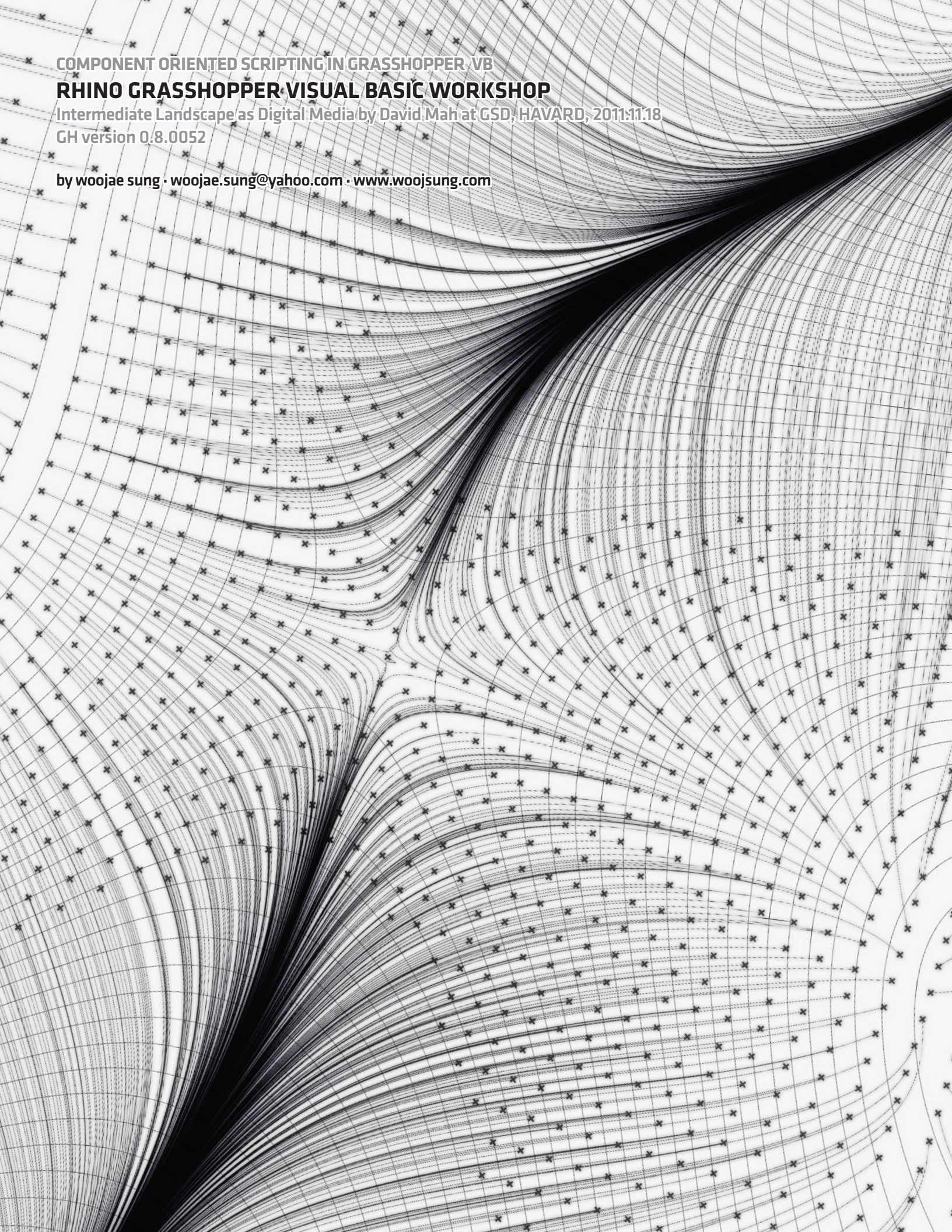
COMPONENT ORIENTED SCRIPTING IN GRASSHOPPER_VB

RHINO GRASSHOPPER VISUAL BASIC WORKSHOP

Intermediate Landscape as Digital Media by David Mah at GSD, HAVARD, 2011.11.18

GH version 0.8.0052

by woojae sung · woojae.sung@yahoo.com · www.woojsung.com



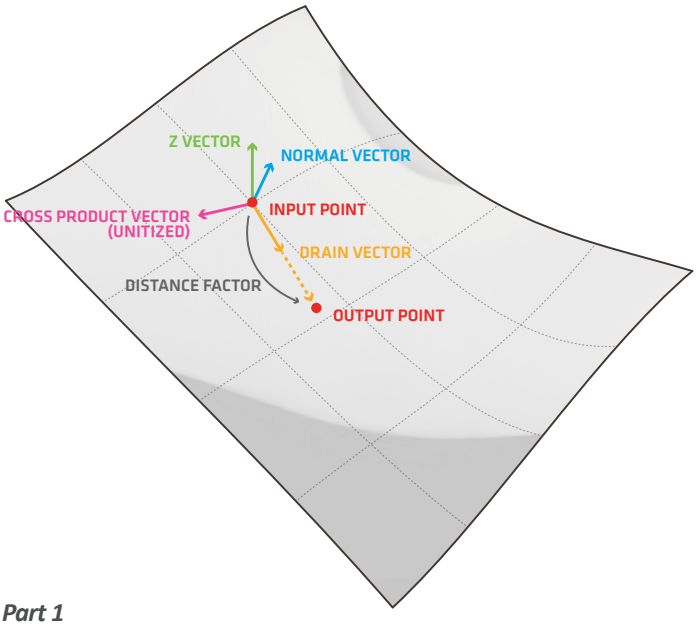
IDEA

이번에는 그래스하퍼의 비쥬얼베이직 스크립팅을 이용하여 경사진 지형에서 흘러 내리는 물의 흐름을 재현해보도록 하겠습니다. 본 프로세스는 크게 세가지 단계로 나누어 집니다.

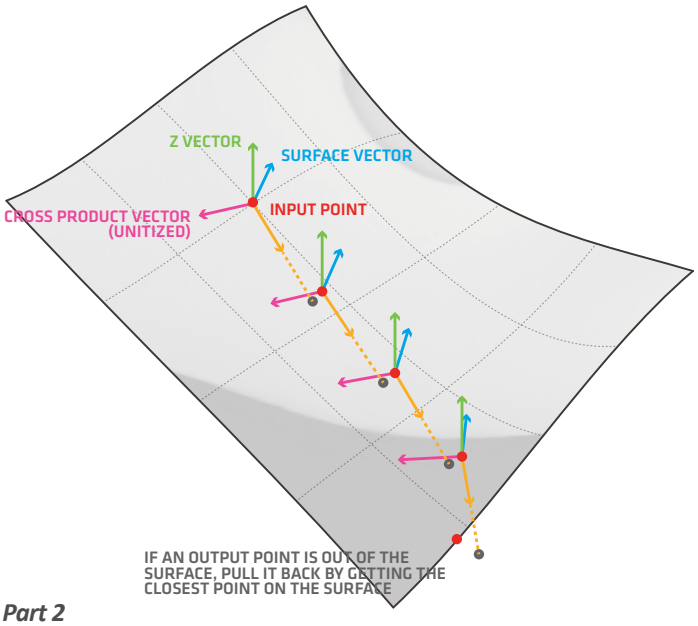
Part 1 - 첫번째로 경사진 지형의 임의의 점**('input_point')**에서 물이 흐르는 방향의 벡터**('drain_vector')**를 찾아냅니다. 그리고 이 벡터를 통해 일정 시간후에 그 점이 어떤 위치로 이동이 되었을지**('output_point')** 를 예측해 봅니다. 이는 벡터에 어떠한 계수를 곱함으로써 찾을 수 있는데, 앞으로 이 계수를 **'distance_factor'**이라고 부르기로 합니다. 이 계수는 앞으로의 프로세스가 얼만큼 정확할지를 결정하게 됩니다.

Part 2 - 첫번째 단계에서는 우리는 임의의 점인 **'input_point'**로 부터 **'output_point'**를 구했습니다. 만일 우리가 **'output_point'**를 또 다른 **'input_point'**로 사용할수 있다면 이를 이용하여 또다른 **'output_point'**를 구할수 있게됩니다. 이 과정을 더이상 유효한 **'output_point'**를 구할수 없을때 까지 반복한다면, 물 흐름을 예측하는 곡선을 그릴수 있게 됩니다.

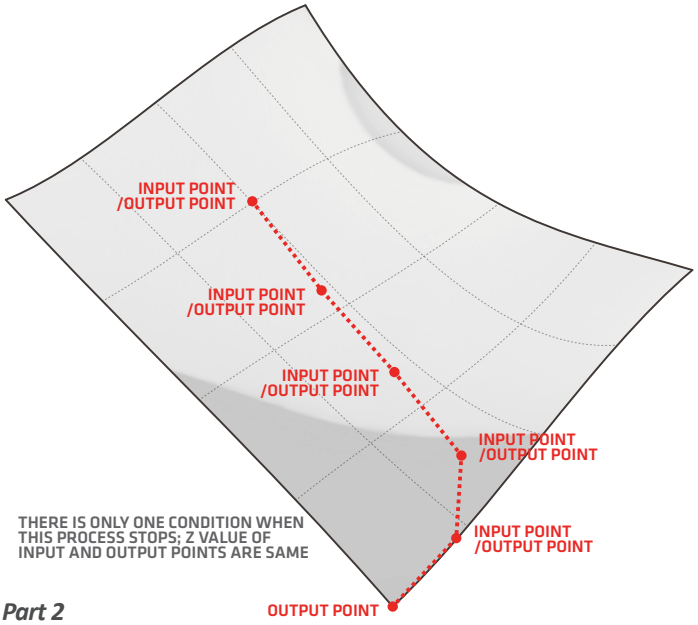
Part 3 - 세번째 단계에서는 복수의 **'input_point'**를 이용하여 복수의 수원에서 시작되는 물의 흐름들을 예측해 보도록 합니다.



