GRASSHOPPER PRIMER

FOR VERSION 0.6.0007

BY ANDREW PAYNE & RAJAA ISSA

한국어판

Korean version rev.01

Translated by KICHAN U

서문

Grasshopper의 환상적인 신세계에 오신 여러분을 환영합니다. 본서는 Primer의 두 번째 판입니다. 이는 Rajaa Issa의 엄청난 공헌 없이 불가능한 작업이었습니다. Rajaa는 Robert McNeel사의 개발자이며 ArchCut을 포함한 여러 라이노 Plugin과 잘 알려진 Paneling Tools를 제작했습니다. 본서는 첫 번째 판보다 70페이지 이상의 새로운 장들을 추가하여 자신만의 맞춤형 scripting 컴포넌트 생성에 대해 훨씬 더 자세한 가이드를 제공하고 있습니다.

본 매뉴얼의 발행은 두 가지 일과 관련이 있습니다. 첫째는 Grasshopper 버전 0.6.0007의 출시이며, 이미 완성도 있게 개발된 Grasshopper Platform 위에 더욱 발전된 기능들을 업그레이드한 것입니다. 기존 사용자들은 미묘한 변화 그리고 그다지 미묘하지 않은 변화들을 발견할 것입니다. 데이터가 현재 버전에 저장되는 방법의 차이가 있고, 이전 버전에서 작성된 정의(definitions)의 일부가 사용 불가능한 경우도 있습니다. 본 매뉴얼은 새로운 사용자와 기존사용자들이 본 소프트웨어 체계의 변화를 이해하는데 도움을 줄 것입니다. 둘째로는 본 매뉴얼은 캘리포니아 미술대학에서 열린 '흐름: 파라메트릭 환경에서의 건축(FLUX: Architecture in a Parametric Landscape)' 컨퍼런스와도 같은 시기에 출판되었습니다. 이 컨퍼런스는 파라메트릭 모델링, 디지털 패브리케이션, 그리고 스크립팅과 같은 디자인 기술의 변화를 통해 현대건축과 디자인을 탐험하는 것이었습니다. 무엇보다도 이는 전시회와 파라메트릭 소프트웨어와 깊게 관련된 일련의 워크샵으로 구성될 것입니다. 필자는 영광스럽게도 그래스호퍼 모델링 입문을 강의하게 되었고, Rajaa Issa와 Gil Akos는 그래스호퍼 모델링 고급과정과 VB.Net 스크립팅 워크샵을 담당하게 되었습니다. 우리는 이번 Primer 매뉴얼에 많은 새로운 정보를 담았습니다. 이 플러그인에 대해 더 많이 알고 싶어 하는 사용자들에게 좋은 자료가 되기를 바랍니다. 그러나 이 소프트웨어의 가장 큰 이점 중 하나는 사용자인 여러분들, 즉 더 많은 사람들이 파라메트릭 모델링을 탐험하고 이해하기 시작하면서, 전체 사용자 그룹이 혜택을 받게 된다는 것입니다. 필자는 본 Primer 매뉴얼을 읽는 여러분 각자가 온라인 커뮤니티에 가입하여 포럼에 여러분의 질문을 올리고, 그 문제의 해결책을 기꺼이 나누고자하는 분들이 더욱 많아지기를 바랍니다.

보다 자세한 정보가 필요하신 분들은 아래의 싸이트를 방문해주시기 바랍니다.
http://www.grashopper.rhino3d.com
감사합니다. 그리고 행운을 빕니다!

앤드류 페인 (Andrew Payne)
리프트 건축설계사무소 (LIFT architects)
www.liftarchitects.com
라자 이싸 (Rajaa Issa)
로버트 맥닐사 (Robert McNeel and Associates)
http://www.rhino3d.com/

번역의 변

"Geometry lies at the core of the architectural design process."

역자가 2008년 비엔나에서 참석했던 "Advanced Architectural Geometry 08" 세미나의 홈페이지의 첫 화면을 장식한 문구다. 역자가 항상 마음에 담고 있던 말이기도 하다. 그러나 내가 그 동안 건축에 몸담고 일해 온 해 수만큼, 기하학은 나와 멀어져 있었던 것이다. 지금까지의 국내 건축계는 여러 부분에서 많은 발전을 이루어 왔다. 반면, 많은 것을 잃어버리기도 했던 것 같다. 디자인의 중심에서 기하학을 버린 것이 그것이 아니었을까. 그렇다면 어떻게 기하학을 건축으로 되돌릴 수 있을까.

거기에 Grasshopper라는 대안이 있다.
역자는 이 비엔나 세미나에서 Grasshopper를 접하게 된다. 이것은 Rhino의 일부 기능인 History 기능이 발전한 것인데, 건축 속의 기하학을 이해하고 디자인에 적극적으로 사용하도록 돕는 탁월한 플러그인이다.

라이노3d와 Grasshopper를 개발한 McNeel사는 파라메트릭 디자인이 다가오는 건축의 트렌드라는 것을 미리 알고 잘 준비해온 것 같다.

역자는 국내 최초 BIM 턴키 프로젝트인 용인시민 체육공원 실무에서 그래스호퍼 기능을 일부 사용하였고, 그 가능성을 확인하였다. 앞으로 우리나라의 많은 건축학도와 실무진들이 이를 배워서 사용하였으면 좋겠다는 생각에 영문 튜토리얼을 번역하기에 이르렀다.

끝으로 공동번역 및 편집작업에 수고한 우리 (주) 알피종합건축사 사무소 직원들, 특히 박아름, 신재원, 이주성, 윤길준에게 감사한다. 또 베타 테스터로 번역판 리뷰를 진행해 준 김한결, 송재우에게도 깊은 감사의 마음을 전한다.

*그래스호퍼 프로그램이 영문버전인 관계로 프로그램명과 프로그램 메뉴는 번역하지 않고 원문 그대로 사용하였습니다.

2010년 11월

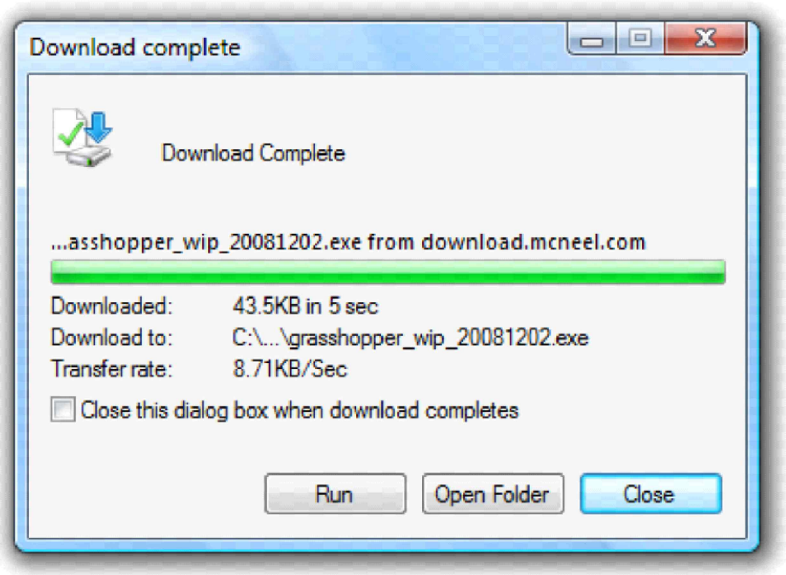
(주) 알피종합건축사 사무소
대표 유 기 찬 (U_BIM)
www.rpaec.com

Table of Contents

1 시작하기

Grasshopper 설치하기

Grasshopper 플러그인을 설치하려면 http://grasshopper.Rhino3d.com에 접속하세요. 홈페이지 좌측상단의 **Download** link를 클릭하고 다음 화면이 뜨면, 자신의 이메일 주소를 입력합니다. 이제, Download link를 클릭하고 메뉴에서 **Save Target As**를 선택합니다. 자신의 하드 드라이브의 위치를 선택하고 실행파일을 저장합니다. (주의: 파일은 반드시 로컬 컴퓨터의 하드 드라이브에 저장되어야만 합니다. 네트워크상에서는 파일이 로드될 수 없습니다.)

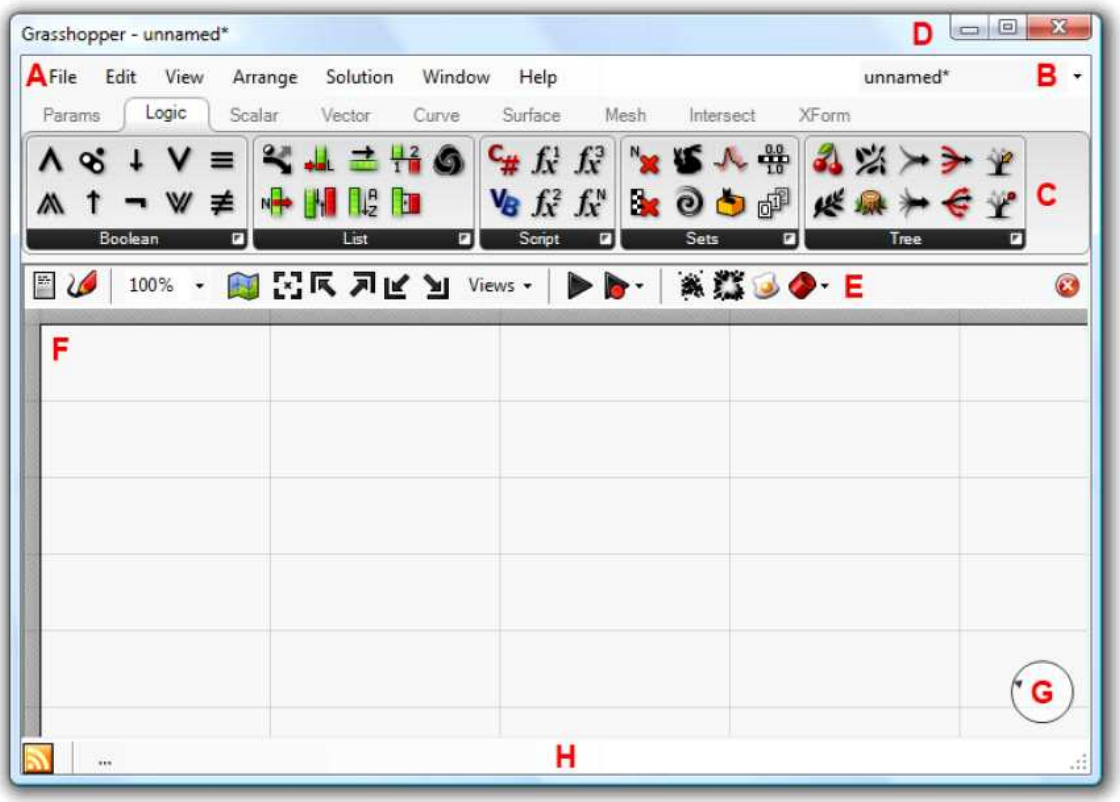


다운로드 대화상자에서 Run을 선택하고 설치 안내문을 따라 설치합니다. (주의: 플러그인이 제대로 설치되기 위해서는 Rhino 4.0의 SR4b 또는 상위버전이 이미 자신의 컴퓨터에 설치되어 있어야합니다.)

2 *인터페이스**

메인 화면

플러그인을 로드하고, Rhino 명령창에서 "Grasshopper"를 입력하면, 아래와 같은 Grasshopper 창이 새로 열립니다.



이 인터페이스는 많은 요소들을 포함하고 있습니다. 그 중 대부분은 라이노 유저들에게는 친숙한 것들입니다.

A. 메인 메뉴 바

이 메뉴는 전형적인 윈도우 메뉴와 유사하며, 추가적으로 오른쪽 상단(B)에 파일검색 메뉴가 위치합니다. 파일검색 메뉴의 드롭다운 박스를 이용하여 로드된 다른 파일들을 빠르게 전환하며 열어볼 수 있습니다. 활성창 위주로 명령이 실행되기 때문에 단축키를 사용할 때는 주의하시기 바랍니다. 활성창은 라이노 창 또는 Grasshopper 플러그인 창 또는 라이노 내부의 다른 창이 될 수도 있습니다. 아직 되돌리기(undo) 기능을 사용할 수 없기 때문에 잘라내기(Ctrl-X), 저장하기(Ctrl-S) 그리고 삭제(Del) 단축키를 사용할 때 주의하시기 바랍니다.

*출처: RhinoWiki
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryPluginInterfaceExplained.html

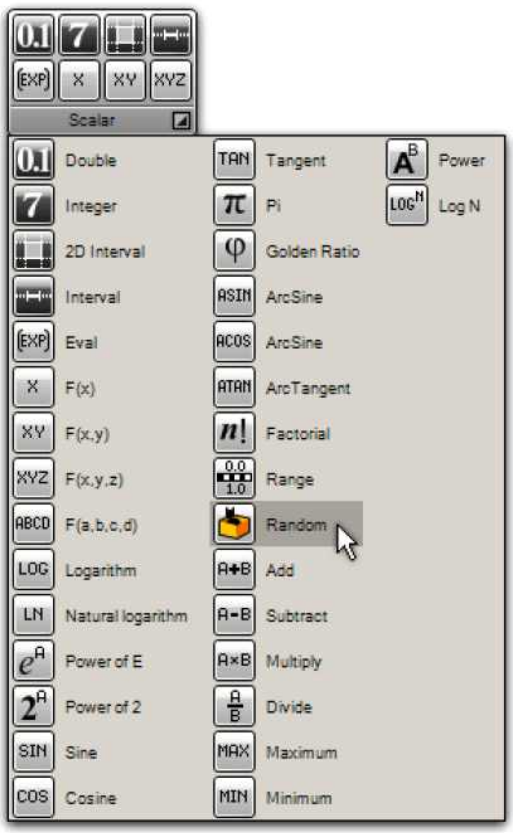
B. 파일검색 메뉴

이 메뉴는 앞에서 살펴본 바와 같이 로드된 파일들을 전환하며 열어볼 때 사용할 수 있습니다.

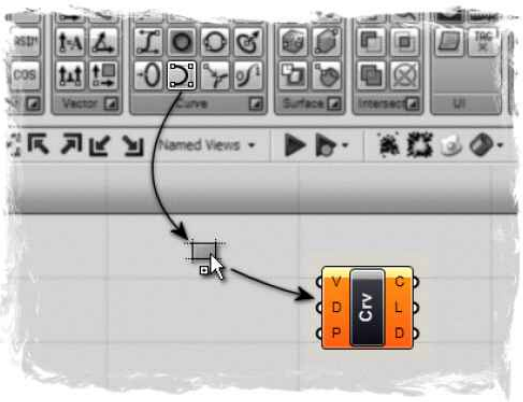
C. 컴포넌트 패널

이 부분에는 모든 컴포넌트 카테고리들이 보입니다. 모든 컴포넌트들은 특정 카테고리 (예를 들면 모든 원시 데이터 유형들을 위한 "Params" 또는 모든 커브를 위한 "Curves" 그리고 모든 카테고리들이 독립적인 툴바 패널 형태로 활용 가능합니다. 툴바의 높이와 폭은 조정할 수 있어서 각 카테고리마다 화면에 떠있는 버튼의 수도 조절할 수 있습니다.

툴바 패널은 그 카테고리에 속한 모든 컴포넌트들을 포함하고 있습니다. 컴포넌트의 수가 상당히 많기 때문에 가장 최근에 사용한 몇 개의 아이템들을 우선적으로 보여줍니다. 전체 내용을 보기 위해서는 패널 하단의 바를 클릭하시면 됩니다.



클릭하면 카테고리 패널이 확장되어 모든 컴포넌트에 접근할 수 있습니다. 팝업 리스트의 컴포넌트를 클릭하거나 리스트로부터 바로 드래그하여 캔버스에 컴포넌트를 불러올 수 있습니다. 카테고리 패널의 컴포넌트를 클릭하면 추후 손쉽게 참조하기 위해 툴바에 위치시킬 수 있습니다. **버튼을 클릭하는 것이 캔버스에 컴포넌트를 추가하는 것은 아닙니다.** 캔버스에 컴포넌트를 띄우려면 반드시 드래그하여 캔버스 위로 올려놓아야 합니다.



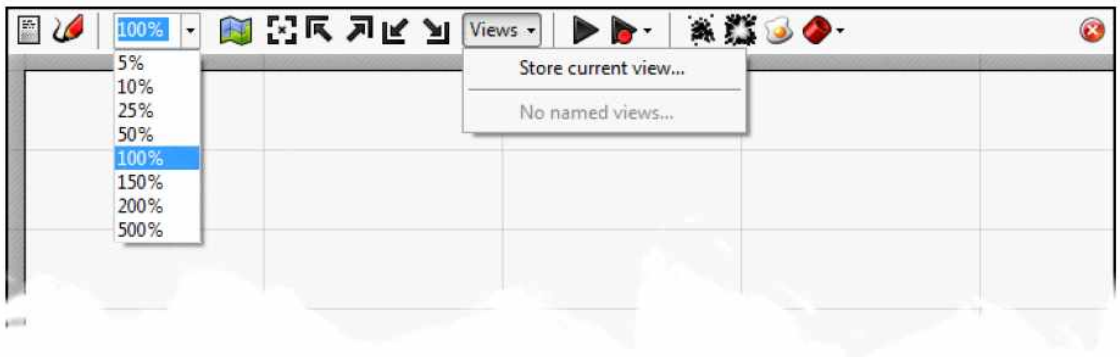
또한 캔버스 상의 어느 곳이든 더블클릭하여 검색 팝업창을 띄우고 컴포넌트의 이름을 입력하여 컴포넌트를 불러올 수 있습니다. 컴포넌트의 이름을 입력하면, 찾고자 하는 매개변수나 컴포넌트의 목록을 볼 수 있습니다.

윈도우 타이틀 바: D

편집용 윈도우 타이틀 바는 마이크로소프트 윈도우의 다른 대화창들과 다르게 조정됩니다. 보통 창을 최소화하거나 최대화 할 때 더블클릭을 하는데, 타이틀 바를 더블클릭하면 대화창을 숨기거나 열게 됩니다. 이는 플러그인 화면과 라이노 화면을 효과적으로 전환하기 위한 최적의 방법입니다. 이 방법으로 플러그인의 창을 화면 아래나 다른 창 뒤로 옮기지 않고도 최소화할 수 있기 때문입니다. 주의할 점은 플러그인 창을 닫으면 뷰포트 상의 Grasshopper geometry 미리보기가 사라진다는 것입니다. 그러나 파일이 실제로 닫히는 것은 아닙니다. 다시 라이노 명령창에 Grasshopper를 입력하면, 플러그인 창은 같은 파일이 로드된 상태로 돌아오게 됩니다.

캔버스 툴바: E

캔버스 툴바는 자주 사용되는 기능들에 대해 빠르게 접근하도록 합니다. 모든 툴은 메뉴를 통해서도 이용가능하며, 필요시 툴바를 숨길 수 있습니다.
(숨겨진 툴바를 다시 열고 싶을 때는 View menu로부터 열 수 있습니다.)



캔버스 툴바에는 아래의 툴들이 포함됩니다. (좌측으로부터 우측 순으로 설명:)

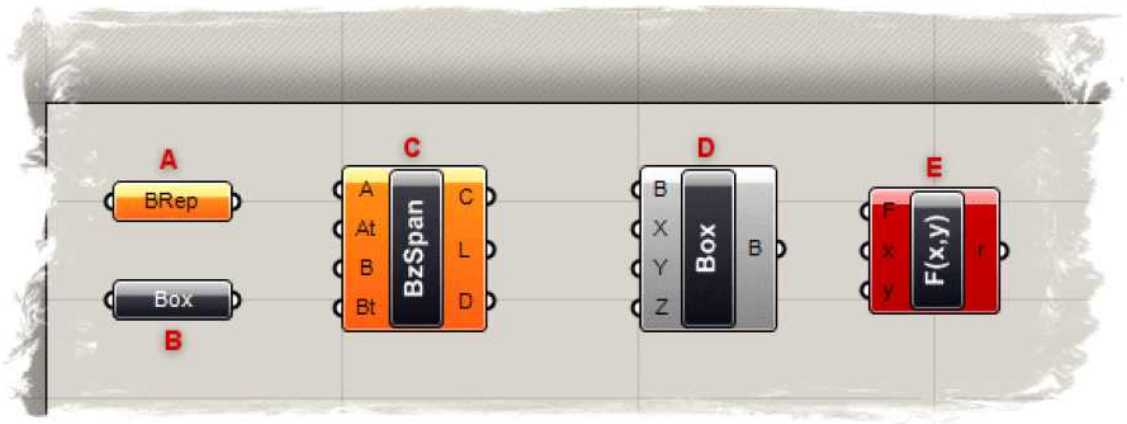
1. Definitiona properties editor [정의속성 편집자]
2. Sketch Tool [스케치 툴]: 포토샵 또는 윈도우 페인트의 Pencil-Type 툴처럼 작동합니다. 선의 유형과 굵기, 색상을 조절할 수 있습니다. 그러나 미리 정의된 형태나 직선을 그리기는 어려울 수 있습니다. 이를 해결하려면, 캔버스 위에 아무 스케치 선을 그리고 선 위에서 오른쪽 클릭합니다. 그리고 "Load from Rhino"를 선택하고, 라이노 화면에서 미리 정의된 선을 선택합니다. (이는 사각형, 원, 별 등과 같은 2차원 형태가 될 수 있습니다.) 참조할 형태를 선택하고 엔터를 치면 사전에 그려진 스케치 선이 라이노에서 참조한 선의 형태에 따라 재구성됩니다.
3. Zoom defaults [기본 Zoom 조절]
4. Navigation Map [네비게이션 팝업 맵] 캔버스 상에 더 작은 도표 창을 열어 스크롤과 팬

을 사용하지 않고도 캔버스 주위를 빠르게 살펴볼 수 있도록 합니다. 포토샵의 네비게이션 창과 유사합니다.
5. Zoom Extents[전체 화면보기] (캔버스 상 작성된 정의가 너무 커서 화면상에 들어가지 않을 경우, 전체 회로도를 화면의 중심영역으로 맞추어 줍니다.)
6. 부분 화면보기 (네 개의 버튼을 통해 캔버스의 네 모서리를 기준으로 화면을 맞추어 줍니다)
7. Named views (저장할 메뉴를 보여주고 이미 저장된 뷰를 불러올 수 있습니다.)
8. Rebuild solution [솔루션 재생성] (History 정의를 완전히 다시 생성하도록 합니다.)
9. Rebuild event [이벤트 재생성] (기본값에 의해, Grasshopper는 라이노와 캔버스 상의 변화에 반응합니다. 이 메뉴로부터 이 기능을 비활성화 할 수 있습니다.)
10. Cluster compactor (선택된 모든 객체들을 하나의 클러스터 객체로 전환해 줍니다.) **클러스터 객체들은 아직 마무리된 것은 아니며 추후 완전히 재작업될 가능성이 더 크다고 볼 수 있습니다. 현재 파일에 이 기능을 사용하는 것을 주의하시기 바랍니다.**
11. Cluster exploder (선택된 모든 클러스터들을 각각의 객체로 분리해 줍니다.) **클러스터 객체들은 아직 마무리된 것은 아니며 추후 완전히 재작업될 가능성이 더 크다고 볼 수 있습니다. 현재 파일에 이 기능을 사용하는 것을 주의하시기 바랍니다.**
12. Bake tool [굽기 툴] (선택된 모든 컴포넌트들을 실제 라이노 객체들로 전환해 줍니다.)
13. Preview settings [미리보기 설정값] (Grasshopper에서 생성된 기하형태는 기본값으로 미리보기가 가능합니다. 각 객체마다 미리보기를 활성화 또는 비활성화 할 수 있으며, 모든 객체에 대해 미리보기 설정을 덮어씌울 수도 있습니다. 미리보기 설정을 꺼두면 커브나 트림된 면을 가진 객체의 경우 화면 보기 속도가 월등히 빨라질 수 있습니다.
14. Hide button [숨기기 버튼] 이 버튼은 캔버스 툴바를 숨깁니다. 숨겨진 툴바는 View 메뉴를 통해 다시 열 수 있습니다.

-캔버스 : 프로젝트를 정의하고 편집하는 실제 편집기입니다. 캔버스에는 정의를 이루는 개체와 일부 UI위젯이 배치됩니다. 캔버스에 있는 개체는 색깔로 구분되고 상태를 표시합니다. 캔버스는 정의를 내리는 곳의 실제 편집기이고, History 네트워크를 편집합니다. 캔버스는 규정된 객체들과 UI위젯 양쪽 모두를 연결합니다. 캔버스 객체들의 상태는 암호로 코드화된 색상입니다.

F: 캔버스

캔버스는 history 네트워크를 정의하거나 편집하는 실제 편집자입니다. 캔버스는 정의들로 이루어진 객체들과 몇몇 UI widget[장치]들 **G** 모두를 포함하고 있습니다.
캔버스 상의 객체들은 보통 각각의 상태에 대한 피드백을 제공하기 위해 설정된 색깔을 나타내게 됩니다.



A) 매개변수
이 매개변수는 경고가 있는 매개변수로 주황색으로 표시됩니다.
대부분의 매개변수는 데이터 값이 없는 상태로 캔버스에 배치되며, 이것이 경고사항으로 간주되어 주황색으로 표시됩니다.

B) 매개변수
이 매개변수는 경고나 오류가 없는 상태입니다.

C) 컴포넌트
컴포넌트는 입력 매개변수와 출력 매개변수를 갖고 있어서 항상 보다 많은 연결이 이루어지는 객체입니다. 이 컴포넌트는 적어도 1개 이상의 경고를 갖고 있는 상태이며, 객체의 콘텍스트 메뉴에서 경고 또는 오류를 찾아볼 수 있습니다.

D) 컴포넌트
이 컴포넌트는 경고나 오류가 없는 상태입니다.

E) 컴포넌트
이 컴포넌트는 적어도 1개 이상의 오류를 갖고 있습니다. 이는 컴포넌트 자체의 오류이거나 이 컴포넌트에 연결된 입력 또는 출력 매개변수의 오류일 수 있습니다. 컴포넌트 구조에 대해서는 다음 장에서 좀 더 자세히 알아보겠습니다.

선택된 모든 객체들은 투명한 녹색으로 표현됩니다.

G: UI Widgets

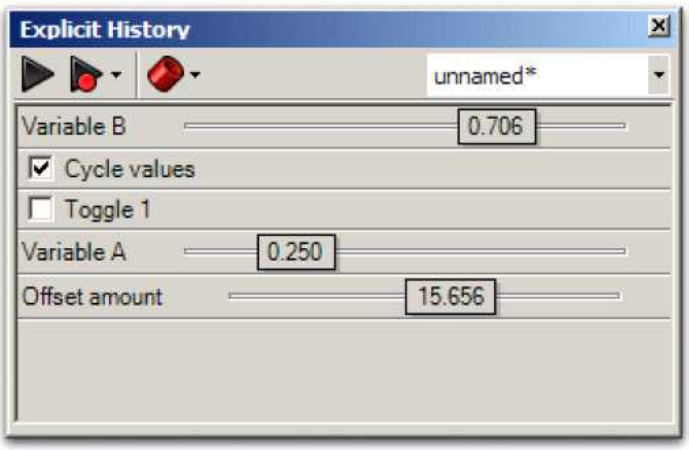
현재 사용가능한 UI widget은 Compass 기능이며, 캔버스의 우측하단에 위치합니다. Compass widget은 작성된 전체 정의 중 현재 작업창에 보여지는 부분의 위치에 대한 그래픽 네비게이션을 제공합니다. 이 widget은 View 메뉴를 통하여 열고 닫을 수 있습니다.

H: 상태바

상태바는 선택된 세트에 대한 피드백을 제공하고 플러그인 상에서 일어나는 주요 이벤트를 보여줍니다. 작성된 정의에 오류나 경고가 있는지에 대한 핵심정보는 여기에 나타납니다.
상태바 좌측 하단에 있는 주황색 사각형 아이콘은 Grasshopper 포럼의 실시간 RSS reader 입니다. 이 아이콘을 클릭하면, Grasshopper 사용자 그룹 웹 싸이트와 연결되어 가장 최근 게시글의 목록이 나타납니다. 이 중 하나를 선택하면 그룹 멤버들이 게시한 글로 바로 연결됩니다. Grasshopper 사용자 그룹의 웹싸이트를 방문해보시기 바랍니다.
http://grasshopper.rhino3d.com/

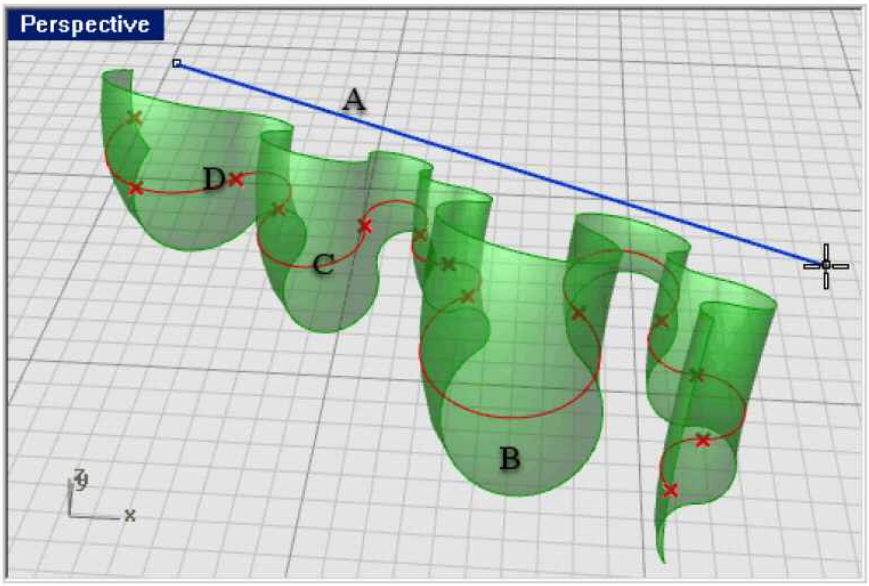
Remote Control Panel[원격 조정 패널]

Grasshopper 창이 상당히 크기 때문에, 항상 창을 띄워놓고 싶지 않을 수도 있습니다. 물론 창을 최소화하거나 닫아버릴 수 있지만, 그렇게 되면 더 이상 설정값을 편집할 수 없게 됩니다. 이때 Remote control panel을 활성화하면, 현재 정의에 포함된 설정값들을 편집하기 위한 최소한의 인터페이스가 나타납니다. 이는 고정된 대화창으로 모든 슬라이더와 boolean 조정의 트랙을 포함하고 있습니다. (그리고 향후 버전에서는 다른 설정값들도 조정할 수 있게 될 것입니다.)



Remote Panel은 또한 기본 미리보기, 이벤트 그리고 파일 전환 조정을 제공합니다. 메인 창의 View 메뉴를 통하여 이 패널을 열거나 닫을 수 있습니다. GrasshopperPanel 명령어를 사용하여 열 수도 있습니다.

뷰포트 미리보기 피드백:



A) **파란색** geometry는 마우스로 현재 선택된 객체입니다.

B) 뷰포트 상 **녹색** geometry는 현재 선택된 컴포넌트에 속해 있는 객체입니다.

C) 뷰포트 상 **빨강색** geometry는 현재 선택되지 않은 컴포넌트에 속한 객체입니다.

D) **Point geometry**는 라이노의 point 객체들과 구분하기 위하여 사각형이 아니라 십자가 형태로 나타납니다.

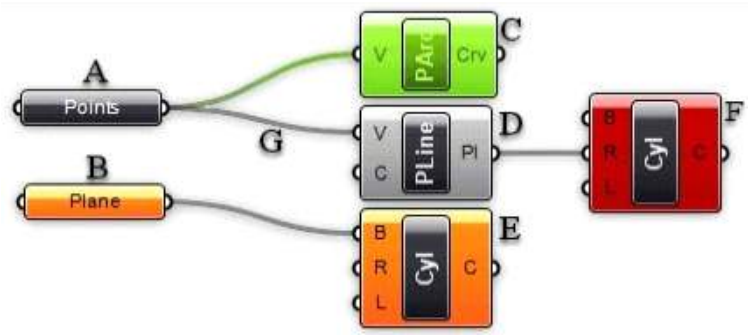
3 *Grasshopper 객체**

Grasshopper 정의 객체

Grasshopper 정의는 많은 종류의 객체들로 구성될 수 있습니다. 그러나 정의 작성을 시작하기 위해서는 다음 두 가지 객체에 익숙해질 필요가 있습니다.

- 매개변수(Parameters)
- 컴포넌트(컴포넌트s)

매개변수는 데이터를 포함하며, **저장**하는 객체입니다. 컴포넌트는 액션을 포함하며 **실행**하는 객체입니다. 다음 그림은 Grasshopper 정의를 사용할 때 마주칠 가능성이 있는 객체들의 사례를 보여줍니다.



A) 이 매개변수는 데이터를 포함하고 있습니다. 객체의 좌측으로부터 나오는 선이 없으므로, 이 객체는 다른 곳으로부터 데이터를 입력받지 않습니다. 오류나 경고가 없는 매개변수는 객체명의 텍스트를 포함한 박스가 검정색으로 나타납니다.

B) 이 매개변수는 아무 데이터도 갖고 있지 않습니다. 데이터를 수집하는데 실패한 어느 객체라도 Explicit History 정의에서 의심스러운 것으로 간주됩니다. 이러한 객체는 모두의 시간과 돈을 낭비하기 때문입니다. 그러므로 모든 매개변수는 (새롭게 추가되었을 경우) 주황색으로 표시되어 아무 데이터도 포함하고 있지 않다는 것을 나타내며, History Solution에 아무 영향을 미치지 못합니다. 매개변수가 데이터를 입력받거나 정의하면 검정색으로 나타납니다.

C) 이 컴포넌트는 선택된 상태입니다. 선택된 모든 객체들은 녹색으로 나타납니다.

D) 정상적인 컴포넌트

* 출처: RhinoWiki
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryPluginObjectsExplained.html

E) 경고를 포함한 컴포넌트. 컴포넌트는 다수의 입력, 출력 매개변수를 포함하고 있습니다. 따라서 단지 컴포넌트를 보는 것만으로는 어떤 객체가 경고를 일으켰는지 알 수 없습니다. 경고의 원인이 둘 이상이 될 수 도 있습니다. 이 때 context 메뉴(아래를 참조)를 활용하여 문제를 찾아갈 수 있습니다. **경고가 반드시 정정되어야할 필요는 없다는 것을 주의하시기 바랍니다.** 경고는 완전히 합법적인 것일 수도 있습니다.

F) 오류를 포함하는 컴포넌트. 경고와 유사하게, 컴포넌트 안에 어떤 오류가 발생했는지 알 수는 없습니다. 이를 확인하려면 Context 메뉴(아래를 참조)를 사용해야 합니다. 경고와 오류를 모두 포함하는 컴포넌트는 빨간색으로 나타나며 오류 표시 색깔이 경고 표시 색깔보다 우선하여 나타납니다.

G) 연결선. 연결선은 입출력 매개변수 사이에 항상 나타납니다. 어느 특정 매개변수에 연결되는 선의 수에는 제한이 없습니다. 그러나 순환되거나 반복적인 연결은 허락하지 않도록 개발되었습니다. 그러한 반복 연결선이 발견되면 전체 솔루션이 합선되고 첫 번째 컴포넌트나 매개변수에 반복연결 오류 메시지로 나타납니다. 연결선에 관한 보다 자세한 설명은 데이터 상속 장을 참조하시기 바랍니다.

컴포넌트 파트

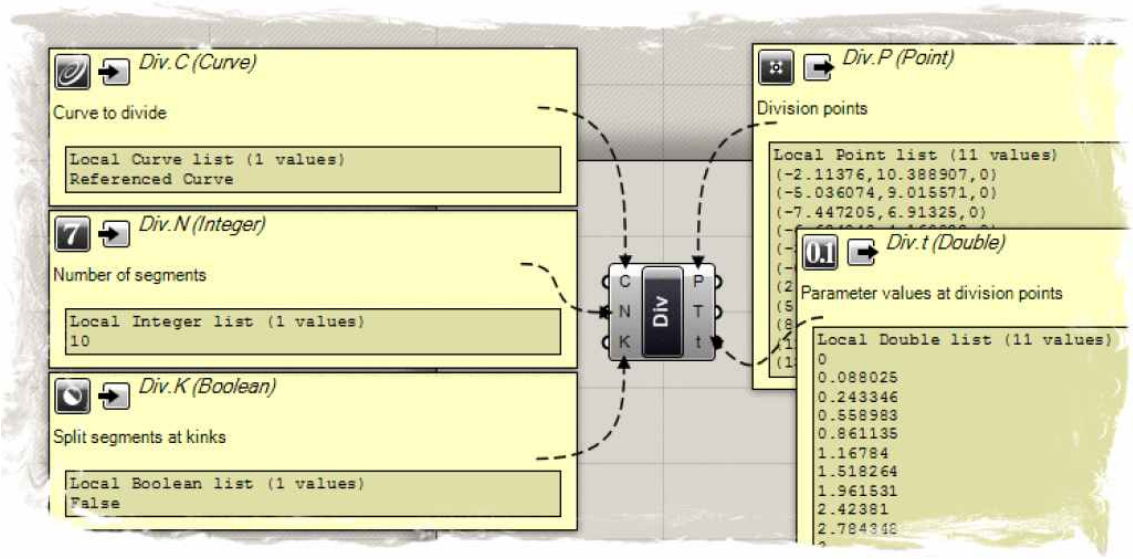
컴포넌트는 일반적으로 액션을 수행하기 위하여 데이터를 필요로 하며, 데이터는 보통 결과값으로 나타납니다. 이는 대부분의 컴포넌트가 내포된 입력과 출력의 매개변수의 셋트를 각각 갖고 있는 이유입니다. 입력 매개변수는 좌측에, 출력 매개변수는 우측에 위치합니다.

A) Division 컴포넌트의 3개의 입력 매개변수입니다. 기본값으로 매개변수 명칭은 항상 축약형으로 나타납니다. 필요에 따라 각 매개변수의 명칭은 자유롭게 변경할 수 있습니다.

B) Division 컴포넌트 영역 (일반적으로 컴포넌트의 명칭을 포함합니다)

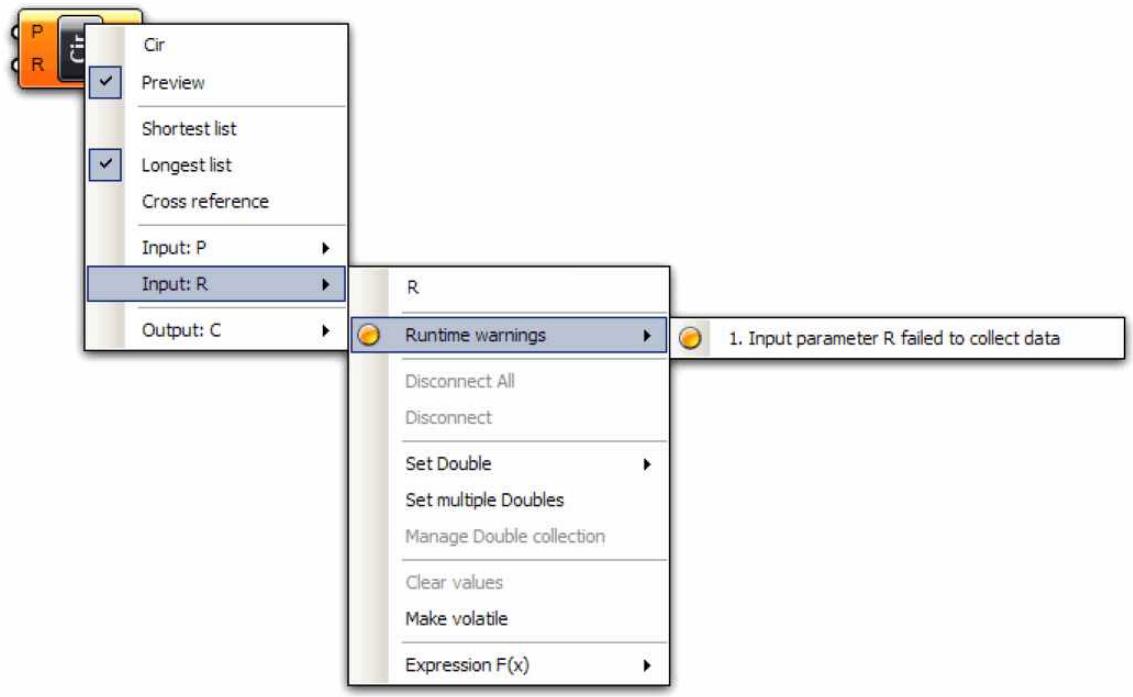
C) Division 컴포넌트에는 3개의 출력 매개변수를 갖고 있습니다.

마우스를 컴포넌트 객체 위의 개별 파트 위에 올려놓으면, 현재 마우스가 올려진 파트에 대한 자세한 툴팁을 볼 수 있습니다. 툴팁은 개별 매개변수에 대한 유형과 데이터 모두에 대한 매우 유용한 정보를 제공합니다.



Context 팝업 메뉴 사용하기

캔버스의 상의 모든 객체들은 각각의 context 메뉴를 갖고 있으며 각 컴포넌트의 특징의 대부분을 포함하고 있습니다. 컴포넌트는 모든 하위메뉴를 단계적으로 노출하도록 설계되어 이용하기 약간 까다로울 수 있습니다. 예를 들어, 컴포넌트 또는 컴포넌트와 연결된 특정 매개변수가 주황색으로 표시된다면 경고를 의미합니다. 무엇이 잘못되었는지 확인하고 싶다면 컴포넌트 context 메뉴를 사용하면 됩니다.



위의 그림에서 메인 컴포넌트 메뉴가 입력 데이터인 R값에 대하여 단계적으로 펼쳐진 모습을 볼 수 있습니다. 메뉴는 보통 해당 메뉴의 명칭을 보여주는 편집 가능한 텍스트 필드로

시작됩니다. 메뉴명칭은 보다 더 자세하게 변경할 수 있지만, 모든 명칭의 기본값은 화면상 차지하는 자리를 최소화하기 위해 최대한 짧게 작성되어 있습니다. 메뉴의 두 번째 아이템 (Preview flag)은 이 객체에 의해 생산되거나 정의된 geometry가 라이노 뷰포트 상에 가시화될지 아닐지를 표시합니다. 필수정보를 포함하지 않는 컴포넌트의 미리보기를 꺼 놓으면 라이노 뷰포트 움직임의 속도를 빠르게 하고 (메쉬가 포함되어 있을 경우) History 솔루션에 소요되는 시간을 줄일 수 있습니다. 매개변수나 컴포넌트의 미리보기를 비활성화하면 엷은 흰색 해치가 뷰포트 상에 나타납니다. 모든 매개변수/컴포넌트가 뷰포트 상에 나타나는 것은 아닙니다. 예를 들면, 숫자와 같은 객체의 경우 미리보기 아이템에서 보통 생략됩니다.

"R" 입력 매개변수를 위한 context 메뉴는 주황색 경고 아이콘을 포함합니다. 이는 차례로 이 매개변수에 의해 생성된 모든 경고 목록을 보여줍니다. 위의 그림은 1개의 경고가 표시된 경우입니다.

4 *Persistent 데이터 관리*

데이터 유형

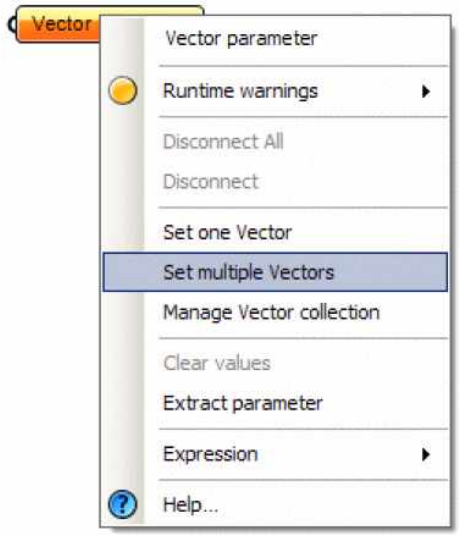
매개변수는 단지 정보를 저장하는데 사용되어 왔습니다. 그러나 대부분의 매개변수는 두 가지 유형의 데이터를 저장할 수 있습니다. 이는 Volatile 데이터와 Persistent 데이터입니다. Volatile 데이터는 하나 또는 그 이상의 소스 매개변수로부터 전달받고 새로운 솔루션이 시작될 때마다 소멸됩니다 (즉, 다시 수집됩니다).

(기록을 저장하거나 소스의 세트를 정의하지 않는 출력 매개변수는 예외이며, 출력 매개변수는 매개변수가 속한 해당 컴포넌트에서 완전히 제어합니다.)

Persistent 데이터는 메뉴를 통해 접근할 수 있으며, 매개변수의 종류에 따라 관리자가 달라집니다. 예를 들면 Vector parameter는 메뉴를 통해 단일 또는 복수의 vector를 설정하는 것을 허용합니다.

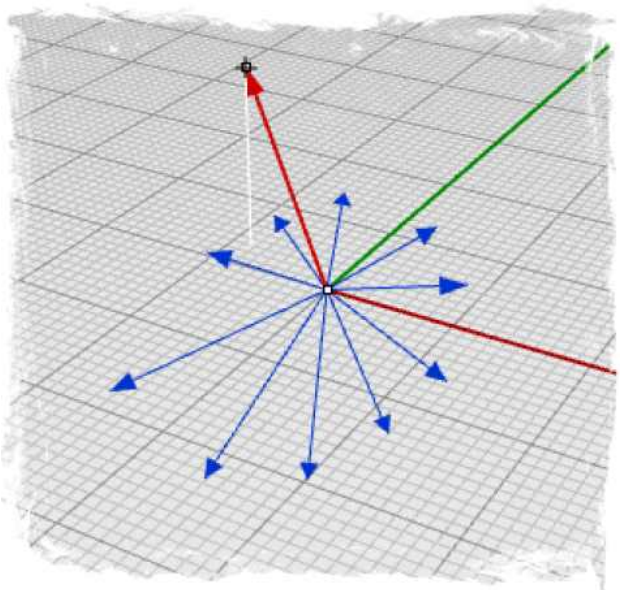
하지만, 몇 단계를 거슬러 올라가서 기본 Vector parameter가 어떻게 기능하는지 살펴보겠습니다. Params Panel로부터 Vector parameter를 드래그하여 캔버스 상에 올려놓으면 다음과 같이 나타납니다.

매개변수는 주황색으로 나타나며, 경고가 발생했음을 의미합니다. 심각할 것도 없이 이 경고는 단지 매개변수가 비어있음을 알려주는 것입니다. (매개변수는 Persistent 기록을 포함하지 않으며 volatile 데이터를 수집하는데 실패한 상태입니다.) 그리고 History 솔루션의 결과물에 아무런 영향을 미치지 않습니다. 매개변수의 context 메뉴는 Persistent 데이터를 설정하는 2가지 방법을 제공합니다. 단일 Vector와 복수 Vector를 설정할 수 있습니다.



* 출처: RhinoWiki
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryPersistentDataRecordManagement.html

이 메뉴 아이템 중 하나를 클릭하면, Grasshopper 창은 사라지고 라이노 뷰포트에서 하나의 Vector를 선택할 수 있습니다.



필요한 모든 Vector를 정의하고 엔터를 누르면, 이 Vector들은 매개변수 Persistent 데이터 기록의 일부가 됩니다. 이는 그 매개변수가 더 이상 비어있지 않음을 의미하며, 매개변수의 색은 주황색에서 검정색으로 바뀝니다.

이 시점부터 이 매개변수를 여러분이 원하는 만큼 많은 수의 객체를 생성하는 'Seed[종자]'로 사용할 수 있습니다. 이 객체들은 독자적인 vector들을 포함합니다.

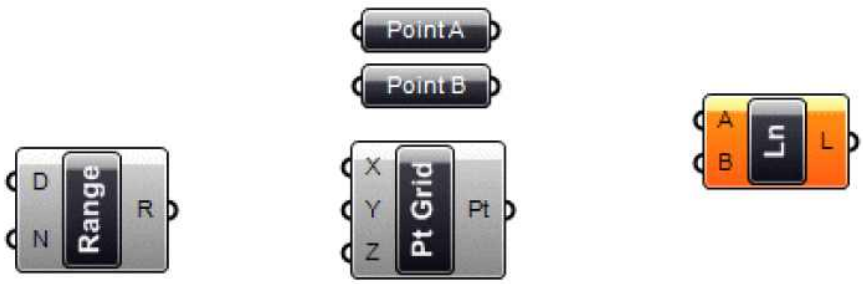
5 *Volatile 데이터의 상속성(Inheritance)**

데이터 상속성

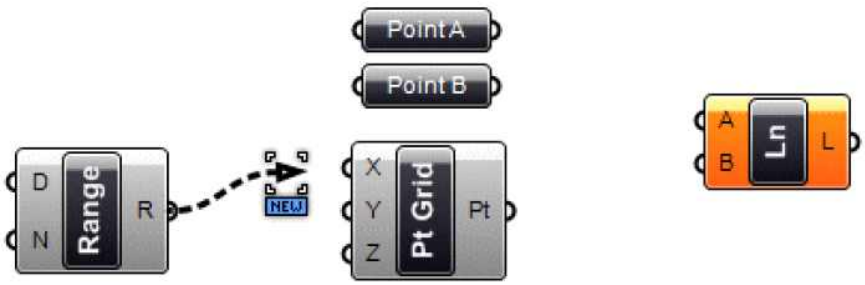
데이터는 매개변수 안에 (Volatile 또는 Persistent 형식으로) 저장되고 컴포넌트에 사용됩니다. 데이터가 매개변수의 영구기록세트에 저장되어 있지 않다면 그것은 다른 곳으로부터 상속된 것입니다. 출력 매개변수를 제외한 모든 매개변수는 어디서 데이터를 얻을 지를 정의하며, 대부분의 매개변수는 특별히 다르지 않습니다. 두 개의 매개변수를 (이는 단지 어떤 십진수의 한 숫자를 의미합니다) 정수[interger] 소스로 연결할 수 있으며, 변환도 이루어집니다. Grasshopper 플러그인은 많은 변환 스키마를 정의하지만 해석과정이 정의되어 있지 않으면, 받는 쪽의 매개변수에서 변환 오류가 생성됩니다. 예를 들어, Point 데이터가 필요할 때, Surface 데이터를 입력하면 Point 매개변수에서 오류 메시지가 생성되고 빨간색으로 표시됩니다. (오류 메시지는 해당 매개젼수의 메뉴로부터 접근할 수 있습니다.) 매개변수가 컴포넌트에 속하는 경우, 빨간색의 상태는 계층 구조의 위쪽으로 전파되며, 해당 컴포넌트도 자체에 오류가 없더라도 빨간색으로 바뀝니다.

연결선 관리

매개변수는 각각 자신의 데이터 소스를 담당하기 때문에, 해당 매개변수를 통해 이러한 설정에 접근할 수 있습니다. 세 개의 컴포넌트와 두 개의 매개변수를 포함하는 아래의 정의를 살펴봅시다.

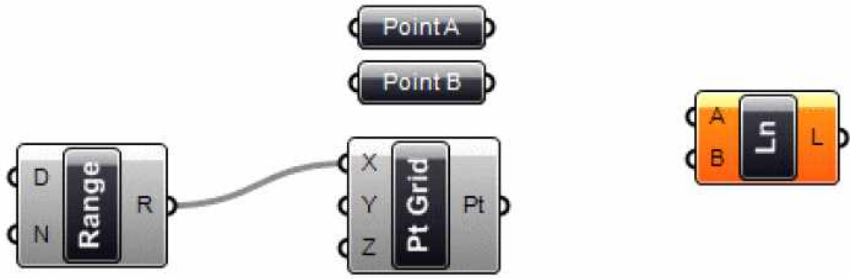


이 단계에서 모든 객체는 연결되어 있지 않은 상태이며, 직접 연결할 필요가 있습니다. 연결하는 순서는 중요하지 않지만, 왼쪽에서 오른쪽으로 연결하기로 합니다. 매개변수의 작은 원 (그립) 가까이를 마우스로 끌기 시작하면 와이어가 마우스에 붙어 두 개의 개체를 연결합니다.

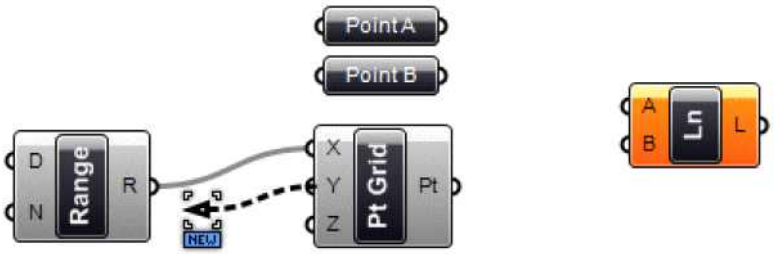


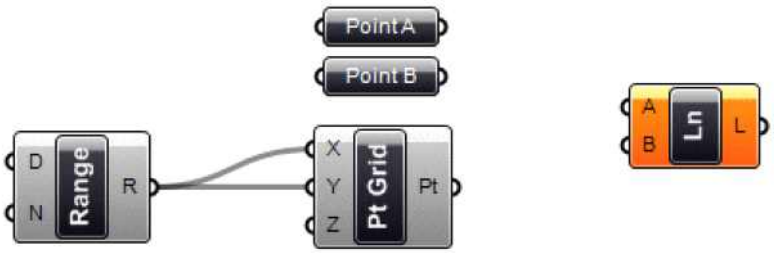
* 출처: RhinoWiki
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryVolatileDataInheritance.html

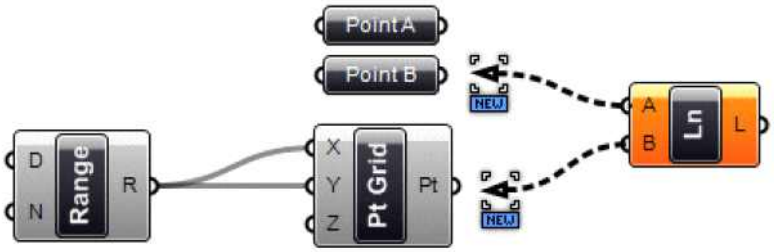
왼쪽 마우스를 클릭한 상태에서 대상 매개변수로 마우스를 끌면 와이어가 연결되어 실선이 됩니다. 왼쪽 마우스를 떼기 전까지는 완전히 연결된 것이 아닙니다.

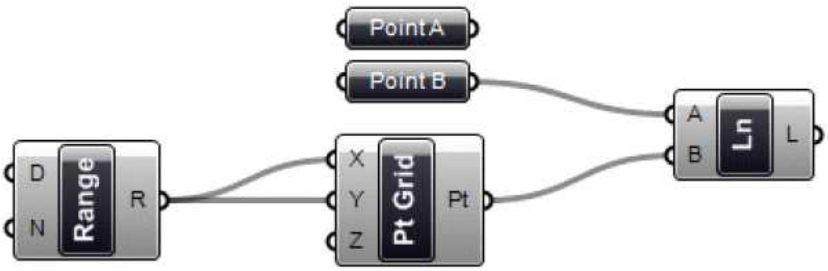


PtGrid 컴포넌트의 매개변수 "Y"와 Line(Ln) 컴포넌트의 매개변수 "A"와 "B"를 같은 방식으로 연결해 봅니다. 클릭+드래그+릴리즈...

