Глава 4. Преобразования

Преобразования являются важными операциями в моделировании и генерации геометрии. Они позволяют получать различные вариации из исходных простых объектов. Преобразования помогают нам масштабировать и вращать наши объекты, перемещать, копировать, зеркально отражать их. Также можно применить несколько преобразований последовательного для получения нужного результата. Существуют различные типы преобразований. С одной стороны, мы можем разделить их на основных вида: линейные и пространственные преобразования. Линейные преобразования производятся на плоскости, в то время как пространственные преобразования выполняются в 3D-пространстве.

С другой стороны мы можем классифицировать преобразования по статусу исходного объекта; такие преобразования как перемещение, поворот и отражение сохраняют первоначальную форму объекта, а масштаб и сдвиг – изменяют исходное состояние объекта. Есть также нелинейные преобразования. В дополнение к переносу, вращению и отражению у нас есть различные типы сдвига и неоднородных масштабных преобразований в 3D-пространстве, а также спиральных и винтовых преобразований и проекций, которые делают больше вариаций в 3D пространстве.

Для того, чтобы трансформировать объекты, концептуально мы должны перемещать и ориентировать объекты (или части объектов, таких как вершины или углы оболочки) в пространстве и для этого нам придется использовать векторы и плоскости, как основу этих математических / геометрических операций. Мы не будем обсуждать здесь основы геометрии и их математическую логику, но сначала давайте посмотрим на векторы и плоскости, поскольку они будут нужны нам для работы.

Рис. 62. Преобразования предоставляют большой потенциал для создания сложных форм из отдельных объектов. В природе можно найти множество примеров, где встречаются преобразования.

4.1. Векторы и плоскости

Вектор - это математической / геометрический объект, который имеет длину и направление. Он начинается от точки и идет к другой точке на определенном расстоянии и в определенном направлении. Векторы имеют широкое применение в различных областях науки, а также в геометрии и преобразованиях.

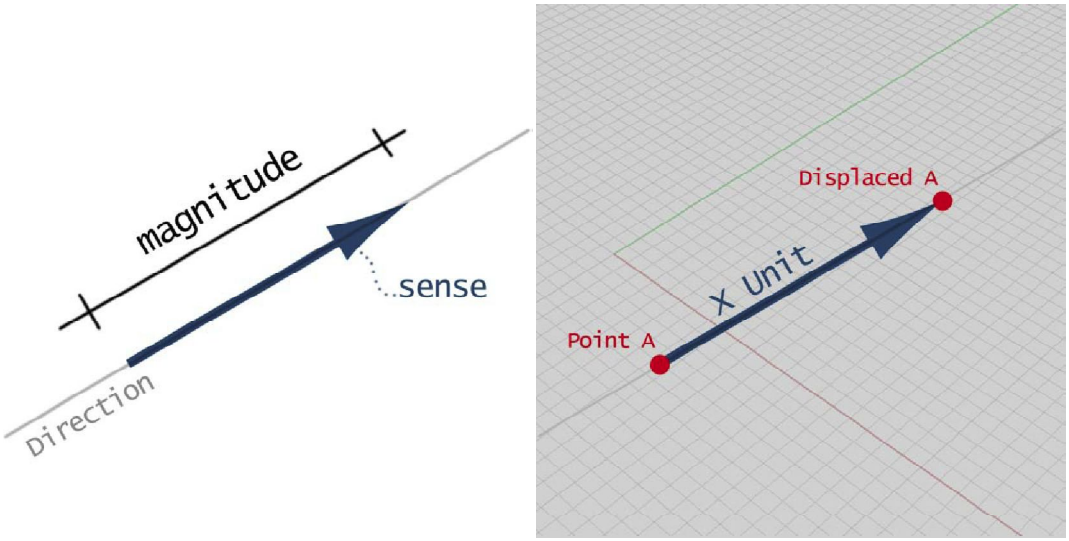


Рис. 63. A: Основные элементы вектора, B: перемещение точки с помощью вектора.

Просто если у нас есть точка и вектор, то этот вектор может сместить точку на расстояние длины вектора и в его направлении, чтобы создать новую точку. Мы используем эту простую идею для создания, перемещения, масштабирования и вращения геометрии в нашем ассоциативном методе.

Плоскости - это другой полезный набор геометрических объектов. Мы можем описать их как бесконечные плоские поверхности, которые имеют исходные точки. Конструкционные плоскости в Rhino также относятся к этому типу объектов. Мы можем использовать эти плоскости, чтобы расположить на них нашу геометрию и сделать некоторые преобразования на основе их ориентации и расположения в пространстве. Например, в 3D-пространстве, мы не можем ориентировать объекты по одному вектору! Нам нужно два вектора, чтобы создать плоскость, и разместить на ней геометрию.

Векторы имеют направление и длину, в то время как плоскости - ориентацию и начало координат. Это два разных типа конструкций, которые могут помочь нам создавать, изменять, трансформировать и создавать наши модели в пространстве.

Grasshopper имеет некоторые основные векторы и плоскости, как предопределенные компоненты. В их число входят единичные векторы X, Y и Z и плоскости XY, XZ, YZ. Есть несколько других компонентов для создания и изменения их, о которых мы будем говорить в наших экспериментах. Так давайте продолжим наши эксперименты и начнем с некоторых простых примеров использования векторов.

4.2. Кривые и линейная геометрия

Поскольку мы уже экспериментировали с точками, которые являются 0 - мерной геометрией, теперь мы можем начать думать о кривых, которые являются 1-мерными объектами. Как и точки, кривые могут быть базой для создания очень многих различных объектов. Мы можем выдавить кривую вдоль другой кривой и сделать поверхность. Мы можем соединить различные кривые вместе, для создания поверхностей и твердых тел. Мы можем распределять

какие-либо объекты вдоль кривой с определенными интервалами. Существует и множество других способов использования кривой в качестве базы геометрии создания других объектов.

Смещение

Мы создавали множества точек в 3 главе с использованием компонентов <series> и <pt>. Но существует несколько компонентов для создания решеток разного типа (прямоугольных, треугольных, шестиугольных), которые расположены в Vector > Point > Grids. Один из них, компонент <Square Grid> (квадратная решетка), который создает решетку из квадратных ячеек. В этом компоненте мы можем контролировать количество ячеек в направлениях X и Y и их размер.

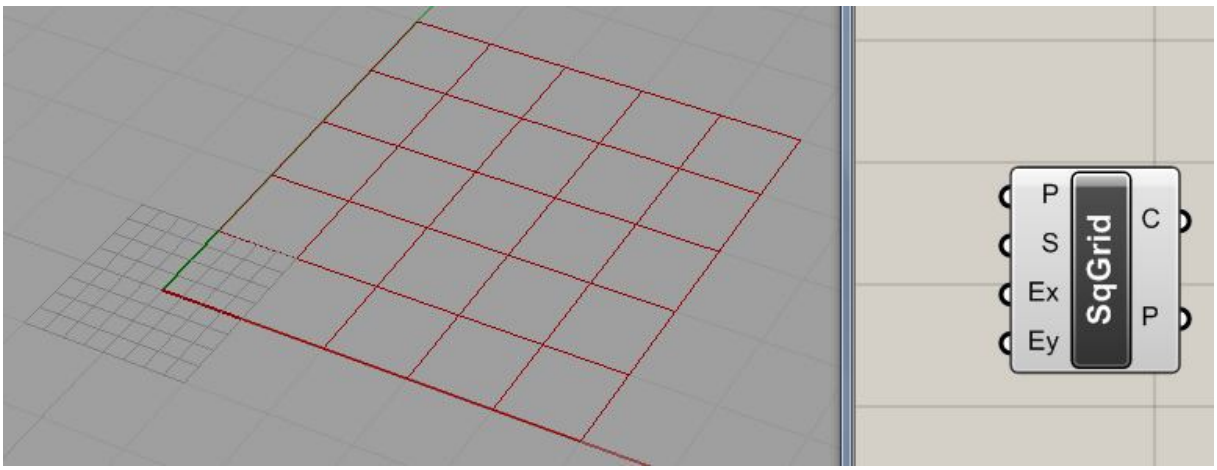


Рис. 64. Компонент < Square Grid> (квадратная решетка сетка) с предопределенными значениями.

Вы можете изменить размер ячеек с помощью <number slider>, подключив его к входу (S). Вы также можете изменить ориентацию решетки в пространстве. Для этого вам нужно задать плоскость, на которой будет располагаться сетка (вход P). Здесь мы подключили компонент <XY plane> (Vector> Plane> XY Plane), который определяет плоскость XY. Затем плоскость перемещена в направлении Z с помощью компонента <unit Z> (Vector> Vector> Unit Z), который представляет собой единичный вектор вдоль оси Z. Для того чтобы изменить расстояние перемещения плоскости, нужно изменить длину вектора. Это можно сделать, если подключить <number slider> к входу компонента <unit Z>. Изменение положения <plane XY> вдоль оси Z будет менять и позицию решетки.

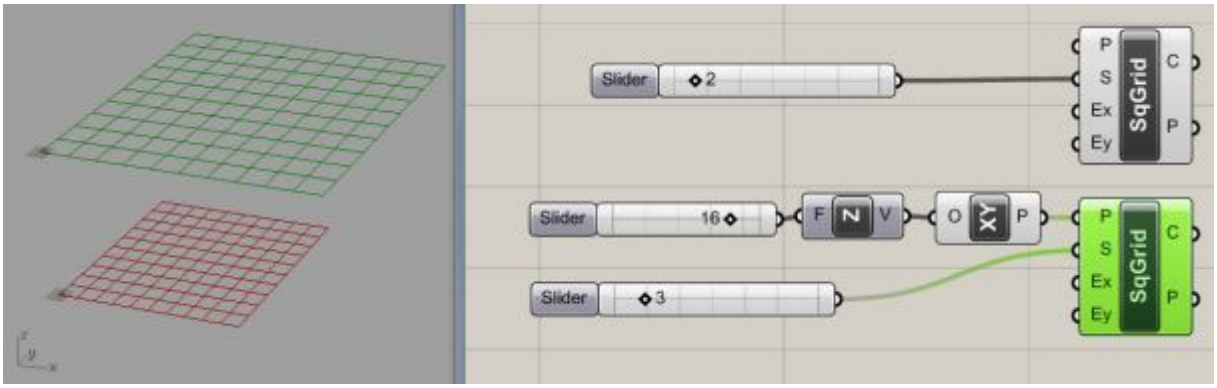


Рис. 65. Настройка компонента <Square Grid< (выделен зеленым цветом) с помощью <number slider> для изменения масштаба сетки (расстояния между точками), а также <unit Z> и <XY plane> для изменения координаты Z сетки.

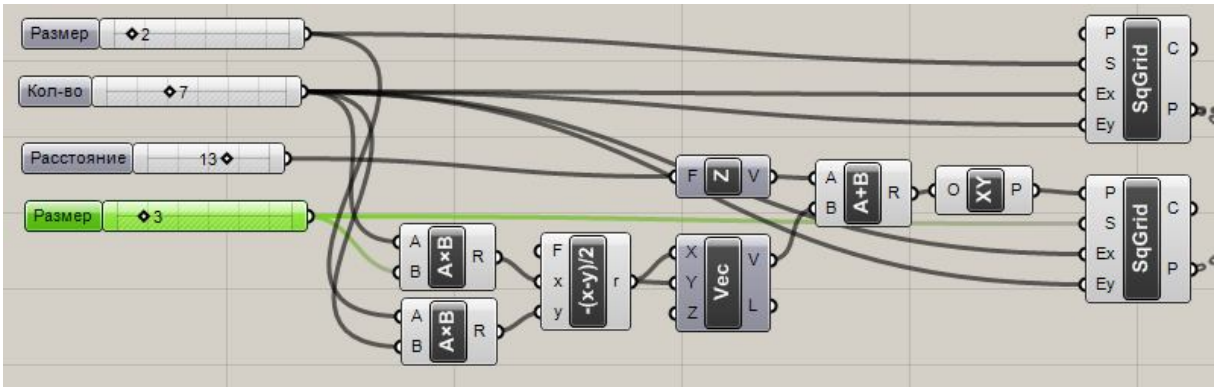


Рис. 66. Горизонтальное выравнивание компонента <Square grid> относительно исходной сетки, путем смещения его в отрицательном направлении на расстояние, равное половине разности размеров сеток. Компоненты <Multiplication> вычисляют размеры сеток. С помощью компонента <F2> с заданной функцией F(x,y) = –(x-y)/2, находится половина разности размеров.

Если посмотреть на выход <Square grid>, то можно увидеть, что мы имеем доступ ко всем линиям решетки и узловым точкам. В этом примере эксперимента, мы нарисуем пучек линий, которые начинаются из точек решетки и расходятся из них в пространстве. Мы можем просто связать точки обеих решеток и с помощью компонента <line> генерировать кучу линий. Тогда изменение размера второй сетки будет влиять на направление всех линий. Проблема заключается в том, что длина линий в этом случае также будет разной. Но мы хотим, чтобы линии были одинаковой длины. Значит, нам нужна другая стратегия и именно поэтому мы будем использовать компонент <line SDL>.

Компонент <line SDL> рисует линию от начальной точки (S) в направлении (D) и длиной (L). Т.е. длина линий управляема. Именно то, что нам нужно. Начальные точки нам известны (точки решетки), длина линии тоже. А что с направлением? Мы хотим использовать точки второй сетки в качестве конечных точек для определения направления наших линий. Чтобы определить эти направления нам нужны векторы, а не линии. Вот почему мы собираемся сделать множество векторов этих двух наборов точек для задания направления компонента <line SDL>.

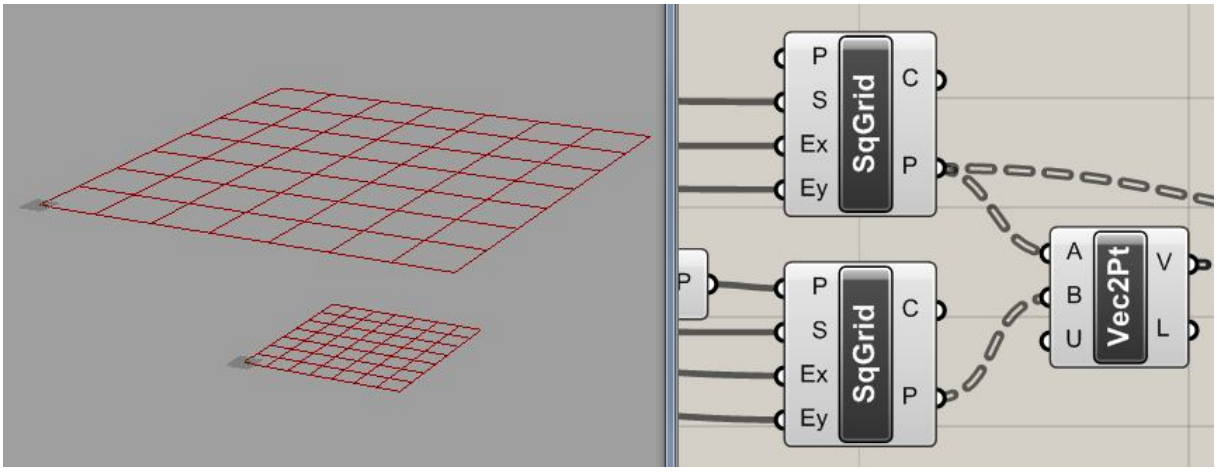


Рис. 67. Создание векторов из ячеек первой сетки к ячейками второй сетки с помощью компонента <vector 2pt> (Vector > Vector > vector 2pt). Этот компонент создает вектор по начальной и конечной точкам.

