3. Наборы данных и математика

В 3D приложениях мы можем выбрать наши геометрические объекты из меню и нарисовать их в явном виде, с помощью мыши, не думая о математических аспектах. Однако, для того, чтобы работать с генерирующими алгоритмами, нам придется думать о данных и о математике, чтобы задать исходные условия алгоритма создания нескольких объектов. Так как мы не хотим рисовать все вручную, нам потребуются некоторые источники данных, как основные компоненты, чтобы сделать возможным создание и использование алгоритма работы более одного раза и получение в результате более чем одного объекта.

То, каким образом работает алгоритм (Workflow) достаточно просто. Он включает в себя исходные данные, процедуру обработки и вывод данных. Этот процесс происходит в алгоритме в целом или, если присмотреться, в каждой его части. Таким образом, вместо обычного метода рисования каждого объекта, мы задаем информацию, эта информация будет обрабатываться с помощью алгоритма, который генерирует результирующую геометрию. Как мы уже говорили, например, вместо копирования объекта, нажав 100 раз на экране, мы можем сказать алгоритму, копировать элемент "100 раз" в "положительном направлении X" с расстоянием "3" между ними. Для этого необходимо определить "100" как число копий, " положительное направление X", как направление копирования и "3", как расстояние между ними. И алгоритм выполнит работу за вас автоматически.

Все, что мы делаем в геометрии, требует некоторых математических расчетов. Мы можем использовать эти простые математические функции в наших алгоритмах с числами и объектами, для получения бесконечных геометрических комбинаций. Начнем с чисел и числовых наборов данных. Давайте посмотрим, это проще, чем кажется!

3.1. Числовые наборы данных

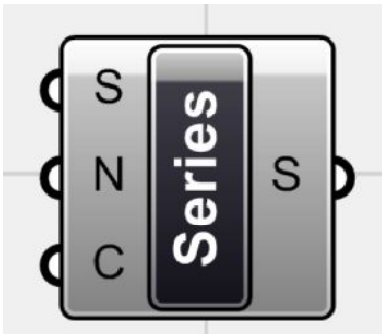
Вся математика и алгоритмы начинаются с чисел. Числа - скрытые коды Вселенной. Для начала, мы быстро пробежимся по числовым компонентам, чтобы увидеть, как мы можем генерировать различные числовые множества данных в Grasshopper и то, как мы можем использовать их для дизайна.

Одно числовое значение

Наиболее используемый генератор чисел это компонент <Number slider> (Params > Special > Number slider), который генерирует одно число, настраиваемое вручную. Оно может быть: целым, вещественным, четным, нечетным и с ограниченными верхним и нижним значениями. Вы можете задать все это, выбрав пункт ‘Edit’ контекстного меню компонента.

Для задания одного фиксированного числового значения в можете использовать компоненты Integer / Number в (Params > Primitive) для выбора соответственно целого или вещественного чисел. Задается значение через контекстного меню компонентов <Int>/<Num>.

Числовой ряд (Series)



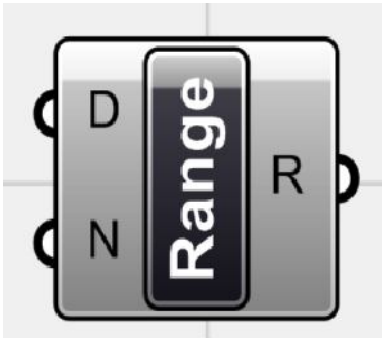
Компонент <series> (Logic > Sets > Series) генерирует список дискретных чисел. Этот компонент создает числовой ряд, для которого можно задать первое число, шаг приращения и количество значений.

0, 1, 2, 3, … , 100

0, 2, 4, 6, … , 100

10, 20, 30, 40, … , 1000000

Диапазон чисел (Range)



Вы можете разделить числовой диапазон между нижним и верхним значениями, равномерно распределенными числами и получить, таким образом, диапазон чисел. Для этого необходимо определить интервал, задав его нижний и верхний пределы, а также число шагов между ними (Logic > Sets > Range).

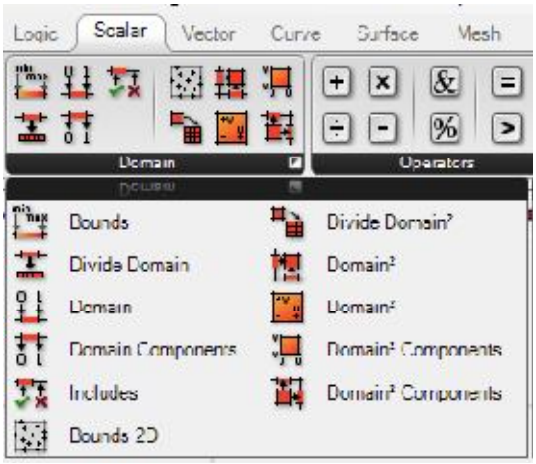
Любой числовой интервал (например, от 1 до 10) может быть разделен на бесконечное число частей:

1, 2, 3, … , 10

1, 2.5, 5, … , 10

1, 5, 10

Области определения (Интервалы) (Domains)



Области определения ("интервалы" в предыдущих версиях) задают диапазон всех действительных чисел между нижним и верхним пределами. Есть одномерная и двумерная области, мы поговорим о них позже. Мы можем определить фиксированную область с помощью компонента Params > Primitive > Domain/Domain^2 или мы можем перейти к Scalar > Domain, который предоставляет набор компонентов для работы с ними более гибкими путями.

Домены сами по себе не дают на выходе числа. Они просто задают границы, с верхним и нижним пределами. Как вы знаете, имеется бесконечное количество действительных чисел между любыми двумя действительными числами. Мы будем использовать различные функции, чтобы разделить их и делящие факторы, как числовые значения.

Чтобы увидеть разницу в использовании компонентов, позвольте перейти к некоторым примерам.

3.2. Точки и сети точек

Точки являются основным базовым элементом, как при ручном построении геометрии, так и при использовании генерирующих алгоритмов. Поскольку точки обозначают (маркируют) определенные позиции в пространстве, то они могут выполнять роль начальных точек кривой, центра окружности, начала координат плоскости, а также выполнять множество других ролей. Создавать точки в Grasshopper можно различными путями:

- Вы можете просто выбрать точку / несколько точек из сцены Rhino и передать их в Grasshopper с помощью компонента <point> (Params > Geometry > point) и использовать их затем для любых целей (Эти точки в дальнейшем могут настраиваться и перемещаться вручную в сцене Rhino, оказывая влияние на ваш проект). См. примеры в главе 2.

- Вы можете задать точки с помощью компонента <point xyz> (vector > point > point xyz) и регулировать координаты таких точек с помощью чисел. Или регулировать их различными наборами данных, теми которые нам требуются.

- Вы можете создать сеть точек компонентами <grid hexagonal> и <grid rectangular> (шестиугольная и прямоугольная сети).

- Вы можете извлечь точки из других геометрических объектов множеством различных способов. Например, конечные точки, средняя точка, точки пересечения и т.д.

- Иногда вы можете использовать плоскости (начало координат) и векторы (точку приложения вектора) как точки, чтобы создавать другие геометрические объекты, и наоборот.

Обратимся к первому очень простому примеру создания точек из Главы 2, и посмотрим, как можно создавать точки и наборы точек используя компоненты <series>, <range> и <number slider> и другие источники числовых данных.

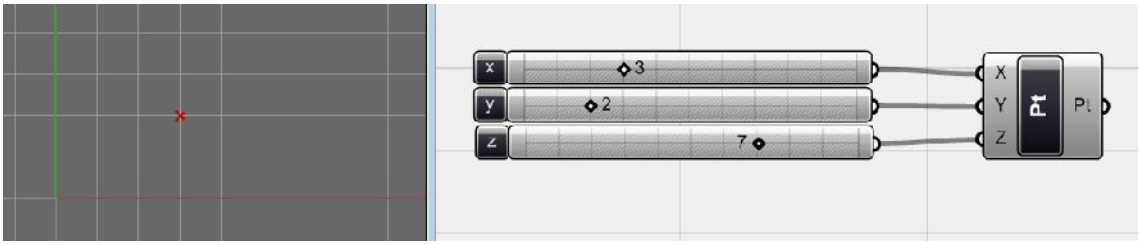


Рис. 15. Регулирование компонента <point xyz> или <pt> тремя <number slider> для создания точки, по координатам X,Y и Z, задаваемым вручную.

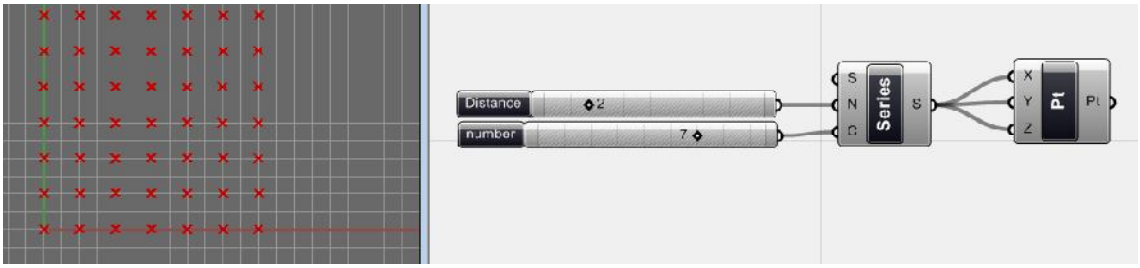


Рис. 16. Генерация сети точек с помощью компонентов <series> и <pt>, где первый управляющий элемент <number slider> задает расстояние между точками (step size), а второй отвечает за количество точек в сетке, контролируя параметр 'number of values' (количество значений) в компоненте <series> (Для создания сети точек, параметр сопоставления данных в компоненте <pt> должен быть установлен в положение 'cross reference', однако вы можете попробовать и другие варианты).

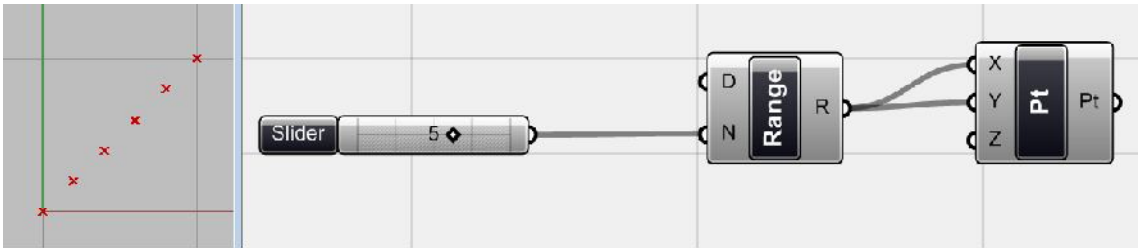


Рис. 17. Разбиение числового диапазона от 0 до 1, контролируемым вручную числом (5), и использование этих чисел в компоненте <pt>, с включенным режимом сопоставления данных 'Longest list'. Здесь мы разделили диапазон на 5 частей, получив таким образом, 6 точек и все эти точки отображаются между начальной (0,0) и точкой с координатами (1, 1) в рабочем окне Rhino (вы можете изменить верхнюю и нижнюю границы компонента <range> для изменения координат точек. Для этого нажмите правой кнопкой мыши на входе D (domain) компонента и измените область определения. Другие способы работы с интервалами и их изменение мы рассмотрим далее).

3.3. Другие числовые множества

Наборы случайных данных

Допустим, мы хотим создать случайно распределенный набор точек для последующего использования. Все что для этого нужно, это только задать случайные числа вместо компонента <series> для управления <pt>. Поэтому выберем компонент <random> из Logic > sets. Компонент <random> создает список случайных чисел. Вы можете контролировать количество этих чисел и диапазон, которому они будут принадлежать. Но компонент <random> создает только один набор случайных чисел, а мы не хотим иметь одинаковые значения всех координат X,Y и Z. Для того чтобы избавиться от одинаковых значений, нам нужны различные случайные числа для каждой из координат. Мы можем получить эти три списка случайных координат, задав три различных компонента <random> с разными значениями параметра seed (задавая входам (S) компонентов <random> разные числа, получим различающиеся случайные значения, в противном случае все компоненты <random> будут генерировать одинаковые начальные значения) или перемешать имеющиеся списки значений.

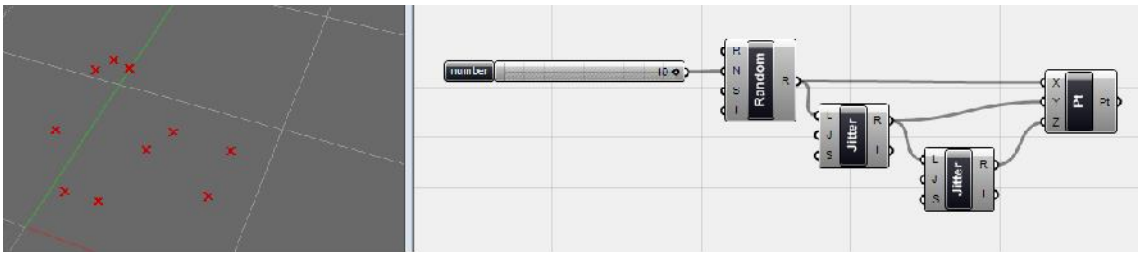


Рис. 18. Генерация случайного набора точек. Компонент <random> создает 10 случайных чисел, что контролируется компонентом <number slider>. Затем этот список перемешивается с помощью <jitter> (Logic > Sets > Jitter). Первый раз для координаты Y, а затем еще раз для координаты Z. В противном случае вы можете заметить некоторую повторяемость (паттерн) координат внутри сетки (подключите компонент <random> к X, Y и Z компонента <pt> без <jitter> и убедитесь в этом!). Параметр сопоставление данных должен быть 'longest list'.

