Table of Contents

1. О программе 0
2. Введение 1
    1. Что такое визуальное программирование 1.1
    2. Что такое Dynamo 1.2
    3. Dynamo в действии 1.3
3. Привет, Dynamo! 2
    1. Установка и запуск Dynamo 2.1
    2. Пользовательский интерфейс 2.2
    3. Рабочее пространство 2.3
    4. Начало работы 2.4
4. Структура визуальных программ 3
    1. Узлы 3.1
    2. Каркасы 3.2
    3. Библиотека 3.3
    4. Управление структурой программы 3.4
5. Компоновочные блоки программ 4
    1. Данные 4.1
    2. Математика 4.2
    3. Логика 4.3
    4. Строки 4.4
    5. Цвет 4.5
6. Геометрия для машинного проектирования 5
    1. Обзор концепции геометрии 5.1
    2. Векторы 5.2
    3. Точки 5.3
    4. Кривые 5.4
    5. Поверхности 5.5
    6. Тела 5.6
    7. Сети 5.7
    8. Импорт геометрии 5.8
7. Проектирование на основе списков 6
    1. Что такое список 6.1
    2. Работа со списками 6.2
    3. Списки списков 6.3
    4. Многомерные списки 6.4
8. Узлы Code Block и DesignScript 7
    1. Что такое Code Block 7.1
    2. Синтаксис DesignScript 7.2
    3. Сокращение 7.3
    4. Функции 7.4
9. Dynamo для Revit 8
    1. Подключение к Revit 8.1
    2. Выбор 8.2
    3. Редактирование 8.3
    4. Создание 8.4
    5. Адаптация 8.5
    6. Выпуск документации 8.6
10. Словари в Dynamo 9
    1. Что такое словарь 9.1
    2. Применение узлов 9.2
    3. Применение узлов Code Block 9.3
    4. Примеры использования 9.4
11. Пользовательские узлы 10
    1. Пользовательские узлы: введение 10.1
    2. Создание пользовательских узлов 10.2
    3. Публикация узлов в библиотеку 10.3
    4. Узлы Python 10.4
    5. Python и Revit 10.5
    6. Шаблоны Python в Dynamo 2.0 10.6
12. Пакеты 11
    1. Пакеты: введение 11.1
    2. Практикум по работе с пакетом: Mesh Toolkit 11.2
    3. Разработка пакетов 11.3
    4. Публикация пакетов 11.4
    5. Импорт Zero Touch 11.5
13. Создание геометрии с помощью DesignScript 12
    1. Основы работы с геометрией посредством DesignScript 12.1
    2. Геометрические примитивы 12.2
    3. Векторная математика 12.3
    4. Кривые: интерполяционные и по управляющим точкам 12.4
    5. Перенос, поворот и другие преобразования 12.5
    6. Поверхности: интерполяционные, лофтированные, по управляющим точкам и поверхности вращения 12.6
    7. Параметризация геометрических объектов 12.7
    8. Пересечение и обрезка 12.8
    9. Логические операции с геометрическими объектами 12.9
    10. Генераторы точек Python 12.10
14. Рекомендуемые практические приемы 13
    1. Методы создания графиков 13.1

2. Методы создания сценариев 13.2
3. Справочник по созданию сценариев 13.3

15. Приложение 14
    1. Ресурсы 14.1
    2. Указатель узлов 14.2
    3. Полезные пакеты 14.3
    4. Файлы примеров 14.4

О программе

Dynamo Primer

Dynamo 2.0

Скачайте [руководство Dynamo v1.3 Primer здесь]

Dynamo — платформа визуального программирования для проектировщиков с открытым исходным кодом.

Добро пожаловать

Представляем Dynamo Primer — исчерпывающее руководство по визуальному программированию в Autodesk Dynamo. Это активный проект, целью которого является распространение сведений по основам программирования. Здесь содержатся практические советы по проектированию на основе правил, рассматриваются такие темы, как работа с вычислительной геометрией, применение программирования в различных направлениях деятельности, а также многие другие возможности платформы Dynamo.

Преимущество Dynamo заключается в широком спектре поддерживаемых операций, связанных с проектированием. Dynamo предлагает широкий спектр возможностей по началу работы.

- **Познакомьтесь** с понятием визуального программирования.
- **Объедините** рабочие процессы из разных программ.
- **Участвуйте** в жизни активного сообщества пользователей, приглашенных участников и разработчиков.
- **Разрабатывайте** проект на платформе с открытым исходным кодом, чтобы способствовать его развитию.

Нам нужно руководство по работе с Dynamo, отвечающее потребностям пользователей на всех перечисленных уровнях, от новичков до опытных. Это и есть Dynamo Primer.

Данное руководство включает главы, составленные Mode Lab. В этих главах основное внимание уделяется принципам, которые позволят вам приступить к разработке собственных визуальных программ в Dynamo, а также полезным советам по повышению эффективности использования Dynamo. Из этого руководства вы сможете получить следующие сведения.

- **Контекст**: что подразумевается под визуальным программированием и какие принципы необходимо изучить для начала работы в Dynamo
- **Начало работы**: как получить Dynamo и создать первую программу
- **Что входит в программу**: каковы функциональные компоненты Dynamo и как их использовать
- **Компоновочные блоки**: что такое данные и каковы основные типы данных, которые можно использовать в программе
- **Геометрия для проектирования**: как работать с геометрическими элементами в Dynamo
- **Списки, списки, списки**: как упорядочивать и координировать структуры хранения данных
- **Код в узлах**: как расширить возможности Dynamo путем добавления собственного кода
- **Вычислительные операции BIM**: как использовать Dynamo с моделью Revit
- **Пользовательские узлы**: как создать собственные узлы
- **Пакеты**: как предоставить доступ к инструментам другим участникам сообщества

Это самое интересное время для изучения и развития приложения Dynamo, а также для работы с ним. Начнем!

Открытый исходный код

Проект Dynamo Primer открыт для всех. Мы стремимся предоставлять качественную информацию и будем рады вашим отзывам. Если вы хотите сообщить о какой-либо проблеме, опубликуйте свой вопрос на соответствующей странице GitHub: https://github.com/DynamoDS/DynamoPrimer/issues.

Если вы хотите предложить новый раздел, правки или другие изменения по этому проекту, ознакомьтесь с нашим проектом на GitHub: https://github.com/DynamoDS/DynamoPrimer.

Проект Dynamo Primer

Dynamo Primer — проект с открытым исходным кодом, который был инициирован Мэттом Ензыком (Matt Jezyk) и рабочей группой по разработке Dynamo в компании Autodesk.

Первая версия этого руководства была составлена специалистами **Mode Lab**. Мы выражаем им благодарность за то, что положили начало этому ресурсу.

Обновление этого руководства в соответствии с изменениями, внесенными в Dynamo 2.0, было выполнено **Джоном Пирсоном (John Pierson), Parallax Team**.

Благодарности

Выражаем особую благодарность Иэну Кио (Ian Keough) за то, что положил начало проекту Dynamo.

Мы благодарим Мэтта Ензыка, Иэна Кио, Зака Крона (Zach Kron), Рэйсел Уильямс (Racel Williams) и Колина МакКрона (Colin McCrone) за активную совместную работу и возможность принять участие в широком спектре проектов Dynamo.

Программное обеспечение и ресурсы

Dynamo: текущая стабильная* версия Dynamo — 2.1.0.

http://dynamobim.com/download/ или http://dynamobuilds.com

- Примечание. Начиная с версии Revit 2020, Dynamo включается в комплект установки Revit, поэтому устанавливать Dynamo вручную не требуется. Подробные сведения см. в этом блоге.

DynamoBIM: веб-сайт DynamoBIM — это лучшее место для получения дополнительной информации, поиска обучающих материалов и общения на форумах.

http://dynamobim.org

Страница Dynamo на GitHub: Dynamo — это проект разработки с открытым исходным кодом, размещенный на GitHub. Если вы хотите внести свой вклад в работу над ним, посетите DynamoDS.

https://github.com/DynamoDS/Dynamo

Контакты: если вы обнаружите какую-либо проблему в этом документе, сообщите нам об этом.

Dynamo@autodesk.com

Лицензия

Copyright 2019 Autodesk

Лицензировано Apache License, Version 2.0 ("Лицензия"); использовать только в совокупности с Лицензией. Копию Лицензии можно получить по следующему адресу:

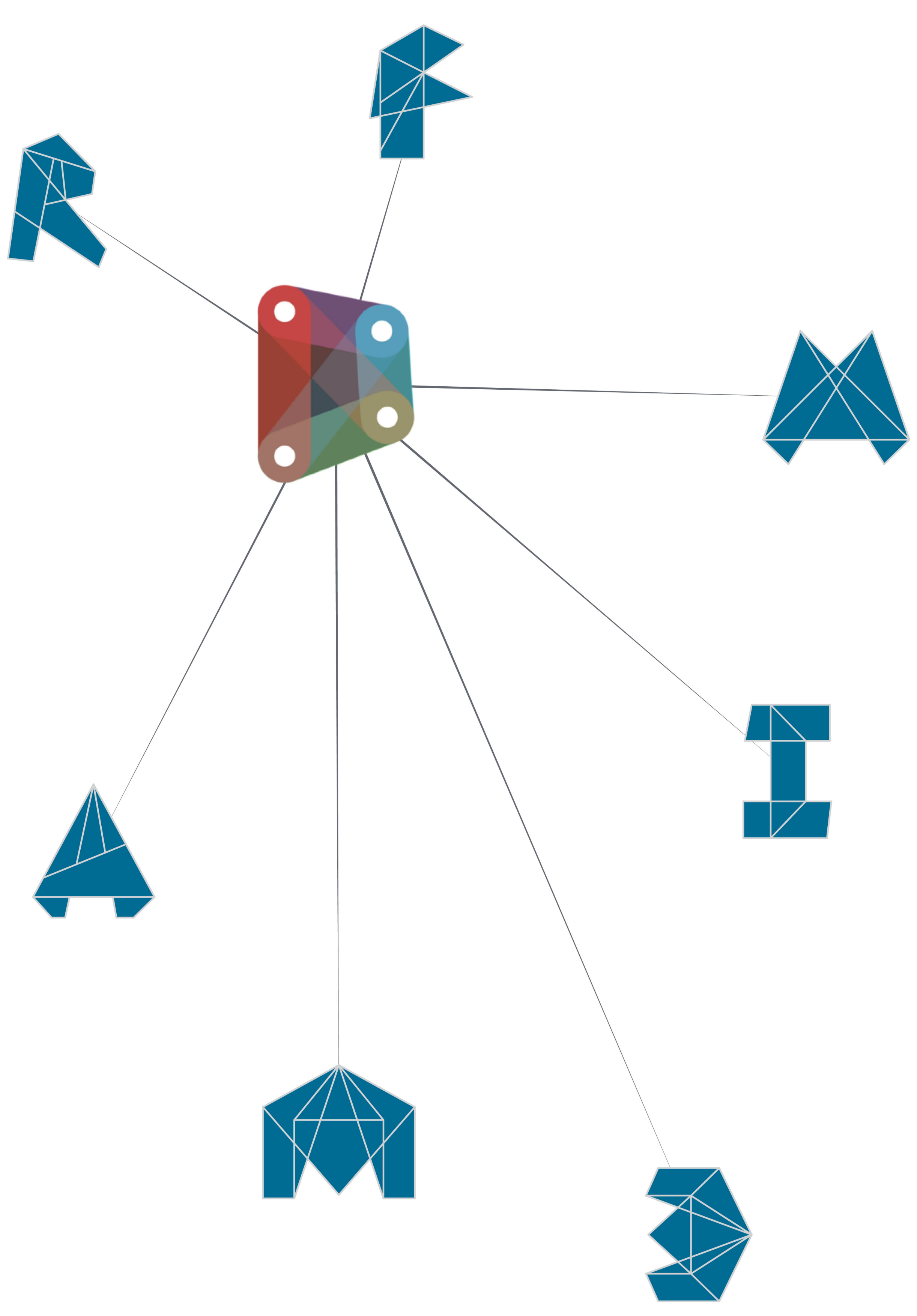
http://www.apache.org/licenses/LICENSE-2.0

За исключением случаев, предусматриваемых соответствующим законом или заключенным в письменном виде соглашением, программное обеспечение, распространяемое на условиях Лицензии, распространяется "КАК ЕСТЬ", БЕЗ КАКИХ-ЛИБО УСЛОВИЙ И ГАРАНТИЙ, выраженных в явной форме или подразумеваемых. Текст Лицензии содержит точные разъяснения разрешенного и запрещенного использования программных продуктов на условиях Лицензии.

Введение

ВВЕДЕНИЕ

Изначально приложение Dynamo задумывалось как надстройка Revit для информационного моделирования зданий, однако со временем оно стало самостоятельным многофункциональным решением. В первую очередь Dynamo — это платформа, позволяющая проектировщикам изучать процессы визуального программирования, устранять возникающие проблемы и создавать собственные инструменты. Знакомство с приложением Dynamo следует начать с изучения того, что оно из себя представляет и как именно его можно использовать.



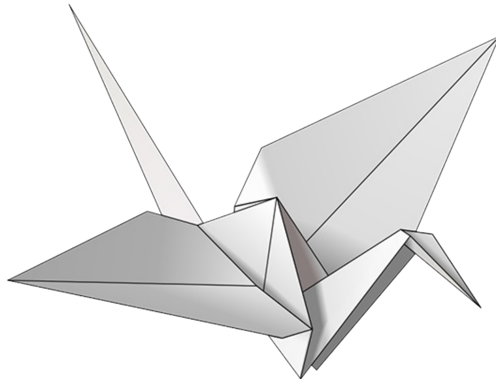
Что такое визуальное программирование

Что такое визуальное программирование

Процесс разработки большинства проектов включает в себя построение визуальных, системных и геометрических связей между элементами. В подавляющем большинстве случаев для создания этих связей используются рабочие процессы, в которых переход от концепции к конечному результату осуществляется за счет применения правил. Таким образом, процесс работы (возможно, непреднамеренно) строится по алгоритмическому принципу, в основе которого лежит использование пошагового набора действий, следующих стандартной логике: ввод данных, их обработка и, наконец, вывод. Инструменты программирования позволяют формализовать эти алгоритмические процессы и повысить эффективность их использования.

Алгоритмы в действии

Алгоритмы являются мощными инструментами, однако само понятие **алгоритма** часто толкуется не совсем правильно. Безусловно, с помощью алгоритмов можно добиться самых неожиданных и поразительных результатов, однако не стоит искать в этом какой-то волшебный секрет. На самом деле алгоритмы по своей сути весьма и весьма ординарны. В качестве наглядного примера можно привести процесс создания бумажного журавлика. Мы берем квадратный лист бумаги (ввод), складываем его определенным образом (действия по обработке) и в результате получаем журавлика (вывод).

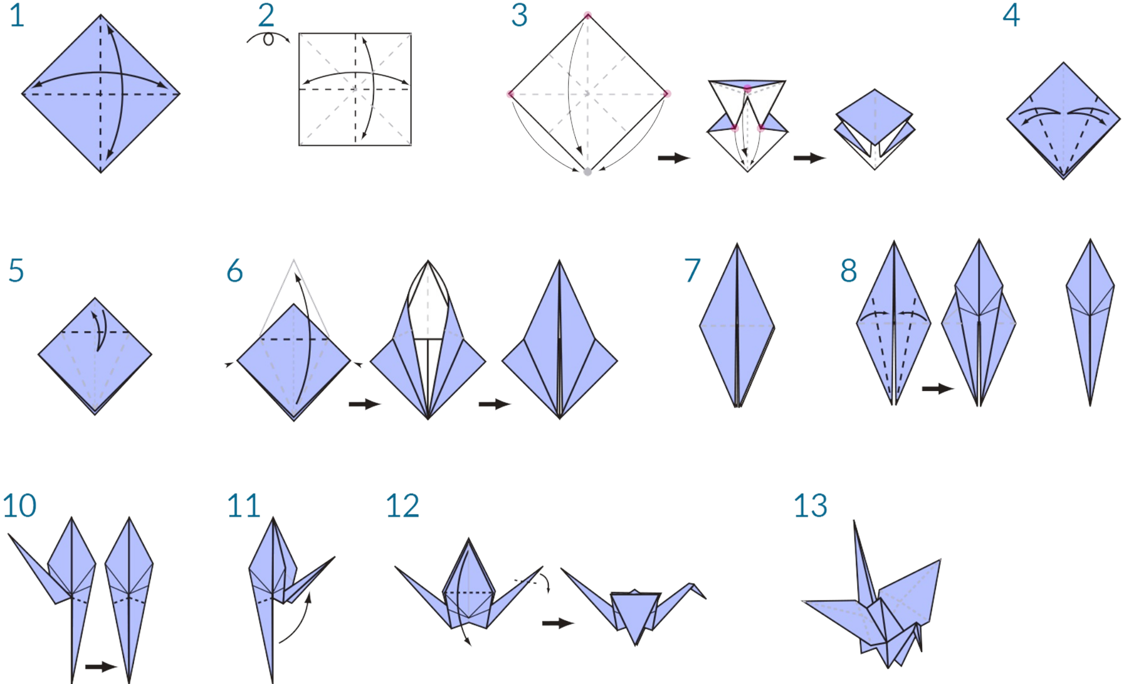


Итак, что же такое алгоритм? Это абстрактный набор шагов, которые можно представить либо в текстовой, либо в графической форме.

Текстовые инструкции

1. Возьмем квадратный лист бумаги и повернем его цветной стороной вверх. Сложим лист пополам, развернем, затем сложим его пополам в другом направлении и снова развернем.
2. Перевернем лист белой стороной вверх, сложим пополам, тщательно разгладим, развернем, затем сложим в другом направлении и опять развернем.
3. Следуя созданным сгибам, сложим лист так, чтобы совместить три верхних угла с нижним, и разгладим полученную конструкцию.
4. Сложим боковые углы верхнего квадрата к центру, затем развернем их обратно.
5. Сложим верхний угол модели книзу, тщательно разгладим и также развернем обратно.
6. Следуя созданным сгибам, раскроем верхний квадрат, потянув нижний угол вверх и сложив боковые стороны к центру. Тщательно разгладим полученную конструкцию.
7. Перевернем конструкцию и повторим шаги 4–6 с другой стороны.
8. Сложим боковые углы верхней части конструкции к центру.
9. Перевернем конструкцию и повторим это действие с другой стороны.
10. Согнем обе «ножки» конструкции в направлении вверх и развернем их обратно.
11. Следуя только что созданным сгибам, согнем обе «ножки» внутрь и вверх.
12. Проделаем аналогичную процедуру с концом одной из «ножек», чтобы сделать клюв, и опустим крылья вниз.
13. Получился журавлик.

Графические инструкции



Определение процесса программирования

Выполнение обоих представленных наборов инструкций приводит к одному и тому же результату — бумажному журавлику. Если вы следовали инструкциям и сделали журавлика, то вы только что применили алгоритм. Единственное различие между этими наборами инструкций заключается в их формальном выражении, что подводит нас к понятию **программирования**. Программирование (а точнее, *компьютерное программирование*) представляет собой создание исполняемой программы для обработки последовательности действий путем их формализации. Если преобразовать приведенные выше инструкции по созданию бумажного журавлика в формат, который сможет прочитать и выполнить компьютер, это и будет программированием.

Первое правило и первая трудность программирования заключается в том, что для эффективного взаимодействия с компьютером человеку приходится прибегать к той или иной форме абстракции. Абстракция достигается за счет использования языков программирования, таких как JavaScript, Python или C. Если у нас есть повторяемый набор инструкций, например для создания бумажного журавлика, то от нас требуется лишь перевести его на язык компьютера. В результате мы сможем создать с помощью компьютера нужного нам журавлика или целое множество журавликов, различающихся между собой по тем или иным параметрам. Таким образом, компьютер позволяет многократно выполнять отдельную задачу или набор задач без задержек и ошибок, возникающих в связи с человеческим фактором. В этом и заключается сила программирования.

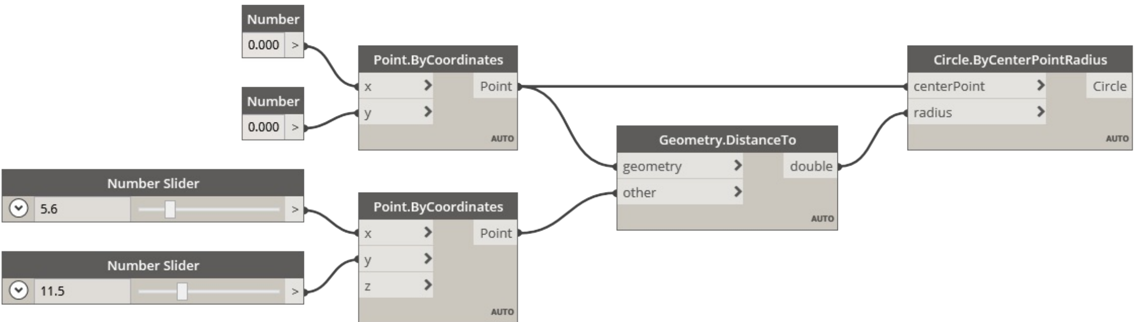
Определение процесса визуального программирования

> Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): Visual Programming - Circle Through Point.dyn. Полный список файлов примеров можно найти в приложении.

Если бы вам было нужно написать инструкцию по созданию бумажного журавлика, что бы вы использовали: графику, текст или их сочетание?

Если вы решили использовать графику, то вам следует обратить внимание на **визуальное программирование**. Процессы текстового и визуального программирования по сути своей ничем не отличаются. И там, и там используется одна и та же платформа формализации, однако при визуальном программировании инструкции и связи в программе определяются посредством графического (т. е. визуального) пользовательского интерфейса и вместо ввода текста, ограниченного возможностями синтаксиса, создается цепочка из готовых к использованию узлов. Для сравнения рассмотрим один и тот же алгоритм рисования окружности через точку, заданный с помощью узлов и с помощью кода.

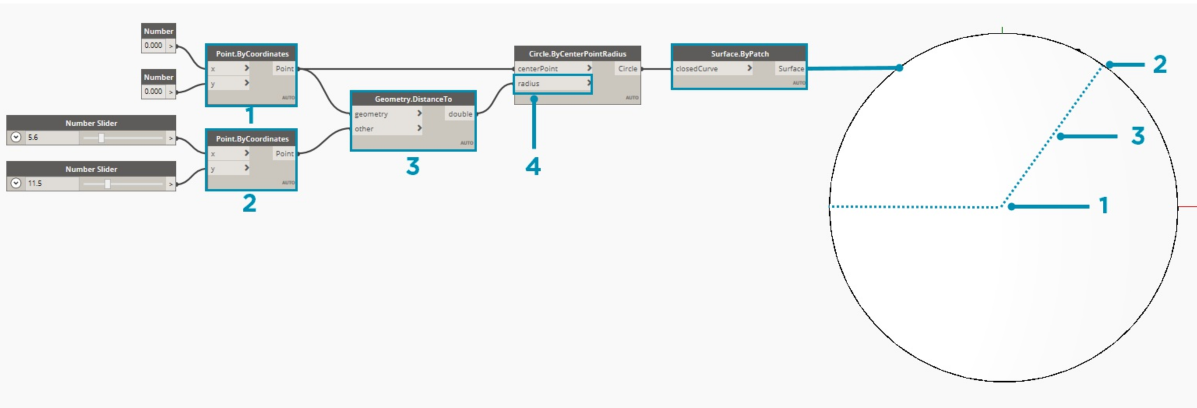
Визуальная программа



Текстовая программа

```
myPoint = Point.ByCoordinates(0.0,0.0,0.0);
x = 5.6;
y = 11.5;
attractorPoint = Point.ByCoordinates(x,y,0.0);
dist = myPoint.DistanceTo(attractorPoint);
myCircle = Circle.ByCenterPointRadius(myPoint,dist);
```

Результаты алгоритма



Визуальная наглядность этого способа программирования делает его более простым и понятным для программистов. Приложение Dynamo относится к средствам визуального программирования, но, как будет показано далее, оно также поддерживает возможности текстового программирования.

Что такое Dynamo

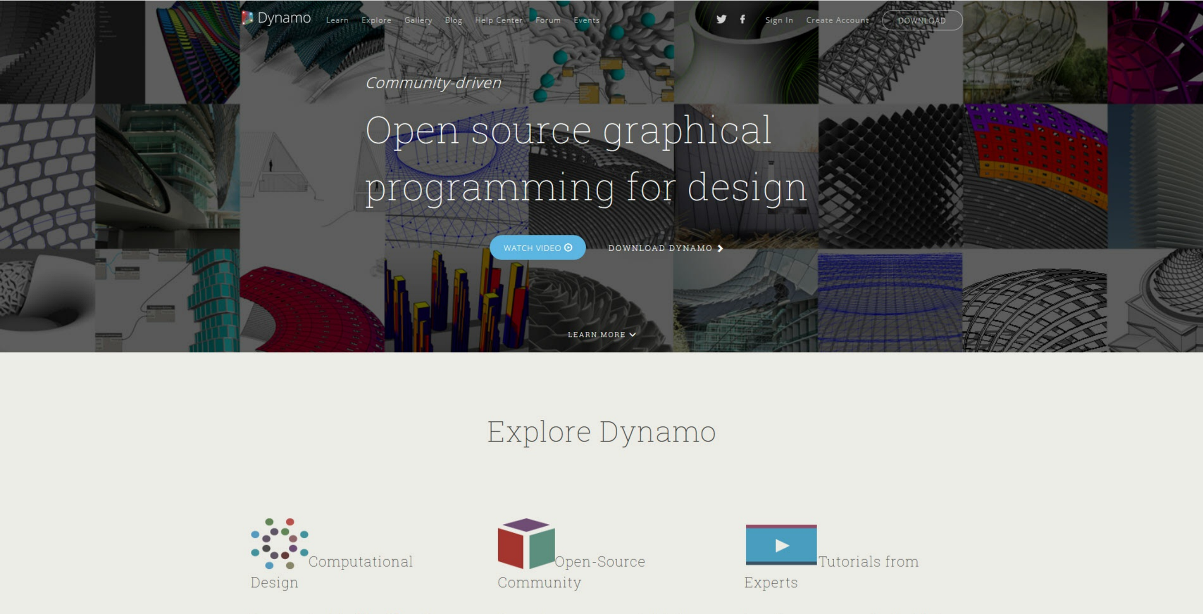
Что такое Dynamo

Dynamo — это инструмент, который может стать в ваших руках буквально чем угодно. Вы можете использовать приложение Dynamo как самостоятельное решение либо в связке с программным обеспечением Autodesk, заниматься с его помощью визуальным программированием, а также принимать участие в жизни обширного сообщества пользователей и профессиональных разработчиков.

Приложение

Приложение Dynamo — это программное обеспечение, которое можно скачать и использовать в однопользовательском режиме песочницы либо в качестве подключаемого модуля для других программ, таких как Revit или Maya. Ему можно дать следующее определение.

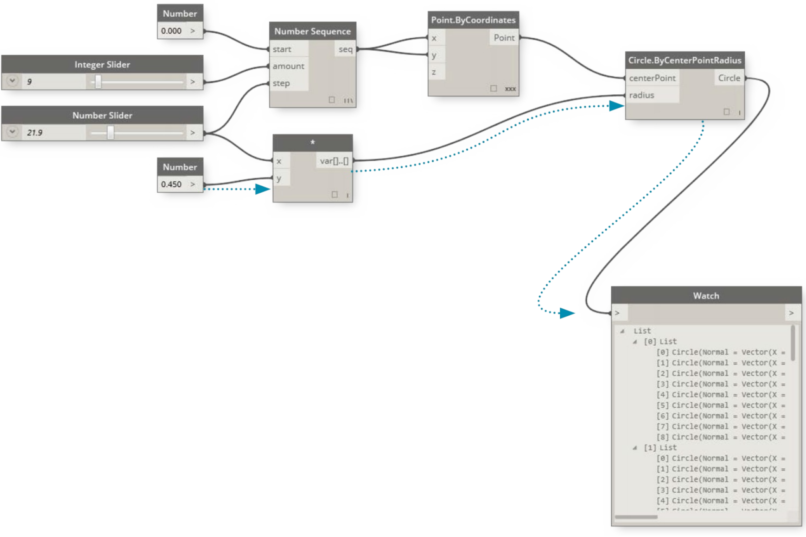
> Это средство визуального программирования, в равной степени доступное как для программистов, так и для специалистов, не связанных с программированием. С помощью Dynamo можно создавать визуальные сценарии для определенных режимов работы, образовывать пользовательские элементы логики и писать сценарии с использованием различных текстовых языков программирования.



1. Ознакомьтесь с примерами использования Dynamo в Revit.
2. Скачайте установщик.

Процесс

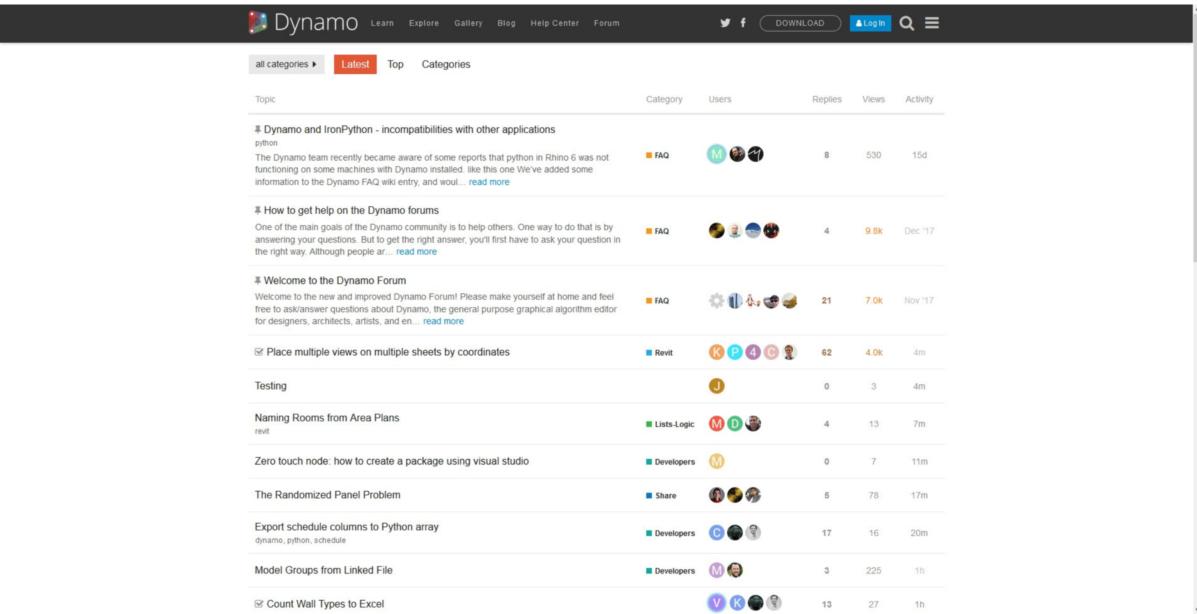
Установив приложение Dynamo, вы получаете доступ к процессу визуального программирования, который позволяет создавать пользовательские алгоритмы путем определения связей между элементами, образующими последовательность действий. Эти алгоритмы можно применять для широкого спектра задач — от обработки данных до создания геометрии, — и все это в реальном времени без написания единой строчки `code`.



> Чтобы создавать визуальные программы, просто добавляйте элементы и соединяйте их друг с другом.

Сообщество

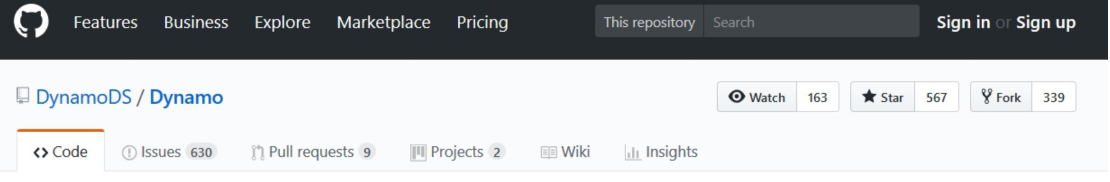
То, чем Dynamo является сегодня, — во многом заслуга постоянного круга активных пользователей и профессиональных программистов, помогающих приложению развиваться. Станьте участником нашего сообщества: читайте блог, публикуйте результаты своей работы в галерее и обсуждайте Dynamo с другими пользователями на форуме.



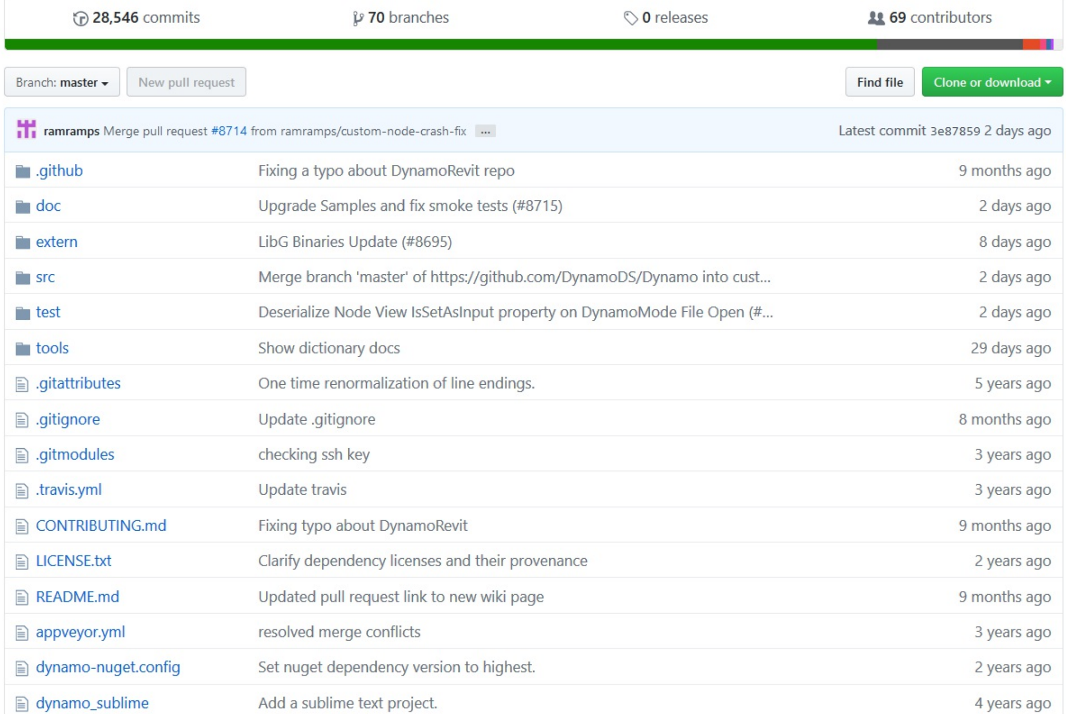
Платформа

Приложение Dynamo задумывалось как средство визуального программирования для проектировщиков, в котором можно создавать инструменты для работы с внешними библиотеками или любой программой Autodesk с API. Dynamo Sandbox позволяет создавать программы в среде песочницы, но экосистема, в которой существует Dynamo, постоянно пополняется и расширяется.

Исходный код проекта является открытым, благодаря чему расширять функциональность приложения можно настолько, насколько хватит фантазии. Посетите страницу проекта Dynamo на сайте GitHub и просмотрите текущие проекты пользователей, занимающихся адаптацией Dynamo.



Open Source Graphical Programming for Design http://dynamobim.org



README.md

Dynamo is a visual programming tool that aims to be accessible to both non-programmers and programmers alike. It gives users the ability to visually script behavior, define custom pieces of logic, and script using various textual programming languages.

Get Dynamo

Looking to learn or download Dynamo? Check out dynamobim.org!

Develop

Create a Node Library for Dynamo

If you're interested in developing a Node library for Dynamo, the easiest place to start is by browsing the DynamoSamples. These samples use the Dynamo NuGet packages which can be installed using the NuGet package manager in Visual Studio.

Documentation of the Dynamo API with a searchable index of public API calls for core functionality. This will be expanded to include regular nodes and Revit functionality.

Просматривайте интересные проекты, создавайте Fork-копии и адаптируйте Dynamo под свои потребности.

Dynamo в действии

DYNAMO В ДЕЙСТВИИ

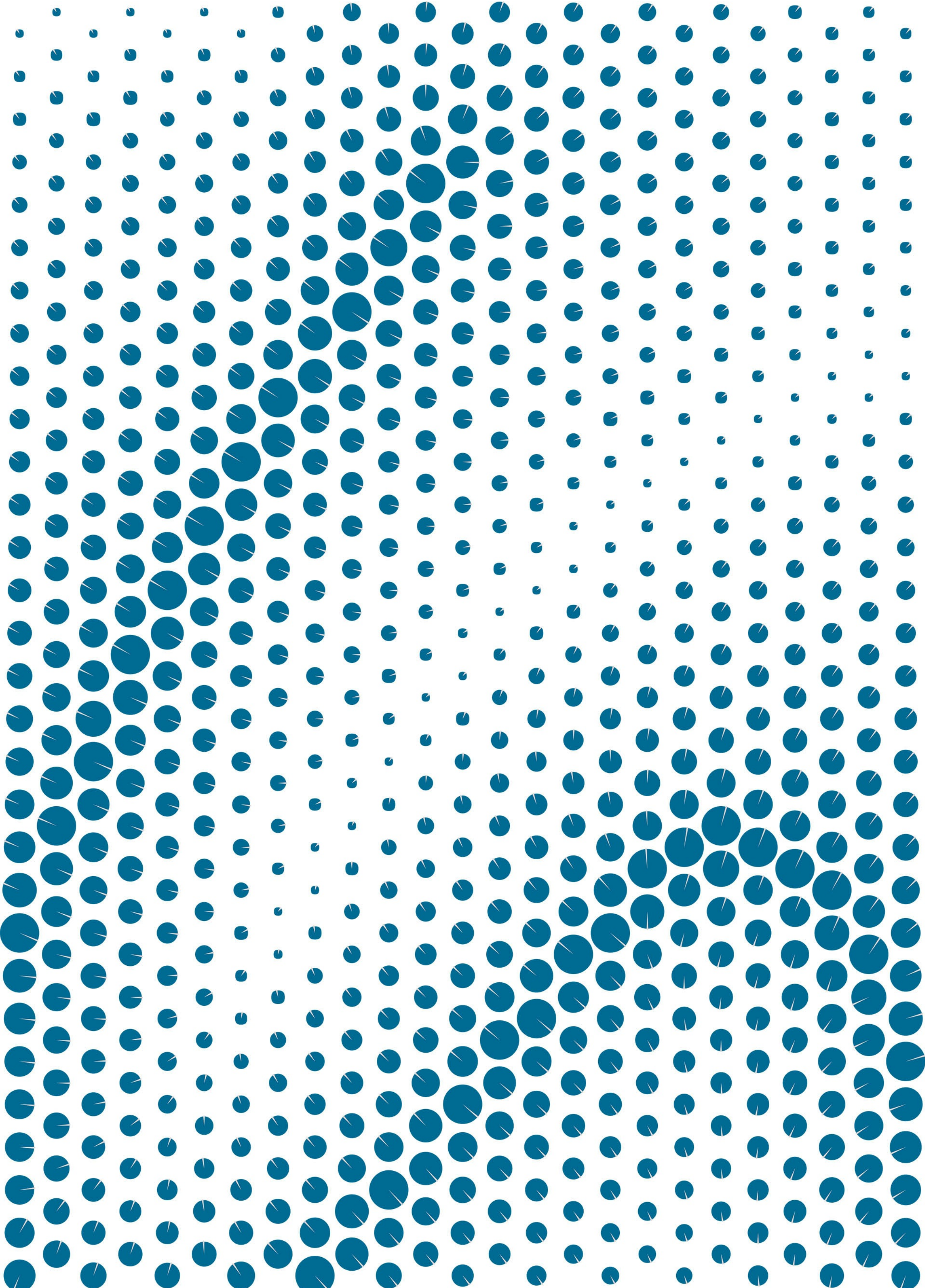
Приложение Dynamo является надежным помощником в решении самых разных задач, начиная с использования средств визуального программирования для проектирования рабочих процессов и заканчивая разработкой специализированных инструментов.

Подписывайтесь на страницу Dynamo в Pinterest.

Привет, Dynamo!

ПРИВЕТ, DYNAMO!

Dynamo — это платформа для визуального программирования, которая является гибким и расширяемым инструментом проектирования. Поскольку Dynamo работает и как однопользовательское приложение, и как надстройка для другого ПО, его можно использовать для разработки широкого ряда творческих рабочих процессов. Предлагаем установить приложение Dynamo и начать знакомство с ним с изучения основных компонентов его интерфейса.



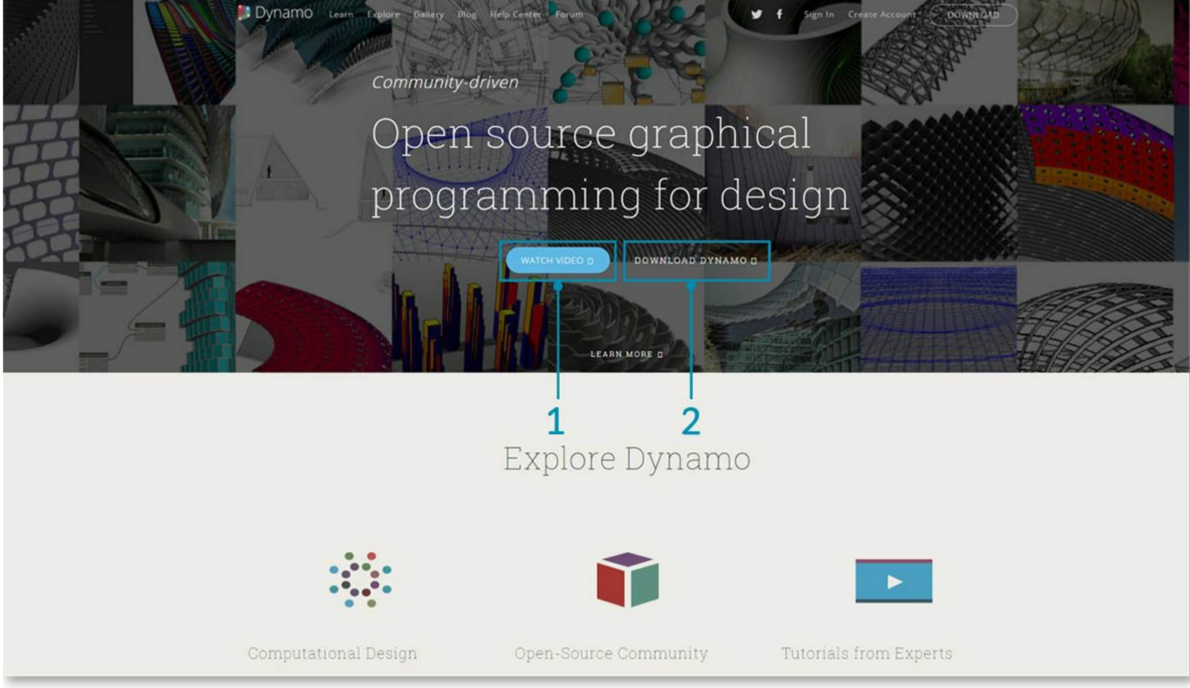
Установка и запуск Dynamo

Установка и запуск Dynamo

Dynamo — это действующий проект с открытым исходным кодом. Вы можете скачать установщик и для официальной версии, и для новейших, «неофициальных» версий от разработчиков. Скачайте официальный выпуск и приступайте к работе либо примите участие в разработке новых версий Dynamo на сайте GitHub.

Скачивание

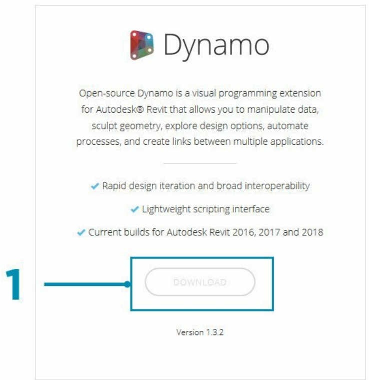
Чтобы скачать официальный выпуск Dynamo, посетите веб-сайт Dynamo. Скачивание можно запустить, нажав соответствующую кнопку на главной странице либо открыв предназначенную для этого страницу.

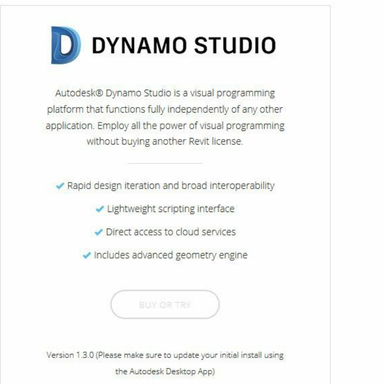


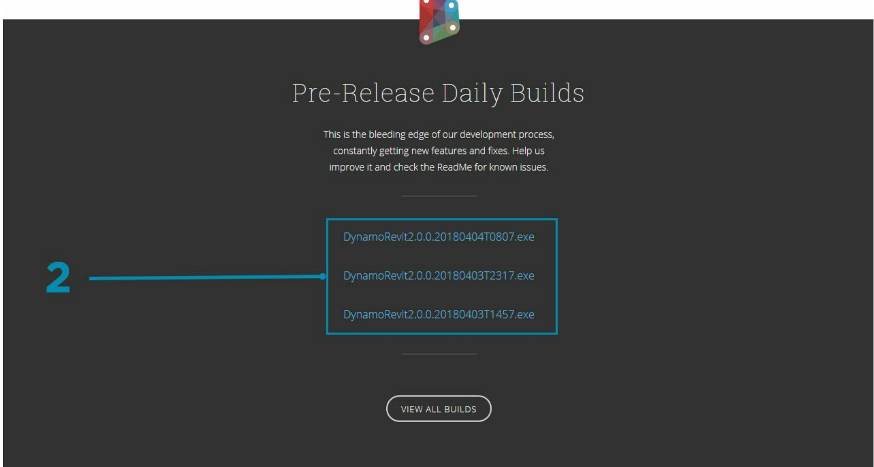
1. Просмотрите видеоролик об использовании возможностей машинного проектирования Dynamo в сфере архитектуры.
2. Или перейдите на страницу скачивания.

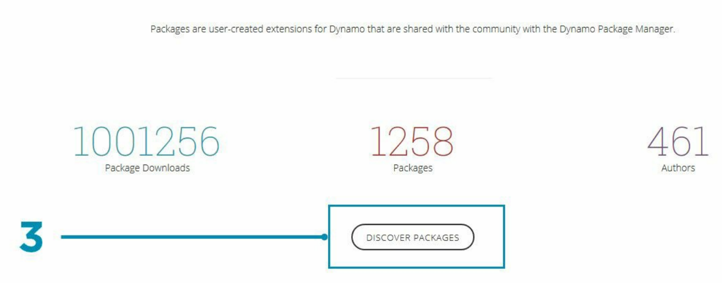
Здесь можно скачать новейшие версии от разработчиков или перейти на страницу проекта Dynamo на сайте Github.

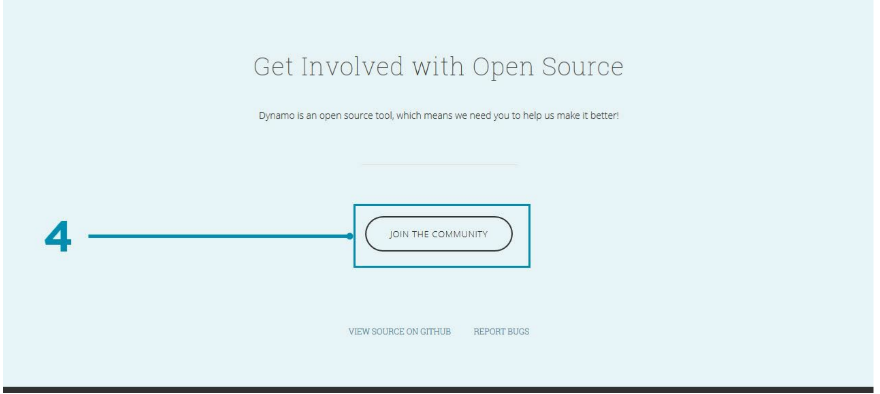
Dynamo
Learn
Explore
Gallery
Blog
Help Center
Forum
Events
Sign In
Create Account
DOWNLOAD
This is Dynamo


Dynamo
Open-source Dynamo is a visual programming extension for Autodesk® Revit that allows you to manipulate data, sculpt geometry, explore design options, automate processes, and create links between multiple applications.
Rapid design iteration and broad interoperability
Lightweight scripting interface
Current builds for Autodesk Revit 2016, 2017 and 2018
1
Version 1.3.2


DYNAMO STUDIO
Autodesk® Dynamo Studio is a visual programming platform that functions fully independently of any other application. Employ all the power of visual programming without buying another Revit license.
Rapid design iteration and broad interoperability
Lightweight scripting interface
Direct access to cloud services
Includes advanced geometry engine
Version 1.3.0 (Please make sure to update your initial install using the Autodesk Desktop App)


Pre-Release Daily Builds
This is the bleeding edge of our development process, constantly getting new features and fixes. Help us improve it and check the ReadMe for known issues.
2
DynamoRevit2.0.0.20180404T0807.exe
DynamoRevit2.0.0.20180403T2317.exe
DynamoRevit2.0.0.20180403T1457.exe
VIEW ALL BUILDS

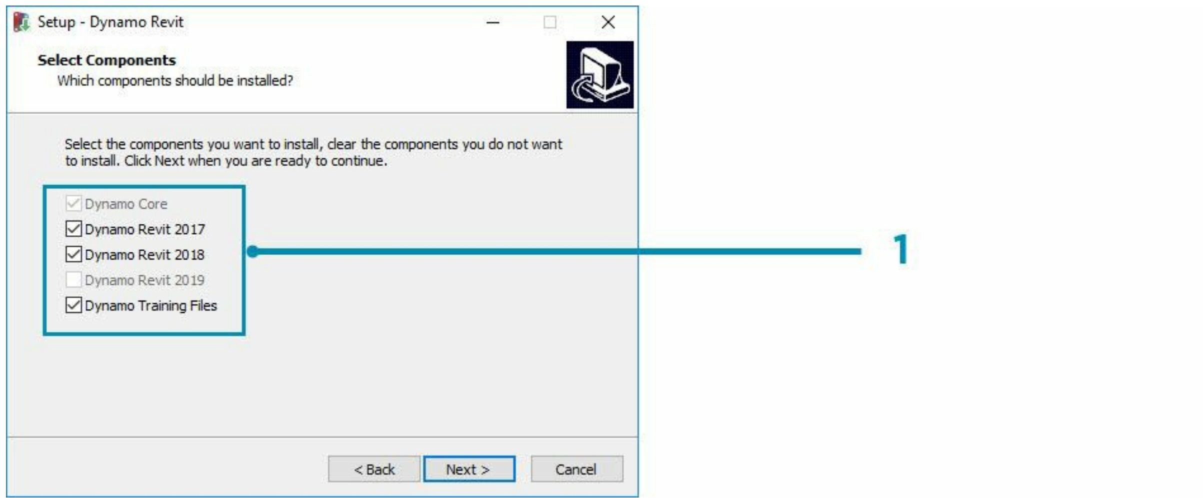

Node Packages
Packages are user-created extensions for Dynamo that are shared with the community with the Dynamo Package Manager.
1001256
Package Downloads
1258
Packages
461
Authors
3
DISCOVER PACKAGES


Get Involved with Open Source
Dynamo is an open source tool, which means we need you to help us make it better!
4
JOIN THE COMMUNITY
VIEW SOURCE ON GITHUB
REPORT BUGS

1. Скачайте установщик для официального выпуска.
2. Скачайте установщик для новейших версий от разработчиков.
3. Ознакомьтесь с пользовательскими пакетами, предоставленными участниками сообщества.
4. Примите участие в разработке проекта Dynamo на GitHub.

Установка

Перейдите в папку со скачанным установщиком и запустите исполняемый файл. В процессе установки можно указать, какие именно компоненты требуется установить.

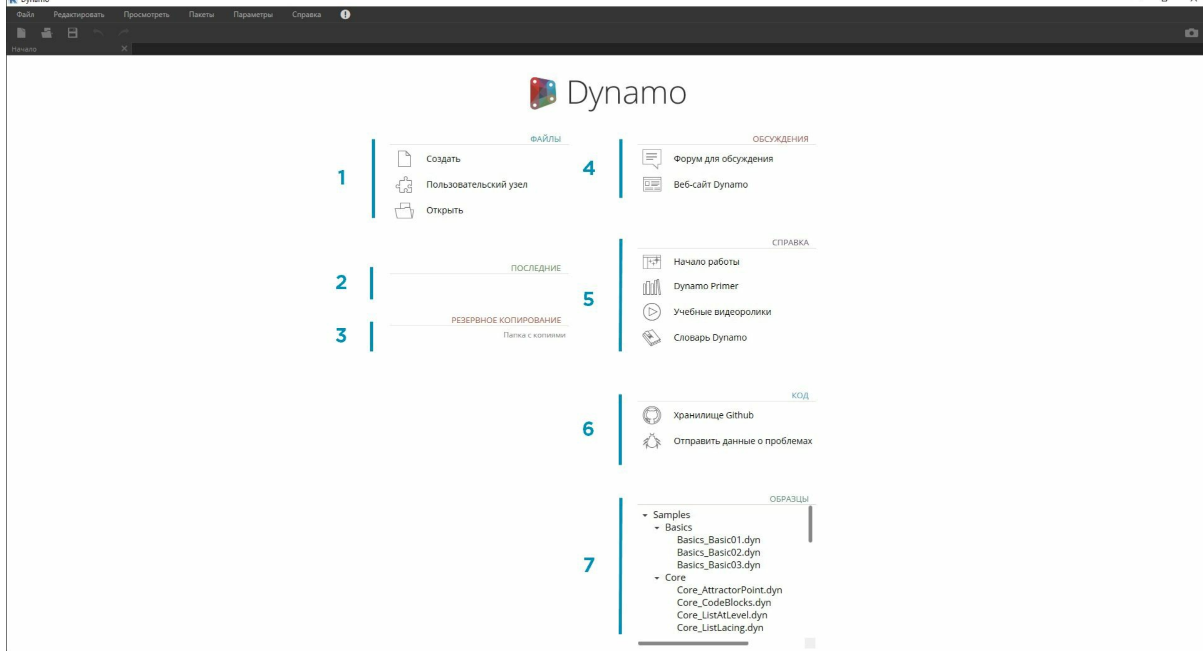


1. Выберите компоненты, которые требуется установить.

Здесь необходимо указать, следует ли устанавливать компоненты, отвечающие за подключение Dynamo к другим установленным приложениям, например Revit. Дополнительные сведения о платформе Dynamo см. в **главе 1.2**.

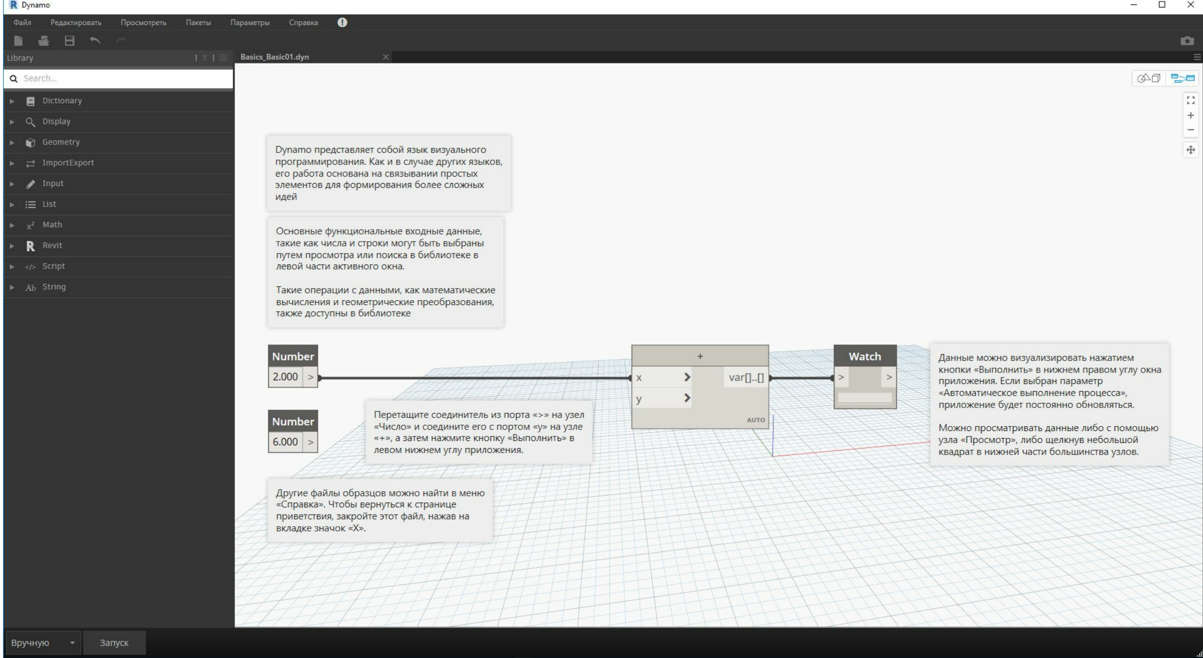
Запуск

Чтобы запустить Dynamo, перейдите в папку \Program Files\Dynamo\Dynamo Revit\x.y и выберите DynamoSandbox.exe. Откроется однопользовательская версия Dynamo и отобразится *начальная страница*. На этой странице доступны стандартные меню и панель инструментов, а также набор ярлыков для работы с файлами и доступа к ресурсам.



1. «Файлы»: создание нового файла или открытие существующего.
2. «Последние»: список последних файлов.
3. «Резервное копирование»: доступ к резервным копиям.
4. «Обсуждения»: непосредственный доступ к форуму пользователей или веб-сайту Dynamo.
5. «Справка»: дополнительные обучающие ресурсы.
6. «Код»: разработка проекта с открытым исходным кодом.
7. «Образцы»: файлы примеров, входящие в комплект установки.

Откройте первый файл примеров, чтобы в первый раз открыть рабочее пространство и убедиться, что приложение Dynamo работает правильно. Выберите Samples > Basics > **Basics_Basic01.dyn**.



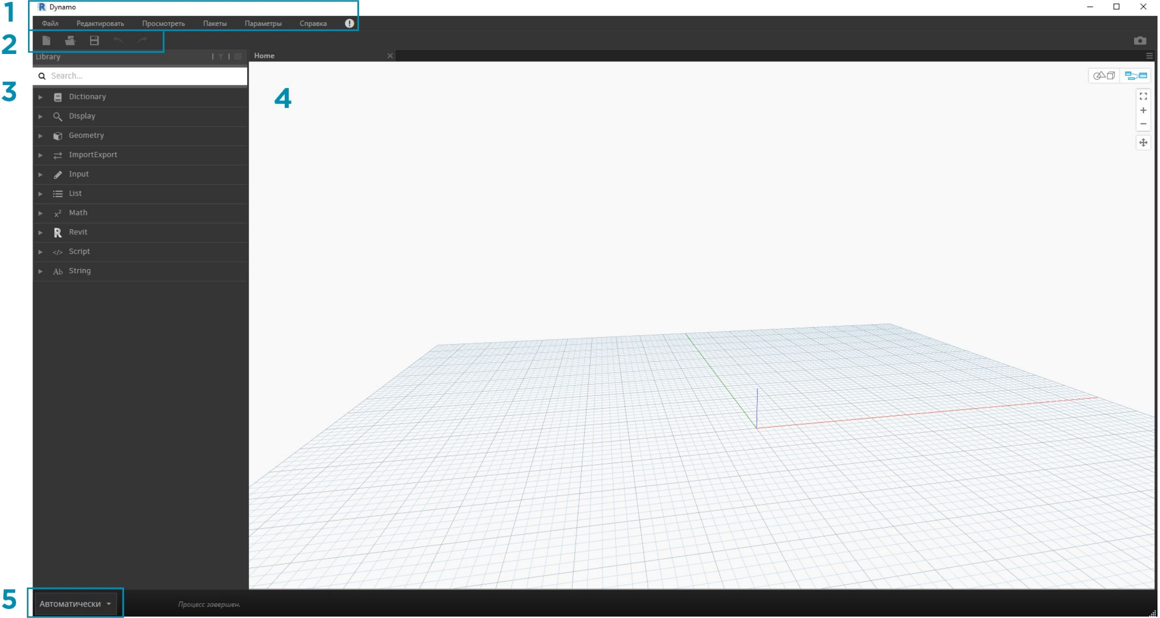
1. Убедитесь, что в строке выполнения выбран параметр «Автоматически», или нажмите кнопку «Запуск».
2. Следуйте инструкциям и соедините узел **Number** с узлом **+**.
3. Убедитесь, что в узле Watch отображается результат выполненной операции.

Если этот файл загружается успешно, то вы сможете запустить свою первую визуальную программу Dynamo.

Пользовательский интерфейс

Пользовательский интерфейс Dynamo

Пользовательский интерфейс Dynamo состоит из пяти основных областей, самая крупная из которых — рабочее пространство, в котором составляются визуальные программы.

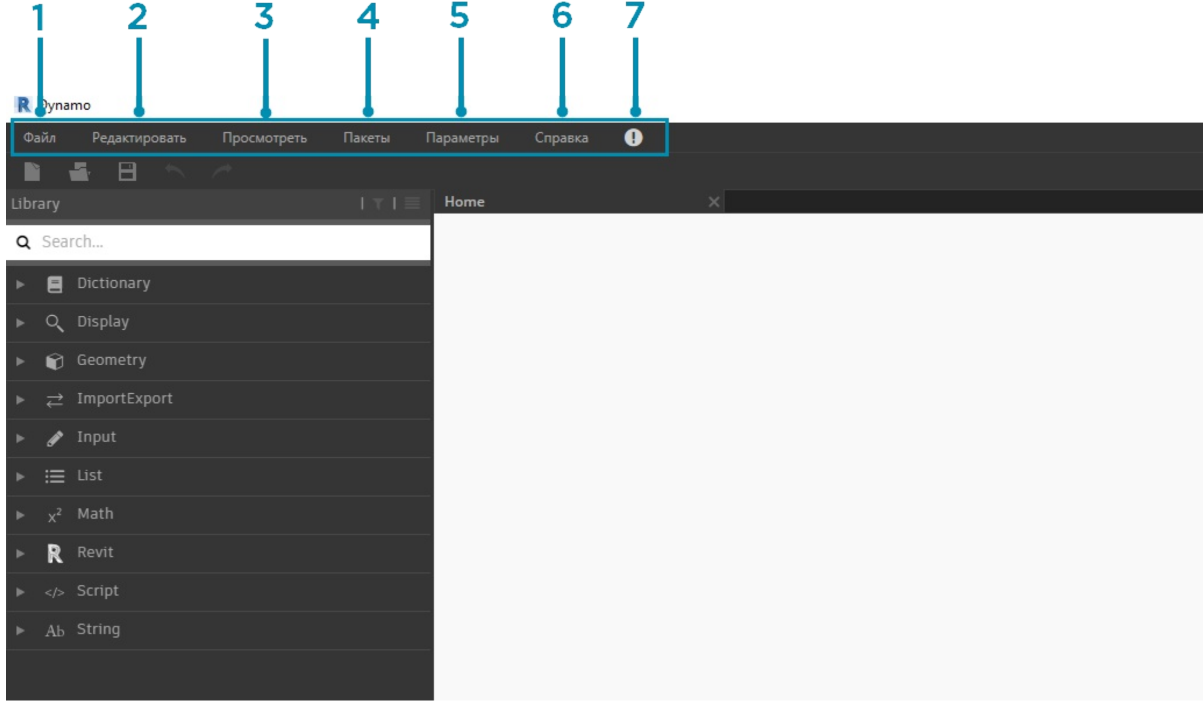


1. Меню
2. Панель инструментов
3. Библиотека
4. Рабочее пространство
5. Панель выполнения

В этом разделе мы подробнее рассмотрим пользовательский интерфейс приложения и функции каждой области.

Меню

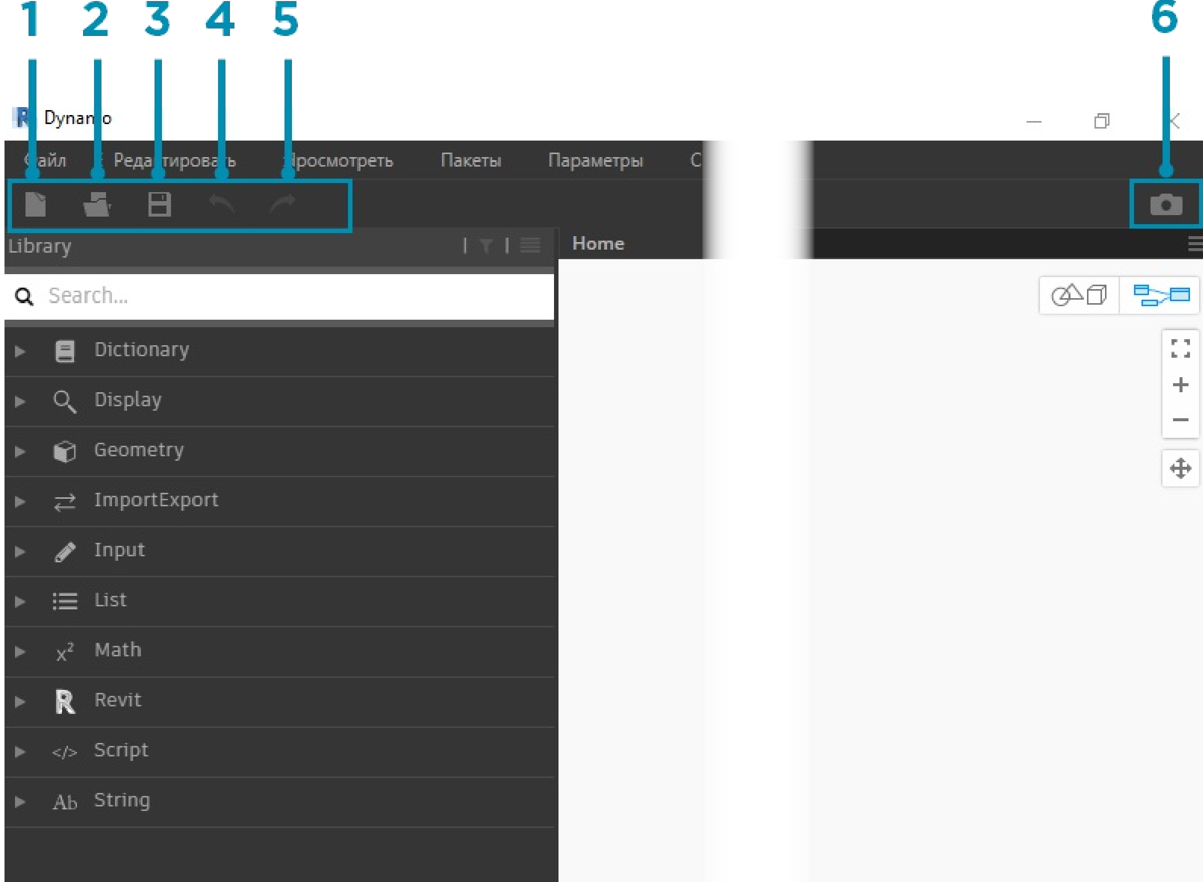
С помощью раскрывающихся меню можно получить доступ к основным функциям приложения Dynamo. Как и в большинстве приложений на базе Windows, в первых двух меню содержатся стандартные функции, связанные с управлением файлами, а также операциями выбора и редактирования компонентов. В остальных меню содержатся функции, присущие именно Dynamo.



1. Файл
2. Редактировать
3. Просмотреть
4. Пакеты
5. Параметры
6. Справка
7. Уведомления

Панель инструментов

Панель инструментов Dynamo содержит ряд кнопок для быстрого доступа к файлам, а также команды «Отменить» [CTRL+Z] и «Повторить [CTRL+Y]». Справа находится еще одна кнопка, с помощью которой можно экспортировать снимок рабочего пространства, что часто требуется при создании документации и обмене данными.



1. «Создать»: создание нового файла DYN.
2. «Открыть»: открытие существующего файла DYN (рабочее пространство) или DYF (пользовательский узел).
3. «Сохранить/Сохранить как»: сохранение активного файла DYN или DYF.
4. «Отменить»: отмена последнего действия.
5. «Повторить»: повтор действия.
6. «Экспорт рабочего пространства в виде изображения»: экспорт видимого рабочего пространства в файл PNG.

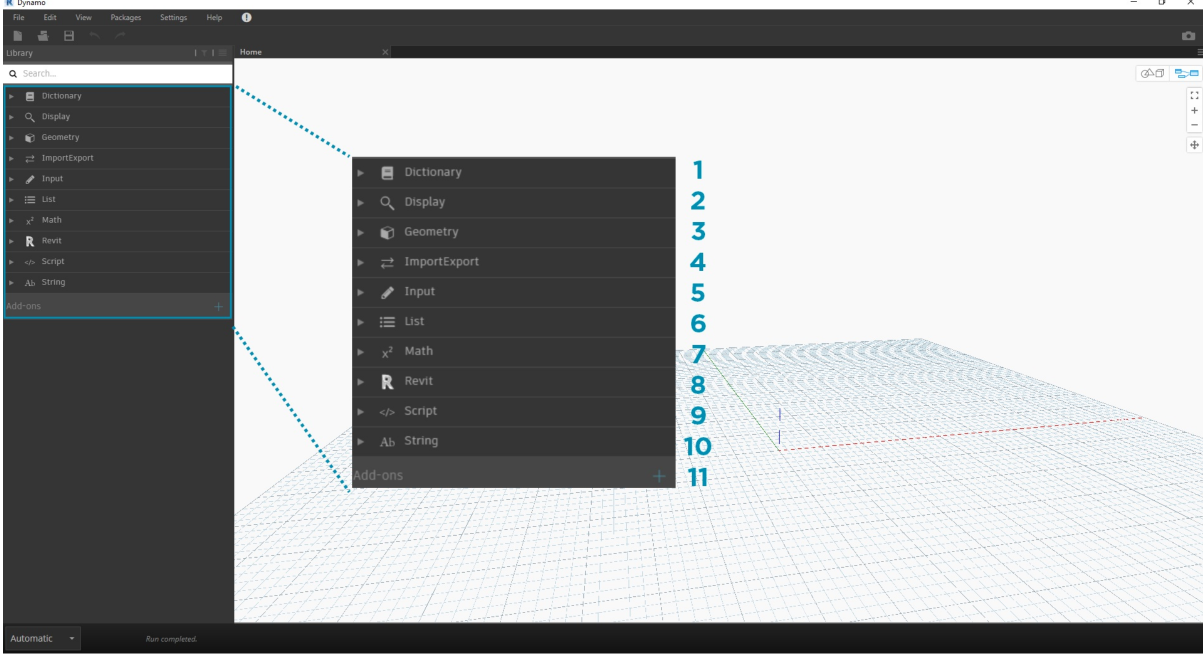
Библиотека

Библиотека содержит все загруженные узлы, включая узлы по умолчанию, входящие в установочный пакет, а также все дополнительно загруженные пользовательские узлы и пакеты. Узлы в библиотеке распределены иерархически по библиотекам, категориям и, при необходимости, подкатегориям в зависимости от функции узла: создание данных (**Create**), выполнение действия (**Action**) или запрос данных (**Query**).

Обзор

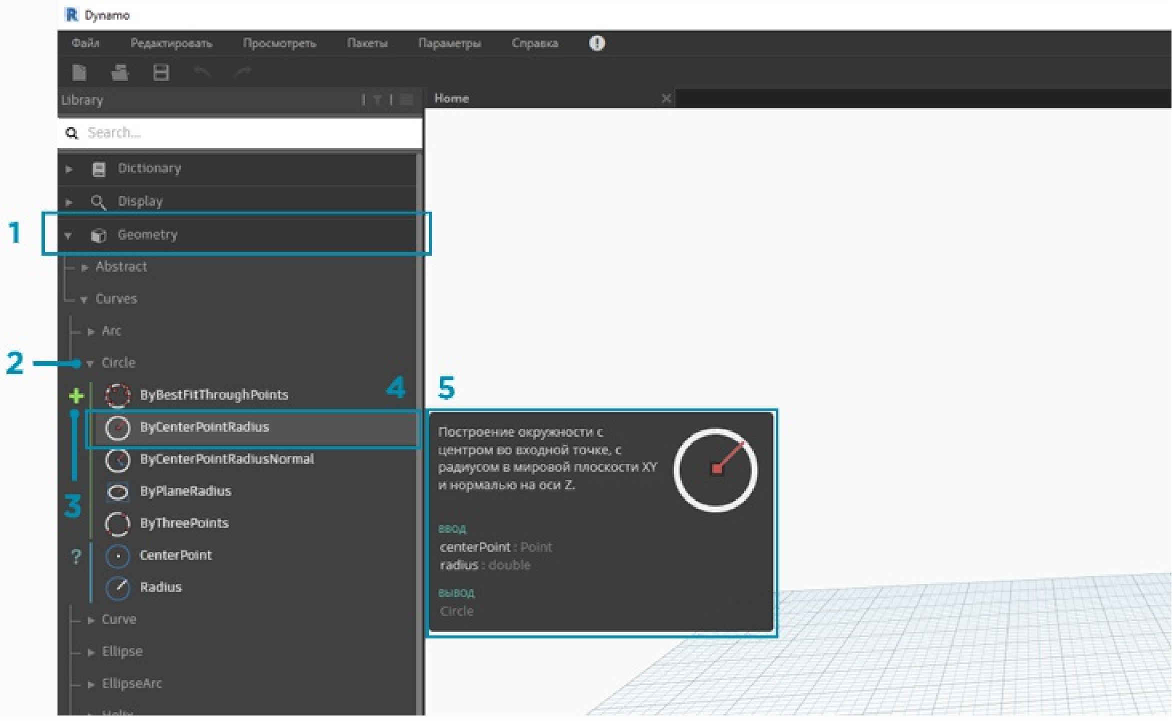
По умолчанию в **библиотеке** доступно восемь категорий узлов. Начинать знакомство с программой рекомендуется с меню **Core** и **Geometry**, в которых представлено наибольшее количество узлов. Обзор узлов по категориям позволяет быстро разобраться в иерархии узлов, которые требуется добавить в рабочее пространство, а также найти узлы, которыми вы еще не пользовались.

Для начала мы рассмотрим стандартный набор узлов, а затем расширим библиотеку за счет пользовательских узлов, дополнительных библиотек и менеджера пакетов.



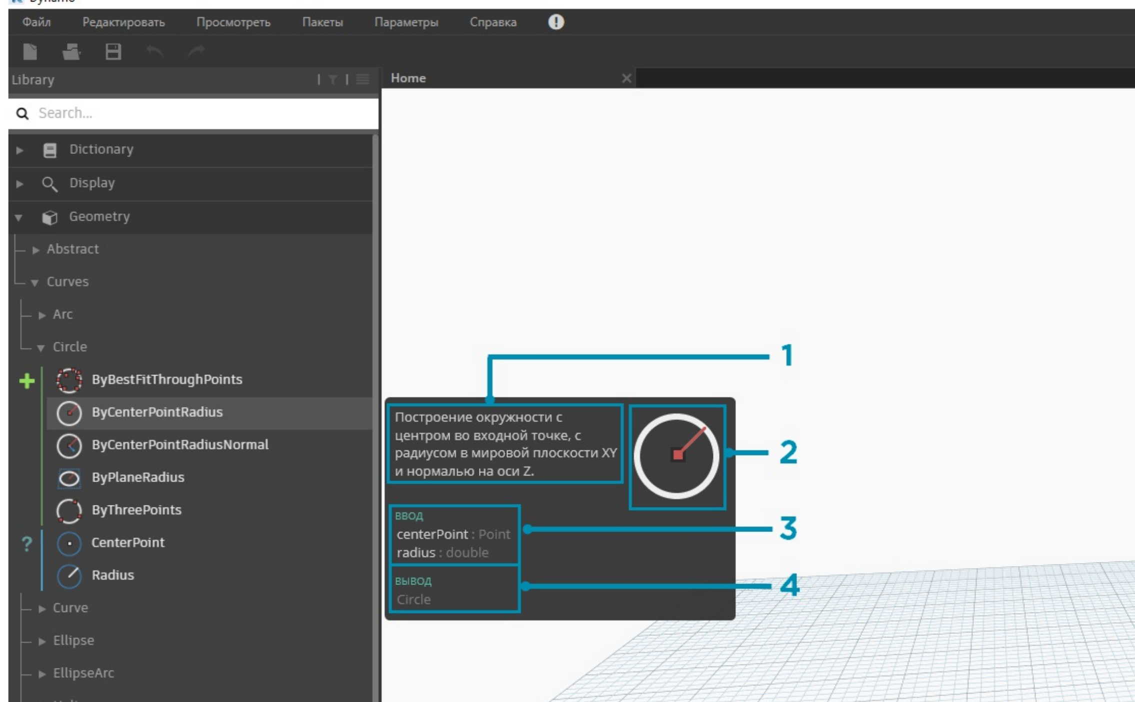
1. Dictionary
2. Display
3. Geometry
4. ImportExport
5. Input
6. List
7. Math
8. Revit
9. Script
10. String
11. Add-ons

Ознакомьтесь с содержимым библиотеки, щелкая различные меню. Выберите Geometry > Curves > Circle. Обратите внимание на новую появившуюся часть меню, а также на метки подкатегорий **Create** и **Query**.



1. Библиотека
2. Категория
3. Подкатегория: Create/Actions/Query
4. Узел
5. Описание и свойства узла: появляется при наведении указателя на значок узла.

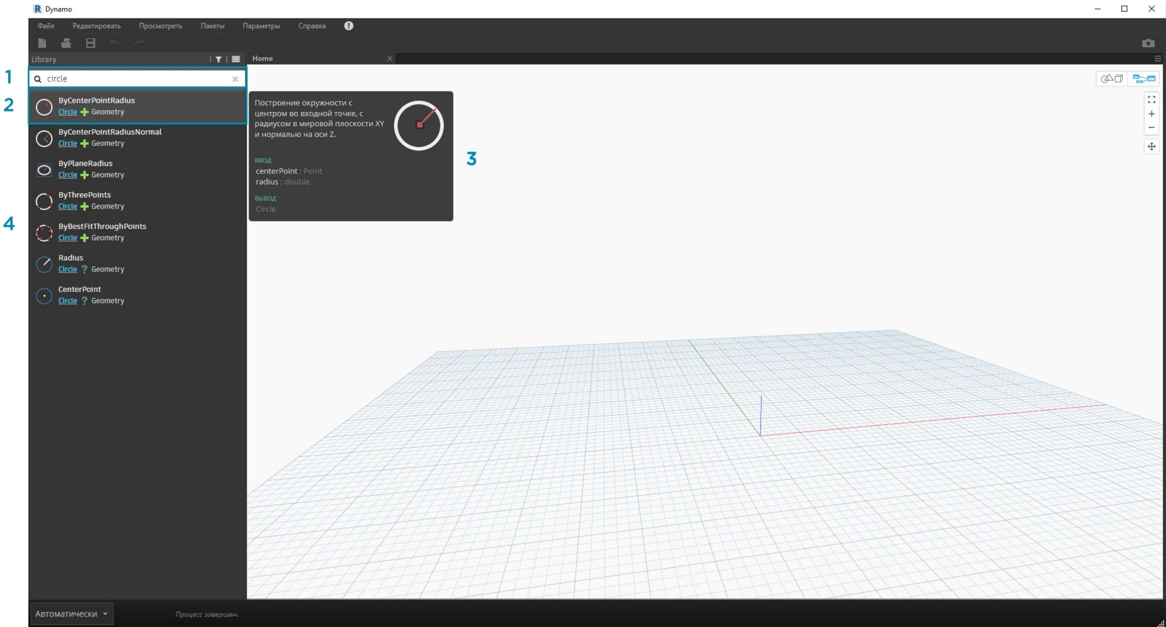
В том же меню Circle наведите указатель на элемент **ByCenterPointRadius**. В окне отобразятся более подробные сведения об узле в дополнение к его имени и значку. Такие подсказки позволяют быстро определить функции, выполняемые узлом, какие данные ему требуются на входе и что он дает на выходе.



1. Описание: описание узла на обычном языке.
2. Значок: увеличенная версия значка, используемого в меню библиотеки.
3. Входные данные: наименование, тип и структура данных.
4. Выходные данные: наименование, тип и структура данных.

Поиск

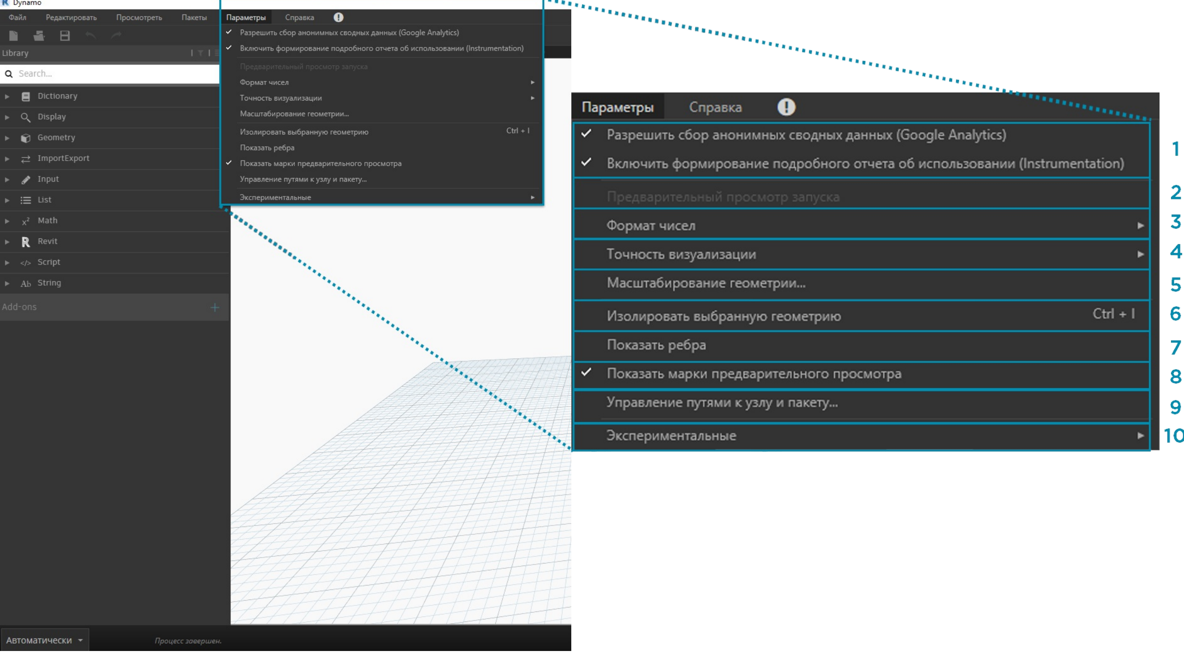
Если вам в общих чертах известно, какой узел требуется добавить в рабочее пространство, можно воспользоваться полем **Поиск**. Курсор всегда находится в этом поле, если только в рабочей области не выполняется редактирование параметров или задание значений. Если начать ввод текста, появится наиболее точное совпадение, найденное в библиотеке Dynamo (с подсказками о расположении объекта в категориях узлов), а также список дополнительных совпадений. При нажатии клавиши ENTER или выборе элемента в компактном обозревателе выделенный узел добавляется в центр рабочего пространства.



1. Поле поиска
2. Наиболее точное / выбранное совпадение
3. Дополнительные совпадения

Параметры

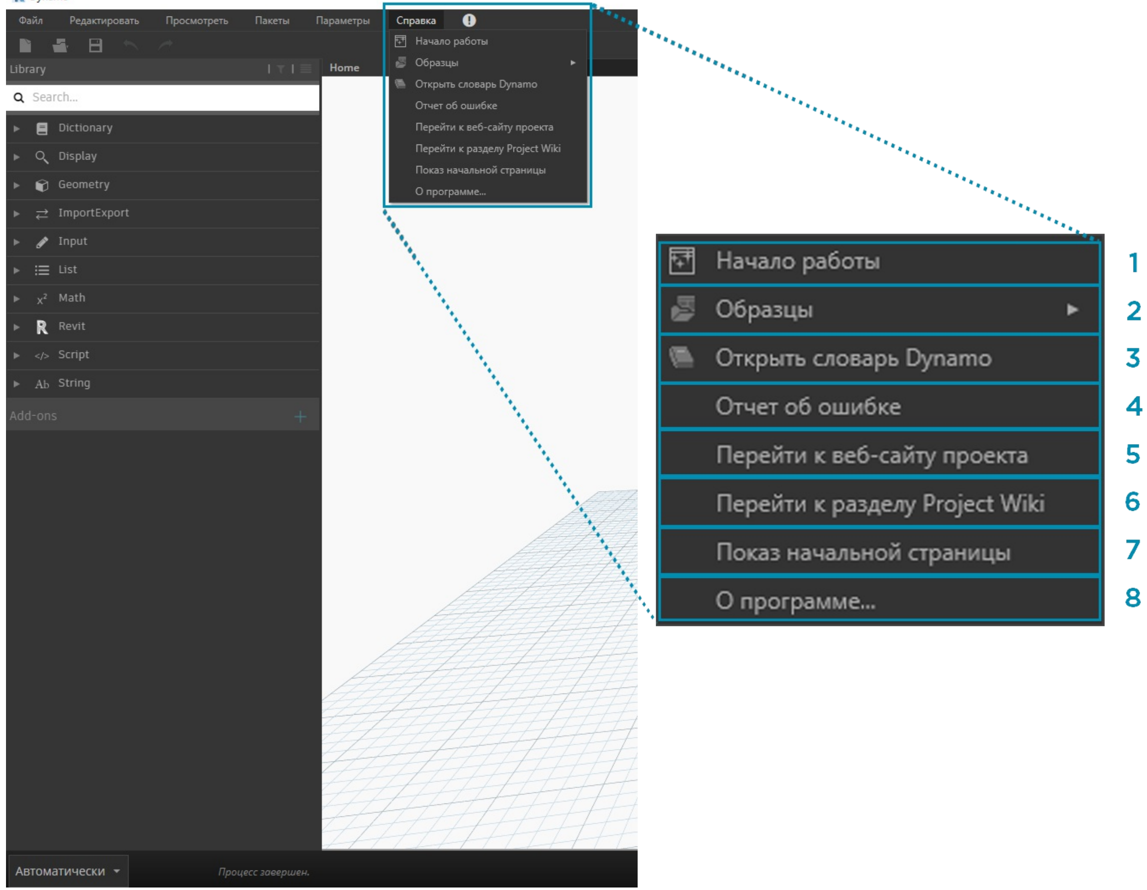
В меню **Параметры** можно найти геометрические, пользовательские и прочие настройки. Здесь можно включить или отключить отправку пользовательских данных для улучшения работы Dynamo, а также задать точность десятичных значений и качество визуализации геометрии.



1. Активация отчетов: параметры для отправки данных пользователя с целью улучшения функций Dynamo.
2. «Предварительный просмотр запуска»: предварительный просмотр состояния графика при выполнении. Используемые при выполнении программы узлы выделяются на графике.
3. Параметры формата чисел: изменение параметров использования десятичных знаков в документе.
4. «Точность визуализации»: повышение или понижение качества визуализации документа.
5. «Масштабирование геометрии»: выбор рабочего диапазона геометрии.
6. «Изолировать выбранную геометрию»: изолирование фоновой геометрия в зависимости от выбранных узлов.
7. Отображение/скрытие ребер геометрии: включение и отключение ребер 3D-геометрии.
8. Отображение/скрытие марок предварительного просмотра: включение и отключение марок предварительного просмотра под узлами.
9. «Управление путями к узлу и пакету»: управление путями к файлам для отображения узлов и пакетов в библиотеке.
10. Включение экспериментальных функций: доступ к бета-версиям функций, недавно появившихся в Dynamo.

Справка

При возникновении вопросов по работе программы можно воспользоваться меню **Справка**. Здесь можно найти файлы примеров, входящие в установочный пакет, а также получить доступ к одному из справочных веб-сайтов Dynamo через веб-браузер. В разделе **О программе** можно выяснить, какая версия Dynamo установлена и является ли она актуальной.



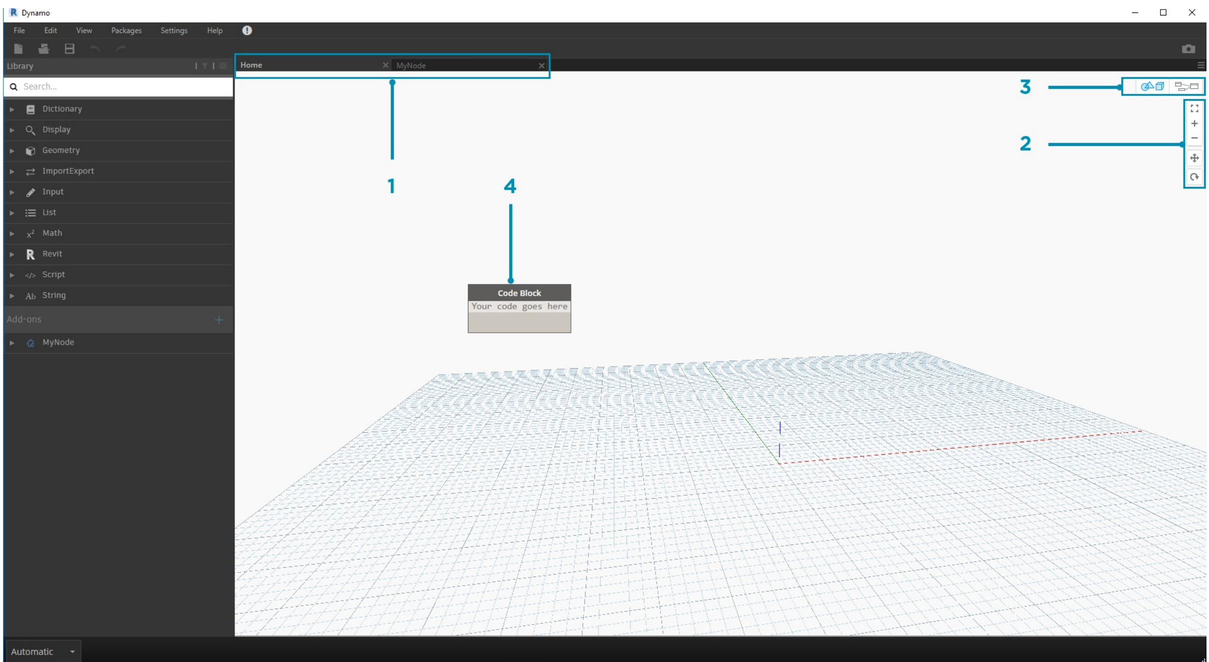
1. «Начало работы»: краткое введение в работу с приложением Dynamo.
2. «Образцы»: файлы примеров для справки.
3. «Открыть словарь Dynamo»: ресурс с документацией по всем узлам.
4. «Отчет об ошибке»: создание инцидента на веб-сайте GitHub.
5. «Перейти к веб-сайту проекта»: просмотр проекта Dynamo на сайте GitHub.
6. «Перейти к разделу Project Wiki»: переход на страницу справки Wiki, посвященную методам разработки с помощью API-интерфейса Dynamo, вспомогательных библиотек и инструментов.
7. «Показ начальной страницы»: возврат на начальную страницу Dynamo при работе с документом.
8. «О программе»: сведения о версии Dynamo.

Рабочее пространство

Рабочее пространство

В **рабочем пространстве** Dynamo осуществляется непосредственная работа над визуальными программами. В нем также можно просмотреть предварительные результаты программирования путем визуализации полученной геометрии. Перемещаться по рабочему пространству, как и по пользовательскому узлу, можно с помощью мыши или кнопок в правом верхнем углу окна приложения. В правом нижнем углу можно выбрать режим просмотра для навигации.

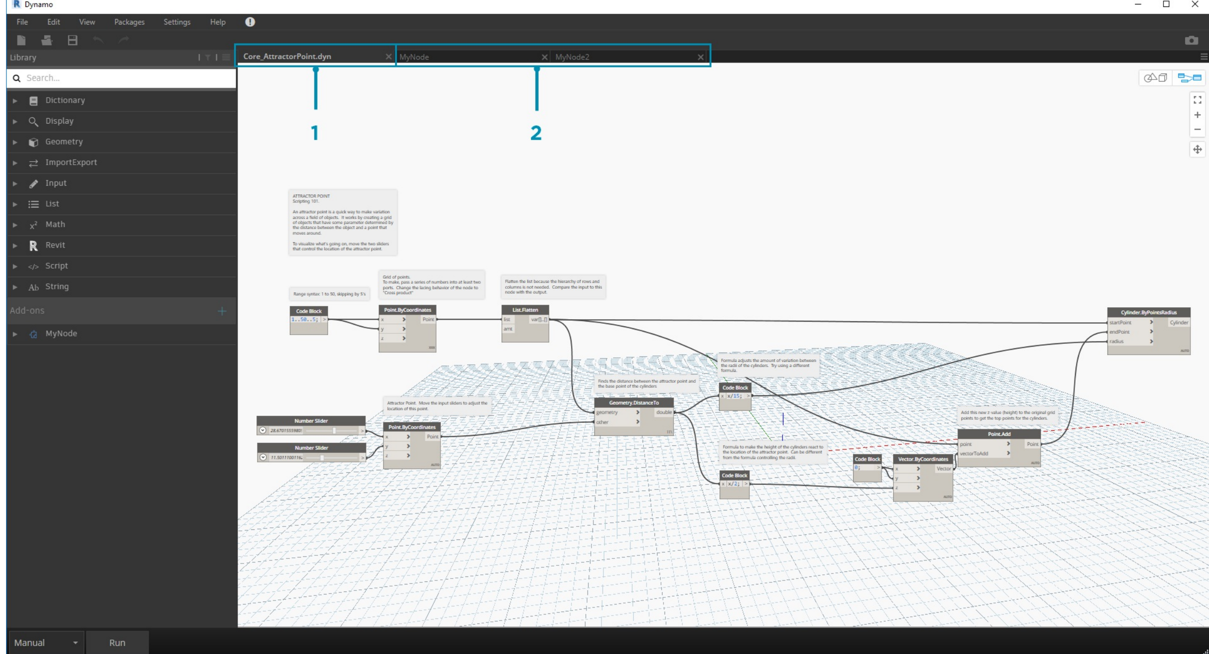
> Примечание. Узлы и геометрия имеют порядок прорисовки, вследствие чего при визуализации объекты могут накладываться друг на друга. Это может произойти при последовательном добавлении нескольких узлов, так как все они могут быть визуализированы в одном и том же месте рабочего пространства.



1. Вкладки
2. Кнопки масштабирования и панорамирования
3. Режим предварительного просмотра
4. Двойной щелчок в рабочем пространстве

Вкладки

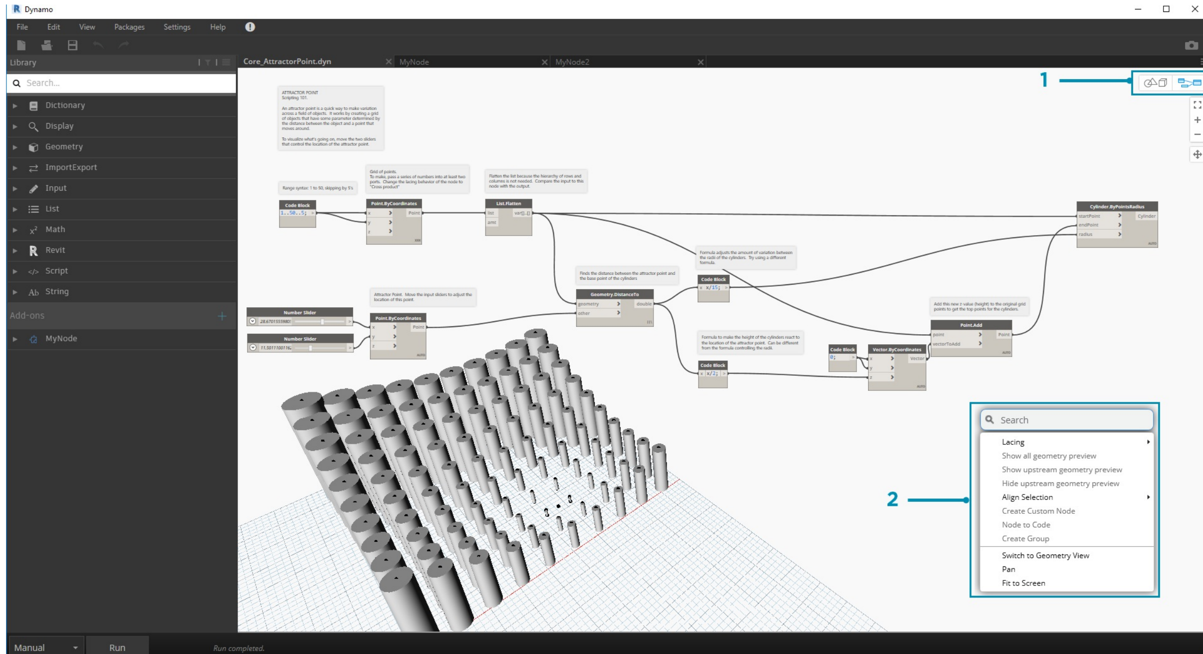
При активной вкладке «Рабочее пространство» можно перемещаться по создаваемой программе и редактировать ее. При открытии нового файла по умолчанию отображается новая **начальная страница рабочего пространства**. Можно также открыть новое **рабочее пространство пользовательского узла**, используя меню «Файл» или указав нужные узлы и выбрав параметр *New Node by Selection* (Создать узел по выбранным объектам) в контекстном меню (подробнее о данной функции будет рассказано в следующих разделах).



Примечание. Одновременно может быть открыта только одна начальная страница рабочего пространства, при этом может быть открыто несколько рабочих пространств пользовательских узлов на дополнительных вкладках.

Навигация по графикам и областям предварительного 3D-просмотра

В Dynamo и графики, и их 3D-результаты (если создаются объекты геометрии) визуализируются в рабочем пространстве. По умолчанию активной областью предварительного просмотра является график. Его панорамирование и масштабирование можно выполнять с помощью кнопок навигации или средней кнопки мыши. Переключаться между активными областями предварительного просмотра можно тремя способами.

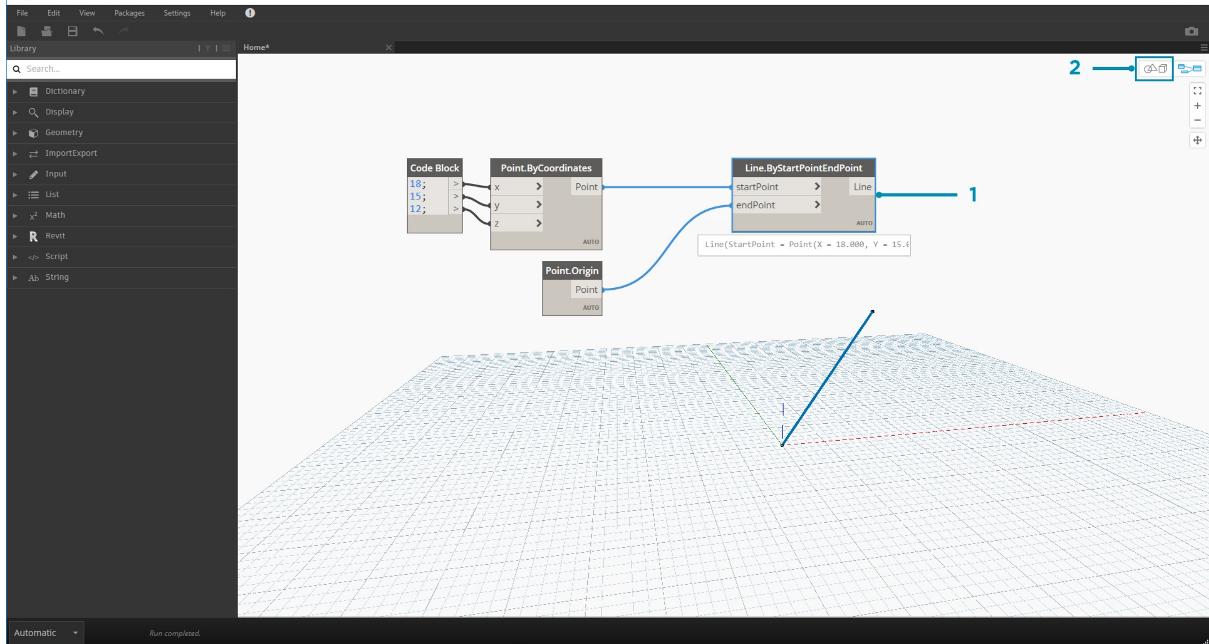


1. Кнопки переключения режима предварительного просмотра в рабочем пространстве
2. Щелчок правой кнопкой мыши в рабочем пространстве и выбор параметра *Перейти к виду ...*
3. Сочетание клавиш (CTRL+B)

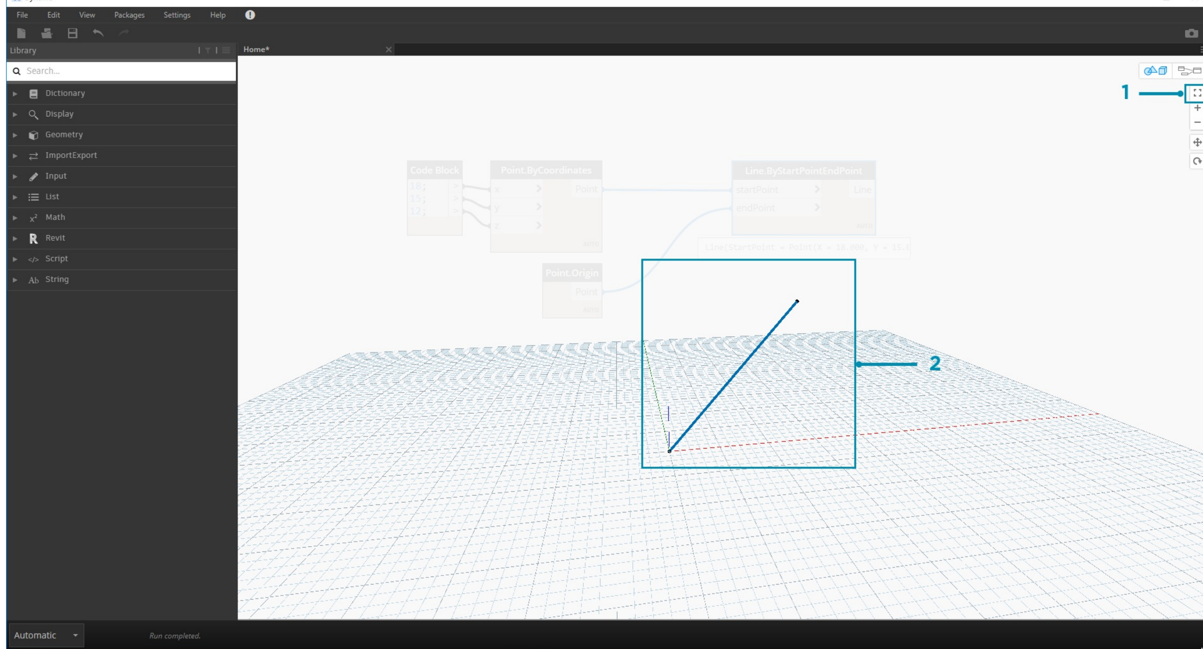
В режиме навигации по области предварительного 3D-просмотра также доступна возможность **непосредственной манипуляции** точками, описанная в разделе Начала работы.

Зумирование по центру

В режиме навигации по области предварительного 3D-просмотра можно свободно перемещать, зумировать и поворачивать модели. Для зумирования определенного объекта, созданного с помощью узла геометрии, можно выбрать отдельный узел, а затем использовать значок «Показать все».



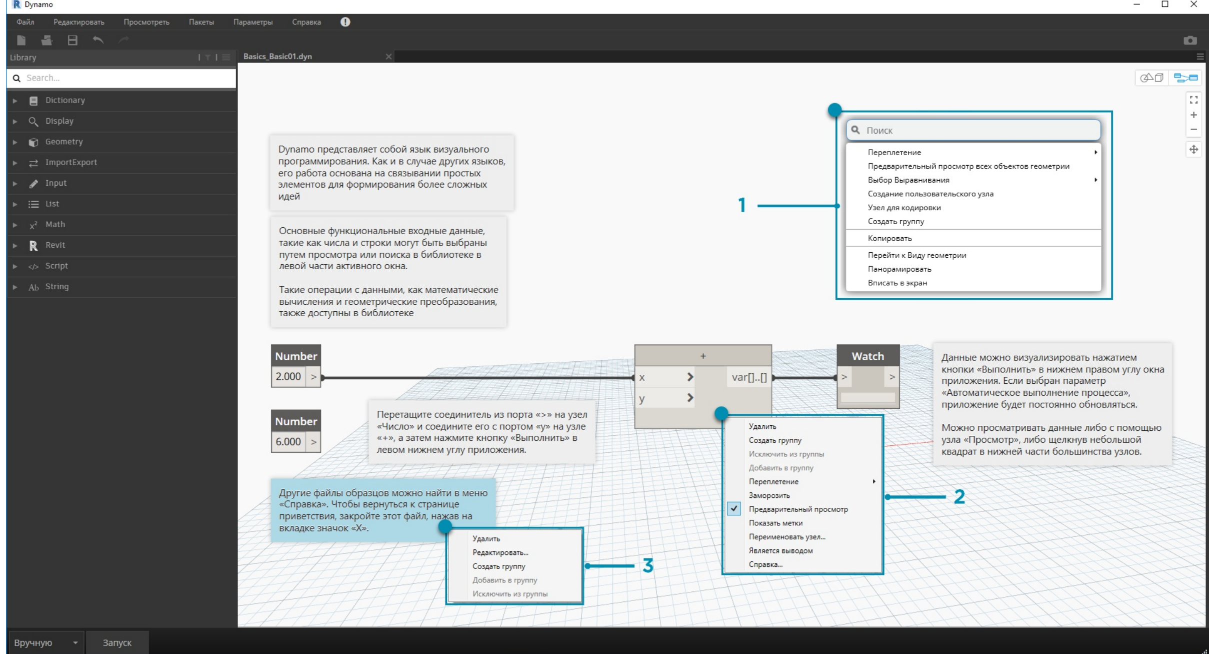
1. Выберите узел, соответствующий геометрии, которая должна располагаться по центру вида.
2. Переключитесь на режим навигации по области предварительного 3D-просмотра.



1. Щелкните значок «Показать все» в правом верхнем углу.
2. Выбранная геометрия будет размещена по центру вида.

Работа с мышью

В зависимости от того, какой режим предварительного просмотра активен, кнопки мыши будут работать по-разному. Обычно левая кнопка мыши позволяет выбирать и задавать входные данные, правая — переходить к параметрам, а средняя — перемещаться по рабочему пространству. Если щелкнуть правой кнопкой мыши, отобразятся контекстные параметры, список которых определяется тем, где был произведен щелчок.



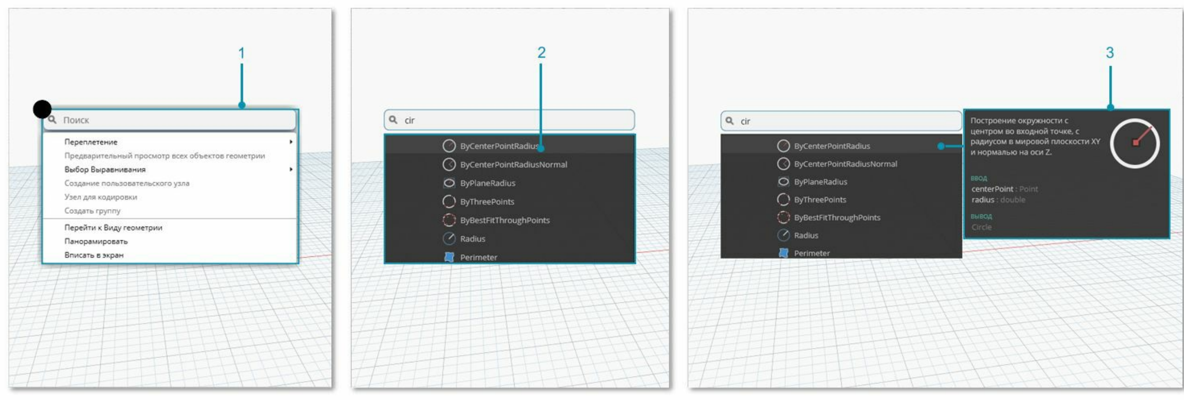
1. Щелчок правой кнопкой мыши в рабочем пространстве
2. Щелчок правой кнопкой мыши по узлу
3. Щелчок правой кнопкой мыши по примечанию

Ниже приведена таблица действий, доступных при использовании мыши в разных режимах просмотра.

Действие мыши	Предварительный просмотр графика	Предварительный 3D-просмотр
Щелчок левой кнопкой	Выбор	Нет
Щелчок правой кнопкой	Контекстное меню	Параметры зумирования
Щелчок средней кнопкой	Панорамирование	Панорамирование
Прокрутка	Зумирование	Зумирование
Двойной щелчок	Создание блока кода	Нет

Поиск в активном окне

Функция поиска в активном окне приложения Dynamo позволяет быстро получать доступ к подсказкам и описаниям узлов независимо от того, над какой частью графика вы работаете. Просто щелкнув правой кнопкой мыши, вы получаете доступ ко всем функциям поиска в библиотеке независимо от того, в какой части рабочей области вы находитесь.

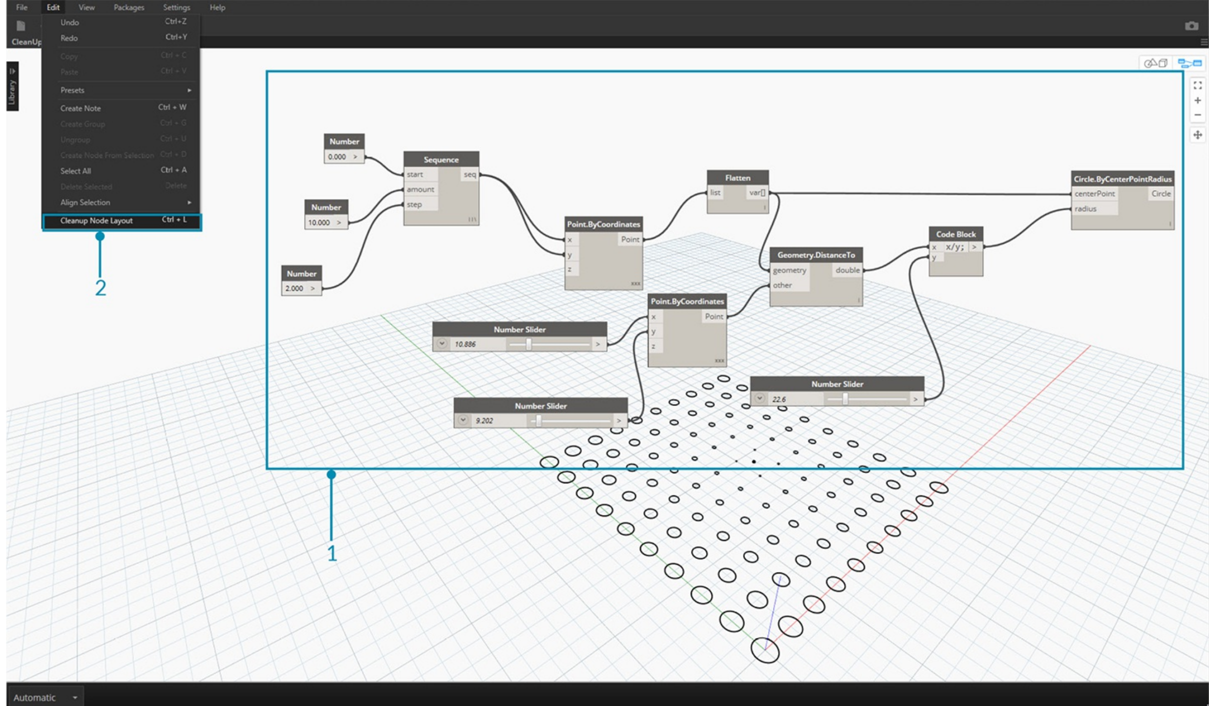


1. Щелкните правой кнопкой в любом месте активного окна, чтобы вызвать функцию поиска. Если строка поиска пуста, отобразится раскрывающееся меню предварительного просмотра.
2. При вводе данных в строку поиска раскрывающееся меню будет обновляться и отображать наиболее подходящие результаты поиска.
3. Наведите указатель на результаты поиска, чтобы вывести соответствующие описания и подсказки.

Очистка компоновки узлов

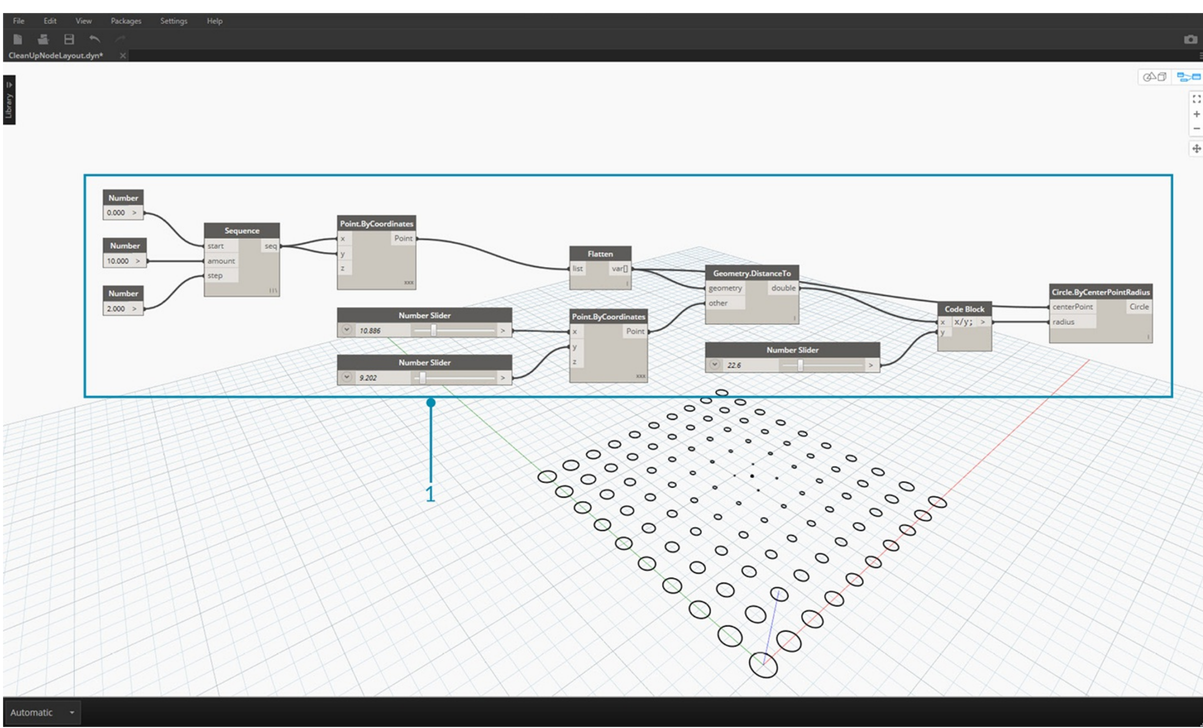
Чем сложнее становятся создаваемые исполняемые файлы, тем важнее становится упорядочение объектов в активном окне Dynamo. Для работы с небольшим числом выбранных узлов в Dynamo используется инструмент **Выбор выравнивания**. Если требуется выполнить очистку по всему файлу, можно воспользоваться инструментом **Очистить компоновку узла**.

Перед очисткой узлов



1. Выберите узлы, которые требуется автоматически упорядочить, или оставьте все узлы невыбранными, чтобы выполнить очистку всех узлов в файле.
2. Функция «Очистить компоновку узла» доступна на вкладке «Редактировать».

После очистки узлов



1. Узлы будут автоматически перераспределены и выровнены, а все смещенные или перекрывающиеся узлы будут очищены и выровнены по соседним узлам.

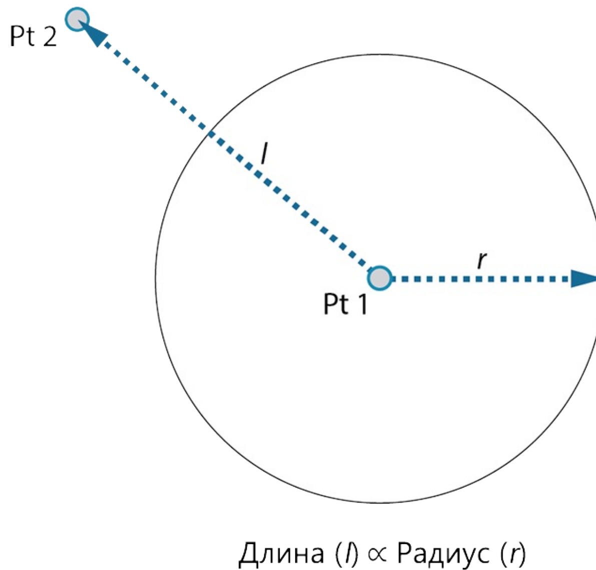
Начало работы

НАЧАЛО РАБОТЫ

Теперь, когда мы ознакомились с интерфейсом и навигацией по рабочему пространству, рассмотрим типовой рабочий процесс создания графиков в Dynamo. Начнем с создания окружности с динамическими размерами, а затем создадим массив окружностей с различными радиусами.

Определение целей и связей

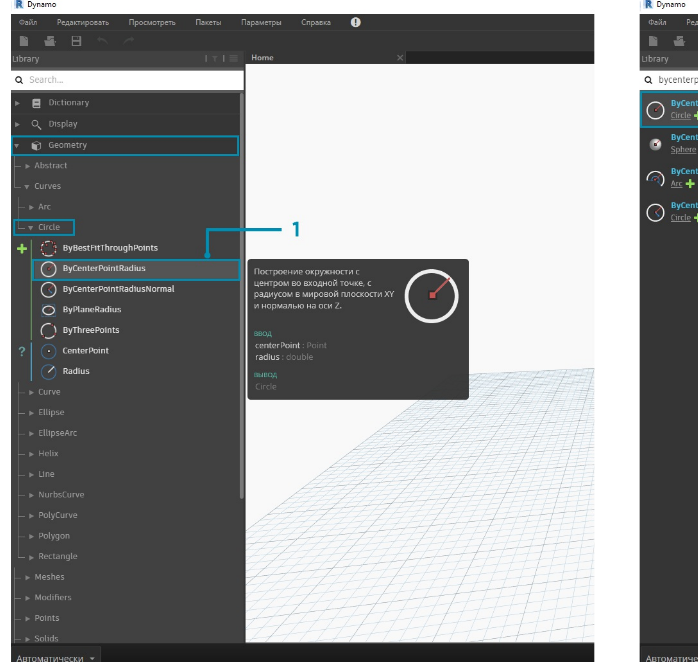
Прежде чем добавлять объекты в рабочее пространство Dynamo, необходимо определить, чего мы пытаемся добиться, и какие связи будут играть наиболее важную роль. Как вы помните, при соединении двух узлов между ними создается явная связь, и хотя направление потока данных впоследствии может измениться, сама связь останется зафиксированной. В рамках этого упражнения требуется создать окружность (*цель*), в которой входные значения радиуса определяются расстоянием до ближайшей точки (*связь*).

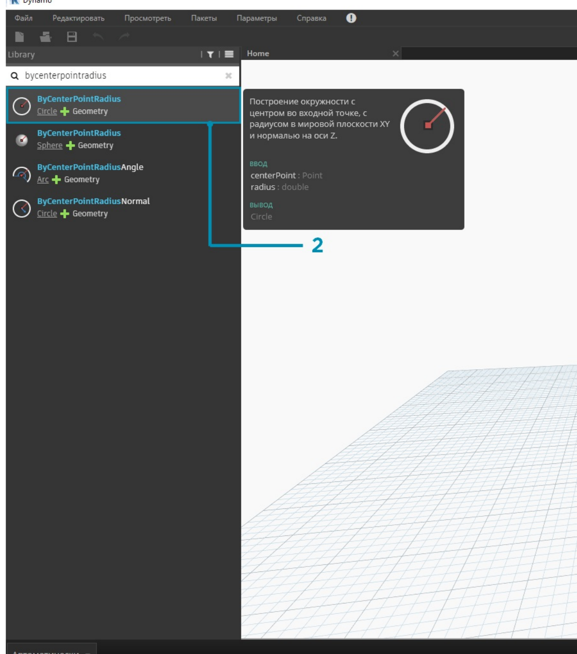


Точка, определяющая отношение на основе расстояния, обычно называется точкой притяжения. В данном случае расстояние до точки притяжения будет использоваться для определения размера окружности.

Добавление узлов в рабочее пространство

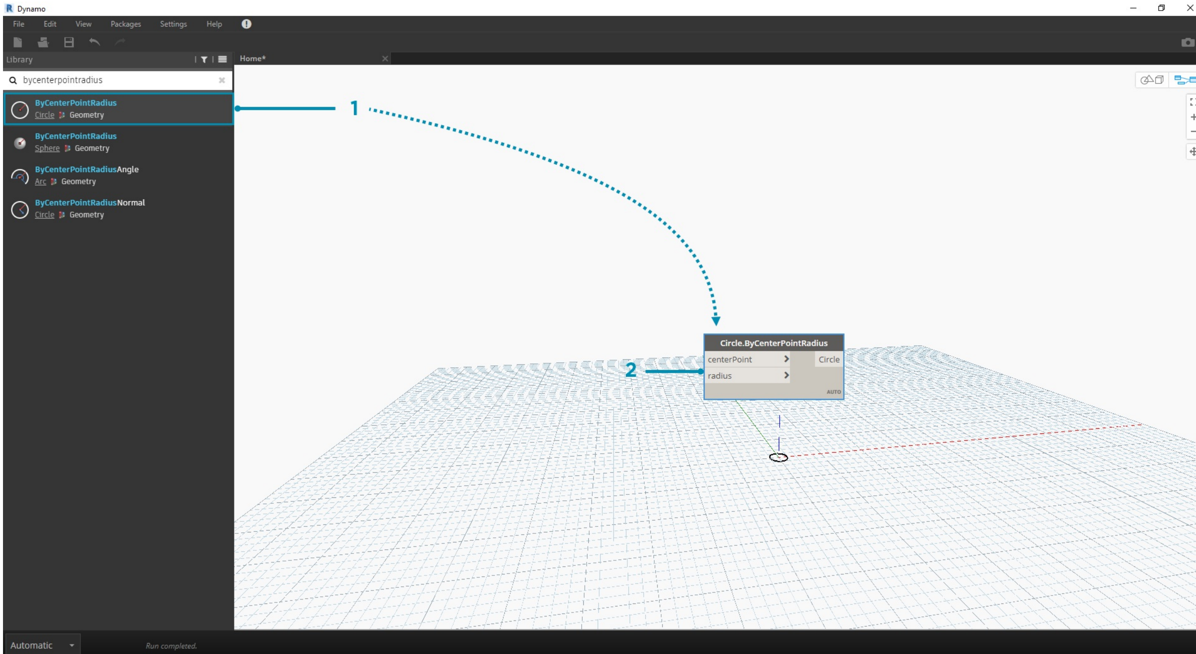
Теперь, когда мы наметили цели и связи, можно приступать к созданию графика. Необходимо выбрать узлы, которые будут представлять последовательность действий, выполняемых приложением Dynamo. Поскольку нам нужно создать окружность, то начнем с поиска узла, позволяющего сделать это. В результате использования поля «Поиск» или обзора библиотеки мы обнаруживаем, что существует несколько способов создания окружности.





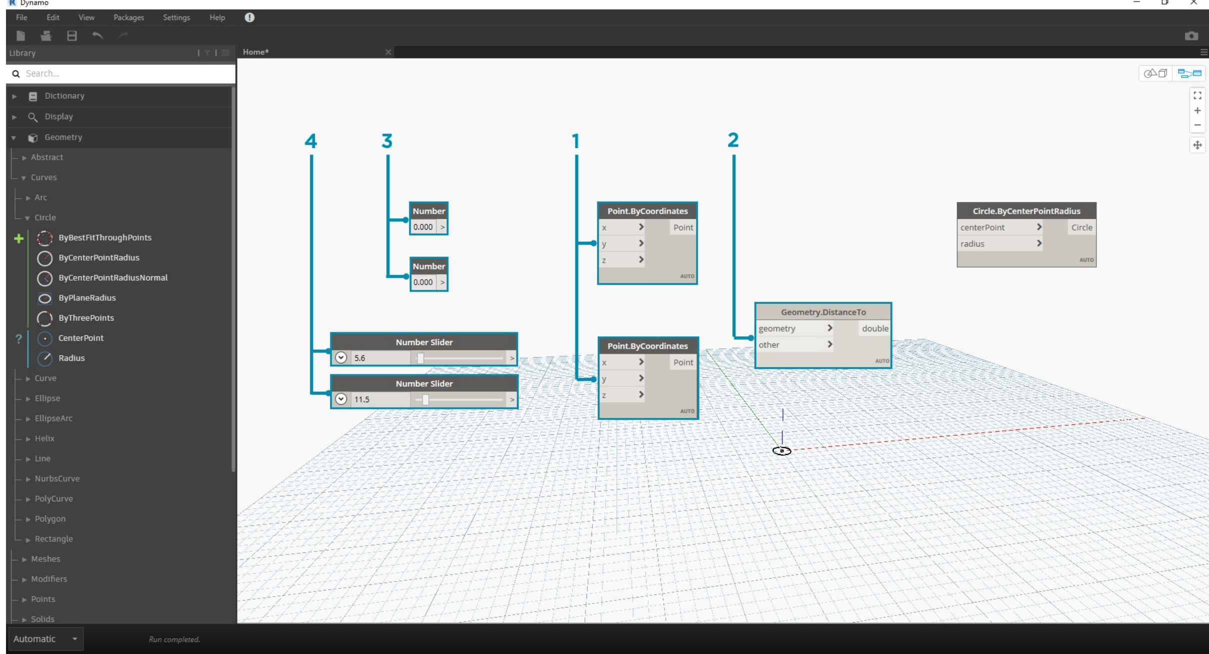
1. Перейдите к разделу Geometry > Curves > Circle > **Circle.ByPointRadius**.
2. Выполните поиск по запросу ByCenterPointRadius.

Выберите узел **Circle.ByPointRadius** в библиотеке, чтобы добавить его в рабочее пространство. Узел должен появиться в центре рабочего пространства.



1. Узел Circle.ByPointandRadius в библиотеке.
2. При выборе узла в библиотеке щелчком мыши он добавляется в рабочее пространство.

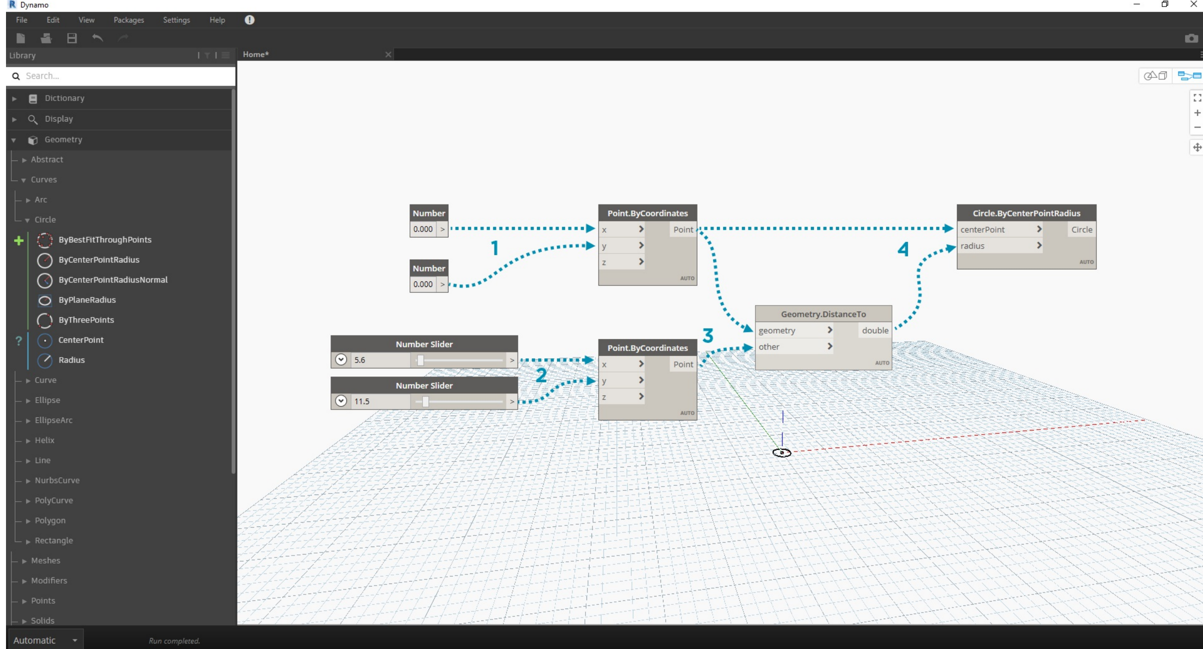
Кроме того, требуются узлы **Point.ByCoordinates**, **Number Input** и **Number Slider**.



1. Geometry > Points > Point > **Point.ByCoordinates**
2. Geometry > Geometry > **DistanceTo**
3. Input > Basic > **Number**
4. Input > Basic > **Number Slider**

Соединение узлов с помощью проводов

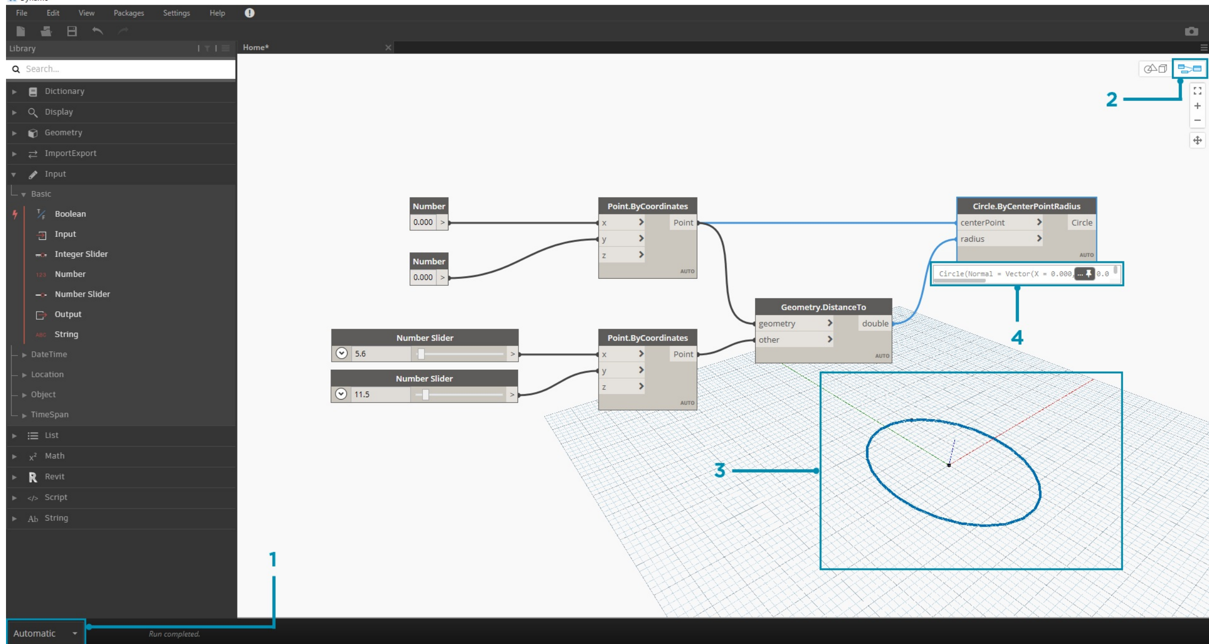
Разместив узлы, необходимо соединить их порты с помощью проводов. Эти соединения будут определять поток данных.



1. От узла **Number** к узлу **Point.ByCoordinates**
2. От узла **Number Sliders** к узлу **Point.ByCoordinates**
3. От узла **Point.ByCoordinates** (2) к узлу **DistanceTo**
4. От узлов **Point.ByCoordinates** и **DistanceTo** к узлу **Circle.ByCenterPointRadius**

Выполнение программы

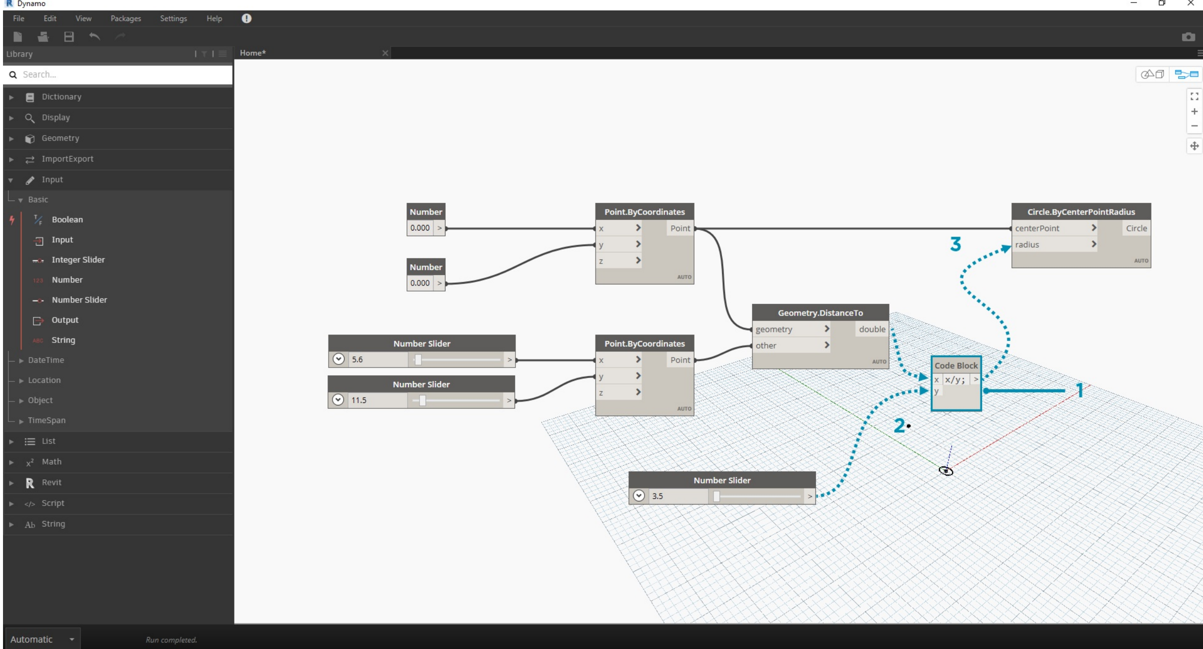
После того как мы определили последовательность потока данных, нам остается только дать Dynamo команду на выполнение программы. После выполнения программы (автоматически или путем нажатия кнопки «Запуск» в ручном режиме) данные пойдут по проводам, а результаты появятся в области 3D-просмотра.



1. Запуск: если на панели выполнения задан ручной режим, нажмите кнопку «Запуск», чтобы запустить выполнение графика.
2. Просмотр узла: при наведении указателя на поле в правом нижнем углу узла появится всплывающее окно результатов.
3. 3D-просмотр: если какой-либо узел создает геометрию, то она будет отображаться в области 3D-просмотра.
4. Выходная геометрия узла создания данных.

Добавление подробностей

Если программа работает правильно, то в области 3D-просмотра должна появиться окружность, проходящая через точку притяжения. Теперь можно добавить дополнительные подробности или элементы управления. Выполним настройку входных портов для узла окружности, чтобы можно было регулировать влияние входных данных на радиус. Добавьте еще один узел **Number Slider** в рабочую область, а затем дважды щелкните в пустом месте рабочего пространства, чтобы добавить узел **Code Block**. Отредактируйте поле в узле Code Block, указав значения X/Y.

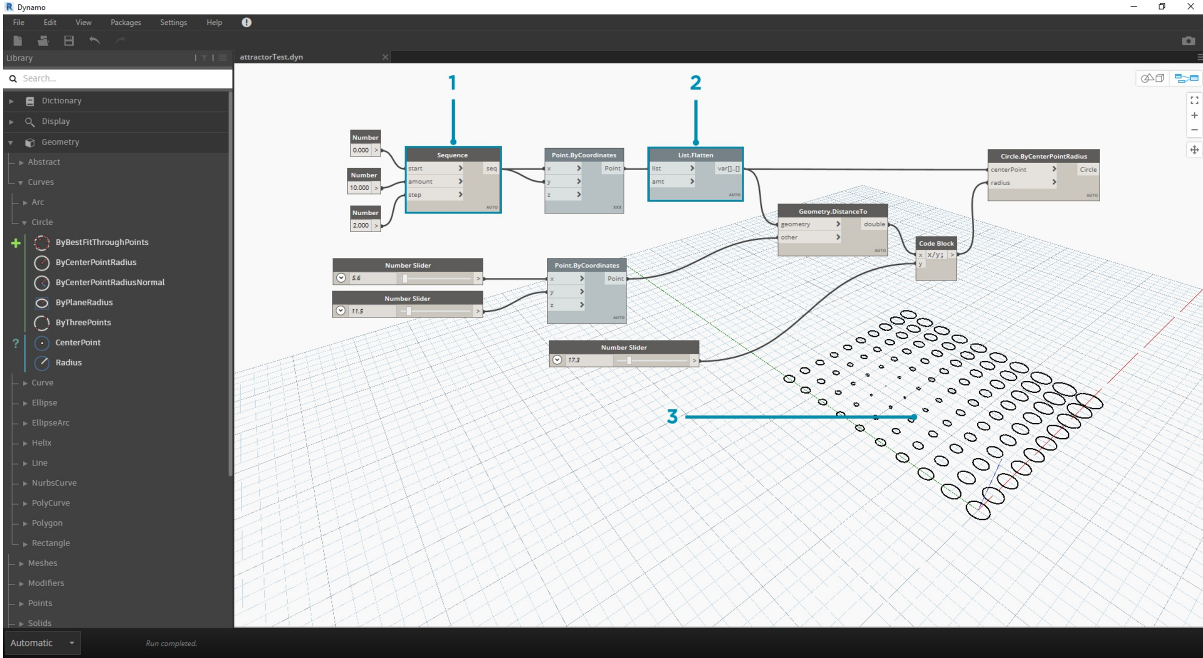


1. **Code Block**
2. От узлов **DistanceTo** и **Number Slider** к узлу **Code Block**
3. От узла **Code Block** к узлу **Circle.ByCenterPointRadius**

Усложнение программы

Чтобы процесс пошаговой разработки программы был эффективен, рекомендуется начинать с простой структуры, которую затем можно постепенно усложнять. Если программа позволяет успешно создавать одну окружность, то ее можно усложнить и использовать для создания сразу нескольких окружностей. Чтобы сделать это, вместо одной центральной точки можно задать сетку точек, данные из которой будут использоваться в итоговой структуре. В результате каждая из полученных окружностей будет иметь уникальное значение радиуса, определяемое

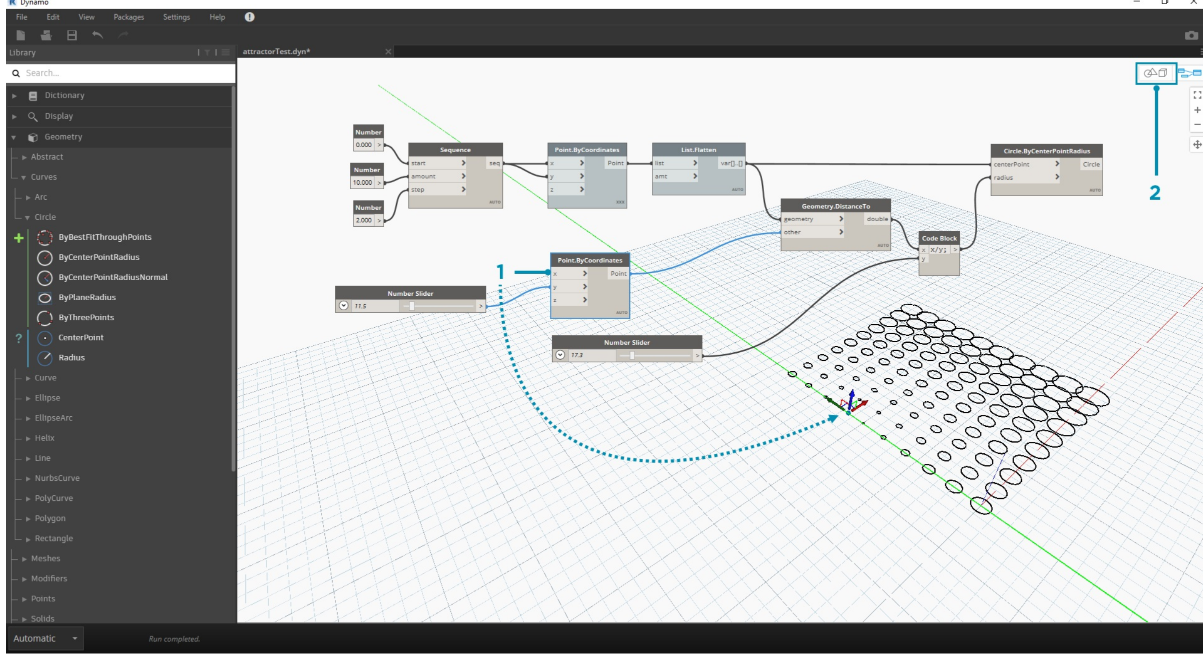
калибруемым расстоянием до точки притяжения.



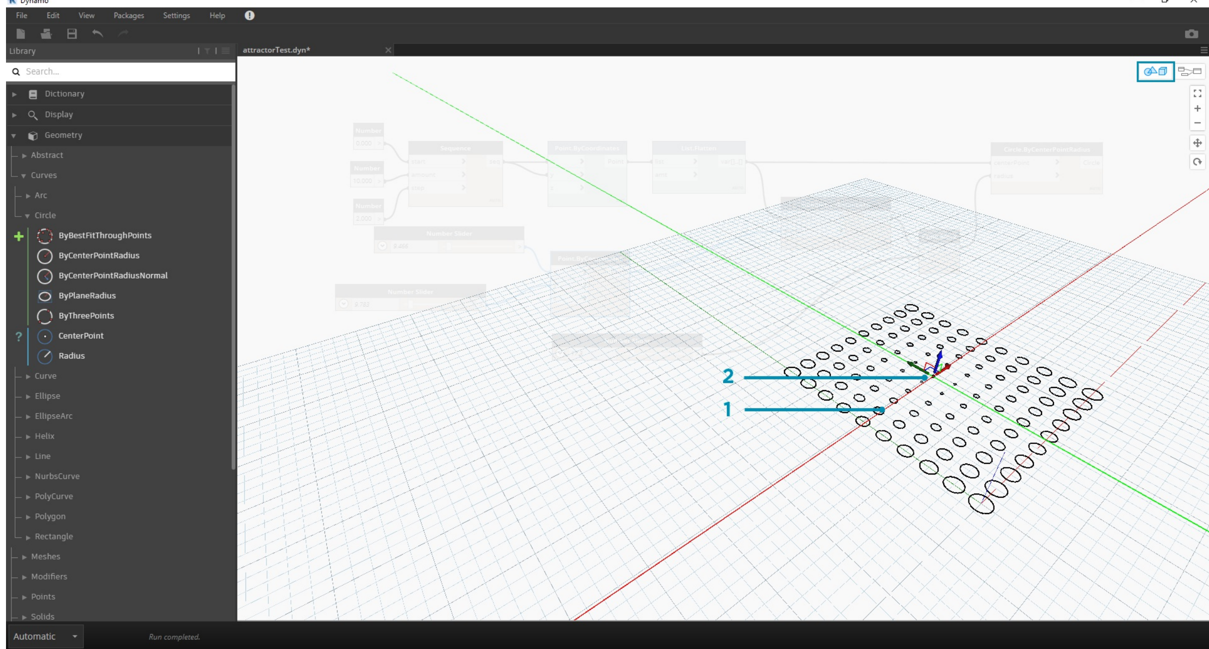
1. Добавьте узел **Number Sequence** и замените входные порты узла **Point.ByCoordinates**. Щелкните узел Point.ByCoordinates правой кнопкой мыши и выберите «Переплетение» > «Перекрестная ссылка».
2. Добавьте узел **Flatten** после узла Point.ByCoordinates. Чтобы выровнять список полностью, оставьте для входного порта `amt` значение по умолчанию (`-1`).
3. Область 3D-просмотра обновляется, и в ней появляется сетка окружностей.

Настройка с помощью непосредственных манипуляций

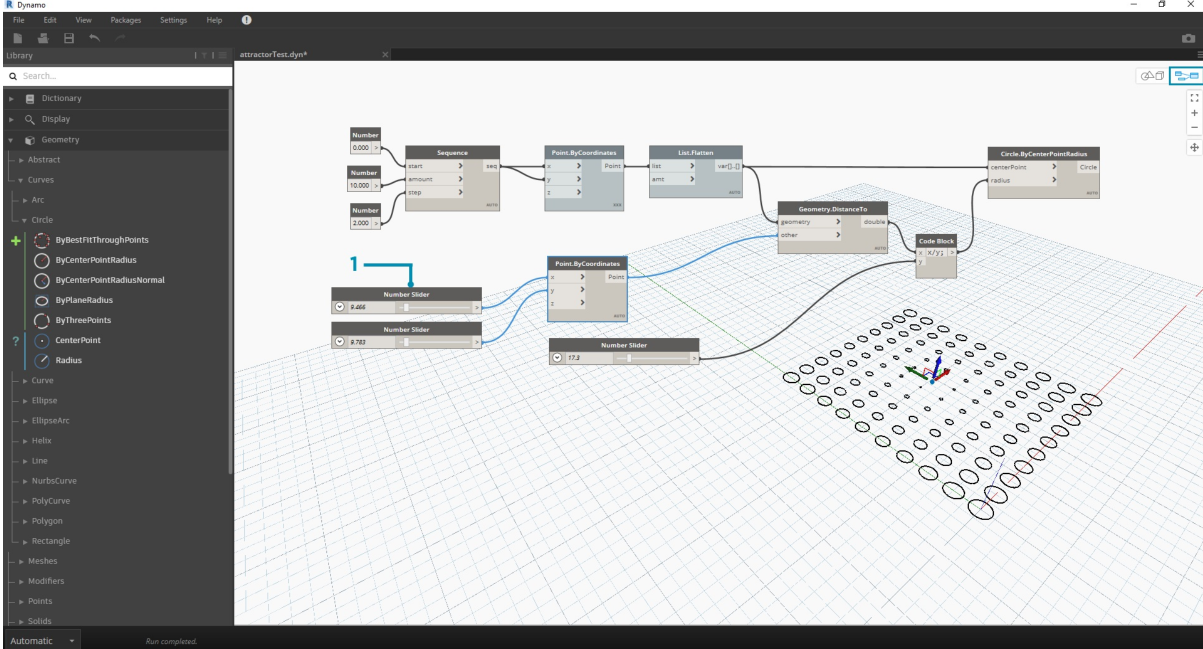
Манипуляции с числами не всегда являются оптимальным подходом. Теперь геометрию точки можно корректировать вручную при навигации по области фонового 3D-просмотра. Можно также управлять другой геометрией, построенной на основе точки. Например, узел **Sphere.ByCenterPointRadius** поддерживает режим непосредственной манипуляции. Можно управлять положением точки, задавая наборы значений X, Y и Z с помощью узла **Point.ByCoordinates**. При использовании метода непосредственных манипуляций можно обновлять значения регуляторов путем перемещения точки вручную в режиме **навигации по области 3D-просмотра**. Это обеспечивает более интуитивный способ управления набором отдельных значений, которые определяют положение точки.



1. Для использования метода **Непосредственная манипуляция** выберите панель перемещаемой точки. Над выбранной точкой появятся стрелки.
2. Переключитесь в режим **Навигация по области 3D-просмотра**.



1. Наведите указатель на точку. Появятся оси X, Y и Z.
2. Чтобы переместить ось, щелкните и перетащите соответствующую цветную стрелку. При перемещении точки вручную значения в узле **Number Slider** динамически обновляются.

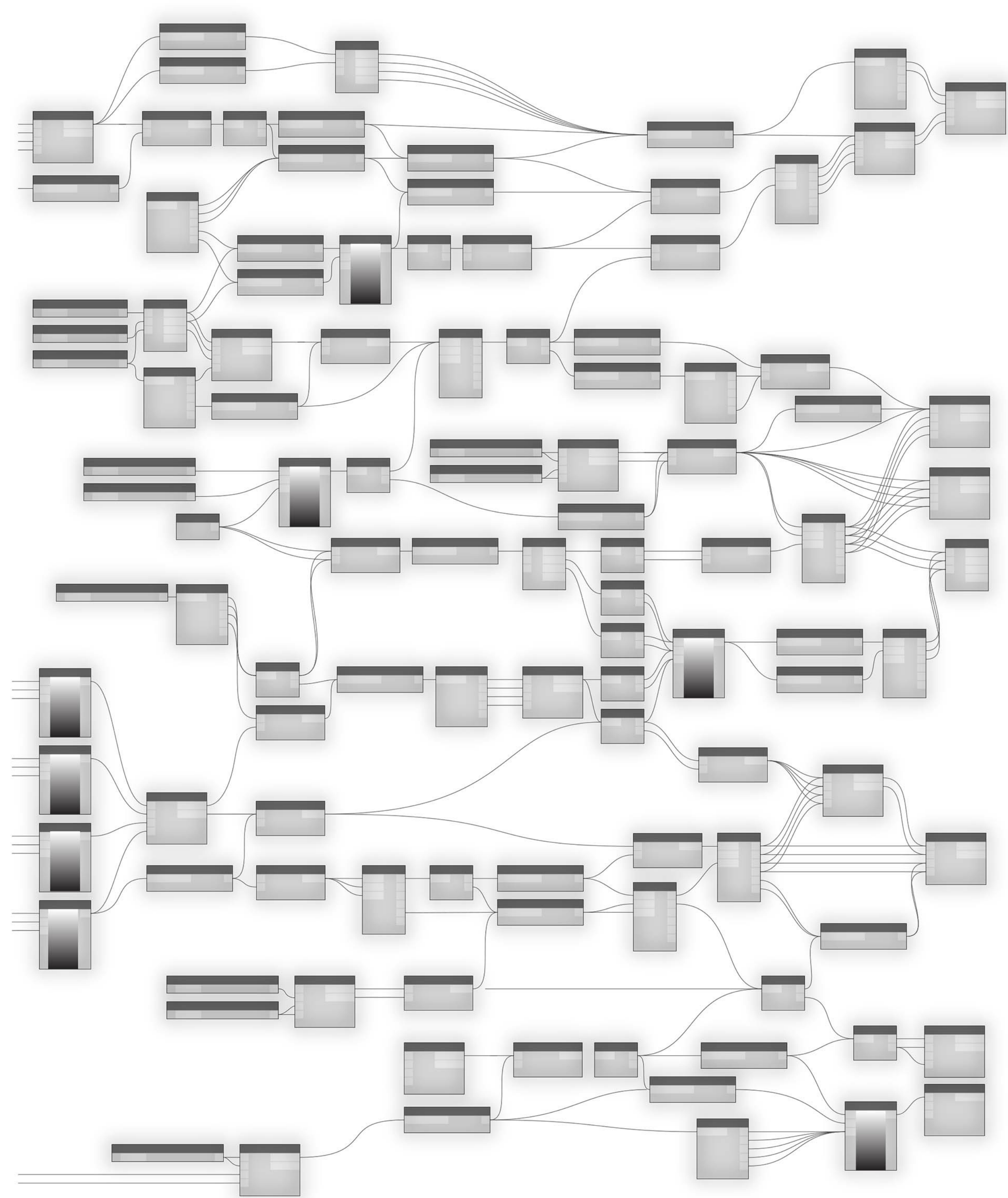


1. Обратите внимание, что до перехода в режим **непосредственной манипуляции** с компонентом **Point.ByCoordinates** был соединен только один регулятор. При перемещении точки в направлении по оси X вручную автоматически создается новый узел **Number Slider** для указания входных данных по оси X.

Структура визуальных программ

СТРУКТУРА ВИЗУАЛЬНЫХ ПРОГРАММ

В приложении Dynamo можно создавать визуальные программы, определяя логическую последовательность операции путем соединения узлов проводами в рабочем пространстве. В этой главе описываются элементы визуальных программ, организация узлов, доступных в библиотеках Dynamo, их компоненты и возможные состояния, а также приводятся практические рекомендации по использованию рабочих пространств.



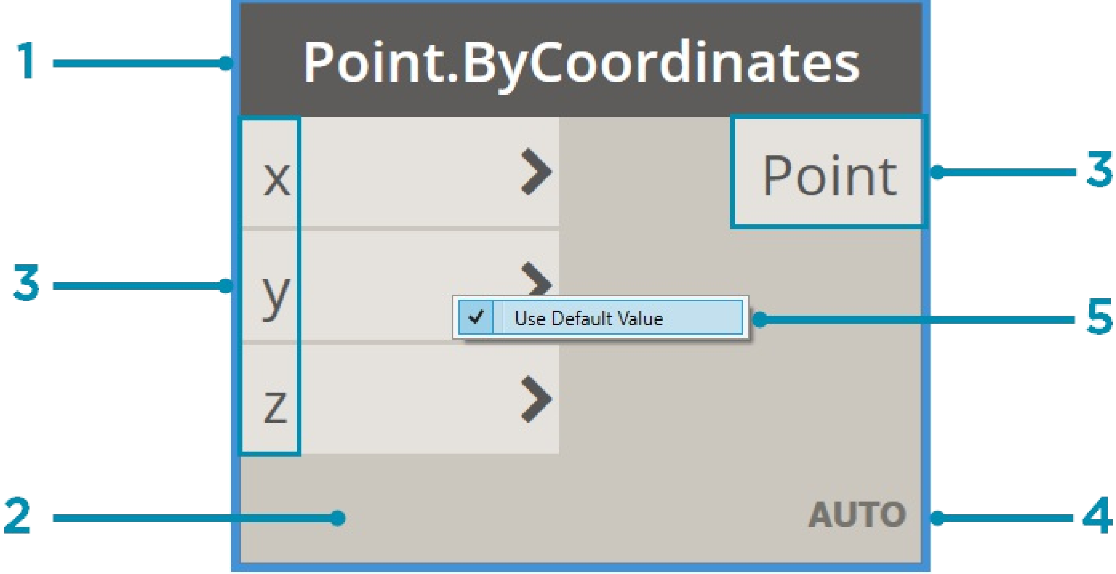
Узлы

Узлы

Узлы Dynamo — это объекты, путем соединения которых создается визуальная программа. Каждый **узел** выполняет ту или иную операцию. Это может быть как простая операция, например хранение числа, так и более сложная, например создание или запрос геометрического объекта.

Структура узла

Большинство узлов Dynamo состоит из пяти частей. За некоторыми исключениями (такими как узлы ввода) узлы в большинстве своем устроены следующим образом:

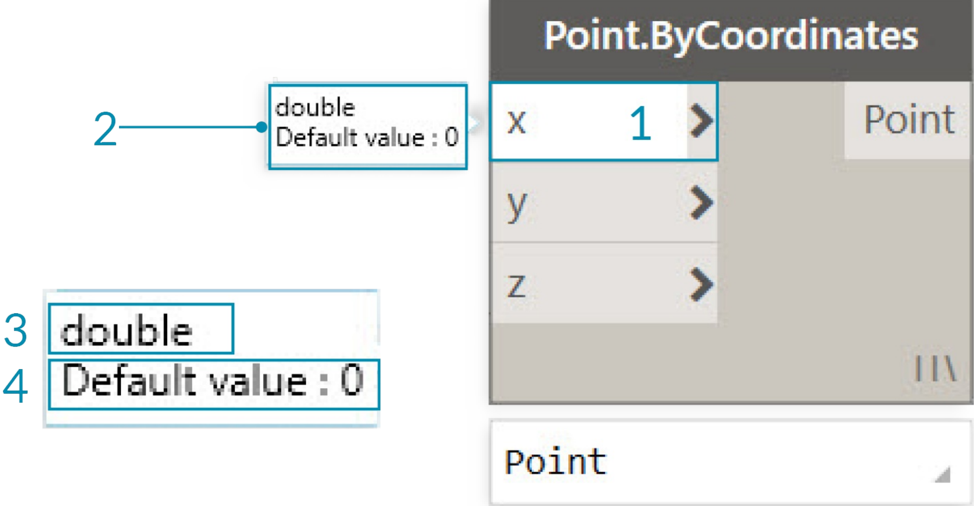


1. Имя: имя узла, составленное по шаблону «Категория.Имя».
2. Основная часть узла: если щелкнуть ее правой кнопкой мыши, отобразятся параметры, действующие на уровне узла.
3. Порты (ввода и вывода): разъемы для проводов, передающих входные данные для узла, а также результаты выполненной узлом операции.
4. Значок переплетения: значение параметра «Переплетение», заданное для совпадающих входных данных списка (подробные сведения см. далее).
5. Значение по умолчанию: щелкните порт ввода правой кнопкой мыши. Для некоторых узлов заданы значения по умолчанию, которые можно использовать или игнорировать.

Порты

Порты — это вводы и выводы узлов, играющие роль разъемов для проводов. В порты, расположенные слева, поступают входящие данные, а из портов, расположенных справа, передаются далее результаты выполненной операции. Каждый порт рассчитан на прием данных определенного типа. Если соединить с портами узла Point.ByCoordinates число, например *2.75*, то операция выполнится успешно и будет создана точка. Но если вместо числа соединить с тем же портом, например, текстовое значение *Red*, это приведет к ошибке.

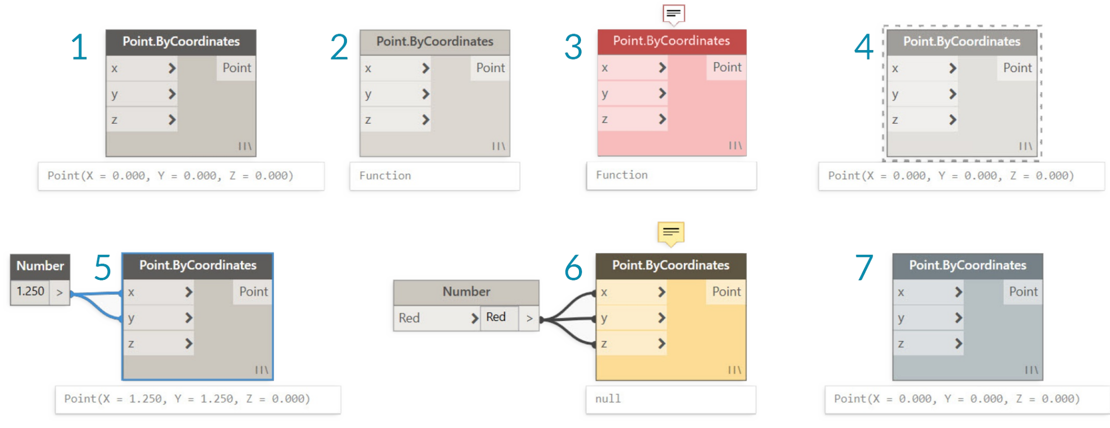
Совет. Наведите указатель на порт, чтобы увидеть подсказку о требуемом типе данных.



1. Метка порта
2. Подсказка
3. Тип данных
4. Значение по умолчанию

Состояния

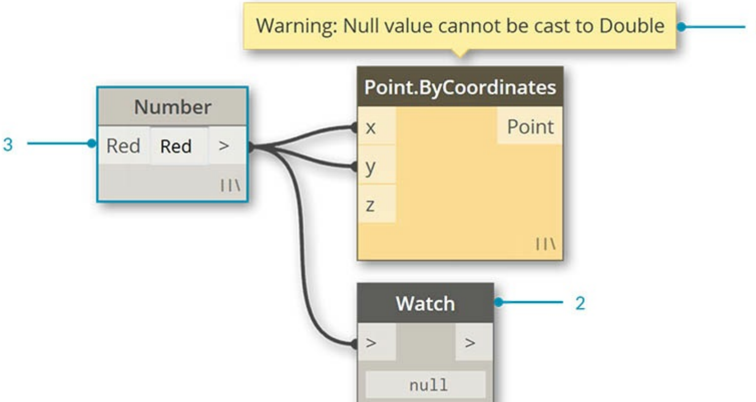
Для демонстрации состояния выполнения операций в узлах визуальной программы в Dynamo используются разные цвета. Кроме того, наведя указатель на имя узла или его порты либо щелкнув их правой кнопкой мыши, можно получить доступ к дополнительным сведениям и параметрам.



1. Активный: если узлы правильно подключены и в них поступают входные данные нужного типа, то их имя отображается на темно-сером фоне.
2. Неактивный: узлы серого цвета неактивны и должны быть подключены к другим узлам программы в активном рабочем пространстве с помощью проводов.
3. Ошибка: красный цвет указывает на то, что в работе узла произошла ошибка.
4. Замороженный: если узел заморожен, выполнение его операции приостанавливается, и он становится прозрачным.
5. Выбранный: выбранный узел выделяется голубой рамкой.
6. Предупреждение: если есть вероятность, что в узел поступают данные неверного типа, он отображается желтым цветом.
7. Фоновый просмотр: если узел отображается темно-серым цветом, это значит, что просмотр геометрии отключен.

Если визуальная программа содержит предупреждения или ошибки, то Dynamo предоставляет подробную информацию о проблеме. Над именем каждого желтого узла отображается подсказка. Наведите указатель на подсказку, чтобы развернуть ее.

> Совет. Используя информацию из подсказки, проверьте узлы, предшествующие текущему, на наличие ошибок в типе или структуре требуемых данных.



1. Подсказка с предупреждением: не заданные данные или значение Null не могут использоваться как данные типа Double, например число.
2. Используйте узел Watch, чтобы просмотреть входные данные.
3. Узел Number, предшествующий текущему, передает на выходе текстовое значение Red, а не число.

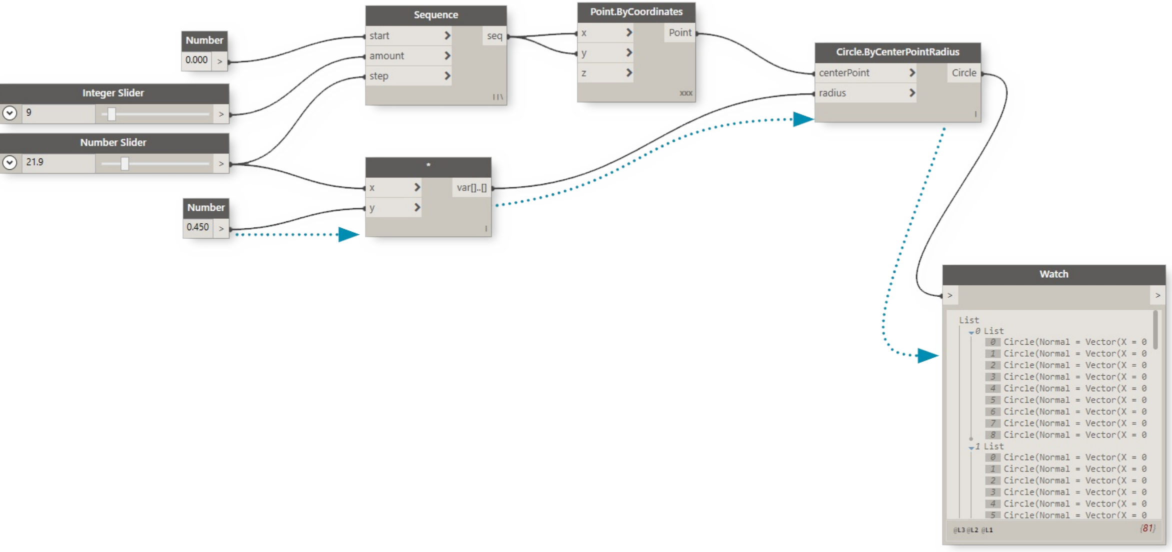
Каркасы

Провода

Провода соединяют друг с другом узлы, создавая тем самым связи и обеспечивая поток выполнения операций в рамках визуальной программы. Их можно воспринимать как настоящие электрические провода, передающие заряды (данные) от одного объекта к другому.

Поток выполнения операций в программе

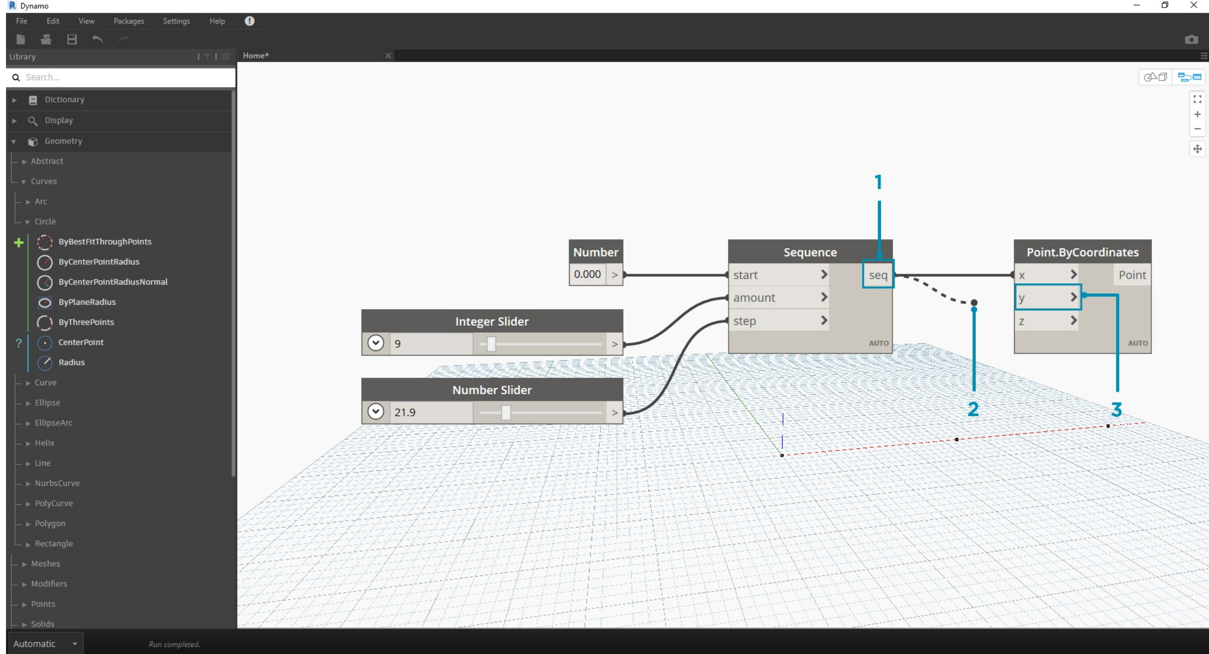
Каждый порт соединяет порт вывода одного узла с портом ввода другого. Такой порядок подключения определяет направление **потока данных** в визуальной программе. Узлы можно расположить в рабочем пространстве как хочется, но учитывая, что порты ввода находятся слева, а порты вывода — справа, можно сказать, что поток выполнения операций в программе направлен слева направо.



Создание проводов

Для создания провода щелкните сначала порт одного узла, а затем порт другого узла, чтобы установить между ними соединение. В процессе создания соединения провод отображается пунктирной линией, после чего становится сплошным. Данные всегда проходят по проводам в направлении от вывода к вводу, однако провод можно создавать в любом направлении в последовательности щелчков соединенных портов.

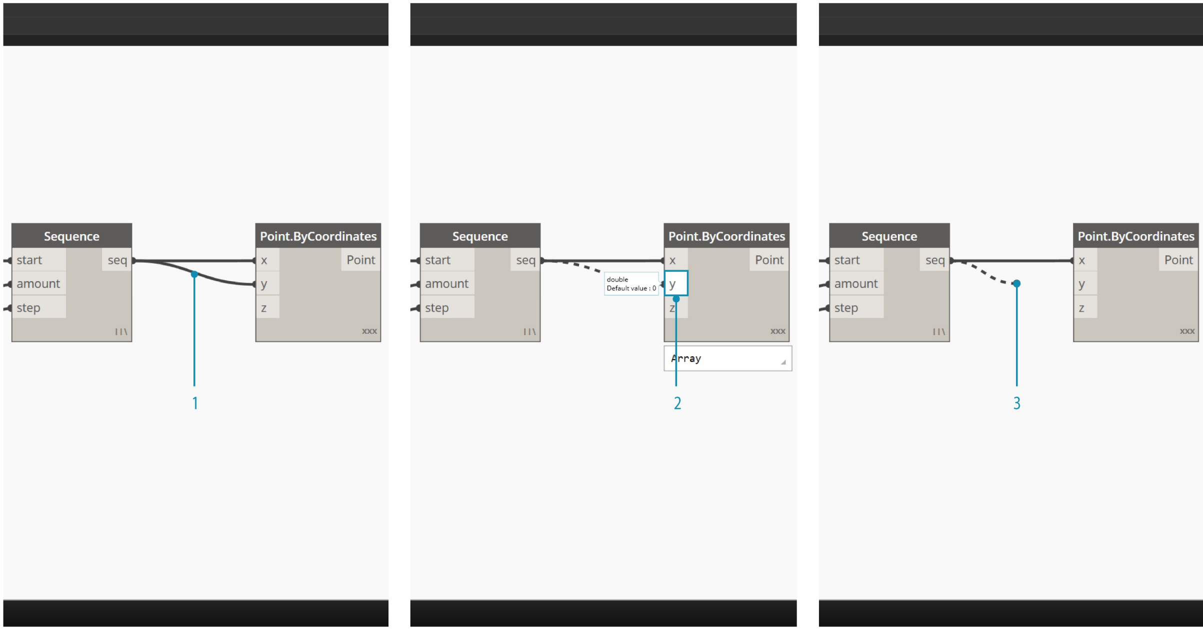
> Совет. Прежде чем щелкнуть левой кнопкой мыши, чтобы завершить создание соединения, дождитесь, когда провод сам присоединится к порту узла, и наведите указатель на этот порт, чтобы посмотреть подсказку.



1. Щелкните порт вывода `seq` узла Sequence.
2. Пока указатель перемещается к другому порту, провод отображается пунктирной линией.
3. Щелкните порт ввода `y` узла Point.ByCoordinates, чтобы завершить создание соединения.

Редактирование проводов

Зачастую при работе над визуальной программой возникает необходимость в корректировке потока выполнения операций путем редактирования проводов, играющих роль соединительных элементов. Чтобы отредактировать провод, щелкните порт ввода подсоединенного узла. Далее выберите один из двух вариантов:

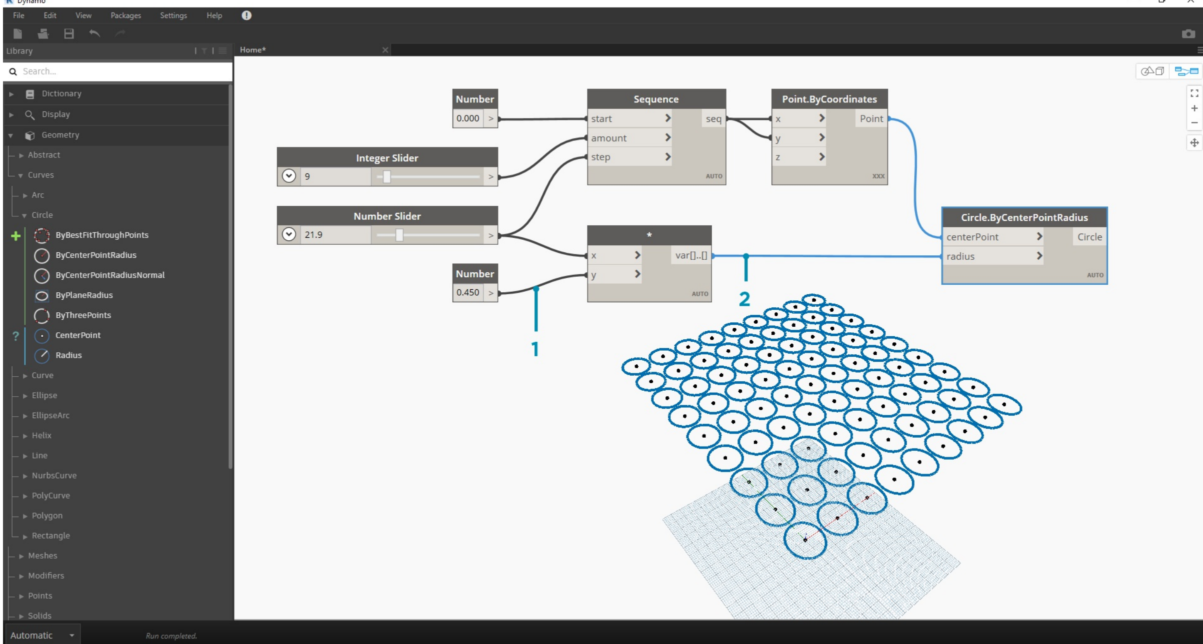


1. Существующий провод.
2. Чтобы изменить подключение к порту ввода, щелкните другой порт ввода.
3. Чтобы удалить провод, перетащите его в сторону и щелкните в рабочем пространстве.

- Примечание. Доступна дополнительная возможность, позволяющая одновременно перемещать несколько проводов. Подробные сведения см. здесь: http://dynamobim.org/dynamo-1-3-release/.

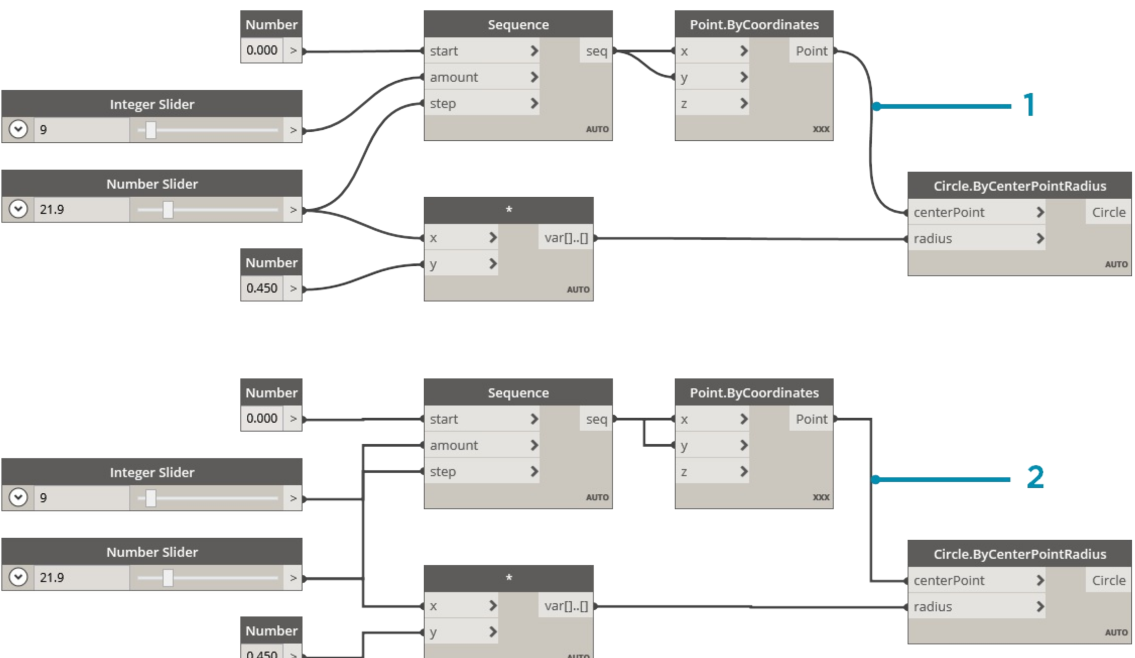
Предварительный просмотр проводов

По умолчанию провода отображаются в режиме предварительного просмотра как серые прерывистые линии. При выборе узла все подключенные к нему провода выделяются тем же синим цветом, что и узел.



1. Провод по умолчанию
2. Выделенный провод

Dynamo также позволяет настроить отображение проводов в рабочем пространстве с помощью меню «Просмотреть» > «Соединители». В этом меню можно выбрать провода в виде полилиний или кривых либо полностью отключить их отображение.



1. Тип соединителя: кривые
2. Тип соединителя: полилинии

Библиотека

Библиотека Dynamo

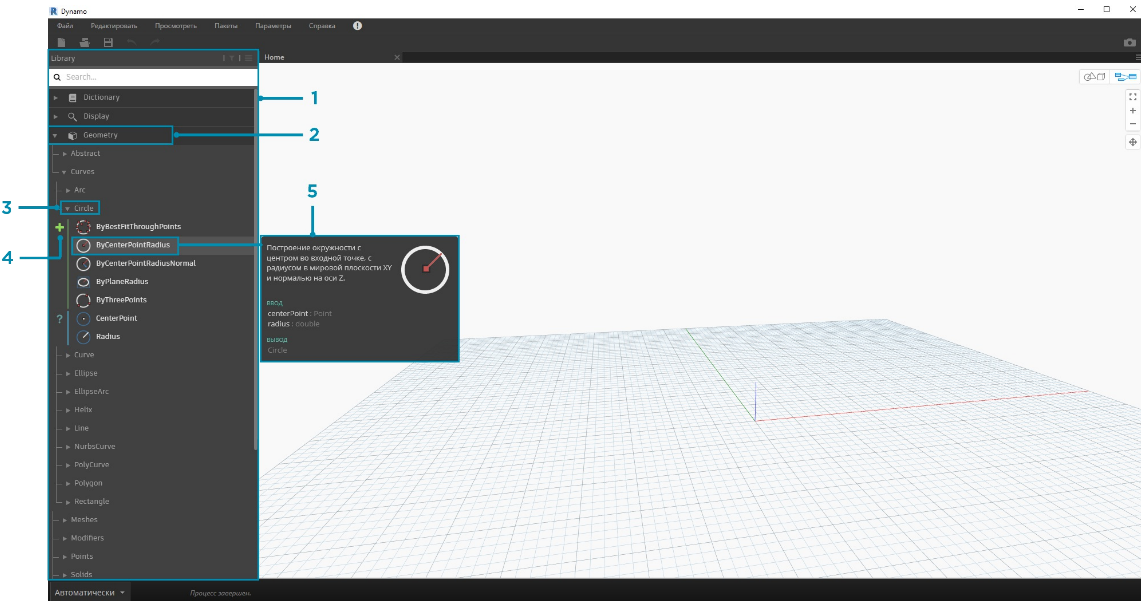
Библиотека **Dynamo** содержит узлы, из которых в рабочем пространстве создаются визуальные программы. Для доступа к узлам в библиотеке можно использовать функции поиска и обзора. Узлы (стандартные узлы из комплекта установки, а также пользовательские узлы и узлы из менеджера пакетов, добавленные в Dynamo) сгруппированы в библиотеке иерархически по категориям. Рассмотрим эту структуру и ознакомимся с основными узлами, которыми мы будем пользоваться чаще всего.

Библиотека библиотек

Библиотека Dynamo, с которой мы работаем в рамках приложения, по сути представляет собой набор из нескольких функциональных библиотек, каждая из которых содержит узлы, сгруппированные по категориям. Подобная гибкая структура, на первый взгляд сложная для восприятия, позволяет эффективно упорядочивать узлы, устанавливаемые по умолчанию с Dynamo, а также пользовательские узлы и дополнительные пакеты, используемые для расширения функциональных возможностей приложения.

Структура

Раздел **Библиотека** пользовательского интерфейса Dynamo состоит из набора иерархически упорядоченных отдельных библиотек. В процессе поиска нужных узлов мы последовательного переходим от общего списка к отдельной библиотеке, затем к нужной категории и подкатегории.

