1_algorithmic modeling with Grasshopper®

> "Grasshopper seems to be winning out in the competitive struggle for domination as the preferred tool for scripting, at least in the avant-garde segment of the discipline".
>
> Patrik Schumacher

The aim of this publication is to provide techniques and strategies which enable designers to master visual scripting utilizing **Grasshopper®**, one of the most popular and advanced algorithmic modeling tool.

This book is intended to guide readers through the essential concepts of algorithmic modeling both for *form-making* and *form-finding*.

Grasshopper, a plug-in for Rhinoceros®, is a node-based editor developed by David Rutten[6] at Robert McNeel & Associates. Created in 2007 for Rhino 4.0 – originally named as *Explicit History* – was re-branded as Grasshopper in 2008. Within a few years the plug-in gained a vast community of users and developers, including students, academics, and professionals. Grasshopper is available as a free download and it runs on a licensed copy of Rhinoceros 5.0 or higher.

NOTE 6

David Rutten is a graduate of TU Delft Architecture and Urbanism faculties. He works for Robert McNeel & Associates since 2006 on several programs, the most important of which is the Grasshopper visual programming environment for Rhinoceros 3D. See A. Tedeschi, 2010, Interview with David Rutten. MixExperience magazine, "Tools" issue.
http://content.yudu.com/Library/A1qies/mixexperiencetoolsnu/resources/index.htm.

Grasshopper's is an unrivaled platform for formal exploration, that benefits from:

- **Wide, dynamic and growing community**

  Grasshopper is not just a piece of software, it includes a dynamic network of users that share works, knowledge, ask questions, and discuss challenging problems at **www.grasshopper3d.com**.

- **Constant updating**

  Grasshopper benefits from constant updating and improvements. Moreover, bug fixing and new features are commonly based on users feedback.

- **Ecosystem**

  A wide set of plug-ins developed for Grasshopper are available from independent programmers. Loosely speaking, Grasshopper benefits from a plug-in ecosystem that extends the software's potentials. For example plug-ins are available for dynamic simulations, physics, structural and environmental analysis.

- **Software interaction**

  Grasshopper has the potential to interact with other software; not only for file-compatibility and interchange, but for real-time interaction between the algorithmic modeling environment and external software (Excel®, Photoshop®, Revit®, Ecotect® etc.).

- **Hardware interaction**

  Grasshopper's plug-ins enable interactions in data input/output between Grasshopper and hardware (Arduino®, Kinect® etc.).

This book will cover visual algorithmic modeling using Grasshopper, from the basic concept to more advanced strategies. The main ambition is to provide the computational skills - based on a solid understanding of mathematics, logics and geometry - required to face the new design challenges.

1.1 Prerequisites and installation

Grasshopper is not a standalone application, it operates as a plug-in for Rhinoceros. The installation file can be downloaded from the website **www.grasshopper3d.com** (*download* section). Presently, Grasshopper exclusively runs on Windows OS (Grasshopper is not currently compatible with Mac OS). Grasshopper is offered as a free download – without an expiration date – and is required to run on a licensed copy of Rhinoceros 5.0 or higher. Future releases of Rhinoceros might embed Grasshopper as a native feature.

Once installed, Grasshopper can be opened by typing *grasshopper* in the Rhino command line.

1.2 Grasshopper user interface

The Grasshopper editor consists of a window (A) that **always works in parallel** with the Rhinoceros 3D-modeling environment (B). Within the editor (A), users can build visual algorithms (C) by properly connecting graphical objects, called **components**. Components are actually the nodes of a parametric diagram that defines and controls a 3D geometry (D) which is displayed in the Rhinoceros window (B). Components represent primitives, geometric operations, logical functions etc.

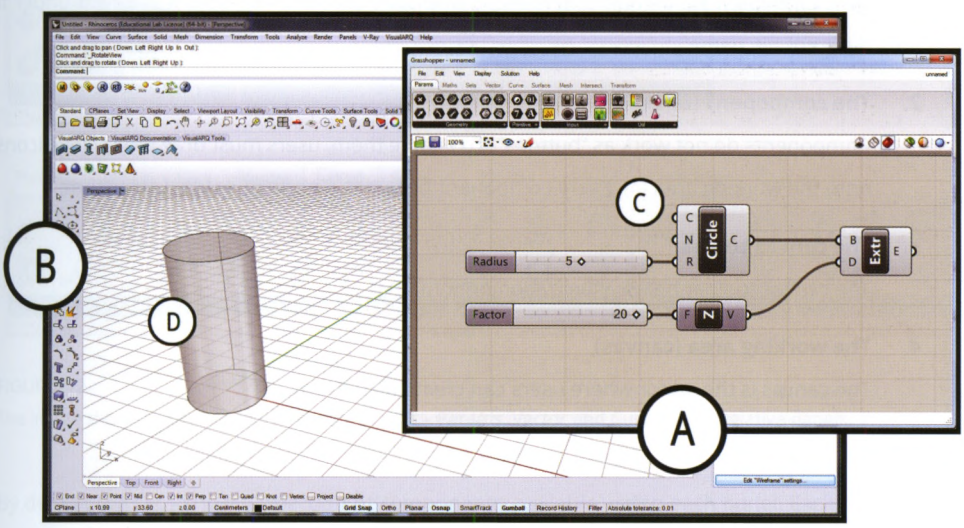
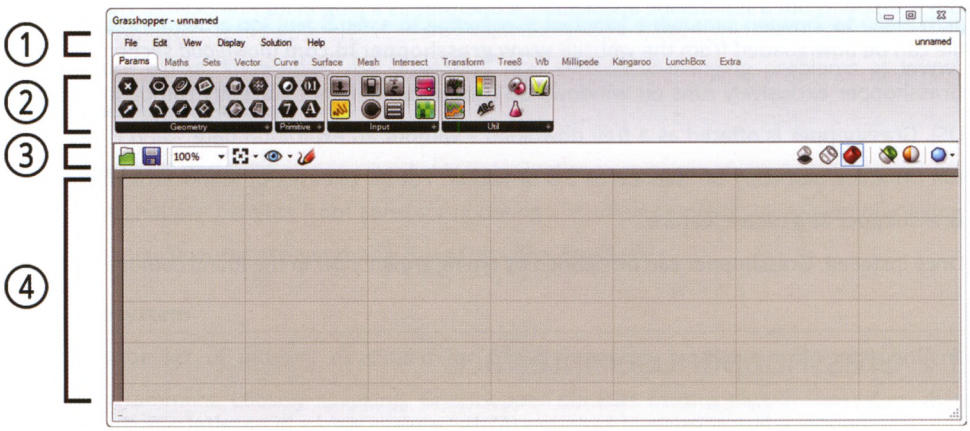


FIGURE 1.1

The image shows the Rhino modeling environment (B) and the Grasshopper editor (A) which is a window that works in parallel with Rhinoceros. D is a geometry "generated" by the visual algorithm (C).

The Grasshopper window is split into four sections, as shown in the following image.



1. **The menu bar**

   Following Windows OS typical bar layout, the menu bar performs basic operations (open, save, etc). On the far right, users can find the *file browser button* which can be used to switch between different loaded files; Grasshopper allows multiple files to be loaded simultaneously. By default a new file is displayed as *unnamed* until it is saved under a different file name.

2. **The component tabs**

   Components do not work as "buttons". To enable them, users must drag the relative icons onto the working area (4). Every component becomes a node of a visual algorithm.

3. **The canvas toolbar**

   The canvas toolbar hosts visualization options.

4. **The working area (canvas)**

   The canvas is the editor where users can create algorithms.

1.2.1 Component tabs

Grasshopper algorithms are node diagrams made of properly connected components. Components represent primitives (e.g. points, curves, surfaces), geometric entities (e.g. vectors), geometric operations (e.g. extrusion, rotation, revolution) and other categories. They are grouped in several **tabs** (*Params*, *Maths*, *Sets*, etc.) each organized in **panels.** For example, the *Params* tab holds four panels: *Geometry*, *Primitive*, *Input* and *Util*. Each panel has a variable number of components. This book will use the notation *component name* (Tab > Panel) to indicate the location of a specific component. Put another way, *Line* (Params > Geometry) means that *Line* component is found within the *Geometry* panel of the *Params* tab.

