Sketching Data 101 010001010

Carson Smuts - GSAPP 2013

This document outlines the core principles behind Parametric and Algorithmic computation. What has become evident is that users tend to learn as much as they need to of the basics, in order to use the program to some degree of success, and then stop. This is completely understandable as time is always a factor in any design field. They never reach an advanced understanding of the software though.

This leads us to quite a dilemma. Advanced experimentation and exploration can be achieved quite easily as long as you know the foundations of any given piece of software. Again , this understanding is hardly ever achieved do to the required time. Think of Malcolm Gladwell's 10,000 hour principle. In reality nobody can afford to devote that much time to learning something that may or may not pay off in the end.

There is good news though, much of this can be circumnavigated so that everyone walks away with some advanced understanding of the inner workings of these programs.

Take any manual you have ever read. It starts from the outside and works its way in. If you buy a computer the first thing the manual will tell you is how to setup the monitor , keyboard, plug them in and turn it on. It does not tell you about the microprocessor, how many transistors it has, and how they work. The same goes for software manuals. They start by introducing the windows presented to you and the menus at your disposal. It will usually go through each and ever pull down menu, explaining every possible choice. Its a long and tedious process, and truth be told, nobody ever reads those things.

Again, you gloss over the basics and the play until you find what you need to know and nothing more.

The next few pages will turn this line of thinking on its head. We will explore Grasshopper (Explicit History) and Rhino starting with its core, computation and programming. Even though this might sound scary to some and boring to others, its quite the opposite as it leads to far greater and exciting design potential.

Open-Style

With the age of Open-Source software and Open-Source frameworks has come new freedom. Free knowledge and data to those of us who want it, allowing us to create our own software. We are no longer bound by the limits of main stream software packages and their generic workflows and styles. What was once accessible only by programmers is now available to us all. The potential is their for designers to have their own style of software, much like we all have our own style of handwriting or signature.

Still, this option remains out of reach for most people and the aim of this course outline is to break the stigma of programming and computation and to help accelerate ones thinking beyond the face value representation of the Graphical User Interfaces presented to us on a daily basis.
Before you get disheartened or dismissive about programming. We will make a strong case for it in the field of design. For every line you draw on the computer there is an algorithm at work. Furiously calculating the the pixels needed to represent the line drawn. What you see is merely a representation of a series of points in the cartesian plane depicting a line or a curve.

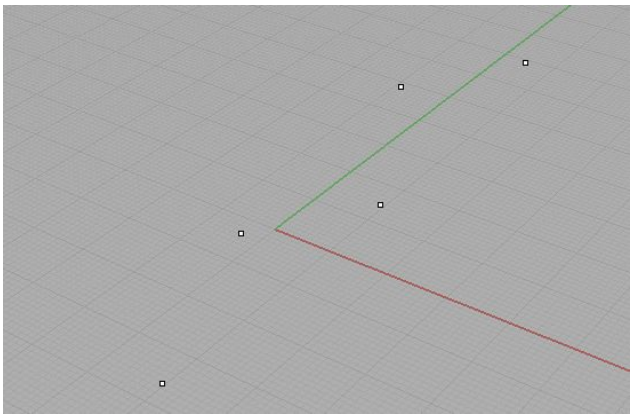
Three dimensional modeling works in the same way. The difference is that now the points create lines and in turn the lines or curves create nurbs surface or mesh.

One starts to understand that if you have total control over the points that create the initial curves, you can ultimately control the resulting surface or space. However, we are talking about thousands of points. This could be overwhelming if you had to try and control each point. We therefor employ computation in order to control these points, rules in tiny programs that control the behavior and responses of each point.