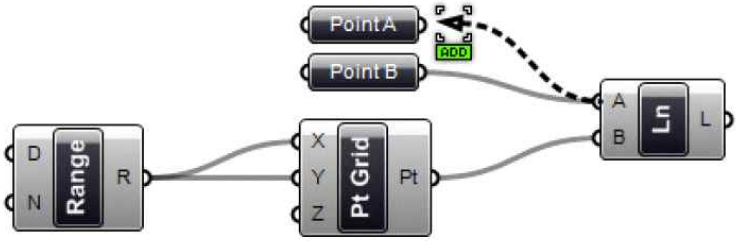


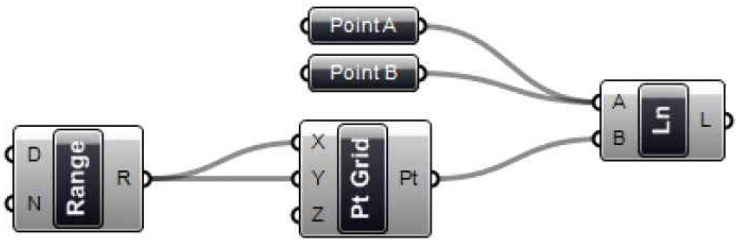






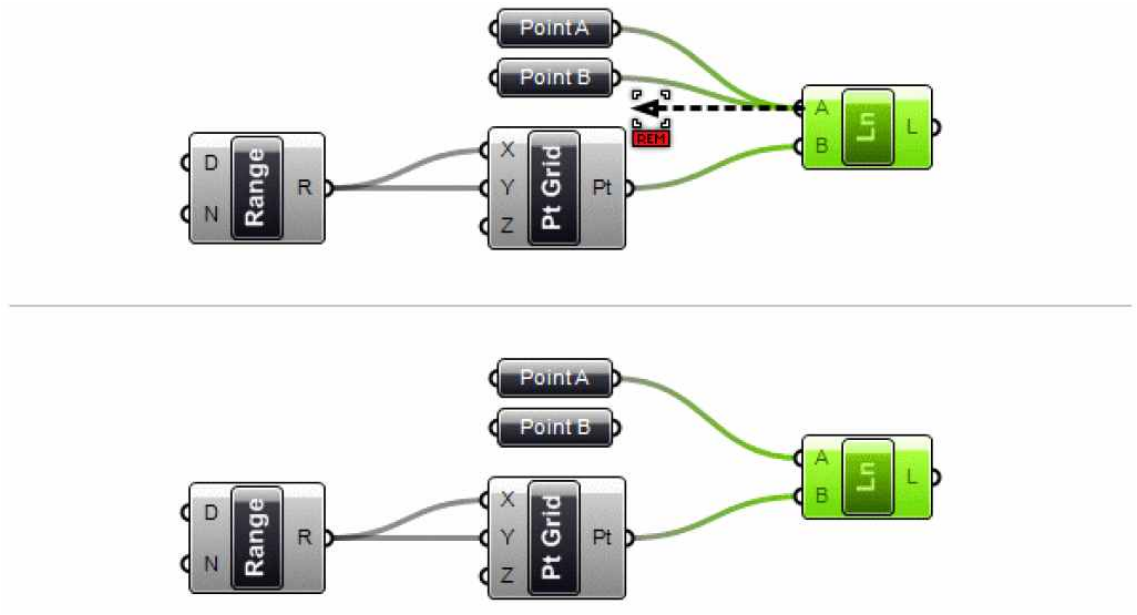
매개변수는 양쪽 방향으로 연결할 수 있고, 기본 설정에 의해 새로운 연결이 기존 연결을 지우도록 되어 있으므로 주의해야 합니다. 단일 연결선을 가장 많이 사용할 것을 가정하고 있으므로 복수의 소스를 정의하기 위해서는 추가적인 단계가 필요합니다. 연결선을 드래그하는 동안 Shift키를 유지한다면, 마우스 포인터는 추가적인 행위를 지시하기 위해 변경될 것입니다.



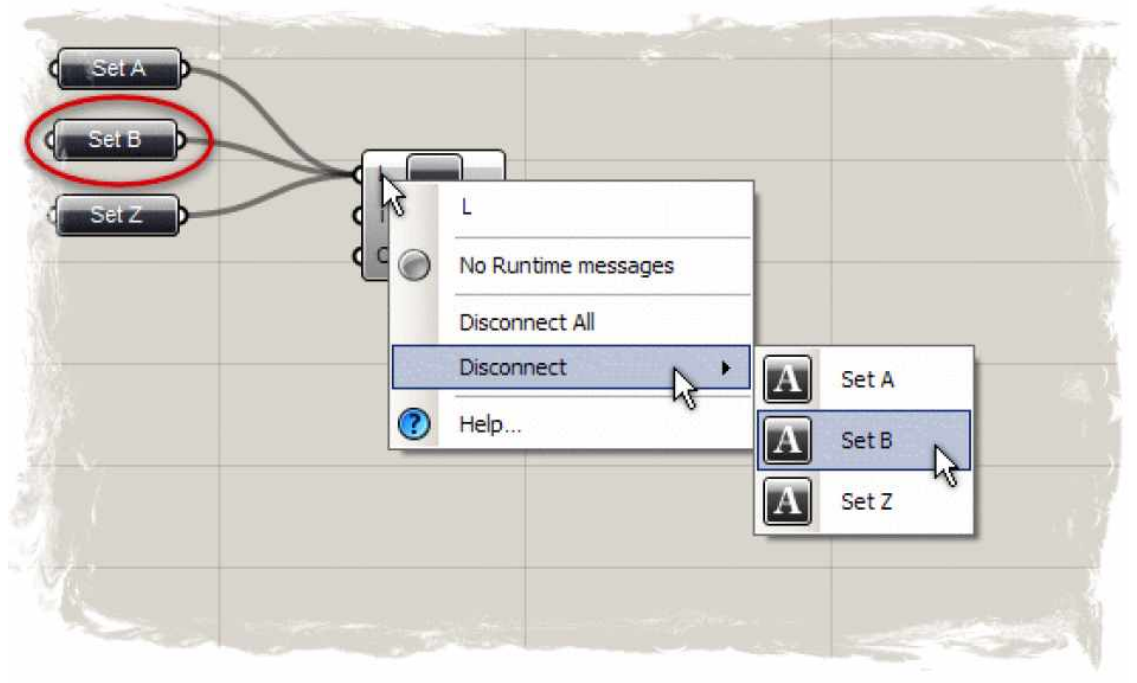


마우스 버튼을 소스 매개변수 위에서 풀 때 "ADD" 커서를 활성화하면, 그 매개변수는 소스 목록에 추가될 것입니다. 만약 소스로 이미 정의되어있는 소스 매개변수를 명시한다면, 아무것도 일어나지 않을 것입니다. 같은 소스를 한번 이상 상속받을 수는 없습니다.

같은 이유로, Control the "REM" 커서를 유지하면 가시화되고 타겟소스는 소스 목록에서 제거될 것입니다. 타겟이 참조되지 않으면 아무 일도 일어나지 않습니다.

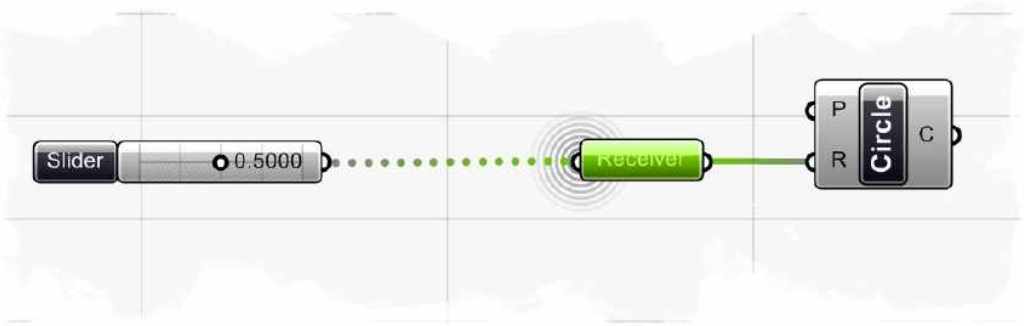


또한, 매개변수 메뉴에서 소스의 연결을 해제할 수 있습니다. (연결은 되지 않습니다.)

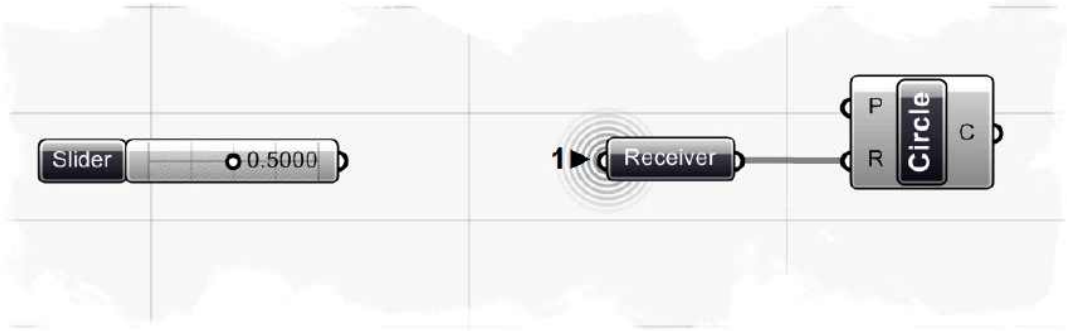


Grasshopper는 Params 탭의 특별한 하위 카테고리에 포함된 Receiver를 사용하여 무선으로 정보를 전달할 수 있습니다. 다른 컴포넌트들을 연결하는 것과 같은 방식으로 Receiver와 객체를 연결할 수 있습니다. 그러나 연결선으로부터 마우스 왼쪽 버튼을 떼는 즉시 연결선은 자동적으로 사라집니다. Receiver는 선택되었을 때에만 점선으로 연결선을 나타내도록 기본 설정 되어있습니다. Receiver를 오른쪽 클릭하면 연결선을 "selected" 일 때 보기, 또는

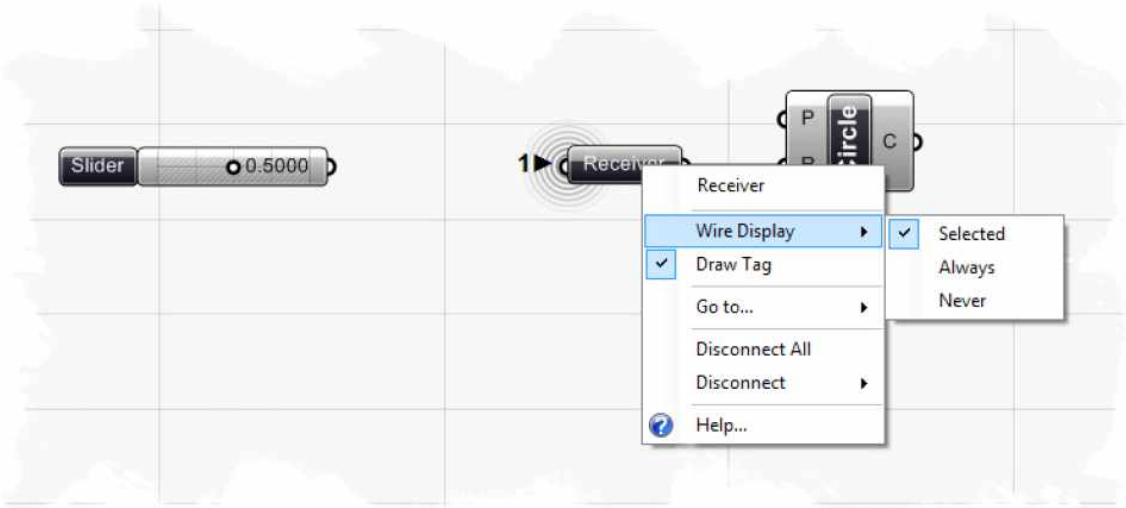
"always"또는 "never" 보기를 설정할 수 있습니다. Receiver의 출력데이터는 필요한 만큼의 많은 다른 컴포넌트에 연결할 수 있습니다.



여기에서 점선으로 된 연결선을 볼 수 있는데, 현재 Receiver가 선택된 상태이기 때문입니다.



Receiver 컴포넌트의 입력부분 앞에 있는 숫자 1은 하나의 연결선이 입력값으로 수신되고 있음을 나타냅니다. 그러나 Receiver 컴포넌트가 선택되어 있지 않기 때문에 연결선은 더 이상 보이지 않습니다. (하지만 정보는 여전히 전송되고 있는 상태입니다.)

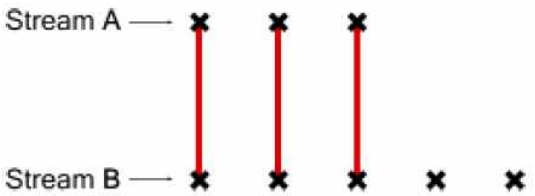


6 데이터 스트림 매칭*

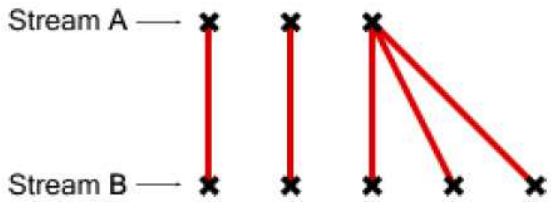
데이터 매칭

데이터 매칭은 해결하기 까다로운 문제입니다. 데이터 매칭은 컴포넌트에 다양한 값을 입력할 때 발생합니다. 두 점을 잇는 선을 컴포넌트라고 가정할 때, 이 선은 두 개의 포인트 좌표값을 입력 매개변수로 갖습니다. (Stream A와 Stream B) 이 매개변수들이 어디로부터 데이터를 수집하는지는 관련이 없습니다. 컴포넌트는 입력 및 출력 매개변수를 넘어서서 "볼" 수가 없습니다.

위의 그림과 같이 이 점들의 세트 사이를 연결하는 다양한 방법이 존재합니다. Grasshopper 플러그인은 현재 세 가지 매칭 알고리즘을 지원하지만 더 많은 방법도 가능합니다. 가장 간단한 방법은 각 스트림의 점 데이터를 가능한 범위까지 일대일 대응하는 것입니다. 이를 "Shortest List 알고리즘"이라고 합니다.

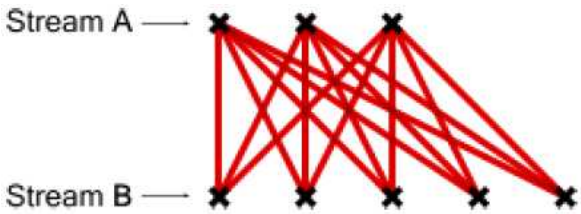


"Longest List 알고리즘"은 모든 스트림이 연결될 때까지 계속 연결합니다. 이는 컴포넌트의 기본설정 동작입니다.



마지막으로, "Cross Reference" 방법은 가능한 모든 연결을 수행합니다.

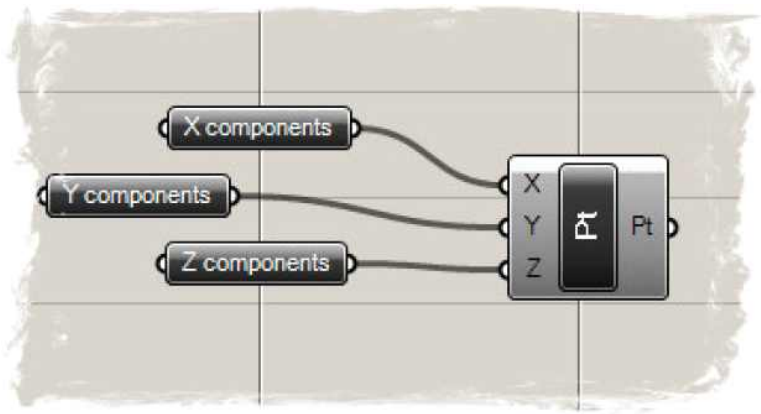
* 출처: RhinoWiki
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryDataStreamMatchingAlgorithms.html



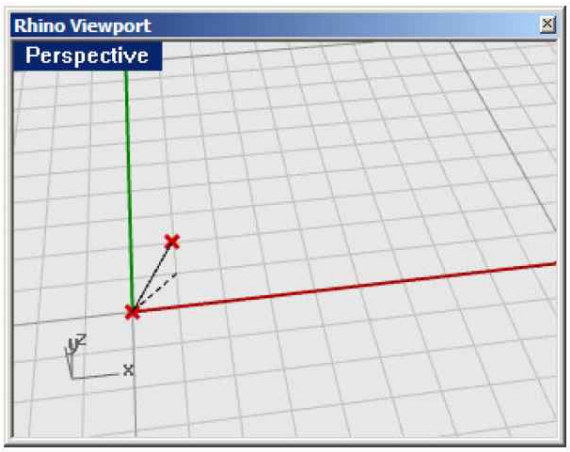
이 방식은 출력된 결과의 수가 엄청날 수 있기 때문에 잠재적인 위험성이 있습니다. 더 많은 입력 매개변수가 포함되고 Volatile 데이터 상속이 데이터를 증가시키면 문제는 보다 복잡해집니다. 그러나 논리는 여전히 같습니다.

원격 매개변수에서 다음의 데이터를 x, y, z 값으로 상속받는 Point(Pt) 컴포넌트가 있다고 가정해봅시다.

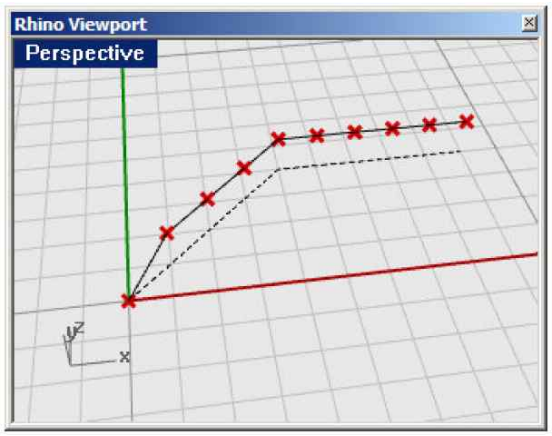
X 좌표: {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0}
Y 좌표: {0.0, 1.0, 2.0, 3.0, 4.0}
Z 좌표: {0.0, 1.0}



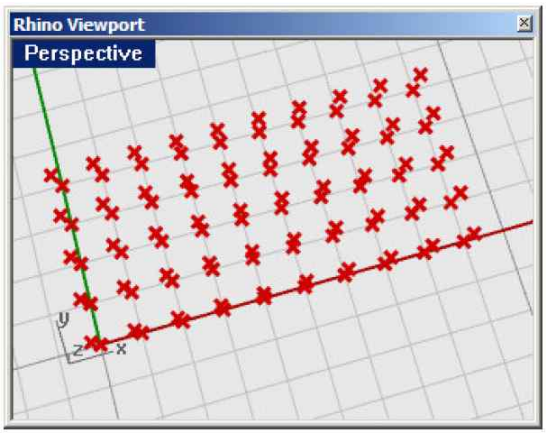
이 데이터를 "Shortest List" 방식으로 결합하면, "Z 좌표"가 단지 두 개의 값을 포함하기 때문에 두 개의 점만을 얻을 수 있습니다. 이것이 shortest list이기 때문에 솔루션의 범위를 다음과 같이 결정합니다.



"Longest List" 알고리즘은 Y와 Z 스트림에 있을 수 있는 가장 높은 값을 재활용하여 열 개의 점을 생성합니다.



"Cross Reference" 방식은 X의 모든 값을 Y와 Z의 모든 값을 연결합니다. 결과적으로 10×5×2=100 개의 점이 생성됩니다.



모든 컴포넌트는 이 규칙 중 한 가지를 만족하도록 설정될 수 있습니다. (컴포넌트 아이콘을 오른쪽 클릭하면 열리는 메뉴를 통해 설정할 수 있습니다.

이 기능에는 큰 예외가 있음을 유의하시기 바랍니다. 특정 컴포넌트는 입력 필드의 하나 또는 그 이상의 데이터 목록을 갖게 됩니다. 예를 들어, Polyline 컴포넌트는 입력 점들의 배열에 의해 Polyline curve를 생성합니다. 입력 매개변수는 List 매개변수라 불리는 하나 이상의 값에 배정됩니다. 데이터 매칭이 이루어질 때 List 매개변수는 무시됩니다.

7 Scalar 컴포넌트 유형

Scalar 컴포넌트 유형들은 일반적으로 여러 가지 수학적인 작용을 위하여 사용되며, 다음 요소들로 구성되어 있습니다.

A) **Constants**[상수]

Pi, 황금비 등과 같은 상수 값을 되돌려줍니다.

C) **Intervals**[구간]

숫자의 범위(양극단 또는 영역)를 나누어 Interval 파트로 분할하는데 사용합니다. Interval 탭에는 많은 컴포넌트들이 있습니다. 이는 다양한 interval 유형을 생성하거나 재구성하는 것을 허락합니다.

D) **Operators**[연산자]

덧셈, 뺄셈, 곱셈 등과 같은 수학적인 연산자들이 사용됩니다.

E) **Polynomials**[다항식]

어떤 거듭 제곱에 의해 수치를 끌어 올리는데 사용합니다.

E) **Trigonometry**[삼각법]

사인, 코사인, 접선 등과 같은 전형적인 삼각법 값을 되돌려줍니다.

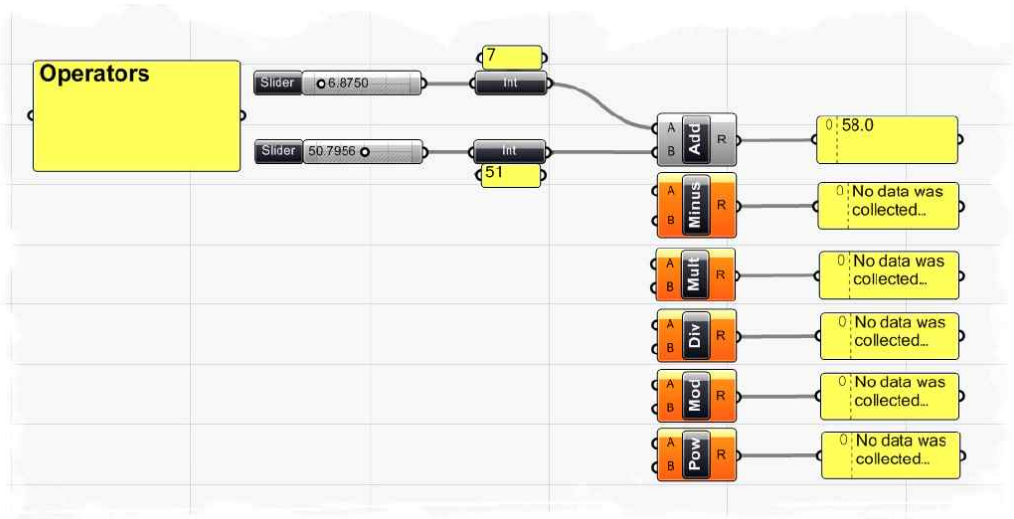
F) **Utility(Analysis)**[분석(유틸리티)]

둘 이상의 많은 수치들의 값을 평가하는데 사용됩니다.

7.1 Operators

앞서 언급했듯이 Operators는 컴포넌트의 집합입니다. 컴포넌트들은 두 개의 수치 입력값을 가진 대수함수를 이용하여 하나의 출력값을 도출합니다. Operators를 보다 자세히 이해하기 위해 다른 Operator 컴포넌트 유형을 탐험하며 간단한 수학정의를 만들 것입니다.

주의: 이 정의의 최종 버전을 보려면, 이 문서와 함께 제공되는 소스파일폴더의 **Scalar_Operators.ghx** 파일을 엽니다. 아래 그림은 완성된 정의를 보여줍니다.



처음부터 정의 생성하기

- Params/Special/Numeric Slider 메뉴에서 Numeric Slider를 드래그하여 캔버스에 올려놓습니다.
- Slider를 더블클릭하고 다음과 같이 설정합니다.
  - 최소값: 0.0
  - 최대값: 100.0
  - 결과값: 50.0 (주의: 이 값은 임의의 값이며, 상·하한값 사이에서 어떤 값으로도 수정할 수 있습니다.)
- Slider를 선택하고 Ctrl+C(복사하기)와 Ctrl+V(붙여넣기)를 사용하여 Slider를 복제합니다.
- Params/Special/Numeric Slider 메뉴에서 2개의 Integer 컴포넌트를 캔버스 위로 드래그 앤 드롭합니다.
- Slider 1과 첫 번째 Integer 컴포넌트를 연결합니다.
- Slider 2와 두 번째 Integer 컴포넌트를 연결합니다.

  Slider의 기본값 유형은 Floating Point[부동 소수점]으로 설정되어 있습니다.(십진수 값으로 어떤 결과를 표현합니다.) Integer 컴포넌트에서 Slider를 연결하면, Floating Point를 Integer 혹은 어떤 전체적 수치로 변환할 수 있습니다. Params/ Special/ Panel 메뉴에서 Post-It panel을 꺼내 각 Integer 컴포넌트의 출력 값으로 연결하면, 실시간으로 변환되는 것을 볼 수 있습니다. Slider를 왼쪽과 오른쪽으로 움직이면, Floating Point가 전체 숫자로 변환되는 것을 볼 수 있습니다. 물론 이 Slider 유형을 Integer로 설정하면 이 단계를 간단하게 처리할 수 있습니다.
- Scalar/Operator/Add 메뉴에서 Add 컴포넌트를 드래그 앤 드롭하여 캔버스에 추가합니다.
- Add 컴포넌트 입력-A에 첫 번째 Integer 컴포넌트를 연결합니다.
- Add 컴포넌트 입력-B에 두 번째 Integer 컴포넌트를 연결합니다.
- Params/ Special/ Panel 메뉴의 Post-It panel을 드래그 앤 드롭하여 캔버스에 추가합니다.
- Add 출력-R을 Post-It panel 입력에 연결합니다.

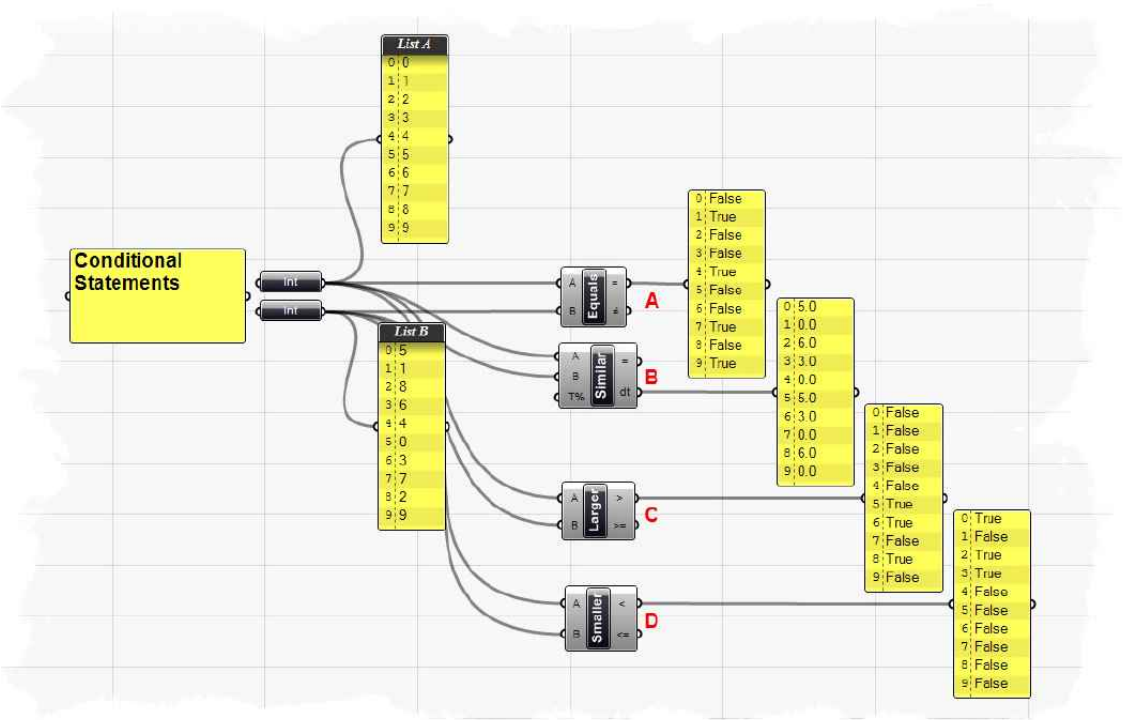
  Post-It panel에서 두 정수 값의 합을 볼 수 있습니다.
- Scalar 메뉴에 속한 다른 Operators 컴포넌트를 캔버스 위로 드래그 앤 드롭 합니다.
  - 뺄셈
  - 곱셈
  - 나눗셈
  - 계수, 율, (복소수의) 절대 값
  - 거듭 제곱, 집합 수
- 첫 번째 Integer 컴포넌트를 각각의 Operator 입력단자 A에 연결합니다.
- 두 번째 Integer 컴포넌트를 각각의 Operator 입력단자 B에 연결합니다.
- 캔버스 위에 다섯 개의 Post-It panel을 드래그 앤 드롭 하고 각 Operator의 출력단자에 연결합니다.

이 정의는 완성되었습니다. 이제 각 Slider의 값들을 바꾸면, Post-It panel 상에서 각 Operator가 적용된 결과를 볼 수 있습니다.

7.2 Conditional Statesments[조건 명령문]

Scalar 탭의 Operators 컴포넌트 중 몇몇 컴포넌트들은 앞장에서 설명되지 않았습니다. 이는 0.6.0007 버전에 새롭게 추가된 4개의 컴포넌트가 수학적인 연산자들과는 약간 다르게 작용하기 때문입니다. 이 컴포넌트들은 대수식 계산을 수행하는 것이 아니라 두 개의 데이터 목록을 비교합니다. 4개의 컴포넌트는 **Equality(등식), Similarity(닮음), Larger Than(보다 크고), Smaller Than(보다 작고)**의 연산을 수행합니다. 보다 자세한 기능은 아래에서 설명하겠습니다.

주의: 이 정의의 최종 버전을 보려면, **Conditional Statements.ghx** 파일을 이 문서에 첨부된 소스 파일에서 찾아서 열면 됩니다. 아래는 완성된 정의의 화면 캡처 이미지입니다.



A) Equality

Equality 컴포넌트는 두 개의 목록을 취하여, 목록 A의 첫 번째 아이템을 목록 B의 첫 번째 아이템과 비교합니다. 만약 두 값이 같다면 True Boolean 값이 생성됩니다. 반대로 두 값이 같지 않다면 False Boolean 값이 생성됩니다. 컴포넌트는 데이터 매칭 알고리즘에 설정된 List를 통해 순환합니다. (기본 값은 Longest List로 설정되어 있습니다.) 이 컴포넌트는 두 개의 출력단자를 갖습니다. 첫 번째 출력단자는 목록의 값들이 서로 같음을 보여주는 boolean 값의 목록을 보여줍니다. 두 번째 출력단자는 값이 서로 같지 않음을 보여주는 목록 또는 첫 번째 출력단자로부터 반전된 목록으로 나타납니다.

B) Similarity

Similarity 컴포넌트는 데이터의 두 가지 목록을 평가하고 두 숫자 간의 similarity를 검토합니다. 이는 Equality 컴포넌트가 두 가지 목록을 비교하는 방식과 거의 동일합니다. 그러나 한 가지 예외가 있는데, 목록 A의 비율을 정의하는 백분율 입력단자가 있습니다. inequality가 가정되기 전까지의 편차를 허용하는

...in that it has a percentage input that defines the ratio of list A that list B is allowed to deviate before inequality is assumed.
Similarity 컴포넌트는 또한 두 입력 목록 사이의 절대값을 결정하는 출력단자를 가집니다.

C) Larger than

컴포넌트는 두 개의 데이터 목록을 가질 것이고, 목록 A의 그 첫 번째 목록 B의 그 첫 번째 항목 보다 더 큰지를 결정할 것입니다. 만약 두 개의 목록을 보다 큰 것 또는 보다 크거나 같은 조건을 따르도록 그 두 개의 목록을 평가하고 싶다면, 그 두 개의 출력을 결정할 수 있게 합니다.

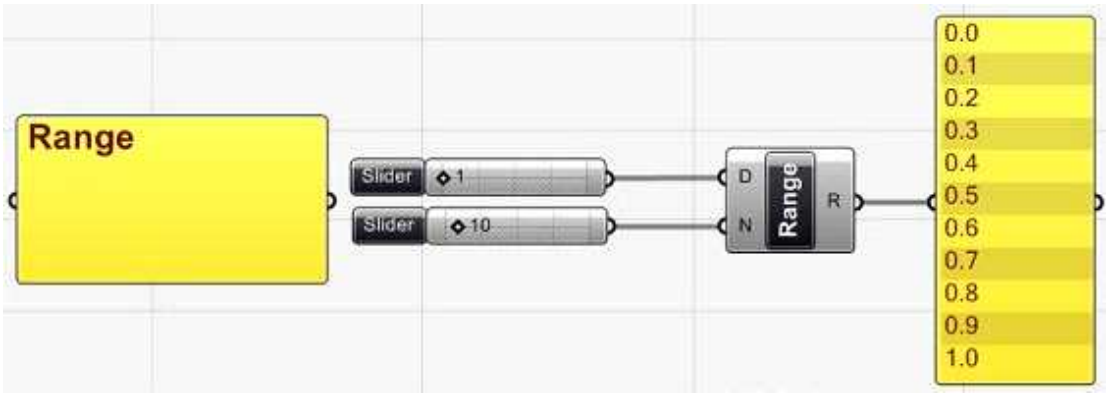
D) Smaller Than

컴포넌트는 보다 큰 컴포넌트 작동과 정반대로 실행합니다. 보다 작은 컴포넌트는 목록 A가 목록 B보다 적다면, 논리 값의 목록을 반환하도록 결정합니다. 마찬가지로, 만약 그 두 개의 목록을 보다 작거나 같은 조건을 따르도록 그 두 개의 목록을 평가하고 싶다면, 그 두 개의 출력을 결정할 수가 있습니다.

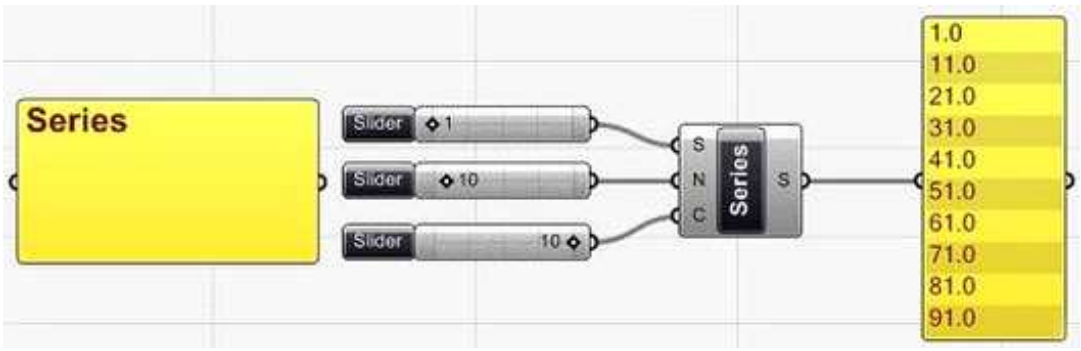
7.2 Range vs Series[일련화] vs Interval

Series 그리고 Interval 컴포넌트들은 두 개의 숫자 맨 앞, 맨 끝의 사이에 값들을 배치하도록 만듭니다. 하지만 컴포넌트들은 서로 다른 방식으로 작용합니다.

Range 컴포넌트는 숫자 범위의 도메인이라 불리는 낮은 값과 높은 값 사이에 고르게 간격을 두게 되는 수들의 목록을 생성합니다. 위의 예제에서, 두 개의 numeric Sliders는 Range 컴포넌트의 입력 값으로 연결되었습니다. 그 첫 번째 Slider는 값의 범위를 위해 도메인 수를 정의합니다. 이번 예제에서 그 Slider는 1로 설정되어 있어서 그 도메인은 0에서 1로 정의 되었습니다. 그 두 번째 Slider는 이번 예제에서 10으로 설정 되어있는 그 도메인을 나누기 위해 단계의 그 수를 정의합니다. 그러므로 그 출력은 0과 1 사이에 고르게 분할된 11개의 수 목록입니다.

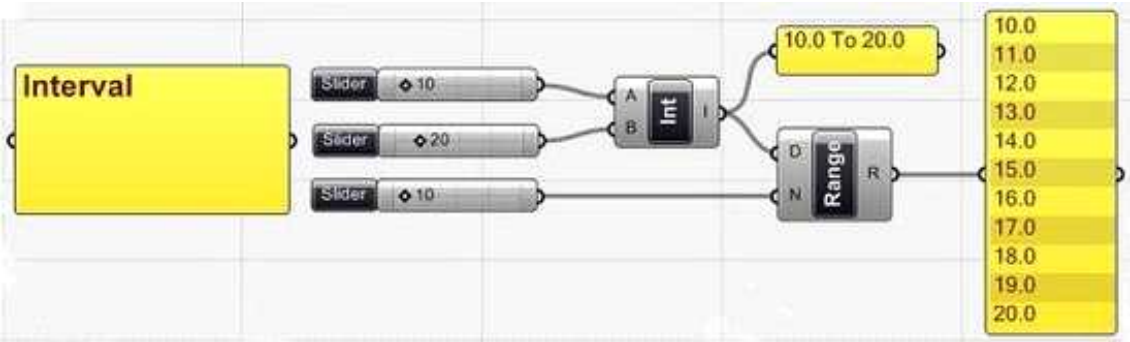


Series 컴포넌트는 Series에서 시작 값, 단계 크기 그리고 값에 기초하는 수들을 설정합니다. 예제에서 컴포넌트들에 연결된 세 개의 numeric Sliders를 나타내고 있습니다. 첫 번째 Slider는 그 연속적 수에 있어서 시작점을 정의하는 Series-S 입력에 연결되어졌을 때, 그 두 번째 Slider가 10으로 그 시리즈를 위한 그 단계 값을 정의합니다. 그 시작 값은 1로 설정되었고, 그 단계 크기는 10으로 설정되었으므로, 다음 값은 11이 될 것입니다. 끝으로 세 번째 Slider는 값이 10으로 설정되었으므로, 정의된 최종 출력 값은 1에서 시작하여 각각 단계가 10으로 증가하는 10개의 수가 보여 집니다.



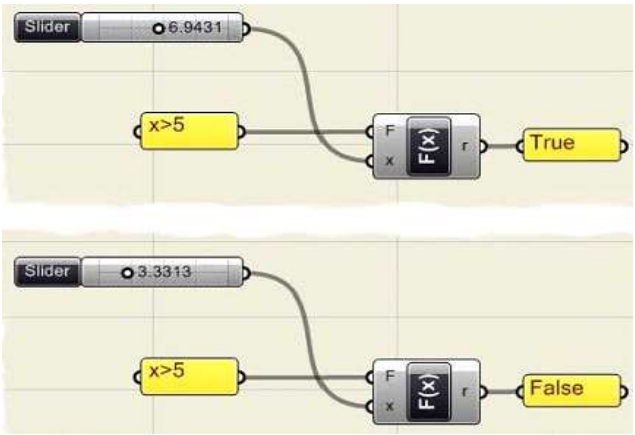
Interval 컴포넌트는 낮고 높은 수 사이에 모든 가능한 수들의 범위를 만듭니다. Interval 컴

포넌트는 Range 컴포넌트를 위해 우리가 정의한 도메인 수와 유사합니다. 주된 차이는 Range 컴포넌트는 0과 어떤 입력 값으로 정의된 것 사이에서 기본 값의 도메인 수를 만듭니다. Interval 컴포넌트는 A와 B 입력 값들에 의해 낮고, 높은 값이 정의될 수 있습니다. 우리는 두 개의 Slider로 10과 20 사이에 모든 가능한 값들의 범위를 정의했습니다. Interval 컴포넌트를 위한 그 출력 값은 우리의 새로운 도메인 수를 반영하여 10에서 20으로 나타납니다. 만약 우리가 지금 Interval -I 출력을 Range-D 입력에 연결하면, 우리는 Interval 값 사이에 수의 범위를 만들 수 있습니다. 이전에 Range 예제 경우처럼, 우리가 그 Range를 위한 단계의 수를 10으로 설정하면, 우리는 10의 낮은 Interval 값과 20의 높은 Interval 값 사이에서 고르게 분할된 11개의 값을 볼 수 있습니다.



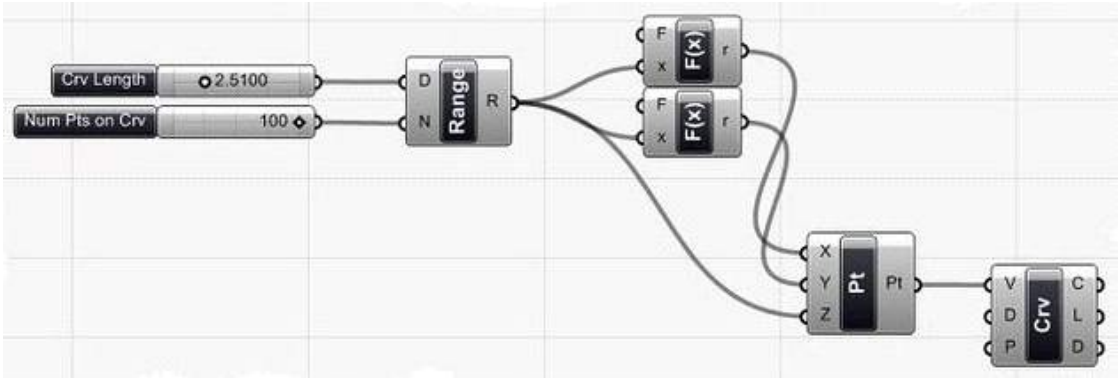
7.3 함수와 Booleans

코드는 첫 번째 Object를 보고, 그것이 곡선인지 아닌지에 대해 하나의 논리 값을 결정합니다. 거기에는 중간이 없습니다. 그 논리 값은 그 Object가 곡선이면 True, 그 Object가 곡선이 아니면 False 입니다. 그 명제의 두 번째 부분이 그 조건 명제의 결과에 의존하는 기능을 실행하는 경우, 만약 그 Object가 곡선이라면, 그 다음 그것을 삭제합니다. 이런 조건 명제는 If/ Else 명제라고 불립니다.
Grasshopper는 함수 컴포넌트의 사용을 통해 조건 명제들을 분석하는 능력을 가집니다. 예제처럼, 함수 컴포넌트(Logic/스크립트/F1)의 x-입력으로 numeric Slider를 연결했습니다.



추가적으로 조건 명제는 "x는 5보다 더 큽니까?" 그 질문을 정의하면서 그 함수의 F-입력에 연결되었습니다. 만약 그 numeric Slider가 5이하일 때는, 그 r-출력은 False 값으로 바뀝니다.

어떤 기능을 실행하기 우해 우리가 함수의 논리 값을 결정했다면, 우리는 Dispatch(Logic/ List/ Dispatch) 컴포넌트로 True/ False 패턴 정보를 공급할 수 있습니다. 그 Dispatch 컴포넌트는 정보의 목록을 이용하여 작동하며, 그 정보 여과기는 하나의 변화하기 쉬운 함수의 논리 패턴 결과에 기초합니다.
만약, 그 패턴이 True 값을 나타내면, 그 정보 목록은 Dispatch-A 출력으로 넘어갈 것입니다.
만약, 그 패턴이 False 값을 나타내면, 그것은 Dispatch-B 출력으로 정보 목록은 넘어갑니다.
이번 예제를 위해 우리는 그 Slider 값이 5보다 더 큰 경우에만 원을 만들기로 결정했습니다. 우리는 논리 값이 Dispatch 컴포넌트 안에서 True로 넘어가는 경우에만 numeric Slider에 의해 지정된 반지름으로 원이 만들어질 수 있도록 하기 위해, Circle 컴포넌트(Curve/ Primitive/ Circle)를 Dispatch-A 출력에 연결했습니다. 어떤 컴포넌트도 Dispatch-B 출력에 연결하지 않은 경우, 그 논리 값이 False 라면, 그때는 원 뿐만 아니라 무엇도 생성되지 않을 것입니다.

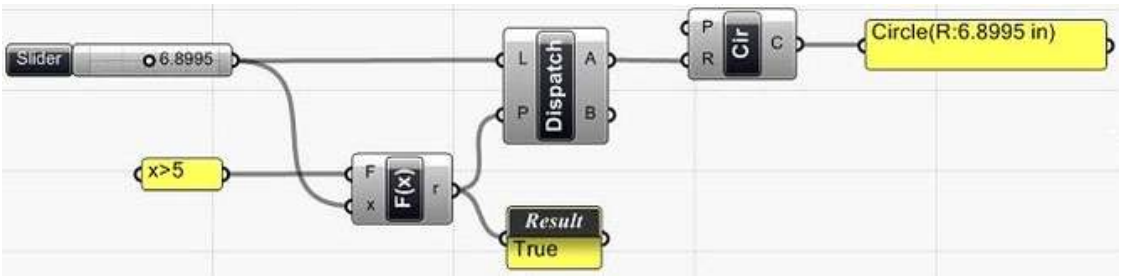


다각형 N-sided로부터 B를 출력해내는 관계에서, 생성되어지는 다각형 R-입력에 연결되는 선으로서 다각형의 반지름을 정의할 수 있다. numeric Slider가 5보다 아래로 떨어진다면, 그때는 number Slider에 정의된 반지름을 가지는 오각형이 원점에서 만들어질 것입니다. 만약, 5보다 높은 Slider 값을 가질 때에는, 원이 만들어질 것입니다. 이러한 방법으로 우리는 정의 전체에 정보를 공급하기 위한 만큼 많은 If/ Else 명제를 만들 수 있습니다.

7.4 함수와 숫자 데이터

함수 컴포넌트는 여러 다른 용도로 사용될 수 있기 때문에 매우 유용합니다. 우리는 이미 조건 명제를 평가하고, 논리 값 출력을 전달하기 위해 우리가 함수 컴포넌트를 어떻게 사용할 수 있는지 논의했습니다. 그러나 우리는 또한 복잡한 수학적 알고리즘을 푸는 기능 컴포넌트를 사용할 수 있고, 출력으로 숫자 데이터를 표시할 수 있습니다.

다음 예제에서, 우리는 David Rutten의 Rhino스크립트101 manual에서 제공된 예제와 유사한 수학적 나선을 만들 것입니다.



처음부터 정의를 만들기 위해

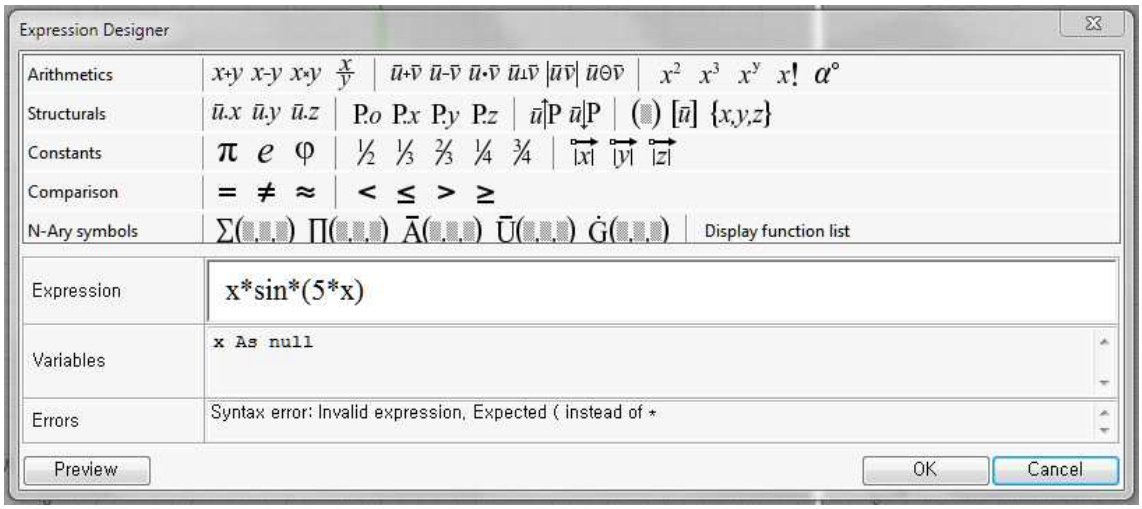
- Logic/ Sets/ Range 캔버스에 Range 컴포넌트를 끌어다 놓습니다.
- Parama/ Special/ Slider 캔버스 위에 두 개의 numeric Slider를 끌어다 놓습니다.
- 첫 번째 Slider를 오른쪽 클릭하고 아래와 같이 설정합니다.
    - o 이름: Cry Length
    - o Slider 유형: Floating 점(기본 값 설정)
    - o 최소 값: 0.1
    - o 최대 값: 10.0
    - o 결과 값: 2.5
- 이제 두 번째 Slider를 오른쪽 클릭하고 아래와 같이 설정합니다.
    - o 이름: Num Pts on Cry
    - o Slider 유형: Integers
    - o 최소 값: 1.0
    - o 최대 값: 100.0
    - o 결과 값: 100.0
- Cry Length Slider Range-D 입력에 연결
- Num Pts on Cry Slider를 Range-N 입력에 연결

    함수 컴포넌트들로 제공할 수 있는 0에서 2.5 범위로 고르게 간격을 둔 101개수들의 범위를 만들었습니다.
- Logic/ Script/ F1 캔버스 위에 하나의 함수 컴포넌트를 끌어다 놓습니다.
- 함수 컴포넌트의 F-입력을 오른쪽 클릭하고 Expression Editor를 엽니다.

- 새로 뜨는 대화상자에서 아래의 방정식을 입력합니다.
    - o **x*sin*(5*x)**

    정확하게 그 편집자 안에 그 알고리즘을 입력했다면, 그 오류 롤 아웃(내부 주기억 장치의 내용을 외부 보조 기억 장치로 옮기는 것.) 아래에서 "No syntax errors detected in expression- 표현식에서 발견된 구문 오류가 없다."라는 명령문을 보아야만 합니다.

    o 알고리즘을 받아들이기 위해 OK를 클릭합니다.



- Expression Editor 대화상자에 아래의 방정식을 입력합니다.
    - o **x*cos*(5*x)**

이 방정식에서 유일한 차이는 우리가 sin함수를 cos함수로 교체한 것입니다.

    o 알고리즘을 받아들이기 위해 OK를 클릭합니다.

- 양쪽 함수 컴포넌트에서 X-입력으로 Range-R 출력을 연결합니다.

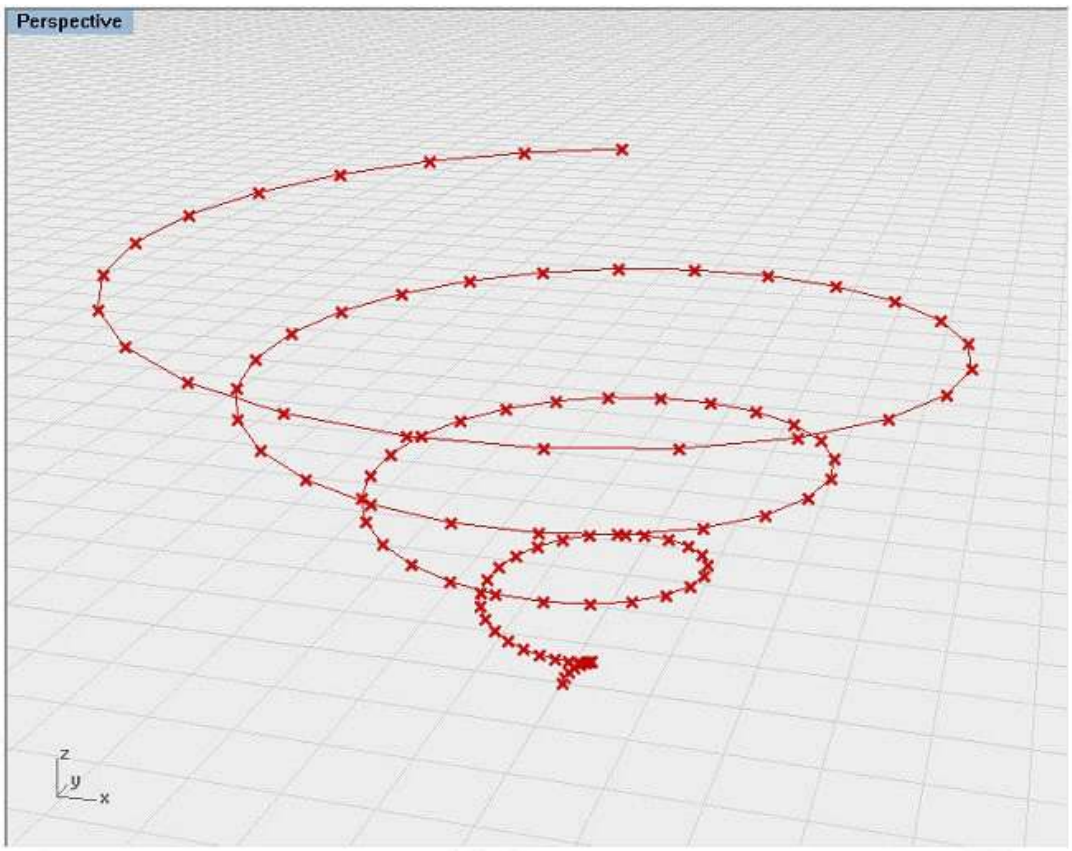
  수학적 알고리즘을 풀고 숫자 데이터의 새로운 목록을 출력하는 그 함수 컴포넌트 안에 Range 구성 요소에 의해 만들어진 101개의 수를 제공했습니다. 각 방정식의 r-출력에 의한 결과 값을 확인 할 수 있습니다.
- 캔버스 위에 점 XYZ 컴포넌트를 끌어다가 놓습니다.
- 점 컴포넌트의 X-입력으로 첫 번째 함수-r 출력을 연결합니다.
- 점 컴포넌트의 Y-입력으로 두 번째 함수-r 출력을 연결합니다.
- 점 컴포넌트의 Z-입력으로 그 Range-R 출력을 연결합니다.

  뷰포트에 나선을 형성하는 포인트의 배치를 확인 할 수 있을 것입니다. 또 나선의 길이 또는 그 나선 위에 포인트들의 수를 변경하기 위해 그 정의를 시작할 때 그 두개의 numeric Slider를 조절할 수 있습니다.
- Curve/ Spline/ Curve 캔버스위에 Curve 컴포넌트를 끌어 놓습니다.

- Curve-V입력에 점-Pt 출력을 연결합니다.

  나선의 각 점을 통과하는 하나의 곡선을 만들었습니다. 곡선 차수를 설정하기 위해 Curve-D입력에 오른쪽 클릭을 하면, 차수=1 곡선은 각 포인트 사이에서 직선 구획을 만들 것이며, 그 곡선이 각 포인트를 실제로 통과하는 것처럼 보여줍니다. 차수=3 곡선의 포인트들은 매끄러운 곡선을 만들어냅니다.

  그러나 그 선들은 각 포인트들을 실제로 통과하지는 않습니다.