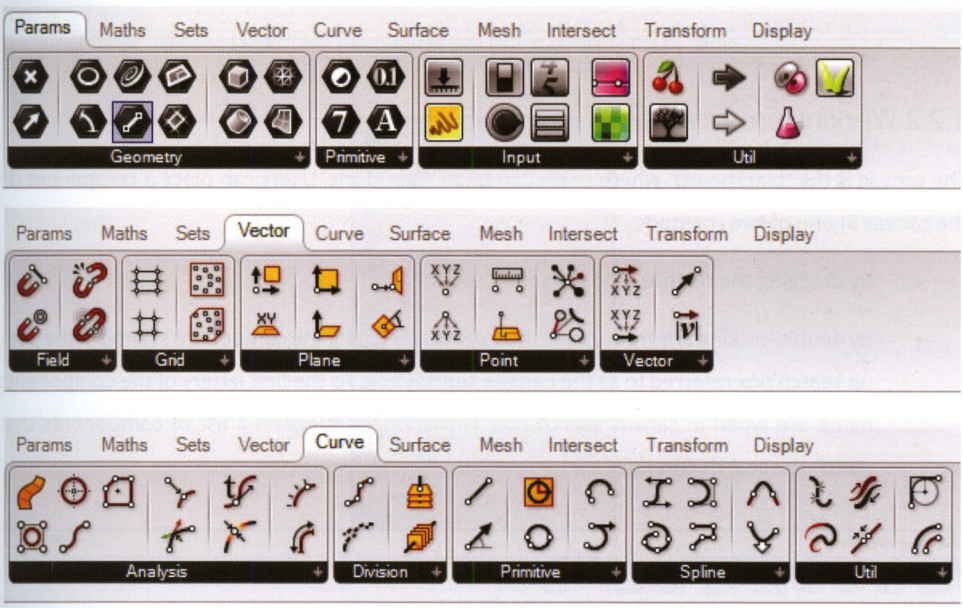
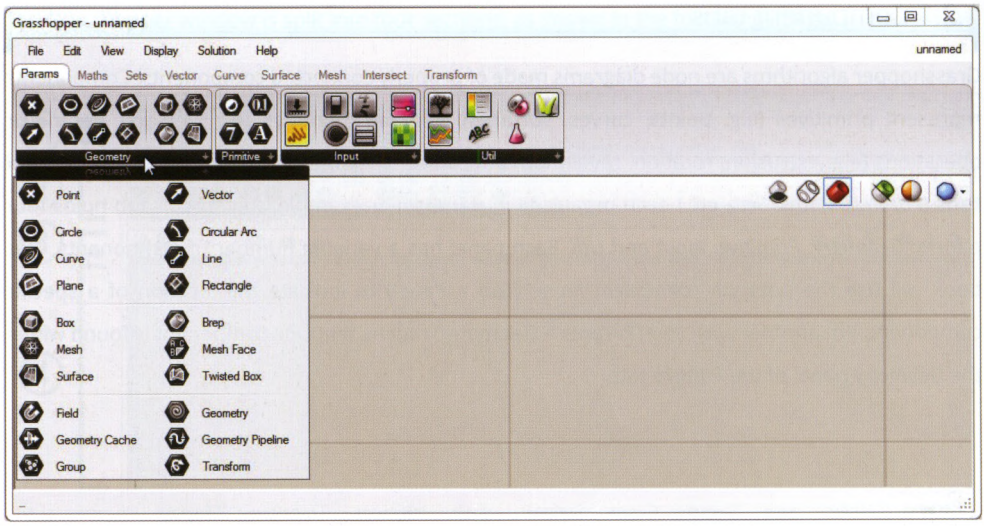


FIGURE 1.2
The image shows three different component tabs – Params, Vector and Curve – with relative panels.

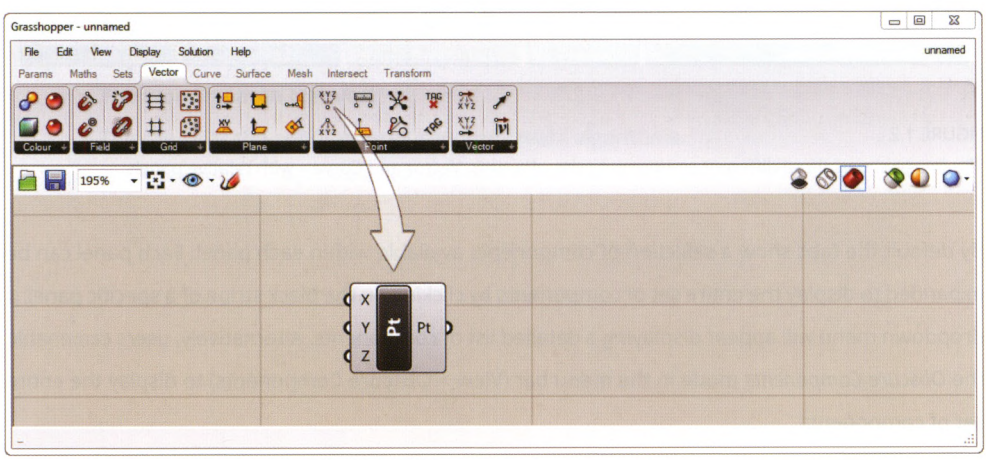
By default the tabs show a selection of components available within each panel. Each panel can be expanded to display the entire set of components by clicking on the black stripe of a specific panel: a dropdown menu will appear displaying a detailed list of components. Alternatively, users can enable the *Obscure Components* mode in the menu bar (View > Obscure Components) to display the entire set of components.



1.2.2 Working area: canvas

The canvas is the "blackboard" where users can build algorithms. Users can place a component on the canvas in one of two methods:

- By dragging the relative icon onto the canvas;
- By double-clicking any free area of the canvas and typing the component's name in the pop-up search box referred to as the **canvas search box**. As the first letters of the components name are typed in canvas search box, Grasshopper suggests a list of components that match the search criteria.



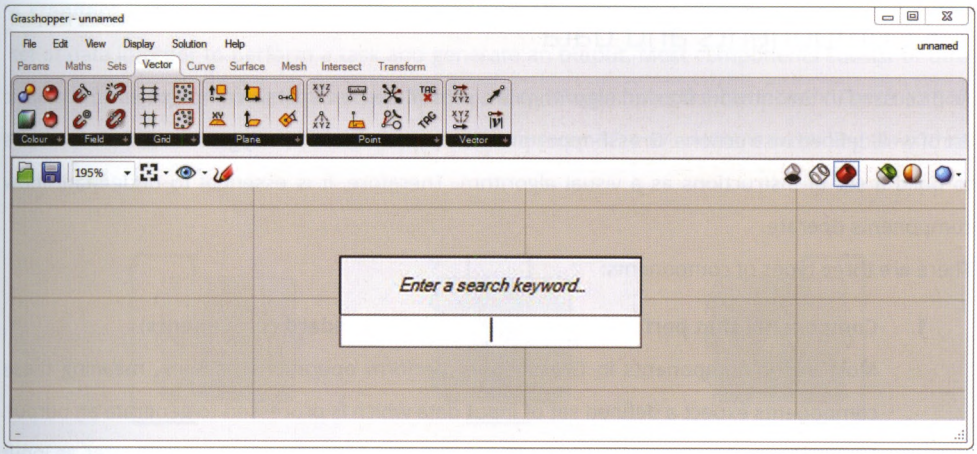


FIGURE 1.3

The image shows the *canvas search box* which allows you to find components by name.

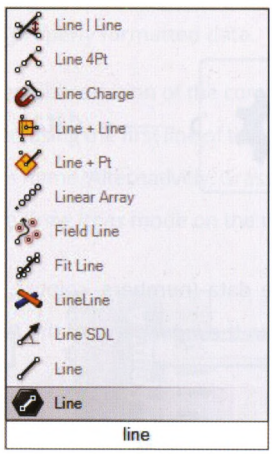


FIGURE 1.4

As you enter the first letters, Grasshopper suggests a list of components that match your search.

The canvas toolbar – positioned above the working area – provides the open/save files buttons as well as quick access to display and preview options.

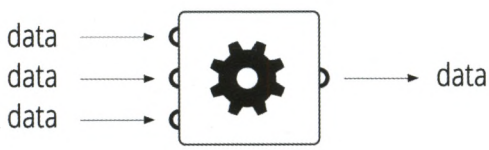
1.3 Components and data

As discussed in the introduction, an algorithm is a procedure that splits a complex task into a basic list of well defined instructions. Grasshopper provides users with a friendly interface of components to build a set of instructions as a visual algorithm. Therefore, it is essential to understand how components operate.

There are three types of components:

1. **Components that perform operations on data (standard components)**

   Most of the components in Grasshopper perform operations on data, meaning these components expect a defined set of input data which is processed to generate an output. For example, the point-component requires a set of numeric {x,y,z} coordinates as input to generate a point as an output, while a loft-component requires a defined set of curves as input to generate a surface as an output. The output of a component can be used as an input for another component.



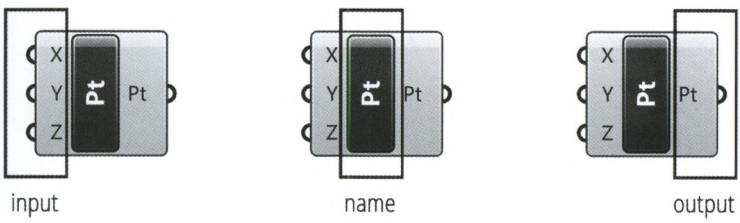
2. **Input-components**

   Input-components provide data (numbers, colours, etc) which can be modified by the user. They are hosted within the *Input* panel of the *Params* tab. Input-components do not expect input data.

3. **Components that store data (container components)**

   These components can be imagined as containers for data. They can collect data in several ways, and can be used as input for other components. Container components are hosted within the *Geometry* and *Primitive* panels of the *Params* tab and they are characterized by a **black-hexagonal icon**.

A **standard component** placed on the canvas is represented by a node that requires a defined set of data in order to perform a task and generate an output. Most components consist of three main sections: ***input, name,*** and ***output***. The following image shows the component *Construct Point* (Vector > Point) which generates a point in the Rhinoceros 3D environment according to its three coordinates:



- The **input section** contains a variable number of input slots that are specific to each component. For example, the *Construct Point* component requires {x,y,z} coordinates as input. Each input requires properly formatted data.
- The **name section** shows an abbreviation of the components full name (e.g. Pt for Point). A component can be renamed using the first line of the context menu, which appears by right-clicking on the component's name. Alternatively, Grasshopper can display an icon instead of a name if users activate the *Draw Icons* mode on the menu bar (*Display* > *Draw Icons*).