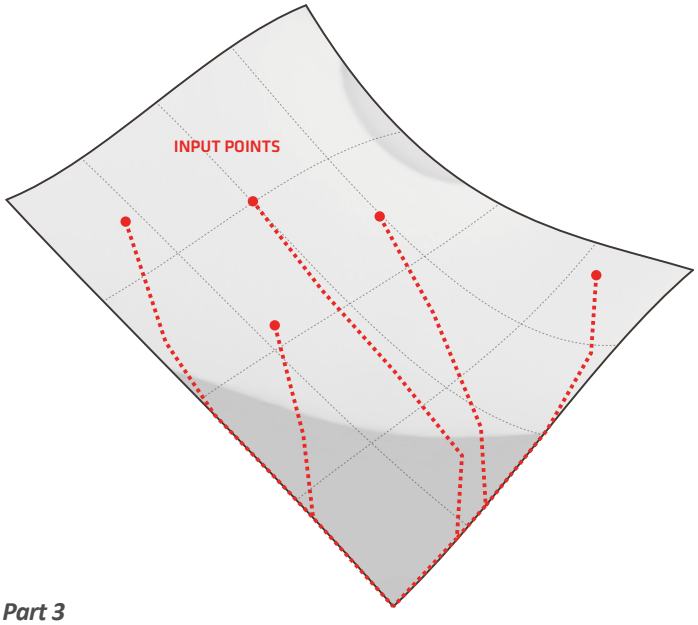
Part 1



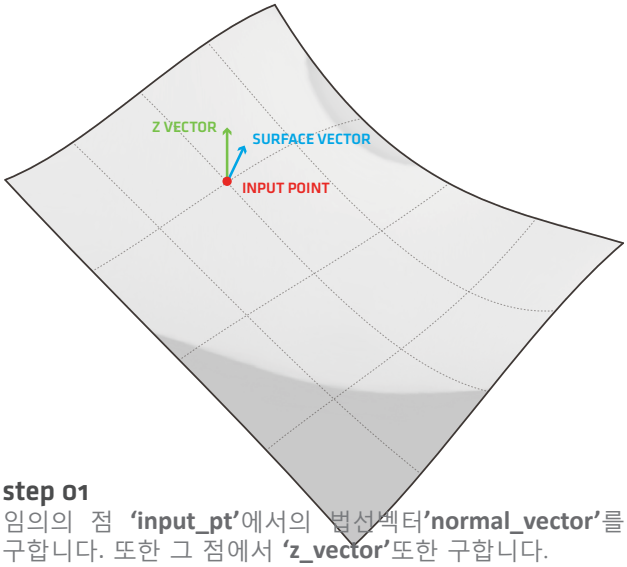
Part 2



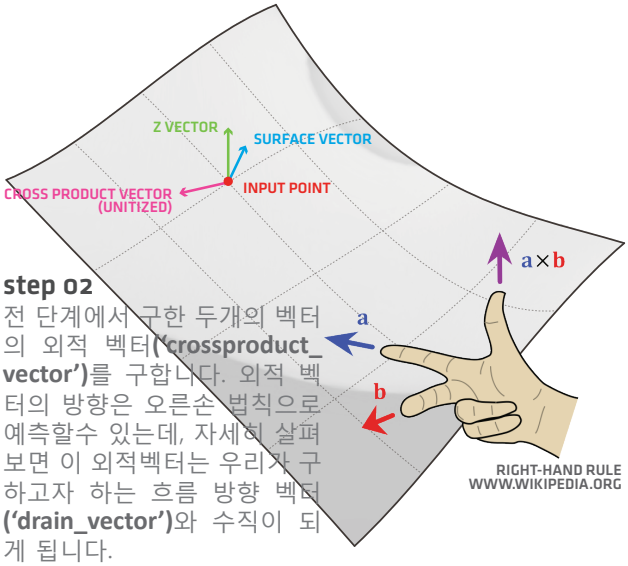
Part 2



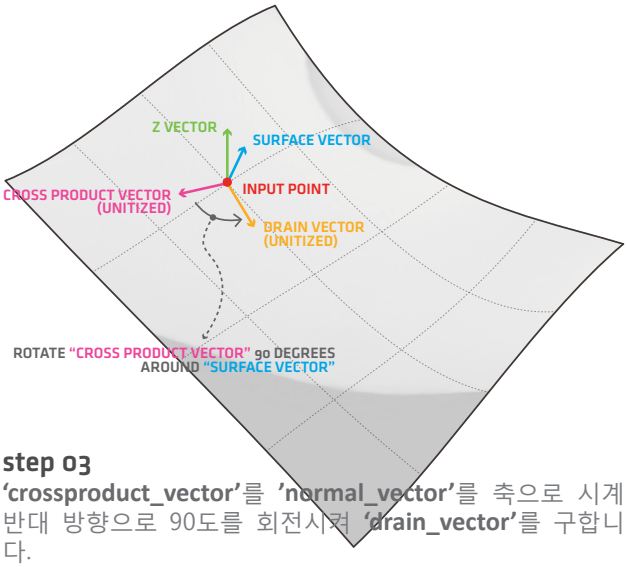
Part 3



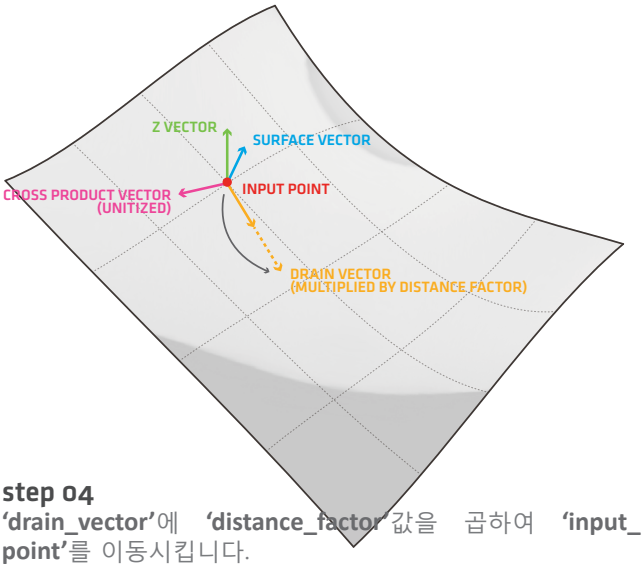
step 01
임의의 점 **'input_pt'**에서의 법선벡터**'normal_vector'**를 구합니다. 또한 그 점에서 **'z_vector'**또한 구합니다.



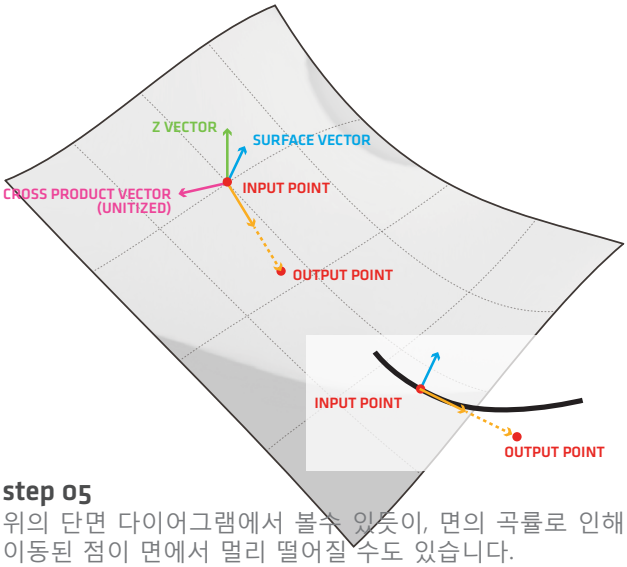
step 02
전 단계에서 구한 두개의 벡터의 외적 벡터**('crossproduct_vector')**를 구합니다. 외적 벡터의 방향은 오른손 법칙으로 예측할수 있는데, 자세히 살펴보면 이 외적벡터는 우리가 구하고자 하는 흐름 방향 벡터**('drain_vector')**와 수직이 되게 됩니다.



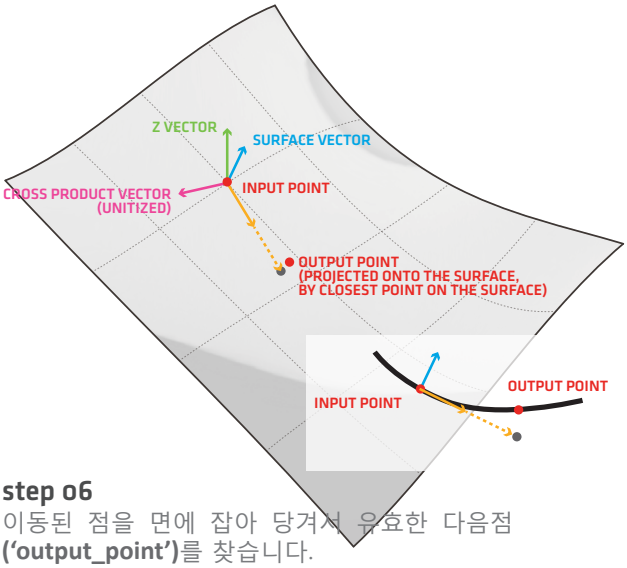
step 03
'crossproduct_vector'를 **'normal_vector'**를 축으로 시계 반대 방향으로 90도를 회전시켜 **'drain_vector'**를 구합니다.



step 04
'drain_vector'에 **'distance_factor'**값을 곱하여 **'input_point'**를 이동시킵니다.

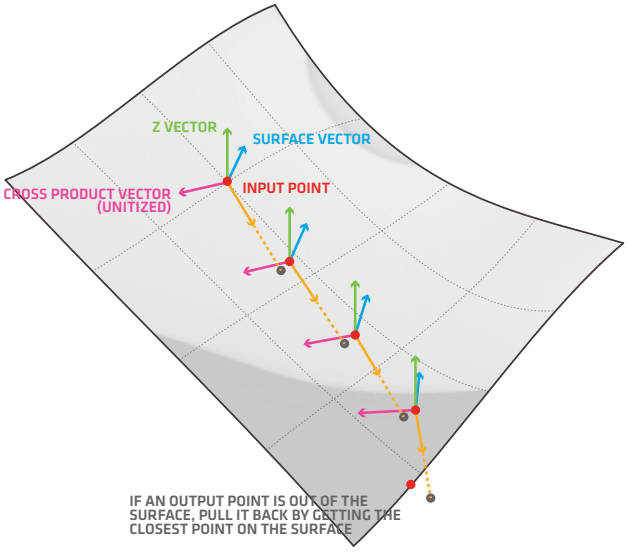


step 05
위의 단면 다이어그램에서 볼수 있듯이, 면의 곡률로 인해 이동된 점이 면에서 멀리 떨어질 수도 있습니다.