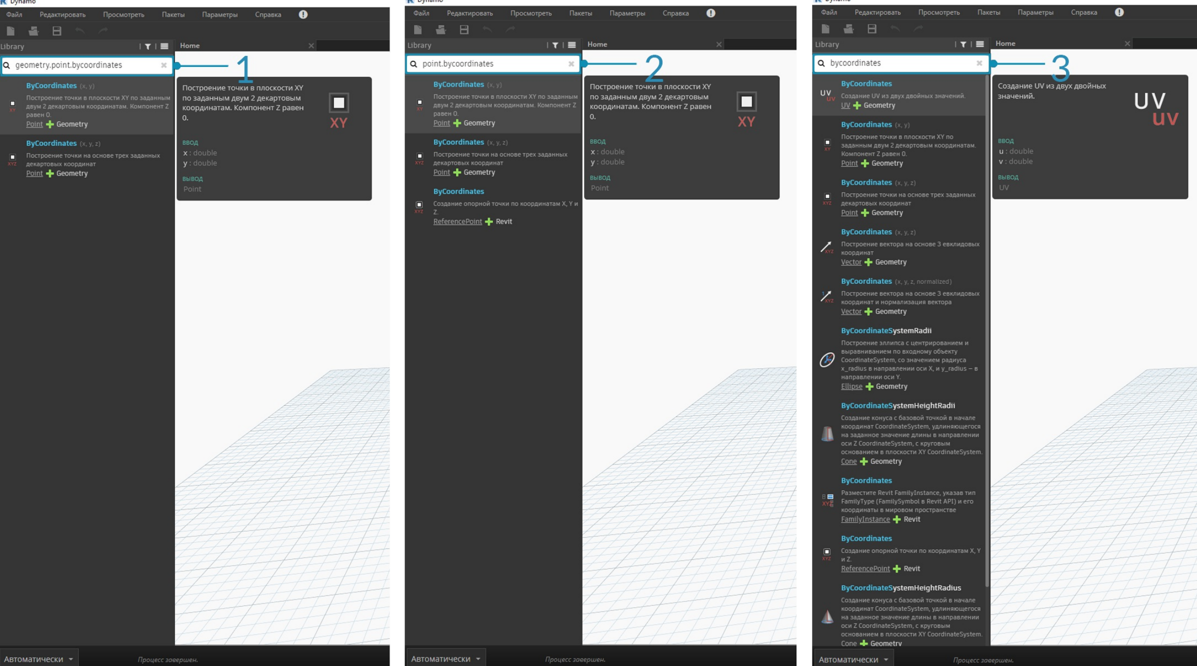


1. Библиотека: область интерфейса Dynamo (библиотека библиотек)
2. Отдельная библиотека: набор связанных категорий (например, **Geometry**)
3. Категория: набор связанных узлов, например всех компонентов, относящихся к **окружностям**
4. Подкатегория: более подробная классификация узлов в рамках категории, обычно по операциям (**Create**, **Action** и **Query**)
5. Узел: объекты, добавляемые в рабочее пространство для выполнения какого-либо действия

Правила именования

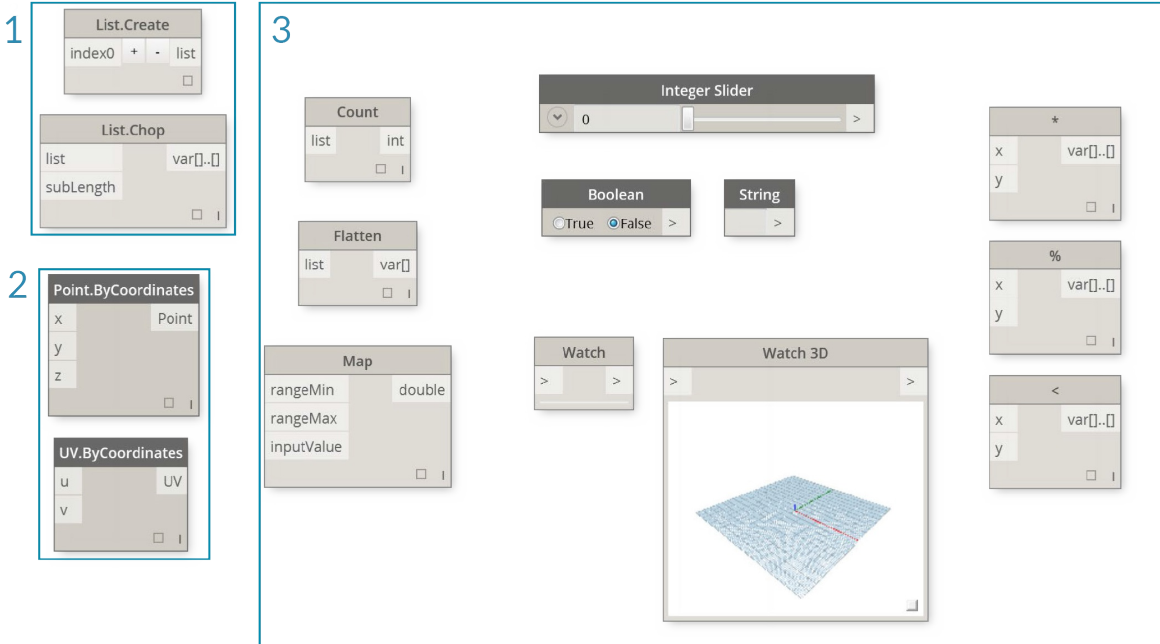
Иерархия каждой библиотеки отражена в именах узлов, добавляемых в рабочее пространство, которые также можно использовать в поле поиска и узлах Code Block, которые поддерживают *текстовый язык Dynamo*. Таким образом, при поиске узлов можно использовать не только ключевые слова, но и иерархические последовательности, разделенные точками.

При вводе различных фрагментов расположения узла в библиотечной иерархии в формате `библиотека.категория.имя_узла` выводятся различные результаты.



1. `Библиотека.Категория.ИмяУзла`
2. `Категория.ИмяУзла`
3. `ИмяУзла` или `ключевое слово`

В рабочем пространстве имя узла обычно представлено в формате `Категория.ИмяУзла` (есть некоторые исключения, в частности, узлы категорий Input и View). Соблюдайте осторожность при использовании узлов с одинаковыми именами и обращайте особое внимание на различия в категориях, к которым они относятся.



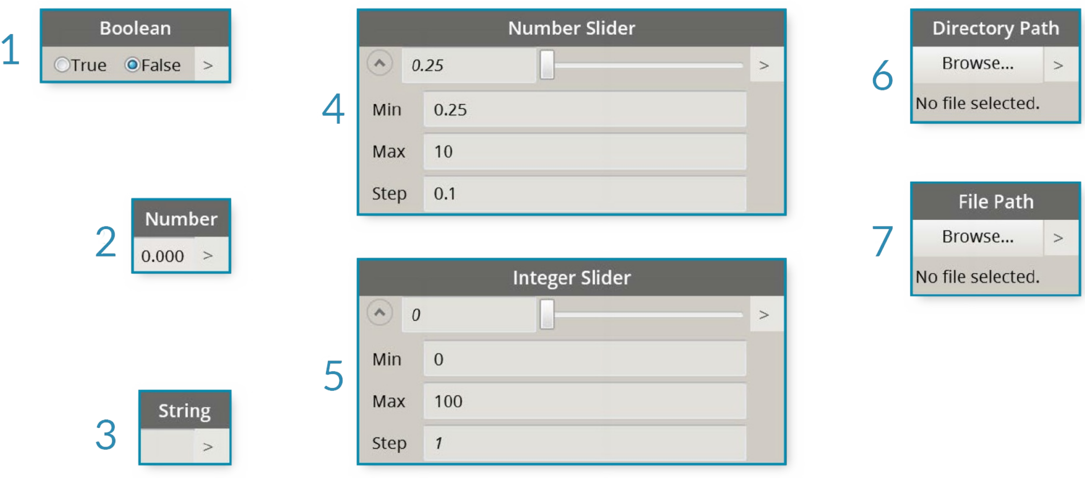
1. Узлы из большинства библиотек содержат наименование категории.
2. Имена узлов `Point.ByCoordinates` и `UV.ByCoordinates` совпадают, но они относятся к разным категориям.
3. К исключениям относятся встроенные функции, Core.Input, Core.View и логические операторы.

Часто используемые узлы

В комплект установки приложения Dynamo входят сотни стандартных узлов. Какие же из них наиболее важны при разработке визуальных программ? Давайте рассмотрим узлы, которые позволяют определять параметры программы (**Input**), отображать результаты действия, выполненного тем или иным узлом (**Watch**), и задавать входные данные и функции ускоренным методом (**Code Block**).

Input

Узлы Input — это ключевой инструмент, благодаря которому пользователи визуальных программ могут работать с основными параметрами. Приведем список узлов, доступных в категории Input основной библиотеки.

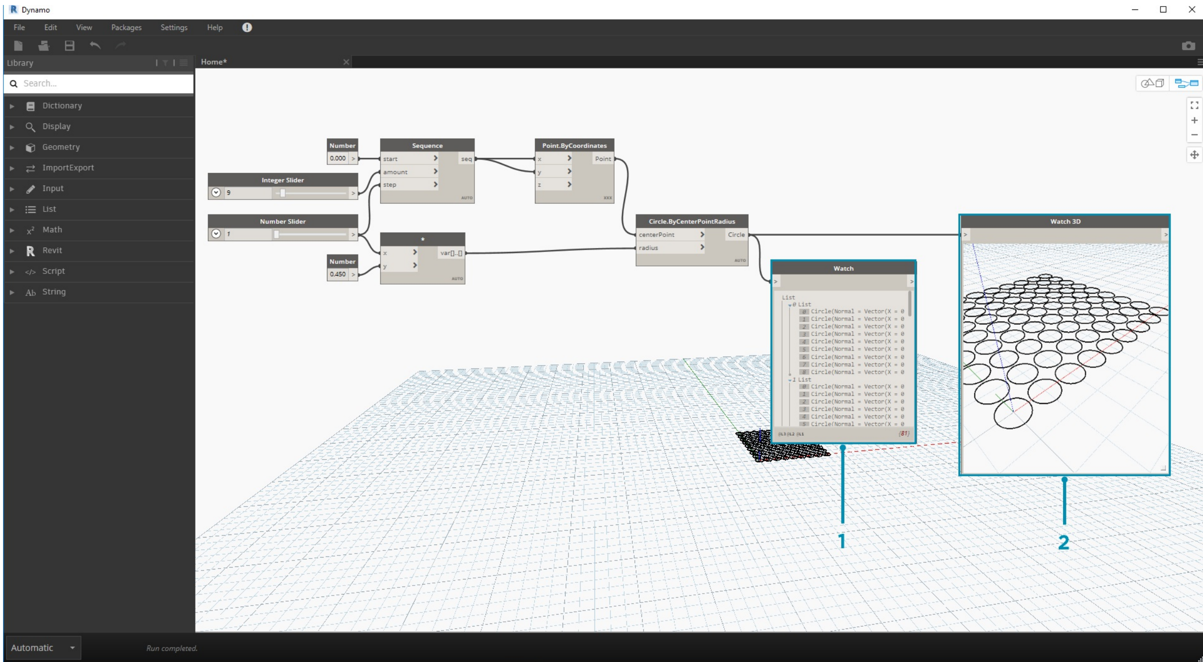


1. Boolean
2. Number
3. String
4. Number Slider
5. Integer Slider
6. Directory Path
7. File Path

Watch

Узлы Watch играют важную роль в управлении потоком данных в визуальной программе. Результаты действия, выполняемого узлом, можно просмотреть в области предварительного просмотра данных узла, однако если требуется, чтобы эти результаты отображались постоянно, следует использовать узел **Watch** или **Watch3D**. Оба этих узла находятся в категории View основной библиотеки.

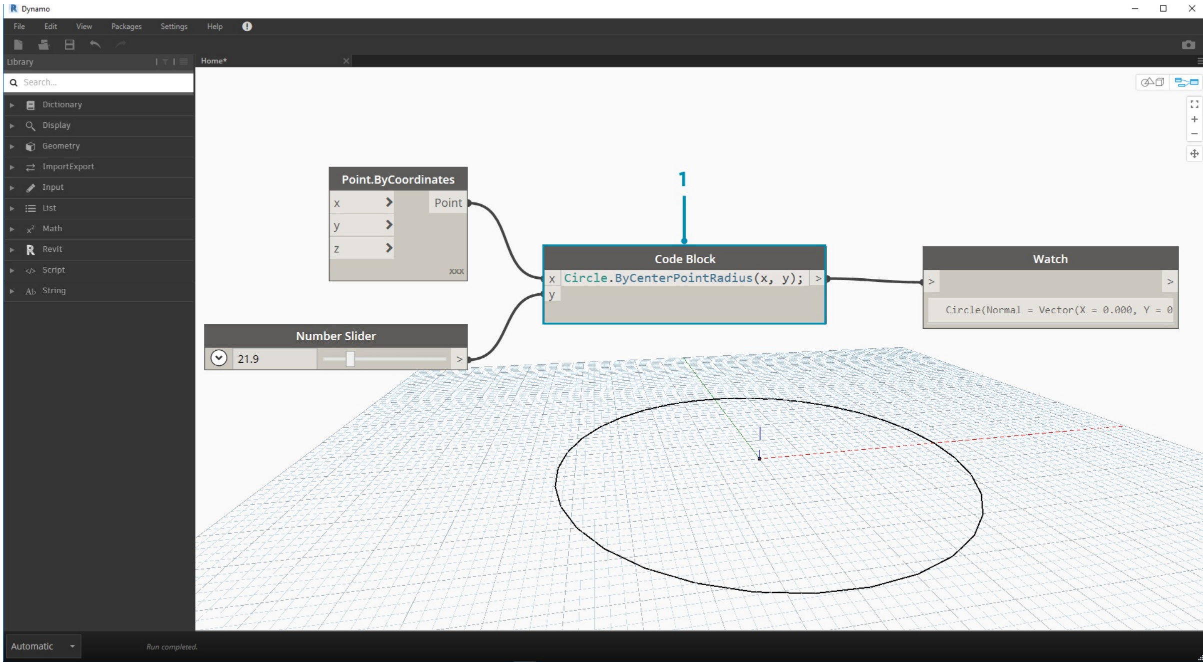
> Совет. Иногда при работе с большим количеством узлов пользоваться функцией 3D-просмотра может быть неудобно. В этом случае можно снять флажок фонового просмотра в меню параметров и использовать узел Watch3D для предварительного просмотра геометрии.



1. Watch: обратите внимание, что при выборе элемента в узле Watch он помечается в узле Watch3D и области 3D-просмотра.
2. Watch3D: с помощью правой нижней ручки можно изменить размер окна, а перемещаться по нему можно с помощью мыши, как при 3D-просмотре.

Code Block

Узлы **Code Block** позволяют создавать блоки кода, состоящие из строк, разделенных запятыми. Данные, используемые в этих узлах, могут быть самыми простыми, например координаты `X/Y`. Кроме этого, узлы Code Block можно использовать в качестве упрощенного метода ввода чисел или вызова функции другого узла. Используемый синтаксис должен следовать правилам именования текстового языка Dynamo DesignScript (см. раздел 7.2). Давайте попробуем создать окружность с помощью этого упрощенного метода.



1. Дважды щелкните для создания узла **Code Block**.
2. Введите `Circle.ByCenterPointRadius(x,y);`.
3. Если щелкнуть в рабочем пространстве, чтобы отменить выбор, автоматически добавляются входные порты `x` и `y`.
4. Создайте узлы **Point.ByCoordinates** и **Number Slider**, затем соедините их с входными портами узла Code Block.

5. В области 3D-просмотра отобразится окружность, являющаяся результатом выполнения визуальной программы.

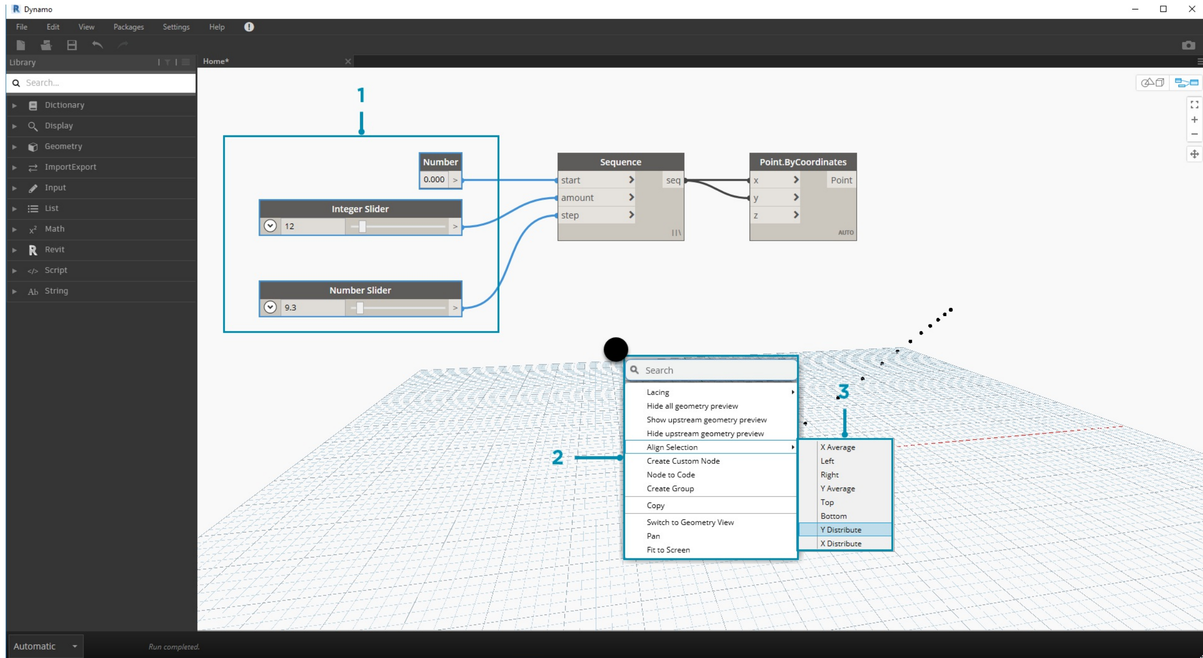
Управление структурой программы

Управление структурой программы

Процесс визуального программирования может быть очень увлекательным и вдохновляющим, однако работа с потоком выполнения операций и ключевыми входными данными пользователя может очень быстро зайти в тупик из-за запутанности программы или неудачной компоновки рабочего пространства. Далее представлены некоторые практические советы по управлению структурой программы.

Выравнивание

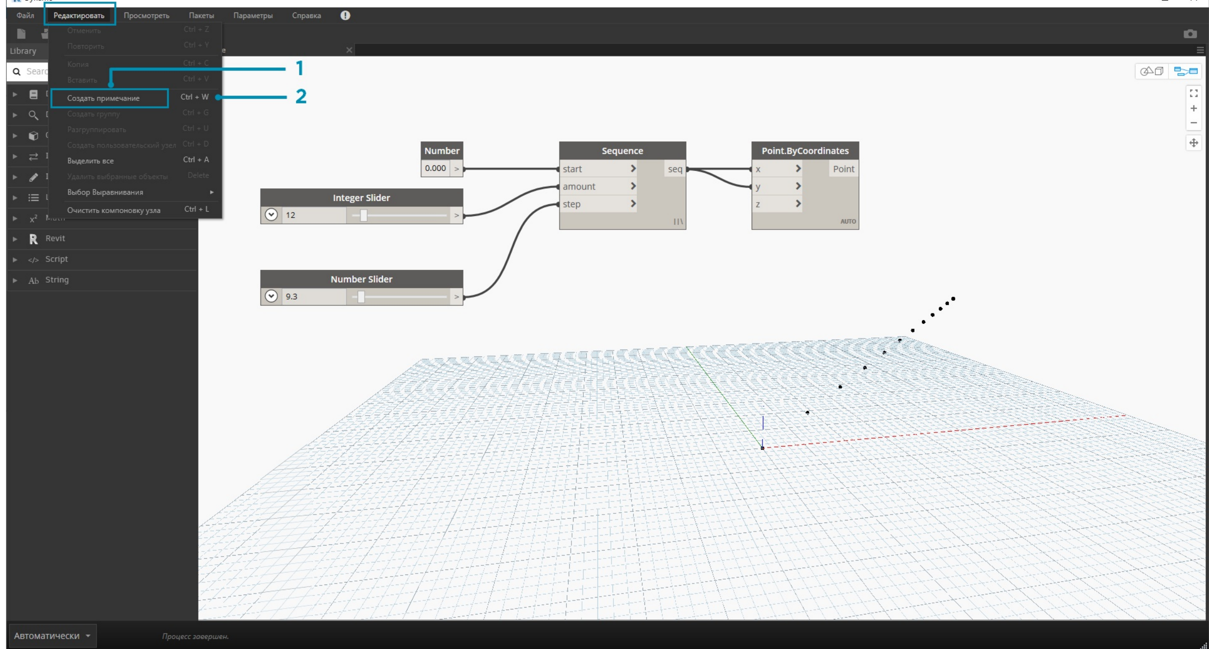
Когда рабочее пространство постепенно начнет заполняться узлами, может потребоваться переупорядочить их для большей наглядности. Выберите несколько узлов и щелкните в рабочем пространстве правой кнопкой мыши. Появится всплывающее окно с меню **Выбор выравнивания**, содержащим параметры выравнивания и распределения по осям X и Y.



1. Выберите несколько узлов.
2. Щелкните в рабочем пространстве правой кнопкой мыши.
3. Воспользуйтесь параметрами меню **Выбор выравнивания**.

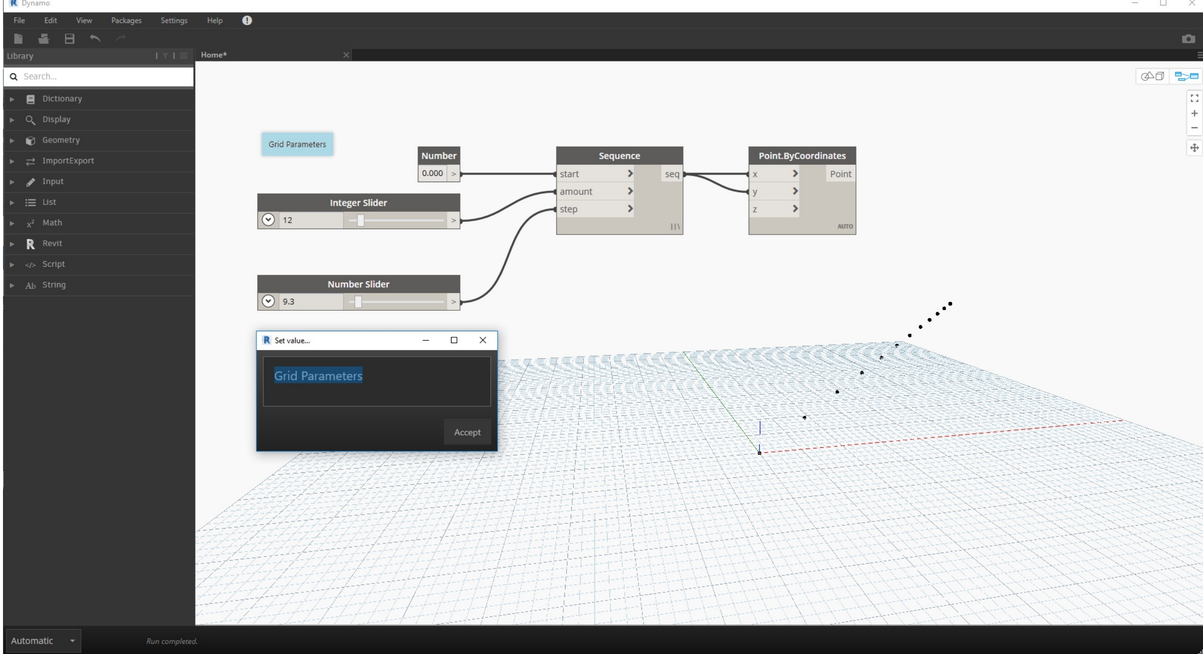
Примечания

По мере накопления опыта вы научитесь «считывать» содержимое визуальной программы, просматривая имена узлов и следуя последовательности выполнения операций. Чтобы дополнительно упростить работу как опытным пользователям, так и новичкам, мы рекомендуем добавлять простые текстовые метки и описания. Для этого в Dynamo можно использовать узел **Notes** с редактируемым текстовым полем. Добавить примечания в рабочее пространство можно двумя способами.



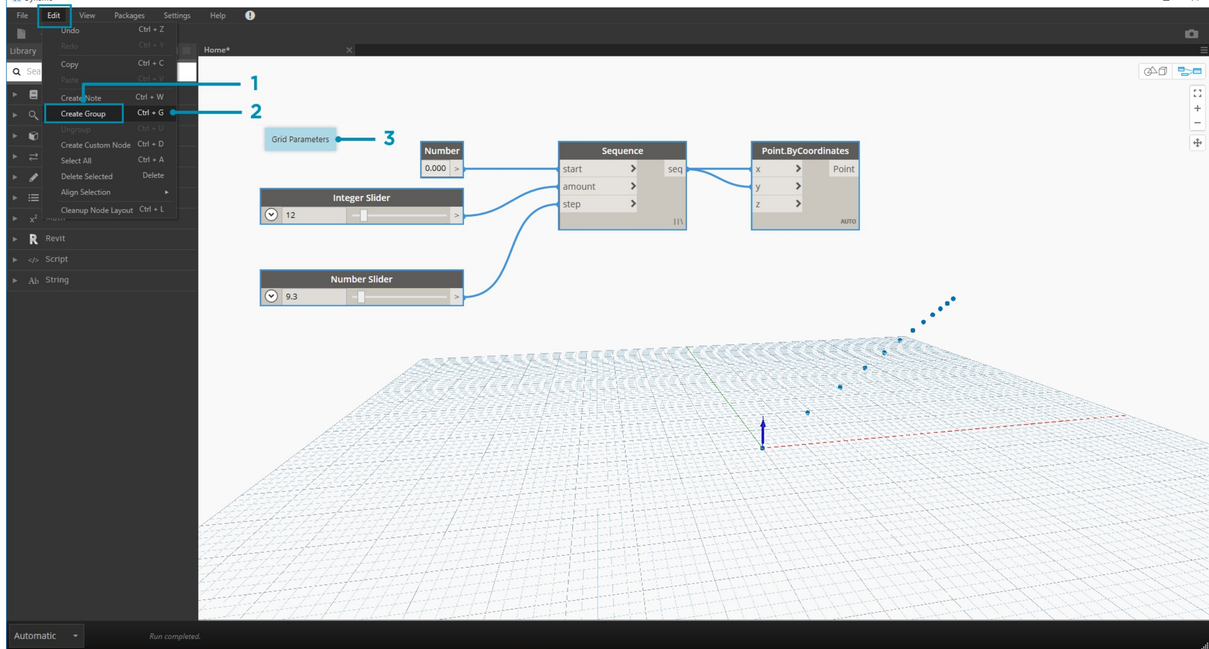
1. Перейдите в меню «Редактировать» > «Создать примечание».
2. Используйте клавиши быстрого вызова CTRL + W.

После добавления примечания в рабочее пространство появится текстовое поле, позволяющее отредактировать текст примечания. Созданные примечания можно изменить, щелкнув узел примечаний правой кнопкой мыши или щелкнув его дважды.



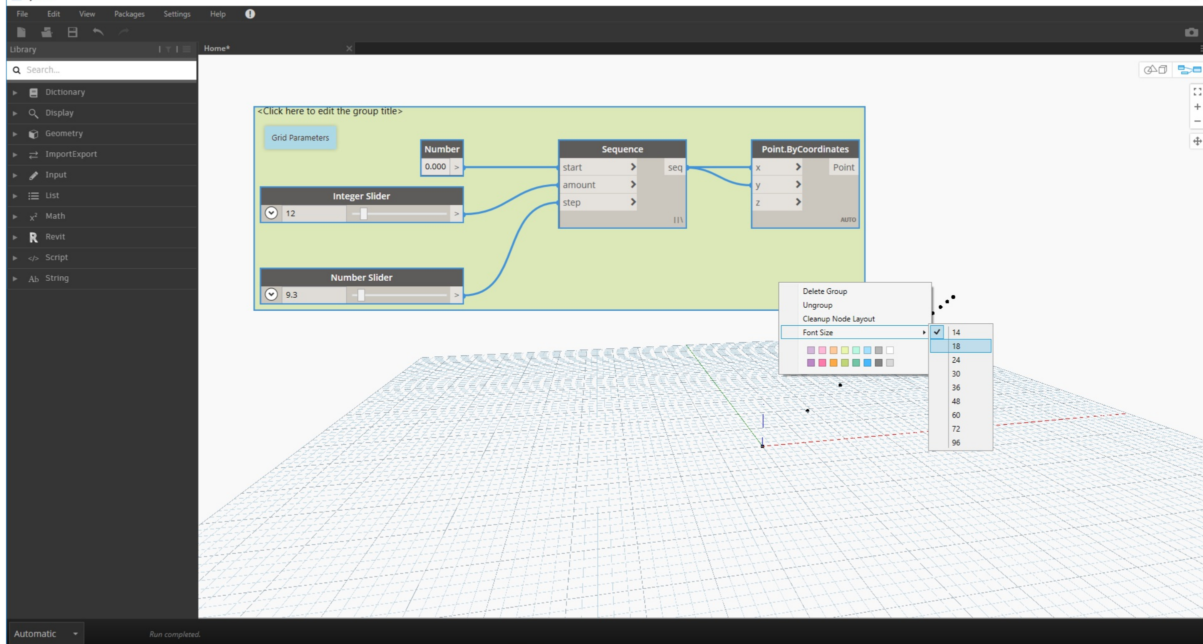
Группирование

Когда число компонентов визуальной программы становится по-настоящему большим, для упрощения работы с ними рекомендуется выделить крупные этапы процесса выполнения. Большие наборы узлов можно объединять в **группы**, в результате чего они помечаются цветным фоновым прямоугольником и заголовком. Создать группу на основе нескольких выбранных узлов можно тремя способами.



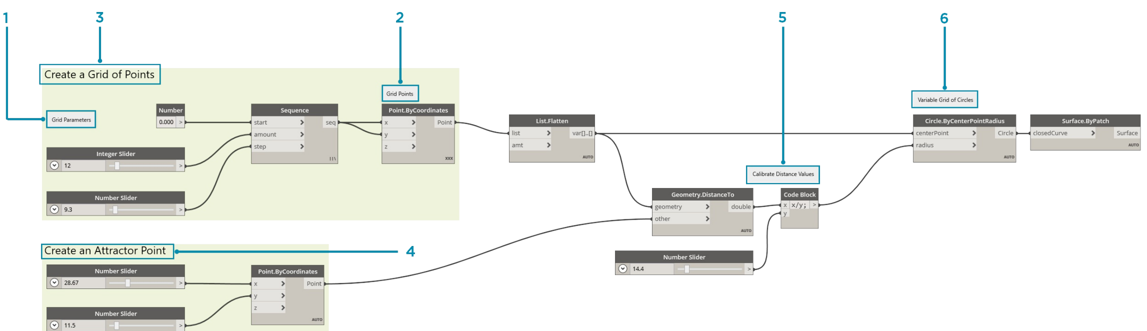
1. Перейдите в меню «Редактировать» > «Создать группу».
2. Используйте клавиши быстрого вызова CTRL + G.
3. Щелкните в рабочем пространстве правой кнопкой мыши и выберите «Создать группу».

После создания группы можно отредактировать ее параметры, например название и цвет фона.



Совет. Использование примечаний и групп является эффективным способом аннотирования файла и повышения его читабельности.

Далее приводится программа из раздела 2.4, к которой были добавлены примечания и группы.



1. Примечание: «Параметры сетки»
2. Примечание: «Точки сетки»
3. Группа: «Создать сетку из точек»
4. Группа: «Создать точку аттрактора»
5. Примечание: «Откалибровать значения расстояния»
6. Примечание: «Переменная сетка окружностей»

Компоновочные блоки программ

КОМПОНОВОЧНЫЕ БЛОКИ ПРОГРАММ

Чтобы углубиться в процесс разработки визуальных программ, нам потребуется более тонкое понимание того, что представляют собой компоновочные блоки программ. В этой главе описываются основные понятия, относящиеся к данным, проходящим по проводам программ Dynamo.

ToString
NurbsCurve.ByPoints
Excel.ReadFromFile
Color.ByARGB
PointAnalysisData.ByPoints
Element.Name
List.Count
NewtonRootFind1DWithDeriv
Map
Math.Cosh
Topology.Faces
Cuboid.ByCorners
Color.ByARGB
TEST Recursion
Number Slider
LunchBox Domain Variables
Number Slider
Number Sequence
XYZ Array on Curve
&&
If

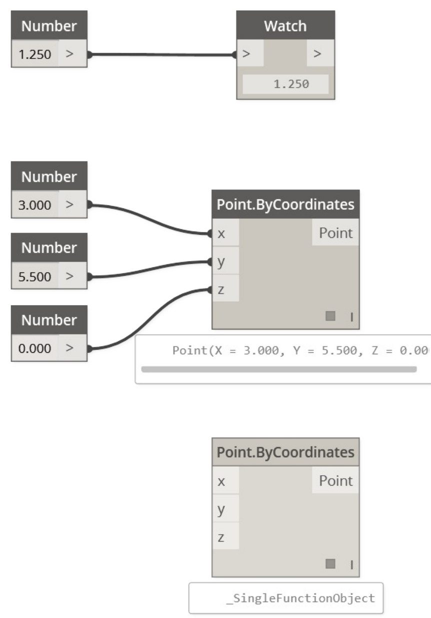
Данные

Данные

Данные — это содержимое программы. Они передаются по проводам, предоставляя входные значения узлам, в которых затем обрабатываются и преобразуются в выходные данные новой формы. Давайте рассмотрим определение данных и их структуру, а затем начнем работу с данными в Dynamo.

Что такое данные?

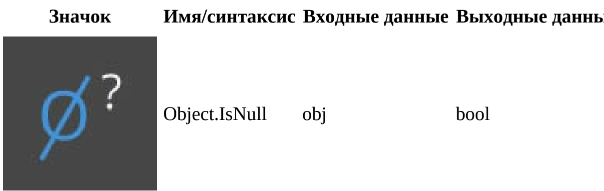
Данные — это набор значений количественных и качественных переменных. Самая простая форма данных — числа, например `0`, `3.14` или `17`. Однако существуют и другие типы данных: переменные, представляющие меняющиеся числа (`высота`); символы (`myName`); геометрические объекты (`окружность`); список элементов данных (`1,2,3,5,8,13,..`). Данные необходимы для добавления в порты входных параметров в узлах Dynamo. Хотя данные могут существовать без действий, они необходимы для обработки действий, которые представлены в форме узлов. Если узел добавлен в рабочее пространство, но не имеет входных данных, результатом будет функция, а не результат самого действия.



1. Простые данные
2. Данные и действие (узел), которое успешно выполняется.
3. Действие (узел) без входных данных возвращает типовую функцию.

Будьте осторожны с нулевыми объектами

Тип `null` означает отсутствие данных. Это абстрактное понятие, с которым можно, тем не менее, столкнуться при визуальном программировании. Если результат действия недопустим, узел возвращает нулевой объект. Проверка наличия нулевых объектов и их удаление из структуры данных крайне важны для создания надежных программ.

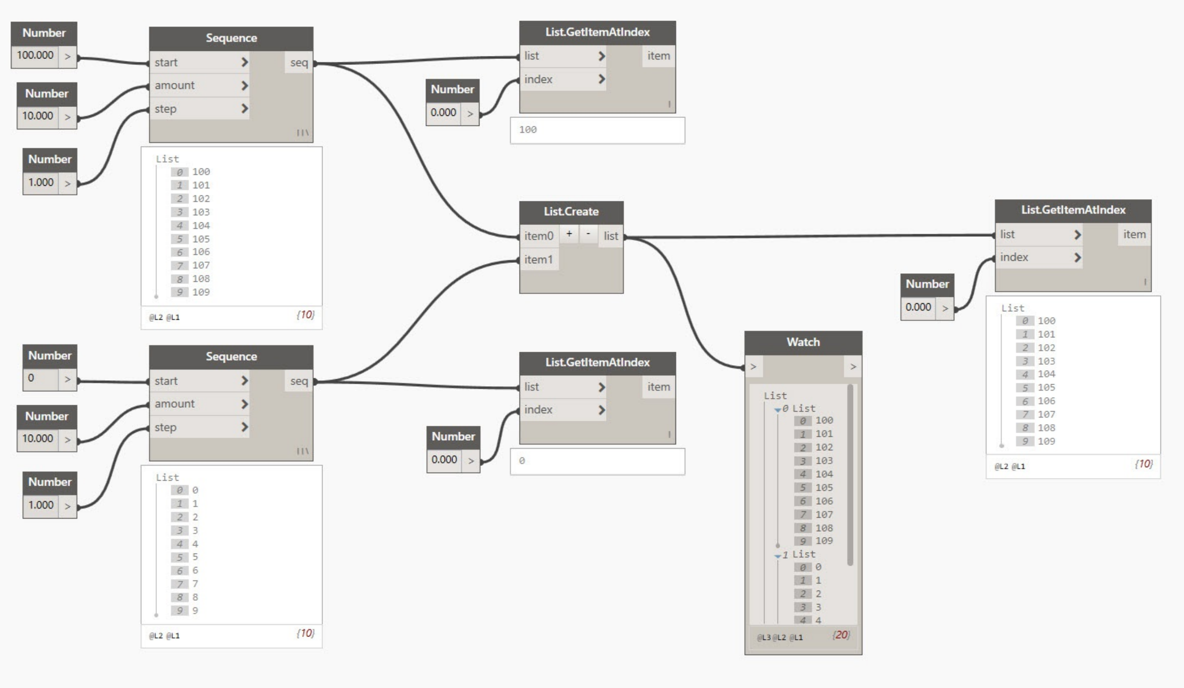
Значок	Имя/синтаксис	Входные данные	Выходные данные
	Object.IsNull	obj	bool

Структуры данных

При визуальном программировании можно очень быстро генерировать большие объемы данных, поэтому необходимы средства для управления их иерархией. Эту роль выполняют структуры данных — организационные схемы, в которых хранятся данные. Особенности структур данных и их использования зависят от языка программирования. В Dynamo для построения иерархии данных используются списки. Они будут подробнее рассмотрены в следующих главах, пока же приведем только общие сведения.

Список — это набор элементов, размещенных в одной структуре данных:

- У меня пять пальцев (*элементы*) на руке (*список*).
- На моей улице (*список*) десять домов (*элементы*).



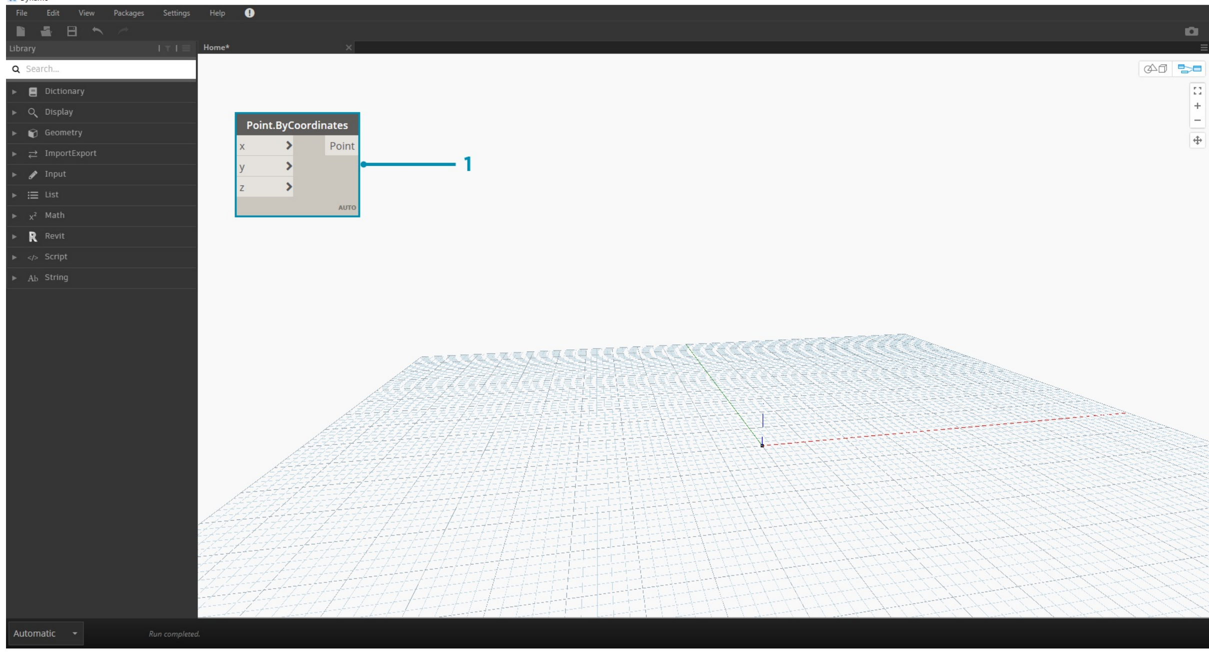
1. Узел **Number Sequence** определяет список чисел на основе входных данных *start*, *amount* и *step*. С помощью этих узлов было создано два отдельных списка из десяти чисел, один из которых охватывает диапазон *100–109*, а другой — *0–9*.
2. Узел **List.GetItemAtIndex** позволяет выбрать элемент в списке по определенному индексу. При выборе значения *0* будет получен первый элемент в списке (в данном случае — *100*).
3. Та же процедура применительно ко второму списку дает значение *0* — первый элемент в списке.
4. Объединим оба списка в один с помощью узла **List.Create**. Обратите внимание, что узел создает *список списков.* Это меняет структуру данных.
5. При повторном использовании узла **List.GetItemAtIndex** с индексом *0* получаем первый список в списке списков. Это означает, что список рассматривается как элемент, в чем и состоит отличие от других языков программирования. В последующих главах операции со списками и структуры данных будут рассмотрены подробнее.

Главное, что следует помнить об иерархии данных в Dynamo — **в случае со структурой данных списки рассматриваются как элементы.** Другими словами, в Dynamo структура данных рассматривается сверху вниз. Что это означает? Рассмотрим пример.

Использование данных для создания цепочки цилиндров

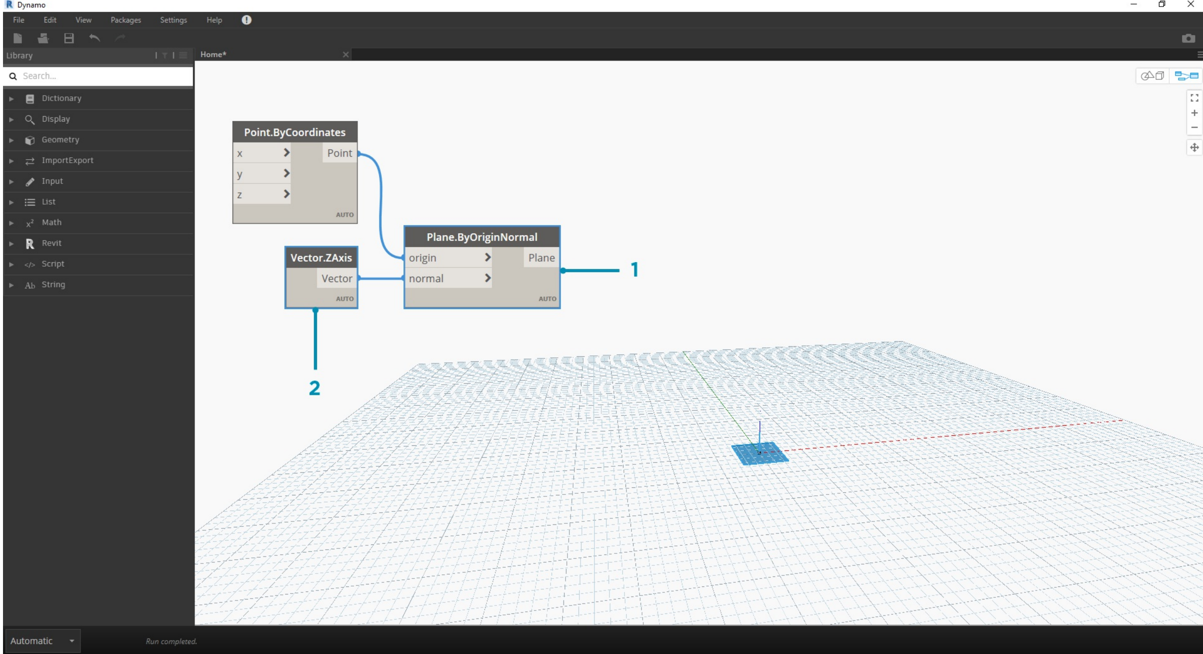
Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): Building Blocks of Programs - Data.dyn. Полный список файлов примеров можно найти в приложении.

В первом примере создадим цилиндр с оболочкой, пройдя по ступеням геометрической иерархии, описанной в этом разделе.

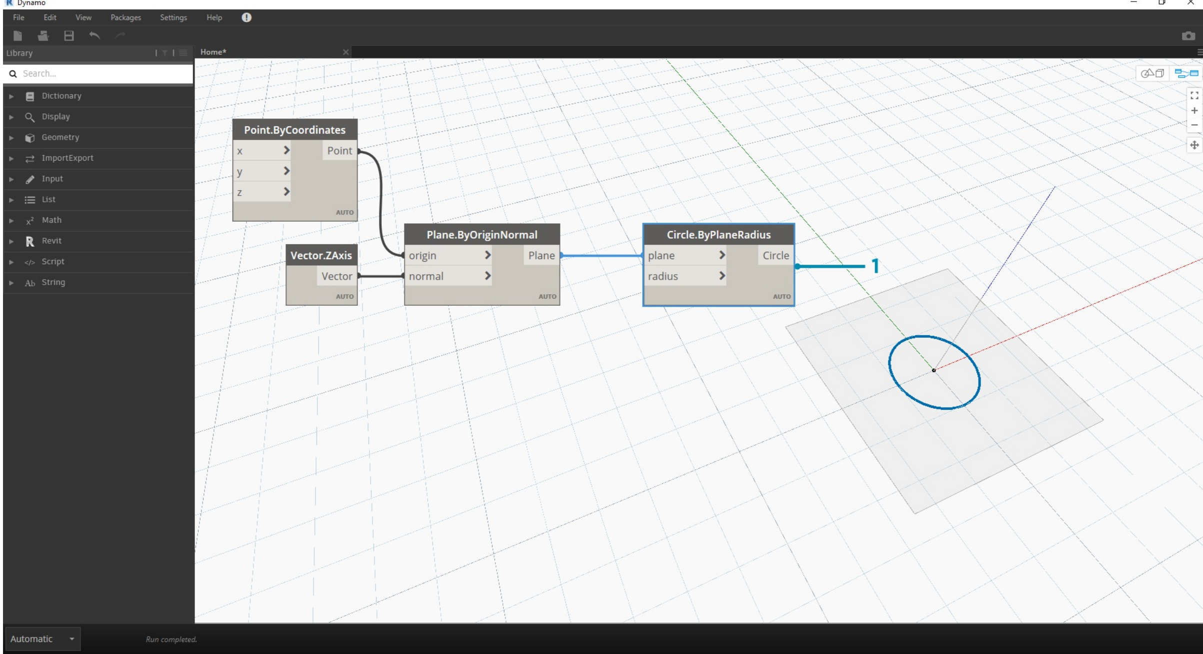


1. **Point.ByCoordinates.** После того как в рабочую область добавлен узел, в начале координат сетки предварительного просмотра

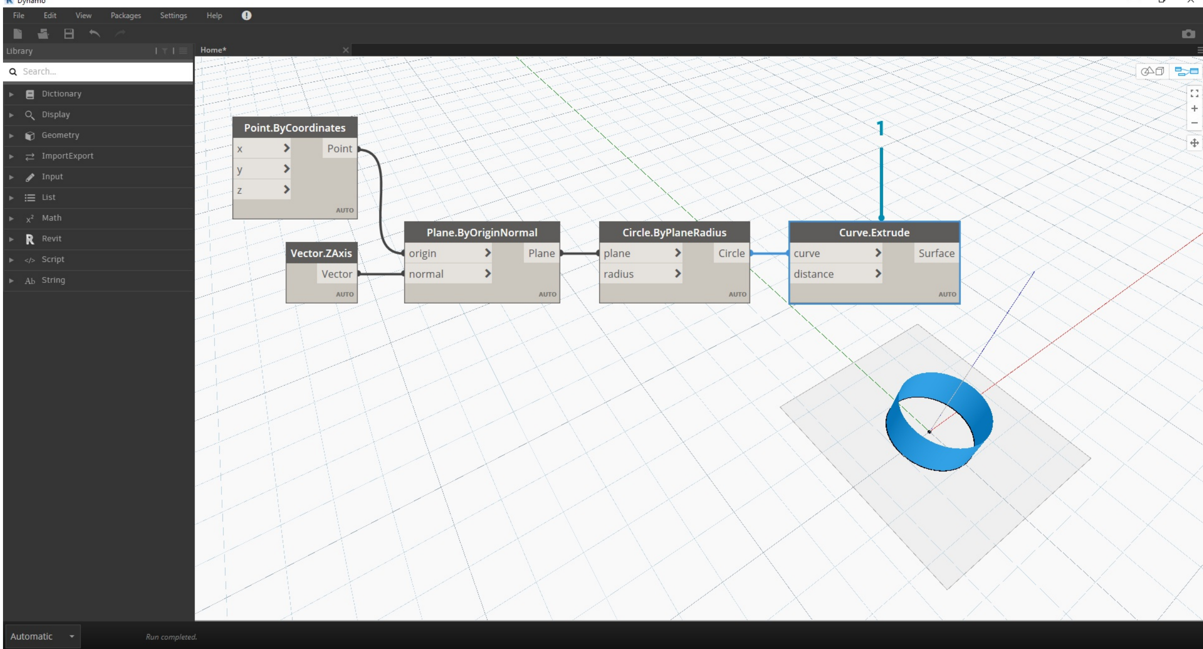
Dynamo появляется точка. Значения по умолчанию для выходных параметров *x,y* и *z* равны *0,0*. В этом месте и была создана точка.



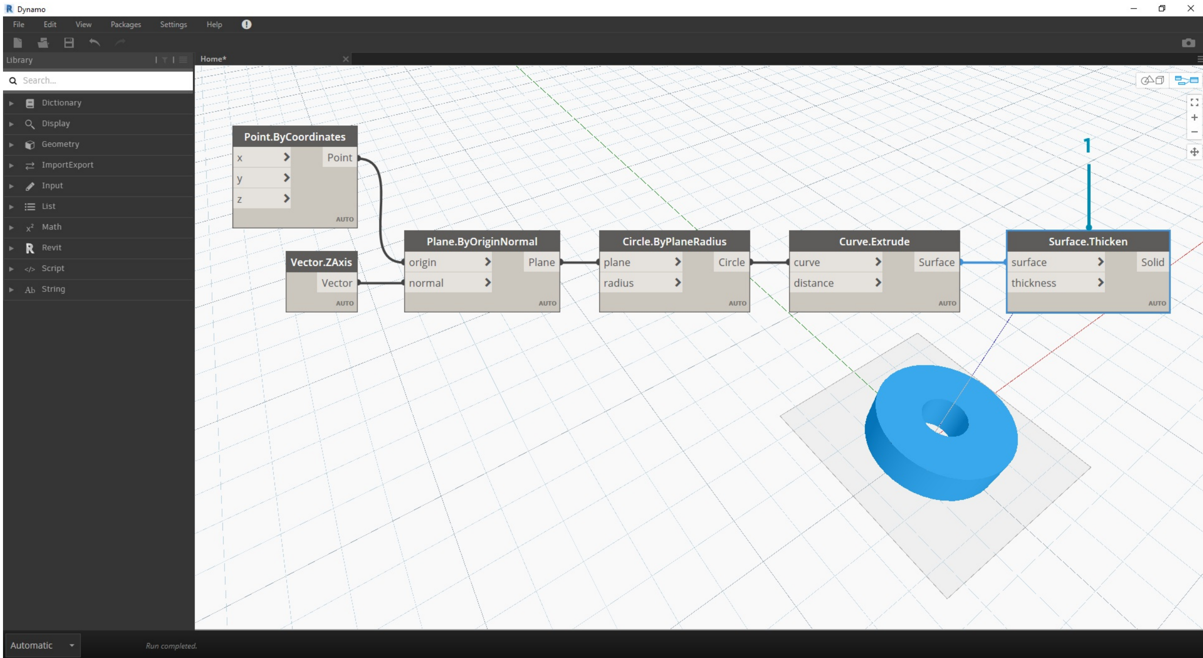
1. **Plane.ByOriginNormal.** Следующий шаг в построении геометрической иерархии — плоскость. Существует несколько способов построения плоскости. В этом случае в качестве входных данных используется начало координат и нормаль. Начало координат — это узел-точка, созданный в предыдущем шаге.
2. **Vector.ZAxis.** Построение унифицированного вектора в направлении Z. Обратите внимание, что здесь входные данные отсутствуют, а есть только вектор со значением [0,0,1]. Он будет использоваться в качестве входных данных *нормали* для узла *Plane.ByOriginNormal*. В результате получается прямоугольная плоскость в области предварительного просмотра Dynamo.



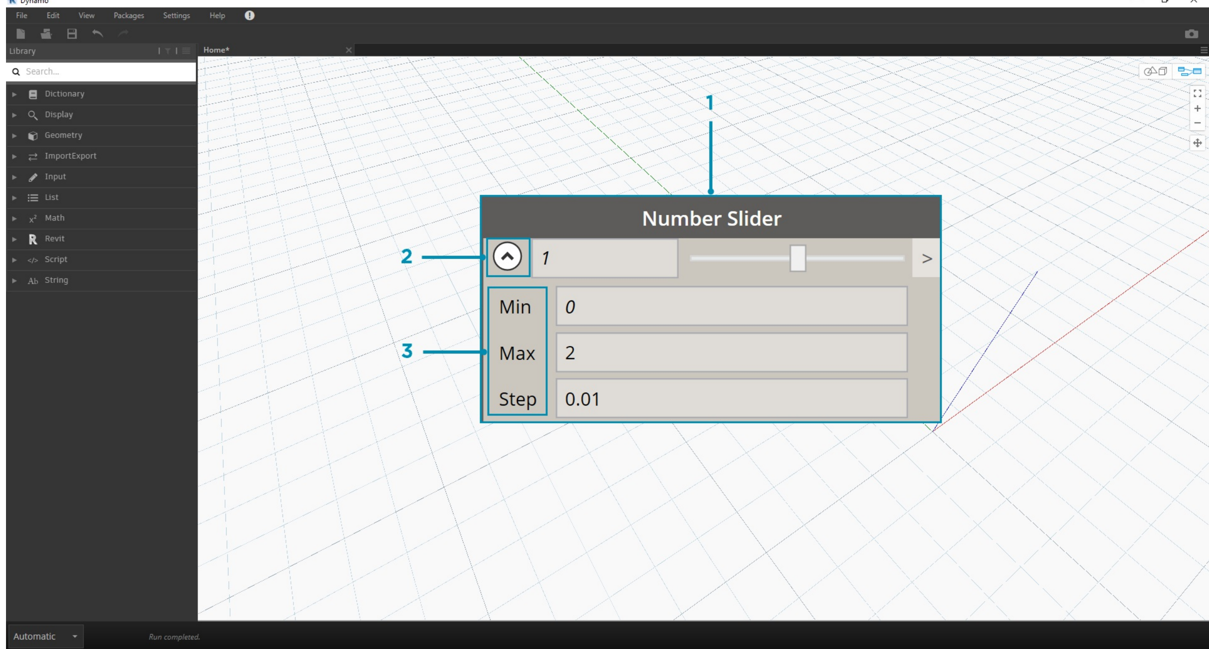
1. **Circle.ByPlaneRadius.** Продвигаясь вверх по иерархии, создадим кривую из плоскости, полученной в предыдущем шаге. После соединения с узлом получаем окружность в начале координат. По умолчанию радиус в узле имеет значение *1*.



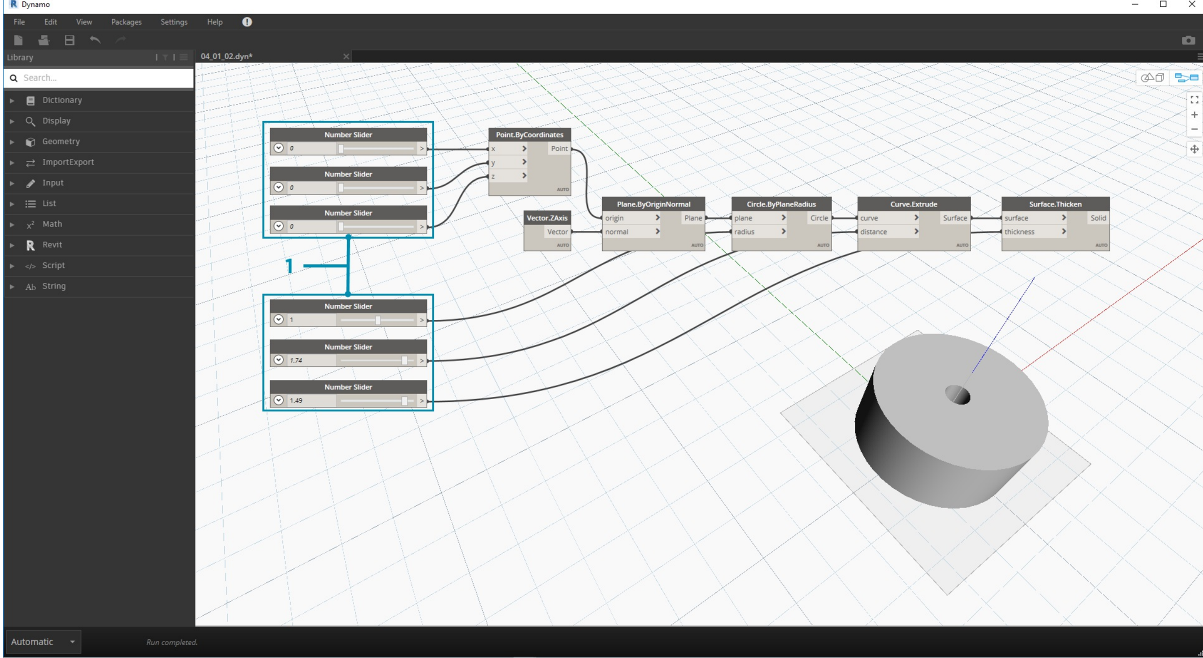
1. **Curve.Extrude.** Теперь выполним выдавливание фигуры, задав глубину и двигаясь в третьем измерении. Этот узел создает поверхность из кривой путем выдавливания. По умолчанию расстояние в узле равно *1*, а на видовом экране должен отображаться цилиндр.



1. **Surface.Thicken.** Этот узел создает замкнутое тело путем смещения поверхности на заданное расстояние и замыкания формы. По умолчанию значение толщины равно *1*, а на видовом экране в соответствии с этими значениями отображается цилиндр с оболочкой.

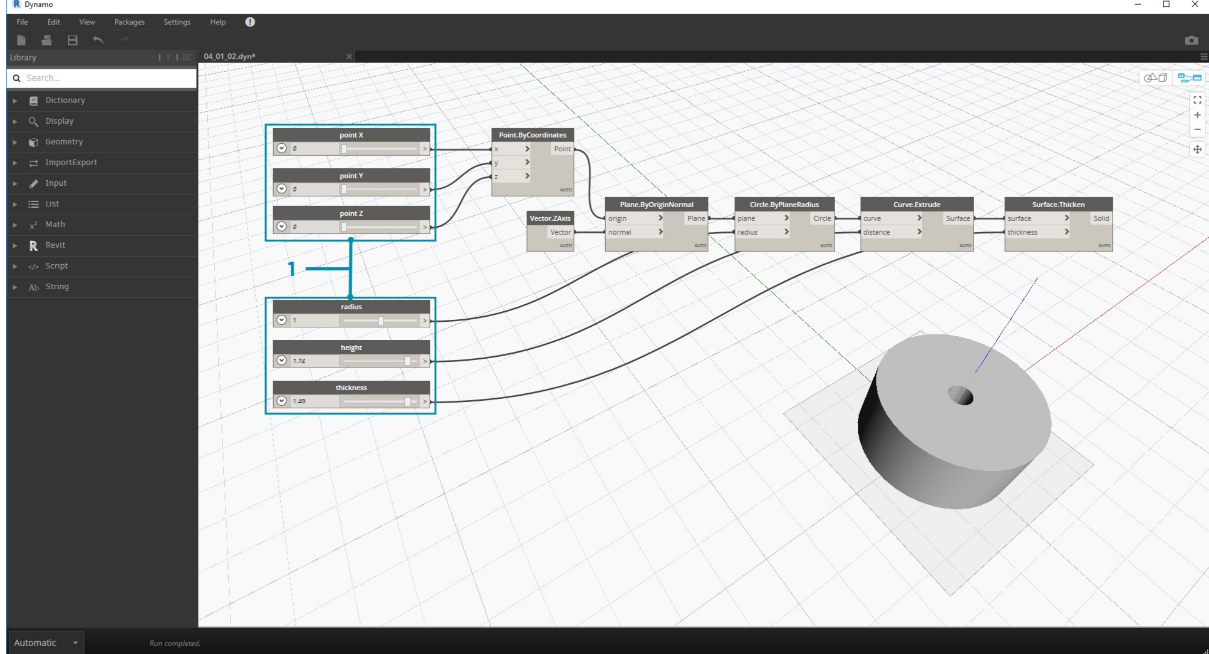


1. **Number Slider.** Вместо использования в качестве значений по умолчанию для входных данных добавим в модель параметрические элементы управления.
2. **Редактирование области.** После добавления регулятора чисел в рабочую область щелкните значок в левом верхнем углу окна, чтобы отобразить параметры области.
3. **Min/Max/Step.** Задайте для *min*, *max* и *step* значения *0*,*2* и *0.01* соответственно. Это необходимо для управления размером всего геометрического объекта.



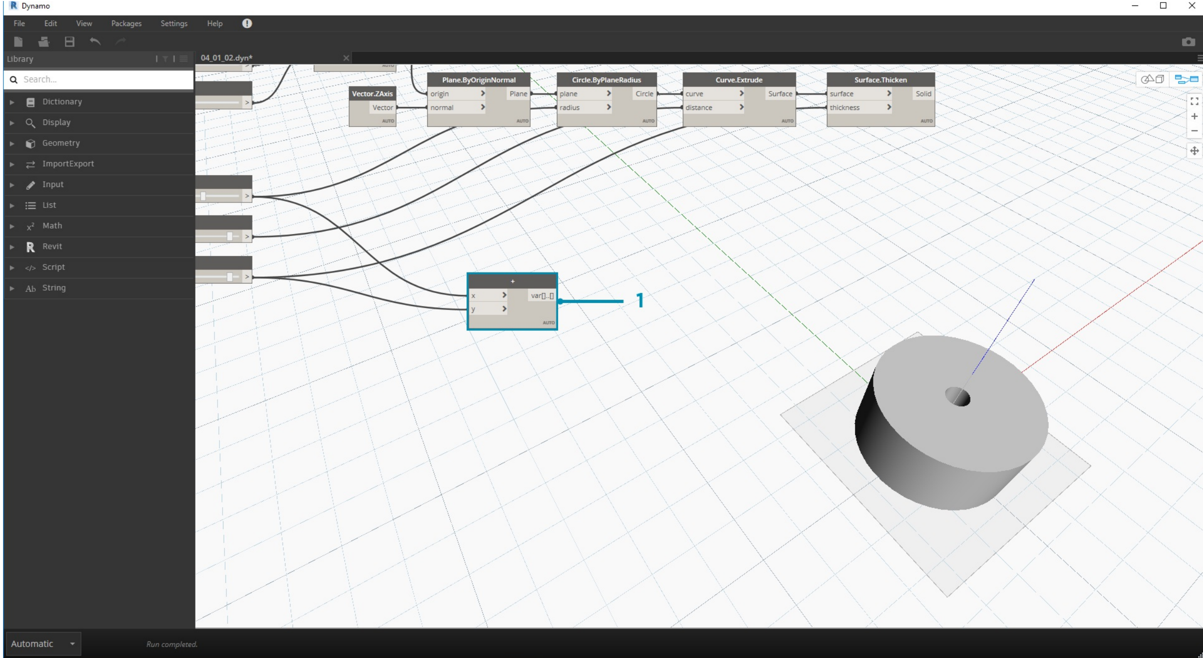
1. **Регуляторы чисел.** Вместо входных значений по умолчанию скопируйте и вставьте этот регулятор чисел (выберите его, нажмите CTRL + C, затем CTRL + V), пока во всех входных параметрах со значениями по умолчанию не будут заданы регуляторы. Чтобы алгоритм действовал, некоторые значения регуляторов должны быть больше нуля (например, для увеличения толщины поверхности требуется глубина выдавливания).

В итоге с помощью регуляторов создан параметрический цилиндр с оболочкой. Попробуйте изменить некоторые из параметров, наблюдая за динамическим обновлением геометрических объектов на видовом экране Dynamo.

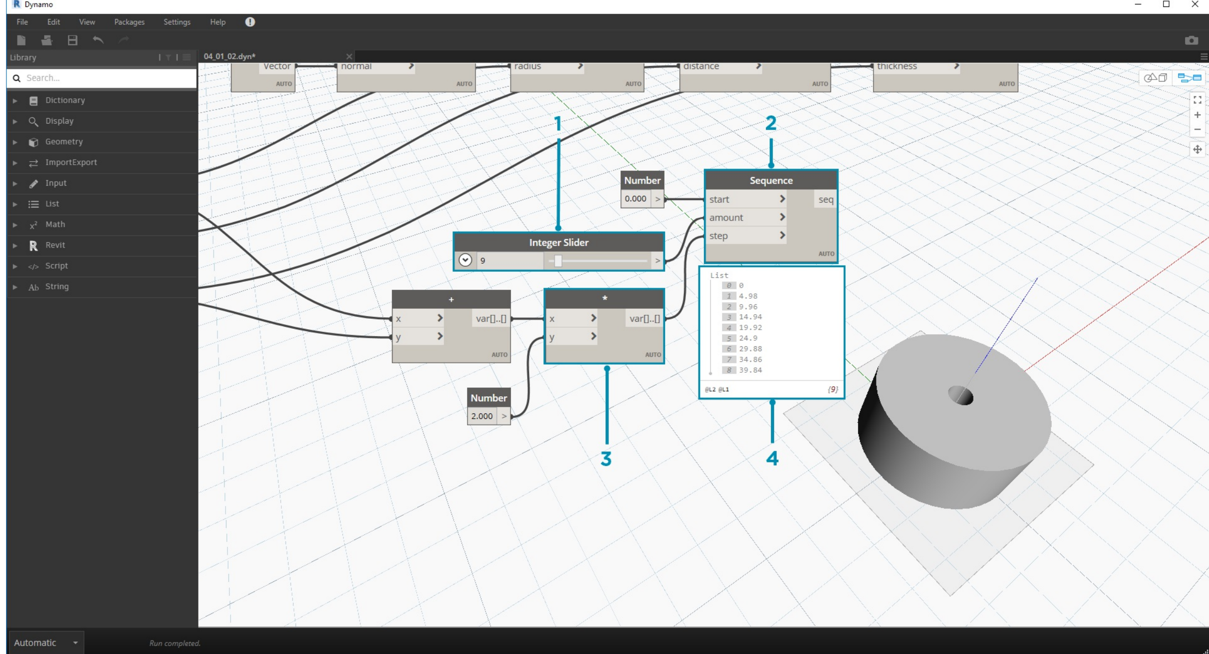


1. **Регуляторы чисел.** На следующем этапе мы добавили в рабочую область множество регуляторов, и теперь необходимо очистить интерфейс только что созданного инструмента. Щелкните один регулятор правой кнопкой мыши и выберите «Переименовать...». Замените имя каждого регулятора именем соответствующего параметра. В качестве источника имен используйте изображение, представленное выше.

На данный момент создан цилиндр с толстыми стенками. Пока это только один объект. Теперь рассмотрим, как создать массив цилиндров, которые динамически связаны друг с другом. Для этого вместо одного объекта создадим список цилиндров.

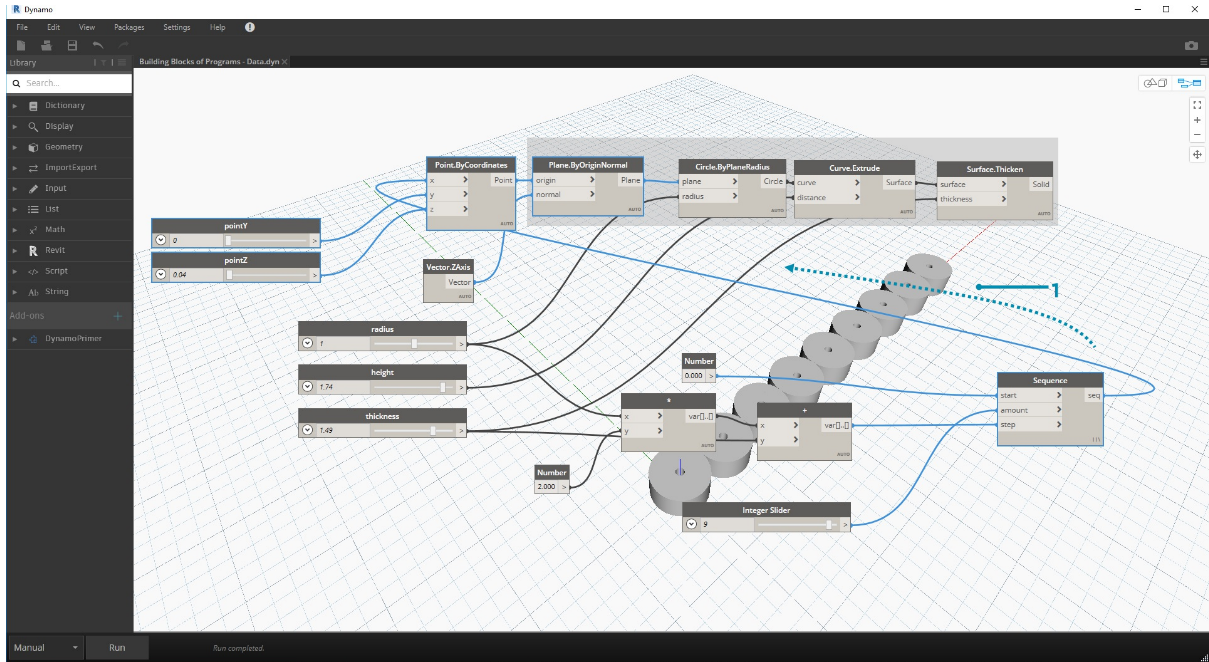


1. **Добавление (+).** Наша цель — добавить ряд цилиндров возле уже имеющегося цилиндра. При вставке еще одного цилиндра рядом с текущим необходимо учитывать радиус цилиндра и толщину его оболочки. Это число можно получить, сложив два значения регуляторов.

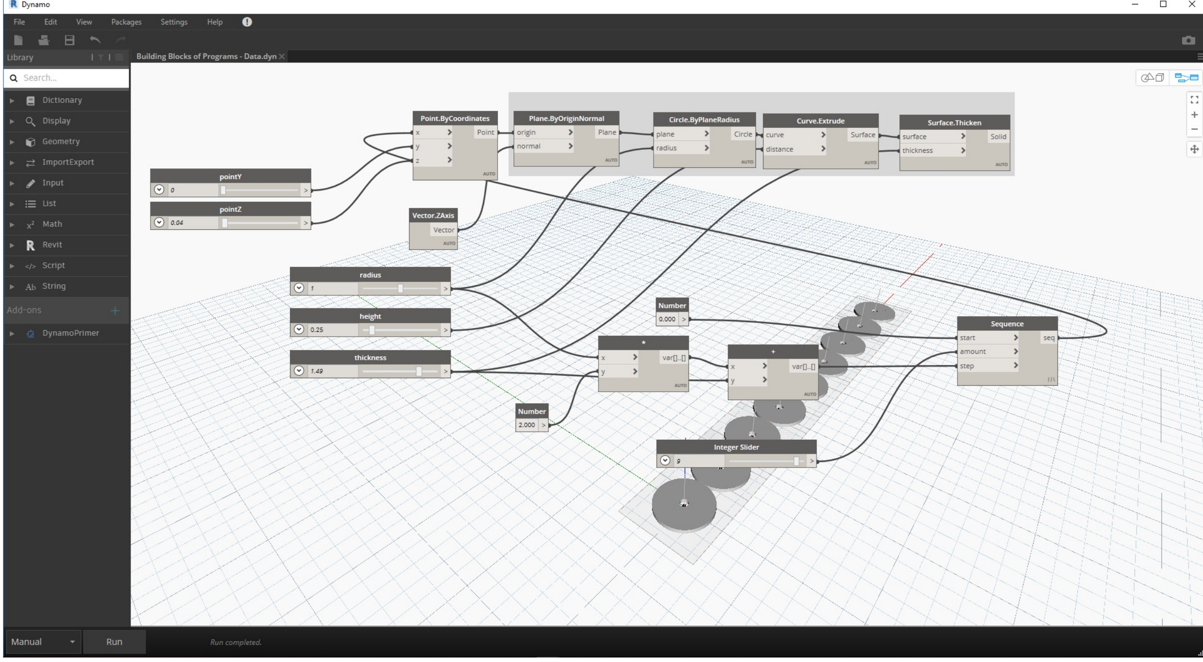


Это более сложный шаг, поэтому рассмотрим его подробнее. Конечная цель — создать список чисел, определяющих местоположение каждого цилиндра в последовательности.

1. **Умножение.** Сначала умножим значение из предыдущего шага на 2. Это было значение радиуса, а цилиндр необходимо переместить на полный диаметр.
2. **Number Sequence.** С помощью этого узла создадим массив чисел. Сначала вставим узел *умножения* из предыдущего шага в качестве значения *step*. В качестве значения *start* можно указать *0.0*, используя узел *number*.
3. **Integer Slider.** Чтобы задать значение *amount*, присоединим регулятор целых чисел. Он будет определять количество создаваемых цилиндров.
4. **Выходные данные.** В этом списке показано расстояние смещения каждого цилиндра в массиве, которое управляется параметрически с помощью первоначальных регуляторов.



1. Этот шаг достаточно прост: соедините последовательность из предыдущего шага с входным параметром *x* исходного узла *Point.ByCoordinates*. При этом регулятор *pointX* будет заменен и его можно удалить. Теперь на видовом экране отображается массив цилиндров (убедитесь, что регулятор целых чисел имеет значение больше 0).



Цепь цилиндров по-прежнему динамически связана со всеми регуляторами. Перемещайте регуляторы и вы увидите, как изменится картина.

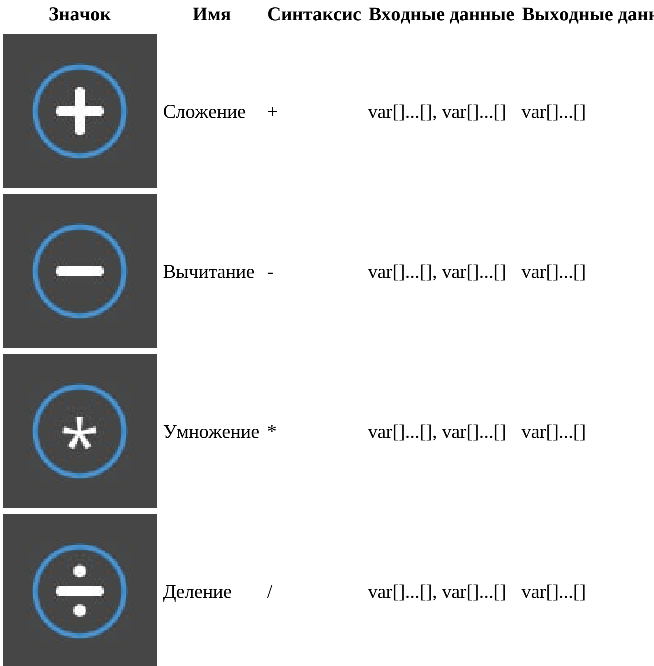
Математика

Математика

Числа являются самой простой формой данных, а самым простым способом связать эти числа между собой является математика. Начиная от элементарных операторов, таких как деление, и заканчивая тригонометрическими функциями и более сложными формулами, математика — отличный способ начать знакомство с отношениями и закономерностями в мире чисел.

Арифметические операторы

Оператор — это набор компонентов, в которых используются алгебраические функции с двумя входными числовыми значениями, результатом которых является одно выходное значение (сложение, вычитание, умножение, деление и т. д.). Они находятся в разделе «Операторы» > «Действия».

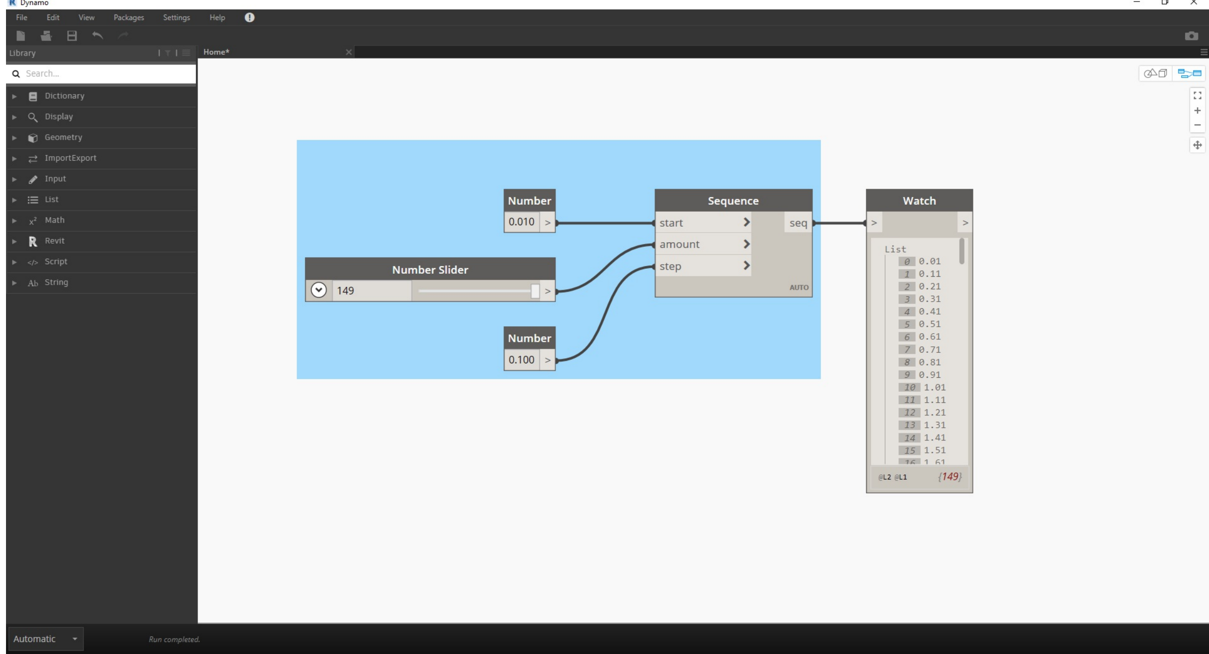


Значок	Имя	Синтаксис	Входные данные	Выходные данные
	Сложение	+	var[]...[], var[]...[]	var[]...[]
	Вычитание	-	var[]...[], var[]...[]	var[]...[]
	Умножение	*	var[]...[], var[]...[]	var[]...[]
	Деление	/	var[]...[], var[]...[]	var[]...[]

Параметрическая формула

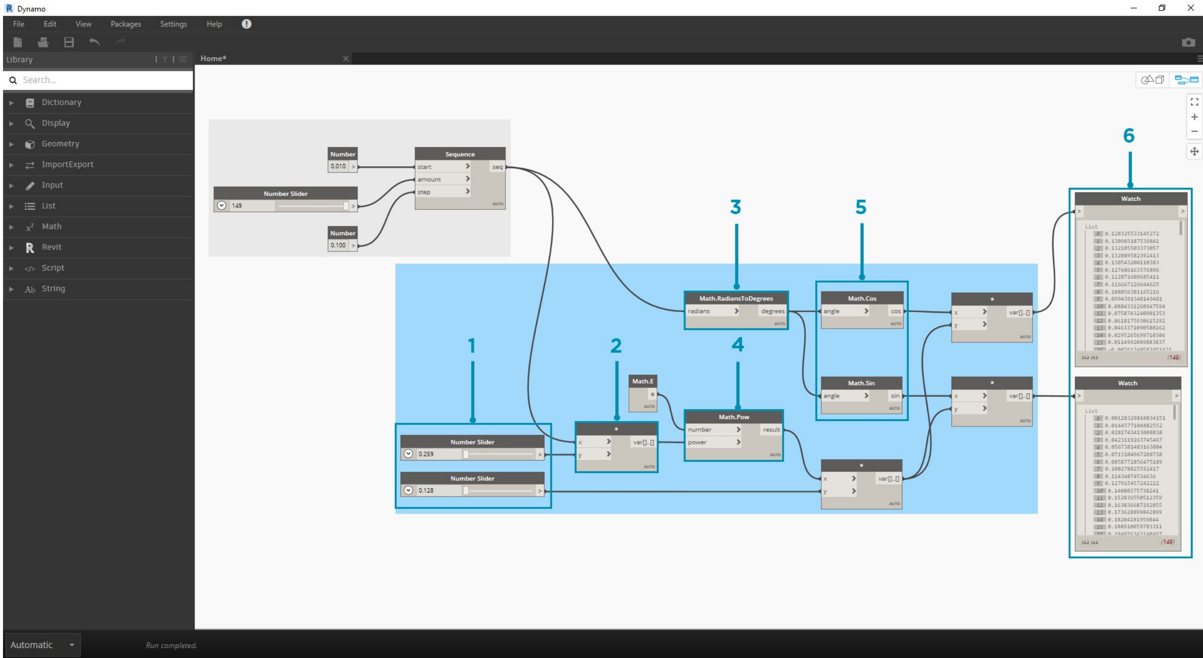
Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): Building Blocks of Programs - Math.dyn. Полный список файлов примеров можно найти в приложении.

Следующий логический шаг при работе с операторами — их комбинирование с переменными для формирования более сложных отношений с помощью **формул**. Создадим формулу, которой могут управлять входные параметры, такие как регуляторы.



1. **Number Sequence.** Определим последовательность чисел на основе трех входных значений: *start, amount* и *step*. Эта последовательность представляет «t» в параметрическом уравнении, поэтому необходим достаточно большой список для определения спирали.

В предыдущем шаге был создан список чисел для определения области параметрических компонентов. Золотая спираль определяется по формуле $x = r\cos\theta = a\cos\theta\, e^{b\theta}$ и $y = r\sin\theta = a\sin\theta\, e^{b\theta}$. Данное уравнение представлено ниже в виде группы узлов в среде визуального программирования.

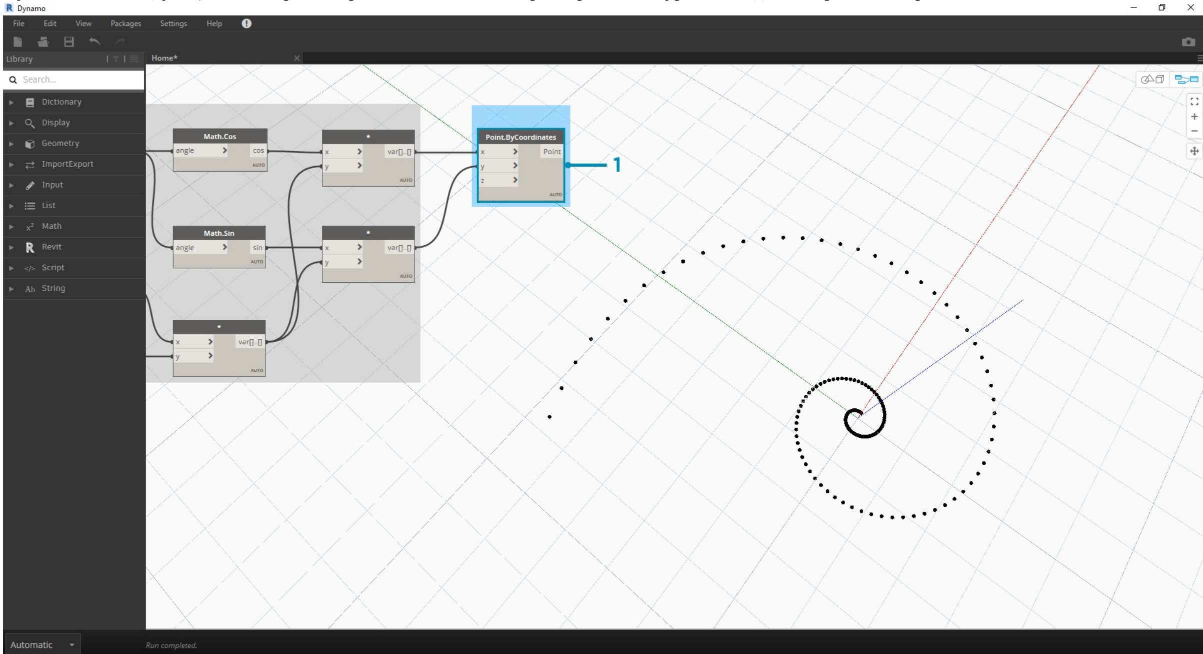


Рассматривая группу узлов, обратите внимание на соответствие между визуальной программой и уравнением в записи.

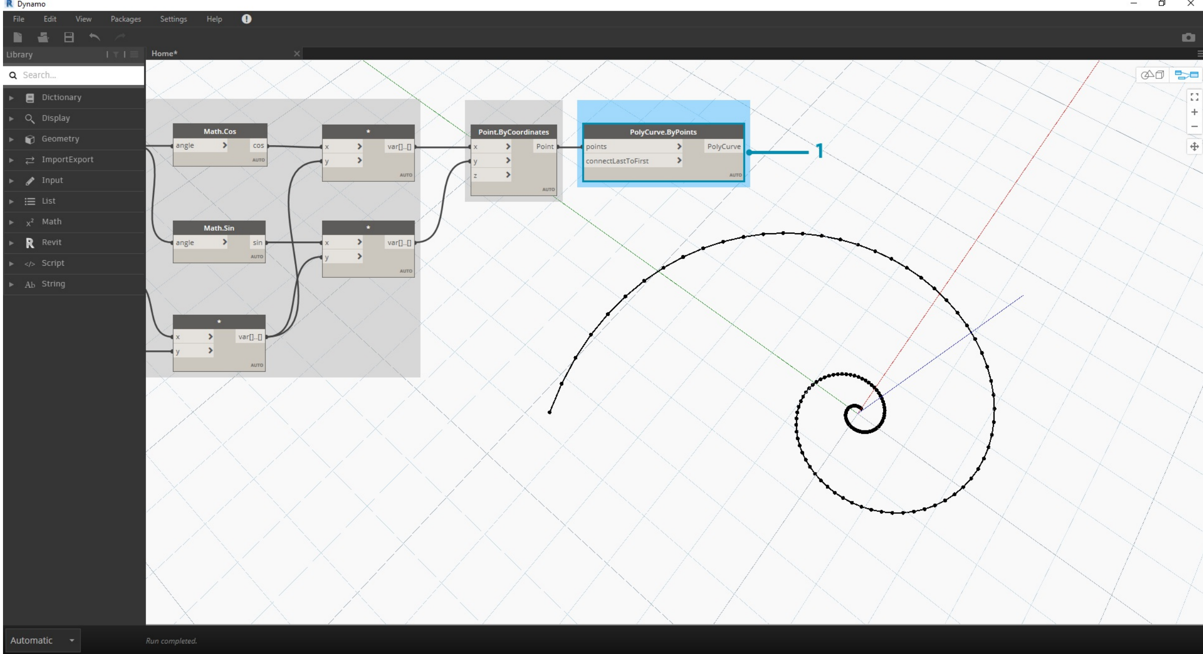
1. **Number Slider.** Добавьте два регулятора чисел в рабочую область. Эти регуляторы будут задавать переменные *a* и *b* параметрического уравнения. Они представляют собой гибкую константу или параметры, которые можно настроить для получения желаемого результата.
2. ***.** Узел умножения обозначен звездочкой. Он будет часто использоваться для соединения умножаемых переменных
3. **Math.RadiansToDegrees.** Значения «*t*» необходимо преобразовать в градусы для их оценки в тригонометрических функциях. Помните, что для оценки этих функций в Dynamo по умолчанию используются градусы.
4. **Math.Pow.** В качестве функции «*t*» и числа «*e*» этот узел создает последовательность Фибоначчи.
5. **Math.Cos и Math.Sin.** С помощью этих двух тригонометрических функций будут различаться координаты X и Y (соответственно) для каждой параметрической точки.
6. **Watch.** В качестве выходных данных отображается два списка, которые будут выступать в качестве координат *x* и *y* точек, используемых для формирования спирали.

От формулы к геометрии

Хотя набор узлов из предыдущего этапа будет выполнять поставленные задачи, этот процесс довольно трудоемкий. Для повышения эффективности работы в разделе **Блоки кода** (3.3.2.3) ознакомьтесь со сведениями о том, как в одном узле разместить строку выражений Dynamo. На последующих этапах рассмотрим использование параметрического уравнения для построения спирали Фибоначчи.



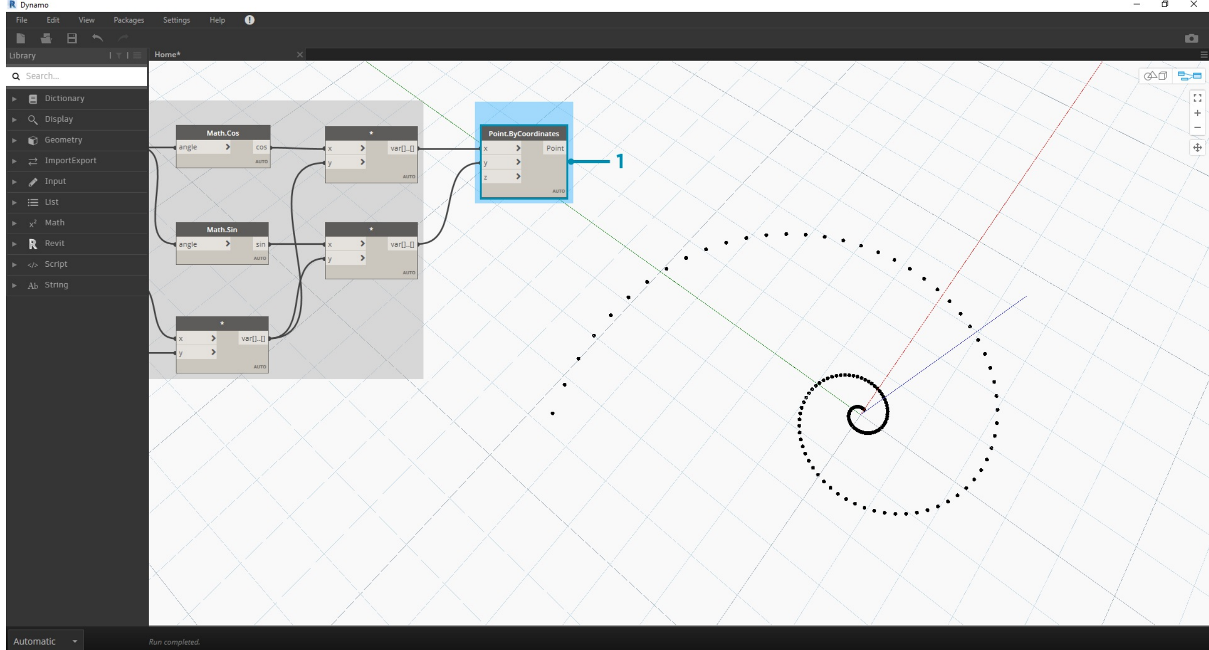
1. **Point.ByCoordinates.** Соедините верхний узел умножения с входным параметром *x*, а нижний с входным параметром *y*. На экране отобразится параметрическая спираль, проходящая через точки.



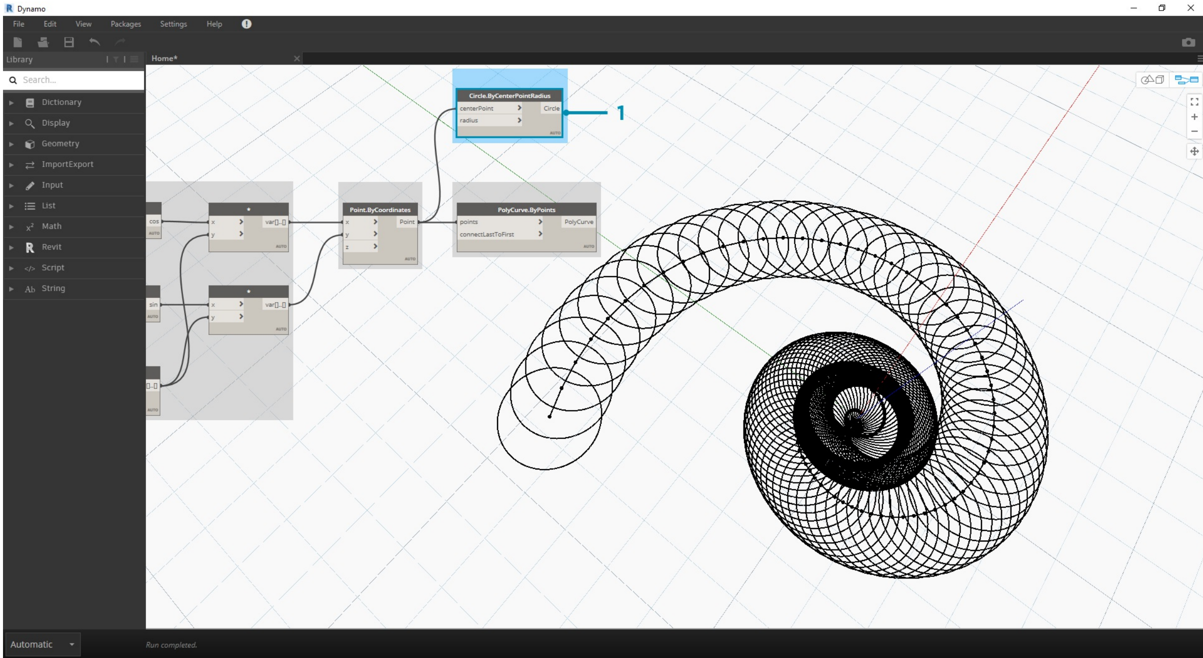
1. **Polycurve.ByPoints.** Соедините узел Point.ByCoordinates из предыдущего шага с входным параметром *points*. Параметр *connectLastToFirst* можно оставить без входных данных, поскольку мы не будем создавать замкнутую кривую. Таким образом, получаем спираль, которая проходит через каждую точку, заданную в предыдущем шаге.

Спираль Фибоначчи создана. Продолжим работу и выполним еще два упражнения, которые назовем «Наутилус» и «Подсолнух». Продемонстрируем два варианта использования спирали Фибоначчи на примере этих представителей естественных систем.

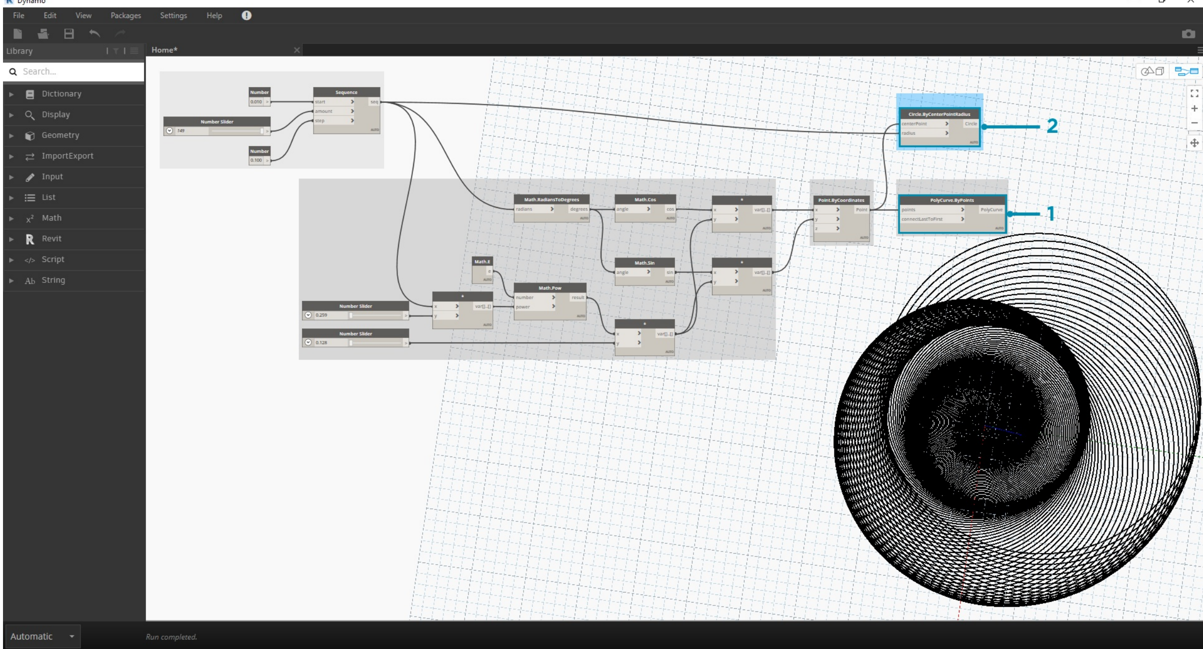
От спирали к наутилусу



1. Начнем с того же шага, что и в предыдущем упражнении: создадим массив точек спирали с помощью узла **Point.ByCoordinates**.



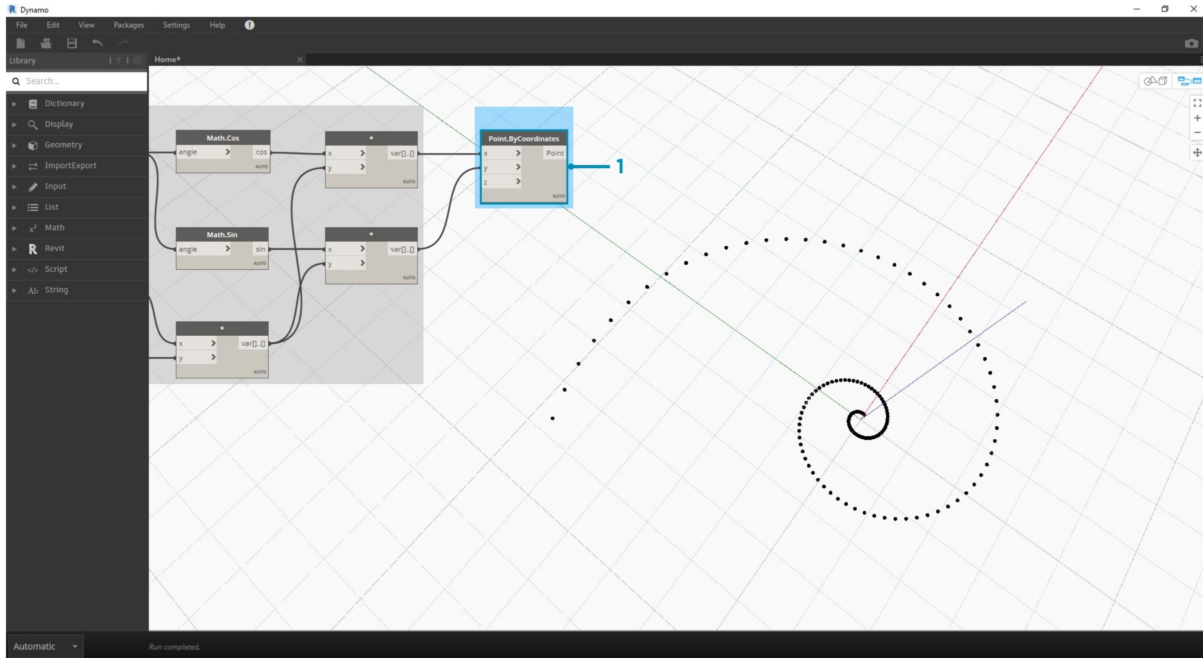
1. **Polycurve.ByPoints.** Этот узел из предыдущего упражнения будет использоваться в качестве базового.
2. **Circle.ByCenterPointRadius.** Узел построения окружности будет иметь те же входные данные, что и в предыдущем шаге. Значение радиуса по умолчанию равно *1.0*, поэтому окружности создаются сразу. Четко видно, каким образом точки отклоняются от начала координат.



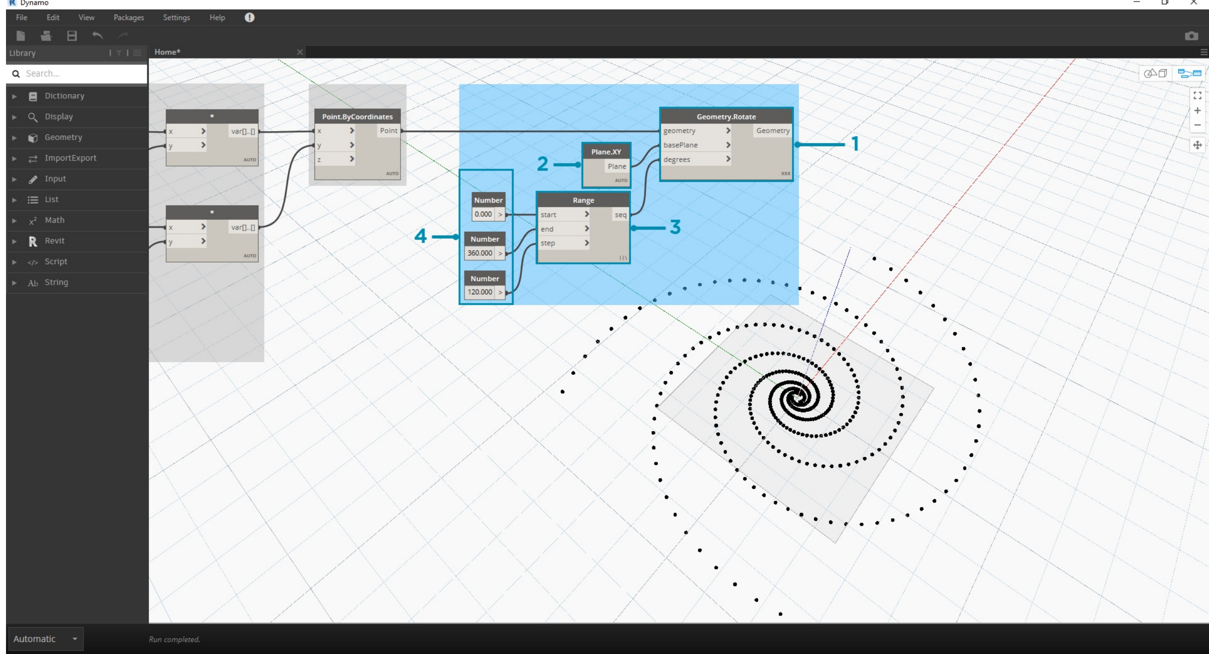
1. **Circle.ByCenterPointRadius.** Чтобы создать более динамический массив окружностей, соединим исходную последовательность чисел (последовательность «*t*») со значением радиуса.
2. **Number Sequence.** Это исходный массив элементов «*t*». Если соединить его со значением радиуса, центры окружностей будут по-прежнему отклоняться дальше от начального положения, но радиус окружностей будет увеличиваться, создавая необычный график спирали Фибоначчи. 3D-изображение этого объекта будет выглядеть замечательно.

От наутилуса к подсолнуху

После создания раковины наутилуса перейдем к параметрическим сеткам. Используя основной угол вращения спирали Фибоначчи, создадим сетку Фибоначчи, а на ее основе — модель расположения семян цветка подсолнуха.

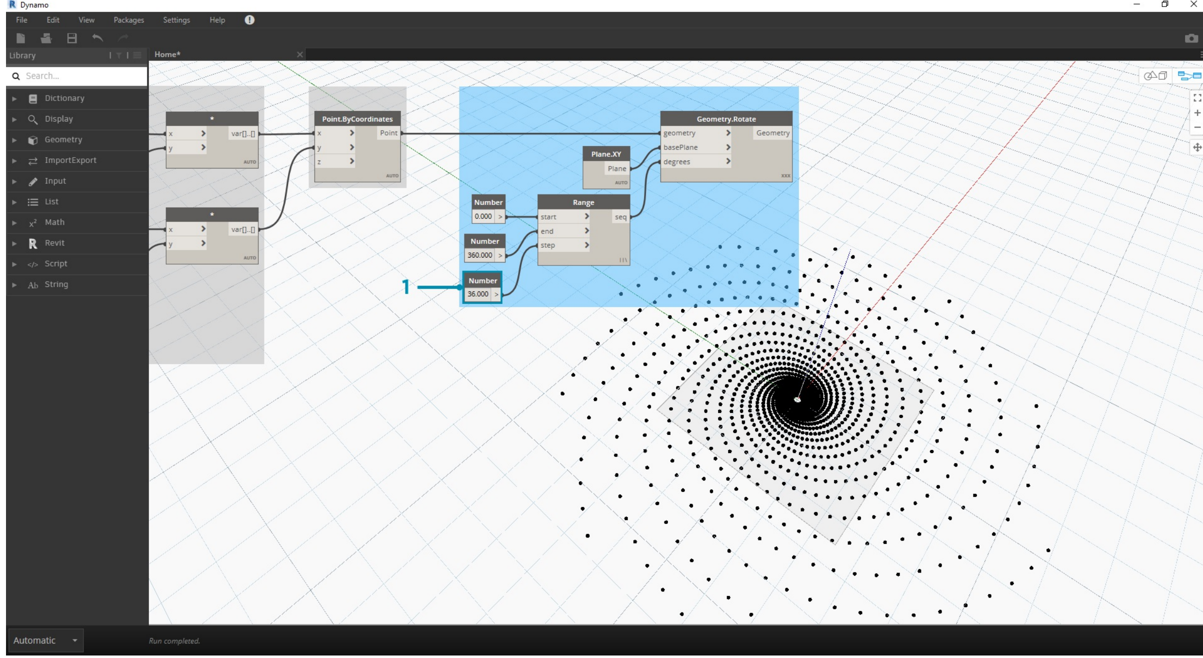


1. Начнем с того же шага, что и в предыдущем упражнении: создадим массив точек спирали с помощью узла **Point.ByCoordinates**.

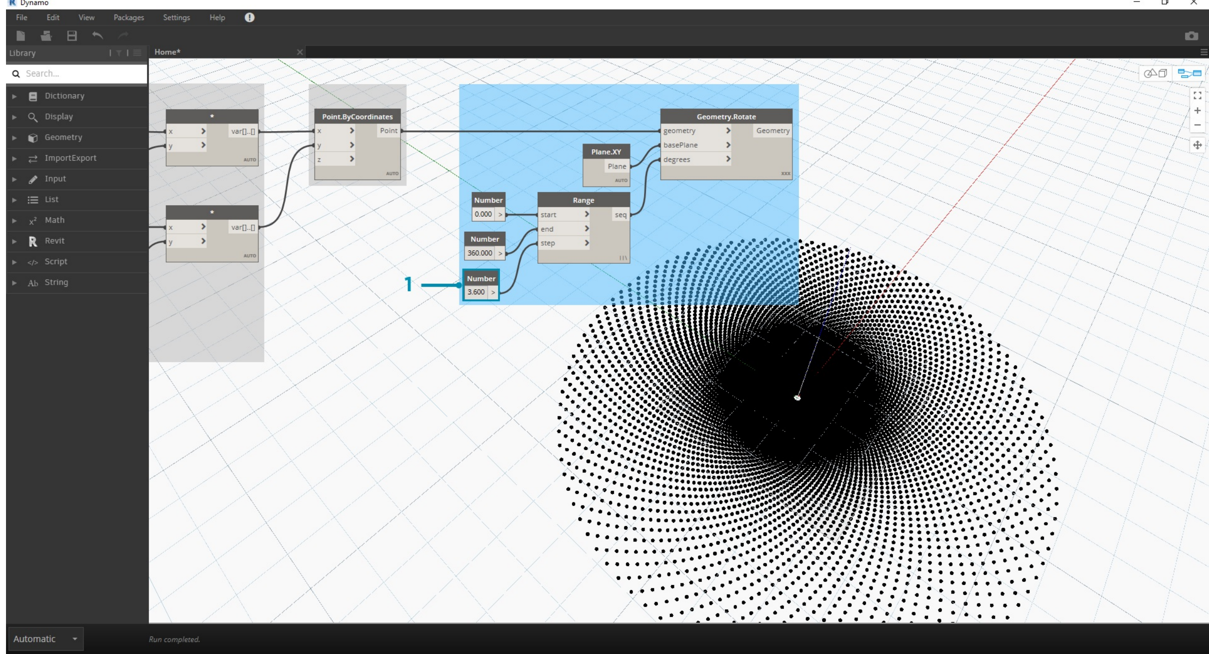


1. **Geometry.Rotate.** Существует несколько вариантов узла Geometry.Rotate. Убедитесь, что выбран узел с входными параметрами *geometry*, *basePlane* и *degrees*. Соедините узел **Point.ByCoordinates** с входным параметром geometry.
2. **Plane.XY.** Соедините узел с входным параметром *basePlane*. Вращение будет выполняться вокруг начала координат, которое совпадает с основанием спирали.
3. **Number Range.** Для входного параметра значений градусов необходимо создать несколько вращений. Это можно сделать быстро с помощью компонента Number Range. Соедините его с входным параметром *degrees*.
4. **Number.** Чтобы задать диапазон чисел, добавьте три узла Number в рабочую область вертикально. В нисходящей последовательности назначьте значения *0.0*,*360.0* и *120.0* соответственно. Они будут определять вращение спирали. Обратите внимание на результаты выходного параметра из узла **Number Range** после соединения с ним трех узлов Number.

Полученное изображение начинает напоминать водоворот. Скорректируем некоторые параметры **Number Range** и посмотрим, как изменятся результаты.



1. Измените размер шага в узле **Number Range**, задав вместо значения *120.0* значение *36.0*. Обратите внимание, что при этом генерируется больше вращений, и, следовательно, создается более плотная сетка.



1. Измените размер шага в узле **Number Range**, задав вместо значения *36.0* значение *3.6*. Получится более плотная сетка, а у спирали будет отсутствовать направление. Итак, мы создали подсолнух.

Логика

Логика

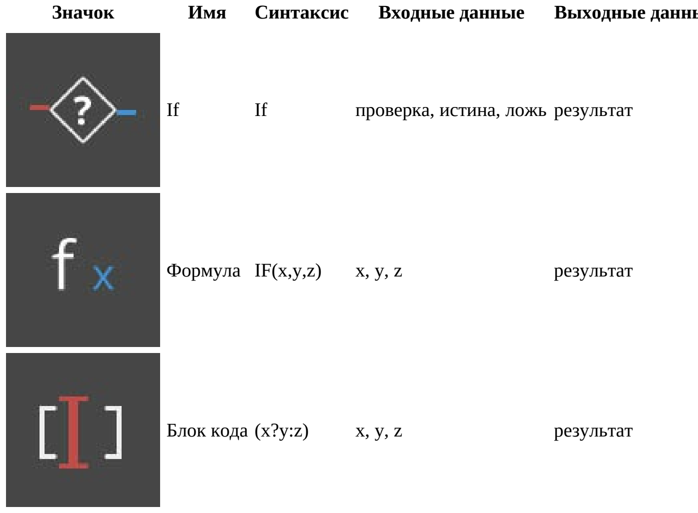
Логика (точнее **условная логика**) позволяет задать действие или набор действий в зависимости от результата проверки. После прохождения проверки выдается логическое значение `Истина` или `Ложь`, которое можно использовать для управления ходом программы.

Логические выражения

Числовые переменные могут хранить целый диапазон различных чисел. В логических переменных хранятся только два значения: «Истина» или «Ложь», да или нет, 1 или 0. Из-за ограниченной применимости логические операции можно не так часто встретить в расчетах.

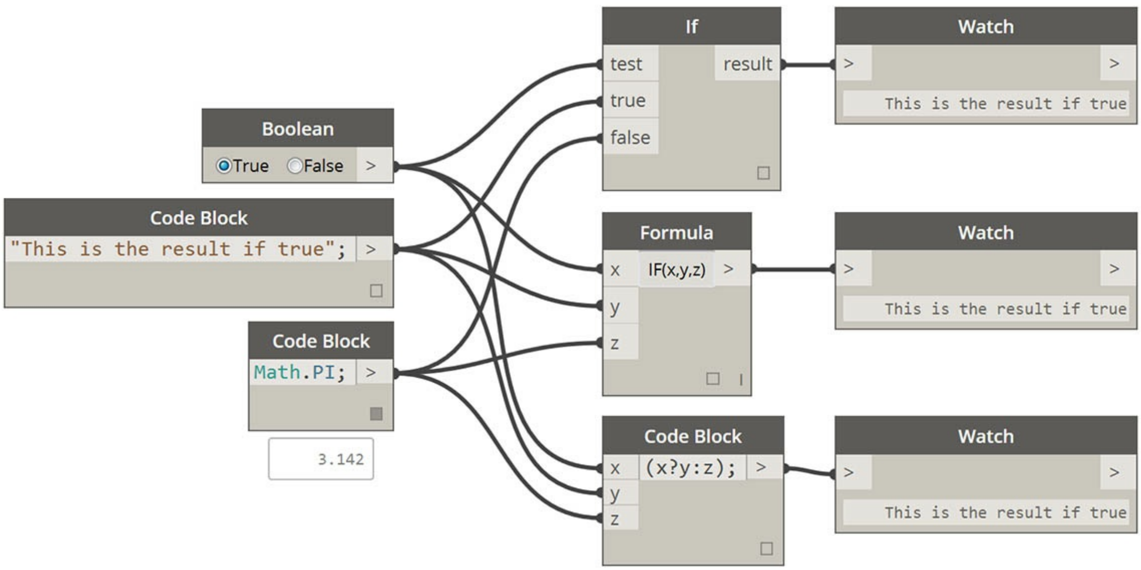
Условные выражения

Выражение If является ключевым для программирования. Если *некое условие* истинно, то происходит*что-то одно*, в противном случае происходит *что-то другое*. Действие, выполняемое после проверки с помощью данного выражения, зависит от логического значения. Существует несколько способов определения выражения If в Dynamo.

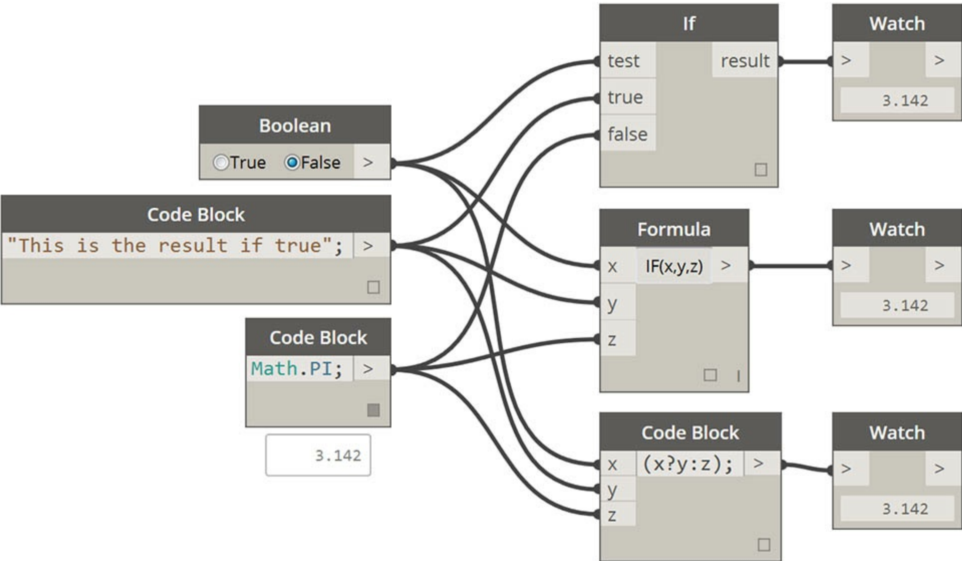


Значок	Имя	Синтаксис	Входные данные	Выходные данные
	If	If	проверка, истина, ложь	результат
	Формула	IF(x,y,z)	x, y, z	результат
	Блок кода	(x?y:z)	x, y, z	результат

Рассмотрим краткий пример с каждым из этих трех узлов в действии, используя условное выражение If.



На этом изображении в узле *boolean* задано значение *true*. Это означает, что на выводе появится строка *this is the result if true (это результат при истинном значении).* Три узла, образующие выражение *If*, работают одинаково.

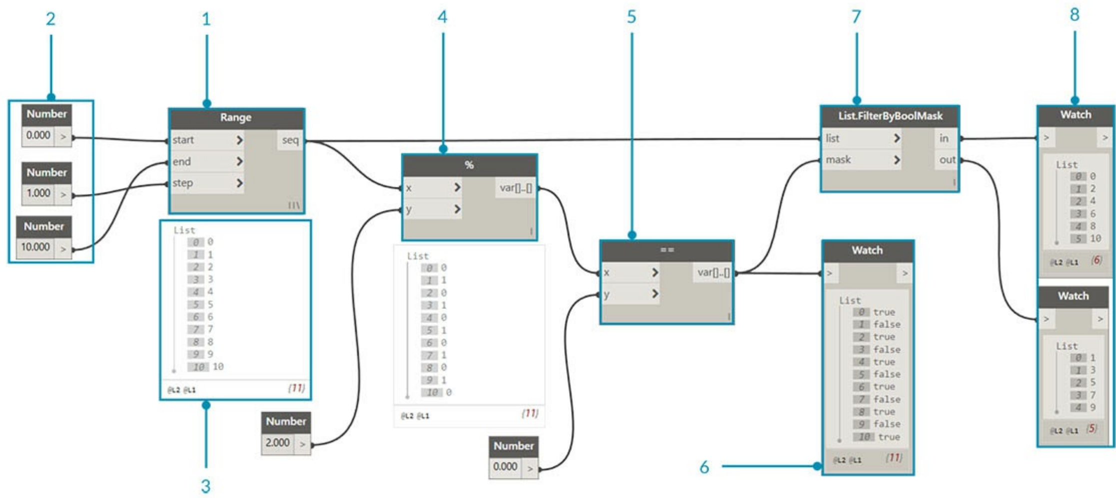


Напомним, что узлы работают одинаково. Если изменить значение *boolean* на *false*, результатом будет число *пи*, как определено в исходном операторе *If*.

Фильтрация списка

Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): Building Blocks of Programs - Logic.dyn. Полный список файлов примеров можно найти в приложении.

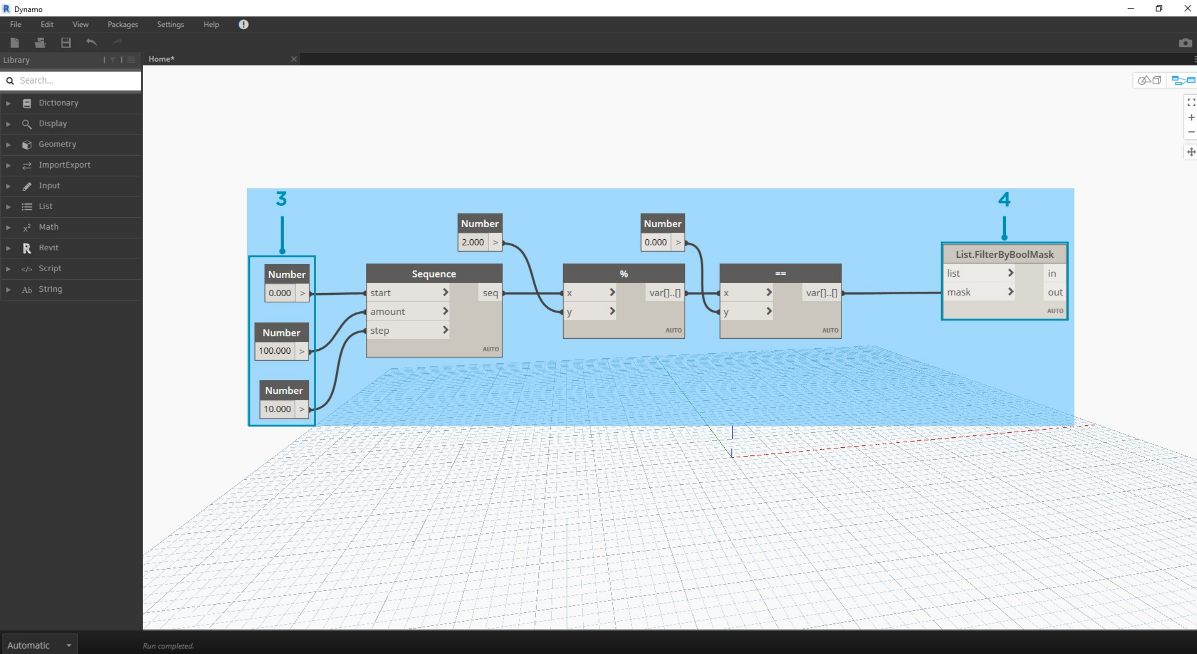
Используем логику, чтобы разделить список чисел на списки четных и нечетных чисел.



1. **Number Range.** Добавьте диапазон чисел в рабочую область.
2. **Числа.** Добавьте три узла Number в рабочую область. Каждый узел Number должен иметь следующие значения: *0.0* для *start*, *10.0* для *end* и *1.0* для *step*.
3. **Выходные данные.** На выходе получается список из 11 чисел в диапазоне от 0 до 10.
4. **Коэффициент (%).** *Диапазон номеров* в качестве входных данных для *x* и значение *2.0* в качестве входных данных для *y*. При этом рассчитывается остаток каждого числа в списке при делении на 2. На выходе из этого списка будет представлен список чередующихся значений 0 и 1.
5. **Проверка равенства (==).** Добавьте в рабочую область проверку равенства. Соедините выходные данные *коэффициента*с входным параметром *x*, а значение *0.0* с входных параметром *y*.
6. **Watch**. В качестве выходных данных проверки равенства будет представлен список значений «Истина» или «Ложь». С помощью этих значений элементы будут разделяться в списке. *0* (или *true*) соответствует четным числам, а *1* (или *false*) — нечетным.
7. **List.FilterByBoolMask.** Этот узел отфильтровывает значения по двум разным спискам в зависимости от вводимого логического выражения. Соедините исходный *диапазон чисел* с входным параметром *list*, а выходной параметр *проверки равенства* с входным параметром *mask*. В выходных данных *in* представлены истинные значения, а в *out* — ложные.
8. **Watch.** В результате мы получили списки четных и нечетных чисел. Итак, с помощью логических операторов мы разделили списки по определенному признаку.

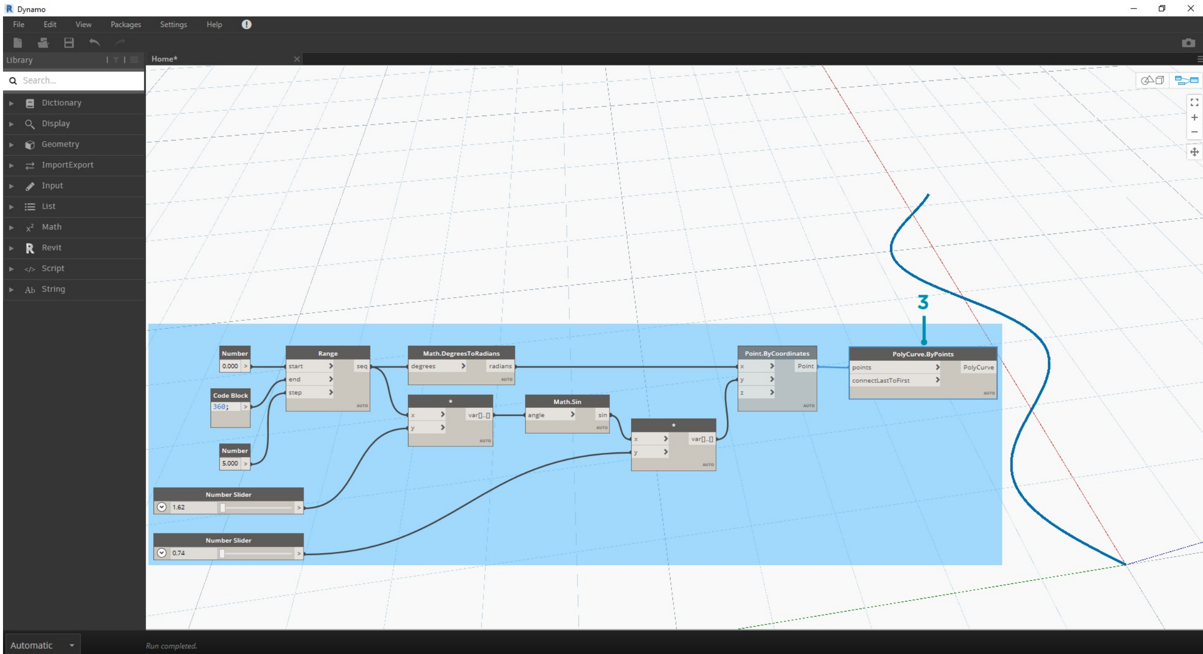
От логики к геометрии

Применим логику из первого упражнения к моделированию.



За основу возьмем предыдущее упражнение с теми же узлами. Помимо изменения формата единственными исключениями будут следующие.

1. Изменены входные значения.
2. Отсоединен входной параметр list из узла *List.FilterByBoolMask*. Эти узлы пока не нужны, но они потребуются позже.

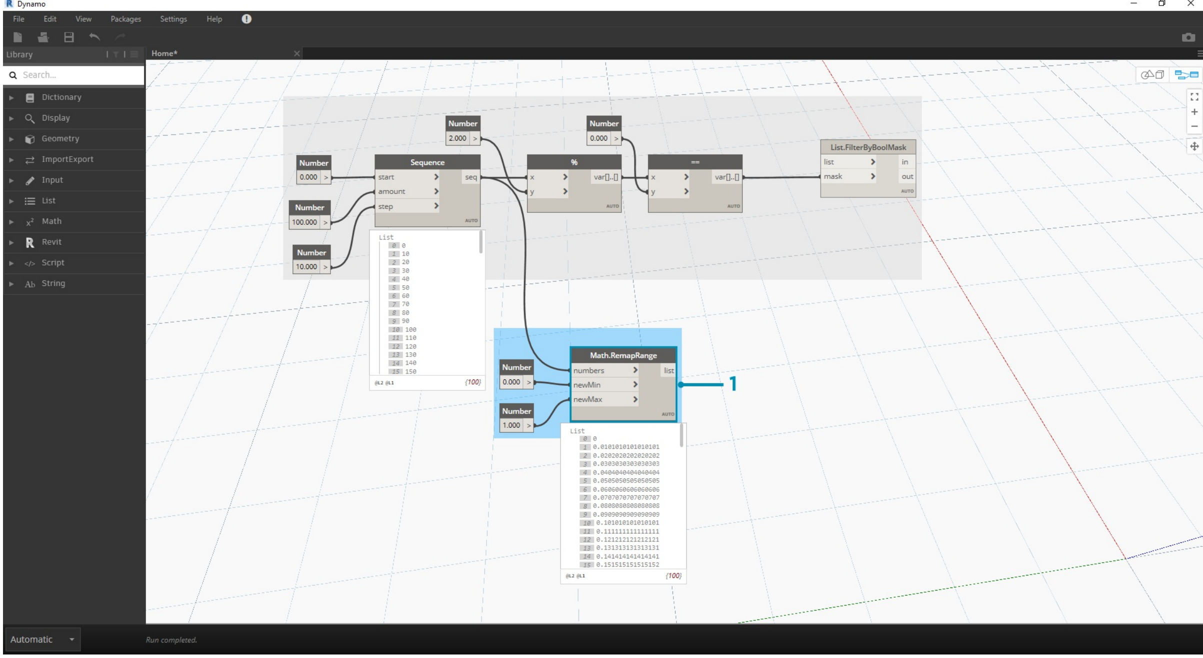


Начнем с соединения узлов вместе, как показано на изображении выше. Эта группа узлов представляет собой параметрическое уравнение для определения линейной кривой. Примечания.

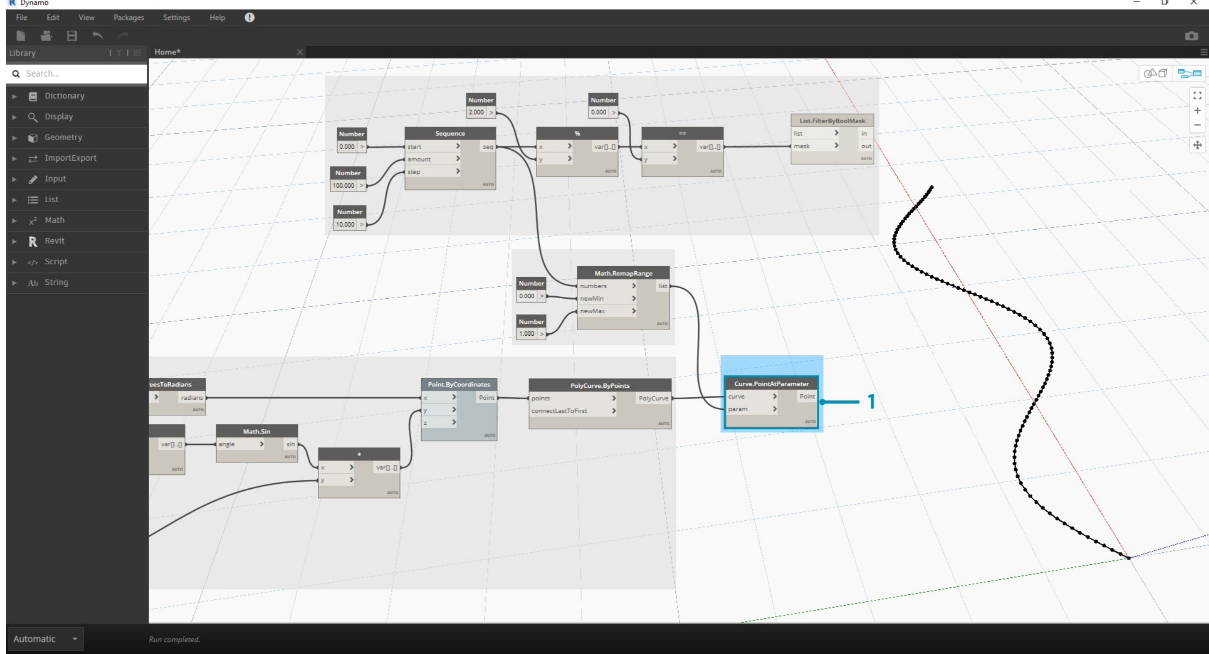
1. **Первый регулятор** должен иметь значение не менее 1, не более 4 и шаг 0,01.
2. **Второй регулятор** должен иметь значение не менее 0, не более 1 и шаг 0,01.
3. **PolyCurve.ByPoints.** Если скопировать приведенную выше схему узлов, на видовом экране предварительного просмотра Dynamo будет создана синусоидальная кривая.

При вводе данных используйте числовые узлы для более статических свойств и регуляторы чисел для более гибких свойств. Необходимо сохранить исходный диапазон чисел, который определяется в начале этого шага. Однако синусоидальная кривая, которую мы пытаемся создать, должна обладать определенной гибкостью. Перемещая регуляторы, можно видеть частотные и амплитудные изменения кривой.

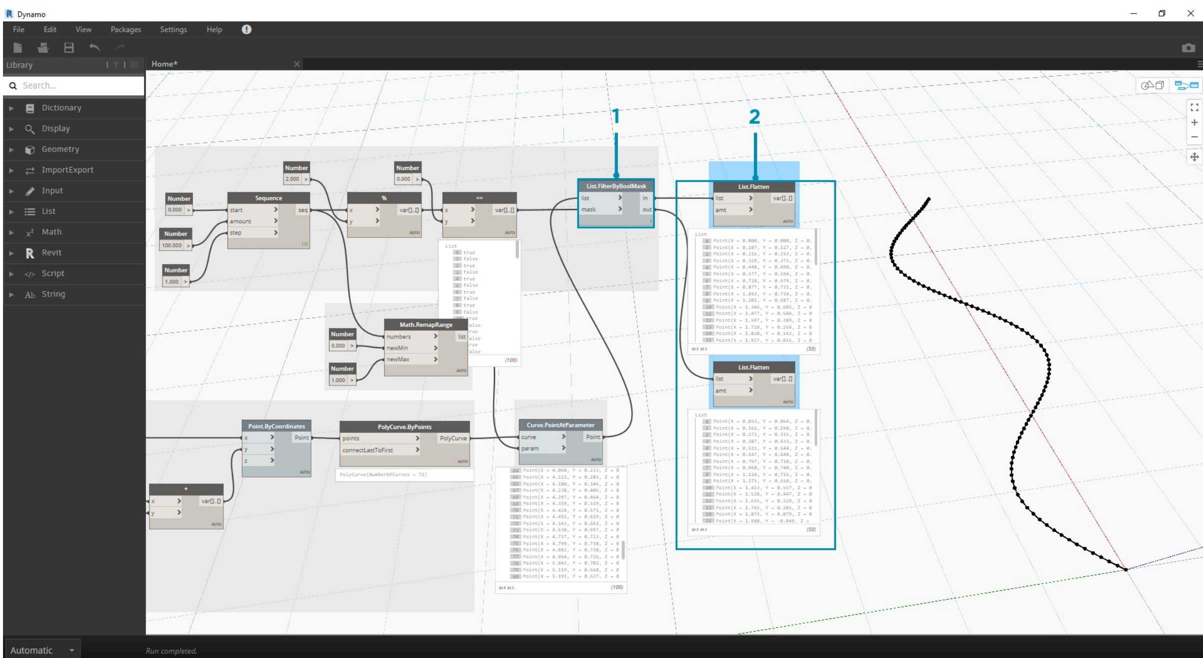
Заглянем немного вперед и посмотрим на конечный результат, чтобы представить, каким он должен быть. Первые два шага выполняются отдельно, теперь их нужно соединить. С помощью базовой синусоидальной кривой будет определяться местоположение компонентов молнии, а с помощью логики «истина/ложь» — элементы меньшего или большего размера.



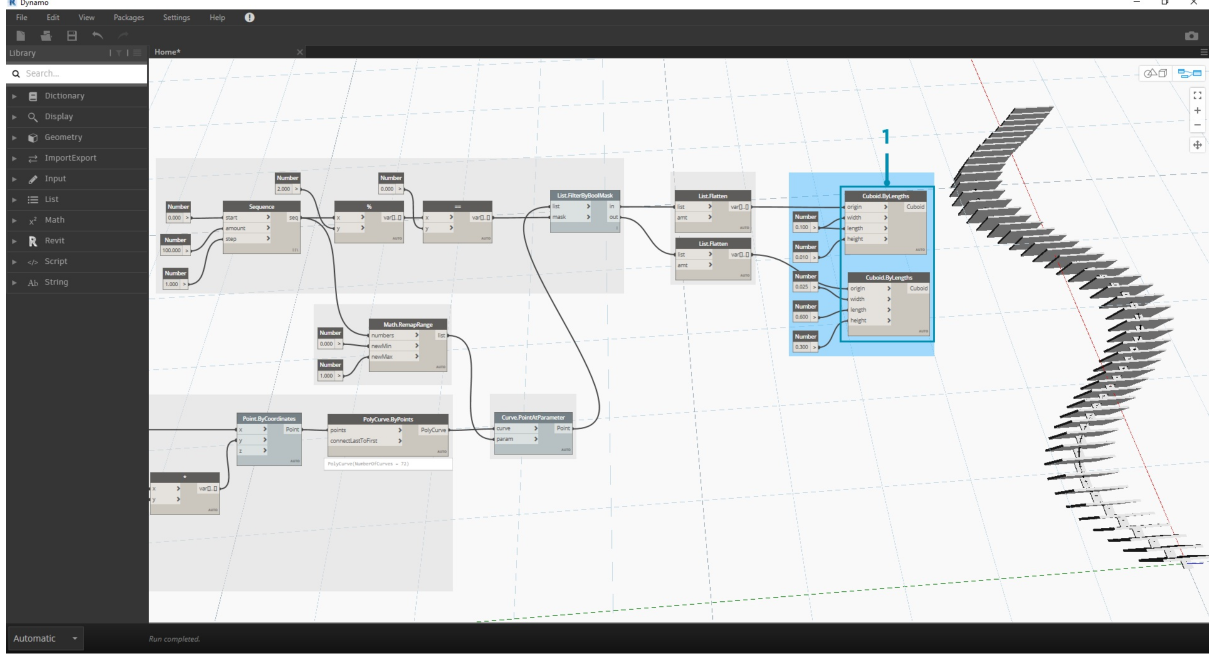
1. **Math.RemapRange.** С помощью последовательности чисел, созданной в шаге 01, сформируем новую последовательность чисел, перенастроив диапазон. Исходные числа из шага 01 имеют диапазон от 0 до 100. С помощью входных параметров *newMin* и *newMax* диапазон значений изменяется на 0–1 соответственно.



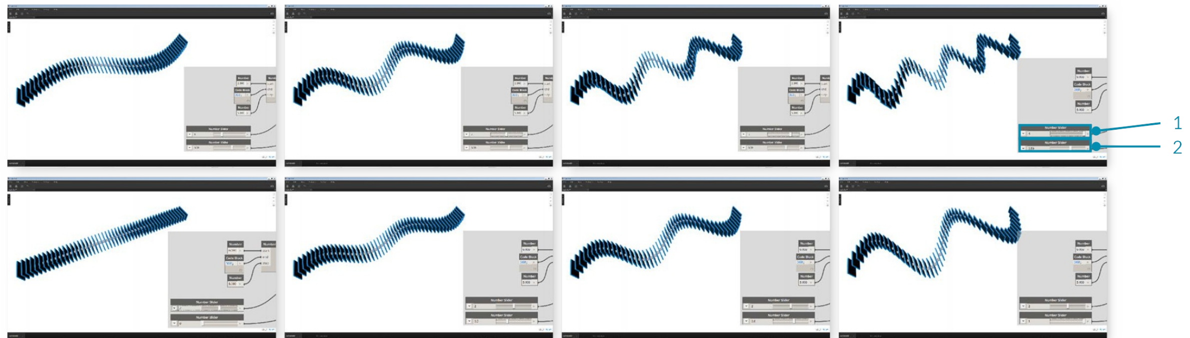
1. **Curve.PointAtParameter.** Соедините узел *Polycurve.ByPoints* (из шага 2) с входным параметром *curve*, а узел *Math.RemapRange* с входным параметром *param*. В этом шаге создаются точки вдоль кривой. Диапазон чисел был перенастроен на 0–1, так как входной параметр *param* ищет значения в этом диапазоне. Значение *0* соответствует начальной точке, а значение *1* — конечным точкам. Все промежуточные числа относятся к диапазону *[0,1]*.



1. **List.FilterByBoolMask.** Соедините узел *Curve.PointAtParameter* из предыдущего шага с входным параметром *list*.
2. **Watch.** Узел Watch для *in* и узел Watch для *out* показывают, что имеется два списка — четных и нечетных индексов. Тот же самый порядок точек используется в кривой, что демонстрируется в следующем шаге.



1. **Cuboid.ByLength.** Для создания молнии вдоль синусоидальной кривой воспроизведите связи, представленные на изображении выше. В данном случае кубоид — это просто рамка, размер которой определяется в зависимости от точки кривой в центре рамки. Теперь логика четных/нечетных делений в модели должна быть понятной.



1. **Number Slider.** Вернувшись в начало процедуры, можно переключить регулятор чисел, наблюдая, как изменится молния. В верхней серии изображений представлен диапазон значений для верхнего регулятора. Это частота волны.
2. **Number Slider.** В нижней серии изображений представлен диапазон значений для нижнего регулятора. Это амплитуда волны.

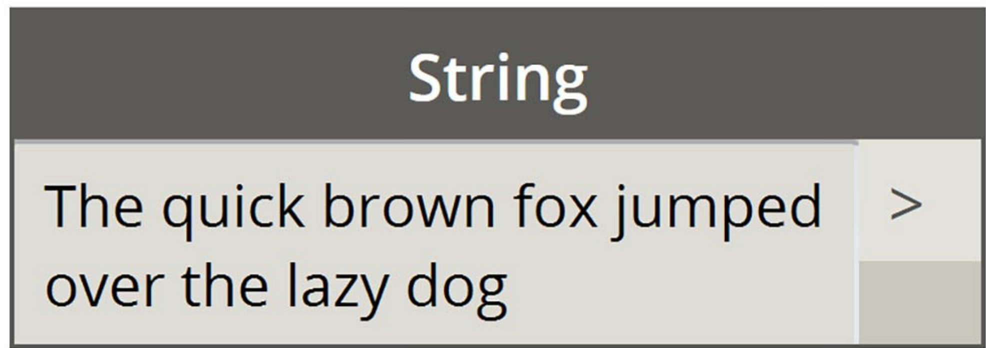
Строки

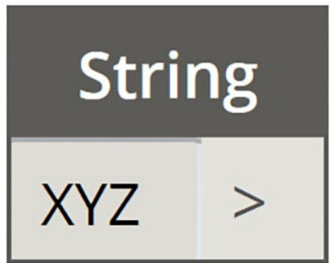
Строки

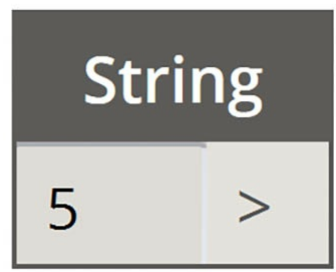
Формально **строка** — это последовательность символов, представляющих литеральную константу или переменную определенного типа. Однако на сленге программистов строкой называется любой текст. Мы уже говорили о том, как можно управлять параметрами с помощью целых и десятичных чисел. Аналогичным образом можно работать и с текстом.

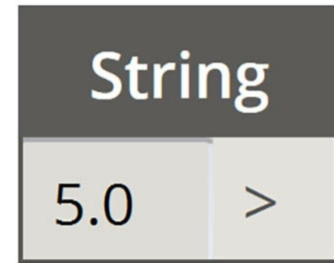
Создание строк

Строки применяются в разных ситуациях, в том числе при настройке пользовательских параметров, аннотировании документации и анализе текстовых наборов данных. Узел String находится в разделе Core > Input Category.









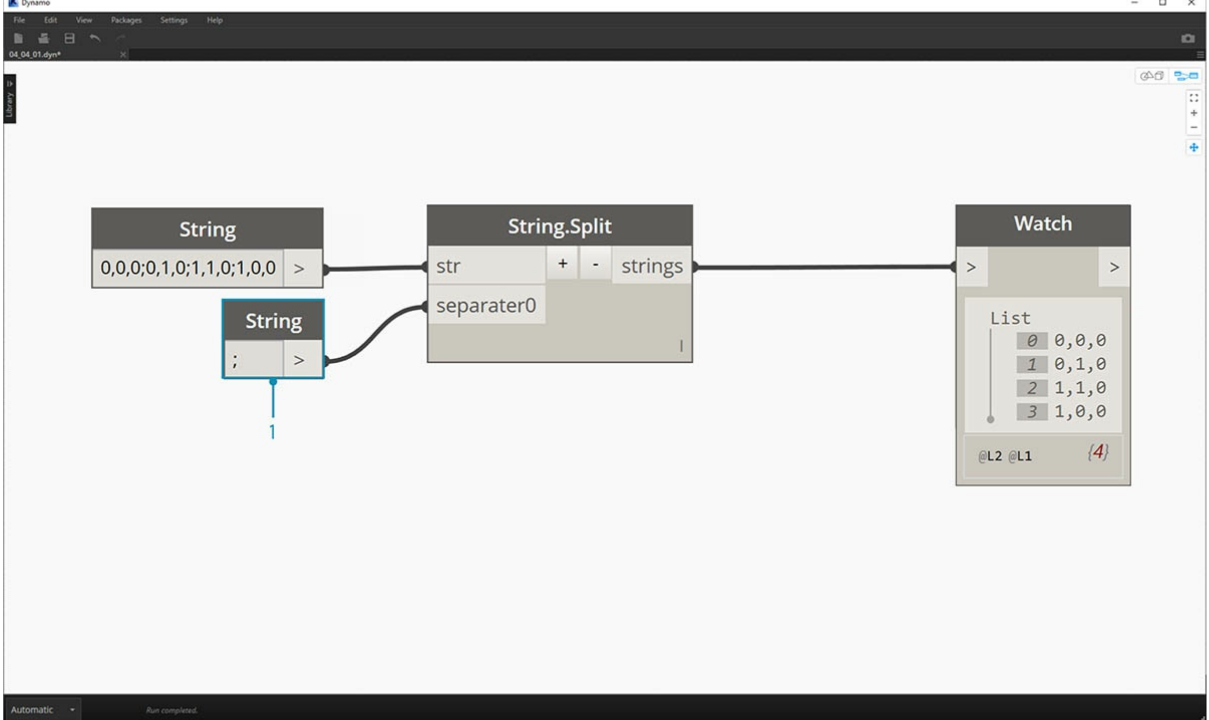
Примеры узлов выше являются строками. В виде строки может быть представлено число, буква или целый массив текста.

Запрос строк

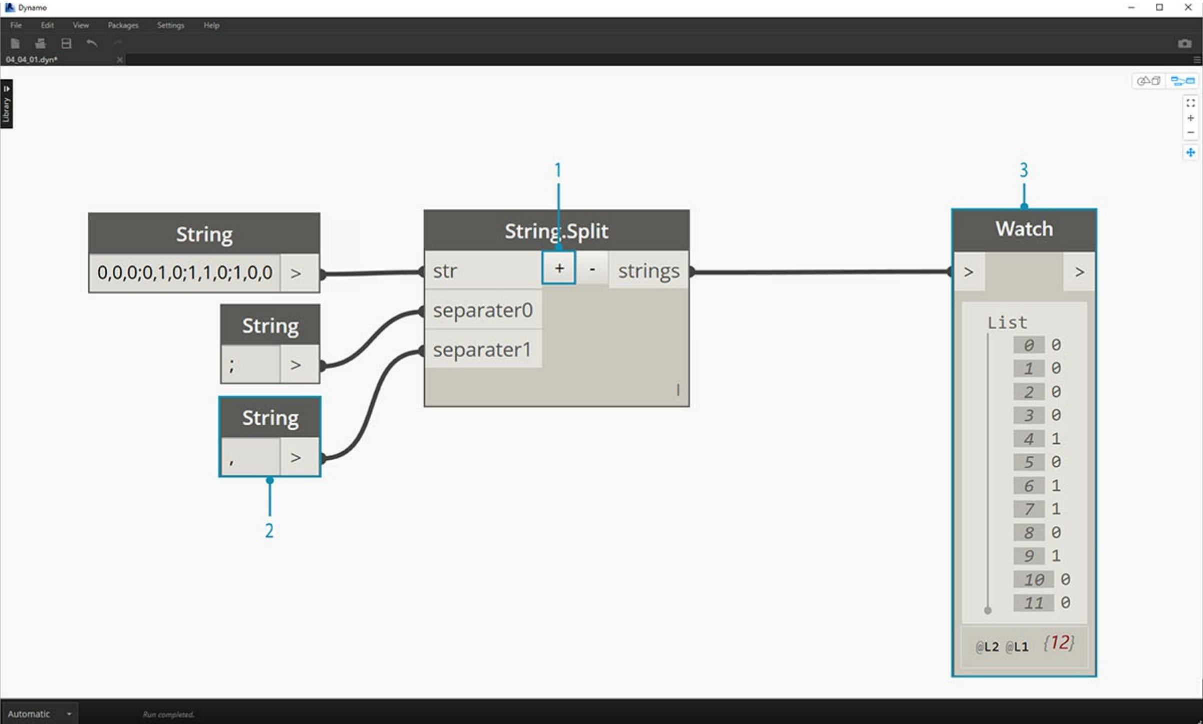
Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): [Building Blocks of Programs - Strings.dyn](). Полный список файлов примеров можно найти в приложении.

С помощью запроса строк можно быстро анализировать большие объемы данных. Поговорим о некоторых основных операциях, которые могут ускорить рабочий процесс и обеспечить совместимость программного обеспечения.

На следующем изображении показана строка данных из внешней электронной таблицы. Строка представляет вершины прямоугольника в плоскости XY. Рассмотрим подробнее некоторые операции по разделению строк в небольшом упражнении.

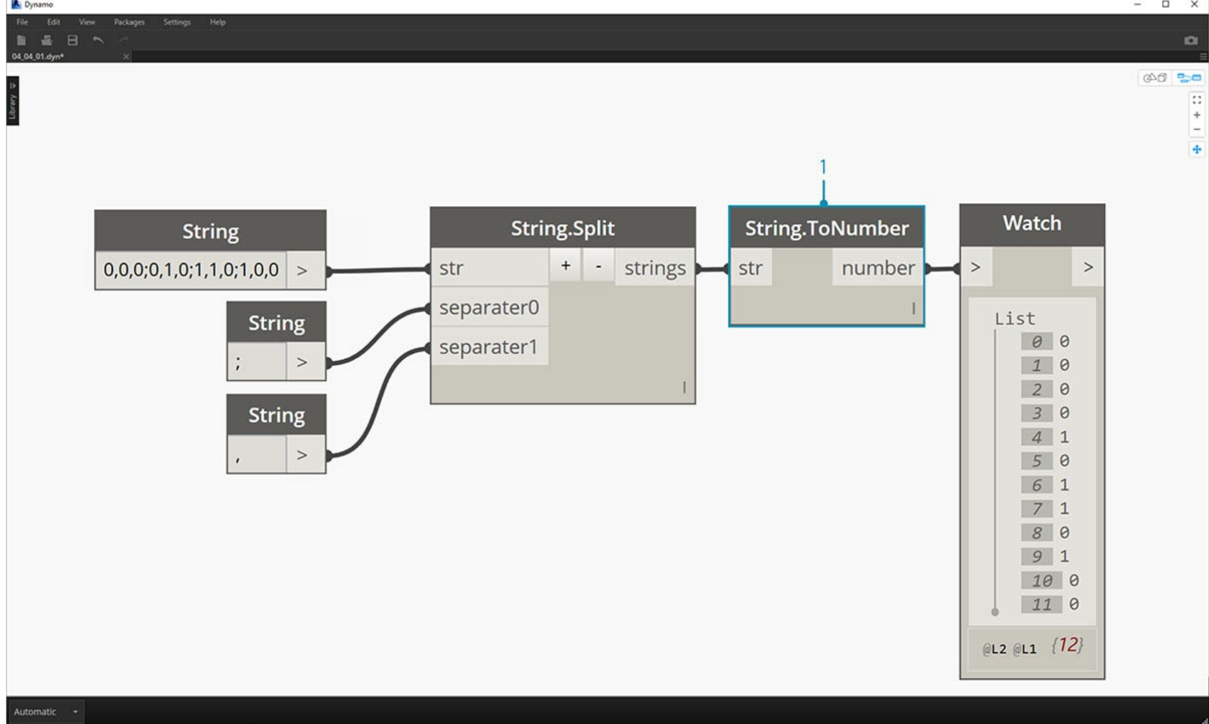


1. В качестве разделителя вершин прямоугольника используется точка с запятой «;». При этом создается список, содержащий 4 элемента для каждой вершины.

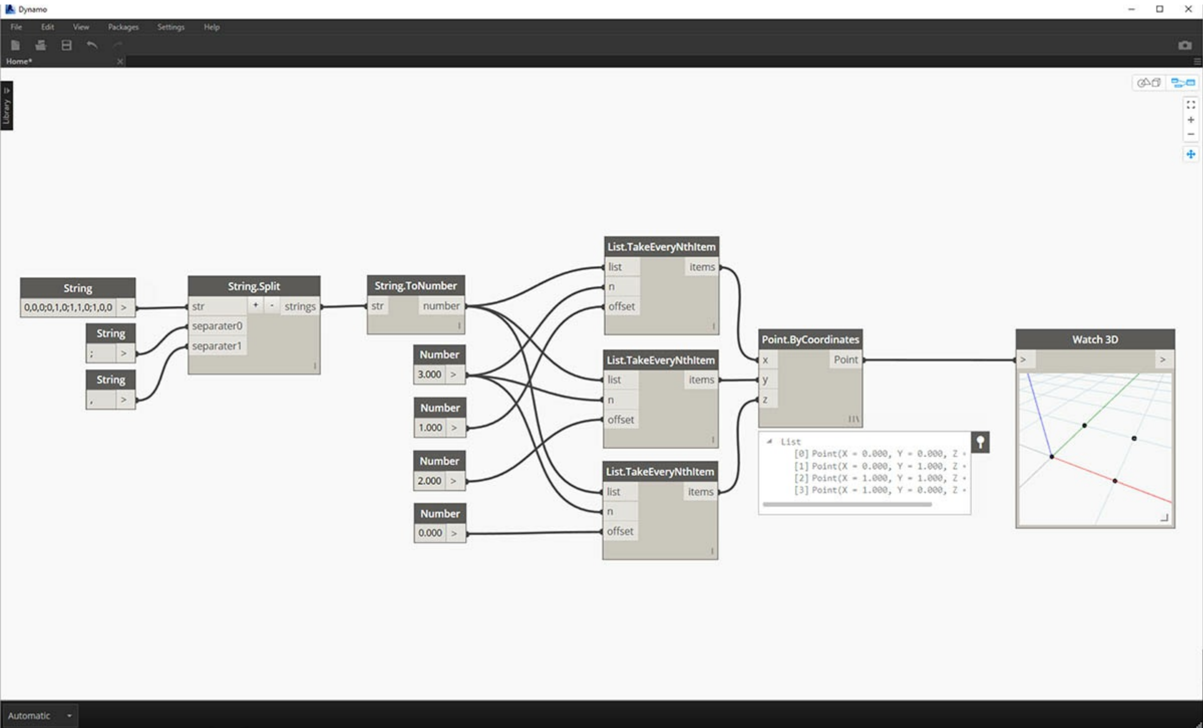


1. Если нажать знак «+» посредине узла, будет создан новый разделитель.
2. Добавьте строку «,» в рабочую область и соедините ее с новым входным параметром separator.
3. Получится список из десяти элементов. Сначала в качестве разделителя узла используется значение *separator0*, а затем — *separator1*.

Хотя элементы списка на изображении выше выглядят как числа, в Dynamo они все так же считаются отдельными строками. Для создания точек тип соответствующих данных необходимо преобразовать из строкового в числовой. Для этого используется узел String.ToNumber.



1. Этот узел достаточно прост. Соедините результаты String.Split с входным параметром. Кажется, что выходные данные не изменились, но теперь типом данных будет *number*, а не *string*.

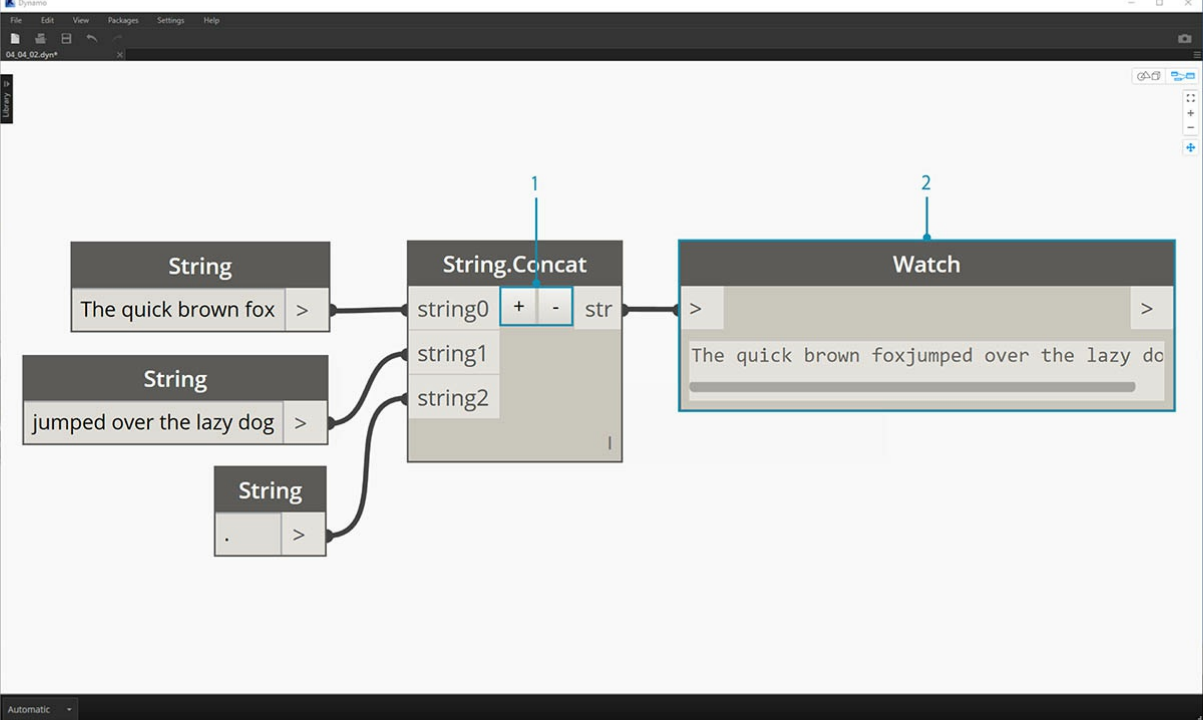


1. Добавив несколько дополнительных операций, на основе исходных строковых входных данных в начале координат создан прямоугольник.

Операции со строками

Так как строка является типовым текстовым объектом, у нее есть множество применений. Рассмотрим основные действия в разделе Core > String Category в Dynamo.

На изображении представлен метод объединения двух строк по порядку. Берется каждая литеральная строка в списке и создается одна объединенная строка.

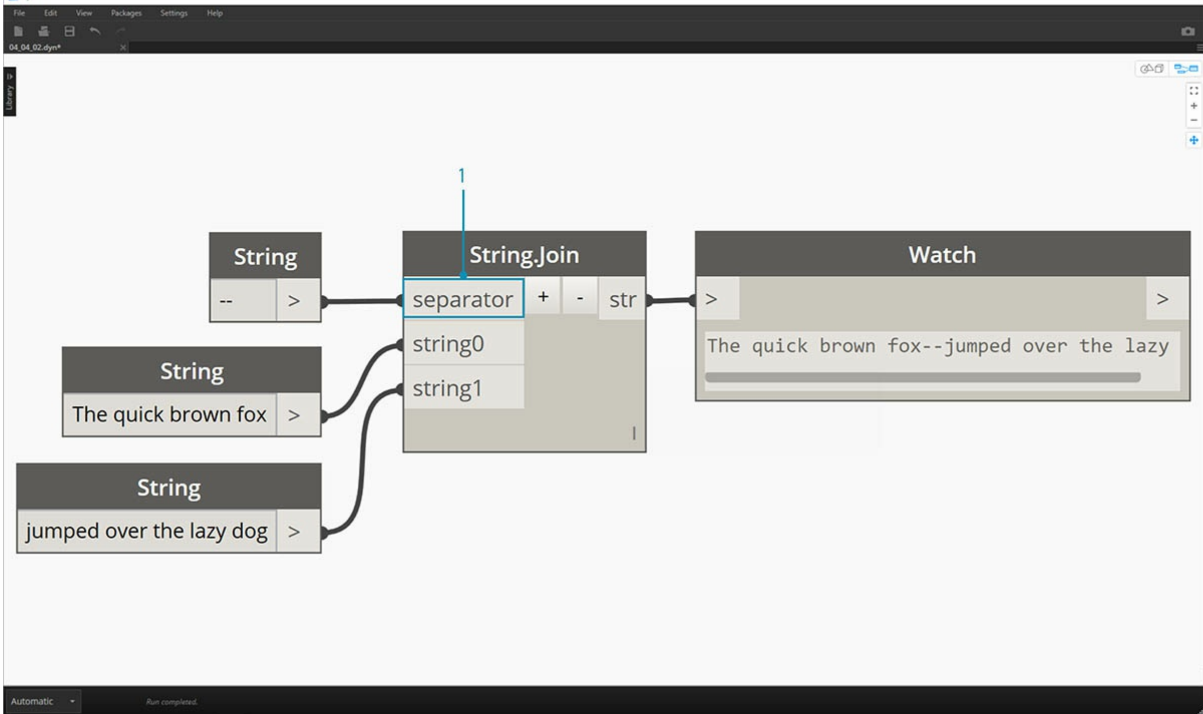


На изображении выше показана операция объединения трех строк:

1. Нажатием кнопок «+/-» в центре узла можно добавлять или удалять строки.
2. На выходе получается одна объединенная строка с пробелами и знаками препинания.

Метод соединения очень похож на метод объединения, но в нем есть дополнительный слой пунктуации.

Пользователям, работающим с Excel, могут быть знакомы файлы CSV. CSV расшифровывается как comma-separated values — значения, разделенные запятыми. Чтобы создать данные с аналогичной структурой, в узле соединения в качестве разделителя можно использовать запятую (или, в данном случае, два дефиса).



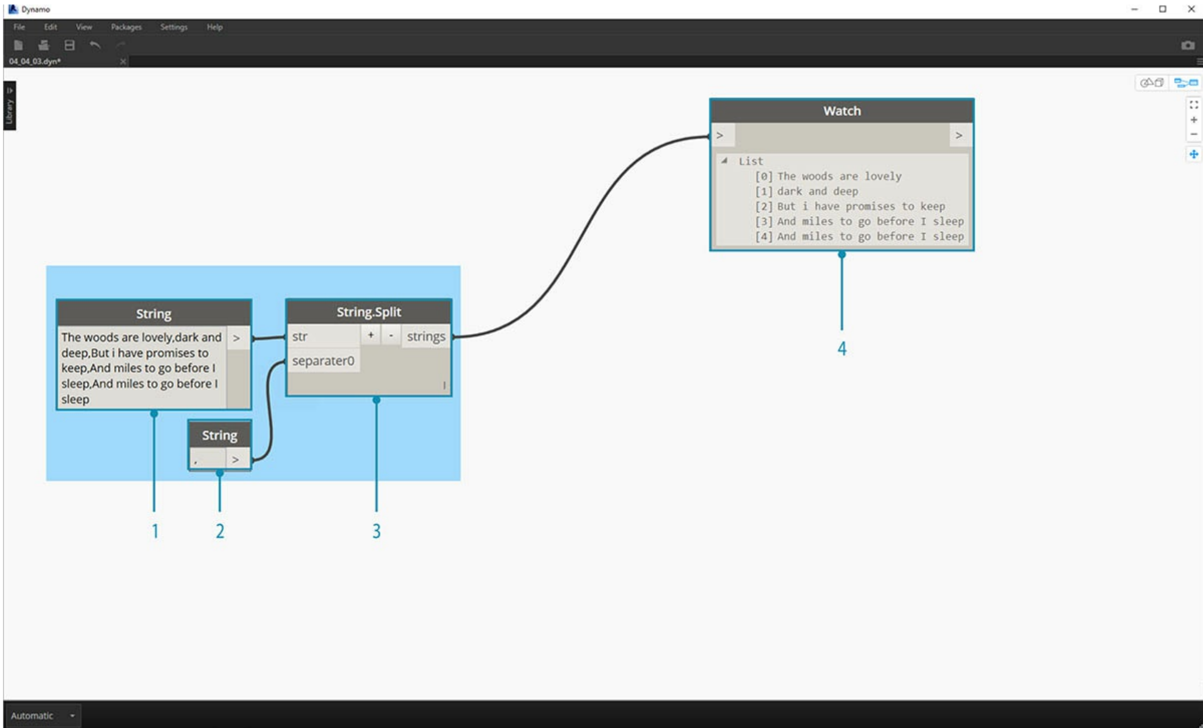
На изображении выше показано соединение двух строк.

1. Входной параметр separator позволяет создать строку, выступающую в качестве разделителя соединенных строк.

Работа со строками

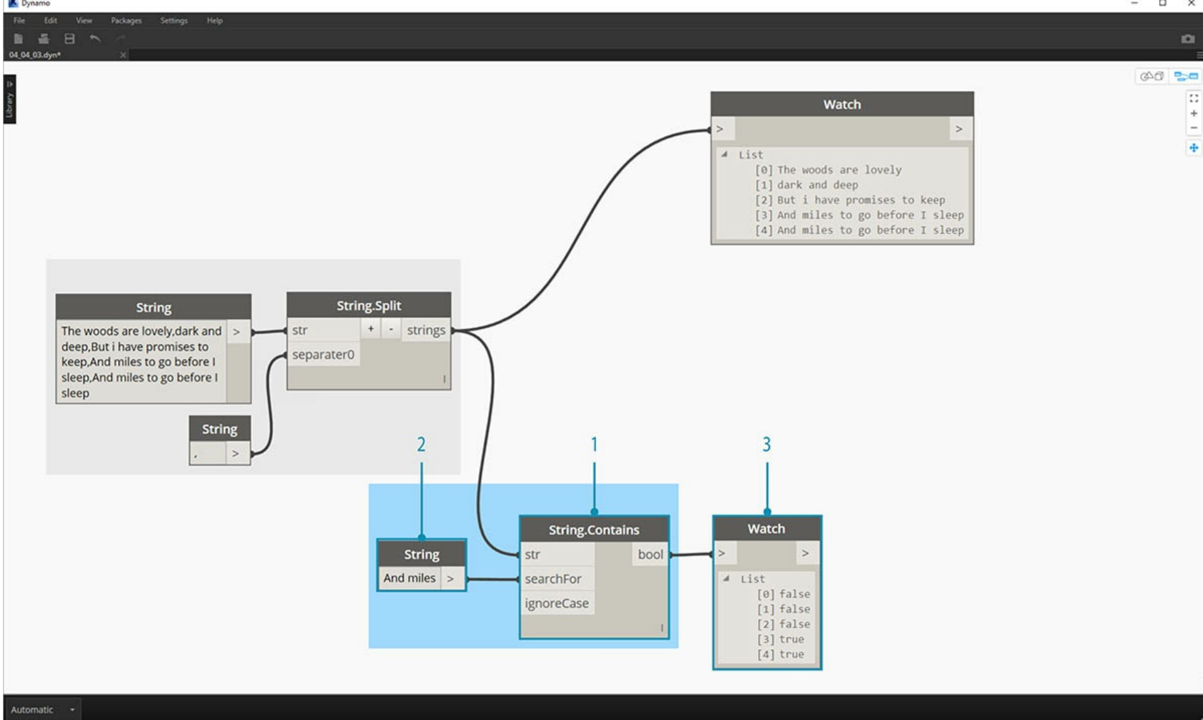
В этом упражнении с помощью методов запроса строк и операций со строками мы разберем последнее четверостишие стихотворения

американского поэта Роберта Фроста Stopping By Woods on a Snowy Evening. Хотя это не самый практичный пример, он поможет понять, как выполнять основные операции со строками в связном тексте с размером и рифмой.



Начнем с разделения строк четверостишия. Прежде всего, обратим внимание на то, что в стихотворении используются запятые. С помощью них можно будет разделить каждую строчку на отдельные элементы.

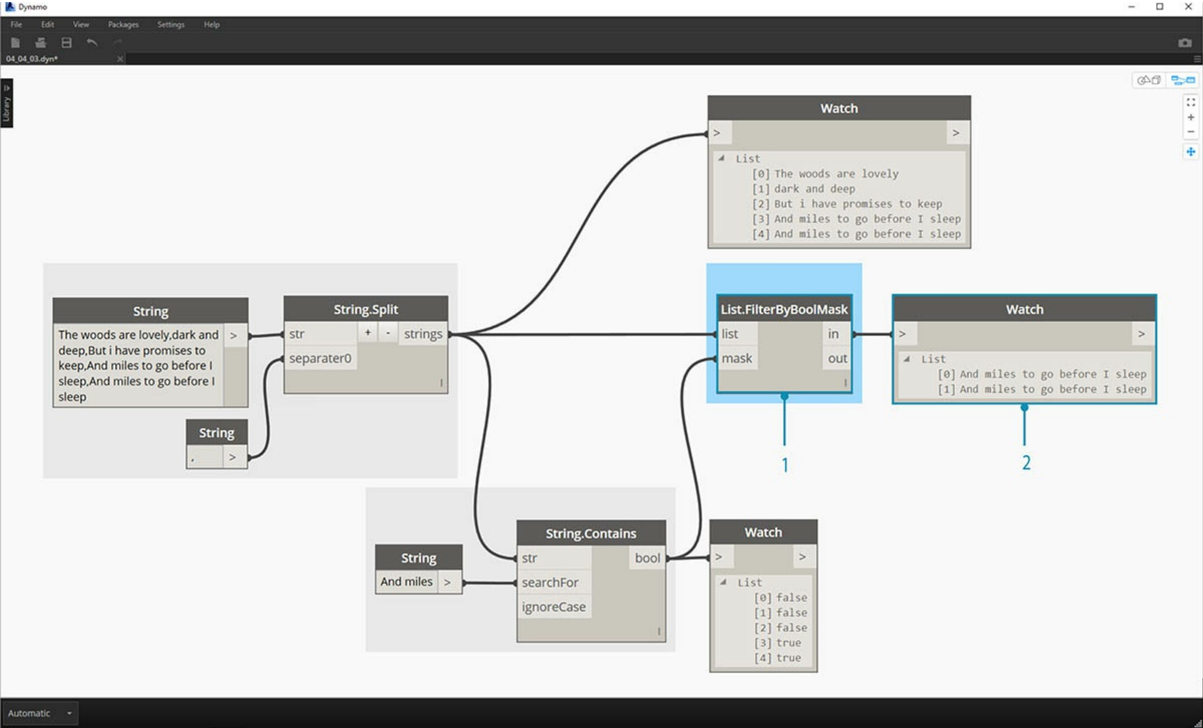
1. Исходная строка вставляется в узел String.
2. Для указания разделителя используется еще один узел String. В данном случае разделителем будет запятая.
3. В рабочую область добавляем узел String.Split, который соединяется с двумя строками.
4. На выходе строчки разделены на отдельные элементы.



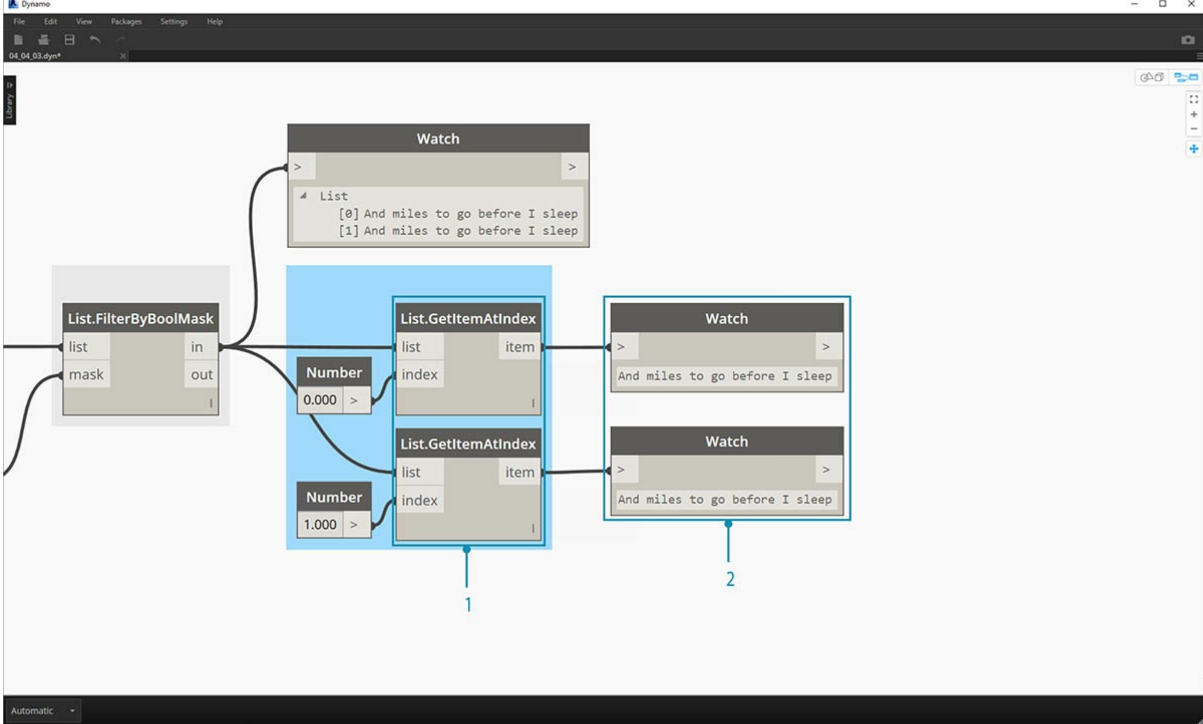
Перейдем к самой интересной части стихотворения: последним двум строчкам. Исходное четверостишие представляло собой один элемент данных. Сперва эти данные были разделены на отдельные элементы. Теперь необходимо найти нужный нам текст. Хотя это *можно* осуществить, выбрав два последних элемента списка, в случае с целой книгой не обязательно прочитывать ее всю и выделять элементы вручную.

1. Вместо того чтобы выполнять поиск вручную, используем узел String.Contains для поиска набора символов. Этот узел аналогичен

команде «Найти» в текстовом редакторе. В этом случае при обнаружении подстроки в элементе возвращается значение «Истина» или «Ложь».
2. Для входного параметра searchFor определим подстроку, которую необходимо найти в четверостишии. Воспользуемся узлом String с текстом And miles.
3. На выходе получим список значений «Истина» и «Ложь». Воспользуемся этой логикой для фильтрации элементов на следующем этапе.

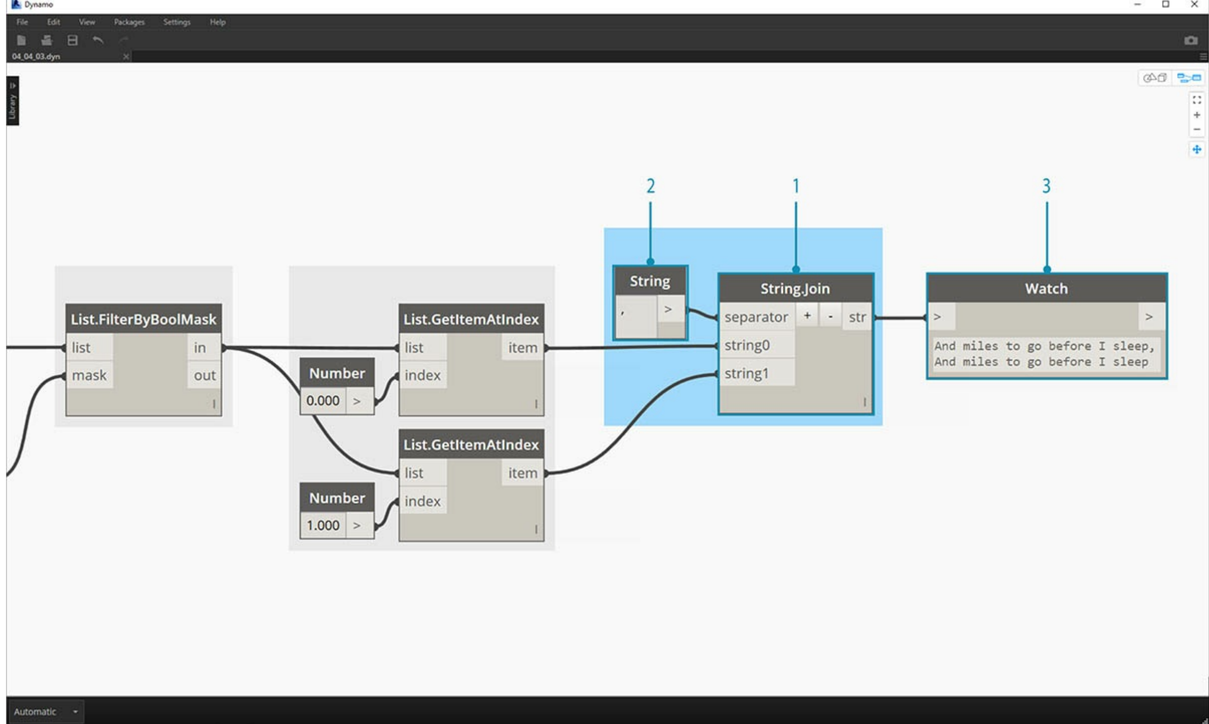


1. Для отбора значений «Истина» и «Ложь» используем узел List.FilterByBoolMask. Выходной параметр in возвращает выражения с входным значением mask, для которых действительно значение «Истина», а выходной параметр out — выражения, для которых действительно значение «Ложь».
2. Результат на выходном параметре in соответствует ожиданиям, то есть мы получаем две последние строки четверостишия.



Теперь необходимо воссоздать повтор, объединив две строки вместе. Если посмотреть на результаты выполнения предыдущего шага, можно заметить, что в списке присутствуют два элемента.

1. Используя два узла List.GetItemAtIndex можно изолировать элементы, используя значения 0 и 1 в качестве входных данных индекса.
2. На выходе из каждого узла получаем последние две строки, расположенные по порядку.



Для объединения этих двух элементов в один используем узел String.Join.

1. После добавления узла String.Join становится понятно, что требуется разделитель.
2. Для создания разделителя добавим узел String в рабочую область и введем запятую.
3. Конечный результат — объединение двух последних элементов в один.

Изолирование двух последних строк может показаться довольно трудоемким процессом. Это действительно так — операции со строками часто требуют предварительной подготовки. Однако эти операции масштабируемы и могут сравнительно легко применяться к большим наборам данных. Если работа с электронными таблицами и настройка совместимости осуществляются на параметрической основе, обязательно помните об операциях со строками.

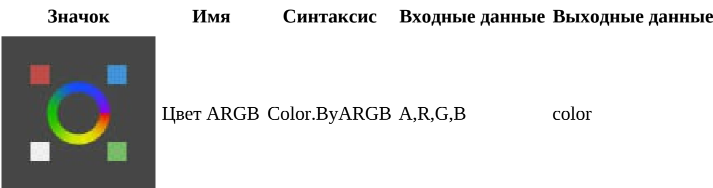
Цвет

Цвет

Цвет — это тип данных, который помогает создавать наглядные визуальные представления, а также отражать различия в результатах визуального программирования. При работе с абстрактными данными и численными переменными бывает трудно определить, что именно изменилось и в какой степени. Для решения этой проблемы можно использовать цвета.

Создание цветов

Цвета в Dynamo создаются с использованием входных данных ARGB, что является сокращенным обозначением комбинации альфа-канала (Alpha) с красным (Red), зеленым (Green) и синим (Blue) каналами. Альфа-канал служит для задания *прозрачности* цвета, а остальные три канала используются как основные цвета для создания всего цветового спектра.

Значок	Имя	Синтаксис	Входные данные	Выходные данные
	Цвет ARGB	Color.ByARGB	A,R,G,B	color

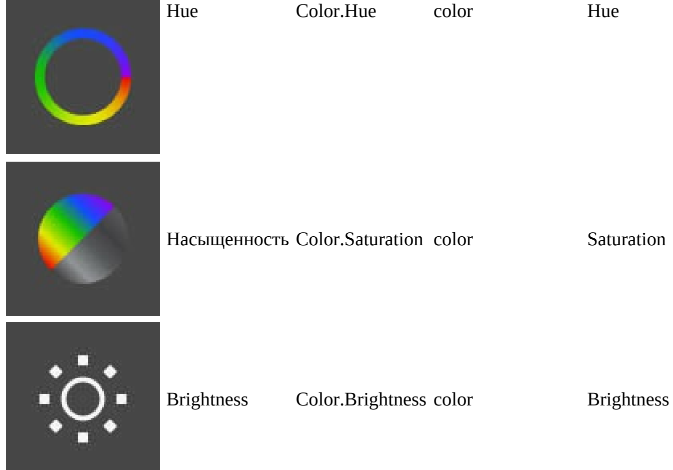
Запрос значений цвета

Цвета в таблице ниже запрашивают свойства, использованные для настройки итогового цвета: альфа, красный цвет, зеленый цвет и синий цвет. Обратите внимание, что узел Color.Components включает все четыре свойства в качестве портов вывода, поэтому его удобнее всего использовать для запроса свойств цвета.

Значок	Имя	Синтаксис	Входные данные	Выходные данные
	Альфа	Color.Alpha	color	A
	Красный цвет	Color.Red	color	R
	Зеленый цвет	Color.Green	color	G
	Синий цвет	Color.Blue	color	B
	Компоненты	Color.Components	color	A,R,G,B

Цвета в таблице ниже соответствуют **цветовому пространству HSB**. Разделение цвета на такие составляющие, как оттенок, насыщенность и яркость, является более понятным и привычным с точки зрения интерпретации цвета. Какого оттенка должен быть цвет? Насколько ярким он должен быть? Насколько светлым или темным? Отвечая на эти вопросы, мы тем самым разбиваем цвет на составляющие, то есть на оттенок, насыщенность и яркость соответственно.

Значок	Имя запроса	Синтаксис	Входные данные	Выходные данные

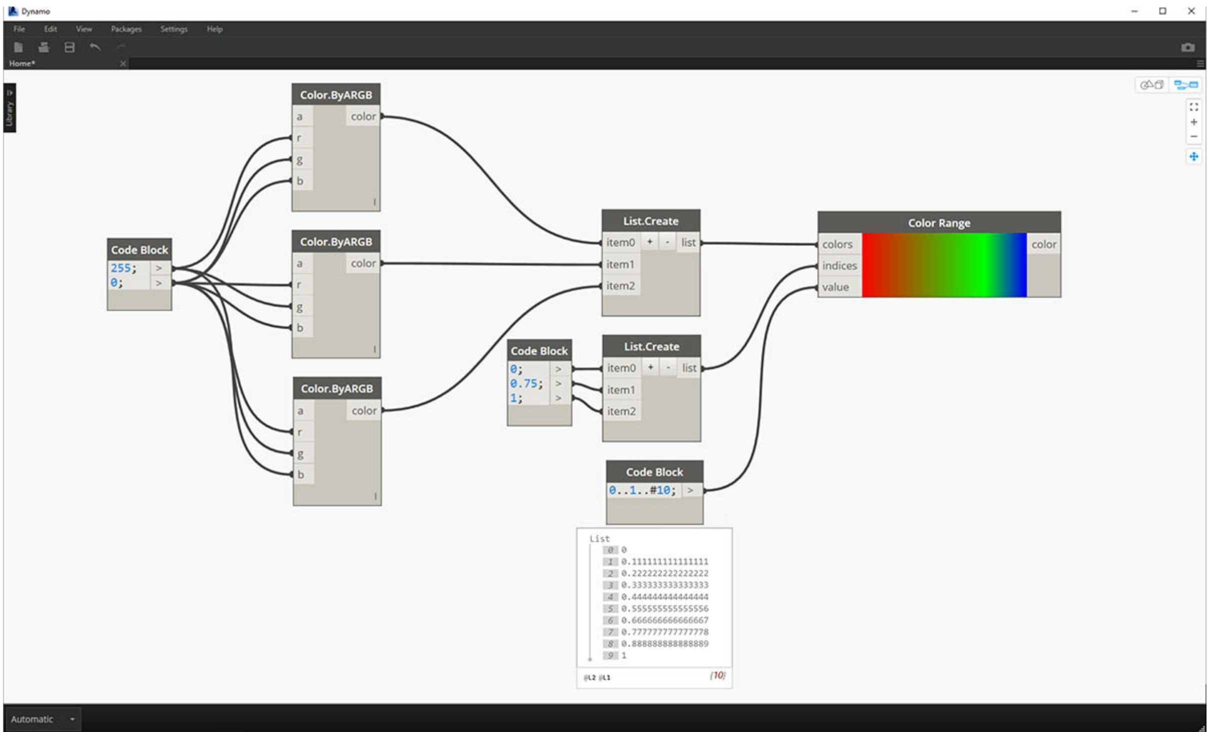


Hue	Color.Hue	color	Hue
Насыщенность	Color.Saturation	color	Saturation
Brightness	Color.Brightness	color	Brightness

Цветовой диапазон

Цветовой диапазон аналогичен узлу **Remap Range** из раздела 4.2: и тот и другой сопоставляет список числовых значений со значениями из другой области. Однако вместо сопоставления с областью *чисел* цветовой диапазон выполняет сопоставление с *цветовым градиентом* в соответствии со входными значениями в диапазоне от 0 до 1.