- The **output section** shows a variable number of outputs that are specific to each component. For example, the *Construct Point* component has one output slot that generates data in the form of a correctly formatted point defined by {x,y,z} coordinates.

If the *Construct Point* component is placed onto the canvas a point appears immediately in the origin of Rhino's space. This happens because *Construct Point*, like many components, contains default data. In particular, *Construct Point* contains, by default, the following coordinates as input: x=0, y=0, z=0. **The objects (points, lines, surfaces, etc.) generated by Grasshopper are displayed by default as red geometries in Rhino**.

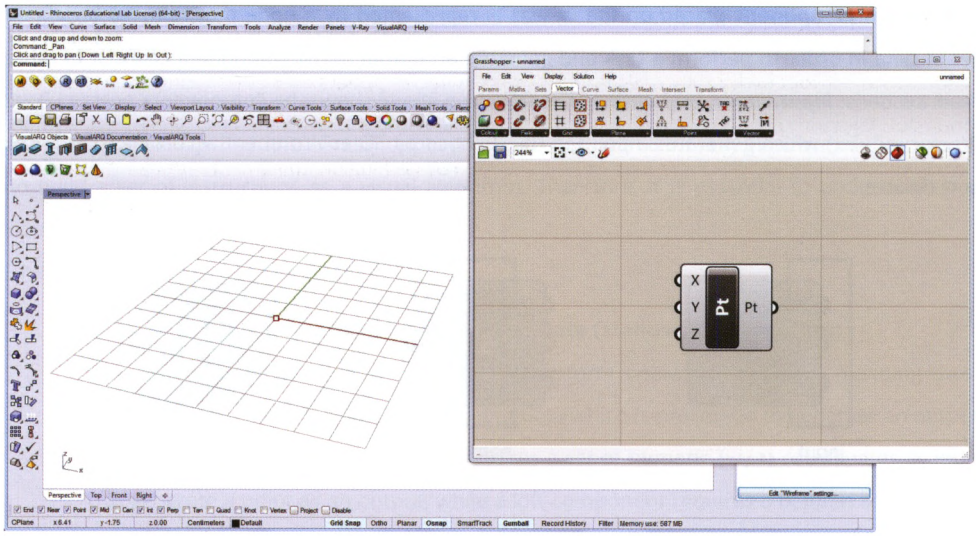


FIGURE 1.5
By default, the *Construct Point* component generates a point in the origin of axis.

To change the position of the point, users have to input values for the {x,y,z} coordinates. In other words, **data must be set inside the component**.

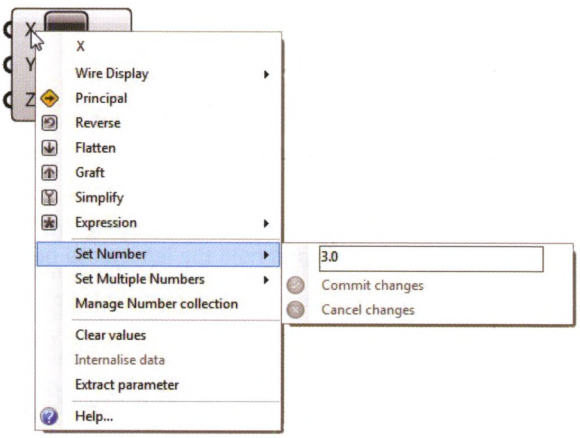
Grasshopper allows users to set data in three ways:

- **Local setting** (1.3.1);
- **Wired connection** (1.3.2);
- **Setting from Rhino** (1.3.4).

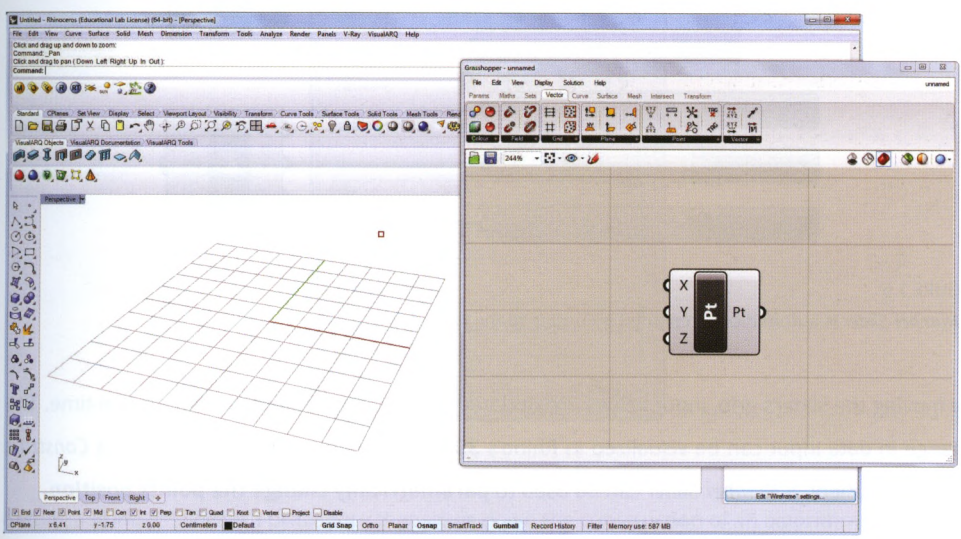
In general, an algorithm is constituted by components which use all three strategies. The next paragraphs discuss each method in detail.

1.3.1 Local setting of data

Data can be set **locally** by the *context pop-up menu* which appears by right-clicking on the specific input slot. For Example, to change the X input, right-click on the X and select the *Set Number* option. A new value (e.g. X=3.0) can be entered and confirmed by clicking on *Commit Changes*. As a result, the point moves three **Rhino units** in the x-direction, creating a point with coordinates {3.0, 0.0, 0.0}.



If we repeat the procedure for Y and Z by entering, for example, Y=3.0 and Z=2.0, the *Construct Point* component will generate a point with the coordinates {3.0, 3.0, 2.0}.



1.3.2 Wired connection

To build more complex algorithms **wired connections** are used to set data between components. Wires conduct data from the output of a component to the input of another component.

The *Construct Point* component can receive data from other components that output numeric data. For instance, the **input-component** *Number Slider* (Params > Input) generates a domain of numeric data. By adjusting the central grip, a number can be output within the domain and can be used as input for other components.

To transfer data to another component, press and hold the left mouse button on the output slot of the number slider. A connecting wire will be extracted which can be used to transfer data to the *Construct Point* component by dragging the end of the wire to the specified input slot. Once a connection is established, the *Number Slider* will be renamed based upon the input the wire is connected to.

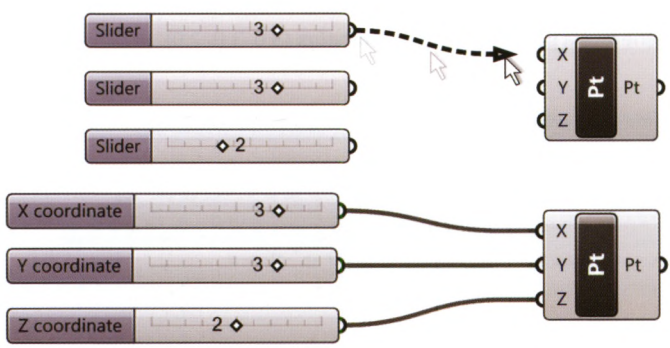
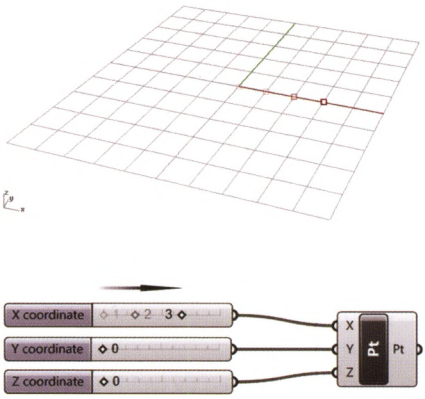
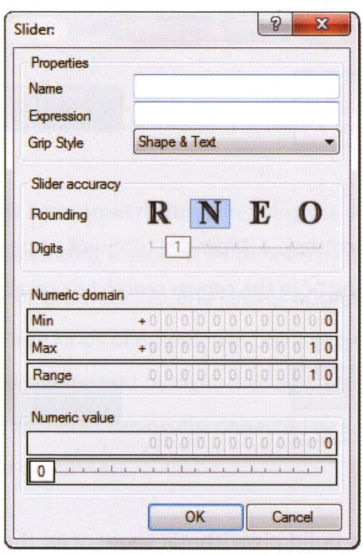


FIGURE 1.6
A *Number Slider* is automatically renamed according to the input name.

By moving the sliders grip, input values are updated changing the points position in real-time. The change in data input can be visualized in Rhino's 3D environment. Once connected to a *Construct Point* component, a *Number Slider* can be used to **parametrically change the points position**.