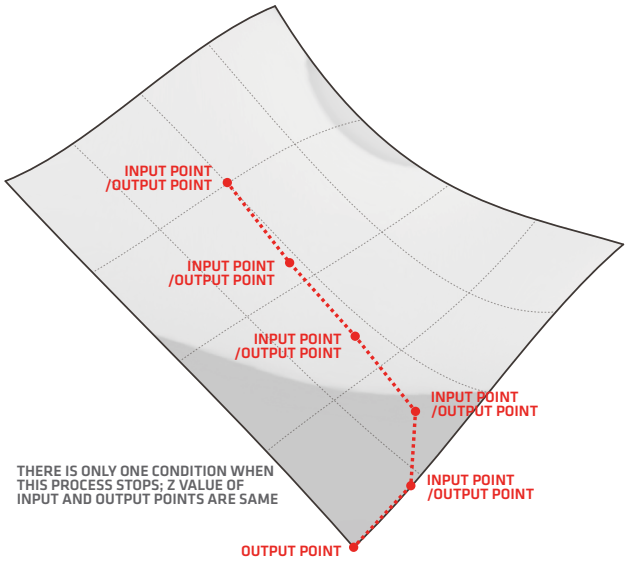
step 06
이동된 점을 면에 잡아 당겨서 유효한 다음점 **('output_point')**를 찾습니다.

PART 2 물흐름선 찾기 (DEFINING A 'FLOW LINE')



step 01

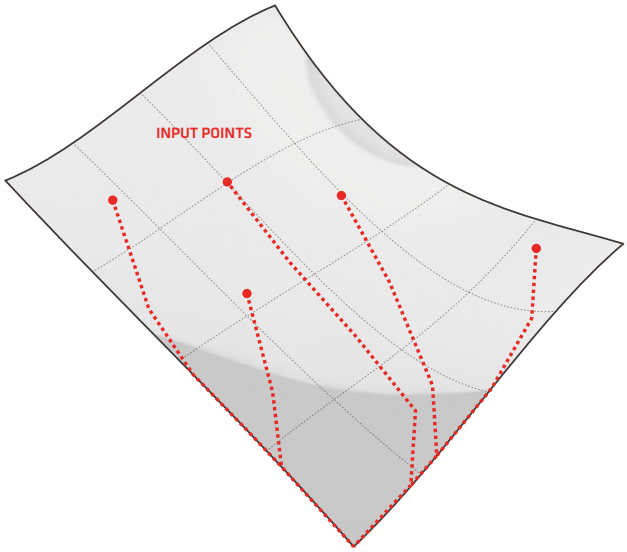
다음점 찾기 과정을 반복합니다. 먼저 첫번째 **'input_point'**를 통하여 첫번째 **'output_point'**를 찾습니다. 그리고 이렇게 찾은 **'output_point'**를 다시 두번째 **'input_point'**로 사용하여 또 다른 **'output_point'**를 찾아냅니다. 그림에서 볼수 있듯이 **'output_point'**가 면 밖으로 나가게 된다면 그 점에 가장 가까운 면 위의 다른점을 찾음으로써 **'output_point'**를 항상 면위에 위치하게 할수 있습니다.



step 02

위의 과정이 끝나게 되는 유일한 경우는 **'input_point'**와 **'output_point'**의 **z**값이 같은 경우입니다. 왜냐하면 이 두점의 높이가 같다는 것은 물이 더이상 흐르지 않는 다는것을 의미하기 때문입니다.

PART 3 복수의 물흐름선 찾기 (APPLYING TO MULTIPLE WATER SOURCES)



step 01

여러개의 **'input_point'**로부터의 물흐름을 예측해 봅니다.

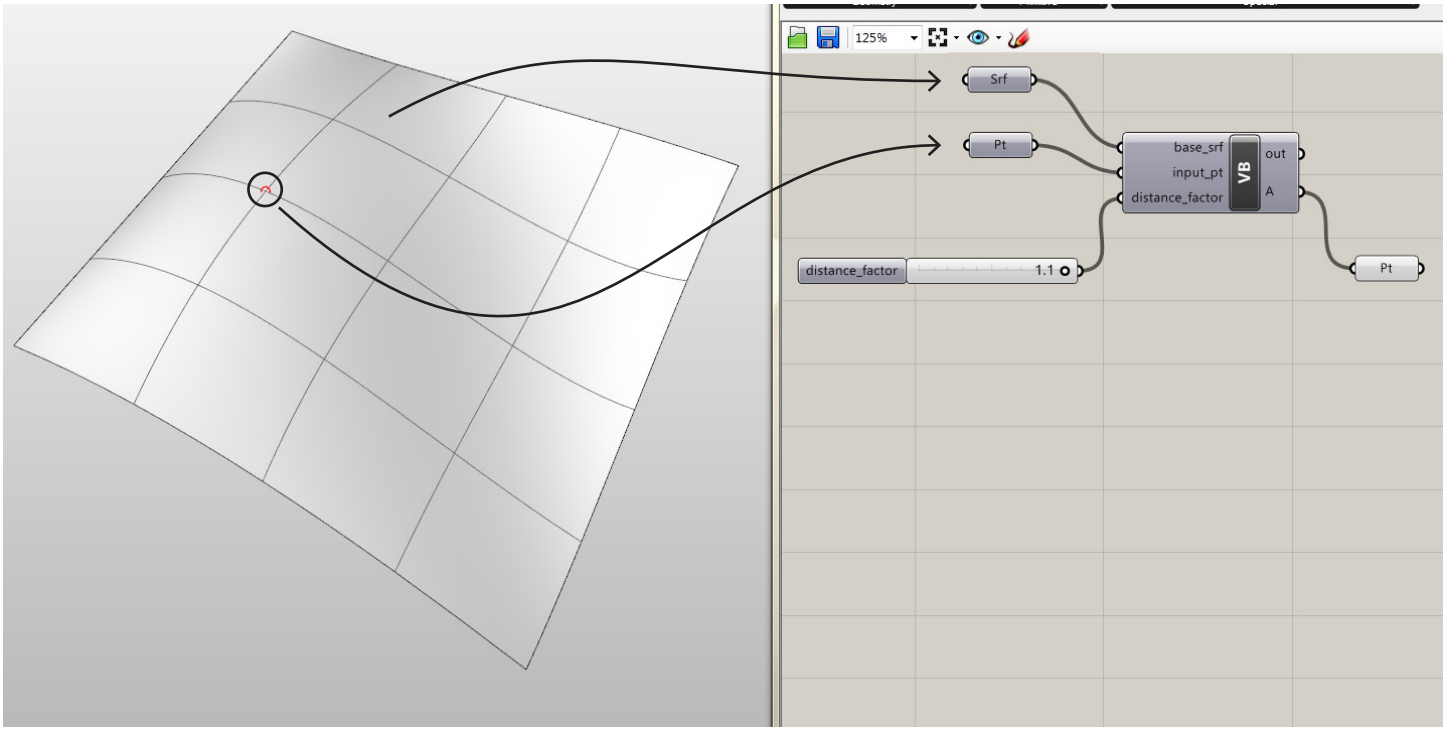
CODE REVIEW

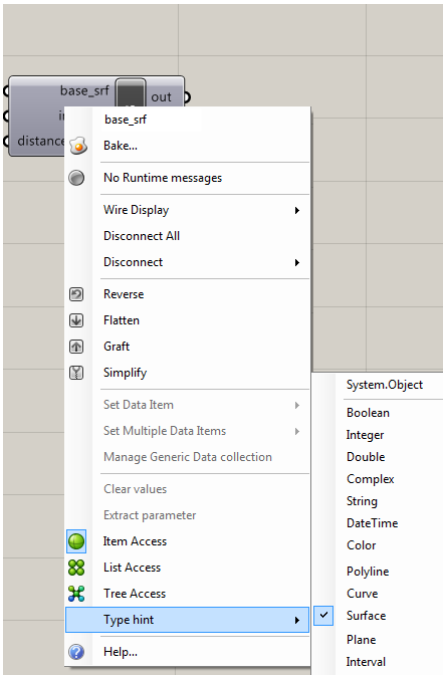
PART 1 다음점 찾기 (FINDING AN 'OUTPUT POINT')

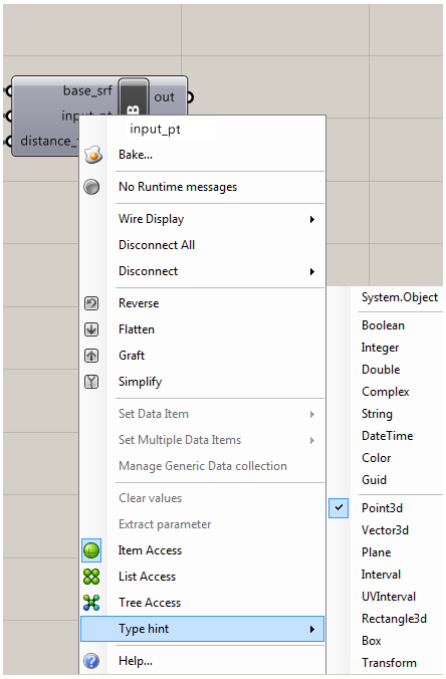
'VB workshop part1.gh' 와 **'VB workshop.3dm'** 파일을 참조합니다.

