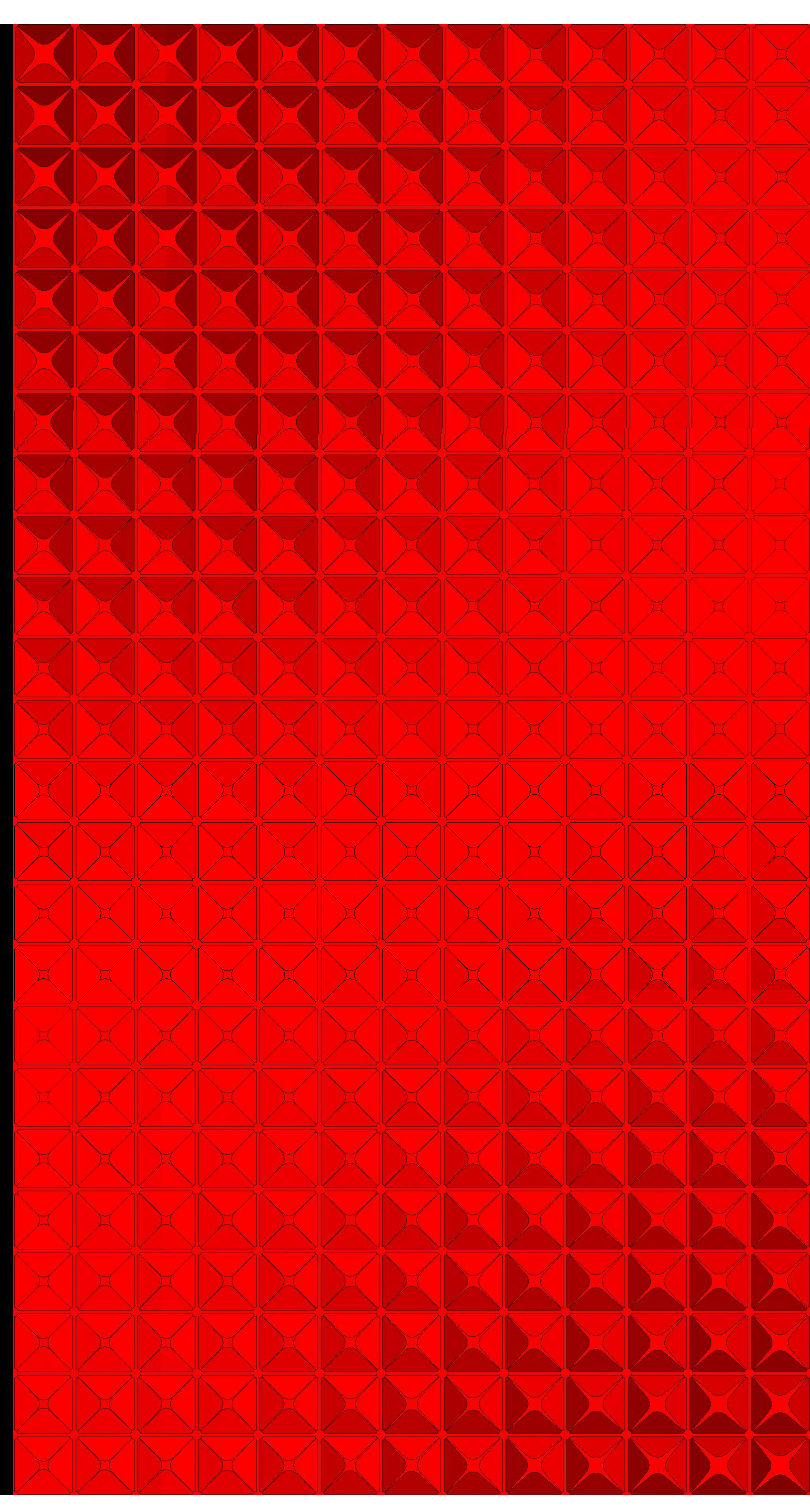
日本語版

GRASSHOPPER PRIMER

FOR VERSION 0.6.0007

BY ANDREW PAYNE & RAJAA ISSA



The Grasshopper Primer, Second Edition . for version 0.6.0007

Copyright 2009 All Rights Reserved

序文

Grasshopper の世界へようこそ。
Grasshopper Primer 2nd Edition は、Rajaa Issa 氏の多大なる貢献無しには完成し得ませんでした。
Rajaa Issa 氏は、Robert McNeel and Associate の開発者で、ArchCut や PanellingTools 等、いくつかの Rhino のプラグインの開発者でもあります。この版は 1 st Edition に比べ、さらに包括的なガイドとなっており、70 ページ以上がカスタマイズしたスクリプトコンポーネントの作成に割かれています。

この版のリリースにあたり、2 つ、大きなものがあります。最初は、まず、既に十分に堅牢であった、全バージョンに加え、Grasshopper の 0.6.0007 というさらに大きなアップグレードバージョンを取り扱う事。既存のユーザーは、内容に微妙な変更（データの格納については、かなり変更があります。）があることが見つかると思います。
またいくつかの前バージョンの Grasshopper 定義ファイルは、古かったり、使用出来ないものもあります。本書が、既存の読者や、新しい読書にとってもソフトウエアの変更に対して手助けになると良いのですが。
もう一つは、FLUX：Architecture in a Parametric Landscape（パラメトリックランドスケープにおける建築）というカリフォルニア美術大学（California College of the Arts.）で開かれたカンファレンスと内容がオーバーラップすることです。このイベントは、パラメトリックモデリング、デジタルファブリケーションやスクリプティングといったデザイン技術の変遷に関連したコンテンポラリーアーキテクチャとデザインを探求するものです。特に、このイベントは、パラメトリックソフトウエアシステムに貢献する展示会や一連のワークショップによって構成されるでしょう。
私にとって、Rajaa Issa 氏、Gil Akos 氏がアドバンス・Grasshopper・モデリングと、VB.Net　スクリプティング・ワークショップを行う傍ら、Grasshopper モデリングの導入に関して教える事が出来たことは非常に名誉な事でした。

本書には、さらに多くの情報が盛り込まれています。このプラグインを習得しようとする皆様にとって、良い情報源であり続けることが出来ればと思います。
しかしながら、このソフトウエアの最も大きな財産の一つはユーザーであるあなただということです。
何故なら、多くの人々がパラメトリックモデリングを探求し、理解するようになると、それは全てのグループにとって助けとなります。
私は、本書を読むそれぞれの人々に、オンラインコミュニティーに参加し、あなたの質問をフォーラムに投げるよう奨めています。誰かが、あなたの問題の解決を共有してくれるでしょう。
さらに Grasshopper を知るには是非、http://www.grashopper3d.com　を訪れてください。

謝意とともに、幸運をお祈りします。

Andrew Payne
LIFT architects
www.liftarchitects.com

Rajaa Issa
Robert McNeel and Associates
http://www.rhino3d.com/

翻訳者より一言

Andy Payne 氏の序文は GRASSHOPPER PRIMER 第 2 版が公開されたときのままのものを掲載しております。
本書、日本語版の GRASSHOPPER PRIMER は 2010 年 4 月に翻訳を終了していますが、この時点での最新の安定した Grasshopper のバージョンは、“0.6.0059”で、本書の内容と若干、異なるところがあります。
中には、若干、内容が古いものもありますが（特に最初の RhinoWiki を元に引用したものの翻訳部）、十分、内容が伝わると判断されるものは若干の注釈を加え、そのまま使用しています。

第 1 版と第 2 版との主な違いは、第 2 版は Andy Payne 氏と Robert McNeel and Associates 社の Rajaa Issa 氏の共著となっており、第 1 版の内容に加え、スクリプト、Visual Basic、Rhino の開発ツールキットにも言及しています。
日本語訳にあたり、内容を伝えやすくするため、原文とは若干、異なる箇所があるかもしれません。
開発に関わる方々には原文のままでも理解は難しくないと思われますので、本書に加え原文の、Grasshopper Primer 第 2 版も参考にしてください。

また、Andy Payne 氏の序文に書かれているように、世界的なオンラインコミュニティーである Grasshopper　GENERATING MODELING FOR RHIONO　に是非、参加され、世界中の参加者と情報共有をされ、情報発信をされてはいかがでしょうか。
http://www.grashopper3d.com

2010 年 4 月
　株式会社アプリクラフト
　取締役　Rhino エバンジェリスト　中島　淳雄

目次

序文
目次

1	Getting Started	1
2	インターフェース	2
3	Grasshopper オブジェクト	11
4	Persistent Data Management（永続性データマネージメント）	15
5	Volatile Data Inheritance（揮発性データの継承）	18
6	Data Stream Matching （データストリームマッチング）	23
7	スカラーコンポーネント	27
7.1	オペレーター	27
7.2	Range vs. Series vs. Interval	31
7.3	Functions & Booleans （関数と論理値）	33
7.4	Functions & Numeric Data（関数と数値データ）	35
7.5	Trigonometric Curves（三角関数曲線）	38
8	The Garden of Forking Paths（階層構造について）	42
8.1	Lists & Data Management（リストとデータ管理）	46
8.2	Weaving Data (データの振り分け)	49
8.3	Shifting Data (データのシフト)	52
8.4	Excel へのデータ出力	55
9	ベクトルの基礎	58
9.1	Point/Vector Manipulation(点・ベクトルの操作)	60
9.2	Using Vector/Scalar Mathematics with Point Attractors (Scaling Circles)	61
9.3	Using Vector/Scalar Mathematics with Point Attractors (Scaling Boxes)	66
10	カーブタイプ	74
10.1	カーブの分析	80
11	サーフェスタイプ	82
11.1	サーフェスの結合	84
11.2	Paneling Tools	88
11.3	Surface Diagrid	93
11.4	Uneven Surface Diagrid	98
12	Scripting とは	101
13	Scripting のインターフェース	102
13.1	Script コンポーネント	102
13.2	入力パラメーター	102
13.3	出力パラメーター	105
13.4	出力ウインドゥとデバッグ情報	105
13.5	Script コンポーネントの内部	107

14	Visual Basic DotNET	**110**
14.1	イントロダクション	**110**
14.2	コメント	**110**
14.3	変数	**100**
14.4	配列とリスト	**111**
14.5	Operators（オペレーター・演算子）	**113**
14.6	Conditional Statements （制御文）	**114**
14.7	ループ処理	**114**
14.8	ループのネスト	**117**
14.9	Sub プロシージャと Function プロシージャ	**119**
14.10	再帰処理	**122**
14.11	Processing Lists in Grasshopper（Grasshopper におけるリスト処理）	**125**
14.12	Processing Trees in Grasshopper（Grasshopper におけるツリー構造の処理	**127**
14.13	File I/O （ファイルの入出力）	**129**
15	Rhino .NET SDK	**131**
15.1	概要	**131**
15.2	NURBS の理解	**131**
15.3	OpenNURBS オブジェクトの階層構造	**135**
15.4	クラスの構造	**137**
15.5	定数インスタンスと定数ではないインスタンス	**138**
15.6	点とベクトル	**138**
15.7	OnNurbsCurve	**140**
15.8	OnCurve から派生しない Curve クラス	**145**
15.9	OnNurbsSurface	**147**
15.10	OnSurface から派生しない Surface クラス	**152**
15.11	OnBrepクラス	**153**
15.12	ジオメトリーの変換	**162**
15.13	グローバルユーティリティーファンクション	**164**
16	Help	**171**

1 *Getting Started*

Grasshopper のインストール

Grasshopper プラグインをダウンロードするには、

http://www.grasshopper3d.com/

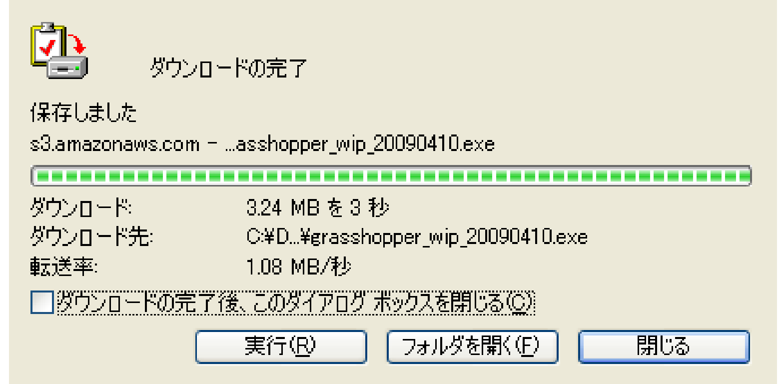
にアクセスしてください。Download　ページから、最新の Grasshopper がダウンロード出来ます。

2010/4/20 現在、安定したビルドとして Grasshopper　0.6.0059 がダウンロード出来ます。

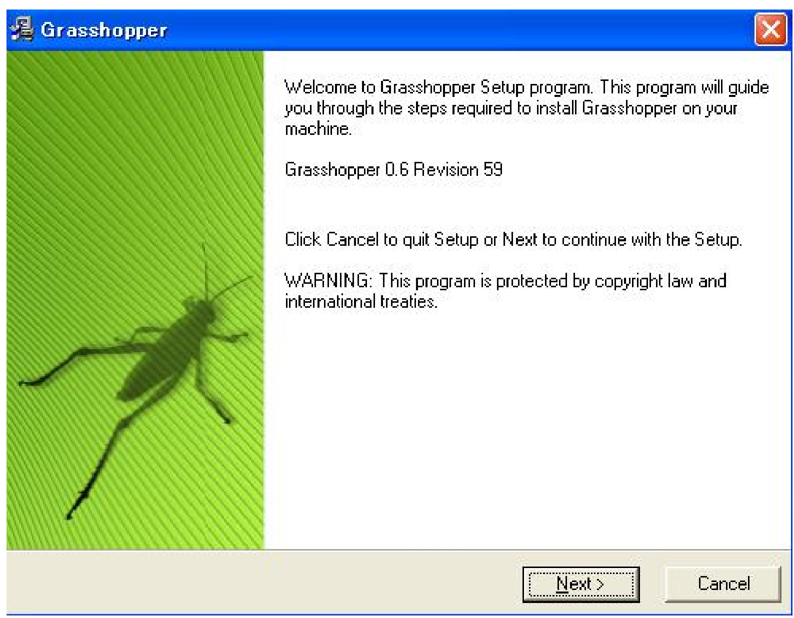
（注：本書は、0.6.0007 を元にして書かれています。最新バージョンと内容が異なる事があります。）

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

から、メールアドレスを入力し、インストーラーをダウンロードしてください。



ダウンロード終了後、インストーラーをダブルクリックすると、Grasshopper のインストールダイアログが立ち上がりますので、指示に従って、インストールを実行してください。

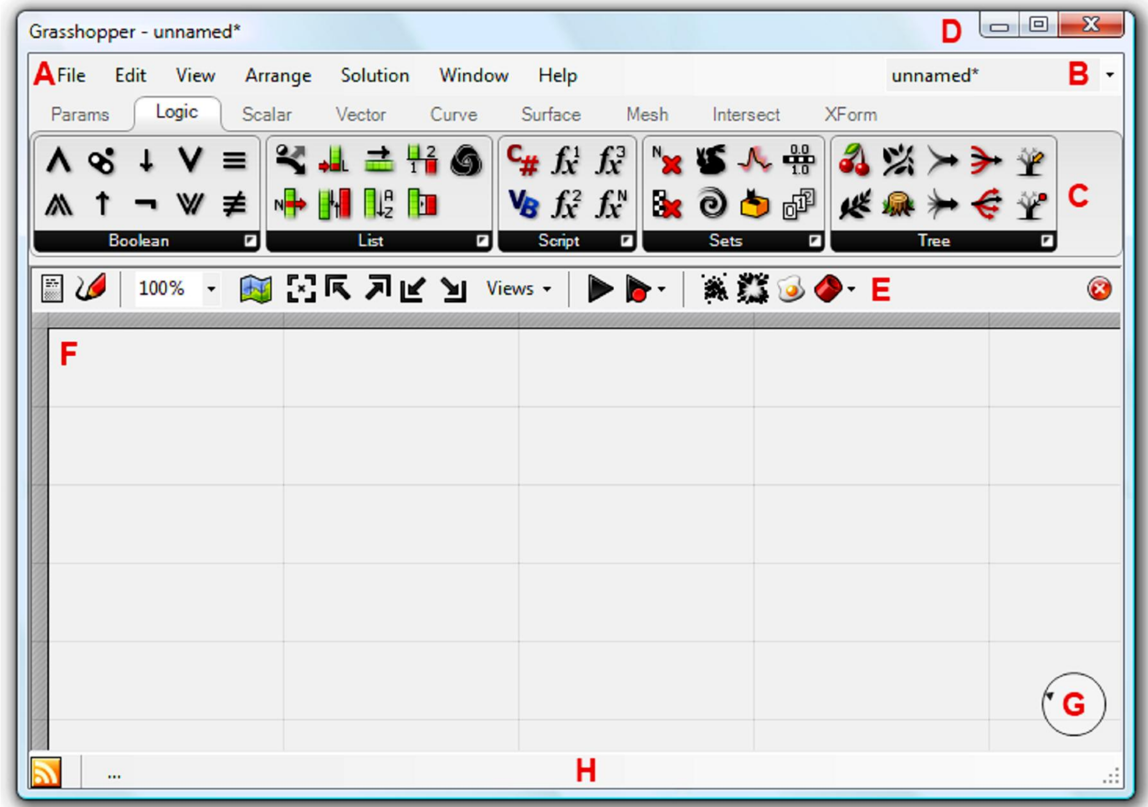


2 インターフェース

メインダイアログ

一度、プラグインがロードされると、**Rhino** のコマンドプロンプトエリアにおいて、”Grasshopper”とキー入力することにより、Grasshopper のメインウィンドウが立ち上がります。

（注：インターフェースは、0.5.0099 を元にして書かれております。最新バージョンと内容が異なる事があります。）



インターフェースは様々なエレメントを含みますが、Rhino ユーザーには分かり易いインターフェースになっています。

A. メインメニューバー

メニューは、一般的な Windows メニューに似ています。違いがあるのはメニュー右上、B の場所にあるファイルブラウザーコントロールです。リストボックスを選択して、異なるロード済のファイルを速やかに選択することが出来ます。

（Grasshopper では、定義ファイルö.ghxöというものを作成します。）

ショートカットはアクティブなウィンドウで作用されますので、使用の際には気をつけてください。このことは、Rhino や Grasshopper プラグイン、又は他の Rhino におけるウィンドウについて同じことが言えます。現時点では、アンドゥ機能はサポートされていませんので、”Ctrl-X”，“Ctrl-S” や”Del” **等のショートカットの使用には気をつけてください。**

B.ファイルブラウザーコントロール

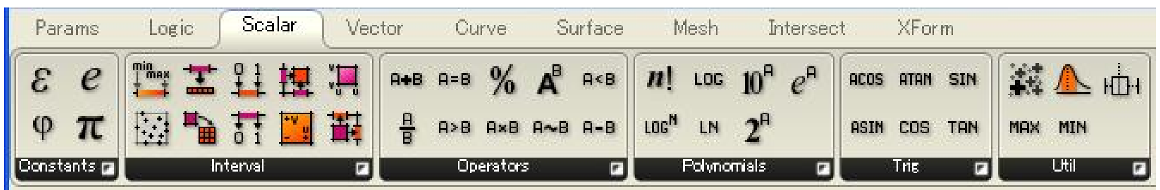
A で記述した通り、リストボックスでロード済の、Grasshopper 定義ファイルを切り替えます。

注；以降、Grasshopper 定義ファイルを、GH 定義と省略して表現する場合があります。

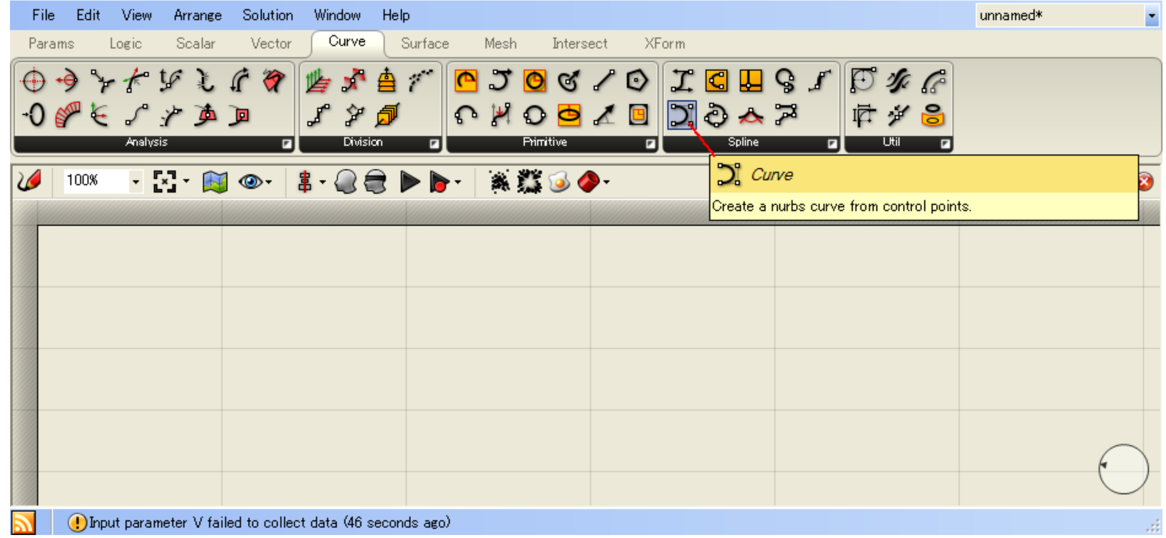
C.コンポーネントパネル

このエリアは全てのコンポーネントの分類を表示しています。
このエリアは全てのコンポーネントをあらわします。全てのコンポーネントはあるカテゴリーに属し（例えば、+Params+カテゴリーは全てのプリミティブなデータタイプ・コンポーネントや、全てのツールに関連+Curves+コンポーネントを表示します。）全ての分類はユニークなツールバーパネルとして使用できます。
ツールバーの高さ・幅は調整し、分類毎のボタン表示を少なくすることも出来ます。

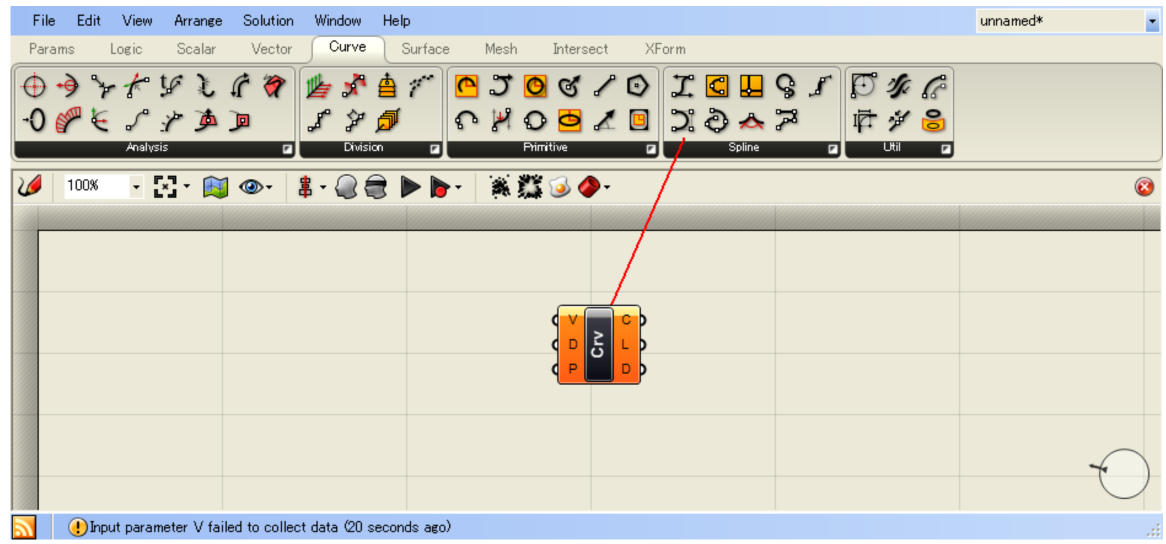
ツールバーパネルは、+Params+、+Logic+、+Scalar+、+Vector+、+Curve+、+Surface+、+Mesh+、+Intersect+、+Xform+のカテゴリーに別れ、それぞれのカテゴリーは、またいくつかのサブカテゴリーに分類されます。その中で Grasshopper オブジェクトへのアクセスがアイコン化され配置されており、全ての Grasshopper オブジェクトにアクセスすることが出来ます。



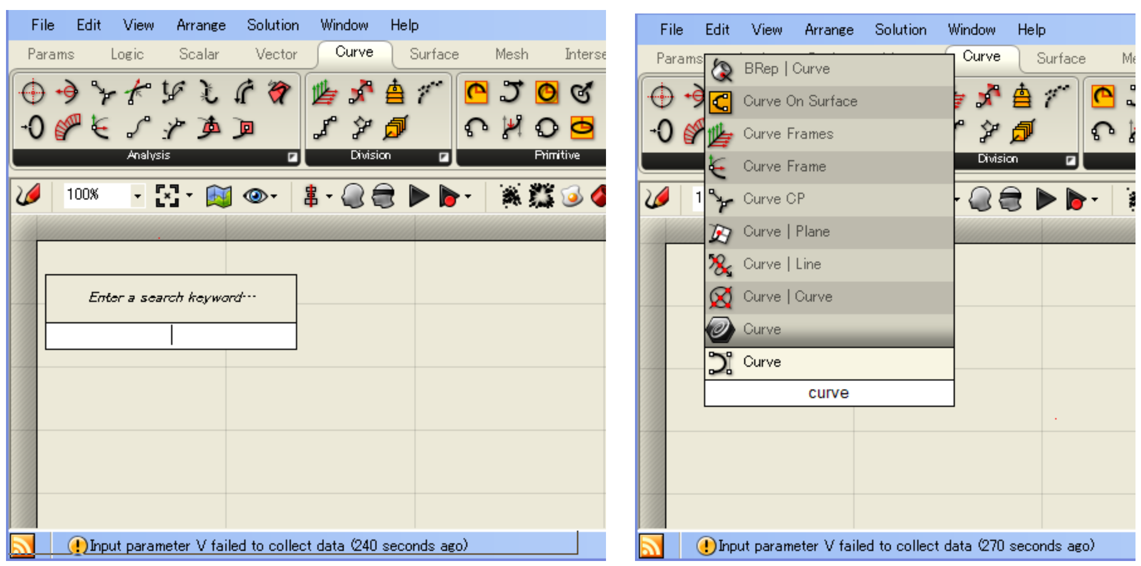
アイコン上に、マウスカーソルを移動すると、簡単な機能が表示されます。



アイコンをクリックし、キャンバス上に移動し、再度、適当な場所でクリックするか、アイコンをクリックしたままキャンバス上にドラッグすることによってコンポーネントを配置することが出来ます。



キャンバス上の任意の位置で、マウスをダブルクリックすると、キーワードによる検索ダイアログが現れ、目的のコンポーネントを探すことが出来ます。

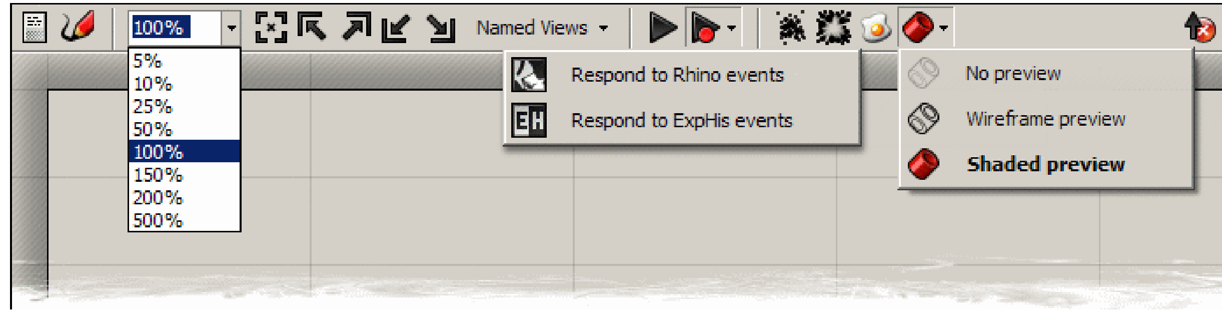


D.ウィンドウタイトルバー

編集ウィンドウのタイトルバー部：D は他の Microsoft のウィンドウと異なる挙動をします。Grasshopper ウィンドウが、最大化または、最小化されていない状態で、+タイトルバーの一部+をダブルクリックするとウィンドウはタイトルバーだけになります。もう一度、ダブルクリックするとまたもとのウィンドウが現れます。
また、ウィンドウを右上の+閉じる+ボタンで閉じると Grasshopper ジオメトリーのプレビューは消えますが Grasshopper の定義ファイルそのものが終了するわけではありません。
次に、また+Grasshopper+コマンドで、このウィンドウを開いた場合、以前作業していた定義ファイルは、最後に閉じた状態で表示されます。

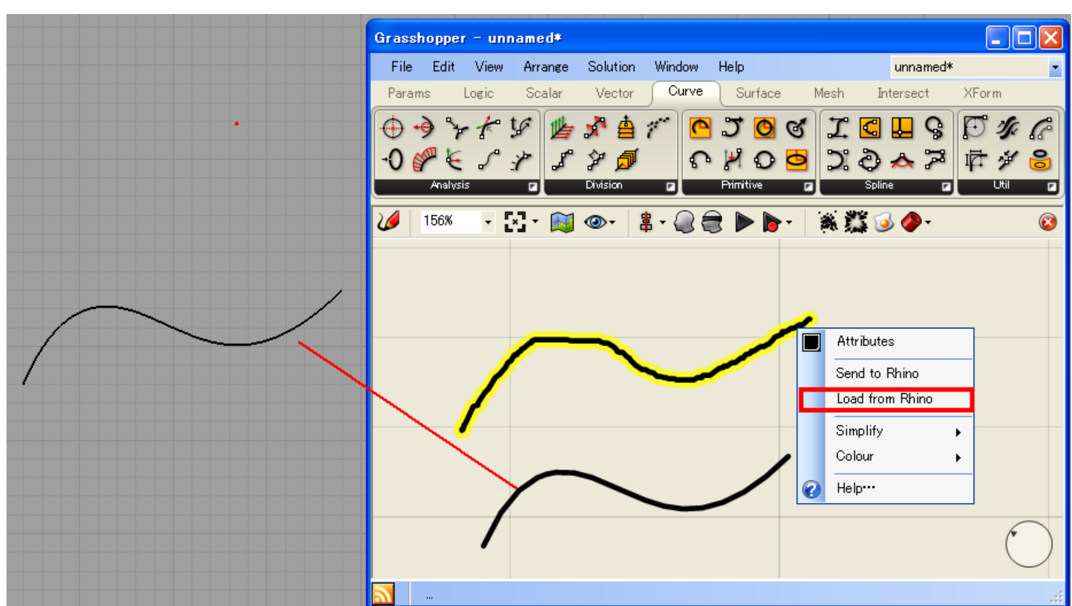
E.キャンバスツールバー

キャンバスツールバー：E によって、頻繁に使用する機能に容易にアクセスすることが出来ます。キャンバスツールバーの全ての機能は、メインメニューから実行することが出来ます。キャンバスツールは非表示にすることが出来ます。もし再度、表示するときは、メインメニューのöViewöメニューから再表示します。



キャンバスツールは左から順に次のような機能を持ちます。

1. スケッチツール
   スケッチツールは、Photoshop や Windows の Paint の鉛筆ツールのように使います。
   アイコンをクリックすると、ラインウエイト、色等の指定が出来ます。
   このツールは、直線や決まった図形を描くには適してはいません。この問題を解決するためには、Rhino の 2D の図形を使用します。
   まず、Rhino で 2D 図形をドローイングしておき、次にスケッチツールで適当に図形を描き、その図形を右クリックします。いくつかオプションが現れますが、そのうちの、+Load from Rhino+というオプションを選択すると Rhino の描画オブジェクト選択することによってその形状を反映させることが出来ます。



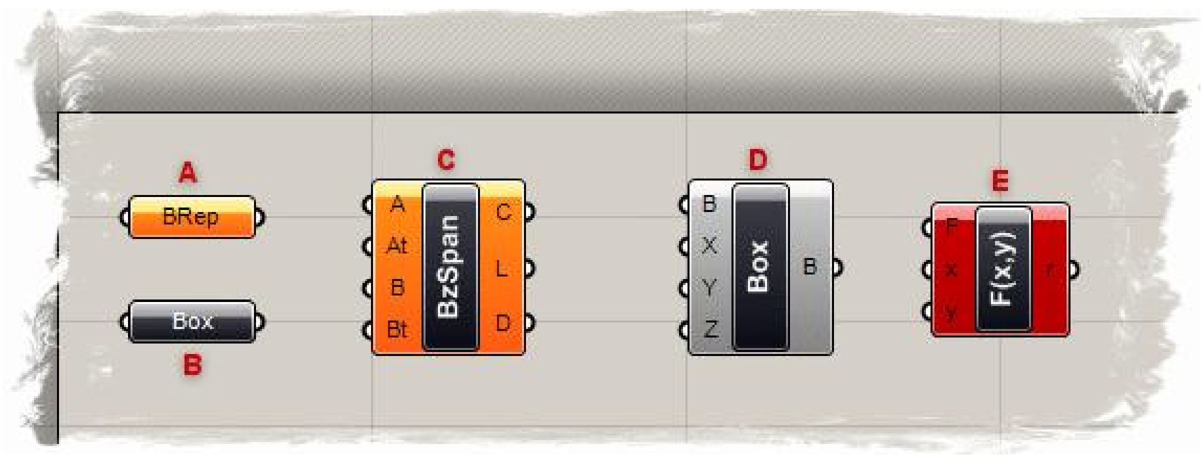
2. ズーム入力
3. ズーム範囲
   キャンバス上の全ての領域表示を行います。またリストボックスから、上左、上右、下左、下右の領域の表示指定することも出来ます。
4. ナビゲーションマップ
   キャンバスの領域をドラッグしながらナビゲーションすることが出来ます。
5. 名前つきのビュー
   ビューに名前をつけ、保存、呼び出しをすることが出来ます。
6. 整列
   選択したオブジェクトを、整列することが出来ます。
   （Version 0.6.0059 ではなくなっています。 2010/1/12 現在）
7. Enable Preview Selection
   選択したコンポーネントの結果を表示します。
8. Disable Preview Selection
   選択したコンポーネントの結果を非表示にします。
9. Rebuild ソリューション
   10.の Rebuild イベントがオフになっているとき、ここをクリックすると Grasshopper で編集した結果を表示します。
10. Rebuild イベント
   初期値では、Grasshopper は Rhino または Grasshopper で編集した結果を直ぐに反映するようになっています。
11. Cluster compactor

選択したオブジェクトを一つのクラスターオブジェクトにまとめます。
（Version 0.6.0059　ではなくなっています。　　2010/1/12 現在）

12. Cluster exploder 選択したクラスターオブジェクトを元のオブジェクトに展開します。
（Version 0.6.0059　ではなくなっています。　　2010/1/12 現在）
13. Bake tool
選択したコンポーネントを Rhino オブジェクトに置き換えます。
14. Preview セッティング
Grasshopper のジオメトリーは初期状態では、プレビュー出来るように設定されています。オブジェクト毎にプレビューを無効にすることも出来ますし全てのオブジェクトについて、プレビューを無効にすることも出来ます。
シェーディングのプレビューをオフにするだけでも格段に早くなります。
15. 非表示ボタン
非表示ボタンをクリックすると、キャンバスツールバーは見えなくなります。
再び、表示するためには、View メニューで+Canvas tool bar+にチェックをいれます。

F.キャンバス

この領域が、実際に編集する場所です。キャンバスは、GH 定義を行うオブジェクトとユーザーインターフェースウイジェット **G.**を持ちます。
キャンバス上のオブジェクトは、通常、色情報でそれらの状態を表します。



A)　パラメーターで内部に警告を含むもので、オレンジ色で表示されます。ほとんどのパラメーターはキャンバスに配置されたときはデータを定義されていないのでこの状態になります。

B)　パラメーターで+警告+も+エラー+も持たない状態です。

C)　コンポーネントは入力と出力パラメーターを持ちますので、常に他のオブジェクトを伴います。このコンポーネントは少なくとも１つの+警告+を含みます。
‰警告+や+エラー+は、オブジェクトのコンテクストメニュー（後述）で見つけます。

D)　コンポーネントで+警告+も+エラー+も持たない状態です。

E)　コンポーネントで少なくとも１つの+警告+を持つものです。
‰エラー+は、そのコンポーネント自体が持つ場合、またはそれの入出力パラメーターから発生する場合もあります。

次の章でコンポーネントの構造について学びます。
全てのオブジェクトは、選択状態において緑色で表示されます。

G.UIWidgets（ユーザーインターフェース　ウイジェット）

現時点では、コンパスという UI　Widget のみ存在し、キャンバスの右下方に表示されます。+View+メニューで表示・非表示出来ます。

H.ステータスバー

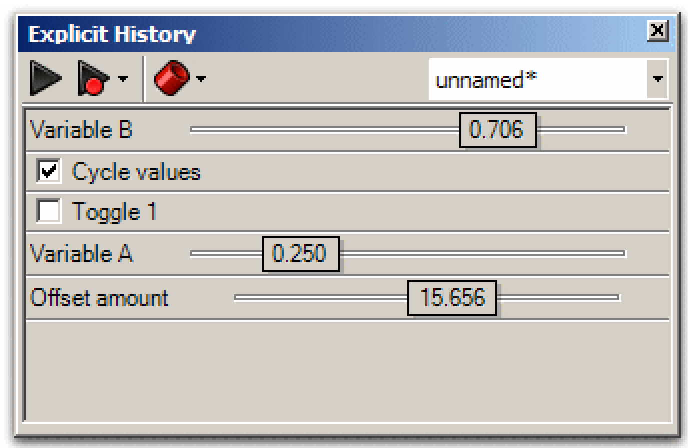
ステータスバーは、選択したもののフィードバックと、このプラグインで起きた主なイベントを表示します。（この機能はまだ組み込まれていません。）
スタータスバーのö省略öシンボルを右クリックすることで、最近、起きたイベントを見ることが出来ます。
左下方の、オレンジ色のアイコンをクリックすると、Grasshopper のユーザーグループウェブサイトにリンクされ、そこでの最近のスレッドを表示します。
スレッドのどれかを選択すると、ユーザーグループのメンバーが投稿したディスカッションに飛びます。

http://www.grasshopper3d.com/

が Grasshopper のユーザーグループのウェブサイトです。

リモートコントロールパネル：

GH 定義ウィンドウは極めて大きいので、常に開いた状態でおきたくない場合もあります。もちろん最小化または、閉じてしまうことも出来ます。数値を編集することは出来ません。リモートンコントロールパネルを使用可能にすることによって現在アクティブな GH 定義において、数値入力に関する最小限のインターフェースを表示することが出来ます。このドッキング可能なダイアログは、現時点ではスライダーとブーリアンスイッチのみ表示します。（将来、拡張されることも考えられます。）

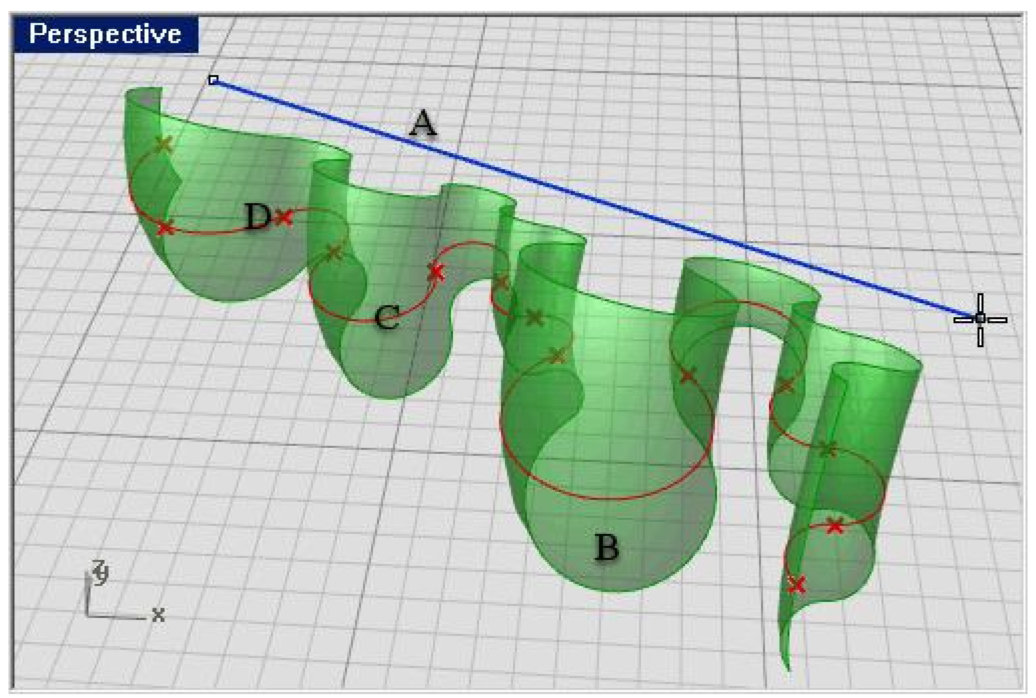


リモートコントロールパネルは、基本的なプレビューのコントロールも出来ます。
このパネルをアクティブにするには+View+メニューの+Remote Control Panel+をチェックするか、Rhino のコマンドプロンプトにおいて、+GrasshopperPanel+とタイプ入力することにより、表示出来ます。

（Version 0.6.0059　では動作しません。　　2010/4/20 現在）

ビューポートのプレビューフィードバック:

キャンバスに配置されたコンポーネントに属するジオメトリーは、Rhino のビューポート上にプレビューされます。



A) 青の部分は現在、マウスで選択したジオメトリーを表します。

B) 緑色で表示されたジオメトリーは、Grasshopper キャンバス上にある選択状態にあるコンポーネントに属するジオメトリーを表します。

C) 赤で表示されたジオメトリーはキャンバス上にある選択状態にないコンポーネントに属するジオメトリーを表します。

D) Grasshopper のポイントのジオメトリーは、Rhino のポイントと区別がつくように、X 点で表示されます。

3 Grasshopper オブジェクト

Grasshopper 定義オブジェクト

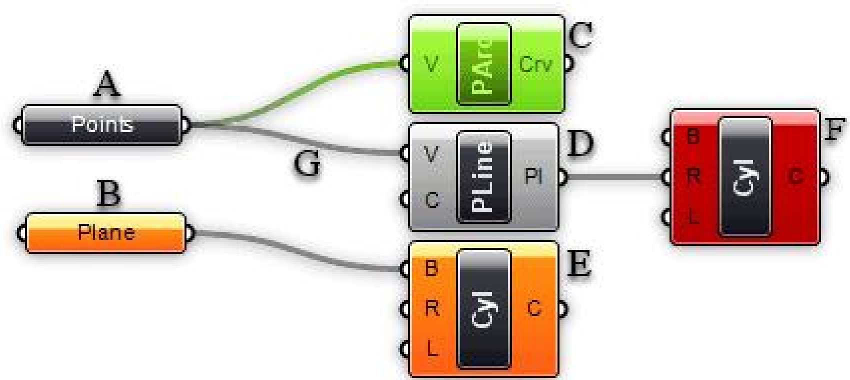
Grasshopper 定義は、多くの異なるオブジェクトから構成されます。
最初に次の 2 つのオブジェクトについて説明します。

- パラメーター（データ）
- コンポーネント（アクション）

注１；コンポーネントの分類
コンポーネントパネル上に配置されているアイコンから作成されるオブジェクトは全てコンポーネントと呼ばれるオブジェクトです。
コンポーネントは、現在、1)パラメーター、2)コンポーネント、3)クラスターコンポーネントの 3 種類があります。（今後、増えていく可能性があります。）
%Params+パネル上にあるアイコンから作成されるオブジェクトは全て、パラメーターと呼ばれるコンポーネントです。
コンポーネントはコンポーネント全体あるいは、以下に解説するデータに対するアクションを起こすものを示すものもコンポーネントと呼ばれます。
これ以降、アクションを起こすコンポーネントを明示する必要がある場合は、コンポーネント（アクション）と記述します。

注 2；Grasshopper 定義
キャンバス上に配置されたもの全体を、Grasshopper 定義（Grasshopper Definition　あるいは単に Definition）と呼びます。
Grasshopper 定義を保存すると、+.ghx+という拡張子を持った定義ファイルが作成されます。以降、Grasshopper 定義、GH 定義等と記述します。

パラメーターはデータを格納します。コンポーネント（アクション）は、データに対するアクションを持ちます。
ここでいうデータは、Rhino の図形オブジェクトや、ベクトル等の数値、論理値を意味します。
下記の図は、Grasshopper 定義において構成される典型的な例です。



A) A は内部にデータを持つパラメーターです。このオブジェクトの左側には、ワイヤによる結線がありませんが、これはこのパラメーターの内部のデータが、他のオブジェクトから継承されていないことを意味します。
パラメーターが含むデータに+エラー+や+警告+が無い場合、オブジェクトは、黒色で表示されます。

B) B は内部にデータを持たないパラメーターオブジェクトです。 GH 定義においてデータを集めないオブジェクトは疑いの対象になります。新たに追加された全てのパラメ

ーターは、初期状態ではオレンジで表示されます。パラメーターが他のオブジェクトのデータを、継承したり、新たにデータを定義されると表示色は黒に変わります。

C) C は選択状態にあるコンポーネントです。選択状態にあるオブジェクトは（パラメーターであれ、コンポーネントであれ）全て、緑色で表示されます。

D) D は通常のコンポーネント（アクション）で、黒で表示されています。

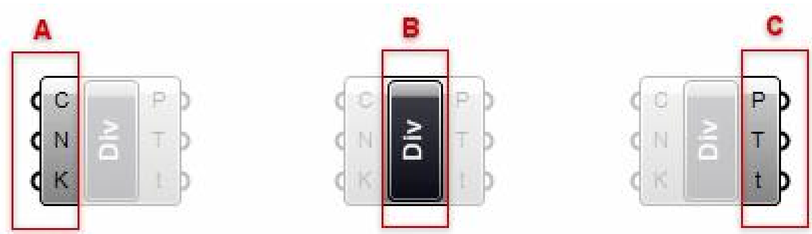
E) E はコンポーネント（アクション）で+警告+を持つ状態のものです。コンポーネント（アクション）は、いくつかの入力と出力パラメーターを持ち、どこが+警告+を生み出しているか一目で判断することは出来ません。また %警告+が複数に及ぶ場合もあります。
この問題を解決するためには、後述するコンテクストメニューを使用して原因をつきとめることになります。
注；+警告+は必ずしも、全て消す必要はありません。

F) F は+エラー+を持つコンポーネント（アクション）です。+エラー+も、+警告+同様、一目で原因を特定出来ませんので、コンテクストメニューを使用して原因を特定します。
注；コンポーネントが、同時に+警告+と +エラー+を持つ場合は、表示色は赤になります。

G) G はオブジェクト間のコネクションでワイヤによる結線で表示されます。
コネクションは、常にオブジェクトの出力と入力に結線されます。
コネクションの数には制限はありませんが、再帰的なコネクションは禁止されています。
再帰的なコネクションが検出された場合は、その最初のコンポーネント又はパラメーターが再帰的コネクションである+エラーメッセージ+が出されます。
詳細は後述の、データの継承を参照してください。

コンポーネント（アクション） のパーツ

コンポーネント（アクション）は、通常、アクションを実行するための+データ+を必要とし、その結果を出力します。これゆえ、多くの場合コンポーネントは複数のパラメーターのネストとなり、入力と出力パラメーターは個別に取り扱われます。入力パラメーターは左、出力パラメーターは右となります。

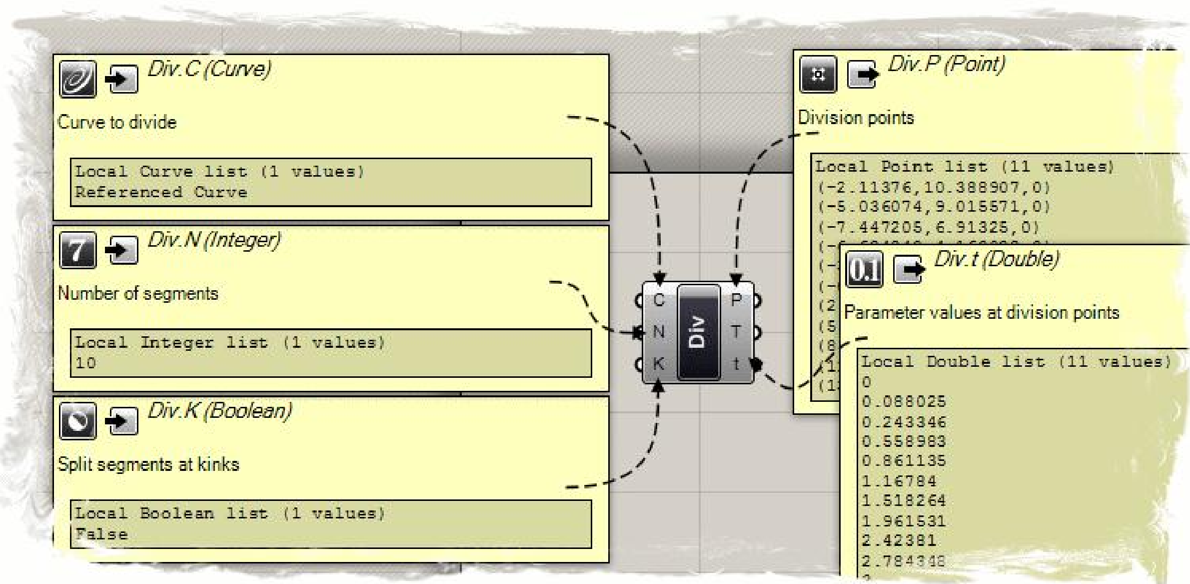


A) A は+Division+コンポーネントの初期状態における入力パラメーター部です。各パラメーター名は（C、N、K 等）初期値では極めて短い表現がされていますが、これらのパラメーター名を変更することが出来ます。

B) B はコンポーネント領域です。通常そのコンポーネントの名前が与えられています。

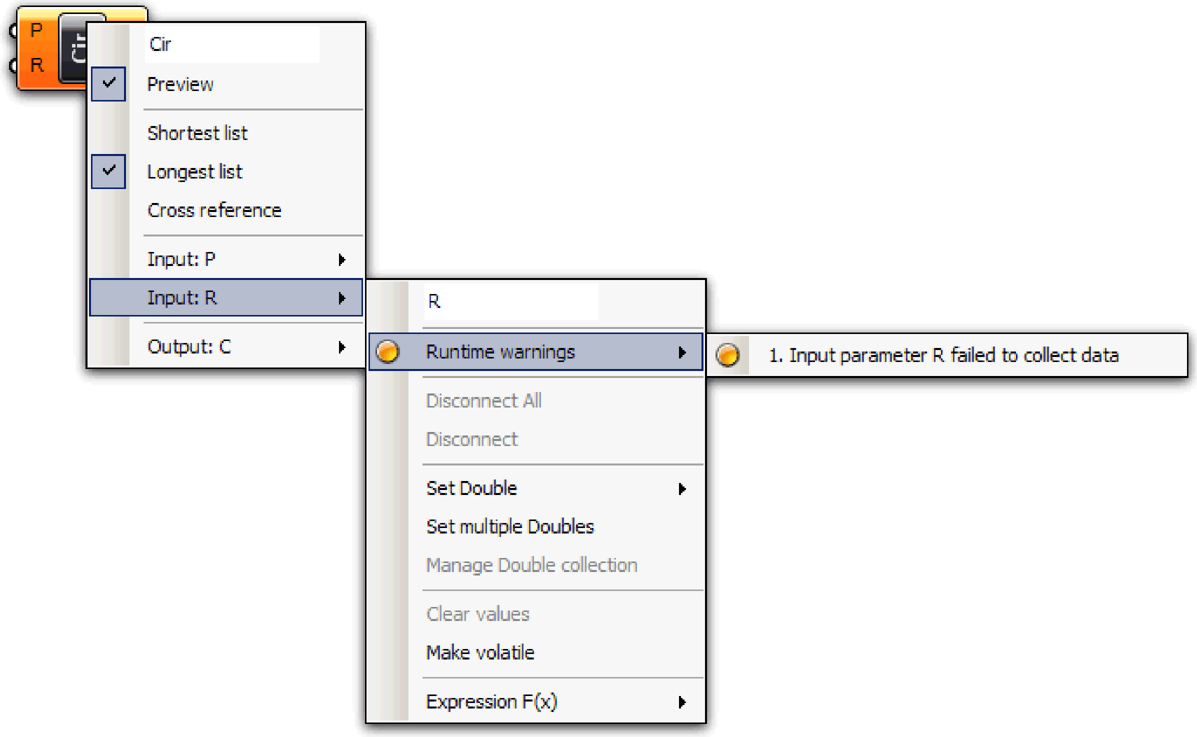
C) C は+Division+コンポーネントの初期状態における出力パラメーター部です。

マウスカーソルをコンポーネントのそれぞれのパートの上に持っていくと、それぞれの+ツールティップス+が表示されます。+ツールティップス+からそれぞれのパラメーターのタイプとデータを知ることが出来ます。



コンテクストポップアップメニュー

キャンバス上の全てのオブジェクトは、それぞれ+コンテクストメニュー+を持ち、それぞれのコンポーネントの機能を知ることが出来ます。
コンポーネントの場合、+コンテクストメニュー+はサブオブジェクトが持つメニューまで表示するので、少し扱いにくいかもしれません。
例えば、もし、コンポーネントがオレンジで表示されていれば、それは+警告+を表しますが、またはいくつかのパラメーターがコンポーネントによって影響され、+警告+を発していることが考えられます。原因を突き止めるためには、コンテクストメニューを使用するのが良いでしょう。



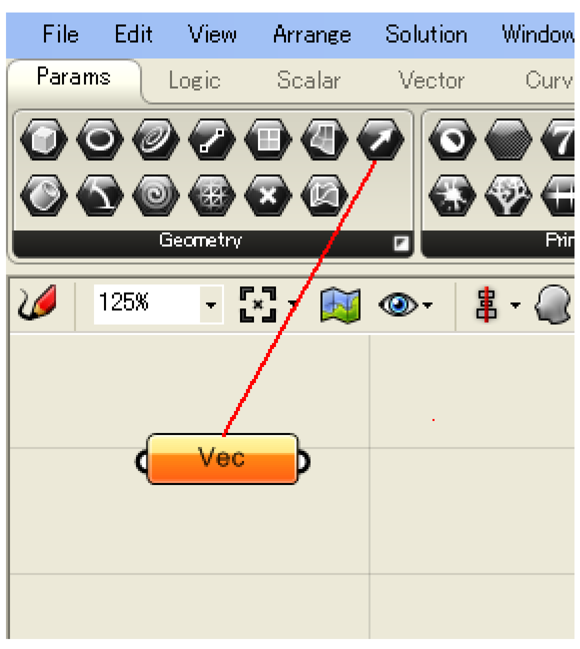
この例では、メインコンポーネントメニューの+R+の入力パラメーターがカスケード表示されている状態を示しています。
まずコンテクストメニューの一番上ですが、そのパラメーターの名前が表示されています。これは編集可能ですが、スクリーン上の領域を有効に利用するため初期値では短い名前が与えられています。（この例では、 %Cir+)
2 番目の+Preview+のフラグは、このオブジェクトによって定義・作成されたジオメトリが+Rhino のビューポート上で表示されるかどうか+をしめしています。
あまり重要でないコンポーネントの場合は、このフラグをオフにすることによって表示を早くすることが出来ます。（特にメッシュが含まれる場合は有効です。）
全てのパラメーターやコンポーネントが描画されるわけではありません。（例として数値）
このような場合、オブジェクトの場合、+Preview+フラグは表示されません。

この例では、öRöの入力パラメーターのところで、ö警告öが出ていることが分かります。

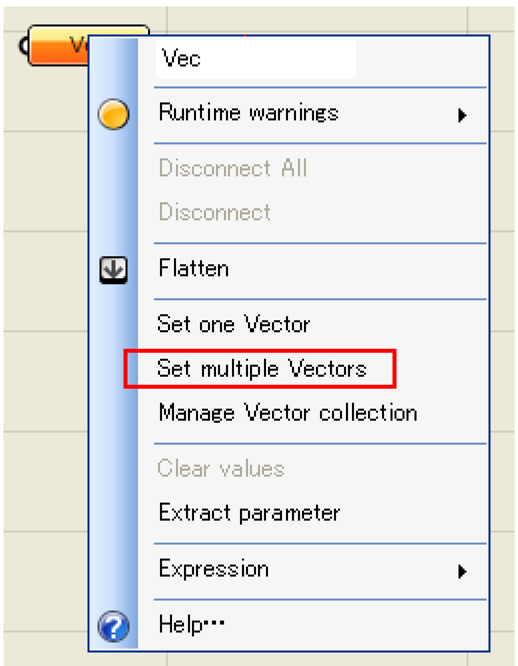
4 Persistent Data Management（永続性データマネージメント）

パラメーターは、情報を格納するためだけに使用されますが、多くのパラメーターは揮発性のデータ（Volatile data）と永続性を持つデータ（Persistent data）の 2 つの異なるタイプを格納できます。
揮発性のデータは、一つあるいは、複数のパラメーターソースから継承されたもので、新しいソリューションがスタートすると、失われるか再定義されます。
永続性のあるデータは、ユーザーによって指定されることで与えられます。
パラメーターがソースとなるオブジェクトに取り付けられたとき、永続性のあるデータは無視されますが、失われる訳ではありません。
（パーマネントな記録としても格納されず、ソースのセットとしても定義されない出力パラメーターは例外です。出力パラメーターは、所有されるコンポーネントに完全にコントロールされます。）

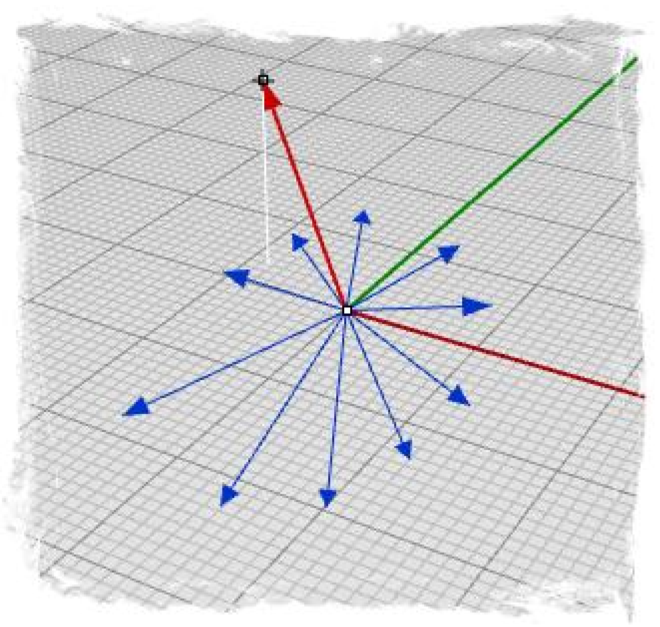
持続性のデータは、メニューからアクセスされ、パラメーターが異なるマネージャーを持つかどうかに依存します。
例えば+Vector+パラメーターは、メニューより、一つ又は複数の Vector データをセットすることが出来ます。
その前に、初期状態の+Vector+パラメーターがどのように挙動するかを見ることにします。
Params パネルから、Vector パラメーターのアイコンをキャンバス上にドラッグ＆ドロップします。



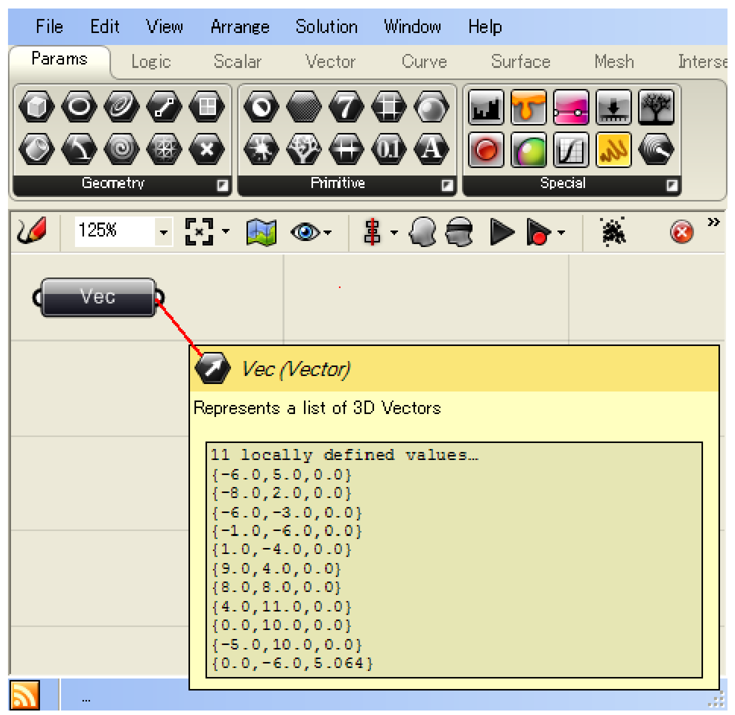
この状態では、+Vector+パラメーターはオレンジ色で表示され、+警告+の状態にあります。これは単に、パラメーターは何もデータを持っていないことを+警告+しているだけです。（永続性のあるデータも揮発性のデータも持ってはいません。）結果として GH 定義において何の影響も与えません。コンテクストメニューを見ると、Vector パラメーターが 1 つのベクトルを定義するか、複数のベクトルを定義することが出来るかを示しています。



いずれかをクリックすると、Grasshopper の Window が消え、Rhino のビューポートが現れ、3 次元ベクトルを選択するように指示されます。



ここで、3 次元ベクトルを指定し、+Enter+キーを押すと、それらのベクトルが永続性のあるデータとして記録されます。これによって、+Vector+パラメーターオブジェクトは+警告+を表すオレンジ色から、通常状態の黒色に変わります。
マウスカーソルを+Vector+パラメーターの出力のところに移動すると、指定したベクトル情報が確認出来ます。


File
Edit
View
Arrange
Solution
Window
Help
Params
Logic
Scalar
Vector
Curve
Surface
Mesh
Geometry
Primitive
Special
125%
Vec
Vec (Vector)
Represents a list of 3D Vectors
11 locally defined values…
{-6.0,5.0,0.0}
{-8.0,2.0,0.0}
{-6.0,-3.0,0.0}
{-1.0,-6.0,0.0}
{1.0,-4.0,0.0}
{9.0,4.0,0.0}
{8.0,8.0,0.0}
{4.0,11.0,0.0}
{0.0,10.0,0.0}
{-5.0,10.0,0.0}
{0.0,-6.0,5.064}

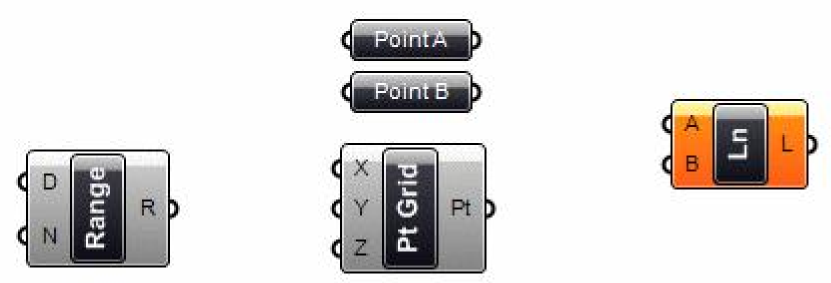
5 Volatile Data Inheritance（揮発性データの継承）

データの継承

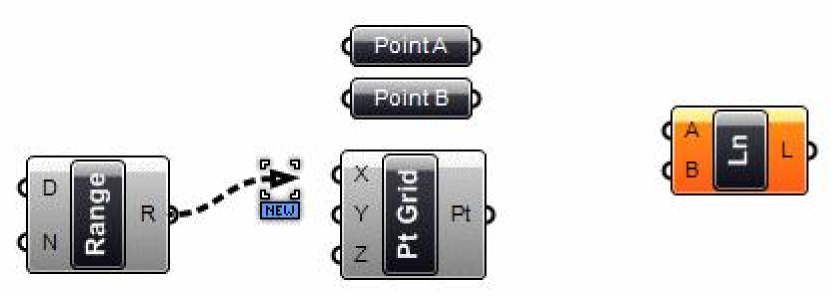
データはパラメーター（揮発性のものであれ、永続性のものであれ）に格納されます。データがパラメーターに永続性データのレコードセットとして格納されない場合は、何処からかデータを継承してくる必要があります。出力パラメーター以外の全てのパラメーターは、どこからデータを持ってくるか定義しますが、ほとんどのパラメーターは特定出来ません。浮動少数点のパラメーターを変換して整数のソースに割り当てることも出来ます。プラグインは、さまざまな変換スキームを定義しますが、もし変換プロセスが定義されないと、受け取り側のパラメーターは変換エラーを生じます。
例えば、もし+Point+データが必要なときに+Surface+データを与えたら、+Point+パラメーターはエラーを起こし、パラメーターは赤で表示されることになります。
もしパラメーターがコンポーネントに属するのであれば、赤の状態（エラー）も次のコンポーネント自体にエラーが無くても継承されていきます。