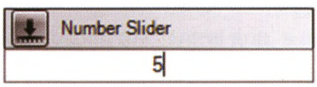


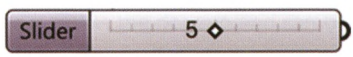
Slider properties can be changed through the context menu accessible by double-clicking the slider name (not on the grip). Within the context menu the *Numeric Domain* or the Min/Max value of the slider as well as numerical *Rounding*: *Floating Point Numbers* (R), *Integer Numbers* (N), *Even Numbers* (E) or *Odd Numbers* (O), can be specified. If the *Number Slider* rounding is set to R, the amount of displayed *Digits* (decimal places) can be set.



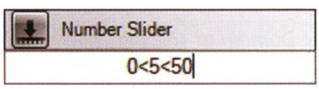
Setting a *Number Slider* is a procedure that slows down the construction of an algorithm: users have to change the *Numeric Domain* (min and max) and define the numerical set.

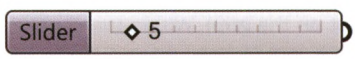
A quicker and easier way to insert a slider with a specific value is to double click on a free area of the canvas to recall the *canvas search box.* If a number (e.g. 5) is typed into the *search box* Grasshopper suggests the *Number Slider.* If the slider is added to the canvas the component will be set on the typed number (i.e. 5) with a defined default domain (i.e. 0-10).



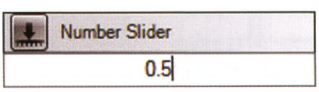


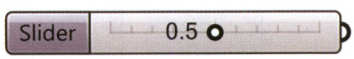
The domain and a value for a slider can specified by typing 0<5<50 in the *canvas search box*. In this case, the domain will be set between 0 and 50 and the slider will be set to a value of 5.



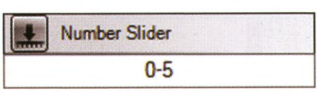


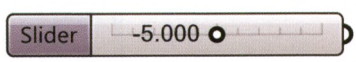
If 0.5 is typed in the *canvas search box* a slider set to *Floating Point Numbers (R)* will appear on the canvas with one digit after the zero, a default domain between 0 and 1 and a value of 0.5.





If 0-5 is typed in the *canvas search box,* a *Slider* set to -5 will appear on the canvas with a default domain between 0 and -10. Typing "-" in the *canvas search box* recalls the component: *Subtraction*.





Wired connections are used to build algorithmic sequences, they transfer data between output and input of the components. For example, a line can be generated by connecting two points. In this case a specific component is used: *Line* (Curve > Primitive). If the sliders values are changed, the points {x,y,z} positions are recalculated and the magnitude and direction of the line updates associatively.

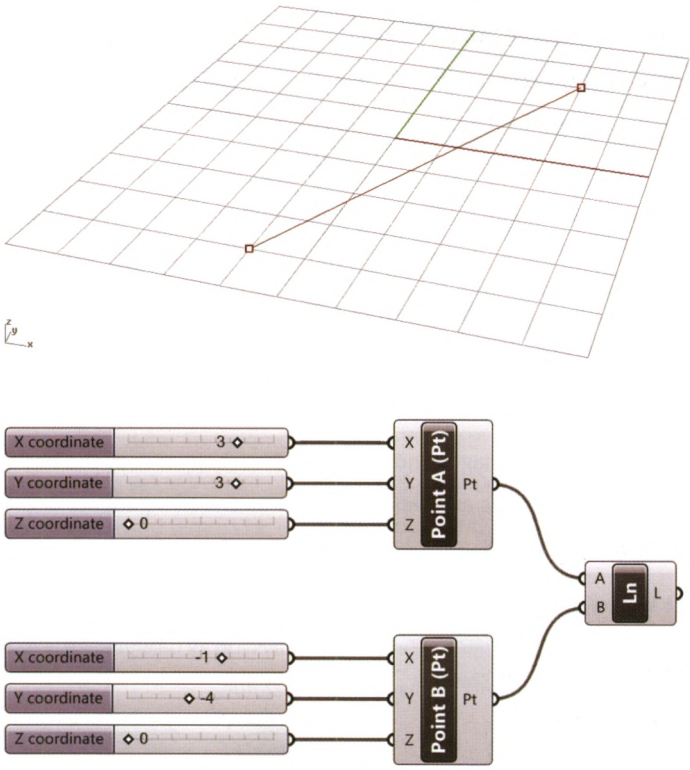


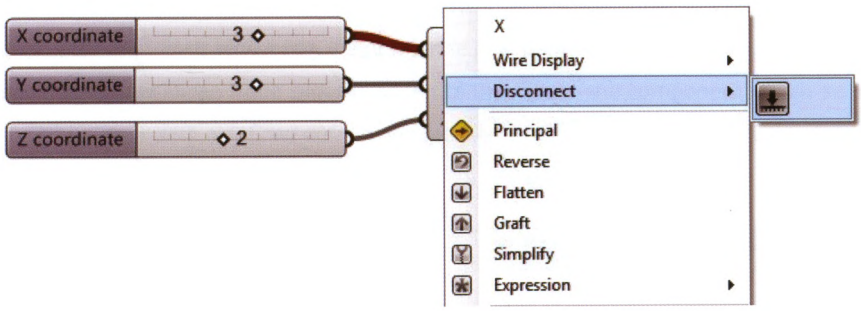
FIGURE 1.7

A line generated as a connection between two points (renamed as Point A and Point B).

The *Line*-output can be defined as a **parametric line**, meaning the line is tied associatively to the six slider parameters of the points, *generating* a set of lines defined by the possible combinations of the sliders domains. Algorithms do not generate just one kind of output.

The algorithm above generates three kinds of output: two point-geometries and one line-geometry.

Established connections can be disconnected by right-clicking on the specific input slot: the *context pop-up menu* will appear, and components can be disconnected by selecting the *disconnect* submenu and specifying the component to disconnect.



1.3.3 Warnings and errors

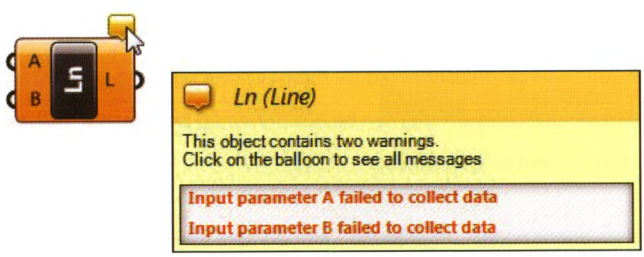
Algorithms can produce error messages and warnings. Grasshopper uses the background color of components to indicate their status and report possible errors. Components display status based on three different colors:

- **Correct status (gray)**

  A component which is properly connected displays a **gray** background. A working algorithm is composed of components displaying *correct status*.

- **Warning status (orange)**

  A warning status is indicated by an **orange** background. Usually, **a warning is related to a lack of data**. For example, when a *Line* component is placed on the canvas it is in warning status: displayed as orange. This is because the component requires two points (A and B), in order to create a line. If points A and B are connected to input A and B, the *Line* component turns gray and generates a line in Rhino.

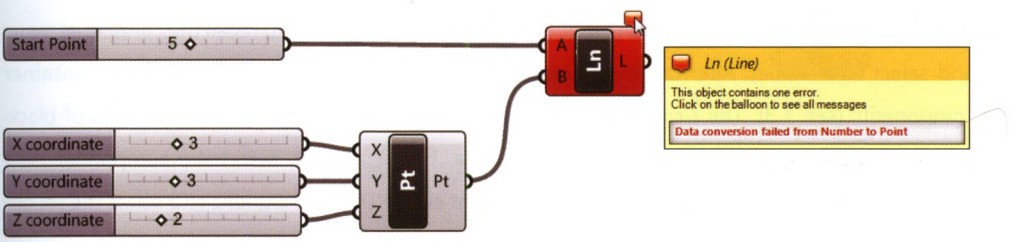


A component in warning status always displays a "balloon" (if it does not appear users can activate this function through *Display* > *Canvas Widgets* > *Message Balloons*). Hovering the mouse over the balloon activates a message-box which suggests the possible causes of the warning.

An algorithm with "orange components" may still work in some cases, but in general, a lack of data leads to unexpected or null results.

- **Error status (red)**

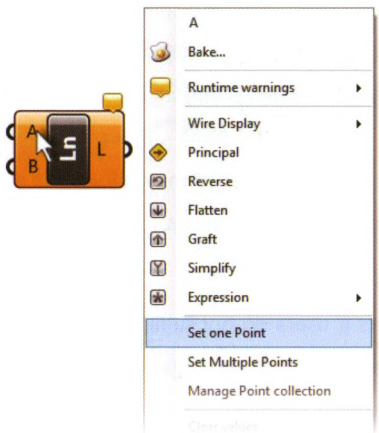
  An error status occurs if users do not fulfill the input requirements. In the comparison between algorithms and recipes it was noted that some steps require "ingredients", others "temperatures", still others require "times". Similarly, each component's input expects a defined type of data. For example, if users connect the A-input of the *Line* component using a numeric value instead of a described point {x,y,z}, the *Line* component turns **red** because input A and B must be formatted as a point.