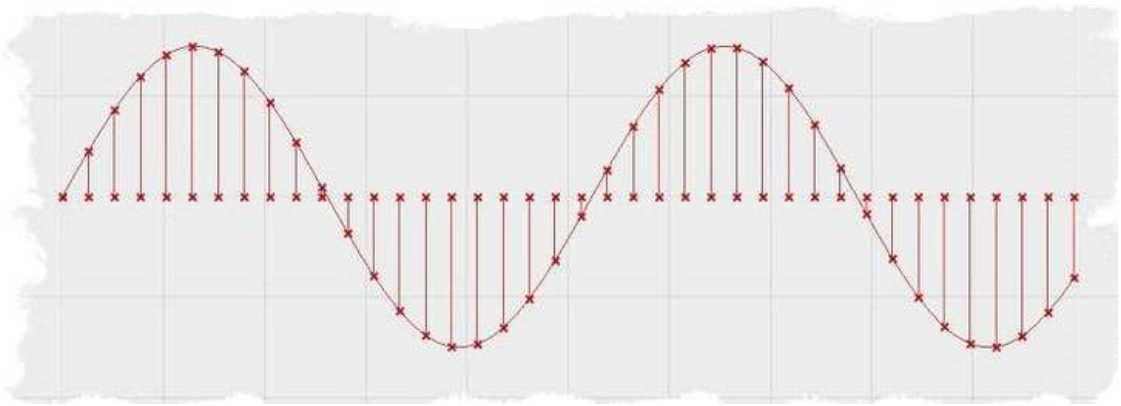


7.5 삼각법에 의한 Curve

다른 수학적 곡선과 나선을 만들기 위해 복잡한 공식을 평가할 수 있도록 함수 컴포넌트를 사용할 수 있습니다. Grasshopper 또한 Scalar 컴포넌트 구성 안에서 삼각 컴포넌트를 만들 수 있습니다. sin, cos, tan같은 함수들은, ɵ라고 불리는 특정 각도를 포함하고 있는 직각 삼각형의 두 개의 측면들 사이 비율을 정의하기 때문에 수학자, 과학자 그리고 엔지니어들에게 중요한 도구입니다. 또한 해파, 음파 그리고 광파의 형태로 자연에서 종종 발견된 sin파 함수와 같은 주기적 구조를 위해 단순한 벽돌이 건축에 사용될 수 있다는 것을 발견하였고, 거의 모든 주기적 파형을 설명했습니다.

그 과정은 푸리에 해석이라고 불립니다.

다음 예제에서는 곡선, 파장, 진동수 그리고 진폭 위에 점들의 수가 numeric Slider의 설정에 의해 제어될 수 있는 sin파 형상을 만듭니다.

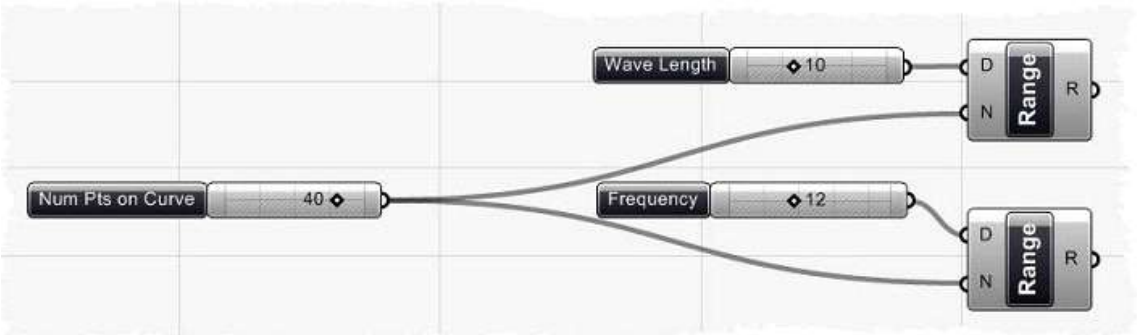


정의를 만들기 위하여

- Params/ Special/ Slider- 캔버스 위에 3개의 numeric Slider를 끌어 놓습니다.
- 첫번째 Slider를 선택하고, 아래와 같이 Parameter를 설정합니다.
    - o 이름: Num Pts on Curve
    - o Slider 유형: Integers
    - o 최소 값: 1
    - o 최대 값: 50
    - o 결과 값: 40
- 두 번째 Slider를 선택하고, 아래와 같이 Parameter를 설정합니다.
    - o 이름: Wave Length(파장)
    - o Slider 유형: Integers
    - o 최소 값: 0
    - o 최대 값: 30
    - o 결과 값: 10
- 세 번째 Slider를 선택하고, 아래와 같이 Parameter를 설정합니다.
    - o 이름: Frequency(진동수)
    - o Slider 유형: Integers

- 최소 값: 0
- 최대 값: 30
- 결과 값: 12

- Logic/ Sets/ Range 캔버스 위에 Range 컴포넌트를 끌어 놓습니다.
- 첫 번째 Range-D입력에 Wave Length Slider를 연결합니다.
- Range-D입력에 Frequency Slider를 연결합니다.
- 양쪽 Range-N입력에 Num Pts on Curve Slider를 연결합니다.



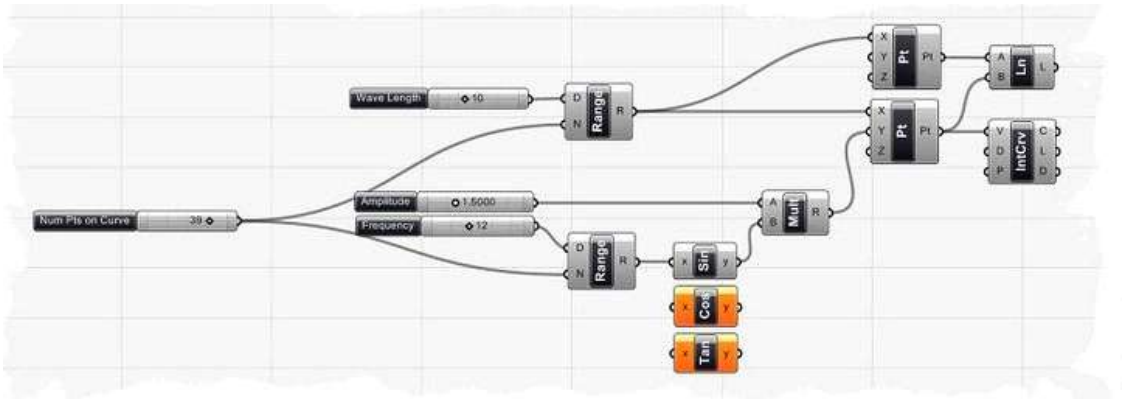
첫 번째는 0에서 10까지 고르게 분할되는 수의 범위, 그리고 두 번째는 0에서 12까지 정렬하는 고르게 분할되는 수의 목록을 정의하였습니다.

- Scalar/ Trigonometry/ Sine 캔버스 위에 Sine 컴포넌트를 끌어 놓습니다.
- 두 번째 Range-R 출력을 Sine 컴포넌트의 x-입력으로 연결합니다.
- Vector/ Point/ Point XYZ-캔버스 위에 점 XYZ 컴포넌트를 끌어 놓습니다.
- Range-R 출력을 점 XYZ 컴포넌트의 X-입력으로 연결합니다.
- Sine 컴포넌트의 y-출력을 점 XYZ 컴포넌트의 Y-입력으로 연결합니다.
- 그럼, Sin파 형태의 연속적인 점들을 볼 수가 있습니다.

  Range 출력이 삼각함수 컴포넌트의 제공 없이 점 컴포넌트의 X-입력에 직접 연결되므로 포인트에서 x값은 일정하고 고르게 간격을 두게 됩니다.
  그러나 Sine 컴포넌트는 Point 컴포넌트의 Y-입력으로 제공하고 있기 때문에 포인트의 y값에 있어서 물결 패턴의 형상을 볼 수 있습니다. 물결 패턴의 형상을 변경하기 위해 numeric Slider를 변경할 수 있습니다.

- Curve/ Spline/ Interpolate- 캔버스 위에 Interpolated Curve 컴포넌트를 끌어 놓습니다.
- Point-Pt 출력을 Curve-V입력으로 연결합니다.

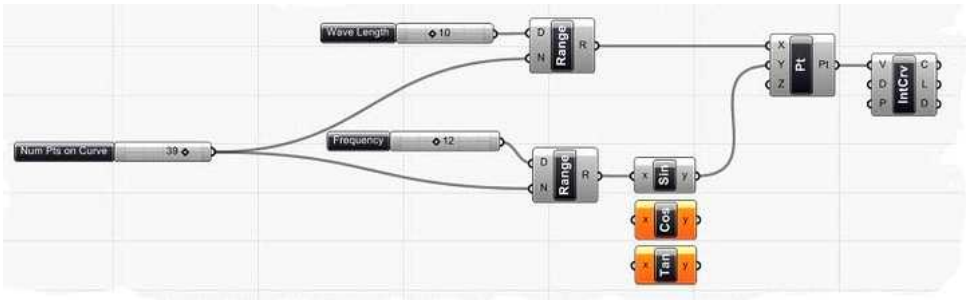
Num Pts on Curve, Wave Length, Frequency Slider를 사용하여 정의를 설정하였고, sin 곡선의 진폭을 제어하기 위하여 마지막 한 개의 Slider를 설정할 수 있습니다.



- 새로운 Slider 위에 오른쪽 클릭하고 아래와 같이 설정합니다.
    - o 이름: Amplitude(진폭)
    - o Slider 유형: Floating 점
    - o 최소 값: 0.1
    - o 최대 값: 5.0
    - o 결과 값: 2.0
- Scalar/ Operators/ Multiplication- 캔버스 위에 Multiplication(곱셈) 컴포넌트를 끌어 놓습니다.
- Amplitude Slider를 Multiplication-A 입력으로 연결합니다.
- Sin 컴포넌트의 y-출력을 Multiplication-B 입력으로 연결합니다.
- Multiplication-r 출력을 점 XYZ 컴포넌트의 Y-입력으로 연결합니다.
- Sin 컴포넌트의 출력으로 이전에 연결되어 있었던 기존의 연결을 교체합니다.

    Amplitude Slider는 Sin 값에 배율을 곱하고 있으며, 그 Slider는 Sin 곡선의 Y-값을 조정하는 것이므로 진폭 값을 증가할 때, 곡선의 진폭을 차례로 늘립니다.
- Vector/ Point/ Point XYZ 컴포넌트를 끌어 놓습니다.
- 첫 번째 Range-R 출력을 새로운 점 XYZ 컴포넌트의 X-입력으로 연결합니다.
- Curve/ Primitive/ Line 캔버스 위에 Line 컴포넌트를 끌어 놓습니다.
- 첫 번째 Point-Pt 출력을 Line-B입력으로 연결합니다.
- 두 번째 Point-Pt 출력을 Line-A입력으로 연결합니다.

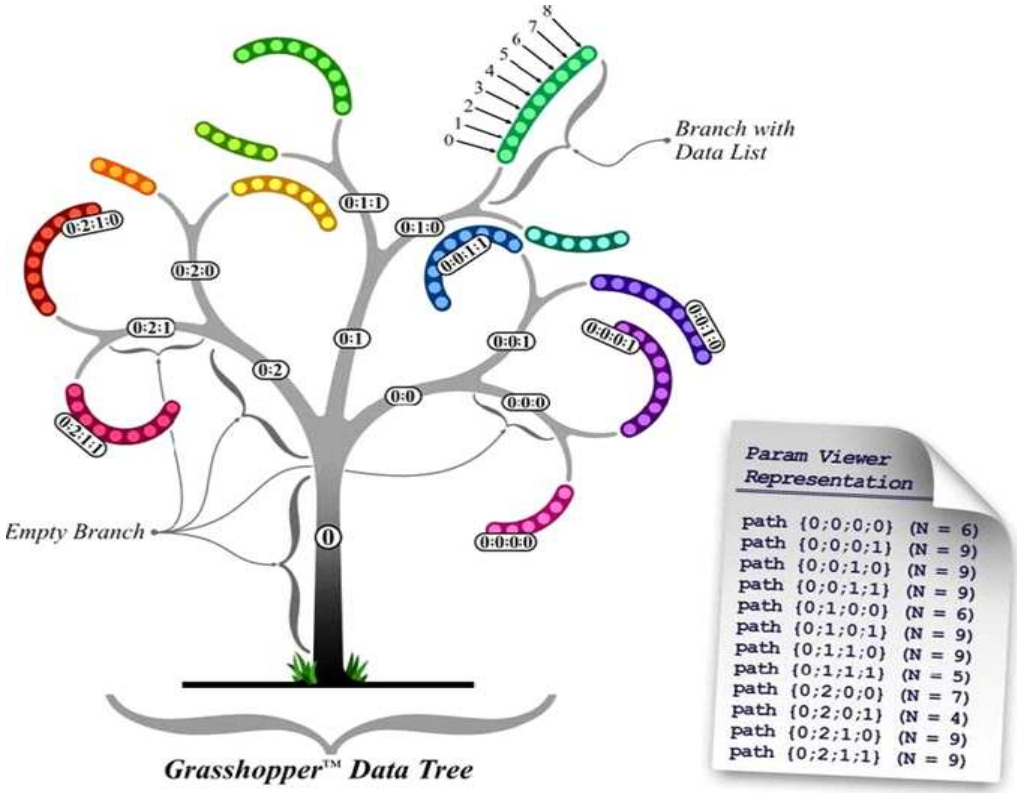
    Sin 곡선의 동일한 x-좌표에 X-축을 따라 고르게 간격을 두는 Point의 두 번째 설정을 정의하였습니다. Line 컴포넌트는 첫 번째 포인트의 목록, sin 곡선을 만듭니다. X-축을 정의하는 두 번째 포인트의 목록 사이에서 선분을 만듭니다. 그 새로운 선들은 물결 형태 패턴에서 수직 이동의 시각적인 대조를 보여줍니다.



sin 파장 곡선을 만드는 방법입니다.
Scalar/ Trigonometry 탭 아래에서 찾게 되는 Cosine 또는 Tangent 컴포넌트로 Sine 컴포넌트를 교체하는 방법을 통해 cos 또는 tan 함수와 같은 물결 형태의 곡선을 구성할 수 있습니다.

8 고정된 점들의 집합

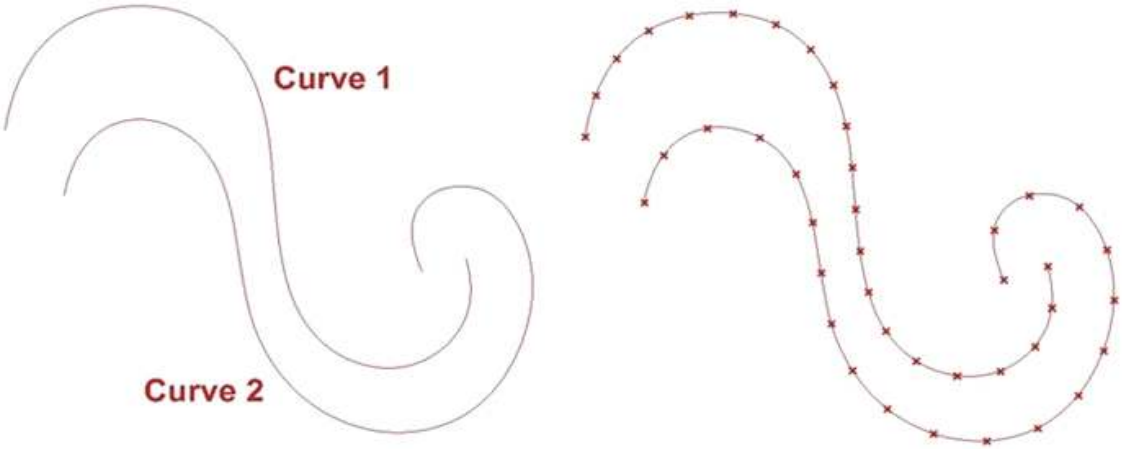
0.6.00xx버전에 앞서 공개되었던 Grasshopper의 모든 버전에서, 매개변수 내부의 데이터는 하나의 목록에 보관되었습니다. 그리고 그와 같은 것은 목록 색인을 필요로 하지 않았습니다. 그러나 데이터가 저장된 방법의 완전한 분해 검사가 Grasshopper에 있었고, 이제 그것이 하나의 매개변수 내부에서 다양한 데이터의 목록을 가지는 것이 가능합니다.
다양한 목록을 이용할 수 있음으로, 각 개개의 목록을 확인하는 방법이 필요합니다. 아래 이미지는 꽤 복잡한 표현이지만 멋지게 구성된 나무 도형으로써, David Rutten에 의해 만들어졌습니다.



위의 이미지에는 { 0 } 경로에서 하나의 줄기가 있습니다. 이 경로는 데이터를 포함하고 있지 않지만, 3개의 보조 가지가 있습니다. 그러한 각각의 보조 가지들은 그 상위가지인 { 0 }의 색인을 상속받고, 그것들은 자신 보조색인 (제각기 0, 1, 2)을 추가합니다. 그것은 단지 하나의 수를 암시하는 것이기 때문에, 이것을 색인이라고 부르는 것이 아니고 원반 모양의 폴더 구조와 닮았으므로 "경로"와 같이 참조합니다.
각각의 보조 가지들은 두개의 하위 가지들에게, 그것들은 다시 어떤 그들 자신의 데이터를 포함하지 않습니다. 이것은 하위 가지들도 마찬가지입니다. 만일 레벨 4에 도달하게 되면 일부 데이터(목록은 색채가 다양한 두꺼운 라인으로 표현되며, 데이터 항목은 원으로 표현됩니다.)를 마주합니다. 모든 하위 가지(또는 레벨 4가지)는 그것들이 그 이상 분할하지 않는 것을 의미합니다.

각각의 데이터 항목은 그 나무에서 오직 하나의 가지이므로, 각각의 항목은 가지 내에 그 위치를 지정하는 색인을 가집니다. 각각의 가지들은 나무 내에 그 위치를 지정하는 경로가 있습니다.

나무 구조를 보다 섬세하게 분석하기 위해 매우 단순한 예제를 가지고 살펴보겠습니다. Grasshopper에서 언급하는 두 개의 곡선을 시작으로 양쪽 곡선을 20개의 개별적인 구간으로 나누기 위해, 우리는 하나의 곡선 분할(Curve/ Division/ Divide Curve) 컴포넌트를 사용할 것입니다. 결국, 각각의 곡선에 21개의 점을 생성하는 것입니다. 그 분할 점들을 통해 새로운 선을 만드는 폴리라인(Curve/ Spline/ Polyline) 컴포넌트로 그러한 점들에 모두 바로 공급합니다.

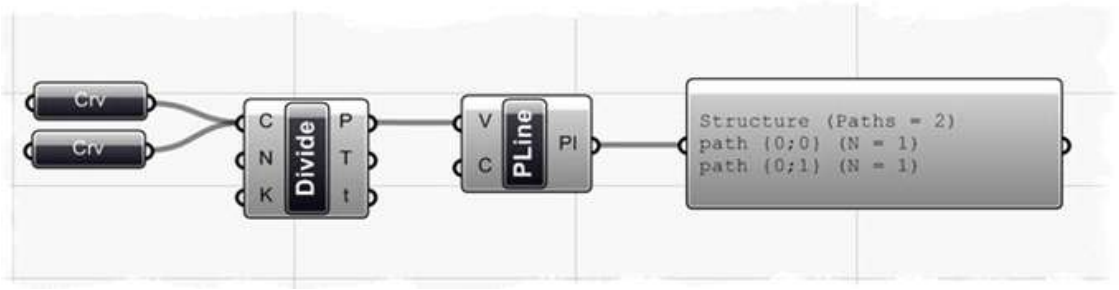


만약 우리가 Grasshopper 0.5또는 이전 버전을 사용하고 있었다면, Polyline 곡선은 전체 점 목록(Divide Curve)컴포넌트의 결과와 같이 41 포인트를 가지는 것을 통하여 하나의 선을 그릴 것입니다. 이것은 포인트 전체가 한 개의 목록(가지 또는 경로 없이)에 보관되었기 때문입니다. 그래서 그 컴포넌트는 목록에 포인트 모두를 통하여 Polyline을 그리도록 되어있습니다. 만약, Grasshopper 0.5 버전을 사용하고 있었다면, 그 포인트 색인과 결과적인 폴리라인은 다음과 같이 보여 졌을 것입니다.
그러나 우리는 Grasshopper가 경로와 가지를 통합하는 능력을 가지는 것을 알고 있으므로, 그 색인을 사용하여 Polyline 컴포넌트가 어떻게 작용하는지 제어할 수 있습니다.
만약, 우리가 Grasshopper 0.6.00xx 버전 또는 높은 버전을 사용하여 이전처럼 정확하게 이 단계를 따랐다면, 그것이 각각 나누어진 경로가 20개의 구분이라고 인정하기 때문에 Polyline의 컴포넌트는 2개의 Polyline을 만들 것입니다. 우리는 매개변수 뷰어(Params/ Special/ Param Viewer)를 이용하여 나무 구조를 조사할 수 있습니다. 아래의 구조가 각각 두 개의 경로로 시작에서 한 개의 결과로서 생기는 곡선을 나타내는 모습입니다.

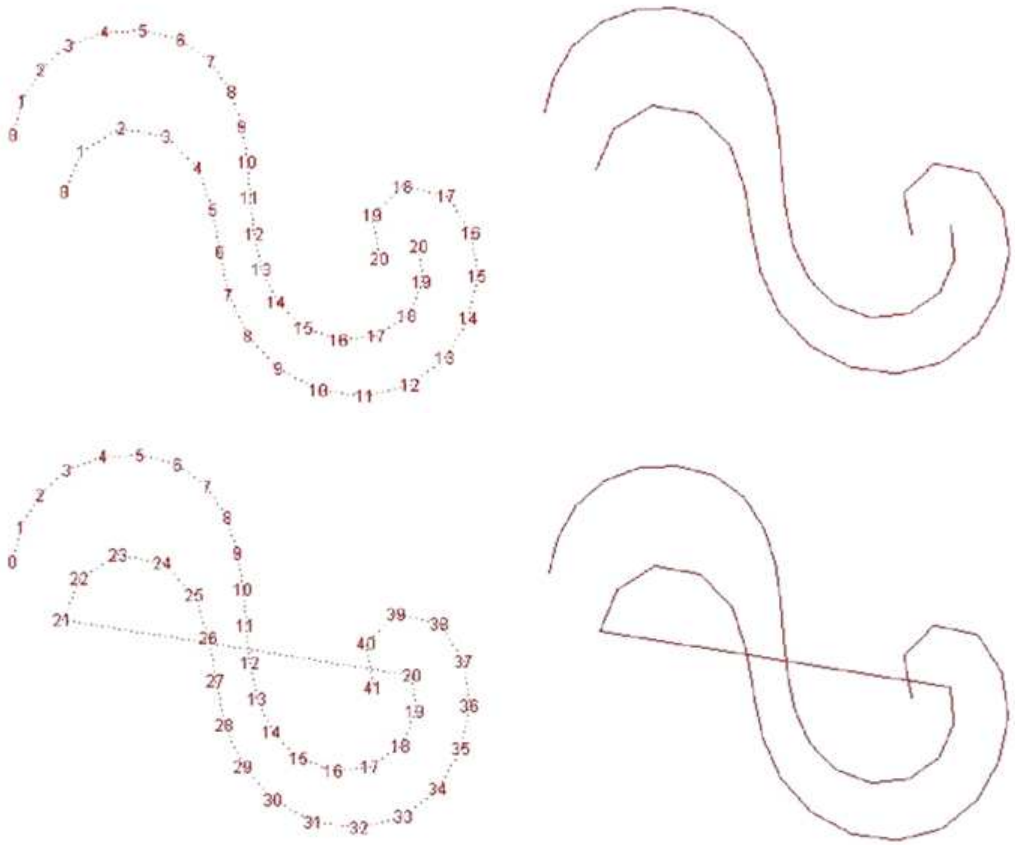
두 개의 폴리라인의 포인트 색인과 그 결과 곡선입니다.

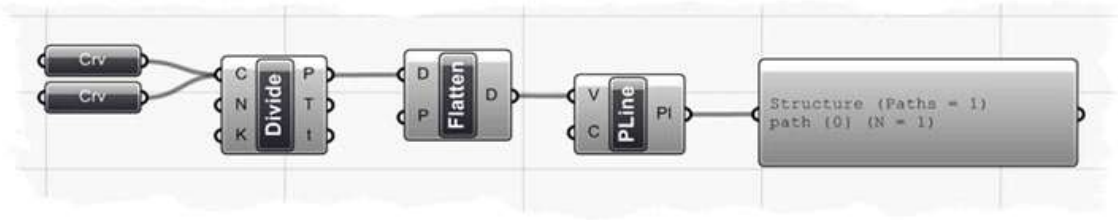
우리는 정의가 두 개의 경로를 가진다는 것을 알고 있습니다. 하지만, 우리가 실제로 한 개의 경로와 그 결과 한 개의 폴리라인의 결과를 원했더라도, 정의는 두 개의 경로를 가진다

는 것을 알고 있습니다. Grasshopper는 나무 구조의 제어를 도울 수 있게 Logic탭의 Tree 하위 카테고리 아래에서 다양한 구조 컴포넌트들을 가지고 있습니다. 그 데이터가 0.5 또는 이전 버전에서 표시되어진 것과 흡사하다면, 하나의 목록으로써 전체 나무 구조를 무너뜨리는 Flatten Tree 컴포넌트(Logic/ Tree/ Flatten Tree)를 사용할 수 있습니다.

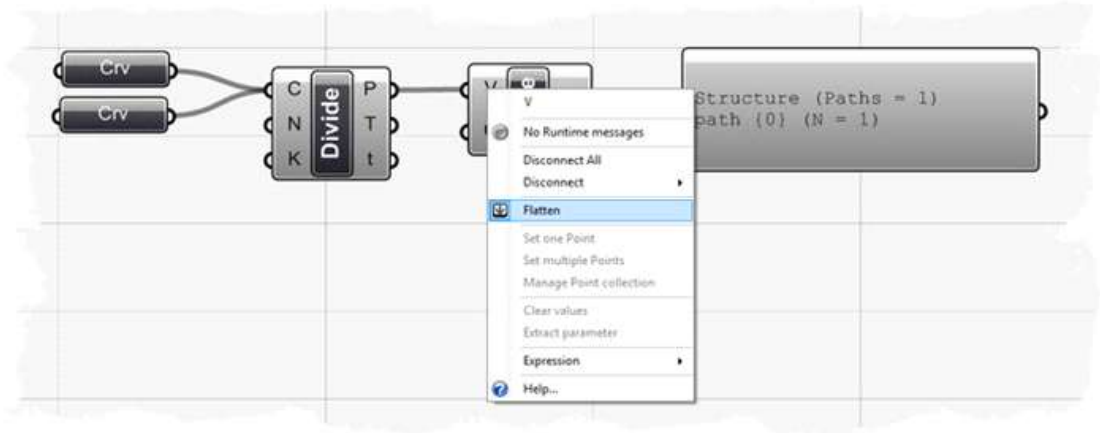


결과 값으로 생기는 곡선과 더불어 구조는 한 개의 경로를 가진다는 것을 이미지에서 알 수가 있습니다. 결국, 이전 Grasshopper 버전에서와 같은 하나의 폴리라인을 갖게 됩니다.



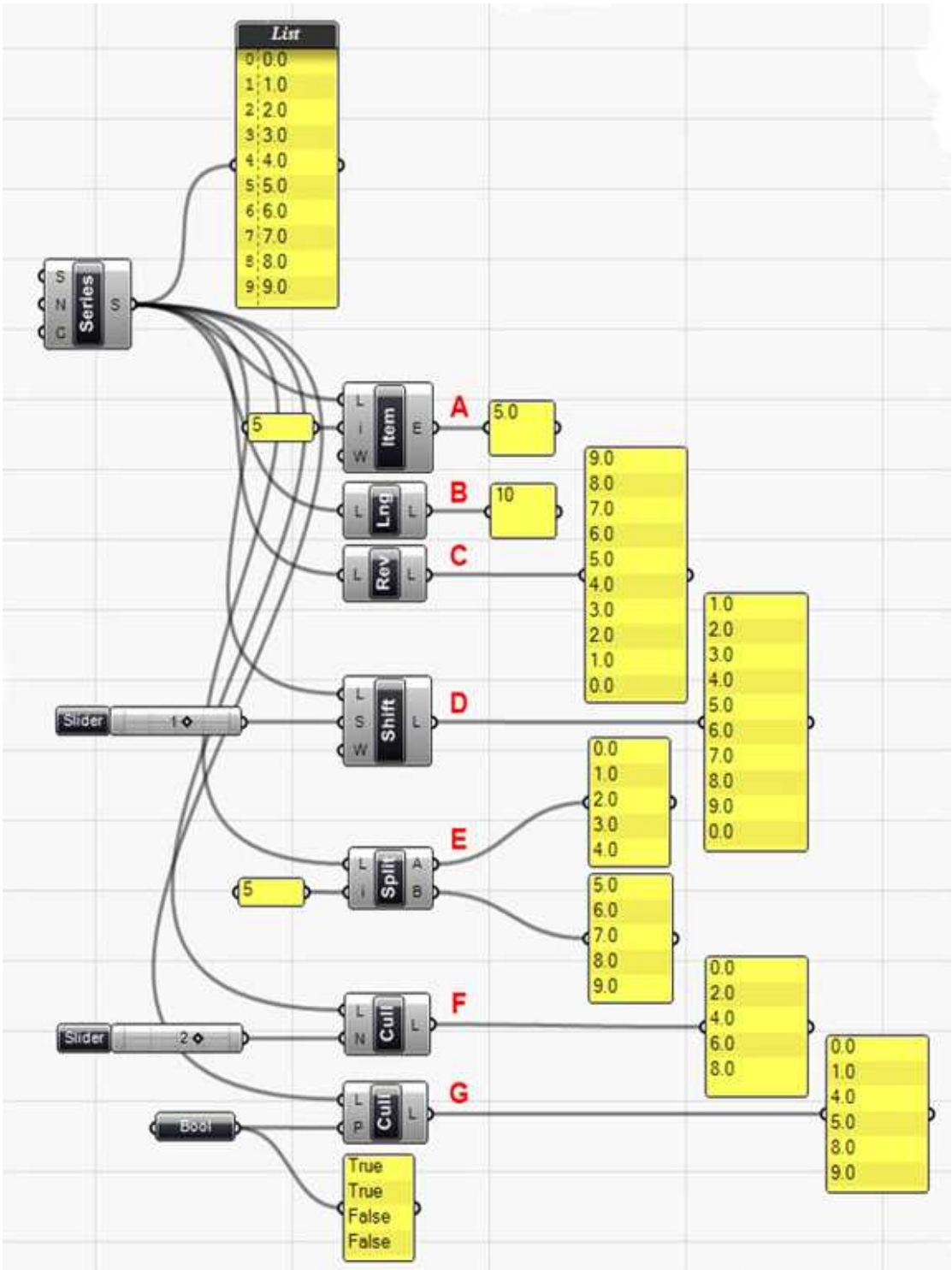


또한, 구성 요소 내부의 Flatten 컴포넌트를 거쳐서 이 정의를 단순화 할 수 있습니다. Polyline 컴포넌트의 V-입력 위에 오른쪽 클릭하고 "Flatten"를 선택하여 전환합니다.
이 방법을 통해 두 개의 나무 구조가 한 개의 경로로 변환합니다.

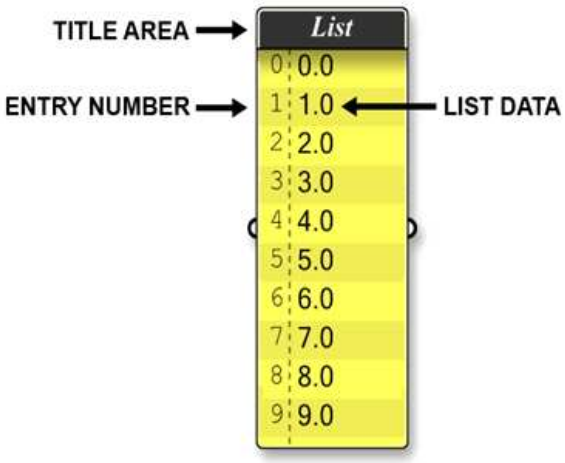


8.1 목록과 데이터관리

Grasshopper의 흐름에 관해서 그래픽 인터페이스는 특별한 형식의 컴포넌트를 안에서 밖으로 흘려 정보를 가지도록 디자인 되었습니다. 그러나 그것은 컴포넌트의 안과 밖으로 흐르는 정보를 정의하는 데이터(포인트, 곡선, 표면, 문자열, 논리, 숫자, 등등)입니다. 목록 데이터를 조작하는 방법을 이해하는 것은 Grasshopper 플러그인을 이해하기 위해 중요합니다.
다음은 Logic 탭의 보조 카테고리 목록 아래에 나타나는 여러 가지 List 컴포넌트를 사용하고 있는 숫자 데이터의 목록을 통제할 수 있는 방법의 예입니다.



시작하기 위해 0.0으로 시작하는 값, 1.0의 단계 값 그리고 10개의 총계와 Series 컴포넌트를 만듭니다. Series-S 출력에 연결된 패널은 다음의 수 목록을 보여줍니다. 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0



그것들은 패널에서 오른쪽 선택을 하고 등록 수 미리보기를 on 또는 off로 변경할 수 있습니다. 그러나 그것이 목록 색인 수에 각각의 값에 할당된 것이 무엇인지 정확하게 보도록 허용할 것이므로 이번 예제에서는 등록 수를 켜지게 할 것입니다.

A) 그 숫자 데이터는 목록 내에 특정의 등록을 회수하기 위해 사용되는 List Item 컴포넌트(Logic/ List/ List Item)로 그 다음 공급됩니다. 목록에서 개개의 항목에 접근할 때, 한 개는 색인 값을 사용해야 합니다. 어떠한 목록에서 그 첫 번째 항목은 위치 0, 두번째 항목은 위치 1, 등등에 항상 보관됩니다.
일반적으로 색인 -5에 접근하기 시작하면 어떤 위치도 존재하지 않으므로 오류가 발생할 것입니다. 그 목록을 Item-L 입력 안으로 Series-S 출력을 연결합니다.
추가적으로 원하는 목록 색인 수를 정의하는 Item-i 입력 안으로 정수를 제공합니다. 이 값을 5.0에 조정했으므로, 출력된 항목은 이 경우 5.0인 다섯번째 등록 수와 관련되는 숫자 데이터를 보여줍니다.

B) List Length 컴포넌트는 본질적으로 목록의 등록 수를 평가하고, 목록의 마지막 등록 수 또는 길이를 출력합니다. 이번 예제는 그 목록에서 10 값을 보여주기 위해 Series-S 출력을 List Length-L입력으로 연결합니다.

C) 목록의 순서를 Reverse List 컴포넌트(Logic/ List/ Reverse List)를 사용하여 거꾸로 할 수 있습니다. Reverse List 컴포넌트로 수의 올라가고 있는 목록을 입력합니다. 그것에 의하여 출력은 9.0부터 0.0으로 내려가고 있는 목록을 보여줍니다.

D) Shift 컴포넌트(Logic/ List/ Shift List)는 오프셋 이동에 의존하는 많은 증가량의 목록을 올리거나 내립니다. 숫자 Slider를 Shift-S 입력으로 연결하는 동안 Series-S 출력을 Shift-L 입력으로 연결합니다.
오프셋 이동은 전체 수에서 발생되기 때문에 숫자 Slider를 형식의 정수로 설정합니다. 만약 그 Slider를 -1로 설정했다면, 그 목록의 모든 값들은 하나의 등록 수만큼 아래로 내릴 것입니다. 마찬가지로, 만약 그 Slider 값을 +1로 변경하면, 그 목록의 모든 값들은 하나의 등록

수만큼 위로 올라갈 것입니다. 또한, 그 입력을 오른쪽 클릭하고 Set Boolean 을 선택하는 것으로 True 또는 False로 Shift-W 또는 포장 값을 설정할 수 있습니다.
이번 예제는, 그 첫 번째 값을 원하는 방법으로 진행하기 위하여 오프셋 이동 값을 +1로 설정합니다. 만약 포장 값을 True로 설정했다면, 그 첫 번째 등록은 데이터 설정으로부터 이번 값을 제거하고 목록의 위와 바깥으로 이동됩니다.

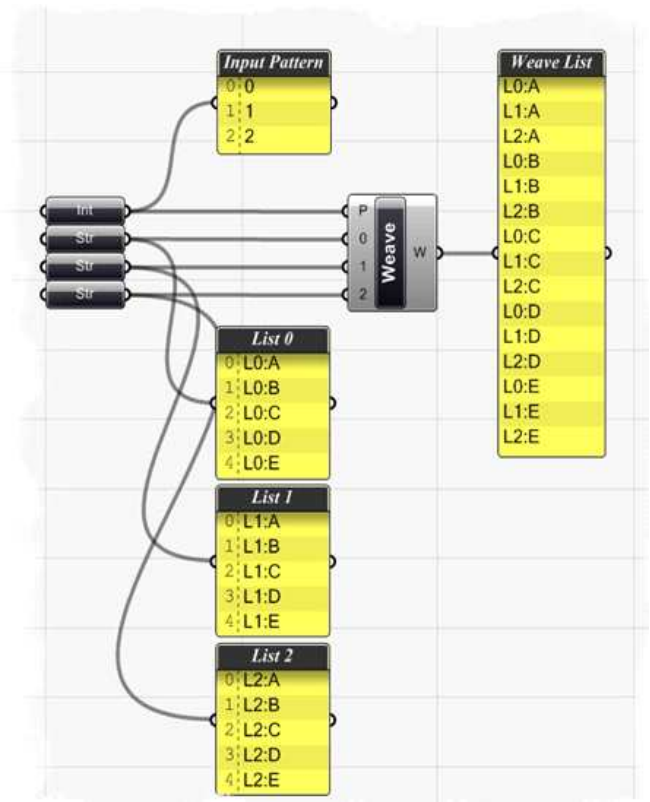
E) Split List 컴포넌트는 목록을 부다 작은 두 개의 목록으로 분할합니다. 그것은 그 목록을 분할하는 곳이 정수이고 분할하는 색인을 갖습니다.
따라서, 출력은 목록 A-0.0, 1.0, 2.0, 3.0, 4.0 목록 B-5.0, 6.0, 7.0, 8.0, 9.0처럼 보일 것입니다.

F) Cull Pattern 컴포넌트(Logic/ Sets/ Cull Pattern)는 숫자 정수에 의해 정의된 N에서 그 목록의 모든 N번째 데이터를 제거합니다. 이번 예제는 숫자 Slider를 Cull Nth-N입력을 연결합니다. Slider를 2.0으로 설정합니다. Cull Nth-L 출력은 홀수마다 삭제된 새로 추려진 목록: 0.0, 2.0, 4.0, 6.0, 8.0을 나타냅니다.
만약 숫자 Slider를 3.0으로 변경한다면 Cull Nth 컴포넌트는 0.0, 1.0, 3.0, 4.0, 6.0, 7.0, 9.0로 나타낼 수 있도록 목록에서 모든 세 번째 수를 제거합니다.

G) Cull Pattern 컴포넌트(Logic/ Sets/ Cull Pattern)는 정의된 값을 근거로 하는 목록의 항목을 제거하는 점에서 Cull Nth 컴포넌트와 비슷합니다. 하지만 이런 경우 숫자 값 대신에 패턴을 형성하는 논리 값의 설정을 사용합니다. 그 논리 값을 True로 설정했다면, 그 데이터 등록은 목록에 남아있을 것입니다. 그러나 False 값은 그 설정으로부터 데이터의 등록을 제거할 것입니다. 이번 예제는 논리 패턴을 True, False, False로 설정합니다. 거기에는 4개의 논리 값과 목록은 10개의 등록으로 그 패턴은 목록의 끝에 이를 때까지 반복됩니다. 패턴과 함께 그 출력 목록은 0.0, 1.0, 4.0, 5.0, 8.0, 9.0처럼 보입니다. Cull Pattern 컴포넌트는 첫 번째, 두번째 등록(0.0과 1.0)을 유지해두고 다음 두 개의 값(2.0과 3.0)이 제거됩니다. 그 컴포넌트는 이 패턴을 그 목록의 끝에 이를 때까지 계속됩니다.

8.2 데이터 조립하기

마지막 섹션에서 Grasshopper에서 목록이 어떻게 다루어지는지 조정하기 위해 다른 요소들이 어떻게 사용될 수 있는지 설명하였다. 그러나 목록의 순서를 조정하기 위해서 Weave 요소 또한 사용할 수 있는데 Logic 탭의 하위계열 리스트에서 Weave 요소를 찾을 수 있습니다.

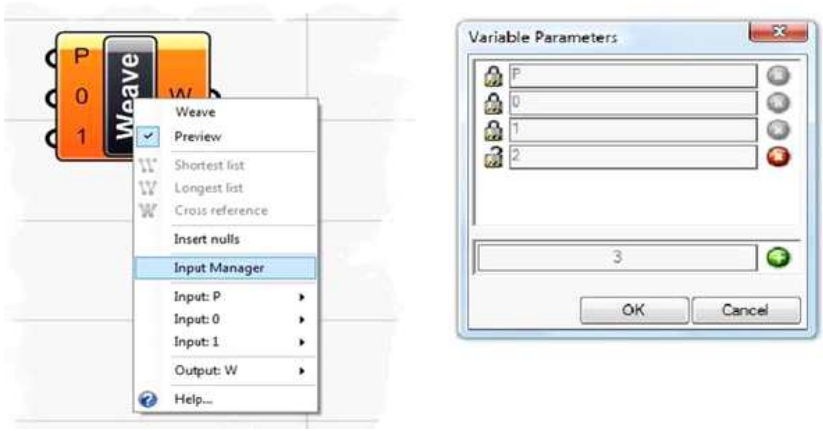


스크래치로부터 명로도 만들기:

Logic/ List/ Weave-Weave 요소를 드래그 하여 캔버스 위에 놓습니다. 첫 번째 P-지점은 Weaving 패턴인데 이것은 자료를 배열하는 순서를 결정합니다. 그 다음 두 개의 입력 아이템 라벨 0과 1은 각각 함께 엮을 두 개의 리스트를 입력합니다. 그러나 2개 이상의 리스트를 더 엮고 싶다면, Weave 요소의 중간에서 오른쪽클릭을 하고 입력 Manager를 열면 필요한 만큼의 리스트를 추가할 수 있습니다.

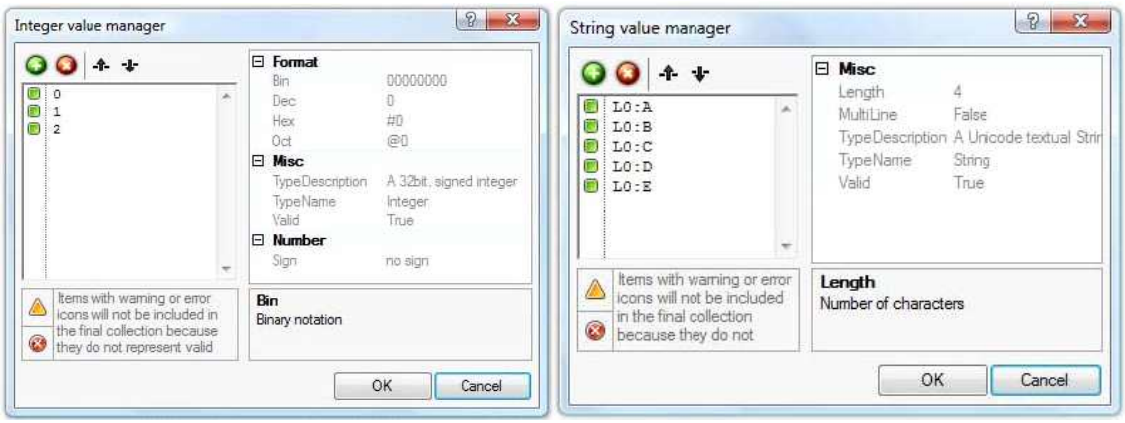
입력 Manager가 열리면 초록색의 플러스 버튼을 누르고 다른 리스트를 추가합니다. 그리고 각 리스트의 숫자 옆에 빨간색 X 컴포넌트 리스트를 지웁니다.

입력 manager를 열고 리스트를 추가하기 위해 초록색 버튼을 선택합니다.



Params/ Primitive/ Integer- Inter Parameter를 캔버스 위에 끌어 놓습니다. Integer Parameter를 오른쪽 클릭하여 "Manager Integer Collection"을 선택합니다.
상단의 녹색 +버튼을 클릭함으로써 컬렉션 정수를 추가할 수 있습니다. 컬렉션에 3개의 숫자를 추가하고 각 값을 0, 1, 2 로 바꿉니다.
이 정수 모음은 패턴의 짜임을 결정짓게 되며, 이러한 숫자들의 순서를 바꿈으로써 빠르게 데이터세트의 순서를 바꿀 수 있습니다.

Params/ Primitive/ String- String 요소를 캔버스 끌어 놓습니다. 오른쪽 클릭을 하고 "Manager String Collection"을 선택합니다.
이제 함께 엮고 싶은 String 리스트를 추가하고 첫 번째 Data List를 만듭니다. 3개의 리스트를 갖기 위해 이것을 2번 복사하여 붙여넣기를 합니다.
이와 같이 컬렉션에 L0:A, L0:B, L0:C, L0:D, L0:E 5개의 스트링을 추가합니다. "L0:" 이것은 리스트 0에 있는 String들을 말해주고 데이터를 추적할 수 있도록 도와줍니다. 다른 요소들과 엮을 경우 다음과 같이 나타납니다.



String Parameter를 선택하고 2개 이상의 String Parameter를 캔버스에 복사하기 위해 Ctrl+C(복사)와 Ctrl+(붙이기)를 실행합니다.
두번째 String Parameter에서 오른쪽 클릭을 하고 Collection Manager를 열고 L1:A, L1:B,

L1:C, L1:D, L1:E 스트링 모음을 바꾸어 줍니다.
세 번째 String Parameter에서 오른쪽 클릭을 하고 Collection Manager를 열고 L2:A, L2:B, L2:C, L2:D, L2:E 스트링 모음을 바꾸어 줍니다.
이제 패턴의 짜임을 바꿔줄 하나의 Integer Parameter를 Weave 요소의 P-입력에 연결합니다. 첫 번째 String Parameter를 Weave 요소의 0-입력에 연결합니다. 두번째 String Parameter를 Weave 요소 1-입력에 연결합니다. 세 번째 String Parameter를 Weave 요소의 2-입력에 연결합니다.
Params/ Special/ Post-it Panel- Post- it Panel을 캔버스에 끌어 놓습니다. Weave-W 결과를 캔버스 위의 Post-it Panel에 연결합니다.
Post-it Panel은 통합모음에 따라 엮어진 데이터의 목록을 보여주며, 첫 번째 아이템은 리스트 0의 첫 번째 아이템이어야 합니다. 따라서 리스트의 두번째 아이템은 리스트 1의 첫 번째 아이템이 됩니다. 리스트 마지막까지 이러한 규칙이 적용되며 목록의 순서를 바꾸기 위하여 Integer 모음의 순서를 바꿀 수 있습니다.

8.3 데이터의 이동

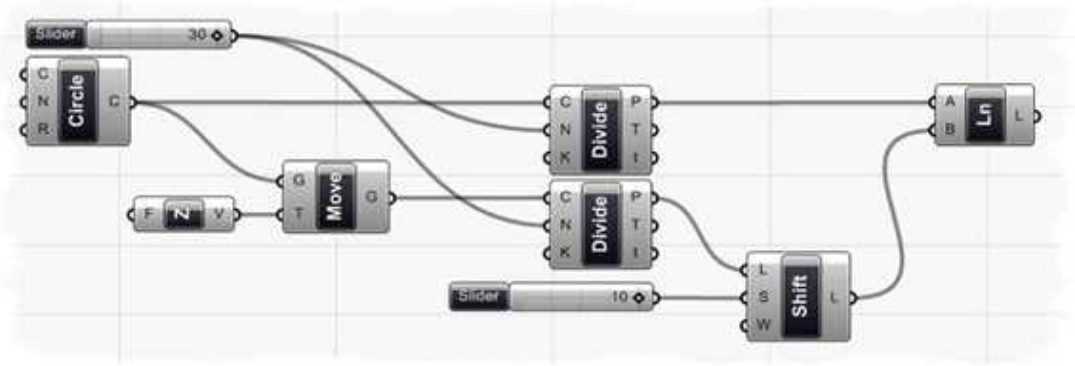
섹션 8.1에서 차감 값의 변화에 따라 리스트의 위, 아래의 모든 값을 이동시키기 위해 Shift 요소를 사용하는지에 대해 논의하였습니다. 아래의 예는 한 원의 두 개 점 리스트에 어떻게 Shift 요소를 사용할 수 있는지 보여주기 위해 David Rutten에 의해 만들어졌습니다.
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplictHistoryShiftExample.html

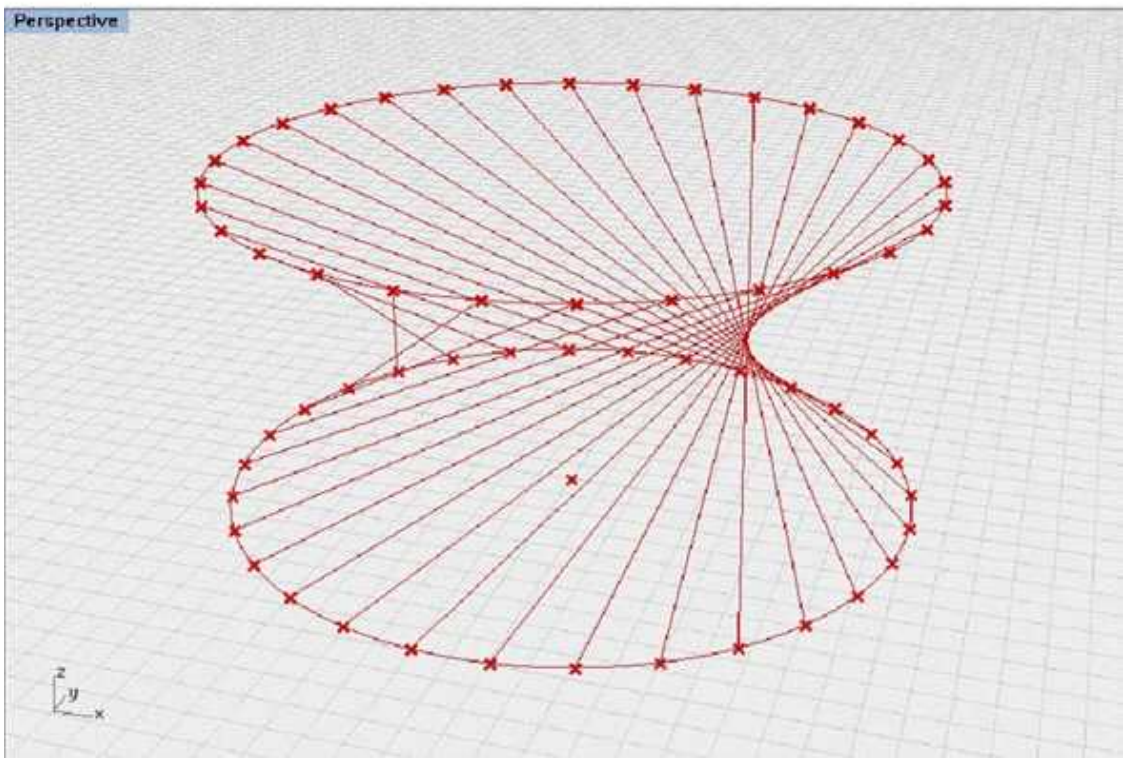
Note: 다음 예는 완성된 정의를 보기 위해 문서가 포함된 Source File 폴더 안의 Shift Circle.ghx 파일을 엽니다. 아래는 이동된 원의 완성된 정의를 보여 줍니다.

스크래치로부터 명료도 만들기:
·Curve/ Primitive/ Circle CNR_ Circle CNR(Center, Normal, Radius)을 드래그 하여 캔버스에 끌어놓습니다.
·Circle-C에서 오른쪽 클릭하고 Set One 점을 클릭합니다.
·Rhino 대화상자에서 0.0.0을 입력합니다.
·Circle-R에서 오른쪽 클릭하고 숫자를 10.0까지 입력합니다.

·Vector/ Constants/ Unit Z-Unit Z Vector요소를 드래그 하여 캔버스에 끌어 놓습니다.
·Unit Z 요소의 F입력에서 오른쪽 클릭을 하고 숫자를 10.0으로 설정합니다.
·X Form/ Euclindean/ Move- Move요소를 캔버스에 끌어 놓습니다.
·Unit Z-V 결과를 Move-T 입력과 연결합니다.

·Circle-C 결과를 Move-G 입력과 연결합니다.
·중심점이 0.0.0에 있고 반지름이 10.0인 원을 하나 만듭니다. 그런 다음 이 원을 복사하고 복사된 원을 Z축 10.0 유닛으로 옮기기 위해 이동요소를 사용합니다.
·Curve/ Division/ Divide Curve- 두 개의 Divide Curve 요소를 드래그 하여 캔버스에 끌어 놓습니다.
·Circle-C 결과와 첫 번째 Divide-C 입력을 연결합니다.
·Move-G 결과와 두번째 Divide-C 입력을 연결합니다.





·Params/ Special/ Slider- 번호가 있는 Slider 하나를 캔버스에 끌어 놓습니다.
·그 Slider를 선택하고 아래 변수와 같이 설정합니다.
o Slider 유형: Integer
o 최소 값: 1.0
o 최대 값: 30.0
o 결과 값: 30.0
·숫자가 있는 Slider를 Divide Curve-N 입력들과 연결합니다.
·각 원을 따라 30포인트로 균일하게 자리 잡은 것을 봅니다.

·Logic/ List/ Shift List- 한 개의 Shift List를 드래그 하여 캔버스에 끌어 놓습니다.
·Divide Curve-P의 결과와 Shift List-L 입력을 연결합니다.

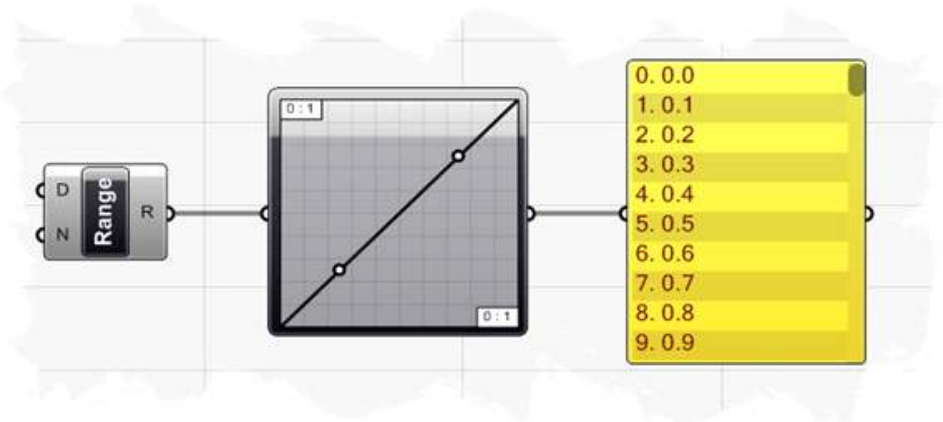
·Params/ Special/ Slider- 숫자Slider를 캔버스에 끌어 놓습니다.
·새로운 Slider를 선택하고 아래와 같이 설정합니다.
o Slider 유형: Integer
o 최소 값: -10.0
o 최대 값: 10.0
o 결과 값: 10.0
·숫자Slider 결과를 Shift List-S 입력에 연결합니다.
·Shift List-W에서 오른쪽 클릭하고 Boolean 값을 True로 설정합니다.
·원에 있는 점을 10을 통해 색인 리스트에 이동합니다. 값을 True로 설정함으로써 데이터를 통한 루프를 만듭니다.