コネクションの管理

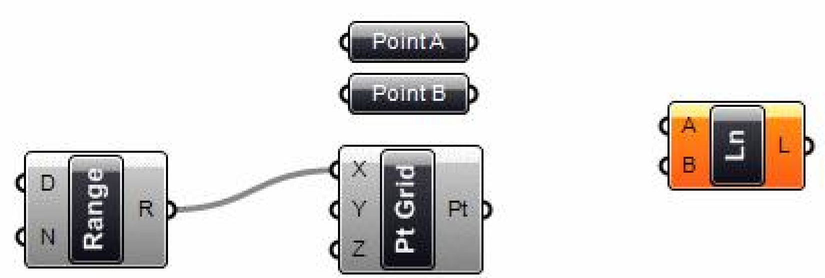
パラメーターは自身のデータソースを持ち、これらのパラメーターが持つデータにアクセスすることが出来ます
ここでは、3 つのコンポーネントオブジェクトと、２つのパラメーターオブジェクトの例で見ます。



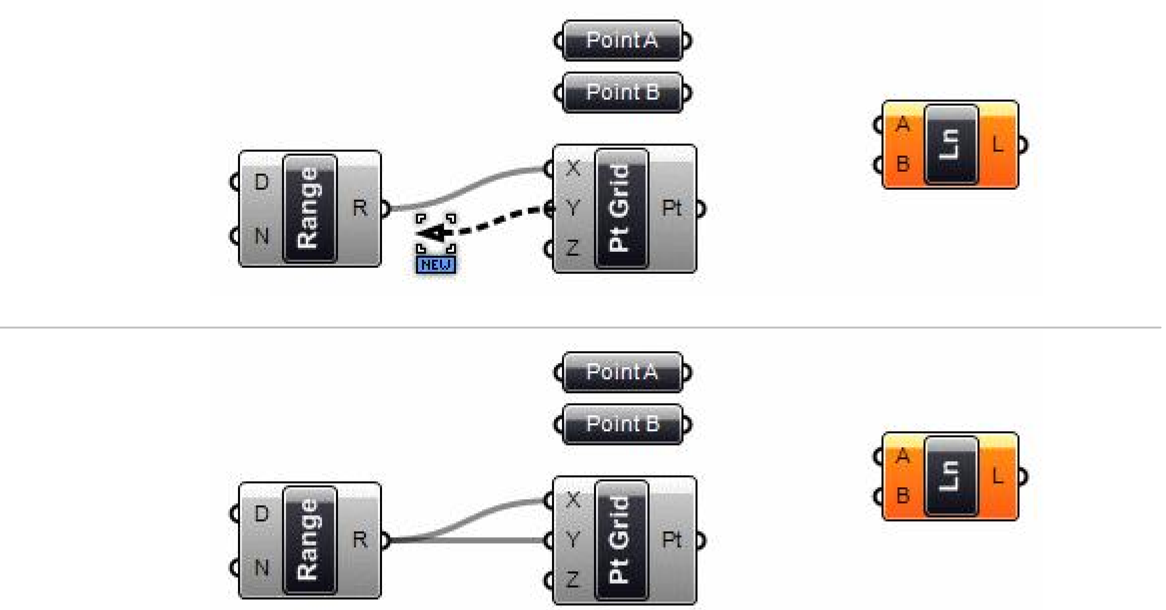
この状態では、全てのオブジェクトが結合されていません。まずこれらをつなげる必要があります。順番は関係ありませんが、左側から右側に向かっておこないましょう。パラメーターの小さなサークル部分（クールに+グリップ+と呼ばれることもあります。）の近くをマウスでドラッグすると結線ワイヤがマウスに連動して作成されます。



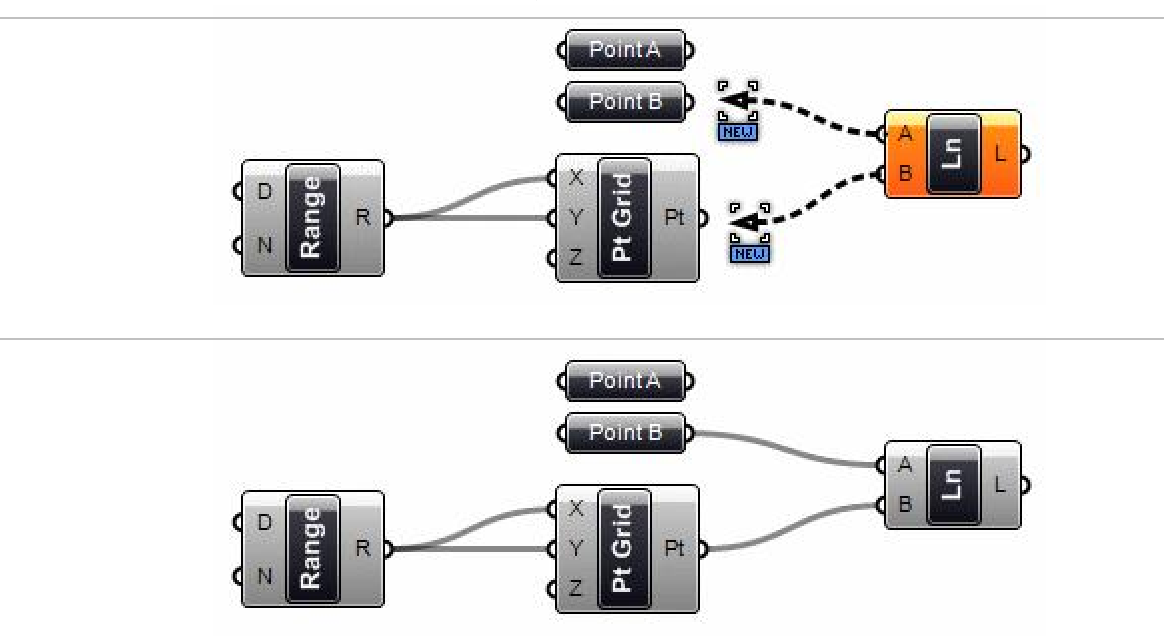
マウスの左クリックの状態で、他の接続可能なパラメーター付近に持っていくと結合状態になります。マウスボタンを離すと接続は終了です。



同様に+PtGrid+コンポーネントの+Y+パラメーターを、+Range+コンポーネントの+R+パラメーターに接続します。



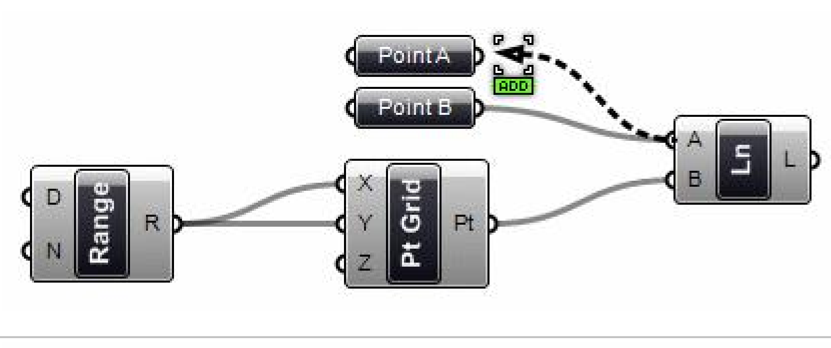
さらに+Line+コンポーネントの+A,+、+B,+パラメーターに下記のように接続します。

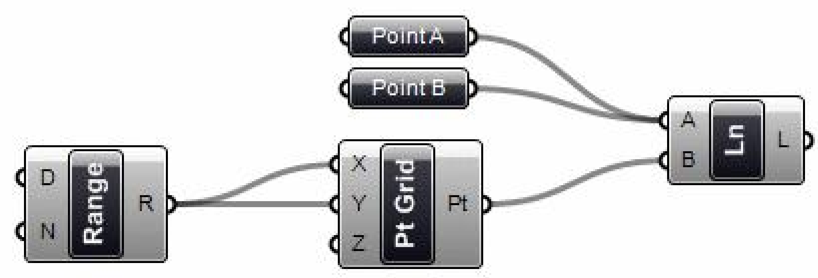


このようにして、接続が可能です。初期設定では、新しい接続を行うと古い接続が、消去されてしまいますので、注意が必要です。これは、ほとんどの場合、接続は一つであると想定され

ているからです。複数の結合を行う場合には、マウスドラッグの際に+Shift+キーを押して実行すると追加することが出来ます。

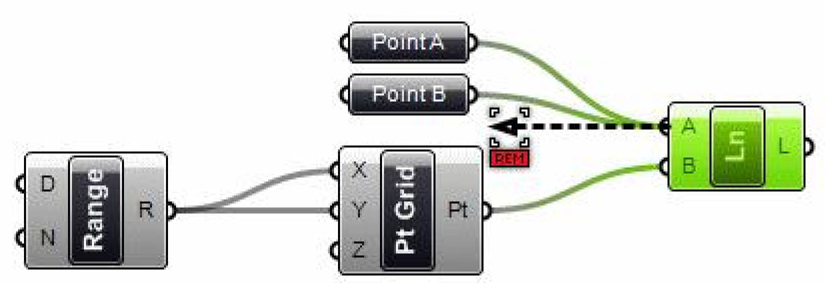
注；Version0.6 では、マウスのポインターが、新規接続は、黒の円で表示され、追加時は、緑色の円に色が変わります。

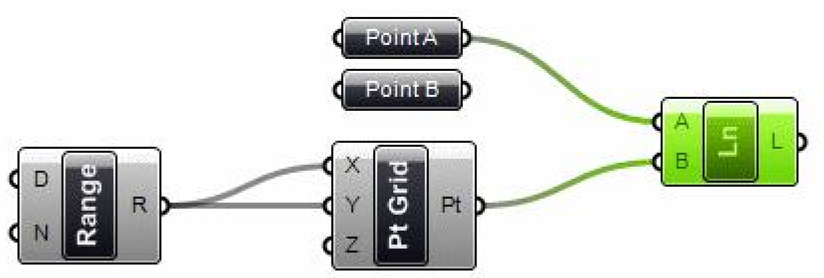




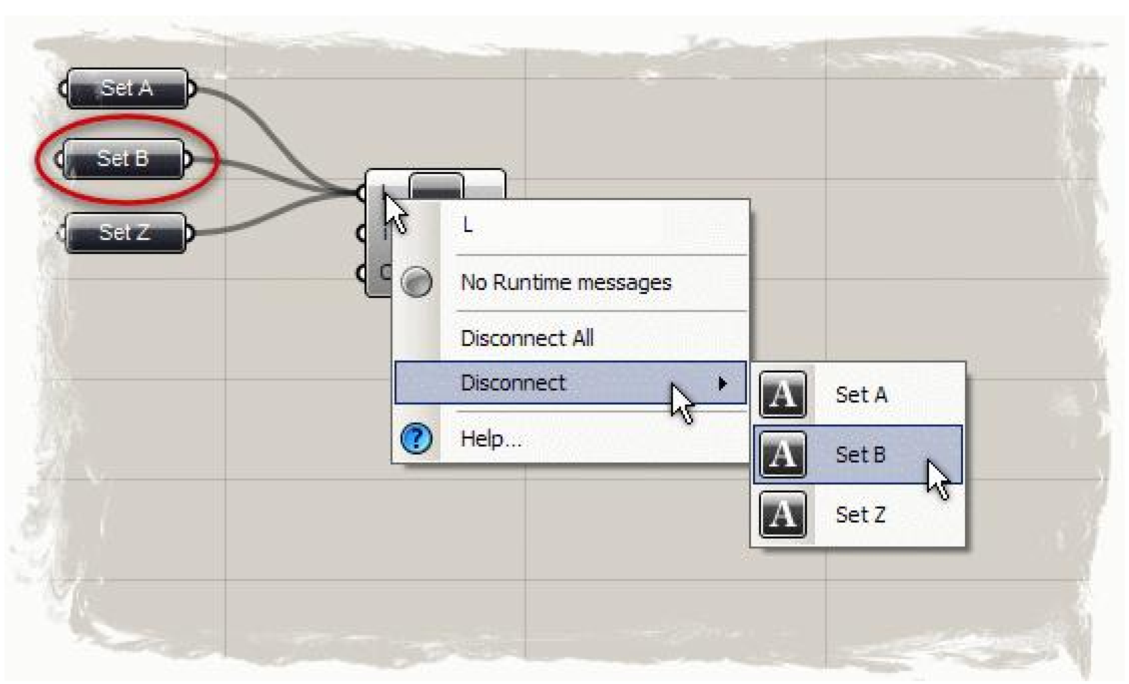
一度、接続が成立しているものに対して+Shift+キーを押しながら、再度、接続しようとしても何も起こりません。
同じソースから 1 回を越えて、データを継承することは禁じられています。

同様に、+CTRL+キーを押しながらドラッグすると、接続を解除することが出来ます。
この時、Version0.6 ではマウスカーソルは、赤い円に変わります。

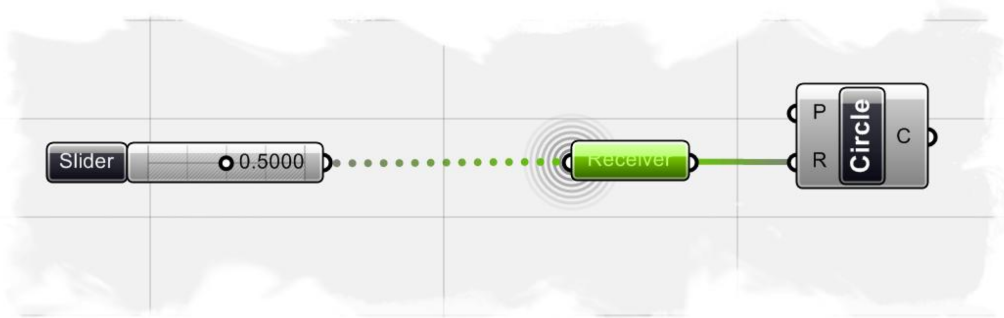




接続解除は、グリップ（パラメーターの小さなサークル部分）にマウスカーソルを持っていき、右クリックで、コンポーネントのコンテクストメニューから解除することも可能ですが、接続することは出来ません。

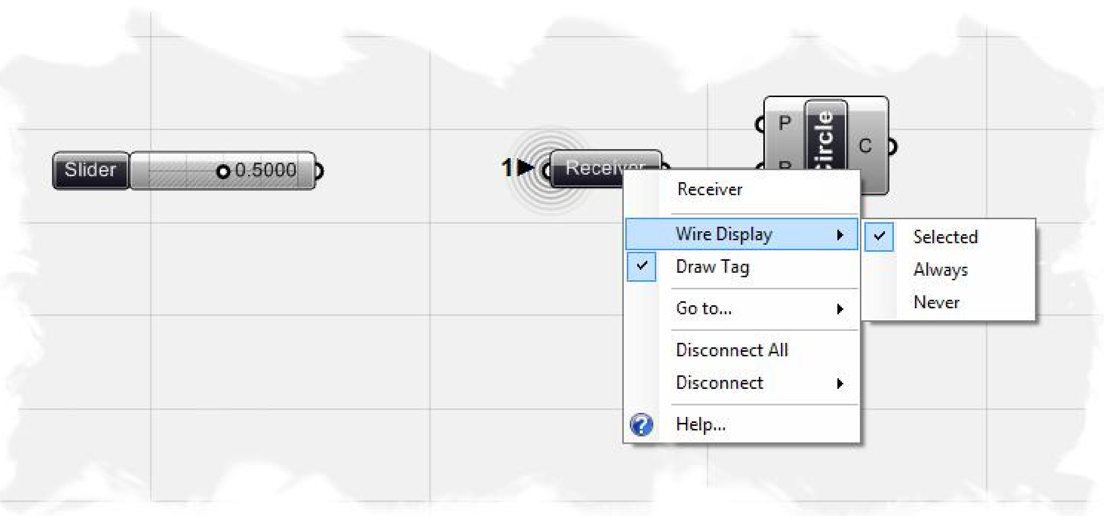

Set A
Set B
Set Z
L
No Runtime messages
Disconnect All
Disconnect
Help...
Set A
Set B
Set Z

Grasshopper は、+receiver+（Params>Special>Receiver）パラメーターを使用して情報をワイヤレスで伝達することが出来ます。接続する方法は他のコンポーネント同様、マウスドラッグをして接続しますが、接続後、マウスの左ボタンを離すと、接続ワイヤは自動的に消えます。接続後、ワイヤの表示オプションは、1)Receiver パラメーターを選択時にワイヤ表示、2)常に表示、3)常に表示しない、の 3 つがあり、初期値は、1)で、マウス左クリックで選択したときにのみワイヤが表示されます。



%slider+パラメーターが、+receiver+パラメーターを介して、+circle+コンポーネントに接続された例、が選択された状態で、ワイヤが表示されている。

%receiver+パラメーターが、選択されていない状態でワイヤは表示されない。+receiver+の左側グリップに表示されている数字+1+、接続されているコンポーネントの数を表示している。



ワイヤの表示オプションの指定方法。

6 *Data Stream Matching（データストリーム マッチング）*

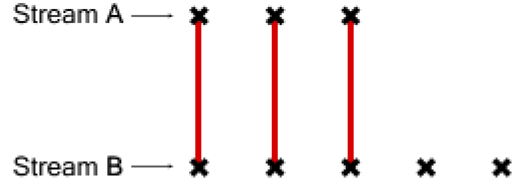
データマッチング

データのマッチングは、コンポーネントに対して、異なる入力の数を持つような場合、扱いが難しくなります。
例えば、点群間から作成されるラインセグメントを考えてみましょう。
ここでは、2 つのパラメーターがあり、それぞれ点の座標値からなるものとします。
（ストリーム A とストリーム B）
パラメーターが、何処からデータを持ってくるかに関しては重要ではありません。
コンポーネントは、それ自体の入出力パラメーターを越えて理解することは出来ません。

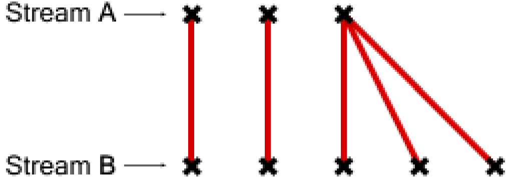
Stream A —— ✖ ✖ ✖

Stream B —— ✖ ✖ ✖ ✖ ✖

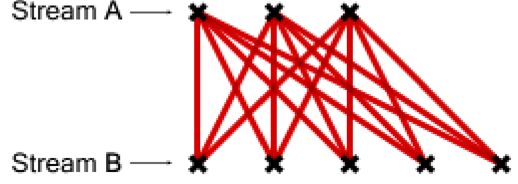
これらの点群間をどのように結線していくかは、いくつかの方法があります。
Grasshopper プラグインは、現時点で 3 種類のマッチングアルゴリズムを持ちますが、これ以外のものも可能です。
最も簡単な方法は、いずれかのストリームから点が無くなるまで一つ一つ接続していく方法です。これは、+Shortest List+アルゴリズムと呼ばれます。



次に、+Longest List+アルゴリズムというものですが、これは、両方のストリームの点が終わるまで接続する方法でこれがコンポーネントの初期状態の挙動になります。



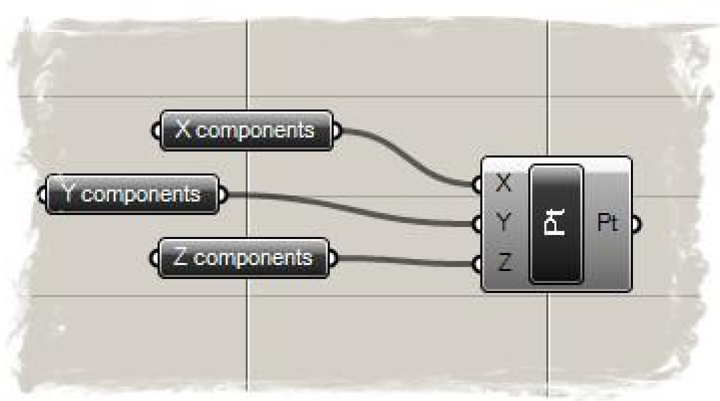
最後に+Cross Reference+アルゴリズムと呼ばれるもので、これは可能な全ての接続を行います。



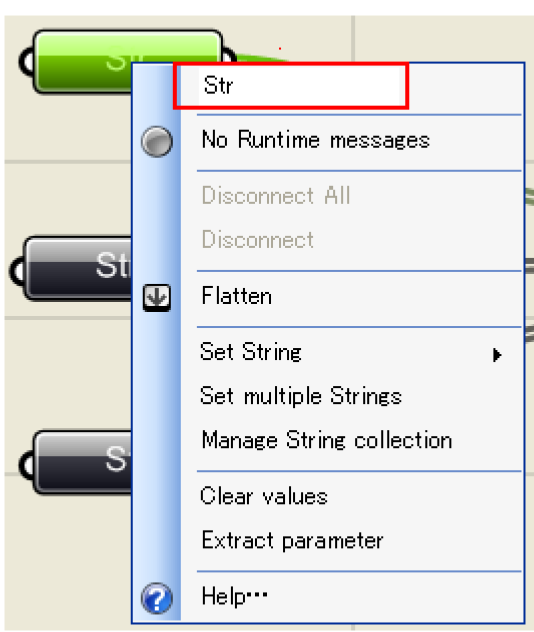
この方法は、結果が途方も無く多くなる可能性を含んでいるので気をつける必要があります。
多くの入力パラメーターがあり、一過性のデータが乗算され、これが継承されていくと
問題はさらに複雑になります。
ここで、ポイントコンポーネントが、外部のパラメーターから、X,Y,Z 値を受け取る例を見てみます。

X 座標: {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0}

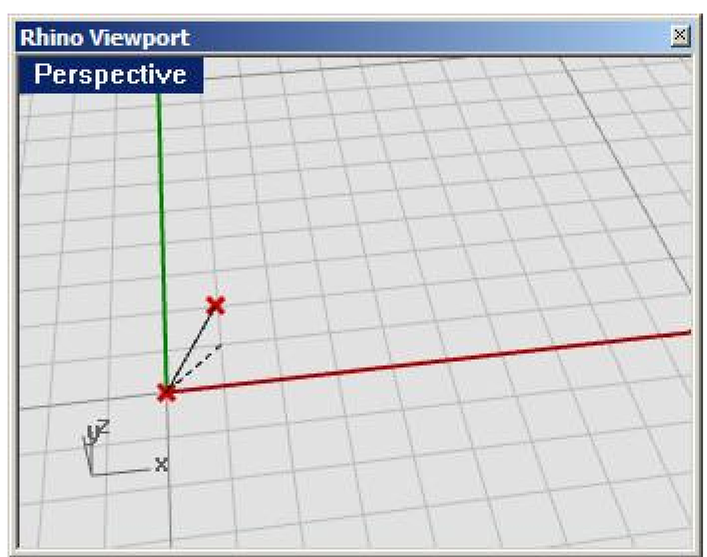
Y 座標: {0.0, 1.0, 2.0, 3.0, 4.0}
Z 座標: {0.0, 1.0}



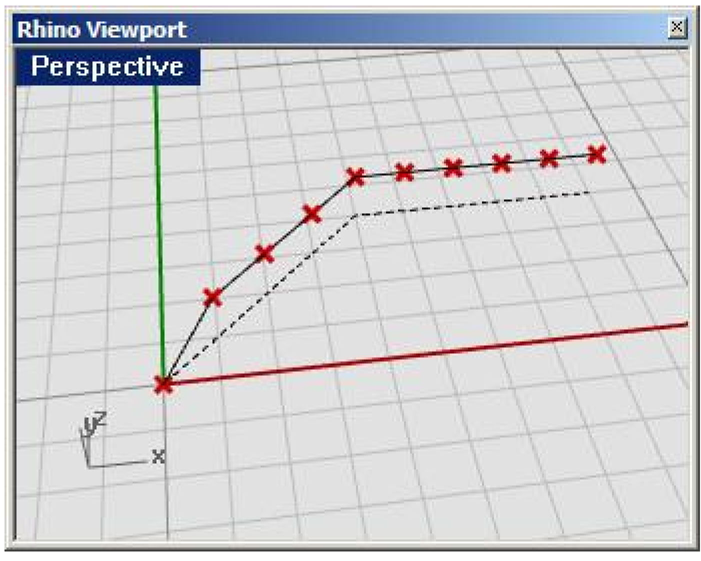
注；+PointXYZ+コンポーネント（Vector>Point >PointXYZ）の入力パラメーターに接続されている 3 つのパラメーターは+String+パラメータ（Params>Primitive>String）です。コンテクストメニューで、パラメーター名を、改名しています。



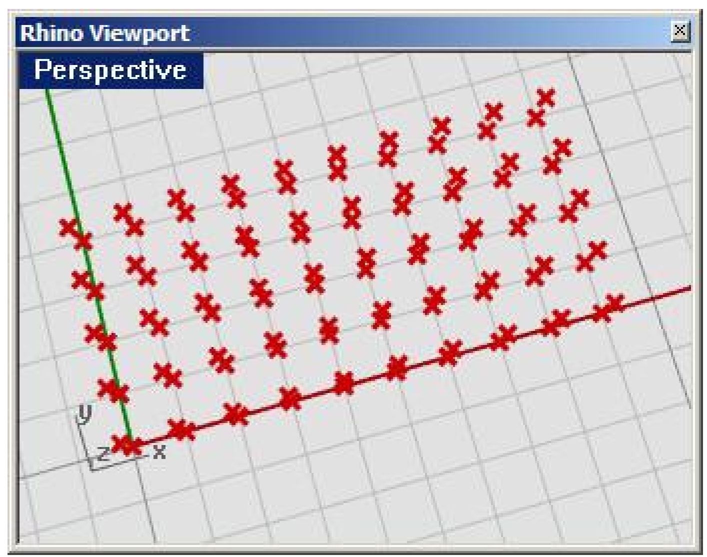
%Shortest List+方式で、接続していくと、結果は、Z 値が 2 つしかないので、入力結果からは 2 つのポイントしか生成されません。



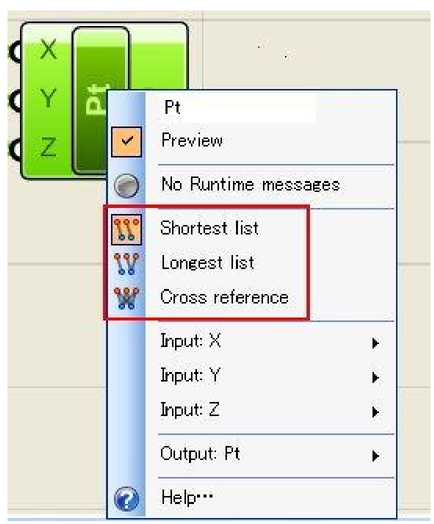
%Longest List+アルゴリズムは、10 個のポイントを作成します。



%Cross Reference+は、X,Y,Z のポイントの数をかけあわせた数だけ作成します。
つまり、10×5×2 = 100 のポイントを作成します。



全てのコンポーネントは、このいずれかの法則に従います。
（設定は、真ん中のコンポーネント部を右クリックしてコンテクストメニューを立ち上げて行います。）



例外もあります。コンポーネントがそれらの入力フィールドに 1 つもしくはそれ以上の、データリストを受け入れる場合、例えば+PolyLine+コンポーネントは点群の入力に対して、ポリラインを作成しますが、点群の入力を増やしていくと、より長いポリラインになり、複数のポリラインになるわけではありません。
入力パラメーターが、一つ以上の値をもつものは、リストパラメーターと呼ばれ、データマッチングは無視されます。

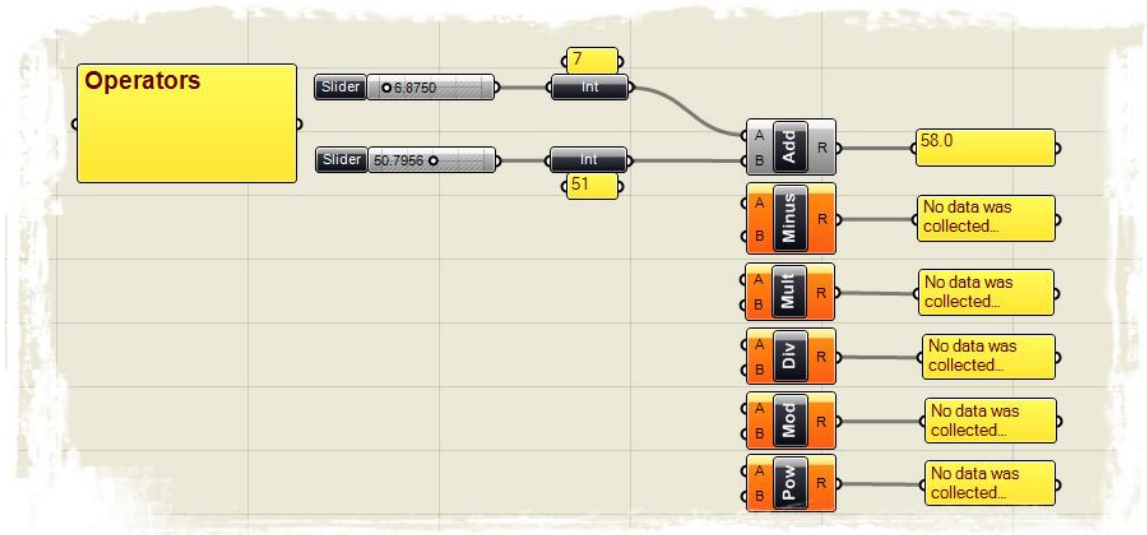
7 スカラーコンポーネントタイプ

スカラーコンポーネントは、様々な数学的な操作に使用され、以下から構成されます。

A) 定数．πや、黄金比等、定数を返します。
B) 数式．一つ又は複数の様々な変数関数（アルゴリズム）に使用されます。
C) インターバル．２つの極値から（またはドメイン=定義域から） インターバルパートにします。様々なインターバルコンポーネントが準備され、異なるインターバルタイプに分解することが出来ます。
D) オペレーター.四則演算等、数学的演算に使用されます。
E) 多項式．多項式に使用されます。
F) 三角法．Sine、Cosine、Tangent 等、一般的な三角法による値を返します。
G) ユーティリティー (Analysis)．2 つ以上の数値について分析します。

7.1 オペレーター

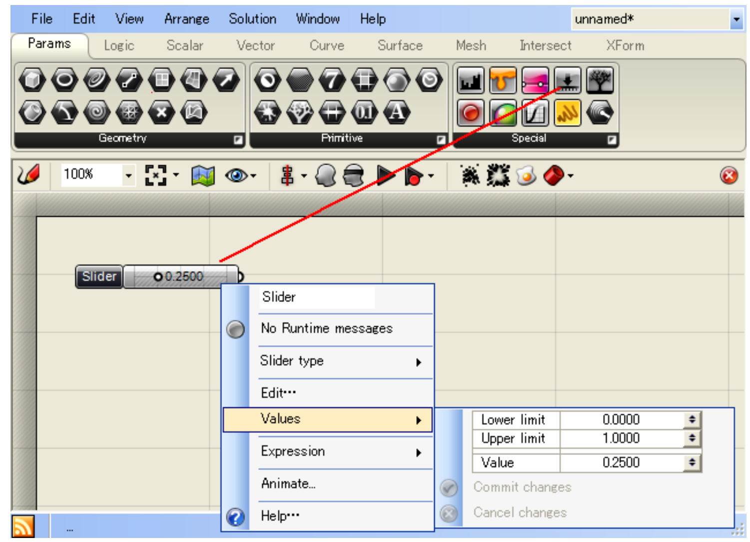
前述した通り、オペレーターは 2 つの数値入力による代数関数を使用するコンポーネントのセットで、一つの出力値を結果として出します。
より良く理解するため、簡単な数学定義で異なるオペレーターコンポーネントのタイプを見ていきます。
ここで紹介する最終の定義ファイルを見るには、ソースファイル中の **Scalar_operators.ghx** を開いてください。以下は最終の GH 定義のスクリーンショットです。



この GH 定義を最初から作成するためには、

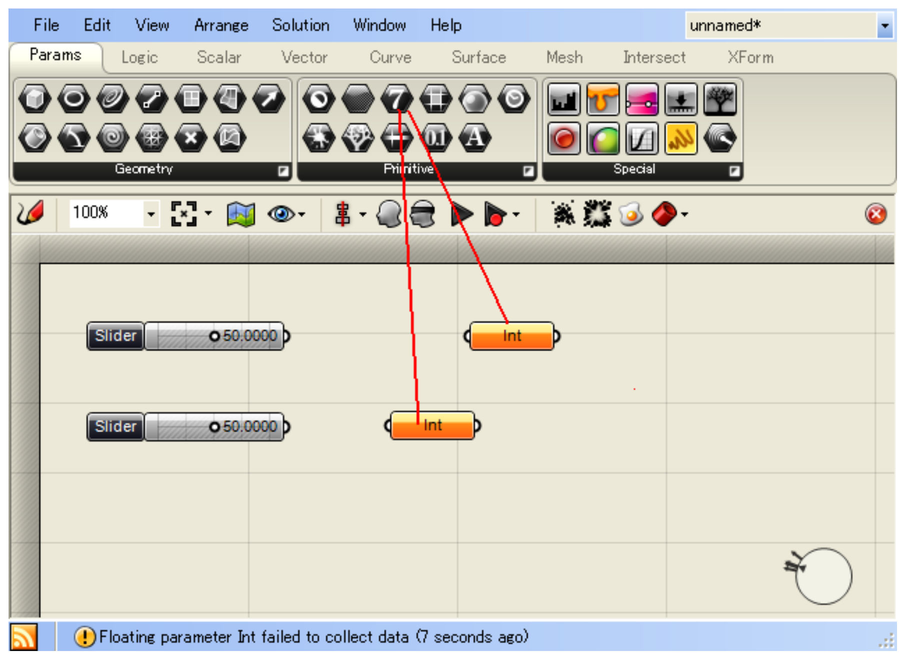
- %Number Slider+パラメーター（Params>Special>Number Slider）を、キャンバスにドラッグ＆ドロップします。
  キャンバスに+Number Slider+パラメーターをが配置されます。
- 次に+Number Slider+パラメーターを右クリックし、コンテクストメニューを開き取り扱う数値の範囲をセットします。
  - Lower limit: 0.0　（最小値を 0.0 にセット）
  - Upper limit: 100.0　（最大値を 100.0 にセット）
  - Value: 50.0　（初期値を 50.0 にセット）

  (注: この値はいつでも編集可能で 最小値～最大値の任意の値を設定しておきます。)



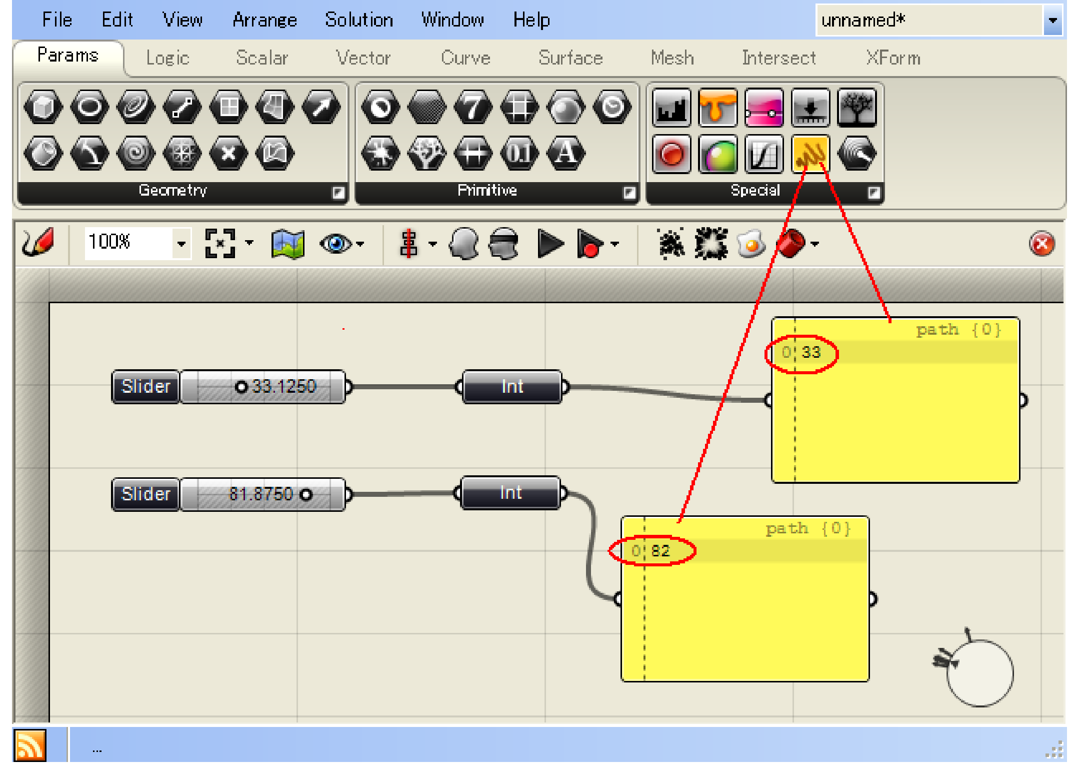
セットした+Number Slider+パラメーターを選択し、Cntrl+C (コピー)、 Cntrl+V (ペースト)を実行し、+Number Slider+パラメーターをコピーします。

- %Integer+パラメーター（Params>Primitive>Integer） をキャンバスにドラッグ＆ドロップします。

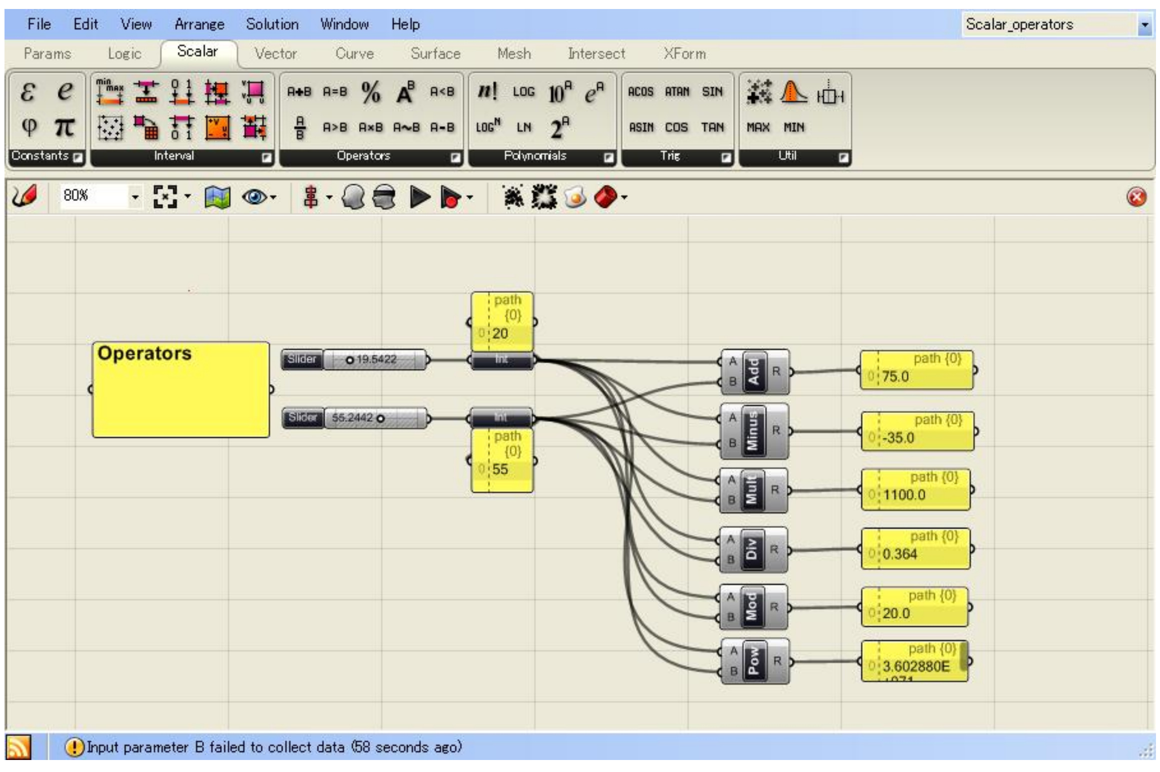


1 つめのöNumber Slideröパラメーターを、最初のöIntegeröパラメーターに接続します。

- 2 つめの+Number Slider+パラメーターを、残りの+Integer+パラメーターに接続します。
スライダーの初期値は、フローティング（浮動少数点）の 10 進数に設定されています。Number Slider+パラメーターを+Integer+パラメーターに接続することで、フローティングを整数に変換することが出来ます。この変換結果を見るためには、+Panel+パラメーター（Params>Special> Panel）をキャンバスに配置し、+Integer+パラメーターの出力を接続するとその変換結果を見ることができます。



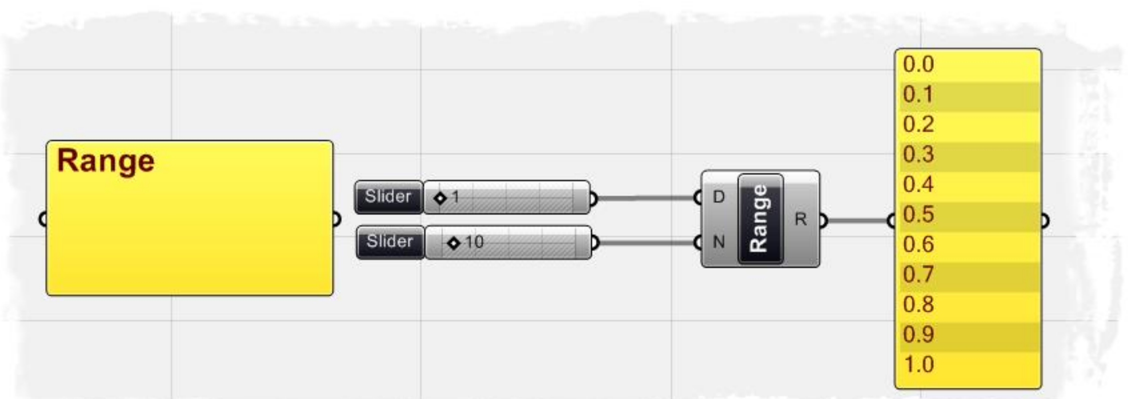
- 次に、+Addition+コンポーネント（Scalar>Operator>Addition）をキャンバス上にドラッグ＆ドロップします。
- 最初の+Integer+パラメーターを、+Add+コンポーネントの A-input に接続します。
- 2 番目の+Integer+パラメーターを、+Add+コンポーネントの B-input に接続します。
- %Panel+パラメーター（Params>Special>Panel）をキャンバス上にドラッグ＆ドロップします。
- %Add+コンポーネントのの R- 出力を+%Panel+パラメーターに接続します。
  これで、2 つの整数のシミュレーション結果を視覚的に確認できます。
- 次に、残りのスカラーオペレーターをキャンバス上にドラッグ＆ドロップします。
  - %Subtraction+コンポーネント（Scalar>Operator> Subtraction）
  - %Multiplication+コンポーネント（Scalar>Operator> Multiplication）
  - %Division+コンポーネント（Scalar>Operator> Division）
  - %Modulus+コンポーネント（Scalar>Operator> Modulus）
  - %Power+コンポーネント Scalar>Operator> Power）
- 最初の+Integer+パラメーターを全てのオペレーターコンポーネントの A-input に接続します。
- 2 番目の+Integer+パラメーターを全てのオペレーターコンポーネントの B-input に接続します。
- Params/Special/メニューの、+Panel+パラメーター をキャンバス上に配置し、オペレーターコンポーネントの数だけ配置し、その出力を+Panel+パラメーター に接続します。
  これで、GH 定義は終了しました。これで、スライダーの値を変えるとそれぞれのオペレーターの演算結果が、"Panel"パラメーターにて確認できます。


File
Edit
View
Arrange
Solution
Window
Help
Scalar_operators
Params
Logic
Scalar
Vector
Curve
Surface
Mesh
Intersect
XForm
Constants
Interval
Operators
Polynomials
Trig
Util
Operators
Add
Minus
Mult
Div
Mod
Pow
75.0
-35.0
1100.0
0.364
20.0
Input parameter B failed to collect data (58 seconds ago)

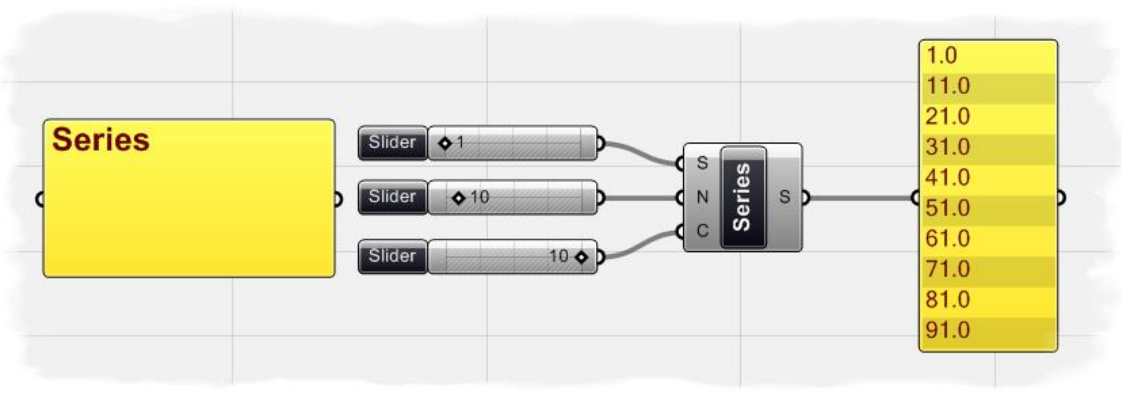
7.2 Range vs. Series vs. Interval”Component

%Range+、+Series+、+Interval+コンポーネントは、2 つの入力値を参照し、それぞれ数値のセットを生成しますが、それぞれ異なる操作を行います。

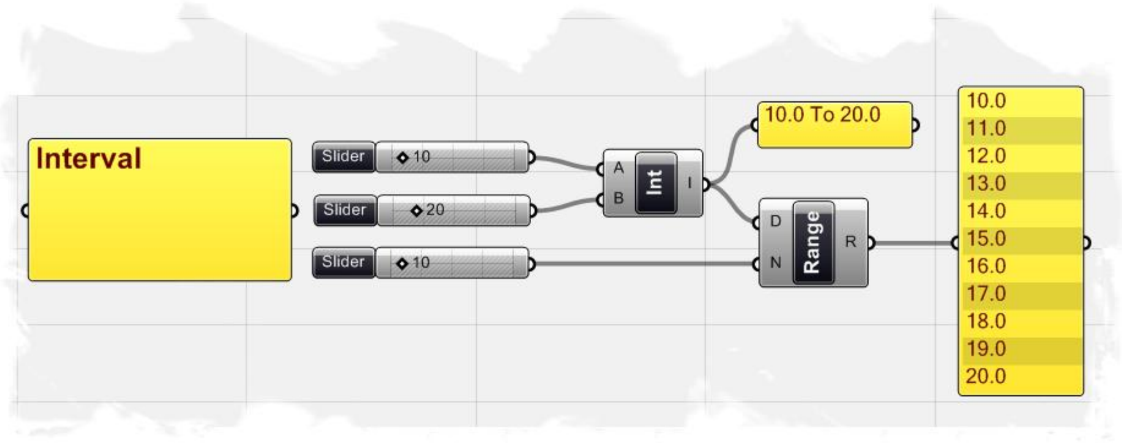
Note: この結果は、**Scalar_intervals.ghx** を開いて見ることが出来ます。



%Range+コンポーネントは、+domain of numeric range（ドメインの数値範囲）+と呼ばれる最小値と最大値を均等に分割した数値列を出力します。
上の例では、2 つの+Number Slider+パラメーターが、+Range+コンポーネントの入力に接続されています。
最初の+Number Slider+パラメーターは、数値範囲のドメインを決めます。この例では、パラメーターにセットされた数値+1+より、ドメインは+0+から+1+となっています。
2 番目の+Number Slider+パラメーターは、ドメインを分割する数を定義しており、この例では 10 に設定されていますので、+Range+コンポーネントの出力値は、+0+から+1+を、均等に分割された 11 の数値列となります。



%Series+コンポーネントは、開始値、ステップ、連続的に増加する数値の数の指定から、数値列を作成します。
この例では、３つの+Number Slider+パラメーターが、+Series+コンポーネントの入力に接続され、最初の+Number Slider+パラメーターは、開始値を定義します。
2 番目の+Number Slider+パラメーターは、連続的に増加するステップ値を、+10+に定義しています。開始値が、+1+ですから、次の値は+11+になります。
3 番目の+Number Slider+パラメーターは、トータルでいくつ連続させるかを定義しますので、+1+からスタートし、+91+までの、10 個の数値列が出力されます。



%Interval+コンポーネントは、最小値、最大値指定による範囲を出力します。
%Interval+コンポーネントは、+Range+コンポーネントに類似していますが、
大きな違いは+Range+コンポーネントは、初期状態では、必ず+0+から始まる範囲に設定されますが、+Interval+コンポーネントは、A 入力、B 入力によって、最小値、最大値による範囲指定となります。
下記の例では、2 つの+Number Slider+パラメーターによって、可能範囲を、+10+から+20+に設定した例です。
%Interval+コンポーネントの出力は、新しい領域定義を反映して+10+から、+20+になっているのが分かります。
ここで、+Interval+コンポーネントの出力を、+Range+コンポーネント D=入力に接続すると、+Range+コンポーネントの出力範囲は、+Interval+コンポーネントの出力値になります。

ここの場合、+Range+コンポーネントの例のように、出力される数値列は 11 個になります。

（注；インターバルを決める方法はいくつかあります、+Scalar>Interval %メニューに他のコンポーネントがあります。ここで紹介したような簡単なインターバルを定義することはありません。その他の方法については、次の章において述べます。）

7.3 Functions & Booleans（関数と論理値）

ほぼ全てのプログラミング言語は、条件文を判断する方法を持ちます。
ほとんどの場合、プログラマーは、簡単な”what if?（もし．．．であったら）文”のようなコードを書きます。例えば、+もし、9.11 のテロが起きていなかったら+、+もし、ガソリンが 1 ガロン、10 ドルしたなら+のような**”what if?**文**”**です。

これらは、より高いレベルの思考を要約するための、重要な問いになります。

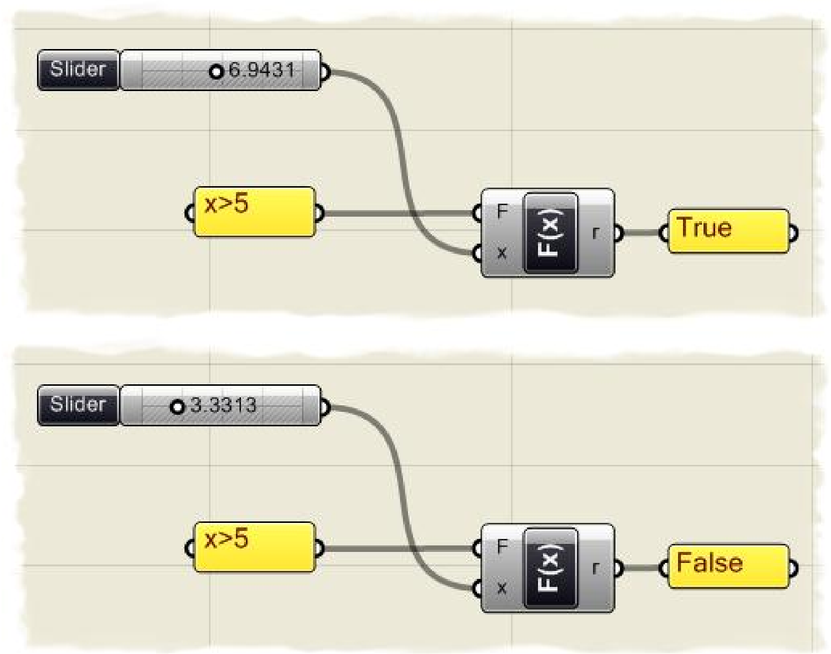
コンピュータプログラムは、+what if?+の問いを解析することが出来、その答えに対するアクションを起こします。
ここで、簡単な条件文を見て行きましょう。

もしもオブジェクトが+カーブ+であったならば、削除せよ。

最初の一文（もしもオブジェクトが+カーブ+であったならば）は、オブジェクトを認識し、+カーブ+であった場合、論理値を決めます。
論理値は、中間は無く、オブジェクトが+カーブ+の場合は**+True+**を、そうでない場合は**+False+**を返します。

2 番目の宣言文（削除せよ）は、条件文による出力結果に対して、アクションを実行します。
この場合、オブジェクトが+カーブ+であったならば、削除せよ。
このような、条件文を、**”If/Else** 文**”**と呼びます。もしも、オブジェクトがある条件を満たしたならば、何かを行い、そうでなければ、何も行いません。

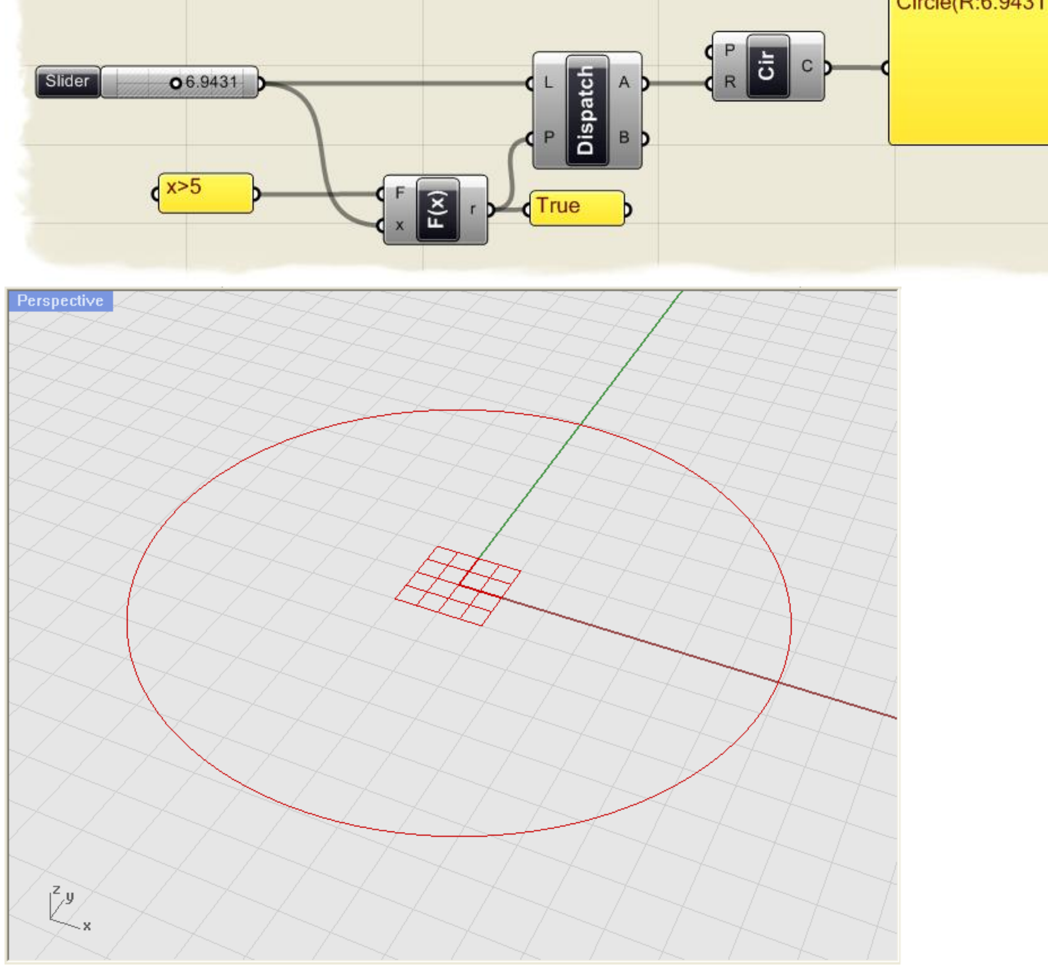
Grasshopper も+Function+コンポーネントによって、同様の条件解析を行うことが出来ます。



上記の例では、+Number Slider+パラメーターを、変数を一つだけ扱う+Function（F1）+コンポーネント（Logic>Script>F1）の、X-入力に接続したものです。
そして、条件文、+X は、+5+より大きいか+を定義し、+Function+コンポーネントの F 入力に接続します。

%Number Slider+パラメーターの値が、5 より大きくセットされると、r-出力値は、論理値+True+を示します。
5 以下であれば、論理値+False+となります。

次に、この論理値を返す機能を、+Dispatch+コンポーネント（Logic>List>Dispatch）P-入力に接続して、あるアクションを実行してみましょう。
%Dispatch+コンポーネントに、L-入力の値を、P-入力に条件判断の論理値を接続します。
もし、論理値+True+が返れば、+Dispatch+コンポーネントの、A-出力に渡され、+False+の場合、B-出力に渡されます。」
例えば、X の値が、5 以上の場合のみ、+円+を描くというアクションを行うとします。
この場合、+Dispatch+コンポーネント、A-出力を+Circle+コンポーネント（Curve>Primitive>Circle）の R-入力に接続します。
%Number Slider+パラメーターによる入力値に対して、+True+値が返る場合のみ、円が作成されます。
Dispatch+コンポーネント B-出力には、何も接続されていませんので、+False+の場合は、何もアクションは起きません。



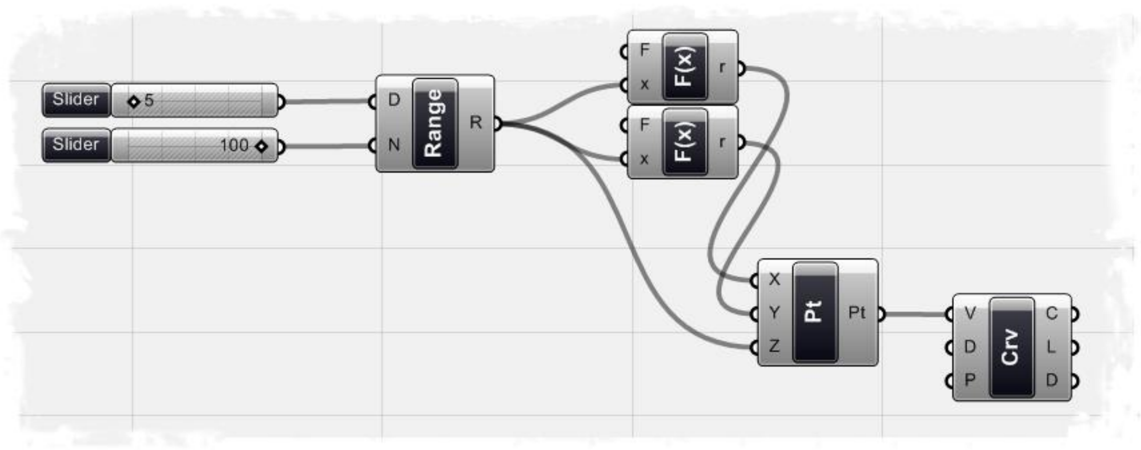
注：この結果は、CircleBooleanTest.ghx で見ることが出来ます。

7.4 Functions & Numeric Data（関数と数値データ）

%Function+コンポーネントはとても柔軟性があり、様々な異なるアプリケーションで使用することができます。
既に条件文の扱い方とその論理値の出力に関しては説明しました。が、+Function+コンポーネントは、複雑な数学的アルゴリズムを解くことができ、数値データを出力することが出来ます。
次の例では、David Rutten 氏の+*Rhinoscript 101*+マニュアルで書いてあるような、数学的な螺旋を作ってみます。

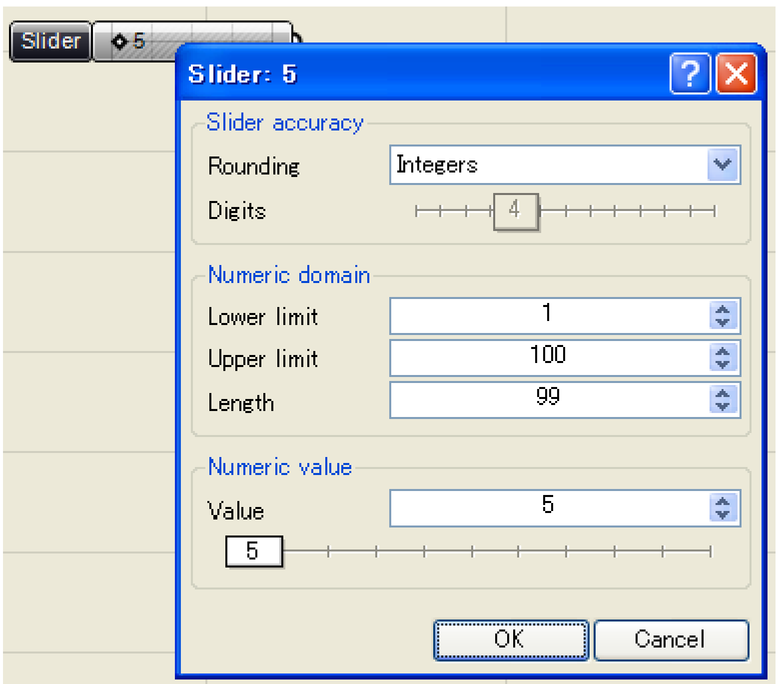
注；+*Rhinoscript 101*+は下記でダウンロードすることができます。
http://en.wiki.mcneel.com/default.aspx/McNeel/RhinoScript101.html

注：この最終結果は **Function_spiral.ghx** を開いて見ることが出来ます。

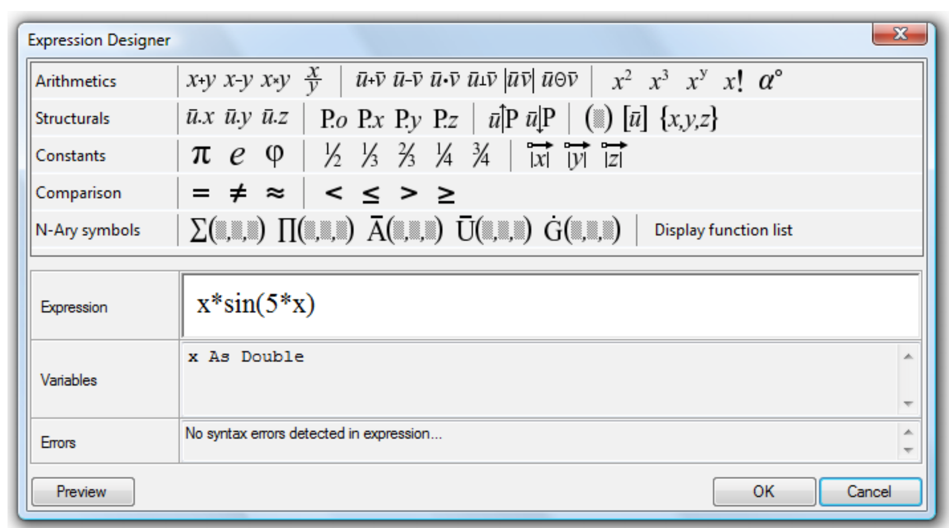


この GH 定義をスクラッチから作成するには

- %Range+コンポーネント（Logic>Sets>Range）をキャンバスにドラッグ＆ドロップします。
- %Number Slider+パラメーター（Params>Special>Number Slider）をキャンバスにドラッグ＆ドロップします。
- それぞれの+%Number Slider+パラメーターを右クリックし、コンテクストメニューにて、+Slider Type+を+Integer+にします。
- 同様に、+Edit+にて、それぞれの Slider パラメーターの下限値、上限値を+1+、+100+に設定します。



- 1 つめの+Slider+パラメーターを+5+に設定します。
- 2 つめの+Slider+パラメーターを+100+に設定します。
- 1 つめの+Slider+パラメーターを+Range+コンポーネント D-入力に接続します。
- 2 つめの+Slider+パラメーターを、+Range+コンポーネント・N-入力に接続します。この結果、+Range+コンポーネントは、0.0 から、5.0 までを、100 に均等に分割された、101 個の数値を持つことになります。
- %Function（F1）+コンポーネント（Logic>Script>F1）をキャンバスにドラッグ＆ドロップします。
- %Function（F1）+コンポーネントの、+F-input+を右クリックし、メニューから+Expresseion Editor+を開きます。
- %Expresseion　Editor+ダイアログで、以下の数式を入力します。
  - x*sin(5*x)
  - %OK+をクリックし、決定します。