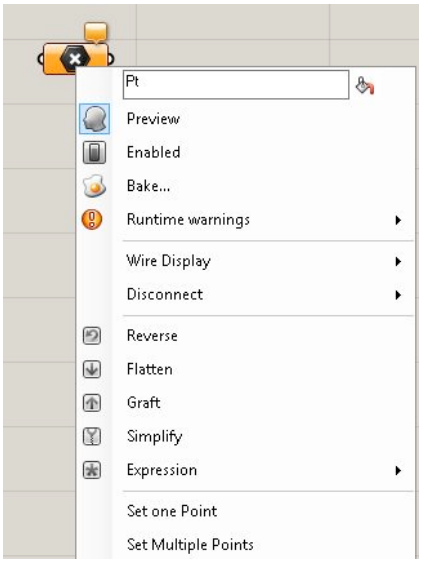
XYZ to Line

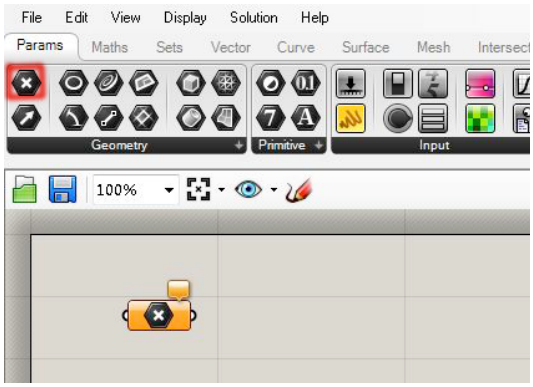
Grasshopper has the ability to reference objects in the Rhino viewport, creating a copy as it were for grasshopper to use. These objects can be anything, from points and curves to surfaces.

- *Lets start with the most basic reference type, a "Point". Draw 5 to 7 points in the Rhino viewport.*

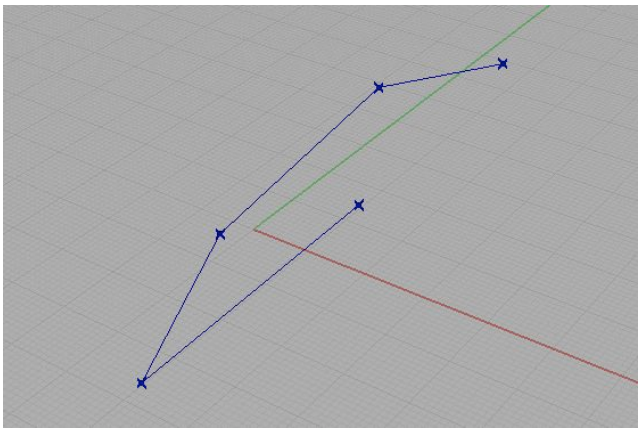


- *Reference these point in Grasshopper by creating a "Point" component in Grasshopper. We can now "set multiple points" (which means reference the points in Rhino) by "Right Clicking" the component we created and choosing "Set Multiple Points" from the drop down menu.*





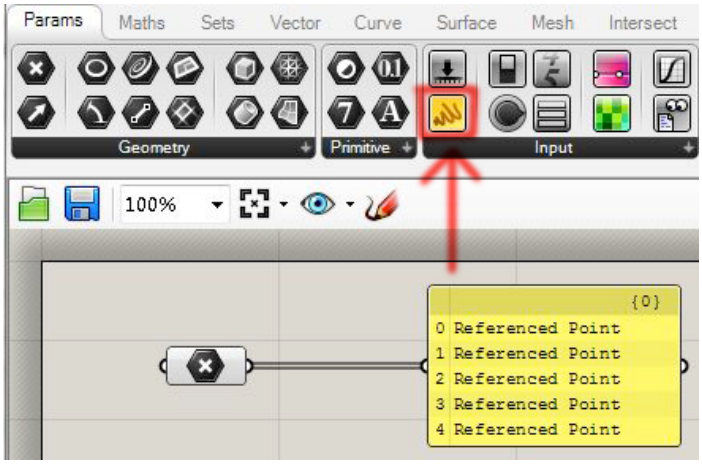
- *Once this is selected, we choose all the points we want to Reference in the Rhino viewport and these points are now referenced inside the "Point" component. Drag the selection window with the cursor over all the points.*



The points should show up as red or green in the Viewport. For clarity sake we will refer to the Rhino window as "Viewport" and the Grasshopper window as the "Canvas".

A component , such as the "Point" component we are using, is a place holder or container for list or arrays of objects or data. We will normally refer to these lists as "arrays". In our case, the component holds an array of points. It helps to think of components as lists.

- *In order to visually see what data is in a given list we can use a "Panel Component". This component opens up a visual list on the canvas, showing us everything held inside any component plugged into it. Connect the "Point" component to the "Panel" and a list of points with their respective XYZ coordinates will appear.*