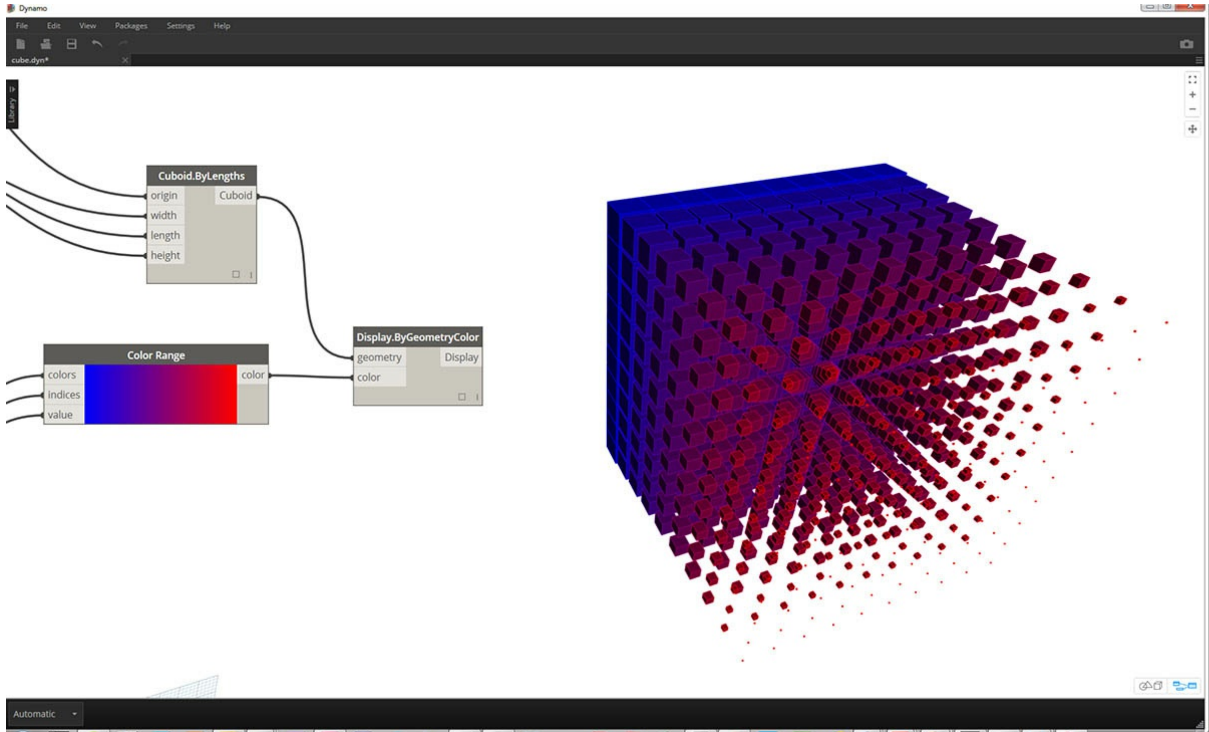
Текущий узел хорошо выполняет свои функции, но с ним может быть трудно добиться нужных результатов с первого раза. Чтобы уверенно работать с цветовым градиентом, стоит несколько раз опробовать его на практике в интерактивном режиме. Выполните небольшое упражнение, чтобы узнать, как настроить градиент, так чтобы цвета на выходе соответствовали заданным числам.



1. **Определите три цвета.** С помощью узла Code Block определите *красный, зеленый* и *синий* цвета, назначив каждому из них соответствующие сочетания значений *0* и *255*.
2. **Создайте список.** Объедините три цвета в один список.
3. **Определите индексы.** Создайте список для определения положения ручек каждого цвета (в диапазоне от 0 до 1). Обратите внимание, что для зеленого цвета задано значение 0,75. Это смещает зеленый цвет на 3/4 вдоль горизонтального градиента в регуляторе цветового диапазона.
4. **Code Block.** Введите значения (от 0 до 1), которые будут преобразованы в цвета.

Образец цвета

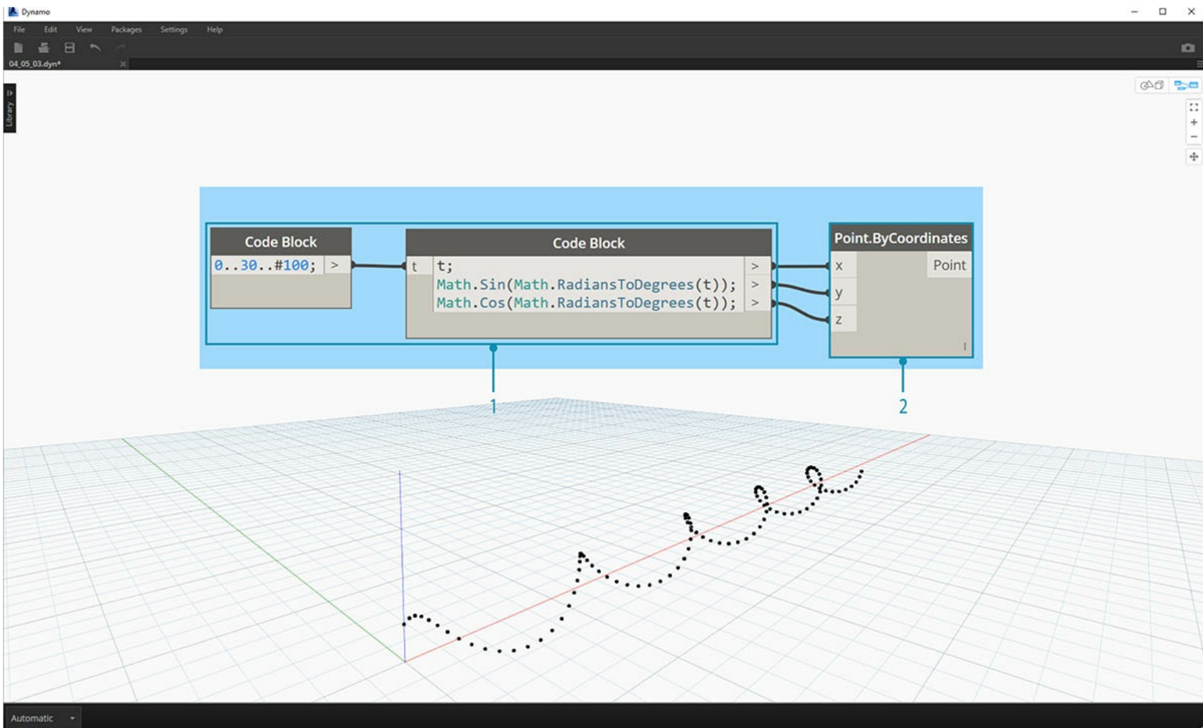
Узел **Display.ByGeometry** позволяет раскрашивать геометрию на видовом экране Dynamo. Это позволяет наглядно показывать различные типы геометрии, демонстрировать параметрические концепции и задавать условные обозначения для расчета при моделировании. В качестве входных данных здесь требуются только геометрия и цвет. Для создания градиента, как на изображении выше, порт ввода color соединяется с узлом **Color Range**.



Упражнение по работе с цветом

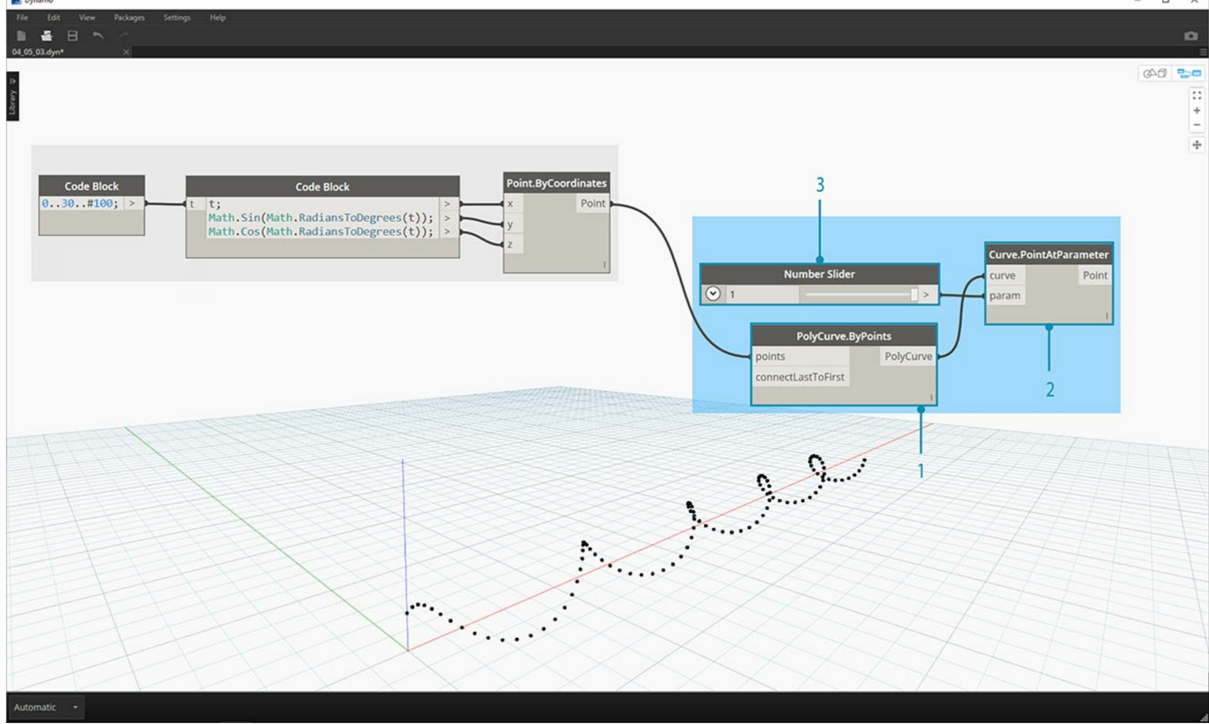
> Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): [Building Blocks of Programs - Color.dyn](). Полный список файлов примеров можно найти в приложении.

В этом упражнении основное внимание уделяется параметрическому управлению цветом параллельно с геометрией. Геометрия — стандартная спираль, определение которой выполнено ниже с помощью узла **Code Block** (3.2.3). Это простой и быстрый способ создания параметрической функции. Так как в данном уроке рассматривается работа с цветом (а не с геометрией), то Code Block идеально подходит для быстрого создания спирали без загромождения рабочей области. Мы будем использовать Code Block все чаще и чаще по мере изучения более сложных процессов.



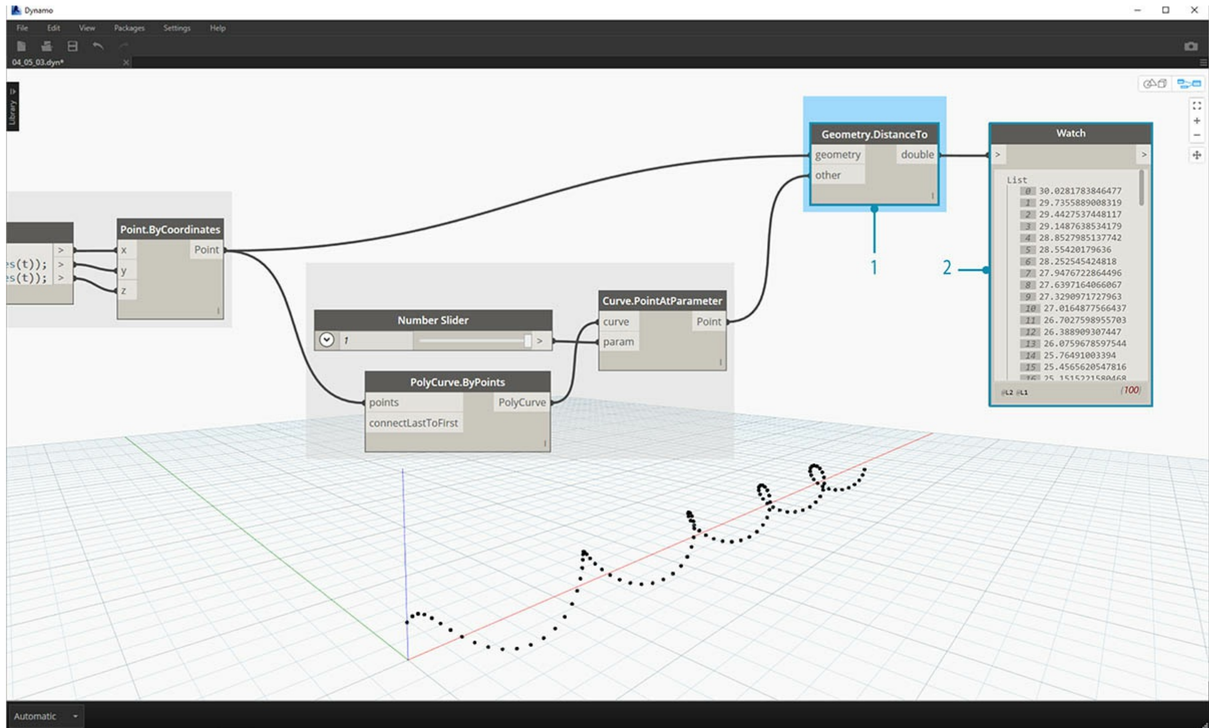
1. **Code Block:** создайте два узла Code Block, используя приведенные выше формулы. Это быстрый параметрический метод создания спирали.
2. **Point.ByCoordinates:** соедините порты координат этого узла с тремя портами вывода узла Code Block.

Отображается массив точек, которые образуют спираль. Далее необходимо создать кривую, проходящую через точки, чтобы получить изображение спирали.



1. **PolyCurve.ByPoints:** соедините порт вывода *Point.ByCoordinates* с портом ввода *points* этого узла. Отображается спиральная кривая.
2. **Curve.PointAtParameter:** соедините порт вывода *PolyCurve.ByPoints* с портом ввода *curve*. Это требуется, чтобы создать параметрическую точку притяжения, которая перемещается вдоль кривой. Так как кривая вычисляет положение точки с помощью параметра, необходимо задать значение *param* в диапазоне от 0 до 1.
3. **Number Slider:** добавьте этот узел в рабочую область и измените значение *min* на *0,0*, значение *max* на *1*, а значение *step* на *0,01*. Соедините порт вывода регулятора с портом ввода *param* узла *Curve.PointAtParameter*. Появляется точка, которая перемещается вдоль спирали в соответствии с положением регулятора (0 — в начальной точке, 1 — в конечной).

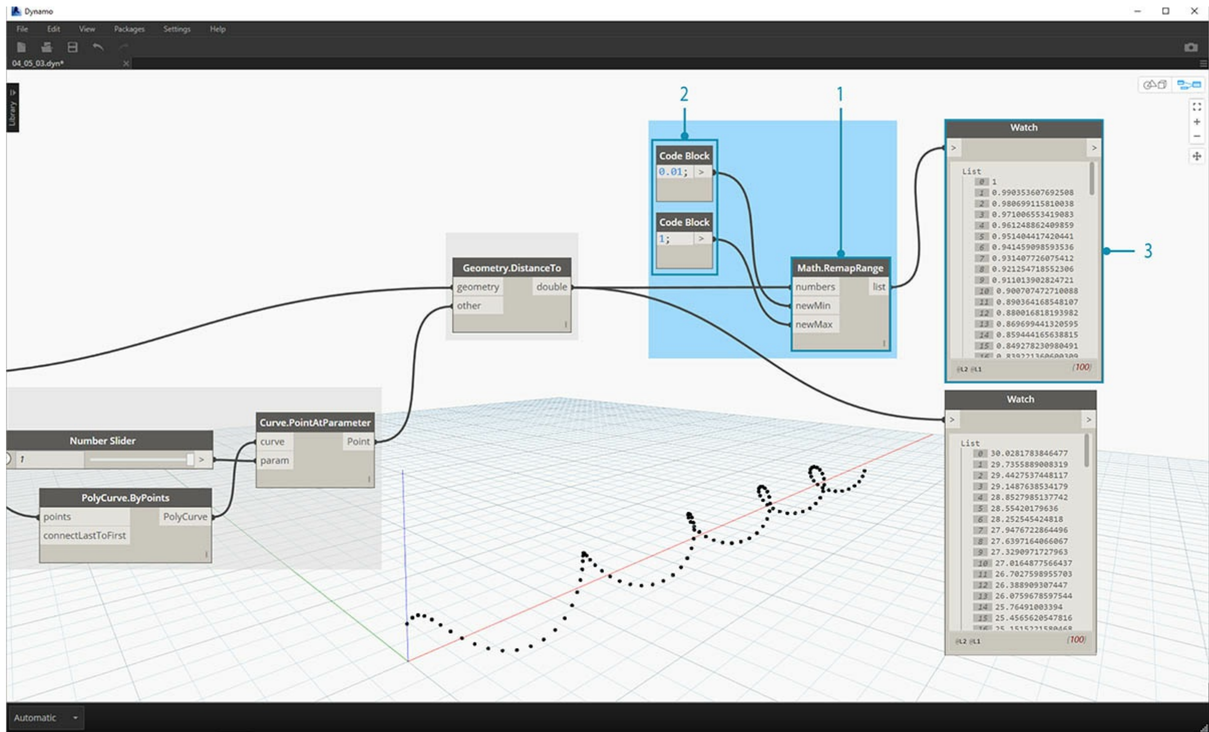
Создав опорную точку, мы можем сравнить расстояние от нее до исходных точек, определяющих геометрию спирали. Данное расстояние будет определять геометрию и цвет.



1. **Geometry.DistanceTo:** соедините порт вывода узла *Curve.PointAtParameter* с портом *ввода* этого узла. Соедините узел *Point.ByCoordinates* с портом ввода geometry.
2. **Watch:** в результате мы получаем список значений расстояния от каждой точки спирали до опорной точки, перемещающейся вдоль кривой.

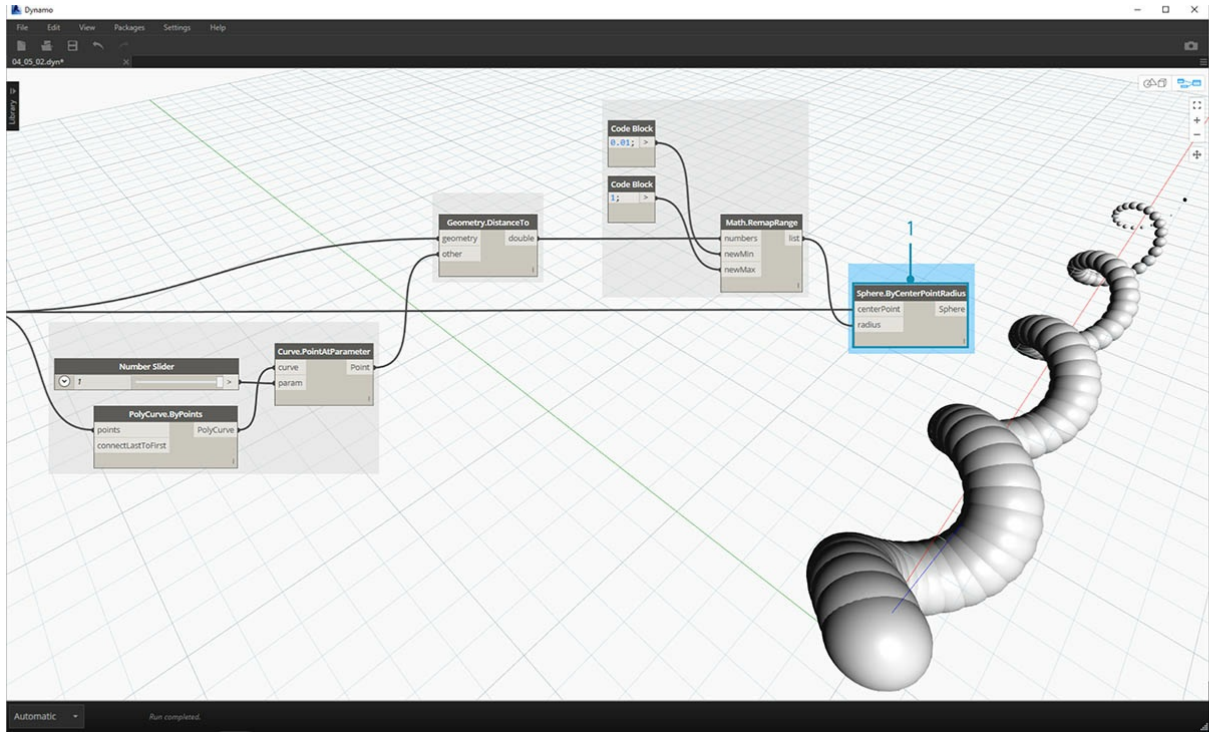
Следующий шаг — определение параметров на основе списка расстояний между точками спирали и опорной точкой. Эти значения расстояний

будут использованы для определения радиусов сфер, размещаемых вдоль кривой. Чтобы обеспечить подходящий размер сфер, необходимо *повторно сопоставить* значения расстояния.

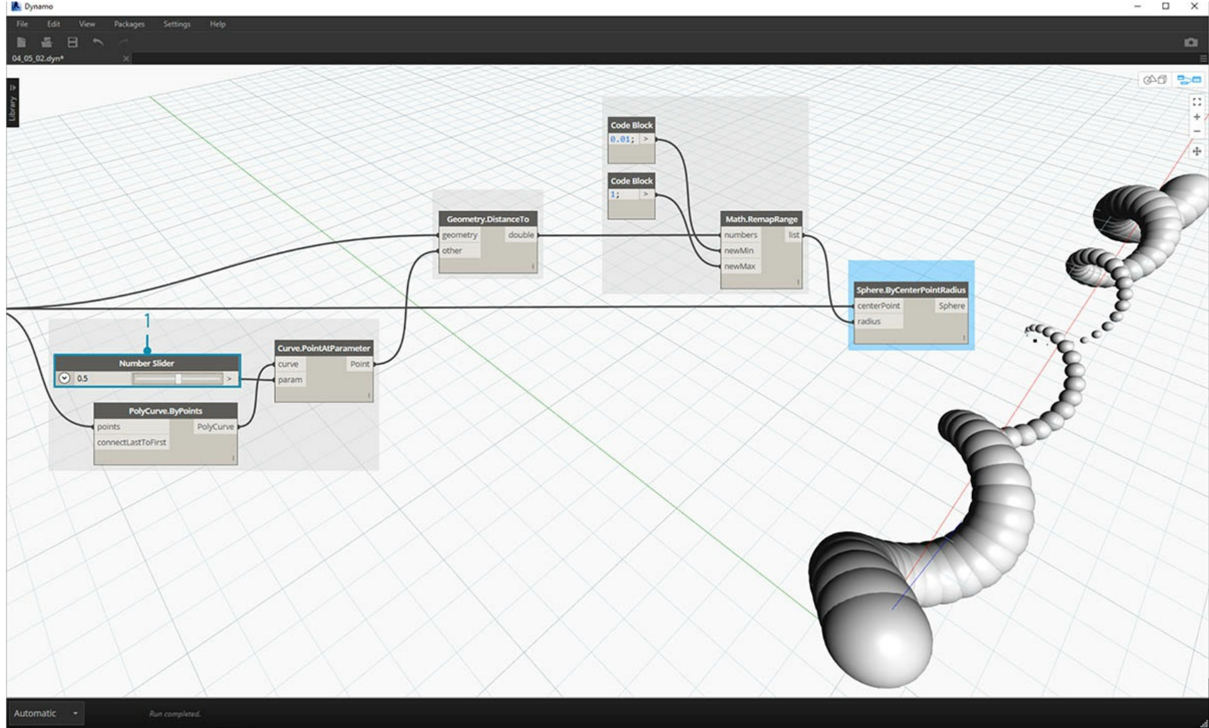


1. **Math.RemapRange:** соедините порт вывода узла *Geometry.DistanceTo* с портом ввода numbers.
2. **Code Block**: соедините узел Code Block со значением *0,01* с портом ввода *newMin*, а узел Code Block со значением *1* с портом ввода *newMax*.
3. **Watch:** соедините порт вывода *Math.RemapRange* с одним узлом Watch, а порт вывода *Geometry.DistanceTo* — с другим. Сравните результаты.

Выполнив этого шаг, мы получили повторно сопоставленный список расстояний меньшего диапазона. При необходимости можно задать другие значения *newMin* и *newMax*. Новые значения будут повторно сопоставлены и будут иметь одинаковый *коэффициент распределения* в пределах области.

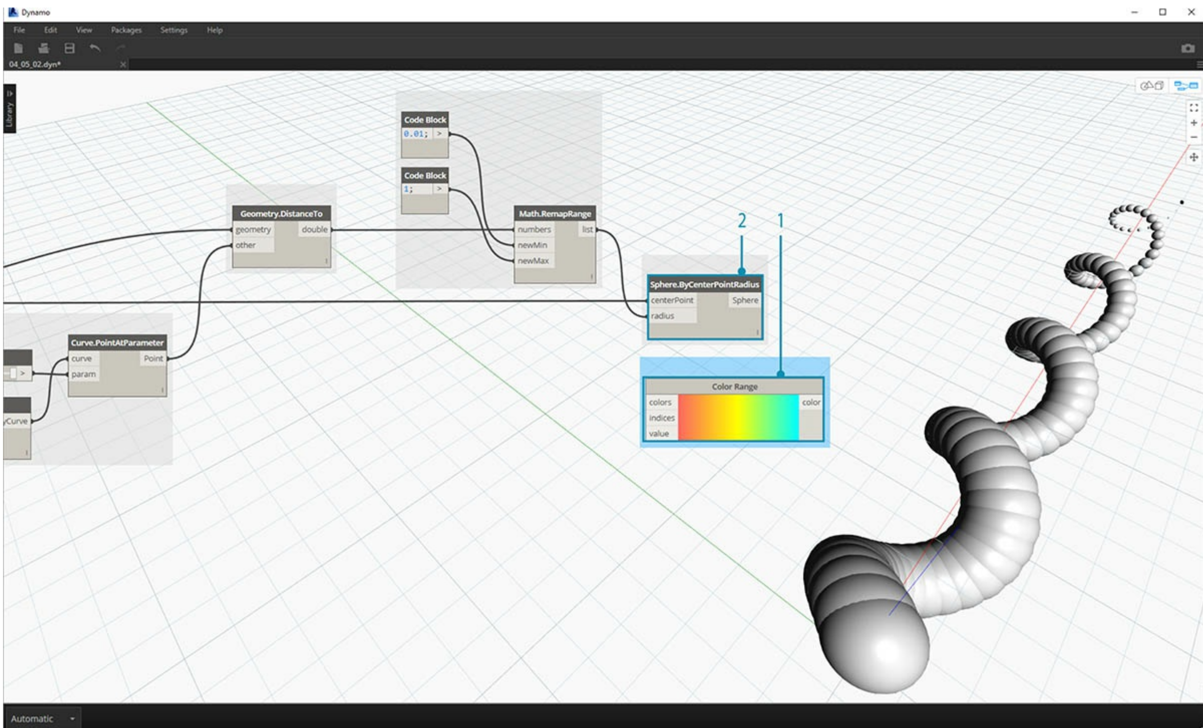


1. **Sphere.ByCenterPointRadius:** соедините порт вывода узла *Math.RemapRange* с портом ввода *radius*, а порт вывода исходного узла *Point.ByCoordinates* — с портом ввода *centerPoint*.

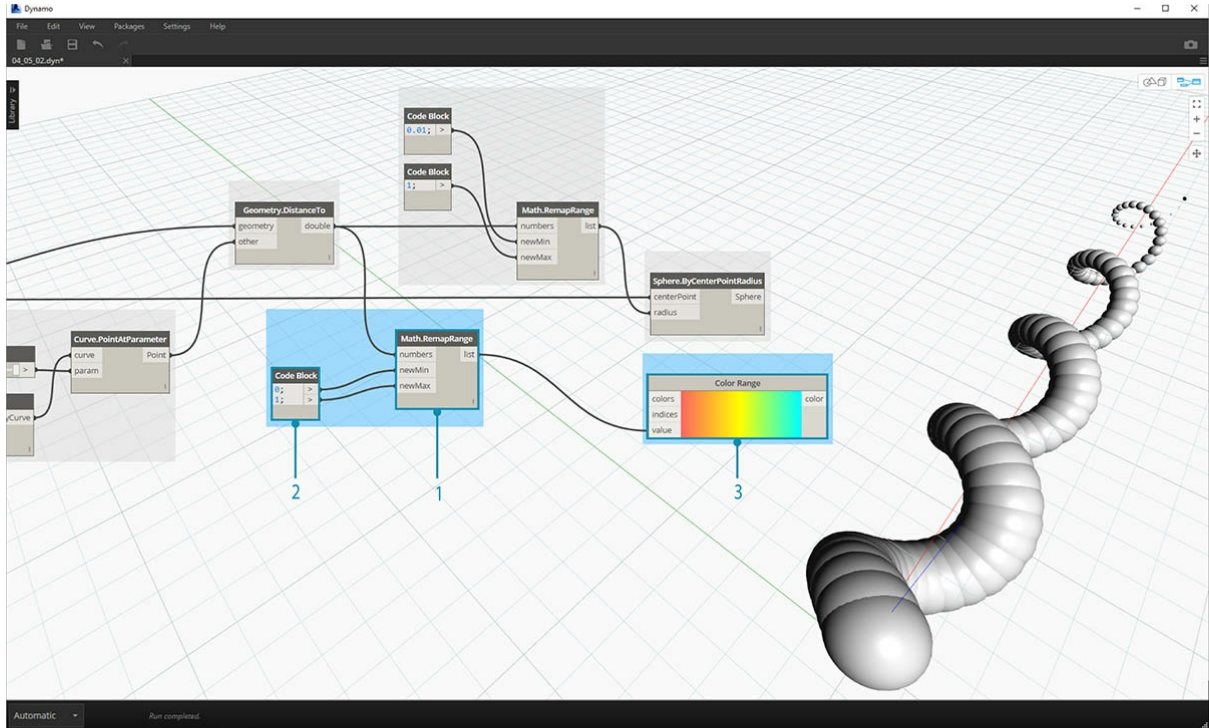


1. **Number Slider:** измените значение числового регулятора и посмотрите, как при этом изменится размер сфер. Теперь у нас есть параметрический шаблон.

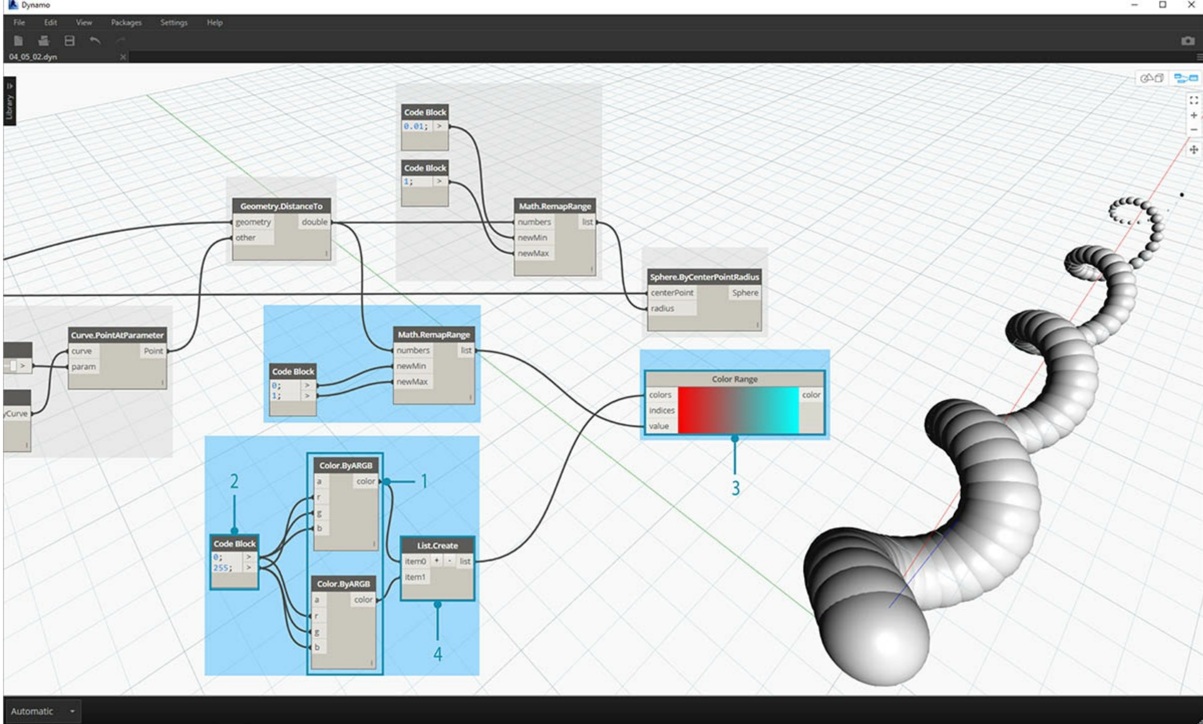
Размер сфер является представлением параметрического массива, определяемого опорной точкой, перемещающейся вдоль кривой. Применим эту же концепцию к радиусу сфер, чтобы определить их цвет.



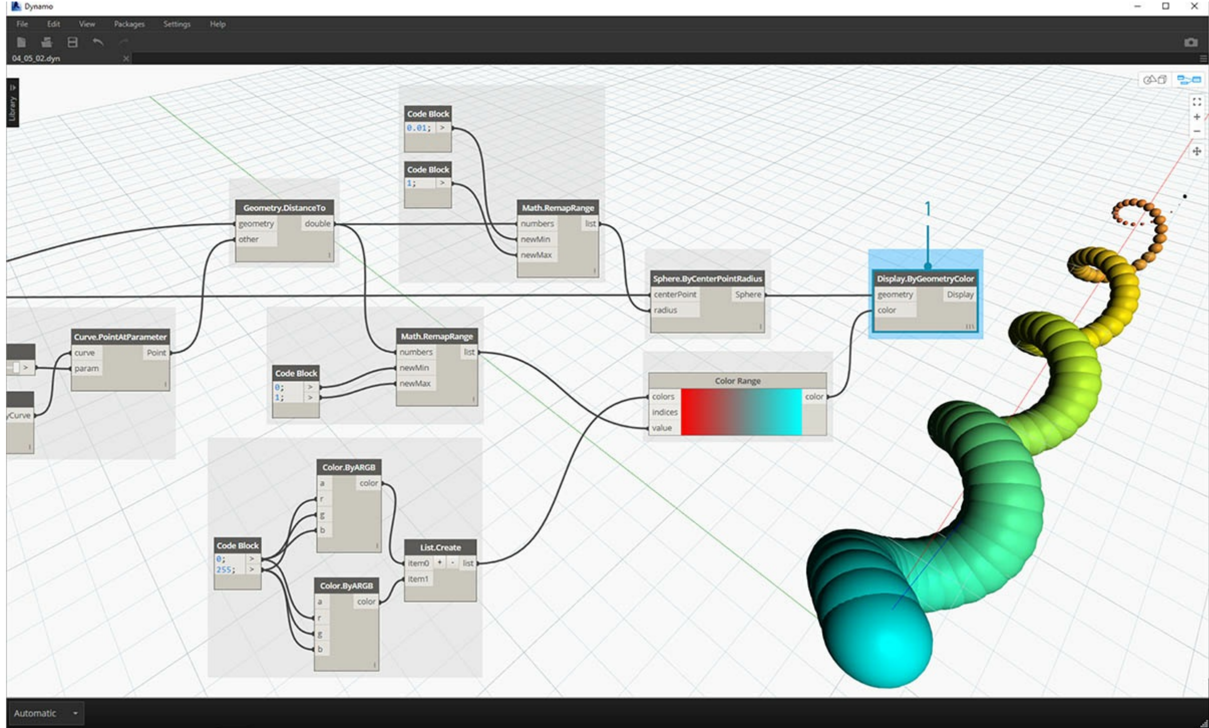
1. **Color Range:** добавьте в рабочую область этот узел. При наведении указателя мыши на порт ввода *value* обратите внимание, что запрашиваемые числа находятся в диапазоне от 0 до 1. Необходимо повторно сопоставить числа, указанные для порта вывода узла *Geometry.DistanceTo*, чтобы они были совместимы с этой областью.
2. **Sphere.ByCenterPointRadius:** временно отключите предварительный просмотр узла (*щелкните правой кнопкой мыши, а затем выберите «Предварительный просмотр»*).



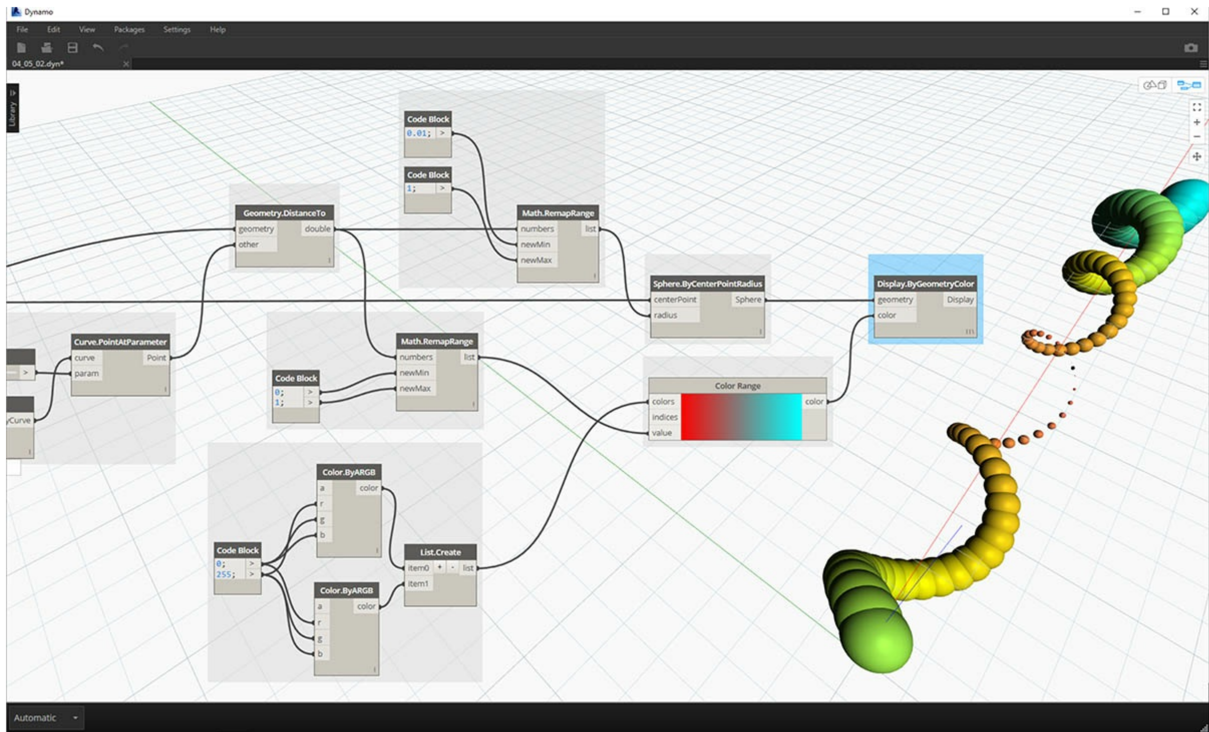
1. **Math.RemapRange:** эта процедура должна быть вам уже знакома. Соедините порт вывода узла *Geometry.DistanceTo* с портом ввода numbers.
2. **Code Block:** аналогично шагу выше задайте значение *0* для порта ввода *newMin* и значение *1* для порта ввода *newMax*. Обратите внимание, что в данном случае мы задаем два порта вывода для одного узла Code Block.
3. **Color Range:** соедините порт вывода узла *Math.RemapRange* с портом ввода *value*.



1. **Color.ByARGB**: этот блок позволит нам создать два цвета. Хотя процесс может показаться не самым очевидным, по сути, это то же самое, что и работа с цветами RGB в другом программном обеспечении, просто здесь мы используем возможности визуального программирования.
2. **Code Block:** создайте два значения: *0* и *255*. Соедините два порта вывода с двумя портами ввода *Color.ByARGB* в соответствии с изображением выше (или создайте другие цвета на ваш выбор).
3. **Color Range:** порт ввода *colors* запрашивает список цветов. Необходимо создать этот список из двух цветов, заданных на предыдущем шаге.
4. **List.Create:**: объедините два цвета в один список. Соедините порт вывода этого узла с портом ввода *colors* узла *Color Range*.



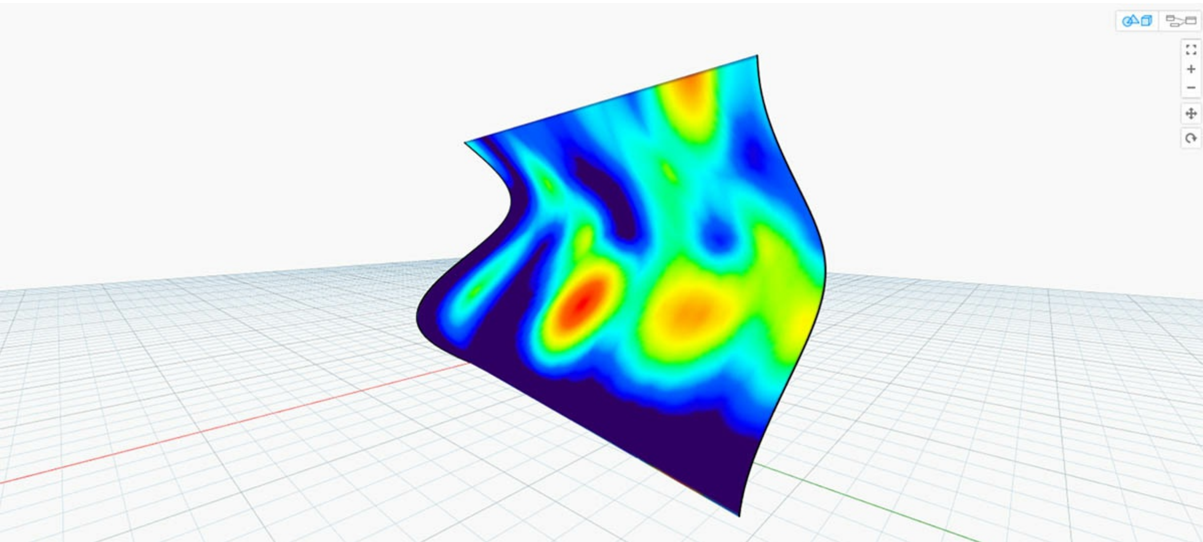
1. **Display.ByGeometryColor:** соедините узел *Sphere.ByCenterPointRadius* с портом ввода *geometry*, а узел *Color Range* — с портом ввода *color*. К области кривой применяется мягкий цветовой градиент.



Если изменить в определении значение узла *Number Slider*, который мы изучили ранее, то цвета и размеры геометрии будут обновлены. Цвета и размер радиуса в данном случае связаны напрямую, и теперь между этими двумя параметрами существует и визуальная связь.

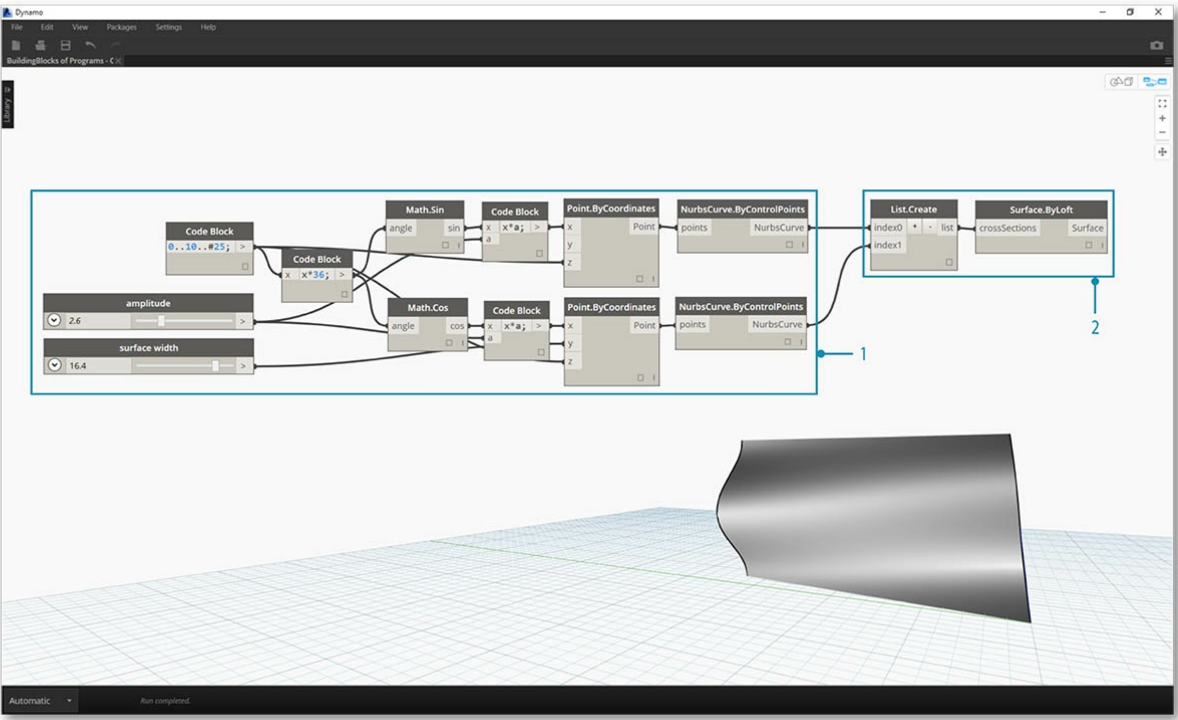
Цвет на поверхностях

Узел **Display.BySurfaceColors** позволяет использовать цвет для сопоставления данных на поверхности. Это дает нам широкие возможности для визуализации данных, полученных с помощью таких типов дискретного анализа, как расчеты инсоляции и энергопотребления, а также анализ близости. Применение цвета к поверхности в Dynamo аналогично применению текстуры к материалу в других средах САПР. Выполните небольшое упражнение ниже, чтобы ознакомиться с этим инструментом.



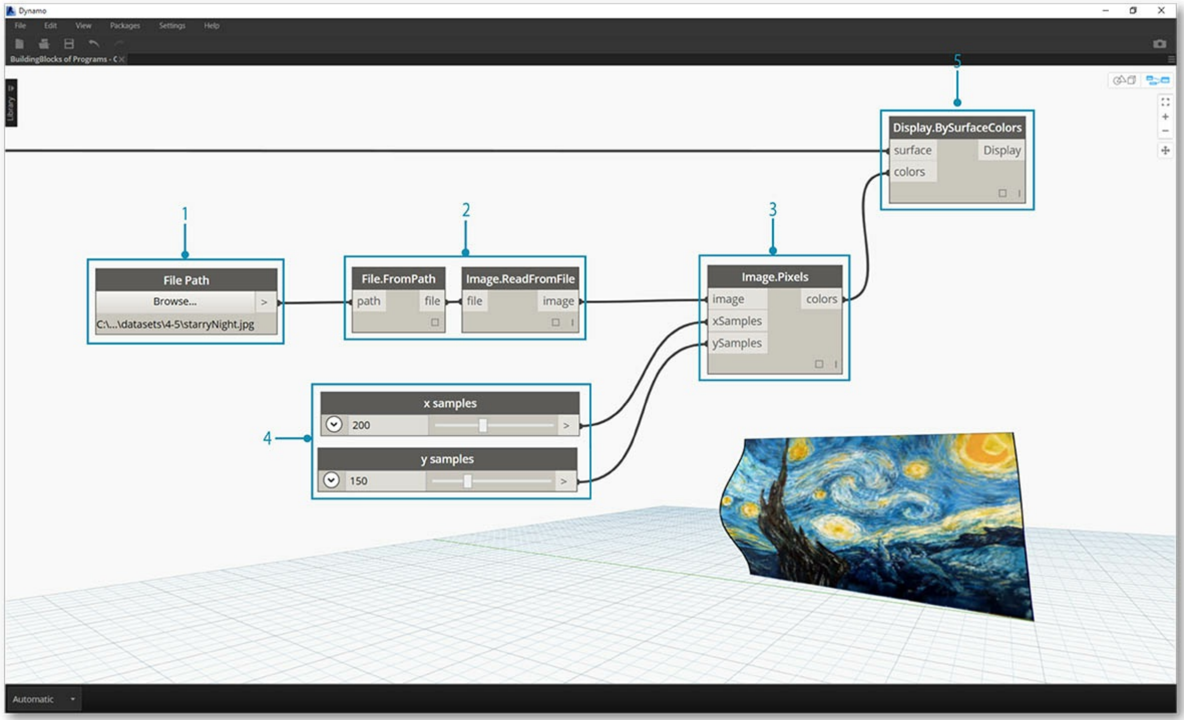
Упражнение по работе с цветом на поверхностях

Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): [Building Blocks of Programs - ColorOnSurface.zip](). Полный список файлов примеров можно найти в приложении.

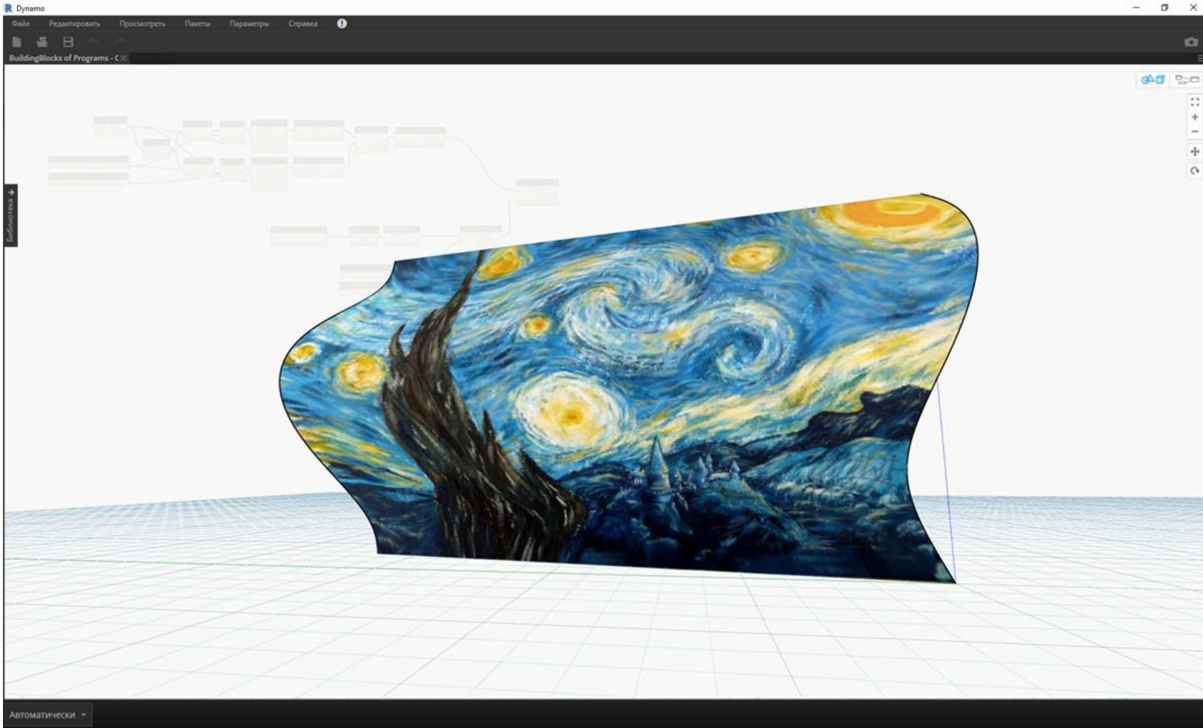


Сначала необходимо создать поверхность (или выбрать существующую поверхность в качестве опорной) для использования в качестве входных данных узла **Display.BySurfaceColors**. В этом примере поверхность образуется путем лофтинга между синусоидой и косинусоидой.

1. Эта **группа** узлов выполняет создание точек вдоль оси Z с последующим смещением в соответствии с функциями синуса и косинуса. Два полученных списка точек затем используются для создания NURBS-кривых.
2. **Surface.ByLoft**: сформируйте интерполированную поверхность между NURBS-кривыми из списка.



1. **File Path**: выберите файл изображения, который будет использоваться в качестве образца для пиксельных данных на последующих этапах.
2. Используйте узел **File.FromPath**, чтобы преобразовать путь к файлу в файл, а затем передайте этот файл в узел **Image.ReadFromFile**, чтобы вывести изображение для использования в качестве образца.
3. **Image.Pixels**: используйте изображение в качестве входных данных и введите значение количества образцов, получаемых вдоль осей X и Y изображения.
4. **Регуляторы**: задайте значения количества образцов для узла **Image.Pixels**.
5. **Display.BySurfaceColors**: сопоставьте массив значений цветов на поверхности со значениями по осям X и Y соответственно.

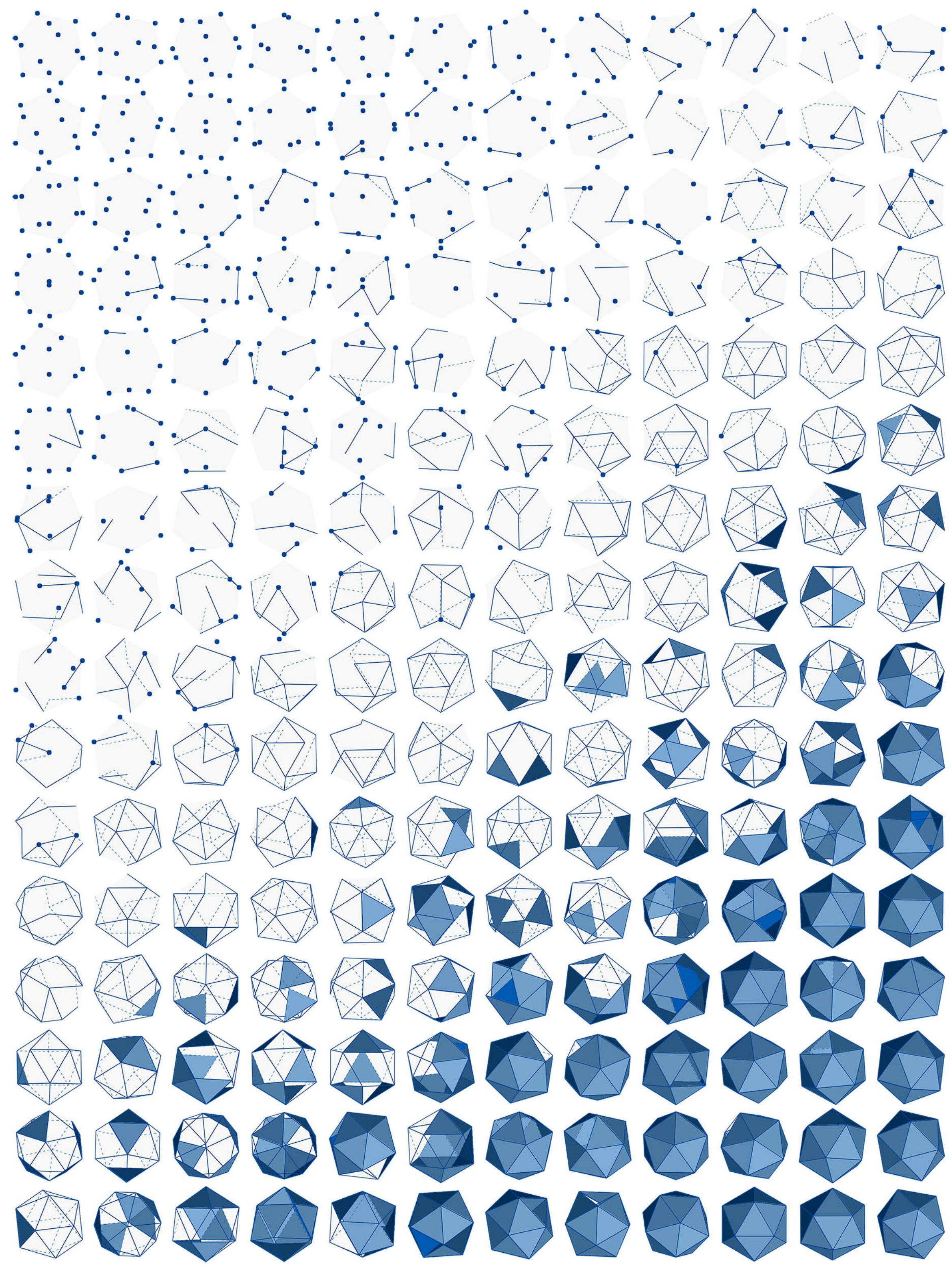


Приближенная предварительная версия итоговой поверхности с разрешением 400 x 300.

Геометрия для машинного проектирования

ГЕОМЕТРИЯ ДЛЯ МАШИННОГО ПРОЕКТИРОВАНИЯ

Dynamo — это среда визуального программирования, которая позволяет пользователям создавать процессы обработки данных. Данные — это числа или текст. Но это также и геометрия. С точки зрения компьютера, геометрия (иногда ее также называют вычислительной геометрией) — это данные, которые можно использовать для создания эстетичных, детализированных или высокопроизводительных моделей. Для этого нужно сначала внимательно изучить различные типы геометрических объектов, доступных для использования.



Обзор концепции геометрии

Обзор концепции геометрии

Геометрия — это язык, на котором осуществляется разработка. Если в основе языка или среды программирования лежит геометрия, это открывает широкие возможности для создания точных и надежных моделей, автоматизации процессов разработки и итерации проектов на основе алгоритмов.

Основы

В обычном понимании геометрия — это исследование формы, размера, относительного положения фигур и свойств пространства. Эта дисциплина имеет богатую историю, берущую свое начало тысячи лет назад. Благодаря появлению компьютеров мы получили мощный инструмент для описания, изучения и генерирования геометрических объектов. В настоящее время можно с легкостью рассчитать сложнейшие геометрические взаимосвязи, и ни для кого не секрет, что мы активно используем эти возможности.

> Если вы хотите узнать, насколько разнообразной и сложной может быть геометрия, разрабатываемая с помощью компьютеров, введите словосочетание «стенфордский кролик» в любой поисковой системе. Это каноническая модель, которая используется для тестирования алгоритмов.

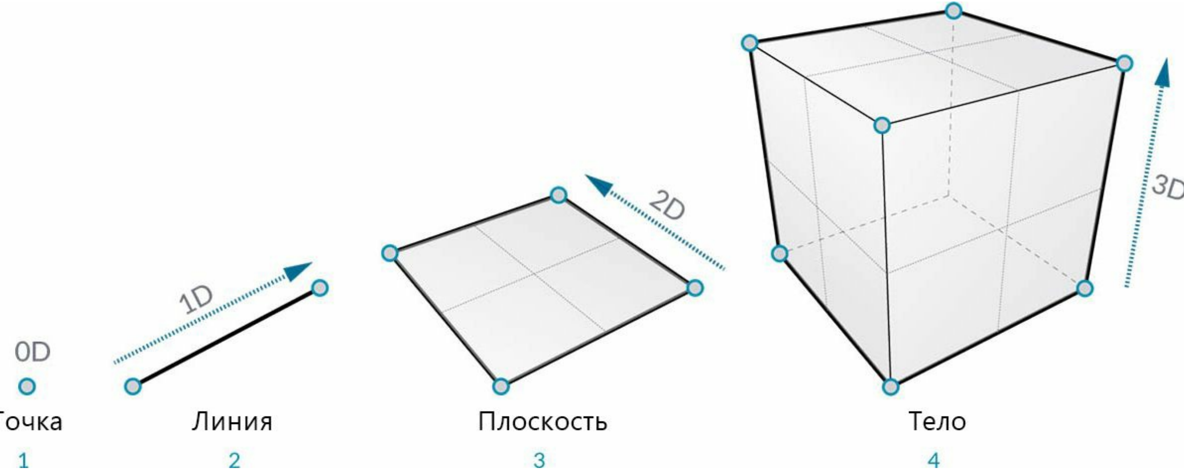
Перспектива применения геометрии в контексте алгоритмов, компьютерных вычислений и повышенной сложности может показаться пугающей. Однако есть несколько относительно простых ключевых принципов, освоив которые, мы сможем приступить к изучению более сложных вариантов применения геометрии.

1. Геометрия —это **данные**, поэтому с точки зрения компьютера и приложения Dynamo геометрический кролик практически ничем не отличается от обычного числа.
2. Геометрия основана на **абстракции**: по сути, все геометрические элементы определяются с помощью чисел, отношений и формул в заданной пространственной системе координат.
3. Геометрия имеет **иерархию**: точки образуют линии, линии образуют поверхности и т. д.
4. Геометрия одновременно описывает **часть и целое**: если есть кривая, то она представляет собой и форму, и все возможные точки вдоль нее.

На практике эти принципы означают, что пользователи должны понимать, над чем они работают (тип геометрии, как она была создана и т. д.). Это понимание позволит нам с легкостью конструировать, разбирать и снова собирать различные геометрические объекты в ходе разработки сложных моделей.

Перемещение между уровнями иерархии

Давайте рассмотрим геометрию с точки зрения принципов абстракции и иерархии. Хоть это и не всегда очевидно, эти принципы тесно взаимосвязаны, и если в этом не разобраться, то при разработке детализированных рабочих процессов и моделей можно столкнуться с серьезным препятствием. Для начала давайте воспользуемся понятием пространственным измерений в качестве основной характеристики моделируемых нами объектов. Зная количество измерений, необходимых для описания формы, мы сможем приблизиться к пониманию иерархического устройства геометрии.



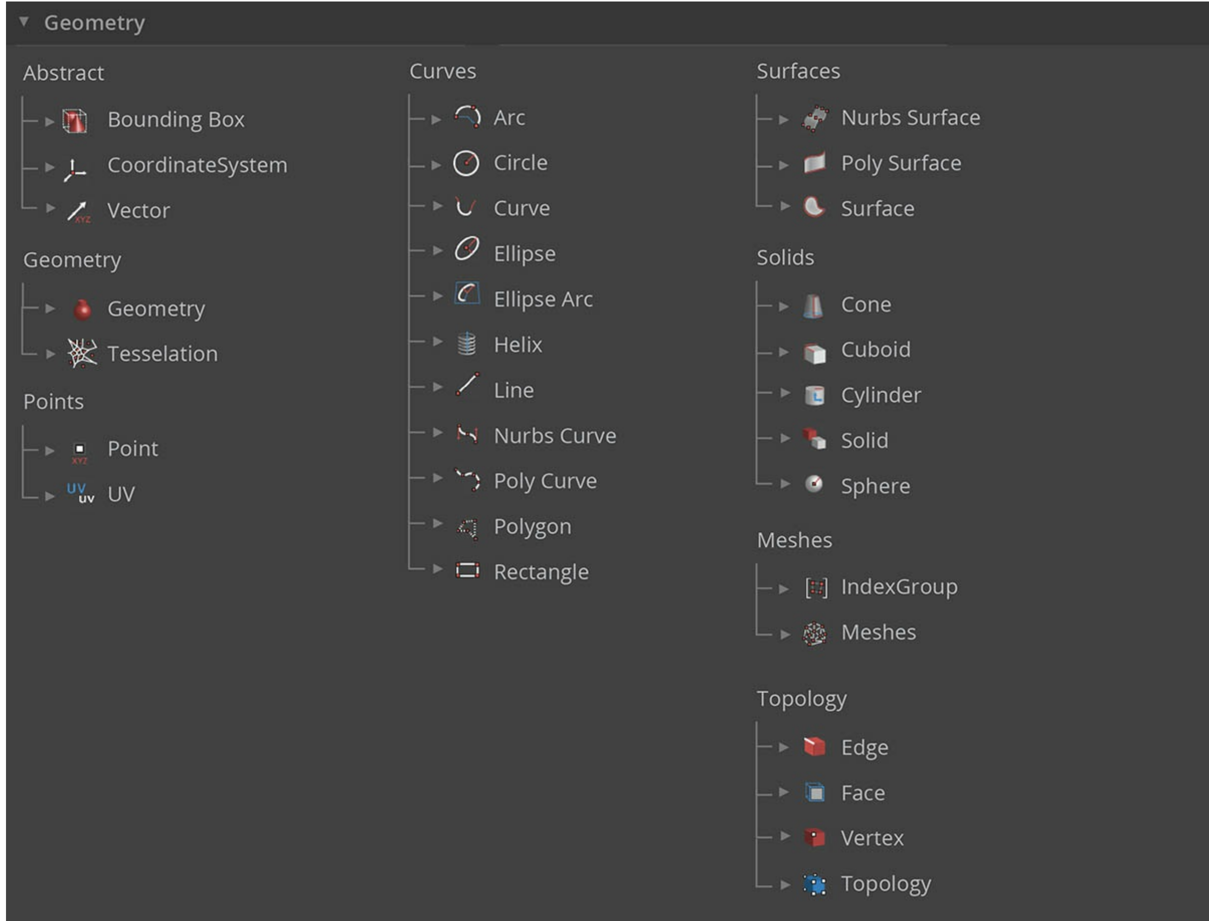
1. **Точка** (определяемая координатами) не имеет измерений. Она представлена только числами, соответствующими каждой из координат.
2. **Отрезок** (определяемый двумя точками), в свою очередь, имеет *одно* измерение: мы можем перемещаться вдоль отрезка вперед (положительное направление) и назад (отрицательное направление).
3. **Плоскость** (определяемая двумя линиями) имеет *два* измерения: мы можем перемещаться не только вперед и назад, но и влево или вправо.
4. **Параллелепипед** (определяемый двумя плоскостями) имеет *три* измерения: в дополнение к указанному выше, мы можем перемещаться вверх и вниз.

Пространственные измерения — это удобный способ классификации геометрический объектов, но не всегда самый лучший. В конце концов, при моделировании используются не только точки, отрезки, плоскости и параллелепипеды. Что если нужен изогнутый объект? Кроме этого, существует другая категория типов геометрических объектов, которые являются полностью абстрактными: они определяют такие свойства, как ориентация, объем или связи между отдельными частями объекта. Такой объект, как вектор, например, абсолютно неосязаем, так как же описать его относительно того, что мы видим в пространстве? Нужна более подробная классификация иерархии геометрических объектов, которая должна отражать разницу между абстрактными типами (вспомогательными средствами), которые можно сгруппировать по выполняемой ими функции, и типами, которые используются для описания формы элементов модели.

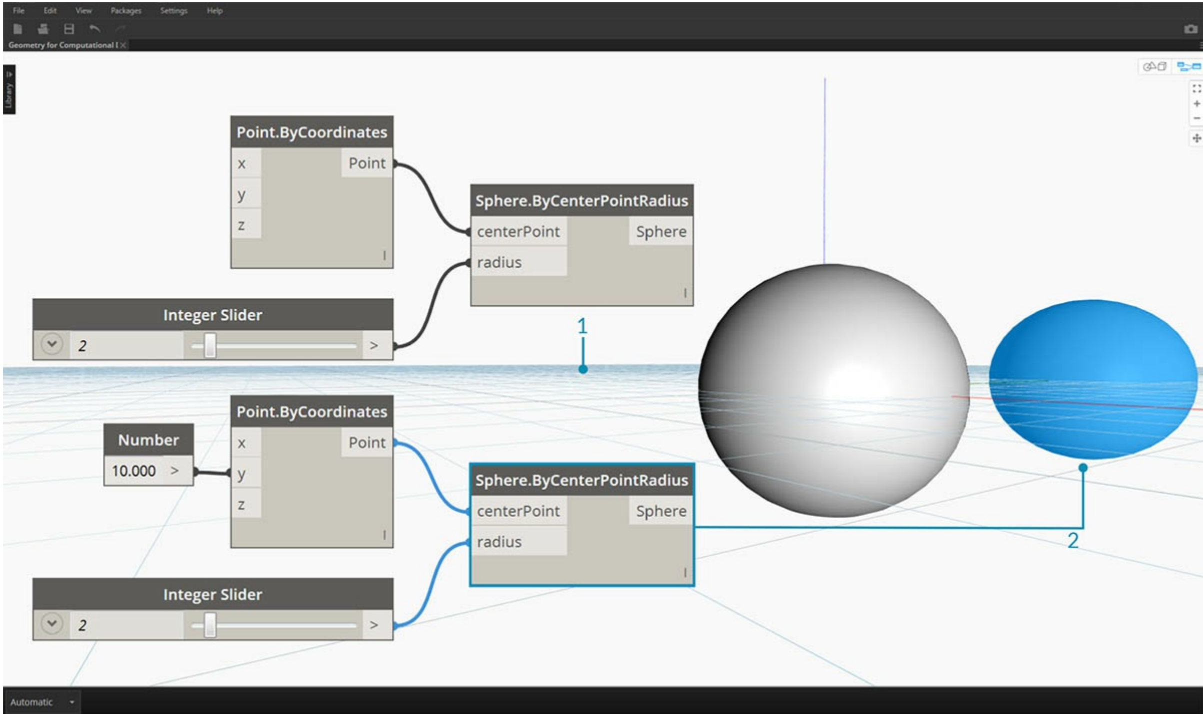
Иерархия типов данных							
Типы абстрактных данных			Типы геометрии				
Определение положения и ориентации	Определение позиции и объема	Определение взаимосвязей	Элементы модели				
Работа с векторами	Ограничивающая рамка	Топология	Точка	Кривая	Поверхность	Тело	Сеть
• Работа с векторами • Плоскость • Система координат	• Ограничивающая рамка	• Вершина • Кромка • Грань	• Координата XYZ • Координата UV	• Линия • Полигон • Дуга • Окружность • Эллипс • NURBS-кривая • Сложная кривая	• Поверхность NURBS • Сложная поверхность	• Кубоид • Сфера • Конус • Цилиндр	• Сеть

Геометрия в Dynamo Sandbox

Каким образом все это работает применительно к Dynamo? Зная типы геометрии и то, как они связаны друг с другом, мы сможем с легкостью ориентироваться в наборе **узлов Geometry**, доступных в основной библиотеке. Узлы Geometry располагаются в библиотеке в алфавитном порядке, а не по иерархическому принципу. Здесь они отображаются примерно так же, как и в интерфейсе Dynamo.



Кроме этого, эти знания позволят упростить и сделать более понятым процесс создания моделей в Dynamo, а также соотнесение потока данных в графике с изображением в области предварительного просмотра.



1. Обратите внимание на предполагаемую систему координат, представленную сеткой и цветными осями.
2. Выбранные узлы визуализируют соответствующую геометрию (если узел создает геометрию) в фоновом режиме.

Скачайте файл примера, прилагаемый к этому изображению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): Geometry for Computational Design - Geometry Overview.dyn. Полный список файлов примеров можно найти в приложении.

Дальнейшая работа с геометрией

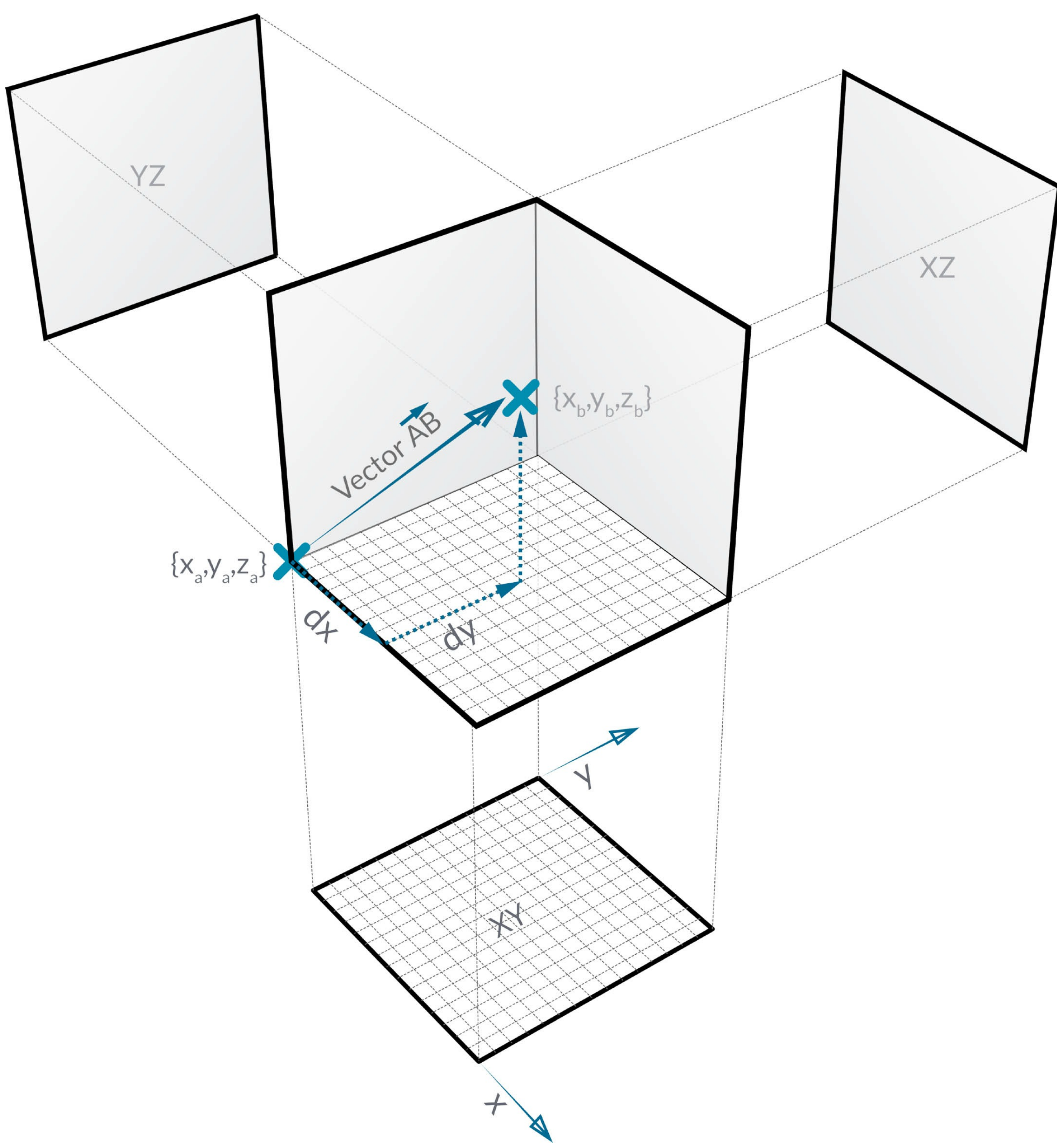
Создание моделей в Dynamo не ограничено объектами, которые можно создать с помощью узлов. Есть несколько способов расширить возможности использования геометрии.

1. Dynamo позволяет импортировать файлы: используйте файлы CSV для создания облаков точек или файлы SAT для добавления поверхностей.
2. При работе с Revit можно ссылаться на элементы Revit, чтобы использовать их в Dynamo.
3. Менеджер пакетов Dynamo содержит дополнительные функции, поддерживающие расширенный набор типов геометрии и операций. Изучите возможности пакета Mesh Toolkit

Векторы

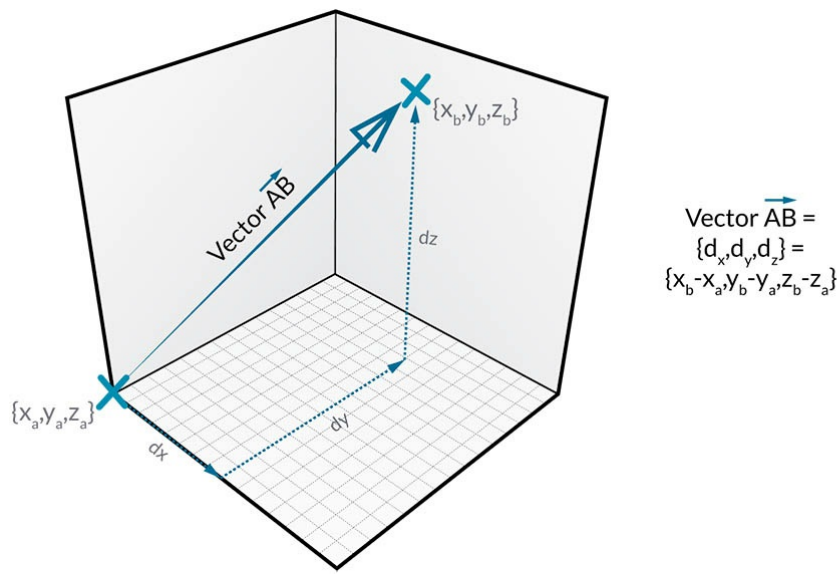
Векторы, плоскости и системы координат

Векторы, плоскости и системы координат составляют основную группу абстрактных типов геометрии. Они помогают задавать расположение, ориентацию и пространственный контекст для других геометрических объектов, определяющих формы. Представим себе человека, который находится в Нью-Йорке на пересечении 42-й улицы и Бродвея (система координат), стоит на тротуаре (плоскость) и смотрит на север (вектор). Мы только что описали местонахождение человека с помощью вспомогательных абстрактных средств. Таким же образом можно задать местонахождение любого объекта, от чехла телефона до небоскреба. Это контекст, необходимый для разработки любой модели.



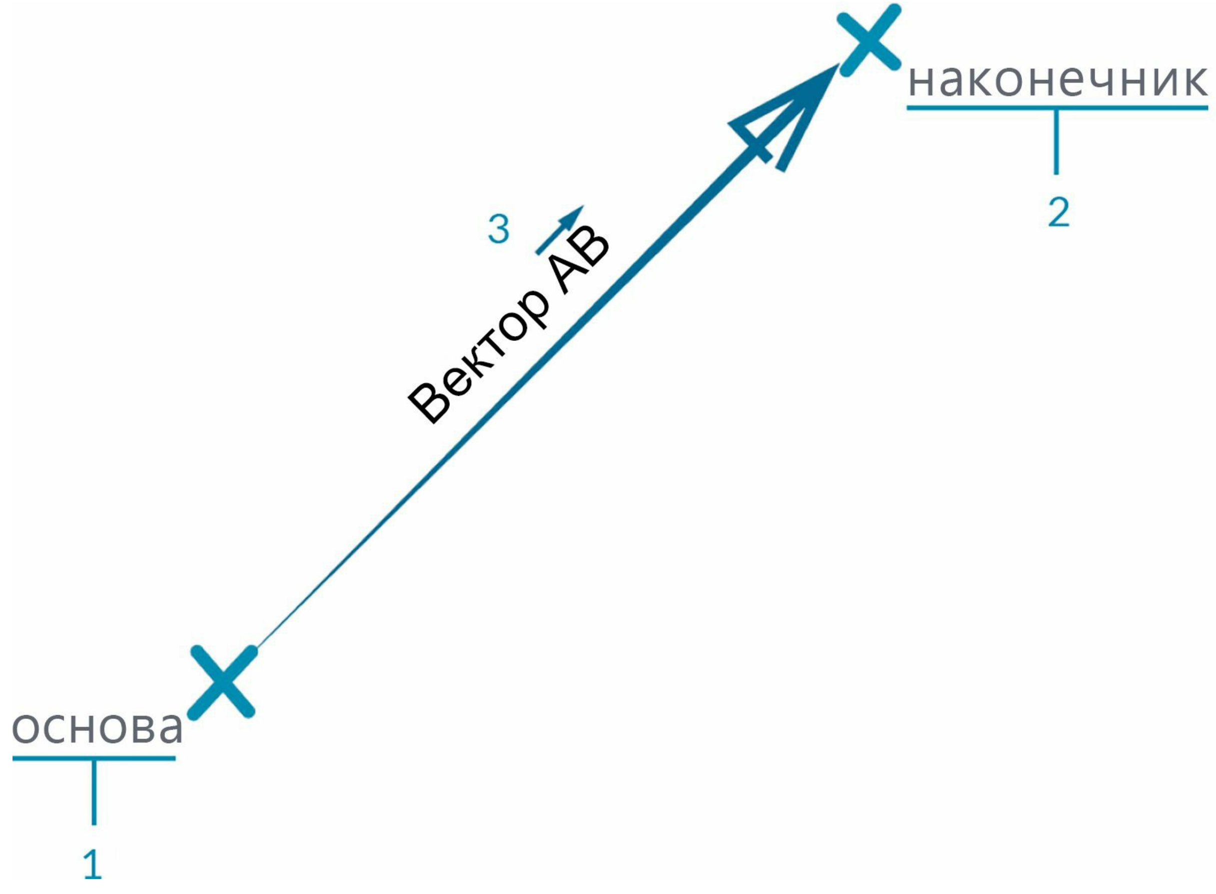
Что такое вектор

Вектор — это геометрическая величина, описывающая направление и длину. Векторы абстрактны, то есть они представляют собой некоторую величину, а не геометрический элемент. Векторы легко спутать с точками, поскольку они также состоят из списка значений. Однако существует одно важное отличие: точки описывают положение в заданной системе координат, а векторы описывают относительную разницу в положении, что аналогично понятию направления.



Если идея относительной разницы не вполне ясна, представьте, что вектор AB означает следующее: вы стоите в точке A и смотрите в точку B. Направление от одной точки (A) до другой (B) и будет вектором.

Рассмотрим подробнее структуру вектором на примере вектора AB.

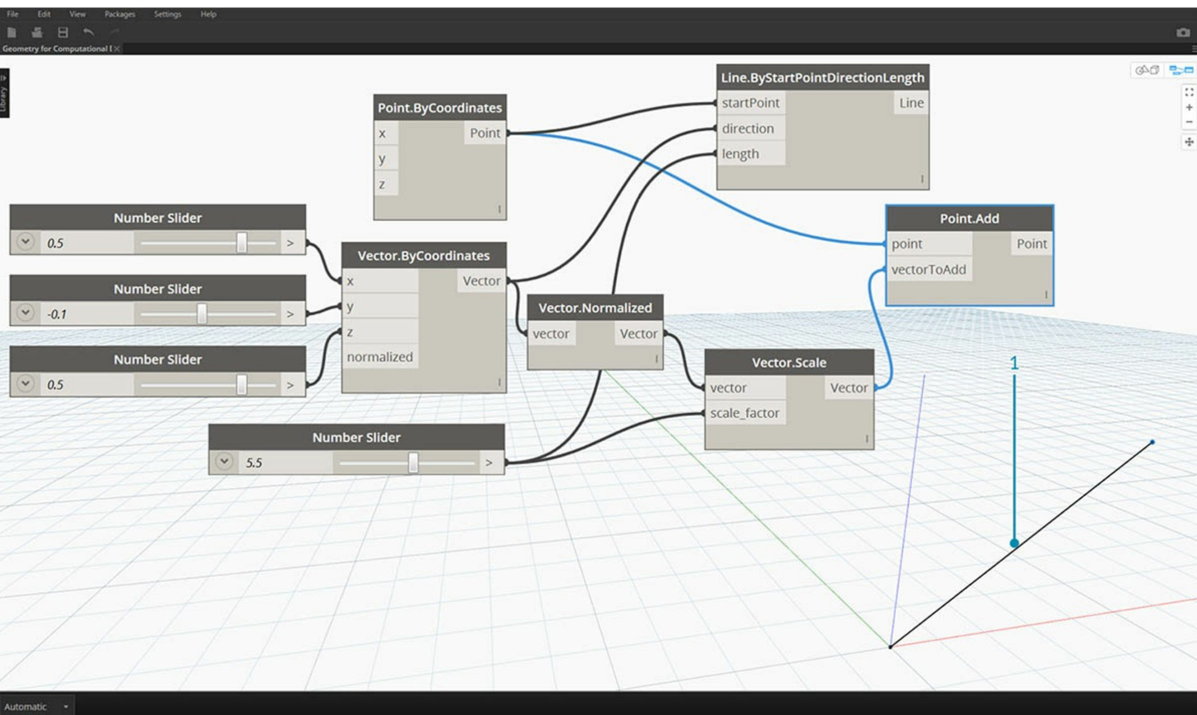


1. **Начальная точка** вектора называется его **началом**.
2. **Конечная точка** вектора называется его **концом** или **направлением**.
3. Вектор AB и вектор BA — это два разных вектора, поскольку они указывают в противоположных направлениях.

Чтобы отдохнуть от слишком формальных и абстрактных определений вектора и посмеяться, посмотрите классическую комедию «Аэроплан!» (Airplane!), где есть следующая знаменитая цитата:

> *Roger, Roger. What's our vector, Victor? (Роджер, Роджер. Каков наш вектор, Виктор?)*

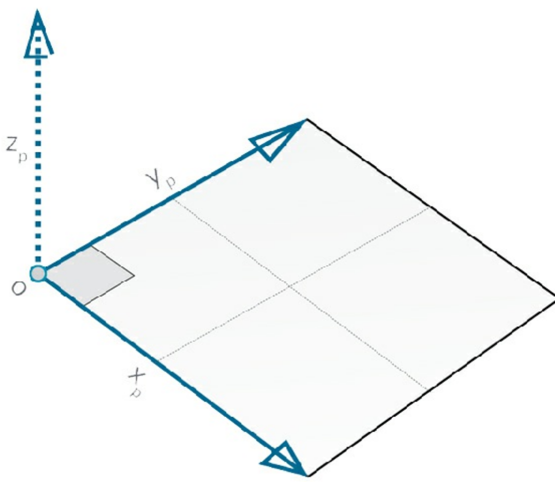
Векторы являются ключевым компонентом при разработке моделей в Dynamo. Обратите внимание, что поскольку они относятся к категории вспомогательных средств, то при создании они не отображаются в области фонового просмотра.



1. В качестве замены вектору в области предварительного просмотра можно использовать отрезок. Скачайте файл примера, прилагаемый к данному изображению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): [Geometry for Computational Design - Vectors.dyn](). Полный список файлов примеров можно найти в приложении.

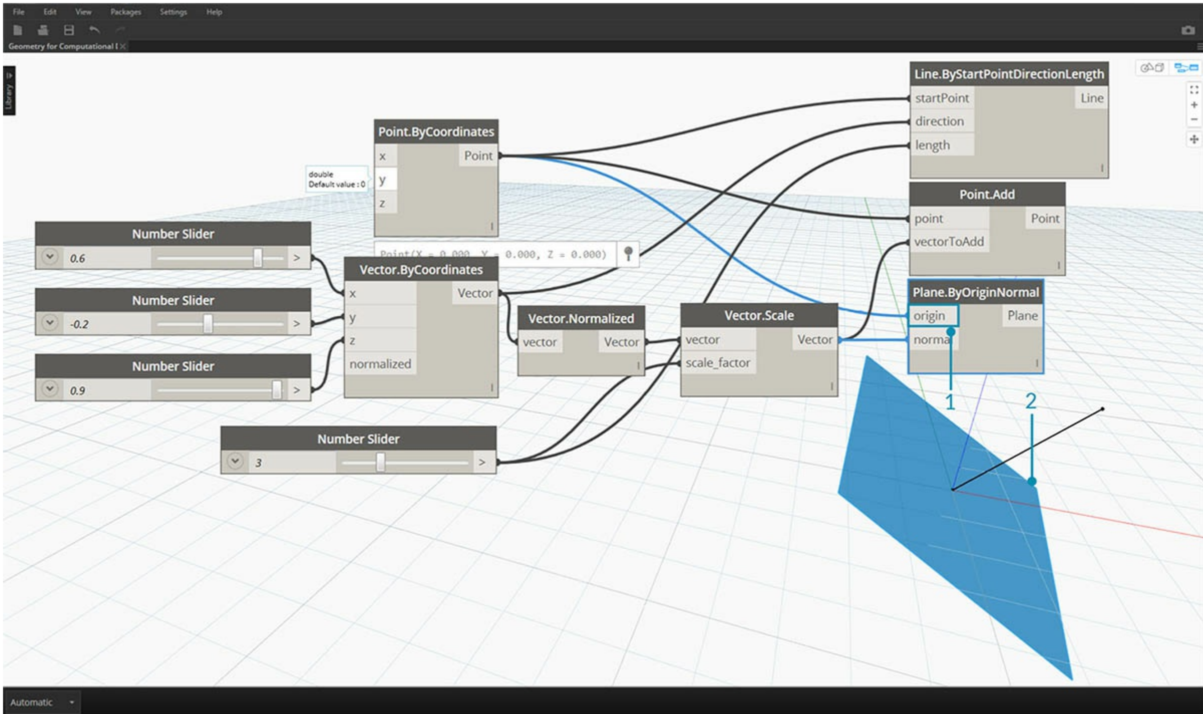
Что такое плоскость

Плоскости являются двумерной абстрактной вспомогательной геометрией. Плоскости по определению являются «плоскими» и бесконечно расширяются в двух направлениях. Обычно они визуализируются в виде небольшого прямоугольника в начале координат.



Но постойте, скажете вы. Начало координат? Как в системе координат, которая используется для моделирования в САПР?

Совершенно верно. В большинстве программ моделирования используются плоскости построения, или «уровни», с помощью которых определяется локальный двумерный контекст для создания черновиков чертежей. XY, XZ, YZ или, возможно, более знакомые вам «север», «юго-восток», «план» — все это плоскости, определяющие бесконечный «плоский» контекст. Плоскости не имеют глубины, но, тем не менее, помогают описать направление: каждая плоскость имеет начало координат, направление по оси X, направление по оси Y и направление по оси Z (вверх).



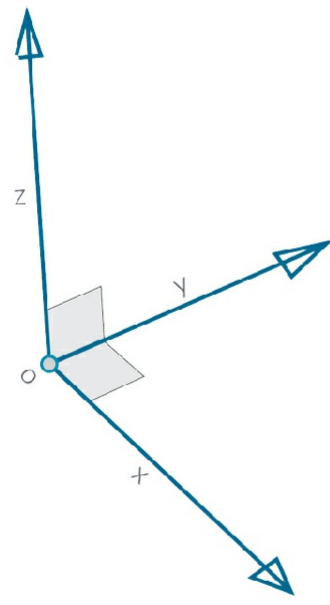
1. Хотя плоскости являются абстрактными, они имеют исходную точку, что позволяет определить их положение в пространстве.
2. В Dynamo плоскости визуализируются в области фонового просмотра. Скачайте файл примера, прилагаемый к данному

изображению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): [Geometry for Computational Design - Planes.dyn](). Полный список файлов примеров можно найти в приложении.

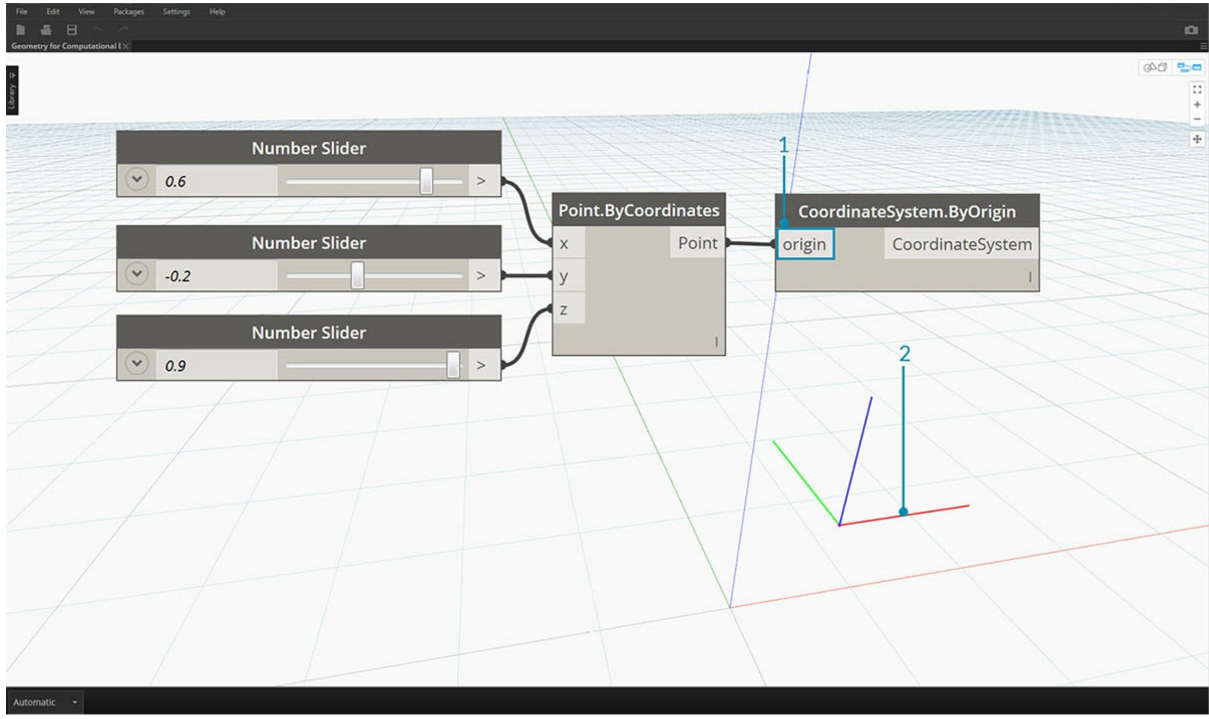
Что такое система координат

Если мы с вами освоили плоскости, то разобраться с системами координат нам не составит труда. Плоскость состоит из тех же компонентов, что и система координат, при условии что это стандартная евклидова система координат или система координат XYZ.

Однако имеются и другие альтернативные системы координат, например цилиндрические или сферические. Как будет показано в следующих разделах, системы координат можно также применять к другим типам геометрии для указания положения в пределах этой геометрии.



Можно добавить альтернативные системы координат — цилиндрические и сферические.

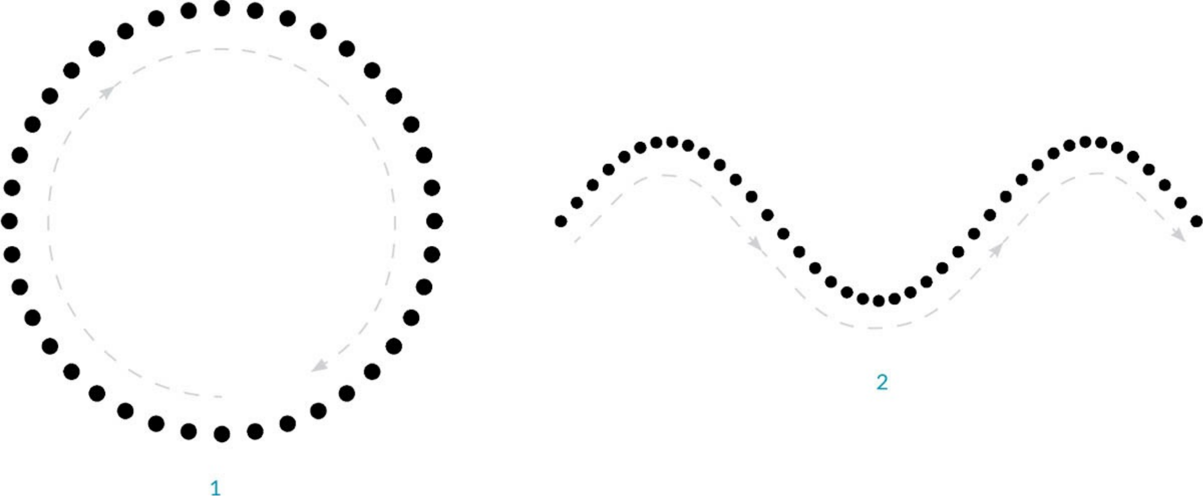


1. Хотя системы координат являются абстрактными, они имеют исходную точку, что позволяет определить их положение в пространстве.
2. В Dynamo системы координат визуализируются в области фонового просмотра в виде точки (начало координат) и линий, определяющих оси (согласно принятым нормам, ось X обозначается красным цветом, Y — зеленым, а Z — синим). Скачайте файл примера, прилагаемый к данному изображению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): [Geometry for Computational Design - Coordinate System.dyn](). Полный список файлов примеров можно найти в приложении.

Точки

Точки

Если геометрия — это язык модели, то точки — ее алфавит. Точки являются основой для создания всех прочих объектов геометрии. Для создания кривой требуется не менее двух точек, для создания полигона или грани сети — не менее трех и т. д. Определение положения, порядка и отношений между точками (например, с помощью функции синуса) позволяет работать с геометрией более высокого порядка, в том числе с такими элементами, как окружности или кривые.



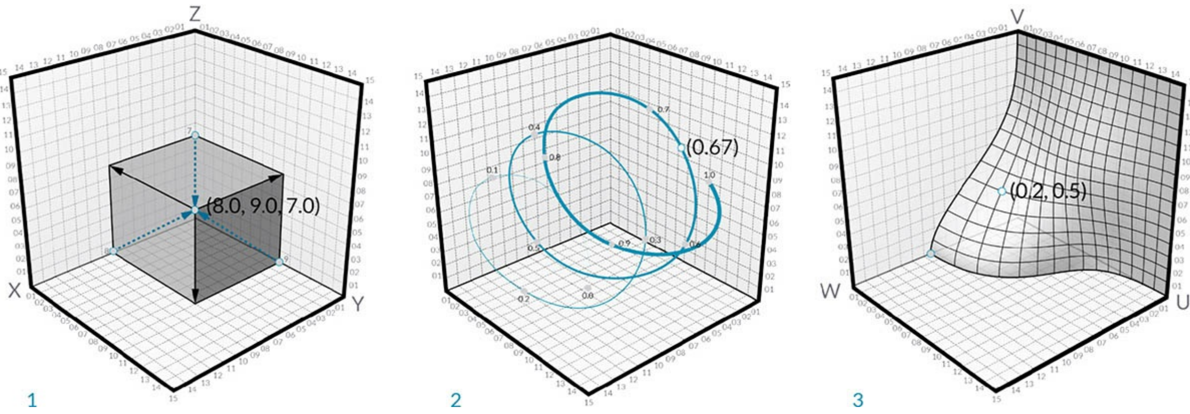
1. Окружность, построенная с помощью функций `x=r*cos(t)` и `y=r*sin(t)`
2. Синусоидальная кривая, построенная с помощью функций `x=(t)` и `y=r*sin(t)`

Что такое точка

Точка определяется одним или несколькими значениями, которые называются координатами. Количество необходимых для определения точки значений координат зависит от системы координат или пространства, в котором она находится. Самый распространенный тип точки в Dynamo существует в трехмерной мировой системе координат и имеет три координаты: [X,Y,Z].

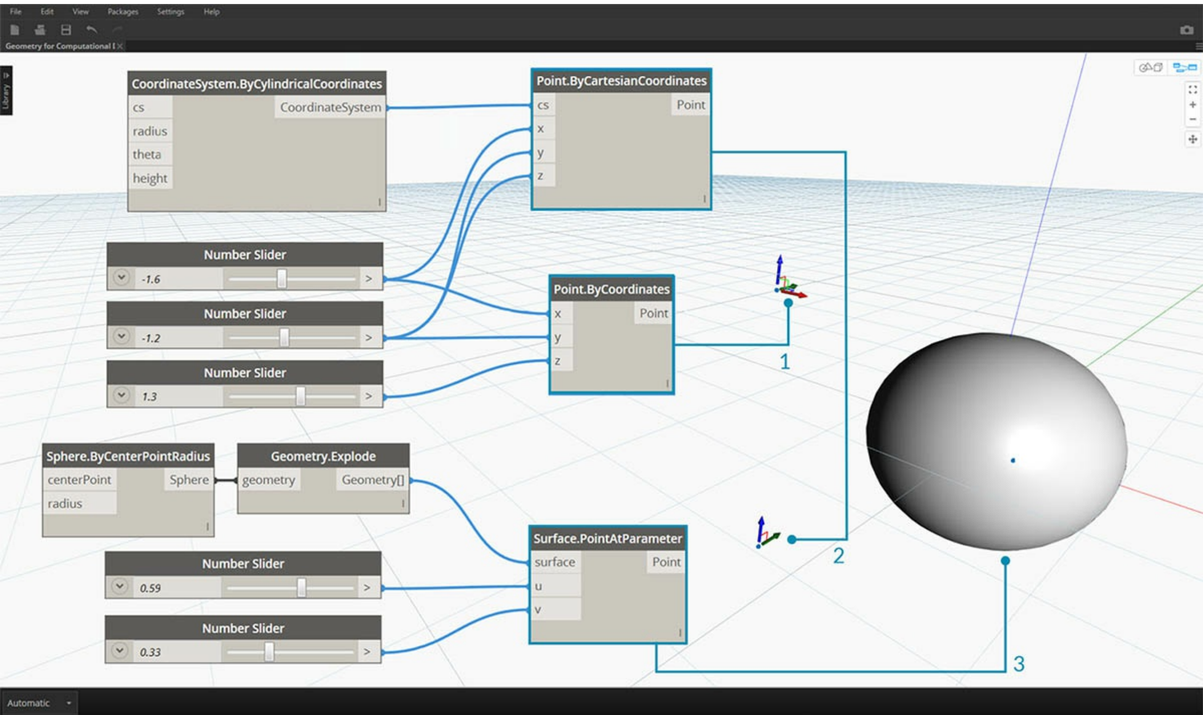
Точка в системе координат

Точки также могут существовать в двумерной системе координат. В зависимости от типа рабочего пространства могут использоваться различные буквенные обозначения — [X,Y] на плоскости и [U,V] на поверхности.



1. Точка в евклидовой системе координат: [X,Y,Z]
2. Точка в системе координат параметров кривой: [t]
3. Точка в системе координат параметров поверхности: [U,V]

Хотя это сложно представить, параметры кривых и поверхностей являются непрерывными и выходят за границы заданной геометрии. Поскольку формы, определяющие параметрическое пространство, находятся в трехмерной мировой системе координат, параметрическую координату легко можно преобразовать в мировую. Например, точка [0.2, 0.5] на поверхности соответствует точке [1.8, 2.0, 4.1] в мировой системе координат.

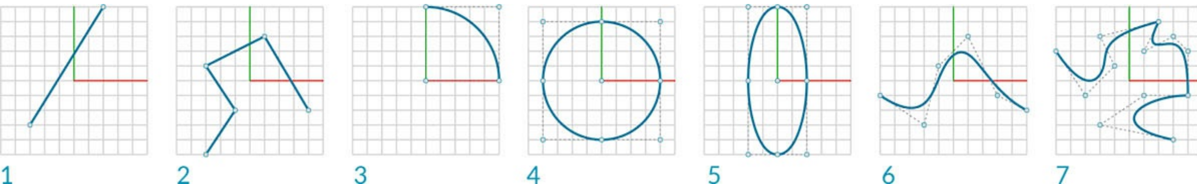


1. Точка в предполагаемой мировой системе координат XYZ.
2. Точка, представленная относительно заданной системы координат (цилиндрической).
3. Точка, представленная координатами UV на поверхности. Скачайте файл примера, прилагаемый к данному изображению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): Geometry for Computational Design - Points.dyn. Полный список файлов примеров можно найти в приложении.

Кривые

Кривые

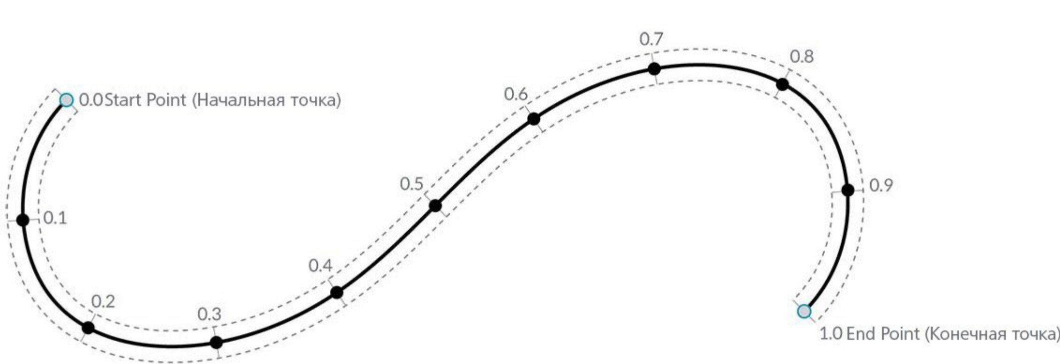
Кривые — это первый из рассматриваемых здесь типов геометрических данных, обладающий привычным набором свойств, определяющих форму объекта (степень изгиба, длина и т. д.). Помните, что основной единицей при построении любых объектов — от отрезка до сплайна и всех прочих типов кривых — остаются точки.



1. Линия
2. Полилиния
3. Дуга
4. Окружность
5. Эллипс
6. NURBS-кривая
7. Сложная кривая

Что такое кривая

Под термином **кривая** обычно понимаются все типы криволинейных форм (даже если они являются прямыми). Таким образом, кривая — это родительская категория, в которую входят все эти типы форм: отрезки, окружности, сплайны и т. д. В техническом плане кривая включает в себя все возможные точки, которые можно найти путем ввода параметра «t» в набор простых (`x = -1.26*t, y = t`) или высших математических функций. Независимо от типа кривой, искомым свойством является данный **параметр**, условно обозначаемый как «t». Кроме этого, все кривые, независимо от своей формы, также имеют начальную и конечную точки, которые соответствуют минимальным и максимальным значениям t, используемым для создания кривой. Это также помогает определить направленность кривой.

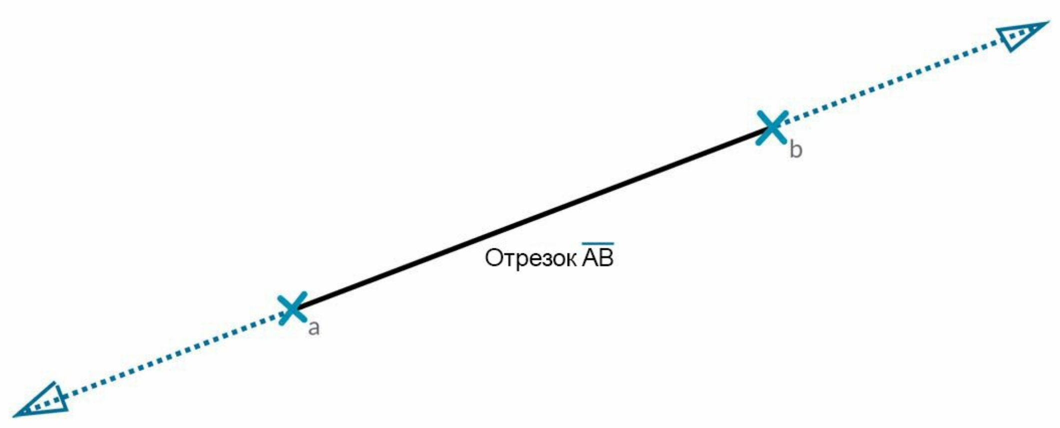


Важно помнить, что в Dynamo область значений t кривой охватывает диапазон от 0,0 до 1,0.

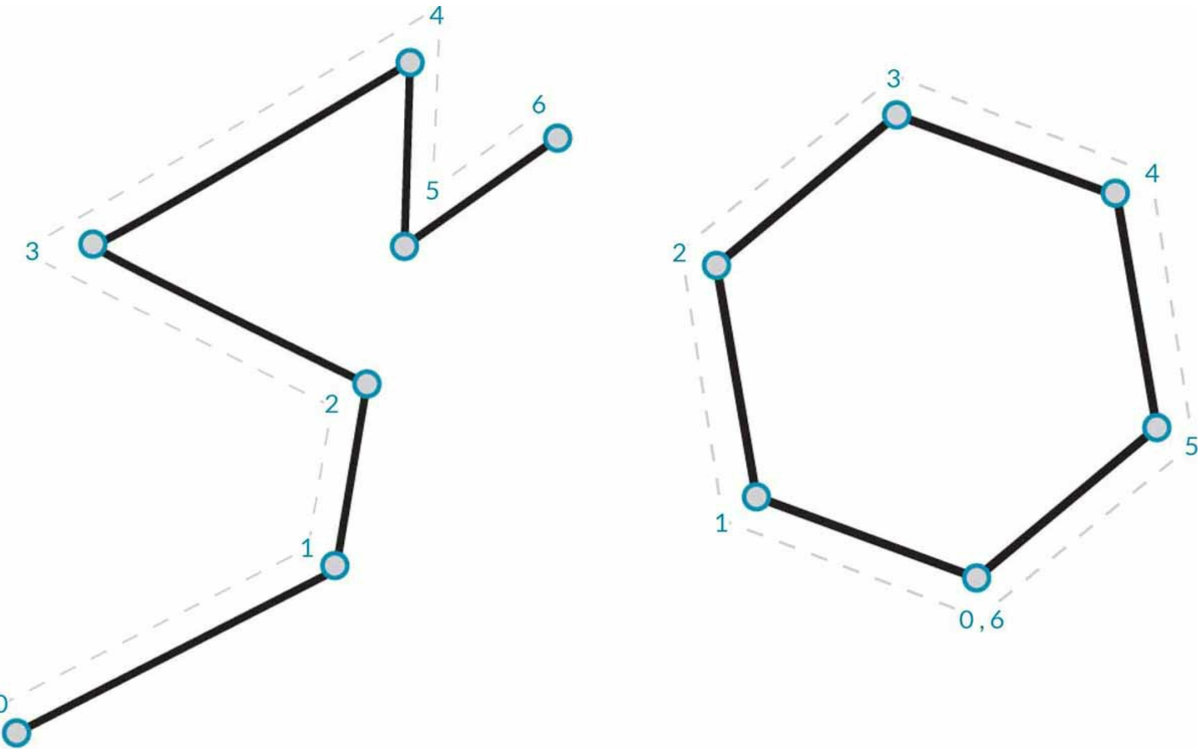
Все кривые также имеют ряд свойств или характеристик, которые можно использовать для их описания или анализа. Если расстояние между начальной и конечной точками равно нулю, кривая будет замкнутой. Кроме этого, каждая кривая имеет несколько управляющих точек. Если все эти точки расположены в одной плоскости, то кривая будет плоской. Некоторые свойства применяются ко всей кривой, другие — только к определенным точкам на кривой. Например, планарность является глобальным свойством, а вектор касательной при заданном значении — локальным.

Отрезки

Отрезки — это простейшая форма кривых. Хотя они могут не выглядеть изогнутыми, на самом деле это кривые, у которых просто отсутствует кривизна. Существует несколько способов создания отрезков, наиболее простым из которых является создание отрезка от точки A до точки B. Форма отрезка AB заключена между этими точками, но математически она бесконечно продолжается в обоих направлениях.

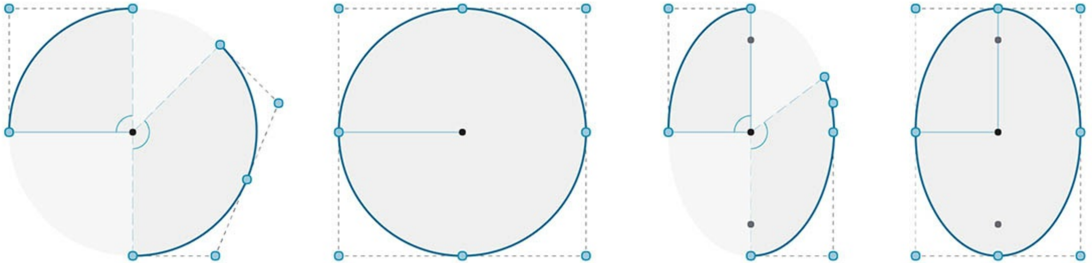


При соединении двух отрезков создается **полилиния**. На этом изображении наглядно показывается, что представляет собой управляющая точка. При изменении положения любой из этих точек изменится и форма полилинии. Если полилиния замкнута, получится полигон. Если длина всех ребер полигона одинакова, он будет правильным.



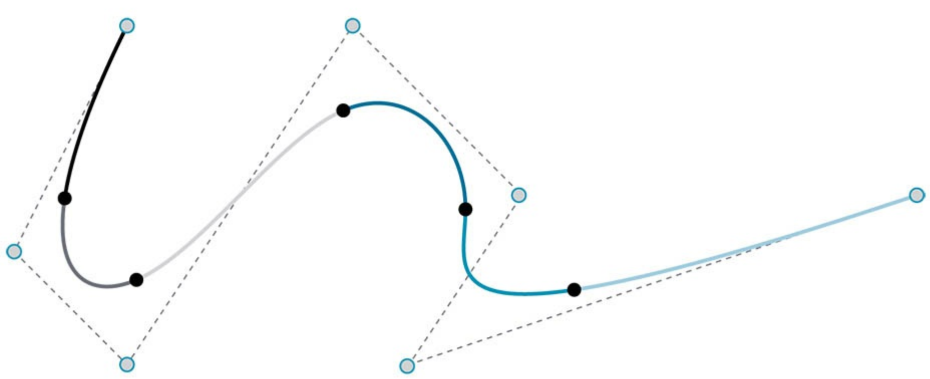
Дуги, окружности, эллиптические дуги и эллипсы

Постепенно усложняя параметрические функции, определяющие форму, можно построить не просто отрезок, а **дугу**, **окружность**, **эллиптическую дугу** или **эллипс**, задав один или два радиуса. Отличие между дугой и окружностью или эллипсом состоит только в том, что последние две формы являются замкнутыми.



NURBS-кривые и сложные кривые

NURBS (неоднородные сплайны с рациональной основой) — это математические представления, которые позволяют точно смоделировать любую форму: от простого двумерного отрезка, окружности, дуги или прямоугольника до сложнейшей трехмерной кривой произвольной формы. Благодаря своей гибкости (плавной интерполяции в зависимости от заданной степени при относительно небольшом количестве управляющих точек) и точности (достигаемой за счет сложных математических вычислений) модели NURBS можно использовать в любом процессе, будь то презентация, анимация или производство.

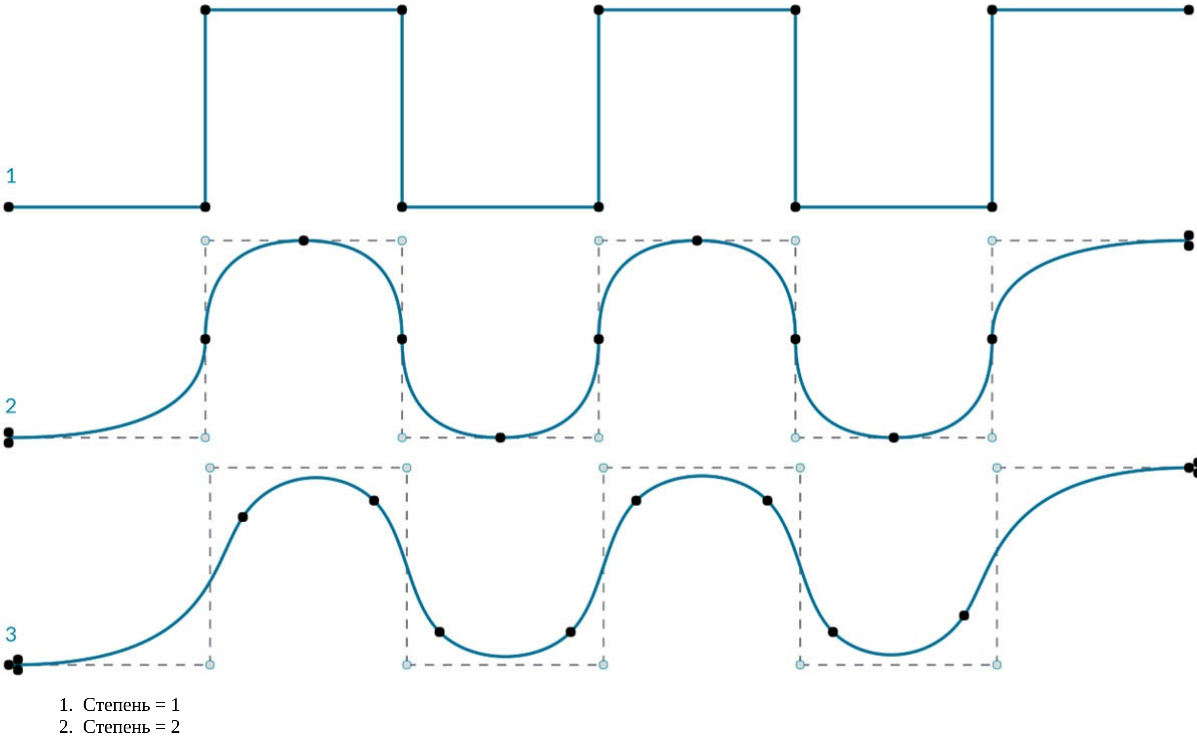


Степень: степень кривой определяет диапазон влияния управляющих точек на кривую (чем выше степень, тем больше диапазон). Степень — положительное целое число. Обычно это число 1, 2, 3 или 5, но вместо него может использоваться любое другое положительное целое число. NURBS-отрезки и полилинии обычно имеют степень 1, а кривые произвольной формы — степени 3 или 5.

Управляющие точки: набор точек в количестве не меньшем, чем «степень + 1». Одним из самых простых способов изменения формы NURBS-кривой является перемещение ее управляющих точек.

Вес: с управляющими точками связано определенное число, которое называется весом. Обычно вес является положительным числом. Если для управляющих точек кривой установлен одинаковый вес (обычно 1), кривая называется нерациональной. В противном случае она считается рациональной. Большинство NURBS-кривых являются нерациональными.

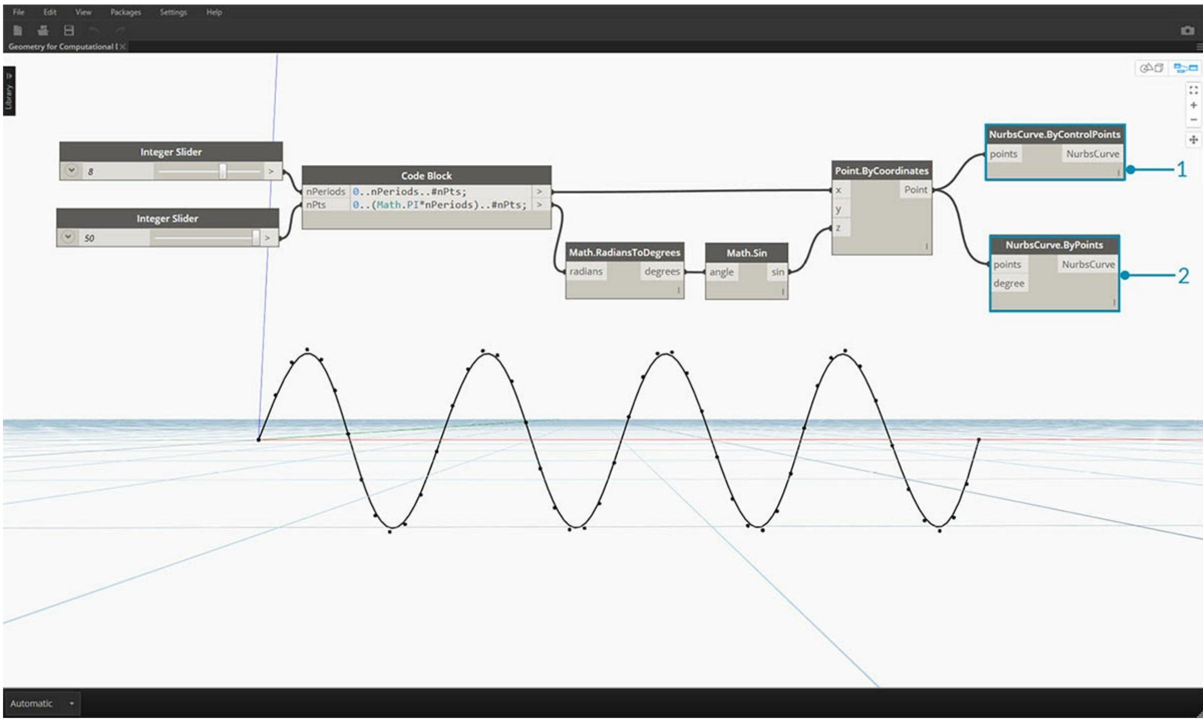
Узлы: список чисел (степень+N-1), где N — количество управляющих точек. Узлы используются вместе со значениями веса для управления влиянием контрольных точек на итоговую кривую. Одной из функций узлов является создание точек излома в определенных точках кривой.



1. Степень = 1
2. Степень = 2
3. Степень = 3

Обратите внимание, что чем выше значение степени, тем больше управляющих точек используется для интерполяции полученной кривой.

Создадим синусоидальную кривую в Dynamo с помощью двух различных методов создания NURBS-кривых и сравним результаты.



1. Узел *NurbsCurve.ByControlPoints* в качестве управляющих точек использует список точек.
2. Узел *NurbsCurve.ByPoints* создает кривую по списку точек. Скачайте файл примера, прилагаемый к данному изображению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): Geometry for Computational Design - Curves.dyn. Полный список файлов примеров можно найти в приложении.

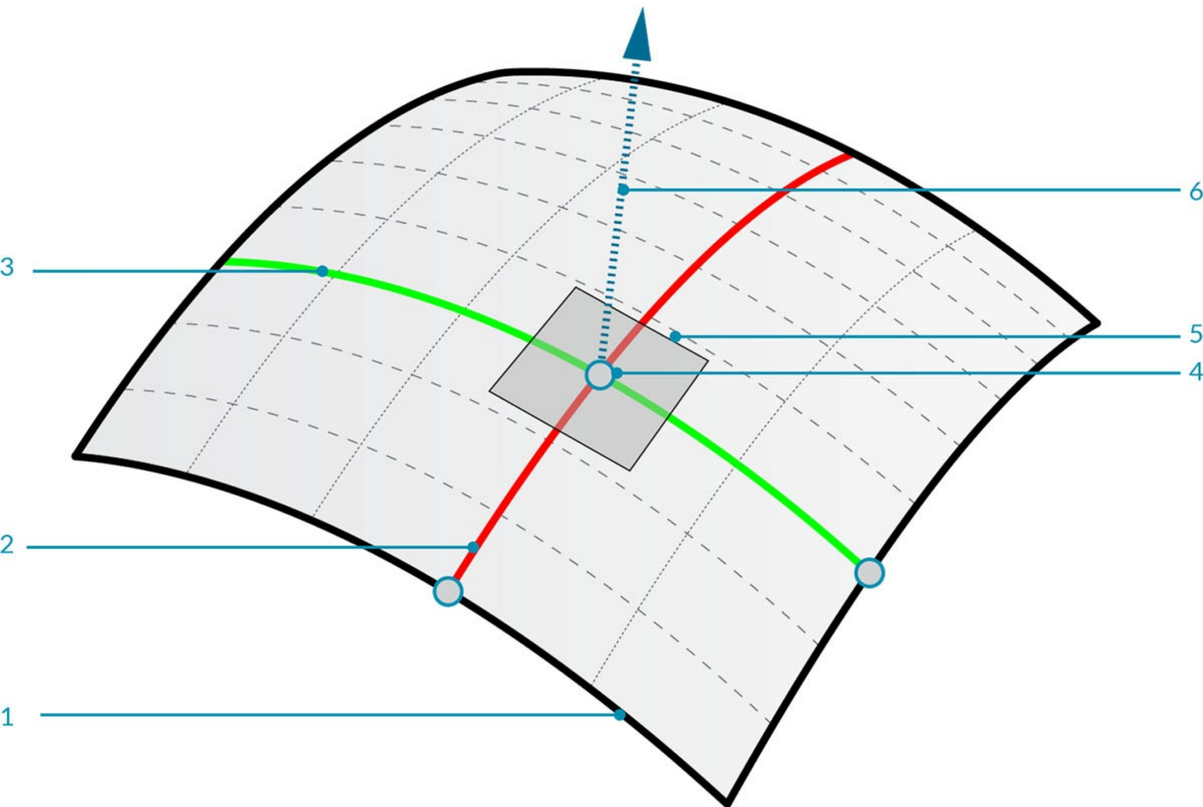
Поверхности

Поверхности

Переход от кривых к поверхностям при работе над моделью позволяет нам добавлять в нее объекты, существующие в реальном трехмерном мире. Несмотря на то что кривые не всегда являются плоскими и по сути трехмерны, пространство, определяемое ими, всегда является одномерным. Поверхности позволяют придать модели дополнительное измерение, а также включают набор специальных свойств, которые можно использовать при выполнении других операций моделирования.

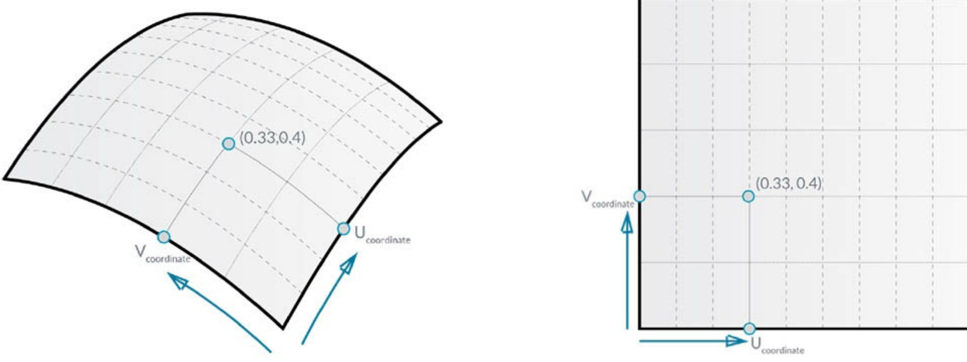
Что такое поверхность

Поверхность — это математическая форма, определяемая функцией и двумя параметрами. Вместо параметра `t`, используемого для кривых, здесь для описания соответствующего пространства используются параметры `U` и `V`. Это означает, что при работе с геометрией этого типа появляются дополнительные данные для использования. Например, у кривых есть касательные векторы и плоскости нормали (которые могут поворачиваться или скручиваться вдоль кривой), а у поверхностей есть векторы нормали и касательные плоскости с последовательной ориентацией.



1. Поверхность
2. Изолиния U
3. Изолиния V
4. Координата UV
5. Перпендикулярная плоскость
6. Вектор нормали

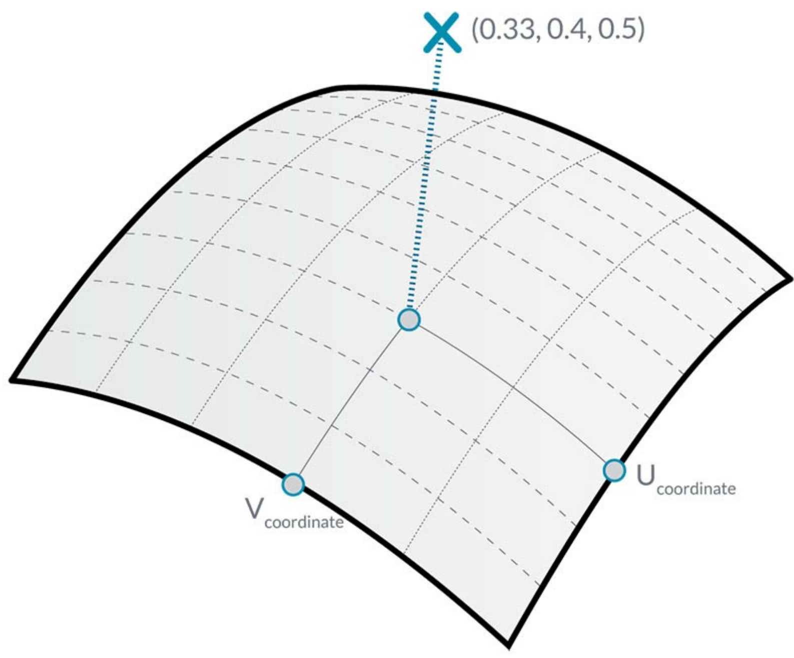
Область поверхности: определяется как диапазон параметров (U,V), каждый из которых соответствует трехмерной точке на этой поверхности. Область каждого измерения (U или V) обычно определяется двумя числами: (от U мин. до U макс.) и (от V мин. до V макс.).



Хотя поверхность может не выглядеть как прямоугольник, а некоторые ее участки могут отличаться более или менее плотным расположением изолиний, «пространство», определяемое областью поверхности, всегда является двумерным. В Dynamo всегда подразумевается, что область поверхности определяется диапазоном значений U и V, где минимальное значение равно 0.0, а максимальное — 1.0. У плоских или обрезанных поверхностей могут быть разные области.

Изолиния (или изопараметрическая кривая): кривая, определяемая постоянным значением для одного направления (U или V) на поверхности и областью значений для другого направления (V или U, соответственно).

Координата UV: точка в пространстве параметров UV, определяемая значениями U, V и иногда W.

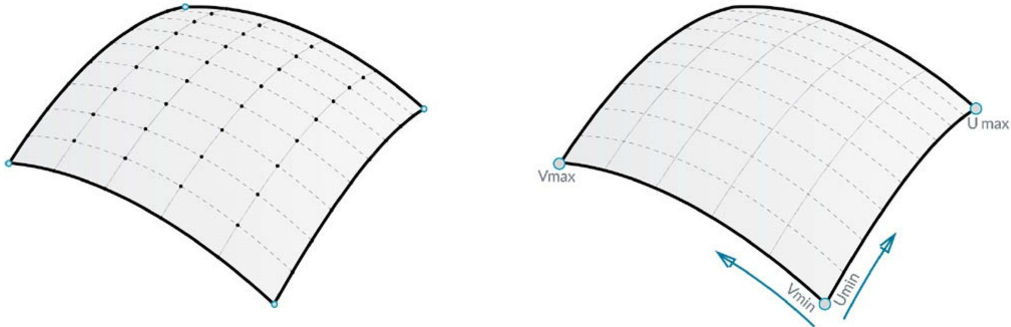


Перпендикулярная плоскость: плоскость, перпендикулярная изолиниям U и V в заданной координате UV.

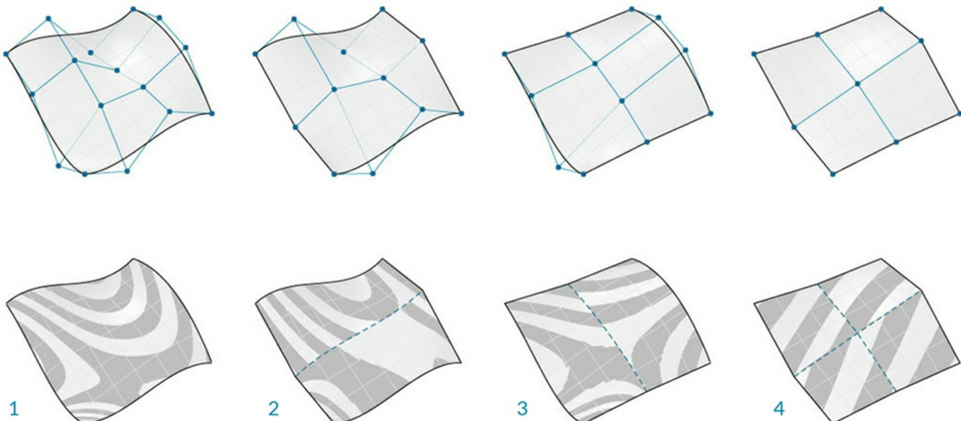
Вектор нормали: вектор, определяющий направление вверх относительно перпендикулярной плоскости.

Поверхности NURBS

Поверхности NURBS очень похожи на NURBS-кривые. Такую поверхность можно представить как сетку из NURBS-кривых, идущих в двух направлениях. Форма поверхности NURBS определяется набором управляющих точек и степенью сглаживания этой поверхности в направлениях U и V. Те же алгоритмы используются для вычисления формы, нормалей, касательных, кривизны и других свойств с помощью управляющих точек, весов и степени сглаживания.



В случае с поверхностями NURBS для геометрии подразумевается два направления, поскольку эти поверхности являются прямоугольными сетками из управляющих точек, хотя они и могут выглядеть совсем по-другому. Эти направления во многих случаях задаются произвольным образом на основе мировой системы координат, однако они часто используются для анализа моделей или создания других геометрических объектов на основе поверхности.

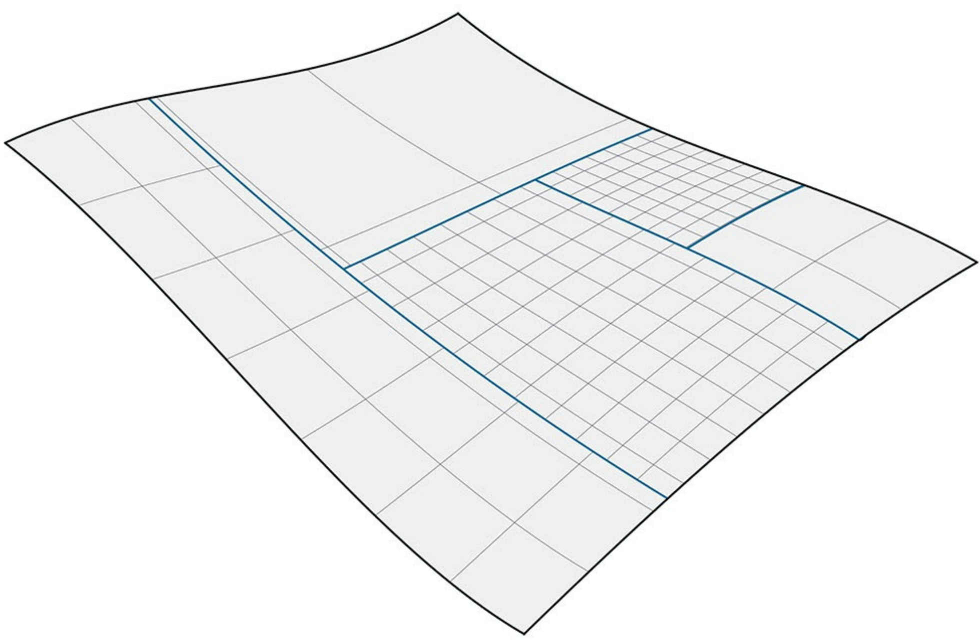


1. Степень сглаживания (U,V) = (3,3)
2. Степень сглаживания (U,V) = (3,1)
3. Степень сглаживания (U,V) = (1,2)
4. Степень сглаживания (U,V) = (1,1)

Полиповерхности

Полиповерхности состоят из нескольких поверхностей, кромки которых соединены. Полиповерхности обеспечивают более детализированные сведения, нежели простое двумерное определение UV, благодаря чему их можно использовать для перехода по соединенным формам посредством их топологии.

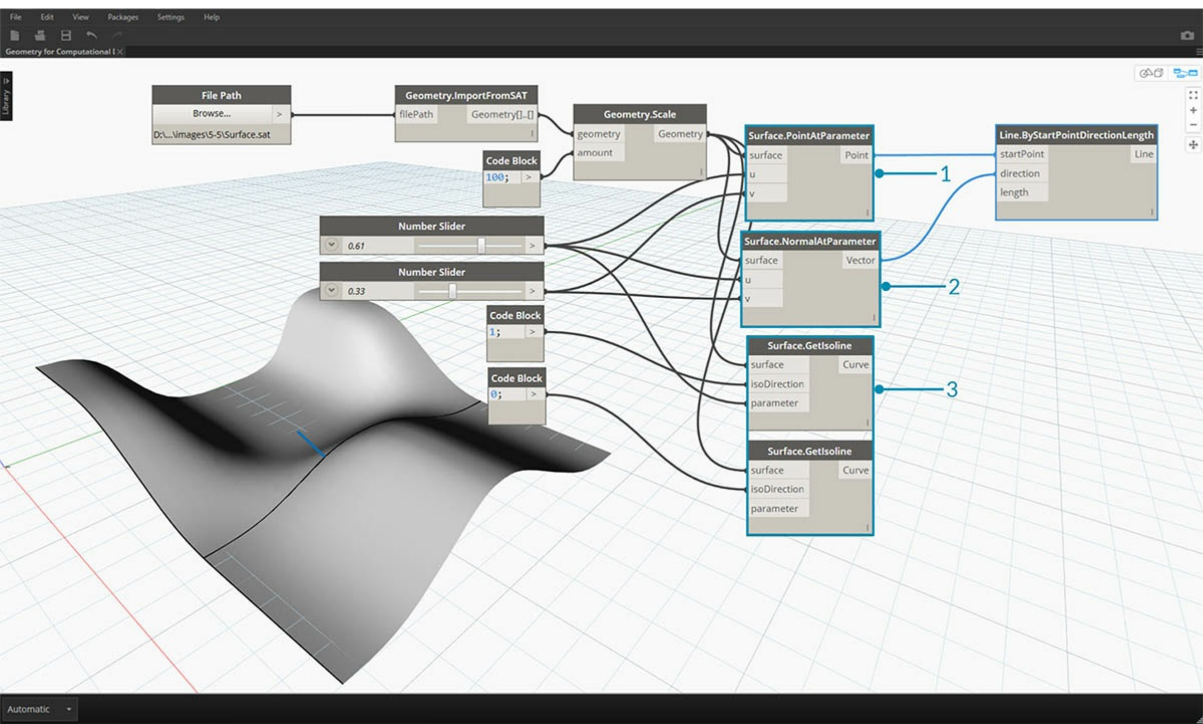
> Термин «топология» в большинстве случаев используется для обозначения того, как различные элементы связаны и взаимодействуют друг с другом. В Dynamo топология (Topology) также является типом геометрии. Topology является родительской категорией таких объектов, как поверхности (Surface), полиповерхности (Polysurface) и тела (Solid).



Объединение поверхностей таким образом (иногда называемым замыканием) позволяет создавать более сложные формы, а также детализировать

стыки. К кромкам объекта Polysurface можно применять операции сопряжения или фаски.

Импортируйте в Dynamo и проанализируйте объект Surface в конкретном параметре, чтобы узнать, какие сведения можно извлечь.



1. *Surface.PointAtParameter* возвращает объект Point в заданной координате UV.
2. *Surface.NormalAtParameter* возвращает вектор нормали в заданной координате UV.

3. *Surface.GetIsoline* возвращает изопараметрическую кривую в координате U или V (обратите внимание на порт ввода isoDirection). Скачайте файлы примера для этого изображения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении.

4. Geometry for Computational Design - Surfaces.dyn

5. Surface.sat

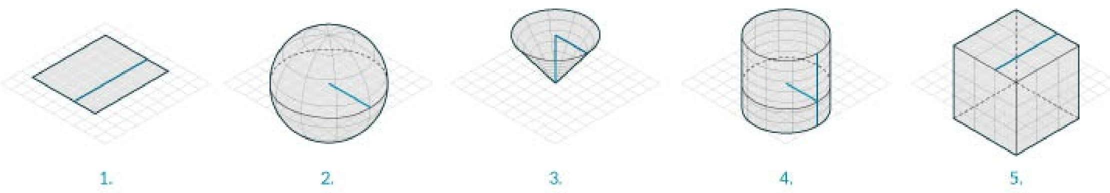
Тела

Тела

Чтобы создавать более сложные модели, которые невозможно получить из одной поверхности, или определять явный объем, следует научиться работе с телами (и полиповерхностями). Даже для самого простого куба требуется целых шесть поверхностей, по одной на каждую грань. Тела позволяют получить доступ к двум ключевым концепциям, не доступным при работе с поверхностями, а именно к уточненным топологическим описаниям (граням, кромкам, вершинам) и логическим операциям.

Что такое тело

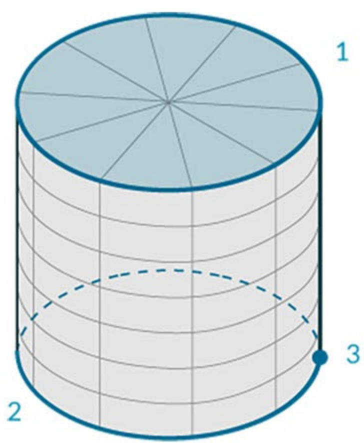
Тела состоят из одной или нескольких поверхностей, внутри которых заключен объем, определенный замкнутым контуром, который отделяет то, что внутри тела, от того, что снаружи. Независимо от количества используемых поверхностей, для того чтобы объект считался телом, содержащийся в нем объем должен быть полностью замкнутым. Тела можно создавать путем объединения поверхностей или полиповерхностей либо с помощью таких операций, как лофтинг, сдвиг и вращение. Такие примитивы, как сфера, куб, конус и цилиндр, также являются телами. Объект Cube, у которого отсутствует хотя бы одна грань, считается полиповерхностью, которая уже не является телом, хотя и обладает многими аналогичными свойствами.

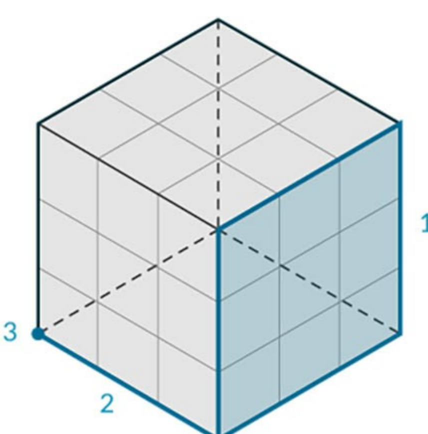


1. Плоскость состоит из одной поверхности и не является телом.
2. Сфера состоит из одной поверхности и *является* телом.
3. Конус состоит из двух соединенных поверхностей и является телом.
4. Цилиндр состоит из трех соединенных поверхностей и является телом.
5. Куб состоит из шести соединенных поверхностей и является телом.

Топология

Элементы, из которых состоят тела, делятся на три типа: вершины, кромки и грани. Грани — это поверхности, образующие тело. Кромки — это кривые, обозначающие области соединения смежных граней, а вершины — это начальные и конечные точки этих кривых. Эти элементы можно запросить с помощью узлов Topology.



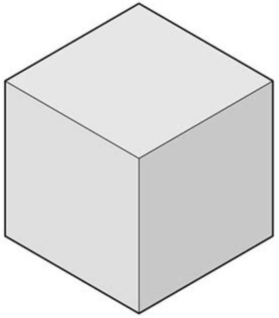
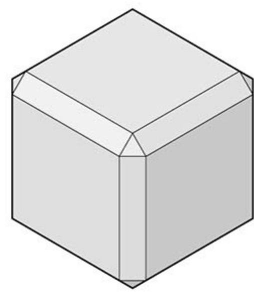
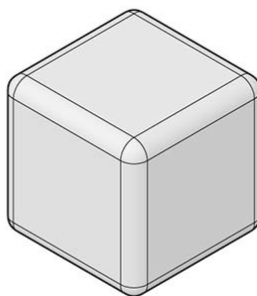


1. Грани
2. Кромки

3. Вершины

Операции

Тела можно изменять путем применения скруглений и фасок к кромкам, чтобы тем самым сгладить острые углы. Операция фаски создает поверхность соединения между двумя гранями, а операция сопряжения сглаживает переход между гранями для сохранения касательности.

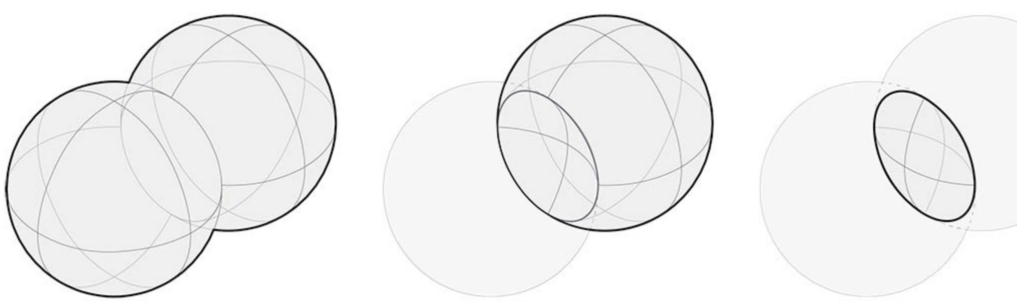
1. Твердотельный куб
2. Куб с фасками
3. Скругленный куб

Логические операции

Логические операции для тел — это методы, позволяющие объединить несколько тел в одно. Каждая логическая операция включает в себя четыре операции:

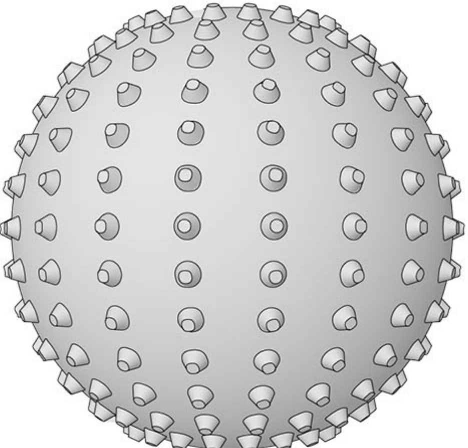
1. **Пересечение** нескольких объектов.
2. **Разделение** их в местах пересечения.
3. **Удаление** ненужных частей геометрии.
4. **Объединение** оставшихся частей.

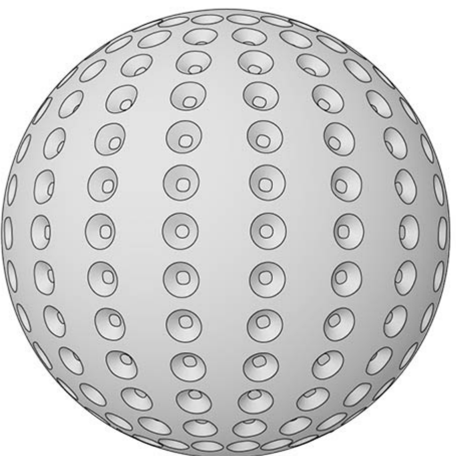
Благодаря этому использование логических операций для тел позволяет значительно сэкономить время. Существует три логические операции для тел, позволяющие определить, как части геометрии должны сохраняться.



1. **Объединение:** несколько тел объединяются в одно с удалением перекрывающихся частей.
2. **Разница:** одно тело вычитается из другого. Тело, которое вычитается, называется инструментом. Обратите внимание, что для сохранения обратного объема вычитаемое и подвергающееся вычитанию тела можно поменять местами.
3. **Пересечение:** при пересечении сохраняются только перекрывающиеся части двух тел.

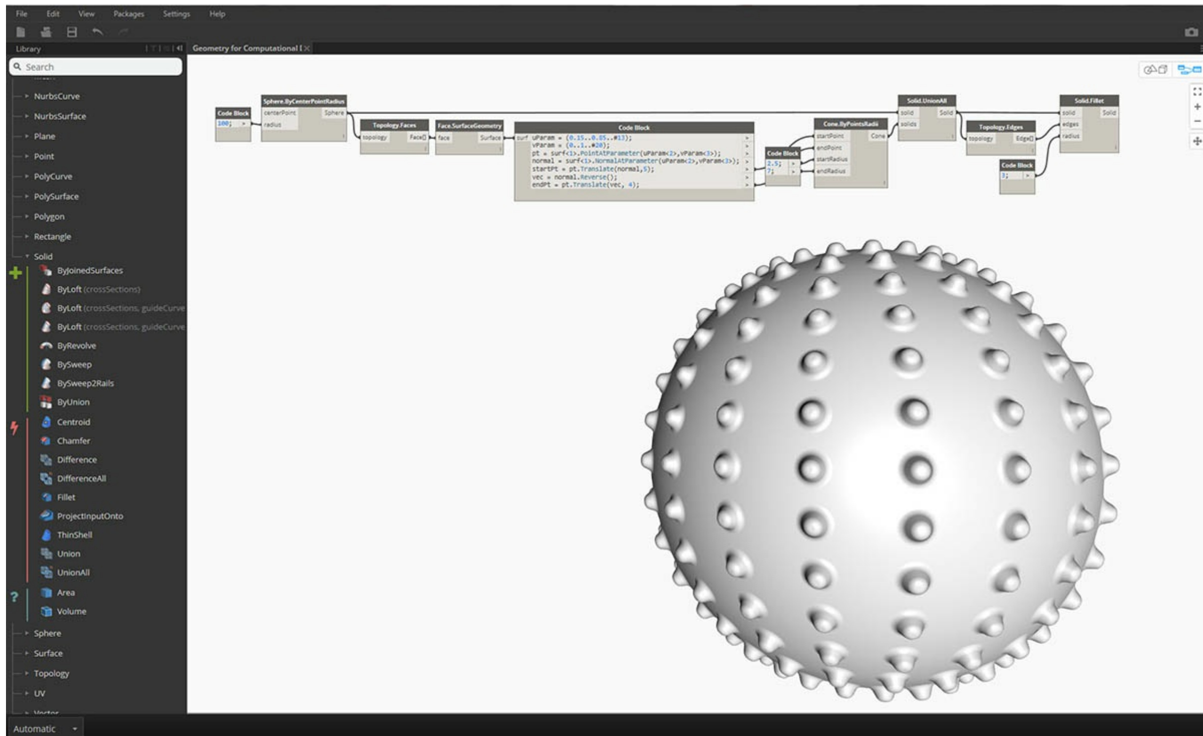
В дополнение к этим трем операциям в Dynamo доступны узлы **Solid.DifferenceAll** и **Solid.UnionAll** для выполнения операций разности и объединения с несколькими телами.





1. **UnionAll:** операция объединения сферы и повернутых наружу конусов.
2. **DifferenceAll:** операция разности между сферой и повернутыми внутрь конусами.

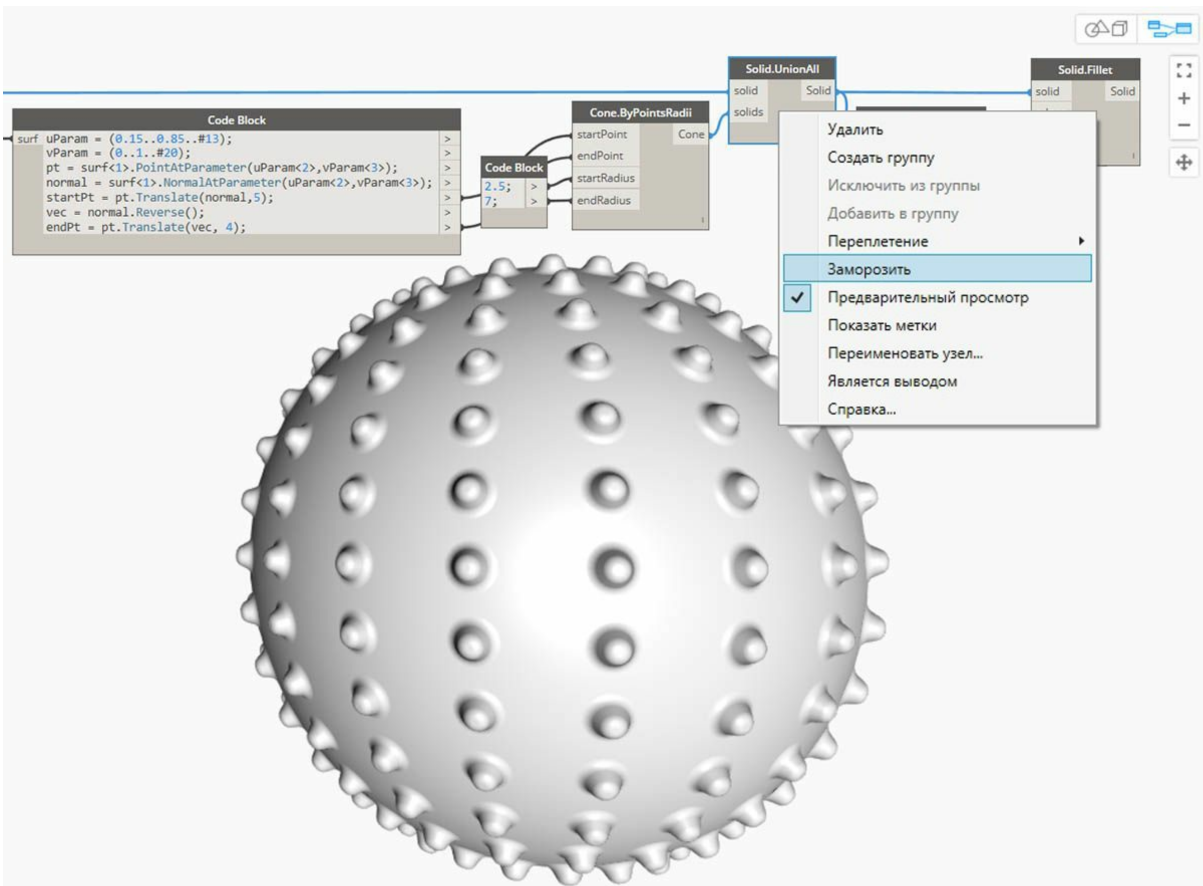
Попробуйте использовать несколько логических операций для создания шара с шипами.



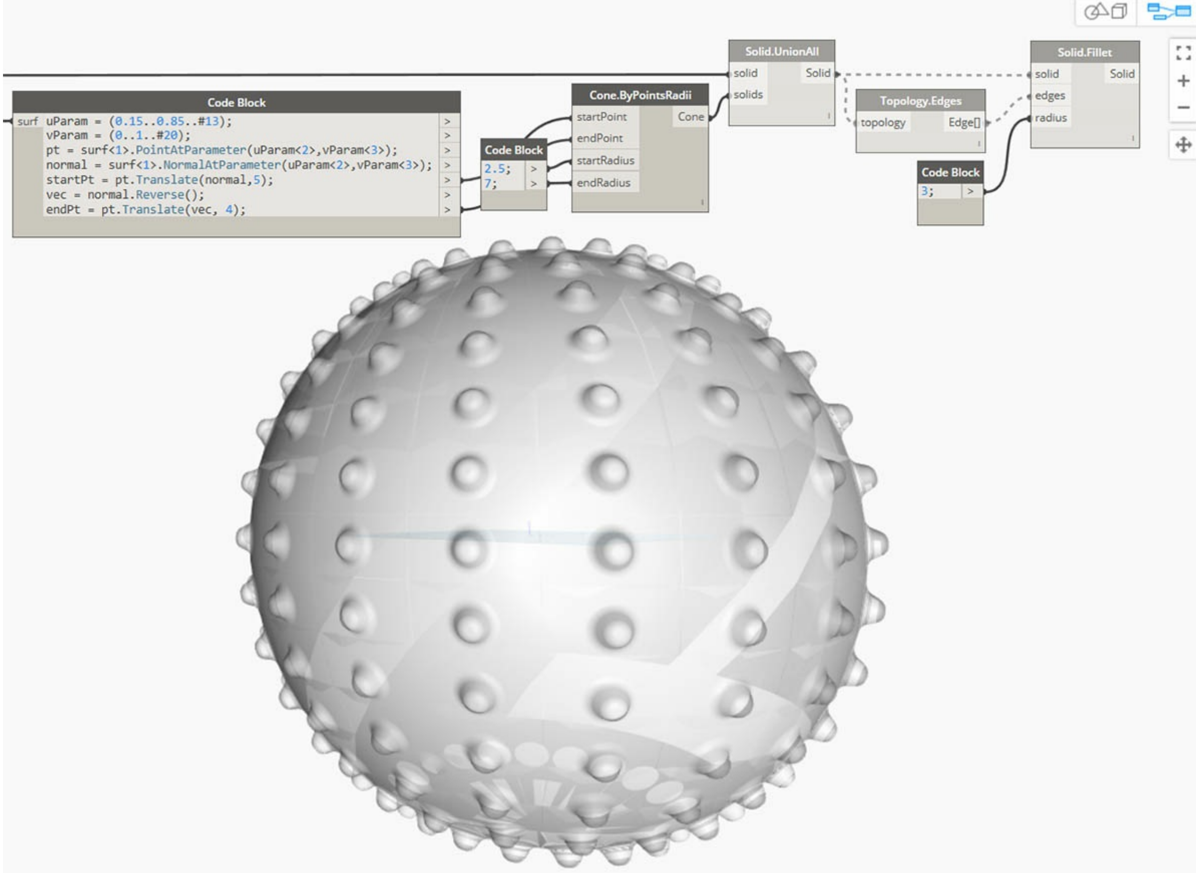
1. **Sphere.ByCenterPointRadius**: создайте базовый объект тела Solid.
2. **Topology.Faces**, **Face.SurfaceGeometry**: выполните запрос граней тела и преобразование в геометрию поверхности (в данном случае сфера имеет только одну грань).
3. **Cone.ByPointsRadii**: постройте конусы, используя точки на поверхности.
4. **Solid.UnionAll**: объедините конусы со сферой.
5. **Topology.Edges**: выполните запрос кромок нового объекта Solid.
6. **Solid.Fillet**: выполните сглаживание кромок шара с шипами. Скачайте файлы примера для этого изображения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. [Geometry for Computational Design - Solids.dyn]

Замораживание

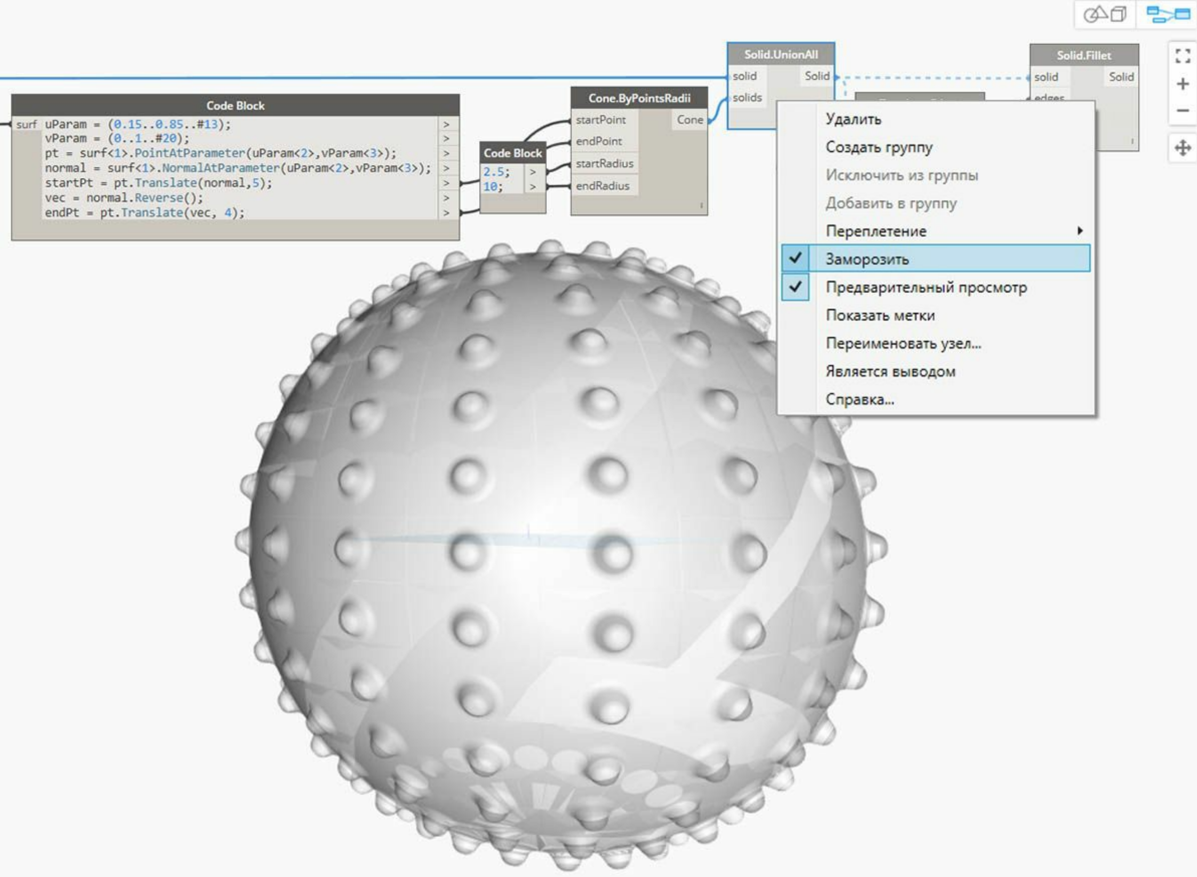
Логические операции сложны, и их вычисление может занимать много времени. Используйте команду «Заморозить», чтобы приостановить выполнение операций в выбранных и следующих за выбранными узлах.



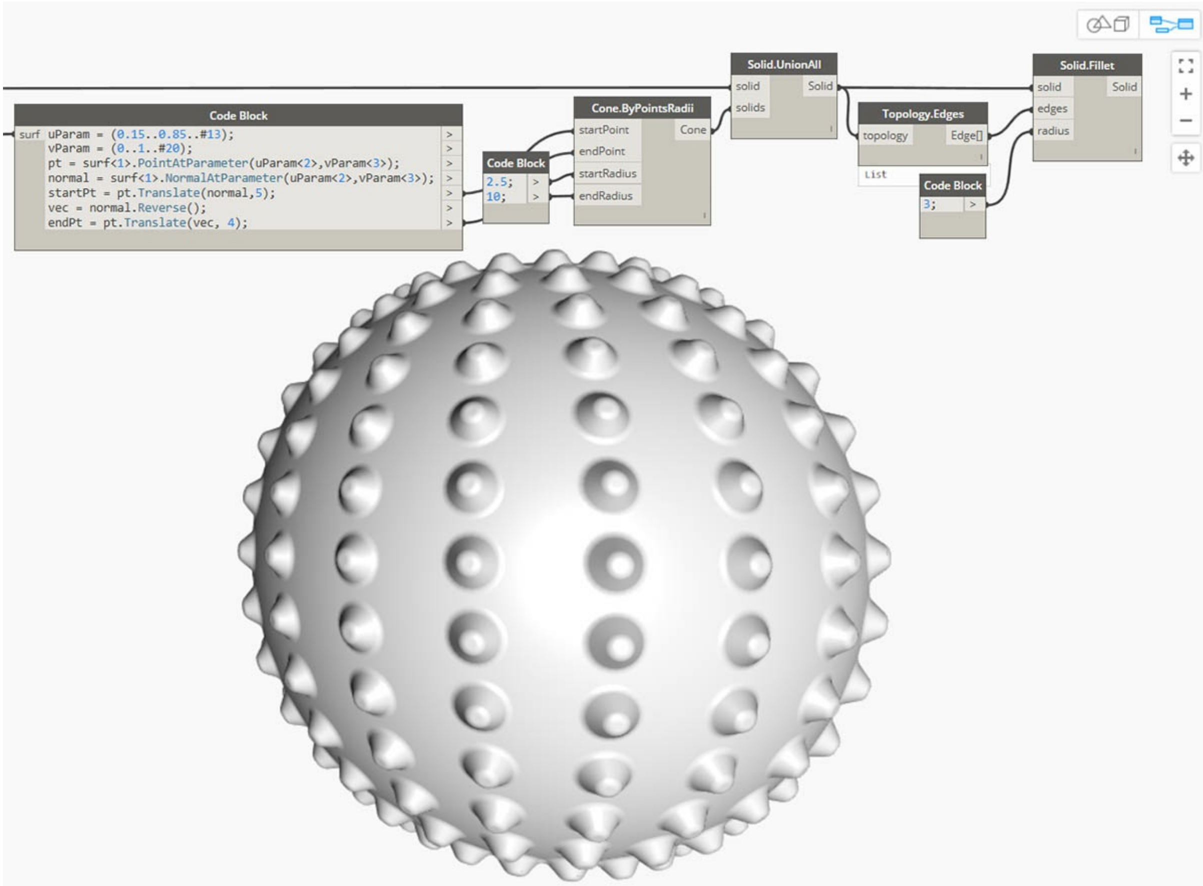
Используйте контекстное меню, чтобы заморозить операцию объединения тел.



Выбранный узел и все следующие за ним узлы отображаются светло-серым полупрозрачным цветом, а все затронутые провода отображаются в виде прерывистых линий. Предварительный просмотр соответствующей геометрии также будет полупрозрачным. Теперь можно изменить значения в узлах, предшествующих выбранному, не перегружая приложение расчетом логической операции объединения.



Чтобы разморозить узлы, щелкните правой кнопкой мыши и снимите флажок «Заморозить».



Все затронутые узлы и изображения предварительного просмотра связанных геометрических объектов обновляются и возвращаются к стандартному виду.

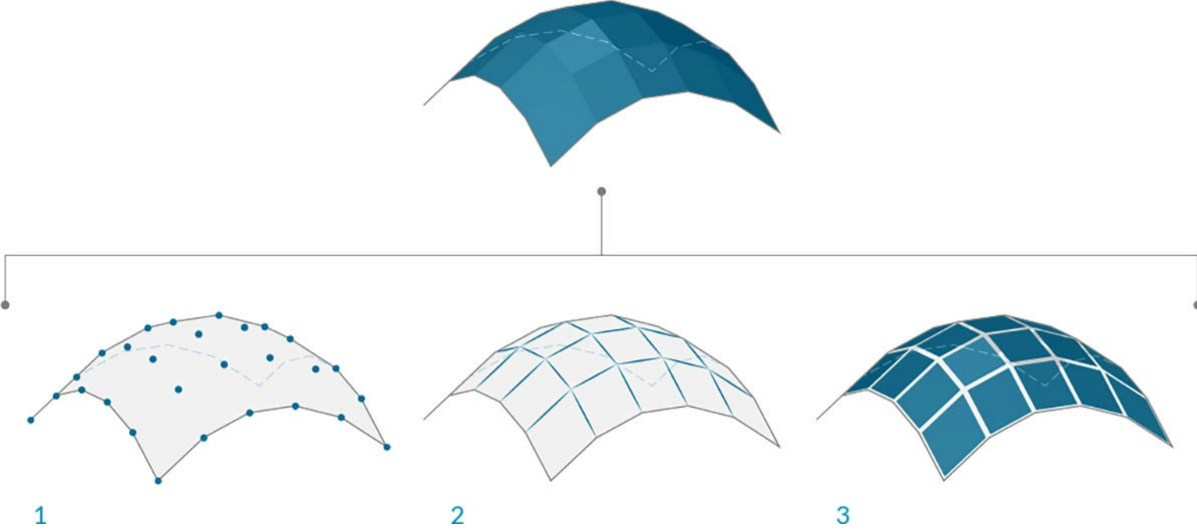
Сети

Сети

В сфере вычислительного моделирования сети представляют собой одну из наиболее распространенных форм представления 3D-геометрии. Геометрия сети может служить более простой и гибкой альтернативой NURBS-поверхностям. Сети используются везде: от визуализации до цифрового производства и 3D-печати.

Что такое сеть

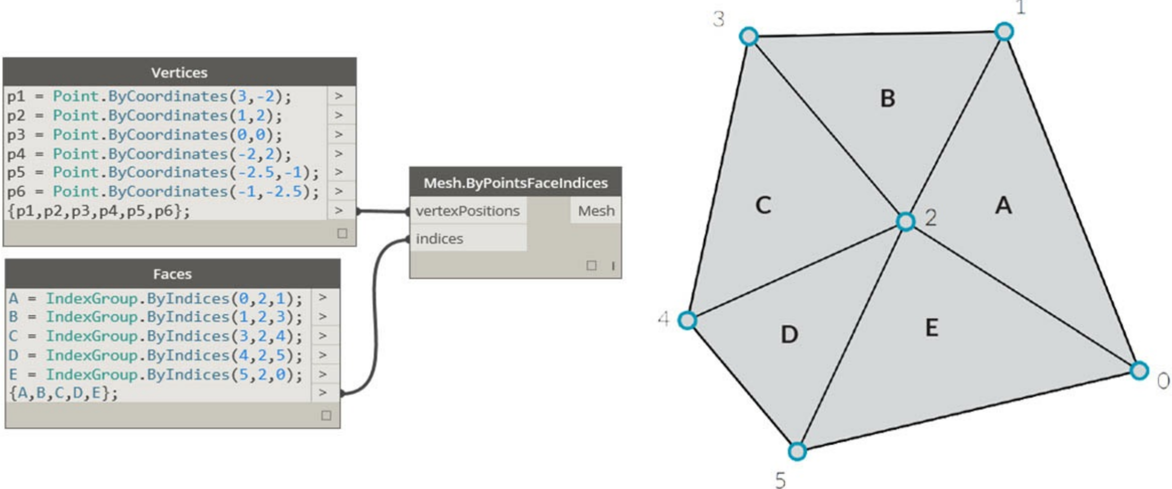
Сеть — это набор четырехугольников и треугольников, образующих геометрию поверхности или тела. Как и в случае с телами, структура объекта-сети включает в себя вершины, ребра и грани. Существуют свойства, которыми обладают только сети, например нормали.



1. Вершины сети
2. Ребра сети (ребра, у которых только одна прилегающая грань, называются открытыми; все остальные ребра являются закрытыми)
3. Грани сети

Элементы сети

Приложение Dynamo определяет сети, используя структуру данных «грань—вершина». На элементарном уровне эта структура представляет собой набор точек, сгруппированных по полигонам. Точки сети называются вершинами, а полигоны, похожие на поверхности, — гранями. Для создания сети требуется список вершин и система, позволяющая группировать эти вершины в грани, называемая группой индексов.

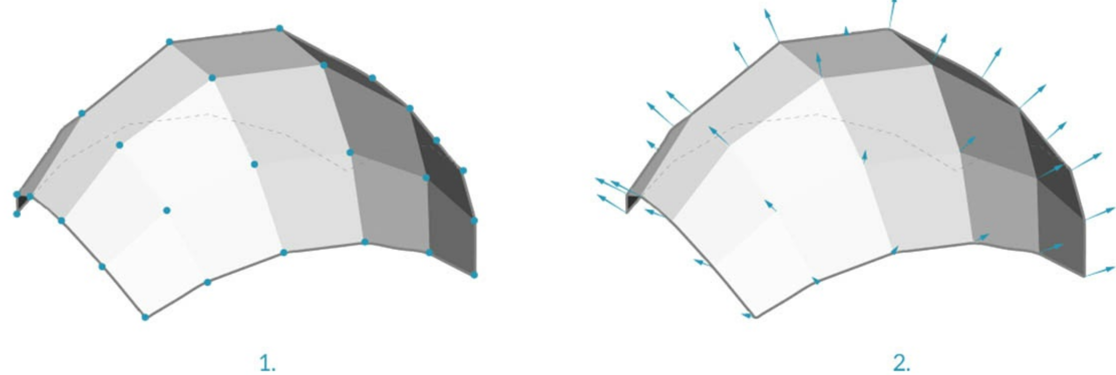


1. Список вершин
2. Список групп индексов для определения граней

Вершины и нормали вершин

Вершины сети представляют собой обычный список точек. Индекс вершин очень важен при создании сети или получении информации о структуре сети. У каждой вершины также есть нормаль (вектор), задающая усредненное направление прилегающих граней и позволяющая

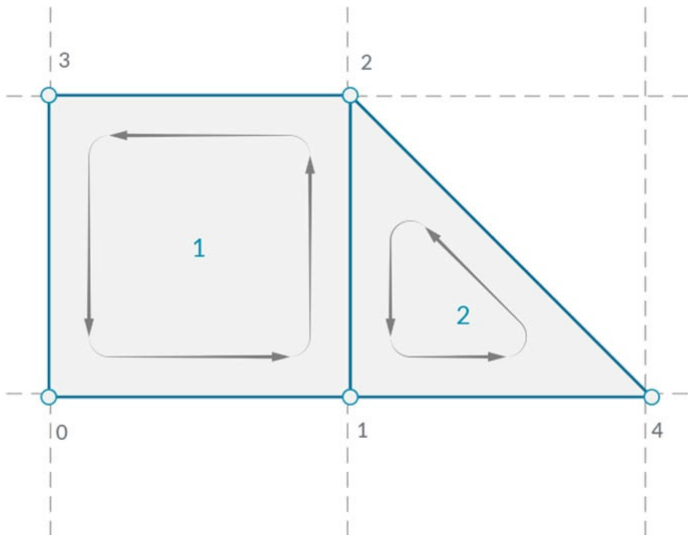
определить ориентацию сети «внутрь» или «наружу».



1. Вершины
2. Нормали вершин

Грани

Грань представляет собой упорядоченный список из трех или четырех вершин. Таким образом, представление грани сети зависит от положения индексируемых вершин. Благодаря наличию списка вершин, образующих сеть, вместо указания отдельных точек для определения грани достаточно использовать индекс вершин. Это также позволяет использовать одну и ту же вершину в нескольких гранях.



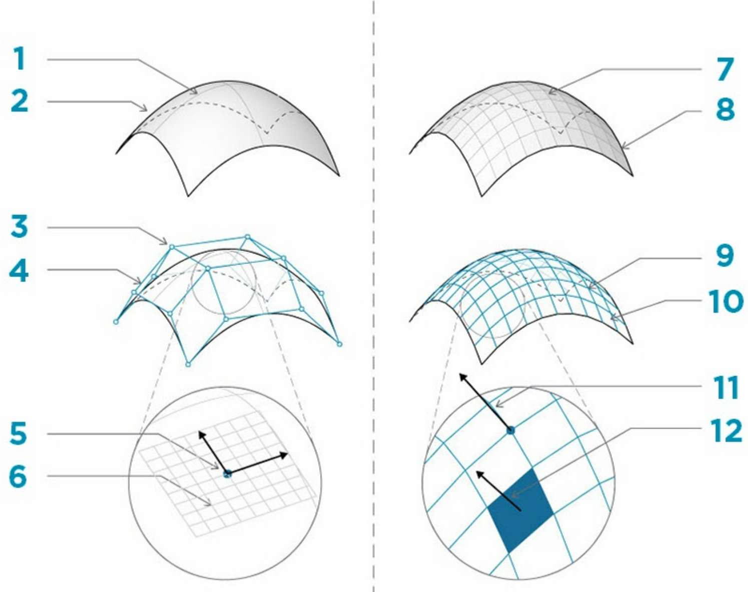
1. Квадратная грань, созданная на основе индексов 0, 1, 2 и 3
2. Треугольная грань, созданная на основе индексов 1, 4 и 2 (обратите внимание, что очередность групп индексов может быть изменена: если очередность следует направлению против часовой стрелки, то грань будет определена правильно)

Сети и NURBS-поверхности

В чем отличие геометрии сети от геометрии NURBS? Когда следует использовать геометрию каждого из этих типов?

Параметризация

В предыдущей главе мы узнали, что NURBS-поверхности определяются набором NURBS-кривых, идущих в двух направлениях. Эти направления обозначаются как U и V и позволяют осуществлять параметризацию NURBS-поверхности в соответствии с областью определения двумерной поверхности. Сами кривые хранятся на компьютере в виде формул, что позволяет рассчитывать итоговые поверхности с произвольно малой степенью точности. Тем не менее скомбинировать несколько NURBS-поверхностей достаточно сложно. Объединение двух NURBS-поверхностей приведет к созданию сложной поверхности, причем различные сегменты геометрии будут иметь различные параметры UV и определения кривых.

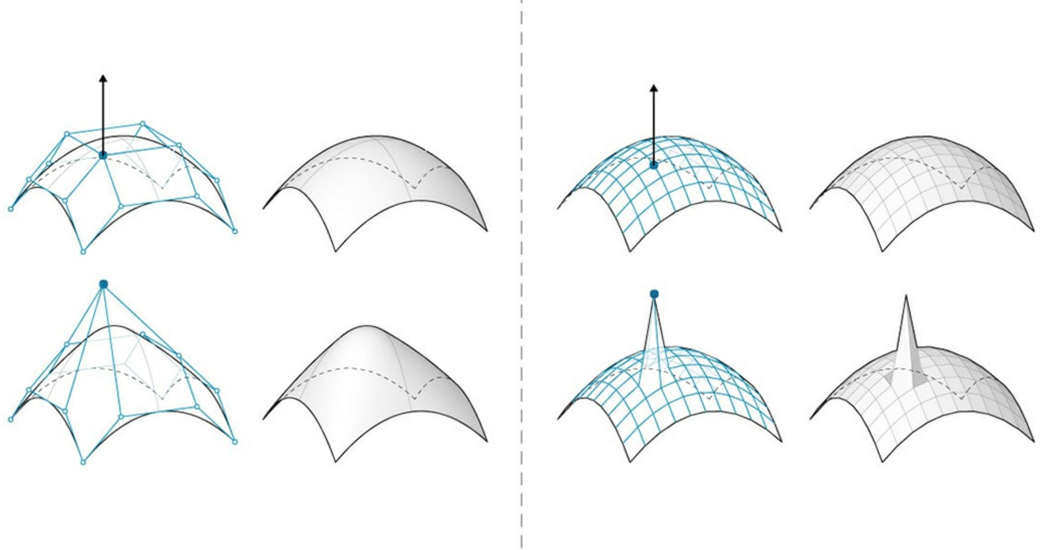


1. Поверхность
2. Изопараметрическая кривая
3. Управляющая точка поверхности
4. Управляющей полигон поверхности
5. Изопараметрическая точка
6. Рамка поверхности
7. Сеть
8. Открытое ребро
9. Сетка сети
10. Ребра сети
11. Нормаль вершины
12. Грань сети/нормаль грани сети

Сети состоят из дискретного количества точно заданных вершин и граней. Сетка вершин, как правило, не может быть определена простыми координатами UV, а так как грани дискретны, степень точности уже встроена в сеть, и ее можно изменить только путем уточнения сети и добавления дополнительных граней. Отсутствие математических описаний у сетей обеспечивает гибкость при работе со сложными геометрическими объектами в пределах одной сети.

Локальное и глобальное влияние

Еще одно важное отличие заключается в том, насколько локальные изменения геометрии сети или геометрии NURBS влияют на всю форму. Перемещение одной вершины сети влияет только на грани, прилегающие к этой вершине. В NURBS-поверхностях механизм влияния более сложен и зависит от степени, а также веса и узлов управляющих точек. Однако в целом при перемещении одной управляющей точки на NURBS-поверхности изменения в геометрии будут более сглаженными и масштабными.



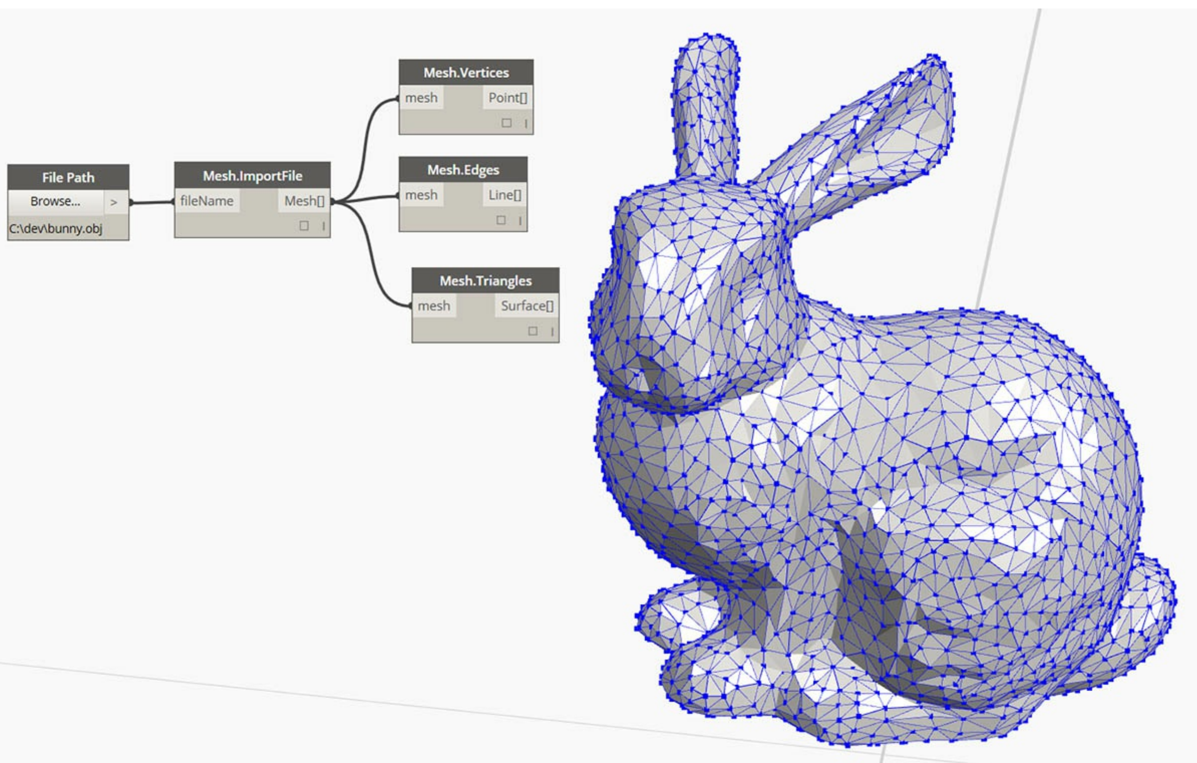
1. NURBS-поверхность: перемещение управляющей точки влияет на всю форму.
2. Геометрия сети: перемещение влияет только на прилегающие элементы.

В качестве аналогии можно сравнить векторное изображение (состоящее из отрезков и кривых) с растровым изображением (состоящим из отдельных пикселей). При увеличении векторного изображения кривые остаются плотными и четкими, а при увеличении растрового изображения увеличивается размер отдельных пикселей. В этом случае NURBS-поверхности можно сравнить с векторным изображением, так как здесь работает плавная математическая связь, а сеть ведет себя так же, как растровое изображение с заданным разрешением.

Mesh Toolkit

Возможности работы с сетями в Dynamo можно расширить за счет установки пакета Mesh Toolkit. Dynamo Mesh Toolkit содержит инструменты для импорта сетей из внешних файлов в других форматах, создания сетей из геометрических объектов Dynamo и построения сетей вручную по вершинам и индексам. В библиотеке также содержатся инструменты для изменения сетей, восстановления сетей или извлечения горизонтальных срезов для использования в производстве.

Пример использования Mesh Toolkit см. в главе 10.2.