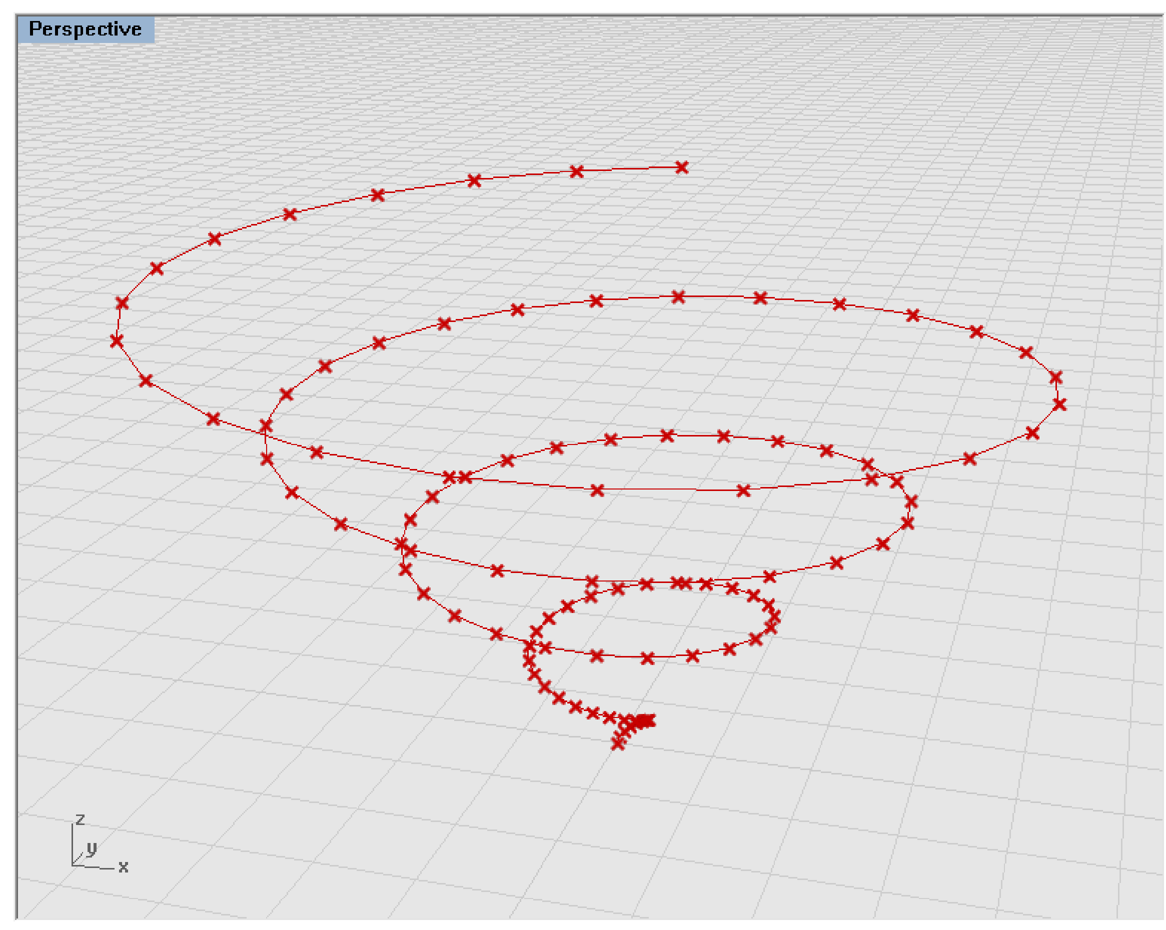
- %Function+コンポーネントを選択し、Ctrl+C (コピー) 、 Ctrl+V (ペースト)でコピーします。
- コピーした+Function（F1）+コンポーネントの、+F-input+を右クリックし、メニューから+Expresseion　Editor+を開きます。

- 次にコピーした、+Function+コンポーネントの数式を+Expresseion　Editor+ダイアログにて、下記のように入力します。
    - x*cos(5*x)
    - %OK+をクリックし、決定します。
- %Range+コンポーネントの R-出力を、2 つの、+Function+コンポーネントの、X-入力に接続します。

  %Range+コンポーネントで作られた 101 個の数値が+Function+コンポーネントに供給され、それぞれの数学的な解が数値データとして出力されることになります。
- %Point+コンポーネント（Vector>Point>PointXYZ）をキャンバスにドラッグ＆ドロップします。
- 最初の+Function+コンポーネント r-出力を+Point+コンポーネント X-入力に接続します。
- 2 番目の+Function+コンポーネント r-出力を、+Point+コンポーネント、Y-入力に接続します。
- %Range+コンポーネント R-出力を+Point+コンポーネント Z-入力に接続します。

  ここで Rhino のビューポートを見ると、点群が螺旋状に配置されているのが見えます。この時に 2 つの+Slider+パラメーターの値を変えることで、点群列の定義を変えることが出来ます。
- %Curve+コンポーネント（Curve>Spline>Curve）をキャンバスにドラッグ＆ドロップします。
- %Point+コンポーネント Pt-出力を、+Curve+コンポーネント V-入力に接続します。

  これで、点群座標値を通過する螺旋のカーブが作成されました。+Curve+コンポーネント D-入力を右クリックして、カーブの次数を設定することが出来ます。次数+1+を定義すると、直線によるポリラインが作成されます。次数+3+で、スムースな Bezier カーブが作成されますが、この場合、この点群座標は通過しません。



注；このビデオチュートリアルは、Zach Downey'氏のブログで見ることが出来ます。
http://www.designalyze.com/2008/07/07/generating-a-spiral-in-rhinos-grasshopper-plugin/

7.5 Trigonometric Curves（三角関数曲線）

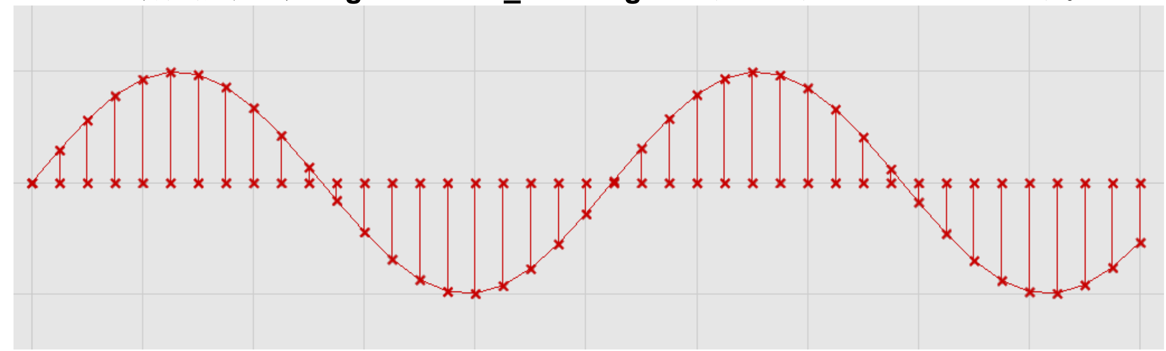
これまでに、%Function+コンポーネントによって複雑な数式から、螺旋や他の数学的なカーブが出来ることを見てきました。
Grasshopper は+三角関数+コンポーネントによって+スカラー+コンポーネントに変換することも可能です。

２つの辺から定義される、角度+θ+によって定義される、+Sine+、%Cosine+、%Tangent+等による、三角関数は、数学者、科学者、エンジニアにとって重要です。

これらの関数は、後述するベクトル解析においても非常に重要です。
ここでは、これらの関数を使用して、自然界においてみられる海の波、音波、光の波動等、正弦波のような周期的現象を定義してみましょう。

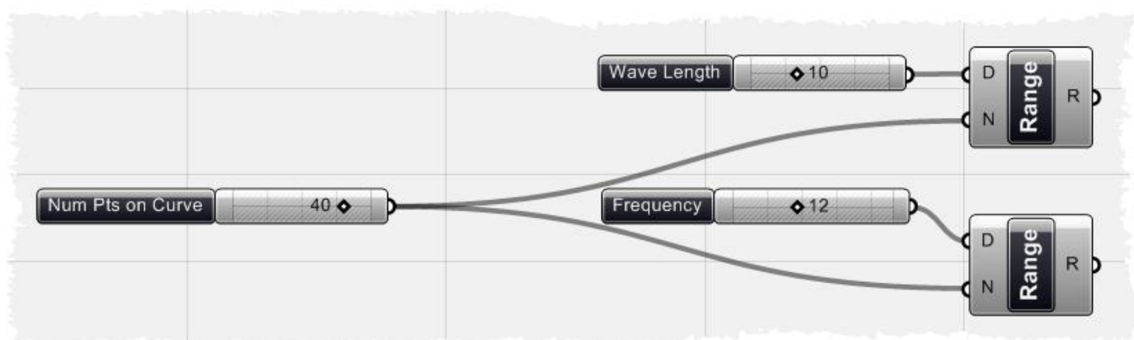
次の例ではカーブ上の点群、波長、周期から、正弦波の形状を作成し、これを数値のスライダーでコントロールする定義を見てみます。

注：この最終結果は、**Trigonometric_curves.ghx** を開いて見ることができます。



この GH 定義をスクラッチから作成するには

- %Number Slider+パラメーター（Params>Special>Number Slider）を、3 つ、キャンバスにドラッグ＆ドロップします。
- 最初の+Slider+に、以下のパラメーターを定義します。
    - Name: Num Pts on Curve　と表示名を変える。
    - Slider Type: Integers　タイプを、Integers（整数）に設定
    - Lower Limit: 1　最小値を、+1 +に設定
    - Upper Limit: 50　最大値を、+50+に設定
    - Value: 40　値を、+40+に設定
- 2 番目の+Slider+に、以下のパラメーターを定義します。:
    - Name: Wave Length　と表示名を変える。
    - Slider Type: Integers　タイプを、Integers（整数）に設定
    - Lower Limit: 0　最小値を、+0+に設定
    - Upper Limit: 30　最大値を、+30+に設定
    - Value: 10　値を、+10+に設定
- 3 番目の+Slider+に、以下のパラメーターを定義します。
    - Name: Frequency　と表示名を変える
    - Slider Type: Integers　タイプを、Integers（整数）に設定
    - Lower Limit: 0　最小値を、+0+に設定
    - Upper Limit: 30　最大値を、+30+に設定
    - Value: 12　値を、+12+に設定
- %Range+パラメーター（Logic>Set>Range）を、2 つ、キャンバスにドラッグ＆ドロップします。
- %Wave Length slider+出力を、1 つめの+Range+パラメーターD-input に接続します。
- %Frequency slider+出力を、2 つめの+Range+パラメーターD-input に接続します。
- %Num Pts on Curve slider+出力を、 2 つの+Range+パラメーター、N-input に接続します。

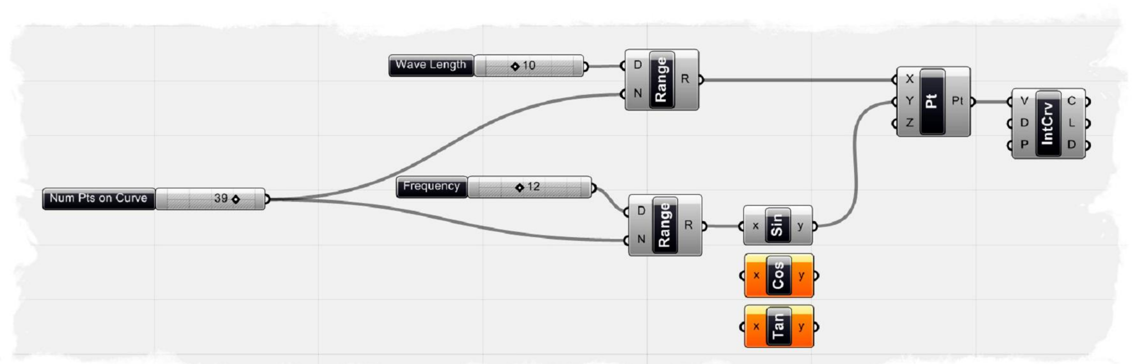


ここまで、実行すると、上図のような GH 定義が出来ているはずです。
これで、2 つのリストデータ作成されました。一つ目は、0～10、2 つめは、0～12 をそれぞれ、均等に分割したものです。

- %Sine+コンポーネント（Scalar > Trig> Sine）をキャンバスにドラッグ＆ドロップします。
- 2 つめの、+Range+コンポーネントの、R-出力を、+Sine+コンポーネントの、X-入力に接続します。
- %PointXYZ+コンポーネント（Vector> Point> PointXYZ）をキャンバスにドラッグ＆ドロップします。
- 1 つめの、+Range+コンポーネントの R-出力を、+PointXYZ+コンポーネント X-入力に接続します。

- %Sine+コンポーネント Y-出力を、PointXYZ+コンポーネント Y-入力に接続します。
  ここで、Rhino のビューポートを見ると、点群が正弦波上に並ぶように配置されているのがみえるはずです。最初の+Range+コンポーネントは、+PointXYZ+コンポーネントに、X 座標の値を渡します。三角関数を適用しない状態では、点群は X 方向に均等に配置されます。+Sine+コンポーネントを介して+PointXYZ+コンポーネントの、Y-入力に接続すると、出力される点群に Y 座標が与えられ、正弦波上に配置されます。
- 次に+Interporate Curve+コンポーネント（Curve> Spline> Interporate）をキャンバスにドラッグ＆ドロップします。
- PointXYZ+コンポーネントの Pt-出力を、+Interporate Curve+コンポーネント V-入力に接続します。

この時点で、GH 定義は下記のようになっているはずです。

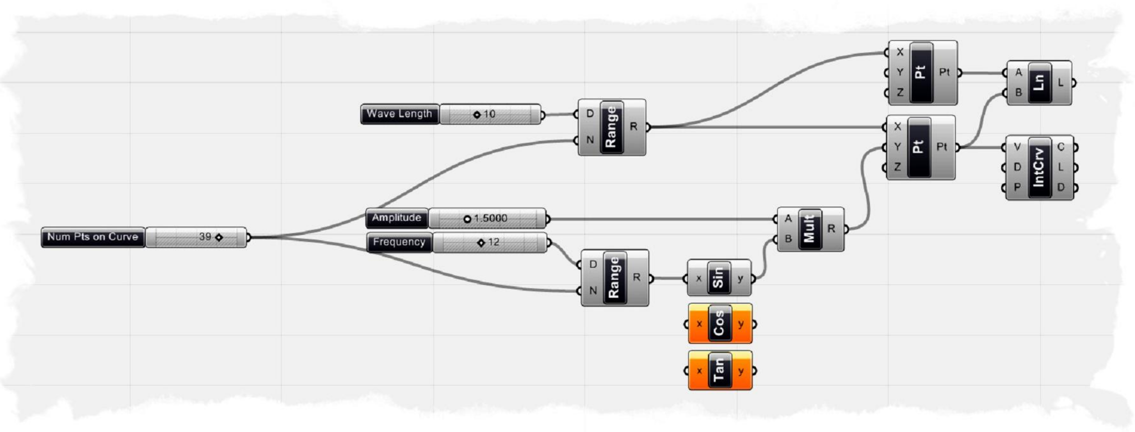


- 次にもう一つ+Slider+パラメーター（Params>Special>Slider）を、キャンバスにドラッグ＆ドロップし、追加します。
- 最初の+Slider+に、以下のパラメーターを定義します。
  - Name: Amplitude　と表示名を変える。
  - Slider Type: Floating　タイプを、浮動少数点に設定
  - Lower Limit: 0.1　最小値を、+0.1+に設定
  - Upper Limit: 5.0　最大値を、+5.0+に設定
  - Value: 2.0　値を、+2.0+に設定
- 次に+Multiplication+コンポーネント（Scalar>Operator>Multiplication）を、キャンバスにドラッグ＆ドロップします。
- %Amplitude+の出力を+Multiplication+コンポーネントの A-入力に接続します。
- %Sin+コンポーネントの y-出力を+PointXYZ+コンポーネントから+Multiplication+コンポーネント B-入力に接続し直します。
- %Multiplication+コンポーネント R-出力を+PointXYZ+コンポーネント Y-入力に接続します。
  これで、以前の接続の間に、新たに正弦波の振幅のパラメーターを設定することで、0.1 倍～2 倍まで、振幅を調整するように設定したことになります。
- 次にもう一つ+PointXYZ+コンポーネント（Vector> Point> PointXYZ）をキャンバスにドラッグ＆ドロップします。
- 1 つめの、+Range+コンポーネントの、R-出力を、+PointXYZ+コンポーネント X-入力に接続します。
- %Line+コンポーネント（Curve> Primitive> Line）をキャンバスにドラッグ＆ドロップします。
- 1 つめの PointXYZ+コンポーネント Pt--出力を、+Line+コンポーネント B-入力に接続します。

- 2 つめの、+PointXYZ+コンポーネント Pt--出力を、+Line+コンポーネント A-入力に接続します。

    この最後の定義は、sine カーブ上の同じ X 座標上に対応し X 軸上に均等に配置された点群を作成したものです。+Line+コンポーネントは、これらの点群を結んだものです。新しい直線群で、垂直方向の偏差を見ることが出来ます。最終的な GH 定義は下記のようになっているはずです。

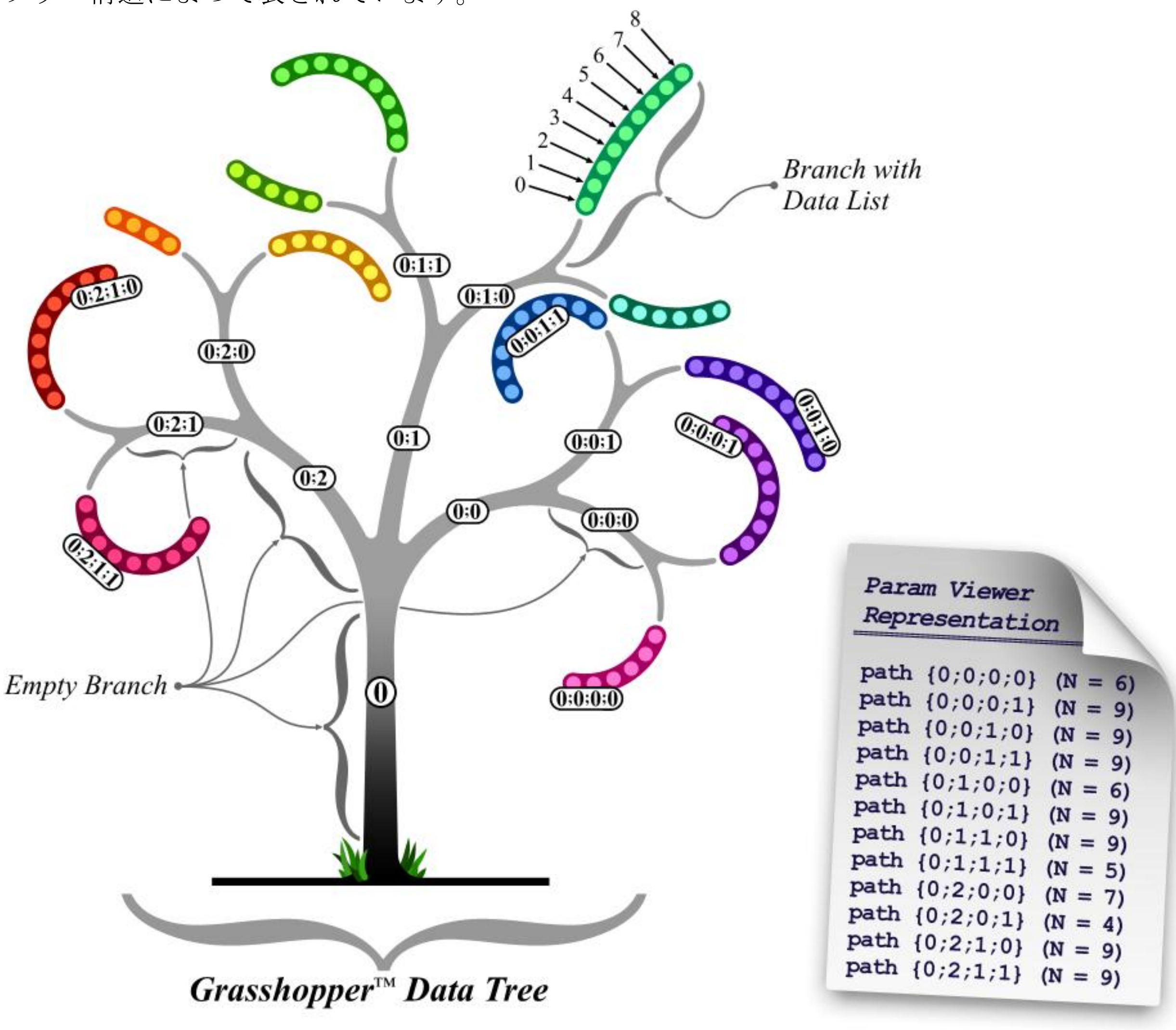
以上、sine カーブでこの結果を見てみましたが、+Cos+コンポーネント、+Tan+コンポーネントに置き換えて異なる結果をみることが出来ます。



Note: David Fano'氏によるビデオチュートリアルが下記 URL で見ることが出来ます。
http://designreform.net/2008/06/01/rhino-3d-sine-curve-explicit-history/

8 The Garden of Forking Paths（階層構造について）

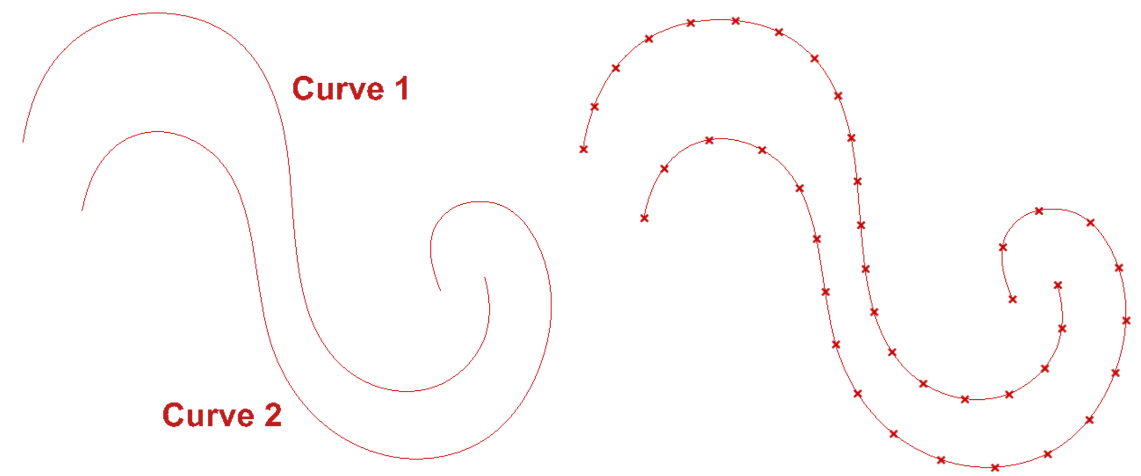
Grasshopper の Version　0.6.00XX 以降では、一つのパラメーターの内部に複数のデータのリストを格納することが出来ます。複数のリストを扱うためには、個々のリストを認識する方法が必要となります。下図は、Grasshopper 開発者の David Rutten によって描かれたイメージでツリー構造によって表されています。



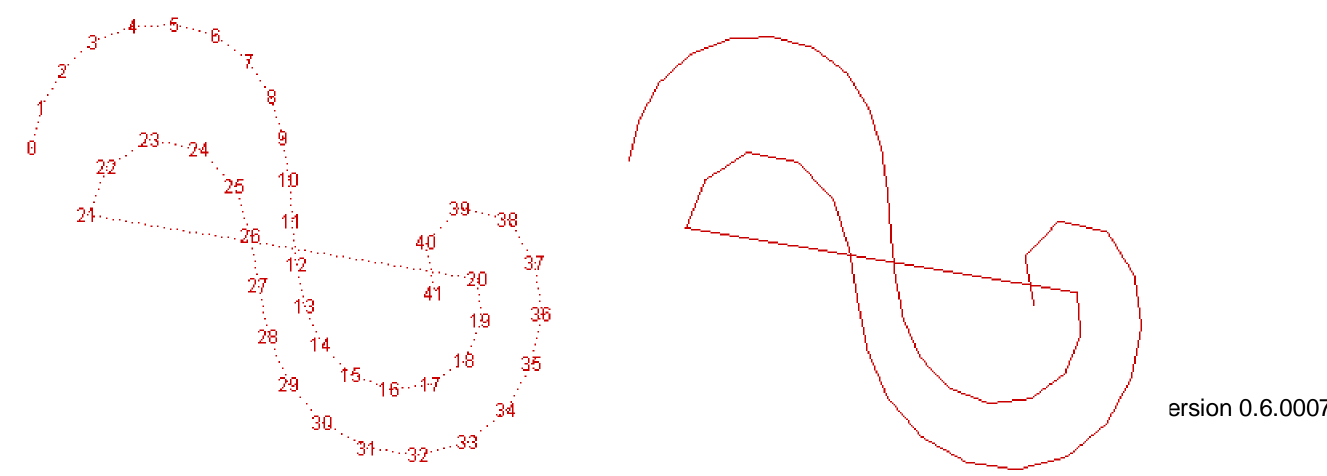
上記のイメージで分かるのは、一つの基本となるブランチ（枝（幹といっても良い））が、path｛0｝に相当します。このパスは、データは持ちませんが、3 つのサブ・ブランチを持ちます。これらの 3 つのサブ・ブランチは、マスター・ブランチのインデックッス｛0｝を持ち、自身のインデックス、0,1,2 を持ちます。

これをインデックスと呼ぶのは間違いかもしれません。というのはマスター・ブランチは一つの数字しか含まないからです。
またこの構造はコンピューターのハードディスクのフォルダ構造と似ていますから、+パス+と呼んだほうが良いかもしれません。
それぞれのサブ・ブランチはまた、2 つのブランチに分岐し、このサブ・ブランチ自身はデータを持ちません。そしてこれは、サブ・サブ・ブランチについても同様の事がいえます。
レベル 4 まで達したときに、やっとデータに出会います。（ここではデータのリストは、色の付いた太い線で表示され、データ自身は明るい円で表されています。）
全ての、サブ・サブ・サブ・ブランチ（またはレベル 4 ブランチ）は、最後のブランチであり、これ以上分岐することはありません。

これ以上のツリー構造については、簡単な例をみることにしましょう。
2 つのカーブを Grasshopper で参照してみます。まず、+Divide+コンポーネント（Curve>Division>Divide Curve）で、カーブを 20 のセグメントに分割します。
結果として、それぞれのカーブは 21 のポイントを持ち、計 42 のポイントのリストを持ちます。
これらの全てのポイントを+Polyline コンポーネント（Curve>Spline>Polyline）+に与えると、新しいラインデータが分割点を通り作成されます。

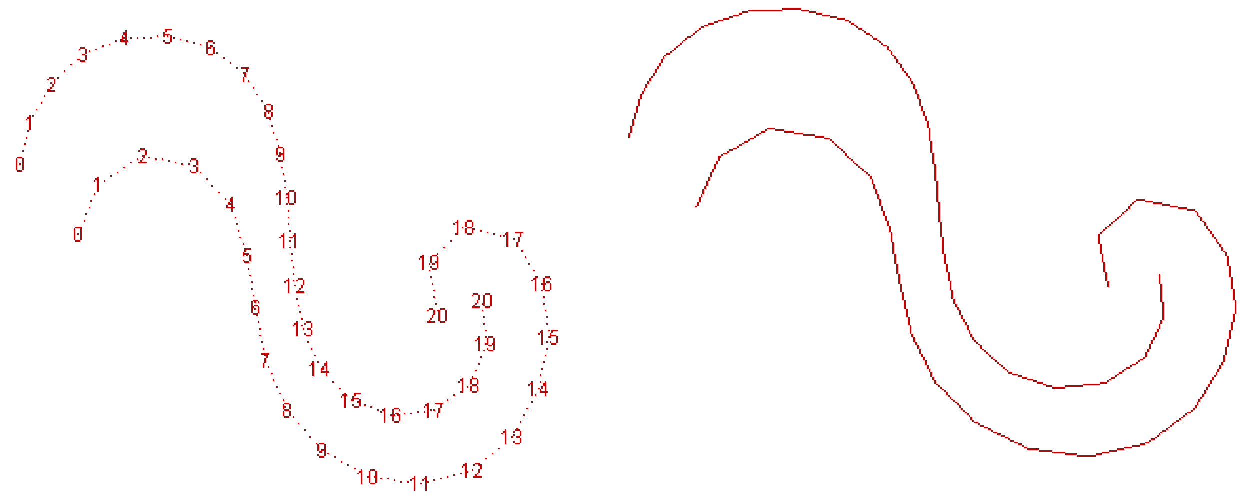


Grasshopper の 0.5 または、それ以前の version では、ポリラインカーブは、全ての点群（ここでは、42 個の点群）から、たった一つのラインしか作成することが出来ませんでした。
これは全ての点群が一つのリスト（ブランチやパスのような構造を持つことなく）に格納されてしまい、コンポーネントは、全ての点群を通るようにプログラムされていました。
もし、Grasshopper の 0.5 で、これを実行すると下記のような結果になります。

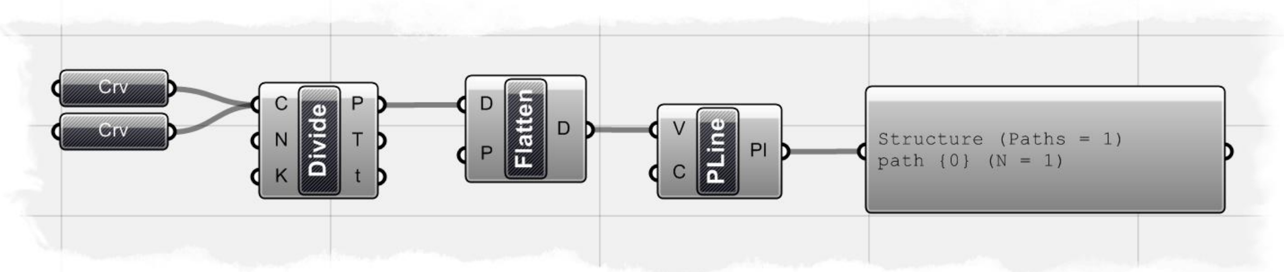


Grasshopper の、Version0.6.00XX を使用した場合は、2 つのパスがそれぞれ 20 のセグメントに分割されたと認識しますので、Polyline コンポーネントは、2 つのポリラインを作成します。Parameter Viewe コンポーネント(Params/Special/Param Viewer)を使用するとこのデータ構造を確認する事が出来ます。
下図はこのサンプルのスクリーンショットですが、2 つの独立したパス（0,0）と（0,1）から構成されていることが分かります。

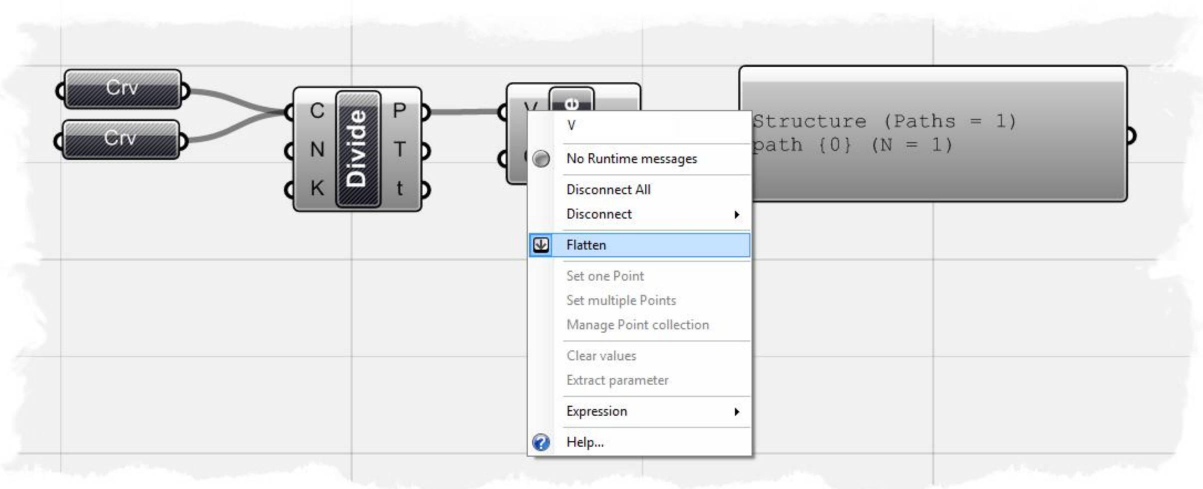
この 2 つのポリラインの点のインデックスと作成されるカーブは下図のようになります。



Grasshopper は、いくつかの新しい階層構造に関するコンポーネントが追加され、階層構造をコントロールすることが出来ます。これらのコンポーネントは+Logic タブ>Tree+に配置されています。
もしも、以前の Grasshopper のような 1 つのパスでポリラインを作成したい場合は、+Flattern Tree+コンポーネント(Logic>Tree>Flatten Tree)で、全ての階層構造を 1 つの平坦なリストに変換します。このコンポーネントを使用すると 0.5 以前の Grasshopper と同じ結果が得られます。
この GH 定義は下記のようになります。



また、もっと簡単に、Polyline コンポーネントの内部階層構造を同一階層のデータに変換する事も可能です。
これを行うには、Polyline コンポーネントの、+V-入力+を右クリックして、+Flattern+トグルを選択します。



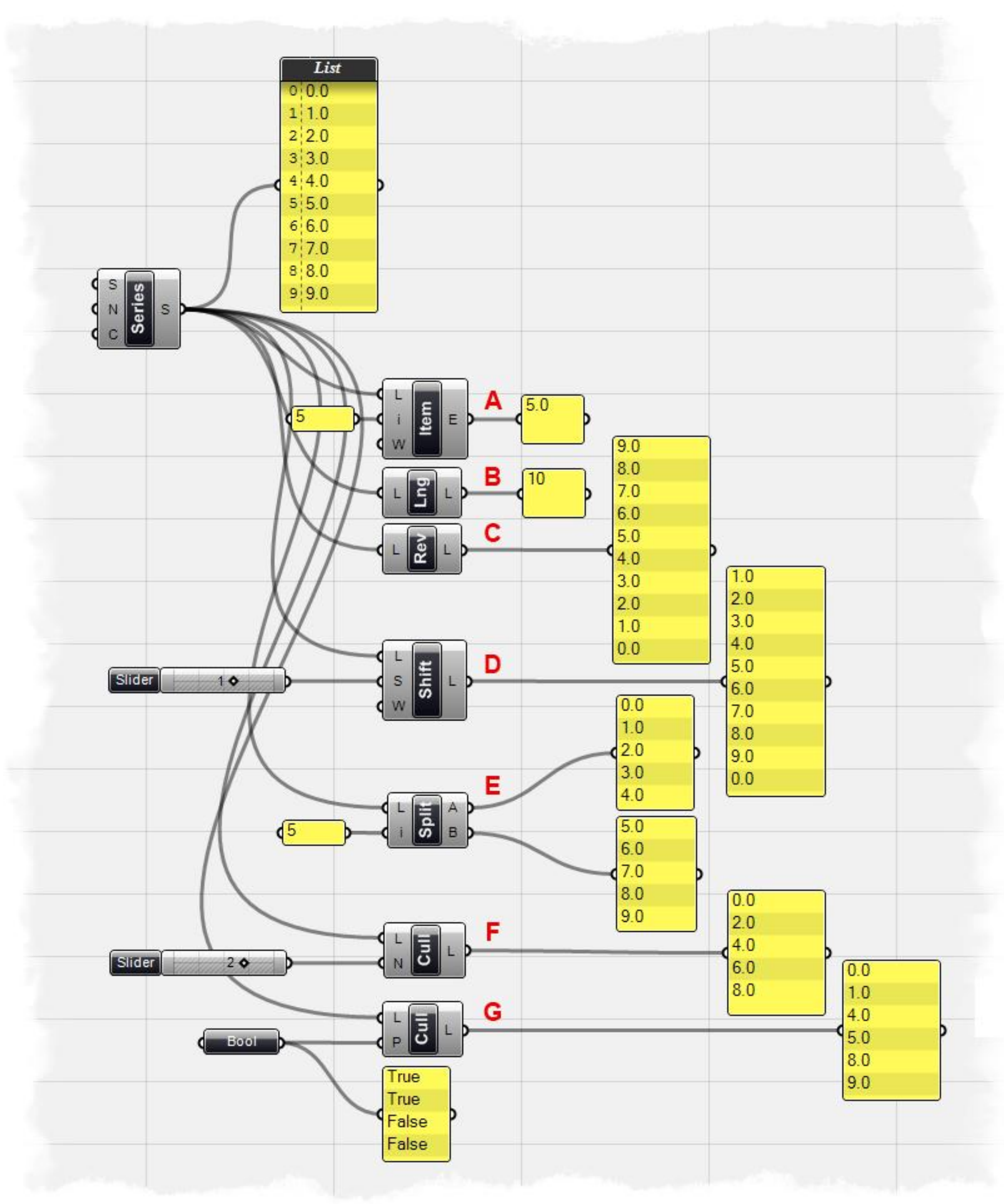
注：この最終結果は、**Curve Paths_base file.3dm** を開いた上で **Curve Paths.ghx** を開いて見ることが出来ます。

8.1 Lists & Data Management（リストとデータ管理）

Grasshopper では、グラフィックインターフェースが、情報の流れがどのコンポーネントに入出力しているか分かるようにデザインされているので、データフローの視点から考えることができます。

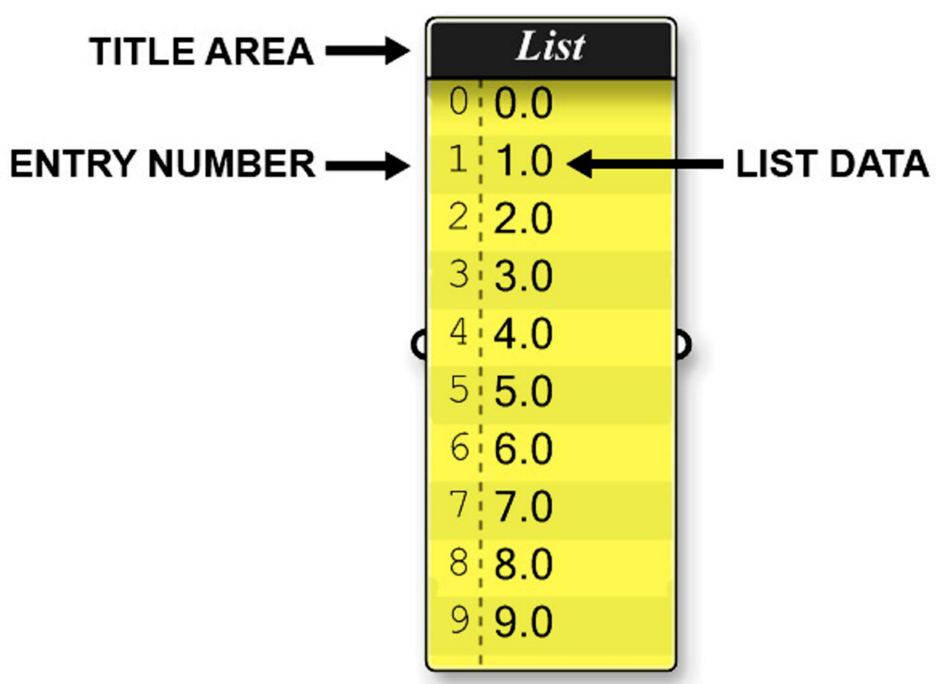
データ（点群、カーブ群、サーフェス群、ストリング、論理値、数値）がコンポーネントの情報の入出力の流れを定義しますので、リストデータの操作を理解することは Grasshopper を理解するうえで重要です。

次の例は、様々なコンポーネントによる数値データリストの操作の例です。
注：この最終結果を見るには、**List Management.ghx** を開いてください。



まず最初に+Series+コンポーネントを配置します。+Series+コンポーネントは初期値として、+0.0+から始まり、+1.0+間隔で、+9.0+まで、計 10 個の数値を持ちます。この出力を+Panel+パラメーターに接続すると内部の情報が確認できます。

注；+Panel+パラメーターの初期設定は、エントリー番号と、リストの値を表示するようになっています。



%Panel+パラメーターを右クリックするとコンテクストメニューで、+Entry Number+の表示・非表示が出来ます。ここでは、エントリー番号を表示しておきます。

A) %List Item+コンポーネント（Logic>List> List Item ）に与えられた数値データのうち、あるエントリーの数値を取得している様子を表しています。
リスト中の個々のアイテムにアクセスする際に、あるインデックスを必要とします。
最初のアイテムは、エントリー番号+0+に格納され、2 番目のアイテムは、エントリー番号+1+に格納されます。例えば、リストのインデックスを+5+とした場合、そのようなインデックスは存在し得ないのでエラーが発生します。
ここでは、+Series+コンポーネントの S-出力を、+List Item+コンポーネント L-入力に接続しています。
さらに、+List Item+コンポーネント i-入力に、どのエントリーの数値を取るかを定義するため整数を与えています。+5+と整数を与えると、この例では 5 番目のエントリーに格納されている数値、+5.0+を出力します。

B) %List Length+コンポーネント（Logic>List> Length）は、エントリーの数を判断します。
ここでは、+Series+コンポーネント S-出力が+List Length+コンポーネント L-入力に接続され、出力結果にエントリーの数+10+（整数）を表示しています。

C) %Reverse List+コンポーネント（Logic>List> Reverse List）は、リストの順番を反対に設定します。

ここでは、昇順で並んでいた数値のリスト入力を+9.0+から+0.0+の降順で出力しています。

D) %Shift+コンポーネント（Logic>List> Shift List）は、与えられたリストと数値の順番を、指定されたシフト・オフセットの値で、増減・加減し出力します。
ここでは+Series+コンポーネントの S-出力が+Shift+コンポーネントの L-入力に接続され、+Slider+パラメーターが S-入力に接続されています。ここで、+Slider+でシフト・オフセット値を整数値で、指定するとエントリー番号に対応する数値が、オフセットして増減・加減するのが分かります。

また W-入力（Wrap Value）で、マウスカーソルを右クリックし、+Set Boolean+値で論理値+を+True+または+False+に切り替えることが出来ます。
この例でシフト・オフセット値に、++1+を与え、W-入力（Wrap Vlue）で+論理値+を+True+にすると、最初のエントリーの数値が、最後のエントリーに移動します。もし、+論理値+を+False+にすると最初のエントリーはシフトアップされ、データセットから削除されます。

E) %Split List+コンポーネント（Logic>List> Split List）はその名前の通り、リストを 2 つの小さなリストに分割します。分割するためのインデックスを持ち、整数値で分割する箇所を指定します。この例では、5 番目以降のエントリー番号から分割し、A-出力には、0.0, 1.0, 2.0, 3.0, 4.0 が B-出力には 5.0, 6.0, 7.0, 8.0, 9.0.　のリストを出力します。

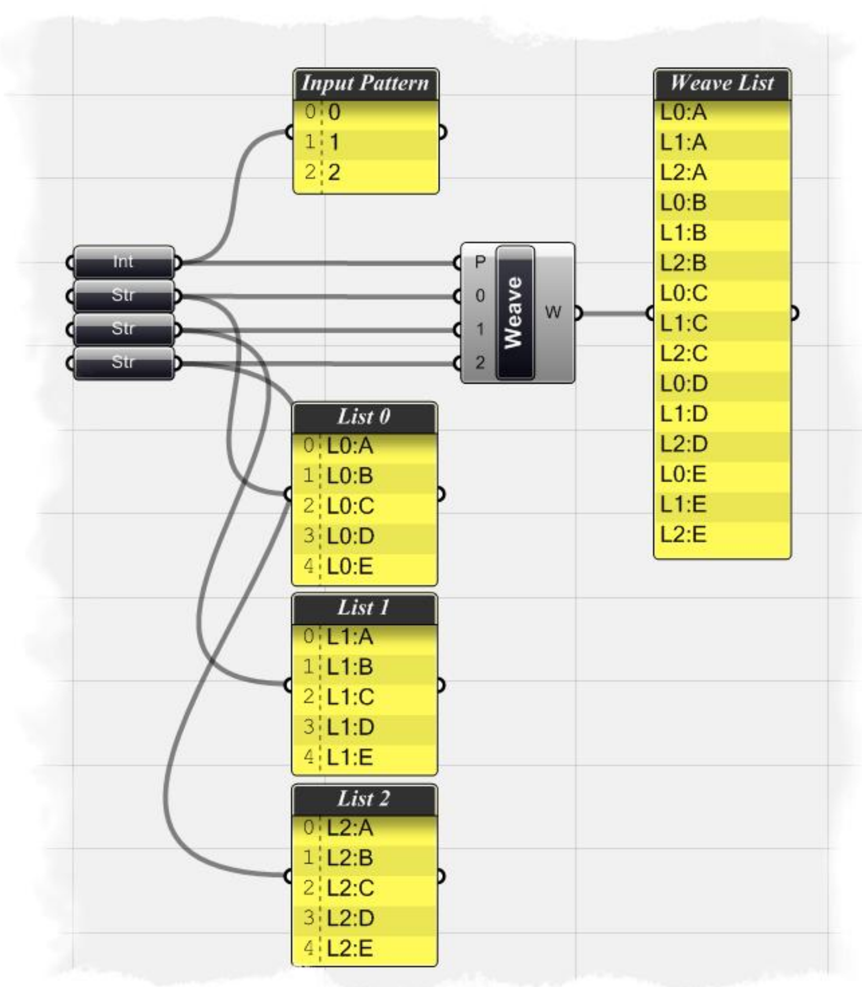
F) %Cull Nth+コンポーネント（Logic>Sets> Cull Nth）は、指定された整数値 N を N 倍した番号のデータエントリーを削除して、出力します。
%Slider+パラメーターに+2+を与えると、指定すると、入力されたデータリストから+0+、+2+、+4+、+6+、+8+、+10+番目のエントリーのデータセットを削除し、出力します。+3+を与えると、同様に+0+、+3+、+6+、+9+番目のエントリーのデータセットを削除します。

G) %Cull Pattern+コンポーネント（Logic>Sets> Cull Pattern）は、定義された値で削除する点で+**Cull Nth+**コンポーネントに似ています。
ここでは、数値の変わりに+論理値+のパターン配列を使用します。+True+値で、データは残り、+False+値で、データは削除されます。
この例では、+True+、+True+、+False+、+False+と論理値+のパターンをセットします。
10 のエントリーに対して、4 つの+論理値+が与えられていますが、このパターンは、エントリーの最後まで、繰り返されます。
結果として、3、4、7、8 番目のエントリーのデータセットが削除されます。

8.2 Weaving Data （データの振り分け）

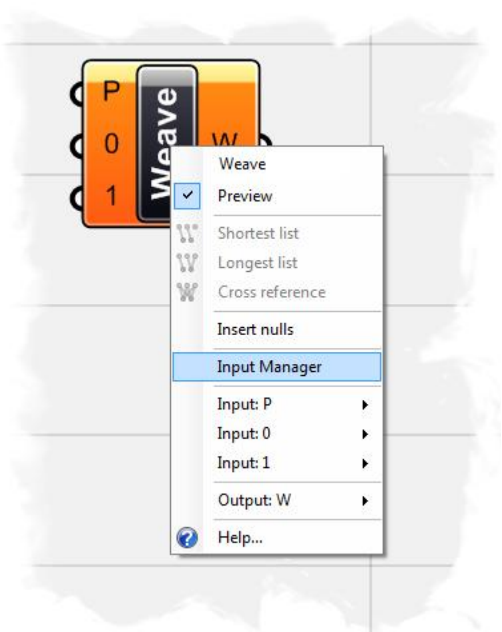
前の章では、GH でリストのマネージメントを行う幾つかのコンポーネントについて説明しましたが、**+Weave+**コンポーネント(Logics/Lists/Weave)でリストの順番をコントロールすることが出来ます。

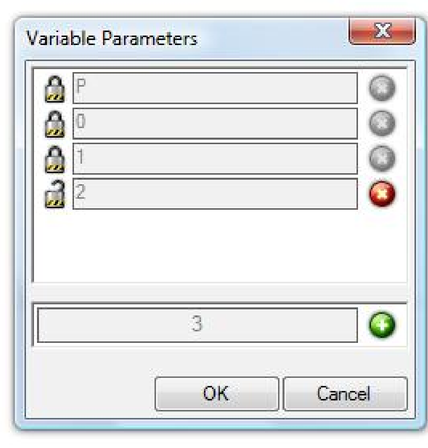
注：この最終結果を見るには、**Weave Pattern.ghx** を開いてください。



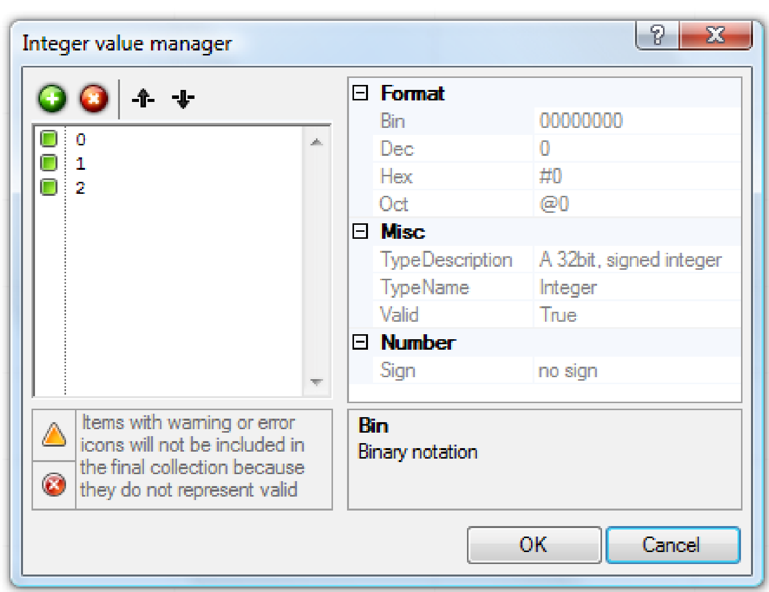
この GH 定義をスクラッチから作成するには:

- %Weave+コンポーネント（Logic/List/Weave）を、キャンバスにドラッグ＆ドロップします。このコンポーネントには 3 つの入力があるのが、分かります。まず、最初の P-入力で、どのようにデータを振り分けるかを指定します。次の 2 つの入力、ラベル-0 と、ラベル-1 は順番を入れ替えてリスト出力することが可能です。振り分けるリストが、2 つ以上有る場合は、+Weave+コンポーネント上でマウスを右クリックし、コンテクストメニューを表示し+Input Manager+を開いて、入力を増やす事が出来ます。+Input Manager+を開いたら、緑色の+追加ボタン+をクリックして別のリストを追加します。同様に赤色の+X ボタン+で、リストをコンポーネントから削除します。





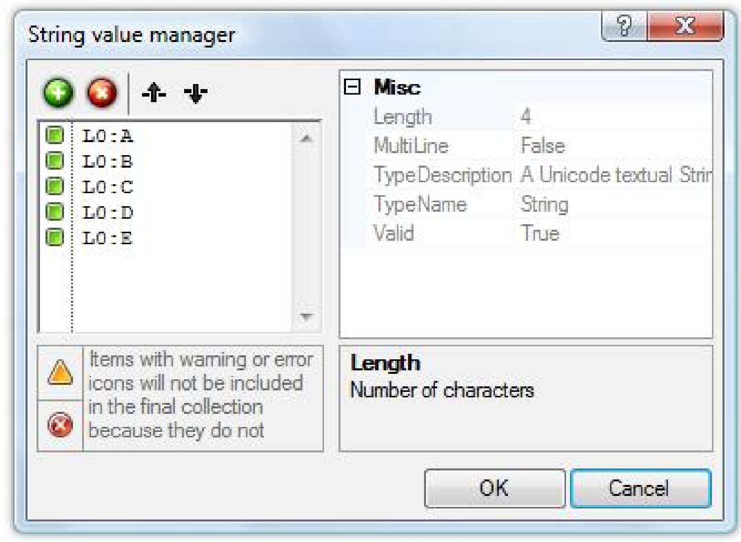
- %Input Manager+を開き、緑色の+追加ボタン+を選択し、コンポーネントに追加します。この時、+Input Manager+は上図のように表示されているはずです。
- %Integer+パラメーター（Params/Primitive/Integer）キャンバスにドラッグ＆ドロップします。
- %Integer+パラメーター上で、マウスを右クリックし、+Manage Integer Collection+ を選択します。
- ダイアログの上部にある緑色の+追加ボタン+をクリックし、このリストに整数値を追加出来ます。リストに 3 つ、数値を追加し、それぞれの値を 0, 1, 2. と編集すると下図のようになっているはずです。



この整数値のリストが、組み合わせのパターンを決定します。数値の順番を変えることによって出力されるデータセットも変わります。

- String+パラメーター（Params/Primitive/String）を、キャンバスにドラッグ＆ドロップします。
- %String+パラメーター上で、マウスを右クリックし、+Manage String Collection ‰ を選択します。

次に+Integer+パラメーター上で行った、+Manage Integer Collection+同様に、組み合わせたいストリングのリストを追加します。L0:A, L0:B, L0:C, L0:D, L0:E.のように 5 つのストリングを追加します。プレフィックスの L0 は、これらがリストの 0 番目であることを示します。この入力でストリングマネージャーは下記のようになっているはずです。



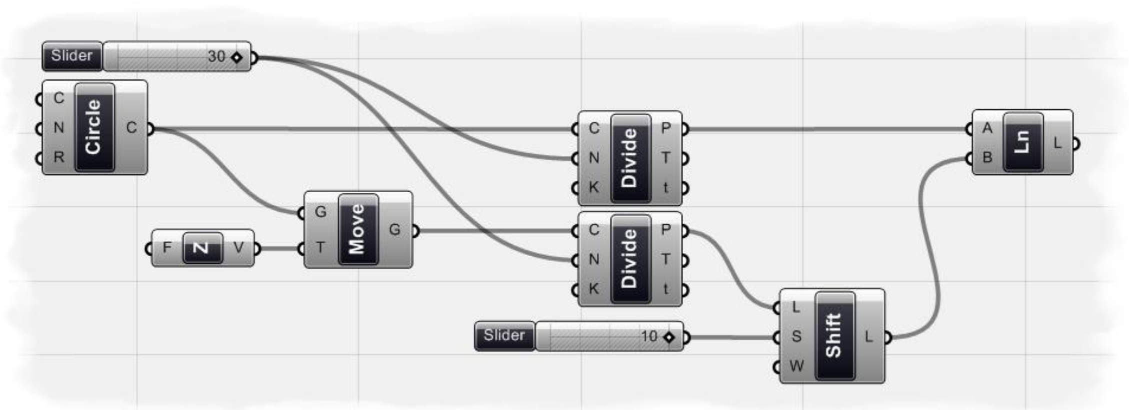
- %String+パラメーターを、選択しコピー（Ctrl+C）、ペースト（Ctrl+V）で後、2 つ、キャンバス上に作成します。
- 2 番目の+String+パラメーター上で、マウスを右クリックし+Manage String Collection‰ を選択し、リストが L1:A, L1:B, L1:C, L1:D, L1:E.なるように編集します。
- 同様に 3 番目の+String+パラメーターのリストが、L2:A, L2:B, L2:C, L2:D, L2:E になるように編集します。
- %Integer+パラメーターを、+Weave+コンポーネントの P-入力に接続します。
- 1 番目の+String+パラメーターを+Weave+コンポーネントの 0-入力に接続します。
- 2 番目の+String+パラメーターを+Weave+コンポーネントの 1-入力に接続します。
- 3 番目の+String+パラメーターを+Weave+コンポーネントの 2-入力に接続します。
- %Panel+パラメーター（Params/Special/Post-it Panel）を、キャンバスにドラッグ＆ドロップします。
- %Weave+コンポーネントの W-出力を+Panel+パラメーターに接続し、データの中が確認出来るようにします。

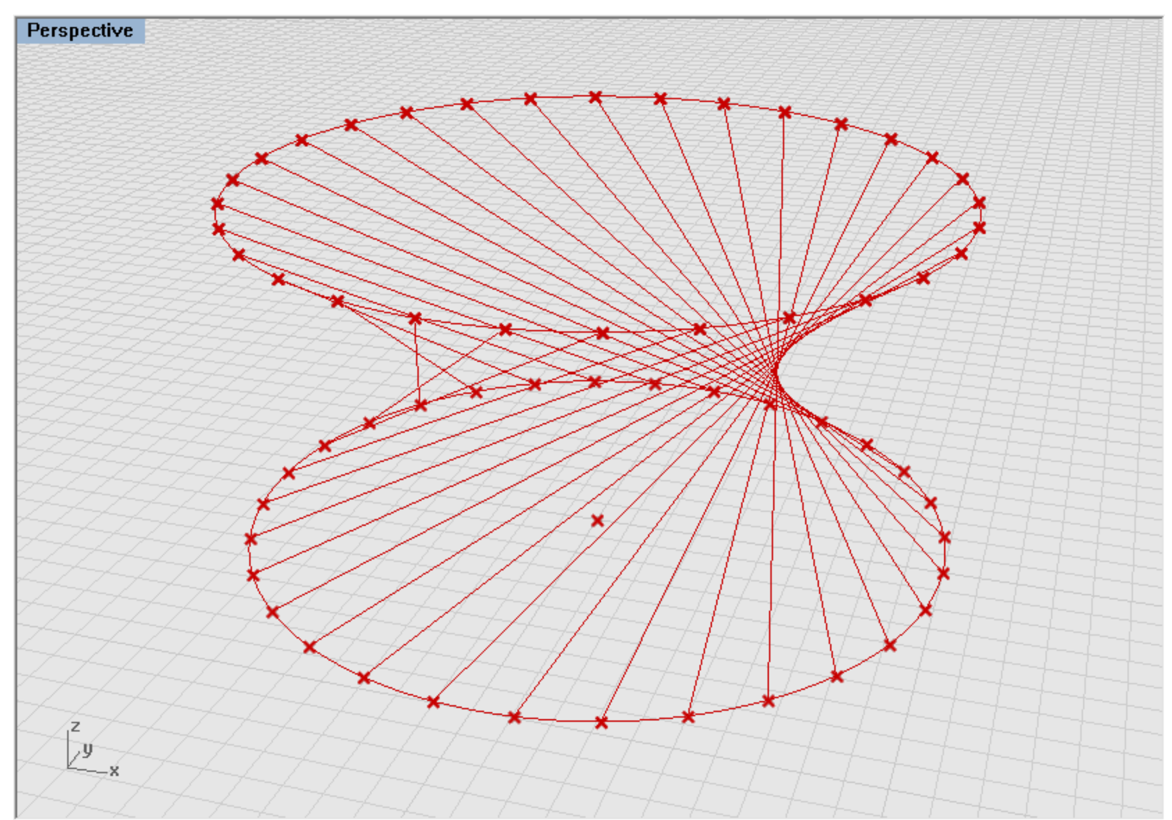
%Panel+パラメーターを見ると+Integer+パラメーターで指定した、整数のリストによって、3 つの+String+パラメーターのリストが振り分けられていることがわかります。
%Manage Integer Collection+ を編集してデータの順番を変わる事を確認してみてください。

8.3 Shiting Data（データのシフト）

前の章で、+Shift+コンポーネントによって、リスト中の全ての数値データを、シフトオフセット値によって、上下に移動する方法を見ました。
ここでは+Shift+コンポーネントを使用した、2 つの円をそれぞれ、当分割した点を直線で結び、その順番をシフトする例をみてみます。

注：この最終結果を見るには、**Shift Circle.ghx** を開いてください。





この GH 定義をスクラッチから作成するには

- %Circle CNR+コンポーネント（Curve>Primitive> Circle CNR）を、キャンバスにドラッグ＆ドロップします。
- %Circle CNR+コンポーネント C-入力で、+Set One Point+右クリックします。
- Rhino に入力が移り、コマンドプロンプトで数値を聞いてきますので、+0,0,0+と入力し、+Enter+キーを押します。
  これで、円の中心座標が定義されました。

- %Circle CNR+コンポーネント R-入力で右クリックし、コンテクストメニュー+Set Number+で+10.0+と入力します。
- %Unit Z+コンポーネント（Vector>Contant> Unit Z）を、キャンバスにドラッグ＆ドロップします。
- %Unit Z+コンポーネント F-入力で、右クリックし、コンテクストメニュー+Set Number+で+10.0+と入力します。
- %Move+コンポーネント（X Form> Euclidean > Move）を、キャンバスにドラッグ＆ドロップします。
- %Unit Z+コンポーネント V-出力を、+Move+コンポーネント T-入力に接続します。
- %Circle CNR+コンポーネント C-出力を、+Move+コンポーネント G-入力に接続します。

  ここまでの操作で、座標+0,0,0+を中心点とした、半径+10.0+単位の円を作成し、次に、+Move+コンポーネントによって、Z 方向に 10.0 移動したところに、オブジェクトをコピーしたことになります。
- %Divide Curve+コンポーネント（Curve> Division > Divide Curve）を、2 つ、キャンバスにドラッグ＆ドロップします。
- %Circle CNR+コンポーネント C-出力を、最初の+Divide Curve+コンポーネント C-入力に接続します。
- %Move+コンポーネントの G-出力を、2 番目の+Divide Curve+コンポーネント C-入力に接続します。
- %Number Slider+パラメーター（Params> Special > Number Slider）を、キャンバスにドラッグ＆ドロップします。
- %Number Slider+パラメーターを選択し、以下のパラメーター与えます。
  - Slider Type: Integers
  - Lower Limit: 1.0
  - Upper Limit: 30.0
  - Value: 30.0
- %Number Slider+パラメーターの出力を、2 つの+Divide Curve+コンポーネントの N-入力に接続します。

  ここで、それぞれの円上に、30 の点群がみえるはずです。
- %Shift List+コンポーネント（Logic> List > Shift List ）を、キャンバスにドラッグ＆ドロップします。
- %Divide Curve+コンポーネント P-出力を、2 番目の+Shift List+コンポーネント L-入力に接続します。

- %Number Slider+パラメーター（Params> Special > Number Slider）を、キャンバスにドラッグ＆ドロップします。
- 新しい+Number Slider+パラメーターを選択し、以下のパラメーターを与えます。
  - Slider Type: Integers
  - Lower Limit: 1.0
  - Upper Limit: 10.0
  - Value: 10.0

- %Number Slider+パラメーターの出力を+Shift List+コンポーネント S-入力に接続します。
- %Shift List+コンポーネントの W-入力（Wrap Values）を右クリックし、+論理値+を+True+に設定します。

  これで、上側の円上の点群のエントリーを 10 シフトしました。W-入力（Wrap Values）を+True+にすることによって、データエントリーの閉じたループを作りました。
- %Line+コンポーネントの（Curve> Primitive> Line）を、キャンバスにドラッグ＆ドロップします。

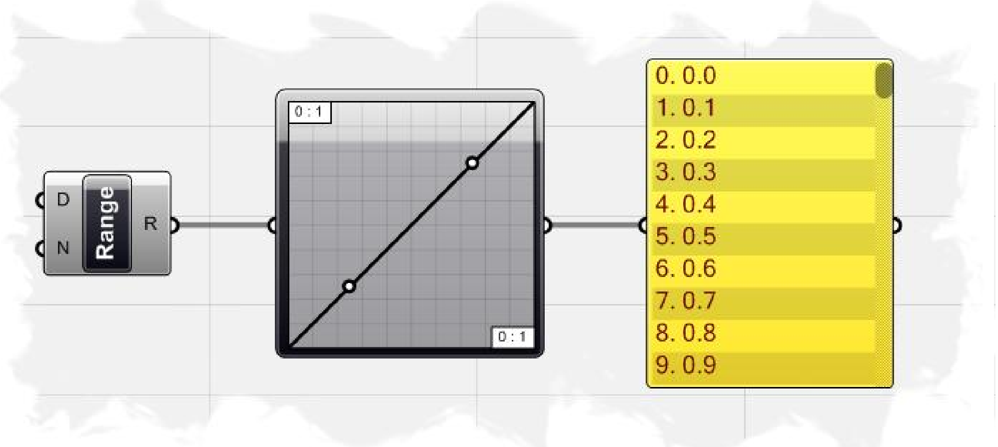
- 最初の+Divide Curve+コンポーネント P-出力を、+Line+コンポーネントの A-入力に接続します。
- %Shift　List+コンポーネントの L-出力を、+Line+コンポーネント B-入力に接続します。

  これで最初の点群と、シフトした点群間に直線を作成しました。スライダーの数値を変えることで、点群の数やオフセットの距離を返ることが出来ます。

8.4 Excel へのデータ出力

他のソフトウエアに Grasshopper からデータを出力し、さらなる解析をするようなことがあるかもしれません。

注：この最終結果を見るには、**Stream Contents_Excel.ghx** を開いてください。



まず、+Range+コンポーネント（Logic>Sets>Range）をキャンバスにドラッグ＆ドロップします。

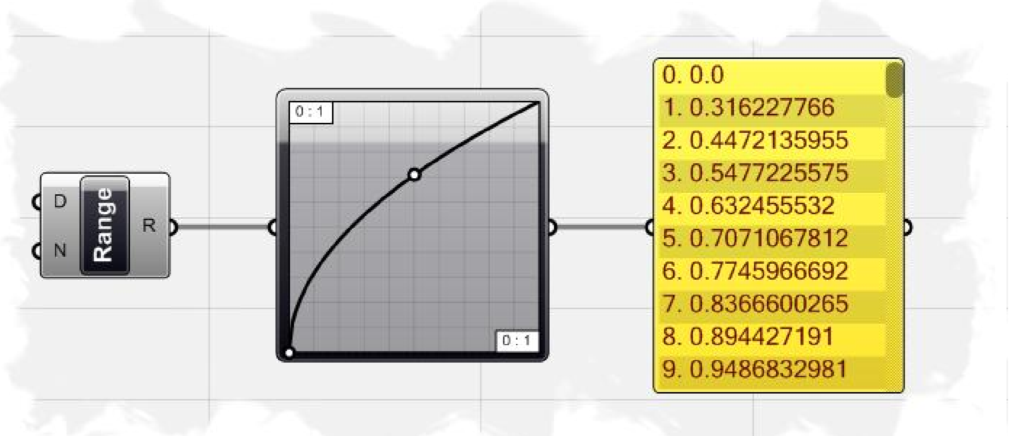
%Range+コンポーネントの D-入力を右クリックし、+Set one interval+で、数値ドメイン（数値範囲）を、+0.0+から+10.0+にセットします。（Rhino のコマンドプロンプトで入力します。）+Range+コンポーネントの N-入力を右クリックし+Set Integer+で数値を、+100+に設定します。
これで、+0.0+から+10.0+を均等に分割する 101 個の数値が出力されます。