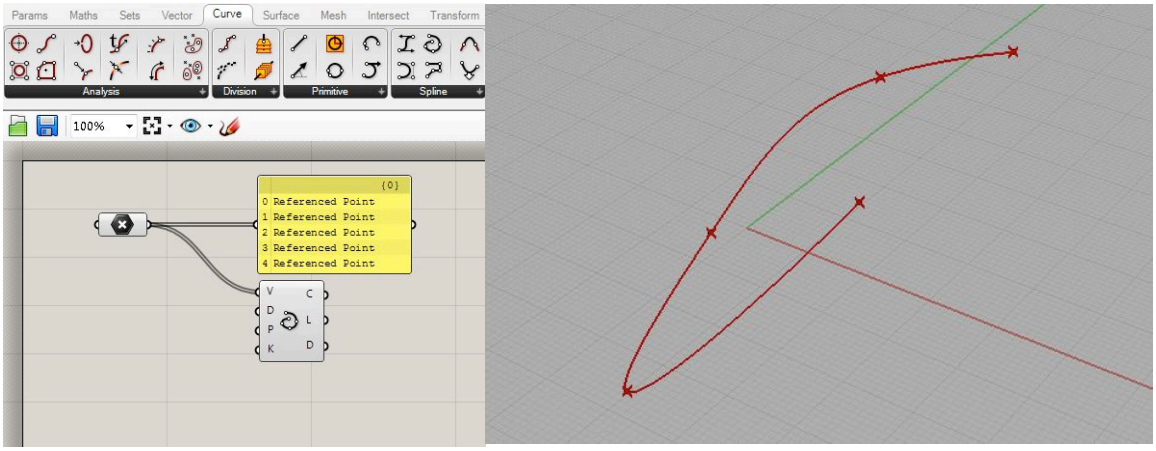


We can now use these points to create other geometry. Just as we were able to extract the points and show them in the "Panel" component, we will extract the points and now use them to create a curve. In

order to do so we need to employ the use of the “Interpolated Curve” component. This type of component requires point data (at least three points) in order to create a curve.

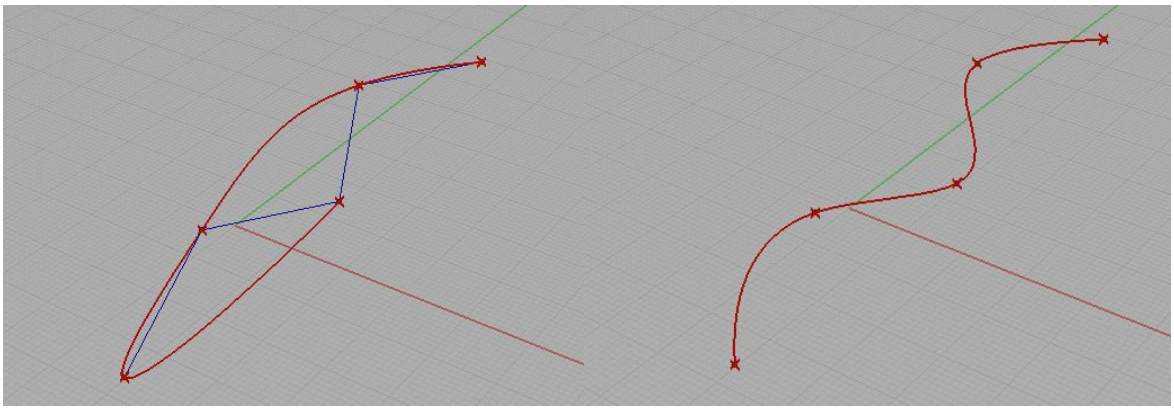
- *Shortcuts: In order to find this “Interpolate” component we can use a quick shortcut. Instead of having to go through all the menus in order to find the component you want, you can simply double click the canvas and type in the name of any given component ie: “Interpolate”. A list of more than one component may appear. Choose the one you want, and it should appear on the canvas.*

- *Connect the output from the “Point” component into the “Interpolated Curve” and a curve will appear in the Viewport.*



The first question one might have is “How does it determine the order in which to utilize the points?” The answer has everything to do with list order. When we first referenced the points we did not specify any order. What makes parametrics wonderful is that we can go back and specify the order without having to redo all our hard work. We do this by re-referencing the points in the “Point” component.

- *Right-click on the “Point” component and choose “set multiple points”. Now choose all the points once again, except this time select each point individually in the order in which you want the curve to be drawn.*



Due to the nature of Parametrics, you can also choose to manipulate the points manually in the Viewpoint and your definition will update itself accordingly.

