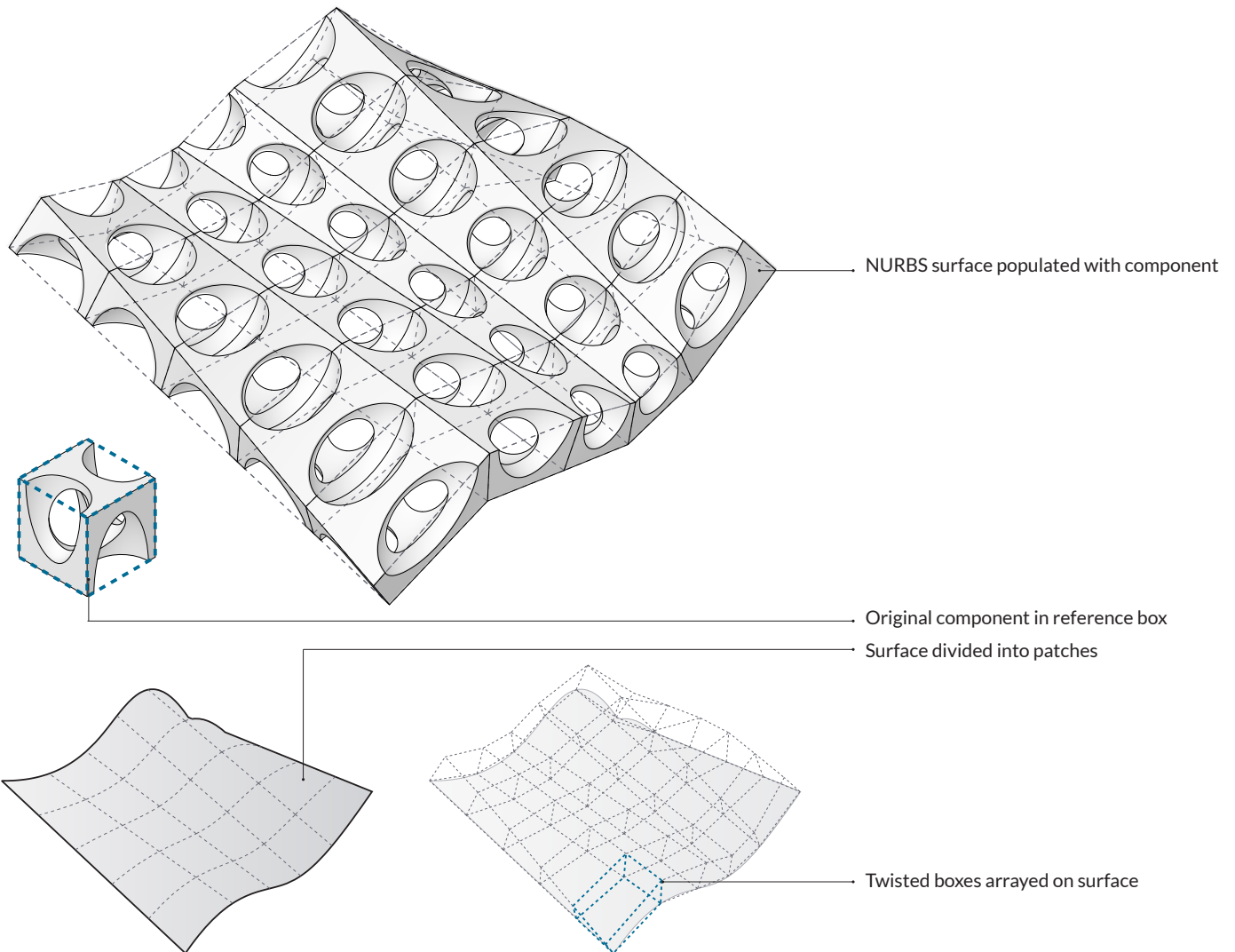
Just as 2d geometry can be projected onto the uv space of a surface, 3d geometry that is contained by a box can be mapped to a corresponding twisted box on a surface patch. This operation is called box morphing and is useful for populating curved surfaces with three dimensional geometric components.



To array twisted boxes on a surface, the surface domain must be divided to create a grid of surface patches. The twisted boxes are created by drawing normal vectors at the corners of each patch to the desired height and creating a box defined by the end points of those vectors and the corner points of the patch.

F.4.0.2 MORPHING DEFINITION

In this example, we will use the box morph component to populate a NURBS surface with a geometric component.



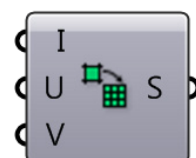
01. Start a new definition, type Ctrl+N (in Grasshopper)
02. Params/Geometry/Surface – Drag and drop a Surface parameter onto the canvas
03. Params/Geometry/Geometry – Drag a Geometry parameter to the canvas
04. Right click the Surface Parameter and select “Set One Surface” – select a surface to reference in the Rhino viewport
05. Right click the Geometry parameter and select “Set One Geometry” – select the your Rhino geometry
06. Maths/Domain/Divide Domain2 – Drag and drop the Divide Domain2 component onto the canvas
07. Params/Input/Number Slider – Drag three Number sliders onto the canvas
08. Double click the first slider and set the following:
 - Rounding: Integer
 - Lower Limit: 0



This is the surface that we will populate with geometric components

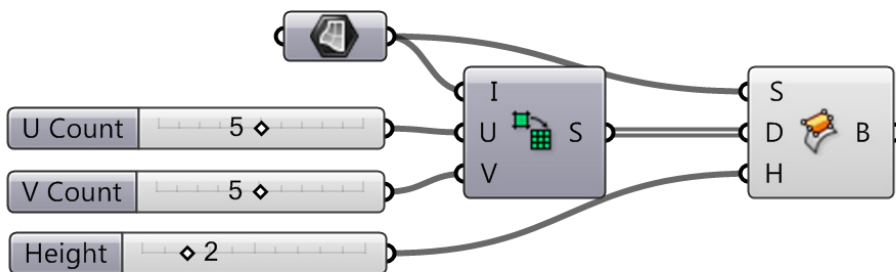


This is the component that will be arrayed over the surface



Upper Limit: 10
Value: 5

09. Set the same values on the second and third sliders
10. Connect the output of the Surface parameter to the Domain (I) input of the Divide Domain2 component
11. Connect the first Number Slider to the U Count (U) input of the Divide Domain2 component
12. Connect the second Number Slider to the V Count (V) input of the Divide Domain2 component
13. Transform/Morph/Surface Box - Drag the Surface Box component to the canvas
14. Connect the output of the Surface parameter to the Surface (S) input of the Surface Box component
15. Connect the Segements (S) output of the Divide Domain2 component to the Domain (D) input of the Surface Box component

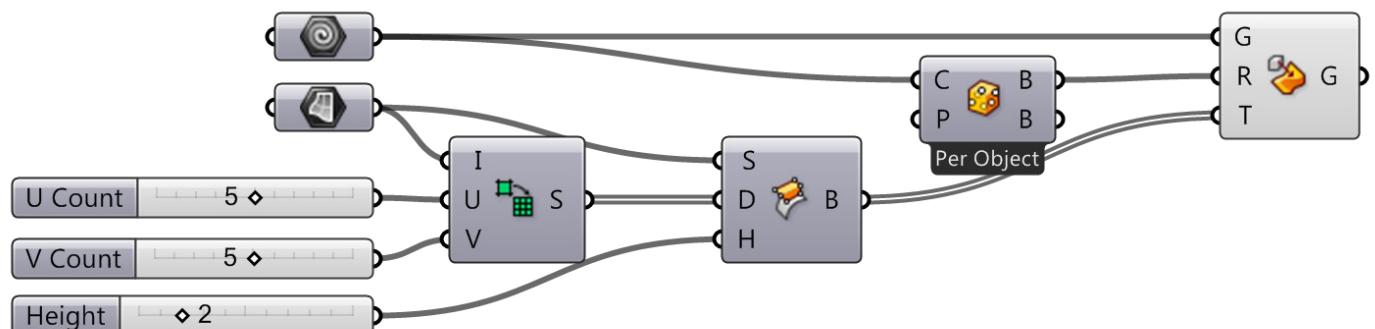


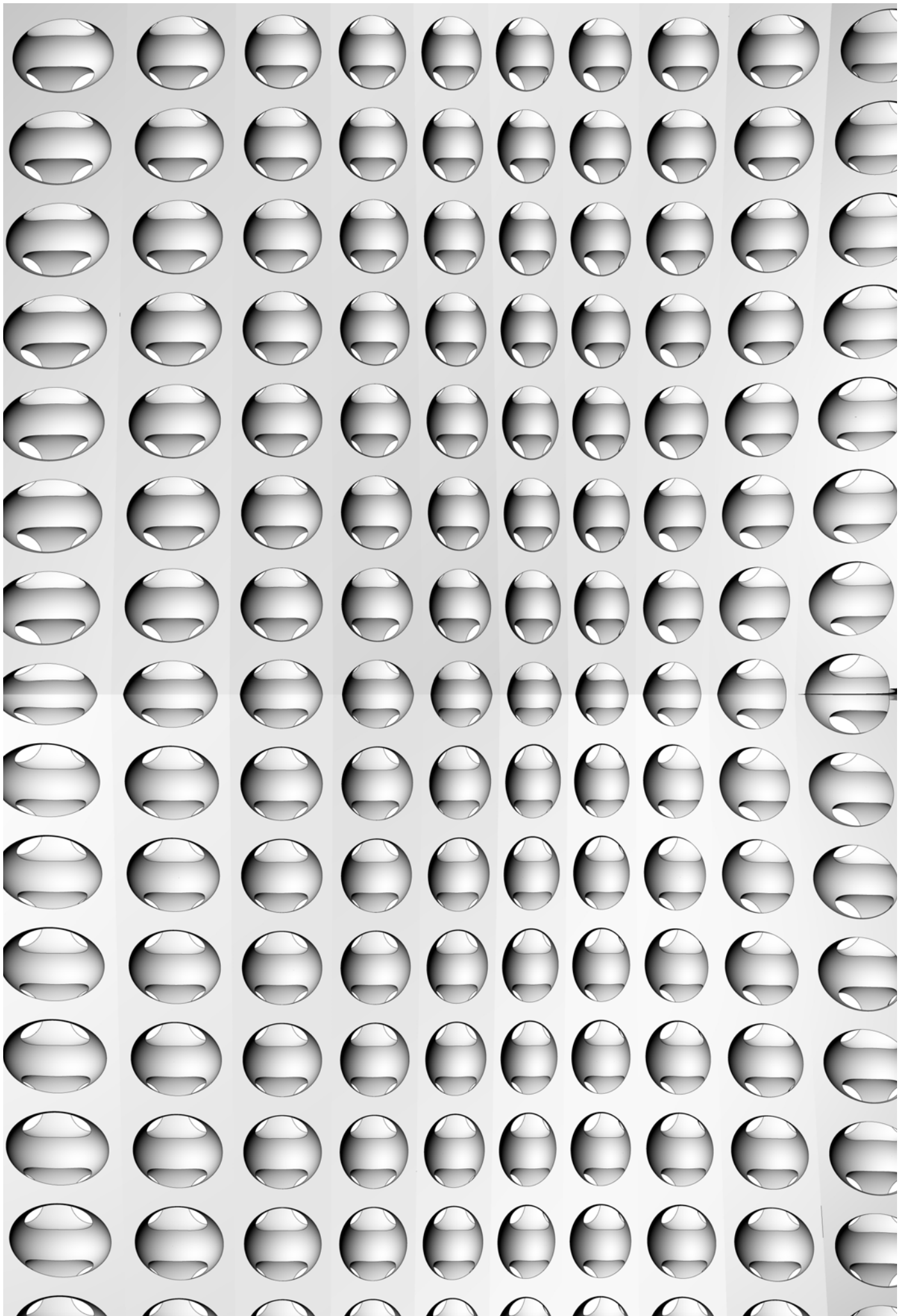
You should see a grid of twisted boxes populating your referenced surface. Change the U and V count sliders to change the number of boxes, and use the height slider to adjust their height.

16. Connect the third Number Slider to the Height (H) input of the Surface Box component
17. Surface/Primitive/Bounding Box - Drag a Bounding Box component to the canvas
18. Transform/Morph/Box Morph - Drag and drop the Box Morph component onto the canvas
19. Connect the output of the Geometry parameter to the Content (C) input of the Bounding Box component
20. Connect the output of the Geometry parameter to the Geometry (G) input of the Box Morph component
21. Connect the Box (B) output of the Bounding Box component to the Reference (R) input of the Box Morph component
22. Connect the Twisted Box (B) output of the Surface Box component to the Target (T) input of the Box Morph component



You should now see your geometry populating your surface.

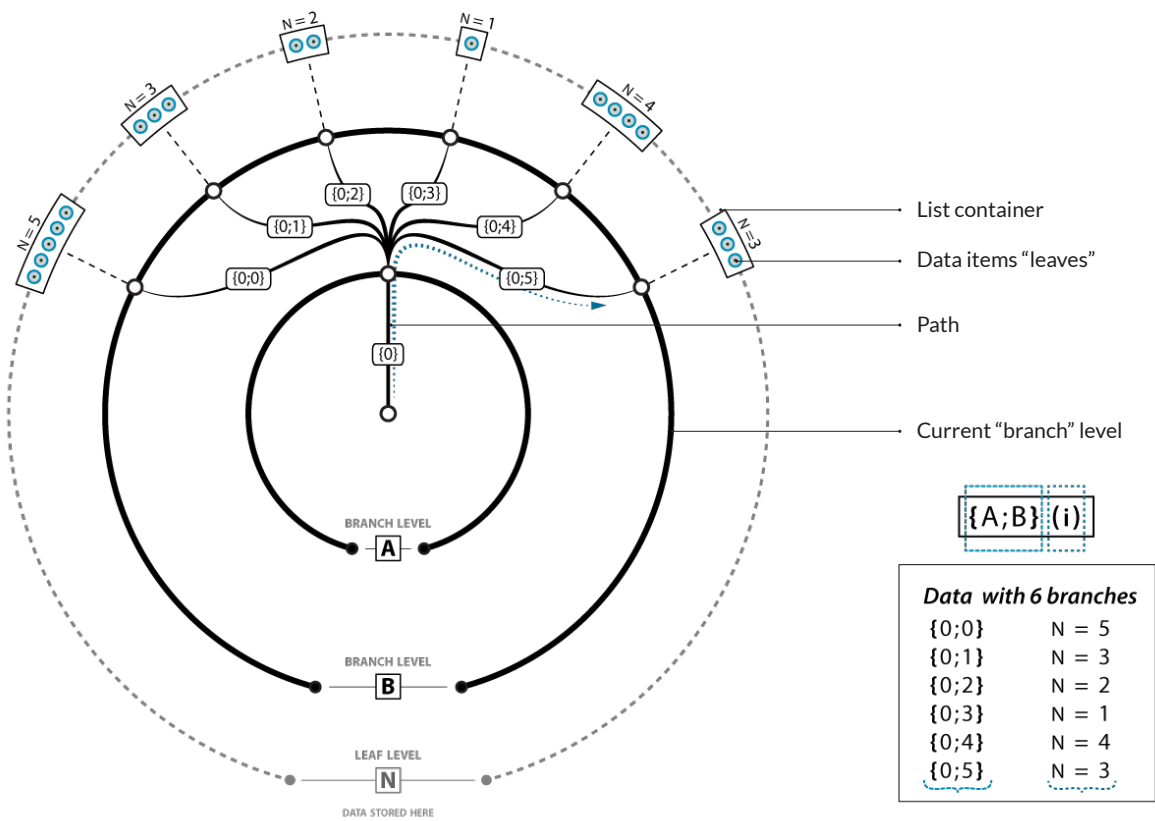




F.4.1 What is a Data Tree?

