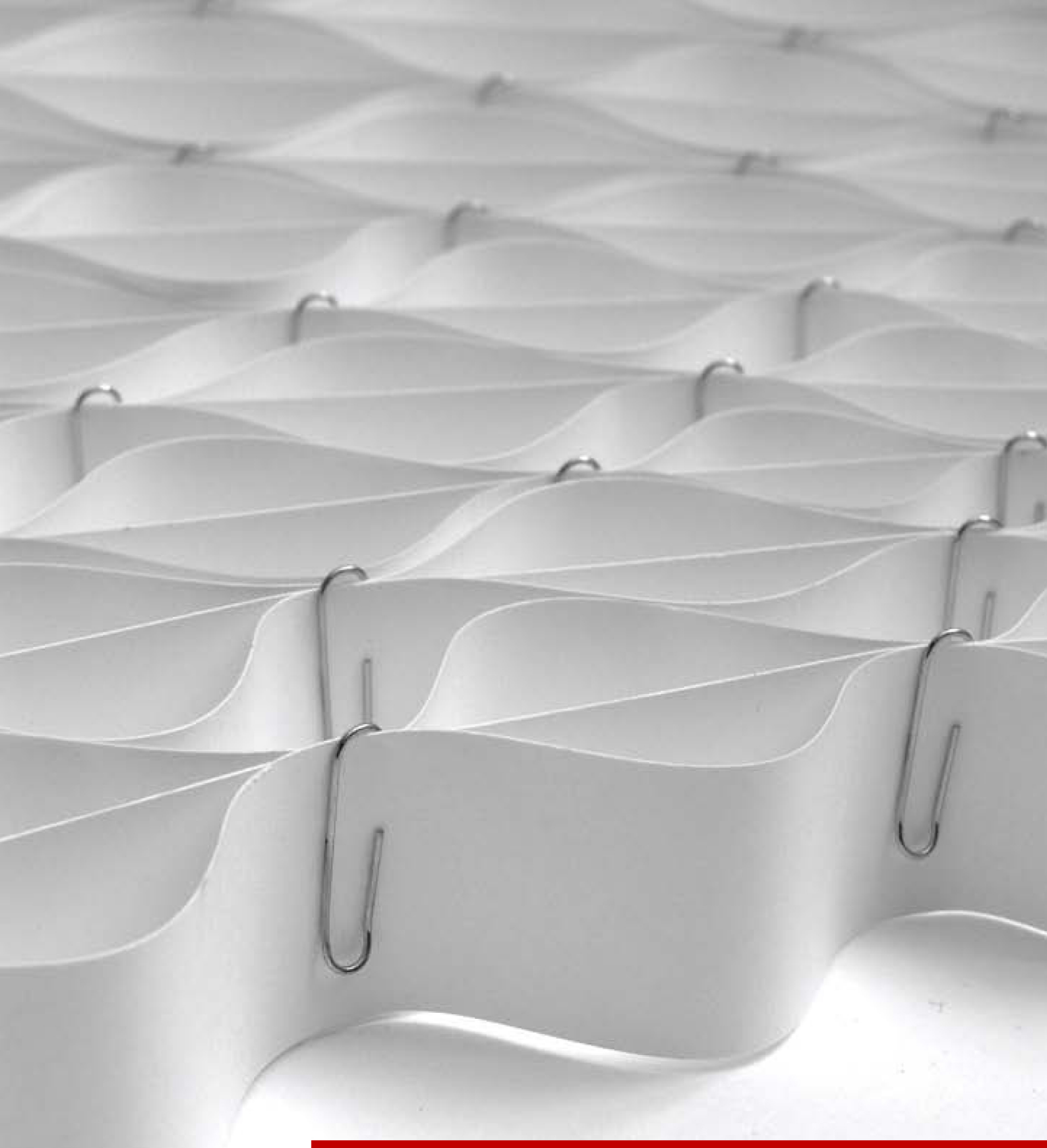


GENERATIVE ALGORITHMS

using GRASSHOPPER

한국어 번역: 김 한결

ZUBIN KHABAZI

`

GENERATIVE ALGORITHMS

using GRASSHOPPER

ZUBIN KHABAZI

한국어 번역: 김 한결

To see the latest updates visit my website or for enquiries contact me at:

www.MORPHOGENESISM.com

zubin.khabazi@googlemail.com

`

소개_Introduction

LEGO Mindstorms NXT robotic set 을 가지고 놀아본 적이 있으신가요? 통합적인 모델링(Associative Modeling)이란 이와 비슷한 것입니다. 이처럼 세상 모든 것들은 점점 더 알고리즘화, 매개변수화 되고 있습니다. 건축은 어떠한가요?

제가 AA(Architectural Association)의 EmTech(Emergent Technologies and Design)석사과정 재학 중에 접하게 된 Grasshopper 는 이러한 알고리즘과 매개변수를 이용한 통합적 모델링을 가능하게 하는 강력한 디자인 도구 입니다. 이와 관련된 저의 경험을 나누고자 이 책을 집필하게 되었습니다.

이번 두 번째 버전에서는 이 책의 이름을 'Algorithmic Modeling'에서 'Generative Algorithms'으로 바꾸게 되었습니다. 또한 아직 개발 중에 있는 Grasshopper 의 변화에 맞춰 새로운 내용들을 추가하였습니다. 이 책에 실린 튜토리얼들이 여러분이 Generative Algorithm 을 이해하고 이에 맞는 Grasshopper 의 사용법을 익히는데 도움이 되기를 바랍니다. Grasshopper 가 아직 개발 중이기 때문에 몇몇 튜토리얼들은 현재 버전의 Grasshopper 와 맞지 않을 수 있습니다. 앞으로도 지속적으로 이 글을 수정 보완할 수 있도록 하겠습니다.

이 책에 다소간의 보완 및 편집이 필요한 것은 사실입니다. 하지만 이 책은 영리목적으로 쓰여지지 않았으므로 여러분들이 이러한 부분을 양해해주실 것이라 생각합니다. 이 책을 무료로 배포하게 된 이후 전 세계에서 관심을 가져주신 덕분에 많은 친구들을 사귈 수 있었습니다. 이러한 관심에 진심으로 감사드립니다. 혹시 질문이나 어려운 점이 있다면 저에게 연락을 주시기 바랍니다.

감사의 글_Acknowledgements

먼저 Grasshopper3D 에 대한 지대한 관심과 지지를 보여주신 Bob McNeel 님께 진심으로 감사들 드립니다. 항상 영감을 주시는 David Rutten 에게도 감사를 전합니다. 이때까지 저를 가르쳐주신 많은 분들이 있습니다. AA/EmTech 의 Mike Weinstock, Michael Hensel, 저에게 매개변수의 개념과 컴퓨터 디자인 방법론을 가르쳐주신 Achim Menge, 컴퓨테이션과 스크립팅, 기하학을 지도해주신 Stylianos Dritsas (AA/KPF), Dr.Toni Kotnik (AA/ETH), 이 모든 분들에게 감사의 말씀을 전합니다. 또한 자신들이 가진 지식을 공유하고 또 이 책의 결점을 보완하는데 많은 도움을 주신 전세계의 건축가, 학생들, 디자이너들에게도 감사드립니다.

Zubin M Khabazi

March 2010

`

역자서문

두 번째 번역작업이었습니다. 추상적인 표현과 긴 문장이 많아 다소간의 의역이 있습니다. 최대한 원문을 부드럽게 해석하고자 노력하였으며 원문만으로 내용전달이 어려운 경우 '역자주'를 통하여 이해를 돕고자 노력하였습니다. Grasshopper component를 지칭할 때는 항상 <>을 사용하였습니다. 예를 들어 <Point>는 point component를 의미합니다. Component를 찾을 수 있는 경로는 그 옆 ()에 표기를 해두었습니다. 자유롭게 배포하시되 동의 없이 수정은 금지합니다. 번역에 대한 지적사항이 있으시면 이메일(geometricmind@gmail.com)로 연락을 주시면 감사하겠습니다.

번역 중간중간에 응원을 해주신 @Ssangheelee님 @kekshaus님 @goodkan님 @ChoungYongsu님 @consyne님 @flexxtudio님 @leejungboram 님 임권웅님, 그리고 유기찬선생님 조재원선생님 윤상훈선생님 교열을 봐준 과후배 고민재 이현수 모두 감사 드립니다. 또한 항상 저에게 가르침을 주시는 황지은 교수님께 진심으로 감사의 말씀을 전합니다.

저의 번역이 여러분이 grasshopper를 이해하시는데 조금이라도 도움이 되었으면 하는 바램입니다. 감사합니다.

서울시립대학교 디자인 정보 연구실
Design Informatics Lab, University of Seoul
석사 과정
Master course Student
김한결
Hangyeol Kim

Twitter: http://twtgeometricmind
Blog: http://geometricmind.wordpress.com

`

`

`

Chapter_1_Generative Algorithms

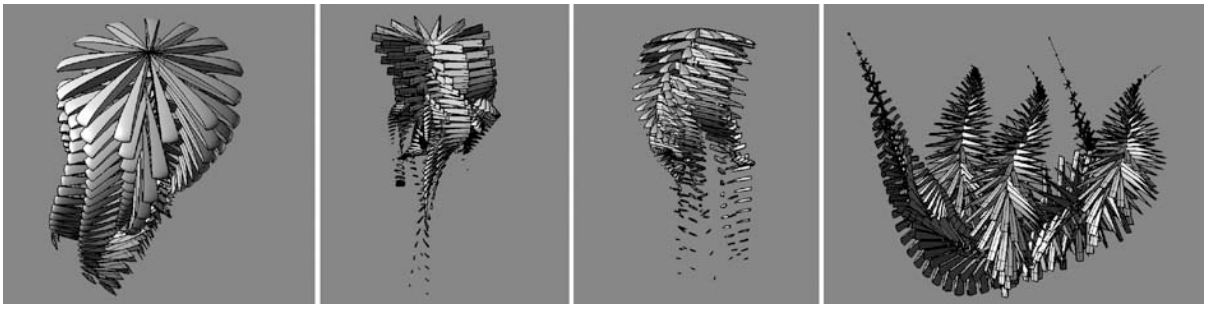
`

1_1 Generative Algorithm

건축을 공간에 표현된 하나의 개체로 보자. 우리가 그 개체를 디자인 하기 위해서는 그것 기하학적 특징을 이해해야 한다. 그리고 이를 위해서는 약간의 수학적 지식이 필요하다. 건축의 역사를 살펴보면 다른 양식들은 각자의 기하학적 특징과 표현의 논리를 가지고 있다. 또한 각각 시대마다 건축가들은 자신들이 처한 기하학적 문제들을 해결하기 위한 고유의 방식을 찾아내어왔다. 현대 건축에서 컴퓨터는 이제 건축가들이 공간을 시뮬레이션하고 기하학적 완결성(geometrical articulation)을 찾아내는데 많은 도움을 주고 있으며 디자인 프로세스에서 빠질 수 없는 일부가 되었다. 컴퓨터를 이용한 기하학(Computational Geometry)은 기하학과 알고리즘의 프로그래밍(algorithmic programming)의 결합의 결과물이며 그 자체로 연구를 하기 위한 흥미로운 주제가 되었다. 또한 이것은 Generative Algorithm 이라고 알려진 알고리즘을 이용한 기하체(algorithmic geometry)의 구현을 가능하게 하였다. 물론 3d 소프트웨어들이 거의 모든 공간을 시각화 시키는 것이 가능하다. 하지만, 건축의 영역에서 매개변수를 이용한 디자인(Parametric Design)과 같은 현재의 디자인의 가능성을 뒷받침 해주는 것은 바로 이 Generative Algorithm 이다.

건축가들은 기존의 방식이었던 '유클리디안 공간(Eucilidian Space)'의 한계를 넘어서 자유곡선과 자유곡면을 공간을 탐구하는데 사용하기 시작하였다. 건축과 디지털의 결합으로 인한 결과물이 이러한 물방울과 같은 형태(the blobs)들이 다뤄지는 것을 가능하게 하였으며, 그것에 대한 탐구를 더욱 심도 있게 하였다. 컴퓨터의 활용(computation)영역이 아주 빠르게 진행되고 있지만, 건축은 이것을 쫓아가는 정도로 변하고 있다.

이러한 물방울형 건축의 시대 이후의 현대건축은 이러한 주제 대해서 좀 더 정확한 개념을 가지게 되었다. 건축 디자인은 이제 알고리즘과 디지털 프로세스를 통하여 높은 복잡성과 다양한 위계들에 영향을 받는 기하체들의 잠재력을 이용할 수 있게 되었다. 전통적인 방법론을 가지고서는 곡면과 곡선을 요소로 가지는 자유로운 형태의 건물을 모델링하고 다양한 패턴과 요소들을 가지게 된 건축을 다룰 수 없다. 이것이 바로 알고리즘과 스크립트의 힘이며 이것을 이용하여 기존의 한계를 넘어설 수 있다. 명백한 사실은 바로 복잡한 기하체(complex geometry)를 다루기 위해서 우리는 적절한 툴, 즉 이러한 기하체를 구현하고 이것의 요소들을 제어할 수 있는 소프트웨어가 필요하다는 것이다. 그 결과로 건축가들은 Swarms 나 Cellular Automata 혹은 Genetic Algorithm 을 이용하여 알고리즘을 활용한 디자인들을 생산해내기 시작하였고, 그들이 기존에 가지고 있던 미디어의 한계를 뛰어넘어 공간과 형태를 다룰 수 있게 되었다.



Genetic Altorithm과 Evolutional Computation을 위한 Parametric Model, Zubin Mohamad Khabazi, Emergence Seminar, AA,

`

여기서 한발 더 나아간다면 재료의 속성을 알고리즘의 일부로 활용하는 것이 가능해진다. 디자인 단계에서 재료의 효과와 환경에 대한 반응을 확인할 수 있는 것이다. 이처럼 부분과 전체 시스템(components and systems)에 내재된 잠재력을 parametric design에 적용시킬 수 있어야 한다. 이 generative algorithm은 형태의 생성뿐만 아니라 재료가 구성되는 논리 자체를 그 안에 담을 수 있는 가능성을 가진 것이다.

"parametric design에 내재된 논리는 우리가 이때까지 사용했던 design 방법에 대한 대안이 될 수 있으며, 생산과정에 수반되는 제약들(manufacturing constraints)을 극복하기 위한 기하학적 정확성(geometric rigour)을 확보하고, 재료의 특성과 구축의 논리에 따라 component화 한 뒤, 이 component를 이것이 속한 시스템에 맞게 구성하고 조직하는데 이용될 수 있다. 이러한 접근 방식은 전체시스템에 필요한 매개변수 값을 구하고 이를 통해 전체시스템이 작동하는 방식(behavior)을 이해할 수 있다. 이것은 다시 이 시스템이 주변 환경 조건의 변화와 외부적 요인에 알맞게 대응하는 전략을 세우는데 사용될 수 있다. (Hensel, Menges, 2008)."

복잡한 객체 만들 때 일반적으로 그 design process는 간단한 객체로부터 시작하여 그것에 부가적인 것들이 덧붙는 방식으로 진행된다. 복잡성을 가지는 형태는 다양한 위계들 가진 부분들로 구성되며 그 부분들은 그것만의 논리와 detail을 가지게 된다. 이때 각 부분들은 서로 긴밀하게 연결되며 한 부분의 변화가 다른 부분에도 영향을 끼치게 된다. 그렇기 때문에 이러한 방법론을 '통합적인(associative)' 방법론 이라고 부를 수 있는 것이다.

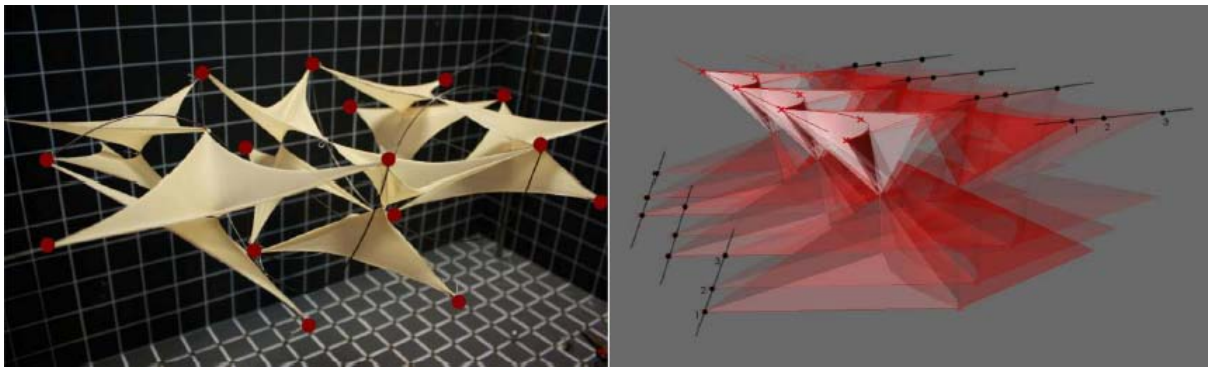
일반적으로 말해서 이러한 통합적인 modeling은 구축의 각 과정이 생성되는 기하체들이 그 다음과정에 생성되는 기하체에 영향을 주게 된다. 즉 각 요소들이 가지는 매개변수들 추출하여 그 다음단계의 요소들을 생성하게 된다. 이러한 과정을 반복하여 전체적인 기하체를 생성하는 것이다. 예를 들어 여러 개의 curve가 있을 때 각 끝점은 원을 생성하는데 필요한 중심점이 될 수 있다. 이때 curve의 형상이 변하고 끝점의 위치가 이동하면 원도 그에 따라 이동하게 되는 것이다. 이러한 design 방법은 많은 양의 data와 계산 과정을 필요로 하는데 이것을 algorithm을 이용하여 자동화 하는 것이다.

여기서 중요한 점은 바로 이러한 process의 각 단계에서 그려진 기하체들을 전체적으로 조율할 수 있다는 것이다. Design들 algorithm을 이용하여 각 요소들의 생성단계부터 그것이 가져야 하는 detail에 이르기 까지 통합적으로 다룰 수 있게 된다. Design의 결과물이 곧 Algorithm의 결과물이기 때문에 design 과정에서 생기는 문제에 유연하게 대처할 수 있는 것이다. 또한 자신이 가진 아이디어를 digital tool을 이용하여 그리고 이것의 대안 수백 가지 대안을 손쉽게 생성할 수 있다. Algorithm에 재료 고유의 특성과 조립 단계의 제약, 구축 방식을 대입하고, 또 이것을 이용하여 전체적인 시스템이 주어진 환경에 적응하도록 조절하는 것이 가능하다.

"parametric design은 기하학적 도형의 생성원리를 이해하고 이것에 작동하는 기능을 부여할 수 있게 한다. 또한 이러한 개개의 요소들을 전체 시스템의 필요에 맞게 조율될 수 있다. 다시 이 시스템은 외부 환경으로부터 지속적으로 발생하는 변화 속에서 매개변수의 조작을 통하여 국지적인 변화준다. 이는 시스템을 개체

`

발생적으로 발달(ontogenic development)시킬 수 있다. (Hensel, Menges, 2008)"



Membrane을 이용한 극소곡면(minimal surface)의 모형과 grasshopper를 이용한 membrane 움직임 모델링. Zubin Mohamad Kahazi, EmTech Core-Studio, AA, (Conducted by Michael Hensel and Achim Menges, fall 2008)

Grasshopper는 Rhino를 이용하여 Generative algorithm을 구축하고 통합적인 모델링을 가능하게 하는 도구이다. 이 후 이어질 예시들을 통하여 algorithm 을 이용하여 여러 기하학적 도형들의 특이성을 살펴보고 이것을 건축에서의 'Algorithmic' 방법론과 연관 지어 설명할 것이다.

`

Chapter_2_처음 시작하기

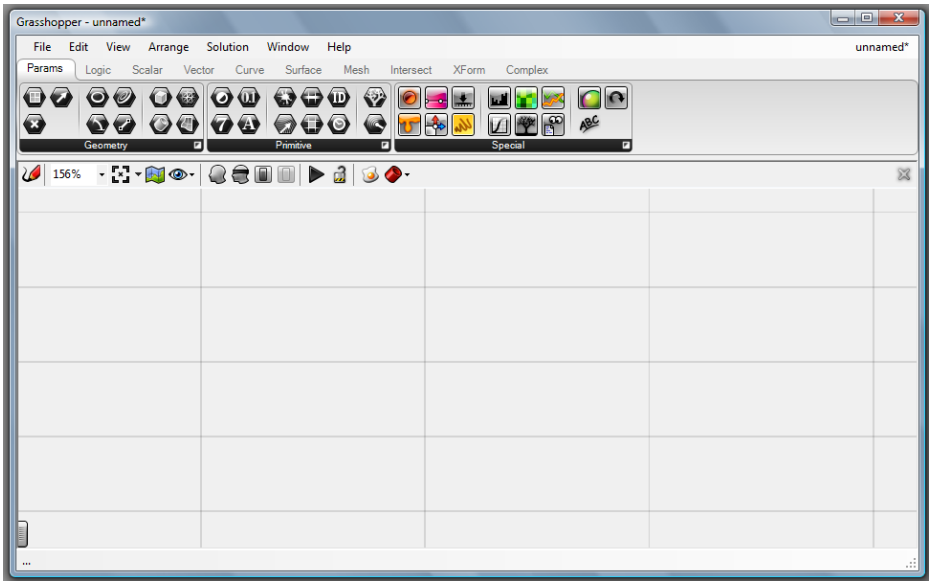
2_1_방법론 (Method)

2_2_Grasshopper의 기초(Basics of Grasshopper)

2_2_1_인터페이스(Interface, Workplace)

일반적인 윈도우 메뉴와 다르게 Grasshopper에는 '컴퍼넌트 패널(Component Panel)'과 '캔버스'라고 하는 두 가지 중요한 인터페이스가 있다. '컴퍼넌트 패널'은 디자인에 필요한 모든 요소들을 제공하며 캔버스는 바로 우리가 이 컴퍼넌트들을 배열하여 알고리즘을 만들 수 있는 곳이다. 패널 상에 있는 컴퍼넌트들을 캔버스로 드래그 하거나 클릭한 뒤 캔버스로 커서를 옮겨와 다시 한 번 클릭을 하여 캔버스에 컴퍼넌트가 생기게 할 수 있다. 다른 인터페이스들은 쉽게 이해할 수 있기 때문에 사용을 하다 보면 쉽게 이해할 수 있을 것이다. 다음 링크에서는 이것에 관한 더 많은 정보를 확인할 수 있다.

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

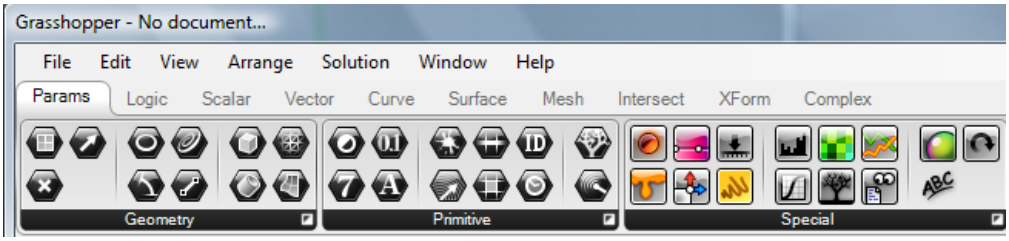


Grasshoper 컴퍼넌트 탭/패널과 캔버스

2_2_2_Components

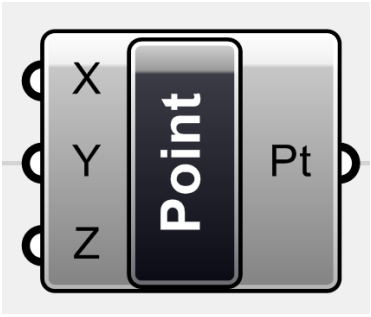
Grasshopper의 패널과 그 컴퍼넌트들은 'You can find them under ten different tabs called: Params, Logic, Scalar, Vector, Curve, Surface, Mesh, Intersect, XForm and Complex '으로 분류되어 있다.

`



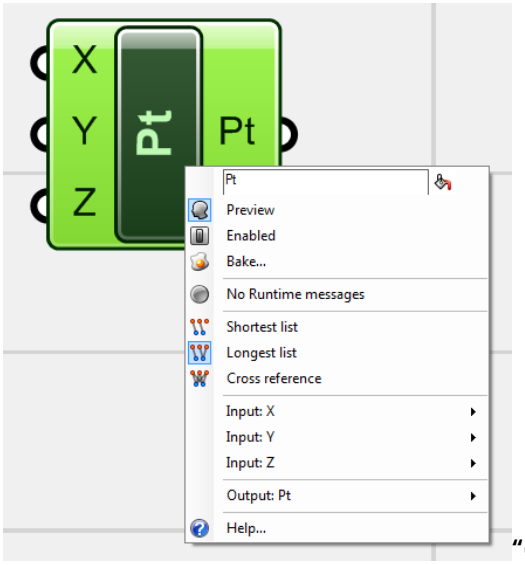
각각의탭들은 여러 개의 패널과 다른 개체들을 가지고 있으며, 커맨드들은 이 패널들 사이에 분류되어 있다. 이러한 컴퍼넌트들의 기하체(geometry)를 그리기 위한 line 이나 circle과 같은 것들로 부터 move, rescale, divide, deform 과 같은 커맨드들이 있다.

몇몇 컴퍼넌트들은 기하체를 생성하거나 data를 만들어내고, 몇몇 컴퍼넌트들은 이미 존재하는 기하체와 data를 활용하는데 사용된다. Rhino객체를 grasshopper와 연동시킬 수 있는 컴퍼넌트도 있고, 혹은 점과 선 같이 필요한 매개변수(parameters)를 서로 연결하여 라이노상의 객체를 만들어 내는 컴퍼넌트들도 있다. 몇몇 컴퍼넌트들은 move, orientate, decompose 와 같은 기능을 하기도 한다. Grasshopper에서는 여러 컴퍼넌트들을 알맞게 조합하는 방식으로 작업을 하게 된다.



<Point>컴퍼넌트

컴퍼넌트를 우클릭 하면 다음과 같은 메뉴가 나오게 된다. 이는 **"컨텍스트 팝업 메뉴(Context pop-up menu)"**로 컴퍼넌트의 기본적인 작동 방식들을 확인할 수 있다.

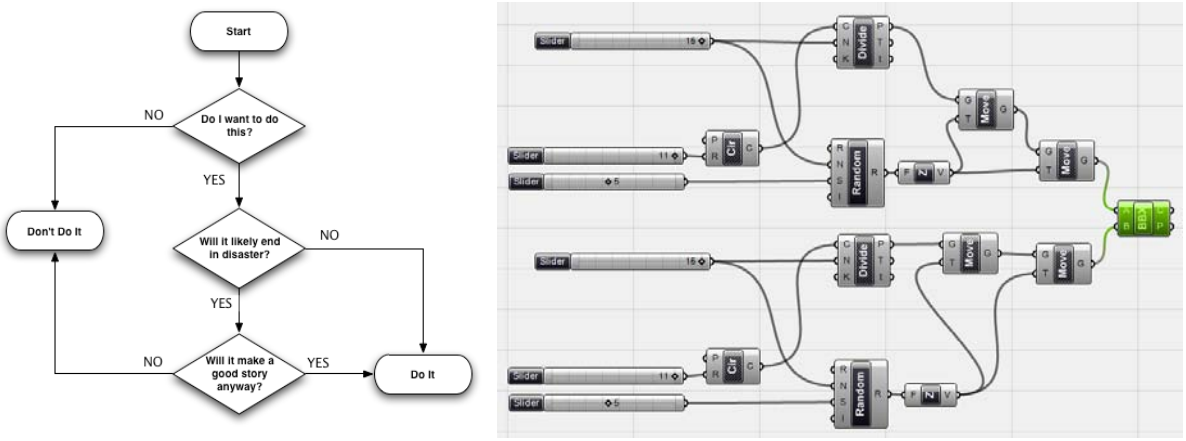


"Context pop-up menu"

지금부터는 서로 연결될 수 있는 컴퍼넌트들을 찾아 연결을 하여 Design Algorithm을 만들 것이다. 이것의 결과물은 Rhino 상에 나타날 것이다. 스크립팅(Scripting)이 알고리즘의 코드를 그대로

`

보여줌으로써 난해해 보인다면 Grasshopper는 이것을 Flowchart처럼 시각화 시켜주기 때문에 디자이너가 유연하게 알고리즘을 생성할 수 있도록 도와준다.

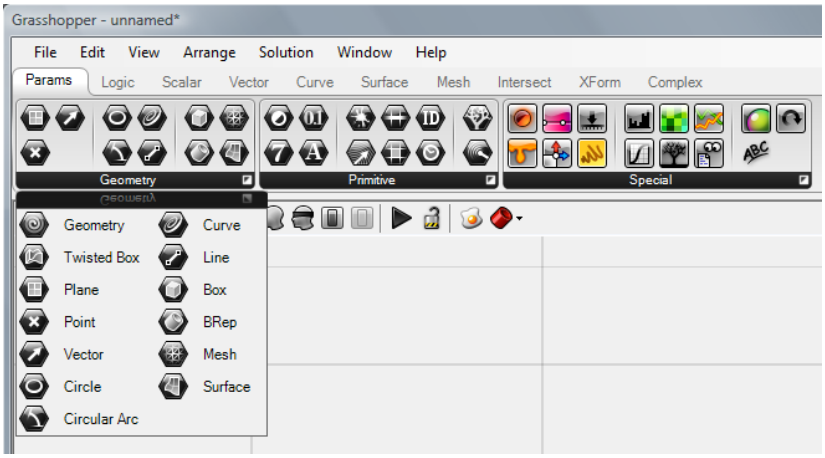


Flowchart vs. Grasshopper Algorithm

외부 기하체를 정의하기(Defining External Geometries)

여기서 소개하는 grasshopper definition 은 대부분 Rhino상의 객체를 Grasshopper에 연동시키는 것으로 시작한다. 이것은 point, curve, surface 와 같은 기본적인 객체부터 복잡한 형상을 가진 객체까지 모두 가능하다. 이것은 우리가 Rhino에서 그린 객체나 스크립트를 통해 생성된 객체까지도 grasshopper에서 연동이 가능하다는 것이다. 이러한 Rhino 객체를 grasshopper와 연동시키기 위해서는 Rhino 객체와 연동되는 grasshopper 상의 컴퍼넌트가 필요하다. 연동을 위해 필요한 컴퍼넌트들은 Params에서 찾을 수 있다. 이를 이용하여 Rhino 객체와 그래스호퍼를 연동시킬 수 있다.

Grasshopper 캔버스로 적절한 geometry component를 불러온 뒤, 우클릭을 context menu를 불러온다. 이 메뉴 중 "set one ... / set multiple ..." 를 선택하고 Rhino 객체를 선택하면 컴퍼넌트와 Rhino 객체를 연동시킬 수 있다.



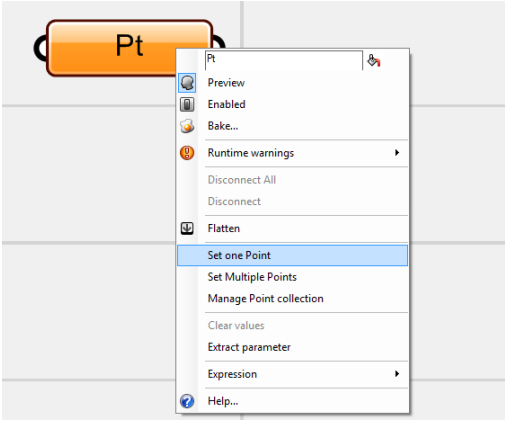
Params 아래의 여러 geometry 들

간단한 예를 들어보자.

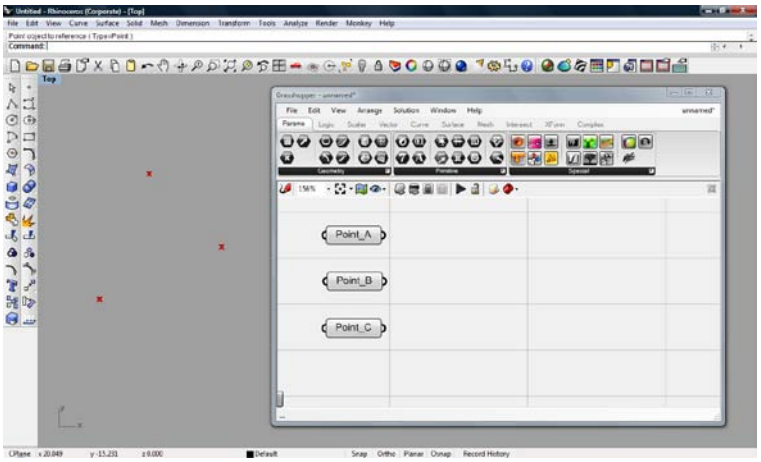
세계의 점을 만든 뒤 그 점들을 연결하여 삼각형을 그려보자. 먼저 그래스호퍼에 3개의 point

`

컴퍼넌트를 불러온다. 이것은 Params > Geometry > Point 에서 찾을 수 있으며 각각의 context menu(우클릭)을 연 뒤 'set one point'를 선택하고 Rhino 뷰포트에서 하나의 점을 선택한다.



Grasshopper 에서 Rhino 의 점<point> 지정하기



Grasshopper 캔버스에 나타난 세 개의 점들. 이는 Rhino 에서 빨간색 x 자로 표시된다. Point 컴퍼넌트들은 <point_A> <point_B> <point_C> 로 이름이 바뀌어져 있으며 이를 통해 무엇이 무엇인지를 쉽게 알 수 있다.

컴퍼넌트의 연결(Components Connectivity)

컴퍼넌트를 이용하여 만들어낼 수 있는 결과물은 매우 다양하다. 일반적으로 하나의 컴퍼넌트는 다른 하나 혹은 여러 개의 컴퍼넌트들로부터 data를 받으며, 이것을 취하여 지정된 기능을 하고 그것에 대한 결과값을 돌려준다. 이를 위해서는 컴퍼넌트들 사이를 선들로 연결해야 하고 이것을 반복하여 그 다음 결과물에 이르게 된다.

위 그림에서 제시한 3개의 점 예를 살펴보자. Curve 탭을 살펴보면 <line>을 찾을 수 있다.[1] 이것을 캔버스로 불러온 뒤 point_A를 A에, 그리고 point_b를 B에 연결한다. 컴퍼넌트들을 연결하기 위해서는 <point>의 우측을 클릭한 뒤 이것을 드래그하여 <line>의 인풋 방향에 넣으면 이것들을 연결할 수 있다. 라이노 view port를 확인하면 두 점 사이에 하나의 선이 그려지는 것을 볼 수 있다.

---

[1] 역자 주: 이 경우 양 선분을 정의하는 방법으로 양 끝 점을 주는 방법이 있다는 것을 생각하면 쉽게 이해할 수 있을 것이다.

`

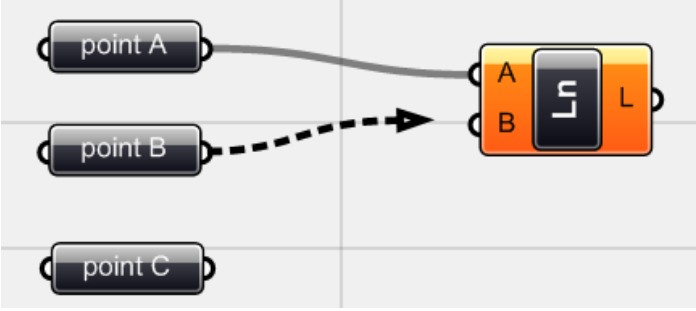
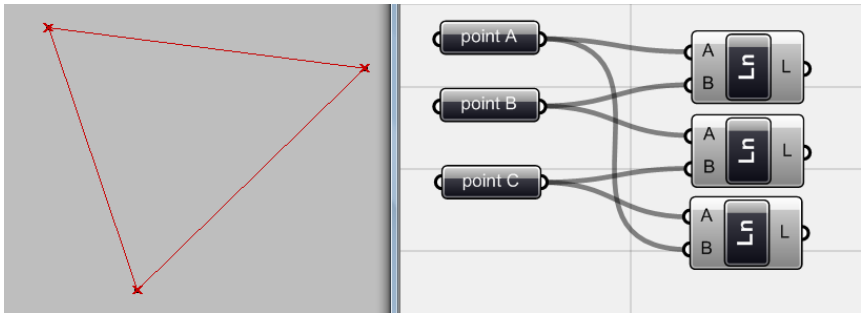
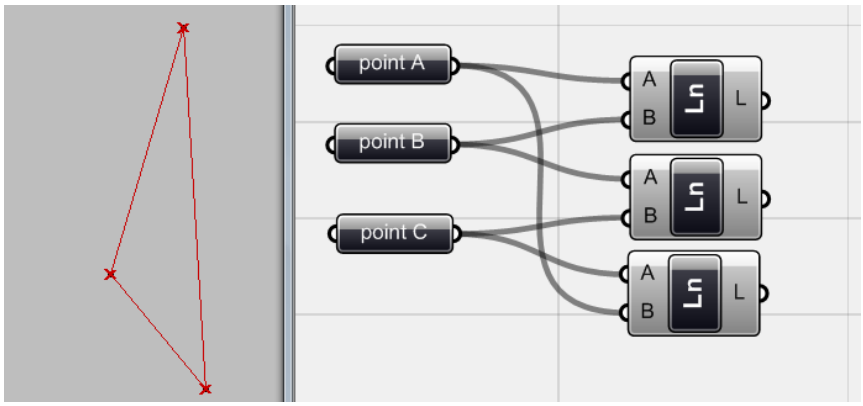


그림 2-6 <point>의 아웃풋을 드래그 하여 <line>의 인풋에 연결하기

이제 같은 작업을 <point_B>와 <point_C>, 그리고 <point_C>와 <point_A>에 반복하면 삼각형을 완성 할 수 있다.



<line>가 점들 사이에 선을 그리는 것을 볼 수 있다. 이 예를 통해 확인한 것 처럼 하나의 컴퍼넌트는 다른 여러 컴퍼넌트의 인풋으로 사용될 수 있다.



grasshopper의 점을 다른 곳에 위치 시키면 이것이 자동적으로 삼각형을 그려주게 된다.

이 예를 통하여 점의 위치를 변경하여도 삼각형은 이에 맞게 계속적으로 그려진다는 것을 볼 수 있다. 즉 기본적으로 grasshopper의 기본적인 아이디어는 컴퍼넌트들(feeding algorithm/input)을 놓고 이것들을 연결하고 다른 컴퍼넌트들을 더해가며 (algorithm's function) 디자인을 생성하는 것(algorithm output)이다. 이것의 자세한 과정은 앞으로 살펴볼 것이다.

취해지는 값/결과 값(Input / Output)

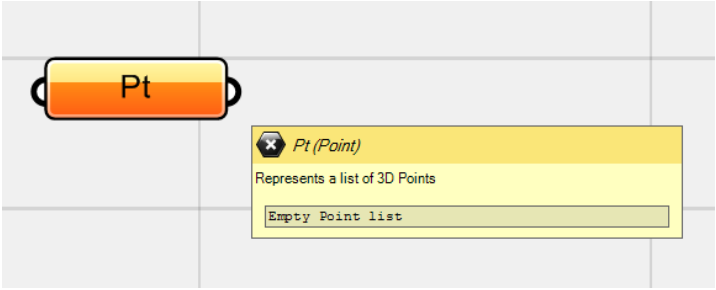
위에서 언급한 것 처럼 Grasshopper의 컴퍼넌트에는 취해지는 값(input)과 그 결과값(output)이

`

있다. 즉 grasshopper의 컴퍼넌트 들은 어떤 값을 취하고 이것을 처리하여 그것에 대한 결과값을 내놓는 것이다. 컴퍼넌트의 왼쪽이 input 오른쪽이 output이다. 즉, input data는 다른 컴퍼넌트로부터 나온 결과값이며 이것이 컴퍼넌트가 가지는 특정 기능의 output이 나오게 된다.

위의 내용에 대한 참고자료
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryVolatileDataInheritance.html [2]

특정한 컴퍼넌트가 가진 함수(function;기능)에 어떠한 input값이 필요한지를 아는 것이 중요하다. [3] 각 컴퍼넌트에 필요한 인풋값이 무엇인지는 앞으로 계속 살펴볼 것이다. 아무 컴퍼넌트나 캔버스에 놓은 뒤 그것의 input과 out에 마우스 오버를 해보면 팝업창이 트면서 컴퍼넌트의 이름과 어떠한 종류의 data를 그 컴퍼넌트에 input해야 하는지, 혹은 기본 값으로 설정된 data가 있는지, 혹은이 컴퍼넌트가 무엇을 하는 것인지 등을 보여줄 것이다.



컴퍼넌트의 input과 output에 마우스 오버를 하면 팝업창이 나와 그 결과값을 보여준다.

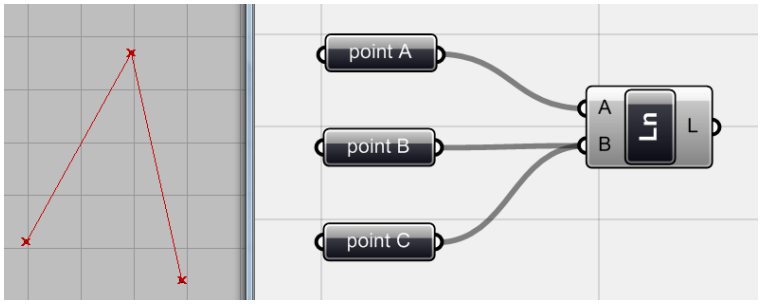
복수개의 data를 연결하기(Multiple Connections)

컴퍼넌트에 두 개 이상의 컴퍼넌트를 넣어야 할 경우가 있다. 위의 경우에 **<point A>**에서 시작하여 **<point B>**와 **<point C>**로 가는 두 선분을 만들 수 있다. 이 경우 <line>의 B 인풋에 두 점을 연결시켜주면 원하는 결과를 얻을 수 있다. 단 복수개의 data를 하나의 input에 연결하고 싶은 경우에는 shift 키를 누른 상태에서 연결을 하면 된다. 이 때 조그맣게 (+)가 보이게 될 것이다. 만약 특정 연결을 끊고 싶은 경우에는 Ctrl 키를 누른 상태에서 연결되어있는 data를 한 번 더 연결하게 되면 (-)표시와 함께 해당 연결이 없어지게 된다. (보통 context menu를 이용하면 연결을 끊을 수있다.

[2] 컴퍼넌트의 input data가 부족하면 주황색이, 잘못된 inputdata가 들어오면 빨간색이 된다는 내용이 있다.

[3] 역자 주: 즉 선분을 정의하기 위해서는 양 끝점을 input으로 주거나 선분의 시작점, 방향, 길이를 input으로 사용할 수 있다. 위에서 살펴본 **<Line>**의 경우 두 점을 그 input값으로 취하며 **<Line SDK>**의 경우 시작점, 방향, 길이를 이용하여 선분을 그리게 된다.

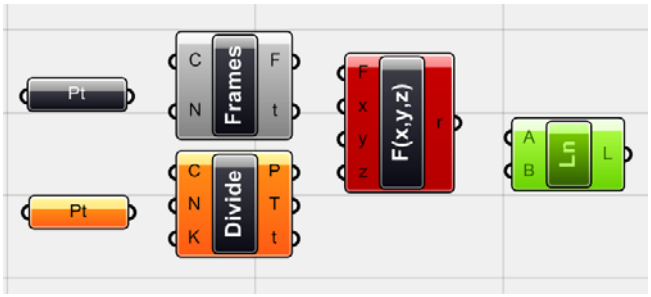
`



Shift 키를 누른 채로 컴퍼넌트를 연결하면 하나의 input에 복수개의 data를 연결할 수 있다.

색을 이용한 상태표시(Colour Coding)

Grasshopper의 컴퍼넌트 색에는 일정한 법칙(color coding)이 적용된다. 각 색은 컴퍼넌트의 작동 상태를 나타낸다.



Colour Coding

회색 컴퍼넌트는 필요한 data가 정상적이며 따라서 컴퍼넌트가 정상적으로 작동하고 있다는 의미이다. 주황색은 경고를 보여주는 것으로 컴퍼넌트에 적어도 한가지 이상의 문제가 있으나 컴퍼넌트가 작동하기는 한다는 의미이다. 빨간색은 에러(error)를 의미하며 이는 컴퍼넌트가 작동하지 않는다는 것을 의미한다. 이것을 해결하기 위해서는 data에 어떠한 문제가 있는지를 살펴보고 그것을 고쳐주어야 한다. 만약 컴퍼넌트에 에러메세지가 뜬다면 context menu를 통하여 무엇이 에러인지를 살펴본다. (context menu > Runtime warning/error) 그리고 인풋 data를 확인하여 에러의 원인을 찾아낼 수 있다. 초록색 컴퍼넌트의 경우 해당 컴퍼넌트가 사용자에 의하여 선택되었다는 것을 의미한다. 이 컴퍼넌트가 만들어낸 라이노상의 기하체 또한 초록색으로 바뀌게 된다. (기본적으로 기하체는 붉은색을 가지고 있게 된다.)

미리보기(Preview)

모든 컴퍼넌트들은 그것의 결과물을 라이노 상에서 '미리보기(Preview)'로 보여준다. Grasshopper에서는 이러한 기하체를 숨기거나(hide) 다시 보이게(unhide) 설정 할 수 있다. 컴퍼넌트가 생산하는 기하체가 숨겨진 경우 컴퍼넌트의 이름에는 빗금이 생긴다. 이러한 unhide 기능은 불필요한 같이 기하체들이 시각적 혼란을 주는 것을 방지하기 위해 사용된다. 특히 알고리즘을 통해 만들어진 결과물이 복잡한 형태를 가질 경우 이를 이용하여 data의 연산에 부담을 줄일 수 있다. 그러므로 시각화 시킬 필요가 없는 컴퍼넌트들은 숨겨가며 작업하는 것이 좋다.

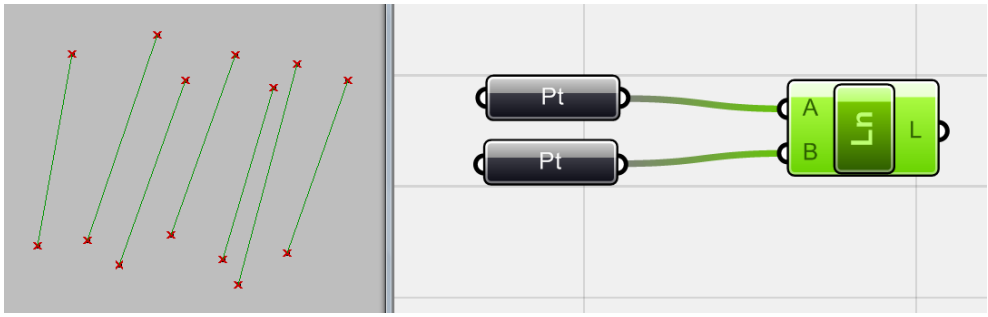
`

2_2_3_Data matching

대부분의 grasshopper 컴퍼넌트들은 하나의 결과값을 주기 보다는 결과값이 연속되는 data리스트(data list)를 산출한다. 예를 들어 점이 연속되는 리스트를 **<line>**에 연결하면 이 **<line>**은 여러 개의 기하체를 생산해낸다. 이러한 data리스트를 이용하면 하나의 컴퍼넌트만을 가지고 수백 개의 기하체를 그려낼 수 있다.

다음 예를 살펴보자.

각각 7개의 점을 가진 두 개의 data리스트가 있다. <point>를 불러온 뒤 context menu에서 set multiple points 을 이용하여 위에 있는 점들을 하나의 컴퍼넌트에 넣고 아래쪽의 점들을 다른 하나의 컴퍼넌트에 넣는다. 이렇게 되면 각각의 점들 사이에 7개의 선분이 만들어지게 된다. (그림 2.12 참조)



복수개의 점과 그것들에 의하여 만들어진 선분들

위의 경우 각 <point>가 취한 점의 개수가 같았지만 만약 이것이 다르면 어떻게 변하게 될까?

아래의 예에서는 위쪽에 7개의 점이 있고 아래쪽에는 10개의 점이 있다. 여기서 우리가 이해해야 할 것은 바로 grasshopper가 data를 관리하는 방식인 'data 매칭(data matching)'이다. 만약 <line>의 context menu를 살펴보면 다음 3개의 옵션이 있는 것을 볼 수 있다.

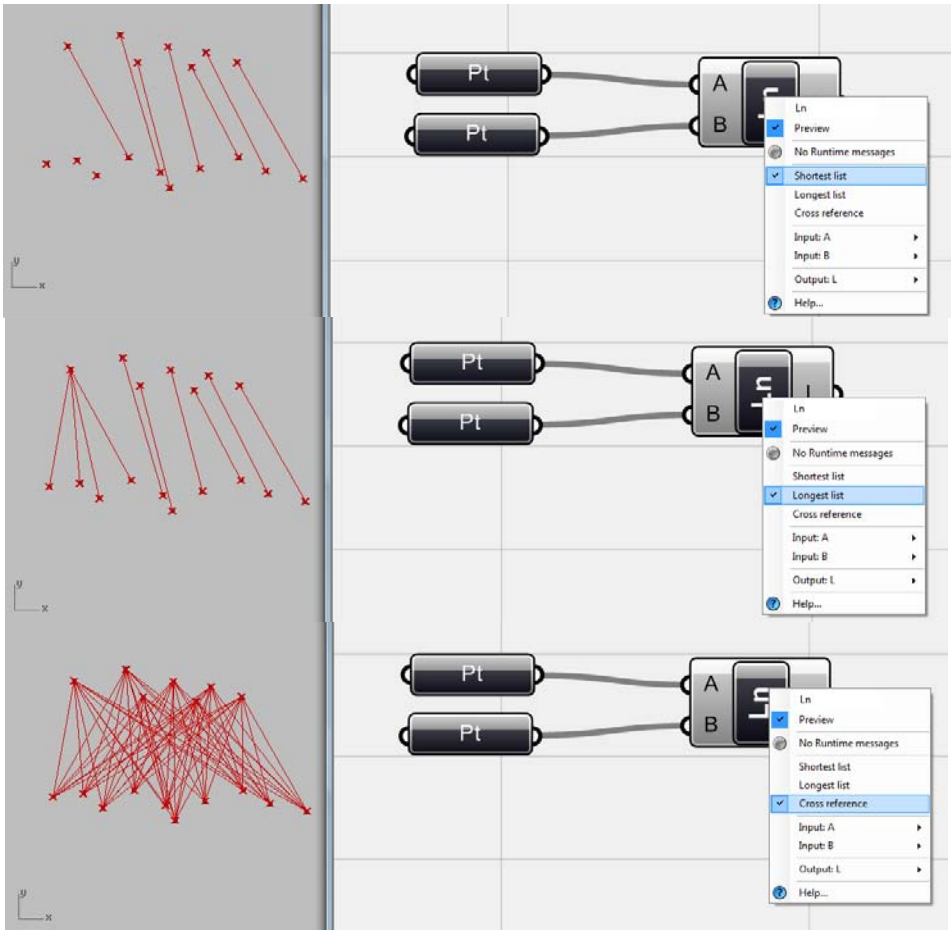
Shortest list
Longest list
Cross reference

아래 그림에서 그 차이를 확인하여 보자.

Shortest list로 설정할 경우 더 짧은 길이를 가진 data를 취하여 선분을 만들게 되고, longest list로 설정할 경우 한 점이 여러 번 연결 된다. Cross reference의 경우 점들간에 가능한 모든 연결을 찾아 그 사이에 선분을 생성해주게 된다. 이것은 컴퓨터의 memory를 많이 차지하는 옵션으로 변화된 미리보기를 업데이트 하는데 다소간의 사긴이 걸릴 수 있다.

`

더 많은 정보를 위해서는 다음 링크를 확인하기 바란다.
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryDataStreamMatchingAlgorithms.html



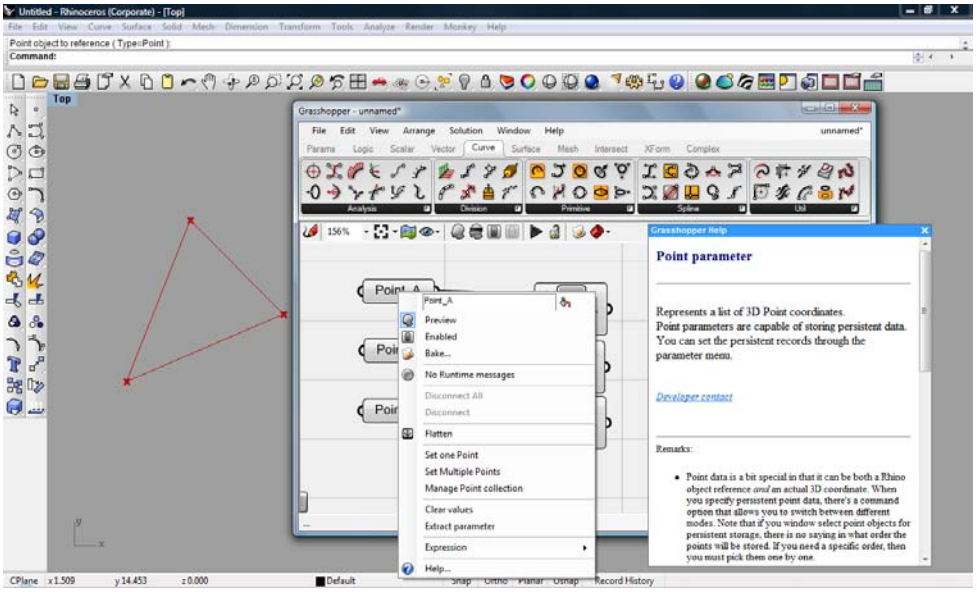
Data 매칭(Data matching) A: shortest list, B: longest list and C: cross reference [4]

2_2_4_Component's Help (Context pop-up menu)

모든 컴퍼넌트를 일일이 소개하는 것이 그렇게 유용한 방법은 아니며 사용자가 실험을 통하여 익히는 것이 좋다. Grasshopper의 여러 컴퍼넌트를 가지고 이것의 context menu에서 제공하는 help를 읽어보면 해당 컴퍼넌트가 어떠한 동작을 하고 어떠한 input이 필요하며 어떠한 output을 주는지를 확인할 수 있다. Context menu가 제공하든 다른 유용한 기능들에 대해서는 차차 다룩 될 것이다.

[4] 역자 주: 즉 7개의 점을 가진 data리스트와 10개의 점을 가진 data리스트를 연결하므로 shortest list에서는 7개의 선분이, longest list에서는 10개의 선분이, cross reference는 70개의 선분이 그려지게 된다. Longest list의 경우 더 짧은 7개의 점을 가진 리스트상의 마지막 점(data list상의 번호가 6번인 점)에 선분이 4번 연결되는 것을 확인할 수 있다.

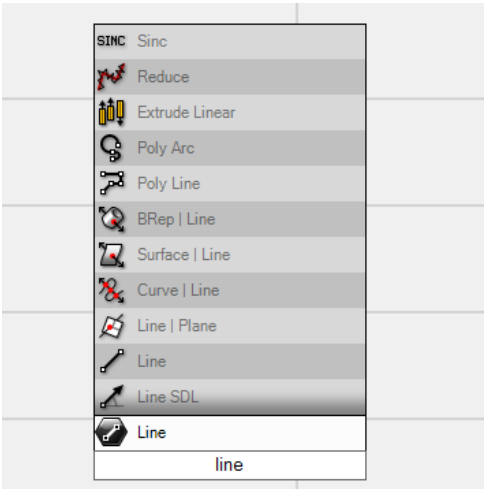
`



Context pop-up Menu와 Help

2_2_5_Type-In Component Search / Add

어떠한 컴퍼넌트를 사용하고 싶을 때 그 이름을 확실히 안다면 컴퍼넌트의 탭을 뒤지는 것 보다는 캔버스를 double click 하고 이름을 처넣는 것이 훨씬 빠르게 원하는 컴퍼넌트를 불러올 수 있다. 키보드를 주로 이용하는 사용자라면 이것이 매우 좋은 방법이 될 것이다.



컴퍼넌트 처서 찾기 를 이용하여 <line>을 찾기.
리스트가 나온 상태에서 원하는 것을 더블클릭하면 캔버스에 컴퍼넌트가 불러진다.

`

Chapter_3_data 묶음과 수학(Data Sets and Math)

`

Chapter_3_ **data 묶음과 수학(**Data Sets and Math)

수학적인 지식이 없어도 클릭만 하면 3d 소프트웨어가 3d 모델을 그리는 것을 도와준다. 하지만 Generative Algorithm을 이용하여 디자인을 하기 위해서는 기하학의 수학적 내용과 data의 흐름을 이해하는 것이 중요하다. 우리의 목적이 모든 것을 일일이 그리고자 하는 것이 아니므로, 이러한 기하체의 생성에 기본적으로 필요한 input들이 무엇인지 살펴보도록 할 것이다.

알고리즘의 작동 방식인 Workflow는 매우 간단하다. 이것은 data의 input을 받아드리고, 처리과정(processing)을 거쳐 data의 output을 만든다. 이러한 처리 과정은 알고리즘의 부분과 부분에서 일어나며 이것이 연결되어 전체적으로 일어나게 된다. 모든 객체들을 일일이 그리는 기존의 방법과 다르게 필요한 정보를 입력하면 이 정보는 알고리즘을 통하여 처리되고 기하체라는 결과물을 주게 된다. 좀 더 구체적으로 설명하자면 하나의 객체를 100번 복사하기 위하여 100번 클릭하는 것이 아닌, 알고리즘에 '해당 객체'와 'X의 양의 방향 (X positive direction)', '100번(100 times)', '사이 간격 3(space of 3)'을 입력하면 알고리즘이 자동적으로 그 결과물을 생성해주는 것이다.

우리가 기하체를 그리는 모든 행위 뒤에는 항상 그 수학적 배경이 존재한다. 이러한 알고리즘에서 수와 객체를 가진 간단한 수학적인 함수(math function)를 이용하면, 무한에 가까운 기하체의 조합을 생산해낼 수 있다. 이것은 수와 수로 이루어진 data들로부터 시작된다.

다음을 보면 이해가 훨씬 쉬울 것이다.

3_1_Numerical Data Sets

모든 수학과 알고리즘은 수로부터 시작한다. '수(number)'란 바로 세계의 뒤에 숨어있는 코드와 같다. 시작하기 전에 먼저 수와 관련된 컴퍼넌트들을 살펴보고 이것이 grasshopper에서 어떻게 작동하며 어떠한 결과물을 내는지를 알아보도록 하자.

단수 값(One numerical value)

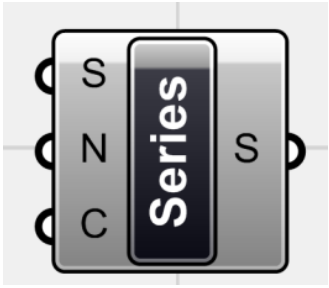
가장 유용한 생성자(generator)는 **<Number slider>** (Params > Special > Number slider) 이다. 이것은 내부의 원을 움직여서 수동적으로 수를 조절해줄 수 있다. 이것은 정수(interger) 소수(floating number), 홀수(odd number), 짝수(even number)[5]중 하나를 선택하고 이것의 최소값과 최대값을 설정할 수 있다. 이러한 설정은 context menu의'edit'을 이용하면 된다.

고정된 단수 값(one fixed numeric value)은 (Params > Primitive > Integer)의 정수**<Int>**혹은 수**<Num>**를 이용할 수 있다.

[5] 역자 주: 원문에는 integer, real, odd, even 정수, 실수, 홀수, 짝수로 표기되어있습니다.

`

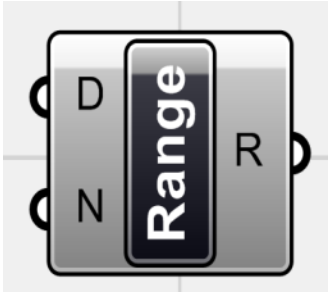
등차 수열(Series of numbers)



<series>(Logic > Sets > Series)는 수의 리스트를 생성해낸다. 이 컴퍼넌트의 경우 등차수열의 시작 수(S:first number), 등차(step size), 그리고 수열의 길이(number of values)를 설정해줄 수 있다.

0, 1, 2, 3, … , 100
0, 2, 4, 6, … , 100
10, 20, 30, 40, … , 1000000[6]

수의 범위 (Range of numbers) [7]



이는 가장 낮은 수와 높은 수를 설정한 뒤 이 범위를 특정 수로 나누어 그 결과값을 수의 리스트로 나타내게 된다. 이것은 (Logic > Sets > Range) 어떠한 값을 가진 수라도 (예를 들어 1~10) 이라면 유한개의 수로 나누어질 수 있다.

1, 2, 3, … , 10
1, 2.5, 5, … , 10
1, 5, 10 [8]

---

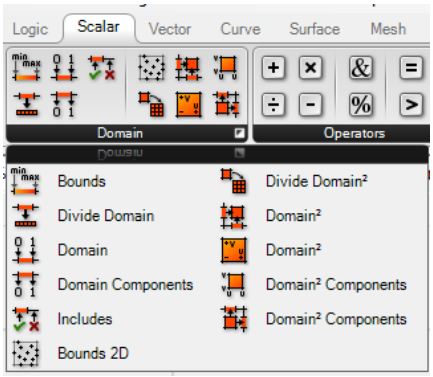
[6] 역자 주: 0, 1, 2, 3, … , 100 의 경우 S=0, N=1, C=100 / 0, 2, 4, 6, … , 100 의 경우 S=0, N=2, C=50. / 10, 20, 30, 40, … , 1000000의 경우 S=10, N=10, C=100000 입니다.

[7] 역자 주: 수학에서 일반적으로 series는 급수, sequence는 수열, interval은 구간, domain은 정의역, range는 치역을 의미합니다. 이 경우 range는 단순히 수의 범위를 뜻합니다.

[8] 역자 주: 이 경우 D를 통하여 1과 10을 최소값 최대값으로 설정합니다. 1, 2, 3, … , 10의 N은 1 / 1, 2.5, 5, … , 10의 N은 4, 1, 5, 10의 N은 2 입니다.

`

정의역 Domains (Intervals)



정의역 (Domain, 이전 버전에서는 'interval'이었음)은 최대값과 최소값 사이에 존재하는 실수의 범위를 만들어줄 수 있다. Domain은 1차 domain과 2차 domain이 있다. 이것에 관한 사항은 이후 좀 더 자세하게 다룰 것이다. 하나의 고정된 domain을 주기 위해서는 Params > Primitive > Domain/Domain² 컴퍼넌트를 이용할 수 있으며 Scalar>Domain의 경우 그 값을 유연하게 줄 수 있다.

Domain은 그 자체로 수를 결과값으로 주지 않는다. 이는 단지 최대값과 최소값을 정의 할 뿐이다. 두 수의 사이에는 무한개의 수가 존재한다. 다른 여러 가지 함수를 이용하여 이러한 범위를 나누고 이것의 결과값인 수들을 data 리스트로 만들 수 있다.

다음 예를 살펴보도록 하자.

3_2_점과 점의 grid(On Points and Point Grids)

점은 생성 알고리즘(Generative Algorithm)과 기하체에서 가장 기본적인 단위이다. 점은 공간에서 특정한 지점의 위치를 지칭하며 커브의 시작점이 되거나 원의 중심점, 혹은 평면의 원점(origin of plane) 이 될 수 있다. 이 외에도 많은 역할을 할 수 있다, Grasshopper에서는 이러한 점을 생성하기 위한 많은 방법들이 있다.

- 라이노에서 그려진 여러 점들을 선택하고 이것을 <point> (Params > Geometry > point)를 이용하여 grasshopper에 불러와 사용할 수 있다. (이러한 점들은 사용자가 일일이 움직이는 것이 가능하다. Chapter2 예시 참고)

- <point xyz> (vector > point > point xyz)에 x,y,z 좌표값을 넣어 사용할 수 있다. 혹은 이것에 필요에 따른 여러 종류의 data 묶음을 input으로 줄 수 있다.

- 점의 grid를 <grid hexagonal> 과 <grid rectangular>을 이용하여 만들 수 있다.

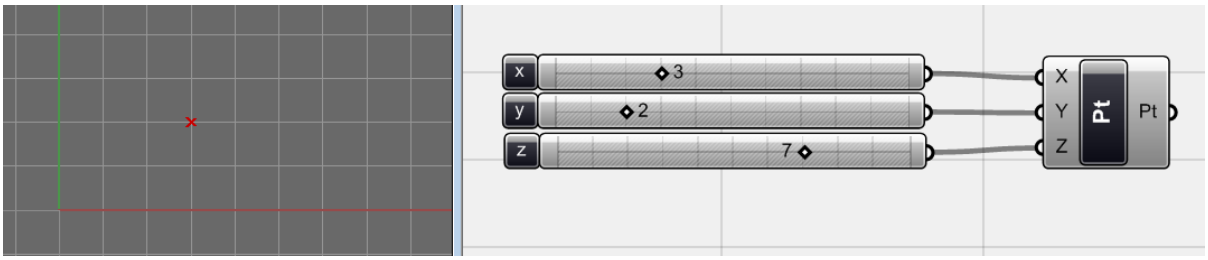
- 이미 존재하는 기하체로부터 점들을 추출하여 사용할 수 있다. 예를 들어 선으로 부터는 끝점이

`

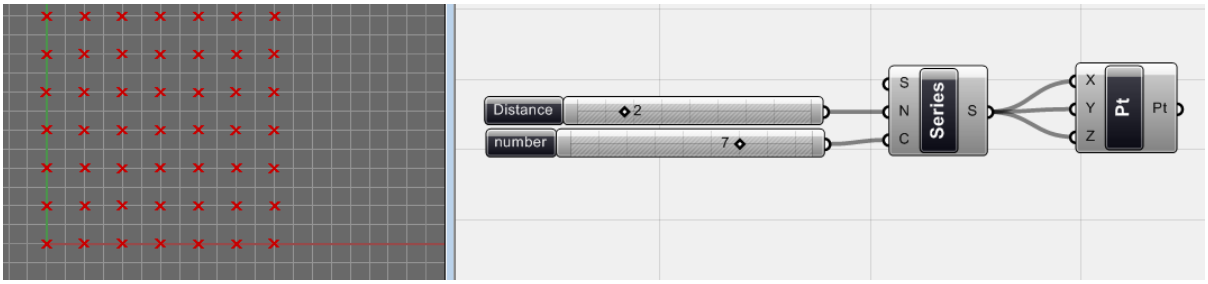
나 중점 등을 추출할 수 있다. (Chapter_3 23 data 묶음(Data Sets)과 Math GA_Ver.02 참조)

- 때로 우리는 하나의 평면과 벡터를 이용하여 점을 정의하고 기하체를 생성할 수 있다. 혹은 그 반대의 과정도 가능하다.

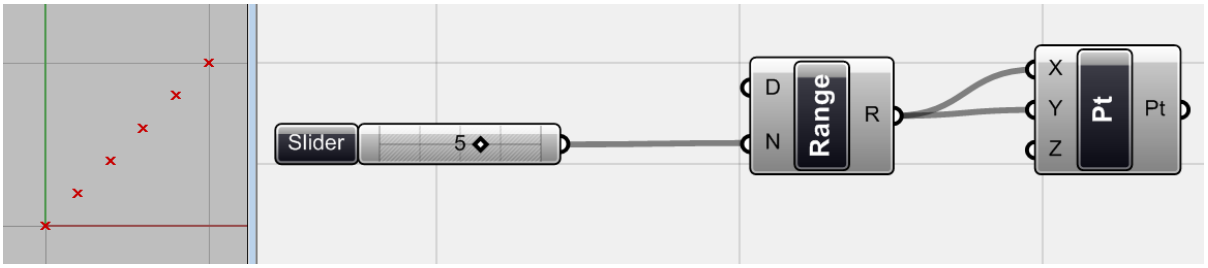
Chapter_2에서는 이에 관한 가장 기본적인 예시를 보았다. 이제 어떻게 하면 <series>, <range>, <number slider>나 다른 수치정보를 이용하여 점을 그릴 수 있는지 살펴볼 것이다.



<point xyz> 에 세 개의 <number slider>를 물려 X,Y,Z좌표를 각각 정의하여 하나의 점을 생성할 수 있다.



<series>, <XYZ point>, <number slider>를 이용하여 점의 격자를 만들 수 있다. 첫 번째 <number slider>는 점들 사이의 거리(step size)를 조절하며 두 번째 <number slider>는 grid에서 한 축이 가지는 점의 개수를 정의한다.[9] <XYZ point>의 context menu에서 data 매칭의 옵션을 cross reference로 설정해 주어야 위와 같은 결과값이 나타날 수 있다.



이 경우 0과 1사이를 5등분 하여 총 여섯 개의 수가 하나의 리스트가 있게 된다. **<XYZ point>**는 'longest list'로 data를 매칭하게 된다. 그렇기 때문에 점(0,0)과 점(1,1)사이에 점이 rhino에 생기게 된다. Range의 최소값과 최대값을 바꾸면 점의 좌표 또한 이것에 영향을 받게 된다. 위

[9] 역자 주: 이 경우 저자는 생성된 점의 grid를 top view로 보았기 때문에 사용된 이미지는 평면상에 생성된 그림 같지만 실제로는 z 축으로도 점이 생성된다. 그렇기 때문에 점의 개수는 총 7*7*7=329 이다.

`

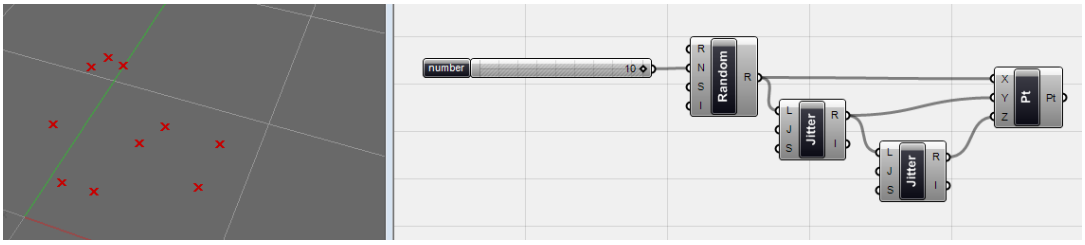
Range 컴퍼넌트의 D를 우클릭 하면 이것의 범위를 바꿀 수 있다. 이 외에 interval 등을 사용할 수 도 있으며 자세한 내용은 뒤에서 다뤄질 것이다.

위 예시는 그렇게 어렵지 않으므로 더욱 깊이 들어가보도록 하자. 위의 예시에 다른 값들을 주어 점의 위치나 점 사이의 거리를 바꿀 수 있다.

3_3_다른 수의 집합(Other Numerical Sets)

무작위 data 묶음(Random Data Sets)

다음으로 살펴볼 것은 무작위로 흩어진 점들에 대한 것이다. <series> 대신에 <random>(Logic > sets)을 이용하여 <point>에 input 값을 줄 것이다. <random>는 무작위의 수를 data 리스트로 그 결과값을 주며, input으로 수의 개수와 범위를 설정할 수 있다. 이 때 각X, Y, Z 좌표에 다른 값을 주고 싶으면 세 개의 <random>을 이용하면 된다. 이 때 다른 seed 값을 주어야 다른 값을 산출할 수 있다. 혹은 첫 번째 <random>의 output을 다시 무작위로 흐트러뜨릴 수 있는 <jitter>를 사용 할 수도 있다.

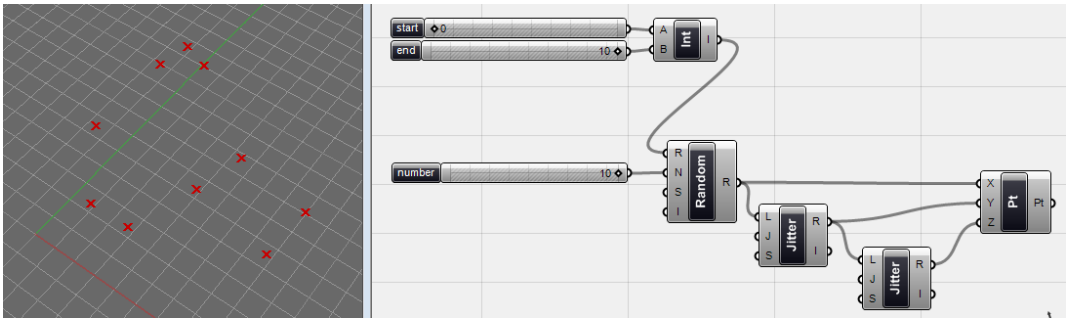


<random>은 10개의 무작위 수를 산출해낸다. (이 개수는 <number slider>에 의하여 제어된다.) 이 값은 다시 <jitter>(Logic > Sets > Jitter)에 의하여 뒤섞이게 되며 이렇게 뒤섞인 data의 리스트는 점의 y 좌표에 사용된다. 이렇게 나오는 값을 또 다시 <jitter>를 이용하여 z 좌표에 연결하면 된다. 만약 <jitter> 없이 <random>에서 나오는 값을 그대로 사용하면 점들이 특정한 패턴을 가지는 것[10]을 볼 수 있다. data 매치는 longest list로 해준다.

그림 3.4를 보면 모든 점들이 각 좌표 방향에서 0과 1사이에 생기는 것을 볼 수 있다. 이렇게 점이 분배되는 범위를 바꾸기 위해서는 <random>에 사용되는 수의 정의역(domain)을 바꿔주면 된다. 이를 위해서는 <random>의 R 값을 우클릭하여 'set domain'에 값을 직접 입력해주거나 <domain>을 이용할 수 있다. <number slider>를 이용하여 정의역의 범위를 바꿔주는 것이 가능하다. (그림 3-05 참조)

[10] 역자 주: 처음 <random>에서 나오는 값을 좌표 값으로 이용하면 점들이 하나의 선 위에 있는 것처럼 보이게 된다.

`



위에서 보다시피 정의역 값을 <domain>을 이용하여 변경하면 점이 생성되는 범위를 바꿔줄 수 있다.

<domain>

피보나치 수열(Fibonacci series)

점의 grid의 간격에 일정한 값이 아닌 점점 더 증가하는 값을 줄 수도 있다. <Fibonacci>를 잉용하면 된다.

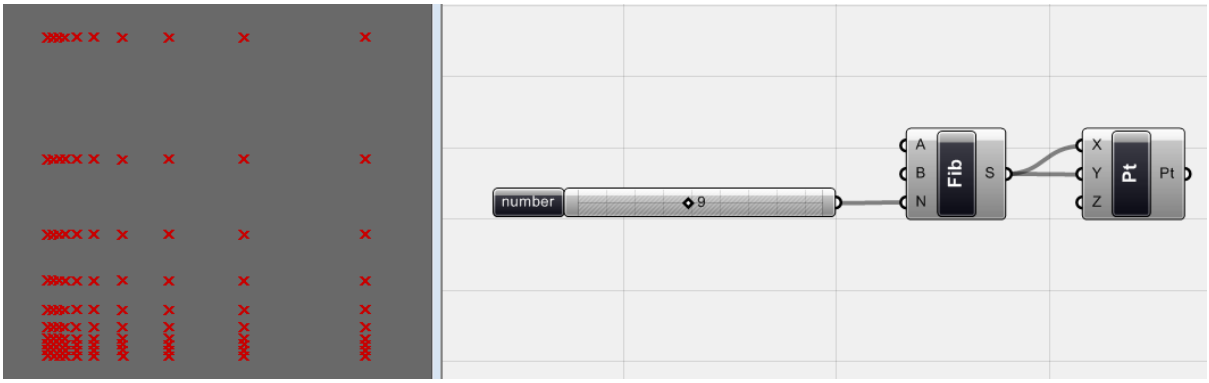
피보나치 수열이란 처음 두 수의 합이 세 번째 수가 되며 다시 두 번째 세 번째 수의 합이 네 번째 수가 되는 것이다.

N(0)=0, N(1)=1, N(2)=1, N(3)=2, N(4)=3, N(5)=5, … , N(i)=N(i-2)+N(i-1)

이러한 수열의 예시는 다음과 같다. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

위에서 볼 수 있듯이 수의 증가량이 무척 크다.

아래는 <Fibonacci> (Logic > Sets > Fibonacci) 를 이용하여 <pt>에 좌표 값을 준 것이다.[11]



<Fibonacci>를 이용하여 일정한 간격이 아닌 서서히 증가하는 수를 점들의 간격으로 사용하였다. 이 수의 개수는 <number slider>를 이용하여 제어될 수 있다.

[11] 역자 주: <Fibonacci>의 경우 A 와 B를 통하여 수열의 첫 두 수를 결정할 수 있으며 N은 수열의 길이를 결정한다.

`

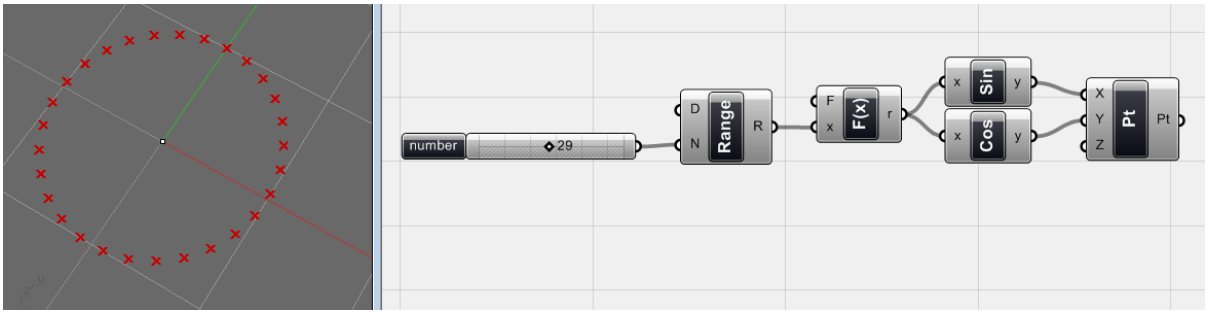
3_4_함수의 활용(Functions)

Grasashopper를 자신만의 디자인에 활용하기 위해서는 그 함수가 미리 정의된 grasshopper 컴퍼넌트들 만을 사용하는 것이 최선은 아니다. 사용자가 자신만의 data 묶음(data set)을 만들거나 기존의 컴퍼넌트의 data를 바꿔가며 활용할 줄 알아야 한다. 이를 위해서는 차수와 거리(distance) 등을 바꿀 수 있는 수학함수를 이용하는 것이 좋다. 함수(function) 컴퍼넌트들은 (Logic > script)에서 찾을 수 있다. 이러한 함수 컴퍼넌트를 이용하여 사용자가 스스로 정의한 함수를 만들어 낼 수 있다. (이러한 data는 숫자 뿐만 아니라 Boolean이나 string 또한 가능하다.) (F)를 우클릭 하여 함수를 직접 입력하거나 더블클릭 하여 Expression Editor를 키고 함수를 입력할 수 있다. 이 Expression Editor에는 미리 정의된 함수들이 존재한다.

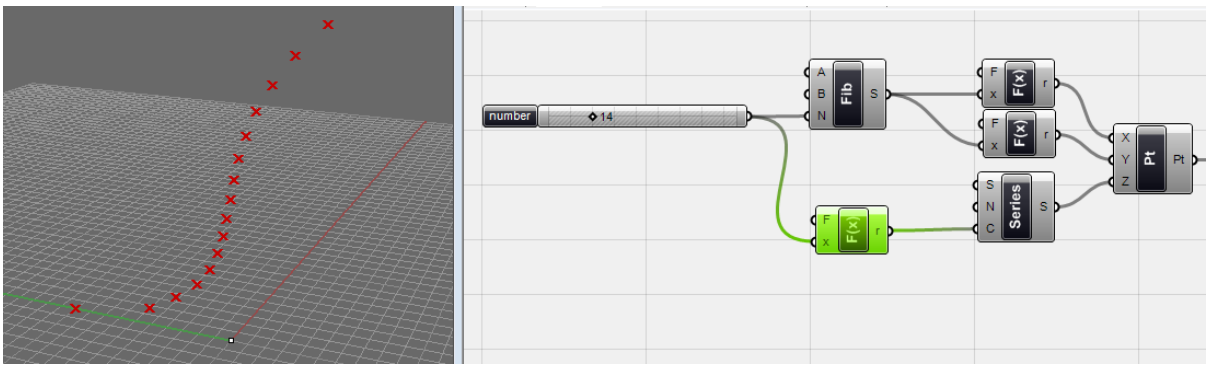
함수의 활용 (Math functions)

위에서 언급된 것 처럼 미리 정의된 컴퍼넌트를 이용하는 것이 항상 최선은 아니다. 하지만 원하는 결과를 얻기 위해서는 먼저 수학적 함수들과 그것의 인풋이 기하체를 만들어 내는데 어떠한 영향을 끼치는 지를 알아야 한다.

함수의 간단한 예를 살펴보도록 하자. 't'는 0과 2pi 사이의 값으로 X=Sin(t) , Y=Cos(t)로 정의된 함수이다. 이를 위해서 <range>를 이용하여 0과 1사이를 29개의 값으로 나눈 뒤 이것을 <f(x)>를 이용하여 2pi만큼 곱하였다. 이것의 결과로 원을 그릴 수 있다.



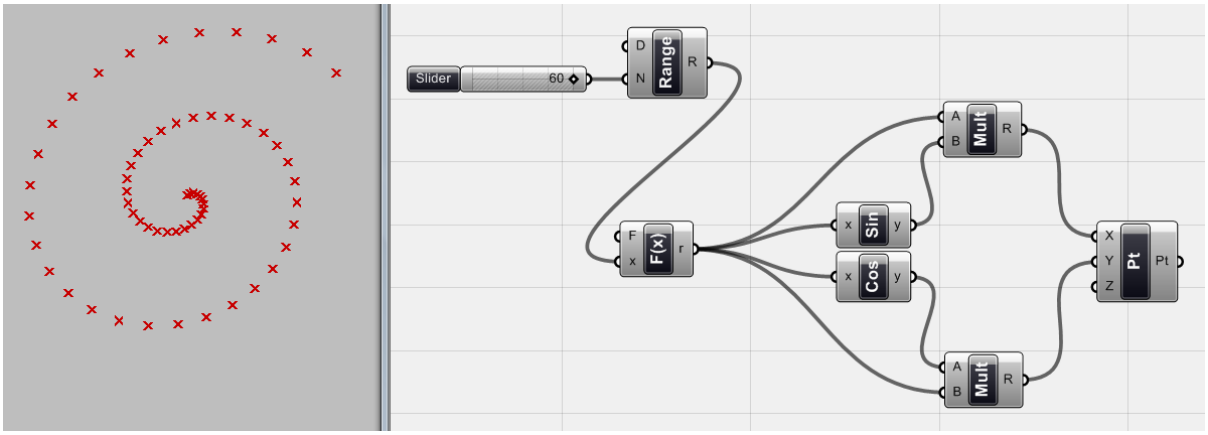
함수를 이용하여 만든 매개변수 원, (Scalar > Trig)에서 <Sin>과 <Cos>을 찾을 수 있으며 함수는 x * 2Pi 로 정의되어 있다.



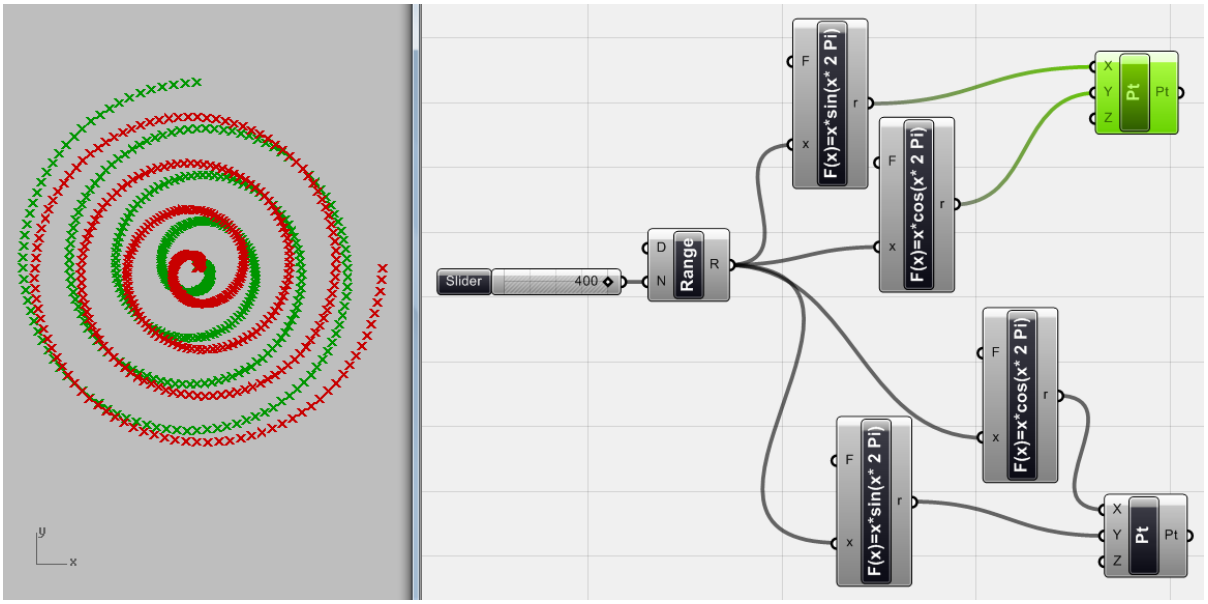
다른 예시로 <Fibonacci>와 다른 몇 가지 간단한 함수(x: F(x)=x/100, y: F(x)=x/10)에 의하여 정의

`

된 점들이다. 초록색으로 표시된 컴퍼넌트의 경우 <number slider>로부터 들어오는 값에 2를 더하여 (F(x)=x+2) <series>가 생성하는 수의 개수를 <Fibonacci>가 생성하는 수의 개수와 같게 만들어 주는 것이다. 이를 조작하여 쉽게 기하체를 만들어 낼 수 있다.

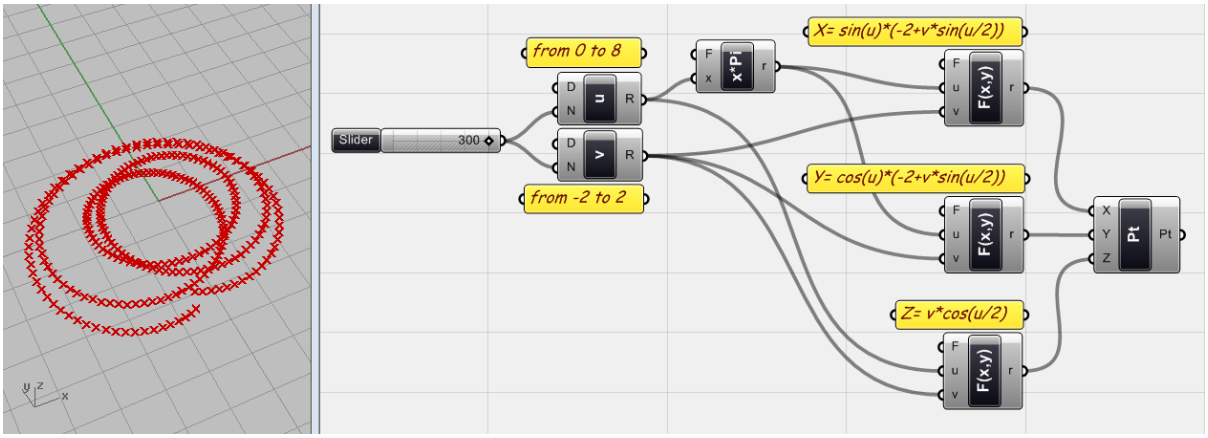


<range>는 0 부터 2 까지의 수를 60로 나누고 있으며 <Function>은 <range>로부터 나온 값에 2Pi를 곱한다. 이 값이 <sin>을 통하여 X=t * Sin(t) 되고 다시, <cos>를 거치며 Y=t * Cos(t) 가 된다. 이를 위하여 <multiplication>이 사용되었다. 이처럼 간단한 연산은 컴퍼넌트를 이용하여 계산할 수 있다. 각 값은 <pt>의 인풋이 된다.



조금 더 복잡한 예이다. Inter tangent spiral은 <range>는 0부터 4의 정의역을 가지며 이것을 400으로 나누고 있다. 첫 번째 <pt>의 X에는 F(x) = x * Sin(x*2 Pi), Y에는 F(x) = x * Cos(x * 2 Pi) 가 들어간다. 두 번째 <pt>는 이 것이 반대로 들어가 있다.

`



점의 그룹으로 이루어진 Moebius 이다. <u> 와 <v>는 <range>의 이름을 바꾼 것이다. 이 Moebius를 생성하기 위해 필요한 정의역은 위 그림에 표시되어있으며 함수는 아래와 같다.

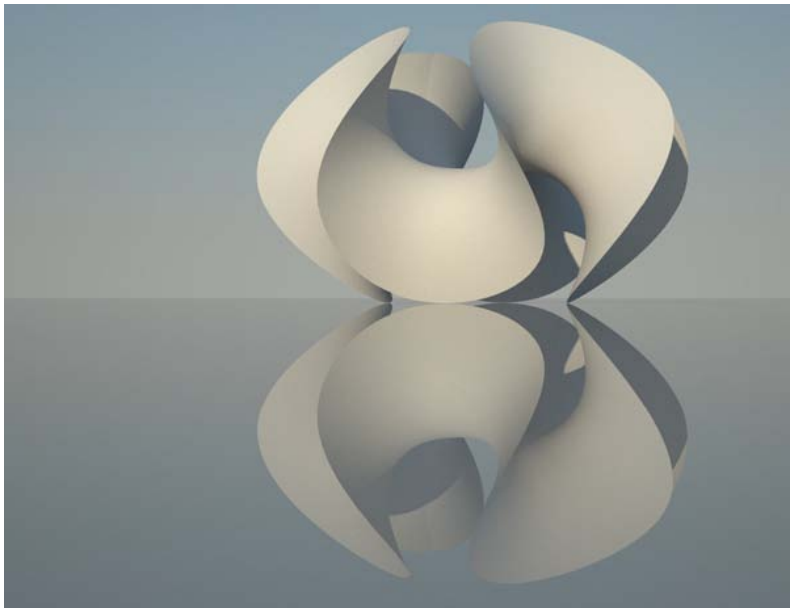
X= Sin(u)*(-2+v*sin(u/2))
Y= Cos(u)*(-2+v*sin(u/2))
Z= v*Cos(u/2)

위 값들은 모두 <point>에 input 되어 점을 만들게 된다.

함수를 이용하여 표현할 수 있는 작업은 끝이 없다. Grasshopper 에서 사용 가능한 수학관련 자료를 쉽게 찾을 수 있다. 중요한 것은 원래 주어진 data를 이용하여 다른 수치를 생성하고 이것을 다른 컴퍼넌트의 인풋으로 사용할 수 있으면 된다.