В предыдущем примере все точки распределены в координатном пространстве между значениями 0 и 1 в каждом направлении. Для изменения области распределения точек нужно сменить числовой интервал, в котором <random> генерирует числа. Это возможно ручной установкой "domain of random numeric range" в командной строке Rhino, если нажать правой кнопкой мыши на входе (R) (random numbers domain) компонента. Или заданием области определения с помощью компонента <domain>, границы которого можно регулировать с помощью слайдеров <number slider>. См. рис.

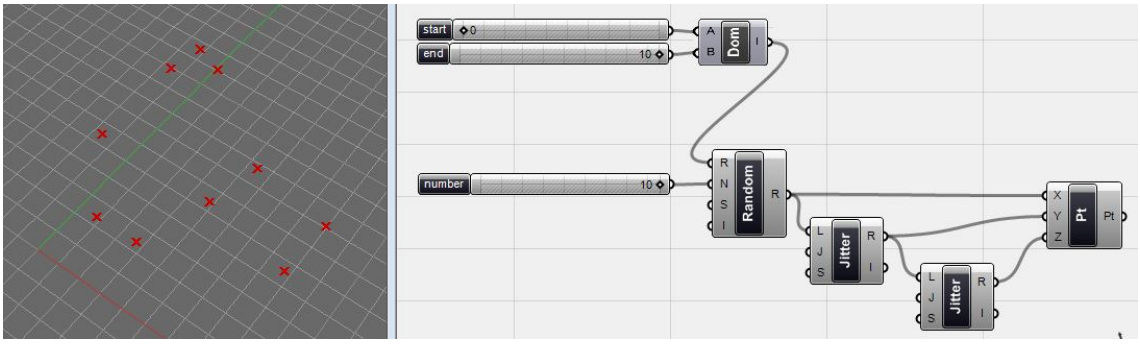


Рис. 19. Настройка области определения с помощь компонента <domain> (Scalar > Domain > Domain) для изменения области расположения точек (обратите внимание на плотность линий сетки в сцене предыдущего примера).

Числа Фибоначчи

А что если нужно создать сетку точек, но не с постоянным расстоянием между ними, а с увеличивающимися значениями? Давайте посмотрим список доступных компонентов. Нам нужна последовательность чисел, которые быстро возрастают и на панели Sets вкладки Logic мы может отыскать компонент <Fibonacci>.

Фибоначчи – это последовательность чисел, в которой первые два числа заданы и равны 0 и 1. А все последующие числа вычисляются как сумма двух предыдущих чисел.

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)

Вот некоторые числа этого ряда: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

Как вы видите, числа быстро возрастают. Здесь мы используем последовательность <Fibonacci> (Logic > Sets > Fibonacci) для получения возрастающих чисел, чтобы управлять с помощью них компонентом <pt>.

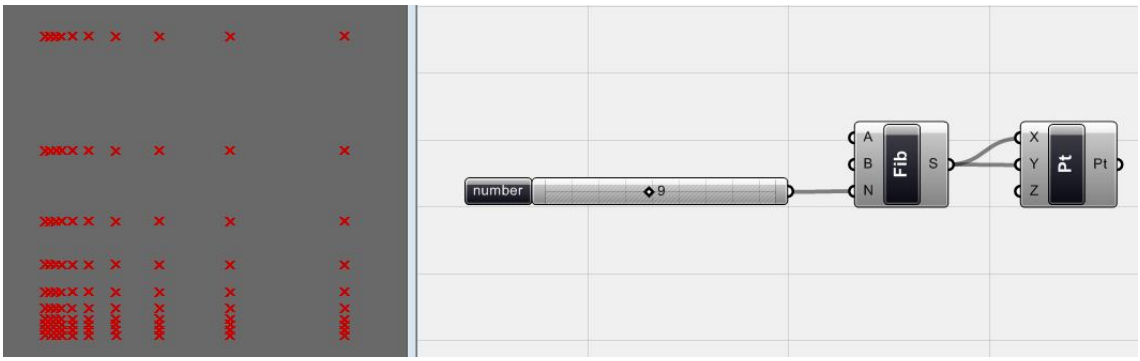


Рис. 20. Использование последовательности <Fibonacci> для получения увеличивающихся расстояний (не равномерно распределенного ряда чисел) при генерации точек. Количество точек контролируется с помощью <number slider>.

3.4. Функции

Предопределенные компоненты в Grasshopper возможно не всегда лучший путь для создания вашего дизайна. Вам может потребоваться создавать ваши собственные наборы данных или манипулировать данными существующих компонентов. Для этого вам необходимо

использовать математические функции и изменять степень, расстояние, … чисел. Функции (Functions) это компоненты, которые предназначены для выполнения математических функций в Grasshopper. Функции могут быть с разным числом переменных (Logic > script). Вам необходимо снабдить компонент-функцию соответствующими данными (не всегда числовыми, но также булевыми или строковыми) и она применит заданную пользователем формулу к исходным данным. Для задания формулы нужно нажать правой кнопкой мыши на входе (F) компонента и ввести ее в поле Expression Editor (Редактор выражений). Редактор выражений обладает большим перечнем предопределенных выражений и библиотекой математических функций для справки.

Обратите внимание, что имена переменных, которые вы используете в ваших выражениях и связанные с ними данные должны совпадать с названием входов компонента!

Математические функции

Как было сказано ранее, использование предопределенных компонентов, не всегда дает возможность получит то, что мы хотим. Но для получения требуемого результата мы можем использовать математические функции, чтобы изменять наборы данных и использовать их для создания геометрии. Простой пример математической функции, это окружность, координаты которой находятся с помощью выражений: X=Sin(t) и Y=Cos(t), где (t) – это диапазон чисел от 0 до 2*Pi. Мы можем получить его с помощью <range>, который создает N чисел в диапазоне от 0 до 1, умножив его с помощью компонента <function> на 2*Pi. В итоге получим результирующий диапазон чисел от 0 до 2*Pi, что соответствует параметру t в радианах для полной окружности.

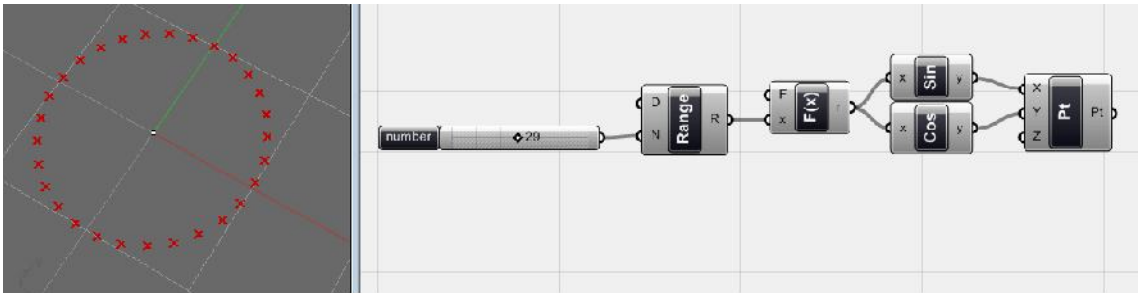


*Рис. 21. Параметрическая окружность с использованием математических функций. Использованы функции <Sin> и <Cos> (Scalar > Trig). (F(x) = x * 2Pi).*

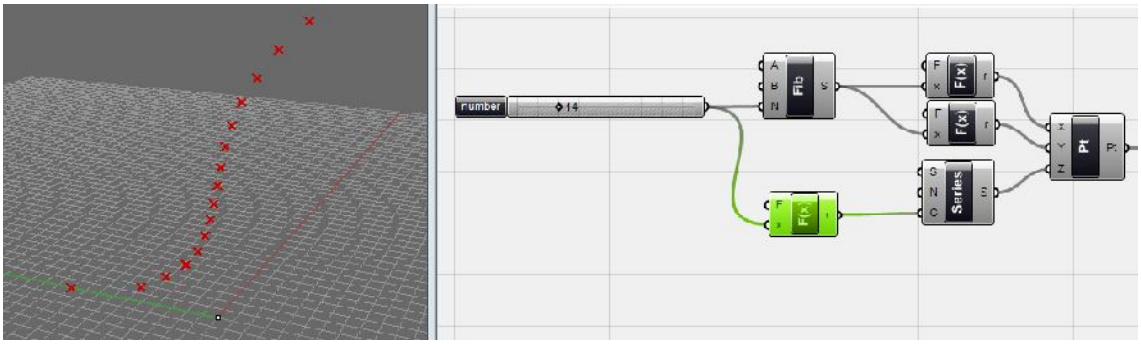


Рис. 22. Ряд точек, созданных с помощью <Fibonacci> и простые математические функции (x: F(x)=x/100, y: F(x)=x/10).Выбранный зеленый компонент F(x) – это функция, которая добавляет 2 к значению <number slider> (F(x)=x+2) для того чтобы сделать количество

значений <series> равным количеству чисел Фибоначчи (Фибоначчи имеет два предопределенных начальных числа). Мыпопытались показать вам, как просто вы можете манипулировать этими наборами данных для создания различной геометрии.

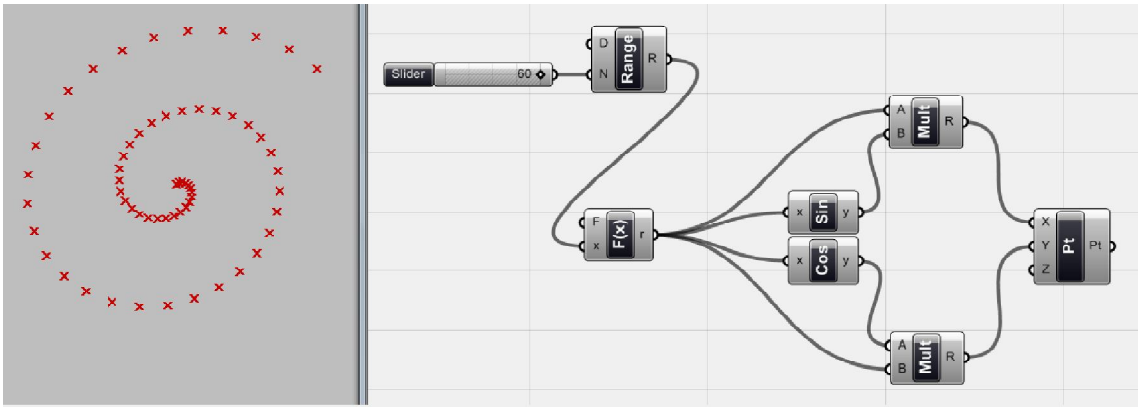


*Рис. 23. Диапазон чисел <range> от 0 до 2, который умножен на 2*Pi с помощью <Function>, задает числовой диапазон от 0 до 4*Pi. Этот диапазон делится на 60 частей. Результат передается компоненту <pt> с использованием следующих математических функций:*

*X=t * Sin(t), Y=t * Cos(t).*

*Компоненты <sin> и <cos> нам уже известны. Для задания выражения t*sin/cos используется компонент <multiplication> (умножение)из Scalar>Operators.*

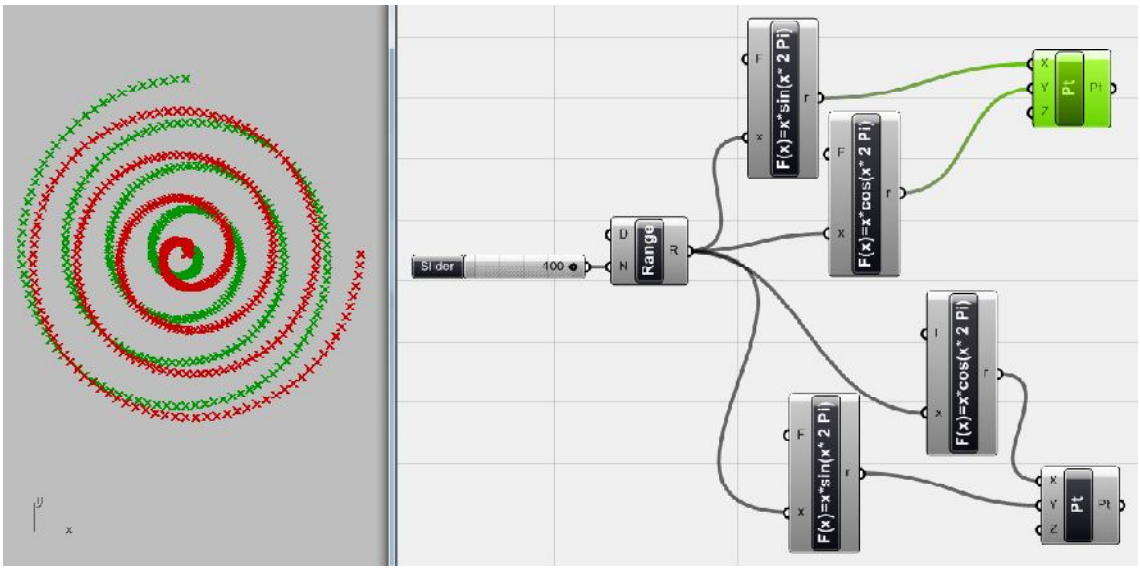


Рис. 24. Усложним задачу! Пересекающиеся касательные кривые из двух инвертированных наборов точек, расположенных по спирали. Компонент <range> задает диапазон от 0 до 4, который делится на 400 точек, и они умножаются на функции:

*Первый <pt>: X: F(x) = x * Sin(x*2 Pi), Y: F(x) = x * Cos(x * 2 Pi).*

Второй <pt> использует те же самые функции, только инвертированные.