·Curve/ Primitive/ Line- Line 요소를 캔버스 위에 끌어 놓습니다.
·첫 번째 Divide Curve-P 결과와 Line-A 입력을 연결합니다.
·Shift-L 결과를 Line-B 입력에 연결합니다.
원 위에 바뀌지 않은 점에서 바뀐 점의 연결을 선으로 만들어 왔습니다. 바뀌지 않은 점의 리스트와 바뀐 점의 리스트 사이의 선의 연결 관계 변화를 보기 위해 Shift Offset을 조정하는 숫자Slider 값을 변경할 수 있습니다.

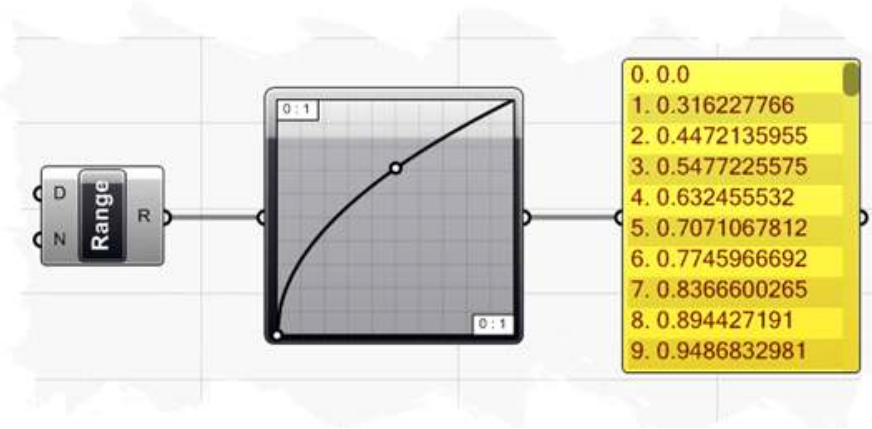
8.4 엑셀로 데이터 내보내기

심화 분석을 위해 Grasshopper에서 다른 소프트웨어에 정보를 보내기 위해 필요한 많은 예들이 있습니다.

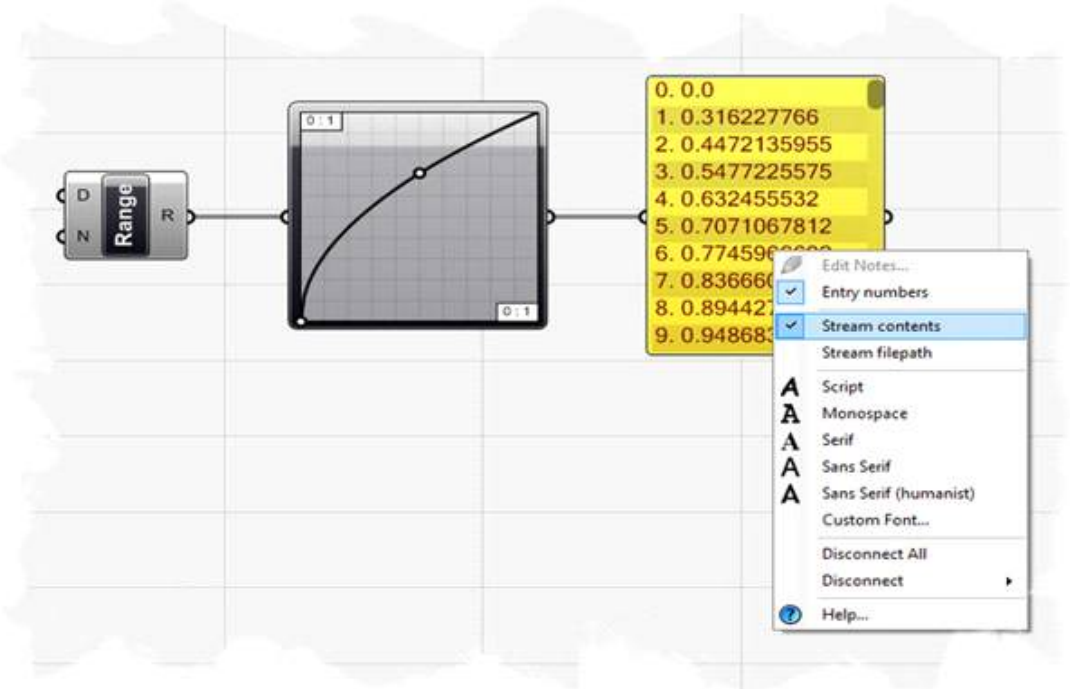
Note: 다음 완성된 정의를 보길 원한다면 문서가 포함되어 있는 Source File 폴더의 Stream Content_Excel.ghx 파일을 엽니다.



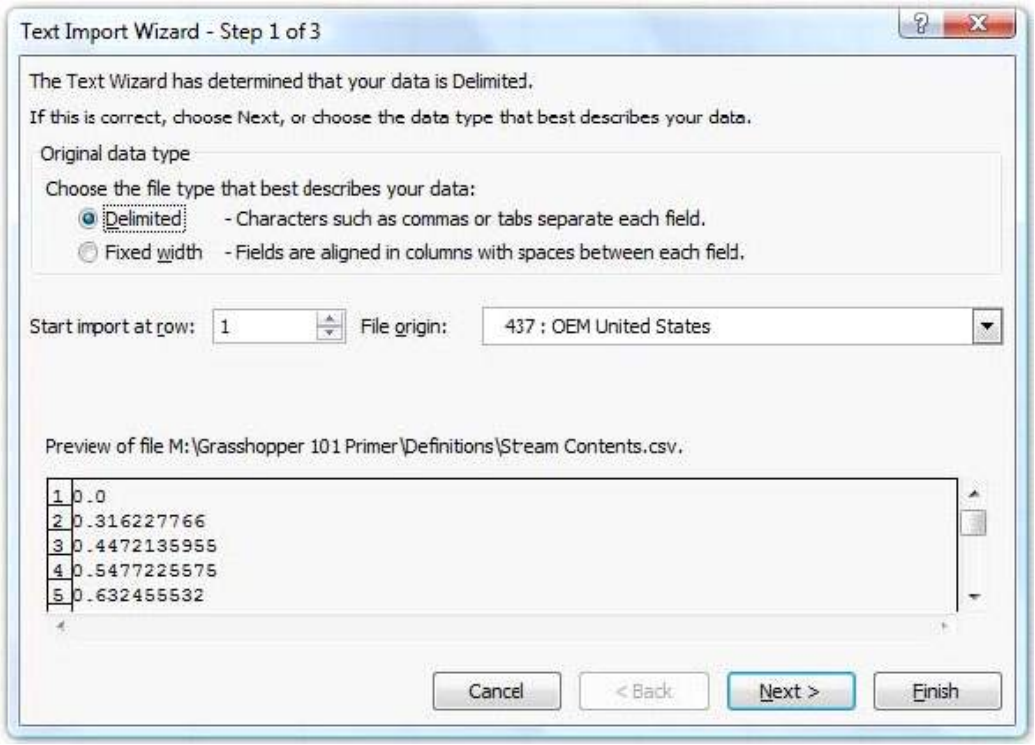
시작하기 위해서 Range 요소(Logic/ Sets/ Range)를 캔버스에 놓습니다. 0.0부터 10.0까지 숫자 도메인을 설정합니다. 결과 목록이 0.0과 10.0 사이에서 동일하게 배열된 101개의 값을 보여주기 위해서 Range-N 입력에서 오른쪽 클릭하여 100단계 숫자를 설정합니다.



다음으로 Graph Mapper 요소(Params/ Special/ Graph Mapper)를 드래그 하여 캔버스에 끌어 놓습니다. 오른쪽 클릭하여 Graph 유형을 Linear로 설정하고, Range-R 결과와 Graph Mapper 입력을 연결합니다. Post-it 요소를 캔버스 위에 드래그해서 올려놓고 Graph Mapper 결과를 Post-it Panel 입력과 연결하여 작업을 마칩니다.

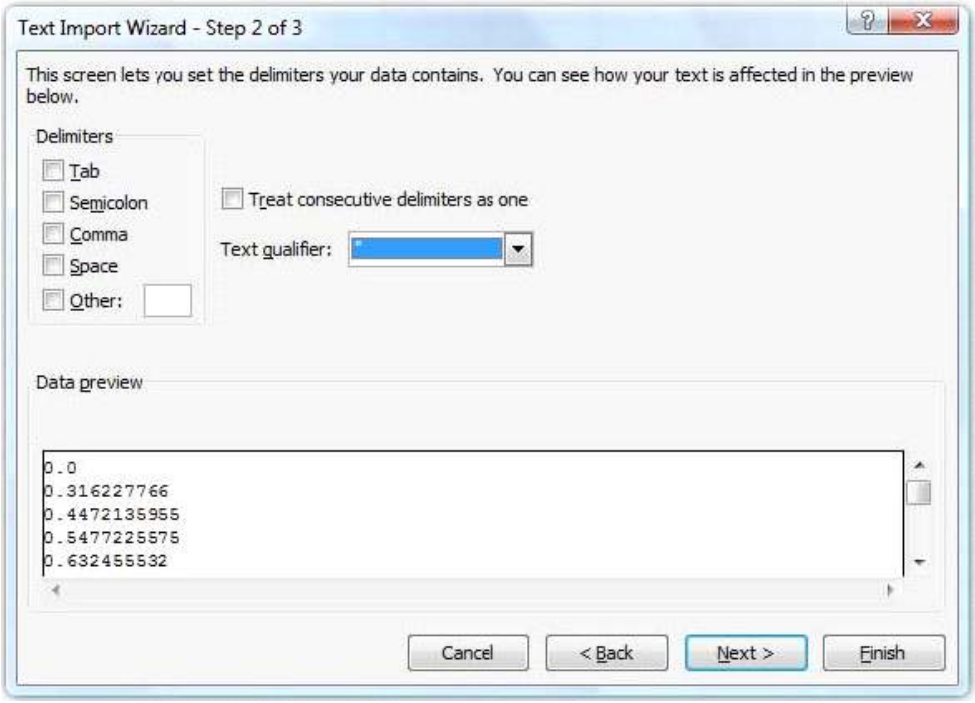


Graph Mapper 유형이 Linear로 설정되었기 때문에(Post-it Panel에서) 결과 목록은 0.0에서 10.0까지 올라가는 숫자 데이터로 선형적 모양이 배열됩니다.
하지만 만약 Graph Mapper 요소에서 오른쪽 클릭하여 Graph 유형을 Square Root로 설정하면 로그함수 데이터가 추출됩니다.

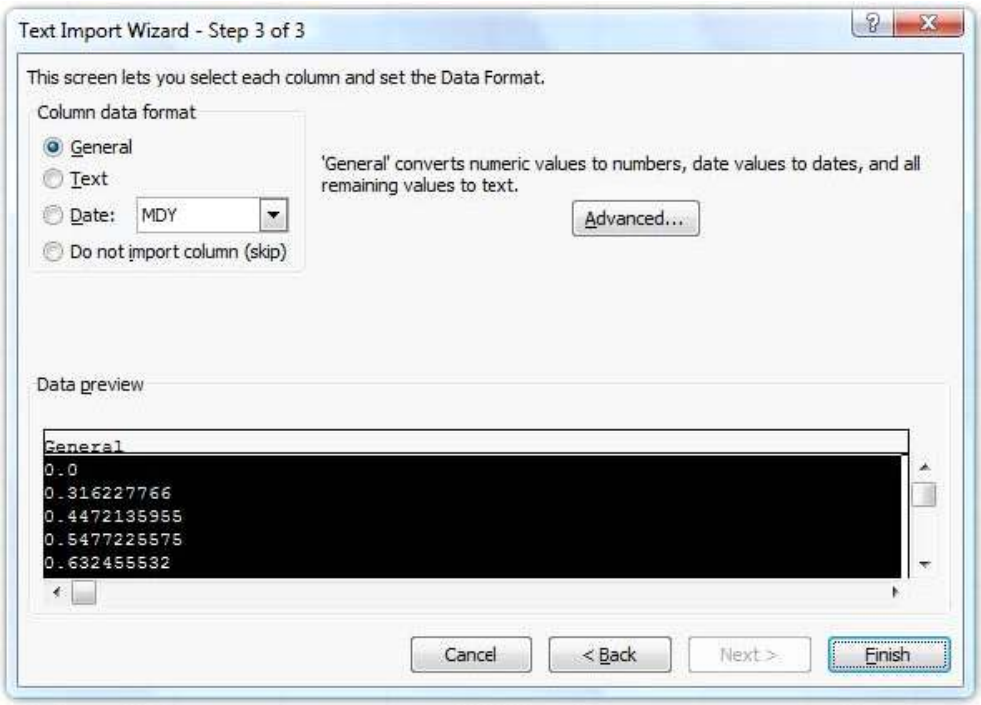


Post-it Panel에 있는 데이터 목록을 내보내기 위해 Stream Contents를 선택한 후, 패널에서 오른쪽 클릭을 합니다. 실행이 되었으면 하드드라이브에서 파일의 저장 위치를 찾습니다. 저장하기 위해 사용할 수 있는 다양한 파일 유형에는 Text File(.txt), Comma Separated 결과값s(.csv), Data File(.dat). CSV파일은 테이블 폼 안에 데이터 저장만을 위한 포맷으로 엑셀로 보내질 수 있습니다. CSV파일에 있는 각각의 라인은 테이블의 각 행과 연관이 있습니다. 라인 안의 필드들은 콤마에 의해 분리되어지고 각 필드는 테이블의 열에 속해 있습니다. 예제에서는 각 라인마다 하나의 값을 가지고 있어 멀티 행을 사용하지 않지만, 정확한 데이터를 보냄으로써 복잡한 스프레스시트를 만드는 것은 가능합니다.

이제 엑셀로 데이터를 보낼 수 있습니다. 먼저 적용을 하고 Data 탭을 선택합니다. 이 탭 아래에 있는 Get External Data from Text를 선택하고 하드디스크에 저장했던 Stream Contents.csv를 찾습니다. 데이터를 어떻게 보내고자 하는지에 대한 몇 개의 질문이 Text Import Wizard에 따라 안내 될 것입니다. Delimited 버튼이 체크되어 있는지 확인하고 Next를 클릭하여 Step2로 갑니다.

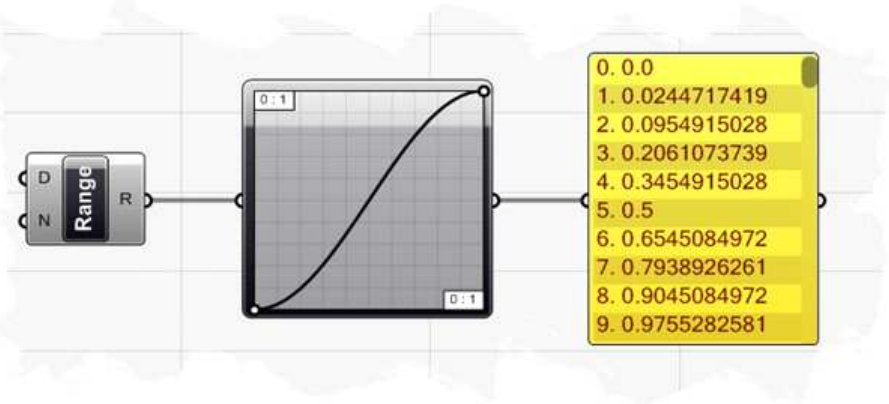


Text Import Wizard의 두번째 단계는 Delimiters의 어떤 유형이 데이터를 나눌 것인지 정의하게 해줍니다. Delimiter는 CSV 파일에 저장된 (세미콜론, 콤마 혹은 스페이스)특성입니다. 이것은 데이터가 또 다른 열로 나뉘어져야 하는 것을 가리킵니다. CSV파일의 각 라인에 저장되어 있는 숫자데이터가 있기 때문에, Delimiter의 구체적인 어떤 특성을 선택하지 않아도 됩니다. Next를 눌러 Step3으로 갑니다.



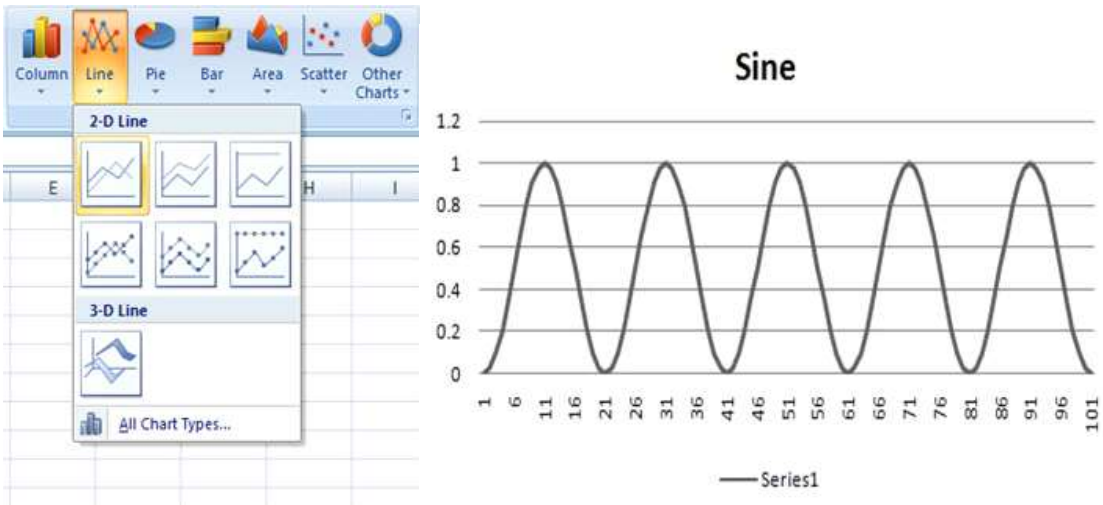
Step3에서는 삽입된 데이터를 엑셀에서 어떻게 표현할지를 설정할 수 있습니다. General 항목을 선택 시에는 모든 수치 데이터는 숫자로, 날짜 정보는 날짜형식으로 그리고 이 외의 기타 데이터는 텍스트로 변환합니다.

데이터를 Import하기 위해 첫 번째 셀을 선택하고 Default 값인 A1을 그대로 사용합니다. 이제 101개의 Grasshopper Post-it Panel의 값들이 엑셀에서 보여 지게 됩니다. Grasshopper 정의 파일은 항시 업데이트 되는 스트리밍 데이터이므로 정의 파일에 변화가 생긴다면 이는 자동적으로 CSV파일을 업데이트 합니다. Grasshopper로 돌아가서 Graph Mapper 타입을 sine으로 바꾸면 Post-it Panel의 데이터가 바뀜을 알 수 있습니다.



엑셀 데이터 탭으로 돌아가면, 모든 항목 새로 고침(refresh all)이라는 버튼을 볼 수가 있습니다. 이 버튼을 클릭하고 확인 차아에서 조금 전 엑셀에서 열었던 CSV 파일을 선택하면 Column A의 값들이 모두 업데이트 되어있음을 확인할 수 있습니다.

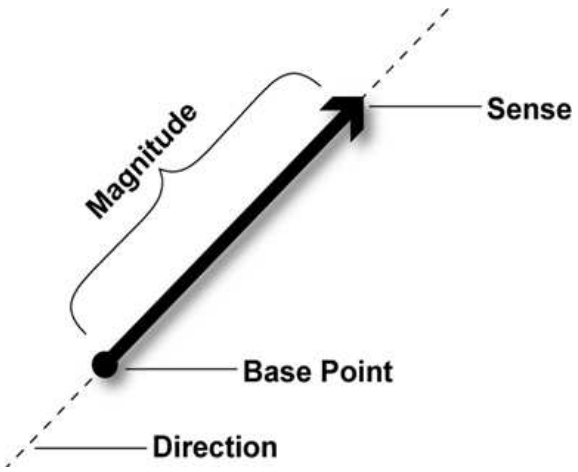
A1에서 A100까지의 셀을 모두 선택합니다. 그리고 insert 탭의 Line chart 유형을 선택하고 첫 번째 2D Line chart 아이콘을 선택합니다.



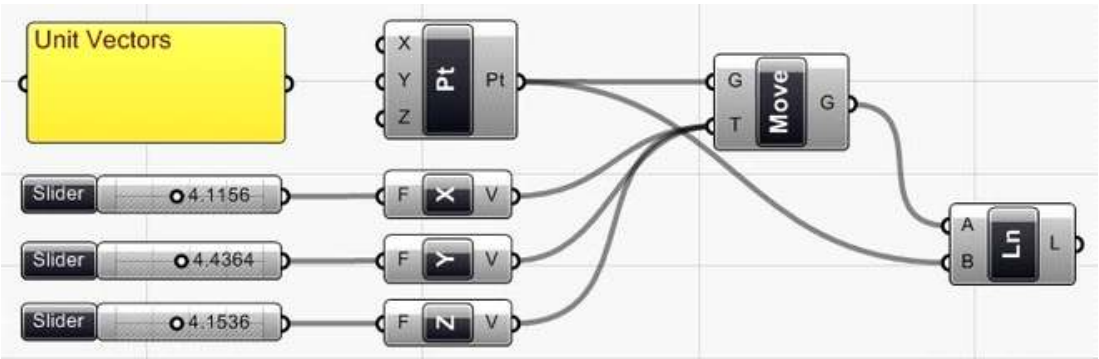
Grasshopper Mapper의 그래프 형태와 같은 형태의 그래프가 그려짐을 볼 수 있습니다. 원하는 그래프를 Grasshopper에서 설정하고 엑셀에서 Refresh하면 엑셀 그래프가 업데이트 됨을 알 수 있습니다.

9 기본 Vector

물리학에서 Vector는 크기와 방향을 가진 Object입니다. Vector는 종종 라인 혹은 화살표로 표현되는데 이는 시작점 A와 끝점 B를 가진, 크기는 라인의 길이이며 방향은 A로부터 B까지의 상대적인 위상차이입니다.

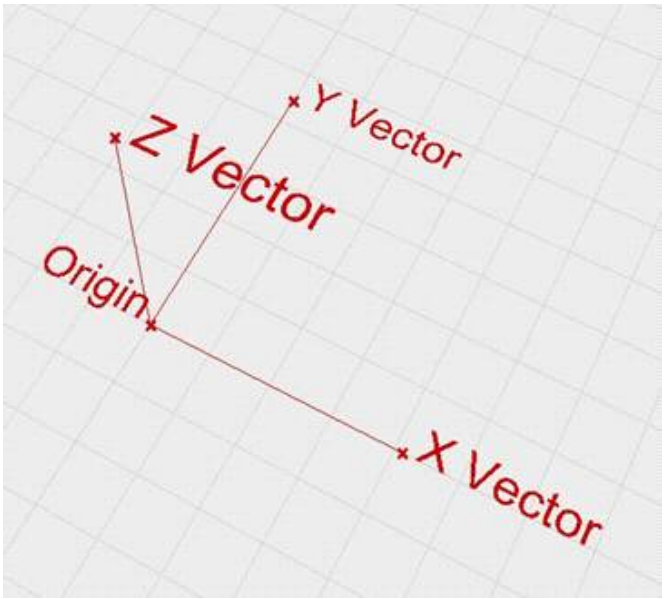


Rhino에서 Vector는 점과 구별이 모호하다. Rhino의 점과 Vector는 모두 세 개의 소수로 표현되기 때문입니다.(각각의 소수는 직교 좌표체계에서 x, y, z 값을 의미한다.)
Rhino에서 점과 Vector의 차이는 점은 절대적인 값을 가지게 되고 Vector는 상대적인 값을 가지게 됩니다. 예를 들어 세 개의 소수로 이루어진 좌표가 있다고 가정했을 때, 이 좌표의 집합을 점으로 생각한다면 이는 절대적 공간상의 위치입니다. 하지만 이 좌표를 Vector로 생각한다면 이는 시작과 끝이 정해지지 않은 공간상의 방향(원점 0, 0, 0 으로부터의)입니다. 이렇듯 Vector에는 절대성이 없고 실제적인 기하학이 아니므로 Grasshopper와 Rhino에서는 Vector를 가시화 하지는 못합니다. 하지만 Vector는 이동, 회전, 방향 등과 같은 연산 시에 사용되어 질 수 있습니다.



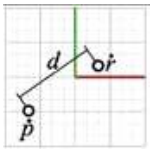
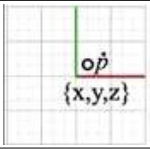
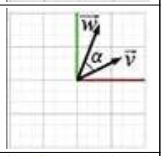
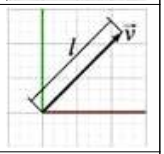
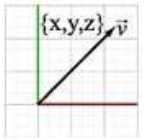
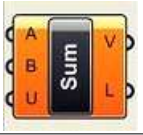
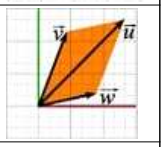
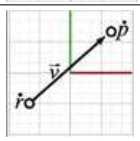
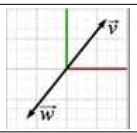
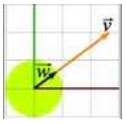
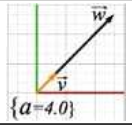
위의 예에서, 점 x, y, z 컴포넌트를 이용해 원점(0,0,0)에 하나의 점을 생성합니다. 이점을 복사 및 이동시키기 위해 점 x y z 컴포넌트를 Move 컴포넌트에 연결합니다. 그리고 이동에

필요한 Vector를 설정하기 위해 각 방향의 단위 Vector를 캔버스 위에 위치시킵니다. 단위 Vector들은 각각 x, y, z 방향의 기본 Vector들이며 그 크기는 각각 1.0입니다. 이 Vector들의 크기를 조절하기 위해 Numeric Slider를 각 기본Vector에 연결합니다. 그 다음 세 방향의 Vector를 실제로 move에 사용하기 위해 Shift를 누른 채로 각 Vector의 출력 탭을 move의 입력 탭에 연결합니다. 이제 Rhino 뷰포트 상에 첫 번째 점(0,0,0)과 세 개의 이동된 점들이 나타나게 되고, Vector를 뷰포트 상에서 보고 싶다면 Line 컴포넌트를 각각의 시작과 끝점에 연결합니다.



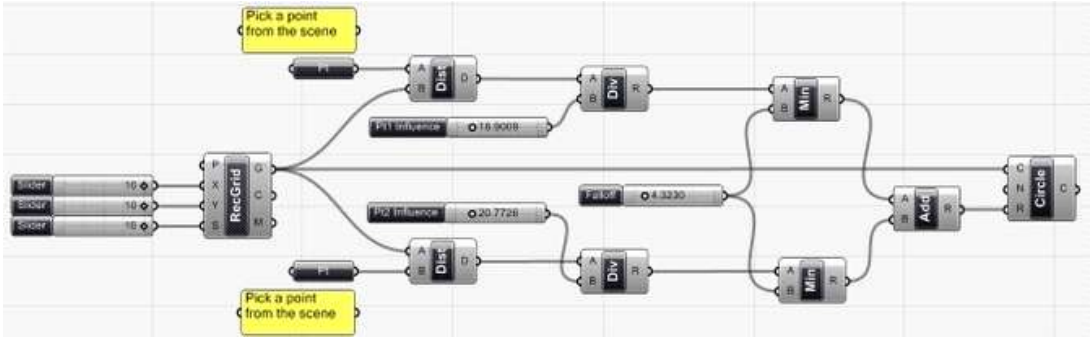
9.1 점/ Vector 정밀 조작(연산)

Grasshopper는 Vector연산을 하는 점/Vector 컴포넌트를 가지고 있습니다. 아래 테이블은 가장 많이 사용되어지는 기능입니다.

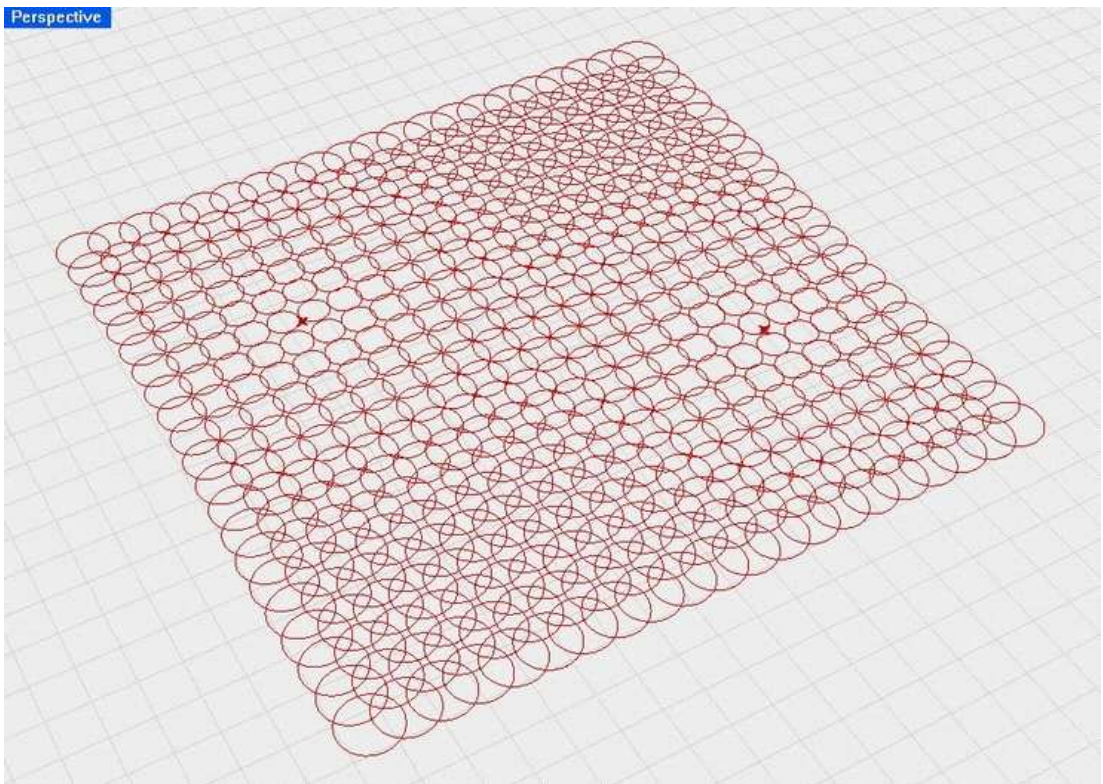
컴포넌트	Location	Descripting	Example
	Vector/ 점/ Distance	입력되어진 두 점 사이의 거리를 계산합니다.	
	Vector/ 점/ Decompose	X, Y 그리고 Z 컴포넌트에서 모아진 점을 분석(분류)합니다. 즉, 점의 집합 데이터를 X, Y, Z 값의 컴포넌트 변환합니다.	
	Vector/ Vector/ Angle	두 점 사이의 각도를 계산하여 부채각(라디안 단위 값)으로 계산되어져 출력합니다.	
	Vector/ Vector/ Length	Vector의 길이를 계산합니다.(넓이, 너비, 진폭, 사정, 출몰, 방위각)	
	Vector/ Vector/ Decompose	Vector를 그것들의 컴포넌트 부분(요소, 단편, 조각)들로 분석(분류)합니다.	
	Vector/ Vector/ Summation	Vector 컴포넌트 1(A 입력 값)과 Vector 2(B 입력 값)의 합계	
	Vector/ Vector/ Vector2pt	정의되어진 두 점으로부터 Vector를 생성합니다.	
	Vector/ Vector/ Reverse	모든 컴포넌트들을 무효(부정적인 작동)로 하고, 방향을 거꾸로 합니다.(반전) 길이는 유지(지속)합니다.	
	Vector/ Vector/ Unit Vector	Vector 길이의 역(함)수에(전도) 의하여 모든 컴포넌트들을 나누어떨어지게 합니다. Vector 결과는 Vector 유닛(단위) 1의 길이를 가집니다. 때때로 정상적인 것으로 보내집니다. 단위 Vector화.	
	Vector/ Vector/ Multiply	정해진 요소(수치)만큼 Vector의 컴포넌트들을 곱하거나 늘립니다.	

9.2 점 견인자(끌어당기는)로 수학적 Vector/Scalar(조직적인, 단계적인)를 사용하는 법

Vector와 스칼라에 대한 기본지식을 바탕으로 외부의 한 점으로부터 거리에 반응하여 크기가 변하는 원들로 이루어진 그리드 시스템의 예를 살펴봅니다.



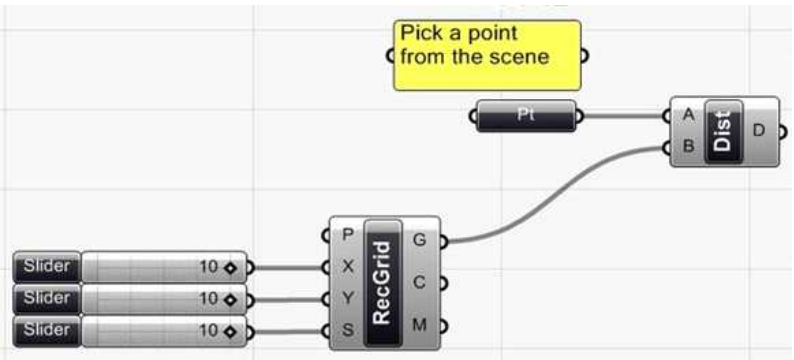
위의 회로는 아래 원들의 크기가 변화되어진 것이 일련하게 생성되는 정의로 정리되었습니다.



캔버스 3개의 숫자 Slider(Params/ Special/ Numeric Slider)를 생성하여 각각의 Slider에 오른쪽 클릭한 후, 다음과 같이 설정합니다.
·Rectangular 점 그리드 컴포넌트 생성(Vector/ 점/ 그리드 Rectangular)

·첫 번째 Slider를 Pt 그리드-X 입력 탭에 연결합니다.
·두번째 Slider를 Pt 그리드-Y 입력 탭에 연결합니다.
·세 번째 Slider를 Pt 그리드-S 입력 탭에 연결합니다.
Rectangular 점 그리드 컴포넌트는 점으로 이루어진 그리드를 생성, P-입력은 이 그리드의 원점이 되며, 본 예제에서는 0,0,0을 원점으로 사용합니다. 이 그리드는 컴포넌트는 x 및 y 숫자 Slider에 의해 정의된 만큼의 점을 각각 x, y방향으로 생성합니다. 그러나 본 예제에서 x, y값을 10으로 설정했음에도 불구하고 각 방향으로 10개 이상의 점이 생성이 되는데 이는 원점을 중심으로 x, y모두 +/-방향으로 10만큼 확장하기 때문입니다. 즉, x, y방향의 수치가 두 배가 되는 점을 얻게 됩니다. S-입력은 점 사이의 거리를 설정합니다.

이 컴포넌트는 전체 시스템의 입력으로써 휘발성이 아닌 영구적인 데이터입니다. 영구적인 데이터 타입(Persistent data 유형)에 대해서는 3장을 참조합니다. 이 컴포넌트는 Rhino 객체로부터 연결 혹은 정의되기 전까지는 점을 생성하지 않습니다. 즉, Rhino에서 점을 생성하고 이를 컴포넌트에 연결해야 합니다. 이 점이 Attractor 점으로 사용되어질 것입니다.
·Rhino에서 임의의 위치 점을 생성합니다. 본 예제의 경우는 top 뷰에서 점을 생성하여 x, y 평면상에서 점을 위치시킵니다. 그 다음 점 attractor를 Grasshopper 점 컴포넌트에 연결합니다.
·점 컴포넌트상에 오른쪽 클릭 후, Set One을 선택합니다.
·Rhino에서 생성한 점을 선택합니다.
이제 포인트 그리드와 Attractor 점을 모두 생성합니다. 다음 단계로 Attractor 점과 각 그리드 상점의 거리를 측정하기 위해 Vector연산을 사용할 것입니다.



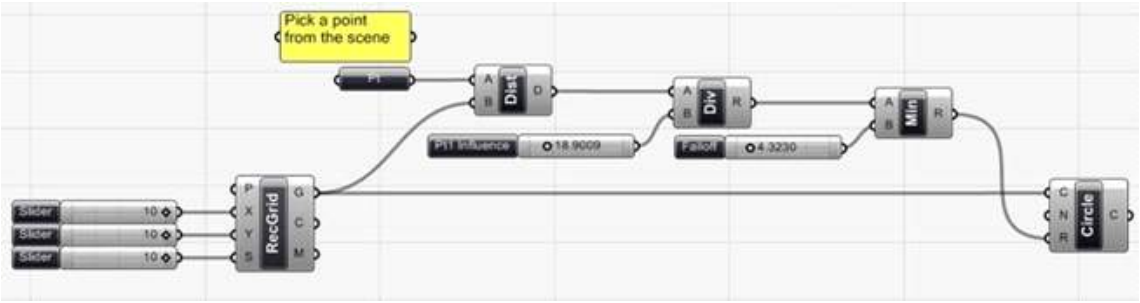
·Distance 컴포넌트 생성(Vector/ Point/ Distance)합니다.
·Attractor 점 출력을 Distance-A 입력에 연결합니다.
·Rectangular 그리드 점-G 출력을 Distance-B 입력에 연결합니다.

첫 번째 단계는 위 그림과 같습니다. 마우스를 Distance-D 출력 탭에 올리면 Attractor Point로부터 그리드 상의 각점까지의 거리에 해당하는 숫자의 리스트가 표시됩니다. 이 거리 값들은 각 점 위에 놓일 원의 반지름을 결정하게 됩니다. 거리 값들이 원들의 반지름이 되기에는 조금 크므로 이 값들을 줄여야 합니다.
·Division operator를 생성합니다.(Scalar/ Operators/ Division)

·Distance-D 출력을 Division-A 입력에 연결합니다.
·Numeric Slider를 생성합니다.(Params/ Special/ Numeric Slider)
·오른쪽 클릭 후, 다음과 같이 설정합니다.
o 이름: PT1 Influence
o Slider 유형: Floating 점
o 최소 값: 0,0
o 최대 값: 100,0
o 결과 값:25,0
·Pt1 Influence Slider를 Division-B 입력에 연결합니다. 거리 값들이 반지름으로 쓰기엔 큰 값이므로, Scale factor를 사용하여 좀 더 작은 값으로 변환합니다. 숫자 Slider를 나누는 값으로 설정합니다. 이 값을 Circle 컴포넌트의 반지름 값으로 입력해도 무관하지만, 조금 더 완성도 있게 만들기 위해 Falloff 거리 값(최소 값)을 정하는 또 다른 Math 컴포넌트를 사용하기로 합니다. 즉, Attractor Point와 포인트 그리드의 거리가 너무 멀어질 경우, 원의 반지름은 그에 비례하여 커지게 되는데, 이 반지름은 포인트 그리드 사의 각 점의 거리에 비해 너무 커지지 않게 조절이 필요합니다.

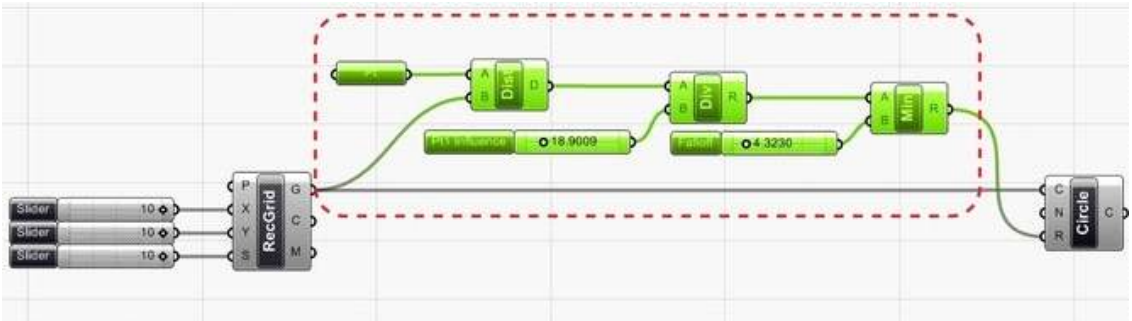
·Minimum 컴포넌트를 생성합니다.(Scalar/ Utility/ Minimum)
·Division-R 출력을 Minimum-A 입력에 연결합니다.
·Numeric Slider를 생성합니다.(Params/ Special/ Numeric Slider)
·오른쪽 클릭 후, 다음과 같이 설정합니다.
o 이름: Falloff
o Slider 유형: Floating 점
o 최소 값: 0,0
o 최대 값: 30,0
o 결과 값: 5,0
·Falloff Slider를 Minimum-B 입력에 연결합니다.
·Circle CNR(Center, Normal, Radius)을 생성합니다.(Curve/ Primitive/ Circle CNR)
각 원의 중심점이 이전 단계에서 만든 포인트 그리드 상의 각 점에 위치하게 만듭니다.
·Rectangular Pint 그리드-G 출력을 Circle-C 입력에 연결합니다.
·Minimum-R 출력을 Circle-R 입력에 연결합니다.
·Rectangular Point 그리드 컴포넌트를 오른쪽 클릭 후, Preview를 꺼줍니다.

Grasshopper 정의 파일은 위와 같습니다. 일련의 원들은 Attractor Point로부터 거리에 의해 스케일이 조정되었습니다.



·추가적인 Attractor 점 설정을 합니다.
기존의 정의 중 일부를 복사하여 붙여 넣음으로서 추가적인 Attractor 점을 설정합니다.
·다음의 컴포넌트들을 선택합니다.; Attractor 점 Parameter, Distance 컴포넌트, Pt 1 Influence Slider, Division 컴포넌트, Falloff Slider, Minimum 컴포넌트, 그 다음 복사하여 붙여 넣습니다.

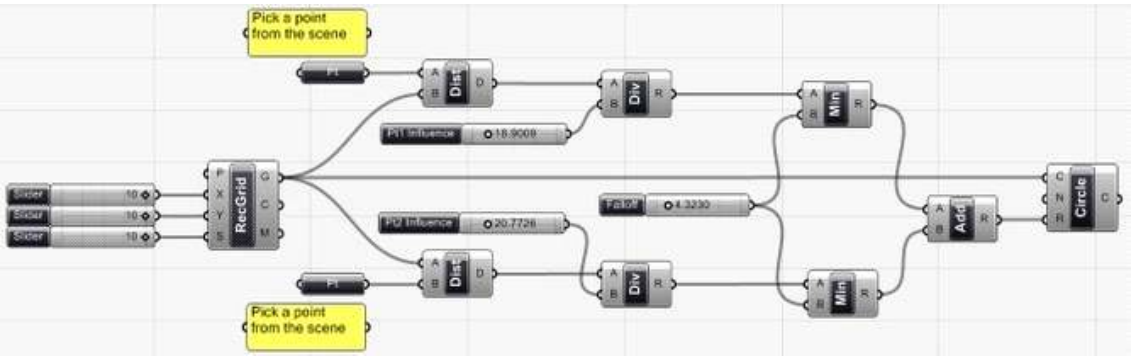
복사된 컴포넌트들을 캔버스 상에서 겹치지 않도록 위치시킵니다.
또 다른 필요의 Attractor 점은 Grasshopper의 외부에서 정의 되어야 합니다. 즉, Rhino 상에서 점을 만들고 이를 Grasshopper에 연결합니다.



·Rhino 뷰포트(top) 상의 임의의 점에 점을 생성하여 그 점이 xy평면상에 위치하도록 합니다.
·복사된 점 params에 오른쪽 클릭하여 Set One 점을 클릭합니다.

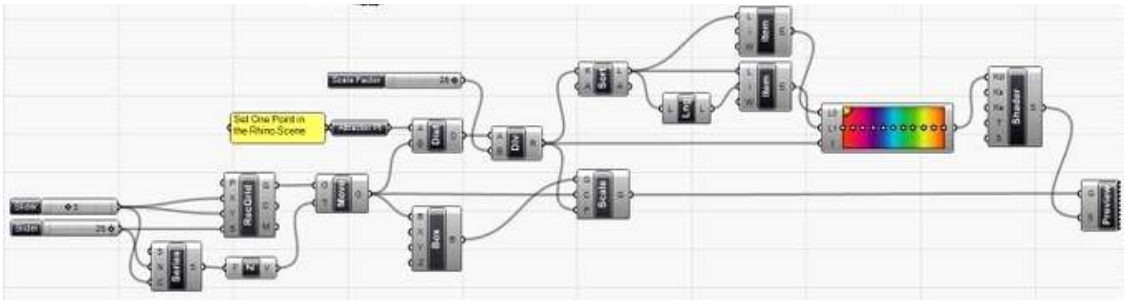
·방금 생성한 Rhino 점 object를 선택하여 연결합니다.
이제 두 개의 Attractor 점으로부터 점 그리드까지의 거리를 측정하여 원의 반지름을 설정할 수 있는 컴포넌트들이 완성됩니다. Scalar addition 컴포넌트를 사용하여 이 두 개의 거리 값 리스트를 하나로 합칩니다.
·Addition 컴포넌트를 생성합니다.(Scalar/ Operators/ Addition)
·첫 번째 Minimum-R 출력을 Addition-A 입력에 연결합니다.
·두번째 Minimum-R 출력을 Addition-B 입력에 연결합니다.
·Addition-R을 Circle-R 입력에 연결합니다.(*기존의 연결에 추가되는 것이 아니라, 기존의 연결을 대체하는 것입니다.)

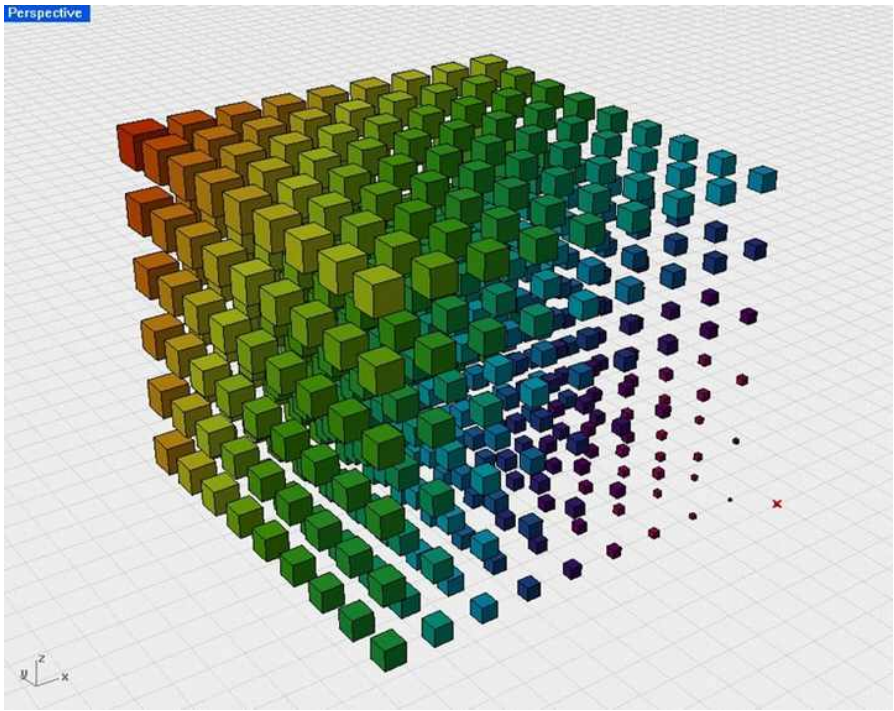
제대로 연결되었다면 정의 파일은 위 그림과 같습니다. Falloff Slider 중 하나는 삭제되고 두 개의 변수를 하나의 Slider로 함께 조절합니다. Pt Influence와 Falloff Slider를 움직이며 원의 크기가 어떻게 변하는지 살펴봅니다.



9.3 견인자(박스들의 크기조작)로 수학적 Vector/ 스칼라를 사용하는 방법

이전 예제를 통해 특정한 점으로부터의 거리를 이용하여 원의 크기를 결정하는 작업을 해보았습니다. 이번에는 같은 원리를 사용해서 Object들의 크기를 결정하고 Grasshopper의 Shader 컴포넌트를 이용하여 Object의 색깔도 결정합니다.



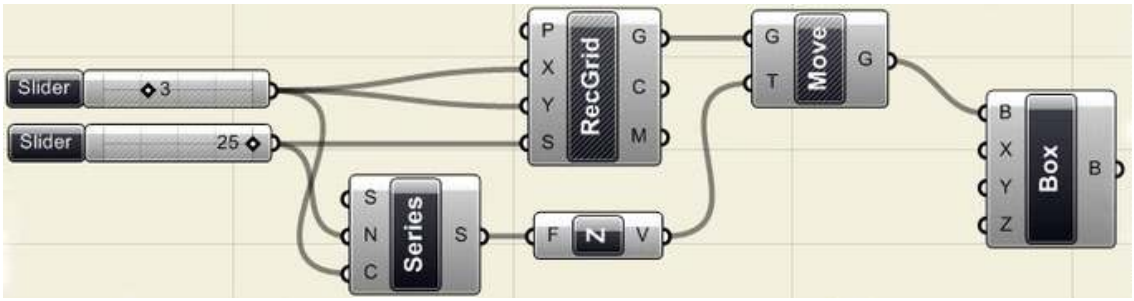


·Step 1: 삼차원의 포인트 그리드를 생성합니다.
·2개의 Slider를 생성합니다.(Params/ Special/ Numeric Slider)
·첫 번째 Slider를 오른쪽 클릭한 후, 다음과 같이 설정합니다.
o Slider 유형: Integer
o 최소 값: 0,0
o 최대 값: 10,0
o 결과 값: 3,0
·두번째 Slider를 오른쪽 클릭한 후 다음과 같이 설정합니다.
o Slider 유형: Integers

o 최소 값: 0,0
o 최대 값:25,0
o 결과 값: 25,0
·Rectangular 점 그리드 컴포넌트를 생성합니다.(Vector/ Point/ Grid Rectangular)
·첫 번째 Slider를 점 그리드의 X와 Y 입력에 연결합니다.
·두번째 Slider를 점 그리드-S 입력에 연결합니다.
첫 번째 Slider는 x와 방향의 점의 개수를 조절하며(Slider 숫자의 두 배에 해당하는 점이 생성됩니다.) 두 번째 Slider는 점과 점 사이의 거리를 조절합니다. 삼차원의 포인트 그리드를 생성하기 위해 이 포인트 그리드를 z축으로 복사합니다.

·Series 컴포넌트를 생성합니다.(Logic/ Sets/ Series)
·두번째 Slider를 Series-N 입력에 연결합니다.
·첫 번째 Slider를 Series-C 입력에 연결합니다.
Series 컴포넌트는 z방향 복사의 개수를 결정하게 됩니다. 하지만 이 Series 컴포넌트는 z축 방향의 양의 방향으로만 복사를 진행하게 됩니다. 이미 생성된 포인트 그리드는 원점을 중심으로 x, y축의 양과 음 방향으로 확장되므로 Slider를 3으로 설정했다고 해도 실제로는 7개의 점을 얻게 되는 것입니다. 궁극적으로 정육면체의 3차원 포인트 그리드를 얻고자하므로 z방향으로도 7개의 점을 가져야 합니다. 이를 위해 Series 컴포넌트의 count 입력 탭에 expression을 수정합니다.
·Series-C 입력을 오른쪽 클릭한 후, Expression 탭을 클릭합니다.
·Expression editor에 (C*2)+1을 입력합니다.
이는 Series-C 입력에 입력되는 값을 2배로 만든 후, 1을 더하게 합니다. 즉, 2가 입력이 되면 7이 출력이 되고 3이라는 숫자대신 7이 count 탭에 입력됩니다.
·Unit Z Vector 컴포넌트를 생성합니다.(Vector/ Constants/ Unit Z)
·Series-S 출력을 Unit Z-F 입력에 연결합니다.
Unit Z-V 출력 탭에 마우스를 올리면 7개의 값이 각각 25만큼의 차이, 혹은 두 번째 Slider에서 정의된 값만큼의 차이를 가지며 정의되어 있음을 알 수 있습니다. 이 값의 리스트를 이용하여 x, y평면상의 2차원 포인트 그리드를 3차원의 형태로 증식시킵니다.
-Move 컴포넌트를 생성합니다.(X Form/ Eucildean/ Move)
-점 그리드-G 출력을 Move-G 입력에 연결합니다.
-Unit Z-V 출력을 Move-T 입력에 연결합니다.
Rhino 뷰포트를 보면 두 개의 면 사이를 연결하는 램프와 같은 형태가 보입니다. 이는 데이터 매칭 알고리즘이 기본적으로 Longest list로 설정이 되어 있기 때문입니다. 이 알고리즘을 Cross reference로 바꾸어주면 삼차원으로 확장된 포인트 그리드를 볼 수 있습니다.(데이터매칭: 6장 참고)
-Center Box 컴포넌트를 생성합니다.(Surface/ Primitive/ Center Box)
-Move-G 출력을 Center Box-B 입력에 연결합니다.
-점 그리드와 Move 컴포넌트를 오른쪽 클릭하여 Preview를 꺼줍니다.

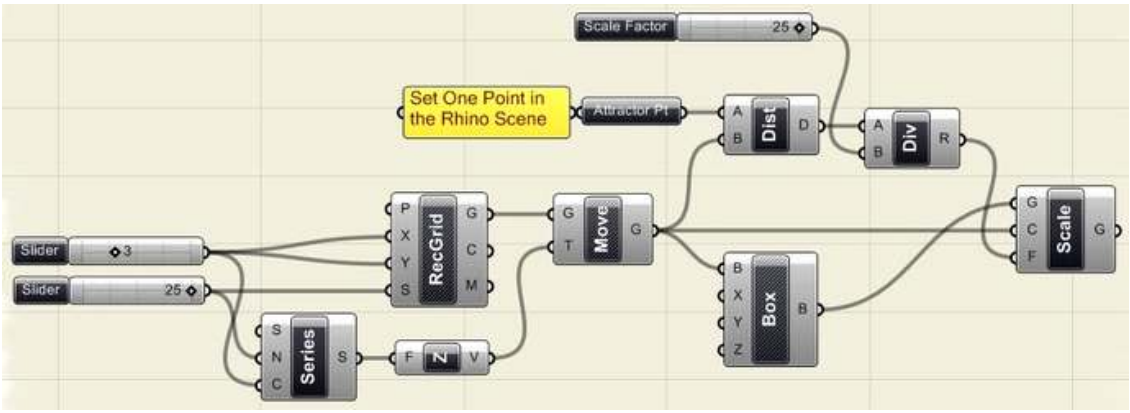
지금까지의 정의 파일입니다.