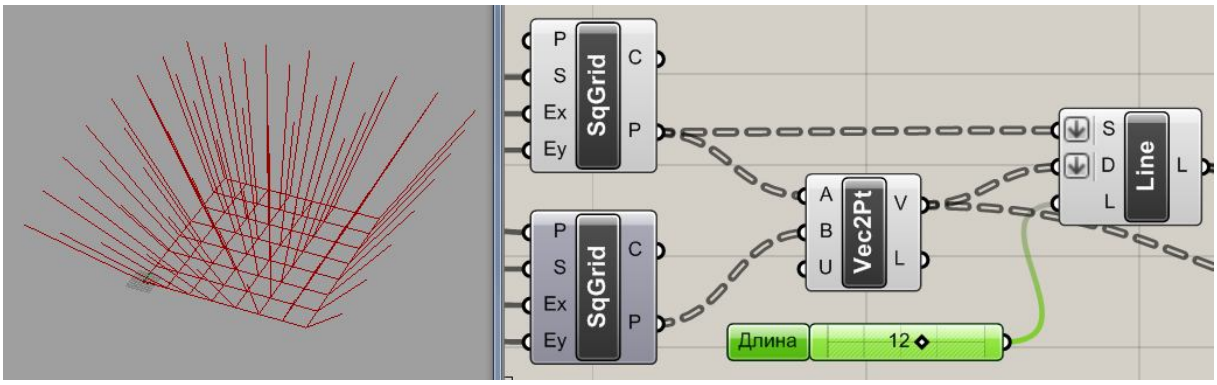


Рис. 68. Компонент <line SDL> порождает пучок линий из ячеек сетки, которые расходятся в пространстве из-за больших размеров второй сетки. Мы можем изменить длину линий, с помощью <number slider> и их направление, изменяя размер второй сетки. В контекстном меню входов S и D компонента <line> должна быть включена опция Flatten, Она объединяет отдельные списки линий для каждого ряда точек сетки в один общий список.

Следующим шагом мы ходим добавить полигон в конце каждой линии и выдавить его к начальной точке линии, чтобы увидеть потенциал компонентов кривой. Для создания полигонов мы должны добавить плоскости в конечных точках наших линий в качестве базовой плоскости, чтобы иметь возможность создания полигонов.

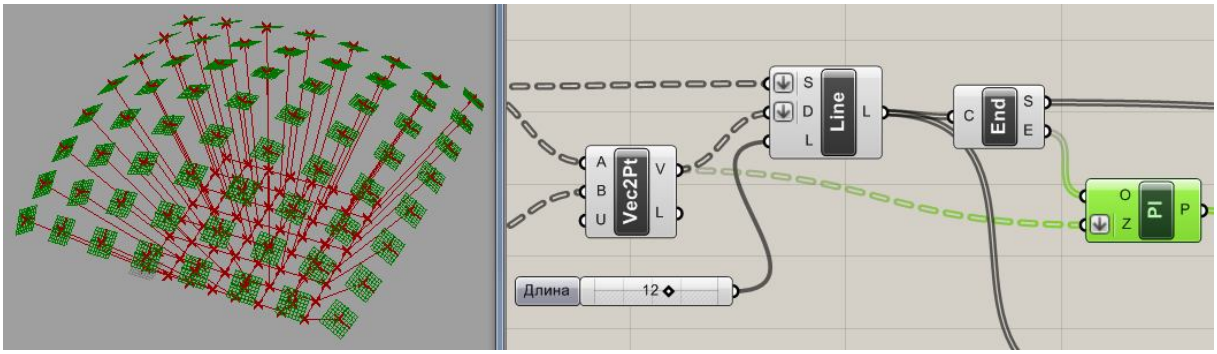


Рис. 69. С помощью компонента <End points> (Curve > Analysis) мы можем определить конечные точки линий и затем использовать эти "конечные точки", как точки начала

координат для множества плоскостей. Здесь мы использовали компонент <Plane Normal> (Vector> Plane), который создает плоскости по исходной точке (конечная точка линии) и вектору Z направлении плоскости (вектор нормали, перпендикулярный плоскости). В качестве векторов нормалей для плоскостей использовали те же векторы, что и для задания направления линий. Для входа Z плоскости необходимо включить опцию Flatten.

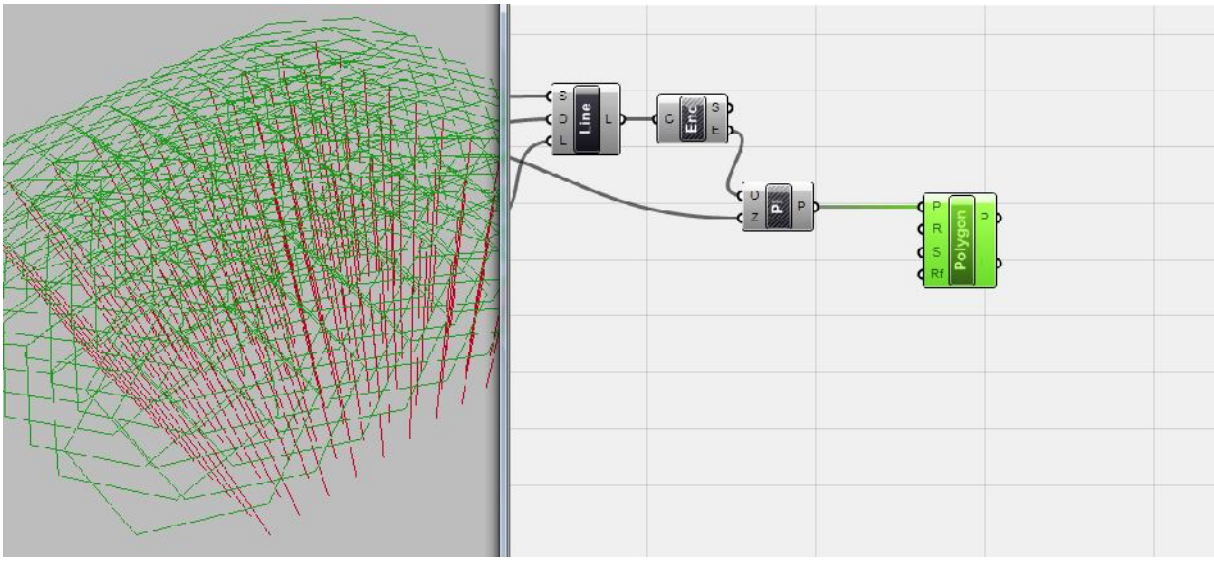
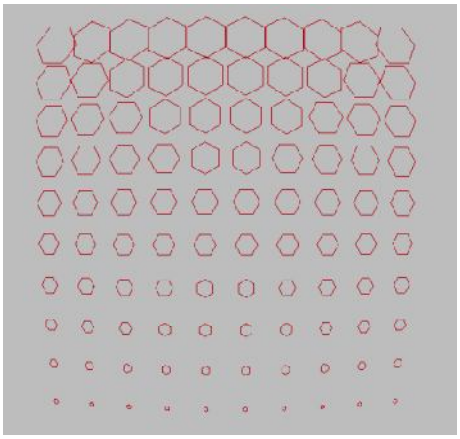


Рис. 70. Добавление компонента <Polygon> и использование созданных ранее плоскостей в качестве базовых для полигонов. Мы имеем множество многоугольников в конце каждой строки и перпендикулярно к ней. Как вы можете видеть, эти полигоны имеют одинаковый размер, но мы хотим, чтобы их размер был дифференцирован, чтобы их иметь плавное изменение формы в конце.



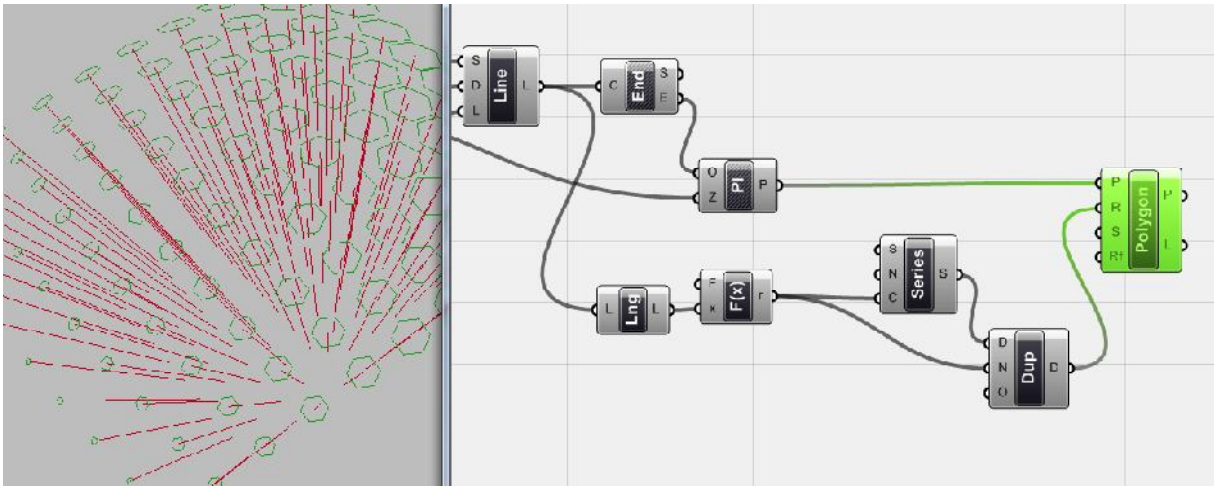


Рис. 71. С помощью компонента <List Length> мы получаем количество линий, а следующий компонент <function>, который находит квадратный корень (F (X) = Sqrt (x)), вычисляет количество линий в каждой строке. Мы использовали компонент <series> с начальным значением и размером шага = 0.1, и числом значений равным числу строк. Так мы создали список постепенно растущих чисел, равный числу полигонов в каждой строке. Для того, чтобы использовать эти значения для всех полигонов, мы продублировали эти данные списка определенное число раз, равное количеству столбцов (здесь равно числу строк) и передали их на вход радиуса полигонов. Как вы можете видеть в модели, в каждой строке, размер полигонов постепенно меняется, и это картина повторяется до последнего.

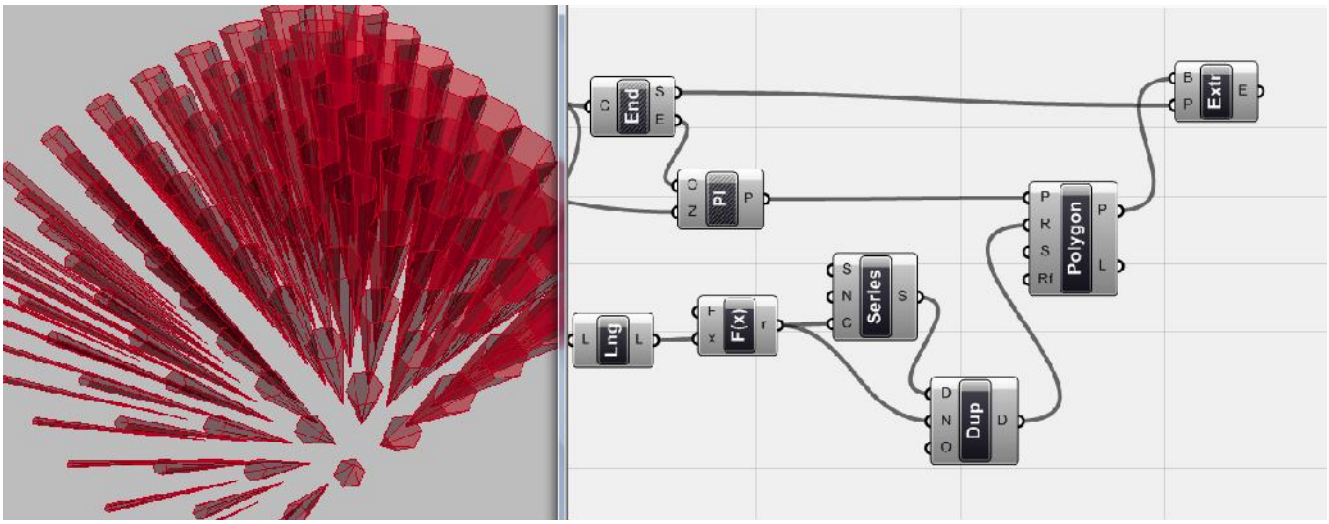
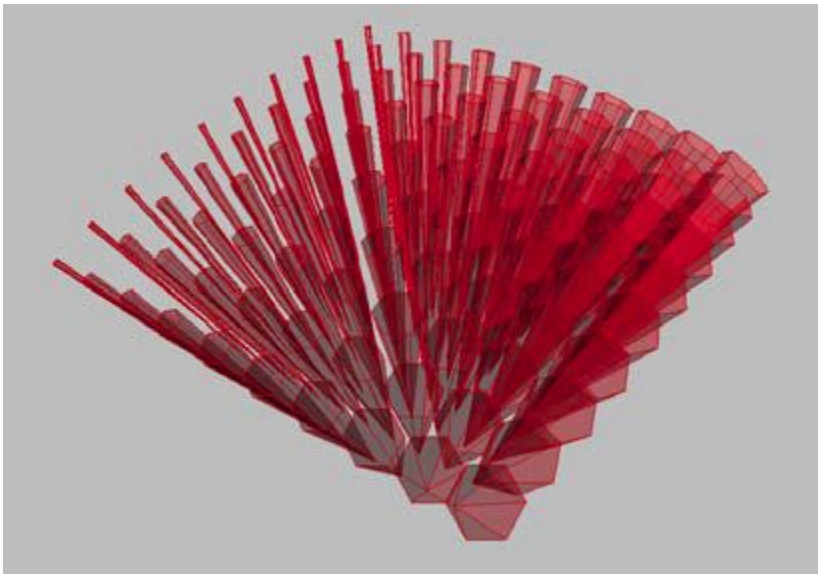


Рис. 72. На последнем шаге, использован компонент <Extrude Point> (Surface> Freeform) и взяты начальные точки линий, как точки, до которых выдавливаются полигоны.