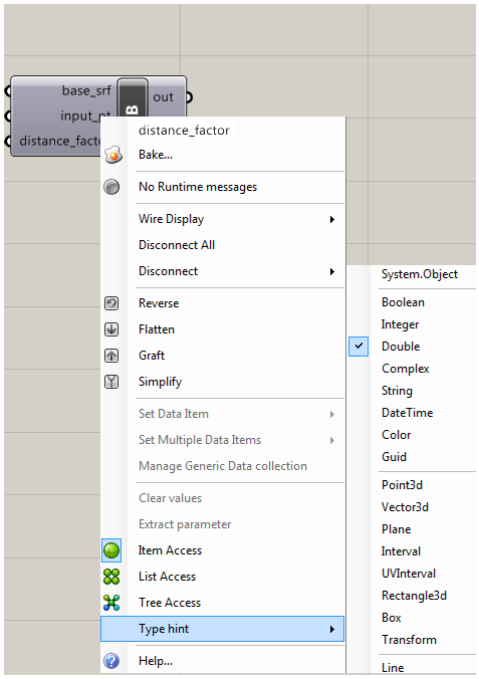
scene setting

다음 그림과 같이 라이노와 그래스하퍼를 설정합니다.









```
85  Private Sub RunScript(ByVal base_srf As Surface, ByVal input_pt As Point3d, ByVal distance_factor As Double, ByRef A As Obj
86
87      Dim u, v As Double
88
89      base_srf.ClosestPoint(input_pt, u, v)
90
91      Dim normal_vector As Vector3d
92
93      normal_vector = base_srf.NormalAt(u, v)
94
95      Dim drain_vector As vector3d = vector3d.CrossProduct(normal_vector, vector3d.ZAxis)
96
97      drain_vector.Unitize
98
99      drain_vector.Transform(Transform.Rotation(Math.PI * 0.5, normal_vector, input_pt))
100
101     Dim moved_pt As point3d = input_pt + distance_factor * drain_vector
102
103     base_srf.ClosestPoint(moved_pt, u, v)
104
105     Dim output_pt As Point3d = base_srf.PointAt(u, v)
106
107     A = output_pt
108
109 End Sub
```

step 1 이번 첫 단계에서는 주어진 경사면**('base_surface')**위의 한점**('input_point')**에서의 법선 벡터**('normal_vector')**와 z방향 벡터**('z_vector')**를 구합니다. 가장 먼저 법선 벡터**('normal_vector')**를 정의하는 것으로 부터 시작합니다.

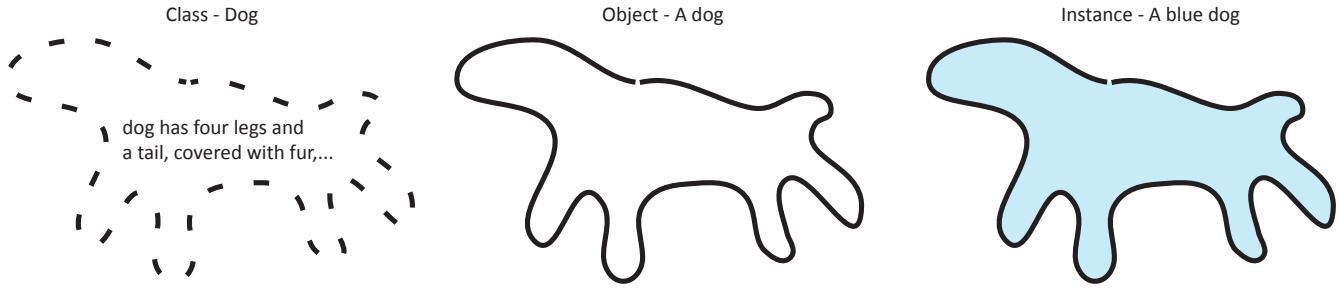
91 Dim normal_vector as Vector3d

우리가 비쥬얼베이직에서 **'Dim A as B'**라고 어떤 변수를 정의할때, **A**는 그 변수의 이름이고 **B**는 그 변수가 어떤 종류에 속하는지를 알려줍니다. 즉 그 종류라고 함은 예를 들어 point3d, integer, vector3d, surface등 라이노에서 찾아 볼수 있는 모든 종류의 객채를 말합니다. 우리는 이러한 종류를 **'class'**라고 부릅니다.

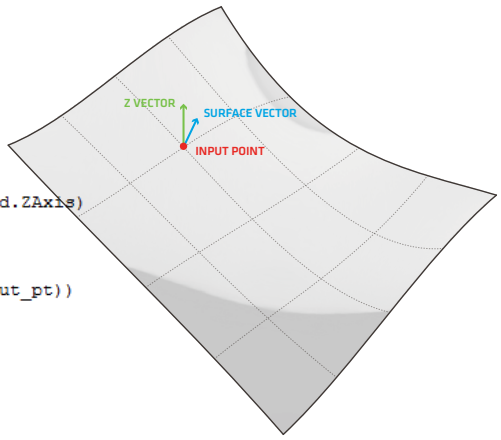
class는 보다 큰 개념으로 그 객채가 어떠한지를 정의합니다. 즉 '개'라는 개념은 '다리가 네개이고 꼬리가 하나이며 털로 덮여있다'는 식으로 정의 될 수도 있는 것입니다. **class**는 결국 어떤 색을 가졌는지 크기가 얼만한 개인지에 대해 아무런 언급도 없이 다리가 몇개이고 꼬리가 몇개인지등의 특정한 속성을 가지면 '개'라고 불리울수 있다는 '분류'의 개념이라고 생각하면 됩니다.

object는 **class**보다 좀더 세분화되는 하위 분류체계입니다. 위 91번째 라인에서 처럼 정의될때, **'normal_vector'**는 벡터라는 클래스에 속하는 하나의 객채이며 아직 여전히 물리적인 특성을 가지고 있지는 않지만 적어도 **'normal_vector'**라는 이름을 가지고 있는 하나의 저장소가 되는 것입니다. 이 저장소에는 어떠한 벡터도 저장이 될수 있습니다. 예를 들어, 우리가 만약에 '철수'라는 개이름을 짓는다고 하면, 우리가 이 이름을 특정한 개에게 부여하기 전에 이 이름은 어떤 개를 위해서도 사용될수 있습니다.

instance는 위의 두개념보다 더욱 작은 하위개념으로 측정이 가능한 물리적인 요소를 가지고 있습니다. 91번째 라인에서 처럼 변수를 정의할때는, 위에서 말했듯이, 저장소를 제공하는 개념입니다. 하지만 만약 **'Dim normal_vector as New Vector3d(pointA, pointB)'**라고 정의한다면, 이는 점 A에서 출발하여 점 B에서 끝나는 **normal_vector**라는 이름을 가진 벡터가 됩니다. 결국 이는 저장소가 아닌 특정한 벡터 자체를 정의하게 되는 것입니다. 다시 개의 예로 돌아가면, 이 개의 이름이 무엇이며 이에 더해 개의 털 색이 어떤지, 키는 어떤지, 몸무게는 얼마나 되는지에 대해 자세히 상술하게 되는 것이 **instance**가 됩니다.



```
85  Private Sub RunScript(ByVal base_srf As Surface, ByVal input_pt As Point3d, ByVal distance_factor As Double, ByRef A As Obj
86
87      Dim u, v As Double
88
89      base_srf.ClosestPoint(input_pt, u, v)
90
91      Dim normal_vector As Vector3d
92
93      normal_vector = base_srf.NormalAt(u, v)
94
95      Dim drain_vector As vector3d = vector3d.CrossProduct(normal_vector, vector3d.ZAxis)
96
97      drain_vector.Unitize
98
99      drain_vector.Transform(Transform.Rotation(Math.PI * 0.5, normal_vector, input_pt))
100
101     Dim moved_pt As point3d = input_pt + distance_factor * drain_vector
102
103     base_srf.ClosestPoint(moved_pt, u, v)
104
105     Dim output_pt As Point3d = base_srf.PointAt(u, v)
106
107     A = output_pt
108
109 End Sub
```



93 normal_vector = base_srf.NormalAt(u, v)

이번에는 앞에서 정의한 '**normal_vector**'라는 저장소에 실제로 법선벡터를 구해서 할당시켜 주도록 해봅니다. 법선벡터를 구하는 것은 당연히 어떠한 '면'이 먼저 존재하지 않으면 성립되지 않습니다. 그렇다면, 이러한 기존에 존재하는 면에 법선벡터를 구하도록 명령하는 어떤 방법이 있어야 할것입니다.

앞장에서 살펴본듯이 라이노/그래스하퍼의 모든 객체들은 **class**, **object**, 또는 **instance**로 정의될수 있습니다. 우리는 항상 이러한 객체들에게 무엇인가를 하도록 부탁해야 하는 경우를 경험하게 됩니다. 크게 세가지의 방법이 있습니다. **constructor**, **method**, 와 **property**가 그러한 방법들입니다.

Constructor는 그 이름에서 알수 있듯이 객체 자체를 만드는 방법에 대해 서술합니다. 즉 우리가 어떤 선을 만들려고 할때 우리는 다음과 같이 코딩할수 있습니다. **'Dim A as New Line(pointA, pointB)'.** 이는 점A과 점B를 잇는 선을 그리라는 말입니다. 이때 이렇게 선을 그리는 **'Line(~)'**에 해당하는 부분이 바로 **constructor**입니다.

constructor가 객체 자신을 정의하는 방법에 대한 기술이었다고 하면, **method**는 기존의 정의되어진 객체로 부터 어떤 부가적인 정보나 파생되는 또다른 객체를 얻을 목적으로 무엇인가를 하도록 명령하는 방법입니다. 93번라인에서의 경우처럼 기존의 정의된 면과 그 위의 한 점으로 부터 법선벡터를 얻어내고자 하는것은 결국 기존 정의된 면과 그위의 점에서 면을 평가하여 특정 정보를 추출하는 과정입니다. 이러한 기존 정의된 객체에 대한 명령을 내리는 것이 **method**입니다.

Property는 **method**와 아주 유사합니다. 차이점은 **method**와는 달리 **property**는 가공된 이차의 정보를 추출하는것이 아니라 특정의 객체가 이미 가지고 있는 특성값을 얻어내는 방법입니다. 위의 예에서처럼 어떠한 계산의 과정을 거쳐 얻을 수 있는 정보가 아닌, 예를 들어 기존의 면이 생성되면서 부터 지니고 있는 면적과 같은 값을 **property**를 통해 추출할수 있습니다.