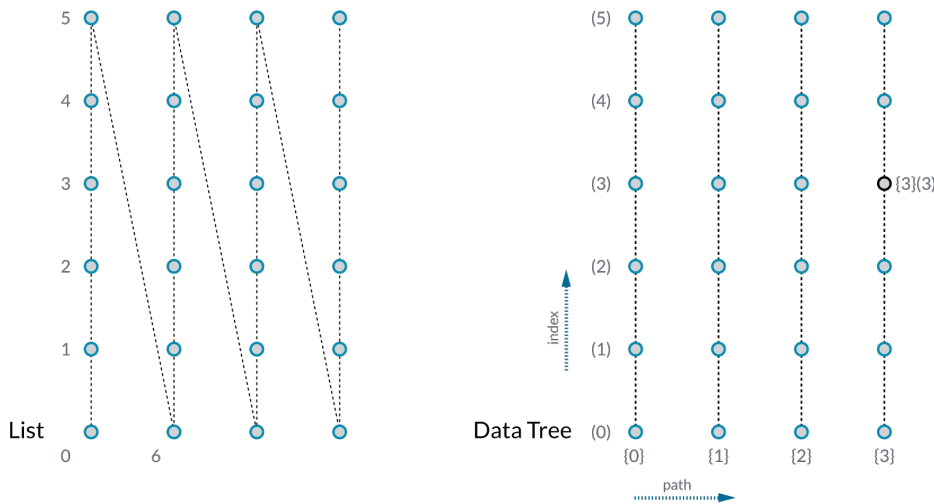
A Data Tree is a hierarchical structure for storing data in nested lists. Data trees are created when a grasshopper component is structured to take in a data set and output multiple sets of data. Grasshopper handles this new data by nesting it in the form of sub-lists. These nested sub-lists work in the same way as folder structures on your computer in that accessing indexed items require moving through paths that are informed by their generation of parent lists and their own sub-index.

It's possible to have multiple lists of data inside a single parameter. Since multiple lists are available, there needs to be a way to identify each individual list. A Data Tree is essentially a list of lists, or sometimes a list of lists of lists (and so on).



In the image above, there is a single master branch (you could call this a trunk, but since it's possible to have multiple master branches, it might be a bit of a misnomer) at path {0}. This path contains no data, but does have 6 sub-branches. Each of these sub-branches inherit the index of the parent branch {0} and add their own sub-index (0, 1, 2, 3, 4, and 5 respectively). It would be wrong to call this an "index", because that implies just a single number. It is probably better to refer to this as a "path", since it resembles a folder-structure on the disk. At each of these sub-branches, we encounter some data. Each data item is thus part of one (and only one) branch in the tree, and each item has an index that specifies its location within the branch. Each branch has a path that specifies its location within the tree.

The image below illustrates the difference between a list and a data tree. On the left, an array of four columns of six points each is all contained in one list. The first column numbered 0-5, the second 6-11, and so on. On the right is the same array of points contained in a data tree. The data tree is a list of four columns, and each column is a list of six points. The index of each point is (column number, row number). This is a much more useful way of organizing this data, because you can easily access and operate on all the points in a given row or column, delete every second row of points, connect alternating points, etc.

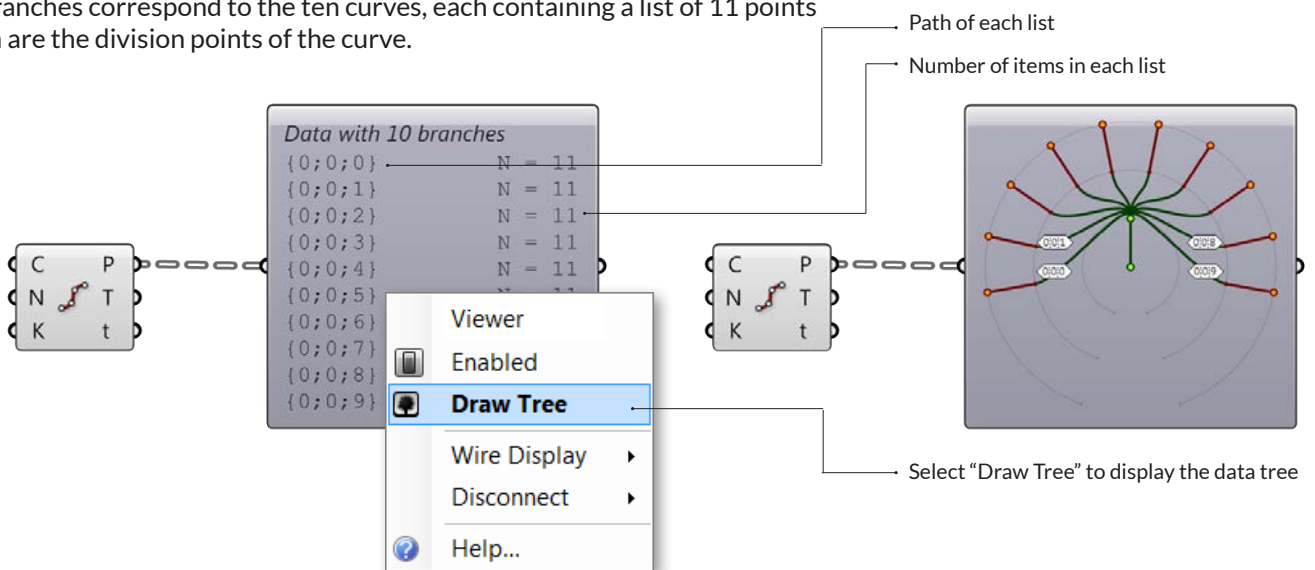


F.4.1.0 DATA TREE VISUALIZATION

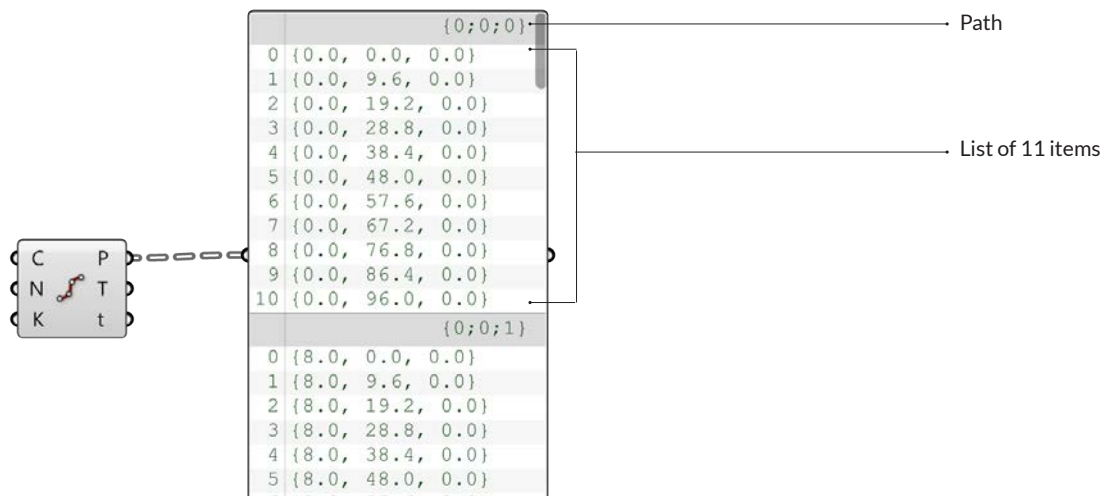
Due to their complexity, Data Trees can be difficult to understand. Grasshopper has several tools to help visualize and understand the data stored in a tree.

The Param Viewer

The Param Viewer (Params/Util/Param Viewer) allows you to visualize data in text form and as a tree. Connect any output containing data to the input of the Param Viewer. To show the tree, right-click the Param Viewer and select "draw tree." In this example, the Param Viewer is connected to the Points (P) output of a Divide Curve component that divided 10 curves into 10 segments each. The ten branches correspond to the ten curves, each containing a list of 11 points which are the division points of the curve.



If we connect a panel to the same output, it displays ten lists of 11 items each. You can see that each item is a point defined by three coordinates. The path is displayed at the top of each list, and corresponds to the paths listed in the Param Viewer.

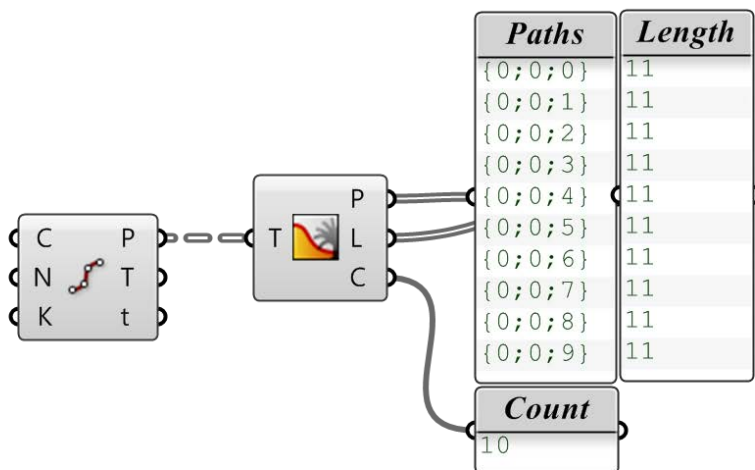


Tree Statistics

The Tree Statistics component (Sets/Tree/Tree Statistics) Returns some statistics of the Data Tree including:

- P - All the paths of the tree
- L - The length of each branch in the tree
- C - Number of paths and branches in the tree

If we connect the Points output of the same Divide Curve component, we can display the paths, lengths, and the count in panels. This component is helpful because it separates the statistics into three outputs, allowing you to view only the one that is relevant.



Both the Param Viewer and the Tree Statistics component are helpful for visualizing changes in the structure of the Data Tree. In the next section, we will look at some operations that can be performed to change this structure.

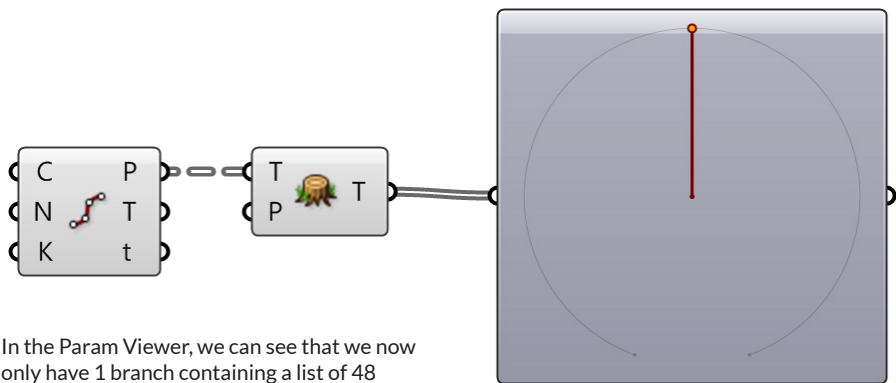
F.4.3 Working with Data Trees

Grasshopper contains tools for changing the structure of a data tree. These tools can help you access specific data within a tree, and change the way it is stored, ordered, and identified.

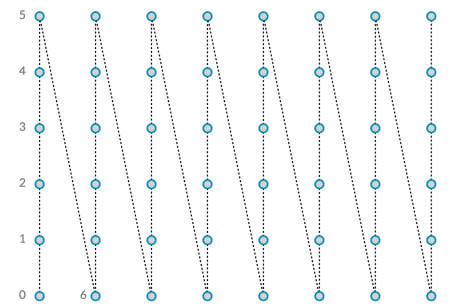
Let's look at some data tree manipulations and visualize how they affect the tree.

F.4.3.0 FLATTEN TREE

Flattening removes all levels of a Data Tree, resulting in a single List. Using the Flatten component (Sets/Tree/Flatten) on the P output of our Divide Curve component, we can use the Param Viewer to visualize the new data structure.

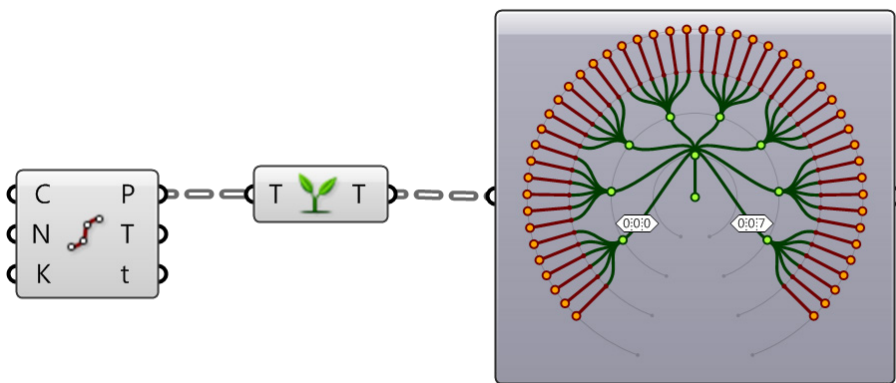


In the Param Viewer, we can see that we now only have 1 branch containing a list of 48 points.