Data to XYZ

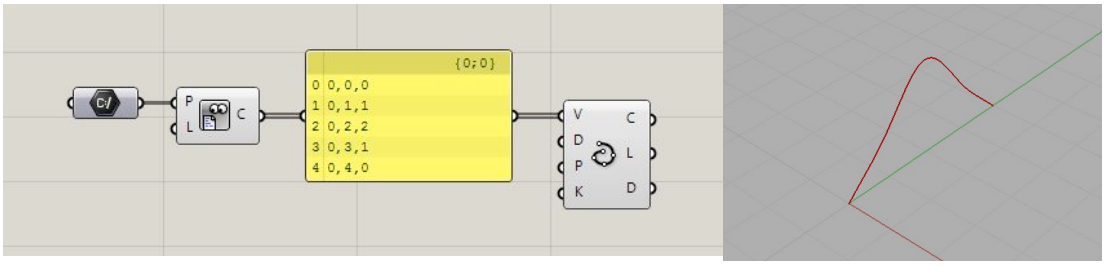
We now have a fairly good idea of how components, data and lists work. What makes programming arrays so powerful is that data can come from anywhere not just Rhinos internal framework. To demonstrate, we are going to create our own data from scratch. One can create a data file by using any text editing program. We will be using Windows Notepad as our text editor.

- *In your text editor create a new file. We are going to create a point data file from scratch.*
- *We are going to create our own points with the following format:* ***X,Y,Z***
- *Each line represents a new point and the XYZ coordinates of each point are separated by commas.*
- *Here is an example of what five points would look like, only using the Y and Z coordinates:*

  0,0,0
  0,1,1
  0,2,2
  0,3,1
  0,4,0

- *Save the file as a standard .TXT file.*
- *In order to bring this data into Grasshopper create a "File Path" component to reference the external TXT file we created.*
- *Right click on the "File Path" component and choose "set File".*

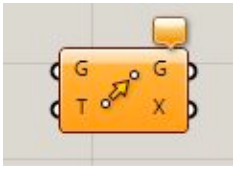
- *Create a "Read File" component and connect the "File Path" component.*
- *Plug in a "Panel" component to view the data from our file.*
- *Next, plug in the output from the "Panel" or the "Read File" component into an "Interpolate" component. The curve should now be visible in the Viewport.*



Space

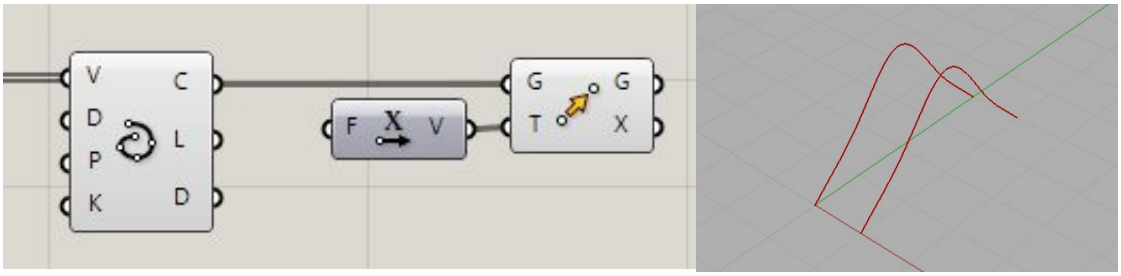
The curve we have created can now be used to create a surface, or perhaps space enclosed by a surface. First we need more than one curve in order to use lofting in this example. To speed things up we will simply copy the curve three times. There are several “Transform” components which can be used to move,rotate or scale an object. It should be evident by now, every time that you pass an object from one component to another, it effectively creates a copy of the data. There is no “copy” component, however the “transform” components do the job.

- *The easiest way to create a copy is to use the “Move” component in the canvas.*



This component requires a translation vector. A vector determines in which direction and how far to move a given object. More than one vector can be plugged in to create multiple copies of an object with different spatial translations.

- *Create an “X Unit” component and plug the output into the “Move” components “T” input. This will move the curve in the X-direction by a factor of 1.*



- *We can change the default value of 1 to anything we want by right-clicking on the “F” input of the “X Unit” component and changing the value manually.*

This is all good and well, however we only get a single copy from this method. As said before, in order to create multiple copies, multiple vectors need to be provided to the “Move” component, or rather an array/ list of multiple values. To create this list of numbers we make use of the “Series” component.