이러한 명령들을 객체에게 내리는 방법은 의외로 간단한데, 바로 객체의 이름 뒤에 점을 찍은후 원하는 명령을 적어넣으면 됩니다. 우리의 경우에는 주어진 면 **'base_srf'**의 어떠한 점에서의 법선벡터 **'normal_vector'**을 구해야 하므로 **'base_srf'**바로 뒤에 **'.'**을 찍음으로써 이 상황에서 가능한 모든 명령어를 호출하게 됩니다. 이경우는 **'NormalAt'**를 사용하여 **'normal_vector'**를 구하는 프로토콜을 호출하도록 합니다. 만일 어떤 명령어를 사용해야할지 모르는 경우에는 **Rhino Common SDK(Software Development Kit)/ http://www.rhino3d.com/5/rhinocommon/index.html** 를 참조하도록 합니다. 이 페이지에서 **Rhino Namespace / Geometry** 란으로 찾아들어가면 **Surface**에 해당하는 명령어 탭에서 다음을 찾아낼수 있습니다.

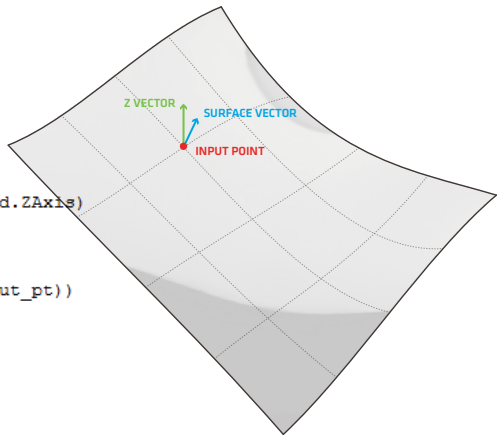
Surface.NormalAt (u As Double, v As Double) As Vector3d

'**NormalAt**'은 위에서 보듯이 **u**와 **v** 혹은 **(u,v)**라는 두개의 실수값을 요구합니다. 여기서 **(u,v)**는 면위의 좌표를 나타내는 지역좌표체계입니다. 아직은 **u**와 **v**가 어떤 값이며 어디서 얻을수 있는지 알수 없으므로 일단 **u**와 **v**라고 써 둡니다.

87 Dim u, v As Double

u와 **v**에 대해서 아는 것은 없지만, 정의에 따르면 이들은 실수임이 분명합니다. 그래서 일단 둘을 실수를 저장할수 있는 저장소로 정의내려 놓습니다.

```
85  Private Sub RunScript(ByVal base_srf As Surface, ByVal input_pt As Point3d, ByVal distance_factor As Double, ByRef A As Obj
86
87      Dim u, v As Double
88
89      base_srf.ClosestPoint(input_pt, u, v)
90
91      Dim normal_vector As Vector3d
92
93      normal_vector = base_srf.NormalAt(u, v)
94
95      Dim drain_vector As vector3d = vector3d.CrossProduct(normal_vector, vector3d.ZAxis)
96
97      drain_vector.Unitize
98
99      drain_vector.Transform(Transform.Rotation(Math.PI * 0.5, normal_vector, input_pt))
100
101     Dim moved_pt As point3d = input_pt + distance_factor * drain_vector
102
103     base_srf.ClosestPoint(moved_pt, u, v)
104
105     Dim output_pt As Point3d = base_srf.PointAt(u, v)
106
107     A = output_pt
108
109 End Sub
```

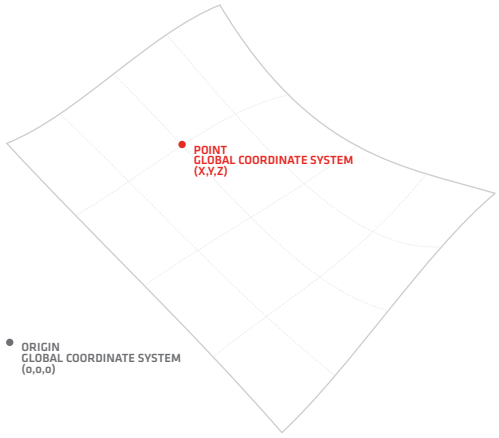


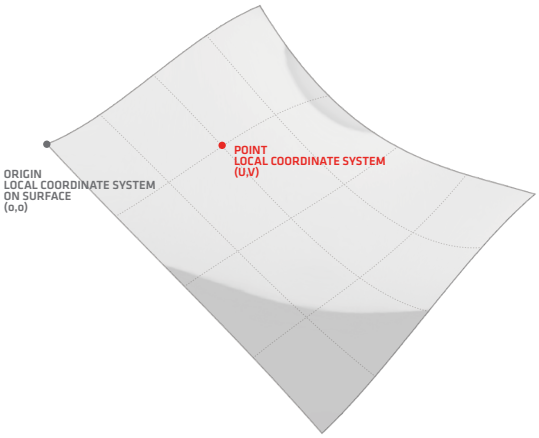
89 base_srf.ClosestPoint(input_pt, u, v)

이번에는 **(u,v)**를 얻을수 있는 방법에 대해 알아봅니다. 앞에서도 잠깐 언급했지만, **(u,v)**는 지역 좌표체계입니다. 우리가 **'input_point'**로 가지고 있는 점은 글로벌 좌표인 **(x,y,z)**로 이루어 져 있습니다. 그래스하퍼에 대해 경험이 있다면 외부의 점으로부터 가장 가까운 면위의 점을 찾는 과정 즉 **'closest point on surface'**를 통해 글로벌 좌표를 지역좌표(면위의 좌표)로 변환하는 방법에 대해 잘 알고 있을 겁니다. 외부의 점으로 부터 가장 가까운 면위의 점을 찾는 방법은 역시 면에 종속되어 있음이 분명합니다. 즉 **surface**라는 **class**에 종속된 **method**를 **Rhino Common SDK**에서 찾아보면 다음과 같은 정의를 찾게 됩니다.

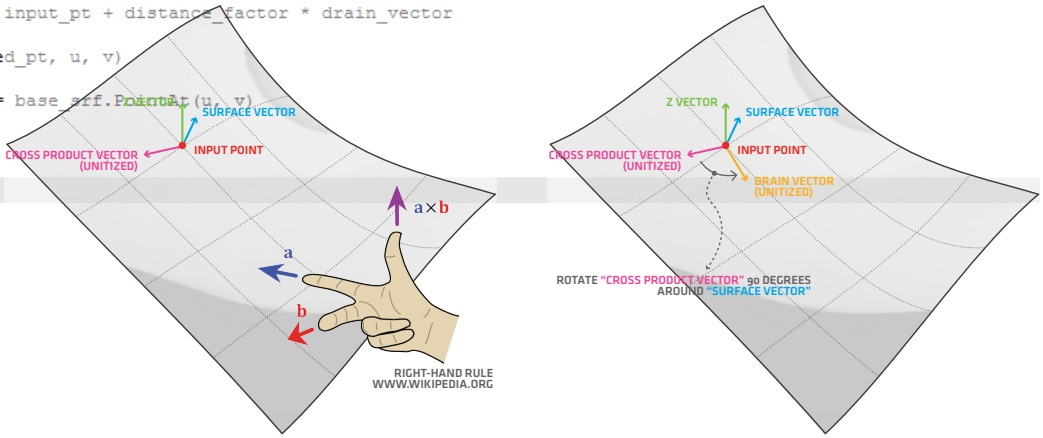
Surface.ClosestPoint (testPoint As Point3d, ByRef u As Double, ByRef v As Double)

위에서 보듯이, **test point**라는 점을 입력하면 그 점에서 가장 가까운 점의 면 상에서의 **(u,v)**좌표를 실수의 형태로 건내주게 됩니다. 전에 이미 **(u,v)**를 위한 저장소를 정의해 놓았으므로 우리가 구한 **(u,v)**값은 그 저장소, 즉 실수 **u**와**v**에 저장되게 됩니다. 이 좌표 값은 라인 93의 **(u,v)**에 입력이 되어서 결국 **'normal_vector'**값을 구할수 있게 됩니다.





```
85  Private Sub RunScript(ByVal base_srf As Surface, ByVal input_pt As Point3d, ByVal distance_factor As Double, ByRef A As Obj
86
87    Dim u, v As Double
88
89    base_srf.ClosestPoint(input_pt, u, v)
90
91    Dim normal_vector As Vector3d
92
93    normal_vector = base_srf.NormalAt(u, v)
94
95    Dim drain_vector As vector3d = vector3d.CrossProduct(normal_vector, vector3d.ZAxis)
96
97    drain_vector.Unitize
98
99    drain_vector.Transform(Transform.Rotation(Math.PI * 0.5, normal_vector, input_pt))
100
101   Dim moved_pt As point3d = input_pt + distance_factor * drain_vector
102
103   base_srf.ClosestPoint(moved_pt, u, v)
104
105   Dim output_pt As Point3d = base_srf.PointAt(u, v)
106
107   A = output_pt
108
109 End Sub
```



step 2 이번에는 앞에서 구한 '**normal_vector**'와 '**z_vector**'를 통해 '**drain_vector**'를 구해봅니다.

95 Dim drain_vector As vector3d = vector3d.CrossProduct(normal_vector, vector3d.ZAxis)

앞선 과정에서는 주어진 면위의 한점 **'input_point'**에서 **'normal_vector'**와 **'z_vector'**를 구하였습니다. 위의 그림에서처럼 주어진 두가지의 벡터를 잘 살펴보면 물 흐름 벡터인 '**drain_vector'**의 방향이 대강 예측이 됩니다. 이를 정확히 예측하기 위해 먼저 오른손 법칙을 사용하도록 합니다. 그림에서 처럼 오른손 법칙은 **'cross product vector'**의 방향을 예측하도록 도와 줍니다. 그후 **'cross product vector'**를 **'normal_vector'**를 축으로 반시계방향으로 90도를 돌려주면 물 흐름 방향의 벡터인 **'drain_vector'**를 얻을수 있게 됩니다. 우선 **'cross product vector'**를 구해보도록 합니다. 두 벡터간의 연산이지만 어느 한 벡터를 평가하여 연산을 하게 되는 특정벡터에 종속적인 연산이 아니므로 일반적인 벡터인 **vector3d**에 점을 찍어서 연산 **method**를 호출하도록 합니다. **SDK**에서 정의 되어지는 **'cross product'**는 다음과 같습니다.