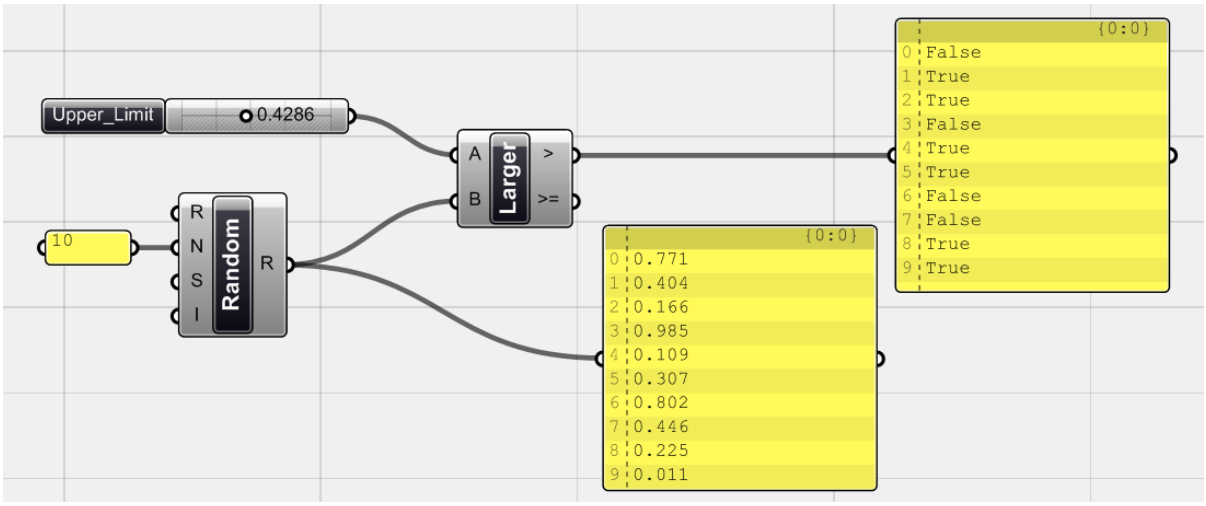
이처럼 간단한 수치값과 함수를 이용하여 복잡한 기하체를 생성할 수 있으며 이것이 바로 Algorithm이 작동하는 방식이다. 지금부터 기하학에서 중요한 개념들을 살펴보고 그것을 어떻게 이용하면 design을 할 수 있는지를 살펴보겠다. 아래 평면은 rhino의 수학 plug-in을 이용하여 생성한 Enneper surface 이다.

`



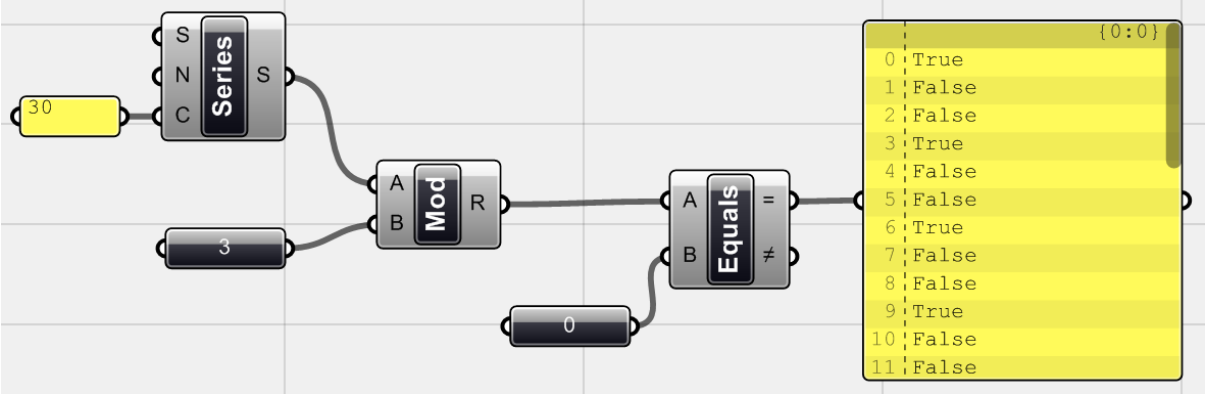
3_5_Boolean data 형식(Boolean Data types)

Data의 형식이란 수로 한정되어 있는 것이 아니다. 다른 Algorithm을 이용하여 다양한 결과물을 만들기 위해서는 다른 방식의 data들이 필요하다. 이 중 하나가 바로 Boolean data 형식으로 이는 'true', 'false' 값만을 가진다. Data list의 data가 특정 조건문(conditional statement)에 부합하는지 그렇지 않은지의 여부에 따라 그 결과물이 'true' 혹은 'false'로 나오게 된다. 이를 다시 다른 data list와 매칭하여 각 data의 사용 여부를 판단하거나 분류할 수 있다.



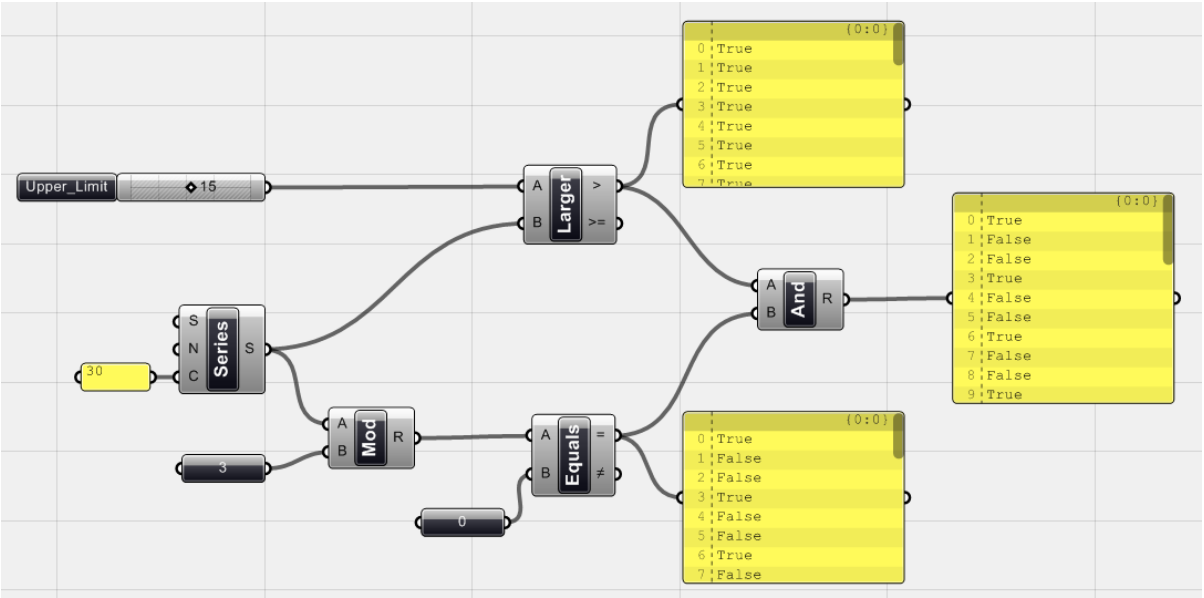
<random>(Scalar>Operators)을 통하여 무작위의 값을 가진 data 리스트를 생성한 뒤 <Number slider>에 의한 특정 최대값<Upper_limit>과 함께 <Larger>에 물린다. <Larger>는 A와 B에 들어오는 값을 비교하여 A보다 B작은 값을 true로, 더 큰 값을 false로 돌려준다. 위의 그림에서 <panel> (Params>Special) 을 이용하여 <Larger>의 작동 결과값을 보여준다.

`



이 다름으로는 <series>를 이용하여 (역자 주 :0부터 29까지의 정수 값) 30개의 값을 가진 리스트를 만든 뒤 이것을 3으로 나눈 값의 나머지를 <modulus> (Scalar > Operators > Modulus) 를 이용하여 찾는다. 이 나머지 값이 0과 같을 경우 <Equals>는 해당 값을 true로 바꿔준다. <panel>을 이용하여 이러한 결과를 확인할 수 있다.

위의 예시들에서 살펴 본 것처럼 우리가 가지고 있는 data의 값이 특정 기준을 충족시키는 지를 살펴볼 수 있다. 이러한 경우 그 결과는 true혹은 false인 Boolean data로 나오게 된다. 만일 우리가 가진 data의 값들이 다른 여러 가지의 기준을 충족시키는지 확인하기 위해서는 다른여러 Boolean 컴퍼넌트들을 활용할 수 있다. 이러한 컴퍼넌트 들은 logic 탭의 Boolean 에서 찾을 수 있다.



이 예시에서는 위에서 살펴본 두 개념을 합친 것이다. <Gate And> (Logic > Boolean > Gate And) 를 이용하여 <equals>와 <larger>를 모두 연결하였다. <gate and>는 인풋되는 두 값이 모두 true인 경우 true를, 그렇지 않은 경우에는 false값을 돌려준다. 결국 위의 연산을 모두 거쳐 true가 되는 값들은 바로 <upper_limit> (위 경우 15)보다 작으면서 3으로 나누어 떨어지는 수 이다.

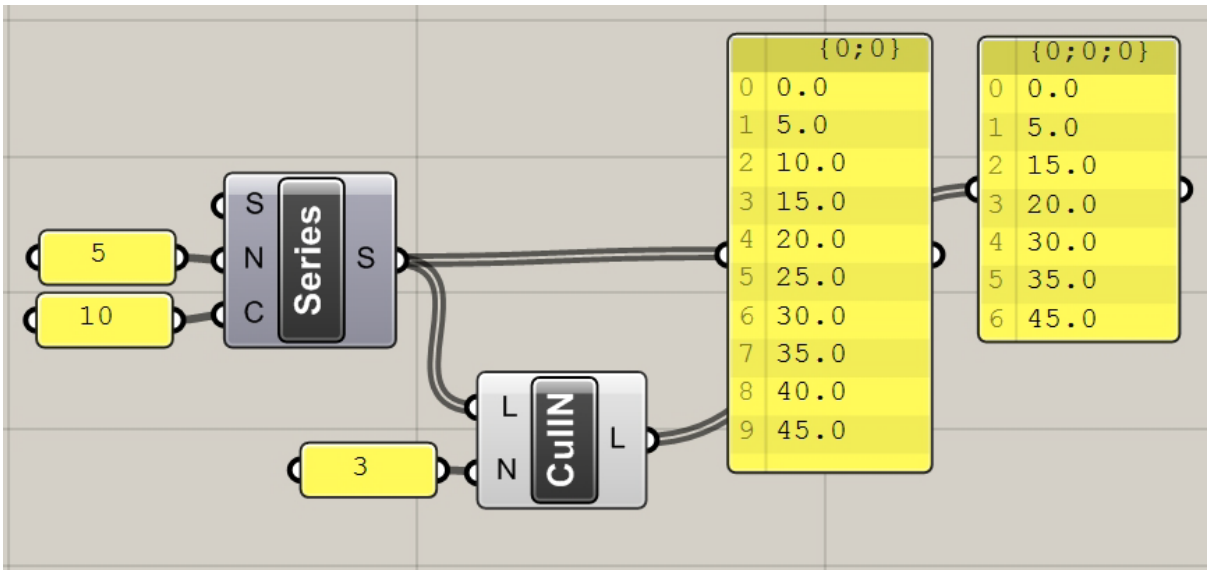
Logic 탭의 Boolean panel에 있는 컴퍼넌트들을 활용하면 사용자가 원하는 기준에 부합하는 값들을 골라내어 활용할 수 있다. 이러한 Boolean 값들을 design에 어떻게 활용할 수 있는 지를 나

`

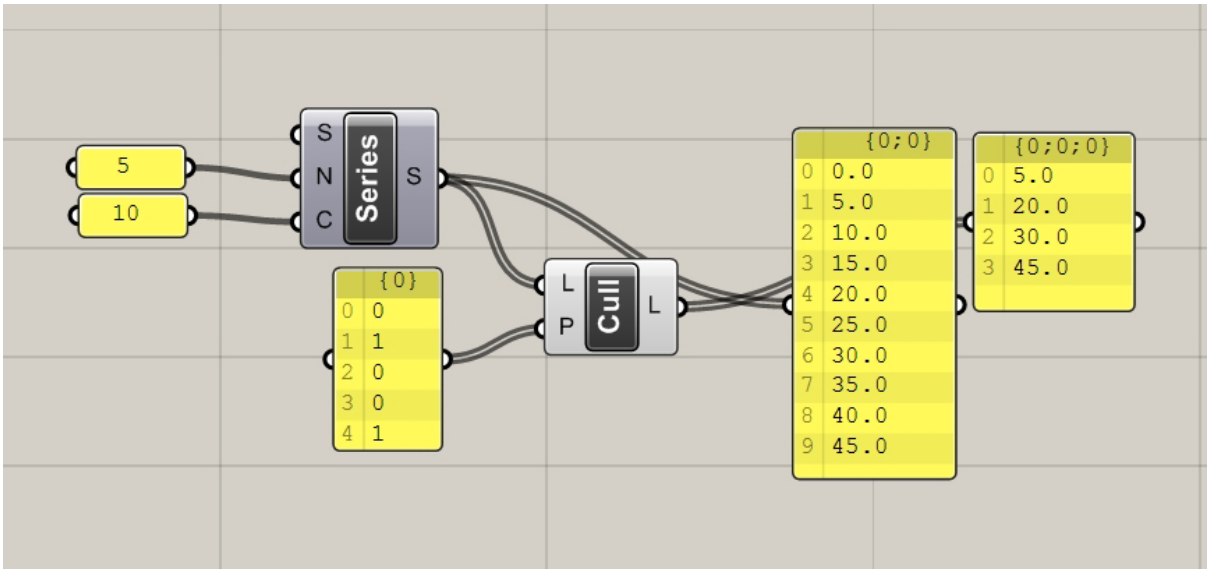
중에 살펴 볼 것이다.

3_6_Cull Lists[12]

주어진 data 중에서 필요한 값을 골라내어 사용해야 하는 경우가 있다. 이 경우 주어진 data 리스트에서 필요한 값만 사용하고 나머지를 버려야 한다. 이러한 논리를 적용할 수 있는 수 많은 방법이 있다. 이 중 data의 리스트를 cull(골라내기) 할 수 있다.



Cull N의 경우 data의 index 값과 상관 없이 N의 배수 번째 있는 값을 제외시켜 준다. 이 경우 N은 3으로 3번째 6번째 9번째 (index 값으로는 2,5,8) 값을 제외한 값이 리턴되었다.

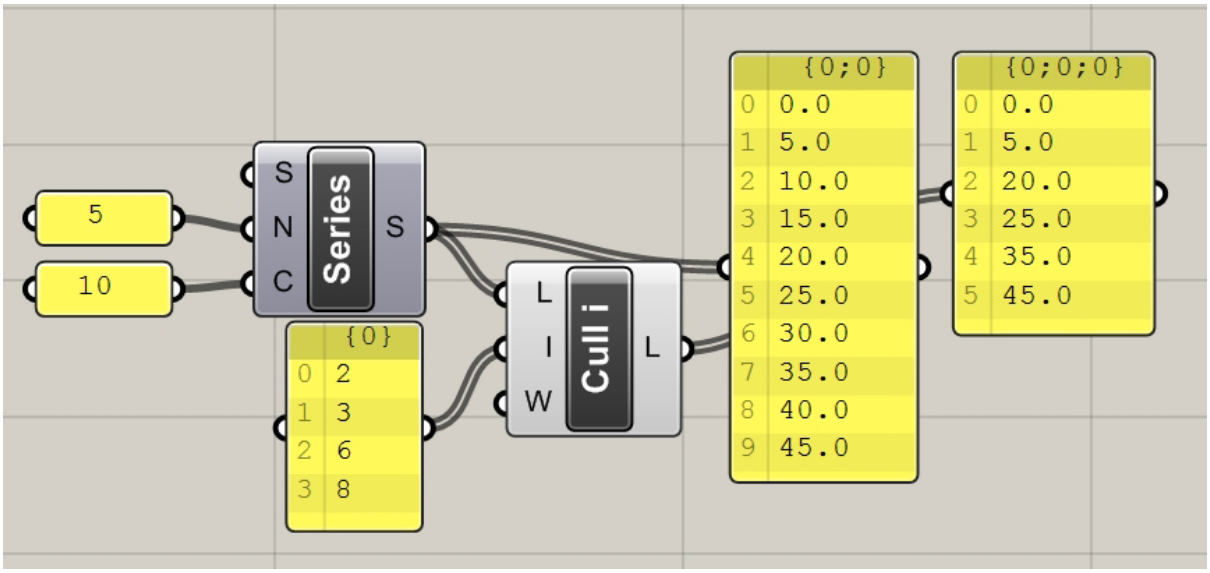


<cull pattern>의 경우 p 인풋이 가지는 true or false 의 패턴을 이용하여 L에 들어오는 data의

[12] 역자 주: 아래 Cull에 대한 설명 및 grasshopper 스크린 샷은 원문에 없는 것을 이해를 돕기 위해 추가하였다.

`

리스트에 패턴을 반복 적용하여 true 값만 리턴 해준다. 이 경우 panel을 이용 0,1,0,0,1 즉 false, true, false, false, true를 인풋으로 주었다. 그래서 나오는 값은 2번째 5번째 7번째, 10번째 값인 5, 20, 30 ,45 이다.



cull i의 경우 해당 하는 data의 index 값을 제외시키고 나머지 값을 리턴 해준다. 위의 경우 series의 2번 값인 10, 3번 값인 15, 6번 값인 30, 8번 값인 40을 제외한 나머지 값이 리턴 되었다.

<distance>의 활용 (Distance example)

한 점으로부터 특정거리 내에 존재하는 점들을 찾고자 한다. <point>에 의하여 미리 정의된 다수의 점들과 또 다른 <point>에 의하여 정의된 한 점이 있다. <distance> (Vector > Point > Distance) 를 이용하여 한 점과 나머지 다른 점들간의 거리를 계산해준다. 이 경우 <distance>는 거리 값을 datalist로 내보낸다. 이 것을 <F2> (Logic > Script > F2 / function with two variable)의 Y에 연결한다. X에는 <number slider>를 이용하여 사용자에 의해서 정의된 값을 연결한다. <f2>에 'x>y'를 넣어주면 f2는 들어오는 두 값들 비교하여 x보다 더 작은 y 값을 true로 하고 더 큰 값을 false로 하는 Boolean data를 내보내준다. (<Larger>와 같은 기능이다.) 이것을 <cull pattern>의 p 값에 연결하고 L값에 복수개의 점이 정의된 <point>를 연결해준다.

위에서 언급했듯이 <cull pattern>은 generic data list와 Boolean data를 인풋으로 하여 그 두 데이터의 리스트를 Data Matching 해준다.[13] Boolean data의 false와 Matching 이 되는 것은 없애주고 true와 Matching 이 되는 것은 그대로 내보내 준다. 이 경우 사용자가 지정한 거리보다 먼 거리 값들은 false가 되고 그 안의 거리들은 true가 되기 때문에 이 패턴을 <point> 리스트에 그대로 적용하면 해당 거리 값을 가진 점은 리스트에서 사라지게 된다. 아래 그림에서는 그렇게

[13] 역자 주: 이 때 Boolean data가 generic data list보다 짧은 경우 이것을 반복 적용해준다.

`

찾아진 점을 더 확실하게 보여주기 위하여 <line>[14]으로 연결하였다.

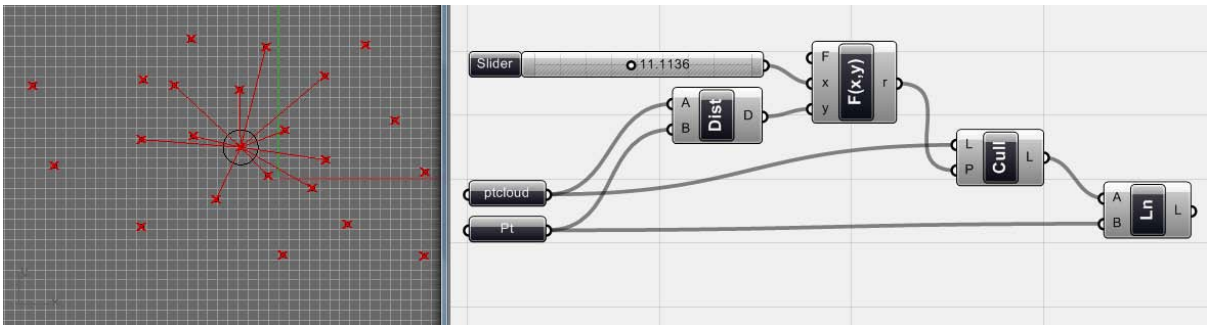
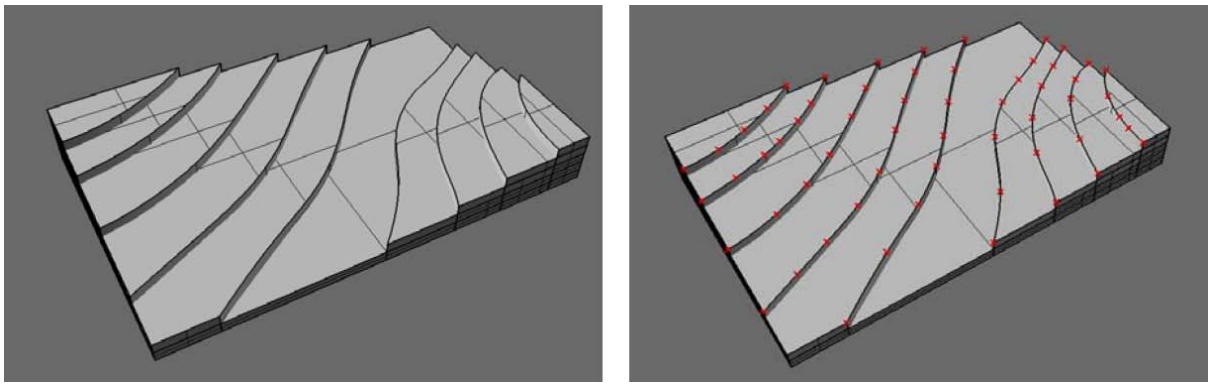


Fig.3.15. 점을 기준점으로부터의 거리를 기준으로 골라내었다. <Cull pattern>이 사용되었다.

Topography example

위에서 간략한 <distance>의 활용을 살펴보았다. 이번 예시는 콘타선 위에 있는 점들을 그들의 높이 값을 기준으로 골라내는 것이다.

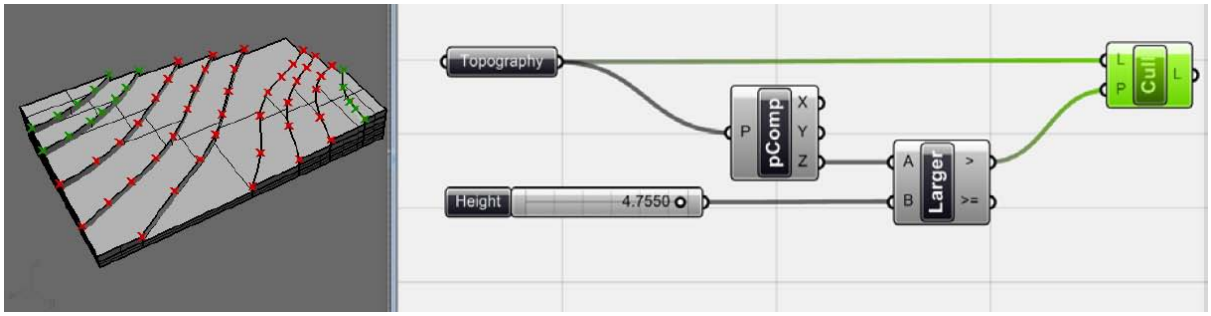


지형과 콘타선 위의 점들

위의 예시에서는 <point>에 의해서 정의된 (이 경우 topography로 이름을 바꾸었음) 점들의 그룹이 있다. 이 점들을 위 distance logic 과 비슷한 방식으로 골라낼 수 있다. 먼저 <point decompose> (Vector > Point > Decompose)를 이용하여 각 점들의 z좌표 값을 얻는다. <Point decompose>는 인풋으로 들어오는 점의 x,y,z 좌표값을 아웃풋으로 내보내준다. 이렇게 얻어진 z 값을 <larger>를 이용하여 사용자가 <number slider>를 이용하여 지정한 값과 비교하여 Boolean data를 얻는다. 이 것을 <cull pattern>을 이용하여 점들의 데이터리스트와 데이터매칭을 시켜주면 지정 높이보다 더 높이 있는 값들을 얻을 수 있다.

[14] 역자 주: 해당 <line>의 경우 양 끝점을 통하여 정의된다. 이 경우 기준이 되는 한 점 (reference point)을 한 끝점으로 하고 사용자가 지정한 값 이내의 점들을 다른 한 점으로 하는 line들이 생기게 된다.

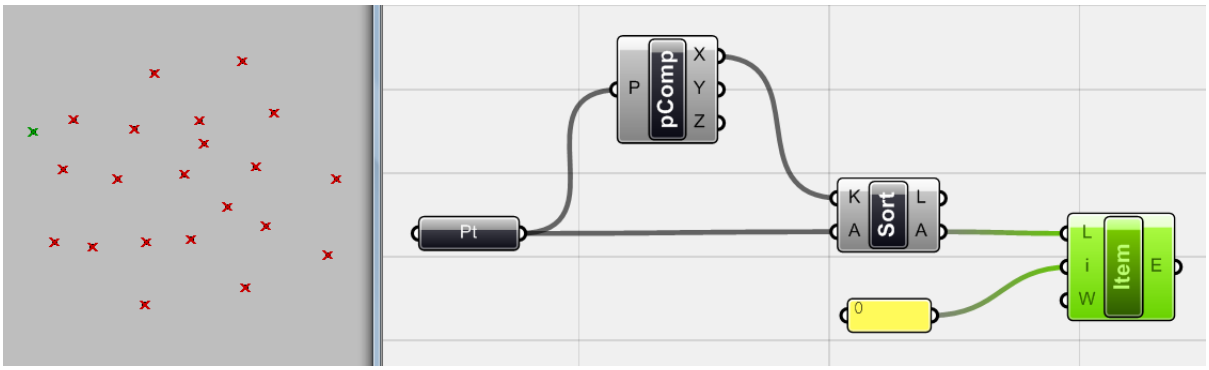
`



. 사용자가 지정한 4.7550 보다 높은 점들이 선택 되었다. 이제 이 점들에 소나무 객체를 가져다 놓자.!!

3_7_ Data Lists

위의 예시에서 살펴보았듯이 algorithm을 이용한 모델링의 기초 중 하나는 바로 data list라는 것이다. 수, 점, 기하체 등 어떠한 종류의 data라도 data list가 될 수 있다. Logic 탭을 살펴보면 data list를 이용할 수 있는 여러 가지 컴퍼넌트들이 있다. 사용자가 지정한 index number의 데이터를 골라주거나 사용자가 지정한 범위 내의 index number를 가진 데이터들을 골라주는 컴퍼넌트가 있다.[15] 이것은 주어진 데이터 리스트를 우리가 원하는 design 목적에 맞게 바꿀 때 사용된다. 아래 예시를 살펴보자.



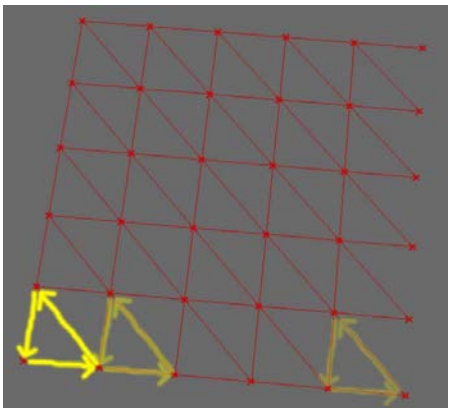
점으로 이루어진 data list가 있다. 이 중 가장 작은 x 좌표값을 가진 점을 골라내려고 한다. 위에서 언급한 것 처럼 <point decompose>를 이용하면 점의 좌표값을 알 수 있다. 이렇게 얻어진 모든 x값 중에서 가장 작은 x값을 알기 위해서 <sort list>를 이용할 수 있다. <sort list>는 어떤 data list를 각 데이터가 가진 특정 수치 값을 기준으로 오름차순으로 바꾸어 준다. 위 경우 점이 무작위로 <point> 에 의해서 정의되어 있으며 이 것을 <sort list>의 A에, 그리고 이 점들의 X 좌표를 K에 넣는다. 이 때 <sort list>가 돌려주는 값은 x 좌표값인 '수' 가 오름차순으로 정리되어 L로 나오게 되며 A로는 이 '점' 자체의 data list가 x좌표 값이 작은 점부터 오름차순으로 나타내어 준다. 이것에 <list item>을 이용하여 index number가 0인 값을 찾으면 위와 같은 점이 나온다.

[15] 역자 주: <list item>의 경우 지정한 index number를 가진 데이터를 골라주며 <sub list>의 경우 list에 domain 값을 주어 list index의 범위를 지정할 수 있다.

`

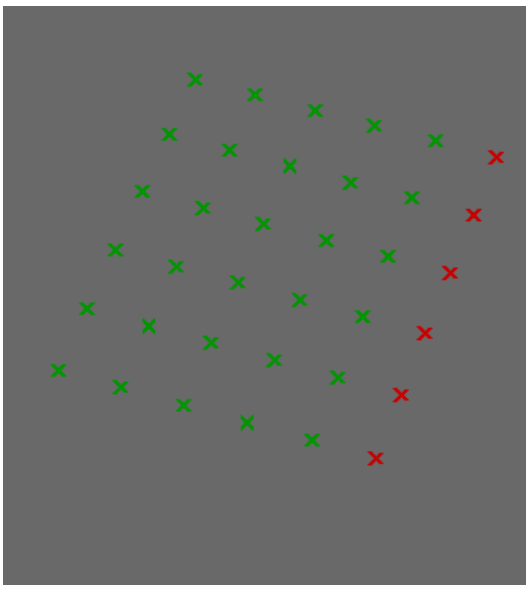
삼각형 패턴 그리기 (Triangles)[16]

Data management 를 이용하여 실험을 계속 해보자. 복수의 점들이 grid의 형태로 놓여있다고 보자. 이것에 그림 3.19와 같은 삼각형을 그려보자. 이러한 개념은 mesh를 만들거나 surface를 panel화 할 때 매우 유용하다. 이것의 기본적인 개념을 알아보도록 하자.



포인트 grid상에 삼각형 패턴 그리기

먼저 점의 grid를 만들기 위해서는 <series>와 <point>가 필요하다. 그 다음으로는 이 점들 중에 필요한 점을 골라 선을 그리는 것이 중요하다. 즉 각 점에서 line이 시작하면 같은 행 바로 옆에 있는 점에 연결되어야 한다. 이 점의 같은 열에 있는 바로 위의 점을 제외하고 그 위의 점과 같은 행의 왼쪽 점과 연결한 뒤 처음 시작한 점으로 돌아와야 한다. 이를 위해서는 삼각형의 꼭지점을 각각의 data list로 만들면 된다. 즉 시작점의 data list와 같은 행 바로 옆 점들로 이루어진 data list, 그리고 시작점의 같은 열 바로 위의 점으로 이루어진 data list를 연결하여 삼각형을 그리는 것이다.

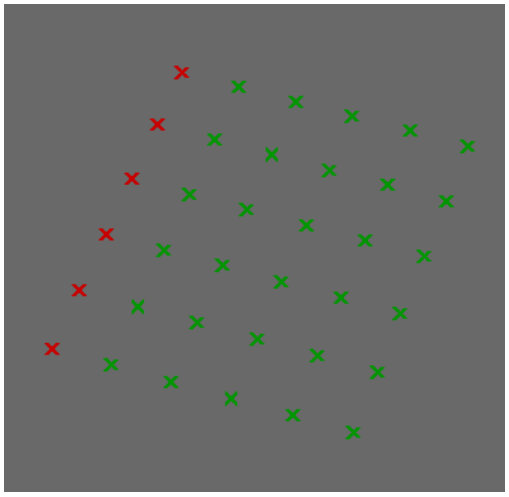


<series>와 <pt>에 의하여 그려진 grid 상의 점들 중 에서 삼각형의 첫 번째 점으로 사용 될 점의 리스트는 위에서 초록색으로 선택된 점과 같다. 이 점들은 원래 grid 상의 점 중들 중에서

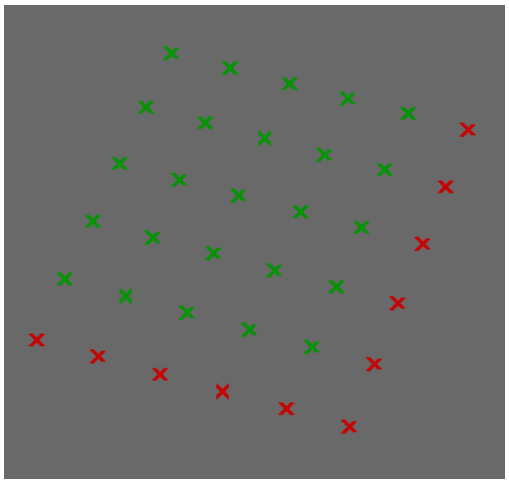
[16] 역자 주: 해당 챕터는 저자가 기술한 내용을 그대로 번역하기 보다는 내용을 좀 더 풀어서 설명하였다.

`

가장 오른쪽 열 위에 있는 점을 제외한 점들이다. 처음 grid 상의 점들이 data list 내에서는 원점의 index number를 0으로 하고 그 오른쪽 점의 index number가 1이 된다. 데이터 리스트의 방향은 일반적으로 U 방향으로 가기 때문이다. 즉, grid의 행의 개수가 되는 <series>의 C 값의 배수에 해당하는 순서에 있는 점들을 제외해주면 된다. 이 경우 <series>의 C에 6이 들어가 있디 때문에 <pt>에 36개의 점이 생겼다. 이 C에 들어간 6을 <cull Nth>의 N에 연결하고 위 grid <pt>를 <cull Nth>의 N에 연결하면 6번째, 12번째, 18번째, 24번째, 30번째, 36번째 점을 제외한 점들의 data list를 얻을 수 있다.

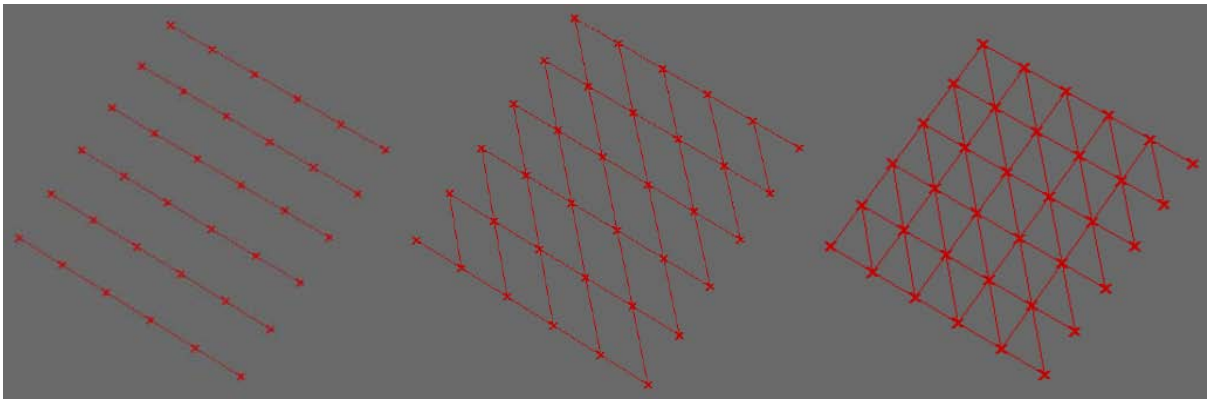


삼각형을 그리기 위한 그 다음의 점의 리스트는 위와 같다. 위와 같은 점의 list는 <series>와 <pt>에 의해서 생긴 점의 data list를 <shift>를 이용하여 U 방향으로 한 칸 밀어낸 뒤 위에서 사용한 <cull Nth>의 N에 똑 같이 6을 넣어주면 된다. 그러면 기존 grid 점의 data list가 가장 첫 번째 점(rhinoceros 상의 원점)에 있는 점을 제외하고 시작하면서 그 이후 6의 배수의 순서에 있는 점들을 제외해주기 때문에 위와 같은 점의 리스트를 얻을 수 있다.

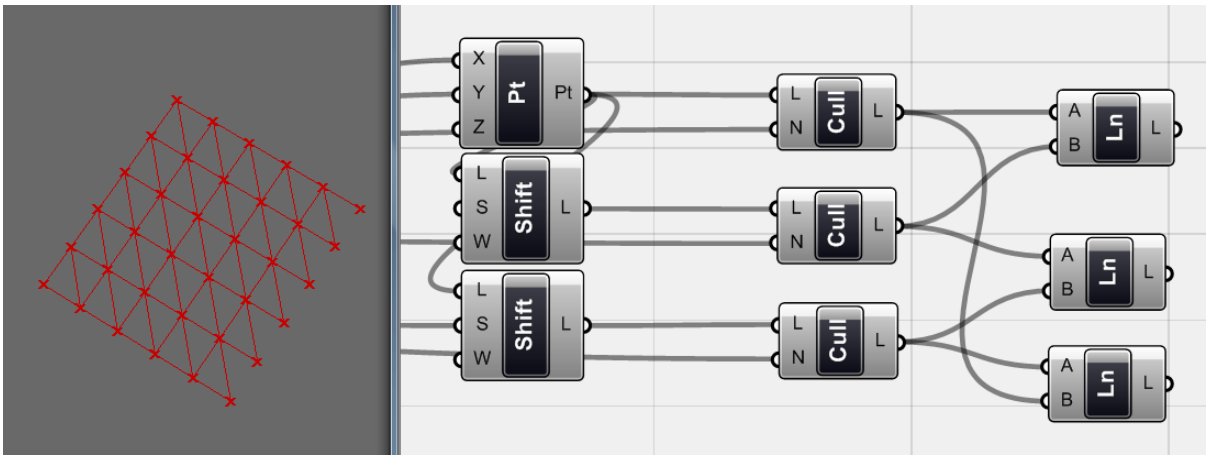


삼각형의 세번째 꼭지점에 해당하는 점은 위와 같다. 위는 grid 점의 list를 <shift>를 이용하여 6만큼 민 뒤에 (그러면 첫번째 열의 점이 제외된다.) 다시 이 data list 상에서 6의 배수에 해당하는 순서의 점들을 <cull Nth>를 이용하여 제외시켜주면 된다.

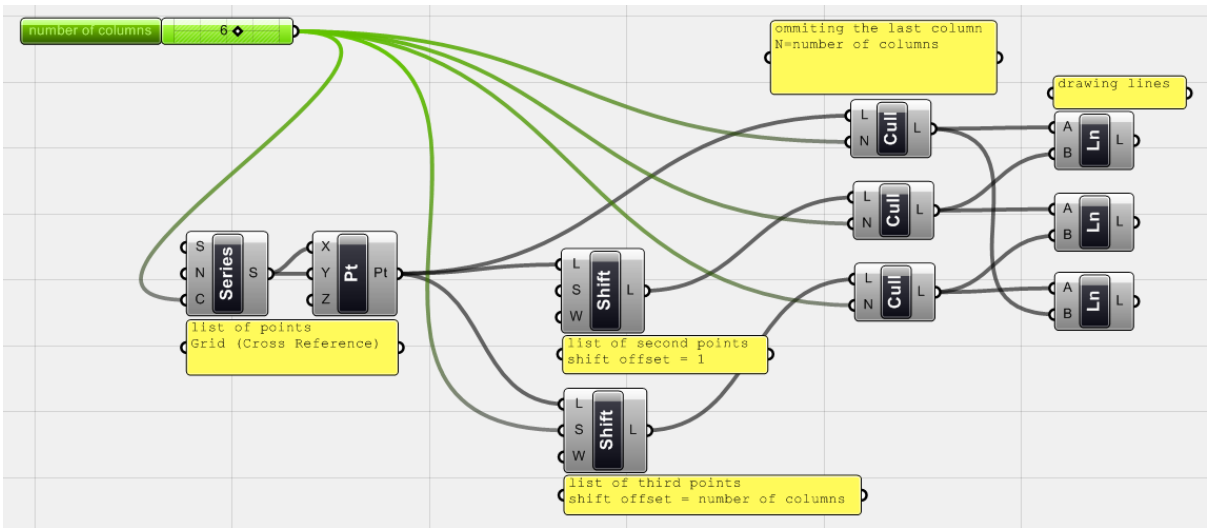
`



이제 각 점들을 양 끝점 A B를 인풋으로 하는 <line>으로 연결시켜주면 위와 같이 삼각형을 만들 수 있다.



<cull Nth>가 적용된 점들을 <line>을 이용하여 연결하였다.



<number slider>의 수를 조절하면 grid의 변화와 함께 삼각형 패턴이 자동적으로 적용되는 것을 볼 수 있다.

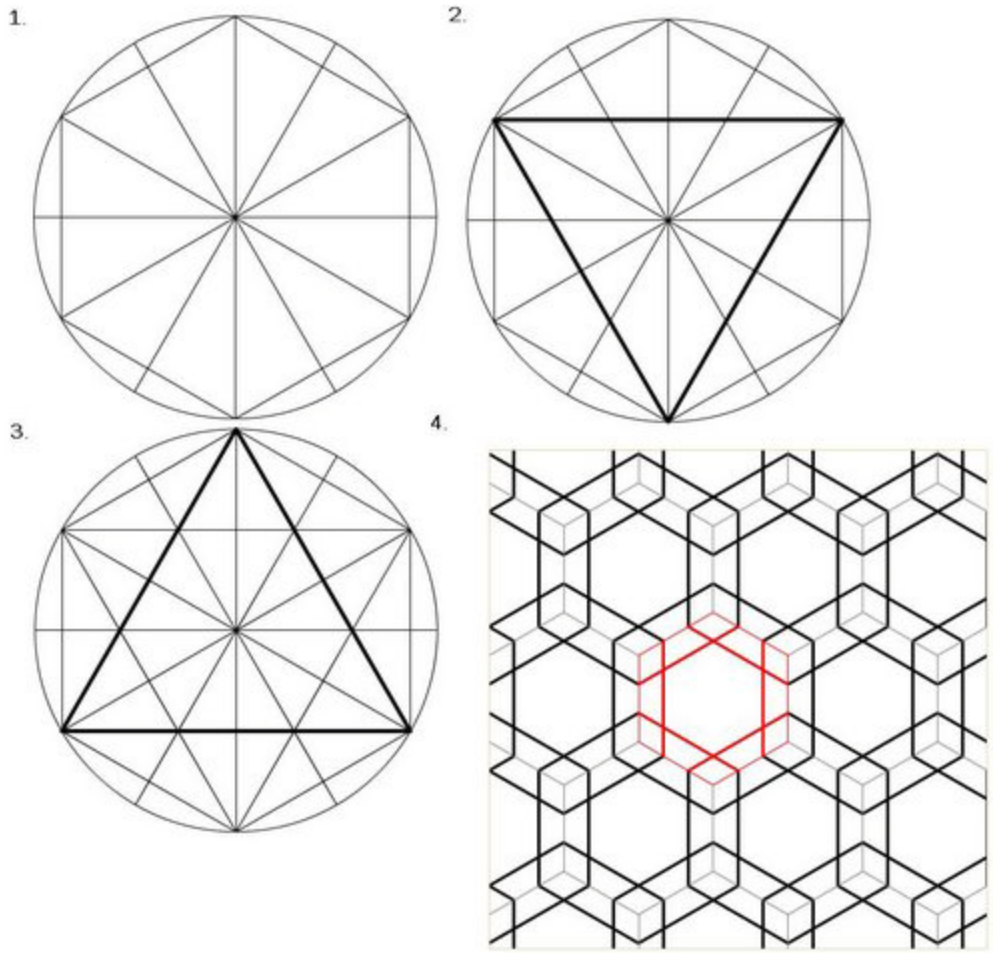
면을 mesh로 만드는 개념에 대해서는 이 후에 좀 더 살펴보도록 하겠다. 위 grasshopper definition의 경우 grid의 제일 위에 있는 열에도 line이 생기기 때문에 완변하가도 할 수는 없다. 하지만 data가 어떻게 생성되고 관리되어야 하는지는 명확하게 전달하고 있다..

`

이것에 대한 이해를 바탕으로 다음 예제를 살펴보도록 하자.

3_8_평면 상의 기하학적 패턴 (On Planar Geometrical Patterns)

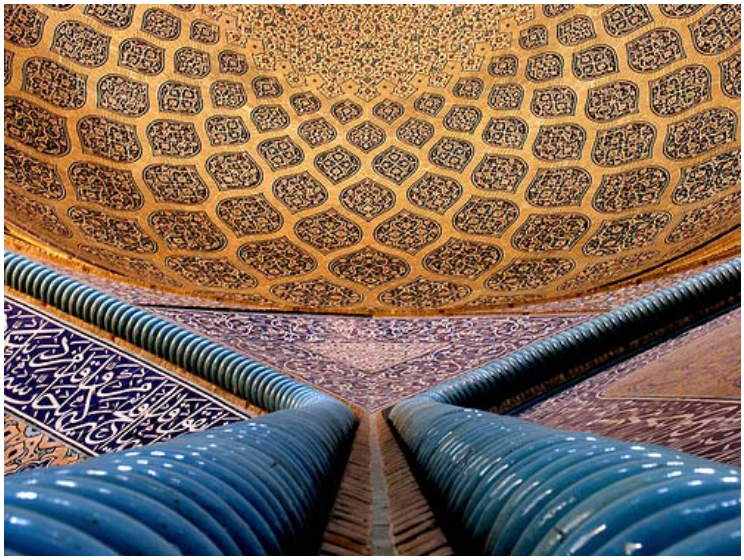
기하학적 패턴을 이용한 design은 Generative algorithm에서 다루고자 하는 디자인 이슈 중 하나로 grasshopper를 통하여 쉽게 구현할 수 있다. 또한 기하학적 패턴은 여러 분야의 design에 적용될 수 있다. 이를 위해서는 패턴을 보았을 때 패턴 속에 존재하는 수학적인 공리를 찾아내는 것이 중요하다. 이를 통해 기본적인 형상을 그리고 이것을 반복함으로써 패턴을 만들어낼 수 있다.



간단한 도형을 이용한 기하학적 패턴 만들기

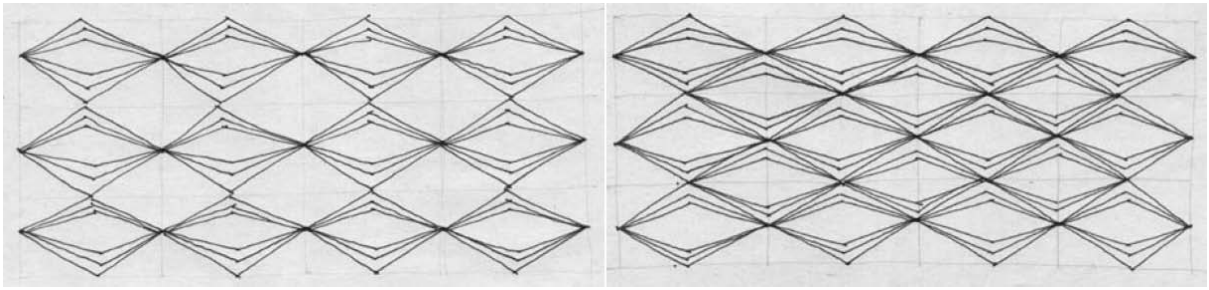
간단한 함수들과 data set, 그리고 수치 데이터를 이용하여 생성할 수 있는 형상과 고전적인 기하학 패턴을 살표보도록 하자.

`



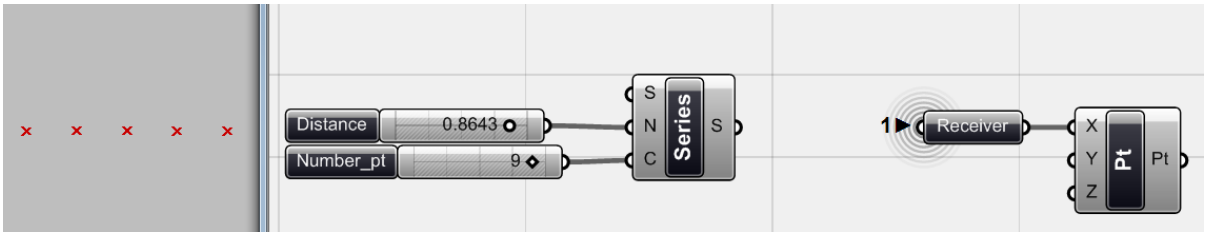
간단한 수학적 기하학적 계산을 통하여 구현된 복잡한 기하학적 형상, *Iran's Sheikh Lotfollah Mosque's*

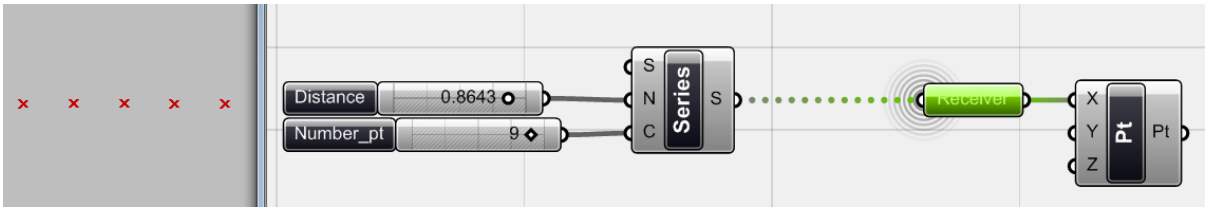
Simple Linear Pattern



패턴 생성을 위한 기본 개념

먼저 패턴에 사용될 점들을 그려본 뒤 그것을 지나는 선을 그려보도록 하자. <series>를 이용하면 생성되는 값의 개수 (이 경우 점의 개수)와 그것의 사이 값(step size; 이 경우 점들 사이의 간격)을 조절할 수 있다. <series>를 이용하여 <pt>의 X 좌표를 지정해주자. (Y와 Z는 0으로 유지한다.)

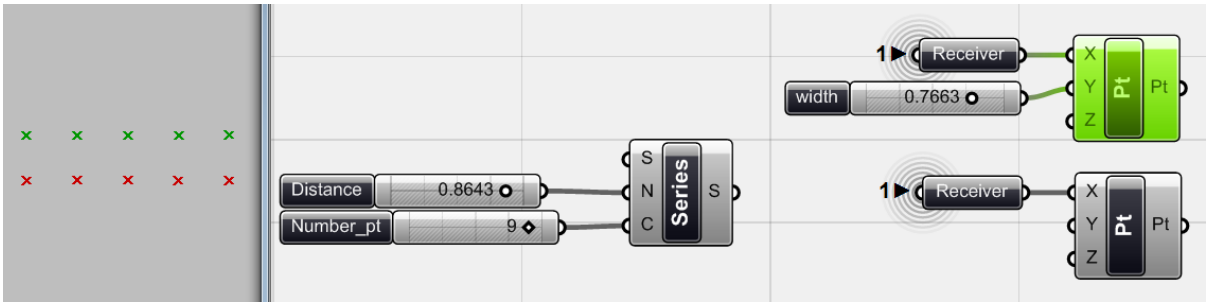




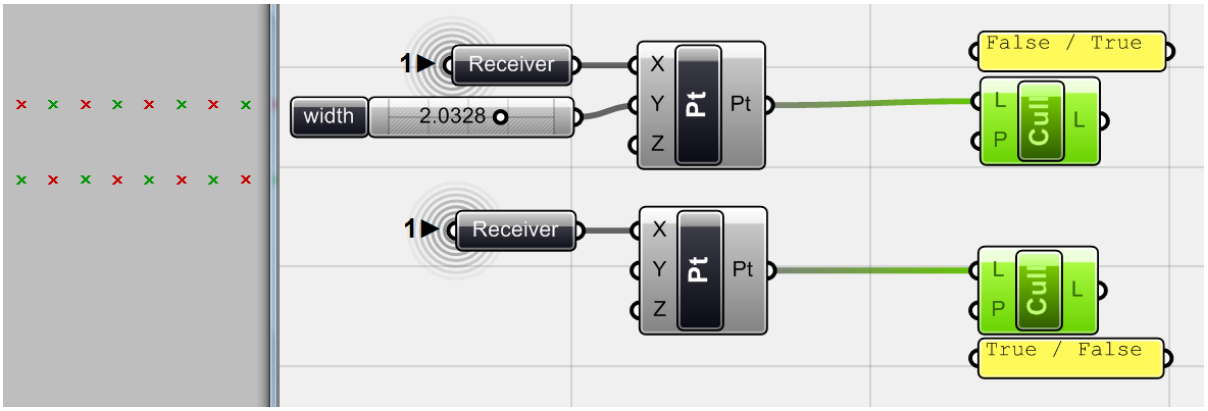
위의 Grasshopper definition은 <series>와 <pt> 를 이용하여 필요한 점을 한 행 생성해본 것이다. 위의 경우 <receiver> (Params > Special > Receiver)를 이용하여 컴퍼넌트들 사이의 선(wire)

`

을 생략하였다. 좀 더 복잡한 definition을 만들게 될 경우 선이 너무 많아져 로직이 복잡해 보이므로 이것을 시각적으로 정리해주는 것이 좋다. 위의 경우 <pt>는 <receiver>를 이용하여 <series>에서 생성되는 수치 data list를 받아 이것을 점의 x좌표로 활용하고 있다.



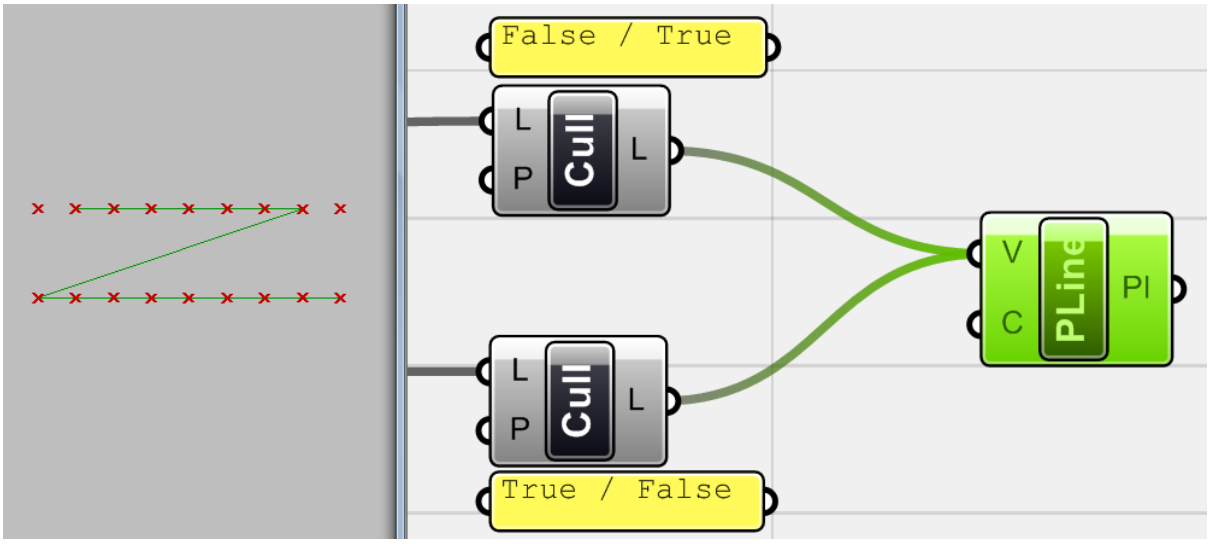
지그재그로 가는 선을 만들기 위해서는 두 행의 점이 필요하다. 이를 만들기 위하여 <pt>를 복사, 붙여넣기 한 뒤 <number slider>를 이용하여 Y값을 준다. 그러면 두 번째 점의 행이 생기게 된다.



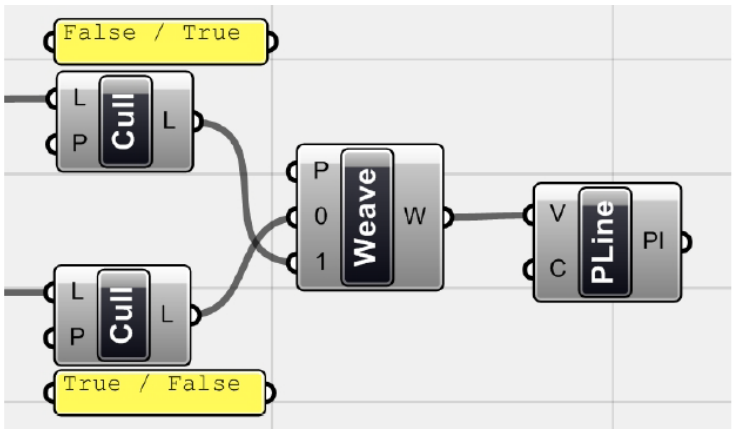
이제 점을 번갈아 가면서 선택하기 위해서는 <cull pattern>의 P에 True or False 값을 가지는 Boolean data를 넣을 수 있다. [17]

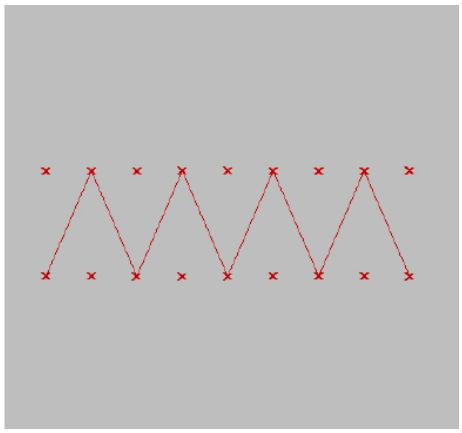
[17] 역자 주: 위 경우 위 <cull pattern>의 P에 false/true가 적용 된 것이고 아래의 <cull pattern>에는 true/false가 적용 된 것이다. <cull pattern>의 P를 우클릭 한 뒤 management Boolean collection을 선택하면 원하는 pattern을 적용시켜줄 수 있다. 이 때 true는 1, false는 0으로 써주면 true와 false로 인식이 된다.

`



두 <cull pattern>을 <polyline>(curve > spline) 에 연결하면 위와 같이 Z자의 형태로 선이 그려지게 된다. 그 이유는 아직 점의 리스트가 첫 번째 줄의 첫 번째, 두 번째 둘의 두 번째, 첫 번째 줄의 세 번째, 두 번째 줄의 네 번째....와 같은 순서로 정리되어있지 않기 때문이다.[18]





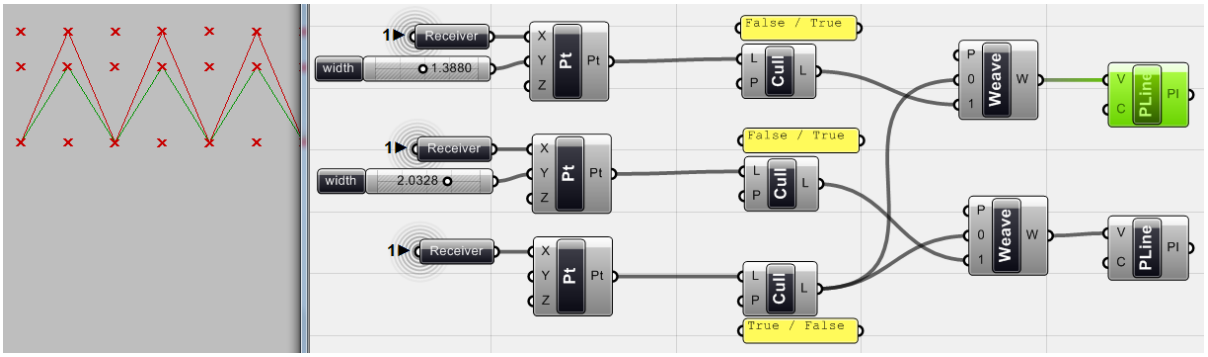
점의 순서를 이렇게 바꿔주기 위해서 사용할 수 있는 component가 바로 <weaver> (logic > list) 이다. 이것은 여러 개의 data list를 P에 정의된 패턴을 적용하여 하나의 data list로 만들어주는 것이다.[19] 이제 이 것에 <polyline>을 적용시키면 지그재그 형태의 선이 생성된다.

---

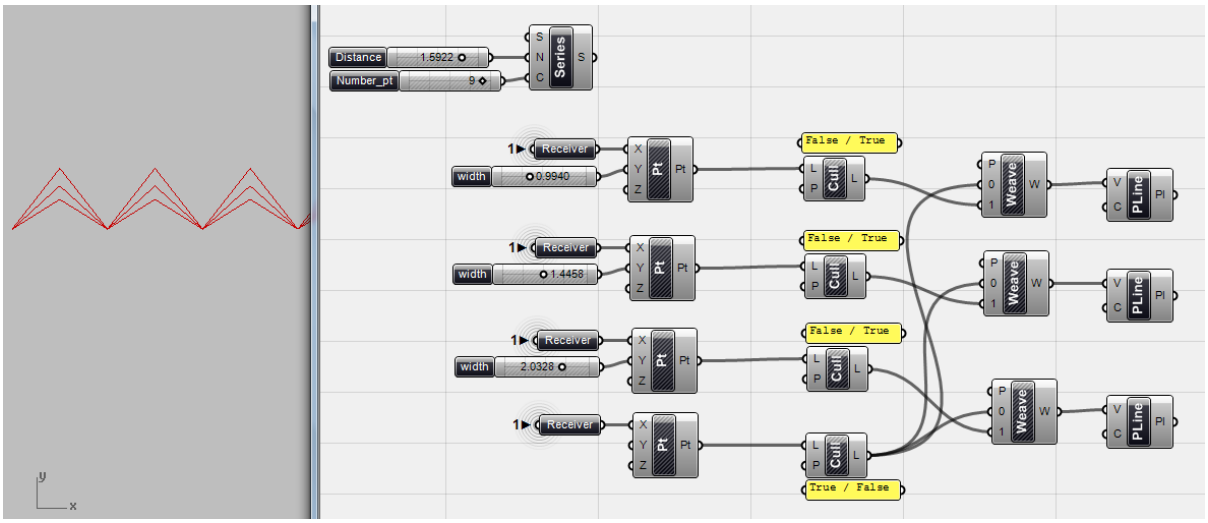
[18] 역자 주: 위 경우 <polyline>에 들어오는 data list를 살펴보면 윗줄의 <cull pattern> 에서 나온 네 개의 점이 0,1,2,3 이고 아랫줄의 <cull pattern>에서 나온 점이 4,5,6,7,8번째 점이 되기 때문이다. 만약 아래 <cull pattern>을 <polylline>과 먼저 연결할 경우 선이 Z가 아닌 Z가 mirror된 것과 같은 모습으로 그려지게 된다.

[19] 역자 주: P에 적용된 기본 값은 0, 1 이다. 이것은 Boolean data가 아니라 0으로 input 되는 data list의 중 첫 번째 값을 받고 1로 input 되는 data list 의 첫 번째 값을 하나를 받은 뒤 0 data list의 두 번째 값, 1 data list의 두 번째 값, 0 data list의 세 번째 값, 1 data list의 세 번째 값,...의 순서로 data list를 만들어 달라는 것이다. 이 P 값에 적용되는 값을 0,0,1 로 바꾸면 0번 list 의 첫 번째 두 번째, 1번 list의 첫 번째, 0번 list의 세 번째 네 번째, 1번 list의 두 번째, 의 순서대로 data를 정리해준다.

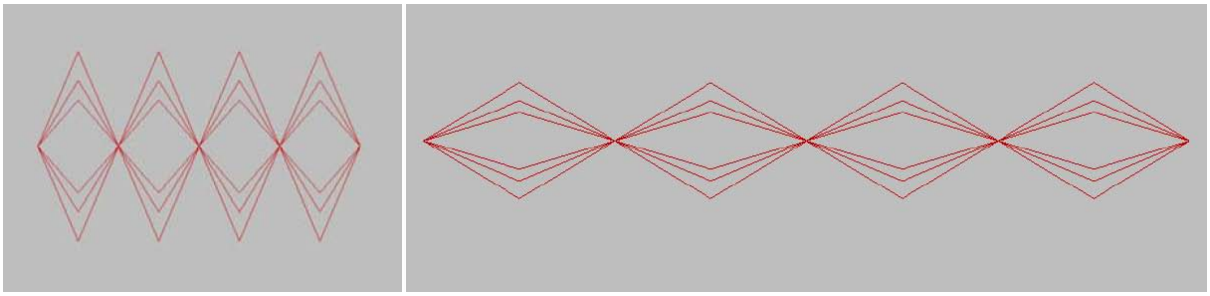
`



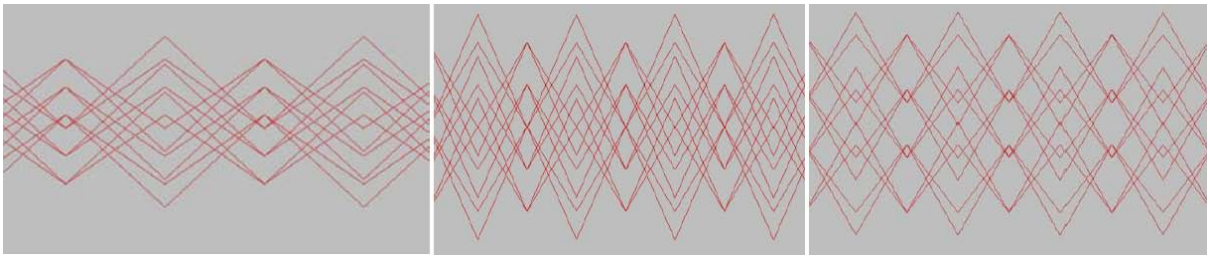
같은 방식으로 각각 다른 Y값을 가지는 점들의 행을 만들어 준 뒤 또 다른 <weave>와 <polyline>을 이용하여 두 번째 지그재그 모양의 선을 생성할 수 있다.



같은 algorithm을 이용하여 세 번째 선을 생성해준다. <Pt>, <Cull>, <Weave>의 context menu를 이용하여 preview를 끄면 polyline만 남게 된다.



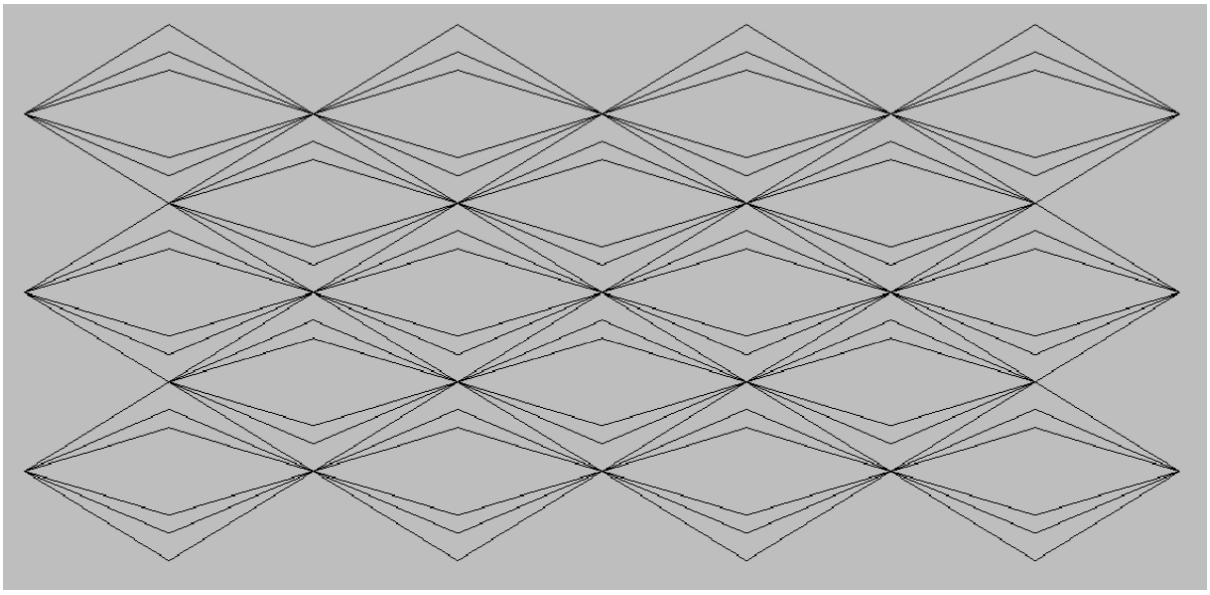
이때까지 생성한 definition을 모두 복사하고 붙여 넣은 뒤 <pt>의 Y 값을 음수로 주면 (기존 y에 들어가는 <number slider>의 값을 <f1 > f(x)=-x 에 연결하면 된다.) 대칭된 polyline을 얻을 수 있다. 거리를 조절해보면 패턴을 다른 형태와 스케일로 만들어낼 수 있다.



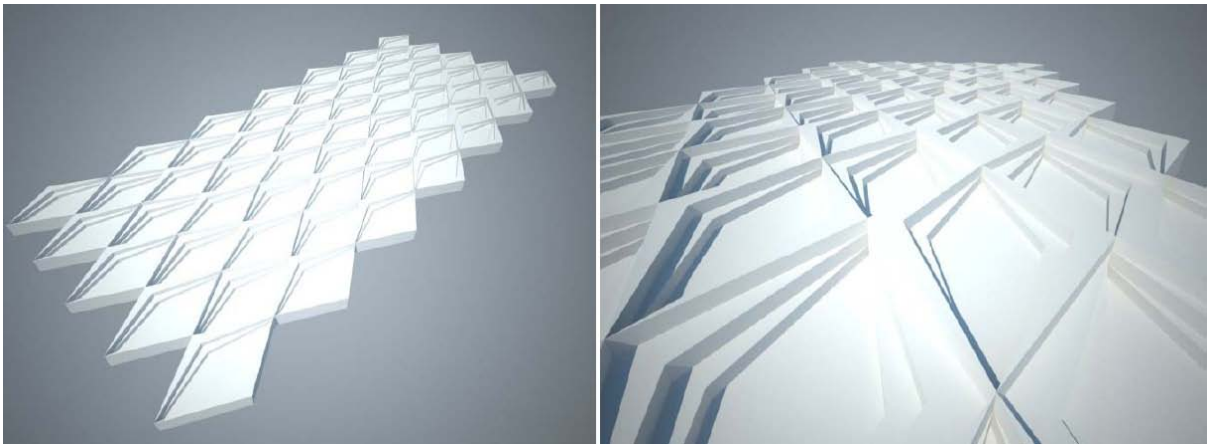
만약 <cull>이나 data list를 바꿔 점을 생성하는 방식을 바꾸면 그 결과는 무척 달라진다. 이것을

`

이용하여 좀 더 복잡성이 높은 패턴을 얻을 수 있다.



이것은 첫 번째 디자인의 결과물이다. 그 기본 개념이 계속적으로 반복되는 것이다. 이것을 사용자가 원하는 목적에 따라 이용할 수 있다.

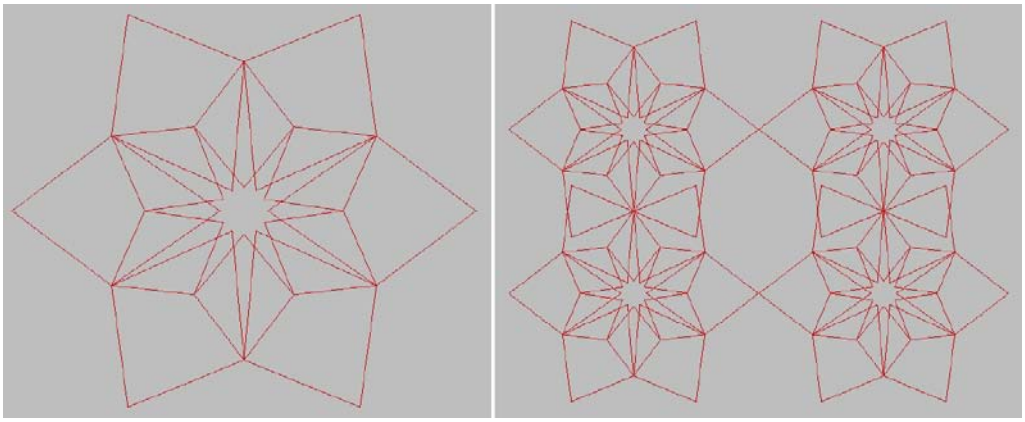


위 예시는 우리가 그렸던 패턴을 이용하여 디자인을 할 수 있는 수백 가지 가능성 중 하나이다. 이 후 이것의 기본이 된 패턴을 조작하여 다른 디자인 결과물을 얻는 것을 연습해볼 것이다.

Rotate 된 형태의 패턴 (Circular patterns)

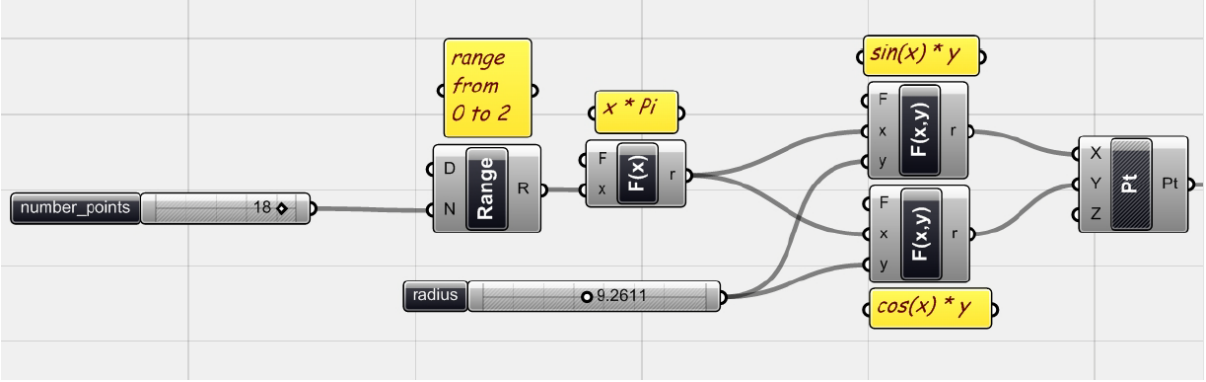
모델링 방법과 관련되어 만들어낼 수 있는 패턴의 가능성은 끝이 없다고 할 수 있다. 아래 fig 3.38은 특정 기하체가 rotate된 형상으로 존재하고 있다. 같은 논리가 적용된 복수개의 커브가 있으므로, 그 논리의 일부를 소개하도록 하겠다. 나머지는 독자가 직접 찾아보기를 바란다.

`



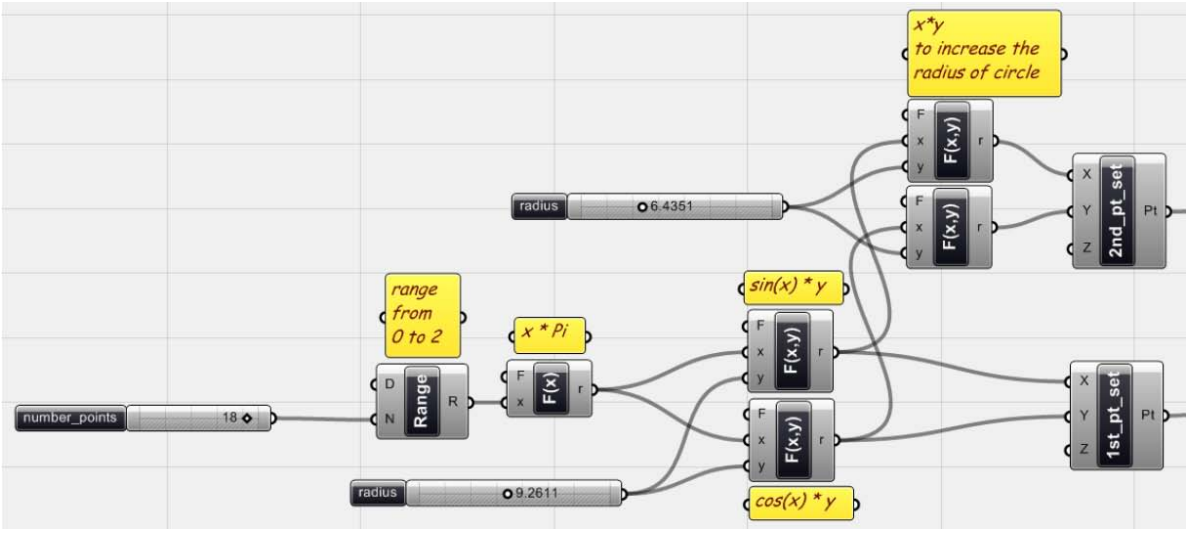
원형으로 된 기하학적 패턴

기본이 되는 점들은 사실 하나의 원 위에 존재하는 여러 개의 점들이다. 이는 앞서 다룬 점들의 내용과 크게 다르지 않다. 이 원을 여러 축적에 맞춰 생성하고 이 점들을 솎아낸(cull) 뒤 그 점들을 지그재그로 연결하였을 뿐이다. 이것의 결과물로 별 모양의 형상이 그려지게 된 것이다. 이러한 별 모양의 형상을 여러 가지로 만들고 이것을 겹치면 위와 같은 형상이 나오게 된다.

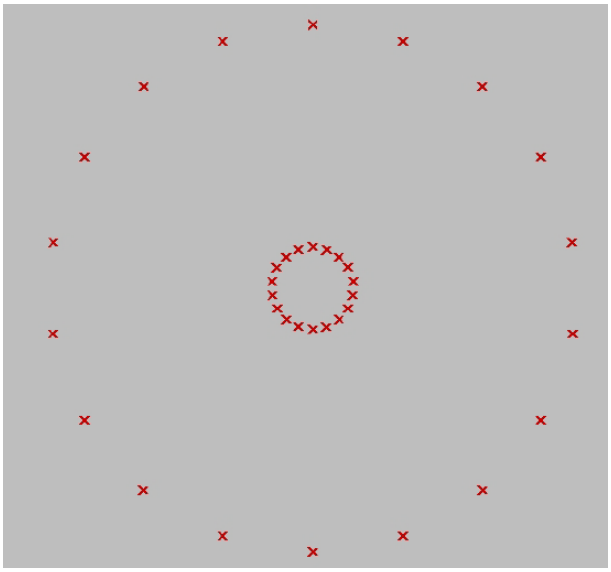


Sin 함수와 Cos 함수에 0에서 2pi 까지의 범위를 줌으로서 기본이 되는 원을 만든다. 두 <f2>에는 원의 기본 형상을 만들기 위한 Sin/Cos 함수와 이 원의 반지름을 결정하는 <number slide>를 연결하게 된다.

`



<number slider>를 이용하여 Sin/Cos 함수가 생성하는 원에 더 큰 반지름을 두 번째로 사용될 원을 그릴 수 있다. 그 결과는 두개의 점들의 data list로 나타나게 된다. 위 그림의 경우 <pt>의 이름을 1st_pt_set 과 2nd_pt_set 으로 바꿨다.



첫 번째 그리고 두 번째 원 위의 점들

이 점들을 솎아내기 위해 지난 예시와 같이 <cull pattern>와 <weave>를 적용하면 두 point list로 부터 각각 하나씩의 점을 순서대로 가지는 리스트를 만들어낼 수 있다. 하지만 이 예시에서는 algorithm의 <sort list>를 사용하였는데 이것이 최종 결과물을 만들어내는데 더욱 유용하기 때문이다. 이 점들은 index number를 기준으로 다시 배열<sorting>이 된다.

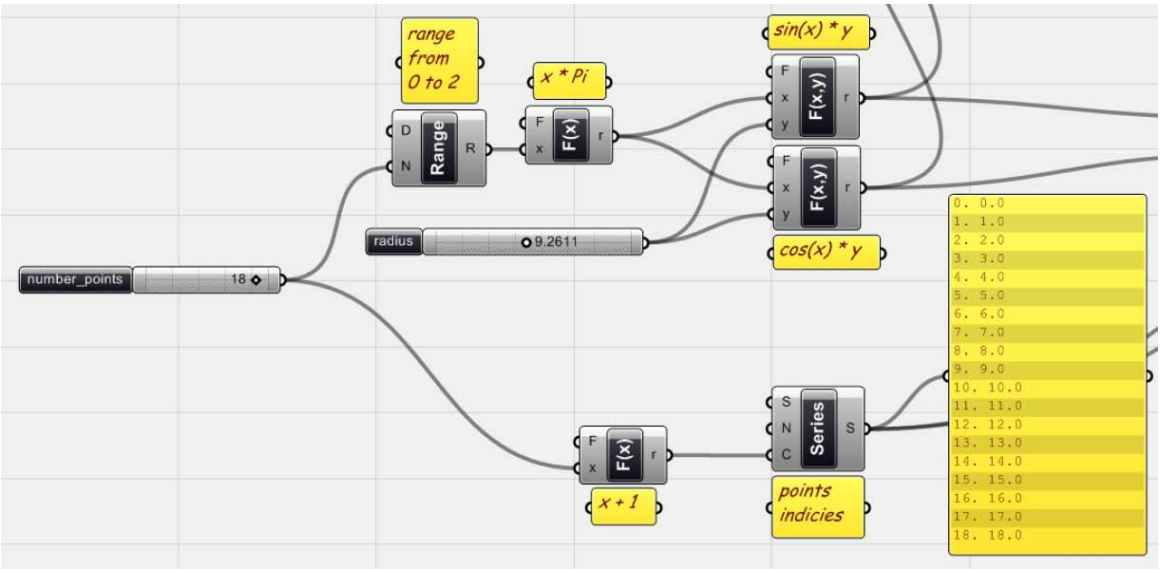
먼저 index number를 생성할 필요가 있다. 이를 위하여 실수가 사용되는 <range> 컴퍼넌트를 만들고 <series> 이용하여 정수의 list를 생성, 점들의 <index number>로 사용한다.

<range>의 매개변수 N은 또 다른 매개변수 D(domain;정의역)에 의해 정의되는 수의 범위를 몇 단계로 나눌지를 결정한다.[20] 이 <range> 내보내는 list의 길이 (즉 list 내 data의 개수)는 N+1과

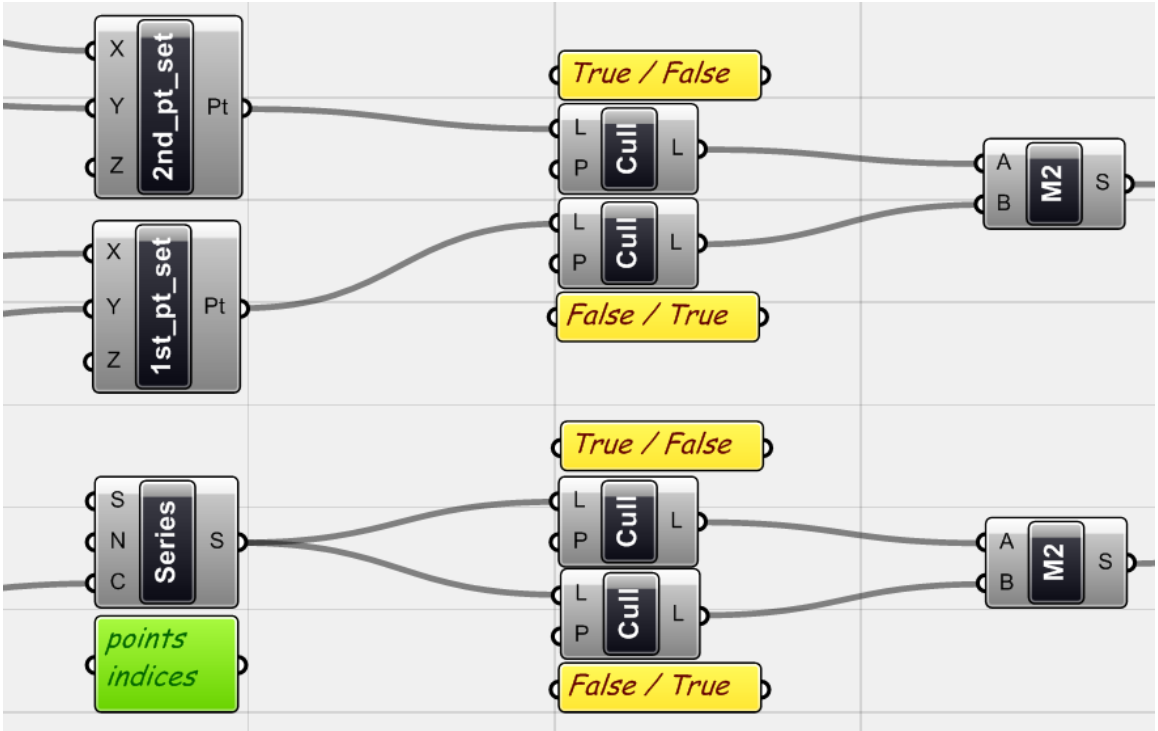
[20] 역자 주: 이 경우 <domain>을 가져온 뒤 A에 0, B에 0을 넣으면 algorithm을 이해하는데 시각

`

같다. 이 N+1에 해당하는 수를 <series>의 C 값에 연결해주면 point list의 index number에 해당하는 값들을 얻을 수 있다.



점들에 index number의 list(0부터 시작하는 정수의 list)를 생성하였다.

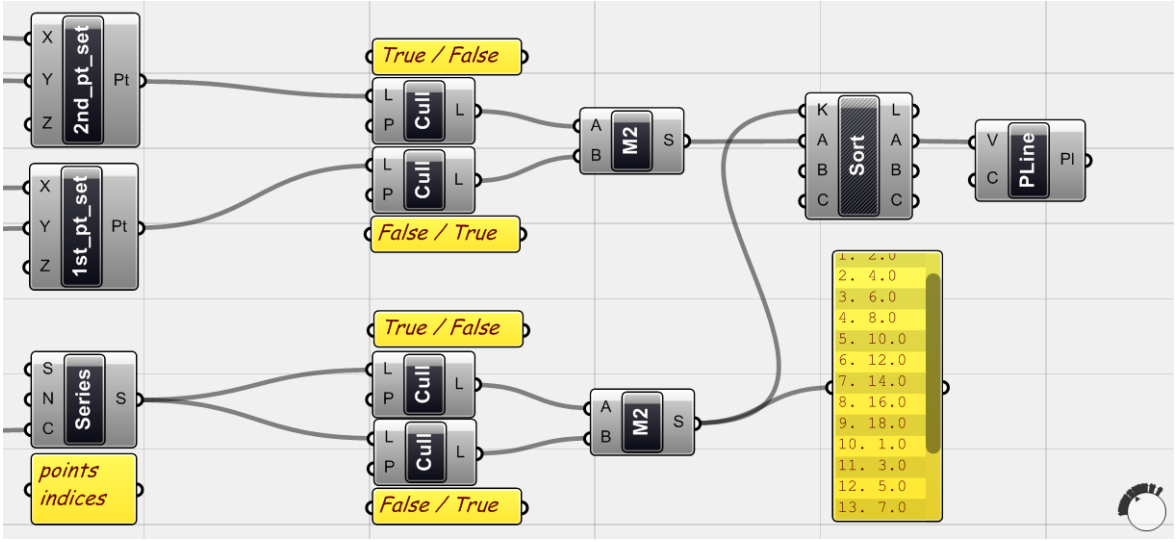


이제 점과 점들의 index number 들을 이 전 예시들처럼 솎아내어야 한다. 이 후 <Merge>(Logic > Tree)를 이용하여 <series>와 두 대개의 <point>에서 <cull>을 통과하여 나오는 data list를 각각 하나로 합쳐준다. 이 <merge>를 통해 나오는 <series>는 같은 수들을 data로 가지고

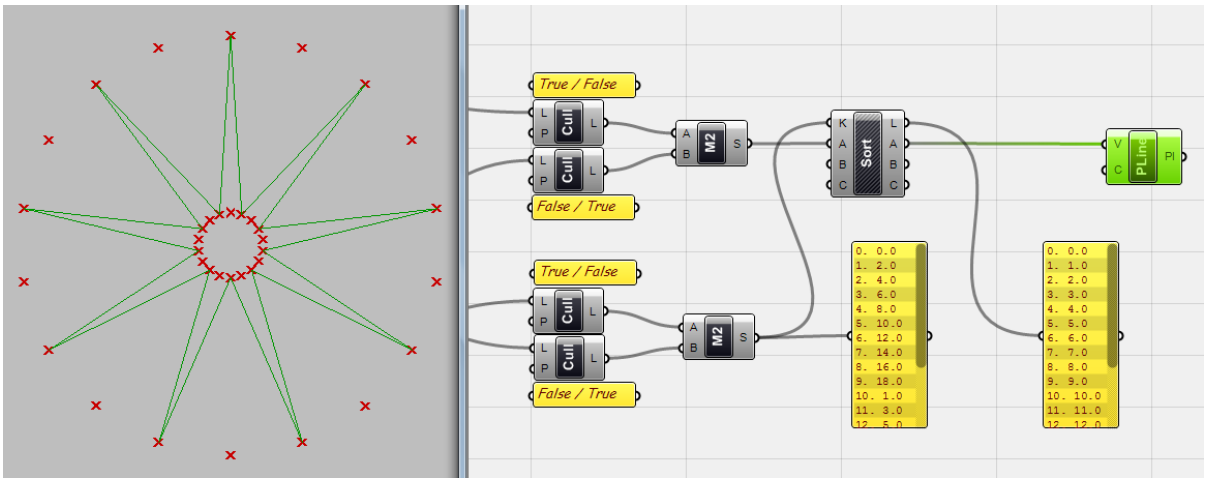
적으로 더욱 도움이 된다. 혹은 D를 우클릭 한 뒤 <set domain>을 선택하여 0 to 2.0 을 입력해 줄 수 도 있다.

`

있지만 그 순서가 달라지게 된다.[21] 이 data list를 key로 하여 <sort>를 통해 점들을 우리가 원하는 순서로 data list 내에서 재배열 해줄 수 있다.



점들의 index number가 아래 <merge>에서 나온 수의 순서와 일치하게 된다. 이제 점들을 <polyline>에 넣어주면 아래와 같은 그림이 그려지게 된다.



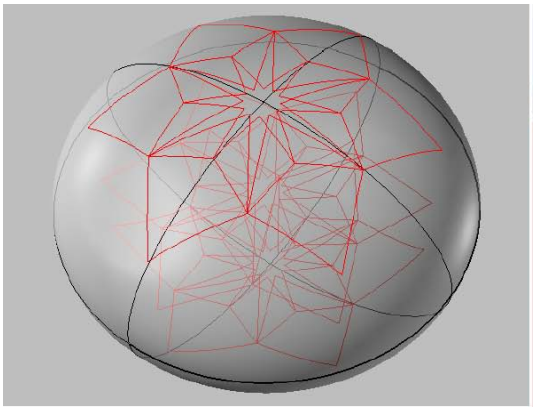
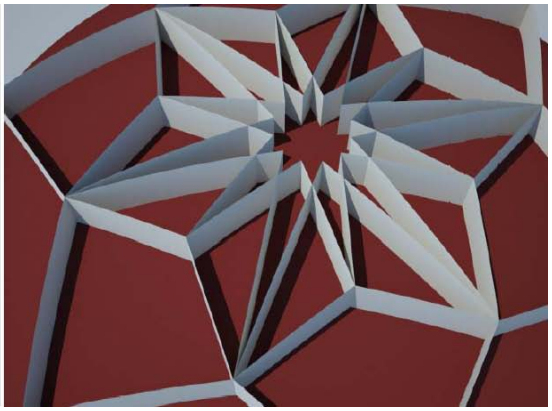
key에 사용된 datalist는 <sorting>의 전과 후에 위와 같다.

같은 logic을 이용하면 더욱 더 복잡한 geometry를 생성할 수 있다. 이 점들을 다시 어떻게 <cull>하고 연결하냐에 따라 수많은 결과물을 얻을 수 있을 것이다.

---

[21] 역자 주: <series>는 < 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 18 >에 해당하는 data list를 내보내게 되고 이것은 위 true/false의 <cull>을 통하여 < 0, 2, 4, 6, 8, 10, 12, 14, 16, 18 >로 아래 false/true 의 <cull>을 통하여 < 1, 3, 5, 7, 9, 11, 13, 15, 17 > 이 된다. 이 둘이 <merge>를 통하여 하나의 data list로 합쳐지면서 < 0, 2, 4, 6, 8, 10, 12, 14, 16, 18 , 1, 3, 5, 7, 9, 11, 13, 15, 17 > 의 data list가 된다.