次に、+Graph Mapper+パラメーター（Params>Special>Graph Mapper）をキャンバスにドラッグ＆ドロップします。

%Graph Mapper+パラメーターを右クリックし、コンテクストメニューで+Graph Types+を、Linear+にセットします。

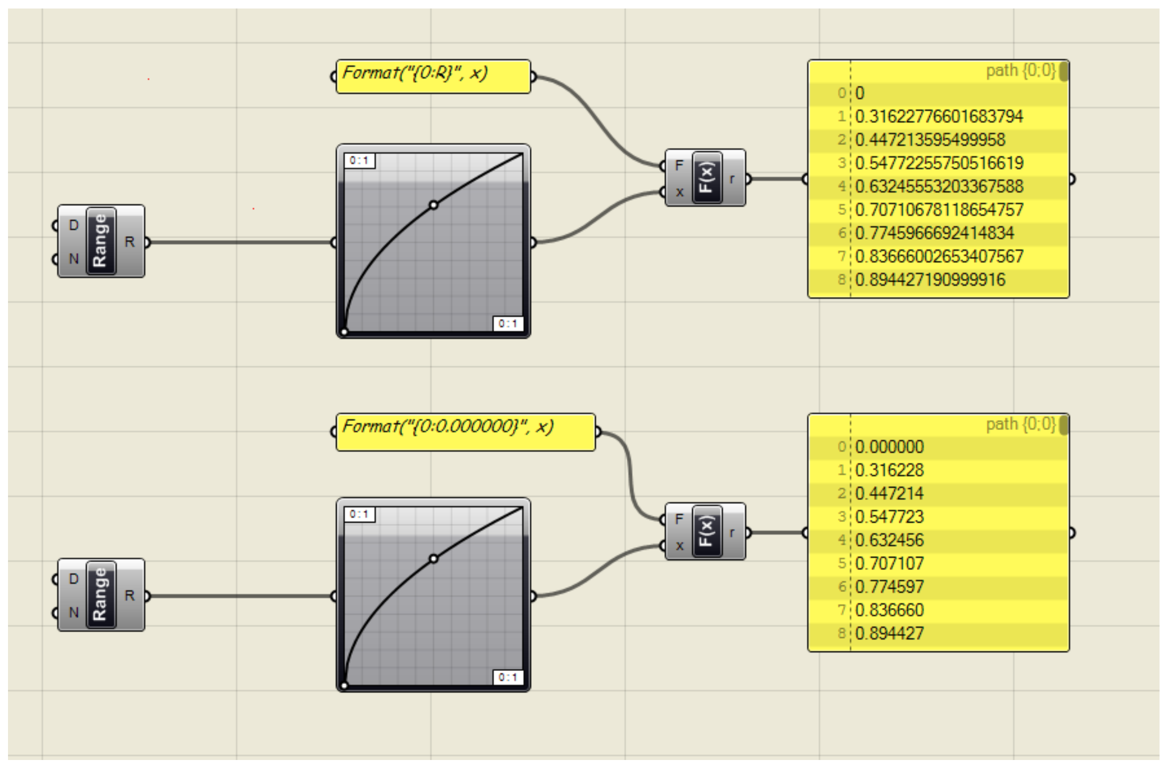
次に、+Range+コンポーネント R-出力を+Graph Mapper+パラメーター入力に接続します。

最後に、+Panel+パラメーター（Params>Special>Panel）をキャンバスにドラッグ＆ドロップし、+Graph Mapper+パラメーター出力を入力に接続します。
%Graph Mapper+パラメーターが、+リニア+に設定されているので、+Panel+パラメーターの表示は、+0.0+から+10.0+を線形に分割されています。+Graph Mapper+パラメーターで、+Graph Types+を+Square Root+にすると、データは対数表現のリストになるのが分かります。

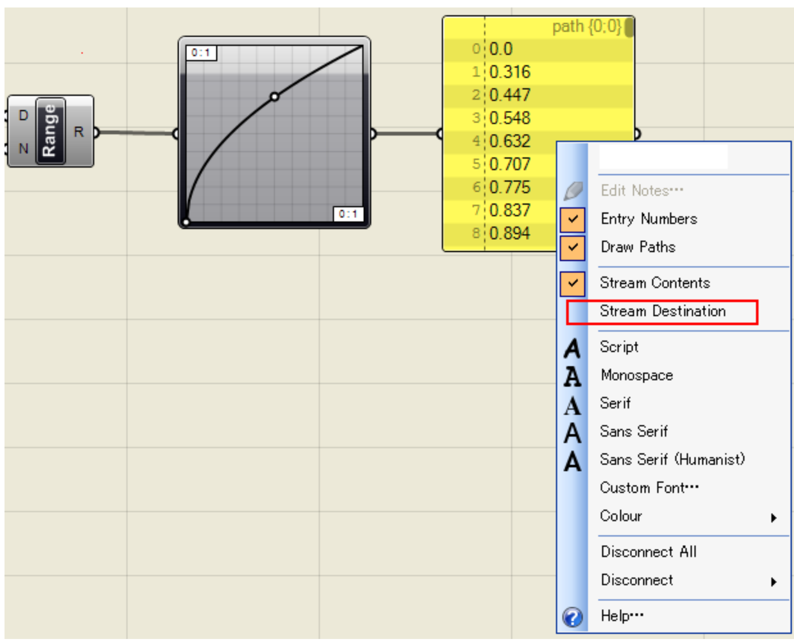


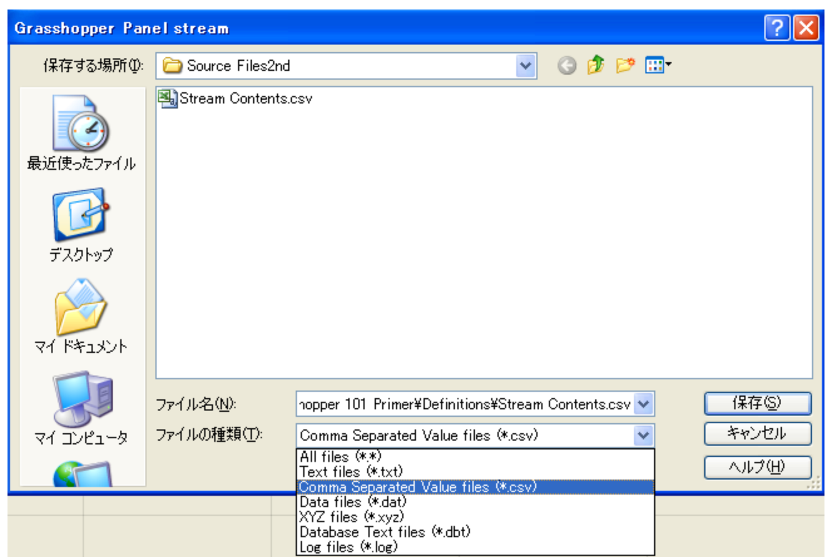
注；この記述は、古いバージョンのもので、Grasshopper 0.60012 では、+Panel+コンポーネントの表示の少数点以下の表示は、3 桁までとなっています。
浮動小数点や桁数指定で、必要な桁数を取るためには+GraphMapper+パラメーターと、+Panel+パラメーターの間に、+F1+コンポーネント（Logic>Script>F1）を介して行います。

%F1+コンポーネント F-入力に、フォーマットを指定するために、+Format(%{0:R}+; x)+
や、+Format(%{0:0.000000}+; x)+のように指定します。



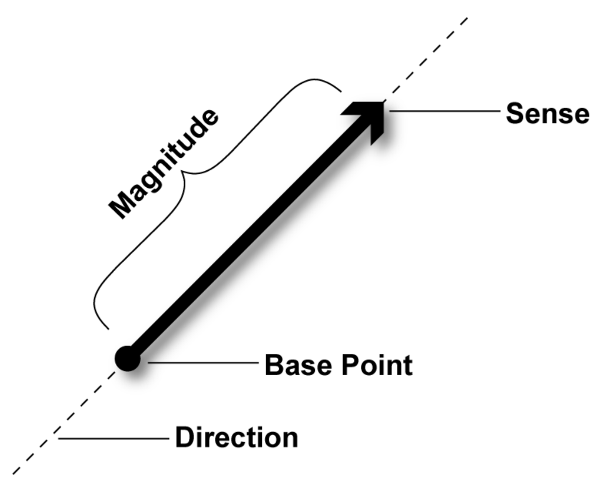
%Panel+パラメーターのリストデータを外部ファイルに出力するためには+Panel+パラメーターを右クリックして、コンテクストメニューを開き、+Stream Contents+にチェックを入れます。次に、+Stream Desitination+をクリックすると、保存するフォルダーの指定と、保存するファイル名、データタイプを指定するダイアログを立ち上がりますので、そこで指定します。指定できるデータタイプは、+.txt+、+.csv+、+.dat+、+.xyz+、+.dbt+、+log+です。



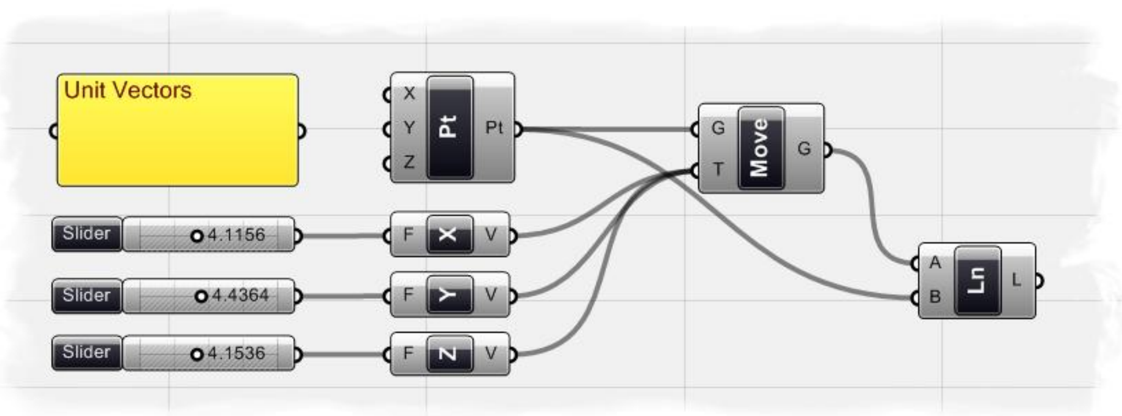


9 ベクトルの基礎

物理学的には、ベクトルは、大きさ（または長さ）、方向、sense (or orientation along the given direction)を持つジオメトリーです。
ベクトルは、始点+A+、終点+B+を結ぶ方向を持つラインセグメントやまたは矢印で表わされます。
ベクトルの大きさはセグメントの長さで、方向は+B+の+A+に対する相対で特徴付けられます。



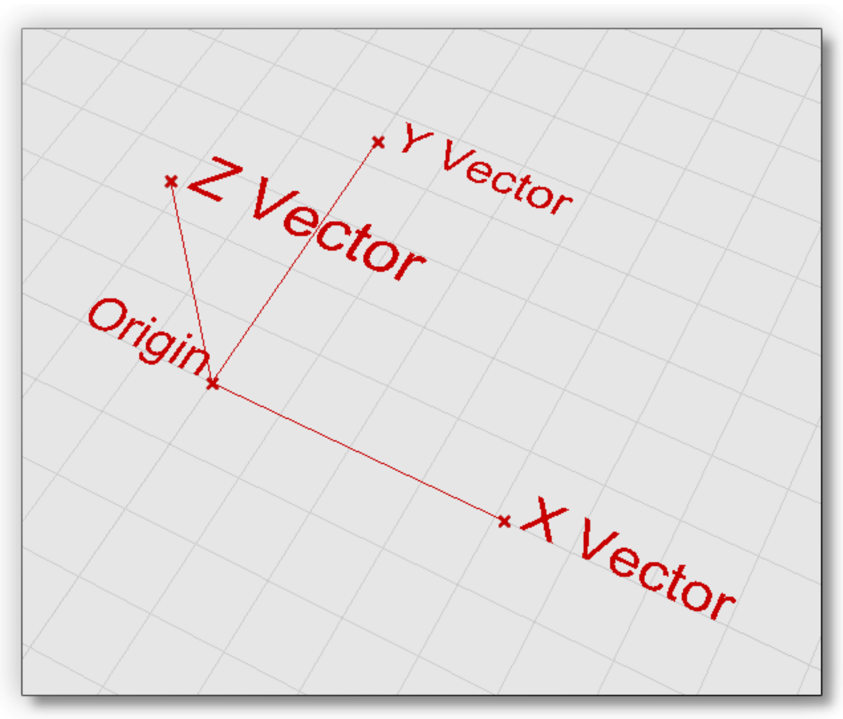
Rhino では、これらの点からベクトルを実際に見ることはできません。
2 つの点は、直交座標系の X,Y,Z 座標値の倍精度浮動小数点値で表わされます。
この違いは、点は、絶対値として扱われますがベクトルは相対的なものだからです。
例えば、3 つの倍精度値の配列を点として扱った場合、これを直交座標系のある空間座標を表わしますが、ベクトル配列として扱った場合、ある方向を表わします。
ベクトルは、始点から終点の違い矢印で示す、相対的なものとして扱われます。
すなわち、ベクトルは実体を持つジオメトリーではなく、情報に過ぎません。
これは、Rhino において、ベクトルを表わす、視覚的な手段はないことを示します。
しかしながら、ジオメトリーに対して、ベクトル情報を与えることで、移動、回転、反転を行うことができます。



上の例で見ると、
まず、+Point XYZ+コンポーネント（Vector>Point>Point XYZ）で原点(0,0,0)を作成します。そして+Point XYZ+コンポーネント Pt-出力を+Move+コンポーネント G-入力に接続し、あるベクトル方向に移動しコピーします。
これを行うには、+Unit X+コンポーネント、+Unit Y+コンポーネント、+Unit Z+コンポーネント（それぞれ、Vector>Constant>メニュー上）をキャンバス上に配置します。

これらのコポーネントは、それぞれベクトル方向を、X,Y,Z 方向へ指定します。
次に、それぞれの+Unit ベクトル+コンポーネント入力に、+Number Slider+パラメーターを接続し、ベクトルの大きさを決めます。

それぞれの、+Unit ベクトル+コンポーネント出力を+Move+コンポーネント T-入力に接続します。この時点で、Rhino ビューポートには、原点と、3 つの新しい点がそれぞれ、X,Y,Z 軸上に配置されているでしょう。
%Slider+パラメーターで、それぞれ移動させてみてください。
%Line+コンポーネント（Vector> Primitive >Line）をキャンバスに配置し、+Move+コンポーネント G-出力を、A-入力に、+Point XYZ+コンポーネント Pt-出力を、B-入力に接続すると、ラインが作成され、ベクトルが視覚化されます。



注：この最終結果を見るには、**Unit Vectors.ghx** を開いてください。

9.1 Point/Vector Manipulation（点・ベクトルの操作）

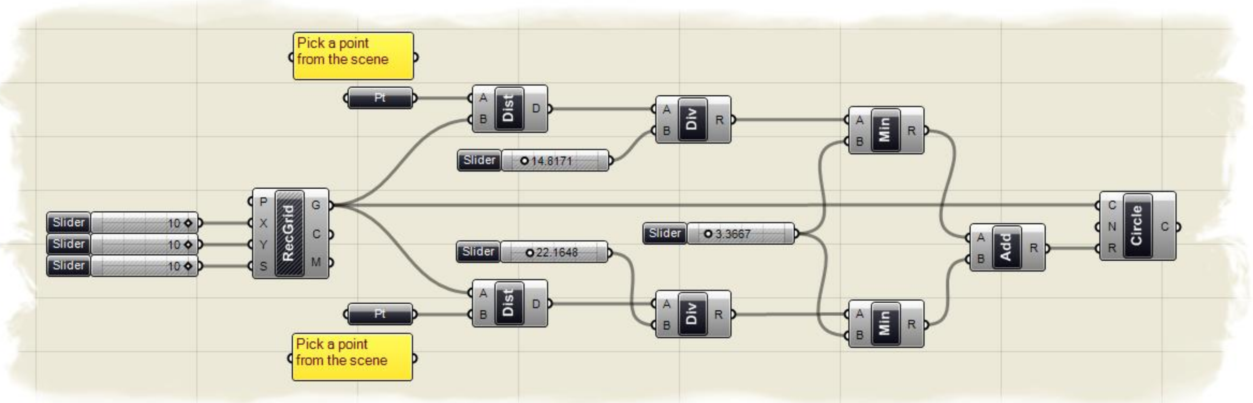
Grasshopper には、基本的なベクトル演算を行うための多数の+Point/Vector+コンポーネントがあります。以下のものは、最も一般的に使用されるものです。

Component	メニュー	概要	Example
A B Dist D	Vector/Point/**Distance**	2 点間、A,B の距離測定	d p r
P pComp X Y Z	Vector/Point/**Decompose**	点データの、座標を X, Y, Z　値に分解	p {x,y,z}
A B Angle A R	Vector/Vector/**Angle**	2 つのベクトルの角度計算、出力はラジアン	w α v
V VLen L	Vector/Vector/**Length**	ベクトルの大きさ（長さ）を計算	l v
V vComp X Y Z	Vector/Vector/**Decompose**	ベクトルデータを 3 つの数値に分解	{x,y,z} v
A B U Sum V L	Vector/Vector/**Summation**	入力ベクトル A、入力ベクトル B を合成	v u w
A B U Vec2Pt V L	Vector/Vector/**Vector2pt**	2 つの点からベクトルを定義	p v r
V Rev V	Vector/Vector/**Reverse**	ベクトル方向を反転、 ベクトルの大きさは保持される。	v w
V Unit V	Vector/Vector/**Unit Vector**	入力される全てのベクトルを単位ベクトル（大きさ=1）に変換	v w
V F Mul V L	Vector/Vector/**Multiply**	ベクトルの大きさを与えられた数値で拡大	w v {a=4.0}

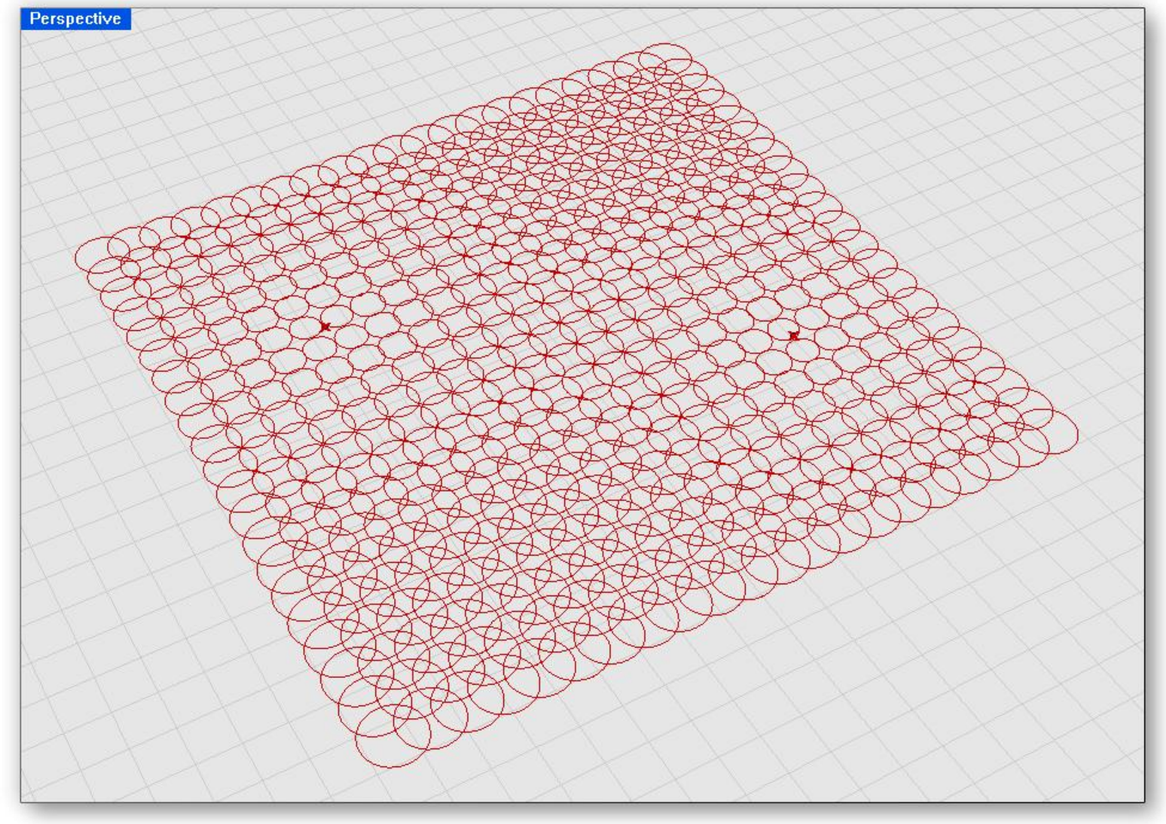
9.2 Using Vector/Scalar Mathematics with Point Attractors (Scaling Circles)

（アトラクターポイントを使用したベクトル・スカラー演算：円のスケール）

スカラーとベクトル演算の基礎を理解したところで、格子状に配置された円の半径を、ある点からの距離によって、スケーリング（拡大・縮小）を行う例を見ます。



注：この最終結果を見るには、**Attractor_2pt_circles.ghx** を開いてください。



この GH 定義をスクラッチから作成するには、

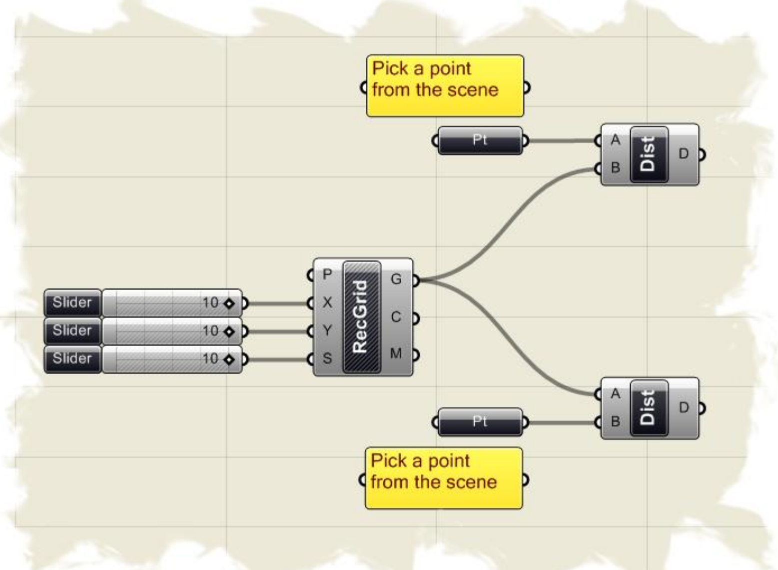
- %Number Slider+パラメーター（Params>Special>Number Slider）を 3 つ、キャンバスにドラッグ＆ドロップします。
- %Slider+に、以下のパラメーターを定義します。
    - Slider Type: Integers
    - Lower Limit: 0.0
    - Upper Limit: 10.0
    - Value: 10.0
- %Grid Rectangular+コンポーネント（Vector>Point> Grid Rectangular）をキャンバスにドラッグ＆ドロップします。
- 最初の+Slider+を+Grid Rectangular+パラメーターX-入力に接続します。
- 2 番目の+Slider+を+Grid Rectangular+パラメーターY-入力に接続します。
- 3 番目の+Slider+を+Grid Rectangular+パラメーターS-入力に接続します

    %Grid Rectangular+パラメーターは、点群を格子状に作成します。P-入力を原点とし（ここでは、0,0,0 を使用）、X 方向、Y 方向の数を+Slider+パラメーターで指定し、作成する格子点の数を指定します。ここで X,Y 方向の数を+10+と指定すると、作成される格子点は、それぞれの方向に 20 個作成されます。これは指定する数は原点を中心に、それぞれの方向にいくつ作成するかという指定になるからです。S-入力は、格子間隔を指定します。
- %Point+パラメーター（Params>Geometry>Point）を、2 つ、キャンバスにドラッグ＆ドロップします。

    このパラメーターは、永続性のあるデータとなります。（詳しくは、第 4 章を参照）このパラメーターは、+Point XYZ+コンポーネントとは異なり、シーンから直接、点を割り当てなければポイントを作成しません。したがって、ここで点を割り当てるためには、Rhino のシーン上に、既に点オブジェクトが存在している必要があります。これらの点が、この GH 定義で使用する ***attractor point*** *（引き寄せる点）*になります
- Rhino のシーンにおいて、点オブジェクトを 2 つ、任意の場所に作成してください。

    これで、Rhino 上に 2 つの attractors points が定義されましたので、Grasshopper 上の 2 つの+Point+パラメーターにこの点を割り当てます。
- %Point+パラメーターを右クリックし、コンテクストメニューで+Set one Point+を選択します。
- Rhino ビューに入力が変わりますので、先に作成した点を選択します。
- 残りの+Point+パラメーターについても同様のステップで、もう一つの点を割り当てます。

    これで、格子点を作成し、2 つの、attractor point を Grasshopper のデータとして割り当てました。
- %Dist+コンポーネント（Vector>Point>Distance）を、2 つ、キャンバスにドラッグ＆ドロップします。
- 最初の attractor point を 1 つめの+Dist+コンポーネント A-入力に接続します。
- %Grid Rectangular+コンポーネント G-出力を 1 つめの+Dist+コンポーネント B-入力に接続します。
- 2 つめの attractor point を 2 つめの+Dist+コンポーネント B-入力に接続します。
- %Grid Rectangular+コンポーネント G-出力を、2 つめの+Dist+コンポーネント A-入力に接続します。



*最初の定義のステップは上記のようになっているはずです。マウスカーソル*を+Dist+コンポーネント D-出力の上に持ってくると、格子点が attractor point からどれだけ距離が離れているかの数値のリストが見えるはずです。
これらの数値を後で、円の半径を決めるのに使用しますが、最初にこれらの数値をスケールダウンさせ、適当な半径寸法を与えるようにします。

- %Division+コンポーネント（Scalar>Operators>Division）を 2 つキャンバス上にドラッグ＆ドロップします。
- 最初の+Dist+コンポーネント D-出力を、最初の+Division+コンポーネント A-入力に接続します。
- 2 つめの+Dist+コンポーネント D-出力を、2 つめの+Division+コンポーネント A-入力に接続します。
- %Number Slider+パラメーター（Params/ Special > Number Slider）を、2 つキャンバスのドラッグ＆ドロップします。
- １つめの+Number Slider+パラメーターを右クリックし次のようにセットします。
  - Name: Pt1 Influence
  - Slider Type: Floating Point
  - Lower Limit: 0.0
  - Upper Limit: 100.0
  - Value: 25.0
- １つめの+Number Slider+パラメーターを右クリックし、次のようにセットします。
  - Name: Pt2 Influence
  - Slider Type: Floating Point
  - Lower Limit: 0.0
  - Upper Limit: 100.0
  - Value: 25.0
- %Pt1 Influence+の+Number Slider+パラメーターを 1 つめの+Division+コンポーネント B-入力に接続します。
- %Pt2 Influence+の+Number Slider+パラメーターを 2 つめの+Division+コンポーネント B-入力に接続します。

Dist+コンポーネントから出力されるデータは、attractor point と、441 個の格子点との距離となりますので、かなり大きな値もあります。
後にこの値を円の半径入力に使用しますが、このままの値では大きすぎるので、スライダーで決定した値、距離を割った数値を後のコンポーネントへ渡します。その

ため、スライダーの範囲は大きめに取ってあります。円を作成するコンポーネントに直接、数値を与えることも、もちろん出来ますが、この GH 定義をさらに強固なものにするために、ここではスカラー演算を使用して半径の最小値を決定することにします。最小の半径の指定方法について、以下に述べます。

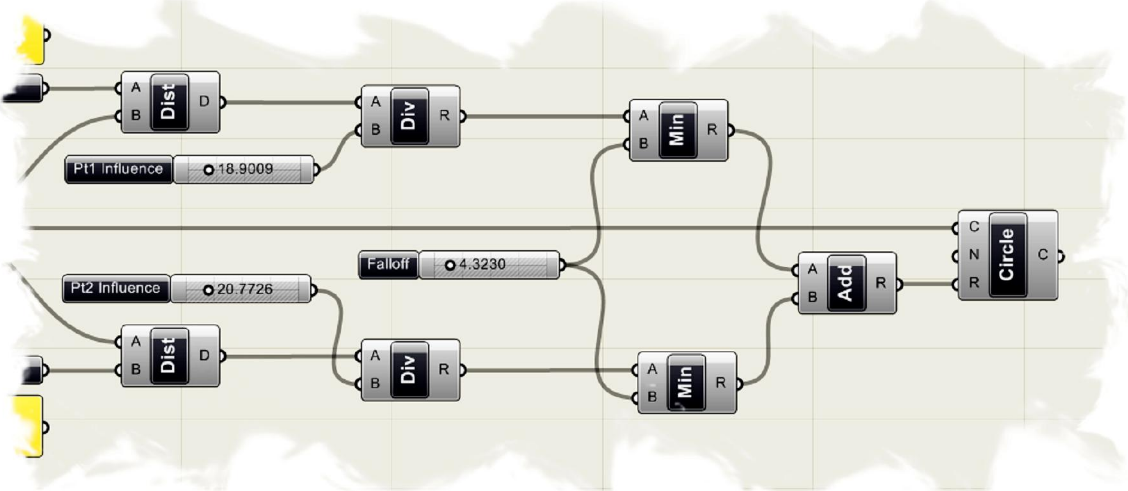
- %Minimum+コンポーネントの（Scalar> Utility> Minimum）を 2 つキャンバスにドラッグ＆ドロップします。
- 最初の+Division+コンポーネント R-出力を最初の+Minimum+コンポーネント A-入力に接続します。
- 2 番目の+Division+コンポーネント R-出力を、2 番目の+Minimum+コンポーネント A-入力に接続します。
- %Number Slider+パラメーター（Params>Special > Number Slider）を、キャンバスのドラッグ＆ドロップします。
- %Number Slider+パラメーターを右クリックし、次のようにセットします。
  - Name: Falloff
  - Slider Type: Floating Point
  - Lower Limit: 0.0
  - Upper Limit: 30.0
  - Value: 5.0
- %Falloff+パラメーターを、2 つの+Minimum+コンポーネント B-入力に接続します。

これで残されたことは、作成された 2 つのリストを合わせ、それぞれの円の半径を定義する 1 つのリストを作成することです。

- %Addition+コンポーネント（Scalar> Operators> Addition）をキャンバスにドラッグ＆ドロップします。
- 最初の+Minimum+コンポーネント R-出力を+Addition+コンポーネント A-入力に接続します。
- 2 番目の+Minimum+コンポーネント R-出力を+Addition+コンポーネント B-入力に接続します。
- %Circle　CNR+コンポーネント（Curve/> Primitive > Circle CNR）をキャンバスにドラッグ＆ドロップします。

ここで、最初に作成した格子点が円の中心になるようにします。

- %Grid Rectangular+コンポーネント G-出力を、+Circle　CNR+コンポーネント C-入力に接続します。
- 次に、+Addition+コンポーネント R-出力を+Circle　CNR+コンポーネント R-入力に接続します。
- %Grid Rectangular+コンポーネントを右クリックし、コンテクストメニューで+Preview+をオフにします。



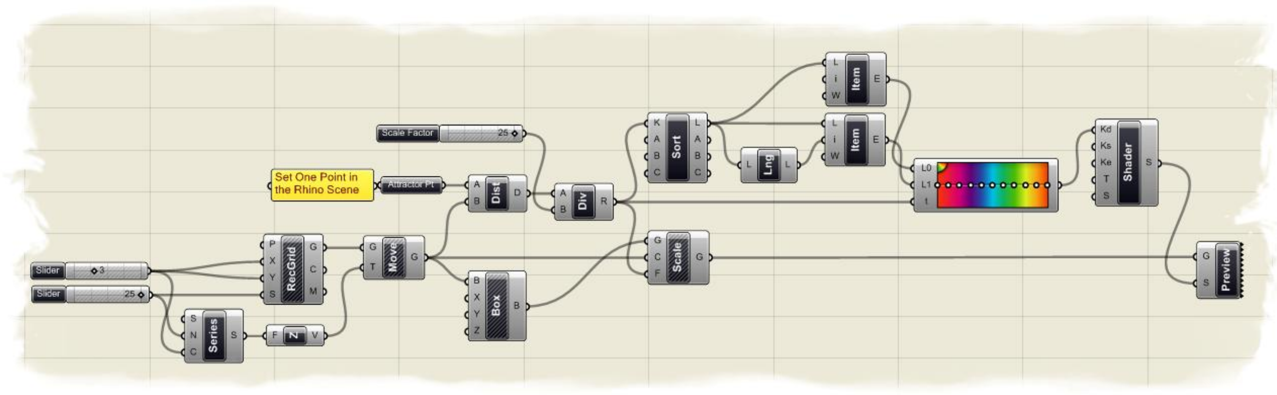
注；このビデオチュートリアルは、David Fano 氏のブログで見ることができます。
http://designreform.net/2008/07/08/grasshopper-patterning-with-2-attractor-points/

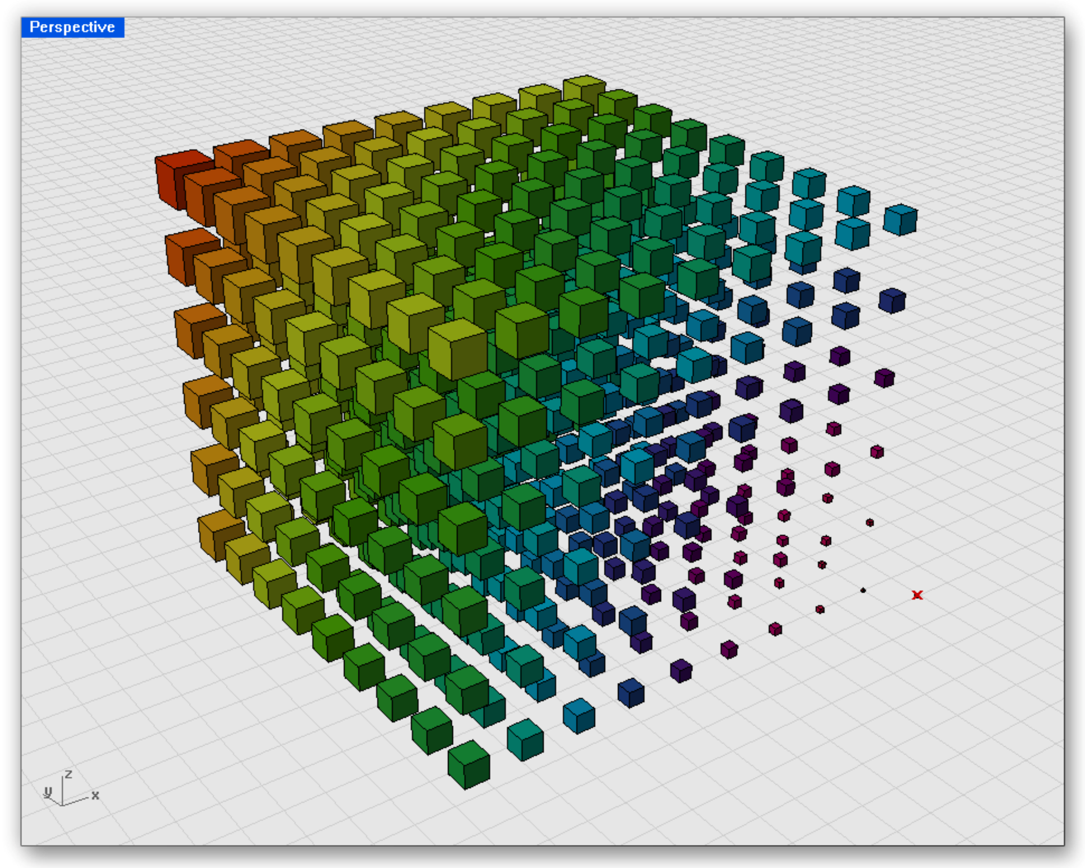
9.3 Using Vector/Scalar Mathematics with Point Attractors (Scaling Boxes)

（アトラクターポイントを使用したベクトル・スカラー演算：ボックスのスケール）

先の章では、点オブジェクトを参照し、ベクトルとスカラー演算で円の半径を決定しました。この主なコンポーネントを流用し、+Grasshopper Shader+コンポーネントを追加使用して、オブジェクトの色設定をしてみます。
次の図が最終の GH 定義です。

注：この最終結果を見るには、**Color Boxes.ghx** を開いてください。





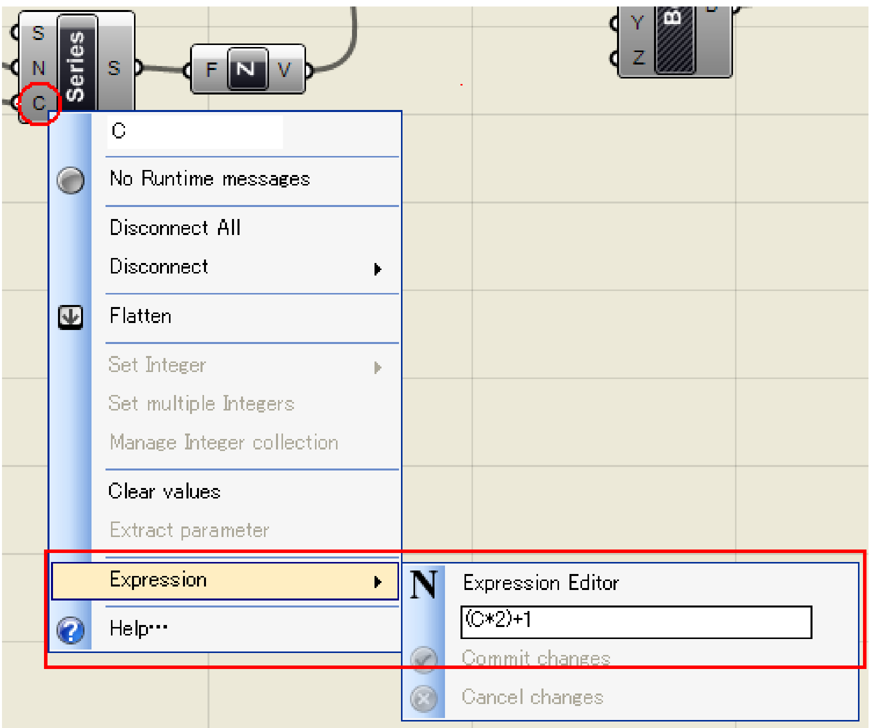
ステップ１：3 次元格子点の作成

- %Number Slider+パラメーター（Params>Special>Number Slider）を 2 つ、キャンバスにドラッグ＆ドロップします。
- 最初の+Slider+を右クリックし、以下のパラメーターを定義します。
  - Slider Type: Integers
  - Lower Limit: 0.0
  - Upper Limit: 10.0
  - Value: 3.0
- 2 つめの+Slider+を右クリックし、以下のパラメーターを定義します。
  - Slider Type: Integers
  - Lower Limit: 0.0
  - Upper Limit: 25.0
  - Value: 25.0
- %Grid Rectangular+コンポーネント（Vector>Point> Grid Rectangular）をキャンバスにドラッグ＆ドロップします。
- 最初の+Slider+を+Grid Rectangular+パラメーターX-入力、Y-入力に接続します。
- 2 番目の+Slider+を+Grid Rectangular+パラメーターS-入力に接続します。

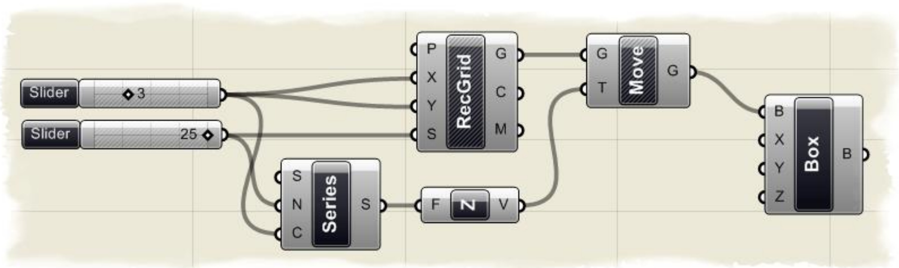
  ここで、Rhino のビューポートにおいて、XY 方向に格子点が見えるはずです。（注：もし格子点が見えない場合は、+Grid Rectangular+パラメーターを右クリックし、コンテクストメニューで、+Preview+をオンにします。）格子点の距離は、2 番目の+Slider+でコントロールされます。次に格子点を 3 次元的にするために Z 方向にコピーします。
- %Series+コンポーネント（Logic>Set> Series）をキャンバスにドラッグ＆ドロップします。
- 2 番目の+Slider+を+Series+コンポーネント N-入力に接続します。
- 最初の+Slider+を+Series+コンポーネント C-入力に接続します。

  Series+コンポーネントは Z 軸方向にコピーする格子点の数をカウントします。
  ここで+小さな+間違い+が、この数式のなかに見つけているかもしれません。前述したようにこの格子点は中心点からの数となります。
  従って、X,Y の格子点の数を+3+としたとき、それぞれに方向に中心から、プラス各+3+、計 7 つの点を持ちます。我々は、3 次元の立方体を作成したいので Z 方向にも 7 つ配置される必要があります。ここでカウントの統一性を持つために、+Series+コンポーネントが受け取る際に数が+2 倍+1+になるように指示する必要があります。
- %Series+コンポーネント C-入力を右クリックし、コンテクストメニューを出し+Expression+タブまで、スクロールダウンします。
- %Expression editor+で、数式を+**(C*2)+1**+と入力します。

  これで出力の数が、+7+になりました。



- %Unit Z+コンポーネント（Vector>Point>Distance）をキャンバスにドラッグ＆ドロップします。
- %Series+コンポーネント S-出力を+Unit Z+コンポーネント F-入力に接続します。
  このとき、Unit Z+コンポーネント V-出力上に、マウスカーソルを持っていくと、Z 値において+25+（2 つめの+Slider+で指定された数値）間隔で増加していく 7 個の数値が確認出来るはずです。このベクトル値を利用して格子間隔を決定します。
- %Move+コンポーネント（X Form> Euclidean > Move）をキャンバスにドラッグ＆ドロップします。
- %Grid Rectangular+コンポーネント G-出力を+Move+コンポーネント G-入力に接続します。
- %Unit Z+コンポーネント出力を+Move+コンポーネント T-入力に接続します。
  このとき、Rhino ビューポートを見ると、点群が、3 次元の立方体のように配置されていないかもしれません。
  これは、コンポーネントのデータマッチングアルゴリズムが、+Longest List+.になっているからです。
  コンポーネントを右クリックして、コンテクストメニューでデータマッチングを+Cross Reference+.に変更してみてください。
  格子点が 3 次元的の立方体のように配列していることが確認できます。
- %Center Box+コンポーネント（Surface>Primitive > Center Box）をキャンバスにドラッグ＆ドロップします。
- %Move+コンポーネント G-出力を+Center Box+コンポーネント B-入力に接続します。
- %Center Box+コンポーネントを右クリックして、コンテクストメニューで、+Preview+オフにします。（初期状態では、作成された格子点に、X、Y、Z 値が、それぞれ+1.0+の値が入っており、各辺長さ+1.0+のボックスが表示されています。）今までのセットアップで下記のようになっているはずです。


Slider
3
Slider
25
P
X
Y
S
RecGrid
G
C
M
G
T
Move
G
B
X
Y
Z
Box
B
S
N
C
Series
S
F
Z
V

ステップ 2：スカラーとベクトル演算

- %Point+パラメーター（Params>Geometry>Point）をキャンバスにドラッグ＆ドロップします。
- %Point+パラメーターを右クリックして表示名を+Attractor Pt+とします。

  円のスケーリングの例のように Rhino ビューポート上で、+Attractor Pt+を定義する必要があります。まず、Rhino 上で点を作成する必要があります。
- Rhino ビューポート上で任意の位置に点オブジェクトを作成してください。
- Grasshopper に戻り、+Attractor Pt+パラメーターを右クリックし+Set one Point+を選択します。
- すると、Rhino ビューポートに入力が移りますので、先に作成した点オブジェクトを選択します。

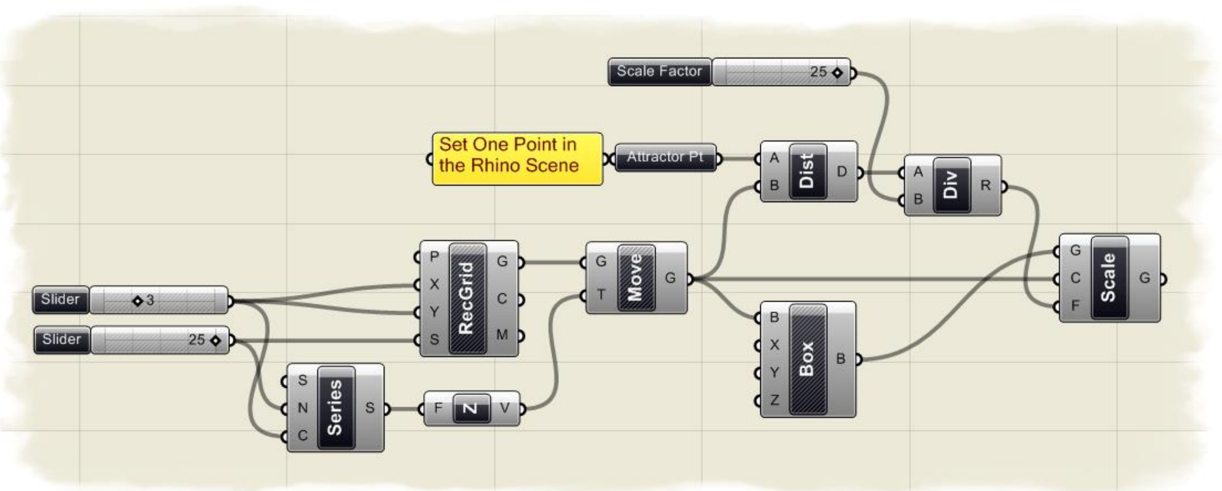
  Rhino ビューポート上で小さな赤い+X+（Grasshopper で定義された点）が表示されているはずです。

  この状態で、Rhino の点オブジェクトを移動すると、Grasshopper で定義された点もアップデートされ移動します。
- %Distance+コンポーネント（Vector > Point/> Distance）をキャンバスにドラッグ＆ドロップします。
- Attractor Pt+を、+Distance+コンポーネント A-入力に接続します。
- %Move+コンポーネント G-出力を+Distance+コンポーネント+B-入力に接続します。

  この時、マウスカーソルを Distance+コンポーネント+D-出力にもっていくと+Attractor Pt+から格子点までの距離が、出力されているのが分かるはずです。

  この値をスケールのファクターとして使用するには、さらに値を小さくする必要があります。
- %Division+コンポーネント（Scalar> Operator> Division）をキャンバスにドラッグ＆ドロップします。
- %Distance+コンポーネント D-出力を+Division+コンポーネント A-入力に接続します。
- %Numeric Slider+パラメーター（Params>Special>Numeric Slider）をキャンバスにドラッグ＆ドロップします。
- %Numeric Slider+パラメーターを右クリックして次の値をセットします。
  - Name: Scale Factor
  - Slider Type: Integers
  - Lower Limit: 0.0
  - Upper Limit: 25.0
  - Value: 25.0
  - %Scale Factor+スライダーを+Division+コンポーネント B-入力に接続します。
- %Scale+コンポーネント（X Form > Affine> Scale）をキャンバスにドラッグ＆ドロップします。
- %Center Box+コンポーネント B-出力を+Scale+コンポーネント G-入力に接続します。
- %Division+コンポーネント R-出力を+Scale+コンポーネント F-入力に接続します。
- %Center Box+コンポーネントを右クリックしコンテクストメニューで+Preview+をオフにします。

  ここまでの GH 定義は下記のようになっているはずです。Rhino のビューポートを見ると、全てのボックスが、+Attractor Pt+からの距離によってスケーリングされているのが分かります。次のステップでは色情報を与え視覚的にスケール要素を見ていくことにします。


Scale Factor
25
Set One Point in the Rhino Scene
Attractor Pt
Dist
Div
Slider
3
Slider
25
RecGrid
Move
Box
Scale
Series
Z

ステップ **3**：それぞれのボックスに色情報を割り当てる

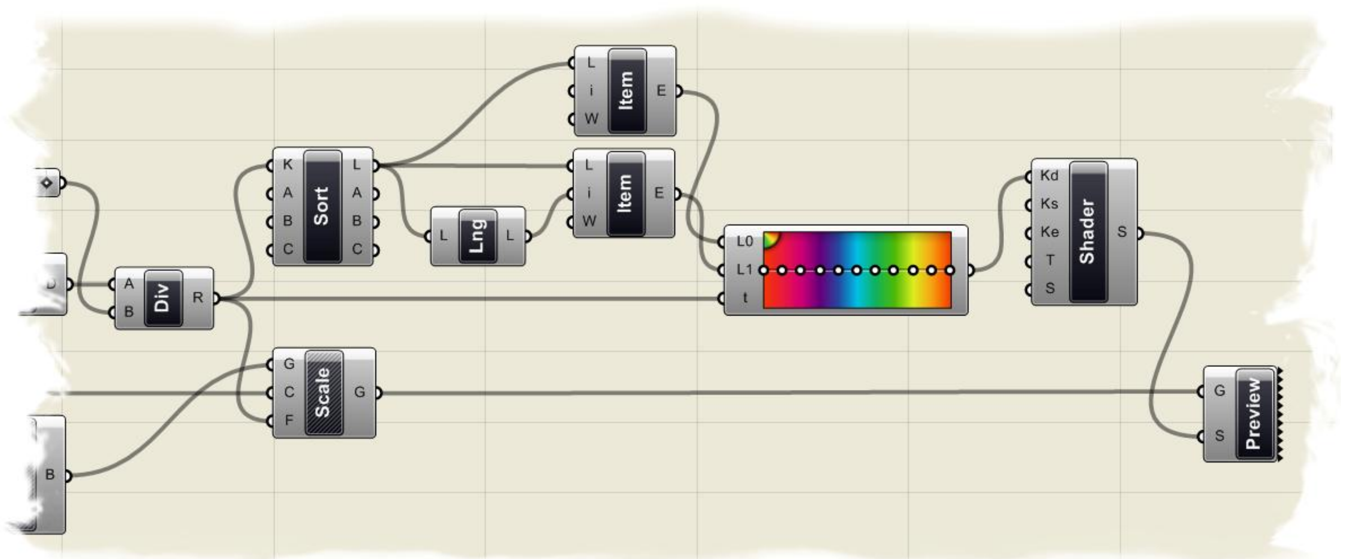
- %Sort List+コンポーネント（Logic>List>Sort List）をキャンバスにドラッグ＆ドロップします。
  色情報を+Attractor Pt+からの距離によってそれぞれのボックスに割り当てるには 2 つの数値が必要です。
  最も近い点と、最も遠い点です。これを行うには、まず、距離の数値リストをソートする（並べ替える）必要があります。
- %List Item+コンポーネント（Logic>List>List Item）をキャンバスにドラッグ＆ドロップします。
- %Sort List+コンポーネント L-出力を+List Item+コンポーネント L-入力に接続します。
- %List Item+コンポーネント i-入力を右クリックして、+Integer+の値を+0.0+に設定します。この値が、リストの最小値を持つ最初のエントリーを検索します。
- %List Length+コンポーネント（Logic>List> List Length）をキャンバスにドラッグ＆ドロップします。
- %Sort List+コンポーネント L-出力を+List Length+コンポーネント L-入力に接続します。+List Length+コンポーネントによって、どれだけエントリーの数があるかが分かります。この情報をもう一つの+List Item+コンポーネントに与え、最後の値を検索します。
- %List Item+コンポーネント（Logic>List>List Item）をもう一つキャンバスにドラッグ＆ドロップします。
- %Sort List+コンポーネント L-出力を+List Item+コンポーネント L-入力に接続します。
- %List Length+コンポーネント L-出力を 2 つめの+List Item+コンポーネント i-入力に接続します。
  この時点で+List Item+コンポーネント E-出力に、マウスカーソルを持ってくると+L-入力+の最初のエントリーの値が出力されているはずです。Grasshopper は常に最初のエントリー番号を+0+にします。
  ここで、2 つめの+List Item+コンポーネントに最後のエントリー番号の値を取得させるには、List Length+コンポーネントの出力値から+1+を引いてやれば良いことになります。
- 2 つめの+List Item+コンポーネントの、i-入力を右クリック+Expression %を選択し、+Expression Editor+を立ち上げます。
- 数式**”i-1”**を入力します。
  ここで、マウスカーソルを 2 番目の+List Item+コンポーネント E-出力に持っていくと、+Attractor Pt+から最も遠い点の距離が表示されているのが分かるはずで。
- %Gradient+パラメーター（Params>Special> Gradient）をキャンバスにドラッグ＆ドロップします。
- 最初の+List Item+コンポーネント出力を（最も近い点の数値を持つ）+Gradient+パラメーターL0-入力に接続します。
- 2 番目の+List Item+コンポーネント出力を（最も遠い点の数値を持つ）+Gradient+パラメーターL1-入力に接続します。
  L0 入力は、グラデーションの最小値（グラデーションの左端部）となります。この例では、+Attractor Pt+に最も近い点の値となります。L0 入力は、グラデーションの最大値（グラデーションの右端部）となります。この例では、+Attractor Pt+に最も遠い点の値となります。T-入力リストの値によって、どのグラデーションを割り当てるかを表わします。これで、入力のスケール要素値は全て決定されました。
  それぞれのボックスのスケーリングは、グラデーション範囲の色と関連します。
- %Create Shader+コンポーネント（Vectors>Color> Create Shader）を、キャンバスにドラッグ＆ドロップします。
- %Gradient+パラメーター出力を+Create Shader+コンポーネント Kd-入力に接続します。
  %Create Shader+コンポーネントは、プレビュー時の色表示をするためにいくつかの入力があります。

Kd: シェーダーの、Diffuse（散乱）カラーを定義します。この色はそれぞれのオブジェクトカラーの基本色です。Diffuse カラーは、RGB 値を示す、+0～255+3 つの整数の組み合わせによって、定義されます。
Ks:Specular（鏡面反射）ハイライトを定義します。 *RGB 値を示す、”0～255”3 つの整数の組み合わせによって、定義されます。*
Ke: シェーダーのの自己発光色を定義します。
T: シェーダーの透明度を定義します。
S: シェーダーの輝度を定義します。数値は+0～100+で、+0+で輝度無し、+100+で輝度最大となります。

ここで、グラデーションのスライダーを、Diffuse に接続していますがこれによってボックスの基本カラーはグラデーションパターンで表示されます。
このグラデーションの小さな+ドット+をクリックすると、グラデーションのパターンと、入力及び、出力の色を変えることが出来ます。ドットの位置をスライドして、変化を調整したり、順番を入れ替えたりすることも可能です。また、この例では、+***Spectrum***.+というタイプのグラデーションを設定していますが、+Gradient+パラメーターを右クリックして、グラデーションパターンを変えることができます。

- %Custom Preview ‰パラメーター（Params/> Special > Custom Preview）を、キャンバスにドラッグ＆ドロップします。
- %Scale-G+コンポーネント出力を+Custom Preview ‰パラメーターG-入力に接続します。
- %Create Shader+コンポーネント出力を+Custom Preview ‰パラメーターS-入力に接続します。
- %Scale-G+コンポーネントを右クリックし、+Preview+オフにします。

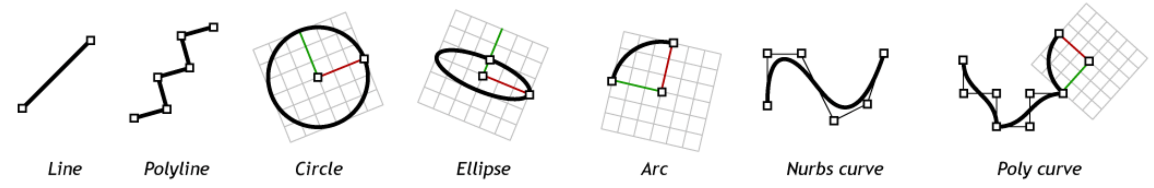
下図が、第 3 ステップの GH 定義になります。Rhino ビューポート上で、+Attractor Pt+を動かすと、ボックスのスケールと色情報が変化するのがわかります。



10 カーブタイプ

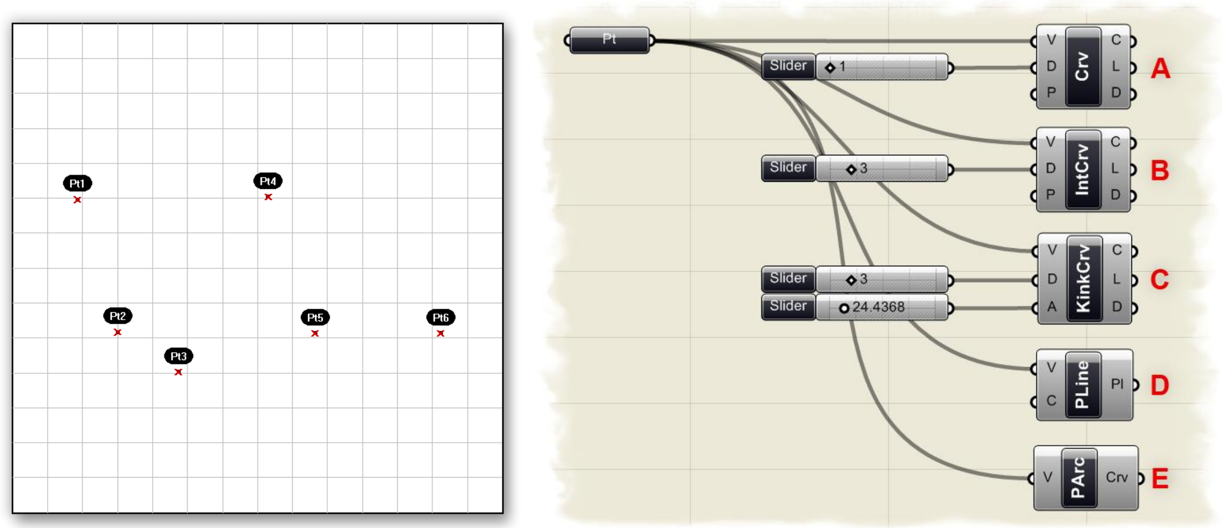
カーブは、ジオメトリーオブジェクトですが、いくつかの属性や特徴がそれらを分析するのに使用されます。
例えば全てのカーブは、始点と終点を持ちます。
もし、これらの距離が+0+であれば、カーブは閉じています。
また、全てのカーブはコントロールポイントを持ちます。もしも全てのコントロールポイントが同じ平面上にあれば、カーブは平面上にのっています。

ある属性がカーブ全体にあるいは、カーブのある複数の点に対して与えられます。
例えば、接線ベクトルはローカルな属性ですが、平面性はグローバルな属性です。
また、いくつかの属性は、あるカーブタイプのみに適用されます。
現時点で、Grasshopper のプリミティブなカーブコンポーネントとして直線、円、楕円、円弧があります。



Grasshopper は、また Rhino のより高度な NURBS カーブや、複合カーブを表現するツールセットを持ちます。
これより、Grasshopper のスプラインコンポーネントの例を見ていきますが、まず、カーブがどのように挙動するのかをみるために、点群を作成しておきます。

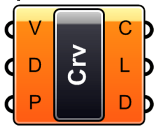
ソースファイル中の、+**Curve Types.3dm**+を開いてください。
Rhino ビューポート、X-Y 平面上に 6 つのポイントがありますが、これらを、Grasshopper 定義で選択していきます。



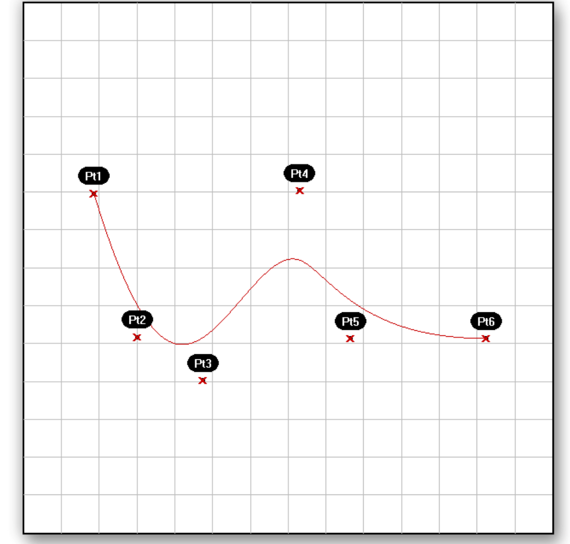
次に Grasshopper で+**Curve Types.ghx**+を開いてください。
ここで+Point+パラメーターがいくつかの異なる+Curve+コンポーネントに接続されているのが分かります。
この違いを順番に見て行きます。最初に+Point+パラメーターを右クリックし、コンテクストメニューで+**Set Multiple Points**+を選択して、Rhino ビューポート上でポイントを+Pt1+から順番に選択していきます。

ポイントを選択していく際に、Rhino のビュー上に青で選択したポイントと、ポイント間の接続線が表示されます。全て選択を終了したら、+Enter+キーで Grasshopper 画面に戻ります。Grasshopper 画面に戻った時点で、選択したポイントは、赤の+X+で表示され、Grasshopper のポイントコンポーネントに戻ったことが分かります。

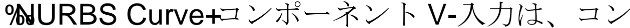
A) %NURBS Curve+コンポーネント



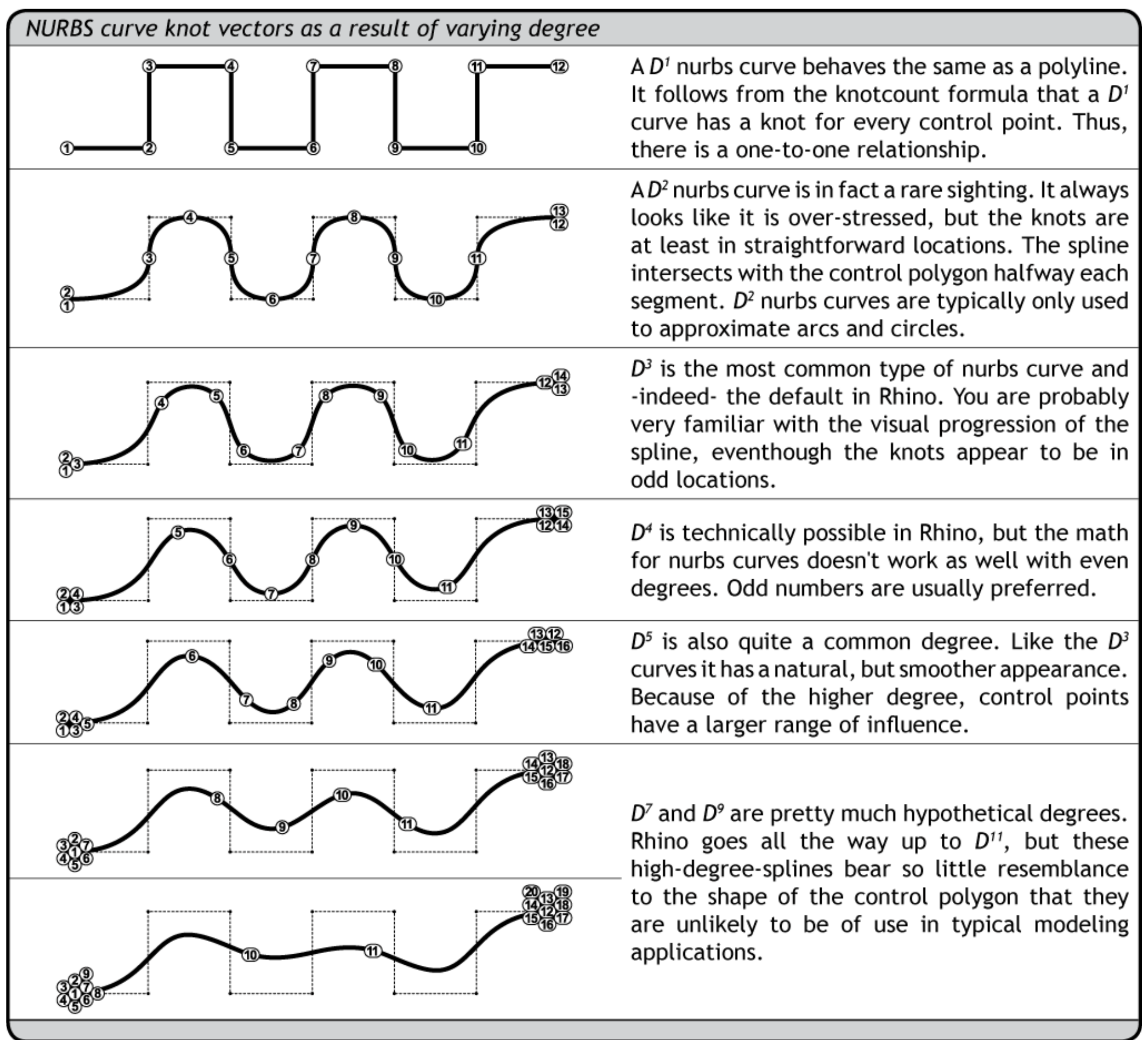
（Curve>Spline>**Curve**）は、Rhino 上で扱える多くのカーブ定義のうちの一つです。