Vector3d.CrossProduct (a As Vector3d, b As Vector3d) As Vector3d

이 method는 두개의 입력을 요구하는데요, 순서에 따라 두 벡터를 넣을시 세번째 벡터, 즉 연산의 결과는 오른손 법칙에 의해 나오게 됩니다. 비록 이 연산의 결과가 **'drain_vector'**는 아니지만, 우선 이 벡터를 **'drain_vector'**의 자리에 대입해봅니다.

97 drain_vector.Unitize

97번째 라인에서는 95라인에서 구한 외적벡터를 단위화 합니다. 이유는 벡터의 외적의 결과물은 연산이 되는 두 입력벡터의 길이가 일정하다고 하더라도 둘 사이의 각도에 의해 다른 크기의 벡터를 생성하므로 외적벡터의 길이가 얼마가 될지 예상할 수 없기때문입니다.

99 drain_vector.Transform(Transform.Rotation(Math.PI * 0.5, normal_vector, input_pt))

99번째 라인에서는 이 연산의 결과물을 **'normal_vector'**를 축으로 반시계방향 90도를 회전시키고 싶습니다.

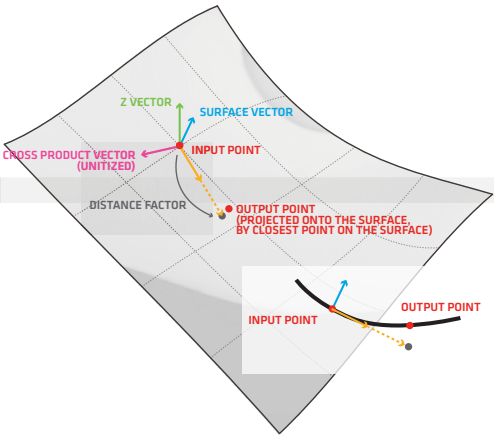
Vector3d.Transform (transformation As Transform)

벡터를 회전시키려면 **tranform**이라는 클래스상의 인스턴스를 입력해 주어야 합니다. **transform**또한 하나의 클래스이며 이는 이동, 회전, 크기조절등의 변환을 담당하는 클래스 입니다.

Transform.Rotation (angleRadians As Double, rotationAxis As Vector3d, rotationCenter As Point3d) As Transform

위에서 처럼 해당하는 변수들을 공급하면 드디어 **'drain_vector'**를 얻게 됩니다.

```
85  Private Sub RunScript(ByVal base_srf As Surface, ByVal input_pt As Point3d, ByVal distance_factor As Double, ByRef A As Obj
86
87    Dim u, v As Double
88
89    base_srf.ClosestPoint(input_pt, u, v)
90
91    Dim normal_vector As Vector3d
92
93    normal_vector = base_srf.NormalAt(u, v)
94
95    Dim drain_vector As vector3d = vector3d.CrossProduct(normal_vector, vector3d.ZAxis)
96
97    drain_vector.Unitize
98
99    drain_vector.Transform(Transform.Rotation(Math.PI * 0.5, normal_vector, input_pt))
100
101   Dim moved_pt As point3d = input_pt + distance_factor * drain_vector
102
103   base_srf.ClosestPoint(moved_pt, u, v)
104
105   Dim output_pt As Point3d = base_srf.PointAt(u, v)
106
107   A = output_pt
108
109 End Sub
```



step 3 이번에는 **'distance_factor'**를 이용해서 **'output_point'**를 구해봅니다.

101 Dim moved_pt As point3d = input_pt + distance_factor * drain_vector

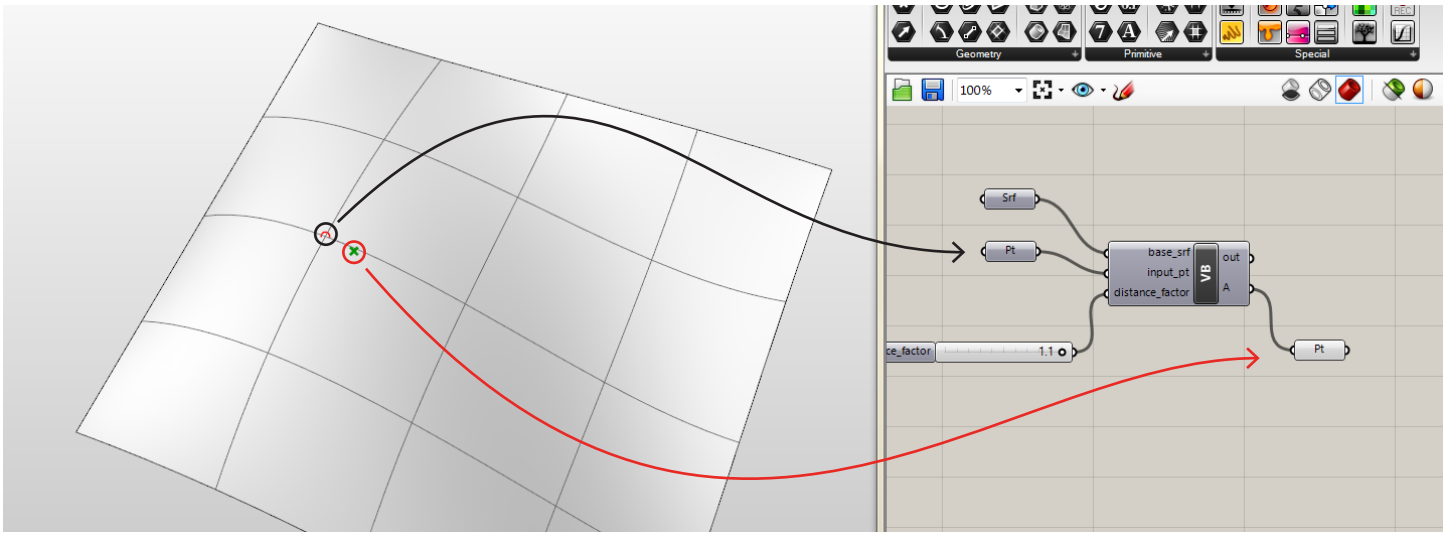
101라인은 아주 간단 명료 합니다. **'drain_vector'**를 이용해 **'distance_factor'**만큼 점을 이동시킵니다. 이 점을 임시로 **'moved_point'**라는 저장소에 저장합니다.

103 base_srf.ClosestPoint(moved_pt, u, v)

그후 위의 단면그림에서 볼수 있듯 면에서 점이 떨어지게 될 가능성이 있으므로 점을 다시 면에 잡아 당겨 붙입니다. 이 과정의 예전에 한번 하였듯이 **'find the closest point'**를 통해 해결 가능합니다. 즉 **'moved_point'**를 주어진 면의 **'closestPoint' method**에 공급하여 **u**와 **v**값을 구합니다.

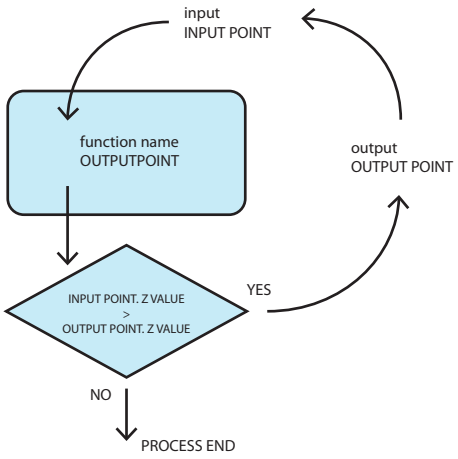
105 Dim output_pt As Point3d = base_srf.PointAt(u, v)
107 A = output_pt

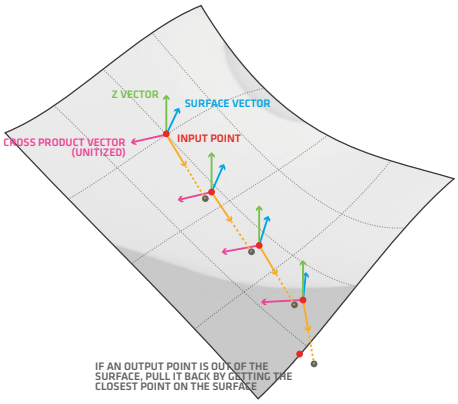
이렇게 구한 **u**와 **v**값을 105 라인에 넣어서 **'output_point'**를 찾고 이를 외부 출력단자인 **A**에 연결하여 그래스하퍼 화면으로 출력합니다.

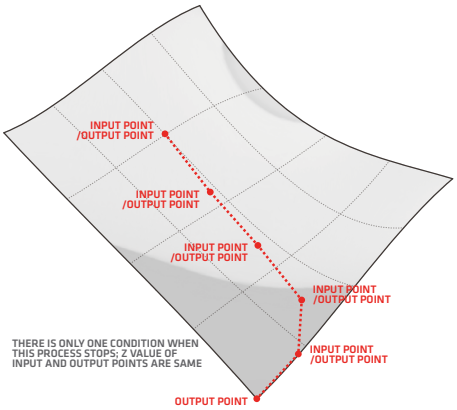


PART 2 물흐름선 찾기 (DEFINING A 'FLOW LINE')

Refer to 'VB workshop part2.gh' and 'VB workshop.3dm' attached.







step 1 첫번째 파트에서는 한 점을 입력하면 그 점에서의 물흐름 벡터를 계산하여 물 흐름의 다음점을 예상하여주는 시스템을 구축하였습니다. 두번째 파트에서는 이러한 시스템을 계속 반복하여 물 흐름의 방향을 예측하는 점들의 집합으로 부터 흐름선을 구하는 것을 살펴보도록 합니다.

```
85  Private Sub RunScript(ByVal base_srf As Surface, ByVal input_pt As Point3d, ByVal distance_factor As Double, ByRef A As Obj
86
87    Dim u, v As Double
88
89    base_srf.ClosestPoint(input_pt, u, v)
90
91    Dim normal_vector As Vector3d
92
93    normal_vector = base_srf.NormalAt(u, v)
94
95    Dim drain_vector As vector3d = vector3d.CrossProduct(normal_vector, vector3d.ZAxis)
96
97    drain_vector.Unitize
98
99    drain_vector.Transform(Transform.Rotation(Math.PI * 0.5, normal_vector, input_pt))
100
101   Dim moved_pt As point3d = input_pt + distance_factor * drain_vector
102
103   base_srf.ClosestPoint(moved_pt, u, v)
104
105   Dim output_pt As Point3d = base_srf.PointAt(u, v)
106
107   A = output_pt
108
109 End Sub
```