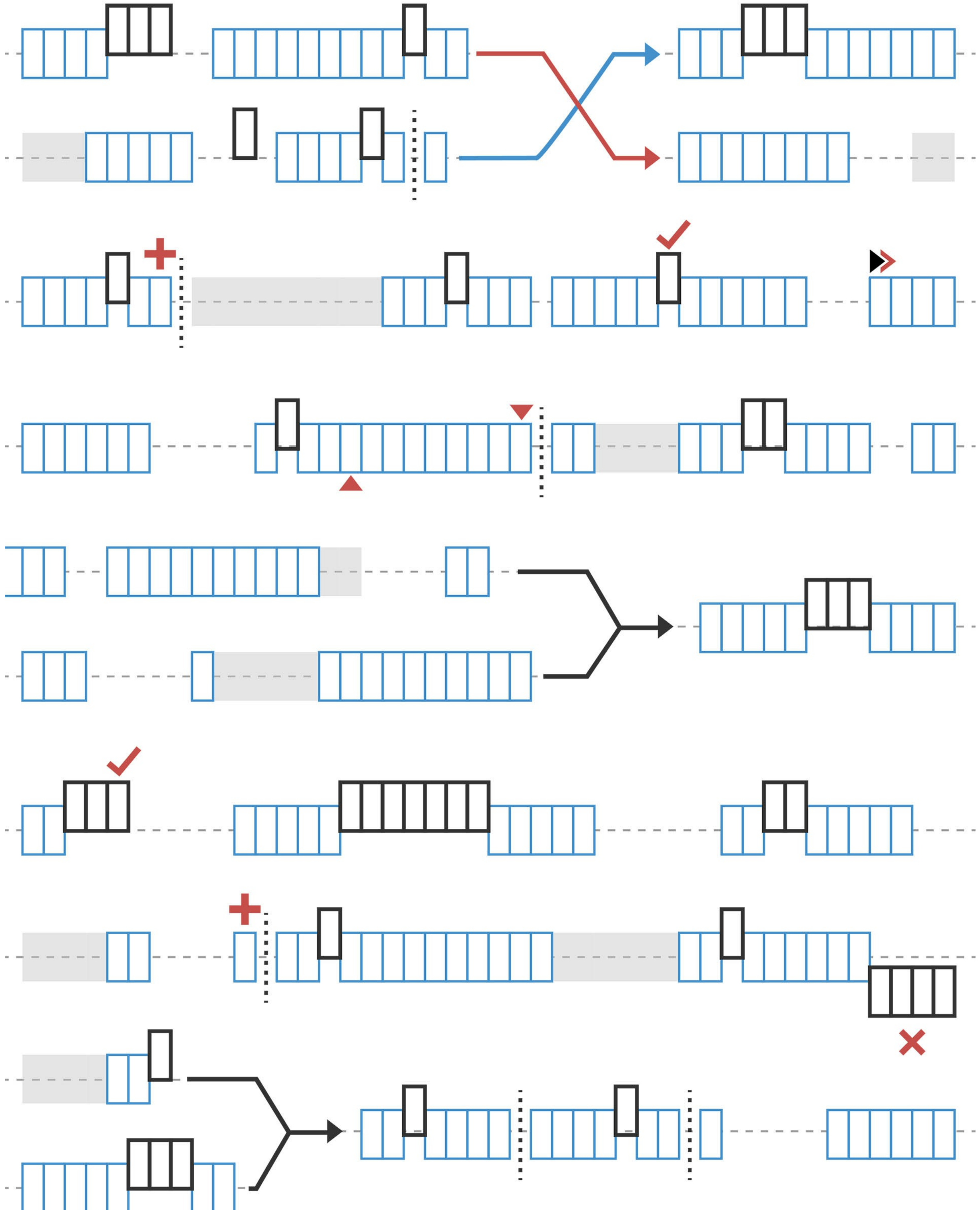
Импорт геометрии

Импорт геометрии

Проектирование на основе списков

Проектирование на основе списков

Списки помогают упорядочивать данные. В операционной системе компьютера существуют файлы и папки. Приложение Dynamo устроено аналогично, только роль файлов в нем играют элементы, а папок — списки. Как и в операционной системе, в Dynamo существует множество способов создания, изменения и запроса данных. В этой главе рассказывается о том, как управлять списками в Dynamo.



Что такое список

Что такое список

Список — это набор элементов или компонентов. Возьмем, к примеру, связку бананов. Каждый банан является элементом списка (или связки). Проще взять в руки связку бананов, чем брать бананы по одному. Точно так же работать с элементами, сгруппированными по параметрическим связям в структуре данных, проще, чем с отдельными элементами.

Фотография предоставлена Августусом Бину (Augustus Binu).

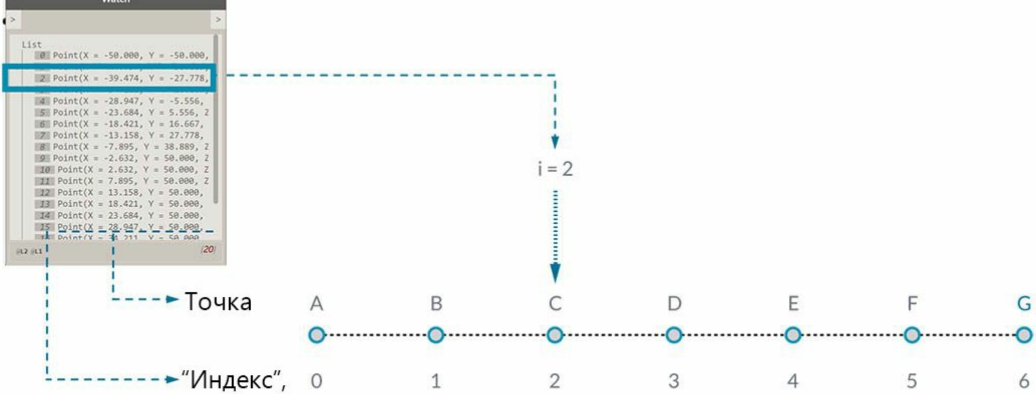
Когда мы идем в магазин, мы кладем все, что купили, в пакет. Этот пакет также является списком. Мы хотим испечь банановый кекс, и нам нужно три связки бананов (мы печем *очень большой* кекс). Пакет представляет собой список связок, а каждая связка представляет собой список бананов. Пакет — это список списков (двумерный), а банан — это просто список (одномерный).

В Dynamo данные в списках упорядочиваются, и первому элементу в каждом списке присваивается индекс 0. Ниже мы рассмотрим то, как задать список в Dynamo, а также то, как разные списки могут быть связаны друг с другом.

Индексы, отсчитываемые от нуля

Первому элементу в списке всегда назначается индекс 0, а не 1, и поначалу это может показаться странным. Поэтому запомните, что, если речь идет о первом элементе в списке, подразумевается элемент с индексом 0.

Например, если бы вам потребовалось посчитать количество пальцев на правой руке, то вы бы начали считать с 1 до 5. Однако если бы вам потребовалось внести ваши пальцы в список, то приложение Dynamo назначило бы им индексы от 0 до 4. Это может показаться странным тем, кто только начинает заниматься программированием, однако индекс, отсчитываемые от нуля, является стандартным для большинства вычислительных систем.



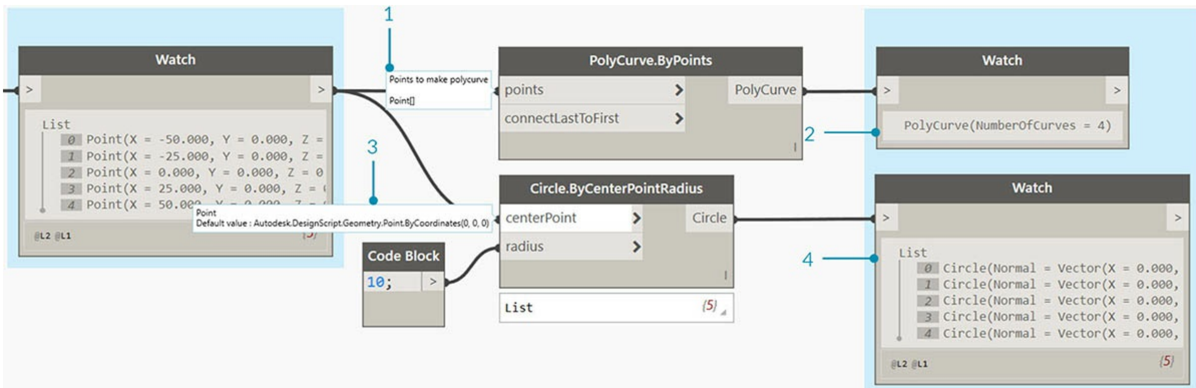
Обратите внимание, что такой список по-прежнему включает пять элементов, просто в нем используется система отсчета от нуля, а не от единицы. Элементы списка не обязательно должны быть числами. Это могут быть данные любого типа, который поддерживается Dynamo, например точки, кривые, поверхности, семейства и т. д.

Зачастую самым простым способом узнать тип данных в списке является подключение узла Watch к порту вывода другого узла. По умолчанию узел Watch автоматически отображает все индексы в левой части списка, а элементы данных — в правой.

Эти индексы играют ключевую роль при работе со списками.

Входные и выходные данные

При работе со списками требуемые входные и выходные данные различаются в зависимости от используемого узла Dynamo. Для примера возьмем список из пяти точек и соединим его порт вывода с двумя разными узлами Dynamo: *PolyCurve.ByPoints* и *Circle.ByCenterPointRadius*:



1. Порт ввода *points* узла *PolyCurve.ByPoints* выполняет поиск элемента *Point[]*. Этот элемент представляет собой список точек.
2. Выходные данные узла *PolyCurve.ByPoints* — это элемент PolyCurve, созданный на основе списка пяти точек.
3. Порт ввода *centerPoint* узла *Circle.ByCenterPointRadius* запрашивает элемент *Point*.
4. Выходные данные *Circle.ByCenterPointRadius* представляют собой список из пяти окружностей, центры которых соответствуют точкам из исходного списка.

Узлы *PolyCurve.ByPoints* и *Circle.ByCenterPointRadius* используют одни и те же входные данные, однако узел Polycurve на выходе дает одну сложную кривую, а узел Circle — пять окружностей для каждой точки из списка. На интуитивном уровне это кажется понятным, так как сложная кривая строится путем соединения всех пяти точек, а при создании окружностей каждая точка используется в качества центра отдельной окружности. Что же происходит с данными?

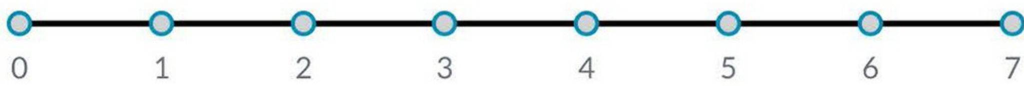
При наведении указателя мыши на порт ввода *points* узла *Polycurve.ByPoints* можно увидеть, что этому порту требуется элемент *Point[]*. Обратите внимание на скобки в конце. Этот элемент представляет список точек, и чтобы создать сложную кривую, в качестве входных данных этому узлу требуется отдельный список точек для каждой кривой. В результате узел объединяет каждый полученный на вводе список в одну сложную кривую.

Порт ввода *centrePoint* узла *Circle.ByCenterPointRadius* запрашивает элемент *Point*. Этому узлу требуется одна точка, являющаяся отдельным элементом, которую он будет использовать в качестве центра окружности. Поэтому на основе тех же входных данных мы получаем пять отдельных окружностей. Знание различий в использовании входных данных в Dynamo помогает лучше понимать, как узлы распоряжаются данными.

Переплетение

Сопоставление данных является проблемой, для которой не существует четкого решения. Это происходит, когда узел получается доступ к входным данных разных размеров. Изменение алгоритма сопоставления данных может привести к существенным различиям в результатах.

Рассмотрим в качестве примера узел, который создает линейные сегменты между точками (Line.ByStartPointEndPoint). У него два входных параметра, которые используются для представления координат точек:

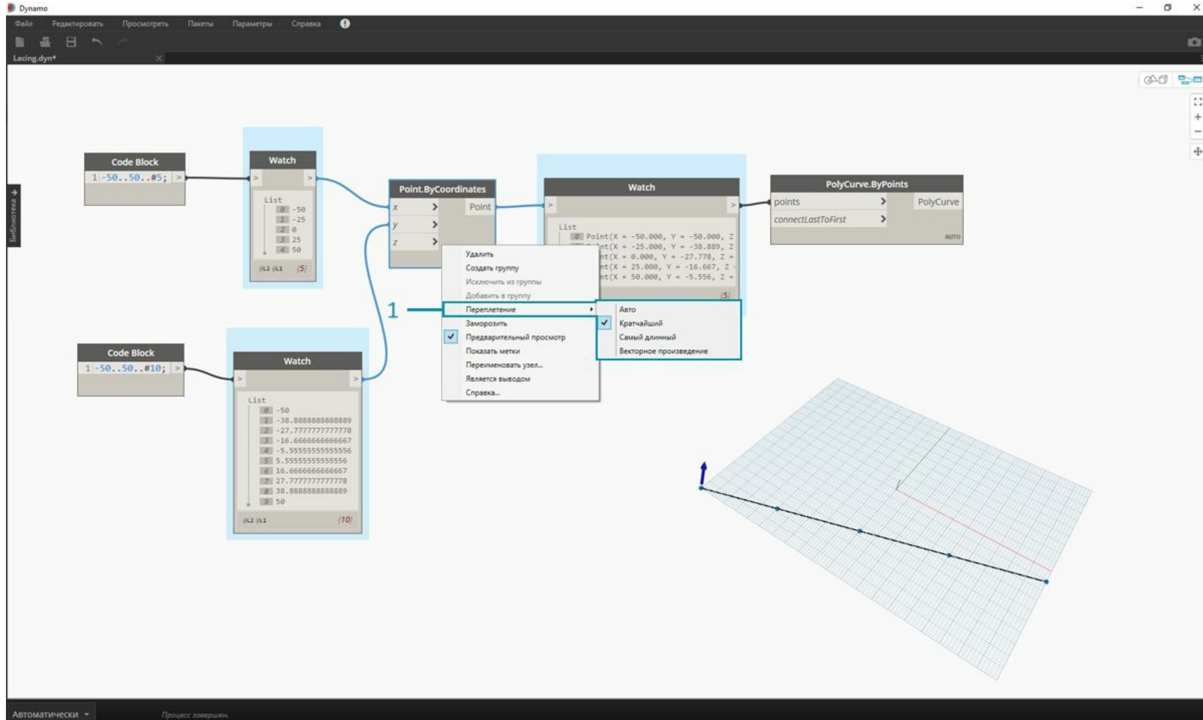


Как мы видим, прочертить линию через эти наборы точек можно разными способами. Параметры переплетения можно просмотреть, щелкнув центр узла правой кнопкой мыши и выбрав меню «Переплетение».

Базовый файл

> Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): Lacing.dyn. Полный список файлов примеров можно найти в приложении.

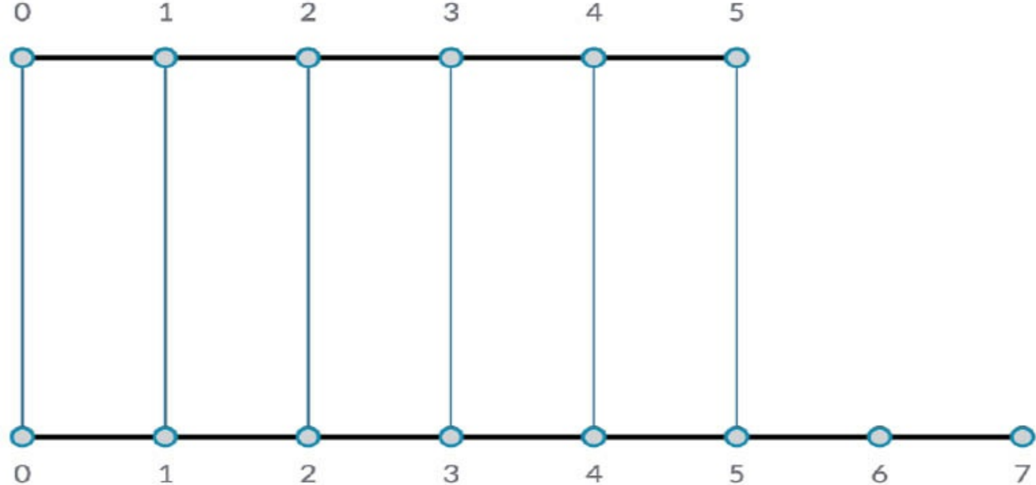
Для изучения операций переплетения ниже мы воспользуемся этим базовым файлом, чтобы определить самый короткий и длинный списки, а также декартово произведение.

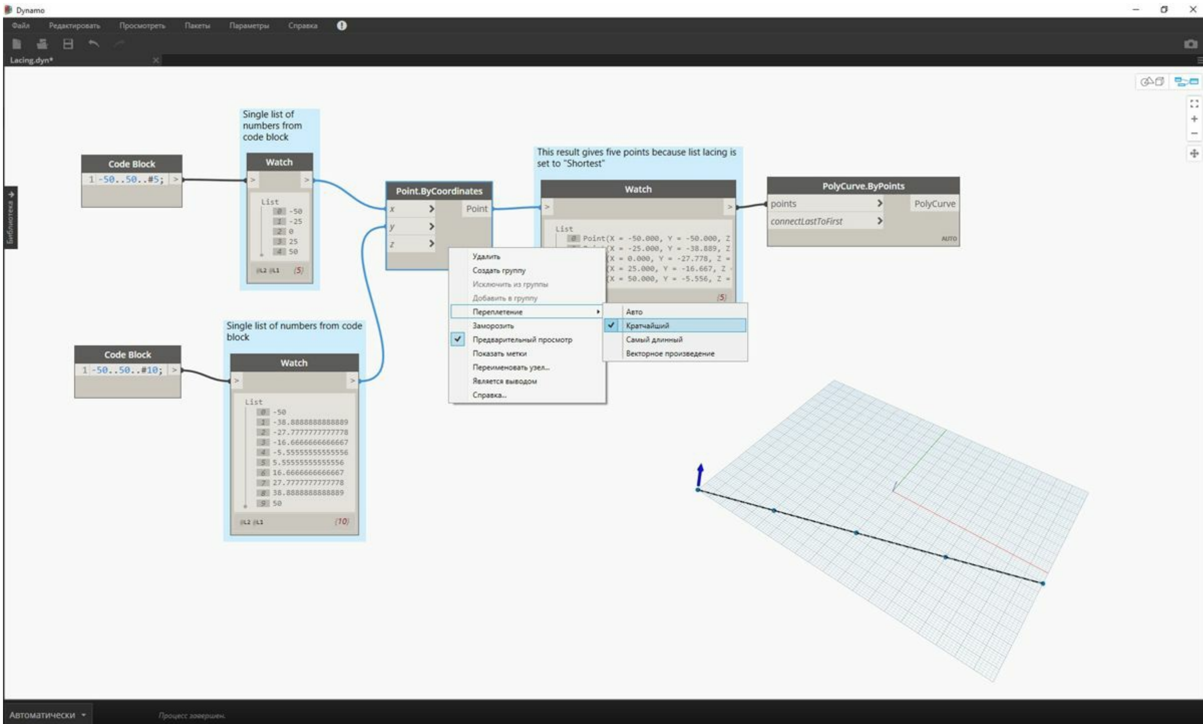


1. Измените настройку переплетения узла *Point.ByCoordinates* в графике выше, оставив остальные элементы без изменений.

Самый короткий список

Самый простой способ — попарно соединять входные данные с одинаковыми индексами, пока один из списков не закончится. Это алгоритм по самому короткому списку. Узлы Dynamo используют этот алгоритм по умолчанию.

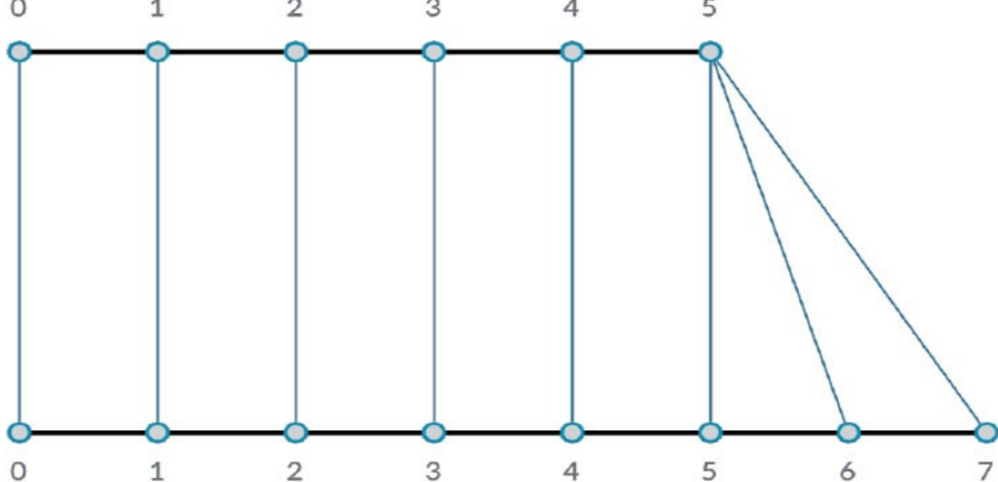


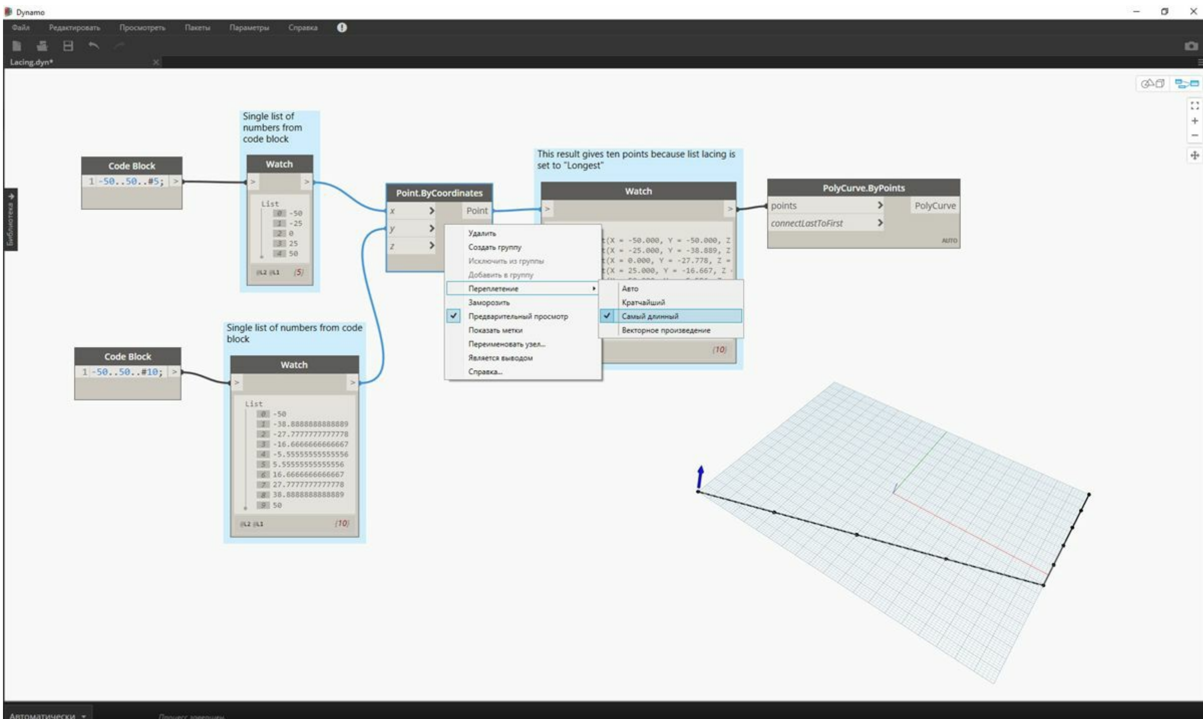


Изменив способ переплетения на *Декартово произведение*, вы получите базовую диагональную линию из пяти точек. Пять точек — это длина наименьшего списка. Таким образом, переплетение по самому короткому списку прекращается по достижении конца этого списка.

Самый длинный список

Алгоритм переплетения по самому длинному списку соединяет все входные элементы, используя некоторые элементы повторно, пока не закончатся оба списка:

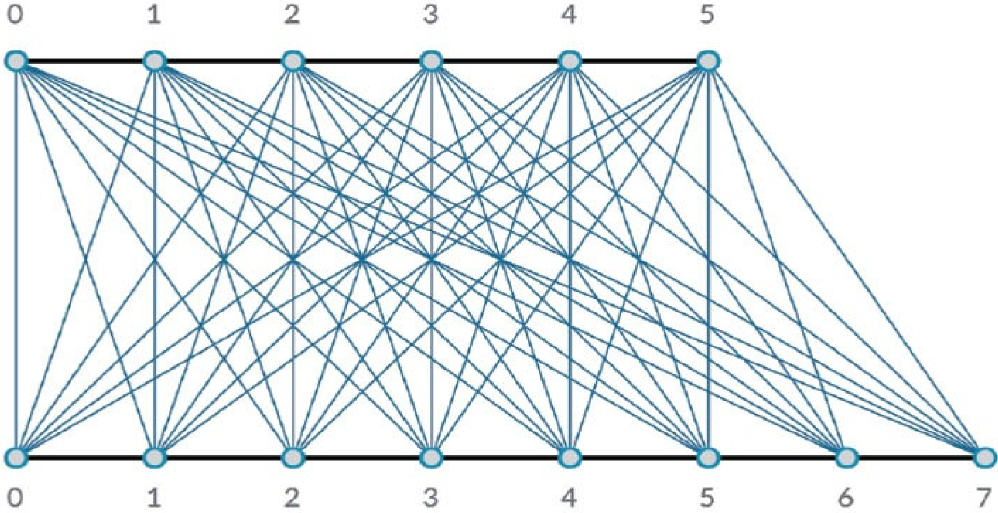


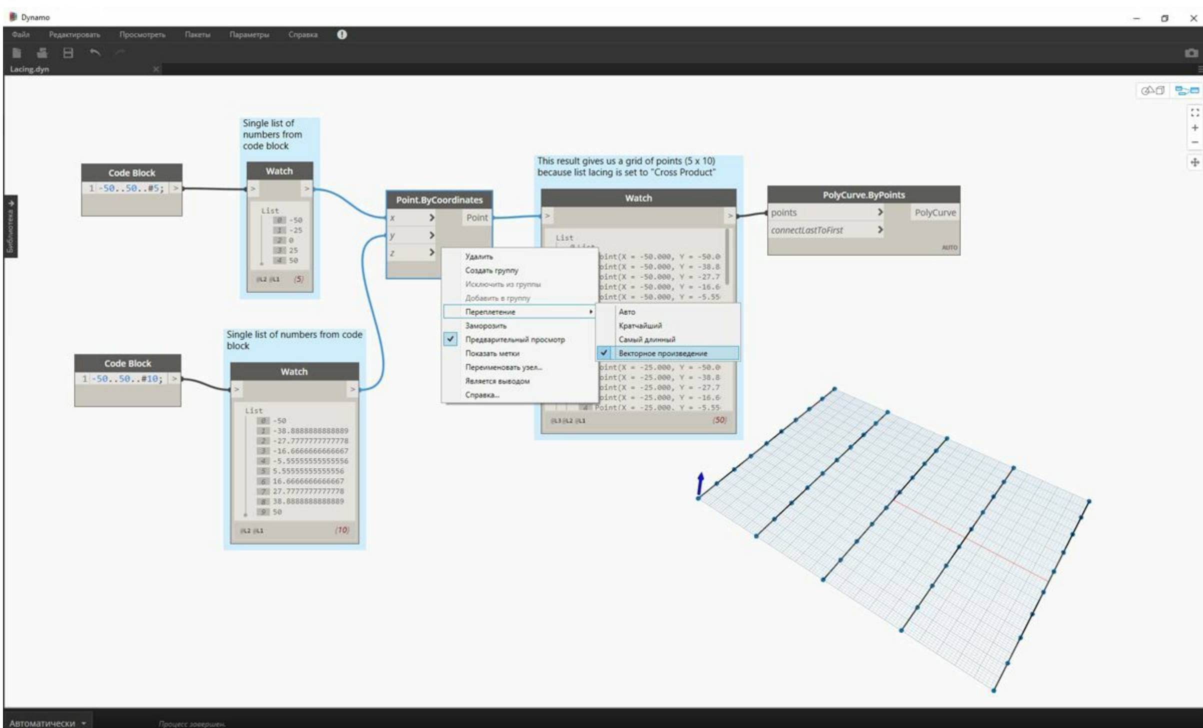


При изменении способа переплетения на *Самый длинный* вы получите диагональную линию, которая имеет продолжение по вертикали. Аналогично схематическому изображению выше, пятый элемент короткого списка используется повторно, пока не будет достигнут конец более длинного списка.

Декартово произведение

Наконец, при использовании метода «Декартово произведение» создаются все возможные соединения:





Изменив способ переплетения на *Декартово произведение*, вы получите все возможные соединения между списками, в результате чего создается сетка 5 x 10 точек. Эта структура данных эквивалентна декартову произведению, показанному в схематическом изображении выше, однако данные теперь являются списком списков. Путем соединения сложной кривой можно увидеть, что каждый список определяется значением X, в результате чего образуется ряд вертикальных линий.

Работа со списками

Работа со списками

Определившись с тем, что такое список, поговорим о том, какие операции можно выполнять с ним. Представим список в виде колоды карт. Колода — это список, а каждая карта — элемент.

Фото предоставлено Кристианом Гидлефом (Christian Gidlöf)

Какие **запросы** доступны в списке? Это возможность вызова существующих свойств.

- Сколько карт в колоде? 52.
- Количество мастей? 4.
- Из какого материала они изготовлены? Бумага.
- Какова их длина? 3,5 дюйма, или 89 мм.
- Какова их ширина? 2,5 дюйма, или 64 мм.

Какие **действия** можно выполнять со списком? Это изменения списка в зависимости от конкретной операции.

- Колоду можно перемешать.
- Колоду можно отсортировать по значению.
- Колоду можно отсортировать по масти.
- Колоду можно разделить.
- Колоду можно раздать отдельным игрокам.
- Можно выбрать отдельную карту из колоды.

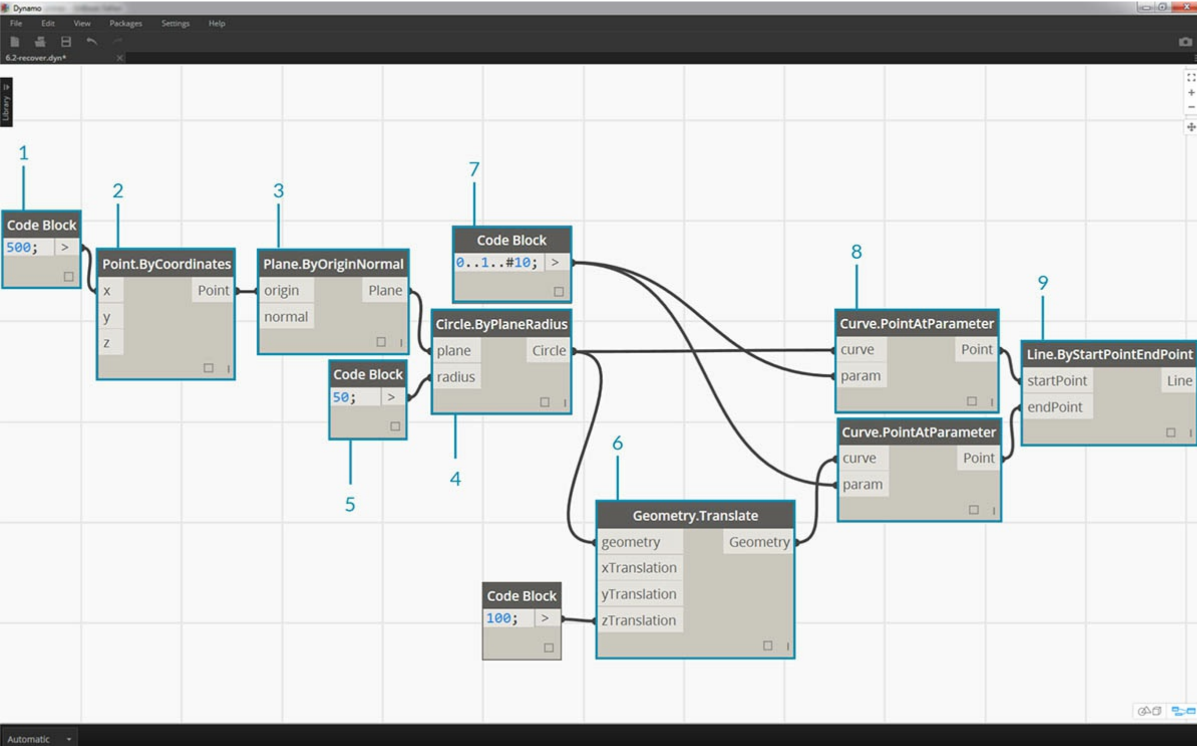
У всех перечисленных выше операций есть аналогичные узлы Dynamo для работы со списками типовых данных. В уроке ниже будут рассмотрены основные операции, которые можно выполнять со списками.

Операции со списками

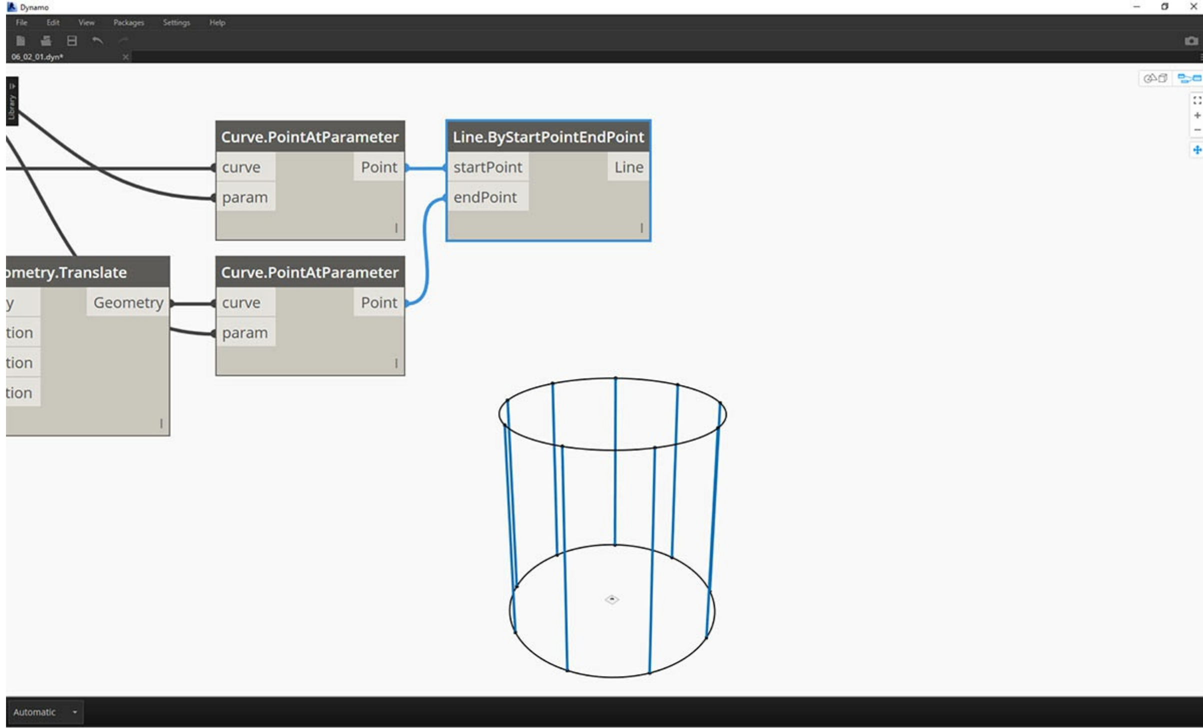
На изображении показан исходный график, который будет использоваться для иллюстрации основных операций со списками. Мы рассмотрим управление данными в списке и представим наглядные результаты.

Упражнение «Операции со списком»

Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): List-Operations.dyn. Полный список файлов примеров можно найти в приложении.

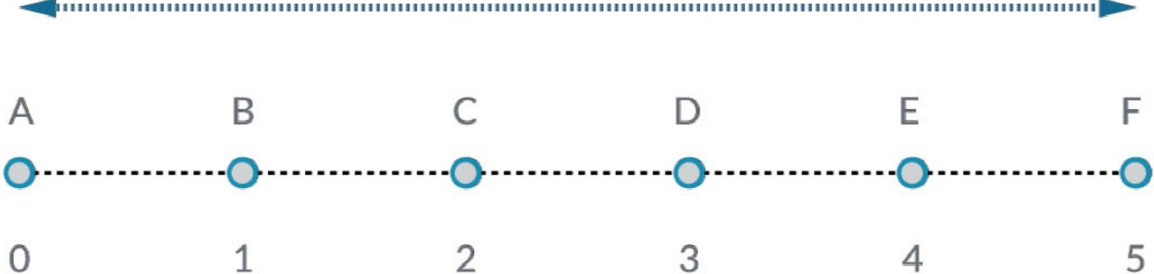


1. Начнем с *блока кода* со значением `500;`
2. Соединим его с входным параметром *x* узла *Point.ByCoordinates*.
3. Соединим узел из предыдущего шага с входным параметром origin узла *Plane.ByOriginNormal*.
4. Соединим узел из предыдущего шага с входным параметром plane узла *Circle.ByPlaneRadius*
5. С помощью *блока кода* укажем `50;` в качестве значения для *radius*. Это будет первая окружность.
6. С помощью узла *Geometry.Translate* переместим окружность вверх на 100 единиц в направлении Z.
7. С помощью узла *Code Block* зададим диапазон из десяти чисел от 0 до 1, используя строку кода `0..1.#10;`
8. Соединим блок кода из предыдущего шага с входным значением *param* двух узлов *Curve.PointAtParameter*. Соединим узел *Circle.ByPlaneRadius* с входным параметром curve верхнего узла, а узел *Geometry.Translate* с входным параметром curve узла под ним.
9. С помощью узла *Line.ByStartPointEndPoint* соединим два узла *Curve.PointAtParameter*.



1. Узел *Watch3D* показывает результаты операции *Line.ByStartPointEndPoint*. Итак, были созданы линии между двумя окружностями. Данный график Dynamo будет использован для демонстрации действий со списками далее.

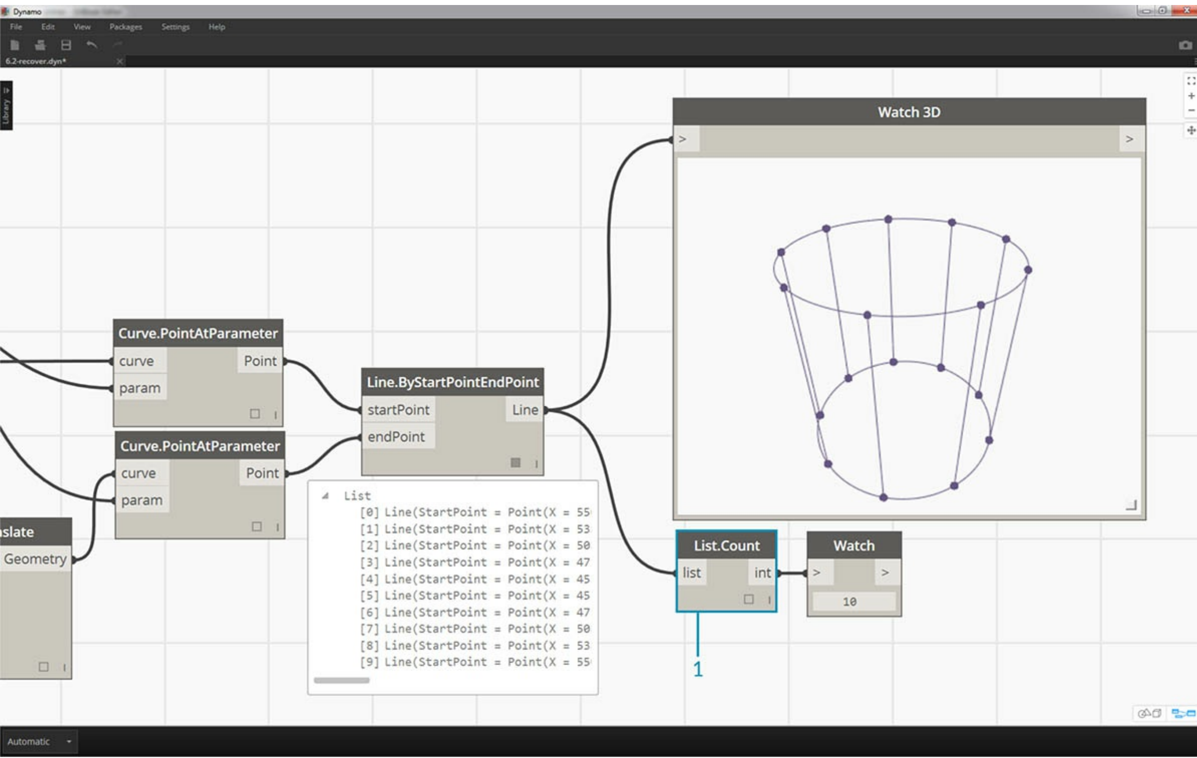
List.Count



Узел *List.Count* сравнительно прост: он подсчитывает количество значений в списке и возвращает это число. При работе со списками списков в использовании этого узла появляются дополнительные нюансы. О них мы поговорим в следующих разделах.

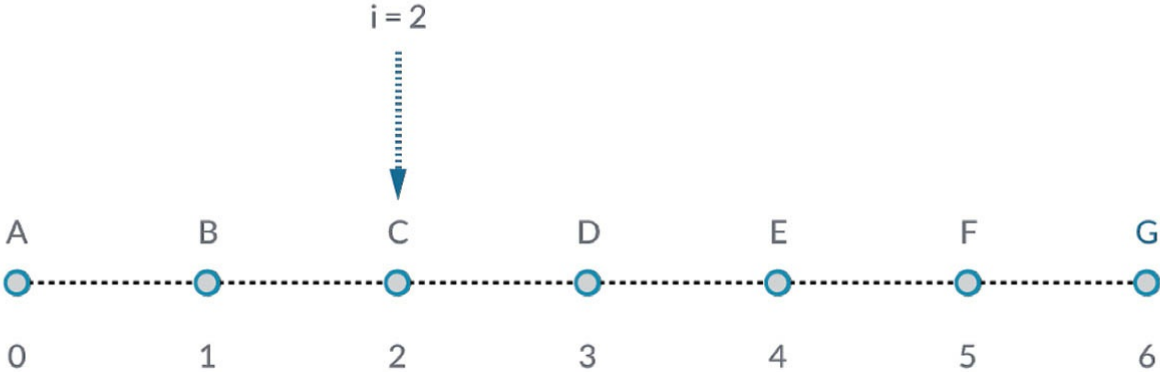
Упражнение List.Count

Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): List-Count.dyn. Полный список файлов примеров можно найти в приложении.



1. Узел *List.Count* возвращает количество линий в узле *Line.ByStartPointEndPoint*. В данном случае значение равно 10, что соответствует количеству точек, созданных с помощью исходного узла *Code Block*.

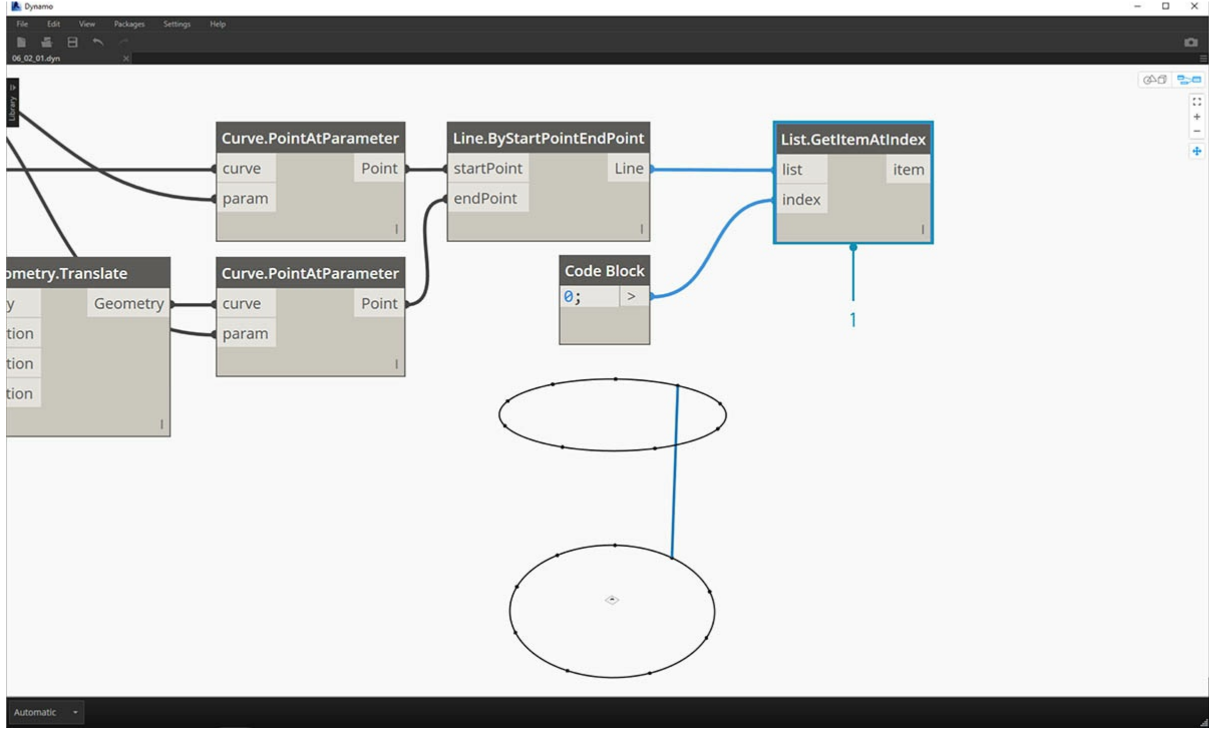
List.GetItemAtIndex



List.GetItemAtIndex — основной способ запроса элементов в списке. На изображении выше для запроса точки с меткой *C* используется индекс *2*.

Упражнение List.GetItemAtIndex

Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): [List-GetItemAtIndex.dyn](). Полный список файлов примеров можно найти в приложении.



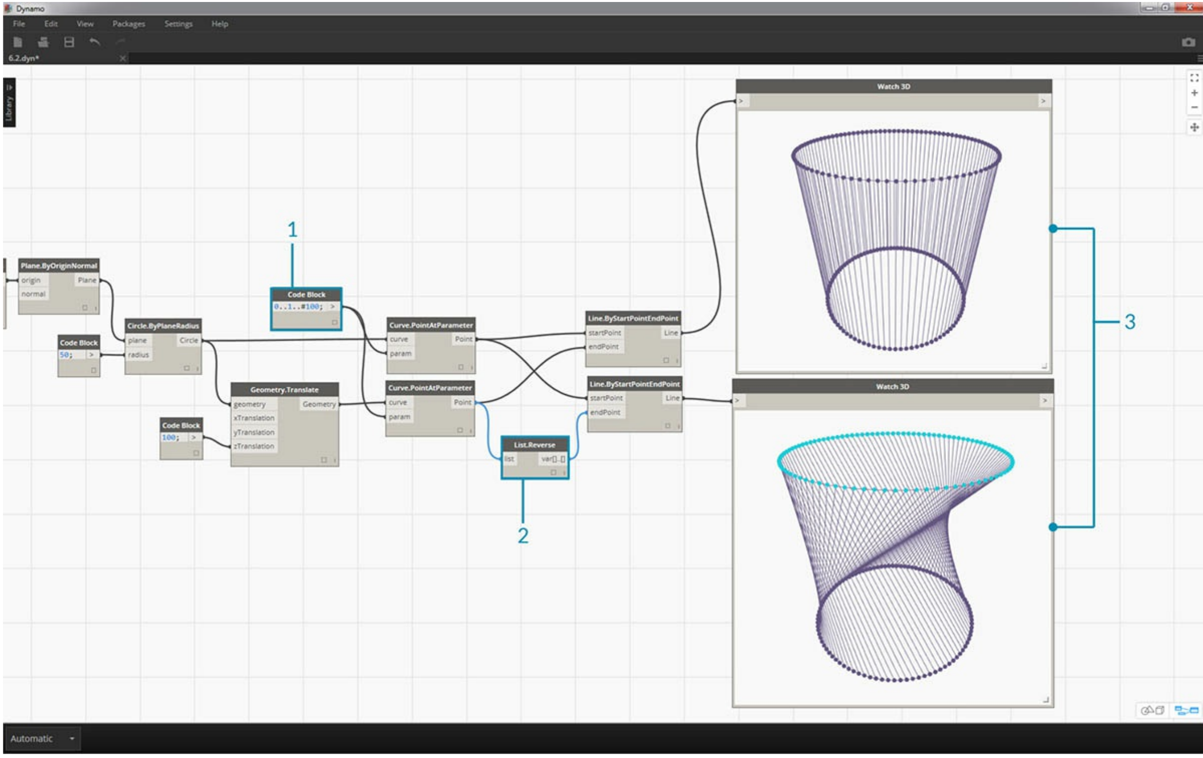
1. С помощью узла *List.GetItemAtIndex* выбираем индекс *0* или первый элемент в списке линий.
2. Узел *Watch3D* показывает, что выбрана одна линия. Примечание. Чтобы видеть это изображение, отключите предварительный просмотр узла *Line.ByStartPointEndPoint*.

List.Reverse

Узел *List.Reverse* располагает все элементы в списке в обратном порядке.

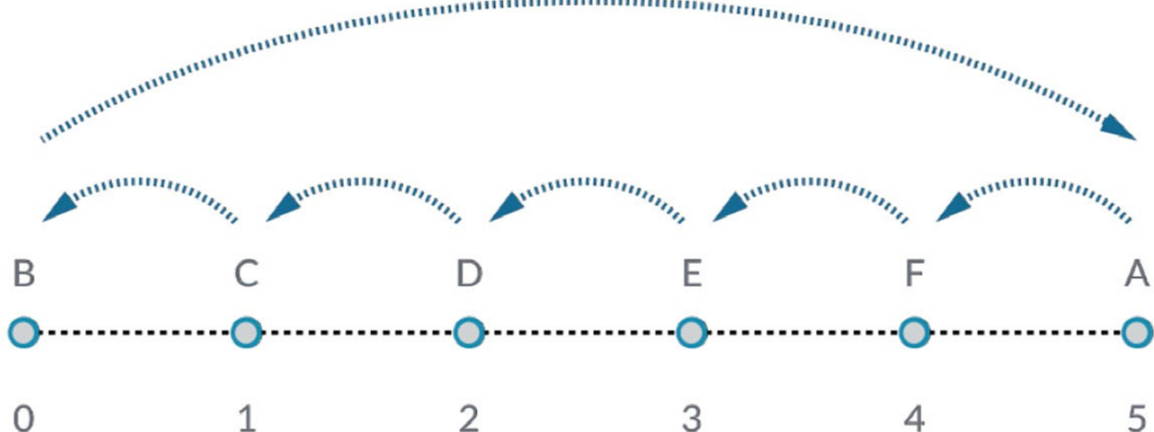
Упражнение List.Reverse

Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): List-Reverse.dyn. Полный список файлов примеров можно найти в приложении.



1. Для правильной визуализации обращенного списка линий создайте дополнительные линии, задав в блоке кода новое значение `0..1..#100;`.
2. Вставьте узел *List.Reverse* между узлами *Curve.PointAtParameter* и *Line.ByStartPointEndPoint* для одного из списков точек.
3. Узлы *Watch3D* показывают два различных результата. Первый узел показывает результат без обращенного списка. Линии соединяются вертикально с точками напротив. Второй узел показывает результат обращения списка, где все точки соединяются с точками напротив в обратном порядке.

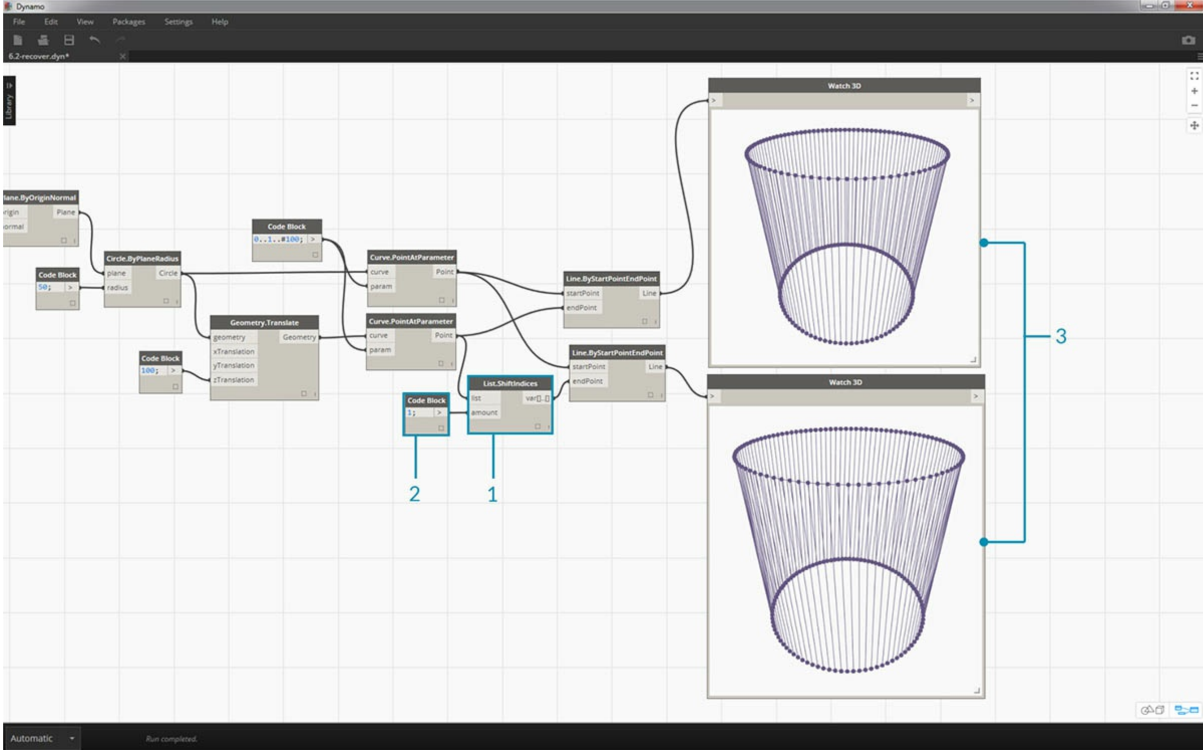
List.ShiftIndices



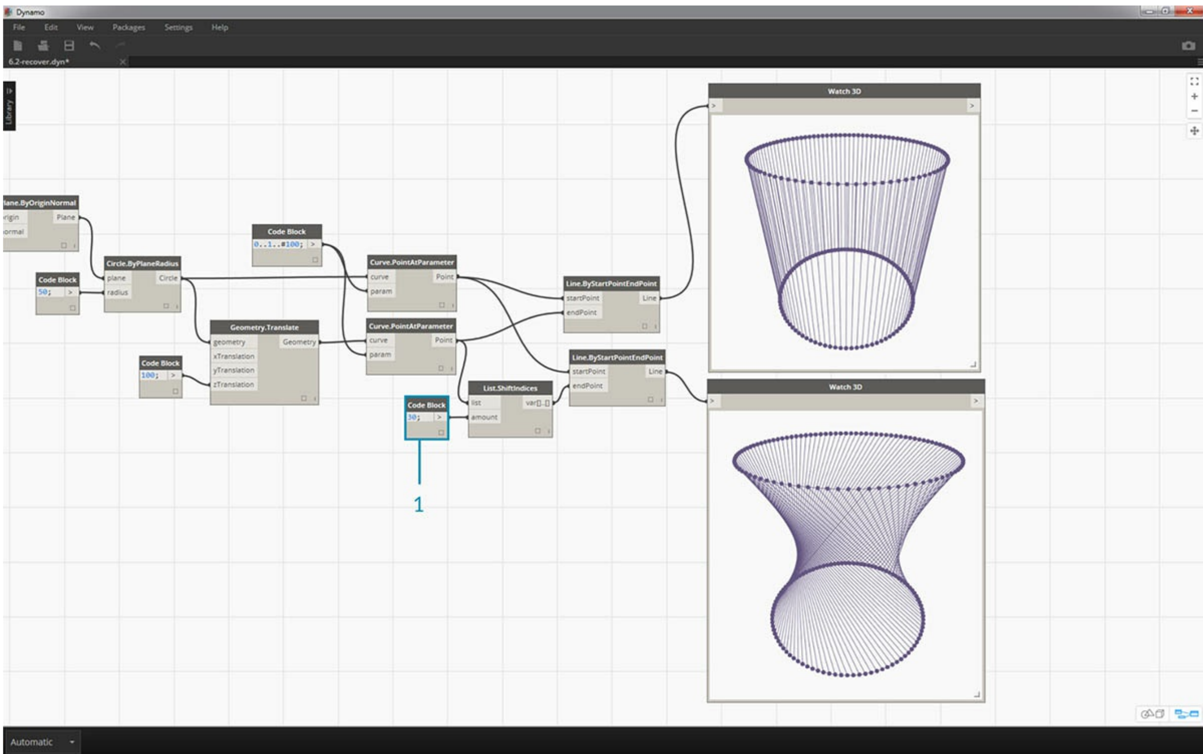
List.ShiftIndices — это удобный инструмент для создания скручиваний или спиралей и других подобных манипуляций с данными. Этот узел смещает элементы в списке на заданное количество индексов.

Упражнение List.ShiftIndices

Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): List-ShiftIndices.dyn. Полный список файлов примеров можно найти в приложении.



1. В том же сценарии, где был создан обращенный список, вставьте узел *List.ShiftIndices* между узлами *Curve.PointAtParameter* и *Line.ByStartPointEndPoint*.
2. С помощью узла *Code Block* укажите значение*1* для сдвига списка на один индекс.
3. Изменение незначительное, но все линии в нижнем узле *Watch3D* сместились на один индекс при соединении с другим набором точек.



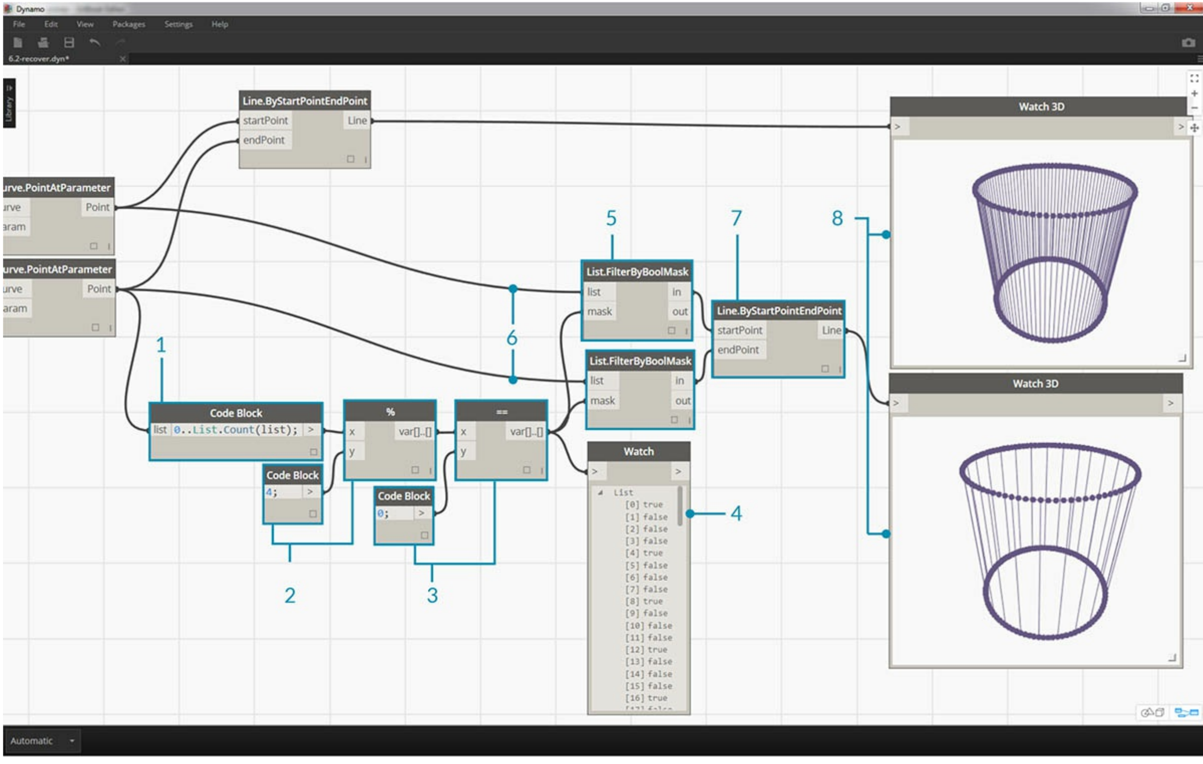
1. Если увеличить значение в узле *Block Code*, например, до *30*, в диагональных линиях появляется существенное различие. В данном случае сдвиг работает аналогично диафрагме камеры, закручивая исходную цилиндрическую форму.

List.FilterByBooleanMask

Узел *List.FilterByBooleanMask* удаляет определенные элементы на основе списка логических операций или значений «Истина»/«Ложь».

Упражнение List.FilterByBooleanMask

Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): List-FilterByBooleanMask.dyn. Полный список файлов примеров можно найти в приложении.



Чтобы создать список значений «Истина» или «Ложь», необходимо выполнить несколько дополнительных действий.

1. С помощью узла *Code Block* зададим выражение с синтаксисом `0..List.Count(list);`. Соединим узел *Curve.PointAtParameter* с входным параметром *list*. Этот процесс будет рассмотрен подробнее в главе о блоках кода, но в данном случае строка кода дает список, где представлены все индексы узла *Curve.PointAtParameter*.
2. С помощью узла *%* (коэффициент) соединим выходной параметр узла *Code Block* с входным параметром *x*, а значение *4* с входным параметром *y*. Это позволит вычислить остаток при делении списка индексов на 4. Узел «Коэффициент» очень полезен при создании массивов. Все значения будут представлять собой возможный остаток от 4: 0, 1, 2, 3.
3. Благодаря узлу *коэффициент* мы знаем, что значение 0 означает делимость индекса на 4 (0, 4, 8 и т. д.). С помощью узла == можно проверить делимость по значению *0*.
4. Узел *Watch* выводит лишь следующий результат: массив истинных и ложных значений в виде *true,false,false,false*.....
5. Соедините этот массив с входным параметром mask обоих узлов *List.FilterByBooleanMask*.
6. Соедините узел *Curve.PointAtParameter* с входными параметрами list узлов *List.FilterByBooleanMask*.
7. Выходными данными *Filter.ByBooleanMask* будут*in* и *out*. *In* — это значения, которым было присвоено значение маски *true*, а *out* — значения, которым было присвоено значение *false*. Соедините выходные параметры *in* с входными параметрами *startPoint* и *endPoint* узла *Line.ByStartPointEndPoint*, создав тем самым отфильтрованный список линий.
8. Узел *Watch3D* показывает, что количество линий меньше, чем количество точек. Отфильтровав только истинные значения, мы выбрали 25 % узлов.

Списки списков

Списки списков

Добавим еще один уровень в иерархию. Если взять колоду карт из первого примера и создать рамку, в которой будет находиться несколько колод, то эта рамка будет представлять собой список колод, а каждая колода — список карт. Это и есть список списков. В качестве аналогичного примера для этого раздела в красной рамке ниже представлен список столбиков монет, каждый из которых содержит список монет.

Фото предоставлено Dori.

Какие **запросы** доступны в таком списке списков? Таким образом можно вызвать существующие свойства.

- Сколько всего типов монет? 2.
- Какова ценность типов монет? 0,01 долл. США и 0,25 долл. США.
- Какие материалы используются для изготовления монет номиналом 0,25 долл. США? 75 % меди и 25 % никеля.
- Какие материалы используются для изготовления цента? 97,5 % цинка и 2,5 % меди.

Какие **действия** можно выполнять со списком списков? Они приведут к изменению списка списков в зависимости от конкретной операции.

- Выбрать один столбик из монет номиналом 1 или 25 центов.
- Выбрать одну монету номиналом 1 или 25 центов.
- Переупорядочить столбики.
- Перемешать столбики.

Для каждой из перечисленных выше операций в Dynamo имеется отдельный узел. Поскольку мы работаем с абстрактными данными, а не с физическими объектами, необходимо установить набор правил, определяющих порядок перемещения вверх и вниз по иерархии данных.

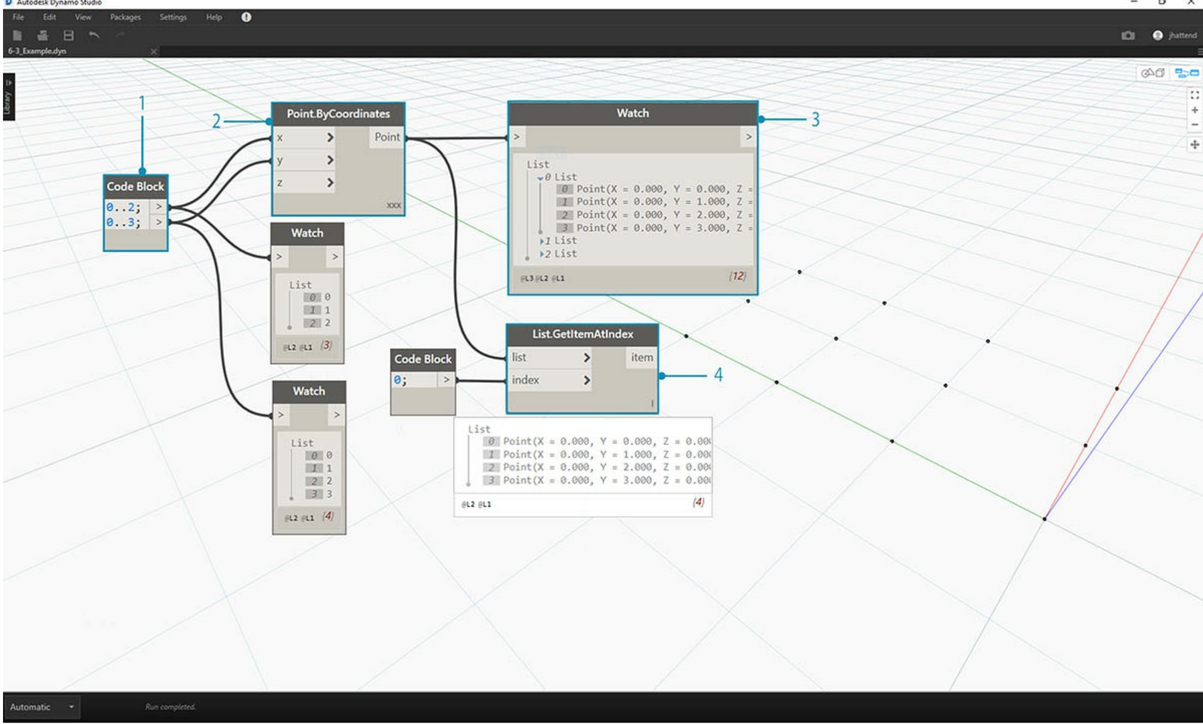
При работе со списками списков данные располагаются по слоям и имеют сложную структуру, но это дает возможность выполнять ряд уникальных параметрических операций. Остановимся подробнее на основных операциях, оставив другие для последующих занятий.

Нисходящая иерархия

В данном разделе необходимо усвоить один базовый принцип: **Dynamo рассматривает списки как объекты самих себя, расположенные в самих себе**. Эта нисходящая иерархия разработана с учетом объектно-ориентированного программирования. Вместо выбора вложенных элементов с помощью команды вроде List.GetItemAtIndex в Dynamo будет выбран индекс основного списка в структуре данных. Этот объект, в свою очередь, может быть другим списком. Рассмотрим этот вопрос подробнее на примере изображения ниже.

Упражнение «Нисходящая иерархия»

Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): Top-Down-Hierarchy.dyn. Полный список файлов примеров можно найти в приложении.



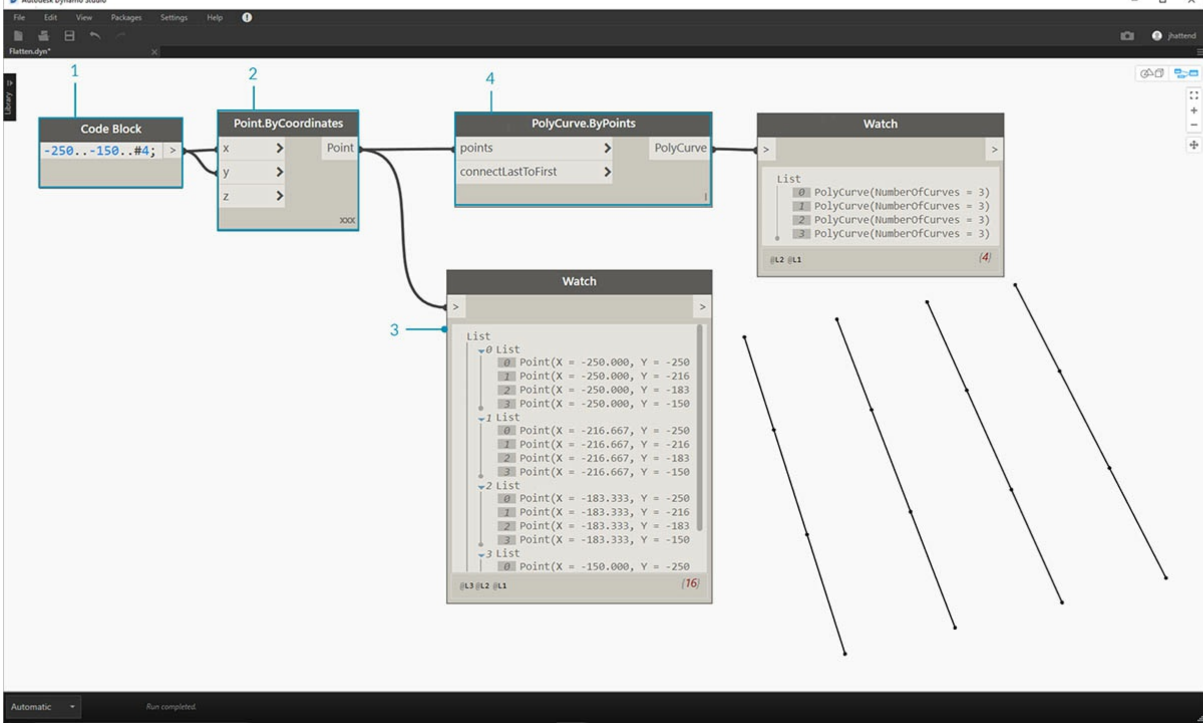
1. С помощью *блока кода* было задано два диапазона: ` 0..2; 0..3;
2. Эти диапазоны соединены с узлом *Point.ByCoordinates*, а в качестве переплетения выбран вариант *Cross Product* (декартово произведение). При этом создается сетка точек, а в качестве выходных данных возвращается список списков.
3. Обратите внимание, что узел *Watch* содержит 3 списка с 4 элементами в каждом.
4. При использовании функции *List.GetItemAtIndex* с индексом 0, Dynamo выберет первый список и все его содержимое. Другие программы могут выбрать первый элемент каждого списка в структуре данных, но в Dynamo при работе с данными используется иерархия «сверху вниз».

Flatten и List.Flatten

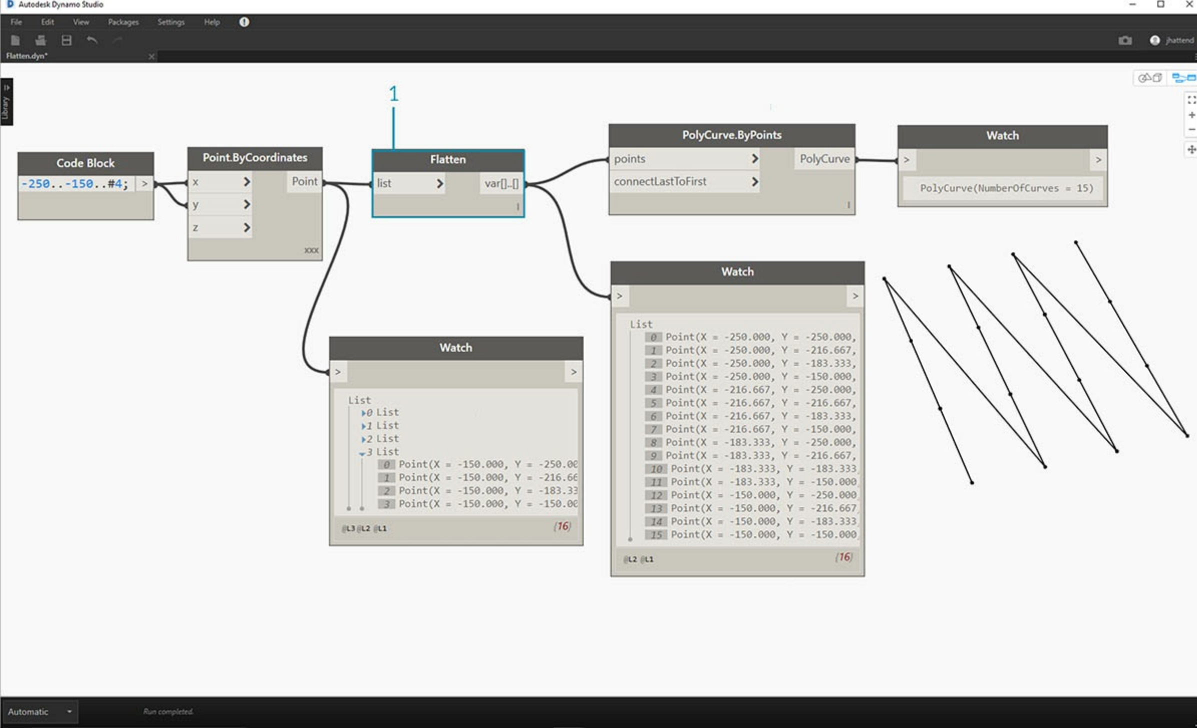
Функция Flatten удаляет все уровни в структуре данных. Это удобно, если для выполнения операции не требуется наличие иерархий данных, но имеются определенные риски, так как удаляется информация. В примере ниже показан результат выравнивания списка данных.

Упражнение «Flatten (выравнивание)»

Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): Flatten.dyn. Полный список файлов примеров можно найти в приложении.



1. Вставьте одну строку кода для определения диапазона в *блоке кода*:`-250..-150..#4;`
2. При вставке *блока кода* во входные данные *x* и *y* узла *Point.ByCoordinates* в качестве варианта переплетения укажем *Cross Product* (декартово произведение), чтобы получить сетку точек.
3. Узел *Watch* показывает наличие списка списков.
4. Узел *PolyCurve.ByPoints* создаст ссылки для каждого списка и построит соответствующую сложную кривую. Обратите внимание, что в области предварительного просмотра Dynamo отобразятся четыре сложные кривые, представляющие каждый ряд сетки.



1. После вставки функции *Flatten* перед узлом сложной кривой был создан один список для всех точек. Узел Polycurve создает ссылку для списка, чтобы создать одну кривую, а так как все точки находятся в одном списке, получается одна зигзагообразная сложная кривая, которая проходит по всему списку точек.

Можно также выровнять изолированные уровни данных. С помощью узла List.Flatten можно указать определенное количество уровней данных, выравниваемых от верхнего уровня иерархии. Это очень полезный инструмент при работе со сложными структурами данных, которые могут быть не нужны в рабочем процессе. Еще один вариант — использовать узел Flatten в качестве функции в List.Map. Далее функция List.Map будет рассматриваться подробнее.

Chop

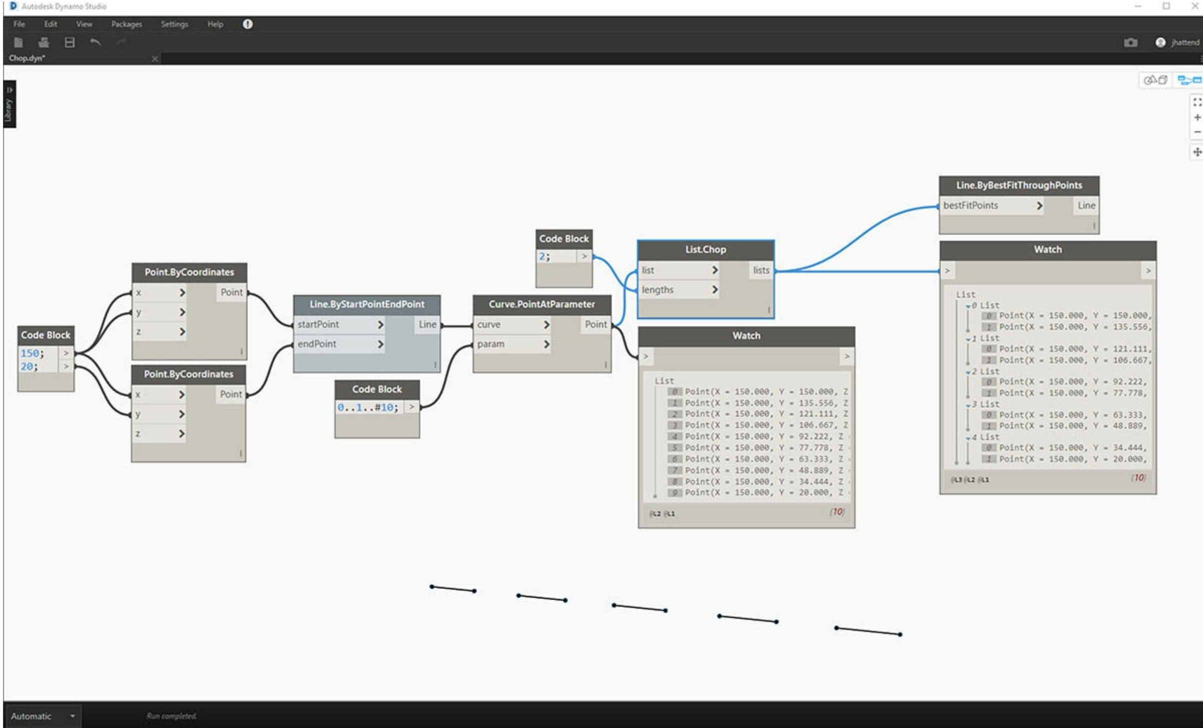
При параметрическом моделировании бывает необходимо расширить структуру данных в существующем списке. С этой целью можно использовать множество других узлов, из которых Chop — самая базовая версия. С помощью функции Chop можно разделить список на вложенные списки с заданным количеством элементов.

Упражнение List.Chop

> Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): Chop.dyn. Полный список файлов примеров можно найти в приложении.

> Команда *List.Chop с длиной вложенного списка* , равной 2, создает 4 списка с 2 элементами в каждом.

Команда Chop (обрезка) делит списки на основе заданной длины списка. В некотором смысле обрезка обратна выравниванию: вместо упрощения структуры данных она добавляет в нее новые уровни. Это удобный инструмент для геометрических операций, таких как в примере ниже.



List.Map и List.Combine

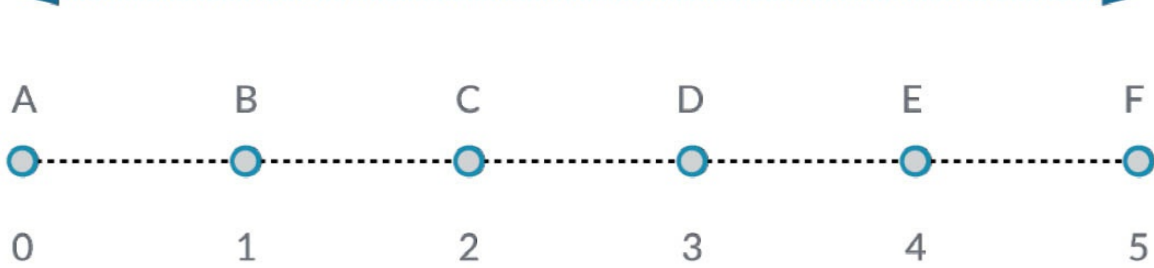
List.Map и List.Combine позволяют применить заданную функцию к списку входных данных, но на один шаг вниз по иерархии. Набор комбинаций аналогичен команде Maps, за исключением наличия нескольких наборов входных данных, соответствующих входным данным заданной функции.

Упражнение List.Map

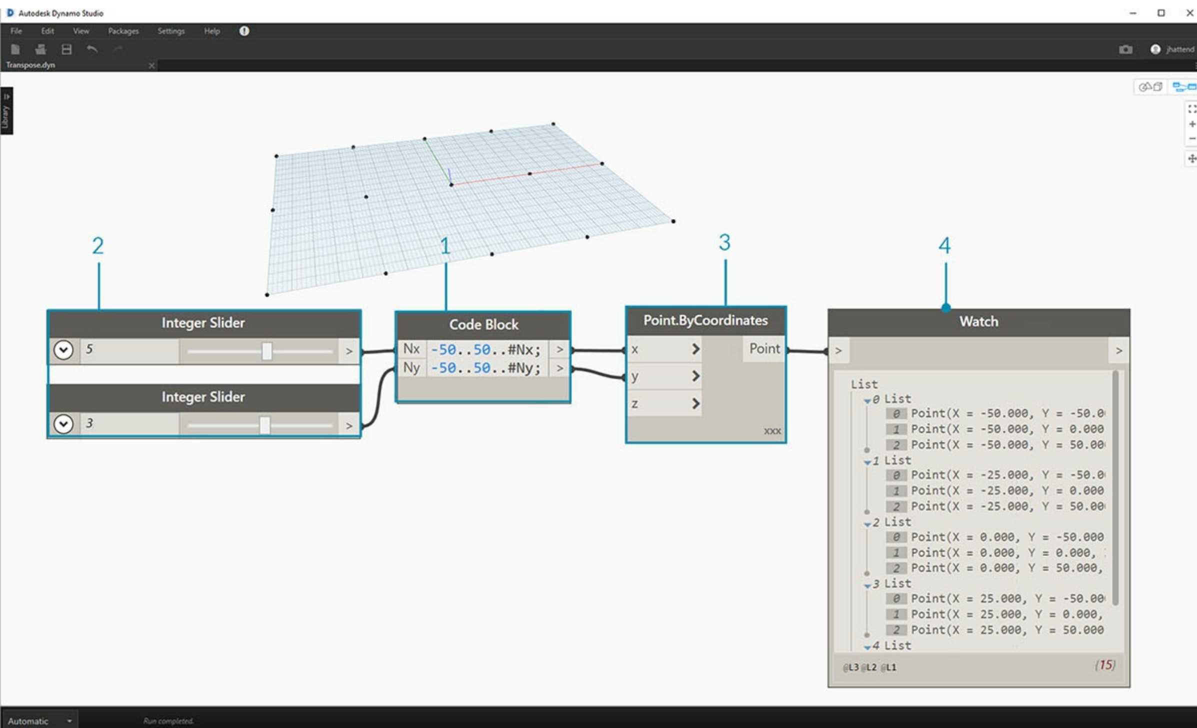
Примечание. Это упражнение было создано в предыдущей версии Dynamo. Большая часть функциональных возможностей List.Map была упразднена с добавлением функции List@Level. Дополнительные сведения см. в разделе List@Level ниже.

> Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): Map.dyn. Полный список файлов примеров можно найти в приложении.

В качестве краткого введения рассмотрим узел List.Count из предыдущего раздела.



Узел *List.Count* подсчитывает все элементы в списке. Мы воспользуемся этим для демонстрации работы *List.Map*.

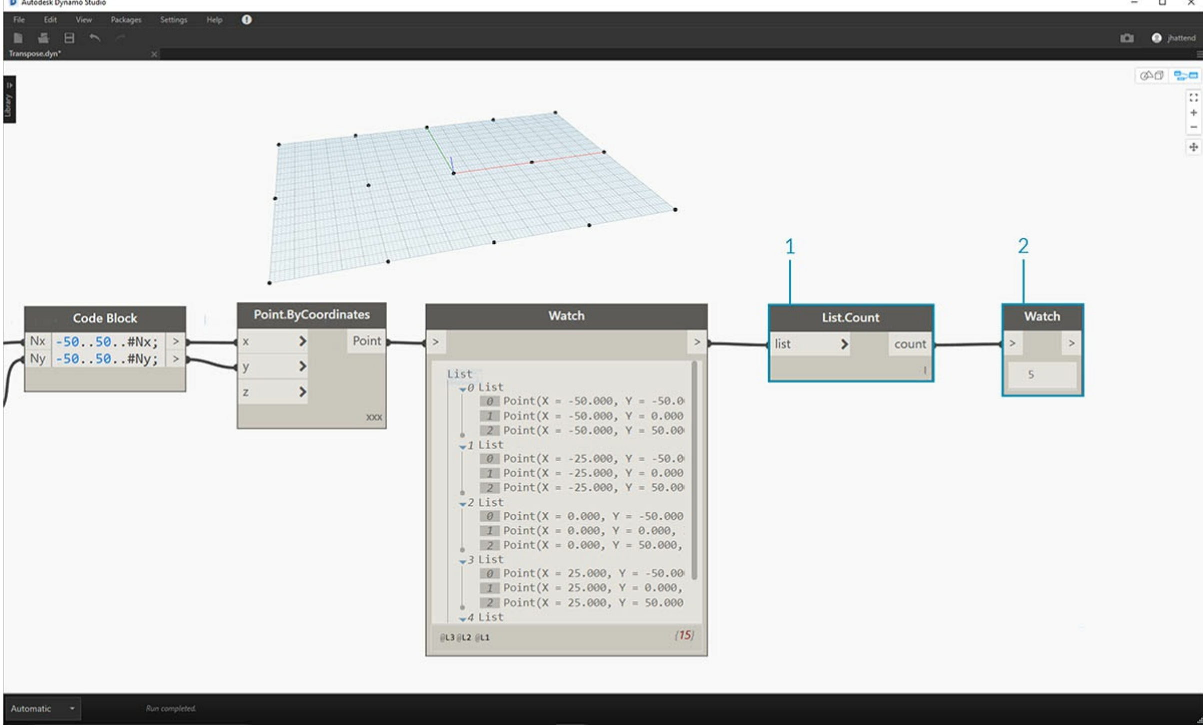


1. Вставьте следующие две строки кода в *блок кода*:

```
-50..50..#Nx;
-50..50..#Ny;
```

После ввода этих данных блок кода создаст два набора входных данных для Nx и Ny.

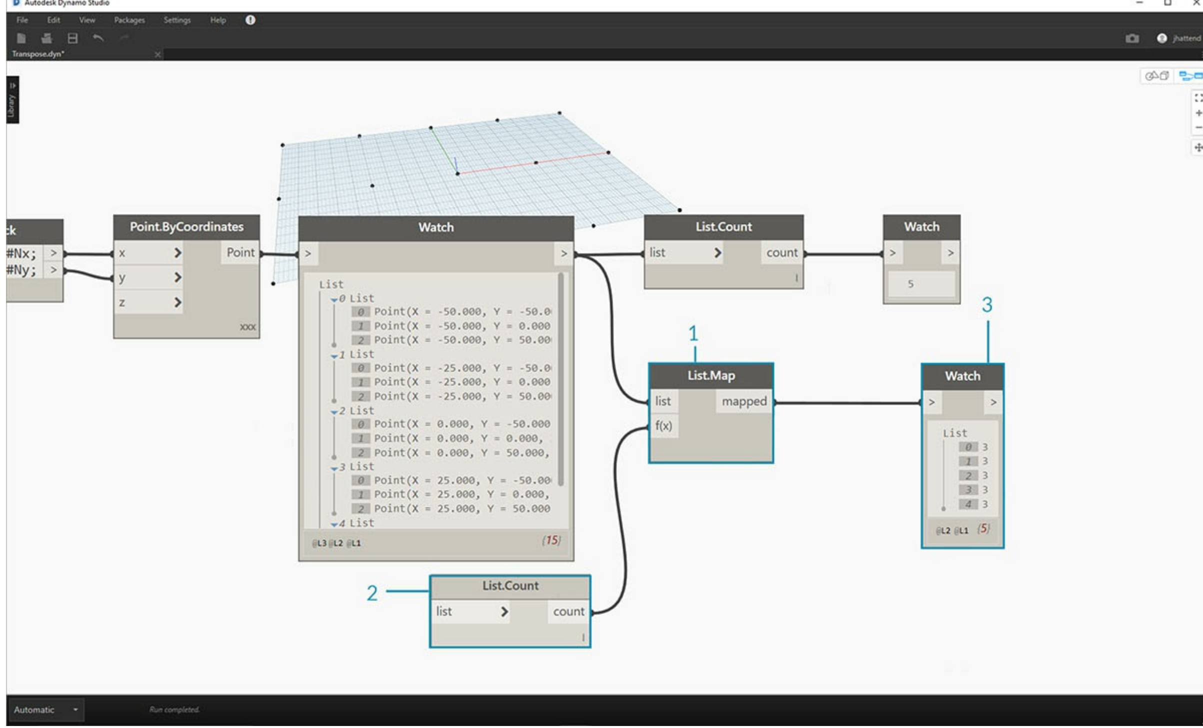
1. С помощью двух узлов *Integer Slider* задайте значения *Nx* и *Ny* путем их присоединения к *блоку кода*.
2. Соедините каждую строку блока кода с соответствующими входными данными *X* и *Y* узла *Point.ByCoordinates*. Щелкните узел правой кнопкой мыши, выберите Lacing(переплетение), а затем *Cross Product* (декартово произведение). Будет создана сетка точек. Так как мы определили диапазон от -50 до 50, он охватывает сетку Dynamo по умолчанию.
3. Созданные точки отображаются в узле *Watch*. Обратите внимание на структуру данных. Мы создали список списков. Каждый список представляет собой ряд точек сетки.



1. Вставьте узел *List.Count* в выходные данные узла Watch из предыдущего шага.
2. Соедините узел *Watch* с выходными данными List.Count.

Обратите внимание, что узел List.Count выдает значение 5. Это значение равно переменной Nx, заданной в блоке кода. Почему?

- Во-первых, в качестве основного входного элемента для создания списков в узле Point.ByCoordinates используется входное значение x. Если Nx равно 5, а Ny — 3, получается список, состоящий из 5 списков, в каждом из которых содержится 3 элемента.
- Так как Dynamo рассматривает списки как объекты самих себя, расположенные в самих себе, то узел List.Count применяется к основному списку в иерархии. В результате получается значение 5 (количество списков в главном списке).



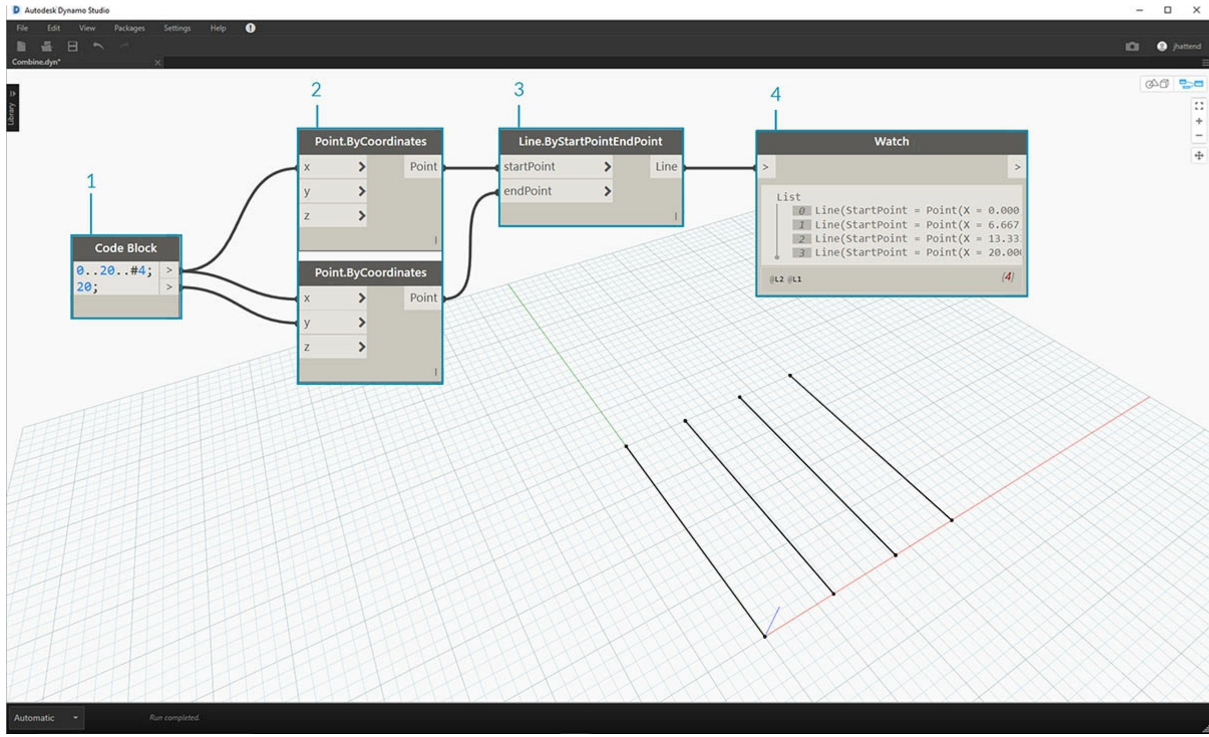
1. С помощью узла *List.Map* спустимся на один шаг вниз по иерархии и на этом уровне выполним *функцию*.
2. Обратите внимание, что узел *List.Count* не имеет входных данных. Так как узел *List.Count* используется в качестве функции, он будет применен к каждому отдельному списку на один шаг вниз по иерархии. Пустые входные данные узла *List.Count* соответствуют входным данным списка в узле *List.Map*.
3. Результаты *List.Count* теперь выдают список из 5 элементов, в каждом из которых имеется значение 3. Это соответствует длине каждого вложенного списка.

Упражнение List.Combine

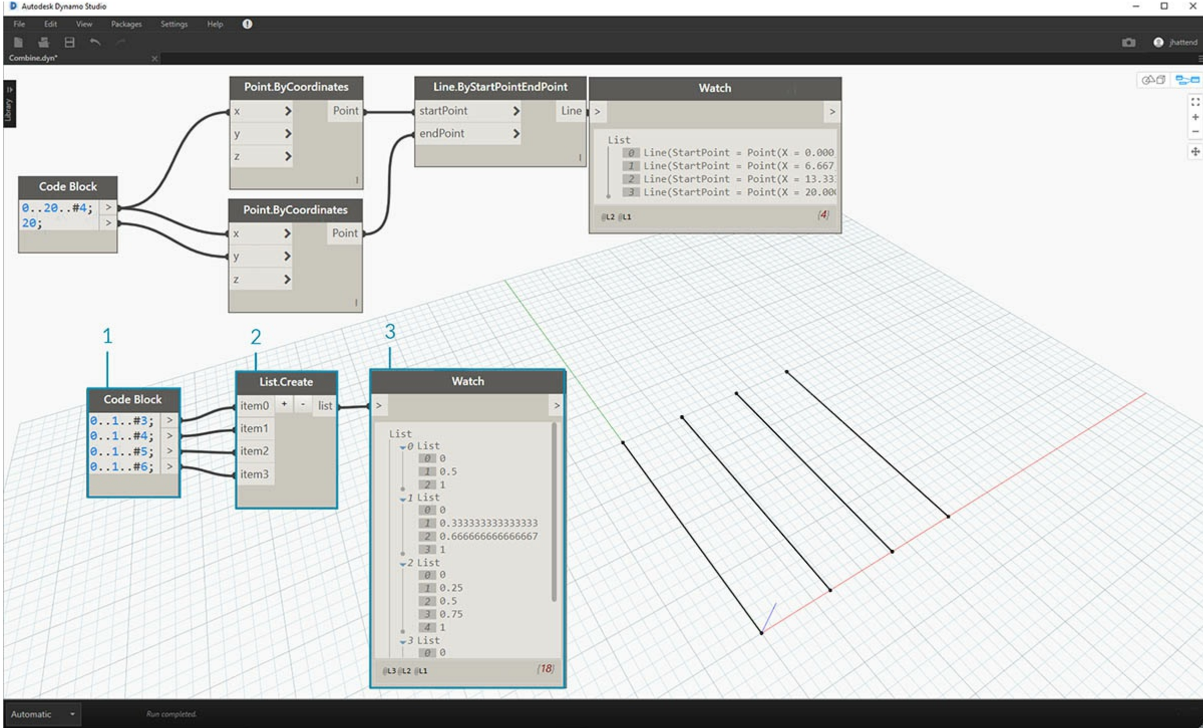
Примечание. Это упражнение было создано в предыдущей версии Dynamo. Большая часть функциональных возможностей List.Combine была упразднена с добавлением функции List@Level. Дополнительные сведения см. в разделе [List@Level](#) ниже.

Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): [Combine.dyn](#). Полный список файлов примеров можно найти в приложении.

В этом упражнении используется та же логическая схема, как в случае с List.Map, но с несколькими элементами. В данном случае необходимо разделить список кривых на уникальное количество точек.



1. С помощью *блока кода* задайте диапазон, используя синтаксис `..20..#4;` и значение`20;` под строкой.
2. Соедините *блок кода* с двумя узлами *Point.ByCoordinates*.
3. Создайте узел *Line.ByStartPointEndPoint* из узлов *Point.ByCoordinates*.
4. В узле *Watch* отображается четыре строки.

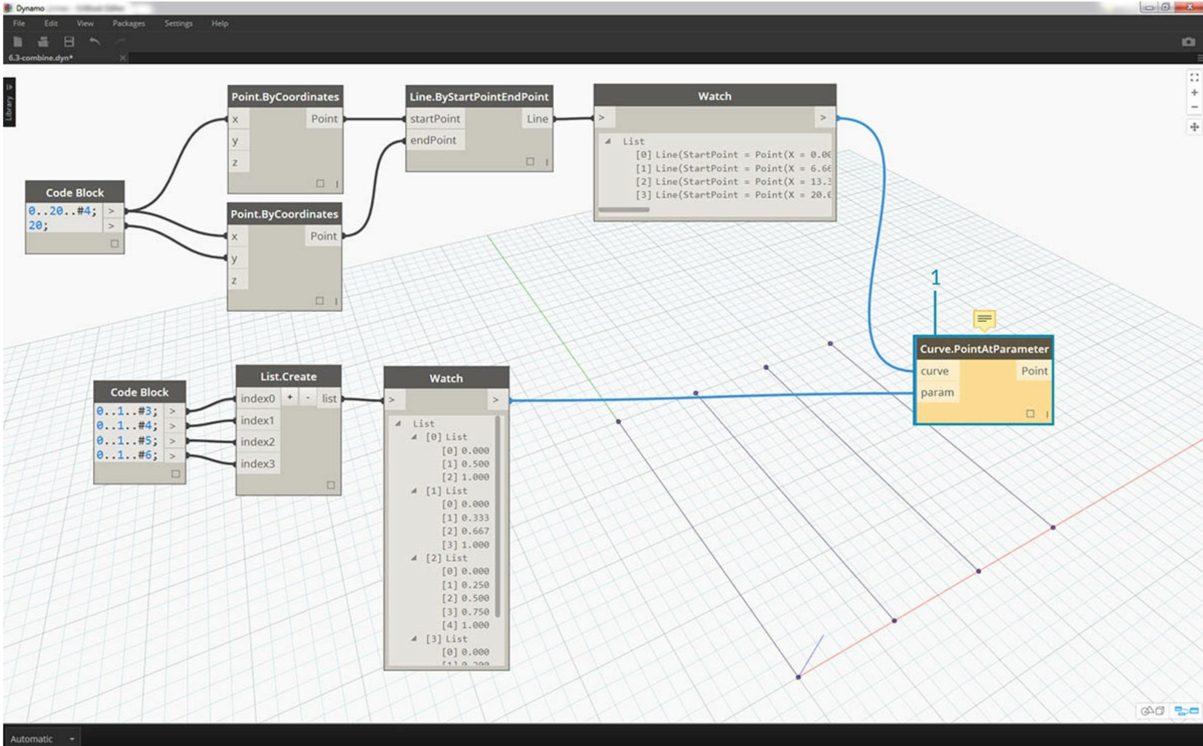


1. Под графиком для создания линии потребуется использовать блок кода, позволяющий создать четыре различных диапазона для

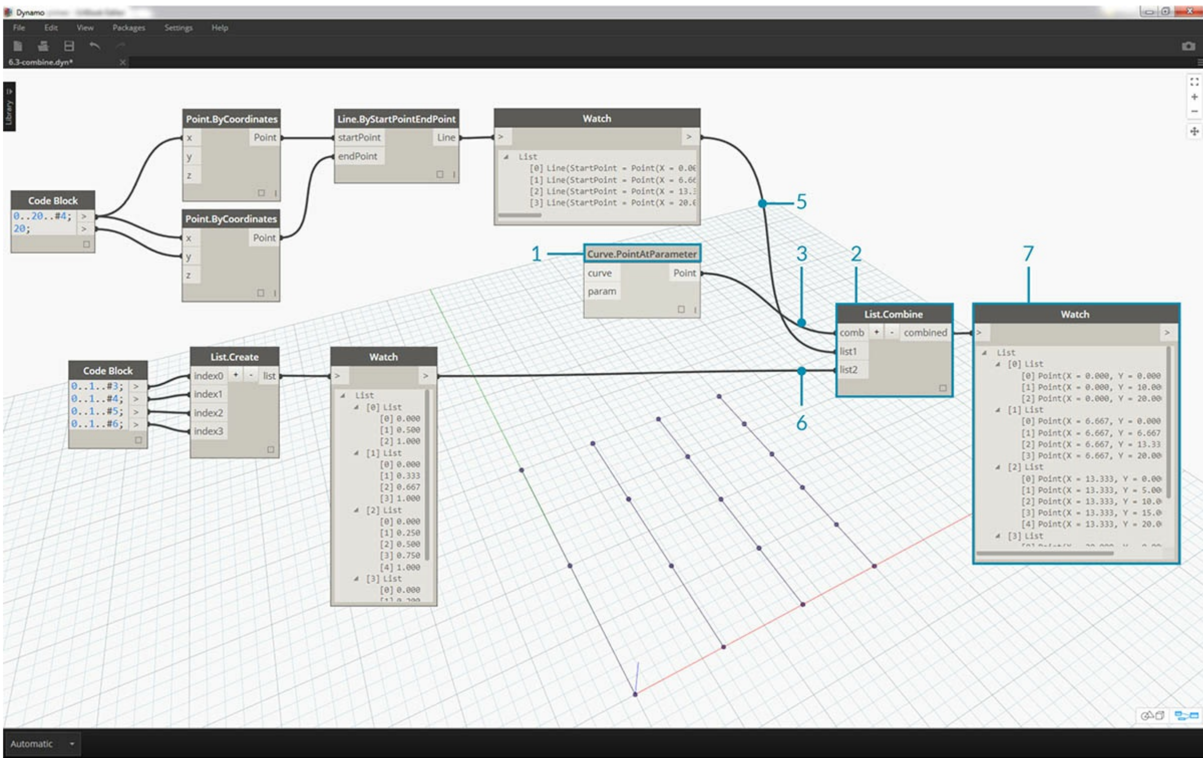
уникального разделения линий. Для этого используем следующие строки кода:

```
0..1..#3;
0..1..#4;
0..1..#5;
0..1..#6;
```

1. С помощью узла *List.Create* четыре строки из *блока кода* объединяются в один список.
2. В узле *Watch* отобразится список списков.



1. Узел *Curve.PointAtParameter* не работает, если соединить линии непосредственно со значениями *параметров*. Необходимо опуститься на один уровень вниз по иерархии. Для этого используем команду *List.Combine*.



Используя команду *List.Combine*, можно успешно разделить каждую линию на заданный диапазон. Это непростая операция, поэтому

разберем ее подробнее.

1. Сначала добавьте узел *Curve.PointAtParameter* в рабочую область. Это будет *функция или* комбинатор*, применяемые к узлу List.Combine*. Подробнее рассмотрим этот момент позже.
2. Добавьте узел *List.Combine* в рабочую область. Нажмите *«+»* или *«-»*, чтобы сложить или вычесть входные данные. В данном случае в узле используется два набора входных данных по умолчанию.
3. Необходимо встроить узел *Curve.PointAtParameter* в набор входных данных *comb* в узле *List.Combine*. Есть еще один важный узел. Правой кнопкой мыши щелкните *набор входных данных param узла Curve.PointAtParameter* и снимите флажок *use default value* (использовать значение по умолчанию). При запуске узла в качестве функции необходимо удалить значения, присвоенные входным данным в Dynamo по умолчанию. Другими словами, необходимо считать, что у значений по умолчанию имеются подсоединенные к ним дополнительные узлы. Поэтому в данном случае необходимо удалить значения по умолчанию.
4. Итак, имеется два набора входных данных: линии и параметры для создания точек. Но как и в каком порядке соединить их с входными данными *List.Combine*?
5. Пустые входные данные узла *Curve.PointAtParameter* необходимо заполнить в комбинаторе в том же порядке — сверху вниз. Итак, линии встраиваются во входные данные *list1* узла *List.Combine*.
6. Аналогичным образом значения параметров встраиваются во входные данные *list2* узла *List.Combine*.
7. Узел *Watch* и область предварительного просмотра Dynamo демонстрируют, что имеется 4 линии, каждая из которых разделена на основе диапазонов, заданных в *блоке кода*.

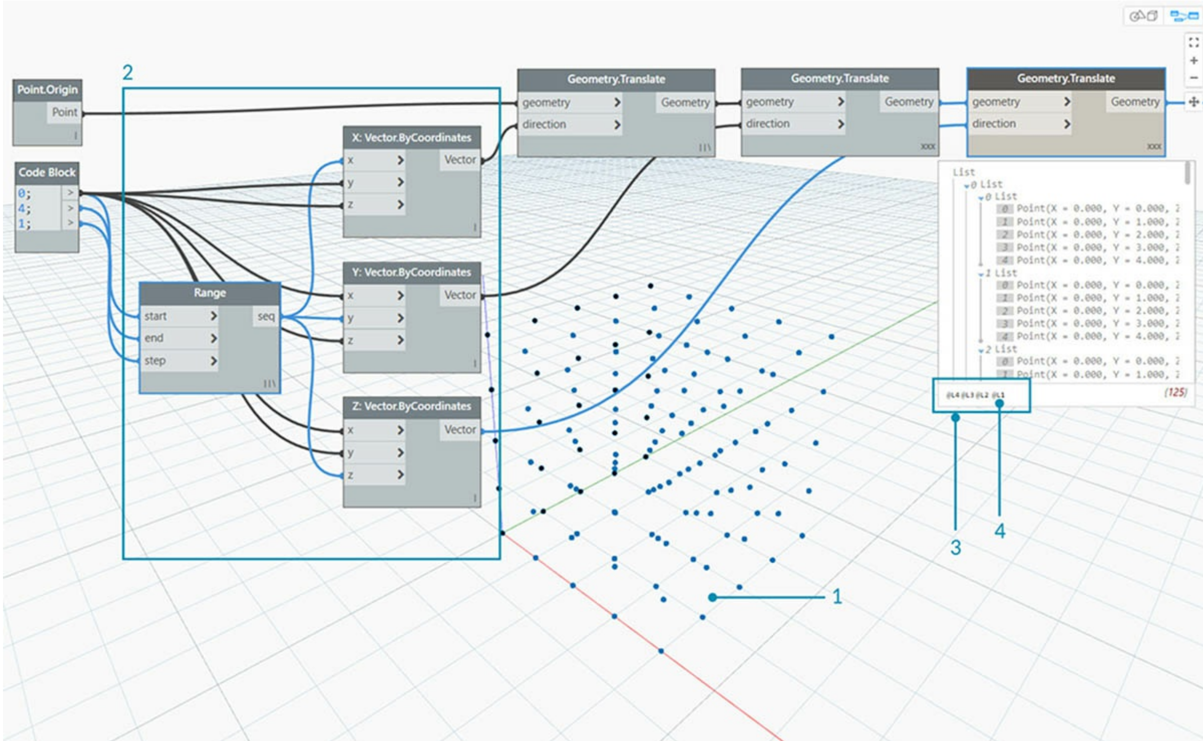
List@Level

В отличие от List.Map функция List@Level позволяет выбрать необходимый уровень списка непосредственно на входном порте узла. Эту функцию можно применять ко всем поступающим входным данным узлов и получать доступ к уровням списков быстрее, чем при использовании других методов. Просто сообщите узлу, какой уровень списка требуется использовать в качестве входных данных, и он сам сделает все необходимое.

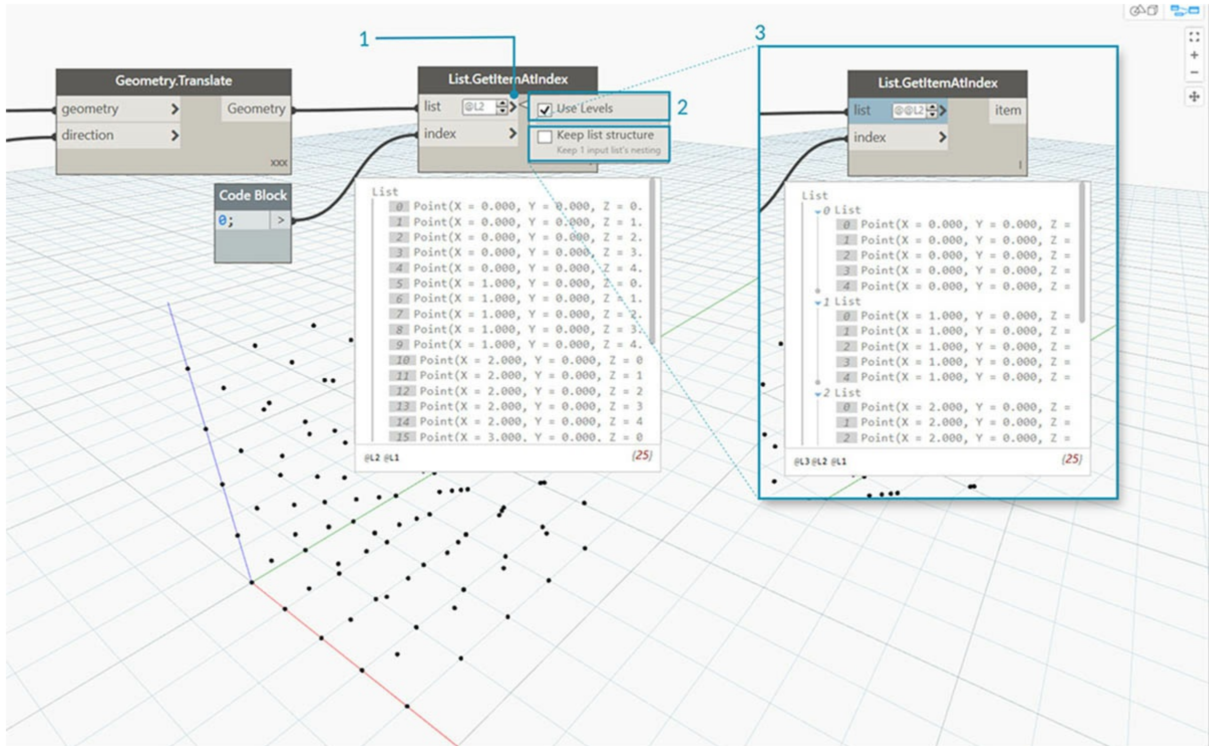
Упражнение List@Level

В этом упражнении с помощью функции List@Level необходимо изолировать определенный уровень данных.

Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): List@Level. Полный список файлов примеров можно найти в приложении.

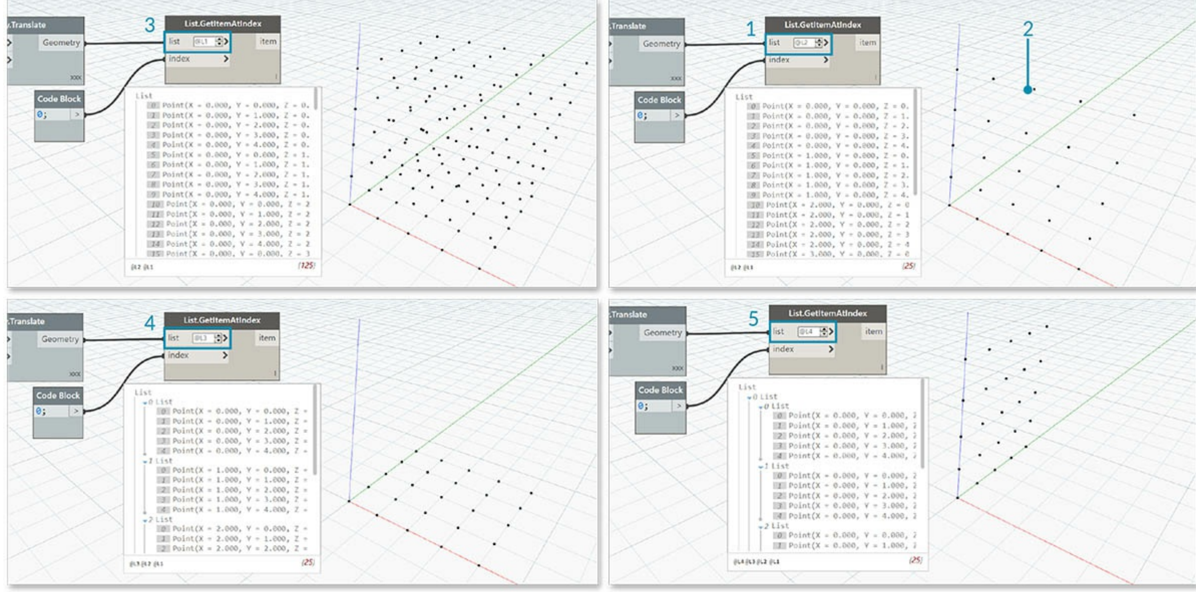


1. Начнем с простой 3D-сетки точек.
2. Поскольку сетка создается с диапазоном для X, Y и Z, структура данных будет иметь 3 уровня: список X, список Y и список Z.
3. Эти уровни расположены на разной высоте (**уровнях**). Они указаны в нижней части марки предварительного просмотра. Столбцы уровня списка соответствуют данным списка выше, что позволяет быстрее найти нужный уровень.
4. Уровни списка располагаются в обратном порядке, так что данные самого низкого уровня всегда находятся на высоте L1. Благодаря этому графики будут функционировать запланированным образом, даже если что-то изменится в предыдущем алгоритме.



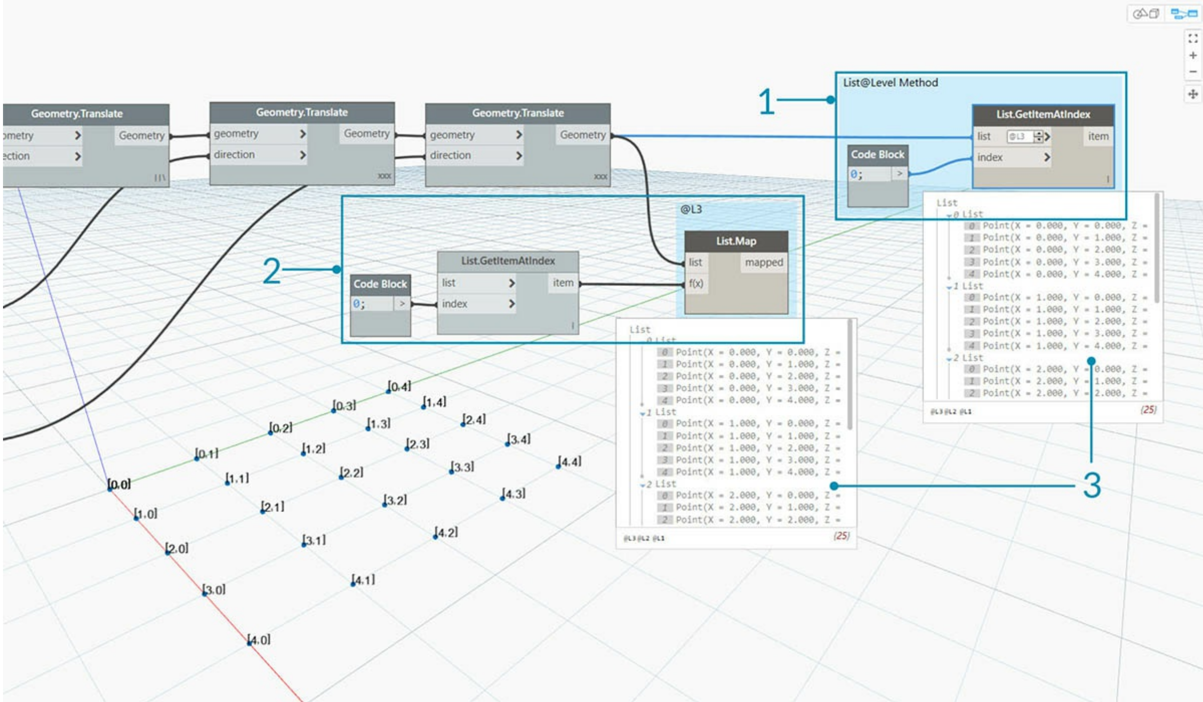
1. Чтобы использовать функцию List@Level, нажмите кнопку «>». В меню отобразятся два флажка.
2. **Используйте уровни**: включение функции List@Level. После выбора этого параметра можно с помощью мыши выбрать уровни списка входных данных, которые будут использованы узлом. С помощью этого меню можно быстро проверить различные конфигурации уровней, щелкая мышью выше или ниже.
3. **Сохранить структуру списков**: если установить этот флажок, можно будет сохранить структуру уровней этих входных данных. Иногда данные бывают сознательно разделены по вложенным спискам. Если установить этот флажок, можно сохранить структуру списка неизменной без какой-либо потери информации.

Благодаря этой простой 3D-сетке можно получить доступ к структуре списка и визуализировать ее, переключаясь между уровнями списка. Любая комбинация уровня списка и индекса возвращает свой собственный набор точек из исходного 3D-набора.



1. С помощью элемента @L2 в DesignScript можно выбрать только список на уровне 2.
2. Список на уровне 2 с индексом 0 включает в себя только первый набор точек Y и возвращает только сетку XZ.
3. Если задать фильтр уровней L1, можно увидеть все содержимое первого уровня списка. Список на уровне 1 с индексом 0 включает в себя все 3D-точки в одноуровневом списке.
4. В аналогичном случае с L3 будут видны только точки третьего уровня списка. Список на уровне 3 с индексом 0 включает в себя только первый набор точек Z и возвращает только сетку XY.
5. В аналогичном случае с L4 будут видны только точки третьего уровня списка. Список на уровне 4 с индексом 0 включает в себя только первый набор точек X и возвращает только сетку YZ.

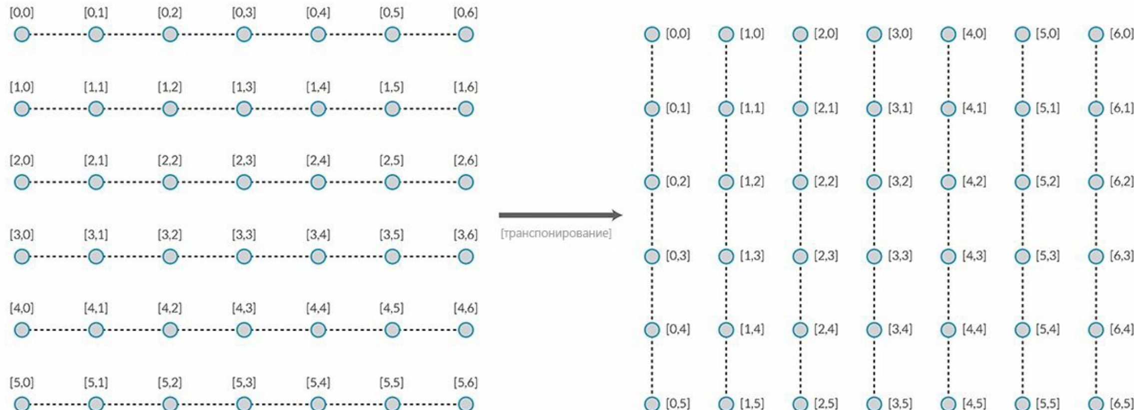
Хотя данный конкретный пример можно воссоздать с помощью List.Map, функция List@Level существенно упрощает операцию и доступ к данным узла. Ниже представлено сравнение методов List.Map и List@Level.



1. Хотя оба метода предоставляют доступ к одним и тем же точкам, метод List@Level позволяет легко переключаться между слоями данных в одном узле.
2. Для доступа к сетке точек с помощью List.Map требуется добавить узел List.GetItemAtIndex в дополнение к List.Map. Для каждого нижестоящего уровня списка необходимо использовать дополнительный узел List.Map. При наличии сложных списков для доступа к нужному уровню информации может потребоваться добавить в график значительное количество узлов List.Map.
3. В этом примере узел List.GetItemAtIndex с узлом List.Map возвращает тот же набор точек и ту же структуру списка, что и List.GetItemAtIndex с выбором @L3.

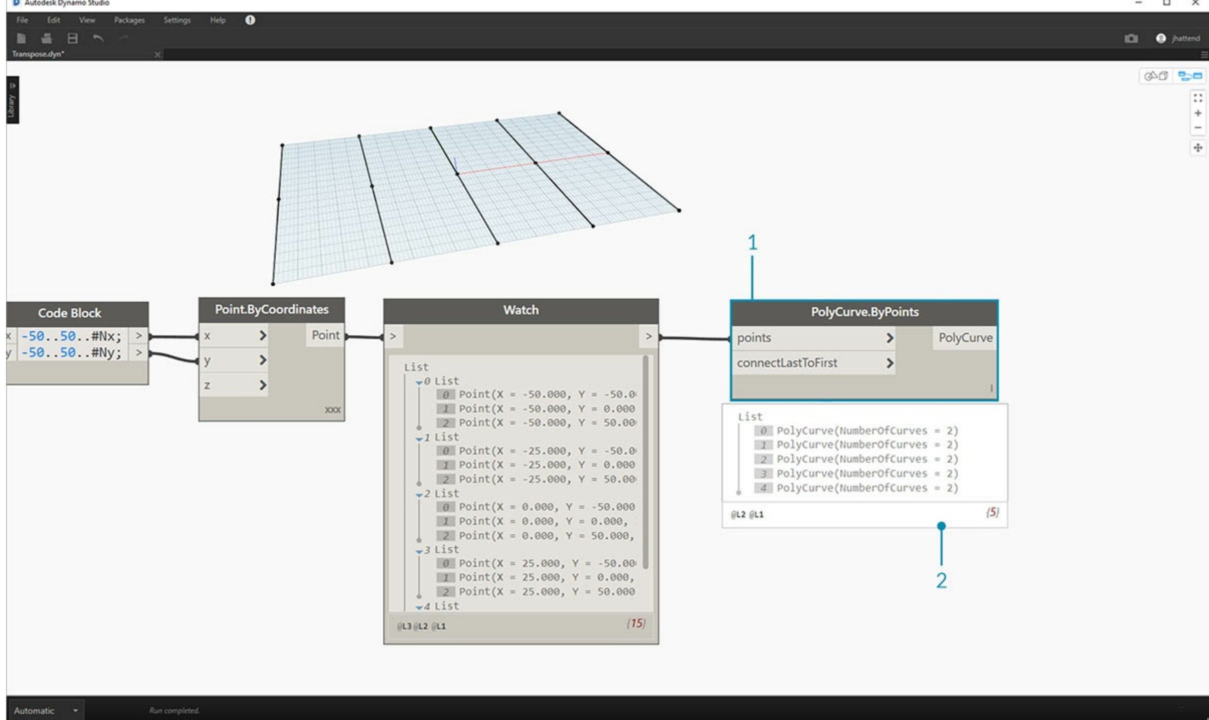
Transpose

Transpose (транспонирование) — это одна из основных функций при работе со списками списков. Как и в электронных таблицах, при транспонировании происходит перестановка столбцов и строк в структуре данных. Продемонстрируем это с помощью следующей базовой матрицы, а в следующем разделе покажем, как с помощью функции транспонирования создавать геометрические взаимосвязи.



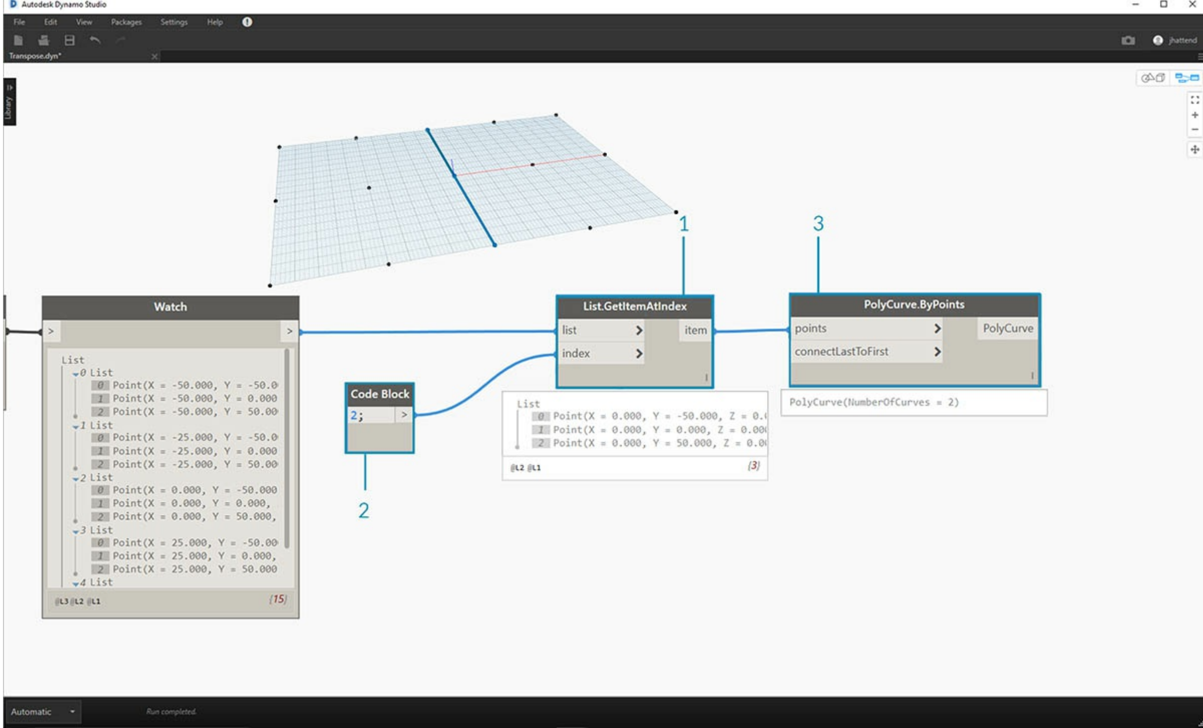
Упражнение List.Transpose

Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): Transpose.dyn. Полный список файлов примеров можно найти в приложении.

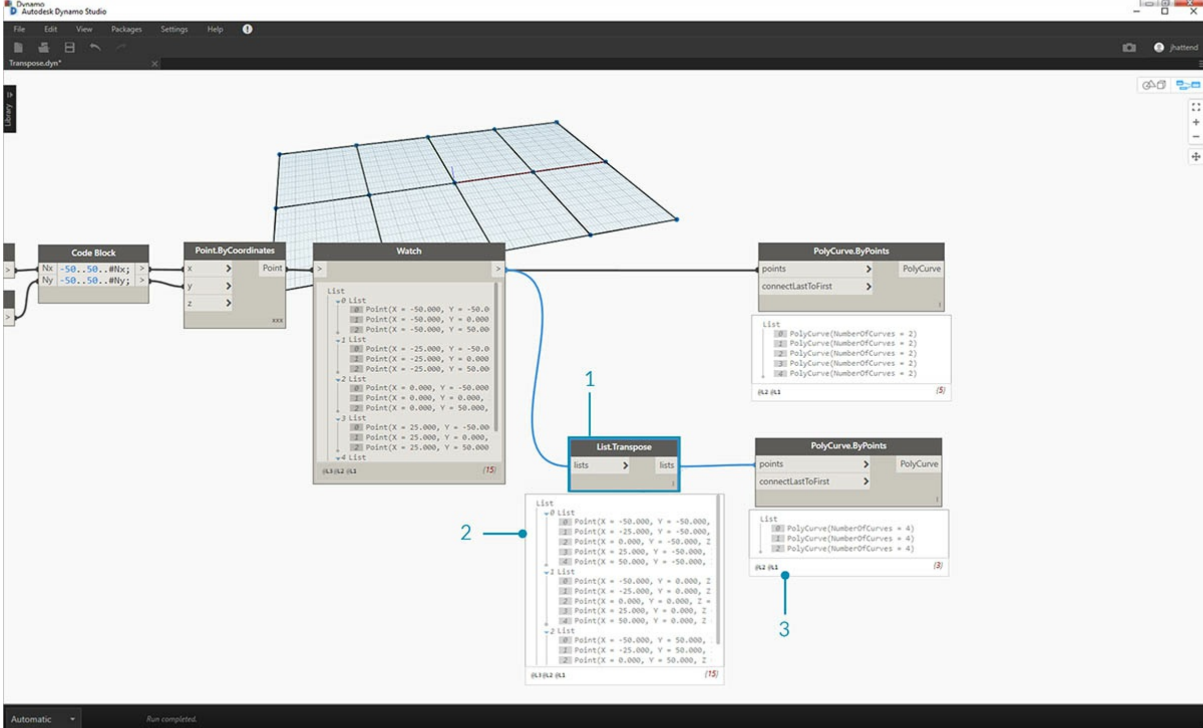


Удалите узлы *List.Count* из предыдущего упражнения и перенесите их на геометрические объекты, чтобы увидеть, как структурированы данные.

1. Соедините узел *PolyCurve.ByPoints* с выходными данными узла Watch от узла *Point.ByCoordinates*.
2. На выходе отобразятся 5 сложных кривых, которые можно видеть в области предварительного просмотра Dynamo. Узел Dynamo выполняет поиск списка точек (в данном случае — списка списков точек) и создает из них одну сложную кривую. По сути, каждый список в структуре данных преобразован в кривую.



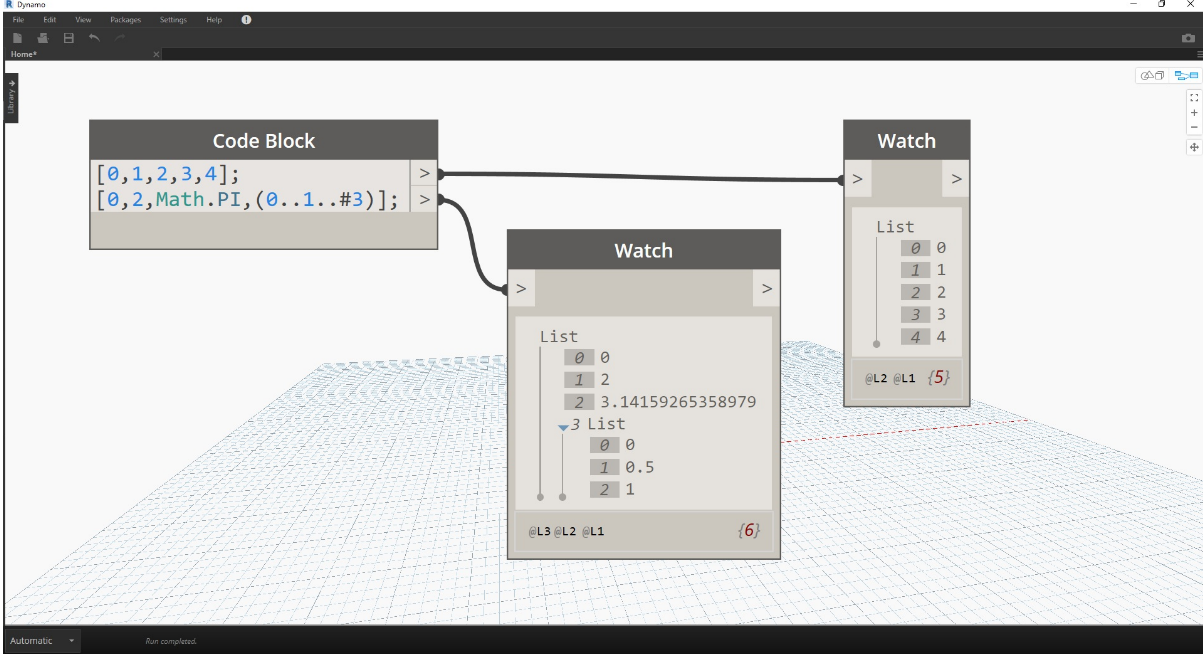
1. Если требуется изолировать один ряд кривых, используется узел *List.GetItemAtIndex*.
2. С помощью значения 2 *блока кода* запросите третий элемент в главном списке.
3. В узле *PolyCurve.ByPoints* отображается одна кривая, так как к нему подсоединен только один список.



1. Узел *List.Transpose* переставляет все элементы со всеми списками в списке списков. Это может показаться сложным, но в Microsoft Excel используется точно такая же логическая схема транспонирования данных: перестановка столбцов со строками в структуре данных.
2. Обратите внимание на изменение в списках: после транспонирования структура, состоявшая из 5 списков с 3 элементами, изменилась на 3 списка с 5 элементами в каждом.
3. Кроме того, обратите внимание на изменение в геометрии: использование узла *PolyCurve.ByPoints* привело к появлению 3 сложных кривых в перпендикулярном направлении к исходным кривым.

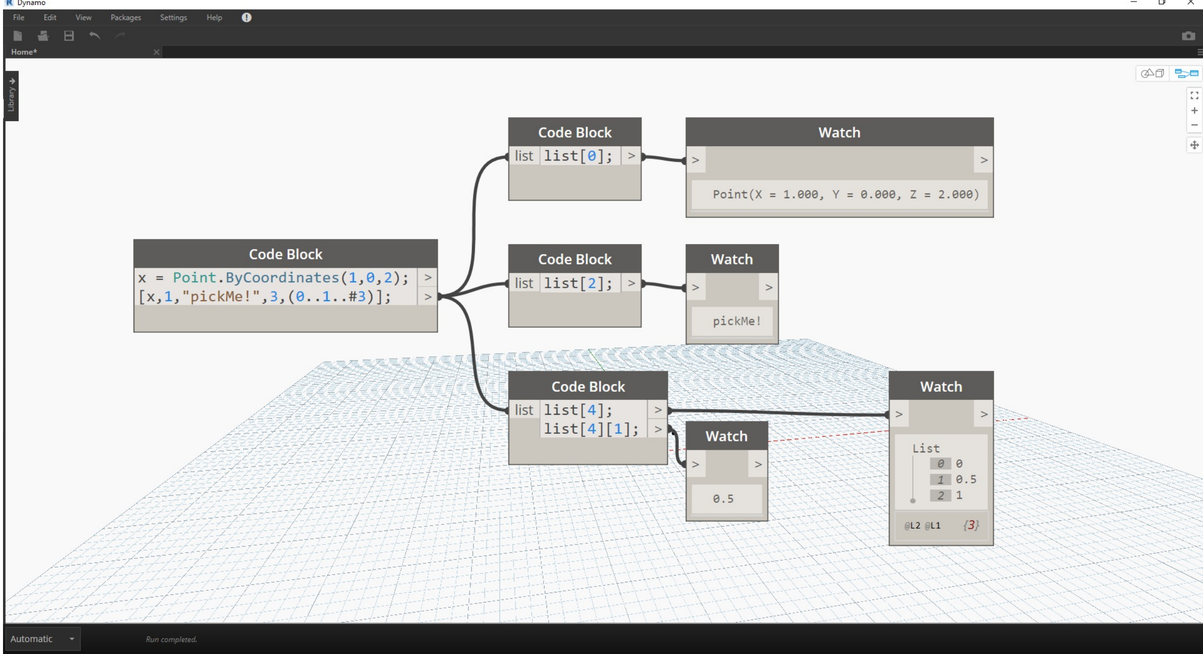
Создание блока кода

Для определения списка в сокращенном языке блока кода используются квадратные скобки ([]). Это гораздо более быстрый и простой способ создания списков, чем с помощью узла List.Create. Более подробно блок кода рассматривается в главе 7. На изображении ниже показано, как можно задать список с несколькими выражениями с помощью блока кода.



Запрос блока кода

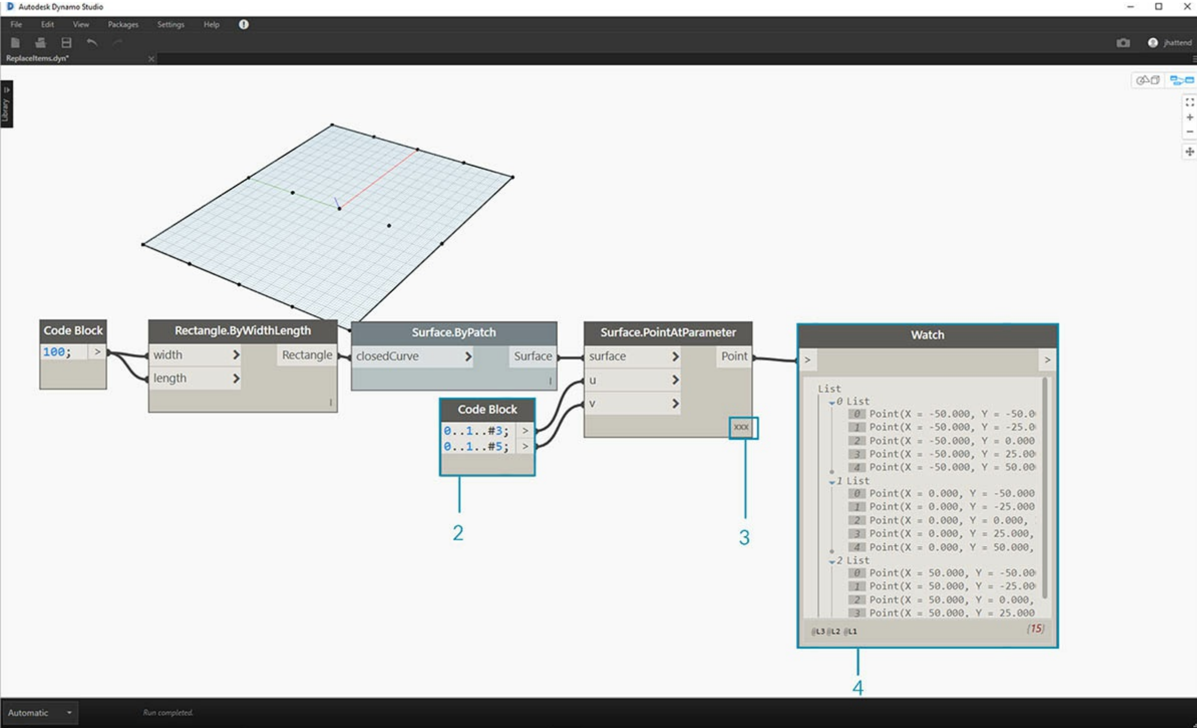
Для упрощенного выбора определенных элементов, которые требуется извлечь из сложной структуры данных, в сокращенном языке блока кода используются квадратные скобки ([]). Более подробно блоки кода рассматриваются в главе 7. На изображение ниже показано, как запросить список с несколькими типами данных с помощью блока кода.



Упражнение «Запрос и вставка данных»

> Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): ReplaceItems.dyn. Полный список файлов примеров можно найти в приложении.

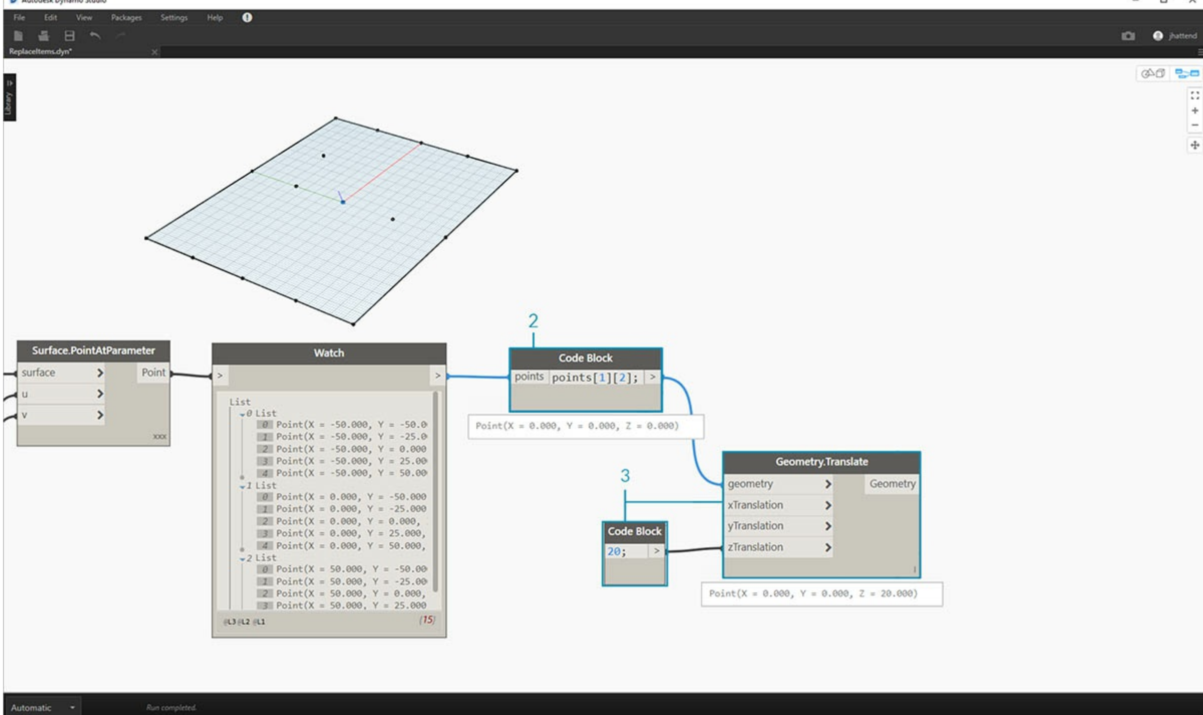
В этом упражнении для редактирования поверхности используется логическая схема из предыдущего упражнения. Эту задачу можно решить интуитивным способом, однако при этом потребуется дополнительная навигация по структуре данных. Необходимо определить поверхность путем перемещения контрольной точки.



1. Начните со строки узлов выше. Создайте базовую поверхность, которая охватывает всю сетку Dynamo по умолчанию.
2. С помощью *блока кода* вставьте следующие две строки кода и соедините их с входными данными *u* и *v* узла *Surface.PointAtParameter* соответственно:

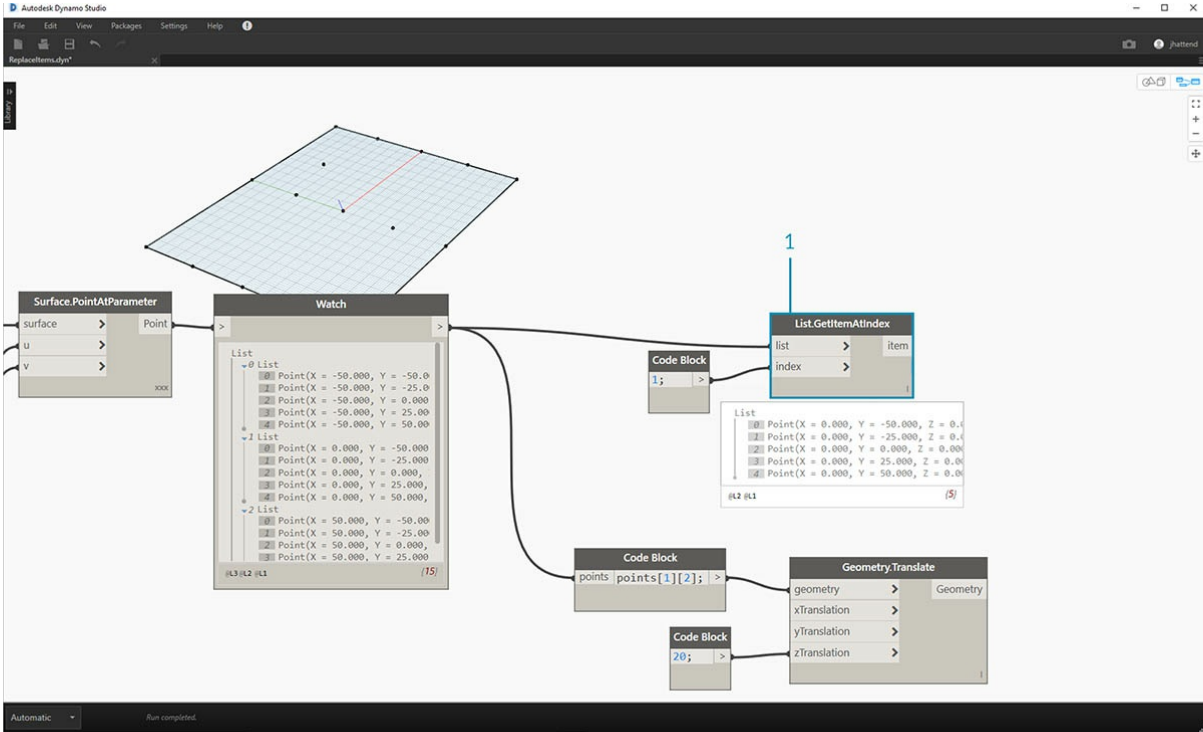
```
-50..50..#3;
-50..50..#5;
```

1. Убедитесь, что для параметра Lacing (переплетение) узла *Surface.PointAtParameter* задано значение *Cross Product* (декартово произведение).
2. Узел *Watch* показывает, что имеется список из 3 списков, каждый из которых содержит 5 элементов.

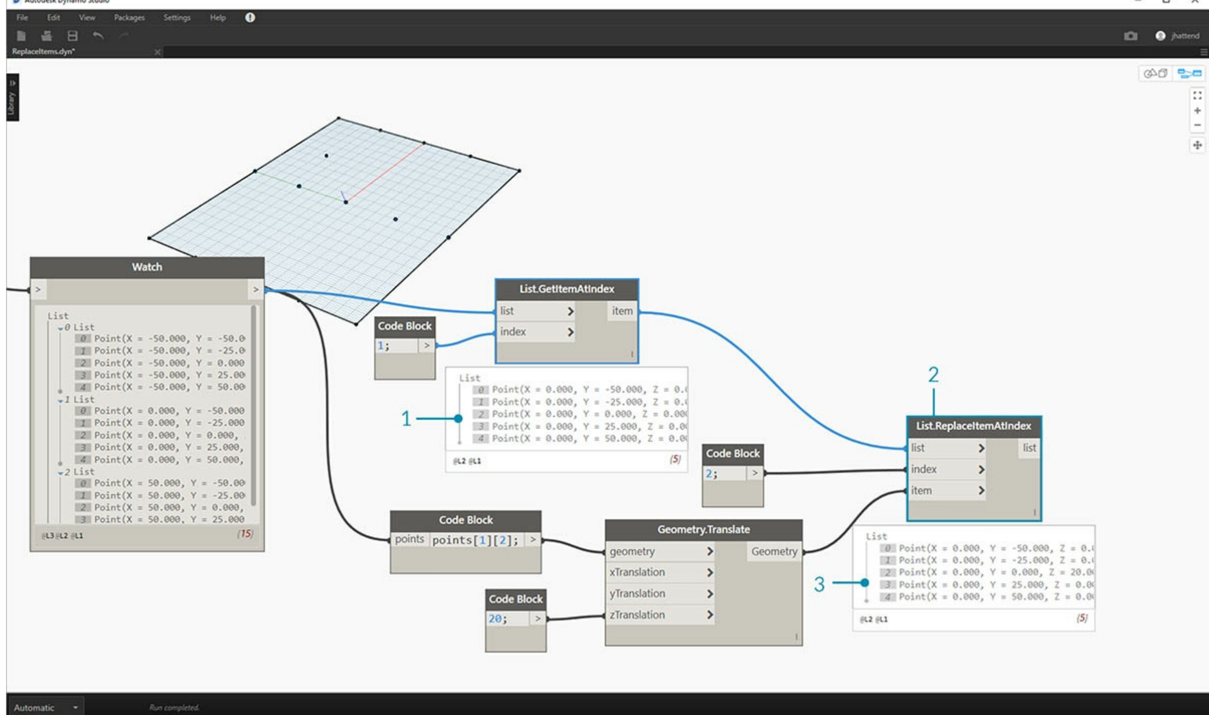


На этом этапе следует запросить центральную точку созданной сетки. Для этого выберите центральную точку в списке посередине. Логично, не так ли?

1. Чтобы убедиться в правильности выбора точки, можно щелкнуть элементы узла Watch для проверки правильности выбора элемента.
2. При помощи *блока кода* создайте базовую строку кода для запроса списка списков: `points[1][2];`
3. С помощью функции *Geometry.Translate* переместите выбранную точку вверх в направлении оси *Z* на *20* единиц.

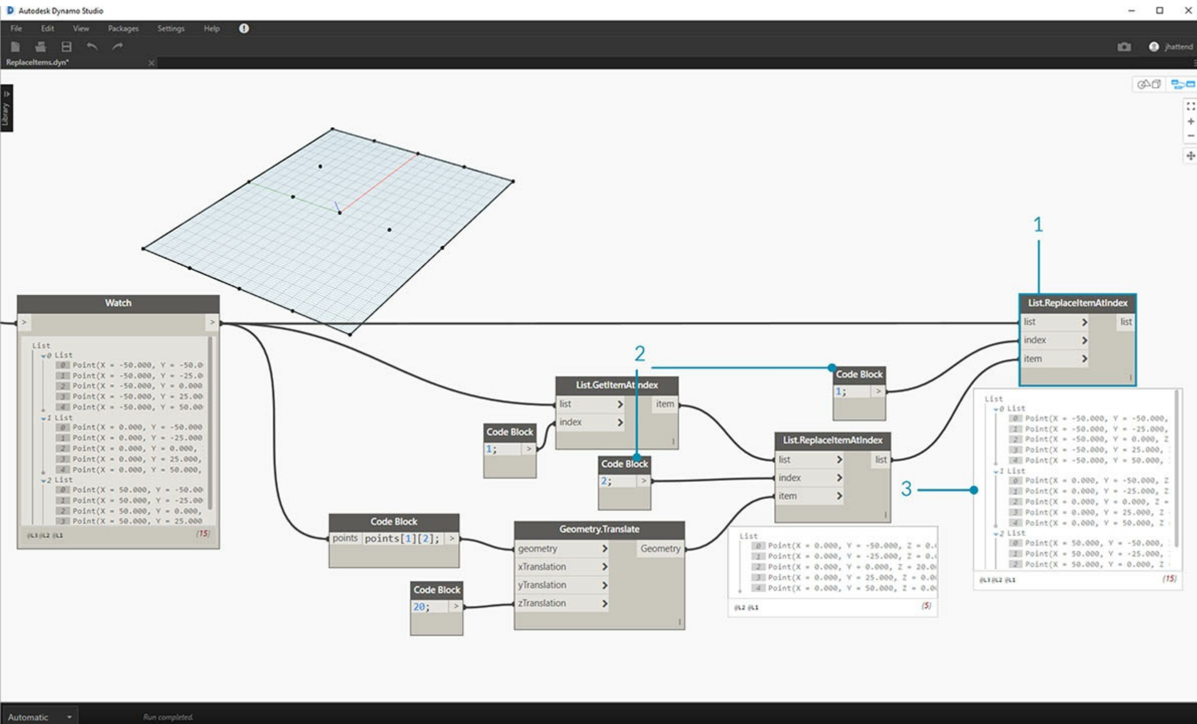


1. Кроме того, выберите ряд точек посредине с помощью узла *List.GetItemAtIndex*. Примечание. Как и при выполнении предыдущего шага, можно запросить список с помощью *блока кода*, используя строку `points[1];`.



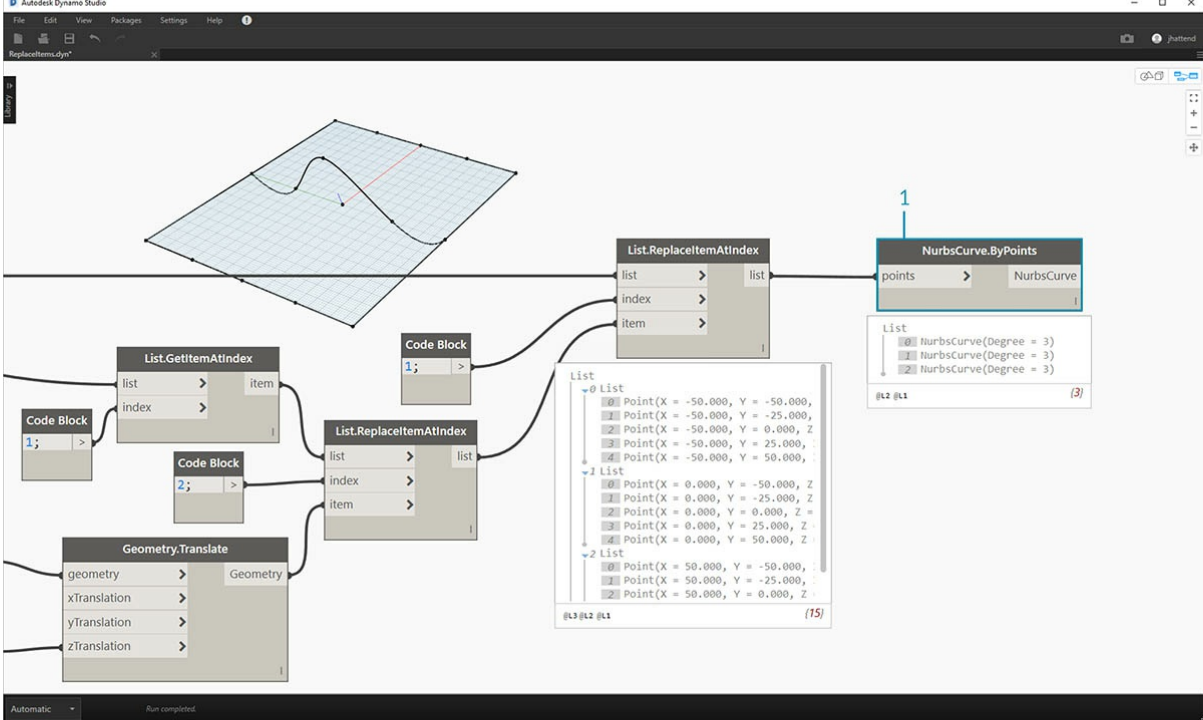
Итак, мы успешно запросили центральную точку и переместили ее вверх. Теперь необходимо вставить эту перемещенную точку обратно в исходную структуру данных.

1. Сначала замените элемент списка, который был изолирован при выполнении предыдущего шага.
2. С помощью узла *List.ReplaceItemAtIndex* замените центральный элемент с помощью индекса *2* на замещающий элемент, соединенный с перемещенной точкой (*Geometry.Translate*).
3. Выходные данные показывают, что перемещенная точка была вставлена в набор входных данных элемента в середине списка.



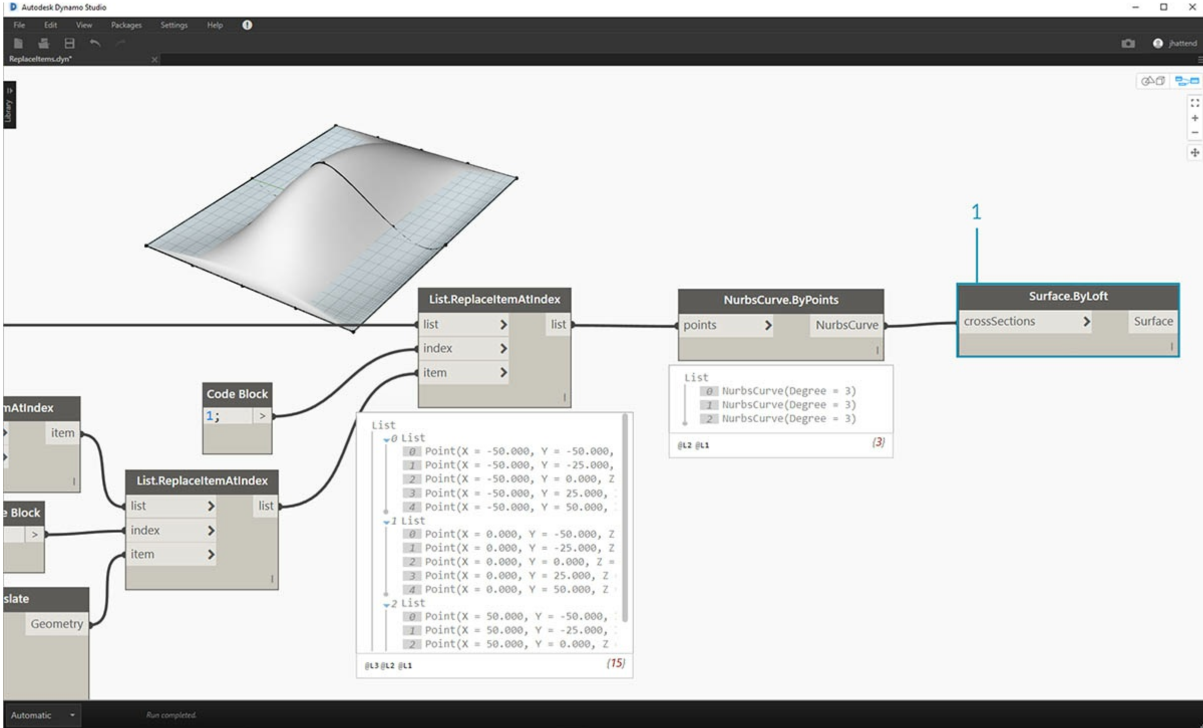
После изменения списка необходимо вставить его обратно в исходную структуру данных — список списков.

1. Используя ту же логическую схему, заменим список в середине на измененный список с помощью узла *List.ReplaceItemAtIndex*.
2. Обратите внимание, что индекс для этих двух узлов определяется *блоками кодов* 1 и 2, что соответствует исходному запросу из *блока кода* (*points[1][2]*).
3. Если выбрать список с помощью *index 1*, то структура данных будет выделена в области предварительного просмотра Dynamo. Итак, мы успешно встроили перемещенную точку в исходную структуру данных.



Существует множество способов создания поверхности из этого набора точек. В данном случае необходимо создать поверхность за счет лофтинга кривых.

1. Создайте узел *NurbsCurve.ByPoints* и присоедините новую структуру данных для создания трех NURBS-кривых.



1. Соедините узел *Surface.ByLoft* с выходными данными из узла *NurbsCurve.ByPoints*. Получится модифицированная поверхность. Можно изменить исходное значение *Z* геометрии. Выполните преобразование и посмотрите, как изменится геометрия.

Многомерные списки

Многомерные списки

Добавим еще больше уровней в иерархию и углубимся в нашу кроличью нору. Структура данных может быть гораздо более объемной, чем простой двумерный список списков. Поскольку списки являются самостоятельными элементами в Dynamo, мы можем создать данные с практически неограниченным количеством измерений.

Это похоже на матрешку. Каждый список можно рассматривать как один контейнер, который содержит несколько элементов. Каждый список обладает собственными свойствами и рассматривается как отдельный объект.

Набор матрешек (фотография предоставлена Zeta) является аналогией многомерных списков. Каждый слой представляет список, и каждый список содержит элементы. В Dynamo каждый контейнер может содержать несколько контейнеров (представляющих элементы каждого списка).

Многомерные списки сложно объяснить визуально, но в данном разделе есть несколько упражнений, которые помогут вам разобраться в работе со списками, число измерений которых превышает два.

Сопоставление и комбинации

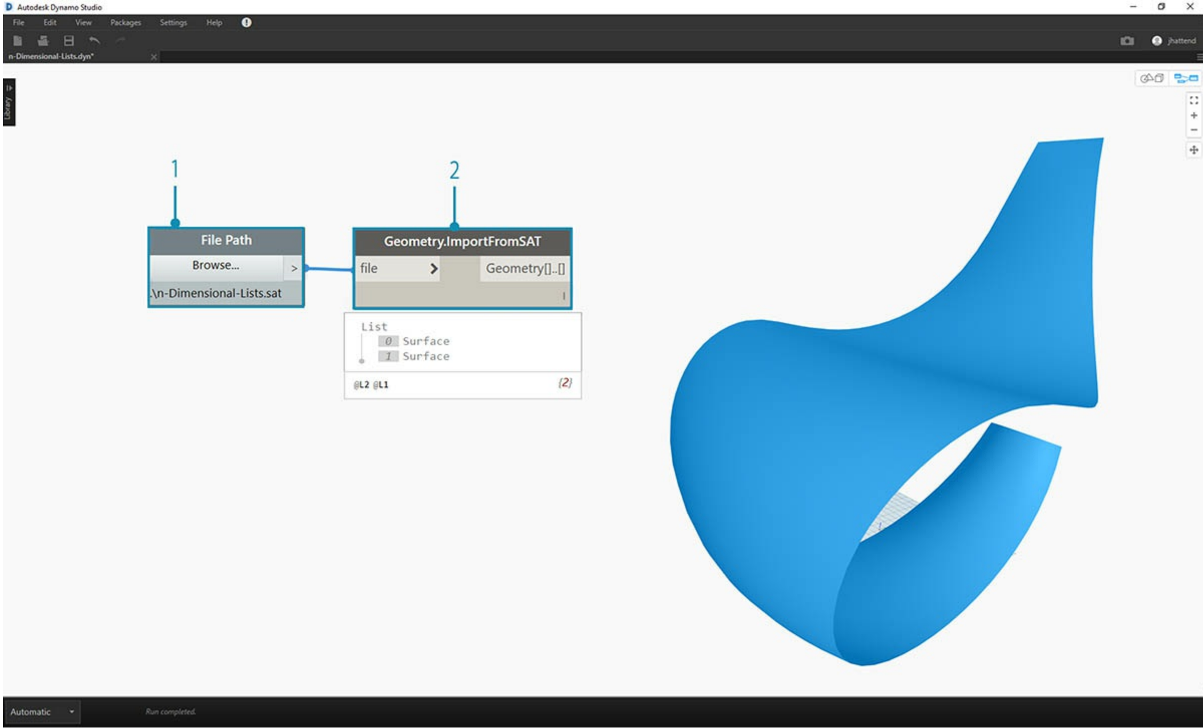
Сопоставление — возможно, самый сложный аспект управления данными в Dynamo, особенно когда речь идет о сложных иерархических структурах, состоящих из списков. В рамках приведенных ниже упражнений мы рассмотрим случаи, в которых следует использовать сопоставление и комбинации при работе с многомерными данными.

Основные сведения по работе с узлами List.Map и List.Combine можно найти в предыдущем разделе. Эти узлы будут использованы для работы со сложной структурой данных в последнем из приведенных ниже упражнений.

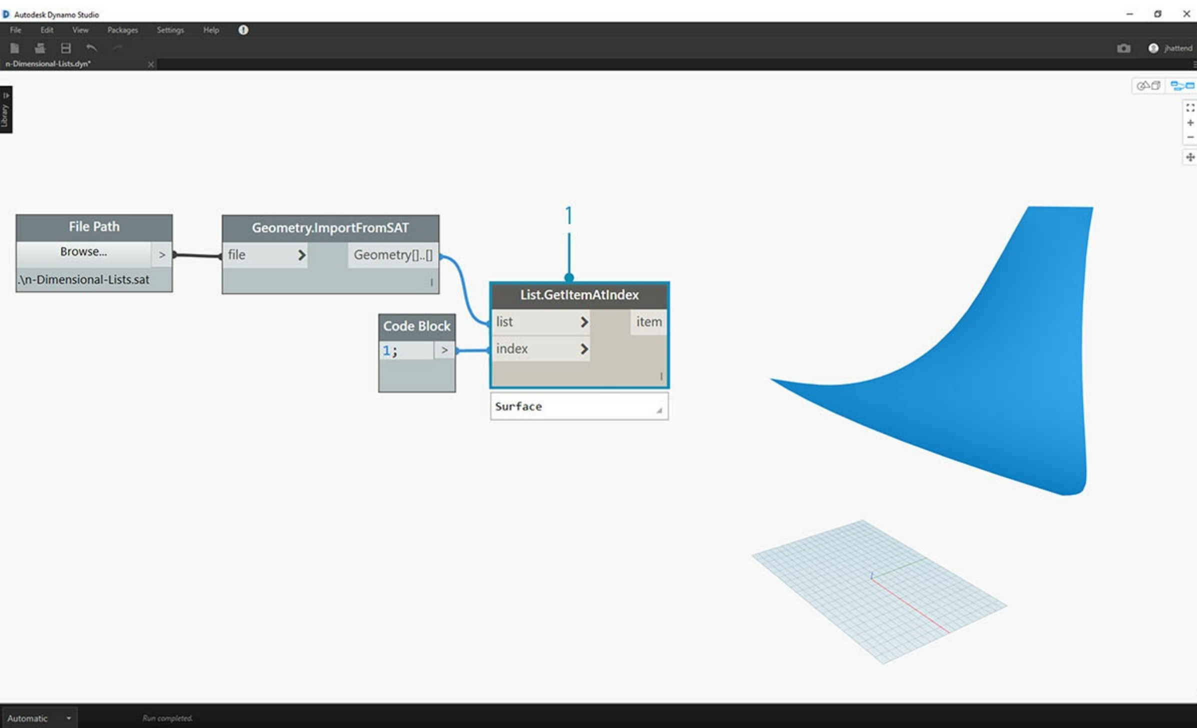
Упражнение. Двумерные списки. Основы

Скачайте файлы примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. 1.n-Dimensional-Lists.dyn 2.n-Dimensional-Lists.sat

Это первое из трех упражнений, направленных на работу с импортированной геометрией. От упражнения к упражнению структура данных будет усложняться.

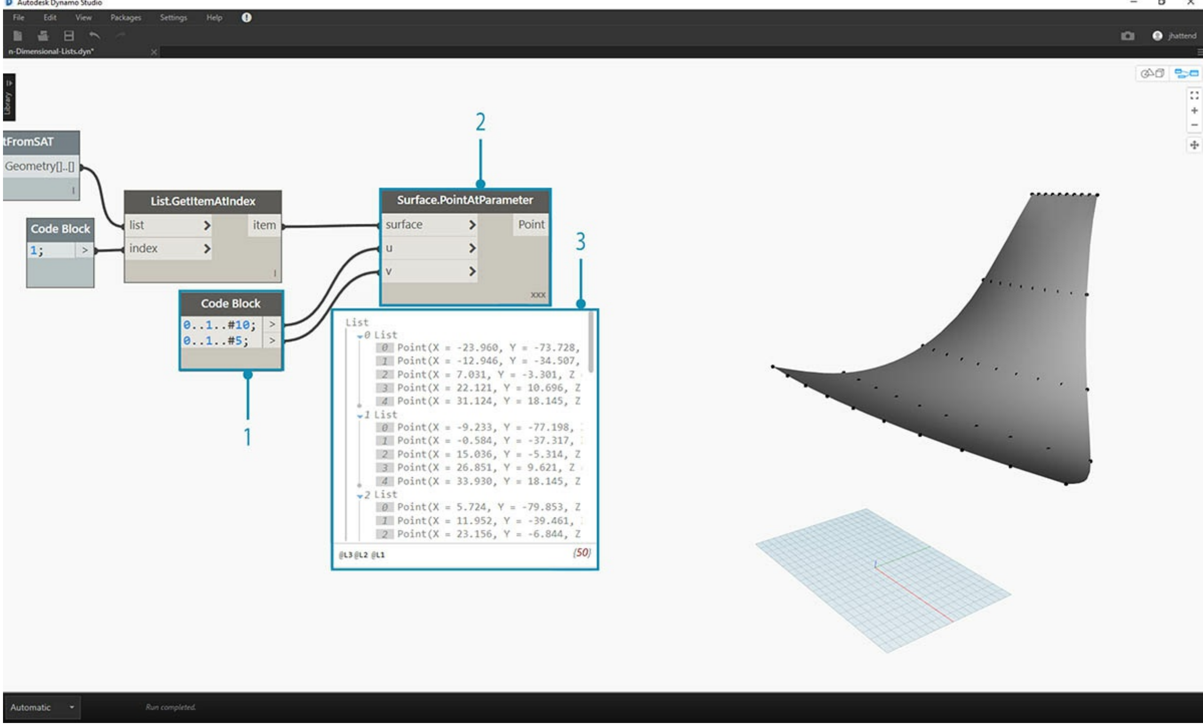


1. Начнем с файла SAT, расположенного в папке с файлами для упражнения. Добавим его в приложение с помощью узла *File Path*.
2. Узел *Geometry.ImportFromSAT* импортирует геометрию в Dynamo и отображает ее в виде двух поверхностей.



Для простоты в этом упражнении вы будете работать только с одной поверхностью.

1. Чтобы выбрать верхнюю поверхность, задайте индекс *1*. Для этого добавьте узел *List.GetItemAtIndex*.

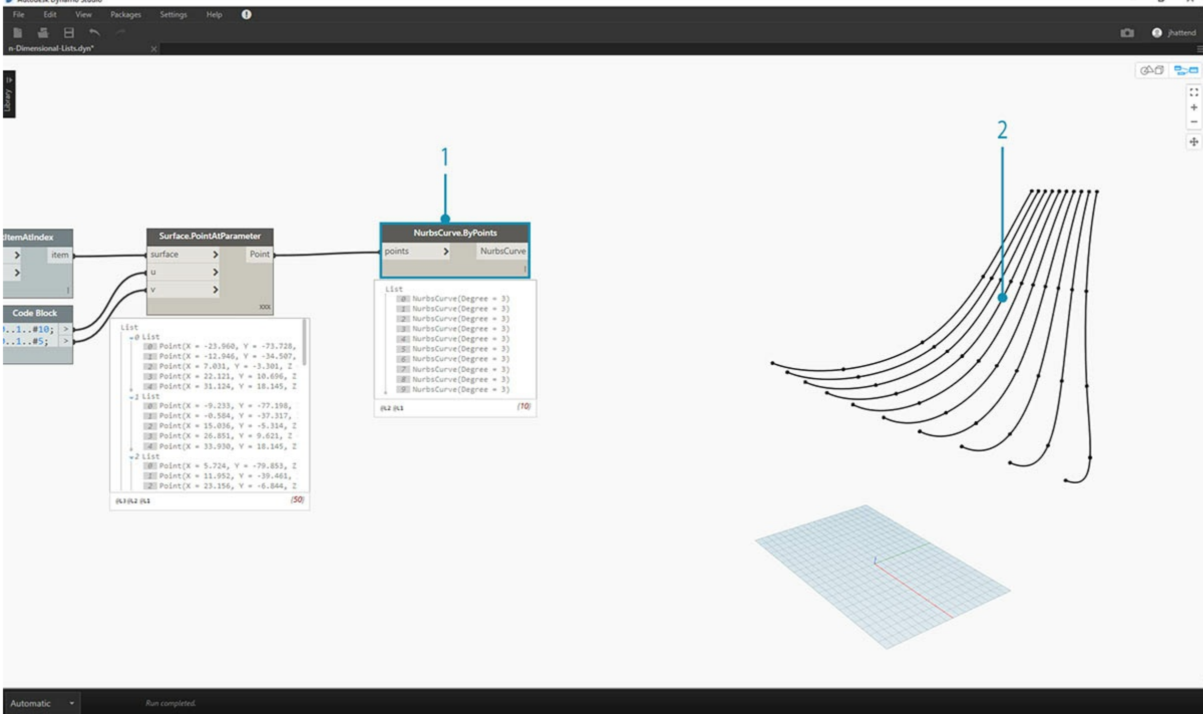


Теперь нужно преобразовать поверхность в сетку из точек.

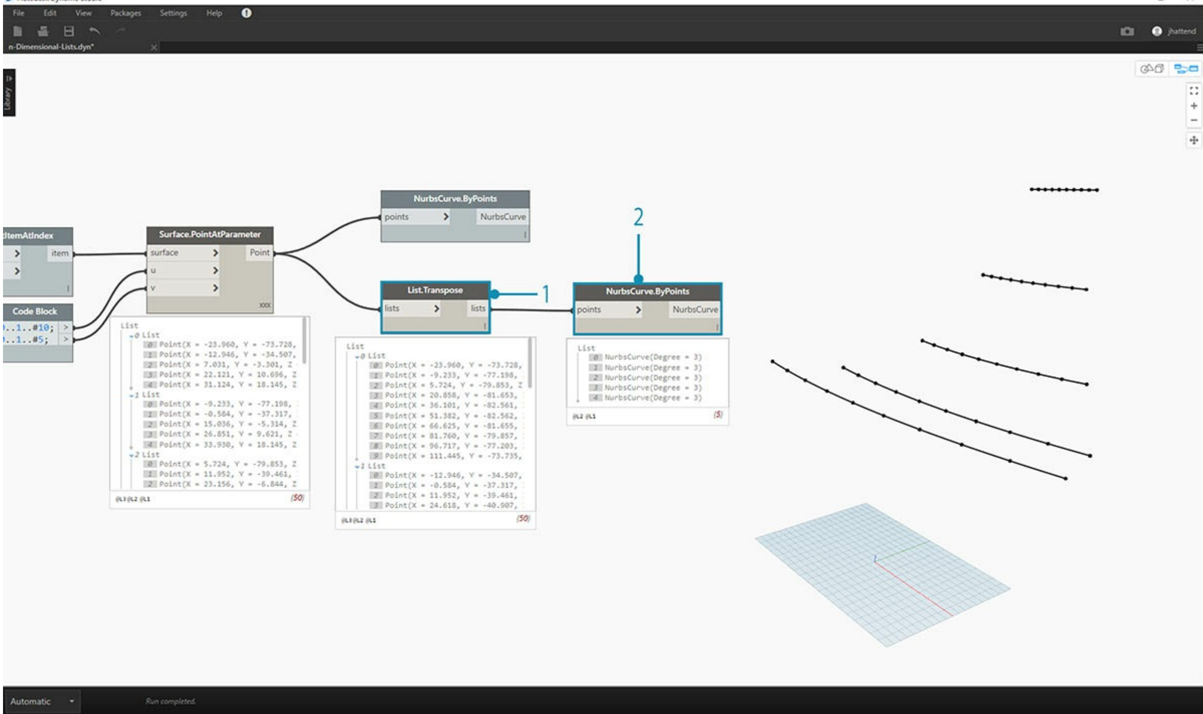
1. С помощью узла *Code Block* вставьте две следующие строки кода:

```
0..1..#10;
0..1..#5;
```

1. Используя узел *Surface.PointAtParameter*, соедините два значения Code Block с портами ввода *u* и *v*. Задайте для параметра *Переплетение* этого узла значение *декартово произведение*.
2. Полученная структура данных отображается в области предварительного просмотра Dynamo.



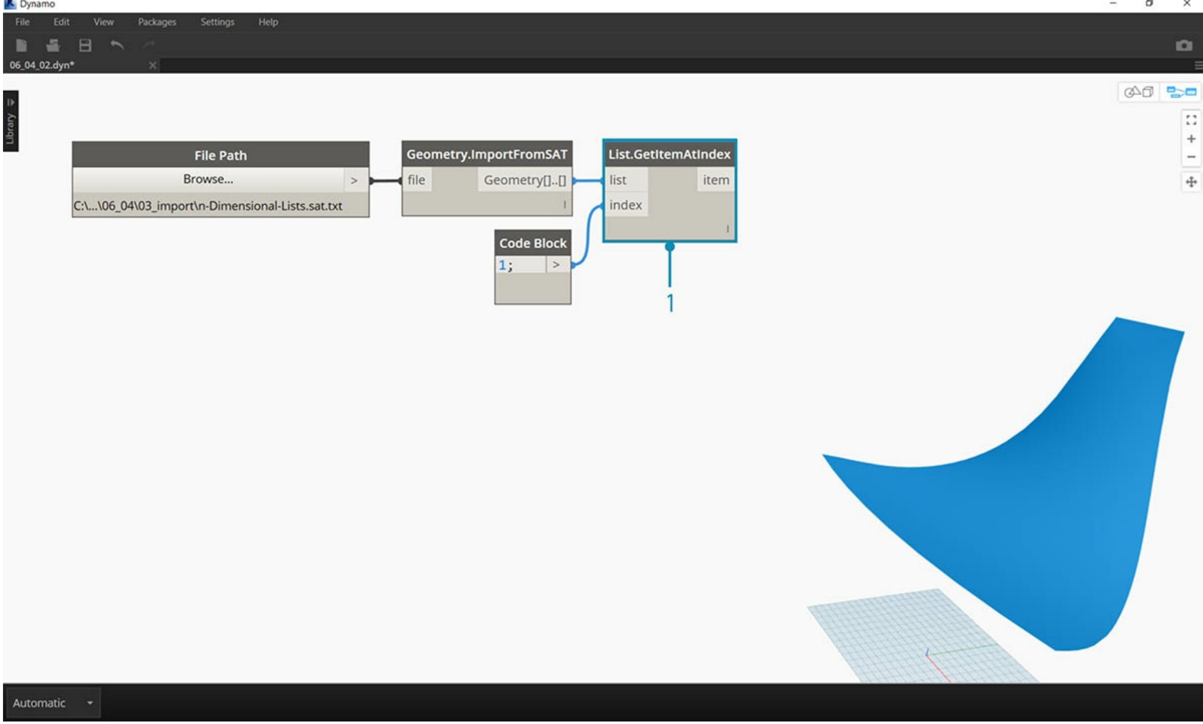
1. Чтобы отобразить структуру данных, соедините узел *NurbsCurve.ByPoints* с портом вывода узла *Surface.PointAtParameter*.
2. В области предварительного просмотра отображаются десять кривых, идущих вертикально вдоль поверхности.



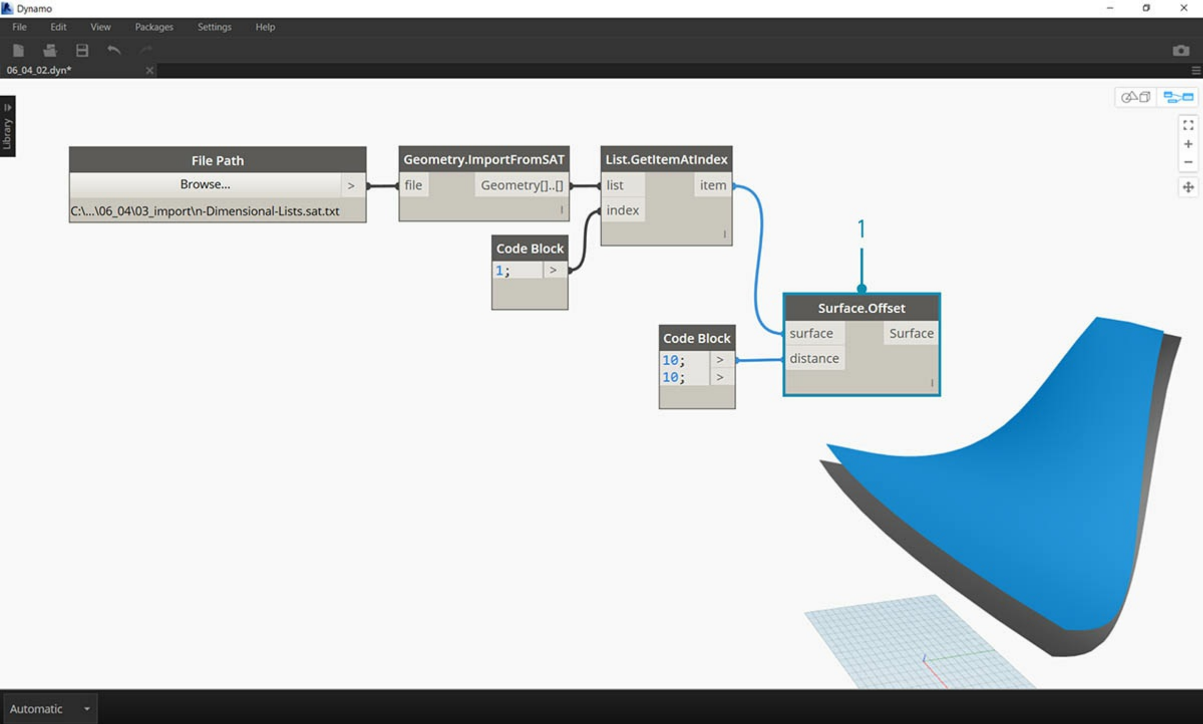
1. Базовый узел *List.Transpose* позволяет поменять местами столбцы и строки в списке списков.
2. При соединении порта вывода узла *List.Transpose* с узлом *NurbsCurve.ByPoints* вы получите пять кривых, идущих горизонтально вдоль поверхности.

Упражнение. 2D-списки. Для опытных пользователей

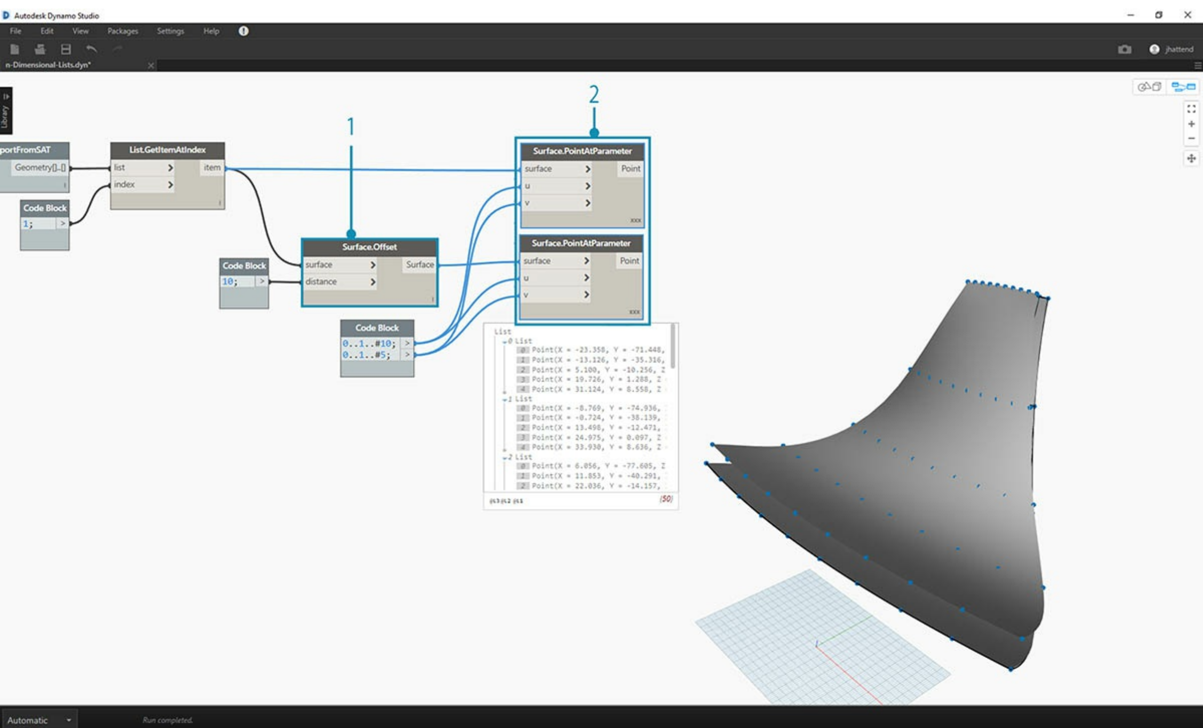
Усложним задачу. Предположим, что нам нужно выполнить определенное действие с кривыми, которые мы получили в предыдущем упражнении. Например, нужно связать эти кривые с другой поверхностью и выполнить лофтинг между ними По сути, логика остается прежней, но задача требует более внимательной работы со структурой данных.