·Step 2: 스칼라/ Vector 연산
·점 컴포넌트를 생성합니다.
·점 컴포넌트에 오른쪽 클릭 후, Attractor Pt로 이름을 설정합니다.
이전 예제와 같이 Rhino 상에 포인트를 생성한 후, 이를 Grasshopper 점 컴포넌트에 연결합니다. 이를 위해 먼저 Rhino 상에서 점을 생성합니다.
·Rhino에서 삼차원 상의 임의의 위치에 점을 생성합니다.
·Grasshopper에서 생성한 점을 선택하여 연결합니다.
붉은 x마크의 점은 Rhino 점과 Grasshopper의 점이 연결되었음을 보여줍니다. Rhino 상의 점이 공간상에서 이동하면 Grasshopper의 점(붉은 x마크) 또한 따라 움직이며 업데이트 됩니다.
·Distance 컴포넌트를 생성합니다.(Vector/ 점/ Distance)
·Attractor Pt 출력을 Distance-A 입력에 연결합니다.
·Move-G 출력을 Distance-B 입력에 연결합니다.
마우스를 Distance-D 출력에 올리면 Attractor Point로부터 삼차원 그리드 상의 점까지의 거리 값의 리스트를 볼 수 있습니다. 이 값들을 Scale factor로 사용하기 전에 값들을 스케일을 낮출 필요가 있습니다.(그렇지 않을 경우, 값들이 너무 커서 상자들이 서로 오버랩 됩니다.)

·캔버스에 Distance 컴포넌트를 생성합니다.(Scalar/ Operators/ Division)
·Distance-D 출력을 Division-A 입력에 연결합니다.
·Slider를 생성합니다.(Params/ Special/ Numeric Slider)
·Slider를 오른쪽 클릭한 후, 다음과 같이 설정합니다.
o 이름: Scale Factor
o Slider 유형: Integers
o 최소 값: 0,0
o 최대 값: 25,0
o 결과 값: 25,0
·Scale Factor Slider를 Division-B 입력에 연결합니다.
·Scale 컴포넌트를 생성합니다.(X Form/ Affine/ Scale)
·Center Box-B 출력을 Scale-G 입력에 연결합니다.
·Division-R 출력을 Scale-F 입력에 연결합니다.
·Center Box 컴포넌트를 오른쪽 클릭한 후, Preview를 끕니다.
지금까지의 정의 파일은 아래의 그림과 같습니다. Rhino 뷰포트 상의 상자들은 Attractor 점

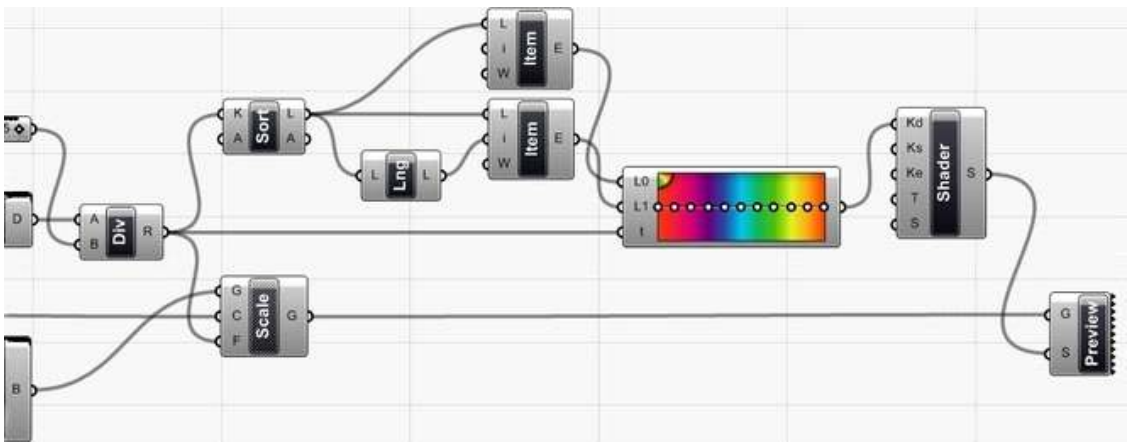
으로부터의 거리에 의해 크기가 조정되어 있음을 볼 수 있습니다. Scale Factor의 시각화를 위해 각 상자들에 색을 적용합니다.



·Step 3: 박스에 색깔 지정하기
·Sort List 컴포넌트를 생성합니다.(Logic/ List/ Sort List)
거리 값에 의해 변화되는 색 값을 적용하기 위해서 가장 가까운 거리 및 가장 먼 거리 값을 알아냅니다. 이를 위해서 거리 값들을 크기순으로 정렬하고 처음 값 및 마지막 값을 찾습니다.
·List Item 컴포넌트를 생성합니다.(Logic/ List/ List Item)
·Sort List-L 출력을 List Item-L 입력에 연결합니다.
·List Item-i 입력을 오른쪽 클릭한 후, Integer 결과 값 값을 0,0으로 설정합니다.
이는 가장 작은 거리 값이 되는 첫 번째 아이템을 전체 리스트로부터 추출합니다.
·List Length 컴포넌트를 생성합니다.
·Sort List-L 출력을 List Length-L 입력에 연결합니다.
·List Length-L 출력을 두 번째 List Item-L 입력에 연결합니다.
마우스를 List Item-E 출력에 가져가면 마지막 항목이 추출되지 않음을 알 수 있습니다. 이는 Grasshopper가 리스트 상의 첫 아이템에 0이라는 순서를 부여하기 때문입니다. 예를 들어 100개의 값이 리스트 상에 존재한다고 하면 첫 번째 아이템은 0이라는 순서 값을 가질 것이고, 마지막 아이템은 100이 아닌 99라는 순서 값을 가지게 됩니다. 이를 수정하기 위해서는 두 번째 List Item-i 입력의 Expression을 수정해야 합니다. 즉, 두 번째 List Item-i 입력에 입력되는 값에서 1을 뺀 값이 최종적인 입력 값이 되도록 수정합니다.
·두번째 List Item-i 입력을 오른쪽 클릭한 후, Expression Editor를 선택합니다.
·다음의 수식을 입력합니다.:i-1
이제 마우스를 List Item-E 출력에 가져가면 가장 먼 거리 값에 해당하는 값이 추출됨을 볼 수 있습니다.

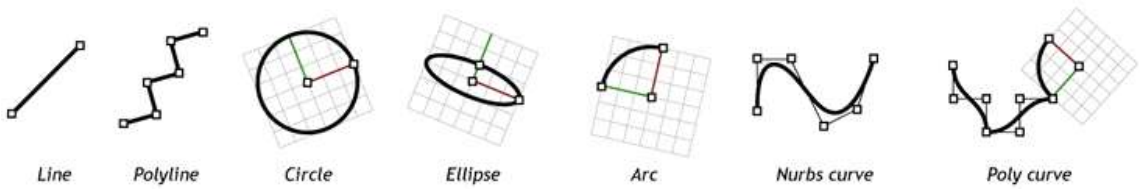
·Gradient 컴포넌트를 생성합니다.(Params/ Special/ Gradient)
·첫 번째 List Item-E 출력(가장 작은 거리 값에 해당하는 값)을 Gradient-L0 입력에 연결합니다.
·두번째 List Item-E 출력(가장 큰 거리 값에 해당하는 값)을 Gradient-L1 입력에 연결합니다.

·Division-R 출력을 Gradient-t 입력에 연결합니다.
L0 입력은 Gradient 왼쪽 편이 표현하게 될 아이템의 번호를 정의합니다. 본 예제에서는 Gradient의 왼쪽 편은 Attractor Point와 가장 가까운 박스까지의 거리, 즉, 첫 번째 아이템의 번호(0)가 되어야 합니다. 반대로 L1 입력은 Gradient의 오른쪽 편이 표현하게 될 아이템의 번호를 의미하며 이는 Attractor Point와 가장 먼 박스까지의 거리, 즉, 마지막 아이템의 번호(99)를 의미합니다.
Gradient 컴포넌트의 t-입력 결과 값은 Gradient를 통하여 표현하고자 하는 값들의 범위, 즉, 거리 값 리스트 그 자체입니다. 리스트 상의 각 값들은 이제 Gradient 상의 색 값과 연결됩니다.
·Create Shader 컴포넌트를 생성합니다.(Vector/ Color/ Create Shader)
·Gradient 출력을 Shader-Kd 입력에 연결합니다.
Shader 컴포넌트는 몇 개의 입력 탭을 가지고 있습니다. 아래는 각 입력 탭의 개략적인 설명입니다.
Kd: Diffuse Color를 설정합니다. Object의 바탕이 되는 색으로써 각 RGB 값(0-255)에 의해 정의됩니다.
Ks: Speculer Highlight를 설정하며 마찬가지로 RGB 값에 의해 정의됩니다.
Ke: Shader's self illumination 색상을 설정합니다.
T: 투명도를 설정합니다.
S: 재질의 발광을 설정합니다.
Gradient 패턴에 의해 각각의 상자들이 바탕색을 가질 수 있도록 Gradient Slider를 Diffuse 입력에 연결합니다. Slider 사이의 흰색 점을 이동시키거나 Gradient의 패턴을 바꾸고 색상을 바꿈으로써 색상에 변화를 가져올 수 있습니다.
본 예제에서는 Spectrum 패턴을 사용합니다.
·Custom Preview 컴포넌트를 생성합니다.(Params/ Special/ Custom Preview)
·Scale-G 출력을 Custom Preview-G 입력에 연결합니다.
·Shader-S 출력을 Custom Preview-S 입력에 연결합니다.
·Scale 컴포넌트를 오른쪽 클릭한 후, Preview를 끕니다.
지금까지의 과정을 통하여 아래의 그림과 같이 Attractor Point를 Rhino 뷰포트 상에서 움직이면 각 상자들의 색과 크기가 변화하는 것을 알 수 있습니다.



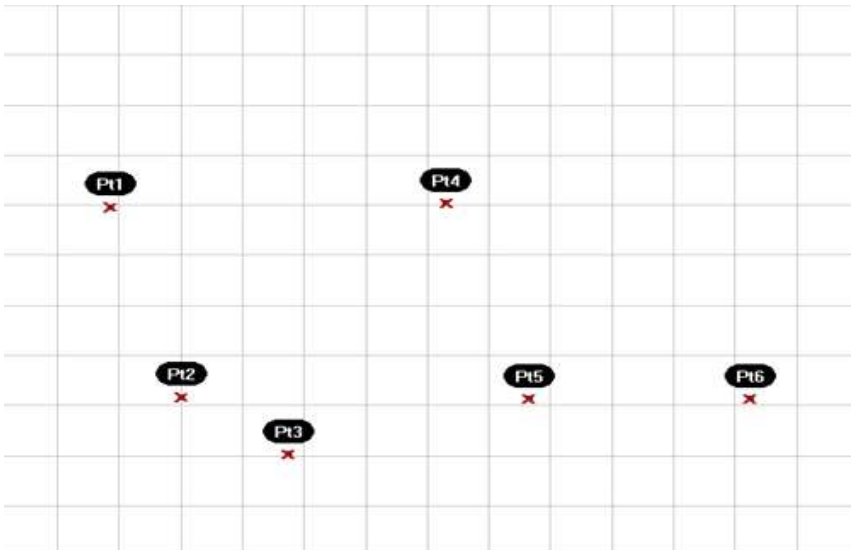
10 *Cueve의 종류*

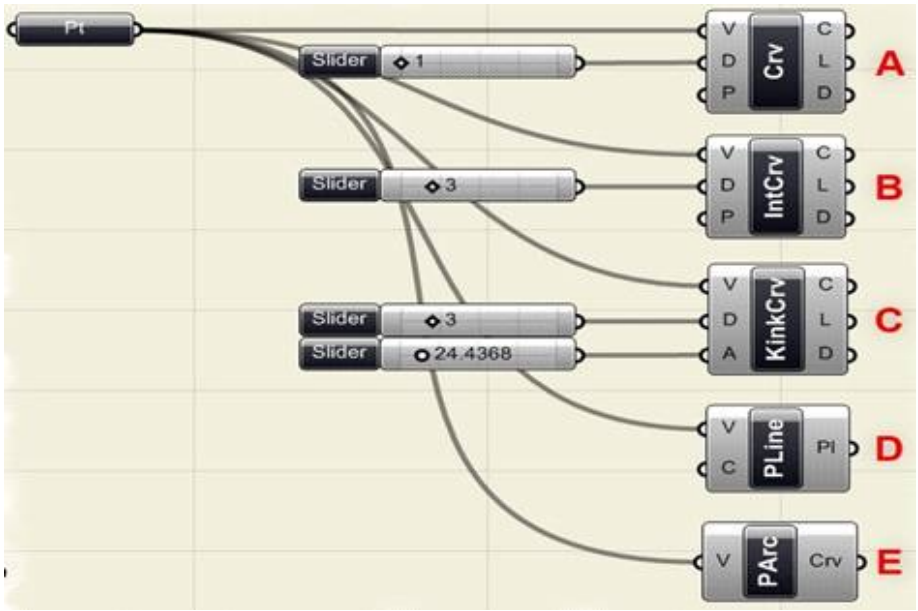
Curve는 기하학적인 Object이므로 그러한 기하학적 특성을 표현하는 값들을 가지게 되는데, 이러한 값들은 Curve를 생성하거나 분석되는데 사용됩니다. 예를 들어 모든 Curve는 시작점과 끝점의 위치를 가지고 있으며, 만약, 이 두 점간의 거리가 0이라면 즉, 시작과 끝이 같은 것입니다. 시작과 끝이 같다면 Curve는 닫힌 Curve가 됩니다. 또한, 모든 Curve는 몇 개의 Control Point를 가지게 되는데 모든 Control 점과 3차원 상에서 같은 평면에 위치하게 된다면 이 Curve는 평면의 Curve가 됩니다.
Curve의 어떤 특성은 Curve 전체에 일정하게 적용되기도 하지만 모든 특성이 Curve 전체를 통해 동일한 것은 아닙니다. 다시 말해 평면성은 특정한 Curve에 있어서는 동일하지만, 법선 Vector라는 특성은 동일한 Curve 내에서도 다른 값들을 가질 수가 있습니다.
또한, 어떤 특성 값은 특정한 Curve에서만 찾아볼 수 있습니다.
지금까지 Lines, Circle, Ellipses, Arcs 와 같은 Grasshopper의 기본적인 컴포넌트에 대해 이야기했습니다.



Grasshopper는 또한 Rhino의 Curve 유형(Nurbs Curve/ Poly Curve)을 표현하기 위한 툴을 갖고 있습니다. 그 중 Spline 컴포넌트를 사용하기 위해 먼저 일련의 점들이 필요합니다.

6개의 점이 x, y 평면상에 놓여 있습니다. 각 점은 좌에서 우로 이름이 적혀있고 이름순서는 Grasshopper 연결하는 순서와 같습니다.

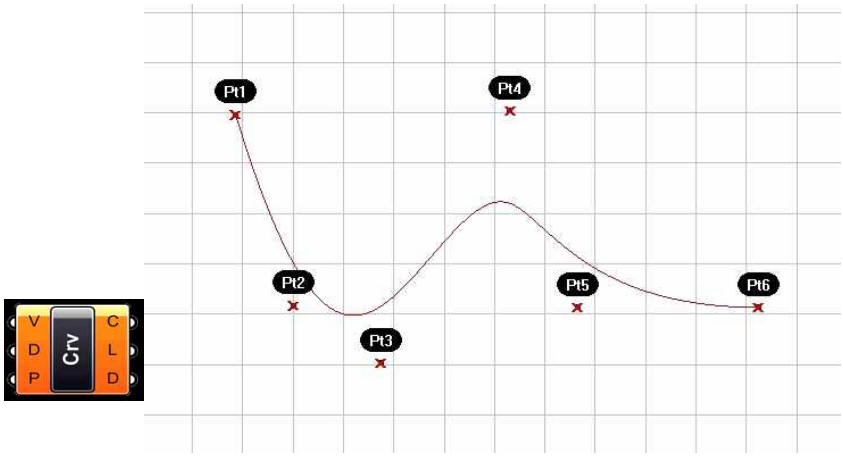




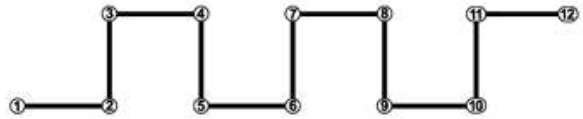
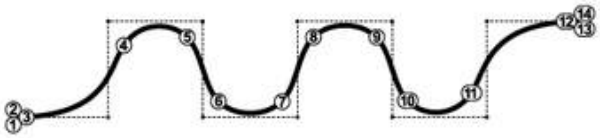
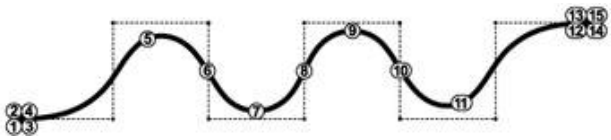
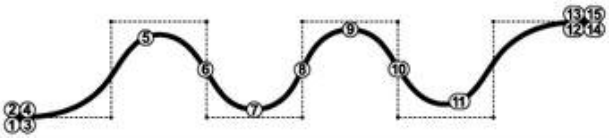
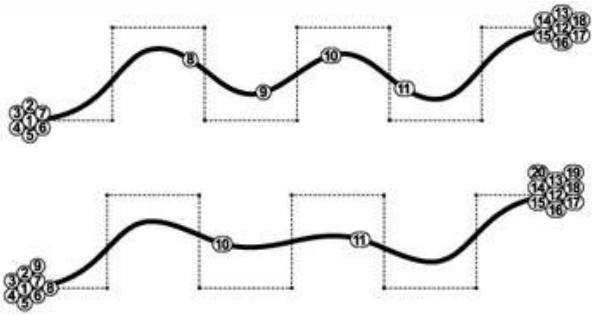
Point 컴포넌트가 여러 개의 Curve 컴포넌트와 연결되어 있습니다. 각각의 Curve 컴포넌트는 각기 다른 형태의 Curve를 생성합니다. 우선 Grasshopper 상의 Point 컴포넌트에 Rhino 상의 점들을 연결합니다.
이를 위해, Point 컴포넌트를 오른쪽 클릭한 후, Set Multiple Point를 선택하여 Rhino 상의 6개의 점을 이름의 순서대로 선택합니다. 선택하는 순서에 의해 파란색 연결선이 보일 것이고, 모두 선택하였으면 엔터를 쳐서 Grasshopper로 돌아옵니다. 6개의 점들은 이제 각각의 위에 붉은 x마크로 표현되어져 각 점들이 Grasshopper에 연결되었음을 보여줍니다.

A) Nurbs Curve(Curve/ Spline/ Curve)

Non-Uniform Rational Basic Spline 또는 Nurbs Curve는 Rhino에서 구현 가능한 여러 가지 Curve 중 하나입니다. Control Point 외에도(6개의 점들) Nurbs Curve는 Degree(등급), Knot-Vector(매듭, 장식-Vector), Weight(무게) 등과 같은 특성이 있습니다. Nurbs의 이론에 대해서는 많은 책들과 논물들이 발표되어 있으나, 그러한 수학은 논외로 합니다.



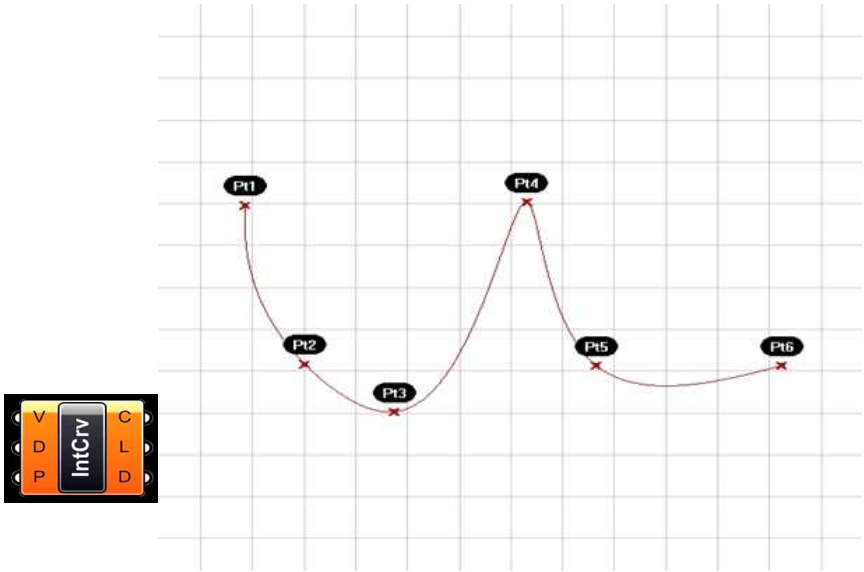
Nurbs Curve-V 입력은 컨트롤 포인트를 정의합니다. Rhino 상의 점을 직접 선택하여 입력할 수도 있고, Grasshopper 상의 다른 컴포넌트로부터 넘겨받은 값을 입력할 수도 있습니다. Nurbs Curve-D 입력은 Curve의 범위를 결정합니다. Curve의 범위는 1과 11을 포함한 1에서 11까지의 정수입니다. Curve의 범위는 각각의 컨트롤 포인트가 Curve에 미치는 영향력의 범위를 설정합니다. 범위가 클수록 영향의 범위 역시 커집니다.
다음 테이블은 David Rutten의 Rhino스크립트 101에서 발췌하였으며, 범위에 따른 허브의 형태 변화를 살펴볼 수 있습니다.

범위 변화 값에 따른 NurbsCurve Vector	
	**범위-1** Curve는 폴리라인과 동일하게 작동. 컨트롤 포인트의 수와 동일한 결절 점 컨트롤 포인트와 결절점이 1대1 대응
	**범위-2** 흔하지 않고 결절 점들은 직관적 위치. 포리곤의 각 세그먼트의 중간점에서 컨트롤 폴리곤과 만납니다. 아크나 원 표현에 사용.
	**범위-3** 가장 흔하고 Rhino NurbsCurve의 기본 값. 결절 점의 위치가 납득하기 힘든 위치이지만, 전체적 Curve와 컨트롤 폴리곤의 배열은 가장 익숙한 형태.
	**범위-4** Rhino 상에서 기술적으로는 가능하지만 짝수 범위를 가지는 Nurbs Curve는 작동 오류가 있습니다. 보편적으로 홀수 범위가 선호.
	**범위-5** 흔하고 범위-3과 같이 자연스럽고 부드러운 모양. 범위가 높으므로 점들이 Curve형태에 미치는 영향력의 범위가 더 큼.(부드러운)
	**범위-7~9** 이론적인 범위 Curve입니다. Rhino는 범위-11까지 올라갈 수 있으나 컨트롤 포인트들이 Curve에 미치는 영향력의 범위가 너무 커서 즉, 부드러운 Curve를 생성하므로 컨트롤 폴리곤과의 연계성이 너무 작아져 실제적인 모델링에 유용하지 않습니다.

예제에서 숫자 Slider를 Curve-D 입력에 연결하여 Nurbs Curve의 범위를 결정하였습니다. Slider를 좌우로 움직이면 각 컨트롤 포인트가 Curve에 미치는 영향력을 살펴볼 수 있습니다. Nurbs Curve-P 입력은 Curve가 닫힐 것인가 열릴 것인가를 결정합니다. False값은 열린 Nurbs Curve를, True값은 닫힌 폐곡선을 생성합니다. 세 개의 출력 탭은 직관적으로 알 수 있듯, C 출력은 생성되어진 Curve를, L 출력은 Curve의 길이 값을, D 출력 Curve의 domain 값을 나타냅니다. (Curve가 6개의 컨트롤 포인트로 만들어졌다면 0에서 6까지에 해당하는 domain 영역을 생성합니다.)

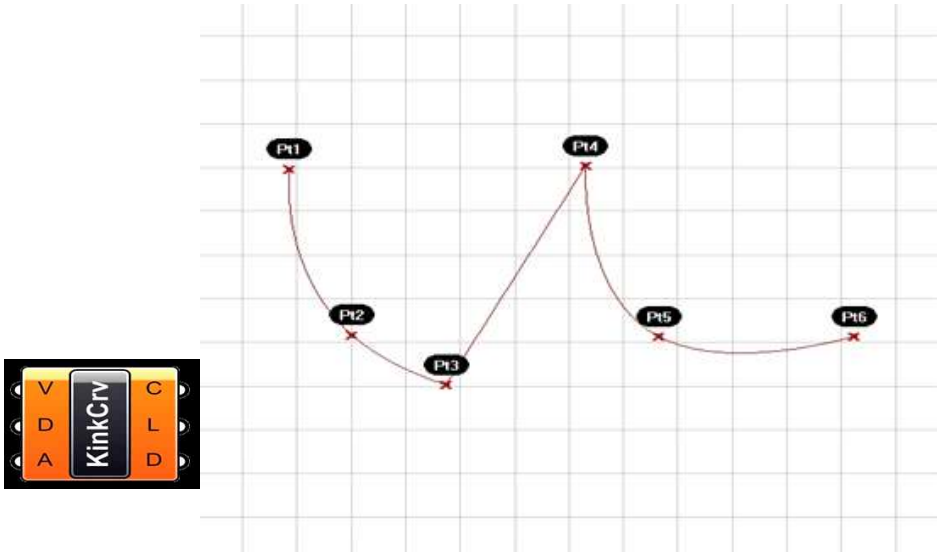
B)Interpolated Curve(Curve/ Spline/Interpolate)

Interpolated Curve는 Nurbs Curve와는 조금 거동이 다릅니다. 이 Curve는 컨트롤 포인트들을 지나면서 생성됩니다.
Nurbs Curve는 특정한 점을 지나도록 만드는 것이 어렵습니다. 심지어 각각의 컨트롤 포인트를 조절한다고 해도 특정 점을 지나는 Nurbs Curve를 만드는 것은 매우 힘든 일입니다.
V-입력은 Nurbs Curve와 마찬가지로 일련의 점들을 요구합니다. Interpolated Curve에서는 차원에 상관없이 Curve는 점들을 지나게 되는데, Nurbs Curve에서 컨트롤 포인트를 지나도록 하는 방법은 범위-1로 설정합니다.
D(Degree;범위)-입력 또한 Nurbs Curve와 같습니다. 그러나 홀수만 가능하므로 범위-2의 Interpolated Curve를 생성하는 것은 불가능합니다. P-입력은 Curve의 닫힘과 열림을 설정하고 세 개의 출력 탭은 Nurbs의 경우와 동일합니다. 세 개의 출력은 대부분의 Curve Object에 있어서 공통적으로 사용되어집니다.



C)비틀어진 Curve

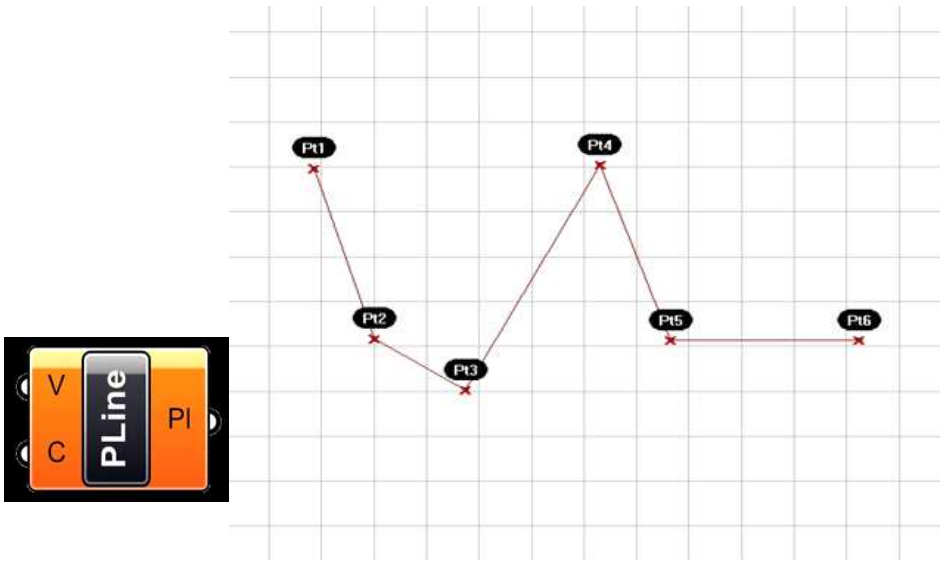
이름에도 불구하고 비틀어진 Curve는 또 다른 Interpolate Curve에 지나지 않습니다. 한 가지 차이를 제외하고는 모든 속성이 동일합니다. 비틀어진 Curve 컴포넌트는 기본적으로 범위가 0인 Nurbs Curve, 혹은 폴리라인과 같은 Curve를 생성합니다. 이 Curve는 angle threshold라는 입력 탭이 있는데 angle threshold에서 정의된 각도 이상으로 Curve가 연결되는 결절 점에서는 각이 진 연결이 부드러운 연결로 바뀝니다. 이때 연결되는 각도라 함은 두 개의 연결된 세그먼트들이 생성하는 각도 중 큰 각도에서 180도를 뺀 각도와 동일합니다. A-입력에는 숫자 슬라이드가 연결되어서 숫자 변화에 따른 Curve의 변화를 볼 수 있습니다. angle threshold의 각도는 Radian 단위로 입력되어야 하며, 예제에서는 범위 값을 Radian으로 변화하는 표현이 포함되어 있습니다.



D)폴리라인 Curve(Curve/ Spline/ Polyline)

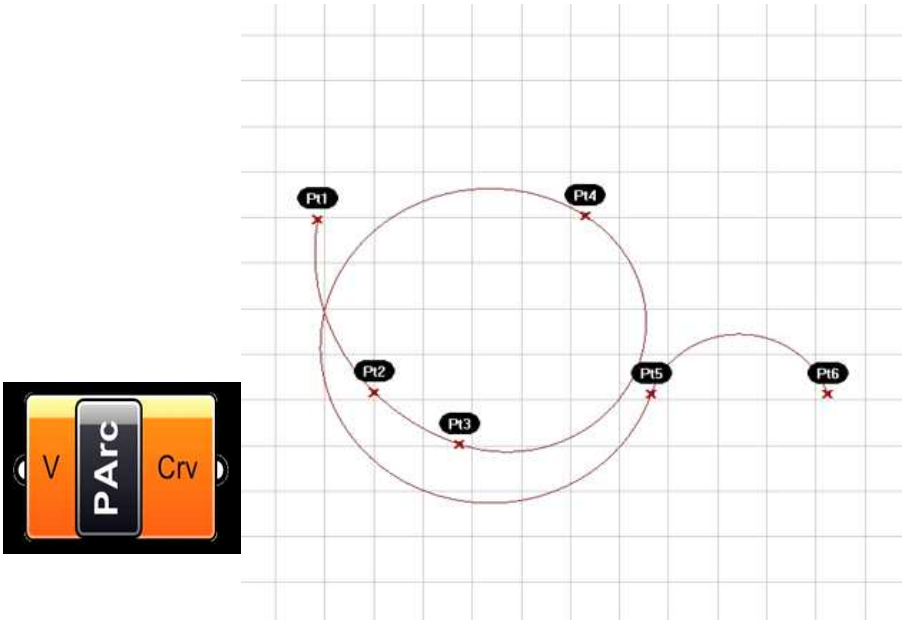
폴리라인 Curve는 아마도 Rhino에 존재하는 Curve 중에 가장 유연성이 좋은 Curve일 것입니다. 폴리라인 Curve는 라인 세그먼트로 만들어질 수도 있고, 폴리라인 세그먼트로 만들어질 수도 있으며, 범위-1의 Nurbs Curve로도 만들어질 수 있기 때문입니다.

기본적으로 폴리라인은 점의 배열과 동일합니다. 유일한 차이점은 폴리라인 Curve에서는 점에 순서를 부여하는 것입니다. 앞서 언급했듯 범위-1의 Nurbs Curve는 폴리라인과 동일합니다. 폴리라인 Curve는 두 개나 그 이상의 점을 연결하는 세그먼트의 집합이므로 얻어지는 Curve는 항상 컨트롤 포인트를 지나게 됩니다. 이는 Interpolated Curve의 속성과도 비슷합니다. V-입력은 각 라인 세그먼트를 구성하는 점들의 집합으로 정의됩니다. C-입력은 열림과 담힘 값을 설정하는 Boolean 변수입니다. 폴리라인 컴포넌트의 출력 탭은 다른 Curve들과 조금 다르게 오직 Curve 그 자체에 해당하는 탭 뿐 입니다. 생성된 Curve의 속성을 이용하고 싶다면 다른 형태의 Curve를 사용합니다.

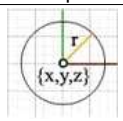
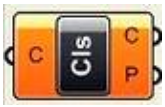
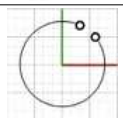
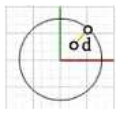
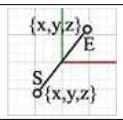
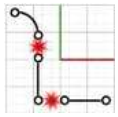
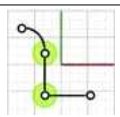
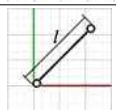
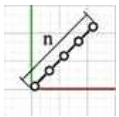
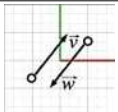
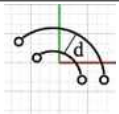
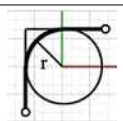


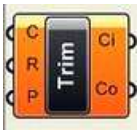
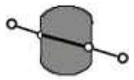
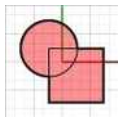
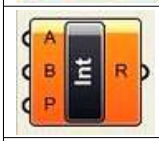
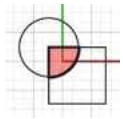
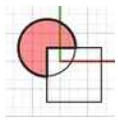
E)폴리아크(Curve/ Spline/ Poly Arc)

폴리아크는 폴리라인과 거의 동일합니다. 그러나 직선의 세그먼트 대신 아크 형태의 세그먼트를 가지는 것이 차이점입니다. 각 아크의 시작과 끝의 법선 값은 이웃하는 아크와 부드러운 연결을 가져서 전체적으로 부드러운 곡선을 생성할 수 있도록 자동적으로 계산되어 집니다. 추가적인 입력이나 출력 탭은 없습니다.



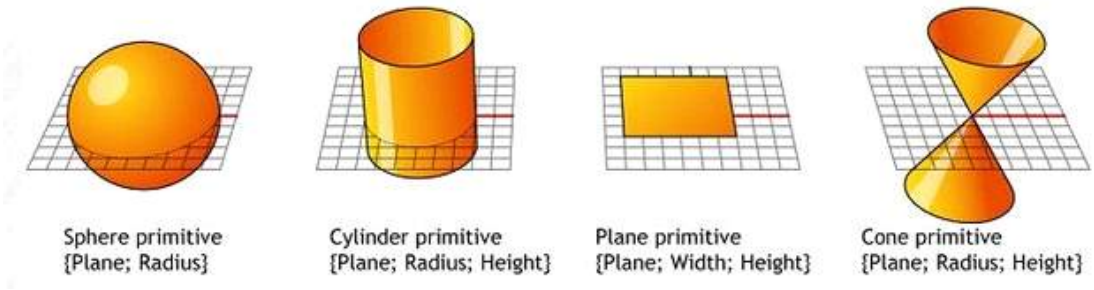
10.1 Curve 분석학

컴포넌트	Location	De스크립트ion	Example
	Curve/ Analysis/ Center	호들이나 원들의 반지름과 중심점을 찾습니다.	
	Curve/ Analysis/ Closed	Curve가 닫혀 있는지 혹은 주기적(정상, 정기)인지 테스트 합니다.	
	Curve/ Analysis/ Closest 점	공간상의 특정한 점으로부터 Curve에 가장 가까운 점을 찾아냅니다.(공간상에서의 Curve에 가장 가까운 어떤 샘플 점을 찾습니다.)	
	Curve/ Analysis/ End 점	Curve의 시작점과 끝점을 추출합니다.	
	Curve/ Analysis/ Explode	Curve를 세그먼트와 정점으로 분리하여 냅니다. Curve들의 안에서 컴포넌트 조각으로 Curve를 분리해냅니다.	
	Curve/ Utility/ Join Curve	가능한 많은 Curve 세그먼트들을 동반하여 결합합니다.	
	Curve/ Analysis/ Length	Curve의 길이를 측정합니다.	
	Curve/ Division/ Divide Curve	Curve를 동일한 간격으로 n등분 합니다.	
	Curve/ Division/ Divide Distance	두 Curve 사이를 미리 설정한 거리 값으로 나눕니다.	
	Curve/ Division/ Divide Length	미리 설정한 세그먼트의 길이 값으로 Curve를 나눕니다.	
	Curve/ Utility/ Filp	Curve 가이드 옵션을 사용하여 Curve의 방향을 바꿉니다.	
	Curve/ Utility/ Offset	지정된 거리 값으로 Curve를 오프셋 합니다.	
	Curve/ Utility/ Fillet	정의된 반지름 값으로 Curve의 날카로운 모서리를 필렛합니다.	

	Curve/ Utility/ Project	(하나의 Brep은 Rhino의 폴리Surface처럼 여러 개의 면이 합쳐진 세트객체입니다.)Brep 위로 Curve를 투영합니다.	
	Intersect/ Region/ Split sith Brep(s)	하나 혹은 여러 개의 Brep들로 Curve를 분할합니다.	
	Intersect/ Region/ Trim with Brep(s)	Curve를 하나 혹은 여러 개의 Brep들로 트림합니다. Curve들의 안쪽과 바깥쪽들의 출력(항목)은 분절된 양쪽 부분을 각각 나타냅니다.	
	Intersect/ Trim with Region(s)	Curve를 하나 혹은 여러 개의 범위들로 트림합니다. Curve들의 안쪽과 바깥쪽들의 출력(항목)은 분절된 양쪽 부분을 각각 나타냅니다.	
	Intersect/ Boolean/ Region Union	두 개의 2차원 평면으로 닫힌 Curve들로 최 외곽선 혹은 병합된 바깥 Curve들을 생성합니다. 다수의 것도 마찬가지입니다.	
	Intersect/ Boolean/ Region Intersection	두 개의 닫힌 Curve들의 교차점 외곽 Curve들을 생성합니다.	
	Intersect/ Boolean/ Region Difference	두 개의 2차원 평면 Curve들 사이에서 다른 부분의(교차하지 않는 부분) 외곽 라인을 찾아냅니다.	

11 표면 유형

구, 콘, 평면 혹은 실린더와 같은 몇 개의 기본적건 표면(Surface) 타입 외에 Rhino는 3종류의 자유로운 곡면 타입을 가지고 있는데, 이들 중 가장 유용한 표면은 Nurbs 표면입니다. Curve와 비슷하게 모든 가능한 면의 형태는 Nurbs 면으로 나타내어질 수 있으며, Rhino의 기본적인 면입니다. Nurbs 표면은 가장 유용하기 때문에 Nurbs 표면에 대하여 중점적으로 이야기 하겠습니다.



Nurbs Surface는 Nurbs Curve와 비슷한 속성을 가지고 있습니다. 같은 알고리즘은 형태, 법선, 접선, 곡률 등을 계산하기 위해 쓰입니다. 그러나 몇 가지 눈에 띄는 차이점이 있습니다. 예를 들어, Nurbs Curve는 접선 Vector(탄젠트)와 법선 평면(노말)을 가지고 있는데 반해, Nurbs Surface는 법선 Vector(노말)와 접선 평면(탄젠트)을 가지고 있습니다. 이는 Nurbs Curve는 목표하는 방향이 없는데 비해 Nurbs Surface는 방위의 방향이 없습니다. 이러한 사실은 모든 Curve와 면에 해당하며 이러한 사실에 익숙해져야 합니다.

Nurbs Surface의 경우에 두 가지 방향이 존재합니다. 이는 Nurbs Surface가 기본적으로 u, v방향을 가지는 사각형의 그리드에 바탕을 두고 있기 때문입니다. 종종 이러한 방향이 임의적이기는 하지만, 이러한 방향성을 사용하는 것은 Nurbs Surface를 컨트롤하는데 유용합니다.
Grasshopper는 Nurbs Surface를 다루는 방법이 Rhino와 비슷합니다. 이는 두 프로그램이 같은 엔진을 가지고 면을 생성하기 때문입니다. 그러나 Grasshopper는 Rhino의 뷰포트를 통해 면을 표현하기 때문에(이러한 이유로 Grasshopper에서 bake를 하지 않고는 Rhino 뷰포트에서 면을 선택할 수 없는 것입니다.) 일부 Mash 설정이 Grasshopper 연산속도를 높이기 위해서 낮게 설정되어 있습니다. Grasshopper에서 만들어진 면들은 다소 각이 져 보이지만 bake를 통해 Rhino 면으로 전환되면 높은 Mash 설정을 갖고 있음을 볼 수 있습니다.

Grasshopper는 면을 두 가지 방식으로 다룹니다. 첫 번째는 이미 논의하였듯이, Nurbs Surface를 통해 이고, 일반적으로 모든 면 분석의 컴포넌트들은(면적을 구하거나, 곡률을 계산하거나 하는) Nurbs Surface에서 사용 가능합니다. 물론 면을 다루는 것 또한 복잡한 수학공식을 바탕으로 하고 있으나, 이는 컴퓨터의 입장에서 보면 다소 간단합니다.
왜냐하면 이러한 면들은 두께와 같은 삼차원적인 요소가 고려되지 않기 때문입니다. McNeel사의 개발자들은 삼차원(솔리드 Object) 물체를 다룰 수 있는 두 번째 방법도 제공

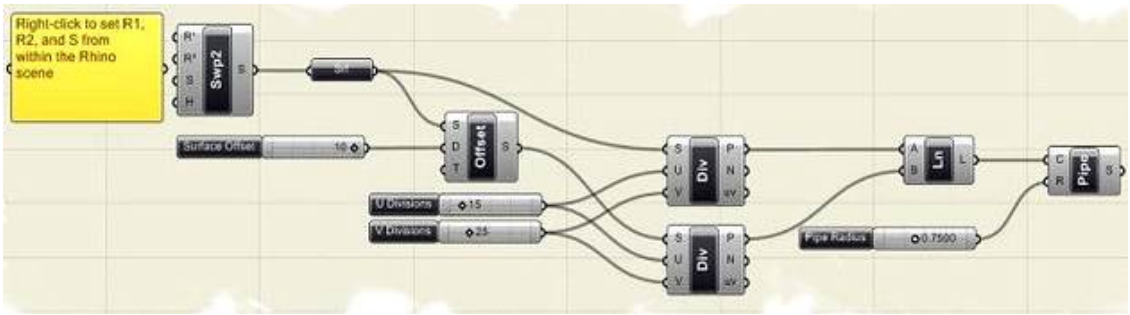
하는데 이것이 Brep(Boundary Representation), Grasshopper의 면(두께를 가진 면)을 해석할 수는 있는 방법입니다.
Brep은 3차원의 솔리드 Object 혹은 Rhino의 폴리 Surface와 같은 방식으로 생각할 수 있습니다. 이는 당연히 Nurbs Surface의 집합이지만, 단일 Nurbs Surface가 이론상으로는 두께가 없는데 반해 Brep은 두께를 갖습니다. Brep이 몇 개의 Nurbs Surface로 이루어져 있으므로 몇몇의 Surface 분석 툴은 Brep에서도 여전히 사용 가능합니다. (모든 Surface 분석 툴이 Brep에서 사용 가능한 것은 아닙니다.) 이는 Grasshopper가 내부적인 변환의 논리를 가지고 있기 때문입니다. 즉, 입력해야 하는 입력 단자에 Brep을 입력한다고 하면 Grasshopper는 자동적으로 Brep을 여러 개의 면으로 분해해서 인식 가능하기 때문입니다. 이는 숫자와 정수, 색과 Vector, 호와 원 등에도 해당합니다.
다음과 같은 아주 특별한 경우도 존재합니다.

·Curve → Number(Curve의 길이를 추출합니다.)
·Curve → Interval(Curve의 도메인을 추출합니다.)
·Surface → Interval2D(면의 u, v도메인을 추출합니다.)
·String → Number(문자열을 평가합니다.)
·Interval2D → Number(2차원 Interval이 만드는 영역의 넓이를 계산합니다.)

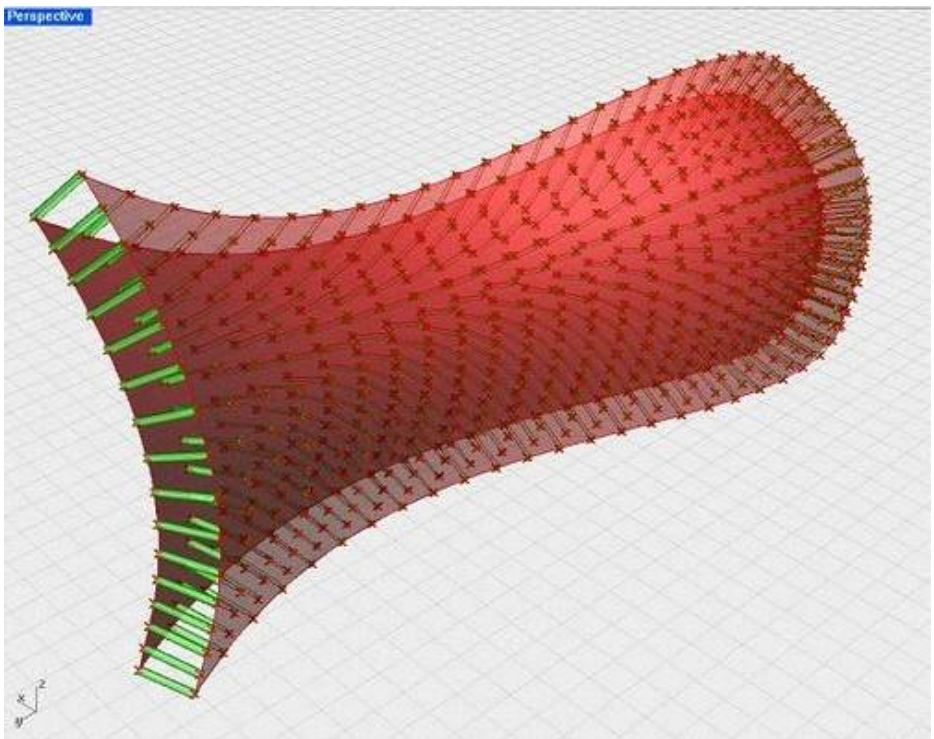
더 많은 자동 변환의 경우가 있지만, 위의 경우만으로 몇 개의 예제를 이해하는 데에는 충분합니다.

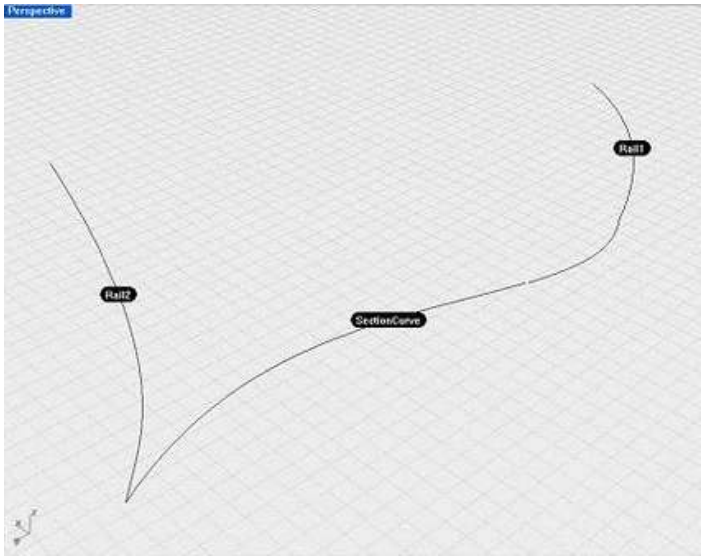
11.1 면의 연결

다음 예제는 면을 컨트롤하는 기술에 대한 좋은 예이며, David Fano(Design Reform)에 의해 만들어졌습니다. 이 예제에서는 Sweep 2Rail, Surface Offset, Surface Division 컴포넌트에 대해 다룰 것입니다. 시작에 앞서 Surface Connect. 3dm (Source File Folder)를 엽니다. 이 파일에는 3개의 Curve(2개의 Rail Curve와 한 개의 단면Curve)가 있습니다.



Note: 완성된 파일의 정의는 Surface Connect.ghx를 참조합니다.



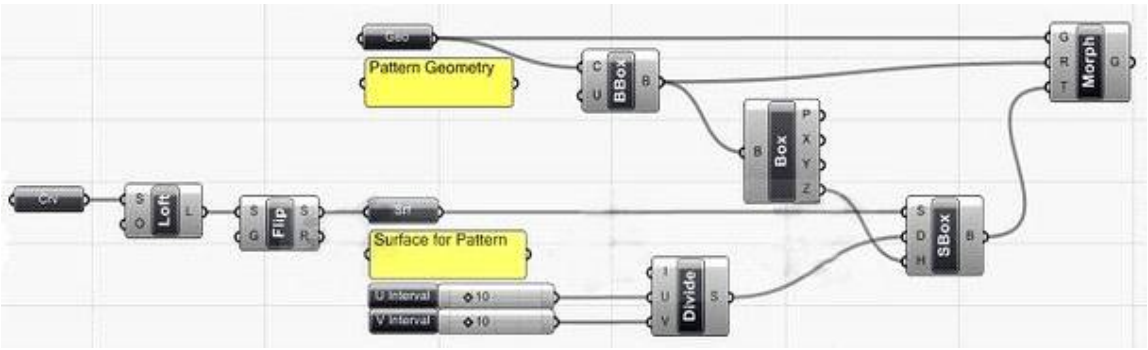


정의 파일을 만드는 순서:
·Sweep 2Rail 컴포넌트를 생성합니다.(Surface/ Freeform/ Sweet 2 Rail)
·Sweep 2Rail-R1 입력에 오른쪽 클릭한 후 Set one Curve를 선택합니다.
·첫 번째 Rail Curve를 선택합니다.
·Sweep 2Rail-R2 입력에 오른쪽 클릭한 후 Set one Curve를 선택합니다.
·단면Curve를 선택합니다.
올바로 선택되었다면 두 개의 레일 Curve 사이에 생성된 면이 보일 것입니다.
·Surface Offset 컴포넌트를 생성합니다.(Surface/ Freeform/ Offset)
·Sweep 2Rail-S 출력을 Offset-S 입력에 연결합니다.
·Numeric Slider를 생성합니다.(Params/ Special/ Slider)
·Slider 오른쪽 클릭한 후 다음과 같이 설정합니다.:
o 이름: Surface Offset
o Slider 유형: Floating 점
o 최소 값: 0,0
o 최대 값: 10,0
o 결과 값: 10,0
·Slider를 Surface Offset-D 입력에 연결합니다.
새로운 면이 10만큼 떨어져서 생성 된 것이 보입니다.
·Divide Surface 컴포넌트 2개를 생성합니다.(Surface/ Utility/ Divide Surface)
·Sweep 2Rail-S 출력을 첫 번째 Divide Surface-S 입력에 연결합니다.
일련의 점들이 첫 번째 Sweep Surface 위에 나타나게 되고, u, v방향 모두 10이 기본 값이며 이는 면을 각 방향별로 10개로 나눕니다. u, v방향의 점을 각각 연결 시에는 iso Curve를 얻게 됩니다.
·Surface Offset-S 출력을 Divide Surface-S 입력에 연결합니다.
새로운 점들이 두 번째 Surface 위에 생성됩니다.
·Numeric Slider 2개를 생성합니다.(Params/ Special/ Slider)
·첫 번째 Slider에 오른쪽 클릭한 후 다음과 같이 설정합니다.:

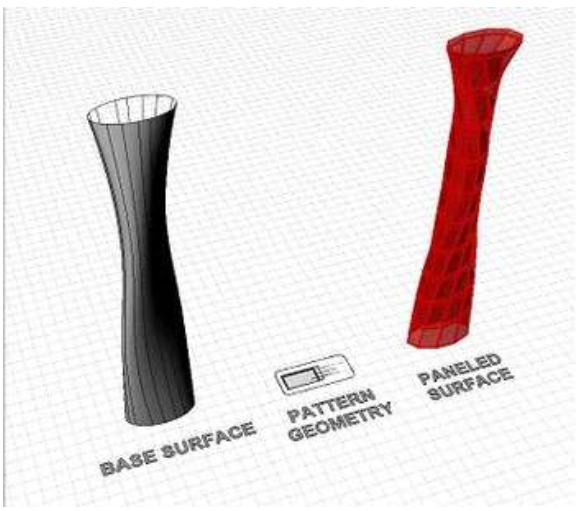
o 이름: U Divisions
o Slider 유형: Integers
o 최소 값: 0,0
o 최대 값: 100,0
o 결과 값: 15,0
·두번째 Slider에 오른쪽 클릭한 후 다음과 같이 설정합니다.:
o 이름: V Divisions
o Slider 유형: Integers
o 최소 값: 0,0
o 최대 값: 100,0
o 결과 값: 25,0
·U Division Slider를 두 개의 Divide Surface-U 입력s에 연결합니다.
·V Division Slider를 두 개의 Divide Surface-V 입력s에 연결합니다.
두 개의 Slider는 면 분할의 개수를 정의합니다. 두 개의 면 모두 동일한 수의 u, v분할 개수를 가지고 있으므로 우리는 두 그룹의 점들을 1대1로 연결하여 내, 외면을 연결하는 선들을 얻을 수 있습니다.
·Line 컴포넌트를 생성합니다.(Curve/ Primitive/ Line)
·첫 번째 Divide Surface-P 출력을 Line-A 입력에 연결합니다.
·두번째 Divide Surface-P 출력을 Line-B 입력에 연결합니다.
일련의 선들이 두 면상의 각점을 잇는 것을 볼 수 있습니다. 다음 단계로 각 선들에게 두께를 부여합니다.
·Pipe 컴포넌트를 생성합니다.(Surface/ Freeform/ Pipe)
·Line-L 출력을 Pipe-C 입력에 연결합니다.
·Numeric Slider를 생성합니다.(Params/ Special/ Slider)
·Slider에 오른쪽 클릭한 후 다음과 같이 설정합니다.:
o 이름: Pipe Radius
o Slier 유형: Floating 점
o 최소 값: 0,0
o 최대 값: 2,0
o 결과 값: 0,75
·Pipe Radius Slider를 Pipe-R 입력에 연결합니다.

11.2 패널링 도구(Tool)

McNeel은 최근에 패널링 도구라는 강력한 Rhino 플러그인을 만들었습니다. 이 플러그인은 주어진 면 위에 특정한 지오메트리를 배열하는데 쓰인다. Grasshopper 역시 이러한 기능을 할 수 있으며, 이번 예제에서는 면을 어떻게 나누고 특정한 지오메트리를 나누어진 면에 배열을 할 수 있는지를 알아봅니다.
http://en.wiki.mcneel.com/default.aspx/PanelingTools.html.



패널링 도구에 대한 더욱 자세한 정보는 다음 링크를 참고합니다.
http://en.wiki.mcneel.com/default.aspx/McNeel/PanelingTool.html.
아래는 지오메트리 패턴(커튼월의 멀리언 등)을 고층 빌딩의 면위에 복사하여 배열하는 정의입니다.



Note: 완성된 정의를 보려면 Paneling Tool.ghx를 참조합니다.

빌딩을 만들기 위하여 로프트에 사용되어질 Curve가 필요합니다. Rhino에서 panel tool_base.3dm 파일을 엽니다. 빌딩의 외부 면을 만들기 위해 4개의 타원이 필요합니다.
·Curve 컴포넌트를 생성합니다.(Params/ Geometry/ Curve)
·Curve 컴포넌트에 오른쪽 클릭한 후, Set Multiple Curves를 선택합니다.