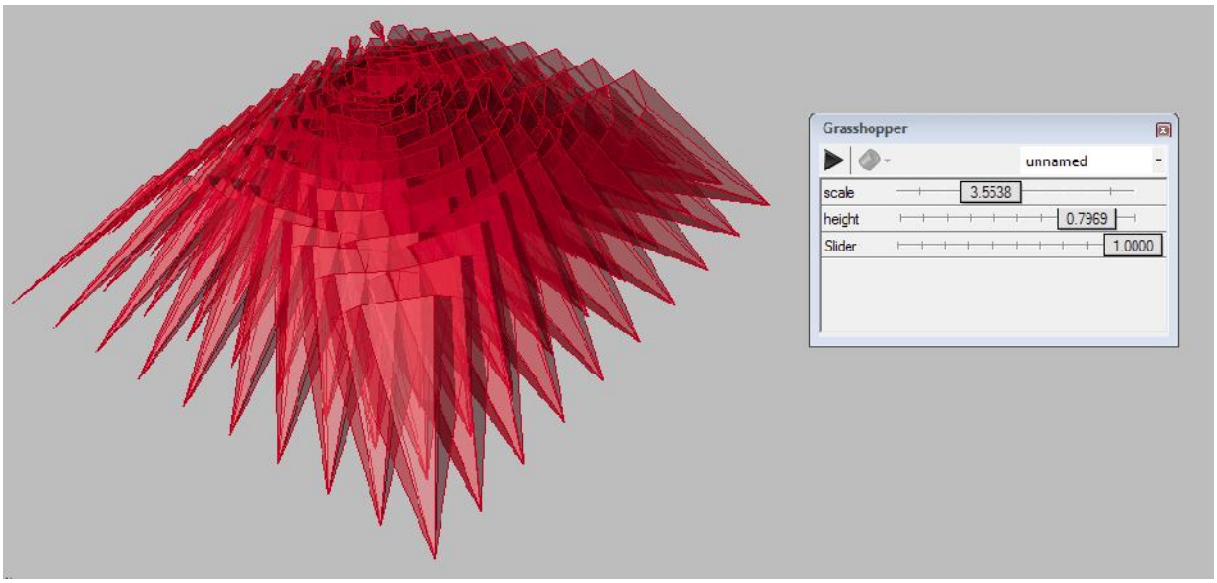
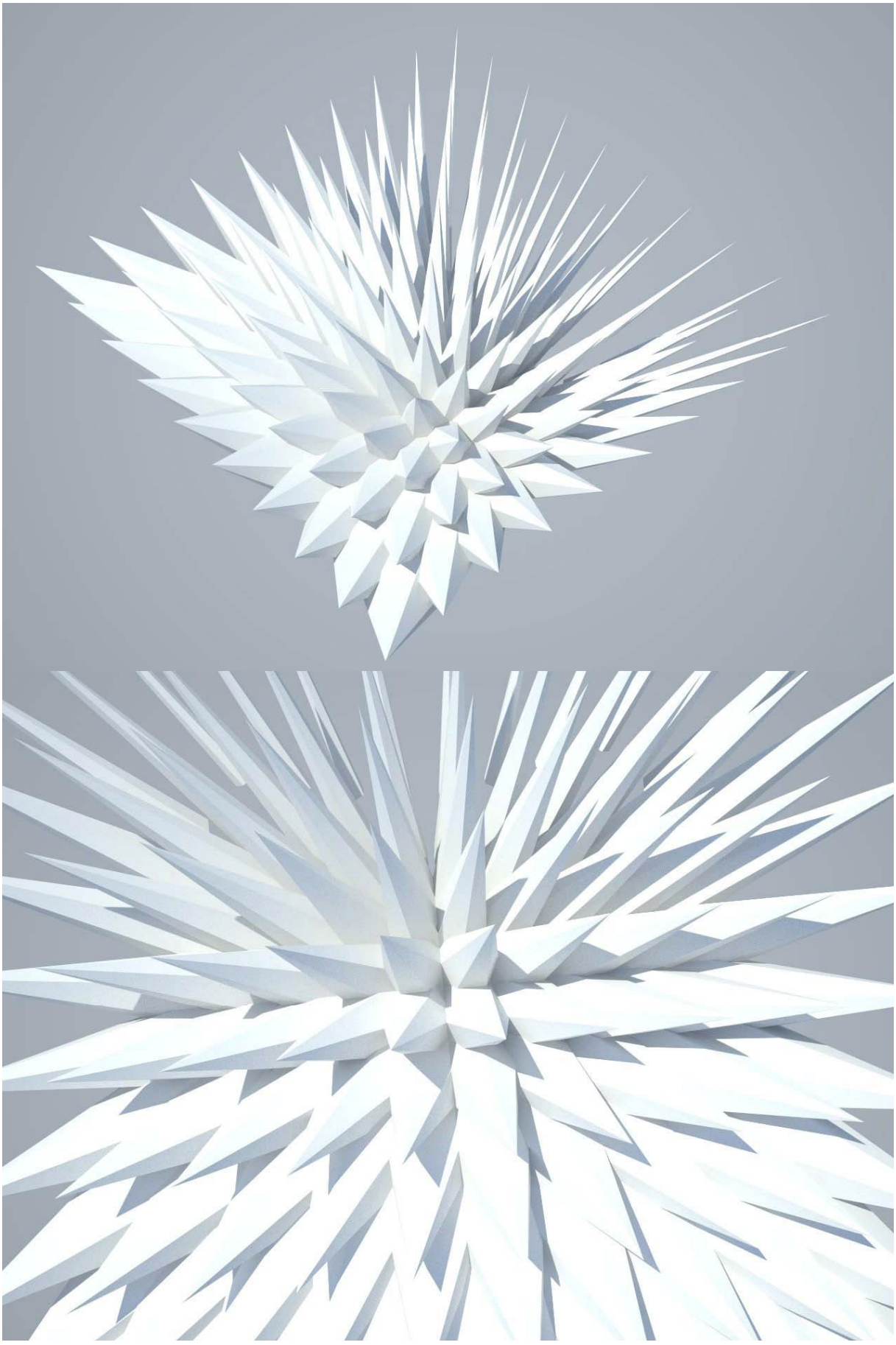


Рис. 73. Теперь с помощью «Remote Control Panel" (Панель дистанционного управления) из меню "View", можно просто изменять значения ползунков, для получения различных вариантов. Можно проверить общий вид модели и выбрать лучший из них. Не забудьте снять флажок "Preview" с ненужных объектов.



4.3. Комбинированный эксперимент: Swiss Re

Сегодня при разработке концепций башен очень распространены ассоциативные методы моделирования. Это позволяет дизайнерам создавать различные модели, просто и быстро. Получая, таким образом, много возможностей изменять дизайн и довольно быстро найти лучшую концепцию. В этом разделе мы будем делать модель башни, и мы думаем, что башня "Swiss Re", спроектированная "Фостер и партнеры" кажется, достаточно сложной для моделирования экспериментов. Сначала посмотрим на проект:

*Рис. 74. Swiss Re HQ, 30 St Mary Axe, London, UK, 1997-2004, Фотографии с сайта «*Фостер и партнеры»*, http://www.fosterandpartners.com.*

Позвольте рассказать вам концепцию. Мы собираемся нарисовать круг, в качестве плана башни и скопировать его, чтобы получить некоторые из этажей, в которых фасад изменяет кривизну. Затем мы будем масштабировать эти окружности в соответствии с формой здания, и после этого с помощью них мы будем делать оболочку башни. Наконец, для структурных элементов фасада, мы добавим секции из полигонов и сделаем эти элементы с помощью них. Чтобы сделать весь процесс более простым и наглядным, мы не будем учитывать размеры здания, и будем учитывать только основные формы модели, т.е. иметь дело с очень простой геометрией.

Давайте начнем с этажей. Мы знаем, что этажи Swiss Re это окружности. Некоторые из них имеют местами V- образные вырезы. Но для упрощения построений, мы используем обычную окружность, чтобы сделать контур башни. Мы хотим расположить эти окружности на определенных высотах, что позволит оперировать пропорциями башни визуально. Как мы уже говорили, эти точки расположены в местах изменения кривизны фасада.

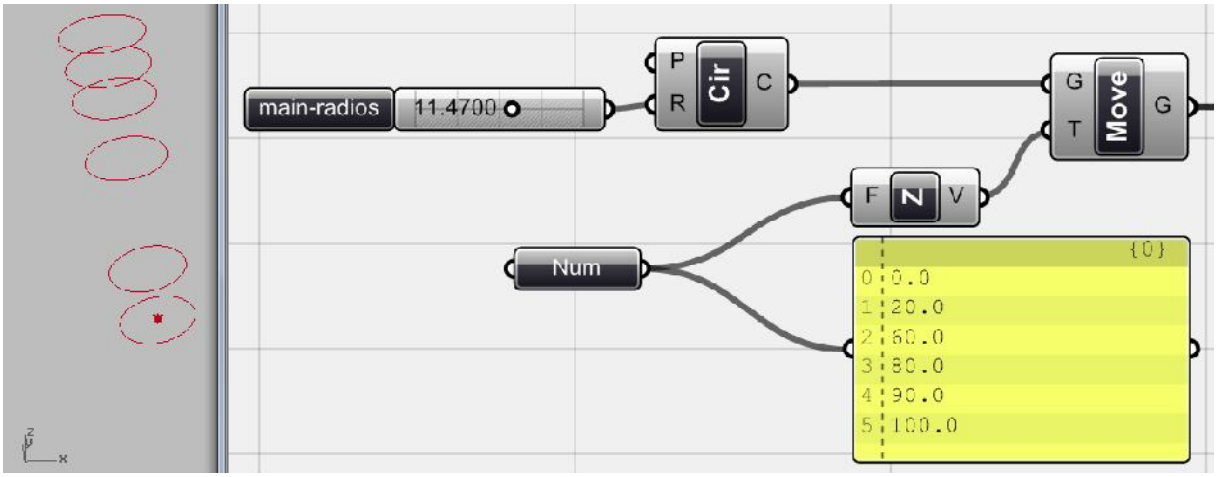


Рис. 75. Компонент <circle> с <number slider> в качестве радиуса, задает контур башни. Эта окружность скопирована 6 раз компонентом <move> вдоль оси Z с помощью вектора <z unit>. Расстояния копирования заданы вручную с помощью компонента <number>, с выбранной в контекстном меню опцией "набор нескольких номеров".

Хотя мы создали эти основные окружности, но все же мы знаем, что все этажи имеют разный размер, поэтому мы должны их масштабировать. Если мы посмотрим на башню, то увидим, что толщина башни сначала увеличивается до определенной высоты. Затем остается практически постоянной, в центральной части башни, а потом становится все меньше и меньше до самой верхней точки башни. Поэтому нам нужно масштабировать эти заготовки этажей, а значит, мы должны задать список масштабных факторов. И здесь мы будем использовать еще один компонент <number> для задания списка масштабных факторов этажей. Вы можете изменить эти числа, чтобы увидеть, как выглядит ваш проект, если вам не нравится оригинальная форма башни.

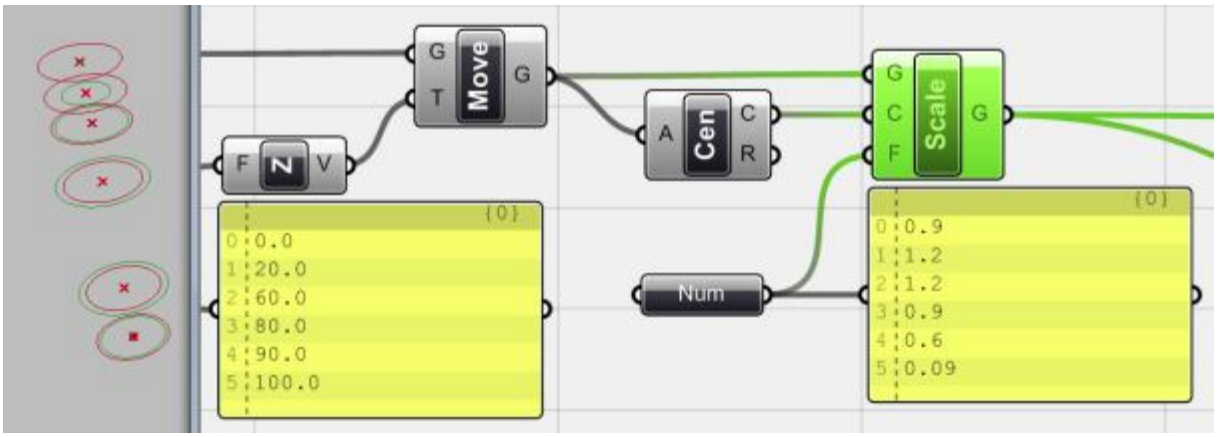


Рис. 76. Нам нужен компонент <scale> (XForm> Affine> Scale), чтобы масштабировать окружности заготовок этажей. Компонент <Scale> выполнят масштабирование геометрии, относительно заданного центра для масштабирования и с масштабным фактором. Поэтому мы должны указать в качестве геометрии наши этажи или окружности, которые представляет собой выход компонента <move>. В качестве центров масштабирования использованы центральные точки окружностей. Это нужно для того, чтобы масштабирование происходило для каждого этажа в своей плоскости (по 2-м координатам). Иначе окружности будут смещаться по вертикали. Поэтому мы

использовали компонент <Center> (Curve > Analysis > Center), который дает нам центры окружностей. Подключив его к <scale> вы можете видеть, что все окружности были масштабированы на своих отметках и без смещения по вертикали.

Еще раз отметим, что масштабные факторы, которые мы задали раньше, могут быть изменены, чтобы увидеть, какая комбинация лучше всего подходит для общего вида. Они все расположены в одном компоненте <number>.

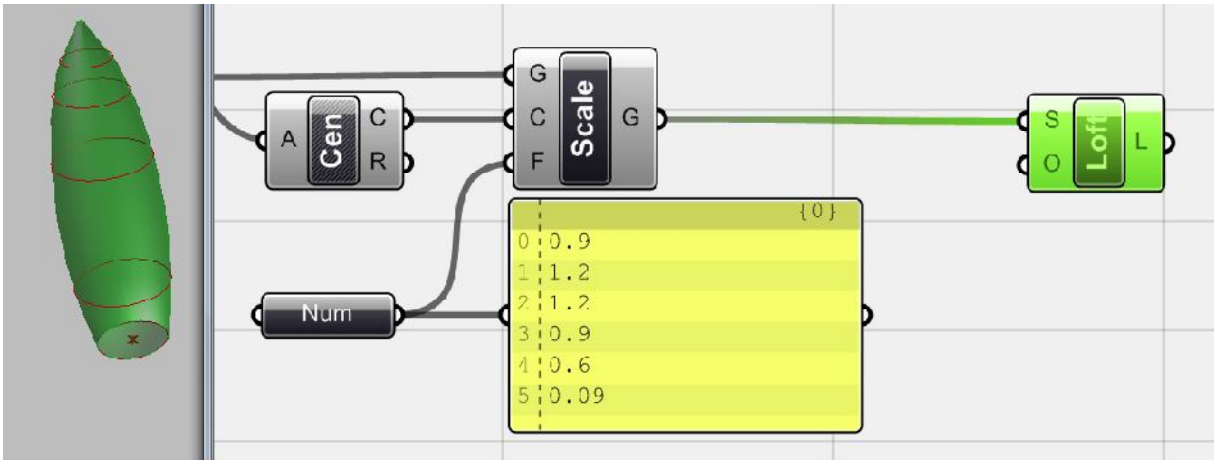


Рис. 77. Теперь, если мы построим поверхность с помощью компонента <loft> (Surface> Freeform> loft), то появится первое изображение башни. Чтобы очистить сцену от созданных ранее точек и кривых, нужно снять переключатели preview у всех лишних компонентов.

Давайте перейдем к фасадным элементам.

Структурными элементами фасада являются линии винтовой формы, которые имеют сечение, похожее на два связанных треугольника. Но опять же, чтобы сделать его простым, мы просто смоделируем только видимые ее части, которые в плане выглядят почти как треугольник. Для придания им объема, нам опять пригодится компонент <loft>.