1. Начнем с операции, уже знакомой вам по предыдущему упражнению. Изолируйте верхнюю поверхность импортированной геометрии с помощью узла *List.GetItemAtIndex*.



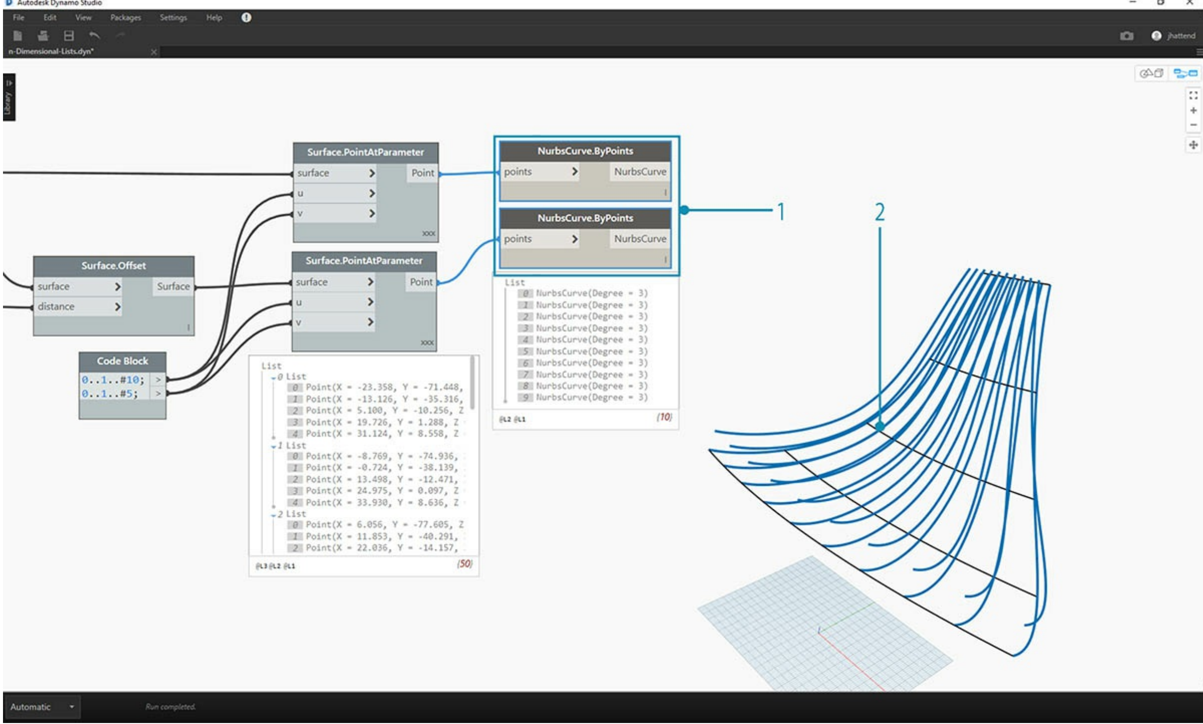
1. Используя узел *Surface.Offset*, задайте значение *10*, чтобы сместить поверхность.



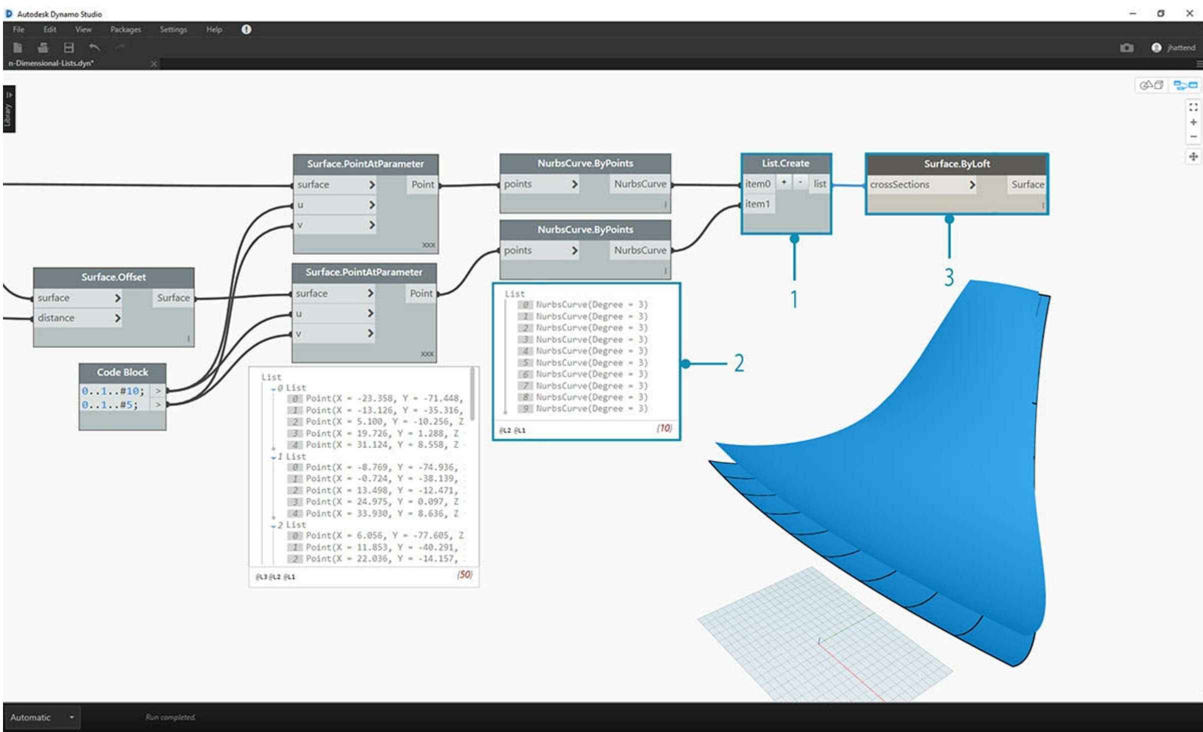
1. Как и в предыдущем упражнении, добавьте узел *Code Block* с двумя строками кода:

```
0..1..#10;
0..1..#5;
```

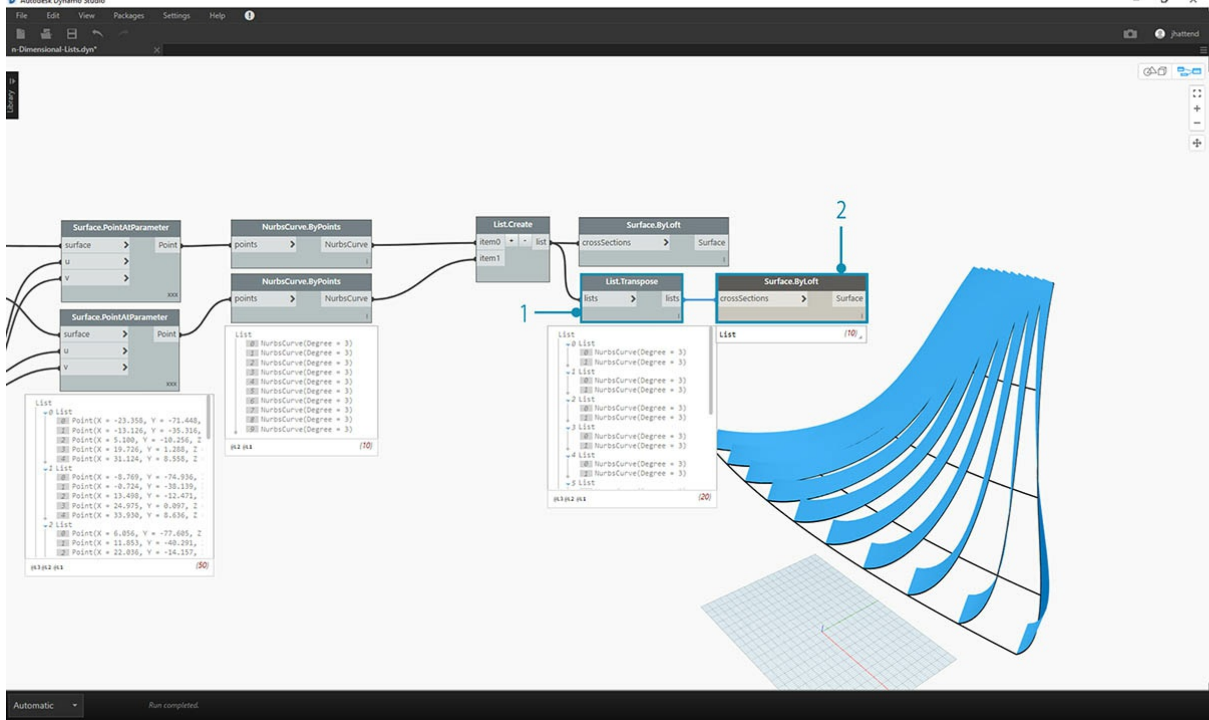
1. Соедините порты вывода этого узла с двумя узлами *Surface.PointAtParameter* и задайте для параметра *Переплетение* каждого из них значение *декартово произведение*. Один из этих узлов соединен с исходной поверхностью, а второй — с поверхностью смещения.



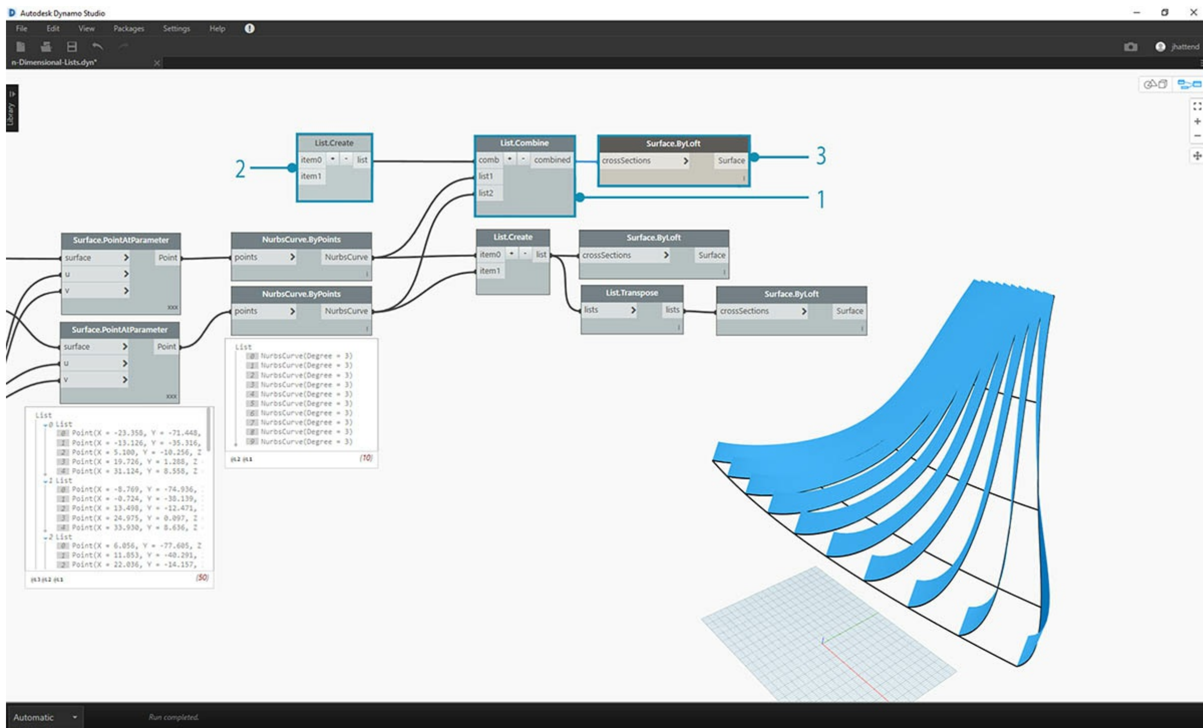
1. Как и в предыдущем упражнении, соедините порты вывода с двумя узлами *NurbsCurve.ByPoints*.
2. В области предварительного просмотра Dynamo отображаются две кривые, соответствующие двум поверхностям.



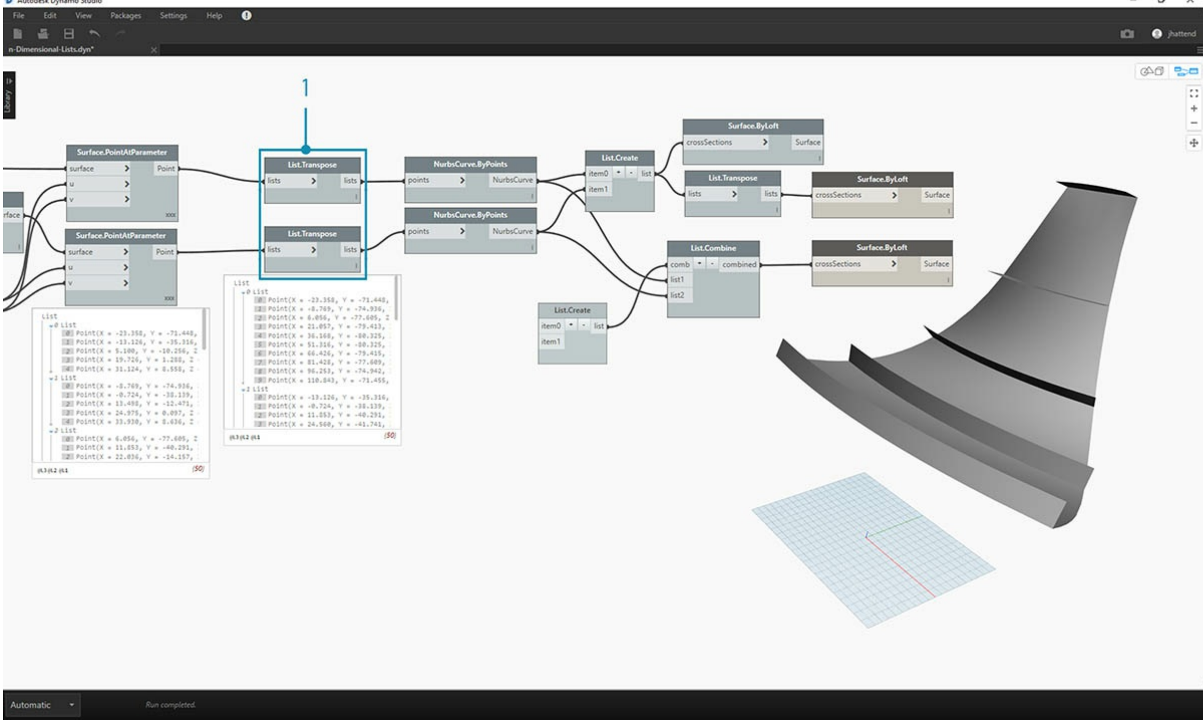
1. С помощью узла *List.Create* можно объединить два набора кривых в один список списков.
2. В результате создаются два списка с десятью элементами, каждый из которых представляет собой связанный набор NURBS-кривых.
3. С помощью узла *Surface.ByLoft* можно создать визуальное представление этой структуры данных. Узел выполняет лофтинг для всех кривых в каждом списке.



1. Как вы помните, узел *List.Transpose* позволяет поменять местами столбцы и строки в списке списков. В результате использования этого узла два списка из десяти кривых каждый преобразуются в десять списков из двух кривых каждый. Теперь каждая NURBS-кривая связана с соседней кривой на другой поверхности.
2. С помощью узла *Surface.ByLoft* мы получили реберную конструкцию.



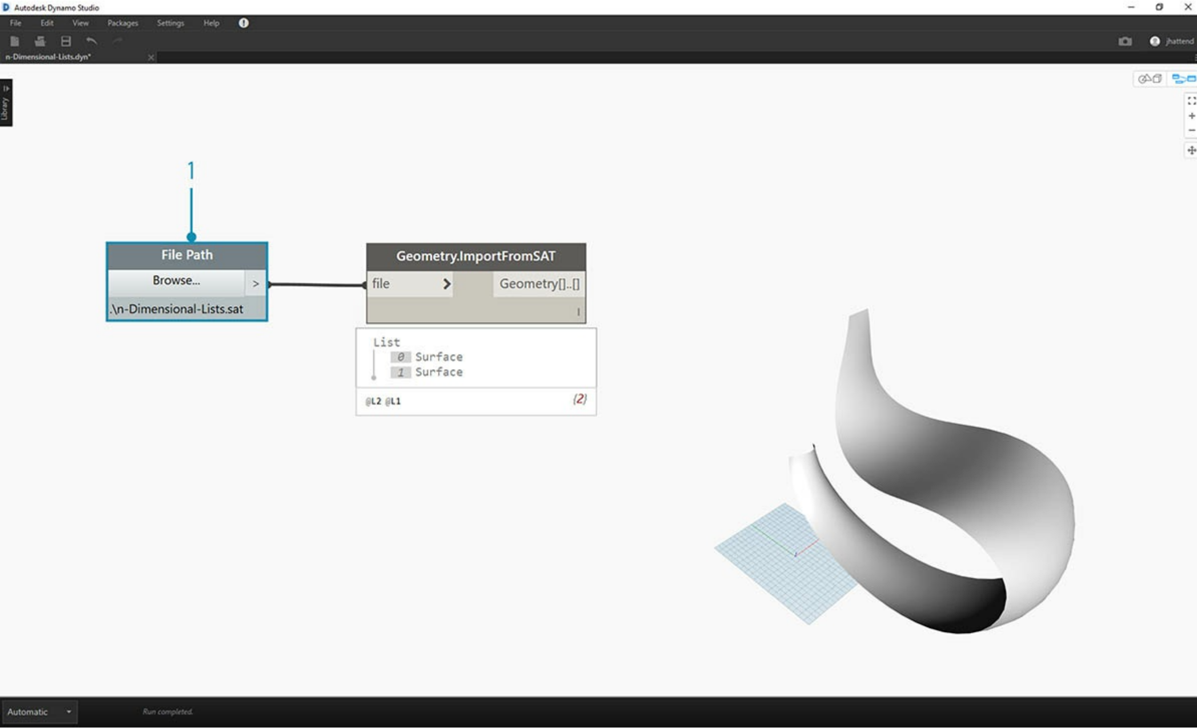
1. Вместо узла *List.Transpose* можно использовать узел *List.Combine*. Он выполняет роль *«обьединителя»* для каждого вложенного списка.
2. В данном случае мы используем *List.Create* в качестве *«обьединителя»* для создания списка по каждому элементу во вложенных списках.
3. Добавив узел *Surface.ByLoft*, мы получаем те же поверхности, что и в предыдущем шаге. В данном случае узел Transpose является более простым вариантом, но при работе с еще более сложной структурой данных надежнее будет использовать узел *List.Combine*.



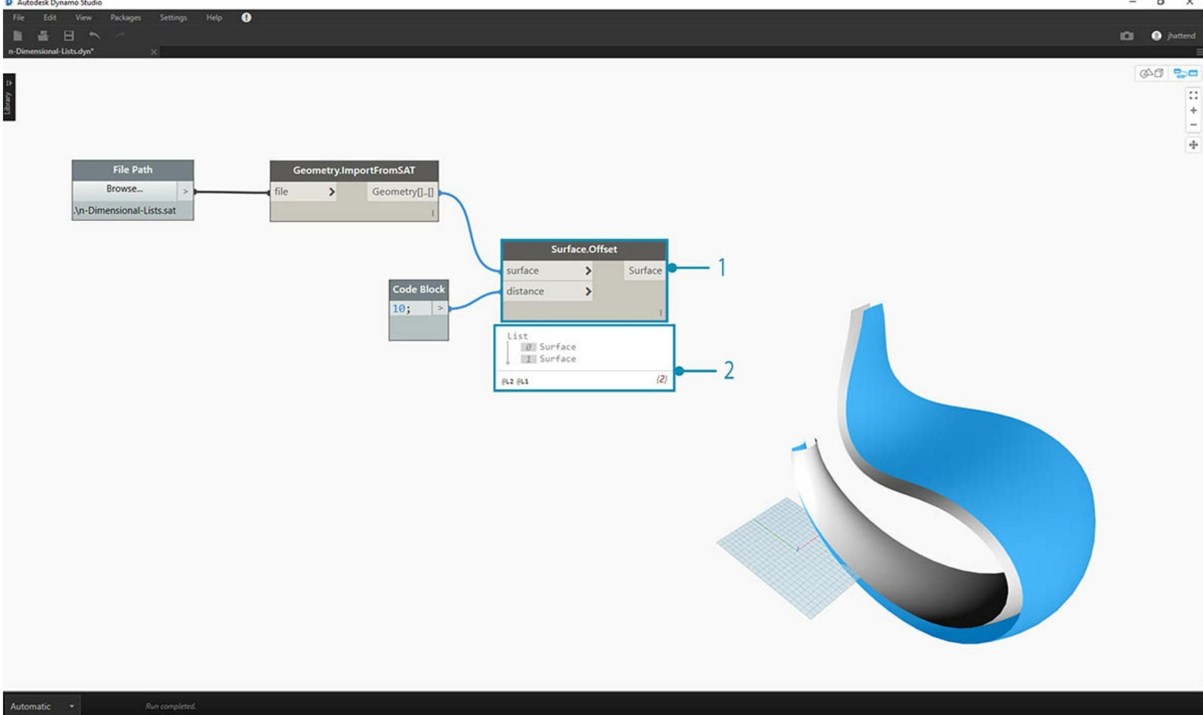
1. Вернемся на несколько шагов назад. Если вы хотите изменить ориентацию кривых в реберной конструкции, узел List.Transpose следует применить до соединения с узлом *NurbsCurve.ByPoints*. В результате столбцы и строки поменяются местами, и мы получим пять горизонтальных ребер.

Упражнение. Трехмерные списки

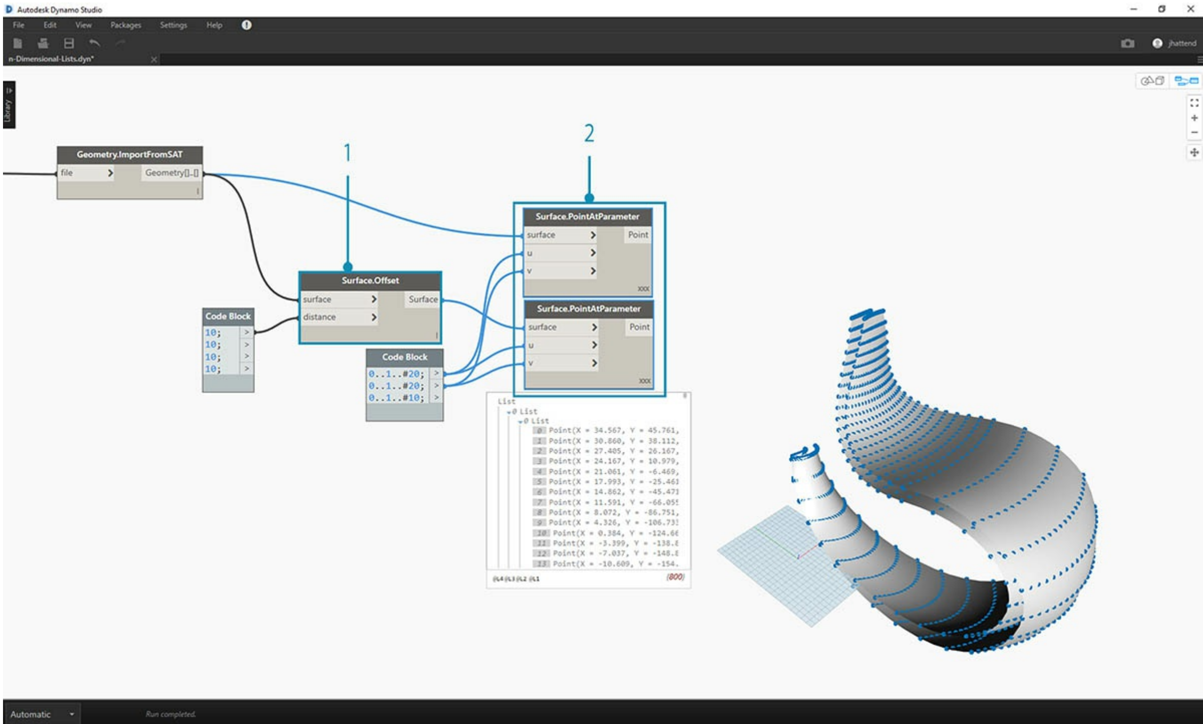
Продолжаем усложнять задачи. В этом упражнении мы используем обе импортированные поверхности, чтобы создать сложную иерархическую структуру данных. По сути, вам предстоит выполнить то же самое действие, пользуясь той же самой логикой, что и ранее.



1. Вернемся к файлу, импортированному в предыдущем упражнении.



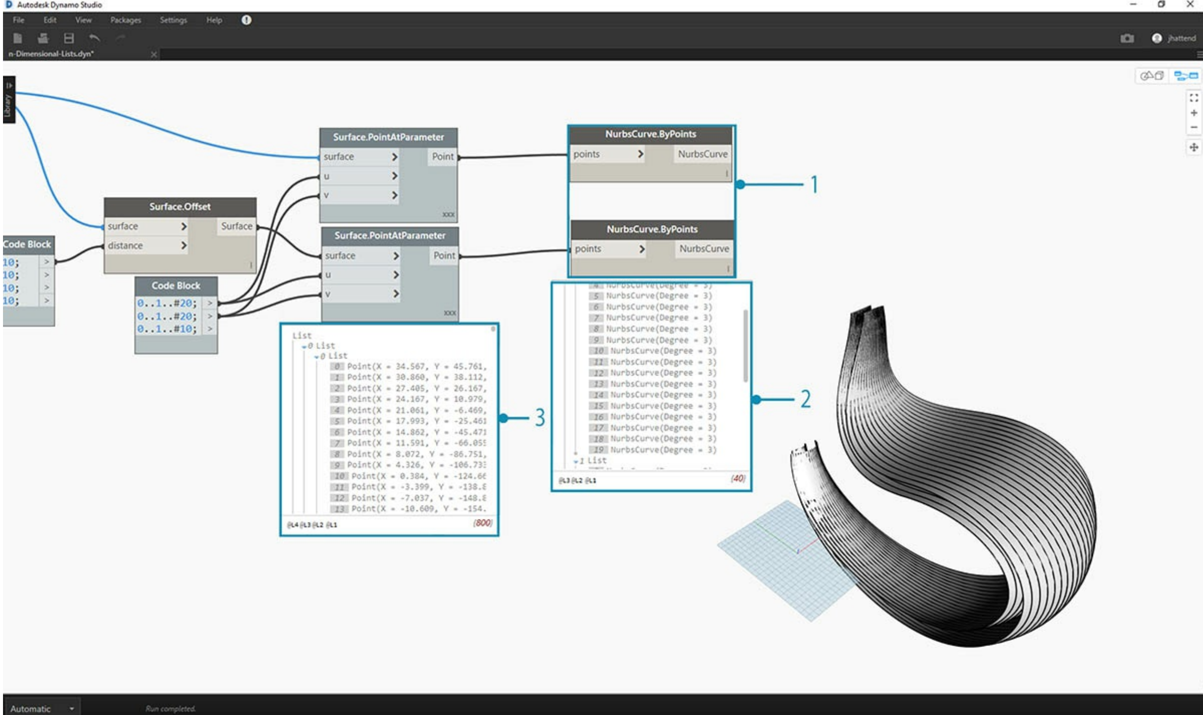
1. Как и в предыдущем упражнении, используйте узел *Surface.Offset*, чтобы задать значение смещения, равное *10*.
2. Обратите внимание, что добавление узла смещения привело к созданию двух поверхностей.



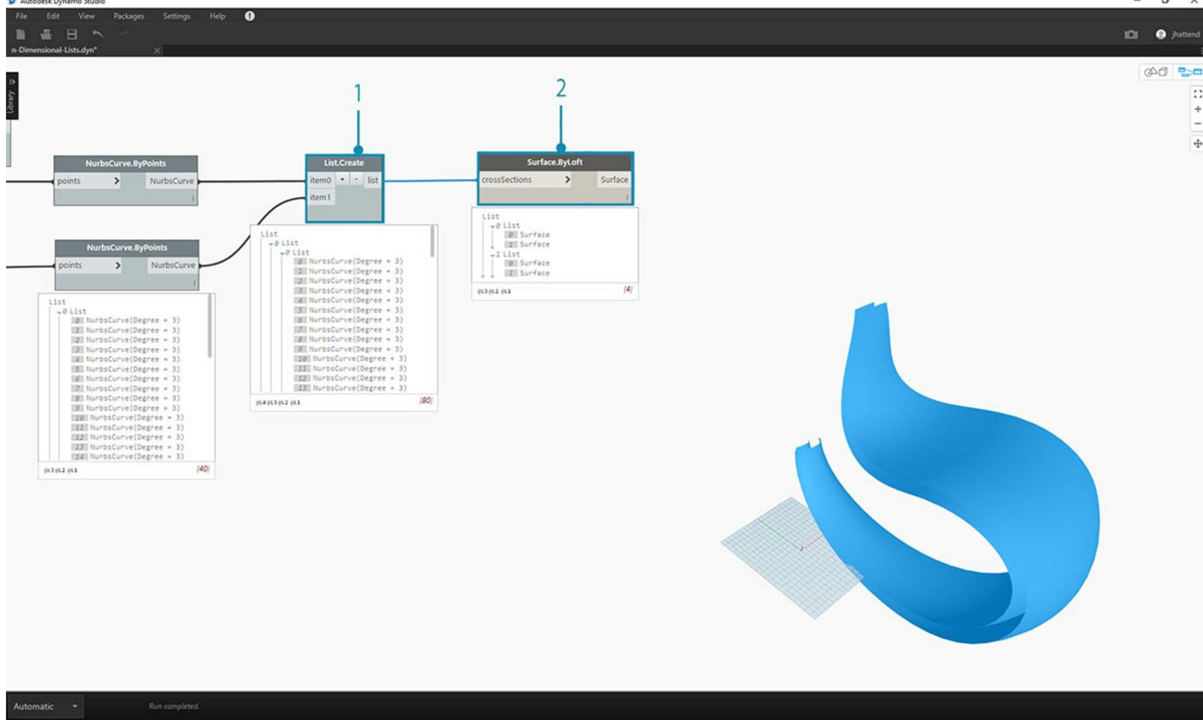
1. Как и в предыдущем упражнении, добавьте узел Code Block с двумя строками кода:

```
0..1..#20;
0..1..#10;
```

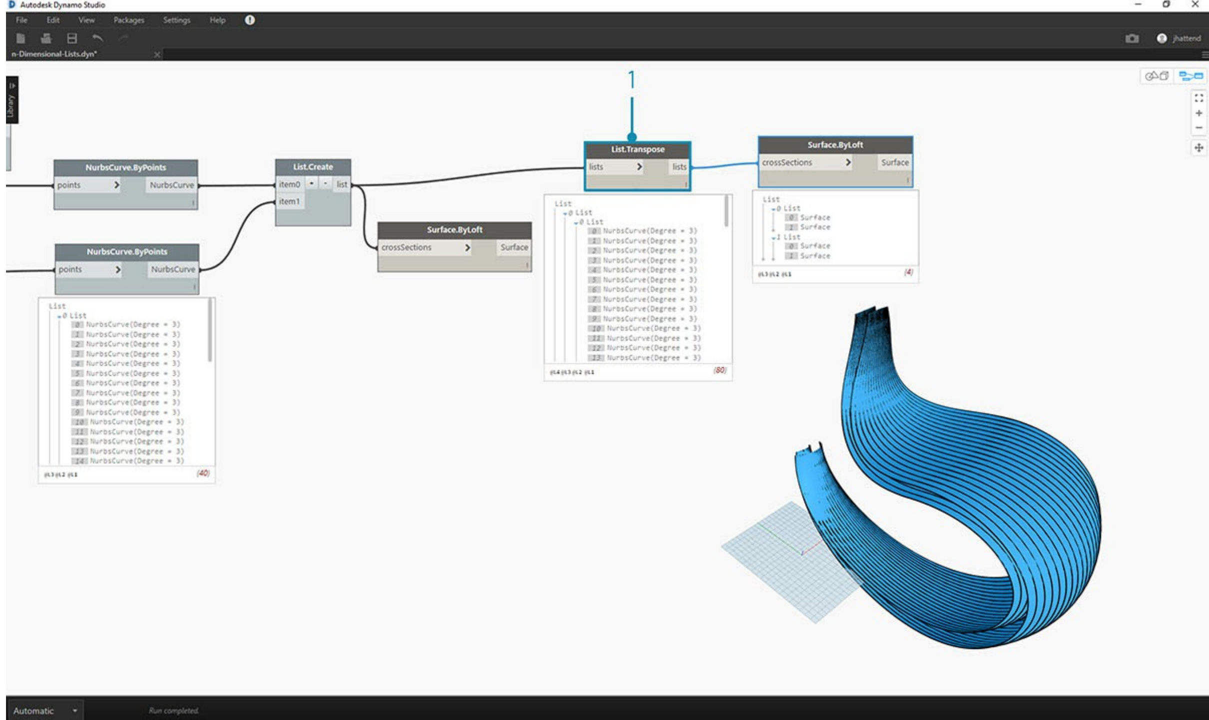
1. Соедините порты вывода этого узла с двумя узлами *Surface.PointAtParameter* и задайте для параметра «Переплетение» каждого из них значение *декартово произведение*. Один из этих узлов соединен с исходными поверхностями, а второй — с поверхностями смещения.



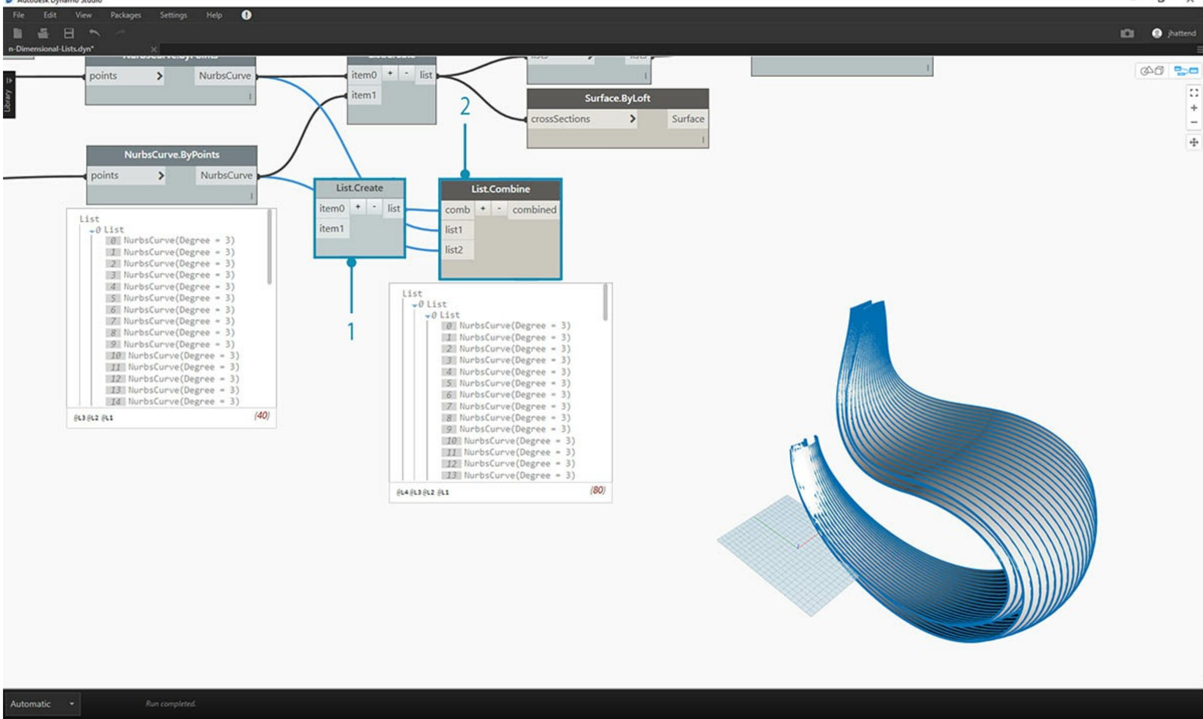
1. Как и в предыдущем упражнении, соедините порты вывода с двумя узлами *NurbsCurve.ByPoints*.
2. Посмотрите на выходные данные узла *NurbsCurve.ByPoints* и обратите внимание, что они представляют собой список, состоящий из двух списков, что является более сложной структурой, чем в предыдущем упражнении. Данные упорядочиваются по базовой поверхности, поэтому в структуру данных добавлен еще один уровень.
3. Обратите внимание, что структура данных в узле *Surface.PointAtParameter* стала более сложной. В нем представлен список, состоящих из списков списков.



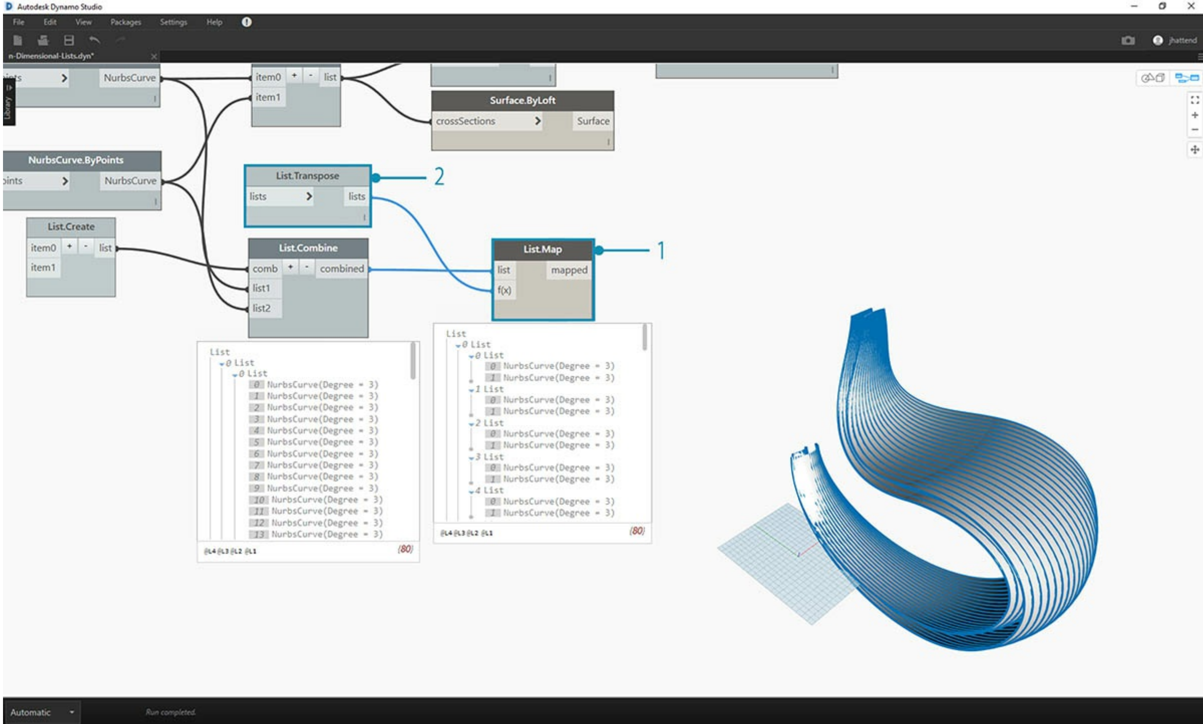
1. С помощью узла *List.Create* объедините NURBS-кривые в одну структуру данных, чтобы создать список, состоящий из списков списков.
2. При подключении узла *Surface.ByLoft* мы получаем новую версию исходных поверхностей, так как они остаются в собственном списке в соответствии с исходной структурой данных.



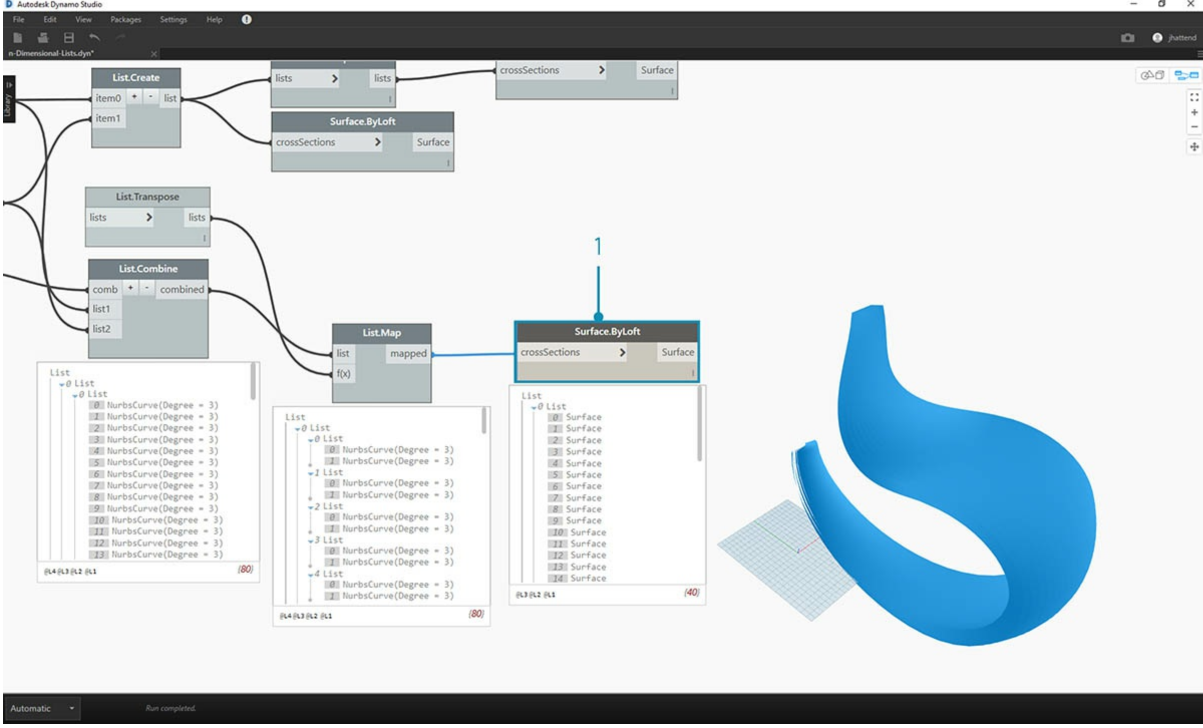
1. В предыдущем упражнении мы использовали узел *List.Transpose* для создания реберной конструкции. В этом случае данная функция не подходит. Перенос следует использовать с двумерными списками, но мы имеем дело с трехмерным списком, поэтому перестановка столбцов и строк не сработает. Поскольку списки являются объектами, то узел *List.Transpose* выполнит перестановку между списками с вложенными списками, но она не затронет NURBS-кривые в списках на уровень ниже.



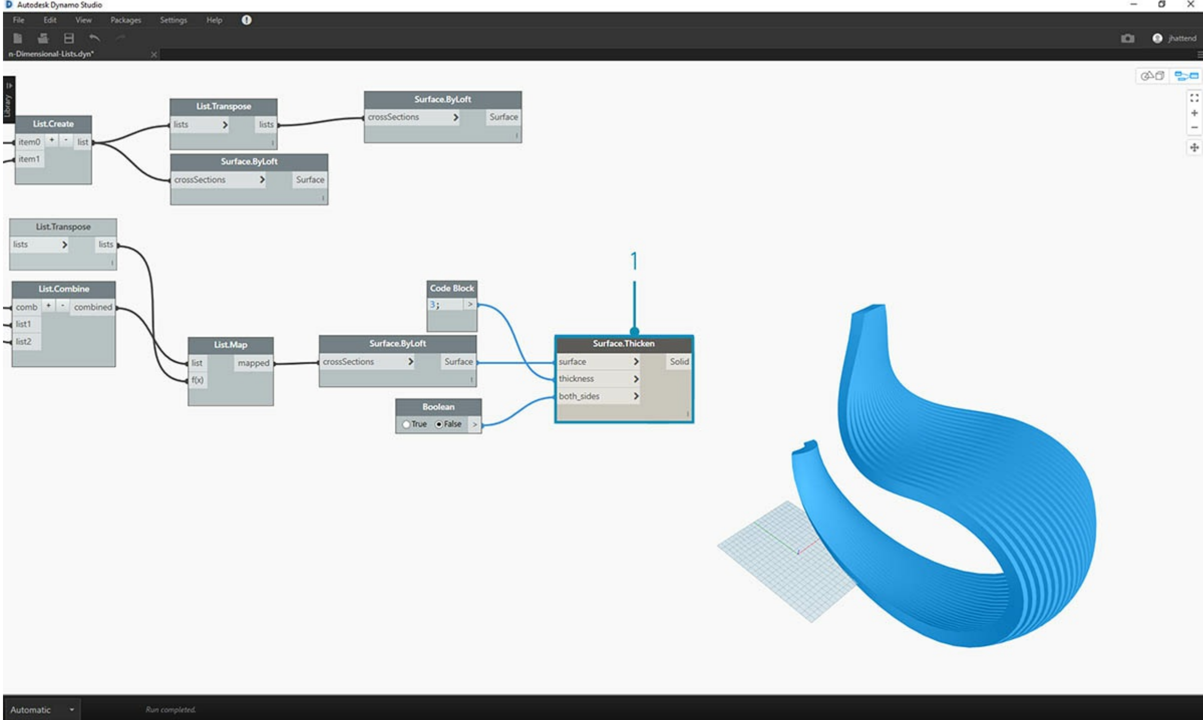
1. В этом случае *List.Combine* является более подходящим инструментом. При работе с более сложными структурами данных используются узлы *List.Map* и *List.Combine*.
2. Используя *List.Create* в качестве *«объединителя»*, создайте структуру данных, которая лучше подойдет для ваших целей.



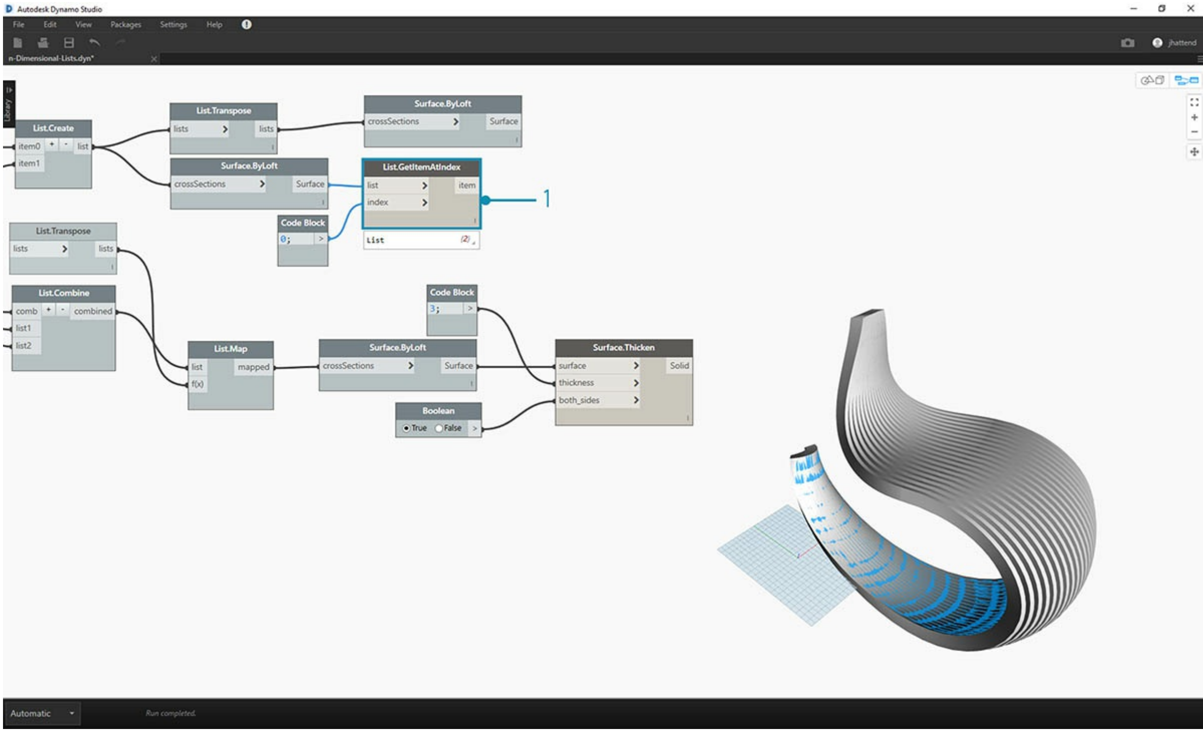
1. Структуру данных все еще требуется перенести на один уровень вниз по иерархии. Для этого используйте узел *List.Map*. Его работа аналогична узлу *List.Combine*, однако в нем используется только один список входных данных, а не два или больше.
2. К узлу *List.Map* будет применена функция *List.Transpose*, которая меняет местами столбцы и строки вложенных списков в главном списке.



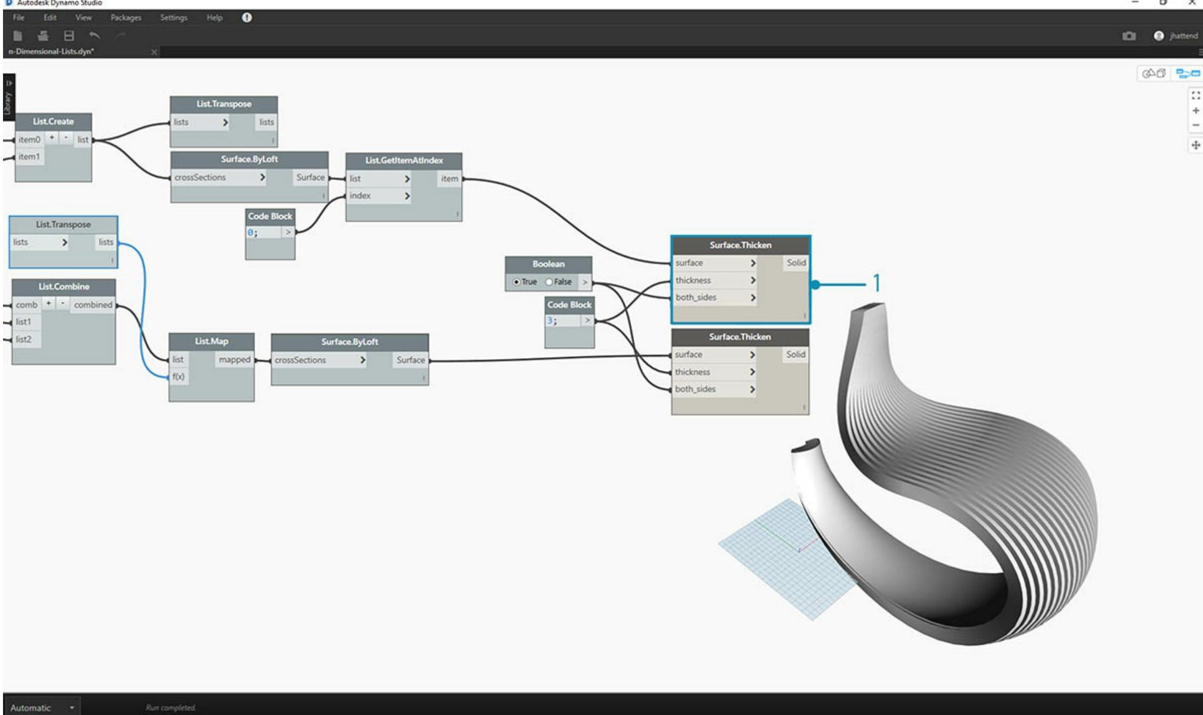
1. Наконец, выполните лофтинг между NURBS-кривыми с использованием соответствующей иерархии данных, чтобы получить реберную конструкцию.



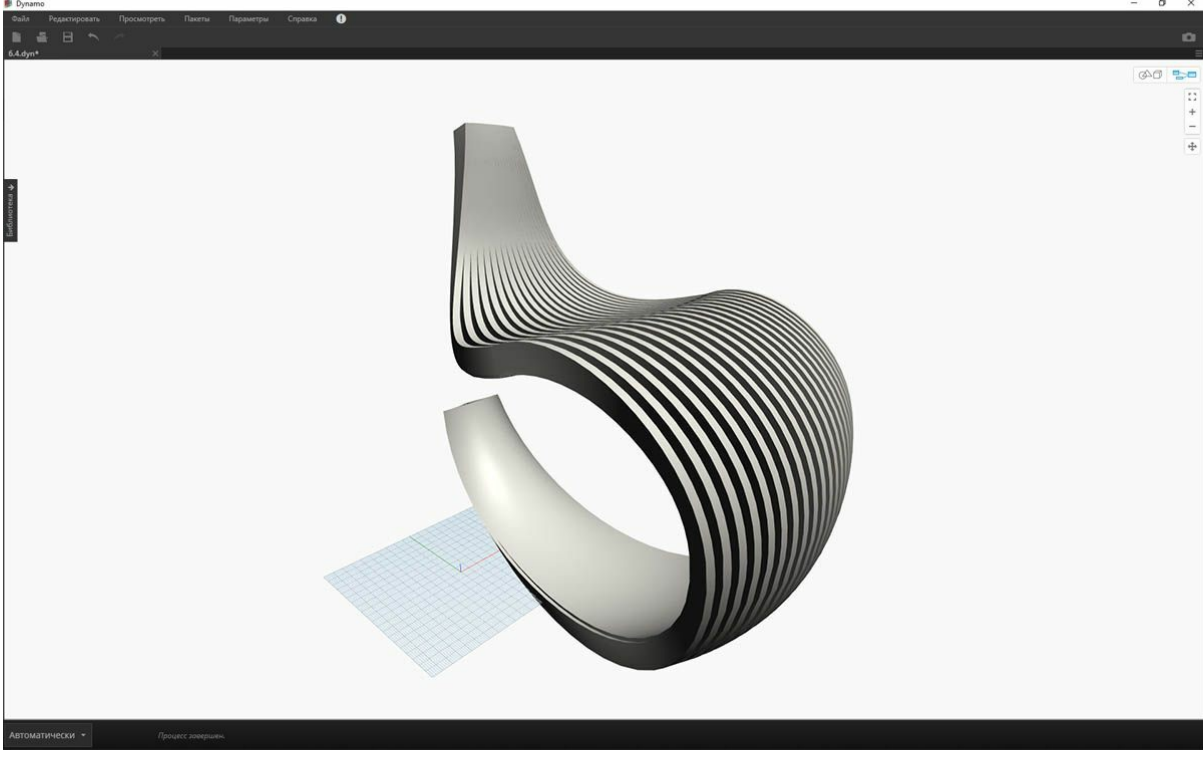
1. Придайте геометрии глубину с помощью узла *Surface.Thicken*.



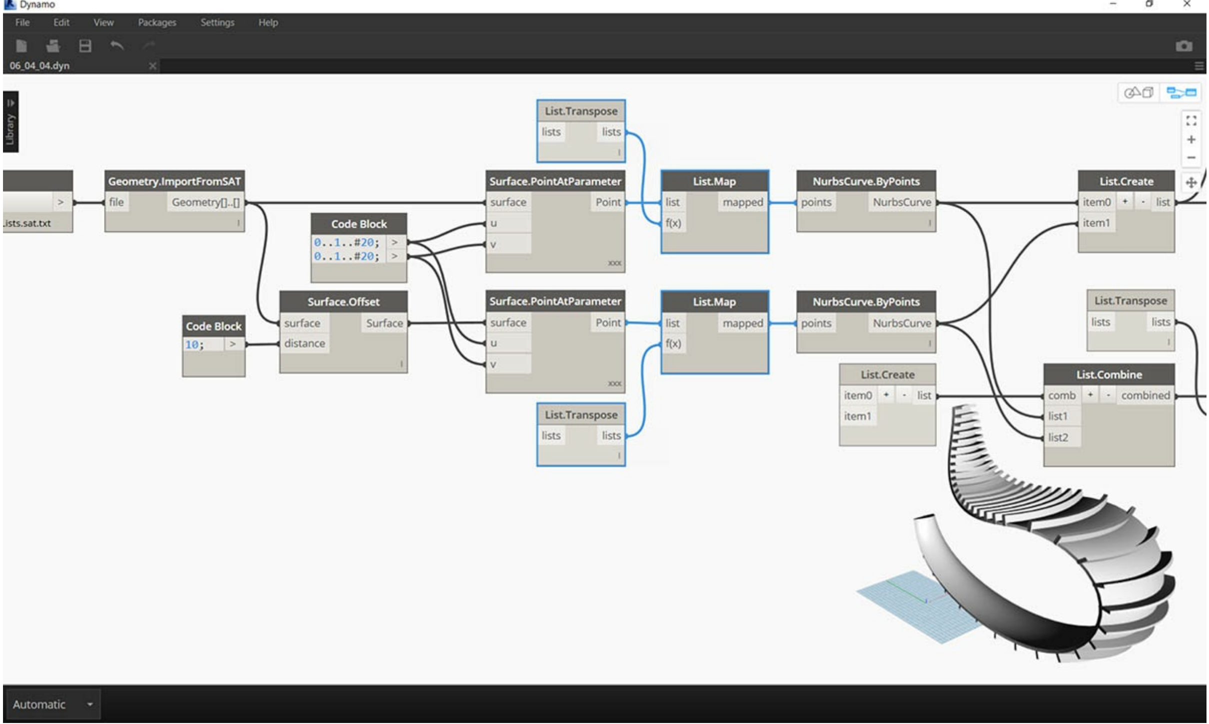
1. Добавим вспомогательную поверхность для этих двух конструкций. Используйте узел *List.GetItemAtIndex*, чтобы выбрать заднюю поверхность из поверхностей лофтинга, созданных в предыдущих шагах.



1. Теперь увеличьте толщину выбранных поверхностей.

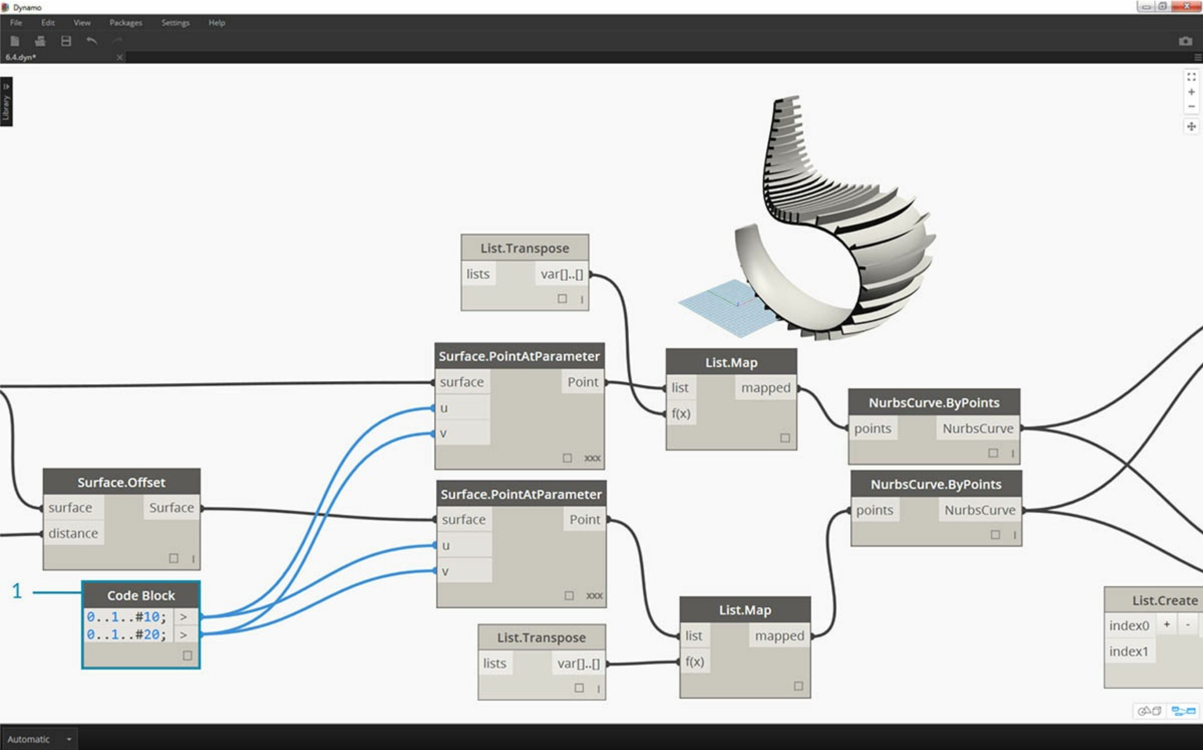


В результате мы получили нечто, похожее на слегка неустойчивое кресло-качалку. Зато сколько данных ушло на его создание!



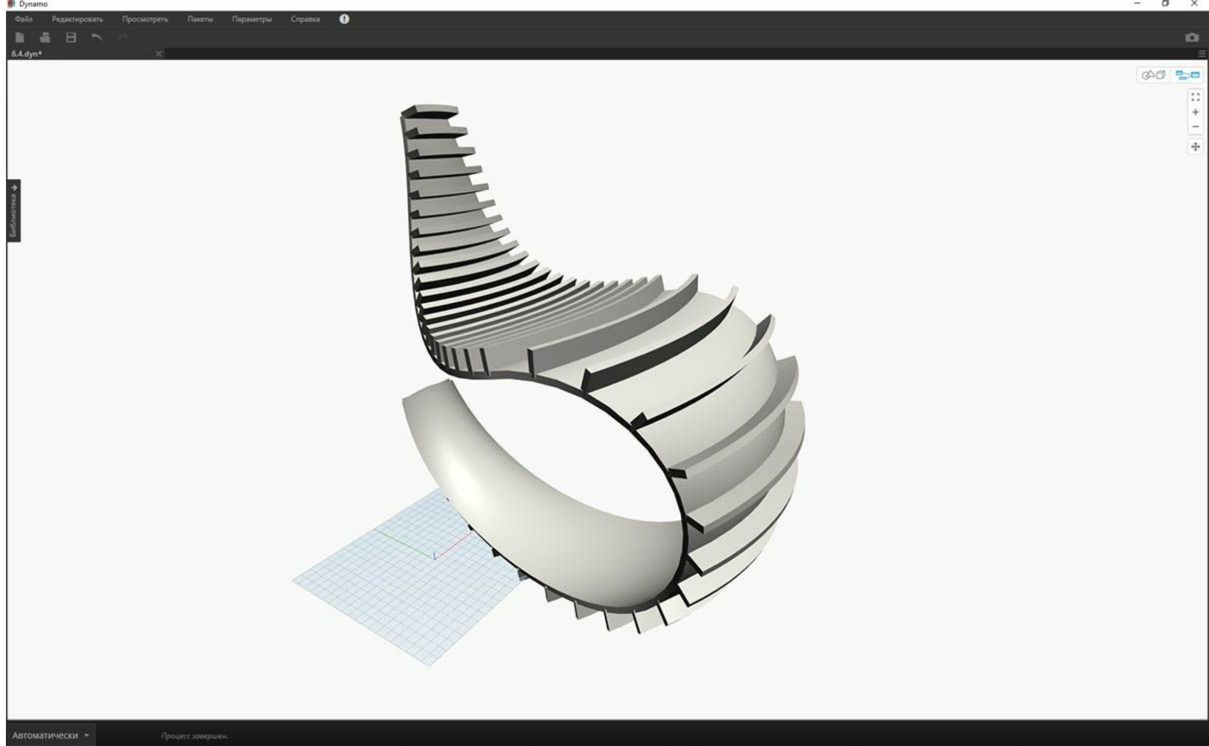
Наконец, изменим направление бороздок. Для этого выполните процедуру, аналогичную преобразованию, которое вы уже использовали ранее.

1. Так как здесь присутствует еще один уровень иерархии, то используйте узел *List.Map* с функцией *List.Tranpose*, чтобы изменить направление NURBS-кривых.



1. Если требуется увеличить количество канавок, то данные узла Code Block можно изменить на следующие:

```
0..1..#20;
0..1..#10;
```

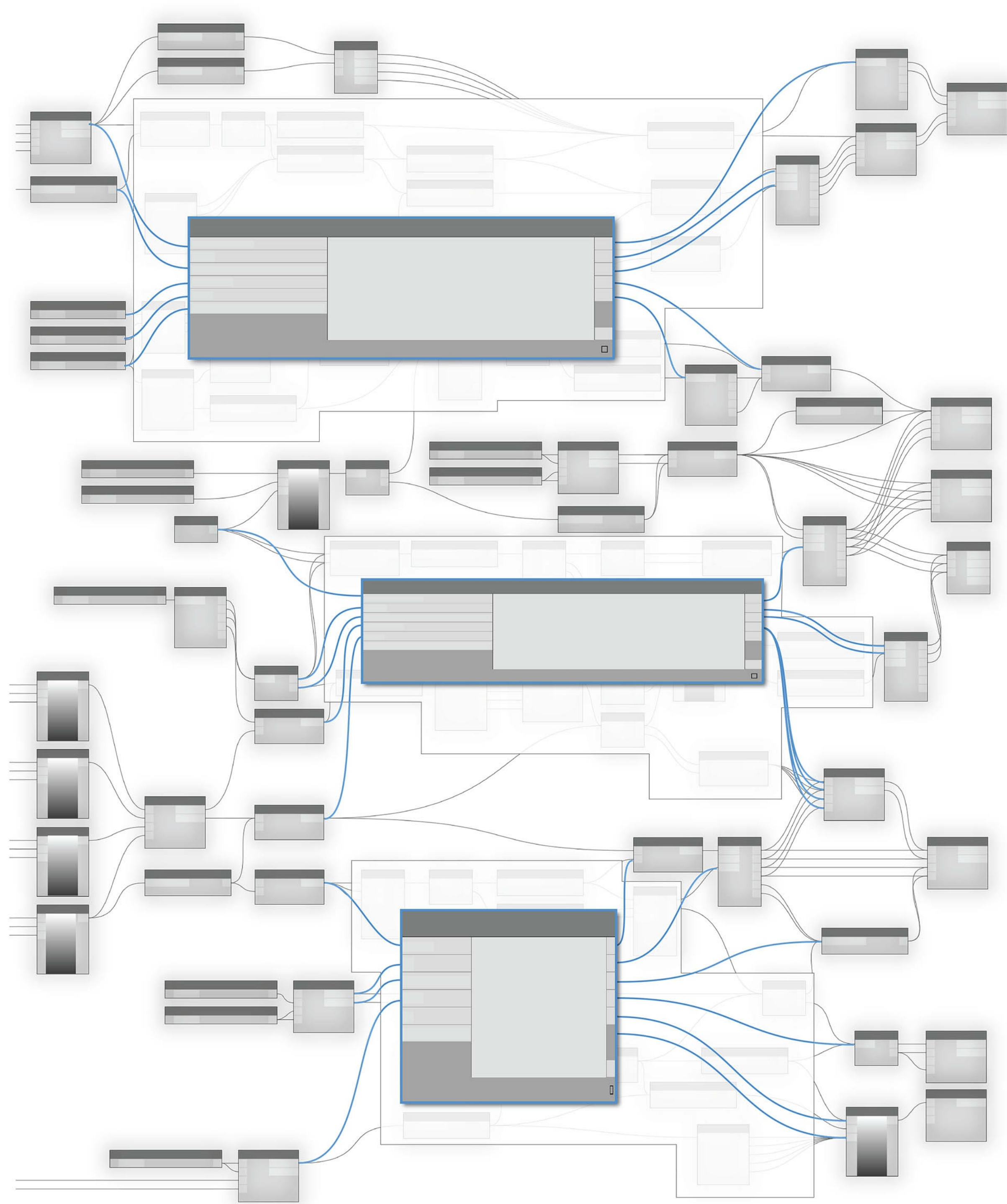


Если первая версия кресла-качалки была обтекаемой, то вторая получилась более похожей на колесо внедорожника.

Узлы Code Block и DesignScript

Узлы Code Block и DesignScript

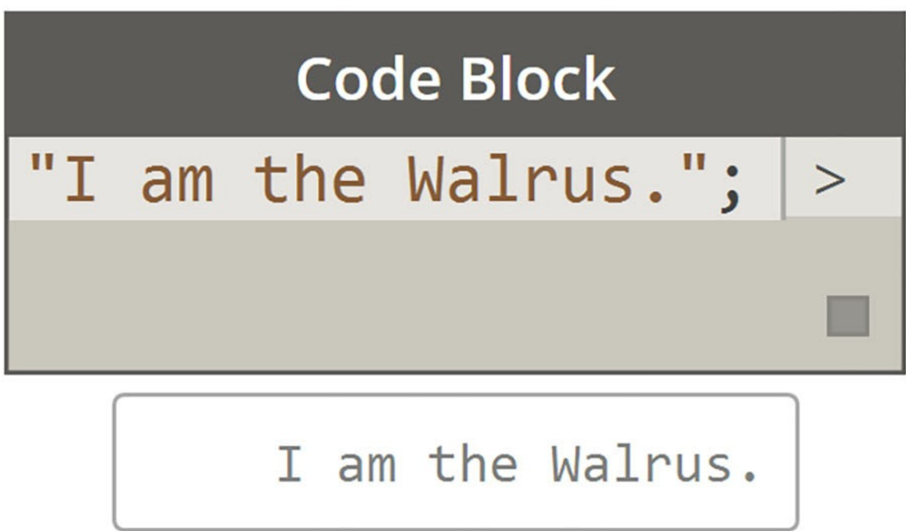
Блоки кода (Code Block) — это уникальная функция Dynamo, обеспечивающая динамическую связь между визуальной и текстовой средами программирования. Блоки кода можно подключать ко всем узлам Dynamo, определяя с их помощью весь график в одном узле. Внимательно ознакомьтесь с этой главой, поскольку блок кода — это основополагающий компонент Dynamo.



Что такое Code Block

Что такое Code Block

Блоки кода — это своеобразные окна, позволяющие заглянуть в самую глубь языка DesignScript, лежащего в основе Dynamo. DesignScript — это язык программирования, созданный специально для использования в рамках рабочих процессов проектирования. Это понятный и удобный для чтения язык, который позволяет моментально получать обратную связь при работе с небольшими частями кода, а также поддерживает масштабирование для работы с большими и сложными функциями. Кроме того, DesignScript — это основа мощного вычислительного «мотора» Dynamo. Поскольку почти все функциональные возможности узлов и операций Dynamo имеют прямые аналоги в языке создания сценариев, пользователи получают уникальную возможность свободно переходить от работы с узлами к работе со сценарием.

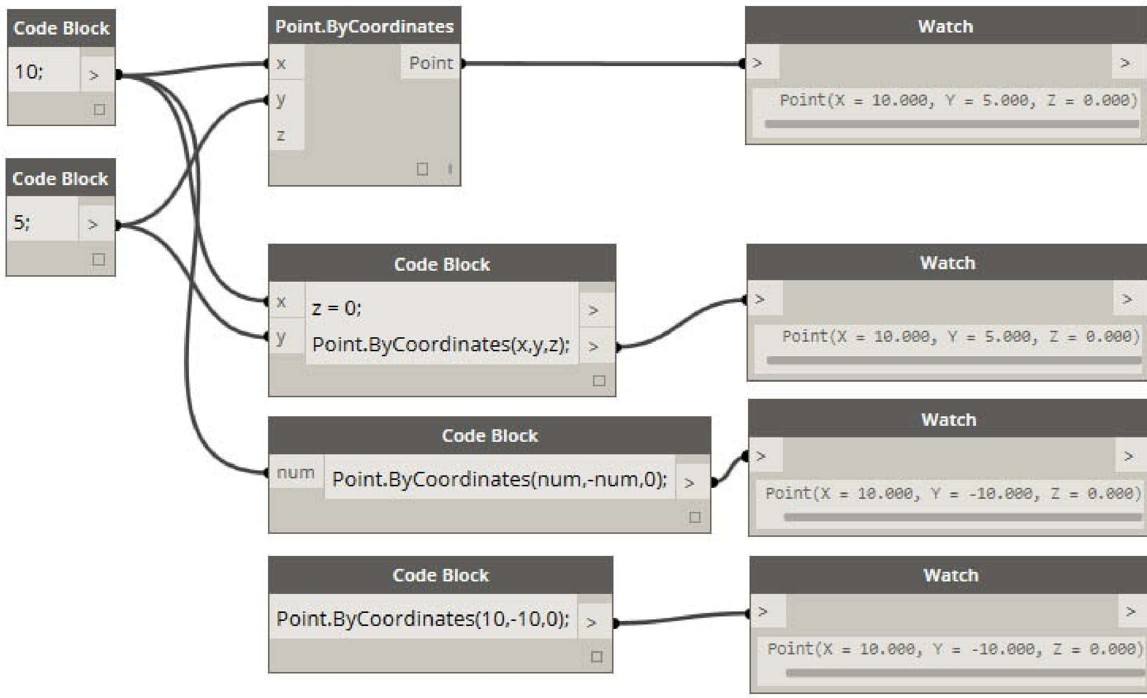


Для удобства начинающих пользователей узлы можно автоматически преобразовать в текстовый синтаксис, чтобы упростить обучение работе с DesignScript или просто уменьшить слишком большие разделы графиков. Это делается с помощью процедуры, называемой «от узла к коду», которая подробно описана в разделе Синтаксис DesignScript. Опытные пользователи могут применять узлы Code Block для создания уникальных комбинаций существующих функций и пользовательских связей на основе разнообразных стандартных парадигм работы с кодом. Все без исключения пользователи, от начинающих до опытных, могут воспользоваться широким спектром упрощенных методов и фрагментов кода, позволяющих ускорить процесс проектирования. Хотя термин «блок кода» может вызывать робость у пользователей, не занимающихся программированием, на деле эта функция столь же проста в использовании, сколь и эффективна. Начинающий пользователь может работать с блоками кода, практически не прибегая к программированию, а опытный — задавать определения на основе сценариев для вызова из других разделов определения Dynamo.

Code Block: краткий обзор

Говоря простым языком, блоки кода — это интерфейс для текстовой работы со сценарием в среде визуального программирования. Эти блоки можно использовать для задания чисел, строк, формул и других типов данных. Поскольку блоки кода разработаны специально для использования в Dynamo, пользователи могут задавать переменные в блоках кода, и эти переменные будут автоматически добавлены к портам ввода узла.

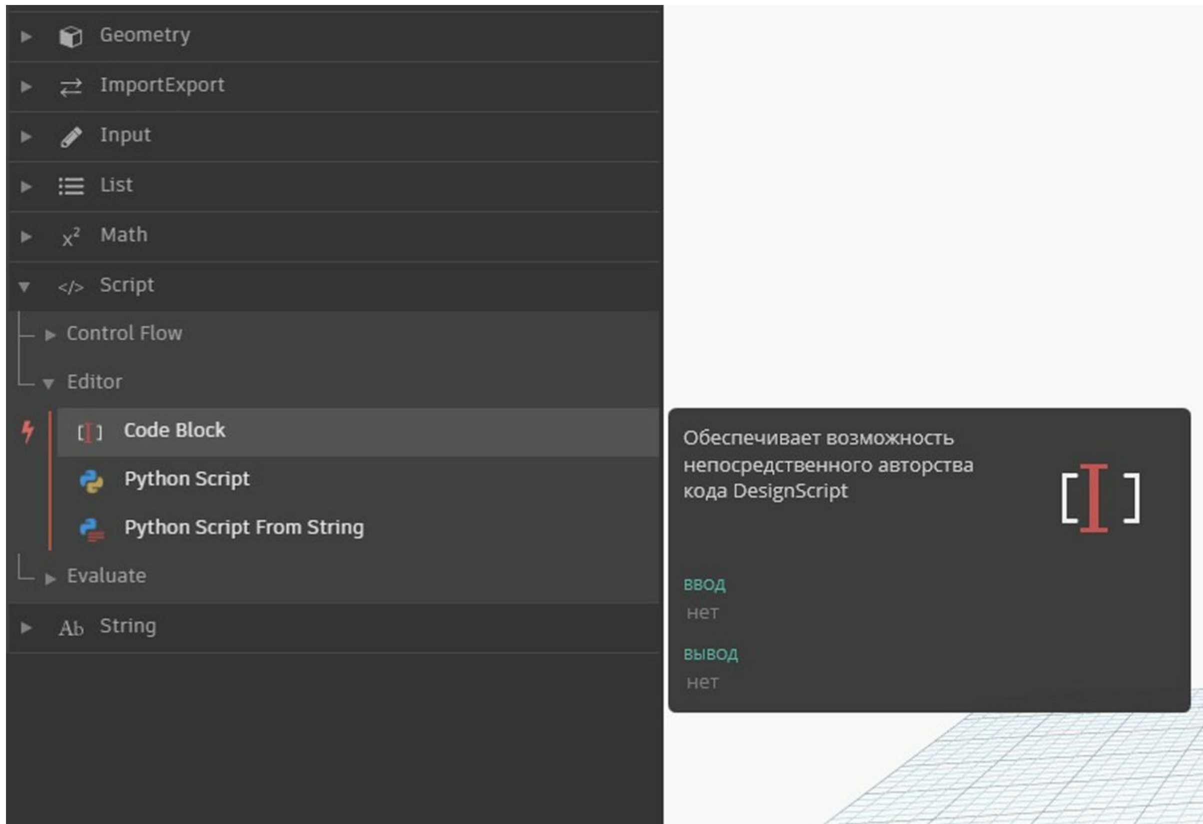
При работе с блоками кода пользователю предоставляется возможность определить, каким образом будут заданы входные данные. Ниже приведены способы построения базовой точки с координатами *(10, 5, 0)*:



По мере изучения различных функций, доступных в библиотеке, вы можете обнаружить, что ввести в поле поиска Point.ByCoordinates гораздо быстрее, чем искать нужный узел по разделам библиотеки вручную. При вводе запроса *Point.*, например, в Dynamo отобразится список подходящих функций, которые можно применить к объекту Point. Это делает процесс работы со сценариями интуитивно понятным и упрощает использование функций в Dynamo.

Создание узлов Code Block

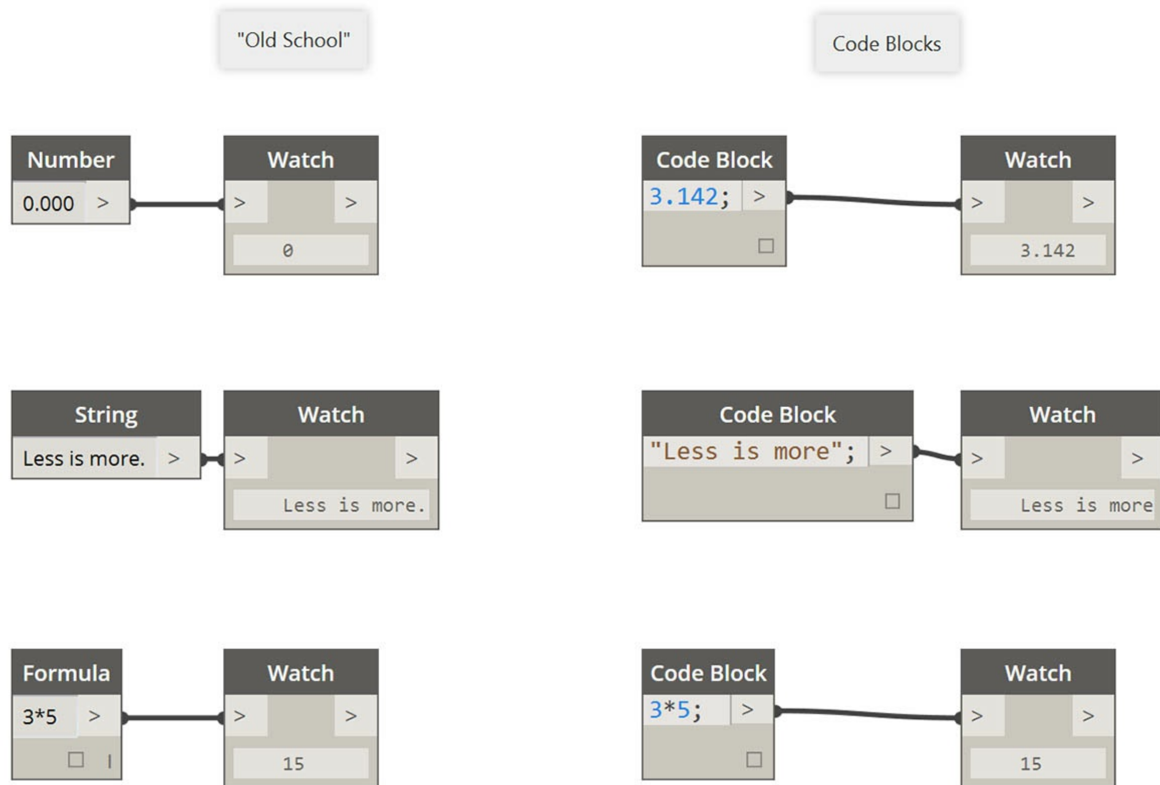
Для доступа к блокам кода перейдите в раздел *Core >Input >Actions >Code Block*. Есть и более быстрый способ: просто дважды щелкните в рабочей области, и в ней появится блок кода. Поскольку этот узел используется крайне часто, его вызов привязан к двойному щелчку.



Числа, строки и формулы

Блоки кода поддерживают использование разных типов данных. Пользователи могут быстро задавать числа, строки и формулы, а блок кода предоставит требуемый результат.

На изображении ниже представлен традиционный способ работы с узлами, который отнимает у пользователей больше времени: сначала нужно найти требуемый узел в библиотеке, затем добавить его в рабочую область и, наконец, ввести входные данные. В случае с блоками кода пользователям достаточно дважды щелкнуть в рабочей области и ввести в появившийся узел нужный тип данных с использованием базового синтаксиса.



Узлы *number*, *string* и *formula* можно привести в качестве примеров узлов Dynamo, которые являются устаревшими по сравнению с *блоками кода*.

Синтаксис DesignScript

Синтаксис DesignScript

Возможно, вы уже заметили закономерность в именах узлов Dynamo: каждое из них состоит из слов, разделенных точкой («.») без пробелов. Это связано с тем, что текст в верхней части каждого узла представляет собой фактический синтаксис для создания сценариев, а символ «.» (или *запись через точку*) отделяет элемент от доступных методов, которые можно вызвать. Это позволяет легко переходить от визуальных сценариев к текстовым.

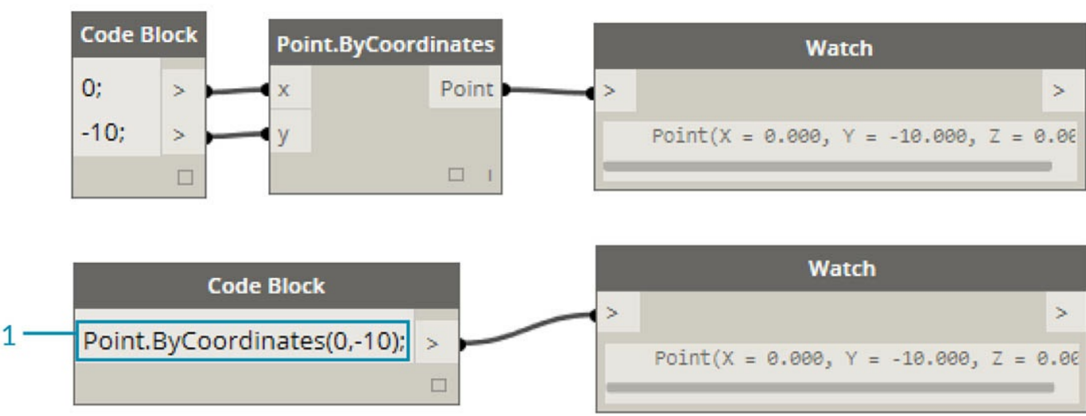
В качестве примера использования записи через точку рассмотрим возможные действия с параметрическим яблоком в Dynamo. Ниже представлены несколько методов, которые можно применить к яблоку, перед тем как съесть (разумеется, на самом деле этих методов в Dynamo не существует, не ищите).

На языке пользователя	Запись через точку	Вывод
Какого цвета яблоко?	Apple.color	Красный
Яблоко зрелое?	Apple.isRipe	true
Сколько весит это яблоко?	Apple.weight	170 г
Откуда взялось это яблоко?	Apple.parent	дерево
Что останется после яблока?	Apple.children	семена
Это яблоко выращено недалеко отсюда?	Apple.distanceFromOrchard	96,5 км

Судя по данным в таблице выше, это очень вкусное яблоко. Я бы его с удовольствием *Apple.eat()*.

Запись через точку в узлах Code Block

Помня про аналогию с яблоком, рассмотрим узел *Point.ByCoordinates* и процесс создания точки с помощью узла Code Block:



Синтаксис *Code Block* (`Point.ByCoordinates(0,10);`) позволяет получить тот же результат, что и узел *Point.ByCoordinates* в Dynamo, но его преимущество заключается в том, что он позволяет создать точку с помощью одного узла. Это проще и эффективнее, чем соединять отдельный узел с портами ввода *X* и *Y*.

1. Используя синтаксис *Point.ByCoordinates* в узле Code Block, нужно указать входные данные в том же порядке, что и в готовом узле *(X,Y)*.

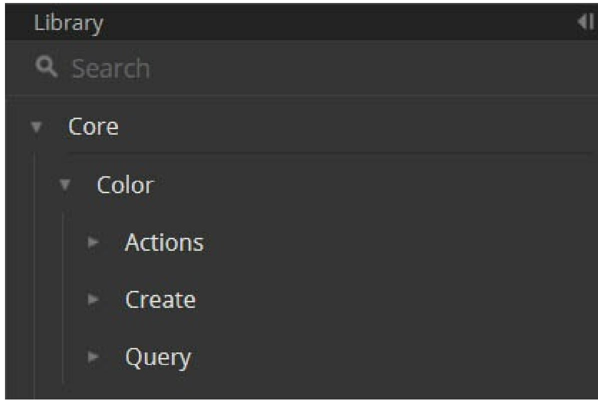
Вызов узлов

С помощью узла Code Block можно вызвать любой стандартный узел библиотеки, если он не является *узлом пользовательского интерфейса*, обладающим специфическими для пользовательского интерфейса функциями. Например, можно вызвать узел *Circle.ByCenterPointRadius*, а вот вызывать узел *Watch 3D* не стоит.

Стандартные узлы, которых в библиотеке большинство, обычно делятся на три типа.

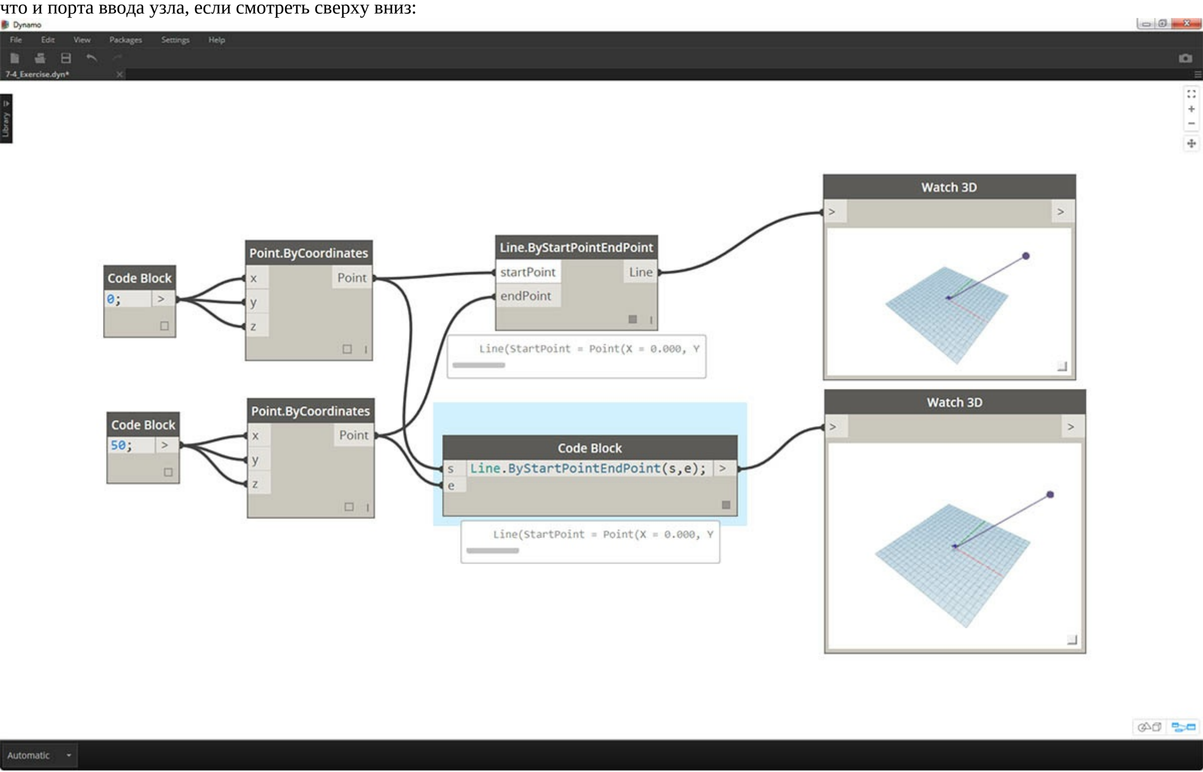
- **Create** — узлы, позволяющие создавать или конструировать что-либо.
- **Action** — узлы для выполнения действий с чем-либо.
- **Query** — узлы для получения свойства существующего объекта.

Как вы видите, вся библиотека упорядочена с учетом этих категорий. Методы (или узлы) этих трех типов обрабатываются иначе при вызове с помощью Code Block.



Create

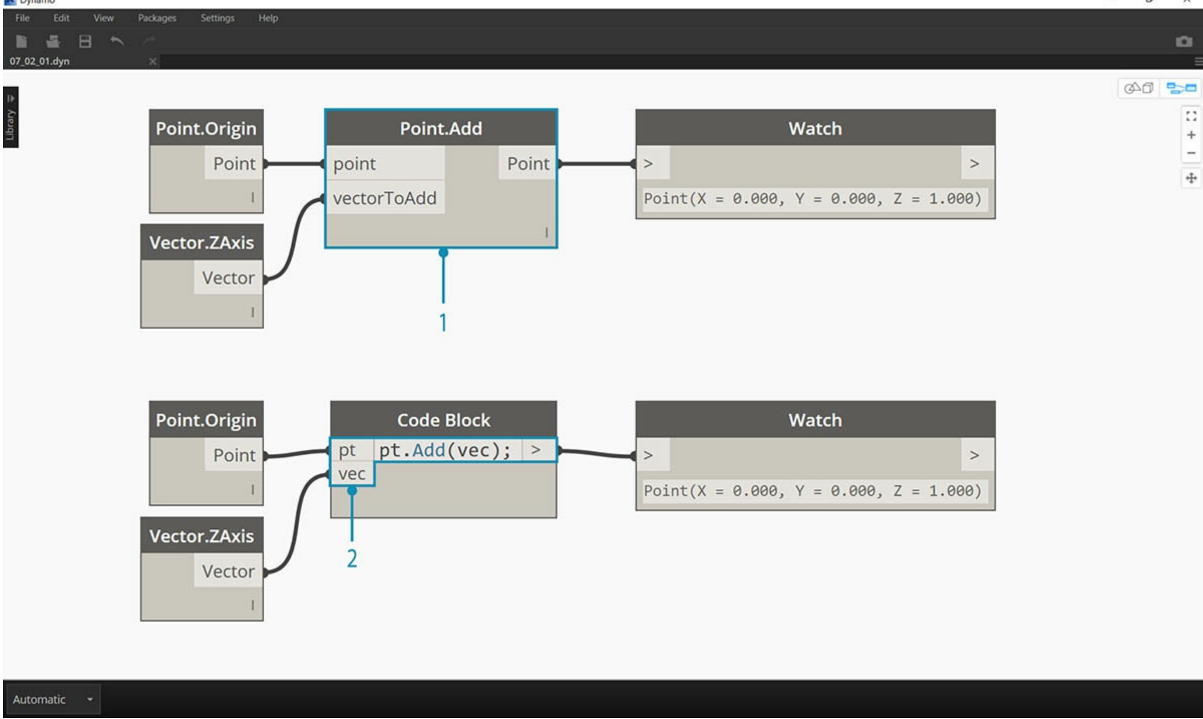
Категория Create позволяет создавать геометрию с нуля. Значения вводятся в Code Block слева направо. Они располагаются в том же порядке, что и порта ввода узла, если смотреть сверху вниз:



При сравнении узла *Line.ByStartPointEndPoint* и соответствующего синтаксиса в узле Code Block мы получаем один и тот же результат.

Action

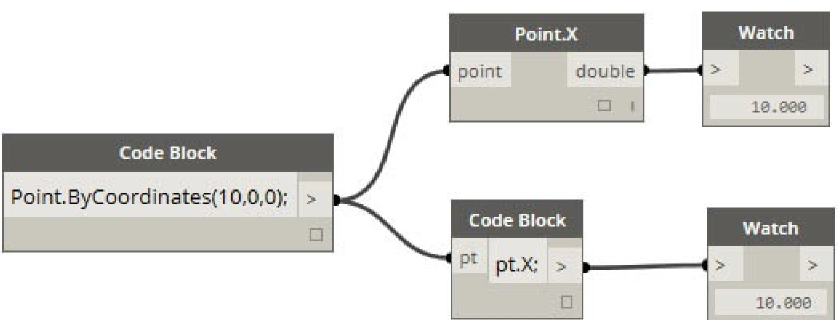
Действие — это операция, выполняемая с объектами определенного типа. Для применения действий к объектам в Dynamo используется *запись через точку*, что является распространенным принципом для многих языков программирования. Если у вас есть объект, введите его название, затем точку, а затем название действия, которое с этим объектом нужно выполнить. Входные данные для метода этого типа помещаются в скобки, как и при использовании методов Create, однако для него не требуется указывать первые входные данные, которые отображаются в соответствующем узле. Вместо этого требуется указать элемент, с которым будет выполняться действие:



1. Поскольку узел *Point.Add* представляет собой узел типа Action, его синтаксис работает несколько иначе.
2. Входные данные включают (1) *точку* и (2) *вектор*, который требуется к ней добавить. В синтаксисе узла *Code Block* точка (объект) обозначена как *pt*. Чтобы добавить вектор (*vec*) к точке (*pt*), нужно ввести *pt.Add(vec)*, то есть «объект, точка, действие». Для операции добавления используются все порты ввода узла *Point.Add*, кроме первого. Первый порт ввода узла *Point.Add* — это сама точка.

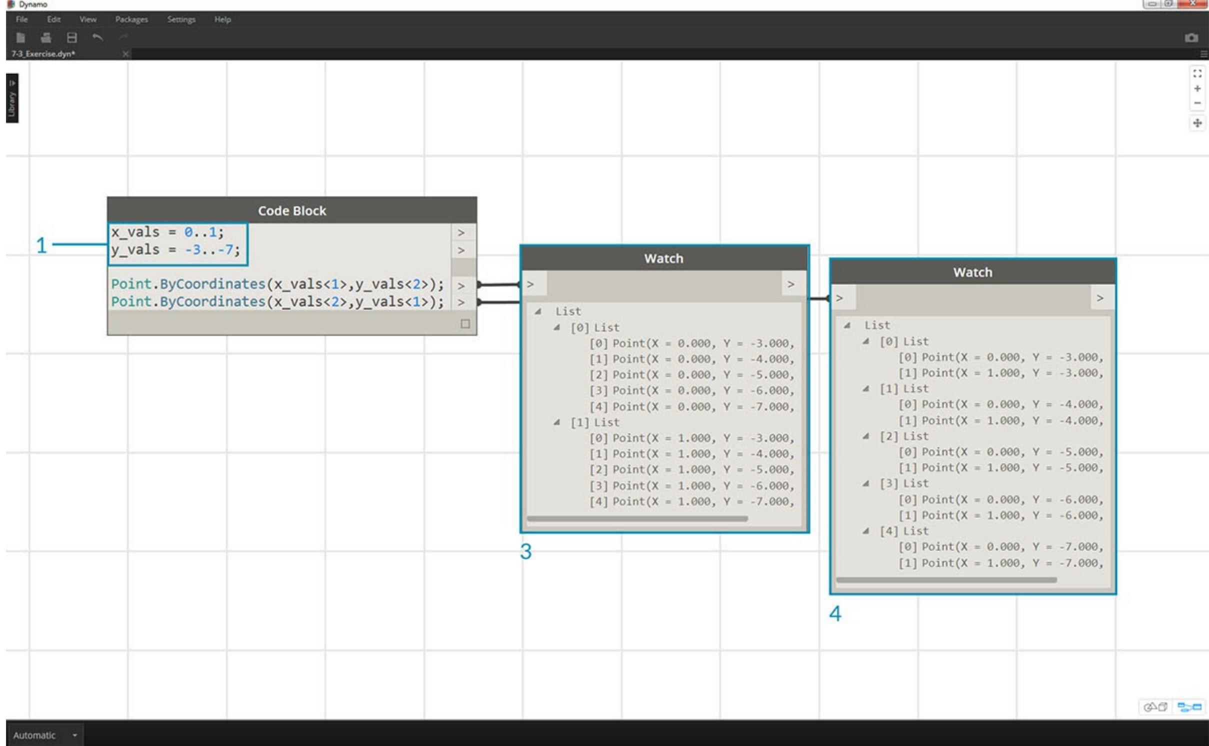
Query

Методы типа Query позволяют получить свойство объекта. Указывать какие-либо входные данные в этом случае не требуется, так как входными данными является сам объект. Скобки также не нужны.



Использование переплетения

Переплетение при использовании узлов отличается от переплетения с помощью Code Block. В первом случае пользователь щелкает узлы правой кнопкой мыши и выбирает параметр переплетения, который требуется применить. При работе с Code Block у пользователя есть гораздо больше возможностей для управления структурой данных. При объединении нескольких одномерных списков в пары с помощью собирательного метода Code Block используются *руководства по репликации*. Цифры в угловых скобках «<>» определяют уровень иерархии для итогового вложенного списка: <1>, <2>, <3> и т. д.

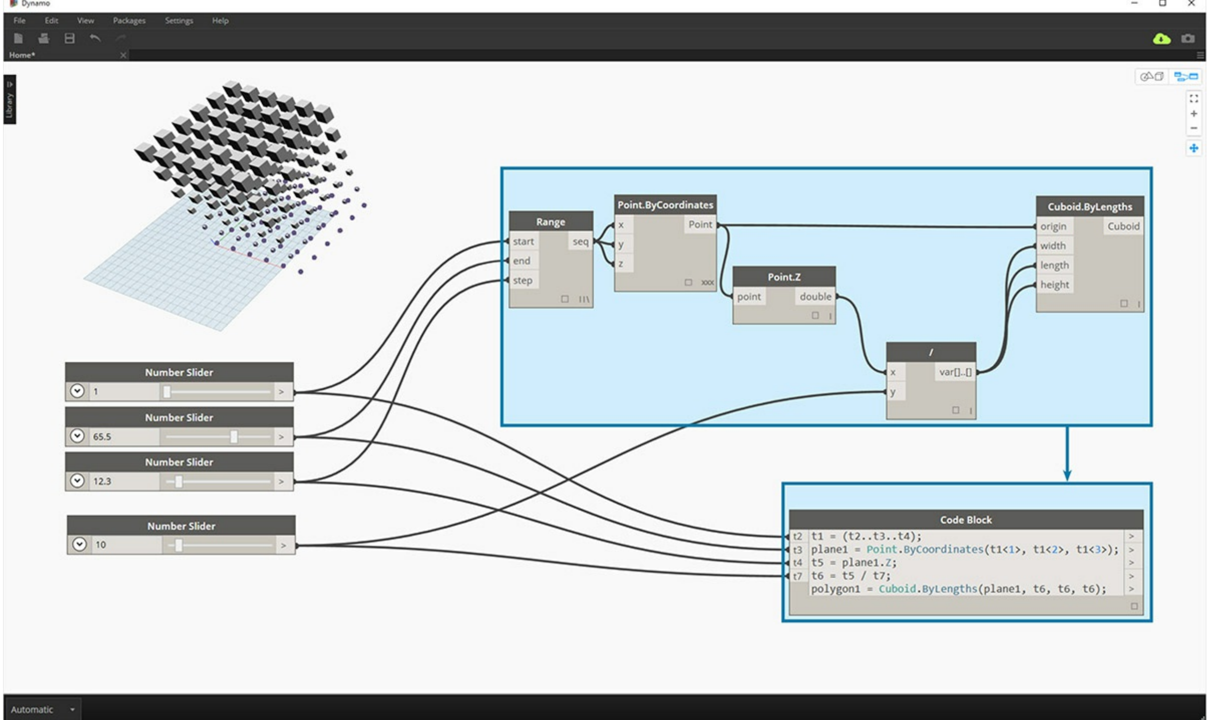


1. В этом примере мы используем собирательный метод для определения двух диапазонов (подробнее о собирательном методе можно узнать в следующем разделе этой главы). По сути, значение `0..1;` эквивалентно `{0,1}`, а значение `-3..-7` эквивалентно `{-3,-4,-5,-6,-7}`. В результате мы получаем список из двух значений X и пяти значений Y. Если работать с этими несогласованными списками без руководств по репликации, то будет получен список, содержащий две точки, что соответствует длине кратчайшего списка. Использование руководств по репликации позволяет найти все возможные сочетания двух и пяти значений координат (а точнее, их **декартово произведение**).
2. Синтаксис `Point.ByCoordinates(x_vals<1>,y_vals<2>);` позволяет получить **два** списка с **пятью** элементами в каждом.
3. Синтаксис `Point.ByCoordinates(x_vals<2>,y_vals<1>);` позволяет получить **пять** списков с **двумя** элементами в каждом.

Такой способ записи позволяет указать, какой список будет основным: два списка из пяти элементов или пять списков из двух. В этом примере результат будет представлять собой список строк точек или список столбцов точек в сетке в зависимости от изменения порядка руководств по репликации.

Узел для кодировки

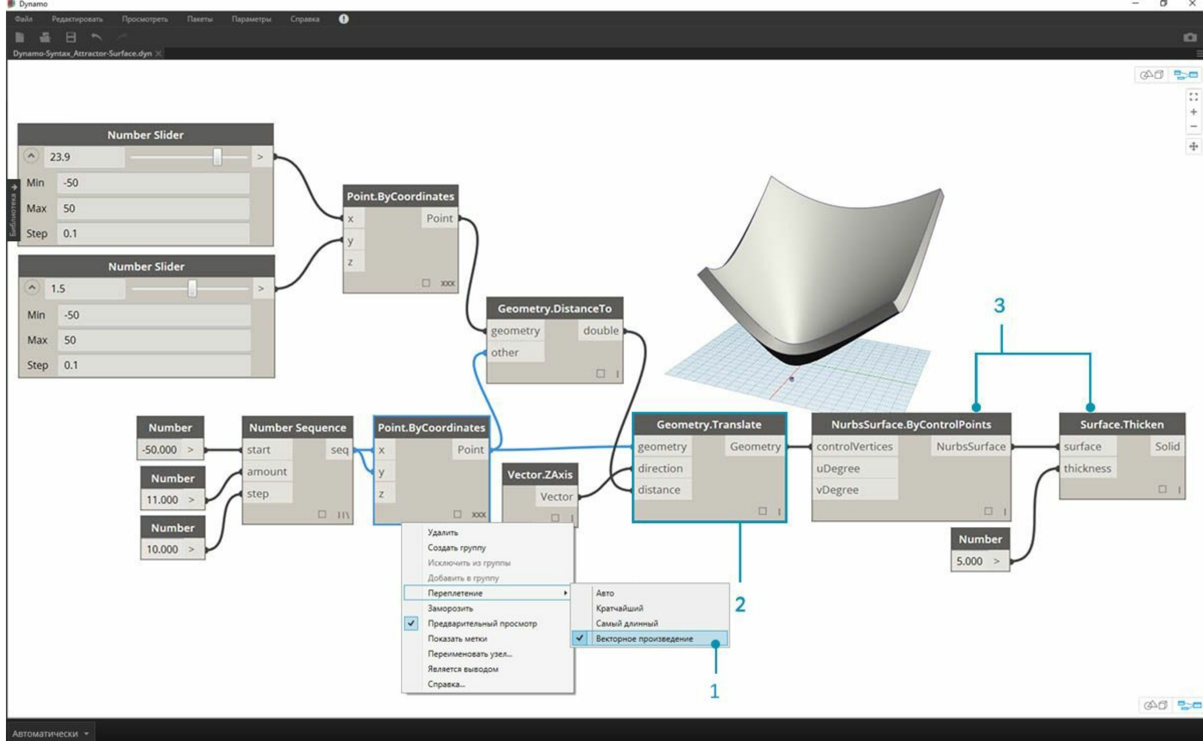
Чтобы овладеть описанными выше методами работы с Code Block, требуется определенное время. Функция «Узел для кодировки» Dynamo значительно упрощает этот процесс. Чтобы использовать эту функцию, выберите массив узлов в графике Dynamo, щелкните правой кнопкой мыши в рабочей области и выберите «Узел для кодировки». Программа Dynamo объединяет эти узлы в единый узел Code Block, содержащий все входные и выходные данные. Это не только отличный инструмент для изучения принципов работы узлов Code Block, но также он позволяет создавать с более эффективные параметрические графики Dynamo. Рекомендуем выполнить упражнение ниже, так как в нем используется функция «Узел для кодировки».



Упражнение

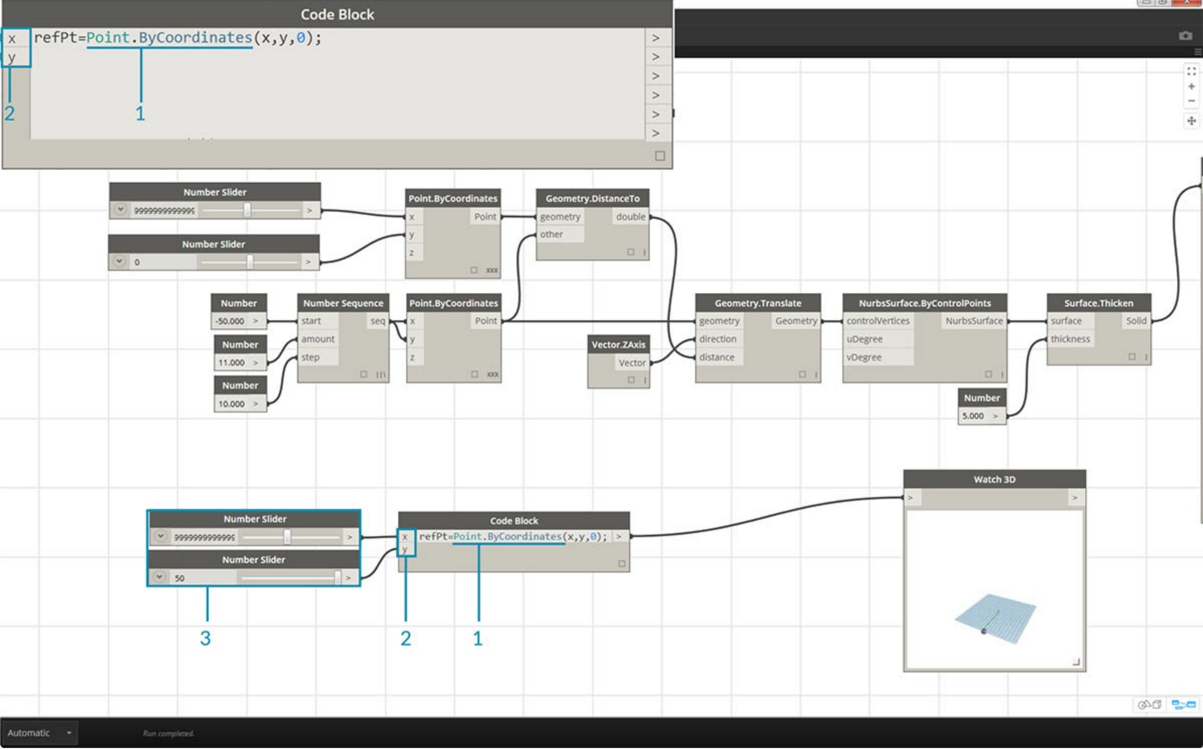
Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. [Dynamo-Syntax_Attractor-Surface.dyn]

Для демонстрации возможностей узла Code Block преобразуйте существующее определение поля притяжения в форму Code Block. Использование существующего определения позволяет продемонстрировать связь Code Block с визуальным программированием, а также помогает в изучении синтаксиса DesignScript.

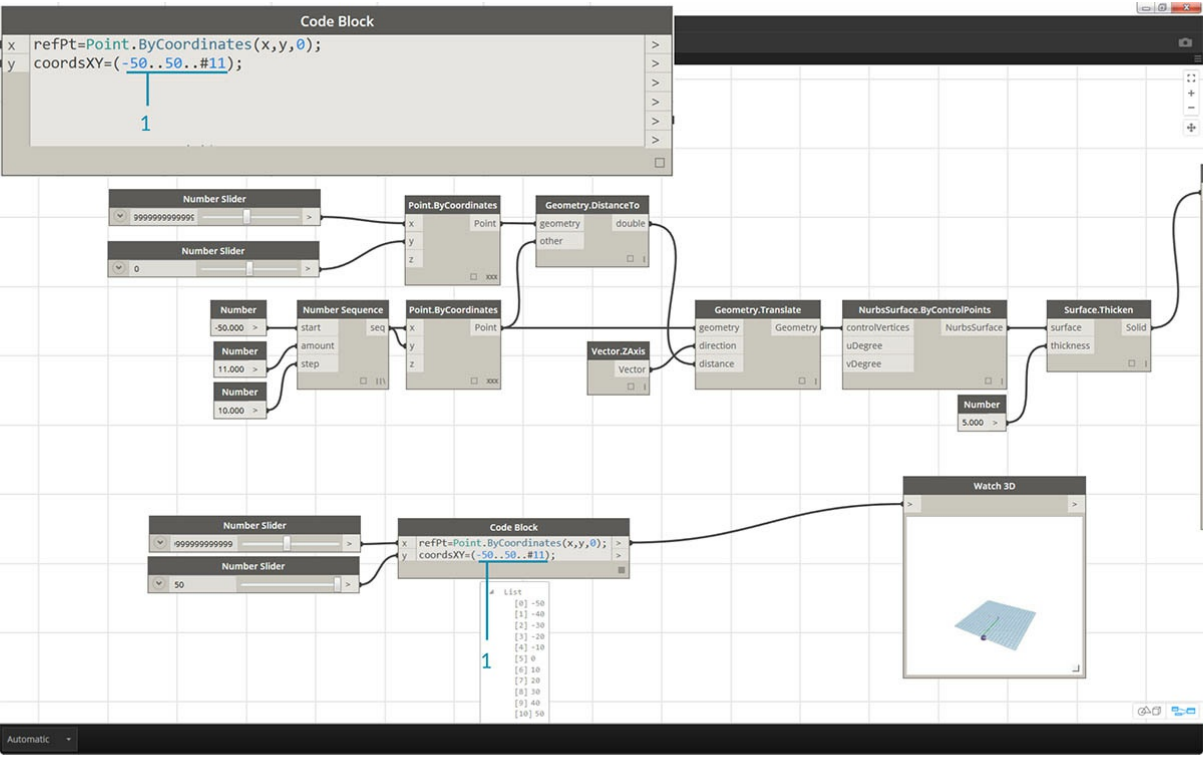


Для начала повторно создайте определение, показанное на изображении выше (или просто откройте файл примера).

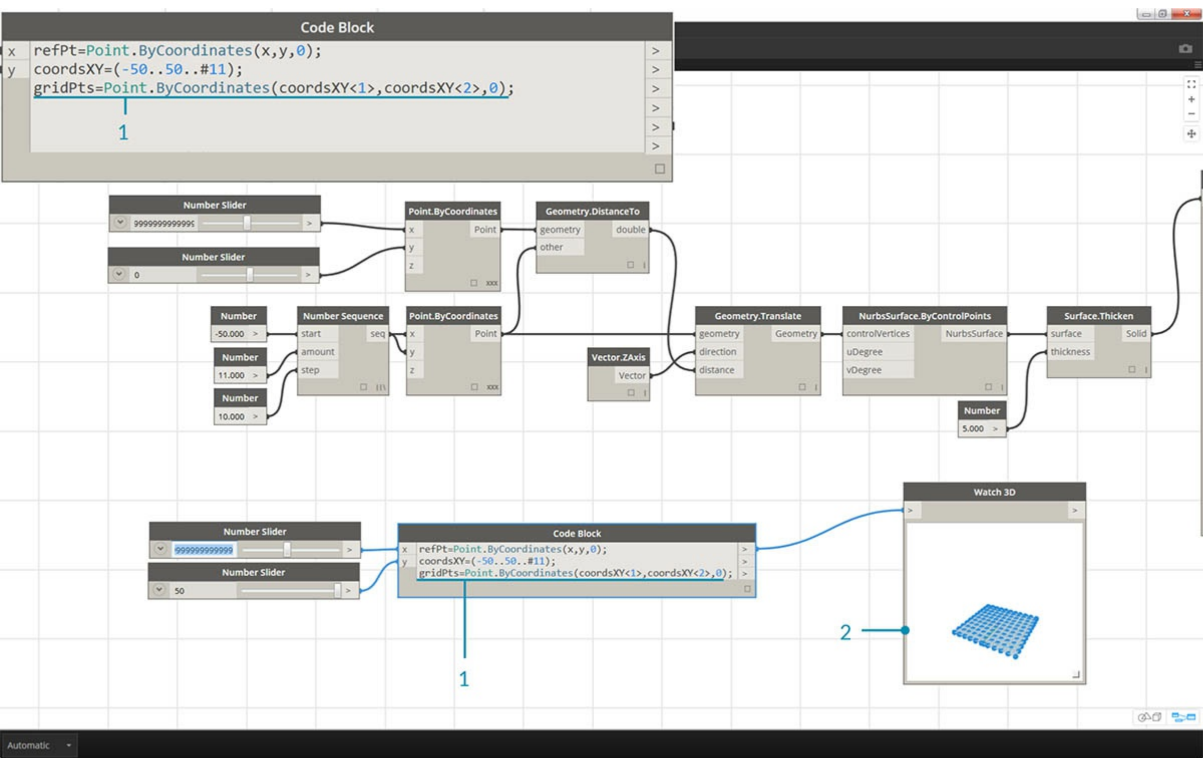
1. Обратите внимание, что для параметра переплетения узла *Point.ByCoordinates* задано значение *Декартово произведение*.
2. Каждая точка сетки перемещена вверх по оси Z в соответствии с расстоянием от опорной точки.
3. Поверхность создана повторно и утолщена, что создает прогиб в геометрии относительно расстояния до опорной точки.



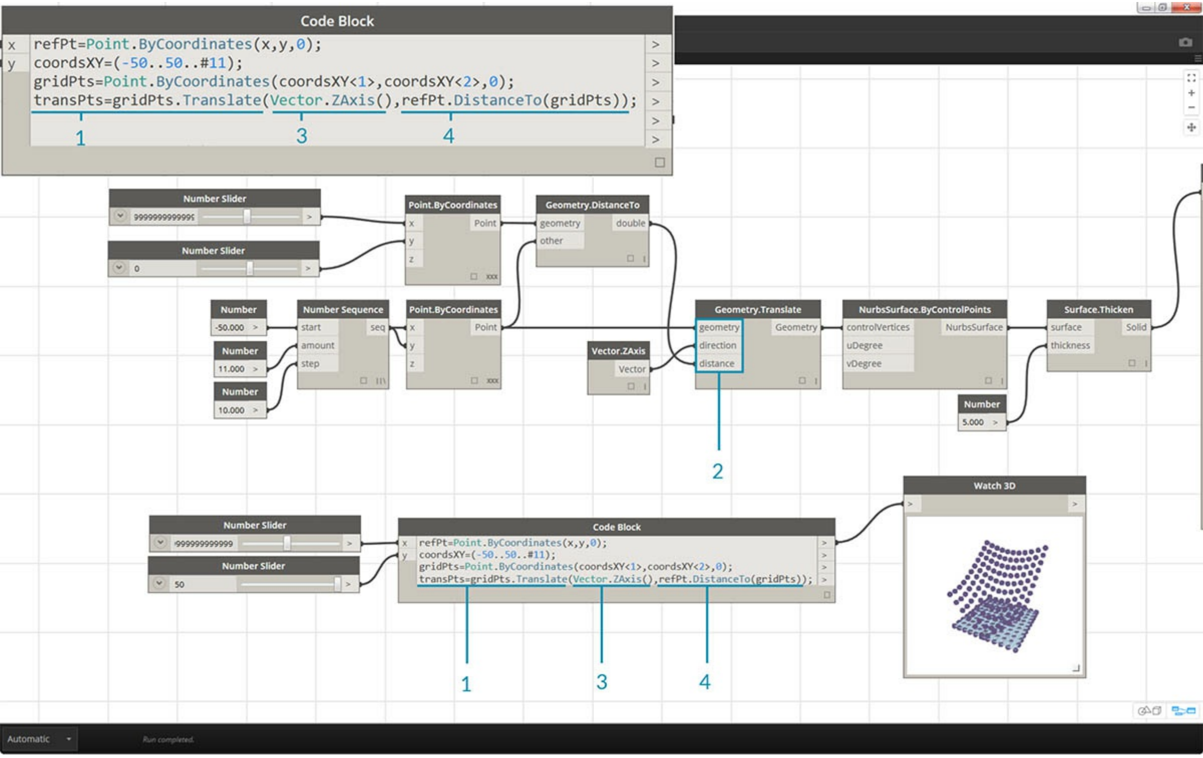
1. Для начала определите опорную точку: `Point.ByCoordinates(x,y,0);`. Используйте синтаксис *Point.ByCoordinates*, указанный в верхней части узла опорной точки.
2. Переменные *x* и *y* вставляются в Code Block, чтобы их можно было динамически обновлять с помощью регуляторов.
3. Присоедините *регуляторы* к портам ввода узла *Code Block* и задайте для них значения в диапазоне от *-50* до *50*. Это позволит работать по всей сетке Dynamo по умолчанию.



1. Во второй строке окне *Code Block* определите собирательный метод, заменяющий узел с числовой последовательностью: `coordsXY = (-50..50..#11);`. Мы рассмотрим эту тему подробнее в следующем разделе. Обратите внимание, что этот собирательный метод эквивалентен узлу визуального программирования *Number Sequence*.



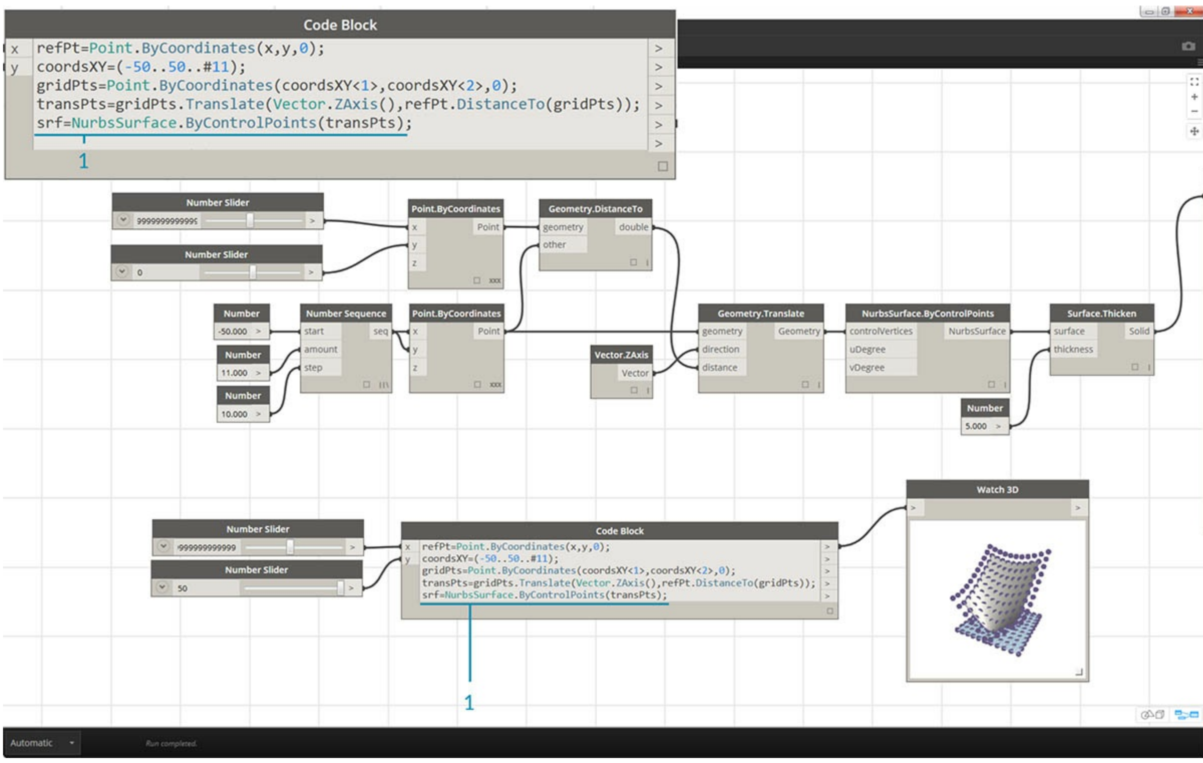
1. Теперь нужно создать сетку из точек последовательности *coordsXY*. Для этого необходимо использовать синтаксис *Point.ByCoordinates*. Кроме того, требуется запустить *декартово произведение* списка так же, как вы делали это при визуальном программировании. Чтобы это сделать, введите строку: `gridPts = Point.ByCoordinates(coordsXY<1>,coordsXY<2>,0);`. Угловые скобки обозначают ссылку на декартово произведение.
2. Обратите внимание на узел *Watch3D*, который отображается сетку точек на сетке Dynamo.



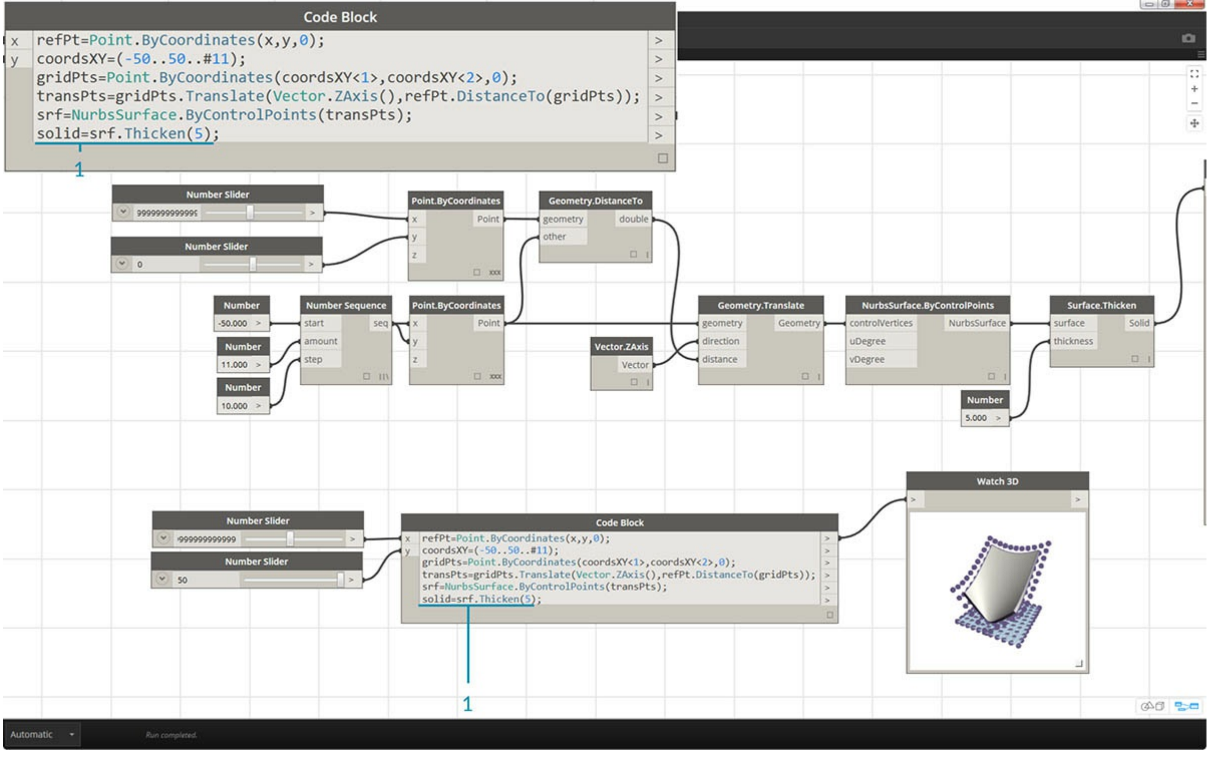
1. Теперь самое сложное: нужно переместить сетку точек вверх в соответствии с расстоянием от опорной точки. Для начала присвоим этому новому набору точек имя *transPts*. Так как преобразование выполняется для уже существующего элемента, вместо узла `Geometry.Translate...` используйте `gridPts.Translate`.
2. Считывая данные из этого узла в рабочей области, мы видим, что он содержит три порта ввода. Преобразуемая геометрия уже определена, так как действие выполняется с текущим элементом (с использованием *gridPts.Translate*). Названия двух оставшихся портов следует поместить в скобки функции: *direction* и *distance*.
3. Определить значение direction несложно, так как для перемещения по вертикали используется `Vector.ZAxis()`.
4. Нам все еще нужно рассчитать расстояние между опорной точкой и каждой точкой сетки. Выполните это действие с опорной точкой

аналогичным образом: `refPt.DistanceTo(gridPts)`.

5. Последняя строка кода создает преобразованные точки: `transPts = gridPts.Translate(Vector.ZAxis(),refPt.DistanceTo(gridPts));`.



1. Теперь у вас есть сетка точек, структура данных которой позволяет создать поверхность NURBS. Создайте поверхность с помощью синтаксиса `srf = NurbsSurface.ByControlPoints(transPts);`.

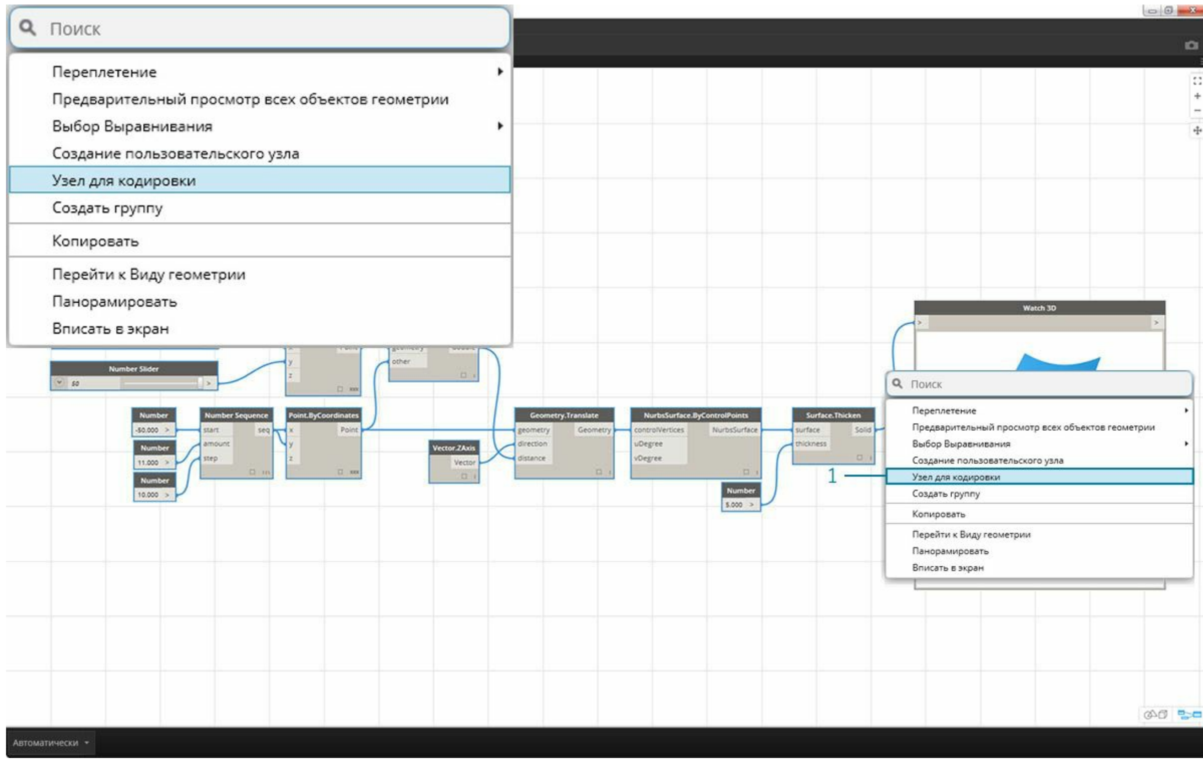


1. Чтобы придать поверхности глубину, создайте тело с помощью синтаксиса `solid = srf.Thicken(5)`. Данный код увеличивает толщину поверхности на пять единиц, однако толщину всегда можно сделать переменной (например, с именем *thickness*), а затем изменять ее значение с помощью регулятора.

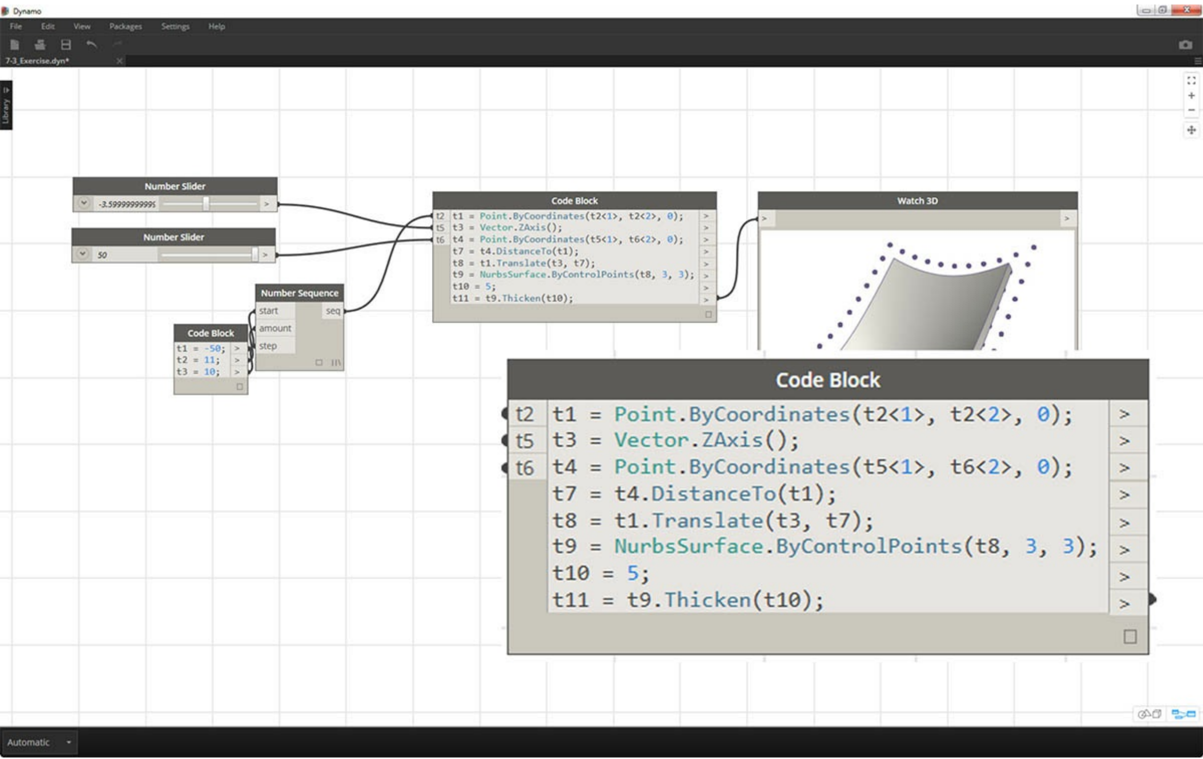
Упрощение графика с помощью функции «Узел для кодировки»

Функция «Узел для кодировки» позволяет автоматизировать все действия, которые вы выполнили в предыдущем упражнении, и применять их

одним нажатием кнопки. Этот мощный инструмент не только позволяет создавать пользовательские определения и узлы Code Block для многократного использования, но и помогает в изучении процесса создания сценариев в Dynamo.



1. Для начала откройте существующий визуальный сценарий из первого шага упражнения. Выберите все узлы, щелкните правой кнопкой мыши в рабочей области и выберите *Узел для кодировки*. Проще простого.

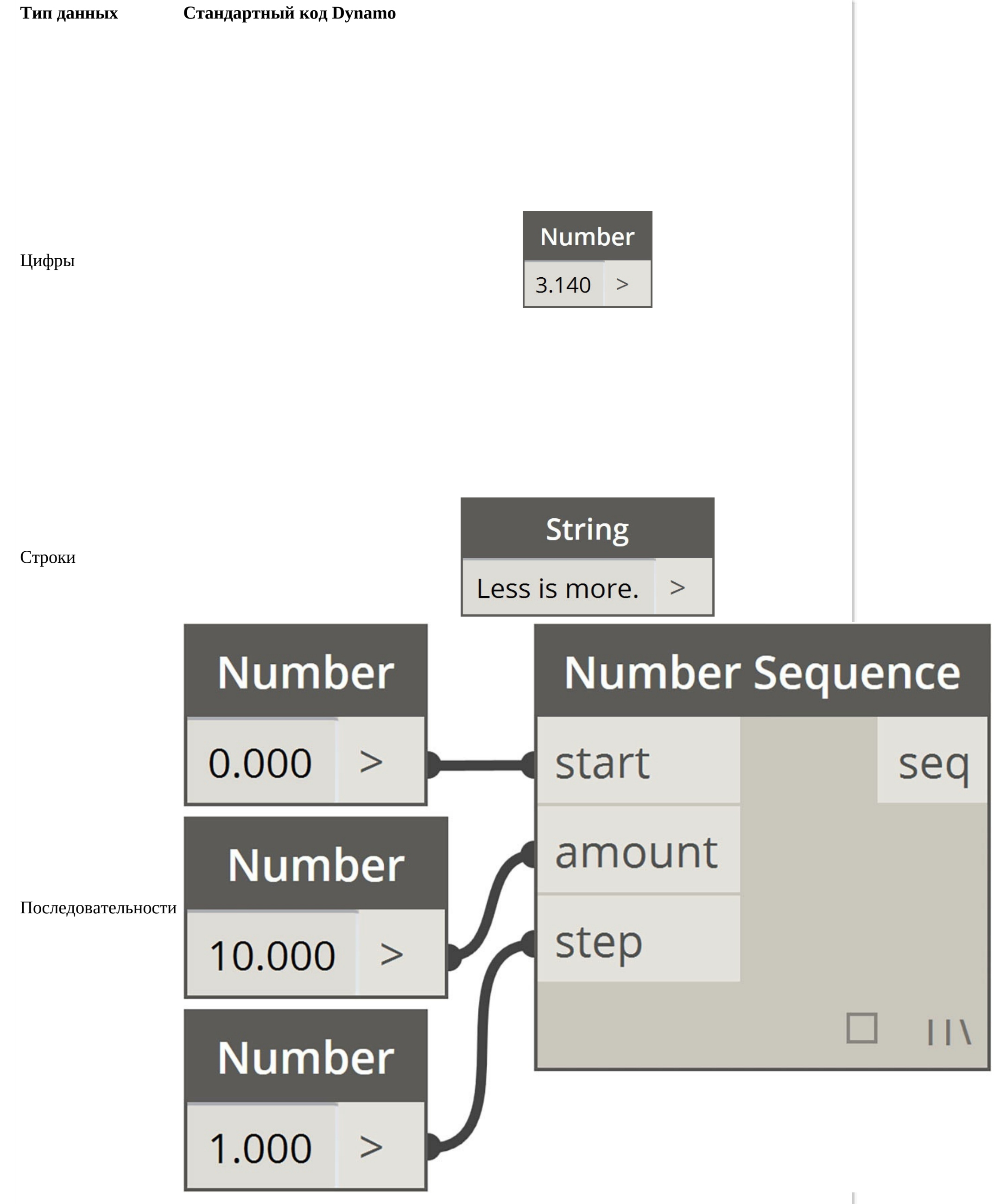


Приложение Dynamo позволяет автоматизировать текстовую версию визуального графика, включая переплетение и прочие операции. Поэкспериментируйте с использованием этой функции при работе с другими визуальными сценариями и откройте все возможности Code Block.

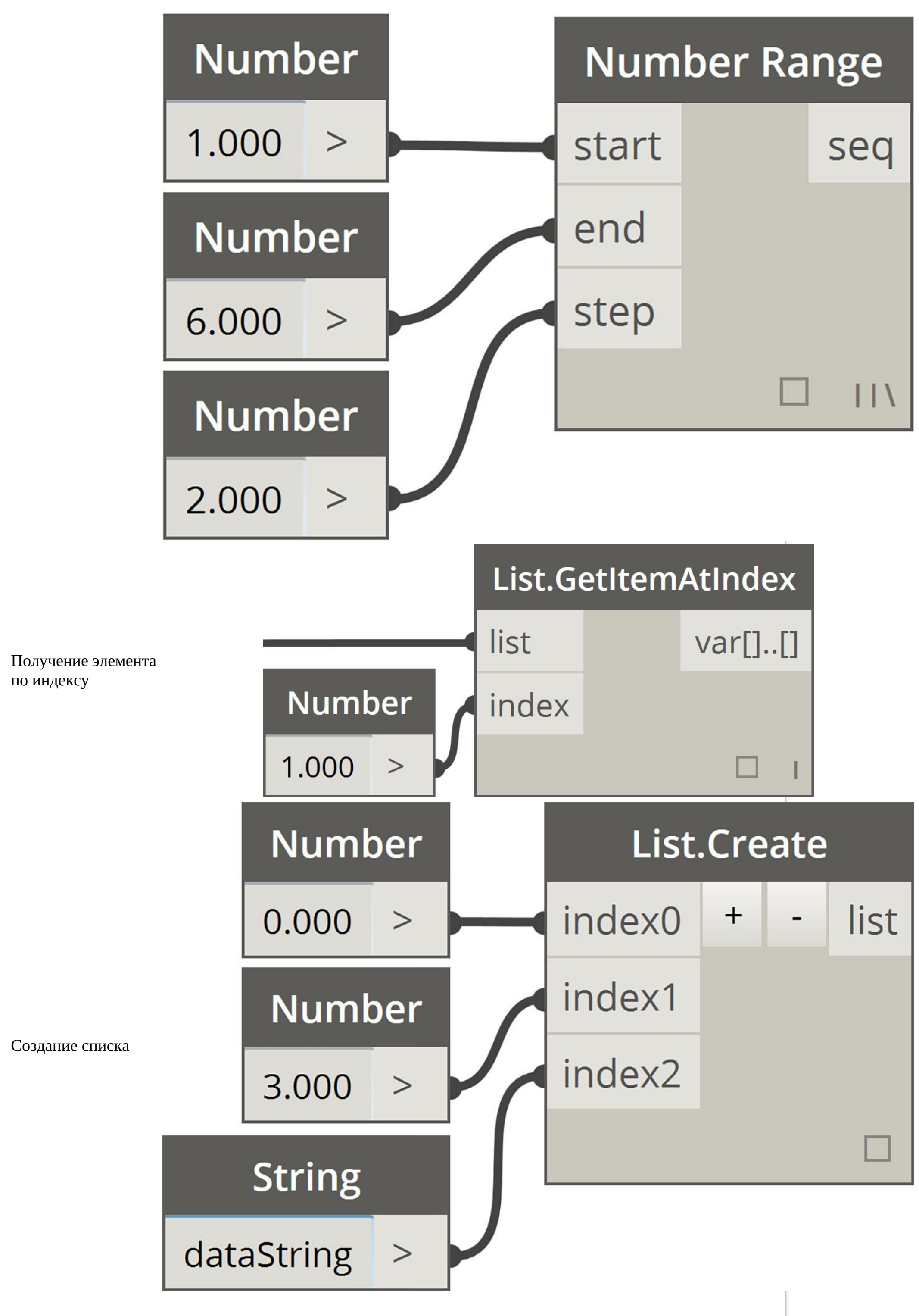
Сокращение

Сокращение

Существует несколько основных способов сокращенной записи команд в блоке кода, что *существенно* упрощает управление данными. Далее мы подробно рассмотрим, как использовать подобные сокращенные записи для создания и запроса данных.



Тип данных	**Стандартный код Dynamo**
Цифры	
Строки	
Последовательности	

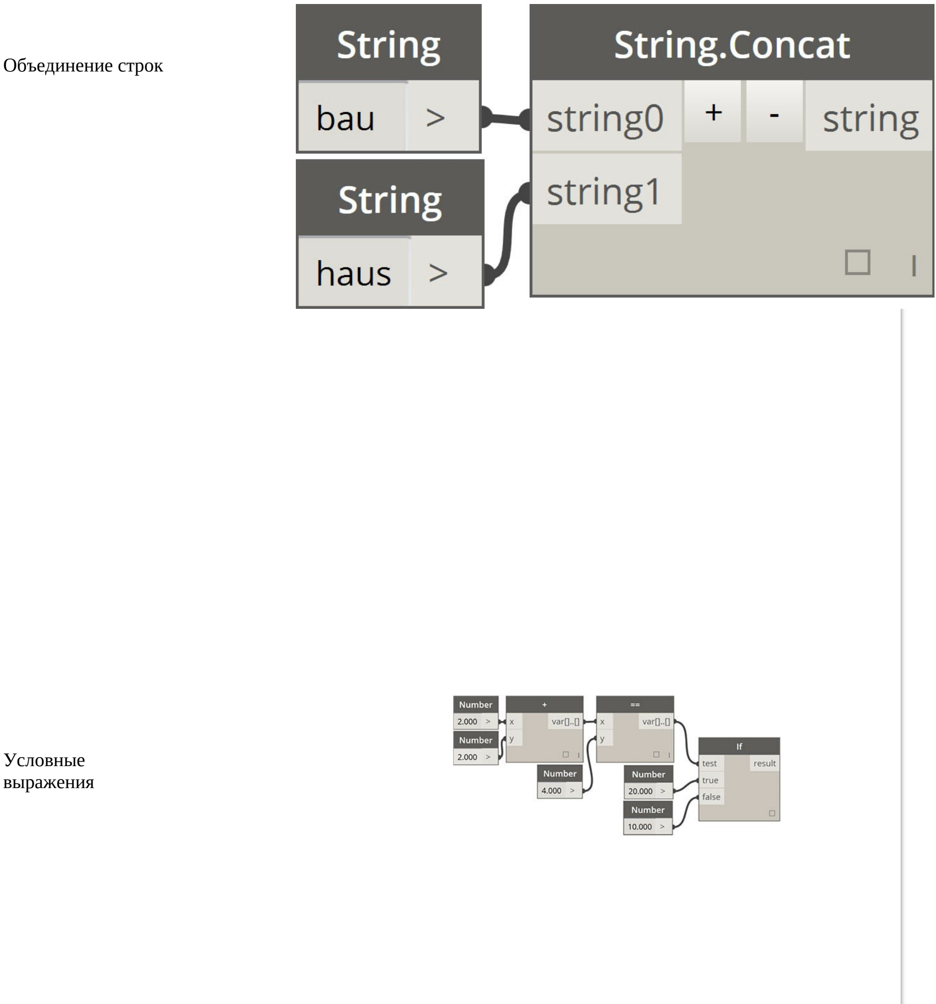


Получение элемента по индексу

Создание списка

Объединение строк

Условные выражения

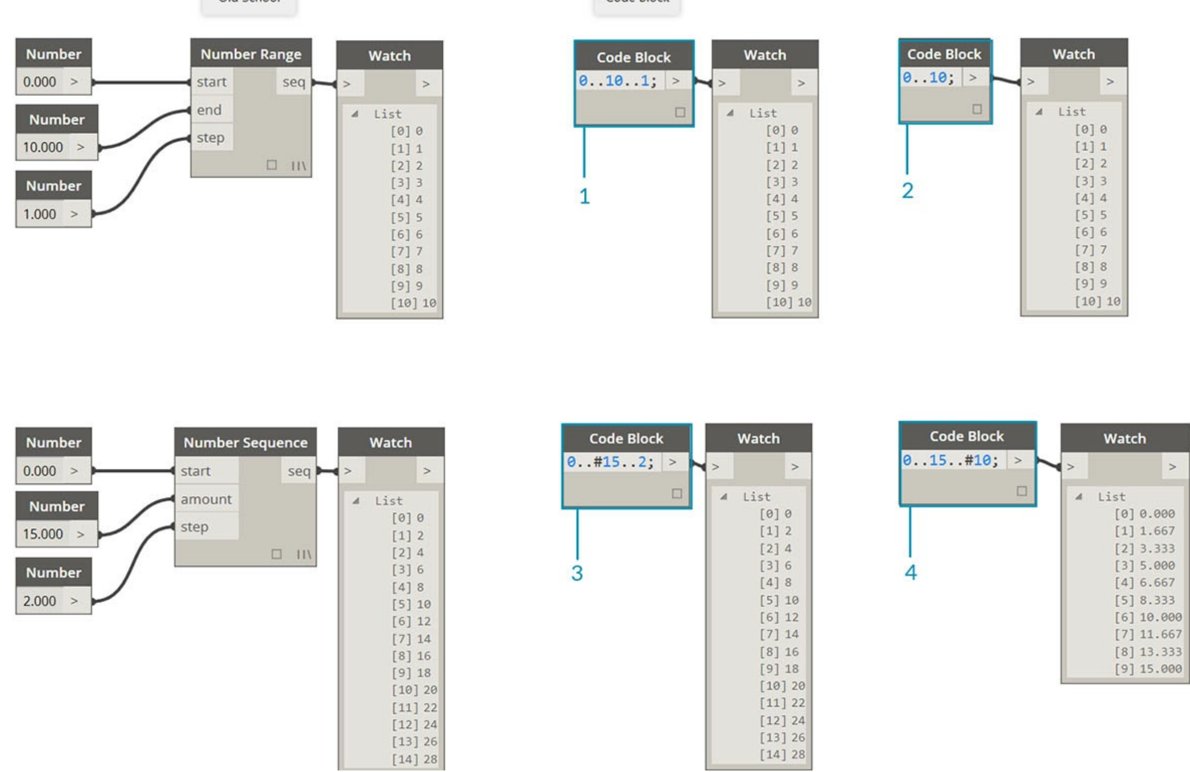


Дополнительный синтаксис

Узлы	Аналог в блоке кода	Примечание
Любой оператор (+, &&, >=, Not и т. д.)	+, &&, >=, !, и т. д.	Обратите внимание, что Not заменяется на «!», однако узел называется Not для отличия от Factorial.
Boolean True	true;	Все в нижнем регистре
Boolean False	false;	Все в нижнем регистре

Диапазоны

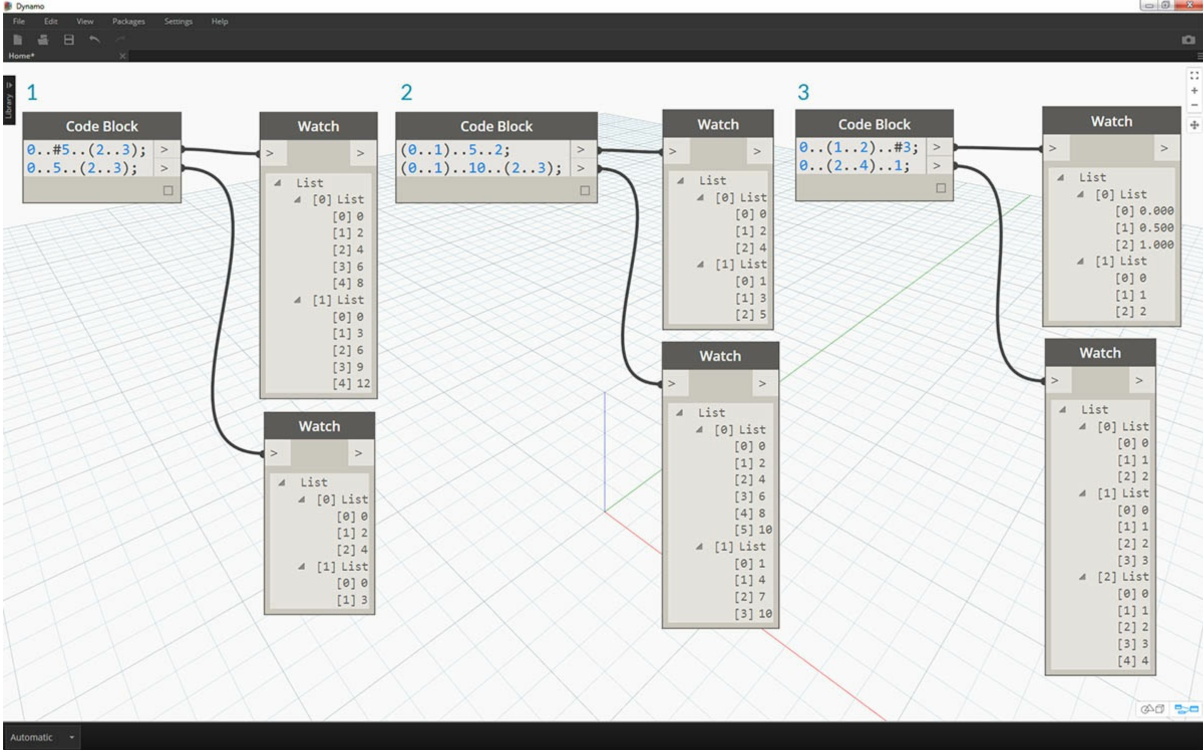
Метод определения диапазонов и последовательностей можно заменить обычной сокращенной записью. Приведенное ниже изображение можно рассматривать как руководство по использованию синтаксиса «..» для определения списка числовых данных с помощью блока кода. Освоение этой системы записи позволит более эффективно формировать числовые данные:



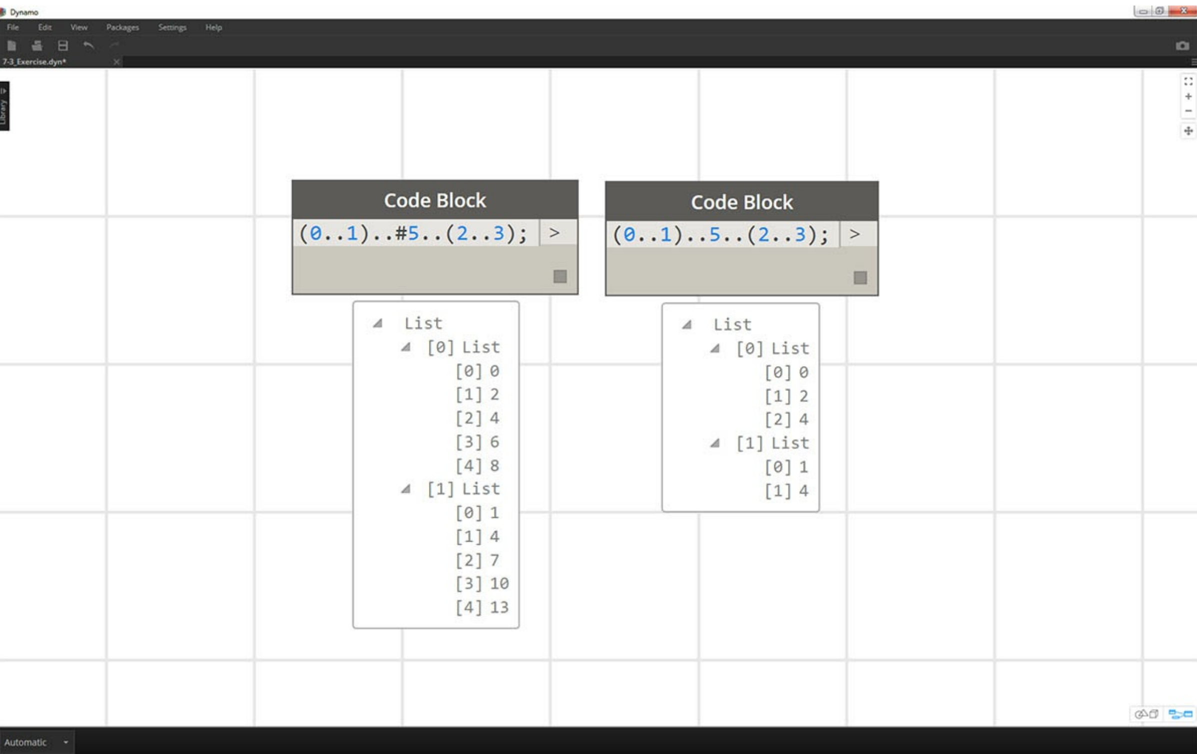
1. В этом примере числовой диапазон заменяется элементарной синтаксической конструкцией в блоке кода, определяющей `начало..конец..размер шага;`. В числовом выражении получаем: `0..10..1;`
2. Обратите внимание на то, что синтаксис `0..10..1;` эквивалентен синтаксису `0..10;`. При сокращенной записи размер шага 1 является значением по умолчанию. Таким образом, значение `0..10;` соответствует последовательности от 0 до 10 с размером шага 1.
3. Похожий пример наблюдаем с *последовательностью чисел*. Единственное исключение — знак #, указывающий на список из 15 значений, а не на список, увеличивающийся до 15. В данном случае задаются значения `начало..кол-во шагов..размер шага:`. Окончательный синтаксис последовательности — `0..#15..2`
4. Разместим символ # из предыдущего шага в компоненте *размер шага* синтаксической конструкции. Теперь у нас есть *диапазон чисел*, идущий от *начала* к *концу*, а *размер шага* равномерно распределяет несколько значений между этими двумя точками: `начало..конец..кол-во шагов`.

Дополнительные диапазоны

Создание дополнительных диапазонов упрощает работу со списками списков. В примерах ниже переменная изолируется от обозначений основного диапазона и создается другой диапазон этого списка.



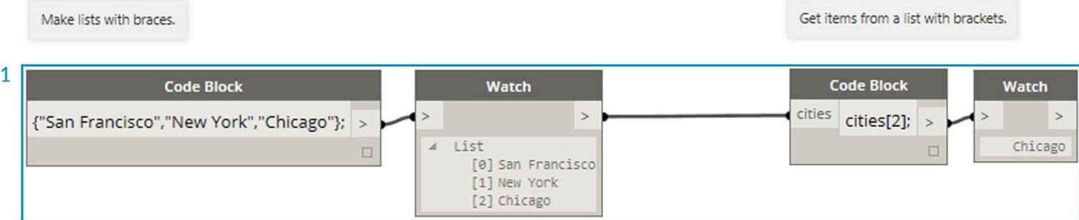
1. При создании вложенных диапазонов сравните запись с символом # и без него. Действует тот же принцип, что и в случае с основными диапазонами, но в усложненном варианте.
2. Вложенный диапазон может находиться в любом месте основного диапазона. Обратите внимание на возможность наличия двух вложенных диапазонов.
3. Используя различные значения *конца* диапазона, можно создавать дополнительные диапазоны с различной длиной.

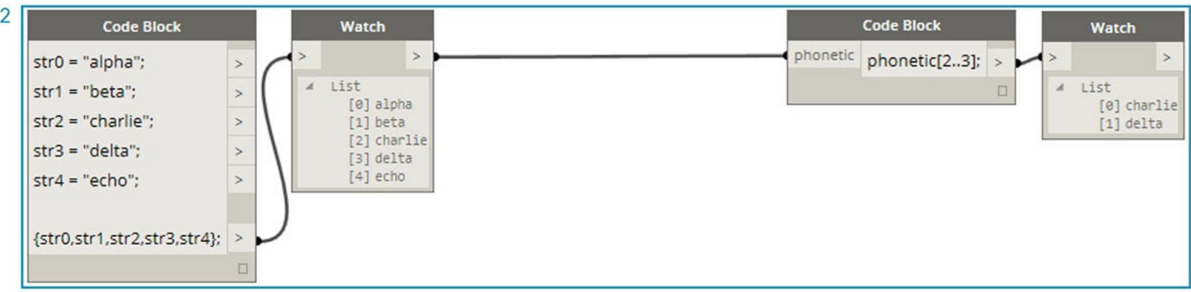


В качестве упражнения сравните две сокращенные записи на изображении выше и попытайтесь проанализировать, как *вложенные диапазоны* и символ # влияют на конечный результат.

Создание списков и получение элементов из списка

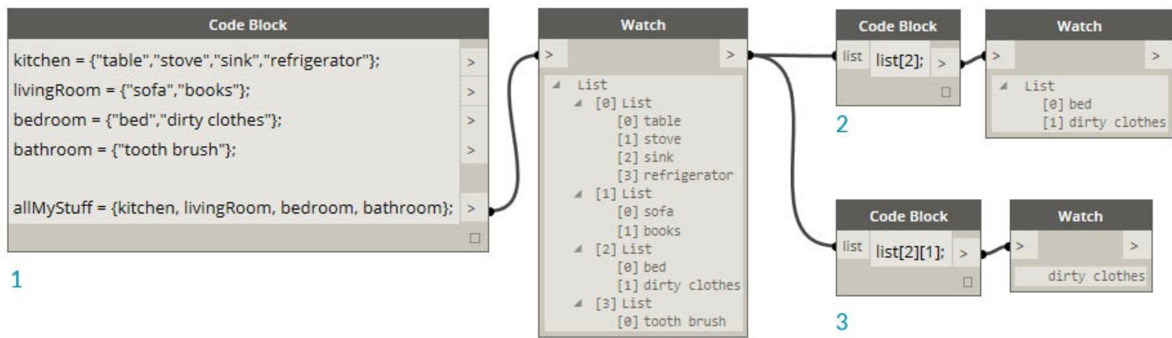
С помощью сокращений можно не только создавать списки, но и делать это динамически. Эти списки могут содержать множество типов элементов и поддерживают запросы (помните, что списки — это объекты в объектах). Наконец, в блоке кода для создания списков используются фигурные скобки, а для запроса элементов из списка — квадратные скобки.





1. Быстрое создание списков с помощью строк и запрос содержимого списков с помощью индекса элементов.
2. Создание списков с переменными и запрос содержимого с помощью сокращенной записи диапазонов.

Аналогичным образом можно работать с вложенными списками. Учитывайте порядок расположения списков, а для вызова используйте несколько наборов квадратных скобок:

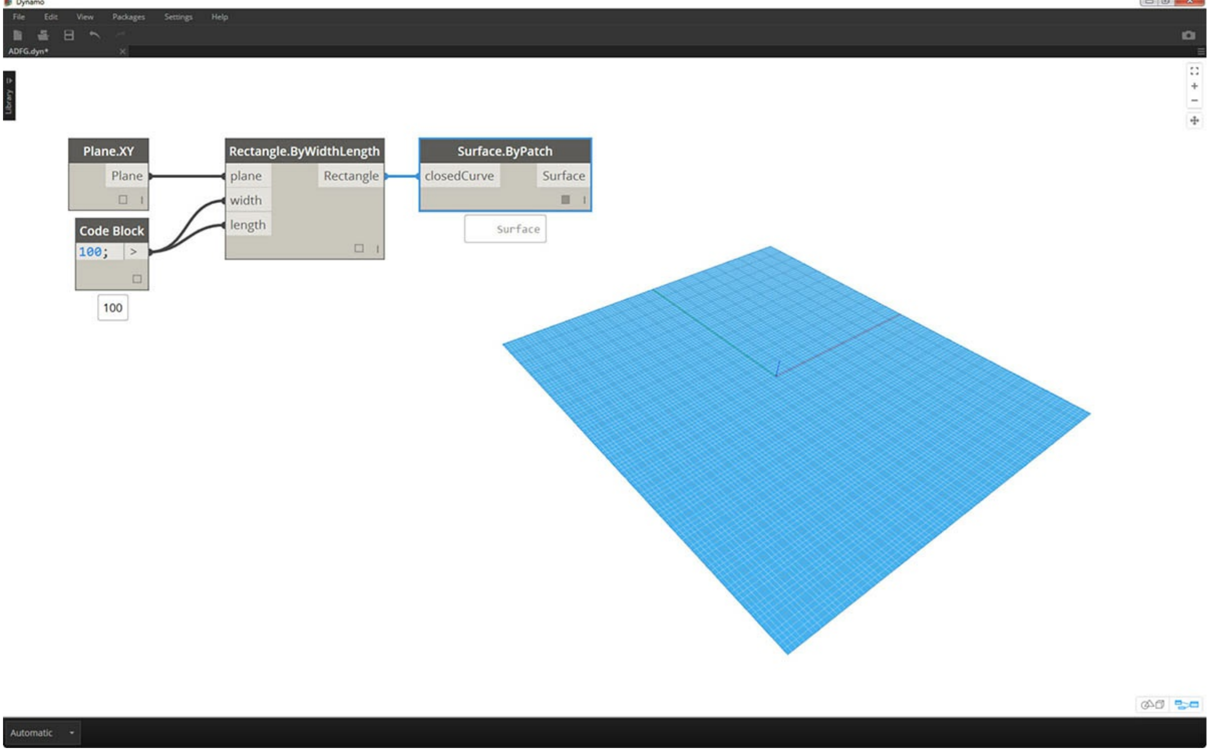


1. Определение списка списков.
2. Запрос содержимого списка с помощью одного набора квадратных скобок.
3. Запрос содержимого списка с помощью двух наборов квадратных скобок.

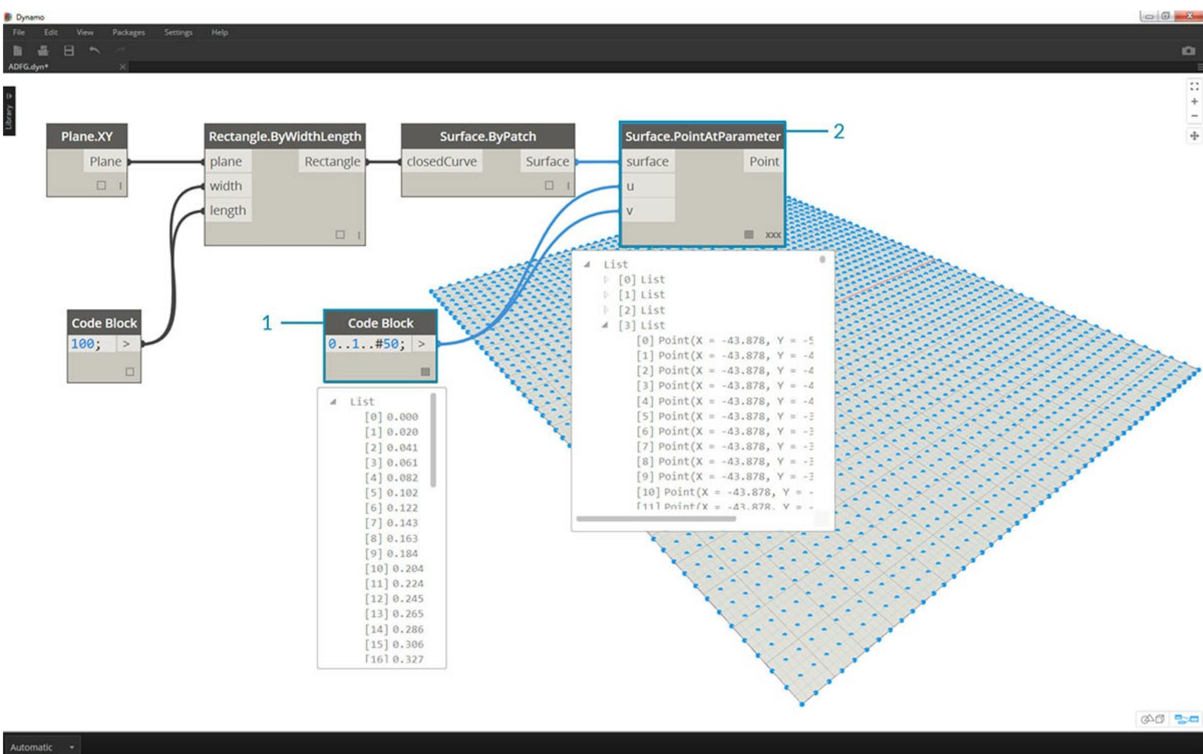
Упражнение

Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. Obsolete-Nodes_Sine-Surface.dyn

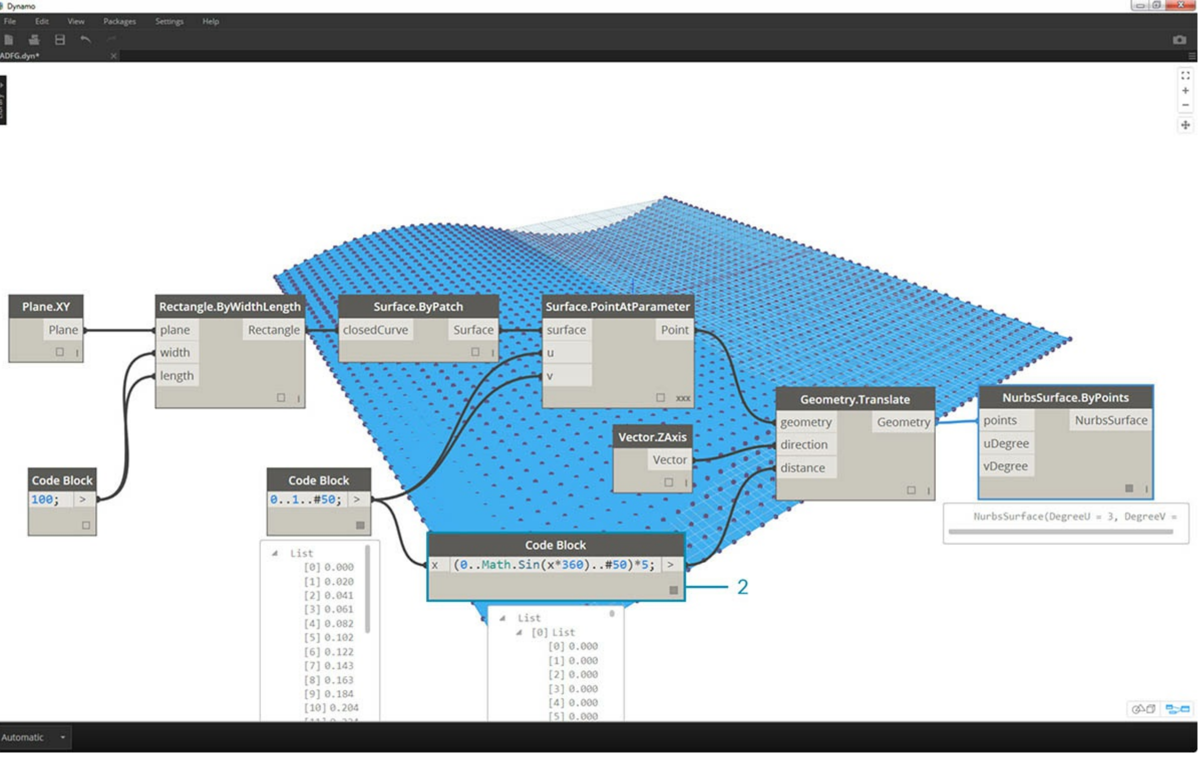
В этом упражнении мы применим навыки сокращенной записи для создания оригинальной яйцевидной поверхности, которая будет определяться диапазонами и формулами. При выполнении упражнения обратите внимание на одновременное использование блока кода и существующих узлов Dynamo. Блок кода используется для обработки больших данных, а узлы Dynamo для наглядности.



Начните с создания поверхности путем соединения узлов, представленных на изображении выше. Вместо использования узла Number для определения ширины и длины поверхности дважды щелкните в рабочей области в блоке кода введите `100;`.

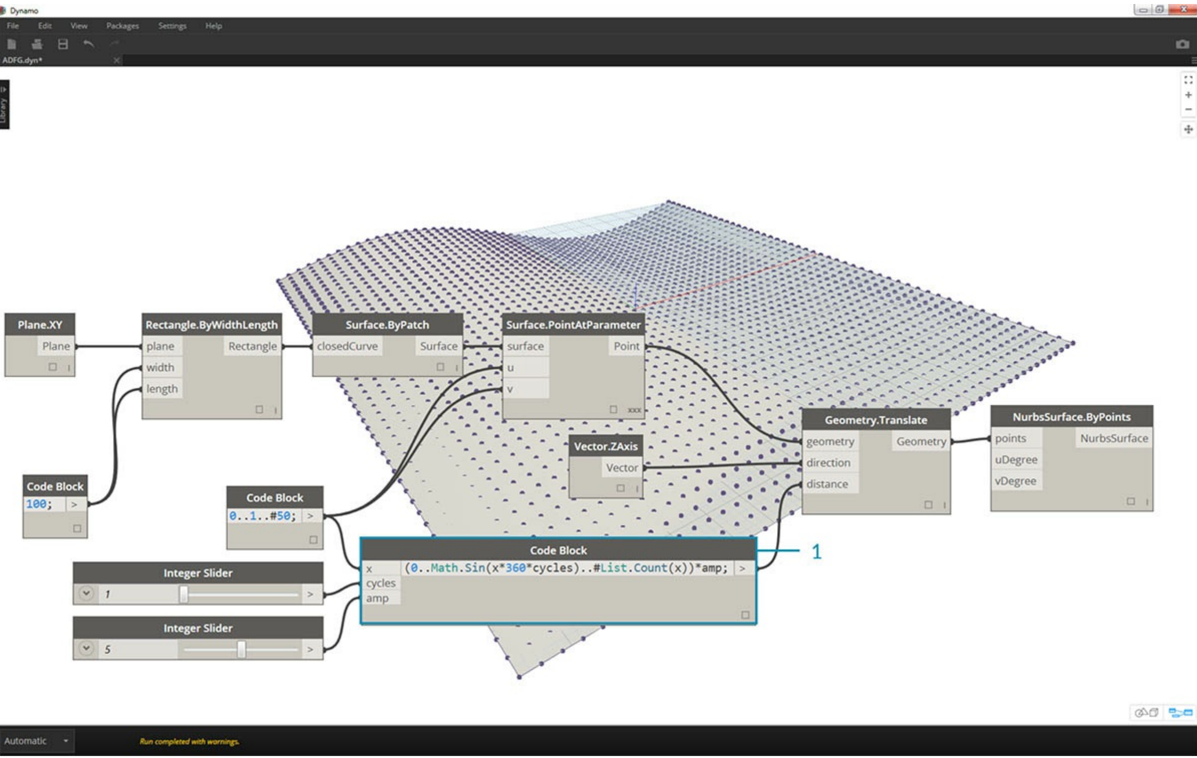


1. Создайте диапазон от 0 до 1 с 50 делениями. Для этого введите `0..1..#50` в блоке кода.
2. Соедините диапазон с узлом *Surface.PointAtParameter*, который извлекает значения *u* и *v* в диапазоне от 0 до 1 по всей поверхности. Не забудьте в качестве режима *Переплетение* выбрать *Декартово произведение*, щелкнув правой кнопкой мыши узел *Surface.PointAtParameter*.

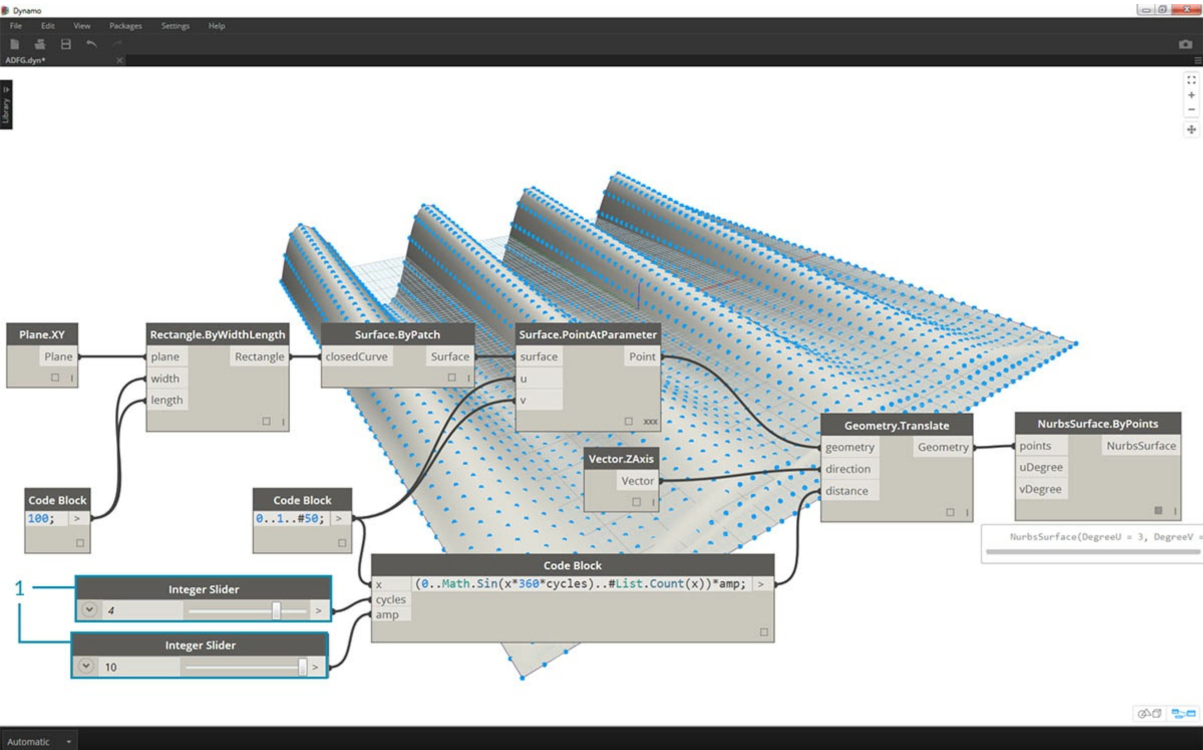


На этом этапе с помощью первой функции переместим сетку точек вверх по оси Z. Эта сетка будет управлять поверхностью на основе базовой функции.

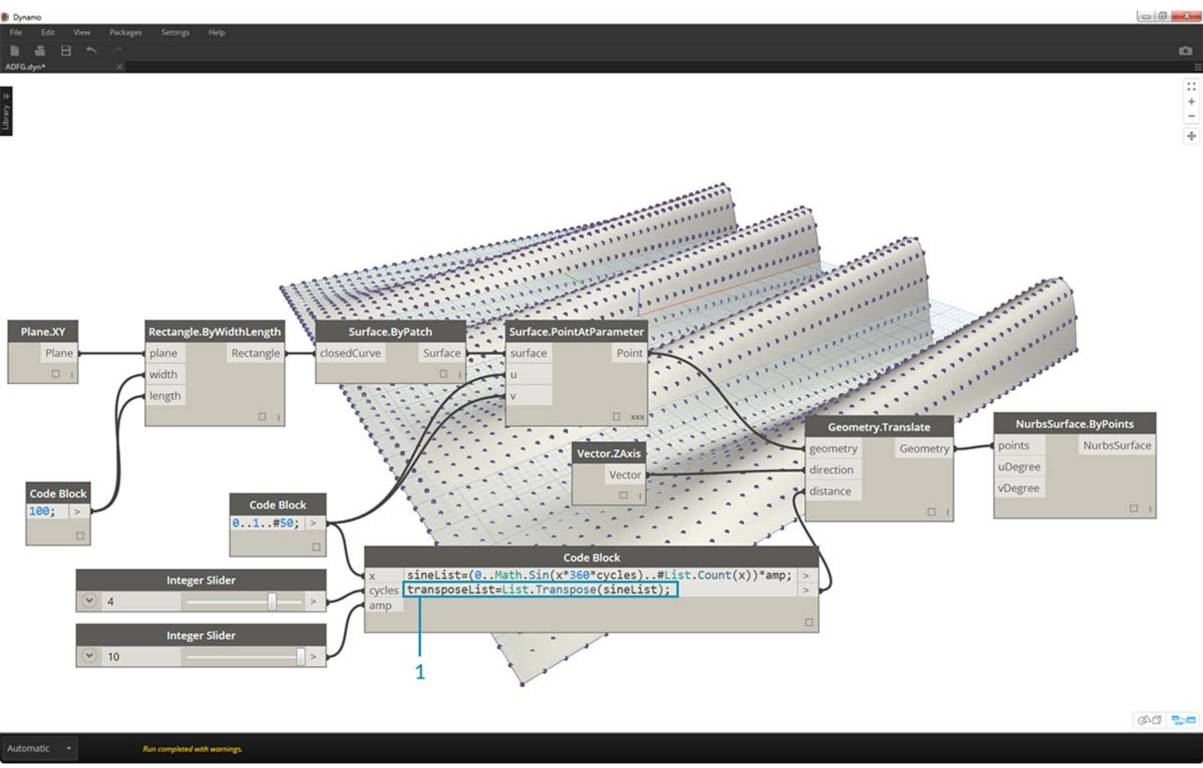
1. Добавьте визуальные узлы в рабочую область, как показано на изображении выше.
2. Вместо узла Formula воспользуемся блоком кода с строкой `(0..Math.Sin(x*360)...#50)*5;`. Для быстроты мы определяем диапазон с помощью формулы внутри него. Эта формула представляет собой функцию синуса. Функция синуса получает входные данные в градусах в Dynamo, поэтому для получения полной синусоиды необходимо умножить значения *x* (диапазон входных значений от 0 до 1) на *360*. Далее необходимо получить количество делений, соответствующее количеству точек управляющей сетки для каждого ряда, поэтому зададим пятьдесят подразделов, введя *#50*. Наконец, с помощью множителя 5 просто увеличим амплитуду преобразования, чтобы увидеть результат в области предварительного просмотра Dynamo.



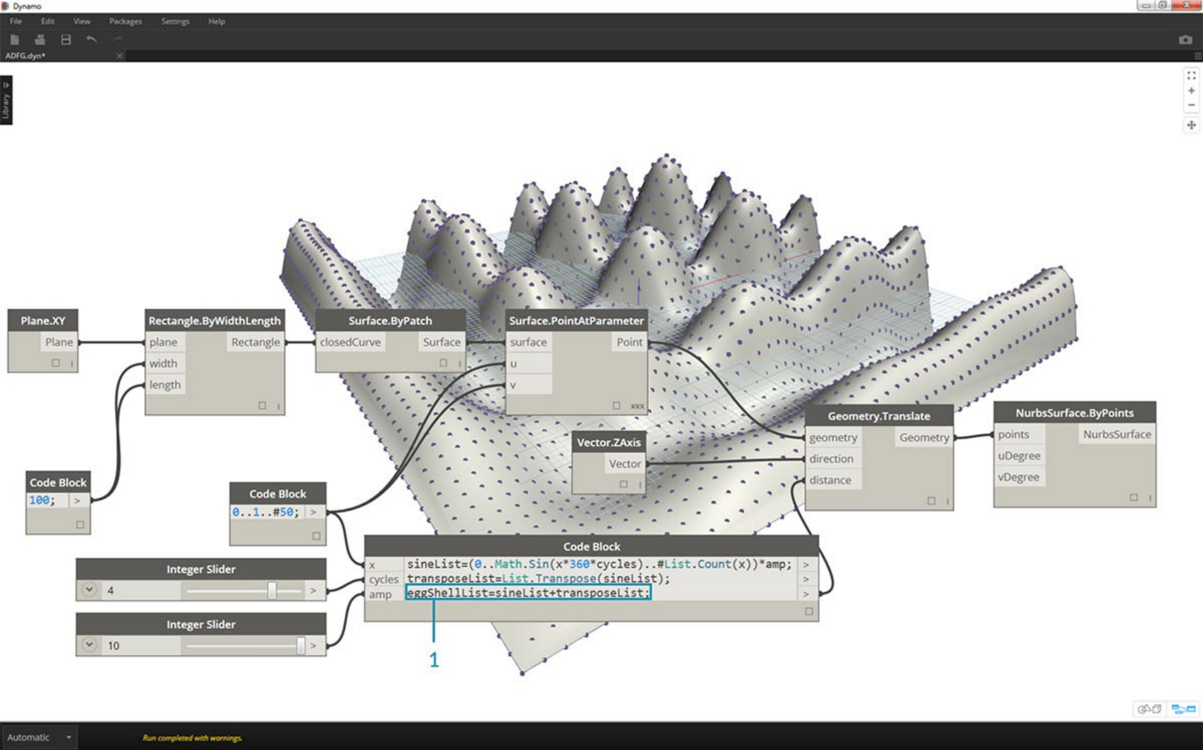
1. Хотя предыдущие блоки кода работали нормально, процесс был не полностью параметрическим. Так как необходимо динамически изменять параметры, заменим строку из предыдущего шага на `(0..Math.Sin(x*360*cycles)..#List.Count(x))*amp;`. Это позволит задавать значения на основе входных данных.



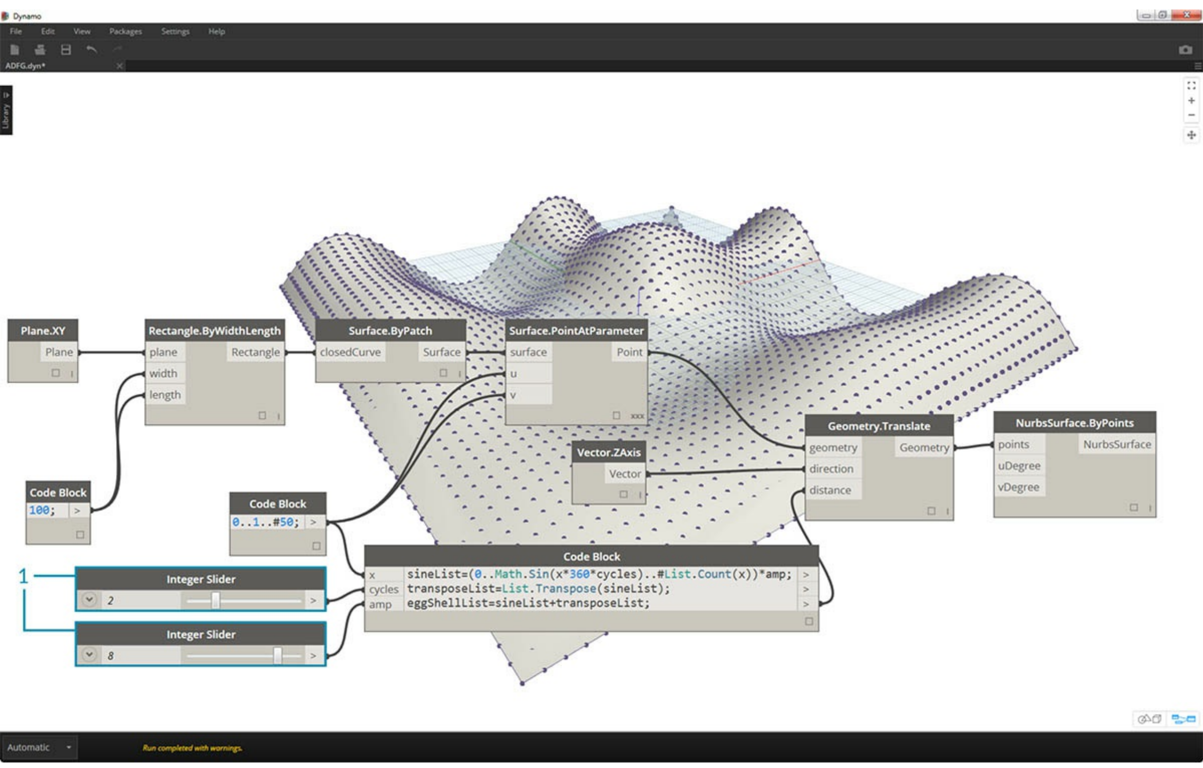
1. Изменим положение регуляторов (в диапазоне от 0 до 10) и получим интересный результат.