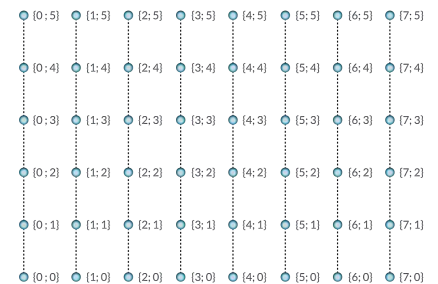


F.4.3.1 GRAFT TREE

Grafting creates a new Branch for every Data Item. If we run the data through the Graft Tree component (Sets/Tree/Graft Tree), each division point now has its own individual branch, rather than sharing a branch with the other division points on the same curve.

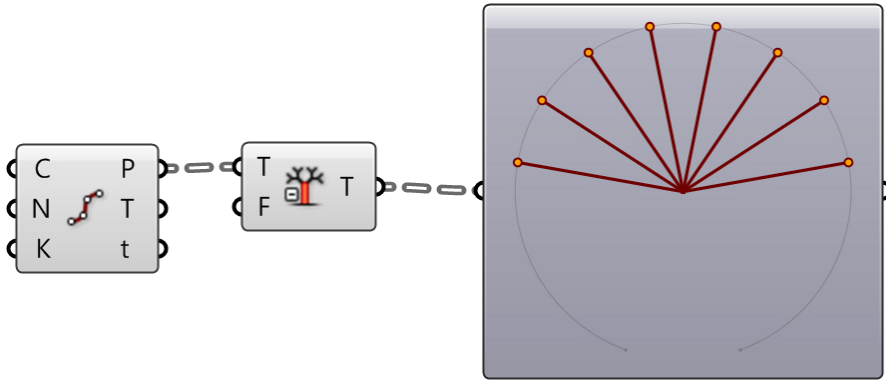


In the Param Viewer, we can see that what was data with 8 branches of 6 items each, we now have 8 branches with 6 sub-branches containing 1 item each.



F.4.3.2 SIMPLIFY TREE

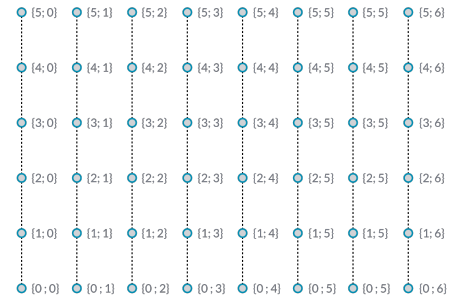
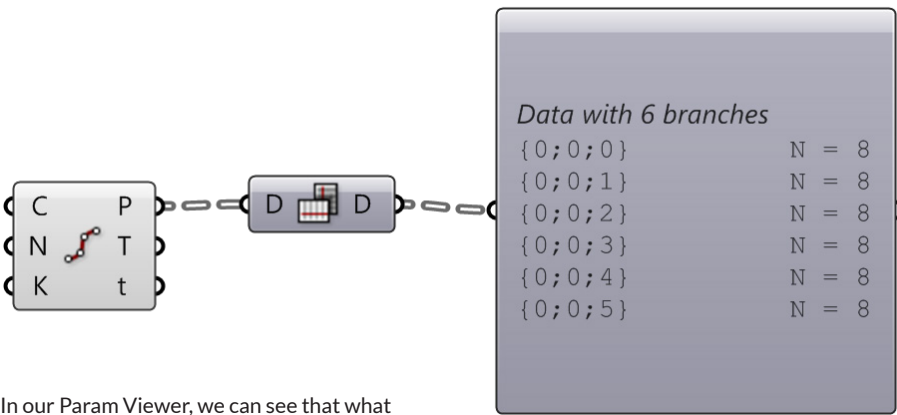
Simplify removes overlapping Branches in a Data Tree. If we run the data through the Simplify Tree component (Sets/Tree/Simplify Tree), the first branch, containing no data, has been removed.



In the Param Viewer, we still have 8 branches of 6 items each, but the first branch has been removed.

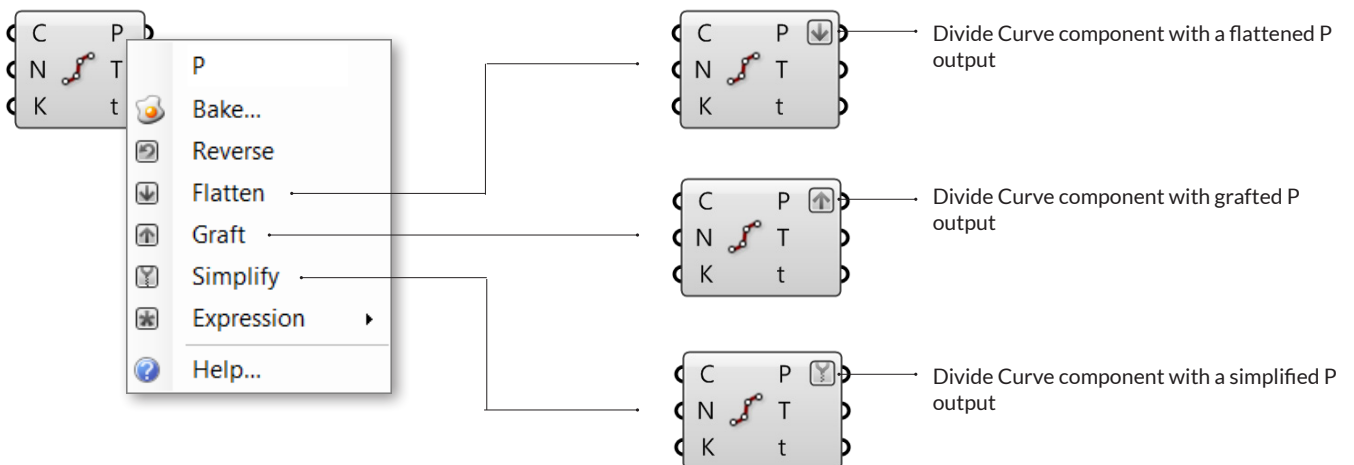
F.4.3.3 FLIP MATRIX

The Flip Matrix component (Sets/Tree/Flip Matrix) Swaps the “Rows” and “Columns” of a Data Tree with two Path Indices.



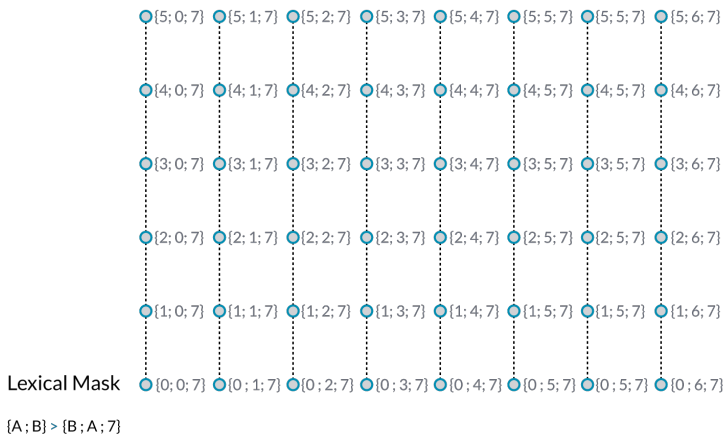
In our Param Viewer, we can see that what was data with 8 branches of 6 items each, we now have 6 branches with 8 items each.

The Flatten, Graft, and Simplify operations can be applied to the component input or output itself, rather than feeding the data through a separate component. Just right-click the desired input or output and select Flatten, Graft, or Simplify from the menu. The component will display an icon to indicate that the tree is being modified. Keep in mind Grasshopper’s program flow. If you flatten a component input, the data will be flattened before the component operation is performed. If you flatten a component output, the data will be flattened after the component performs its action.

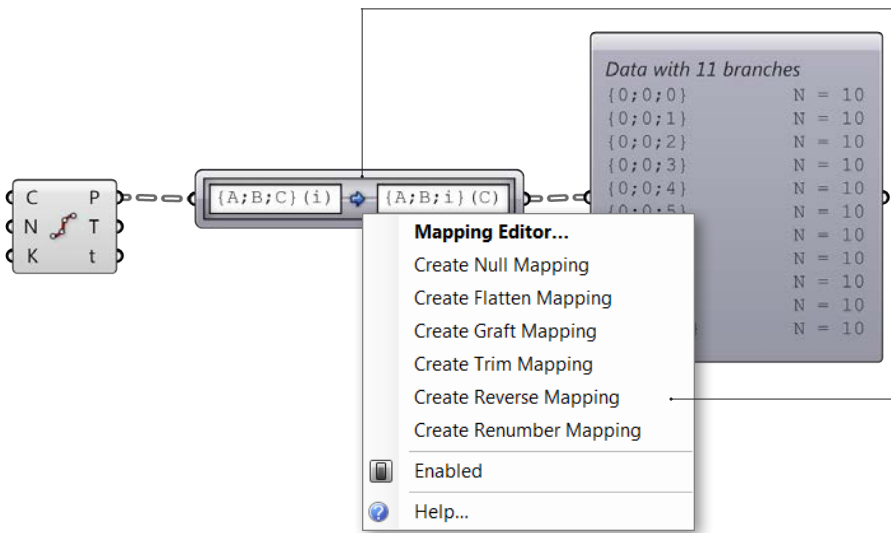


F.4.3.4 THE PATH MAPPER

The Path Mapper component (Sets/Tree/Path Mapper) allows you to perform lexical operations on data trees. Lexical operations are logical mappings between data paths and indices which are defined by textual (lexical) masks and patterns.



The Path Mapper component



Right-click the Path Mapper component and select a predefined mapping option from the menu, or open the mapping editor.

Lexer Combo Editor

Source	Target
{A;B;C}(i)	{A;B;i}(C)

You can use the following constants:

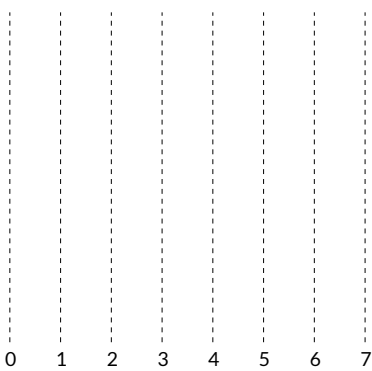
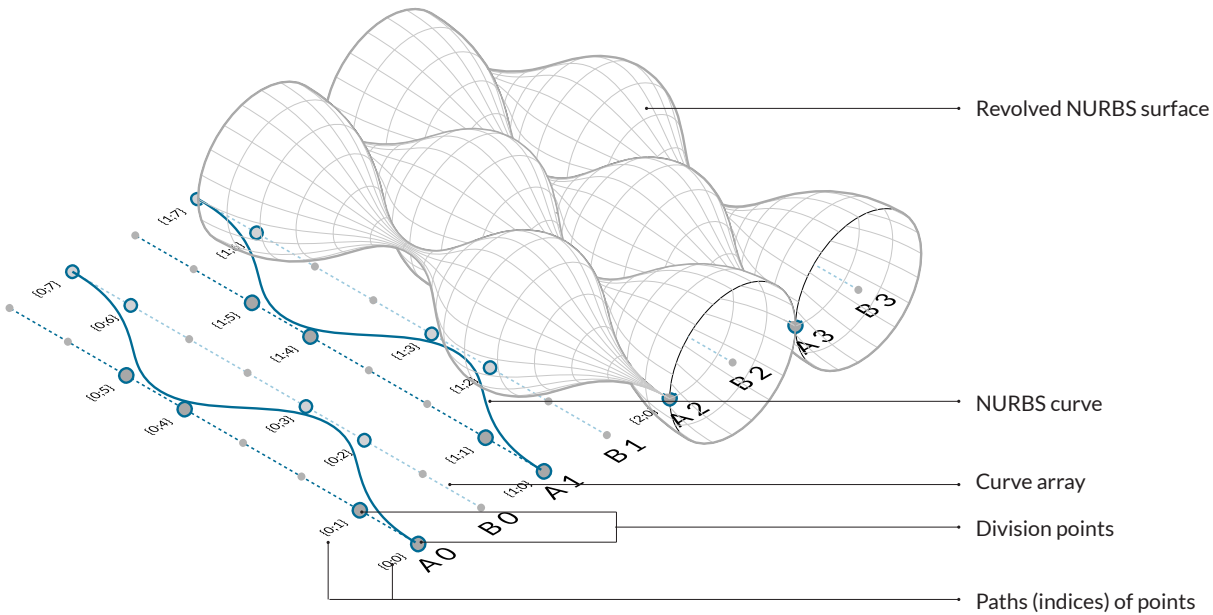
- item_count = number of items in the current branch
- path_count = number of paths (branches) in the tree
- path_index = index of current path

The Mapping Editor

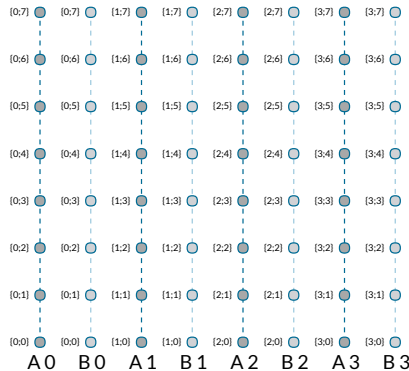
You can modify a data tree by re-mapping the path index and the desired branch.

F.4.3.5 WEAVING DEFINITION

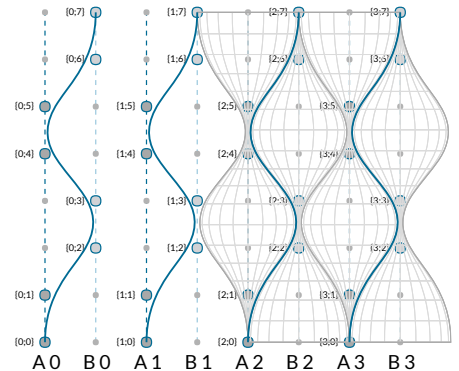
In this example, we will manipulate lists and data trees to weave lists of points, define a pattern, and create surface geometry.