- *Create a “Series” component on the canvas.*
- *Plug the output into a new “Panel” component. An array from 0 to 9 should appear in the panel.*
- *This can now be plugged straight into the “Move” components “T” input. Ten curves should appear in the Viewport.*

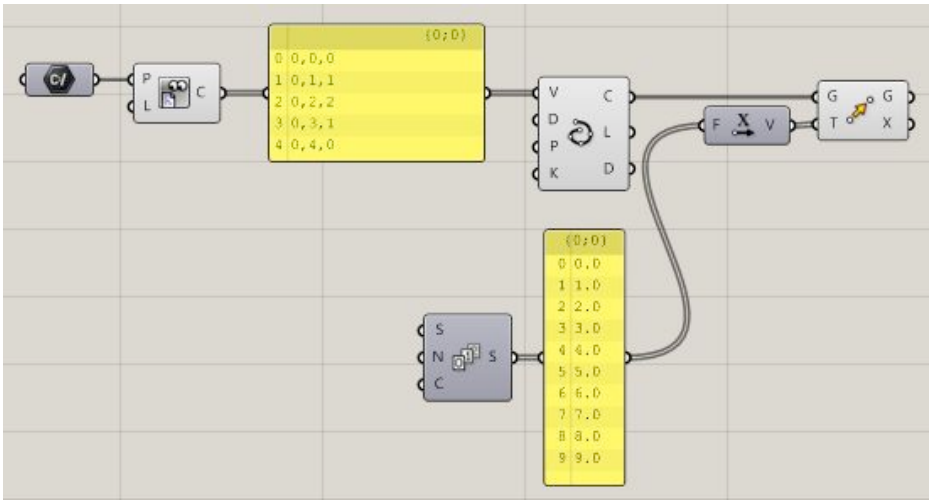
- *This component has three values that you can specify in order to change the output.*

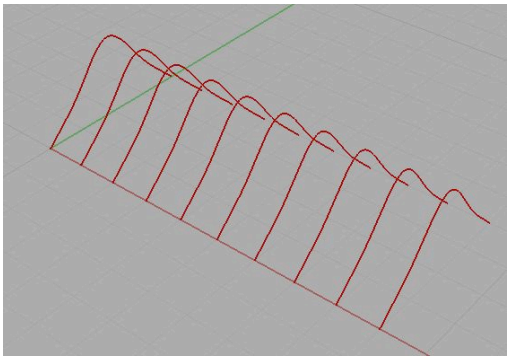
  S: Start value

  N: Interval size

  C: Count or number of values to create

- *You can change the inputs manually by right clicking on them or by plugging in "Slider" components.*

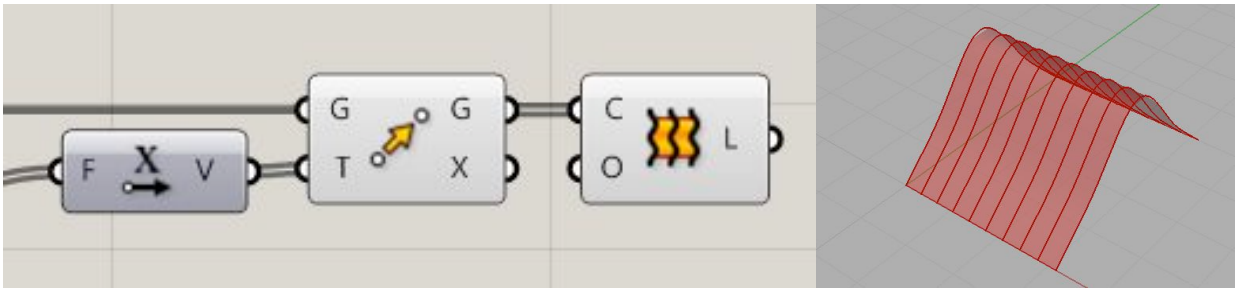




Now there should be an array of ten curves that we can use. Once again , it is always a good idea to check and see what the component contains by connecting its output to a panel. It should display a lit of ten curves. These curves can now be used to create other geometries, such as a surface or a mesh. Grasshopper has an entire toolset for creating surface geometries. These can be accessed via the surface toolbar.

- *Create a "Loft" component.*

- *This component requires a list of curves. The list of curves from our "Move" component needs to be plugged in and a new surface is created in the viewport.*