Мы хотим создать эти треугольные сечения на фасаде. Чтобы сделать это, сначала нам нужно найти положение этих треугольников на фасаде. Если мы сможем генерировать кривую на поверхности фасада и разделить ее, то это было бы подходящим местом для размещения всех треугольников, перед выполнением каких либо преобразований.

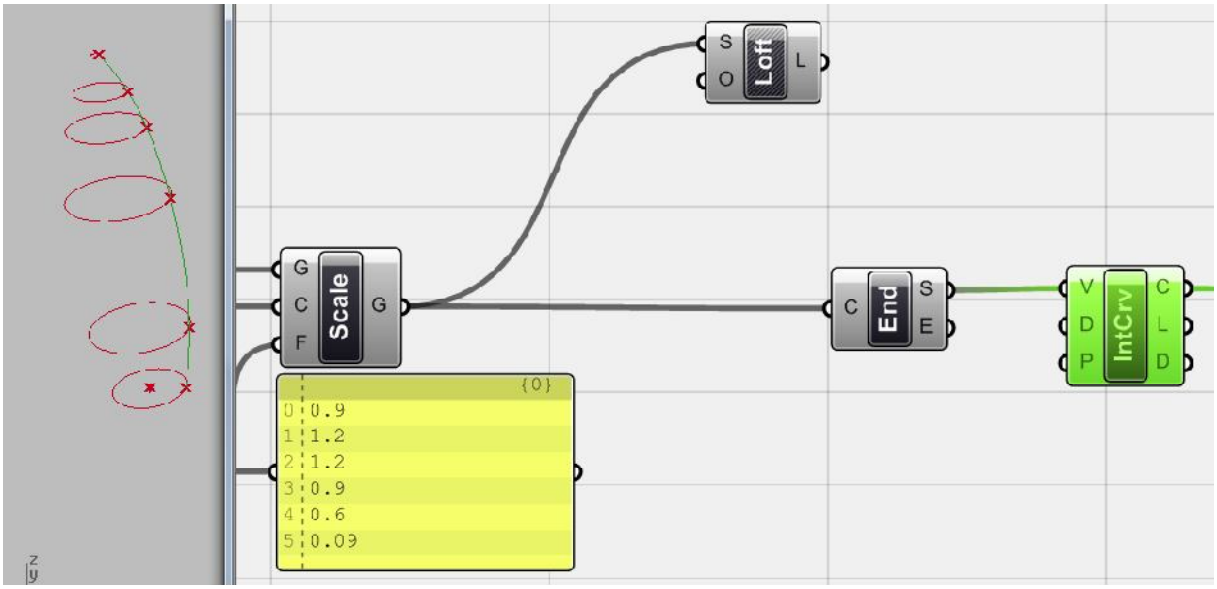


Рис. 78. Мы использовали <End points> компонент, чтобы получить начальные / конечные точки окружностей этажей. Соединив эти точки, как вершины компонента <interpolate> (curve > spline > interpolate) получаем кривую, которая расположена на фасаде.

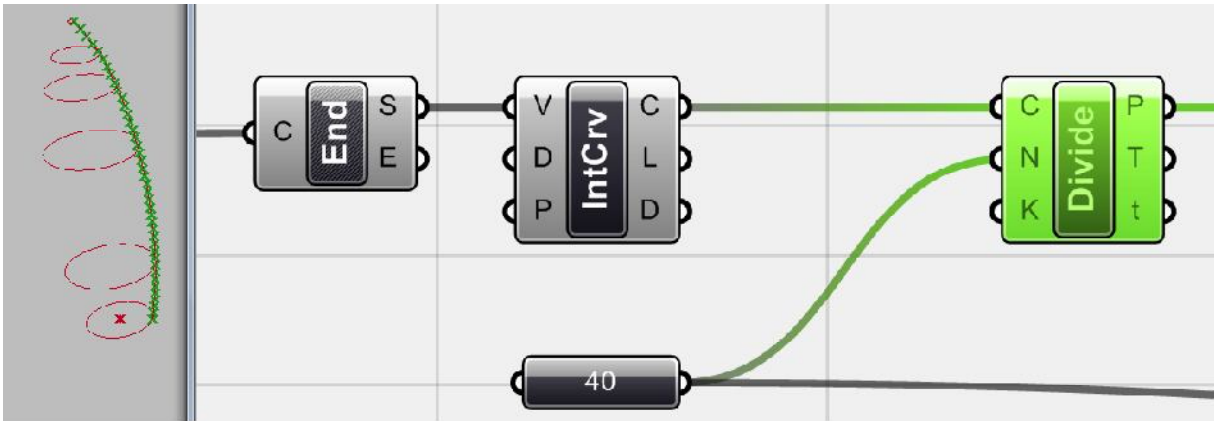


Рис. 79. Здесь мы разделили кривую на 40 частей. Количество делений помогает настроить гладкость элемента на фасаде, если это потребуется.

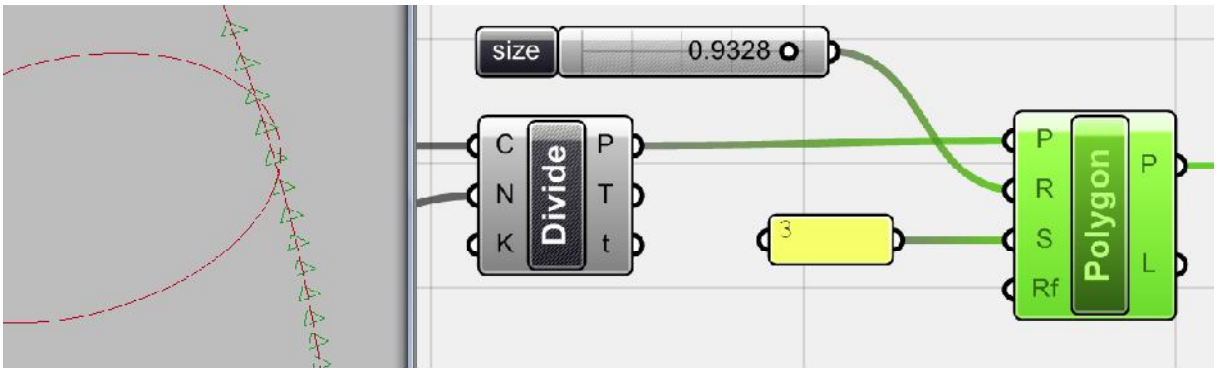


Рис. 80. Теперь точки деления стали базой для создания <polygon> на фасаде. Мы задали число сторон "sides" равным 3, для создания треугольников. Размер элементов 'R', контролируется с помощью <number slider>.

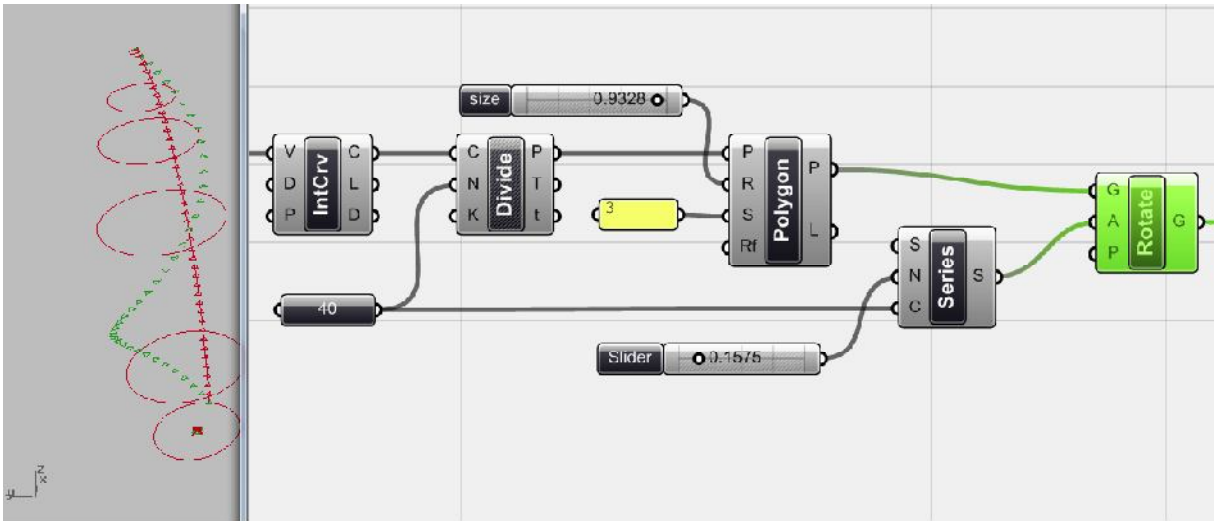


Рис. 81. Структурные элементы фасада, представляют собой спирали и повернуты вокруг оболочки здания до верхней точки башни. Для достижения этой цели, последовательно поворачиваем все треугольные секции. Мы использовали компонент <Rotate> и для этого нам необходимо задать углы поворота. Как уже было сказано, углы поворота должны представлять список постепенно возрастающих чисел. Компонент <series> порождает наши углы поворота и имеет сколько же элементов (точек треугольников), как и компонент <divide>. Как результат, все треугольные секции вращаются вокруг фасада.

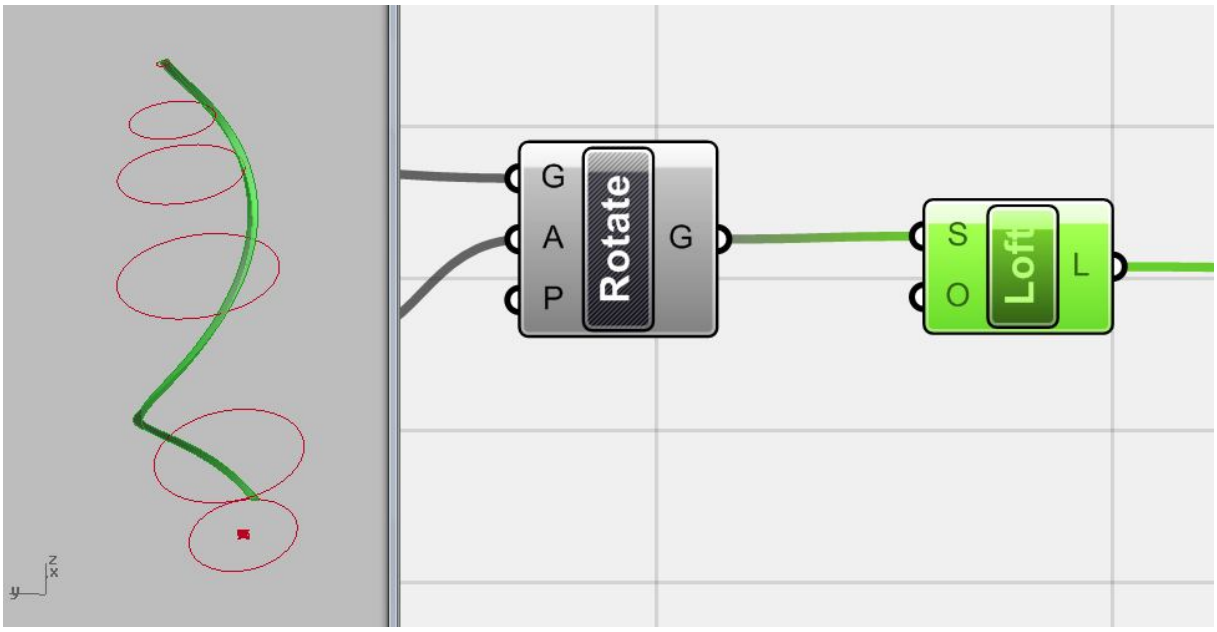


Рис. 82. Теперь, если мы все треугольные секции передадим компоненту <loft>, то появится один элемент фасада. Угол поворота и размер элемента управляемы, поэтом мы можем подобрать форму, чтобы она наилучшим образом соответствовала фасаду.

Домены

Как мы уже упоминали, домены (или интервалы) - это числовые диапазоны. Это вещественные числа от нижней границы до верхней границы. Так как мы сказали: "действительные числа", то это означает, что мы имеем бесконечно число значений между двумя границами интервала. Это значит, что нам нужны различные варианты использования этих численных областей. Как мы экспериментировали раньше, мы можем разделить числовой диапазон и получить

последовательность равномерно распределенных чисел между двумя границами.
Здесь мы хотим распределять фасадные элементы вокруг основной окружности. Для этого нам нужен интервал для покрытия всей окружности основания.

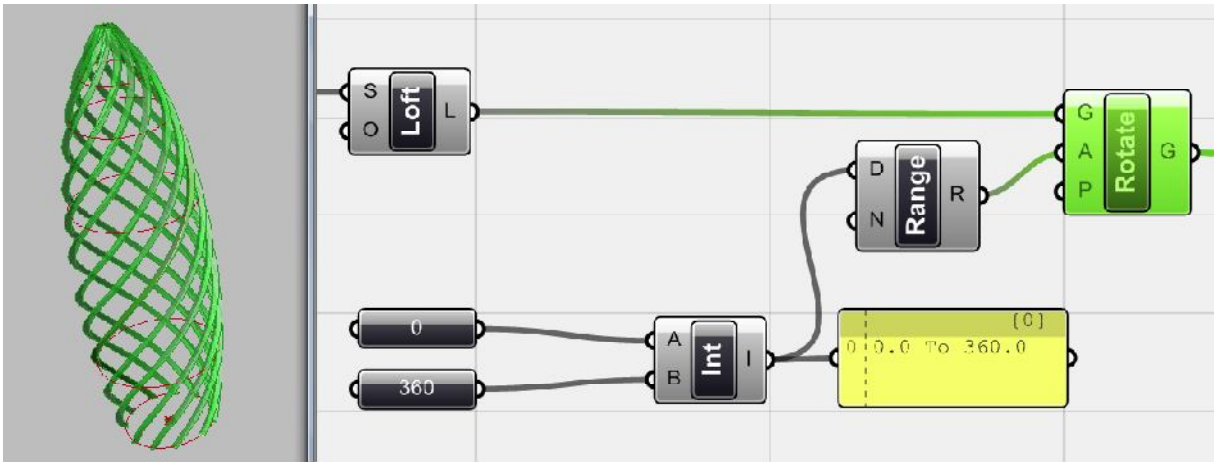


Рис. 83. Компонент <domain> (в предыдущих версиях <interval>) (Scalar > domain > domain) используется для определения численного значения от 0 до 360. Этот числовой диапазон делится в компоненте <range> на 10 частей, и результат используется как угол поворота в компоненте <rotate>. Как показано на изображении, фасадные элементы распределены по всему контуру основной окружности.