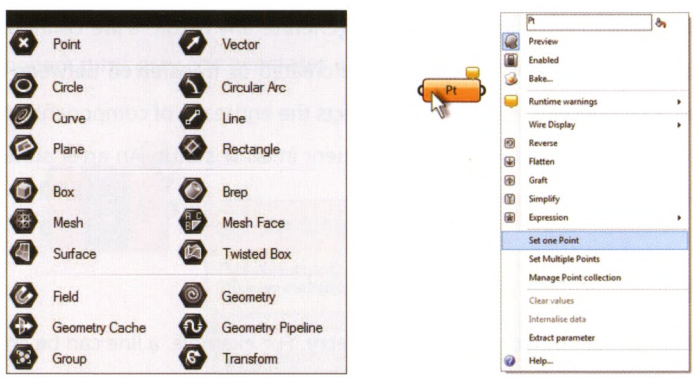
A component in error status does not generate any result. If we connect a component with a "red component" no data will be created or transferred between them. For this reason, a component in error status affects the entire set of components that are directly or indirectly connected with the component in error status. An error status can be fixed by correcting the input data.

1.3.4 Setting from Rhino

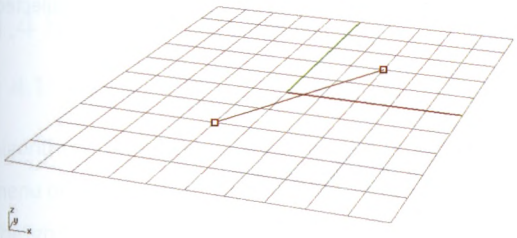
Data can also be set from existing Rhino geometry. For example, a line can be created between two points that exist in Rhino. Using the *Line* component users can right-click on the A-input slot and select the *Set one Point* option from the contextual menu. The *Set one Point* option allows users to select a point from Rhino. When users click on *Set one Point*, Grasshopper's window minimizes, allowing the selection.

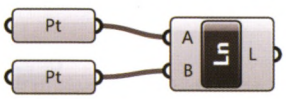


Once users set both A and B input points, the component turns gray and generates a line. Such a method directly stores a Rhino's objects inside an input. If the points move in Rhino the line will be associatively updated. To "empty" an input, right-click on the input slot to recall the contextual menu and select *Clear values*. Alternatively, Rhino's geometry can be collected using specific **container** components (see 1.3). The *Geometry* and *Primitive* panels of the *Params* tab hold a set of black-hexagonal icons for **components that store data.** To set a point, select the component *Point* (Params > Geometry) and place it on the canvas, right-click on the component and select the *Set one Point* option within the context menu.

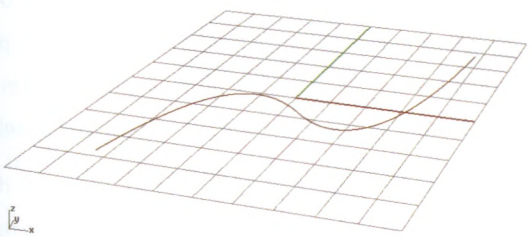


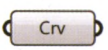
For example, to create a line between two Rhino points: collect the points data with two *Point* components and connect them with wires to a *Line* component.



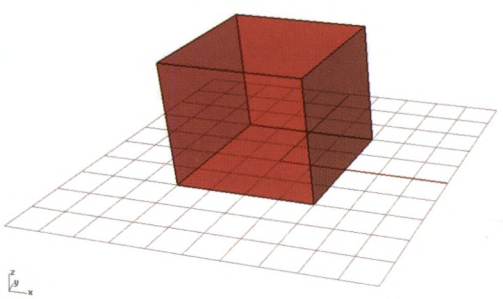


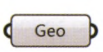
The container components do not perform any operations, they just store data. These components can be used to collect any type of geometry or object from Rhino. Each container component expects a defined type of geometry. In order to collect curves, the *Curve* component would be selected. Correspondingly, a *Point* component is unable to collect curves.



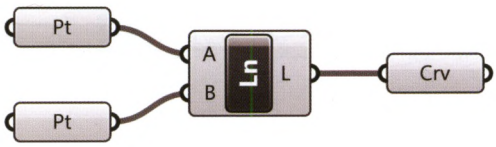


The *Geometry* container is a versatile component which can be used to store any kind of geometry (points, curves, solids etc.). To set a geometry or multiple geometries right-click on the *Geometry* component and select the *Set one Geometry* or *Set Multiple Geometries* option within the context menu, respectively.





The container components also provide an input slot that allows components to store data collected from other components.



Grasshopper's geometry, displayed in red, overlaps the imported geometries that exist in Rhino. When geometry is "set", Grasshopper establishes a continuous link with Rhino, meaning if geometry is deleted in Rhino, the specific container component turns orange because it is no longer collecting data.

Container components allow data to be permanently stored by right-clicking on the geometry component and selecting the *Internalise data* option from the context menu. If this option is selected, Grasshopper will import the geometries data, breaking the link with Rhino, meaning if the original geometry is deleted the specific component will continue to store the data. For example, if a point is imported from Rhino using the *Point* component and internalised, the point can be deleted in Rhino and still exist as stored data in Grasshopper. The points {x,y,z} location can be changed by selecting the *Point* component and using Rhino's *Gumball* to change the point's position.

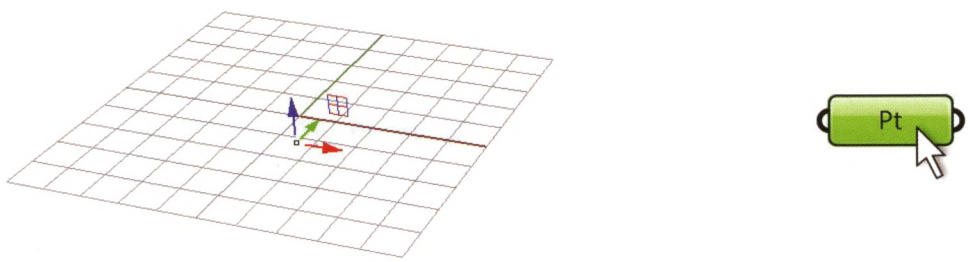


FIGURE 1.8
If you *internalise* a point in Grasshopper and then you click on the relative box-component, the Rhino's Gumball appears on the point, so you can easily change its position.

1.4 Save and bake

1.4.1 Save

Algorithms can be saved using the *Save Document* or *Save Document As* options available in the *File* menu of the *Menu Bar*. The default extension for Grasshopper's files is **.gh**.
The .gh files are not executable. Therefore, to open a previously saved algorithm, users have to start Rhinoceros, load Grasshopper and open the desired file.

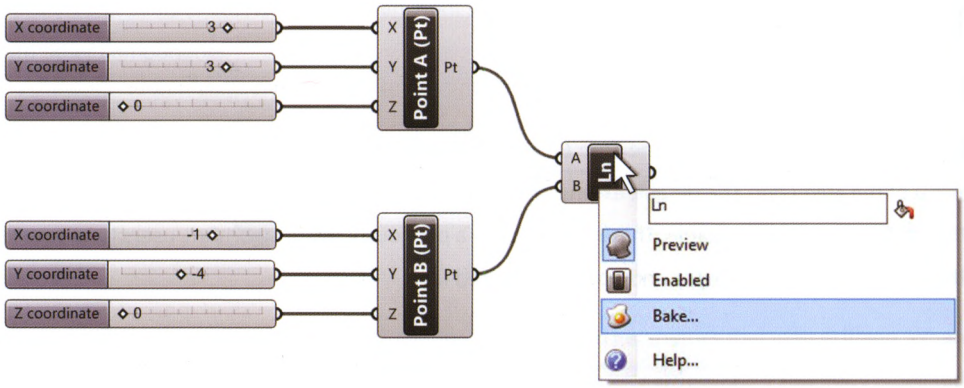
grasshopper file gh.gh

grasshopper file ghx.ghx

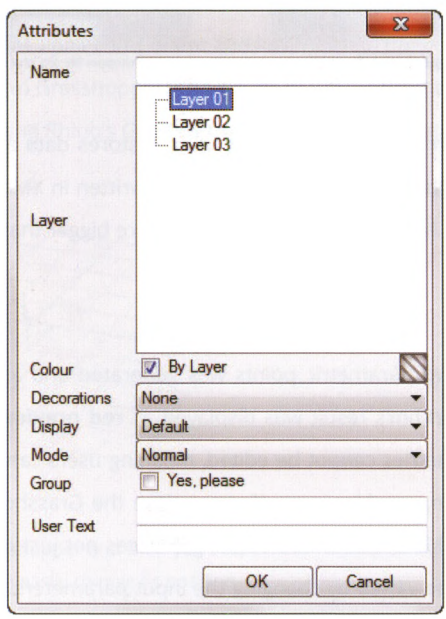
The .gh file extension is formatted in binary, meaning it stores data as pure bytes. Grasshopper files can also be saved as **.ghx**. The .ghx file extension is written in XML format, meaning it can be modified using text-editors. For this reason, the .ghx files are bigger than .gh files.

1.4.2 Bake

In the previous sections, two parametric points was generated and associated to a line creating a basic algorithm. The algorithm's result was displayed as **red preview geometries** in the Rhino environment. Preview geometries cannot be edited, meaning users cannot select them, save them as a Rhino file, render them, etc. Moreover, if users close the Grasshopper file these objects will disappear. The reason for this is that Grasshopper generates not just **one** geometry but an entire set of geometries that can be varied by changing the input parameters. To edit one of the possible configurations generated by an algorithm, users have to "**bake**" the Grasshopper preview into the Rhino environment. Every component that generates a preview can be baked by right-clicking on the component and selecting the *bake* option from the context menu. For example to bake the *Line* component, right-click on it and select *Bake...*. Baking the component makes the line editable in Rhino.