Array curves

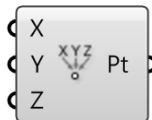
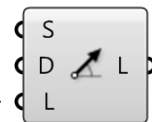


Dispatch curves into lists A and B, divide curves



Cull points, weave, and revolve

01. Start a new definition, type Ctrl+N (in Grasshopper)
02. Curve/Primitive/Line SDL – Drag and drop the Line SDL component onto the canvas
03. Vector/Point/Construct Point – Drag and drop the Construct Point component onto the canvas
04. Connect the Point (Pt) output of the Construct Point component to the Start (S) Input of the Line SDL component
05. Vector/Vector/Unit Y – Drag and drop the vector Unit Y component onto the canvas
06. Connect the Unit Y component to the Direction (D) input of the Line SDL component
07. Params/Input/Number Slider – Drag and drop the Number Slider component onto the canvas



The factor of Unit Vector components is 1.0 by default

08. Double-click on the Number slider and set the following:

Name: Length
 Rounding: Integer
 Lower Limit: 0
 Upper Limit: 96
 Value: 96

09. Connect the Number slider to the Length (L) input of the Line SDL component

10. Transform/Array/Linear Array – Drag and drop the Linear Array component onto the canvas

11. Connect the Line (L) output of the Line SDL component to the Geometry (G) input of the Linear Array component

12. Vector/Vector/Unit X – Drag and drop the Vector Unit X component onto the canvas

13. Params/Input/Number Slider – Drag and drop two Number Slider components onto the canvas

14. Double-click on the first slider and set the following:

Name: Offset Distance
 Rounding: Integer
 Lower Limit: 1
 Upper Limit: 10
 Value: 4

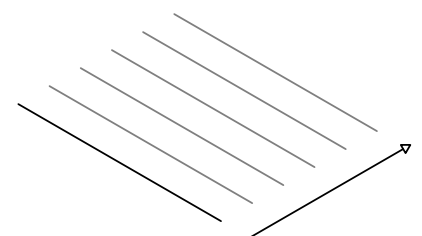
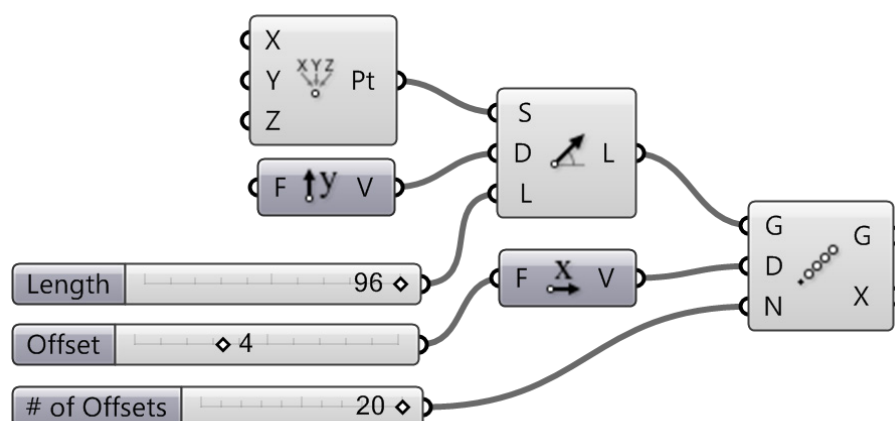
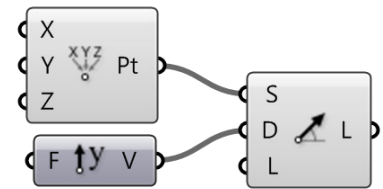
15. Double-click on the second slider and set the following:

Name: # of Offsets
 Rounding: Even
 Lower Limit: 2
 Upper Limit: 20
 Value: 20

16. Connect the Number Slider (Offset Distance) to the Factor (F) input of the Unit X component

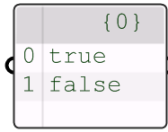
17. Connect the Vector (V) output of the Unit X component to the Direction (D) input of the Linear Array component

18. Connect the Number Slider (# of Offsets) to the Count (N) input of the Linear Array component

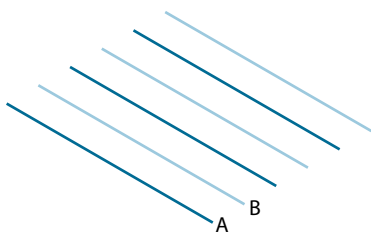
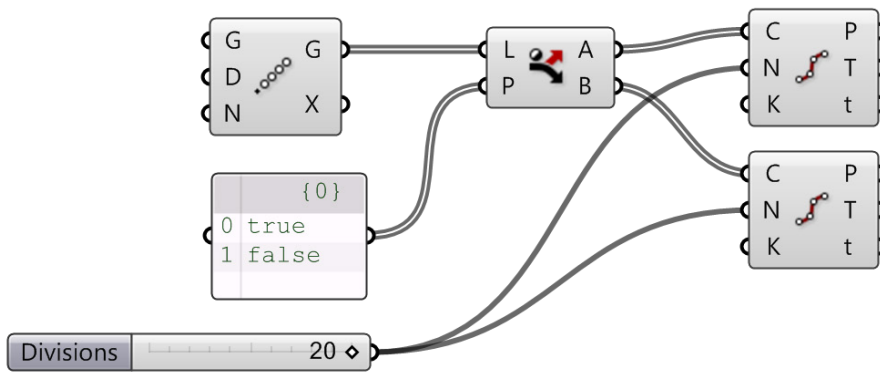
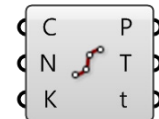


You should now see an array of lines in the Rhino viewport. The three sliders allow you to change the length of the lines, their distance from each other, and the number of lines in the array.

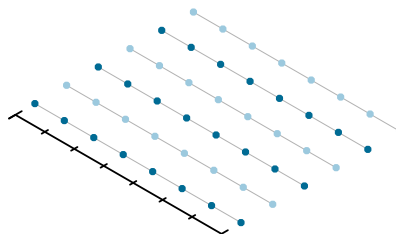
19. Sets/Lists/Dispatch – Drag and drop the component onto the canvas
20. Connect the Geometry (G) output of the Linear Array component to the List (L) input of the Dispatch component
21. Params/Input/Panel – Drag and drop the component onto the canvas
22. Double-click the panel and enter the following:
 - true
 - false
23. Connect the Panel to the Pattern (P) input of the Dispatch component
24. Curve/Division/Divide Curve – Drag and drop two Divide Curve components onto the canvas
25. Connect the List A (A) output of the Dispatch component to the Curve (C) input of the First Divide Curve component
26. Connect the List B (B) output of the Dispatch component to the Curve (C) input of the Second Divide Curve component
27. Params/Input/Number Slider – Drag and drop the Number Slider component onto the canvas
28. Double-click on the Number slider and set the following:
 - Name: Divisions
 - Rounding: Integer
 - Lower Limit: 0
 - Upper Limit: 20
 - Value: 20
29. Connect the Number Slider (Divisions) to the Count (N) input of both Divide Curve components.



Be sure to select: Multiline Data, Wrap Items, and Special Codes

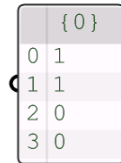


The dispatch component sends every second curve in the array to a separate list.



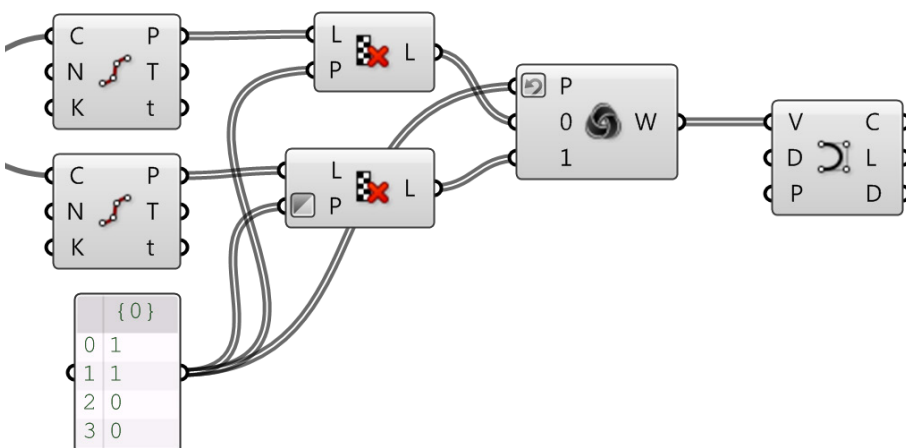
The Divide Curve component divides the curves into the number of segments specified by the slider. Adjust the slider to change the number of points.

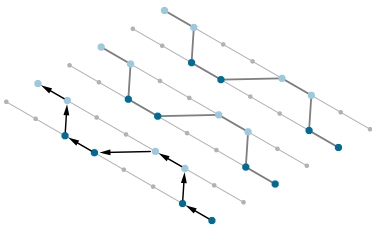
30. Sets/Sequence/Cull Pattern – Drag and drop two Cull Pattern components onto the canvas
31. Connect the Points (P) output of the First Divide Curve component to the List (L) input of the First Cull Pattern component
32. Connect the Points (P) output of the Second Divide Curve component to the List (L) input of the Second Cull Pattern component
33. Params/Input/Panel – Drag and drop a Second Panel component onto the canvas
34. Double-click the Second panel and deselect: Multiline Data, Wrap Items, and Special Codes. Then enter the following:
 - 1
 - 1
 - 0
 - 0
35. Connect the Second Panel to the Pattern (P) input of the First Cull Pattern component
36. Connect the Second Panel to the Pattern (P) input of the Second Cull Pattern component
37. Right-click on the Pattern (P) input of the Second Cull Pattern component and select Invert
38. Sets/List/Weave – Drag and drop the Weave component onto the canvas
39. Connect the Second Panel to the Pattern (P) input of the Weave component
40. Right-click the Pattern (P) input of the Weave component and select reverse
41. Connect the List (L) output of the First Cull Pattern component to the Stream 0 (0) input of the Weave component
42. Connect the List (L) output of the Second Cull Pattern component to the Stream 0 (0) input of the Weave component
43. Curve/Spline/Nurbs Curve – Drag and drop the Nurbs Curve component onto the canvas
44. Connect the Weave (W) output of the Weave component to the Vertices (V) input of the Nurbs Curve component.



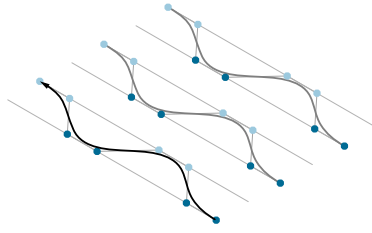
We are using 1 and 0 in place of true and false. These are the two syntaxes that Grasshopper accepts for boolean values.

This will invert the Cull Pattern, a useful trick to keep definitions short.





The cull patterns remove alternating points from each list.

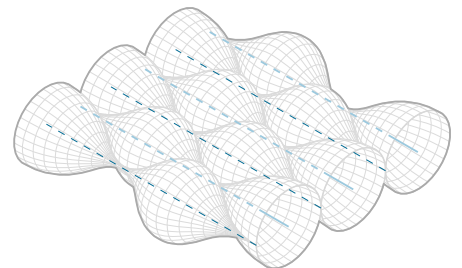
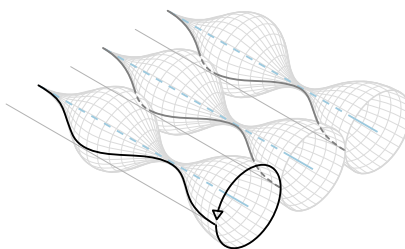
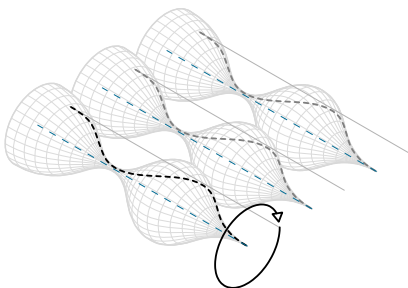
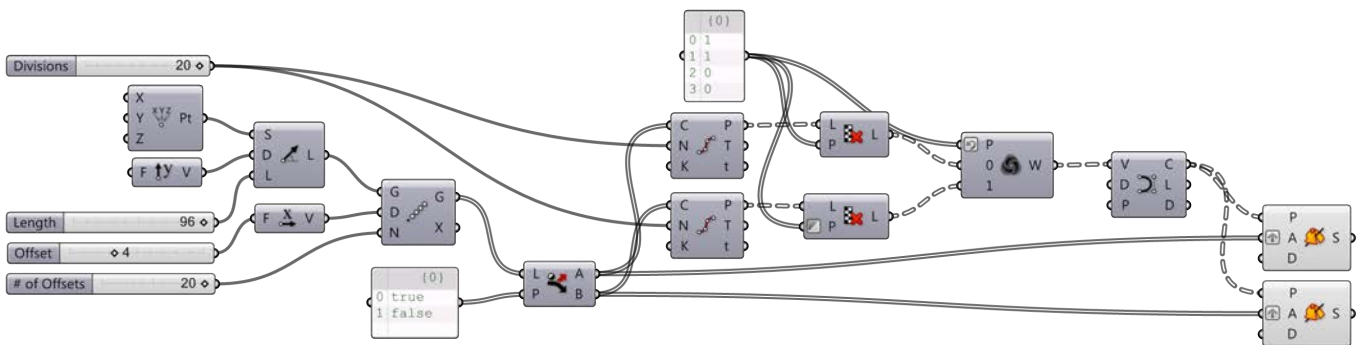