`

위와 같은 patten을 project 하여 얻을 수 있는 결과물은 위와 같다.

이번 장에서는 간단한 수학 함수들을 활용하여 data list를 조작하는 방법에 대해서 공부하였다. 이러한 것을 잘 활용하면 같은 결과를 얻는데 필요한 과정을 훨씬 짧게 만들 수 있다는 것을 명심하자.

`

Chapter_4_변환 Transformations

`

Chapter_4_변환 Transformations

변환(transformation) 은 3d 모델링에서 가장 중요한 연산 중 하나이다. 이를 이용하면 초기의 간단한 객체로부터 다양한 결과를 얻을 수 있다. 이 변환을 통하여 객체의 스케일을 바꾸고 방향을 바꾸거나 움직이고, 복사하고, 대칭을 시킬 수 있다. 혹은 객체를 집합시킬 수도 있다. 변환에는 여러 종류가 있다. 이를 '선형 변환(linear transformation)'과 '나선형 변환(spiral transformation)'으로 분류할 수 있다. 선형변환은 2d에, 나선형변환은 3d 공간좌표에 적용이 된다.

혹은 변환을 이에 사용되는 객체의 상태 변화에 따라 분류할 수 있다. 회전(rotation), 투영(reflection) 등은 초기 객체를 그대로 유지한다. 하지만 스케일 변환이나 전단 변환(shear transformation)은 초기 객체를 변화시킨다. 또한 비선형적 변환(non-linear transformation)도 존재한다. 변환 뿐만 아니라 회전과 투영에도 다른 여러 종류가 있으며 3d 공간 좌표상에서 스케일 변환에 복수개의 상수가 사용되는 변환 (non-uniform scale transformation) 도 있다. 또한 나선형의 변환을 이용하더라도 여러 변형을 만들어낼 수 있다.

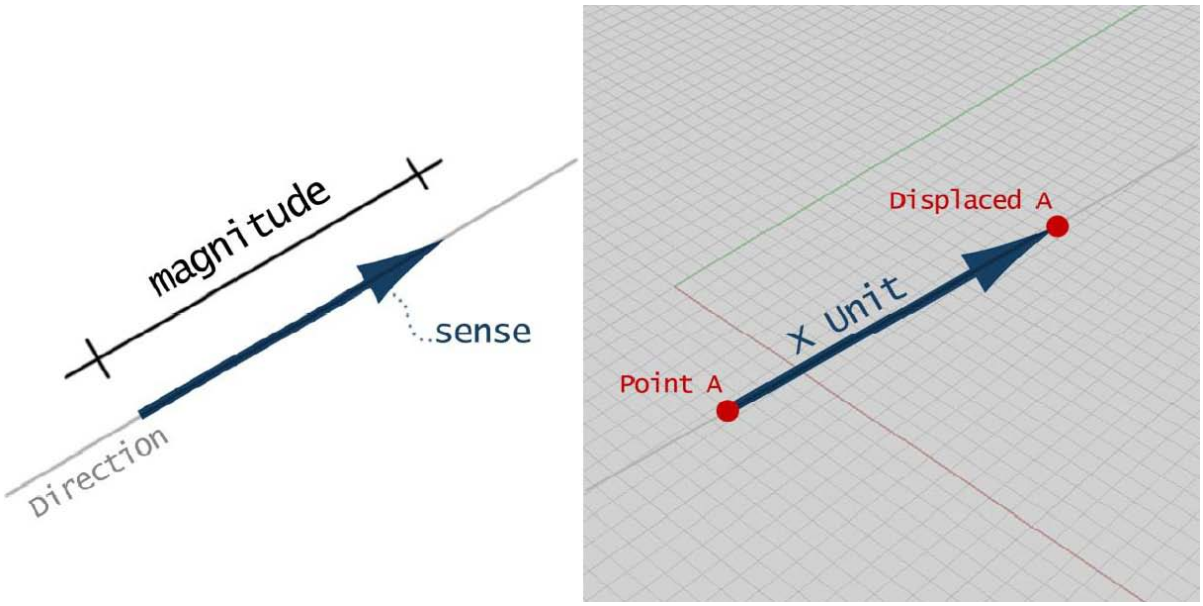
객체를 변환시키기 위해서는 기본적으로 객체 자체나 그 일부를 이동시켜야 한다. 이것을 하기 위해 가장 기본적으로 필요한 기하학적 요소가 바로 평면(plane)과 벡터(vector)이다. 이 장에서는 바로 이 평면과 벡터를 이용하여 디자인을 할 수 있도록 하겠다.

변환은 간단한 객체들을 이용하여 복잡한 형태를 만들어 내는데 무척 유용하다. 자연에 존재하는 생물 중에는 이러한 변환의 개념으로 설명할 수 있는 것들이 무척 많다

`

4_1_벡터와 평면 Vectors and Planes

벡터란 수학적 기하학적 객체로 일정한 크기(magnitude) 혹은 길이(length)와 방향[22], 그리고 을 가진다. 벡터는 하나의 점에서 시작하여 다른 하나의 점으로 향하며, 이 두 점 사이의 길이가 그 크기가 되고 시작점과 끝점에 의해 그 방향이 결정된다. 이러한 벡터는 기하학과 변환에서 폭넓게 사용된다.



점 A와 이것이 벡터에 의하여 이동된 점 B

하나의 점과 하나의 벡터가 있다고 가정해보자. 이 벡터를 이용하여 그 점을 이동시키면 그 위치는 벡터의 세기(magnitude)와 그것의 방향(direction)에 영향을 받게 된다. 이러한 기본 개념을 이용하면 기하체를 이동, 스케일 변환, 위치 위동(orientate) 시킬 수 있다.

평면(plane)또한 유용한 기하학적 요소이다. 평면은 무한한 평평한 면으로 원점을 가지고 있다. Rhinoceros의 construction plane 또한 마찬가지 이다. 우리는 이러한 평면 위에 기하체를 놓고 평면의 방향과 원점을 기준으로 하여 변환을 적용하게 된다. 이러한 평면을 만들기 위해서는 적어도 두 개의 벡터가 필요하며[23] 이 평면을 기준으로 하여 기하체에 변환을 적용할 수 있다.

벡터는 방향(direction)과 강도(magnitude)를 가지고 평면은 원점과 좌표축(orientation)을 가진다. 이 둘을 이용하여 모델을 생성하고 수정하고 변환시킬 수 있다.

---

[22] 역자 주: 이 경우 한글로 방향이라고 표현되었지만 영어에서는 direction과 sense로 세분화 되어 있다. 더 자세한 내용은 아래 링크에 소개되어 있다.
http://darkwing.uoregon.edu/~struct/courseware/461/461_lectures/461_lecture4/461_lecture4.html

[23] 역자 주: 하나의 벡터로는 평면을 정의할 수 없다. 평면이 벡터를 축으로 하여 회전된 무한의 결과값을 얻을 수 있기 때문이다. 이 때문에 적어도 두 개의 벡터가 필요하며 이 두 벡터의 외적(cross product) 이 평면의 법선(일반적으로 z축)벡터가 된다. 자세한 내용은 다음을 참조
http://geometricmind.wordpress.com/2010/11/18/essential-mathematics-for-computational-design-09-%EB%B2%A1%ED%84%B0%EC%9D%98-%EC%99%B8%EC%A0%81/

`

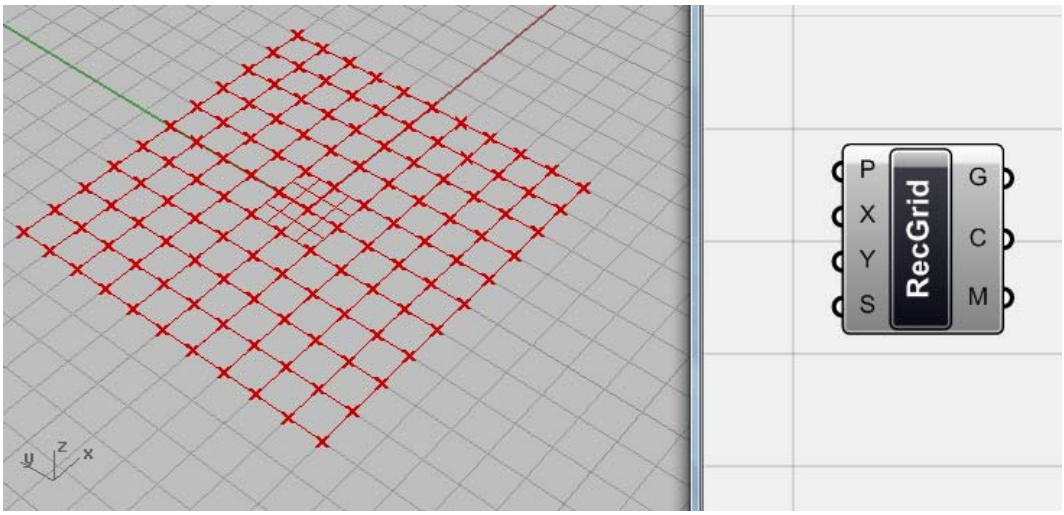
Grasshopper에는 컴퍼넌트화 된 기본적인 벡터들과 평면들이 있다. 바로 X 방향 단위벡터,Y 방향 단위벡터 그리고 Z 방향 단위벡터와 XY 평면, XZ평면, 그리고 YZ 평면이다. 이번 장에서는 이것에 컴퍼넌트를 추가하여 모델링을 하는 방법들을 살펴보도록 하겠다.

4_2_커브와 선형 기하체(On Curves and Linear Geometries)

위에서는 0차원 기하체 (0-dimensional geometry)에 대해 보았으므로 이번에는 1차원 기하체(1-dimensional geometry)에 대해서 살펴보도록 하자. Point와 curve는 다른 수많은 객체들을 생성하기 위한 기본단위가 될 수 있다. curve를 다른 하나의 curve를 따라 extrude하여 surface를 생성할 수 있고 다른 curve들을 연결하여 surface와 solid를 만들 수 있다. 혹은 어떠한 객체를 curve 위에 일정한 간격으로 늘어놓을 수 있다.

이탈(Displacements)

Chapter 3에서는 <series>와 <pt>를 이용하여 여러 종류의 point grid를 생성해 보았다. 이번에는 <Grid rectangular> (Vector > Point > Grid rectangular) 를 이용하여 다른 종류의 point grid를 생성할 것이다. 이 component에서는 X와 Y방향으로의 각각의 점의 개수와 점들 사이의 간격 (간격은 X, Y 방향에 같은 값이 적용)을 제어할 수 있다. [24]



<Grid Rectangular> 에 미리 정의된 값에 의하여 생성된 point grid

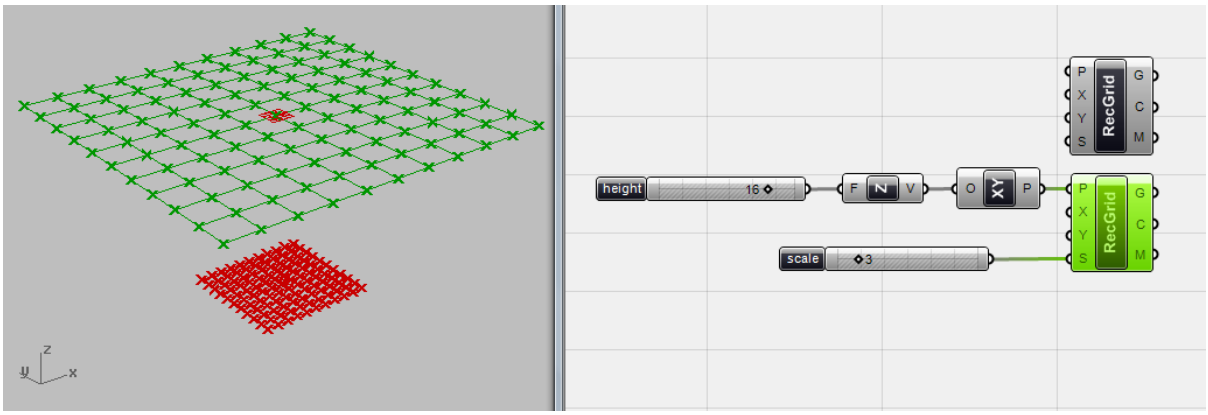
<number slider>를 (S)에 적용시키면 길이를 변화시킬 수 있다. 또한 point grid의 방향(orientation)을 바꿀 수 있다. 이것을 위해선 평면을 정의하고 그 위에 이 grid를 생성해줘야 한다.

---

[24] 역자 주: 이 예제는 예전 build의 grasshopper에서 만들어진 것으로 XY평면의 중심점이 곧 grid의 중심이 된다가 생성된다. 하지만 Build 0.80004 의 경우 gird전체의 왼쪽 아래 모서리가 중심점에 오게 된다. 그렇기 때문에 각 grid cell의 너비와 총개수를 <multiplication>을 이용하여 곱한 뒤 그것을 다시 <division>을 이용 2로 나누고 이를 <move>에 연결하여 X와 Y 방향으로 이동시켜주면 중심점을 맞출 수 있다. 각 grid cell의 중심점은 <area>를 이용하면 찾아줄 수 있다.

`

<XY plane> (Vector > Constants > XY plane)은 미리 정의된 평면으로 X축과 Y축 방향으로 생성된 평면이다. 이것을 <Z unit> (Vector > Constants > Z unit)을 이용하여 이동시켜준다. <Z unit>의 경우 그 크기는 1이다. 이것에 <number slider>를 연결하고 그 크기를 조절하면 point grid가 생성되는 평면의 높이를 조절할 수 있다. [25]



초록색으로 표시된 것이 위의 과정을 통해 생성된 point grid 이다. <number slider>를 이용하여 점들 사이의 간격과 Z 좌표의 위치를 조절하였다.

<grid rectangular>의 output은 각 grid cell의 curve 와 그 중심점[26], 그리고 모서리 점들이다. 이제 이 두 grid cell 각각의 중심점을 두 점으로 정의되는 <line>을 이용하여 연결하면 공간으로 퍼져나가는 듯한 형상을 가지는 선들을 만들어낼 수 있다. grid의 크기를 바꾸면 선들의 방향과 길이를 조절할 수 있다. 이 경우의 문제점은 모든 선들이 다른 길이를 가진 다는 것이다. 이 선들을 모두 같은 길이로 만들기 위해서는 <line SDL>을 이용하면 된다.

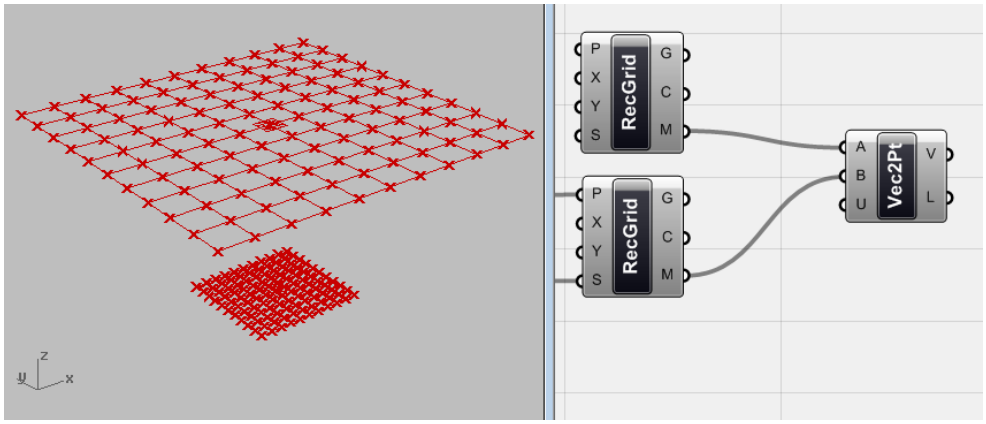
<line SDL>은 선을 선의 시작점(Start point; S), 방향(Direction ;D), 길이(Length; L)을 정의하여 선을 그려주는 역할을 한다. 즉 이 component를 이용하면 '길이'를 제어할 수 있다. 시작점을 grid cell의 중심점으로 하고 길이를 원하는 대로 정의한 뒤 그것의 길이를 원하는 만큼 <number slider>를 이용하여 정의해준다. 이에 필요한 vector는 <Vector 2pt>를 이용하여 두 grid의 중심점을 연결한 뒤 이것을 D에 넣어주면 같은 길이를 가지면서 퍼져나가는 형상을 유지할 수 있다.

---

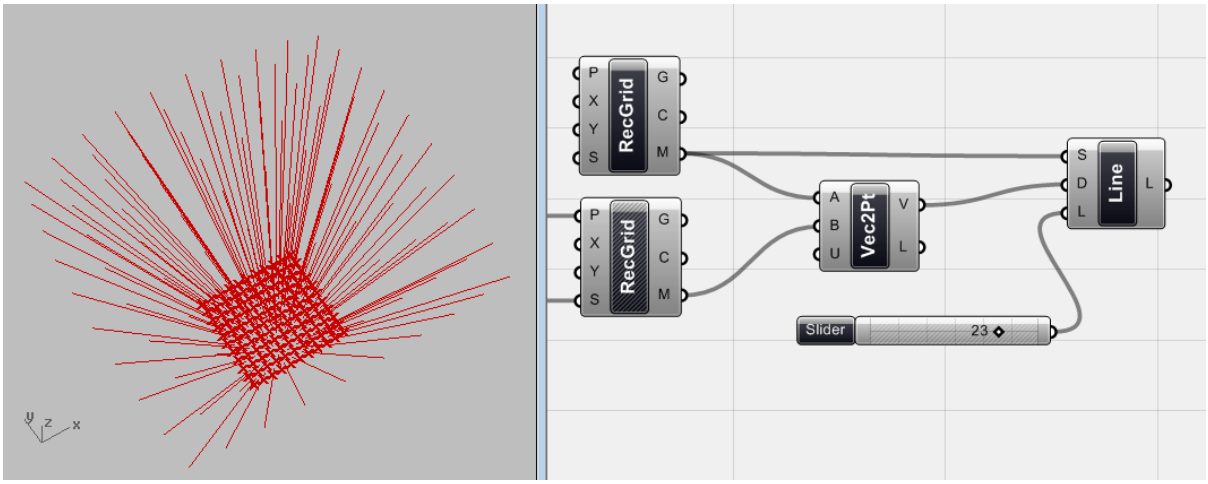
[25] 역자 주: grasshopper에서는 일반적으로 평면을 정의할 때 원점만을 정의해주면 된다. 보통은 XY평면에 평행하는 평면의 경우 Z축이 그 법선으로 자동 설정되며 따로 XY의 방향을 설정해줄 수는 없다. 이 예제의 경우 XY평면의 법선 (z축이 시작하는 점을 새로 정의해준다고 생각하면 된다. <Move>를 이용해도 되나 이렇게 하는 것이 더욱 간단하다.

[26] 역자 주: build 0.80004의 경우 M은 더 이상 output이 아니다. 이것은 C에 <area>를 연결하면 각 cell의 중심점을 찾을 수 있다.

`

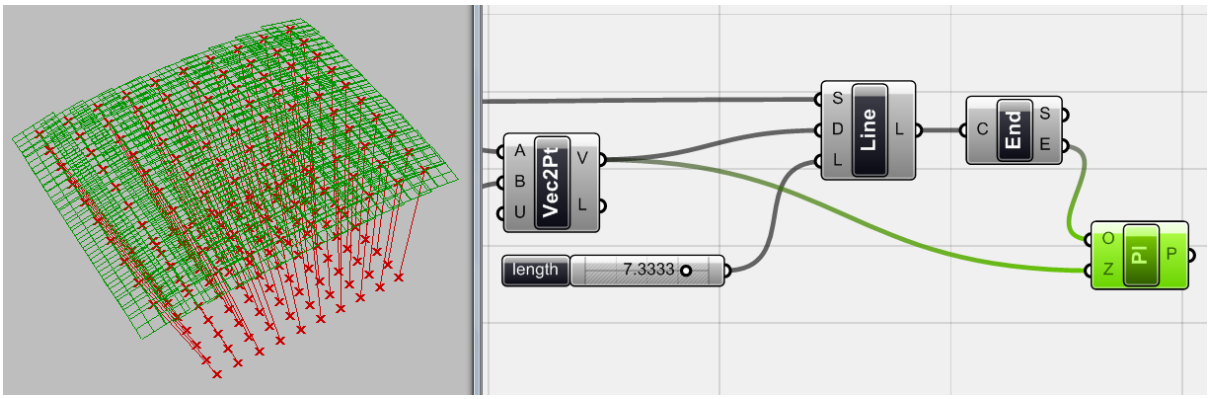


각 grid cell의 중심점을 연결하여 이것을 vector화 시킨 것이다. <vector 2pt>(vector > vector> vector 2pt)를 이용하면 된다.



<line SDL>을 이용하여 아래쪽 grid cell의 중심점으로부터 시작하여 퍼지는 형상의 선을 그릴 수 있다. 이 선들의 길이 또한 일정하게 제어할 수 있다. 두 번째 grid cell의 크기를 조절하면 선들의 방향도 조절할 수 있다.

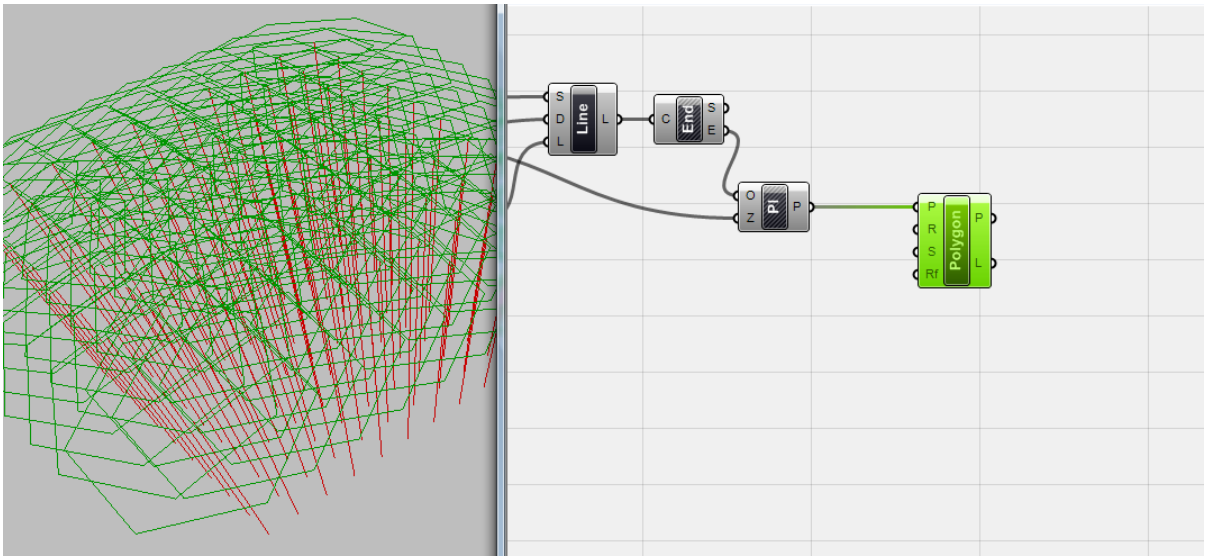
이제 <line SDL>의 한쪽 끝에 polygon을 그려보자. 이를 위해서는 먼저 line의 한쪽 끝에 polygon의 base가 되는 plane을 생성해주어야 한다.



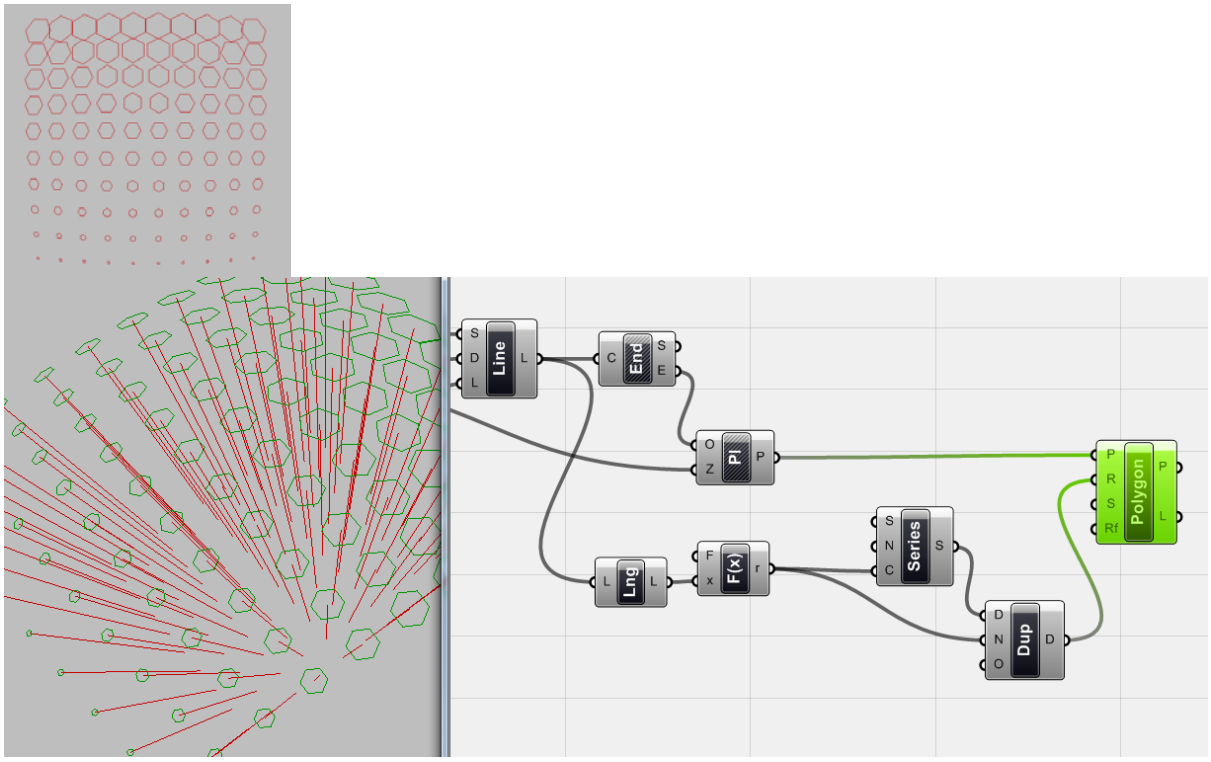
<end points>(curve > analysis)를 이용하여 선의 시작점과 끝점을 찾은 뒤 이 끝점들을 원점으로

`

그리고 위에서 찾아준 <Vec2pt>를 법선(plane의 수직방향 vector; normal vector)으로 하는 plane을 만든다. 이 경우 <plane normal> (vector > plane)을 이용하여 뭔점인 O에는 <end points>의 E값을, 그리고 법선 input인 Z에는 <vec2pt>를 연결하여 원점 data list와 vector 의 data list를 data matching 시켜준다.

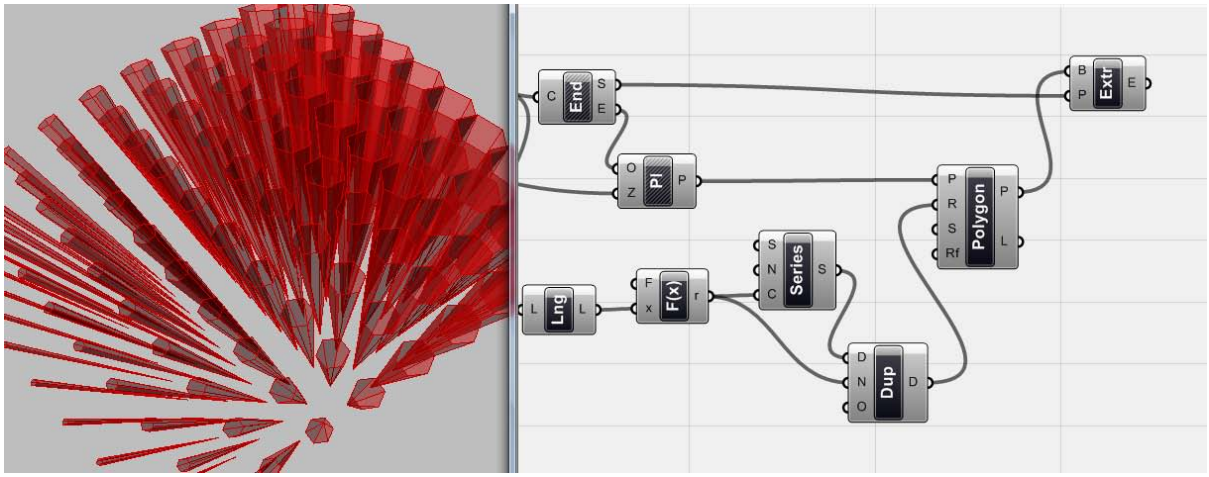


즉, <plane normal>을 이용하여 이 끝점과 vector를 input으로 받아들이면 각 점을 원점으로 하고 각 vector를 법선(normal vector)으로 하는 평면을 생성할 수 있다. 이렇게 생성된 평면을 <polygon>이 그려지는 기본 평면으로 삼는다. 즉 각각의 <polygon>들은 이 선에 수직으로 만나게 된다. 이제 이 <polygon>의 크기가 한 방향으로 서서히 작아지게 해보자.

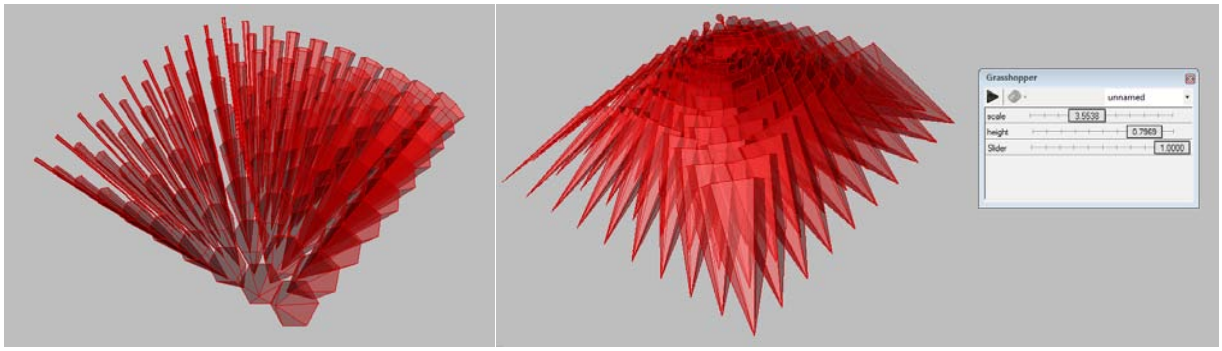


`

<list length>를 이용하여 선들의 개수[27]를 찾은 뒤 이것을 <function>(F(x)=Sqrt(x))을 이용하여 선의 개수의 제곱근(square root)을 구해준다. 이것은 각 열에 있는 선의 개수를 의미한다. 이제 <series>를 이용하여 '시작점(S)'과 '너비값(N)'을 0.1로 설정한다. 이렇게 나온 <0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7>은 한 행에 있는 7개의 polygon 각각의 Radius 값이 될 것이다. 이 data list를 <dup>에 연결하여 열의 개수만큼 반복 시킨다. [28]



이제 마지막으로 <extrude point>(Surface>Freeform)를 이용하여 각 line의 시작점을 P에, <polygon>에 연결해주면 위와 같은 poly surface를 만들 수 있다.



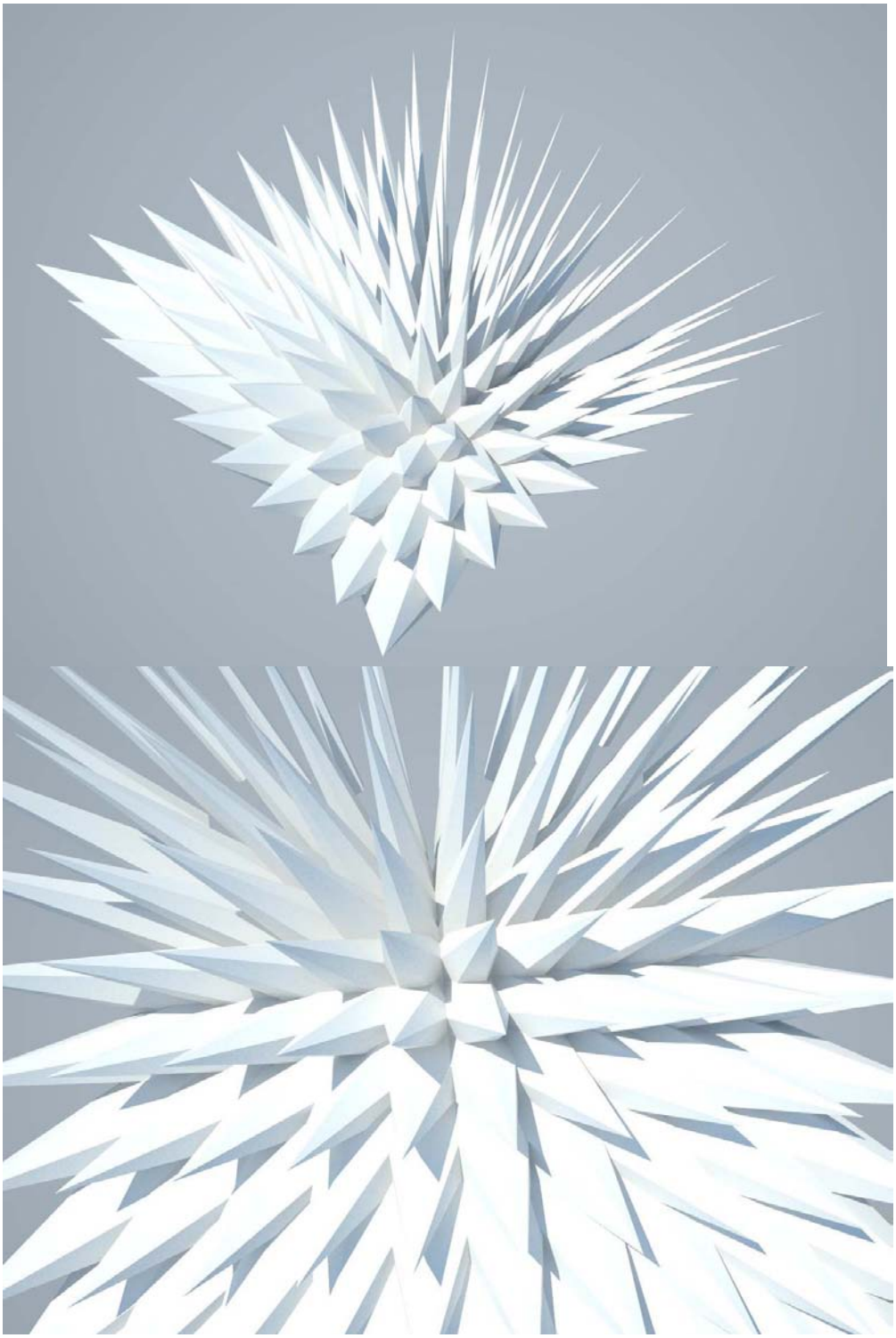
View menu에서 'remote control panel'을 이용하면 값을 조정하여 model의 전반적인 형상을 바꿔가며 마음에 드는 것을 찾아낼 수 있다.

---

[27] 역자 주: <list length>는 주어진 data list의 길이를 구하는 component이다. 즉 이 경우 선행하는 것은 선의 data list이므로 선의 개수를 알아내는 것과 마찬가지 이다.

[28] 역자 주: <dup>을 통해서 나오는 data list는 <0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7> 이다. 이 <dup>의 O를 true로 바꿔주면 data list는 <0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7>로 바꿔줄 수 있다.

`



`

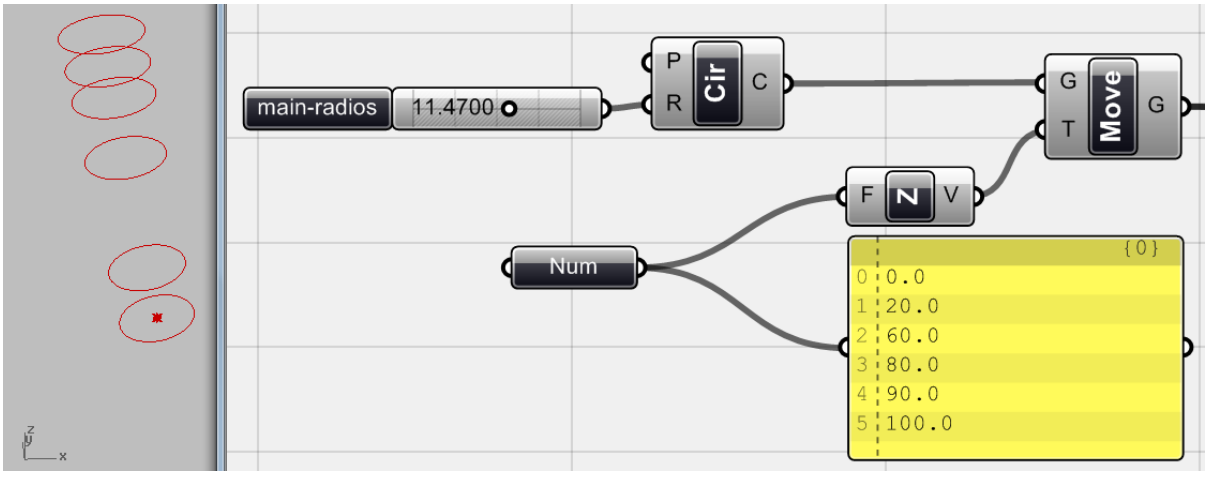
4_3_Combined Experiment: Swiss Re

오늘날에 지어지는 고층건물들의 design concept은 모델링 방식과 연관되는 것이 많다. 이는 designer들이 더 빠르고 간편하게 다양한 제안을 하는 것을 가능하게 한다. "Foster and partners" 에서 설계한 "Swiss Re" 의 모델링을 통하여 더 자세한 내용을 살펴보도록 하자.

Swiss Re Head Quarter, 30 St Mary Axe, London, UK, 1997-2004, Photos from Foster and Partners website, http://www.fosterandpartners.com.

모델링 과정을 간단하게 기술하자면 먼저 수직선상에 여러 개의 원을 그린 뒤 그것의 scale을 조절한다. 이는 건물의 전반적인 형태를 결정해줄 것이다. 그리고 이 원들을 연결하여 건물의 표면을 만든다. 이 표면에 입면의 구조체의 기본이 되는 삼각형을 배열할 것이다. 여기서 사용된 수치는 기본적인 비례를 얻기 위한 것으로 정확한 것은 아니다.

먼저 floor를 그려보도록 하자. 원래 건물은 원형이 아닌 톱니형이지만 이 예제의 편의상 원형을 사용하였다. 이 원형을 특정 높이로 복사한 뒤 scale을 조절하여 건물의 전반적인 형상을 잡아보자.

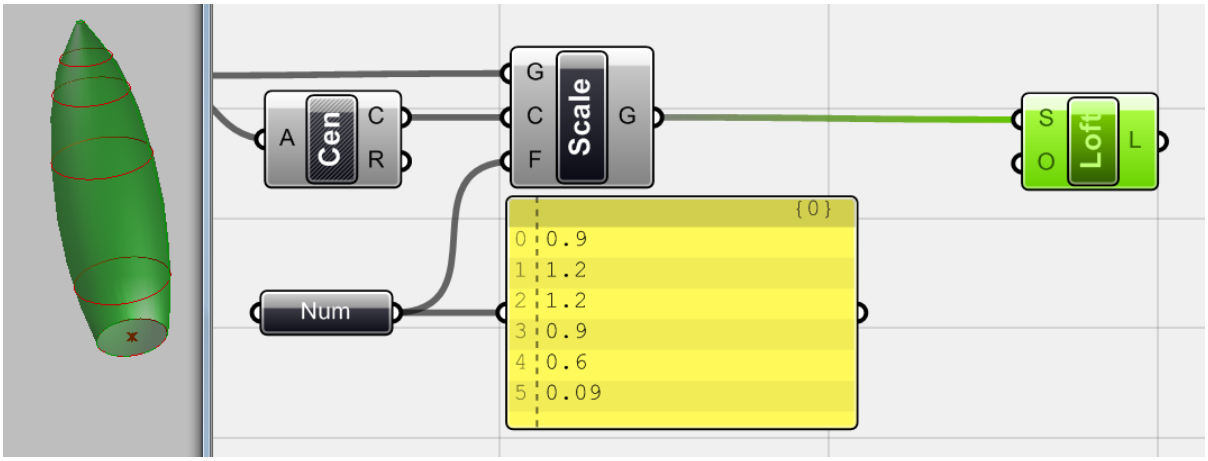


<scale> (XForm > Affine > Scale) 을 이용하여 주어진 원들의 scale을 조절한다. 이것에 필요한 input은 scale이 되는 기준점(이 경우 원의 중심)과 축적이 변경될 기하체이다. 이 <scale>의 G에

`

<move>로부터 나오는 원들을 물린다. Scale의 기준점은 각 원들의 중심점이 된다. <move>의 g를 <centre> (Curve > Analysis > Centre) 에 연결시켜주면 각 원의 중심점을 찾을 수 있다. 이 의 c를 <scale>의 c에 연결하면 각 중심점과 스케일이 data matching 되며 원들의 스케일이 바뀐다. F에 들어가는 것은 scale에 적용될 배율이다.

이 때 사용된 수치들은 모두 추측에 의한 값으로 가장 알맞은 형태를 찾기 위하여 수치를 바꿔가며 작업할 수 있다. 이 수치들은 모두 <num>에 저장 된다.[29]

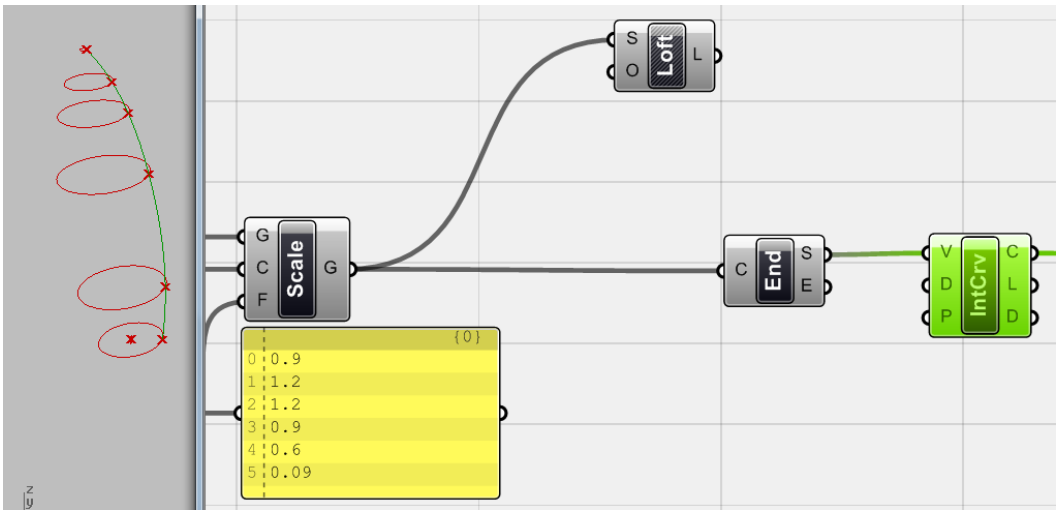


이 원들을 <loft> (surface > freeform > loft)를 이용하여 연결해주면 처음 사진의 건물과 같은 곡면이 생성된다.

이제 입면을 그려보자.

입면 구조체들은 helix와 같은 형태를 띄고 있다. 이것의 실제 단면(cross section)은 다이아몬드 형태이나 보이는 부분만을 고려하여 이번 예시에서는 삼각형을 사용하도록 하겠다. 이것들을 <loft> 하면 입면 구조체를 그릴 수 있다.

먼저 적절한 위치에 이 기본 삼각형을 배열하도록 하자. 이를 위하여 입면을 가로지르는 curve를 만든 뒤 이것을 일정 간격으로 나누고 (division) 각 점 위에 삼각형을 배열해보도록 하겠다.

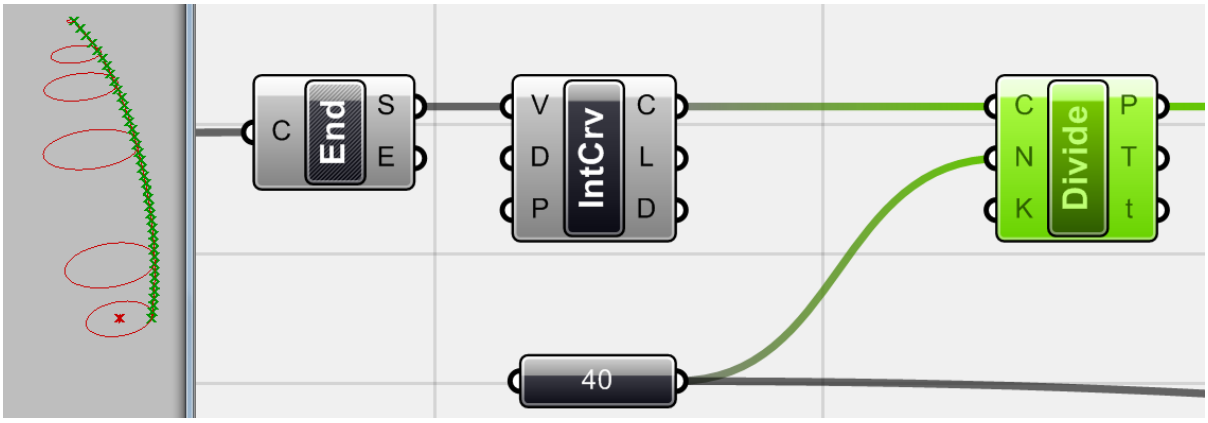


<end points>를 이용하여 각 원의 시작점과 끝점을 찾는다. 그리고 각 점을 지나가는 curve를

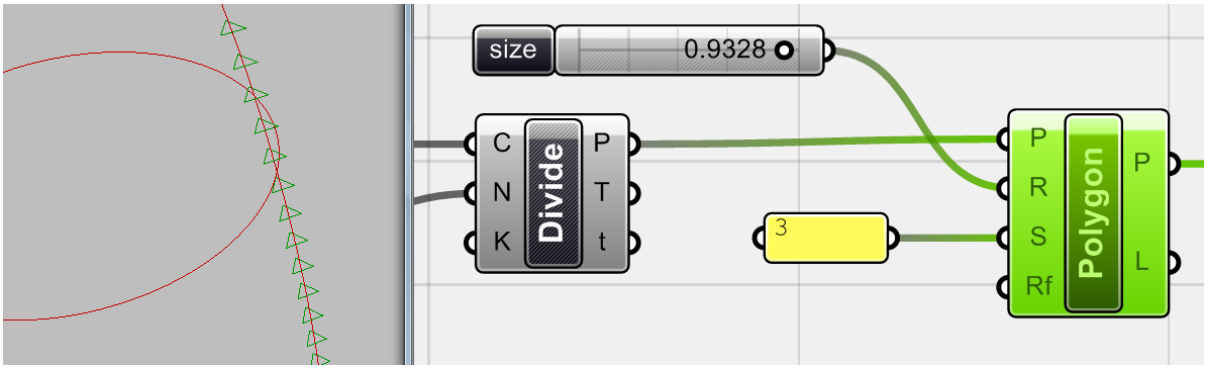
[29] 역자 주: context menu에서 'manage number collection'을 사용한다.

`

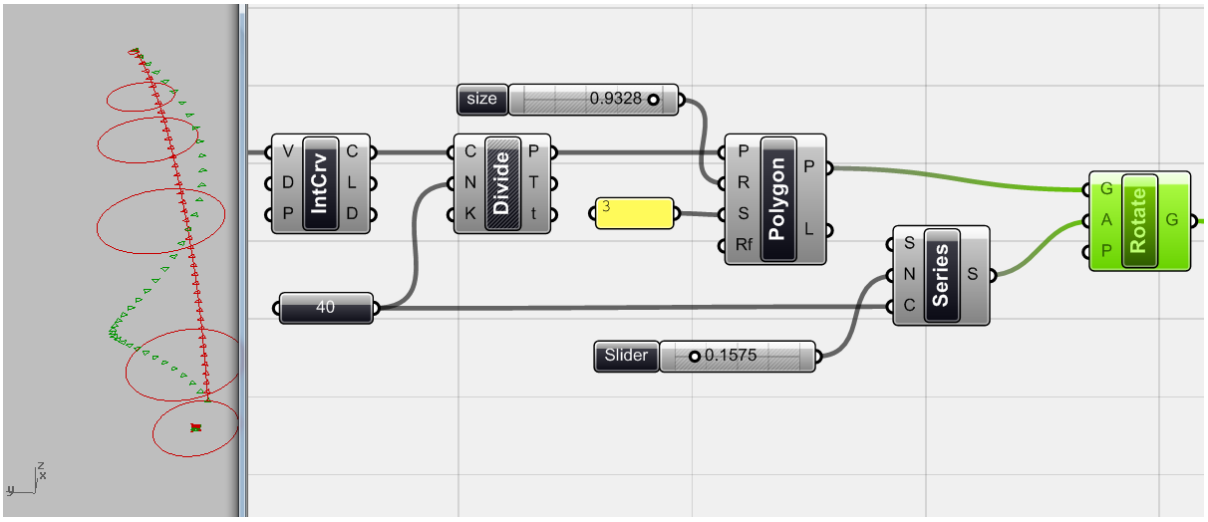
<interpolate> (curve > spline > interpolate)를 이용하여 그려준다.



이제 <divide>를 이용하여 <interpolate>에서 나오는 curve를 40등분 해준다. 이 N값이 클수록 커브가 부드러워 진다.



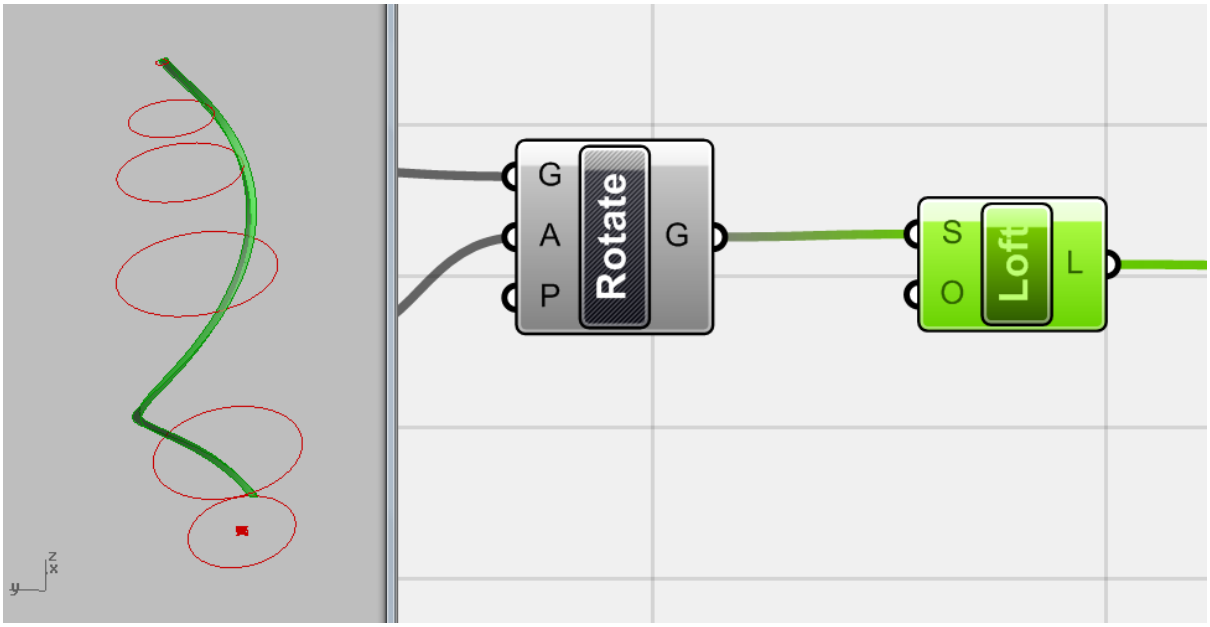
이렇게 등분된 점들이 바로 삼각형이 놓이는 평면의 원점이 된다. <polygon>을 이용하여 façade 위에 삼각형을 그린다. <polygon>의 S는 변(side)의 개수를 R은 외접원의 반지름(radius)을 의미한다. S에 3을 연결하고 <number slide>를 이용하여 적절한 수를 R에 넣어준다.



입면 구조체는 입면을 휘감아 올라가게 된다. 이를 위하여 삼각형들을 일정한 각으로 회전시켜주면된다. 이를 위하여 <rotate>가 필요하다. G는 기하체(geometry), A는 <Angle> p는 <plane>이다. <series>를 이용하여 angle 값을 만들어준다. 먼저 <divide>의 C에 사용된 40이라는 수를 c에

`

넣어준다. N은 각 삼각형의 회전 각도 값이다. 이 N값을 등차로 하는 40개의 수를 수열로 만들어 40개의 삼각형에 적용시켜준다. [30] 이 결과로 삼각형이 입면을 휘감아 올라가는 것을 확인할 수 있다.



이제 <loft>를 연결해주면 위와 같이 입면의 구조체가 생성되는 것을 볼 수 있다. 각도 값[31]을 조절하여 가장 이상적인 형태를 찾도록 하자.

정의역(Domains)

위에서 언급한 것처럼 <domain[32]>은 수의 범위이다. 실수(real number)를 이용하여 양 끝값을 정의해줄 수 있다. '실수(real number)'의 범위이므로 그 사이에는 무한개의 수가 있을 수 있다. 이 범위 내에 일정한 간격의 수열을 만들기 주기 위해서는 <range>가 필요하다.

이것을 이용하여 입면의 구조를 일정한 간격으로 skin 둘레에 배열해보자.

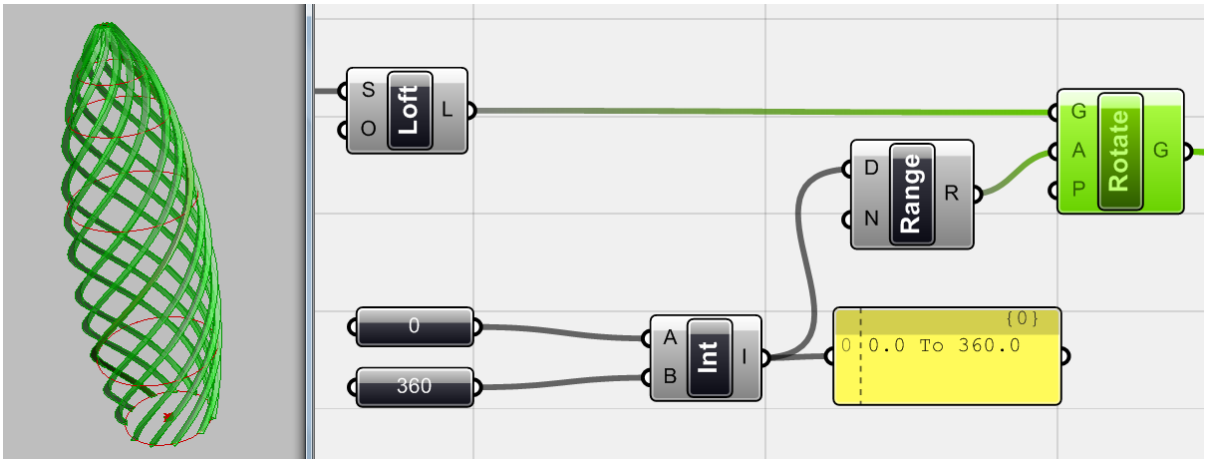
---

[30] 역자 주: 엄밀하게 따지면 curve를 40등분 하기 때문에 점은 41개가 생기고 그러므로 삼각형 또한 41개가 된다. 저자의 방법대로 하면 제일 위에 있는 삼각형이 그 바로 아래 삼각형에 비하여 회전하지 않은 것을 확인할 수 있다. <series>의 C를 우클릭 한 뒤 expression에 c+1 을 넣어주면 c에 들어오는 값에 1을 더하여 input으로 활용하게 된다. 또한 <rotate>에 필요한 angle 값은 radian 값이다. 우리가 일상적으로 사용하는 각도값으로 바꾸기 위해서는 <rotate>의 A를 우클릭 한 뒤 expression에 rad(A)를 넣어주면 된다.

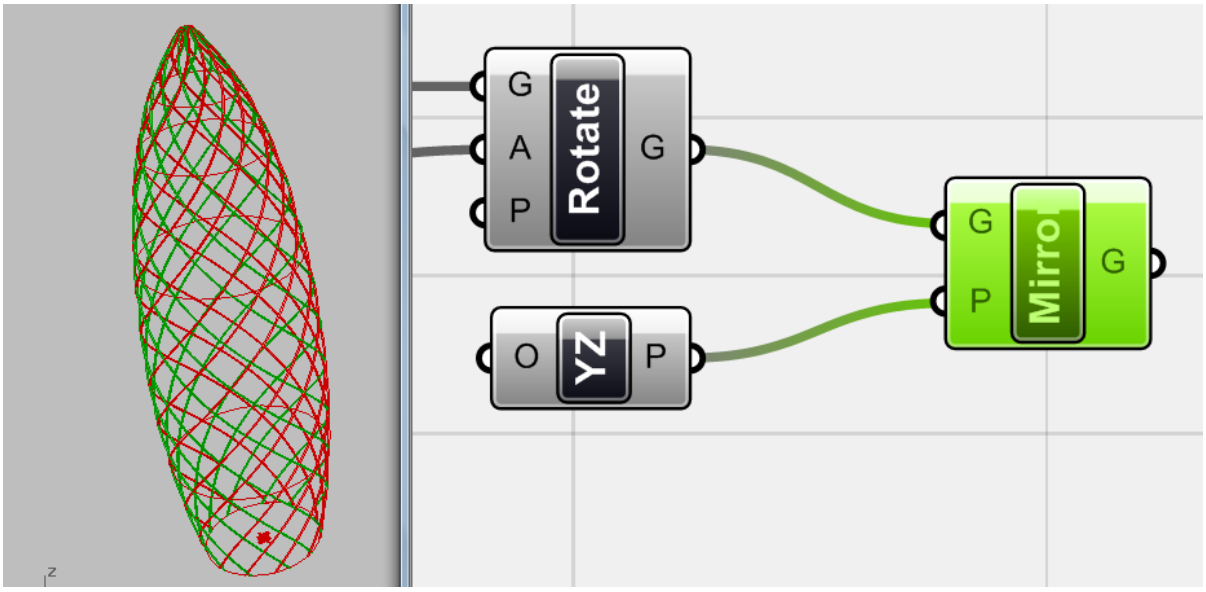
[31] 역자 주: 이 경우 series의 N 값이 바로 그 각도 값이 된다.

[32] 역자 주: 원문의 경우 그림<interval>이 사용되었으나 현재 버전에서는 <domain>으로 통합되었다.

`



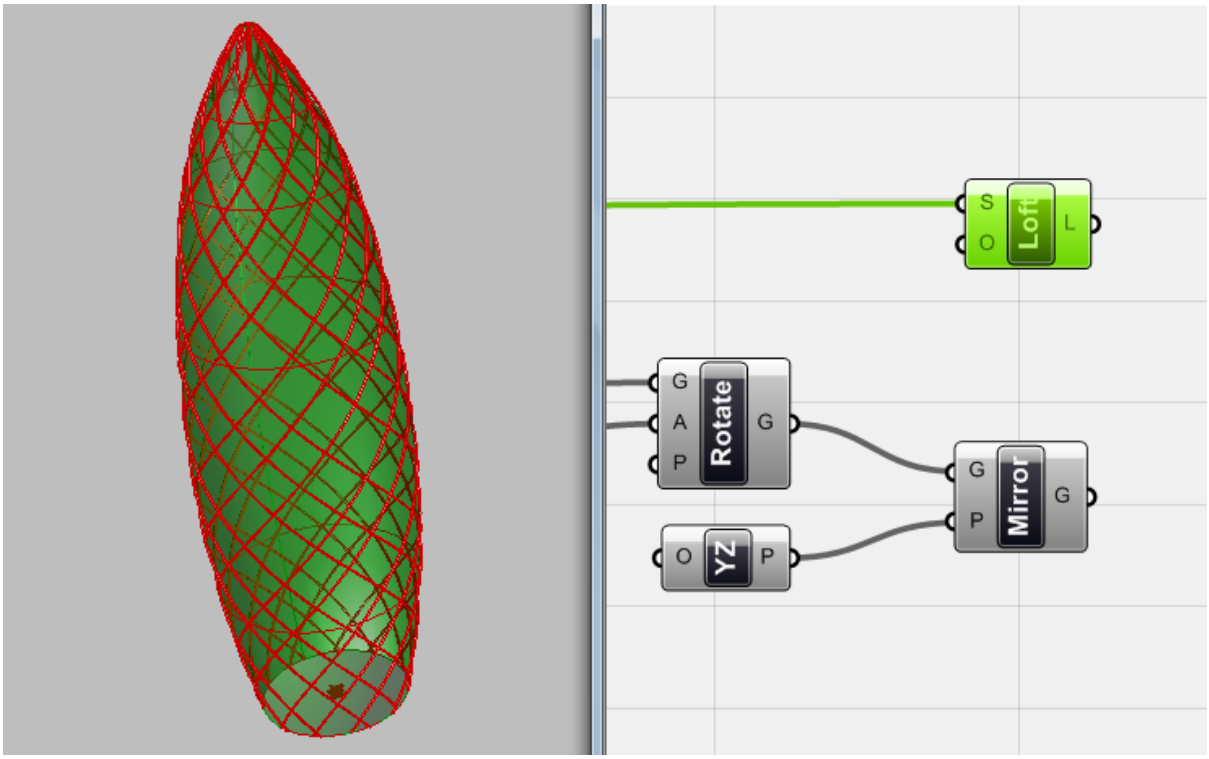
<domain>을 통하여 수의 범위를 0에서 360까지 주어준다. 그리고 <range>를 이용하여 이 범위를 10등분 해준다. 이 값을 <rotate>의 A에 연결해준다.[33] 그 결과물은 위와 같다.



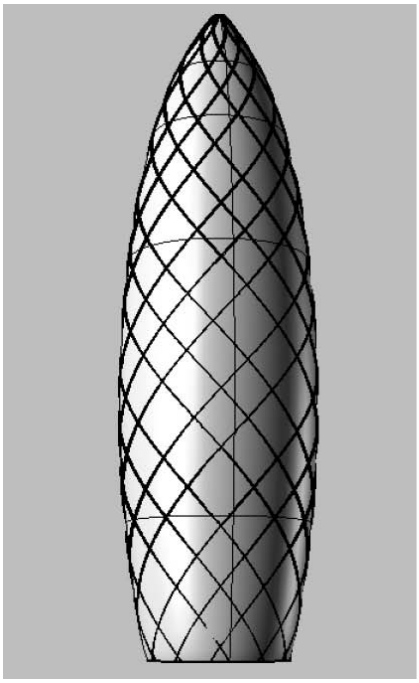
이제 <mirror> (XForm > Euclidian > Mirror)를 이용하여 구조체들을 <YX plane>에 대칭 시켜준다. 이 결과로 skin 주변에 위와 같은 격자무늬가 생성된다.

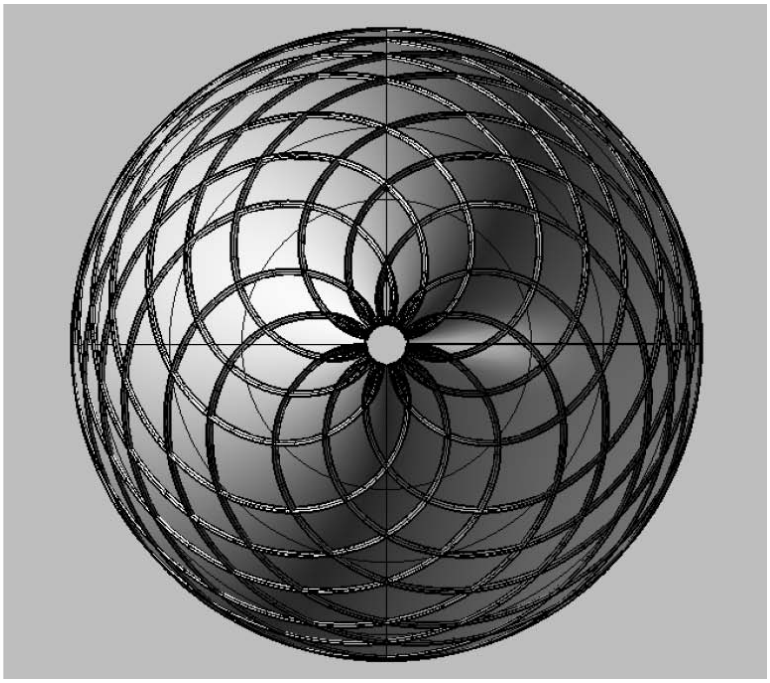
[33] 역자 주: 이 경우 또한 마찬가지 이다. 위와 같은 결과를 얻기 위해서는 <rotate> A의 expression에 rad(A)를 넣어주어야 한다.

`



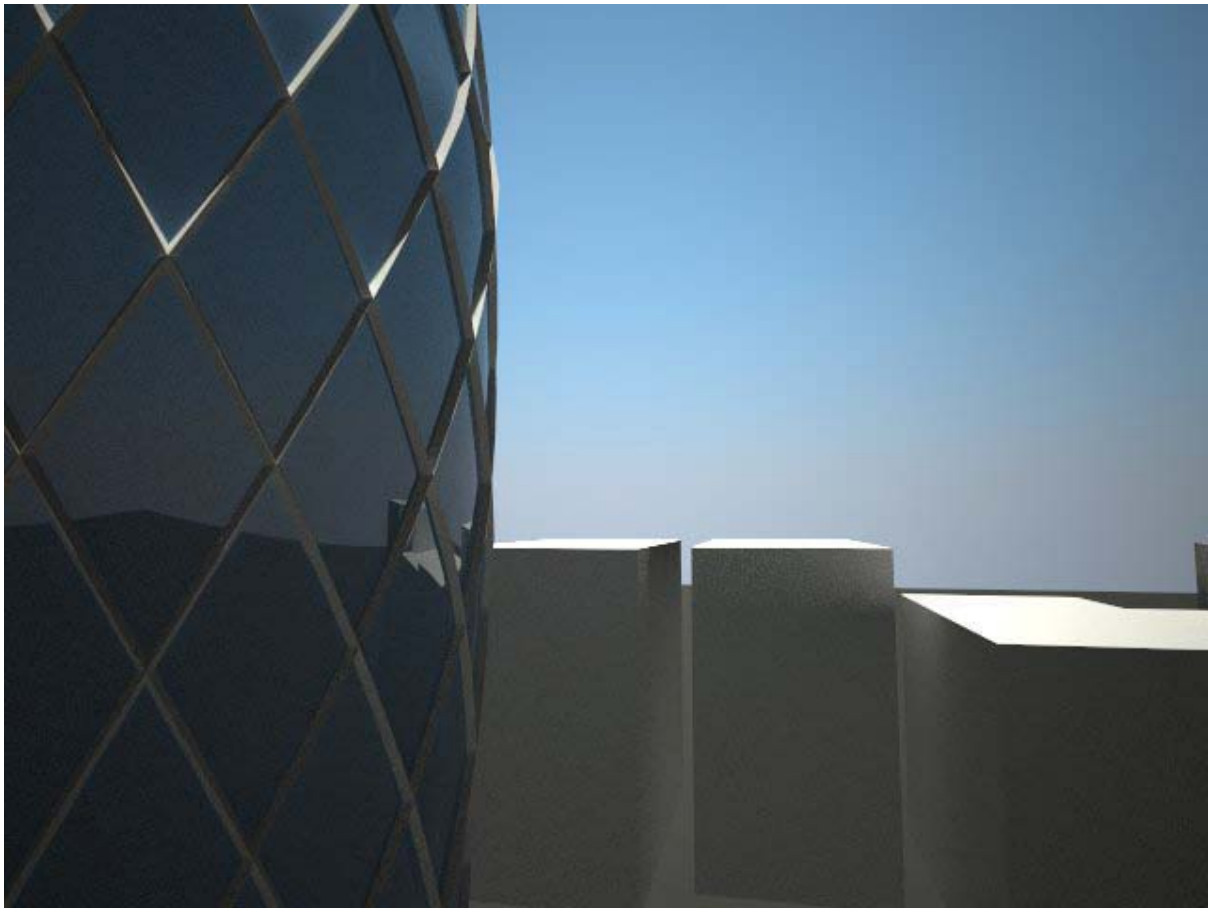
Skin을 생성하는 <loft>와 구조체를 생성하는 <mirror>를 제외하고 preview를 꺼주면 위와 같은 모습이 된다. 위 예시를 통하여 "Swiss Re"를 간략하게 나마 모델링 해보았다.





위의 component들을 선택한 뒤 터진 계란프라이 모양의 'Bake Selected Objects'를 누르면 위와 같은 rhino object를 생성할 수 있다.

`



최종 모델의 모습이다. Grasshopper를 활용하면 원래 건물과 완벽히 같지는 않더라도 간략한 모델을 짧은 시간에 생성할 수 있다.

입면을 자세히 보면 개중에 열리거나 하는 등 조금 다른 형태를 가진 것들이 있다. 이때까지 배운 것들을 잘 활용하면 이 또한 어렵지 않게 만들 수 있다.

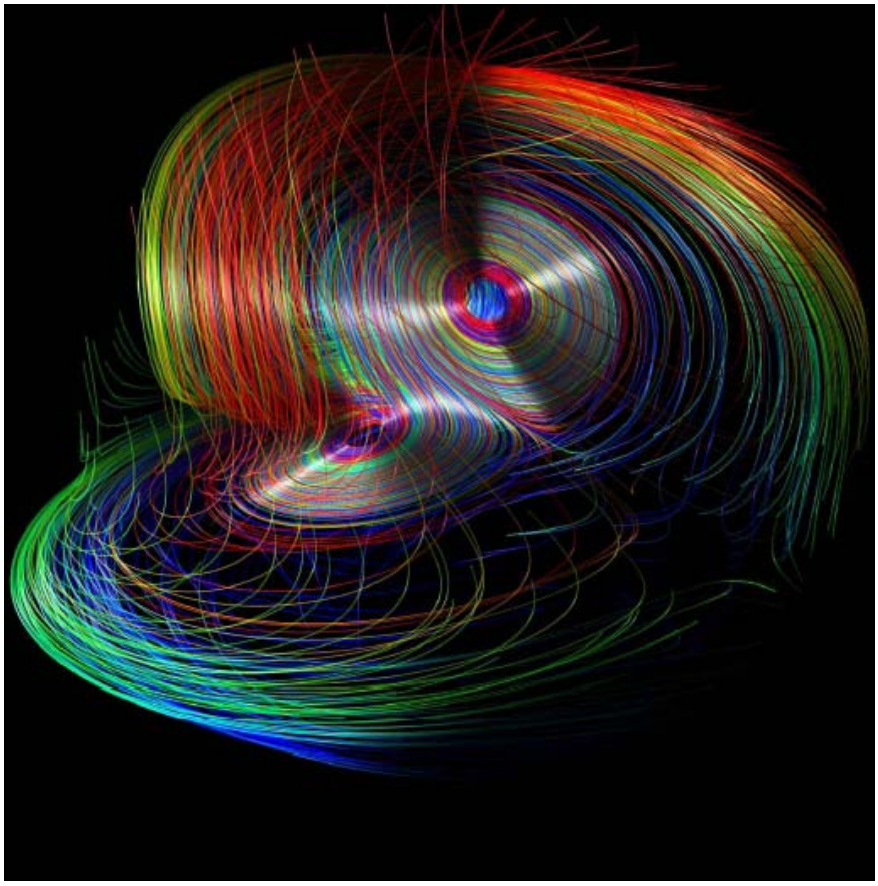
`

`

4_4_끌개(On Attractors)

"Attractor is a set of states of a dynamic physical system towards which that system tends to evolve, regardless of the starting conditions of the system. A **point attractor** is an attractor consisting of a single state. For example, a marble rolling in a smooth, rounded bowl will always come to rest at the lowest point, in the bottom center of the bowl; the final state of position and motionlessness is a point attractor." (Dictionary.com/Science Dictionary)

끌개(Attractor)는 역학계의 개념의 하나로서 변수의 시(時)적 변화, 또는 위상공간상의 비동시성(asynchronism)의 표현으로, 위상수학에 바탕을 두고 연구되며, 물리학과 생물학에 주로 적용된다.



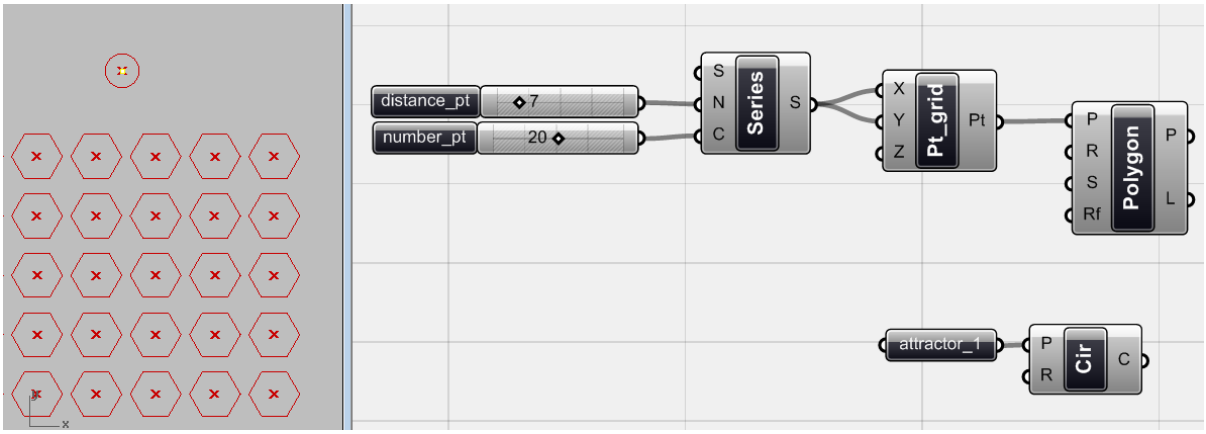
이상끌개(Strange Attractor) (http://www.cs.swan.ac.uk/~cstony/research/star/)

Design과 기하학에 있어서 끌개 (attractor)는 공간 내에 있는 기하체에 영향을 끼치면서 그것들이 반응하는 방식에 영향을 끼친다. Re-orientate 하거나 축적을 줄이거나 원래의 위치로부터 일탈시키는 것 등이 그것이다. 끌개는 주변 공간을 분절시키기도 하고 미치는 영향의 범위를 결정할 수도 있다. 끌개들은 parametric design에서 여러가지로 적용이 가능하다. 이것이 주변 모든 객체를 일정한 규칙에 따라 변화시킬 수 있기 때문이다. 또한 하나의 장(field)에 여러 개의 끌개를 넣는 것도 가능하다. 끌개가 영향을 끼치는 방식과 그 영향력 모두 조절이 가능하다. 이러한 끌개의 개념을 활용하여 여러가지 실험을 해볼 것이다. 간단한 예시부터 살펴보도록 하자.

`

Point Attractors

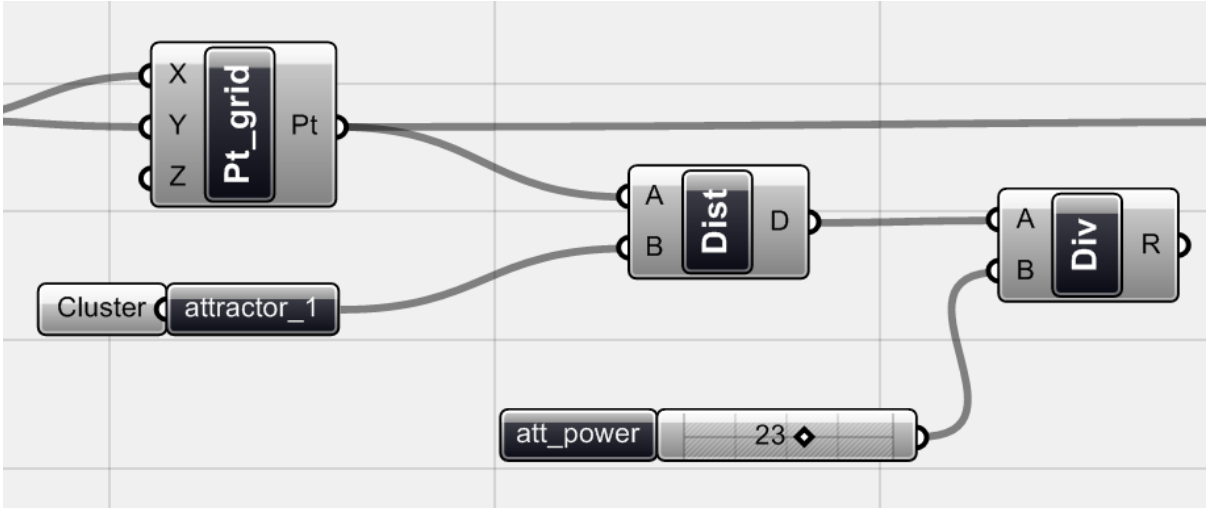
Point grid를 이용하여 일단의 polygon을 생성한다. 또한 rhino에서 점을 하나 그린 뒤 이것을 <point> 에 연동시킨다. 이름 또한 <attractor_1>으로 바꿔주도록 하자. 이 점을 <circle>에 연결하여 이 점을 중심으로 하는 원을 그리도록 하자. 이 <attractor_1>이 끌개가 되어 주변에 있는 polygon의 장(field)에 일정한 영향을 끼치게 한다. 즉, 이 <attractor_1>가 영향을 끼치는 범위 않에서는 각각의 polygon들이 그 크기를 변화시키게 된다.



바탕이 되는 <point_grid> 와 <polygon>들 그리고 끌개<attractor_1>.

이것의 algorithm은 무척 간단하다. <point grid>상의 점과 <attractor_1> 사이의 거리(distance)가 바로 polygon들의 외접원의 반지름이 되는 것이다. 즉 끌개와 polygon들의 관계는 바로 이 `거리`인 것이다.

<distance>를 이용하여 <attractor_1>과 각 점들 사이의 거리를 측정해준다. A에 <point grid>를, B에 <attractor_1>을 연결시켜준다. 이 거리 값이 바로 polygon의 지름이 되기엔 너무 크다. 이 값들을 <division> (Scalar > Operators > Division) 을 이용하여 나눠주도록 하자. 거리 값을 <division>의 A에, <number slider>를 B에 연결하고 적절한 수를 설정해주도록 한다.

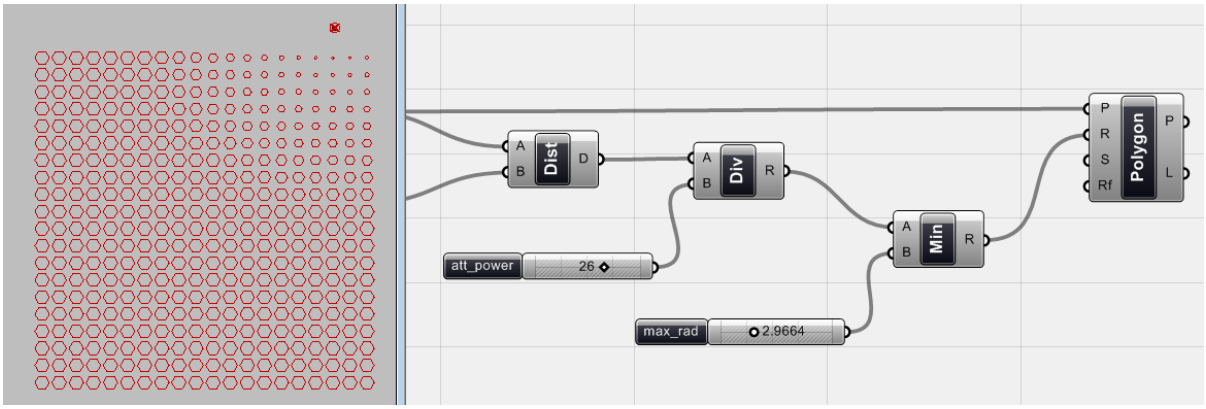


이렇게 나눠져서 나오는 수가 바로 끌개의 영향력이 된다. 위 그림의 경우 <point>와 <circle>이cluster로 묶여 있는데 새로운 grasshopper 버전에서는 cluster 기능이 사라졌음을 참고하길 바

`

란다.

이제 <division>에서 나오는 값을 <polygon>의 R에 연결해주면 끌개로부터 너무 멀리 떨어져 있는 polygon의 경우 그 크기가 지나치게 커지는 것을 알 수 있다. 이 거리가 일정 값을 넘지 않도록 조절해주도록 하자. 즉 R 값이 커질 수 있는 최대값을 정하는 것이다.

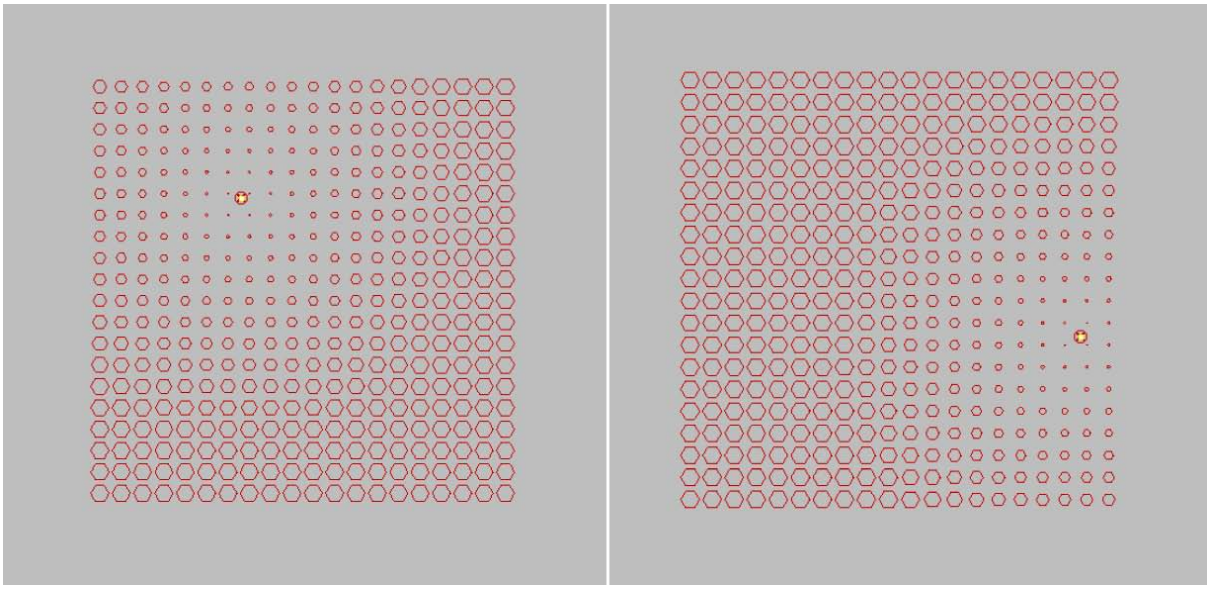


이러한 역할을 하는 것이 바로 <minimum> (Scalar > Util > Minimum)이다. 이는 input으로 주어진 두 값을 비교하여 더 작은 값을 결과물로 내보내준다. 즉 이 경우 A로 들어오는 값 중 B에 설정된 값보다 큰 값이 있으면 B로 대체해 주는 것이다.[34] 이를 통해 <polygon>의 R 값에 최대값을 설정해줄 수 있다. 이렇게 최대값의 영향을 받는 polygon들은 끌개의 영향력 바깥에 있다고 생각할 수 있다.

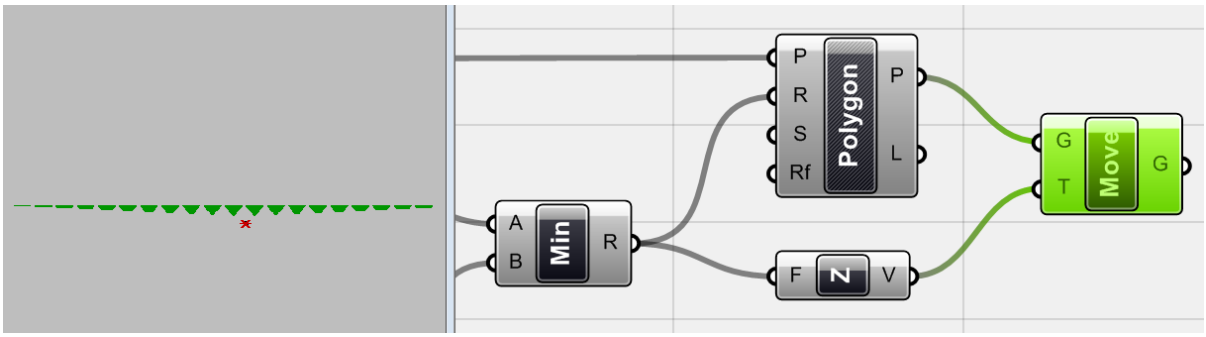
이제 rhino 상에서 <attractor_1>를 움직이면 영향력의 범위 내에 있는 polygon들의 크기가 변하는 것을 볼 수 있다.

[34] 역자 주: 만약 A에 (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) 이라는 data list가 들어오고 B에 (5.5)가 있으면 <minimum>을 통해 나오는 data list는 (1, 2, 3, 4, 5, 5.5, 5.5, 5.5, 5.5, 5.5) 이다.

`

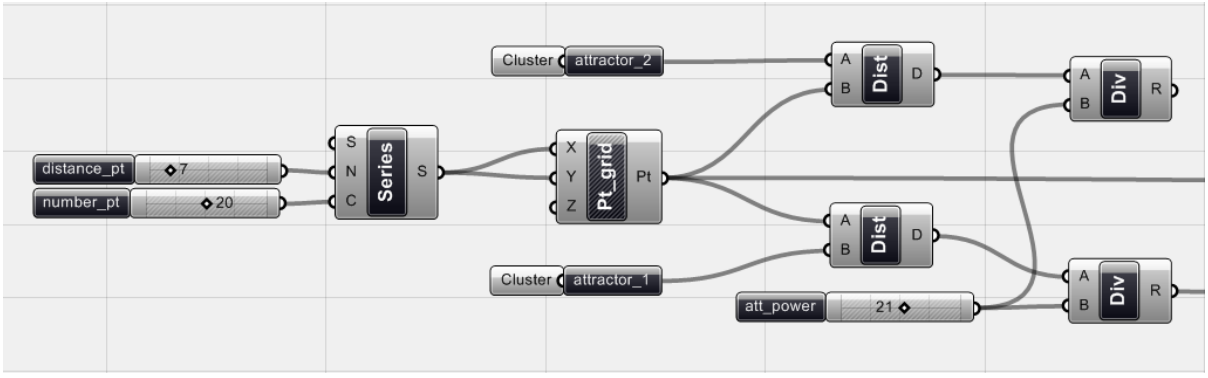


<attractor_1> 의 영향을 받는 polygon들. <attractor>의 위치에 따라 그 크기가 변한다.



같은 개념을 이용하여 polygon을 z 방향으로 <move>시킬 수 있다. 즉 <minimum>에서 나오는 값을 <move>의 z에 적용시켜주는 것이다. 끌개의 개념을 크기 뿐만 아니라 높이에도 적용시킬 수 있는 것이다. 이 뿐만 아니라 끌개와의 거리에 따라 polygon이 회전한다 던가 그 색을 바꿔줄 수 있다.

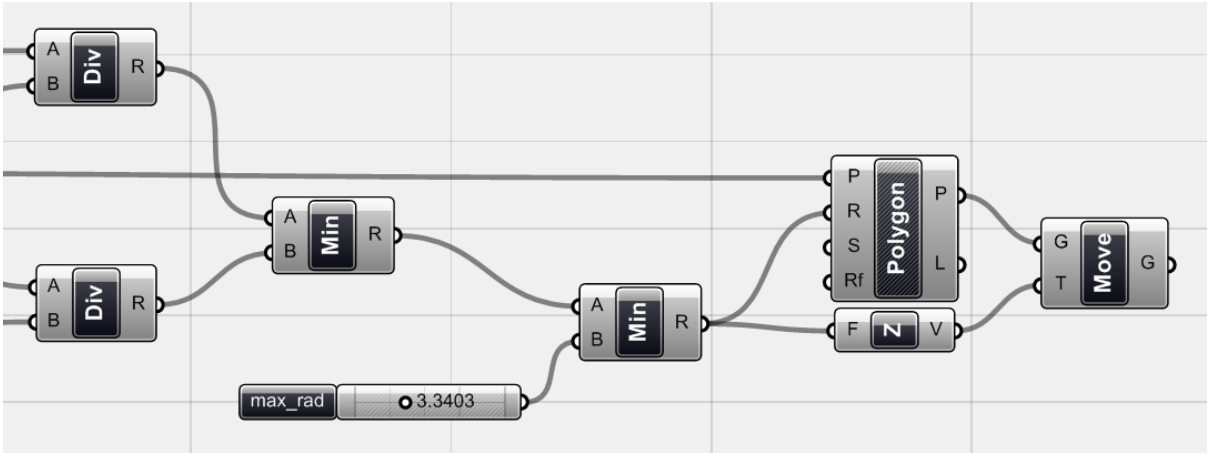
만약 두개의 끌개가 있다면 어떻게 될까? 또 다른 <point>와 <circle>을 이용하여 <atrractor_2>를 만들고 <distance>에 연결해준다. 여기서 나오는 거리값을 <division>할 때 사용되는 분모에는 <attractor_1>에 사용했던 <number slide>를 똑같이 적용해준다.



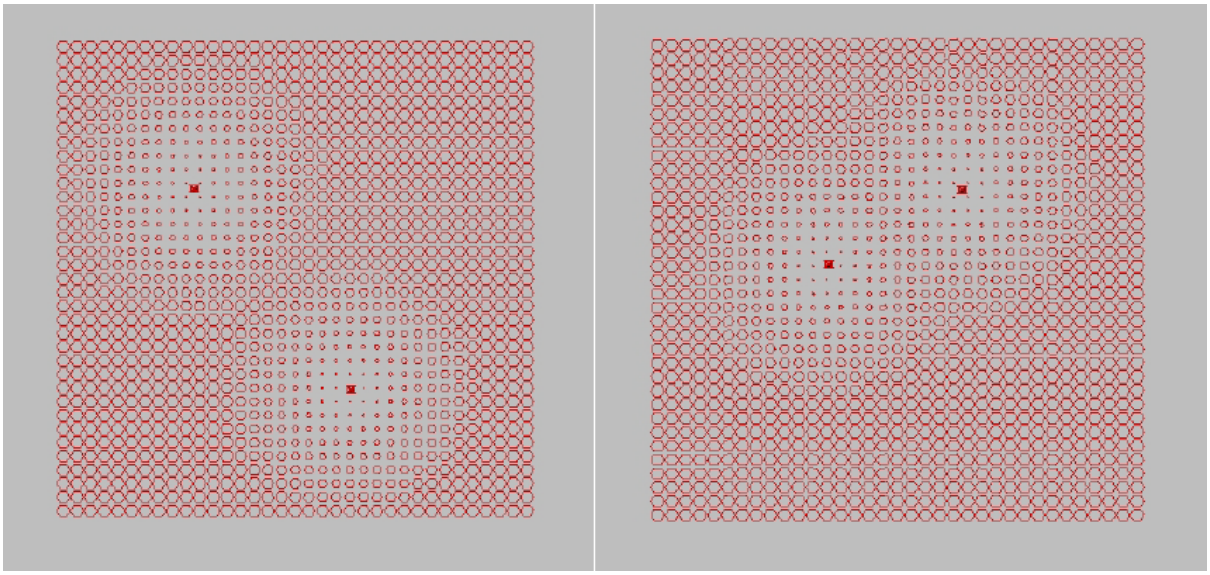
두 번째 끌개를 적용시키고 같은 algorithm 을 적용시켜 준다.