The *Bake Attribute* window, which opens when *Bake...* is selected allows users to set several attributes, such as: the **target layer**, the color of the baked geometry, and the possibility to group objects.



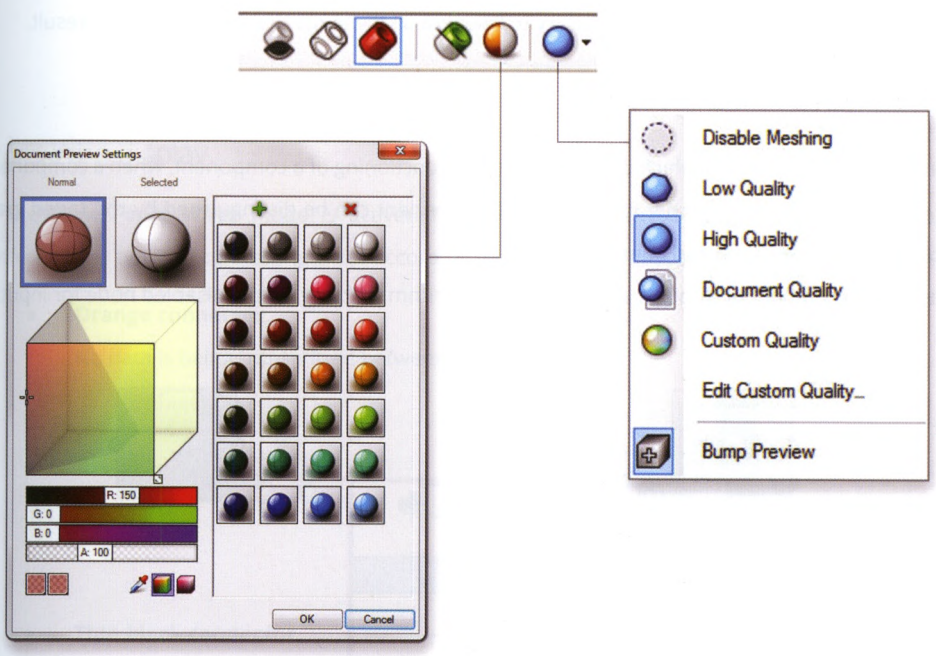
After baking, the geometry can be changed by manipulating the input parameters within Grasshopper. Grasshopper geometry is parametric and the number of solutions are only limited by the input parameters combinations. To undo a bake operation, simply delete the geometry in Rhino or type *undo* in the Rhino command line. Grasshopper cannot bake geometry if any command is running on Rhino.

1.5 Display and control

Grasshopper uses colors and graphics to report component states and display-modes.

1.5.1 Preview quality. Enable/disable preview

Working components generate a result that are, by default, represented by a red preview color in Rhino. Color and quality of the preview can be set through the *Canvas Toolbar*. In particular, the preview color can be set by the *Document Preview Setting* option, while the quality of visualization can be modified through the *Preview Mesh Quality option.* Both options are contained in the canvas toolbar.



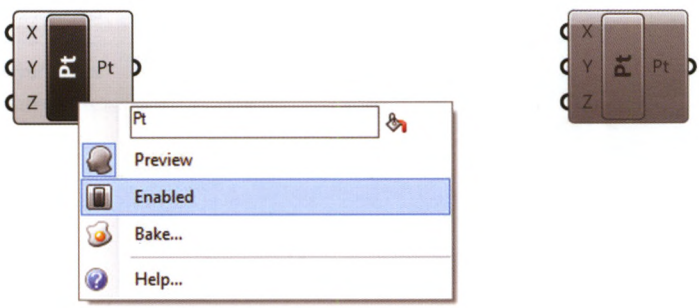
The preview of a geometry can be disabled by right-clicking on the name of the component and selecting the *preview* option from the context menu. When you disable the preview the component turns **dark grey**.

Enabling/disabling preview is crucial when algorithms become complex. Algorithms can be imagined as a construction history, it is unnecessary to visualize all the steps that lead to the final result.

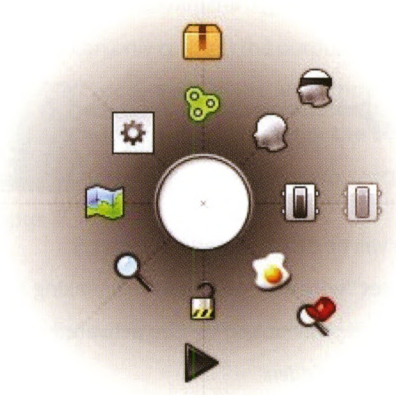
1.5.2 Enable / disable objects

Unlike the *Preview* option, which does not affect the functioning of a component, when a component is disable it will no longer operate. A disabled component can be distinguished by the **faded gray** color of the background and name section.

Disabling a component disables all parts of the algorithm that rely on the disabled node for input.



Many productivity tools so far discussed are included in the ***Radial Menu*** which can be opened by clicking the spacebar, including: navigation, preferences, group, cluster, preview/hide, enable/disable, bake, zoom, disable solver, recompute, and find. For example, to disable the preview of two or more components: select the components, press the spacebar and click on the "masked face" to disable the preview. Other tools of the *Radial Menu* follow a similar logic.



1.5.3 Wire display

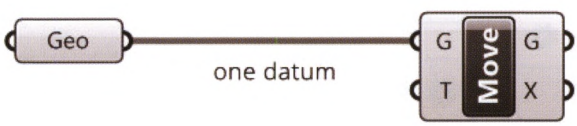
If the ***Draw Fancy Wires*** mode is activated (*Display menu* > *Draw Fancy Wires*), Grasshopper differentiates types of connecting wires according to their type of data structure. In particular:

- **Orange connector**

  No data is being transmitted between components.

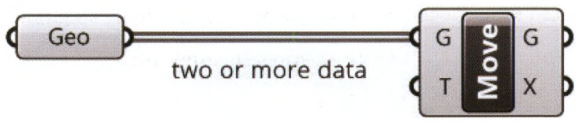
- **Thin black connector**

  Just one datum (one number, one geometry, etc) is being transmitted between components.



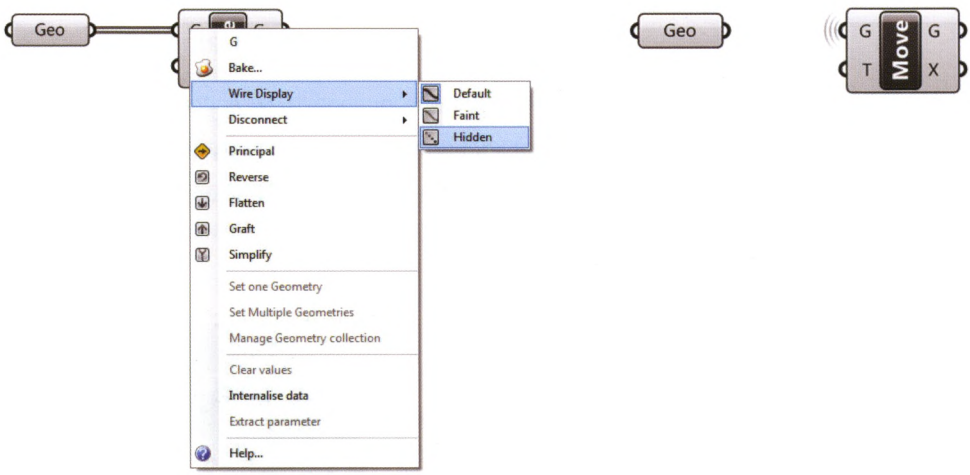
- **Wide black connector**

  Two or more data items are being transmitted between components.



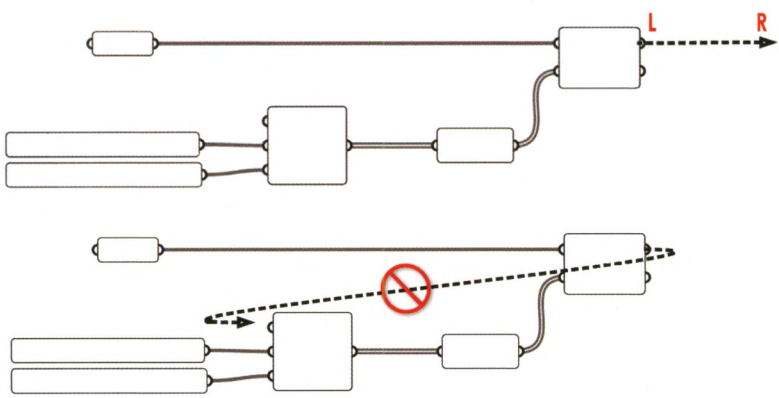
A fourth type, the dashed wire will be discuss later in chapter 5.