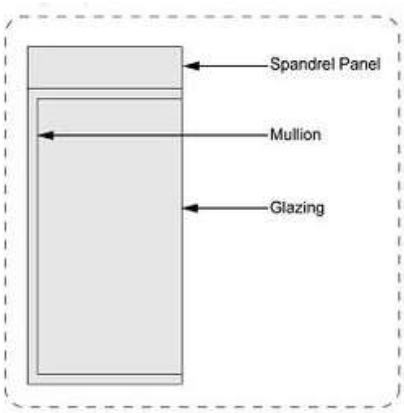
·가장 아래쪽의 타원으로부터 시작하여 상단의 타원 순으로 선택합니다.
·선택 완료 후 엔터를 누릅니다.
전의 예제에서와 같이 Rhino Object를 Grasshopper의 Object에 연결하였습니다.
·Loft 컴포넌트를 생성합니다.(Surface/ Freeform/ Loft)
·Curve 출력을 Loft-S 입력에 연결합니다.
Loft-O 입력에서 오른쪽 클릭 시 Rhino의 로프트 명령 창에서 로프트 옵션을 볼 수 있습니다. 본 예제의 경우에는 모든 값을 Default 값으로 설정합니다. 추후에 이 옵션을 다룰 일이 있을 것입니다.
*추가 과정: Grasshopper 상에서는 현재 만들어진 로프트 면이 내·외부가 뒤집혔는지에 대하여 알 수 없습니다. 본 예제의 경우, 면이 내부를 향함을 뒤늦게 알게 되었습니다. 그래서 Flip 컴포넌트를 사용하여 면의 방향을 뒤집기로 합니다.
하지만, 면의 방향의 대해서는 확신할 수 없으므로 나중에 지오메트리 패턴들이 원하는 방향과 반대의 방향(내부로 향해 배열)으로 배열되어 있다면, Flip 컴포넌트를 삭제함으로써 문제를 해결할 수 있습니다.
·Flip 컴포넌트를 생성합니다.(Surface/ Utility/ Flip)
·Loft-L 출력을 Flip-S 입력에 연결합니다.
·Surface 컴포넌트를 생성합니다.(Params/ Geometry/ Surface)
·Flip-S 출력을 Surface 컴포넌트's 입력에 연결합니다.
·Surface 컴포넌트를 오른쪽 클릭한 후, Reparameterize에 체크를 합니다.
현재 우리가 만든 면의 도메인 영역 값으로 0부터 면의 최상단에 해당하는 값(실제적인 값-Rhino에서 Distance를 통해 얻을 수 있는 값)을 가지고 있습니다. 하지만 우리는 Rhino에서 타원의 위치 조절을 통해 얼마든지 전체 빌딩의 높이를 수정할 수 있기에 면의 최상단에 해당하는 도메인 값은 변할 여지가 있습니다. 만약 빌딩의 외측면의 도메인이 0에서 시작해 1로 끝나는 상대적인 값이라면, 즉 0은 최 하단의 도메인 값이고 1은 최상단의 도메인 값이라면, 빌딩의 전체적인 높이 값이 실제적으로 얼마가 되던 이 면을 컨트롤하기에 용이할 것입니다. 이는 아주 중요한 과정이며 이 과정이 없이는 면을 정확하게 나눌 수 없을 것입니다.
Reparameterize는 임의의 범위를 가지는 도메인의 영역을 0에서 시작해 1로 끝나는 상대적인 도메인 값으로 전환합니다.

·2개의 수치 간격 컴포넌트를 생성합니다.(Scalar/ Interval/Divide Interval2)
면을 나누기에 앞서 범위를 설정합니다.

·Divide Interval 컴포넌트의 I-입력을 오른쪽 클릭한 후, Manage Interval Collection을 선택합니다.
·녹색의 +버튼을 눌러서 Interval을 추가합니다.
기본 값으로 Interval은 u:{0.0 ~ 0.0} v:{0.0 ~ 0.0}로 정의 됩니다. 하지만 우리는 u, v 두 방향의 범위{0 ~ 1}이 필요합니다.
·u의 끝 값을 1.0으로 설정합니다.
·v의 끝 값을 1.0으로 설정합니다.

Interval 설정은 위의 그림과 같을 것입니다. OK를 눌러 창을 빠져나옵니다.
이제 마우스를 Divide Interval-I 입력 탭에 올리면 u, v 두 방향 모두 간격이 0~1로 설정된 것을 볼 수 있습니다. 이는 Reparameterize 시킨 면의 Interval, 0~1과 동일한 값이 됩니다.
·두 개의 Numeric Slider를 생성합니다.(Params/ Special/ Slider)
·첫 번째 Slider를 오른쪽 클릭한 후, 다음과 같이 설정합니다.:
o 이름: U Interval
o Slider 유형: Integers
o 최소 값: 5.0
o 최대 값: 30.0
o 결과 값: 10.0
·두번째 Slider를 오른쪽 클릭한 후, 다음과 같이 설정합니다.:
o 이름: V Interval
o Slider 유형: Integers
o 최소 값: 5.0
o 최대 값: 30.0

o 결과 값: 10.0
·U Interval Slider를 Divide Interval-U 입력에 연결합니다.
·V Interval Slider를 Divide Interval-V 입력에 연결합니다.
·Surface Box 컴포넌트를 생성합니다.(Xform/ Morph/ Surface Box)
·Surface 컴포넌트 출력을 Surface Box-D 입력에 연결합니다.
·Divide Interval-S 출력을 Surface Box-D 입력에 연결합니다.
·Curve, Loft, Surface를 오른쪽 클릭하여 Preview를 끕니다.



빌딩의 면을 두 개의 Slider를 통해 100개의 영역으로 나누었습니다. 이제까지 작업한 것을 돌아보면, 첫 번째로 0~1까지 해당하는 Interval을 생성하였으며, 이는 면의 영역인 0~1과 동일합니다.

그 다음 그 Interval을 u, v방향으로 각각10개씩 나누었으며, 이는 곧 100개의 Interval 영역을 생성한 것입니다. Slider를 움직임으로써 전체 면을 몇 개의 Interval로 나눌 것인가에 대한 정의가 가능합니다.

이제 면은 잠시 잊고 지오메트리 패턴에 대해 생각해봅니다. Rhino 파일 내에는 윈도우 시스템을 발견할 수 있을 것입니다. 윈도우 시스템은 spandrel panel, mullion, glazing panel로 구성되어 있습니다. 우리는 이제 이 지오메트리 패턴을 사용하여 기존에 생성된 면 위에 이 패턴을 배열할 것입니다.
·Geometry 컴포넌트를 생성합니다.(Params/ Geometry/ Geometry)
·Geometry 컴포넌트를 오른쪽 클릭한 후, Ser Multiple Geometries를 선택합니다.
·spandrel panel, mullion, glazing panel을 Rhino 뷰포트에서 선택합니다.
·모든 객체를 선택한 후, 엔터를 누릅니다.
·Bounding Box 컴포넌트를 생성합니다.(Surface/ Primitive/ Bounding Box)
·Geometry 컴포넌트 출력을 Bounding Box-C 입력에 연결합니다.
Bounding Box 컴포넌트를 사용하는 2가지 이유가 있는데, 첫째로 Bounding Box 컴포넌트는 지오메트리 패턴의 높이(두께)를 결정하게 돕습니다. 이 예제의 경우 우리의 지오메트리 패턴은 단순한 박스이므로 직관적으로 높이를 알 수는 있으나, 유기적인 지오메트리 패턴을 사용할 때에 전체의 높이를 알기 어려운 경우도 더러 있었을 것입니다. 이제는 지오메트리 패턴의 높이 값을 Surface Box 컴포넌트의 입력 값으로 사용할 것입니다.

두 번째 이유는 Bounding Box를 조금 후에 다룰 Box Morph 컴포넌트의 Reference 값으로 사용할 것이기 때문입니다.

·Bounding Box 컴포넌트의 U-입력을 오른쪽 클릭한 후, Boolean 값을 True로 설정합니다. 이 값을 True로 설정하는 것은 여러 개의 Brep 객체를 통해 단 하나의 Bounding Box를 생성하기 위함으로 중요한 과정입니다.

·Box 컴포넌트를 생성합니다.(Surface/ Analysis/ Box 컴포넌트)

·Bounding Box-B 출력을 Box 컴포넌트-B 입력에 연결합니다.

·Box 컴포넌트-Z 출력을 Surface Box-B 입력에 연결합니다.

·Box Morph 컴포넌트를 생성합니다.(Xform/ Morph/ Box Morph)

·Pattern Geometry 출력을 Box Morph-G 입력에 연결합니다.

·Bounding Box-B 출력을 Box Morph-R 입력에 연결합니다.

·Surface Box-B 출력을 Box Morph-T 입력에 연결합니다.

·Morph Box 컴포넌트를 오른쪽 클릭한 후, Data matching을 Cross Reference로 설정합니다.

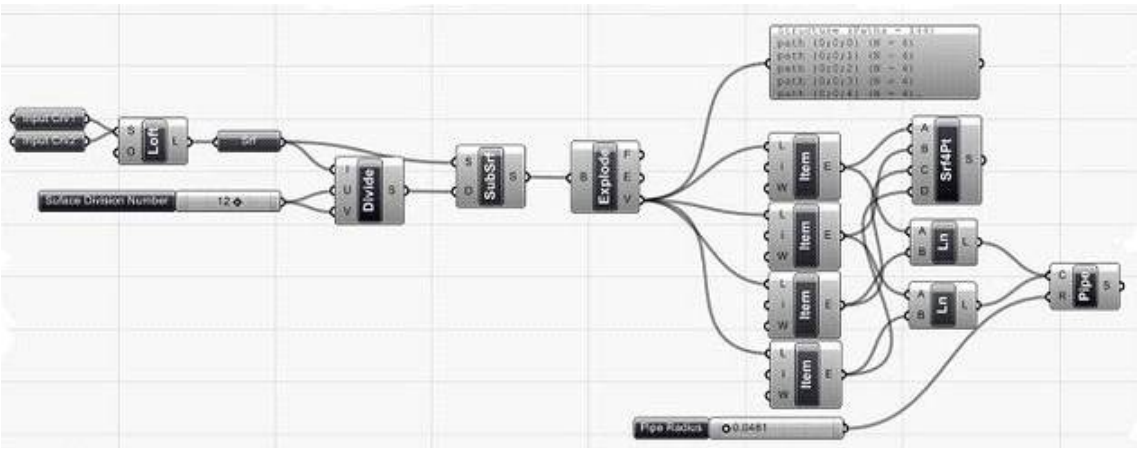
·Surface Box 컴포넌트를 오른쪽 클릭한 후, Preview를 끕니다.

정리하자면, 지오메트리 패턴을 Morph Box 컴포넌트에 입력하였습니다. 이로써 어떤 지오메트리 패턴을 복제하여 배열할 것인가를 정의하였습니다. 그 다음 지오메트리 패턴으로부터 생성된 Bounding Box를 참조 지오메트리로 사용하였는데, 이로써 지오메트리 패턴을 구성하고 있는 개별의 Object들이 하나의 Object로 인식되어 면 위에 위치할 수 있도록 설정하였습니다.

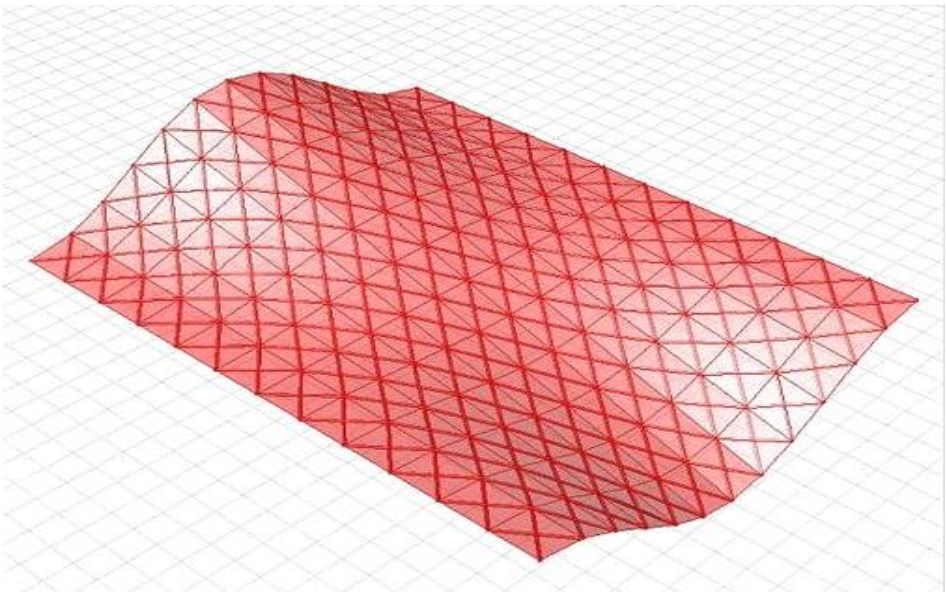
그리고 마지막으로 100개의 분할된 박스를 개별의 참조 지오메트리를 입력할 타겟으로 설정하였습니다. 모든 과정이 제대로 진행되었다면, 이제 지오메트리 패턴을 수정하거나 u, v 값을 수정함으로써 전체적인 시스템이 어떻게 변화하는지 볼 수 있을 것입니다.

11.3 대각선 그리드(다이아몬드 모양의 그리드 패턴) 표면

우리는 이전 예제를 통하여 파사드를 패널링 하는 방법에 대해 알아보았습니다. 이번 예제는 앞서 다룬 예제와 비슷한 과정을 통해 구조적인 대각선 그리드 부재를 면 위에 생성하는 방법을 알아볼 것입니다. Surface Dia그리드.3dm을 Rhino에서 엽니다. 파일 내에 있는 뒤집힌 코사인 Curve를 이용하여 로프트 면을 생성할 것입니다.



Note: 완성된 정의 파일을 보려면 Surface Dia그리드.ghx를 참조합니다.



정의 파일을 만드는 순서:

·2개의 Curve 컴포넌트를 생성합니다.(Params/ Geometry/ Curve)

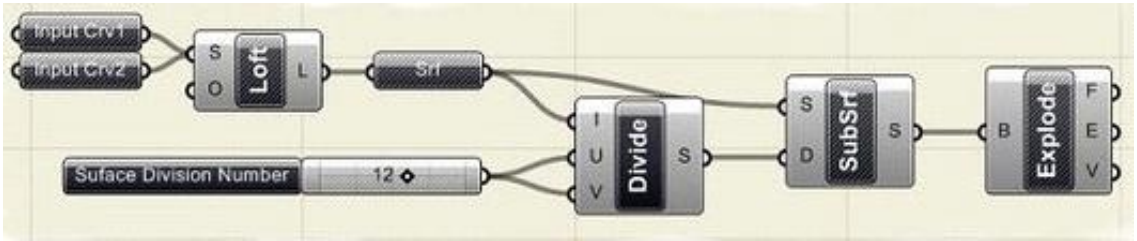
·첫 번째 Curve 컴포넌트를 오른쪽 클릭한 후, 이름을 "입력 Crv1"로 바꿉니다.

·입력 Crv1 컴포넌트를 오른쪽 클릭한 후, Set One Curve를 선택합니다.

·Rhino 뷰포트 상의 한 Curve를 선택합니다.

·두번째 Curve 컴포넌트를 오른쪽 클릭한 후, 이름을 "입력 Crv2"로 바꿉니다.

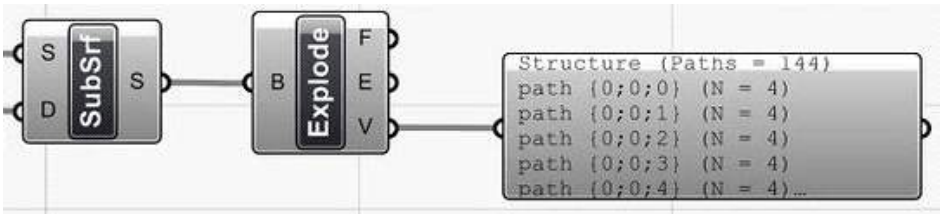
·입력 Crv2 컴포넌트를 오른쪽 클릭한 후, Set One Curve를 선택합니다.
·Loft 컴포넌트를 생성합니다.(Surface/ Freeform/ Loft)
·입력 Crv1 컴포넌트를 Loft 입력-S 입력에 연결합니다.
·Shift 키를 누른 채로, 입력 Crv2 컴포넌트를 Loft 입력-S 입력에 연결합니다.
두 개의 Curve 사이에 면이 생성됨을 알 수 있습니다.
·Surface 컴포넌트를 생성합니다.(Params/ Geometry/ Surface)
·Loft-L 출력을 Surface 컴포넌트 입력에 연결합니다.
·2개의 Dimensional Interval 컴포넌트를 생성합니다.(Scalar/ Interval/ Divide Interval2)
전의 예제와 동일하게 면을 분할할 것입니다. 이를 위해 면 분할함으로써, u, v방향으로 Interval을 생성합니다.
·Surface 컴포넌트 출력을 Divide Interval-I 입력에 연결합니다.
·Slider를 생성합니다.(Params/ Special/ Numeric Slider)
·Slider를 오른쪽 클릭한 후, 다음과 같이 설정합니다.:
o 이름: Surface Division Number
o Slider 유형: Integers
o 최소 값: 0.0
o 최대 값: 20.0
o 결과 값: 12.0
·Slider를 Divide Interval 컴포넌트의 u, v 입력에 연결합니다.
·Isotrim 컴포넌트를 생성합니다.(Surface/ Utility/ Isotrim)
·Surface 컴포넌트 출력을 Isotrim-S 입력에 연결합니다.
·Divide Interval-S 출력을 Isotrim-D 입력에 연결합니다.
·Loft, Surface 컴포넌트를 오른쪽 클릭한 후, Preview를 끕니다.
숫자 Slider에서 분할된 값에 대응하는 분할된 면의 설정 값을 볼 수 있습니다. 하나의 Slider가 Interval을 위해 u, v 입력에 연결되며, 숫자 Slider가 오른쪽, 왼쪽 동일하게 나누어진 것을 볼 수 있습니다.
u, v가 같은 Slider에 연결되어 있으므로 하나의 Slider로 u, v값을 모두 조절합니다. 독립적으로 u, v값을 조절하려면 Slider를 하나 더 추가합니다.
·Brep 컴포넌트를 생성합니다.(Surface/ Analysis/ Brep 컴포넌트)
이 컴포넌트는 Brep을 면, 모서리 그리고 절점으로 분해합니다. 이 경우에는 양 코너에 있는 점의 위치를 알아내어 대각선의 연결을 만들어야 합니다.
·Isotrim-S 출력을 Brep 컴포넌트-B 입력에 연결합니다.



지금까지의 정의 파일의 모습은 위 그림과 같습니다. 하나의 로프트 된 큰 면을 분할해서

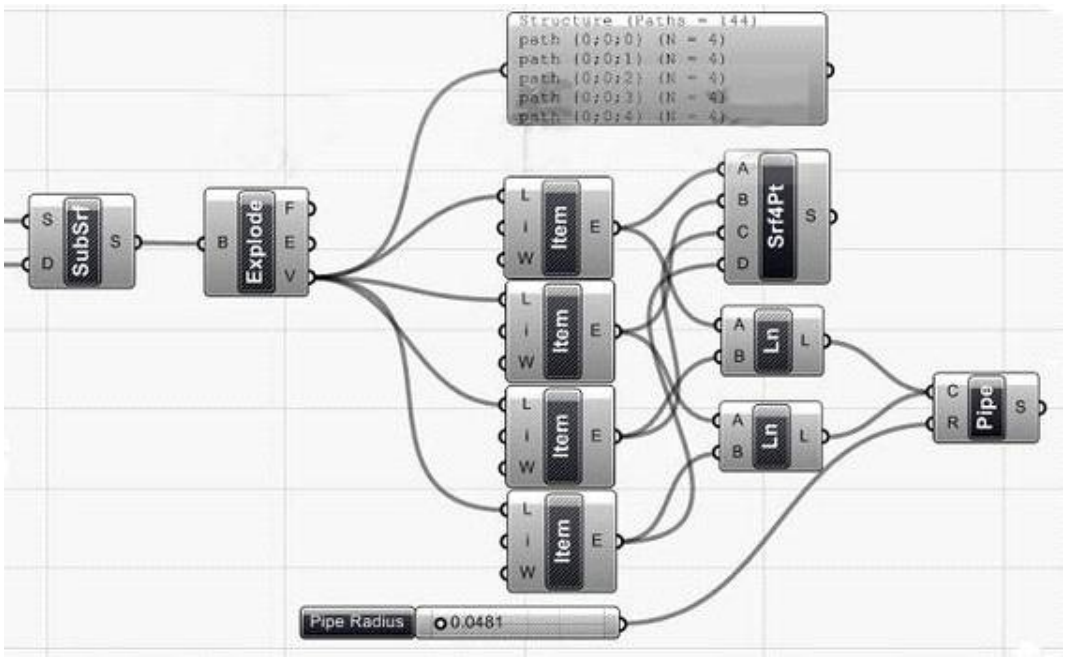
각각의 작은 면을 생성한 후, 이를 Explode하여 서로 분리합니다. 다음으로 각각의 분해된 면에 위치한 점들이 데이터 구성방식에서 어떻게 구성되어 있는지 살펴봅니다.

·Parameter Viewer 컴포넌트를 생성합니다.(Params/ Special/ Parameter Viewer)
·Brep 컴포넌트-V 출력을 Parameter Viewer 입력에 연결합니다.
Parameter Viewer를 통해 Tree Structure가 어떻게 구성되어 있는지 볼 수 있습니다.
본 예제의 경우에는 144개의 패스가 존재하며 각각의 패스에는 4개의 데이터들이 들어 있습니다. (작은 면의 개수가 144개이며, 각각의 면에는 4개의 꼭지점들이 있습니다.) List Item 컴포넌트를 이용하면 각각의 패스에서 특정한 데이터 값을 추출해 낼 수가 있습니다. 대각선의 연결을 만들기 위해 각각의 좌표 공간에 위치하는 4개의 꼭짓점의 순서를 알아야 합니다.



·List Item 컴포넌트 4개를 생성합니다.(Logic/ List/ List Item)
·Brep 컴포넌트-V 출력을 4개의 List Item-L 입력s에 연결합니다.
·첫 번째 List Item의 i-입력 탭에 오른쪽 클릭한 후, Integer 값을 0으로 설정합니다.
·두번째 List Item의 i-입력 탭에 오른쪽 클릭한 후, Integer 값을 1로 설정합니다.
·세 번째 List Item의 i-입력 탭에 오른쪽 클릭한 후, Integer 값을 2로 설정합니다.
·네 번째 List Item의 i-입력 탭에 오른쪽 클릭한 후, Integer 값을 3으로 설정합니다.
List Item의 Index 수를 0에서 시작해서 각각 1씩 증가시킨 것을 알 수 있습니다. 이는 Tree Structure로 설정된 Grasshopper 상에서 인지할 수 있는 각각의 다른 144개의 점과 총 144개의 면에서 각각 첫 번째, 두 번째, 세 번째 그리고 네 번째 Item들을 추출한 것입니다. 그래서 4개의 Item 컴포넌트는 각각 첫 번째 점들의 집합, 두 번째 점들의 집합, 세 번째 점들의 집합 그리고 네 번째 점들의 집합이 됩니다. 이제 각각의 4 꼭짓점이 서로 분리되었음으로 대각선 그리드(Dia그리드)를 만들기 위해 점들을 연결하는 선을 생성하면 됩니다.
·2개의 Line 컴포넌트를 생성합니다.(Curve/ Primitive/ Line)
·첫 번째 List Item-E 출력을 첫 번째 Line-A 입력에 연결합니다.
·세 번째 List Item-E 출력을 첫 번째 Line-B 입력에 연결합니다.
·두번째 List Item-E 출력을 두 번째 Line-A 입력에 연결합니다.
·네 번째 List Item-E 출력을 두 번째 Line-B 입력에 연결합니다.
여기서 각각의 그리드가 표면 위에 생성됨을 볼 수 있습니다.
·Pipe 컴포넌트를 생성합니다.(Surface/ Freeform/ Pipe)
·첫 번째 Line-L 출력을 Pipe-C 입력에 연결합니다.
·Shift 키를 누른 체로, 두 번째 Line-L 출력을 Pipe-C 입력에 연결합니다.
파이프의 반지름을 컨트롤하기 위하여 숫자 Slider를 생성해서 이를 Pipe radius에 연결합니다.

·Numeric Slider를 생성합니다.(Params/ Special/ Number Slider)
·Slider를 오른쪽 클릭한 후, 다음과 같이 설정합니다.:
o 이름: Pipe Radius
o Slider 유형: Floating 점
o 최소 값: 0.0
o 최대 값: 1.0
o 결과 값: 0.05
·Pipe Radius Slider를 Pipe-R 입력에 연결합니다.
마지막으로 각각의 4Point를 이용하여 4Point를 모두 포함하는 평면을 생성합니다. 이는 원래의 작은 면들은 곡률을 가지고 있어서 방금 생성한 대각선 그리드와 만나지 않는 경우가 생길 수 도 있기 때문입니다.
·4 Point Surface 컴포넌트를 생성합니다.(Surface/ Freeform/ 4Point Surface)
·첫 번째 List Item-E 출력을 4 Point Surface-A 입력에 연결합니다.
·두번째 List Item-E 출력을 4 Point Surface-B 입력에 연결합니다.
·세 번째 List Item-E 출력을 4 Point Surface-C 입력에 연결합니다.
·네 번째 List Item-E 출력을 4 Point Surface-D 입력에 연결합니다.
4 Point Surface 컴포넌트와 Pipe 컴포넌트 이외의 Preview를 끕니다.



마지막 부분은 위의 그림과 같을 것입니다. 이 정의는 어떤 종류의 단일 면에서든지 작동할 것이며 로프트를 사용하는 대신에 다른 방법으로 면을 생성해도 상관없이 작동할 것입니다.

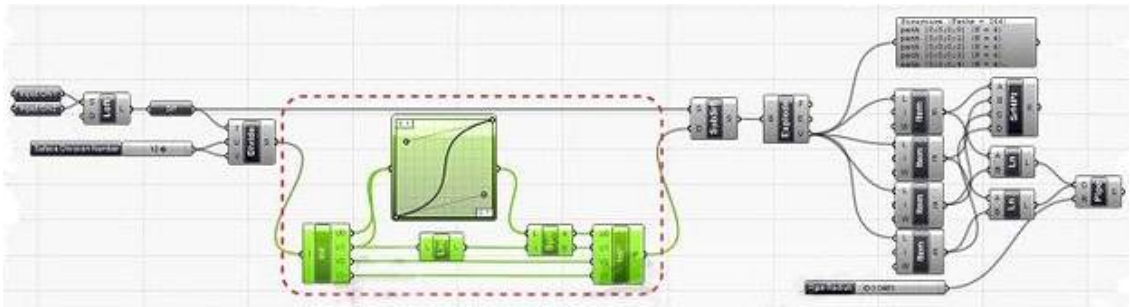
11.4 균일하지 않은 대각선 그리드 생성하기

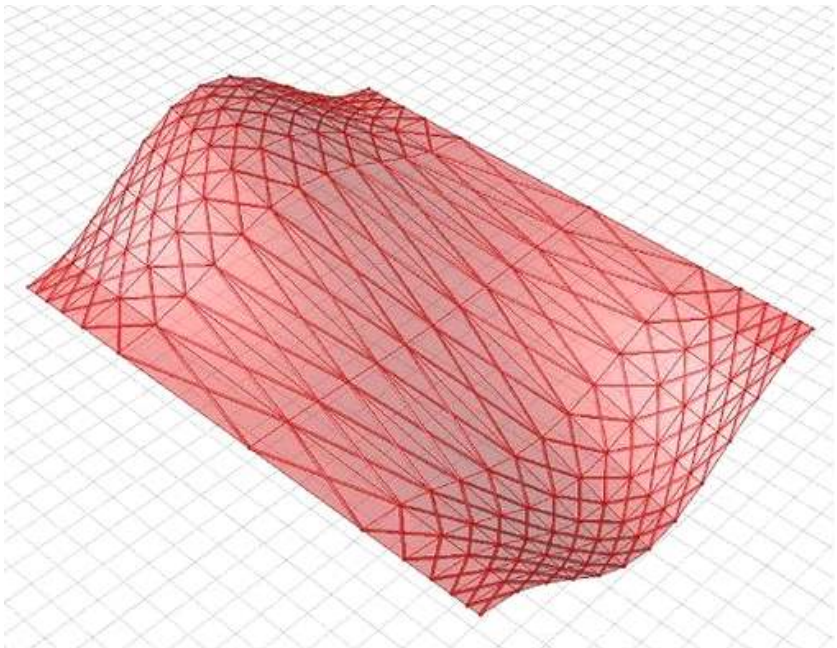
이전 예제를 통하여 분할되는 균일한 공간의 대각선 그리드 구성이 어떻게 이루어지는지 알아보았습니다. 면을 나누는 Interval을 균일하게 나누었기 때문에 대각선 그리드는 역시 균일하게 분할됩니다.

그러나 몇 개의 새로운 컴포넌트들과 Graph Mapper로 간격과 대각선 그리드의 공간을 조절할 수 있습니다.

우리는 이미 이전에 대각선 그리드 공간의 정의를 알고, 이 튜토리얼에서 하나의 대각선 그리드를 만들어 냈습니다. 아래의 이미지는 균일하지 않은 대각선 그리드의 정의가 완료된 모습입니다. 우리가 추가적으로 설치해야 할 컴포넌트들은 초록색으로 표시됩니다.

Note: 완성된 정의 파일을 보려면 Grasshopper에서 Uneven Surface Dia그리드.ghx를 열어 참조합니다.

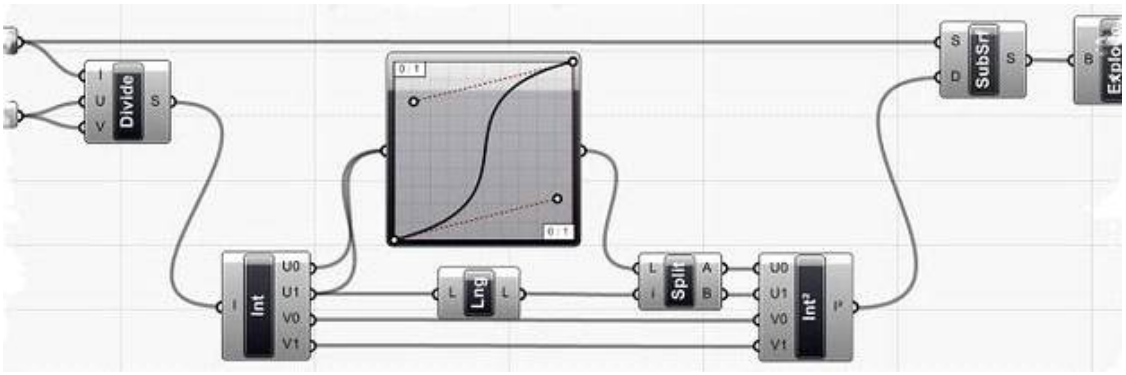




이전 예제에서 우리는 Divided Interval 컴포넌트를 Isotrim 컴포넌트에 연결하였습니다. 본 예제에서는 이 연결을 끊는 것으로부터 출발합니다.

·Isotrim-D 입력을 오른쪽 클릭한 후, "Disconnect All"을 선택합니다.

연결을 끊고 새로운 컴포넌트들을 몇 개 삽입할 예정이므로 나머지 컴포넌트들을 우측으로 이동시켜 공간을 확보합니다.
·Interval 컴포넌트를(Interval을 4개로 분해하는) 생성합니다.(Params/ Special/ Graph Mapper)
·Divide Interval-S 출력을 Interval 컴포넌트-I 입력에 연결합니다.
·Graph Mapper 컴포넌트를 생성합니다.(Params/ Special/ Graph Mapper)
·U0-출력을 Graph Mapper의 입력에 연결합니다.
·Shift 키를 누른 체로 U1-출력을 Graph Mapper 입력에 연결합니다.
·Graph Mapper를 오른쪽 클릭하고 Graph 유형을 선택합니다.
Bezier Graph가 이 상황에 가장 맞을 거라 생각되지만, 선택은 각자의 취향입니다. Bezier Handle로 Graph Mapper 컴포넌트를 이동함으로써, Graph를 조정할 수 있습니다.
Graph의 배열을 조금 조정해 보기 위해 균등한 간격으로 분포되어 있는 대각선 그리드의 배열을 변화하려고 합니다.
·List Length 컴포넌트를 생성합니다.(Logic/ List/ List Length)
·U1-출력을 List Length-L 입력에 연결합니다.
·Split List 컴포넌트를 생성합니다.(Logic/ List/ Split List)
·Graph Mapper 출력을 Split List의 L-입력 컴포넌트에 연결합니다.
·List Length-L 출력을 Split List-i 입력에 연결합니다.
·2개의 Dimensional Interval 컴포넌트를 생성합니다.(Scalar/ Interval/ Interval 2d)
·Split List-A 출력을 2개의 Dimensional 컴포넌트의 U0-입력에 연결합니다.
·Split List-B 출력을 2개의 Dimensional 컴포넌트의 U1-입력에 연결합니다.
·Decomposed Interval의 V0과 V1의 출력을 2개의 Dimensional Interval 컴포넌트의 V0과V1 입력에 각각 연결합니다.
두 개의 리스트를 Graph Mapper의 입력(단일 입력)에 연결하였으므로 Graph Mapper의 출력 단자에는 단 하나의 리스트만이 출력됩니다.
이 리스트는 먼저 연결한 144개의 U0에 해당하는 값들을 먼저 출력하고, 이어서 나중에 연결한 144개의 U1에 해당하는 값들을 따라서 출력합니다. 동일한 Graph Mapper로 두 개의 다른 리스트를 처리했지만, 결국 두 개의 다른 리스트를 필요로 하므로 결과적으로 Graph Mapper에서 나온 하나의 리스트를 두 개로 다시 나누어야 합니다.
Split List 및 Length Item 컴포넌트는 두 개의 독립된 리스트에 몇 개의 Item이 있었는지를 확인 하고 합쳐진 리스트를 절반이 되는 지점에서 분리합니다.
U와는 다르게 V에 대해서는 Graph Mapper를 사용하지 않습니다. 이제 위의 과정으로 다시 4개의 값들이 생성되었고, 이제 이 값들을 이용해서 2차원 Interval에 다시 연결하여 변화된 2차원 Interval을 생성합니다.
·2개의 Dimensional Interval-I2 출력을 Isotrim-D 입력에 연결합니다. 면의 분할이 이제 Graph Mapper에 의해 조절이 가능하게 되었습니다.
정의의 일부가 끝난 장면은 같습니다. Graph 유형은 분할되는 공간을 조정할 것입니다.; 그 다음 대각선 그리드를 살펴봅니다.
필요로 하는 면의 구성을 Bezier Curve로써 확장 시킵니다. 정의된 중간 부분의 이미지는 아래와 같습니다.


I
U
V
Divide
S
0 1
0 1
I
U0
U1
V0
V1
Int
L
Lng
L
L
I
Split
A
B
U0
U1
V0
V1
Int²
I²
S
D
SubSr
S
B

12 스크립트 소개

Grasshopper의 기능은 VB DotNet이나 C#언어를 이용해 확장이 가능합니다. 아마도 장래에는 더 많은 언어들이 지원될 수도 있을 것입니다. 사용자가 생성한 코드는 능동적으로 발생되는 Class Template 내에 위치하게 되는데, 이는 DotNet Framework에 포함되어 있는 CLR 편집자에 의해 Assembly로써 편집됩니다. 이러한 Assembly는 컴퓨터의 메모리 속에 남아 있으며 Rhino를 끄기 전까지는 사라지지 않습니다.

Grasshopper의 스크립트 컴포넌트는 Rhino의 DotNet SDK의 Class 및 함수들에 접근이 가능한데 이러한 Class 및 함수는 플러그인 개발자들이 Rhino를 위한 플러그인을 만들기 위해 사용하기도 합니다. Grasshopper는 스크립트ing 컴포넌트에 접근이 가능한 DotNet SDK에 기반을 두고 만들어졌습니다.

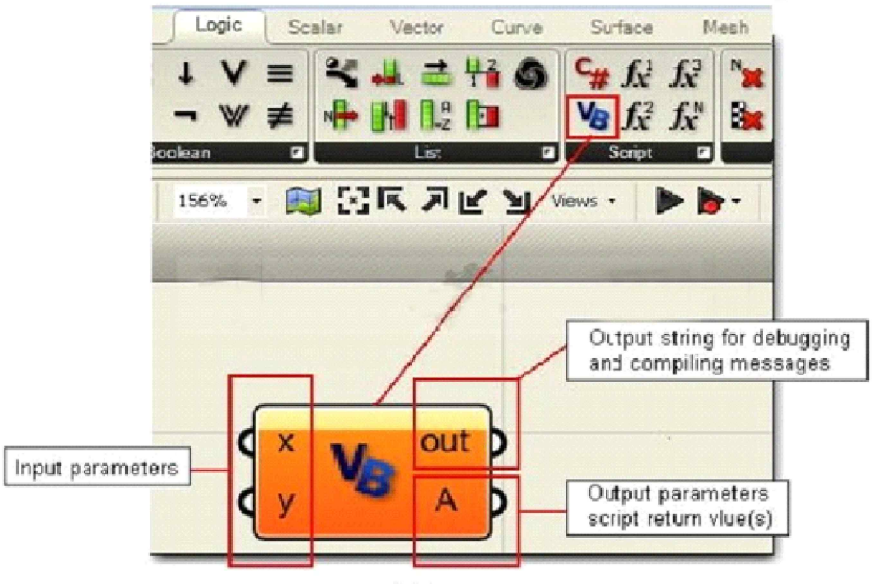
왜 스크립트 컴포넌트를 사용해야 하는가. 사실 필요가 없을 수도 있지만, 어떤 경우는 필요할 수도 있습니다. Grasshopper가 지원하지 않는 기능을 원하거나 프랙탈과 같은 반복을 요구하는 발생적인 시스템을 만들 경우에 스크립트 컴포넌트를 사용하게 됩니다.

이 교재는 스크립트 컴포넌트를 Grasshopper에서 사용하기 위한 일반적인 개론입니다. 세 가지 장으로 구성되며, 첫 장은 스크립트 컴포넌트의 Interface입니다. 두 번째는 VB DotNet에 대한 개요이며, 마지막은 Rhino DotNet SDK에 대해서입니다. 마지막 페이지에는 보다 상세한 도움을 위한 리스트가 있습니다.

13 Scripting Interface

13.1 VB DotNet Script

VB DotNet Script 컴포넌트는 Logic 탭에서 찾을 수 있습니다. 현재 두 가지 종류의 컴포넌트가 있는데, 하나는 Visual Basic, 다른 하나는 C#입니다. 장래에는 더 많은 종류의 언어가 추가 될 것입니다.
Script 컴포넌트를 생성하기 위해 컴포넌트아이콘을 생성합니다.

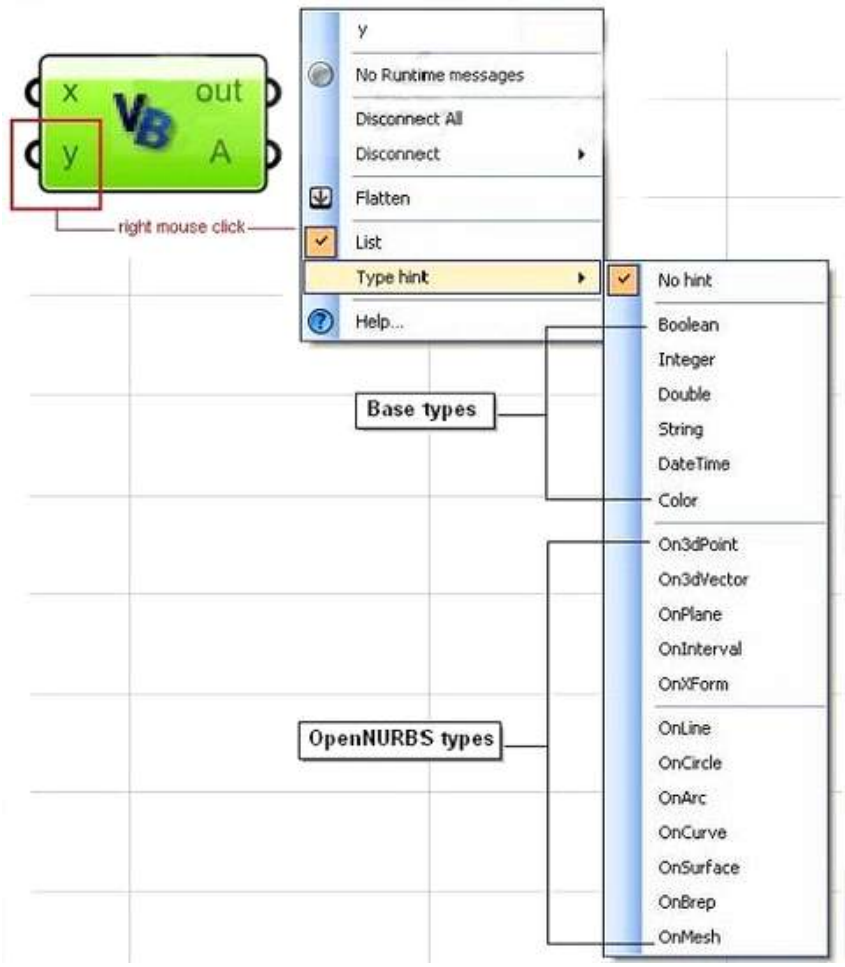


기본적으로 스크립트 컴포넌트는 두 개의 입력과 두 개의 출력을 갖고 있습니다. 사용자는 이름을 바꿀 수도 있고 입력과 출력의 숫자도 마음대로 조절할 수 있습니다.
·X: 첫 번째 입력, 일반적인 타입의 Object
·Y: 두 번째 입력, 일반적인 타입의 Object
·out: 편집 오류 메시지를 보내는 탭
·A: 스크립트의 결과물이 출력되는 탭

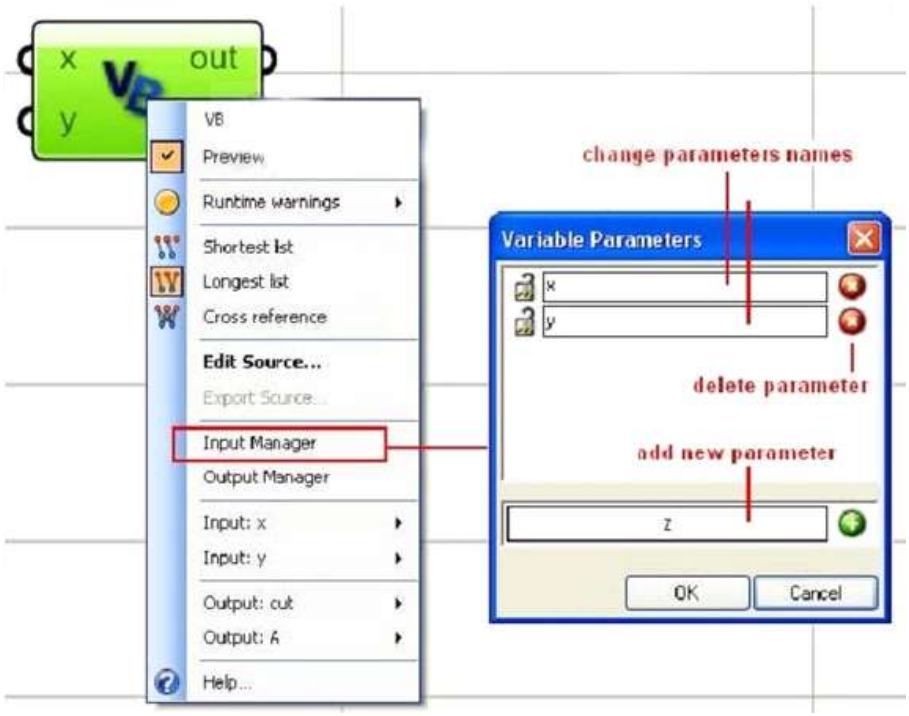
13.2 입력 Parameters

기본적으로 두 개의 입력 Parameter가 있습니다. x, y Parameter들의 이름을 바꿀 수도 있고, 삭제할 수도 있으며 형식을 지정할 수도 있습니다. 입력 Parameter에 오른쪽 클릭하면 다음 메뉴를 볼 수 있습니다.
·Parameter 이름: 클릭 후, 이름 변경 가능
·Run time Message: 오류 및 경고 메시지 확인
·Disconnect and Disconnect All: 연결 끊음
·Flatten: 데이터를 단순 배열로 변환
·List: 입력 데이터가 리스트인지, 아닌지를 설정
·유형 hint: 입력 데이터의 종류를 지정. 기본적으로 Object로 설정되어 있습니다. 코드를 더

욱 읽기 쉽고 효율적으로 관리하기 위해 입력 데이터의 형식을 지정해 주는 것이 좋습니다. "On"으로 시작되는 형식은 OpenNurbs를 의미합니다.

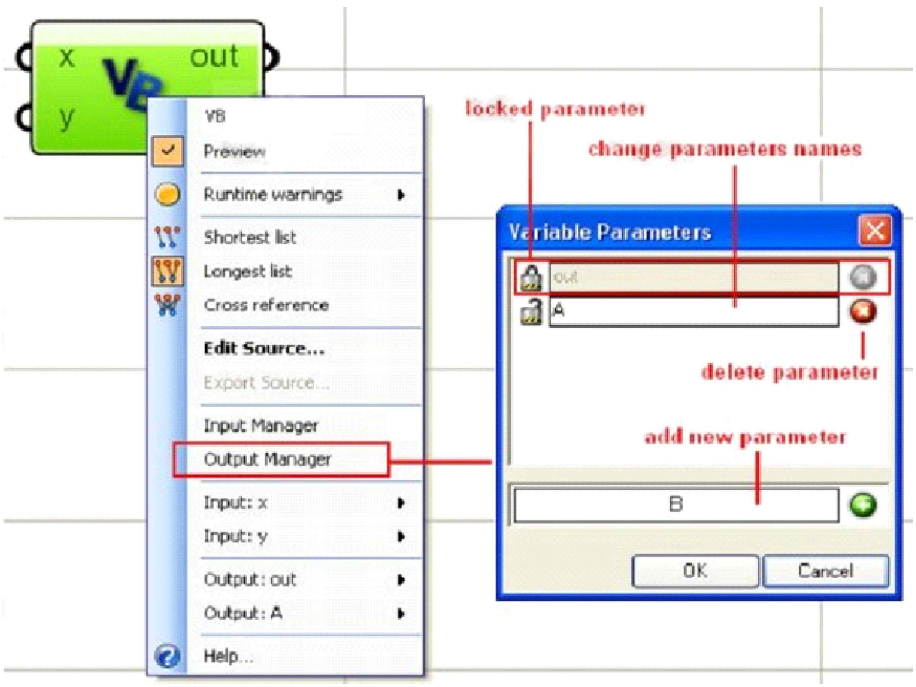


입력 Parameter들은 메인 컴포넌트 메뉴에서도 관리될 수 있습니다. 컴포넌트의 중간에 오른쪽 클릭하면, 입력과 출력의 상세 내역이 나오는 메뉴를 볼 수 있습니다.
이 메뉴를 사용하여 입력 Manager를 열고 Parameter 이름을 변경하고 새로운 Parameter를 추가하거나 삭제하는 등의 일을 할 수 있습니다.



13.3 출력 Parameters

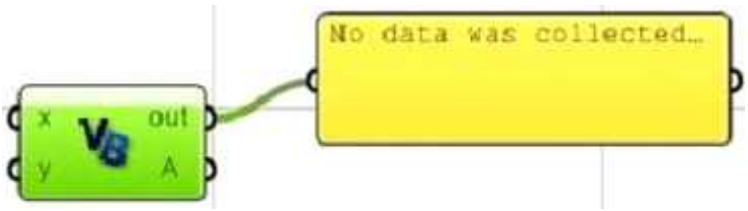
메인 컴포넌트 메뉴를 통해서 출력 Parameter를 설정하고 추가 혹은 삭제하는 것이 가능합니다. 입력 Parameter들과는 다르게 Parameter의 종류를 설정하는 탭은 없습니다. 단지 일반적인 형식인 Object로만 정의 됩니다. 아래 그림은 출력 Parameter를 어떻게 핸들링 하는지를 보여줍니다. "Out" Parameter는 코드의 오류 및 디버깅에 대한 메시지 창이므로 삭제될 수 없습니다.



13.4 Out Window and Debug Information

출력 윈도우는 "out"이라 칭하며, 디버그 정보를 보여줍니다. 이는 모든 편집 오류 및 경고를 보여줍니다. 사용자는 포스트잇 컴포넌트를 연결해 정보들을 시각화 할 수도 있습니다.

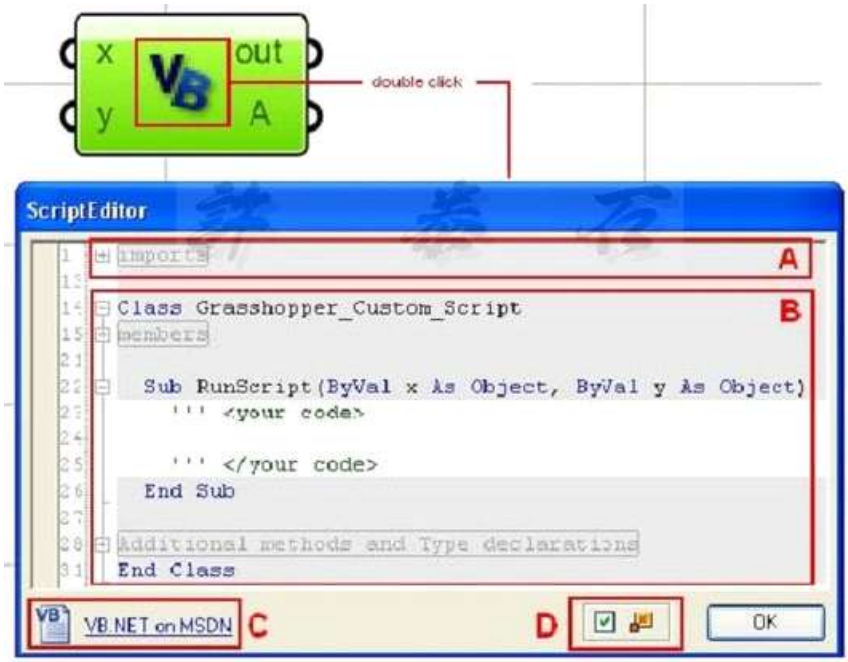
직접적으로 디버그 정보와 메시지들이 하나의 텍스트 컴포넌트로 출력단자에서 일련 되게 연결되는 것을 볼 수 있습니다.



13.5 스크립트 컴포넌트의 내부

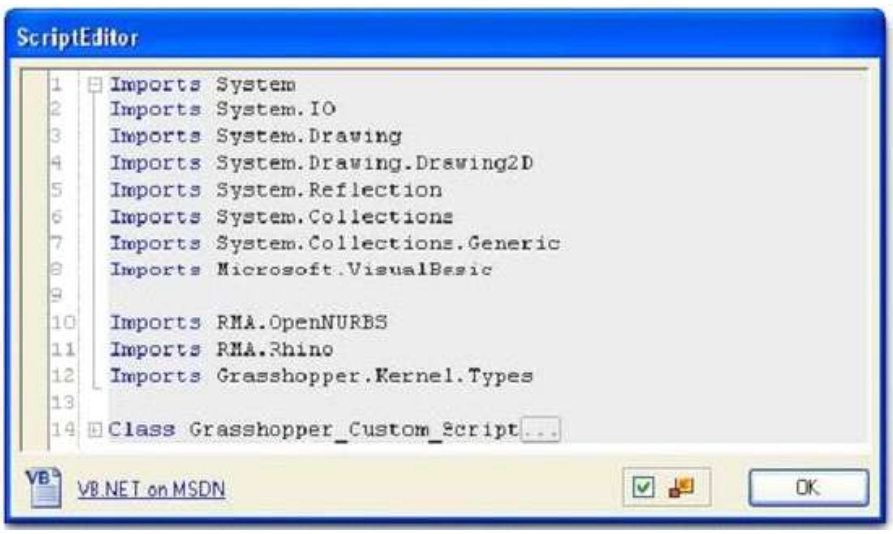
스크립트 컴포넌트를 열기 위해서는 스크립트 컴포넌트의 중앙에 더블 클릭을 합니다. 혹은 오른쪽 클릭하여 Edit source를 누릅니다. 스크립트 컴포넌트는 두 부분으로 나뉩니다.

A: Import
B: Grasshopper_Custom_스크립트 class.
C: Link to Microsoft developer network help on VB.NET.
D: Check box to activate the out of focus feature of 스크립트ing 컴포넌트.



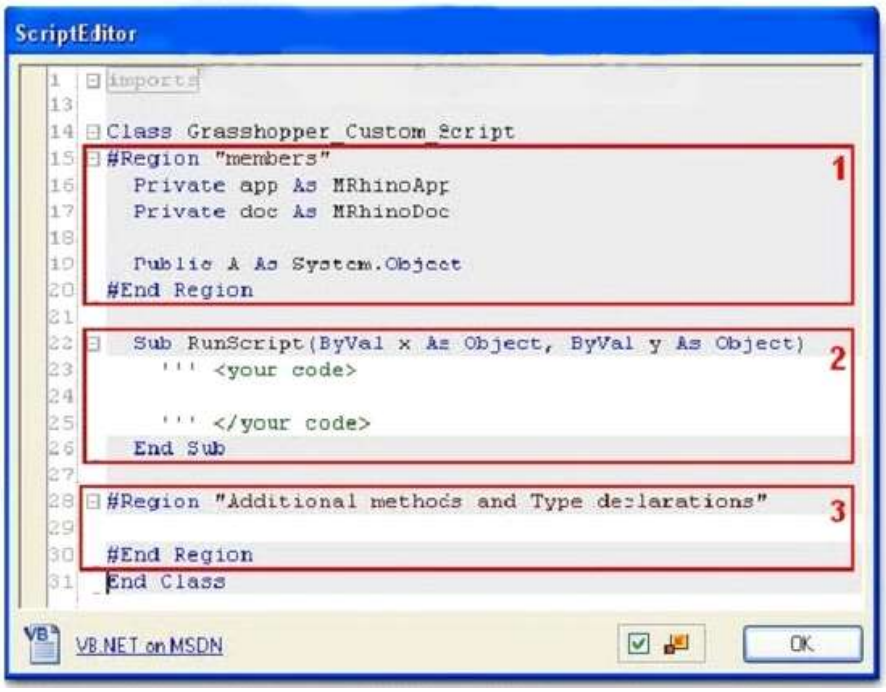
A: Import

Import는 코드 내에서 사용하게 될 외부의 dll입니다. 대부분은 DotNET 시스템 Import이고 두 개의 Rhino dll도 있습니다. RMA.openNurbs, RMA.Rhino. 이들은 모든 Rhino의 지오메트리와 기능들을 포함합니다. 또한 Grasshopper에서만 사용가능한 것들도 포함됩니다.



B: Grasshopper_Custom_스크립트 class.

Grasshopper_Custom_스크립트 class는 세 부분으로 나뉩니다.