このカーブは **NURBS** （ナーブス：**N**on-**U**niform **R**ational **B**asic **S**plines）カーブで、NURBS カーブ、単に NURBS 等と呼ばれます。
コントロールポイントの位置情報に加え、NURBS カーブは、次数、ノットベクトル、ウエイト等の属性を持ちます。
ここで、NURBS カーブの数学的背景について説明はしませんが、興味のある方は、http://en.wikipedia.org/wiki/NURBS に書かれていますので、参照してください。



%NURBS Curve+コンポーネント V-入力は、コントロールポイントの位置を定義します。コントロールポイントの定義は、Rhino ビューポート上においてポイントを選択するか、揮発性のデータ（Volatile data）として他のコンポーネントから、継承させることも出来ます。
%NURBS Curve+コンポーネント D-入力は、カーブの次数を定義するもので、1 次から 11 次までの整数値で定義されます。カーブの次数は、コントロールポイントの影響範囲を決めます。高次であれば、より大きな範囲に影響します。次の表は、David Rutten 氏のマニュアル+*Rhinoscript 101”*からの抜粋で、次数がどのように NURBS カーブの形状を定義するかが分かります。



上から、順に概要は、

1 次のカーブ

1 次の NURBS カーブはポリラインと同じように挙動する。ノット関数に従い、1 次のカーブは、全てのコントロールポイントにノットを持つ。従って、コントロールポイントとノットとの関係は、1:1 である。

2 次のカーブ

2 次のカーブは、あまり使用されない。形状は常に強調された形になっているが、ノットは単純な配置になっている。ノットはコントロールポリゴンの中間に位置し、スプラインはその点で接続している。2 次のカーブは通常、円弧、円を近似するために使用される。

３次のカーブ

３次のカーブは、最も一般的なタイプの NURBS カーブで、Rhino では初期値に３次のカーブが設定されている。ノットの位置が変な場所にありますが、恐らくこの読者の方にはなじみのある形状のはずである。

４次のカーブ

４次のカーブは Rhino において技術的に可能である。しかしながら一般的に NURBS の数学表現では、偶数の次数のカーブはうまく機能しない。（2 次は例外）
通常、奇数の次数のカーブが使用される。

５次のカーブ

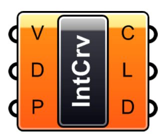
５次のカーブも、一般的に使用されるものである。3 次のカーブのような自然な形を持つが、より滑らかな形状を持つ。
次数が高いため、コントロールポイントの影響は大きくなる。

7 次,9 次のカーブ

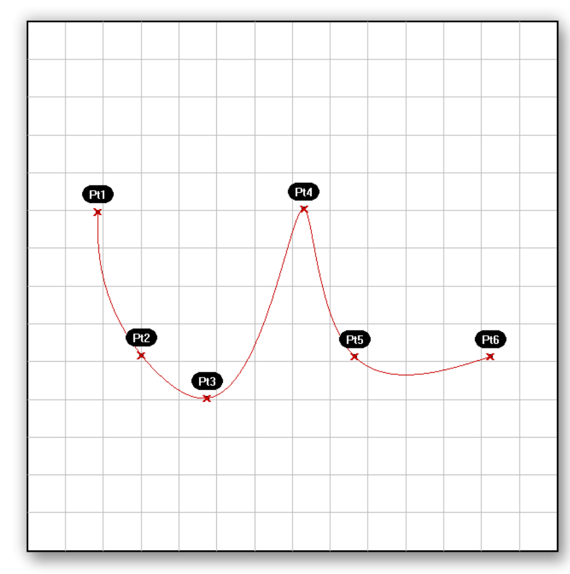
7 次,9 次のカーブは、仮想的な次数である。Rhino は 11 次までの次数を扱えるが、これらの高次のスプラインは、コントロールポリゴンに比べて形状がかけ離れているので、一般的なモデリングアプリケーションでは使用されない。

この例では、次数を指定する+Slider+パラメーターが+Curve+コンポーネント D-入力に接続されています。スライダーをドラッグして、次数の変更によって、どのように形状に影響を与えるかを確認することができます。
%Curve+コンポーネント P-入力は、+論理値+の定義で、カーブを周期カーブにするか否かを決めます。+False+は開いた NURBS カーブを、+True+は閉じた NURBS カーブを定義します。+Curve+コンポーネントの 3 つの出力は見て分かる通りです。
C-出力は、カーブの最終形状を定義し、L-出力はカーブの+長さ+を、D-出力はカーブのドメイン（定義域を）または、カーブの次数によって分割される数値を出力します。

B) “Interpolated Curve”コンポーネント (Curve>Spline>**Interpolate**)



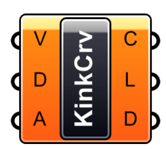
Interpolated Curves（通過点指定のカーブ）は、先の NURBS Curve(コントロールポイント指定のカーブ)とは若干、異なります。
通常、コントロールポイント指定で、始点・終点以外で、ある座標を通過させるのは非常に困難です。仮にコントロールポイントを引っ張るなりして作業したとしても空間のある一点を通過させることは容易ではありません。
%Interpolated Curves+は V-入力において、ある点群を渡すという意味では似ていますが、通過点指定の方法では結果として作成されるカーブは、次数に関係なく、その入力した点を通過するように作成されます。



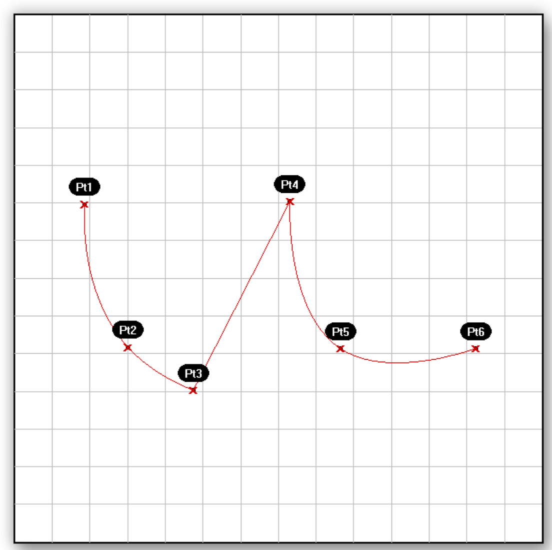
（先の+NURBS Curve+コンポーネントは、通過点を指定させることが出来るのは、次数が+1+のときのみです。）

D-入力は次数を指定しますが、このコンポーネントにおいては、次数+2+のような偶数の次数ではカーブが作成されません。
P-入力は+NURBS Curve+コンポーネント同様、カーブを閉じるかどうか指定する+論理値+入力です。
C-出力、L-出力、D-出力は+NURBS Curve+コンポーネントと同じです。

C) “IKinky Curves”コンポーネント (Curve>Spline> **Kinky Curve**)

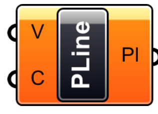


その名前に似つかわしくなく+Interpolated Curves+コンポーネント と変わりませんが一つだけ異なる機能があります。
%Kinky Curves+コンポーネントは滑らかな通過点指定カーブに対してある角度を指定することによって、+折れた+部分を作成することが出来ます。

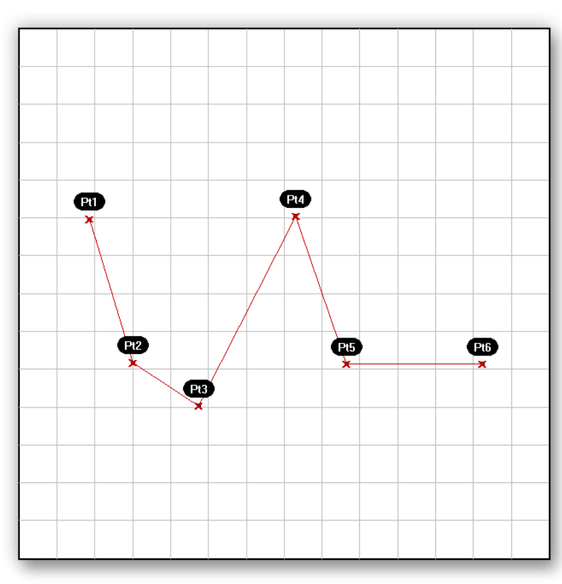


ここでは、A-入力にスライダーを接続し角度入力すると、ラジアンに変換されます。

D) “Polyline Curves”コンポーネント (Curve>Spline>Polyline)



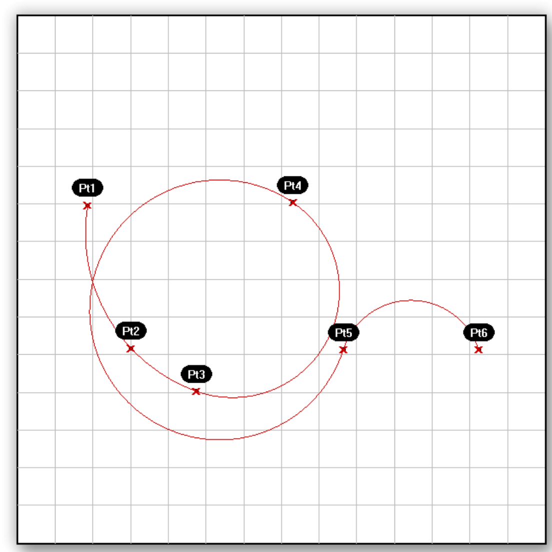
ポリラインは、Rhino において最も柔軟性のあるカーブタイプかもしれません。ポリラインカーブはラインセグメントから構成され、ポリラインの次数は+1+です。
実質的にポリラインは点の配列です。違いは、ポリラインの点を連続するものとして扱い、それらの連続線を描きだします。
先に述べたように、1 次の NURBS カーブは、ポリラインと認識されます。
ポリラインは、2 つ以上のラインセグメントの集合体で、必ずコントロールポイント上を通過します。この点では、+Interpolated Curve+コンポーネントに似ています。



%Polyline Curves+コンポーネント V-入力は、ポリラインを構成するそれぞれのラインセグメントの範囲を決める点群です。
V-入力は、ポリラインを閉じたものにするか、開いたものにするかの+論理値+です。
最初の点の座標が最後の点の座標と一致していなければ、閉じたループを作成するため、新しいラインセグメントが追加されます。
%Polyline Curves+コンポーネントの出力は、前のタイプと異なり、作成されたポリラインカーブそのものとなります。

E) “Poly Arc”コンポーネント(Curve>Spline>Poly Arc)

%Poly Arc+コンポーネントは、+Polyline Curves+コンポーネントと異なるところは、ラインセグメントの変わりに、それぞれの点の間に、連続した複合円弧を作成するところです。



%Poly Arc#は、与えられた点群に対して、接線方向の連続性を持たせながら複合の円弧を作成するユニークなコンポーネントです。
入力パラメーターは一つだけです。出力も、作成された複合円弧だけです。

10.1 カーブの分析

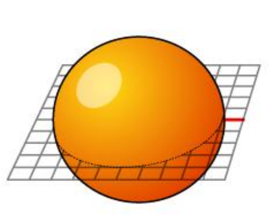
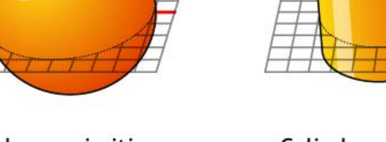
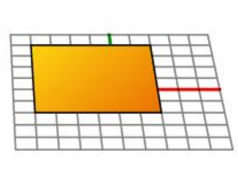
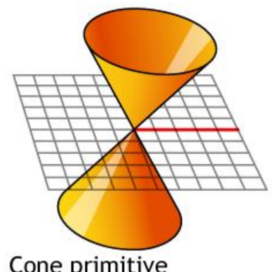
全てのカーブの分析のチュートリアルを作成するのは困難です。
以下に、分析ツールの一覧と概要を記述します。

Component	メニューの場所	概要	Example
A Cen C R	Curve/Analysis/**Center**	円弧・円の中心点と半径を検出	r {x,y,z}
C Cls C P	Curve/Analysis/**Closed**	カーブが閉じているかチェック	
P C Crv CP P t D	Curve/Analysis/**Curve CP**	指定した点からカーブへの最短距離を検出	d
C End S E	Curve/Analysis/**End Points**	カーブの終点を検出.	{x,y,z} E S {x,y,z}
C R Explode S V	Curve/Analysis/**Explode**	ポリカーブを分解	
C P Join C	Curve/Utility/**Join Curves**	複数のカーブセグメントを可能な範囲で結合	
C Len L	Curve/Analysis/**Length**	カーブの長さを測定	*l*
C N K Divide P T t	Curve/Division/**Divide Curve**	カーブを指定した数で分割した位置に点を作成	n
C D Divide P T t	Curve/Division/**Divide Distance**	カーブを指定した距離で分割した位置に点を作成	d
C L Divide P T t	Curve/Division/**Divide Length**	カーブを指定した長さで分割した位置に点を作成	*l*
C G Flip C F	Curve/Utility/**Flip**	カーブの方向を反転	$\vec{v}$ $\vec{w}$

C D P **Offset** C	Curve/Utility/**Offset**	指定した数値でカーブをオフセット	d
C B D **Project** C	Curve/Utility/**Project**	カーブを Brep に投影 (Brep は、Rhino でいうポリサーフェスのように結合されたサーフェスの集まり。)	
C B **Split** C P	Curve/Utility/**Split with Brep(s)**	カーブを一つ以上の Brep で分割	
C B **Trim** Ci Co	Curve/Utility/**Trim with Brep(s)**	カーブを一つ以上の Brep でトリム **Ci** (カーブインサイド) と **Co** (カーブアウトサイド)でトリムする部分を指定	
C R P **Trim** Ci Co	Curve/Utility/**Trim with Region(s)**	カーブを一つ以上の領域でトリム **Ci** (カーブインサイド) と **Co** (カーブアウトサイド)でトリムする部分を指定	
C P **Union** R	Intersect/Boolean/**Region Union**	2 つの閉じたプラナーカーブの最大外形（和）を作成	
A B P **Int** R	Intersect/Boolean/**Region Intersection**	2 つの閉じたプラナーカーブの交差部を作成	
A B P **Diff** R	Intersect/Boolean/**Region Difference**	2 つの閉じたプラナーカーブの差を作成	

11 サーフェスタイプ

球、円錐、平面や円柱等のプリミティブなサーフェスタイプを除き、Rhino は 3 つの自由曲面タイプをサポートします。最も使いやすいのが、NURBS サーフェスです。
カーブ同様、全てのサーフェスで表現可能な形状は NURBS サーフェスで表現することができます。それは最も役に立つサーフェスの定義で、我々がこれよりフォーカスしていくものです。

Sphere primitive
{Plane; Radius}

Cylinder primitive
{Plane; Radius; Height}

Plane primitive
{Plane; Width; Height}

Cone primitive
{Plane; Radius; Height}

NURBS サーフェスは NURBS カーブ同様、形状、法線方向、接線方向、曲率や他の属性を計算するのに同じアルゴリズムが使用されています。ただ明白な違いがあります。例えばカーブは接線ベクトルと法線平面を持ちますが、サーフェスは法線ベクトルと接線平面を持ちます。
これは、サーフェスが方向を持たないのに対してカーブは向きの情報を欠いているということで、全てのカーブ及びサーフェスに当てはまることですので、このことは知っておく必要があります。カーブやサーフェス含むコーディングをする際に方向や向きを仮定する必要がありますが、この予測は時として間違うことがあります。
NURBS サーフェスにおいてはジオメトリーによって 2 つの方向がありますが、これは、NURBS サーフェスが、{u} と{v}のカーブ直交したグリッドで定義されているからです。
これらの方向は時として任意に設定されてしまいますが、結局、その事実を受け入れて使用するのが現状です。

Grasshopper は Rhino と同じように NURBS サーフェスを扱いますが、Grasshopper 定義のサーフェスは、Rhino のビューポート上に表示されるため（これが、何故 Grasshopper で作成されたジオメトリーが、Grasshopper 側で、+Bake+して Rhino オブジェクトに変換しない限り、Rhino ビューポート上で選択することが出来ない理由です。）、Grasshopper 上でのパフォーマンスを保つため、メッシュの設定が少し粗くなっています。
サーフェスのメッシュ表示について、この点に気がついている方もいるかも知れませんが、これは Grasshopper 側のドローイング設定によるものです。
一旦、Rhino オブジェクトに+Bake+されてしまえば、Rhino 側のメッシュ設定が使用出来ます。

Grasshopper は、サーフェスを 2 つの方法で扱います。
まず最初に、NURBS サーフェスの使用について既に見てきました。

一般的に、全てのサーフェス解析コンポーネントは NURBS サーフェスに対して使用されます。例えば、サーフェスの領域や、あるサーフェスの曲率の検出等です。
そこには多くの複雑な数学的な問題が含まれていますが、この問題は、コンピューターにとってはボリュームの深度や厚み等を 3 次元で扱う必要がないので、まだ簡単です。

では、Grasshopper はどのように 3 次元のサーフェスを理解するのでしょうか？McNeel 社の開発は、私たちに任せることに決めました。そして、2 番目の方法を作りました。それは、ソリッドのオブジェクトを Rhino の中で扱うのと同じように、コントロールする方法を与えたのです。Brep や、境界表現を使用する方法です。

Brep は Rhino のポリサーフェスのように、3 次元のソリッドとして考えられています。
Brep は NURBS サーフェスの集合体ですが、1 枚の NURBS サーフェスが理論的に厚みを持たないのに対して、Brep は厚みを持つためにそれぞれのサーフェスが結合されています。

Brep は、本質的には結合された複数のサーフェスから構成されますので、NURBS サーフェスの一般的なサーフェス解析コンポーネントは、Brep に使用することが出来ます。
これは、Grasshopper が、オブジェクトを求められる入力に変換を行うようなロジックを持つようにプログラムされているからです。

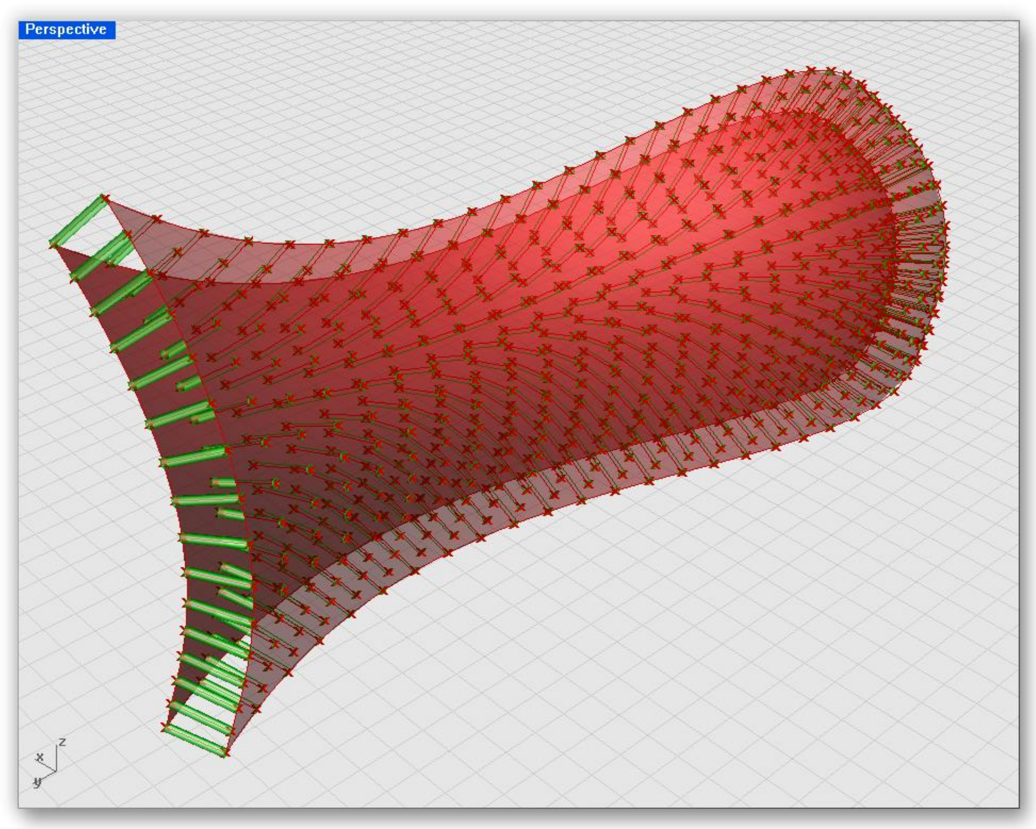
もし、コンポーネントが Brep を入力値として求め、それをあなたがサーフェスとして与えた場合、サーフェスは即座に Brep に変換されます。
これは数値と整数、色とベクトル、円弧と円等についても同様です。
実際には、幾つかのかなりエキゾチックな変換がされます。例として

- カーブ　数値(カーブの長さを与える数値)
- カーブ　間隔 (カーブのドメインを表わす間隔)
- サーフェス　2D の間隔(サーフェスの UV ドメインを表わす間隔)
- 文字列　数値 (文字列を変換
- 2D の間隔　数値(領域の間隔)

実際にはもっと多くの自動変換が有り、他のデータタイプが追加された場合、このリストは増えていきます。これで、サーフェスタイプの背景については十分ですので、これからいくつかの異なる例を見て行きます。

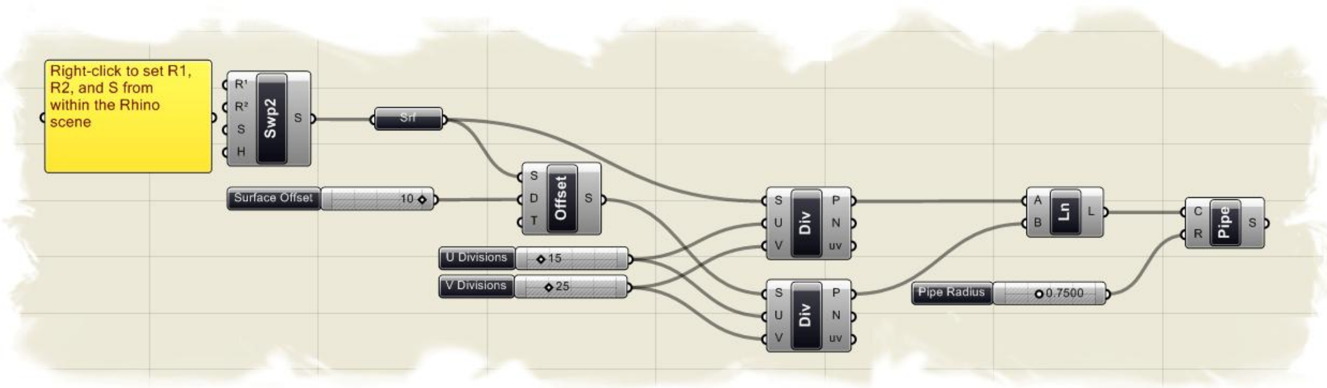
11.1 Surface Connect　（サーフェスの結合）

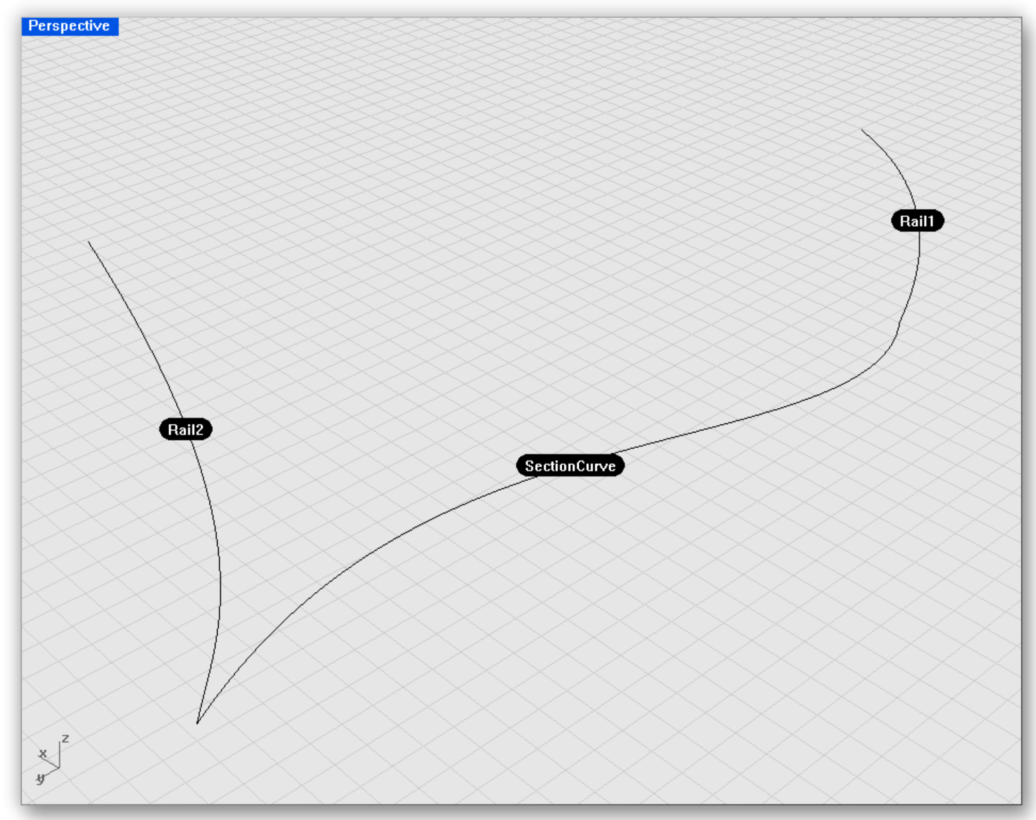
次の例は、Design Reform の David Fano 氏が作成したもので、サーフェスを扱う技術のチュートリアルとして大変、素晴らしいものです。



この例で、+Sweep2Rail+、+Surface Offset+、+Surface Division+コンポーネント理解します。まず、**SurfaceConnect.3dm**　をソースフォルダから開いてください。このモデルには、3 つのカーブがあり（2 つはレールカーブ、1 つはセクションカーブです。）、この例のフレームワークとなります。

注：この最終結果を、みるには、**SurfaceConnect.ghx** を開いてください。





この GH 定義をスクラッチから作成するには

- %Sweep2Rail+コンポーネント（Surface>Freeform>Sweep2Rail）をキャンバスにドラッグ＆ドロップします。
- %Sweep2Rail+コンポーネント R1-入力を右クリックし+Set one Curve+を選択します。
- 入力が Rhino ビューポートに移るので、最初のレールカーブを選択します。
- %Sweep2Rail+コンポーネント R2-入力を右クリックし+Set one Curve+を選択します。
- 入力が Rhino ビューポートに移るので、2 番目のレールカーブを選択します。
- %Sweep2Rail+コンポーネント S--入力を右クリックし+Set one Curve+を選択します。
- 同様に、セクションカーブを Rhino ビューポート上で選択します。
  カーブがそれぞれ適切に選択されると、2 つのレールに沿ったサーフェスが作成されるはずです。
- %Offset+コンポーネント（Surface>Freeform> Offset）をキャンバスにドラッグ＆ドロップします。
- %Sweep2Rail+コンポーネント S-出力を+Offset+コンポーネント S-入力に接続します。
- %Number Slider+コンポーネントをキャンバスにドラッグ＆ドロップします。
- %Number Slider+コンポーネントを右クリックして以下を設定します。
  - Name: Surface Offset
  - Slider Type: Floating Point
  - Lower Limit: 0.0
  - Upper Limit: 10.0
  - Value: 10.0
- %Number Slider+コンポーネントを+Surface Offset+コンポーネント D-出力に接続します。
  これで、オリジナルのサーフェスから、10 ユニット（あるいは、スライダーで設定した任意の値分）、オフセットされたサーフェスが作成されているはずです。
- %Divide Surface+コンポーネント（Surface>Utility>Divide Surface）を 2 つキャンバスにドラッグ＆ドロップします。

- %Sweep2Rail+コンポーネント S-出力を最初の+Divide Surface+コンポーネント S-入力に接続します。
  ここで、最初の 2 レースサーフェス上に点群が作成されているのが確認できます。これは、+Divide Surface+コンポーネントに初期設定されている、U 方向、V 方向の値が、それぞれ、+10+だからです。
  基本的に、+Divide Surface+コンポーネントは、U 方向、V 方向にそれぞれ 10 個ずつ分割し、最後に、サーフェス上に格子点を生成します。
  もしも、それぞれの分割したラインに沿って点を接続すると、サーフェスの内部的な構成要素である+iso カーブ+を得ることができます。
- %Surface Offset+コンポーネント S-出力を、+Divide Surface+コンポーネント S-入力に接続します。
  これで、新しくオフセットされたサーフェスが作成されます。
- %Number Slider+コンポーネント（Params>Special>Slide）を、2 つキャンバスにドラッグ＆ドロップします。
- 最初の+Number Slider+コンポーネントを右クリックして以下を設定します。
  - Name: U Divisions
  - Slider Type: Integer
  - Lower Limit: 0.0
  - Upper Limit: 100.0
  - Value: 15.0
- 2 番目の+Number Slider+コンポーネントを右クリックして以下を設定します。
  - Name: V Divisions
  - Slider Type: Integer
  - Lower Limit: 0.0
  - Upper Limit: 100.0
  - Value: 25.0
- %U Division+スライダーを 2 つの+Divide Surface+コンポーネント U-入力に接続します
- %V Division+スライダーを 2 つの+Divide Surface+コンポーネント V-入力に接続します。
  2 つのスライダーは、それぞれのサーフェスの UV 方向への分割数をコントロールします。それぞれのサーフェスが、同じ数のポイントを持っていますので、それぞれのポイントが、同じインデックス番号を持っています。それゆえに、簡単に内側のサーフェス上の点と外側のサーフェス上の点を直線で結合することが出来ます。
- %Line+コンポーネント（Curve> Primitive > Line）をキャンバスにドラッグ＆ドロップします。
- 最初の+Divide Surface+コンポーネント P-出力を+Line+コンポーネント A-入力に接続します。
- 2 番目の+Divide Surface+コンポーネント P-出力を+Line+コンポーネント B-入力に接続します。
  これで、2 つのサーフェス間の点群間に作成された直線群を見ることが出来ます。次のステップで、直線群にパイプを作成し、その半径をコントロールする定義を行います。
- %Pipe+コンポーネント（Surface> Freeform > Pipe）を、キャンバスにドラッグ＆ドロップします。
- %Line+コンポーネント出力を+Pipe+コンポーネント C-入力に接続します。
- %Number Slider+コンポーネント（Params>Special>Slide）を、キャンバスにドラッグ＆ドロップします。
- 最初の+Number Slider+コンポーネントを右クリックして以下を設定します。
  - Name: Pipe Radius
  - Slider Type: Floating Point
  - Lower Limit: 0.0
  - Upper Limit: 2.0

    - Value: 0.75
- %Pipe Radius+スライダーを+Pipe+コンポーネント R-入力に接続します。

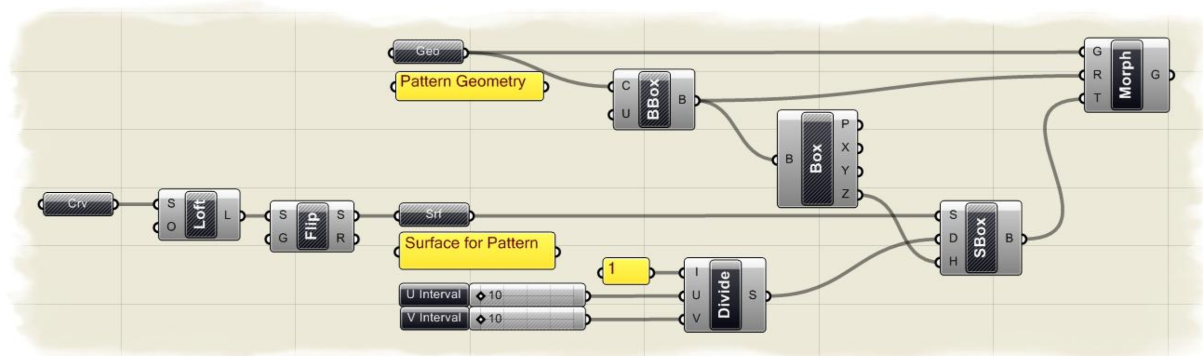
11.2 Paneling Tools

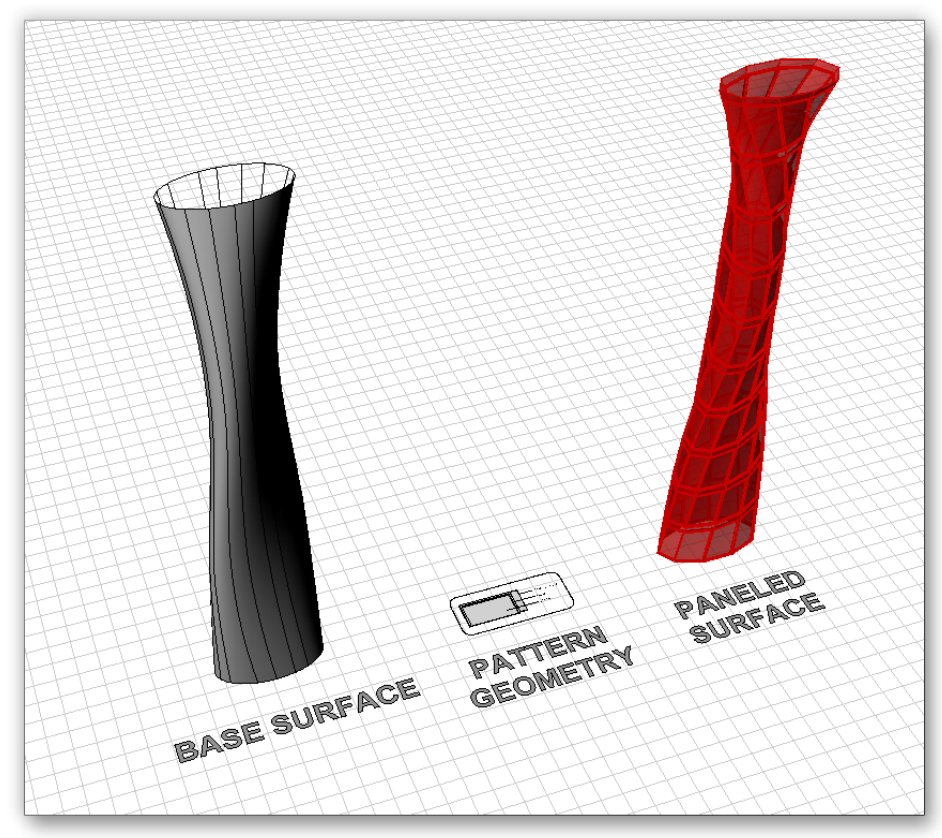
現在、McNeels 社で+**Paneling Tools.**+というプラグインが公開されています。
このプラグインは、ある特定のジオメトリーを与えられたサーフェス上に配列していくことができます。
Grasshopper もまた+Paneling Tools+の手法で作成することのできる幾つかのコンポーネントを有しています。
このチュートリアルでは、インターバルコンポーネントを使用してサーフェスを細分割し、変形するコンポーネントの変形・再配置する様子を学びます。

%Paneling Tools+について知りたい方は下記 URL を参照してください。
http://en.wiki.mcneel.com/default.aspx/McNeel/PanelingTools.html

下の図が、窓枠などのジオメトリーのパターンを高層のタワーに割り当てる GH 定義の最終イメージです。

注：この最終結果を見るには、**Paneling Tool.ghx** を開いてください。





この GH 定義をスクラッチから行うには、タワー部のサーフェスを作成するための断面が必要です。ソースファイルから、**Panel Tool_base.3dm** を開いてください。
そこに、4 つのカーブ断面がありますので、これを使用してサーフェスを作成します。

まず、ロフトサーフェスを作成する GH 定義から始めます。

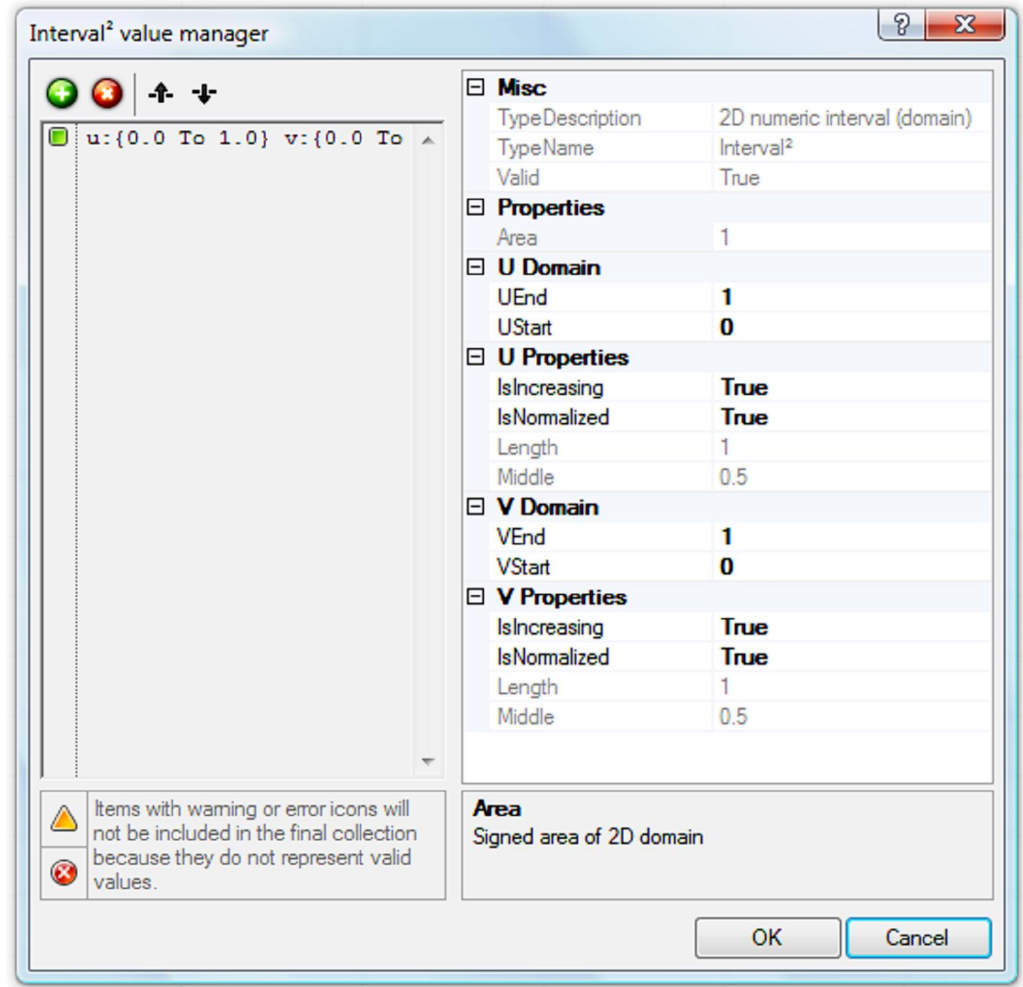
- %Curve+パラメーター（Params>Geometry>Curve）をキャンバスにドラッグ＆ドロップします。
- %Curve+パラメーターを右クリックし、コンテクストメニューで+Set Multiple Curves+します。
- 入力が Rhino のビューポートに移ったら、4 つの楕円を、下から選択していきます。
- 最後の、楕円の選択が終わったら、+Enter+キーを押します。
  前の例でも行ったようにこれで、Rhino のジオメトリーを Grasshopper に定義しました。
- %Loft+コンポーネント（Surface > Freeform > Loft）をキャンバスにドラッグ＆ドロップします。
- %Curve+パラメーターの出力を+Loft+コンポーネント S-入力に接続します。
  もしここで、+Loft+コンポーネント O-入力をクリックすると、Rhino の Loft コマンドにあるような、Loft コマンドの典型的な+オプション+があることが分かります。ここでは、初期値を使用しますが、各オプションを試されると良いでしょう。

* オプションステップ：Grasshopper のインターフェースにおいて、作成したロフトサーフェスが正しい方向を向いているかを判断することは出来ません。このトライアルでは、ロフトサーフェスの方向が内側を向くことが多く結果としてそれ以降に作成されるパネルも内側を向いてしまいます。そこで、+Flip+コンポーネントを使用して、サーフェスの法線方向を反転するステップを設けますが、最終結果が、反対方向を向いている場合は、次に設定する+Flip+コンポーネントの部分を削除してください。

- %Flip+コンポーネント（Surface >/Utility > Flipt）をキャンバスにドラッグ＆ドロップします。
- %Loft+コンポーネント L-出力を+Flip+コンポーネント S-入力に説即します。
- %Surface+パラメーター（Params>Geometry>Surface）をキャンバスにドラッグ＆ドロップします。
- %Flip+コンポーネント S-出力を+Surface+パラメーターに接続します。
- %Surface+パラメーターを右クリックし、コンテクストメニューで+**Reparameterize**+を選択します。
  この時点では、サーフェスのドメインは+0+（サーフェスの基点を示す）から任意の上限値（サーフェスの終値を示す。）を持っています。この値はサーフェスによって異なりますので、ここで、再パラメーター定義を行い、ドメインを+0+から+1+にセットします。
  これは、非常に重要なステップで、ここで再パラメーター定義をしないと、サーフェスを正しく分割出来ません。

- %Divide Interval^2+コンポーネント（Scalar>/ Interval > Divide Interval^2）をキャンバスにドラッグ＆ドロップします。
  この後、サーフェス分割するための範囲を設定する必要があります。

- %Divide Interval^2+コンポーネント I-入力をダブルクリックし、コンテクストメニュー+Manage Interval^2 Collection+を選択し、+Interval^2 value manager+を立ち上げます。
- %Interval^2 value manager+ダイアログが立ち上がりますので、緑色の++ボタンをクリックします。
  初期値ではインターバルの設定は、u:{0.0To 0.0}、v:{0.0To 0.0}となっていますが、ここでインターバルの範囲を、U 方向、V 方向について、それぞれ+0+から+1+にする必要があります。
- %U Domain+の、+U End+の値を、+1+に変えます。
- %V Domain+の、+V End+の値を、+1+に変えます。

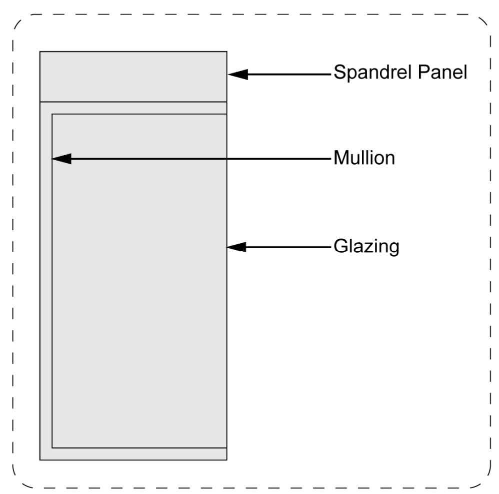


%Interval^2 value manager+ダイアログが、上図のように設定されたなら、+OK+ボタンをクリックして確定します。この時点で、マウスカーソルを、+Divide Interval^2+コンポーネント I-入力のところに持っていくと、インターバルの範囲が、U 方向、V 方向について、それぞれ+0+から+1+になっていることが分かります。

- %Number Slider+コンポーネント（Params>Special>Slider）を 2 つキャンバスにドラッグ＆ドロップします。
- 最初の+Number Slider+コンポーネントを右クリックして以下を設定します。：
  - Name: U Interval
  - Slider Type: Integers
  - Lower Limit: 5.0
  - Upper Limit: 30.0
  - Value: 10.0
- 2 つめの Number Slider+コンポーネントを右クリックして以下を設定します。：

- Name: V Interval
- Slider Type: Integers
- Lower Limit: 5.0
- Upper Limit: 30.0
- Value: 10.0

- %U Interval+パラメーターを+Divide Interval²+コンポーネント U-入力に接続します。
- %V Interval+パラメーターを+Divide Interval²+コンポーネント V-入力に接続します。
- %Surface Box+コンポーネント（Xform> Morph > Surface Box）をキャンバスにドラッグ＆ドロップします。
- %Surface+パラメーターを+Surface Box+コンポーネント S-入力に接続します。
- %Divide Interval²+コンポーネント S-出力を+Surface Box+コンポーネント D-入力に接続します。
- %Curve+パラメーター、+Loft+コンポーネント、+Surface+パラメーターを右クリックして、+Preview+をオフにします。



これで、我々は U,V のインターバル値をスライダーによって、指定し、100 個に細分割しました。もともとは、インターバルのレンジは、+0+から+1+でした。後に、U 方向、V 方向のインターバルをそれぞれ、10 に設定し、100 個のユニークな組み合わせを作成しました。この値を変えて、再分割の数をコントロールすることが出来ます。
ここで、ジオメトリーパターンの作成に戻ります。このジオメトリーパターンは、新しく細分割して作成された、サーフェスに変形して配置することが出来ます。Rhino のビューポートで、窓枠を構成する+Spandrel Panel（スパンドレル・パネル）+、+Mullion（縦仕切り）+、+Glazing（板ガラス）+があるのが分かります。この 3 つのジオメトリーを利用してサーフェス上にファサードを作成します。

- %Geometry+パラメーター（Params>/Geometry> Geometry）をキャンバスにドラッグ＆ドロップします。
- %Geometry+パラメーターを右クリックして+Set Multiple Geometries+を選択します。
- Rhino ビューポートに入力が移ったら、Rhino 上にオブジェクト、+Spandrel Panel（スパンドレル・パネル）+、+Mullion（縦仕切り）+、+Glazing（板ガラス）+を選択します。
- この 3 つの B-Rep を選択し終わったら、+Enter+キーを押します。
- %Bounding Box+コンポーネント（Surface>Primitive>Bounding Box）を、キャンバスにドラッグ＆ドロップします。
- %Geometry+パラメーターを+Bounding Box+コンポーネント C-入力に接続します。

ここで、+Bounding Box+コンポーネントを使用した理由は 2 つあります。まず、まず+Bounding Box+コンポーネントは、このジオメトリーパターンの、高さを決定します。
ここでは、矩形のパターンしか使用していませんので、高さを指定することは非常に簡単です。しかしながら、もしより有機的な形状をコピーして配置しようとしたら、もう少し難しいでしょう。
ここでは、高さ情報を+Bounding Box+コンポーネントへの入力情報とします。そして次に、+Bounding Box+コンポーネントを次の、+BoxMorph+コンポーネントの参照 B-Rep として使用するからです。

- Bounding Box+コンポーネント U-入力をを右クリックし、論理値を+True+にします。

これは重要なステップです。この設定によって、+Bounding Box+コンポーネントは 3 つの Brep オブジェクトから、一つのボックスを作成します。

- %Box+コンポーネント（Surface>Analysis>Box）を、キャンバスにドラッグ＆ドロップします。
- %Bounding Box+コンポーネント B-出力を、+Box+コンポーネント B-入力に接続します。
- %Box+コンポーネント Z-出力を、+Surface Box+コンポーネント H-入力に接続します。
  We're on the home stretch now.
- %Box Morph+コンポーネント（Xform>Morph>Box Morph）を、キャンバスにドラッグ＆ドロップします。
- %Pattern Geometry+パラメーターを+Box Morph+コンポーネント G-入力に接続します。
- %Bounding Box+コンポーネント B-出力を+Box Morph+コンポーネント R-入力に接続します。
- %Surface Box+コンポーネント、B-出力を+Box Morph+コンポーネント T-入力に接続します。
- %Surface Box+コンポーネントを右クリックして、+Preview+をオフにします。

最後の定義について説明します。我々は、+Box Morph+コンポーネントに、パターンジオメトリーを入力し、これらを分割されたサーフェス上にこれらのジオメトリーのコピーを作成しました。

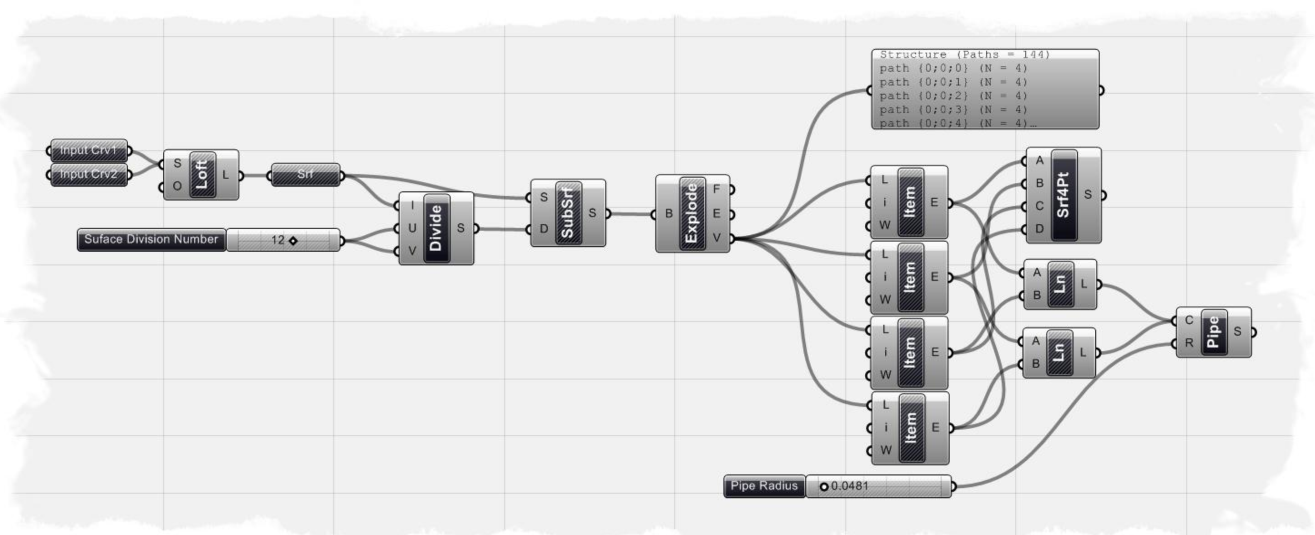
%Bounding Box+コンポーネントは、この窓のシステムが参照するジオメトリーを指定しました。

そして+Surface Box+コンポーネントは、2 つのスライダー値により、サーフェスを 100 に細分割しました。+
Box Morph+コンポーネントは参照するジオメトリーを、細分割したサーフェスをターゲットとして、元の形を変形して再配置しました。

元のカーブや、スライダー値を変えることで、異なる結果が出ますので試してください。

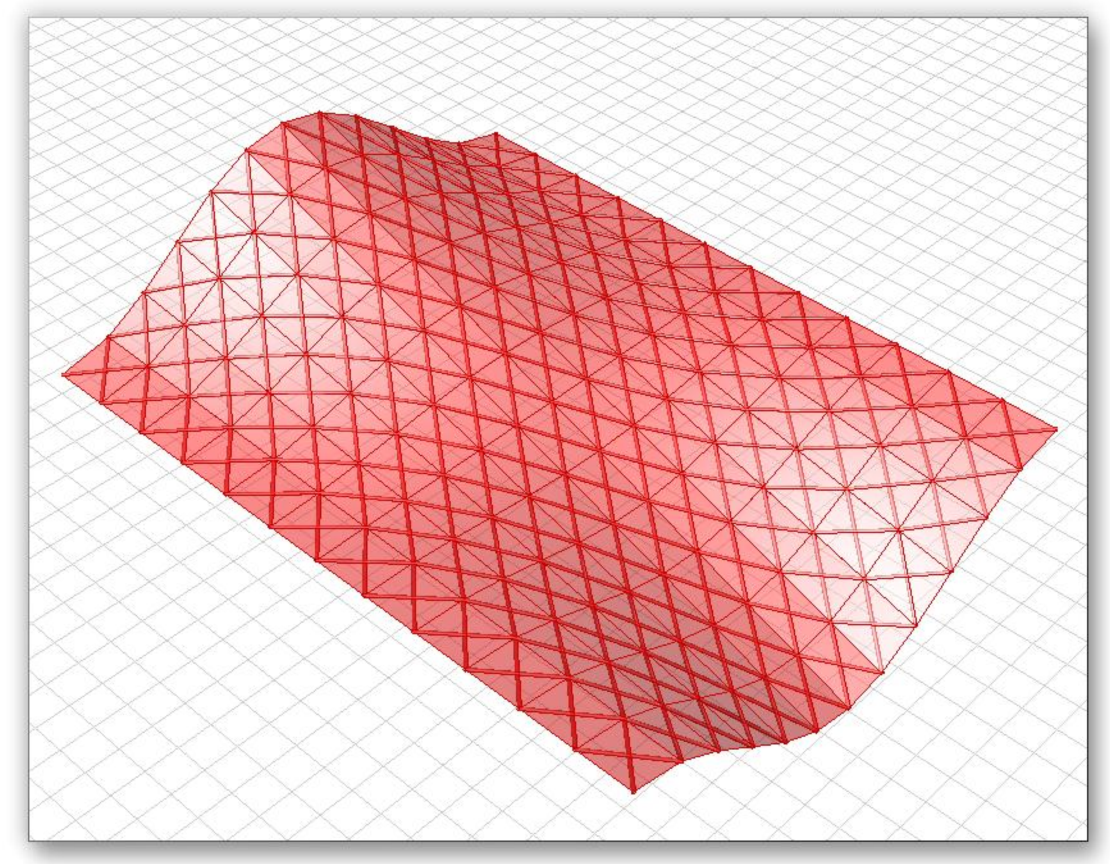
11.3 Surface Diagrid

前の例で、+Paneling Tools+の定義を使用して、ファサードをサーフェス上に作成する方法を学びました。次の例では、データの流れを使用して、どのサーフェス上にもダイアモンドグリッド構造（又はダイアグリッド）を作成する操作を学びます。
まず、**Surface Diagrid.3dm** を開いてください。Rhino のビューポート上に、ミラーされたコ



サインカーブがあります。そしてこれらが、ロフトサーフェスの境界を定義します。

注：この最終結果を見るには、**Surface Diagrid.ghx** を開いてください。



この GH 定義をスクラッチから始めるには

- %Curve+パラメーター（Params>Geometry>Curve）を 2 つキャンバスにドラッグ＆ドロップしてください。
- 最初の+Curve+パラメーター右クリックして+Input Crv1+と名前を変更します。
- 次に、+Set One Curve+を選択します。
- Rhino のビューポートに入力が移りますので、2 本のカーブのうちの 1 本を選択します。
- 2 番目の+Curve+パラメーター右クリックして+Input Crv2+と名前を変更します。
- 次に、+Set One Curve+を選択します。
- Rhino のビューポートに入力が移りますので、2 本のカーブのうちの残りのカーブ選択をします。
- %Loft+コンポーネント（Surface > Freeform> Loft）をキャンバスにドラッグ＆ドロップします。
- %Input Crv1+パラメーターを、+Loft+コンポーネント S-入力に接続します。
- 次に+Shift キー+を押しながら+Input Crv2+パラメーターを+Loft+コンポーネント S-入力に接続します。

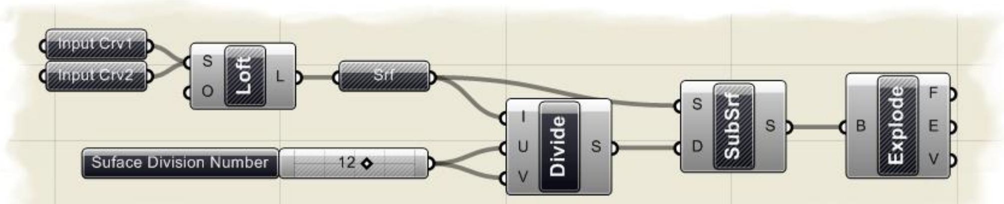
  この操作で、Rhino ビューポート上にサーフェスが作成されているはずです。
- %Surface+パラメーター（Params > Geometry> Surface）をキャンバスにドラッグ＆ドロップします。
- %Loft+コンポーネント L-出力を、Surface+パラメーター入力に接続します。
- %Divide Interval^2+コンポーネント（Scalar>/ Interval > Divide Interval^2）をキャンバスにドラッグ＆ドロップします。

  先の例同様、サーフェスを細分割します。

  これを行うためには、U 方向、V 方向に分割する数値を指定する必要があります。
- %Surface+パラメーターを+Divide Interval^2+コンポーネント I-入力に接続します。
- %Number Slider+パラメーター（Params>Special>Numeric Slider）をキャンバスにドラッグ＆ドロップします。
- %Number Slider+コンポーネントを右クリックして以下を設定します。:
  - Name: Surface Division Number
  - Slider Type: Integers
  - Lower Limit: 0.0
  - Upper Limit: 20.0
  - Value: 12.0
- 定義したスライダーを+Divide Interval^2+コンポーネントの U 及び V-入力に接続します。
- %Isotrim+コンポーネント（Surface>/ Utility> Isotrim）をキャンバスにドラッグ＆ドロップします。
- %Surface+パラメーターを+Isotrim+コンポーネント S-入力に接続します。
- %Divide Interval^2+コンポーネント S-出力を+Isotrim+コンポーネント D-入力に接続します。
- %Loft+コンポーネントを右クリックし、+Preview %をオフにします。

  スライダーによって設定された細分割の様子が見えるはずです。+Divide Interval^2+U 及び、V-入力には同じスライダーが接続されていますので、それぞれの方向の分割数は同じになります。もう一つスライダーパラメーターを配置し、それぞれの方向の分割数をコントロールすることも可能です。
- %Brep+コンポーネント（Surface> Analysis > Brep）をキャンバスにドラッグ＆ドロップします。

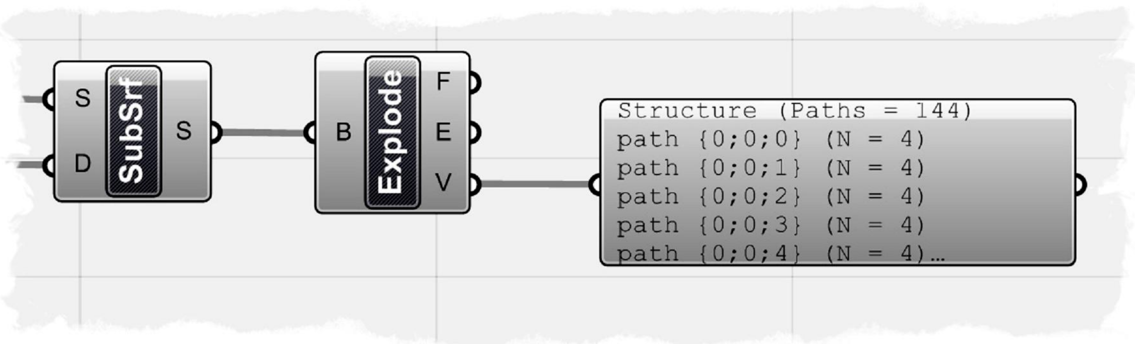
  このコンポーネントは、B-rep オブジェクトをフェイスや、エッジ、頂点等の部品に分解します。ここで知りたいことは、コーナーの点の座標です。この位置によって細分割されるダイアモンドグリッドの位置が決まります。



今までの GH 定義は、上図のようになります。我々は、サーフェスを細分割し、独立したサブサーフェスに分解し、それぞれのサーフェスのコーナーの点の座標を得ました。次のステップで、点座標のダイアモンドグリッド構造を作成するための、点群のリストをインデックスします。

- ‰Parameter Viewer ‰パラメーター（Params>Special > Parameter Viewer）をキャンバスにドラッグ＆ドロップします。
- ‰Brep+コンポーネント V-出力を、+Parameter Viewer ‰パラメーターに接続します。

  ‰Parameter Viewer ‰は、ツリー構造を確認するのに役に立ちます。ここでの例は、144 のパスがあり、それぞれの終点として 4 つのデータエントリーがあることを示しています。（この場合、細分割されたサーフェスのコーナーポイントです。）+List Item+コンポーネントを使用して、それぞれのパス、サブパスを調べ、データエントリーを取得することにします。まず、細分割されたサーフェスの何処が、空間上で、1 番目、2 番目、3 番目、4 番目にあたるのかを知る必要があります。



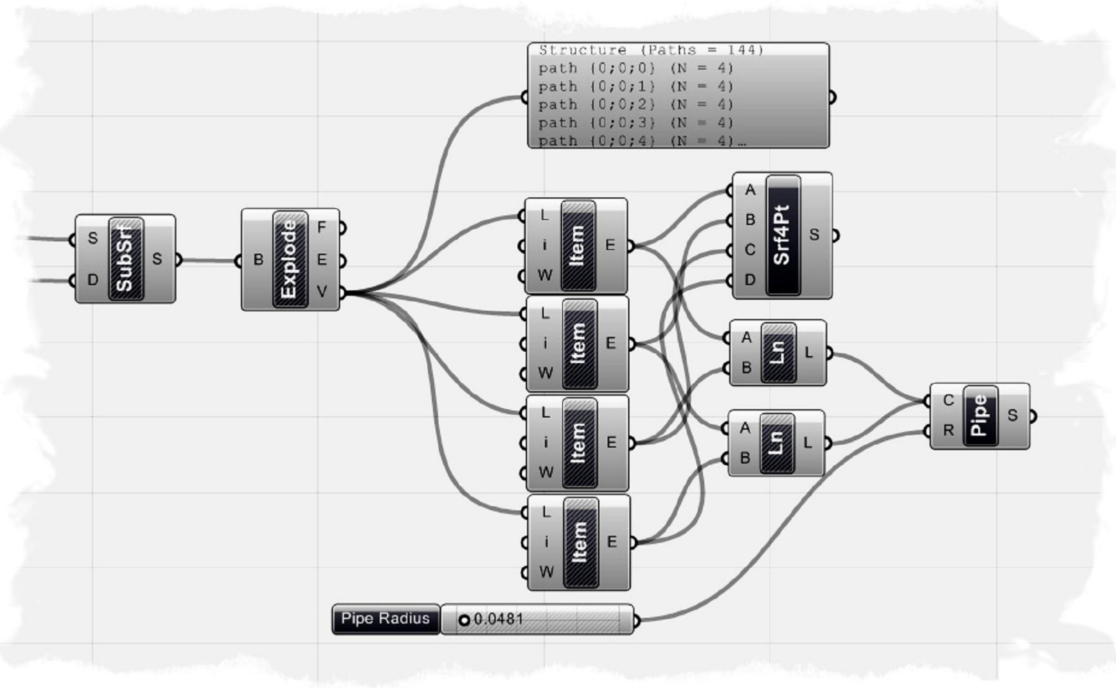
- ‰List Item+コンポーネント（Logic> List> List Item）を 4 つキャンバスにドラッグ＆ドロップします。
- ‰Brep+コンポーネント V-出力を 4 つの+List Item+コンポーネント L-出力に接続します。
- 最初の+List Item+コンポーネント i-入力を右クリックし、+Set Integer+に+0+を設定します。
- 2 番目の+List Item+コンポーネント i-入力を右クリックし、+Set Integer+に+1+を設定します。
- 3 番目の+List Item+コンポーネント i-入力を右クリックし、+Set Integer+に+2+を設定します。
- 4 番目の+List Item+コンポーネント i-入力を右クリックし、+Set Integer+に+3+を設定します。

  144 のパスがそれぞれ、4 つのアイテムを持っているわけですから、順番にインデックスを指定し、4 つのリストデータを作成します。

- ‰Line+コンポーネント（Curve>Primitive> Line）を、2 つキャンバスにドラッグ＆ドロップします。
- 最初の+List Item+コンポーネント出力を最初の+Line+コンポーネント A-入力に接続します。
- 3 番目の+List Item+コンポーネント出力を最初の+Line+コンポーネント B-入力に接続します。

- 2 番目の+List Item+コンポーネント出力を 2 番目の+Line+コンポーネント A-入力に接続します。
- 4 番目の+List Item+コンポーネント出力を 2 番目の+Line+コンポーネント B-入力に接続します。

  この時点で、ダイアモンド上の格子を作るようにライン作成されているのが見えるはずです。
- %Pipe+コンポーネント（Surface>Freeform> Pipe）をキャンバスにドラッグ＆ドロップします。
- 最初の+Line+コンポーネント出力を Pipe+コンポーネント C-入力に接続します。
- Shift キーを押しながら、2 番目の+Line+コンポーネント出力を Pipe+コンポーネント C-入力に接続します。
- %Number Slider+パラメーター（Params>Special>Numeric Slider）をキャンバスにドラッグ＆ドロップします。
- %Number Slider+コンポーネントを右クリックして以下を設定します。：
  - Name: Pipe Radius
  - Slider Type: Floating Point
  - Lower Limit: 0.0
  - Upper Limit: 1.0
  - Value: 0.05
- %Pipe Radius+スライダーを Pipe+コンポーネント R-入力に接続します。
- %4Point Surface+コンポーネント（Surface>Freeform> Point Surface）を、つキャンバスにドラッグ＆ドロップします。
- 最初の+List Item+コンポーネント出力を+4Point Surface+コンポーネント A-入力に接続します。
- 2 番目の+List Item+コンポーネント出力を+4Point Surface+コンポーネント B-入力に接続します。
- 3 番目の+List Item+コンポーネント出力を+4Point Surface+コンポーネント C-入力に接続します。
- 4 番目の+List Item+コンポーネント出力を+4Point Surface+コンポーネント D-入力に接続します。