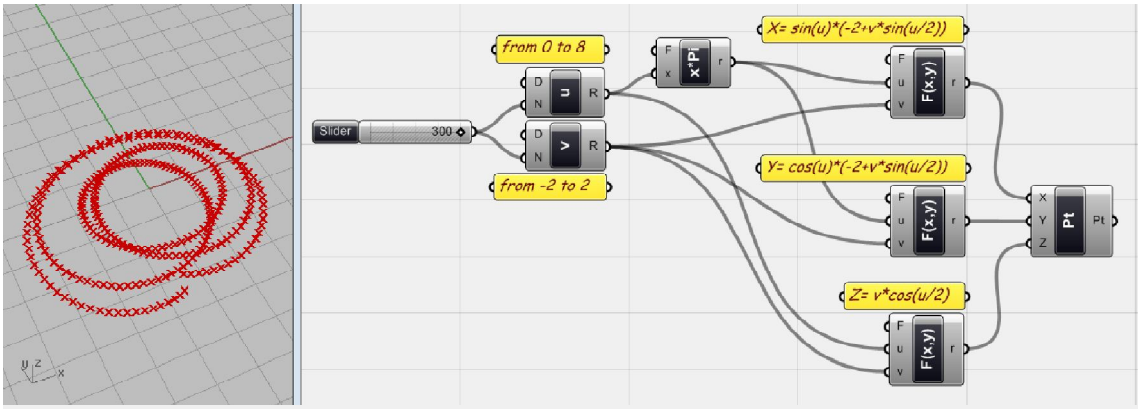


Рис. 25. Точки поверхности Мёбиуса. <u> и <v> это компоненты <range>, которые переименованы. Область определения каждого диапазона показана на рисунке. Математические функции для генерации точек поверхности Мёбиуса:

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

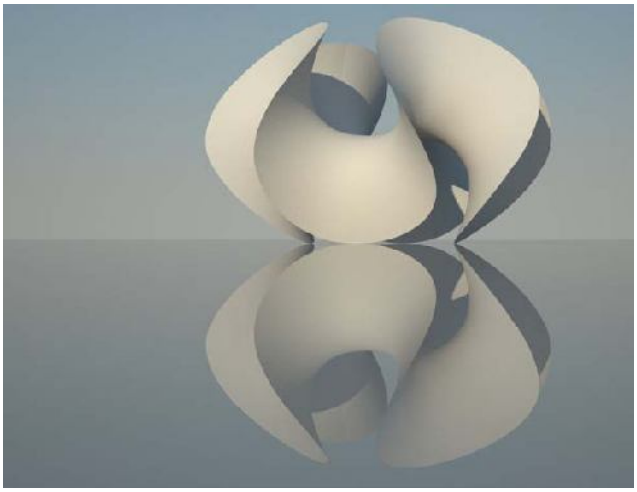
Y= Cos(u)(-2+v*sin(u/2))*

*Z= v*Cos(u/2)*

*(Для u от 0 до 8*Pi и v от -2 до 2, которые создаются компонентами <function>. Все созданные координаты передаются компоненту <pt>).*

Эксперименты с математическими функциями можно продолжать до бесконечности. Вы можете найти множество математически ресурсов (книг, сайтов) для манипуляции вашими наборами данных с помощью них. Важный момент состоит в том, что вы можете манипулировать исходными наборами данных и генерировать различные числовые значения, а затем передавать их другим компонентам.

Как вы убедились на простых наборах числовых данных, вы можете начать создавать геометрические объект и теперь знаете как работают эти алгоритмы. С данного момента, мы будем создавать проекты основываясь на наших знаниях различных геометрических концепций и алгоритмическом подходе к дизайну, подобно тому как, как эта очень красивая поверхность Эннепера (создано с помощью плагина Math for Rhino):



3.5. Булевы типы данных

Данные не ограничиваются числами. Существуют и другие типы данных, которые полезны для различных целей в области программирования и алгоритмов. Поскольку мы имеем дело с алгоритмами, мы должны знать, что последовательность инструкций алгоритма не всегда линейна. Иногда мы хотим, чтобы он решал, следует сделать что-то или нет. Программисты называют это условными операторами. Нам нужно знать, соответствует ли какое-то выражение определенным критериям или нет. Ответом на логическое выражение (вопрос) является просто да или нет. В алгоритмах мы используем булевы данные для представления этих ответов. Тип данных Boolean имеет лишь два возможных значения: True (Да) или False (нет). Если выражение отвечает критериям, ответ **True**, иначе **False**. Как вы увидите позже, этот тип данных является очень полезным в различных случаях: когда вы хотите принять решение о чем-то, выбрать несколько объектов по определенным критериям, отсортировать объекты и т.д.

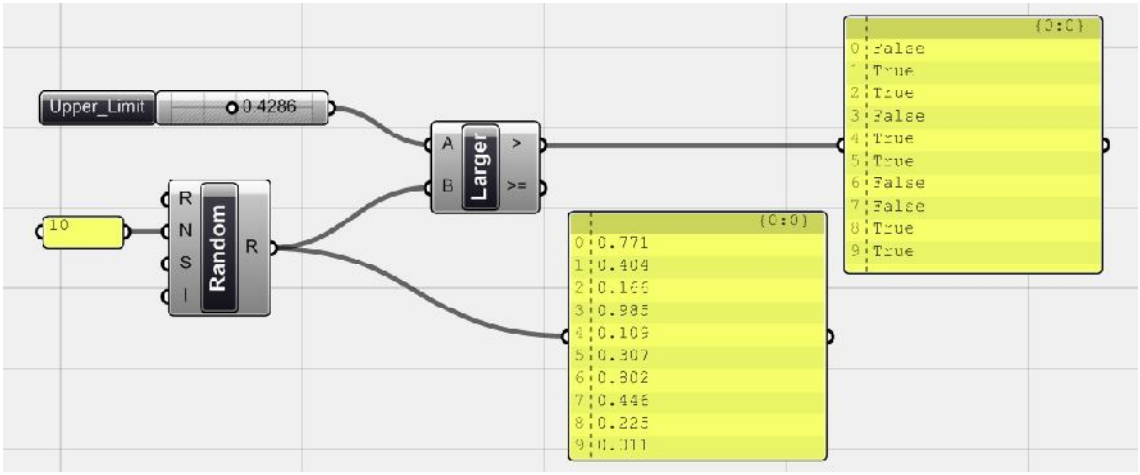


Рис. 26. Здесь создается десять случайных значений с помощью <random>, а с помощью компонента <Larger> (Scalar>Operators) мы можем увидеть меньше эти числа определенного значения <Upper_limit> (любое значение, задаваемое <number slider>) или нет. Как вы видите, когда числа отвечают критерию (означает, что они меньше, чем <Upper_limit>), то компонент <Larger> выдает в качестве результата 'True', в противном случае 'False'. Здесь использован компонент <Panel> (Params>Special) чтобы показать содержимое <Random> и результат компонента <Larger>.

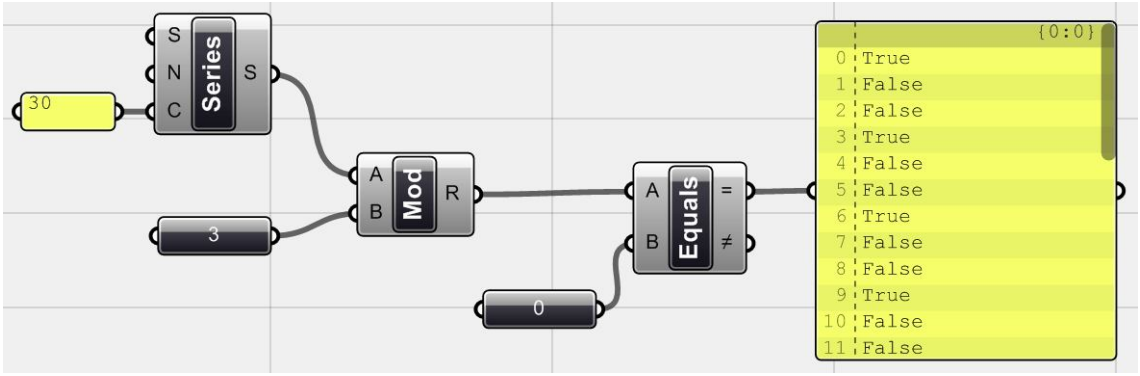


Рис. 27. Для данного примера сгенерировано 30 значений с помощью компонента <series> и использован компонент <Modulus> (Модуль) (Scalar > Operators > Modulus), чтобы найти остаток от деления числовых значений на <3>. Затем этот результат передается

компоненту <Equals>, чтобы узнать есть ли остатки равные нулю или нет. Результат операции можно видеть на <panel> в виде значений True/False.

Так как видно из этих примеров, существуют различные возможности для проверки критериев числовые значений и получения логических значений в качестве результата. Но иногда, требуется проверить одновременно несколько критериев, и на основе этого сделать какое-то заключение. Например, в описанных выше примерах мы хотим видеть, является ли значение меньше, чем заданное upper_limit и в то же время, делится ли оно на 3. Чтобы узнать итог, мы должны оперировать результатами обеих функций, т.е. мы должны работать с булевыми значениями. Если вы проверите, на вкладке Logic в панели Boolean существуют различные компоненты, которые работают с логическим типом данных.

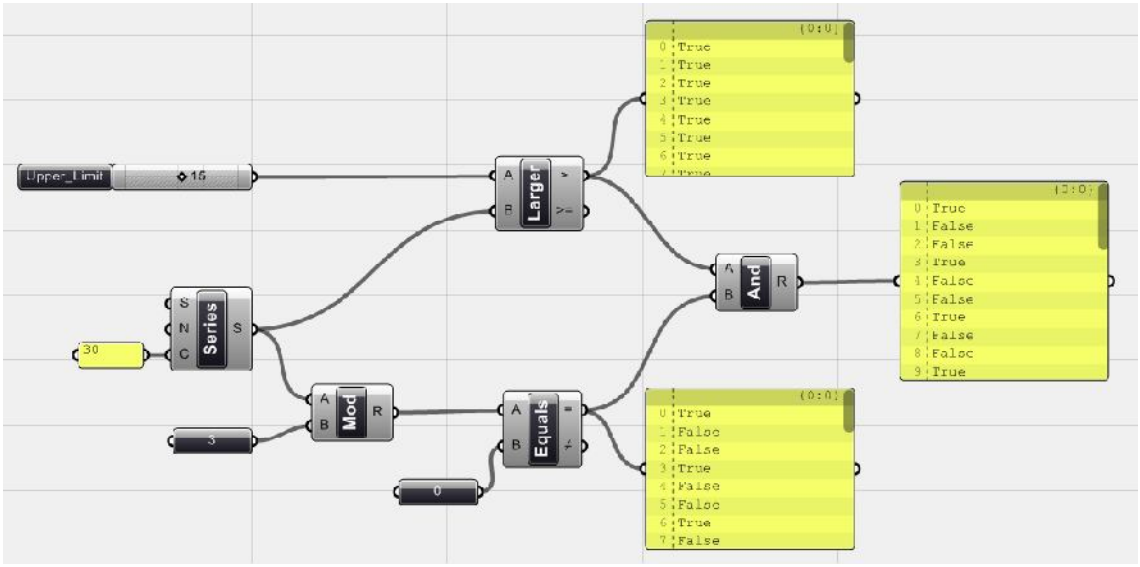


Рис. 28. Здесь объединены оба понятия. Использован компонент <Gate And> (Logic > Boolean > Gate And), к которому присоединены оба компонента <function>, для выполнения операции их булева умножения. Результатом операции является значение True, когда оба входных булевых значения равны True, в противном случае результат False. Как видно, данному условию отвечают числовые значения, которые меньше <Upper_limit> и делятся на 3.

Есть несколько логических операторов на вкладке Logic панели Boolean. Вы можете использовать и сочетать многие из них для создания критериев, принятия решений и создания своего дизайна на основе этих решений. Мы обсудим, как использовать эти логические значения позже.

3.6. Фильтрация списков

В ряде случаев, требуется выбрать лишь некоторые элементы из данного набора данных, а не применять функцию к каждому из них. Чтобы это сделать, мы должны либо выбрать некоторые конкретные пункты из списка, либо пропустить ненужные элементы. Существуют различные способы для достижения этой цели, но давайте начнем с пропуска или фильтрации (выбраковки) списков данных.

На данный момент в Grasshopper присутствует три компонента <cull> для фильтрации списков данных. Компонент <cull Nth> пропускает каждый N-й элемент заданного списка. <cull pattern>

использует шаблон логических значений (True / False) и отбирает список данных по этому шаблону. Это означает, что каждый элемент списка данных ассоциируется с булевым значением из шаблона. В результирующий список попадают те значения, которым соответствует значение True, а те элементы, которым соответствует False, исключаются из списка. <Cull Index> фильтрует список данных по индексам элементов.

Если количество значений в списке данных и в списке логических значений одинаково, то каждый элемент списка данных оценивается по соответствующему пункту списка логических значений. Но вы можете задать шаблон логических значений (например: False / False / True / True, который предопределен в компоненте) и компонент <cull> будет повторять этот шаблон для всех элементов из списка данных.

Для лучшего понимания, здесь ниже представлены некоторые из способов, которыми мы можем выбрать нужные нам геометрические объекты (в данном случае точки) из предопределенного набора данных.