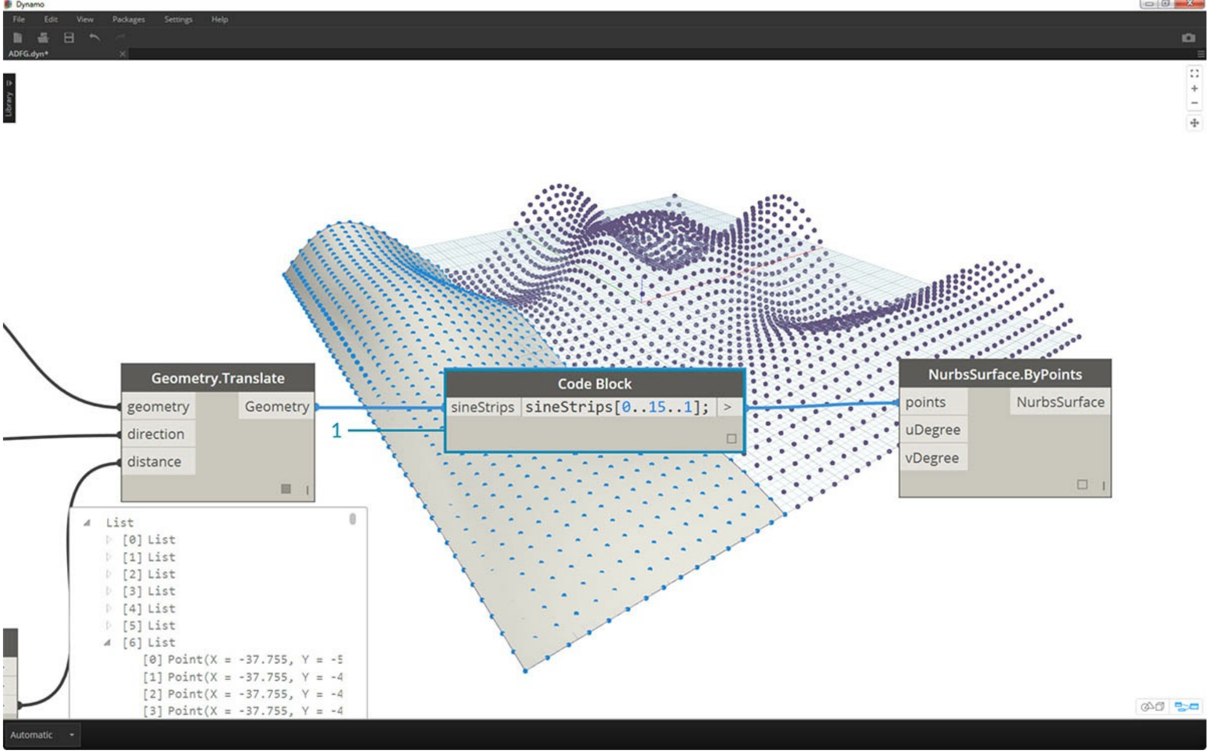
1. Транспонируем диапазон чисел, чтобы обратить направление волны: `transposeList = List.Transpose(sineList);`.



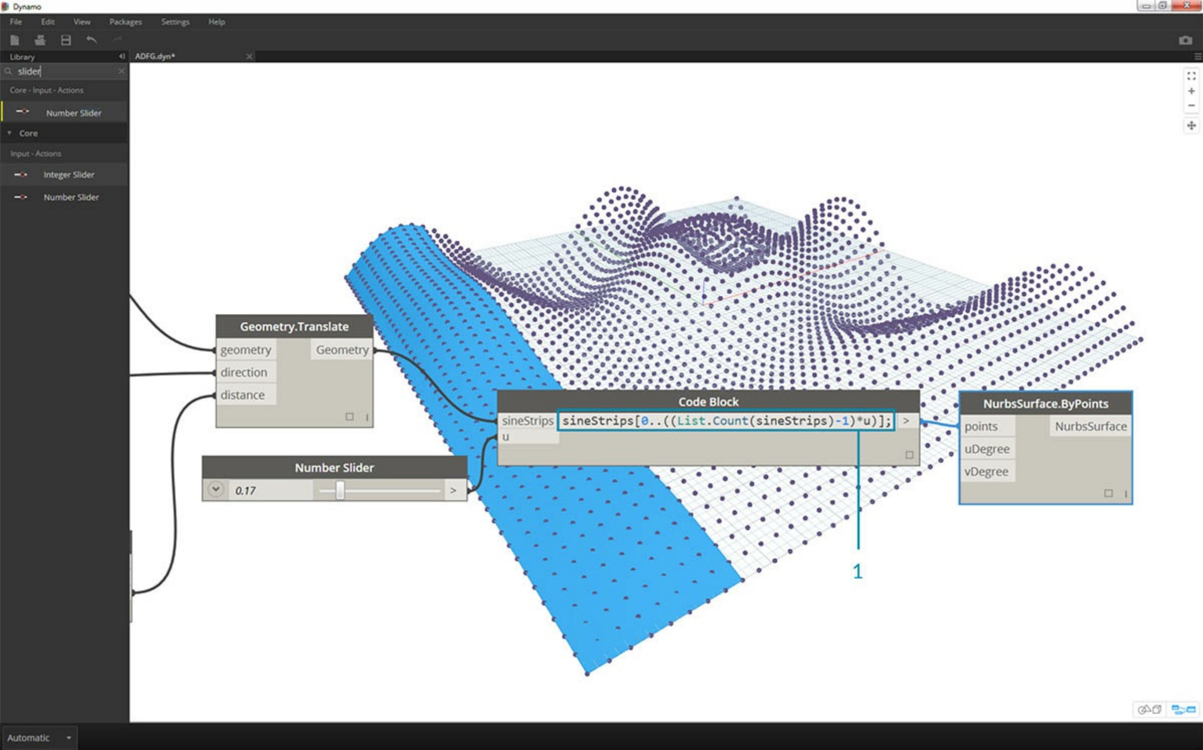
1. Добавим элементы sineList и TransitionList, чтобы получить деформированную яйцевидную поверхность: `eggShellList = sineList+transposeList;`.



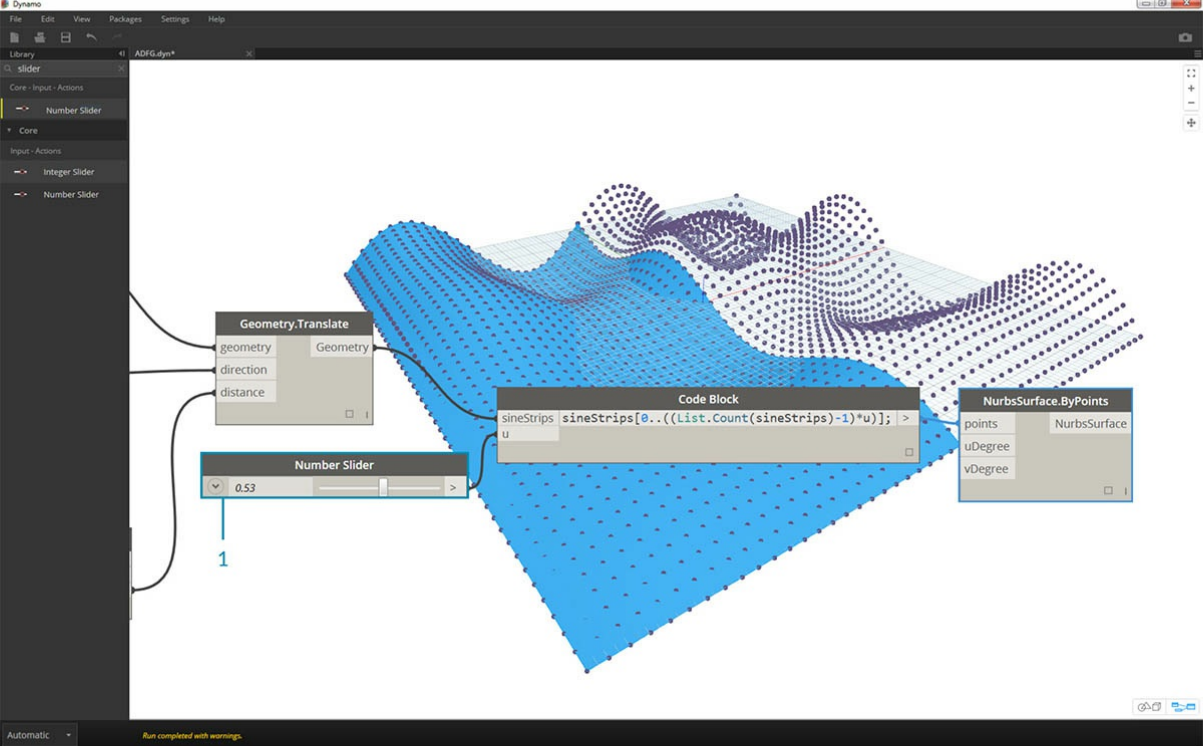
1. Снова изменим положение регуляторов, чтобы сбалансировать алгоритм.



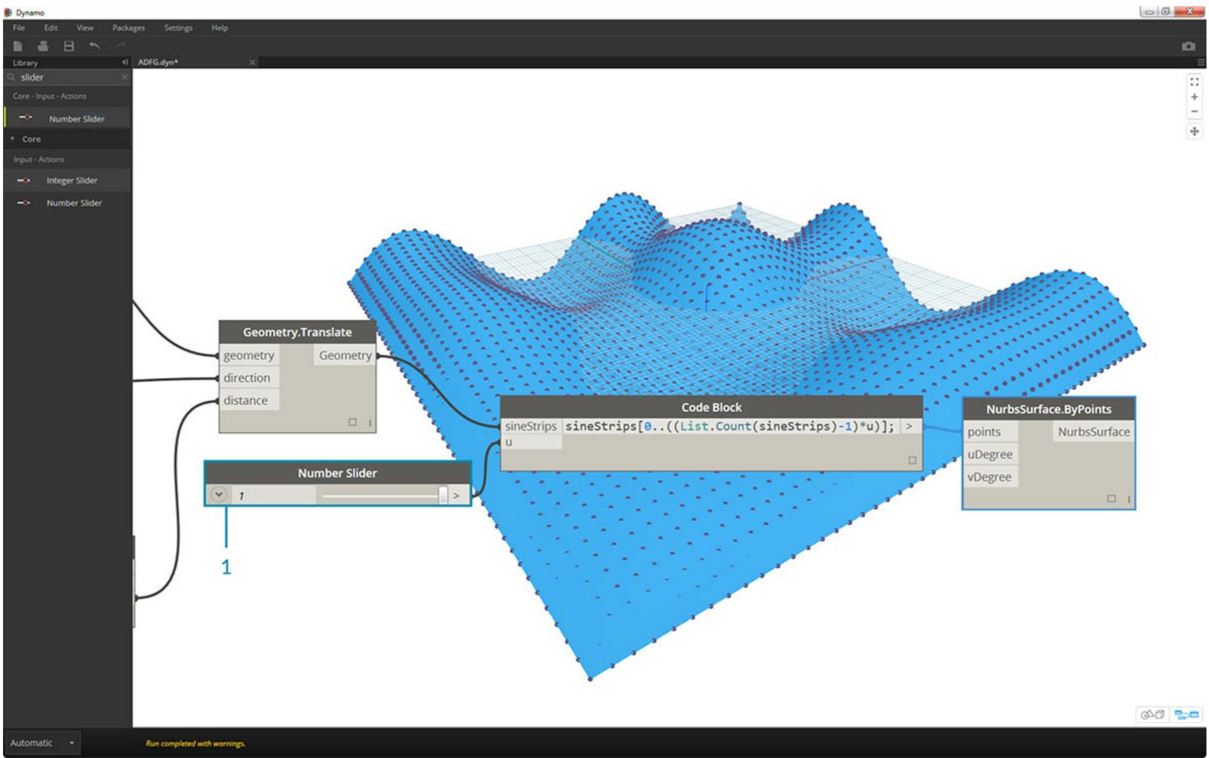
1. Осталось запросить отдельные элементы данных с помощью блока кода. Чтобы сформировать поверхность с определенным диапазоном точек, добавим блок кода, который показан на изображении выше, между узлами *Geometry.Translate* и *NurbsSurface.ByPoints*. В нем содержится строка текста: `sineStrips[0..15..1];`. С помощью нее будут выбраны первые 16 рядов точек (из 50). Если снова сгенерировать поверхность, можно увидеть, что была создана отдельная часть сетки точек.



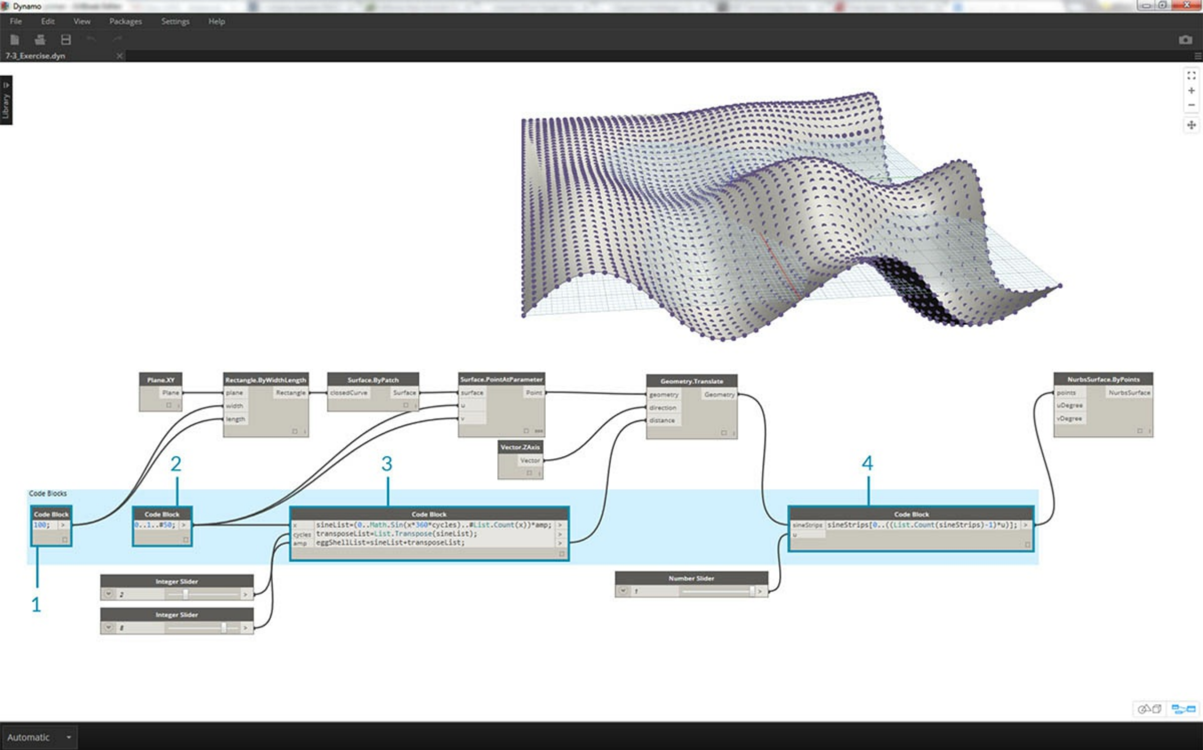
1. Наконец, чтобы сделать этот блок кода более параметрическим, для запроса воспользуемся регулятором с диапазоном от 0 до 1. Для этого введем следующую строку кода: `sineStrips[0..((List.Count(sineStrips)-1)*u)];`. Для большей ясности — данная строка кода позволяет представить длину списка в виде множителя с диапазоном от 0 до 1.



1. Значение *.53* регулятора позволяет создать поверхность сразу за центром сетки.



1. Как и ожидалось, если с помощью регулятора указать значение*1*, поверхность будет создана из всей сетки точек.



На полученном графике можно выделить блоки кода и увидеть их функции.

1. Первый блок кода заменяет узел *Number*.
2. Второй блок кода заменяет узел *Number Range*.
3. Третий блок кода заменяет узел *Formula* (а также *List.Transpose*, *List.Count* и *Number Range*).
4. Четвертый блок кода запрашивает элементы в списке списков, заменяя узел *List.GetItemAtIndex*.

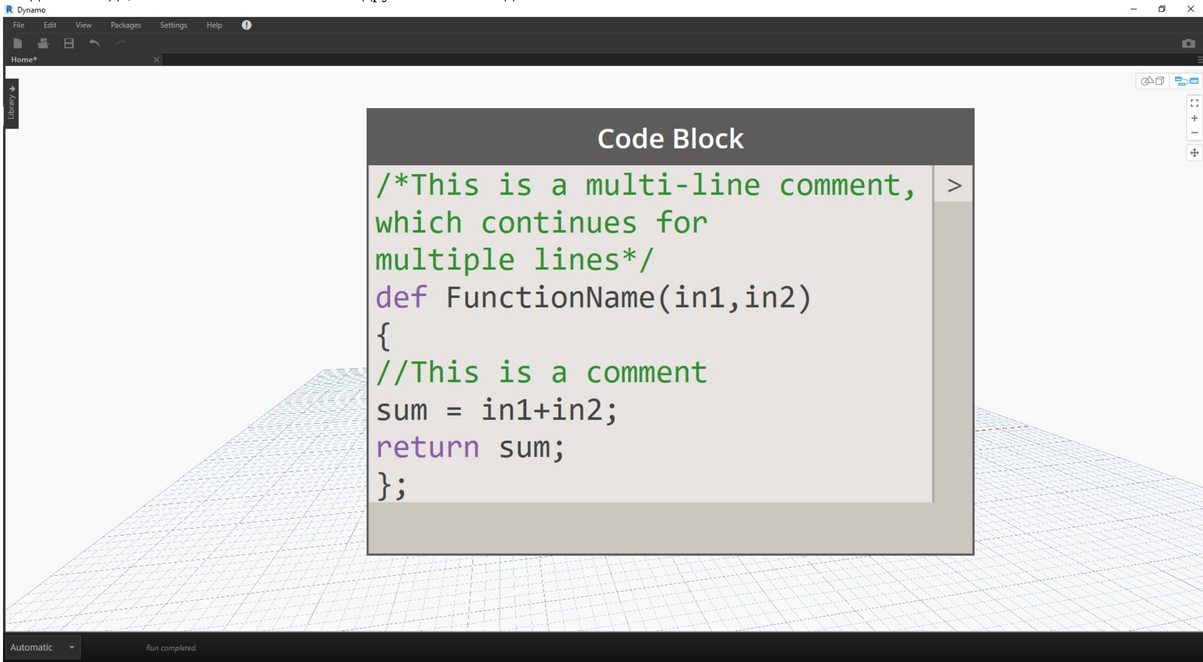
Функции

Функции в узлах Code Block

Узлы Code Block позволяют вставлять в график функции, к которым смогут обращаться другие компоненты программы Dynamo. При этом в параметрическом файле создается еще один слой управления, который можно рассматривать как текстовую версию пользовательского узла. При этом «родительский» узел Code Block легко доступен и может находиться в любом месте графика. И никаких проводов!

Родительский узел Code Block

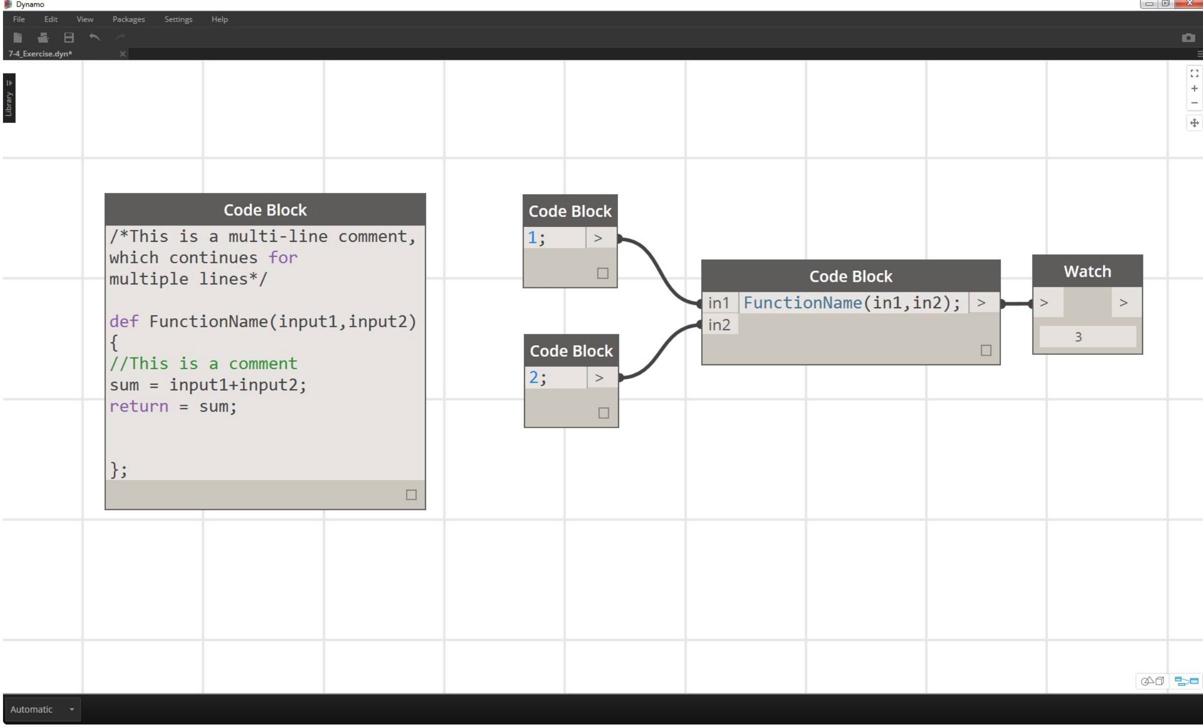
Первая строка содержит ключевое слово «def», затем название функции и наименования входных данных в скобках. Тело функции заключено в фигурные скобки. Значение возвращается с помощью оператора «return =». В узлах Code Block, определяющих функцию, отсутствуют порты ввода и вывода, так как они вызываются из других блоков кода.




```
/*This is a multi-line comment,
which continues for
multiple lines*/
def FunctionName(in1,in2)
{
//This is a comment
sum = in1+in2;
return sum;
};
```

Дочерние узлы Code Block

Функцию можно вызвать с помощью другого узла Code Block в том же файле, указав имя и такое же число аргументов. Этот процесс аналогичен использованию готовых узлов из библиотеки.

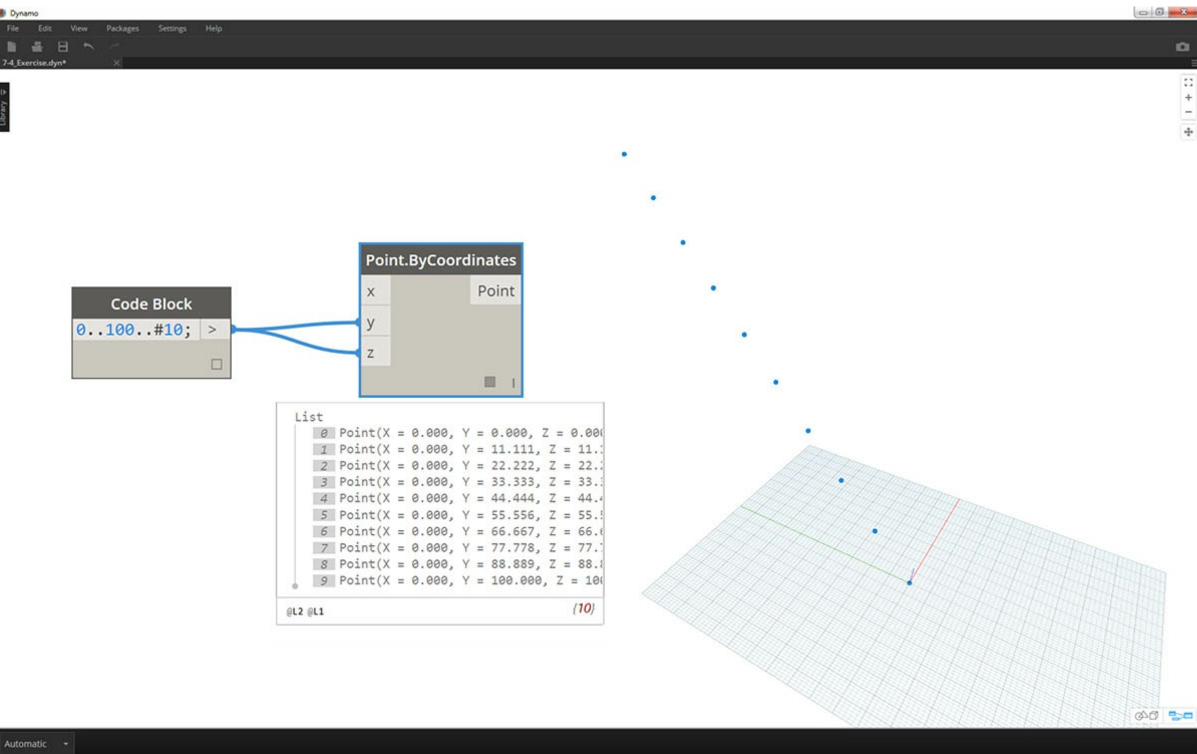



```
FunctionName(in1,in2);
```

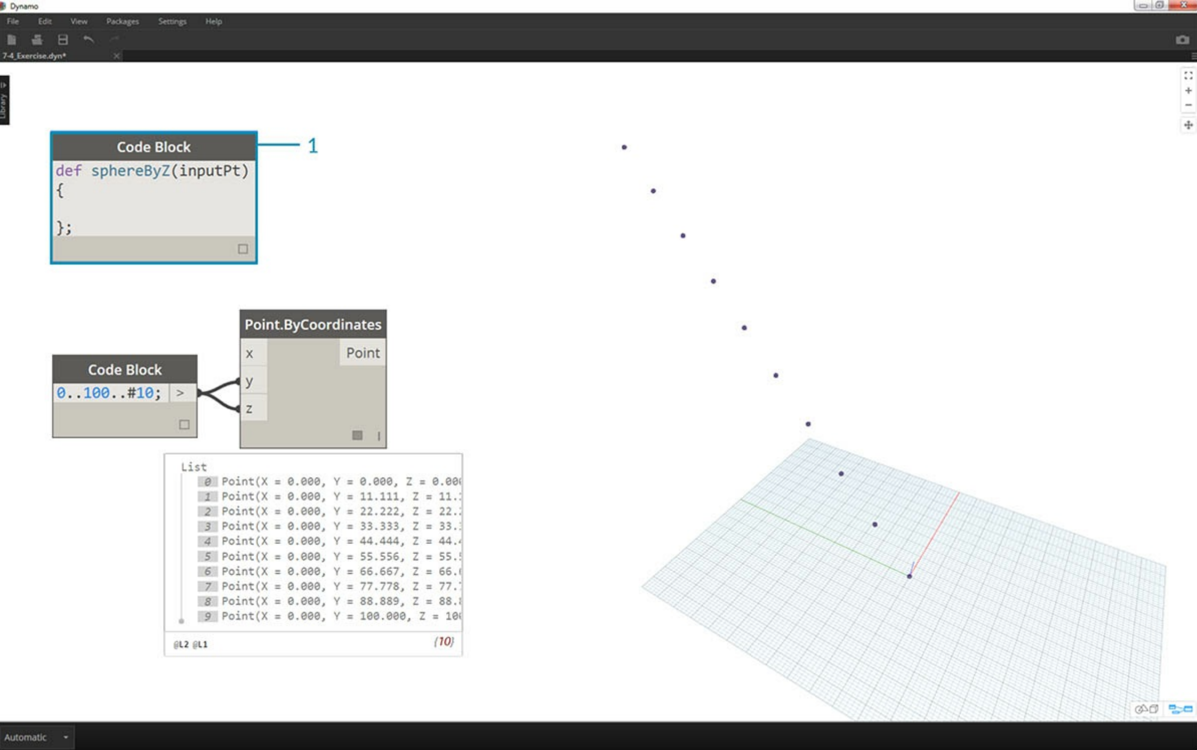
Упражнение

> Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. [Functions_SphereByZ.dyn]

В этом упражнении мы создадим типовую программу, которая будет генерировать сферы на основе вводимого списка точек. Радиус сфер определяется свойством Z каждой точки.



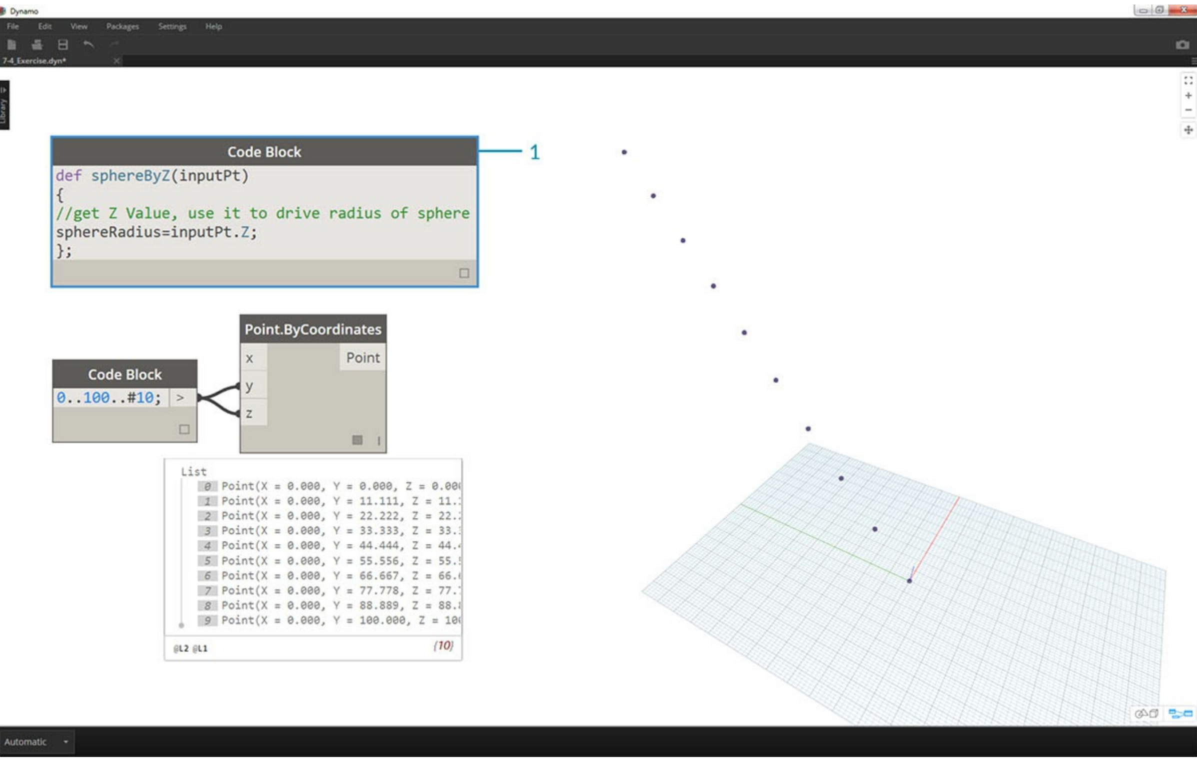
> Для начала зададим десять числовых значений в диапазоне от 0 до 100. Соединим этот узел с узлом *Point.ByCoordinates* для создания диагональной линии.



1. Создайте *Code Block* и введите в него следующую строку кода:

```
def sphereByZ(inputPt){
};
```

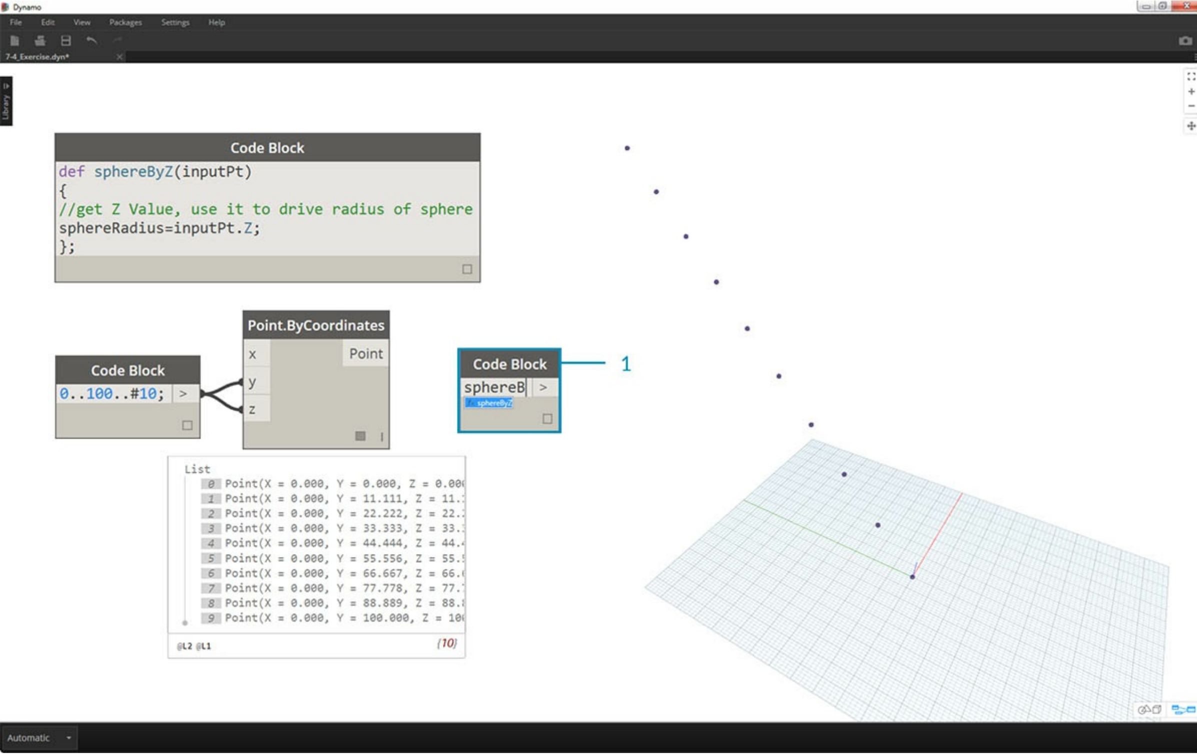
inputPt — это имя, заданное для точек, которые будут определять функцию. В настоящее время эта функция не выполняет никаких действий, но она будет дополнена по ходу работы.



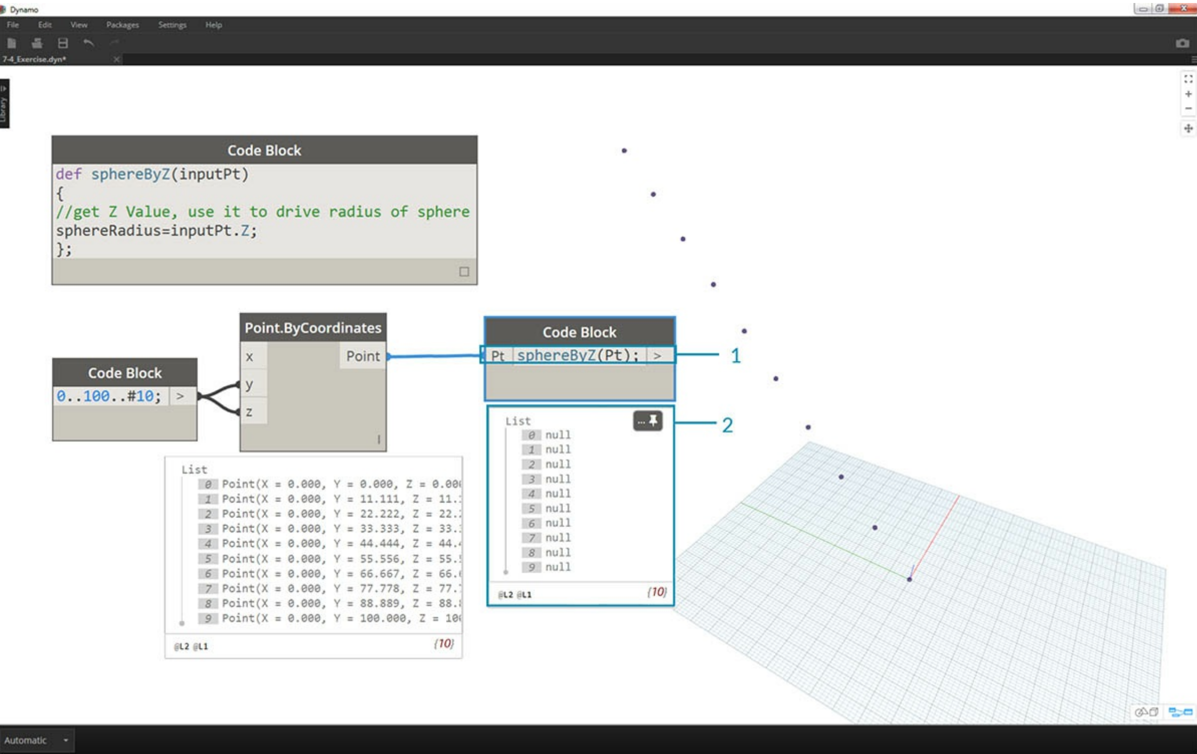
1. Дополним функцию *Code Block*, разместив комментарий и переменную *sphereRadius*, которая запрашивает положение каждой точки по оси *Z*. Напомним, что метод *inputPt.Z* не нужно заключать в скобки. Это *запрос* свойств существующего элемента, поэтому указывать входные данные не требуется.

```
def sphereByZ(inputPt,radiusRatio)
{
```

```
//get Z Value, use it to drive radius of sphere
sphereRadius=inputPt.Z;
};
```



1. Вызовем функцию, созданную в другом узле *Code Block*. Если дважды щелкнуть в активном окне для создания нового узла *Code Block* и ввести *sphereB*, то вы увидите, что Dynamo предложит использовать созданную выше функцию *sphereByZ*. Функция была добавлена в библиотеку IntelliSense! Неплохо.

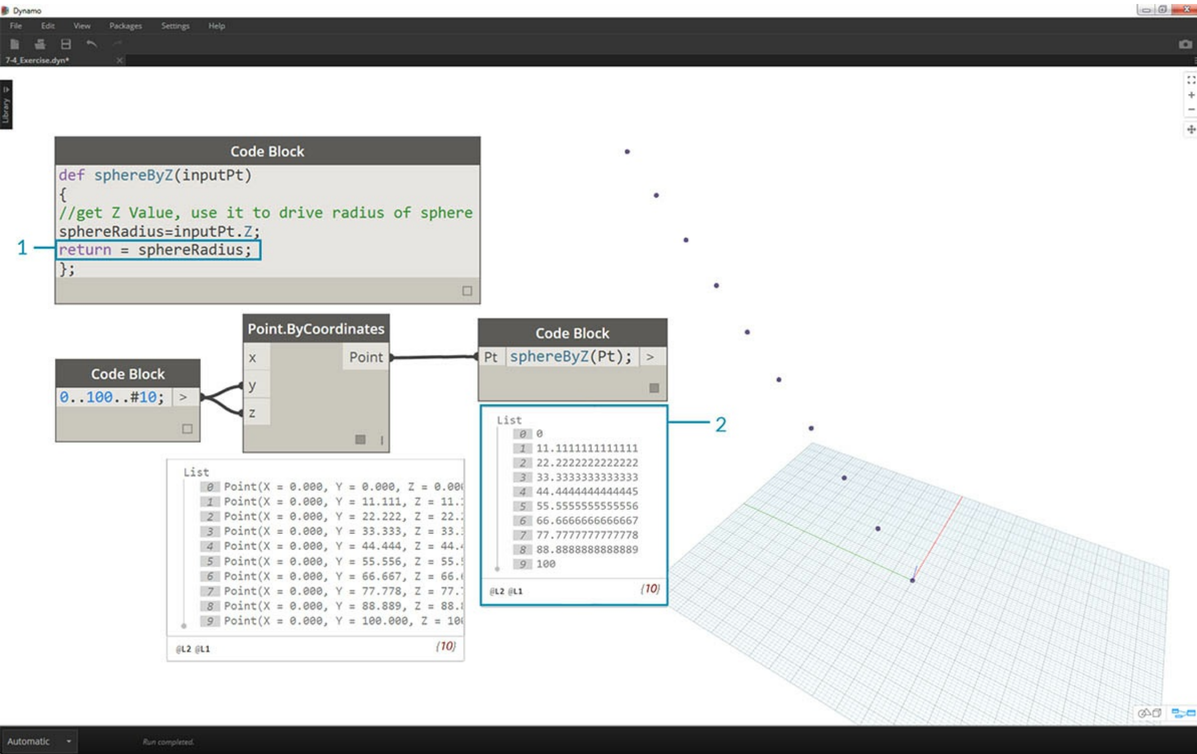


1. Теперь вызовем функцию и создадим переменную *Pt*, чтобы использовать созданные ранее точки.

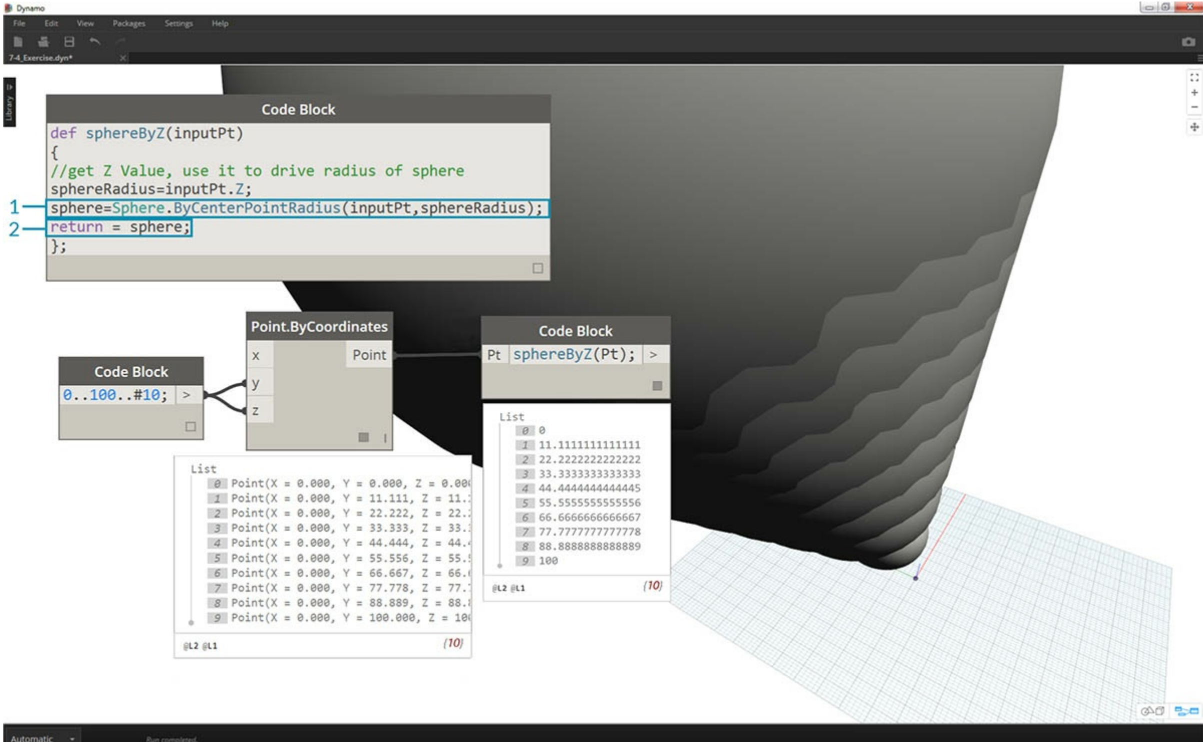
   ```
   sphereByZ(Pt)
   ```

1. Обратите внимание, что на выходе мы получили нулевые значения. С чем это связано? При определении функции был задан расчет переменной *sphereRadius*, но не было указано, что именно функция должна *возвращать* в качестве *выходных данных*. Наших следующим

шагом будет исправление этого упущения.

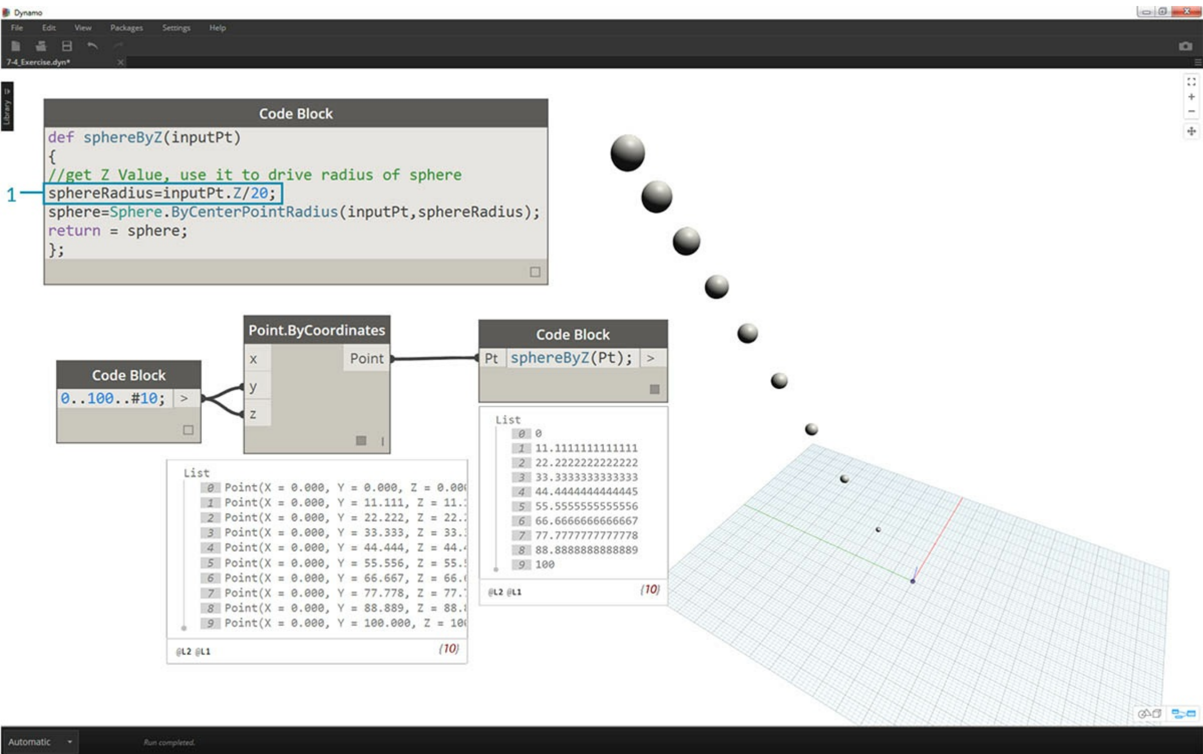


1. Это важный шаг: необходимо задать выходные данные функции, добавив строку `return = sphereRadius` в функцию *sphereByZ*.
2. Теперь *Code Block* выводит координаты Z каждой точки.

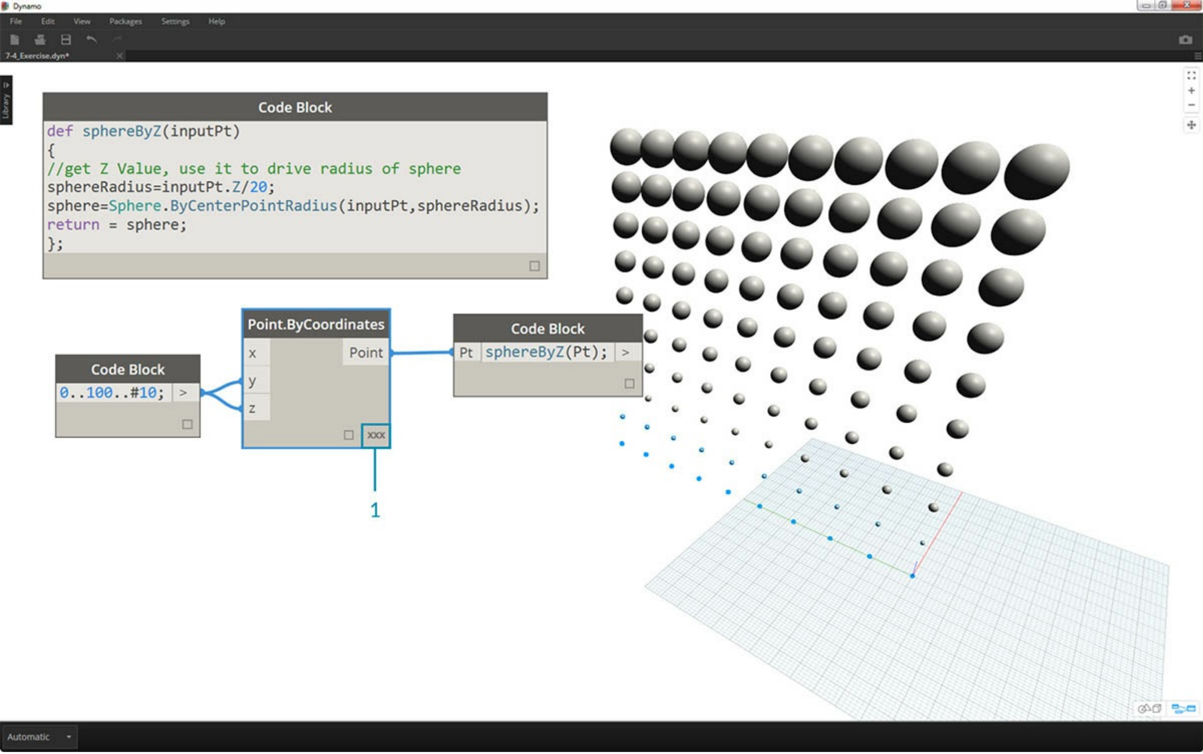


Создадим сферы, отредактировав *родительскую* функцию.

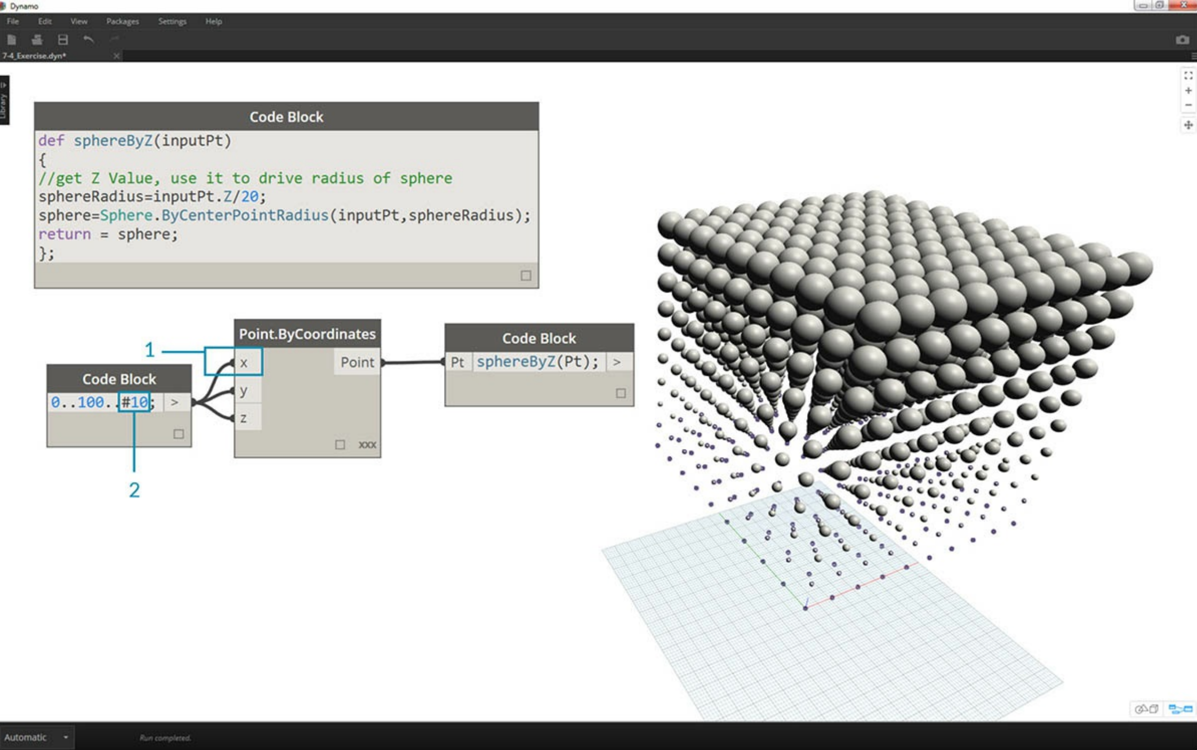
1. Сначала определим сферу с помощью следующей строки кода: `sphere=Sphere.ByCenterPointRadius(inputPt,sphereRadius);`.
2. Затем изменим возвращаемое значение на *sphere* вместо *sphereRadius*: `return = sphere;`. В области предварительного просмотра Dynamo появились огромные сферы.



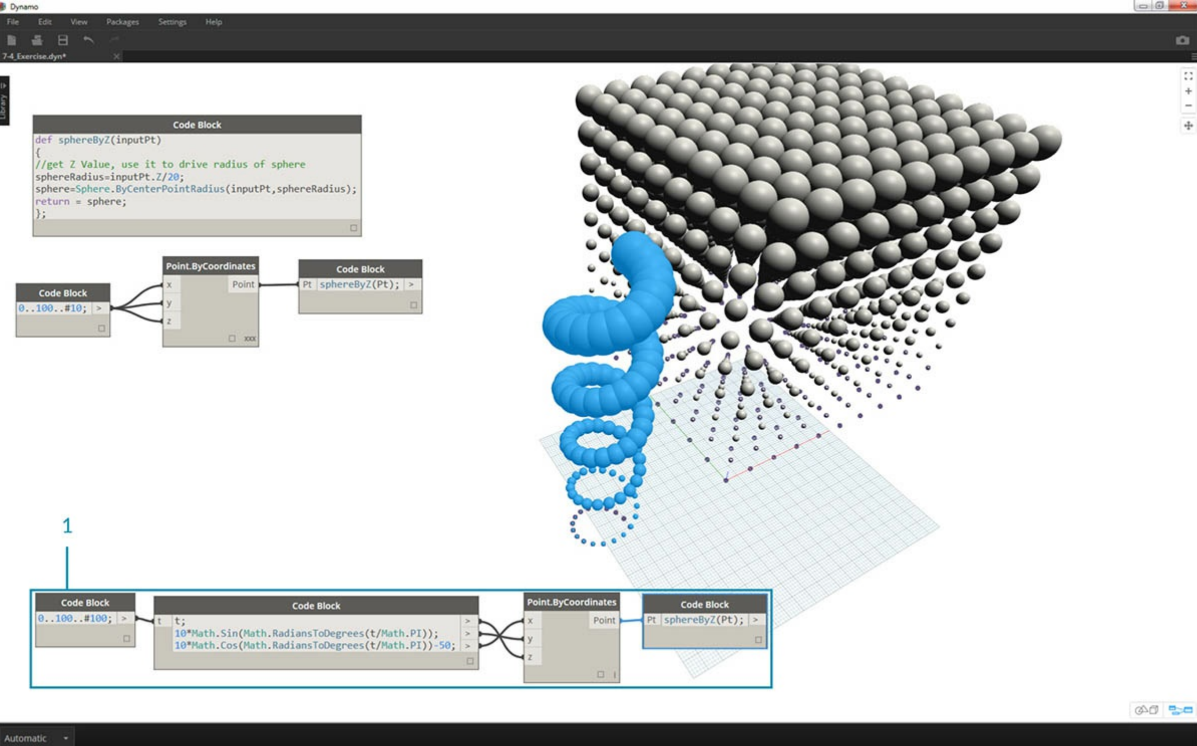
1. Чтобы отрегулировать размер этих сфер, изменим значение *sphereRadius*, добавив делитель: `sphereRadius = inputPt.Z/20;`. Теперь можно различить отдельные сферы и проследить связь между радиусом и значением Z.



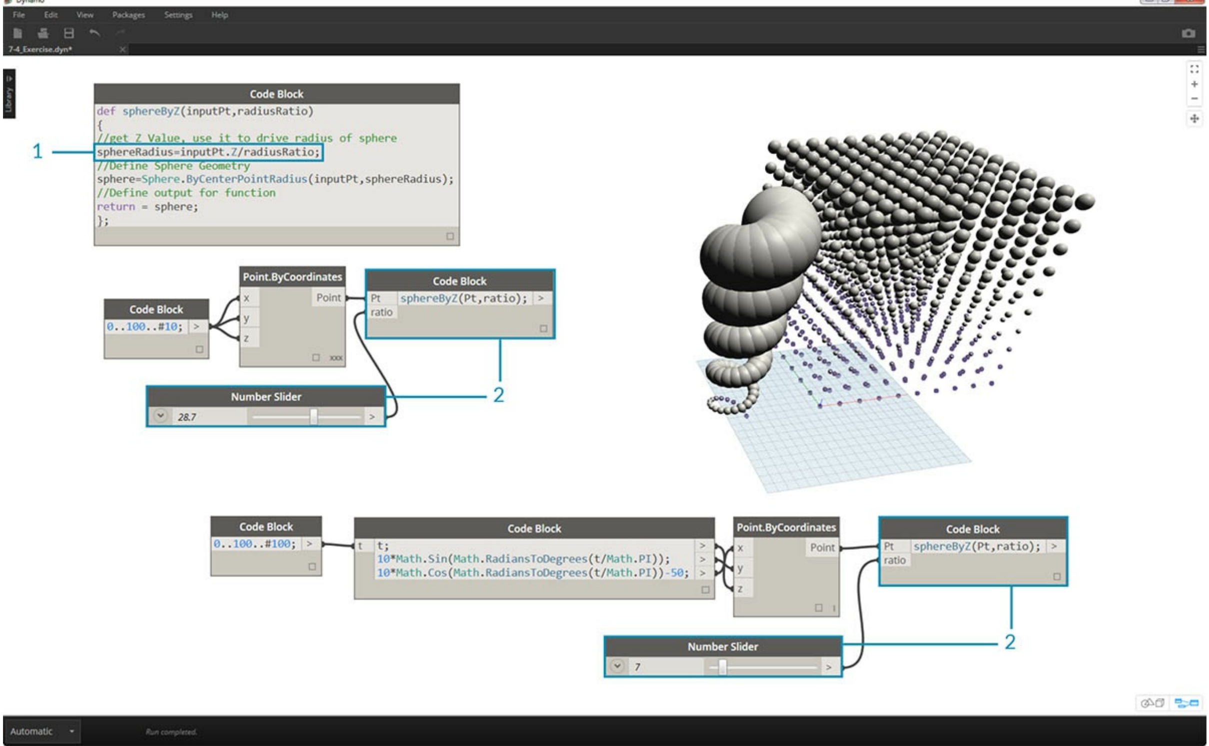
1. В узле *Point.ByCoordinates* изменим режим переплетения с *Кратчайший* на *Декартово произведение*, в результате чего будет создана сетка точек. Функция *sphereByZ* все еще действует, поэтому все точки создают сферы с радиусами, основанными на значениях Z.



1. Для проверки соединим исходный список чисел с входным портом X узла *Point.ByCoordinates*. Мы получили куб из сфер.
2. Примечание. Если для расчета компьютеру требуется много времени, попробуйте уменьшить значение *#10* и задать вместо него, например, *#5*.



1. Поскольку созданная нами функция *sphereByZ* является типовой, можно вызвать спираль из предыдущего урока и применить эту функцию к ней.



И последний шаг: настройка коэффициента радиуса с помощью пользовательских параметров. Для этого необходимо создать новый входной порт для функции и заменить делитель *20* параметром.

1. Изменим определение *sphereByZ* на следующее:

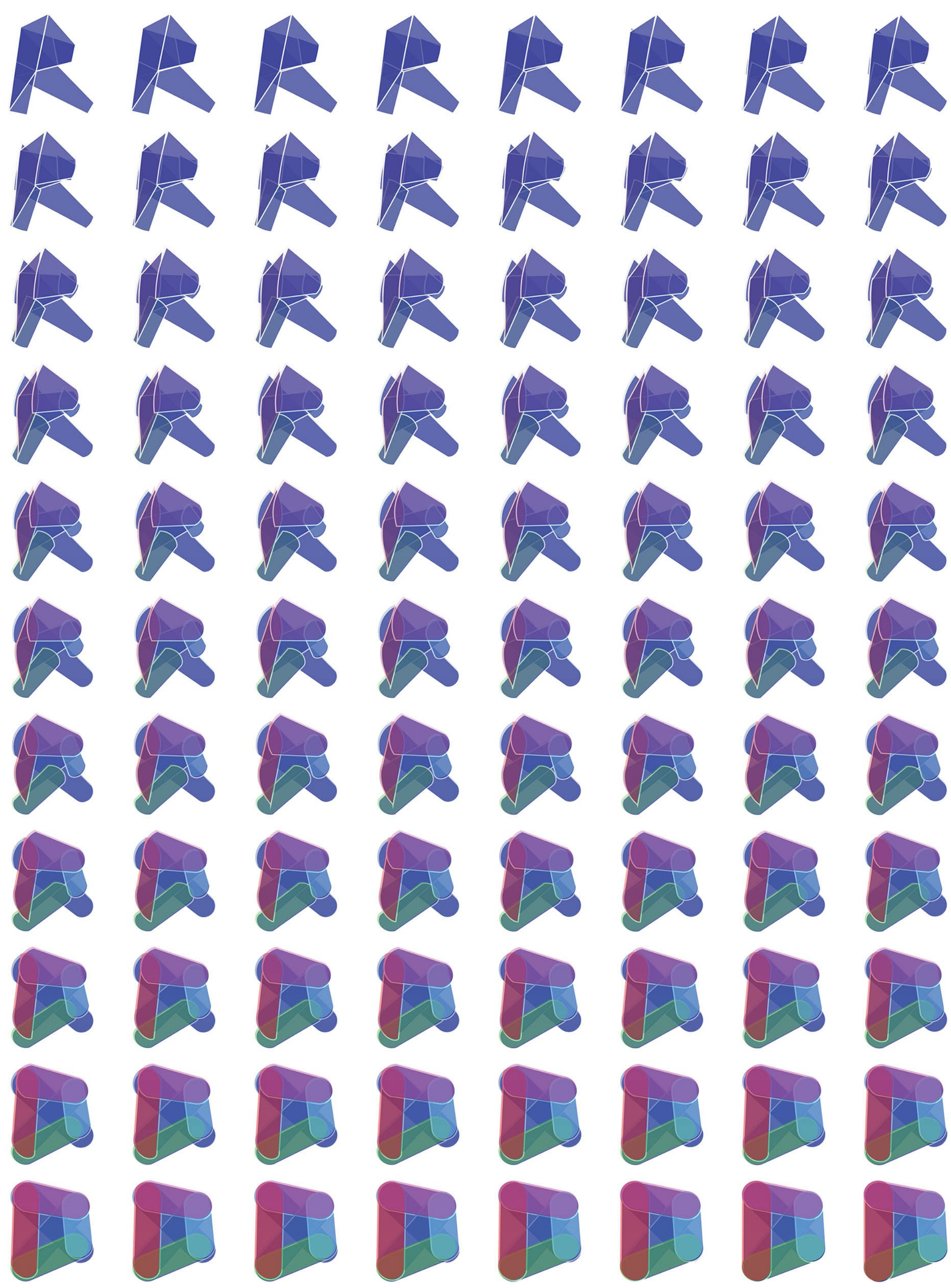
```
def sphereByZ(inputPt,radiusRatio)
{
//get Z Value, use it to drive radius of sphere
sphereRadius=inputPt.Z/radiusRatio;
//Define Sphere Geometry
sphere=Sphere.ByCenterPointRadius(inputPt,sphereRadius);
//Define output for function
return sphere;
};
```

1. Обновим дочерние узлы Code Block, добавив переменную *ratio* к входным данным: `sphereByZ(Pt,ratio);`. Добавим регулятор к только что созданному узлы Code Block. Теперь можно изменять размер радиусов на основе коэффициента.

Dynamo для Revit

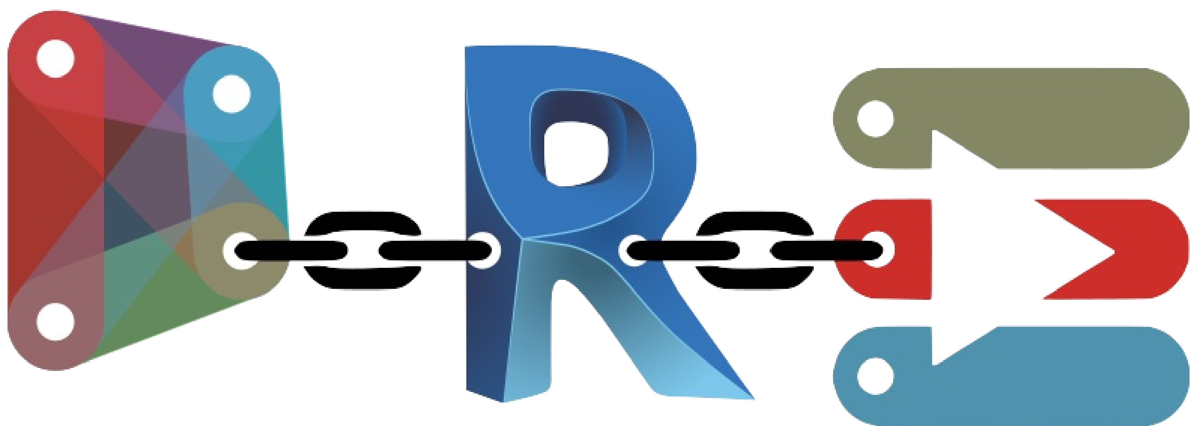
Dynamo для Revit

Dynamo — это гибкая среда, предназначенная для совместного использования с широким спектром программ, однако изначально она разрабатывалась для использования в Revit. Средства визуального программирования обеспечивают мощные дополнительные возможности для информационного моделирования зданий (BIM). Dynamo включает отдельный набор узлов, разработанных специально для Revit, а также библиотеки сторонних разработчиков из числа участников сообщества пользователей в сфере архитектуры и строительства. В этой главе рассматриваются основные принципы работы с Dynamo в Revit.



Подключение к Revit

Подключение к Revit



Надстройка Dynamo для Revit расширяет возможности информационного моделирования зданий за счет среды логики и данных, предоставляемой графическим редактором алгоритмов. Ее гибкие возможности в сочетании с обширной базой данных Revit позволяют перевести BIM на новый уровень.

В этой главе рассматриваются рабочие процессы Dynamo для BIM. Почти каждый раздел включает упражнения, так как знакомство с графическим редактором алгоритмов для BIM эффективнее всего проводится на практике. Но для начала изучите истоки этой программы.

#

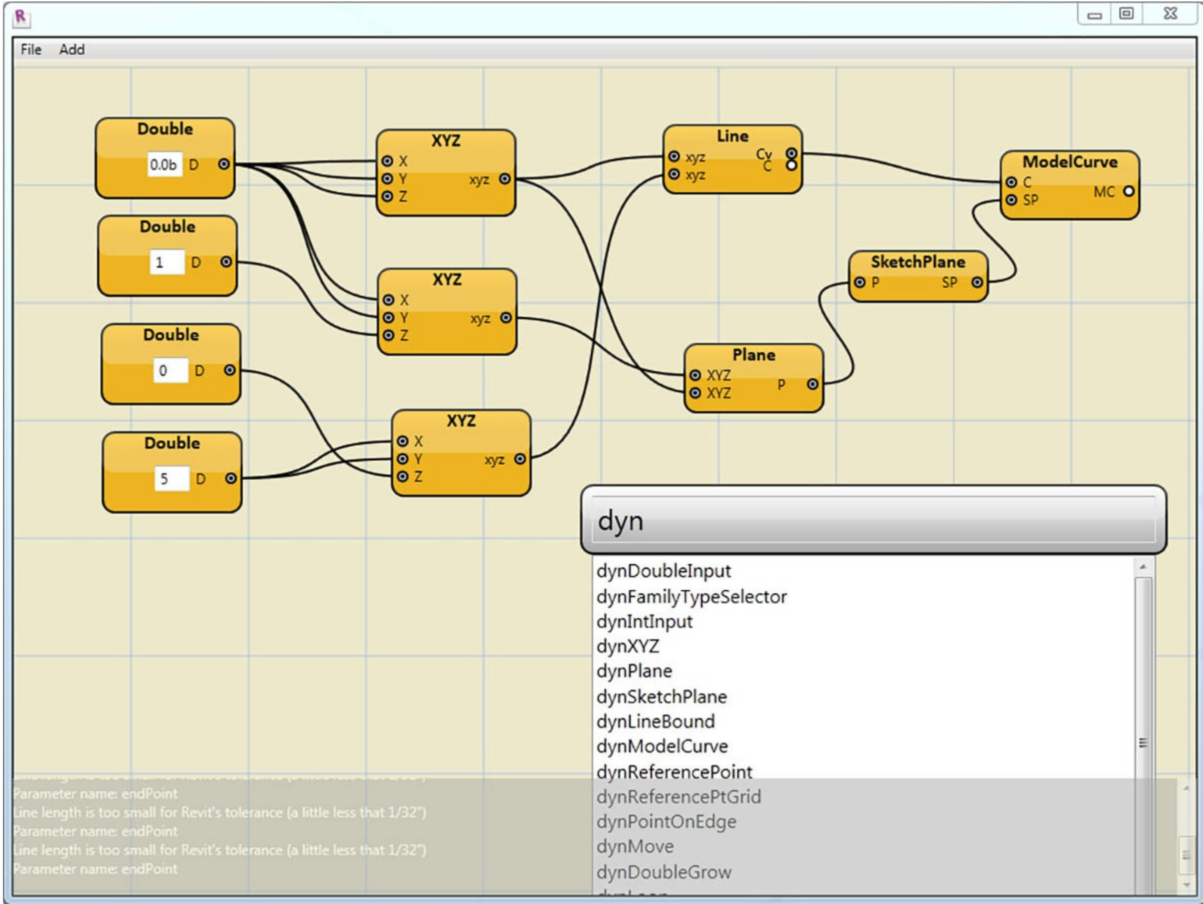
Совместимость с разными версиями Revit

Поскольку Revit и Dynamo постоянно обновляются, в какой-то момент используемая версия Revit может оказаться несовместимой с установленной на компьютере версией Dynamo для Revit. Ниже приведены сведения о том, какие версии надстройки Dynamo для Revit совместимы с программой Revit.

Версия Revit	Первая стабильная версия Dynamo	Последняя поддерживаемая версия Dynamo для Revit
2013	0.6.1	0.6.3
2014	0.6.1	0.8.2
2015	0.7.1	1.2.1
2016	0.7.2	1.3.2
2017	0.9.0	1.3.4/2.0.3
2018	1.3.0	1.3.4/2.0.3
2019	1.3.3	1.3.4/2.0.3
2020	2.1.0 (Revit 2020 теперь включает Dynamo и обновляется в соответствии с графиком обновлений Revit.)	Отсутствует

#

История Dynamo

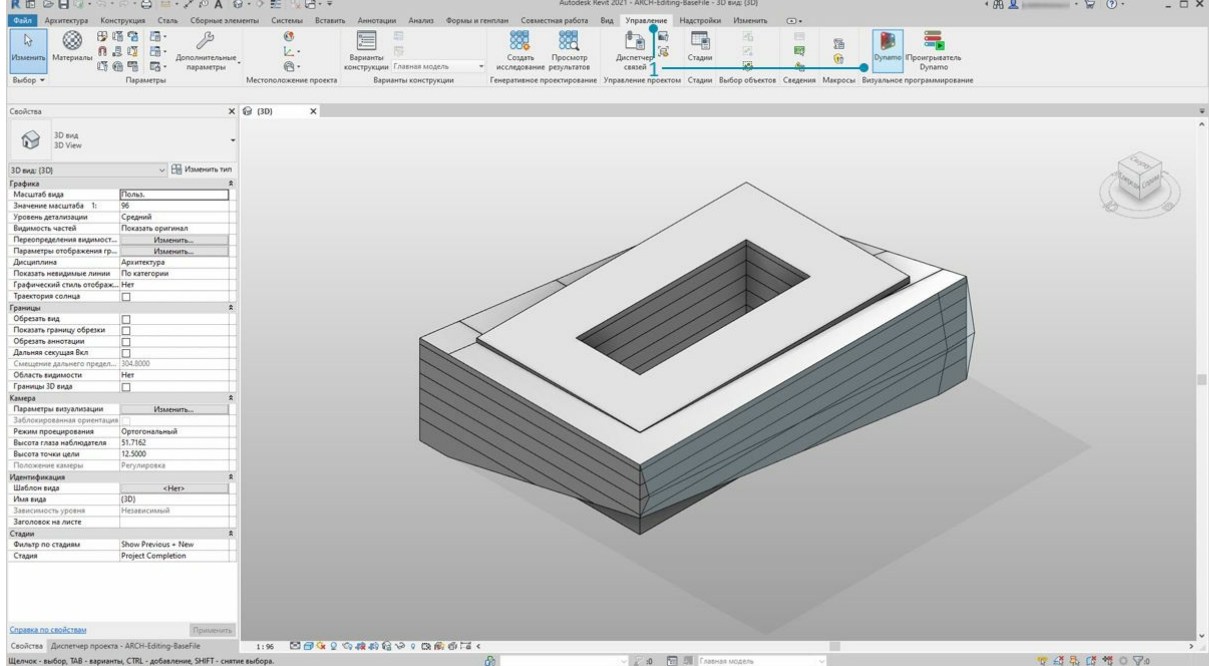


Благодаря упорству разработчиков и активному вкладу сообщества пользователей этот проект прошел большой путь от скромной надстройки до того, чем он является сейчас.

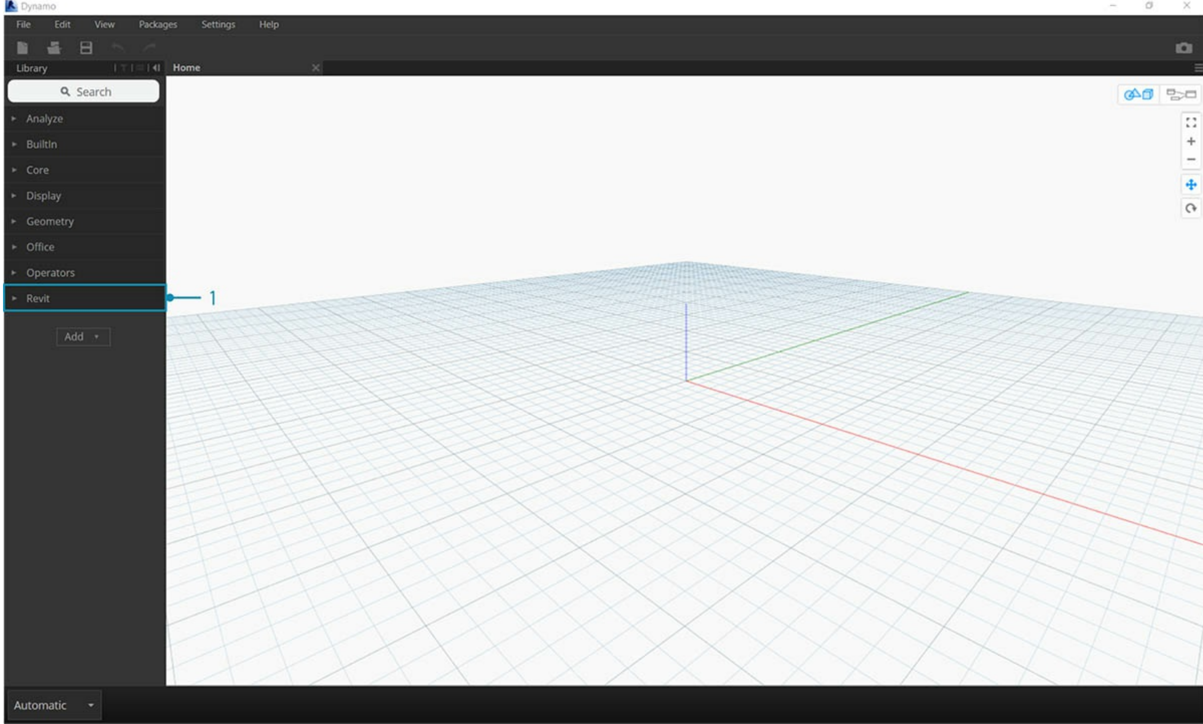
Изначально приложение Dynamo разрабатывалось как средство оптимизации рабочих процессов Revit для сферы архитектуры и строительства. Хотя в Revit для каждого проекта создается обширная база данных, пользователи со средним уровнем подготовки могут столкнуться с проблемами при доступе к информации за пределами интерфейса. Revit располагает полнофункциональным интерфейсом API, благодаря которому сторонние разработчики могут создавать специализированные инструменты. И программисты пользовались этим API на протяжении многих лет, однако стоит признать, что создание текстовых сценариев под силу далеко не всем. Благодаря Dynamo и понятному графическому редактору алгоритмов данные Revit становятся более доступными для пользователей с разными уровнями подготовки.

Используя базовые узлы Dynamo в сочетании со специализированными узлами Revit, пользователи могут существенно расширить параметрические рабочие процессы для обеспечения совместимости, выпуска документации, анализа и генерации объектов. Dynamo позволяет автоматизировать повседневные рабочие процессы и направить все силы проектировщиков на анализ и изучение проектов.

Работа с Dynamo в Revit



1. В редакторе проектов или семейств Revit перейдите в раздел «Надстройки» и выберите *Dynamo*. Примечание. Надстройка Dynamo будет работать только в том файле, в котором она была открыта.



1. При открытии Dynamo в Revit отображается новая категория с именем *Revit*. Это мощное дополнение к пользовательскому интерфейсу, включающее узлы, специально предназначенные для использования в рабочих процессах Revit*.

** Примечание. При использовании семейства узлов, предназначенного для Revit, график Dynamo будет работать только при открытии в Dynamo для Revit. Если график, предназначенный для работы в Dynamo для Revit, открыть в однопользовательской версии Dynamo, то все узлы Revit из него пропадут.*

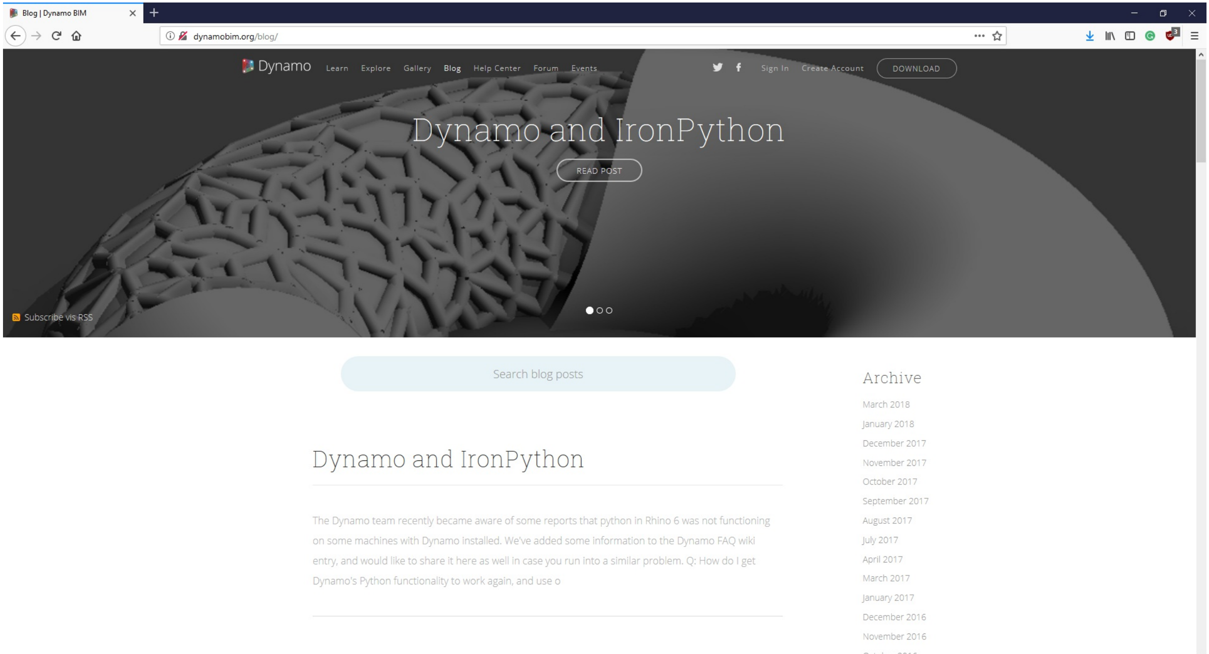
Замораживание узлов

Поскольку Revit является платформой, которая обеспечивает широкие возможности по управлению проектами, параметрические операции в Dynamo могут быть сложными, а их расчет может занимать много времени. Если Dynamo требуется много времени для расчета узлов, можно воспользоваться функцией заморозки, чтобы приостановить выполнение операций Revit во время создания графика. Для получения дополнительных сведений о замораживании узлов ознакомьтесь с соответствующим разделом в главе о твердых телах.

Сообщество

Приложение Dynamo изначально разрабатывалось для использования в сфере архитектуры и строительства, и наше постоянно растущее сообщество пользователей — это отличный ресурс для обучения и общения с отраслевыми экспертами. В сообщество пользователей Dynamo входят архитекторы, инженеры, программисты и проектировщики, которые любят изобретать и делиться своими изобретениями.

Dynamo — это проект с открытым исходным кодом, который постоянно развивается, и многие нововведения касаются Revit. Если вы пока что новичок, переходите на форум и начинайте задавать вопросы. Если вы программист и хотите принять участие в разработке Dynamo, посетите страницу проекта на сайте GitHub. Кроме того, если вас интересуют библиотеки сторонних разработчиков, рекомендуем ознакомиться с менеджером пакетов Dynamo. Многие из этих пакетов предназначены для сферы архитектуры и строительства. В этой главе рассматривается использование пакетов сторонних разработчиков для создания дополнительных панелей.

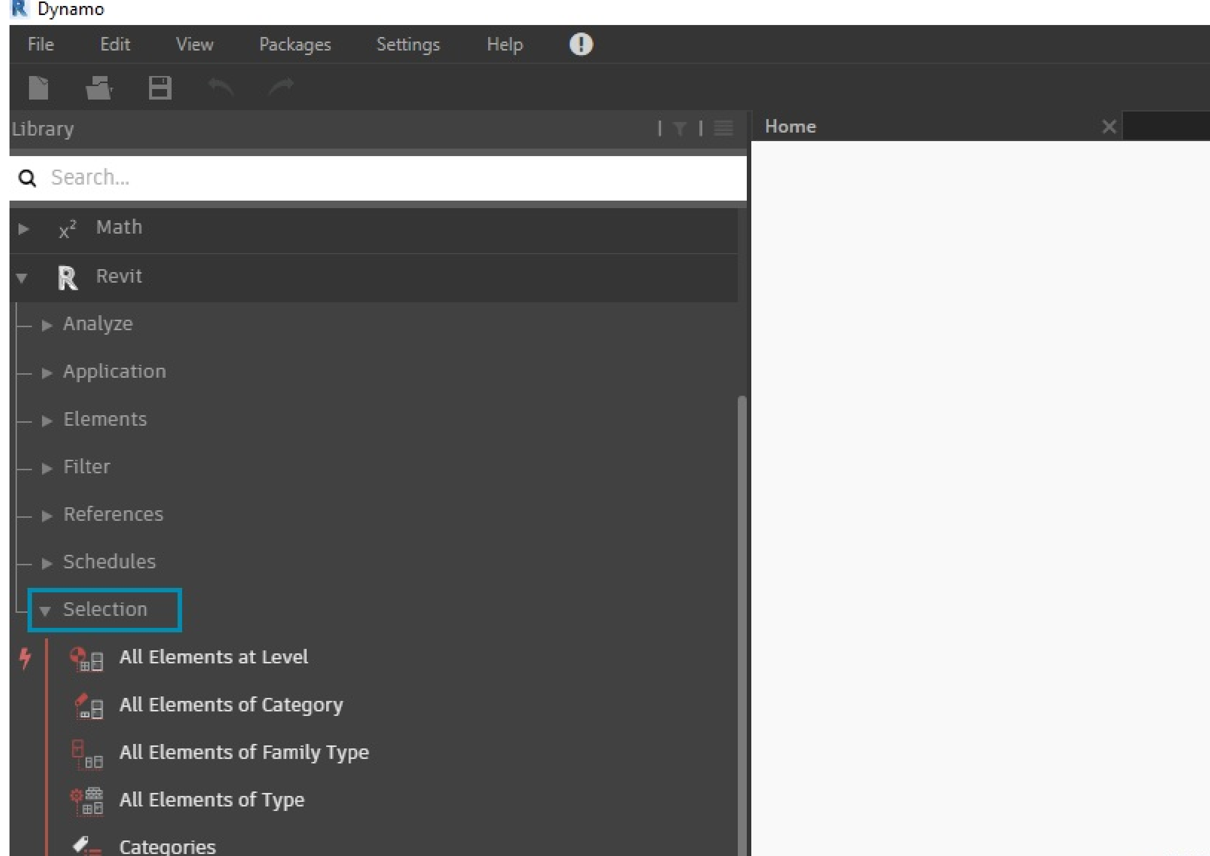


Разработчики Dynamo также ведут активный блог. Ознакомьтесь с последними публикациями, чтобы быть в курсе всех новостей.

Выбор

Выбор

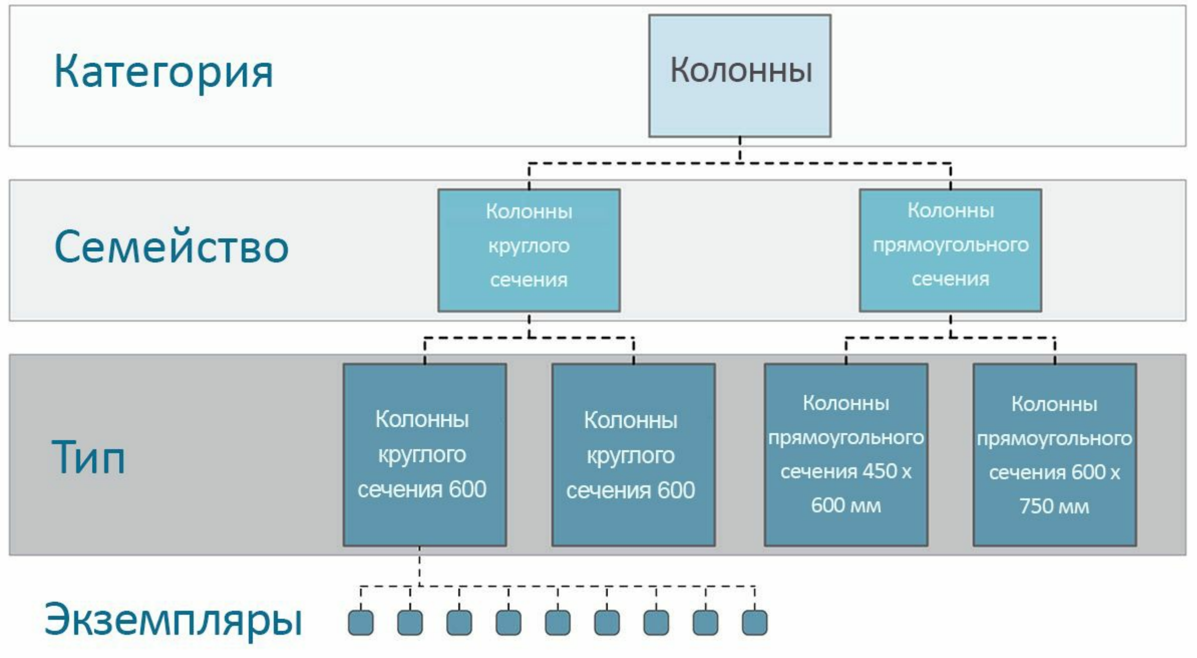
Revit — это насыщенная данными среда. Она поддерживает самые разные возможности выбора объектов, не ограничиваясь стандартным щелчком кнопкой мыши. Dynamo позволяет опрашивать базу данных Revit и динамически связывать элементы Revit с геометрией Dynamo во время выполнения параметрических операций.



Библиотека Revit, доступная в пользовательском интерфейсе, включает категорию Selection, которая предлагает несколько способов выбора геометрии.

Для использования правильного метода выбора элементов Revit необходимо иметь четкое представление об их иерархии. Необходимо выбрать все стены в проекте? Используйте выбор по категории. Требуется выбрать все кресла фирмы Eames для приемной в стиле 1950-х? Используйте выбор по семейству. Перед переходом к упражнению ознакомьтесь с кратким обзором иерархии Revit.

Иерархия Revit

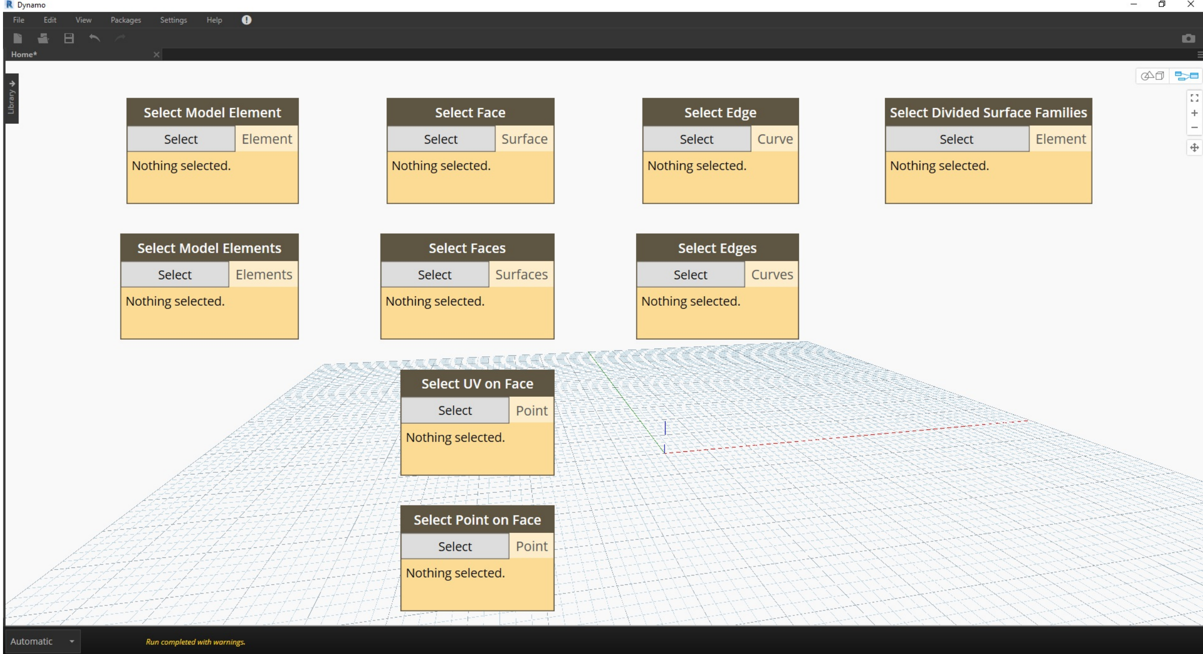


Помните таксономические ранги, которые проходят на уроках биологии? Согласно им, все организмы в природе делятся на царства, типы, классы, порядки, семейства, рода и виды. Элементы Revit упорядочены аналогичным образом. На базовом уровне иерархию Revit можно разделить на категории, семейства, типы* и экземпляры. Экземпляр представляет собой отдельный элемент модели (с уникальным идентификатором), а категория определяет типовую группу (например, «стены» или «полы»). Организация базы данных Revit подобным образом позволяет выбрать один элемент и все аналогичные ему элементы на основании указанного уровня иерархии.

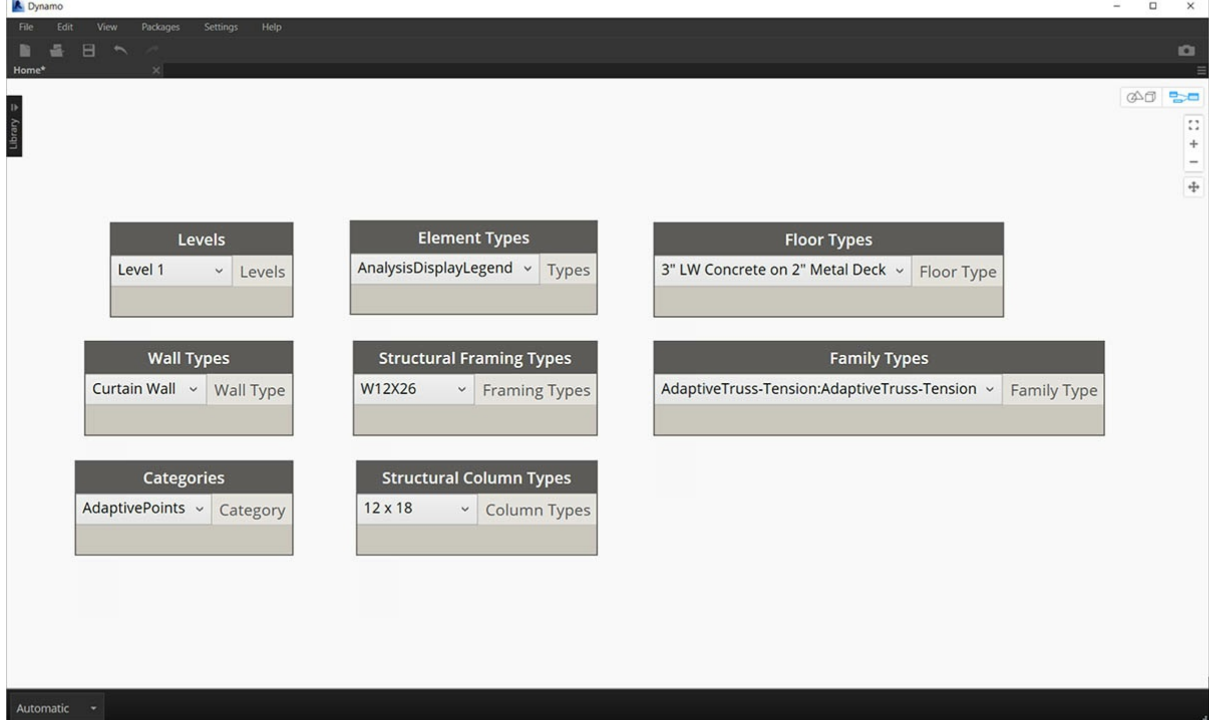
** Примечание. Определение типов в Revit отличается от определения типов в программировании. В Revit термин «тип» относится к ветви иерархии, а не к типу данных.*

Навигация по базе данных с помощью узлов Dynamo

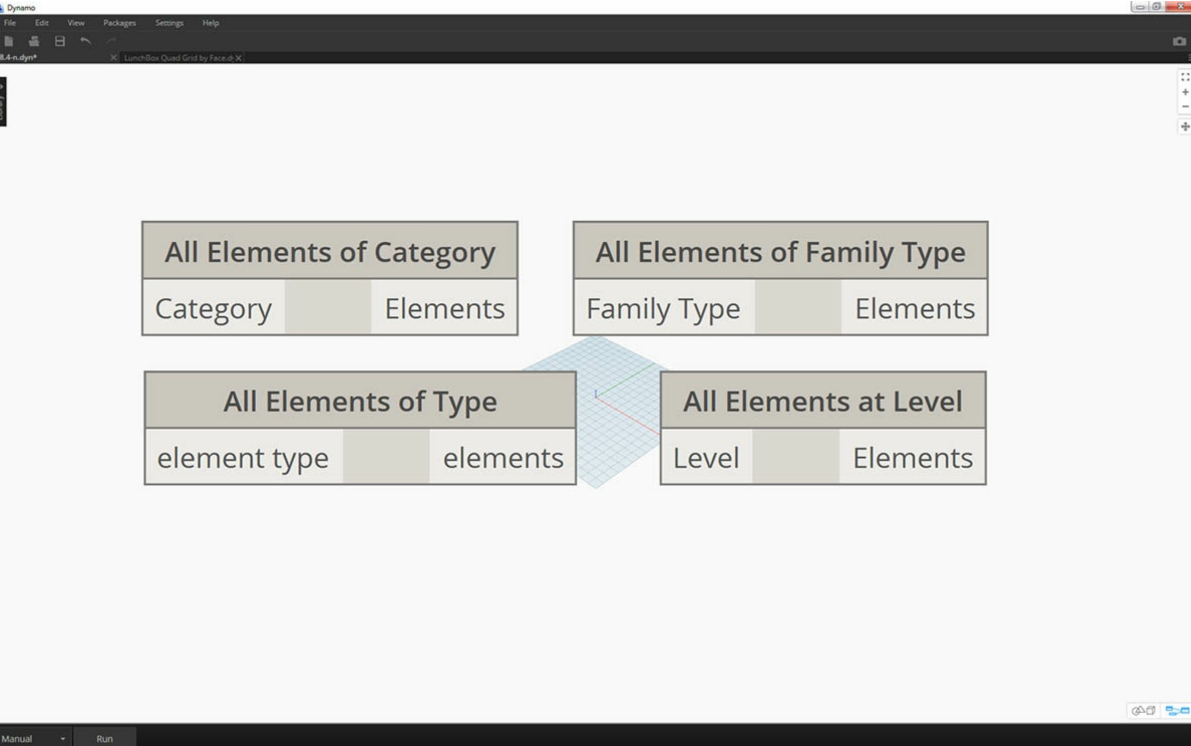
На трех изображениях ниже представлены основные категории выбора элементов Revit в Dynamo. Эти инструменты прекрасно сочетаются друг с другом, в чем вы убедитесь в последующих упражнениях.



Щелчок кнопкой мыши — самый простой способ непосредственного выбора элементов в Revit. Таким образом можно выбрать весь элемент модели или части его топологии (например, грань или ребро). При этом сохраняется динамическая связь с объектом Revit, благодаря чему при обновлении местоположения или параметров файла Revit связанный элемент Dynamo будет обновлен на графике.



Раскрывающиеся меню отображают список всех доступных элементов в проекте Revit. Их можно использовать для задания ссылок на элементы Revit, которые могут не отображаться на виде. Это удобный инструмент для опроса существующих элементов или создания новых в проекте Revit или редакторе семейств.



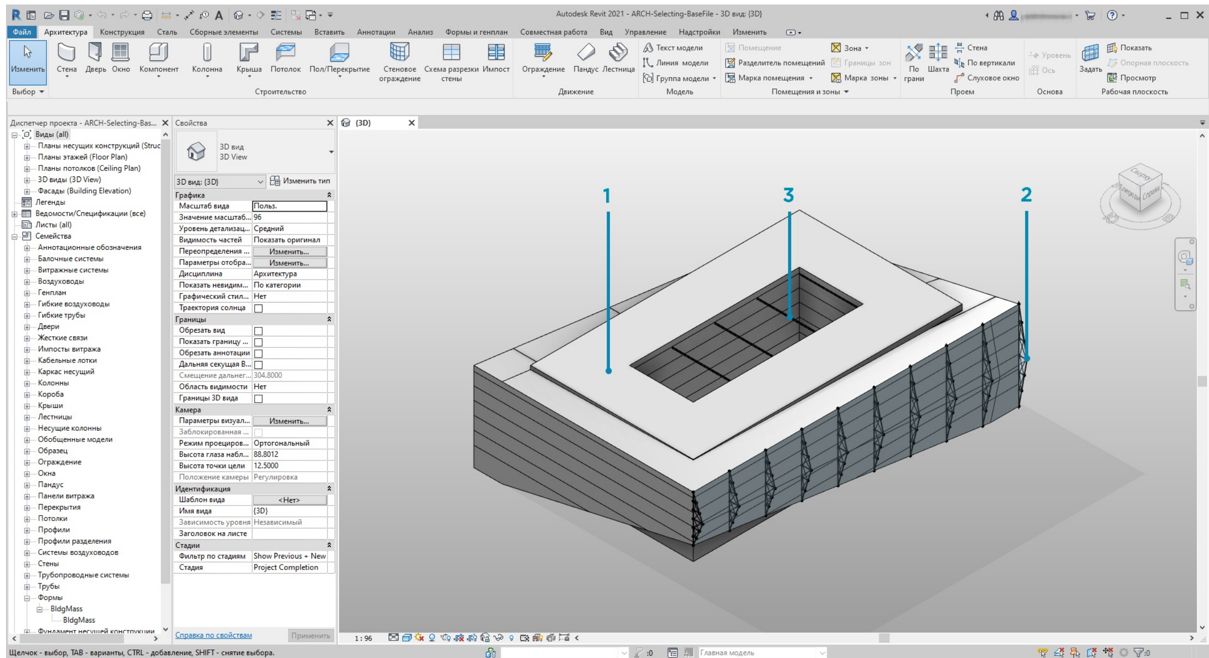
Кроме того, элементы Revit можно выбирать по определенному уровню *иерархии Revit*. Это удобно при адаптации крупных массивов данных для подготовки документации или генеративного создания экземпляров и адаптации.

Теперь, когда вы ознакомились с приведенными выше изображениями, перейдите к упражнению для выбора элементов базового проекта Revit, чтобы подготовиться к параметрическим операциям, которые вам предстоит создать в последующих разделах главы.

Упражнение

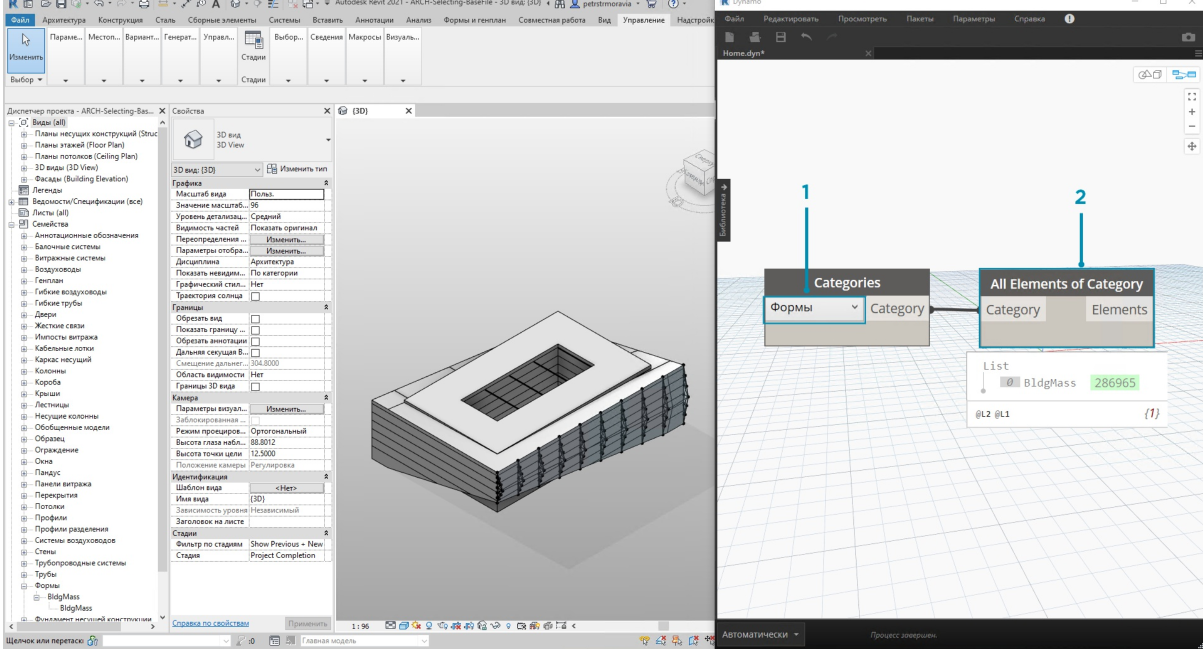
Скачайте файлы примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении.

1. [Selecting.dyn]
2. [ARCH-Selecing-BaseFile.rvt]



В этом примере файл Revit содержит три типа элементов стандартного здания. Мы используем его в качестве примера для выбора элементов Revit в контексте иерархии Revit.

1. Формообразующий элемент здания
2. Фермы (адаптивные компоненты)
3. Балки (несущий каркас)

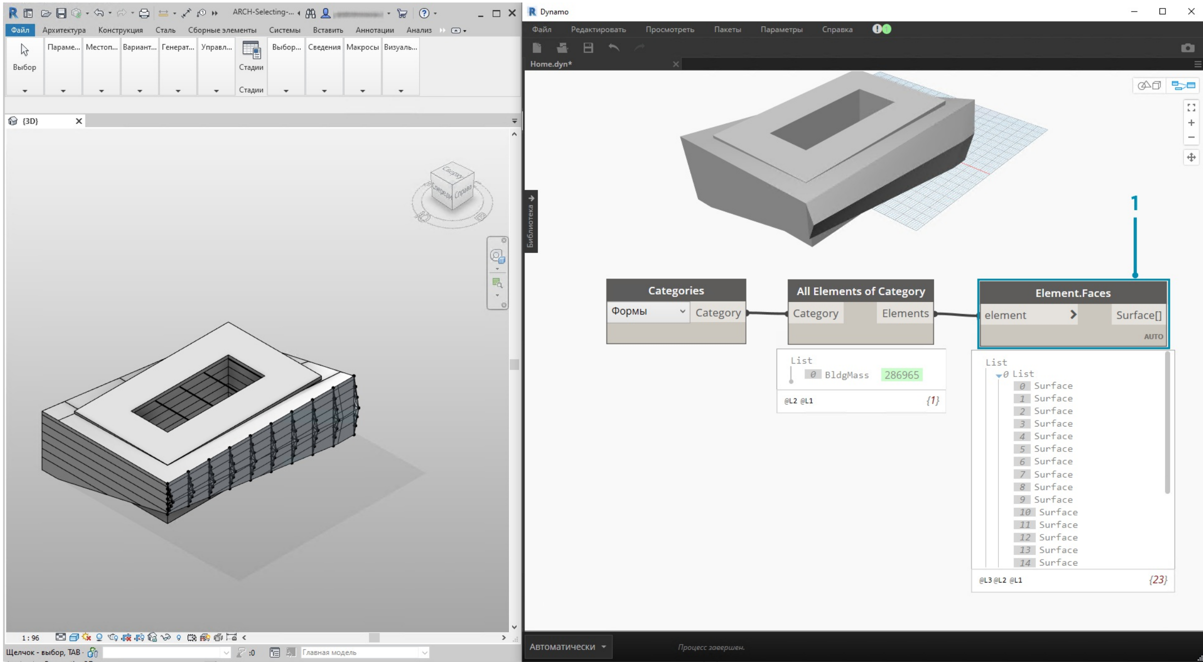


Какие выводы можно сделать на основе элементов, отображаемых на виде проекта Revit? Как глубоко в иерархии находятся соответствующие элементы? Чем масштабнее проект, тем сложнее найти ответы на подобные вопросы. Доступно множество вариантов: элементы можно выбирать по категориям, уровням, семействам, экземплярам и т. д.

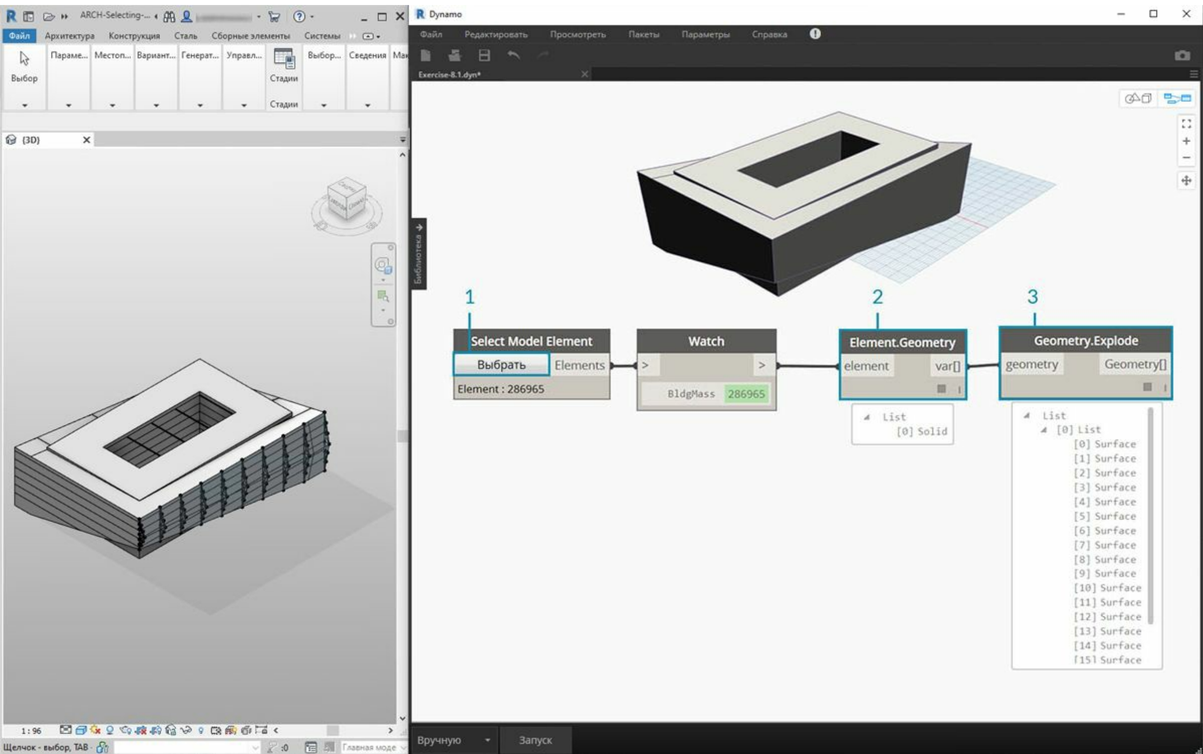
1. Так как мы используем базовую настройку, выберите формообразующий элемент здания, щелкнув *Mass* в раскрывающемся меню узла Categories. Эта функция также доступна на вкладке Revit > «Выбор».
2. На выходе узла категории Mass мы получаем только саму категорию. Необходимо выбрать элементы. Для этого используйте узел *All Elements of Category*.

Обратите внимание, что на этом этапе в Dynamo геометрия не отображается. Вы выбрали элемент Revit, но еще не преобразовали его в геометрию Dynamo. Важно различать эти операции. Когда требуется выбрать большое количество элементов, нежелательно отображать их все в Dynamo, так как это замедлит работу программы. Dynamo — это инструмент для управления проектом Revit без обязательного выполнения операций с геометрией. Мы остановимся на этом подробнее в следующем разделе главы.

В данном случае вы работаете с простой геометрией, поэтому ее вполне можно добавить в область предварительного просмотра Dynamo. Рядом с элементом BldgMass узла Watch отображается зеленое число*. Это идентификатор элемента, который позволяет понять, что вы работаете с элементом Revit, а не геометрией Dynamo. Теперь необходимо преобразовать этот элемент Revit в геометрию Dynamo.

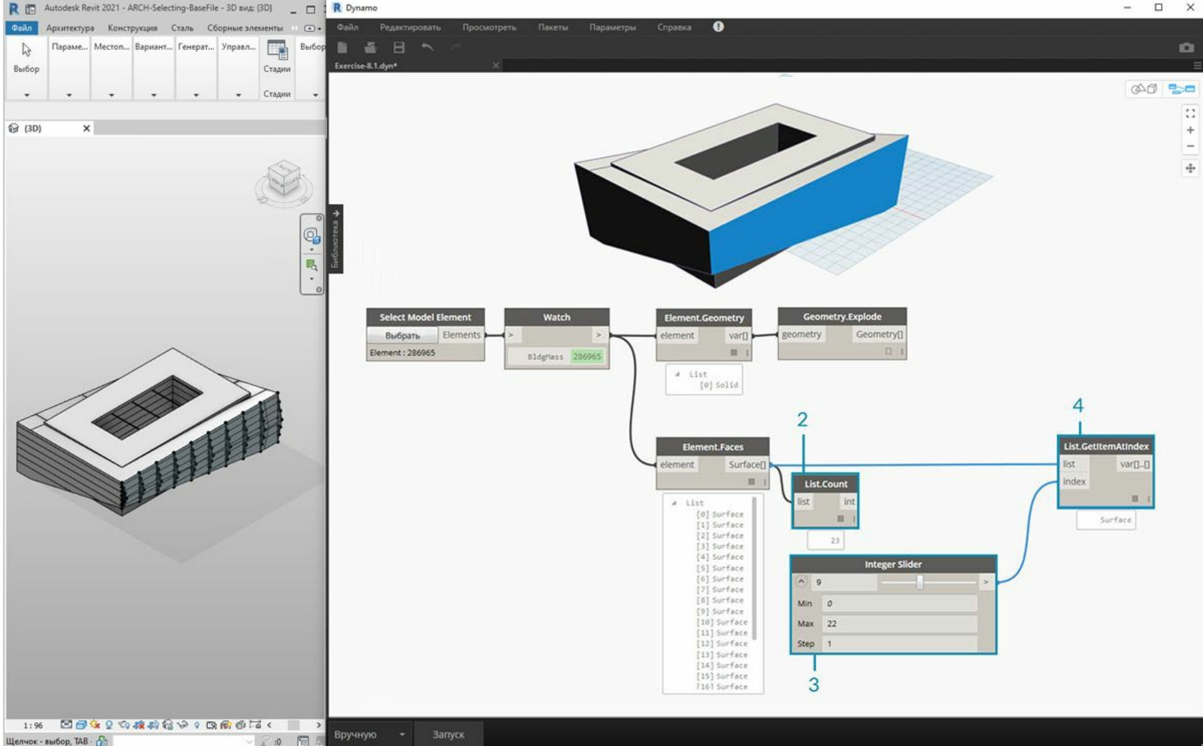


1. С помощью узла *Element. Faces* мы получаем список поверхностей, представляющий каждую грань формообразующего элемента. Теперь можно просмотреть геометрию на видовом экране Dynamo и использовать грань как опорный элемент для параметрических операций.

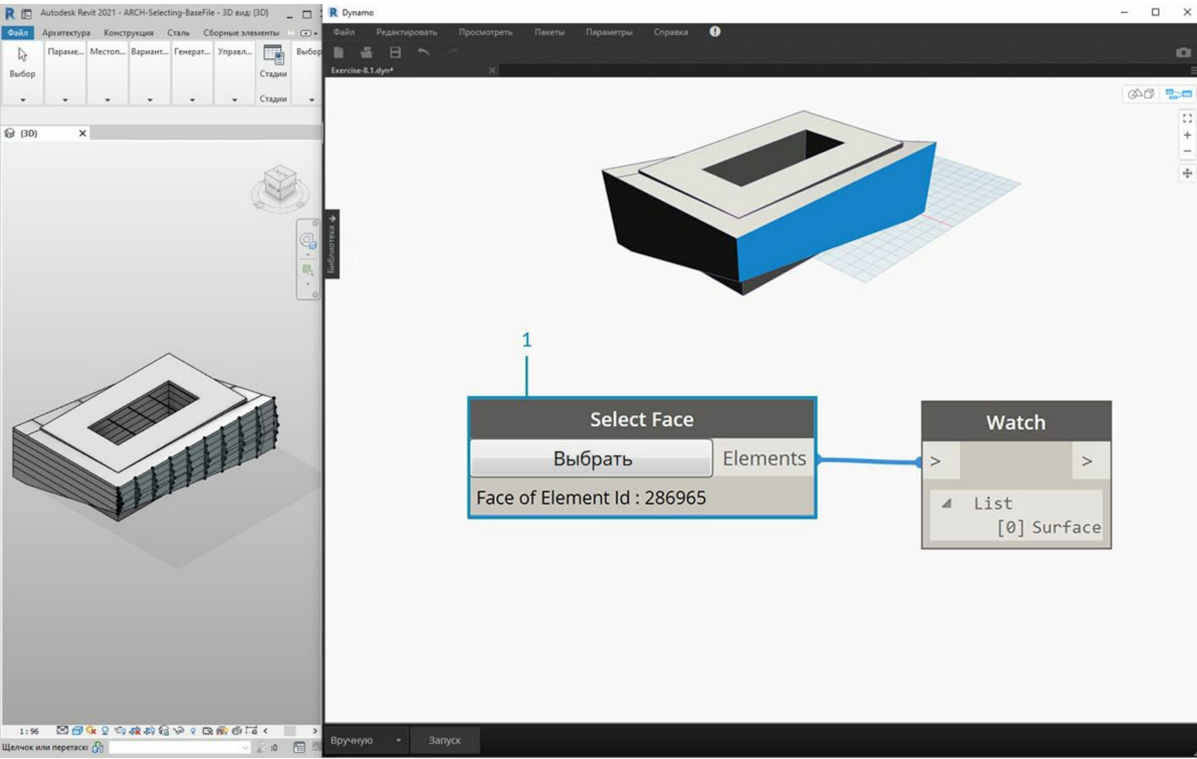


Есть и другой способ. Вместо выбора с помощью иерархии Revit *(узел All Elements of Category)* можно выбрать геометрию непосредственно в Revit.

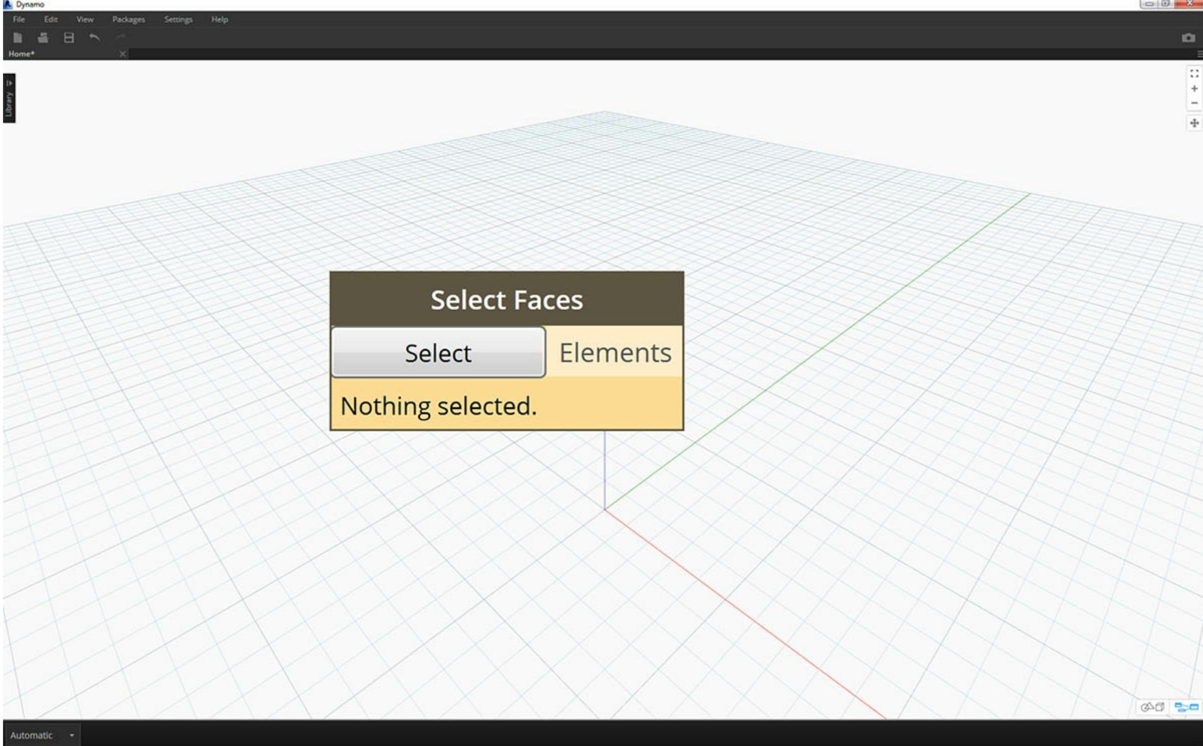
1. В узле *Select Model Element* щелкните кнопку *«Выбрать»* (или *Изменить*). На видовом экране Revit выберите нужный элемент. В данном случае следует выбрать формообразующий элемент здания.
2. Вместо выбора с помощью узла *Element.Faces* можно выбрать весь формообразующий элемент как единое геометрическое тело, применив *Element.Geometry*. При этом будет выбрана вся геометрия в пределах формообразующего элемента.
3. С помощью *Geometry.Explode* можно снова сформировать список поверхностей. Эти два узла работают аналогично *Element.Faces*, но содержат дополнительные параметры для изучения геометрии элемента Revit.



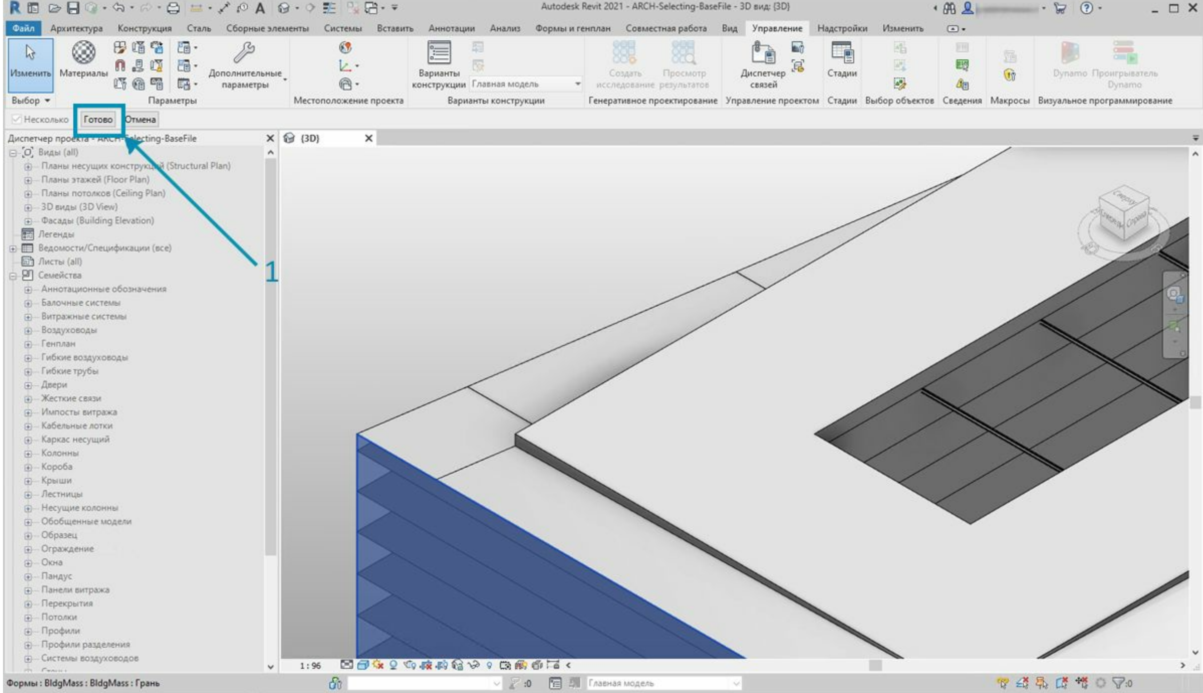
1. Используя базовые операции для списков, можно опросить нужную грань.
2. Узел *List.Count* показывает, что в пределах формообразующего элемента присутствует 23 поверхности.
3. Учитывая эти сведения, измените максимальное значение узла *Integer Slider* на *22*.
4. С помощью узла *List.GetItemAtIndex* мы используем списки в качестве входных данных и соединяем *Integer Slider* с портом ввода *index*. Изменяя положение регулятора с выбранными элементами, остановитесь на *индексе 9*, когда будет изолирован главный фасад с фермами.



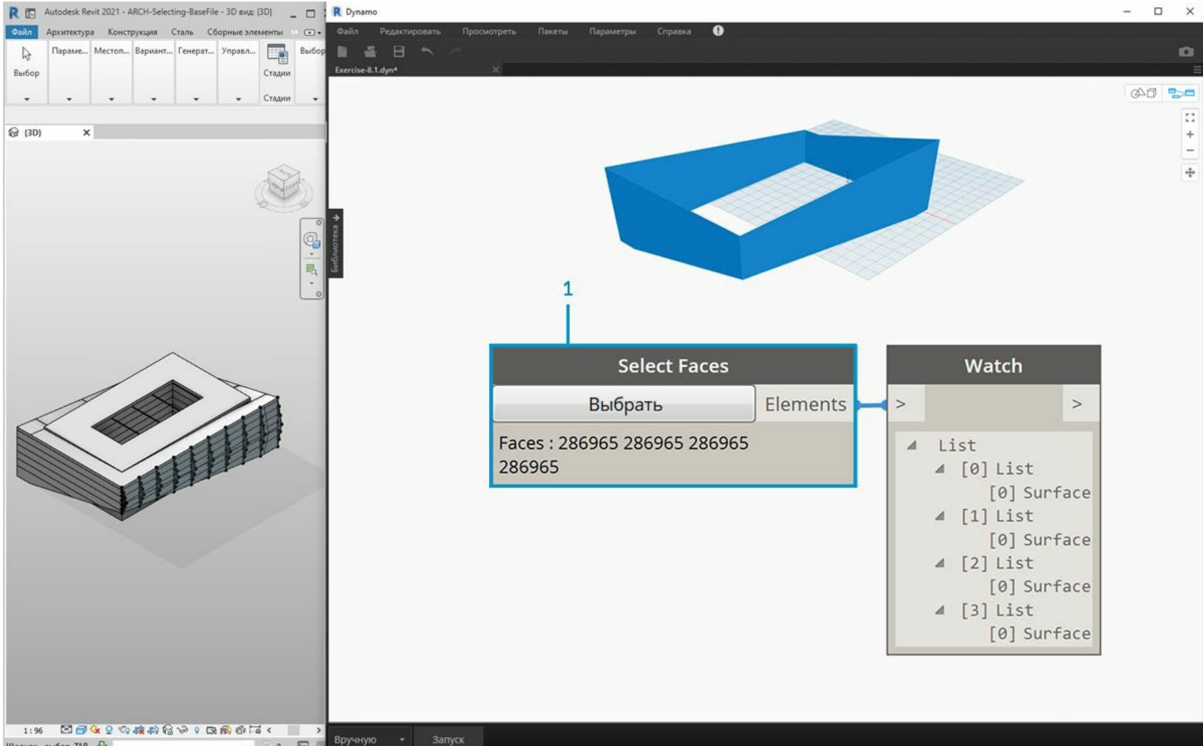
1. Предыдущий шаг был довольно трудоемким. Его можно выполнить гораздо проще и быстрее с помощью узла *Select Face*. Он позволяет изолировать в проекте Revit грань, которая не является самостоятельным элементом. Это же действие можно выполнить с помощью *Select Model Element*, выбрав поверхность вместо целого элемента.



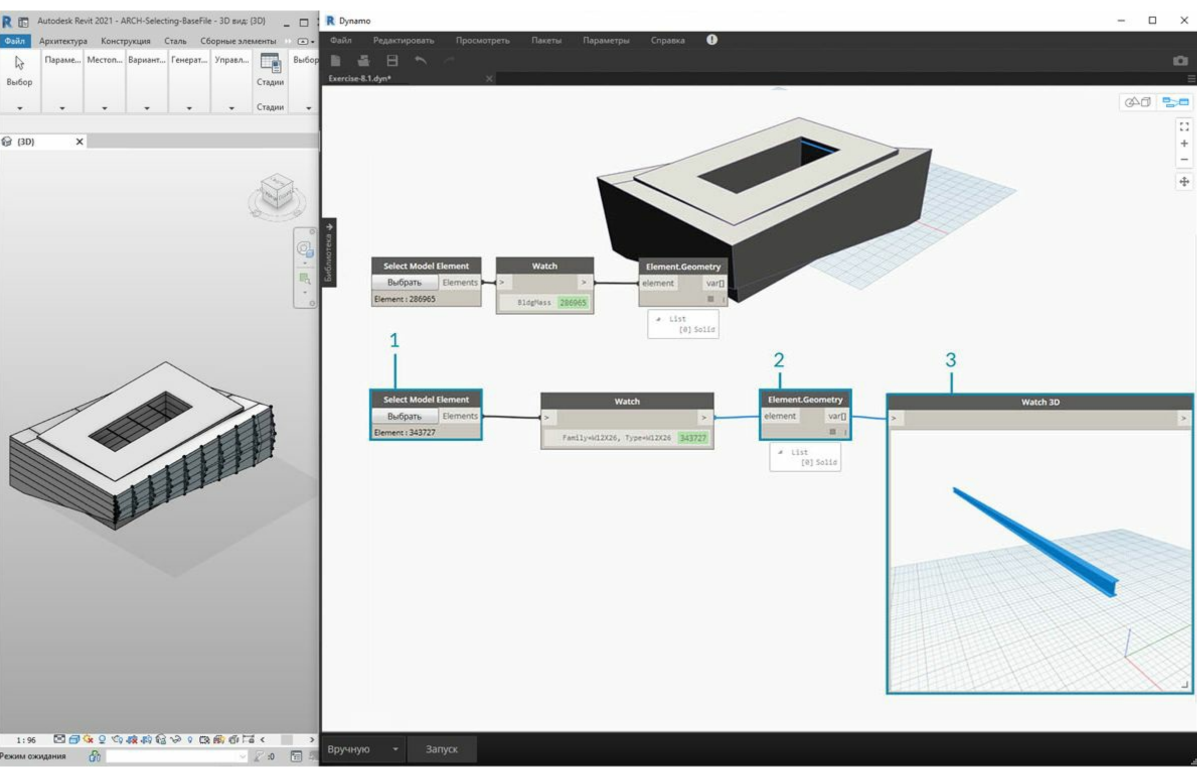
Предположим, вам нужно изолировать стены главного фасада здания. Для этого можно использовать узел *Select Faces*. Нажмите кнопку выбора, а затем выберите четыре основных фасада в Revit.



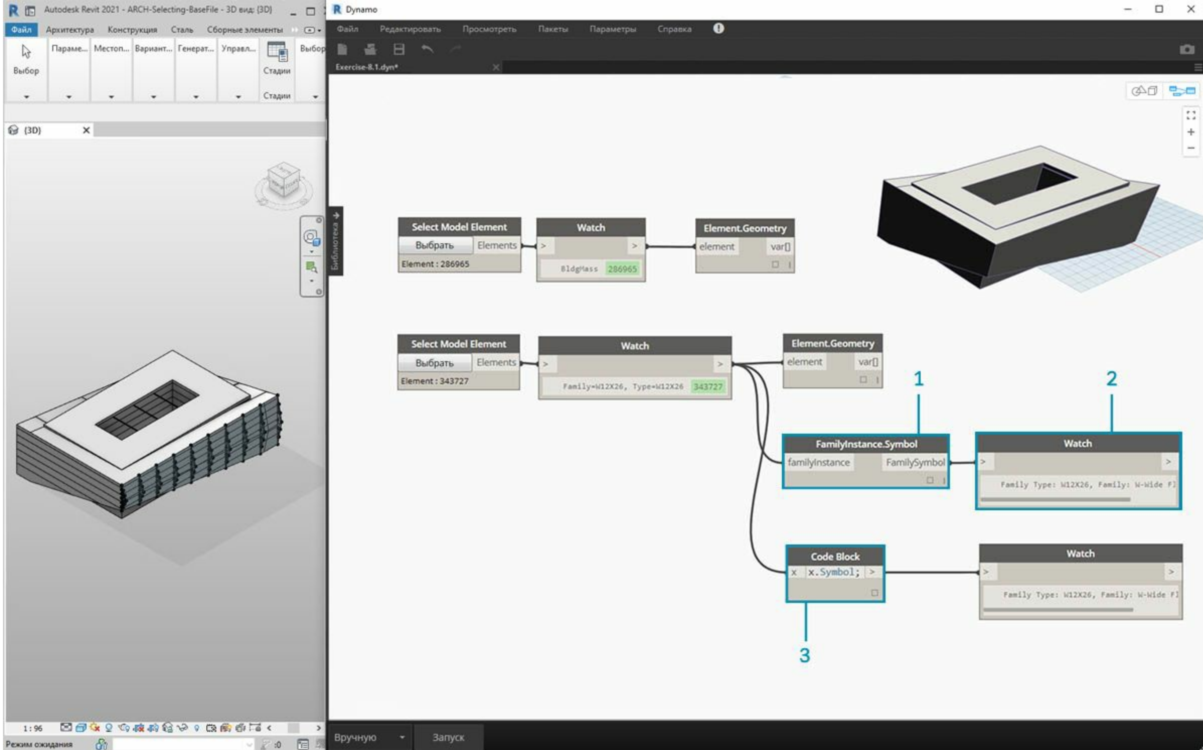
1. Выбрав четыре стены, нажмите в Revit кнопку *Готово*.



1. Грани были импортированы в Dynamo в качестве поверхностей.



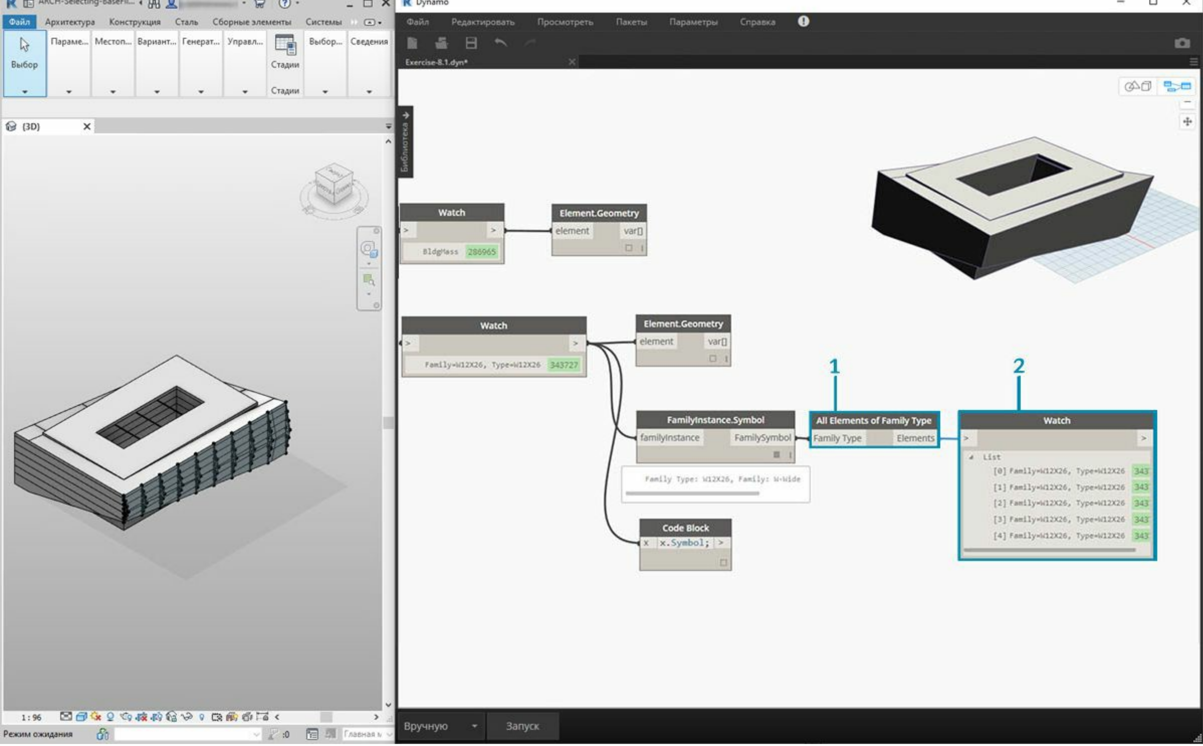
1. Теперь рассмотрим балки над атриумом. С помощью узла *Select Model Element* выберите одну из балок.
2. Соедините элемент балки с портом ввода узла *Element.Geometry*, после чего балка появится на видовом экране Dynamo.
3. С помощью узла *Watch 3D* можно увеличить геометрию (если балка не отображается в Watch 3D, щелкните правой кнопкой мыши и выберите «Вписать»).



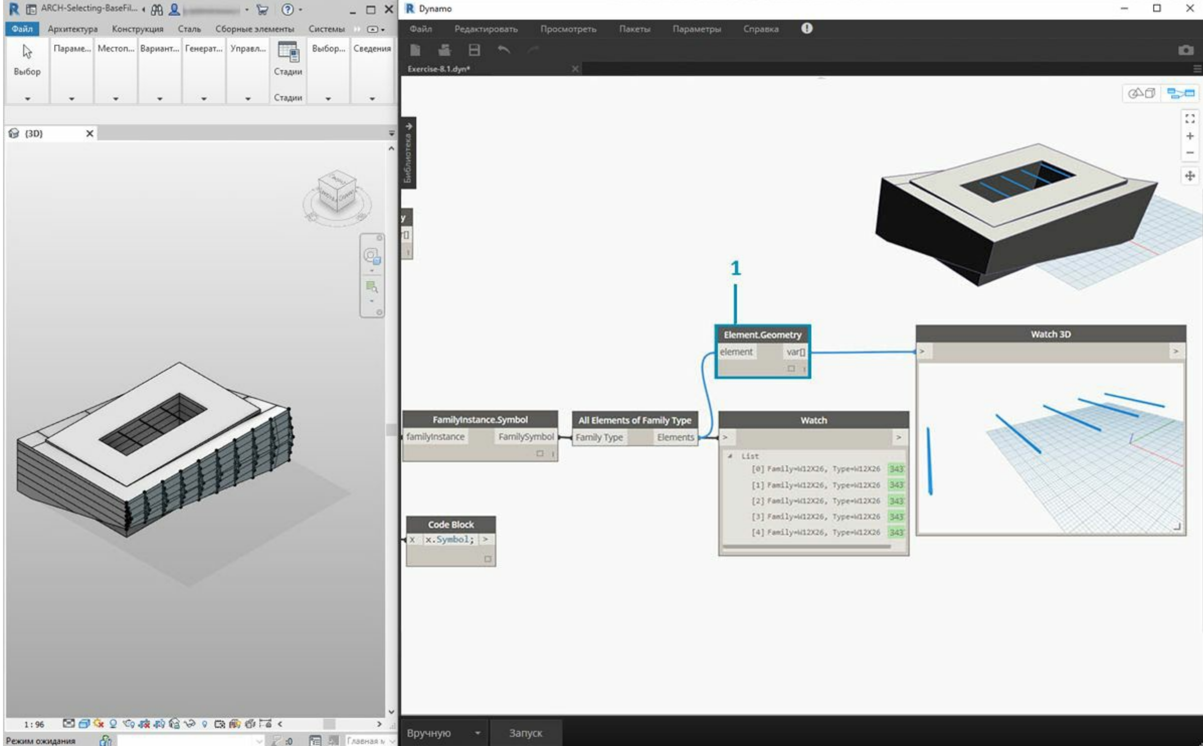
При работе в Revit и Dynamo часто возникает вопрос, как, выбрав один элемент, выделить все аналогичные элементы. Так как выбранный элемент Revit содержит всю иерархическую информацию, можно запросить его типоразмер в семействе и выбрать все элементы данного типа.

1. Соедините элемент балки с портом ввода узла *FamilyInstance.Symbol**.
2. Изображение в узле *Watch* показывает, что выходные данные теперь являются обозначением семейства, а не элементом Revit.
3. *FamilyInstance.Symbol* — это простой запрос, который можно легко выполнить в узле Code Block с помощью синтаксиса `x.Symbol;` и получить те же результаты.

** Примечание. Обозначение семейства — это термин API-интерфейса Revit для типоразмера в семействе. Чтобы не вызывать путаницы, в следующих выпусках термин будет обновлен.*



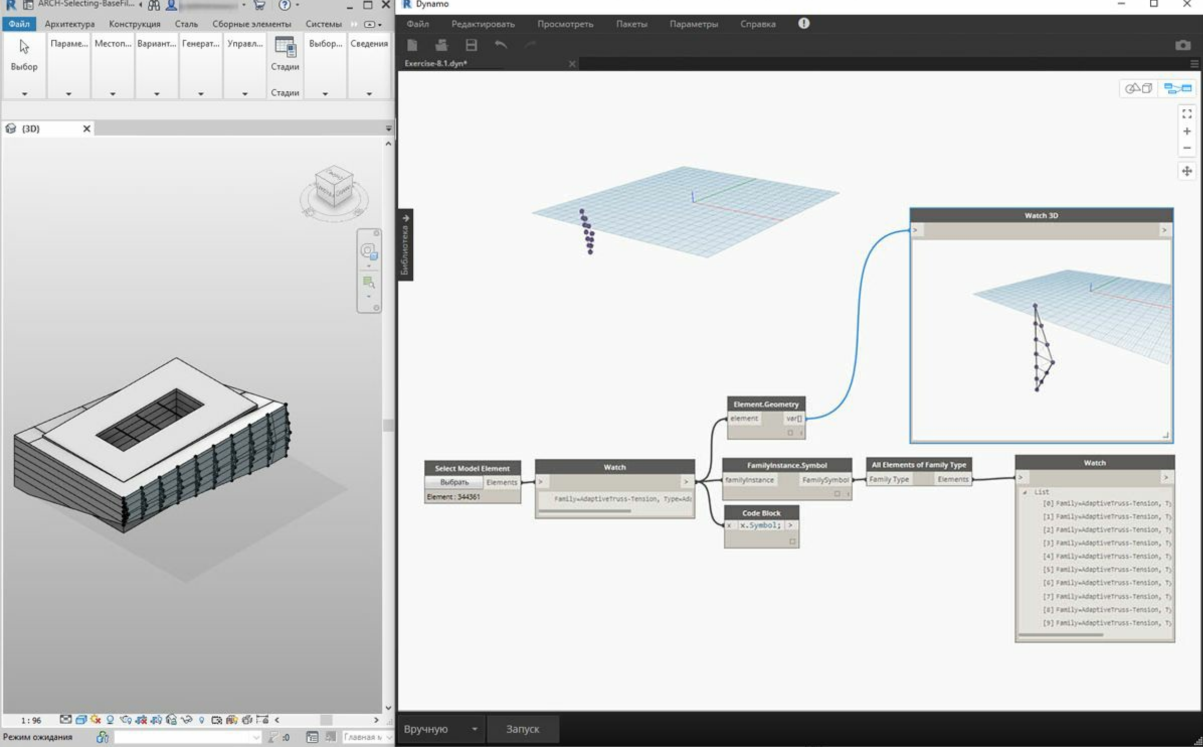
1. Для выбора остальных балок используйте узел *All Elements of Family Type*.
2. Узел Watch показывает, что выбрано пять элементов Revit.



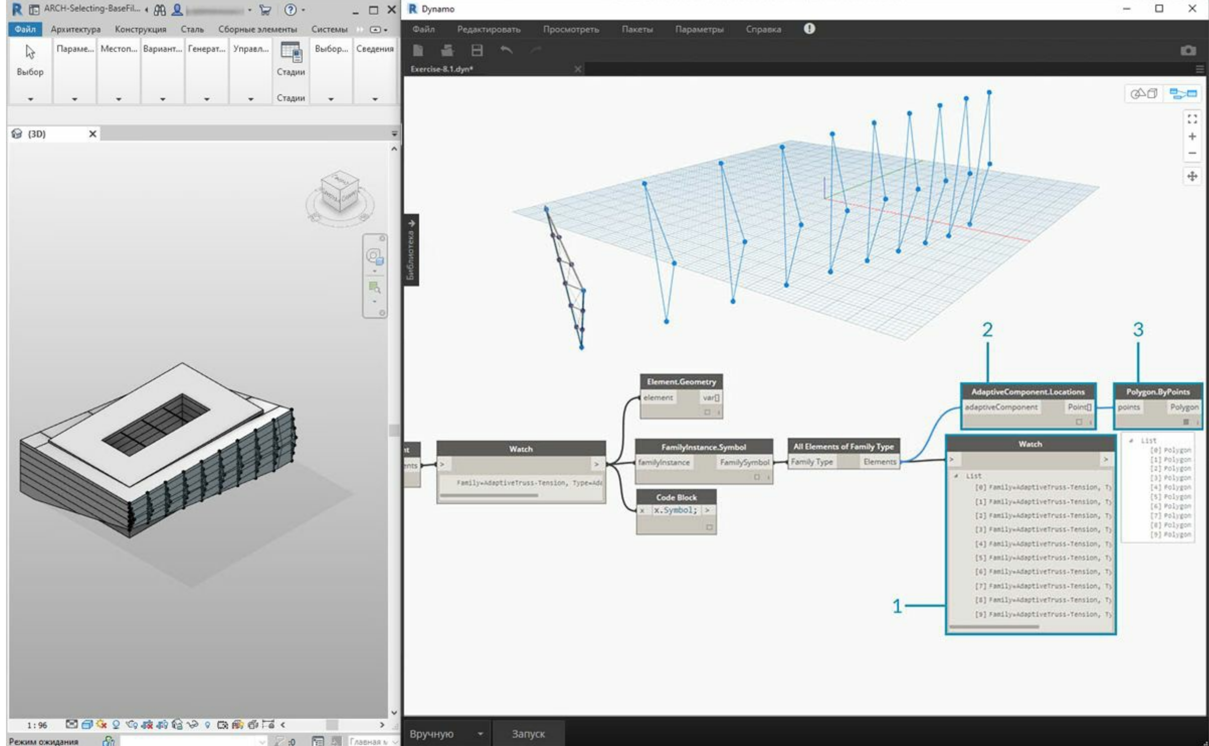
1. Эти пять элементов также можно преобразовать в геометрию Dynamo.

Что если бы в проекте было 500 балок? Преобразование всех этих элементов в геометрию Dynamo заняло бы очень много времени. Если Dynamo требуется много времени для расчета узлов, возможно, вам следует воспользоваться функцией заморозки, чтобы поставить на паузу выполнение операций Revit во время создания графика. Для получения дополнительных сведений о заморозке узлов ознакомьтесь с разделом, посвященным заморозке, в [главе о твердых телах](#).

В любом случае, даже если бы мы и хотели импортировать 500 балок, нужны ли нам все поверхности для выполнения задуманной параметрической операции? Или же мы можем извлечь основную информацию из балок и выполнить генеративные задачи с помощью фундаментальной геометрии? Подумайте над этим вопросом, пока мы продолжаем разбирать данную главу. Для примера рассмотрим систему ферм.



С помощью того же графика узлов выберите элемент фермы вместо балки. Перед этим удалите узел Element.Geometry, добавленный в предыдущем шаге.



1. В узле *Watch* отображается список адаптивных компонентов из Revit. Так как необходимо извлечь основную информацию, начните с адаптивных точек.
2. Соедините узел *All Elements of Family Type* с узлом *AdaptiveComponent.Location*. В результате получится список списков, каждый из которых содержит три точки, представляющие местоположения адаптивных точек.
3. При присоединении узла *Polygon.ByPoints* образуется сложная кривая. Она отображается на видовом экране Dynamo. Благодаря этому методу вы визуализировали геометрию одного элемента и абстрагировали геометрию оставшегося массива элементов (которых может быть больше, чем в данном примере).

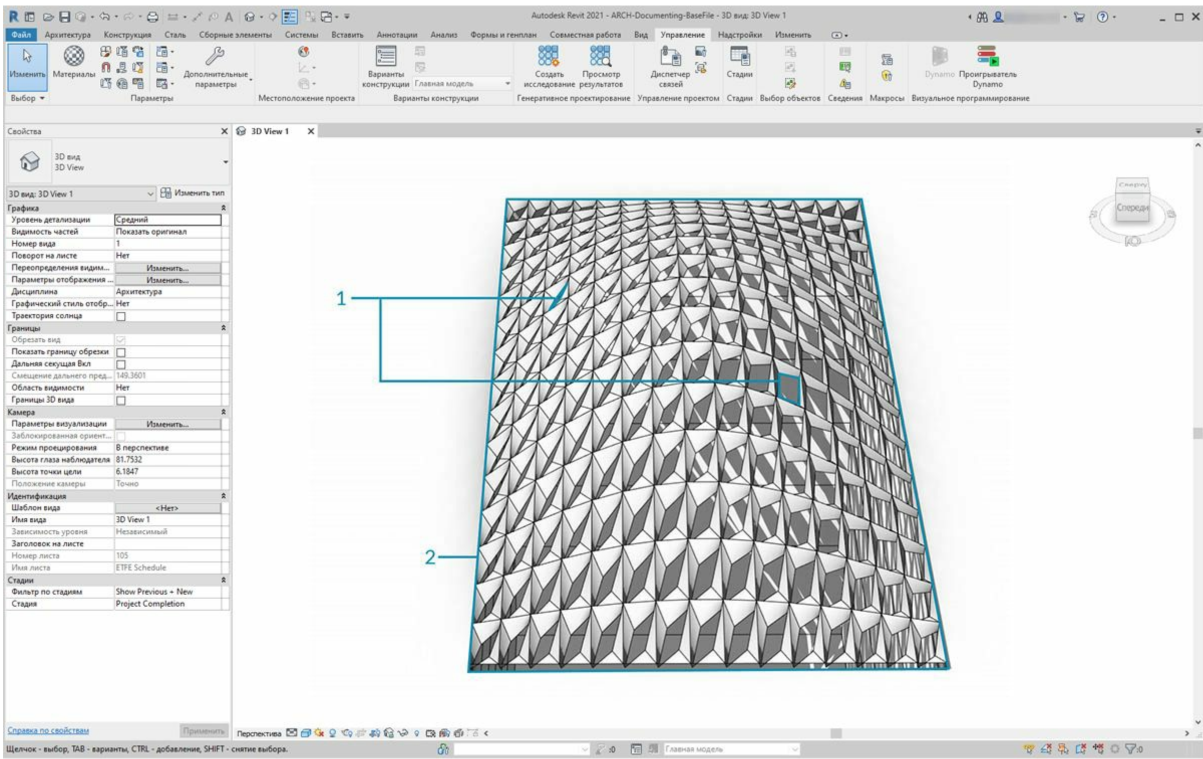
** Совет. Если щелкнуть зеленый номер элемента Revit в Dynamo, видовой экран Revit увеличит этот элемент.*

Редактирование

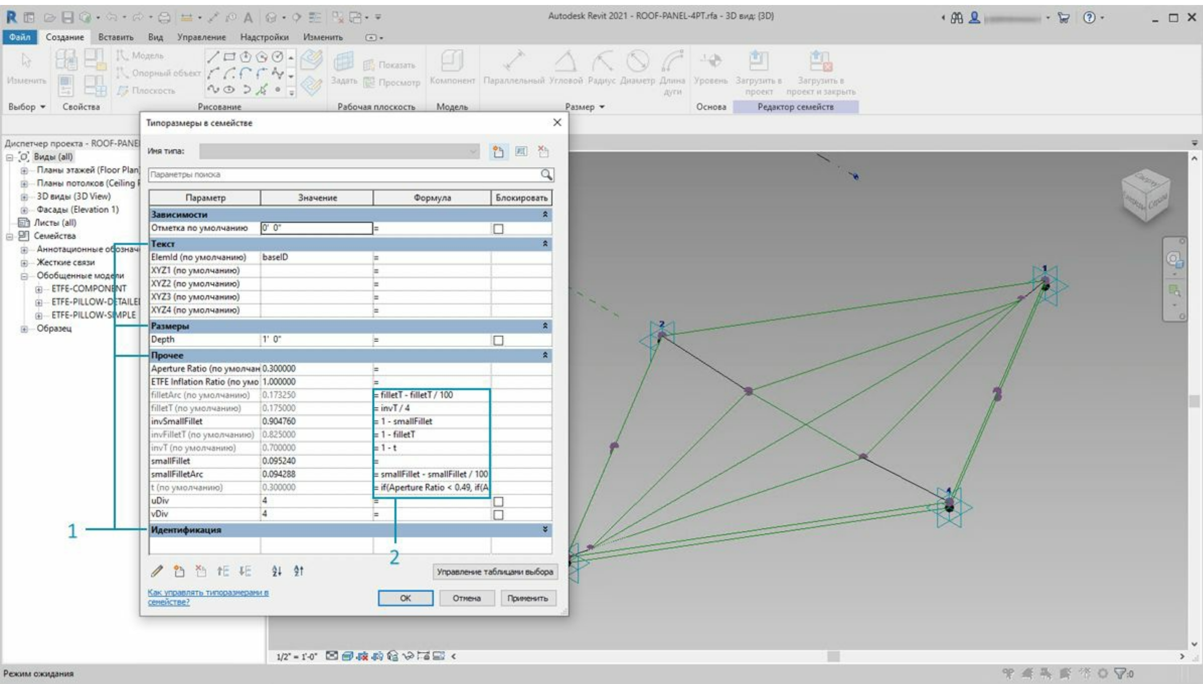
Редактирование

Одной из мощнейших возможностей Dynamo является редактирование параметров на параметрическом уровне. Например, для управления параметрами массива элементов можно использовать генеративный алгоритм или результаты моделирования. Таким образом, в проекте Revit набору экземпляров из одного семейства можно присвоить пользовательские свойства.

Параметры типов и экземпляров



1. Параметры экземпляра определяют апертуру панелей на поверхности крыши в диапазоне значений коэффициента апертуры от 0,1 до 0,4.
2. Параметры на основе типа применяются к каждому элементу на поверхности, так как они относятся к одному и тому же типоразмеру в семействе. Например, материал каждой панели может определяться параметром на основе типа.



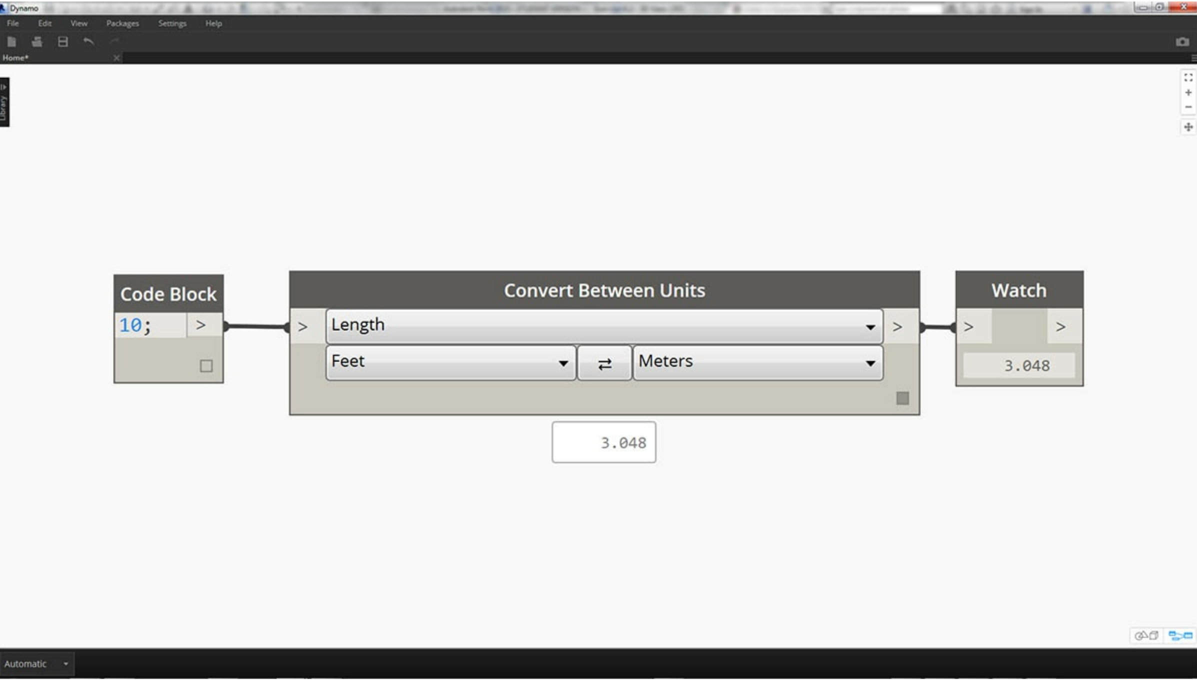
1. Если вы уже настраивали семейства Revit, то должны помнить, что необходимо назначить тип параметра (строка, номер, размер и т. д.). При назначении параметров в Dynamo убедитесь, что выбран правильный тип данных.
2. Dynamo можно также использовать в сочетании с параметрическими зависимостями, определенными в свойствах семейства Revit.

Напоминаем, что в Revit существуют параметры типа и параметры экземпляра. Их все можно редактировать в Dynamo, но в данном упражнении рассматриваются параметры экземпляра.

Примечание. По мере открытия новых возможностей, которые обеспечивают параметры редактирования, будет увеличиваться и количество элементов Revit, которые можно изменить с помощью Dynamo. Это может потребовать *дополнительных вычислительных ресурсов*, что, естественно, повлияет на скорость работы. При редактировании большого количества элементов можно воспользоваться узлом заморозки, чтобы приостановить выполнение операций Revit во время создания графика. Для получения дополнительных сведений об узлах заморозки ознакомьтесь с разделом, посвященным заморозке, в главе о твердых телах.

Единицы измерения

Начиная с версии 0.8, в Dynamo практически не используются единицы измерения. Это позволяет модулю оставаться абстрактной средой визуального программирования. Узлы Dynamo, которые взаимодействуют с размерами Revit, будут ссылаться на единицы измерения проекта Revit. Например, если в Revit с помощью Dynamo задается параметр длины, число, указанное в качестве значения в Dynamo, будет соответствовать единицам измерения по умолчанию в проекте Revit. В приведенном ниже упражнении используются метры.



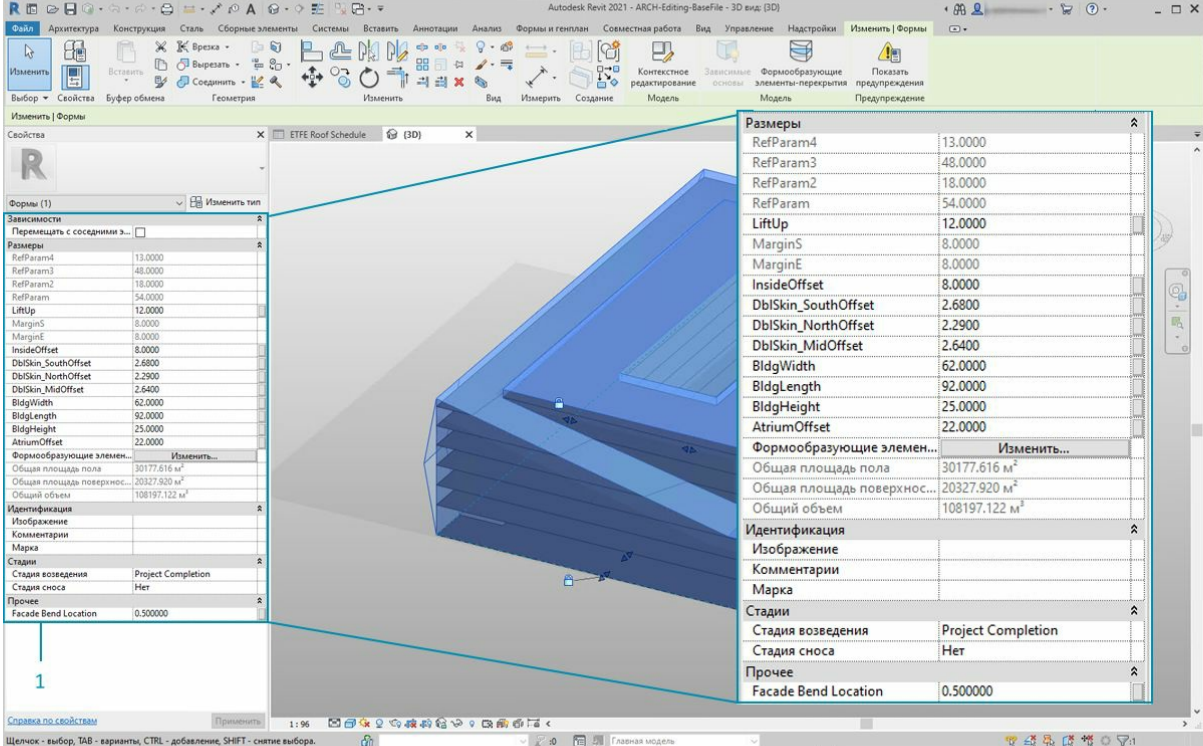
Для быстрого преобразования единиц измерения используйте узел *Convert Between Units*. Это удобный инструмент для динамического преобразования единиц измерения длины, площади и объема.

Упражнение

Скачайте файлы примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении.

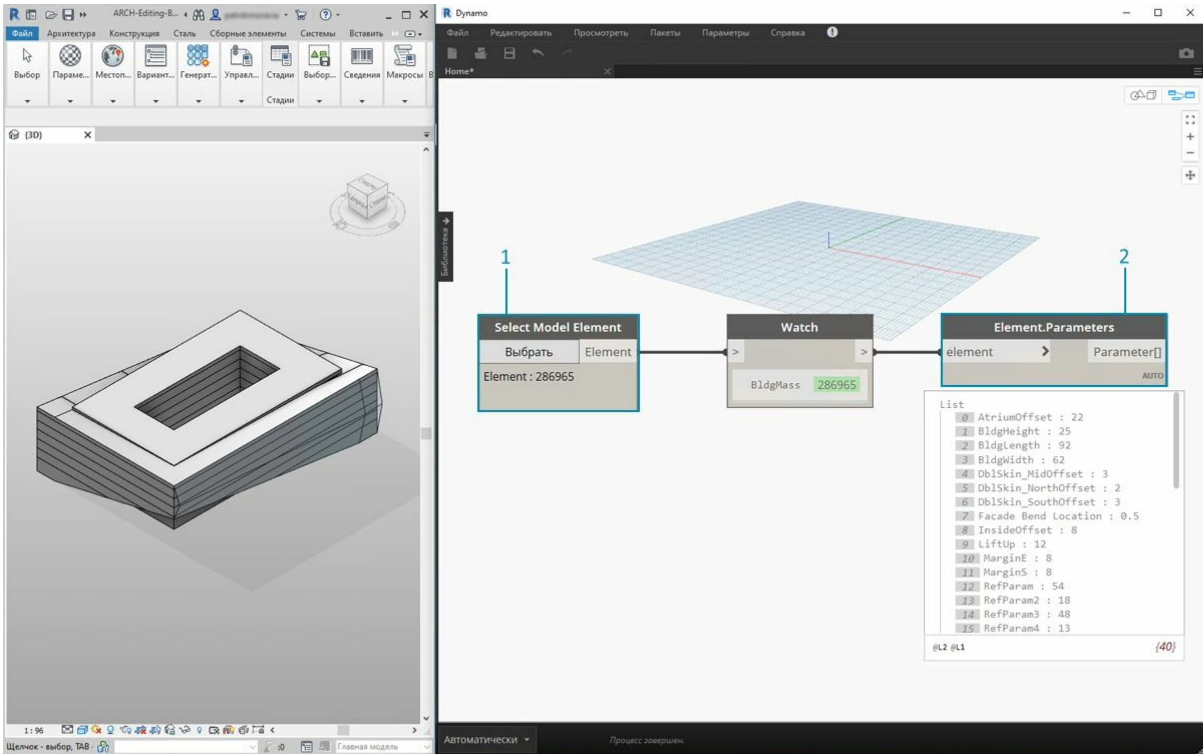
1. Editing.dyn
2. ARCH-Editing-BaseFile.rvt

В этом упражнении основное внимание уделяется редактированию элементов Revit без выполнения геометрических операций в Dynamo. В данном случае геометрия Dynamo не импортируется, а просто редактируются параметры в проекте Revit. Это упражнение базового уровня. Опытным пользователям Revit следует обратить внимание на то, что хотя речь идет о параметрах экземпляров формообразующего элемента, тот же принцип можно применить и к целому массиву элементов. Все действия выполняются с помощью узла Element.SetParameterByName.

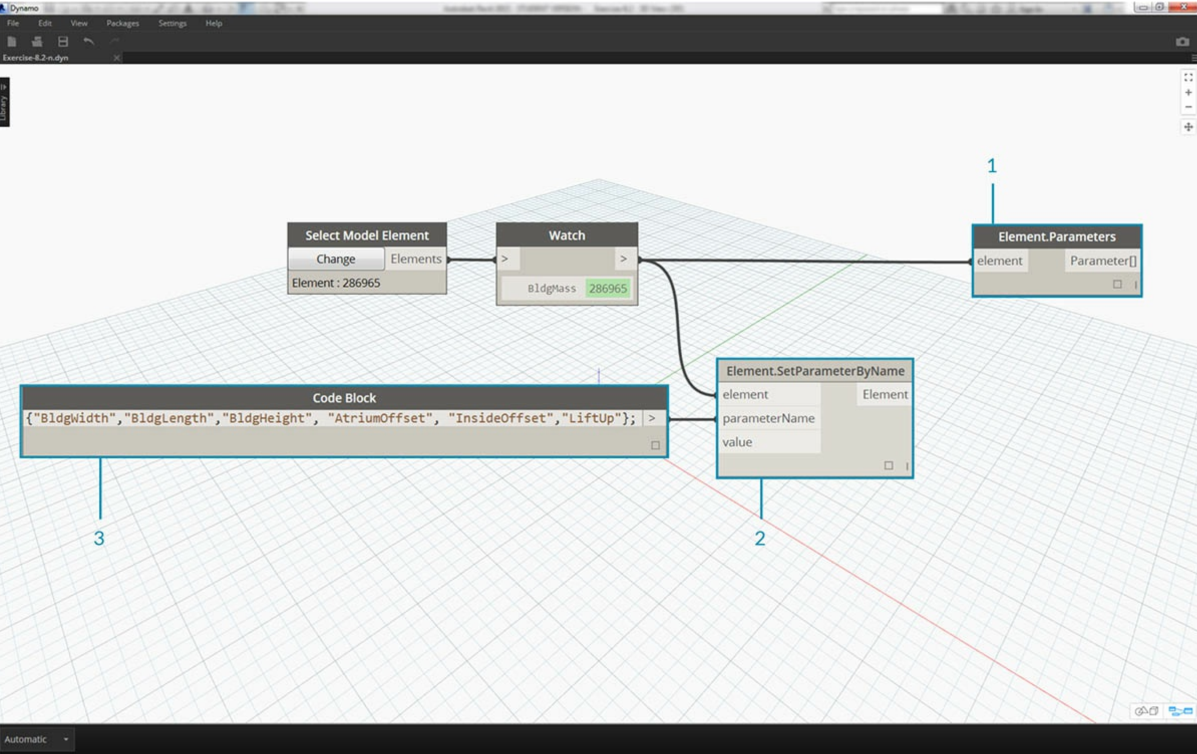


Начнем с файла Revit, используемого в этом разделе в качестве примера. Несущие элементы и адаптивные фермы из предыдущего раздела были удалены. В этом упражнении мы рассмотрим параметрическую оснастку в Revit и манипуляции в Dynamo.

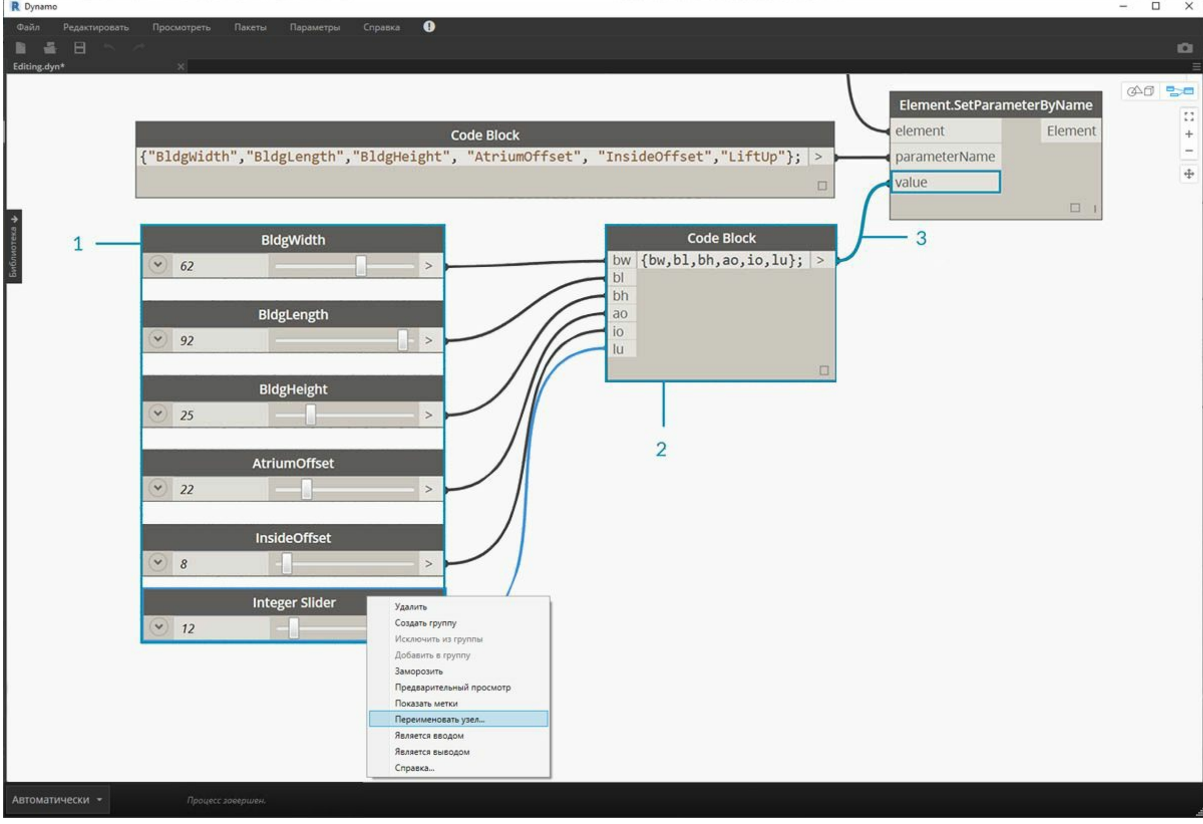
1. При выборе здания в разделе «Формообразующий элемент» в Revit на панели «Свойства» отображается массив параметров экземпляра.



1. Выберите формообразующий элемент здания с помощью узла *Select Model Element*.
2. Можно запросить все параметры этого формообразующего элемента с помощью узла *Element.Parameters*. Сюда входят параметры типа и экземпляра.



1. Для поиска нужных параметров используйте узел *Element.Parameters*. Кроме того, можно выбрать имена параметров для редактирования, изучив панель свойств из предыдущего шага. В данном случае необходимы параметры, которые влияют на большие геометрические перемещения формообразующего элемента здания.
2. Внесем изменения в элемент Revit, используя узел *Element.SetParameterByName*.
3. С помощью *блока кода* зададим список этих параметров, заключая каждый элемент в кавычки, чтобы обозначить его как строку. Можно также использовать узел List.Create с последовательностью узлов *string*, соединенных с несколькими входными параметрами. Работать с блоками кода быстрее и проще. Необходимо лишь убедиться, что строка точно соответствует имени в Revit с учетом регистра: `{"BldgWidth","BldgLength","BldgHeight", "AtriumOffset", "InsideOffset","LiftUp"};`.

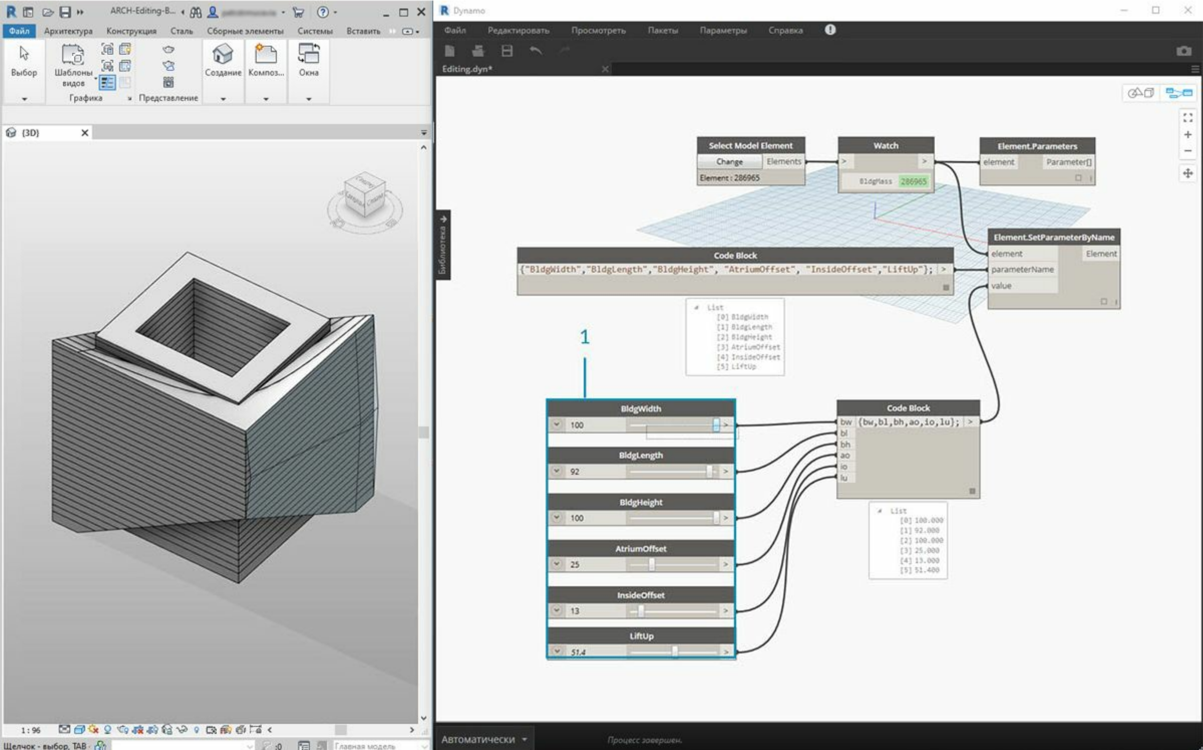


1. Кроме того, необходимо указать значения для каждого параметра. Добавьте шесть узлов *Integer Slider* в рабочую область и переименуйте их в соответствии с параметром в списке. Для каждого регулятора установите значение, показанное на изображении

выше (сверху вниз — `62,92,25,22,8,12`).

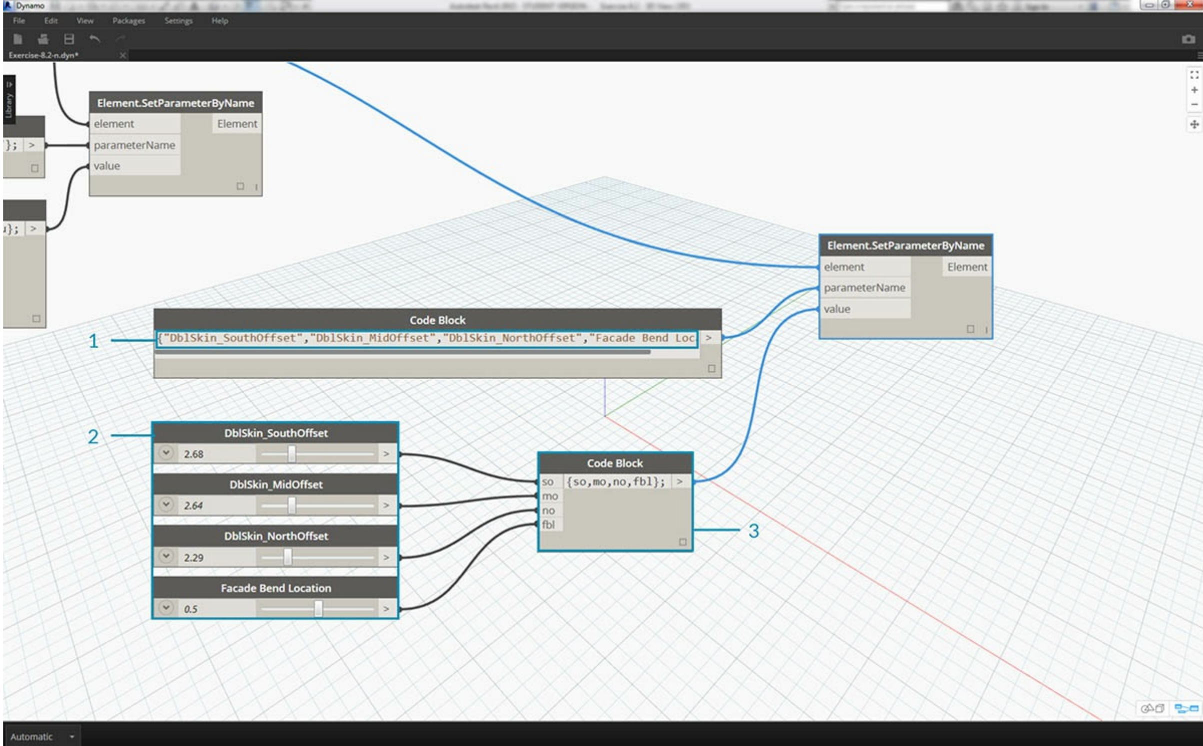
2. Создайте другой *блок кода*, содержащий список той же длины, что и имена параметров. В этом случае мы присваиваем имена переменным (без кавычек), которые создают входные данные для *блока кода*.Соедините *регуляторы* с соответствующими входными параметрами: `{bw,bl,bh,ao,io,lu};`.
3. Соедините *блок кода* с узлом *Element.SetParameterByName**. Если установлен флажок «Автоматическое выполнение процесса», результаты отобразятся автоматически.

** Примечание. В этом примере рассматриваются параметры экземпляра, а не параметры типа.*

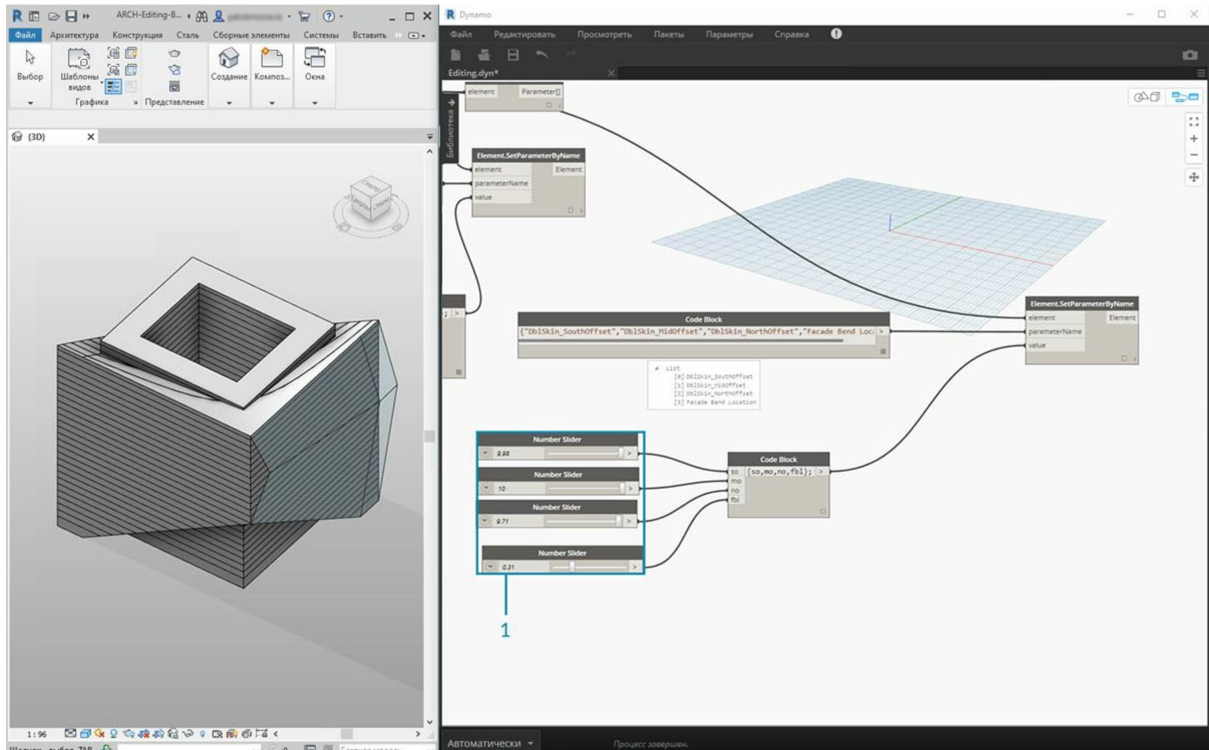


Как и в Revit, многие параметры зависят друг от друга. Существует множество комбинаций, когда геометрия может нарушаться. Эту проблему можно решить с помощью формул, заданных в свойствах параметров. Кроме того, аналогичный алгоритм можно воспроизвести с помощью математических операций в Dynamo (можно выполнить в дополнение к этому упражнению).

1. С помощью следующей комбинации можно придать оригинальный дизайн формообразующему элементу здания: `100,92,100,25,13,51.4`



1. Скопируем график и рассмотрим остекление фасада, где будет находиться ферма. В данном случае изолируем четыре параметра: `{"DblSkin_SouthOffset","DblSkin_MidOffset","DblSkin_NorthOffset","Facade Bend Location"};`.
2. Кроме того, создадим *регуляторы чисел* и присвоим им имена соответствующих параметров. Первые три регулятора (сверху вниз) необходимо перенастроить на область [0,10], а последний (*Facade Bend Location*) — на область [0,1]. Эти значения в нисходящем порядке должны начинаться со следующих чисел (хотя сами они произвольные): `2.68,2.64,2.29,0.5`.
3. Создайте новый *блок кода* и соедините регуляторы: `{so,mo,no,fbl};`.