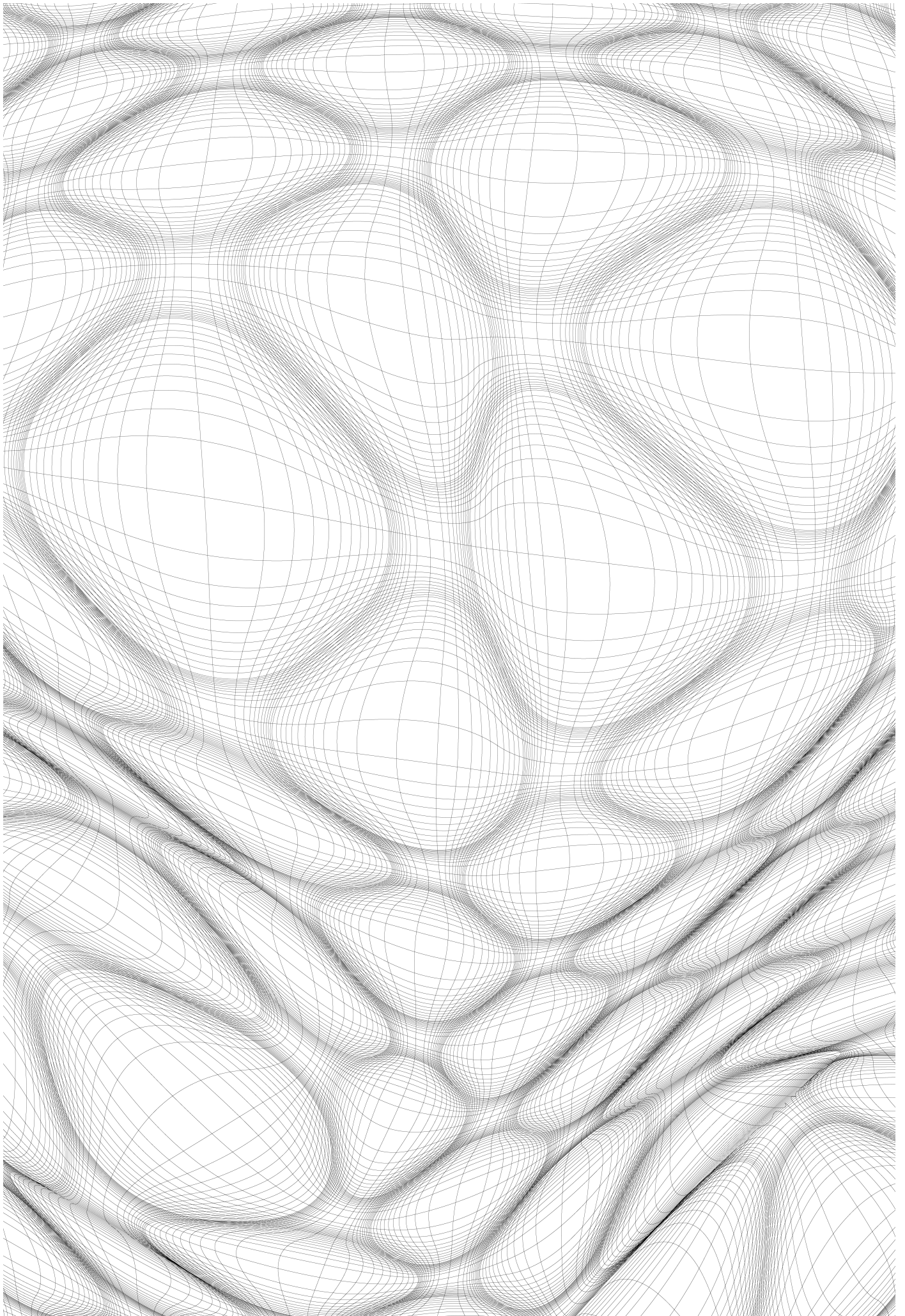
The weave component collects data from the point lists according to a custom pattern. This data is fed into the interpolate component to create curves.

45. Surface/Freeform/Revolution – Drag and drop two Revolution components onto the canvas
46. Connect the Curve output of the Nurbs Curve component to the Profile Curve (P) input of both Revolution components.
47. Right Click on Axis (A) input of both Revolution components and select Graft.
48. Connect the List A (A) output of the Dispatch component to the Axis (A) input of the First Revolution component
49. Connect the List B (B) output of the Dispatch component to the Axis (A) input of the Second Revolution component



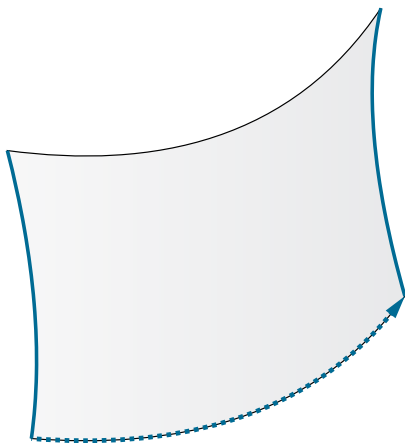
Select all the components except the two Revolution components and turn the preview off - it is helpful to turn previews off as you build the definition to focus on the most recent geometry



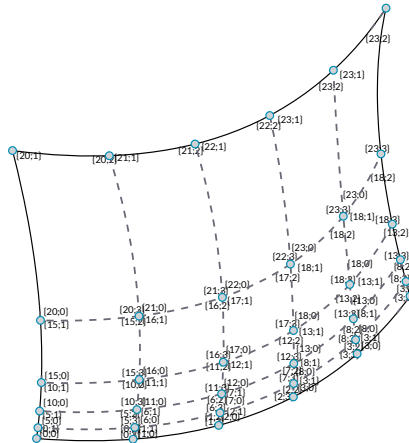


F.4.3.6 RAIL INTERSECT DEFINITION

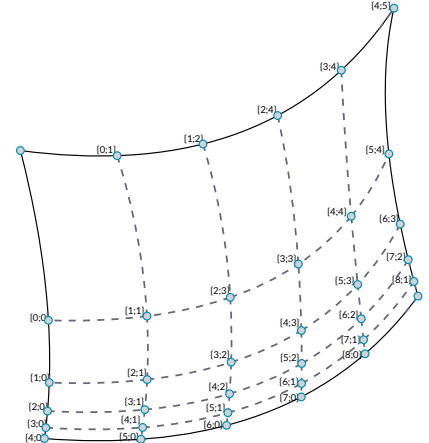
In this example, we will use some of Grasshopper's tools for manipulating data trees to retrieve, reorganize, and interpolate the desired points contained in a data tree and create a lattice of intersecting fins.



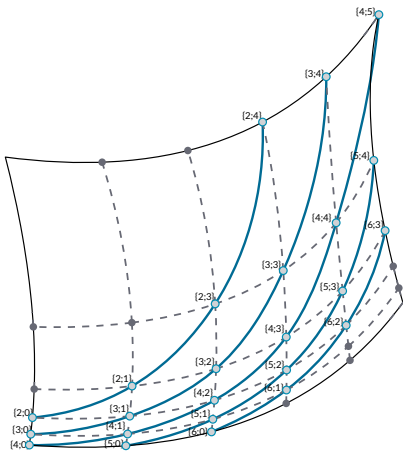
Sweep with two rails to create a NURBS surface



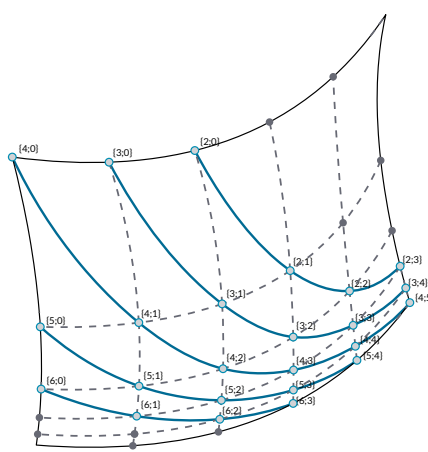
Divide the surface into variable sized segments, extract vertices. Data comprised of one list with four items for each segment.



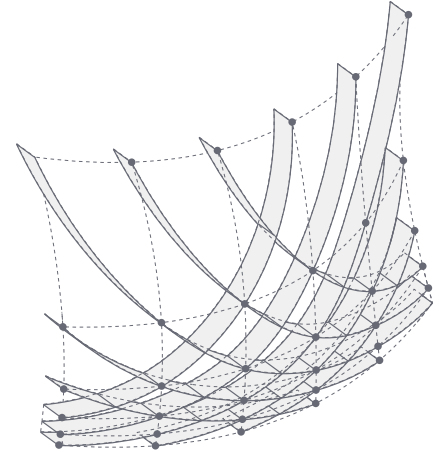
Flip the Matrix to change the data structure. Data comprised of four lists, each containing a single corner point of each segment.



Explode the tree to connect corner points and draw diagonal lines across each segment.

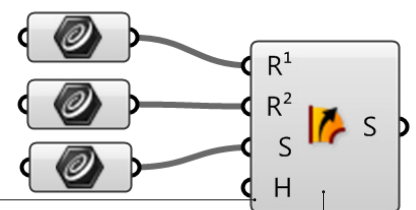


Prune the tree to cull branches containing insufficient points to construct a degree 3 NURBS curve and interpolate points.



Extrude the curves to create intersecting fins.

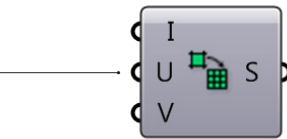
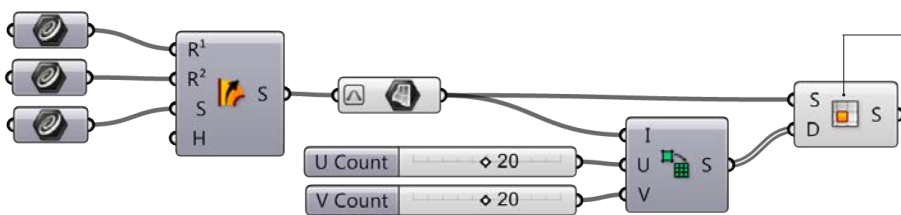
01. Start a new definition, type Ctrl+N (in Grasshopper)
02. Params/Geometry/Curve – Drag and drop three curve parameters onto the canvas
03. Surface/Freeform/Sweep2 – Drag a Sweep2 component onto the canvas
04. Right-click the first curve parameter and select “Set one curve.” Select the first rail curve in the Rhino viewport
05. Right-click the second curve parameter and select “Set one curve.” Select the second rail curve in the Rhino viewport
06. Right-click the third curve parameter and select “Set one curve.” Select the section curve in the Rhino viewport
07. Connect the outputs of the curve parameters to the Rail 1 (R1), Rail 2 (R2), and Sections (S) inputs respectively
08. Params/Geometry/Surface – drag a surface parameter to the canvas
09. Connect the Brep (S) output of the Sweep2 component to the input of the surface parameter



We have just created a NURBS surface

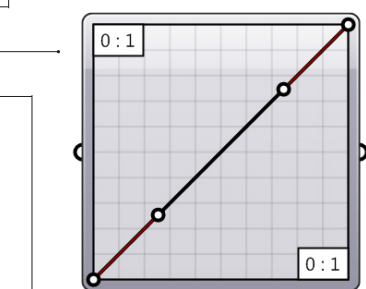
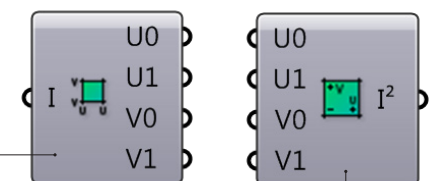
10. Right-click the surface parameter and select "Reparameterize"
11. Maths/Domain/Divide Domain2 – drag and drop a Divide Domain2 component onto the canvas
12. Params/Input/Number Slider – drag two number sliders onto the canvas
13. Double click the first slider and set the following:
 - Rounding: Integer
 - Lower Limit: 1
 - Upper Limit: 40
 - Value: 20
14. Set the same values on the second slider
15. Connect the output of the reparameterized surface parameter to the Domain (I) input of the Divide Domain2 component
16. Connect the first number slider to the U Count (U) input of the Divide Domain2 component
17. Connect the second number slider to the V Count (V) input of the Divide Domain2 component
18. Surface/Util/Isotrim – Drag and drop the Isotrim component onto the canvas
19. Connect the Segments (S) output of the Divide Domain2 component to the Domain (D) input of the Isotrim component
20. Connect the output of the surface parameter to the Surface (S) input of the Isotrim component

In this step, we re-mapped the u and v domains of the surface between 0 and 1. This will make future operations possible.



We have now divided out surface into smaller, equally sized, surfaces. Adjust the U and V Count sliders to change the number of divisions. Lets add a Graph Mapper to give the segments variable size.

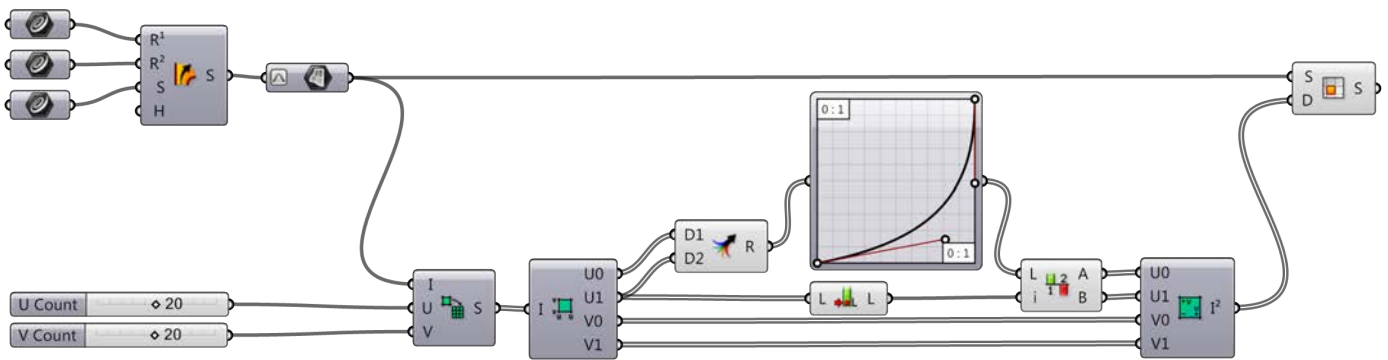
21. Maths/Domain/Deconstruct Domain2 – Drag a Deconstruct Domain2 component onto the canvas
22. Maths/Domain/Construct Domain2 – Drag a Construct Domain2 component to the canvas
23. Params/Input/Graph Mapper – Drag a Graph Mapper to the canvas
24. Sets/List/List Length – Drag a List Length component to the canvas
25. Sets/Tree/Merge – Drag a Merge component to the canvas
26. Sets/List/Split List – Drag a Split List component to the canvas
27. Connect the U min (U0) and U max (U1) outputs of the Deconstruct Domain2 component to the Data 1 (D1) and Data 2 (D2) inputs of the Merge component
28. Connect the Result (R) output of the Merge component to the input of the Graph Mapper
29. Right-click the Graph Mapper and select "Bezier" under "Graph Types"
30. Connect a second wire from the U max (U1) output of the Deconstruct Domain2 component to the List (L) input of the List Length component
31. Connect the Graph Mapper output to the List (L) input of the Split List component



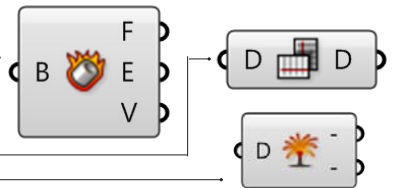
The Merge and Split components are used here so that the same Graph Mapper could be used for both the U min and U max values.