Пример с расстоянием

Допустим, необходимо выбрать некоторые точки из множества точек на основе их расстояния до другой точки (точки отсчета). Набор точек и точка отсчета задаются с помощью компонента <Point>. Прежде всего, нам потребуется компонент <distance> (Vector> Point> Distance), который измеряет расстояние между точками множества и точкой отсчета, и в результате формирует список чисел (расстояний). Полученные расстояния мы сравниваем с помощью компонента <F2> (Logic > Script > F2 / функция двух переменных), задавая определенное число с помощью <number slider>. Это сравнение генерирует на выходе логические значения (True / False), чтобы показать, является ли значение меньше (True) или больше (False), чем верхний предел F = X> Y (это то же самое, что компонент <Larger>). Эти логические значения передаются компоненту <Cull pattern>.

Как упоминалось ранее, компонент <Cull pattern> берет список исходных данных и список логических значений и исключает тех членов исходного списка данных, которые соответствуют значениям 'False' списка логических значений. Таким образом, на выходе из компонента <Cull pattern> получается множество точек, которые соответствуют значению True. Это означает, что они ближе, чем число заданное <number slider>, к исходной точке, так как функция X > Y всегда возвращает True для меньших значений Y (Y = расстояние). Чтобы нагляднее изобразить полученный результат, мы просто соединим их с точкой отсчета, с помощью обычного <line>.

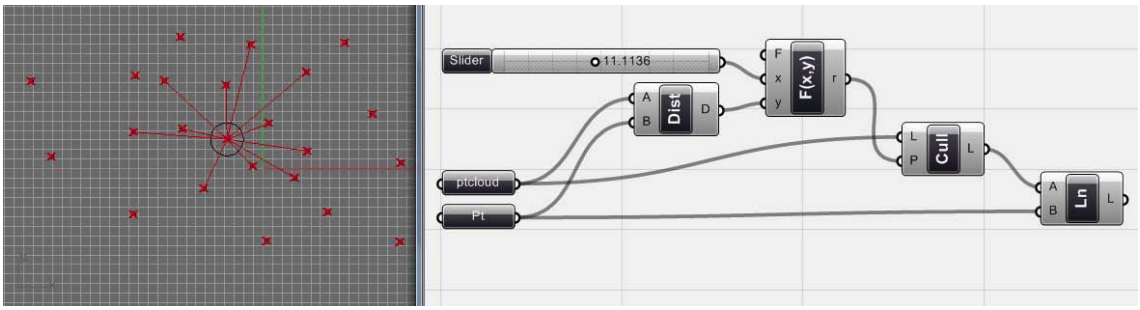


Рис. 29. Выбор точек из множества по расстоянию до опорной точки, используя компонент <Cull pattern>.

Пример с топографией

Протестировав первый пример на логику расстояний, попробуем теперь выбрать некоторые точки, которые связаны с горизонталями на топографической модели, основываясь на их высоте.

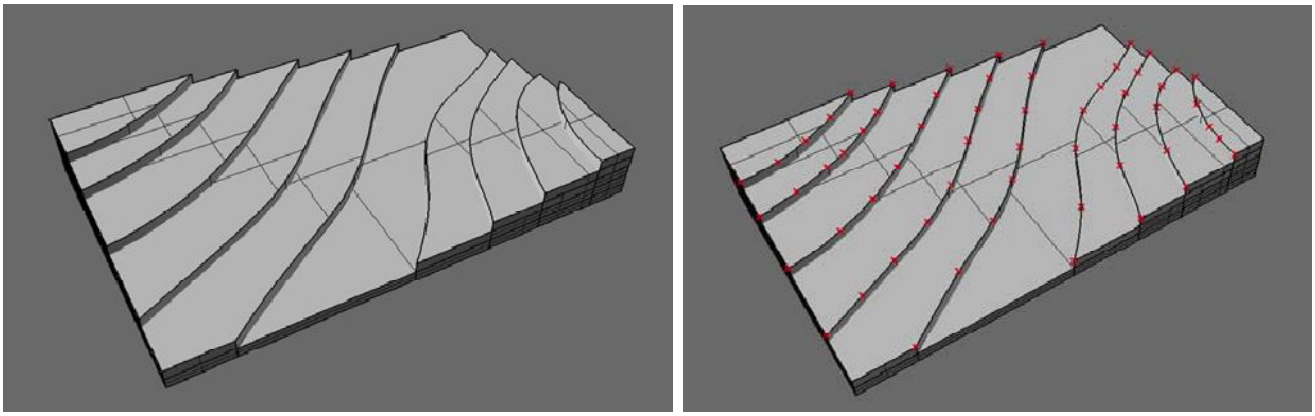


Рис. 30. Модель ландшафта с точками, расположенными на горизонталях.

Имеется множество точек, которые задаются с помощью компонента <Point> (названного Topography). Нам нужна высота точек. Следуя той же логике, что и при нахождении расстояний, мы можем выбрать желаемые точки. Здесь использован компонент <Decompose> (Vector> Point> Decompose), чтобы получить Z координаты (высоты) этих точек. Компонент <Decompose> возвращает координаты X, Y и Z каждой получаемой точки. Эти значения сравниваются с заданным числом (<number slider>) в компоненте <Larger> для получения списка ассоциативных логических значений. Компонент <Cull pattern> отбирает те точки, которые связаны со значениями True. Это означает, что выбранные точки выше, чем заданное пользователем значение высоты.

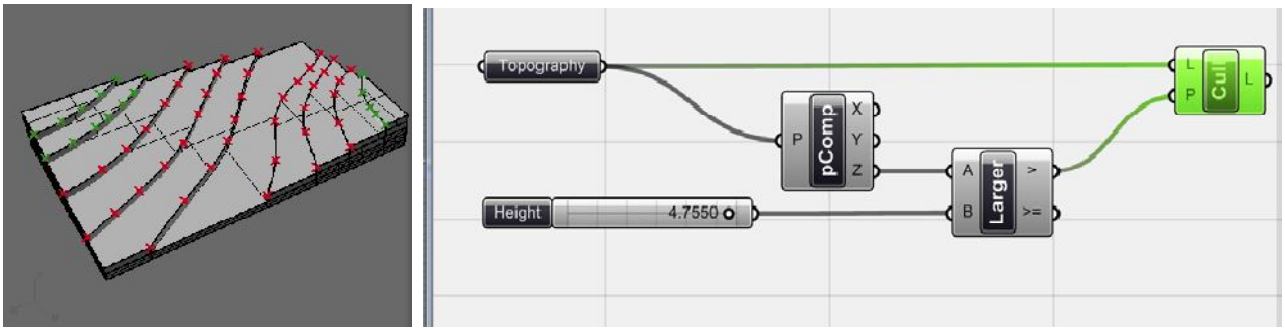


Рис. 31. Выбраны точки с координатой Z > 4,7550 единиц! (Значение, заданное пользователем). Эти точки теперь готовы для посадки ваших сосен!

3.7. Списки данных

Теперь нам уже ясно, что одной из основ алгоритмического моделирования являются списки данных. В списках данных могут храниться любые данные: числа, точки, геометрические объекты и т. д. На панели List, вкладки Logic есть несколько компонентов, которые манипулируют списками данных. Мы можем извлечь один элемент из списка данных, по его порядковому номеру. Можно выделить часть списка между нижним и верхним индексами и так далее. Эти компоненты управления списками пригодятся нам, чтобы получить желаемый список данных для наших целей дизайна. Посмотрите на некоторые примеры:

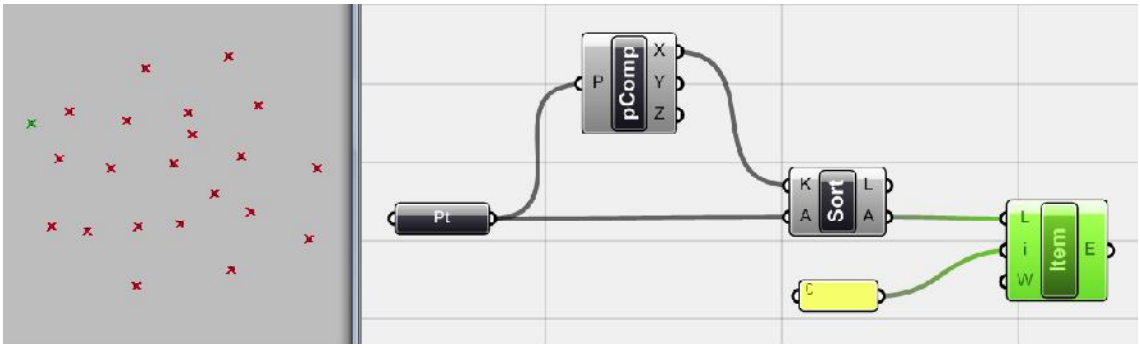


Рис. 32. Здесь показан список точек. Мы хотим выбрать точку с минимальной координатой X. Как уже было сказано выше, компонент <point decompose> дает нам координаты точек. Что нам нужно сделать, так это найти минимальное значение X среди всех точек. Для достижения этого, необходимо отсортировать всех координаты X, чтобы найти минимум. Сортировку координат можно выполнить с помощью компонента <Sort List>. Проще говоря, компонент <sort>, сортирует список (или несколько списков) данных на основе списка числовых значений, используемых как ключи сортировки. Поэтому, когда он сортирует ключи, связанные с ними данные также будут отсортированы. Так вот, мы отсортировали все точки, используя в качестве ключей координаты X. Все что нам сейчас нужно, это выбрать первый пункт этого списка. Для этого нам нужен компонент <Item>, который извлекает точку из списка по ее порядковому номеру. Первый элемент (индекс 0) имеет минимальное значение X. Мы извлеки индекс 0 из списка и поэтому на выходе компонента будет точка с минимальным значением X на множестве точек.

Треугольники

Давайте развивать наши эксперименты с управлением данными. Представьте, у нас есть сеть точек, и мы хотим нарисовать линии таким образом, чтобы образовались треугольники с шаблоном (паттерном), как на рисунке. Эта концепция полезна при создании полигональных сетей, панелизации и в других подобных случаях. Поэтому с данного момента важно чтобы вы были в состоянии генерировать эту основную концепцию.

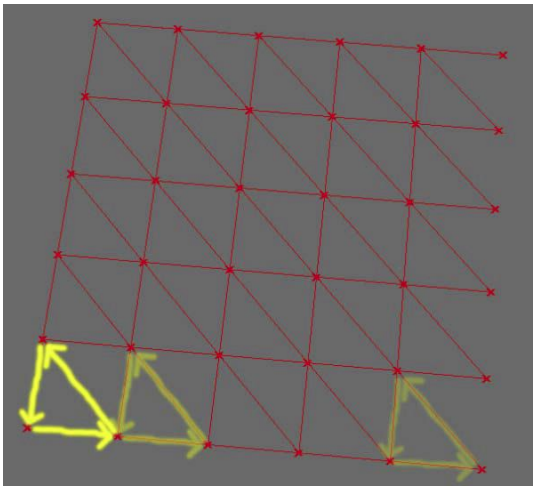


Рис. 33. Создание треугольников на основе сети точек.

Первым шагом является создание сетки точек компонентами <series> и <pt>. Следующим шагом является нахождение соответствующих точек, для рисования линий между ними. Всякий раз, когда нам нужен треугольник, мы рисуем линию из начальной точки в конечную, которая располагается на той же самой строке, но в следующем столбце. Следующую линию рисуем из этой конечной точки, к следующей строке и на один столбец назад. Последняя линия возвращается обратно, в исходную точку. Чтобы сделать это, нам потребуется иметь три различных списка точек, один для всех ‘первых точек’, один за всех ‘вторых точек’, а другой для всех ‘третьих точек’, а затем нарисовать линии между ними.

Мы можем использовать исходные точки в качестве списка для всех ‘первых точек’. В качестве первой ‘второй точки’ можно использовать вторую точку исходного множества точек, а затем этот список можно продолжить, один за другим. Так, чтобы выбрать ‘вторые точки’ мы просто сдвигаем оригинальный список компонентом <Shift> (Logic>List>Shift list) с параметром shift offset= 1. Это означает, что второй пункт списка (индекс 1) становится первым пунктом (индекс 0) и с остальной частью списка будет тоже самое. Этот новый список и есть список ‘вторых точек’.

'Третьи точки' треугольников находятся в том же столбце, что и 'Первые точки’, но в следующей строке. Что касается индексов, то если решетка состоит из N столбцов, то первая точка во второй строке имеет индекс = индекс первой точки (0) + N

В сетке с 6 столбцами, индекс первой точки второй строки будет равен 6. Поэтому мы должны сдвинуть первоначальный список точек снова, используя параметр shift offset = количеству столбцов, чтобы получить первую точку следующей строки (величина сдвига задается <number slider>, который равен числу столбцов), чтобы найти все 'третьи точки’.

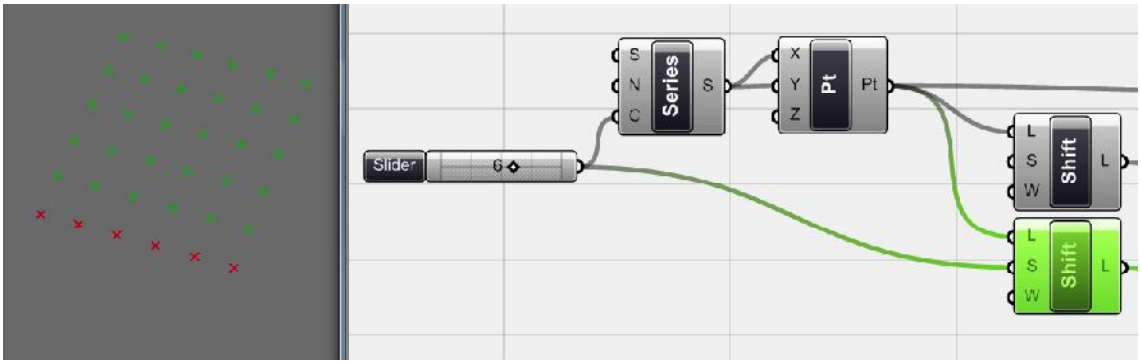


Рис. 34. Выбранный элемент смещает точки на величину shift offset, равную числу столбцов, тем самым задавая все ‘третьи точки’ треугольников.