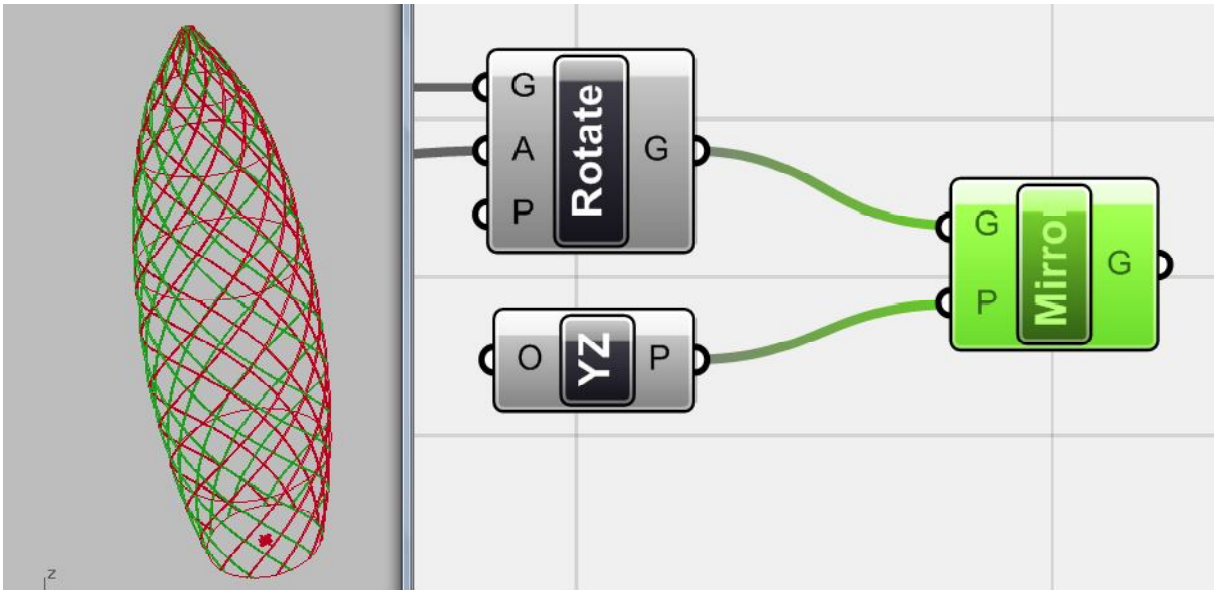


Рис. 84. Теперь, мы отразим с помощью <mirror> (XForm> Euclidian > Mirror) нашу геометрию относительно плоскости <YZ plane> (Vector > Constants > YZ plane). Итак, в конце мы имеем геометрию решетки вокруг формы башни.

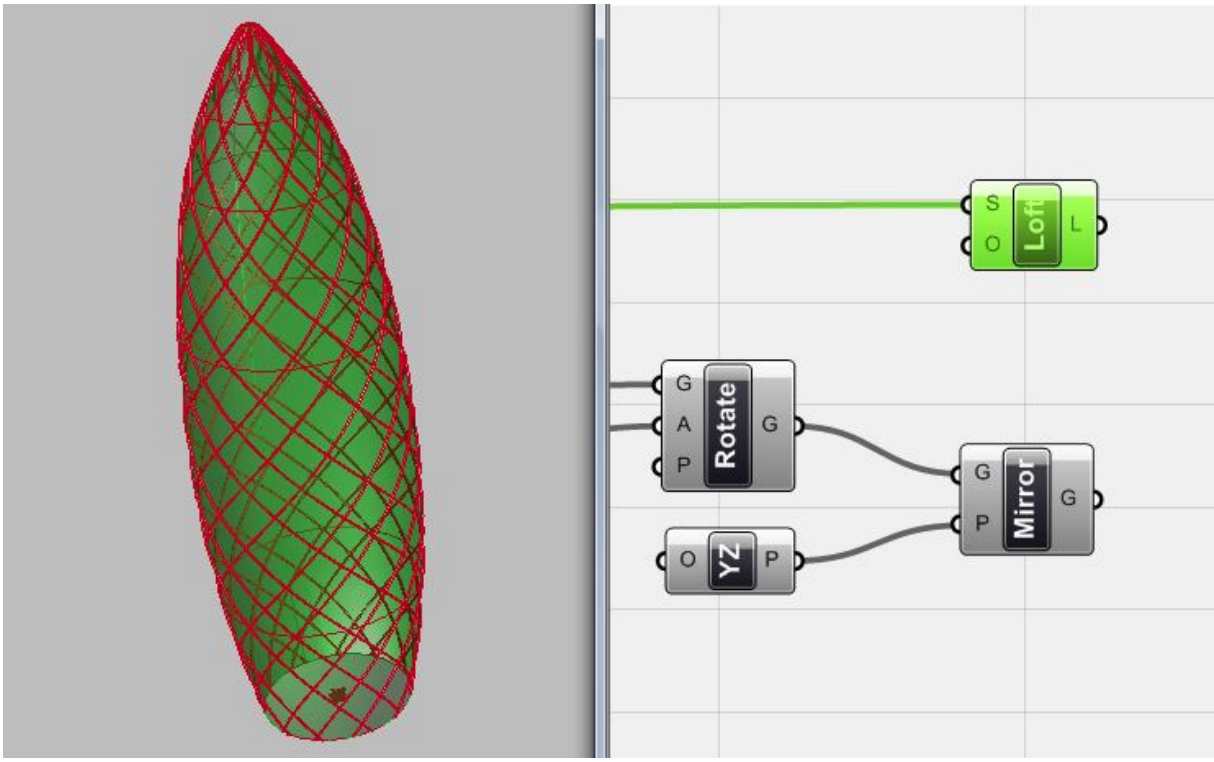


Рис. 85. Снова включим предварительный просмотр фасад и у нас теперь есть грубые представления о том как сделать "Swiss Re" используя ассоциативные техники.

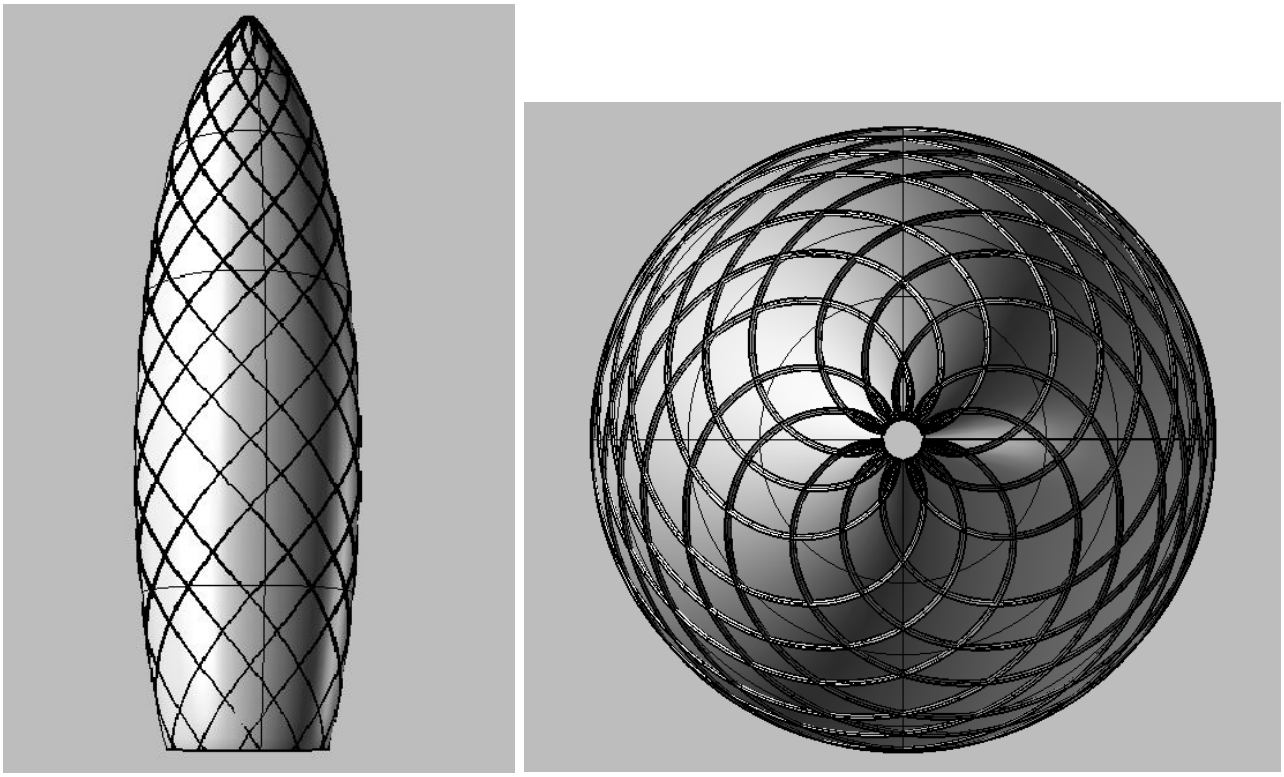


Рис. 86. Для создания геометрии в Rhino, выберите те компоненты, которые желаете превратить в геометрию сцены, и выберите ‘Bake Selected Objects’ с панели инструментов холста или контекстном меню компонента.

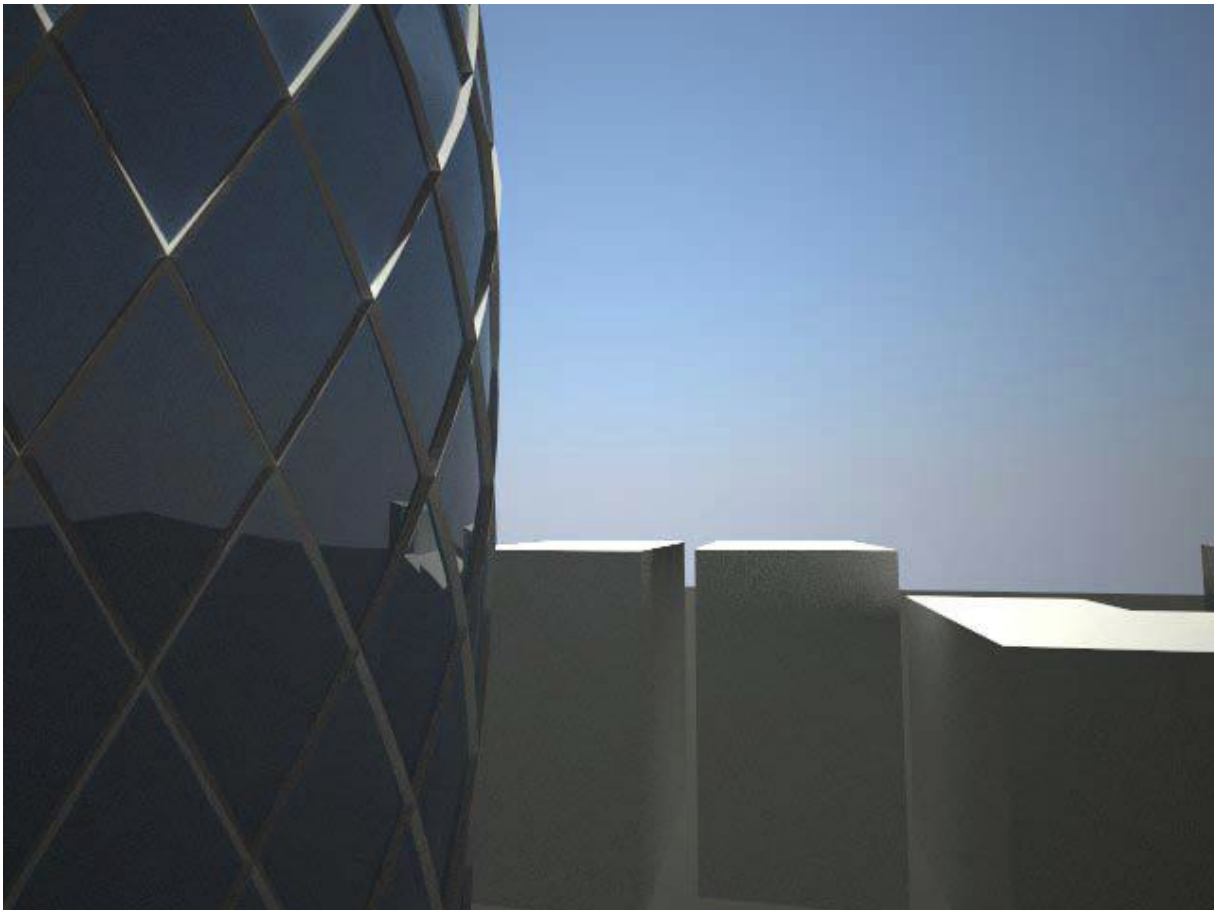


Рис. 87. Окончательная модель. Хотя она не совсем точно соответствует оригиналу, но может служить в качестве эскиза модели созданной в короткий срок.

4.4. Аттракторы

"Аттрактор множество состояний динамической физической системы по отношению к которым, система стремится развиваться, независимо от начальных условий системы. **Точечный аттрактор** - аттрактор, состоящий из одного состояния. Например, шар, катящийся по поверхности гладкой, округлой чаши всегда остановится в самой низкой точке, в центре нижней части чаши; конечное состояние расположения и неподвижности является точечным аттрактором " (Dictionary.com/Science Dictionary)

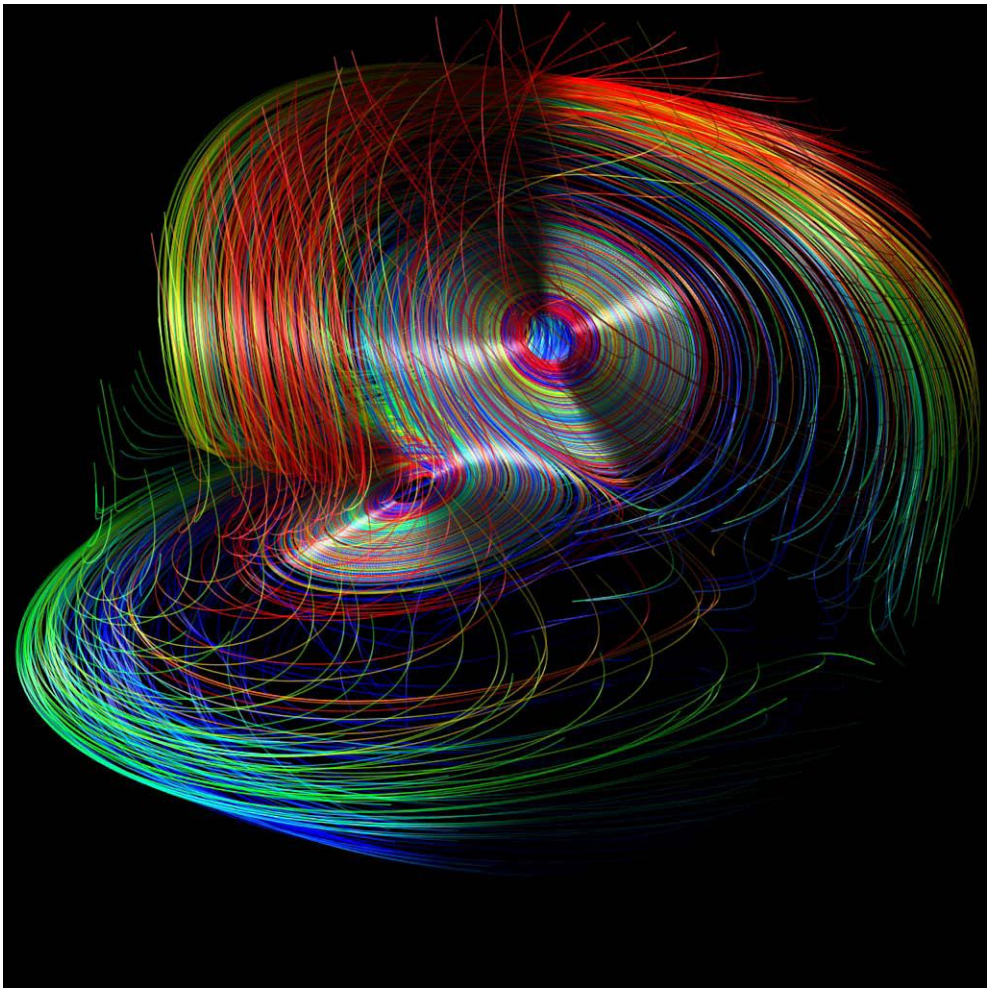


Рис. 88. Странный аттрактор (Иллюстрация с сайта: http://www.cs.swan.ac.uk/~cstony/research/star/)

В случае дизайна и геометрии, аттракторы - это элементы (обычно точки, но могут быть кривые или другая геометрия), которые оказывают воздействие на другие объекты пространства, изменяя их поведение, смещая, ориентируя или масштабируя их и т.д.