32. Connect the Length (L) output of the List Length component to the Index (i) input of the Split List component
33. Connect the List A (A) output of the Split List component to the U min (U0) input of the Construct Domain2 component
34. Connect the List B (B) output of the Split List component to the U max (U1) input of the Construct Domain2 component
35. Connect the V min (V0) output of the Deconstruct Domain2 component to the V min (V1) input of the Construct Domain2 component
36. Connect the V max (V1) output of the Deconstruct Domain2 component to the V max (V1) input of the Construct Domain2 component
37. Connect the 2D Domain (I2) output of the Construct Domain2 component to the Domain (D) input of the Isotrim component, replacing the existing connection

We have just deconstructed the domains of each surface segment, remapped the U values using a Graph Mapper, and reconstructed the domains. Adjust the grips of the Graph Mapper to change the distribution of the surface segments. Let's use Data Trees to manipulate the surface divisions.

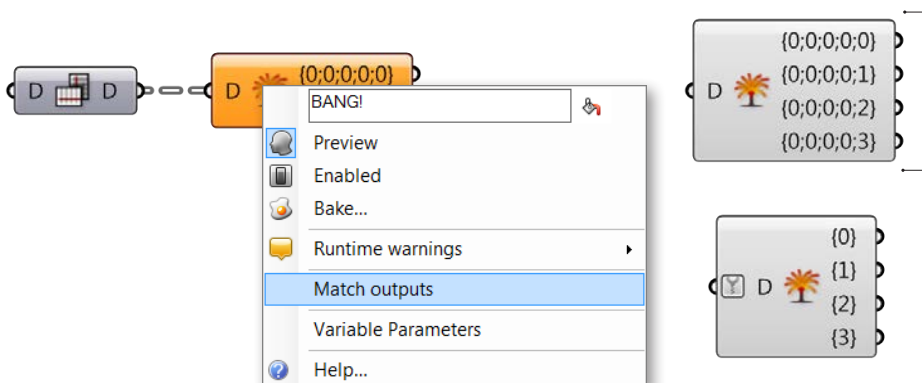


38. Surface/Analysis/Deconstruct Brep – Drag the deconstruct Brep component onto the canvas
39. Sets/Tree/Flip Matrix – Drag the Flip Matrix Component to the canvas.
40. Sets/Tree/Explode Tree – Drag the Explode Tree component to the canvas
41. Connect the Surface (S) output of the Isotrim component to the Brep (B) input of the Deconstruct Brep component
42. Connect the Vertices (V) output of the Deconstruct Brep component to the Data (D) input of the Flip Matrix component
43. Connect the Data (D) output of the Flip Matrix component to the Data (D) input of the Explode Tree component
44. Right-click the Explode Tree component and select “Match Outputs”
45. Right-click the Data (D) input of the Explode Tree component and select



The Deconstruct Brep component deconstructs a Brep into Faces, Edges, and Vertices. This is helpful if you want to operate on a specific constituent of the surface.

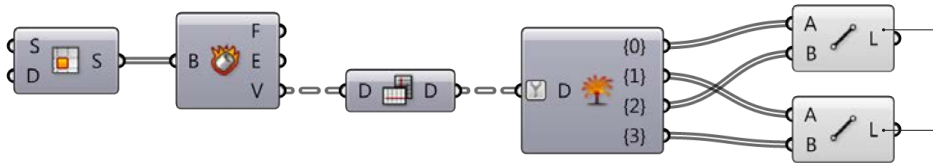
We just changed the Data tree structure from one list of four vertices that define each surface, to four lists, each containing one vertex of each surface.



Each output of the Explode Tree component contains a list of one vertex of each surface. In other words, one list with all the top right corners, one list with all the bottom right corners, one list of top left corners, and one list of bottom left corners.

simplify

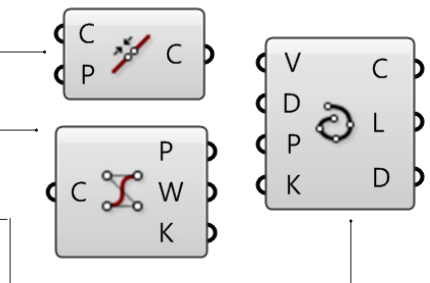
46. Curve/Primitive/Line – Drag and drop two line components onto the canvas
47. Connect the Branch 0 {0} output of the Explode Tree component to the Start Point (A) input of the first Line component
48. Connect the Branch 1 {1} output of the Explode Tree component to the Start Point (A) input of the Second Line component
49. Connect the Branch 2 {2} output of the Explode Tree component to the End Point (B) input of the first Line component
50. Connect the Branch 3 {3} output of the Explode Tree component to the



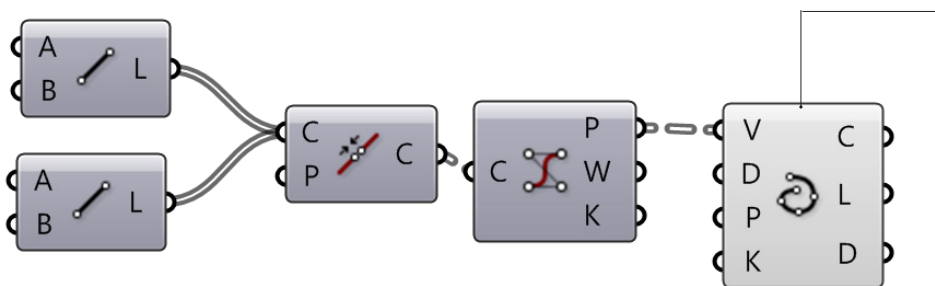
We have now connected the corner points of each surface diagonally with lines.

End Point (B) input of the Second Line component

51. Curve/Util/Join Curves – Drag and drop the Join Curves component to the canvas
52. Curve/Analysis/Control Points – Drag a Control Points component onto the canvas
53. Curve/Spline/Interpolate – Drag and drop the Interpolate component onto the canvas
54. Connect the Line (L) outputs of each Line component to the Curves (C) input of the Join Curves component
55. Connect the Curves (C) output of the Join Curves component to the Curve (C) input of the Control Points component
56. Connect the Points (P) output of the Control Points component to the



Hold down the Shift key to connect multiple wires to a single input



We have now joined our lines into polylines and reconstructed them as NURBS curves by interpolating their control points. In the Rhino viewport, you might notice that the shorter curves are still straight lines. This is because you cannot make a degree three NURBS curve with fewer than four control points. Let's manipulate the data tree to eliminate lists of control points with less than four items.

Vertices (V) input of the Interpolate component

57. Sets/Tree/Prune Tree – Drag and drop the Prune Tree component onto the canvas
58. Params/Input/Panel – Drag a Panel onto the canvas
59. Connect the Points (P) output of the Control Points component to the Tree (T) input of the Prune Tree component
60. Double click the Panel and enter 4.
61. Connect the output of the Panel to the Minimum (N0) input of the Prune Tree component
62. Connect the Tree (T) output of the Prune Tree component to the Vertices (V) input of the Interpolate component



If you connect one Param Viewer to the Points (P) output of the Control Points component, and another to the Tree (T) output of the Prune Tree component, you can see that the number of branches has been reduced.

63. Surface/Freeform/Extrude - Drag and drop the Extrude component onto the canvas

64. Vector/Vector/Unit Y - Drag a Unit Y component onto the canvas

65. Params/Input/Number Slider - Drag a number slider onto the canvas

66. Double click the Number Slider and set the following:

Rounding: Integer

Lower Limit: 1

Upper Limit: 5

Value: 3

67. Connect the Curve (C) output of the Interpolate component to the Base (B) input of the Extrude component

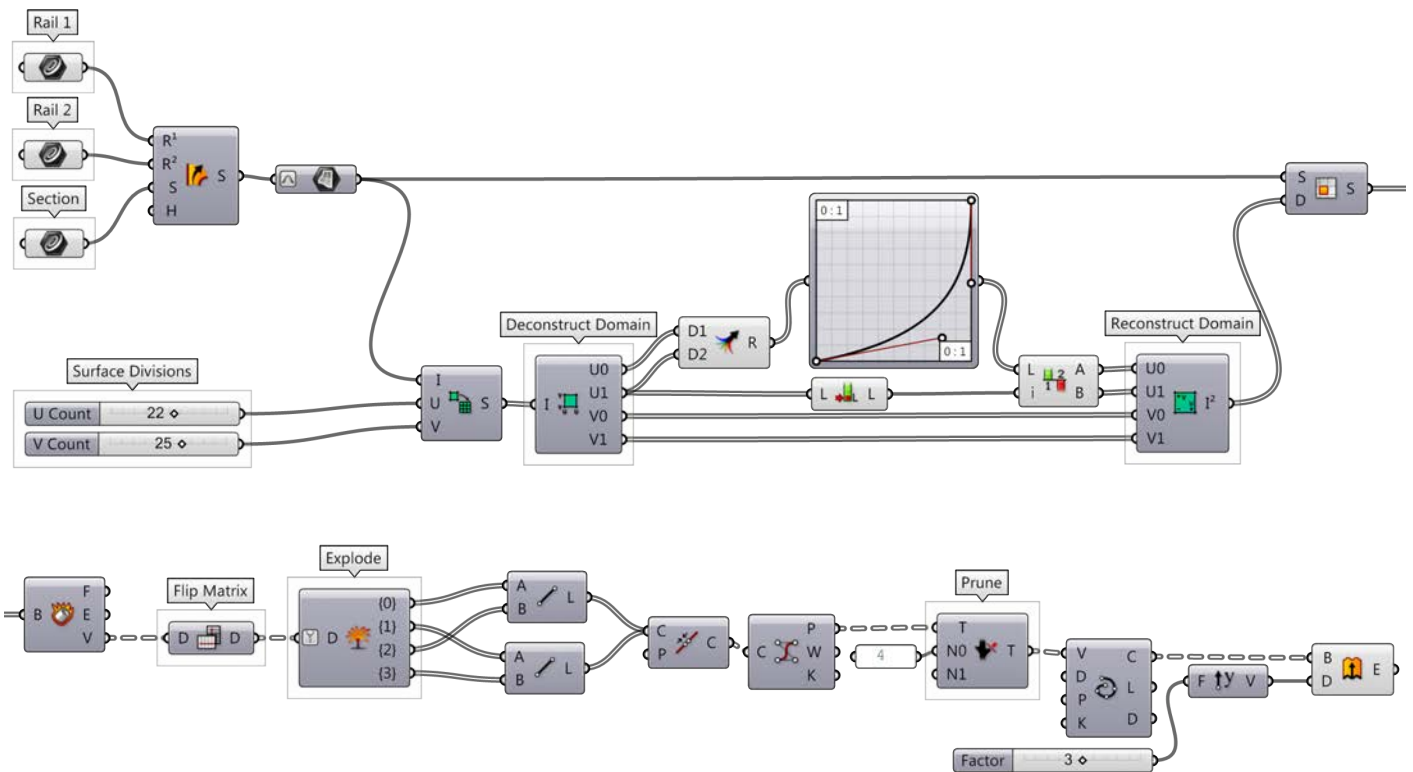
68. Connect the Number Slider output to the Factor (F) input of the Unit Y component

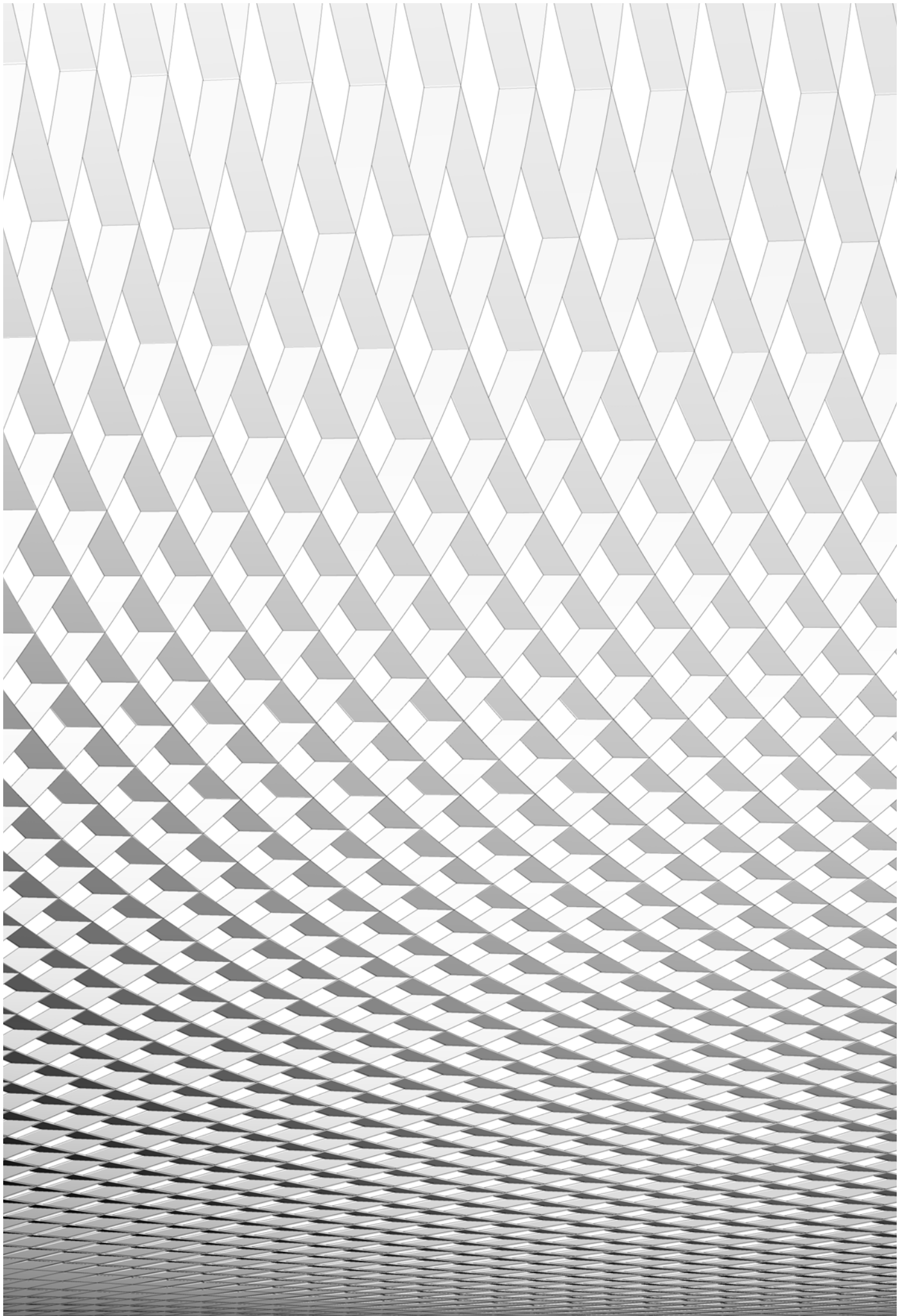
69. Connect the Unit Vector (V) output of the Unit Y component to the Direction (D) input of the Extrude component



You may need to use a Unit X vector, depending on the orientation of your referenced geometry in Rhino

You should now see a diagonal grid of strips or fins in the Rhino Viewport. Adjust the Factor slider to change the depth of the fins





[A.] APPENDIX




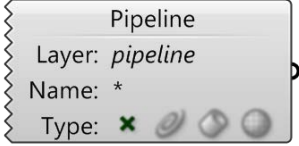


The following section contains useful references including an index of all the components used in this primer, as well as additional resources to learn more about Grasshopper.