Для завершения задачи нам нужно еще поработать с этими списками, сосредоточившись на этот раз немного больше:

1. Прежде всего, в списке 'первых точек', точки в последнем столбце никогда не могут быть первыми точками треугольников, так что нам нужно исключить их из списка ‘первых точек'.

2. Точки в первом столбце также никогда не могут быть ‘вторыми точками', поэтому нам нужно исключить их из списка ‘вторых точек'.

3. То же самое для 'третьих точек’, для которых точки последнего столбца никогда не могут быть третьими точками треугольника.

Если объединить все эти три части и изобразить то, что получится, то мы поймем, что во всех трех списках, точки в последней колонке не могут быть использованы.

Поэтому мы должны пропустить последний столбец из каждого списка данных. Вот почему мы присоединили для каждого из списков точек по одному компоненту <Cull Nth>. <Cull Nth> пропускает каждый N-й элемент из списка данных. N является частотой пропуска (см. рис). В этом случае, во всех списках данных для пропуска используется количество столбцов. Например, если у нас 6 столбцов, то компонент пропускает каждый шестой пункт списка, т.е. точки в последнем столбце. И в результате новый список не содержит последнего столбца.

Так что мы просто подключаем <number slider>, который задает число столбцов, к каждому компоненту <Cull Nth> как частоту.

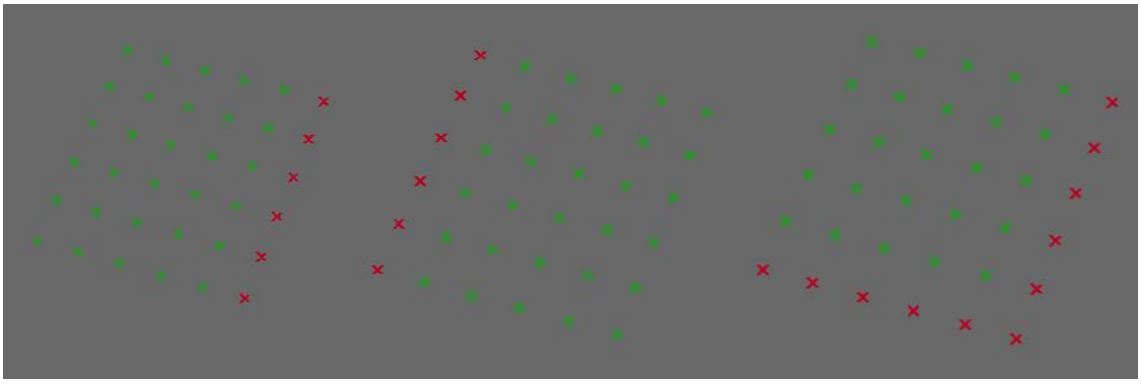


Рис. 35. Использование <Cull Nth> для пропуска последнего столбца первого, второго и третьего списков точек.

Последним этапом является задание трех компонентов <line>, чтобы связать первые точки со вторыми, потом вторые с третьими и, наконец, третьи снова с первыми.

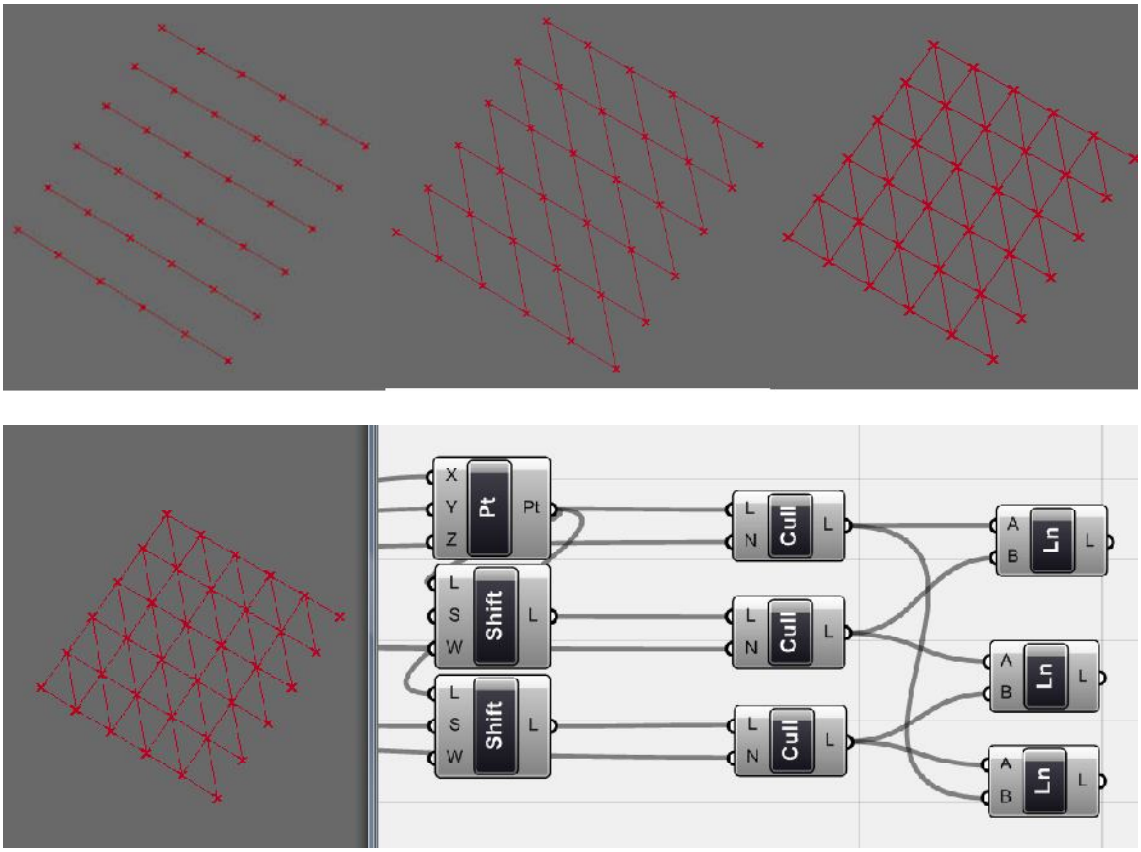


Рис. 36. Создание линий при соединении отфильтрованных списков точек компонентом <line>. Не забывайте, что согласование данных в компоненте <Pt> установлено в значение Cross Reference, а для компонентов <line> в Longest List.

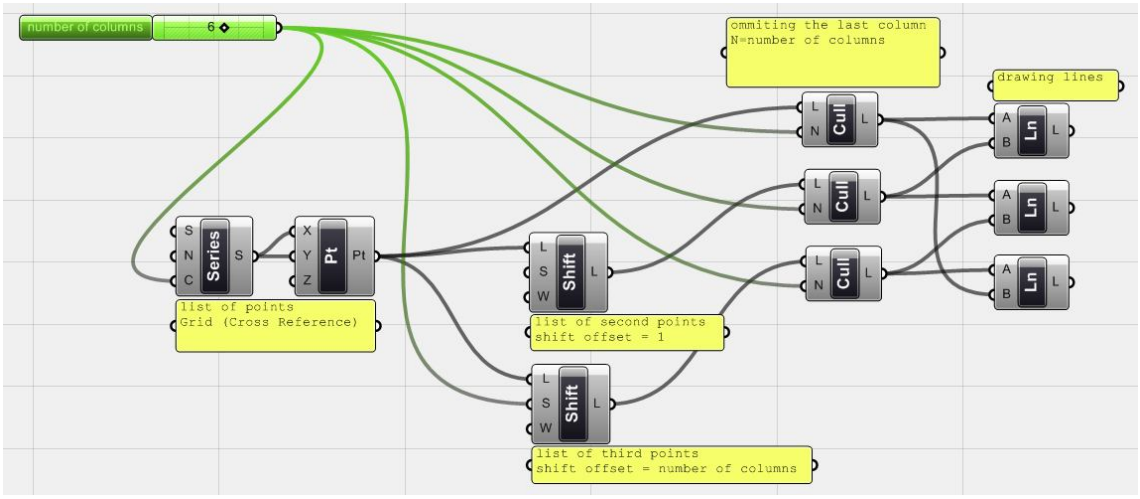


Рис. 37. Теперь, изменяя значение <number slider>, вы можете получать сети точек различного размера, которые будут состоять из треугольников.

Хотя остались еще некоторые проблемы с нашим дизайном и мы знаем, что мы не должны начинать строить треугольники с точек последней строки (мы должны опустить их из списка "первые очки"), но концепция ясна ... ... поэтому идем дальше. Мы вернемся к этой идее, когда будем обсуждать полигональные сети и тогда мы постараемся оптимизировать ее. Основная

идея состоит в том, чтобы увидеть, как создавать данные и как ими управлять. Давайте развивать наше понимание этого через эксперименты.

3.8. Геометрические узоры на плоскости

Геометрические узоры являются одними из возможных результатов дизайна, с использованием, как генеративных алгоритмов, так и в Grasshopper. Мы задаем основу для разработки мотива, а затем он распространяться в качестве образца, который может быть использован в качестве базы других продуктов дизайна. В случае разработки узоров мы должны иметь концептуальный взгляд на наш дизайн / модели и искать логику, с помощью которой создается повторяющаяся форма. Так, рисуя простую геометрию, мы можем скопировать ее для получения рисунка, такого размера, который нам нужен (см. рис.).

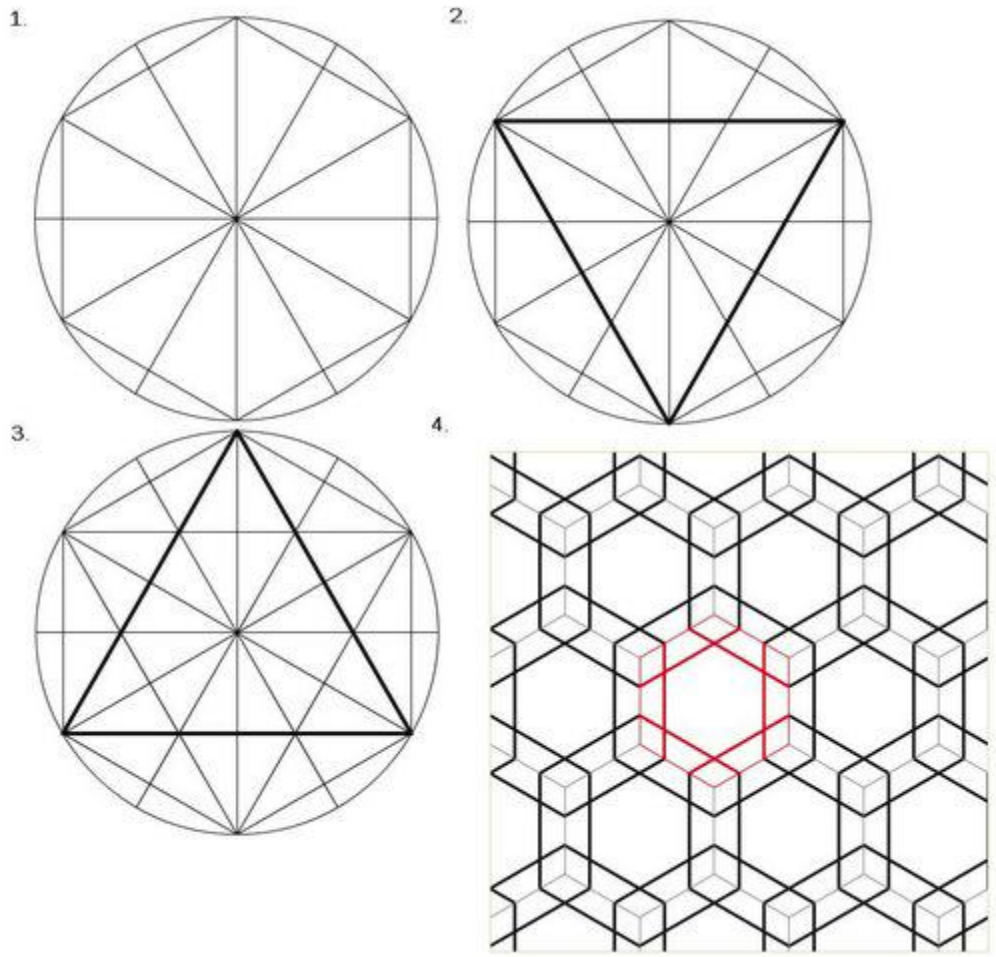


Рис. 38. Извлечение идеи паттерна (шаблона) из простой геометрии.

Мы по-прежнему будем рассматривать при работе над этой темой наборы данных и простые математические функции, а не другие полезные компоненты, чтобы увидеть, что эти простые операции и числовые множества данных, несут большой потенциал для создания формы, даже классической геометрии.

Рис. 39. Сложная геометрия плитки иранской мечети Шейх-Лотф-Аллах состоит из простых узоров, которые созданы с помощью математико-геометрических расчетов.

Простой линейный узор

Сейчас мы разработаем узор с некоторыми основными точками и линиями, и наша цель состоит в использовании простой идеи, как на рисунке.