Grasshopper allows users to set "wireless" connections, making a wire invisible unless one of the connected components is selected. This option can be activated within the context menu of every input slot by selecting *Wire Display* > *Hidden.*

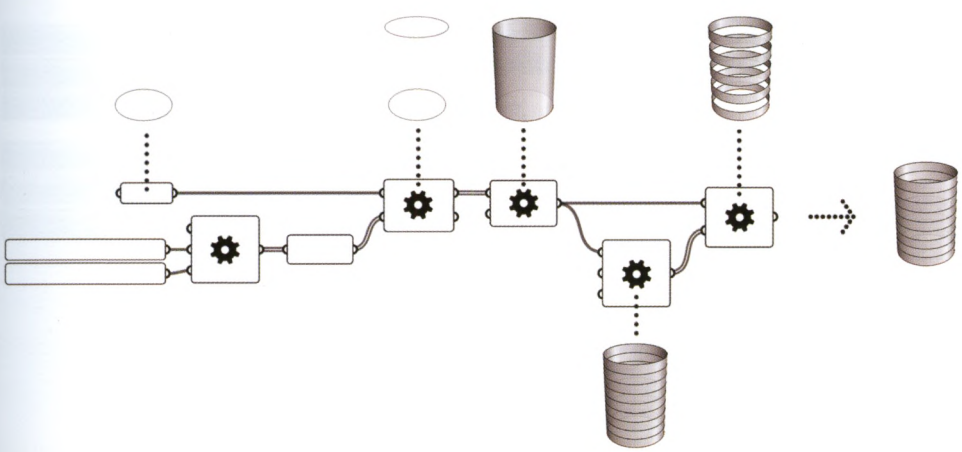


1.6 Grasshopper flow

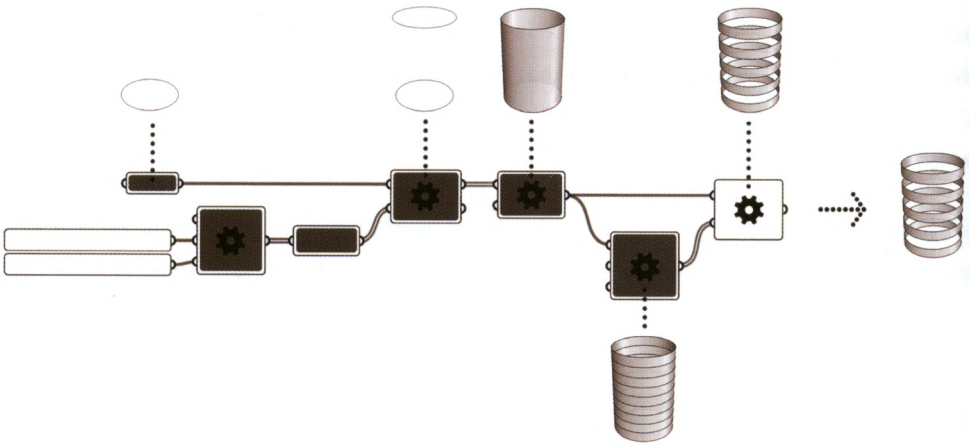
It is important to point out that wires can only connect the output of a generic component (A) with the input of a component (B) that doesn't precede (A) in the algorithmic sequence. In other words, **the data stream can be imagined as a fluid that flows through the components from left to right**. Consequently, is not possible to create a **loop** in Grasshopper, except using special components which can be discussed in chapter 7.



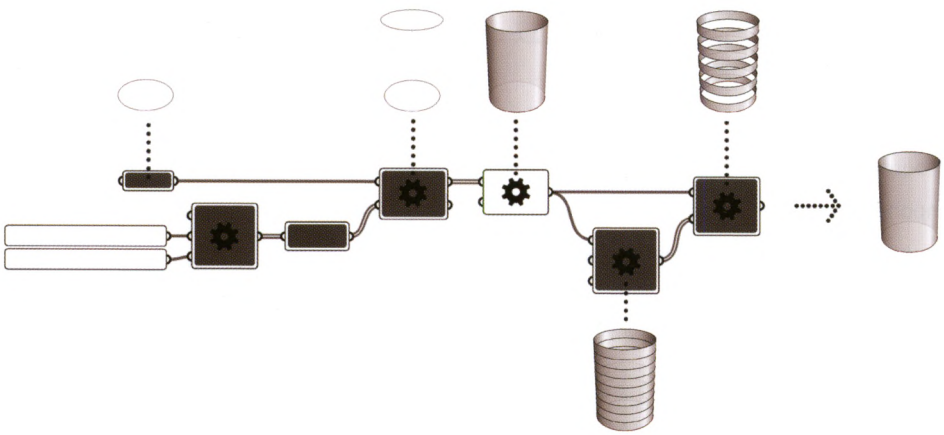
The Grasshopper logic sequence can be imagined as a ***construction history***: Grasshopper stores the output of each instruction. To display the final output of an algorithm all nodes except last component must be un-previewed. If users do not hide components the algorithm will generate geometry in which the final step overlap the previous steps.



In the following image all the components are turned off except the last component, allowing the final output of the algorithm to be visualized without overlap the previous steps.



In the following image all the components are turned off except an intermediate step.



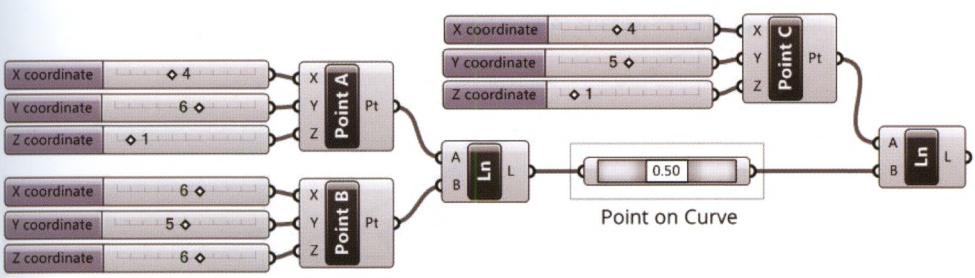
1.7 Basic concepts and operations

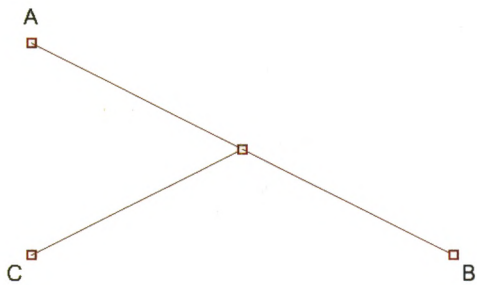
This section is a preview of topics that will be discussed, in depth, in following chapters. Here we will focus on components that find specific points on objects or perform basic operations such as moving objects, as well as visualization components.

1.7.1 Object snap in Grasshopper

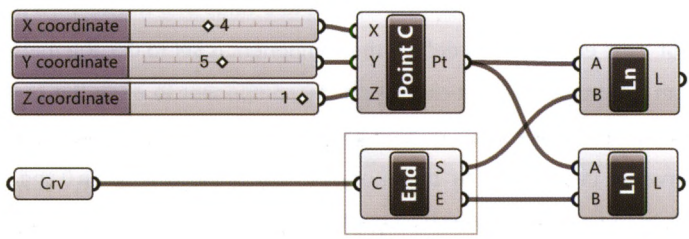
The drawing accuracy of CAD software relies on *Object Snaps* or *Osnap controls.* The *Object Snap* tools allows users to select strategic locations to connect objects. For example, a line has three main points that can be snapped to: a start point, a mid point, and an end point. Grasshopper does not function under the CAD snap logic, instead, it relies on defined algorithmic logic to create the same functionality.

Specific points on a curve can be found using the component *Point on Curve* (Curve > Analysis). The *Point on Curve* slider defines a point on a curve by "reparameterizing" a curve's length as a unit of 1. Correspondingly, the *Point on Curve* component set to a value of 0 will define a point at the start point and value of 1 the end point. By default the component is set to 0.5, the mid point. A specific numeric value can be chosen by right-clicking on the component to open the context menu and selecting the desired input value *(start, quarter, third, mid, two thirds, three quarters, end)*. Arbitrary points along a curve can be "snapped to" by adjusting the slider's grip between 0 and 1. The following example shows how to create a line which connects a point C to the midpoint of a line between two points A and B.

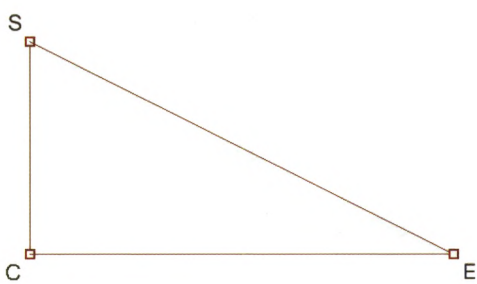




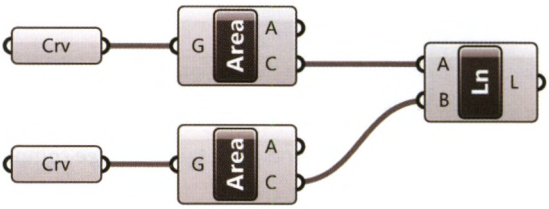
The component *End Points* (Curve > Analysis) can be used to quickly exctract the start point (S-output) and end point (E-output) of a given curve. The following example demonstrates how the *End Points* component can be used to access the end points of a curve set in Rhino (the curve could alternatively be described in Grasshopper), then use the *End Points* component output to generate two lines between the start and end points and a third defined point.