A.0 Index




This index provides additional information on all the components used in this primer, as well as other components you might find useful. This is just an introduction to over 500 components in the Grasshopper plugin.





Parameters

GEOMETRY





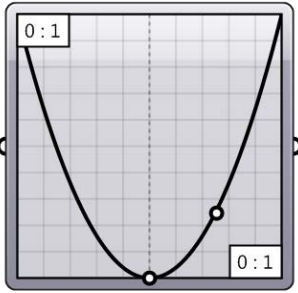
P.G.Crv	Curve Parameter Represents a collection of Curve geometry. Curve geometry is the common denominator of all curve types in Grasshopper.	
P.G.Circle	Circle parameter Represents a collection of Circle primitives.	
P.G.Geo	Geometry Parameter Represents a collection of 3D Geometry.	
P.G.Pipeline	Geometry Pipeline Defines a geometry pipeline from Rhino to Grasshopper.	
P.G.Pt	Point Parameter Point parameters are capable of storing persistent data. You can set the persistent records through the parameter menu.	
P.G.Srf	Surface Parameter Represents a collection of Surface geometry. Surface geometry is the common denominator of all surface types in Grasshopper.	


PRIMITIVE

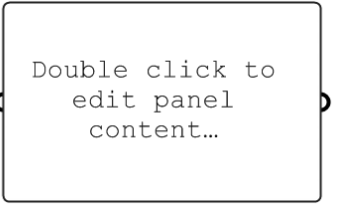
P.P.Bool	Boolean parameter Represents a collection of Boolean (True/False) values.	
P.P.D	Domain Parameter Represents a collection of one-dimensional Domains. Domains are typically used to represent curve fragments and continuous numeric ranges. A domain consists of two numbers that indicate the limits of the domain, everything in between these numbers is part of the domain.	
P.P.D ²	Domain² Parameter Contains a collection of two-dimensional domains. 2D Domains are typically used to represent surface fragments. A two-dimensional domain consists of two one-dimensional domains.	

P.P.ID	Guid Parameter Represents a collection of Globally Unique Identifiers. Guid parameters are capable of storing persistent data. You can set the persistent records through the parameter menu.	
P.P.Int	Integer Parameter Represents a collection of Integer numeric values. Integer parameters are capable of storing persistent data. You can set the persistent records through the parameter menu.	
P.P.Num	Number Parameter Represents a collection of floating point values. Number parameters are capable of storing persistent data. You can set the persistent records through the parameter menu.	
P.P.Path	File Path Contains a collection of file paths.	

INPUT


P.I.Toggle	Boolean Toggle Boolean (true/false) toggle.	
P.I.Button	Button Button object with two values. When pressed, the button object returns a true value and then resets to false.	
P.I.Swatch	Color Swatch A swatch is a special interface object that allows for quick setting of individual color values. You can change the color of a swatch through the context menu.	
P.I.Grad	Gradient Control Gradient controls allow you to define a color gradient within a numeric domain. By default the unit domain (0.0 ~ 1.0) is used, but this can be adjusted via the L0 and L1 input parameters. You can add color grips to the gradient object by dragging from the color wheel at the upper left and set color grips by right clicking them.	
P.I.Graph	Graph Mapper Graph mapper objects allow you to remap a set of numbers. By default the {x} and {y} domains of a graph function are unit domains (0.0 ~ 1.0), but these can be adjusted via the Graph Editor. Graph mappers can contain a single mapping function, which can be picked through the context menu. Graphs typically have grips (little circles), which can be used to modify the variables that define the graph equation. By default, a graph mapper objects contains no graph and performs a 1:1 mapping of values.	


P.I.Slider **Number Slider** 
 A slider is a special interface object that allows for quick setting of individual numeric values. You can change the values and properties through the menu, or by double-clicking a slider object. Sliders can be made longer or shorter by dragging the rightmost edge left or right. Note that sliders only have an output (ie. no input).


P.I.Pannel **Panel** 
 A panel for custom notes and text values. It is typically an inactive object that allows you to add remarks or explanations to a Document. Panels can also receive their information from elsewhere. If you plug an output parameter into a Panel, you can see the contents of that parameter in real-time. All data in Grasshopper can be viewed in this way. Panels can also stream their content to a text file.


P.I.List **Value List** 
 Provides a list of preset values from which to choose.

UTILITIES

P.U.Cin **Cluster Input** 
 Represents a cluster input parameter.

P.U.COut **Cluster Output** 
 Represents a cluster input parameter.

P.U.Dam **Data Dam** 
 Delay data on its way through the document.

P.U.Jump **Jump** 
 Jump between different locations.


P.U. Viewer **Param Viewer** 
 A viewer for data structures.


P.U.Scribble **Scribble** 
 A quick note.

DoubleClick Me!

Maths

DOMAIN

M.D.Bnd **Bounds** 
 Create a numeric domain which encompasses a list of numbers.

M.D. Consec **Consecutive Domains** 
 Create consecutive domains from a list of numbers.

M.D.Dom	Construct Domain Create a numeric domain from two numeric extremes.	
M.D. Dom ² Num	Construct Domain² Create a two-dimensional domain from four numbers.	
M.D. DeDomain	Deconstruct Domain Deconstruct a numeric domain into its component parts.	
M.D. DeDom ² Num	Deconstruct Domain² Deconstruct a two-dimensional domain into four numbers.	
M.D. Divide	Divide Domain² Divides a two-dimensional domain into equal segments.	
M.D.Inc	Includes Test a numeric value to see if it is included in the domain.	
M.D. ReMap	Remap Numbers Remap numbers into a new numeric domain.	

OPERATORS

M.O.Add	Addition Mathematical addition.	
M.O.Div	Division Mathematical division.	
M.O. Equals	Equality Test for (in)equality of two numbers.	
M.O.And	Gate And Perform boolean conjunction (AND gate). Both inputs need to be True for the result to be True.	
M.O.Not	Gate Not Perform boolean negation (NOT gate).	
M.O.Or	Gate Or Perform boolean disjunction (OR gate). Only a single input has to be True for the result to be True.	
M.O. Larger	Larger Than Larger than (or equal to).	
M.O. Multiply	Multiplication Mathematical multiplication.	

M.O. Smaller	Smaller Than Larger than (or equal to).	
M.O. Similar	Similarity Test for similarity of two numbers.	
M.O.Sub	Subtraction Mathematical subtraction.	

SCRIPT

M.S.Eval	Evaluate Evaluate an expression with a flexible number of variables.	
M.S. Expression	Expression Evaluate an expression.	

TRIG

M.T.Cos	Cosine Compute the cosine of a value.	
M.T.Deg	Degrees Convert an angle specified in radians to degrees.	
M.T.Rad	Radians Convert an angle specified in degrees to radians.	
M.T.Sim	Sine Compute the sine of a value.	

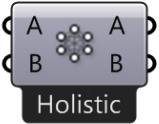

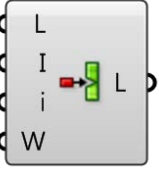


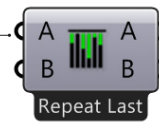
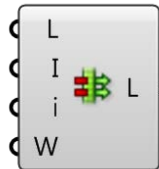

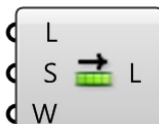

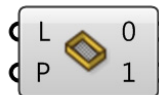
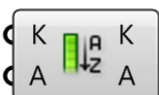
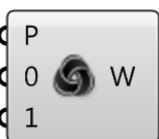
UTILITIES

M.U.Avr	Average Solve the arithmetic average for a set of items.	
M.U.Phi	Golden Ratio Returns a factor of the golden ratio (Phi).	
M.U.Pi	Pi Returns a factor of Pi.	






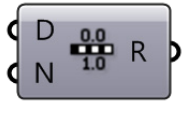


Sets

LIST



S.L. Combine	Combine Data Combine non-null items out of several inputs.	
-----------------	---	--









S.L. CrossRef	Cross Reference Cross Reference data from multiple lists.	
S.L. Dispatch	Dispatch Dispatch the items in a list into two target lists. List dispatching is very similar to the [Cull Pattern] component, with the exception that both lists are provided as outputs.	
S.L.Ins	Insert Items Insert a collection of items into a list.	
S.L.Item	List Item Retrieve a specific item from a list.	
S.L.Lng	List Length Measure the length of a list. Elements in a list are identified by their index. The first element is stored at index zero, the second element is stored at index one and so on and so forth. The highest possible index in a list equals the length of the list minus one.	
S.L.Long	Longest List Grow a collection of lists to the longest length amongst them.	
S.L. Replace	Replace Items Replace certain items in a list.	
S.L.Rev	Reverse List Reverse the order of a list. The new index of each element will be N-i where N is the highest index in the list and i is the old index of the element.	
S.L.Shift	Shift List Offset all items in a list. Items in the list are offset (moved) towards the end of the list if the shift offset is positive. If Wrap equals True, then items that fall off the ends are re-appended.	
S.L.Short	Shortest List Shrink a collection of lists to the shortest length amongst them.	
S.L.Sift	Sift Pattern Sift elements in a list using a repeating index pattern.	
S.L.Sort	Sort List Sort a list of numeric keys. In order for something to be sorted, it must first be comparable. Most types of data are not comparable, Numbers and Strings being basically the sole exceptions. If you want to sort other types of data, such as curves, you'll need to create a list of keys first.	
S.L.Weave	Weave Weave a set of input data using a custom pattern. The pattern is specified as a list of index values (integers) that define the order in which input data is collected.	

SETS

S.S.Culli	<p>Cull Index —————</p> <p>Cull (remove) indexed elements from a list.</p>	
S.S.Cull	<p>Cull Pattern —————</p> <p>Cull (remove) elements in a list using a repeating bit mask. The bit mask is defined as a list of Boolean values. The bit mask is repeated until all elements in the data list have been evaluated.</p>	
S.S.Dup	<p>Duplicate Data —————</p> <p>Duplicate data a predefined number of times. Data can be duplicated in two ways, either copies of the list are appended at the end until the number of copies has been reached, or each item is duplicated a number of times before moving on to the next item.</p>	
S.S.Jitter	<p>Jitter —————</p> <p>Randomly shuffles a list of values. The input list is reordered based on random noise. Jittering is a good way to get a random set with a good distribution. The jitter parameter sets radius of the random noise. If jitter equals 0.5, then each item is allowed to reposition itself randomly to within half the span of the entire set.</p>	
S.S. Random	<p>Random —————</p> <p>Generate a list of pseudo random numbers. The number sequence is unique but stable for each seed value. If you do not like a random distribution, try different seed values.</p>	
S.S.Range	<p>Range —————</p> <p>Create a range of numbers. The numbers are spaced equally inside a numeric domain. Use this component if you need to create numbers between extremes. If you need control over the interval between successive numbers, you should be using the [Series] component.</p>	
S.S.Repeat	<p>Repeat Data —————</p> <p>Repeat a pattern until it reaches a certain length.</p>	
S.S.Series	<p>Series —————</p> <p>Create a series of numbers. The numbers are spaced according to the {Step} value. If you need to distribute numbers inside a fixed numeric range, consider using the [Range] component instead.</p>	

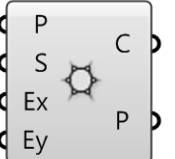
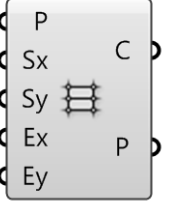
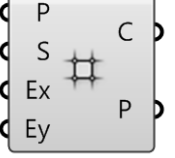
TREE

S.T.Explode	<p>Explode Tree —————</p> <p>Extract all the branches from a tree.</p>	
S.T.Flatten	<p>Flatten Tree —————</p> <p>Flatten a data tree by removing all branching information.</p>	

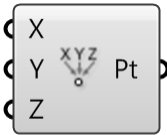
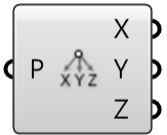

S.T.Flip	Flip Matrix Flip a matrix-like data tree by swapping rows and columns.	
S.T.Graft	Graft Tree Typically, data items are stored in branches at specific index values (0 for the first item, 1 for the second item, and so on and so forth) and branches are stored in trees at specific branch paths, for example: {0;1}, which indicates the second sub-branch of the first main branch. Grafting creates a new branch for every single data item.	
S.T.Merge	Merge Merge a bunch of data streams.	
S.T.Path	Path Mapper Perform lexical operations on data trees. Lexical operations are logical mappings between data paths and indices which are defined by textual (lexical) masks and patterns.	
S.T.Prune	Prune Tree Removes all branches from a Tree that carry a special number of Data items. You can supply both a lower and an upper limit for branch pruning.	
S.T.Simplify	Simplify Tree Simplify a tree by removing the overlap shared amongst all branches.	
S.T.TStat	Tree Statistics Get some statistics regarding a data tree.	
S.T. Unflatten	Unflatten Tree Unflatten a data tree by moving items back into branches.	

Vector


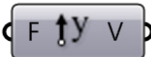
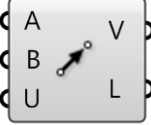
GRID

V.G. HexGrid	Hexagonal 2D grid with hexagonal cells.	
V.G. RecGrid	Rectangular 2D grid with rectangular cells.	
V.G.SqGrid	Square 2D grid with square cells.	