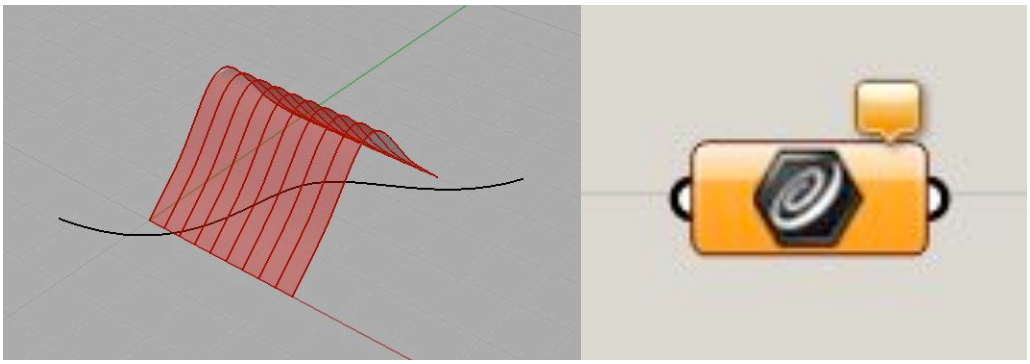


Movement

The true potential of this sudo-programming environment is the ability to access and change either the data source or the result of our definition. If the referenced TXT file is edited, the entire definition is updated automatically. This allows for a fourth dimension. If the data is changed over a span of time, then the spatial model will update accordingly over that duration. In order to do this, the data from the TXT file can be changed every second. There are ways to stream data to a TXT file from a source on the internet, however, this will be introduced at a later point.

For now, a slider will act as a source of streaming data. This data will drive the position of an attractor point and manipulate the control points of the curves, in turn manipulating the surface.

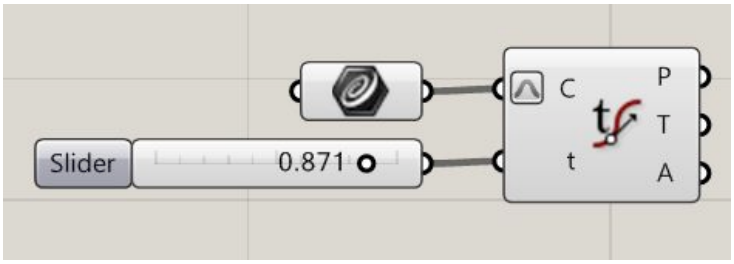
- *Create a single curve tin the Rhino viewport hat passes through or nearby the existing surface. This curve will act as the path along which the point will move.*
- *Reference the curve in Grasshopper using the "Curve" component.*



Next, evaluate the curve. When you evaluate a curve a point is created for the parameter you specify. The curve can be evaluated between the start and end. This is usually a parameter ranging from 0 to 1. So 0.5 would be half way along the curve. However, the parameters are most often than not, random. In order to reset the range back to 0 and 1 the curve can be re-parameterized.

- *Create a "Curve Evaluation" component and connect the "Curve" component.*
- *Right-Click on the curve input and select "Re-Parameterize".*

- *Create a “Slider” component and plug the output into the “Curve Evaluation”.*
- *By default a “Slider” outputs a number between 0 and 1. Use the “Slider: The slider becomes a timeline allowing us to evaluate the curve from beginning to end over time.*



The streaming data or value between 0 and 1 now determines the motion of a point along the curve. Imagine that the data could be coming from somewhere else , perhaps another TXT file , or perhaps and XML table from the internet. Either way, that data would determine the travel of this point.

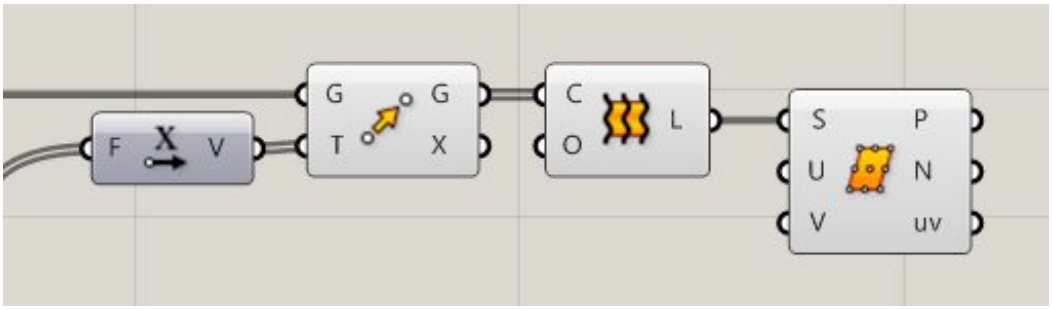
Relations

This is probably the most fundamental thing to know about programming spatially: Almost everything is based on relationships. Relationships are in fact what drive parametrics ie: if one object is modified , that will in turn modify another object. What we are concerned with in 3D design are spatial relationships, or simply distance and location. These two attributes determine the relationship. Conditional statements can be setup to determine how they should relate, consider this conditional statement as an example:

> **Is the object close enough or far away enough in order to affect any other object, and if so, does the magnitude of its impact on another object relate to that distance or not???**

Lets setup a definition that puts this into practice. The point which now moves along the curve, using the “Slider”. This will affect the control points of the curves that form the lofted surface we created and in turn deform the surface itself.