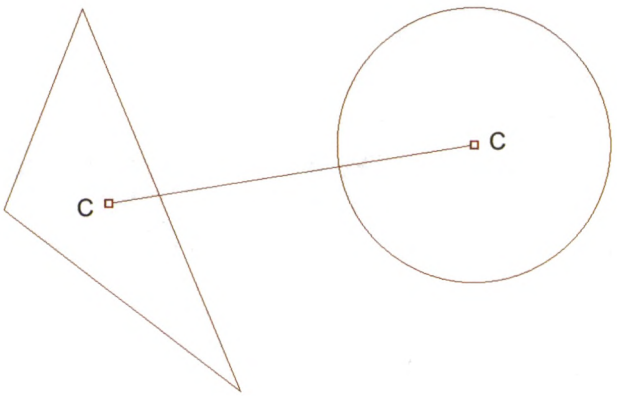


End Points

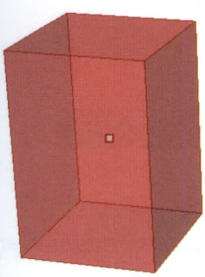


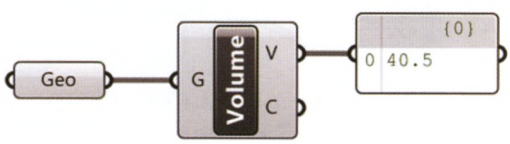
The component *Area* (Surface > Analysis) can be used to extract the geometric *centroid* of a planar closed curve. The component also calculates the area (A-output) of the input curve. The following example demonstrates how the *Area* component can be used to access the centroids of two closed curves, then connect the points to create a line.





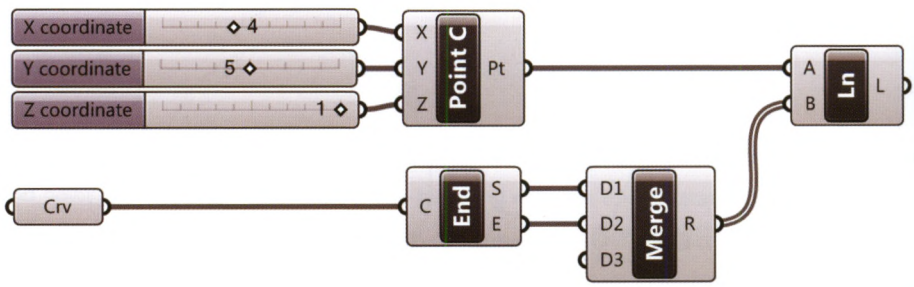
The component *Volume* (surface > Analysis) can be used to extract the geometric *centroid* of a closed three-dimensional geometry. The *Volume* component output calculates volume (V-output) and centroid (C-output). Data output, such as area and volume for the *Volume* component, can be displayed using the component *Panel* (Params > Input). The following example calculates the volume of a Rhino set geometry and displays the result as a single output in a panel. If multiple geometries were set, the panels would calculate the volume of each object, and display the results as specific items in the panel.





1.7.2 Merging data

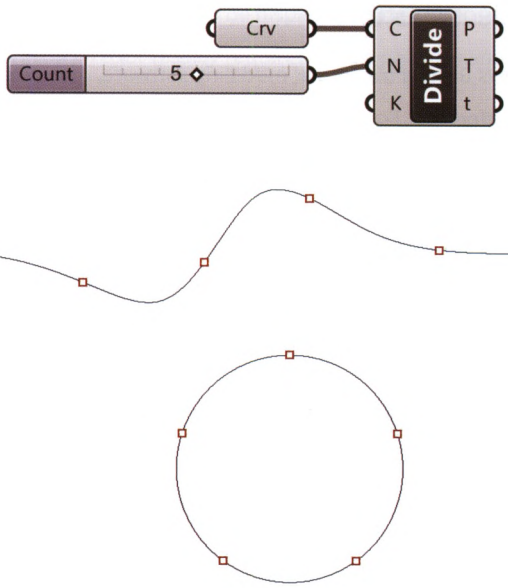
The component *Merge* (Sets > Tree) allows two or more data flows to converge in a single list. With reference to the previous exercise, the following example shows how to use just one *Line* component, instead of two, by collecting the start point and end point by the ***Merge*** component.



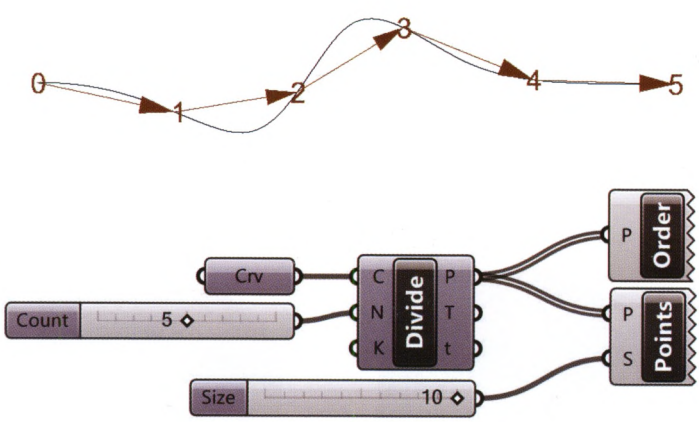
Wires containing more than one item are displayed as a thickened line. For instance, after the merging operation the wire contains more than one item and is displayed as a thicker line (see 1.5.3).
When the *Merge* component is placed on the canvas, it displays two inputs (D1 and D2). Once the D2-input is receiving data (i.e. has a wired connected), *Merge* automatically adds a D3-input. The *Merge* component maintains a record of the connection-order via the progressive index of the D-input.
Additional D-inputs can be manually added by using the ***Zoomable User Interface***. The interface (available on some components) adds an input by clicking on the "+" buttons which are visible after zooming-in on the specific component.

1.7.3 Dividing a Curve

The component *Divide Curve* (Curve > Division), divides an open or closed curve into equal arc length segments. More specifically, given a curve set from Rhino (C-input) and an integer for the number of subdivision (N-input), N+1 points (P-output) will be generated in case of an open curve and N points in case of a closed curve. The points will be stored in a single list.



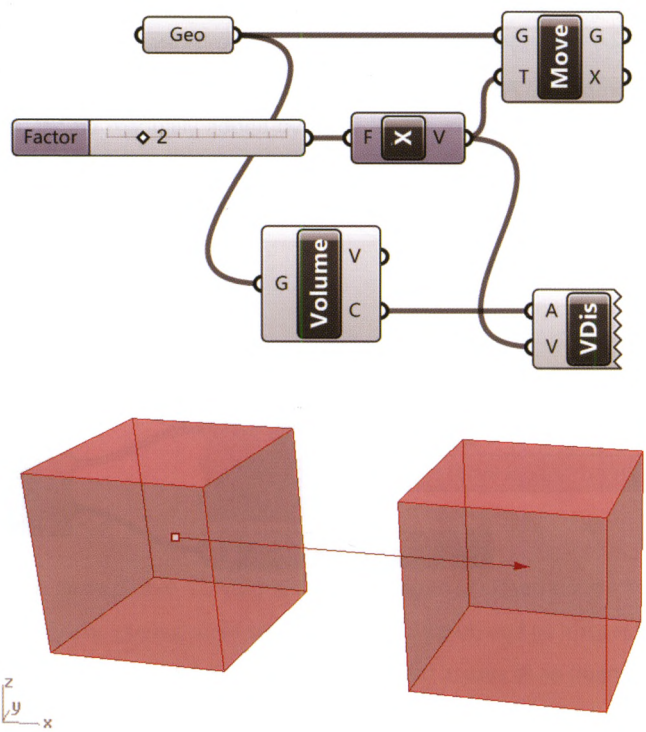
The index number of each point and the points' order can be displayed using the component *Point List* (Display > Vector). Display components do not have an output section as they are used for visualization. The component *Point List* allows users to set the text-size by the S-input.



1.7.4 Moving an object / vectors

When manipulating objects using a traditional modeling software, many mathematical/geometrical operations are blindly performed. For example, when vertically moving an object, the movement is performed according to a specific vector. **Vectors** are a geometric entities defined by **direction, sense and length (or magnitude)**.

The component *Move* (Transform > Euclidian), moves geometries according to a specified vector. Vectors can be defined by several components (Vector > Vector). For instance, vectors can be defined by three component parts using the component *Vector XYZ* (Vector > Vector) or by specifying a default unit vector according to an axis: *Unit X*, *Unit Y* or *Unit Z* (*Vector* > *Vector*). The following example uses the *Move* component to translate an input geometry (G) along a vector (T). The vector is defined through a *Unit X* component which, by default, is a unit vector whose direction and sense is parallel to x-axis and length equal to 1. In order to amplify the vector (and consequently, the movement) it is enough to connect a *Number Slider* to the F-input of *Unit X*. According to the *construction history* (see 1.6) Grasshopper displays the original and the translated geometry. In order to display just the moved geometry, the *Geometry* component must be turned off.



By default vectors are not previewed, they can be displayed using a specific visualization component: *Vector Display* (Display > Vector). *Vector Display* requires a start point (A) and the vector to display (V).

1.7.5 Custom preview

The default red-preview of Grasshopper geometries can be modified through the *Document Preview Settings*. **Users can alternatively customize the preview of single components using Custom Preview (Display > Preview)**. *Custom Preview* relies on the component *Colour Swatch* (Params > Input). The color and transparency level can be set by left-clicking on the node to recall a context menu.