`

이제 polygon에서 각각의 끌개까지의 거리를 가지고 있는 두 개의 data list를 가지게 된다. 이 두 data list 를 <minimum>의 A와 B에 연결하여 작은 수만으로 이루어진 data list로 만들어준다. 이것을 또 다시 <minimum>에 연결하여 그 최대값을 제한해준다.



두 개의 <minimum>을 이용하여 각 polygon들이 더 가까운 끌개의 영향을 받게 하였다. 이 polygon의 장은 두 끌개의 영향을 받는 것 처럼 보이게 되었다.



두 끌개의 위치에 따라 polygon의 크기가 영향을 받게 된다.

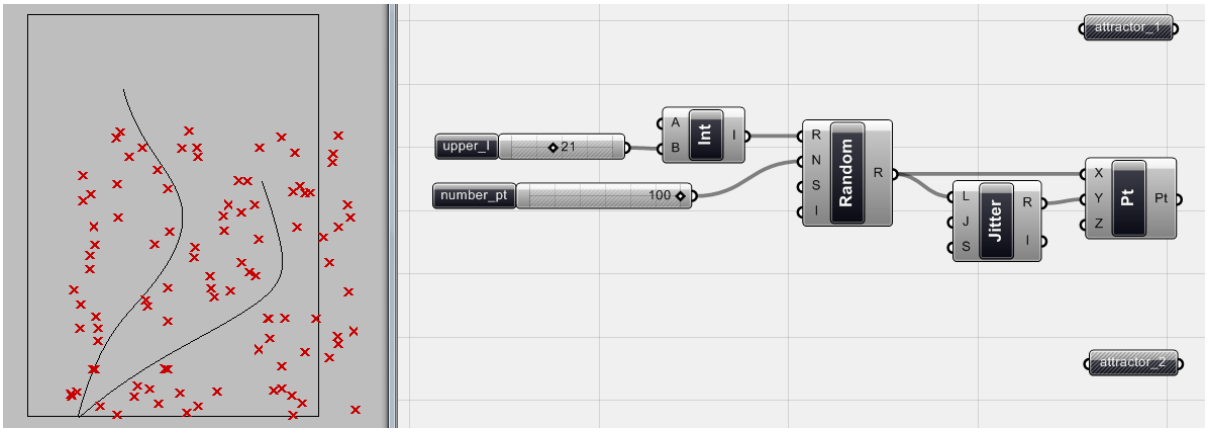
여기에 더 많은 수의 끌개를 더할 수 있다. 기본적인 개념은 polygon들이 가장 가까운 끌개로부터 영향을 받아 미리 정해진 반응을 보이는 것이다. 이것은 아주 많은 수의 요소들을 한번에 조정하기에 유용하다.

선형끌개 (Curve Attractors: Wall project)

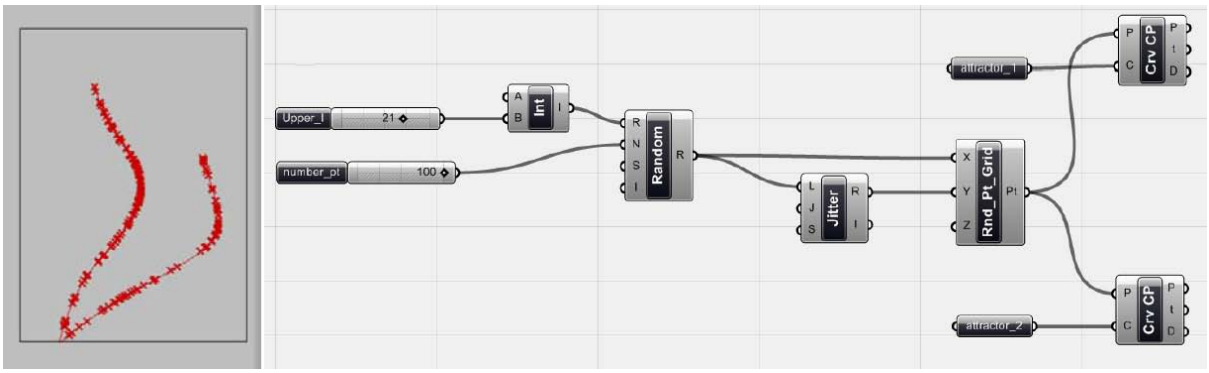
이제 curve를 끌개로 활용하는 법에 대하여 살펴보도록 하자. 이 예제의 목표는 벽을 만드는 것

`

으로 내부에서 외부를 여러 개의 구멍을 통해 바라볼 수 있게 하는 것이다. 이를 위해서는 먼저 벽이 될 평면과 두 개의 curve, 그리고 불규칙하게 뿌려진 점들이 있어야 한다. 이 점을 중심으로 그려진 사각형이 벽에서 잘려 나가는 부분이 될 것이다. 이 때 사각형의 위치가 무작위로 흩뿌려진 것이 아닌, 두 개의 curve 를 따라 그 주변에 올 수 있도록 하려고 한다. 이 curve는 거시적으로 사각형의 전반적인 위치를 결정하며 미시적으로 사각형의 위치에 무작위성을 부여한다.

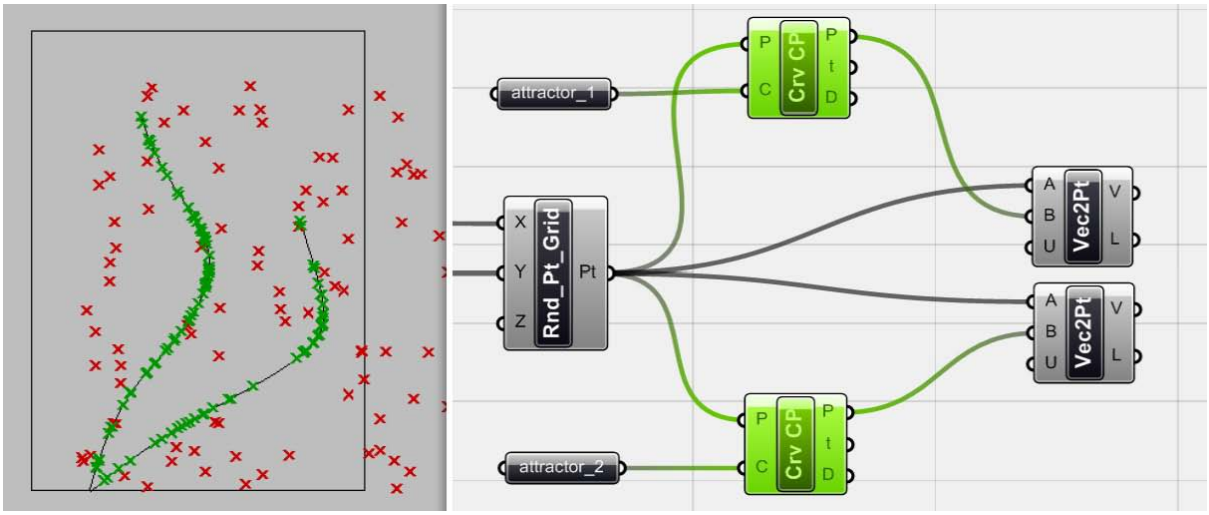


<Random>으로 그려진 점들과 이것들의 끌개로 사용될 <curve> (Params > Geometry> Curve)를 준비한다. <random>한 값이 생기는 범위를 R에 <domain>연결하여 설정해준다. 즉 이것은 점이 생기는 X, Y 좌표의 범위가 된다. 이 경우 A는 0으로 설정되어 있으며 B는 <number slider>를 이용하여 설정해준다. <jitter>는 <random>에서 생성된 값의 순서를 한번 더 섞어 <point>의 X,Y 좌표가 다른 값을 가질 수 있게 해준다.

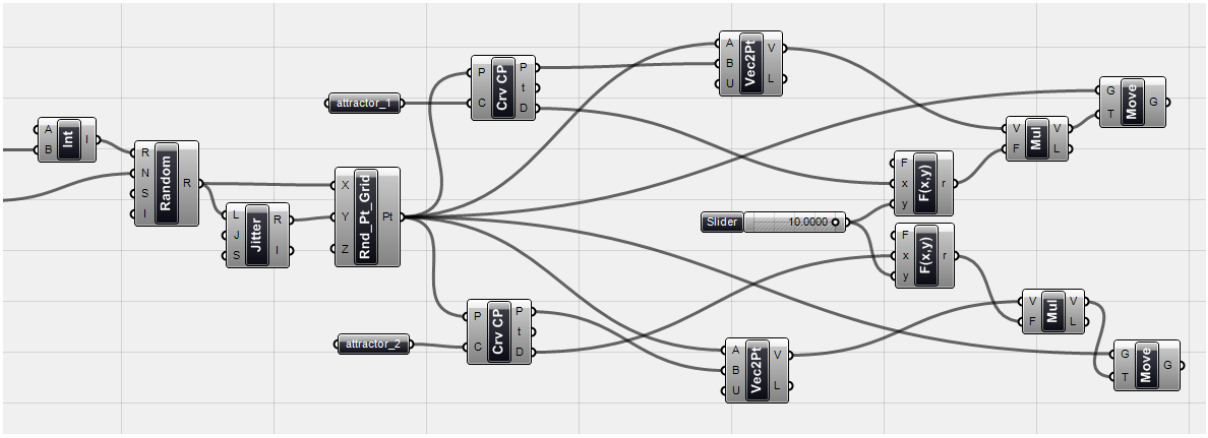


끌개가 점이었을 때는 점들에 생기는 기하체가 끌개점을 향하게만 하면 되었다. 하지만 이것이 curve일 경우에는 먼저 점들이 curve상의 어떠한 위치로 향하게 될 것인지를 설정해주어야 한다. 또한 이 점은 각각의 점들과 1대1의 관계를 가지고 있어야만 한다. 만약 끌개가 자석이라고 가정한다면 이 자석은 점들을 끌개의 가장 가까운 지점으로 잡아 당길 것이다. 이러한 기능을 가진 component 는 바로 <Curve CP>(Curve > Analysis > Curve CP)이다. 즉 어떠한 점과 curve가 있을 때 그 점으로부터 curve상에 있는 가장 가까운 점(closest point)을 찾아주는 것이다. 각각의 끌개와 <Rnd_Pt_Grid>에 <Curve CP>를 연결하여 점들로부터 가장 가까운 끌개상의 점들을 찾아준다.

`



후에 점들 위에 그려질 기하체들이 끌개로 이동하도록 하기 위해서는 점들로부터 각각의 끌개로 향하는 vector를 정의해줄 필요가 있다. <vector 2pt>의 A에 <Rnd_Pt_Grid>의 점들을, 그리고 B에 끌개 위에 있는 점들을 연결하여 준다.[35]



이제 모든 <Rnd_Pt_Grid>를 두개의 <move>에 연결하여 이 점들이 끌개로 이동하게 해준다. 이전 단계에서 사용한 <vector 2pt>를 <move>의 t에 연결해줄 경우, 이것은 점들을 모두 끌개 위로 이동시켜 준다. 이 경우 단순히 점들 위에 생성될 기하체를 그저 끌개 위에 가져다 두는 것이 아닌, 원래 있던 점들과 끌개상에 있는 점들 사이의 거리에 따라 마지막 결과물의 위치가 영향을 받게 하는 것이다. <Curve CP>의 output중 하나인 D가 거리 값을 리턴 해준다.

이를 위하여 <function 2>을 이용하여 함수로 X/Log(Y)를 입력한 뒤 X에 거리 값을, Y에는 <number slider>를 연결한다.[36] (log 함수는 <function2>가 리턴하는 값과 거리 사이의 선형적인

---

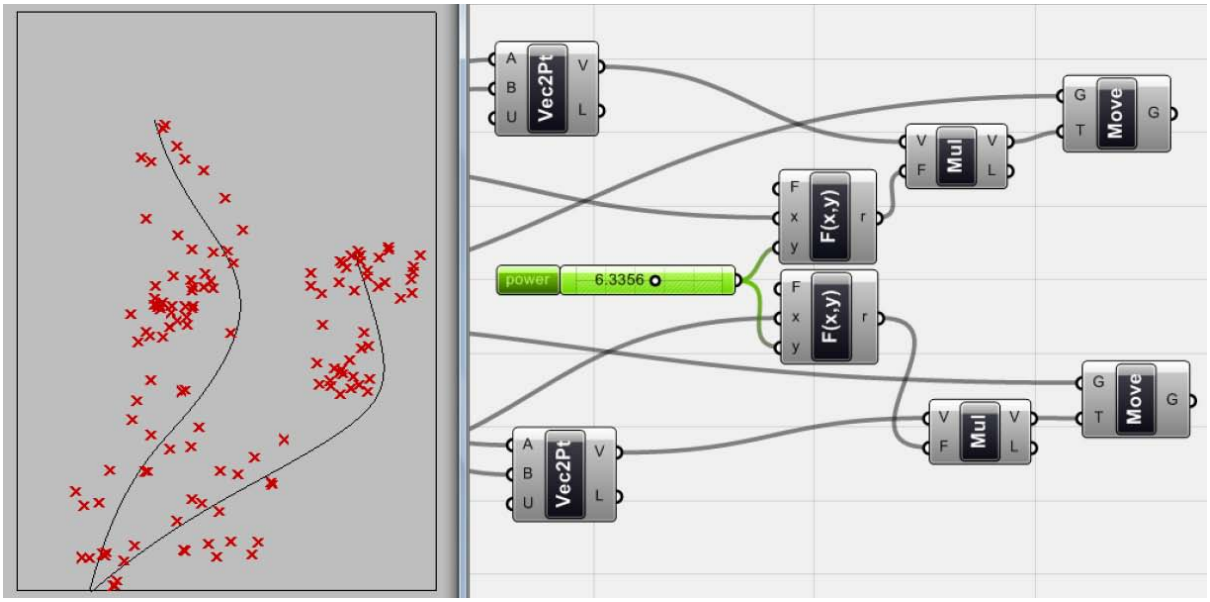
[35] 역자 주: 즉 A가 시작점이고 B가 끝점인 vector를 만들어준 것이다.

[36] 역자 주: 이것이 끼치는 영향은 일반적인 log 의 그래프를 생각해보면 쉽다. y값이 0과 1사이에 있을 경우 <function 2>의 분모인 log(y)가 1과 0사이의 값이 되기 때문에 이는 값을 크게 만든다. 특히 0으로 가까이 갈수록 그 값이 무한하게 커진다. y값이 1이면 log(y)는 0이 되고 이것은 분모가 0이 되기 때문에 성립하지 않는다. 이 때 <function 2>는 빨간색으로 변하게 된다. 1보다 클 경우 y값이 증가 할수록 분모인 log(y)의 증가폭은 줄어들게 된다. 그렇기 때문에 값이 커져도

`

관계를 바꿔주게 된다.)

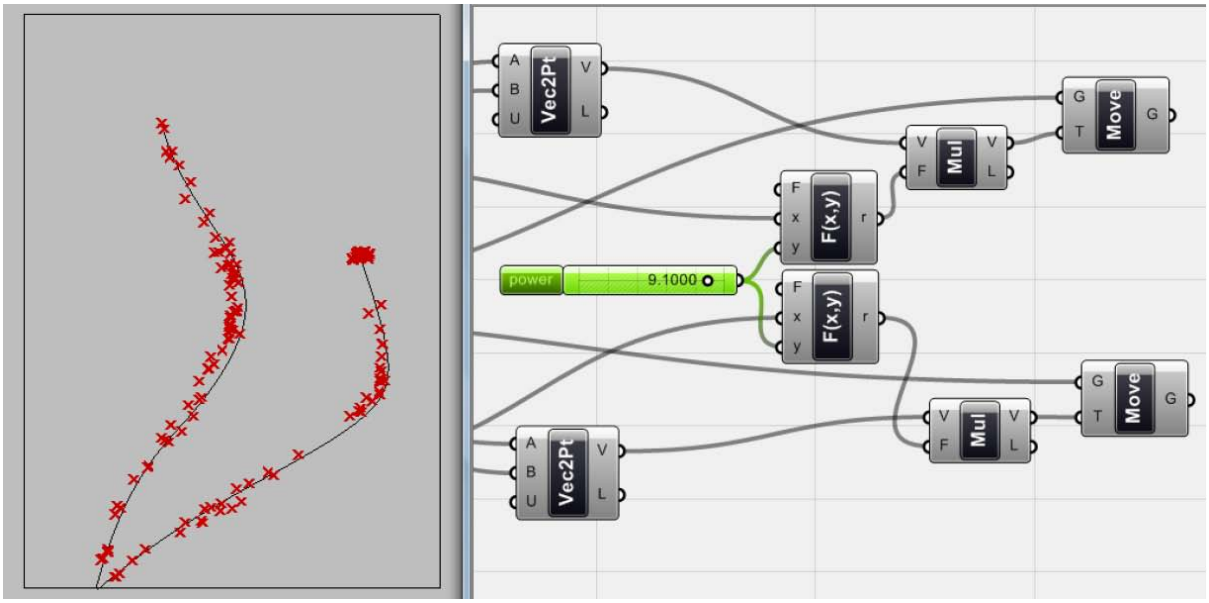
이제 새롭게 만들어진 계수를 이용하여 vector의 크기를 바꿔줄 수 있다. 이를 위하여 <multiply> (Vector > Vector > Multiply)를 이용할 수 있다.[37] 이것은 input으로 들어오는 vector의 크기에 factor 를 곱해주고 그 결과물인 vector를 output의 V로 내보낸다. Y값을 조절함에 따라 <multiply>의 F값을 조절할 수 있다. 이제 이것을 <move>의 T에 연결해준다. <move>는 기하체들을 끌개를 향하여 이동하게 해줄 것이다.



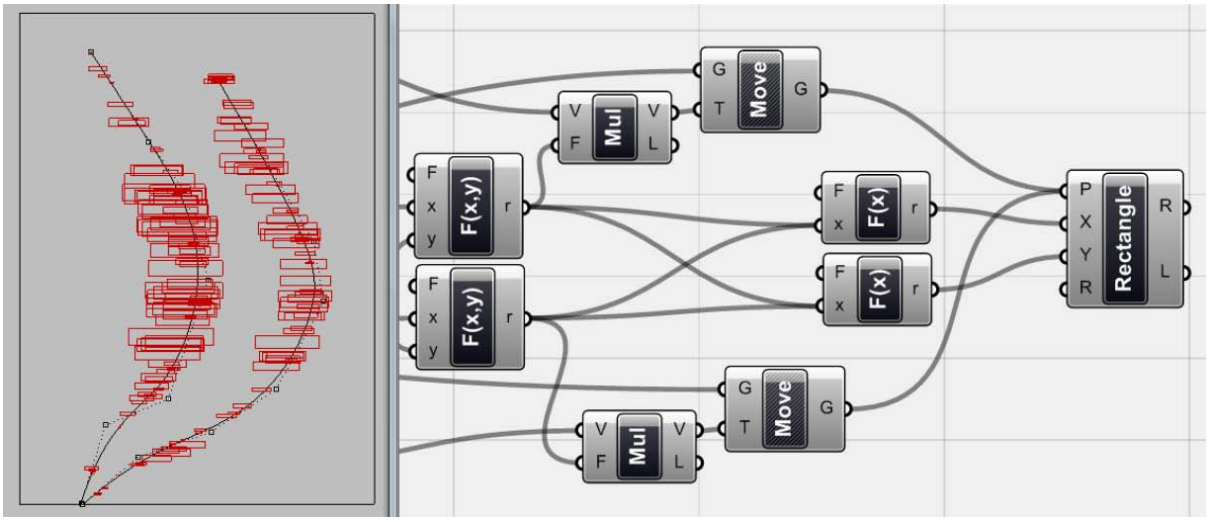
기하체의 이동 폭은 매우 작아진다.

[37] 최근 버전의 grasshopper에서는 해당 component가 사라졌다. 일반적인 수식에 활용되는 곱하기 component <multiplication>을 사용해도 같은 결과를 얻을 수 있다. A에 vector를, B에 <function 2>로 부터 나오는 값을 연결해주면 된다.

`

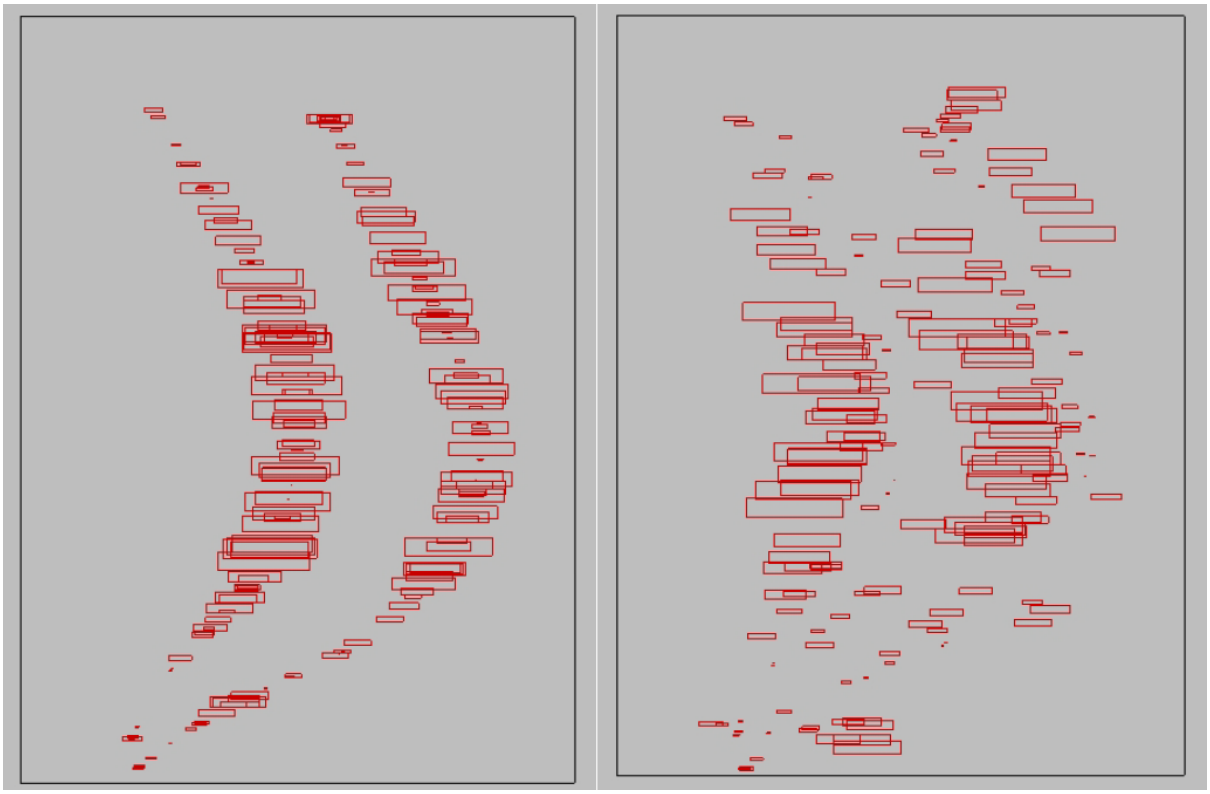


<number slider> 는 끌개의 영향력을 조절하여 객체가 끌개를 향해 이동하는 정도를 나타내게 된다.



이 다음 과정은 바로 직사각형을 생성하는 것이다. <rectangle> (curve > primitive> 을 이용하면 된다. 이것의 input중 하나인 P에 <move>에서 나오는 점을 연결해준다. 이 직사각형의 크기 또한 거리에 영향을 받게 하기 위해서 <rectangle>의 X에 나오는 점들은 모두5로 나누어 주고 Y로 들어가는 값은 25로 나누어 준다. 이제 factor 값을 조절하면 위치와 직사각형의 크기가 바뀌게 된다. 이 5, 25는 각자의 취향에 따라 바꿔줄 수 있다.

`



변수를 변화시킴에 따라 비례와 형태가 변하게 된다. 이것 중에 디자인 목적에 맞는 것을 골라주면 된다.

`

마지막 결과물의 model로 직사각형의 개구부를 가진 벽체와 그것이 바닥에 드리운 그림자 이다. 원하는 그림자의 형태에 맞게 factor값을 조절해주면 된다.

`

Chapter_5_Parametric Space

`

Chapter_ 5_Parametric Space

우리가 다루고 있는 공간 속의 객체에 대한 기하학적 탐구를 통하여 디지털을 이용한 형태와 구법(tectonic)의 재현과 기하학적 요소들의 분절과 결합 그리고 형태를 생성시키는데 필요한 다양한 방식들을 살펴볼 것이다. 이것은 대칭에 대한 고전적인 관념부터 NURBS와 Mesh에 이르는 패턴까지 다양하다.

우리는 객체(object)를 다루게 된다. 이러한 객체의 종류는 박스(box), 구(sphere), 고깔(cone), 곡선(curve), 면(surface) 그리고 이것들에 의해 생성되는 모든 것을 말한다. 공간 내에서 그것들이 존재하는 방식에 대해서 설명하자면, 점(point)는 0-dimensional, 곡선(curve)는 1-dimensional, 곡면(surface)[38]는 2-dimensional, 그리고 입방체(Solid)는 3-dimensional한 객체이다.

우리는 좌표계를 이용하여 공간 내에서 객체의 위치와 방향, 그리고 치수등의 기본적인 사항을 표현할 것이다, 데카르트 좌표(Cartesian coordinate)은 3차원의 공간으로 원점 O=(0,0,0) 그리고 이 점을 X,Y,Z 방향으로 가로지르는 세 개의 축을 가지고 있다. 이 3차원 공간좌표는 2차(평평한 공간flat space (x, y)) and one-dimension (선형공간;linear space (x)) 와 1차 공간 좌표 또한 표함한다. 우리가 매개변수를 이용한 design (parametric design)이란 이처럼 자유로운 형태를 가지는 곡선과 면에는 어떠한 매개변수가 있고 이것을 어떻게 이용하여 design을 할 수 있는지를 살펴보도록 하겠다.

5_1_1차 매개변수 (One Dimensional (1D) Parametric Space)

X축이란 무한한 선으로 축 내부의 위치에 따른 수치 값을 가지게 된다. 간단하게 말해서 x=0 은 원점을 의미하고 x=2.35 란 양의 방향으로 2.35 unit 만큼 떨어져 있는 것을 의미한다. 이러한 1차원 좌표는 공간내의 어떠한 곡선에도 적용시킬 수 있다. X축이 축 위에 있는 한 지점의 위치를 실수(real number)로 표현하는 것처럼, 곡선 위에서도 여러 실수를 이용하여 각 위치상의 점을 표현할 수 있다. 즉 공간에서 1차원의 매개변수가 의미하는 것은 바로 점(point)이다. 이것의 위치는 실수로 표현될 수 있다.

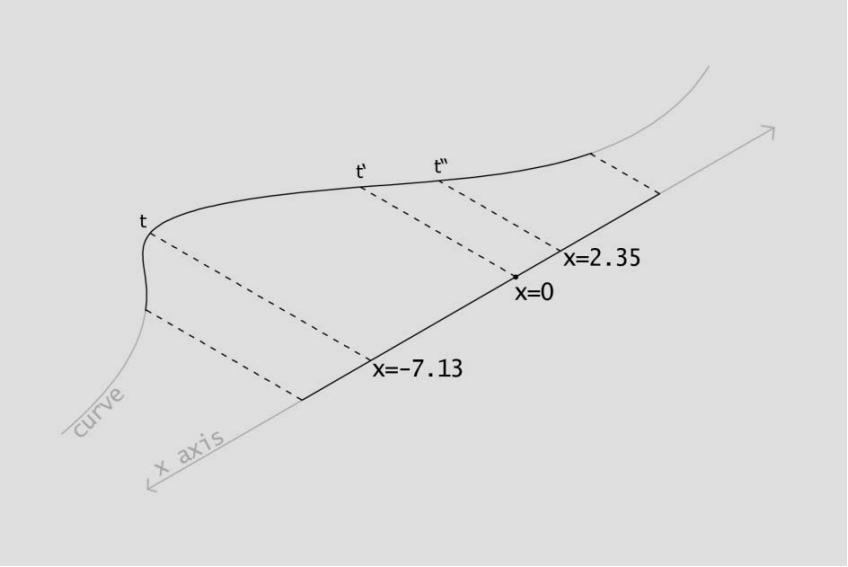
우리가 다루게 되는 것은 단지 X축만을 가진 계(world)가 아니라는 것이다. 모든 곡선이 고유의 매개변수를 가지고 있으며 이것은 그 객체가 속한 데카르트 좌표계(universal measurement)의 그것과 명확하게 일치하지 않는다. Grasshopper에서 사용되는 매개변수는 0으로부터 시작하여 양의 정수로 끝나게 된다. [39]

---

[38] 역자 주: 곡선과 곡면으로 번역하였지만 넓은 의미에서 선과 면도 포함한다.

[39] 역자 주: Grasshopper 에서 <evaluate curve>에서 t에 수치를 입력해주면 바로 그 수치만큼의 길이에 해당하는 점을 찾아주게 된다. 이 때 C를 우클릭하여 reparametize를 해주면 점의 시작점은 t=0일 때, 끝점은 t=1 로 바뀌게 된다. 그리고 곡선 위의 점은 0과 1사이의 실수로 표현할 수

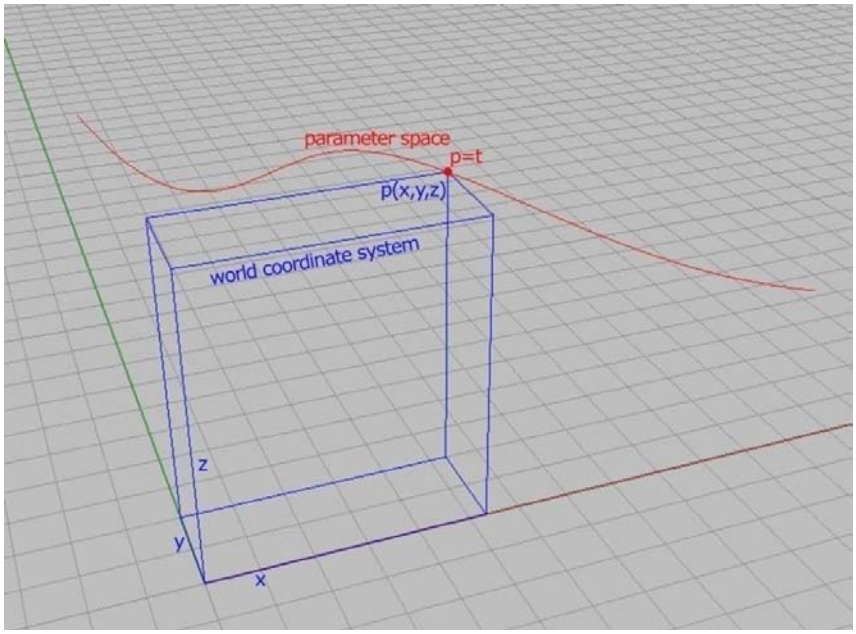
`



곡선 위에 있는 1차원 매개변수. 곡선위에 있는 점은 양수 t를 이용하여 찾을 수 있다

.

아래 그림을 살펴보면 더욱 명확하다. 곡선 위의 점을 p를 그것의 공간좌표인 p(x,y,z)로 정의하는 것이 아니라 p=t 라는 특정한 매개변수를 이용하여 정의하는 것이다. 물론 이 점은 공간좌표로 p(x,y,z)로 치환이 가능하다.



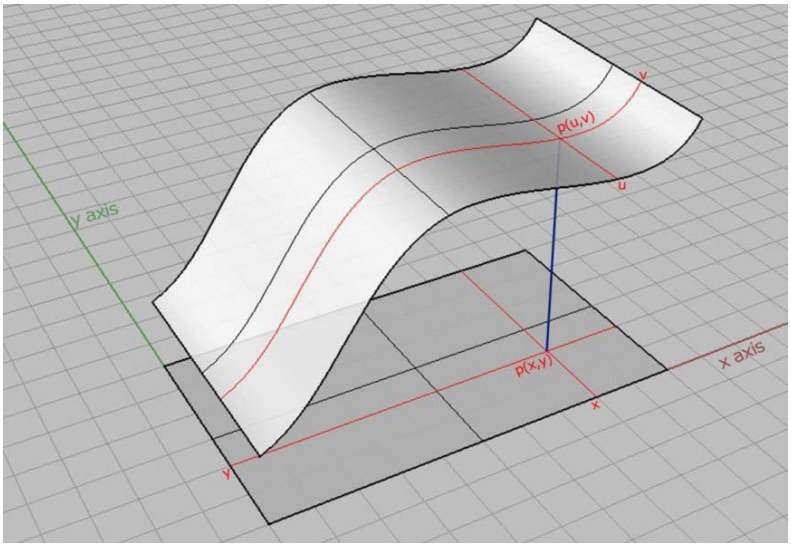
1차원 공간좌표t와 기존의 3차원 공간좌표의 비교

있게 된다. 이 값은 곡선 내부에 독립적으로 존재하는 매개변수로 곡선이 속한 데카르트 좌표계의 좌표 값과 상관이 없다. 이것은 NURBS의 개념으로 연결된다. 더욱 자세한 내용은 Essential Mathematics for Computational Design을 참고.

`

5_2_2차 매개변수 (Two Dimensional (2D) Parametric Space)

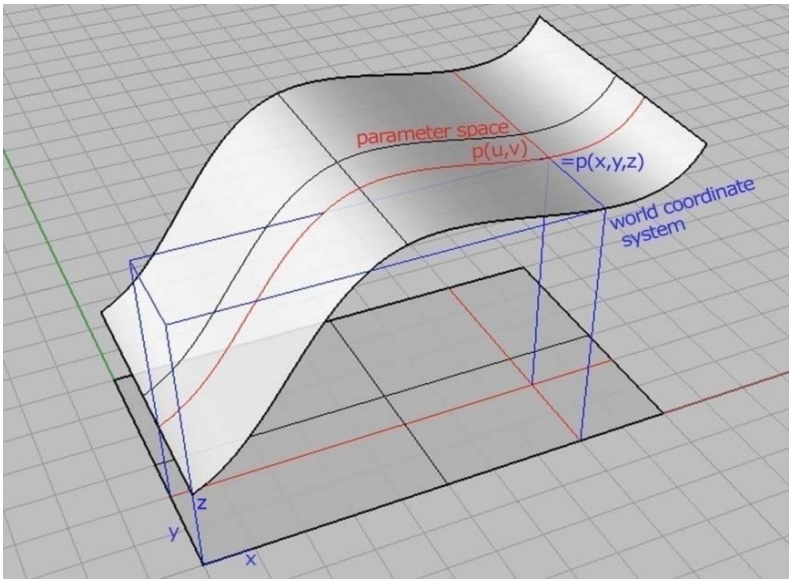
두 개의 축을 가진 공간좌표는 무한한 평면(infinite flat surface)과 같다. 이 평면 위의 모든 점은 p(x,y)라는 좌표로 표현된다. 2차 매개변수 또한 1차 매개변수와 마찬가지 이다. 어떠한 면 위의 점을 정의할 때 그 면이 속한 좌표계 내에서의 좌표뿐만 면 고유의 매개변수로 표현해줄 수 있다.
즉 곡면이 있다고 했을 때 이 곡면상에 U와 V라는 두 방향으로 가상의 선들이 있다고 보고 선들의 교차점을 이용하여 점을 정의하는 것이다. 이렇게 되면 점은 P=(U,V)로 표현될 수 있다.
즉 어떤 면 위의 점을 면이 속한 데카르트 좌표계의 좌표뿐만 면 고유의 UV 매개변수(parameter)로 표현이 가능하다.



UV와 2d 매개변수

이러한 매개변수(parameters)들은 각각의 면 위에 고유 값으로 존재하며 데카르트 좌표 시스템처럼 포괄적인 (generic) data가 아니다. 그렇기 때문에 이것을 우리는 Parametric이라고 부를 수 있다. 이를 이용하면 면 위의 어떠한 점도 정의할 수 있다.

`



Point P=(U,V)는 좌표계의 p=(X,Y,Z)와 같은 점을 정의한다.

5_3_매개변수와 데카르트좌표, 고유번호 (Transition between spaces)

parametric design을 하기 위해서는 고유의 매개변수(parameter)뿐만 아니라 데카르트 좌표 (the world coordinate system)를 모두 이용할 수 있어야 한다. 객체를 생성하거나 이를 전환(transformation)[40]시키기 위해서는 객체 고유의 매개변수뿐만 아니라 데카르트 좌표정보 모두가 필요하기 때문이다. 이러한 전환과 이용이 Scripting에서는 더욱 복잡하나 grasshopper의 경우 scripting 처럼 코드가 아닌 시각화된 인터페이스를 가지고 있기 때문에 어떠한 기능을 하는데 매개변수가 필요한지 데카르트 좌표가 필요한지를 쉽게 알 수 있다.

또 한가지 이용할 수 있는 것은 data의 고유 번호(index number)이다. 만약 여러 개의 data나 점, 선, 면등의 객체를 하나의 data list에 넣어야 할 경우 각각의 것에 index number를 부여된다. 즉 객체를 정의하고 이것을 이용하기 위해서는 그 객체가 data list 내에서 가진 index number로 객체를 식별해줄 수 있는 것이다. 이 index number는 0부터 시작한다.

grasshopper를 이용하여 design을 하게 되는 위 세가지 정보를 모두 이용할 수 있어야 한다.

---

[40] 역자 주: 변환에 대한 자세한 내용은

아핀변환: http://geometricmind.wordpress.com/2010/11/24/essentail-mathematics-for-computational-design-15/

회전변환: http://geometricmind.wordpress.com/2010/11/26/essential-mathematics-for-computational-design-16/

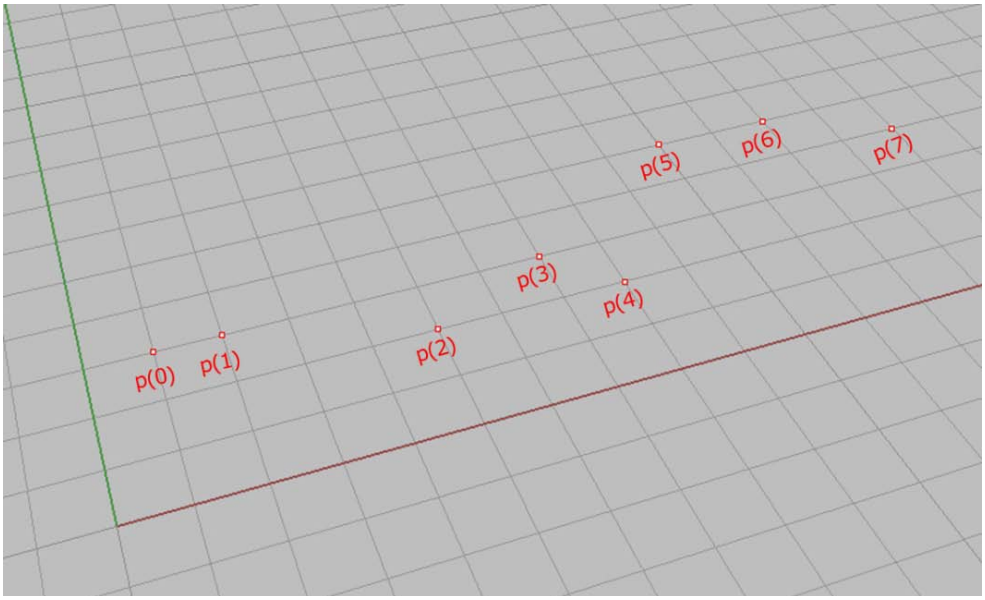
스케일 변환: http://geometricmind.wordpress.com/2010/11/28/essential-mathematics-for-computational-design-17/

전단 변환: http://geometricmind.wordpress.com/2010/11/29/essential-mathematics-for-computational-design-18/

평면투영변환: http://geometricmind.wordpress.com/2010/11/29/essential-mathematics-for-computational-design-19/

참고

`



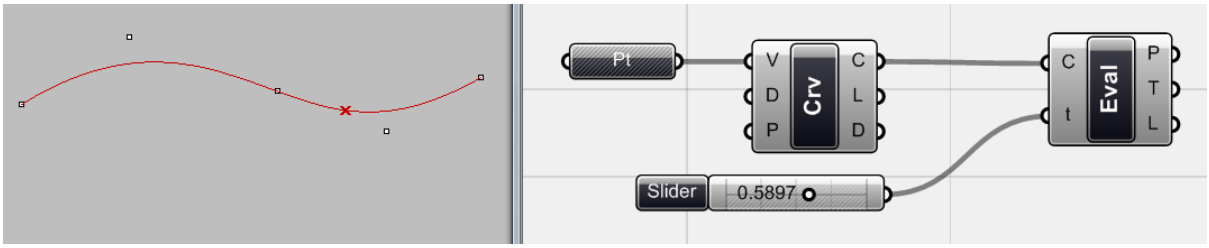
여러 객체의 집합이 있을 경우 Index number 를 주면 그것을 불러 사용하기 훨씬 편하다. 이러한 index number 는 0 부터 시작한다.

`

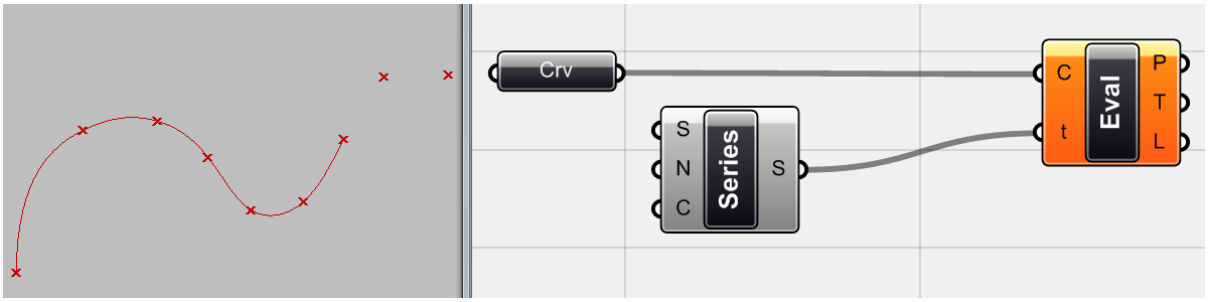
5_4_Basic Parametric Components

5_4_1_Curve Evaluation

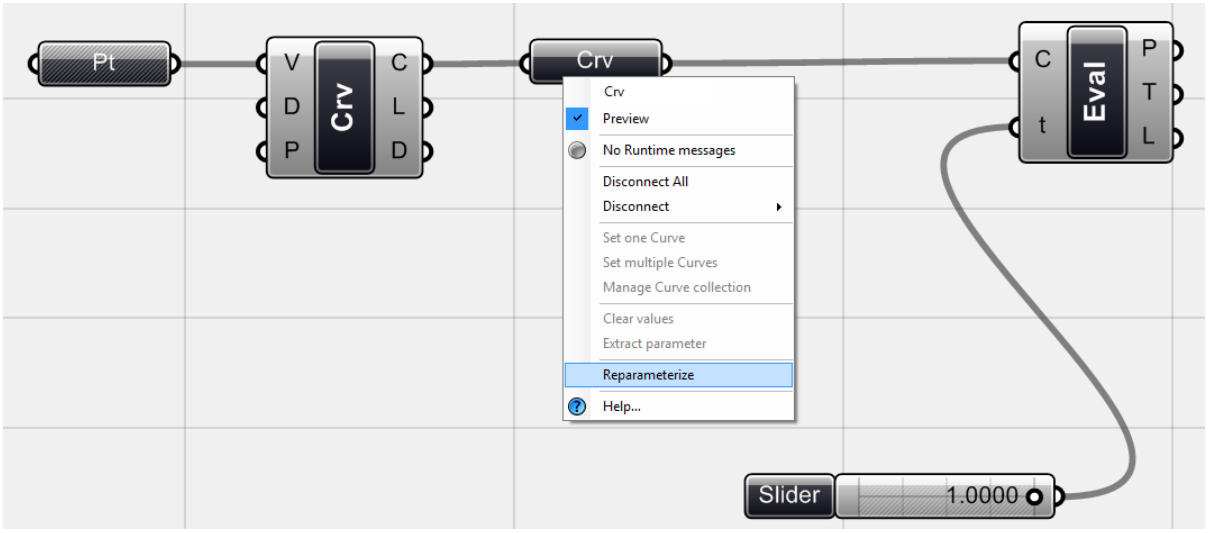
<evaluate curve> (Curve > Analysis > Evaluate curve) 는 input 인 T에 주어진 변수로 정의되는 curve위의 점을 찾아주는 것이다. input으로는 curve와 숫자인 parameter 이다.



<curve>위에 표시된 점은 <number slider>를 매개변수로 하는 점이다.



Rhino 상에 그려진 어떠한 curve라도 그 input으로 사용할 수 있으며 위 경우 <series>를 이용하여 복수개의 수를 가진 data list를 input으로 이용하여 각 수를 매개변수로 가지는 점을 찾아내었다. 여기서 <evaluate curve>가 오렌지색인 이유는 <series>에서 나온 값이 curve가 가진 영역을 넘기 때문이다. curve의 영역을 넘는 매개변수는 curve를 가상으로 연장하여 그 위에 점을 표시해준다.



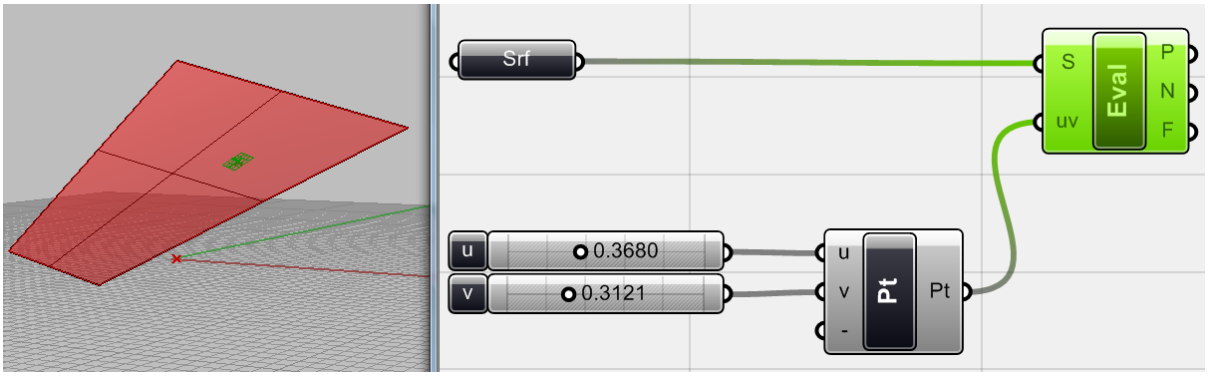
<curve>의 output인 D는 curve의 영역 (영역 내의 최소값과 최대값)을 표시해준다. 위의 그림에서 처럼 <curve>를 우클릭 한 뒤에 reparameterize를 선택하면 curve의 영역이 0에서 1로 바뀌

`

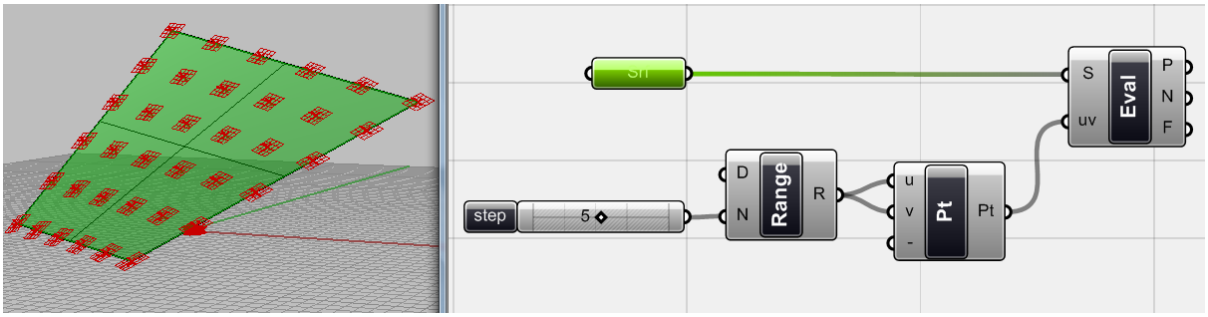
게 된다. 이제 0부터 1까지를 정의한 <number slider>를 이용하여 curve상의 모든 점을 찾아줄 수 있다. 이렇게 하면 사용자가 점의 위치를 예측하기 쉬우며 input으로 들어가는 수가 curve의 영역을 넘는 것을 방지할 수 있다.
비슷하면서도 유용한 component로는 <divide curve> (Curves > Analysis > Division)가 있으며 이것에 대해서는 다음에 살펴볼 것이다.

5_4_2_Surface Evaluation

Curve의 점을 정의하기 위해 사용되는 매개변수는 하나이다. 반면 surface 상의 점을 정의할 경우에는 U와 V 각각에 해당하는 값이 필요하다. <evaluate surface> (Surface > Analysis > Analysis)를 이용하면 면 위의 한 점을 정의할 수가 있다.
<point> 를 이용하면 X를 U값으로, Y를 V 값으로 인식하여 점을 찾아줄 수 있다. Z 값은 무시된다.

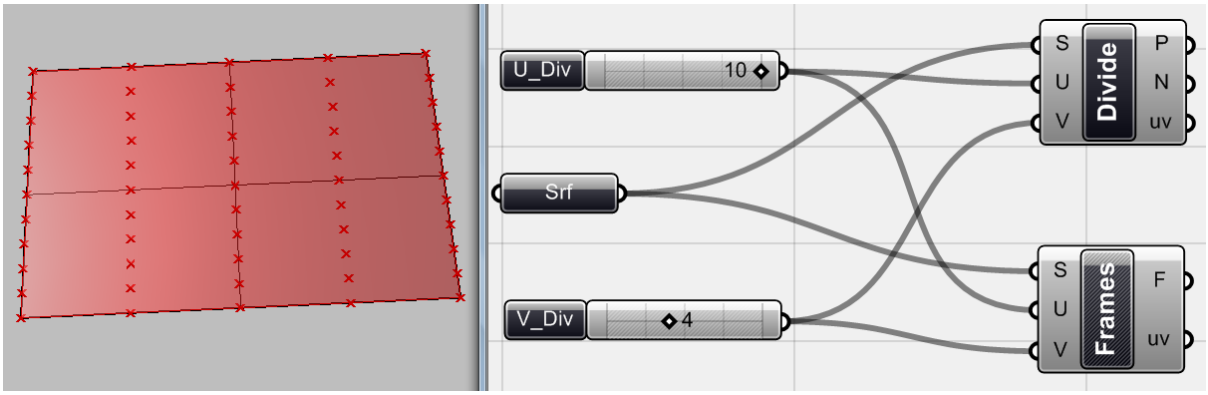


UV값은 <number slider>로 input해줄 수 있고 surface를 S를 우클릭하고 context menu에서 reparameterize를 선택해주면 0과 1사이의 값으로 점을 찾아줄 수 있다. U와 V값을 바꿔가며 점의 위치를 바꿔보자. (여기서 X,Y,Z를 각각 U,V,-로 바꿔주었다.)



위의 경우 <point>를 이용하여 UV 값을 넣어주었지만 그 외에도 여러 가지 방법으로 복수개의 점을 찾아줄 수 있다. <

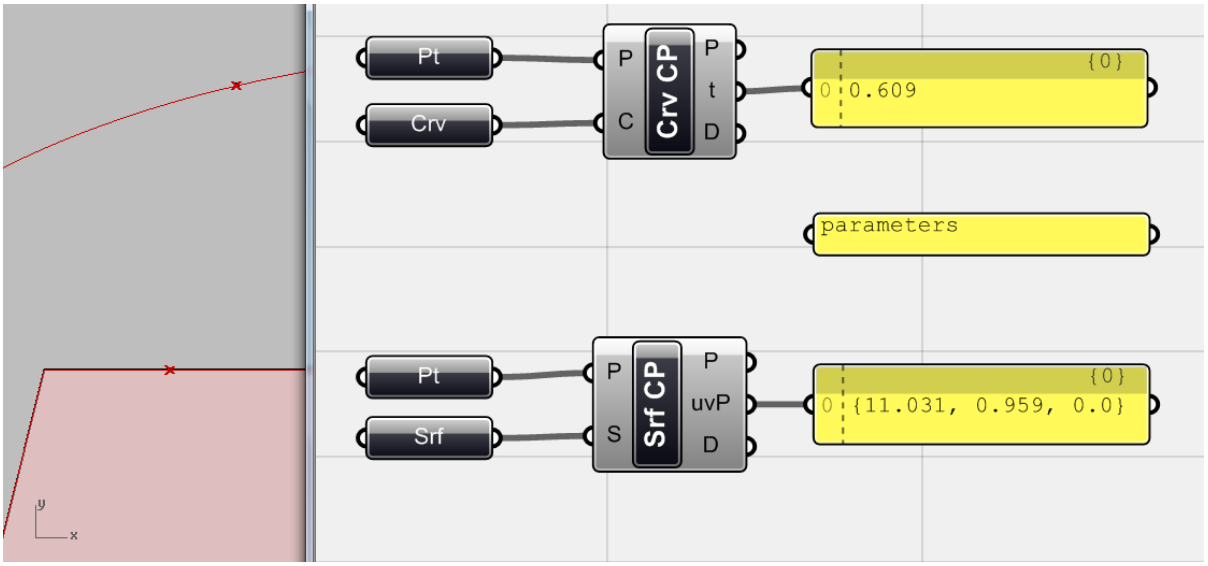
`



Surface를 uv 방향으로 원하는 수만큼 가상으로 나눈 뒤 각 교차점을 찾기 위해서는 <divide surface>를 이용할 수 있다. <frame>은 그 점을 원점으로 하는 평면(plane)을 정의해줄 수 있다.

5_4_3_Curve and Surface Closest Point

Curve나 surface 상의 점을 찾기 위해서 항상 parameter를 이용해야 하는 것은 아니다. 가끔은 이 t나 uv의 값을 알아낸 뒤 다음단계에 이 data를 이용해야 할 수 있다. Curve와 surface가 있을 때 어떤 점으로부터 각 객체 상의 가장 가까운 점을 찾고자 할 때는 <Curve CP>나 <surface CP> (curve/surface closest point)를 이용할 수 있다.



<Curve Cp>와 <Surface Cp>는 curve나 surface 위의 점을 찾는데 유용하다.

5_5_On Object Proliferation in Parametric Space

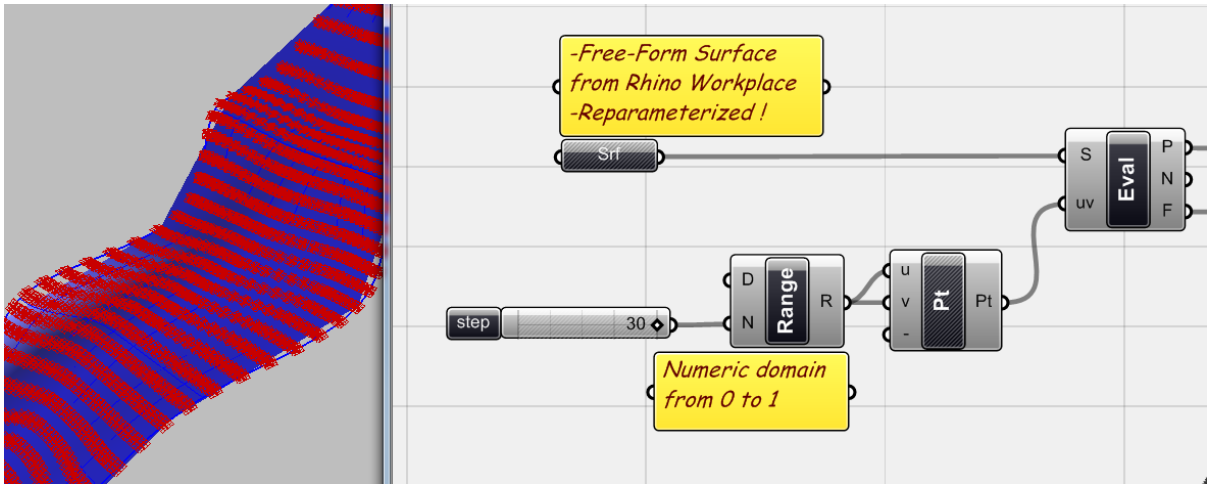
Rhino에서 그려진 Surface는 Grasshoppper의 input으로서 기하체들을 그 위에 늘어놓는데 사용될 수 있다. 먼저 이러한 surface를 panel화 시키는 것이 가능하다. 혹은 기하체들을 surface 상의 특정 위치에 늘어놓을 수 있다. Rhino에서 Surface의 형상을 바꾸면 이것이 grasshopper definition에 반영되고 다시 이것이 rhino상의 preview에 반영된다. Surface를 input으로 사용할 수 있는 방법은 많이 있는데 간단한 예시를 통하여 이를 살펴보도록 하자.

`

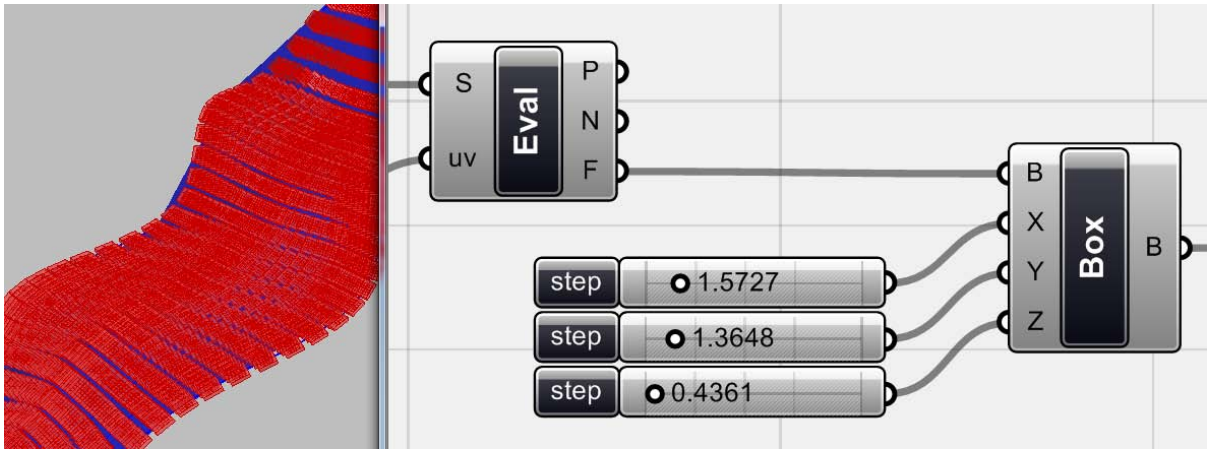
자유로운 형태를 가진 surface와 box가 있다. 어떻게 하면 box를 surface 위에 늘어놓을 수 있는지 살펴보자.

먼저 surface를 원하는 수로 나눈 뒤 각각의 나뉘어진 surface 위에 box를 생성하고 이것을 조절해주는 것이다. 즉, 전체적인 형태(Macro scale)를 조율하기 위해서는 surface의 형상을 바꿀 수 있고 국지적으로 (micro scale) 박스의 크기를 바꾸거나 회전시킬 수 있다.

Surface 위에 원하는 위치를 생성하는 것은 매우 쉽다. 우리가 원하는 수치정보를 입력해주면 그것에 따라 surface를 나눈 뒤 나눠진 지점에 point가 표시된다. 이 점을 기준으로 box를 생성한 뒤 국지적으로 box를 회전시키거나 이동, 혹은 크기 조절을 하게 된다.



원하는 형태의 surface를 rhino에서 그린 뒤 이것을 grasshopper와 연동시킨다. 그리고 이것을 <evaluate surface>에 연결하고 S를 우클릭 하여 reparameterize를 해준다. <range>의 경우 domain 은 0 to 1 이고 N에 <number slider>를 연결하여 원하는 수를 입력해준다. 이렇게 하면 0과 1 사이를 30등분한 결과값이 data list로 생성된다. 이것을 <XYZ point> X와 Y값으로 넣어주면 나오는 그 결과값은 <evaluate surface>에 필요한 uv좌표 값으로 사용될 수 있다. (위 경우 이전 chapter에서 사용한 방식을 그대로 사용하였지만 <divide surface>를 이용할 수 있다.)



<evaluate surface>는 위 UV좌표에 근거하여 생긴 모든 `점(point)들과 그 점들의 '법선(normal)'과 '평면(plane; frame)'값을 output으로 준다. 이제 이 평면에 의하여 생긴 frame들을 <center box> (surface>primitive> center box) 의 중심점 B에 연결해준다. 그리고 중점으로부터의 거리

`

를 X, Y, Z를 <number slider>를 이용하여 정의해줄 수 있다.[41]

이 box들을 국지적으로 조절하여 보자. <rotate>의 input은 '기하체(G)', '각도(A)', '평면(P)' 이다. 회전은 이 평면의 중심점을 기준으로 하게 된다. 이를 위하여 각 점에 <Plane XZ>를 연결하여 각 점을 중심점으로 하는 XY평면을 생성한 뒤 이것을 P값에 연결한다.[42]

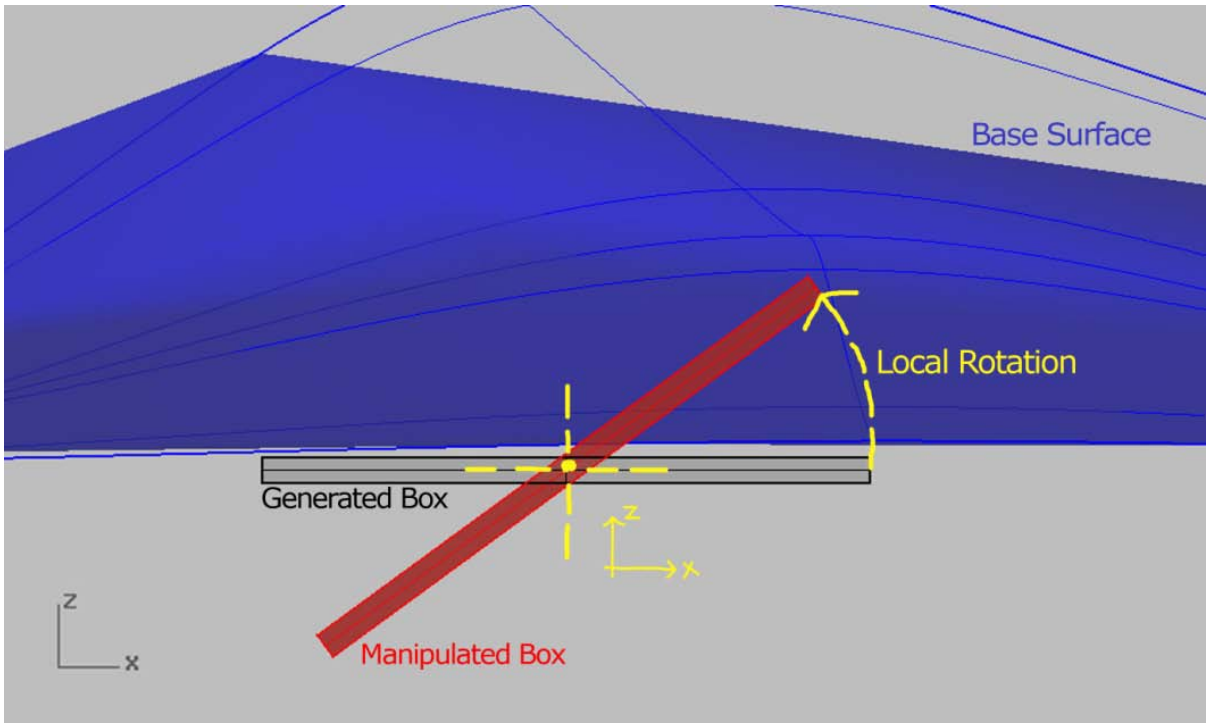
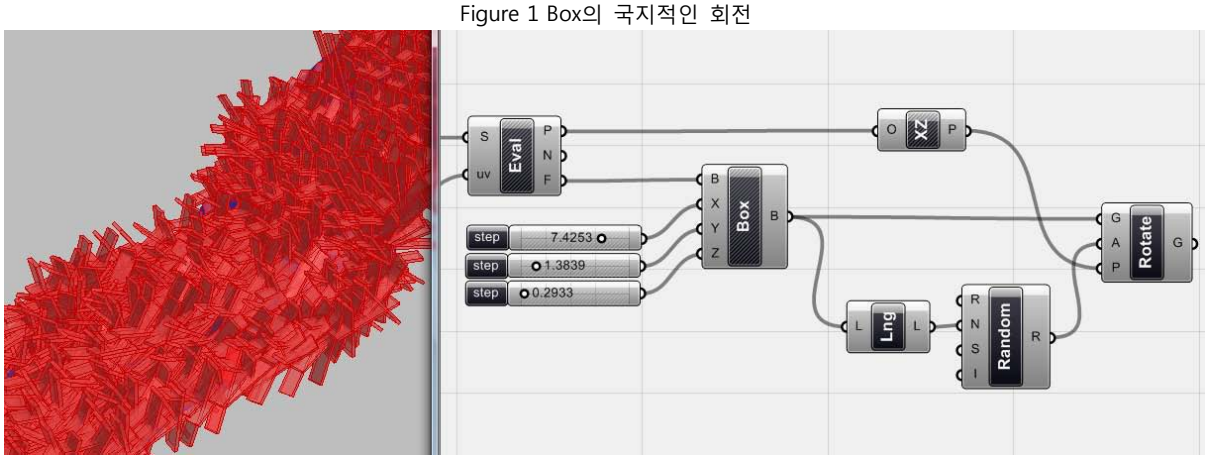


Figure 1 Box의 국지적인 회전

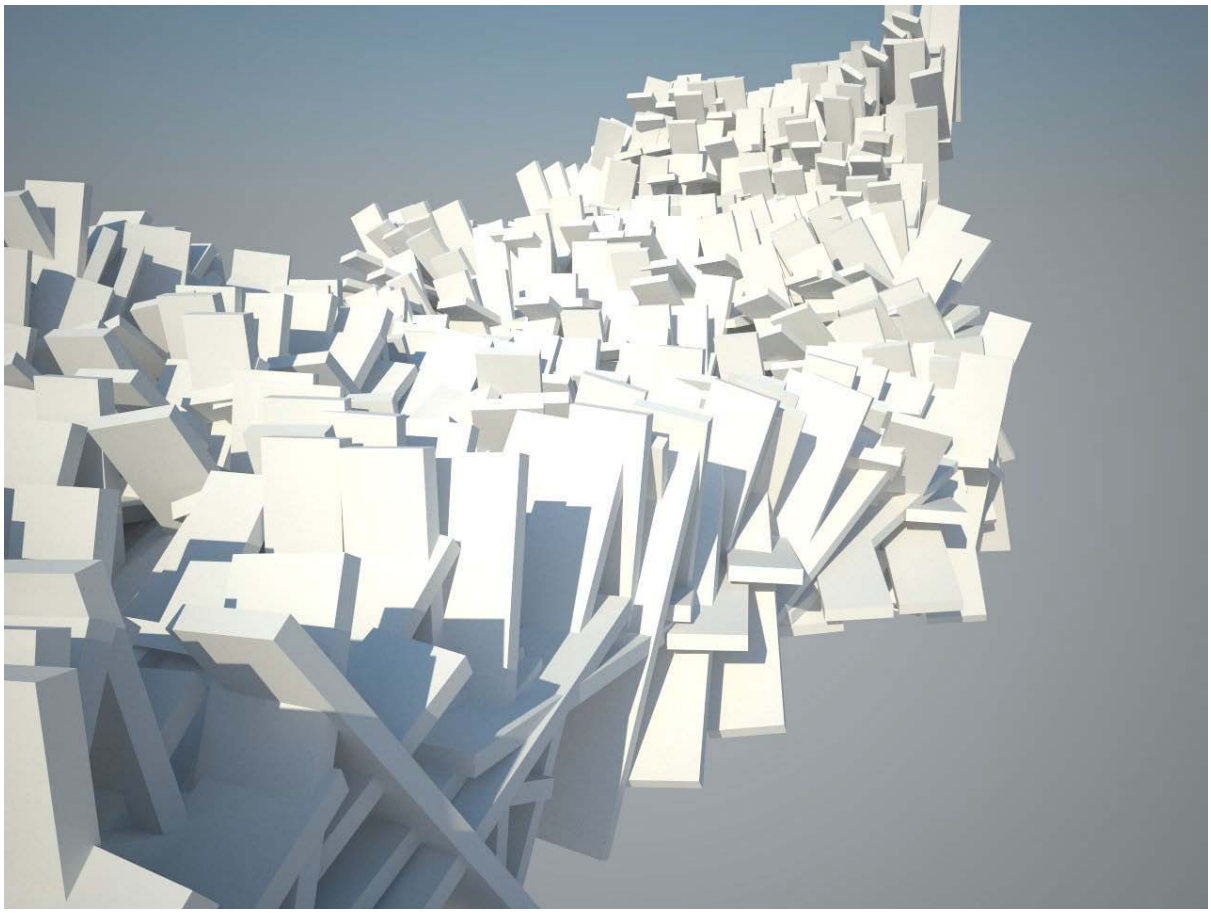


[41] 역자 주: 즉 각 box의 변의 실제 길이는 input값의 두 배가 된다.

[42] 역자 주: 만약 surface 위의 점 p를 그대로 <rotate>의 p에 연결하면 각 점을 원점으로 하는 xy 평면에서 회전이 된다.

`

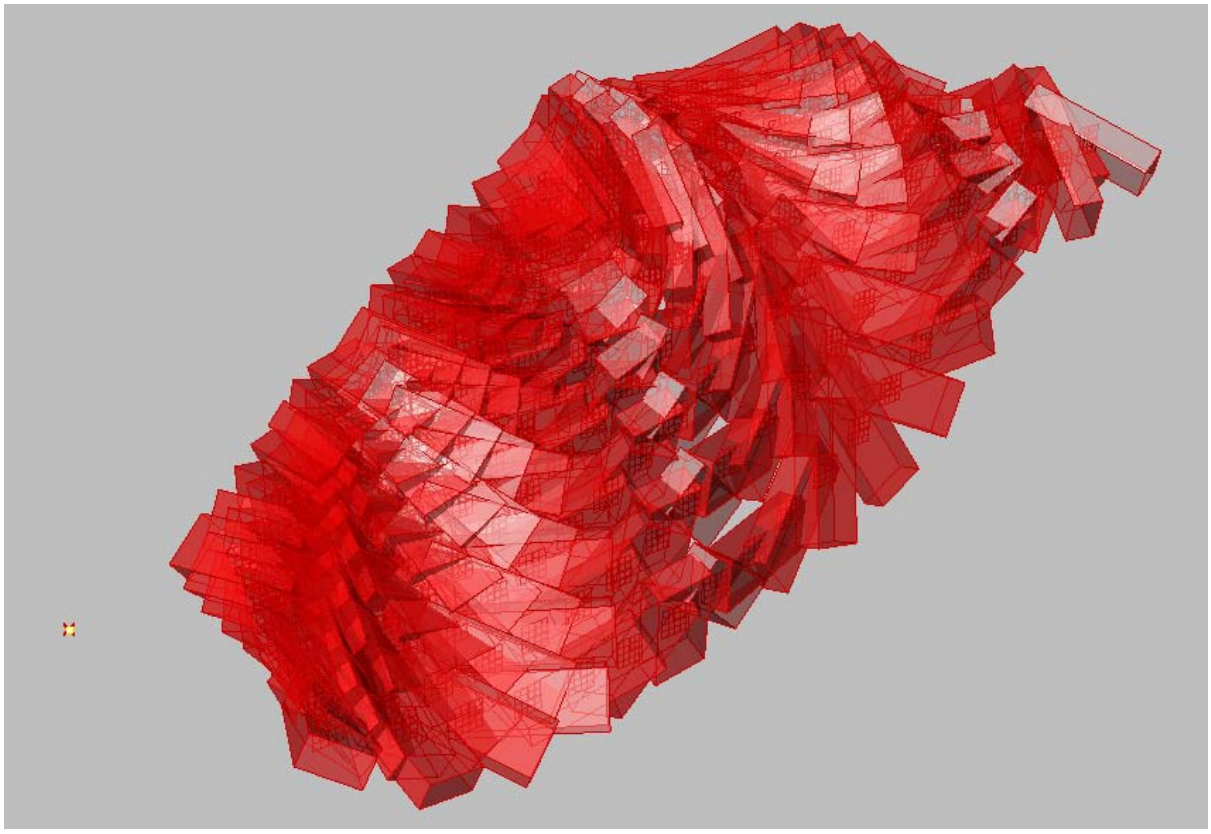
이렇게 하면 box들들 각 점을 중심으로 XZ 방향으로 회전시킬 수 있다. 각도의 경우 <random>을 이용하여 무작위로 생성해준다. 이때 나오는 값들의 수는 box의 개수와 일치하여야 한다. <Box>에 <list length>를 연결하여 Box의 개수를 찾아준 뒤 이것을 <random>의 N에 연결하여 이 개수만큼의 random한 값들을 만들어준다. 이렇게 나온 값을 <rotate>의 A에 연결해준다.[43]



최종 결과물

[43] 역자 주: <random>의 경우 R을 0.0 to 180.0 으로 바꿔주고 <rotate>의 A에는 rad(A)를 입력하여주면 0도와 180도 내에서 box의 개수 만큼의 값을 무작위로 추출해낼 수 있다.

`



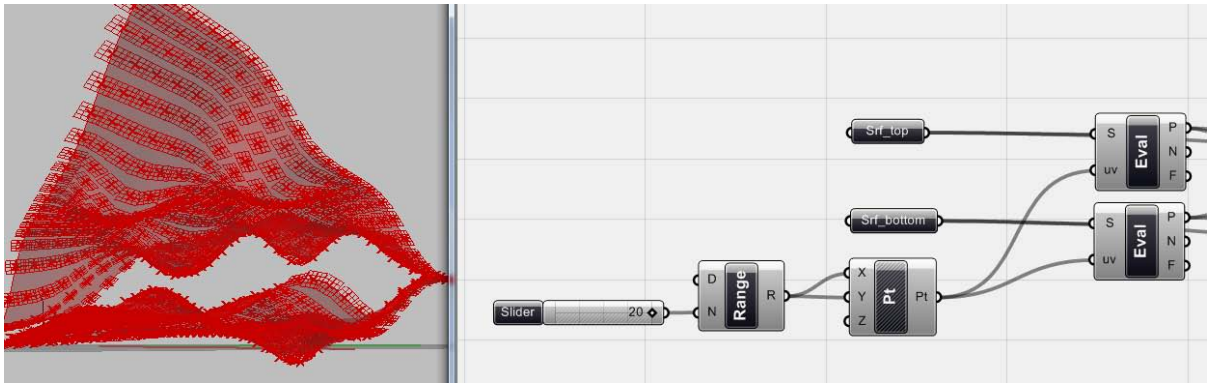
위의 경우 비슷한 logic 에 '각도 값'을 random 이 아닌 '끌개<attractor>'로부터 각 점까지의 거리 값을 각도로 치환한 것이다. 이러한 끌개는 결과물에서 국지적인 회전에 영향을 끼치게 된다. 위에서 배운 것들을 이용하면 위와 같은 결과물을 얻을 수 있다.

Evaluation의 개별적 적용(Non-uniform use of evaluation)

위 예제를 진행하며 든 생각은 어떻게 하면 제 surface 위의 점을 선택적으로 사용해보도록 하자.

Columns Example

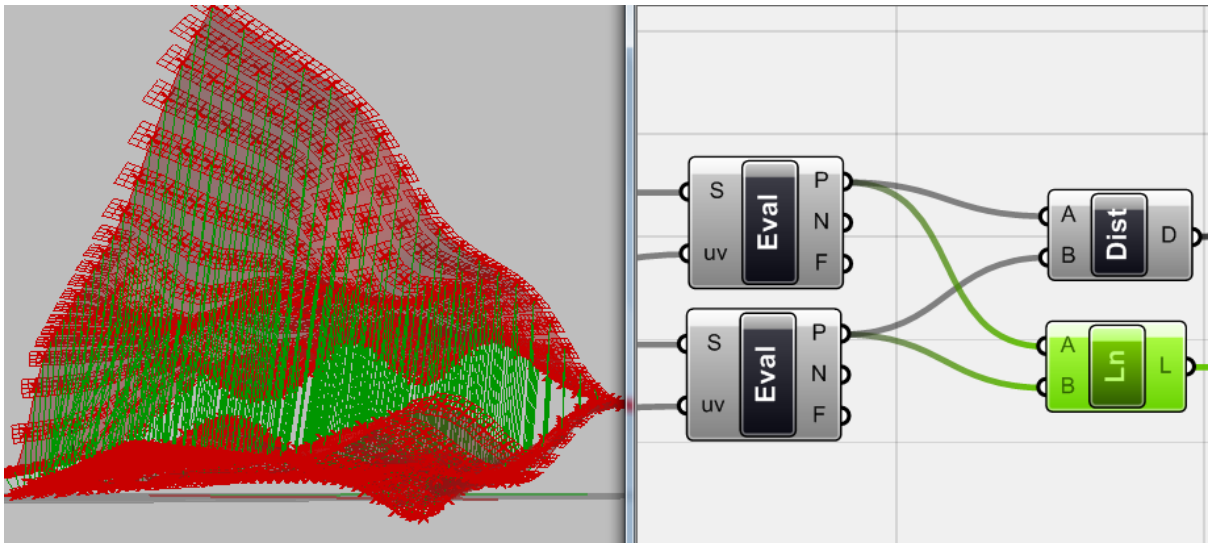
두 개의 자유곡면 사이에 공간을 만들어보자. 이 두 면 사이에 서로 마주보면서 포개지는 원뿔형의 기둥을 배열한다. 이때 기둥의 길이와 위치는 제한이 된다.



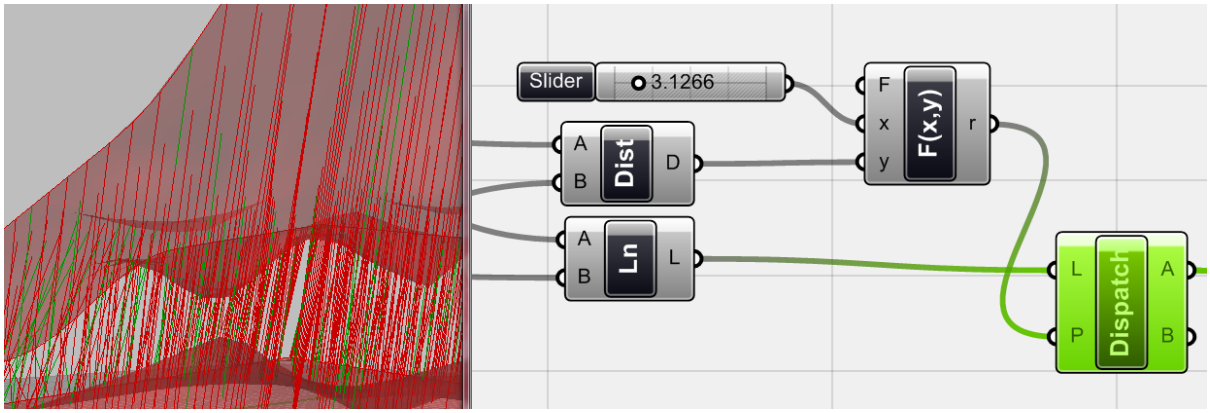
두 surface를 grasshopper의 <surface> parameter에 연동시킨 뒤 각각을 <srf_top> 그리고

`

<srf_bottom>으로 바꾼다. 이 둘을 각각의 <evaluate surface>에 연결한 뒤 reparameterize를 해준다. <range>의 domain을 0.0 to 1.0 으로 정해주고 그 사이를 N등분하여준다. 이것을 <XYZ point>에 input 한 뒤 나오는 점의 좌표값을 <evaluate surface>의 uv 좌표값으로 활용해준다.

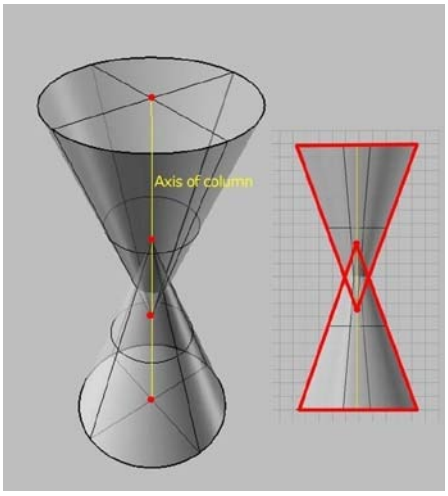


다음으로 이 점들 사이를 <line>을 이용하여 연결해준다. 또한 각 점들 간의 거리도 <distance>를 이용하여 측정해준다.

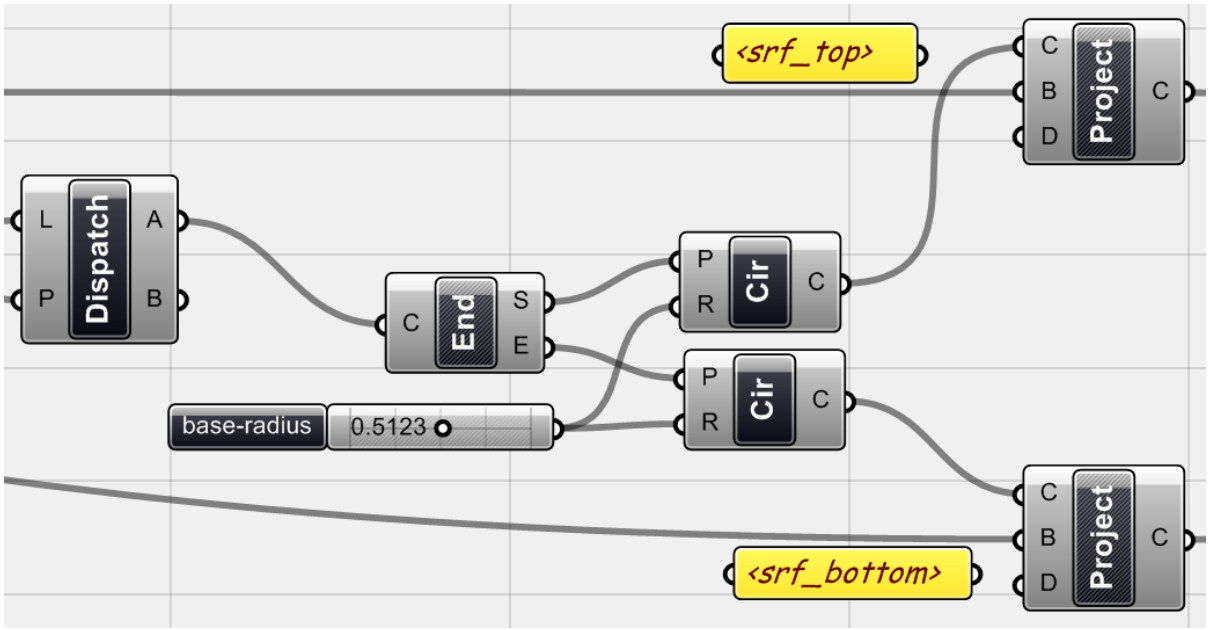


이제 원하는 선을 추출하여야 한다. 여기서 사용되는 것이 <dispatch> (Logic >Llist > Dispatch) 이다. 이 <dispatch>는 Boolean data가 필요하다. 즉 L로 data list를 받고 P로 true 혹은 false 값을 가지는 Boolean data list를 입력하여 둘을 '데이터 매칭'시키게 된다. 이것에 필요한 Boolean data는 간단한 <function> 'x>y' 를 이용하여 얻을 수 있다. 즉 x에 <number slider>가 정의하는 값과 y에 들어오는 <distance>의 y값을 비교하여 이것이 x값보다 작으면 true 즉 'A'로, 그렇지 안으면 false인 'B'로 data를 내보내주게 된다.

`

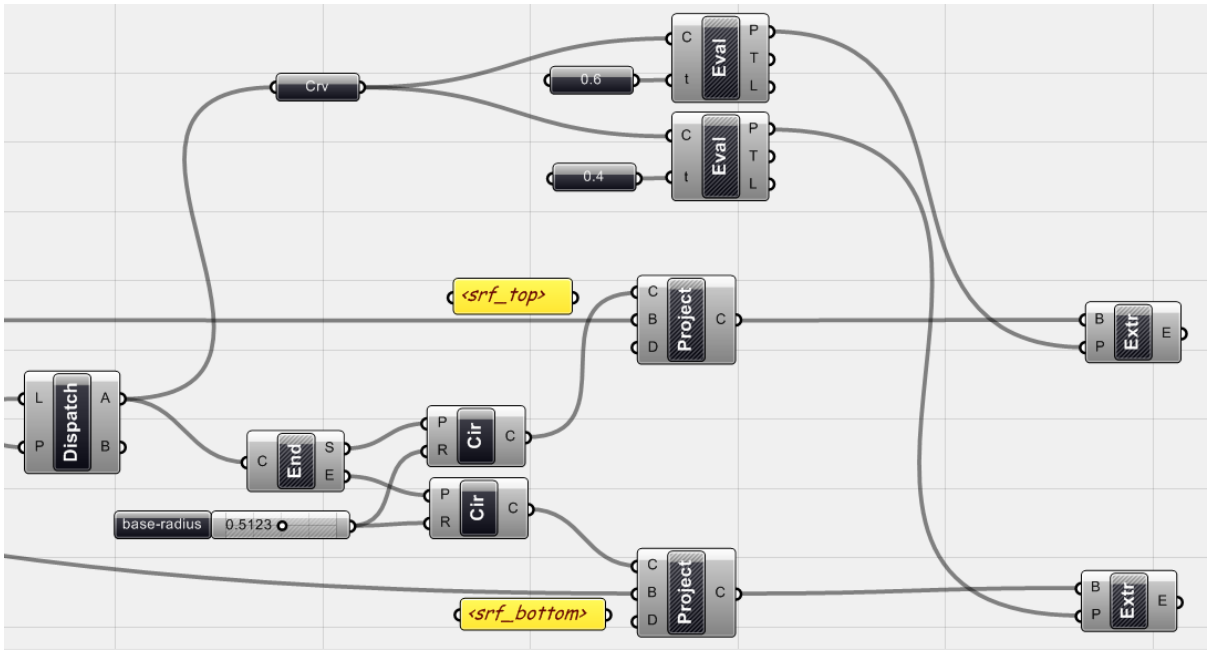


기둥은 바로 두 원뿔이 마주보고 포개어진 형상을 가진다. 이 경우 기둥을 만들기 위한 input으로 활용할 수 있는 것이 바로 축이다. 이 축의 양 끝에 원을 만들고 축 위에서 점을 찾아 <extrude point>에 연결하여 원뿔을 만들어주면 된다.

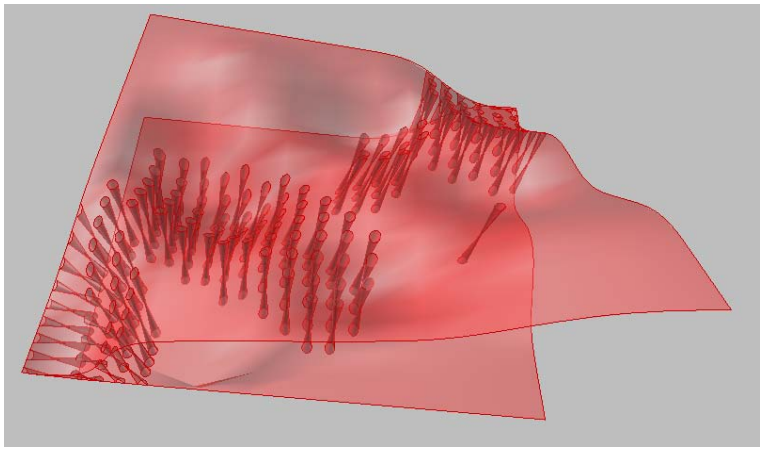


이제 <dispatch>의 A에서 나오는 선을 <end point>를 이용하여 양 끝점을 찾아준 뒤 이 끝에 <circle>을 연결하여 원을 만들어준다. <circle>의 반지름 값은 <number slider>를 이용하여 설정해준다. 이제 이 원들을 면에 <project>(Curve > Util > Project) 를 이용하여 투영시킨다.

`

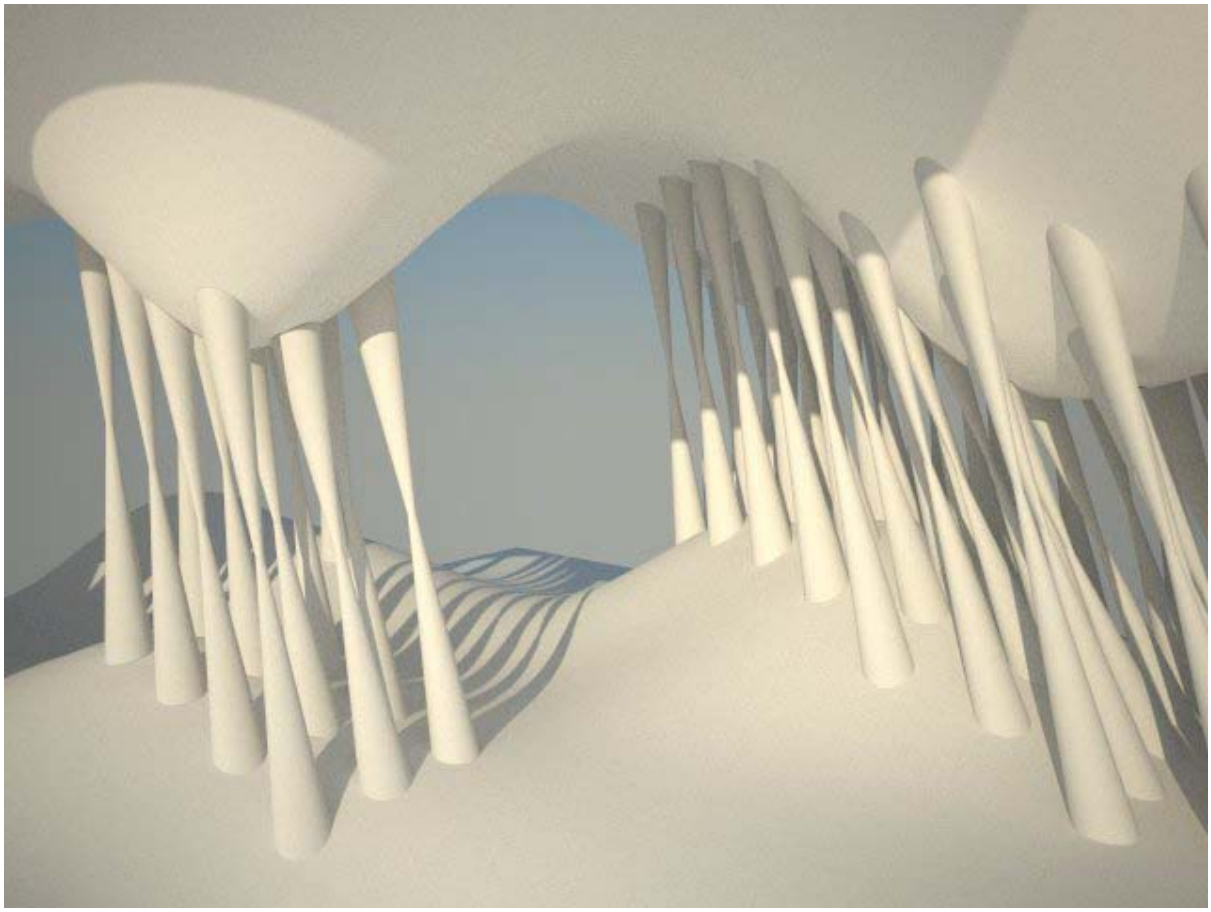


이를 위한 마지막 과정은 바로 <extrude point>를 이용하여 <surface>에 투영된 원의 curve와 점을 연결하여 원뿔을 만들어 주는 것이다. 이 점을 찾아주기 위해서 기둥의 축으로 사용된 line을 <evaluate>하여 reparamaterize한 뒤 0.4와 0.6이라는 parameter를 적용하여 찾아주면 된다.