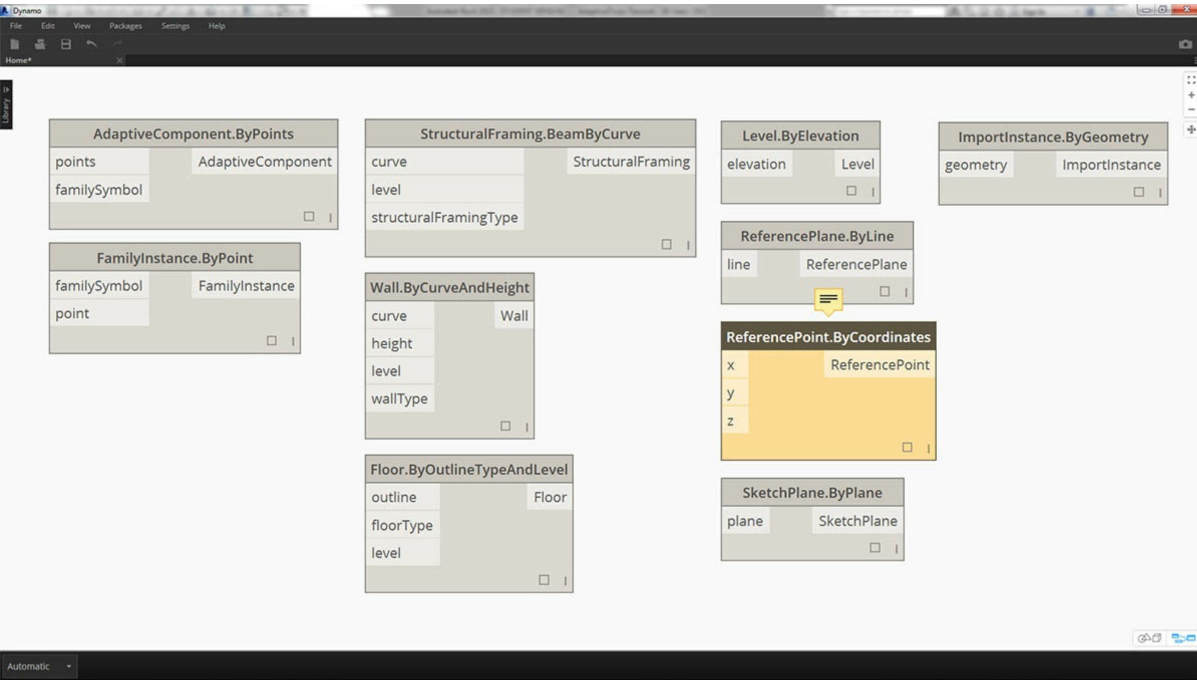


1. Изменяя *регуляторы* в этой части графика, можно сделать остекление фасада значительно более прочным: `9,98,10,0,9.71,0,31`.

Создание

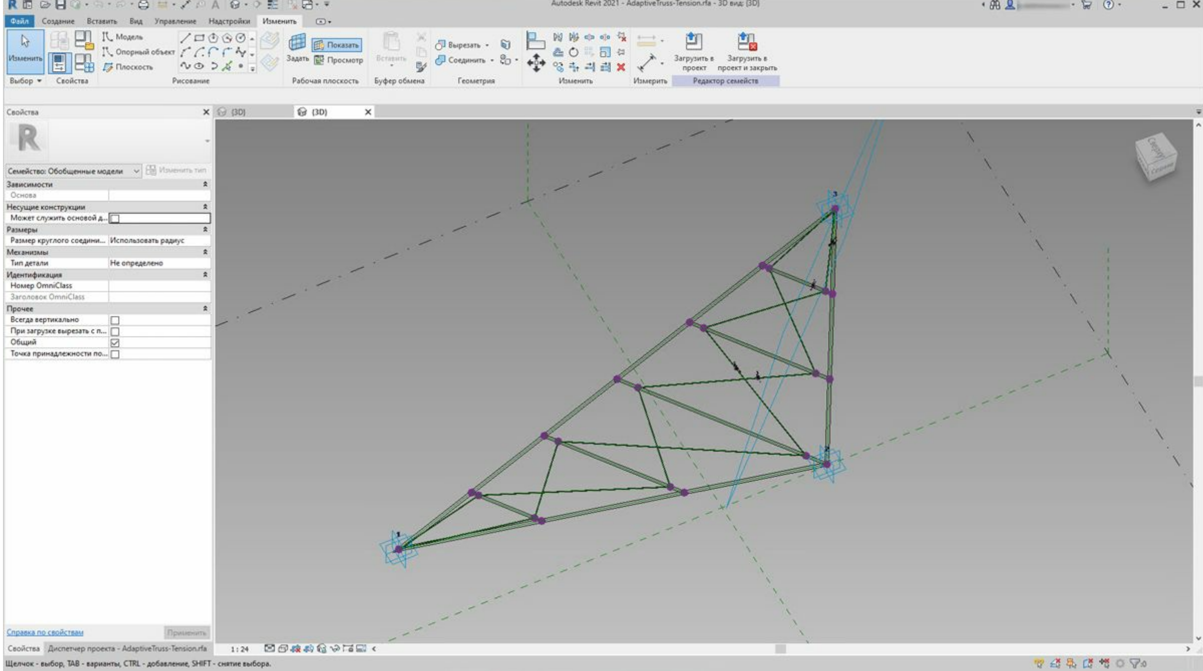
Создание

В Dynamo можно создать массив элементов Revit с полным параметрическим управлением. Узлы Revit в Dynamo позволяют импортировать элементы из типовых геометрических объектов в категории определенных типов (например, стены и перекрытия). В этом разделе рассматривается импорт параметрически гибких элементов с адаптивными компонентами.



Адаптивные компоненты

Адаптивный компонент — это гибкая категория семейства, которая хорошо подходит для генеративных приложений. После создания экземпляра можно построить сложный геометрический элемент, который определяется исходным положением адаптивных точек.



Пример адаптивного компонента на основе трех точек в редакторе семейств. Создается ферма, определяемая положением каждой адаптивной точки. В упражнении ниже с помощью этого компонента будет создана серия ферм по ширине фасада.

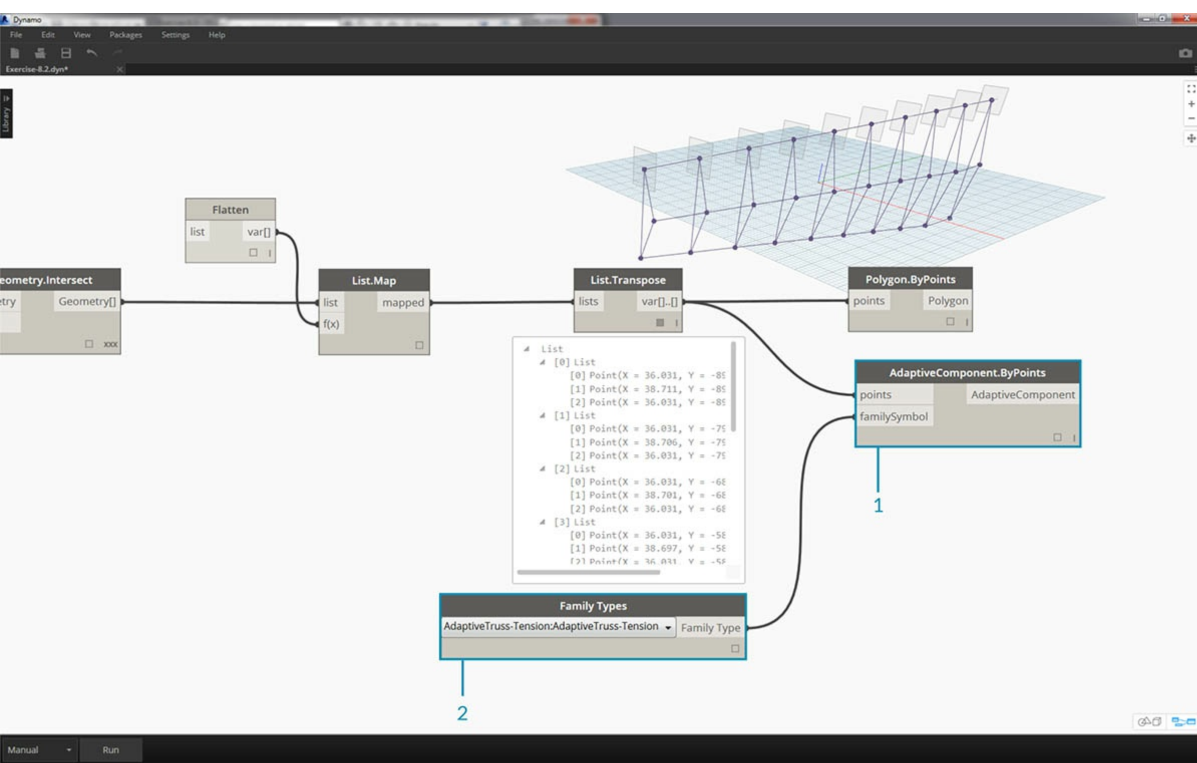
Принципы взаимодействия

Адаптивный компонент — хороший пример применения взаимодействия. Задав опорные адаптивные точки, можно создать массив адаптивных компонентов. В свою очередь, при переносе этих данных в другие программы есть возможность свести геометрию к простым данным. Примерно такая же логическая схема используется при импорте и экспорте в программе Excel.

Предположим, что консультант по фасадным работам хочет узнать местоположение элементов фермы без разбиения готовой геометрии. При подготовке к производству консультант может указать местоположение адаптивных точек для регенерации геометрии в такой программе, как Inventor.

Рабочий процесс, который будет рассмотрен в упражнении ниже, позволяет получить доступ ко всем этим данным во время создания определения формирования элементов Revit. Благодаря этому можно объединить этапы создания концепции, разработки документации и производства в единый рабочий процесс. В результате формируется более интеллектуальный и эффективный механизм взаимодействия.

Несколько элементов и списков

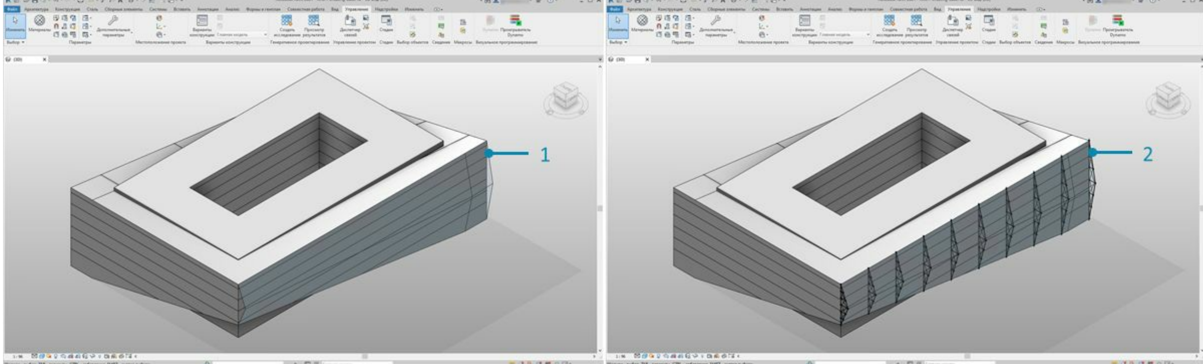


В упражнении ниже описывается, каким образом в Dynamo создаются ссылки на данные для создания элементов Revit. Для формирования нескольких адаптивных компонентов необходимо создать список списков, где в каждом списке будет три точки, соответствующие трем точкам адаптивного компонента. Это будет необходимо учитывать при управлении структурой данных в Dynamo.

Упражнение

Скачайте файлы примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении.

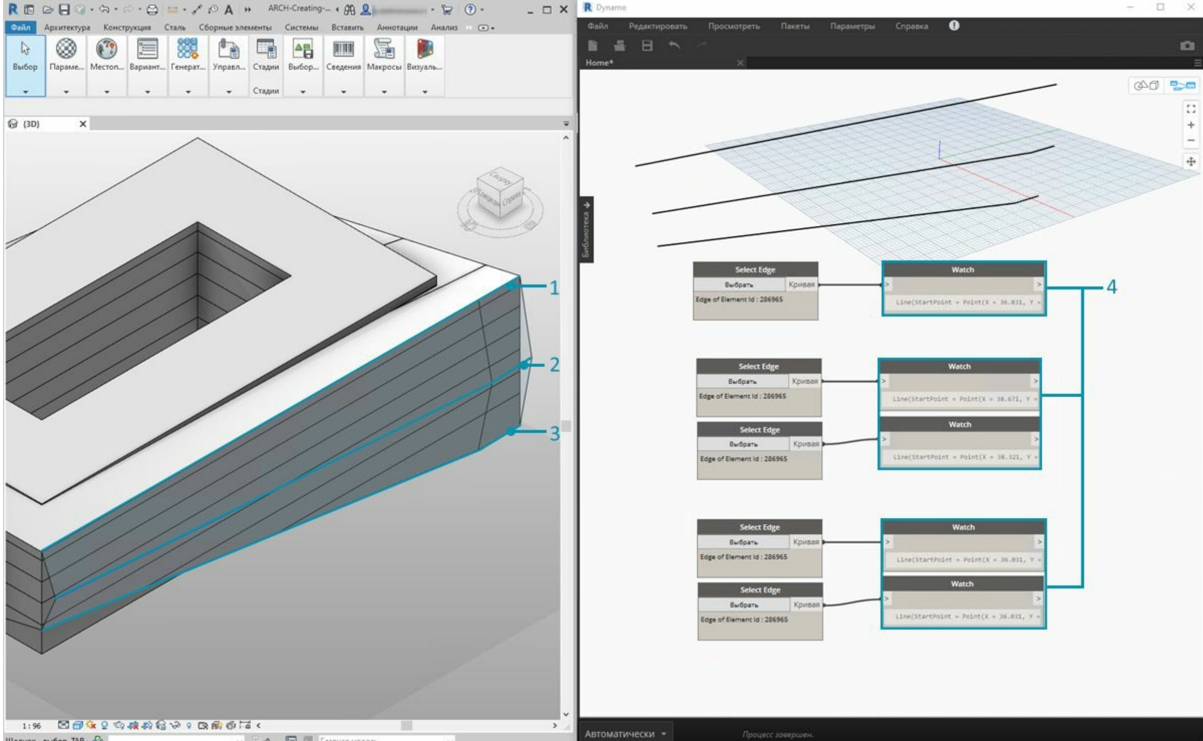
1. Creating.dyn
2. ARCH-Creating-BaseFile.rvt



Возьмите файл примера из этого раздела (или продолжите работу с файлом Revit из предыдущего сеанса). В файле есть знакомый формообразующий элемент Revit.

1. После открытия файл выглядит следующим образом.
2. А в данном случае видна система ферм, созданная при помощи Dynamo и интеллектуально связанная с формообразующим

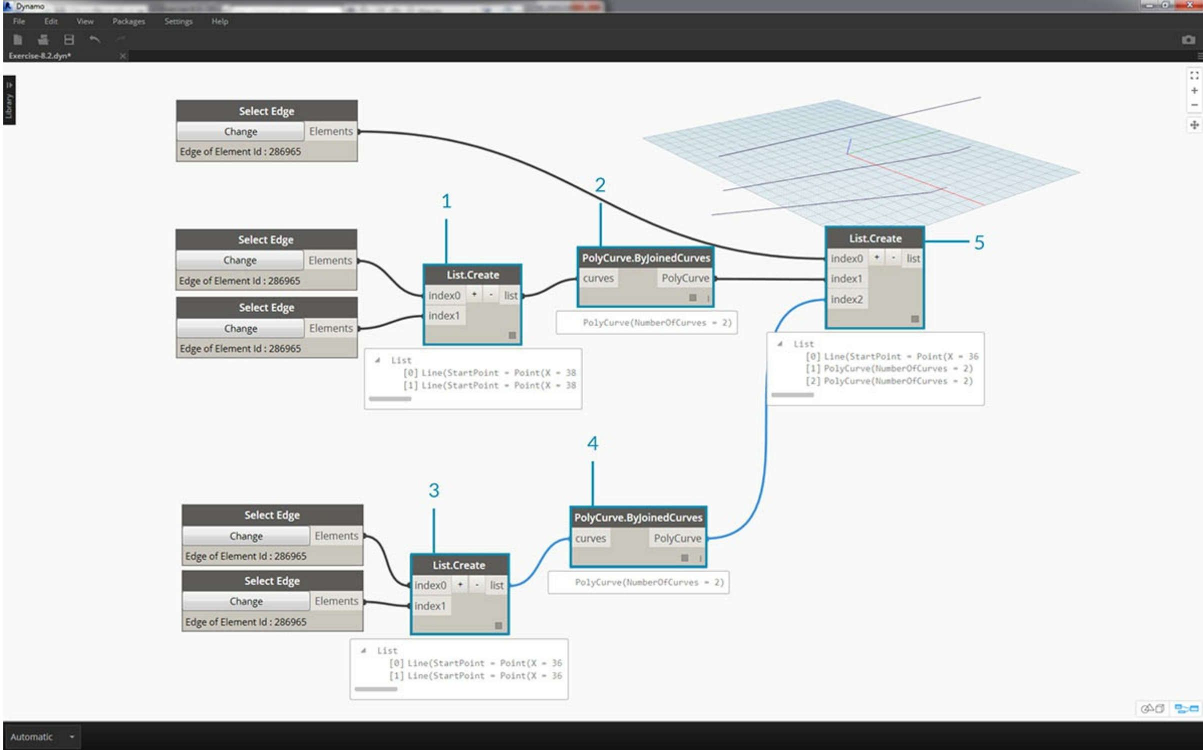
элементом Revit.



Применив узлы *Select Model Element* и *Select Face*, опустимся на одну ступень вниз по иерархии геометрии и воспользуемся узлом *Select Edge*. Если решатель Dynamo находится в *автоматическом* режиме, то при внесении изменений в файл Revit график будет постоянно обновляться. Кромка, которую необходимо выбрать, динамически привязана к топологии элементов Revit. Пока топология* остается неизменной, связь между Revit и Dynamo не прерывается.

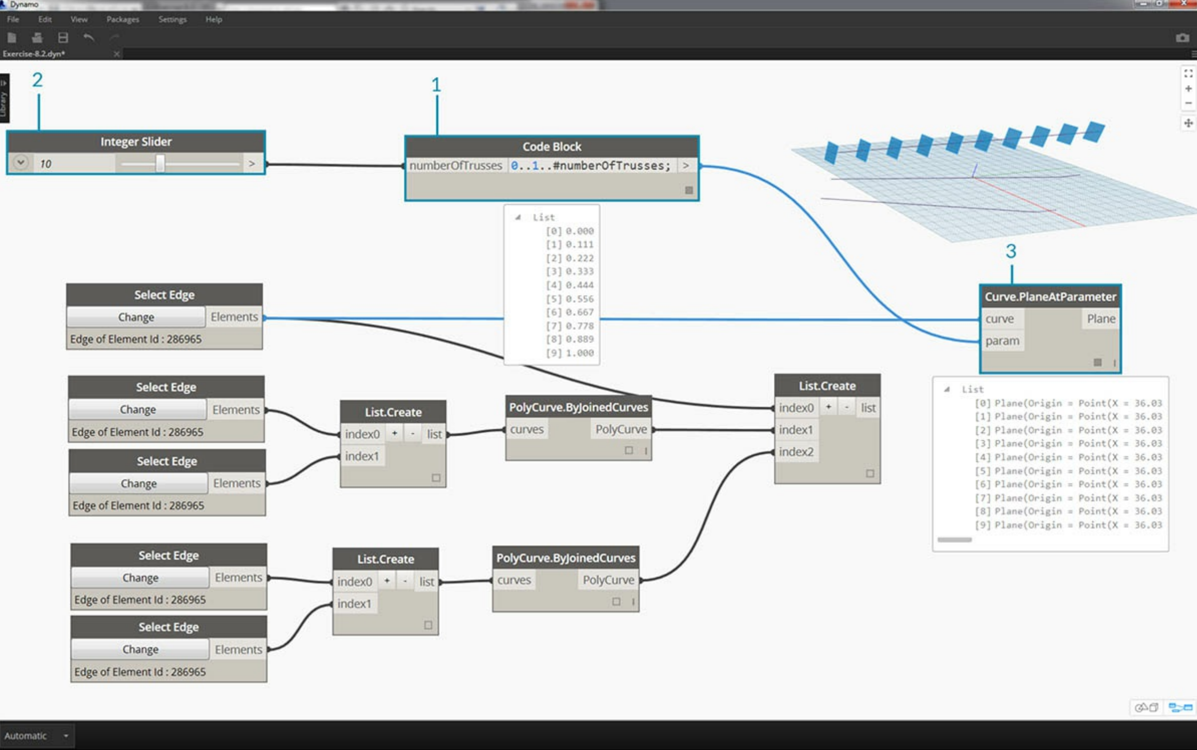
1. Выберите самую верхнюю кривую остекленного фасада. Она проходит по всей длине здания. Если выбрать кромку не удается, в Revit можно навести на нее курсор и нажимать клавишу *TAB* до тех пор, пока этот объект не будет выделен.
2. С помощью двух узлов *Select Edge* выберите кромки, представляющие скос в центре фасада.
3. Проделайте то же самое с нижними кромками фасада в Revit.
4. Узлы *Watch* теперь демонстрируют наличие линий в Dynamo. Данные автоматически преобразуются в геометрию Dynamo, так как сами кромки не являются элементами Revit. Эти кривые будут использоваться в качестве опорных элементов для создания экземпляров адаптивных ферм по ширине фасада.

** Примечание. Чтобы сохранить единообразную топологию, используется модель, в которую не были включены дополнительные грани или кромки. Параметры могут изменять ее форму, однако способ ее построения остается неизменным.*



Сначала нужно соединить кривые и объединить их в общем списке. Это позволит *«сгруппировать»* кривые для выполнения операций с геометрией.

1. Создайте список для двух кривых в центре фасада.
2. Объедините две кривые в сложную кривую, встроив компонент *List.Create* в узел *Polycurve.ByJoinedCurves*.
3. Создайте список для двух кривых в нижней части фасада.
4. Объедините эти две кривые в сложную кривую, встроив компонент *List.Create* в узел *Polycurve.ByJoinedCurves*.
5. Наконец, объедините три основные кривые (одну линию и две сложные кривые) в один список.

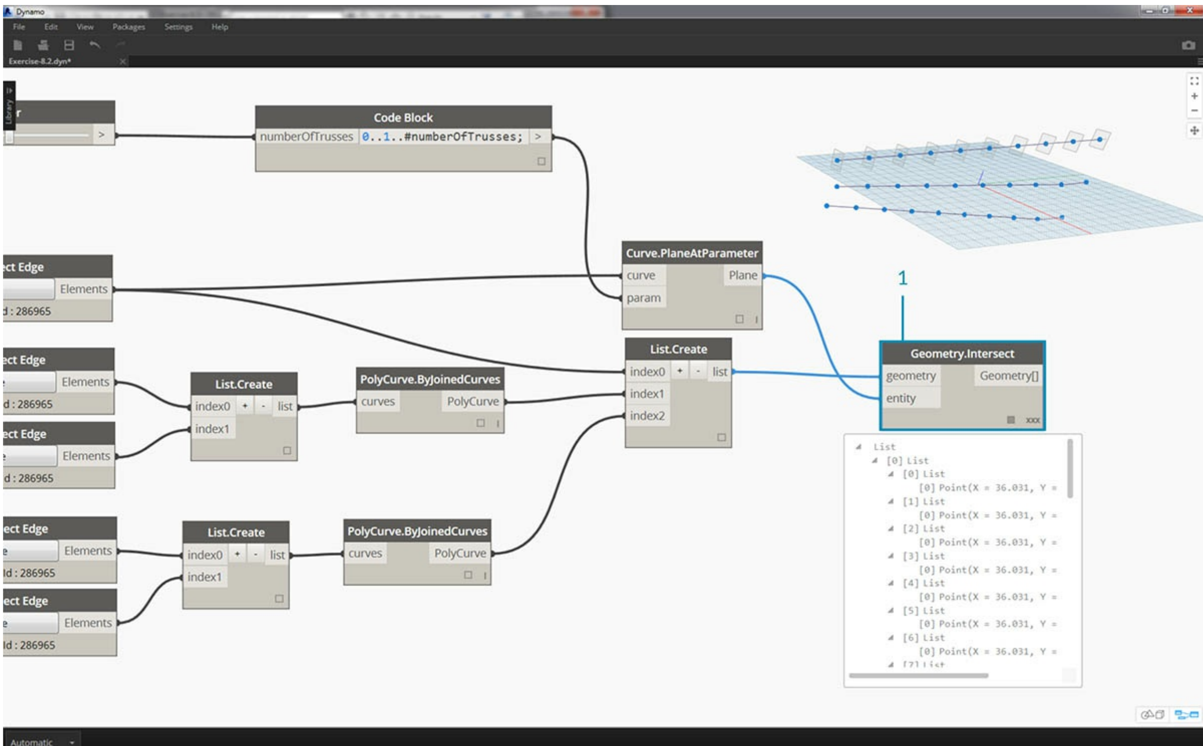


Воспользуйтесь верхней кривой, которая представляет собой линию, расположенную по всей ширине фасада. Создайте плоскости вдоль этой линии для пересечения с набором кривых, которые были сгруппированы в списке.

1. С помощью *блока кода* задайте диапазон, используя синтаксис `0..1..#numberOfTrusses;`
2. Встройте узел *Integer Slider* в набор входных данных для блока кода. Как можно догадаться, он будет задавать количество ферм.

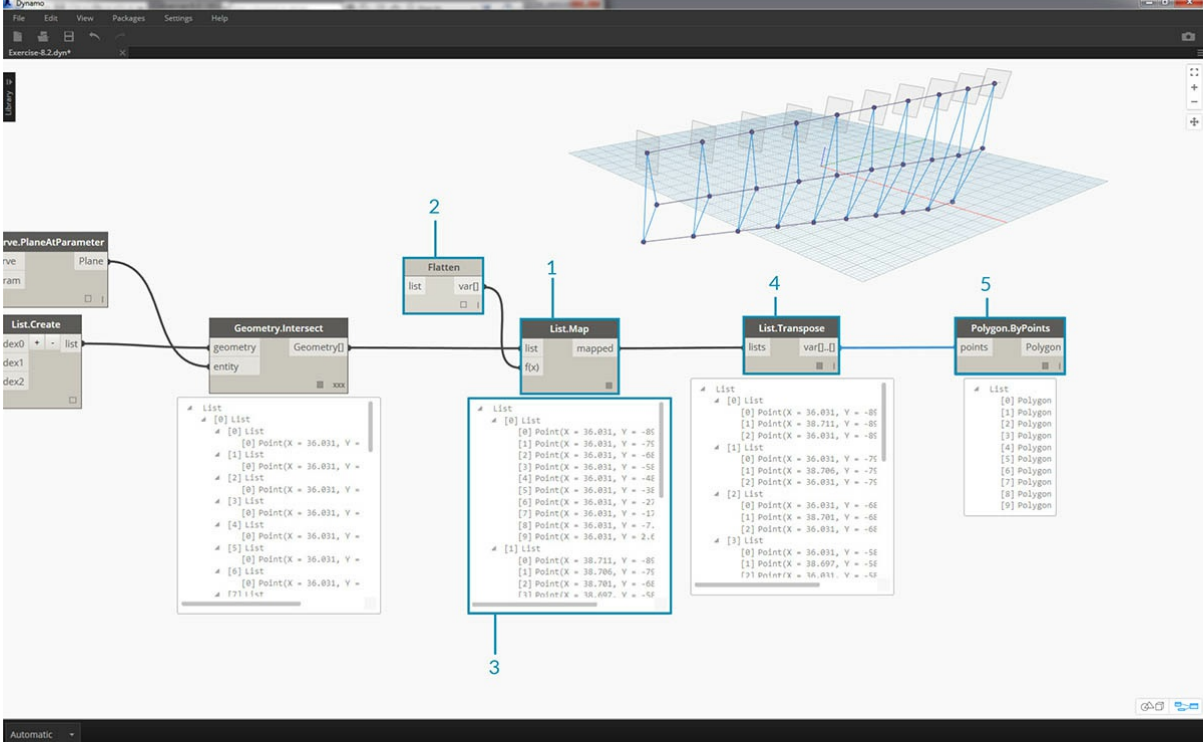
Обратите внимание, что регулятор контролирует количество объектов в диапазоне от *0* до *1*.

3. Встройте *блок кода* в набор входных данных *param* узла *Curve.PlaneAtParameter*, а верхнюю кромку — в набор входных данных *curve*. Будет создано десять плоскостей, равномерно распределенных по ширине фасада.



Плоскость — это абстрактный элемент геометрии, представляющий собой бесконечное двумерное пространство. Плоскости отлично подходят для создания контуров и пересечений, что и требуется на данном этапе.

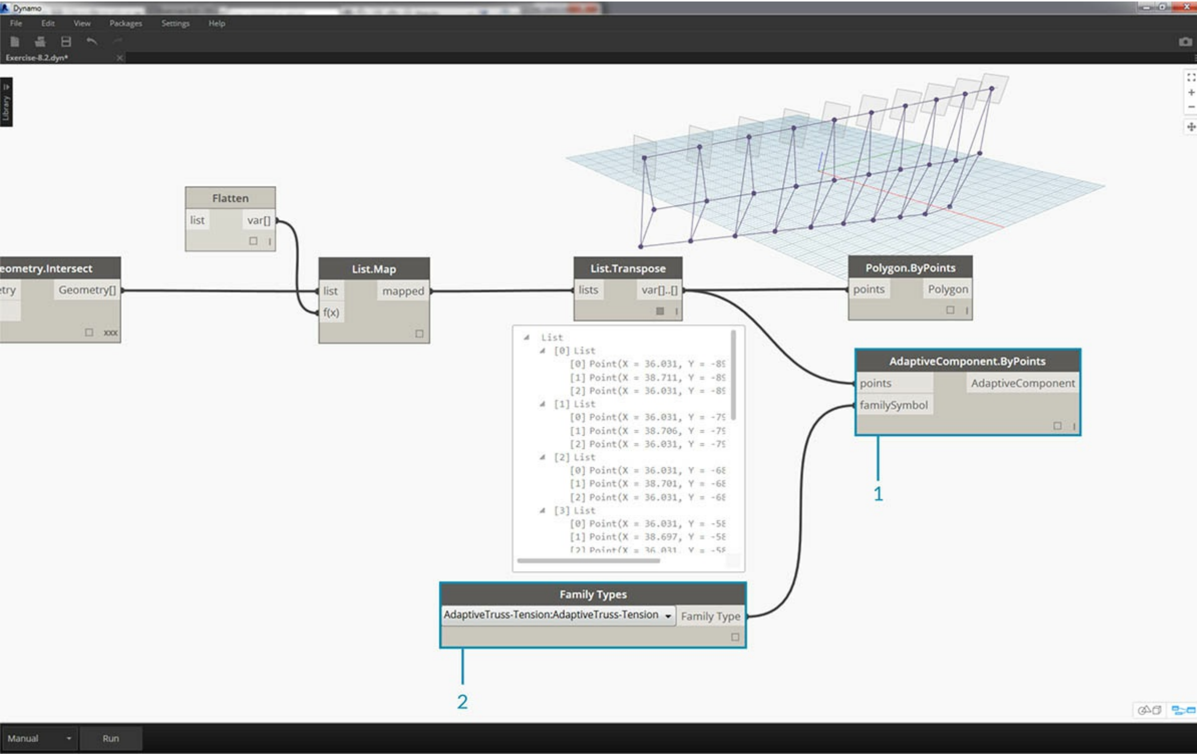
1. Используя узел *Geometry.Intersect* (обратите внимание на режим переплетения «Декартово произведение»), встройте компонент *Curve.PlaneAtParameter* в набор входных данных *entity* узла *Geometry.Intersect*. Встройте основной узел *List.Create* в набор входных данных *geometry*. Теперь на видовом экране Dynamo можно увидеть точки, обозначающие пересечение кривых с заданными плоскостями.



Обратите внимание, что выходные данные содержат список, в который вложен список с еще одним вложенным списком. Слишком большое число списков для решаемой задачи. Необходимо частично выровнять их. Спустимся на шаг вниз по списку и применим к результату

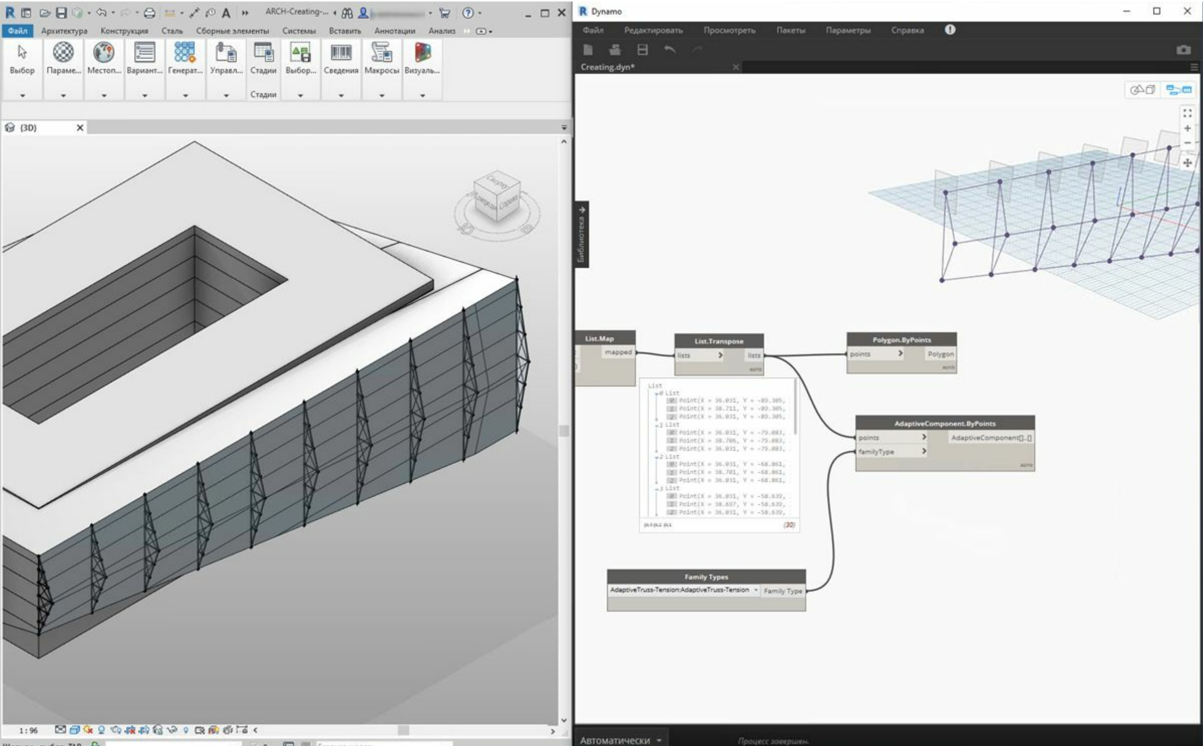
функцию выравнивания. Для этого используем операцию *List.Map*, описанную в главе о списках.

1. Встройте узел *Geometry.Intersect* в набор входных данных List узла *List.Map*.
2. Встройте узел *Flatten* в набор входных данных f(x) узла *List.Map*. В результате получится 3 списка с количеством элементов, соответствующим количеству ферм.
3. Необходимо изменить эти данные. Для создания экземпляра фермы следует использовать такое же количество адаптивных точек, какое определено в семействе. Так как адаптивный компонент состоит из трех точек, вместо трех списков, содержащих по 10 элементов (numberOfTrusses), необходимо получить 10 списков с тремя элементами в каждом. Так можно создать 10 адаптивных компонентов.
4. Встройте узел *List.Map* в узел *List.Transpose*. Теперь получены нужные данные.
5. Чтобы убедиться в правильности данных, добавьте узел *Polygon.ByPoints* в рабочую область и проверьте результат в области предварительного просмотра Dynamo.

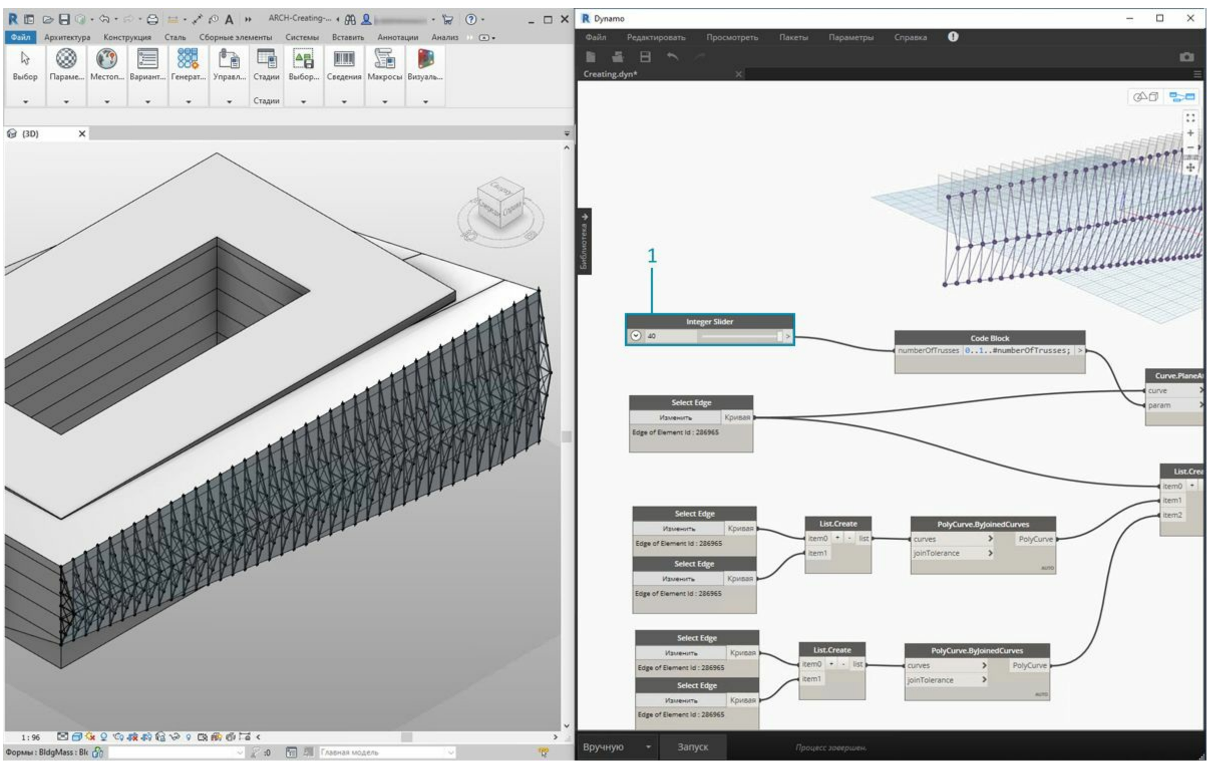


Массив адаптивных компонентов создается так же, как полигоны.

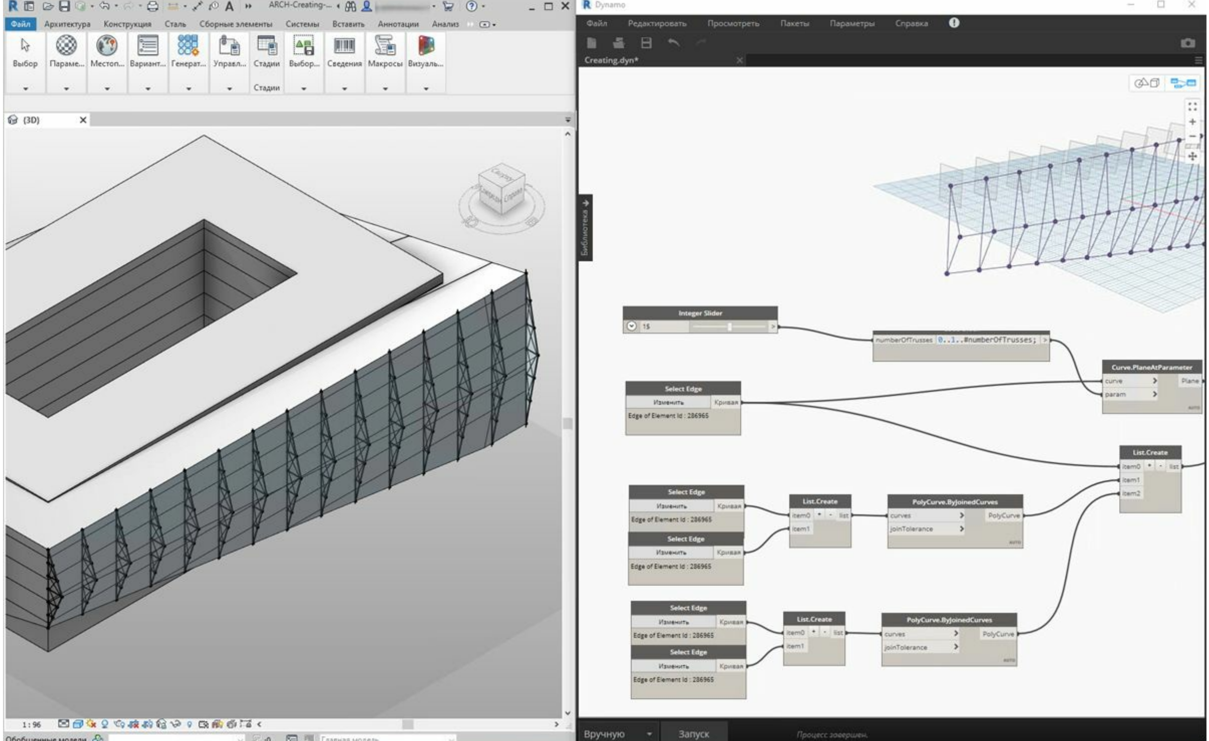
1. Добавьте узел *AdaptiveComponent.ByPoints* в рабочую область, встройте узел *List.Transpose* в набор входных данных *points*.
2. С помощью узла *Family Types* выберите семейство *AdaptiveTruss* и встройте его в набор входных данных *familySymbol* узла *AdaptiveComponent.ByPoints*.



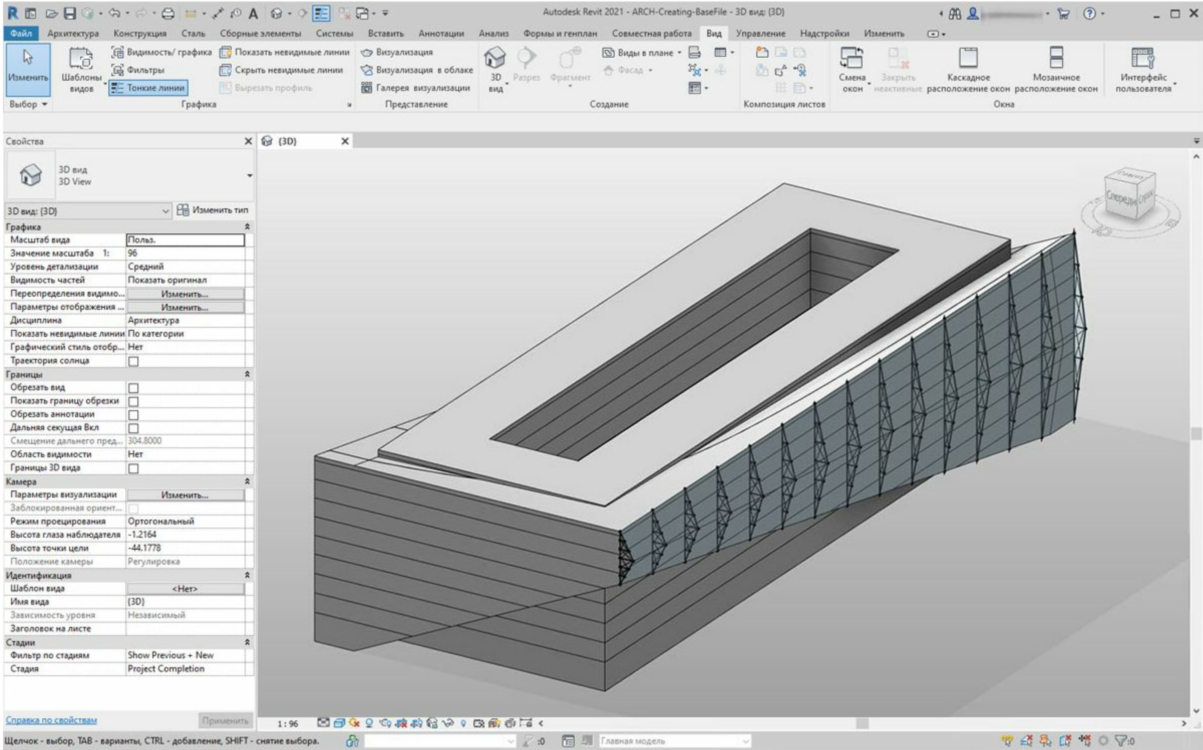
Если проверить результаты в Revit, можно увидеть десять ферм, равномерно размещенных по ширине фасада.



1. Для «зондирования» графика увеличим значение *numberOfTrusses* до *40* с помощью *Integer Slider*. В итоге получилось слишком большое и нереальное количество ферм, но при этом параметрическая связь действует.



1. Чтобы упростить систему ферм, уменьшим их количество (*numberOfTrusses*) до *15*.



В качестве финальной проверки выберем формообразующий элемент в Revit и отредактируем параметры экземпляра. После изменения формы здания ферма должна тоже измениться. Обратите внимание, что это изменение можно наблюдать, только если график Dynamo открыт. Сразу после закрытия связь будет разорвана.

Элементы DirectShape

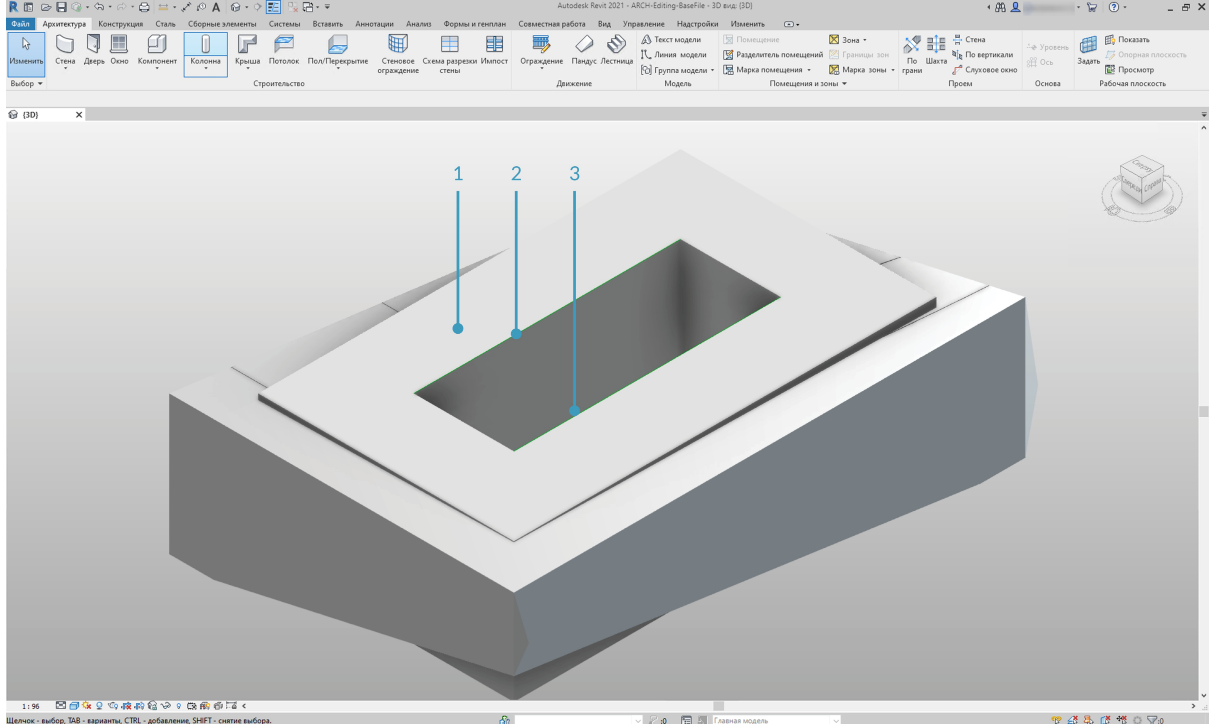
Еще одним способом импорта параметрической геометрии Dynamo в Revit является DirectShape. В целом элемент DirectShape и связанные классы отвечают за хранение созданных во внешних программах геометрических форм в документах Revit. Геометрия может включать в себя замкнутые тела или сети. Основной задачей DirectShape является импорт форм из других форматов данных, например IFC или STEP, когда недостаточно информации для создания реального элемента Revit. Как и при работе с форматами IFC и STEP, функция DirectShape подходит для импорта созданных в Dynamo геометрических объектов в проекты Revit в качестве реальных элементов.

Рассмотрим импорт геометрии Dynamo в проект Revit с помощью DirectShape. С помощью этого метода можно назначить категорию, материал и имя импортированной геометрии, сохранив при этом параметрическую связь с графиком Dynamo.

Упражнение

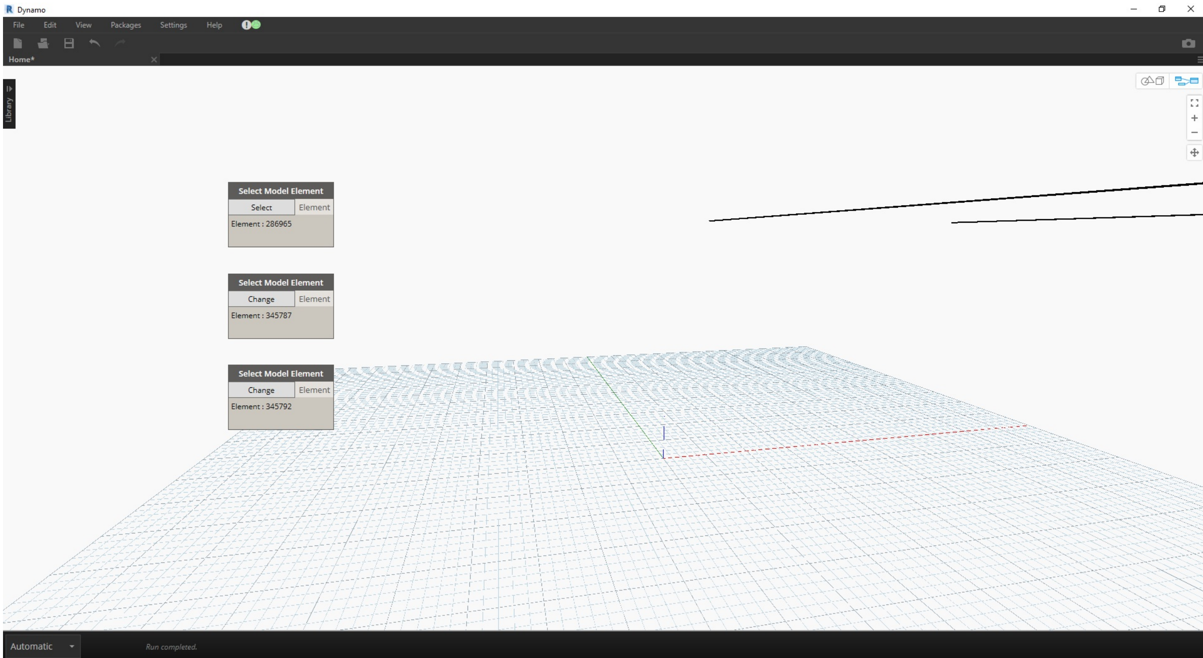
Скачайте файлы примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении.

1. DirectShape.dyn
2. ARCH-DirectShape-BaseFile.rvt

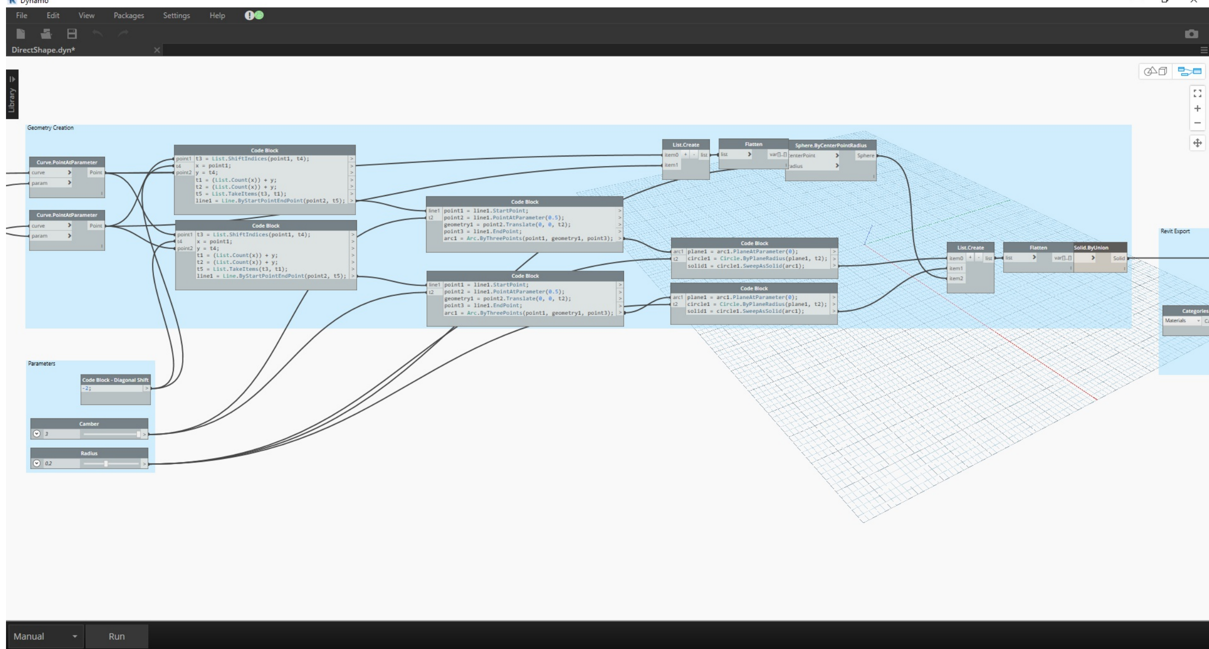


Сначала откройте файла примера для этого урока — ARCH-DirectShape-BaseFile.rvt.

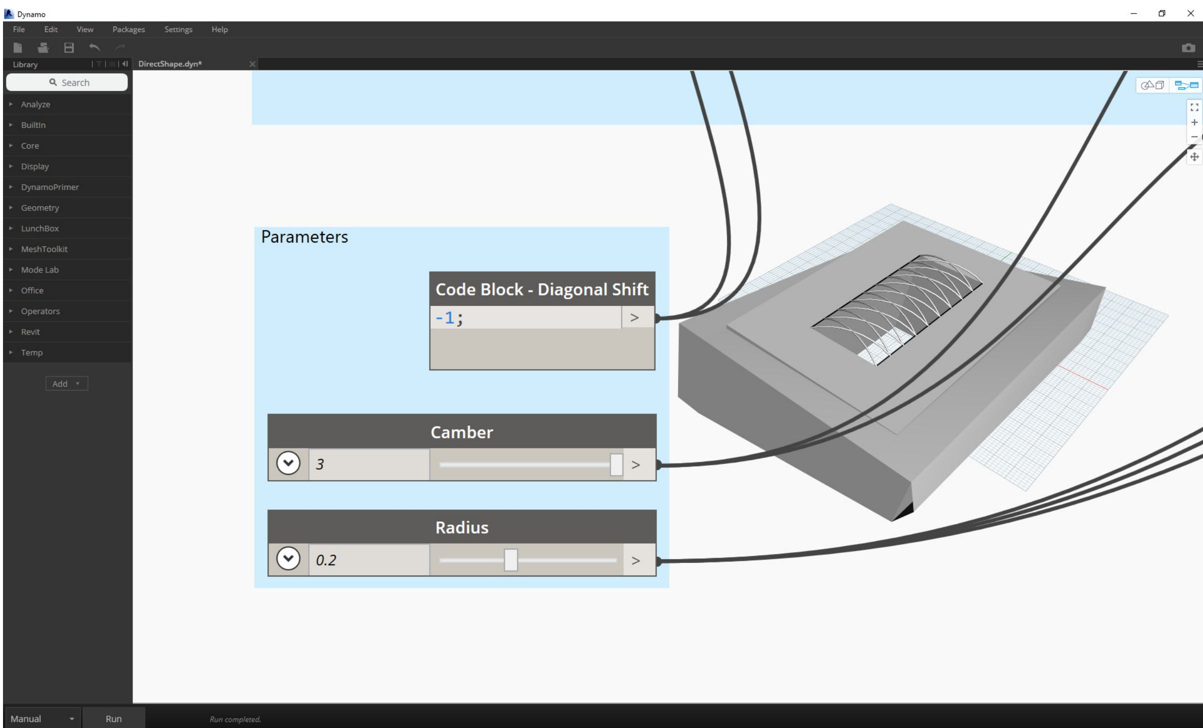
1. На 3D-виде отображается формообразующий элемент здания из предыдущего урока.
2. Вдоль кромки атриума имеется одна базовая кривая. Она будет использоваться в Dynamo в качестве опорной.
3. Вдоль противоположной кромки атриума проходит еще одна базовая кривая. Она также будет использоваться в Dynamo в качестве опорной.



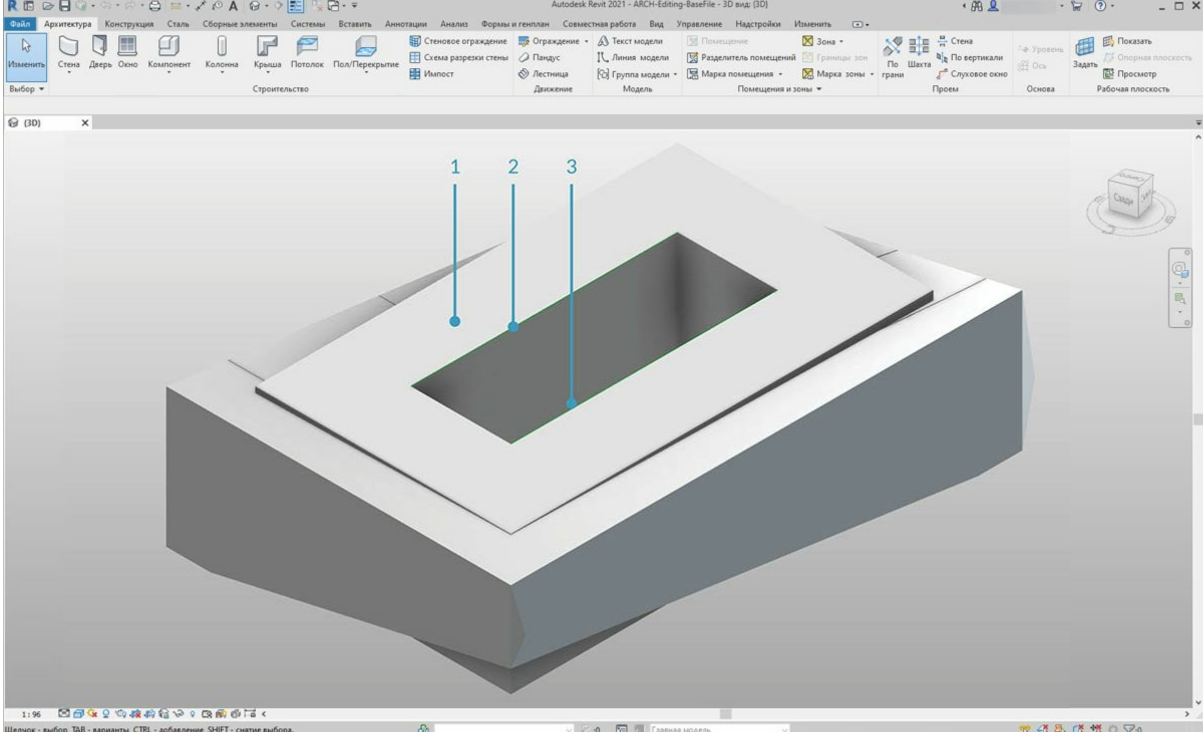
1. Чтобы установить связь с геометрией в Dynamo, используем узел *Select Model Element* для каждого элемента в Revit. Выберите формообразующий элемент в Revit и импортируйте геометрию в Dynamo с помощью функции *Element.Faces*. Формообразующий элемент должен отображаться в области предварительного просмотра Dynamo.
2. Импортируйте первую базовую кривую в Dynamo с помощью функций *Select Model Element* и *CurveElement.Curve*.
3. Импортируйте вторую базовую кривую в Dynamo с помощью функций *Select Model Element* и *CurveElement.Curve*.



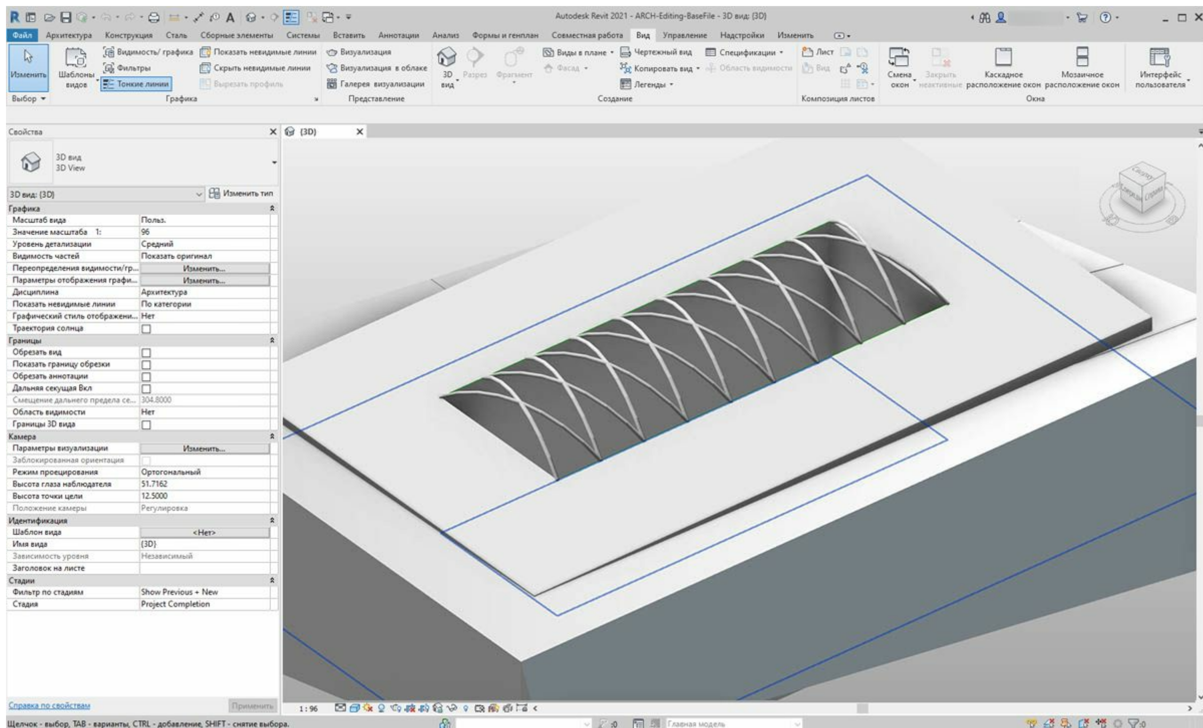
1. Если уменьшить масштаб графика и панорамировать его вправо, можно увидеть большую группу узлов. Это геометрические операции, которые создают решетчатую конструкцию на крыше в области предварительного просмотра Dynamo. Эти узлы генерируются с помощью функции *Node to Code*, описанной в разделе о блоках кода данного учебника.
2. Конструкция определяется тремя основными параметрами: Diagonal Shift, Camber и Radius.



Увеличьте масштаб отображения параметров этого графика. Можно выполнить их зондирование, чтобы получить другие выходные данные геометрии.



1. Если поместить узел *DirectShape.ByGeometry* в рабочую область, можно увидеть, что он имеет четыре набор входных данных: **geometry, category, material** и **name**.
2. Геометрия представляет собой твердое тело, созданное на базе части графика, с помощью которой формируется геометрия.
3. Входные данные категории выбираются с помощью узла *Categories*. В данном случае используется значение Structural Framing.
4. Входные данные материала выбираются с помощью вышеупомянутого массива узлов, хотя в данном случае проще задать значение по умолчанию.



Вернитесь в Revit после выполнения сценария Dynamo. Импортированную геометрию можно видеть на крыше в проекте. Это скорее не обобщенная модель, а элемент несущего каркаса. Параметрическая связь с Dynamo сохраняется.

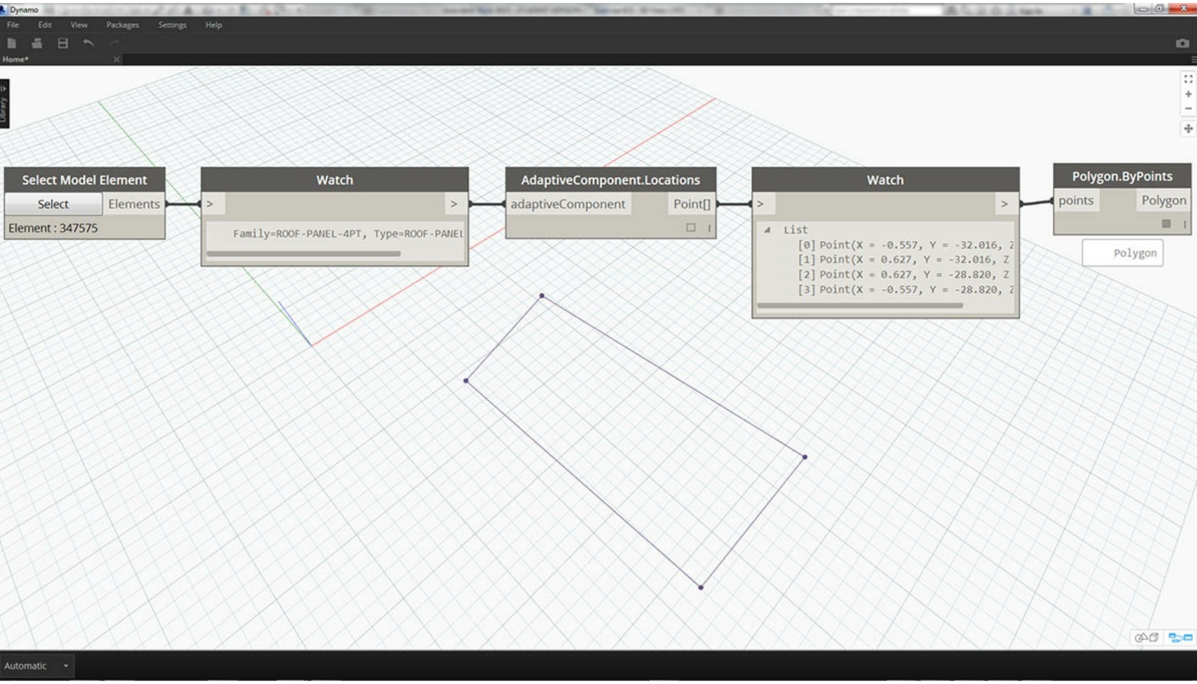
Адаптация

Адаптация

Ранее мы рассмотрели редактирование базового формообразующего элемента здания. Теперь мы предлагаем подробнее изучить взаимодействие Dynamo и Revit путем редактирования большого количества элементов за один подход. Адаптация усложняется при увеличении масштаба, так как структура данных требует более сложных операций со списками. Однако основные принципы работы при этом остаются прежними. Рассмотрим возможности расчета с помощью набора адаптивных компонентов.

Местоположение точки

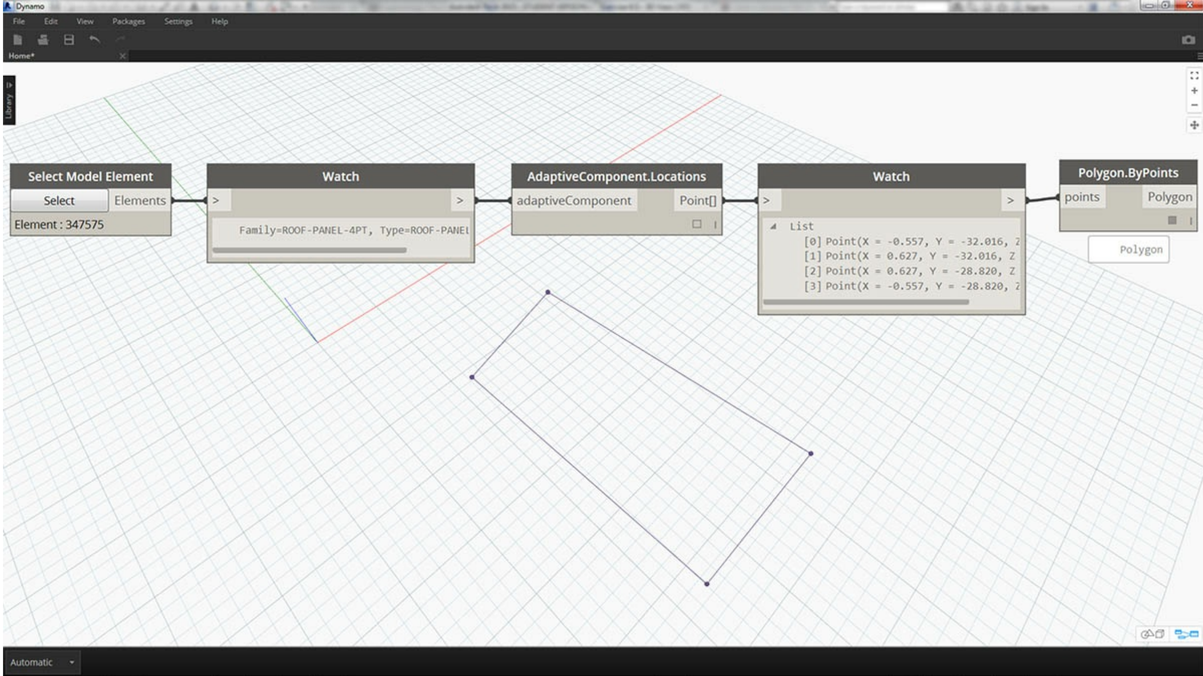
Предположим, что вы создали набор адаптивных компонентов и хотите отредактировать параметры на основании местоположений точек. Например, точки могут определять параметр толщины, который связан с площадью элемента. Кроме того, они могут отвечать за параметр непрозрачности, связанный с уровнем инсоляции за год. Dynamo позволяет связать расчет с параметрами за несколько простых шагов. Основы этого процесса будут приведены в упражнениях ниже.



Опросите адаптивные точки выбранного адаптивного компонента с помощью узла *AdaptiveComponent.Locations*. Это позволяет выполнить расчет для абстрагированной версии элемента Revit.

Получив местоположение точек адаптивных компонентов, можно запустить несколько расчетов для элемента. Например, адаптивный компонент по четырем точкам позволяет изучить отклонение от плоскости для заданной панели.

Расчет ориентации относительно солнца



Используя повторное сопоставление, можно сопоставить набор данных с диапазоном параметров. Это основной инструмент параметрического моделирования, который вы сможете изучить подробнее, выполнив упражнение ниже.

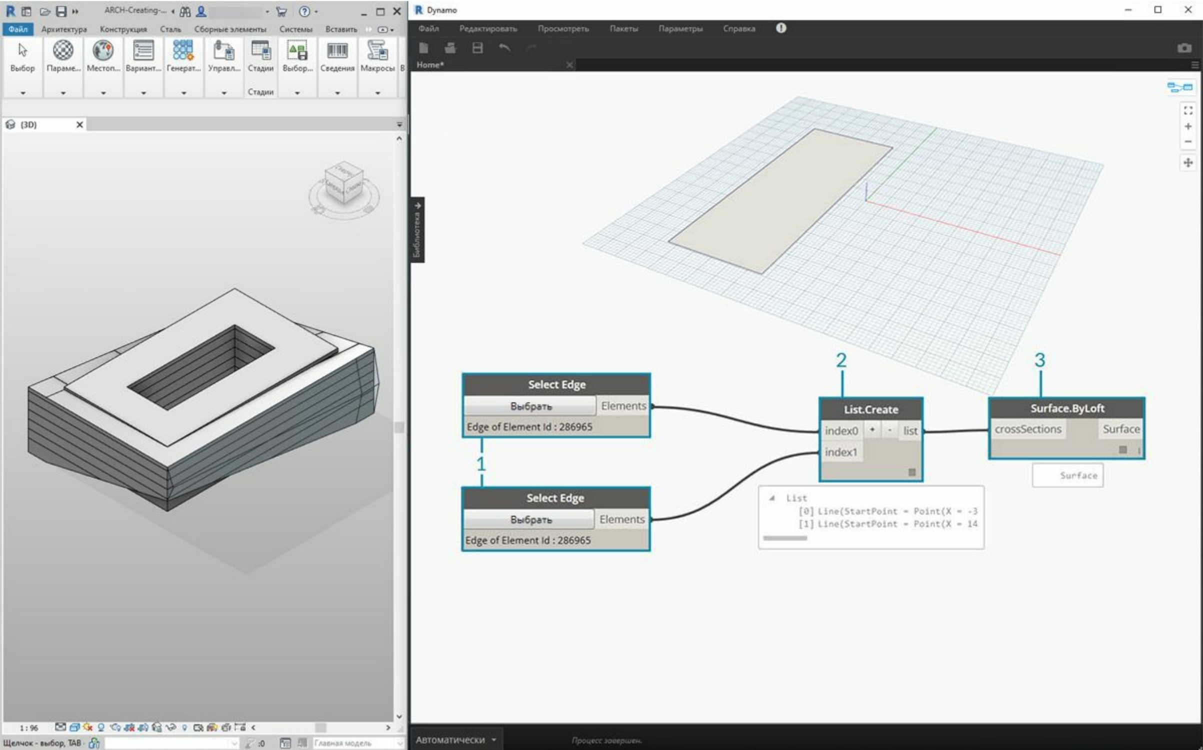
В Dynamo местоположения точек адаптивных компонентов можно использовать для создания плоскости наилучшего вписывания для каждого элемента. Кроме того, можно запросить положение солнца в файле Revit и изучить относительную ориентацию плоскости к солнцу по сравнению с другими адаптивными компонентами. Давайте выполним это путем создания алгоритмической кровли в упражнении ниже.

Упражнение

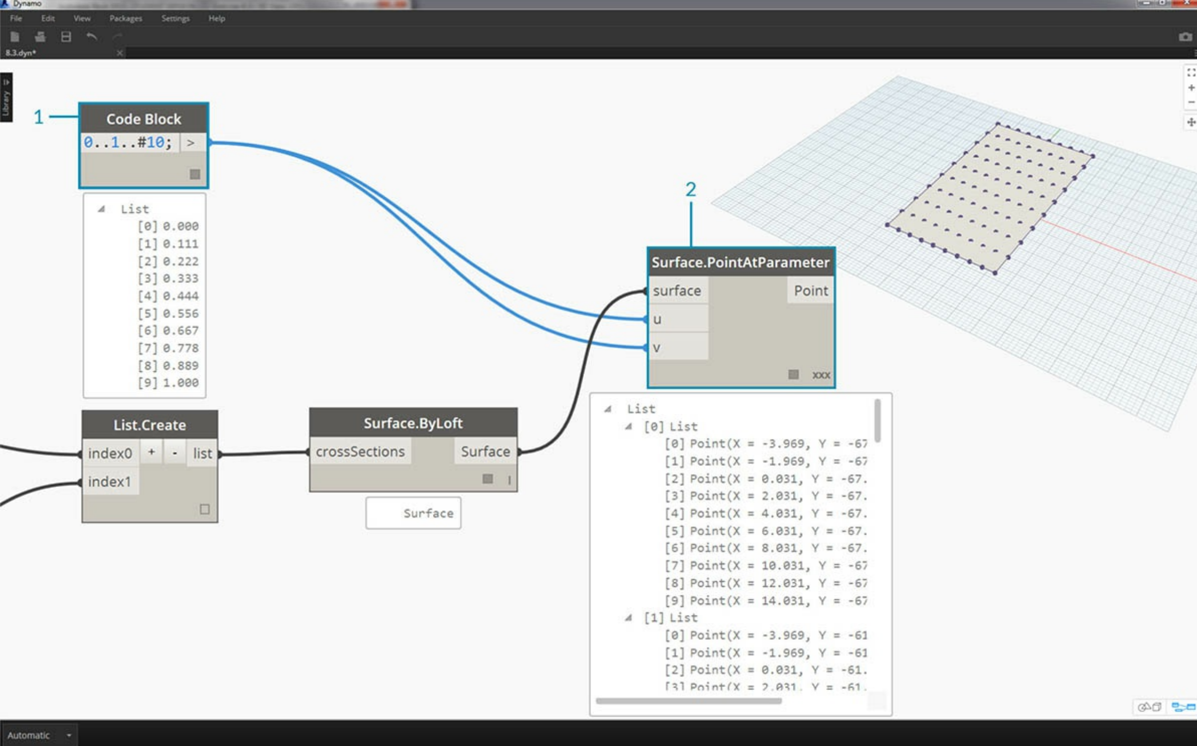
Скачайте файлы примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении.

1. Customizing.dyn
2. ARCH-Customizing-BaseFile.rvt

В данном упражнении будут подробнее рассмотрены техники из предыдущего раздела. Вам понадобится определить параметрическую поверхность на основе элементов Revit, создать экземпляры адаптивных компонентов по четырем точкам, а затем отредактировать их на основании ориентации относительно солнца.

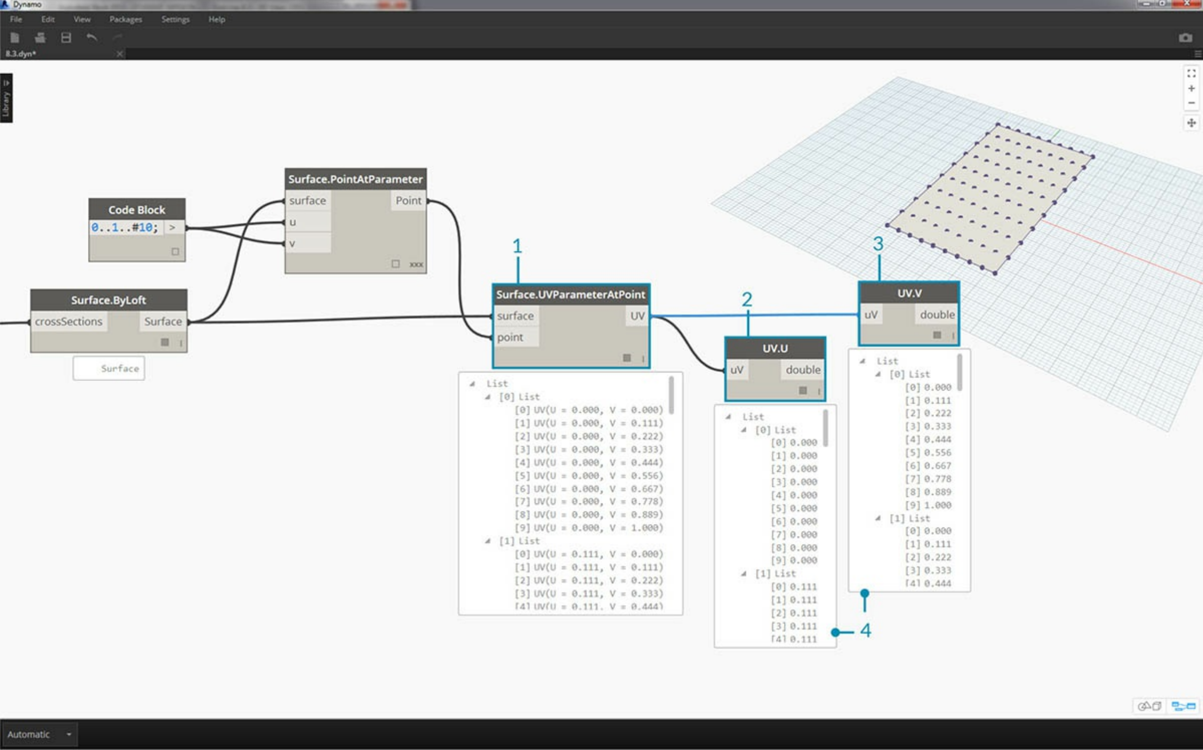


1. Для начала выберите два ребра с помощью узла *Select Edge*. Два ребра представляют собой длинные пролеты атриума.
2. Объедините два ребра в один список, используя узел *List.Create*.
3. Создайте поверхность между двумя ребрами с помощью узла *Surface.ByLoft*.



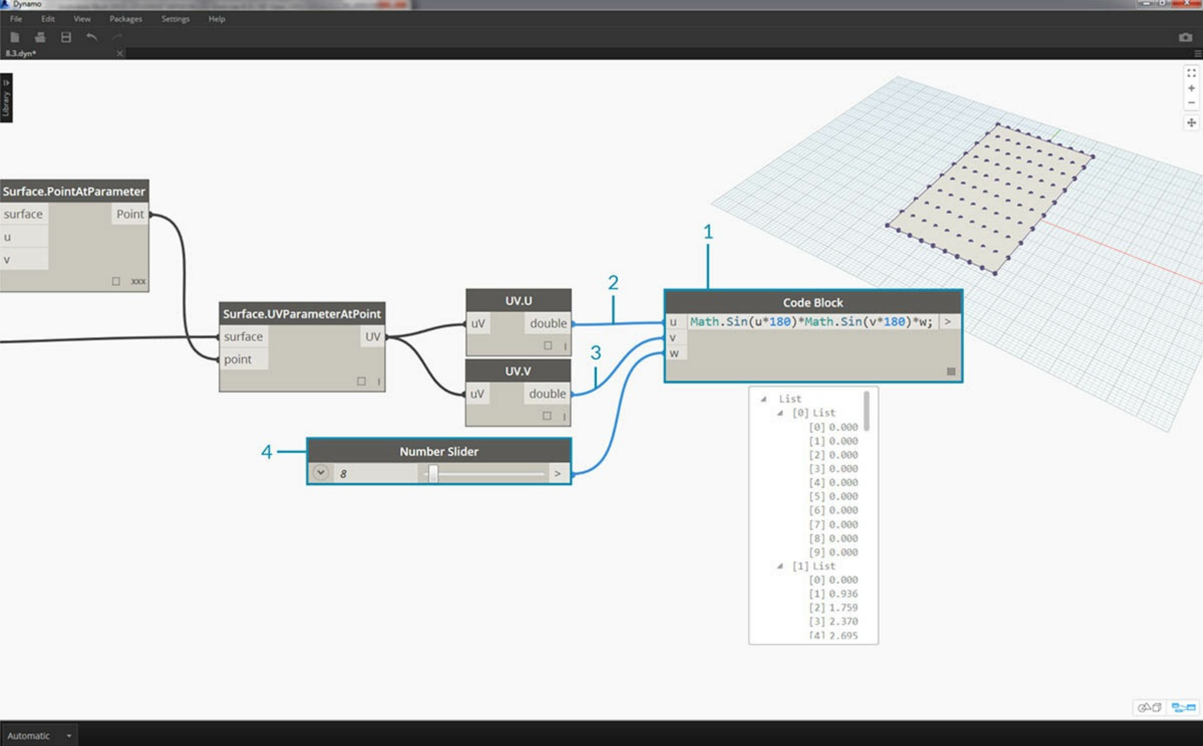
1. В узле *Code Block* задайте диапазон от 0 до 1 с 10 равномерно распределенными значениями: `0..1..#10;`.
2. Соедините *Code Block* с портами ввода *u* и *v* узла *Surface.PointAtParameter*, а также соедините узел *Surface.ByLoft* с портом ввода *surface*. Щелкните узел правой кнопкой мыши и задайте для параметра *Переплетение* значение *Декартово произведение*. На поверхности появится сетка из точек.

Эта сетка из точек служит в качестве управляющих точек для параметрической поверхности. Необходимо извлечь положения u и v каждой из этих точек, чтобы ввести их в параметрическую формулу и сохранить существующую структуру данных. Это можно сделать, запросив местоположения параметров только что созданных точек.

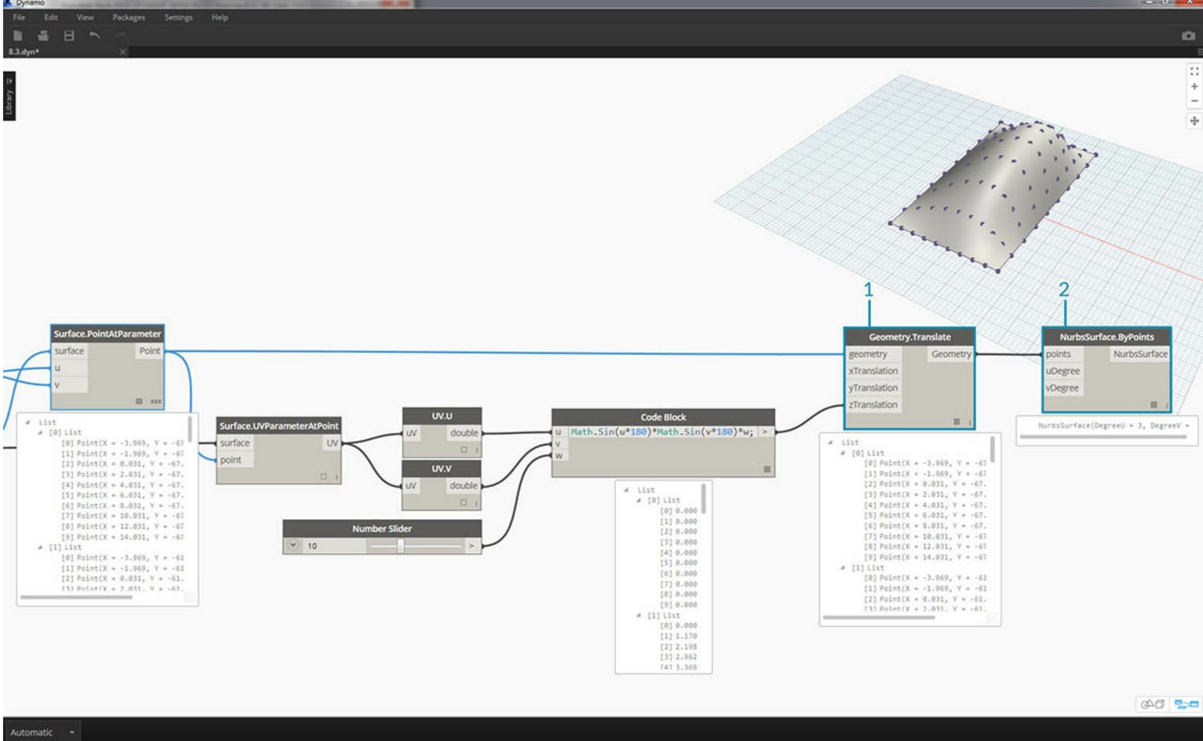


1. Добавьте узел *Surface.ParameterAtPoint* в рабочую область и соедините порты ввода, как показано выше.

2. Запросите значения *u* для этих параметров с помощью узла *UV.U*.
3. Запросите значения *v* для этих параметров с помощью узла *UV.V*.
4. В выходных данных отображаются соответствующие значения *u* и *v* для каждой точки поверхности. Вы получили диапазон от *0* до *1* для каждого значения, обеспечена правильная структура данных, и можно применять параметрический алгоритм.



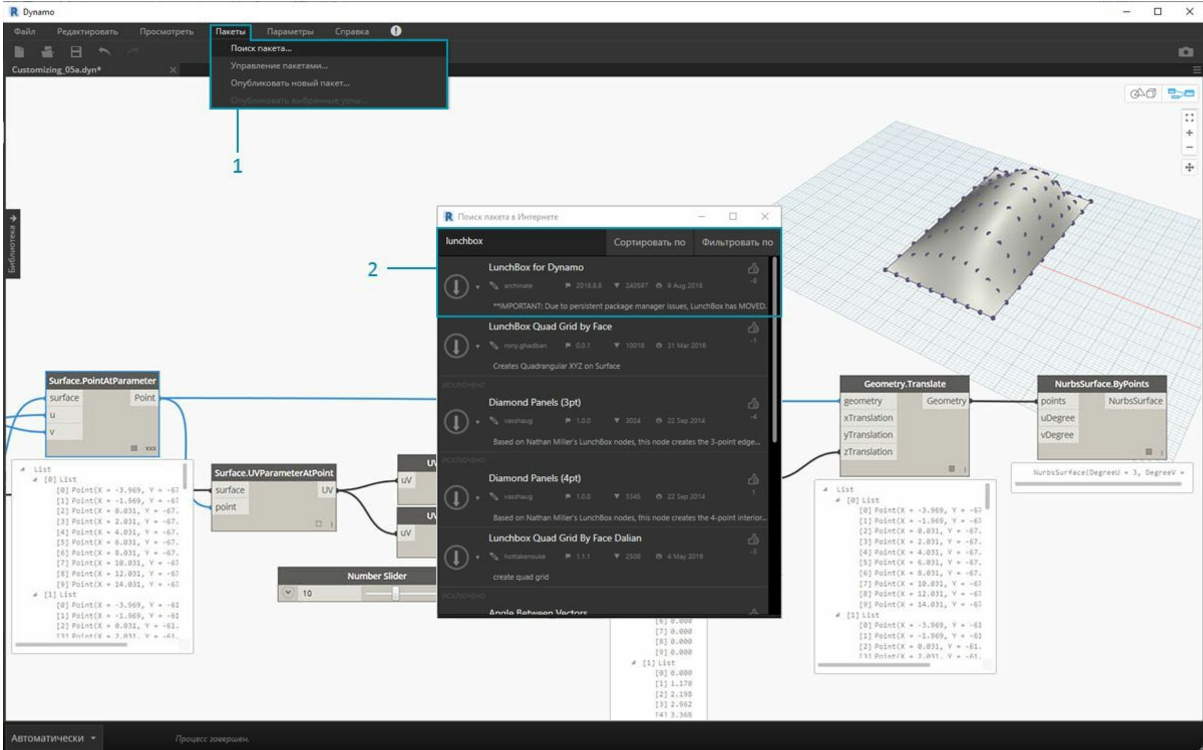
1. Добавьте узел *Code Block* в рабочую область и введите код: `Math.Sin(u*180)*Math.Sin(v*180)*w;`. Это параметрическая функция, которая создает синусоидную выпуклость на основе плоской поверхности.
2. Порт ввода *u* соединяется с узлом *UV.U*.
3. Порт ввода *v* соединяется с узлом *UV.V*.
4. Порт ввода *w* представляет *амплитуду* формы, поэтому к нему нужно присоединить узел *Number Slider*.



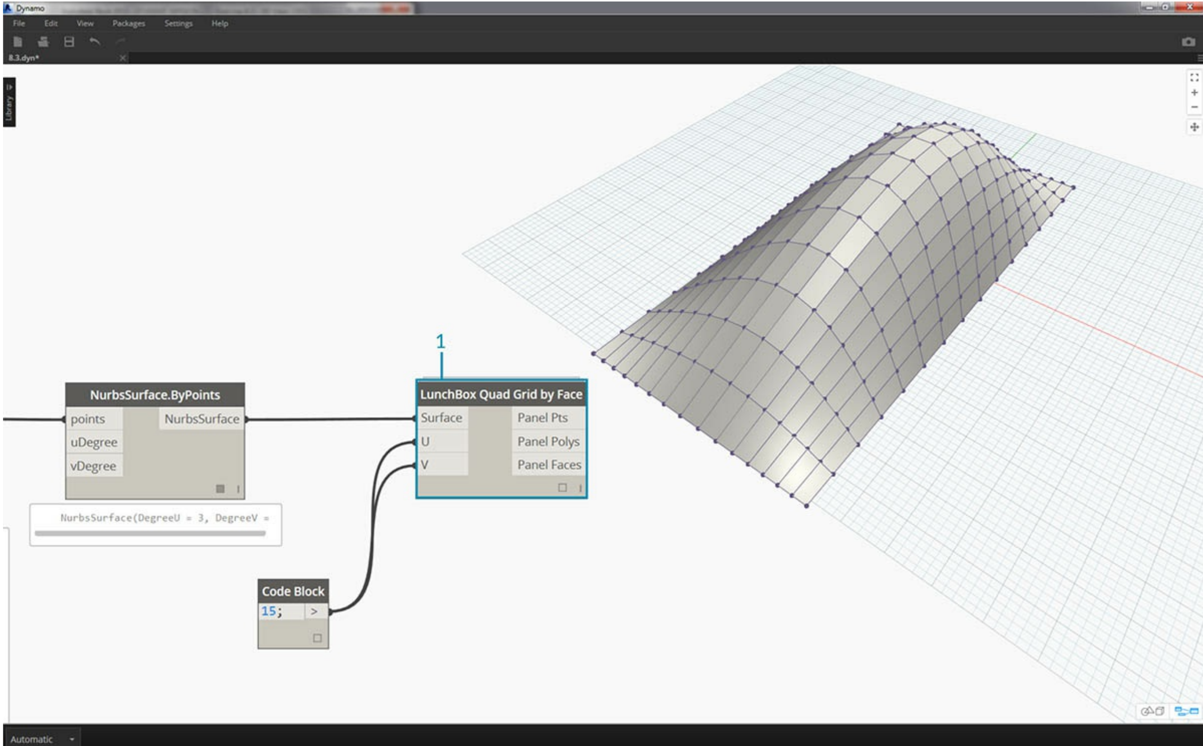
1. Теперь у вас есть список значений в соответствии с алгоритмом. Используйте этот список для перемещения точек вверх по оси *+Z*. Используя узел *Geometry.Translate*, соедините *Code Block* с портом ввода *zTranslation*, а *Surface.PointAtParameter* — с портом ввода *geometry*. В области предварительного просмотра Dynamo отображаются новые точки.

2. Наконец, создайте поверхность с помощью узла *NurbsSurface.ByPoints*, соединив узел из предыдущего шага с портом ввода points. В результате получается параметрическая поверхность. Перетаскивайте регулятор, чтобы уменьшить или увеличить выпуклость.

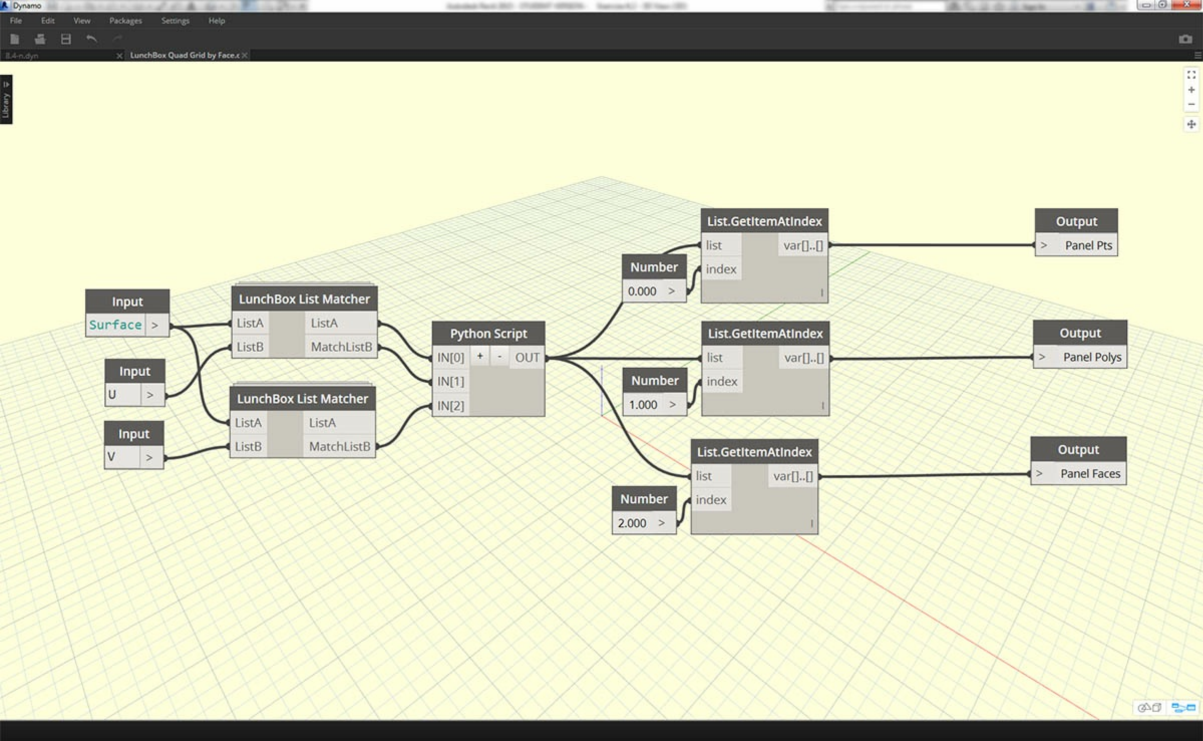
Теперь нам нужно найти способ разбить полученную параметрическую поверхность на панели, чтобы разместить адаптивные компоненты по четырем точкам в формате массива. В Dynamo нет готовой функции для работы с панелями, поэтому обратимся к сообществу пользователей за полезными пакетами Dynamo.



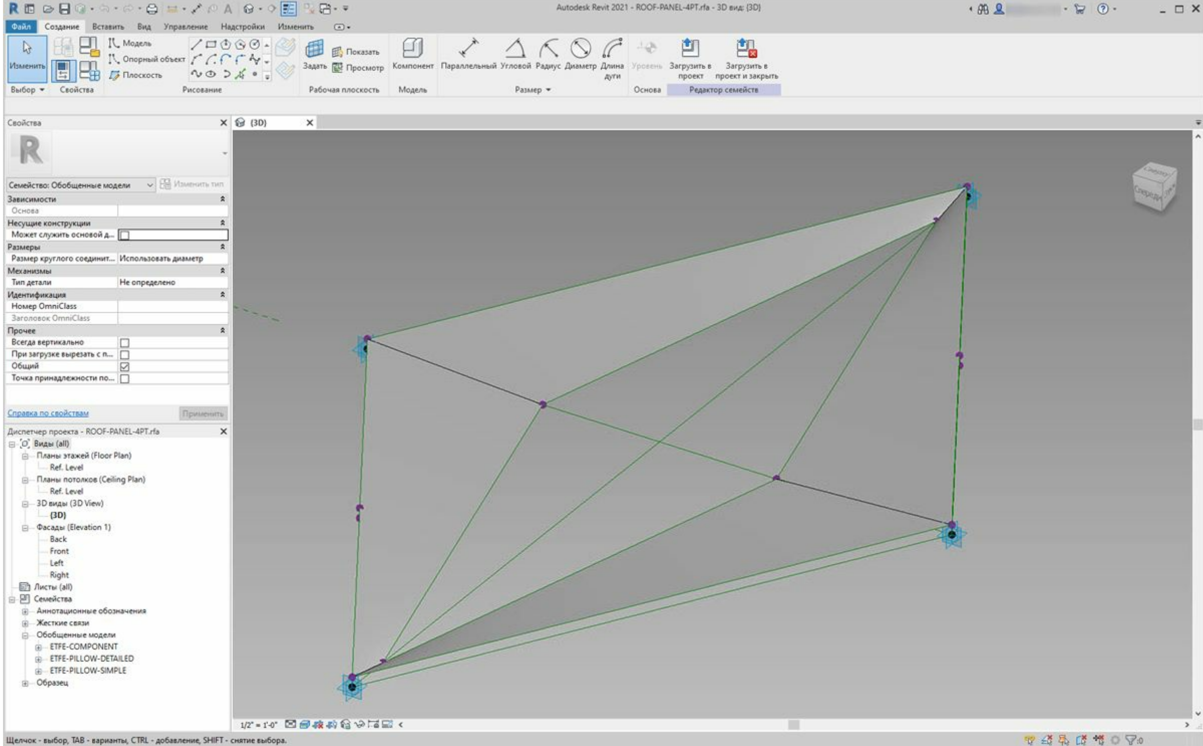
1. Перейдите к разделу *«Пакеты»* > *«Поиск пакета...»*.
2. Найдите *LunchBox* и скачайте файл *LunchBox for Dynamo*. Это очень полезный набор инструментов для подобных операций с геометрией.



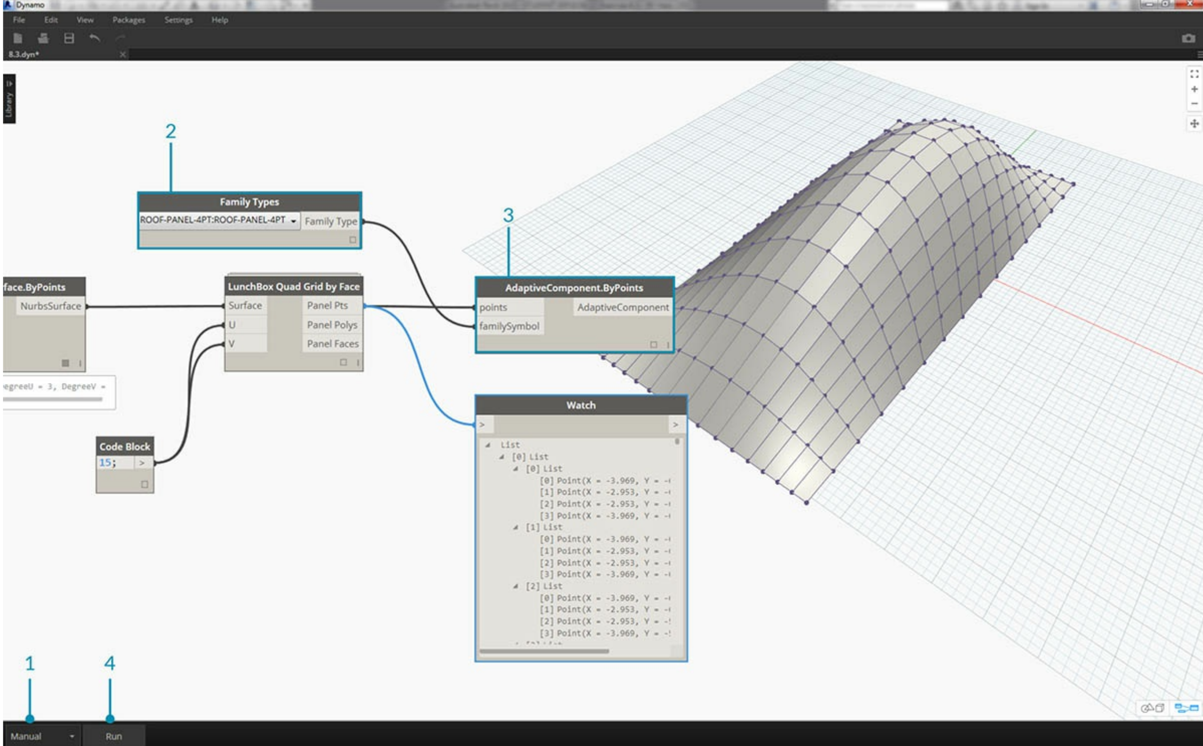
1. После скачивания вы получите полный доступ к пакету LunchBox. Найдите *Quad Grid* и выберите *LunchBox Quad Grid By Face*. Соедините параметрическую поверхность с портом ввода *surface* и установите деления *U* и *V* на *15*. В области предварительного просмотра Dynamo должна отобразиться поверхность с прямоугольными панелями.



Если вас заинтересовала ее настройка, дважды щелкните узел *Lunch Box* и просмотрите параметры.

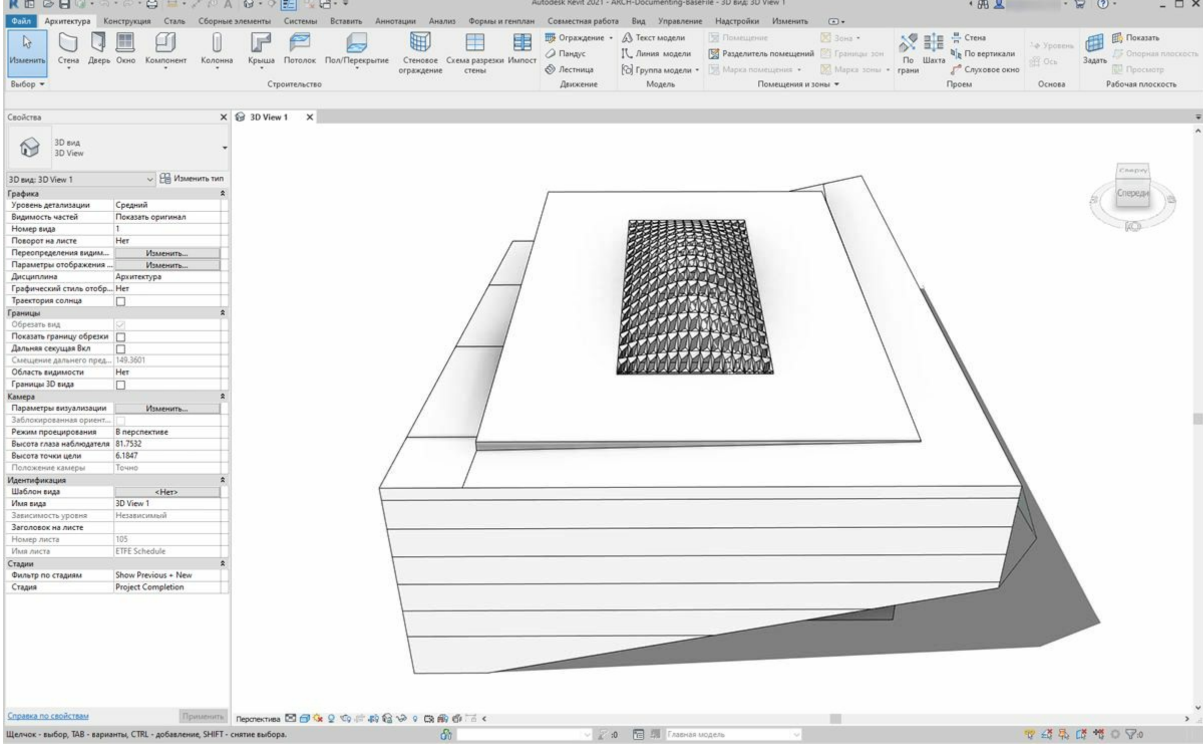


Вернитесь в Revit, чтобы узнать, какой адаптивный компонент используется в этой программе. Нет необходимости его повторять, но это панель крыши, которая будет использоваться в качестве экземпляра массива. Это адаптивный компонент по четырем точкам, который является грубым представлением системы ЭТФЭ. Апертура центральной полости находится на параметре *ApertureRatio*.

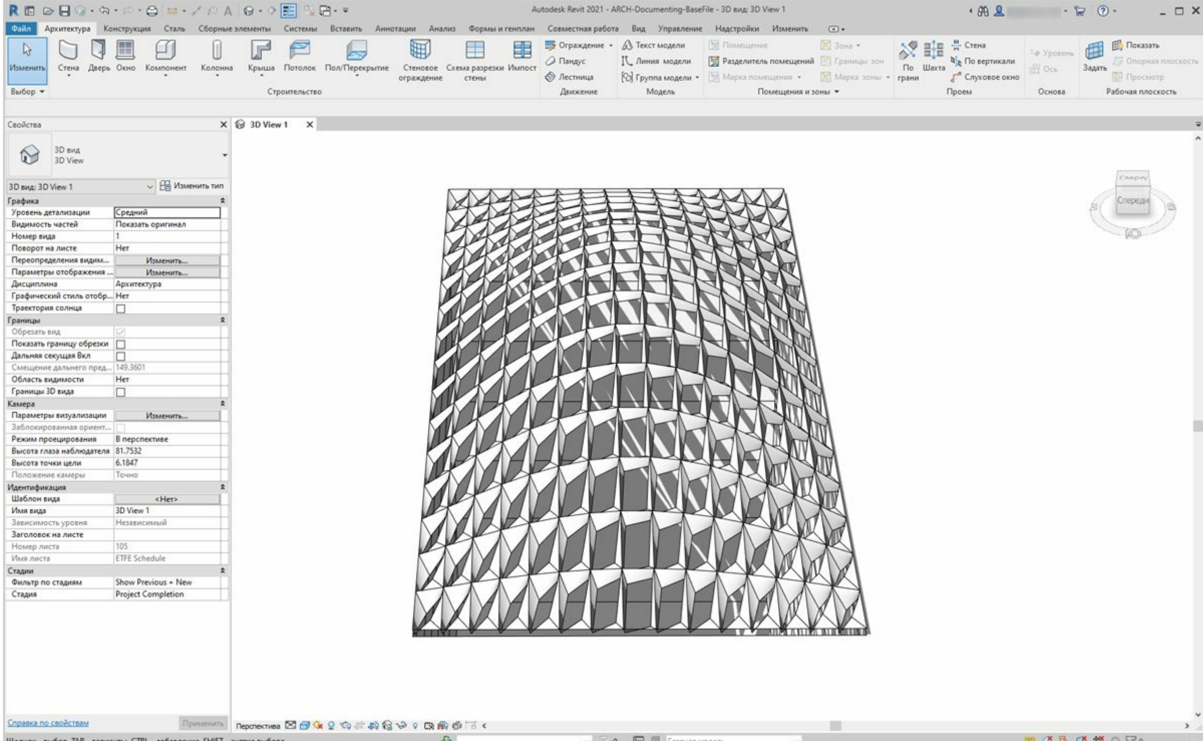


1. Так как вы собираетесь создать большое количество экземпляров геометрии в Revit, убедитесь, что решатель Dynamo находится в *ручном* режиме.
2. Добавьте узел *Family Types* в рабочую область и выберите *ROOF-PANEL-4PT*.
3. Добавьте узел *AdaptiveComponent.ByPoints* в рабочую область, соедините порт вывода *Panel Pts* узла *LunchBox Quad Grid by Face* с портом ввода *points*. Соедините узел *Family Types* с портом ввода *familySymbol*.
4. Нажмите кнопку *Запуск*. Для создания геометрии программе Revit потребуется немного времени *на раздумья*. Если это занимает слишком много времени, слегка уменьшите значение 15 в узле *Code Block*. Это уменьшит количество панелей на крыше.

Примечание. Если Dynamo требуется много времени для расчета узлов, возможно, вам следует воспользоваться функцией заморозки, чтобы поставить на паузу выполнение операций Revit во время создания графика. Для получения дополнительных сведений о заморозке узлов ознакомьтесь с разделом, посвященным заморозке, в [глаге о твердых телах](#).

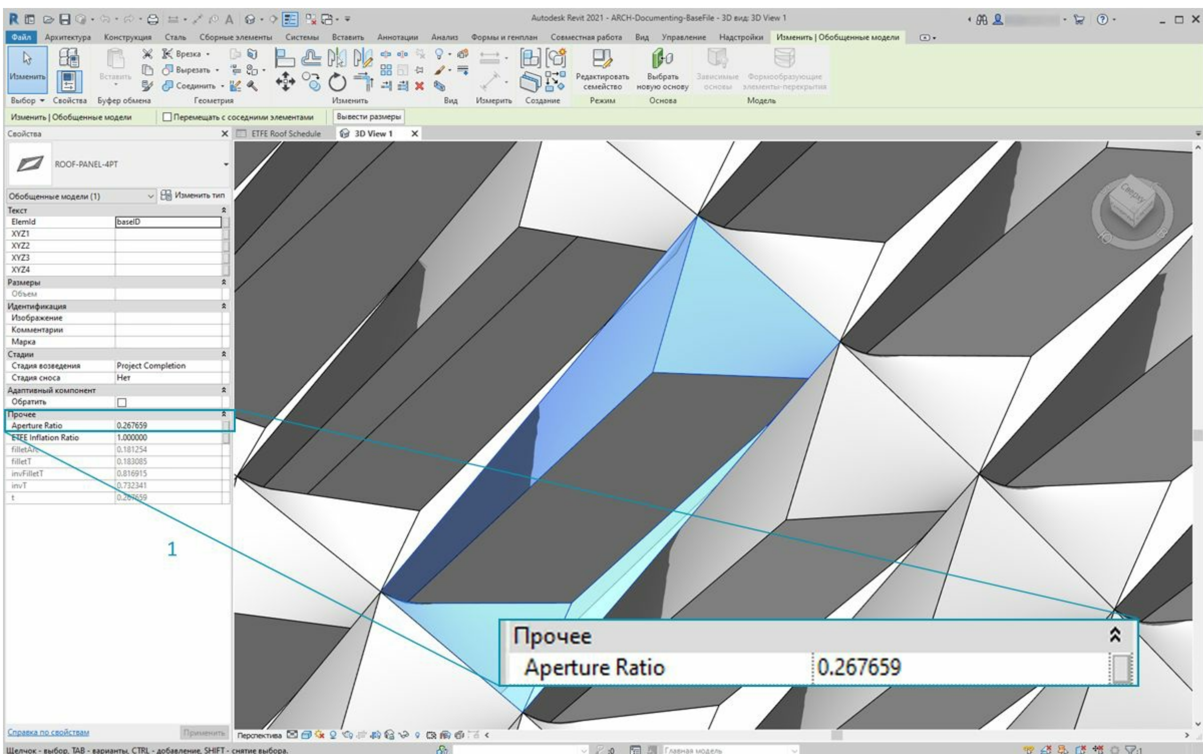


В Revit мы видим массив панелей на крыше.

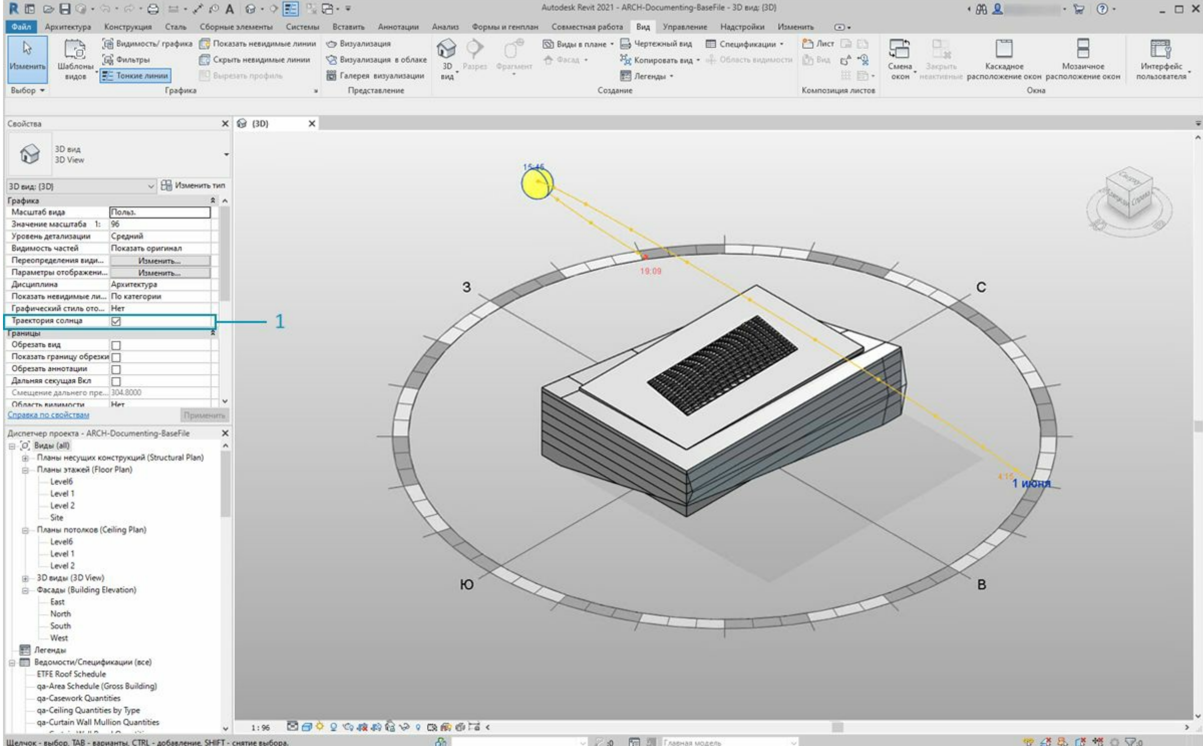


Увеличив масштаб, можно детально рассмотреть свойства их поверхности.

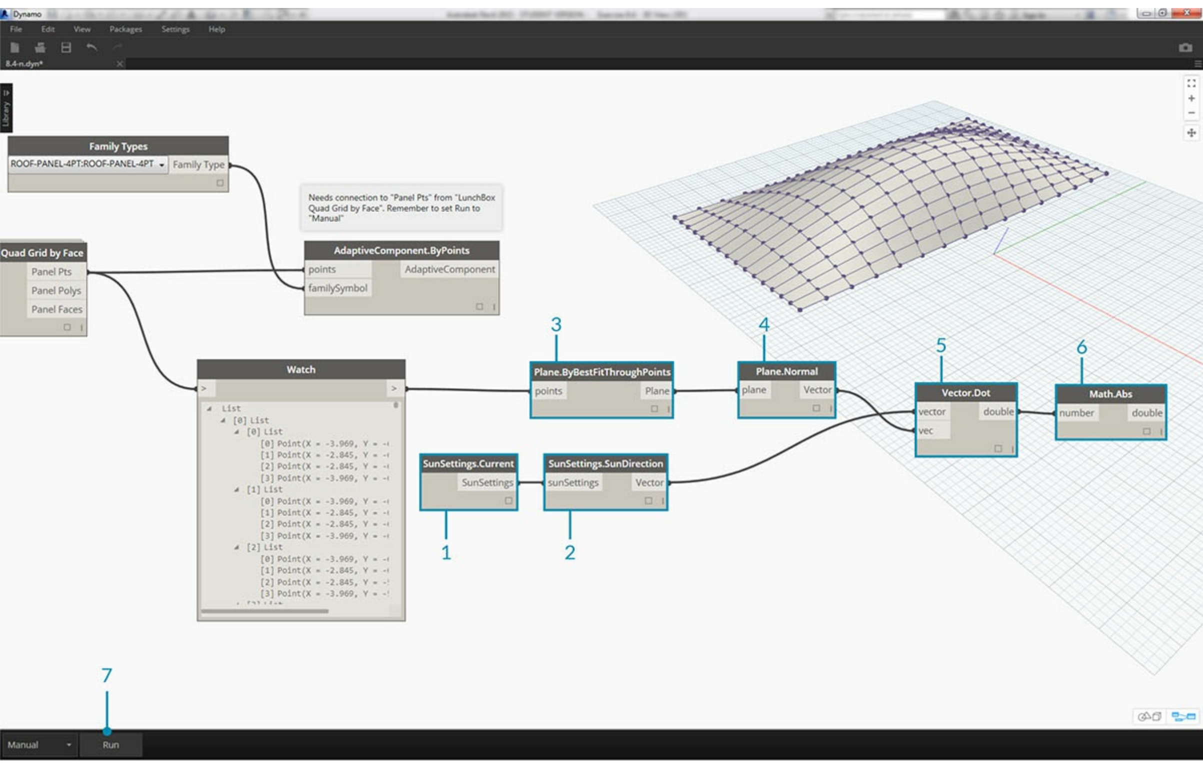
Анализ



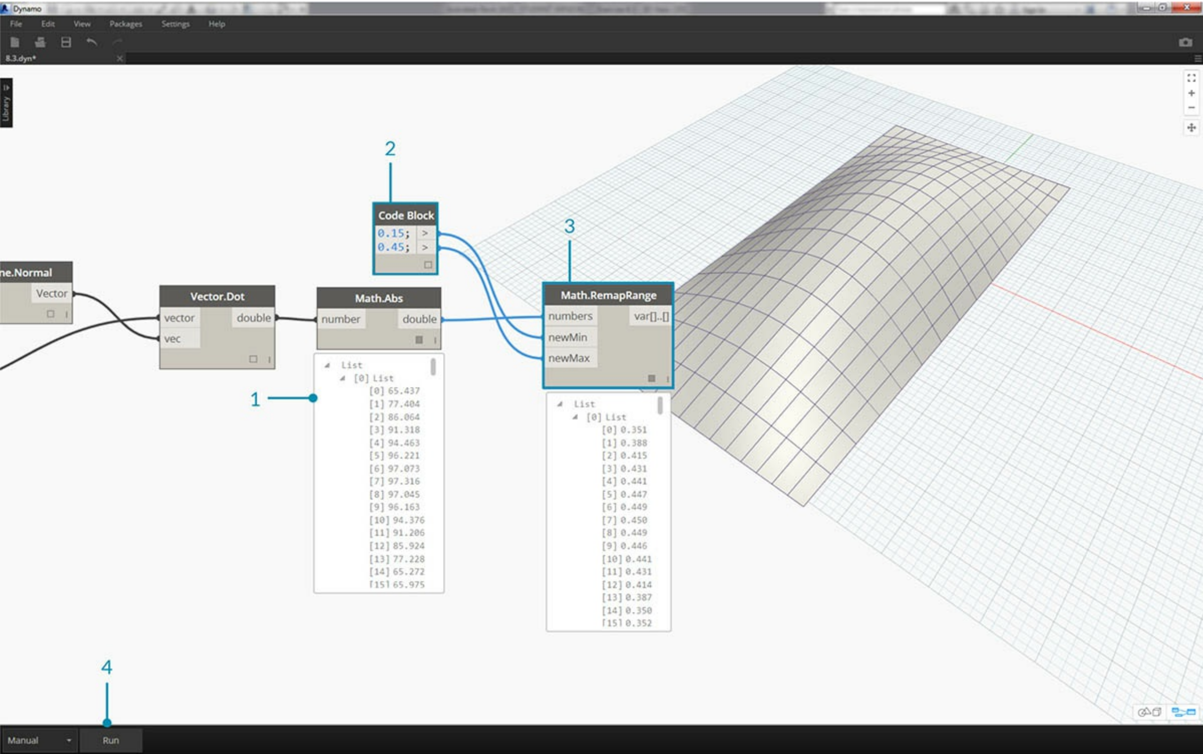
1. Используя результаты, полученные на предыдущем шаге, настройте апертуру каждой панели с учетом инсоляции. Увеличьте масштаб в Revit и выберите одну панель, чтобы увидеть на панели свойств параметр *коэффициента апертуры*. В настройках семейства задан диапазон апертуры от *0,05* до *0,45*.



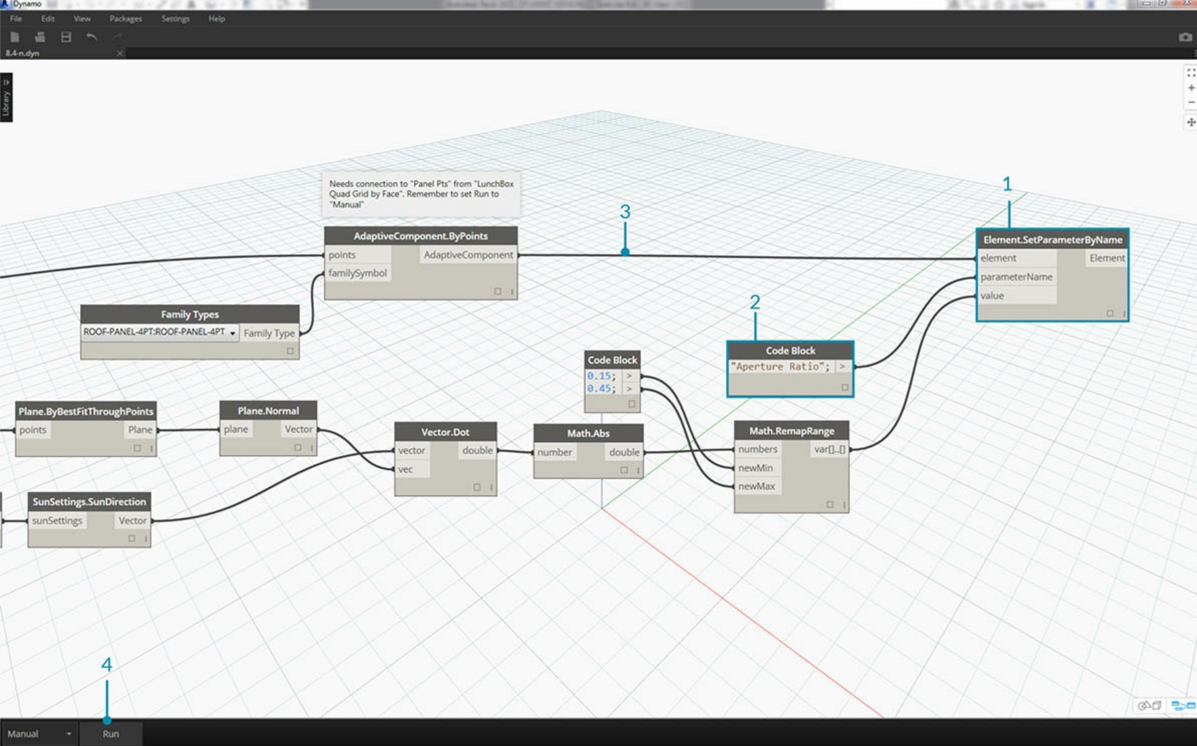
1. Если включить траекторию солнца, можно увидеть текущее положение солнца в Revit.



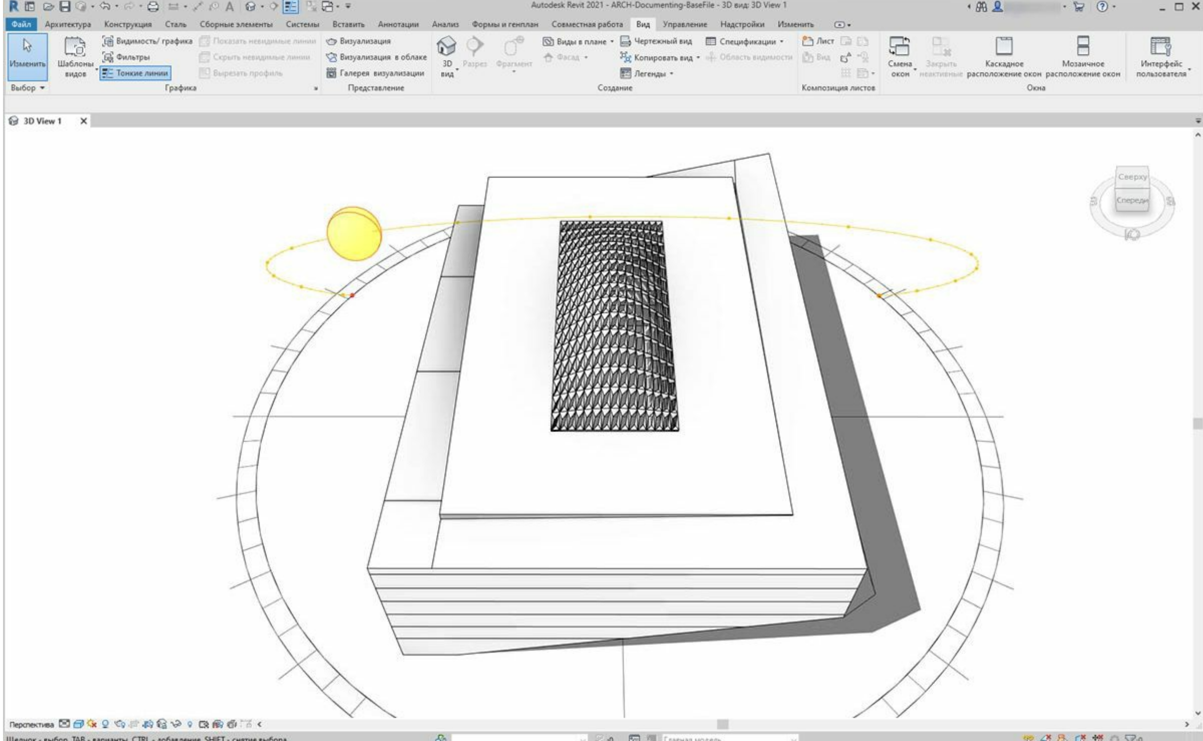
1. С помощью узла *SunSettings.Current* можно задать положение солнца в качестве опорного значения.
2. Соедините узел SunSettings с узлом *Sunsetting.SunDirection*, чтобы получить вектор солнца.
3. На основе значения, полученного из порта вывода *Panel Pts*, использованного для создания адаптивных компонентов, выполните аппроксимацию плоскости для компонента с помощью узла *Plane.ByBestFitThroughPoints*.
4. Запросите *нормаль* этой плоскости.
5. Для расчета ориентации инсоляции используйте *скалярное произведение*. Скалярное произведение — это формула, которая определяет, насколько параллельны или не параллельны два вектора. Чтобы приблизительно смоделировать ориентацию инсоляции, вы сравниваете нормаль плоскости каждого адаптивного компонента с вектором солнца.
6. Извлеките *абсолютное значение* результата. Это гарантирует точность скалярного произведения, если нормаль плоскости повернута в обратном направлении.
7. Нажмите кнопку *Запуск*.



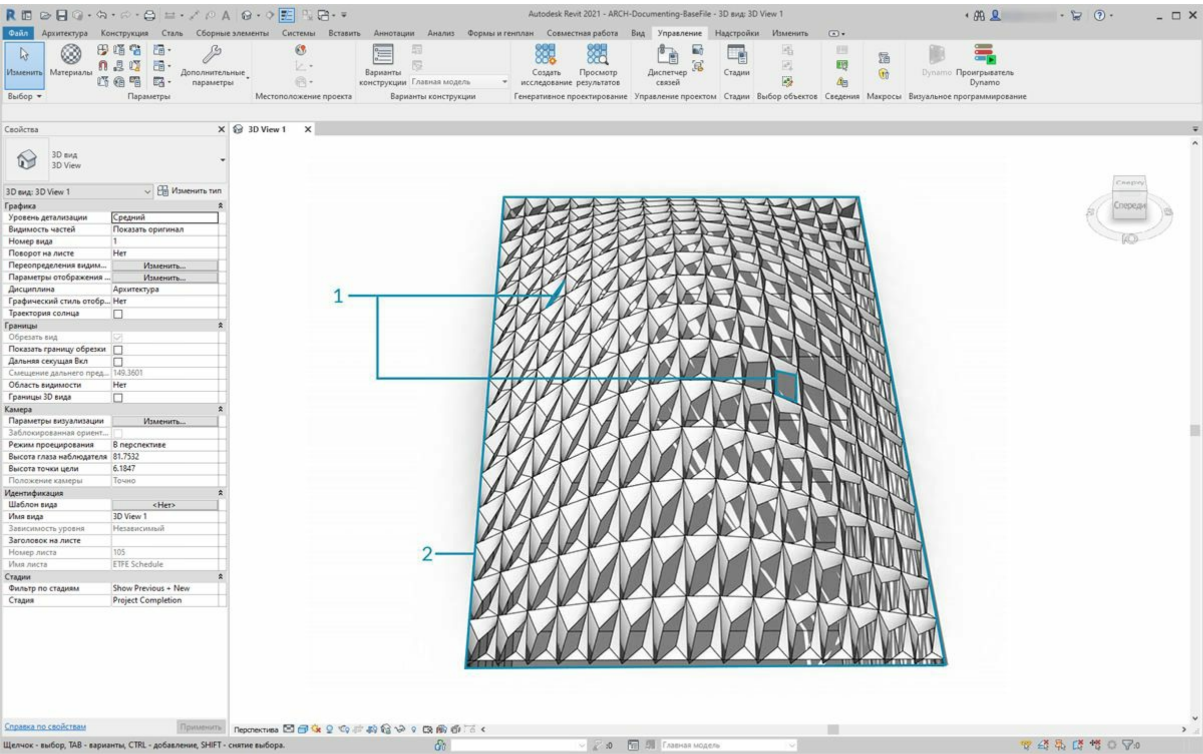
1. *Скалярное произведение* предоставляет широкий диапазон чисел. Нужно использовать их относительное распределение, однако необходимо объединить числа в соответствующем диапазоне редактируемого параметра *коэффициента апертуры*.
2. Узел *Math.RemapRange* отлично для этого подходит. Он сопоставляет пределы списка входных данных с двумя целевыми значениями.
3. Задайте целевые значения *0,15* и *0,45*** с помощью узла Code Block.
4. Нажмите кнопку *Запуск*.



1. Соедините сопоставленные значения с узлом *Element.SetParameterByName*.
2. Соедините строку *Aperture Ratio* с портом ввода *parameterName*.
3. Соедините элемент *адаптивных компонентов* с портом ввода *element*.
4. Нажмите кнопку *Запуск*.



Уменьшив масштаб в Revit, можно изучить влияние ориентации солнца на апертуру панелей ЭТФЭ.



При увеличении изображения видно, что панели ЭТФЭ более закрыты по направлению солнца. Цель данного упражнения — уменьшить перегрев из-за инсоляции. Если требуется увеличить проникновение света в помещение на основании инсоляции, просто смените область на *Math.RemapRange*.

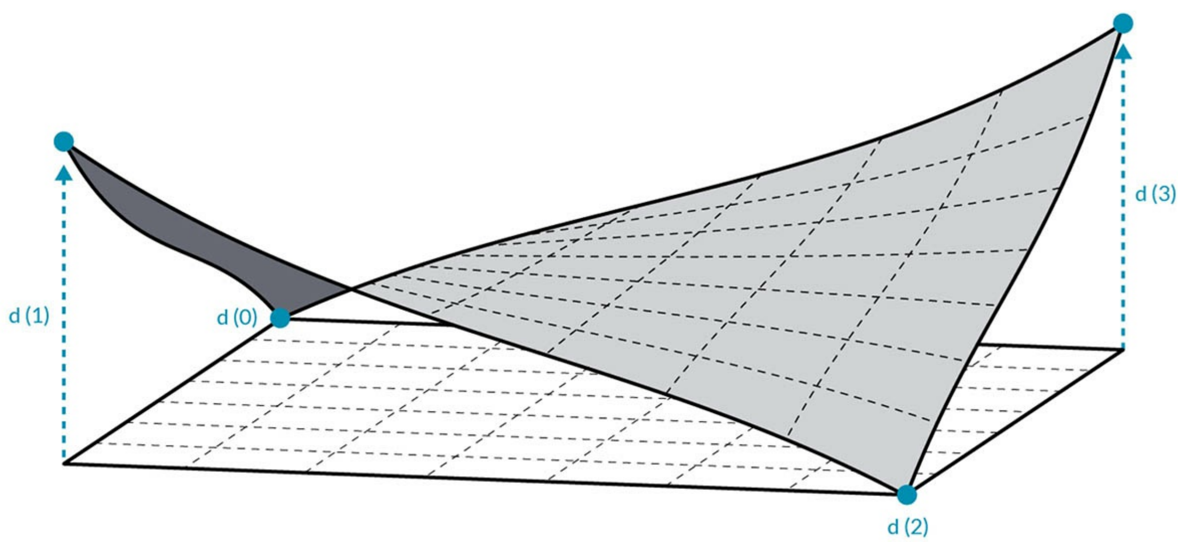
Выпуск документации

Выпуск документации

Параметры выпуска документации редактируются по тому же принципу, что и в предыдущих разделах. В этом разделе мы рассмотрим редактирование параметров, которые не влияют на геометрические свойства элементов, но необходимы для подготовки файла Revit к выпуску документации.

Отклонение

В упражнении ниже, чтобы создать лист Revit для документации, будет использоваться стандартный узел Deviation from Plane (отклонение от плоскости). Каждая панель в конструкции крыши, определенной параметрически, имеет собственное значение отклонения. Необходимо указать диапазон значений, используя цвет и составив спецификацию адаптивных точек, для передачи консультанту, инженеру или подрядчику, работающим с фасадом.



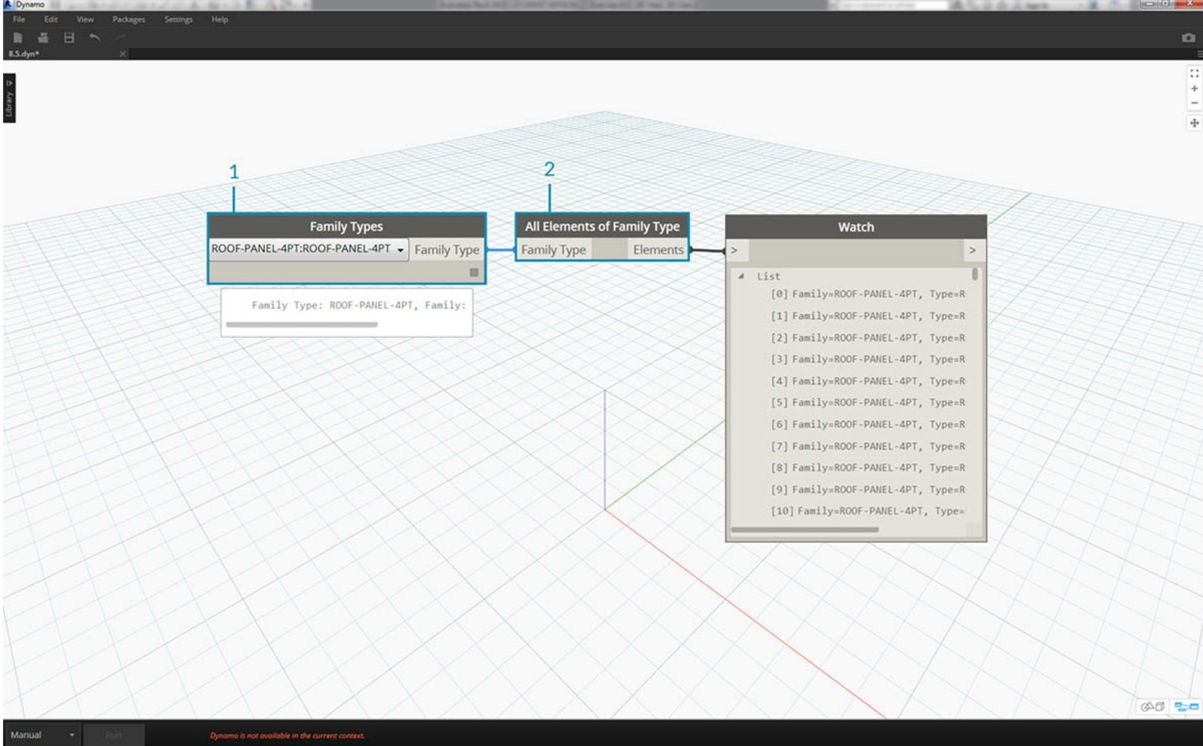
Узел Deviation from Plane вычисляет расстояние, на которое отклоняется набор из четырех точек относительно оптимально вписанной между ними плоскости. Это простой и быстрый способ проверки технологичности конструкции.

Упражнение

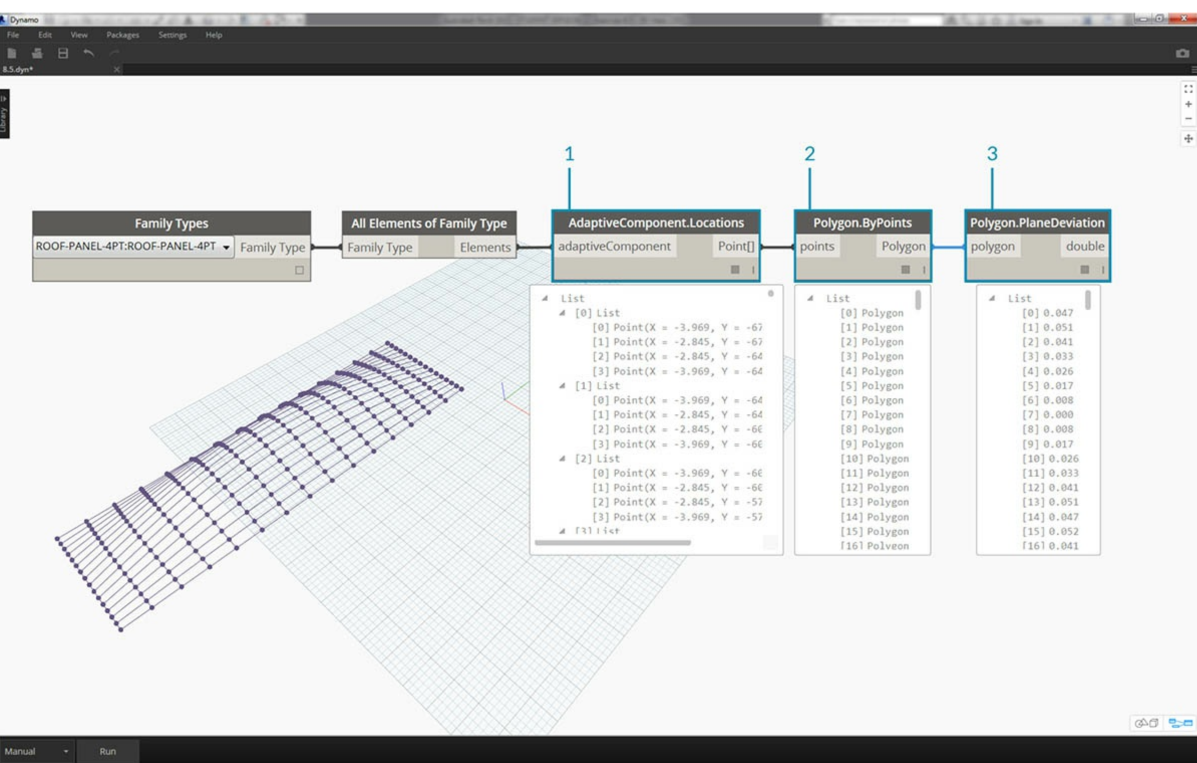
Скачайте файлы примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении.

1. Documenting.dyn
2. ARCH-Documenting-BaseFile.rvt

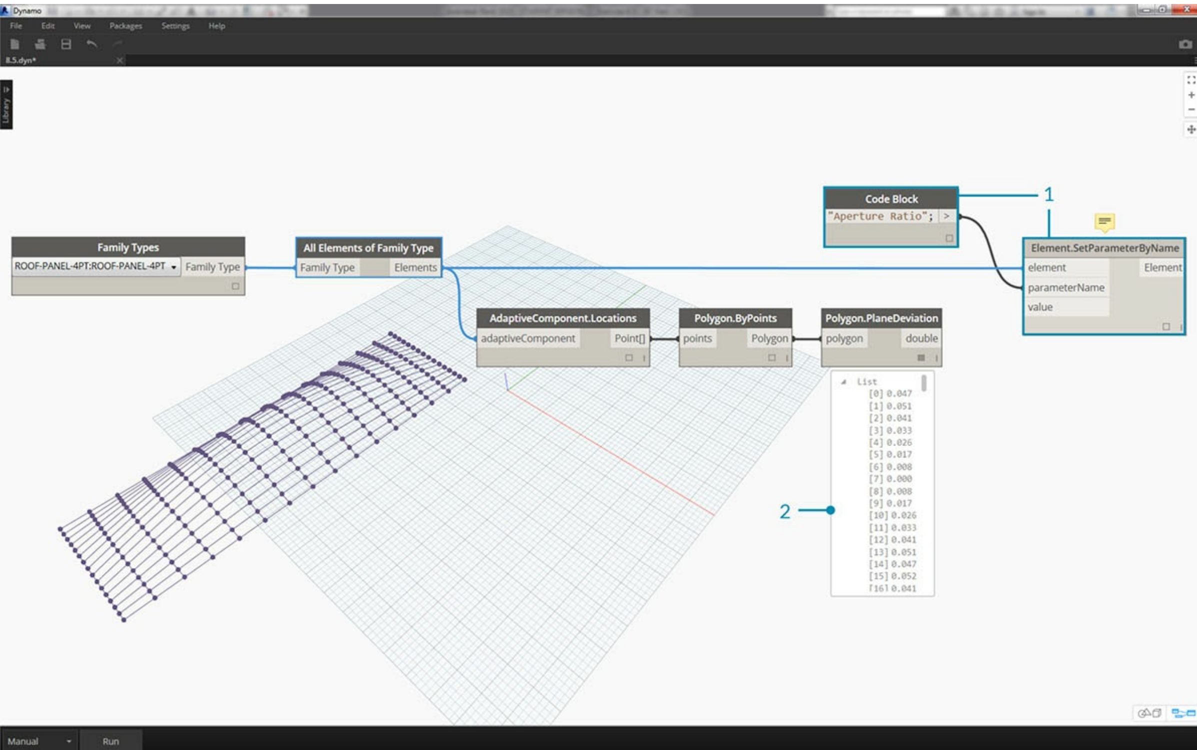
Начнем с файла Revit (можно также продолжить работу с файлом из предыдущего раздела). В этом файле представлен массив панелей ETFE на крыше. Мы будем использовать эти панели в ходе выполнения упражнения.



1. Добавьте узел *Family Types* в рабочую область и выберите *ROOF-PANEL-4PT*.
2. Соедините этот узел с узлом *All Elements of Family Type*, чтобы перенести все элементы из Revit в Dynamo.

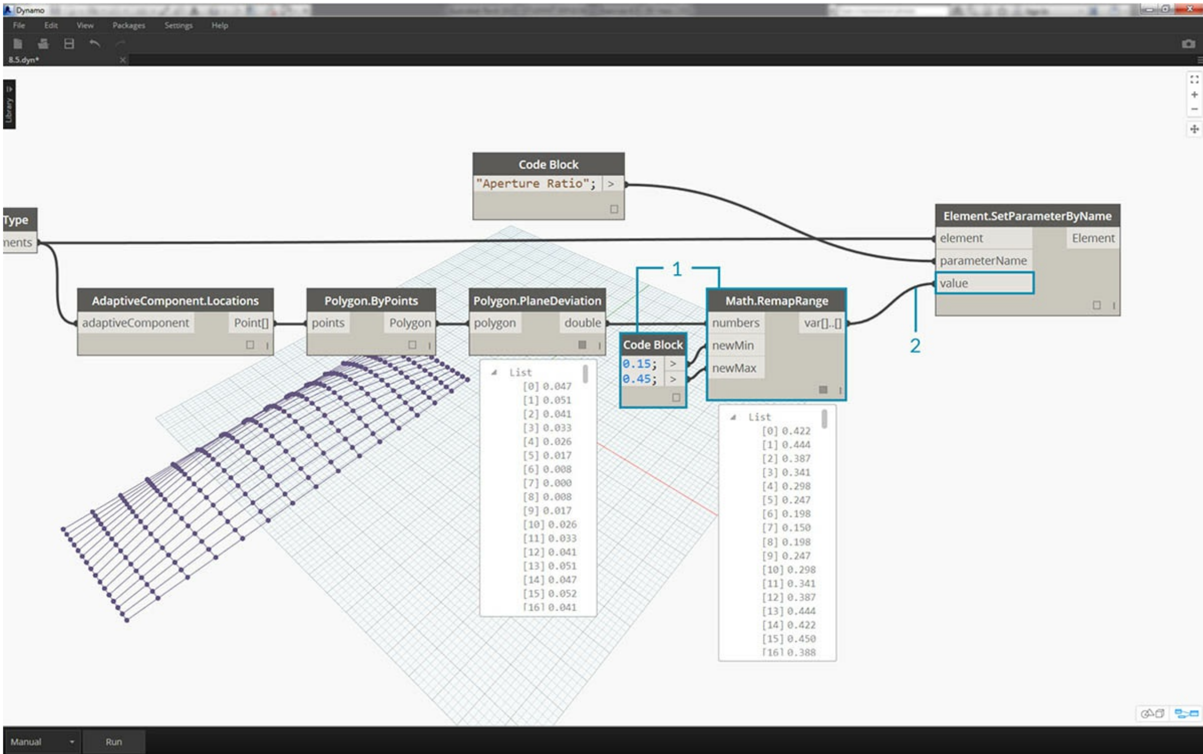


1. Запросите местоположение адаптивных точек каждого элемента с помощью узла *AdaptiveComponent.Locations*.
2. Создайте полигон по этим четырем точкам с помощью узла *Polygon.ByPoints*. Обратите внимание, что в Dynamo используется абстрактная версия системы панелей (без необходимости импорта всей геометрии элемента Revit).
3. Вычислите отклонение от плоскости с помощью узла *Polygon.PlaneDeviation*.

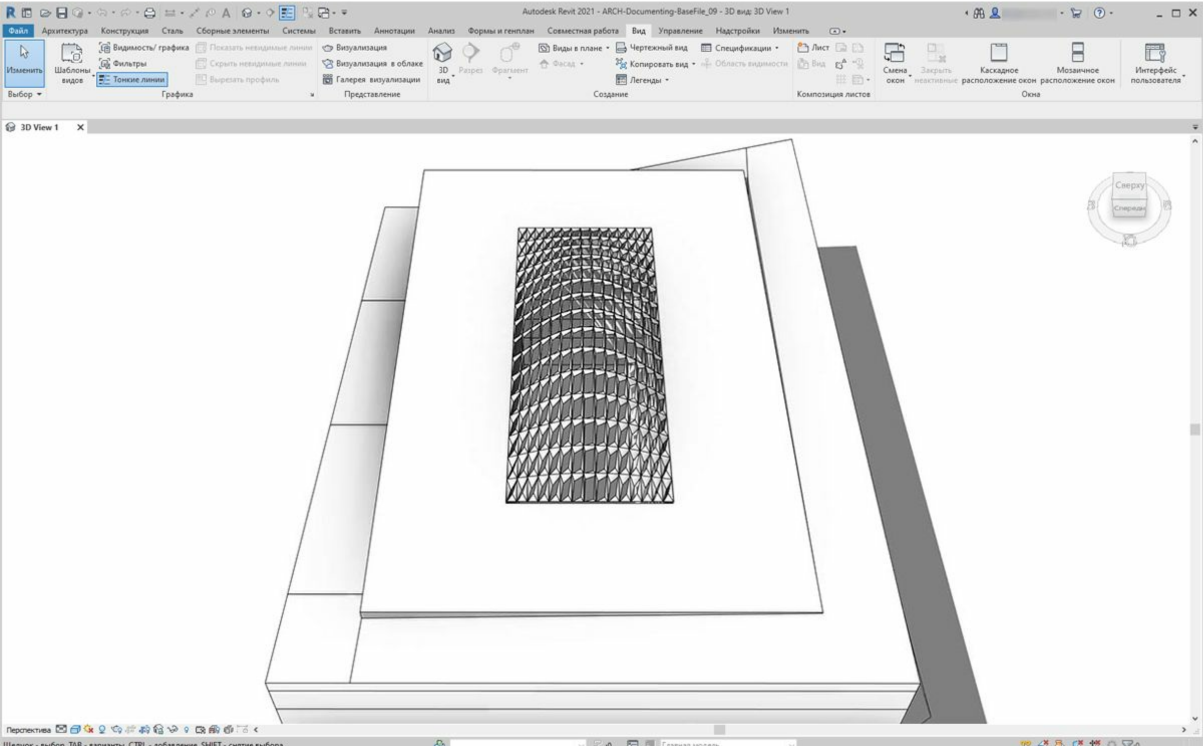


Просто в качестве дополнения, как и в предыдущем упражнении, укажем *коэффициент апертуры* каждой панели на основе отклонения от плоскости.

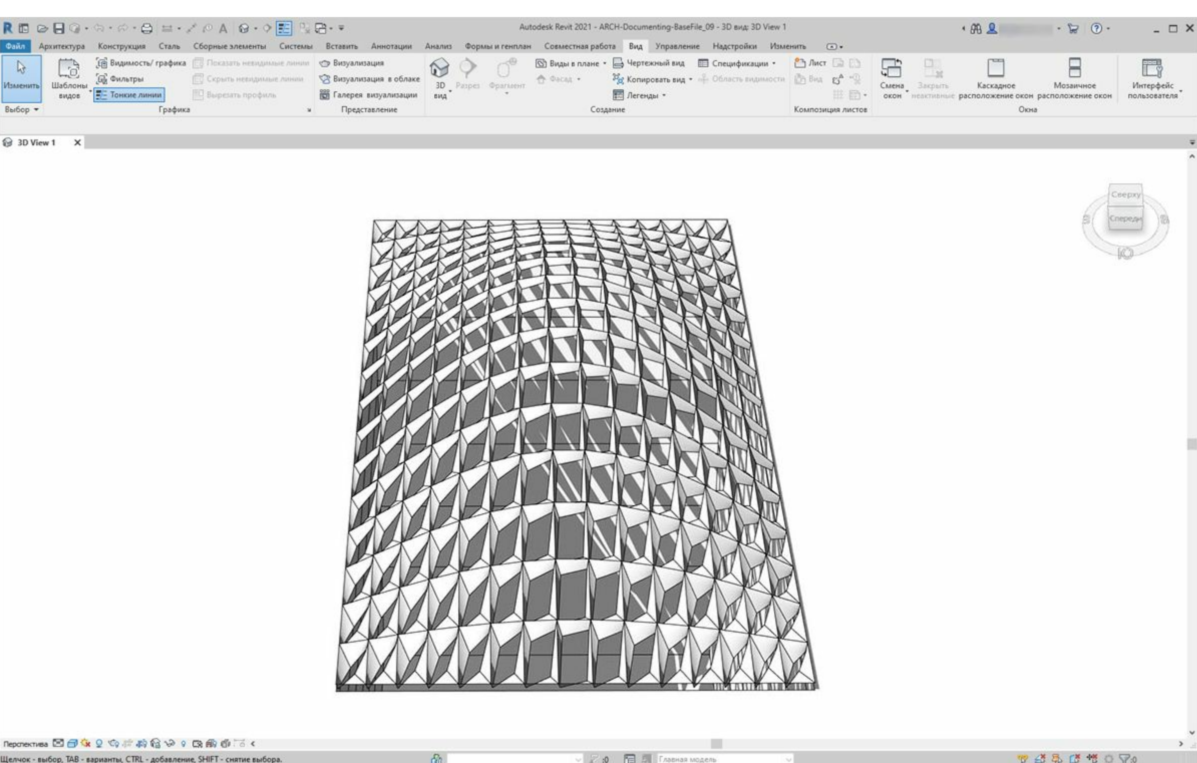
1. Добавьте узел *Element.SetParameterByName* в активное окно и соедините адаптивные компоненты с входным параметром *element*. Соедините *блок кода*, содержащий строку *Aperture Ratio*, с входным параметром *parameterName*.
2. Напрямую соединить результаты отклонения с входным параметром значения невозможно, так как необходимо перенастроить значения на другой диапазон параметров.



1. С помощью узла *Math.RemapRange* перенастройте значения отклонения на область между *.15* и *.45*.
2. Соедините эти результаты с входным параметром value узла *Element.SetParameterByName*.



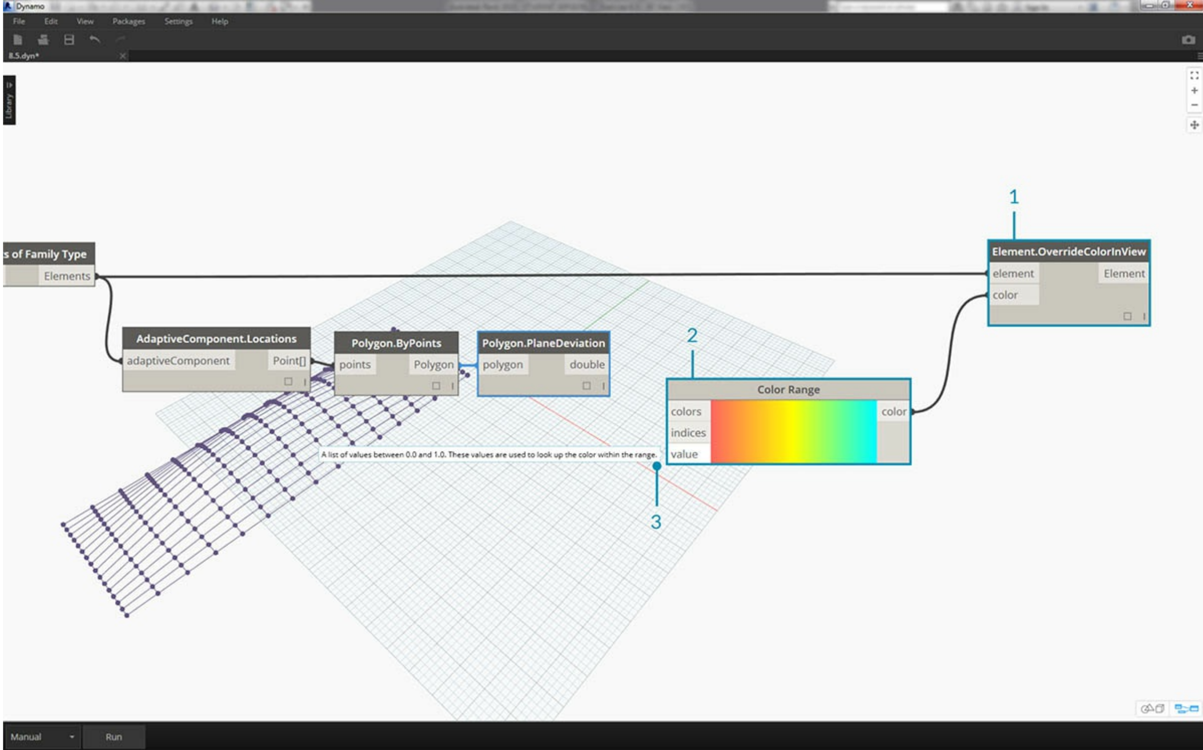
Вернувшись в Revit, можно *примерно* оценить изменение в апертуре на поверхности.



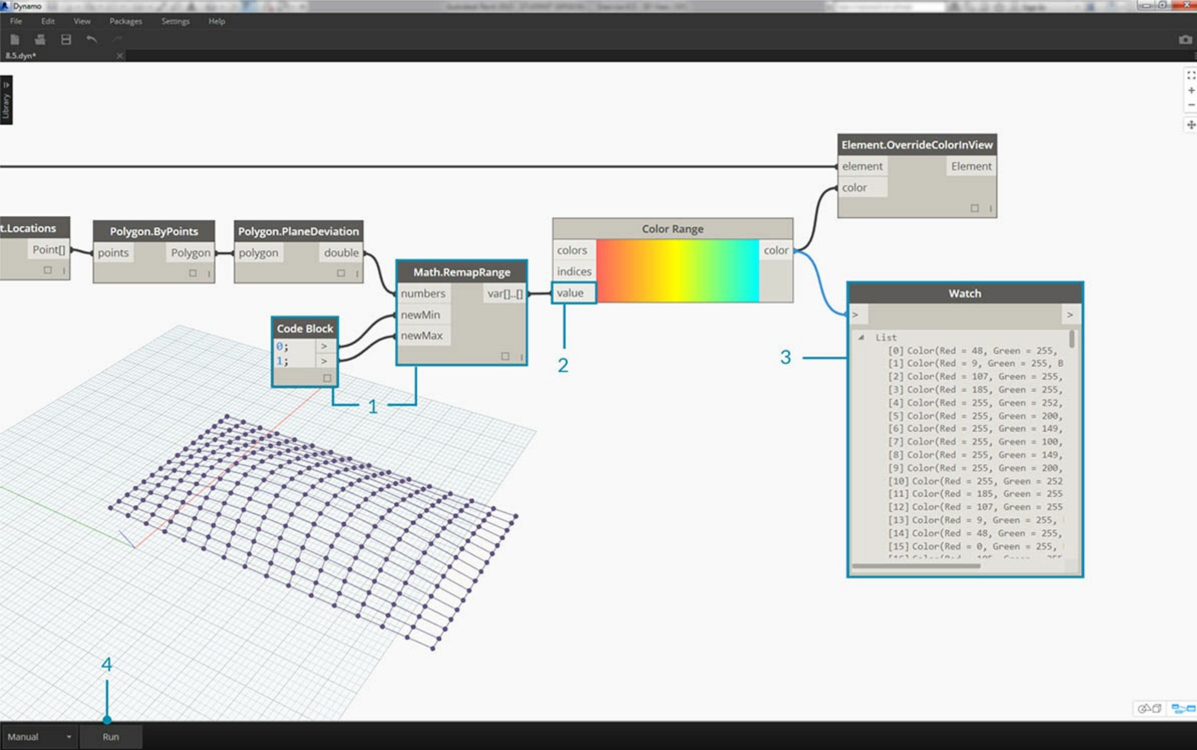
При увеличении масштаба становится понятно, что вес замкнутых панелей направлен к углам поверхности. Незамкнутые углы направлены вверх. Углы представляют собой участки более значительных отклонений, в то время как выпуклость имеет минимальную кривизну, что вполне оправдано.

Цвет и документация

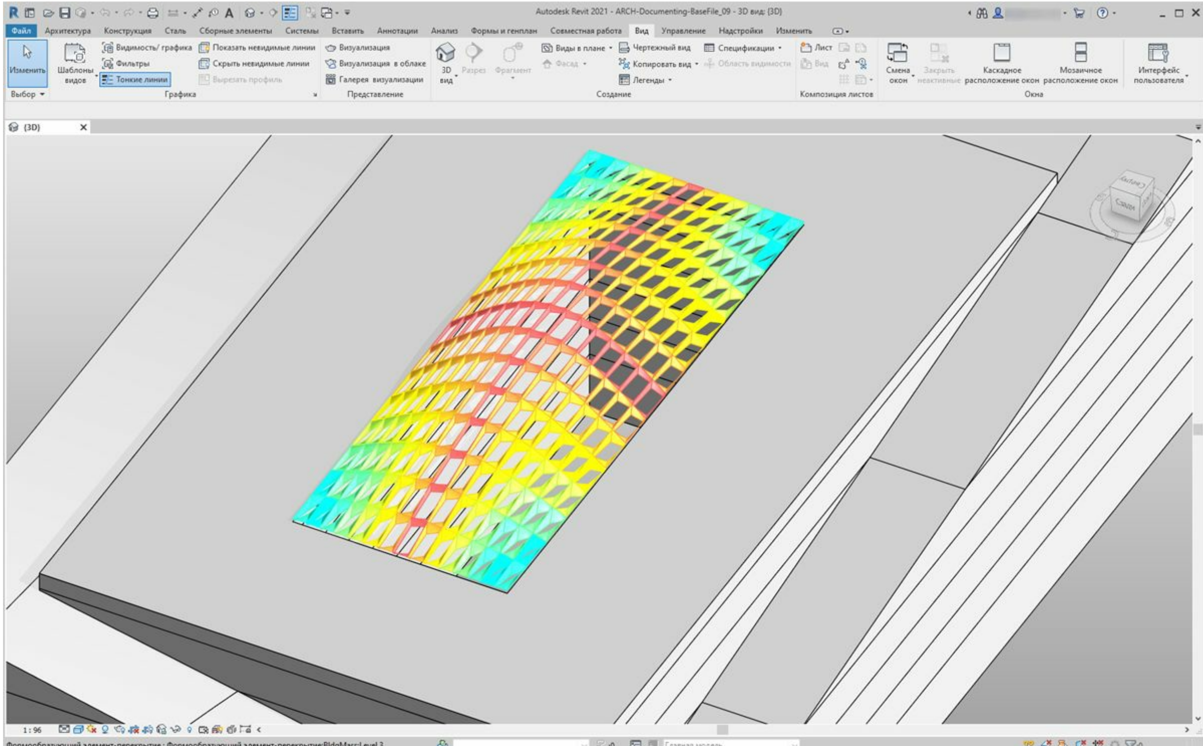
Указание коэффициента апертуры не дает четкой картины отклонения панелей на крыше. Кроме того, изменяется геометрия самого элемента. Предположим, нужно просто изучить отклонение с точки зрения технической осуществимости изготовления. Для этого в документации можно окрасить панели в различные цвета в зависимости от диапазона отклонений. Это можно сделать при помощи следующих шагов, которые очень похожи на действия, описанные выше.



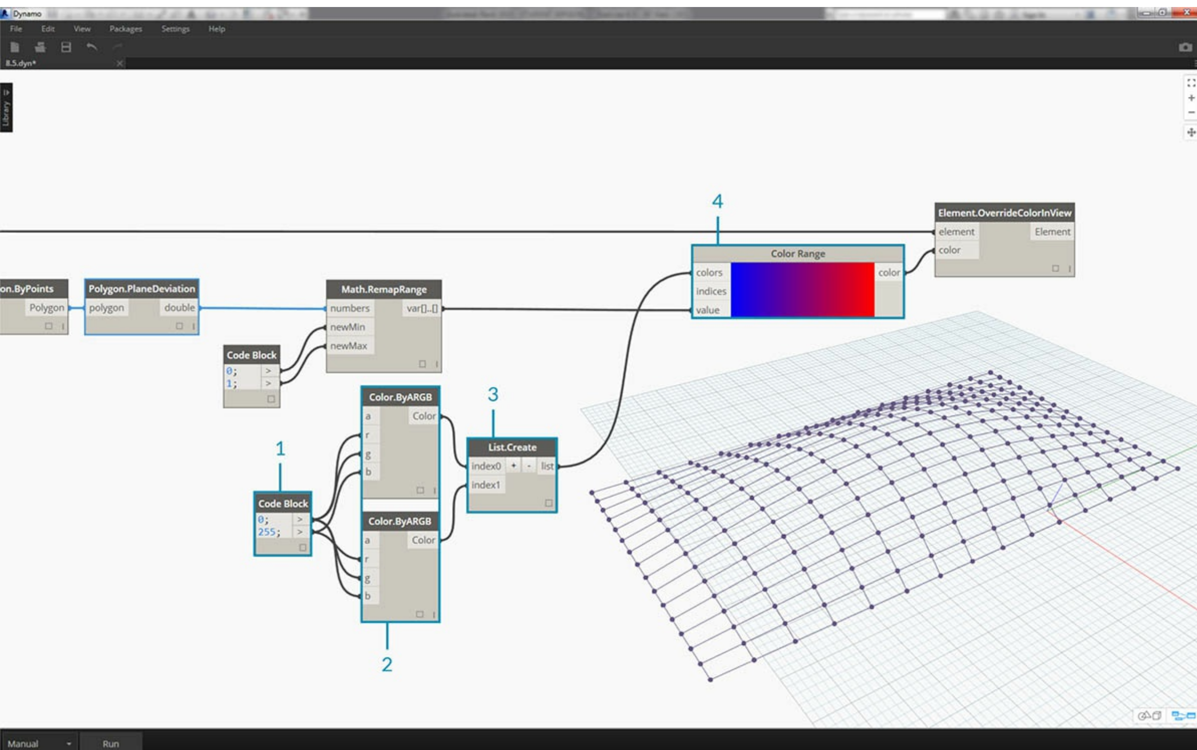
1. Удалите узлы *Element.SetParameterByName* и добавьте узел *Element.OverrideColorInView*.
2. Добавьте узел *Color Range* в рабочую область и соедините с входным параметром color узла *Element.OverrideColorInView*. Для создания градиента необходимо также соединить значения отклонения с цветовым диапазоном.
3. При наведении курсора на входной параметр *value* видно, что значения входного параметра должны быть в диапазоне от *0* до *1*. Только в этом случае можно сопоставить цвета со значениями. Необходимо перенастроить значения отклонения, задав этот диапазон.



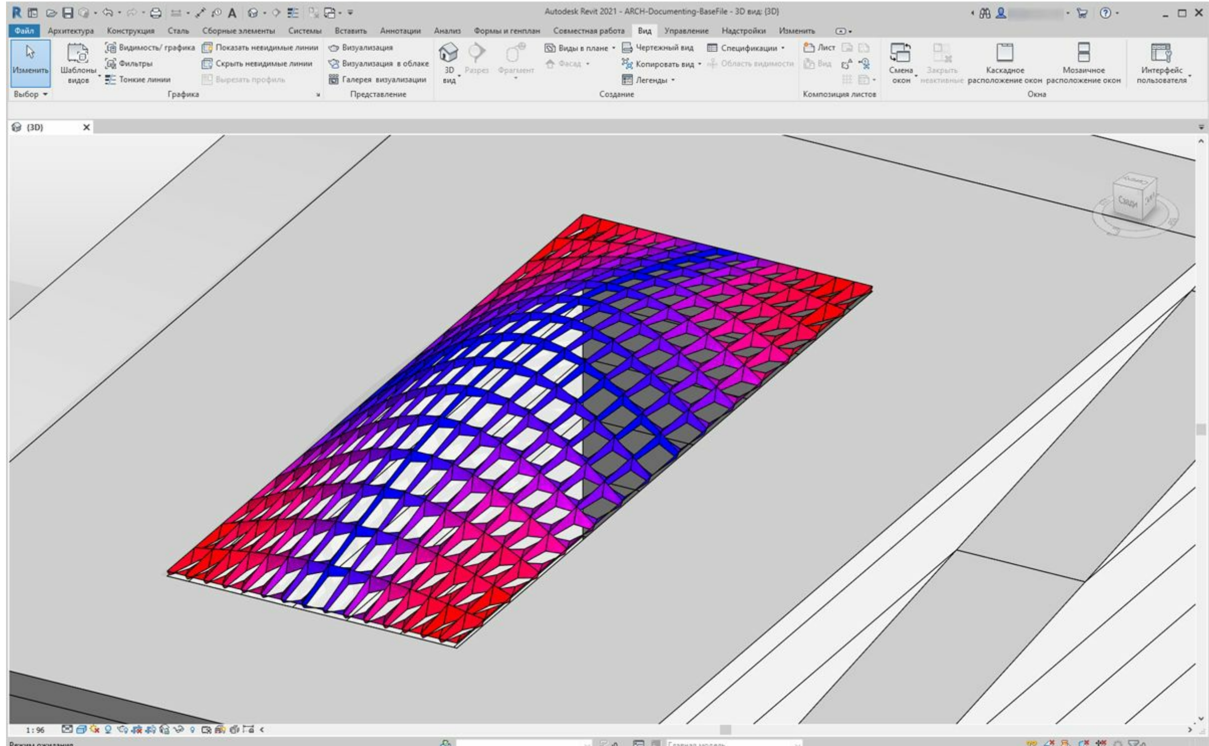
1. С помощью узла *Math.RemapRange* перенастройте значения отклонения от плоскости, задав диапазон от *0* до *1*. (Примечание. Чтобы задать исходную область, можно также использовать узел *MapTo*).
2. Соедините результаты с узлом *Color Range*.
3. Обратите внимание, что в результате получается диапазон цветов, а не диапазон чисел.
4. Если используется режим «Вручную», нажмите кнопку *Запуск*. С этого момента необходимо избегать работы в автоматическом режиме.



Вернувшись в Revit, видим значительно более наглядный градиент, характеризующий отклонение от плоскости на основе цветового диапазона. Но что, если требуется изменить цвета? Обратите внимание, что минимальные значения отклонения обозначены красным цветом, что не вполне логично. Необходимо, чтобы максимальному отклонению соответствовал красный, а минимальному — более спокойный цвет. Вернемся в Dynamo и устраним этот недостаток.

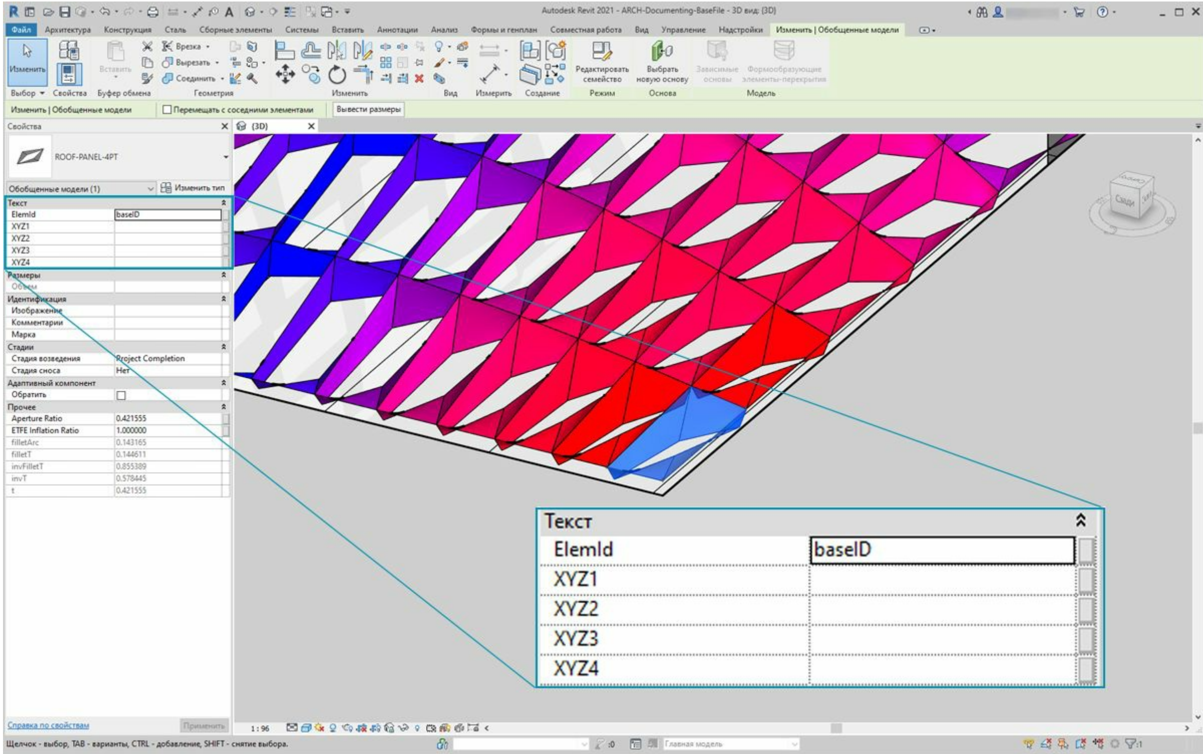


1. Используя *блок кода*, добавьте два числа в две разные строки: `0;` и `255;`.
2. Создайте красный и синий цвета, соединив соответствующие значения с двумя узлами *Color.ByARGB*.
3. Создайте список из этих двух цветов.
4. Соедините этот список с входным параметром *colors* узла *Color Range*, наблюдая за изменением пользовательского цветового диапазона.

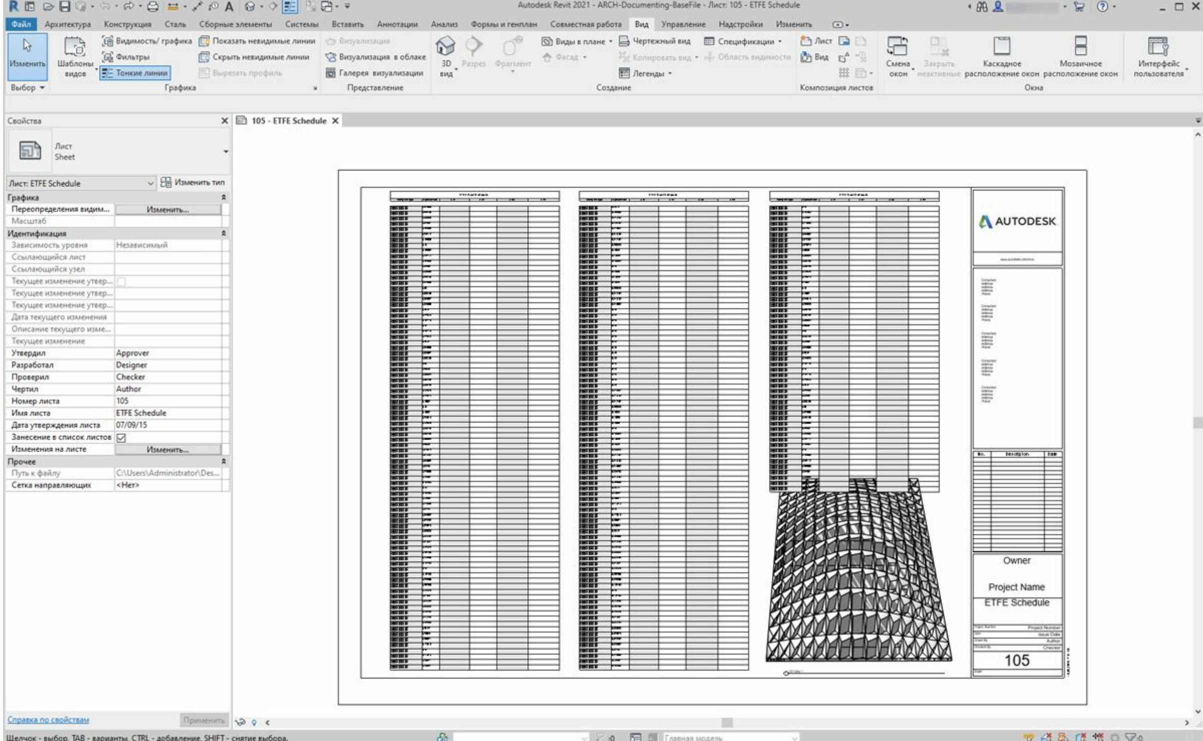


Вернувшись в Revit, теперь можно получить более наглядное представление о максимальном отклонении в углах. Помните, что этот узел служит для переопределения цвета на виде, поэтому он может оказаться действительно полезным, если в наборе чертежей определенный лист предназначен для определенного типа расчета.

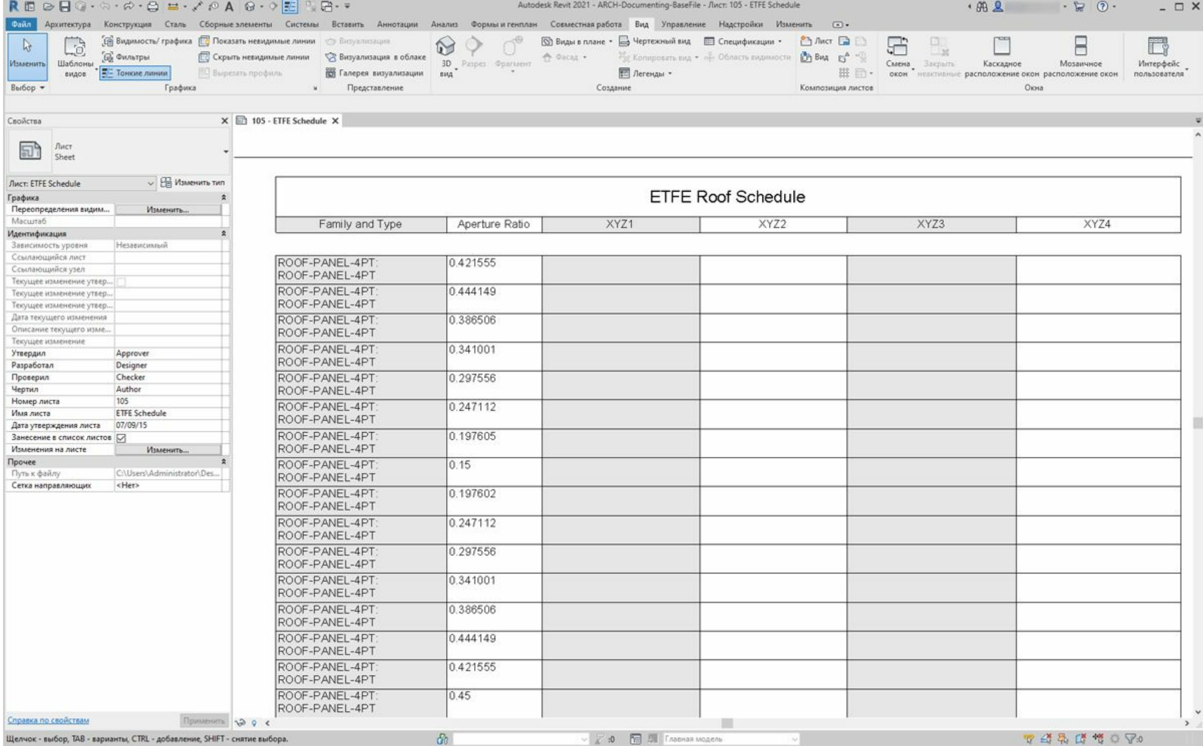
Создание спецификаций



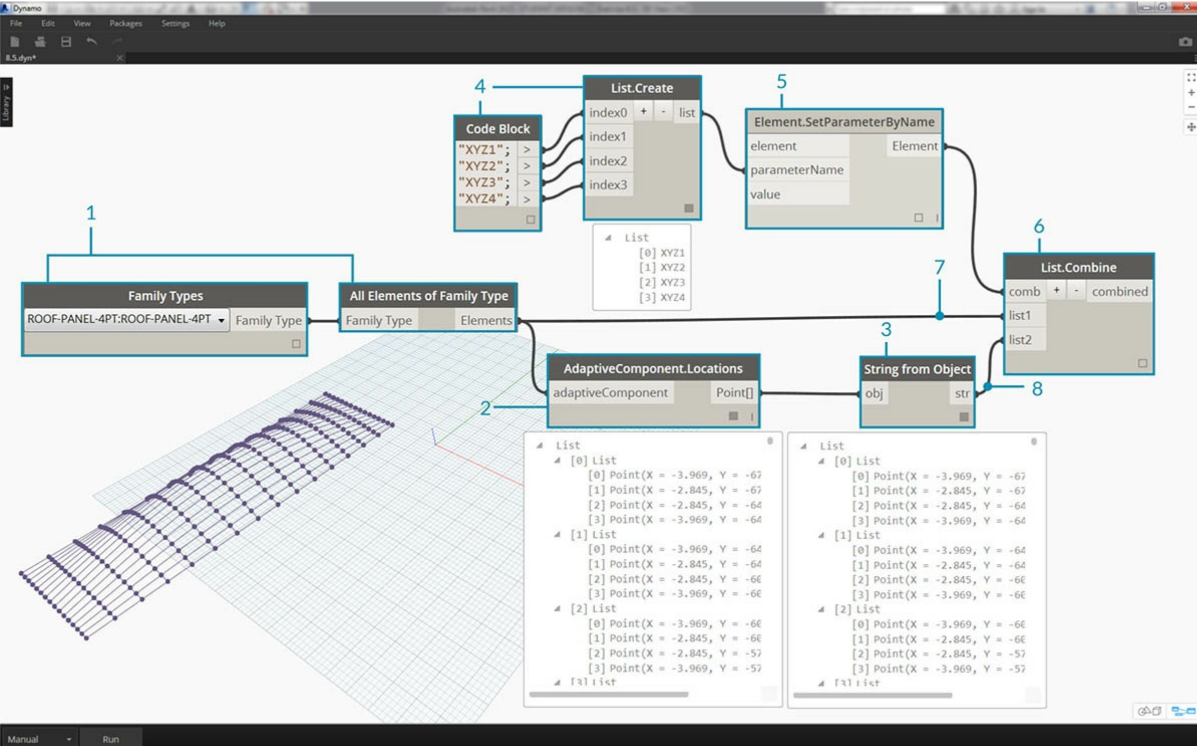
1. При выборе в Revit одной панели ETFE отображаются четыре параметра экземпляра: *XYZ1*, *XYZ2*, *XYZ3* и *XYZ4*. После создания все они будут пустыми. Это текстовые параметры, для которых требуется задать значения. С помощью Dynamo создадим местоположения адаптивных точек для каждого параметра. Это способствует взаимодействию, если необходимо отправить геометрический объект инженеру или консультанту по фасадам.