Они могут формировать пространство вокруг себя и создавать силовые поля с конкретным радиусом воздействия. Аттракторы имеют различные приложения в параметрическом проектировании, так как они имеют потенциал постоянного изменения всех объектов дизайна. Определив силовое поле, аттракторы могут также влиять на несколько систем агентов и выполнять несколько действий. То, как аттракторы они могут повлиять на продукт, а также сила

их воздействия является регулируемым. Мы рассмотрим понятие аттракторов в различных случаях, так что давайте перейдем к некоторым очень простым экспериментам.

Точечные аттракторы

У нас есть сетка точек, и мы хотим создать набор полигонов на них. У нас также есть точка, которую мы назвали <attractor_1> и мы нарисовали вокруг нее окружность <circle>, чтобы ее было лучше видно. Мы хотим, чтобы <attractor_1> влиял на все наши многоугольники <polygon> в области своего действия. Это означает, что в зависимости от расстояния между каждым <polygon> и <atractor_1>, и от области воздействия <attractor_1>, каждый <polygon> реагирует на аттрактор изменением своего размера.

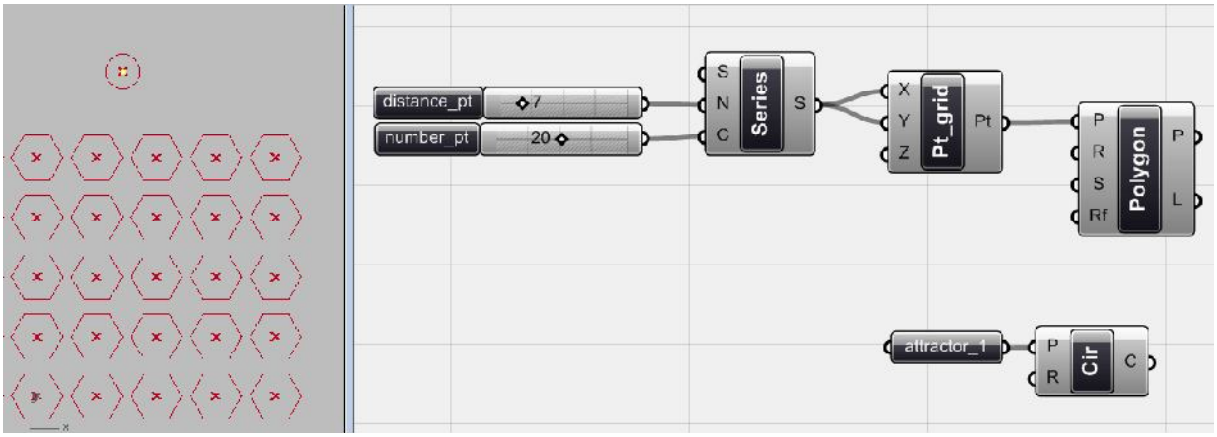


Рис. 89. Базовая сеть точек <point_grid>, полигоны <polygon> и аттрактор <attractor_1>.

Алгоритм очень прост. На основании расстояния <distance> между <attractor_1> и точками сетки <Pt-grid>, мы хотим изменять радиус полигонов <polygon>. Таким образом "отношение" между аттрактором и полигонами определяется их расстоянием. Нам нужен компонент <distance> для измерения расстояния между <attractor_1> и центрами многоугольников или <pt_grid>. Поскольку это число может быть довольно большим, нам нужно разделить <divide> (Math > Operators > Division) это расстояние на заданное число <number slider> для уменьшения силы <attractor_1> так, как мы хотим.

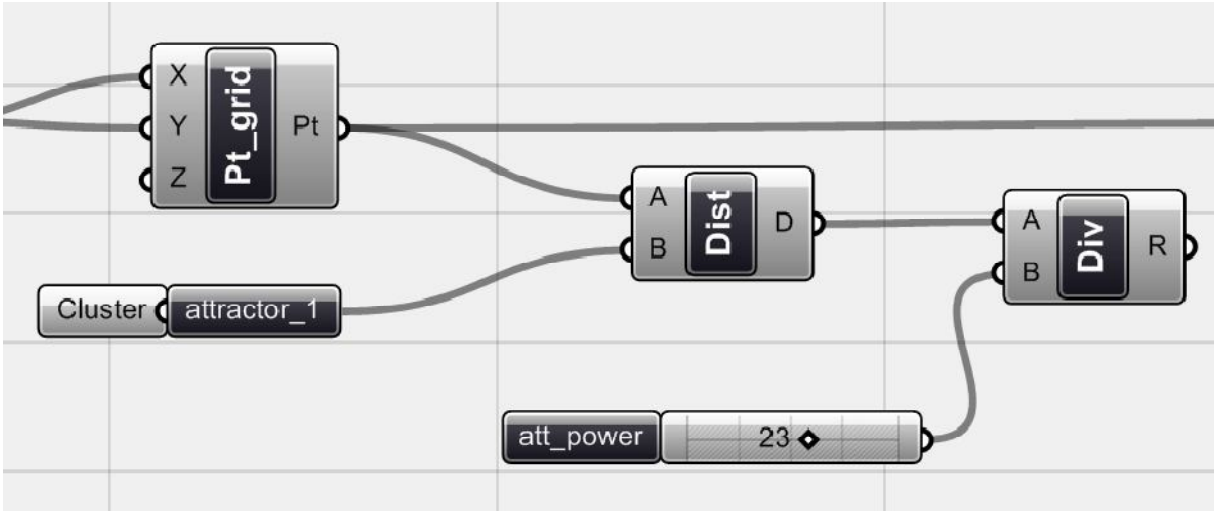


Рис. 90. Расстояние <Distance> делится на число для того, чтобы контролировать силу аттрактора<attractor_1>.

Теперь, если вы подключите компонент к радиусу (R) полигона <polygon>, то вы можете наблюдать, что размеры полигонов увеличиваются, с увеличением расстояния до аттрактора <attractor_1>. Хотя это может быть хорошо для первого раза, но мы должны также контролировать максимальный радиус полигонов, в противном случае, если они располагаются дальше и дальше от аттрактора, то становятся слишком большими и могут пересекаться друг с другом (это происходит и в том случае, если сила аттрактора слишком высока). Поэтому для контроля максимального значения радиуса, зададим его допустимое значение вручную.

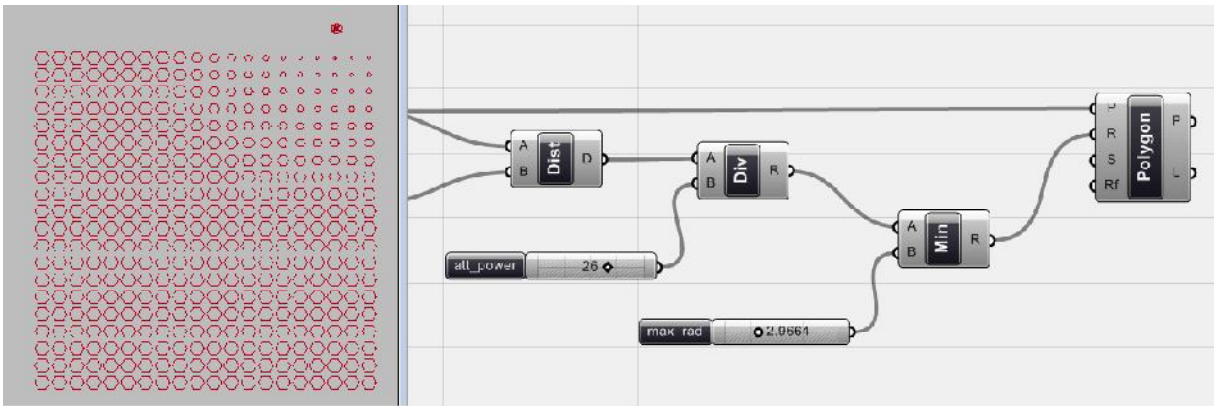


Рис. 91. Используя компонент <minimum> (Math> Util> Mimimum) и заданное пользователем число, мы заставляем алгоритм выбрать значение из компонента , если он меньше, чем максимальный радиус, задаваемый с помощью <number slider>. Как вы можете видеть на изображении, полигоны, которые больше, чем <max_rad> остаются постоянными, и мы можем буквально говорить, что они располагаются вне области воздействия аттрактора.

Теперь, если вы вручную измените положение <attractor_1> в окне Rhino, то можете увидеть, что все полигоны получают радиус в соответствии с позицией <attractor_1>.

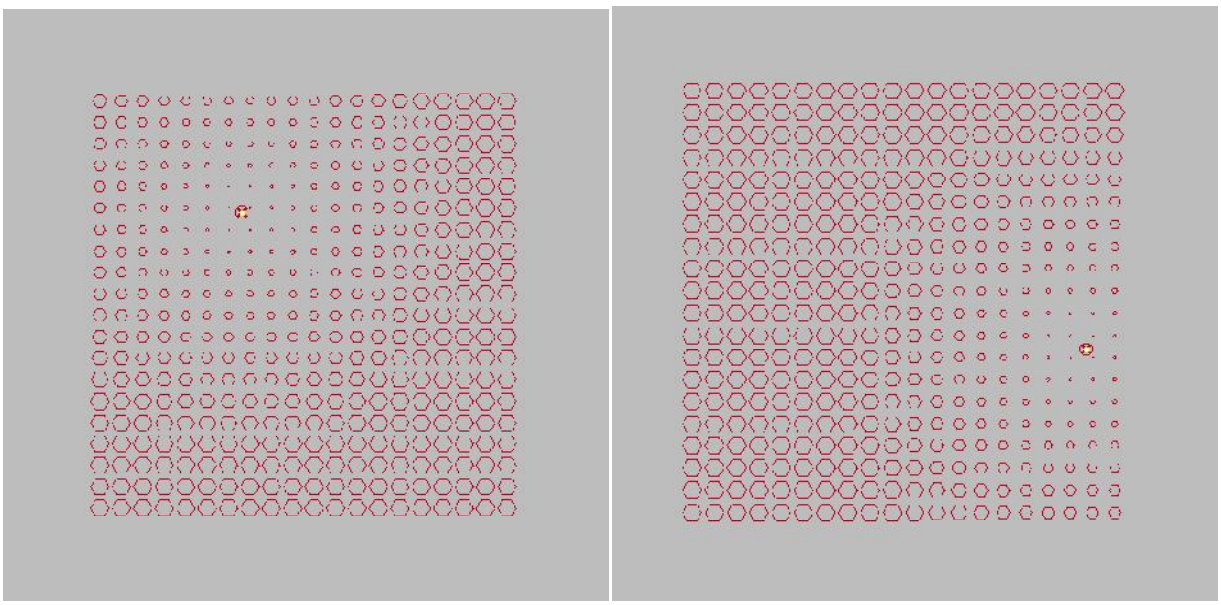


Рис. 92. Влияние <attractor_1> на все полигоны. Смещение аттрактора, соответствующим образом влияет на все полигоны.

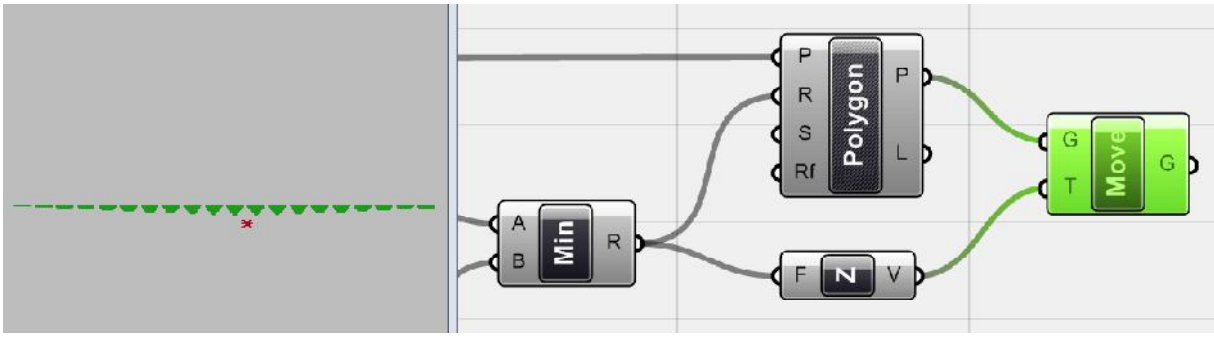


Рис. 93. Использую ту же самую идею, мы можем сместить полигоны в направлении оси Z, на основании значений компонента <Min> или изменяя их с помощью математических функций, при необходимости. Так что использование аттракторов не ограничивается только изменением размеров!

Просто !!!!!!! Мы можем задавать любые другие функции для этих полигонов, например, вращать, изменять цвета и т. д. Но давайте подумаем, что произойдет, если у нас будет поле от двух аттракторов. Мы делаем еще один кластер, т.е. еще одну точку в Rhino, и связанные с <Point> и <circle> в Grasshopper.
Первая часть алгоритма будет такой же. Мы снова должны измерить расстояние между центрами полигонов или <pt_grid> и <attractor_2>, а затем разделить его на той же <number slider> (переименованный в <att_power>), чтобы контролировать влияние аттрактора.

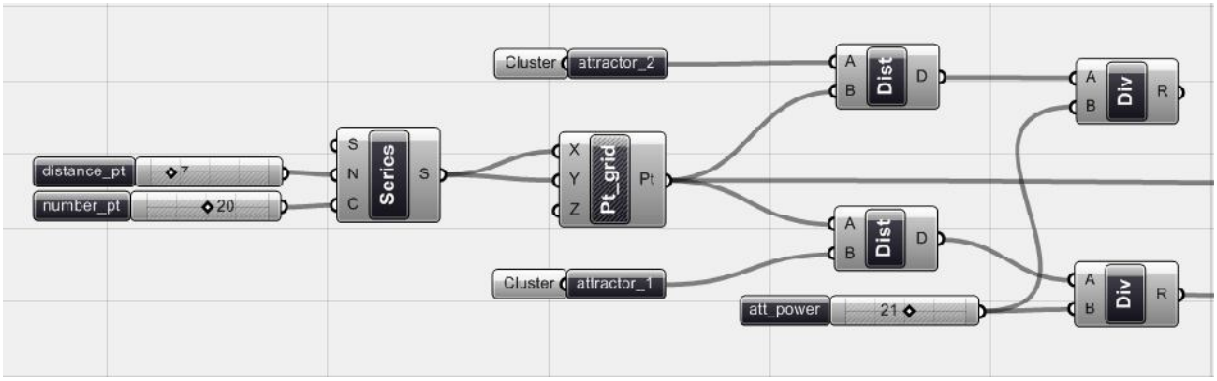


Рис. 94. Добавление второго аттрактора <attractor_2> и применение того же самого алгоритма к нему.