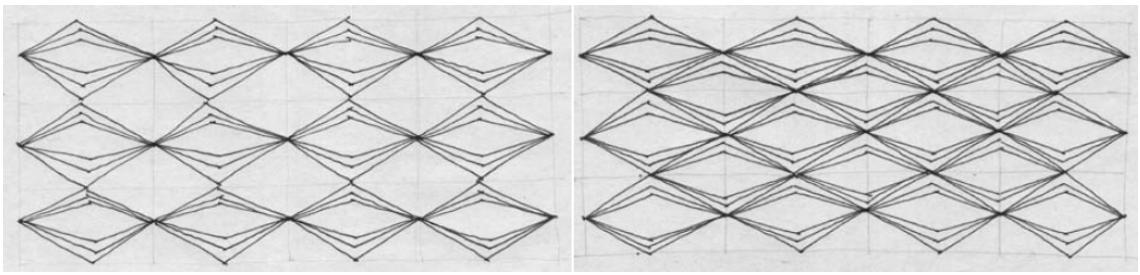
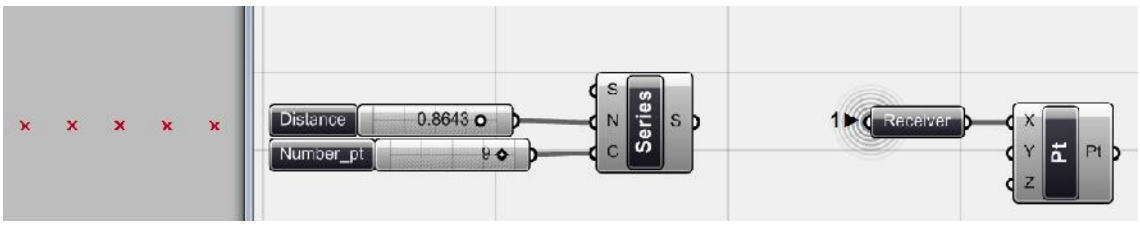


Рис. 40. Основная идея создания узора.

Прежде всего, мы хотим сгенерировать базовые точки, как основу геометрии, а затем соединить их между собой линиями. Мы начнем создание схемы построения с компонента <series>, что позволит контролировать количество значений (здесь число точек) и размер шага (здесь расстояние между точками). С помощью <series> мы создаем ряд точек, только с координатой X (Y и Z = 0).



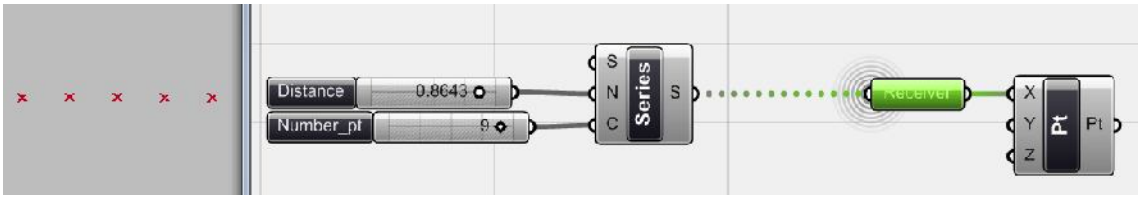


Рис. 41. Здесь создан первый набор точек с помощью компонентов <series> и <pt>. Новый секрет состоит в использовании компонента <Receiver> из Params > Special > Receiver. Этот компонент принимает данные от одного компонента и передает его на другой, при этом на холсте не отображается соединительная линия между ними. Таким образом, в сложных проектах, когда мы хотим использовать один источник данных для многих компонентов, она помогает нам очистить холст и не отвлекаться на очень длинные соединительные линии. Как видно на втором рисунке, компонент <Receiver>, получает данные из <series>.

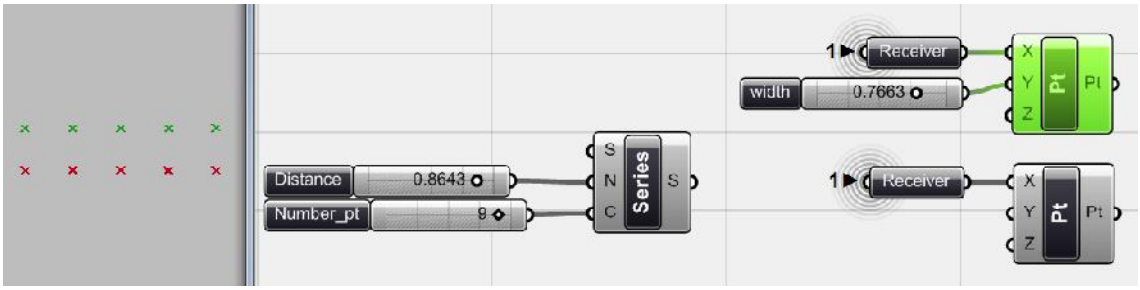


Рис. 42. Для создания зигзагообразной формы соединения нам нужно два ряда точек в качестве базовой геометрии. С этой целью использован еще один <Receiver>, для получения данных из <series>, а с помощью другого <pt> создается вторая строка точек, для которой значения Y задаются <number slider>.

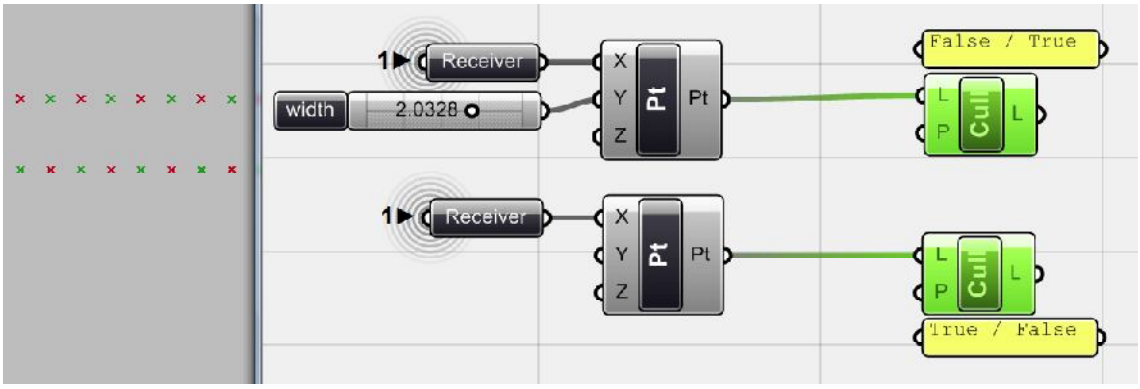


Рис. 43. На следующем этапе мы должны исключить некоторые точки, из каждого списка, чтобы получить базовые точки для пилообразного узора. Точки здесь пропускаются с помощью <cull pattern>. В первом случае используется логический шаблон True / False, а во втором False /True.

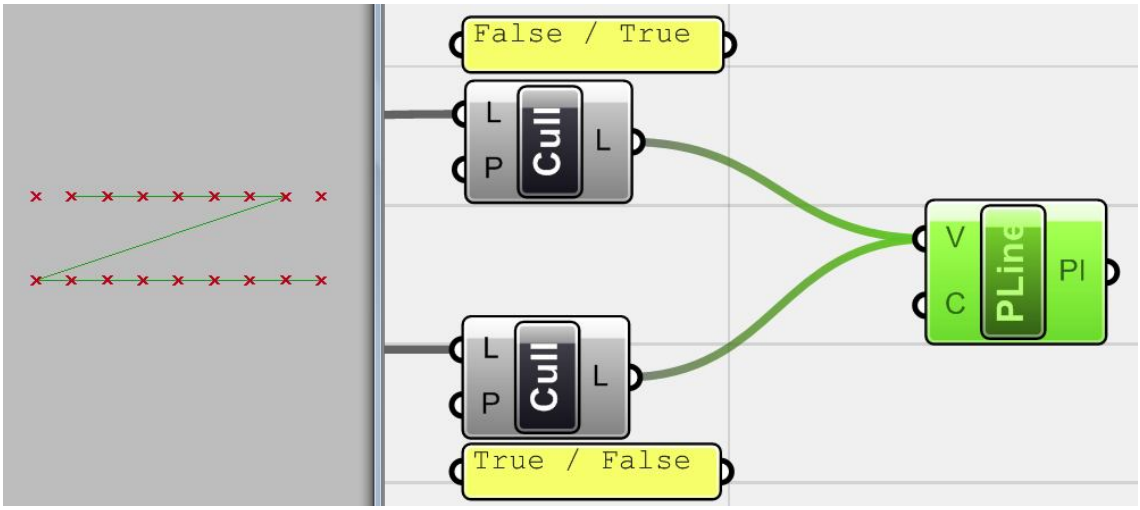


Рис. 44. Если мы сейчас подключим оба компонента <Cull> к <Poly line> из Curve > Spline, который рисует линии по нескольким точкам сразу, а не по двум, то увидим, что форма линии напоминает букву Z. Это потому, что точки не отсортированы, и их нужно отсортировать в списке следующим образом: 1-я_pt из 1-й строки, 1-я_pt из 2-й строки, 2-я_pt из 1-й стоки, 2-я_pt из 2-й строки, …

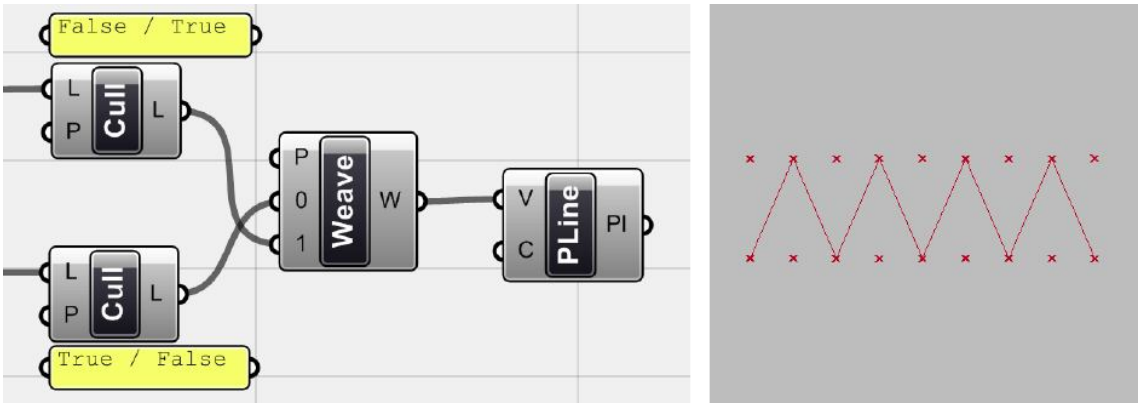


Рис. 45. Компонент, который сортирует точки способом, описанным выше - это <Weave> (Logic > List). Он принимает данные из нескольких ресурсов и сортирует их на основе паттерна, который должен быть задан на входе P (как всегда прочитайте справку по компоненту, чтобы увидеть подробную информацию). В результате получаем список отсортированных данных, и когда вы подключите его к <Pline> вы видите, что первая пилообразная линия нарисована.

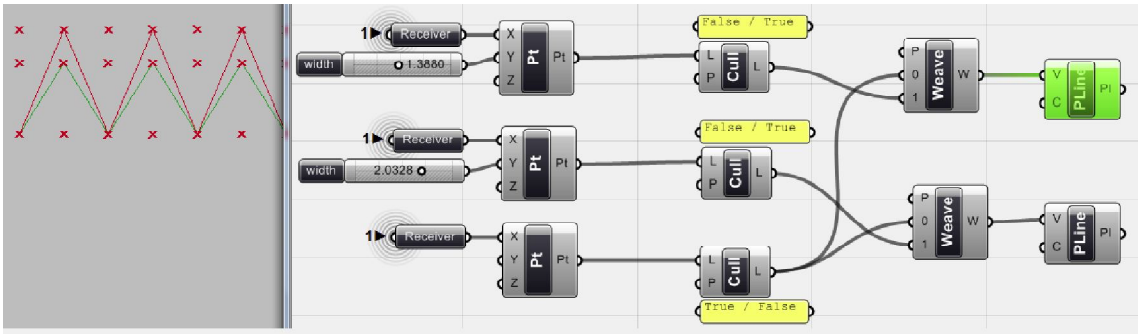


Рис. 46. Таким же образом, мы создали третий ряд точек, а с помощью других компонентов <Weave> и <Pline>, мы нарисовали вторую линию узора.

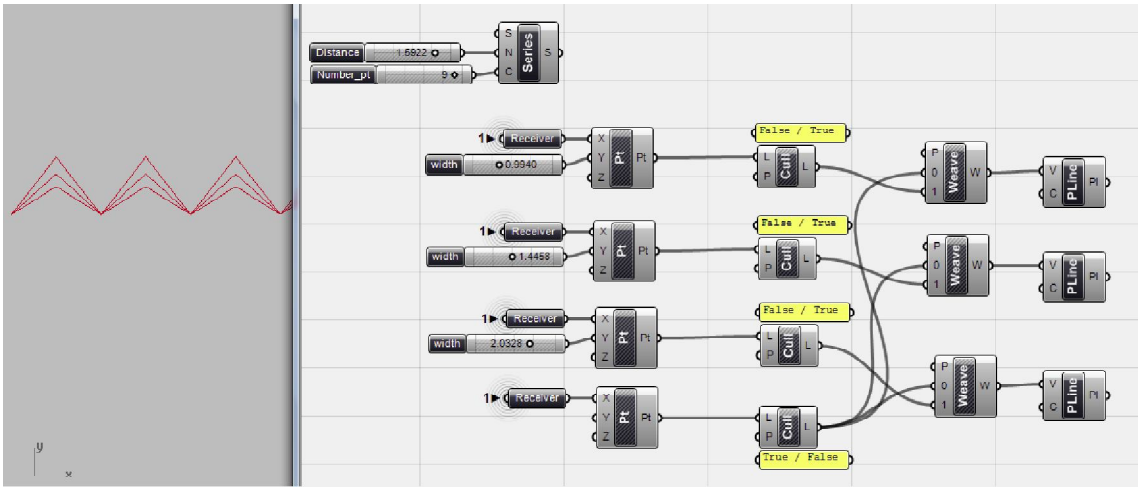


Рис. 47. Хотя существуют и более короткие способы создания этих линий, и здесь мы использовали ту же концепцию для точек и <pline> третьей строки. Мы отключили опцию Предварительный просмотр компонентов <Pt>, <Cull> и <Weave> (в их контекстном меню), чтобы скрыть все точки и увидеть только Plines.