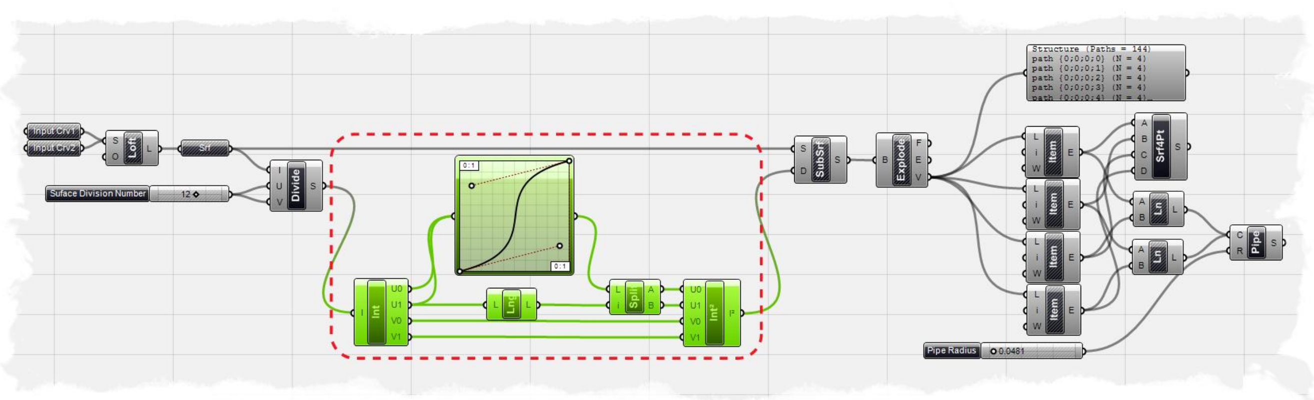
以上のプロセスで上手のような GH 定義ができているはずです。

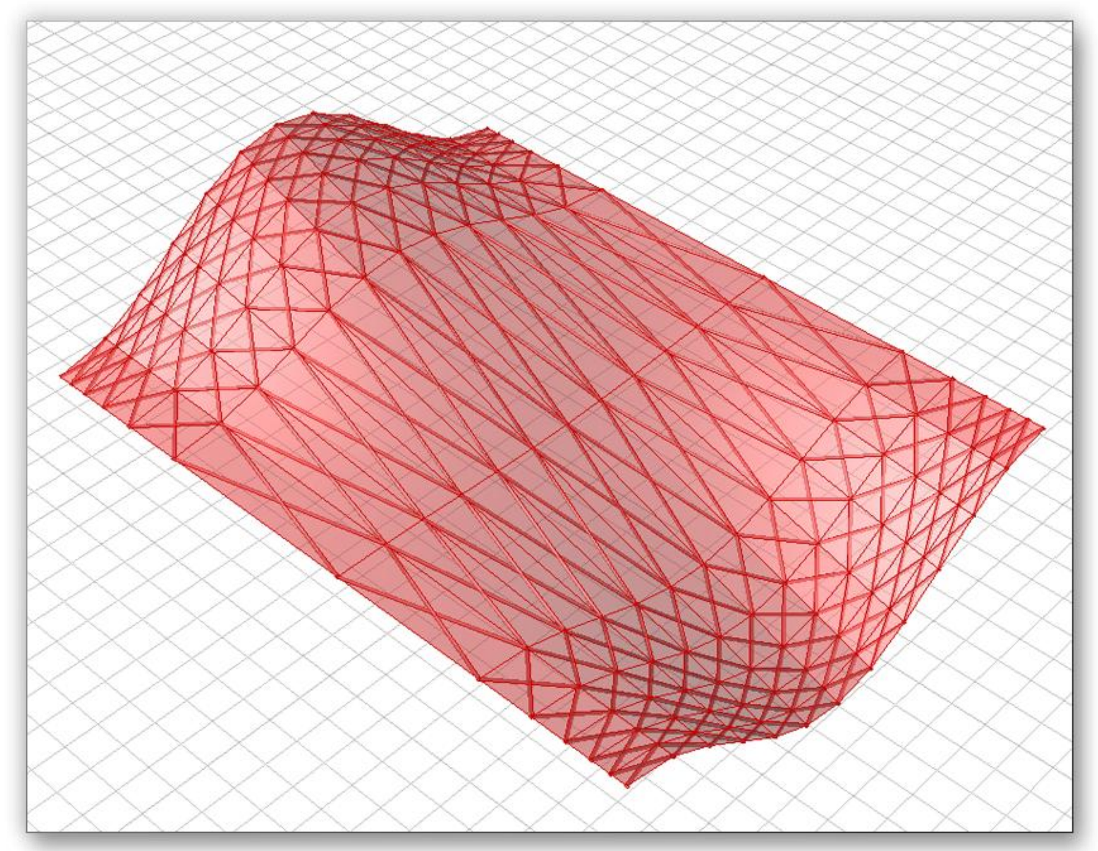
この GH 定義は、どのようなシングルサーフェス上にも適用できますので、最初のロフトサーフェスをより複雑な形状を持つサーフェスに置き換えて、試してください。

11.4 Uneven Surface Diagrid

前の例で、均等なダイアモンドグリッド構造にサーフェスを分割する方法を学びました。
ダイアモンドグリッド構造は、均等なスペースインターバルを与えられ、均等に分割されました。
たここでは、+Graph Mapper+コンポーネントと、他のコンポーネントを使用して、分割間隔を調整することにより、ダイアモンドグリッド構造の間隔も調整する方法を学びます。
このチュートリアルでは、11.3 同様に、**Surface Diagrid.3dm**　を使用しますので、既に均等なダイヤモンド構造の結果を確認されていることと思います。
下図が、この例の GH 定義ですが、11.3 の **Surface Diagrid.ghx** に加え、緑色のコンポーネントの部分を追加します。

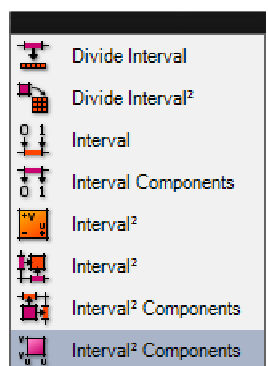
注：この最終結果を、みるには **Uneven Surface Diagrid.ghx** を開いてください。





前の例では、+Isotrim+コンポーネントを+Divide Interval+コンポーネントに接続していましたが、ここでは、まずこの接続を外すことから始めます。

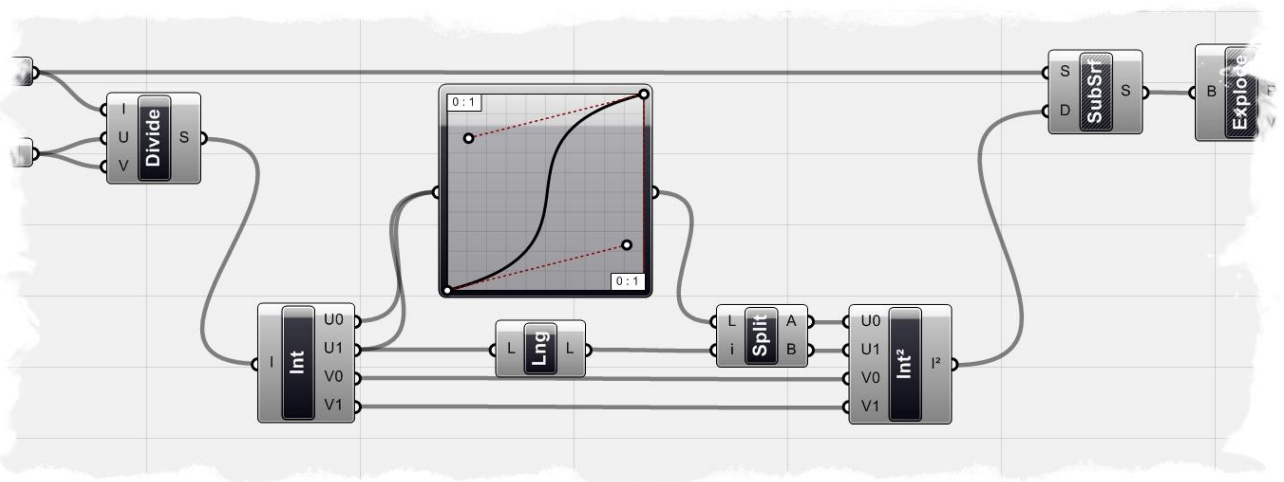
- %Isotrim+コンポーネントの D-入力を右クリックし、%Disconnect All+を選択します。この後、幾つかの新しいコンポーネントを追加することになるので、%Isotrim+コンポーネントから、右のコンポーネントをキャンバスの右方向に移動して、スペースを確保しておくと良いでしょう。.
- %Int+コンポーネント（Scalar/Interval/Interval）をキャンバスにドラッグ＆ドロップします。
  注、Version　6.0059 ではこのコンポーネントは、%Dom+コンポーネント（Scalar/Domain/Domain^2 Component）に変更されています。
- +Divide Interva+コンポーネントの、S-出力を、%Int+コンポーネント（Dom+コンポーネント）の I-入力に接続します。



- %Graph Mapper+パラメーター（Params/Special/Graph Mapper）をキャンバスにドラッグ＆ドロップします。
- %Int+コンポーネント（Dom+コンポーネント）の U0-出力を%Graph Mapper+パラメーター接続します。
- シフトキーを押しながら、U1-出力も%Graph Mapper+パラメーター接続します。
- %Graph Mapper+パラメーター上でマウスを右クリックし、グラフタイプを選択します。ここでは、+Bezier+タイプを使用します。+Bezier+タイプのグラフを、+Bezier+ハンドルを移動することでグラフの調整が出来ます。この作業を行う理由は、+Divide Interval+コンポーネントによって均等に分割されている U,V 値の分割の間隔を編集するためです。この一連の接続で U 値の値を、%Graph Mapper+パラメーターに渡され、U 値の間隔が、グラフによって再定義されます。
- %List Length+コンポーネント（Logic/List/List Length）をキャンバスにドラッグ＆ドロップします。
- %Int+コンポーネント（Dom+コンポーネント）の U1-出力を%List Length+コンポーネントの L-入力に接続します。
- %Split List+コンポーネント（Logic/List/Split List）をキャンバスにドラッグ＆ドロップします。
- %Graph Mapper+パラメーター出力を、%Split List+コンポーネントの L-入力に接続します。
- %List Length+コンポーネントの L-出力を、%Split List+コンポーネントの i-入力に接続します。
- %Int^2+コンポーネント（Scalar/Interval/Interval 2d）をキャンバスにドラッグ＆ドロップします。注、Version　6.0059 ではこのコンポーネントは、%Dom^{2+}コンポーネント（Scalar/Domain/Domain^2）に変更されています。
- %Split List+コンポーネントの A-出力を、%Int^2+コンポーネント（%Dom^{2+}コンポーネント）の U0-入力に接続します。
- 同様に Split List+コンポーネントの B-出力を、%Int^2+コンポーネント（%Dom^{2+}コンポーネント）の U1-入力に接続します。
- %Int+コンポーネント（Dom+コンポーネント）の V0-出力、V1 出力を、%Int^2+コンポーネント（%Dom^{2+}コンポーネント）の V0-入力、V1 入力にそれぞれ接続します。　%Graph %Graph Mapper+パラメーターには、%Int+コンポーネント（Dom+コンポーネント）の U0 と U1 出力のリストが接続されていますから、また 2 つのリストに戻す必要があります。全ての U 値が、%Graph Mapper+パラメーターに接続されていますから、+Divide+コンポーネントで再分割された U 値は、+Bezier+タイプのカーブで、間隔をコントロールする

ことが出来ます。V0 及び V1 出力は、そのまま、%Int^{2}+コンポーネント（%Dom^{2+}コンポーネント）の V0 及び V1 入力に接続されていますので、分割の間隔は均等のままです。V 値の分割に関しても、同じように%Graph Mapper+パラメーターに接続することで間隔のコントロールを行う事が可能です。

%Int^{2}+コンポーネント（%Dom^{2+}コンポーネント）の出力を、+Isotrim+コンポーネントに接続します。これ以降の GH 定義は、13.3 と同じですので、同様にダイアモンド構造のグリッドが作成されます。以上の一連の操作で、下図のような定義が出来上がっているはずです。



12 *Scripting とは*

GH の機能は、Scriting コンポーネントを使用して VB　DotNET もしくは、C#プログラム言語でコーディングすることで拡張することが出来ます。
将来的にはさらに多くの言語がサポートされるかもしれません。
ユーザーが書いたコードは、クラステンプレート中にダイナミックに生成され、DotNet フレームワークが供給している CLR コンパイラーによって、アセンブリにコンパイルされます。
アセンブリはコンピューターのメモリー上に存在し、Rhino が存在する限り削除される事はありません。

GH の Script コンポーネントは、Rhino DotNET SDK のクラスとファンクションへのアクセスを持っています。これらのクラスやファンクションは、プラグイン開発者が Rhino の Plug-in をビルドするものと同じものです。
実際のところ、GH は Rhino のプラグインであり、DotNET plug-in が使用しているこれらの Scripting コンポーネントにアクセスするためのものと全く同じ SDK を使用してコーディングされております。

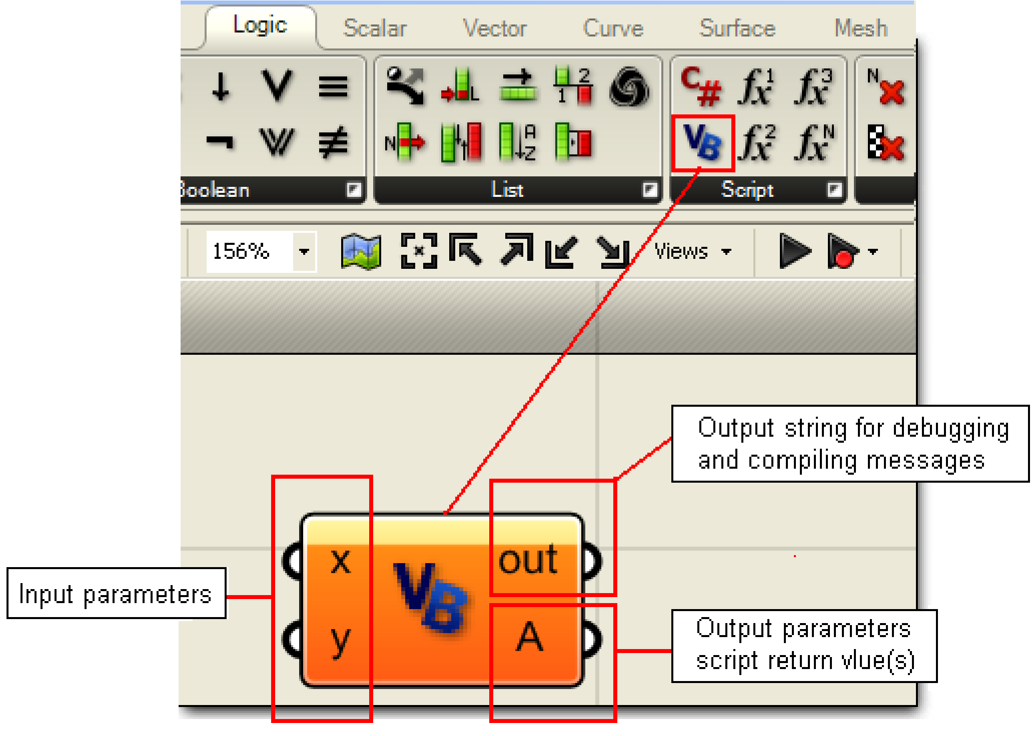
何故、Script コンポーネントを使用する必要があるのでしょうか？
ある種のケース以外、Script コンポーネントは必要としないかもしれません。
実際に使う必要があるのは、GH コンポーネントでサポートされていない機能が必要な場合です。
あるいは、フラクタルのような再帰的関数を使用する場合は必要になります。

本書は Grasshopper が VB DotNET プログラム言語において Scripting コンポーネントを、どのように使用するかの概要のみを紹介します。
最初は、Script コンポーネントのインターフェースに関して、次に VB DotNET 言語について簡単に説明し、さらに次の章では、Rhino DotNET SDK の、ジオメトリークラス、ユーティリティー関数について触れ、最後に手助けとなる情報源をリストします。

13 Scripting のインターフェース

13.1 Script コンポーネント

VB DotNet Script コンポーネントは、+Logic+タブにあります。現在、2 つの Script コンポーネントがあります。一つは Visual Basic でもう一つは C#.です。将来的には他の Script コンポーネントがサポートされるはずです。
Script コンポーネントをキャンバスに配置するには、アイコンをメニューからキャンバスにドラッグ＆ドロップします。



Script コンポーネントの初期設定では、それぞれ 2 つの入出力を持ちます。ユーザーは名称や、タイプ、そして入出力の数を変更する事が出来ます。

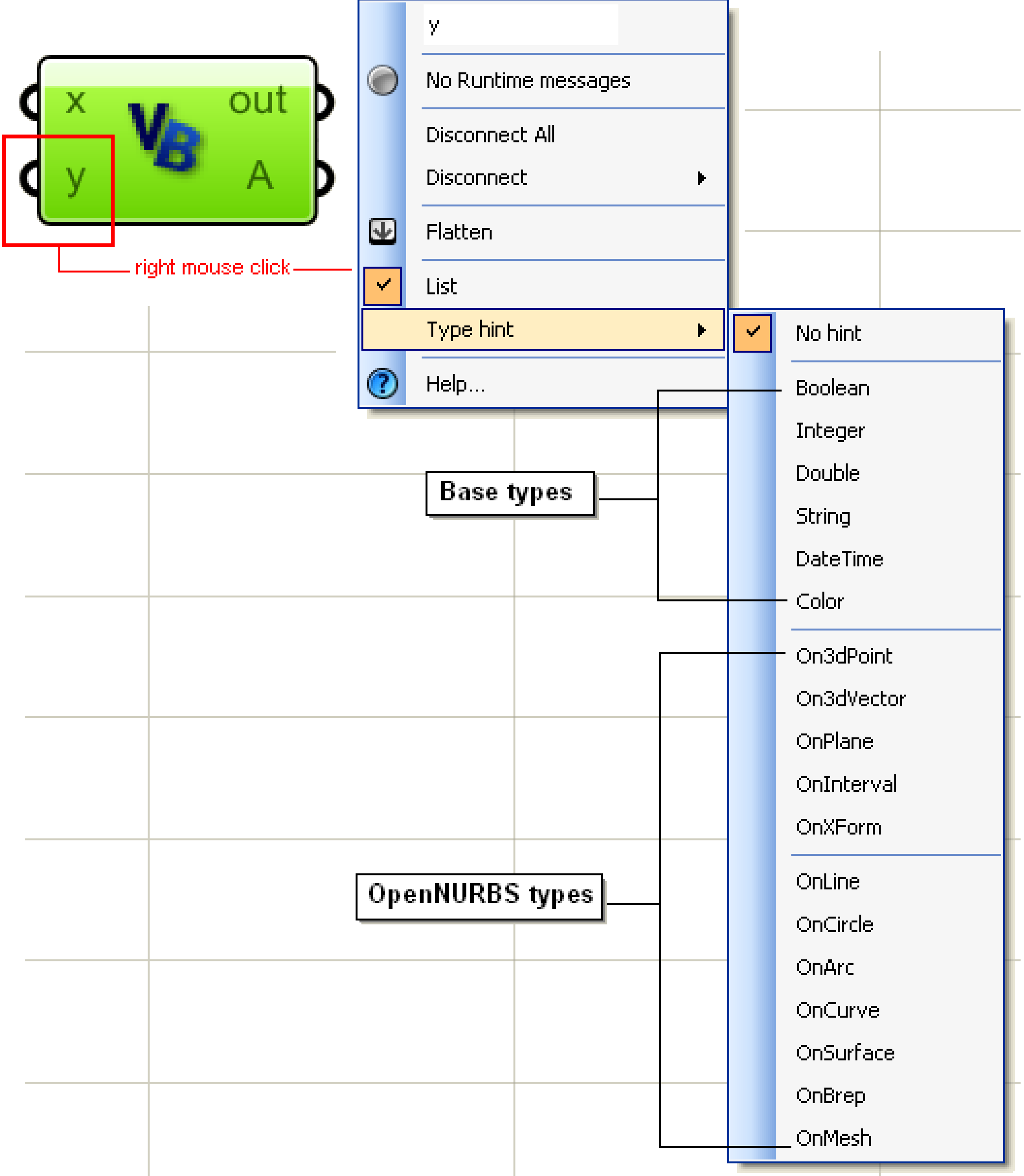
- **X**: 1 番目の入力で、object のタイプは、generic タイプです。
- **Y**: 2 番目の入力で、object のタイプは、generic タイプです。
- **Out**: コンパイルしたメッセージを文字として出力します。
- **A**: 出力として返される値です。

13.2 入力パラメーター

初期値では、x と y の 2 つの入力パラメーターが定義されています。パラメーター名を編集したり、削除、あるいは追加し、タイプを割り当てる事も出来ます。マウスを入力パラメーターに置き、右クリックすると、次のメニューが現れます。

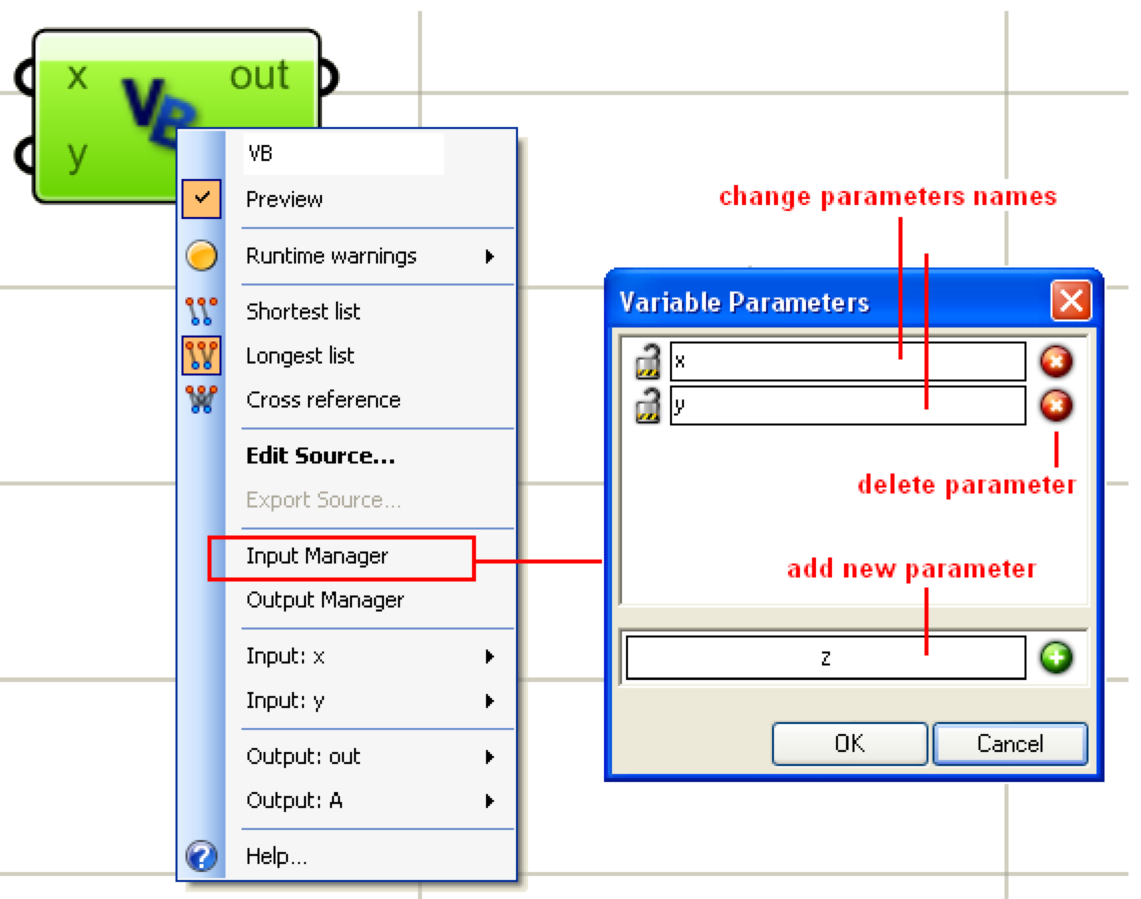
- パラメーター名:　右クリックして名称変更が出来ます。
- **Run time message**: エラーメッセージと警告が表示されます。
- **Disconnect** and **Disconnect All**. 他の GH コンポーネント同様、接続解除が出来ます。
- **Flatten**: リスト配列要素を、1 行の配列要素に変換します。

* **List**: 入力をリスト（配列）として定義します。
* **Type hint**: 入力パラメーターの初期値は、一般的な+オブジェクト+タイプに設定されています。コードを、より効果的にするためには、タイプを指定します。+On+で始まるタイプは OpenNURBS のタイプを表します。



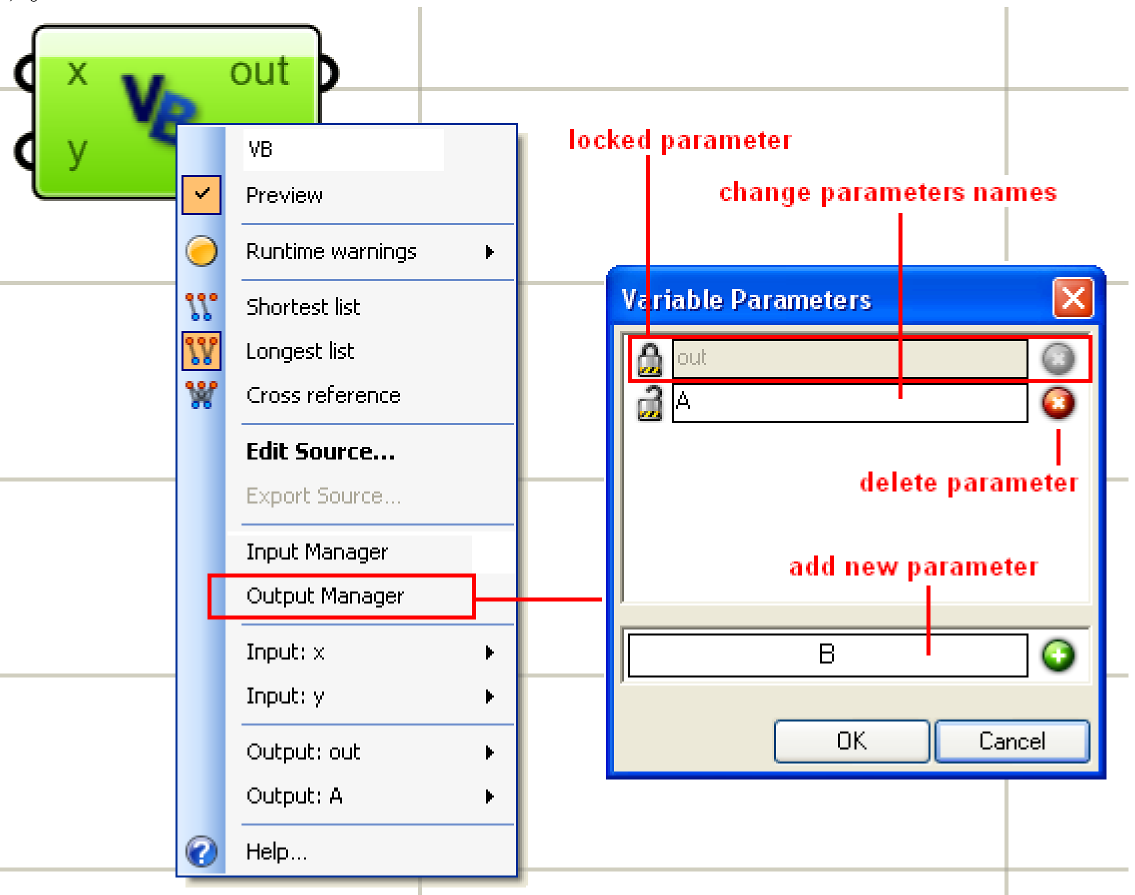
入力パラメーターは、メインコンポーネントメニューからも指定する事が出来ます。コンポーネントの中央部を、マウスで右クリックすると、入出力の詳細が見ることができます。このメニューを使用して、+Input Manager+を開き、下図のようにパラメーターの名前の変更、追加、削除をすることが出来ます。

注；Scripting のファンクションの署名（入力パラメーターとそのタイプ）は、このメニューでしか変更出来ません。実際にソースの編集時は、ファンクションのボディーは変更出来ますが、パラメーターを変更することは出来ません。


x
out
y
VB
Preview
Runtime warnings
Shortest list
Longest list
Cross reference
Edit Source...
Export Source...
Input Manager
Output Manager
Input: x
Input: y
Output: out
Output: A
Help...
change parameters names
Variable Parameters
x
y
delete parameter
add new parameter
z
OK
Cancel

13.3 出力パラメーター

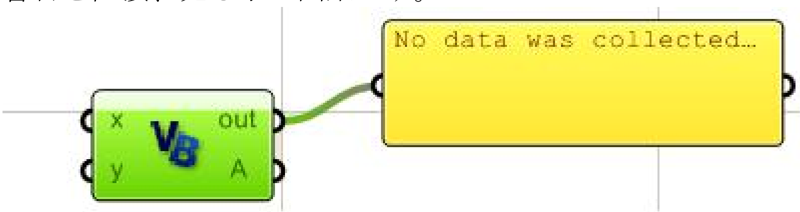
同様に、メインコンポーネントメニューで多くの出力や戻り値を定義することが出来ます。入力パラメーターと異なり出力にはタイプが存在しません。
それらは一般的な+オブジェクト+と関数として定義され、タイプや配列を定義します。
下図は+Output Manager+による出力設定のイメージです。
注：+Out+パラメーターは削除する事が出来ません。ここには、デバッグ用の文字列が表示されます。



13.4 出力ウインドゥと デバッグ情報

初期値で+out+と名づけられた出力ウインドゥには、デバッグの情報が表示されます。
ここには、コンパイルエラーと警告がリストされます。
ユーザーはコード内でデバッグをヘルプするために値をプリントすることが出来ます。
コードが正しく実行されないとき、コンパイルメッセージを注意深く読んでください。

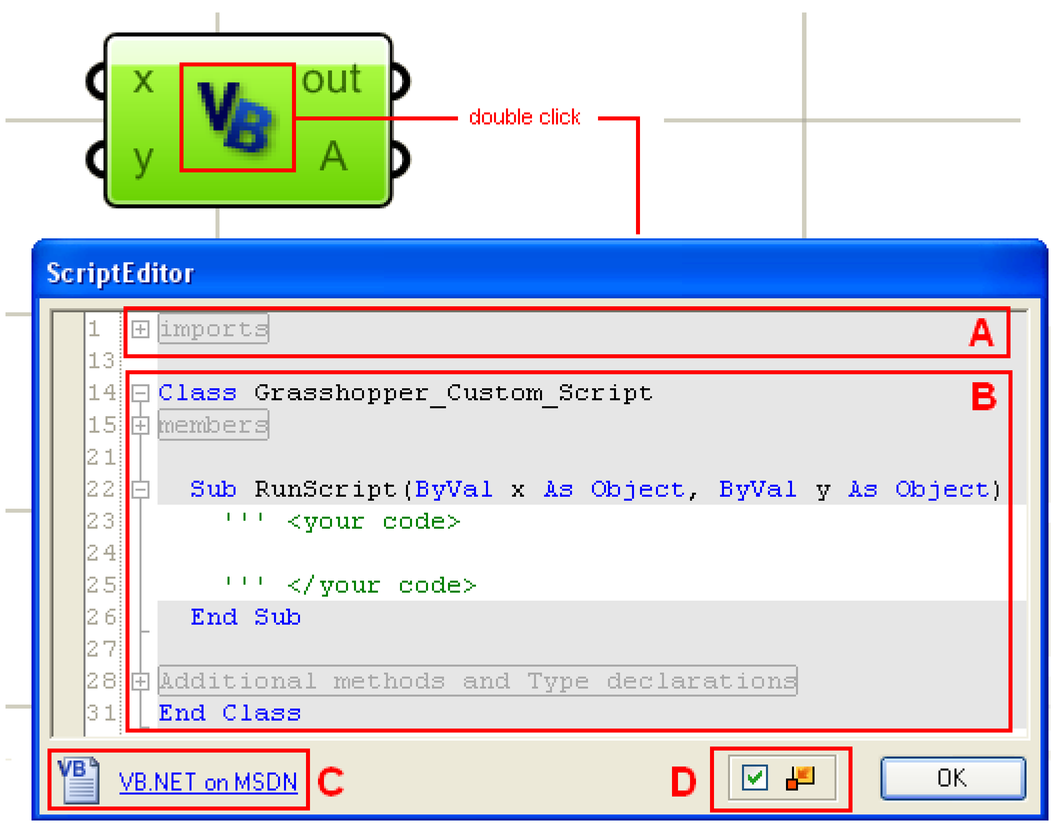
%Panel+パラメーターを、出力パラメーターに接続することによってコンパイル時のメッセージと、警告を直接、見る事が出来ます。



13.5 Script コンポーネントの内部

Script コンポーネントを開くには、コンポーネントの中央部をダブルクリックするか、メインコンポーネントメニューの+Edit Sourceõ +を選択します。
Script コンポーネントは、4 つのパーツから構成されます。

A: Imports
B: Grasshopper_Custom_Script class.
C: Microsoft developer network help on VB.NET へのリンク
D: この箇所にチェックを入れると他の Scripting コンポーネントを編集するとエディターが最小化されます。複数の Script コンポーネントを同時に開いておきたい場合は、チェックボックスを外しておきます。



A: Imports

Import は、コードで使用する外部+dll+のリストです。
その大部分は、DotNET system imports　ですが Rhino の+dll+も存在します。
それらは+RMA の openNURBS+と +RMA.Rhino+です。これらは、全ての Rhino ジオメトリー（幾何情報）とユーティリティーを含みます。また GH の固有のタイプでもあります。

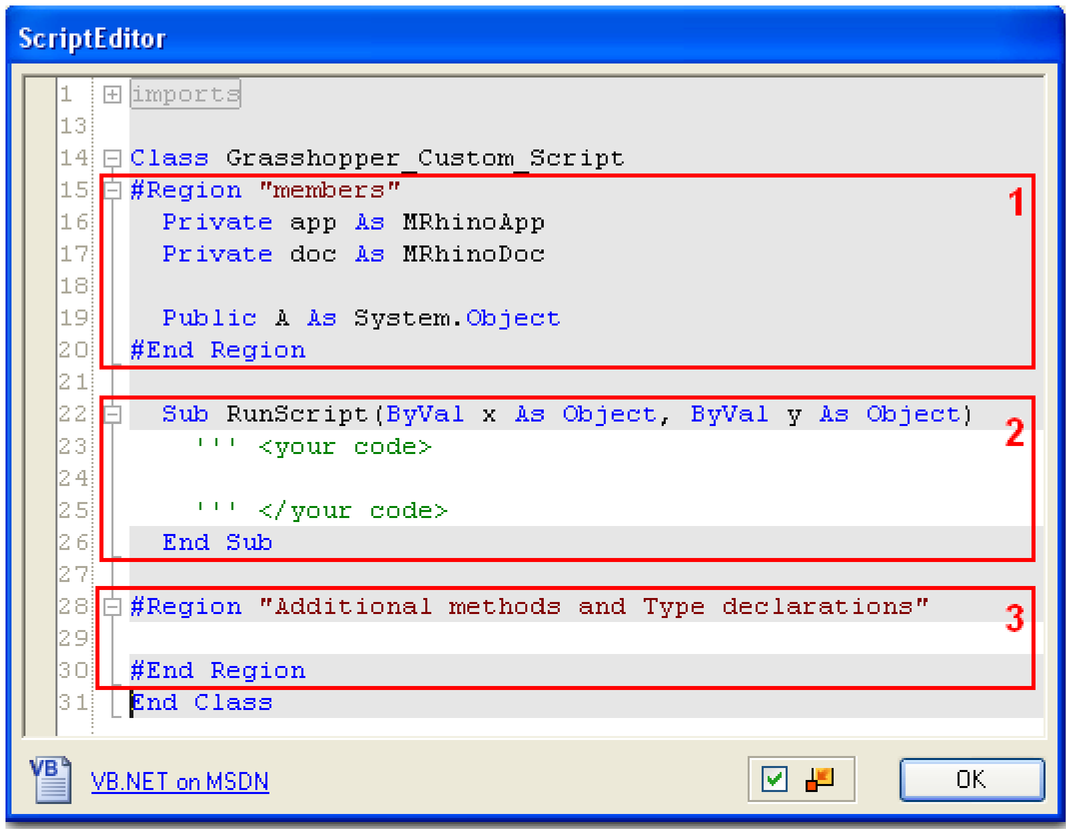
ScriptEditor

```
1  Imports System
2  Imports System.IO
3  Imports System.Drawing
4  Imports System.Drawing.Drawing2D
5  Imports System.Reflection
6  Imports System.Collections
7  Imports System.Collections.Generic
8  Imports Microsoft.VisualBasic
9
10 Imports RMA.OpenNURBS
11 Imports RMA.Rhino
12 Imports Grasshopper.Kernel.Types
13
14 Class Grasshopper_Custom_Script...
```

VB.NET on MSDN　OK

B: Grasshopper_Custom_Script クラス

Grasshopper_Custom_Script クラスは 3 つのパートから構成されます。



1. **Members**（メンバー）：メンバーは 2 つの参照を持ちます。一つは Rhino の現在の Rhino アプリケーション（app）で、もう一つはアクティブなドキュメント（doc）です。Rhino アプリケーションとドキュメントは、Rhino ユーティリティーを使用して直接、アクセスすることが出来ます。メンバー領域は、Script 機能の、戻り値か Script 機能の出力値も含

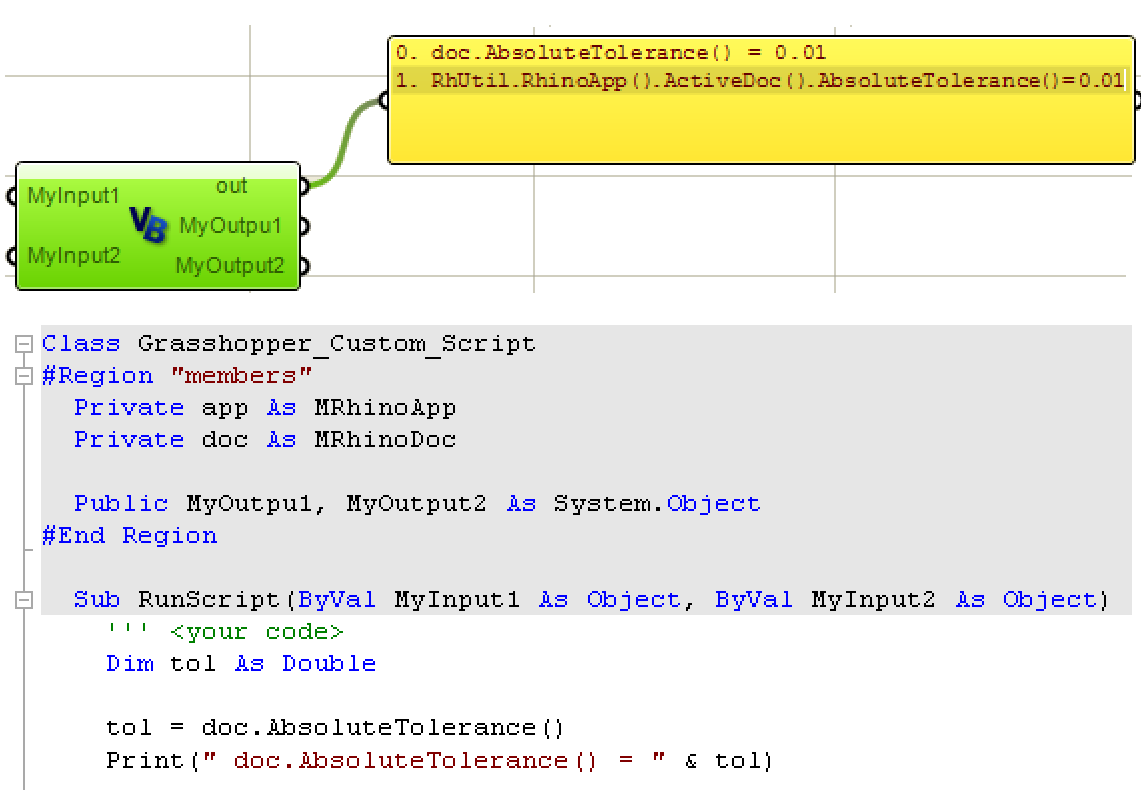
みます。戻り値は、一般的なシステムタイプで、ユーザーはこのタイプを変更することができません。

2. **RunScript**: この部分がコードを書く際の主な機能となります。

3. **Additional methods and type declarations**:追加機能やタイプを記述する箇所となります。

以下の例はドキュメントの絶対許容差にアクセスする 2 つの方法に関するサンプルです。
最初の例はドキュメントを Script コンポーネントによって参照し、2 番目の例は *RhUtil* (Rhino Utility Functions、Rhino のユーティリティー機能)を使用したものです。

注：許容差を出力ウインドーにプリントした場合、どちらも同じ結果になります。
またこの例の中では、2 つの出力(MyOutput1、と myOutput2)があり、Script クラスの中の、+members+領域で記述されています。




```
Class Grasshopper_Custom_Script
#Region "members"
  Private app As MRhinoApp
  Private doc As MRhinoDoc

  Public MyOutpu1, MyOutput2 As System.Object
#End Region

  Sub RunScript(ByVal MyInput1 As Object, ByVal MyInput2 As Object)
    ''' <your code>
    Dim tol As Double

    tol = doc.AbsoluteTolerance()
    Print(" doc.AbsoluteTolerance() = " & tol)

    tol = RhUtil.RhinoApp().ActiveDoc().AbsoluteTolerance()
    Print(" RhUtil.RhinoApp().ActiveDoc().AbsoluteTolerance()=" & tol)

    ''' </your code>
  End Sub
```

14 Visual Basic DotNET

14.1 イントロダクション

VB.NET に関してはインターネットで多く紹介されていますので、ここでは必要な項目のみを簡単に説明します。

14.2 コメント

コーディング中にコメントを出来る限り多く記述する事を推奨します。
我々はコーディングしたものに関して驚くほど早く、忘れてしまいます。
VB.NET 中でのコーディングの際、アポストロフィ以降の箇所はコメントとして理解されコンパイルの対象になりません。

```
'This is a comment… I can write anything I like!
'Really… anything
```

14.3 変数

変数はデータを含むものと考えられます。異なる変数のサイズはそれらが収容するデータのタイプによって異なります。
例えば、int32 の変数はメモリ上に 32 ビット確保し、変数の名前はコンテナの名前として予約されます。
一度、変数が定義されると残りのコードはコンテナ中のコンテンツを変数名を使用して検索します。

以下は int32 タイプの+X+と呼ばれるコンテナまたは変数の値を+10+と定義し、次に、新しい値+20+再定義した例です。
このようにして VB DotNET で使用されます。

```
Dim x as Int32 = 10
±If you print the value of x at the point, then you will get 10
x = 20
±From now on, x will return 20
```

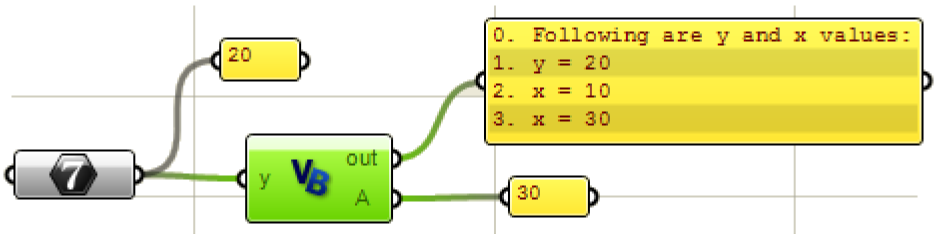
以下が一般的に使用されるタイプです。

```
Dim x as Double = 20.4          ±Dim キーワードで変数を倍精度で定義
Dim b as Boolean = True         ±Boolean として定義
Dim name as String = %Joe+      ±ストリングとして定義
```

次の GH 例は、3 つの変数を使用しています。

x: コード中で指定される整数値の変数
y:関数への入力値として送られる整数値の変数
A: 出力の変数

ここでは変数の値をコーディングによって出力ウインドーにプリントしています。
前述しましたが、内部で何が起こっているか確認することはデバッグに際して有効です。




```
Sub RunScript(ByVal y As Integer)
  'Print variables values
  Print("Following are y and x values:")

  'Print input value (y)
  Print("y = " & y)

  'Declare x as an integer variable
  Dim x As int32 = 10

  'Print x initial value
  Print("x = " & x)

  'Set x value to be whatever was there plus input y
  x = x + y
  Print("x = " & x)

  'Assign x to output
  A = x
End Sub
```

関連性のある変数名を割り当てることによりコードが読み易くなり、かつデバッグが楽に行えます。この章で、いくつかのコーディングの実習を行っていきましょう。

14.4 Arrays and Lists　（配列とリスト）

VB.NET において、配列を定義する方法がいくつかあります。

配列はサイズを定義する事もダイナミックにすることも可能です。配列は 1 次元、もしくは多次元の配列が可能で、配列のサイズをダイナミックな配列によって定義することができます。
もし事前に配列要素の数が分かっているのであれば下記のようにサイズを宣言します。

```
'1 次元配列
Dim myArray(1) As Integer
myArray (0) = 10
myArray (1) = 20
```

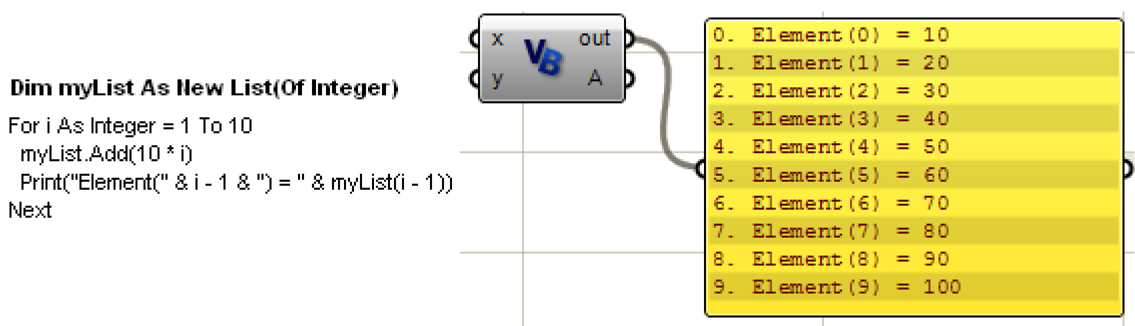
```
'2 次元配列
Dim my2DArray (1,2) As Integer
my2DArray (0, 0) = 10
my2DArray (0, 1) = 20
my2DArray (0, 2) = 30
my2DArray (1, 0) = 50
my2DArray (1, 1) = 60
my2DArray (1, 2) = 70
```

```
'Declare and assign values
Dim myArray() As Integer  = {10,20}
```

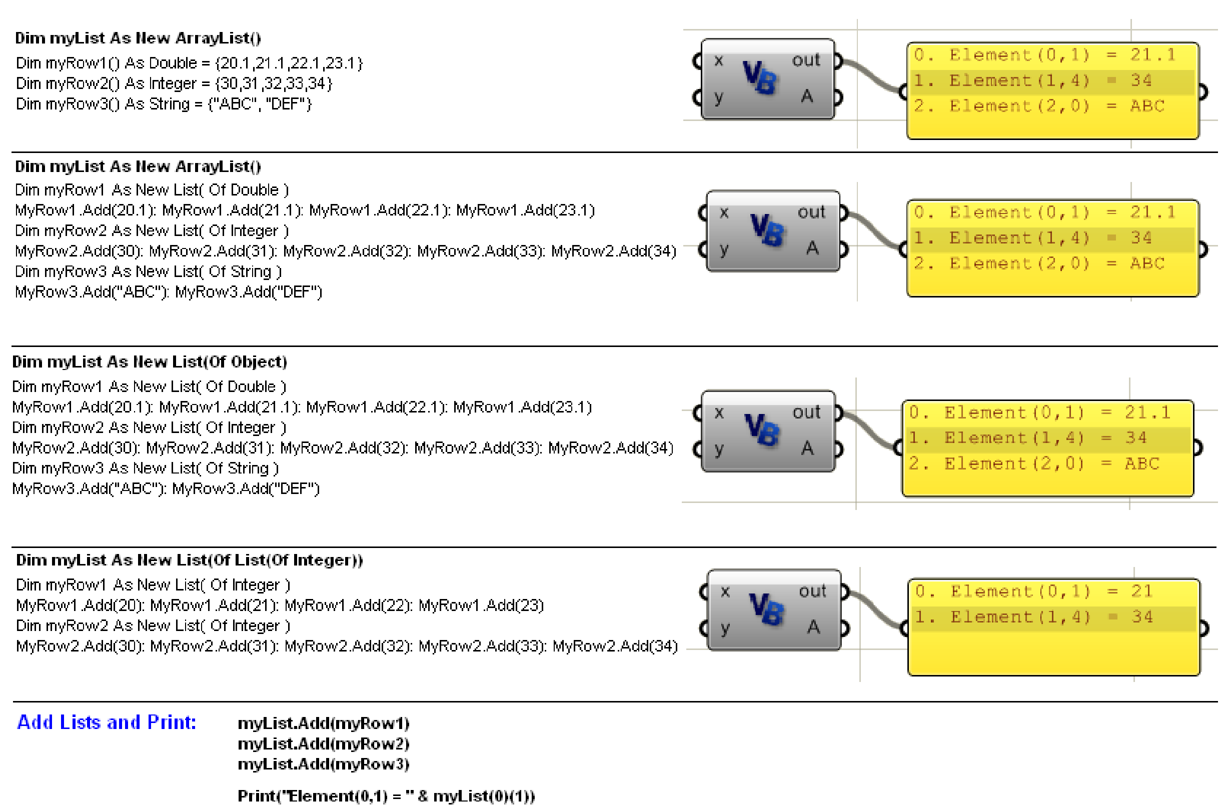
```
'Declare and assign values
Dim my2DArray(,)As Integer = {{10,20,30},{40,50,60}}
```

VB.NET では配列は+0+を基準とします。従って、配列のサイズを+9+とした場合、配列は 10 個の要素を持つ事になります。同じ事が多次元配列にも適用されます。

1 次元配列の場合、新しいリストを下記のように宣言することが出来、そのリストに要素を追加することが出来ます。



ネストされた配列のリストを使用してダイナミックな多次元配列を作る事が出来ます。
以下は、異なるコーディングの例です。



Add Lists and Print:

```
myList.Add(myRow1)
myList.Add(myRow2)
myList.Add(myRow3)

Print("Element(0,1) = " & myList(0)(1))
Print("Element(1,4) = " & myList(1)(4))
Print("Element(2,0) = " & myList(2)(0))
```

14.5 Operators　（オペレーター：演算子）

VB.NET では、既に組み込みの演算子が多数用意されています。演算子は一つ以上のオペランドを操作します。
下記は一般的なオペレーターです。

Type	Operator	Description
算術演算子	**^**	べき乗　ある数をべき乗の数だけ繰り返し掛け合わせる
	*****	乗算
	/	除算
	****	除算を行い、整数値を返す。
	Mod	余り
	+	加算
	-	減算
代入演算子	**=**	基本的な代入処理　例：a = b
	^=	複合的な代入処理　例：a = a **^** b
	***=**	複合的な代入処理　例：a = a ***** b
	/=	複合的な代入処理　例：a = a **/** b
	\\=	複合的な代入処理　例：a = a **** b
	+=	複合的な代入処理　例：a = a **+** b
	-=	複合的な代入処理　例：a = a **–** b
	&=	複合的な代入処理　例：a = a **&** b
比較演算子	**<**	～より小さい
	<=	～より小さいか等しい
	>	～より大きい
	>=	～より大きいか等しい
	=	等しい
	<>	等しくない
文字列連結演算子	**&**	文字列を連結する
	+	文字列を連結する
論理演算子	**And**	論理積
	Not	否定
	Or	論理和（または）
	Xor	排他的論理和

14.6 Conditional Statements　（制御文）

制御文を使用して条件に合うものだけを実行することが出来ます。
制御文はほとんどの場合、+if+文の後に+条件+、そして+Then+の後に+実行するコードのブロックを指定+となります。

```
'If 文が 1 行のステートメントで終了する場合は、End If 文は必要有りません。
If x < y Then x = x + y

'行が複数になる場合は、コードブロックを終了させるために End If 文が必要になります。
If x < y Then
   x = x + y
End If
```

%Else If+文を使用すると、条件に当てはまらない場合について実行するコードのブロックを選ぶ事が出来ます。

```
If x < y Then
   x = x + y        'このラインを実行し、End  If 文後へ抜ける。
Else If x > y Then
   x = x . y        'このラインを実行し、End  If 文後へ抜ける。
Else
   x = 2*x          'このラインを実行し、End  If 文後へ抜ける。
End If
```

また+Select Case+文を使用して、条件の値に従って、対応する実行するコードのブロックを指定する事が出来ます。

```
Select Case index
   Case 0           ±index=0  のとき次のラインを実行。そうでない場合は次の Case 文に飛ぶ。
      x = x * x
   Case 1
      x = x ^ 2
   Case 2
      x = x ^ (0.5)
End Select
```

14.7 Loops　（ループ処理）

ループを使用することにより、そのループ条件を満たすまでコードブロックを繰り返し実行させる事が出来ます。
ループにはいくつか異なる種類のループが有ります。
ここでは、最も一般的に使用されている 2 つのタイプを紹介します。

"For … Next" ループ
%For ō . Next+ループ文は最も良く使用されるループで、以下のような構造から成ります。

```
For < インデックス = 開始値> To <終了値> [Step <間隔>]

   'ループ文の始まり
   [ ループ内で実行される条件文]

  [ Exit For ]  'オプション:  ループを終了

   [ 他の条件文]

   [ Continue For ]   'オプションフ:  残されたループの実行をスキップする。
```

```
  [他の条件文]

  'For loop body ends here (just before the "Next")
  'Next means: go back to the start of the for loop to check if index has passed end_value
  'If index passed end_value, then exit loop and execute statements following "Next"
  'Otherwise, increment the index by "step_value"
Next
[他の条件文]
```

次の例は配列に記入された都市名を、Loop 文を使用して出力した例です。

```
'都市名の配列定義
Dim places_list As New List( of String )
places_list.Add( "Paris" )
places_list.Add( "NY" )
places_list.Add( "Beijing" )

'ループのインデックス
Dim i As Integer
Dim place As String
Dim count As Integer = places_list.Count()

'ループを"0"から始め、配列の数から-1 した数字までループを実行。この例では配列の数は、3 個で、
最後の配列のインデックスは、"2")
For i=0 To count-1
   place = places_list(i)
   Print( place )
Next
```

For～Next 文を使用してインデックスを使用することなく配列データを繰り返し出力することが出来ます。この条件文で上記を下記のように書き直すことが出来ます。

```
Dim places_list As New List( of String )
places_list.Add( "Paris" )
places_list.Add( "NY" )
places_list.Add( "Beijing" )

For Each place As String In places_list
   Print( place )
Next
```

"While ... End While" ループ

%While õ　End While+ループも幅広く使用されるループです。
このループの構造は下記の通りです。

```
While < ある条件が 真の場合 >

  'While loop body starts here
  [ループの中で実行される宣言文]

 [ Exit While ]         'Optional to exit the loop

  [他の宣言文]

  [ Continue While ] 'optional to skip executing the remaining of the loop statements.

  [他の宣言文]

  'While loop body ends here
  'Go back to the start of the loop to check if the condition is still true, then execute the body
  'If condition not true, then exit loop and execute statements following "End While"
```

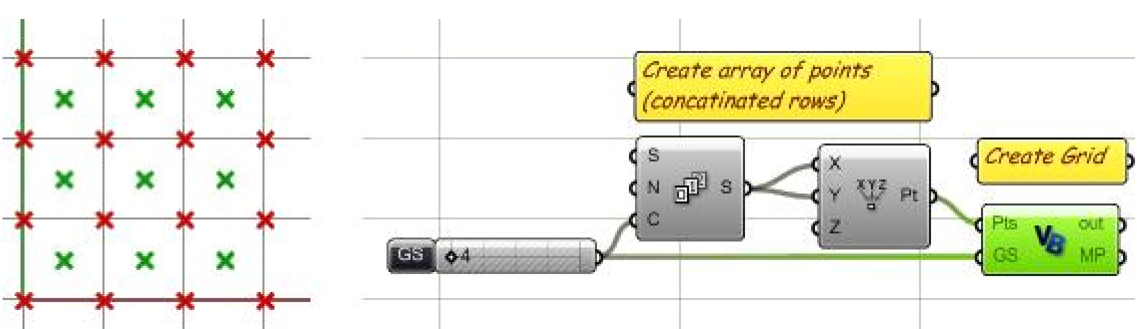
```
End While
[ループの間の宣言文]
```

以下がループ文を使用した例です。

```
Dim places_list As New List( of String )
places_list.Add( "Paris" )
places_list.Add( "NY" )
places_list.Add( "Beijing" )

Dim i As Integer
Dim place As String
Dim count As Integer = places_list.Count()

i = 0
While i < count              '(i<count) evaluates to true or false
   place = places_list(i)
   Print( place )
   i = i + 1
End While
```

14.8 ループのネスト

ネストされたループとは、その body がさらに別のループを持つものです。
例えばグリッドの点群がある場合、それぞれの点をリストから取る場合、ネストされたループを使う必要があります。次の例は 1 次元配列の点群を、2 次元配列の点群グリッドに変換する方法です。次にさグリッドの中点を見つけます。



この script は 2 つの部分から成ります。

- 1 次元配列を 2 次元配列(グリッド)に変換
- グリッドから格子の中点を見つける。

どちらの場合も、ループのネストを使用します。

```
Sub RunScript(ByVal Pts As List(Of On3dPoint), ByVal GS As Integer)

  'Create a grid of points
  Dim Grid As New ArrayList()

  Dim i As Integer
  Dim j As Integer

  'Nested loop to covert 1D array to 2D grid
L1 For i = 0 To Pts.Count() - 1 Step GS
    'Declare a row of points
    Dim Row As New List( Of On3dPoint )
L2  For j = i To i + GS - 1
      'Get a reference od the point
      Dim pt As On3dPoint
      pt = Pts(j)

      'Add point to the row
      Row.Add(pt)
    Next
    'Add row to the grid
    Grid.Add(Row)
  Next

  'Process the grid to find mid points of cells
  Dim mid_points As New List( Of On3dPoint )
L1 For i = 1 To Grid.Count() - 1
    'Get first and second rows
    Dim Row0 As List( Of On3dPoint )
    Row0 = Grid(i - 1)
    Dim Row1 As List( Of On3dPoint )
    Row1 = Grid(i)

L2  For j = 1 To Row0.Count() - 1
      Dim mid_pt As New On3dPoint
      mid_pt = (Row0(j-1) + Row0(j) + Row1(j-1) + Row1(j)) / 4
      mid_points.Add(mid_pt)
    Next
  Next

  'Assign mid point to output
  MP = mid_points
End Sub
```

14.9 Sub プロシージャと Function プロシージャ

RunScript は全ての Script コンポーネントが使用するメインファンクションです。
下記が Grasshopper における Script コンポーネントの初期状態です。

```
Sub RunScript(ByVal x As Object, ByVal y As Object)
        '< your code >
End Sub
```

Sub… End Sub: この間で、コードが実行されます。
“RunScript”: Sub プロシージャの名前です。
“(…)”: 括弧内に、入力パラメータが記されます。
“ByVal x As Object,…”: 入力パラメーター定義です。

それぞれの入力パラメーターは下記を定義する必要があります。

- **ByRef** or **ByVal**: パラメーターが、他を参照する（ByRef）か、数値（ByVal）かを指定します。
- パラメーターの名前
- **“As”** キーワードでパラメーターのタイプを指定します。

Grasshopper の RunScript sub ステートメントの入力パラメーターの呼び出しは数値（***ByVal*** キーワード）となります。
これらは、最初の入力のコピーであり、これらを Script 内で変更してもその変更は反映されません。
しかし、Script 内で sub ファンクションを追加して定義し、そこでパラメーターの呼び出しに (***ByRef***)を指定することは可能です。
パラメーターを参照で呼び出すことはファンクションが存在する間、数値で呼び出された初期の値を変更します。

RunScript のボディー内で全てのコーディングを行う事は可能ですが、必要に応じて外部の sub プロシージャと Function プロシージャを定義する事が可能です。
外部のファンクションを使用する理由は下記の通りです。

- メインのファンクションコードを単純にすること.
- コーディングを読み易くする。
- 再利用する共通のファンクションを隔絶しておく。
- 再帰計算等、特別なファンクションを定義する。

Sub プロシージャと Function プロシージャの違いを説明します。
Sub プロシージャは戻り値を必要としない場合に定義します。
Functio プロシージャは一つの結果を返す事を許可します。基本的に Function 名に対して値を割り当てます。

```
Function AddFunction( ByVal x As Double, ByVal y As Double )
        AddFunction = x + y
End Function
```

次は戻り値を持つための Function を持つ必要のない例です。
Sub によって、入力パラメーター呼び出しを“ByRef”にして、参照値を呼び出すことが出来ます。
ここでは、rc が戻り値として使用されています。

```
Sub AddSub( ByVal x As Double, ByVal y As Double, ByRef rc As Double )
```

```
        rc = x + y
    End Sub
```

下記はどのようにcallファンクションが、Functionプロシージャと Subプロシージャを使用しているかの例です。

```
Dim x As Double = 5.34
Dim y As Double = 3.20
Dim rc As Double = 0.0
'Can use either of the following to get result
rc = AddFunction( x, y )     'rc に、ファンクションの結果を割り当てている例
AddSub( x, y, rc )           'rc が、参照として送られ追加の結果を持つ例t
```

ネストされたループで点群からグリッド（格子）を作成し、後に、その中点を計算する例を見ていきます。
2つのファンクションは外部のsubプロシージャによって明確に分けてあり、コードを再利用可能です。

```
Sub RunScript(ByVal Pts As List(Of On3dPoint), ByVal GS As Integer)

    'Create a grid of points
    Dim Grid As New ArrayList()

    'Call grid function
1   Call CreateGrid(Pts, Grid, GS)

    'Call mid points function
    Dim mid_points As New List( Of On3dPoint )
2   Call FindMidPoints(Grid, mid_points)

    'Assign mid point to output
    MP = mid_points
End Sub

#Region "Additional methods and Type declarations"

  'Function to convert 1d array to 2d array
1 Sub CreateGrid( ByVal Pts As List(...

  'Function to find grid mid points
2 Sub FindMidPoints(ByVal Grid As ArrayList, mid_points As List(...

#End Region
End Class
```

以下に2つのsubプロシージャの拡張をした例を示します。

```
#Region "Additional methods and Type declarations"

  'Function to convert 1d array to 2d array
  Sub CreateGrid( ByVal Pts As List(Of On3dPoint), ByRef Grid As ArrayList, ByVal GS As Integer )

    Dim i As Integer
    Dim j As Integer

    For i = 0 To Pts.Count() - 1 Step GS
      'Declare a row of points
      Dim Row As New List( Of On3dPoint )
      For j = i To i + GS - 1
        'Get a reference od the point
        Dim pt As On3dPoint
        pt = Pts(j)

        'Add point to the row
        Row.Add(pt)
      Next
      'Add row to the grid
      Grid.Add(Row)
    Next

  End Sub

  'Function to find grid mid points
  Sub FindMidPoints(ByVal Grid As ArrayList, mid_points As List(Of On3dPoint ))

    Dim i As Integer
    Dim j As Integer

    For i = 1 To Grid.Count() - 1
      'Get first and second rows
      Dim Row0 As List( Of On3dPoint )
      Row0 = Grid(i - 1)
      Dim Row1 As List( Of On3dPoint )
      Row1 = Grid(i)

      For j = 1 To Row0.Count() - 1
        Dim mid_pt As New On3dPoint
        mid_pt = (Row0(j - 1) + Row0(j) + Row1(j - 1) + Row1(j)) / 4
        mid_points.Add(mid_pt)
      Next
    Next

  End Sub
#End Region
```

14.10 Recursion（再帰処理）

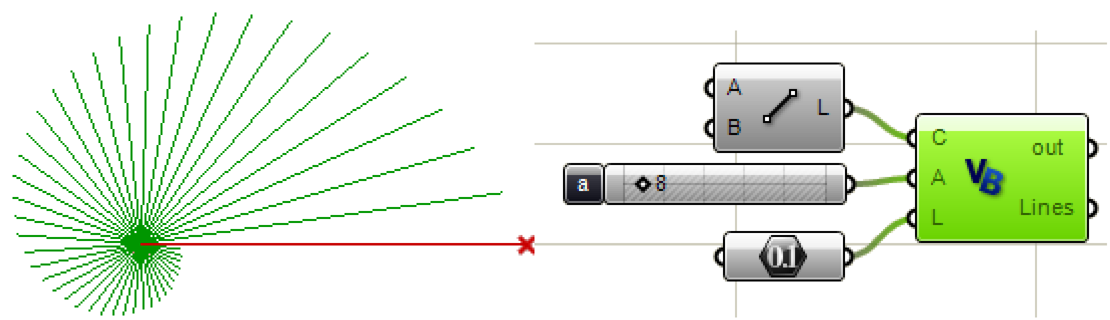
再帰処理は、ある条件下で、自分自身を呼び出すプロシージャです。
再帰は、一般的にデータの検索、細分割やジェネレーティブ・システム（パターンの繰り返し生成）等に使用されます。
ここでは、どのように再帰処理が機能するかを見ます。
再帰処理についてのさらなる例は、Grasshopper の WIKI や、ギャラリーをチェックしてください。

下記の例は、小さなライン入力に対して与えられた角度で、回転し、ライン長が指定長さの範囲内で実行していきます。
入力パラメーターは、

- 開始ライン (C).
- スライダーによる角度指定がラジアンに変換(A).されます。
- 最小長さ (L) . 実行時の最終値.