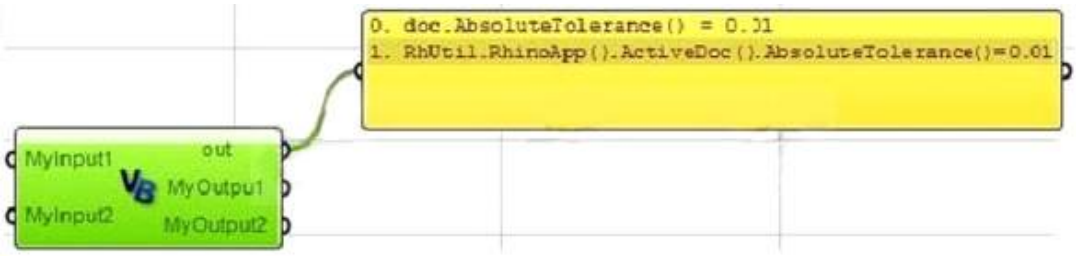


1. Members: 이것은 두 개의 참조를 포함합니다. 하나는 현재 Rhino app이며 다른 하나는 활성화 된 문서 doc입니다. Rhino 어플리케이션(app)과 다큐먼트는(doc) Rhino Util을 통해서 직접적 접근이 가능합니다. 멤버 영역은 또한 출력이나 Return 값을 포함합니다. Return 값은 일반적인 시스템 형식으로 정의되며 사용자는 이를 변경할 수 없습니다.

2. Run스크립트: 이것은 주된 스크립트 영역이며 사용자가 자신의 코드를 입력하는 곳입니다.

3. Additional methods and 유형 declarations: 함수나 형식을 추가하는 곳입니다.

아래의 예는 Document absolute tolerance에 접근하는 두 가지 방법을 보여줍니다. 첫 번째 Document reference를 이용합니다. 두 번째는 tolerance 값을 출력 윈도우로 내보낼 시에 두 개의 함수 모두 같은 결과를 보여줍니다. 또한 본 예제에서는 두 가지의 출력이 있습니다. 그들은 Script Class의 멤버 영역에 나열되어 있습니다.




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

  Public MyOutput1, 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 개요

VB.NET에 관해서는 인터넷이나 책을 통해 많은 정보를 구할 수 있습니다. 다음은 코드를 작성하는데 있어서 가장 기본적인 것에 대한 개요입니다.

14.2 설명

코드에 가능한 많은 설명이나 주석을 첨가하는 것은 좋습니다. 자신이 만든 코드를 얼마나 빨리 잊게 되는지 알게 될 것입니다. VB.NET에서는 홑 따옴표를 사용하여 그 뒤의 내용들이 모두 부가 설명임을 설정합니다. Grasshopper에서 부가 설명은 회색으로 나타냅니다.

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

14.3 변수

변수는 데이터를 담는 그릇이라고 생각하면 됩니다. 다른 종류의 변수들은 변수의 종류에 따라 다른 크기를 가지고 있습니다. 예를 들어 int32(정수)의 경우는 32비트를 메모리에서 할당합니다. 변수가 한번 정의되면, 나머지 코드에서 그 변수의 내용물을 언제든지 추출하여 사용 가능합니다. 변수 x는 정수 32비트의 변수이며 초기 값은 10입니다. 그 다음 새로운 정수 값 20을 x에 대입합니다. 아래는 VB.NET에서의 모습입니다.

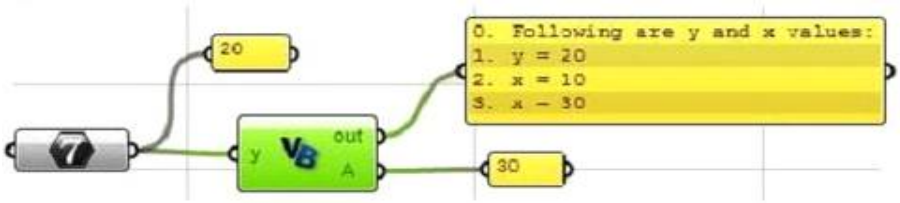
```
Dim × as Int32 = 10
'If you print the 결과 값 of × at the 점, then you will get 10
× = 20
'From now on, × will return 20
```

아주 많이 사용 되는 다른 예입니다.

```
Dim × as Double = 20.4          'Dim keyword means that we are about to define a variable
Dim b as Boolean = True         'Double, Boolean and String are all examples of base or
Dim 이름 as String = "Joe"      'system define 유형
```

다음의 예제는 세 가지 변수를 사용할 것입니다.:
x :는 정수 변수로써 코드 내에서 정의됩니다.
y :는 정수 변수로써 입력의 형태로 외부에서 전달됩니다.
A :는 출력 변수입니다.

이 예제는 변수 값을 출력 윈도우에 프린트 합니다. 이미 언급했듯이, 출력 윈도우의 메시지를 확인하는 것이 좋습니다.




```
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
```

변수 이름을 쉽게 알아볼 수 있도록 지정하면 코드가 읽기 편해지고, 오류를 고치기 쉬워집니다. 우리는 계속해서 좋은 코드를 지정하는 방법을 이번 순서의 예시들을 통하여 연습해 나갈 것입니다.

14.4 목록과 정렬

정렬은 VB.NET.에서 많은 방법으로 정의 내려집니다. 일차원 혹은 다차원의 정렬들이 존재하며, 정렬의 크기를 정의하거나 동적 정렬을 사용할 수 있습니다. 정렬 안의 요소의 앞의 수를 알고 있다면, 다음과 같은 방법으로 정렬의 크기를 만들 수 있습니다.

```
'One dimensional array
Dim myArray(1) As Integer
myArray(0)=10
myArray(1)=20
```

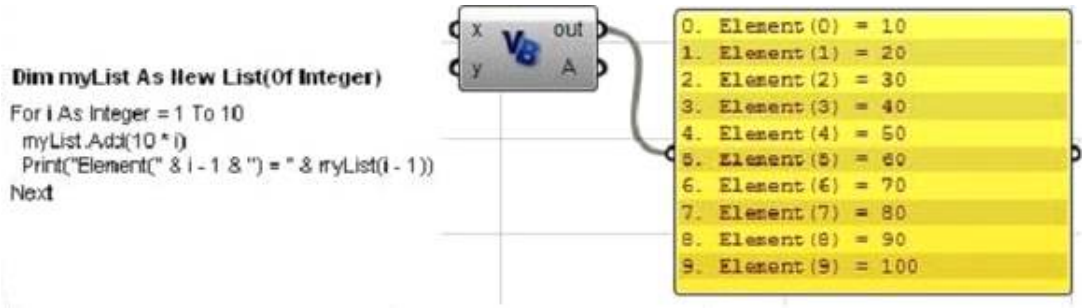
```
'Two-Dimensional array
Dim my2DArray(1,2)As Integer
my2DArray(0,0)=10
my2DArray(0,1)=20
my2DArray(0,2)=30
my2DArray(1,0)=50
my2DArray(1,1)=60
my2DArray(1,2)=70
```

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

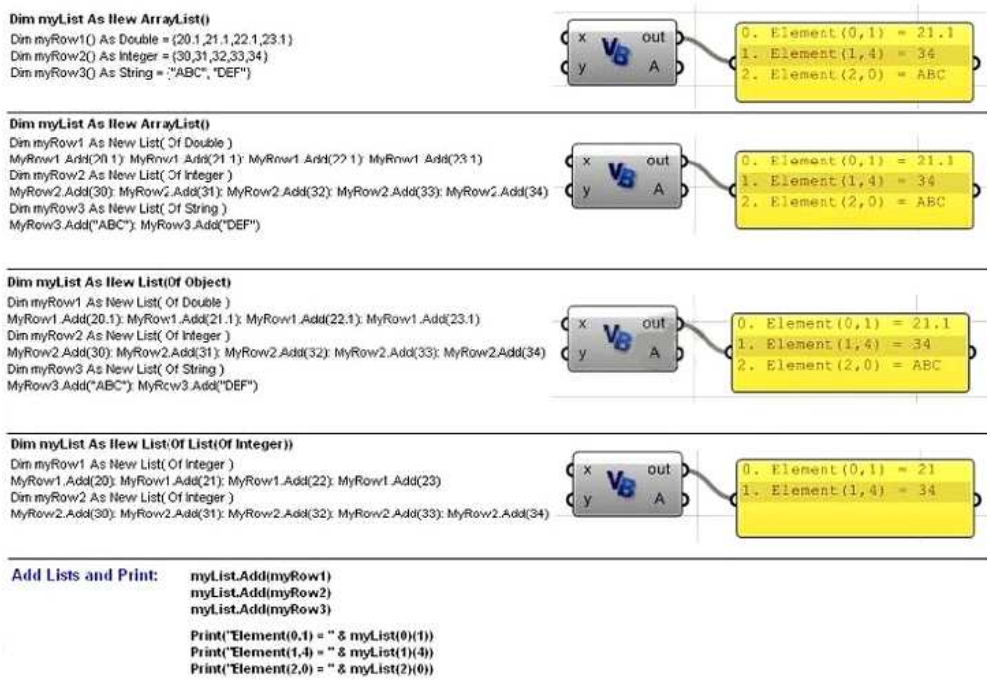
```
'Declare and assign 결과 값s
Dim my2DArray(,)As Integer
={{10,20,30},{40,50,60}}
```

VB.NET은 0에서 9까지 다차원 정렬과 같이 10개의 요소로써 정렬됩니다.

일차원 동적 정렬에서는 보여 지는 예시와 같이 새로운 "목록"을 명령하고 요소를 추가할 수 있습니다.



다음의 예시에서는 중첩된 목록이나 정렬 목록을 동일한 혹은 혼합 된 형식의 동적 다차원 정렬로 명령할 수 있음을 볼 수 있습니다.



14.5 조작

많은 연산자들이 VB.NET.에 내제되어 있습니다. 그 연산자들은 하나 혹은 여러 개의 피 연산자를 연산합니다. 다음은 일방적인 연산자 테이블을 보여줍니다.

Type	Operator	Description
Arithmetic Operators	^	Raises a number to the power of another number.
	*	Multiplies two numbers.
	/	Divides two numbers and returns a floating-point result.
	₩	Divides two numbers and returns an integer result.
	Mod	Divides two numbers and returns only the remainder.
	+	Adds two numbers or returns the positive value of a numeric expression.
	-	Returns the difference between two numeric expressions or the negative value of a numeric expression
Assignment Operators	=	Assigns a value to a variable
	^=	Raises the value of a variable to the power of an expression and assigns the result back to the variable.
	*=	Multiplies the value of a variable by the value of an expression and assigns the result to the variable.
	/=	Divides the value of a variable by the value of an expression and assigns the floating-point result to the variable.
	₩=	Divides the value of a variable by the value of an expression and assigns the integer result to the variable.
	+=	Adds the value of a numeric expression to the value of a numeric variable and assigns the result to the variable. Can also be used to concatenate a String expression to a String variable and assign the result to the variable.
	-=	Subtracts the value of an expression from the value of a variable and assigns the result to the variable.
	&=	Concatenates a String expression to a String variable or property and assigns the result to the variable or property.
Comparison Operators	<	Less Than
	<=	Less or equal
	>	Greater than
	>=	Greater or equal
	=	Equal
	<>	Not equal
Concatenation Operators	&	Generates a string concatenation of two expressions.
	+	Concatenate two string expressions.
Logical Operators	And	Performs a logical conjunction on two Boolean expressions
	Not	Performs logical negation on a Boolean expression
	Or	Performs a logical disjunction on two Boolean expressions
	Xor	Performs a logical exclusion on two Boolean expressions

14.6 조건부 상황

조건문과 함께 실행되는 게이트와 코드 블록이 있습니다. 조건문은 거의 "If"와 "IF<condition>Then<code>IF"형식을 따라 구성될 때 사용되어집니다.

```
'Single line statement doesn't need End if: condition=(x<y), code=(x=x+y)
If x < y Then x = x + y
'Multiple line needs End If close the block of code
If x < y Then
  x = x + y
End If
```

이것은 또한, "Else If ... Then" 그리고 "Else"를 사용하여 대안 코드 블록을 실행시킬 수 있습니다. 예를 들어,:

```
If x < y Then
  x = x + y         'execute this line then go to the step after "End If"
Else If x > y Then
  x = x - y         'execute this line then go to the step after "End If"
Else
x =2*x              'execute this line then go to the step after "End If"
End If
```

또한, "Select Case"라는 명령어도 있습니다. 표현의 값을 기본으로 한 코드의 여러 블록들을 실행 시킬 때 사용되어 집니다.

```
Select Case index
  Case 0              'If index=0 the execute next line, otherwise go directly to the next case
   x= x * x
  Case 1
   x= x ^ 2
  Case 2
   x= x ^ (0.5)
End Select
```

14.7 Loop

Loop 조건이 만족하는 동안은 Loop 안에 있는 코드 블록의 실행을 반복할 수 있습니다. 여러 종류의 Loop가 존재하며, 가장 많이 쓰이는 두 가지 종류의 Loop에 대해서 설명합니다.

"For ... Next" Loop
Loop를 만들 때 가장 많이 사용되어지는 방법입니다. 구성은 다음과 같습니다.

```
For < index = start_결과 값 > To < end_결과 값 > [Step < step_결과 값 >]

'For loop body starts here
[ statements/code to be executed inside the loop]
[Exit For]    'Optional: to exit the loop at any 점
[ other statements ]
[ Continue For ]          'optional: to skip executing the remaining of the loop statements.
[ other statements ]
'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_결과 값
'If index passed end_결과 값, then exit loop and execute statements following "Next"
Next
[ statements following the For Loop ]
```

다음 예제는 장소의 배열을 통해 반복되는 Loop를 사용합니다.

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

'Loop index
Dim i As Integer
```

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

'Loop starting from 0 to count -1 (count = 3, but last index of the places_list = 2)
For i=0 To count-1
  place = places_list(i)
  print( place )
Next
```

만약의 배열 안의 개체를 루핑하게 되면, "For ... Next" Loop를 인덱스를 사용하지 않고 배열의 요소를 통해 반복하는 데 사용할 수 있습니다.
위의 예제는 이렇게 다시 쓰여 질 수 있습니다.:

```
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" Loop
이 Loop 유형 또한 널리 사용되어 집니다. 이 Loop의 구조는 다음과 같습니다.

```
While < some condition is True >

'While loop body starts here
[ statements to be executed inside the loop ]
[ Exit While ]          'Optional to exit the loop
[ other statements ]
[ Continue While ]    'optional to skip executing the remaining of the loop statements.
[ other statements ]
'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
[ statements following the While Loop ]
```

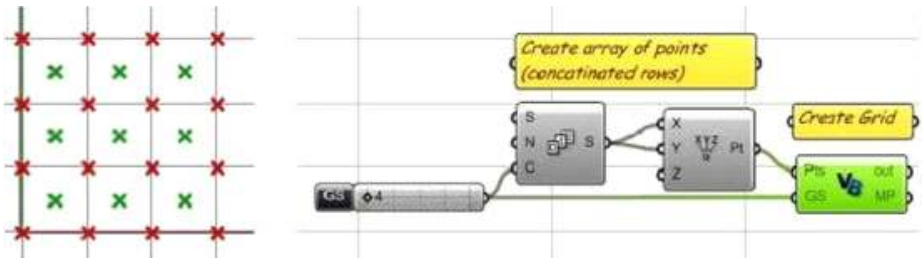
While Loop를 사용해서 조금 전 장소 예제를 다시 작성해 봅니다.

```
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 다중(중첩) Loop

중첩된 Loop들은 다른 Loop들을 포함하고 있는 Loop를 말합니다. 예를 들어 점으로 된 그리드를 가지고 있다면, 각 지점의 목록을 얻기 위해 중첩 Loop를 사용합니다. 다음 예제는 어떻게 1차원적 점들의 배열이 2차원 그리드로 들어가게 되는지를 보여줍니다. 그 다음 그리드 내부의 중간 지점을 찾도록 진행합니다.



스크립트는 두 부분으로 구성됩니다.:
- 첫째, 1차원 배열을 우리가 "그리드"라고 부르는 2차원 배열로 변화시켜 줍니다.
- 둘째, 그리드 내부의 중간 지점을 찾도록 돕습니다.

두 부분 모두 중첩 Loop를 사용하게 되며, 다음은 Grasshopper의 스크립트를 정의합니다.

```
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 Sub와 함수

RunScript는 모든 Script 컴포넌트 중 가장 중요한 기능입니다. Grasshopper에서 기보의 Script 컴포넌트를 열었을 때 보여 지는 모습입니다.

Sub RunScript(ByVal × As Object, ByVal y As Object)
    '''<your code...>
End Sub
Sub ... End Sub: 코드의 블록 기능을 닫는 키워드
"Run스크립트": Sub의 이름입니다.
"(...)": 괄호 후에 오는 Sub 이름의 입력 매개변수입니다.
"ByVal × As Object,...": 입력 매개변수로 불리는 것 들입니다.

각 입력 매개변수들은 다음 정의가 필요합니다.:
·ByRef or ByVal: 매개변수가 값 또는 Ref.에 의해 전달될 경우 구체화합니다.
·매개변수의 이름

·매개변수의 종류가 "As"키워드에 의해 선행되어 집니다.

Grasshopper의 Run스크립트에서의 모든 입력 매개변수들은 값을 전달합니다.(ByVal keyword).
이 말은 그 것들이 본래 입력들의 복사본이며, Script안에서의 매개변수의 변화는 본래의 입력에서는 영향을 미치지 않습니다. 그러나 만약에 더해진 첨자들이나 함수들은 스크립트 컴포넌트 안에서 정의하게 된다면 Ref.에 의해서 전달할 수 있습니다.(ByRef) Ref.에 의해서 전달되는 매개변수의 의미는 함수 안의 어떤 매개변수의 변화도 함수를 빠져나갈 때 전달되어진 원래의 값을 변하게 합니다.

RunScript 안에 모든 코드를 포함할 수 있습니다. 그러나 외부 Subs와 함수가 정의를 내리기 위해 필요합니다. 외부 함수를 이용하는 이유는:
·중요함수 코드를 단순화 시켜줍니다.
·코드를 읽기 쉽게 만들어줍니다.
·일반적인 함수성을 분리하고 재사용하게 해줍니다.
·특별한 함수를 정의합니다.

Sub와 함수의 다른 점은 Sub는 돌아오는 값이 필요 없지만, 함수는 한 개의 대응하는 결과를 돌아오게 합니다. 기본적으로 함수 이름에 값을 지정할 수 있습니다. 예를 들어,:

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

여기서 말하는 것은, 함수에 돌아오는 값을 가지지 않아도 된다고 해석할 수 있습니다. Subs는 "ByRef"를 지나는 입력 매개변수를 통해서 가능합니다. 다음에 오는 "rc"는 돌아오는 값에 사용됩니다.

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

여기서 호출 함수를 사용하면 함수와 얼마나 Sub이 비슷한 지를 보여줍니다.

```
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 )        'Assign function result to "rc"
AddSub( x, y, rc )              'rc is passed by reference and will have addition result
```

중첩 Loop 섹션에서 우리는 중간점들을 그렸고 점들의 목록에서 그리드를 만드는 예를 들었습니다.
이러한 각각의 두 함수의 함수성을 외부 Sub 안에서 별도로 구별해주고, 다시 사용할 수도 있습니다. 외부함수를 이용한 예제입니다.

```
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
```

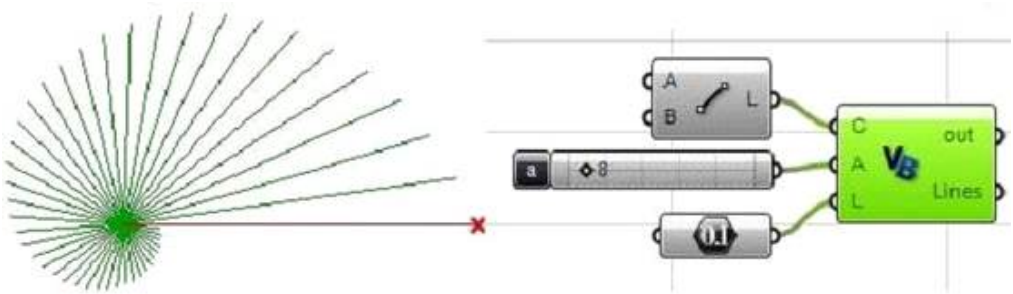
여기는 각각의 Subs가 연장되어 질 때의 모습입니다.:

```
#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 회귀

회귀 함수들은 막히는 조건이 충족되어지는 특별한 함수들을 일컫습니다. 회귀는 일반적으로 데이터 찾기, 하위분류 그리고 생성 시스템을 위해 쓰여 집니다. 우리는 어떻게 회귀가 사용되어 지는 것인지에 대한 예제를 살펴봅니다. 좀 더 많은 관련 예제는 Grasshopper Wiki와 갤러리 페이지를 확인할 수 있습니다.

다음 예제는 입력된 선의 작은 부분을 사용하여 주어진 각으로 회전합니다. 선의 길이가 최저값 이하가 될 때까지 지속합니다.
입력 매개변수들은:
·시작되는 선(C).
·호 안의 각(A). Slider는 각을 각도로 표시하지만, 호로 변환합니다.
·최저 길이(L) - 정지 조건
출력은:
·선의 정렬



회귀 예제를 반복적으로 살펴보면서 비교합니다.

회귀의 해결 방법입니다. "DivideAndRotate" sub 안에 다음을 입력합니다.:
·정지 조건을 Sub의 출구로 합니다.
·동일한 함수를 불러옵니다.(회귀 함수는 자신들을 불러옵니다.)

```
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
```

·"모든 선"은 Ref를 지나 계속해서 새로운 선을 목록에 추가합니다.
이것을 함수성을 가진 "while" 루프를 사용하면서 같은 해답을 반복합니다.

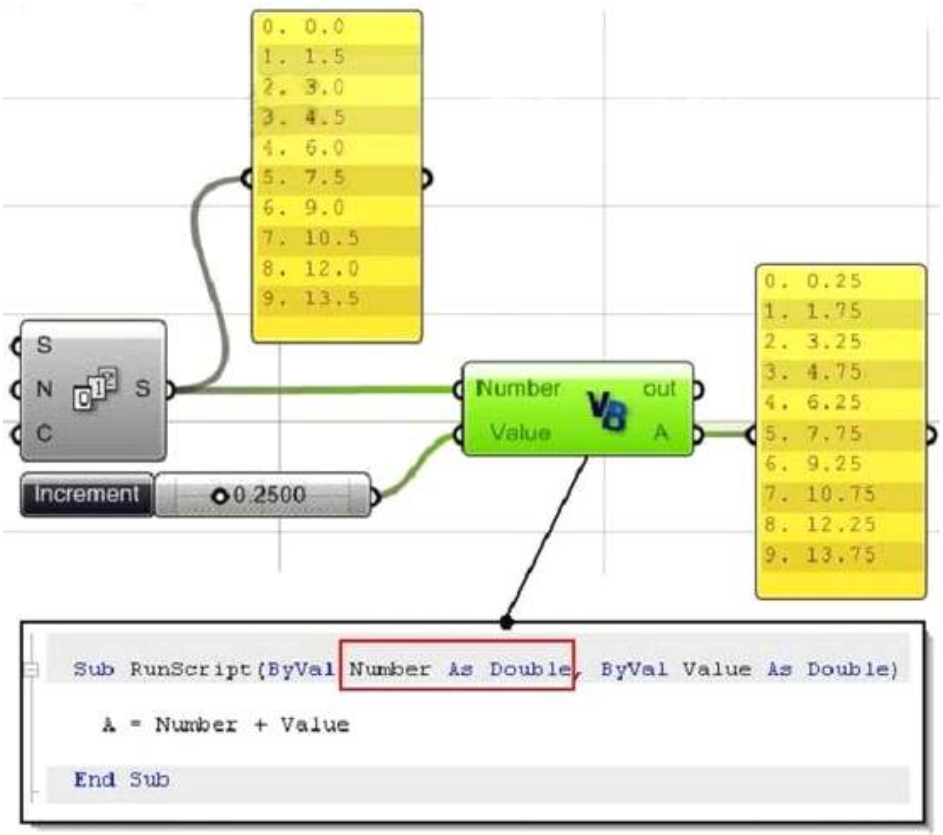
```
Sub RunScript(ByVal C As OnLine, ByVal A As Double, ByVal L As Double)

  '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 Grasshopper에서의 목록의 처리과정

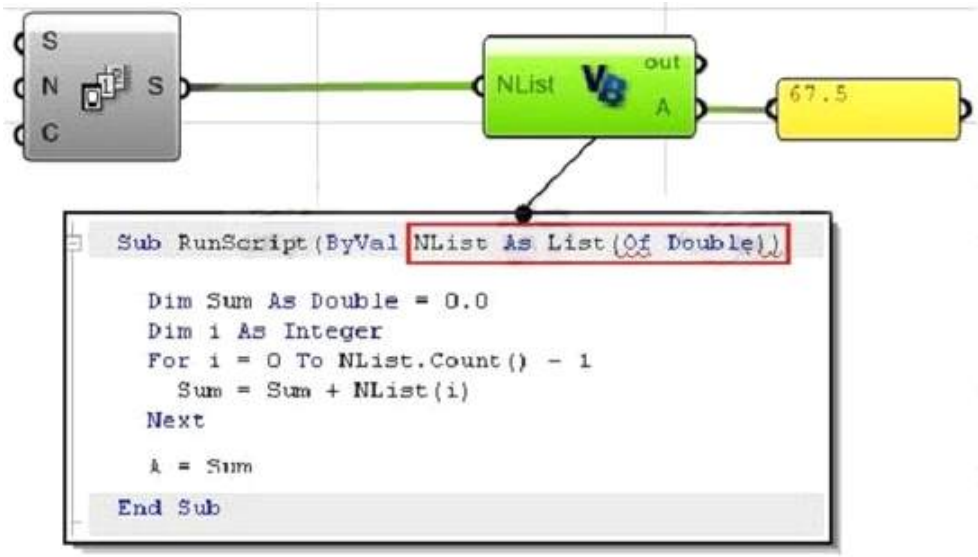
Grasshopper 스크립트 컴포넌트는 입력 처리목록을 두 가지 방법으로 할 수 있습니다.:
1. 하나의 입력 값을 한 번에 처리합니다.(컴포넌트는 입력 정렬 안의 수와 같은 수로 불립

니다.)
2. 모든 입력 값을 한 번에 처리합니다.(컴포넌트는 한 번만 부릅니다.)
만약 다른 요소들을 독립적인 목록 안에서 처리가 필요하다면, 첫 번째 방법으로 다가가는 것이 쉬울 것입니다. 예를 들어, "10"씩 증가하는 가지는 각각의 숫자 목록을 가지고 싶다면, 첫 번째 방법을 사용할 수 있습니다. 하지만 모든 요소를 더하는 합의 함수가 필요할 경우에는 두 번째 방법을 사용해야 합니다. 모든 목록을 하나의 입력으로 보냅니다.

다음에 나오는 예제는 첫 번째 방식을 이용하여 하나의 입력을 한 번에 처리하는 정보 목록의 처리과정을 보여줍니다. 이 경우에는 Run스크립트 함수가 10회 불리는데,(각각의 수에 한번씩) 중요한 것은 "수" 입력 매개변수가 "Double"과 같고 "List(of Double)"와 반대되는 것으로 전달되는 것을, 다음 예제에서 볼 수 있습니다.



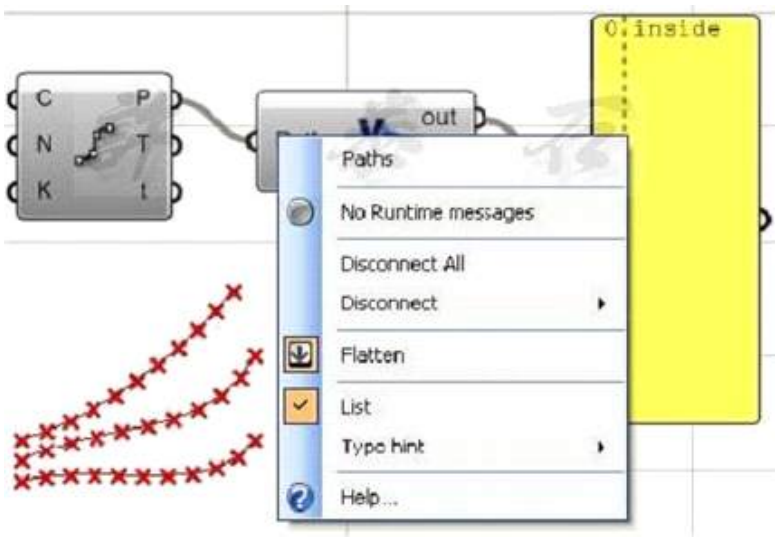
다음 예제에서 숫자 목록을 입력하고, 입력 매개변수에 오른쪽 클릭하여 "List"에 체크합니다. Run스크립트 함수는 한 번만 불러집니다.



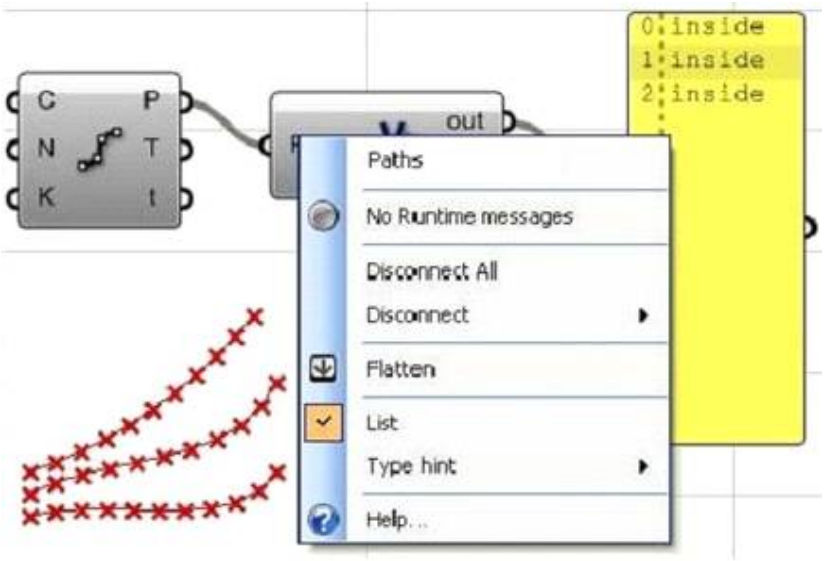
14.12 Grasshopper에서의 다차원 정보 처리

트리(혹은 다차원정보)는 하나의 요소에 한 번에 처리할 수 있거나 한 번에 한 가지씩 또는 한 번에 모든 통로가 처리되어 질 수 있습니다. 예를 들어 세 개의 Curve들을 10개로 나누고자 할 때, 우리는 11개의 점들을 가진 세 개의 가지 또는 통로 구조를 갖게 됩니다. 이것을 입력하게 되면 다음을 갖게 됩니다.

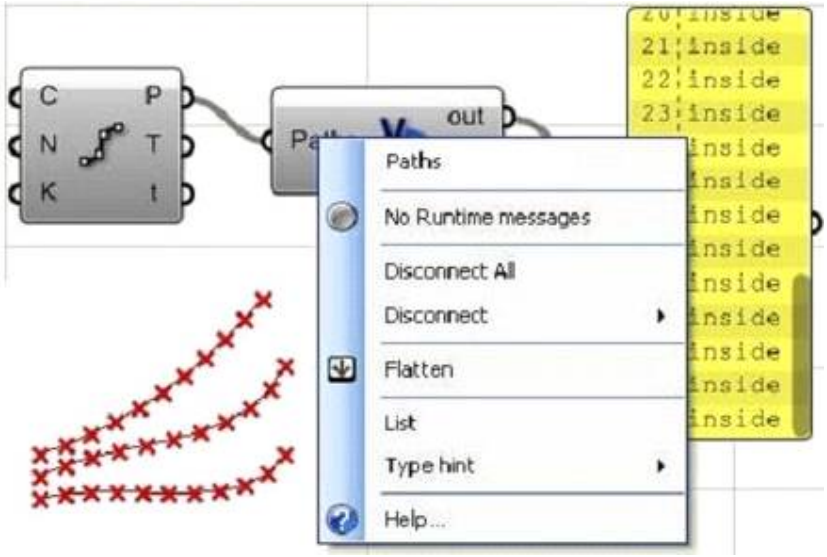
A: "Flatten"과 "List"가 둘 다 체크 되어있다면, 컴포넌트는 한 번만 소환되고, 모든 점들의 플랫 목록이 전달됩니다.



B: 만일 "List"에만 체크가 되어있다면, 컴포넌트는 각 세 번씩 소환되고, 하나의 곡선의 나누어진 점들의 목록에 전달됩니다.



C: 아무것도 체크 되어 있지 않을 땐, 함수는 각각의 나눠진 점들을 위해 한 번씩 소환됩니다.(이 예제에서 함수가 33번 소환됩니다.) "Flatten"에만 체크되어 있을 때에도 동일합니다.



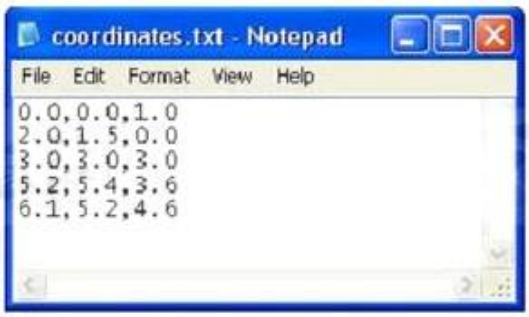
이것은 VB 컴포넌트 안의 코드입니다. 기본적으로 "inside"란 단어의 입력만으로 컴포넌트의 소환을 대신 할 수 있습니다.

```
Sub RunScript(ByVal Paths As Object)

  Print("inside")

End Sub
```

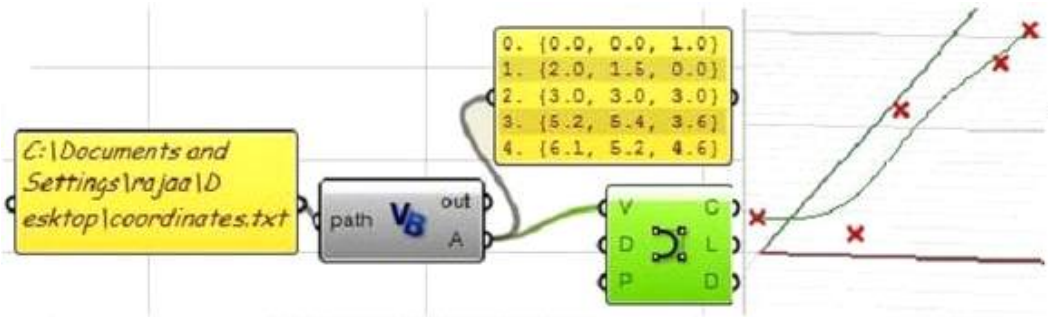
14.13 파일 I/O

VB.NET안에 있는 파일을 읽어오거나 파일에 쓸 수 있는 많은 방법이 있으며, 많은 튜토리얼과 문서가 인터넷과 서적으로 많이 나와 있습니다. 일반적으로 파일읽기는 다음 내용을 포함합니다.:
- 파일을 엽니다. 일반적으로 포인트로 향하는 경로가 필요합니다.
- 문자열을 읽습니다. (전체의 문자열이나 각각의 선)
- 몇몇의 기호문자를 사용하여 문자열을 표시합니다.
- 결과를 저장합니다.

텍스트 파일에서 문장을 해석하는 간단한 예제입니다. 다음에 보이는 텍스트 파일로 각각의 선을 하나의 선으로 읽고, 처음 값을 X좌표로, 두 번째 값을 Y좌표로, 세 번째 값을 Z점으로 읽습니다. 그 3가지 점들을 Curve 컨트롤 포인트로 사용합니다.



VB 컴포넌트는 파일의 경로를 읽을 수 있는 문자열을 입력으로 출력은 On3dPoint로 합니다.



스크립트 컴포넌트 안의 코드가 여기 있습니다. 거기에 파일 출구와 컨텐츠를 가지고 있는지 알아보는 몇 가지 오류 코드가 있습니다.:

```
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. SDK

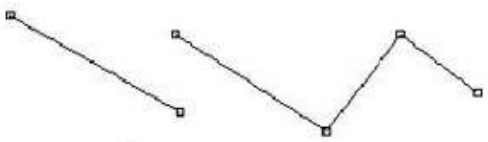
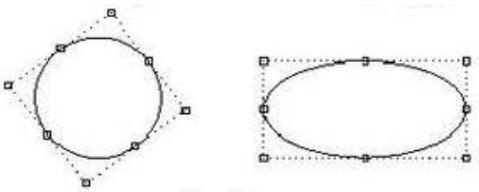
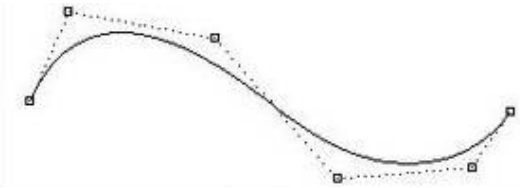
15.1 개관

Rhino.NET SDK는 OpenNurbs 지오메트리와 유틸리티 함수로 접근하게 합니다. NET SDK를 다운로드 할 때 Help 파일도 같이 따라오게 되는데, NET SDK를 사용하는데 도움이 됩니다. 이곳으로 가면 다운 받을 수 있습니다.
http://en.wiki.mcneel.com/default.aspx/McNeel/Rhino4DotNetPlugins.html
이번 섹션에서는 SDK가 Rhino 지오메트리와 유틸리티 함수를 처리하는 부분에 초점을 맞춰 Grasshopper의 VB 스크립트 컴포넌트를 사용해서 어떻게 지오메트리가 형성되고 조종되어 지는지 예를 보여줍니다.

15.2 Nurbs 이해하기

Rhino는 Nurbs 모델러로서 Curve와 Surface를 Non Uniform Rational Basic Spline(Nurbs)를 사용해서 정의를 내리게 됩니다. Nurbs는 Curve와 Surface의 정확한 수학적 표현이며 직접적으로 편집할 수 있는 기능을 갖고 있습니다.
http://en.wikipedia.org/wiki/Nurbs
Nurbs에 대하여 좀 더 알 수 있습니다.
기본적인 Nurbs에 대한 이해는 SDK 과정과 함수를 사용하는 데에 좀 더 실용적인 도움이 됩니다.
각(도), 컨트롤 포인트, 매듭, 값을 구하는 룰, 4가지 방법으로 Curve를 정의할 수 있습니다.

- 범위
전부 양의 수, 1, 2, 3, 4, 5와 같습니다.
Rhino에서도 1~11도 사이의 등급을 사용합니다. 몇 Curve들과 각의 예시들입니다.:

	선과 폴리라인은 1도 Nurbs Curve
	원형과 타원형은 2도 Nurbs Curve 유리수이거나 불균등한 Curve
	자유 곡선은 일반적으로 3도 Nurbs Curve 4,5도 또한 일반적이며, 나머지는 많은 가설

- 컨트롤 포인트

Nurbs Curve의 컨트롤 포인트는 적어도 하나 이상의 포인트를 갖습니다.(Degree + 1) Nurbs Curve의 모양을 바꾸기 위한 가장 일반적인 방법은 Curve의 컨트롤 포인트를 직접 움직이는 것입니다. 컨트롤 포인트와 조합되는 수가 있는데 그것을 weight라 칭합니다. 몇몇을 제외하고는 weight는 양수로 표현됩니다. 하나의 Curve가 같은 weight(보통1)를 가지고 있을 때, Curve는 무리수라 불립니다. 앞으로 Grasshopper에서 weight의 변화와 컨트롤 포인트의 상관관계를 예제를 통해 살펴봅니다.

- 매듭과 매듭 Vector

각각의 Nurbs Curve는 매듭 Vector라 불리는 수의 나열을 갖습니다. 매듭은 조금 이해하기 까다롭지만, 다행히도 SDK 함수가 이해를 돕습니다.
매듭 Vector에 대한 몇 가지의 유용한 정보를 살펴봅니다.

- 다양한 매듭

매듭 값의 회수가 중복 되었을 때, 우리는 매듭의 다양성이라 부릅니다. 어떤 매듭 값도 Curve의 각보다 더 복제될 수는 없습니다. 여기에 매듭에 대해 알아두면 좋은 것들이 있습니다.

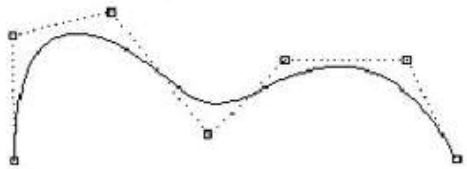
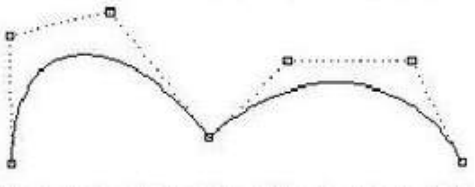
- 다중성 매듭

Curve의 각과 동등하게 중복됩니다. 채워진 Curve는 양쪽 끝에 다중성을 가지고 있으며 이것 때문에 Curve 끝에 컨트롤 포인트가 Curve와 포인트에 동시에 생기게 됩니다. 만약 거기에 전체 중복 매듭이 매듭 Vector 중간에 있다면, Curve는 컨트롤 포인트를 지나게 될 것이며, 비틀어질 수 있습니다.

- 단순 매듭

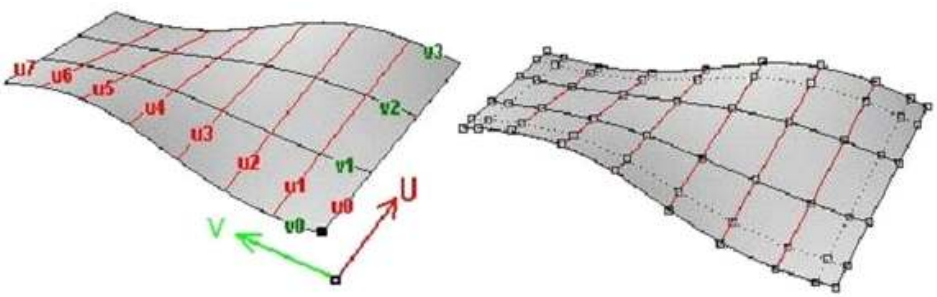
오직 하나의 값을 갖는 매듭입니다.

- 동일한 매듭 Vector는 두 가지 조건을 만족합니다.:
1. 매듭의 수 = 컨트롤 포인트 수 + degree - 1
2. 매듭이 전체 중복 매듭과 같은 시작은 단순 매듭에 의해 따라오고, 전체 중복 매듭으로 종결되는데, 값은 증가하고 같은 크기의 공간을 갖게 됩니다. 이것은 일반적인 채워진 Curve입니다. 주기 Curve는 다르게 사용되어 집니다.
여기 컨트롤 포인트가 활성화 된 다른 Vector 값을 갖는 두 개의 Curve가 있습니다.

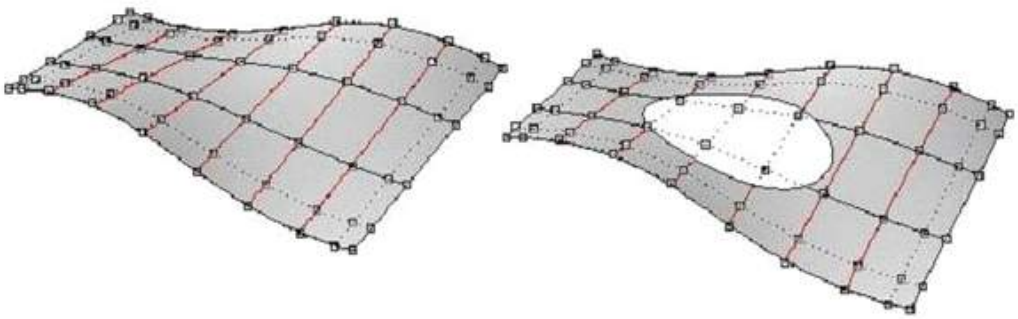
	범위 = 3 컨트롤 포인트 수 = 7 매듭 Vector = {0,0,0,1,2,3,5,5,5}
	범위 = 3 컨트롤 포인트 수 = 7 매듭 Vector = {0,0,0,1,1,1,4,4,4} Note: 중간에 전체 매듭 중복성은 꼬임을 일으키고 Curve는 컨트롤 포인트로 꺾이게 됩니다.

- 평가 기준
값을 구하는 법은 수학 공실을 사용하게 되는데 수와 포인트를 지정합니다. 공식은 범위, 컨트롤 포인트, 그리고 매듭을 포함합니다.
이 공식을 사용해서 Curve 매개변수를 갖는 SDK 함수를 Curve에 상응하는 점을 산출합니다. 매개변수는 Curve 영역 안에 있는 수입니다. Curve 영역은 보통 증가하며 두 가지 수로 구성되어 있습니다.: 최소영역 매개변수(m_t(0))는 보통 Curve와 같이 시작하고 최대영역 매개변수(m_t(1))는 Curve의 끝에 존재합니다.

- Nurbs Surface
Nurbs Surface는 Nurbs Curve의 그리드와 같이 두 가지 방향으로 흐른다고 생각할 수 있습니다. Nurbs Surface의 모양은 컨트롤 포인트 수와 Surface의 두 방향의 각에 의해서 결정됩니다. Rhino 도움말에 더 자세한 내용이 나와 있습니다.



Nurbs Surface는 잘라지거나 그렇지 않을 수 있습니다. 잘라진 Surface를 사용해 Nurbs Surface 밑에 두고 특정한 모양으로 잘라내었다고 생각해봅니다. 각 Surface는 하나의 막힌 Curve를 가지고 외부의 경계로 정의 되고 교차되지 않은 안쪽 Curve는 구멍으로 정의됩니다. 외부 고리에 접한 Surface는 밑에 있던 Nurbs Surface와 같고, 앞에서 언급했듯 잘라지지 않은 Surface로 구멍을 갖지 않습니다.

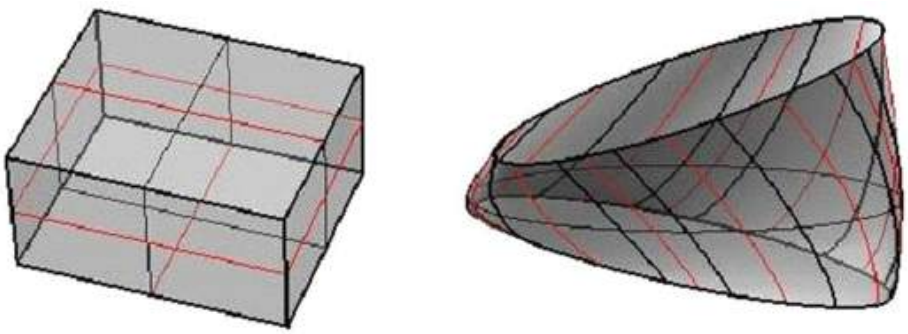


왼쪽의 Surface는 잘리지 않았습니다. 오른쪽의 Surface는 타원형의 구멍이 잘렸습니다. Surface의 Nurbs 구조는 잘려도 변하지 않습니다.

- 폴리Surface

폴리Surface는 하나 이상의 Surface들의 조합으로 이루어집니다. 각각의 Surface는 매개변수를 가지고 있고 u, v 방향은 동일하지 않습니다. 폴리Surface와 잘려진 Surface는 경계의 표시를 사용해 표시됩니다. 이것은 기본적으로 Surface로, 모서리들과 꼭지점들, 지오메트리, 잘려진 데이터와 각기 다른 부분의 관계들을 표현합니다.
예를 들어 이것은 각 면과 이것을 감싸는 모서리와 잘려진 부분, Surface와 관계된 정상적인 방향, 접하는 다른 면들과의 관계 등을 표현합니다.

OnBrep은 OpenNurbs에서 가장 복잡한 데이터 구조일 것이며, 이해하기 어려울지 모르겠지만 많은 툴과 Rhino SDK의 전역 함수가 Breps를 만들거나 도와줍니다.



15.3 열린 Nurbs 객체들의 체계

도움말 파일은 등급 분류를 보여줍니다. 여기에 지오메트리 구성과 조정에 관련된 나눠진 분류들로 앞으로 스크립트를 쓸 때 매우 유용하게 쓰입니다. 이것들은 매우 불완전한 상태이며, SDK에 대한 도움말 파일을 알아보도록 권장합니다.

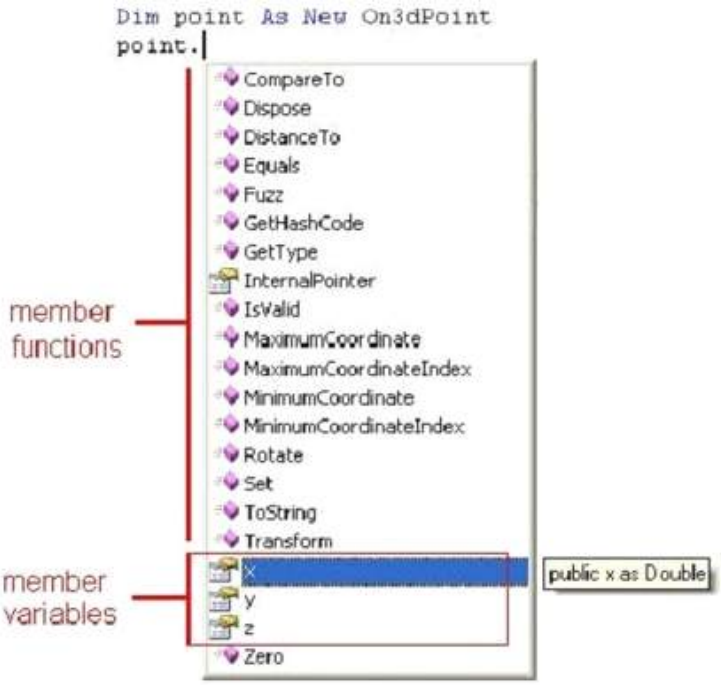
OnObject(all Rhino classes are derived from or inherits OnObject)
- OnGeometry(class is derived or inherits OnObject)
o OnPoint
 ·OnBrepVertex
 ·OnAnnotationTxtDot
o OnPointGrid
o OnPointCloud
o OnCurve(abstaract class)
 ·OnLineCurve
 ·OnPolylineCurve
 ·OnArcCurve
 ·OnNurbsCurve
 ·OnCurveOnSurface
 ·OnCurveProxy
  ·OnBrepTrim
  ·OnBrepEdge
o OnSuface(abstract classt)
 ·OnPlaneSurface
 ·OnRevSurface
 ·OnSumSurface
 ·OnNurbsSurface
 ·OnProxySurface
  ·OnBrepFace
  ·OnOffsetSurface
o OnBrep
o OnMesh
o OnAnnotation
- Point and Vectors(not derived from OnGeometry)
o On2dPoint(good for parameter space Point)
o On3dPoint
o On4dPoint(good for representing control Point with x, y, z and w for weight)
o On3dVector
- Curves(not derived from OnGeometry)
o OnLine
o OnPolyline
o Oncircle

o OnArc
o OnEllipse
o OnBezierCurve
- Surface(not derived from OnGeometry)
o OnPlane
o OnSphere
o OnCylinder
o OnCone
o OnBox
o OnBezierCurve
- Miscellaneous
o OnBoundingBox(For objects bounding box calculation)
o OnInterval(Used for Curve and Surface domains)
o OnXform(for transforming geometry object: move, rotate, scale, etc.)
o OnMassProperties(to calculate volume, area, centroid, etc)

15.4 계층 구조

전형적인 계층(사용자가 한정한 데이터 구조)은 네 개의 주요 부분이 있습니다.
·Construction: 계층이 생성되는 경우에 사용됩니다.
·Public member variables: 열린 Nurbs 변수는 "m_"으로 시작되어야 빠르게 분리합니다.
·Public member function: 이것은 계층 멤버 변수를 생성하고, 업데이트하며 매개변수나 확실한 상관관계의 이행을 위한 모든 계층 함수에 포함됩니다.
·Private members: 내부에서 사용하기 위해 필요한 계층 효용 함수와 변수입니다.

한 번 계층 군에 만족시키지 못하면, 자동으로 완성된 현상을 모든 계층군 멤버의 함수와 멤버 변수를 볼 수 있습니다. 어떠한 함수나 변수를 만족하지 못할 경우에는 그 함수의 기호들이 보여 집니다. 함수의 매개변수들을 채워 넣기 시작하면, 자동 완성 기능은 사용할 수 있는 매개변수와 그것의 타입을 보여줍니다. 이것은 각각의 계층 군에서 이용 가능한 함수를 탐색할 수 있는 가장 좋은 방법입니다. 여기 "ON3D점"라는 계층군의 예가 있습니다.



이미 존재하는 계층 군에서 새로운 계층 군으로 데이터를 복사한다는 것은 계층 구조에 따라 다르긴 하지만, 한 가지 혹은 여러 가지 방법에 의해서 행해질 수 있습니다. 예를 들면, 여기 새로운 "ON3D점"라는 계층 군을 만들고 이미 존재하는 "점"의 내용을 새로 만든 계층 군에 복사합니다. 아래의 내용이 어떻게 되어 지는가를 보여줍니다.

```
Use the constructor when you instance of the Point class
Dim new_pt as New On3dPoint( input_pt )

Use the "= operator" if the class provided one
Dim new_pt as New On3dPoint
new_pt =( input_pt )

You can use the "New" function if available
Dim new_pt as New On3dPoint
new_pt. New( input_pt )

There is also a "Set" function sometimes
Dim new_pt as New On3dPoint
new_pt. Set( input_pt )

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
```

Open Nurbs geometry classes provide "Duplicate" function that is very efficient to use
Dim new_crv as New OnNurbsCurve
new_crv = input_crv.DuplicateCurve()

15.5 상수와 비상수의 경우
Rhino.NET SDK는 두 가지 세트의 계층 구조를 제공합니다.
첫 번째는 "상수"인데 이것은 "I"라는 이름으로 시작하는 계층 구조입니다. 예를 들면 "ION3D점"입니다. 이에 대응하는 "비 상수" 계층군은 대부분 "I"로 시작되는 것 없이 같은 이름으로 쓰이는 것들입니다. 예를 들면 "ON3D점"입니다.
상수 계층 군을 복사 한다거나, 그것들의 멤버 상수들 혹은 몇몇 함수들을 볼 수는 있지만, 그것들의 변수를 바꿀 수는 없습니다.

15.6 점과 Vector들

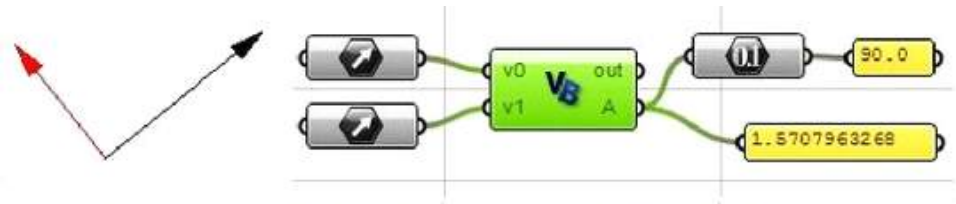
점과 Vector들을 저장하고 조작하는데 쓰일 수 있는 많은 계층 군이 있습니다. 두 개의 정확한 점들을 예로 들어 봅니다. 여기 세 가지 종류의 점들이 있습니다.