해당 예시에서 기둥의 축에 사용되는 선을 선택하는 기준이 길이였다면, 전의 끌개 등의 logic을 적용하여 다르게 지정해줄 수 있다.

`



마지막 model

Parameter를 이용하여 design을 하는 것은 바로 여러 가지 대안을 탐구해볼 수 있다는 것에 있다. 위의 기본적인 definition을 다양하게 응용하여 보자.

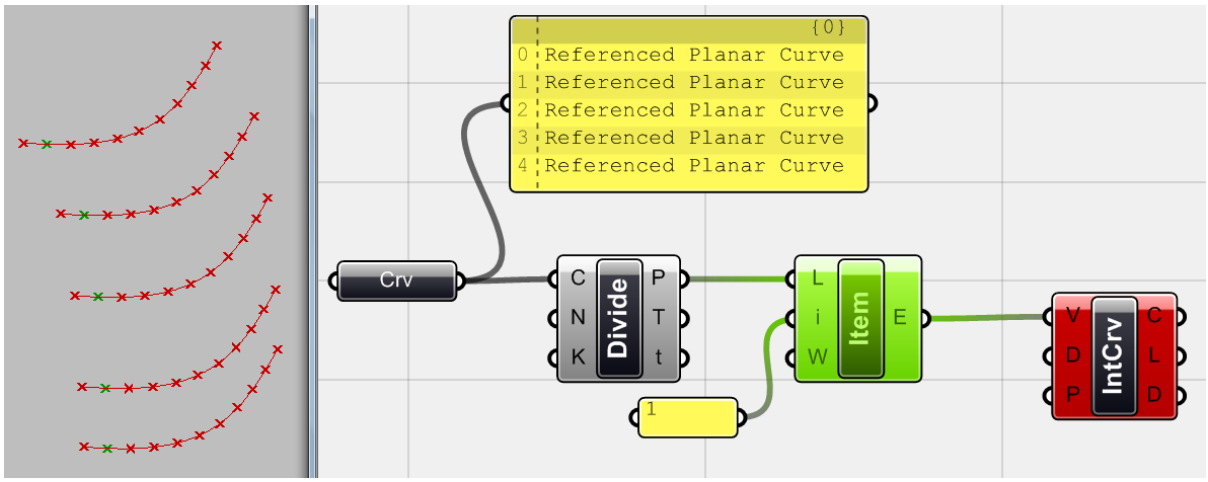
`

5_6_On Data Trees

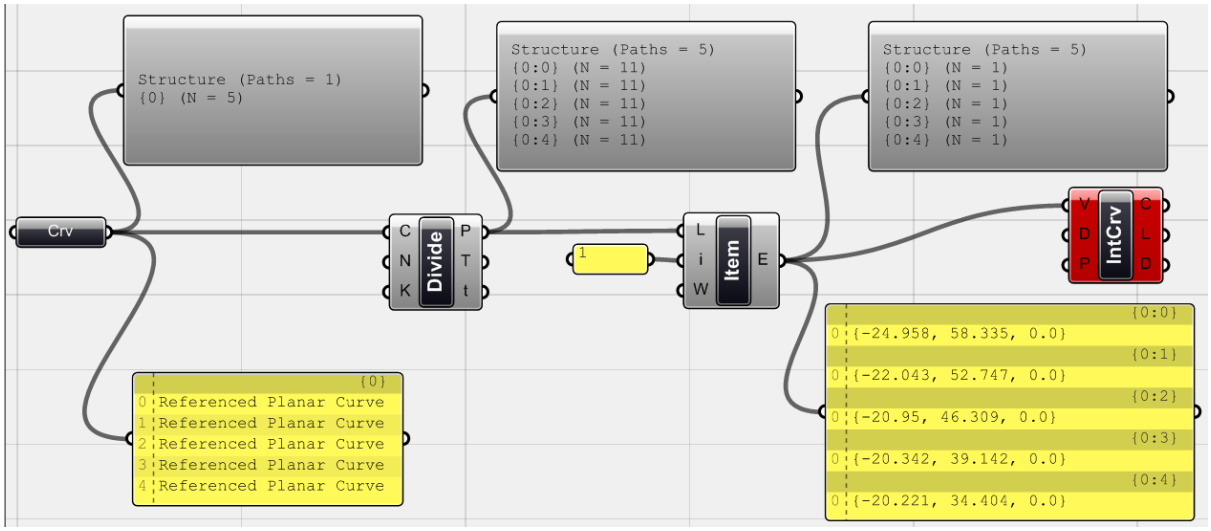
이때까지 component와 parameter에 대해서 이야기 해 보았다. 이제 grasshopper의 data 구조 방식 중 하나인 'Data Tree'에 대해서 이야기 해보자. 이것은 조금 복잡도가 높은 definition에서 꼭 필요하다.

Generative Algorithm의 가장 큰 잠재력 중 하나는 바로 수백 개의 객체를 한번에 다룰 수 있다는 것이다. 이렇게 많은 양의 객체를 다룰 때 필요한 것은 모든 객체에 하나의 명령어를 한번에 적용시키거나 혹은 특정 객체를 추출하고 그것에만 특정 명령어를 적용시킬 수 있다. 이를 위해서는 data의 구조를 이해하고 이것을 다르게 바꿀 수 있어야 한다.

5개의 curve가 있다. 이것을 각각 10개 등분 한 뒤 2번째 점들을 추출하고 이것을 <interpolate curve>를 이용하여 연결하여보자.



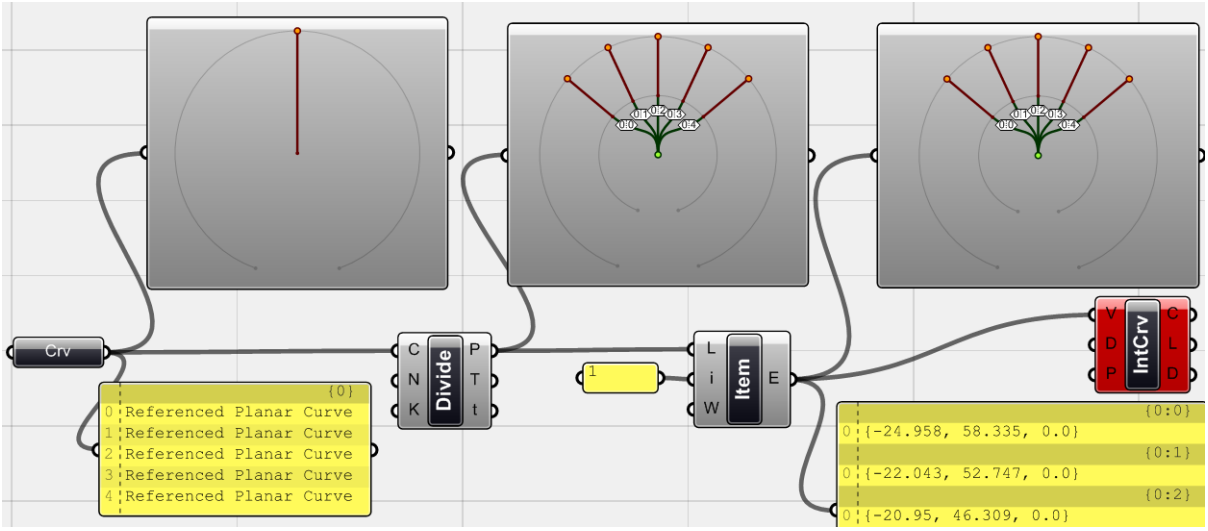
<curve> 를 이용하여 5개의 curve를 연동한 뒤, 이것을 <divide>를 이용하여 10등분 해준다. 이제 index number가 1인 항목들을 <list item>을 이용하여 추출해준다. 이렇게 하면 각 점들의 집합에서 두 번째인 점들을 선택해줄 수 있다. 이 점들을 <interpolate curve>를 이용하여 연결하여 주면 error 표시가 뜨게 되고 아무것도 그려지지 않는다.



이것을 <param viewer>를 이용하여 data 정보를 확인할 수 있다. 이것을 우 클릭 하여 'draw tree'를 선택하거나 더블클릭을 하면 data가 어떻게 가지로 분할되어있는지를 확인할 수 있다.

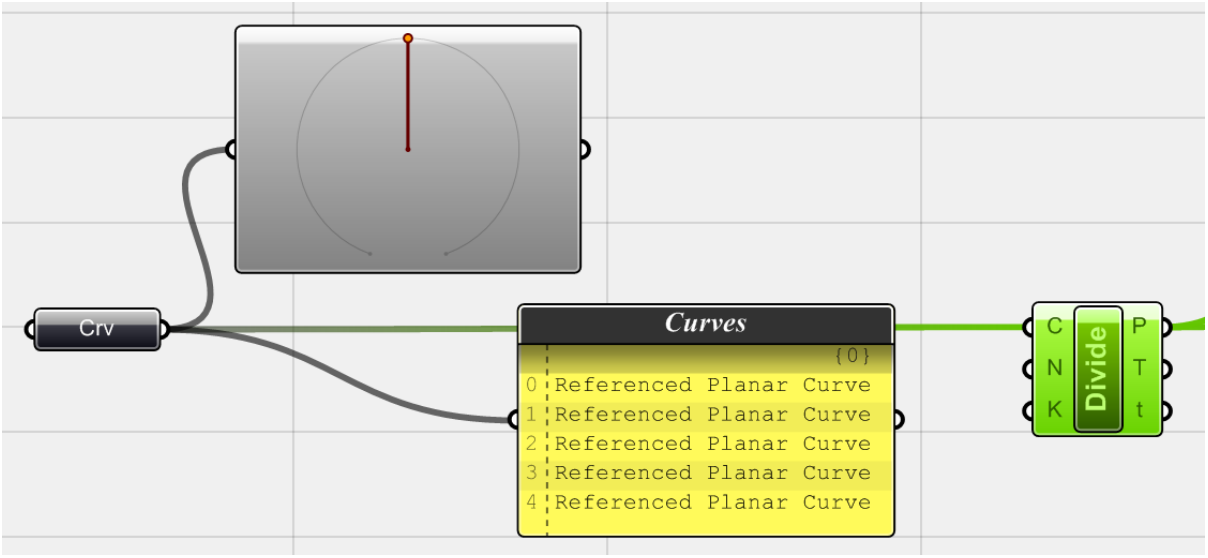
`

이렇게 <param viewer>를 이용하면 grasshopper 에서 사용되는 'data tree'의 개념을 시각화 시킬 수 있다. 위 그림에서 본 것처럼 <curve>는 5개의 항목을 가지고 있지만 이것이 <divide>를 거치면 각각의 curve 위에 11개의 점이 생기게 된다. 즉 하나의 data list 안에 55개의 점이 들어있는 것이 아니라 각각의 가지에 11개의 점이 나눠져 있는 것이다. 즉 55개의 data가 들어있는 'tree'가 다섯 개의 '가지(branch)'로 나누어 담겨 있는 것이다. 이렇게 되면 각각의 list에 담겨있는 특정 점을 고르기가 훨씬 수월해진다. 즉 <list item>에 1을 선택해주면 각각의 data list에서 고유 번호(index number)가 1인 항목을 골라낼 수 있다.

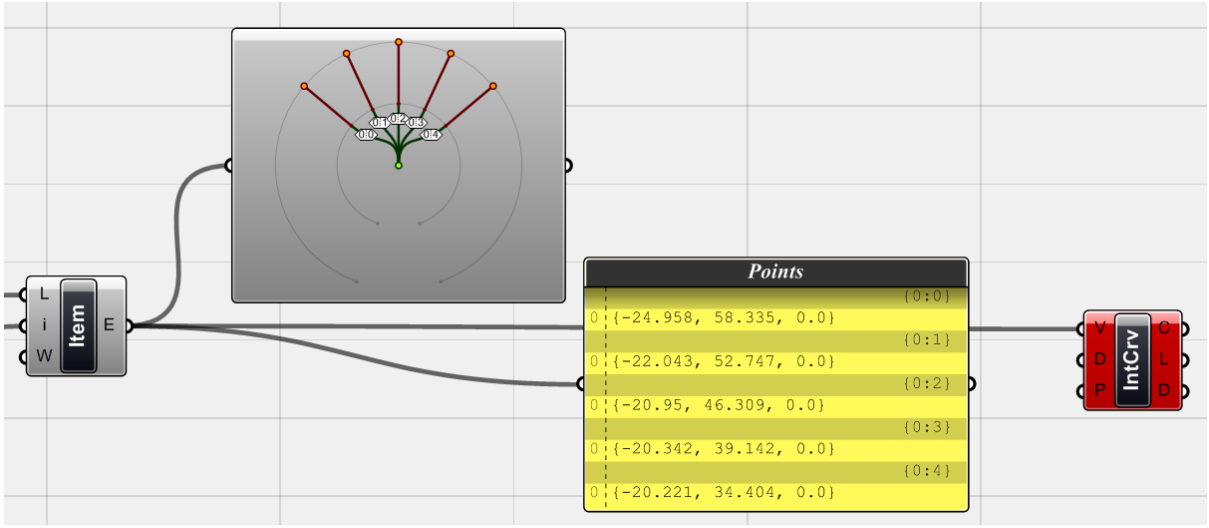


<param viewer>의 draw tree를 이용하여 각각의 component의 data 구조를 볼 수 있다.

이제 왜 <interpolate curve>가 아무런 선도 그리지 않았는지를 살펴보자.



`

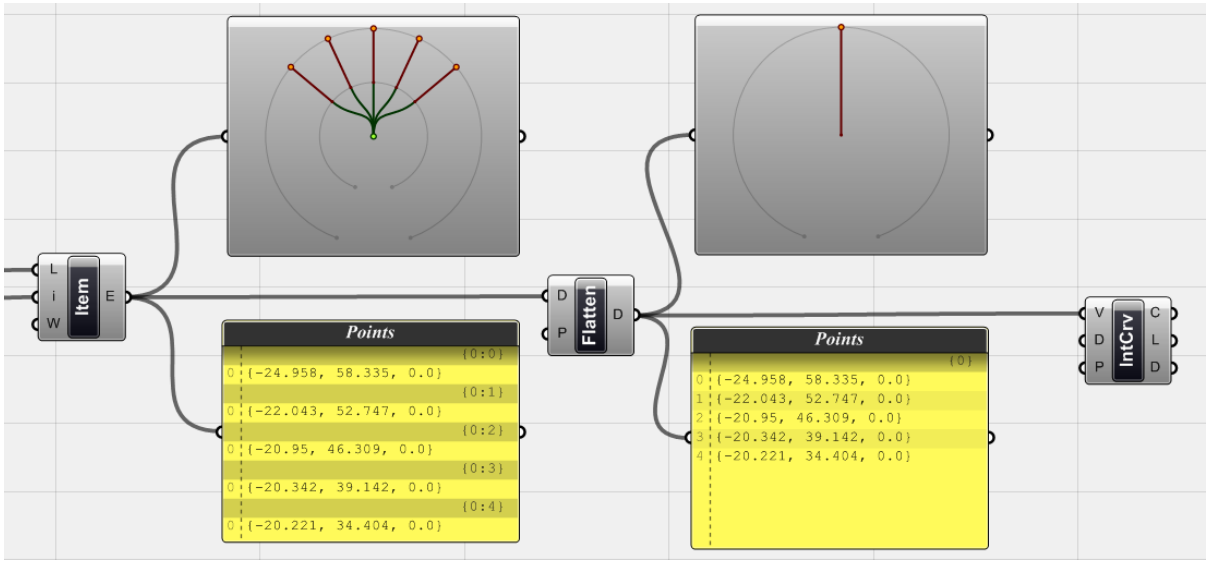


Curve와 <list item>의 point 각각의 <Param Viewer> 와 <Panel> 비교

위에 있는 그림에서 <param viewer>와 <curve>는 하나의 가지를 가진 data list를 보여준다. Panel을 살펴보면 모든 data는 {0}이라는 이름을 가진 curve의 list 에 속해있음을 알 수 있다. 즉 다섯 개의 curve모두 하나의 data list 안에 속해있다. <param viewer>에서 보여지는 {0} (N=5)는 {0}이라는 이름을 가진 data list에 5개의 항목이 있다는 의미이다.

아래 그림의 panel을 살펴보면 data들은 {0:0} {0:1} {0:2} {0:3} {0:4} 이라는 이름을 가진 data list 안에 각각 하나의 점들이 속해있음을 알 수 있다. <param viewer>를 보면 다섯 개의 가지를 가지고 있으며 각각의 가지에 (N=1), 즉 1개의 항목을 가지고 있다는 것을 보여준다. 즉 이것은 각 점들이 개별적인 list에 들어있기 때문에 복수개의 점이 하나의 data list에 있을 때 작동하는 <interpolate curve>가 작동하지 않는 것이다.

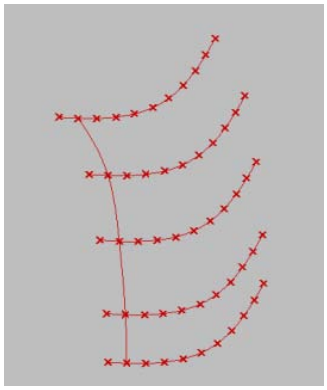
이제 이것을 해결해 보자.



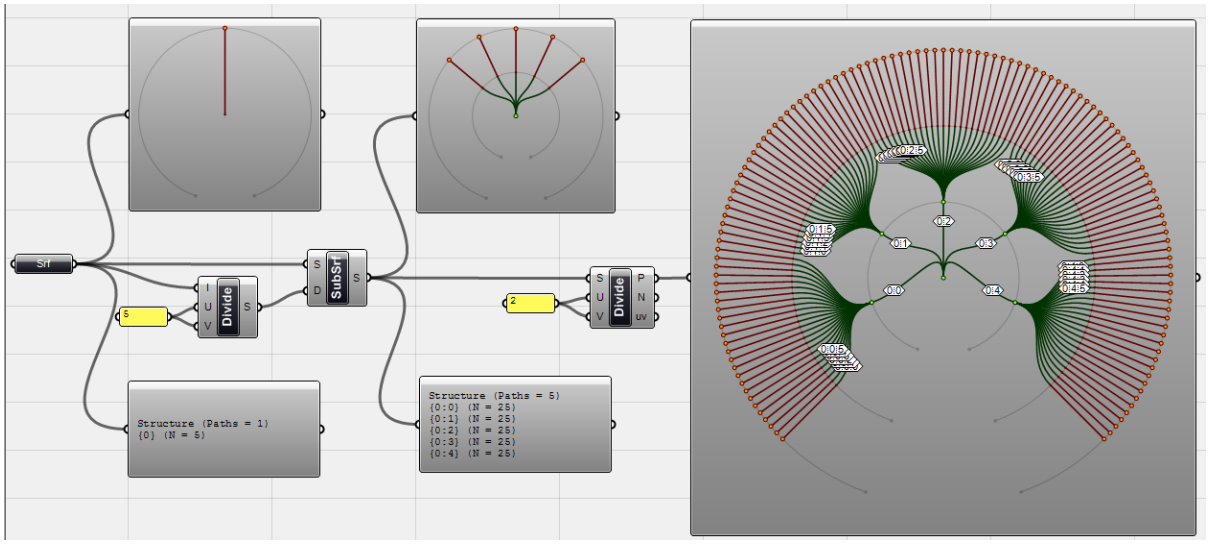
이것을 해결하기 위해서는 <flatten>(Logic>Tree) 을 이용할 수 있다. 이는 그 이름처럼 복수개의 가지를 가진 data 구조를 하나의 branch로 다시 만들어 주는 것이다. 그 구조의 차이는

`

<param viewer>에서 확인할 수 있으며 data 항목은 panel을 통해 확인할 수 있다. 이제 {0}에 모여 있는 다섯 개의 point data를 <interpolate>를 이용하여 연결해준다.



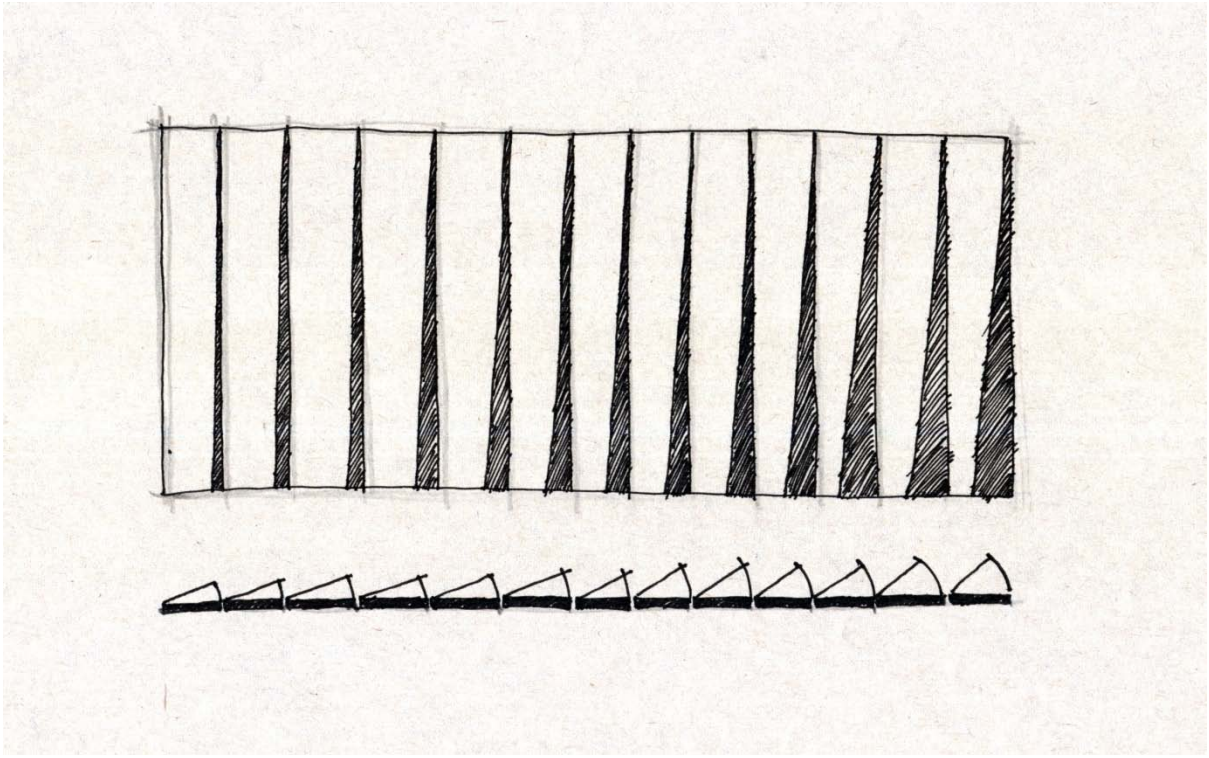
정리하자면, grasshopper를 이용하여 작업하다 보면 다양한 위계를 가진 복수개의 객체를 다뤄야 하는 경우가 많다. Tree 구조 내에서 각각의 가지(branch)들은 그 내부에서 위계와 {0:1}과 같은 고유번호를 가진다. 각각의 가지는 독립된 data list를 가지고 있는 것이다. 이러한 개념을 이해하고 이것을 이용하여 data를 다루는 방식을 아는 것은 무척 중요하다. 즉 필요에 따라 <graft tree> 혹은 <flatten>을 이용하여 data list 내부의 위계를 없애거나 만들어가며 작업을 할 수 있어야 한다. 다음 예시를 통하여 data를 다루는 것을 이해해보도록 하자.



'DATA tree'의 data 가지

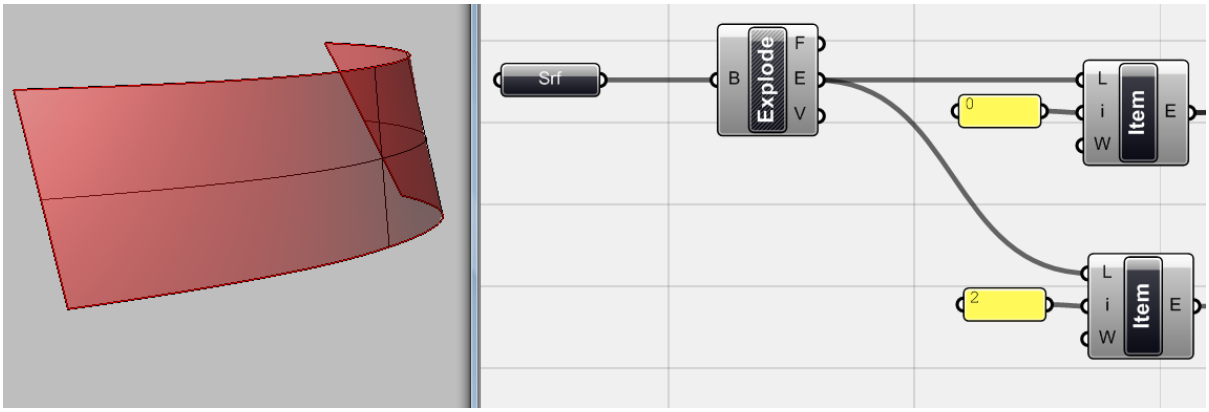
아래와 같은 형상을 가진 면이 있다고 하자. 이것을 주어진 하나의 면에 그려보도록 하겠다.

`



만들고자 하는 면의 스케치

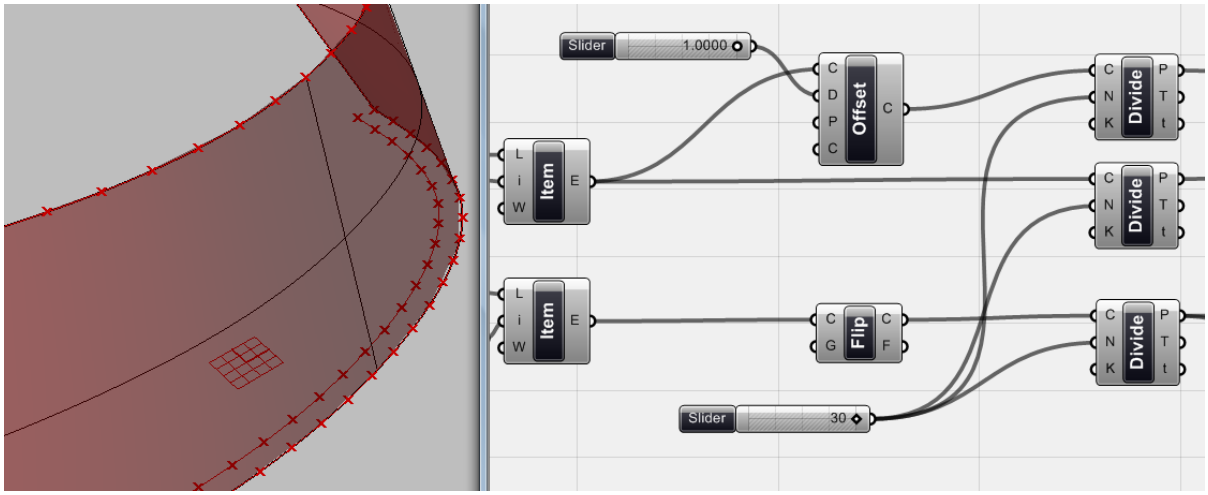
위와 같은 개구부를 가진 면을 만들기 위해서는 먼저 면의 위쪽과 아래쪽의 모서리를 잘게 잘라야 한다. 이를 이용하면 위와 같은 세장형의 면을 만들어낼 수 있다. 이제 아래쪽 선들의 방향을 조절하여 위와 같이 회전의 정도가 서서히 커지도록 해보자.



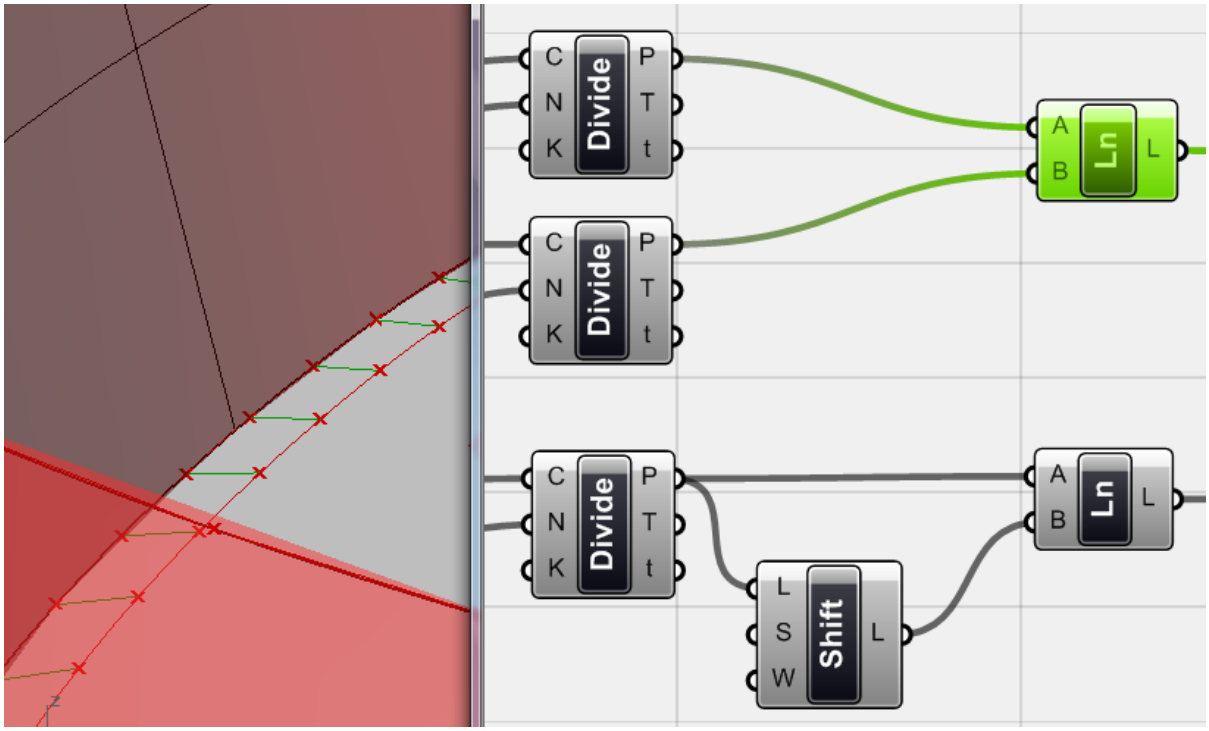
<surface>를 이용하여 rhino 상의 surface를 연동한 뒤 이것을 <BRep Components>(Surface>Analysis)에 연결하여 edge를 output으로 내보낸다. 위의 경우 아래 edge와 위쪽edge의 index number가 0과 2이다.[44] <list item>를 이용하여 각각의 edge를 찾아준다.

[44] 역자 주: 하나의 면에는 4개의 edge가 있다. 이 0과 2번의 경우 면을 생성한 방식에 따라 달라질 수 있다.

`



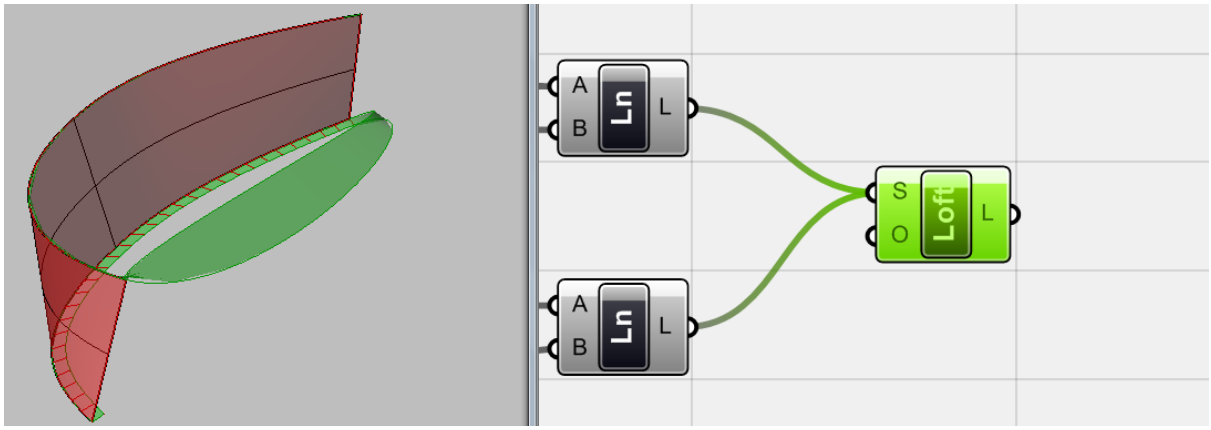
이제 아래쪽 edge를 <offset> (Curve>Util) 하고 위쪽 edge를 <Flip>(Curve>Util)해준다. 이는 아래쪽 edge와 위쪽 edge의 방향이 다르기 때문이다.[45]

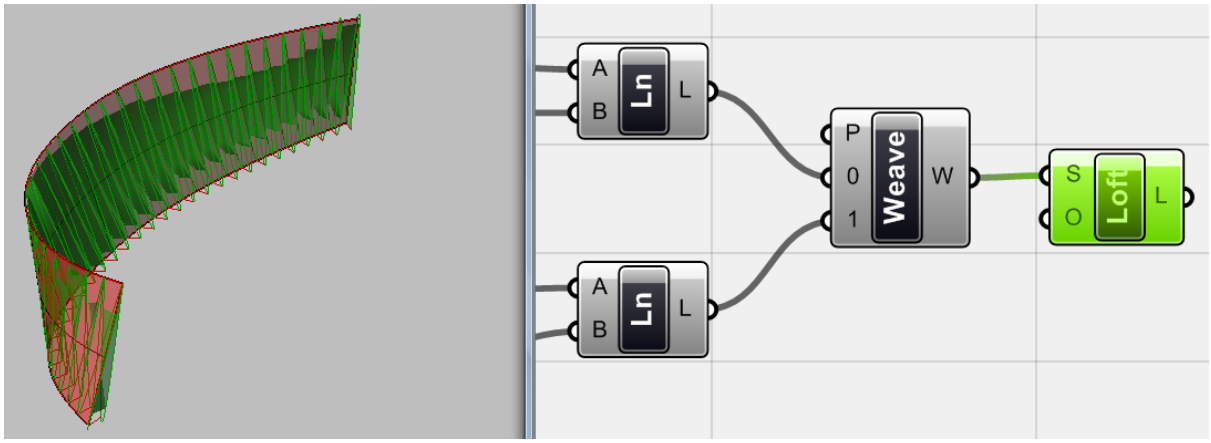


<divide>를 이용하여 각각의 edge에 같은 수의 point를 만들어준다. 아래쪽의 경우 offset 된 curve 또한 <divide>를 이용하여 나눈 뒤 나온 data list (point)와 아래쪽 curve를 나눈 data list(point)를 <line>을 이용하여 연결해준다. 위쪽 edge의 경우 <divide>를 이용해 나눈 점과 그 점의 list를 <shift>해준 점들을 <line>을 이용하여 연결해준다.

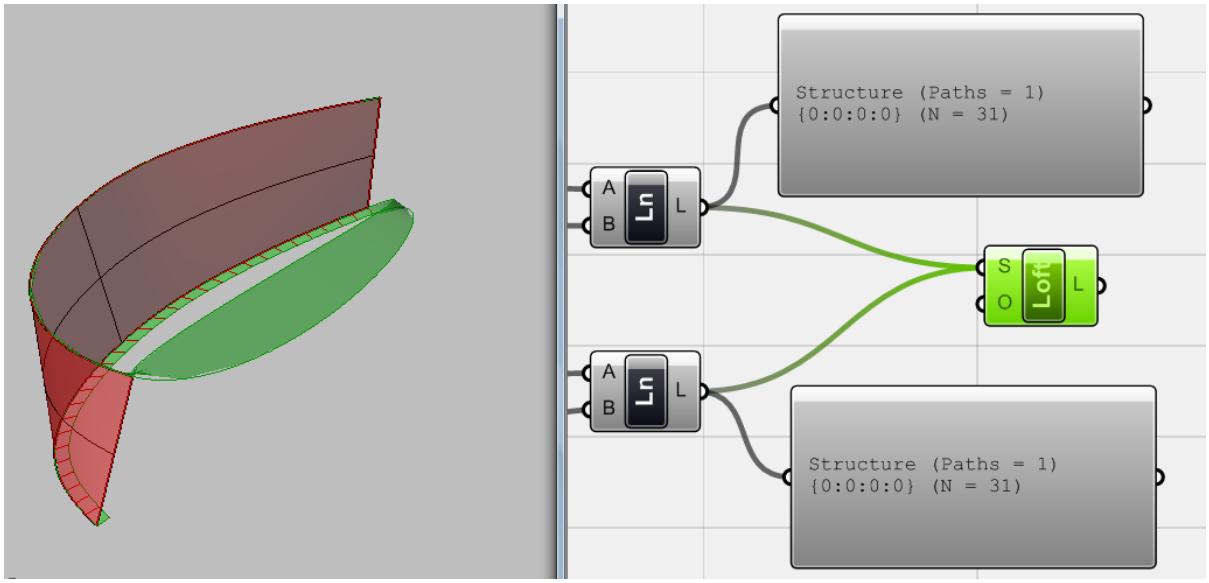
[45] 역자 주: rhino에서 한 면을 4개의 점으로 정의한다고 생각했을 때 점의 순서는 시계방향 혹은 그 반대방향일 것이다. 그러면 위쪽 edge와 아래쪽 edge의 시작점과 끝점의 위치가 엇갈리게 된다. 그러므로 각 edge의 방향이 다르게 된다.

`





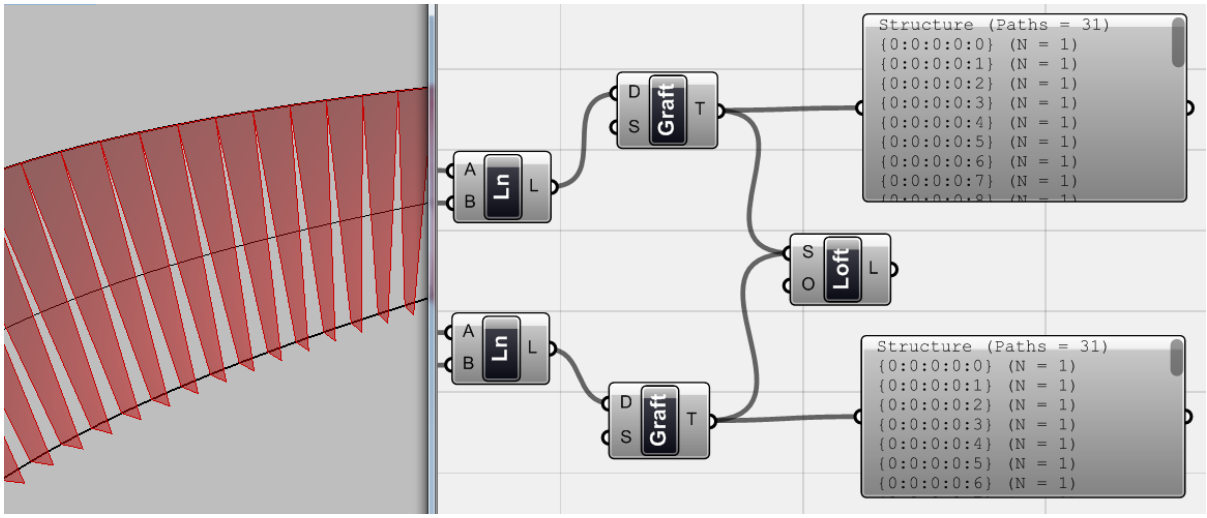
이제 이 두 <line>을 연결해야 하는데 위와 같이 그냥 연결하여도 <weave>를 이용하여도 원하는 결과가 나오지 않게 된다. 위의 그림처럼 loft를 하는 경우 아래쪽 <line>과 위쪽 <line>이 하나의 data list 내에 있게 되기 때문이다. <weave>를 이용하는 경우 각각의 list에 있는 line을 번갈아 가며 input할 수 있지만 두 개씩 쌍으로 묶여 분절되는 것이 아닌 연속하는 data list가 되기 때문에 역시 원하는 결과물이 아니게 된다.



<param viewer>를 이용하여 두 <line>을 살펴보면, 각각의 list가 하나의 가지 안에 들

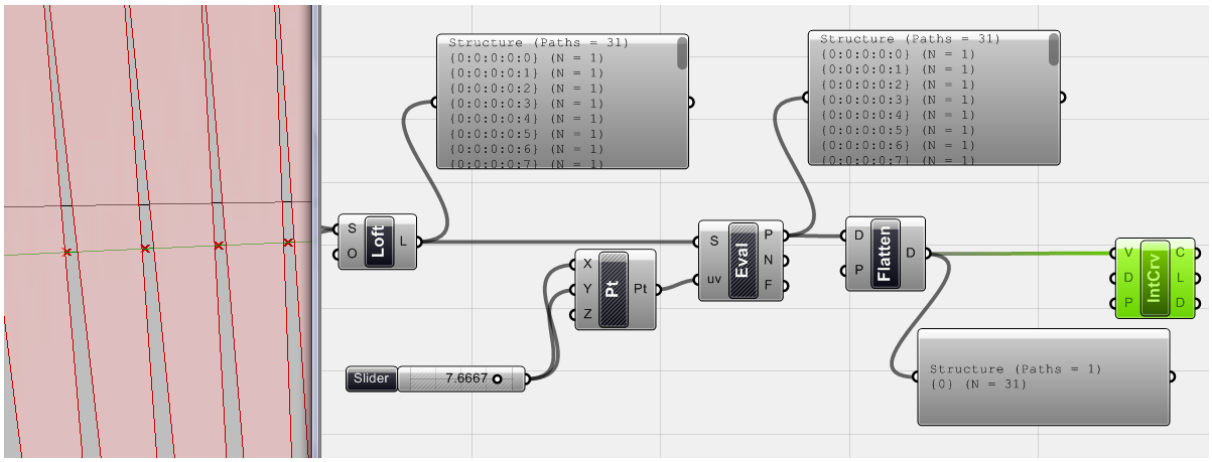
`

어있고 다시 이 두 가지가 하나의 리스트로 결합되어 input되기 때문에 위와 같은 결과가 일어나는 것이다. 우리가 원하는 것은 바로 각 list에 있는 데이터들이 1대1로 data matching 되는 것이다.



이러한 문제를 해결하기 위해서는 각 데이터를 <Graft> (Logic>Tree>Graft Tree)를 이용하여 tree 구조로 만들어줘야 한다. 이렇게 하면 두 data list안에 있는 각 data 들이 다시 하나의 list로 변하게 된다. 즉 data list 안에 하나의 data를 가지는 data list가 연속되고 다시 이 하부 data list들이 1대1로 data matching 되며 위와 같은 형상을 그리게 되는 것이다.

즉 위에서 살펴본 예들은 그 결과들이 모두 하나의 surface 이다. 하지만 <graft>를 이용하여 생성된 결과물은 여러 surface의 연속이 되는 것이다.

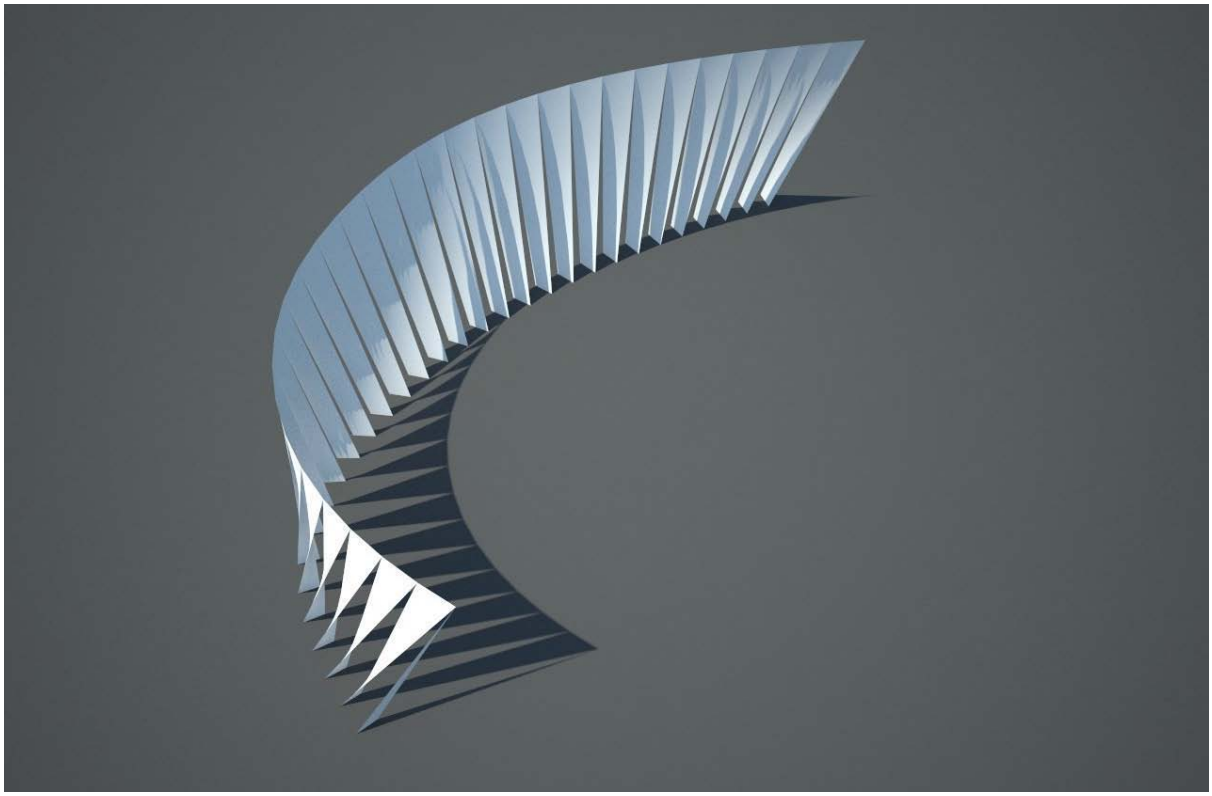


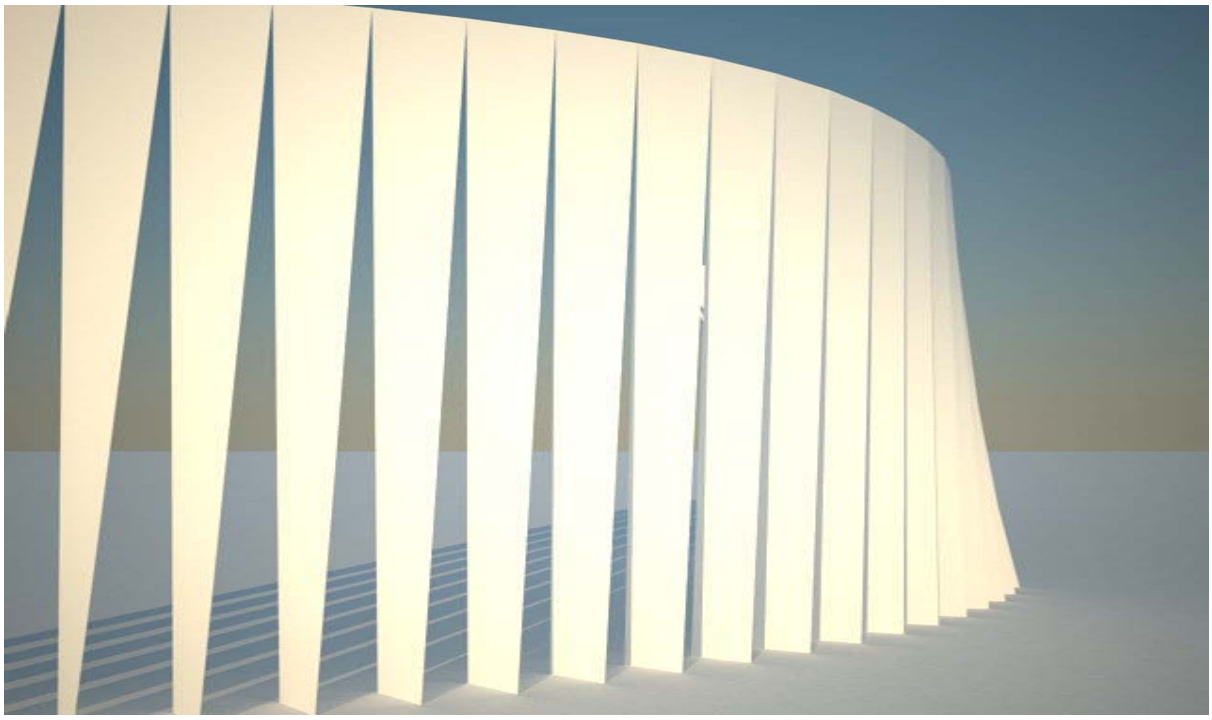
이렇게 생성된 면들 위에서 하나의 point를 evaluate 하고 다시 이것을 연결하여 <interpolate curve>를 생성해보자. 이 때 각 point들은 모두 다른 data list 속에 있으므로 <interpolate curve>를 <evaluate surface>에 직접 연결할 경우 아무것도 생기지 않는다. 기본적으로 curve란 최소한 두 개의 point가 있어야 생성될 수 있는데 각 data list에는 하나의 점 밖에 없기 때문이다. 이제 이것을 <flatten>을 이용하여 하나의 data list안

`

에 들어있게 바꿔주면 이 point들을 통과하는 interpolate curve를 그릴 수 있다.

이 tree 구조로 생성된 data들을 바꾸기 위해서는 (logic> Tree)에 있는 component들을 사용할 수 있다.





최종 3d 모델

`

`

Chapter_6_ Deformations and Morphing

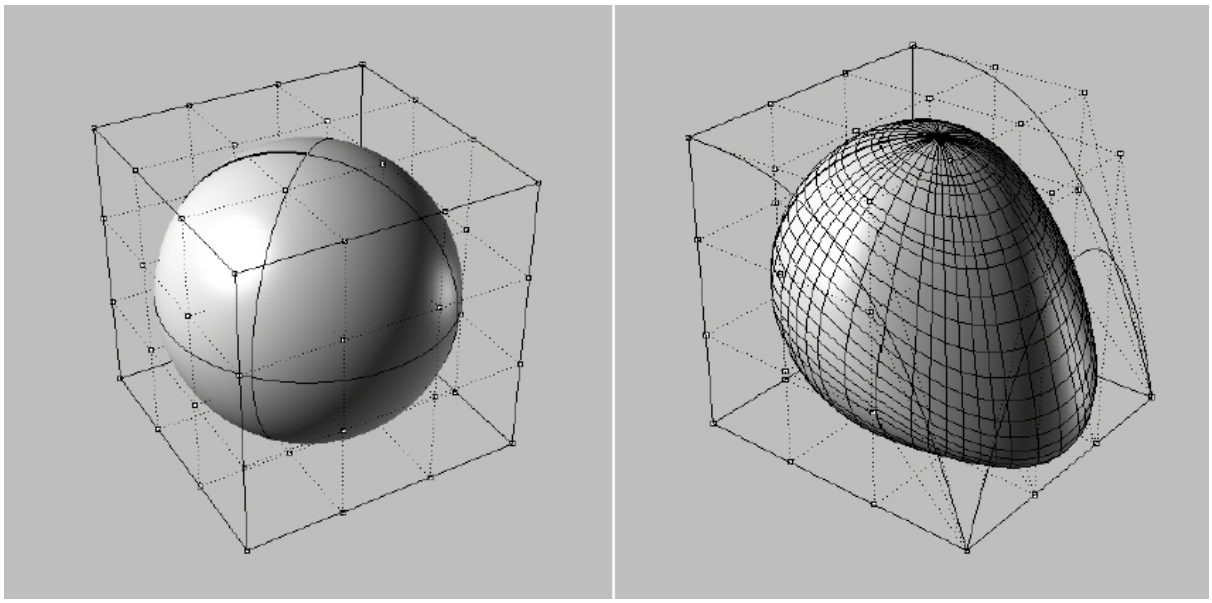
`

6_1_Deformations and Morphing

기하체(geometry)란 순수한 입방체뿐만 아니라 이것을 변형하여 디자인에 적용한 모든 것을 아우른다. Deformation과 Morphing은 순수한 기하체를 변형시키는데 사용되는 방법 중 하나이다.

이 둘은 자유로운 형상(free-form)을 만드는데 무척 강력한 힘을 가지고 있다. deformation에는 비틀기, 전단, 구부리기 (twist, shear, bend)등이 있으며 morphing은 기하체를 감싸는 경계를 만든 뒤 이 경계를 변형시켜 기하체의 전체적인 형상을 변형시키는 것이다.

간단한 deformation의 예를 살펴보자. 구(sphere)가 있다고 했을 때, 이것을 모두 감싸는 박스를 만들고 그것을 변형시키면 구 전체를 변형시킬 수 있다는 것을 쉽게 알 수 있다.



Bounding Box를 이용하여 객체를 변형시키기

기하체 각 부분의 좌표가 변하는 각도나 이동하는 정도에 따라서 우리는 이를 전단(shear), 구부리기(bend), 자유 변형(free deformation)등으로 나눈다. 이렇게 deformation을 해주기 위해서는 위처럼 기하체 전체를 box로 감싸서 변형시키거나 아니면 각각의 면을 나누어 변형시킬 수 있다. 혹은 하나의 꼭지점만을 이동시켜도 기하체를 변형시킬 수 있다. Grasshopper에는 기본적인 deformation을 위한 여러 component들을 제공하고 있다.

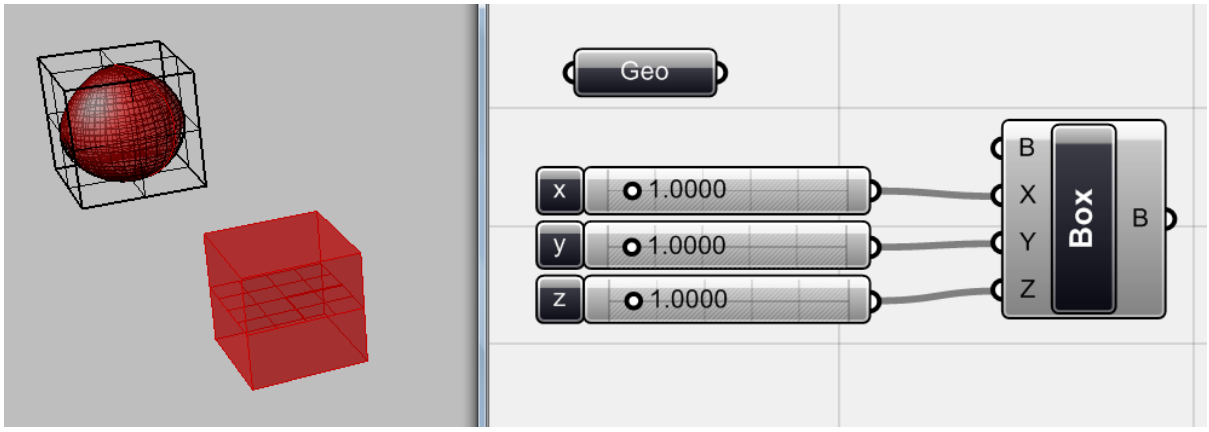
Animation에서 Morphing이란 하나의 장면에서 다른 장면으로의 전환이 부드럽고 끊김 없이 일어나는 것을 의미한다. 3d에서는 이것은 하나의 기하체를 감싸는 경계의 형상(boundary condition)이 변하여 다시 이것이 그 내부의 기하체를 변화된 경계에 맞는 형상으로 변회시키는 것을 의미한다. Grasshopper에서의 morphing 또한 같은 의미를 가진다. 예를 들어 <Box Morph> (XForm >Morph)의 경우에는 하나의 객체와 그것을 감싸는 box 모양의 경계(bounding box)를 기준으로 하여 객체를 다른 형상을 가진 경계(target box)에 알맞게 변화시켜주는 것이다. <Surface

`

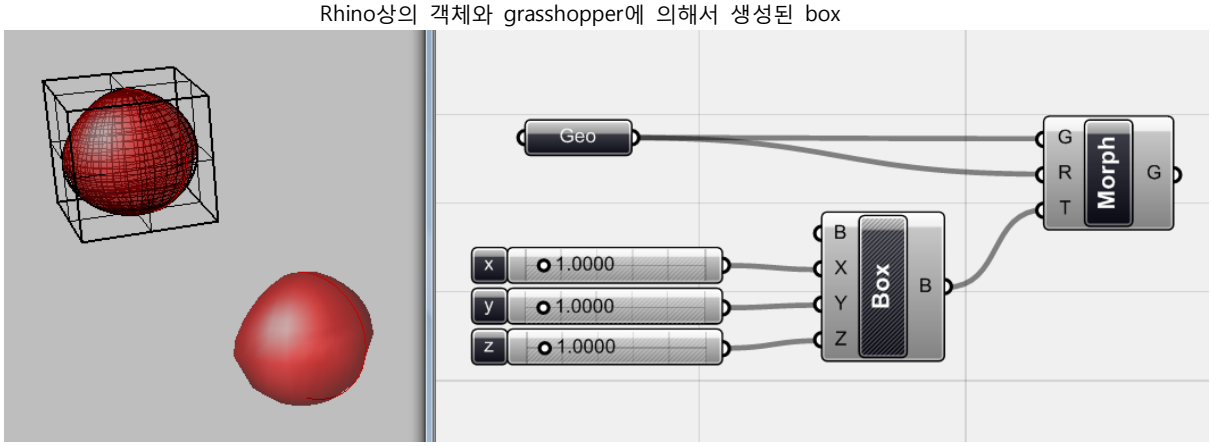
morph>는 surface를 기본으로 한다. 즉 surface 위에서 특정 영역을 설정한 뒤 그 위에 기하체를 옮겨놓고 그 높이를 조절할 수 있다.

<Box Morph>와 <Surface morph>은 XForm tab의 Morph panel에서 찾을 수 있다. Grasshopper를 이용하면 몇 가지 component를 이용하여 Box를 변형시킬 수 있다. 하나의 기하체를 변형된 box들을 target box로 삼아 <box morph>에 연결해주면 기하체를 쉽게 그릴 수 있다.

아래 그림의 경우 <geometry>를 이용하여 rhino상의 객체를 grasshopper에 연결시켰다. 모든 객체의 경우 그것이 시각화 되어있지는 않지만 그 주위에 bounding-box가 있다. 아래 그림에서는 필자가 직접 박스를 그려 이것을 시각화 시켰다. 또한 <center box>를 이용하여 다른 하나의 box를 그렸다.

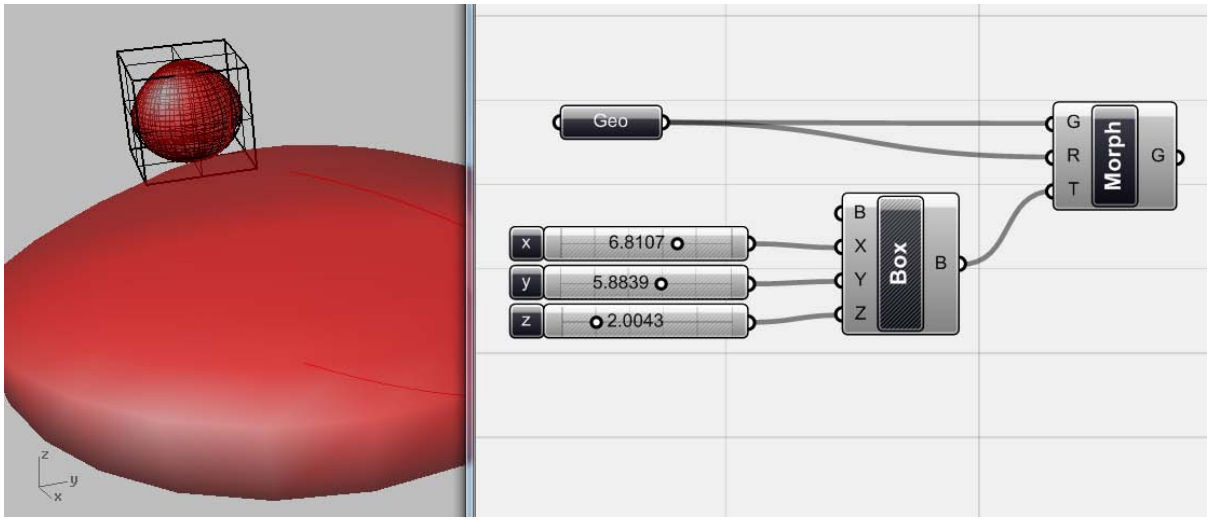


Rhino상의 객체와 grasshopper에 의해서 생성된 box

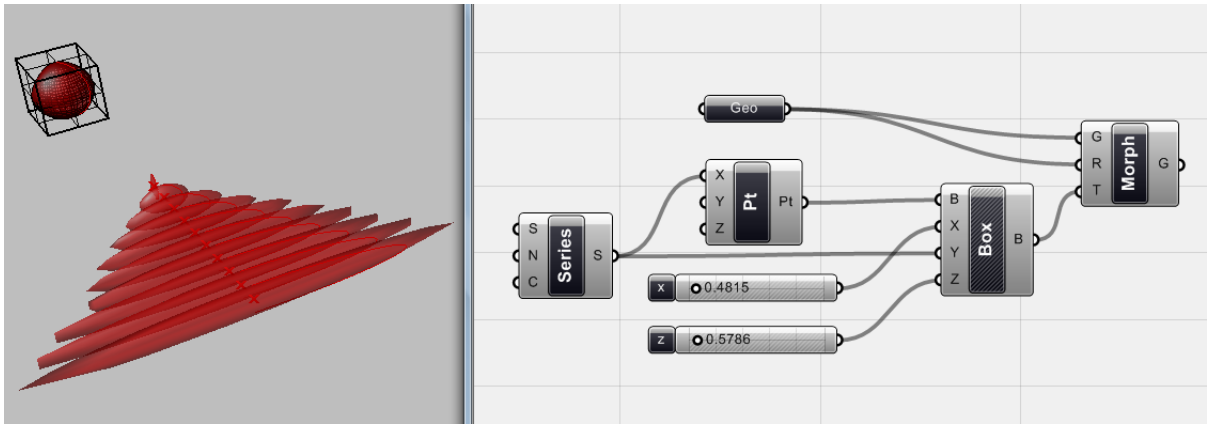


<box morph> XForm > Morph > Box morph)는 하나의 객체에 그것을 감싸는 bounding box가 있다고 가정한 뒤 그것을 target box와 비교하여 객체를 변형하고 이것을 target box내에 생성시켜 주는 것이다. <Bounding box> (Surface > Primitive > Bounding box)를 이용하면 객체를 감싸고 있는 bounding box를 시각화 시켜줄 수 있다. 아래의 경우 <box>의 preview를 꺼서 내부의 변형된 객체가 더 잘 보이게 하였다.

`



위의 그림에서 보는 것과 같이 bounding box의 크기를 변형시키면 이에 맞춰서 객체의 형상이 변하는 것을 볼 수 있다.



위 경우 복수개의 box를 생성시킨 뒤 이것들에 맞춰 객체를 변형시킨 것이다. 이처럼 <series>를 이용하여 각 box의 y방향 길이를 다르게 적용하였는데 객체 또한 이에 맞춰 변화되는 것을 볼 수 있다.

6_2_곡면의 패널화(On Panelization)

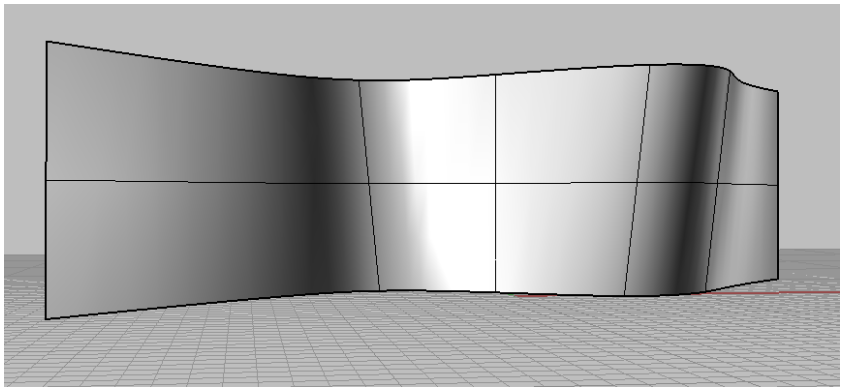
Morphing을 이용하여 할 수 있는 가장 흔한 것이 곡면을 패널로 재해석 하는 것이다. 즉, 자유곡면을 물리적으로 구현하기 위해서 이를 제조(fabrication)가 가능한 조각들로 잘게 쪼개는 것이다. 이러한 조각들이 생산되면 이것들 다시 연결하여 자유곡면을 만들어낼 수 있다. 이러한 자유곡면이 자동차 산업에서는 패널화 없이 그대로 사용되지만 건축의 경우 훨씬 커다란 면을 만들기 때문에 자유곡면을 그대로 표현하기가 무척 어렵다. 이러한 패널화의 장점으로는 먼저 상대적으로 곡면을 만들기 쉽고 또 이 조각들을 운송하기 쉽다는 것이다. 또한 마지막 결과물의 정확도를

`

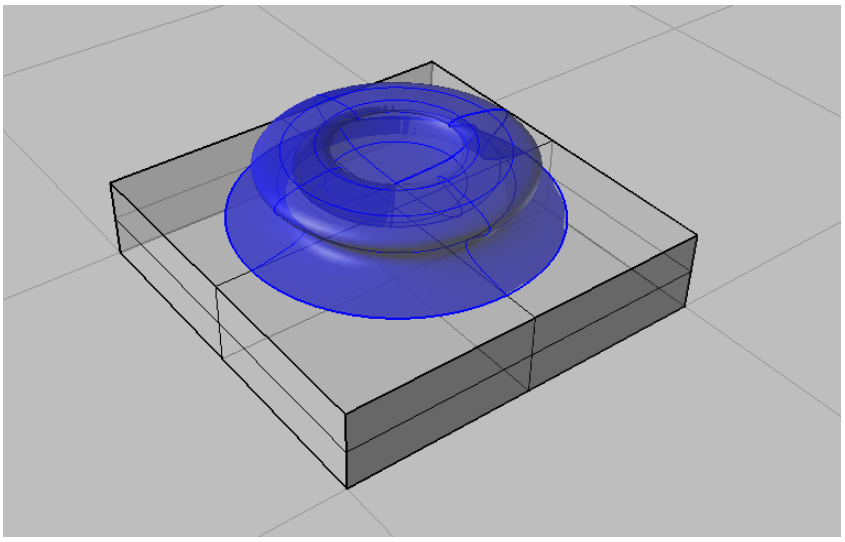
높일 수 있다.

다른 방법은 면을 평평한 면을 가지는 조각들로 나누고 이를 연결하여 곡면과 그 형상이 비슷한 면을 만들어내는 것이다. 이 경우 각 면들의 크기와 곡률, 조정 값 등을 고려하여야 한다.

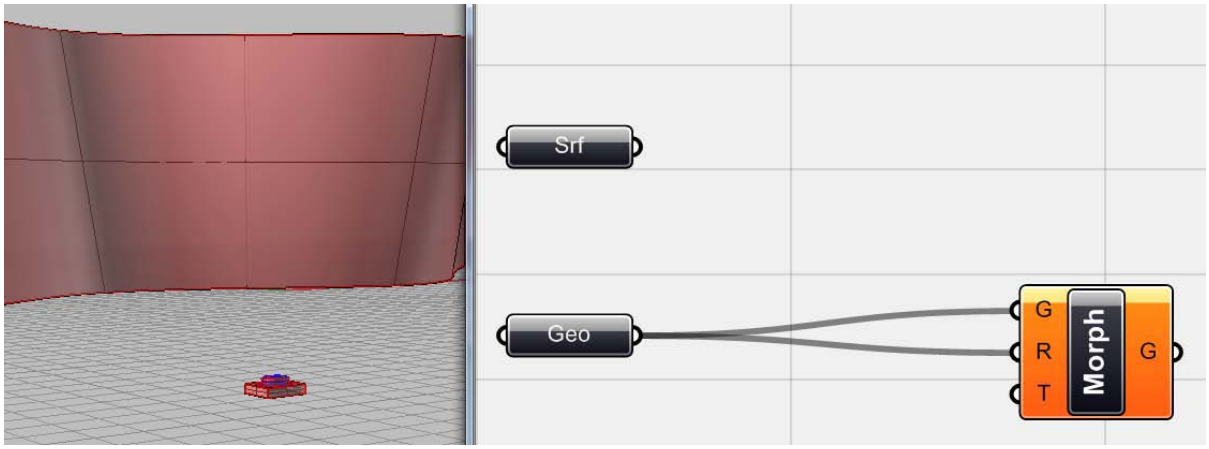
간단한 면을 패널화(panelization)해보자.



두 개의 곡률을 가지는 surface (double-curved surface for panelization)

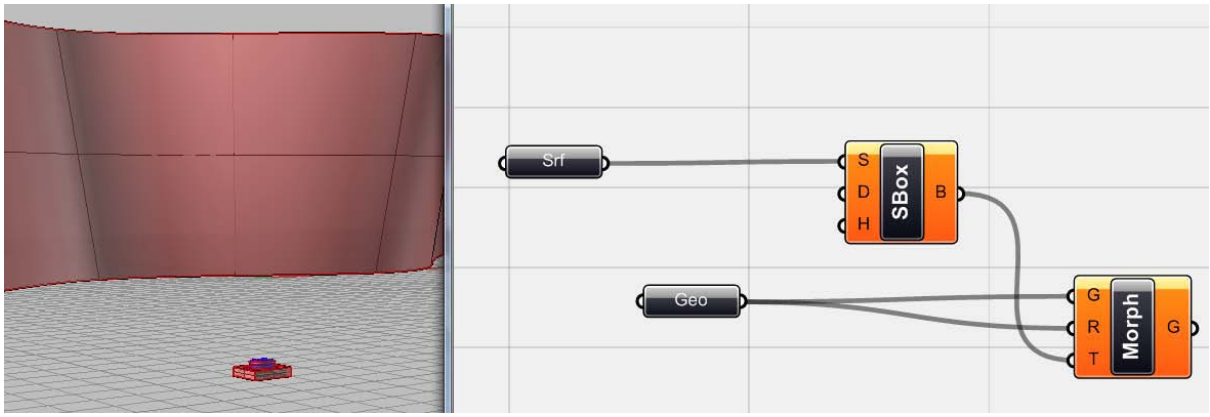


패널화에 사용될 component

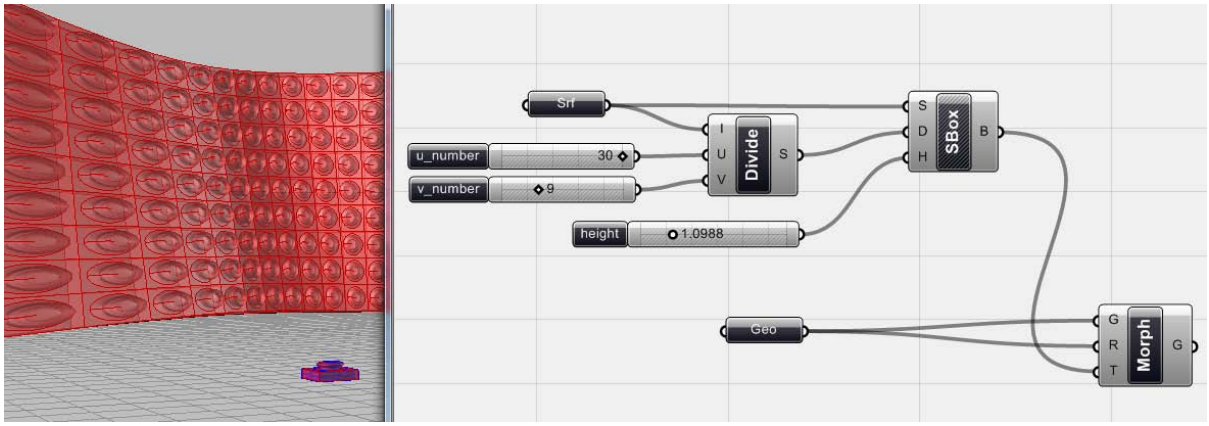


`

먼저 <surface>를 이용하여 면을 grasshopper에 연동시킨다. Grasshopper에서 제공하는 component의 목록과 위에서 배운 내용을 생각해보면 이 surface 위에 원하는 수만큼의 box를 만든 뒤 이것을 target box로 사용하는 것이다. <geometry>를 이용하여 rhino 상의 기하체를 연동시킨 뒤 이것을 <box morph>를 이용하여 target box에 복사하는 것이다.



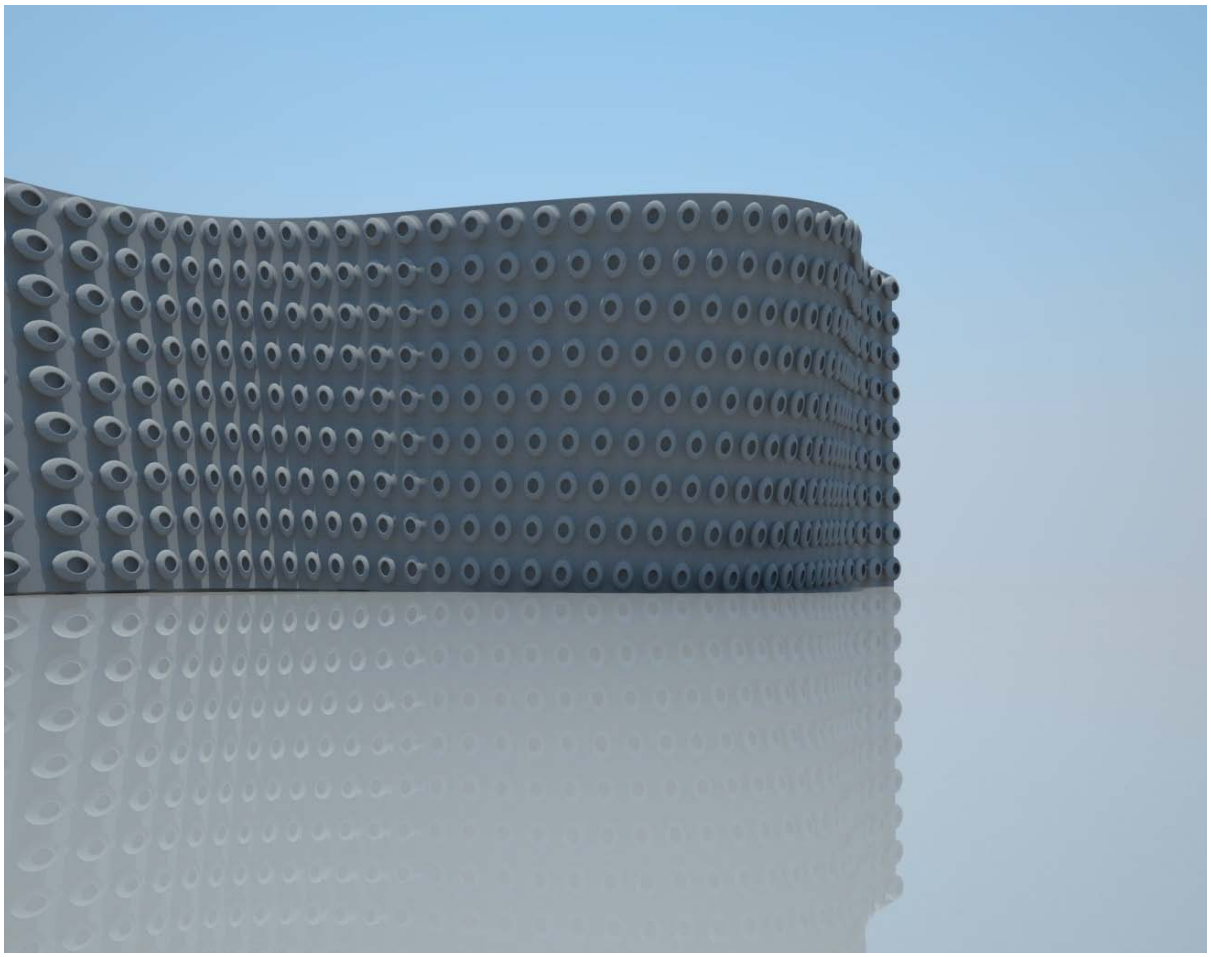
이를 위해서는 <surface box>(XForm>Morph>Surface Box)를 이용하여야 한다. 이 component는 하나의 surface위에 UV방향으로 원하는 수만큼의 box를 생성해주는 역할을 한다. 또한 각 box의 높이(Height)값을 설정할 수 있다. 이렇게 생기는 box들이 바로 <box morph>의 target box가 될 것이다. 그러기 위해서는 먼저 domain과 UV값을 설정해주기 위한 수치값 등이 필요하다.



surface를 정의역(domain)으로 설정하기 위해서는 <divide domain2>이 필요하다. 이것의 I(base domain)에 <surface>를 연결한 뒤 <number slider>를 이용하여 원하는 UV값을 넣어준다. 다른 하나의 <number slider>는 높이 값을 정해준다.

즉 기본적인 아이디어는 무척 단순하다. 하나의 module과 하나의 surface를 만들어준다. 이 surface를 원하는 수로 나누고 그 나눠진 개개의 면을 box로 만든 뒤 그 box를 target box로 하여 module을 알맞게 변형하여 복사해주는 것이다. 이제 각 U방향과 V 방향에 적용되는 수를 조절하면 그에 따라 surface가 자동적으로 변하게 된다.

`



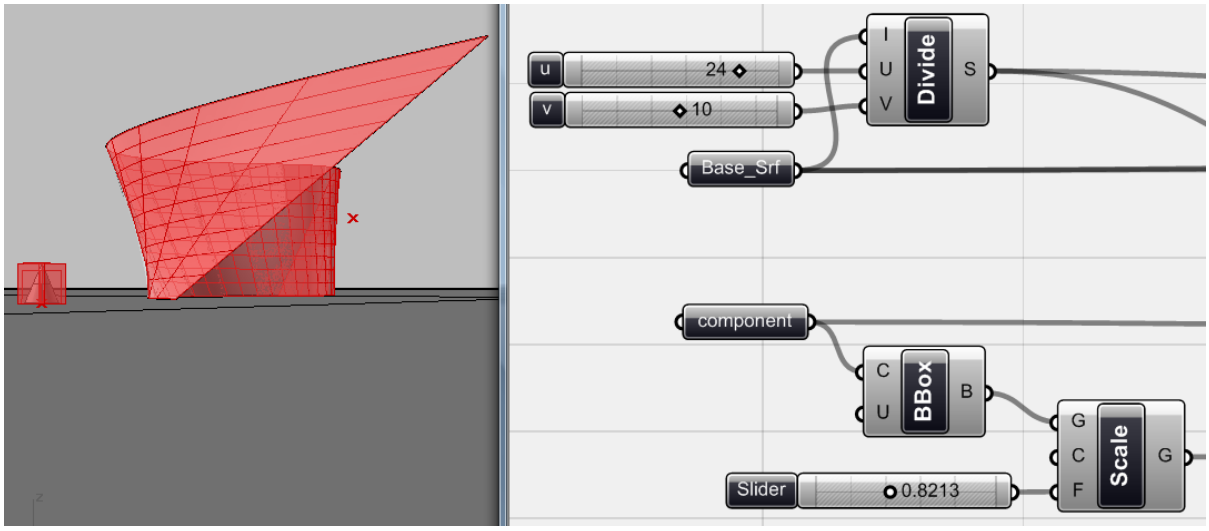
Module이 적용된 surface의 결과물

`

6_3_Micro Level Manipulations

위의 예시처럼 하나의 component를 surface에 맞춰 배열하는 것은 design의 가장 일반적인 방법 중에 하나이다. 이것을 이용하면 면 위에 생성되는 module의 개수를 조절하고 그 높이를 바꾸는 것이 가능하다. 하지만 이것은 전반적인 형상을 바꾸는 것일 뿐 국지적으로 각 module을 조정해줄 수 있는 것은 아니다.

이번 예제에서는 국지적으로 각 모듈들을 조절해줄 수 있는 grasshopper definition을 짜보도록 하자. 앞서 살펴본 개념 중에 끌개(attractor)를 적용시켜 보도록 하자. 이것을 이용하면 각 모듈의 높이와 너비 등을 따로따로 적용시켜줄 수 있다.

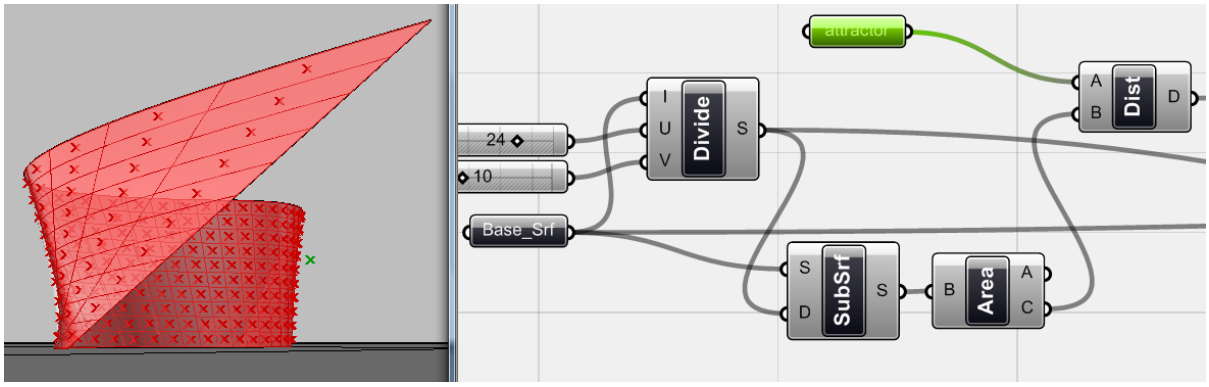


기본적으로 필요한 component들을 살펴보자. 먼저 두 개의 곡률을 가진 면을 <surface>를 이용하여 grasshopper에 입력한 뒤 이것의 이름을 <Base_srf>로 바꿔주자. 또한 Rhino에서 그려준 원뿔을 <geometry>를 이용하여 연동시키고 이것의 이름을 <component>로 바꿔주자. 이제 <divide interval 2>를 이용하여 surface를 나눠주고, component는 bounding box를 이용하여 원뿔을 감싸는 최소한의 크기를 가진 box를 만들어준다. 이것은 후에 reference box로 쓰일 것이다. 이제 <scale>을 적용하여 bounding box의 크기를 조절해준다. 이는 후에 surface 위에 생성되는 원뿔의 크기를 조절하는데 사용된다.

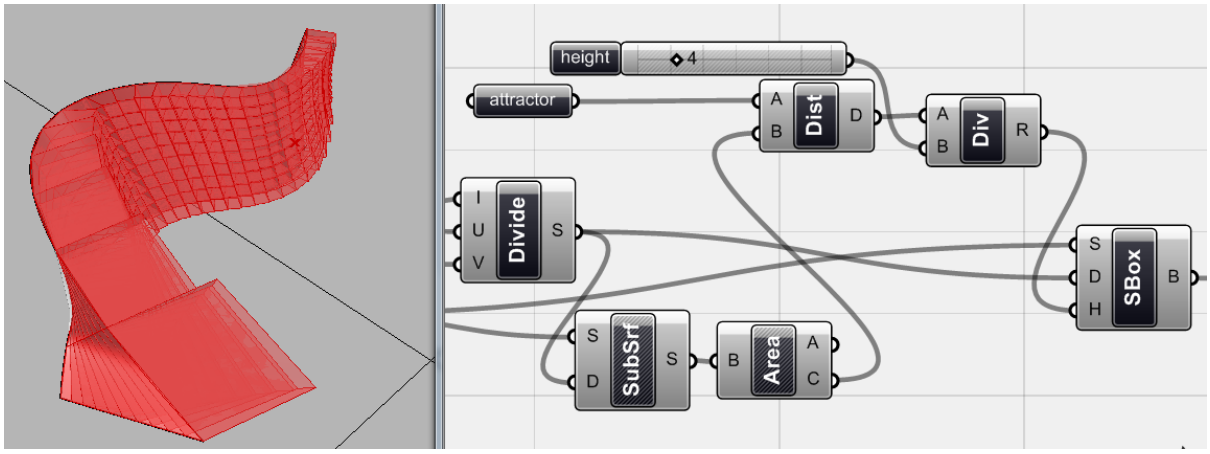
<surface box>를 이용하면 면의 각 부분에 생기는 box의 높이를 조절해줄 수 있다. 여기서 이 높이를 하나의 절대값이 아닌 위치에 따른 상대 값을 적용시켜주도록 하자. 즉 <number slider>를 이용하여 하나의 수를 모든 box에 적용시켜 주는 것이 아닌, 위에서 이야기한 것처럼 끌개를 활용할 수 있다.

이제 끌개와 각 box 사이의 거리를 측정하여야 한다. 여기서 문제점은 바로 box의 어떤 점을 거리를 측정하는 기준점으로 삼느냐는 것이다. 이 예제에서는 각 box가 생성되는 면들의 중심점을 사용해주도록 한다.

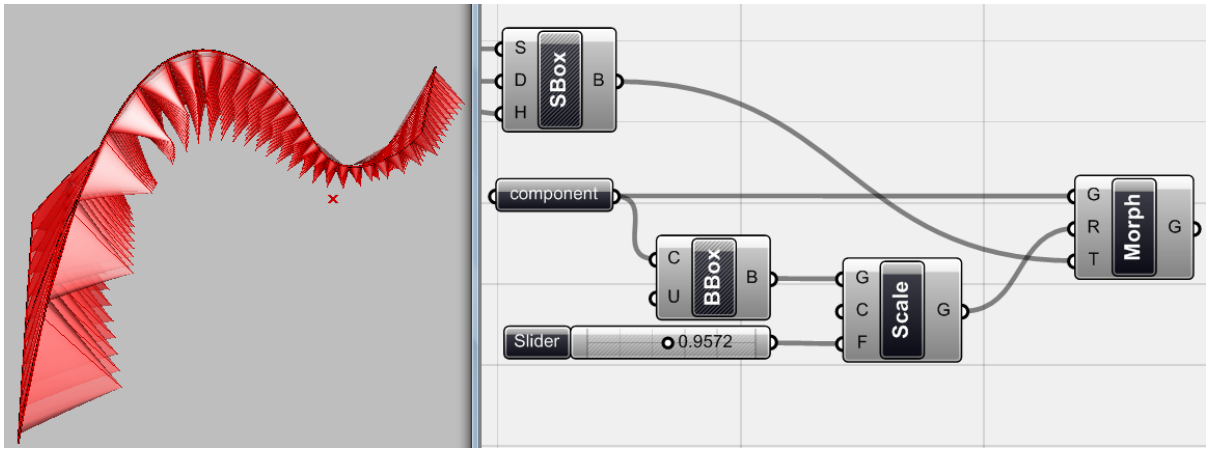
`



위 그림을 보면 먼저 <divide interval2>과 <iso trim>(Surface > Util > Isotrim) 을 이용하면 면을 여러 개의 면으로 나눠줄 수 있다. 이렇게 생성된 면들의 중심점을 찾기 위해서는 <Brep area>(Surface > Analysis > BRep area)를 이용하면 된다. <Brep area>는 input의 실제 넓이(A; Area)와 그것의 중심점(C; Area centroid)을 출력해준다. Rhino상에 하나의 점을 그려준 뒤, 이것을 <point>를 이용하여 grasshopper에 연동시키고 <distance> 를 이용하여 각 점으로부터의 거리를 찾아준다.



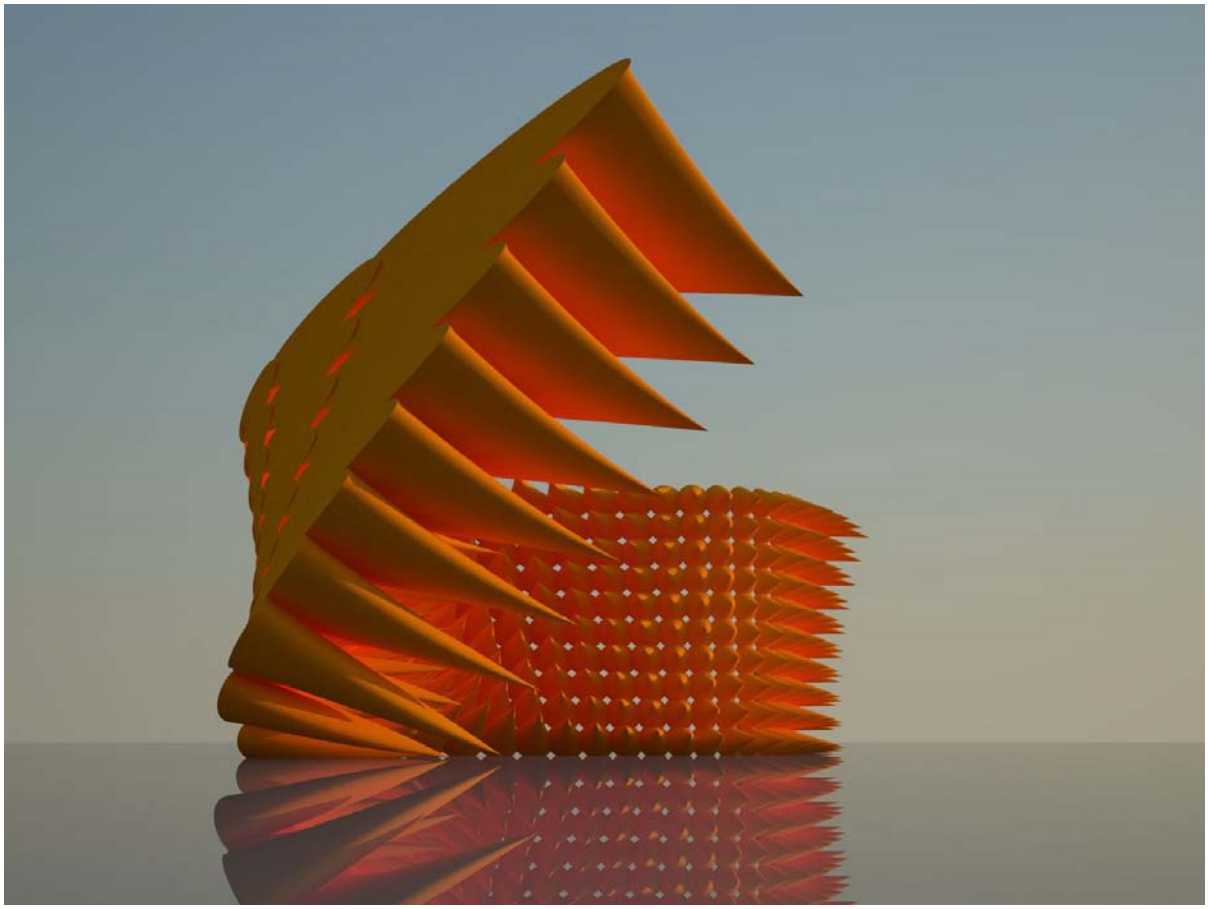
이 <distance>의 출력값을 <divide>와 <number slider>를 이용하여 적절하게 나눠준다. 그리고 이 값을 <surface box>의 높이(H; height)값에 적용시켜준다. 이제 이 '끌개'의 위치를 바꿔주면 이것에 따라 각 box들의 높이가 변하게 된다. <surface box>의 S값에는 <base_srf>가, 그리고 D에는 <divide interval 2>가 적용된다.