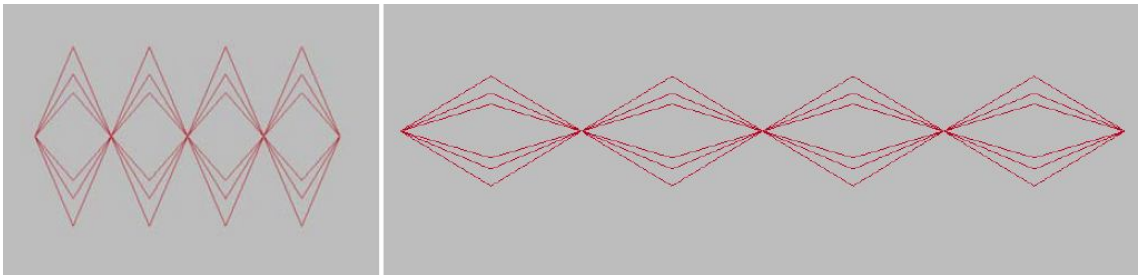


Рис. 48. Если вы скопируете весь процесс снова и в этот раз замените значения Y компонентов <pt> на отрицательные (с использованием того же <number slider> с функцией f (x) = – x), то получите зеркальную копию множества Plines. Теперь, манипулируя расстояниями, вы можете получать узоры в различных формах и масштабах.

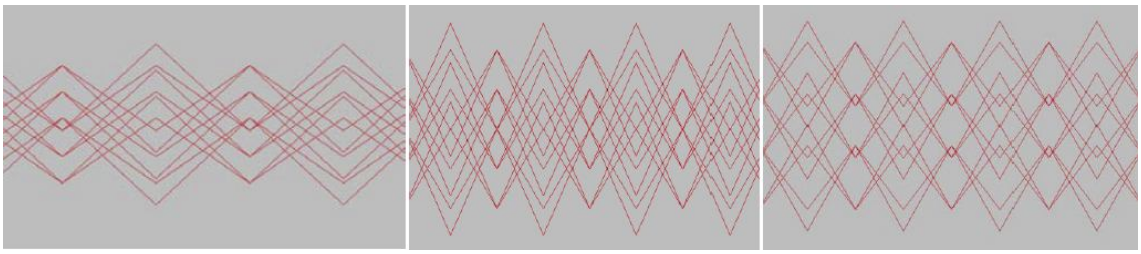


Рис. 49. Вы можете изменить способ создания своих базисных точек или фильтрации списков данных. В результате могут быть получены разные узоры из пересекающихся линий, которые являются простыми сами по себе, но могут быть использованы при создании геометрии для производства сложных моделей.

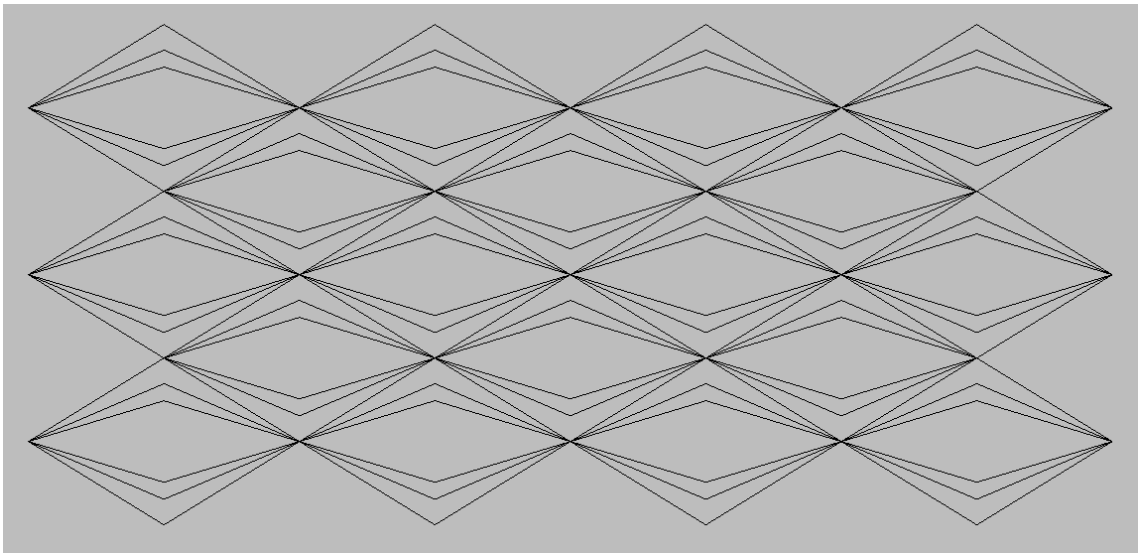


Рис. 50. Это первый результат дизайна. Мотив повторяется просто и результат может быть использован любым желаемым путем, который зависит от ваших целей.

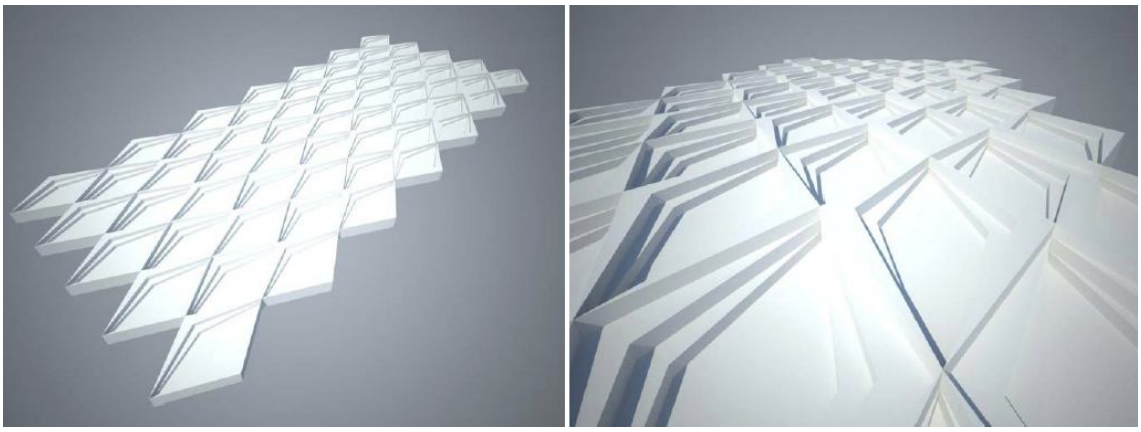


Рис. 51. И это только один из примеров среди сотни возможностей использования этих основных шаблонов для разработки дизайна. Позже у вас накопится потенциал, чтобы различать базовые узоры и получать результаты, манипулируя дизайном проекта.

Круговые узоры

Существуют бесконечные возможности для создания мотивов и узоров, используя аналогичные методы моделирования. На рис. показан еще один мотив, который составляется на основе не линейной, а круговой геометрии. Начав с нескольких кривых, которые все имеют ту же логику, мы опишем одну часть алгоритма, а оставшуюся возложим на вас.

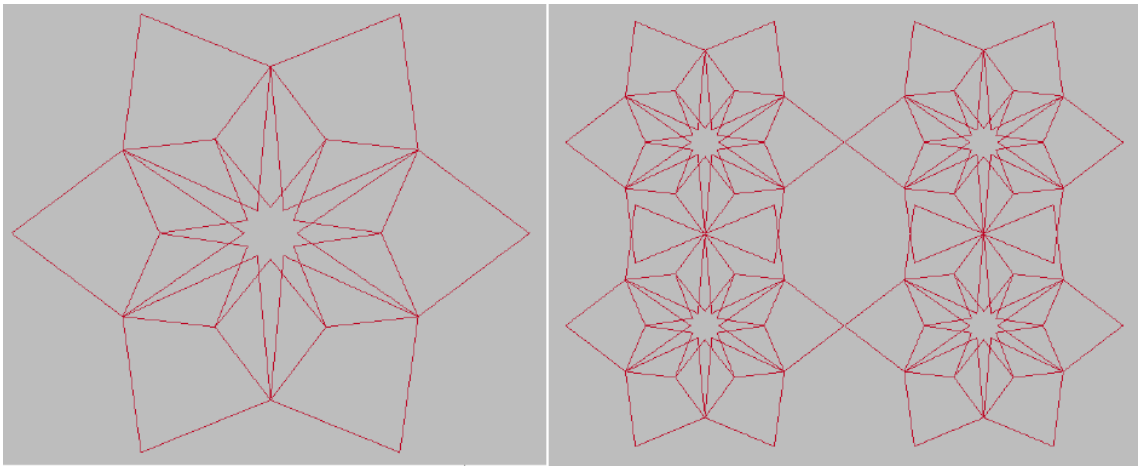


Рис. 52. Круговые геометрические узоры

Основой этой модели является числовой ряд, на основе которого вычисляется совокупность точек, расположенных по окружности, подобно тому, как мы делали раньше. Этот набор данных может быть масштабирован из центра, чтобы получить больше и больше окружностей вокруг того же центра. Мы будем отбирать эти множества точек, так же, как в последнем примере. Затем мы будем генерировать повторяющиеся зигзагообразные узоры из этих отмасштабированных круговых точек, чтобы соединить их друг с другом и получить кривую в форме звезды. Совмещение этих звезд позволяет сделать одну часть мотива.

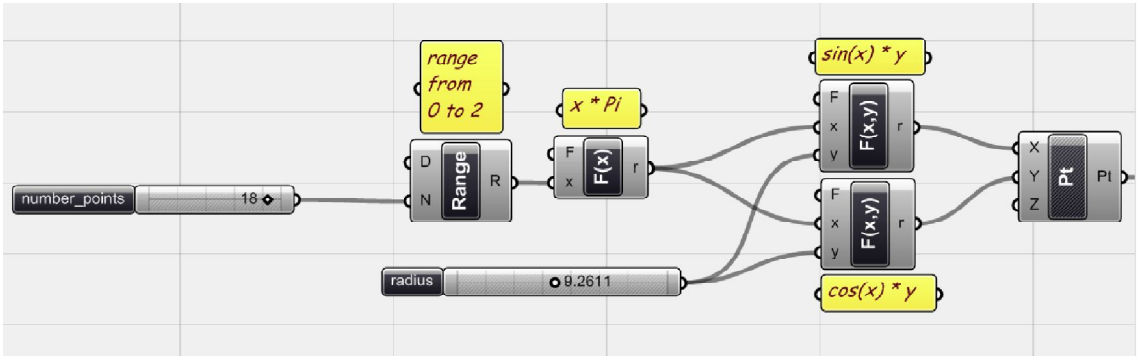


Рис. 53. Задание диапазона от 0 до 2Pi и использование функций Sin / Cos, для создания первого набора точек круговой геометрии. Мы использовали функцию с двумя переменными для умножения результата Sin / Cos на значение <number slider>, чтобы контролировать радиус окружности.

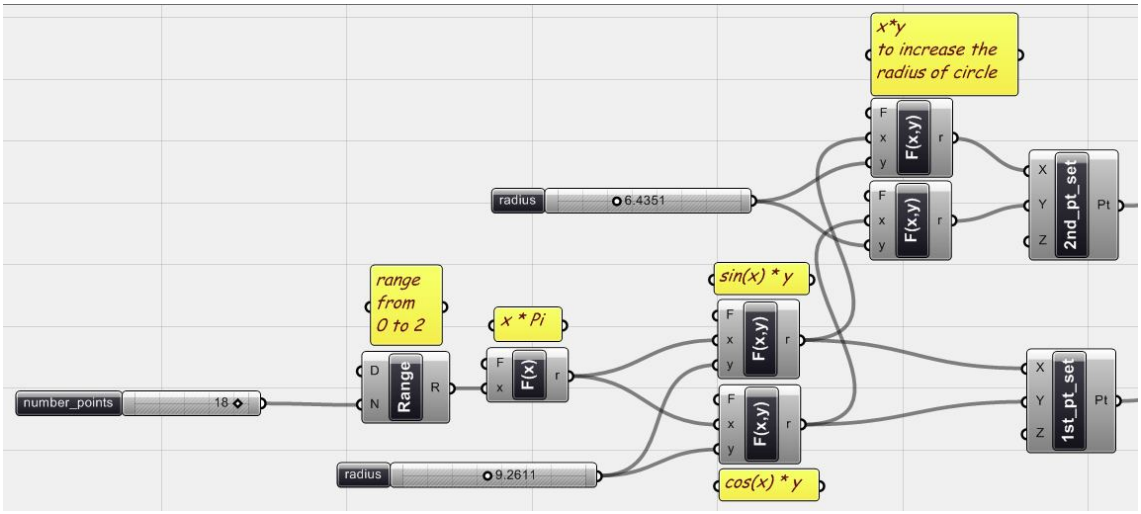


Рис. 54. Увеличение результата функций Sin / Cos умножением на <number slider> создает второй набор точек с большим радиусом. Как видно, результатом этой части алгоритма, является два точечных множеств. Компоненты <pt> здесь переименованы.