POINT


V.P.Pt	Construct Point Construct a point from {xyz} coordinates.	
V.P.pDecon	Deconstruct Deconstruct a point into its component parts.	
V.P.Distance	Distance Compute Euclidean distance between two point coordinates.	

VECTOR


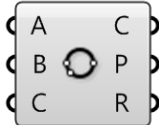


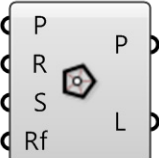
V.V.X	Unit X Unit vector parallel to the world {x} axis.	
V.V.Y	Unit Y Unit vector parallel to the world {y} axis.	
V.V.Vec2Pt	Vector 2Pt Create a vector between two points.	

Curve

DIVISION

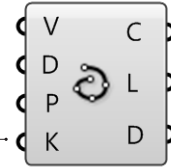
C.D.Divide	Divide Curve Divide a curve into equal length segments.	
------------	---	---

PRIMITIVE

C.P.Cir	Circle Create a circle defined by base plane and radius.	
C.P.Cir3Pt	Circle 3Pt Create a circle defined by three points.	
C.P.CirCNR	Circle CNR Create a circle defined by center, normal and radius.	
C.P.Line	Line SDL Create a line segment defined by start point, tangent and length.	
C.P.Polygon	Polygon Create a polygon with optional round edges.	

SPLINE

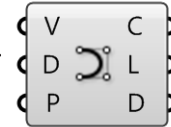
C.S.IntCrv Interpolate
Create an interpolated curve through a set of points.



C.S.KinkCrv Kinky Curve
Construct an interpolated curve through a set of points with a kink angle threshold.



C.S.Nurbs Nurbs Curve
Construct a nurbs curve from control points.



C.S.PLine PolyLine
Create a polyline connecting a number of points.

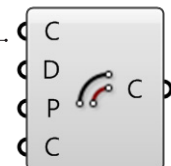


UTIL

C.U.Explode Explode
Explode a curve into smaller segments.



C.U.Offset Offset
Offset a curve with a specified distance.



Surface

ANALYSIS

S.A.DeBrep Deconstruct Brep
Deconstruct a brep into its constituent parts.



FREEFORM

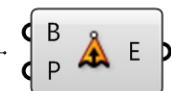
S.F.Boundary Boundary
Create planar surfaces from a collection of boundary edge curves.



S.F.Extr Extrude
Extrude curves and surfaces along a vector.



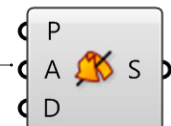
S.F.ExtrPt Extrude Point
Extrude curves and surfaces to a point.



S.F.Loft Loft
Create a lofted surface through a set of section curves.



S.F.RevSrf Revolution
Create a surface of revolution.



S.F.Swp2 Sweep2
Create a sweep surface with two rail curves.



PRIMITIVE

S.P.BBox **Bounding Box**
Solve oriented geometry bounding boxes.



UTIL

S.U.
SDivide **Divide Surface**
Generate a grid of {uv} points on a surface.



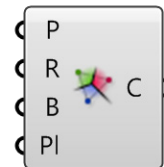
S.U.SubSrf **Isotrim**
Extract an isoparametric subset of a surface.



Mesh

TRIANGULATION

M.T.
Voronoi **Voronoi**
Planar voronoi diagram for a collection of points.



Transform

AFFINE

T.A.
RecMap **Rectangle Mapping**
Transform geometry from one rectangle into another.



ARRAY

T.A.
ArrLinear **Linear Array**
Create a linear array of geometry.



MORPH

T.M.Morph **Box Morph**
Morph an object into a twisted box.



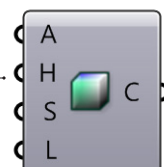
T.M.SBox **Surface Box**
Create a twisted box on a surface patch.



Display

COLOR

D.C.HSL **Colour HSL**
Create a colour from floating point {HSL} channels.



DIMENSIONS

D.D.Tag **Text tags**
A text tag component allows you to draw little Strings in the viewport as feedback items. Text and location are specified as input parameters. When text tags are baked they turn into Text Dots.



D.D.Tag3D **Text Tag 3D**
Represents a list of 3D text tags in a Rhino viewport.



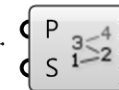
PREVIEW

D.P.
Preview **Custom Preview**
Allows for customized geometry previews.



VECTOR

D.V.Points **Point List**
Displays details about lists of points.



A.1 Resources

There are many resources available to learn more about Grasshopper and parametric design concepts. There are also over a hundred plugins and add-ons that extend Grasshopper's functionality. Below are some of our favorites.

PLUG-IN COMMUNITIES



food4Rhino (WIP) is the new Plug-in Community Service by McNeel. As a user, find the newest Rhino Plug-ins, Grasshopper Add-ons, Textures and Backgrounds, add your comments, discuss about new tools, get in contact with the developers of these applications, share your scripts.
<http://www.food4rhino.com/>



Grasshopper add-ons page
<http://www.grasshopper3d.com/page/addons-for-grasshopper>

ADD-ONS WE LOVE



DIVA-for-Rhino allows users to carry out a series of environmental performance evaluations of individual buildings and urban landscapes.
<http://diva4rhino.com/>



Fold panels using curved folding and control panel distribution on surfaces with a range of attractor systems.
<http://www.food4rhino.com/project/robofoldingkong>



Firefly offers a set of comprehensive software tools dedicated to bridging the gap between Grasshopper and the Arduino micro-controller.
<http://fireflyexperiments.com>



LunchBox is a plug-in for Grasshopper for exploring mathematical shapes, paneling, structures, and workflow.
<http://www.food4rhino.com/project/lunchbox>



GhPython is the Python interpreter component for Grasshopper that allows you to execute dynamic scripts of any type. Unlike other scripting components, GhPython allows the use of rhinoscriptsyntax to start scripting without needing to be a programmer.
<http://www.food4rhino.com/project/ghpython>



Meshedit is a set of components which extend Grasshopper's ability to work with meshes.
<http://www.food4rhino.com/project/meshedittools>



HAL is a Grasshopper plugin for industrial robots programming supporting ABB, KUKA and Universal Robots machines.
<http://hal.thibaultschwartz.com/>



Parametric tools to create and manipulate rectangular grids, attractors and support creative morphing of parametric patterns.
<http://www.food4rhino.com/project/pt-gh>



Extends Grasshopper's ability to create and reference geometry including lights, blocks, and text objects. Also enables access to information about the active Rhino document, pertaining to materials, layers, linetypes, and other settings.
<http://www.food4rhino.com/project/human>



Platypus allows Grasshopper authors to stream geometry to the web in real time. It works like a chatroom for parametric geometry, and allows for on-the-fly 3D model mashups in the web browser.
<http://www.food4rhino.com/project/platypus>



Karamba is an interactive, parametric finite element program. It lets you analyze the response of 3-dimensional beam and shell structures under arbitrary loads.
<http://www.karamba3d.com/>



TT Toolbox features a range of different tools that we from the Core Studio at Thornton Tomasetti use on a regular basis, and we thought some of you might appreciate these.
<http://www.food4rhino.com/project/tttoolbox>



Kangaroo is a Live Physics engine for interactive simulation, optimization and form-finding directly within Grasshopper.
<http://www.food4rhino.com/project/kangaroo>



Weaverbird is a topological modeler that contains many of the known subdivision and transformation operators, readily usable by designers. This plug-in reconstructs the shape, subdivides any mesh, even made by polylines, and helps preparing for fabrication.
<http://www.giulioiacentino.com/weaverbird/>

ADDITIONAL PRIMERS

The Firefly Primer

This book is intended to teach the basics of electronics (using an Arduino) as well as various digital/physical prototyping techniques to people new to the field. It is not a comprehensive book on electronics (as there are already a number of great resources already dedicated to this topic). Instead, this book focuses on expediting the prototyping process. Written by Andrew Payne.

<http://fireflyexperiments.com/resources/>

Essential Mathematics

Essential Mathematics uses Grasshopper to introduce design professionals to foundation mathematical concepts that are necessary for effective development of computational methods for 3D modeling and computer graphics. Written by Rajaa Issa.

<http://www.rhino3d.com/download/rhino/5.0/EssentialMathematicsThirdEdition/>

Generative Algorithms

A series of books which is aimed to develop different concepts in the field of Generative Algorithms and Parametric Design. Written by Zubin Khabazi.

<http://www.morphogenesism.com/media.html>

Rhino Python Primer

This primer is intended to teach programming to absolute beginners, people who have tinkered with programming a bit or expert programmers looking for a quick introduction to the methods in Rhino. Written by Skylar Tibbits.

<http://www.rhino3d.com/download/IronPython/5.0/RhinoPython101>

FURTHER READING

Burry, Jane, and Mark Burry. *The New Mathematics of Architecture*. London: Thames & Hudson, 2010.

Burry, Mark. *Scripting Cultures: Architectural Design and Programming*. Chichester, UK: Wiley, 2011.

Hensel, Michael, Achim Menges, and Michael Weinstock. *Emergent Technologies and Design: Towards a Biological Paradigm for Architecture*. Oxon: Routledge, 2010.

Jabi, Wassim. *Parametric Design for Architecture*. Laurence King, 2013.

Menges, Achim, and Sean Ahlquist. *Computational Design Thinking*. Chichester, UK: John Wiley & Sons, 2011.

Menges, Achim. *Material Computation: Higher Integration in Morphogenetic Design*. Hoboken, NJ: Wiley, 2012.

Peters, Brady, and Xavier De Kestelier. *Computation Works: The Building of Algorithmic Thought*. Wiley, 2013.

Peters, Brady. *Inside Smartgeometry: Expanding the Architectural Possibilities of Computational Design*. Chichester: Wiley, 2013.

Pottmann, Helmut, and Daril Bentley. *Architectural Geometry*. Exton, PA: Bentley Institute, 2007.

Sakamoto, Tomoko, and Albert Ferré. *From Control to Design: Parametric/algorithmic Architecture*. Barcelona: Actar-D, 2008.

Woodbury, Robert. *Elements of Parametric Design*. London: Routledge, 2010.

A.2 Notes



A.2 Notes



A.3

About this Primer

AUTHORS



GIL AKOS, MODE LAB

Gil Akos is a founding partner and Director of Technology at Mode Lab, a multidisciplinary design consultancy specializing in technology-driven process innovation. He brings diverse professional experience, technical expertise in digital platforms, and a passion for generative design to the service model of the studio. His personal interests surround the relationship between simulation and materialization and ways by which this connection can be made tangible.

<http://modelab.is>

<http://modelab.is/education>



RONNIE PARSONS, MODE LAB

Ronnie Parsons is a founding partner and Director of Education at Mode Lab, a multidisciplinary design consultancy specializing in technology-driven process innovation. At Mode Lab, Ronnie identifies new ways to connect and configure client workflows by strategically aligning product vision with UX-centered technology platforms. Ronnie's expertise resides in the areas of advanced computational modeling, instructional design, and research and development.

<http://modelab.is>

<http://modelab.is/education>



MODE LAB TEAM:

Sharon Jamison

Andrew Reitz

Armon Jahanshahi

Luis Quinones

Erick Katzenstein

Kimberly Parsons

Roberto Godinez

CONTRIBUTORS



ANDREW PAYNE, PRINCIPAL, LIFT ARCHITECTS

Andrew Payne is a registered architect who founded LIFT architects in 2007. Andrew's work explores embedded computation, intelligent buildings, and generative design and he has published papers and taught workshops throughout North America and Europe. In 2010, Andrew and Jason K. Johnson published Firefly - a comprehensive software plug-in dedicated to bridging the gap between Grasshopper, the Arduino microcontroller, the internet, audio/visual tools, and more.

<http://www.liftarchitects.com/>

This primer provides a comprehensive guide to the most current Grasshopper build, version 0.90076, highlighting what we feel are some of the most exciting feature updates. It is our goal that this primer will serve as a field guide to new and existing users looking to navigate the ins and outs of using Grasshopper in their creative practice.



Mode Lab is a multidisciplinary design consultancy specializing in technology-driven process innovation.

From its inception, Mode Lab has been a space for experimenting with the methods and technology used to design and make the world around us. We are compelled to understand and improve upon the process of materializing ideas – this is a journey we undertake in collaboration with our clients.
<http://modelab.is>



Our Mode Lab Education brand provides blended learning solutions for consumers and businesses looking to get ahead. We design and develop targeted learning experiences that keep the learner at the center of every technique and design method we share with our community.
<http://modelab.is/education>



McNeel is a software development company with worldwide sales, support, and training. Founded in 1980, McNeel is a privately-held, employee-owned company with sales and support offices and affiliates in Seattle, Boston, Miami, Buenos Aires, Barcelona, Rome, Tokyo, Taipei, Seoul, Kuala Lumpur, and Shanghai with more than 700 resellers, distributors, OEMs, and training centers around the world.
<http://www.en.na.mcneel.com/>



For designers who are exploring new shapes using generative algorithms, Grasshopper is a graphical algorithm editor tightly integrated with Rhino's 3D modeling tools. Unlike RhinoScript, Grasshopper requires no knowledge of programming or scripting, but still allows designers to build form generators from the simple to the awe-inspiring.
<http://www.grasshopper3d.com/>

LICENSING INFORMATION

The Grasshopper Primer is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported license. The full text of this license is available here: <http://creativecommons.org/licenses/by-nc-sa/3.0/us/legalcode>

Under this license, you are free:

TO SHARE - to copy, distribute and transmit the work

TO REMIX - to adapt the work

Under the following conditions:

ATTRIBUTION - You must attribute the work in the manner specified as "Mode Lab's Attribution" below. You cannot attribute the work in any manner that suggests that Mode Lab endorses you or your use of the work.

NONCOMMERCIAL - You may not use this work for commercial purposes

SHARE ALIKE - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported license.

Please see the full text of this license (<http://creativecommons.org/licenses/by-nc-sa/3.0/us/legalcode>) to view all rights and restrictions associated with it.

MODE LAB'S ATTRIBUTION:

©2014 Studio Mode, LLC. All rights reserved.
<http://modelab.is>

TRANSLATIONS:

If you create translated versions of this Primer (in compliance with this license), please notify Mode Lab at hello@modelab.is. Mode Lab may choose to distribute and/or link to such translated versions (either as-is, or as further modified by Mode Lab)



MODELAB