出力:

- ラインの配列



直線の長さを比較し、再帰的に繰り返すことによってこの問題を解きます。
下記の例で、+DivideAndRotate+sub プロシージャによって再帰的に結果を導いています。
（直線の長さを、再帰的に 0.95 倍し、L の指定入力値より小さくなるとプロシージャを終了）

- Sub プロシージャを抜けるための条件
- 同じ function を呼ぶ call ステートメント（自分自身を再帰的に呼び出す）
- %AllLines+(直線の配列) は新しい直線を、リストに追加していきます。

```
Sub RunScript(ByVal C As OnLine, ByVal A As Double, ByVal L As Double)

  'Declare all lines
  Dim AllLines As New List( Of OnLine )

  'Call recursive function
  Call DivideAndRotate(Line, AllLines, A, L)

  'Assign return value
  Lines = AllLines
End Sub

#Region "Additional methods and Type declarations"

  Sub DivideAndRotate(ByVal Line As OnLine,
                      ByRef AllLines As List(Of OnLine),
                      ByVal angle As Double,
                      ByVal MinLength As Double)

    'Check the stopping condition
    If Line.Length() < MinLength Then Exit Sub

    'Take a portion of the line
    Dim new_line As New OnLine(Line)
    Dim end_pt As New On3dPoint
    end_pt = new_line.PointAt(0.95)

    new_line.To = end_pt

    'Rotate
    new_line.Rotate(angle, OnUtil.On_zaxis, Line.from)

    AllLines.Add(new_line)

    'Call self
    Call DivideAndRotate(new_line, AllLines, angle, MinLength)

  End Sub

#End Region
```

次の例は+while+ループを使用した再帰的な解の例です。

```
Sub RunScript(ByVal C As OnLine, ByVal A As Double, ByVal L As Double)

    'Declare all lines
    Dim AllLines As New List( Of OnLine )

    'Find current length
    Dim current_L As Double = C.Length()

    Dim new_line As OnLine
    new_line = C

    'Loop until length is less than min length
    While current_L > L
      'Generate the new line
      new_line = DivideAndRotate(new_line, A)

      'Add to list
      AllLines.Add(new_line)

      'Stopping condition
      current_L = new_line.Length()
    End While

    'Assign return value
    Lines = AllLines
  End Sub

#Region "Additional methods and Type declarations"

 Function DivideAndRotate(ByVal L As OnLine, ByVal A As Double) As OnLine

    'Take a portion of the line
    Dim new_line As New OnLine(L)
    Dim end_pt As New On3dPoint
    end_pt = new_line.PointAt(0.95)

    new_line.To = end_pt

    'Rotate
    new_line.Rotate(A, OnUtil.On_zaxis, L.from)

    'Function return
    DivideAndRotate = new_line

 End Function

#End Region
```

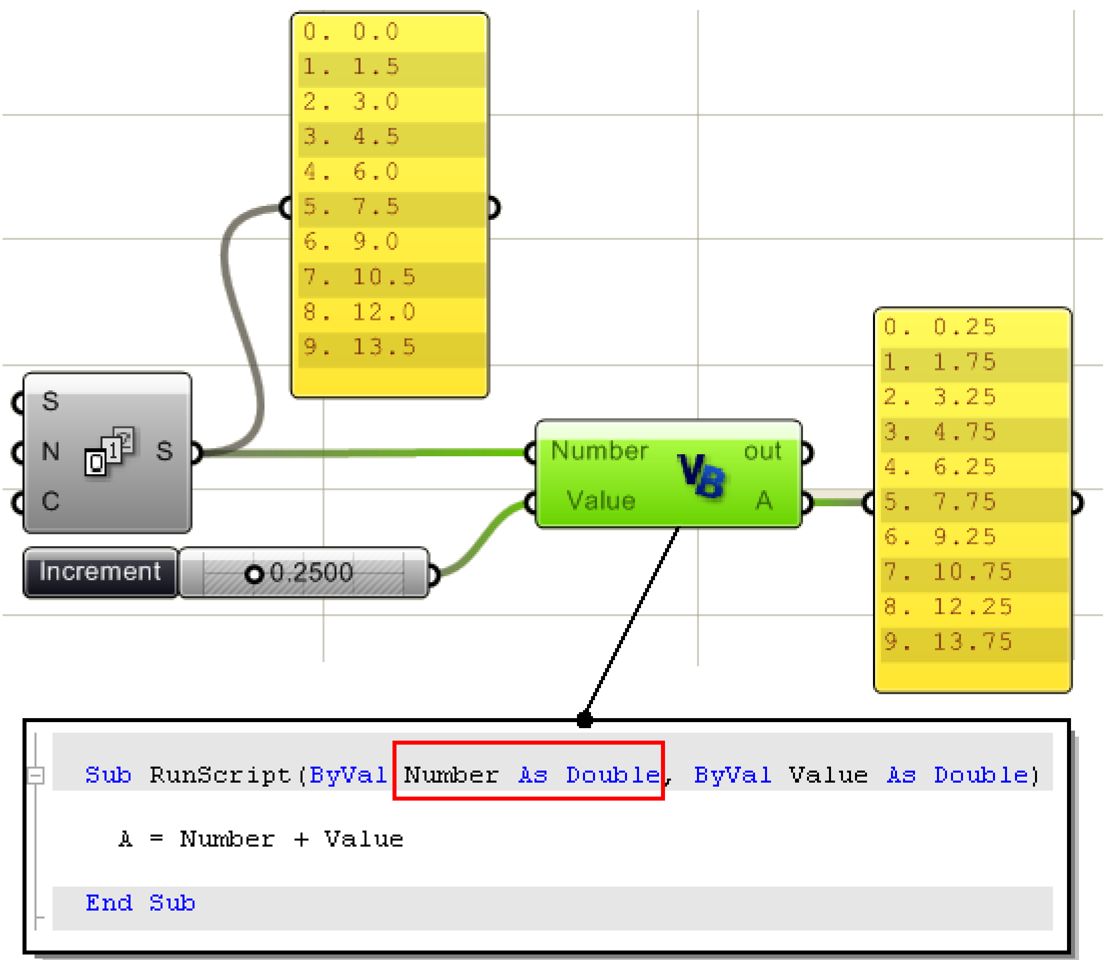
14.11 Processing Lists in Grasshopper（Grasshopper におけるリスト処理）

Grasshopper スクリプトは、リスト入力の処理を 2 つの方法で行う事が出来ます。

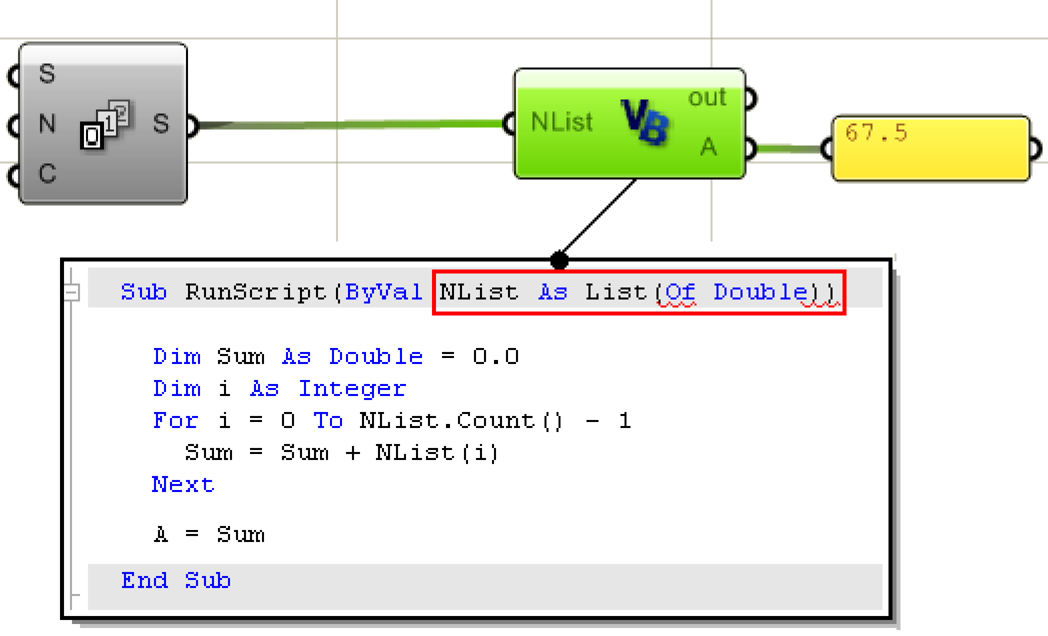
1. 1 回につき、一つの入力を処理する方法（コンポーネントは、入力の配列の数だけコールされます。）
2. 全ての入力値を一度に処理する方法（コンポーネントは、一度しかコールされません。

もし、リストにあるそれぞれの要素を独立して処理する必要があるのであれば、最初のアプローチがやりやすいでしょう。
例えば、処理したい 10 の配列をもったリストがあるのであれば、最初のアプローチが適しています。
もし、全ての要素の合計を取るのであれば、後者のアプローチで行い、全てのリストを一つの入力として処理します。

次のリストは、最初の方法でリストデータを一度に処理する方法です。
このケースでは、RunScript ファンクションは 10 回、コールされています。
ここで重要な事は入力パラメーターが、浮動小数点で定義されていることです。
その次の例では、入力パラメーターは浮動小数点のリストで定義されています。



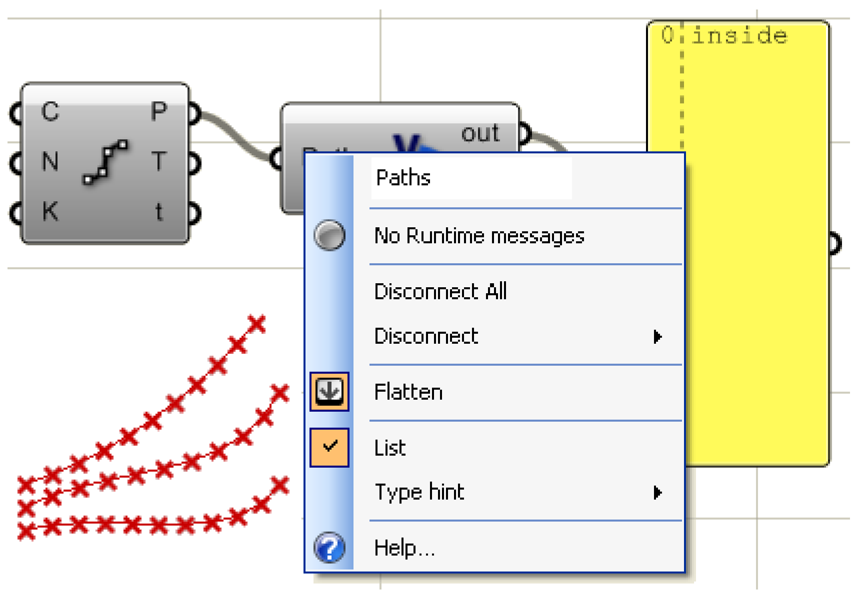
次の例では、数値のリストとして入力しています。この定義を行うのは、入力パラメーターにおいて右クリックし、+List+の項にチェックを入れます。
RunScript　ファンクションは一度だけ、コールされて実行されます。



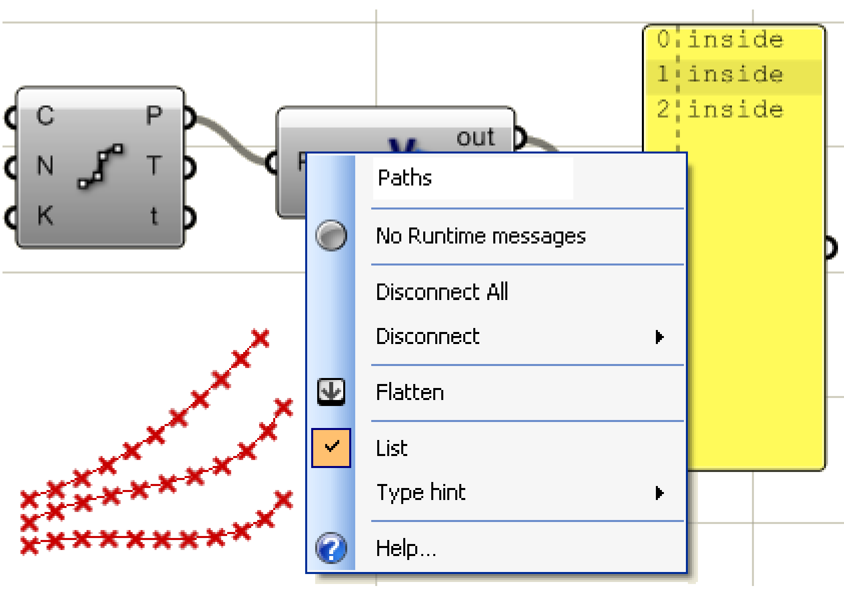
14.12 Processing Trees in Grasshopper（Grasshopper におけるツリー構造の処理）

ツリー構造（または、多次元データ）は、一つの要素単位にあるいは、一つの分岐（branch）単位に、あるいは全てのパス（path）を一度に処理する事が出来ます。
例えば、3 つのカーブをそれぞれ 10 のセグメントに分割するのであれば、それぞれ、11 のポイントを持つ 3 つのツリー構造を持った枝もしくは、パスに分解します。
次のような処理方法があります。

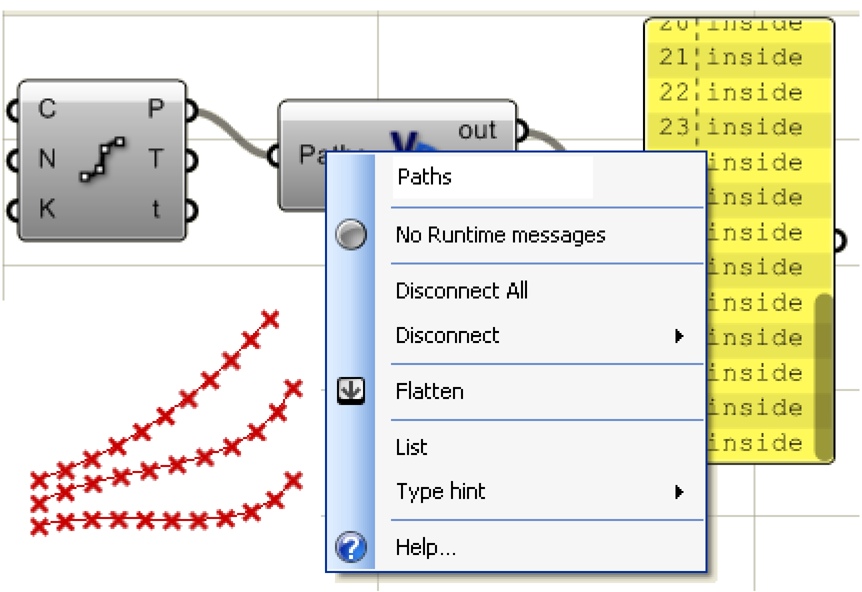
A: もし+Flatten+と、+List+がチェックされている場合、コンポーネントは 1 度だけコールされて、全てのポイントのフラットなリスト（階層が無い）が出力されます。



B: %List+のみがチェックされている場合、コンポーネントは 3 回コールされ、それぞれのカーブを分割したポイントのリストが出力されます。



C: なにも、チェックしなかった場合、ファンクションはそれぞれ分割されたポイント毎にコールされます。（この例では、33 回：Divide コンポーネントの初期値は+10+で 11 個のポイントが生成される。）



This is the code inside the VB component. Basically just print the word %inside+to indicate that the component was called:

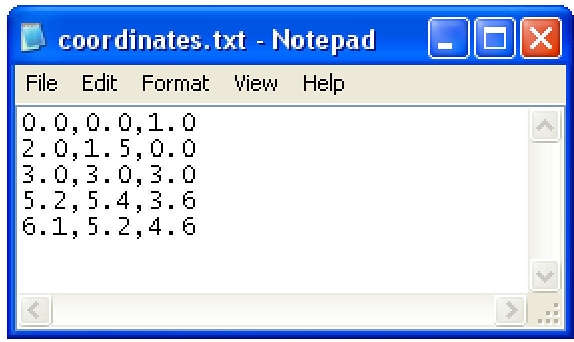
```
Sub RunScript(ByVal Paths As Object)

  Print("inside")

End Sub
```

14.13 File I/O　（ファイルの入出力）

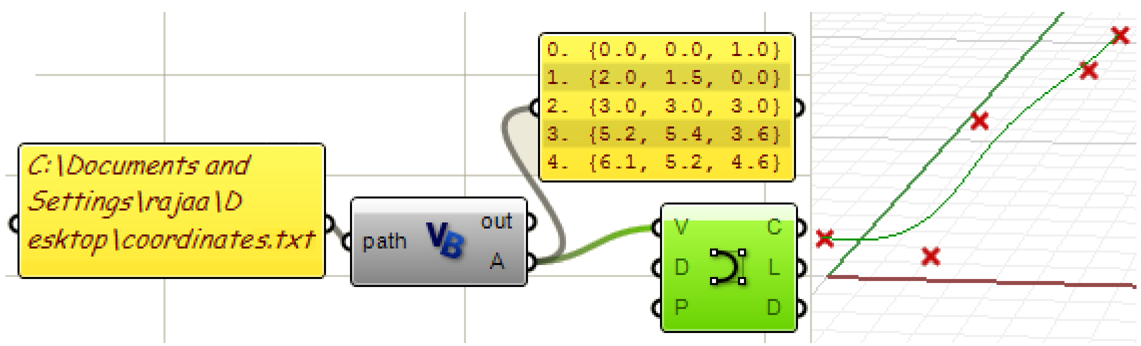
VB.NET では、ファイルの読込・書出しにいくつかの方法がインターネットや印刷物としてあります。
一般的にファイルの読込は下記のようなプロセスとなります。

- ファイルを開く（パスの指定が必要）
- ストリングデータを読む（全てのストリングデータをライン毎に）
- いくつかの区切り文字を使用するためにストリングをトークンに分割する
- トークンを変換（cast）する。（この場合は、浮動小数点）
- 結果を保存

テキストファイルを読み込み、ポイント出力を行う簡単な例を紹介します。
下記のようなテキストファイルを使用して、ライン毎に読込み、最初の値をポイントの x、2 番目を y、3 番目を z 値とし、それらの点群からカーブのコントロールポイントとします。



VB コンポーネントは、指定したパスのファイルをストリング入力として受け取り On3dPoints（Rhino の点）として出力する事ができます。



Here is the code inside the script component. There are few error trapping code to make sure that the file exits and has content:

```
Sub RunScript(ByVal path As String)

    'Check if file exists
    If (Not IO.File.Exists(path)) Then
      Print("Exit without reading")
      Return
    End If

    'Read the file
    Dim lines As String() = IO.File.ReadAllLines(path)

    'Check that file is not empty
    If (lines Is Nothing) Then
      Print("File has no content")
      Return
    End If

    'Declare list of points
    Dim pts As New List(Of On3dPoint)

    'Loop through lines
    For Each line As String In lines
      'Tokenize line into array of strings separated by ","
      Dim parts As String() = line.Split(",".ToCharArray())

      'Make sure that each line has exactly 3 values
      If UBound(parts) <> 2 Then Continue For

      'Convert each coordinate from string to double
      Dim x As Double = Convert.ToDouble(parts(0))
      Dim y As Double = Convert.ToDouble(parts(1))
      Dim z As Double = Convert.ToDouble(parts(2))

      pts.Add(New On3dPoint(x, y, z))
    Next

    A = pts

End Sub
```

15 *Rhino .NET SDK*

15.1 概要

Rhino .NET SDK は、OpenNURBS ジオメトリーとユーティリティーへのアクセスを供給します。
下記から Rhino .NET SDK をダウンロードするとヘルプファイルが付いてきます。
http://en.wiki.mcneel.com/default.aspx/McNeel/Rhino4DotNetPlugIns.html

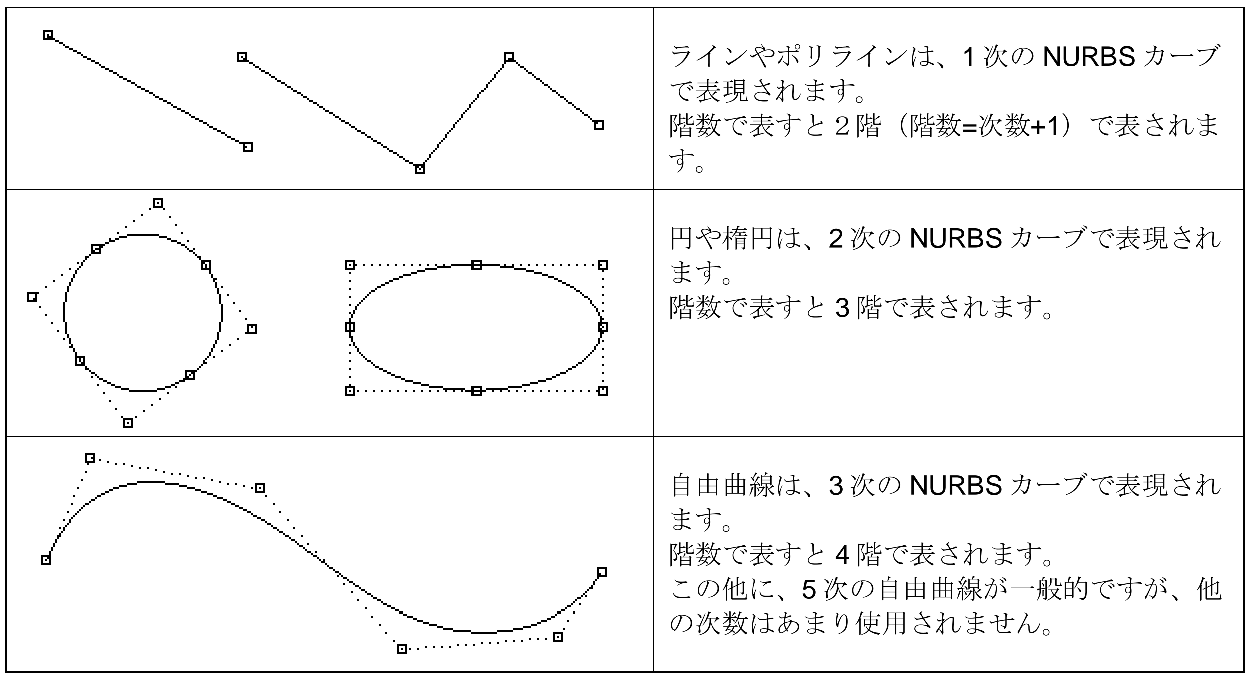
このセクションでは、Rhino のジオメトリークラスを扱う SDK の一部とユーティリティーについてフォーカスし、Grasshopper の VBScript コンポーネントを使用して、ジオメトリーを作成し操作する例を紹介します。

15.2 NURBS の理解

Rhino は NURBS モデラーで、カーブとサーフェスを Non-Uniform Rational Basis Spline（非均一な有理 B スプライン、通称 NURBS：ナーブス）で定義します。
NURBS は、精度の高い数学的な形状表現で、カーブとサーフェスを直感的に編集する事が出来ます。
NURBS に関しては様々な文献があるので、詳細はそちらを参照してください。
NURBS の基礎的理解は、SDK のクラスと機能を効果的に扱う際の助けになります。
NURBS カーブを定義するものは、4 つ有り、それは、Degree（次数）、コントロールポイント、ノットと NURBS の公式です。

Degree（次数）

次数は、1 次、2 次、3 次、5 次のように全て整数で表され、Rhino では、1~11 次までを許可します。以下が次数とカーブの関係です。



	ラインやポリラインは、1 次の NURBS カーブで表現されます。 階数で表すと 2 階（階数=次数+1）で表されます。
	円や楕円は、2 次の NURBS カーブで表現されます。 階数で表すと 3 階で表されます。
	自由曲線は、3 次の NURBS カーブで表現されます。 階数で表すと 4 階で表されます。 この他に、5 次の自由曲線が一般的ですが、他の次数はあまり使用されません。

Control points（コントロールポイント）

NURBS のコントロールポイントは、最小でも（次数+1）の整数から成るリストです。
一番、一般的な NURBS カーブの編集方法は、コントロールポイントを移動する事です。
コントロールポイントには、weight（ウエイト）と呼ばれる数値が割り当てられています。
一部の例外を除き、ウエイトは、正の値を持ちます。全てのカーブのコントロールポイントの値が+1+を持つときに、カーブは非有理と呼ばれます。
ここでは、Grasshopper によって、ウエイトの値をインタラクティブに変更する方法を紹介します。

Knots or knot vector（ノット、またはノットベクトル）

それぞれの NURBS カーブは、ノットベクトルと呼ばれる数字のリストが関連付けられています。
ノットは理解し、設定するのは若干困難ですが、幸運なことに、SDK によってこれらの操作をお来ぬ事が出来ます。
ここでは、いくつか必要な知識を挙げておきます。

Knot multiplicity（多重ノット）

同一のノットの値が続いている部分をノットの多重と呼びます。どのノット値もカーブの次数以上、持つ事は出来ません。ここでいくつかノットに関して必要なことを挙げておきます。

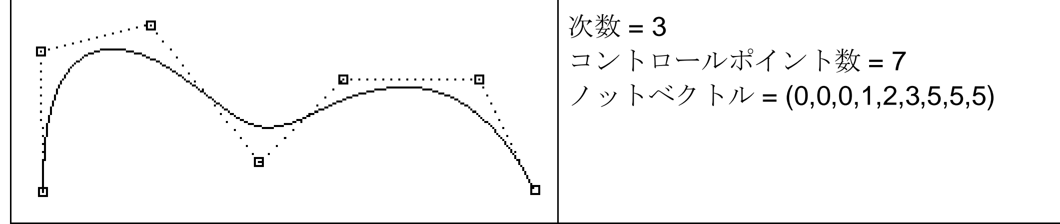
Full multiplicity knot（完全多重ノット）は、ノット多重が、次数と同じ値を持つときに言います。
クランプカーブは、2 つの端点で、完全な多重性を持ち、これによって端点の位置とコントロールポイントの位置が一致します。
もしも多重ノットカーブの途中に有る場合には、カーブの途中でも、コントロールポイントを必ず通過し、結果としてキンクを作ります。

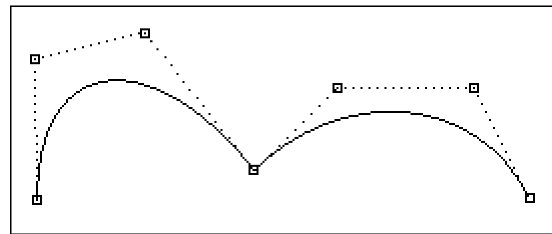
Simple knot（単純ノット）は、多重性が無く一度しかその数値は表れません。

Uniform knot vector（均一ノットベクトル）は次の 2 つの条件を満たします。

1. ノットの数=コントロールポイントの数＋次数の数-1
2. ノットは、多重ノットで始まり、単純ノットが続き、終点で多重ノットを持ち、ノットの値が、均等の間隔で増えていく。これが、典型的なクランプカーブです。周期カーブは後述するように異なる構造を持ちます。

下記が、同一のコントロールポイントの座標を持ちながら異なるノットベクトルを持つ例です。



	次数 = 3 コントロールポイント数 = 7 ノットベクトル = (0,0,0,1,1,1,4,4,4) *注: 中点の完全多重ノットがキンクを作成し、カーブがそのコントロールポイントを通過します。*

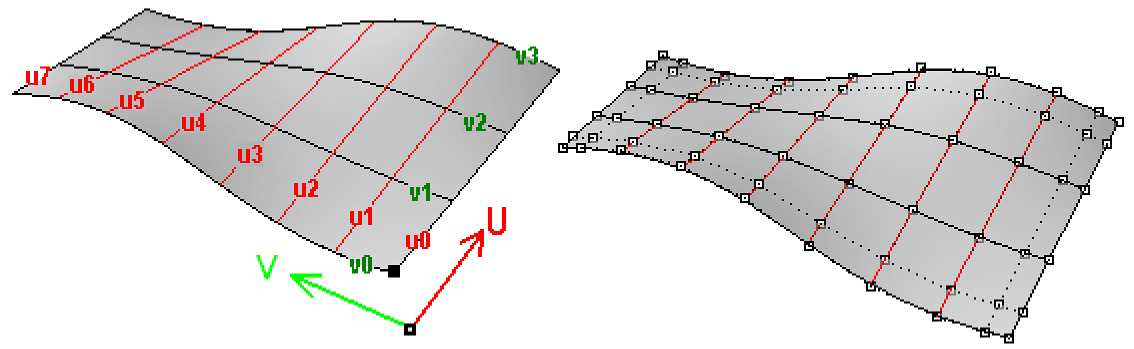
Evaluation rule（NURBS の評価規則・公式）

NURBS の公式は数値と割り当てられた点を使用した数学的な方程式で表されます。
公式は次数、コントロールポイント、ノットを含みます。

この公式を使用するために、SDK は、カーブのパラメーターを取得し、カーブ上の点を取得する機能があります。
パラメーターは、カーブ領域内の数値となります。
領域は 2 つの数値から定義され、その数値は通常、増加していきます。領域の最小パラメーター値(m_t(0))、カーブの始点で、最大のパラメーター値(m_t(1))が終点となります。

NURBS サーフェス

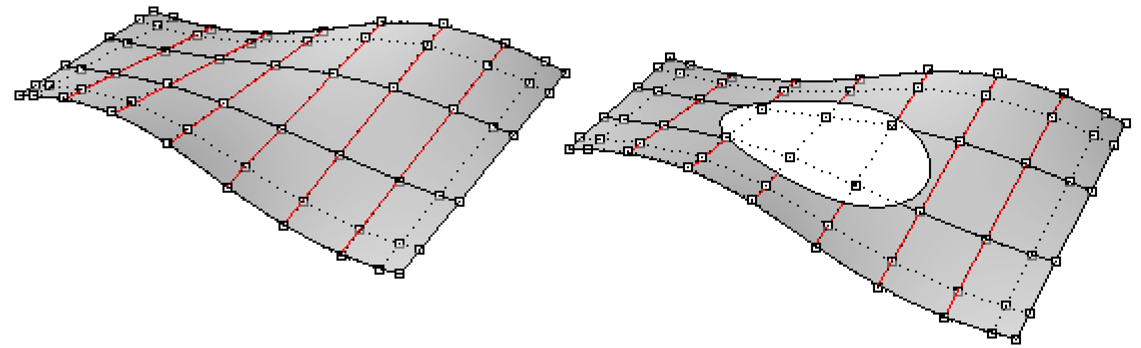
NURBS サーフェスは、NURBS カーブのグリッドが、2 つの方向に向かった軌跡と考える事ができます。NURBS サーフェスの形状は、2 つの方向（U-方向と V-方向）に配置されたコントロールポイントの数と次数によって定義されます。詳細は、Rhino のヘルプの用語集を参照ください。



NURBS サーフェスはトリムし、トリム解除をする事ができます。トリムサーフェスは NURBS サーフェスの母面とサーフェス上にある特定の形状をトリムする閉じたカーブから構成されます。

それぞれのサーフェスは、サーフェスの境界（outer loop）を定義する閉じたカーブと、穴を定義する交差していない閉じた内側のカーブ（Inner loop）を持ちます。

Outer loop を持つサーフェスは、NURBS サーフェスの母面と同じで、穴を持たないので非トリムサーフェスとなります。

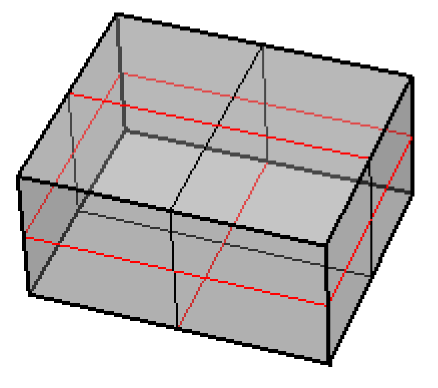


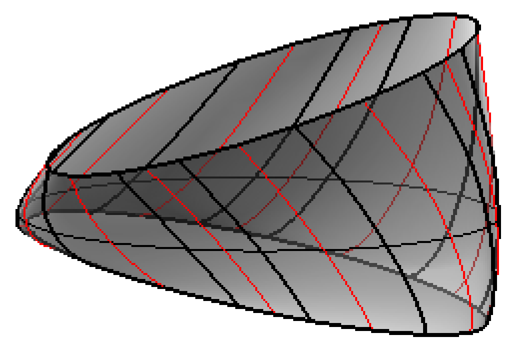
上図で、左が、非トリムサーフェス、右側が楕円でトリムされたサーフェスです。
注：トリムしてもサーフェスの NURBS 定義は変わりません。（同じ、コントロールポイント、次数、ノットベクトルを持つ）

Polysurfaces（ポリサーフェス）

ポリサーフェスは複数のサーフェス（多くの場合、トリムサーフェス）がお互いに結合された状態になっているものです。
それぞれのサーフェスは、固有のパラメーターと UV 方向を持ちます。
ポリサーフェスとトリムサーフェスは、境界表現（Brep：ビーレップ）と呼ばれるもので定義されます。
境界表現は基本的にサーフェス、エッジ、頂点、等の幾何要素をトリムデータや異なるパート間との関係とともに記述します。
例えば、それぞれのフェース、それを囲むエッジやトリム、サーフェスに関連した法線方向、隣り合うフェースとの関係等です。Brep についての詳細は後述します。

OnBrep が OpneNURBS において一番複雑なデータ構造を持ち、簡単には消化されませんが、RhinoSDK には、多くのツールと機能が用意されており Brep を作成し、操作することが出来ます。





15.3 OpenNURBS オブジェクトの階層構造

SDK ヘルプファイルは、全てのクラスの階層構造について公開しています。
以下に、ジオメトリーを作成し操作するための Script をコーディングする際に使用されるであろうクラスのサブセットを記します。
このリストは完全ではありませんので、詳細はヘルプファイルを参照ください。

OnObject (全ての Rhino クラスは、OnObject から派生します。)

- **OnGeometry** (*OnObject から派生又は、継承されるクラス*)
  - OnPoint
    - OnBrepVertex
    - OnAnnotationTxtDot
  - OnPointGrid
  - OnPointCloud
  - OnCurve (*abstaract class)*
    - OnLineCurve
    - OnPolylineCurve
    - OnArcCurve
    - OnNurbsCurve
    - OnCurveOnSurface
    - OnCurveProxy
      - OnBrepTrim
      - OnBrepEdge
  - OnSurface (*abstract classt*)
    - OnPlaneSurface
    - OnRevSurface
    - OnSumSurface
    - OnNurbsSurface
    - OnProxySurface
      - OnBrepFace
      - OnOffsetSurface
  - OnBrep
  - OnMesh
  - OnAnnotation

- **Points and Vectors** *(OnGeometry から派生しないクラス)*
  - On2dPoint (good for parameter space points)
  - On3dPoint
  - On4dPoint (コントロールポイントの x,y,z,座標とウエイト値、w を持ちます。)
  - On3dVector

- **Curves** *(OnGeometry から派生しないクラス)*
  - OnLine
  - OnPolyline (OnPointArray から派生するクラス)
  - OnCircle
  - OnArc
  - OnEllipse
  - OnBezierCurve

- **Surfaces** *(OnGeometry から派生しないクラス)*
  - OnPlane
  - OnSphere
  - OnCylinder

- OnCone
- OnBox
- OnBezierSurface

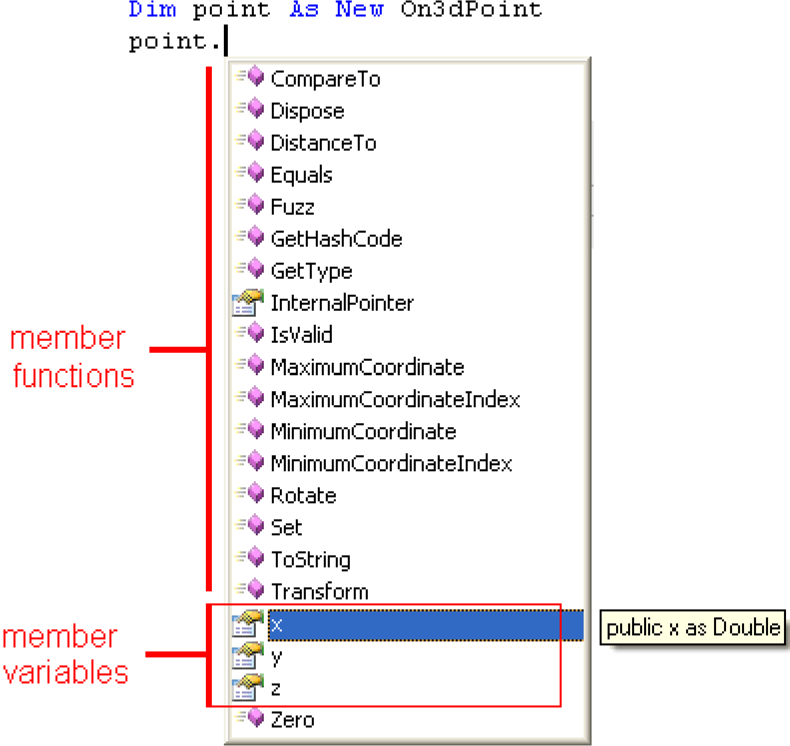
- その他
  - OnBoundingBox (バウンディングボックスを計算するためのクラス)
  - OnInterval (カーブとサーフェスの領域のために使用するクラス)
  - OnXform (移動、回転、スケール等幾何オブジェクトの変形を行うクラス)
  - OnMassProperties (体積、重心等マスプロパティーを計算)

15.4 クラスの構造

一般的なクラス（ユーザーが定義したデータ構造）は 4 つのパートから成ります。

- **Constructor**（コンストラクター）：クラスのインスタンスを生成する。.
- **Public member variables**（パブリックメンバー、変数）：クラスのデータが格納されるところ。OpenNURBS のメンバーは、一般的に‟m_”で始まり分離される。
- **Public member functions**（パブリックメンバー、ファンクション）：This パブリックメンバーの変数を生成、更新、操作する全てのクラスのファンクションを持ちます。また特定の機能を実行します。
- **Private members**（プライベートメンバー）: these are class utility functions and variables for internal use.

Script コンポーネント内で、クラスを実体化させるとき、オートコンプリート機能により、全てのメンバーと変数の一覧を見ることが出来ます。一覧中のファンクションや変数の上にマウスカーソルを持っていくと、各ファンクションの概要が表示されます。
以下が、On3dPoint クラスの変数を宣言する例です。
ここでは、‟point.”までタイプしたときに、オートコンプリート機能により、On3dPoint クラスで使用できるメンバーファンクション、変数が表示されているのが分かります。



既存のクラスのデータをコピーし、新しいものを作るには、そのクラスにより、1 つもしくはそれ以上の方法があります。
例えば、新しい On3dPoint を作成し、
Copying data from an exiting class to a new one can be done in one or more ways depending on the class. For example, let‟s create a new On3dPoint and copy the content of an existing point into it. This is how we may do it:

```
'Point クラスのインスタンスを実体化するときにコンストラクターを使用
Dim new_pt as New On3dPoint( input_pt )

'"= 演算子"を使用する Use the "= operator" if the class provides one
```

```
Dim new_pt as New On3dPoint
new_pt = input_pt

'"New" ファンクションを使用する方法 You can use the "New" function if available
Dim new_pt as New On3dPoint
new_pt.New( input_pt )

'"Set" ファンクションを使用する方法 There is also a "Set" function sometimes
Dim new_pt as New On3dPoint
new_pt.Set( input_pt )

'メンバー変数を一つずつコピーする方法 Copy member variables one by one. A bit exhaustive method
Dim new_pt as New On3dPoint
new_pt.x = input_pt.x
new_pt.y = input_pt.y
new_pt.z = input_pt.z

'OpenNURBS のジオメトリークラスの"Duplicate" ファンクションを使用する方法 Dim new_crv as New OnNurbsCurve
new_crv = input_crv.DuplicateCurve()
```

15.5 定数インスタンスと定数ではないインスタンス

Rhino .NET SDK は 2 つのクラスのセットを供給します。
一つ目は、+IOn3dPoint+のように+I+で始まるクラスでこれらは定数です。定数以外のクラスは+I+が先頭に付かないこと以外は、+On3dPoint+のように同じ名前になります。
定数のクラスはコピーしたり、そのメンバー変数と幾つかのファンクションを参照することが出来ますが変数を変えることは出来ません。

Rhino .NET SDK は、Rhino C++SDK がベースになっています。C++言語は、クラスの定数のインスタンスを渡す機能を許可し、完全な SDK の機能となります。一方、DotNET はこのような概念を持たないので、2 つのバージョンがそれぞれのクラスに存在します。

15.6 点とベクトル

点群とベクトルを格納し、操作するクラスは多くあります。ここで、倍精度の点を考えてみると 3 種類のタイプの点のクラスがあります。

クラス名	メンバー変数	注
On2dPoint	**x** as Double **y** as Double	主に、パラメーターの空間点に使用される。 クラス名中の+d+は、倍精度浮動小数点型であることを意味します。+f+がつくクラスは、単精度型です。
On3dPoint	**x** as Double **y** as Double **z** as Double	3 次元座標で最も一般的に使用される点のクラスです。
On4dPoint	**x** as Double **y** as Double **z** as Double **w** as Double	grip points に使用されます。 grip は、3 次元座標に加えて weight（ウエイト）の値を持ちます。

Points and vectors operations include:

Vector Addition:

```
Dim add_v As New On3dVector = v0 + v1
```

Vector Subtraction:

```
Dim subtract_vector As New On3dVector = v0 . v1
```

Vector between two points:

Dim dir_vector As New On3dVector = p1 . p0

Vector dot product (if result is positive number then vectors are in the same direction):

Dim dot_product As Double = v0 * v1

Vector cross product (result is a vector normal to the 2 input vectors)

Dim normal_v As New On3dVector = OnUtil.ON_CrossProduct(v0, v1)

Scale a vector:

Dim scaled_v As New On3dVector = factor * v0

Move a point by a vector:

Dim moved_point As New On3dPoint = org_point + dir_vector

Distance between 2 points:

Dim distance As Double = pt0.DistanceTo(pt1)

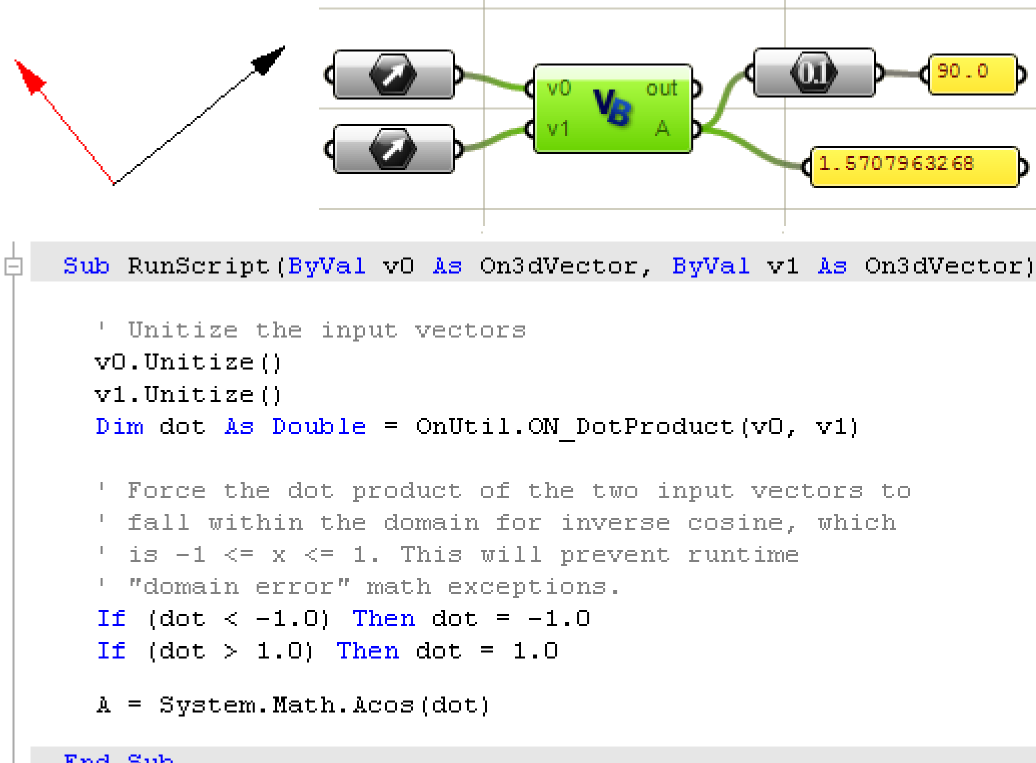
Get unit vector (set vector length to 1):

v0.Unitize()

Get vector length:

Dim length As Double = v0.Length()

Following example show how to calculate the angle between two vectors.



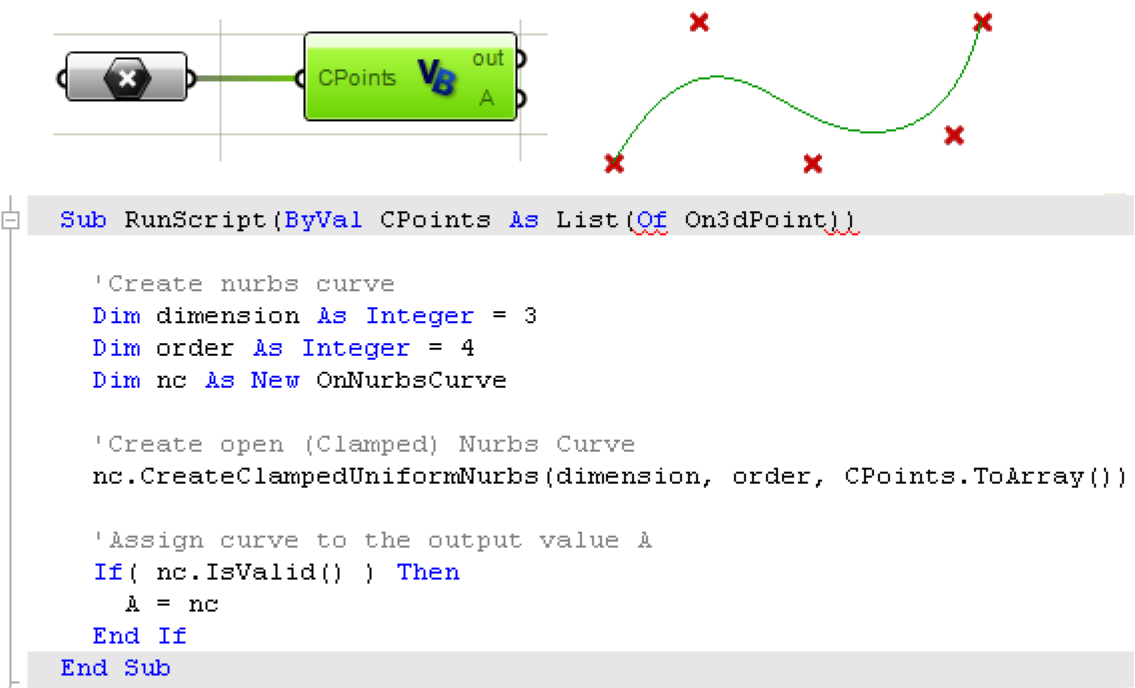

```
Sub RunScript(ByVal v0 As On3dVector, ByVal v1 As On3dVector)

    ' Unitize the input vectors
    v0.Unitize()
    v1.Unitize()
    Dim dot As Double = OnUtil.ON_DotProduct(v0, v1)

    ' Force the dot product of the two input vectors to
    ' fall within the domain for inverse cosine, which
    ' is -1 <= x <= 1. This will prevent runtime
    ' "domain error" math exceptions.
    If (dot < -1.0) Then dot = -1.0
    If (dot > 1.0) Then dot = 1.0

    A = System.Math.Acos(dot)

End Sub
```

15.7 OnNurbsCurve

NURBS カーブを生成するためには、下記の手順が必要になります。

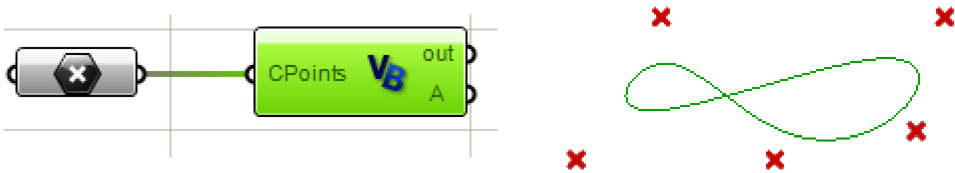
- 次数, 一般的 = 3.
- 階数: 次数 + 1.
- コントロールポイント (点群の配列).
- ノットベクトル (数値の配列).
- カーブのタイプ (clamped：（クランプ、非周期で開いたカーブ）か periodic：周期).

次に見られる通り、ノットベクトルを生成する機能があります。まず、次数とコントロールポイントのリストを持つ必要があります。次の例がクランプカーブを作成する例です。




```
Sub RunScript(ByVal CPoints As List(Of On3dPoint))

  'Create nurbs curve
  Dim dimension As Integer = 3
  Dim order As Integer = 4
  Dim nc As New OnNurbsCurve

  'Create open (Clamped) Nurbs Curve
  nc.CreateClampedUniformNurbs(dimension, order, CPoints.ToArray())

  'Assign curve to the output value A
  If( nc.IsValid() ) Then
    A = nc
  End If
End Sub
```

スムースな閉じたカーブは、クランプカーブと同じ入力となりますが、周期カーブとして作成する必要があります。（CreatePeriodicUniform クラスを使用）次の例が周期カーブを作成する例です。

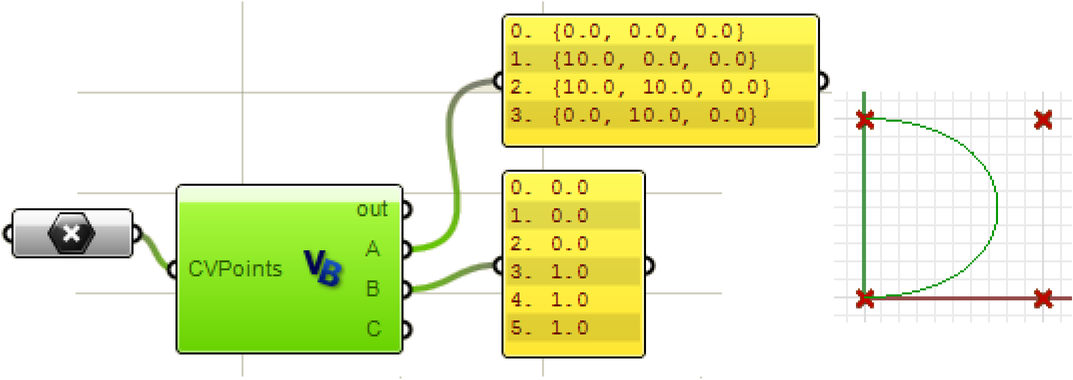


```
Sub RunScript(ByVal CPoints As List(Of On3dPoint))

  'Create nurbs curve
  Dim dimension As Integer = 3
  Dim order As Integer = 4
  Dim nc As New OnNurbsCurve

  'Create closed (Periodic) Nurbs Curve
  nc.CreatePeriodicUniformNurbs(dimension, order, CPoints.ToArray())

  'Assign curve to the output value A
  If( nc.IsValid() ) Then
    A = nc
  End If
End Sub
```

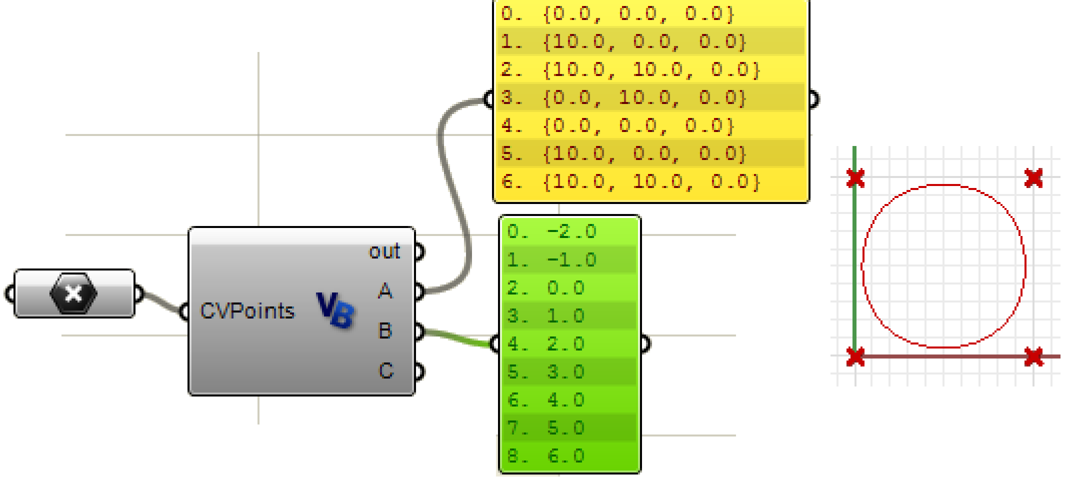
Clamped vs periodic NURBS curves　クランプカーブと周期カーブ

クランプカーブは、通常、開いたカーブで、端点の位置がコントロールポイントの位置と一致します。周期カーブは、滑らかな閉じたカーブです。
この違いを理解するのには、お互いのコントロールポイントを比較してみると良いでしょう。

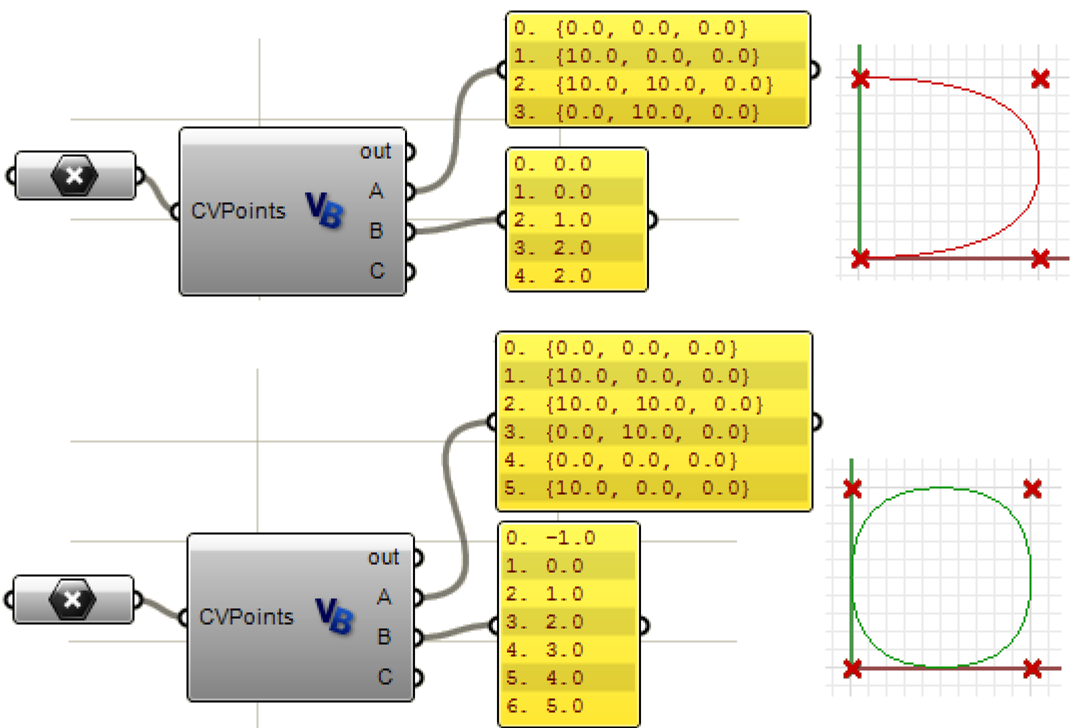
次の VBScript コンポーネントは、クランプカーブを作成し、出力するものです。



次が、全く同じコントロールポイントの座標と次数を持つ周期カーブの例です。



周期カーブは、クランプカーブが、4 つのコントロールポイントのみ使用するのに対して、7 つのコントロールポイントに変えます（7=4＋次数）。
周期カーブのノットベクトルは、単純なノットになりますが、クランプカーブは、始点・終点で、完全な多重ノットを持ちます。
以下が次数を 2 にしたサンプルです。次数を変えると、周期カーブのコントロールポイントの数とノットの数が、変わるのが分かります。

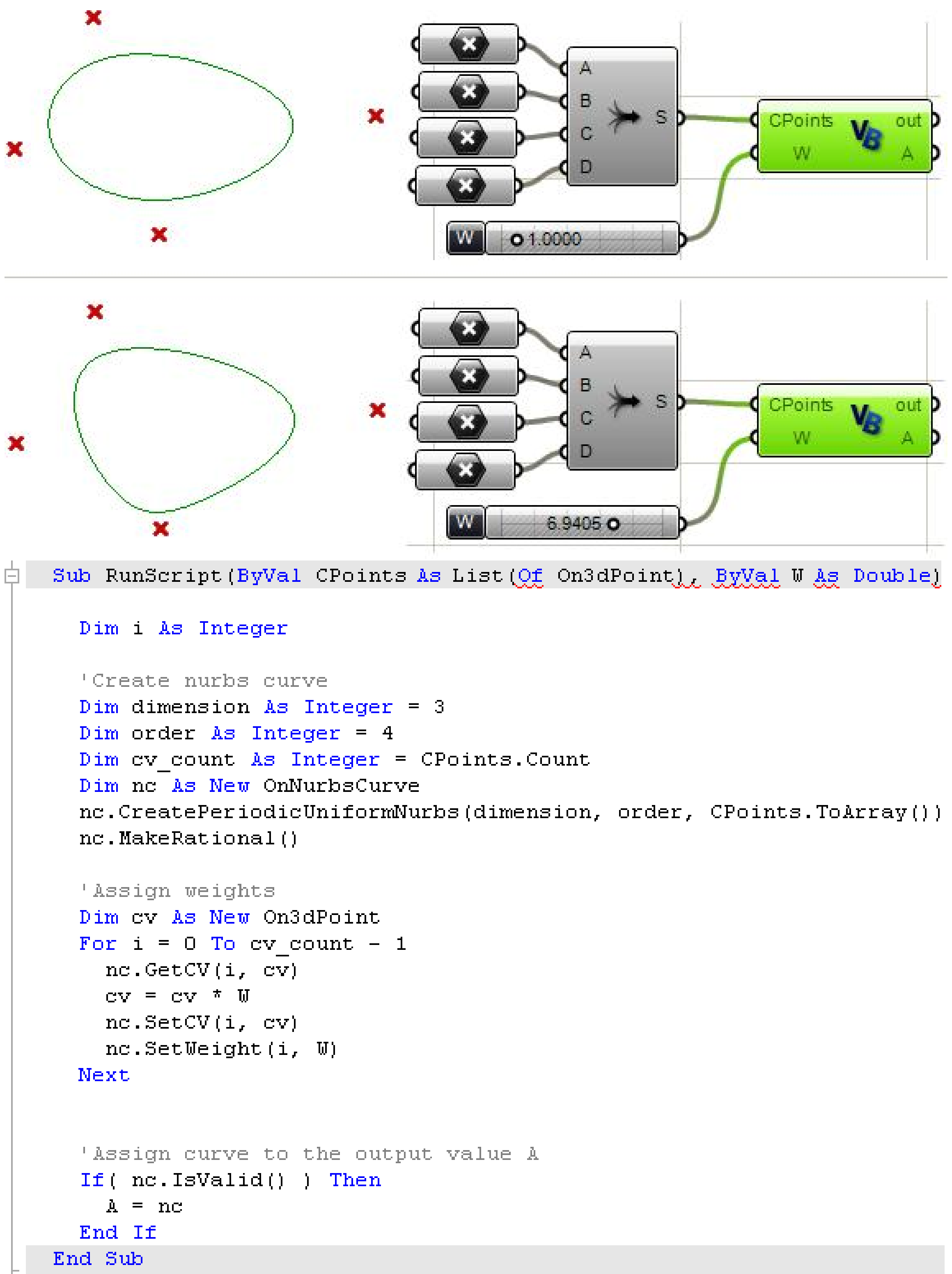


This is the code used to navigate through CV points and knots in the previous examples:

```
'Output control points
Dim count As Double = nc.CVCount()
Dim i As Integer
Dim cvs As New List( Of On3dPoint )
For i = 0 To count - 1
  Dim cv As New On3dPoint(0, 0, 0)
  nc.GetCV(i, cv)
  cvs.Add(cv)
Next

'Output knots
Dim knots As New List( Of Double )
count = nc.KnotCount()
For i = 0 To count - 1
  knots.Add(nc.Knot(i))
Next
```

Weights　ウエイト

コントロールポイントにおけるウエイトの値は、均一な NURBS カーブの場合は、+1+です。
この値は、有理 NURBS の場合は、値が変わります。
次の例は、Grasshopper において、インタラクティブに、コントロールポイントにおけるウエイトの値を変更するものです。



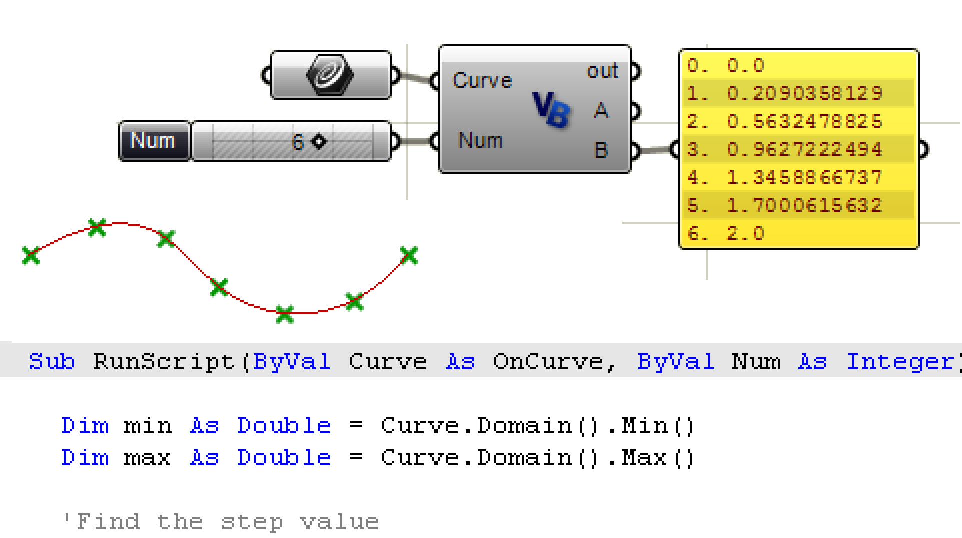

```
Sub RunScript(ByVal CPoints As List(Of On3dPoint), ByVal W As Double)

  Dim i As Integer

  'Create nurbs curve
  Dim dimension As Integer = 3
  Dim order As Integer = 4
  Dim cv_count As Integer = CPoints.Count
  Dim nc As New OnNurbsCurve
  nc.CreatePeriodicUniformNurbs(dimension, order, CPoints.ToArray())
  nc.MakeRational()

  'Assign weights
  Dim cv As New On3dPoint
  For i = 0 To cv_count - 1
    nc.GetCV(i, cv)
    cv = cv * W
    nc.SetCV(i, cv)
    nc.SetWeight(i, W)
  Next


  'Assign curve to the output value A
  If( nc.IsValid() ) Then
    A = nc
  End If
End Sub
```

Divide NURBS curve　NURBS カーブの分割

カーブを指定した数のセグメントに分割するには次のステップになります。

- カーブの空間領域を見つけます。
- 均等なセグメントになるように分割するカーブのパラメーターのリストを作成します。
- 3D カーブの点を見つけます。

これを行うには次の例のようなコーディングになります。
注：後述する RhUtil（Rhino ユーティリティー）で、カーブを指定したセグメントの数や、指定長で分割するグローバル関数あります。



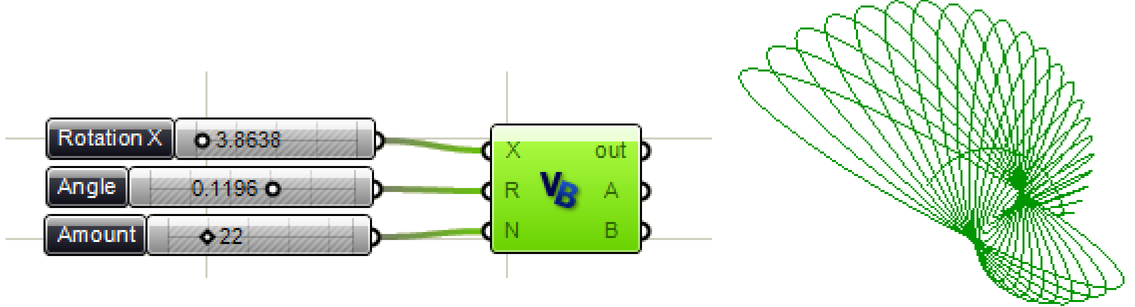

```
Sub RunScript(ByVal Curve As OnCurve, ByVal Num As Integer)

  Dim min As Double = Curve.Domain().Min()
  Dim max As Double = Curve.Domain().Max()

  'Find the step value
  Dim step_value As Double = (max - min) / (Num - 1)

  Dim Points As New List( Of on3dPoint )

  Dim t_list(Num) As Double
  For i As Integer = 0 To Num
    t_list(i) = i / Num
  Next

  If (Curve.GetNormalizedArcLengthPoints(t_list, t_list)) Then
    For i As Integer = 0 To Num
      Dim pt As On3dPoint = Curve.PointAt(t_list(i))
      Points.Add(pt)
    Next
  End If


  A = Points
  B = t_list

End Sub
```

15.8 OnCurve から派生しない Curve クラス

全てのカーブは、NURBS カーブとして表現出来ますが、他のカーブタイプのジオメトリーを使用することも便利です。その一つの理由は、それらの数学表現は、NURBS よりも理解しやすく、データとしても軽いからです。
相対的に必要に応じて OnCurve から派生しないこれらのカーブを使用するほうが NURBS 形状を取得しやすいです。基本的に、対応するクラスに変換する必要があります。
以下のテーブルがその対応表です。

Curves Types	OnCurve Derived Types
OnLine	OnLineCurve
OnPolyline	OnPolylineCurve
OnCircle	OnArcCurve or OnNurbsCurve (use GetNurbsForm() member function)
OnArc	OnArcCurve or OnNurbsCurve (use GetNurbsForm() member function)
OnEllipse	OnNurbsCurve (use GetNurbsForm() member function)
OnBezierCurve	OnNurbsCurve (use GetNurbsForm() member function)

以下は、+OnEllipse+と+OnPolyline+クラスを使用した例です。:



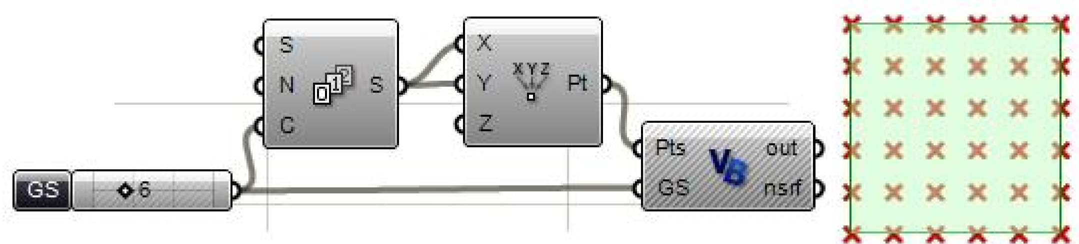

```
Sub RunScript(ByVal X As Object, ByVal R As Object, ByVal N As Object)
  'Declare a new list of OpenNURBS circles
  Dim c_list As New List(Of OnEllipse)

  'Declare list of lines
  Dim p_list As New On3dPointArray

  For i As Int32 = 1 To N
    'Declare a new circle
    Dim c As New OnEllipse(OnUtil.On_xy_plane, i / 2, i)
    'Rotate the circle
    C.Rotate(R * i, New On3dVector(0, 1, 0), New On3dPoint(X, 0, 0))
    'Add the circle to the list
    c_list.Add(c)
    'Add center point
    p_list.Append(C.Center())
  Next

  Dim polyline As New OnPolyline(p_list)
  'Assign the list to the output value A
  A = c_list
  'Assign polyline to output value B
  B = polyline
End Sub
```

15.9 OnNurbsSurface

OnNurbsCurve クラスに関して論じたように、OnNurbsSurface を生成するには、次のことを知っておく必要があります。

- 次数、通常 = 3
- U,V 方向の階数、次数 + 1
- コントロールポイント
- U,V 方向のノットベクトル
- サーフェスのタイプ(クランプ、又は周期サーフェス).