Теперь у нас есть два разных списка данных, которые содержат расстояния от полигонов до каждого из аттракторов. Так как ближайший аттрактор будет оказывать на полигон большее воздействие, то мы должны найти, какой из них ближе и использовать его как источник воздействия. Мы будем использовать компонент <min> чтобы выяснить, какое из расстояний является минимальным или какая точка ближе.

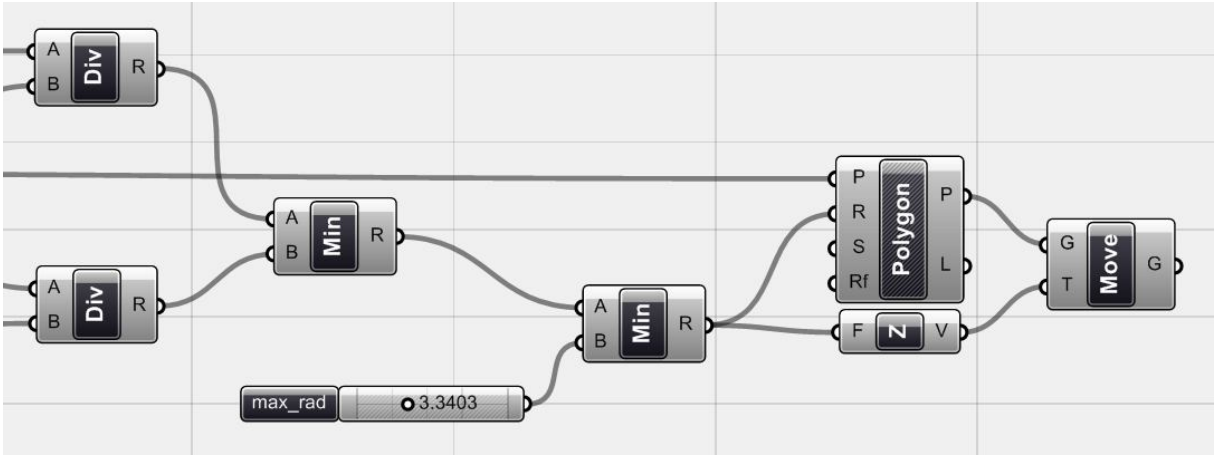


Рис. 95. Поиск ближайшего аттрактора. После нахождения ближайшего, с помощью компонента <min>, остальной процесс остается тем же самым. Теперь все полигоны <polygon> подвержены воздействию двух аттракторов.

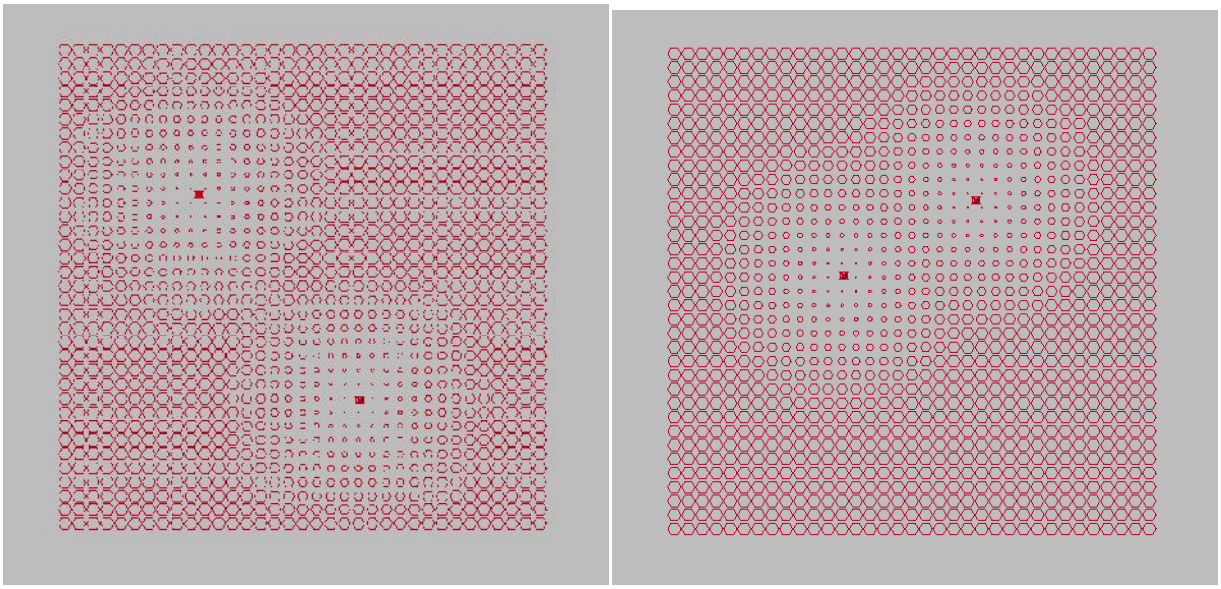


Рис. 96. Again you can change the position of the attractors and see how all polygons reacting accordingly.

Мы можем добавлять больше и больше аттракторов. Идея состоит в том, чтобы найти аттрактор, который ближе для каждого полигона и применять предопределенный эффект. Это понятие является полезным для решения вопросов дизайна с огромным количеством мелких элементов.

Линейные аттракторы: Проект стена

Давайте в завершение этого обсуждения рассмотрим другой пример, но на этот раз с аттрактором в виде кривой. Потому что во многих случаях необходимо задавать ваше поле объектов линейными аттракторами вместо точек.
Наша цель здесь заключается в разработке стенки для интерьера, имеющей несколько

сквозных прямоугольных вырезов. Такая стенка может быть вырезана из листовых материалов. Мы имеем плоский лист (стена), две кривые и кучу случайно распределенных точек, в качестве базисных точек резки формы. Мы решили создать прямоугольники по этими точками и вырезать их из листа, чтобы сделать эту стенку. Мы также хотим, организовать эти прямоугольники по двум данным кривым так, чтобы в конечном итоге, они представляли не просто некоторое количество рассеянных прямоугольников, а случайно распределялись по этим кривым. Чтобы можно было управлять степенью разброса прямоугольников на макро уровне и контролировать их случайный размер в микро-масштабе.
Что нам нужно, для создания этой группы случайных точек и смещения их в направлении кривых, так это буквальном смысле величина силы притяжения, которую они получают от кривых. Поскольку мы также решили сместить точки на обеих кривых, то нам нужно выбрать ту кривую, которая ближе к точке. И когда, мы будем создавать наши прямоугольники по этими точками, то будем определять размер этих прямоугольников в зависимости от их расстояния до аттракторов.

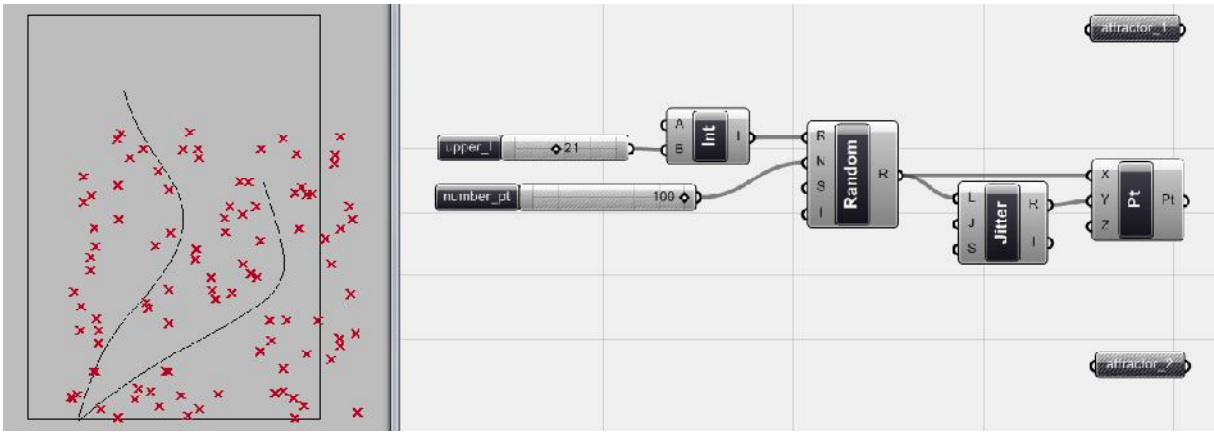


Рис. 97. Создание списка случайно распределенных точек <Point> и внедрение аттракторов двумя компонентами <curve> (Params> Geometry> Curve) в пространстве листа. Мы использовали компонент <interval> для определения числового интервала от 0 (задано вручную) и до <number slider> для диапазона случайных точек (в новых версиях вы должны использовать компонент <Domain>, как упоминалось выше). Компонент <Pt> переименован в <Rnd_Pt_Grid>.

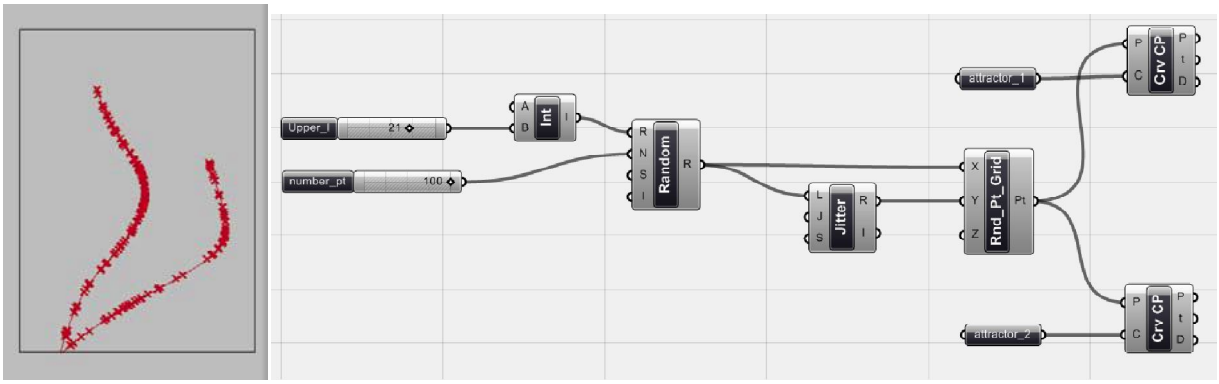


Рис. 98. Когда аттрактор точечный, то вы можете просто перемещать вашу геометрию к нему. Но когда аттрактор линейный, то вам нужно будет найти относительные точки на кривой и смещать вашу геометрию к этим конкретным точкам. И этот момент должен быть уникальным для каждой геометрии, потому что здесь должна быть учтена связь

между формой аттрактора и любой геометрией в области его воздействия. Если представить себе аттрактор, как магнит, то он должен тянуть геометрию от ближайшей точки на объект. Поэтому в первую очередь необходимо найти ближайшие точки для <Rnd_pt_grid> на обоих аттракторах. Эти точки, будут ближайшими точками на аттракторах для каждого члена <Rnd_Pt_Grid> в отдельности. Мы использовали компонент <Curve CP> (Curve > Analyse> Curve CP), который дает нам ближайшую точку на кривой для нашей <Rnd_Pt_Grid>.

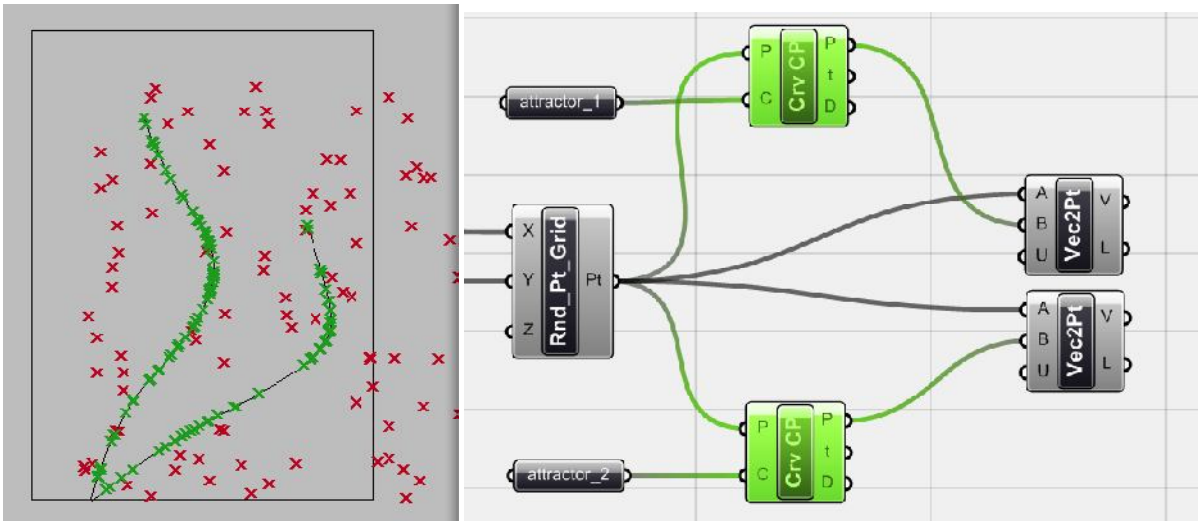


Рис. 99. Для смещения точек в направлении аттракторов, нам нужно определить вектор для каждой точки в <Rnd_Pt_Grid>, от точки к ее ближайшей точке на аттракторах Поскольку у нас есть начальная и конечная точки вектора, то мы используем компонент <vector 2Pt>. Вторая точка вектора (вход B), это ближайшая точка на кривой.

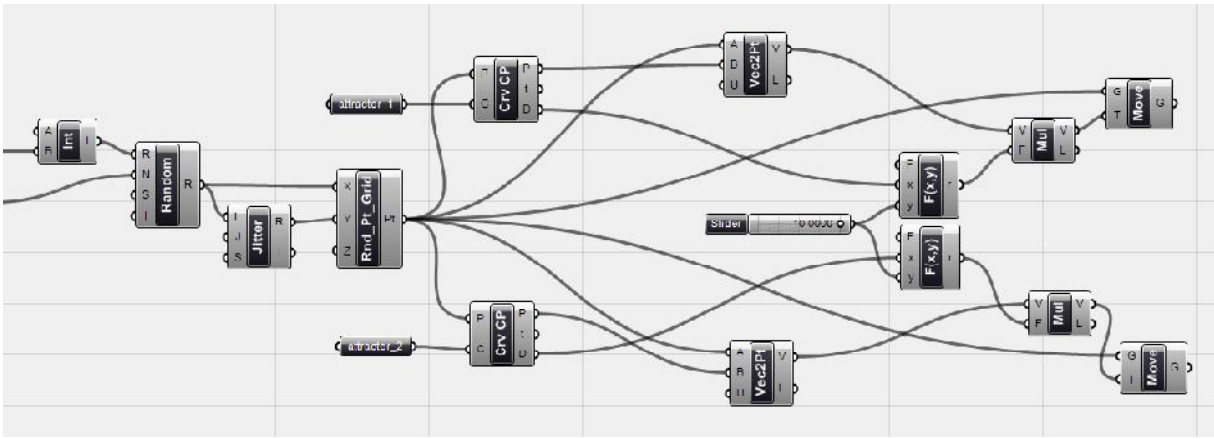


Рис. 100. Теперь мы подключили все <Rnd_Pt_Grid> к двум компонентам <move> для смещения их к аттракторам. Но если мы используем вектор, который мы создали на последнем шаге, то он переместит все точки на кривые. Но это не то, что нам нужно. Мы хотим, чтобы точки смещались в зависимости от их расстояния до аттракторов. Если вы посмотрите на компонент <Curve CP>, он имеет выходы, которые дают нам расстояние от каждой точки, и соответствующую ближайшую точку на кривой. Хорошо! Нам не нужно измерять расстояние от другого компонента. Мы просто использовали компонент <FUNCTION> и подключили расстояние к X, а <number slider> к Y и разделили: X / Log (Y) для контроля множителей перемещения (Функция Log задает линейную зависимости между расстоянием и результирующим фактором).