이제 남은 것은 <component>, <Scale> 그리고 <surface box>를 <morph box>에 연결하여 원뿔

`

을 각 box의 크기에 맞춰 복사해주는 것이다. 이때 이 scale factor를 조절하면 surface위에 있는 모든 원뿔들의 크기를 조절할 수 있고 또 끌개의 위치에 따라 그 길이를 조절할 수 있다.



마지막 모형

위에서 볼 수 있는 것처럼 각 원뿔은 국지적으로 작용하는 변인들을 받아드려 각기 다른 형상을 가지고 있다. 이것이 기본적인 끌개의 개념을 이용한 것일 지라도 매우 흥미로운 결과물을 낼 수 있는 것이다. 이제 morphing과 panelization의 활용이 항상 일괄적이지 않다는 것을 알 수 있다. 이제 다음 예로 넘어가보자.

`

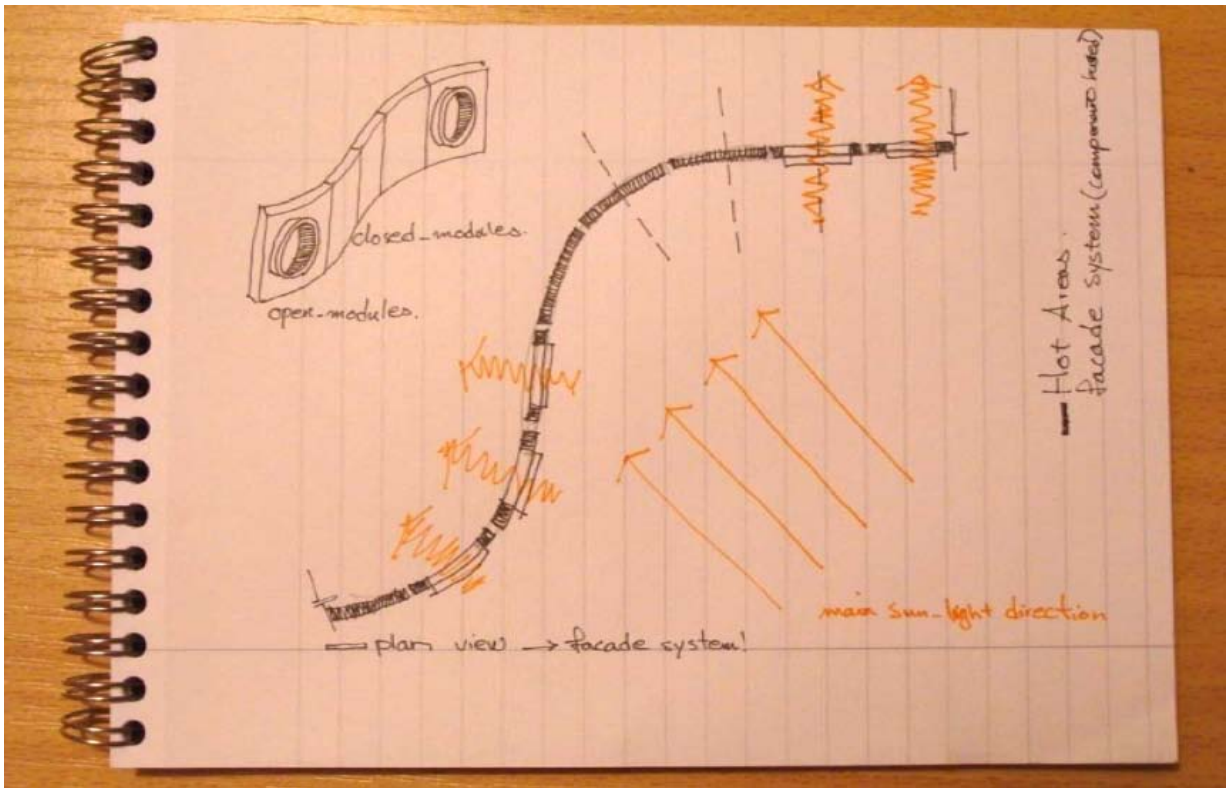
6_4_하나의 면에 여러 모듈을 적용하기 On Responsive Modulation

이번 장에서 다뤄볼 디자인 개념은 하나의 surface에 여러 종류의 module을 적용하는 것이다. 이 때 각 module의 배치가 종류별로 사용자가 지정하는 특정 기준에 의해서 정해지게 된다. 이러한 배치의 기준은 시간에 따른 주변 환경의 변화나 module의 기능 혹은 시각적인 효과 등 module의 반응을 필요로 하는 여러 가지가 될 수 있다.

이번 예제에서는 건물의 외피가 주변 환경의 변화에 따라 반응하는 것을 살펴보려고 한다. 이 때 환경의 변화요인은 바로 태양의 빛이다. 이는 물론 바람이나 비 혹은 둘 이상의 조합도 가능하다.

건물의 표면을 덮고 있는 하나의 면이 있다. 이것은 다른 두 종류의 module에 의해 감싸지게 된다. 첫 번째 종류는 닫힌 것으로 태양 빛의 투과를 막는 역할을 한다. 다른 한 종류는 열린 것으로 태양의 빛이 들어올 수 있는 것이다. Surface 위에서 이 둘의 배치는 태양의 위치에 따라 변하게 된다. 즉 태양의 고도를 각도 값으로 치환하고 이것을 module 배치의 기준으로 삼는 것이다.

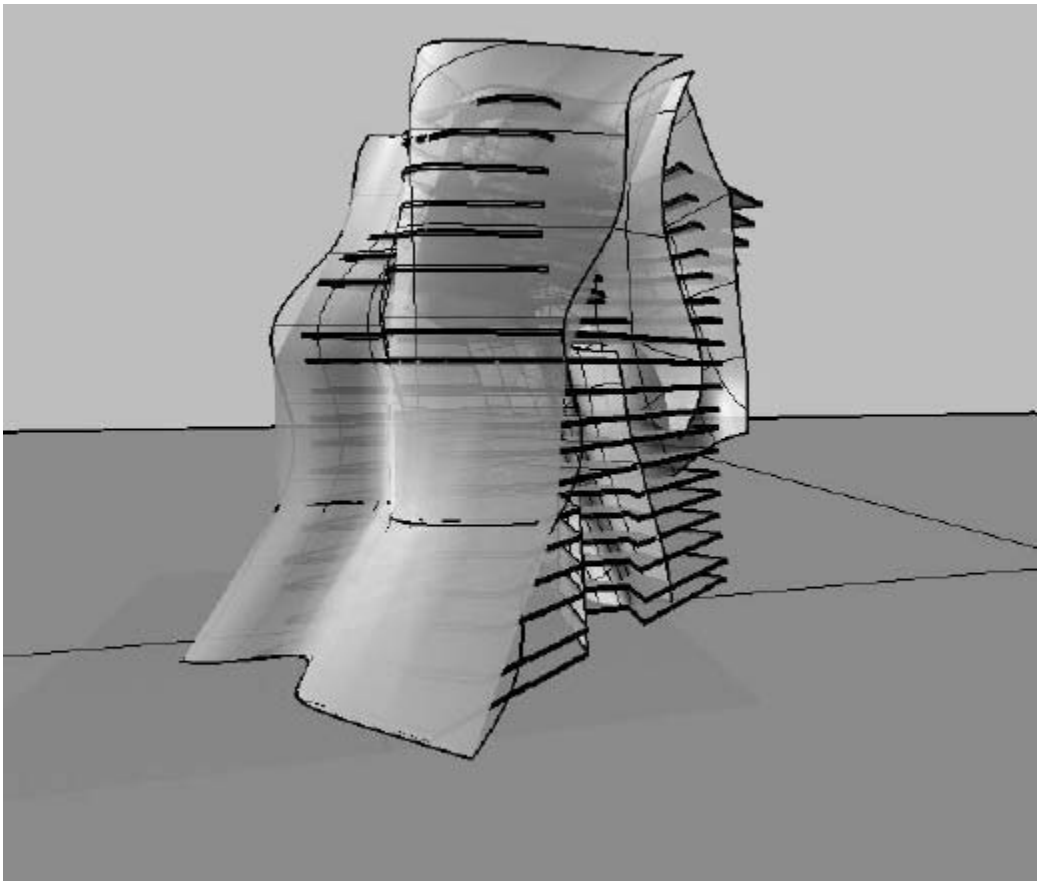
물론 이러한 개념은 일반 건물에도 적용된 것으로 완전히 새로운 것이라고 할 수는 없다. 하지만 기존의 입면과 다르게 그 구성을 다양하게 실험해보고 이것의 변화를 살펴볼 수 있을 것이다. 기본적으로 surface는 자유로운 형태를 가지고 있으며 각 지점에서의 방위가 모두 다르다. 즉 각 지점이 태양빛을 받게 되는 각도가 모두 다른 것이다. 이러한 입면의 구성은 마치 하나의 system을 만드는 것과 같다.



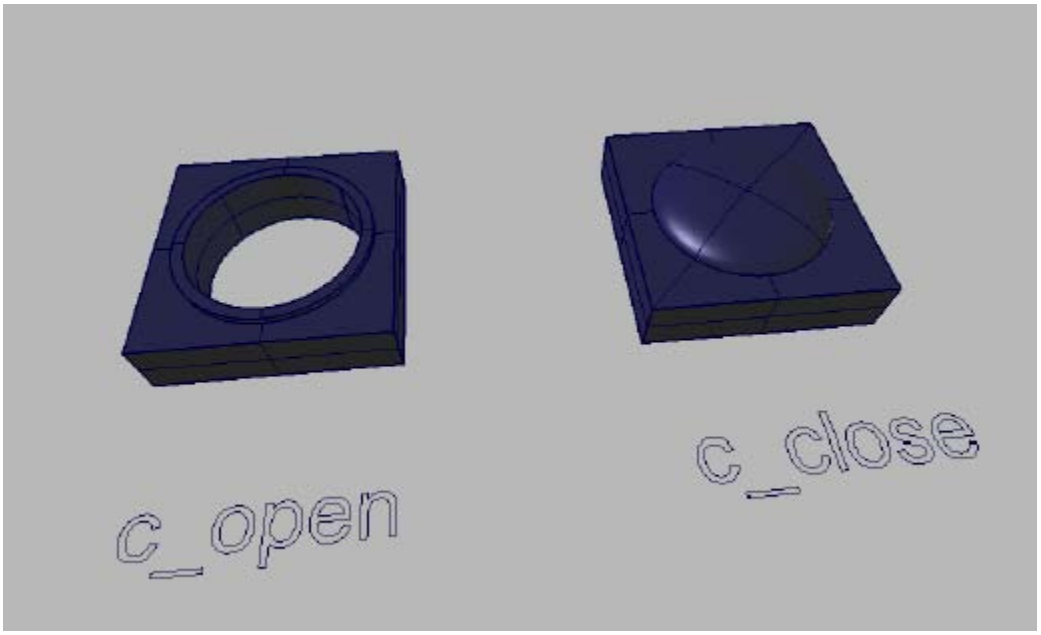
입면 system의 다양한 변화에 대한 개념

`

이것이 필요한 요소들은 다음과 같다.

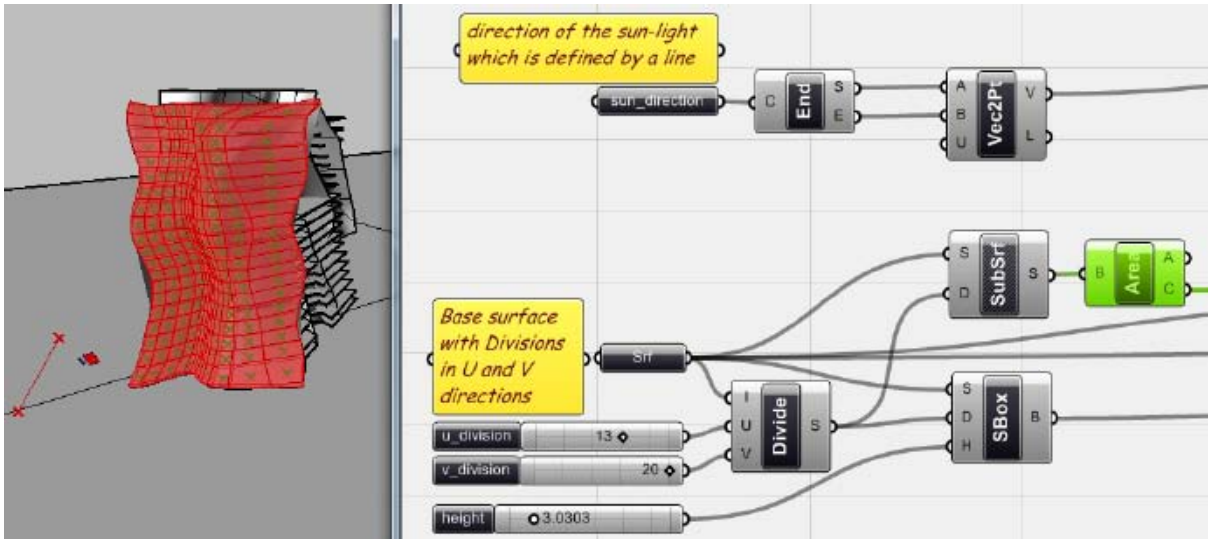


Panelization에 필요한 Surface



panelization에 사용되는 두 종류의 module

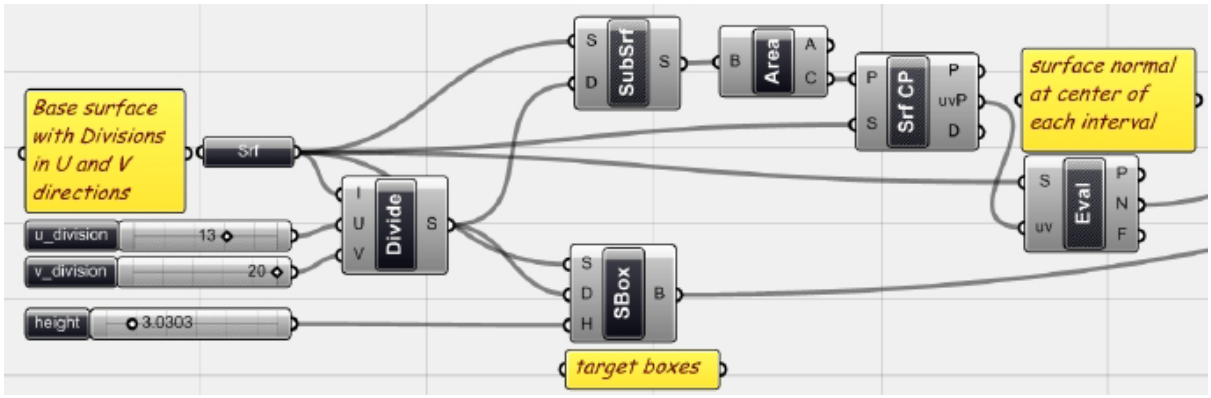
`



첫 번째 단계는 이전의 예제들과 유사하다. 먼저 <surface>와 <divide interval2>를 이용하여 UV 방향으로 surface를 나눌 개수를 정해준 뒤 <isotrim; subsrf>과 <surface box>를 이용하여 그 수에 맞는 box와 면을 각각 생성해준다. 또한 <isotrim>에 <Brep area>(위 그림에서 초록색으로 표시되어 있다.) 를 연결하여 각 면의 중심점을 찾아준다.

또한 rhino에서 선을 그린 뒤 이를 <curve>를 이용하여 grasshopper에 연동시킨다. 이는 태양의 입사각으로 사용 될 것이다. 이 <curve>에 <end point>를 연결하여 시작점과 끝점을 찾아주고 다시 이것을 <vector 2pt>의 시작점과 끝점으로 이용해준다. 이러면 선과 그 방향을 vector화 하여 태양의 방향을 정해줄 수 있다. 이제 rhino에서 그려진 선을 이동시키거나 회전 시키면 각 surface로 들어오는 태양의 입사각을 표현해줄 수 있다.

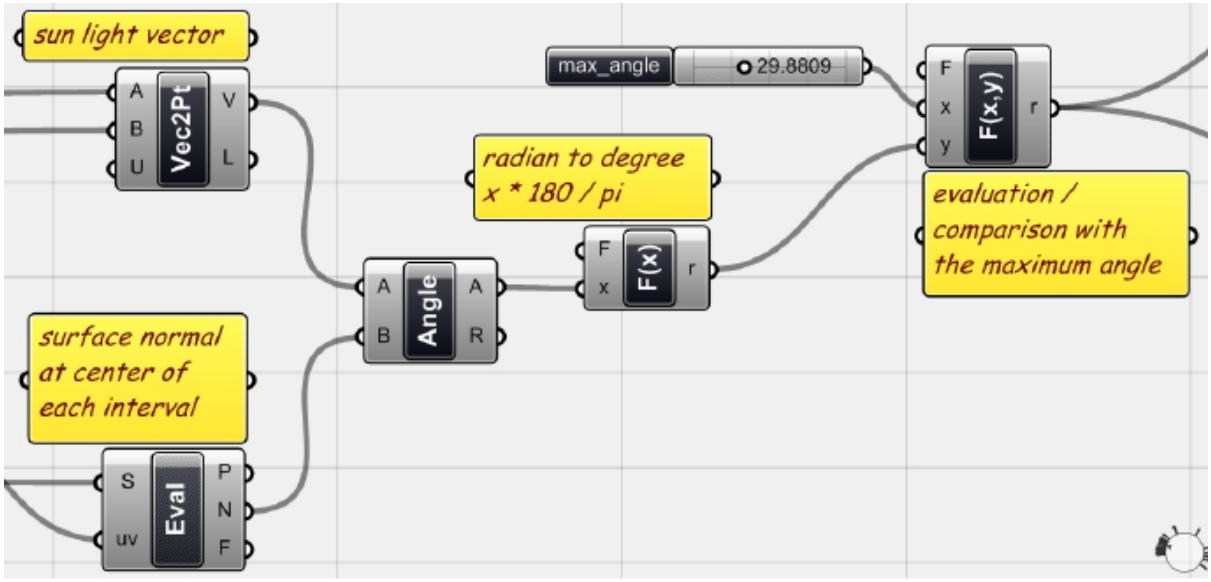
이 surface들로 들어오는 입사각은 바로 <vector 2pt>와 각 면들이 그 중심점에서 가지는 법선 vector(normal vector)에 의하여 계산할 수 있다. 법선(Normal)이란 surface위의 특정 점에 수직으로 만나는 vector를 의미한다. 각 면의 중심점 위에 생기는 normal값과 <vector 2pt>사이의 각도를 각 면에서의 입사각으로 사용해준다.



이 각도를 계산하기 위해서는 먼저 면의 중심점에서 생기는 normal vector들을 찾아줘야 한다. 이를 이용하여 잘려진 면의 각 중심점이 원래 <surface>상에서 가지는 UV값을 찾아줘야 한다.

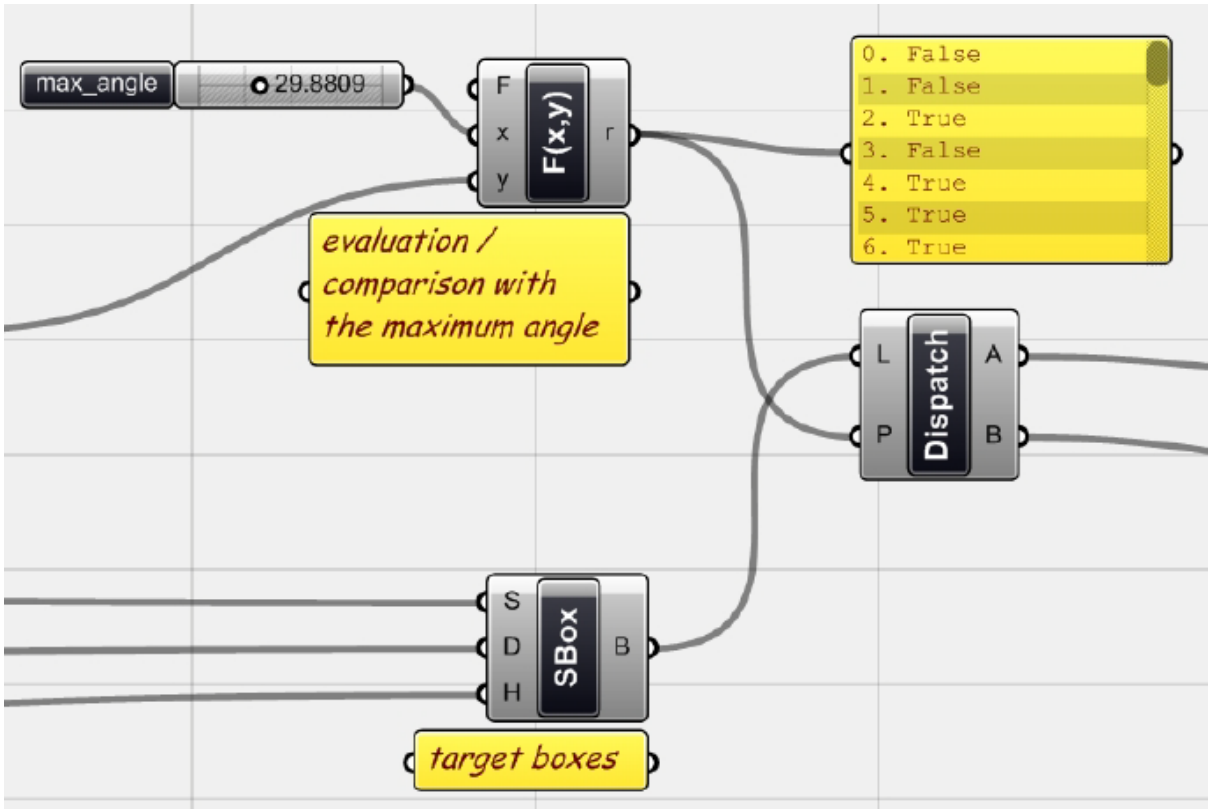
`

즉 <brep area>에서 나오는 중심점들 <surface>를 <surface CP>를 이용하여 data matching을 시키면 각 점의 UV값을 찾아줄 수 있다. 이제 다시 <evaluate surface>를 이용하여 앞에서 찾은 각 점의 UV값을 <surface>에 다시 적용시켜주면 이 점에서의 normal값을 찾을 수 있다.



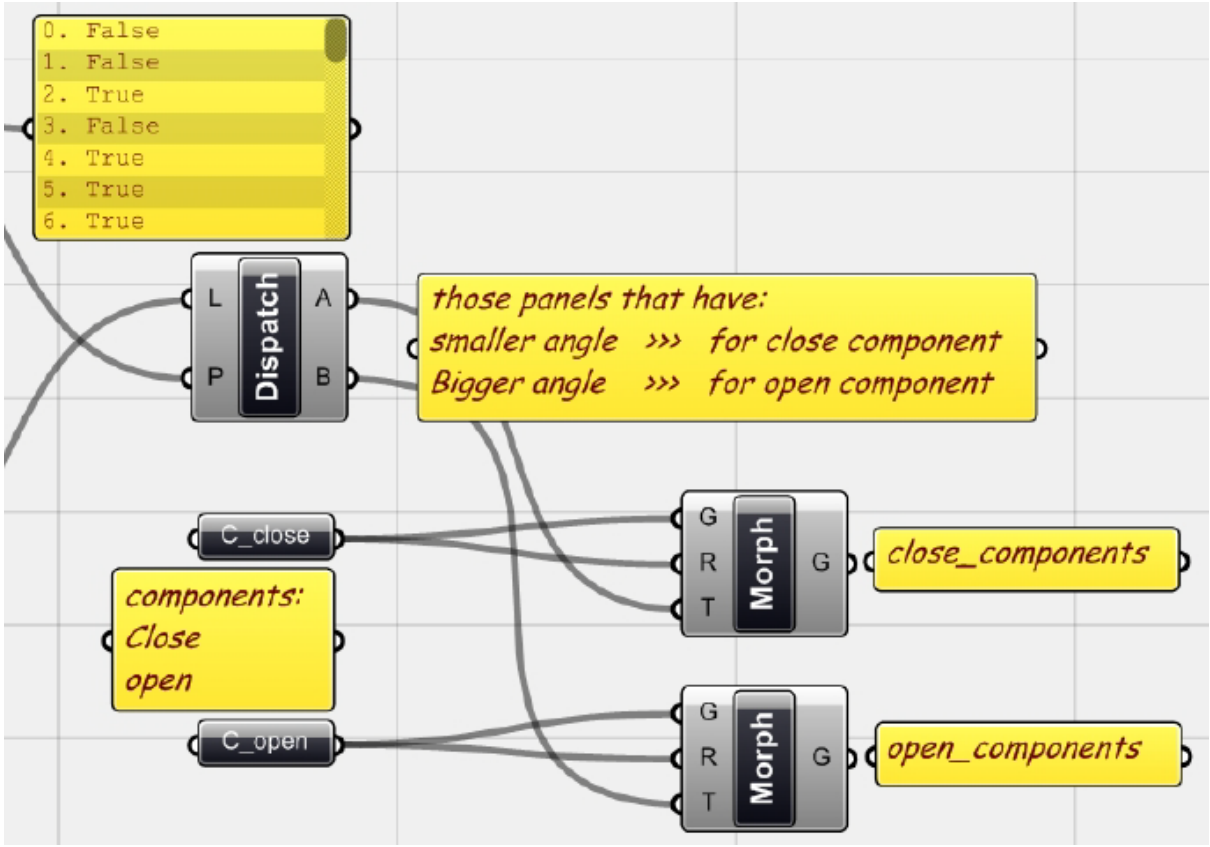
이제 이 <angle>(Vector > Vector > Angle)을 이용하여 태양빛의 입사 방향<vector 2PT>을 각 normal값에 data matching시키면 각각 잘린 surface들 중심에서 태양의 입사각을 찾아줄 수 있다. 이때 이 결과 값은 radian 값이다. 이것을 <function 1>을 이용하여 x*180/pi를 적용시켜주면 각도 값으로 변하게 된다. 이렇게 나온 값을 <function 2>에 x>y를 적용시켜준다. 이것의 x값에는 <number slider>를 연결하여 사용자가 원하는 최대 각도값을 입력해줄 수 있다. 이렇게 비교된 결과물은 boolean data로 나오게 된다.

`

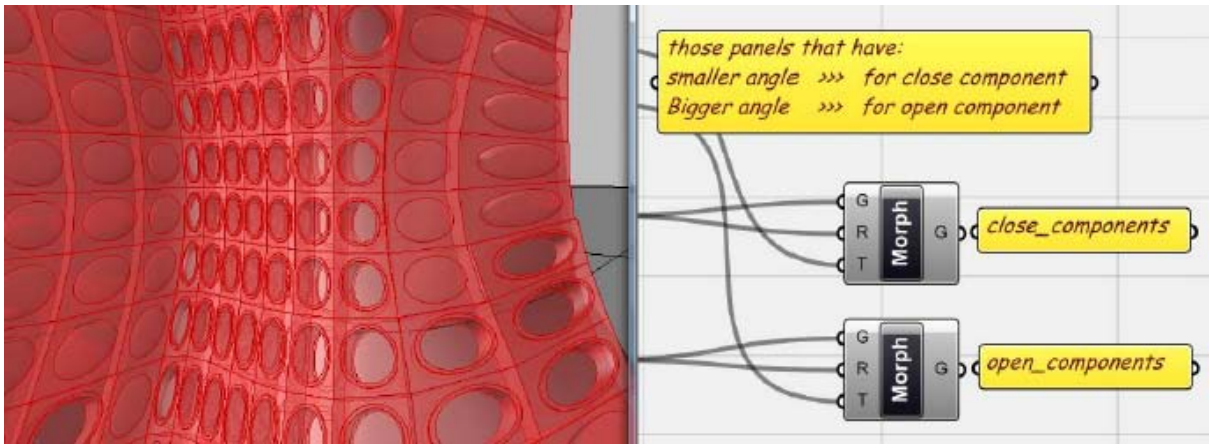


이렇게 나온 Boolean data의 pattern을 <dispatch>를 이용하여 <surface box>에 적용시켜주면 이 최대 각도 값을 기준으로 더 큰 입사각을 가지는 위치의 box와 이보다 더 작은 입사각을 가지는 위치의 box들을 나눌 수 있다. 나머지 algorithm은 우리가 이때까지 해왔던 그것과 크게 다르지 않다. <morph>를 두 개 가져온 뒤 앞에서 분류된 box들을 각각의 target box로 연결시켜준다.

`

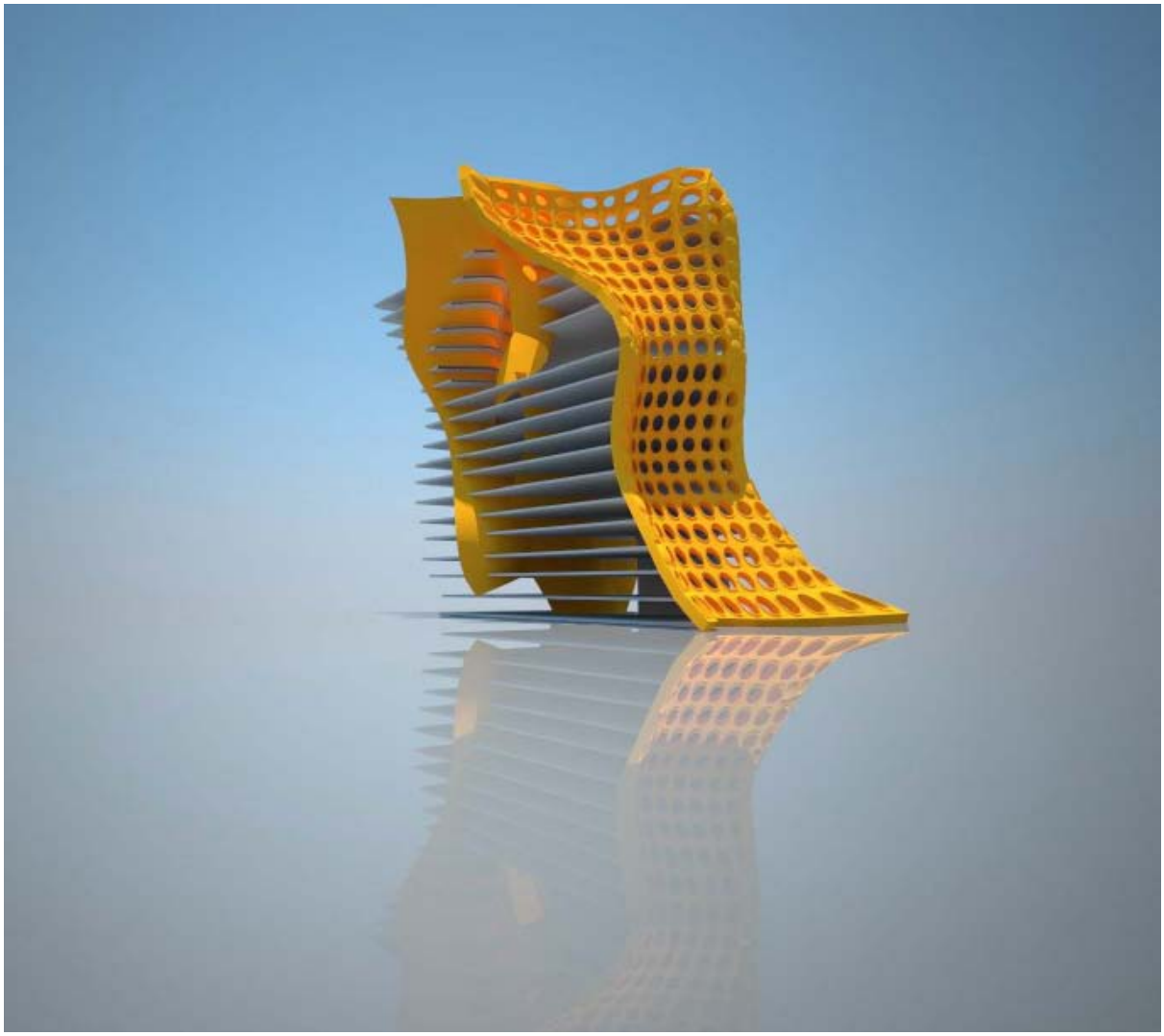


위의 그림처럼 입사각이 더 작은 경우에는 닫힌 component를 적용시켜 주고 더 큰 경우에는 열린 component를 적용시켜 준다.



그러면 위의 그림처럼 닫히고 열린 component들이 surface위에 적용되는 것을 볼 수 있다. 이러한 배치는 위에서 사용자가 설정한 논리에 의거하게 된다.

`



위는 마지막 모형의 그림이다. 이러한 algorithm을 응용하면 surface에 적용되는 module의 종류를 두 개 이상으로 늘릴 수 있다.

이러한 입면 module의 열리고 닫힘은 내부의 기능이나 태양 이 외의 다른 외부요인에 의하여 결정될 수 있다. 이러한 algorithm을 이용하면 면의 모든 위치에 같은 module이 적용되는 것을 피하고 각 부분에 적용되는 design에 변화를 줄 수 있다. 이제 surface를 조절하면 전반적인 형상을 조절할 수 있고 태양의 위치와 각도가 되는 curve를 조절하면 component의 적용 범위를 변화시켜줄 수 있다.

`

Chapter_7_NURBS Surfaces and Meshes

`

7_1_NURBS면의 매개변수(Parametric NURBS Surfaces)

전장에서 surface를 input으로 이용한 예제들을 살펴보았다. 이전 자에서는 loft와 pipe를 이용하여 surface들을 생성하였다. 또한 free-form surface를 input으로 하여 이것을 analysis 탭에 있는 component들에 연결하였을 때 나오는 결과물을 어떻게 활용할 수 있는지도 살펴보았다. 이처럼 많은 경우에 rhino상에서 그릴 수 있는 curve나 surface와 같은 기본적인 기하체들을 input으로 활용할 수 있다. 이처럼 Grasshopper를 잘 활용하기 위해서는 어떠한 시점에 어떠한 rhino input이 필요한지를 잘 판단할 수 있어야 한다.

벽체나 입면과 같이 surface로 구성되는 기하체가 보통 design의 결과물이 된다. 이러한 결과물을 내기 위해서는 process 중간중간에 필요한 curve와 점 등을 input으로 활용할 수 있어야 한다. 이번 예시에서는 surface에 관련된 다양한 component들을 살펴보고 이것들이 가지는 가능성들을 살펴보도록 하겠다. 이 경우 design process 나 결과물 자체 보다는 새로운 component를 활용해 본다는 것에 그 초점을 맞추도록 하겠다.

매개변수를 이용한 tower (Parametric Tower)

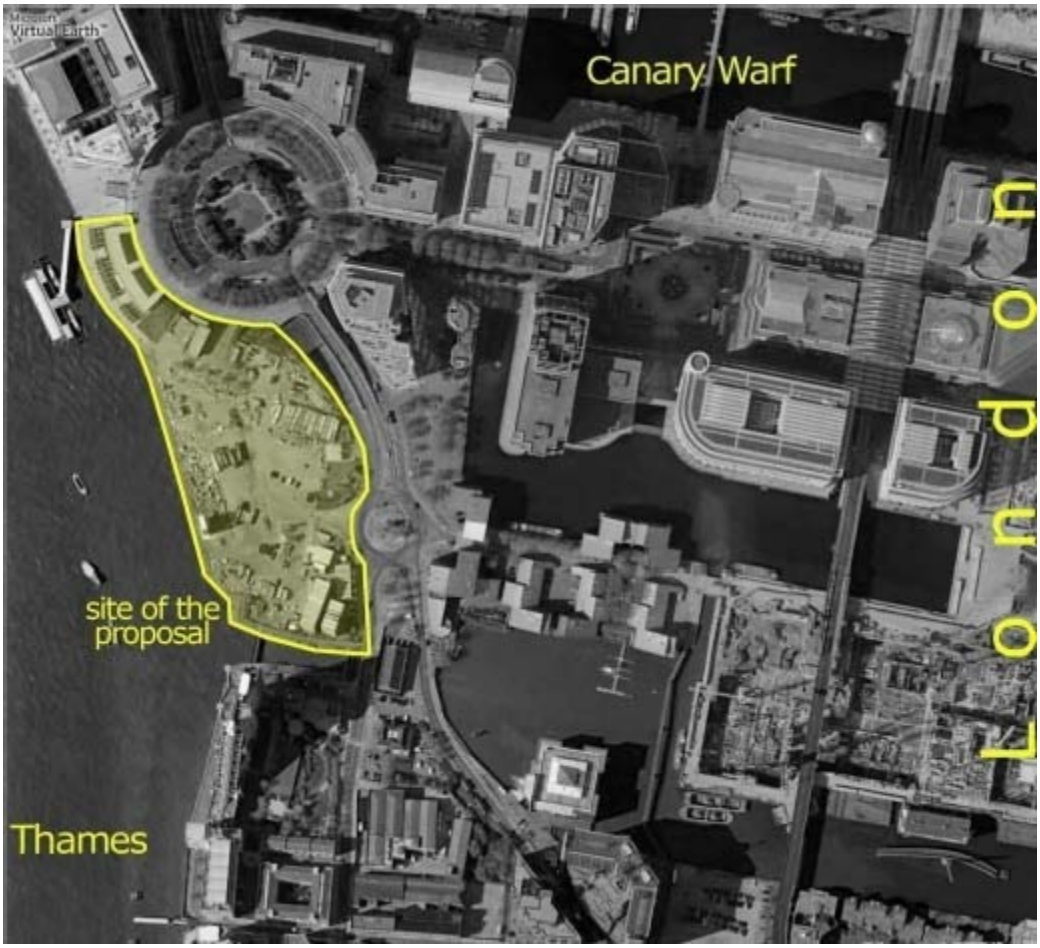
London의 Thames 강변에 있는 Docklands 지역을 site로 하는 tower를 제안하여 보자. 제안은 간단하면서도 기본적인 design을 제안할 것이다. 이 때 free-form surface를 활용할 수 있는 여러 가지 기본 idea들을 살펴보도록 하자.

먼저 지역을 살펴보자.

항공사진, Canary Warf, London (image: www.maps.live.com, Microsoft Virtual Earth).

Site는 Thames 강변의 둑 위에 위치하고 있다. 강쪽으로 좋은 조망권을 형성하고 있으며, Canary Warf (Westferry Road)의 중심부에 있는 광장의 입구에 면해있다. Site에 대한 이야기는 이 정도로 하고 형태적인 issue들에 대하여 이야기 해보도록 하자.

`



Site

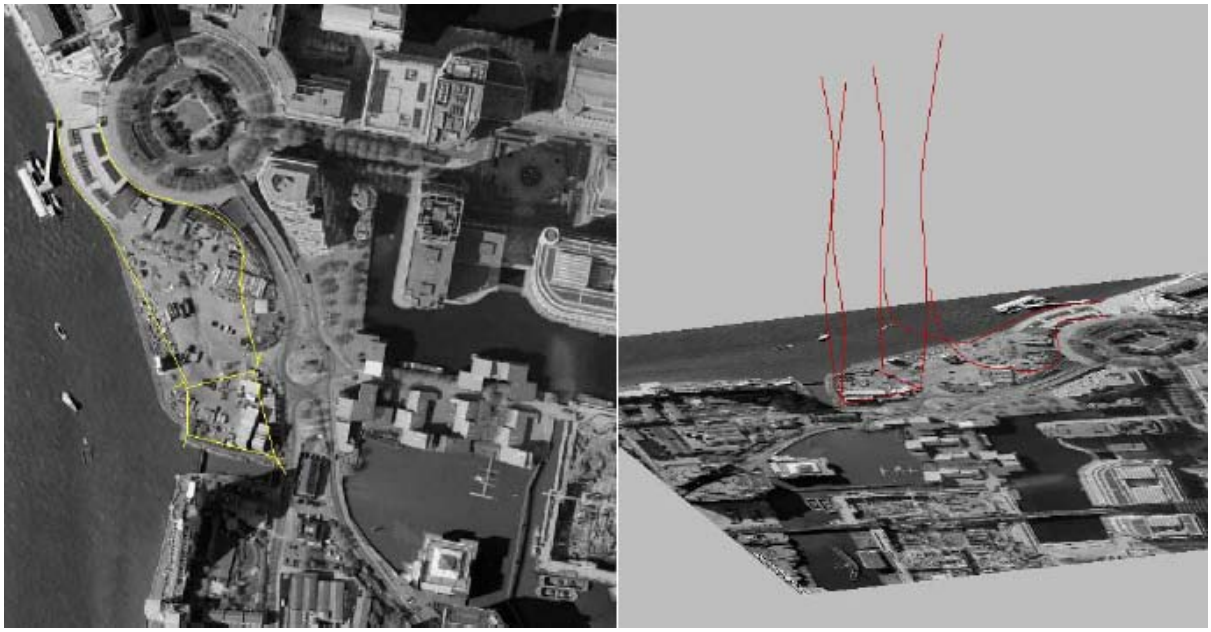
Rhino input의 생성(Manual drawings)

건물의 모형을 그리는 데에는 여러 가지 방법이 있다. 먼저 1층의 평면을 그린 뒤 그것을 수직 방향으로 필요한 만큼 복사하고 detail을 더해갈 수 있다. 이러한 방법이 활용된 예시는 다른 곳에서 쉽게 찾아볼 수 있다. 이 예시에서는 몇 가지 surface component를 이용하려고 한다.

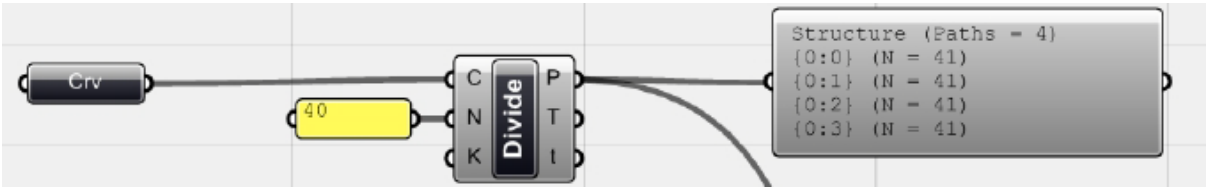
개념은 간단하다. Tower의 표면은 유리로 마감되며 그 바깥에 선적인 요소들이 더해지게 된다. 이 입면에 공간이 더해지게 되고 이 공간은 입면 위에 무작위로 배치된다. 이러한 공간들은 입면의 선적 요소들을 자르게 된다. 또한 이러한 입면의 선적 요소는 입면에서 그 주변의 강변까지 연결되며 public space를 형성하게 된다.

아래 그림에서 볼 수 있는 것처럼 먼저 rhino에서 기본적으로 필요한 curve를 그린다. 이 curve들은 site에서 필요한 요구사항들이나 그 형상, 경계 그리고 높이제한 등 다양한 조건에 의하여 조절될 수 있다. 이 중 두 개의 curve에 각각에 인접하는 또 다른 curve를 하나씩 그린다. 이 두 curve들은 지상에서부터 시작하여 건물의 최고층까지 연결되게 된다. 이러한 curve의 개수나 형상은 이후 사용자가 원하는 대로 바꿔줄 수 있다.

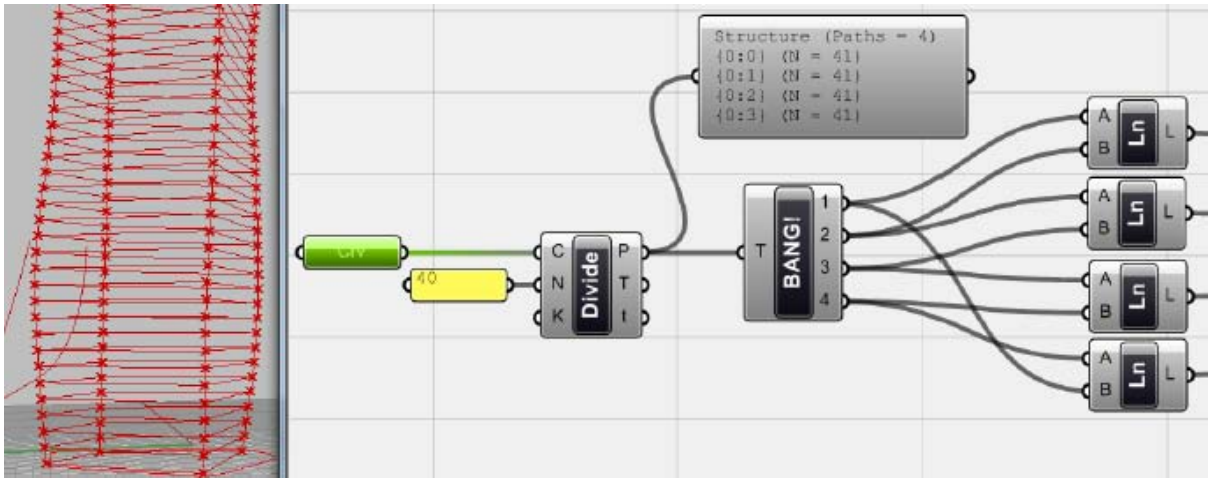
`



기본적인 입면 요소(Basic façade elements)



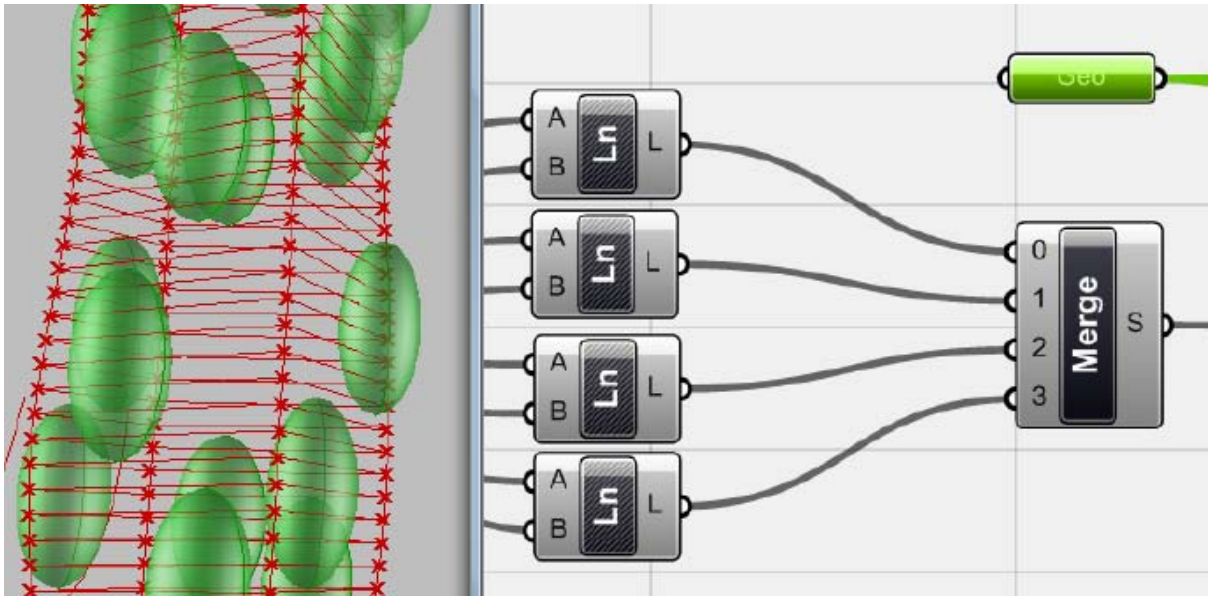
먼저 해야 할 일은 위에서 그린 rhino상의 curve를 grasshopper에 연결하는 것이다. <curve>를 이용하면 된다. 이것을 다시 <divide curve>에 연결하고 N에 40을 줘서 위에서 연결된 curve들이 40등분 한다. 위를 보면 각 curve 위에 그려진 41개의 point들이 4개의 다른 가지(branch)에 들어 있는 것을 알 수 있다.



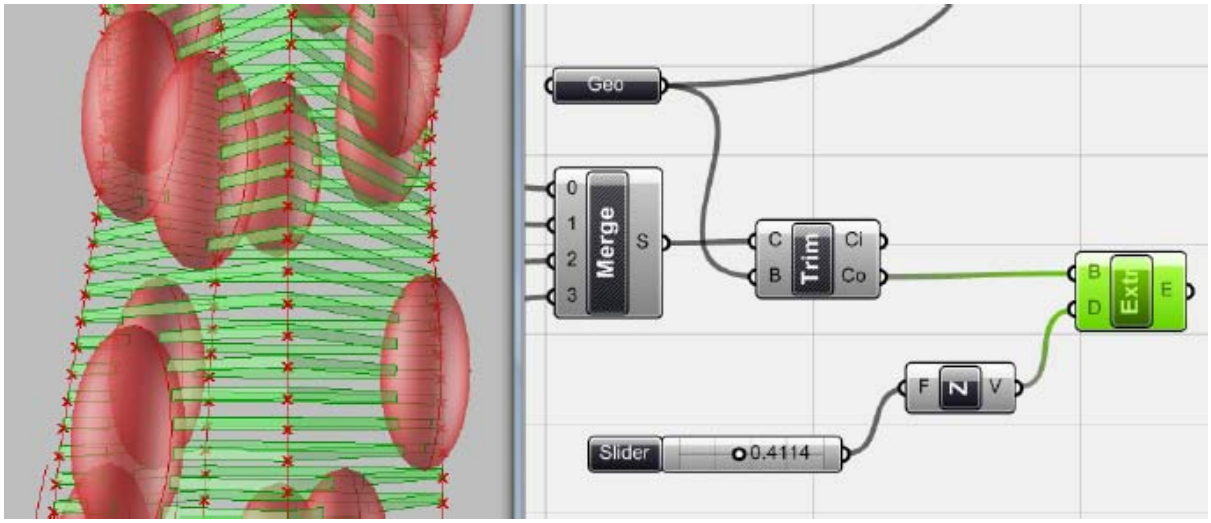
이제 이것들을 연결하여 입면의 기본적인 요소가 되는 line을 그려준다. 이를 위해서는 먼저 각 branch안에 있는 점들을 각각의 data list에 분류해주어야 한다. 이러한 역할을 하는 것이 <explode tree> <division curve>(Logic > Tree > Explode Tree)이다. 이것을 이용하여 각 branch 들을 data list화 한 뒤 <line>을 이용하여 하나의 branch 안에 있는 point들을 그 옆 branch의

`

point들과 연결하면 점을 그릴 수 있다. 이것을 반복하여 위와 같이 line을 생성해준다.

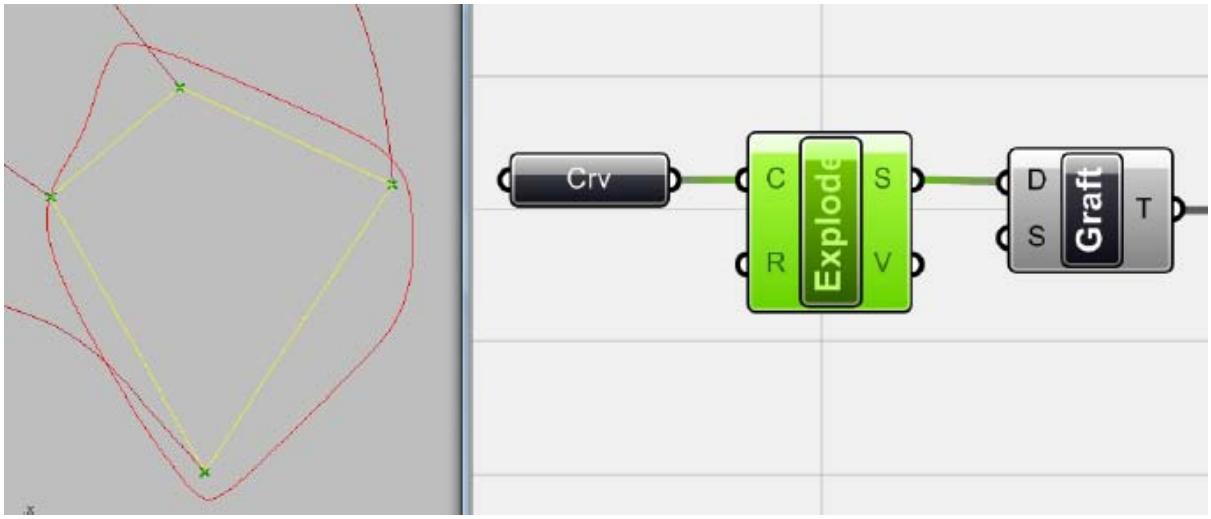


이제 위에서 생성된 기본 입면 line에 이와 겹치는 기하체들을 그려주는 것이다. 위처럼 타원체를 구려 둔 뒤 입면 위에 무작위로 배치한다. 이것을 <geometry>를 이용하여 grasshopper에 연결시켜 준다. 그리고 위에서 생성된 <line>을 <merge multiple>을 이용하여 모두 합쳐주게 된다.



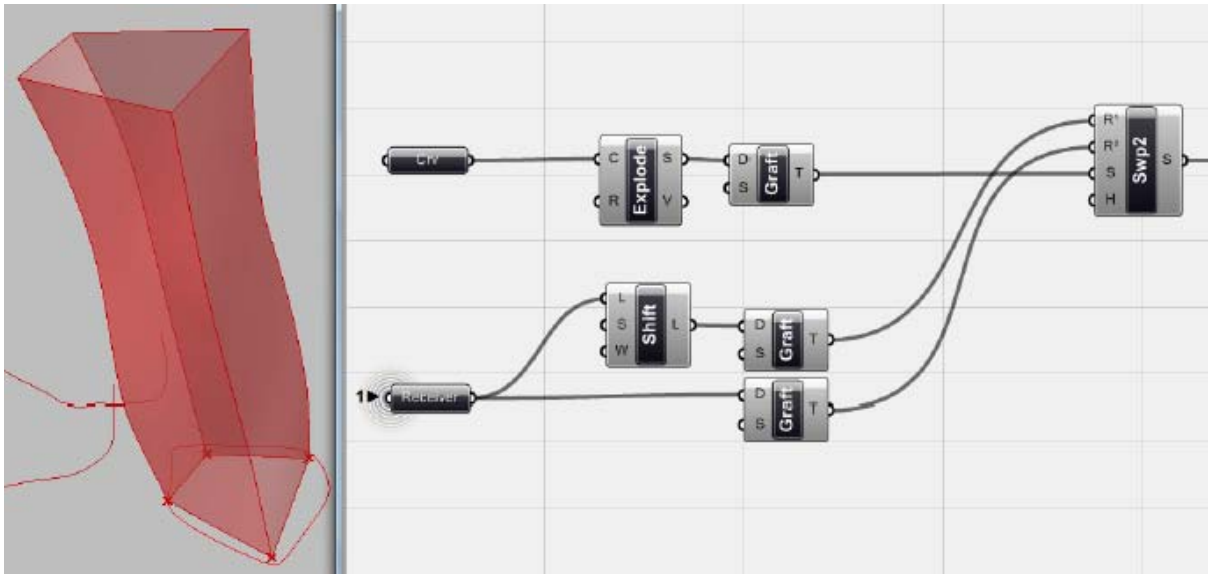
이 다음 단계에서는 < trim with Breps >을 이용하며 <merger>된 모든 line을 <geometry>에 연결된 타원체를 이용하여 잘라준다. 이때 <trim with Breps>의 output은 curve inside, 즉 타원체의 내부에 있는 line부분과 curve outside 즉 타원체의 바깥에 있는 line부분으로 나누어지게 된다. 이제 타원체 바깥에 있는 line들을 원하는 만큼 <extrude>시켜주자.

`



이제 이 선들을 감싸는 면을 그려보자. 이 때 이용할 component는 바로 <sweep2>이다. 이것에는 section curve와 두 개의 rail curve를 필요로 한다. 먼저 이에 필요한 section curve를 그려보자.

rhino로 돌아가서 각 curve의 끝점을 연결하는 polyline을 그려준다. 이것을 <curve>를 이용하여 grasshopper에 연결시켜 준 뒤 다시 <explode>를 이용하여 위의 polyline들을 최소한의 선 단위로 나누어 준다. 이 결과물은 네 개의 line-like curve이다. 이를 <graft>를 이용하여 각 line-like curve가 하나씩 들어있는 네 개의 data list를 만들어 준다.

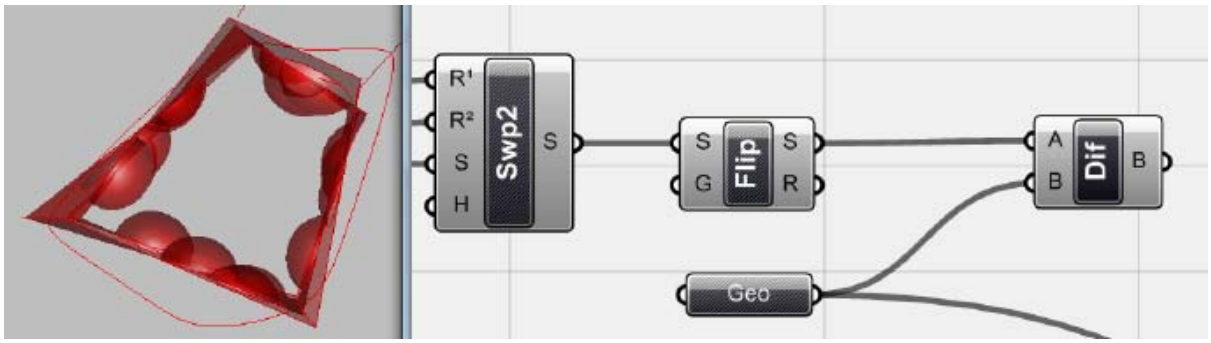


Rail curve로는 앞서 사용한 네 개의 curve를 이용하면 된다. 이것을 바로 <graft>를 한 것과 <list shift>를 한 뒤 Graft 한 것을 이용해주면 된다.[46] 이제 이것들을 위와 같이 <sweep2>로 연결하면 면이 생기게 된다.

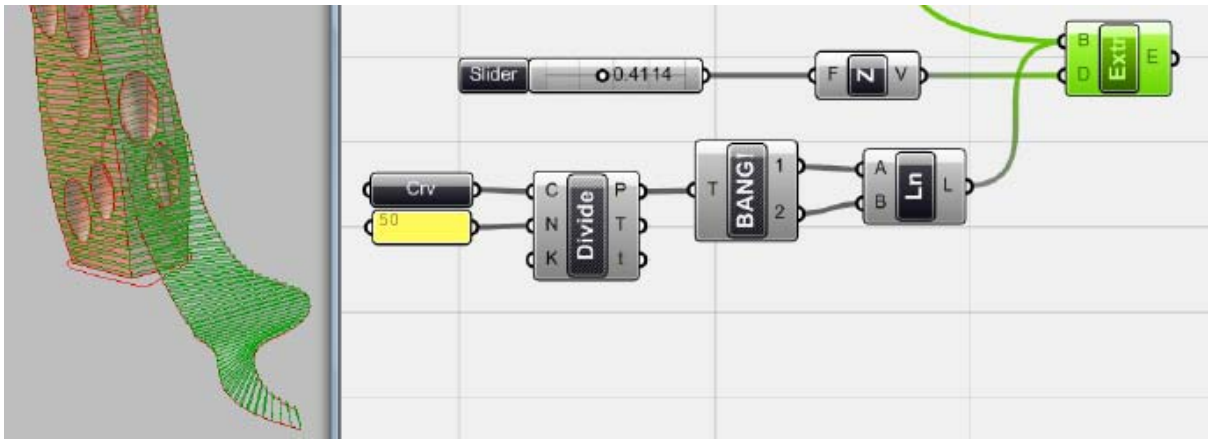
[46] 역자 주: 이 때 section curve가 양 끝에 있는 rail curve와 data matching 될 수 있도록 curve의 선택 순서에 유의 하여야 한다. 또한 <shift list>의 W를 우클릭 하여 boolean을 true로 만들어 준다.

`

주의할 것은 만일 결과가 위의 그림과 같지 않다면 rail curve가 data list에 선택된 순서와 section curve의 순서가 일치하지 않기 대문이다. 이럴 때는 제일 처음 <curve>에서 네 개의 curve를 선택할 때 그 순서를 바꿔주거나 <explode>의 S에서 나오는 curve의 list를 <shift list>를 이용하여 그 순서를 바꿔준다.

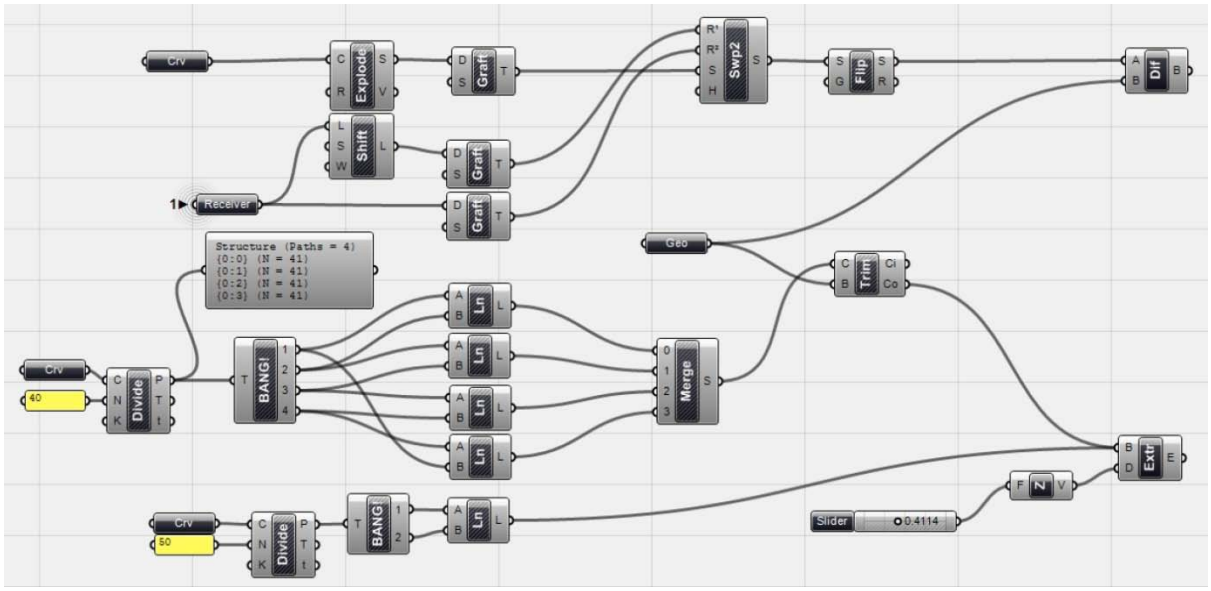


이제 이렇게 생성된 면들을 이용하여 타원체들을 잘라보자. 이를 위해 주의해야 할 것은 바로 면의 방향(normal vector)이다. 이것에 따라 잘리고 남는 면이 달라지게 된다. <flip>을 이용하면 면의 방향을 뒤집을 수 있다. 이것을 <solid difference> (Intersect > Boolean > Solid Difference)에 연결하여 surface 내부의 타원체들을 남기도록 하자.

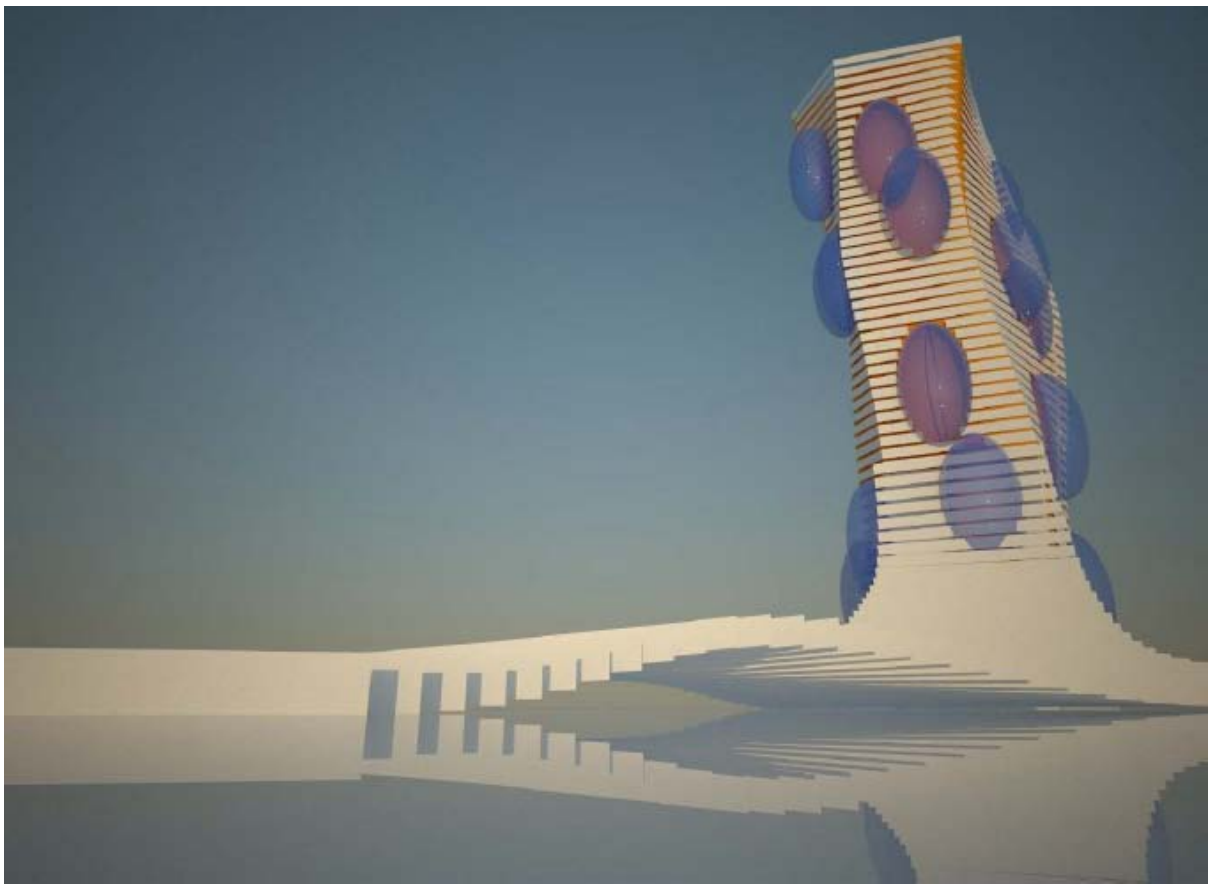


위의 방법과 마찬가지로 curve를 연결하여 두 개의 curve를 그린 뒤 이것을 grasshopper에 연결하고 나누어 선을 그리고 입면을 그어준다. 이것을 <extrude> 시켜주면 기존의 입면과 같은 효과를 줄 수 있다.

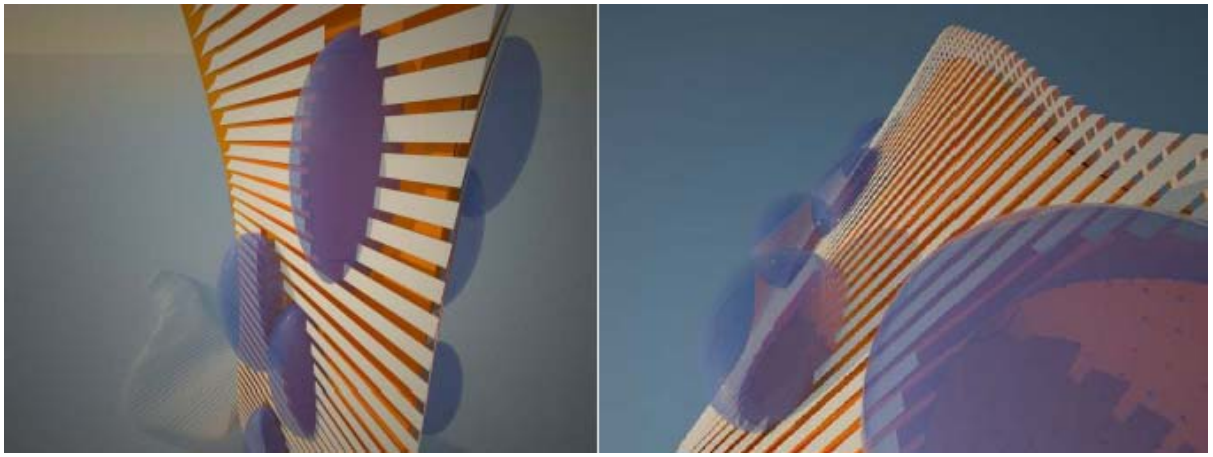
`



모든 기하체를 생성한 뒤 <difference>와 <extrude>를 bake 하면 된다.



`



마지막 모델

`

7_2_기하학과 위상수학(Geometry and Topology)

이때까지는 NURBS에 적용되는 component들에 대하여 살펴보았다. Grasshopper에는 이 뿐만 아니라 다른 종류의 surface에도 적용 가능한 여러 component들이 있다. NURBS가 그려내는 부드러운 곡면이 꼭 원하는 결과물이 아닐 수 있다. 혹은 좀 더 미세한 제어와 간단한 함수에 의한 처리가 가능한 surface를 그리고 싶을 수도 있다. 예를 들면 B-spline이나 Besier surface 같은 경우가 있을 수 있다. 이번 장에서는 Mesh에 대해서 다루고자 한다.

Mesh란 free-from surface의 한 종류로 아주 작은 면(face)들이 모여 하나의 surface를 형성하는 것이다. 이러한 surface에는 그 내부에 고유한 규칙을 가지고 있다거나 수학적 함수에 의거하지 않는다. 대신 각 face가 모여 전반적 surface의 형상을 결정한다. 즉 하나의 Mesh surface 에는 각 face를 만들어내는 점(grid of point)이 분포되어 있고 이러한 각 face는 삼각형, 사각형, 혹은 육각형일 수도 있다. 이러한 점들이 mesh surface를 형성하는 기본 단위이며 이 점들은 모두 연결되어 있다.

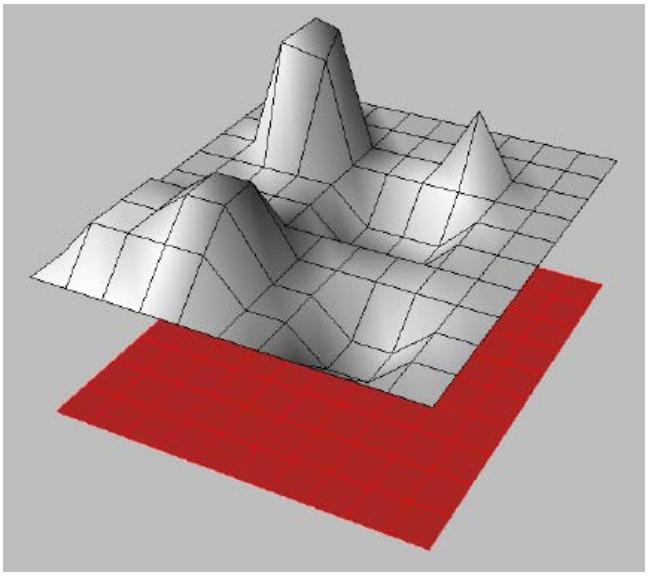
Mesh가 가진 가장 중요한 두 가지 특징으로는 바로 mesh를 형성하는 점들의 위치와 각 점들 간의 연결이다. 점들의 위치는 각 face가 가진 면의 형상을 결정하고 이것들의 연결성은 topology와 연관된다.

Geometry VS. Topology

기하학(geometry)가 공간에서 객체의 위치를 결정짓는다면 위상수학(topology)는 각 객체간의 관계를 결정하게 된다. 수학적으로 말하자면 topology란 객체가 가진 한 특징으로서 이것은 그 객체를 변환(transformation)하거나 변형(deformation)하여도 변하지 않는 것이다. 즉 원(circle)과 타원(ellipse)은 위상수학적(topologically)으로 같은 것으로 이 둘은 기하학적(geometrical) 차이만을 가지게 된다. 아래 그림을 살펴보면 각 네 개의 point들이 서로 연결되어 있는 것을 볼 수 있다. 왼쪽 두 개의 그림을 살펴보면 A와 B는 이 둘은 각기 같은 방식으로 연결되어있기 때문에 같은 topology를 가지고 있다고 말할 수 있다. 하지만 이 둘은 기하학적으로 같지 않은데 이는 한 점의 위치가 다르기 때문이다. 반면 오른쪽에 있는 두 A 와 B는 같은 위치에 점을 가지고 있지만 다른 방식으로 연결되어 있기 때문에 이 점들은 기하학적으로 같고, 위상수학적으로 다른 형상이라고 할 수 있다.

이러한 topology의 개념은 mesh를 이해하는데 매우 중요한 역할을 한다. 각 mesh의 face들은 그 모서리에 point를 가지며 각 face들의 corner point들은 각자 같은 순서와 방식으로 서로 연결된다. 이러한 mesh를 변환(transformation)하게 되었을 때 각 face의 모서리 점들(vertices)의 위치가 변하게 된다. 각 점들이 가지는 변환의 정도 또한 각기 다를 수 있다. 그러나 각 vertex들이 연결되는 방식은 유지되게 된다.

`



회색 surface나 빨간색 surface 모두 mesh로 같은 face와 모서리 점들(vertices)을 가진다. 단지 회색 surface의 vertices들이 다른 위치를 가지게 되면서 mesh의 기하학적 형상(geometrical configuration)을 바꾸고 있을 뿐이다. 하지만 각 mesh들이 같은 방식으로 연결 되고 있으므로 두 mesh surface는 위상수학적(topologically)으로 같다고 할 수 있다.

이러한 mesh 객체들의 위상수학적인 측면을 이해한다면 point들의 뭉치가 있을 때 그것을 연속하는 mesh surface로 표현할 수 있게 된다. 또한 여러 algorithm을 이용하여 point를 조절하여도 이 mesh surface는 유지되는데, 이는 바로 각 점들이 가지는 topology 때문이다.

예를 들어 Dynamic relaxation이나 particle system과 같이 유한한 요소를 분석하는 것에는 이것은 mesh를 이용하여 작업하는 것이 다른 종류의 surface를 이용하는 것보다 훨씬 쉬운데 이러한 함수들은 mesh vertices의 위치를 변화시키면서도 그 연속성은 유지되기 때문이다.

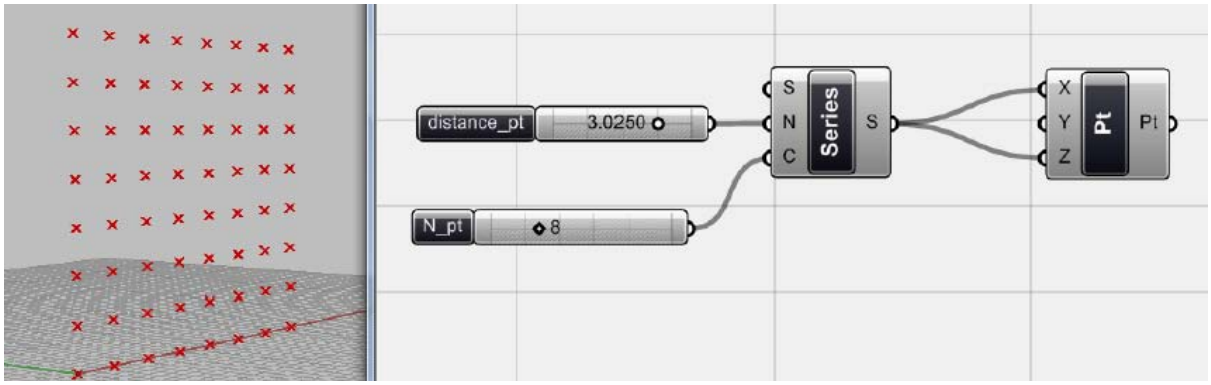
또한 mesh surface는 그 위에 구멍을 뚫거나 면을 불연속(discontinuity)하게 처리할 수 있기 때문이다. 또한 여러 가지 algorithm을 이용하여 mesh를 다듬고 이를 더욱 부드럽게 만들 수 있다. 또한 mesh는 각 face가 다른 색을 가질 수 있기 때문에 분석의 결과를 표현하는데 유용하게 사용할 수 있다.

Grasshopper의 Mesh 탭을 살펴보면 mesh를 조작할 수 있는 여러 개의 component를 찾을 수 있다. 이제 mesh를 처음부터 다뤄보고 이것에 대한 이해를 넓혀보도록 하자.

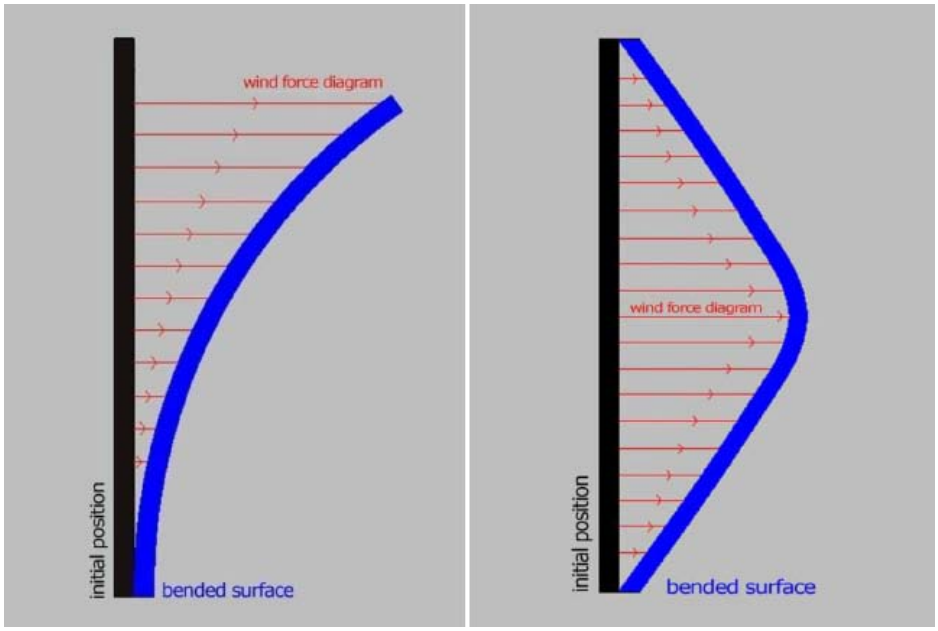
7_3_On Meshes

Point로 이루어진 격자가 있다. 이제 이 point들을 이용하여 면을 만들어보자. 이러한 point grid가 surface를 구성하고 있고 이 surface가 위로 갈수록 바람(혹은 vector를 가진 여러 종류의 힘)의 영향을 많이 받는다고 가정해보자. 그렇다면 이러한 영향은 각 point가 받는 것이라고 볼 수 있다. 이러한 point의 위치 변화는 surface를 변형시킬 것이다. 또한 풍량을 조절하여 그 형상을 바꿀 수 있을 것이다.

`



첫 번째 단계는 무척 간단하다. <series>와 <number slide>를 이용하면 점의 위치와 그 간격을 조절 할 수 있다. 이<number slide>는 각각 <N_pt>와 <distance_pt>로 그 이름이 변경되었다. 이 중 <N_pt>는 'even number'가 되도록 설정하자. 이 때 <point>의 data matching 방식을 'cross reference'로 바꾸는 것을 잊지 말자.

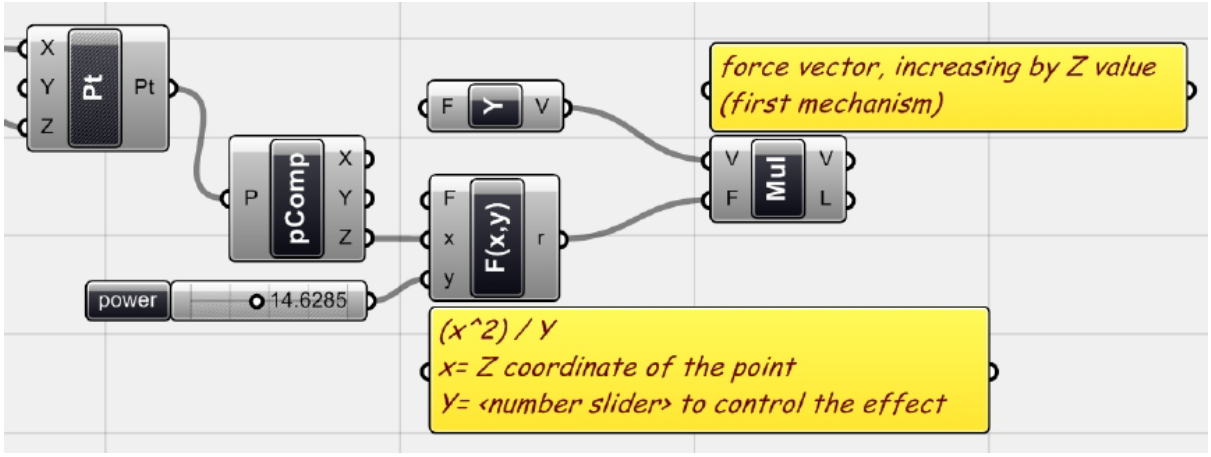


위는 바람의 힘과 그것의 영향을 받는 surface의 관계를 나타내는 diagram이다. 왼쪽은 수직 방향으로 생기는 힘과 그 영향이고 오른쪽은 위에서 보았을 때의 그것이다.

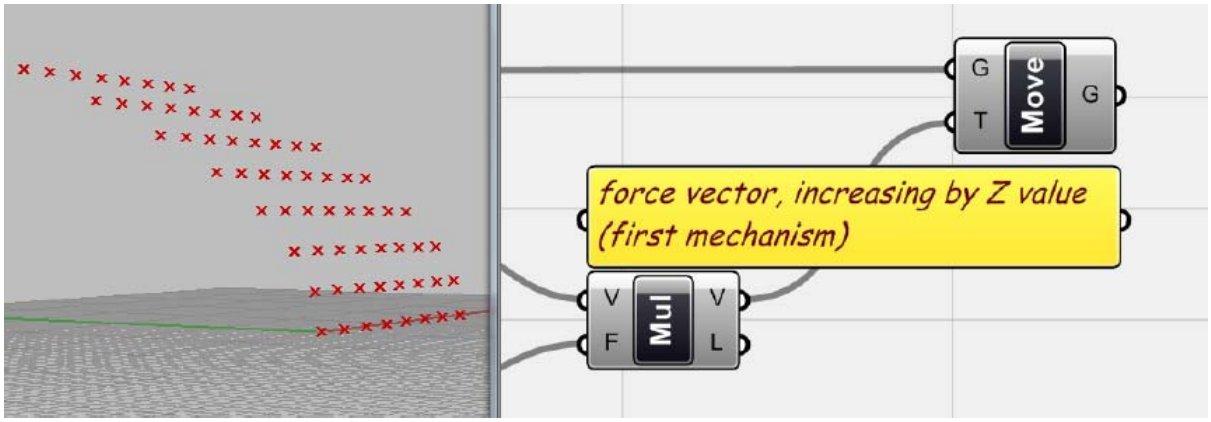
위 두 가지를 이용하면 바람의 영향을 modeling할 수 있다. 첫 번째 것은 간단한 수학 방정식 (X^2)로 표현 될 수 있다. 이 때 변수 X는 바로 각 point의 Z 좌표값을 적용하게 된다. 이 <point>를 <point decompose>에 연결하여 각 점의 Z좌표값을 추출하자.

Y 축 방향으로 풍력이 작용한다고 가정해보자. Grid 위의 각 점들은 위에 있을수록 더 큰 z값을 가지기 때문에 위에 있는 점이 더 큰 영향을 받게 되는 것을 알 수 있다. 즉 Z좌표값에 비례하는 Y방향의 vector를 만들고 이것을 각 점에 적용시켜 Y방향으로 각 값만큼 이동시키는 것이다.

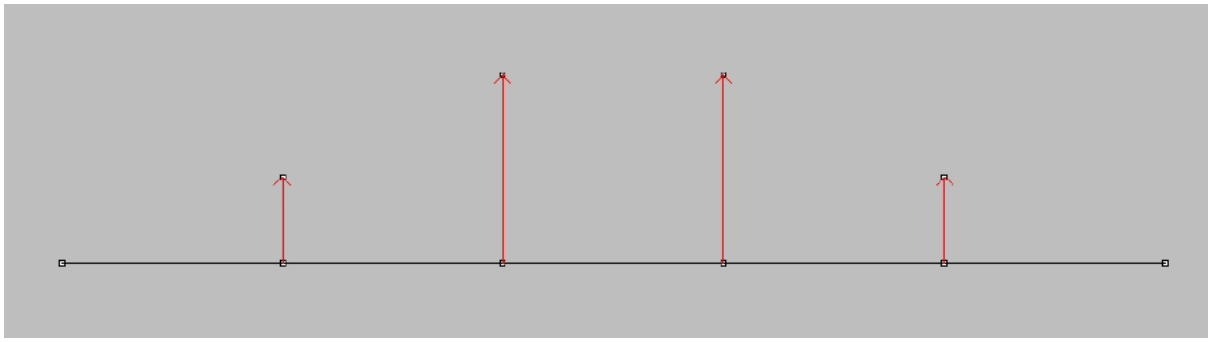
`



<decompose point>에 의하여 추출된 Z좌표 값은 <f2;(x^2)>에 의하여 제곱이 되고 다시 <number slider>에 의하여 그 영향의 정도를 조절되게 된다. 그 결과물은 <multiply>(Vector > Vector > Multiply)를 이용하여 <unit Y>에 곱해지게 된다.



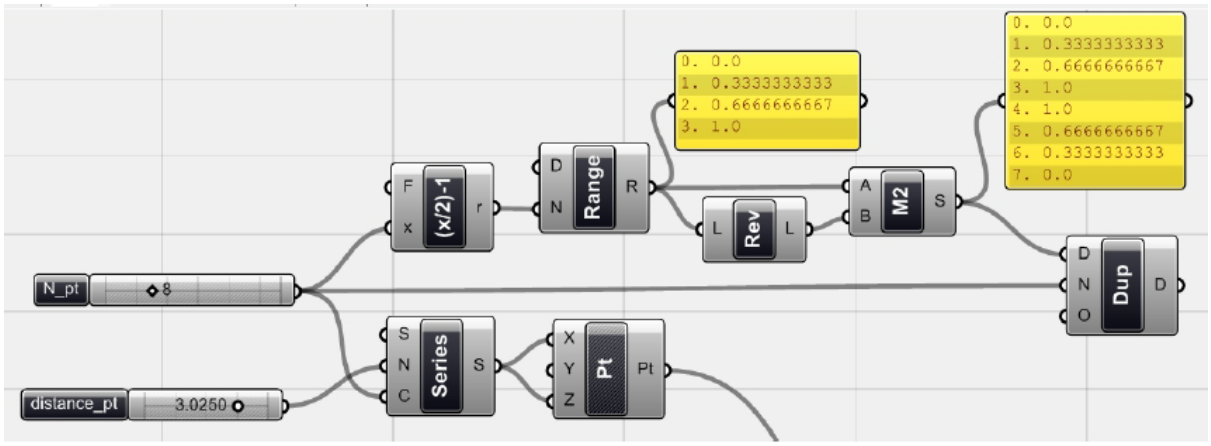
이렇게 만들어진 vector를 이용하여 점을 이동시키면 위와 같이 우리가 원하는 결과물을 얻을 수 있다. 이제 두 번째 mechanism을 적용시켜 보자. 위에서 언급한대로 평면상에서 보았을 때 가운데 있는 부분이 바람의 영향을 더욱 많이 받게 된다. 아래 그림은 각 열(row)위에 있는 점들이 어떻게 이동하는 지를 보여주는 것이다.



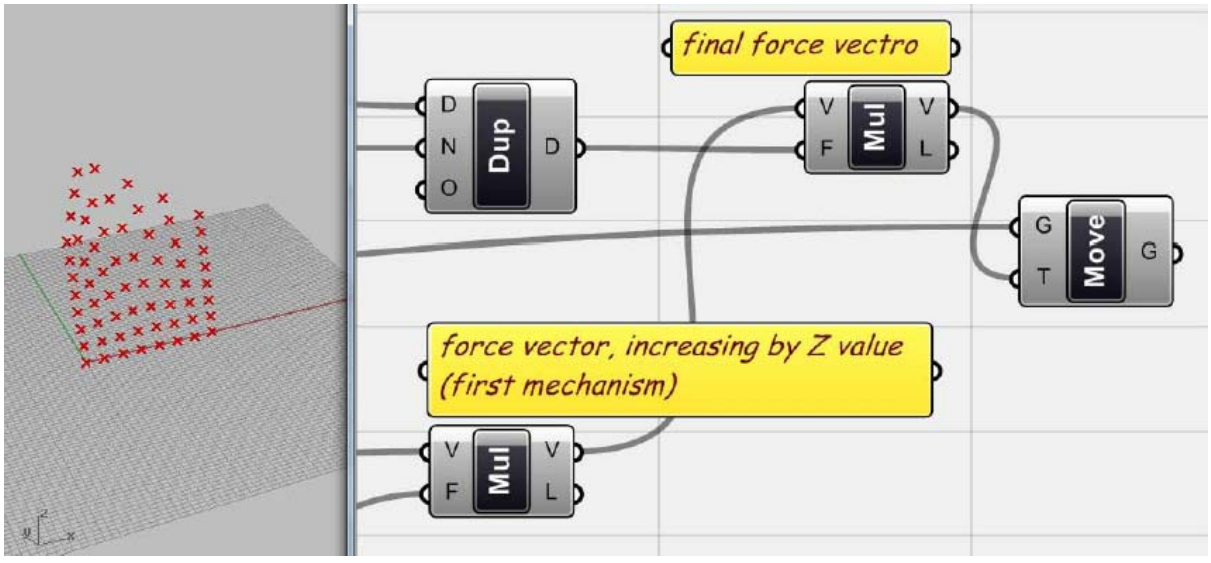
열 위에서 점 위치의 이동

위에서 적용된 vector값에 다시 인수의 data list를 생성하고 이를 곱한 vector값을 이용하여 point들을 이동시켜주면 된다. 즉 오른쪽과 왼쪽 양 끝에 있는 점의 vector에는 0을 곱하면 vector값이 0이 되기 때문에 원래 자리에 남게 된다. 나머지 점들은 가운데 쪽으로 갈수록 더 큰 값을 곱해주면 위와 같은 결과물을 얻을 수 있다. 그러한 인수의 data list를 만드는 방식은 아래와 같다.