85에서 109 라인까지를 모두 복사하여서 아래 보이는 란에 붙여 넣습니다.

```
87    End Sub
88
89  '<Custom additional code>
90
91  '</Custom additional code>
92
93 End Class
```

```
117  /**/
118
119  Private Function outputpoint(ByVal base_srf As Surface, ByVal input_pt As Point3d, ByVal distance_factor As D
120                                                                                    ouble, ByRef A As Object) As Boolean
121    Dim u, v As Double
122
123    base_srf.ClosestPoint(input_pt, u, v)
124
125    Dim normal_vector As New Vector3d(base_srf.NormalAt(u, v))
126
127    Dim drain_vector As vector3d = vector3d.CrossProduct(normal_vector, vector3d.ZAxis)
128
129    drain_vector.Unitize
130
131    drain_vector.Transform(Transform.Rotation(Math.PI * 0.5, normal_vector, input_pt))
132
133    Dim moved_pt As point3d = input_pt + distance_factor * drain_vector
134
135    base_srf.ClosestPoint(moved_pt, u, v)
136
137    Dim output_pt As Point3d = base_srf.PointAt(u, v)
138
139    If output_pt.Z >= input_pt.Z Then
140
141      outputpoint = False
142
143    Else
144
145      outputpoint = True
146
147    End If
148
149    A = output_pt
150
151  End Function
152
153  /**/
```

복사하여 붙여 넣은 후에는 코드의 일부분을 바꿀 필요가 있습니다.

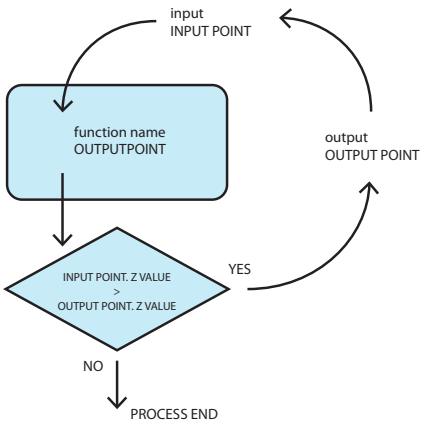
```
Private Sub RunScript(..., ..., ..., ByRef A As Object)
119  Private Function outputpoint(..., ..., ..., ByRef output_pt As Object) As Boolean
```

붉은 색이 방금 붙여 넣은 부분의 첫 줄이며 우리는 그것을 푸른색과 같이 바꿉니다.

Private 은 붙여 넣은 부분이 코드의 메인 부분, 즉 우리의 코드가 잘라내기 이전에 있었던 부분, 과 변수의 이름을 공유하지 않는다는 것을 의미합니다. 이부분을 **public** 으로 만들수도 있지만 이럴경우 변수명이 서로 겹치지 않게 신경의 써야 함으로 지금 단계에서는 그대로 **private** 을 유지합니다. **Runscript** 는 우리가 붙여 넣은 코드의 일부분을 일컷는 이름입니다. 우리는 조금더 의미가 있는 다른 이름을 정합니다. '**outputpoint**'.

또한 두번째의 **Sub**를 **Function**으로 바꿉니다. **Sub**와 **Function**은 둘다 코드의 메인 부분으로부터 떨어져 있는 일부의 코드를 부르는 방법인데 둘의 차이는 서브는 어떤 일을 하기만 할뿐 자체적으로 변수값을 지닐수 없으나 펑션은 일을 하기도 할 뿐더러 스스로 변수처럼 값을 가지기도 합니다. 우리는 아래의 다이어그램처럼 코드의 일부분이 유효성의 여부에 따라 불리언 값을 가져서 유효성 검사를 하는데 사용할수 있기를 바랍니다. 이 둘의 차이에 대해 혼돈이 있으시다면 다음 링크를 참조하세요. http://www.homeandlearn.co.uk/NET/vbNet.html

```
139  If output_pt.Z >= input_pt.Z Then
141      outputpoint = False
143  Else
145      outputpoint = True
147  End If
```



이부분은 유효성을 검사하는 부분입니다. 위에서 언급한 데로 서브대신 펑션을 사용하여서 **outputpoint**라는 펑션 자체가 **true** 또는 **false**의 값을 가질수 있게 되었읍니다. 이는 입력과 출력점의 높이를 비교하여 출력점의 높이가 입력점의 높이보다 높거나 같을 경우, 즉 물이 더이상 흐르지 못하는 경우 펑션값을 **false**로 설정함으로써 코드의 다른 부분에 더이상 이 프로세스를 돌리지 말고 중단할것을 전달합니다.

```
If (conditional statement) Then
    (do this)
Else
    (do that)
End If
```

'**if statements**'는 비쥬얼 베이직의 조건문의 하나로 다음 웹사이트에서 더 많은 정보를 찾을수 있습니다. http://www.home-andlearn.co.uk/NET/vbNet.html

```
80  /**/
85  Private Sub RunScript(ByVal base_srf As Surface, ByVal input_pt As Point3d, ByVal distance_factor As Double,
86
87    Dim pt As point3d = input_pt
88
89    Dim output_pts As New List(Of Point3d)
90
91    output_pts.Add(pt)
92
93    Dim output_pt As point3d
94
95    Do
96
97      outputpoint(base_srf, pt, distance_factor, output_pt)
98
99      output_pts.add(output_pt)
100
101     pt = output_pt
102
103     If output_pts.Count > 100 Then
104
105       Exit Do
106
107     End If
108
109   Loop While outputpoint(base_srf, pt, distance_factor, output_pt) = True
110
111   Dim output_crv As New PolylineCurve(output_pts)
112
113   A = output_crv
114
115 End Sub
```

step 2 전 단계에서는 '**output_point**'를 정의 했습니다. 이번 단계에서는 이 펑션을 값이 '**false**'가 될때까지 호출해 보겠습니다.

```
95    Do
109   Loop While outputpoint(base_srf, pt, distance_factor, output_pt) = True
```

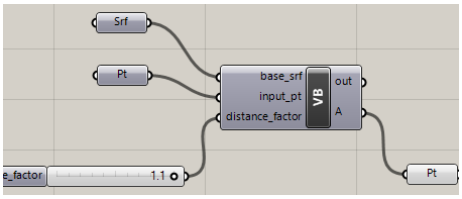
먼저, 이 펑션 '**outputpoint**'의 값이 '**True**'인 동안은 어떤 동작을 반복하는 반복문을 작성합니다.

```
Do
(do this)
Loop While (conditional statement)
```

이것이 비쥬얼 베이직의 '**Do ~ Loop**'문 입니다. '**while ()**'부분은 반복의 조건을 말해주는 조건절로써 'Do' 혹은 'Loop'뒤에 올 수 있습니다. 또한 펑션 '**outputpoint**'을 호출하는 방법을 잘 살펴보면 **methods**를 불러내는 방식과 비슷하다는 것을 알수 있습니다.

```
97    outputpoint(base_srf, pt, distance_factor, output_pt)
```

97번째 라인에서는 드디어 펑션을 호출합니다. 주지하다시피, 세개의 입력값과 한개의 출력값이 있습니다. 우리는 두개의 입력값에 대해서는 확실하게 알고 있습니다. 즉, '**base_surface**'와 '**distance_factor**'는 이미 정해진 입력값이므로 그대로 입력합니다. 하지만, 연속적인 반복의 특성상 입력점과 출력점은 항시 변하게 됩니다. 즉, 입력점과 출력점은 프로세스를 반복함에 따라 서로의 위치를 바꾸게 되며 특히나 출력점의 경우는 단일 점이 아니라 여러개의 점을 지니게 되며 이는 점의 리스트 형식으로 표현이 되게 됩니다. 그래서 우리는 여러개의 출력점을 저장할수 있는 저장소가 필요하게 됩니다.



```
80    /**/
85    Private Sub RunScript(ByVal base_srf As Surface, ByVal input_pt As Point3d, ByVal distance_factor As Double,
86
87      Dim pt As point3d = input_pt
88
89      Dim output_pts As New List(Of Point3d)
90
91      output_pts.Add(pt)
92
93      Dim output_pt As point3d
94
95      Do
96
97        outputpoint(base_srf, pt, distance_factor, output_pt)
98
99        output_pts.add(output_pt)
100
101       pt = output_pt
102
103       If output_pts.Count > 100 Then
104
105         Exit Do
106
107       End If
108
109     Loop While outputpoint(base_srf, pt, distance_factor, output_pt) = True
110
111     Dim output_crv As New PolylineCurve(output_pts)
112
113     A = output_crv
114
115   End Sub
```

87 Dim pt As point3d = input_pt
93 Dim output_pt As point3d
89 Dim output_pts As New List(Of Point3d)
91 output_pts.Add(pt)