Class 이름	Member variables	Notes
On2dPoint	x as Double y as Double	공간상의 매개변수의 점을 위해 주로 사용합니다. 정확한 두 개의 숫자 점을 위해 계층 이름을 "d"로 합니다. 다른 계층의 점들은 정확한 단수를 사용하기 위해 "f"라고 명명합니다.
On3dPoint	x as Double y as Double	3차원 점을 나타내기 위해 대부분 일반적으로 사용됩니다.
On4dPoint	x as Double y as Double z as Double w as Double	세 개의 정보를 갖고 있는 점들을 잡기 위해 사용합니다.

점과 Vector들의 작용은 아래의 내용을 포함합니다.:
- Vector Addition:
Dim add_v As New On3dVector = v0 + v1
- Vector Subtraction:
Dim subtract_Vector As New On3dVector = v0 - v1
- Vector between two Point:
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 입력 Vector):
Dim normal_v As New On3dVector = OnUtil.ON_CrossProduct(v0, v1)
- Scale a Vector:
Dim scaled_v As New On3dVector = factor * v0

- Move a Pint by a Vector:
Dim moved_Point As New On3dPoint = org_Point + dir_Vector
- Distance between 2Point:
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()

아래의 예들은 두 Vector 간의 각도를 계산하는 방법을 보여줍니다.

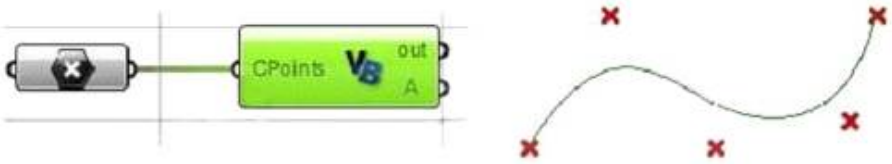



```
Sub RunScript(ByVal v0 As On3dVector, ByVal v1 As On3dVector)

  ' 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

다음을 설정하여 NurbsCurve를 생성합니다.
- 치수 : 보통 "3"으로 설정되어 있습니다.
- 순서 : Curve의 차수(1차, 2차, 3차 등등)+1
- 컨트롤 포인트
- 매듭 Vector(숫자들의 배열)
- Curve 유형(clamped 혹은 periodic)

간략하게 볼 수 있듯, 기본적으로 치수의 결정과 컨트롤 포인트들의 리스트를 갖고 있는 매듭 Vector를 만드는데 도움이 되는 함수들이 있습니다.




```
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
```

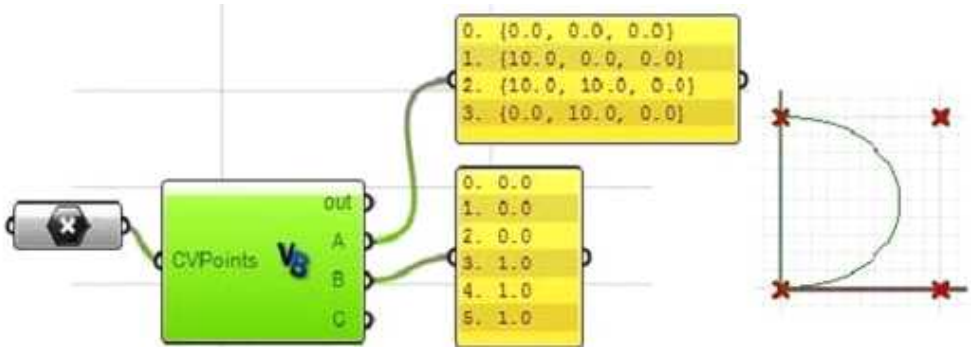
부드럽게 닫힌곡선을 만들기 위해서 PERIODIC Curve를 만들어야 합니다. 입력된 같은 컨트롤 포인트와 Curve의 차수를 사용하기 위해서 아래의 보기가 어떻게 PERIODIC Curve를 만드는지 보여줍니다.


```
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
```

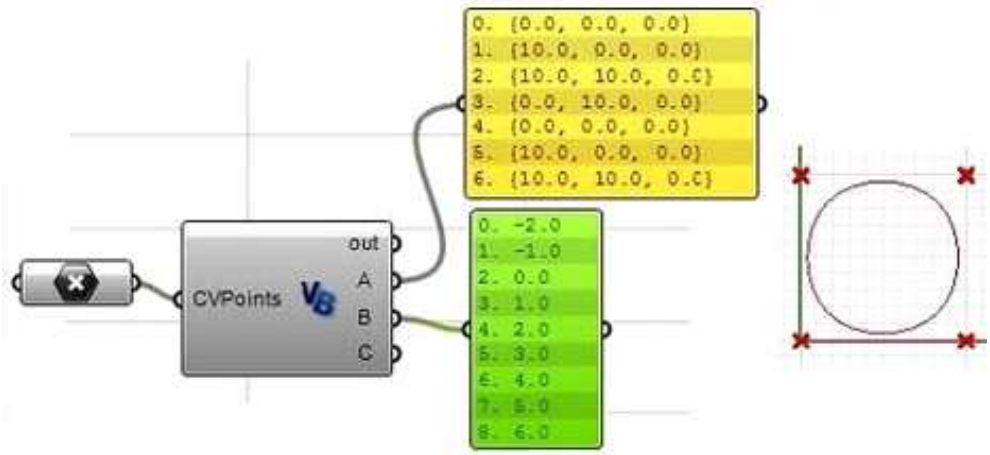
- 고정(강제) NurbsCurve와 주기적 NurbsCurve

고정된 곡선은 Curve가 끝나는 점과 컨트롤 포인트가 끝나는 점이 같은, 대부분이 열린 Curve입니다. PERIODIC Curve는 부드러운 닫힌 Curve입니다. 컨트롤 포인트들을 비교해 보

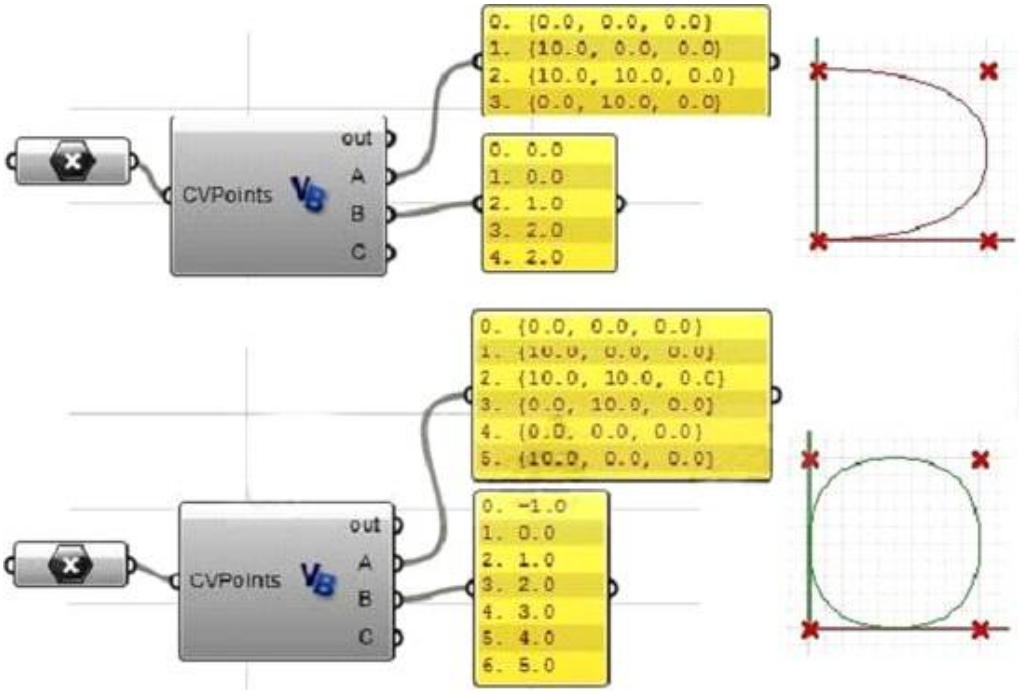
는 것이 CLAMPED Curve와 PERIODIC Curve의 차이를 이해할 수 있는 가장 좋은 방법입니다. 아래의 구성 요소들은 CLAMPED Nurbs Curve와 결과물을 만들어냅니다.



아래의 컴포넌트들은 CLAMPED Nurbs Curve와 결과물을 만들어냅니다.



CLAMPED Curve는 단지 4개의 컨트롤 포인트만을 (4+차수) 사용하는 반면에 PERIODIC Curve는 4개의 입력 점에서 7개의 컨트롤 포인트로 바꿨다는 것을 기억합니다. CLAMPED Curve는 완전히 다양한 매듭들로 시작과 끝을 사용한 반면에 PERIODIC Curve의 매듭 Vector들은 간단한 매듭만을 사용하였습니다.
아래는 2차원 Curve들로 된 같은 예입니다. 컨트롤 포인트와 PERIODIC 곡선의 매듭 개수는 "차수"가 변함에 따라 변하게 됩니다.

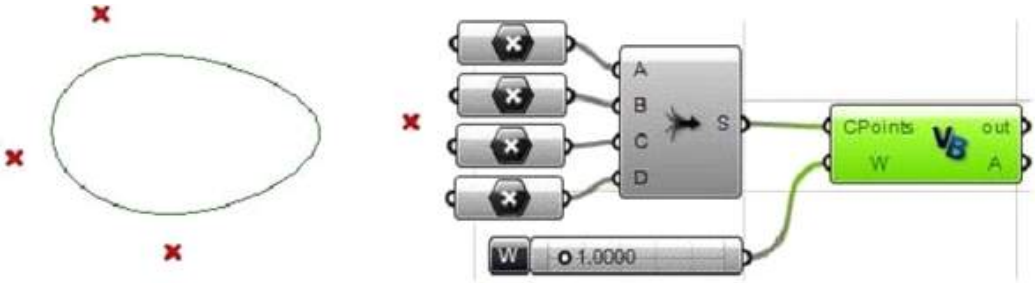


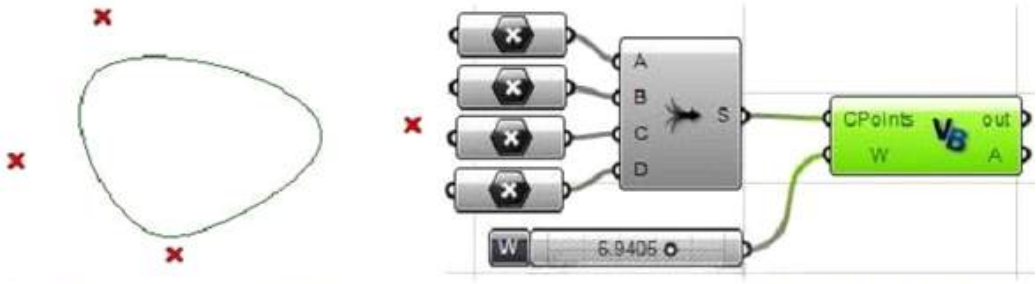
이것이 바로 앞의 예에서 보신 바와 같은 "CV Point"와 "매듭"을 통해서 본 조합입니다.

```
'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
```

- 무게

일정한 Nurbs 곡선의 컨트롤 포인트의 무게는 1로 지정되어 있습니다. 하지만 이 숫자들은 논리적인 Nurbs Curve에서 다양할 수 있습니다. 아래의 예는 Grasshopper에서 컨트롤 포인트의 무게를 어떻게 수정하는지를 보여줍니다.





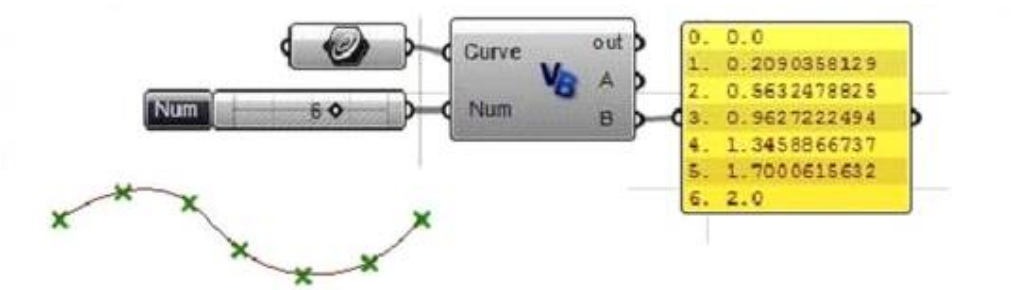

```
Sub RunScript(ByVal CPoints As List(Of On3dPoint), ByVal W As Double)

  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
```

- Nurbs Curve 나누기

아래의 순서를 통해서 하나의 Curve를 몇 개의 부분으로 나눌 수 있습니다.

- 간격을 구분하는 컴포넌트인 Curve의 도메인을 찾습니다.
- 하나의 Curve를 같은 부분으로 나누는 컴포넌트의 리스트를 만듭니다.
- 3D Curve에서 그 점들을 찾습니다.

아래의 예에서는 Curve 나누는 방법을 보여줍니다. RhUtil이라는 이름의 공간 아래 Curve의 몇 개 세그먼트나 호의 길이로 곡선을 나누는 글로벌 기능이 있습니다.




```
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 OnCurve에서 파생된 Curve 계층

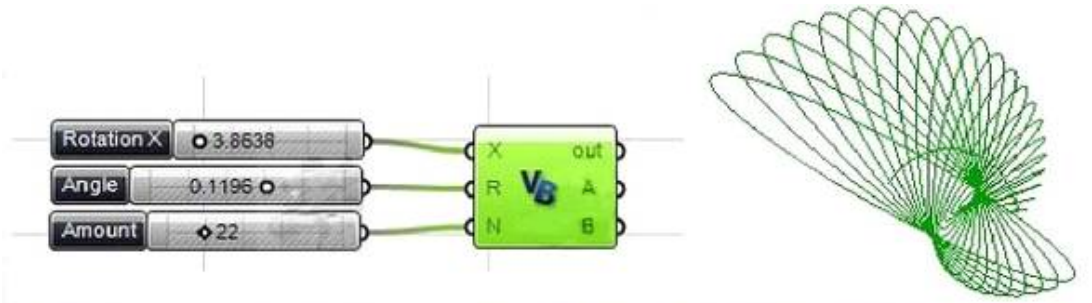
모든 Curve를 통하여 Nurbs Curve를 나타낼 수 있습니다. 이는 때때로 기하학 Curve의 다른 유형에 사용됩니다. 수학적인 표현이라는 하나의 이유는 Nurbs를 이해하기 쉽게 하며, 대체적으로 좀 더 가벼워집니다.

상대적으로 하나의 Nurbs를 필요로 하는 경우, OnCurve에서 유래되지 않는 Nurbs 형태를 만들기 쉽습니다. 사용자는 기본적으로 일치하는 Class로 전환되길 원합니다.

아래 테이블을 따라 상응함을 확인합니다.

Curve 유형s	OnCurve Derived 유형s
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)

아래에서는 ONELLIPSE와 OnPolyline의 계층 군을 이용하는 예를 보여줍니다.




```
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
```

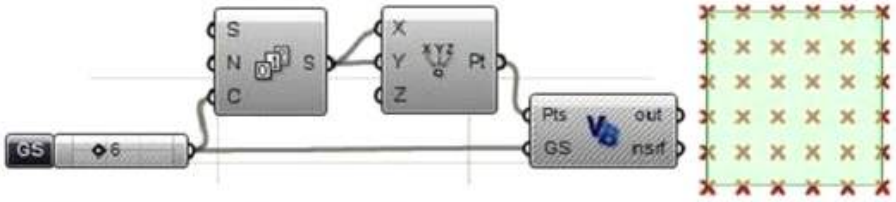
15.9 OnNurbsSurface

OnNurbsCurve 계층 구조와 유사하게 OnNurbsSurface를 만들기 위해서 아래의 내용들을 알고 있어야 합니다.

- 치수, 보통은 "3"으로 설정되어 있습니다.

- u 방향과 v 방향의 순서: Curve의 차수(1차, 2차, 3차 등등)+1
- 컨트롤 포인트(점들의 배열)
- u와 v 방향의 매듭 Vector
- Surface의 종류 (Clamped Surface 또는 PERIODIC Surface)

아래의 예는 컨트롤 포인트의 그리드에서 Nurbs Surface를 만드는 것을 보여줍니다.



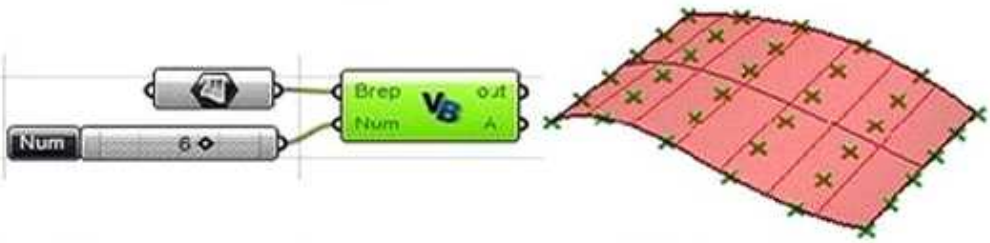

```
Sub RunScript(ByVal Pts As List(Of On3dPoint), ByVal GS As Integer)
  'Create a grid of points
  Dim Grid As New ArrayList()
  'Call grid function
  Call CreateGrid(Pts, Grid, GS)
  'Call create nurbs surface function
  Dim ns As OnNurbsSurface
  ns = CreateNS(Grid, GS)

  '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
```

다른 일반적인 예는 Surface 도메인을 분할하는 것입니다. 아래의 예는 Surface 도메인을 양 방향에서(포인트의 수는 당연히 1보다 커야만 합니다.) 같은 숫자의 점들로 분할하는 것을 보여줍니다.

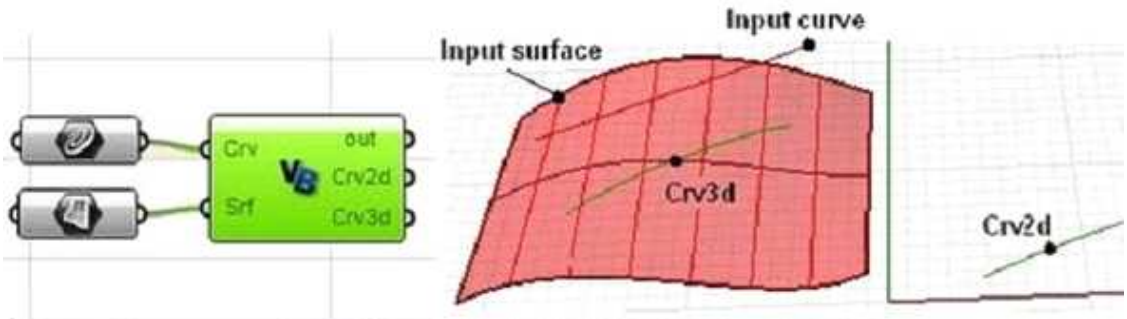
- Surface 도메인을 통일합니다.(즉, 도메인의 간격을 0에서 1로 합니다.)
- 점들의 수를 이용해서 각 단계의 값들을 계산합니다.
- u 방향과 v 방향의 구성 요소들을 사용한 Surface 포인트들을 계산하기 위해서 정리된 루프를 사용합니다.




```
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 계층군은 Surface와 연계되고 조작하기에 굉장히 유용한 많은 함수들을 가지고 있습니다. 다음의 예는 Curve를 Surface로 만드는 방법을 보여줍니다.

Grasshopper 스크립팅 컴포넌트에는 2가지의 결과물이 있습니다. 첫 번째 Surface는 도메인과 연계된 Parameter space Curve입니다. (x, y 좌표계에서 3D Curve의 평평한 부분) 두 번째는 3D 공간에서의 곡선입니다. 2D Parameter space Curve를 Surface에 "밀어냄"으로 3D 곡선을 얻을 수 있습니다.



```
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
```

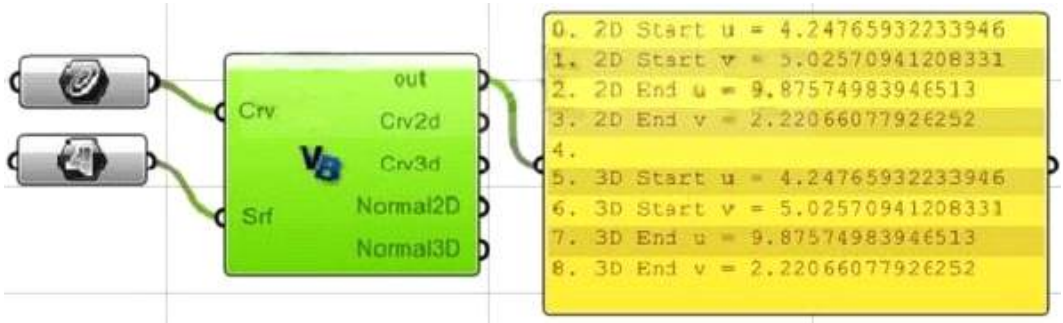
당겨진 Curve의 시작과 끝점의 Normal Vector를 앞의 예를 통해서 계산할 것입니다. 여기 2가지 방법이 있습니다.

- 2D 포인트로 시작되고 끝나는 "당겨진 2D Curve"를 사용하고, 또 이것은 Parameter space의 Surface 상에서 시작점과 끝점이 될 것입니다.
- 혹은 점들이 끝나는 "밀어내어진 3D Curve"를 이용합니다. 그리고 Surface에 가장 가까운 점들을 찾아서, 그 결과로 만들어진 Parameter를 Surface Normal을 찾기 위해 사용합니다.

```
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
```

이것은 두 함수를 사용한 끝점에서의 Parameter 값의 결과물을 보여주는 구성도입니다. 유념할 것은 두 가지의 방법이 예상대로 같은 Parameter들을 만들어냅니다.

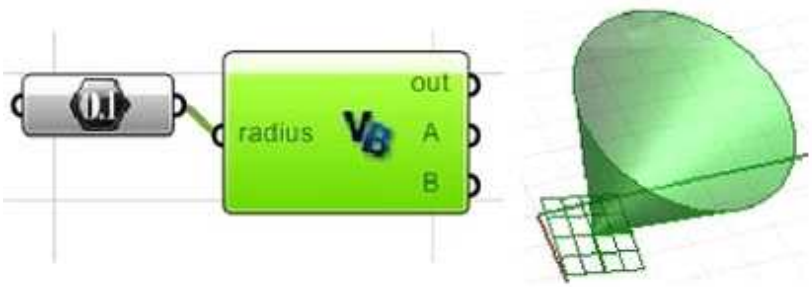


15.10 OnSurface로부터 파생되지 않은 Surface 계층 군

OpenNurbs는 OnSurface로부터 파생되지 않은 Surface 계층 군을 제공합니다. 그것들은 확실한 수학적인 Surface 정의들이고, 이것들은 ONSurface로 변환될 수 있습니다. 여기에 Surface 계층군의 리스트와 그것들의 상호적인 ONSurface로 파생된 계층 군이 있습니다.

기본 Surface 유형	OnSurface 유형 분류
OnPlane	OnPlaneSurface or OnNurbsSurface (use OnPlane.GetNurbsForm()function)
OnShpere	OnPlaneSurface or OnNurbsSurface (use OnShpere.GetNurbsForm()function)
OnCylinder	OnPlaneSurface or OnNurbsSurface (use OnCylinder.GetNurbsForm()function)
OnCone	OnPlaneSurface or OnNurbsSurface (use OnCone.GetNurbsForm()function)
OnBezierSurface	OnPlaneSurface or OnNurbsSurface (use GetNurbsForm()member function)

아래의 예는 OnPlane과 OnCone 계층 군을 이용한 것입니다.




```
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

영역표시(B-rep)는 영역 Surface에 대해서 명백하게 Object들을 나타내기 우해 사용됩니다. 여러분들은 OnBrep을 세 개의 명확한 부분으로 나누어 생각할 수 있습니다.

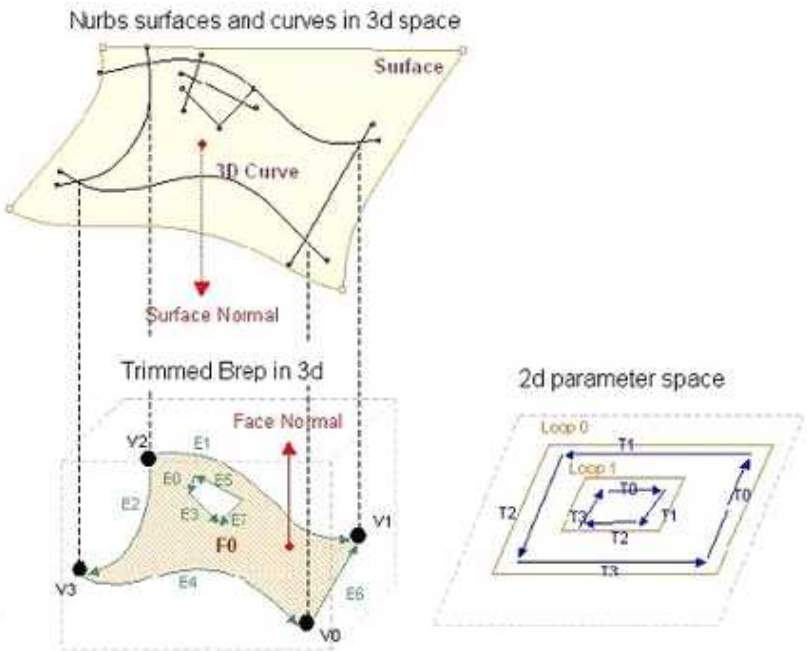
·Geometry: Surface와 Nurbs Curve의 3D Geometry. 또한 Parametric space의 2D Curve 혹은 다듬어진 Curve들입니다.

·3차 위상 기하학: 면, 모서리, 점. 각각의 면들은 하나의 Nurbs Surface를 말합니다. 또한 면은 그에 속한 모든 루프 정보에 대해 알 수 있습니다. 변은 3D Curve를 말합니다. 각각의 모서리는 변으로도 쓰이며, 두 점을 된 가장자리에 대한 List를 갖고 있습니다.

점들은 공간에서의 3D 점들을 말합니다. 또한 각각의 점들은 두 점을 잇는 변에 대한 List를 갖고 있습니다.

·2차 위상 기하학: 2D Parametric 공간은 면과 변을 말합니다. Parametric 공간에서는 2D trim Curve들은 어떠한 면의 내부 고리인지, 혹은 외부 고리인지에 따라 시계 방향과 반시계 방향으로 갈 수 있습니다. 각각의 유효한 면들은 반드시 하나의 외부 고리를 가져야 합니다. (하지만, 이는 많은 수의 요구되어지는 만큼의 내부 고리를 가질 수도 있습니다.) 각각의 다듬어진 가장자리는 하나의 변, 2개의 끝점, 하나의 Loop 그리고 2D Curve를 말합니다.

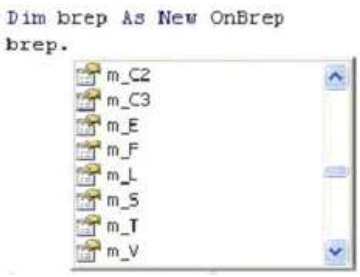
다음의 다이어그램은 이 세부분을 보여주며, 또 이것들이 각각 어떻게 관계가 되는지를 보여줍니다. 가장 상단 부분은 근본적인 3D Nurbs Surface와 구멍과 마주하여 단일 면의 영역 표시 Curve의 개체를 보여줍니다. 중간은 영역 표시의 면, 변의 외부, 변의 내부(구멍이 있는 부분)와 점들을 포함하는 3D 위상 기하학 부분입니다. 그리고 다듬어진 부분의 Loop로 된 Parameter 공간이 있습니다.



-OnBrep member variables

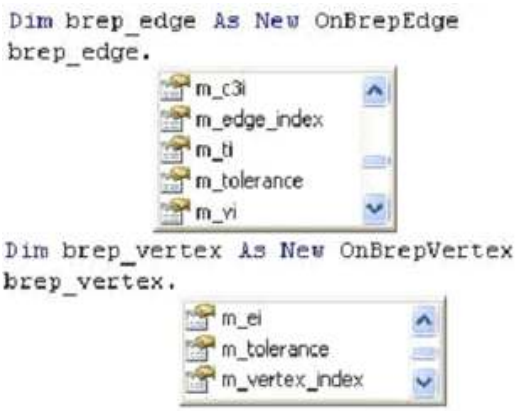
영역표시(B-REP) 계층군의 멤버 변수들은 모든 3D와 2D Geometry 와 Topology 정보를 포

함합니다. 영역표시 계층 군을 한번 만들고 나면, 모든 멤버의 함수 관계와 변수들을 볼 수 가 있습니다. 다음은 자동 완성된 멤버의 함수 관계를 보여줍니다.
아래의 표는 데이터의 종류와 그에 대한 설명의 리스트를 보여줍니다.



Topology members: describe relationships among different brep parts	
OnBrepVertexArray m_V	Array of brep vertices (OnBrepVertex)
OnBrepEdgeArray m_E	Array of brep deges (OnBrepEdge)
OnBrepFaceArray m_T	Array of brep trims (OnBrepTrim)
OnBrepLoopArray m_F	Array of brep faces (OnBrepFace)
OnBrepLoopArray m_L	Array of loop (OnBrepLoop)
Geometry member: geometry data 3d Curve and Surface and 2d trims	
OnCurveArray m_C2	Array of time Curve (2D Curve)
OnCurveArray m_C3	Array of edge Curve (3D Curve)
OnSurfaceArray m_S	Array of Surface

각각의 OnBrep(영역표시) 멤버 함수 관계는 기본적으로 다른 계층군의 배열이라는 것을 유념해야 합니다. 예를 들면 "m_F"는 "OnBrepFace"를 말합니다.
또한, "OnBrepFace"는 OnSurFaceProxy에서 파생되어 나온 계층 군이며, 이는 스스로의 변수와 멤버 함수를 갖습니다. 아래에는 "OnBrepFace, OnBrepEdge, OnBrepVertex, OnBrepTrim, OnBrepLoop" 계층군의 멤버 변수들입니다.

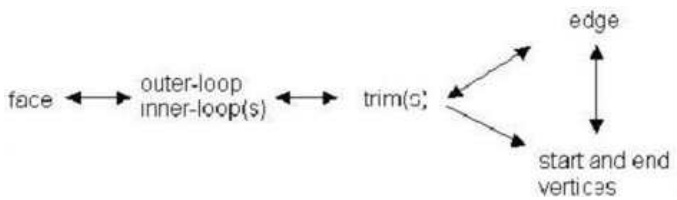


```
Dim brep_trim As New OnBrepTrim
brep_trim.
```

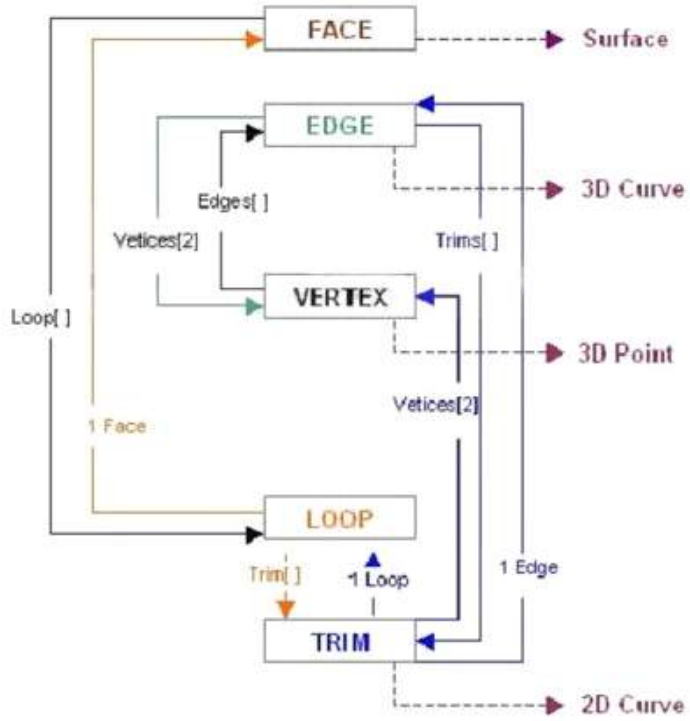

```
Dim brep_loop As New OnBrepLoop
brep_loop.
```


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

아래의 다이어그램은 "OnBrep" 멤버 변수들과 서로가 어떻게 관계되는지를 보여줍니다. 이러한 "Brep"의 어떠한 특정한 부분의 정보를 이용할 수 있습니다. 예를 들면, 각각의 면은 그것들의 루프 리스트가 갖고 있고, 그 다음어진 부분으로부터 2개의 끝점들로 연결된 변의 정보를 얻을 수 있습니다.

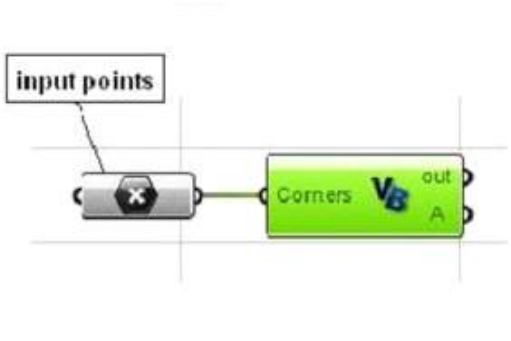


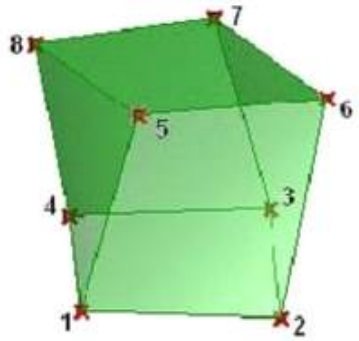
여기 또 다른 자세한 다이어그램은 어떻게 "Brep"부분들이 함께 연결 되는지 또 한 부분으로부터 다른 부분을 어떻게 얻을 수 있는지를 보여줍니다.



몇 가지 예를 통해서 "OnBrep"을 어떻게 만들어지는지를 볼 수 있습니다. 서로 다른 부분들을 살펴보고 "Brep" 정보를 추출해봅니다. 또한 우리는 글로벌 함수뿐 만 아니라, 계층 구조와 함께 오는 몇몇의 함수 관계들이 어떻게 사용되는지를 볼 수 있습니다.

- OnBrep 생성하기
 여기 새로운 "OnBrep" 계층 구조를 만드는 몇 가지 방법들이 있습니다.
- 기존의 Brep을 복사합니다.
- 기존의 Brep에서 면을 복사하거나 혹은 추출합니다.
- 입력 매개변수로 OnSurface를 이용하는 "CREATE함수"를 사용합니다.
o from SumSurface
o from RevSurface
o from PlanarSurface
o from OnSurface
- 글로벌 유틸리티 함수를 사용합니다.
o On_BrepBox나 On_BrepCone등의 OnUtil을 사용합니다.
o RhinoCreatEdgeeurface나 RhinoSweep1 등등의 RhUtil을 사용합니다.
아래의 예는 모서리의 점들을 사용하여 Brep Box를 만드는 방법입니다.






```
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
```

- 데이터 진행하기

다음의 예제는 Brep 박스의 여러 가지 점들을 추출하는 방법입니다.

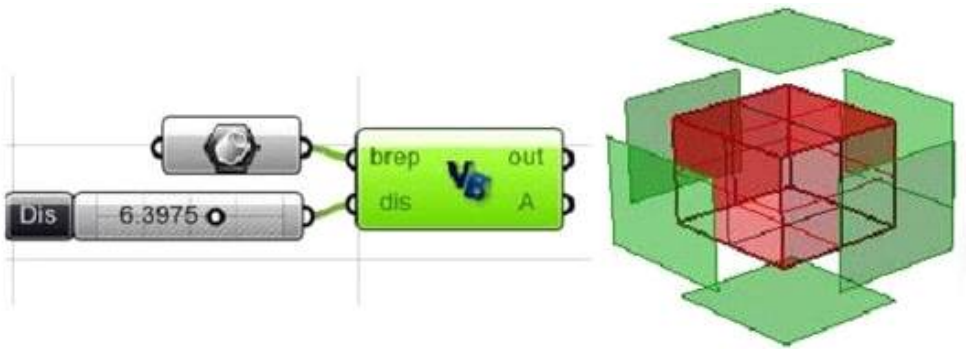
```
Sub RunScript(ByVal Corners As List(Of On3dPoint))

  ' 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
```

아래는 "Brep Box(면, 모서리, 절단, 꼭지점)"에서 어떻게 몇 개의 Geometry와 Topology 부분들을 얻어내는지 보여줍니다.

- OnBrep들의 변형

OnGeometry에서 파생된 모든 계층 구조들은 4개의 변형 함수들을 상속합니다.
첫 번째 세 개의 함수는 Rotate, Scale, Transform 가장 많이 쓰이는 것들입니다. 하지만 여기에는 OnXform 계층 구조와 함께 정의된 특유의 4×4 형식의 변형 행렬도 있습니다. 다음 섹션에서 OnXform에 대해서 다뤄보도록 합니다.

- OnBrep 편집하기
대부분의 OnBrep계층 멤버 함수들은 Brep들을 생성하고 편집하는 전문적 사용자 도구입니다. 하지만 다른 섹션에서 다루게 될, Brep들을 Boolean, Intersect, Split하는 많은 글로벌 함수들도 있습니다.
McNeel의 Wiki DotNet 샘플에는 처음 작업부터 OnBrep을 만들기까지 도움이 되는 좋은 예입니다.
여기 Brep부터 OnBrep 면들을 추출하고 또 그것들을 이동하는 예가 있는데, 이것은 BoundingBox를 이용합니다.

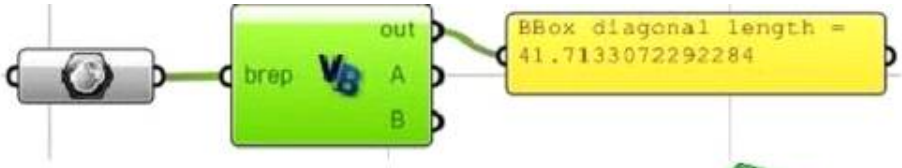


```
Sub RunScript(ByVal brep As OnBrep, ByVal dis As Double)

  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 center 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
```

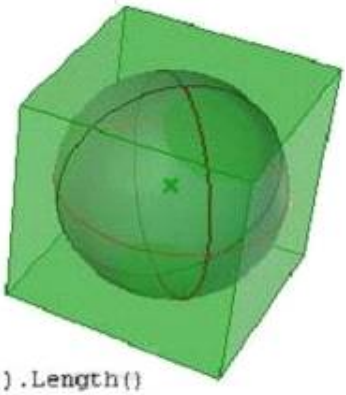
- 다른 OnBrep 멤버 함수

OnBrep 계층 구조는 부모 계층 구조 혹은 구체적으로 OnBrep 계층 구조에서 쓰이는 다른 많은 함수들을 가지고 있습니다. OnBrep을 포함하는 모든 Geometry 구조들은 "Bounding Box"라 불리는 멤버 함수를 가지고 있습니다. OpenNurbs 계층 구조 중의 하나는 OnBoundingBox인데, 이것은 유용한 Geometry 영역 정보를 줍니다.

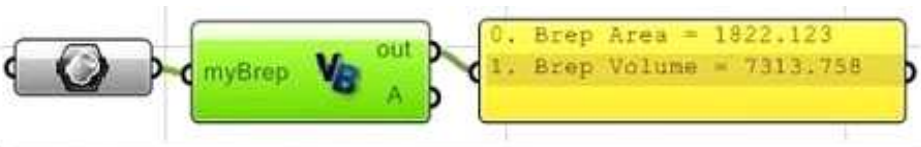
아래의 예는 그것의 중심과 대각선의 길이를 가진 Brep Bounding BOX를 알 수 있습니다.



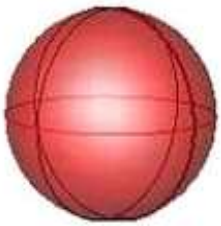

```
Sub RunScript(ByVal brep As OnBrep)

  '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
```



Mass Properties(매스 속성)이라는 영역이 있는데, OnMassProperties라는 계층 구조와 이에 대한 함수는 아래의 예에서 알 수 있습니다.

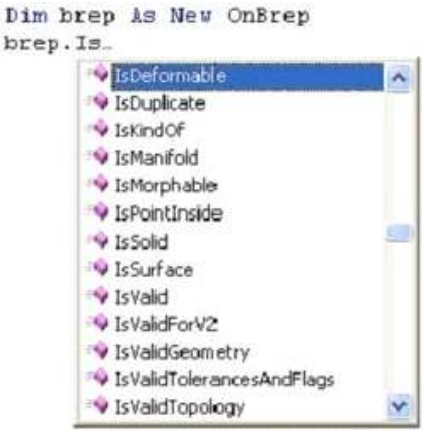



```
Sub RunScript(ByVal myBrep As Object)

  '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
```

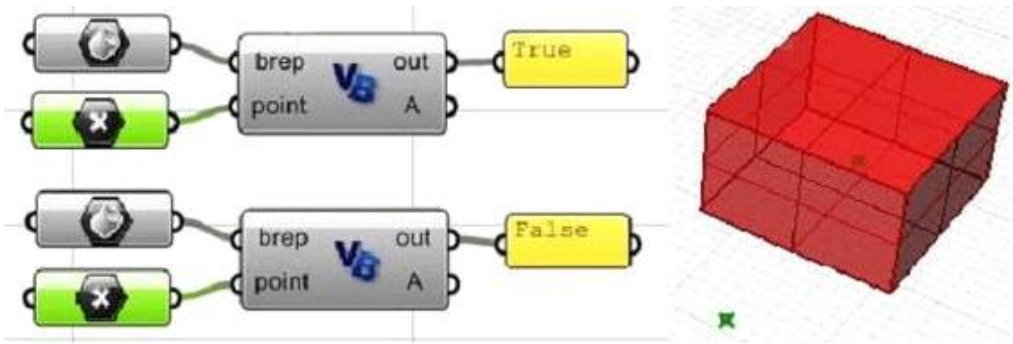


"IS"로 시작하는 몇 개의 함수들이 있는데 이것들은 대부분 BOOLEAN을 재조정합니다. 이것들은("IS"로 시작하는 함수들) 지금 수행하고 있는 "Brep"에 대해서 물어 오는데, 예를 들면 Brep이 닫힌 PolySurface인지 아닌지를 알고 싶다면, "OnBrep.IsSolid()"라는 함수를 사용할 수 있습니다. 이것은 또한 "Brep"이 유용한지, 혹은 유용한 Geometry를 가지고 있는지를 확

인할 수 있습니다.
여기 "OnBrep" 계층 구조에서 물어보는 함수의 리스트가 있습니다.



아래의 예에서는 주어진 점이 "Brep"의 내부에 있는 점인지 확인합니다.



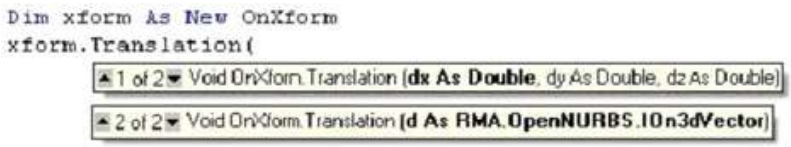
아래의 예에서는 주어진 점이 "Brep"의 내부에 있는 점인지 확인합니다.

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

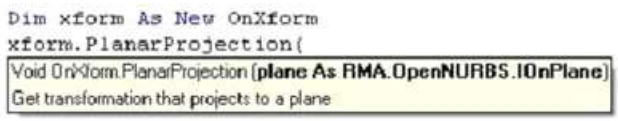
15.12 기하학적 구조의 변환

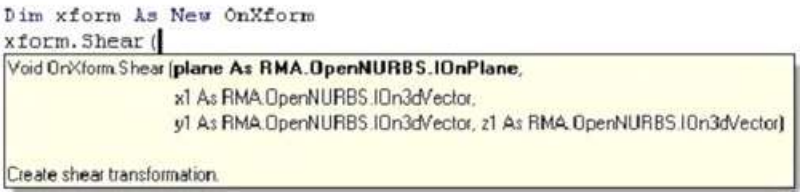
OnXform은 변환 매트릭스를 저장하고 조작하기 위한 클래스입니다. 이것은 객체의 이동, 회전, 확대, 축소 및 절단을 위해 매트릭스를 정의하는 것을 포함하지만, 국한되지는 않습니다. OnXform의 m_xform은 Double precision numbers(64비트형 소수 표시)의 4×4 매트릭스입니다. 또한 클래스는 매트릭스 연산을 지원하는 기능도 갖고 있습니다.
아래의 것은 상이한 변환을 야기하는 것과 연관된 몇몇의 멤버 함수입니다.

한 가지 좋은 자동 완성의 특징(모든 함수에 적용될 수 있는)은 일단 하나의 함수가 선택되면, 자동 완성 기능이 모든 연산 기능을 보여주는 것입니다. 예를 들면, 변형은 그림에 보여지는 것처럼 Tree numbers 또는 Vector를 수용합니다.

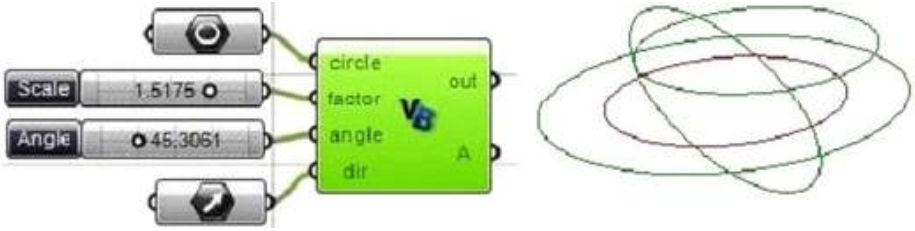


여기 OnXform 기능 몇 가지가 더 있습니다.





아래의 예시는 한 개의 원을 입력하여 3개의 원을 산출하는 것입니다. 처음의 원은 원래의 원이 축적에 의해 산출된 원이고, 둘째 원은 회전된 원이며, 셋째 원은 변형된 원입니다.



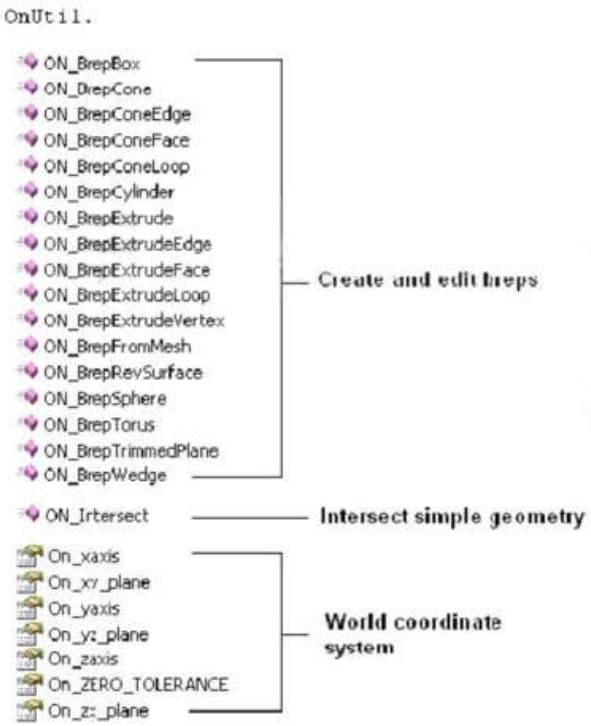

```
Sub RunScript(ByVal circle As OnCircle,
              ByVal factor As Double,
              ByVal angle As Double, ByVal dir As On3dVector)

  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 글로벌 유틸리티 함수

각 클래스 내부에 있는 멤버 함수와는 별개로 Rhino.NET SDK는 OnUtil and RhUtil 이름 공간 하에서 글로벌 함수를 제공합니다. 이러한 기능 몇 가지를 이용한 예를 들어봅니다.

- OnUtil

다음은 기하학과 연관이 있는 OnUtil 하에서 이용할 수 있는 기능들의 요약입니다.



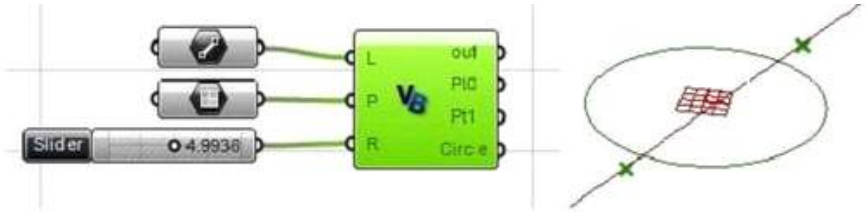
- OnUtil 교차점

On_Intersect 유틸리티 기능은 11가지의 연산 기능을 가지고 있습니다. 아래는 교집합 구조와 Return 결과 값s의 리스트입니다.

("IOnLine"에서와 같이 앞에 있는 "I"는 지속되는 상황이 패스되었다는 것을 의미합니다.)

Intersected	**출력**
IOnLine with IOnArc	Line parameter (t0 & t1) and Arc 점(p0 & p1)
IOnLine with IOnCircle	Line parameter (t0 & t1) and Circle 점(p0 & p1)
IOnSpere with IOnShere	OnCircle
IOnBoundingBoc with IOnLine	Line parameter (OnInterval)
IOnLine with IOnCylinder	2Point (On3dPoint)
IOnLine with IOnSphere	2Point (On3dPoint)
IOnPlane with IOnSpere	OnCircle
IOnPlane with IOnPlane with IOnPlane	On3dPoint
IOnPlane with IOnPlane	OnLine
IOnLine with IOnPlane	Parameter t (Double)
IOnLine with IOnLine	Parameter a & b (on first and second line as Double)

아래는 선과 평면 그리고 구형의 교집합의 결과를 보여주는 예입니다.




```
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
```

- Rhino 유틸(RhUtil)

Rhino 유틸리티는 기하학과 관련된 함수를 아주 많이 가지고 있습니다. 리스트는 사용자 요청에 따른 새 버전 출시와 함께 확장됩니다.

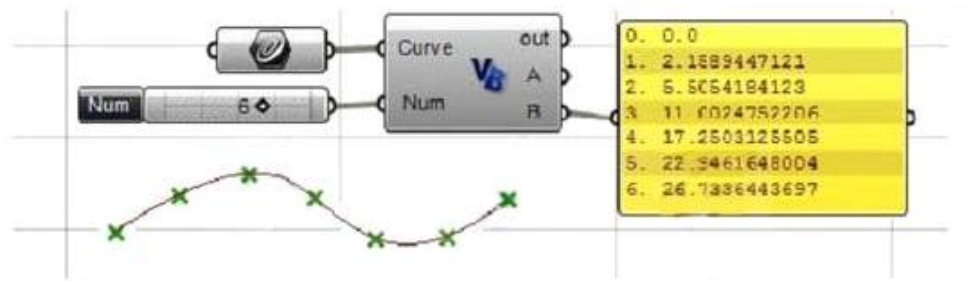
아래에 보이는 것은 기하학과 관련된 함수들입니다.

- RhUtil Curve 자르기

RhUtil.RhinoDivideCurve 유틸리티 기능을 사용하여 수로 나뉘거나 길이로 나누어지도록 Curve를 자릅니다. 이것은 손상된 기능의 변수:

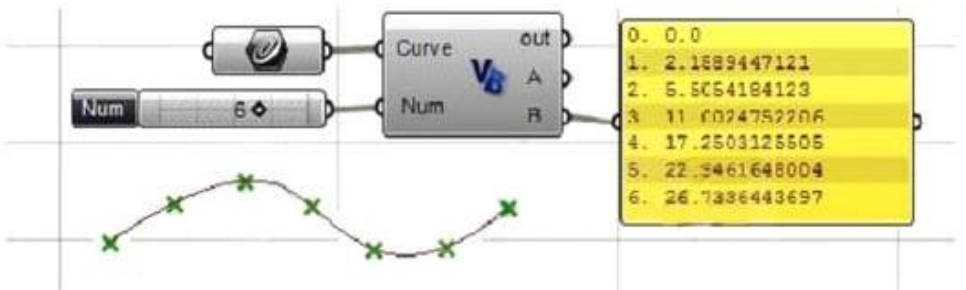
Rhino Curve 자르기: 기능 이름
Curve: 상수로 Curve 나누기
Num: 분할된 조각의 수

Len: 나누어진 Curve의 길이
False: 파기된 Curve의 표시(참이나 거짓으로 값을 결정할 수 있습니다.)
True: 끝점을 포함(참이나 거짓으로 값을 결정할 수 있습니다.)
crv_t: Curve 상의 나누어진 포인트의 매개변수 목록을 끕니다.
정수 분할로 나뉜 Curve의 예:




```
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
```

호의 길이로 나누어진 Curve의 예:




```
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
```

- 점을 통과하는 RhUtil Curve(보간Curve)
RhinoInterpCurve: 기능 이름
3: Curve 차수
pt_array: Curve를 통과한 점들
Nothing: 접점의 시작
Nothing: 접점의 끝
0: 균일 매듭점
예제를 따라 On3d점의 목록을 입력하여 이 3개의 점의 포인트를 통과하는 하나의 Nurbs Curve를 산출하는 예입니다.

```
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 Surface의 모서리 생성하기
예제를 따라 입력된 내용은 4개의 Curve 리스트이고, 출력되는 것은 모서리 Surface입니다.


```
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 도움말

- 더 많은 정보 제공

어디에서 Rhino DotNet SDK에 대한 더 많은 정보와 샘플은 McNeel Wiki에 있습니다. 개발자들의 활발한 활동으로 새로운 재질과 예제를 이곳에 항상 추가하고 있습니다.
http://en.wiki.mcneel.com/default.aspx/McNeel/Rhino4DDotNetPlugins.html 훌륭한 리소스를 여기서 찾을 수 있습니다.

- 포럼과 토론

http://grasshopper.Rhino3d.com/ 포럼과 토론 그룹의 커뮤니티는 이로 인해 매우 활동적이고 활동에 뒷받침이 될 것입니다. Rhino Grasshopper 토론 포럼은 시작하기 좋은 장소가 될 것입니다.
http://www.Rhino3d.com/developer.htm
궁금한 사항은 Rhino 개발자 뉴스 그룹에 게시할 수 있습니다. 또한 개발자 McNeel의 웹페이지 http://www.Rhino3d.com/developer.htm 으로부터 Rhino 개발자 뉴스 그룹에 접속할 수 있습니다.

- Visual Studio로 프로그램 결함 수정하기

보다 복잡한 코드는 Grasshopper 스크립트 컴포넌트에서 결함을 수정하기는 어렵습니다. Microsoft가 공급하고 있는 스튜디오 풀 버전이나 3버전인 Visual Studio Express를 사용할 수 있습니다.
자세한 내용은 Express visual studio에서 제공 받을 수 있으며, 어떻게 사용하는지 알 수 있습니다.
http://en.wiki.mcneel.com/default.aspx/McNeel/DotNetExpressEditions
Visual Studio 풀버전에 접속한다면 더 좋을 것입니다. 아래 페이지를 따르면 설정 작업에 도움이 될 것입니다.
http://en.wiki.mcneel.com/default.aspx/McNeel/Rhino4DotNetPlugins.html

- Grasshopper Scripting 샘플

Grasshopper 갤러리와 토론 포럼에서 여러분은 분명이 유용한 많은 예제와 Script된 컴포넌트를 얻게 될 것입니다.