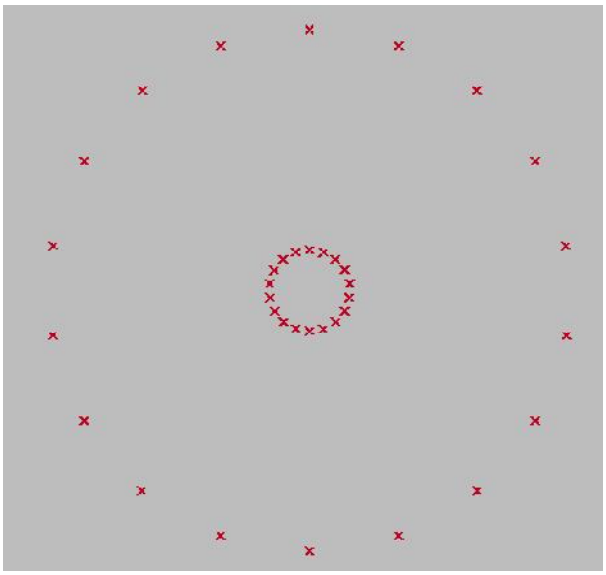


Рис. 55. Первая и вторая окружности из точек

Для того чтобы отобрать нужные точки, мы можем просто использовать для точек <Cull pattern> с логическим шаблоном True / False, как в последнем примере. Но как нам отсортировать список точек в конечном итоге? Можно снова использовать компонент <weave>. Но здесь мы воспользуемся другим способом сортировки, который будет полезен в дальнейшем. Мы хотим отсортировать точки в зависимости от их порядкового номера в списке.

Прежде всего, нам нужно создать список индексов точек. Поскольку точки мы создавали с помощью компонента <range> с вещественными числами, то сейчас нам нужен компонент <series>, чтобы использовать целые числа в качестве индексов точек в списке. Параметр N компонента <range> определяет число шагов или подразделений, поэтому <range> генерирует N +1 чисел. Вот почему нам нужен <series> с N +1 значениями для индексов точек.

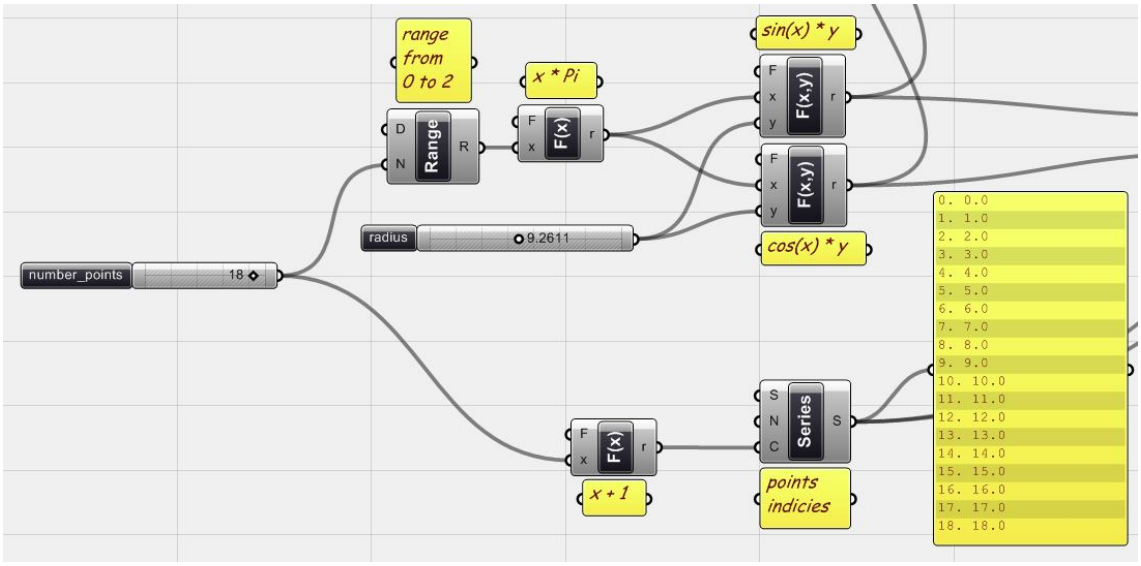


Рис. 56. Создание индексов точек (список целых чисел, начинающийся с нуля)

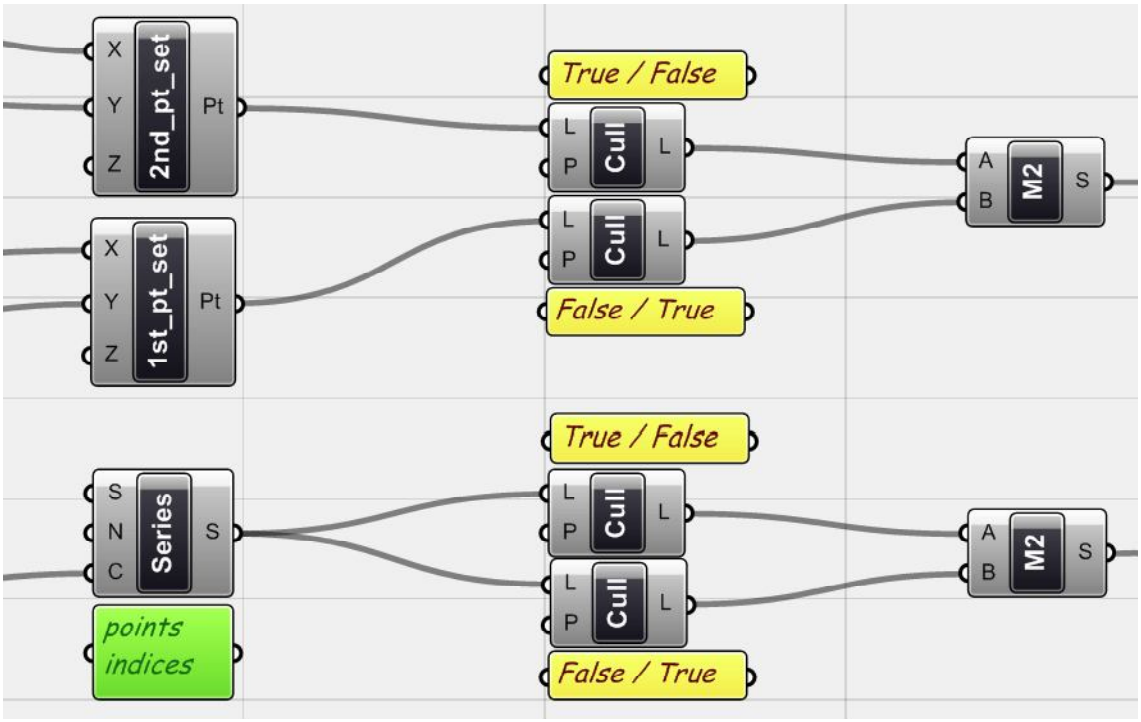


Рис. 57. Теперь нам нужно отфильтровать точки и индексы, также как в предыдущем примере. Поэтому мы использовали компонент <merge> (Logic > Tree), чтобы создать один список данные из отфильтрованных с помощью <cull> списков. Это сделано как для точек, так и для индексов. Хотя результатом слияния для <series> снова будут числа всего набора данных, но их порядок будет уже другим. Теперь при сортировке индексов в качестве ключей сортировки мы можем сортировать также и связанные точки.

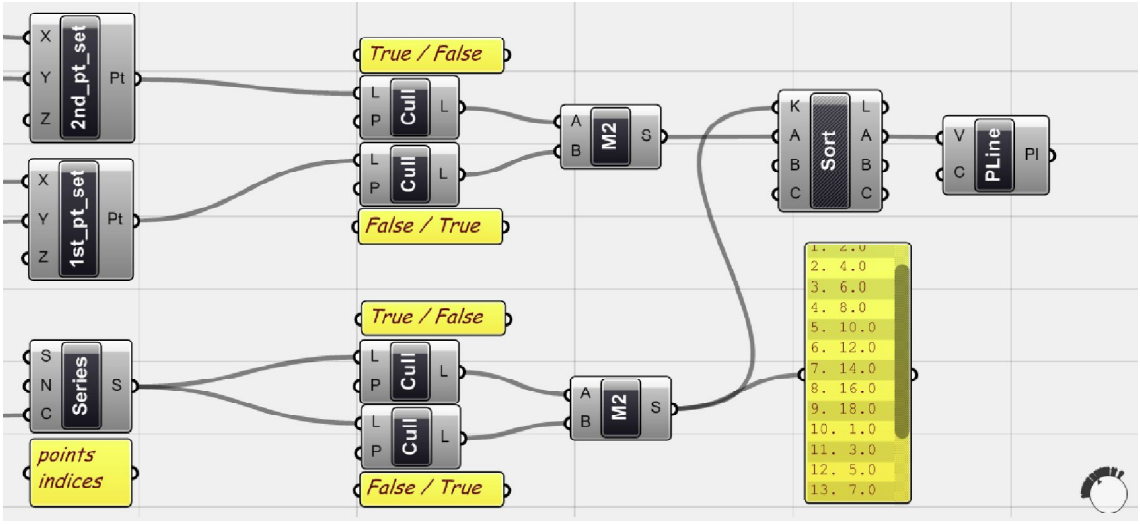


Рис. 58. Точки сортируются с помощью компонента <sort>, который использует список ключей сортировки в качестве индексов. Полилиния строится по отсортированным точкам.

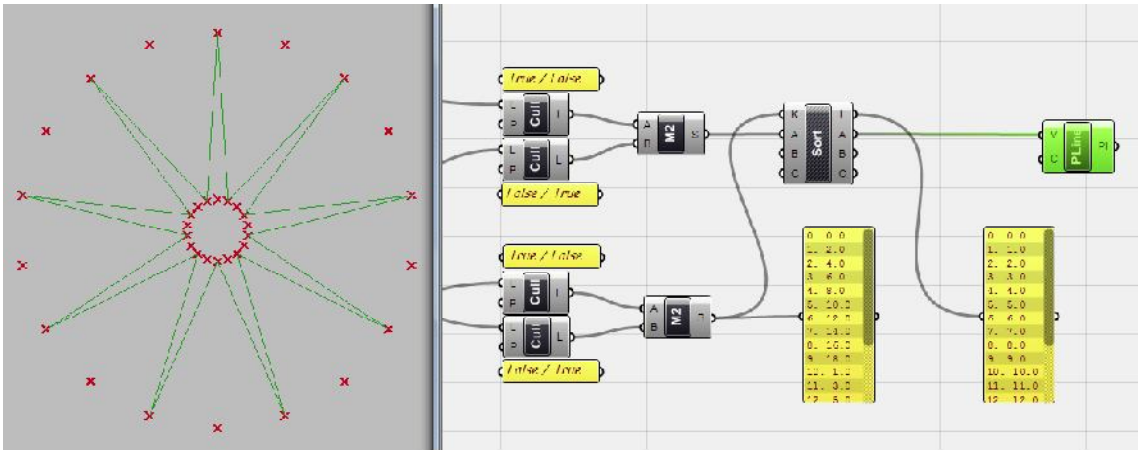


Рис. 59. Индексы до и после сортировки, и сортируемые в соответствии с ними точки задают звездообразную форму линии.

Та же логика может быть использована для создания более сложной геометрии, простой генерацией других множеств точек, фильтрацией и соединением их вместе, чтобы в итоге получить желаемый узор. Весь фокус в том, чтобы выбрать лучшие группы точек и в том, каким способов вы соедините их с другими наборами.

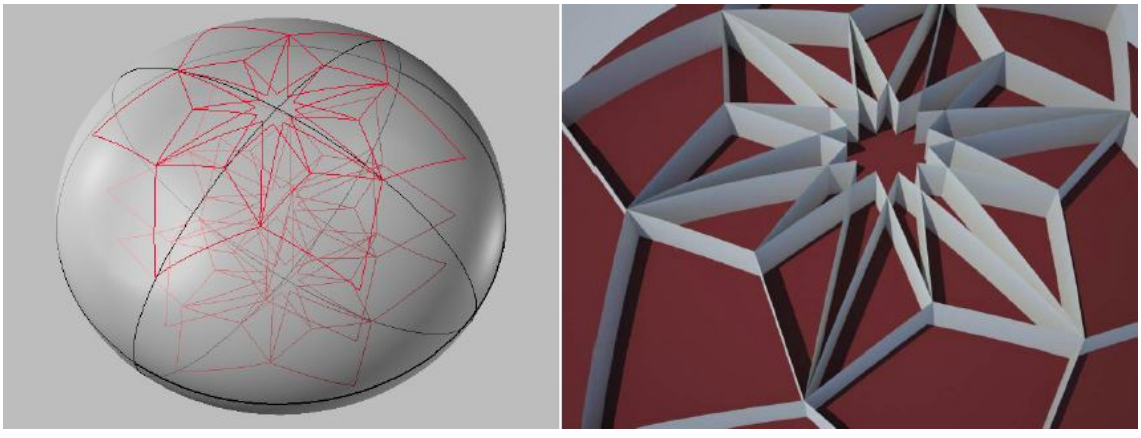


Рис. 60. Вы можете придумать и другие способы получения и использования узоров и линейной геометрии, такие как проецирование их на другую геометрию.

Хотя мы настаивали, чтобы все предыдущие модели создавались по наборам данных и с использованием простых математических функций, но мы познакомимся в дальнейшем и с другими простыми компонентами, которые позволяют сократить весь процесс или изменить способ, которым мы должны предоставить данные. Мы будем обсуждать их ниже.

Рис. 61. Конечная модель