Теперь мне нужно изменить размер наших векторов на основе этих вновь созданных множителей. Здесь нам нужет компонент для изменения размер вектора. Именно поэтому мы использовали компонент <multiply> (Vector> Vector > Multiply), который сделает это за нас. Поэтому мы подключили <vector 2P> как базисные векторы, и изменили их размер с помощью множителей, а результирующие векторы подключили к компонентам <move>, которые смещают <Rnd_Pt_Grid> в зависимости от их расстояния до аттракторов, в направлении кривых.

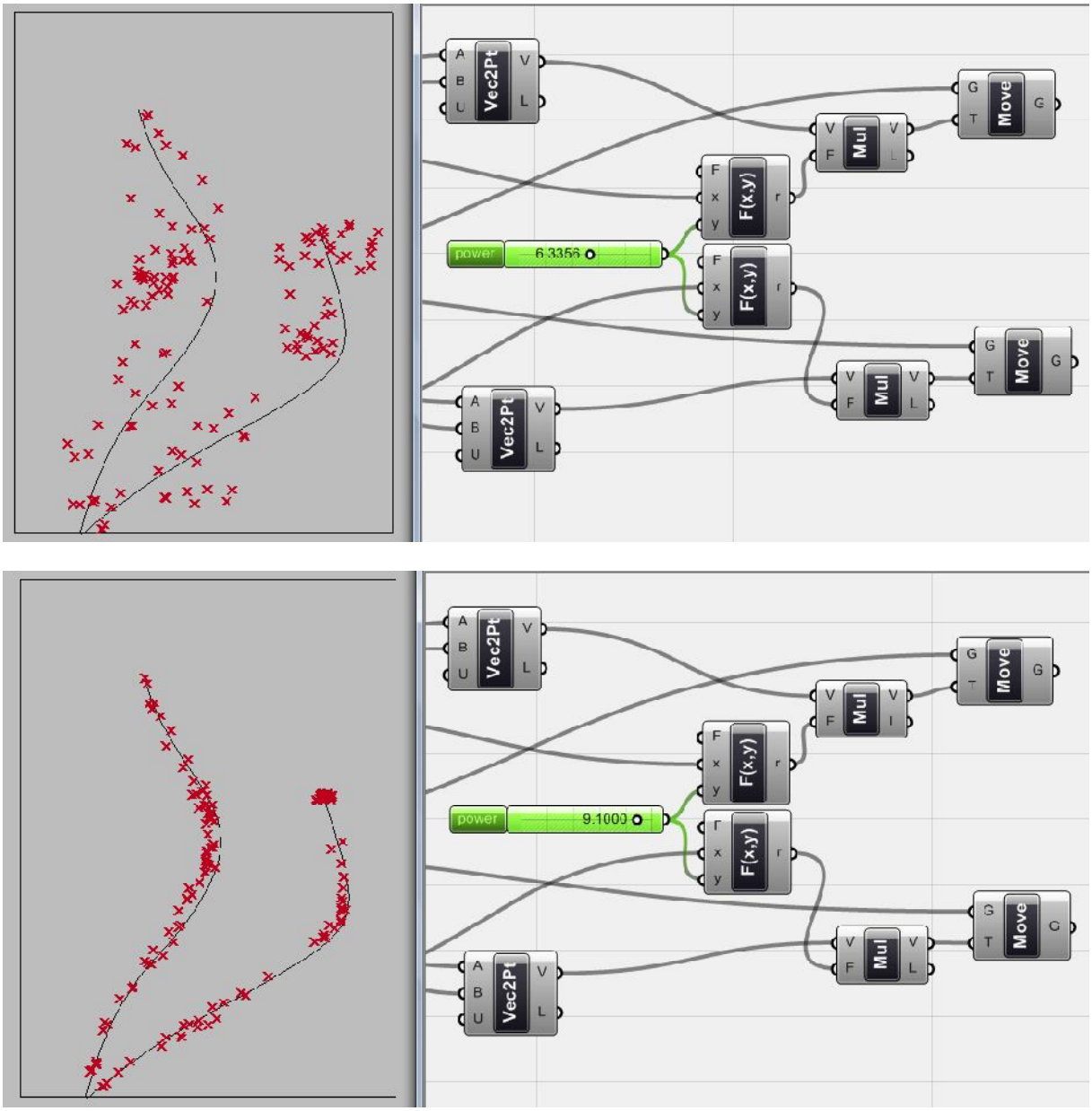


Рис. 101. <Number slider> изменяем силу, с которой аттракторы смещают объекты по отношению к себе.

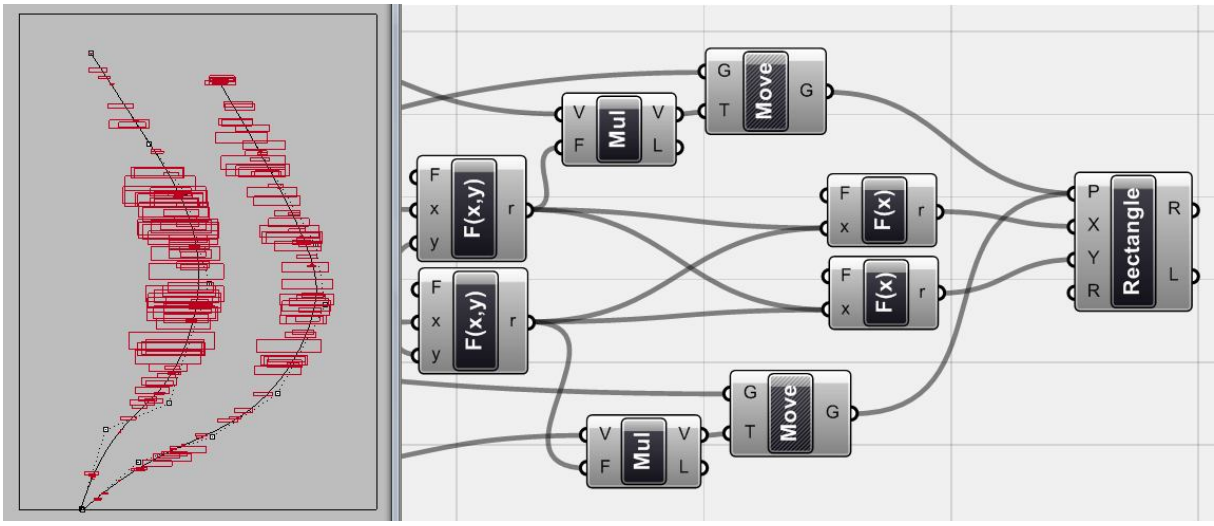


*Рис. 102. Следующим шагом является генерация прямоугольников. Мы использовали компонент <rectangle> (*curve>primitive*) и подключили смещенные с помощью <move> точки, в качестве базисных. Но, как говорилось выше, мы также хотим изменить размер <rectangle> с учетом их расстояния до каждого аттрактора. Используем те же численные значения, которые мы использовали в качестве множителя векторов, и изменим их двумя функциями. Мы разделили эти факторы на 5 для значения X прямоугольников и на 25 для значения Y. Как вы можете видеть, прямоугольники имеют различные размеры, основанные на их оригинальных расстояниях от аттракторов, но все они имеют одинаковое соотношение сторон из-за вышеперечисленных делителей. Вы можете изменить эти делители (5, 25), как пожелаете или использовать слайдеры для постепенного изменения, чтобы увидеть, какие отношения является более подходящими на ваш вкус.*

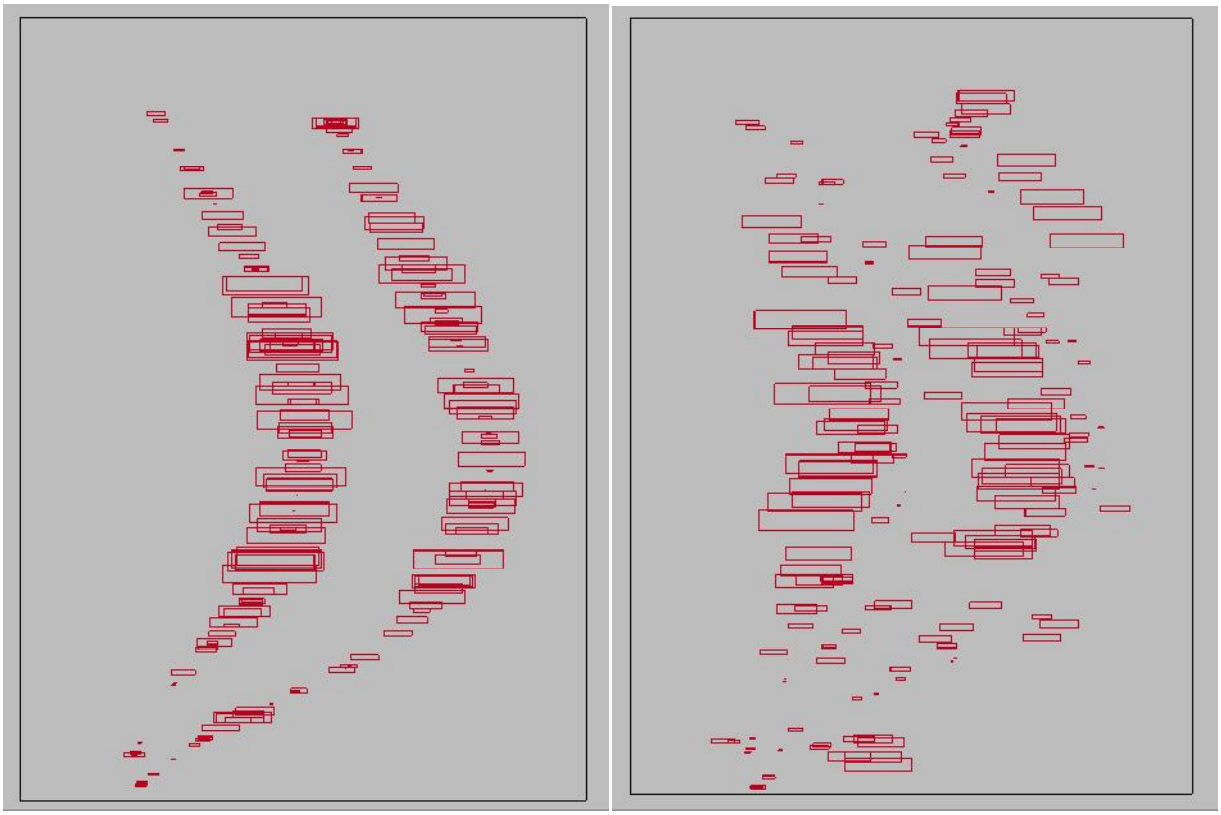


Рис. 103. Манипуляции переменных приводят к различным моделям и мы можем выбрать наиболее подходящий для нашего случая.

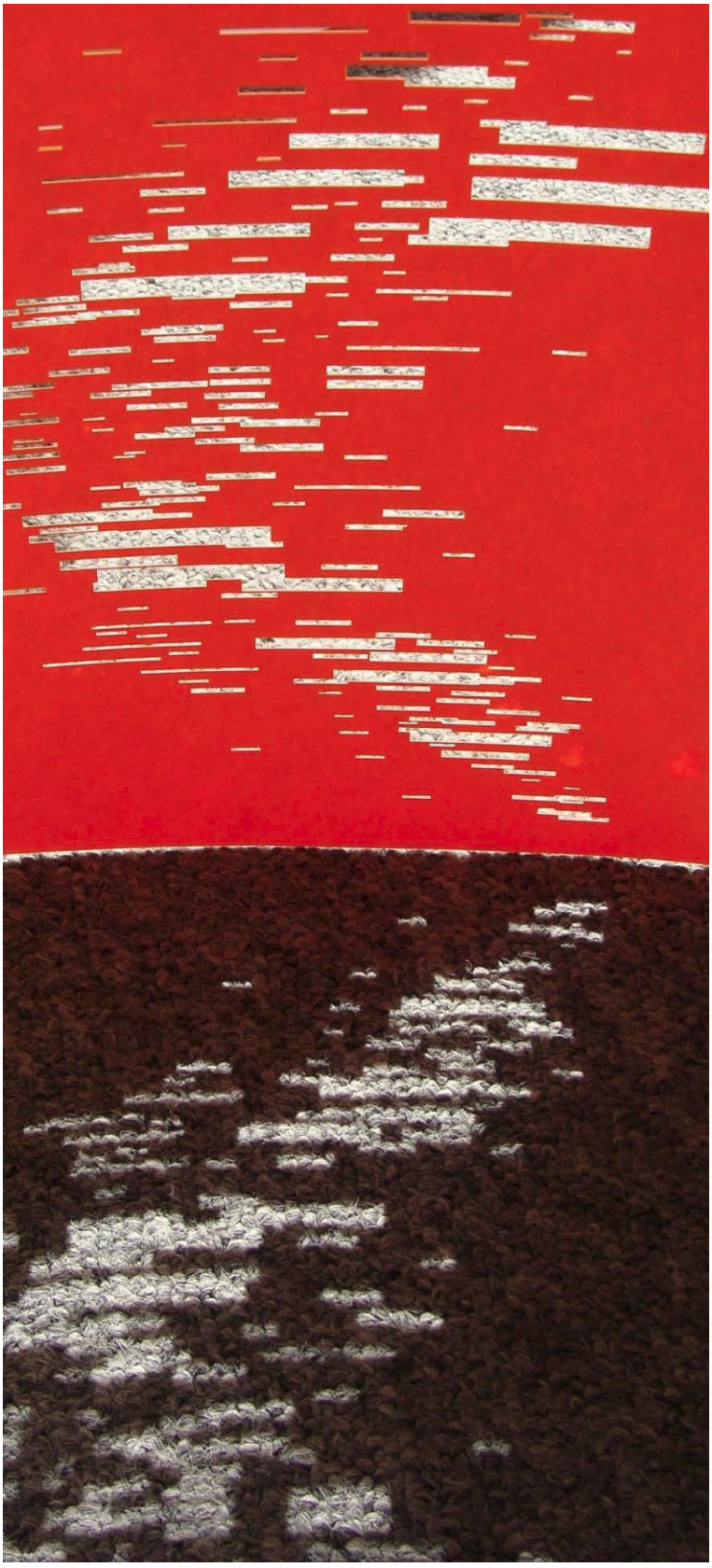


Рис. 104. Модель конечного продукта дизайна, в виде стенки с отверстиями. Различные эффекты тени, которую можно рассматривать как фактор, чтобы контролировать размер отверстия.