`



먼저 <range>를 이용하여 0을 처음 값, 1을 마지막 값으로 가지는 data list를 생성해준다. 즉 0은 양끝 쪽에, 1은 가운데 쪽에 있는 점의 vector에 적용되는 것이다. 이를 위해서는 <range>를 활용하면 된다. 먼저 D에 들어있는 정의역(domain)의 범위를 '0 to 1' 로 정해준 뒤 이것을 <N_pt>를 <f1: (x/2)-1>에 연결하여 점의 개수를 2로 나누고 다시 1만큼 빼준다. <range>가 만들어내는 data list의 길이는 N에 들어오는 수보다 항상 1만큼 크기 때문이다. 이 경우 0과 1사이의 수를 3등분 하게 되면 (0, 0.333, 0.666, 1)이라는 수를 가지게 된다. 다시 이것을 <reverse>를 이용하여 data의 순서를 뒤집으면 (1, 0.666, 0.333, 0)이된다. 이 두 list를 <merge2>를 이용하여 합쳐주면 (0, 0.333, 0.666, 1, 1, 0.666, 0.333, 0) 이라는 data list를 생성할 수 있다. 위의 그림에 panel을 살펴보면 쉽게 이해할 수 있을 것이다. 이제 이것을 <duplicate>를 이용하여 복사를 해주게 된다. 이때 이것에 들어오는 N값은 <N_pt>를 이용해주면 된다. 이 경우 점은 8개의 X좌표값과 8개의 Z좌표값이 조합된 64개가된다. 위에서 생성된 data list가 8개이고 여기에 다시 8을 곱하면 점의 개수만큼의 data list를 생성할 수 있다.[47]

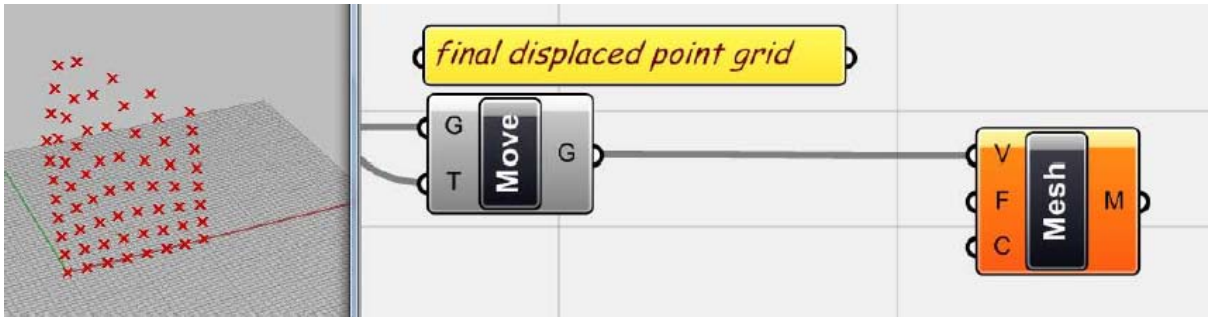


[47] 즉 (0, 0.333, 0.666, 1, 1, 0.666, 0.333, 0, 0, 0.333, 0.666, 1, 1, 0.666, 0.333, 0, 0, 0.333, 0.666, 1, 1, 0.666, 0.333, 0, 0, 0.333, 0.666, 1, 1, 0.666, 0.333, 0, 0, 0.333, 0.666, 1, 1, 0.666, 0.333, 0, 0, 0.333, 0.666, 1, 1, 0.666, 0.333, 0, 0, 0.333, 0.666, 1, 1, 0.666, 0.333, 0, 0, 0.333, 0.666, 1, 1, 0.666, 0.333, 0) 이라는 data list가 생성된다. 이것이 '행(row)'의 방향으로 반복되는 점에 적용된다.

`

이번 예시의 경우 약간의 분석적인 사고를 요구하는 것이 사실이다. 현실에서는 Particle Spring System이나 Finite Element Analysis 의 경우 복수개의 vector가 점 전체에 영향을 끼치며 점들 사이에도 각자의 힘이 서로에게 영향을 끼치게 된다. 즉 하나의 surface을 만드는 point grid가 있을 때 여기에 하나의 힘이 작용하면 이것은 모든 점에 영향을 끼치게 될 뿐만 아니라 그와 동시에 점들끼리도 서로 영향을 끼치게 되는 것이다. 이러한 과정은 반복적인 loop을 거쳐 계산되며 이것의 결과물이 전체(system)내에서 각 점의 위치를 결정하게 된다. 이 예제의 경우 각 점이 서로에게 끼치는 영향을 배제하고 매우 간단한 수학적인 계산을 통하여 각 점이 받는 힘만을 고려한 것이다. 현실에서의 영향은 이것보다 훨씬 더 복잡함을 알아두도록 하자. 여기서는 이러한 결과물을 'mesh'를 이용하여 표현해보는 것이 그 목적이다. 이제 이 점들을 이용하여 mesh를 생성해보자.

메쉬(Mesh)

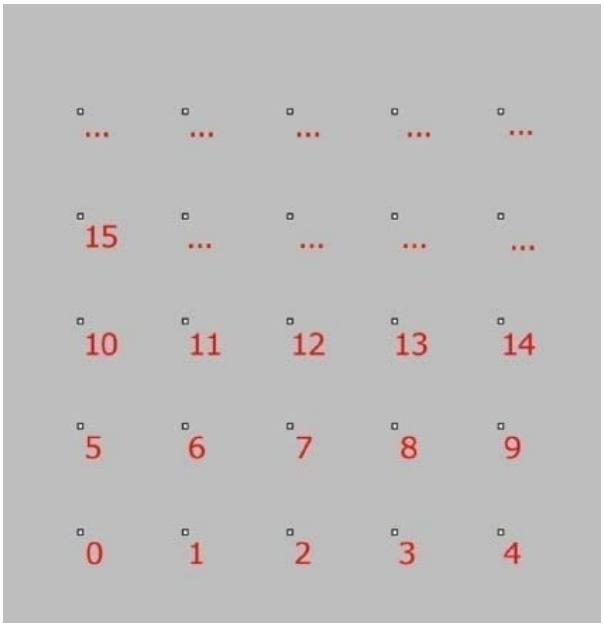


먼저 <mesh>(Mesh > Primitive > Mesh)를 불러온 뒤 이것에 위에서 생성한 점들을 연결하여 준다. 하지만 이것으로 mesh가 생성되는 것은 아니다. Mesh surface를 생성하기 위해서는 이것을 구성하게 될 face정보를 제공해줘야 한다. 이 정보는 일련의 수로 이루어진 data list로 이것은 각 vertex[48]가 연결되는 방식을 결정해주고 이것을 통해 surface를 생성해주게 된다. 즉 이 vertices는 mesh의 기하학적인 면을 결정해준다. 하지만 이 뿐만 아니라 위상학적인 정의(topological definition)을 해줘야 mesh를 생성할 수 있는 것이다.

Point grid위에 있는 각 점들을 4개씩 연결하여 사각형의 면(quadrant face)을 생성하여보자. Point grid를 살펴보면 각 point들이 고유의 index number를 가지는 것을 알 수 있다. 즉 각 점의 좌표값을 알고 그것을 이용하지 않아도 이 index number를 이용하면 점을 다룰 수 있게 된다.

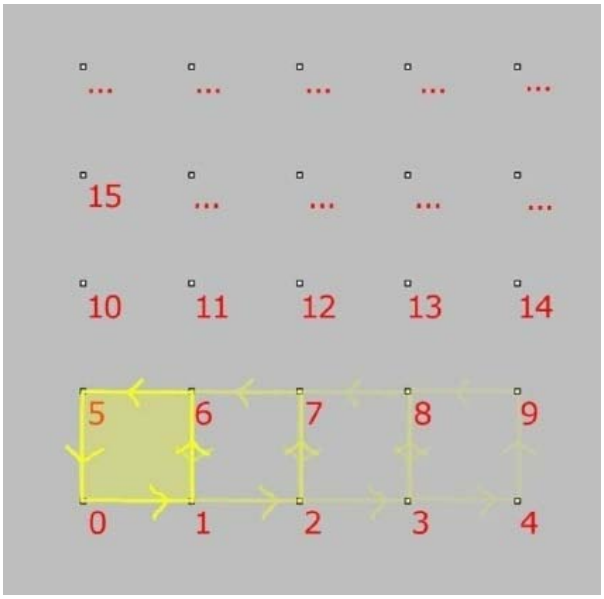
[48] Vertex는 꼭지점을 의미한다. 그 복수형이 vertices 이다. 참고는
http://ko.wikipedia.org/wiki/%EA%BC%AD%EC%A7%80%EC%A0%90
http://en.wikipedia.org/wiki/Vertex_(geometry)

`



각 점들이 가지고 있는 index number

Mesh를 정의 하기 위해서는 각 face들이 가지는 네 개의 vertex을 정의해주어야 한다.

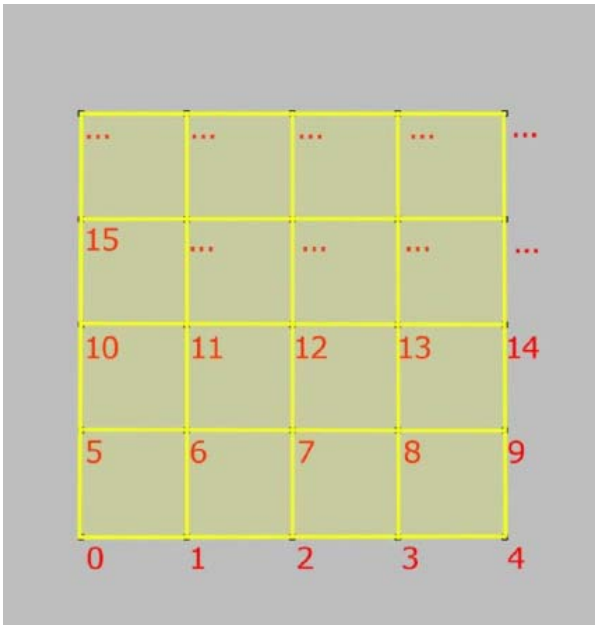


위에서 보이는 것과 같은 순서로 점들을 연결해주면 quadrant face을 생성해줄 수 있다. 이 순서는 바로 하나의 점으로부터 출발하여 같은 행(row) 위에 있는 오른쪽 옆에 있는 점, 그 다음은 같은 열(column) 바로 위에 있는 점, 그 다음은 같은 행(row)바로 왼쪽 옆에 있는 점, 그리고 그 시작점으로 돌아오게 된다. 여기서 가장 처음에 생기는 face에 필요한 점은 바로 [0, 1, 6, 5]이다. 그 다음 face는 [1, 2, 7, 6]이다.

Mesh surface 전체를 정의하기 위해서는 위와 같은 순서로 점을 연결하는 algorithm을 짜야 한다. 위의 matrix를 살펴보면 **n** 번째 점이 있을 때 그 다음 점의 index number는 바로 **n+1** 이다. 그 다음 점에는 n에 열(column)의 개수 c를 더해주고 다시 1을 더해주면 된다. 즉 **n+c+1**로 위 경우 c=5이다. 그 다음 점은 **n+c**가 된다.

`

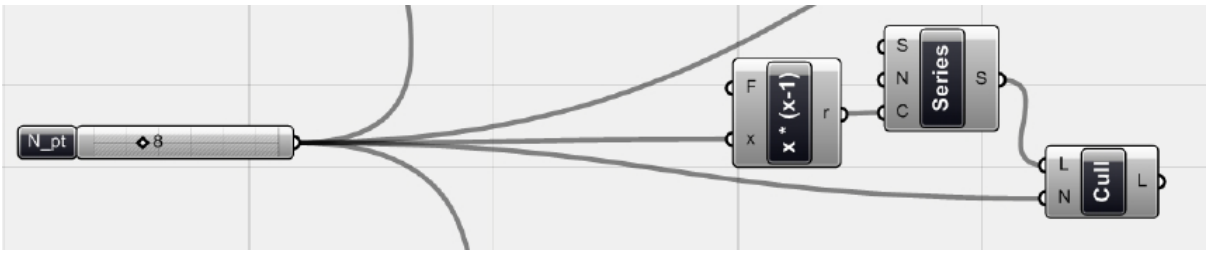
즉 어떠한 점의 index number n이 주어졌을 때 quadrant face가 가지는 vertex의 index number는 [n, n+1, n+c+1, n+c] 이다.



각 face를 정의해주면 mesh가 생성되게 된다.

위의 그림을 보면 위처럼 vertex의 index number를 정의해준다 하더라도 약간의 문제가 있음을 눈치챌 수 있을 것이다. 이 전에 다룬 삼각형 pattern을 만드는 예제를 떠올려보자. 마지막 열과 마지막 행위에 있는 점들은 면의 시작점이 될 수 없다는 것을 알 수 있다. 즉 여기서 필요한 것은 mesh의 face를 생성하기 위한 algorithm 뿐만 아니라 face의 '시작점'이 될 수 있는 점과 그렇지 않은 점을 구분해주는 data management가 필요하다.

간단하게 말해서 마지막 열과 마지막 행을 제외한 나머지 점들이 각 면의 시작점으로 정의해주면 된다. 즉 이에 해당하는 점들의 index number를 제외하는 수의 data list를 만들어야 한다.

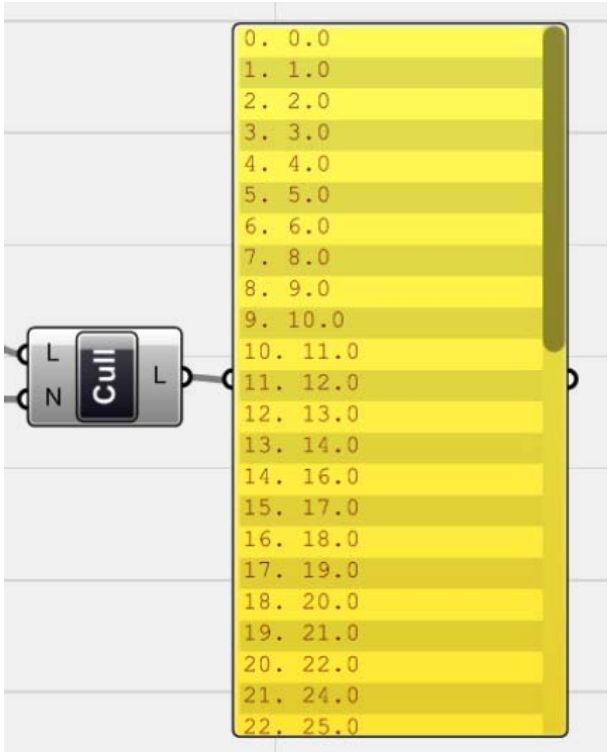


위에서 언급한 data list를 생성해주기 위한 algorithm이다. 기본적으로 <series>를 이용해주면 된다. <N_pt>에서 나오는 수는 row와 column의 수와 같다. 이것을 <f1: x*(x-1)>를 이용하면 이것은 열(column)의 수 * 행(row)의수 -1 과 그 의미가 같다. 즉 이렇게 되면 마지막 row 만큼의 index number를 생성하지 않게 된다.[49] 즉 이것은 columns*(rows-1) 와 같다. 이제 이것을 <cull Nth>에 연결하여 n번째 수를 모두 버려주자. 그러면 오른쪽 열 위의

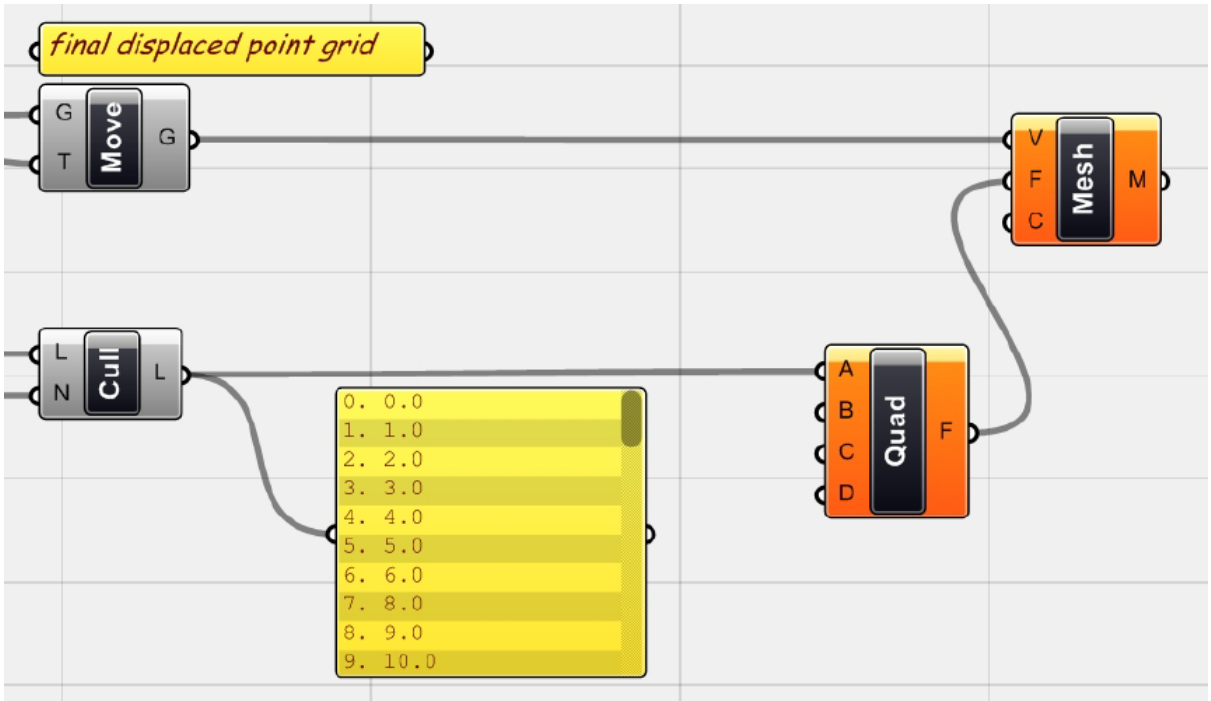
[49] Data는 원칙적으로 U방향으로 생성됨을 잊지 말자. 이 경우 U 방향은 X축 방향, row 방향 으로 생각할 수 있다.

`

점의 index number에 해당하는 수를 버릴 수 있다.



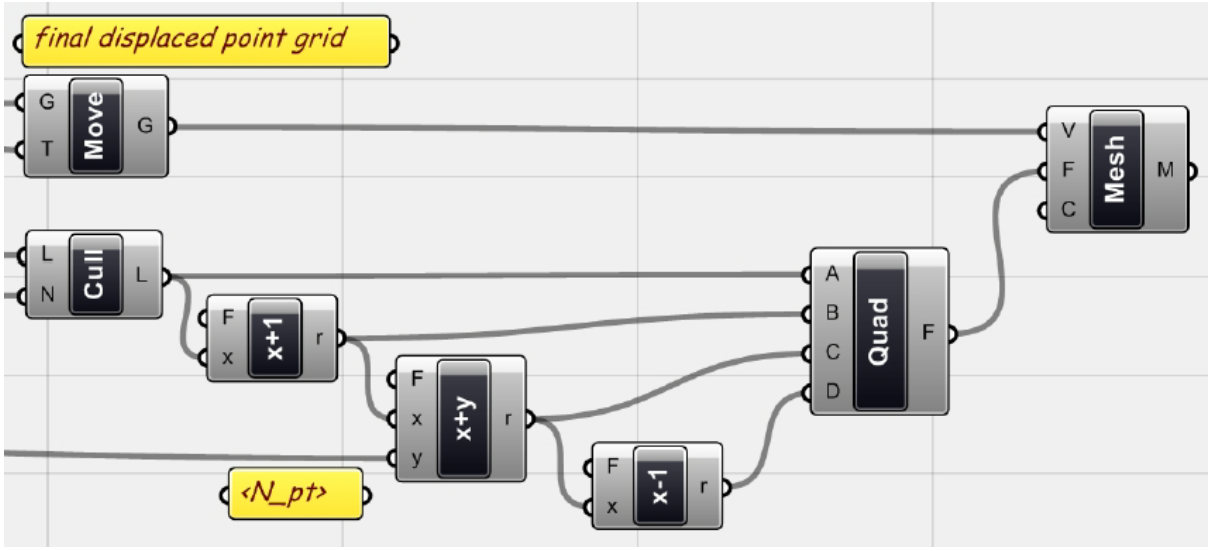
이렇게 생성된 수의 data list는 위와 같다. 이제 이것을 index number로 가지는 vertex들을 골라보자.



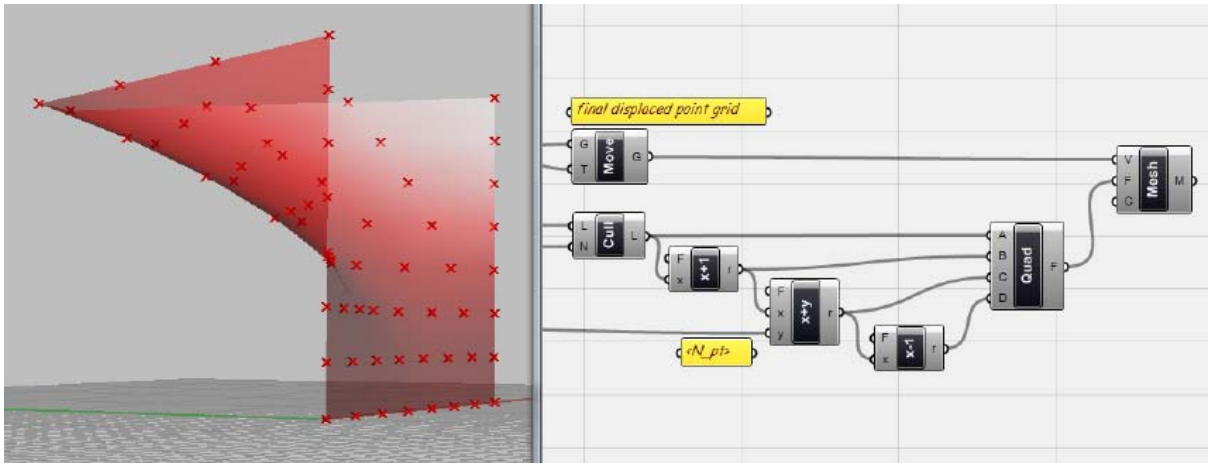
<Mesh quad> (Mesh > Primitive > Mesh quad)는 Grasshopper에서 mesh의 face를 생성하는 역할을 해준다. 이것에 각 면의 첫 번째 vertex에 해당하는 index number를 A에 넣

`

어준다.



두 번째 점은 n+1과 같으므로 <f1: x+1>을 적용시켜줄 수 있다. 그 다음 점은 ((n+1)+c)이므로 여기에 위 <f1: x+1>에서 나온 수를 x로 하고 <N_pt>를 y로 하는 <f2: x+y>를 생성해준다. 마지막 점은 다시 여기서 1을 빼주면 된다. 즉, <f1: x-1>을 적용시켜준다. 이것을 <quad>의 각 A, B, C, D에 연결시켜주고 <mesh>의 F에 연결해준다.

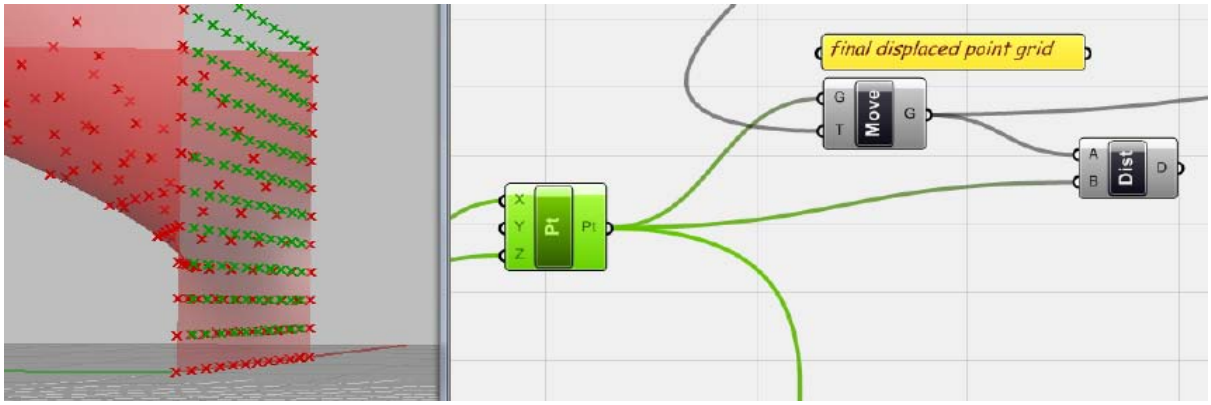


7_4_색을 이용하여 분석결과 시각화하기 (On Colour Analysis)

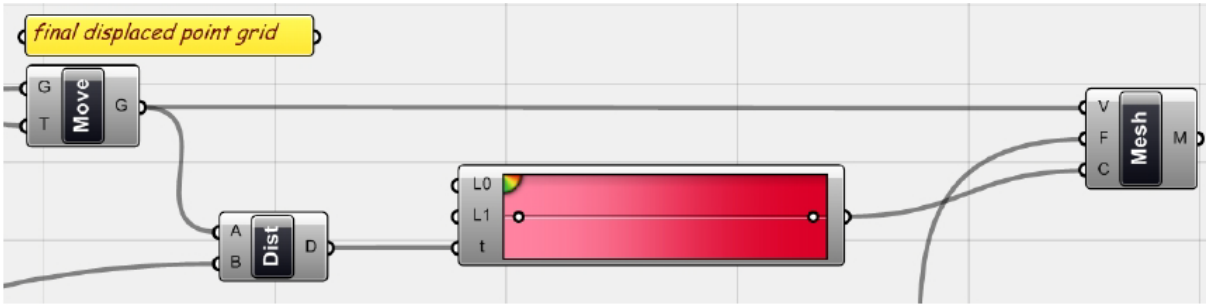
이렇게 생성된 mesh에 색을 이용하여 분석 결과를 시각적으로 표현해보도록 하자. 기본적인 개념은 위치변화가 없는 점과 위치변화가 최대인 점에 각각의 색을 부여하고 그 사이의 점에는 점차적인 색의 변화를 주는 것이다.

먼저 점의 위치 변화 정도를 알기 위해서는 먼저 각 점들의 원래 좌표와 이동 후 좌표를 측정한 뒤 그 둘 사이의 거리를 측정해주면 된다. 이것에 따라 mesh의 face에 색을 지정해주면 된다.

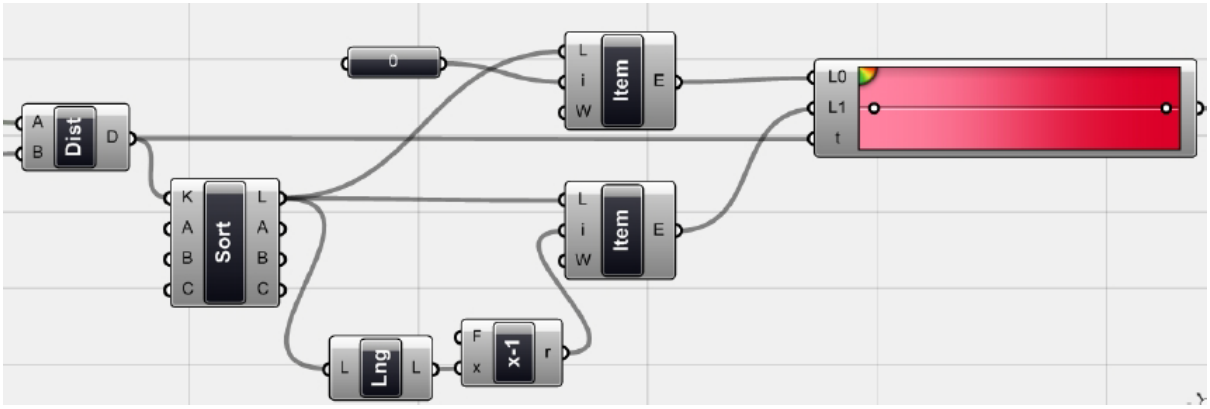
`



가장 처음이 생성한 XZ plane 상의 point grid로 돌아가자. 이것을 <distance>에 연결하고 나머지 하나에는 가장 마지막으로 적용된<move>를 연결해준다.



이제 서서히 변하는 색(Gradient colour)을 적용해보자. 이것에는 <Gradient> (Params > Special > Gradient)를 이용하면 된다. 이것의 t에 parameter를 적용시켜주면 된다. 그것이 바로 <distance> 값이다. L0과 L1에는 t로 들어오는 parameter 중에서 최대값과 최소값을 각각 연결해줘야 한다. 이를 위해서 <distance>가 생성하는 data 중 가장 큰 값과 가장 작은 값을 찾아보도록 하자.

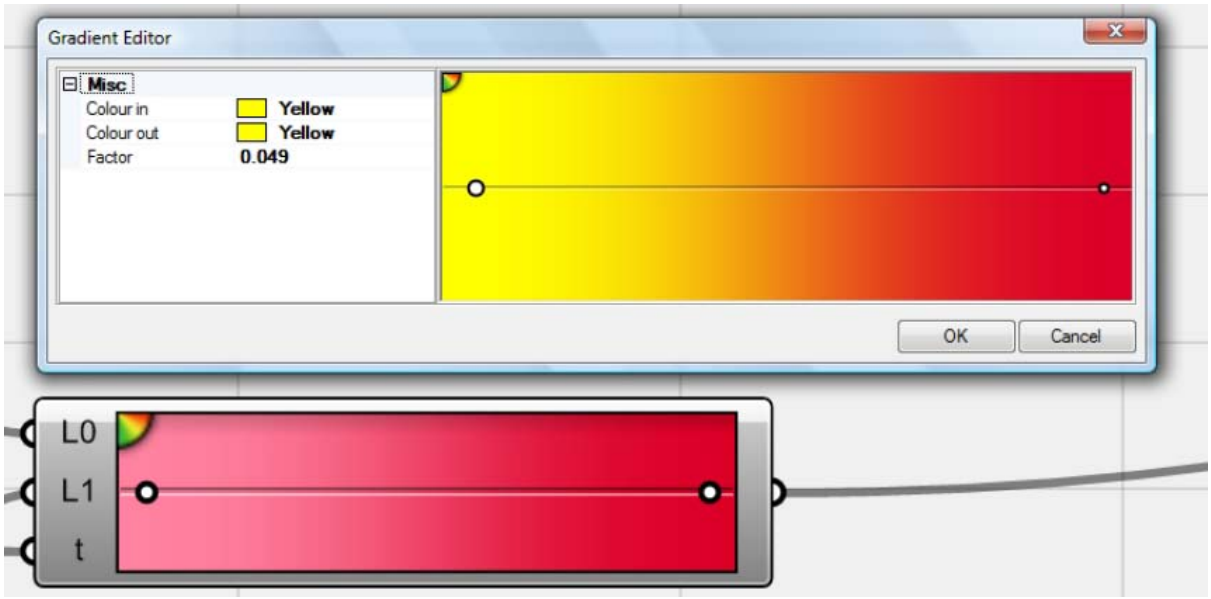


<sort>[50]를 이용하여 <distance>가 생성하는 data list의 값을 오름차순으로 정리해준다. 이것 중 가장 큰 값과 가장 작은 값은 <list item>을 이용하여 추출할 수 있다. 가장 작은 값의 index

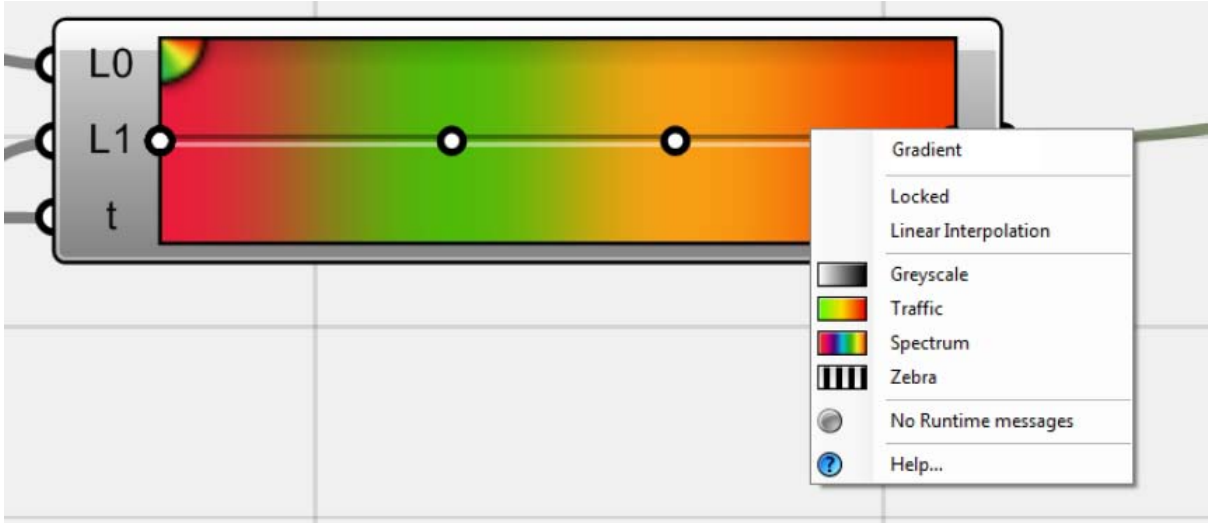
[50] 위 화면에 나온 sort의 경우 pop-up menu의 input manager를 이용하여 input의 개수를 늘려준 것이다. 이 예제에서는 아무런 의미가 없다.

`

number는 0이므로 i에 0을 적용시켜주면 된다. 마지막이 경우 <list length>에서 1을 뺀 값을 i에 적용시켜 주면 된다.[51] 이제 이 두 값을 <gradient>의 L0과 L1에 적용시켜주면 된다.



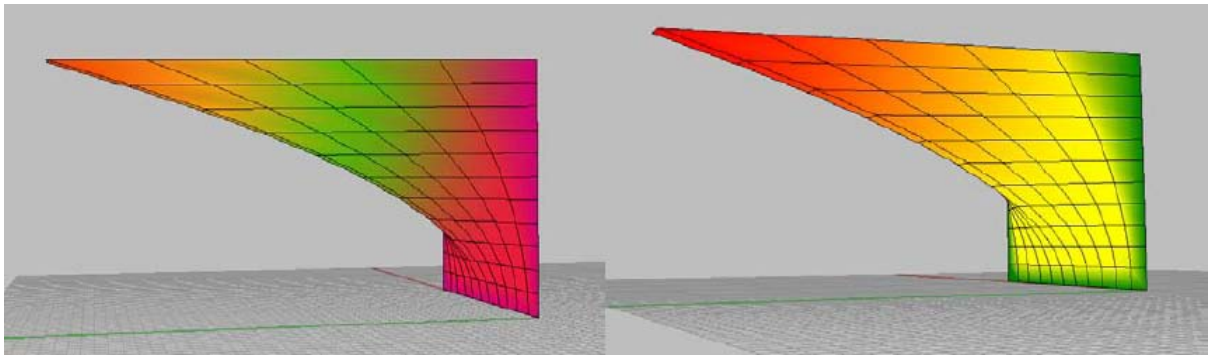
<gradient>의 왼쪽 위에 있는 icon을 클릭하면 원하는 색을 골라줄 수 있다.



<gradient>를 우클릭 하면 pop-up menu가 나타나게 되고 여기서 원하는 색의 적용 방식을 선택할 수 있다. 원하는 분석 목적에 맞는 방식을 선택해주면 된다.

[51] Index number는 0부터 시작하기 때문에 마지막 data의 index number는 data list의 길이에서 1을 뺀 것과 같다.

`

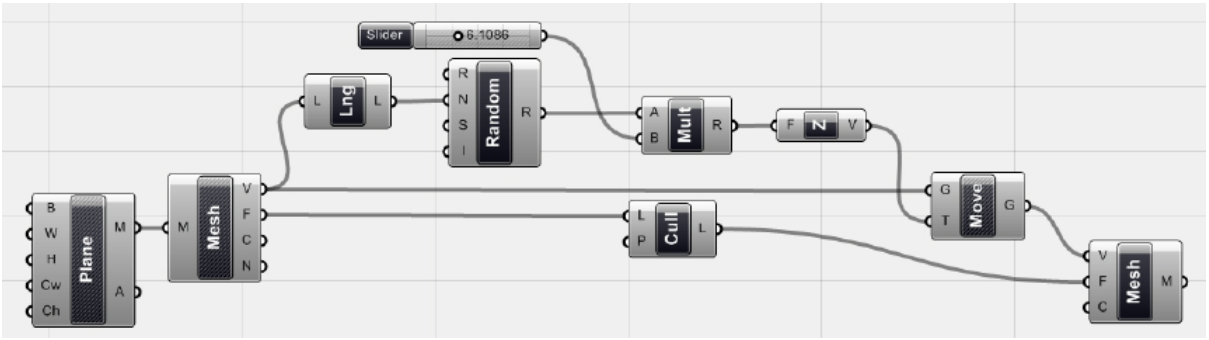


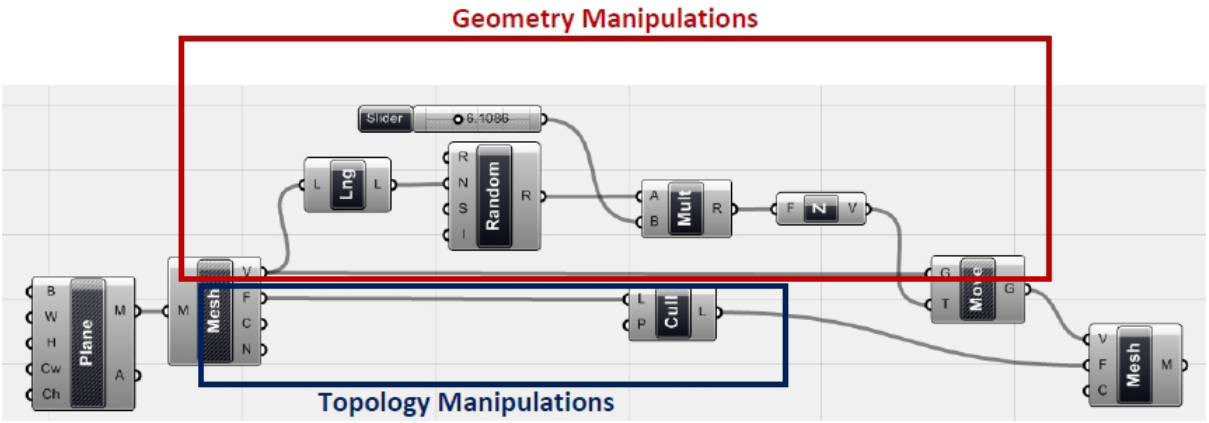
Spectrum과 Traffic을 선택한 경우의 차이

`

7_5_ Mesh 객체를 활용한 Design (Manipulating Mesh objects as a way of Design)

Modeling의 목적에 따라서 그 결과물이 간단해도 상관 없을 경우에는 더욱 쉬운 방법으로 mesh surface를 생성할 수 있다. 즉 위의 예제처럼 처음부터 원하는 형상에 맞춰 Point grid를 생성한 뒤 이것을 vertices로 하는 mesh를 그리는 것이 아니라 미리 정의되어있는 mesh를 조작하는 것이다. 즉 미리 정의된 mesh object를 생성하는 component를 가져온 뒤 이것의 parameter를 추출할 수 있은 다른 component에 연결하고 이를 조작해주는 것이다.

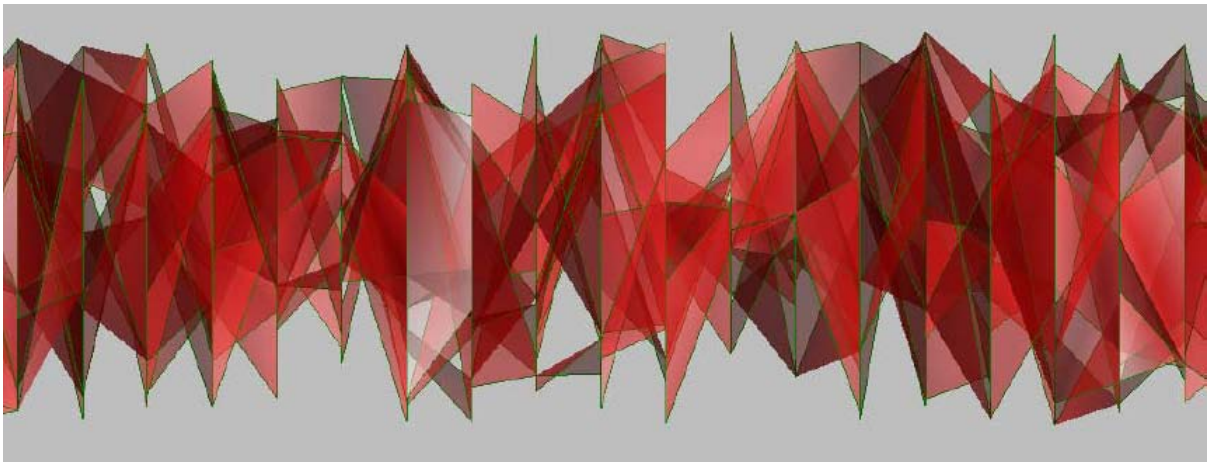




이번 예제에서는 <mesh plane>을 이용하여 mesh surface를 생성한 뒤 이것을 <mesh component>를 이용하여 이 mesh의 vertices와 face를 조작할 수 있게 한다. 먼저 이 vertices들을 Z 방향으로 무작위의 값만큼 이동시키게 된다. 이 때 생성되는 값은 <random>의 R에서 설정된 domain 값인 0 to 1 이므로 여기에 <multiplication>을 연결하여 일괄적으로 <number slider>의 수를 곱해준다. 한편으로 면의 경우 <cull pattern>을 이용하여 사용하고 싶은 face를 솎아내어 준다. 이렇게 골라내어진 face 의 data list와 위에서 얻어진 vertices의 data list를 <Mesh>를 이용하여 matching시켜준다. 이렇게 얻어진 mesh surface는 처음의 mesh plane과 기하학적(geometrical)이면서 위상학적(topological)인 차이를 가지게 된다.

이렇게 기하학적으로 vertices를 조작하고 위상학적으로 면을 변화시키는 방식은 design 목적에 따라 여러 가지 가능성을 가지고 있다. 특히 Mesh의 경우 NURBS와 다르게 surface 내에서 face를 선택적으로 고를 수가 있기 때문에 다공질의 면(porous surface)를 생성하는데 유리하다. 이러한 점을 이해하고 이것을 자신만의 design idea로 승화시켜 보자.

`

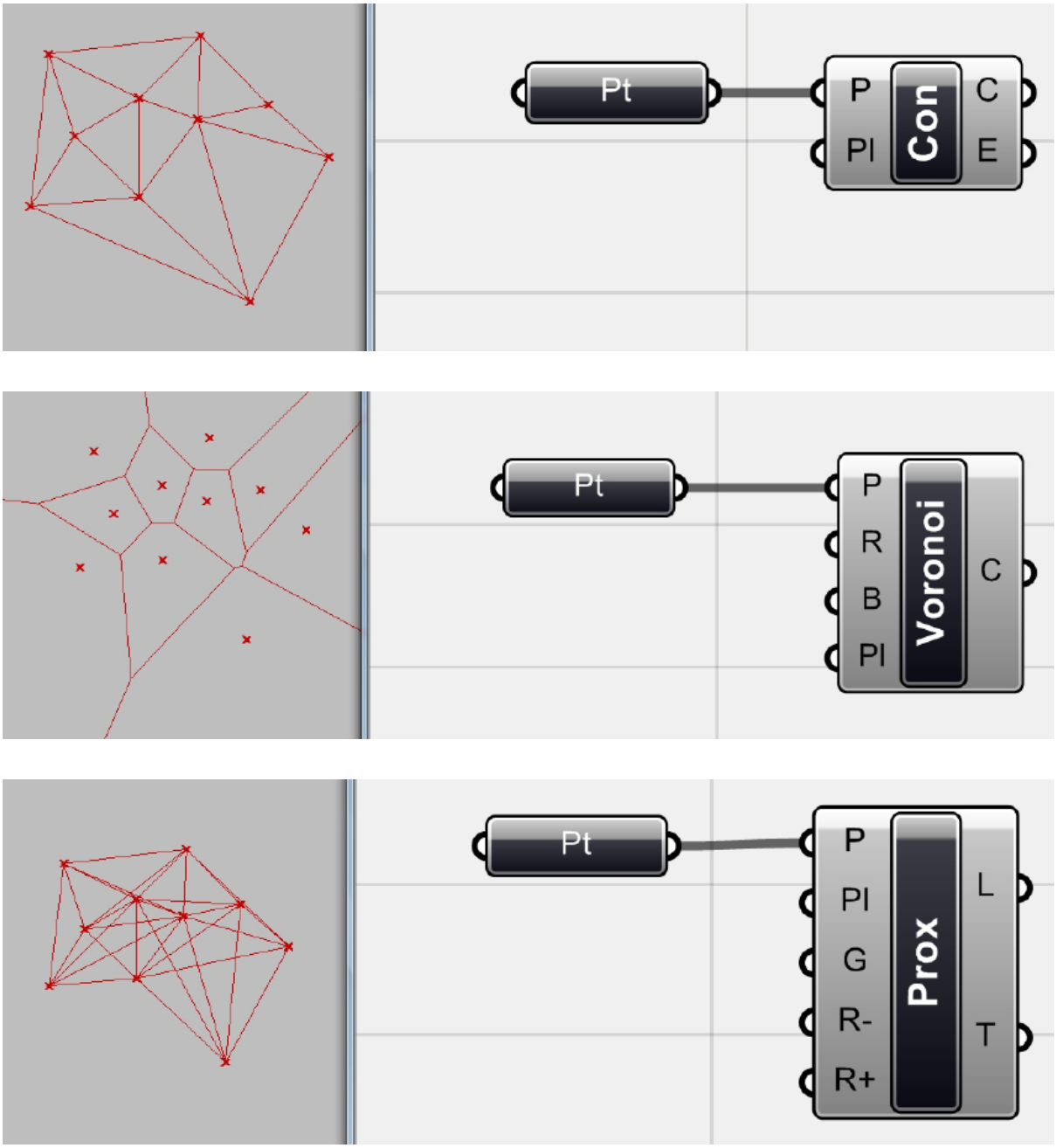


Ramdon하게 조작된 Mesh surface

Design에 적용시키기

`

삼각형 패턴 생성하기 (Triangulation)



삼각형의 pattern을 만드는데 사용되는 component의 종류

Grasshopper의 Mesh tab에는 triangulation에 사용 가능한 다양한 componenent들이 있다. Delaunay 나 Voronoi 혹은 Convex Hull등이 그 예이다. 이 component들이 그려내는 결과물은 복잡한 객체를 design할 때 유용하게 사용될 수 있다. 또한 중요한 점은 이 component들은 점을 input으로 하여 다른 기하체를 output으로 내놓는 다는 것이다. 모두 쉽게 사용할 수 있고 또 이것을 이용한 예제들을 쉽게 찾아볼 수 있을 것이다.

`

`

Chapter_8_Fabrication

`

최근 들어 computer를 이용한 제작(Computer Aided Manufacturing)과 digital fabrication이 큰 관심을 끌고 있다. 이러한 Design process의 변화와 새로운 trend는 design 영역에서 가지고 있는 기존의 생산 개념을 'digital fabrication'으로 바꿔놓았다. 즉 digital modeling을 통하여 design에 관한 사항들을 결정하고 다른 scale에서의 fabrication과 조합을 시험해볼 수 있게 되었다. 이러한 변화 현장에서 전통적으로 사용되었던 건설기술 뿐만 아니라 다양한 생산방식이 적용될 수 있게 하였다. 예를 들어 CNC machine은 요즘에도 사용하고 있는 새로운 생산방식 중 하나이다. 또한 이것을 가능하게 하는 다양한 기술과 생산기계들이 계속해서 발명되고 있다.

이것을 design process에 적용하기 위해서는 먼저 다양한 재료에 따른 fabrication의 과정을 간략하게나마 이해하는 것이 중요하다. 우리가 design 하려는 결과물과 재료에 따라 그것의 조립 방식(assembly logic)이나 이동(transportation), 크기(scale) 등이 달라지기 때문이다. 원하는 결과물을 얻기 위해서는 그것에 적합한 data를 생성하고 생산에 알맞은 기계를 선택하여야 한다. 이때까지의 생산이 평면, 단면, 그리고 디테일 도면에 의해서였다면 이제는 data 와 code를 통해 생산에 대한 정보를 기계에 바로 입력할 수 있기 때문이다.

중요한 점은 designer 도한 이러한 data를 다룰 줄 알아야 한다는 것이다. 이러한 것이 design의 과정과 결과에 끼치는 영향이 더더욱 커지고 있기 때문이다. Designer는 때때로 fabrication의 결과물을 참고하여 data를 재조정 하여야만 한다. 또한 design 결과물이 기계나 조립 과정에서 가질 수 있는 제약으로 인하여 변해야 할 수 있기 때문이다.

지금부터는 grasshopper를 이용하여 design과 fabrication을 통합적으로 다룰 수 있는 방법을 보도록 하겠다. 먼저 데이터 준비(data-preparation) 단계에서 해야 할 일을 살펴보고 design에서 필요한 data를 추출(extraction)하는 과정을 살펴보도록 하겠다. 이 뿐만 아니라 기술적인 내용과 이것에 사용되는 기계, 재료까지 살펴보도록 하자.

8_1_Datasheets

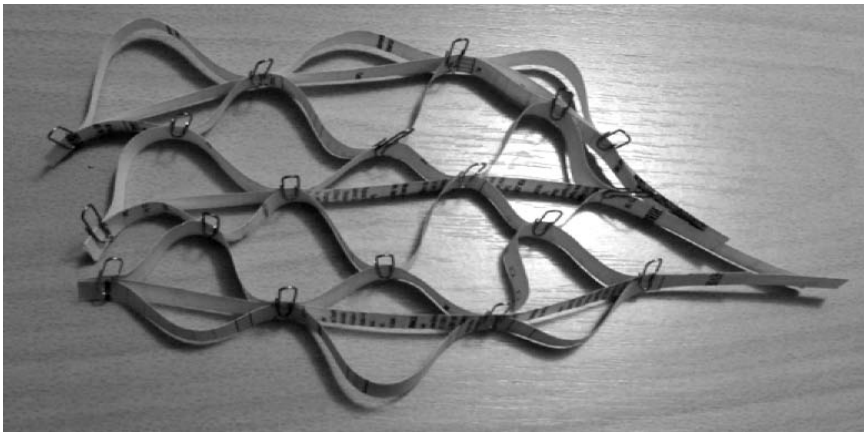
객체를 생성하기 위한 data로는 기본적으로 '치수', '각도(angle)', '좌표(coordinates)' 등과 같은 수치 data가 필요하다. Grasshopper에는 이러한 객체의 치수, 객체간의 거리, 각도 등을 입력하고 추출하기 위한 여러 가지 component들이 있다. 중요한 것은 design 목적에 맞는 정보를 적재적소에 input으로 활용하는 것이다. 이를 위해서는 먼저 결과물이 되는 객체의 기하학적 생성원리를 이해하는 것이 중요하다. 그 다음으로는 제작(fabrication)을 위하여 필요한 data와 그렇지 않은 data를 구분하는 것이다. Algorithm의 과정에서 많은 data가 생성되지만 그 중에는 마지막 fabrication 단계에서 필요 없는 경우가 많다. 마지막으로 필요한 것은 3d software에서 data를 spreadsheet과 datasheet로 추출하여 우리가 원하는 방향으로 수정하는 것이다.

종이띠를 이용한 fabrication (Paper Strip Project)

이번 예시는 필자의 개인 project의 일부이다. 종이조각들이 조립되는 논리를 이해하기 위해서 먼저 실재 종이를 잘라 간단한 모델을 만들어보았다. 이러한 간단한 physical modeling은 3d model이 생성되는 논리를 쉽게 이해할 수 있고 또 향 수 발생할 수 있는 문제를 예측해볼 수 있는 과정이다. 첫 번째 종이모형의 경우 만드는 과정에서 지나치게 복잡해졌고 또 원하는 결과물

`

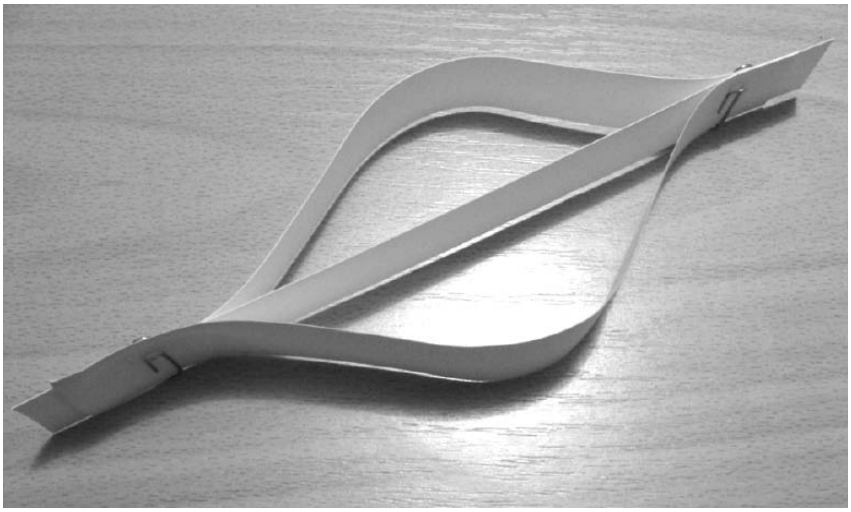
이 나오지 않았다.



종이 띠, 첫 번째 결과물

다음 단계에서는 digital modeling에 참고하기 위하여 좀 더 간단한 모형을 만들어 보았다. 바로 digital modeling을 시작하기 보다는, 간단한 physical model을 만들어 보면서 그릴 대상의 지하학적인 주성 원리를 이해해보는 것이 중요하다.

여기서는 세 개의 종이끈이 사용되었고, 이것의 각 끝을 연결하여 아래와 같은 모양을 만들었다. 즉 가운데 종이가 양쪽보다 더 짧은 것을 볼 수 있다. 이러한 기본 module이 조합되며 더 큰 surface를 만들어낼 것이다.



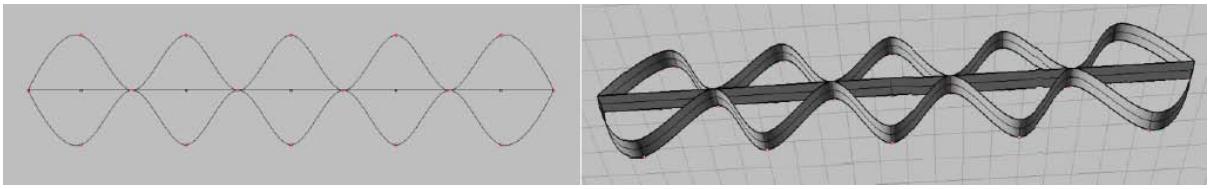
연결과 논리를 이해하기 위한 간단한 종이띠 module

Digital modelling

물리적으로 module이 만들어지는 방식을을 이해한 뒤, module을 digital tool을 이용하여 이 module을 modeling 해보자. 먼저 매우 간단한 curve를 그린 뒤 이것을 중심에 두고서 이를 일정한 간격으로 나눈다. 그 위에 생기는 점들을 한 점은 취하고, 다른 한 점은 버리는 방식으로 점을 솎아내어 준다. 이 때 취하는 점들을 중심이 되는 curve의 양 옆으로 이동시킨다. 이렇게 생성된 point들을 두 interpolated curve를 생성하는 점으로 사용한다. 마지막으로 중앙에 있는 curve와

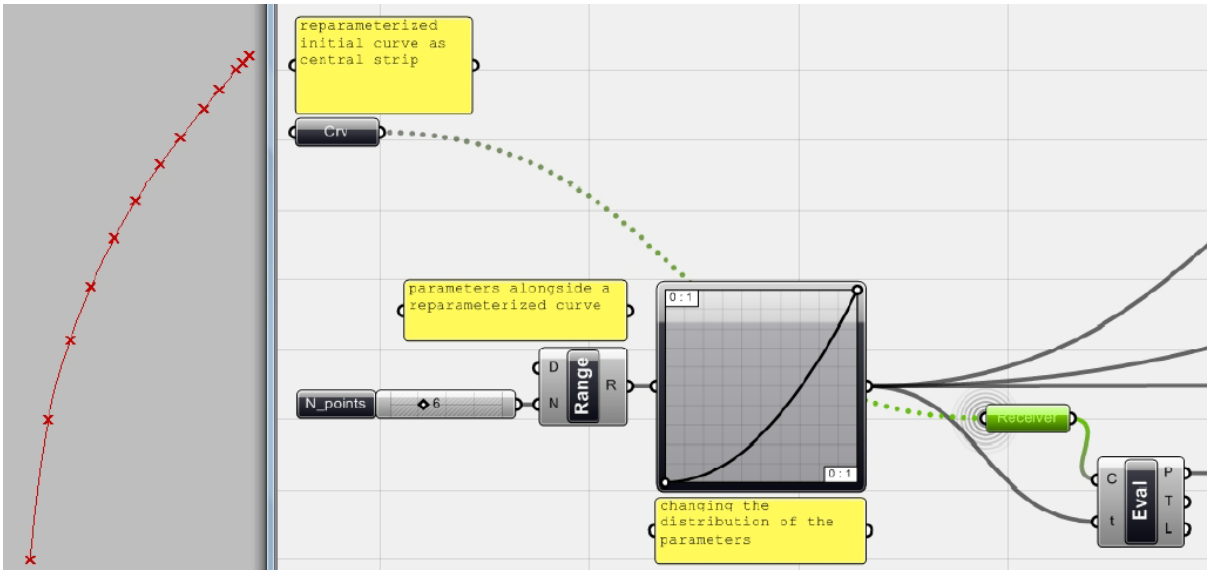
`

양쪽의 두 interpolated curve를 extrude하면 종이로 만든 model과 유사한 형상을 얻을 수 있다.



중심선과 양쪽의 interpolated curve를 이용한 modeling

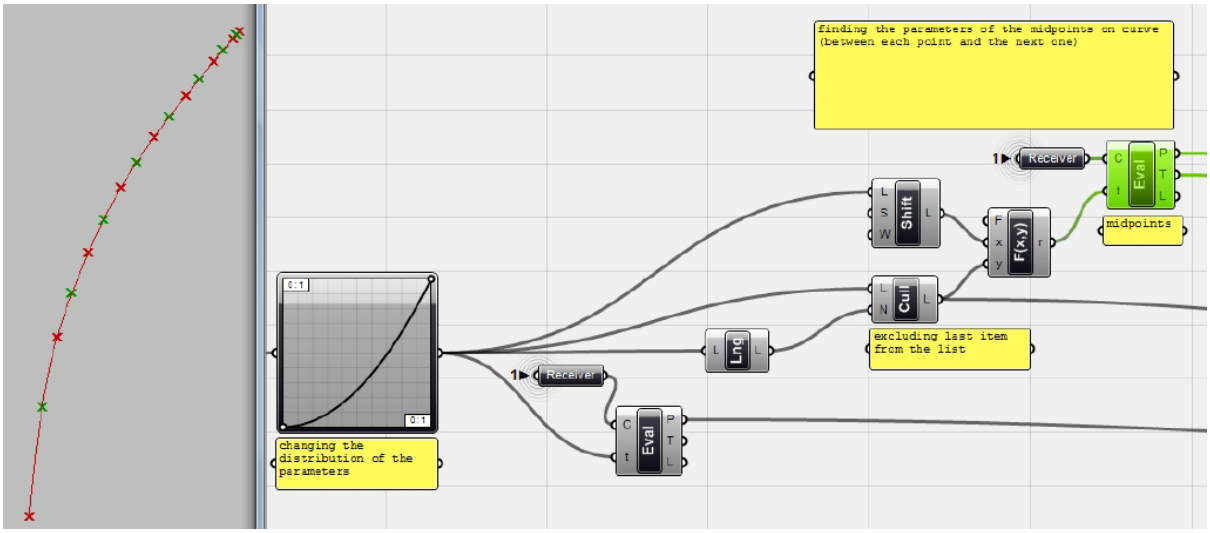
완성된 모델은 간단하지만 같은 크기와 형태가 계속 반복되고 있는 것을 볼 수 있다. 여기에 각 모듈의 크기를 변화시킬 수 있는 가능성을 주도록 하자. 이제 Grasshopper를 이용하여 같은 것을 modeling 해보자. 여기서 기본적인 개념을 설명하여보도록 하겠다.



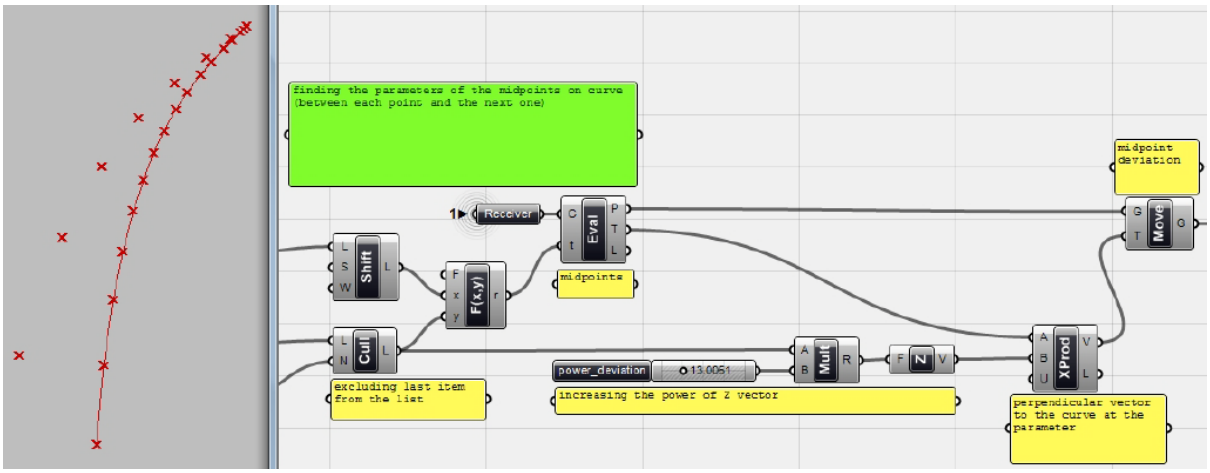
중심이 되는 띠를 Rhino에서 그린 뒤 이것을 <curve>를 이용하여 grasshopper와 연결시킨다. 이 선의 길이를 나눌 때 그 간격을 일정한 것이 아닌 점점 감소하도록 해보자. 먼저 <range>를 이용하여 0과 1사이의 수를 일정한 간격으로 나눠준다. 이렇게 나온 수의 data list를 <graph mapper> (Params > Special > Graph Mapper)에 연결해주도록 하자. 여기서 graph의 종류는 pop-up menu의 'graph type' 항목을 통하여 선택할 수 있다.[52] 이 값을 <evaluate curve>에 연결하면 앞서 그린 중심 curve상에 있는 점들 중에서 <graph mapper>에서 나오는 값에 해당하는 parameter를 가지는 점들을 표시해줄 수 있다. (위 경우 <receiver>가 <curve>를 <evaluate curve>에 연결해주고 있다.

[52] 이 경우 <range>에서 나오는 수는 (0.0, 0.1666, 0.3333, 0.5, 0.6666, 0.8333, 1.0)이다. 또한 graph의 x축, y축의 범위도 0 to 1로 설정되어 있다. 이 경우 위 <range>에서 나온 값들을 X좌표로 가지는 graph 상의 점들이 가지는 y 좌표 값이 바로 graph의 output이 되며 그 수 또한 0과 1사이에 있게 된다. 이 output은 실시간으로 graph의 형태에 따라 변하게 된다.

`



이제 각 점들 사이의 중간 점을 찾아주도록 하자. 원리는 간단하다. <graph mapper>로 부터 얻어진 수들 사이의 평균값을 parameter로 하는 curve상의 점을 찾아주면 된다. 두 수의 평균은 각 수를 더한 뒤 이것을 2로 나누면 되는 것이다. 이것을 위해서는 먼저 <graph mapper>에서 나오는 data list를 더해지는 두 수중 첫 번째 값의 list 그리고 두 번째 값의 list로 나누어 주면 된다. 첫 번째 값의 list의 경우에는 마지막 수가 필요가 없으므로 이것을 버려줘야 한다. 먼저 <list length>를 이용하여 data의 개수를 찾아준다. 이 수는 마지막 번째 data의 수와 일치한다. 이것을 <cull Nth>에 연결하면 이 값에 해당하는 data를 버릴 수 있다. 더해지는 두 번째 값의 list는 원래 <graph mapper>에서 나오는 값 중 가장 앞에 있는 값을 필요로 하지 않으므로 이 수를 버려주어야 한다. 이제 이 두 data list를 <f2: (x+y)/2>로 연결하여 matching 시켜주면 각 중간점에 해당하는 parameter를 얻을 수 있다. 이 수를 다시 <evaluate curve>에 연결하여 curve상에서 중점을 찾아주도록 하자.



이제 이 중간점을 curve 의 바깥으로 움직여보자. 이때 움직이는 방향은 각 점들의 법선 방향을 취해주면 된다. 중간점들의 parameter에 연결된 <evaluate curve>의 output은 점뿐만 아니라 각 점에서의 접선 vector (tangent vector)를 얻을 수 있다.

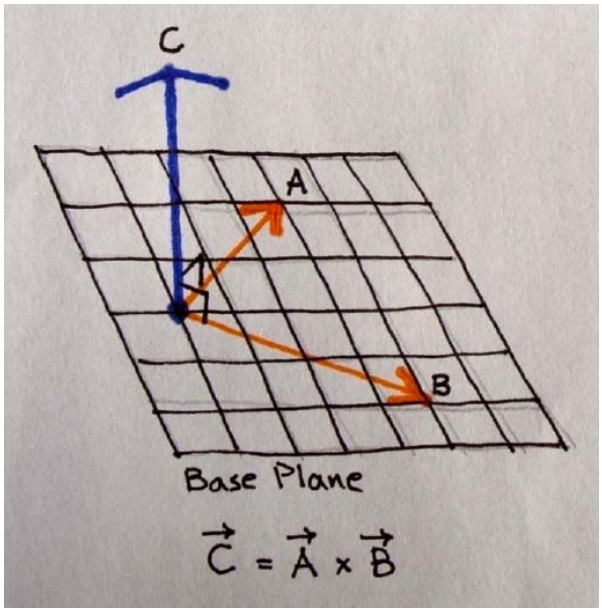
이 때 이용할 수 있는 것이 바로 벡터의 외적(Vector Cross product)[53]이다. 이것은 두 vector가

[53] vector cross product에 관한 내용은 다음 link를 참고하기 바란다.

`

있을 때 이 두 vector 모두에 수직으로 만나는 하나의 vector를 찾아주는 연산이다. 예를 들어서 unit Z vector는 모두 unit X vector와 unit Y vector에 수직으로 만난다. 이것의 개념은 아래 그림을 참조하도록 하자. 이 경우 각 점에서의 Z 방향 vector와 tangent vector의 cross vector product를 찾아주면 이것이 바로 각 점에 수직으로 만나는 vector가 된다. 또한 이것은 중심이 되는 curve가 속해있는 평면 위에 있게 된다. <XProd>가 바로 이러한 연산을 해주는 component이다. 이 때 <cull Nth>에서 나오는 값을 <multiplication>을 이용하여 곱해주면 <unit Z>에 입력하면 이것의 크기를 조절할 수 있다. 이 값이 클수록 <XProd>가 내보내는 vector의 크기도 늘어나게 된다. 이것을 <move>의 T에 연결하여 주자. [54]

이 때 <graph mapper>의 graph type을 바꿔주면 다양한 결과를 얻을 수 있다.



위 그림은 **Vector Cross product**을 설명해주고 있다. 즉 Vector A와 Vector B가 base plane을 정의하고 이 plane에 수직으로 만나는 Vector C를 찾아주는 것이다.

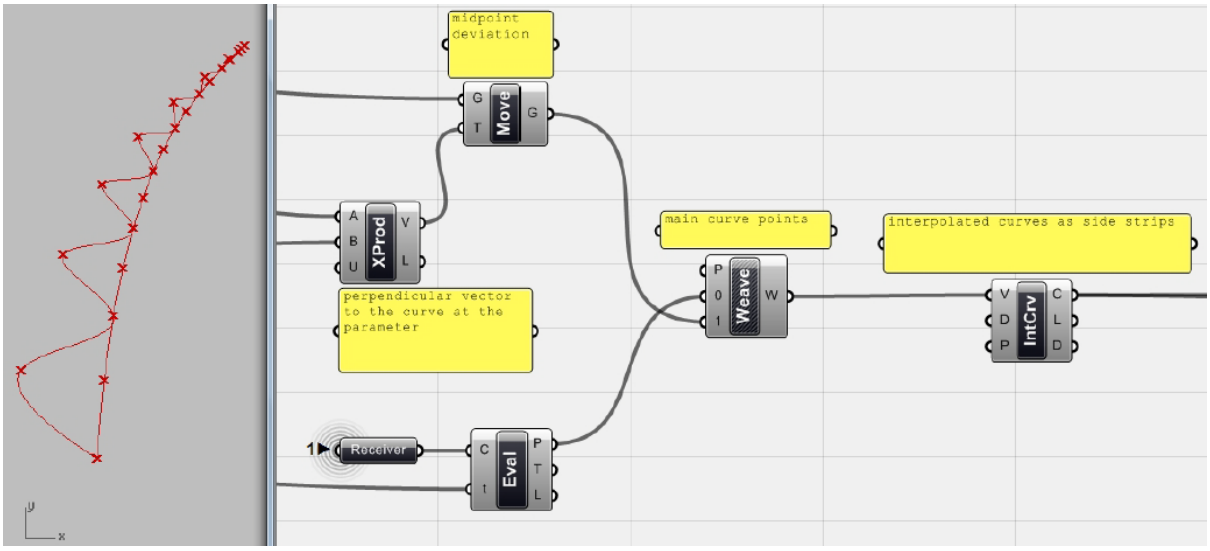
Vector Cross product의 개념은 위와 같다. 즉 Vector A와 Vector B가 base plane을 정의하고 이 plane에 수직으로 만나는 Vector C를 찾아주는 것이다.

---

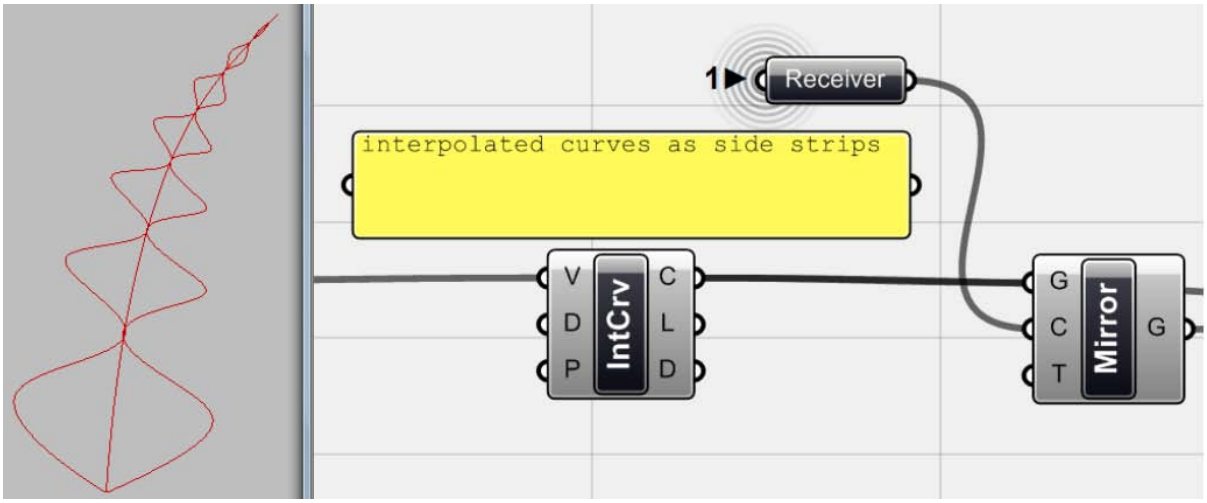
http://geometricmind.wordpress.com/2010/11/18/essential-mathematics-for-computational-design-09/

[54] 혹은 <XProd>에서 나오는 값을 <Amplitude>를 이용하여 크게 해줄 수 있다.

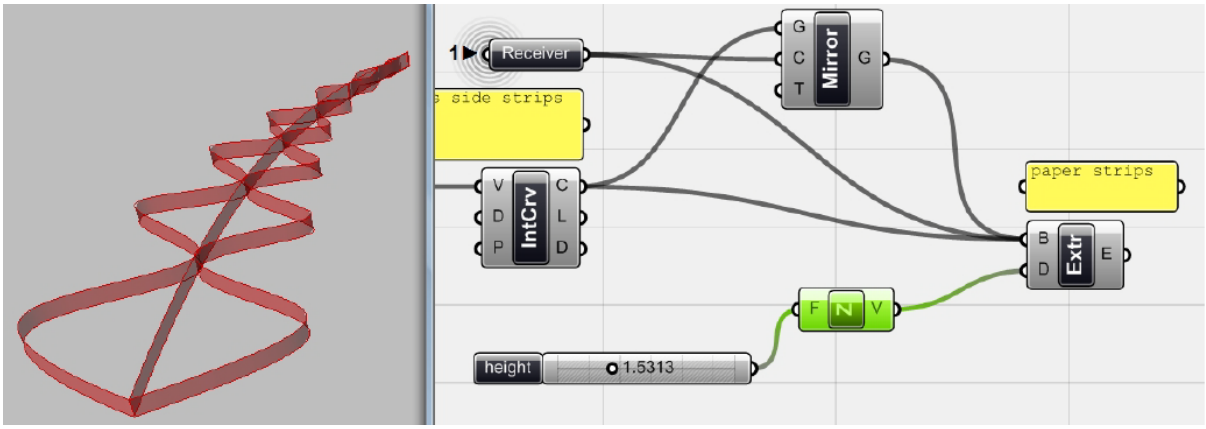
`



이제 중심이 되는 curve 위에서 찾아줬던 point의 data list와 위에서 움직여준 중간 점을 <weaver>를 이용하여 연결하여 준다. (<weave>의 사용 개념은 위에서 살펴보았다.) 이것의 결과물을 이용하여 <interpolate curve>를 만들어 주면 위와 같은 결과를 얻을 수 있다.

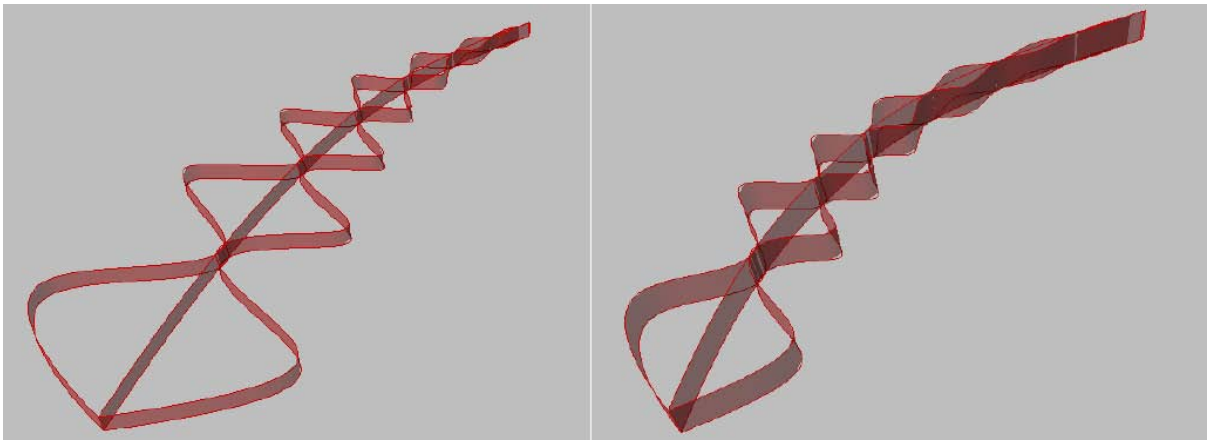


다른 한 쪽은 <Mirror Curve> (XForm > Morph > Mirror Curve) 를 이용하여 그려줄 수 있다. 즉, <interpolate curve>를 중심이 되는 curve를 기준으로 mirror를 해주는 것이다. 위 그림의 <receiver>는 중심이 되는 <curve>와 연결되어 있다. G는 mirror가 되는 기하체(geometry)를 의미한다.



`

이제 중심이 되는 curve와 양 옆에 생긴 curve들을 Z 방향으로 <extrude> 해주면 처음에 의도했던 결과물을 얻을 수 있다.



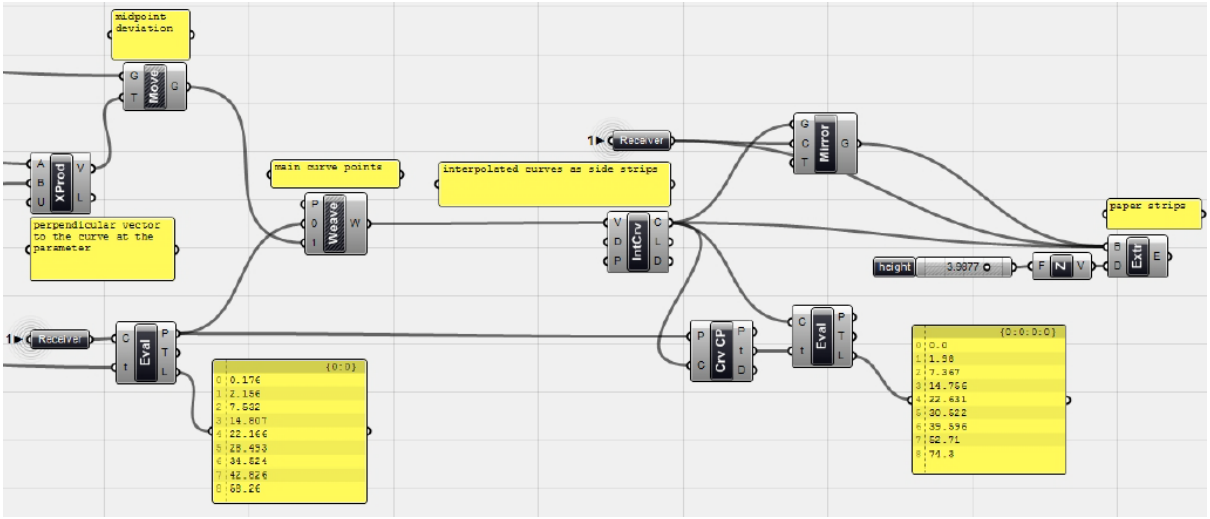
Rhino상에서 처음 그렸던 curve를 변형시켜주면 최종 결과물의 형태를 바꿔줄 수 있다. 혹은 <number slider>를 이용하여 이것이 extrude 되는 정도를 조절해줄 수 있다.

3d modeling을 마친 뒤 여기서 찾아진 정보를 이용하여 종이띠 model을 만들어 보자. 그릴 위해선 먼저 만들어진 model의 치수정보를 추출해야 한다. 물론 laser cutter를 이용하면 아래와 같은 내용 없이도 physical model을 만들 수 있다. 하지만 위에서 일반화된 data를 추출하면 단지 한 종류의 기계나 하나의 방식에 의존하지 않아도 된다.

먼저 중심이 되는 종이띠와 왼쪽과 오른쪽 종이띠 모두 다른 길이를 가지고 있는 것을 볼 수 있다. 그리고 이것들이 각각 만나는 지점이 다르다. 각 길이를 계산하고 이 지점을 각 종이 띠에 표시하여주면 이것을 이용하여 정확한 physical 모델을 만들 수 있다. 이제 grasshopper를 이용하여 이러한 길이정보와 접점의 위치정보를 추출하도록 하자.

이 경우 우리는 curve를 가지고 작업을 하였으므로 그 치수를 찾는데<distance>는 도움이 되지 않는다. 각 접점들이 펼쳐진 curve 위에서 위치하는 지점을 찾아줘야 하기 때문이다.

위에서 작성한 algorithm으로 돌아가 필요한 정보를 찾아보도록 하자.



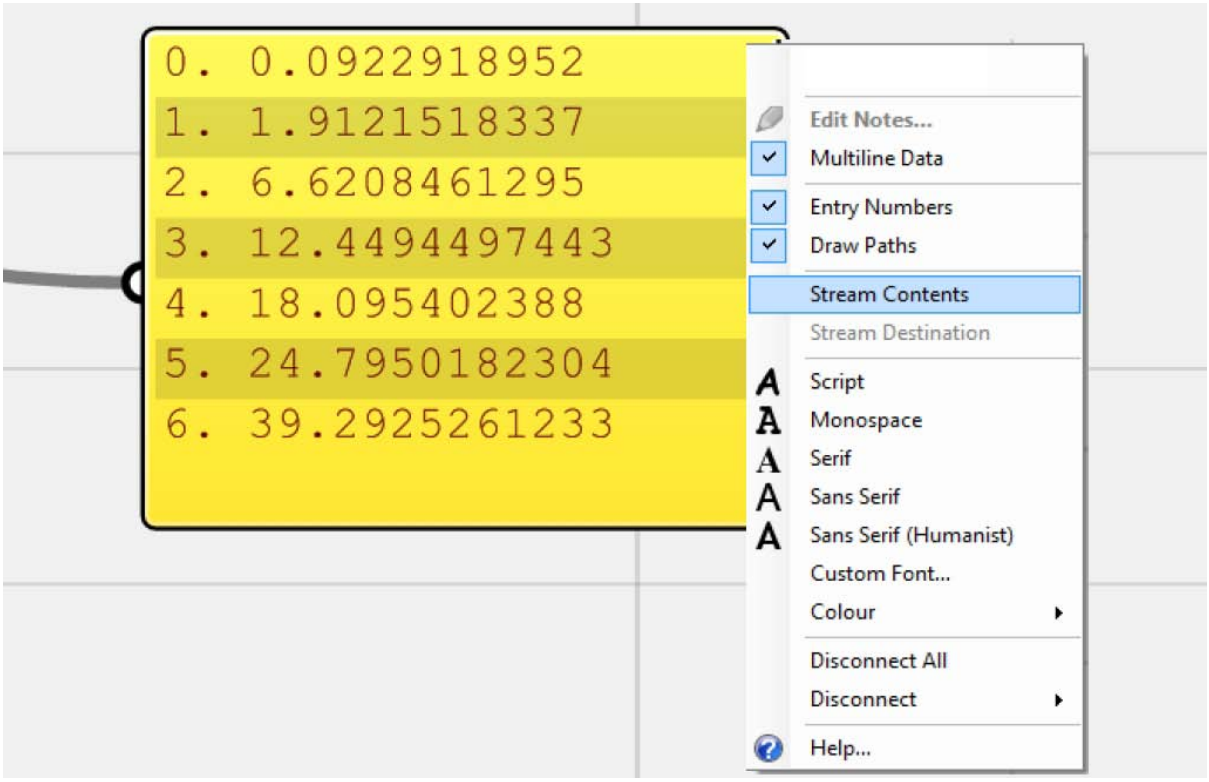
위 그림에서 볼 수 있는 것처럼 <evaluate curve>의 L은 각 점들이 선위에서 시작점으로부터 가지는 길이정보를 보여준다. 위 그림에서 가장 위쪽에 있는 <evaluate curve>의 경우 처음 입력된 중심이 되는 <curve>와 연결되어 있으며 각 점들이 시작점으로부터 떨어진 길이를 보여주고 있

`

다. 오른쪽에 있는 <evaluate curve>의 경우 조금 다르다. 이는 <interpolate curve>의 input으로 사용된 점을 그것의 결과물인 curve와 <curve closest point>를 통해 data matching을 하여 각 점의 길이를 찾아준 것이다. 즉 curve 위에 점을 다시 올려놓은 뒤 이것의 parameter를 추출하고 이것을 다시 <evaluate curve>에 연결하여 각 점들이 가지는 길이 정보를 찾아준다. 이것을 다른 쪽 curve에도 똑같이 적용시켜준다.

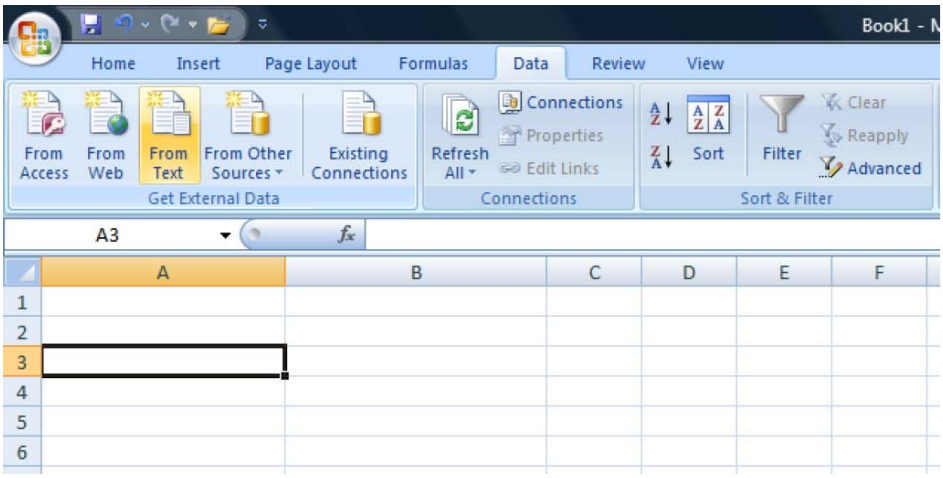
이 때 중심이 되는 curve와 두 interpolate curve의 방향이 같도록 해줘야 한다. 즉 이 세 curve가 가지는 시작점의 위치(길이 계산의 기준점)이 같아야 하는 것이다.

Exporting Data

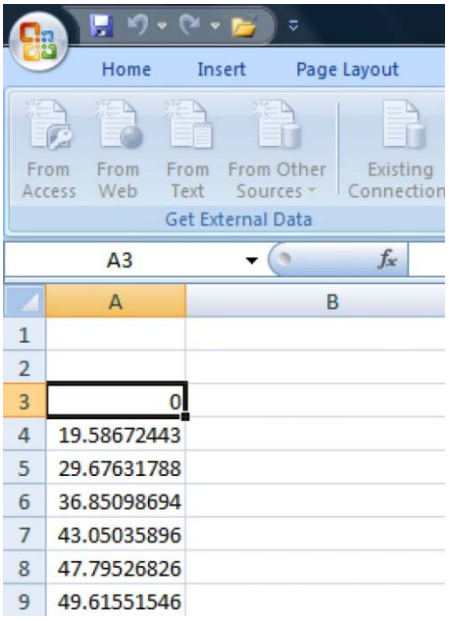


위에서 길이 값을 표시해주는 <panel>을 우클릭 한 뒤, 'stream contents'를 선택해준다. 이것은 panel이 가지고 있는 정보를 다른 format의 수치data로 변환시켜준다. 여기서는 이것을 .txt 로 저장하였다. 이제 이 정보를 excel에서 불러와보자.

`



Excel sheet을 연 뒤 빈 cell을 클릭하고 '**Data**' tab의 '**Get External Data**', '**From text**' 에서 위에서 저장한 text file을 불러와준다.



이제 excel data sheet에 저장된 data를 볼 수 있다. 위와 같은 작업을 나머지 정보에도 똑같이 사용해준다.

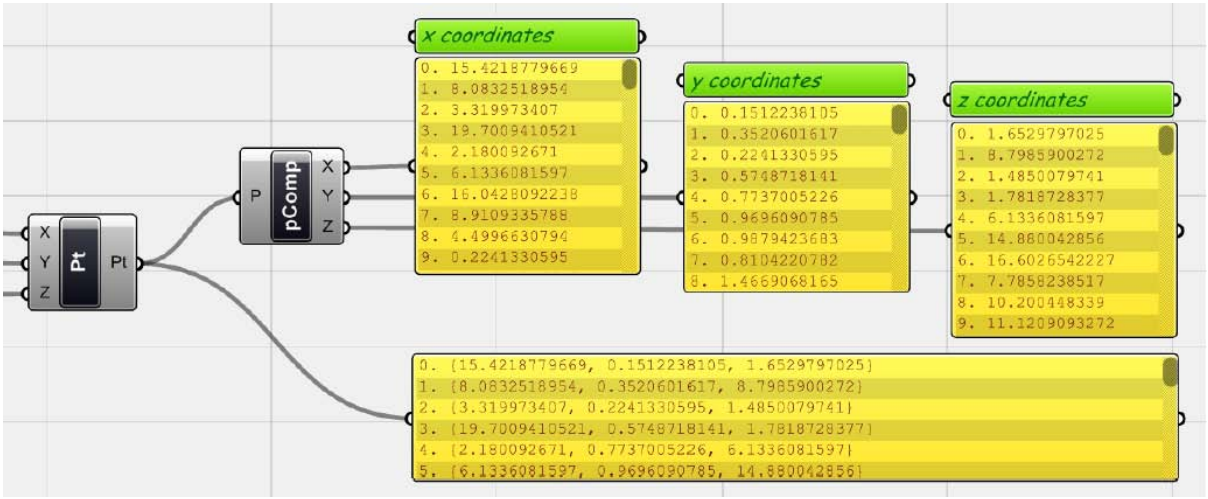
	Connection point distances from start point		
Connection points	Right_strip	Middle_strip	Left_strip
start point	0	0	0
1st connection point	19.58672443	14.49750789	18.71121037
2nd connection point	29.67631788	21.19712374	28.22812292
3rd connection point	36.85098694	26.84307638	34.95597765
4th connection point	43.05035896	32.67167999	41.0266449
5th connection point	47.79526826	37.38037429	45.75826257
end point_(strip length)	49.61551546	39.20023423	47.57834643

각curve위의 점들의 길이값이 정리되었다.

`

만약 여러 점들의 3d 공간좌표가 있다고 했을 때 이것의 좌표값들 또한 excel로 불러내줄 수 있다. 만약 이것의 좌표값을 그대로 불러오면 Comma나 bracket등을 일일이 지워야 한다. 이 때 excel의 옵션을 이용하여 위와 같은 정보를 다른 colomn에 붙여넣은 뒤 이것을 지워도 되지맘ㄴ 이 경우 data의 정렬이 흐트러질 수 있다.

이 때 사용 가능한 방법은 애초에 각각의 x,y,z 좌표값을 따로 불러오는 것이다.



<decompose point>를 이용하면 X,Y,Z값을 따로 data sheet에 정리해줄 수 있다.