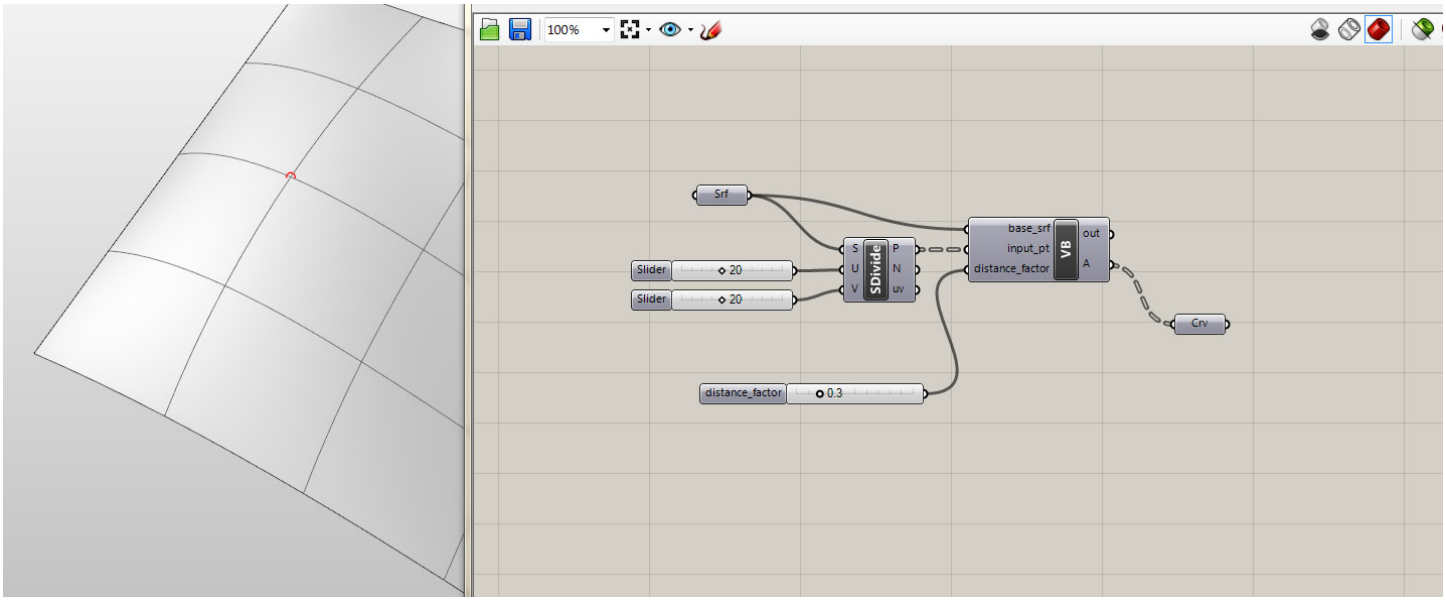
87번째 라인에서는 입력점의 휘발적인 성격상, 입력점을 임시로 저장할수 있는 공간 '**pt**' 를 만들어 준후 가장 먼저의 입력점을 그 저장소에 보관합니다. 93번째 라인에서는 출력점을 임시로 보관할수 있는 저장 공간 '**output_point**'을 만들어 줍니다. 89번째 라인에서는 '**output_points**'라고 불리우는 출력점을 여러개 저장할수 있는 리스트 공간을 만들어 줍니다. 91번째 라인에서 아무것도 가공하지 않은 입력점을 바로 그 리스트에 저장하는 이유는 그렇지 않으면 우리가 궁극적으로 얻고자 하는 곡선이 첫번째 프로세스의 출력점으로 부터 시작하기 때문입니다.

```
95    Do
97      outputpoint(base_srf, pt, distance_factor, output_pt)
99      output_pts.add(output_pt)
101     pt = output_pt
103     If output_pts.Count > 100 Then
105       Exit Do
107     End If
109   Loop While outputpoint(base_srf, pt, distance_factor, output_pt) = True
```

99번째 라인에서는 97번째의 펑션 호출로 인해 발생한 output_point의 값을 출력점 리스트에 입력합니다. 이때의 점은 91번째 라인에서 저장한 점에 이어 두번째 점이 될겁니다. 101번째 라인은 이번 회의 출력값을 다음회의 입력값으로 전환하는 과정입니다. 103에서 107번째 라인은 만약 이 리스트의 점의 숫자가 100개 이상을 넘을시 프로세스를 멈추라는 것을 말해 주는 과정입니다. 아주 좋은 컴퓨터를 가지신 분들은 아마 상관이 없겠지만 보통의 경우는 어느정도 이상의 연산 부하가 걸릴시 컴퓨터가 작동을 하지 않을 수 있으며 이 부분의 코드는 이를 방지하기 위함입니다.

111 Dim output_crv As New PolylineCurve(output_pts)
113 A = output_crv

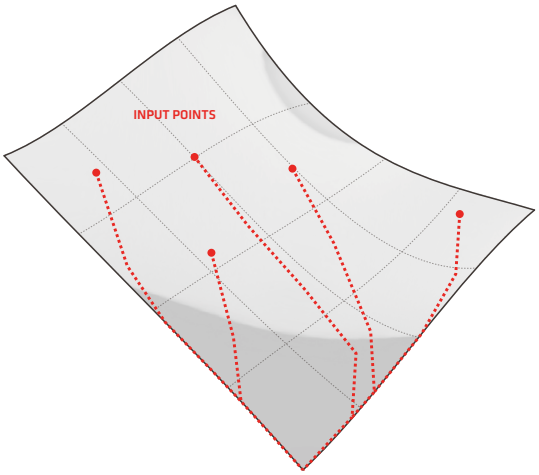
이 부분에서는 얻어진 점의 리스트로 부터 '**output_points**'를 만들고 이를 출력 탭인 **A**에 연결하는 과정입니다.

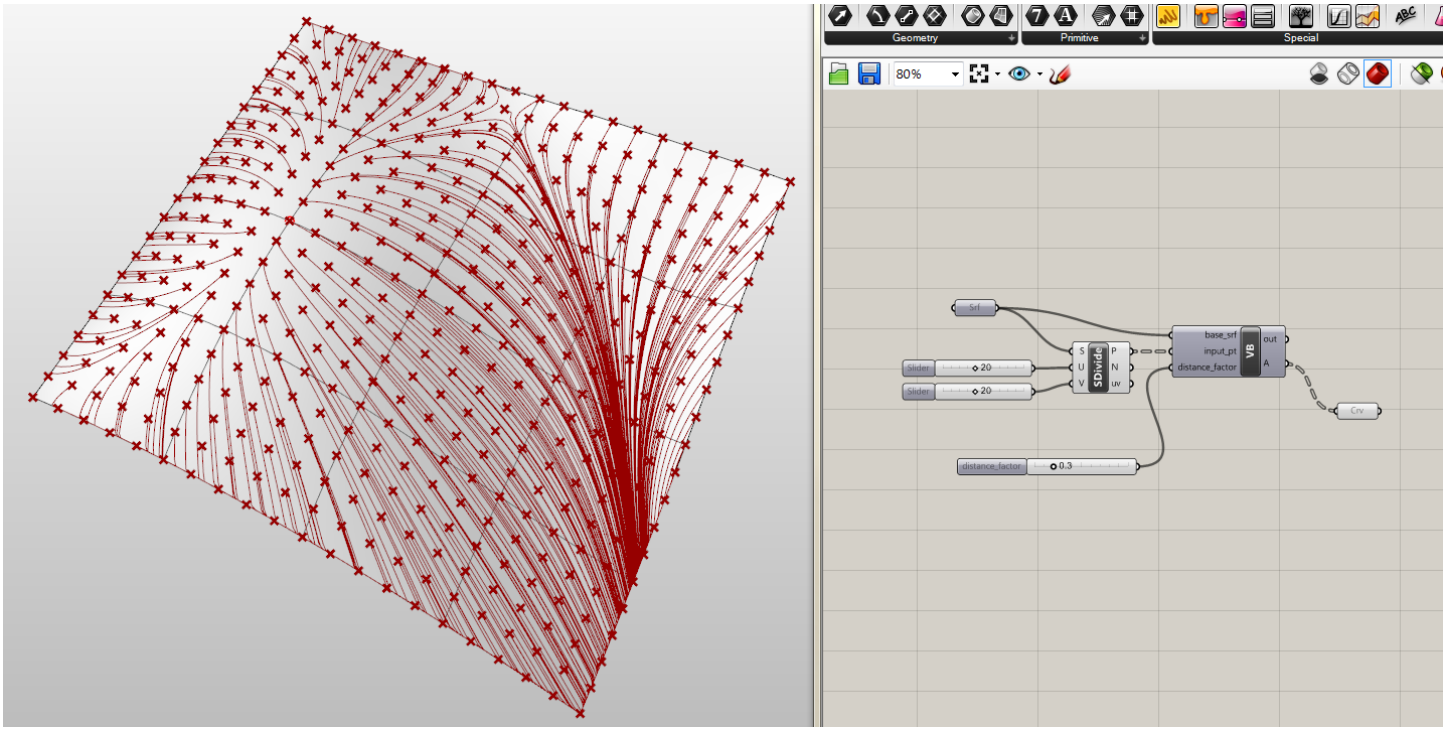


PART 3 복수의 물흐름선 찾기 (APPLYING TO MULTIPLE WATER SOURCES)

Refer to 'VB workshop part3.gh' and 'VB workshop.3dm' attached.

이제 마지막 파트에서는 복수의 수원에 위의 프로세스를 적용해 봅니다.





```
80  /**/
85  Private Sub RunScript(ByVal base_srf As Surface, ByVal input_pt As List(Of Point3d), ByVal distance_factor As
86
87    Dim output_crvs As New List(Of PolylineCurve)
88
89    For Each pt As point3d In input_pt
90
91      Dim output_pts As New List(Of Point3d)
92
93      output_pts.Add(pt)
94
95      Dim output_pt As point3d
96
97      Do
98
99        outputpoint(base_srf, pt, distance_factor, output_pt)
100
101       output_pts.add(output_pt)
102
103       pt = output_pt
104
105       If output_pts.Count > 100 Then
106
107         Exit Do
108
109       End If
110
111     Loop While outputpoint(base_srf, pt, distance_factor, output_pt) = True
112
113     Dim output_crv As New PolylineCurve(output_pts)
114
115     output_crvs.Add(output_crv)
116
117   Next
118
119   A = output_crvs
120
121 End Sub
```

step 1 복수의 점을 입력점으로 사용하기 위한 반복과정을 설정하는 과정입니다.

87 **Dim output_crvs As New List(Of PolylineCurve)**

모든 연산의 마지막에 결국 얻고자 하는 것은 다수의 곡선이며 이를 저장하는 곡선의 리스트가 저장소로 필요하게 됩니다.

89 **For Each pt As point3d In input_pt**
....
115 **output_crvs.Add(output_crv)**
117 **Next**

위는 또다른 비주얼베이직의 순환문인 '**for ~ next**'입니다. 이는 앞에서 본 '**do ~ Loop**'과는 달리 조건문이 없습니다. 단순히 입력으로 정한 리스트 안의 모든 객체들에 같은 동작을 반복시키게 됩니다. 이렇게 얻어진 하나하나의 커브들을 '**output_curve**'에 저장함으로써 본 정의의 마지막 과정을 마칩니다.