次の例は、コントロールポイントの格子点から NURBS サーフェスを生成する例です。
(注：PT(Point XYZ コンポーネントの List は、+Cross Reference+を指定します。)



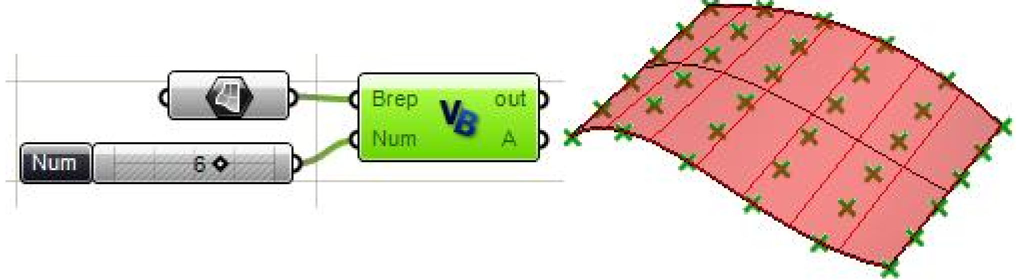

```
Sub RunScript(ByVal Pts As List(Of On3dPoint), ByVal GS As Integer)
    'Create a grid of points
    Dim Grid As New ArrayList()
    'Call grid function
    Call CreateGrid(Pts, Grid, GS)
    'Call create nurbs surface function
    Dim ns As OnNurbsSurface
    ns = CreateNS(Grid, GS)

    'Assign mid point to output
    nsrf = ns

End Sub
```

```
Sub CreateGrid( ByVal Pts As List(Of On3dPoint),
                ByRef Grid As ArrayList, ByVal GS As Integer )
  Dim i As Integer
  Dim j As Integer
  For i = 0 To Pts.Count() - 1 Step GS
    'Declare a row of points
    Dim Row As New List( Of On3dPoint )
    For j = i To i + GS - 1
      'Get a reference od the point
      Dim pt As On3dPoint
      pt = Pts(j)
      'Add point to the row
      Row.Add(pt)
    Next
    'Add row to the grid
    Grid.Add(Row)
  Next
End Sub
```

```
Function CreateNS(ByVal cvpoints As ArrayList,
                  ByVal GS As Integer) As OnNurbsSurface
  Const Degree As Integer = 3

  'Make the surface
  Dim orderU As Integer = Degree + 1
  Dim orderV As Integer = Degree + 1

  Dim ns As New OnNurbsSurface
  ns.Create(3, False, orderU, orderV, GS, GS)

  'Add cv points
  Dim i As Integer
  Dim j As Integer
  Dim pt As On3dPoint
  For i = 0 To GS - 1
    For j = 0 To GS - 1
      pt = cvpoints(i)(j)
      ns.SetCV(i, j, pt)
    Next
  Next

  'Set knots for open surface
  ns.MakeClampedUniformKnotVector(0)
  ns.MakeClampedUniformKnotVector(1)

  CreateNS = ns
End Function
```

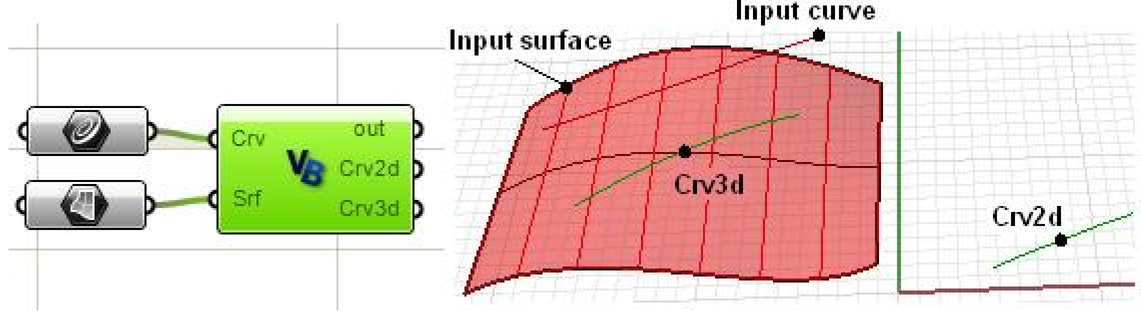
もう一つの一般的な例は、サーフェス領域で分割する方法です。
以下の例は、サーフェスを均等に U,V 方向に分割して、分割点に点データを配置する例です。

- まず、サーフェスの UV の領域を正規化します。（0～1 にセット）
- 分割するポイントのステップを決めます。Calculate step value using number of points.
- サーフェス上に生成するポイントの UV パラメーターをネストするループで計算します。



```
Sub RunScript(ByVal Brep As OnBrep, ByVal Num As Integer)
  'Find step - Num must be > 1
  Dim StepValue As Double = 1 / (Num - 1)

  Dim nSrf As New OnNurbsSurface
  nSrf = Brep.Face(0).NurbsSurface

  'Normalize domain in u and v directions
  nSrf.SetDomain(0, 0, 1)
  nSrf.SetDomain(1, 0, 1)

  Dim Points As New List( Of on3dPoint )
  Dim i As Double = 0
  Dim j As Double = 0
  For i = 0 To 1 Step StepValue
    For j = 0 To 1 Step StepValue
      Dim Pt As New On3dPoint
      Pt = nSrf.PointAt(i, j)
      Points.Add(Pt)
    Next
  Next

  A = Points
End Sub
```

OnSurface クラスは、多くの機能を持ち、サーフェスを操作するにあたり便利です。
次の例は、どのようにサーフェスをプル投影する例です。
ここでは、Grasshopper の VBScript コンポーネントに 2 つの出力を設けています。
最初の出力（Crv2d）は 2 次元のパラメーター空間カーブ（3D カーブを XY 平面に展開したもの）で、3D カーブがサーフェス上にプルされた領域を参照します。2 番目の出力（Crv3d）は、プル投影された 3 次元カーブです。この 3D カーブは、2D のパラメーター空間カーブを、サーフェスに+pushing（プッシュする）+ことにより得られます。



```
Sub RunScript(ByVal Crv As OnCurve, ByVal Srf As OnBrep)

  'Get pulled curve in 2D parameter space
  Dim pull_crv As OnCurve
  pull_crv = Srf.m_S(0).Pullback(Crv, doc.AbsoluteTolerance())

  'Get the pulled curve in 3D space
  Dim push_crv As OnCurve
  push_crv = Srf.m_S(0).Pushup(pull_crv, doc.AbsoluteTolerance())

  'Output both curves
  Crv2d = pull_crv
  Crv3d = push_crv

End Sub
```

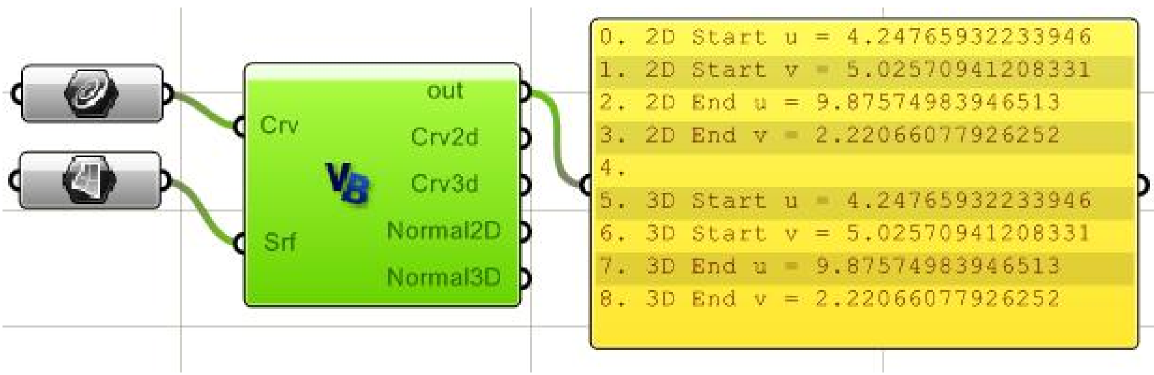
ここでは、さらにプル投影されたカーブ始点と終点の法線ベクトルを計算してみます。
方法は 2 つあります。

- プルされた 2D カーブの始点・終点を使用する。これらの点がサーフェス上の始点・終点が持つパラメーター空間となります。
- プッシュされた、3D カーブの終点を使用し、サーフェス上の最も近い点を見つけ、パラメーター値から、サーフェスの法線を得る。

```
Sub GetEndNormals2D(ByVal crv2d As OnCurve,
                    ByVal srf As OnSurface,
                    ByRef EndVectors2D As List(Of On3dVector ))
  Dim start_normal As On3dVector
  Dim end_normal As On3dVector

  'find start and end points in parameter space
  Dim start2d As New On2dPoint
  start2d = crv2d.PointAtStart()
  Dim end2d As New On2dPoint
  end2d = crv2d.PointAtEnd()

  'Output parameters
  'Surface parameters are the x and y of the 2d curve end points
  Print("2D Start u = " & start2d.x)
  Print("2D Start v = " & start2d.y)
  Print("2D End u = " & end2d.x)
  Print("2D End v = " & end2d.y)
  Print("")

  'Call surface normal function
  start_normal = srf.NormalAt(start2d.x, start2d.y)
  end_normal = srf.NormalAt(end2d.x, end2d.y)

  EndVectors2D.Add(start_normal)
  EndVectors2D.Add(end_normal)
End Sub
```

```
Sub GetEndNormals3D(ByVal crv3d As OnCurve,
                    ByVal srf As OnSurface,
                    ByRef EndVectors3D As List(Of On3dVector ))

  'Declare start and end normal
  Dim start_normal As On3dVector
  Dim end_normal As On3dVector

  'Find start and end points in parameter space
  Dim start3d As New On3dPoint
  start3d = crv3d.PointAtStart()
  Dim end3d As New On3dPoint
  end3d = crv3d.PointAtEnd()

  'Declare parameters
  Dim u As Double
  Dim v As Double

  'Get surface closest point
  srf.GetClosestPoint(start3d, u, v)
  start_normal = srf.NormalAt(u, v)

  'Output start parameters
  Print("3D Start u = " & u)
  Print("3D Start v = " & v)

  srf.GetClosestPoint(end3d, u, v)
  end_normal = srf.NormalAt(u, v)

  'Output end parameters
  Print("3D End u = " & u)
  Print("3D End v = " & v)

  EndVectors3D.Add(start_normal)
  EndVectors3D.Add(end_normal)
End Sub
```

This is the component picture showing output of parameter value at the end points using both functions. Notice that both methods yield same parameters as expected.

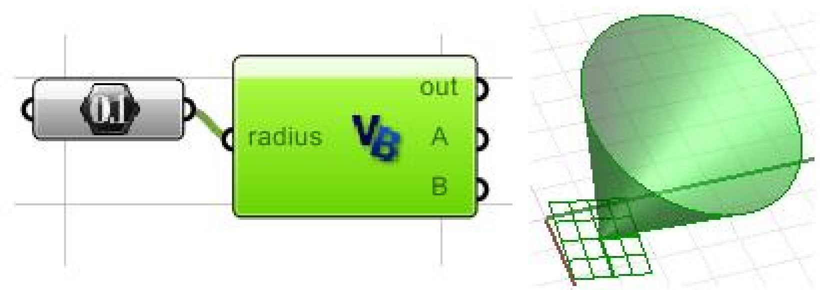


15.10 OnSurface から派生しない Surface クラス

OpenNURBS は、OnSurface クラスから派生しないサーフェスクラスを供給します。
これらは、数学的に有効なサーフェス定義で、OnSurface から派生するタイプに変換することが出来ます。
以下に、サーフェスクラスのリストを記します。

基本サーフェスタイプ	**OnSurface** から派生するタイプ
OnPlane	OnPlaneSurface 又は OnNurbsSurface (OnPlane.GetNurbsForm() function　を使用)
OnShpere	OnRevSurface 又は OnNurbsSurface (OnShpere.GetNurbsForm() function　を使用)
OnCylinder	OnRevSurface 又は OnNurbsSurface (OnCylinder.GetNurbsForm() function　を使用)
OnCone	OnRevSurface 又は OnNurbsSurface (OnCone.GetNurbsForm() function　を使用)
OnBezierSurface	OnNurbsSurface (GetNurbsForm() member function　を使用)

以下は、OnPlane と OnCone クラスを使用した例です。



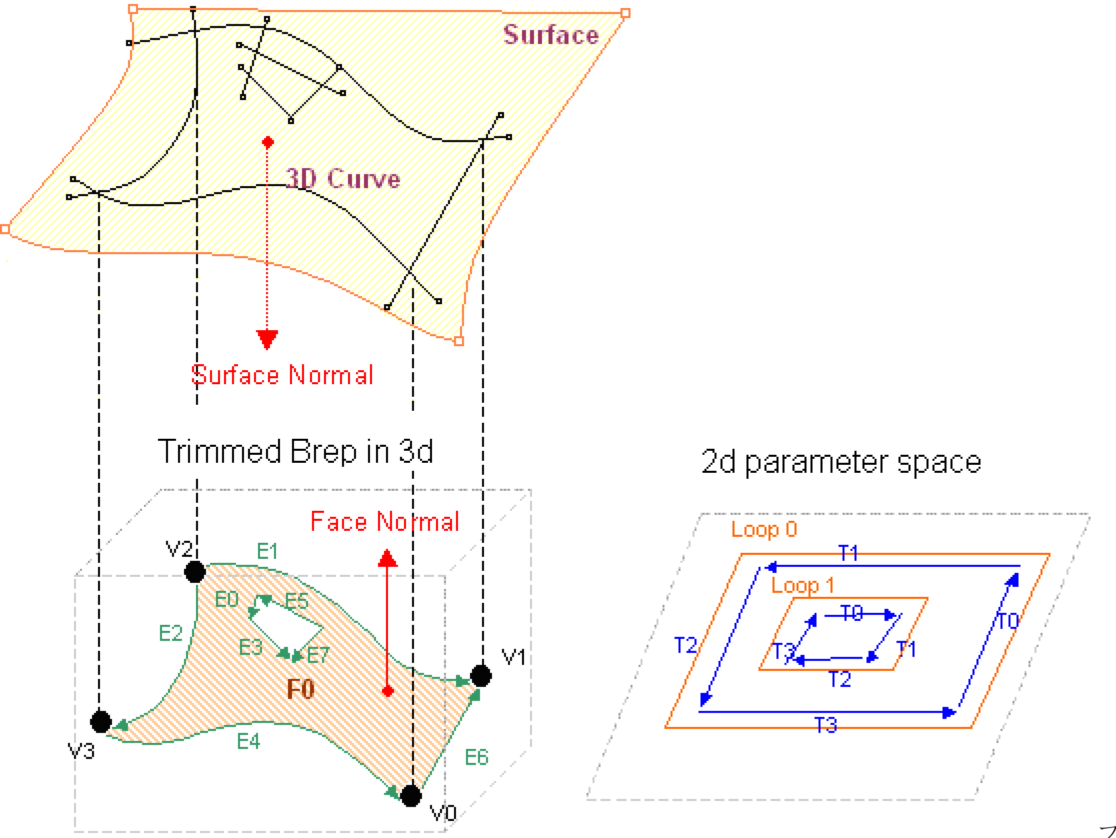

```
Sub RunScript(ByVal radius As Double)
  'Create a plane from origin and normal
  Dim plane As New OnPlane
  Dim origin As New On3dPoint(1, 1, 0)
  Dim normal As New On3dVector(1, 1, 3)
  plane.CreateFromNormal(origin, normal)

  'Define height value
  Dim height As Double = 5
  'Create cone
  Dim cone As New OnCone(plane, height, radius)

  'Assign output parameter
  A = cone
  B = plane
End Sub
```

15.11 OnBrep クラス

B-Rep（境界値表現）は、その境界のサーフェスによって明確にオブジェクトを定義します。
OnBrep は、次の 3 つの構成要素で考えることができます。

- ジオメトリー（幾何形状）:3D の NURBS カーブとサーフェスです。またパラメトリック空間の 2D カーブとトリムカーブも含みます。
- 3D トポロジー: フェイス、エッジ、頂点です。それぞれのフェイスは一つの NURBS サーフェスを参照します。フェイスは、フェイスによるループを理解します。エッジは、3D カーブを参照します。それぞれのエッジは、トリムのリストを持っています。頂点は、空間上の 3D ポイントを参照し、それぞれの頂点データは、端点情報を持つエッジのリストを持ちます。
- 2D トポロジー:フェイスとエッジの 2D のパラメーター空間を表します。パラメーター空間内では、2D のトリムカーブは、時計回り、あるいは反時計回りとなりますが、これはどのフェイスの外側あるいは、内側のループかに依存します。一つの有効なフェイスは一つだけ外側のループを持ちますが、内側のループ（ホール）は必要な数だけ持つ事が出来ます。それぞれのトリムは、一つのエッジ、2 つの端点、2D カーブの 1 つのループを参照します。

次の図は、3 つの要素がどのようにそれぞれ関係するかを示しています。最上階の構成要素は基本となる 3DNURBS サーフェスと、一つのホールを持つフェイスの境界を定義するカーブジオメトリーを表します。
2 番目の構成要素は、フェイスの境界、外側のエッジ、内側のエッジ（ホールのバウンディングボックス）と頂点を含む、3D のトポロジー（位相関係）です。そして、最後に、トリムとループを含む 2D のパラメーター空間となります。

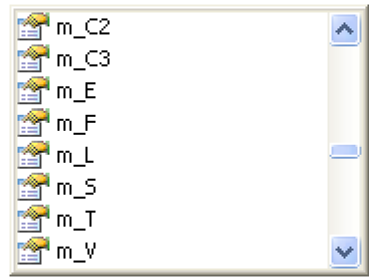


フェイス

OnBrep member variables　OnBrep のメンバー変数

OnBrep クラスのメンバー変数は全ての 2D 及び 3D ジオメトリーと位相情報を含みます。
一度、Brep クラスのインスタンスを作成すると、全てのメンバーファンクションと変数を見ることが出来ます。次のイメージは、+brep.+と Script コンポーネント中で、コーディングするとオートコンプリートによって表示されるメンバーファンクションです。

```
Dim brep As New OnBrep
brep.
```

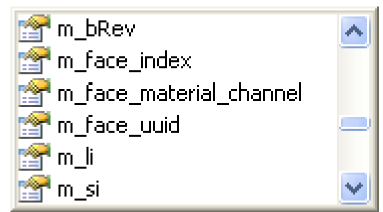


このテーブルは、データタイプを記述したリストです。

トポロジーメンバー: 異なる **brep** パーツ間の関係を記述	
OnBrepVertexArray **m_V**	brep の頂点の配列 (OnBrepVertex)
OnBrepEdgeArray **m_E**	brep のエッジの配列(OnBrepEdge)
OnBrepTrimArray **m_T**	brep のトリムの配列(OnBrepTrim)
OnBrepFaceArray **m_F**	brep のフェイスの配列(OnBrepFace)
OnBrepLoopArray **m_L**	ループの配列 (OnBrepLoop)
ジオメトリーメンバー: ３**D** カーブ、サーフェスと **2D** トリムのジオメトリーデータ	
OnCurveArray **m_C2**	トリムカーブの配列 (2D curves)
CnCurveArray **m_C3**	エッジカーブの配列(3D curves)
ONSurfaceArray **m_S**	サーフェスの配列

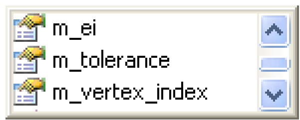
注；それぞれの OnBrep メンバーファンクションは、基本的に他のクラスの配列です。
例えば、+m_F+は、OnBrepFace クラスを参照する配列です。OnBrepFace クラスは、OnSurfaceProxy クラスから派生したもので、変数とそれ自身のメンバーファンクションを持ちます。 下記は OnBrepFace, OnBrepEdge, OnBrepVertex, OnBrepTrim と OnBrepLoop クラスのメンバー変数の例です。

```
Dim brep_face As New OnBrepFace
brep_face.
```

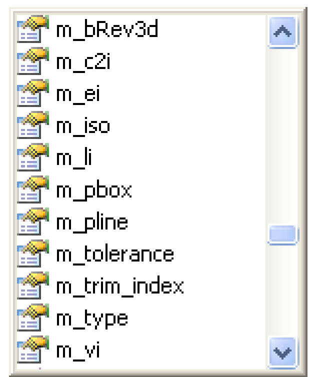


```
Dim brep_edge As New OnBrepEdge
brep_edge.
```

```
Dim brep_vertex As New OnBrepVertex
brep_vertex.
```

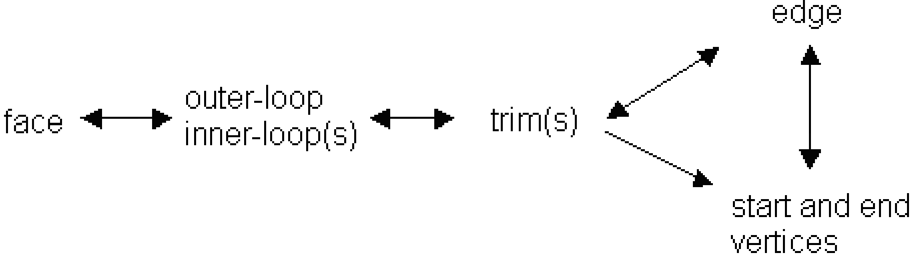



```
Dim brep_trim As New OnBrepTrim
brep_trim.
```

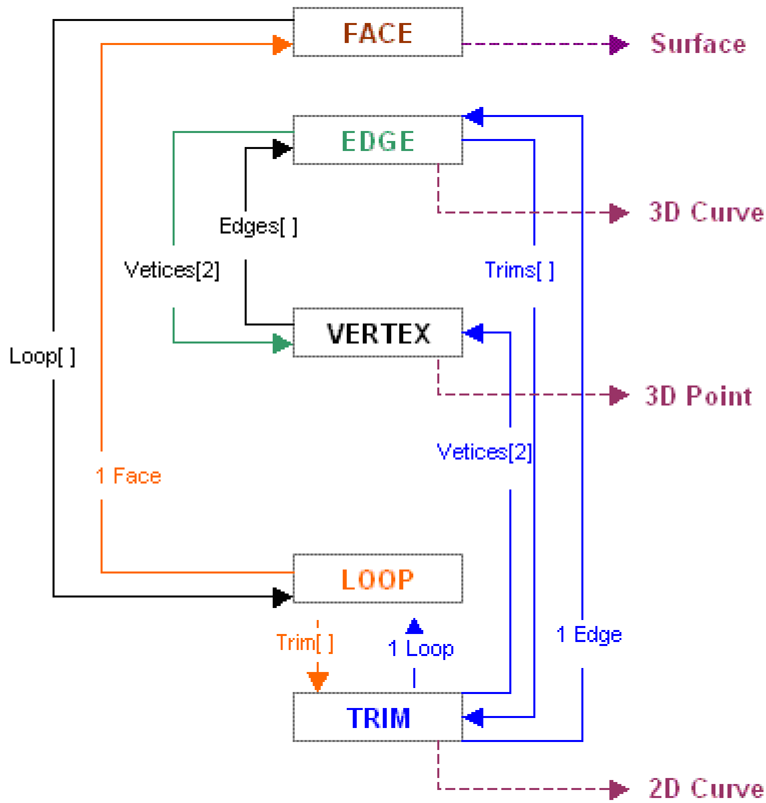



```
Dim brep_loop As New OnBrepLoop
brep_loop.
```

OnBrep メンバー変数とそれらがどのように参照しているかを下の図の通りです。
この情報から brep のどの部分の情報でも取得することが出来ます。
例えば、それぞれのフェイスは、そのループを知っており、それぞれのループはトリム情報をリストして持っており、そこから、エッジと始点・終点の頂点情報につながります。



以下に、brep 構成要素がお互いにどのように接続されているかのより詳細のダイアグラムを示します。



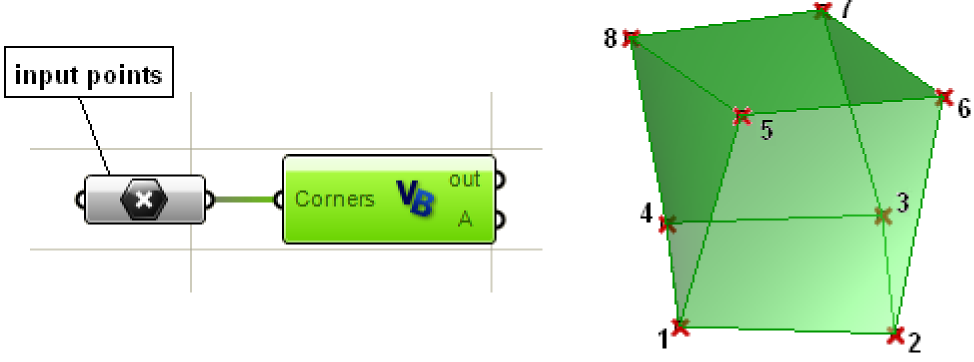
次のいくつかの例で、どのように OnBrep クラスが生成され、異なる構成要素をナビゲートし、brep 情報を抽出するかをを見ます。
また、グローバルファンクション同様に、いくつかのファンクションが使用されるのかを紹介します。

Create OnBrep

新しい OnBrep クラスのインスタンスを生成するにはいくつかの方法があります。

- 既存の brep をコピーする。
- 既存の brep の、フェイスをコピーあるいは抽出する。
- OnSurface を入力パラメーターとする Create ファンクションを使用する。現在、5 つの異なるオーバーロードファンクションがあります。
    - SumSurface から作成
    - fRevSurface から作成
    - fPlanarSurface から作成
    - OnSurface から作成
- global utility ファンクションを使用する方法
    - OnUtil から、ON_BrepBox, ON_BrepCone 等を使用して作成
    - RhUtil から、RhinoCreatEdgeurface や RhinoSweep1 等を使用して作成.

以下は、コーナー点から Brep のボックスを作成する例です。



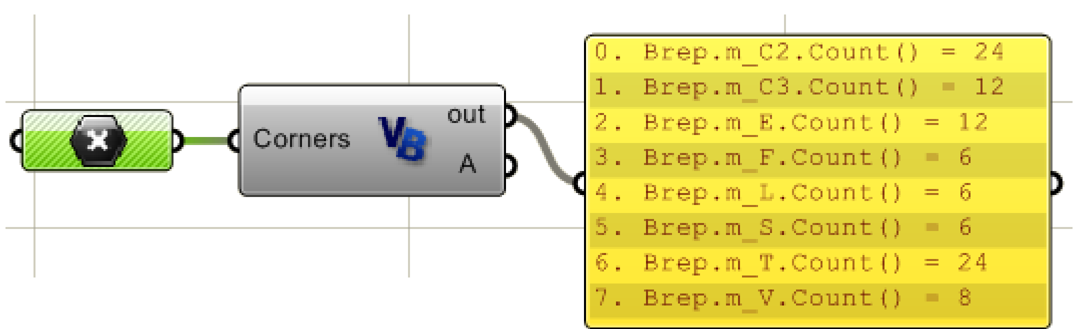

```
Sub RunScript(ByVal Corners As List(Of On3dPoint))

  ' Build the brep from corners
  Dim Brep As OnBrep = OnUtil.ON_BrepBox(Corners.ToArray())

  A = Brep
End Sub
```

Navigate OnBrep data （**OnBrep** データを導く）

以下は brep　ボックスの頂点の座標を抽出する例です。

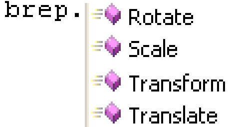
```
Sub RunScript(ByVal Corners As List(Of On3dPoint))

  ' Build the brep from corners
  Dim Brep As OnBrep = OnUtil.ON_BrepBox(Corners.ToArray())

  Dim myCorners As New List( Of On3dPoint )
  Dim v As OnBrepVertex
  Dim i As Integer

  For i = 0 To Brep.m_V.Count() - 1
    'get reference to OnBrepVertex
    v = Brep.m_V(i)

    'Get vertex point (loction)
    Dim pt As New On3dPoint
    pt = v.point

    'Add point to array
    myCorners.Add(pt)
  Next

  A = Brep
  B = myCorners
End Sub
```

次の例は、brep　box のジオメトリーとトポロジー構成要素(faces, edges, trims, vertices, etc)の数を取り出す例です。



OnBreps　の変形

OnGeometry から派生する全てのクラスは、4 つの変形機能を継承します。最初の 3 つは、最もポピュラーな、回転、スケール、移動になります。しかし、+Translate+ファンクションは、OnXform クラスによって 4x4 の変形マトリックスを定義します。
OnXform クラスは次のセクションで述べます。

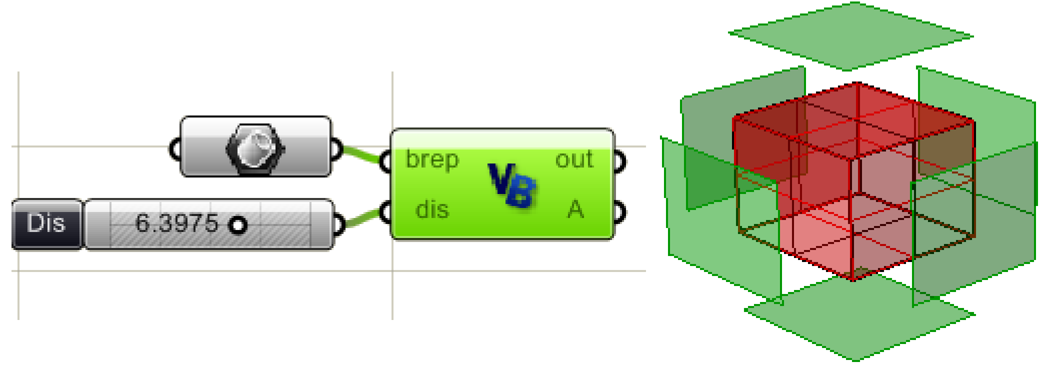


OnBrep　の編集

ほとんどの OnBrep クラスのメンバーファンクションは brep を生成し、編集する優れたツールです。しかしながら、Brep をブール演算、交差、分割等を行う多くのグローバルファンクションがあります。

McNeel$ wiki DotNET　に brep をスクラッチから作成するサンプルがありますので、それらを参考にすると良いでしょう。

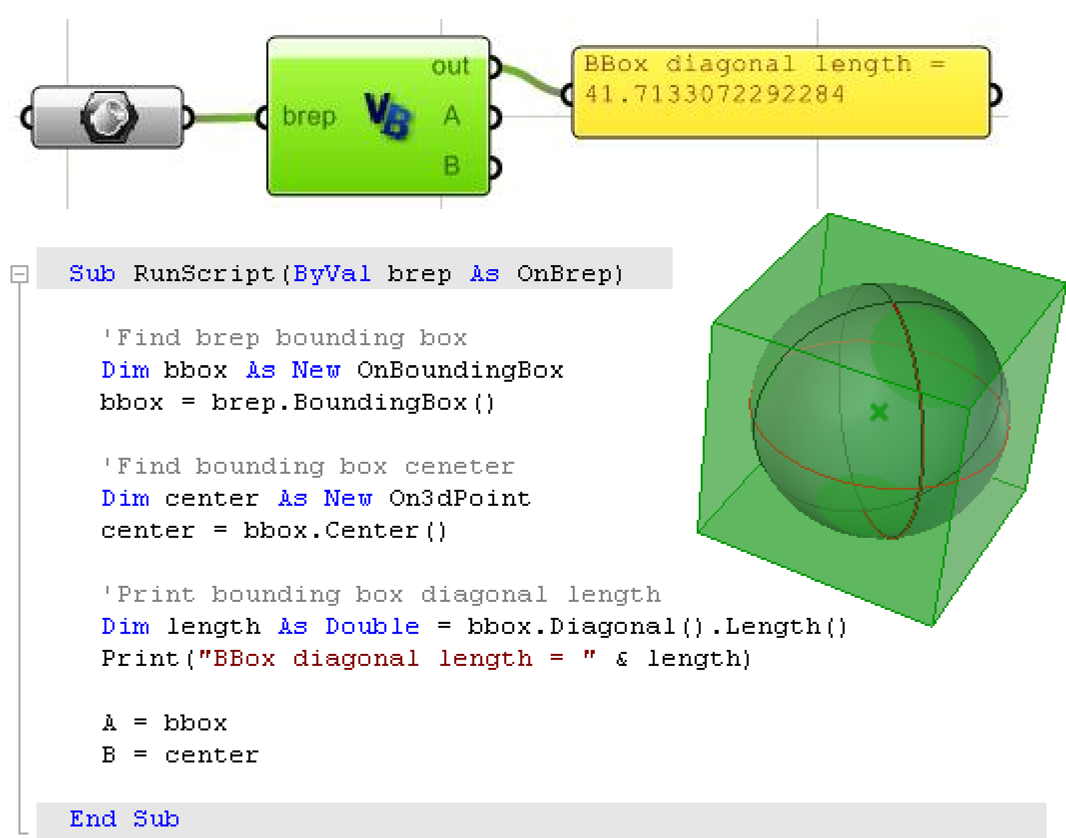
次の例は OnBrep フェイスを抽出し、バウンダリーボックスを使用して brep の中心から、指定距離だけ移動させる例です。



```
Sub RunScript(ByVal brep As OnBrep, ByVal dis As Double)

  Dim faces As New List(Of OnBrep)

  'Loop through brep faces to extract them
  For fi As Integer = 0 To brep.m_F.Count() - 1
    'Decalre new brep
    Dim face As New OnBrep
    face = brep.DuplicateFace(fi, False)

    'Add to faces array
    faces.Add(face)
  Next

  'Find brep bounding box center
  Dim center As New On3dPoint
  center = brep.BoundingBox().Center()

  'Loop through faces and move away from center by dis
  Dim dir As New On3dVector
  For i As Integer = 0 To faces.Count() - 1
    Dim face As OnBrep
    face = faces(i)

    'Find ceneter of each extracted face
    Dim face_center As On3dPoint
    face_center = face.BoundingBox().Center()

    'Find translation vector
    dir = face_center - center
    dir.Unitize()
    dir *= dis

    'Move face away from center
    face.Translate(dir)
  Next

  'Assign output
  A = faces


End Sub
```

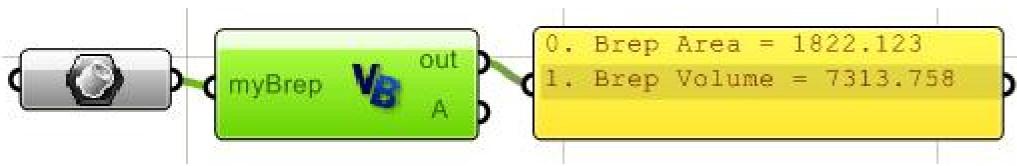
その他の **OnBrep** メンバーファンクション

OnBrep クラスは他にも多くのファンクションがありますが、それらは親のクラスもしくは OnBrep クラスから継承されます。OnBrep クラスを含む全てのジオメトリークラスは、+BoundingBox()+というメンバーファンクションを持っています。OnBoundingBox は、OpenNURBS クラスの一つで、ジオメトリーの占める空間情報（バウンディングボックス）を持っています。次の例は brep のバウンディングボックスを見つけ、その中心と、対角の距離を得る例です。




```
Sub RunScript(ByVal brep As OnBrep)

    'Find brep bounding box
    Dim bbox As New OnBoundingBox
    bbox = brep.BoundingBox()

    'Find bounding box ceneter
    Dim center As New On3dPoint
    center = bbox.Center()

    'Print bounding box diagonal length
    Dim length As Double = bbox.Diagonal().Length()
    Print("BBox diagonal length = " & length)

    A = bbox
    B = center

End Sub
```

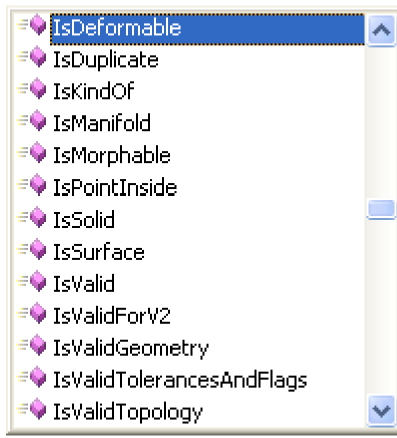
次は、マスプロパティーに関する例です。OnMassProperties クラスとそのいくつかのファンクションを紹介しています。



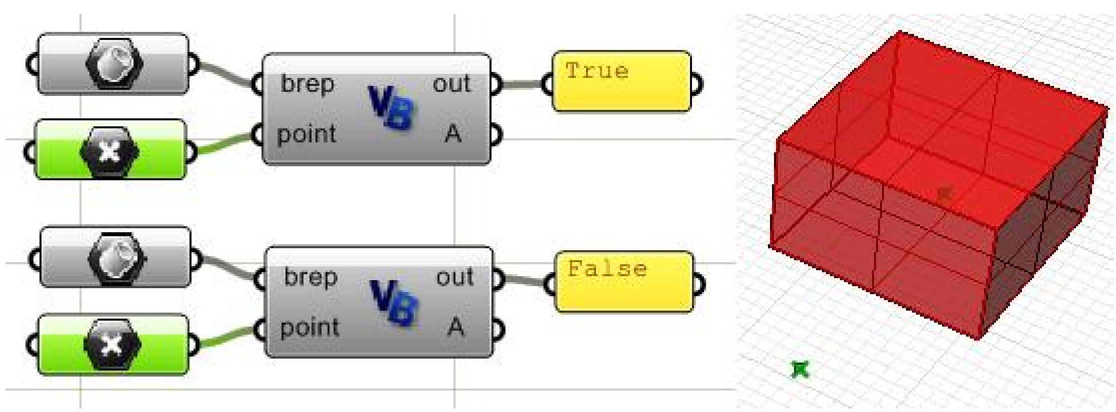
```
Sub RunScript(ByVal myBrep As Object)

    'Find and print brep area
    Dim a_mass As New OnMassProperties
    myBrep.AreaMassProperties(a_mass)
    Dim area As Double = a_mass.Area()
    Print("Brep Area = " & area)

    'Find and print brep volume
    Dim v_mass As New OnMassProperties
    myBrep.VolumeMassProperties(v_mass)
    Dim vol As Double = v_mass.Volume()
    Print("Brep Volume = " & vol)

End Sub
```

%Is+で始まるファンクションがありますが、通常、ブール値（+true+または+false+）を返します。それらは、brep のインスタンスについて問い合わせます。例えば、もし、brep が閉じたポリサーフェスであるかどうかを知りたい場合は、OnBrep.IsSolid()ファンクションを使用します。また brep が有効か、あるいは有効なジオメトリーを含むかのチェックは役に立ちます。
次は、OnBrep クラスの中の+問い合わせの+ファンクションのリストです。

```
Dim brep As New OnBrep
brep.Is…
```



次の例は点が、brep の内側にあるか、外側にあるかを判定するものです。



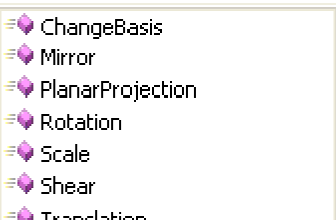
コードは以下の通りになります。

```
Sub RunScript(ByVal brep As OnBrep, ByVal point As On3dPoint)
  'Test if input point is inside brep
  Dim tol As Double = doc.AbsoluteTolerance()
  Dim strickly_inside As Boolean = True
  Dim is_inside As Boolean

  'Call brep function to test the point
  is_inside = brep.IsPointInside(point, tol, stricktly_inside)

  Print(is_inside)

End Sub
```

15.12 ジオメトリーの変換

OnXform クラスは変換マトリックスを格納し、操作するものです。これは、オブジェクトの移動、回転、スケール、シアーの定義を含みますが、限定されるものではありません。
OnXform の+m_xform+は、倍精度の 4x4 のマトリックスです。
このクラスはまた、逆行列や転置などの行列演算もサポートします。下は、変換の作成に関するメンバ関数のいくつかです。

```
Dim xform As New OnXform
xform.
```



auto-complete 機能（この機能は全てのファンクションに適用されます。）は一度、そのファンクションが選択されると、関連する全てのファンクションが表示されます。例えば、Translation ファンクションは、下図のように以下の 3 つの X,Y,Z 座標の数値、もしくはベクトルの入力を許可します。

```
Dim xform As New OnXform
xform.Translation(
```

1 of 2 Void OnXform.Translation (**dx As Double**, dy As Double, dz As Double)

2 of 2 Void OnXform.Translation (**d As RMA.OpenNURBS.IOn3dVector**)

以下は、OnXform の他のファンクションです。

```
Dim xform As New OnXform
xform.PlanarProjection(
```

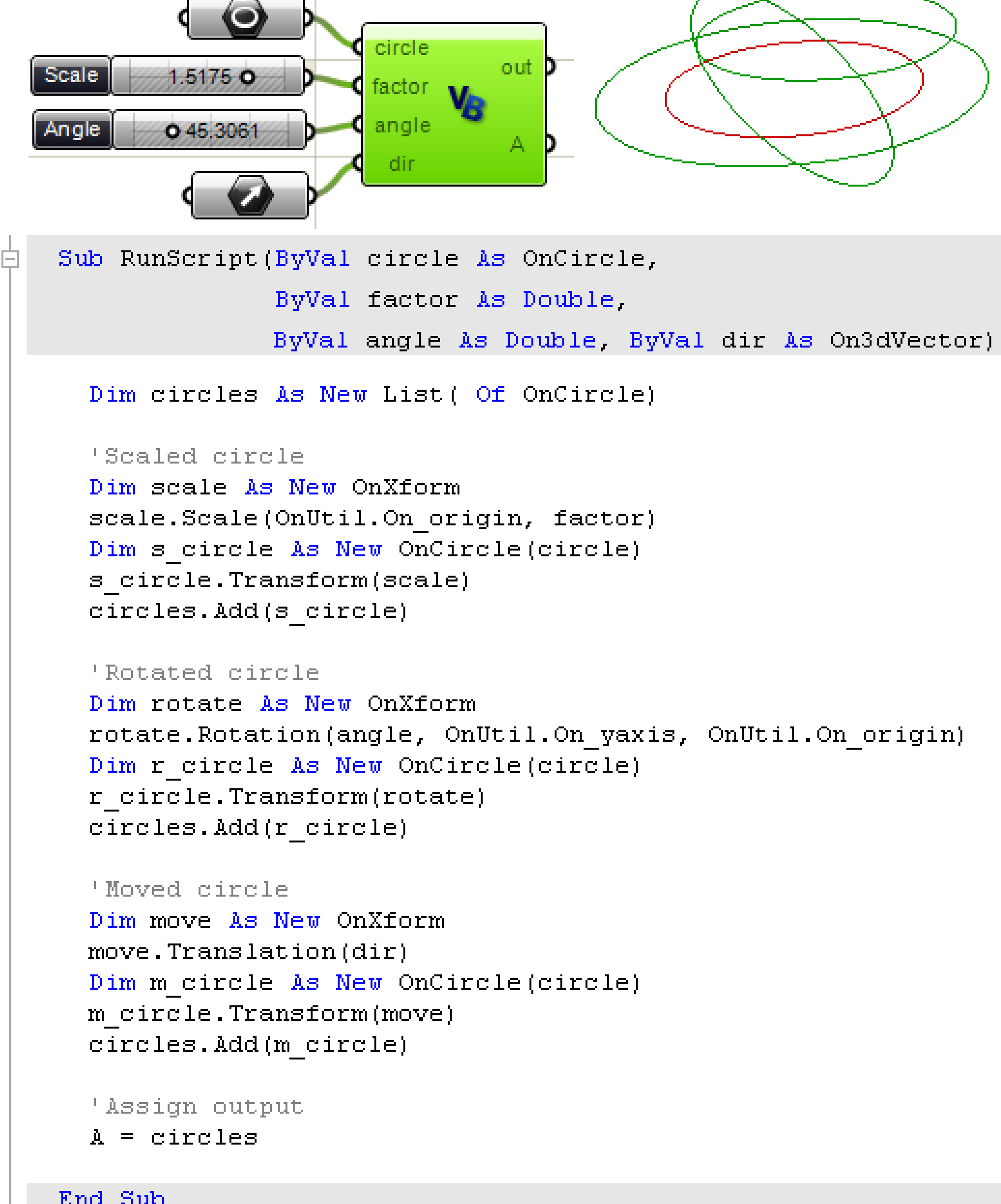
Void OnXform.PlanarProjection (**plane As RMA.OpenNURBS.IOnPlane**)
Get transformation that projects to a plane

```
Dim xform As New OnXform
xform.Shear(
```

Void OnXform.Shear (**plane As RMA.OpenNURBS.IOnPlane**,
x1 As RMA.OpenNURBS.IOn3dVector,
y1 As RMA.OpenNURBS.IOn3dVector, z1 As RMA.OpenNURBS.IOn3dVector)

Create shear transformation.

次の例は、入力された円に対して 3 つの円を出力する例です。最初の出力はオリジナルのものにスケールをかけて出力します。2 番目は、回転を、3 番目は移動を行います。



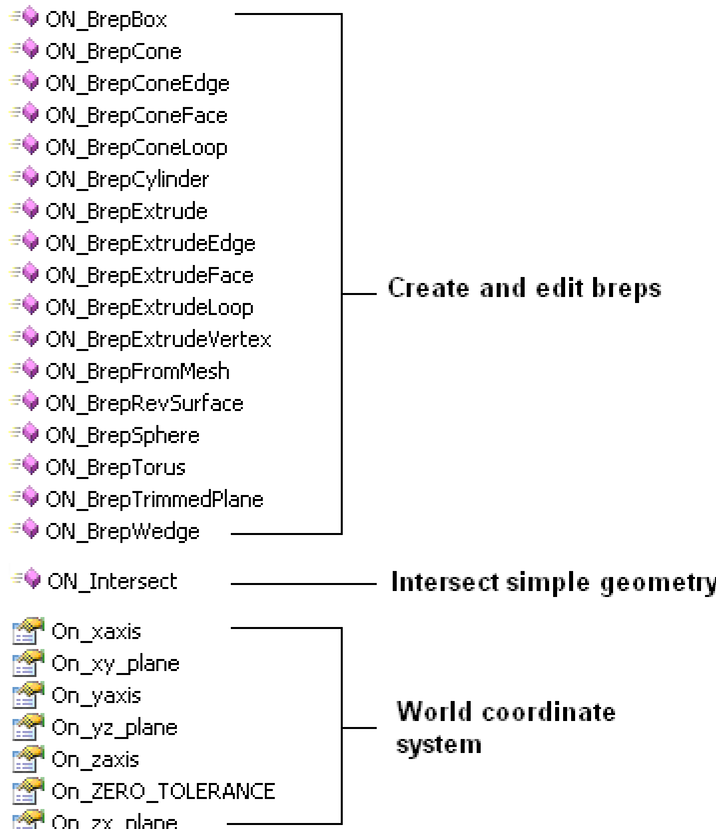

```
Sub RunScript(ByVal circle As OnCircle,
              ByVal factor As Double,
              ByVal angle As Double, ByVal dir As On3dVector)

  Dim circles As New List( Of OnCircle)

  'Scaled circle
  Dim scale As New OnXform
  scale.Scale(OnUtil.On_origin, factor)
  Dim s_circle As New OnCircle(circle)
  s_circle.Transform(scale)
  circles.Add(s_circle)

  'Rotated circle
  Dim rotate As New OnXform
  rotate.Rotation(angle, OnUtil.On_yaxis, OnUtil.On_origin)
  Dim r_circle As New OnCircle(circle)
  r_circle.Transform(rotate)
  circles.Add(r_circle)

  'Moved circle
  Dim move As New OnXform
  move.Translation(dir)
  Dim m_circle As New OnCircle(circle)
  m_circle.Transform(move)
  circles.Add(m_circle)

  'Assign output
  A = circles

End Sub
```

15.13 Global utility functions　グローバルユーティリティーファンクション

それぞれのクラスに付随するメンバーファンクションとは Rhino.Net SDK は、OnUtil と RhUtil という名前でグローバルファンクションを持ちます。
それらの例を幾つか紹介します。

OnUtil

以下が、ジオメトリーに関する OnUtil で、使用出来るファンクションです。



OnUtil intersections　OnUtil の交差

ON_Intersect Utility ファンクションは、11 の機能を持ちます。以下が、交差したジオメトリーの一覧と、その戻り値のリストです。（+I+が先行して %IOnLine+になっているのは、定数のインスタンスを意味します。）

交差しているジオメトリー	出力
IOnLine with IOnArc	ラインパラメーター(t0 & t1)と円弧の点(p0 & p1)
IOnLine with IOnCircle	ラインパラメーター(t0 & t1)と円の点(p0 & p1)
IOnSphere with IOnShere	OnCircle
IOnBoundingBoc with IOnLine	ラインパラメーター (OnInterval)
IOnLine with IOnCylinder	2 つのポイント (On3dPoint)
IOnLine with IOnSphere	2 つのポイント (On3dPoint)
IOnPlane with IOnSphere	OnCircle
IOnPlane with IOnPlane with IOnPlane	On3dPoint
IOnPlane with IOnPlane	OnLine
IOnLine with IOnPlane	パラメーター (倍精度)
IOnLine with IOnLine	パラメーターa & b (on first and second line as Double)

以下は、ラインとプレーンと球が交差した結果を出力する例です。

```
Sub RunScript(ByVal L As OnLine, ByVal P As OnPlane, ByVal R As Double)

  Dim point0 As New On3dPoint
  Dim point1 As New On3dPoint
  Dim circle0 As New OnCircle

  'Declare the sphere
  Dim sphere As New OnSphere(OnUtil.On_origin, R)

  'Intersect line with sphere
  OnUtil.ON_Intersect(L, sphere, point0, point1)

  'Intersect plane with sphere
  OnUtil.ON_Intersect(P, sphere, circle0)

  'Assign output
  Pt0 = point0
  Pt1 = point1
  Circle = circle0
End Sub
```

RhUtil

Rhino ユーティリティー(RhUtil)は、さらに多くのジオメトリーに関連したファンクションを持ちます。このリストは、新しいバージョンがリリースされるときにユーザーのリクエストによって拡張されていきます。
このスナップショットは、ジオメトリーに関連するものです。

RhUtil.

Points

- RhinoArePointsCoplanar
- RhinoPointInPlanarClosedCurve
- RhinoProjectPointsToBreps
- RhinoIsPointInBrep
- RhinoIsPointOnFace

Curve

- RhinoConvertCurveToPolyline
- RhinoCurveBrepIntersect
- RhinoCurveFaceIntersect
- RhinoDivideCurve
- RhinoDoCurveDirectionsMatch
- RhinoExtendCrvOnSrf
- RhinoExtendCurve
- RhinoExtendLineThroughBox
- RhinoExtrudeCurveStraight
- RhinoExtrudeCurveToPoint
- RhinoFairCurve
- RhinoFitCurve
- RhinoFitLineToPoints
- RhinoInterpCurve
- RhinoInterpolatePointsOnSurface
- RhinoMakeCubicBeziers
- RhinoMakeCurveClosed
- RhinoMakeCurveEndsMeet
- RhinoMergeCurves
- RhinoOffsetCurve
- RhinoOffsetCurveOnSrf
- RhinoPlanarClosedCurveContainmentTest
- RhinoPlanarCurveCollisionTest
- RhinoProjectCurvesToBreps
- RhinoPullCurveToMesh
- RhinoRebuildCurve
- RhinoRemoveShortSegments
- RhinoRepairCurve
- RhinoShortPath
- RhinoSimplifyCurve
- RhinoSimplifyCurveEnd

Surface

- RhinoChangeSeam
- RhinoCreateSurfaceFromCorners
- RhinoExtendSurface
- RhinoFitSurface
- RhinoIntersectSurfaces
- RhinoMakeG1Surface
- RhinoRailRevolve
- RhinoRebuildSurface
- RhinoRepairSurface
- RhinoRetrimSurface
- RhinoRevolve
- RhinoSrfControlPtGrid
- RhinoSrfPtGrid

Mesh

- RhinoMeshBooleanDifference
- RhinoMeshBooleanIntersection
- RhinoMeshBooleanSplit
- RhinoMeshBooleanUnion
- RhinoMeshBox
- RhinoMeshCone
- RhinoMeshCylinder
- RhinoMeshObjects
- RhinoMeshOffset
- RhinoMeshPlane
- RhinoMeshSphere
- RhinoRepairMesh
- RhinoSplitDisjointMesh
- RhinoUnifyMeshNormals

Brep

- RhinoBooleanDifference
- RhinoBooleanIntersection
- RhinoBooleanUnion
- RhinoBrepCapPlanarHoles
- RhinoBrepClosestPoint
- RhinoBrepGet2dProjection
- RhinoBrepGet2dSection
- RhinoBrepSplit
- RhinoCreate1FaceBrepFromPoints
- RhinoCreateEdgeSrf
- RhinoIntersectBreps
- RhinoJoinBrepNakedEdges
- RhinoJoinBreps
- RhinoMakePlanarBreps
- RhinoMergeAdjoiningEdges
- RhinoMergeBrepCoplanarFaces
- RhinoMergeBreps
- RhinoRepairBrep
- RhinoSplitBrepFace
- RhinoStraightenBrep
- RhPlanarRegionBoolean
- RhPlanarRegionDifference
- RhPlanarRegionIntersection
- RhPlanarRegionUnion
- RhinoSdkLoft
- RhinoSdkLoftSurface
- RhinoSweep1
- RhinoSweep2

Utility

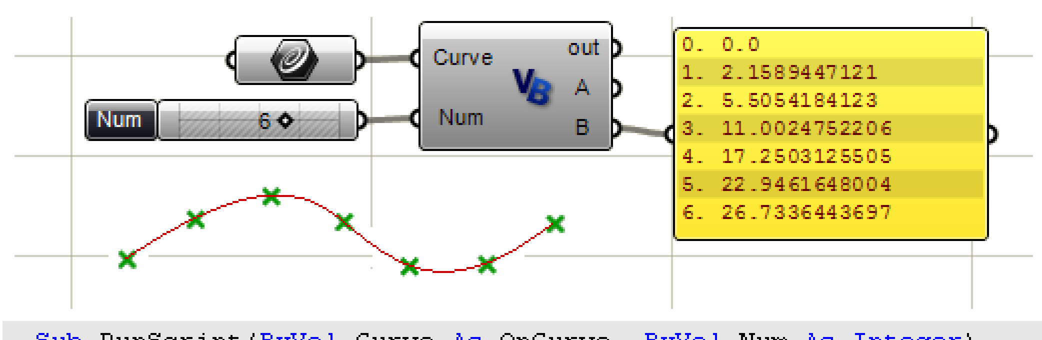
- RhinoActiveCPlane
- RhinoApp
- RhinoFitPlaneToPoints
- RhinoPlaneThroughBox
- RhinoProjectToPlane
- RhinoTriangulate3dPolygon

RhUtil Divide curve

カーブを、指定されたセグメントの数や、長さで分割するには+RhUtil.RhinoDivideCurve+ファンクションを使用します。
ファンクションのパラメーターは下記の通りです。

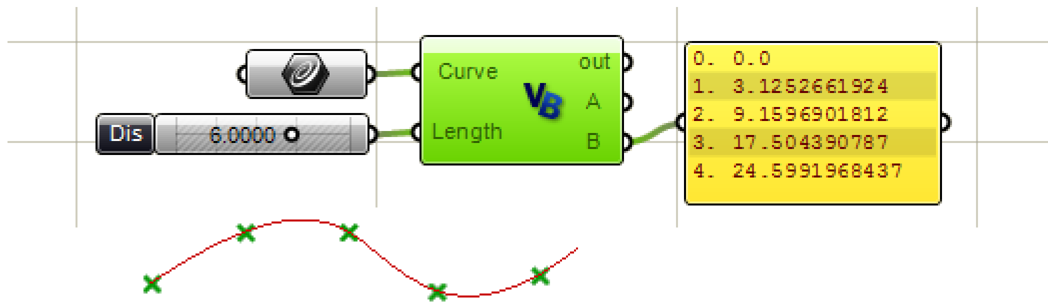
RhinoDivideCurve: ファンクション名
Curve: 分割するカーブ
Num: セグメントの数
Len: 分割するときのカーブの長さ
False: False 時、分割は終点から始まる。
True: True 時、分割は始点から始まり、終点を戻り値として返す。
crv_p: 分割する点群のリスト
crv_t: カーブ上で分割する点群の持つパラメーターのリスト

セグメントの数で、カーブを分割する例:




```
Sub RunScript(ByVal Curve As OnCurve, ByVal Num As Integer)

  Dim crv_p As New On3dPointArray
  Dim crv_t As New Arraydouble

  'A utility function to divide by number or curve length
  RhUtil.RhinoDivideCurve(Curve, Num, 0, False, True, crv_p, crv_t)

  A = crv_p
  B = crv_t

End Sub
```

弦弧長で、カーブを分割する例:



```
Sub RunScript(ByVal Curve As OnCurve, ByVal Len As Double)

  Dim crv_p As New On3dPointArray
  Dim crv_t As New Arraydouble

  'A utility function to divide by number or curve length
  RhUtil.RhinoDivideCurve(Curve, 0, Len, False, True, crv_p, crv_t)

  A = crv_p
  B = crv_t

End Sub
```

RhUtil Curve through points (interpolate curve)

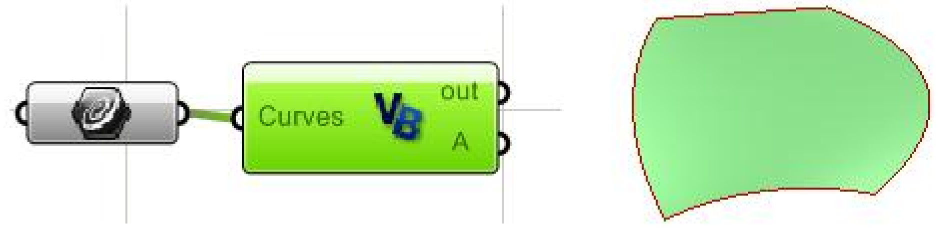
RhinoInterpCurve: ファンクション名.
3: カーブの次数
pt_array:カーブを生成するポイントの配列
Nothing: 開始接点を指定しない。
Nothng: 終了接点を指定しない。
0: ノットは均等に分割される

次の例は、点群を+On3dPoints+として読込、その点を通過する NURBS カーブとして出力する例です。

```
Sub RunScript(ByVal Points As List(Of On3dPoint))

  Dim pt_array As New ArrayOn3dPoint
  Dim i As Integer

  For i = 0 To Points.Count() - 1
    pt_array.Append(Points(i))
  Next

  'Create an interpolated nurbs curve
  Dim crv As New OnNurbsCurve
  crv = RhUtil.RhinoInterpCurve(3, pt_array, Nothing, Nothing, 0)

  If( crv.IsValid() ) Then
    'Set return vlue to list
    A = crv
  End If

End Sub
```

RhUtil を使用してエッジサーフェスを作成する。

以下は、4 つのカーブのリストから、4 エッジサーフェスを作成する例です。




```
Sub RunScript(ByVal Curves As List(Of OnCurve))

  Dim nc_list(3) As OnNurbsCurve
  For i As Integer = 0 To 3
    nc_list(i) = New OnNurbsCurve()
  Next

  ' Get nurb form of each curve
  For i As Integer = 0 To 3
    Curves(i).GetNurbForm(nc_list(i))
  Next

  ' Create the edgesurface
  Dim Brep As OnBrep = RhUtil.RhinoCreateEdgeSrf(nc_list)

  A = Brep
End Sub
```

16 *Help*

Rhino DotNET SDK の情報

McNeel Wiki サイトでは、様々な情報とサンプルが公開されており、随時、アップデートされます。アドレスは下記となります。
http://en.wiki.mcneel.com/default.aspx/McNeel/Rhino4DotNetPlugIns.html

フォーラムとディスカッショングループ

Rhino のコミュニティーは非常にアクティブです。
下記の Grasshopper のディスカッションフォーラムに参加するとよいでしょう。
http://grasshopper.rhino3d.com/

また、下記の McNeel の開発者向ページの Rhino Developer Newsgroup（Rhino の開発者用ニュースグループ）に質問を投稿する事も出来ます。
http://www.rhino3d.com/developer.htm

Visual Studio でのデバッグ

さらに複雑で、Grasshopper の Script コンポーネントでデバッグが困難な場合は、Microsoft 社が提供する+Visual Studio Express+の 無償のものあるいはフルバージョンを使用すると良いでしょう。詳細は下記 URL を見てください。
http://en.wiki.mcneel.com/default.aspx/McNeel/DotNetExpressEditions

もし、+Visual Studio Express+の フルバージョンを持っているのであれば、下記 URL がより参考になります。
http://en.wiki.mcneel.com/default.aspx/McNeel/Rhino4DotNetPlugIns.html

Grasshopper スクリプトサンプル

前述の、+http://grasshopper.rhino3d.com/%のギャラリーとディスカッションフォーラムでサンプルのスクリプトが入手できるでしょう。

Notes