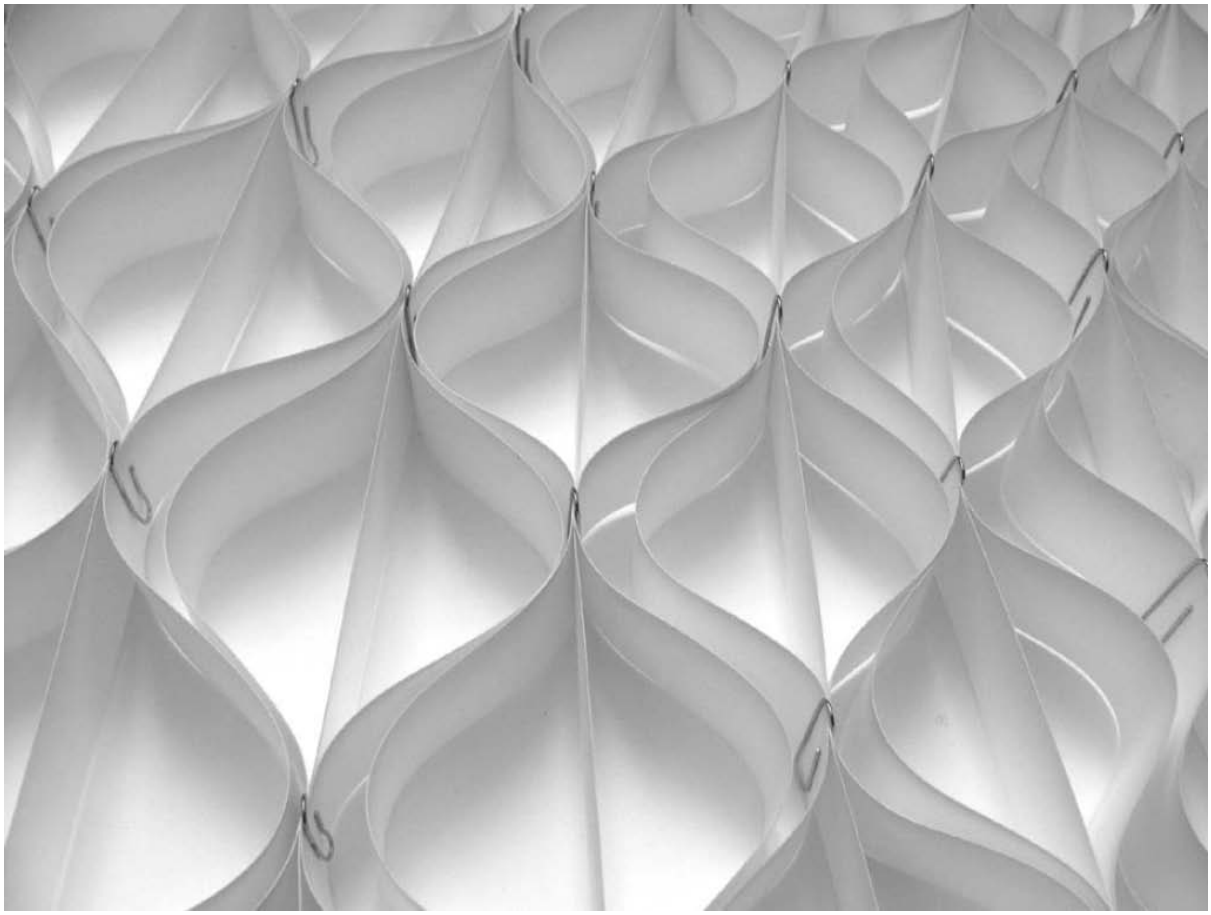
이처럼 Excel의 고급 기능을 잘 활용하면 우리가 가진 data를 manage 하기 훨씬 쉬워진다.

이제 위에서 얻어진 길이정보를 이용하면 아래와 같은 종이 모형을 만들 수 있다. 가끔은 AA의 EmTech 튜터 Achim Menges가 창안한 HAM(Hand Aided Manufacturing)을 활용하는 것이 좋다.[55] 저자의 경우 종이를 자르고 붙이는데 약 한 시간 정도를 사용하였다.

[55] 원 저자가 농담을 한 것인데 제가 번역을 더 잘하지 못하겠습니다. 죄송합니다.

`

`



최종적으로 만들어진 종이 띠 모형

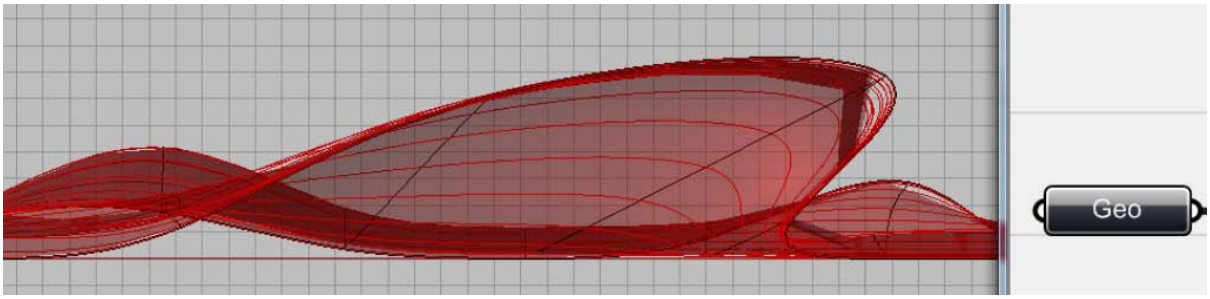
`

8_2_Laser Cutting and Cutting based Manufacturing

복잡한 형태를 가진 3d model을 Laser cutter 가공이 가능하도록 만들어보자. Laser cutter를 이용하면 특히 가전면(developable surface)을 가진 모형이나 여러 면을 가진 접어서 만들어야 하는 모형(folded ones)을 제작할 때 매우 편리하다. Digital model에서 연결된 면들을 하나의 면이 되도록 편 뒤(unfold) 그 형태에 맞춰 모형재료를 자르고 다시 접거나 말아서 붙이는 방식으로 모형을 제작할 수 있기 때문이다. 즉 digital model의 면을 평평한 면의 단위로 분리하여 자른 뒤 이것을 다시 붙여가며 조립해가는 것이다. 복곡면(Double curved surface)의 경우도 위와 같은 방법으로 제작이 가능하다. 방울과 같은(blobby) 형태를 만들 경우에는 먼저 면을 두 방향으로 연속하는 section을 나눈 뒤 이 모양에 따라 재료를 자르고 다시 Bridle joint[56]등을 이용하여 격자형으로 조립할 수 있다.

즉 laser cutter란 판형재료를 가공할 수 있는 도구라는 것이다. 어떠한 형상이라도 연속하는 section과 같이 평평한 면으로 해석해내면 그 모형을 제작할 수 있다. 이때 재료는 종이, 금속, cardboard, 나무 등 무엇이든 가능하다.

이제부터 grasshopper를 이용하여 주어진 곡면을 sectioning을 이용하여 평평한 면으로 해석해보도록 하자.



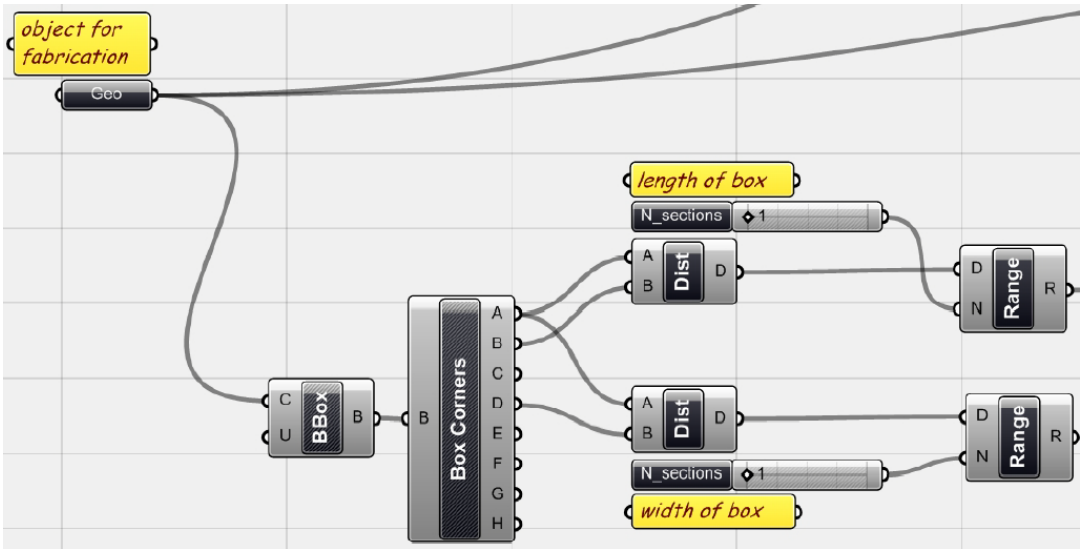
Rhino에서 원하는 형상을 가진 surface를 그리자. 그리고 이 surface를 <geometry>를 이용하여 Grasshopper와 연동시킨다.

단면으로 해석하기(Ribs as Sections)

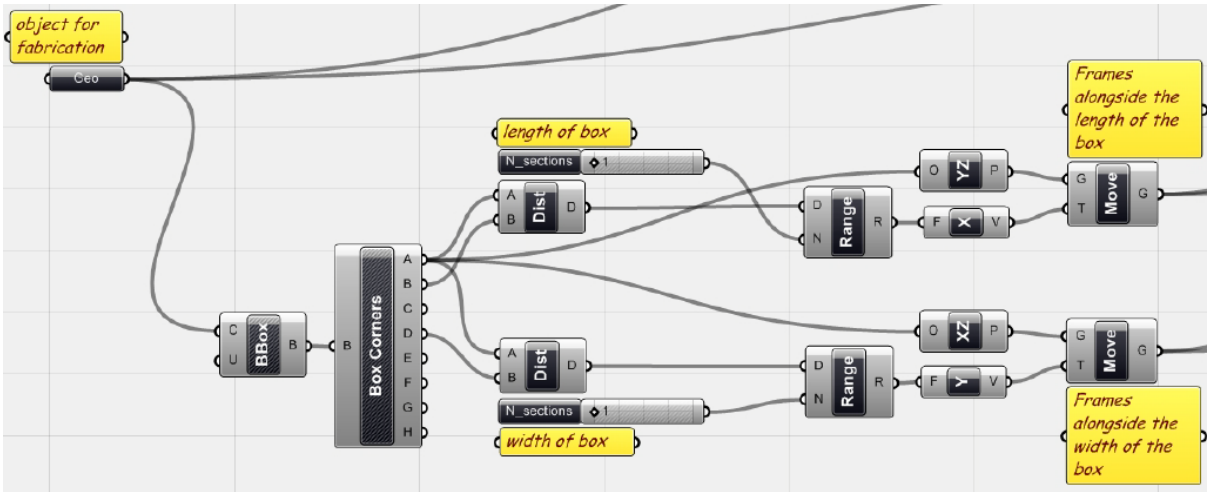
이 free-form surface의 모형을 제작하기 위해서는 먼저 면의 단면을 추출한 뒤 이것을 모형재료의 크기와 맞는 평평한 surface 위에 다시 그리도록 하자. 이때 Rhino에서 그려진 surface에는 두께가 없으므로 원하는 두께는 따로 설정해줘야 한다.

[56] http://en.wikipedia.org/wiki/Bridle_joint

`



Sectioning을 하는데 사용되는 원리는 바로 면과 연속하는 평면을 교차시켰을 때 생기는 'curve를 사용하는 것이다. 먼저 <bounding box>를 이용하여 우리가 가공하고자 하는 3d model을 감싸는 최소한의 box를 구한다. 여기에 <box corners>(Surface > Analysis > Box corners)를 이용하여 이box의 각 꼭지점들을 추출하고 이중 두 끝을 잡아 각각 XZ plane과 YZ plane을 생성한다. 이제 이것을 가로와 세로 방향으로 이동시켜야 한다. 이를 위해 사용 되는 것이 <distance>와 <range> 이다. <distance>는 위에서 생성된 두 plane이 각각 y방향과 x 방향으로 이동하는데 사용되는 거리의 범위가 된다. 이것을 <range>를 이용하여 N만큼 나누어서 생기는 data list의 각 위치에 plane이 이동하게 된다.[57]

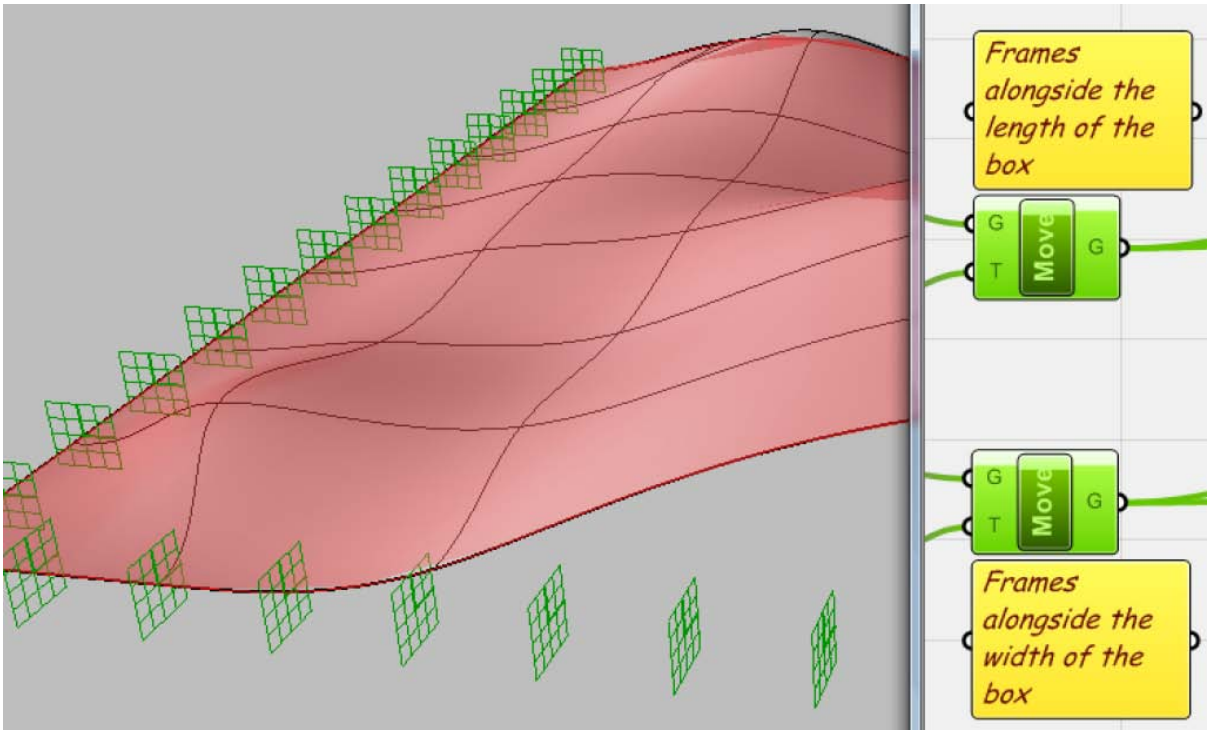


즉 주어진 면을 감싸고 있는 bounding box의 x와 y방향의 모서리의 길이와 꼭지점 하나를 추출한다. 이 꼭지점에 XZ plane을 생성하여 Y방향으로 연속 이동 시키고 YZ plane을 생성하여 X 방향으로 연속해서 이동시킨다. 이제 이 plane 들이 처음 연동시킨 surface와 교차되는 부분을 추출하면 우리가 원하는 section을 얻을 수 있다. 이 때 section의 개수는 <number slider>이용하여 N

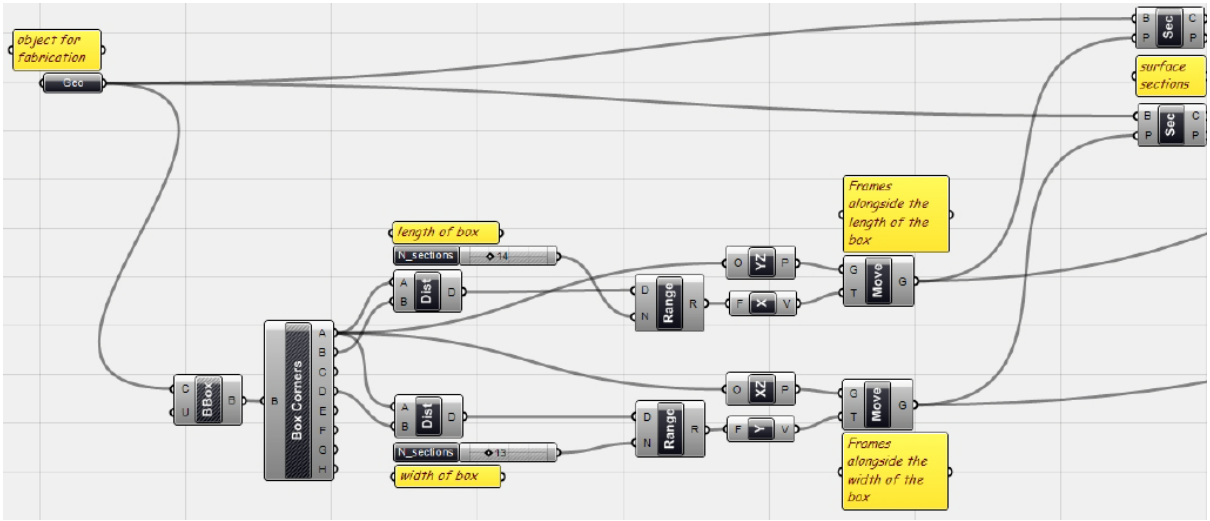
[57] 예를 들어 X방향으로의 길이가 50이라고 했을 때 이 길이를 <range>의 D에 넣고 N에 4를 넣으면 (0.0, 12.5, 25.0, 37.5, 50.0) 이라는 data list가 생긴다. 이는 현재 0.0에 위치하는 YZ plane이 12.5, 25.0, 37.5, 50.0 에도 생기게 되는 것을 의미한다.

`

값을 조절해주면 우리가 원하는 만큼 조절해줄 수 있다.

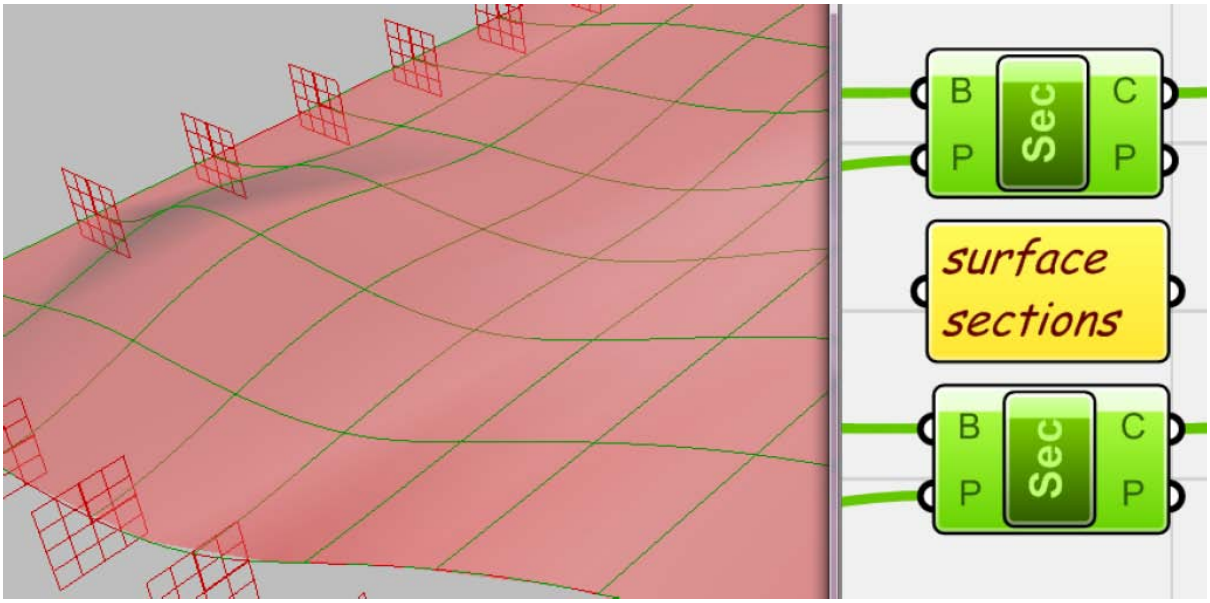


Plane의 원점(Origin)과 각 XY 방향은 위와 같이 frame으로 표시되게 된다. 위 그림에서 볼 수 있는 것처럼 각 frame들은 모서리에 수직방향으로 생기게 된다.



이제 이 plane들과 surface가 교차(intersect)하는 부분을 찾아보도록 하자. 이 경우 surface(Brep)과 plane이 교차하는 것이므로 intersect component 중 <Brep | Plane intersect>(Intersect > Mathematical > BRep | Plane) 를 사용하면 된다. 즉 두 개의 <Brep | Plane Intersect>를 만든 뒤 이 둘의 Brep에 처음 surface를 연동시킨 <geometry>를 연결시켜 준다. 또한 이 전 단계에서 얻어진 X방향으로 연속된 plane과 Y방향으로 연속된 plane들을 P에 연결시켜주면 된다.

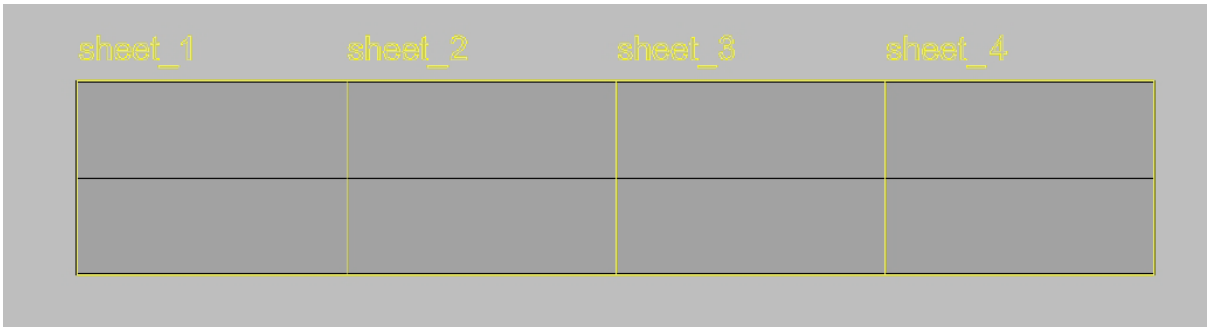
`



이것의 결과물은 위에서 보는 것과 같이 각각 X, Y 방향으로 연속되는 curve이다.

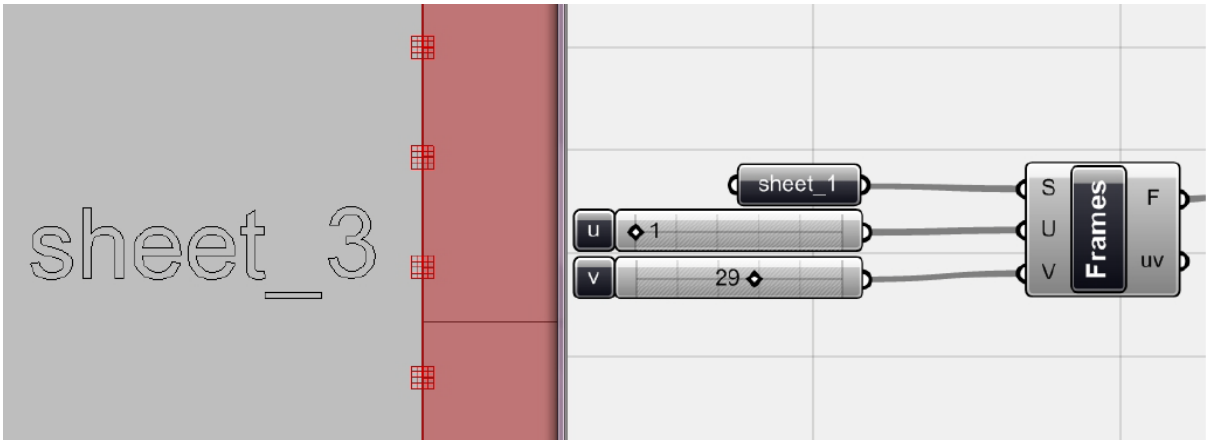
Section 배열하기 (Nesting)

다음 단계는 바로 위에서 얻어진 curve들을 평평한 평면 위에 배열하는 것이다. 이것은 이 후 laser cutter를 이용하여 재료를 자르는데 사용될 것이다. 아래 그림과 같이 재료와 같은 크기를 가지는 직사각형의 Surface 을 그린다. 그리고 이 surface를 아래와 같이 연속하여 배열시킨다.

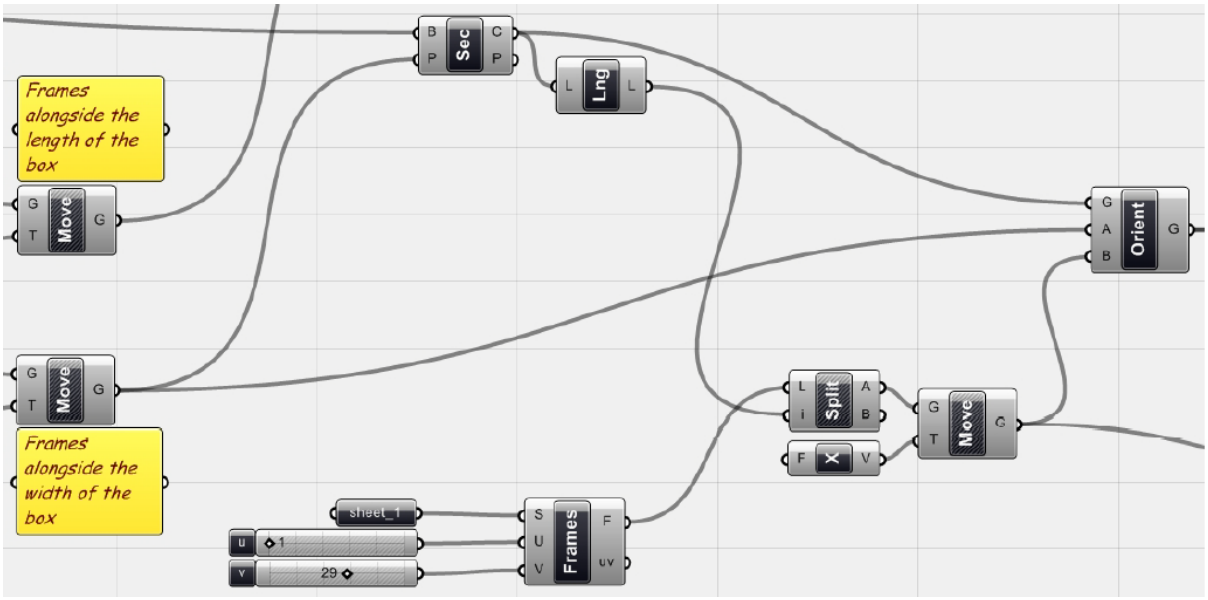


이제 <Orient>(XForm > Euclidian > Orient)를 이용하여 위에서 얻은 curve를 위 surface 위에 그려보자. <Orient>는 먼저 방향을 바꿀 G: geometry와 이동의 기준으로 사용 될 A: reference plane, 그리고 이 geometry가 이동되어 그려질 대상이 되는 B: target plane으로 이루어진다. 먼저 <Orient>에 G는 위에서 <Brep | Plane Intersect>에서 나오는 Curve가 될 것이고 A: reference plane은 위에서 section curve를 얻기 위해서 사용했던 plane들을 사용할 수 있다. 이제 target plane을 만들어보자.

`



먼저 위에서 그린 직사각형의 surface를 <surface>를 이용하여 Grasshopper와 연결시킨다. 이것에 <Surface Frame>(Surface > Util > Surface Frames)을 연결하도록 하자. 이것은 면 위에 UV 방향으로 우리가 원하는 수 만큼의 frame (plane의 원점과 방향이 표시된)을 생성할 수 있다.

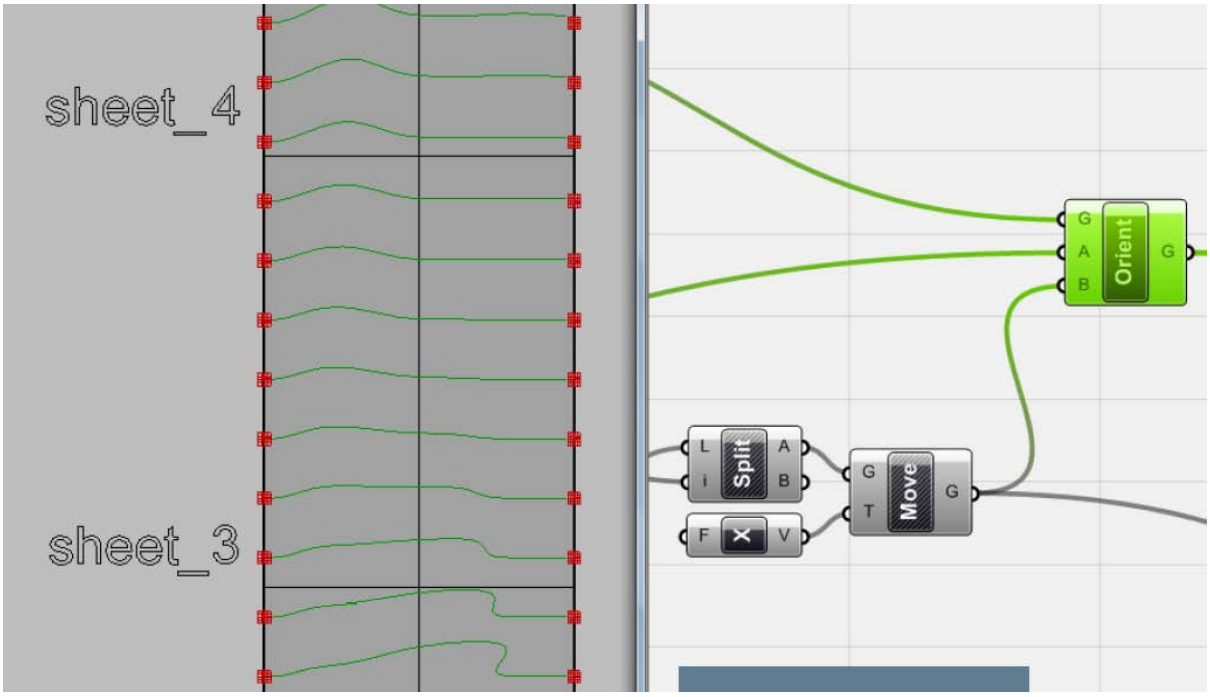


<surface frame>에 생기는 frame의 개수는 U,V에 입력되는 수를 통하여 조절할 수 있다. 이 때 주의해야 할 것은 U에 1을 입력하더라도 이 방향으로 생성되는 frame은 총 2열이라는 것을 알 수 있다. Surface를 U 방향으로 1로 나누는 것이기 때문에 시작인 0과 끝이 1에 각각 frame이 생성되기 때문이다.[58] 즉 우리가 필요한 것 보다 더 많은 수의 frame이 생기게 된다. 이를 위해서 우리가 필요한 수만큼의 plane을 골라내어야 한다. 이 때 사용 가능한 것이 <split>이다. 이것은 L을 통하여 data list가 들어올 때 i: integer에 입력되는 수 만큼의 data 를 A로, 그 나머지를 B로 내보내주는 역할을 한다. 먼저 <Brep | Plane Intersection>에서 나오는 C의 개수를 <list length>를 이용하여 찾아준다. 이것을 split의 i에 연결시키면 우리가 원하는 수 만큼의 plane을 취할 수 있다. 또한 section이 surface의 모서리와 닿는 것을 방지하기 위하여 X방향으로 1만큼 이동시켜

[58] 이 때 주의할 점은 0.8.0004build에서는 <surface frame>에서 나오는 frame이 tree data의 형태로 나온다는 것이다. 즉 <surface frame>과 <split>사이에 <flatten tree>를 사용하면 위와 같은 결과를 얻을 수 있다.

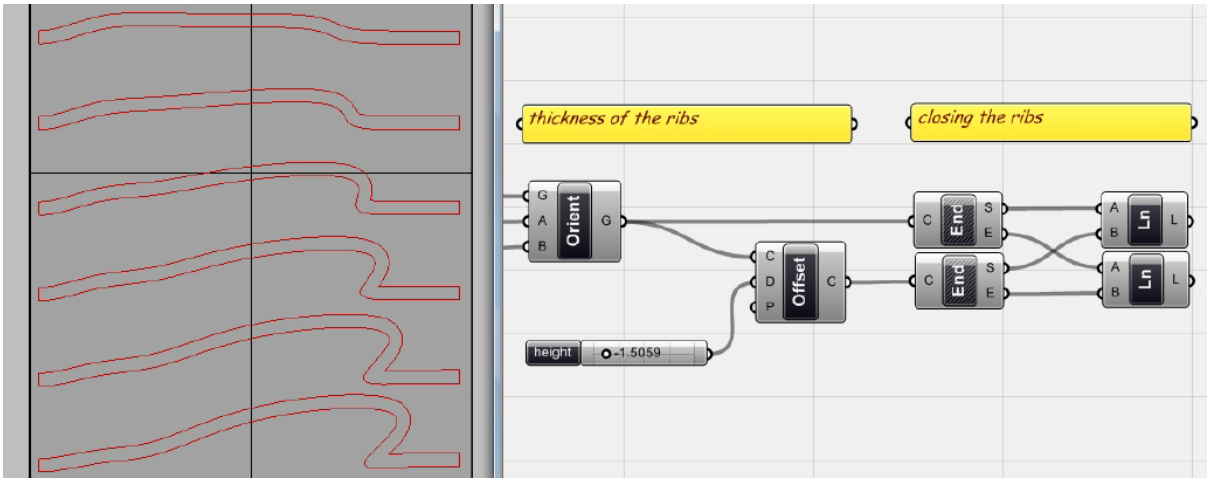
`

주었다.



위의 algorithm의 결과물로 위처럼 curve들이 cutting sheet에 그려졌다.

두께 주기(Generating Ribs)

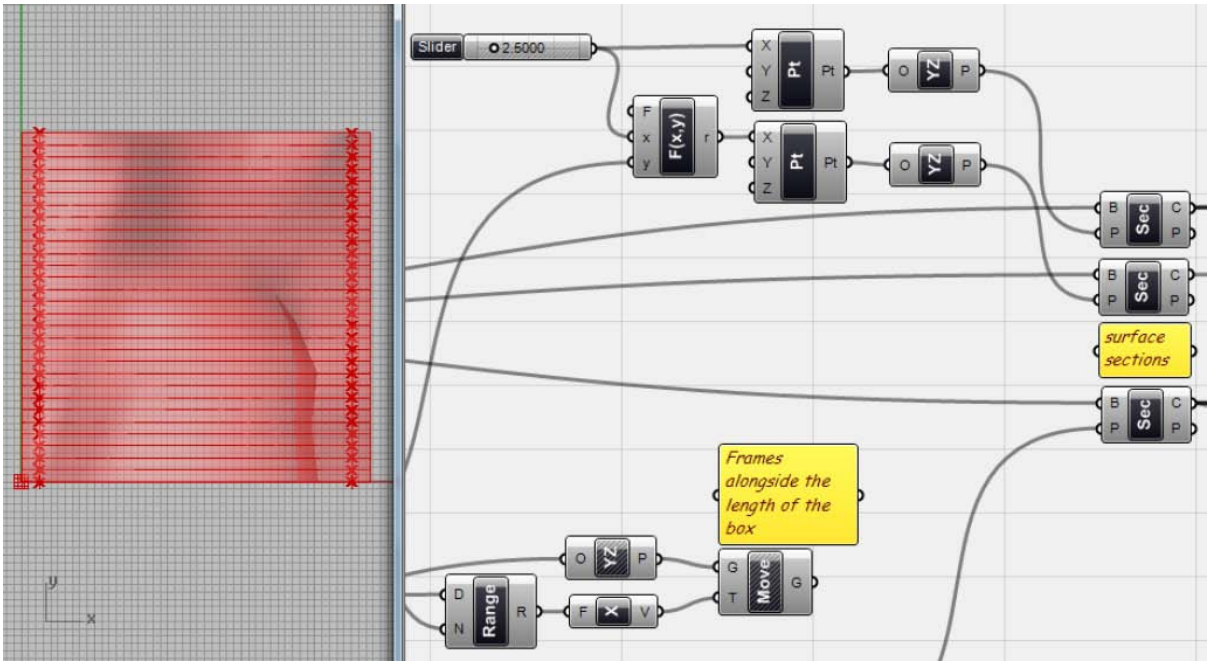


이렇게 재배열 된 section curve들에 뭔하는 만큼의 두께를 주도록 하자. 먼저 <offset>을 이용하여 curve를 원하는 만큼 <offset> 시킨다. 원래 curve와 <offset> 된 curve의 각 양끝을 찾은 뒤 이것에 <line>을 더하여 이어준다. 이러면 원하는 두께를 가진 구조체가 완성된다. 위와 같은 과정을 다른 방향으로 생성한 section curve에도 적용시켜 준다.

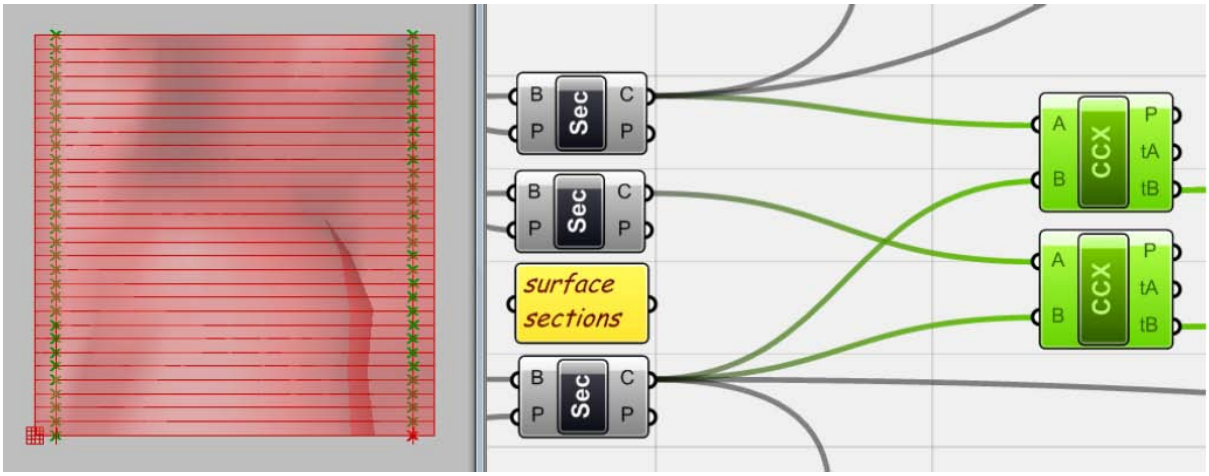
Generating Joints (Bridle Joints)

다음 단계로는 이 구조체들이 서로 얽힐 수 있도록 교차되는 부분을 반씩 파주는 것이다.

`



여기서 사용된 다른 방향의 plane은 위에서 그린 plane이 아닌 저자가 새롭게 따로 그린 plane이다. 이 plane이 input surface와 만나며 curve를 생성하게 된다.[59] <xyz point>는 <YZ plane>의 원점이 되는데 이때의 x 좌표는 <number slider>를 통해 input 해준다. 다른 <xyz point>는 <f2: y-x>의 결과값을 x 좌표로 가지게 되는데 이 <f2>에 input에 되는 x는 <number slider>에서 나오는 값이고 y는 <bounding box>의 x방향 길이인 <distance>값이다. 이렇게 하면 두 <YZ plane>이 <number slider>에 따라 대칭(mirror)된 형태를 가지게 된다. 이 두 <YZ plane>을 <Brep | Plane Intersection>에 연결하여 section curve를 얻도록 하자.

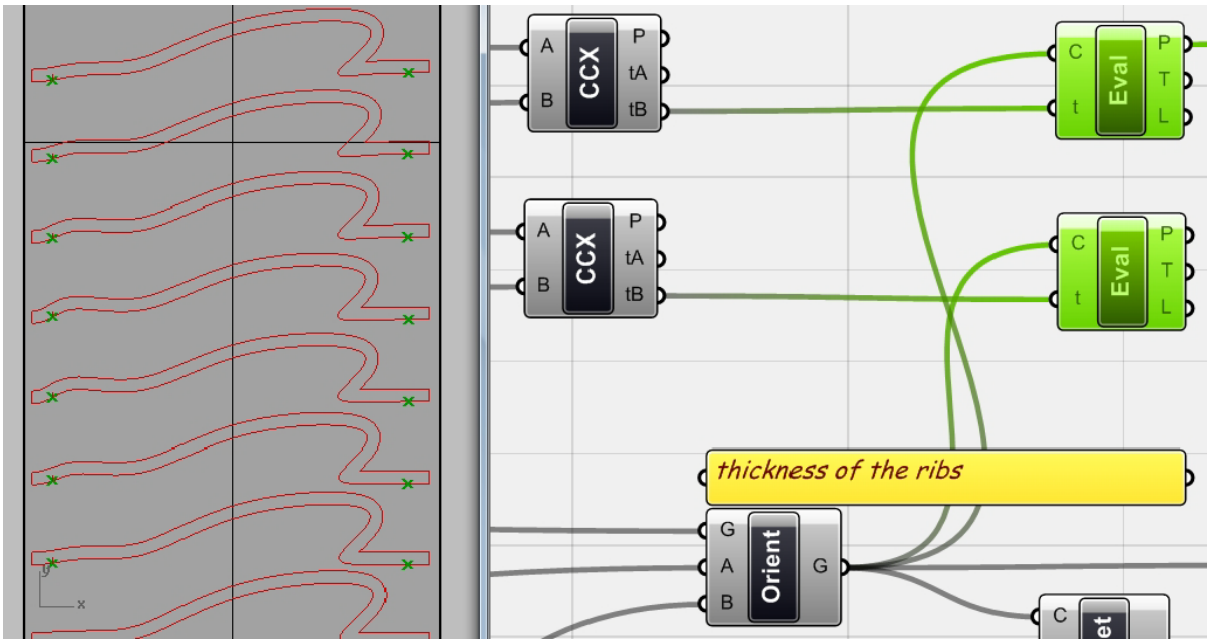


이 section curve와 위에서 그린 curve가 교차하는 점을 찾기 위해서는 <CCX>(Intersect > Physical > Curve | Curve)를 사용하면 된다. 이때 data matching의 방식은 'cross reference'로 해주

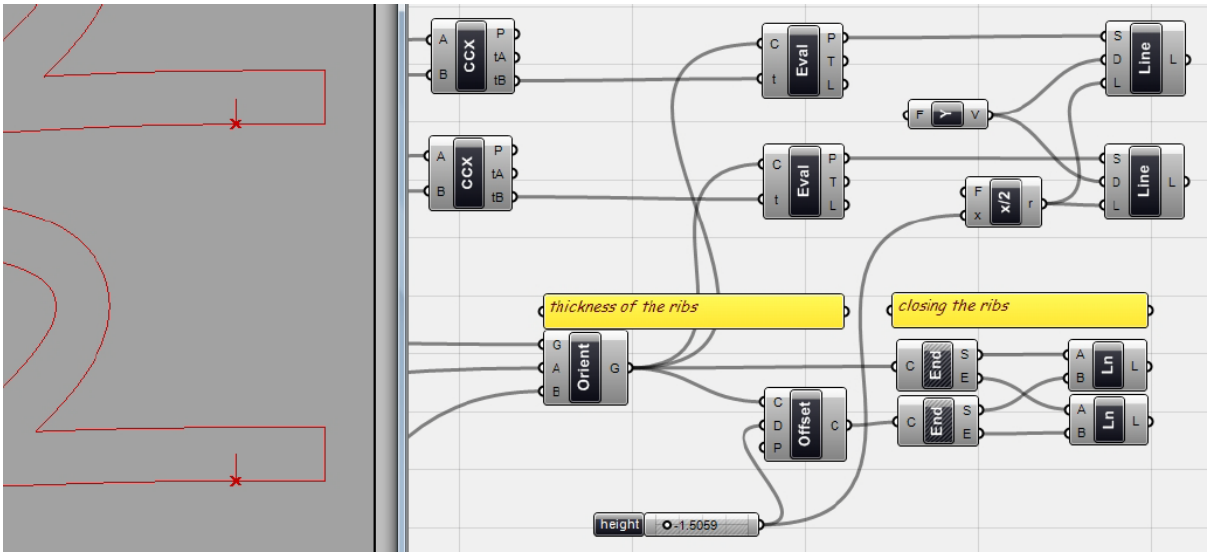
[59] 이 전에 저자가 언급한 바는 없지만 위의 그림에서 보시다시피 top view에서 본surface의 왼쪽 아래 코너의 x, y좌표는 0이다. 만약 surface를 다른 곳에 그렸을 경우 이를 위의 그림에 맞게 옮기면 된다.

`

면 된다. 이제 이 교차점을 이용하여 재배열한 curve들 위에 bridle joint를 그려보자

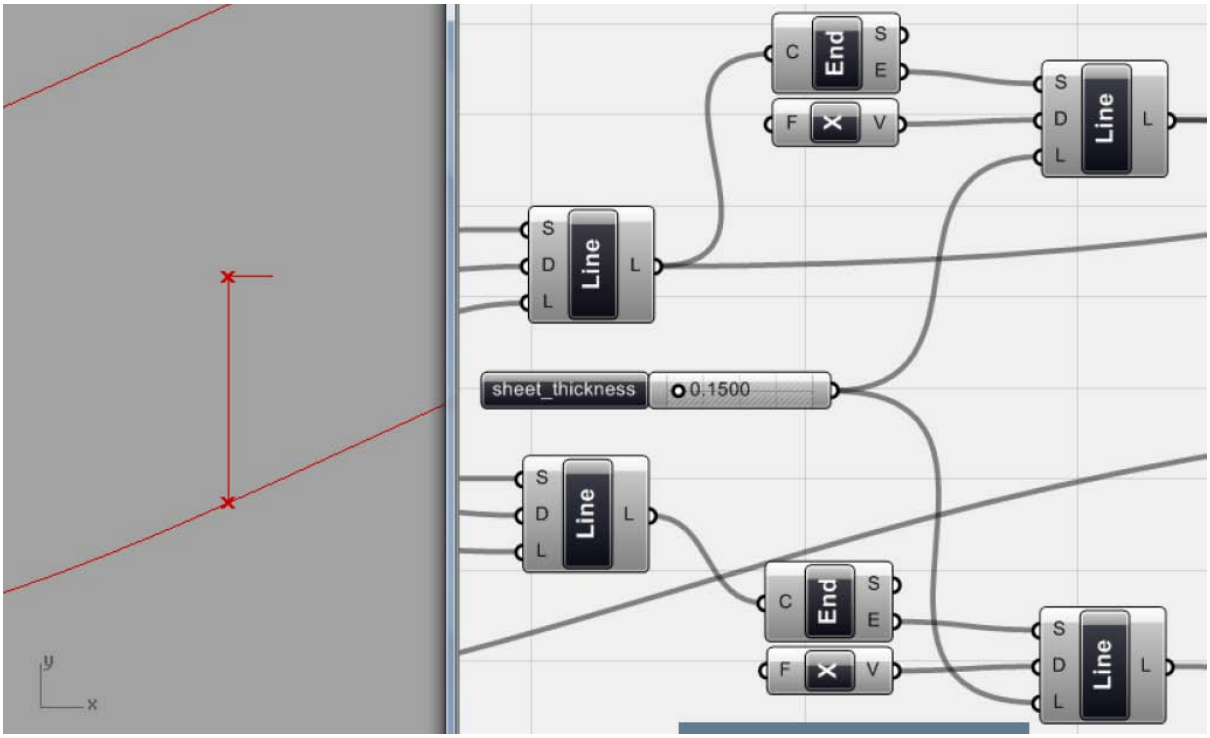


<CCX>를 보면 point 뿐만 아니라 tA와 tB가 있는 것을 알 수 있다. 이는 각 curve 위 교차 point의 parameter를 내보내는 것이다. 이것을 <evaluate curve>의 T에 연결하고 재배열 된 curve들을 C에 연결하면 해당하는 위치의 point를 재배열 된 curve 위에 찾아줄 수 있다. 즉 이 경우 point는 <orient>가 아닌 <evaluate curve>를 이용해서 찾아 준 것이다.

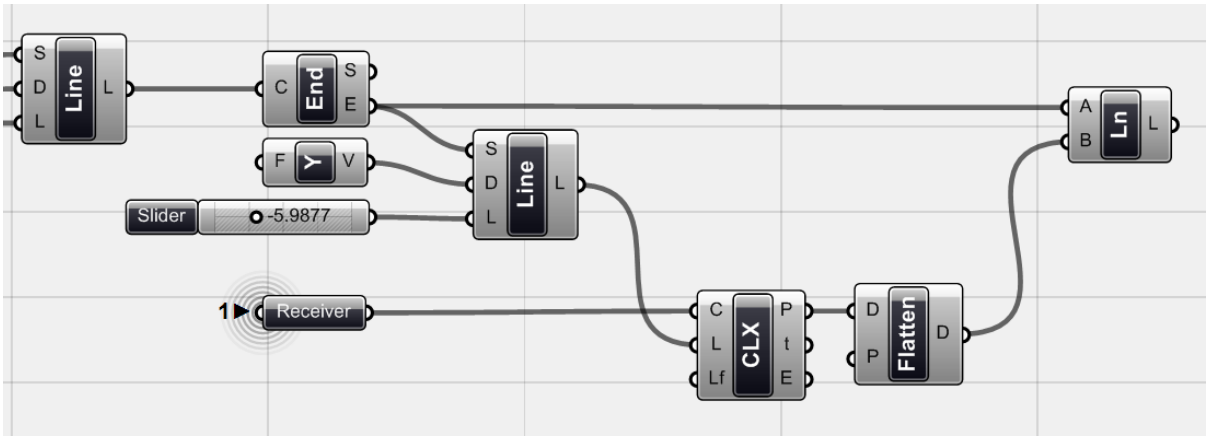


이제 이점으로부터 시작되는 line을 그려보자. 이 때 사용할 수 있는 것은 <line SDL>이다. 이는 S: Start point, D: Direction, L: Length를 input으로 가진다. <unit Y>를 방향에 연결하고 <offset>에 사용한 값을 <f2: x/2>를 이용하여 L에 연결하여주자.

`



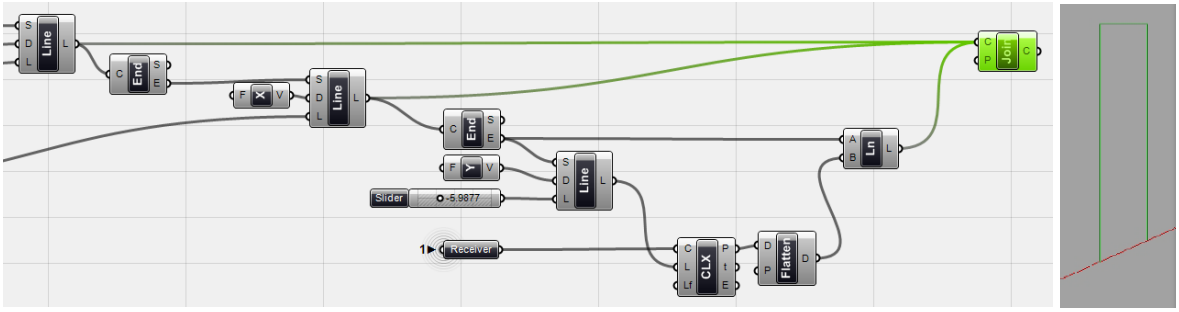
이렇게 그려진 line에 <end point>를 이용하여 찾은 끝점을 찾아주자. 이것에 다시 <line SDL>을 연결하여 X 방향으로 line을 그려준다. 이 길이는 <sheet_thickness>가 된다.



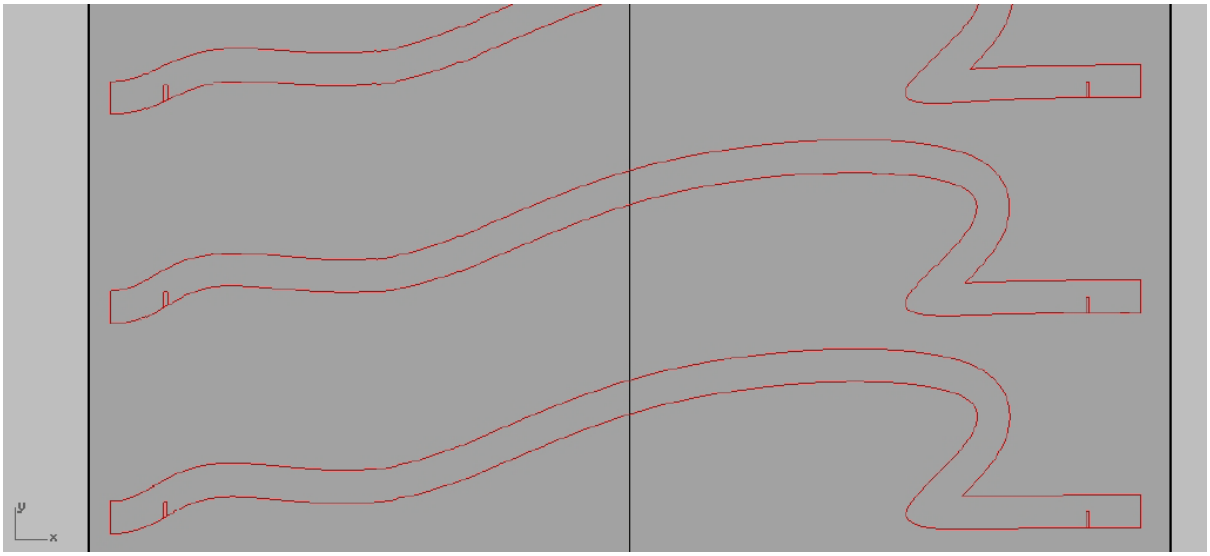
이제 이 X방향으로 그려진 line의 끝점에서 처음 시작했던 curve로 돌아가보자. 먼저 <end point>를 이용하여 끝점을 찾아준다. 여기에 <line SDL>을 연결해주자. 이 때 line의 방향은 Y의 반대방향이므로 <unit Y>와 함께 음수를 L에 연결해주면 된다. 길이의 경우 정확한 길이를 알 수 없으므로 일단 충분히 긴 line을 그린 뒤 이것의 교차점을 찾아주자. 이번에는 line과 curve의 교차이므로 <CLX>(Intersect > Mathematical > Curve | Line)을 이용하면 된다. 이 point는 tree 구조를 가지고 있게 되는데 이를 <flatten>을 이용하여 data list로 바꿔주도록 하자.

위에서 X방향으로 그린 선의 end point와 방금 전에 찾은 교차point를 각 시작과 끝으로 하는 line을 그려주면 된다.

`



이제 <joint curves>(Curve > Util > Join curves)를 이용하여 각 line을 join 시켜주자.
위를 똑같이 다른 쪽의 점에도 적용시켜 준다.



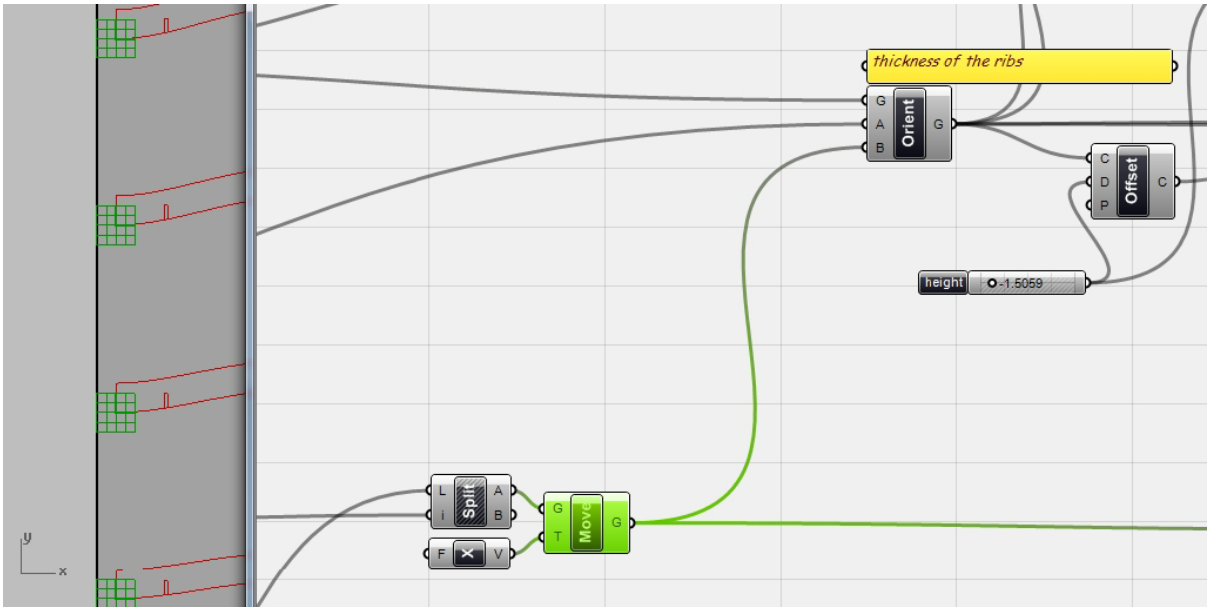
Labeling

fabrication으로 넘어가기에 앞서 각 부재들에 labeling을 해보자. 잘라야 할 것이 수백 개에 이른다면 부재를 구분하기 위해서라도 꼭 labeling을 해주는 것이 좋다. 이때 label은 조립의 순서에 따르는 것이 좋다.

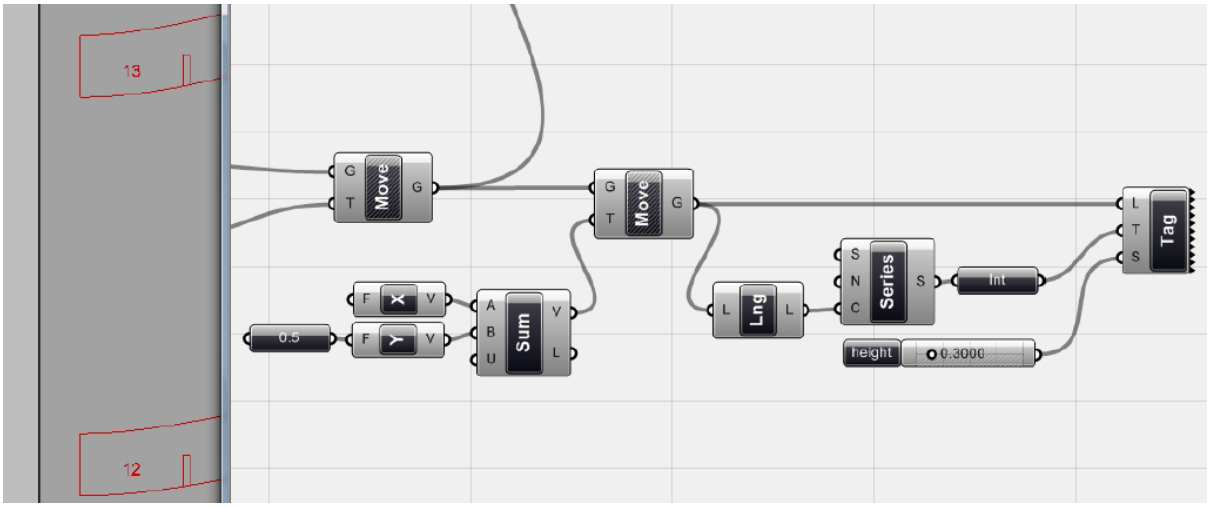
Label은 수와 text의 조합으로 이루어질 것이다. 즉 하나의 열에 여러 개의 부재들이 있다고 했을 때 left_wing_01 과 같은 방식으로 labeling을 할 수 있다.

이 예제에서는 일련의 수를 적용하여 각 부재의 위치를 표시해보도록 하자. <text tag>의 input으로는 label이 표시 될 frame과 표시할 text의 내용, 그리고 text의 크기가 있다.

`



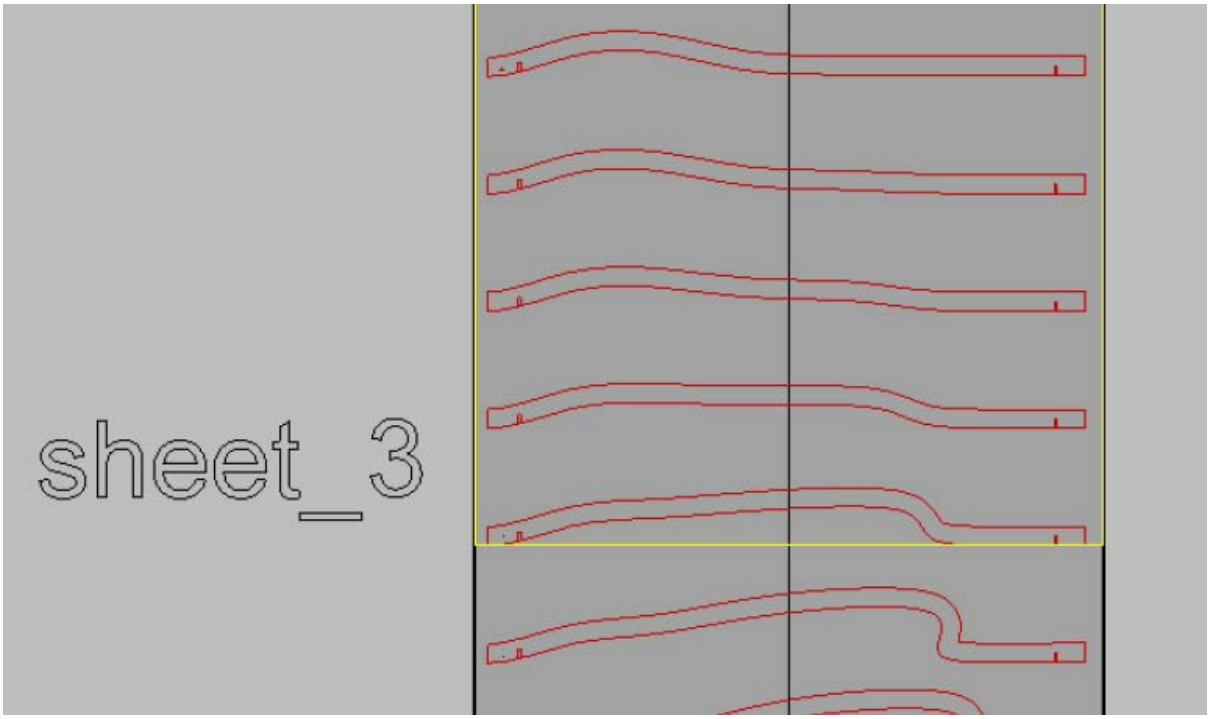
각 방향의 부재의 수는 section curve를 <orient>하는데 사용했던 frame과 그 수가 일치한다. (위에서 선택된 component가 바로 우리가 <orient>에 연결하였던 frame의 list 이다.) 이 plane의 개수를 <list length>를 이용하여 찾은 뒤 그 수를 <series>의 C에 연결해준다.



여기서 나오는 수의 data list를 <text tag>의 T에 연결해주자. 이 때 <integer>는 수에 붙어있는 .0을 없애주는 역할을 한다. 즉 수가 12.0 인 경우 이것을 12로 바꿔주는 것이다.

부재의 list를 <text tag>의 L에 그대로 연결하면 label들이 각 부재의 꼭지점에 있는 frame의 중심에 표시되게 된다. 그러므로 이 전에 frame을 이동시키고 이것을 꼭지점에 있는 plane을 X 방향으로 1만큼, Y 방향으로 0.5 만큼 이동시켜보자. 이 경우 <sum> 을 이용하여 두 vector를 합한 뒤 <move>에 적용하였다. 이것을 <text tag>의 L에 연결시켜주면 된다.

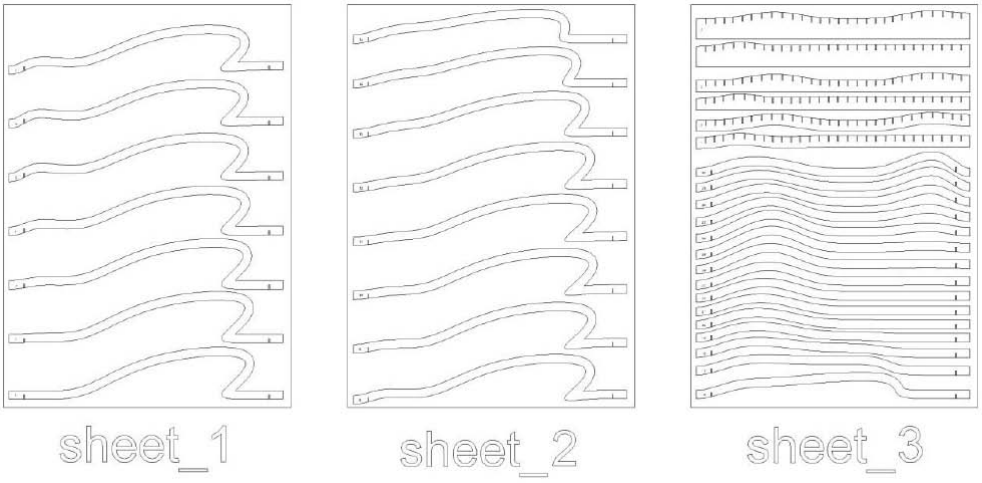
`



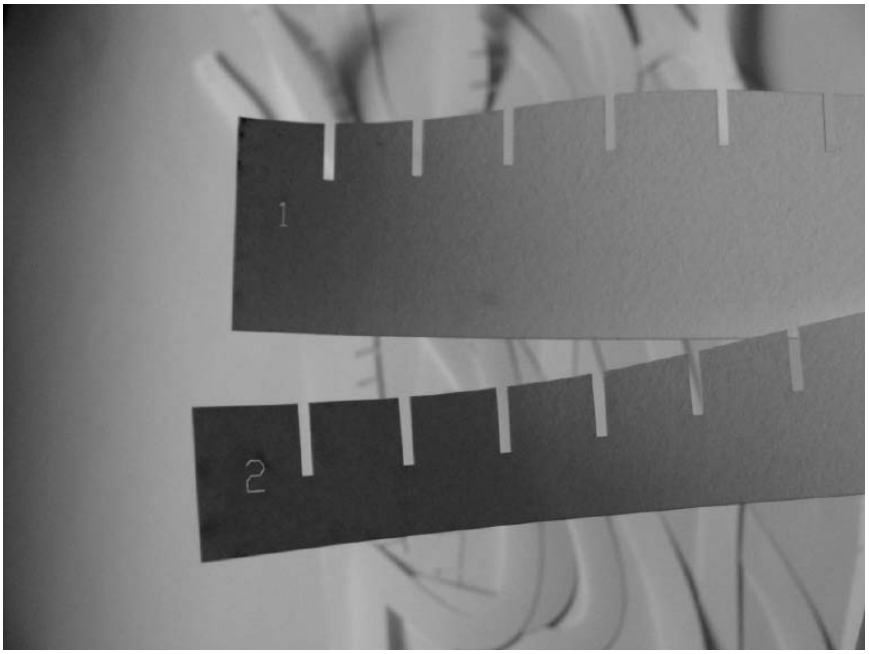
이제 부재들이 그려지는 surface를 바꿔가며 부재들을 그려주면 된다. 이때 재료를 낭비하지 않도록 주의하자. 위 그림의 경우 sheet_3에서 부재의 일부가 sheet의 끝에 닿는다는 것을 알 수 있다. 이것을 조절해주기 위해 Grasshopper의 algorithm을 바꿀 수도 있다. 하지만 이것을 bake 한 뒤 Rhino에서 자르기 전에 이동시켜주는 것이 더 간단하다. 즉 어떻게 작업을 하는 것이 더 쉬울지를 잘 판단하는 것이 중요하다.

자르기(Cutting)

모든 준비가 끝났으면 이제 curve를 bake 한 뒤 부재를 자를 준비를 해보자. 아래에서 보이는 것 처럼 모든 부재를 재료의 사이즈와 양에 맞춰 잘 배열해보자. 이제 laser cutter를 사용할 준비가 모두 끝났다.



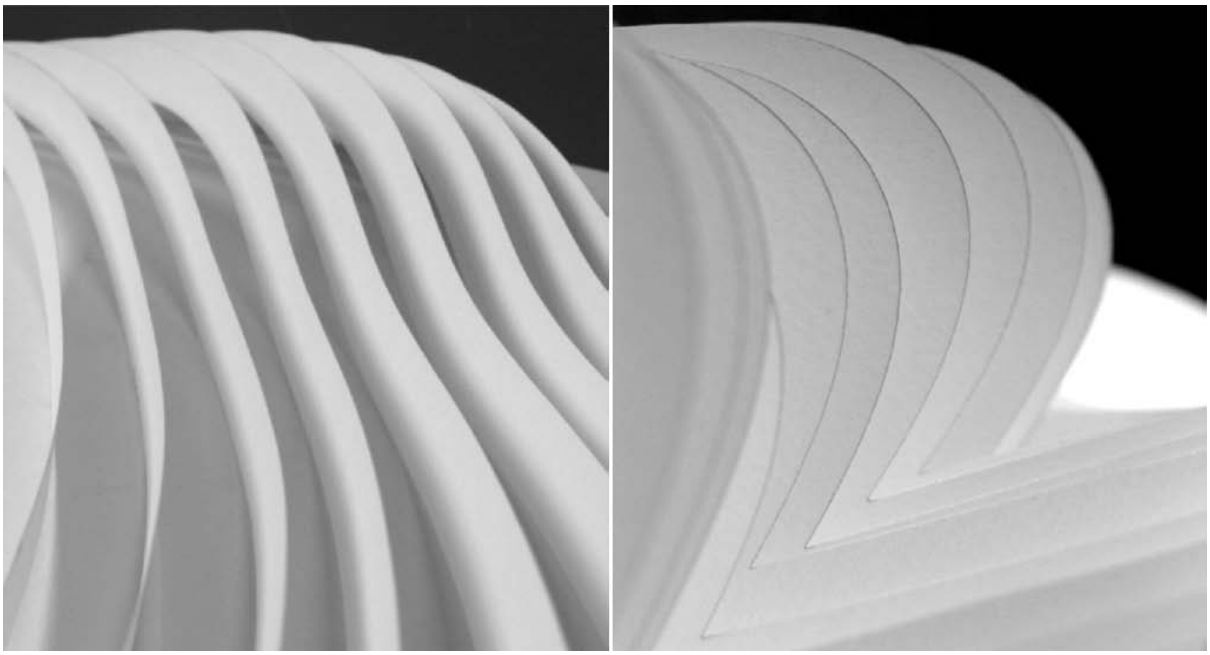
`



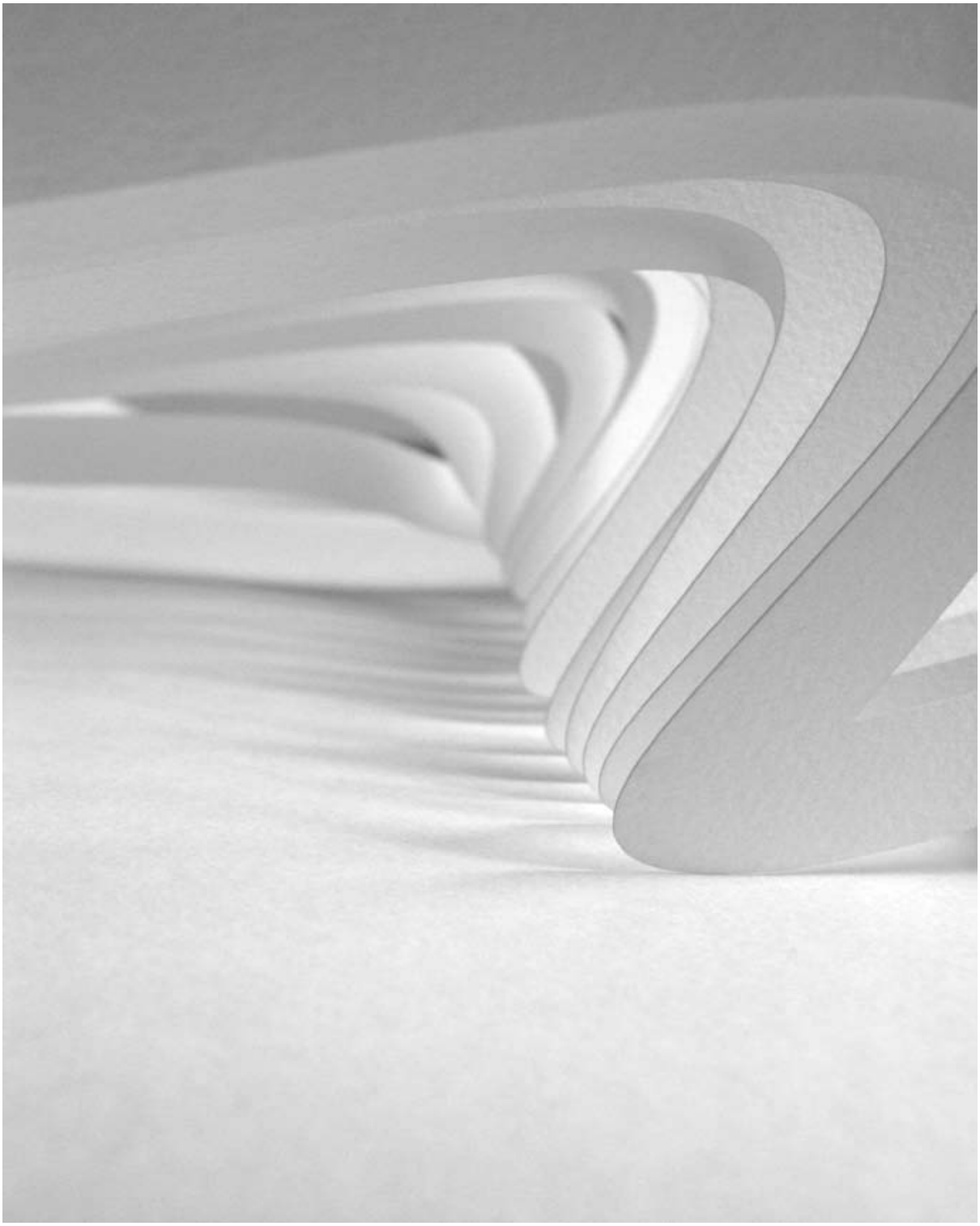
잘려진 부재, 조립할 준비가 되었다

조립(Assembly)

이 예제의 경우 부재의 조립은 어렵지 않다. 3d model과 비교해가며 조립하면 조립 순서와 부재의 위치를 쉽게 찾을 수 있을 것이다. 조립이 끝나면 종이를 이용하여 만든 surface가 완성될 것이다.



`

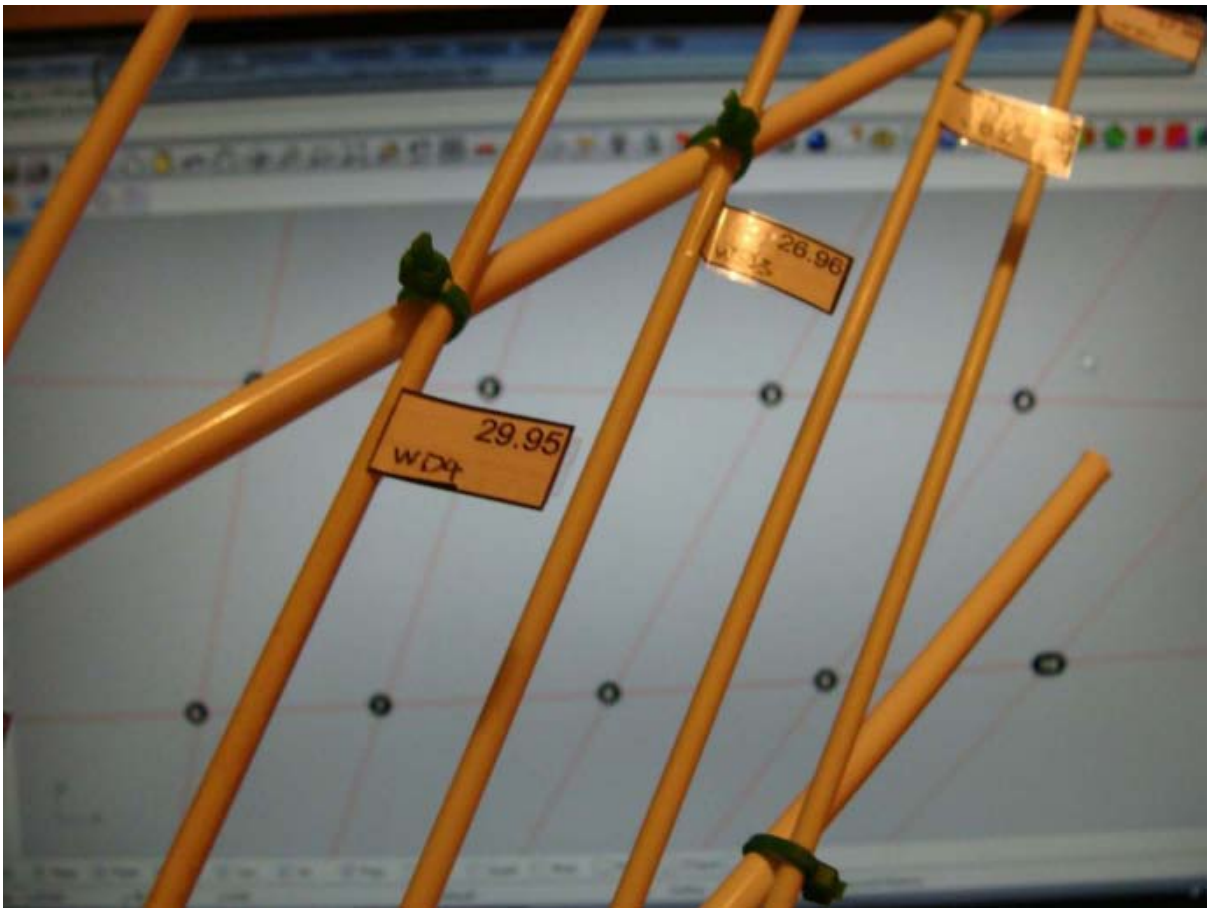


완성된 model

Fabrication을 잘 하기 위해서는 여러 가지 측면을 고려해야 한다. 어떠한 형상이어야 하는가, 어떠한 재료를 사용해야 하는가, 어떤 제작도구를 이용하는가, 어떻게 조립할 것인가 등의 여러 사항에 따라 그 결과물이 달라질 수 있기 때문이다. 또한 project에 따라 fabrication에 필요한 data를 잘 추출해내는 것이 중요하다. 특히 결과물이 복잡한 기하체라면 조립의 규칙을 먼저 생각해

`

보는 것 또한 무척 중요하다.



조립 방식; 흔한 재료와 간단한 joint로 제작하였지만 재료가 어떻게 만나는가와 점의 위치 data가 없다면 모형을 원하는 대로 제작할 수 없을 것이다.

`

Chapter_9_Design Strategy

`

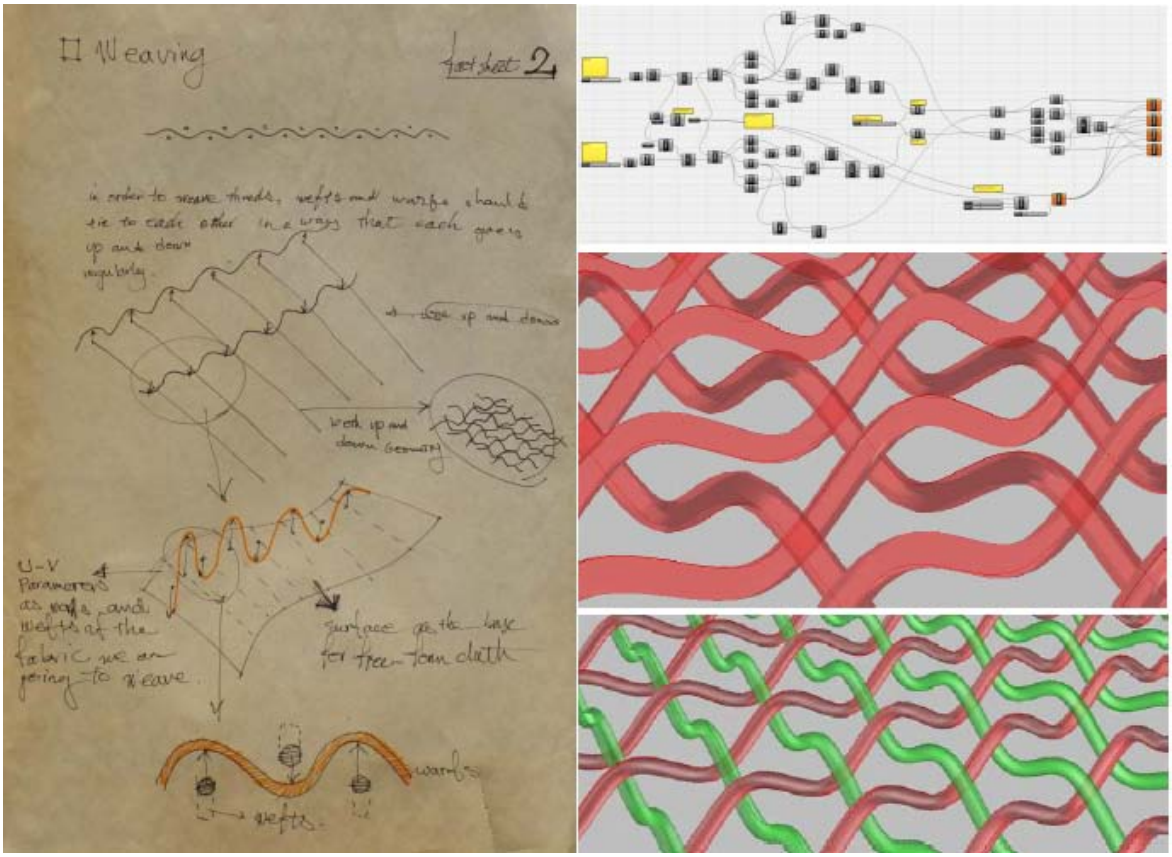
Design 전략(Design Strategy)

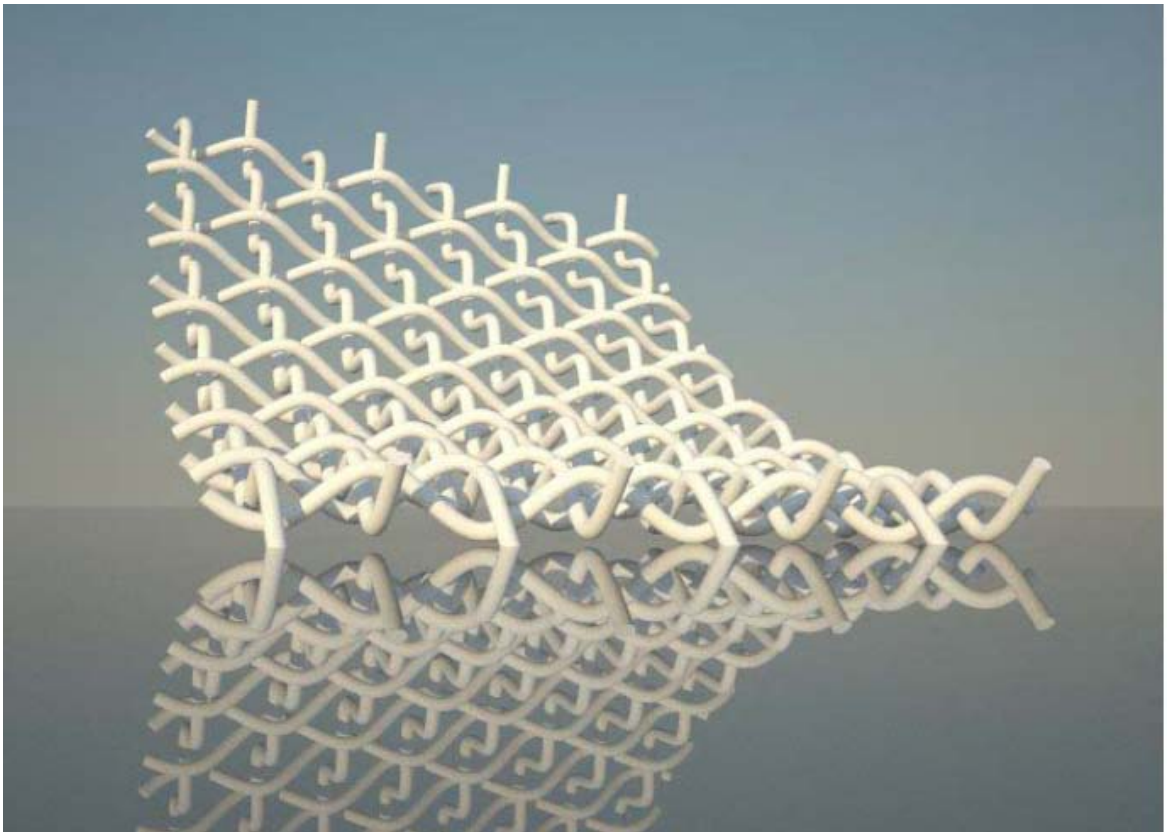
Generative Algorithm 이란 parameter와 algorithm을 이용하여 Design 과정에서 생길 수 있는 기하학적 issue들을 해결하여 가는 것이다. 이것은 기존의 방식에 비하여 훨씬 많은 양의 data와 계산과정을 포괄할 수 있는 방법이다. 이는 일반적인 tool이나 이것의 명령어에 의하여 한정되어 있는 design 방식을 뛰어넘어 무한한 가능성을 살필 수 있는 것이다. Design Algorithm을 짤 수 있는 방식은 무척 여러 가지가 있으며 이것의 변용에 따라 여러 결과물을 도출해낼 수 있다.
물론 grasshopper 와 같은 parametric modeling software 또한 이것의 component에 의한 제약이 있다고 생각할 수 있다. 하지만 이것은 사고의 process를 시각화시킨 것으로 이것을 다양하게 응용한다면 창의력을 제안하는 한계를 분명 넘어설 수 있을 것이다.

무언가를 design 하기 위해서는 명확한 design strategy가 필요하며 이는 가장 효율적인 algorithm을 짜기 위해 꼭 필요한 것이다. Design 결과물의 특징을 생각하며 특정 부분은 그려보고, 또 physical model을 만들어 보는 것이 더욱 명확한 algorithm의 logic을 짜는데 도움을 줄 것이다. Algorithm을 짜기에 앞서 그 값이 변하지 않는 parameter와 변수가 되는 parameter를 구분해보고, 결과물이 필요로 하는 수치 data가 무엇인지 미리 생각해보자. 결과물을 분석적으로 이해해보고 그것에 이르는 과정에서 생길 수 있는 design issue들을 예상해본다면 algorithm을 짜는데 많은 도움이 될 것이다.

즉 Algorithm을 이용하여 design을 하려면 그것의 방식으로 사고하는 것이 중요하다.!

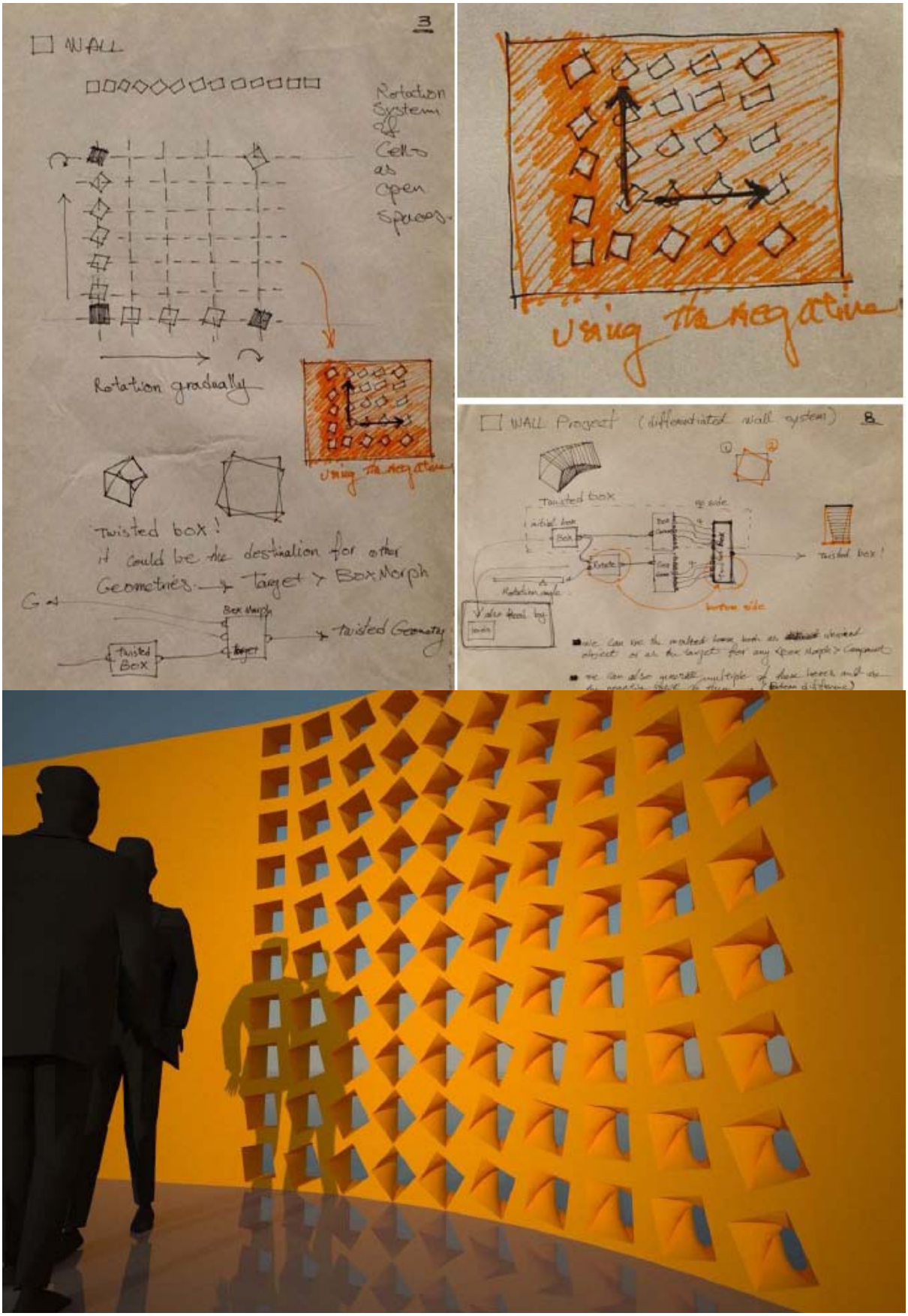
`





Weave project; 분석적 사고를 이용한 통합적인 modeling의 결과물

`



Porous Wall project; 분석적 사고를 이용한 통합적인 modeling의 결과물

`

Bibliography

Pottman, Helmut and Asperl, Andreas and Hofer, Michael and Kilian, Axel, 2007: 'Architectural Geometry', Bently Institute Press.

Hensel, Michael and Menges, Achim, 2008: 'Morpho-Ecologies', Architectural Association.

Rutten, David, 2007: 'Rhino Script 101', digital version by David Rutten and Robert McNeel and Association.

Flake, Gary William, 1998: 'The computational beauty of nature, computer explorations of fractals, chaos, complex systems, and adaptation', The MIT Press.

De Berg, Mark and Van Kreveld, Marc and Overmars, Mark, 2000: 'Computational Geometry', Springer.

Grasshopper tutorials on Robert McNeel and Associates wiki:
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryExamples.html

Axel Kilian and Stylianos Dritsas: 'Design Tooling - Sketching by Computation',
http://www.designexplorer.net/designtooling/inetpub/wwwroot/ s/sketching/index.html

Wolfram Mathworld: http://mathworld.wolfram.com/

Stylianos Dritsas, http://jeneratiff.com/

Main Grasshopper web page: http://grasshopper3d.com/

GENERATIVE ALGORITHMS

using GRASSHOPPER

Zubin Khabazi

Ver.02

www.MORPHOGENESISM.com