- *In order to gain access to the surface control points we use the “Surface Divide” component.*

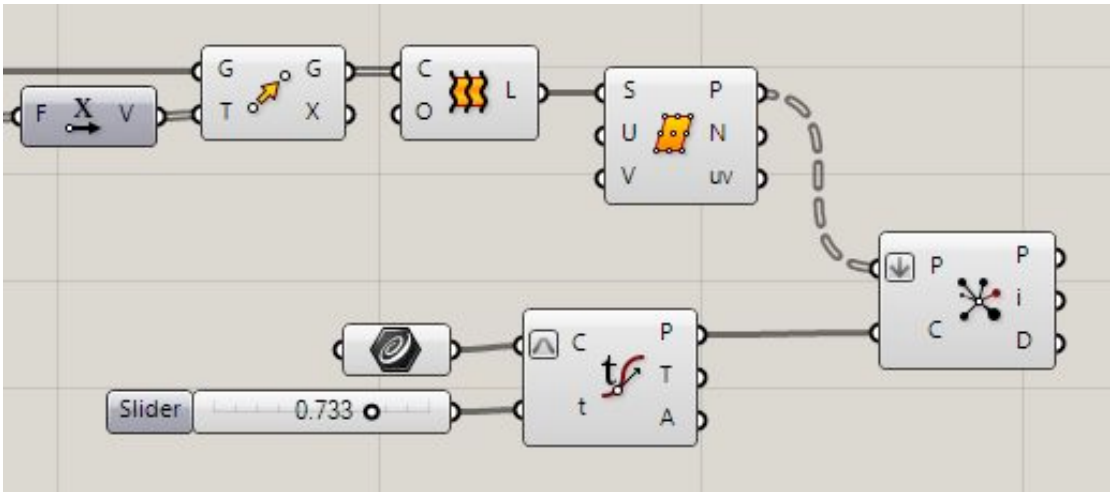


- *Create a “Closest Point” component. This component calculates the distance between points. There is a “Distance” component that does the same thing, however it does not handle multiple sets of points very well.*

- *Connect the “P” output from the “Surface Divide” component to the “Closest Point” components “P” input and the point from the “Curve Evaluation” component to the “C” input.*

- *The “P” output from the “Closest Point” component tells us which points are closest to which. The “D” output is the distance we want.*

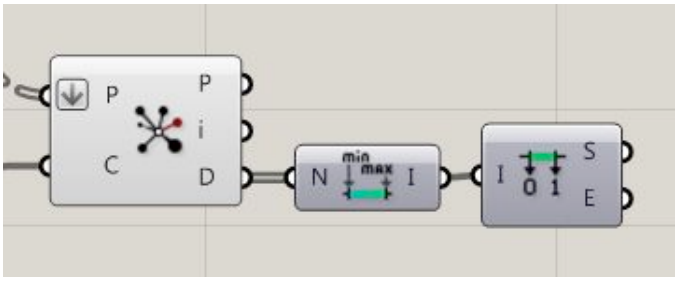
- *There is some house keeping that needs to occur here. Note how the wires are now dashed. This is because the list of points coming from “P” is not a single list , but rather a list of lists. For each row of points a list is created. This will be explained in the future, for now though, the list needs to be flattened into a single list. Right-Click on “P” on the “Closest Point” component and select “Flatten”. A small downward arrow will pop up beside “P” indicating that the list coming in is been flattened.*



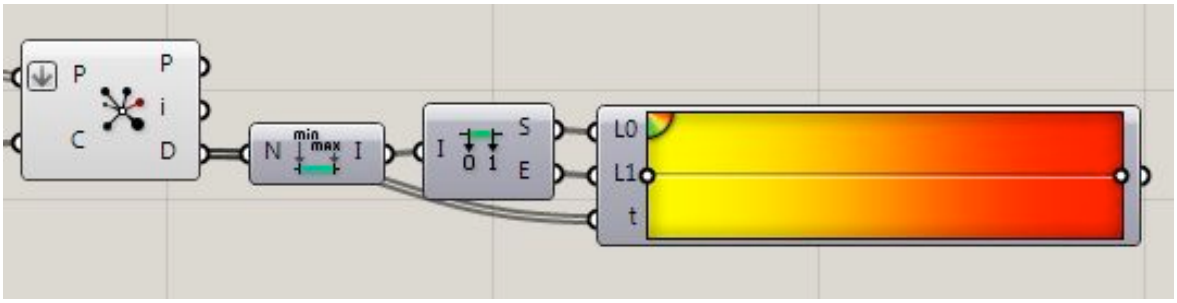
Once the distance is calculated we can use it to manipulate the points in numerous ways. For now , we will modify only the color of the points. The closer the traveling point gets to the control points the more red it will become and inversely the more yellow.

- *Create a “Bounds” component and plug the distance output into this.*

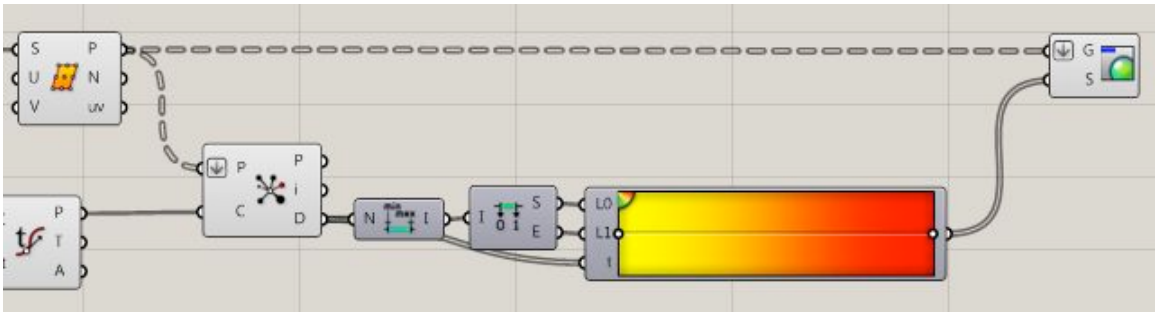
- *Next, create a “Domain Components” component and connect the output from “Bounds” in. This should give you the smallest and largest distances calculated.*

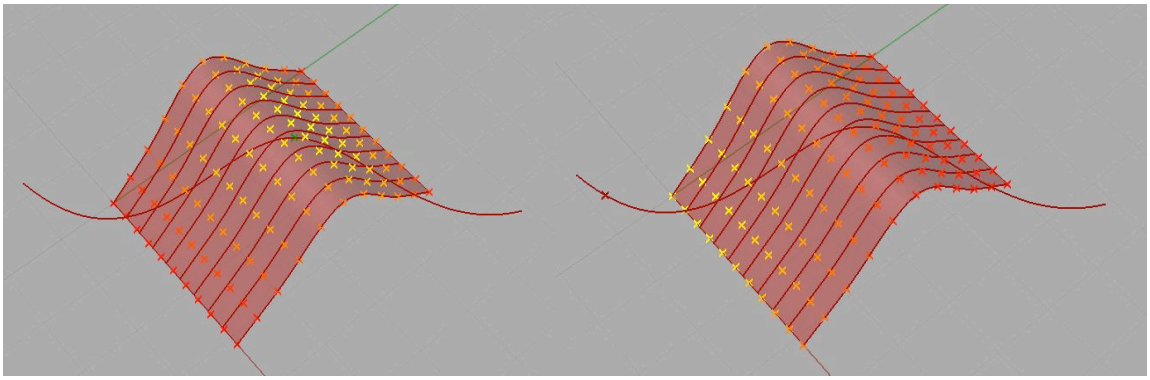


RGB values. It requires the smallest and largest numbers as a range though, T0 and T1. These are already given to us by the "Domain Components". Plug these in.



- *Next we need the distance values, plug the "D" output from "Closest Point" into the remaining gradient input.*
- *The last step is to create a "Preview" component which allows us to plug in any geometry and a set of RGB values and these objects will be colored accordingly in the Viewport. Since we previously flattened the points coming from the "Subdivide Surface" component going into "Closest Point", we need to flatten the input "G" on the "Preview" component as well.*





***Assignment**:*

Taking is one step further, the distance can also be used to translate the position of the points. Your assignment is to created your own surface, just as we have done. Make it unique, try creating different shaped curves and lofting those together. Take it all the way to the last step we went through. Then move

the points in the Z direction based on distance. These points could then be used to recreate a mesh, representing the altered surface.

As a hint below is a screenshot of what the canvas looks like and the resulting mesh.

Send me both your final Rhino and Grasshopper files along with any TXT files you might have created. Please include screenshots of your final Viewport.

Important: You MUST save both the Rhino file and the Grasshopper file separately. Saving one will not save the other!