На образце листа представлена большая пустая спецификация. Параметры XYZ являются общедоступными параметрами в файле Revit, что позволяет добавить их в спецификацию.

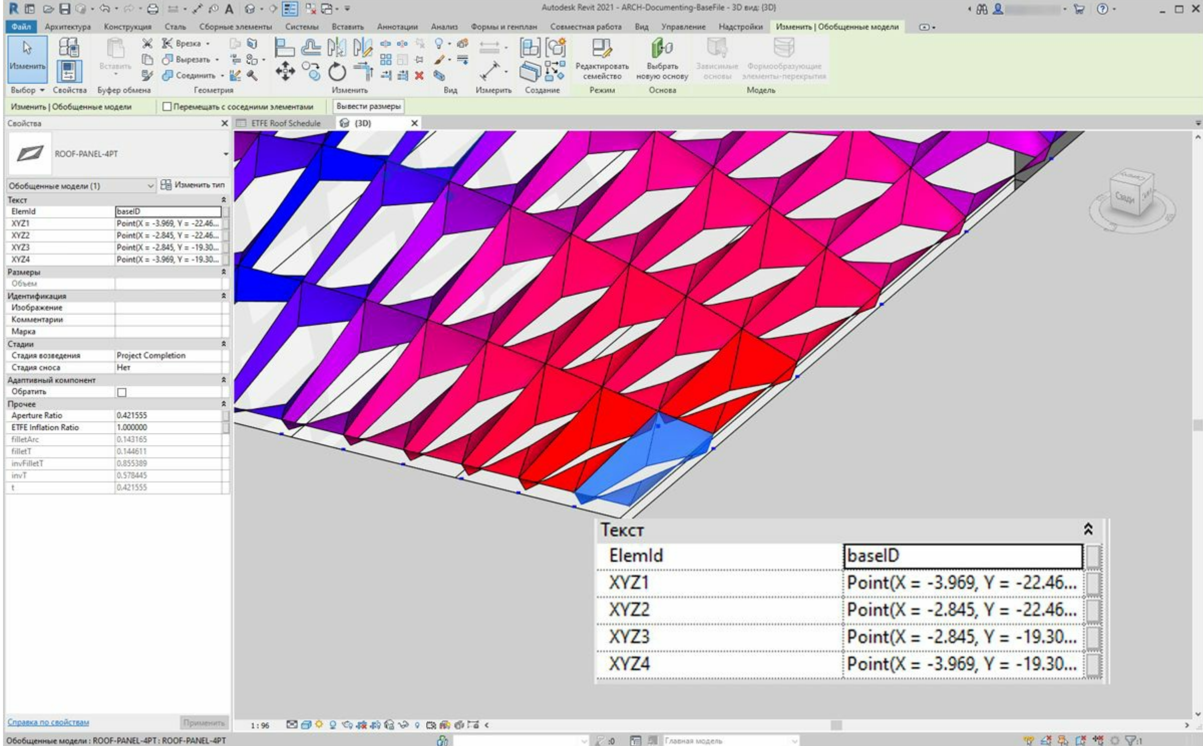


Если увеличить масштаб, видно, что параметры XYZ еще не заполнены. Первые два столбца автоматически заполняются из Revit.

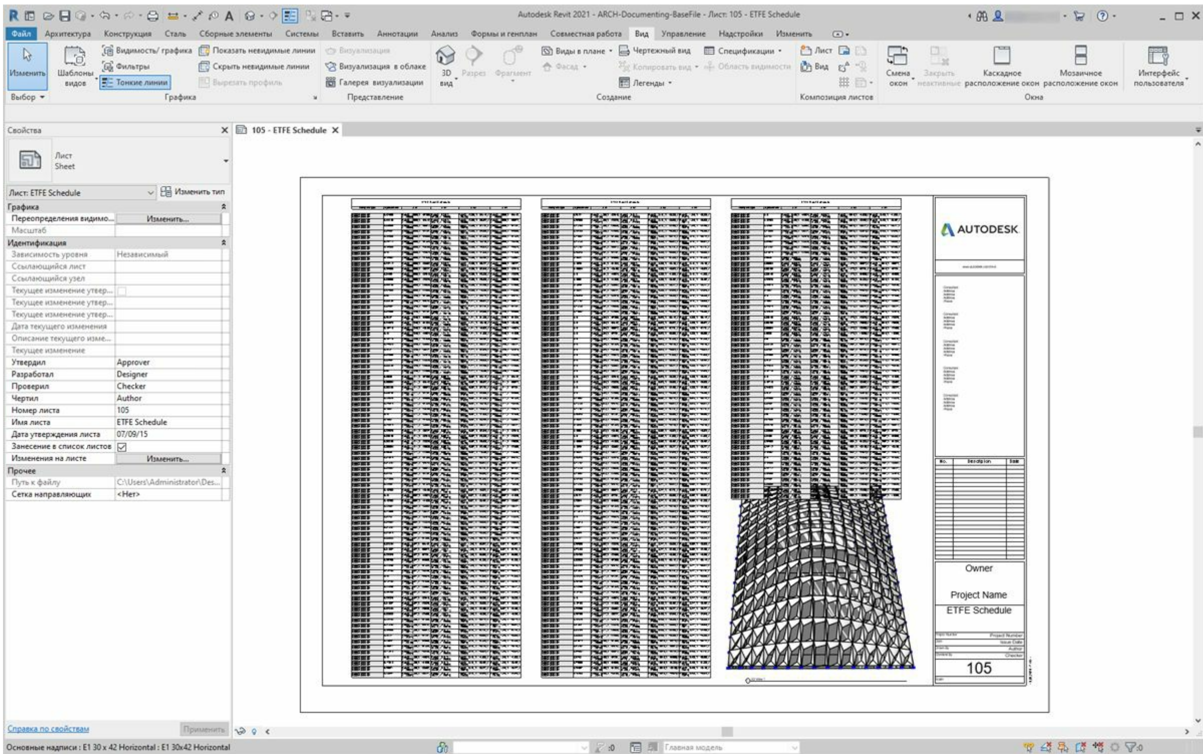


Для ввода этих значений выполним сложную операцию со списком. Сам график довольно прост, но его логика строится на основе сопоставления списков, которое рассматривалось в разделе, посвященном спискам.

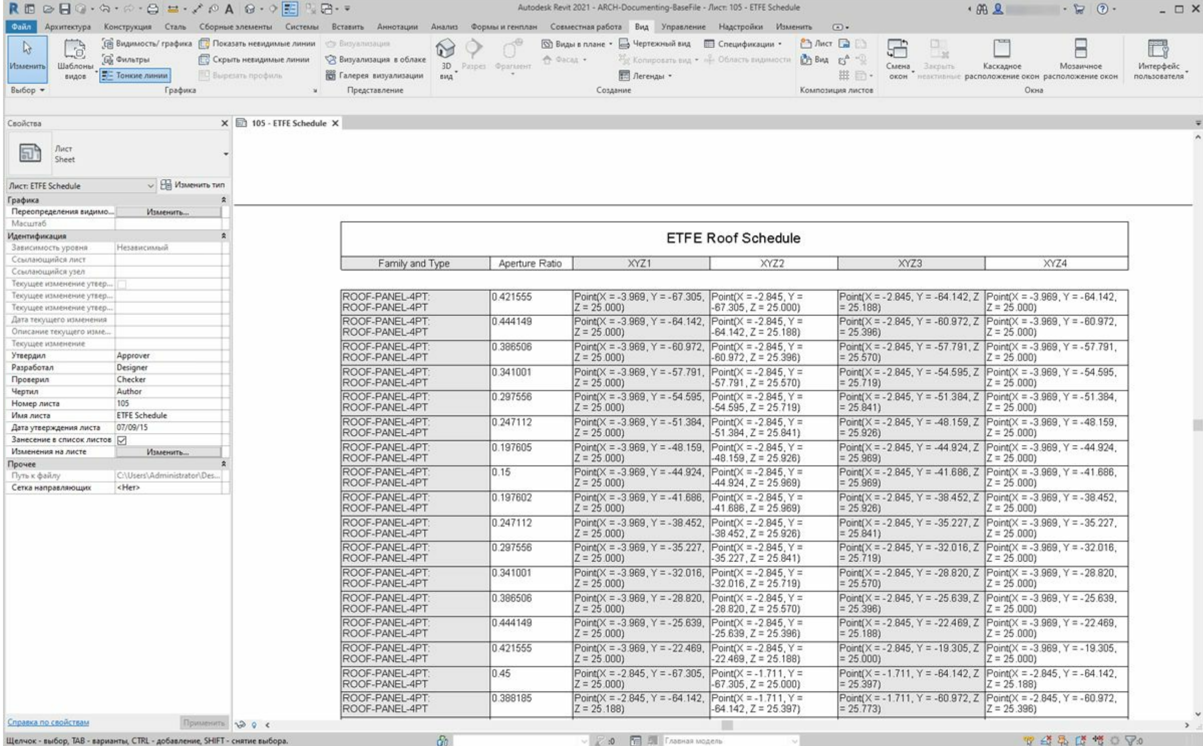
1. Выберите все адаптивные компоненты с двумя узлами.
2. Извлеките местоположение каждой точки с помощью узла *AdaptiveComponent.Locations*.
3. Преобразуйте эти точки в строки. Следует помнить, что параметр является текстовым, поэтому необходимо ввести правильный тип данных.
4. Создайте список из четырех строк, которые определяют изменяемые параметры: *XYZ1*, *XYZ2*, *XYZ3* и *XYZ4*.
5. Соедините этот список с входным параметром *parameterName* узла *Element.SetParameterByName*.
6. Соедините узел *Element.SetParameterByName* с входным параметром *combinator* узла *List.Combine.*
7. Соедините *адаптивные компоненты* с входным параметром *list1*.
8. Соедините узел *String from Object* с входным параметром *list2*.
9. В данном случае мы сопоставляем списки, так как вводим четыре значения для каждого элемента, что создает сложную структуру данных. Узел *List.Combine* управляет операцией, выполняемой на один шаг вниз по иерархии данных. Именно поэтому входные параметры element и value остаются пустыми. Узел *List.Combine* соединяет вложенные списки своих входных данных с пустыми входными параметрами узла *List.SetParameterByName* в зависимости от порядка их подсоединения.



При выборе панели в Revit видим, что для каждого параметра есть строковые значения. В реальном проекте для создания точки (X,Y,Z) использовался бы более простой формат. Это можно сделать с помощью строковых операций в Dynamo, но этот метод не рассматривается в рамках данного раздела.



Вид образца спецификации с заполненными параметрами.



ETFE Roof Schedule

Family and Type	Aperture Ratio	XYZ1	XYZ2	XYZ3	XYZ4
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.421555	Point(X = -3.969, Y = -67.305, Z = 25.000)	Point(X = -2.845, Y = -67.305, Z = 25.000)	Point(X = -2.845, Y = -64.142, Z = 25.188)	Point(X = -3.969, Y = -64.142, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.444149	Point(X = -3.969, Y = -64.142, Z = 25.000)	Point(X = -2.845, Y = -64.142, Z = 25.188)	Point(X = -2.845, Y = -60.972, Z = 25.396)	Point(X = -3.969, Y = -60.972, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.386506	Point(X = -3.969, Y = -60.972, Z = 25.000)	Point(X = -2.845, Y = -60.972, Z = 25.396)	Point(X = -2.845, Y = -57.791, Z = 25.570)	Point(X = -3.969, Y = -57.791, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.341001	Point(X = -3.969, Y = -57.791, Z = 25.000)	Point(X = -2.845, Y = -57.791, Z = 25.570)	Point(X = -2.845, Y = -54.595, Z = 25.719)	Point(X = -3.969, Y = -54.595, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.297556	Point(X = -3.969, Y = -54.595, Z = 25.000)	Point(X = -2.845, Y = -54.595, Z = 25.719)	Point(X = -2.845, Y = -51.384, Z = 25.841)	Point(X = -3.969, Y = -51.384, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.247112	Point(X = -3.969, Y = -51.384, Z = 25.000)	Point(X = -2.845, Y = -51.384, Z = 25.841)	Point(X = -2.845, Y = -48.159, Z = 25.926)	Point(X = -3.969, Y = -48.159, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.197605	Point(X = -3.969, Y = -48.159, Z = 25.000)	Point(X = -2.845, Y = -48.159, Z = 25.926)	Point(X = -2.845, Y = -44.924, Z = 25.969)	Point(X = -3.969, Y = -44.924, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.15	Point(X = -3.969, Y = -44.924, Z = 25.000)	Point(X = -2.845, Y = -44.924, Z = 25.969)	Point(X = -2.845, Y = -41.686, Z = 25.969)	Point(X = -3.969, Y = -41.686, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.197602	Point(X = -3.969, Y = -41.686, Z = 25.000)	Point(X = -2.845, Y = -41.686, Z = 25.969)	Point(X = -2.845, Y = -38.452, Z = 25.926)	Point(X = -3.969, Y = -38.452, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.247112	Point(X = -3.969, Y = -38.452, Z = 25.000)	Point(X = -2.845, Y = -38.452, Z = 25.926)	Point(X = -2.845, Y = -35.227, Z = 25.841)	Point(X = -3.969, Y = -35.227, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.297556	Point(X = -3.969, Y = -35.227, Z = 25.000)	Point(X = -2.845, Y = -35.227, Z = 25.841)	Point(X = -2.845, Y = -32.016, Z = 25.719)	Point(X = -3.969, Y = -32.016, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.341001	Point(X = -3.969, Y = -32.016, Z = 25.000)	Point(X = -2.845, Y = -32.016, Z = 25.719)	Point(X = -2.845, Y = -28.820, Z = 25.570)	Point(X = -3.969, Y = -28.820, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.386506	Point(X = -3.969, Y = -28.820, Z = 25.000)	Point(X = -2.845, Y = -28.820, Z = 25.570)	Point(X = -2.845, Y = -25.639, Z = 25.396)	Point(X = -3.969, Y = -25.639, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.444149	Point(X = -3.969, Y = -25.639, Z = 25.000)	Point(X = -2.845, Y = -25.639, Z = 25.396)	Point(X = -2.845, Y = -22.469, Z = 25.188)	Point(X = -3.969, Y = -22.469, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.421555	Point(X = -3.969, Y = -22.469, Z = 25.000)	Point(X = -2.845, Y = -22.469, Z = 25.188)	Point(X = -2.845, Y = -19.305, Z = 25.000)	Point(X = -3.969, Y = -19.305, Z = 25.000)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.45	Point(X = -2.845, Y = -67.305, Z = 25.000)	Point(X = -1.711, Y = -67.305, Z = 25.000)	Point(X = -1.711, Y = -64.142, Z = 25.397)	Point(X = -2.845, Y = -64.142, Z = 25.188)
ROOF-PANEL-4PT: ROOF-PANEL-4PT	0.388185	Point(X = -2.845, Y = -64.142, Z = 25.188)	Point(X = -1.711, Y = -64.142, Z = 25.397)	Point(X = -1.711, Y = -60.972, Z = 25.773)	Point(X = -2.845, Y = -60.972, Z = 25.396)

Каждая панель ETFE теперь имеет координаты XYZ, которые были созданы для каждой адаптивной точки, соответствующей углам изготавливаемой панели.

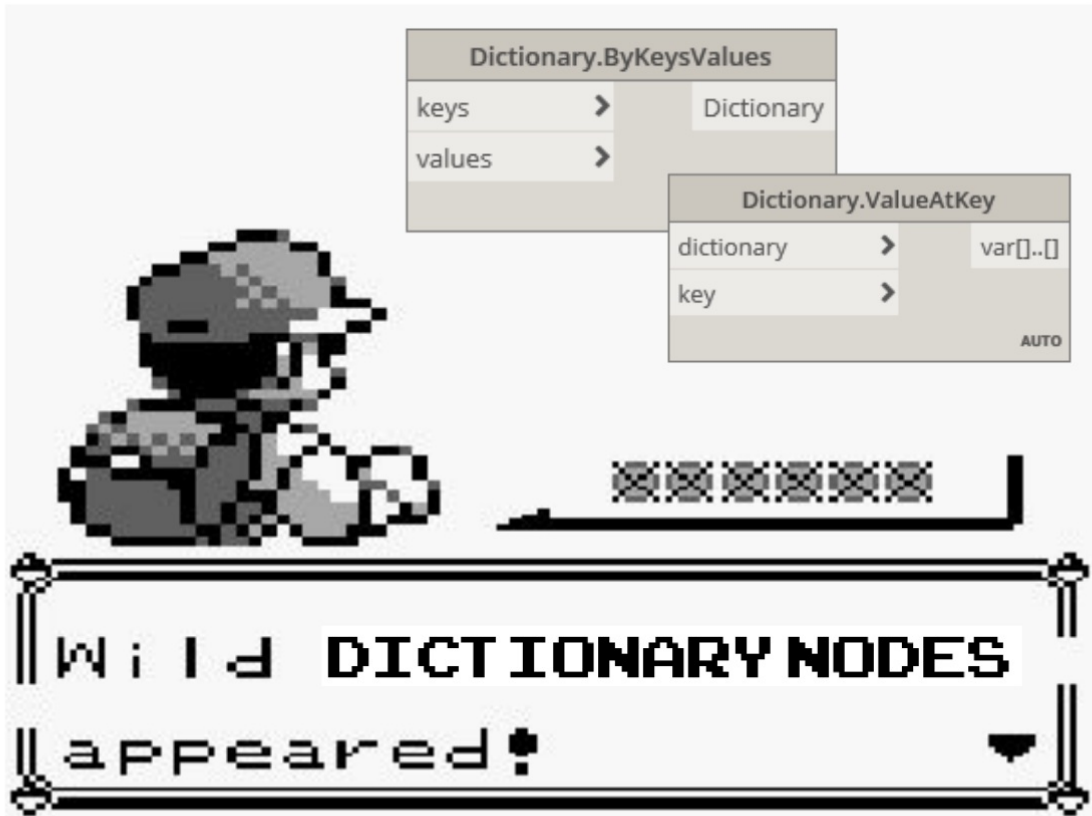
Словари в Dynamo

Словари в Dynamo

Словари представляют собой набор данных, который связан с другим объектом данных, известным как ключ. Словари позволяют выполнять поиск, удаление и вставку данных в коллекциях.

По сути, словарь является эффективным механизмом поиска данных.

Функции работы со словарями присутствовали во многих версиях Dynamo. В приложении Dynamo 2.0 представлен новый способ управления данными этого типа.



Изображение предоставлено [sixtysecondrevit.com]

Что такое словарь

Словари

Dynamo 2.0 включает новый тип данных — список, являющийся ответвлением типа данных словаря. Это нововведение может повлечь за собой существенные изменения в подходах к созданию и использованию данных в рамках рабочих процессов. До версии 2.0 словари и списки относились к одному типу данных. Если говорить коротко, то списки — это словари с целочисленными ключами.

- **Что такое словарь**

  Словарь — это тип данных, образуемый набором пар «ключ — значение». Ключ каждого набора является уникальным. Содержимое словаря не упорядочено. Поиск данных осуществляется с использованием ключа, а не по значению индекса, как в списке. *В Dynamo 2.0 поддерживаются только строковые ключи.*

- **Что такое список**

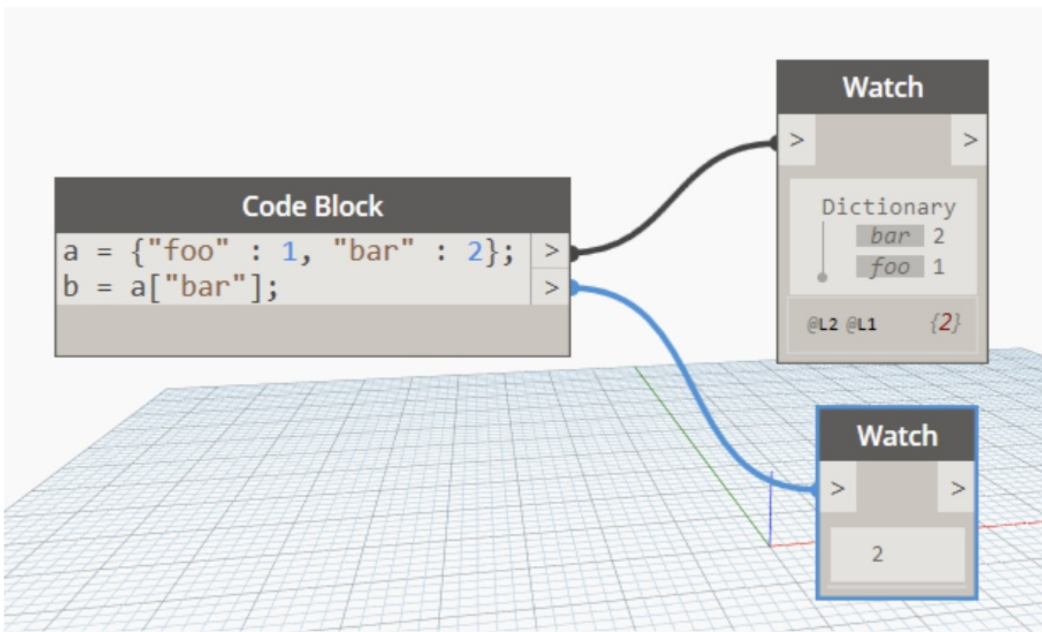
  Список — это тип данных, образуемый набором упорядоченных значений. В списках Dynamo в качестве индексов используются целочисленные значения.

- **Зачем были сделаны эти изменения и как это касается пользователей**

  В результате разделения словарей и списков словари получили своеобразный статус высшего класса: они позволяют легко и быстро хранить и находить значения, и для этого не нужно запоминать значения индекса или поддерживать строгую структуру списка на протяжении всего рабочего процесса. На этапе тестирования было выявлено значительное уменьшение размеров графиков при использовании словарей вместо нескольких узлов `GetItemAtIndex`.

- **В чем заключаются изменения**

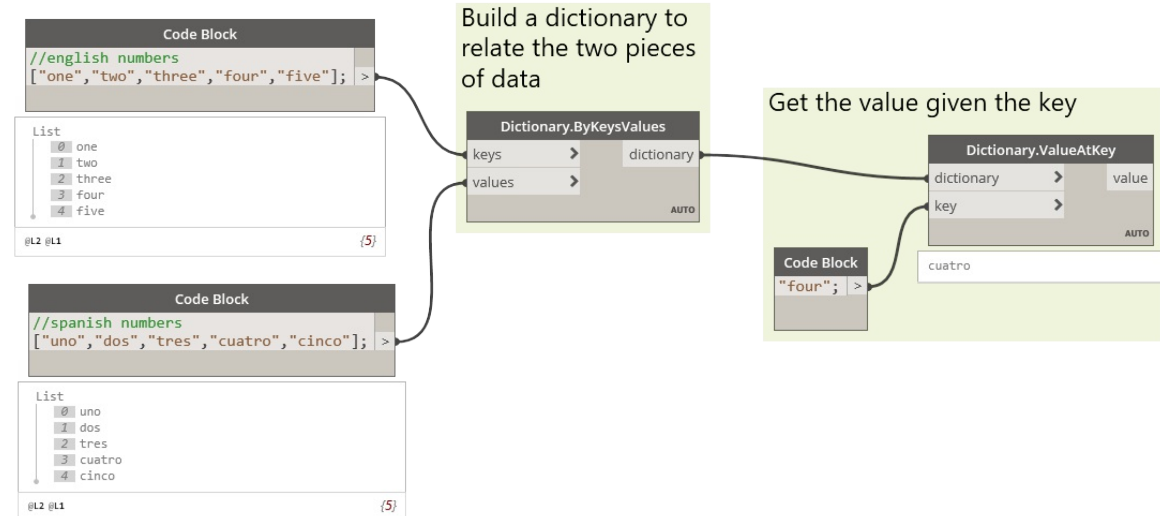
  - *Синтаксис*: обновления привели к изменениям в процессах запуска и использования словарей и списков в блоках кода.
    - В словарях используется следующий синтаксис: `{ключ:значение}`
    - В списках используется следующий синтаксис: `[значение,значение,значение]`
  - В библиотеку добавлены *новые узлы*, которые позволяют создавать, изменять и запрашивать словари.
  - Списки, созданные в блоках кода версии 1.x, при загрузке сценария автоматически обновляются до нового синтаксиса списка, в котором используются квадратные скобки `[ ]` вместо фигурных `{ }`.



- **Как эти изменения повлияют на пользователей и что они им дают**

  С точки зрения информатики как науки, словари, как и списки, являются наборами объектов. Элементы в списках хранятся с соблюдением определенного порядка. Содержимое словарей *не упорядочивается*. В словарях не используется последовательная нумерация (индексы). Вместо этого в них используются *ключи.*

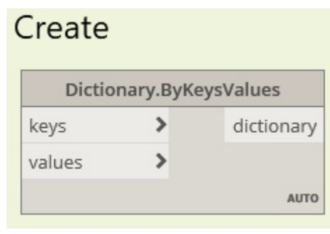
На изображении ниже представлен пример возможного использования словаря. Во многих случаях словари используются для соотнесения двух элементов данных, которые могут не иметь прямой корреляции. В нашем случае испанский перевод английского слова соотносится с его оригиналом для поиска в дальнейшем.

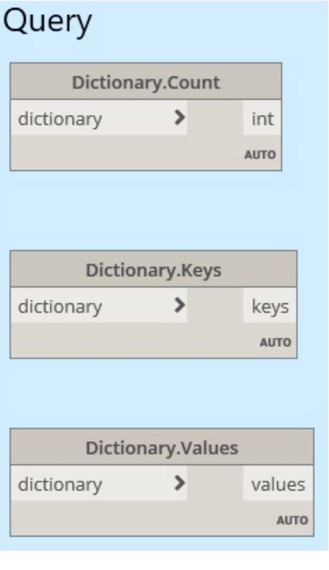

Code Block
//english numbers
["one","two","three","four","five"];
List
0 one
1 two
2 three
3 four
4 five
@L2 @L1
{5}
Code Block
//spanish numbers
["uno","dos","tres","cuatro","cinco"];
List
0 uno
1 dos
2 tres
3 cuatro
4 cinco
@L2 @L1
{5}
Build a dictionary to relate the two pieces of data
Dictionary.ByKeysValues
keys
values
dictionary
AUTO
Get the value given the key
Dictionary.ValueAtKey
dictionary
key
value
AUTO
Code Block
"four";
cuatro

Применение узлов

Узлы Dictionary

В Dynamo 2.0 доступны различные узлы Dictionary для работы со словарями. К ним относятся узлы *создания*, *действия и запроса*.





- С помощью метода `Dictionary.ByKeysValues` можно создать словарь с заданными значениями и ключами. *Количество записей определяется кратчайшим входным списком.*
- С помощью метода `Dictionary.Components` создаются компоненты входного словаря. *Это функция, противоположная функции узла создания.*
- С помощью метода `Dictionary.RemoveKeys` создается новый объект словаря с удаленными входными ключами.
- С помощью метода `Dictionary.SetValueAtKeys` создается словарь на основе входных ключей и значений, заменяющих текущее значение в соответствующих ключах.
- Метод `Dictionary.ValueAtKey` возвращает значение во входном ключе.
- С помощью метода `Dictionary.Count` можно узнать, сколько пар «ключ — значение» хранится в словаре.
- Метод `Dictionary.Keys` возвращает наименования ключей, хранящихся в настоящий момент в словаре.
- Метод `Dictionary.Values` возвращает значения, хранящиеся в настоящий момент в словаре.

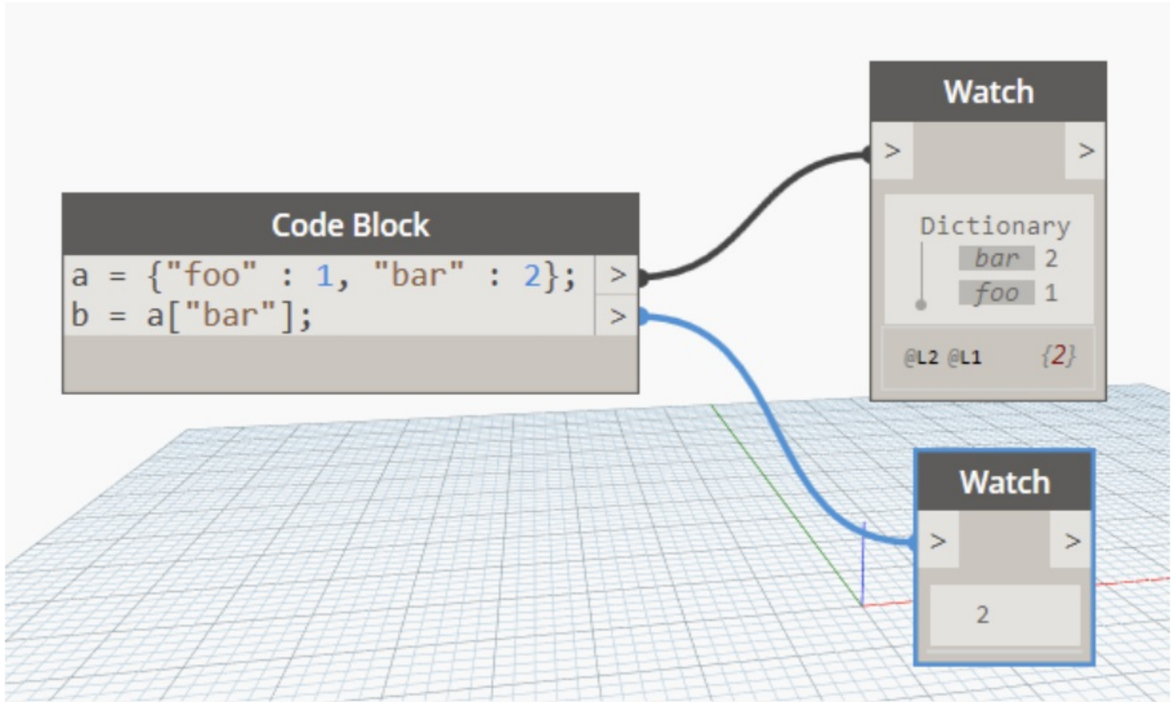
Соотнесение данных с помощью словарей — это отличная альтернатива традиционный методам работы с индексами и списками.

Применение узлов Code Block

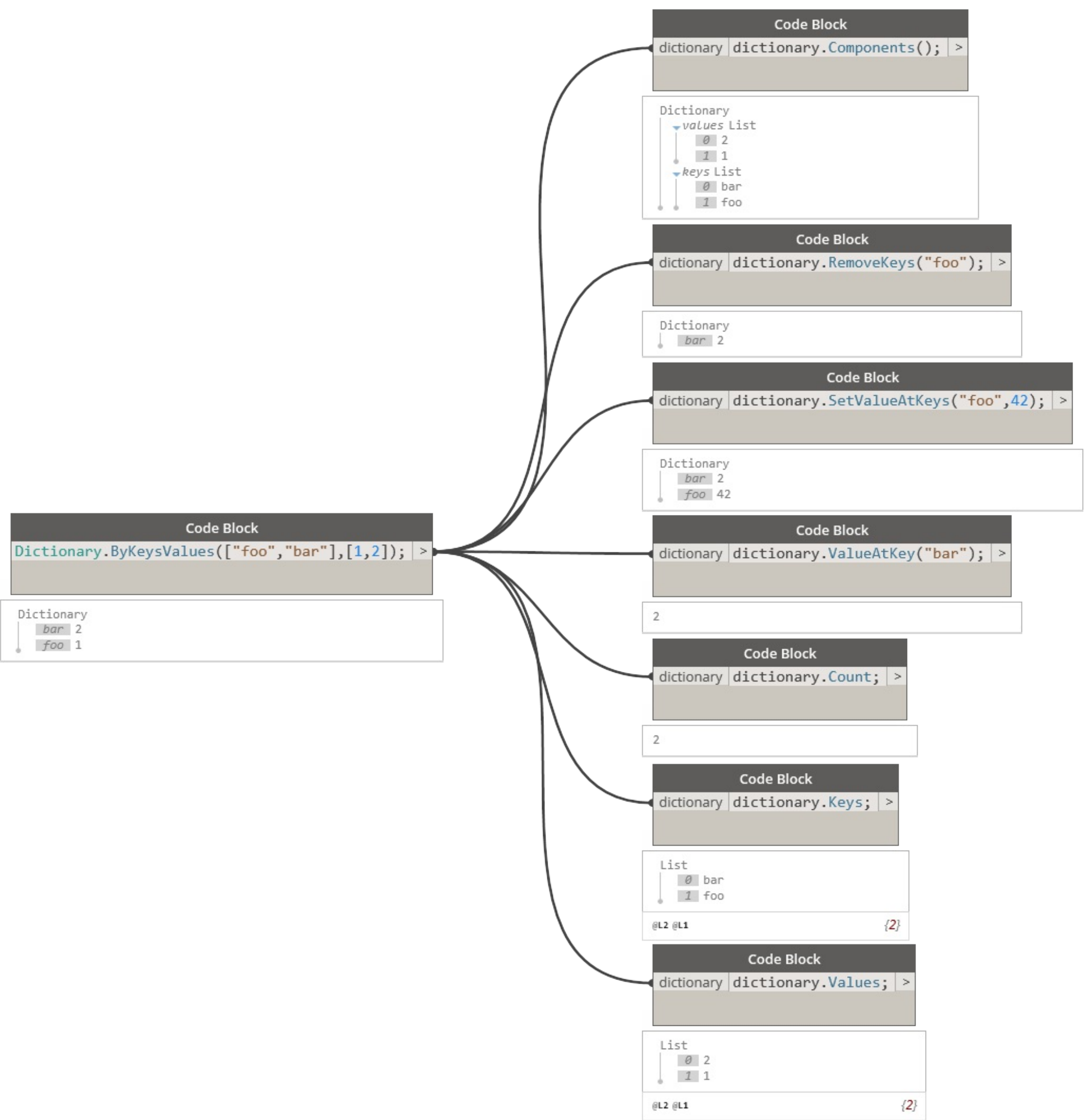
Словари в узлах Code Block

Dynamo 2.0 не только включает в себя узлы для работы со словарями, но и аналогичные новые функции в кода блоках.

Можно использовать как приведенный ниже синтаксис, так и представления узлов на основе DesignScript.



Поскольку словарь является типом объекта в Dynamo, с ним можно выполнять следующие действия.



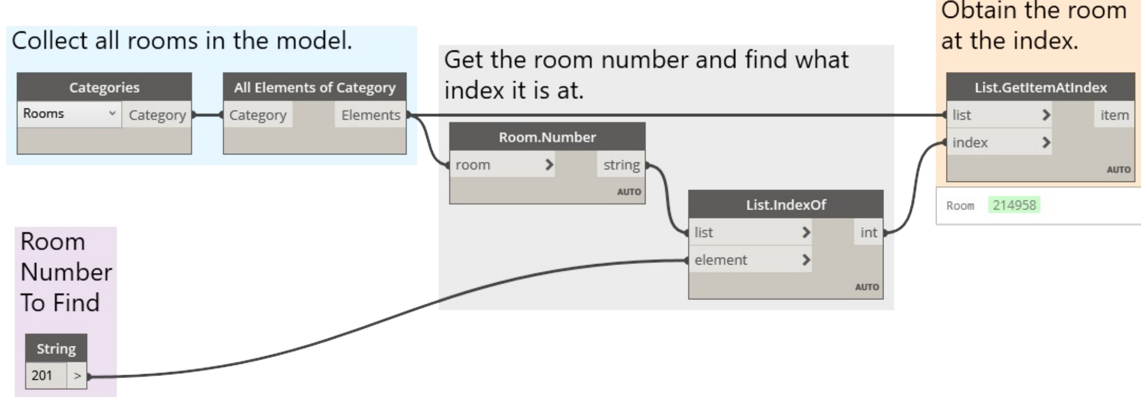
Подобные виды взаимодействия особенно важны при соотнесении данных Revit со строками. Далее приводятся примеры использования этой функции Revit.

Примеры использования

Словари. Примеры использования в Revit

Вам когда-нибудь приходилось искать в Revit информацию по фрагменту данных?

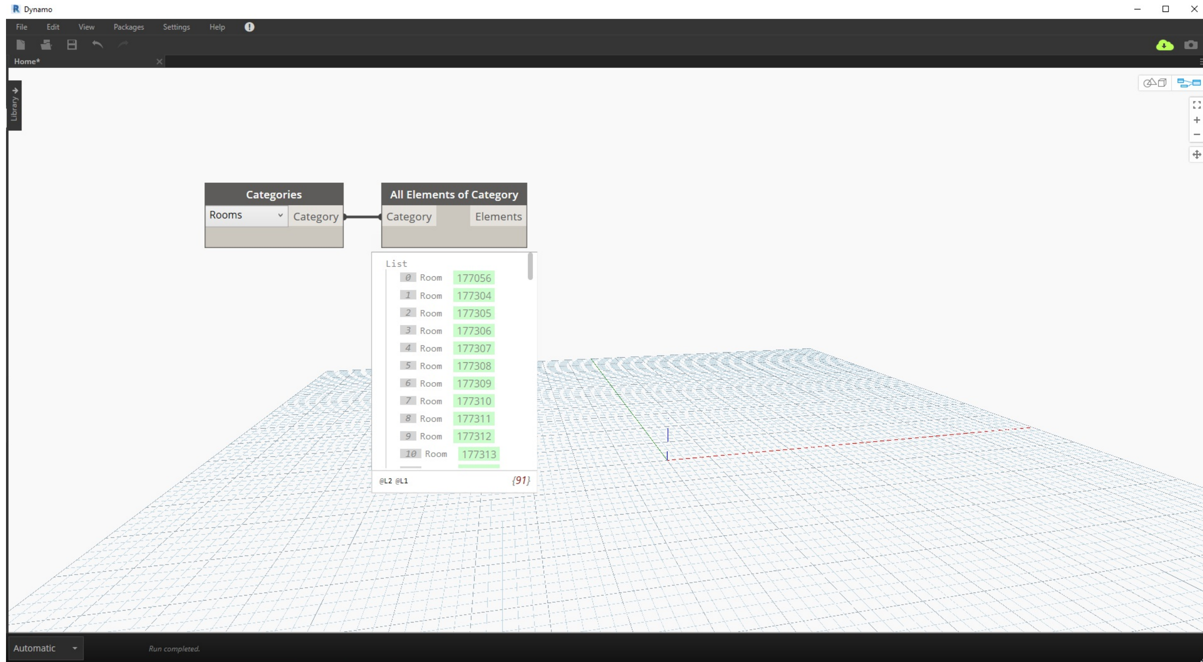
Вы наверняка делали что-то подобное:



Процесс, проиллюстрированный выше, включает следующие этапы: сбор всех помещений в модели Revit, получение индекса нужного помещения (по его номеру) и, наконец, извлечение этого помещения из индекса.

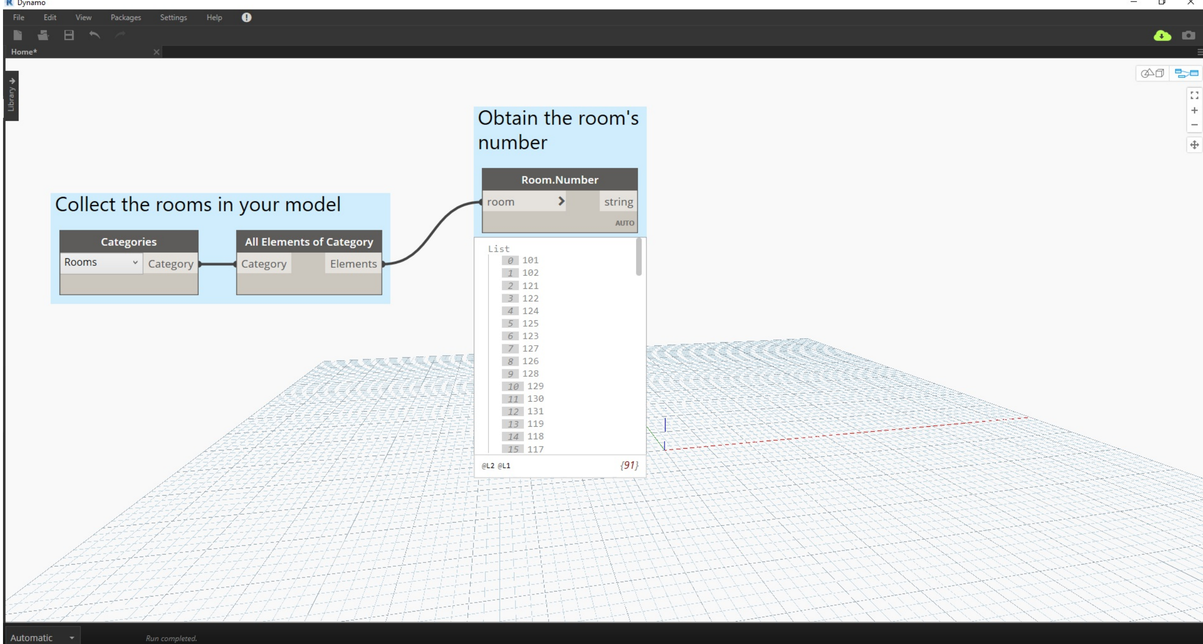
А теперь посмотрите, как тот же процесс выглядит при использовании словарей.

Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»): RoomDictionary.dyn. Полный список файлов примеров можно найти в приложении.



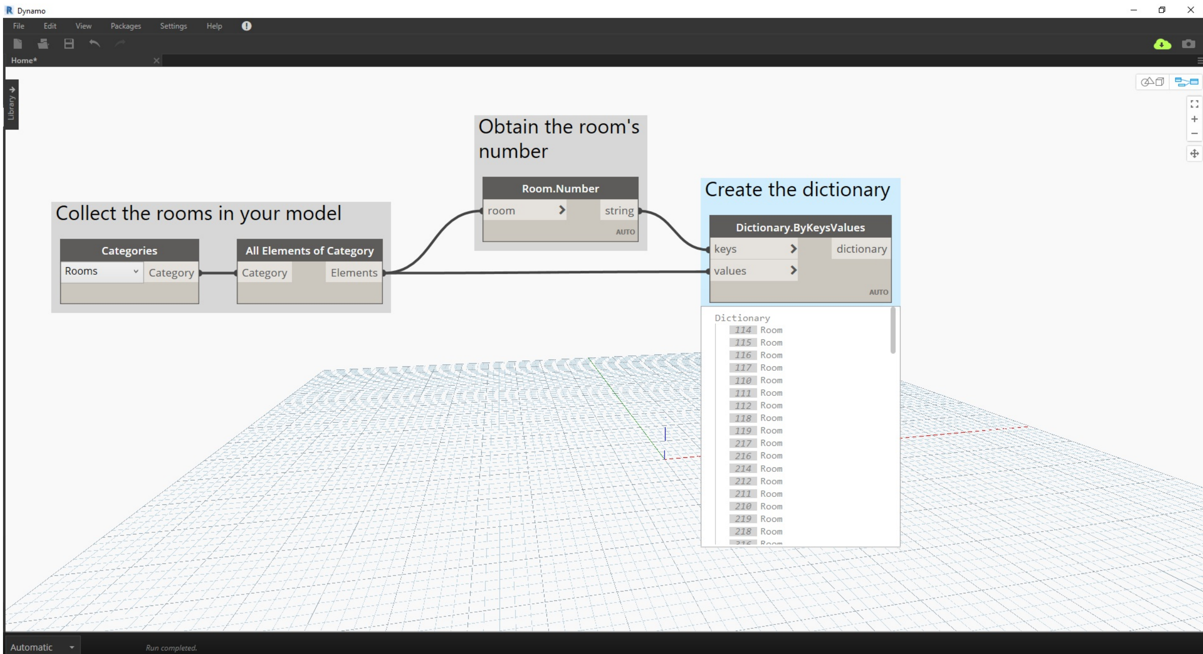
Сначала необходимо сгруппировать все помещения в модели Revit.

- Выберите нужную категорию Revit (в данном случае работа ведется с помещениями).
- Запрограммируйте сбор всех таких элементов в Dynamo.



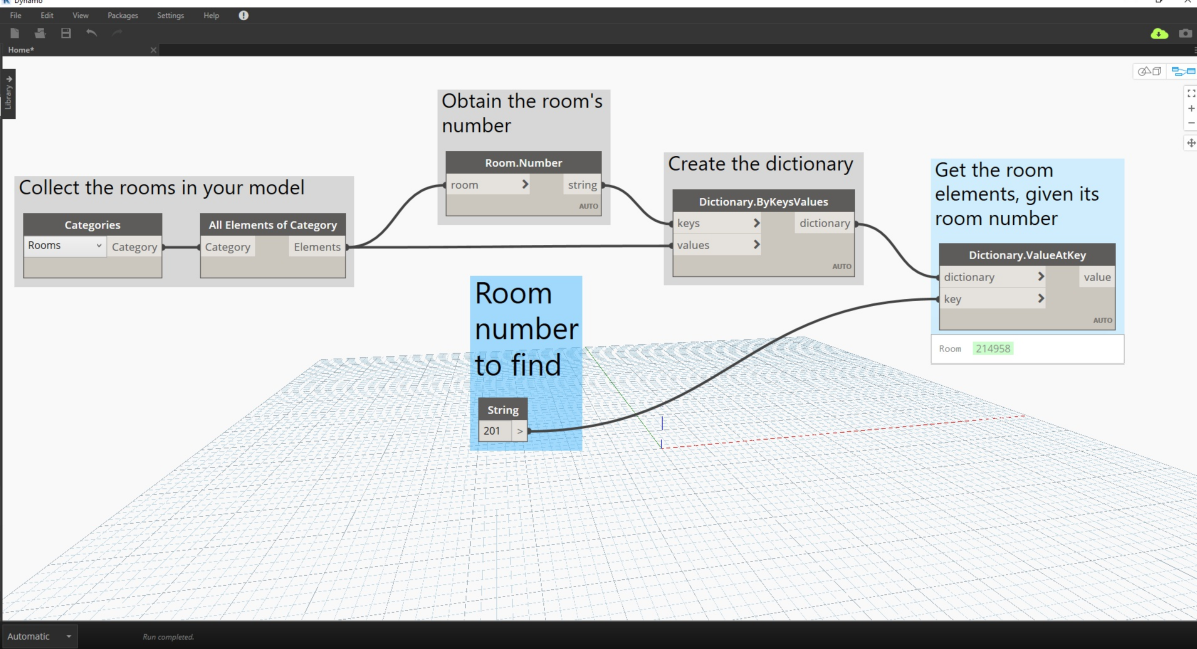
Затем решите, какие ключи будут использоваться для поиска этих данных. Сведения о ключах см. в разделе 9-1. Что такое словарь.

- Данные, которые будут использоваться, — это номер помещения.



Теперь создайте словарь по заданным ключам и элементам.

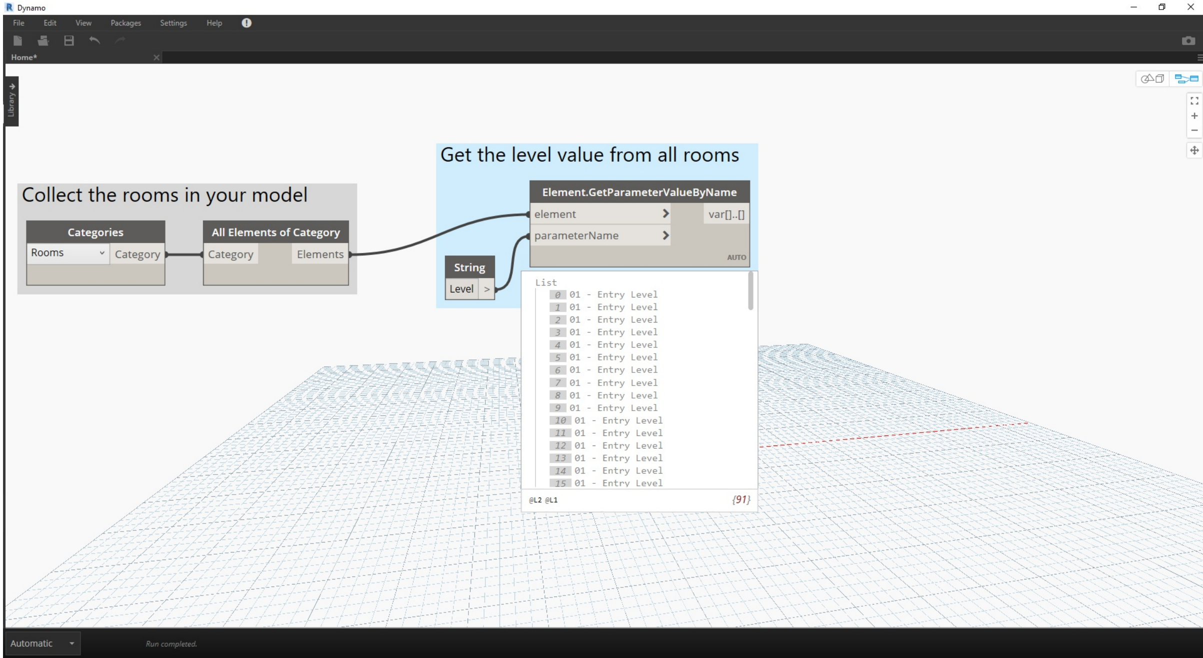
- Узел `Dictionary.ByKeysValues` создает словарь с соответствующими входными данными.
- Данные, поступающие в порт `keys`, должны быть строковыми. Поле `values` поддерживает разные типы объектов.



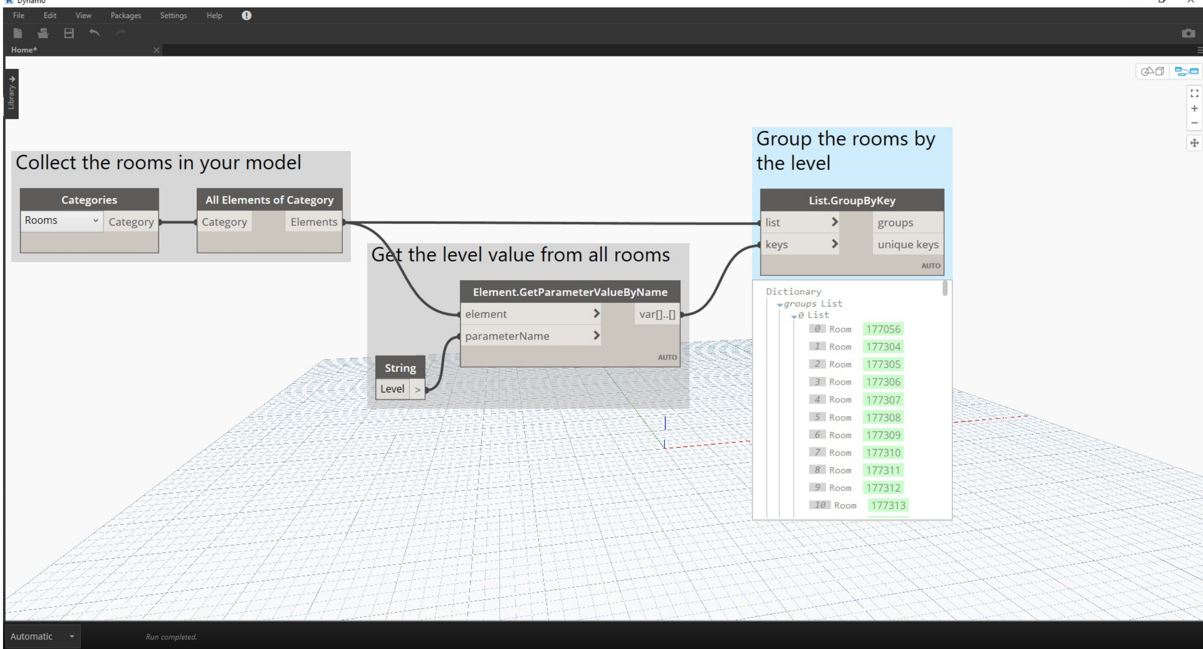
Наконец, извлеките помещение из словаря с номером помещения.

- Порт `string` выдает ключ, который используется для поиска объекта в словаре.
- Метод `Dictionary.ValueAtKey` получает объект из словаря.

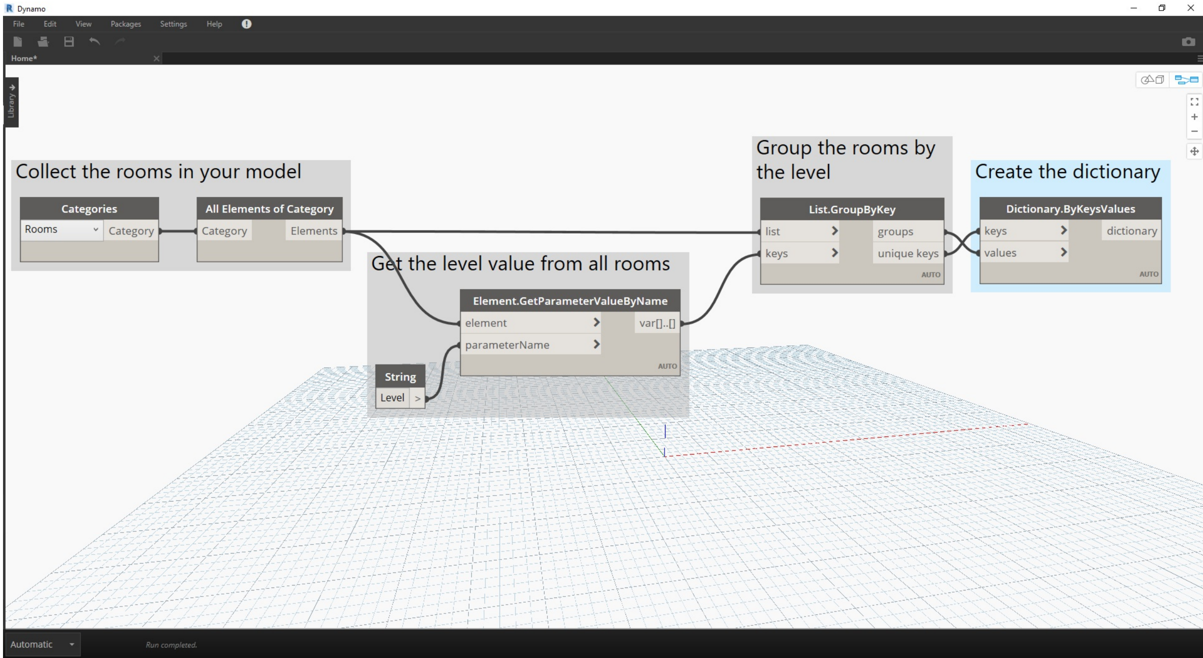
Используя аналогичную логику работы со словарями, можно создавать словари, содержащие сгруппированные объекты. Если нужно найти все помещения на конкретном этаже, измените приведенный выше график следующим образом.



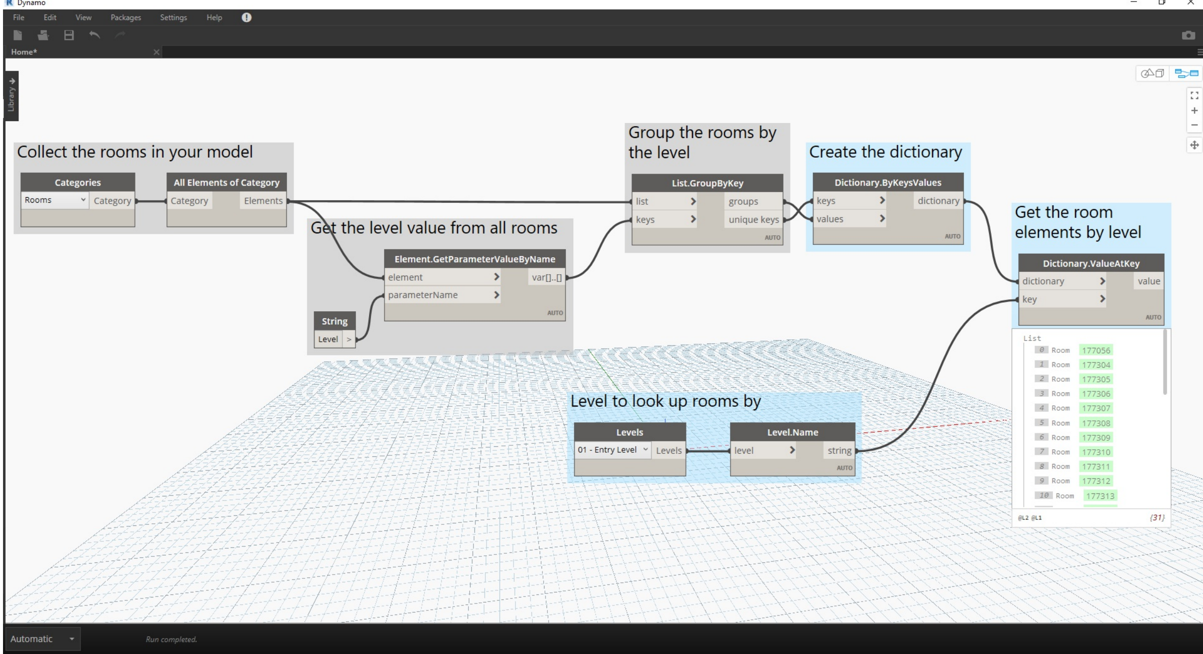
- Вместо того чтобы использовать номер помещения в качестве ключа, можно использовать значение параметра (в данном случае этаж).



- Теперь можно сгруппировать помещения по этажу, на котором они находятся.



- Когда элементы сгруппированы по этажам, можно использовать общие (уникальные) ключи в качестве ключей для словаря, а списки помещений — в качестве его элементов.



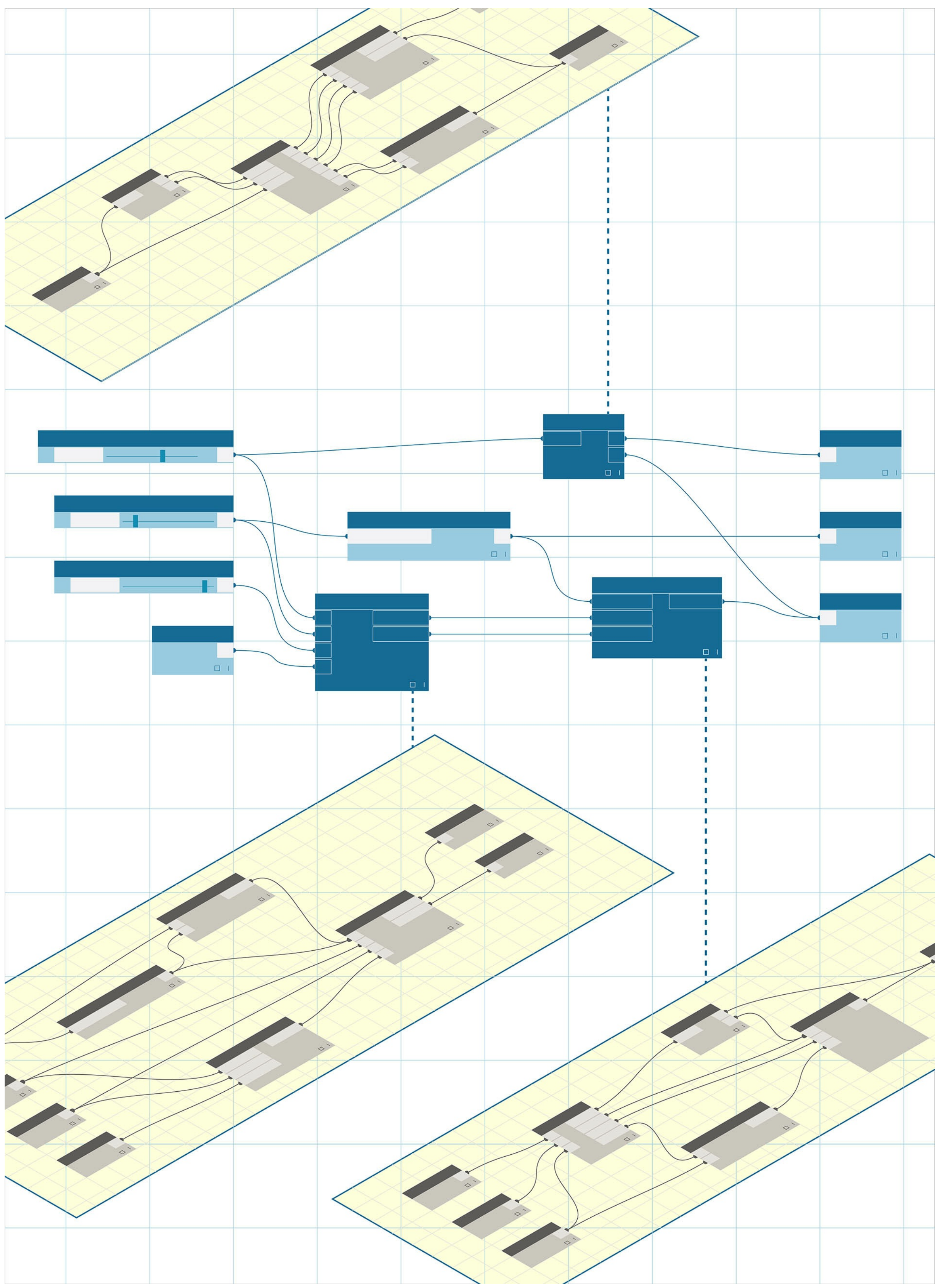
- Наконец, используя этажи, заданные в модели Revit, выполните поиск в словаре всех помещений, расположенных на нужном этаже. Метод `Dictionary.ValueAtKey` использует имя этажа в качестве входных данных и возвращает объекты помещений на этом этаже.

Возможности использования словарей практически безграничны. Сама по себе возможность соотнесения данных BIM в Revit с тем или иным элементом открывает широкий спектр вариантов применения.

Пользовательские узлы

Пользовательские узлы

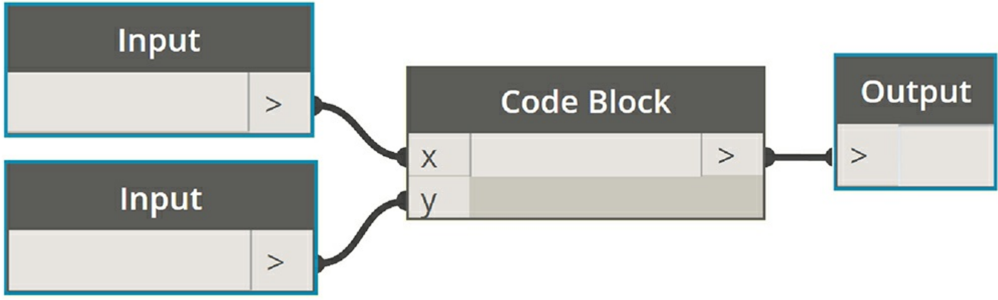
Готовые узлы, доступные в библиотеке Dynamo, поддерживают широкий спектр функциональных возможностей. Если требуются дополнительные возможности, например для внедрения повторяющихся процедур или разработки особого графика для демонстрации коллегам, в Dynamo можно создать собственные пользовательские узлы.



Пользовательские узлы: введение

Пользовательские узлы

В Dynamo доступно большое количество стандартных узлов, позволяющих решать широкий спектр задач визуального программирования. Однако иногда создание пользовательских узлов позволяет прийти к более быстрому, оригинальному или удобному для совместного использования решению. Такие узлы можно многократно использовать в разных проектах, они помогают упростить и очистить график. Опубликовав их в менеджере пакетов, этими узлами можно поделиться с другими пользователями Dynamo по всему миру.



Удаление лишних элементов из графика

Пользовательские узлы создаются путем помещения стандартных и созданных разработчиками узлов в так называемый пользовательский узел Dynamo, который можно рассматривать как своего рода контейнер. При запуске такого узла-контейнера в графике происходит запуск всех содержащихся в нем компонентов. В результате получается удобная комбинация узлов для многократного использования и предоставления другим пользователям.

Адаптация к изменениям

Если в графике присутствует несколько копий пользовательского узла, их все можно отредактировать одновременно, изменив базовый экземпляр этого узла. Это позволяет с легкостью редактировать график и адаптировать его к изменениям, которые могут возникнуть в проекте или рабочем процессе.

Совместная работа

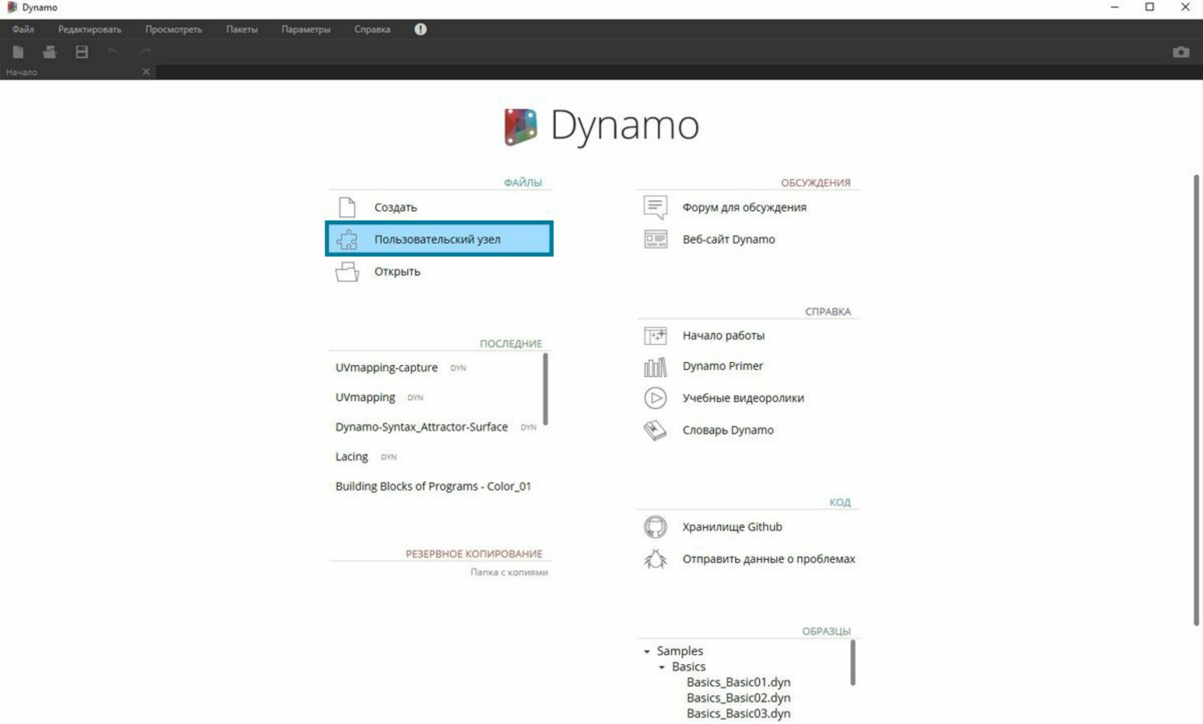
Главное преимущество пользовательских узлов — удобство совместной работы. Например, если опытный программист создает в Dynamo сложный график, который затем требуется передать проектировщику, не работавшему с Dynamo, то программист может упростить представление графика и сделать его максимально доступным для проектировщика. Пользовательский узел-контейнер, в который помещается график, можно открывать для редактирования его содержимого, однако внешний вид контейнера при этом может оставаться простым и аккуратным. Таким образом, пользовательские узлы позволяют делать графики Dynamo лаконичными и интуитивно понятными.

Способы создания узлов

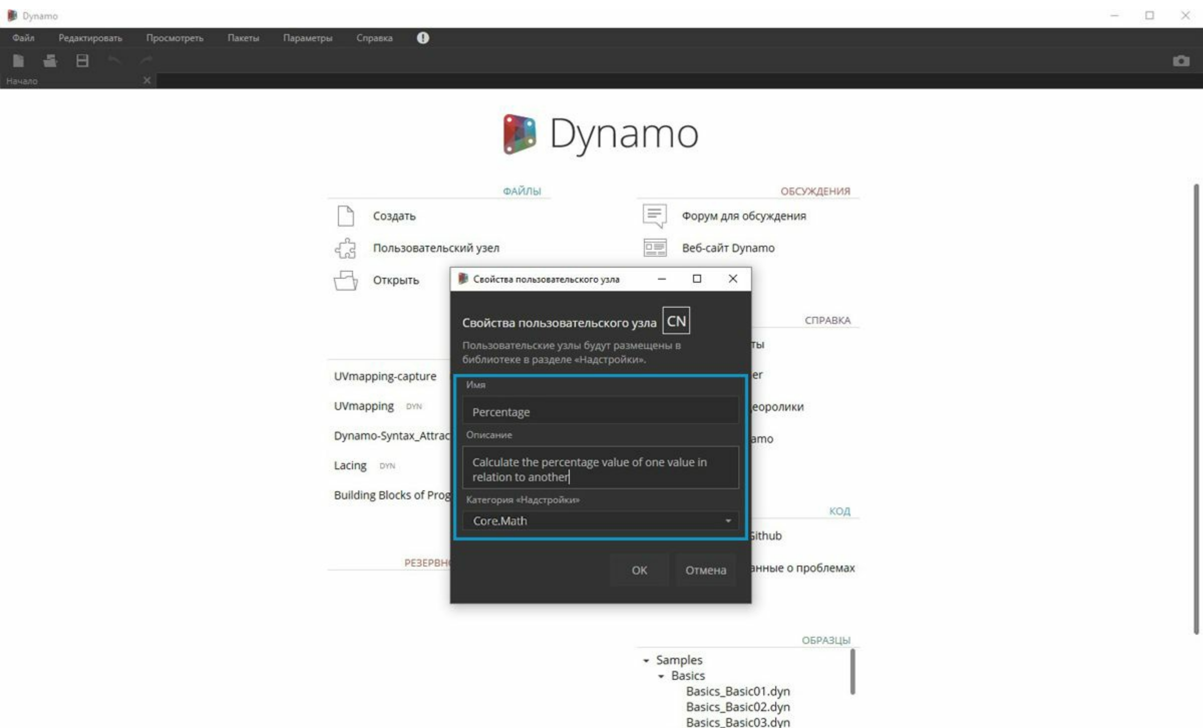
Создавать пользовательские узлы в Dynamo можно самыми разными способами. В примерах, которые приведены в этой главе, пользовательские узлы создаются с помощью функций интерфейса Dynamo. Программисты, интересующиеся возможностями форматирования в C# или Zero Touch, могут ознакомиться с подробным обзором на этой странице справки Wiki по Dynamo.

Среда пользовательского узла

Перейдите в среду пользовательского узла, чтобы создать простой узел для расчета процентного соотношения. Среда пользовательского узла отличается от среды графика Dynamo, но работа в ней осуществляется на основе тех же принципов. Итак, пора создать первый пользовательский узел.

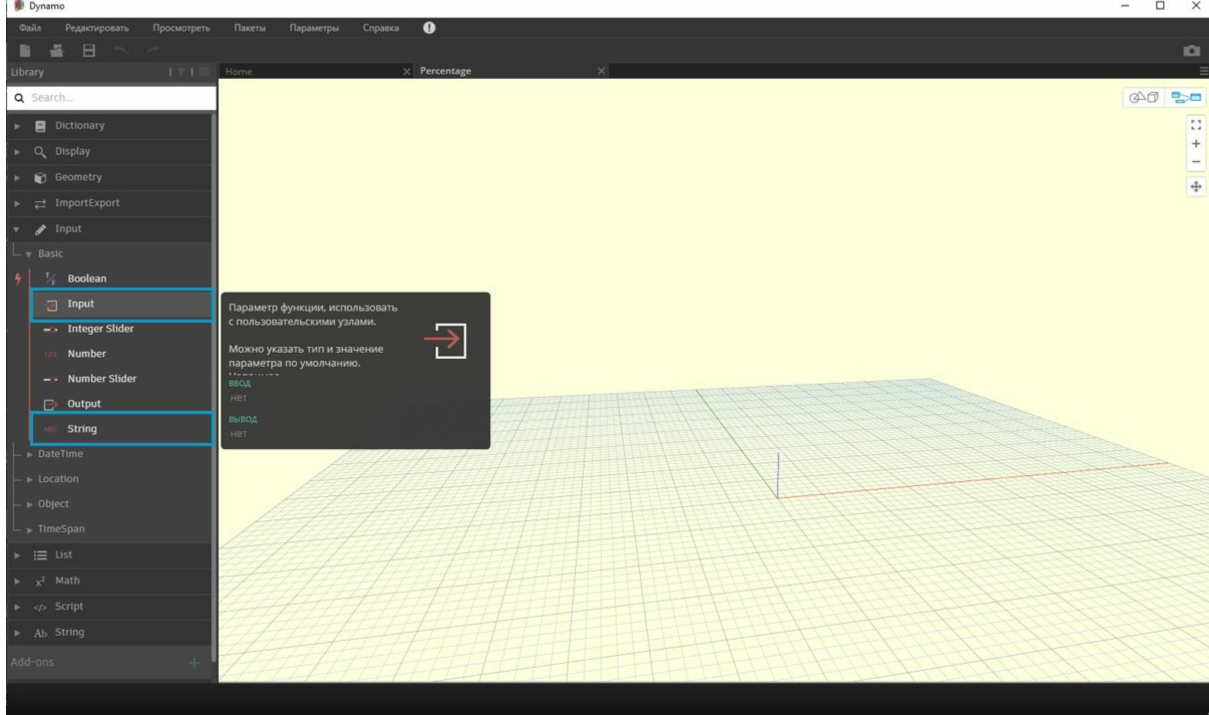


Чтобы создать пользовательский узел с нуля, запустите Dynamo и выберите «Пользовательский узел» или используйте сочетание клавиш CTRL + SHIFT + N в рабочей области.

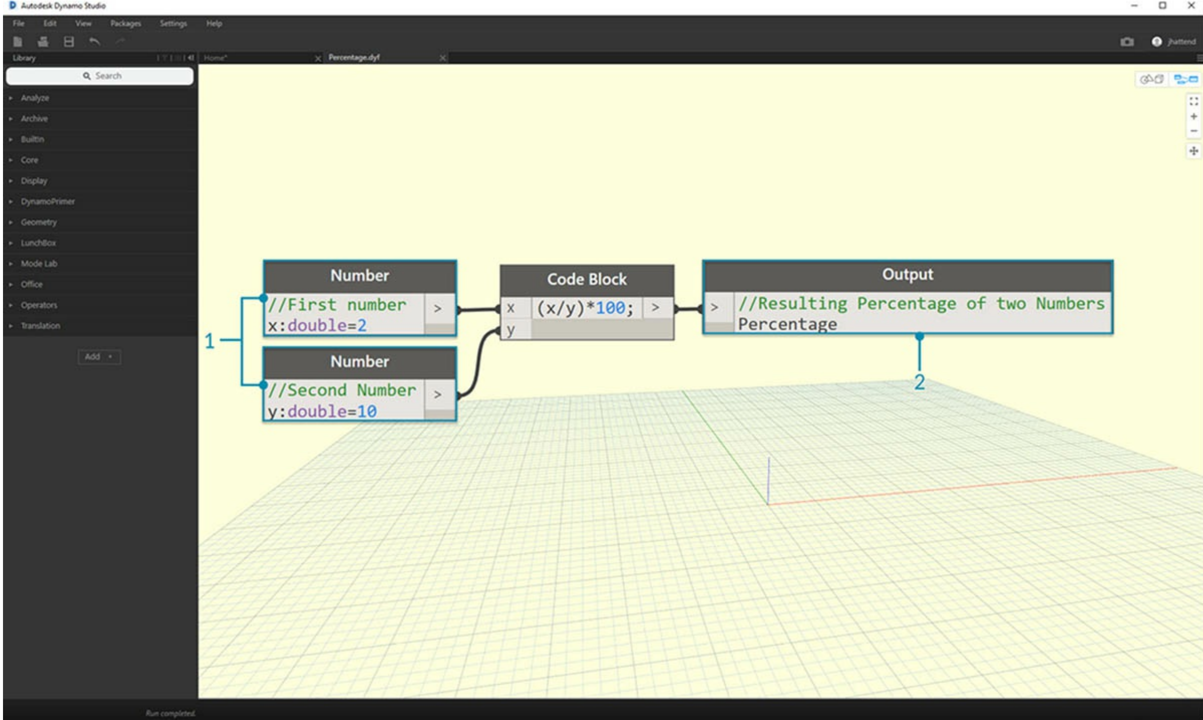


Задайте имя, описание и категорию в диалоговом окне «Свойства пользовательского узла».

1. **Имя:** Percentage
2. **Описание:** расчет процентного соотношения между двумя значениями.
3. **Категория:** Core.Math



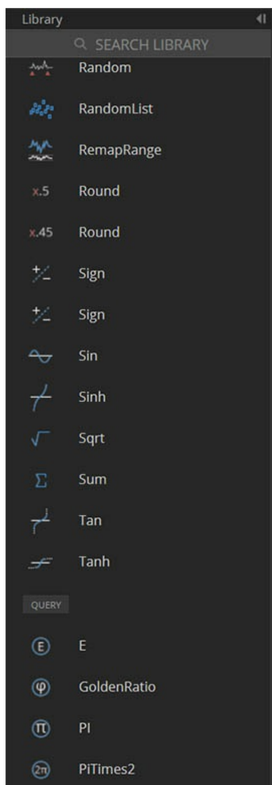
Откроется рабочая область с желтым фоном, указывающим на то, что вы находитесь в среде пользовательского узла. В этом окне можно получить доступ ко всем стандартным узлам Dynamo, а также к узлам **ввода** и **вывода**, которые используются для обозначения данных, поступающих в пользовательский узел на входе и получаемых из него на выходе. Они находятся в разделе *Core > Input*.

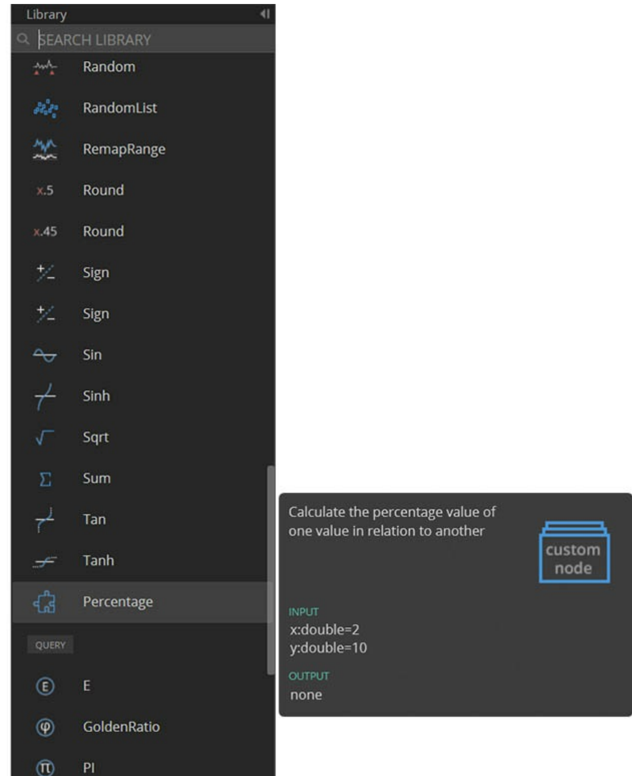


1. **Ввод:** узлы ввода используются для настройки портов ввода пользовательского узла. Синтаксис узла ввода: *имявходногопорта : типданных = значениепо_умолчанию(необязательно).*

1. **Вывод:** аналогично узлам ввода узлы вывода используются для настройки портов вывода пользовательского узла. К портам ввода и вывода можно добавлять **пользовательские комментарии**, указывающие на тип входных и выходных данных. Подробную информацию см. в разделе Создание пользовательских узлов.

Этот пользовательский узел можно сохранить в файле DYF (вместо стандартного формата DYN), после чего он автоматически станет доступен в текущем и будущих сеансах. Найти этот узел можно будет в библиотеке, открыв категорию, заданную в свойствах узла.





Слева: категория Core > Math библиотеки по умолчанию. Справа: категория Core > Math, в которую добавлен новый пользовательский узел.

Дальнейшее изучение

После создания первого пользовательского узла можно переходить к дальнейшим разделам, в которых подробнее рассматриваются функциональные возможности пользовательских узлов и процесс публикации типовых рабочих процессов. В следующем разделе представлен процесс разработки пользовательского узла, позволяющего переносить геометрию с одной поверхности на другую.

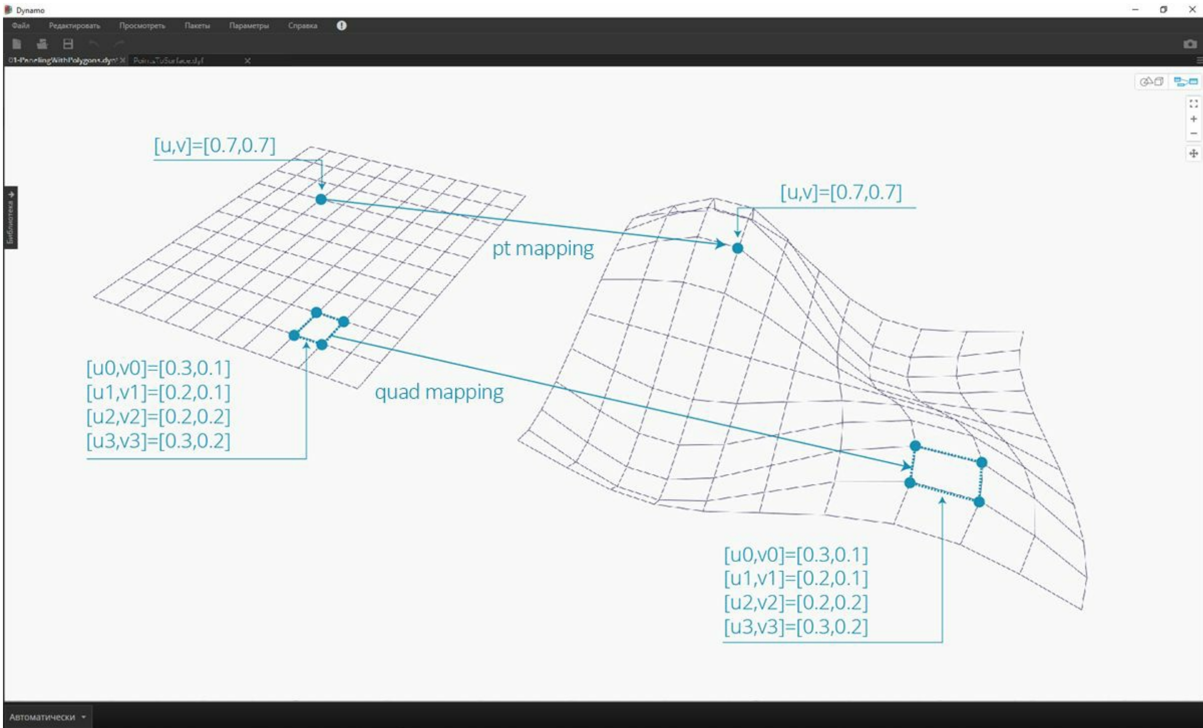
Создание пользовательских узлов

Создание пользовательских узлов

В Dynamo предусмотрено несколько способов создания пользовательских узлов. Их можно создавать с нуля на основе существующего графика или непосредственно на языке C#. В этом разделе рассматривается создание пользовательского узла в интерфейсе Dynamo на основе существующего графика. Этот метод идеально подходит для очистки рабочего пространства, а также для упаковки последовательности узлов с целью их повторного использования в другом месте.

Пользовательские узлы для UV-наложения

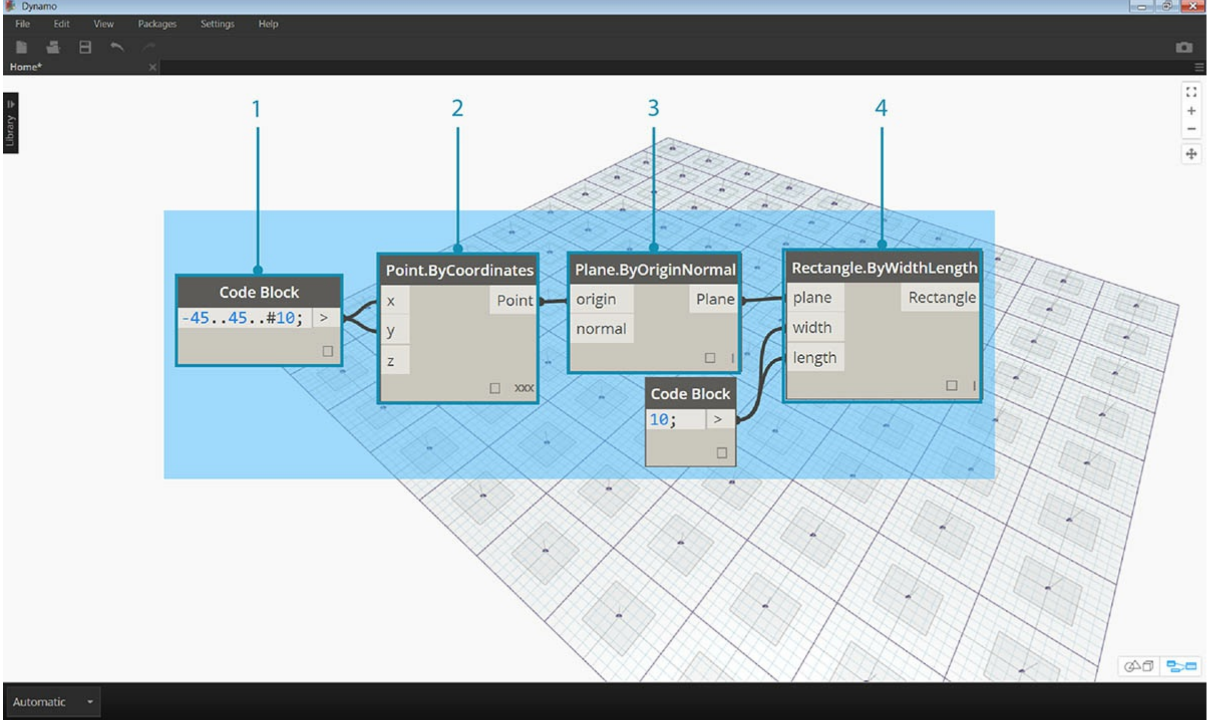
На изображении ниже точка одной поверхности сопоставляется с другой точкой с помощью UV-координат. Этот принцип будет использоваться для создания панелей поверхности, ссылающихся на кривые в плоскости XY. В данном случае сформируем прямоугольные панели, но этим же способом можно создавать разнообразные панели с использованием UV-наложения. Это отличная возможность для разработки пользовательских узлов, так как данную процедуру можно повторять в этом же графике или в других рабочих процессах Dynamo.



Создание пользовательского узла на основе существующего графика

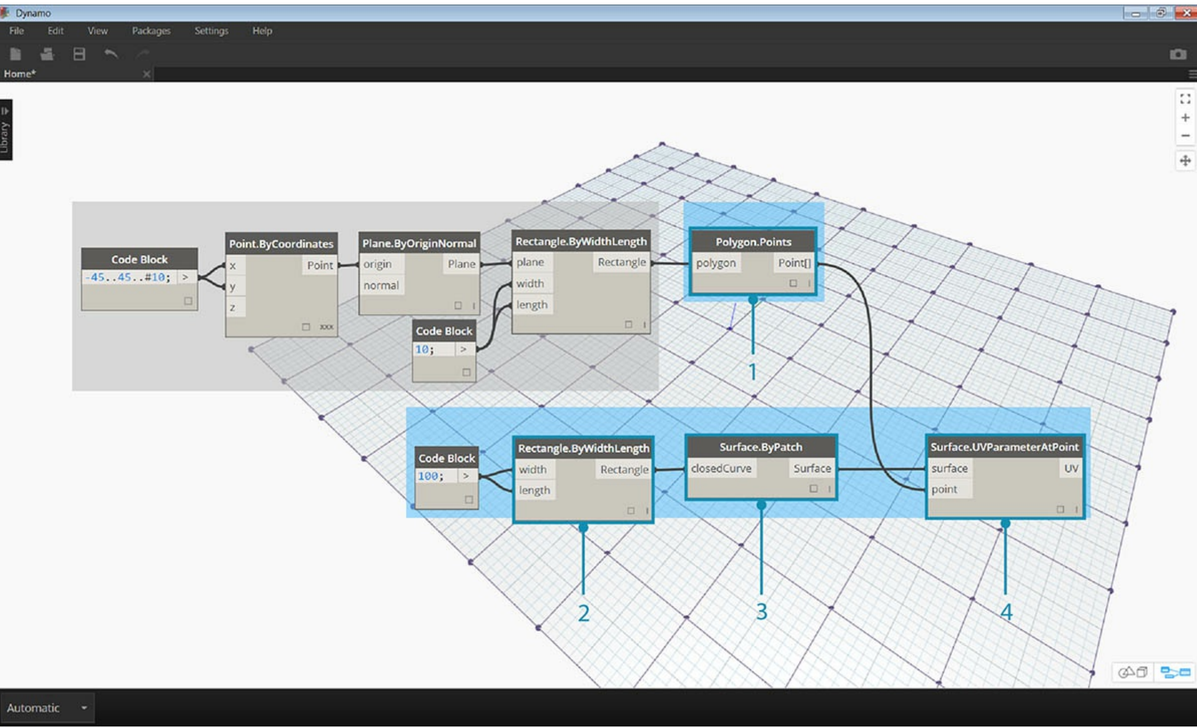
> Скачайте и распакуйте файлы примеров для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. UV-CustomNode.zip

Начните с создания графика, который будет вложен в пользовательский узел. В этом примере с помощью UV-координат мы создадим график, который сопоставляет полигоны базовой поверхности с целевой поверхностью. Эта процедура UV-наложения используется часто, благодаря чему хорошо подходит для создания пользовательских узлов. Дополнительные сведения о поверхностях и UV-пространстве см. в разделе 5.5. Полная версия графика представлена в файле *UVMapping_Custom-Node.dyn* (в скачанном ранее ZIP-файле).



1. **Блок кода.** Создайте диапазон из 10 чисел от 45 до -45 с помощью блока кода.
2. **Point.ByCoordinates.** Соедините выходные данные блока кода с входными данными x и y, в качестве переплетения выбрав перекрестную ссылку. При этом будет создана сетка точек.
3. **Plane.ByOriginNormal.** Соедините выходные данные *Point* с входными данными *origin*, чтобы создать плоскость в каждой из точек. Будет использован вектор нормали по умолчанию (0,0,1).
4. **Rectangle.ByWidthLength.** Соедините плоскости из предыдущего шага с выходными данными *plane* и с помощью блока кода со значением *10* задайте ширину и длину.

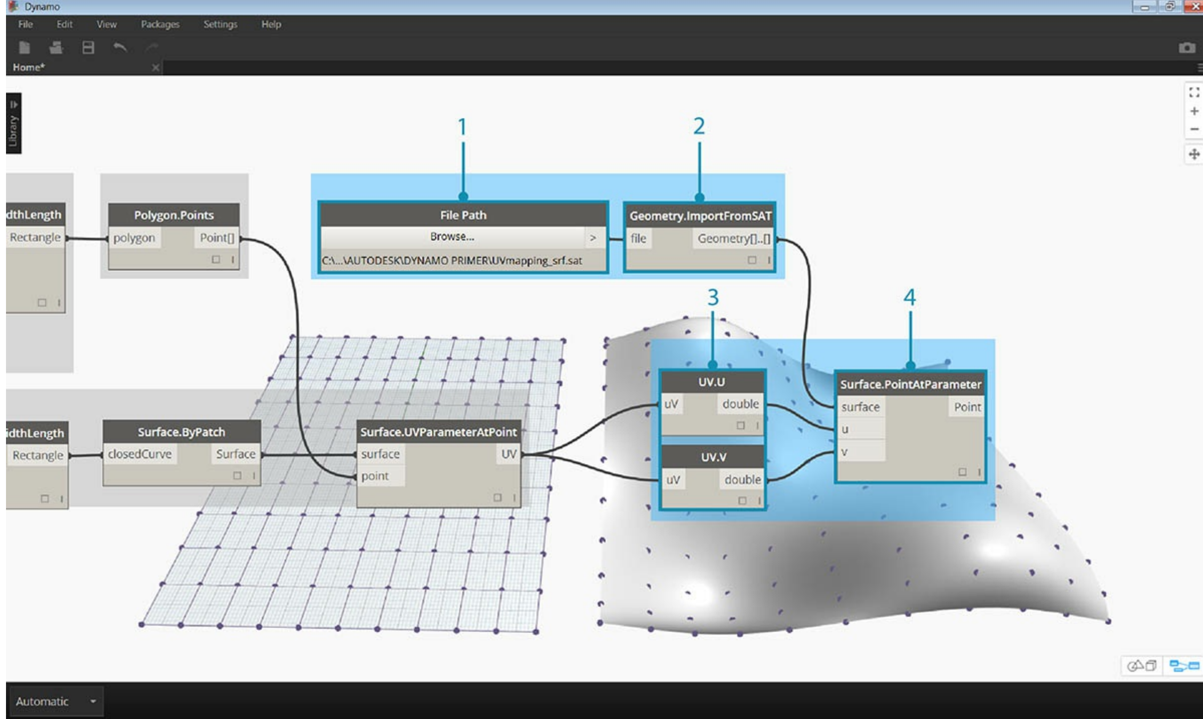
При этом появится сетка прямоугольников. Сопоставьте эти прямоугольники с целевой поверхностью, используя UV-координаты.



1. **Polygon.Points.** Соедините выходные данные Rectangle из предыдущего шага с входными данными *polygon*, чтобы извлечь угловые точки каждого прямоугольника. Именно эти точки будут сопоставляться с целевой поверхностью.
2. **Rectangle.ByWidthLength.** С помощью блока кода со значением *100* задайте ширину и длину прямоугольника. Это будет граница базовой поверхности.
3. **Surface.ByPatch.** Соедините выходные данные Rectangle из предыдущего шага с входными данными *closedCurve* для создания базовой поверхности.
4. **Surface.UVParameterAtPoint.** Соедините выходные данные *Point* узла *Polygon.Points* с выходными данными *Surface* узла

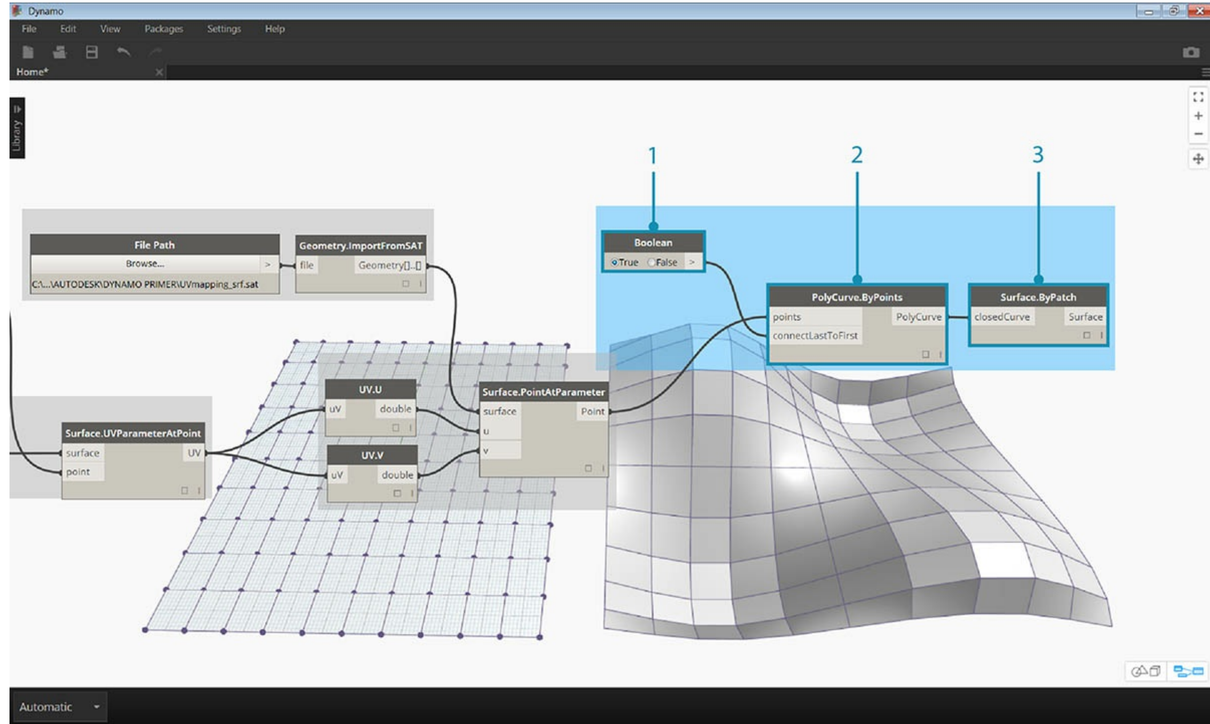
Surface.ByPatch для получения параметра UV в каждой точке.

Теперь, имея базовую поверхность и набор UV-координат, можно импортировать целевую поверхность и сопоставить точки между поверхностями.



1. **Путь к файлу.** Выберите путь к файлу поверхности, которую требуется импортировать. Файл должен иметь тип SAT. Нажмите кнопку *Обзор...* и перейдите к файлу *UVMapping_srf.sat* из скачанного ранее ZIP-файла.
2. **Geometry.ImportFromSAT.** Для импорта поверхности присоедините путь к файлу. При этом в области предварительного просмотра геометрии должна появиться импортированная поверхность.
3. **UV.** Соедините выходные данные параметра UV с узлами *UV.U* и *UV.V*.
4. **Surface.PointAtParameter.** Присоедините импортированную поверхность, а также координаты u и v. Теперь на целевой поверхности должна появиться сетка 3D-точек.

Последний шаг — построение прямоугольных участков поверхности с помощью 3D-точек.

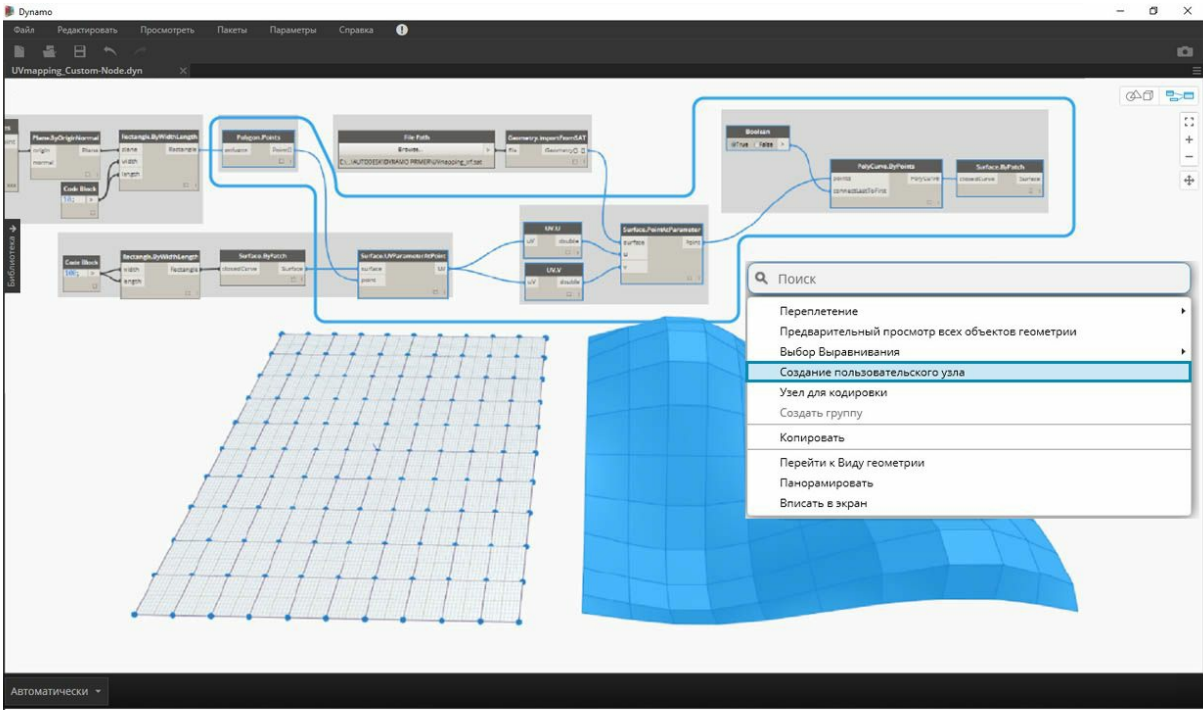


1. **PolyCurve.ByPoints.** Соедините точки на поверхности, чтобы построить поликривую через точки.
2. **Логический оператор.** Добавьте в рабочее пространство логический оператор (Boolean) и соедините его с входными данными

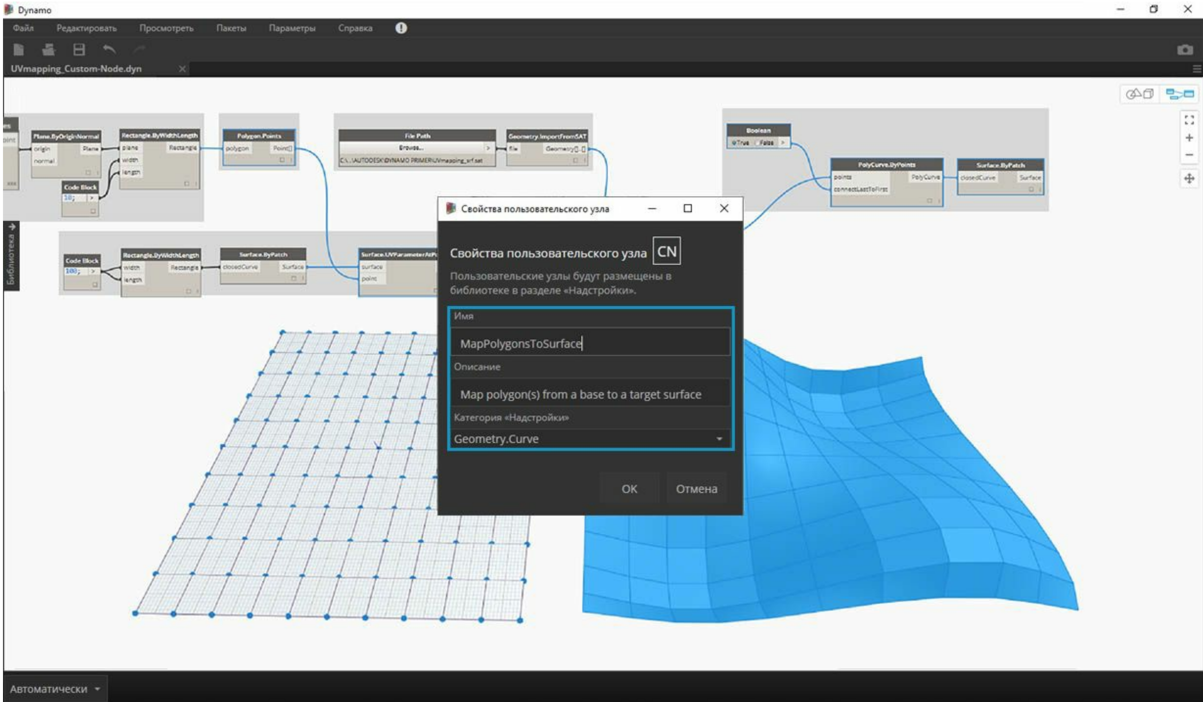
connectLastToFirst, задав значение «Истина» (True), чтобы замкнуть поликривые. При этом должны появиться прямоугольники, сопоставленные с поверхностью.

3. **Surface.ByPatch.** Соедините поликривые с входными данными *closedCurve* для создания участков поверхности.

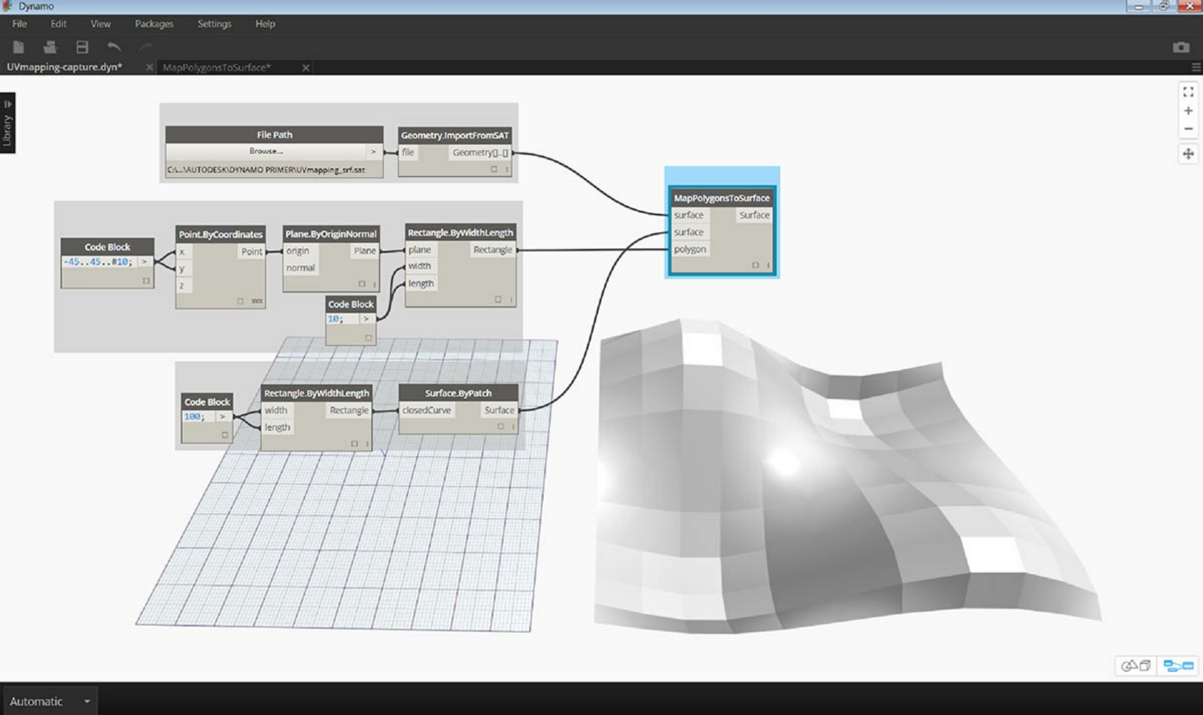
Теперь выберите узлы, которые необходимо вложить в пользовательский узел, учитывая, какие входные и выходные данные должны быть у конечного узла. Пользовательский узел должен быть максимально гибким и пригодным для сопоставления любых полигонов, а не только прямоугольников.



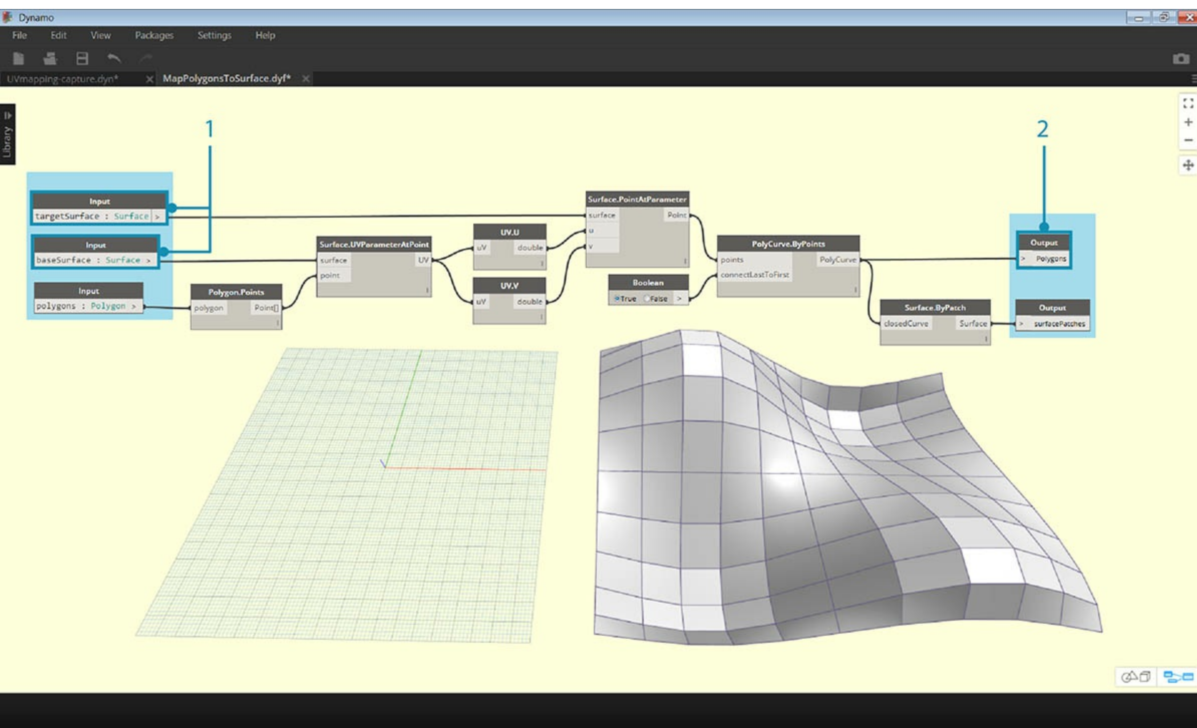
Выберите указанные выше узлы (начиная с *Polygon.Points*), щелкните правой кнопкой мыши в рабочем пространстве и выберите *создание узла из выбранных объектов*.



В диалоговом окне «Свойства пользовательского узла» присвойте пользовательскому узлу имя, укажите описание и категорию.

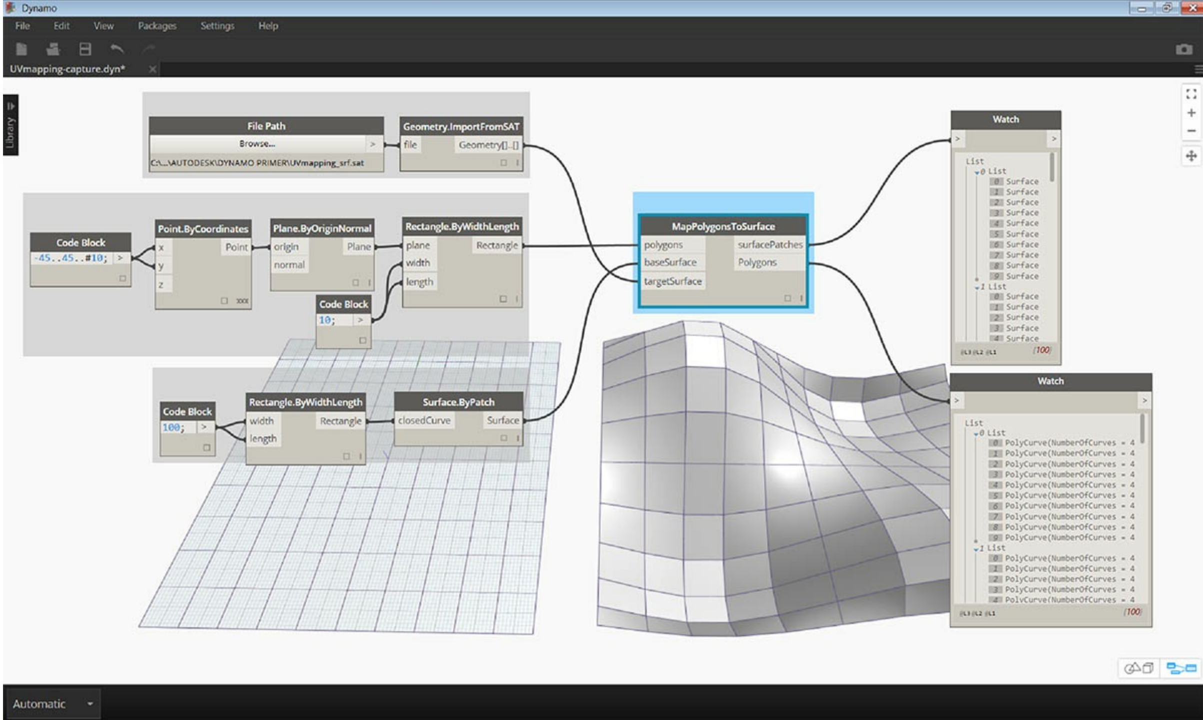


Пользовательский узел в значительной мере очистил рабочее пространство. Обратите внимание, что входным и выходным данным были присвоены имена, соответствующие исходным узлам. Отредактируйте пользовательский узел, чтобы сделать имена более описательными.



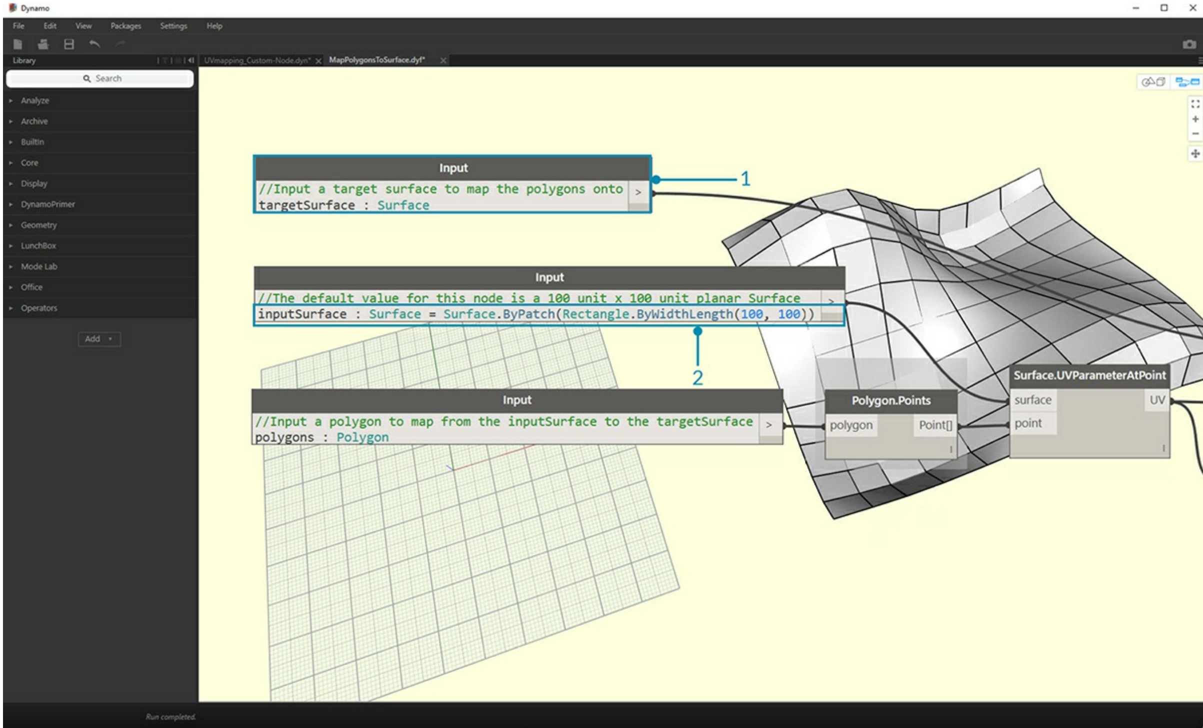
Дважды щелкните пользовательский узел, чтобы отредактировать его. Откроется рабочее пространство с желтым фоном, представляющее собой внутреннюю часть узла.

1. **Входные данные.** Измените имена входных параметров, задав *baseSurface* и *targetSurface*.
2. **Выходные данные.** Добавьте дополнительных выходной параметр для сопоставленных полигонов. Сохраните пользовательский узел и вернитесь в исходное рабочее пространство.



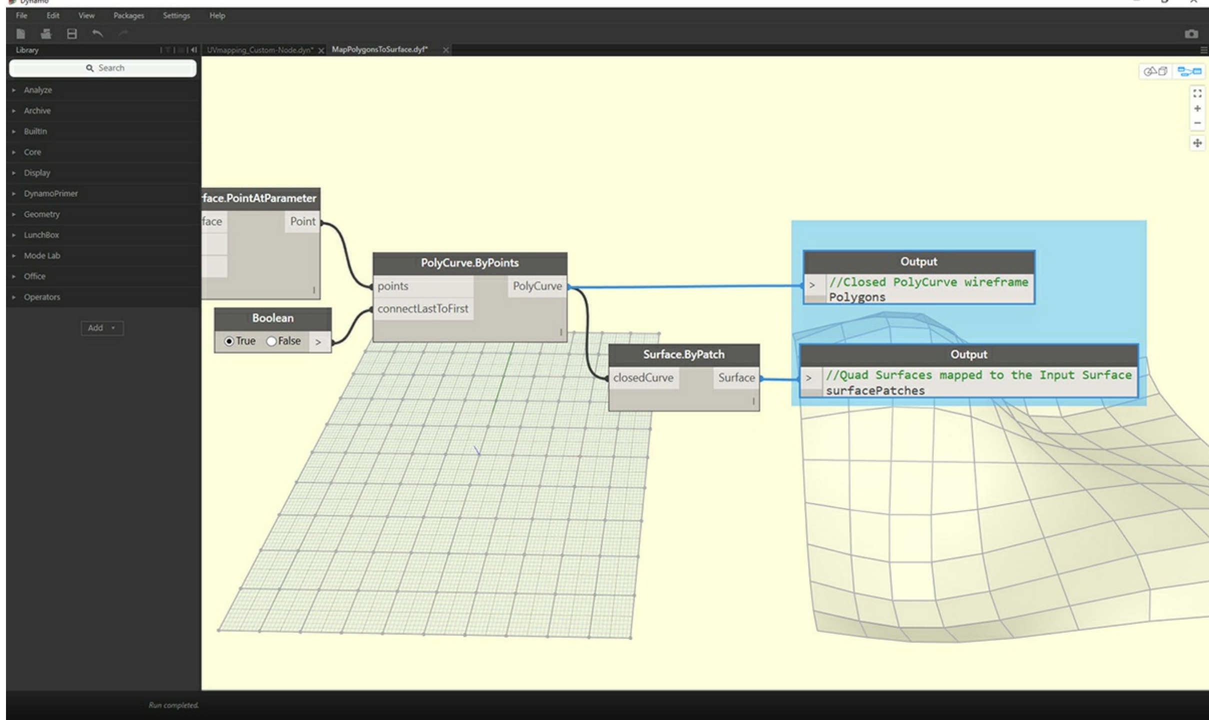
Внесенные изменения отражаются в узле **MapPolygonsToSurface**.

Для наглядности можно добавить к узлу **Пользовательские комментарии**. В комментариях можно задать сведения о типах входных и выходных данных или разъяснить функции узла. Комментарии отображаются при наведении курсора на входной или выходной параметр пользовательского узла.

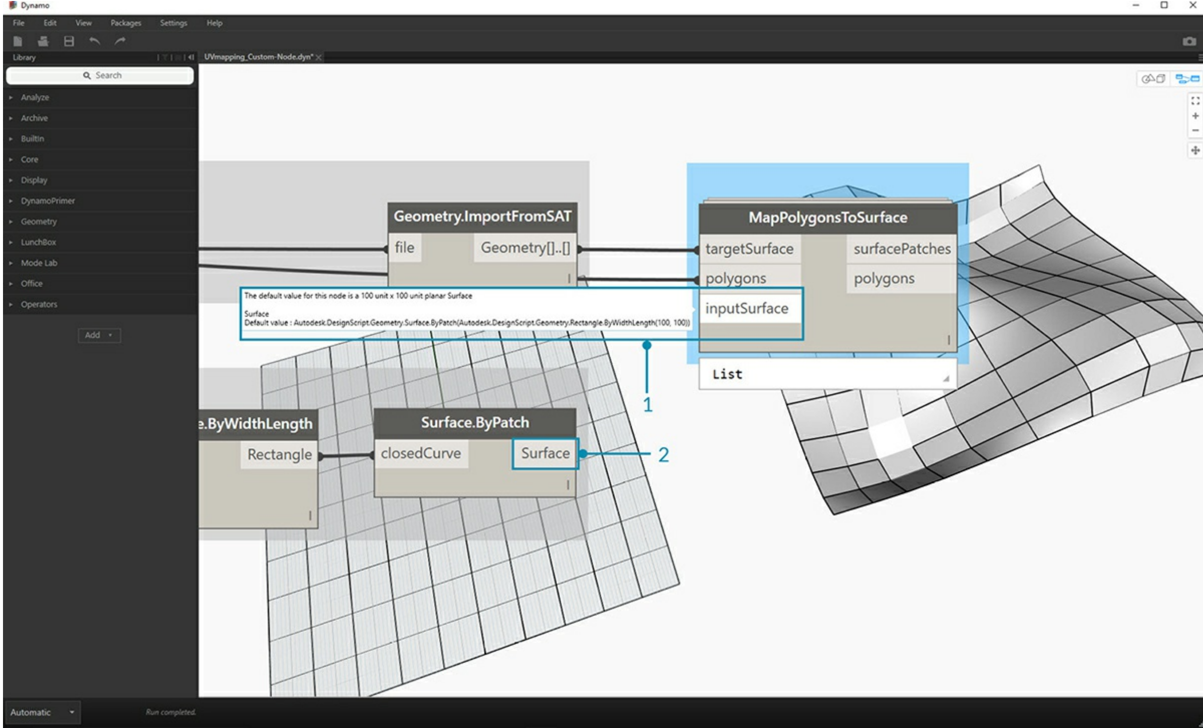


Дважды щелкните пользовательский узел, чтобы отредактировать его. Снова откроется рабочее пространство с желтым фоном.

1. Начните редактирование блока кода Input. Чтобы создать комментарий, введите символы «//», а затем текст комментария. Добавьте любые пояснения к узлу. В данном случае будет дано описание входного параметра *targetSurface*.
2. Кроме того, задайте значение по умолчанию для входного параметра *inputSurface*, указав это значение в качестве типа входных данных. В данном случае в качестве значения по умолчанию будет задан исходный набор Surface.ByPatch.



Комментарии также можно применить к выходным параметрам. Начните редактирование текста в блоке кода Output. Чтобы создать комментарий, введите символы «//», а затем текст комментария. Добавьте пояснения к выходным параметрам *Polygons* и *surfacePatches*, добавив для них подробное описание.



>

1. Наведите курсор на пользовательский узел Inputs, чтобы просмотреть комментарии.
2. Так как для входного параметра *inputSurface* задано значение по умолчанию, при запуске определения можно не вводить значение поверхности.

Публикация узлов в библиотеку

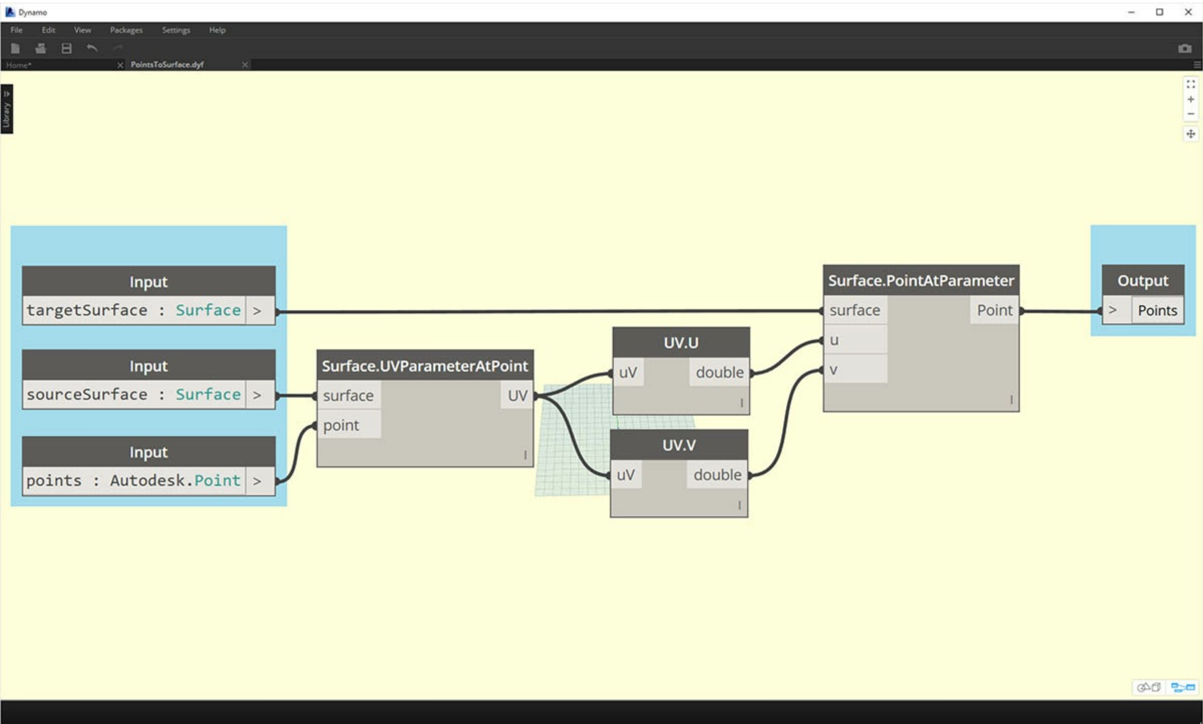
Добавление узлов в библиотеку

Вы создали пользовательский узел и применили его к определенному процессу на графике Dynamo. Это очень полезный узел, поэтому его надо сохранить в библиотеке Dynamo для использования в других графиках. Для этого нужно опубликовать узел локально. Этот процесс аналогичен публикации пакета, который будет рассмотрен подробнее в следующей главе.

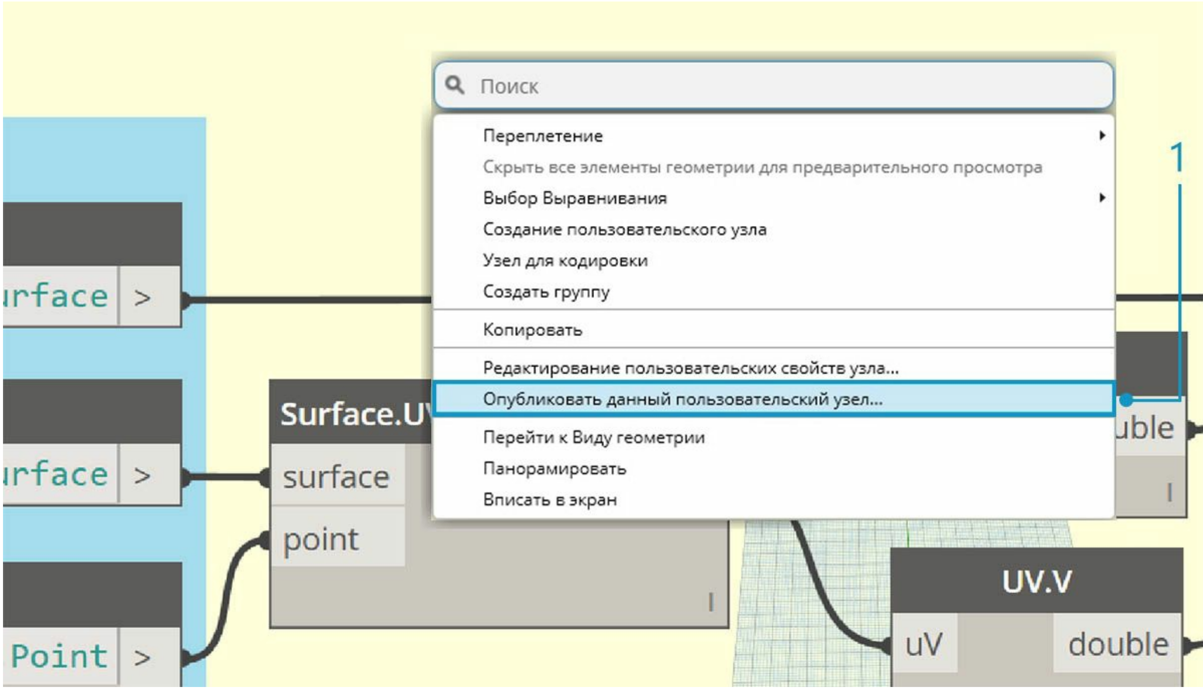
Локальная публикация пользовательского узла

Продолжайте работу с пользовательским узлом, созданным в предыдущем разделе. Узел, опубликованный локально, станет доступен в библиотеке Dynamo после запуска нового сеанса. Если пользовательский узел не опубликован, его необходимо разместить в папке графика Dynamo, ссылающегося на этот узел (либо этот узел нужно импортировать в Dynamo с помощью меню *Файл > Импорт библиотеки*).

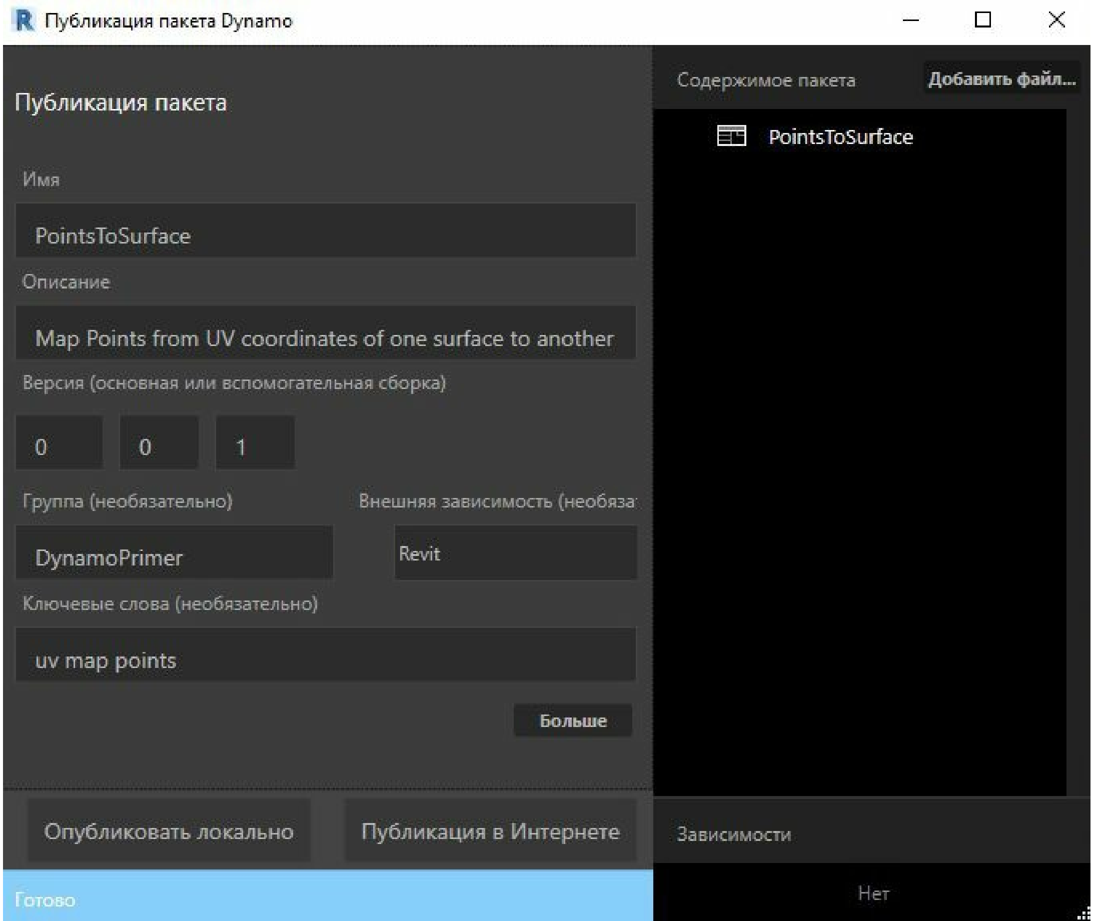
> Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. [PointsToSurface.dyf]



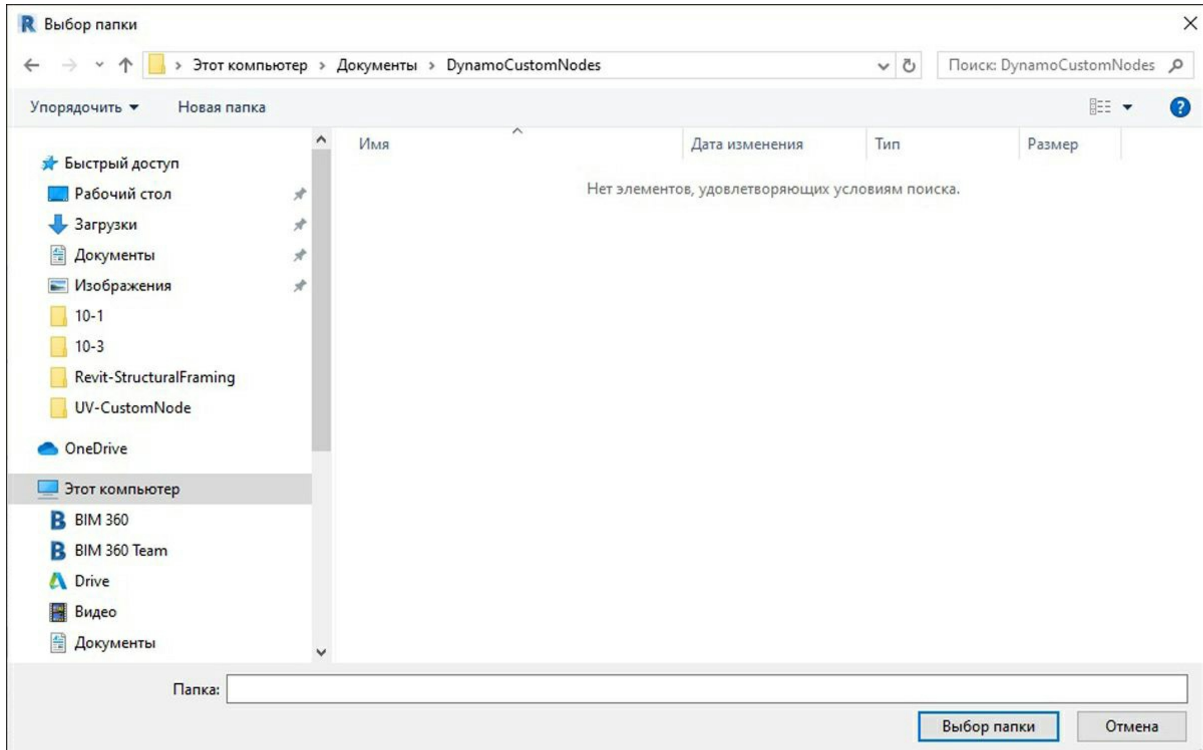
> После открытия пользовательского узла PointsToSurface в редакторе пользовательских узлов Dynamo отобразится показанный график. Пользовательский узел также можно открыть, дважды щелкнув его в редакторе графика Dynamo.



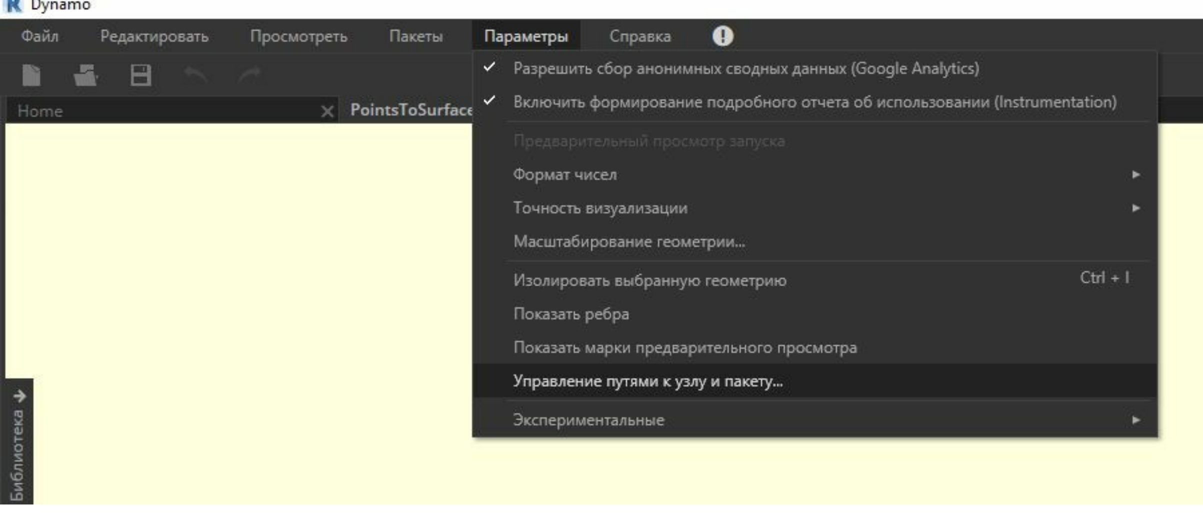
1. Чтобы опубликовать пользовательский узел локально, щелкните правой кнопкой мыши в рабочей области и выберите *Опубликовать данный пользовательский узел...*



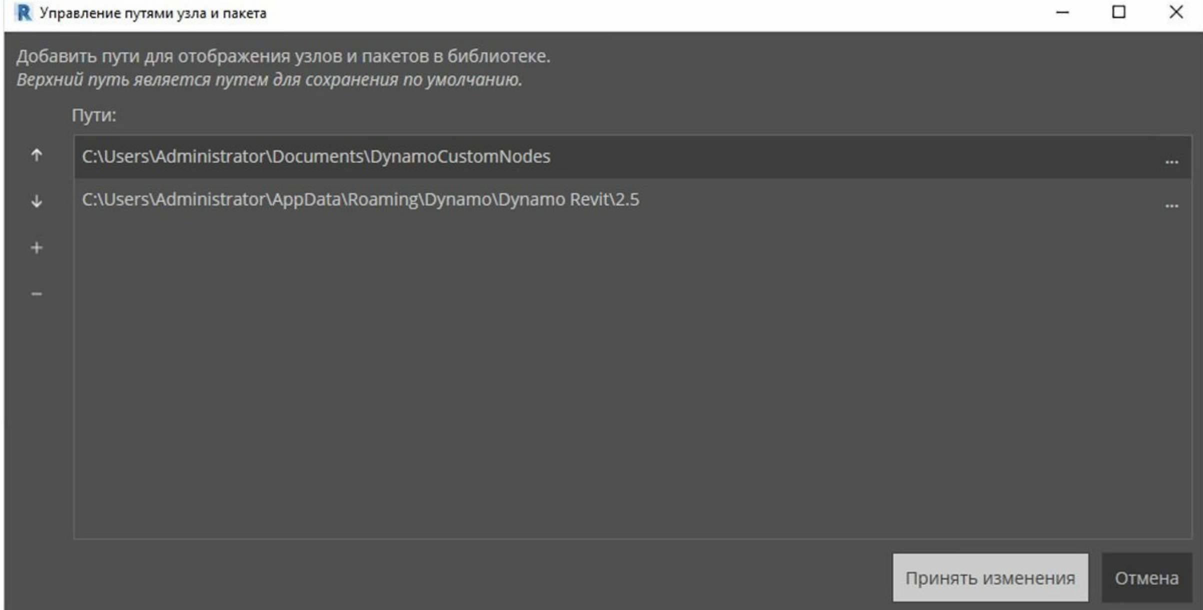
Укажите все необходимые сведения, как показано на изображении выше, и выберите *Опубликовать локально*. Обратите внимание, что в поле «Группа» задается основной элемент, который будет доступен в меню Dynamo.



Выберите папку для хранения всех пользовательских узлов, которые планируется опубликовать локально. Приложение Dynamo будет проверять эту папку каждый раз при загрузке, поэтому она должна находиться в постоянном расположении. Перейдите к этой папке и нажмите *Выбрать папку*. Пользовательский узел Dynamo публикуется локально и теперь будет отображаться на панели инструментов Dynamo при каждой загрузке программы.

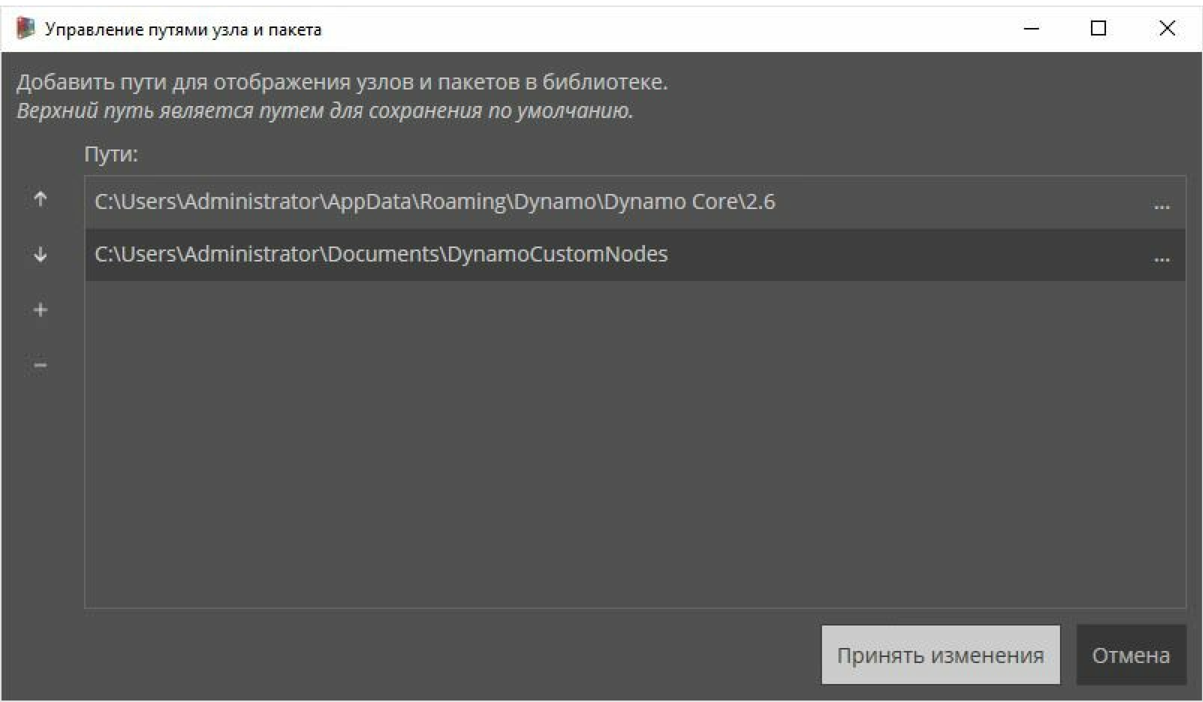


1. Чтобы проверить расположение папки пользовательского узла, откройте меню *Параметры* > *Управление путями к узлу и пакету...*

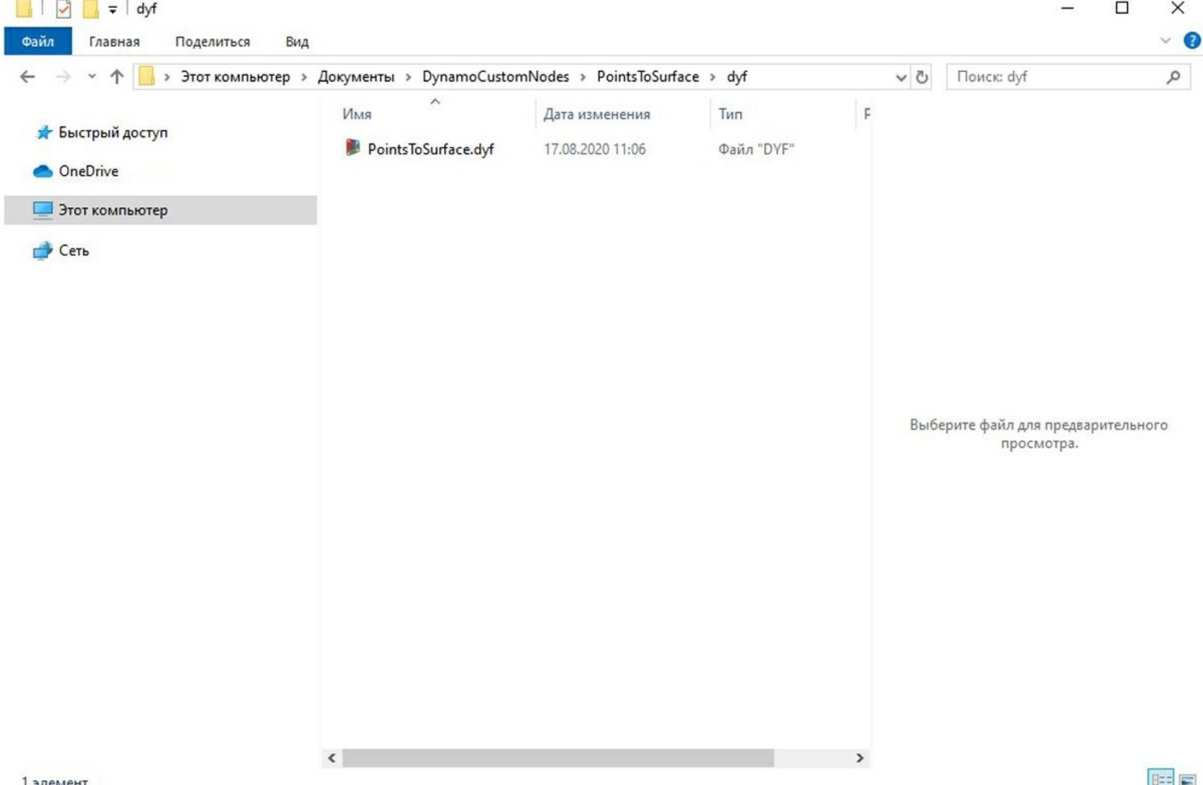


В этом окне отображаются два пути. *AppData\Roaming\Dynamo...* — расположение по умолчанию для пакетов Dynamo, установленных через интернет. *Documents\DynamoCustomNodes...* — расположение пользовательских узлов, которые были опубликованы локально*.

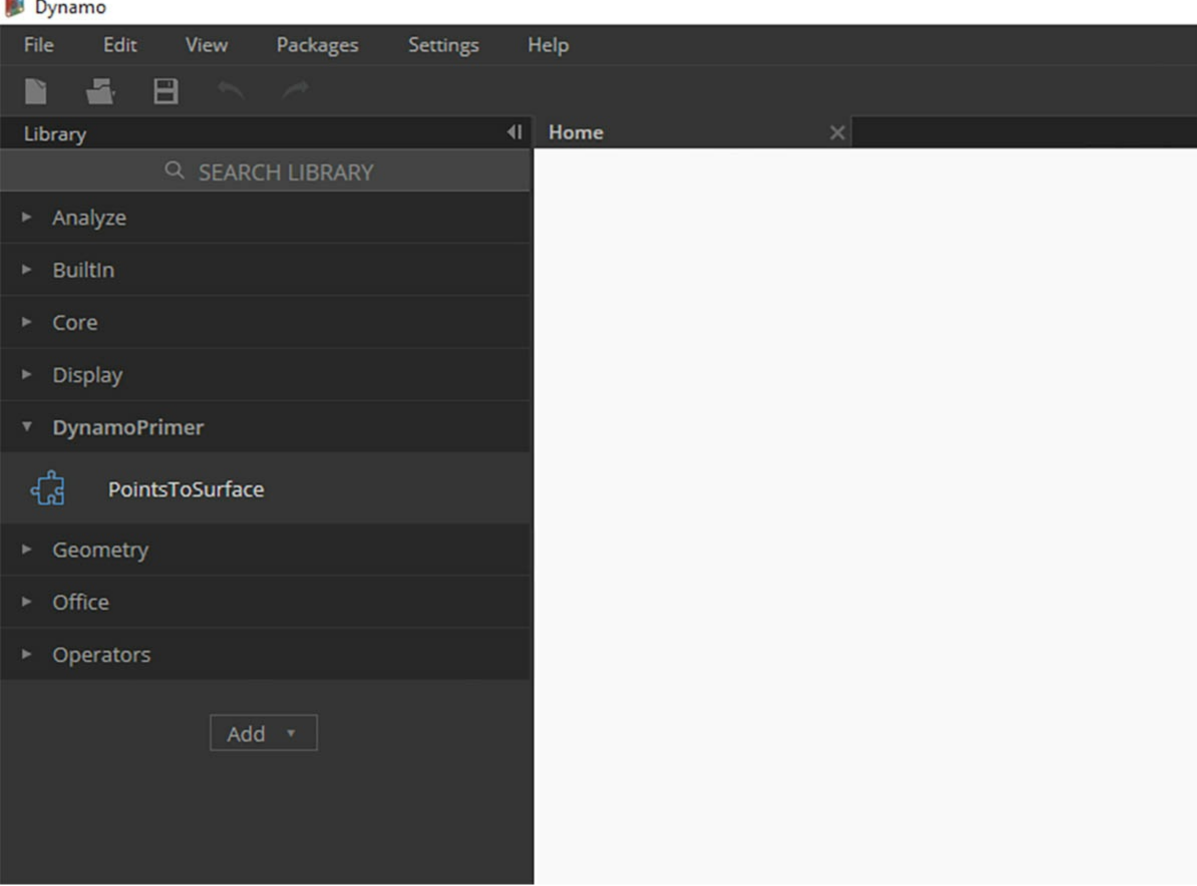
1. Рекомендуется переместить путь к локальной папке вниз по списку (для этого выберите путь к папке и щелкните стрелку вниз на панели слева от строк с путями). Верхняя папка всегда по умолчанию используется как расположение для установки пакетов. Поэтому если в качестве верхней папки будет указан путь для установки пакетов Dynamo, все полученные через интернет пакеты будут храниться отдельно от узлов, опубликованных локально*



Порядок путей к папкам изменен, чтобы в качестве пути по умолчанию в Dynamo использовалась папка установки пакетов.



Перейдите в эту локальную папку. Исходный пользовательский узел находится в папке *dyf*, имя которой является расширением для файлов пользовательских узлов Dynamo. Если отредактировать файл в этой папке, соответствующий узел будет обновлен в пользовательском интерфейсе. Кроме того, можно добавить дополнительные узлы в главную папку *DynamoCustomNode*, и после перезапуска Dynamo они появятся в библиотеке.



Теперь при каждой загрузке Dynamo узел *PointsToSurface* будет отображаться в группе *DynamoPrimer* библиотеки Dynamo.

Узлы Python

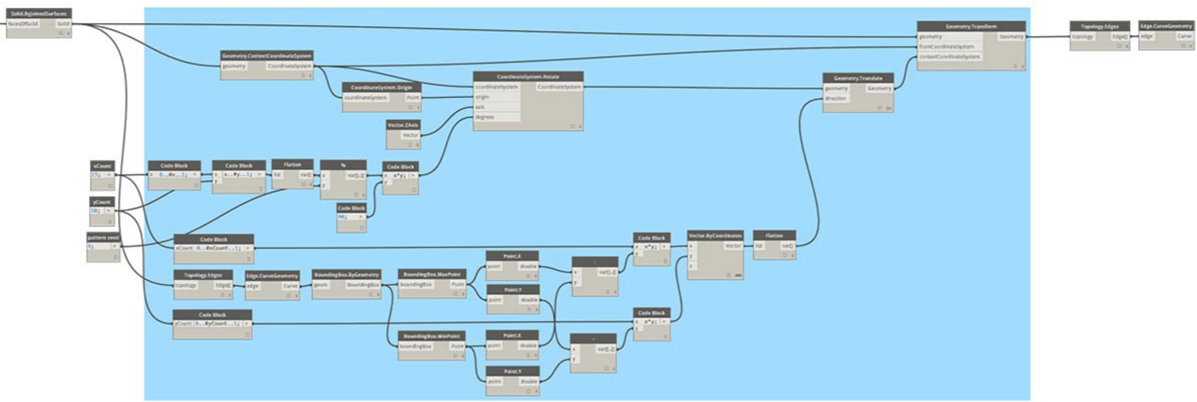
Python

Python — это широко распространенный язык программирования, получивший большую популярность благодаря стилю используемого синтаксиса. Этот язык значительно доступнее и проще, чем многие другие языки программирования. Python поддерживает модули и пакеты, а также может быть внедрен в существующие приложения. Примеры в этом разделе предполагают наличие базовых знаний о языке Python. Тем, кто только начинает работу с Python, мы рекомендуем посетить страницу Getting Started на веб-сайте Python.org.

Визуальное и текстовое программирование

Зачем использовать текстовое программирование в среде визуального программирования Dynamo? Как уже говорилось в главе 1.1, визуальное программирование имеет ряд преимуществ. Оно позволяет создавать программы в интуитивно-понятном визуальном интерфейсе, не обладая навыками работы со специальным синтаксисом. Однако визуальная программа может оказаться перегруженной, а порой и недостаточно функциональной. Для сравнения, в Python реализованы гораздо более доступные способы записи условных выражений (если/то) и создания циклов. Python — это мощный инструмент, который позволяет расширить возможности Dynamo и заменить большое количество узлов компактными строками кода.

Визуальная программа

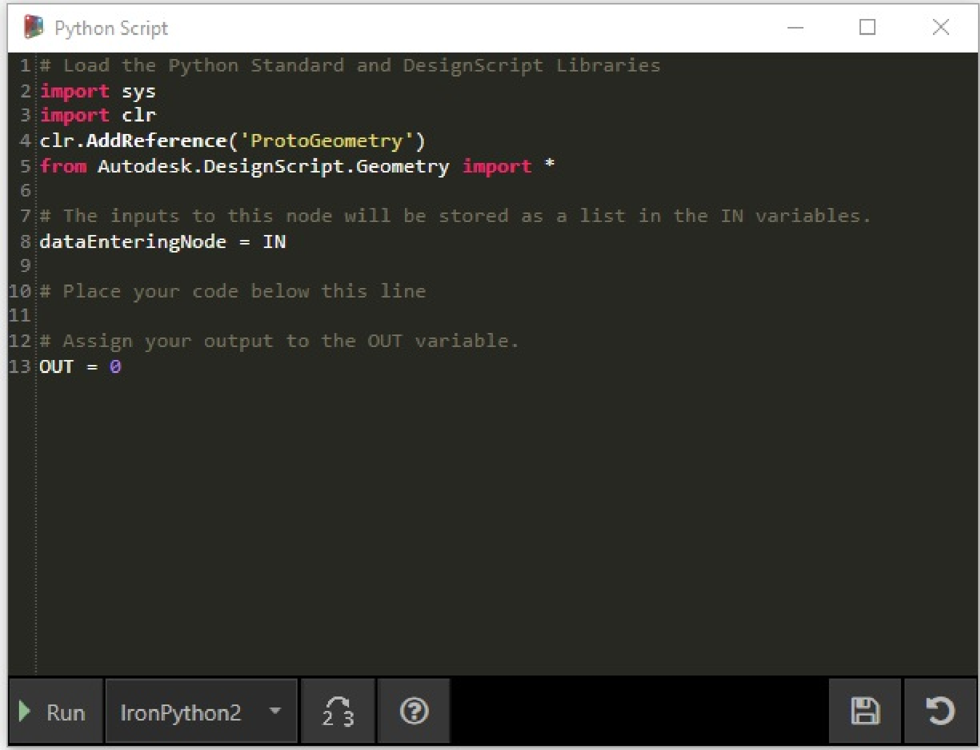


Текстовая программа

```
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

solid = IN[0]
seed = IN[1]
xCount = IN[2]
yCount = IN[3]

solids = []

yDist = solid.BoundingBox.MaxPoint.Y-solid.BoundingBox.MinPoint.Y
xDist = solid.BoundingBox.MaxPoint.X-solid.BoundingBox.MinPoint.X

for i in xRange:
for j in yRange:
fromCoord = solid.ContextCoordinateSystem
toCoord = fromCoord.Rotate(solid.ContextCoordinateSystem.Origin,Vector.ByCoordinates(0,0,1),(90*(i+j%val)
vec = Vector.ByCoordinates((xDist*i),(yDist*j),0)
toCoord = toCoord.Translate(vec)
solids.append(solid.Transform(fromCoord,toCoord))

OUT = solids
```

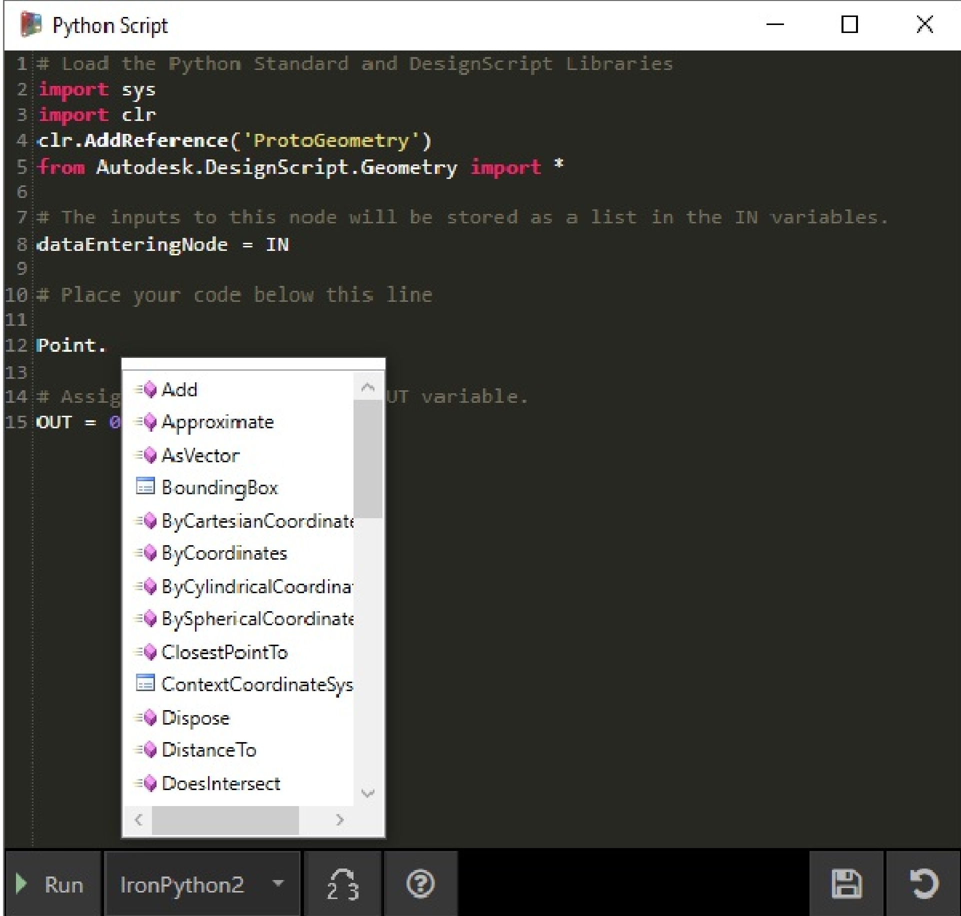
Узел Python

Подобно блокам кода узлы Python представляют собой интерфейс сценариев в среде визуального программирования. Узел Python находится в разделе *Core* > *Scripting* в библиотеке. Двойной щелчок на узле приводит к открытию редактора сценариев Python (можно также щелкнуть узел правой кнопкой мыши и выбрать команду *Редактировать*...).



Сверху находится подсказка, которая поможет обратиться к нужным библиотекам. Входные данные хранятся в массиве IN. Значения возвращаются в Dynamo путем назначения переменной OUT.

Библиотека Autodesk.DesignScript.Geometry позволяет использовать точечные обозначения, аналогичные блокам кода. Дополнительные сведения о синтаксисе Dynamo см. в главе 7.2, а также в Руководстве по DesignScript. При вводе определенного типа геометрии (например, Point.) отображается список методов, доступных для создания и запроса точек.

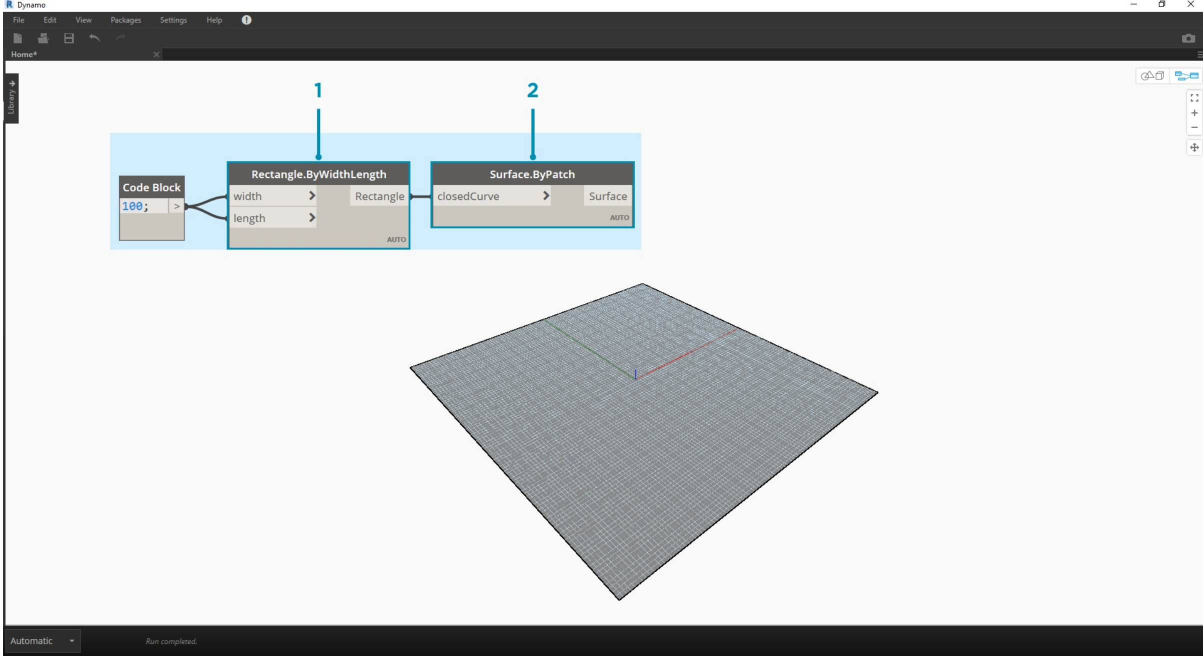


Методы включают в себя конструкторы (например, *ByCoordinates*), действия (например, *Add*) и запросы (например, координаты *X*, *Y* и *Z*).

Упражнение

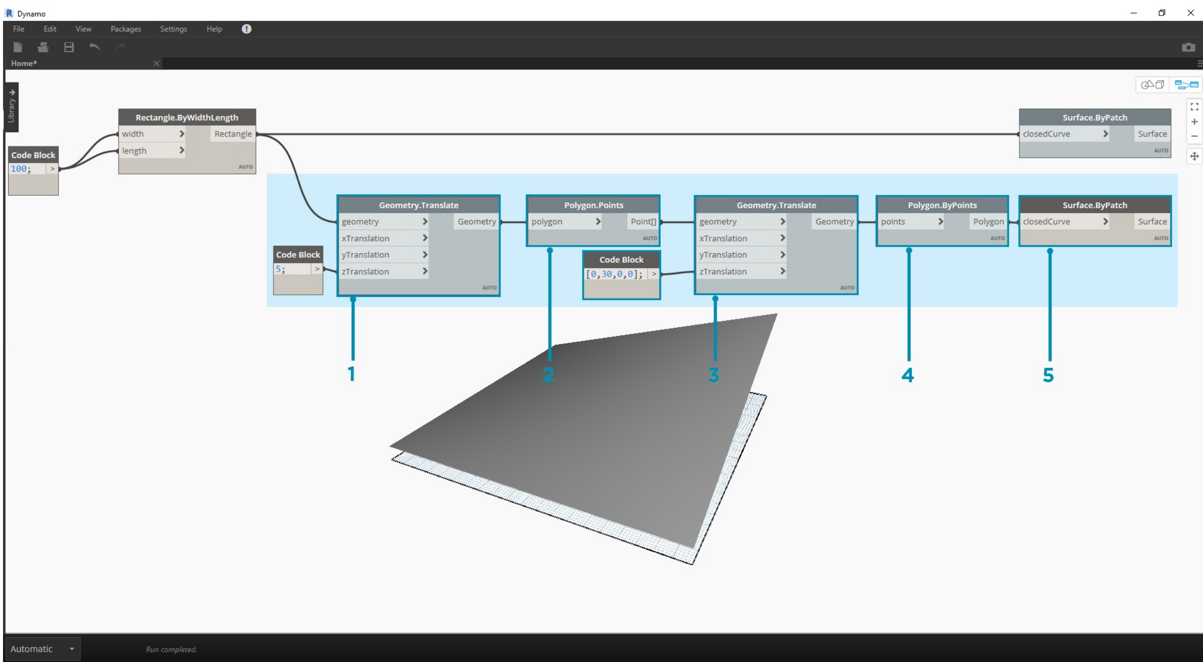
Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. Python_Custom-Node.dyn

В этом примере мы напишем сценарий Python для создания образцов на основе твердотельного модуля и преобразуем этот сценарий в пользовательский узел. Сначала создадим твердотельный модуль с помощью узлов Dynamo.



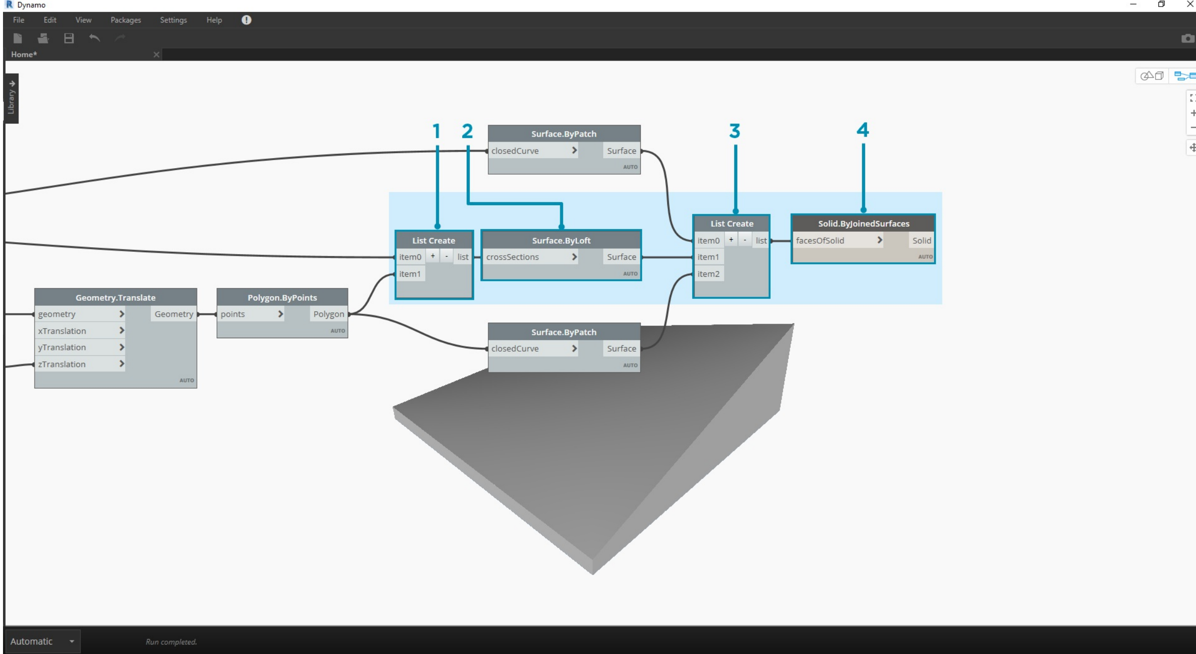
1. **Rectangle.ByWidthLength.** Создайте прямоугольник, который будет служить основой твердого тела.

2. **Surface.ByPatch.** Соедините прямоугольник с входным параметром *closedCurve* для создания нижней поверхности.



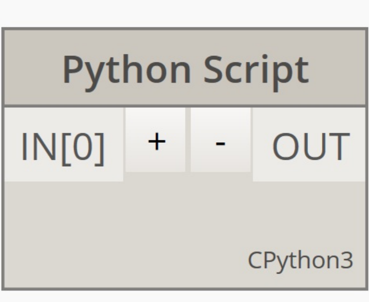
1. **Geometry.Translate.** Соедините прямоугольник с входным параметром *geometry* для его перемещения вверх, используя блок кода для указания толщины основания тела.
2. **Polygon.Points.** Запросите извлечение угловых точек из преобразованного прямоугольника.
3. **Geometry.Translate.** Используйте блок кода для создания списка из четырех значений, соответствующих четырем точкам, перемещающим один угол тела вверх.
4. **Polygon.ByPoints.** С помощью преобразованных точек воссоздайте верхний полигон.
5. **Surface.ByPatch.** Присоедините полигон для создания верхней поверхности.

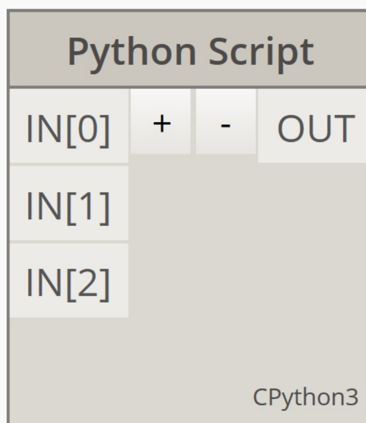
Теперь, имея в распоряжении верхнюю и нижнюю поверхности, выполним лофтинг между двумя профилями, чтобы создать стороны тела.



1. **List.Create.** Соедините нижний прямоугольник и верхний полигон с входными параметрами индекса.
2. **Surface.ByLoft.** Выполните лофтинг двух профилей для создания сторон тела.
3. **List.Create.** Соедините верхнюю, боковую и нижнюю поверхности с входными параметрами индекса для создания списка поверхностей.
4. **Solid.ByJoinedSurfaces.** Соедините поверхности для создания твердотельного модуля.

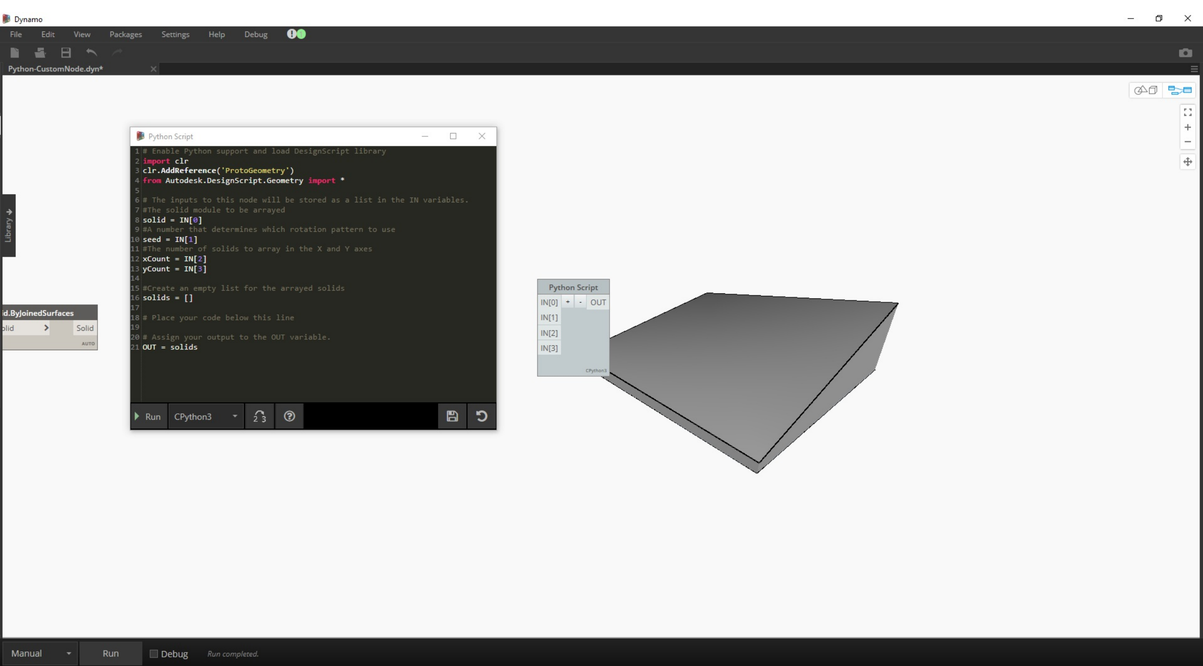
Теперь, получив твердое тело, перетащите в рабочее пространство узел сценария Python.





Чтобы добавить дополнительные входные параметры к узлу, закройте редактор и щелкните значок «+» на узле. Входным параметрам присваиваются имена IN[0], IN[1] и т. д. Это говорит о том, что они представляют элементы в списке.

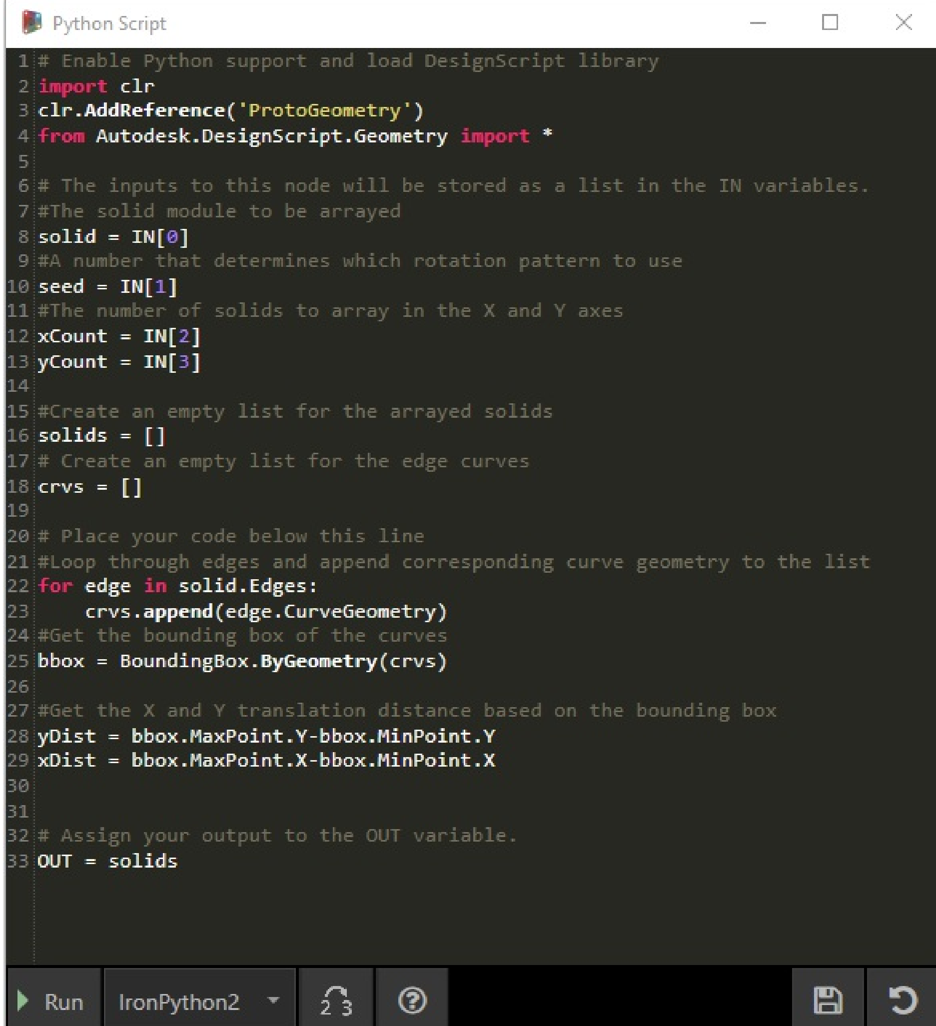
Начнем с определения входных и выходных параметров. Дважды щелкните узел, чтобы открыть редактор Python.




```
# Enable Python support and load DesignScript library
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.
#The solid module to be arrayed
solid = IN[0]
#A number that determines which rotation pattern to use
seed = IN[1]
#The number of solids to array in the X and Y axes
xCount = IN[2]
yCount = IN[3]

#Create an empty list for the arrayed solids
solids = []

# Place your code below this line
```

```
# Assign your output to the OUT variable.
OUT = solids
```

Смысл этого кода будет понятен позже по мере выполнения упражнения. Далее необходимо подумать о том, какая информация необходима для создания массива на основе имеющегося твердотельного модуля. Во-первых, необходимо знать размеры тела, чтобы определить расстояние переноса. Из-за ошибки, связанной с ограничивающей рамкой, для ее создания необходимо использовать геометрию кривой кромки.



Пример узла Python в Dynamo. Обратите внимание, что используется тот же синтаксис, что и в заголовках узлов Dynamo. Код с комментариями показан ниже.

```
# Enable Python support and load DesignScript library
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.
#The solid module to be arrayed
solid = IN[0]
#A number that determines which rotation pattern to use
seed = IN[1]
#The number of solids to array in the X and Y axes
xCount = IN[2]
yCount = IN[3]

#Create an empty list for the arrayed solids
solids = []
# Create an empty list for the edge curves
crvs = []

# Place your code below this line
#Loop through edges and append corresponding curve geometry to the list
for edge in solid.Edges:
crvs.append(edge.CurveGeometry)
#Get the bounding box of the curves
bbox = BoundingBox.ByGeometry(crvs)
```

```
#Get the X and Y translation distance based on the bounding box
yDist = bbox.MaxPoint.Y-bbox.MinPoint.Y
xDist = bbox.MaxPoint.X-bbox.MinPoint.X

# Assign your output to the OUT variable.
OUT = solids
```

Поскольку твердотельные модули будут не только преобразовываться, но и поворачиваться, воспользуемся операцией Geometry.Transform. Если посмотреть на узел Geometry.Transform, становится понятно, что для преобразования тела потребуется исходная система координат и целевая система координат. В качестве первой выступает контекстная система координат тела, а в качестве второй — система координат, которая различна для каждого модуля массива. Таким образом, чтобы система координат каждый раз преобразовывалась по-разному, необходимо перебирать значения координат по осям X и Y.

Python Script

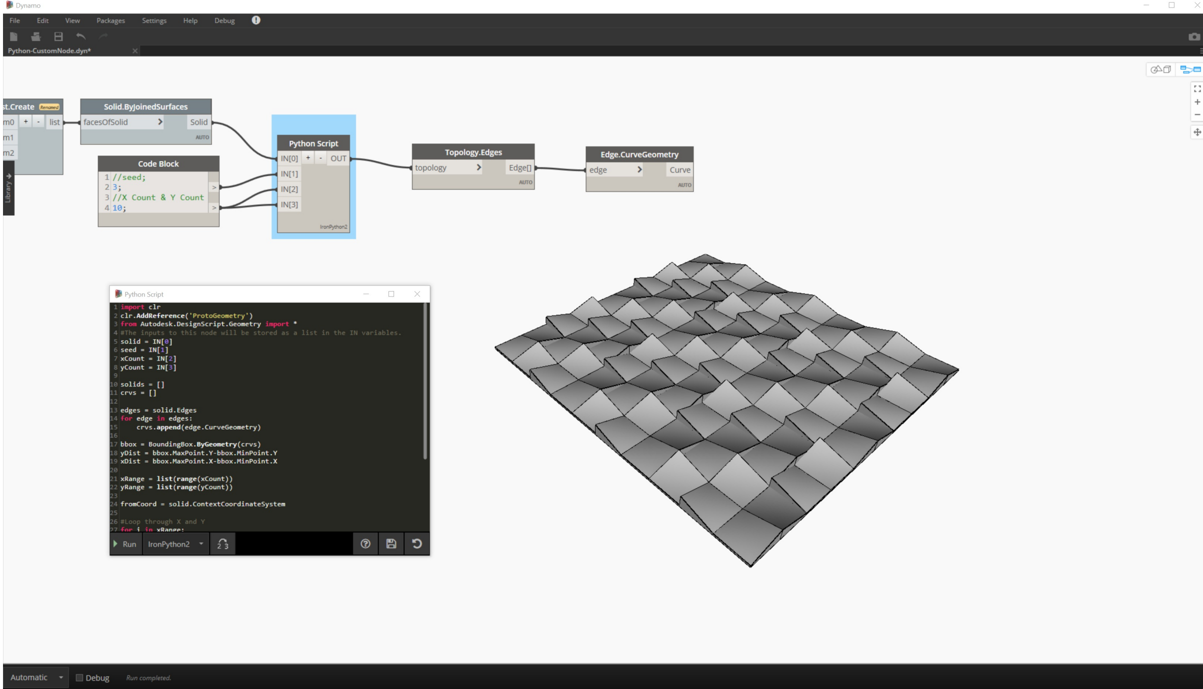
```
# Enable Python support and load DesignScript library
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.
#The solid module to be arrayed
solid = IN[0]
#A number that determines which rotation pattern to use
seed = IN[1]
#The number of solids to array in the X and Y axes
xCount = IN[2]
yCount = IN[3]

#Create an empty list for the arrayed solids
solids = []
# Create an empty list for the edge curves
crvs = []

# Place your code below this line
#Loop through edges and append corresponding curve geometry to the list
for edge in solid.Edges:
    crvs.append(edge.CurveGeometry)
#Get the bounding box of the curves
bbox = BoundingBox.ByGeometry(crvs)

#Get the X and Y translation distance based on the bounding box
yDist = bbox.MaxPoint.Y-bbox.MinPoint.Y
xDist = bbox.MaxPoint.X-bbox.MinPoint.X
#get the source coordinate system
fromCoord = solid.ContextCoordinateSystem

#Loop through X and Y
for i in range(xCount):
    for j in range(yCount):
        #Rotate and translate the coordinate system
        toCoord = fromCoord.Rotate(solid.ContextCoordinateSystem.Origin,Vector.ByCoordinates(0,0,1),(90*(i+j%seed)))
        vec = Vector.ByCoordinates((xDist*i),(yDist*j),0)
        toCoord = toCoord.Translate(vec)
        #Transform the solid from the source coord system to the target coord system and append to the list
        solids.append(solid.Transform(fromCoord,toCoord))

# Assign your output to the OUT variable.
OUT = solids
```

Run IronPython2

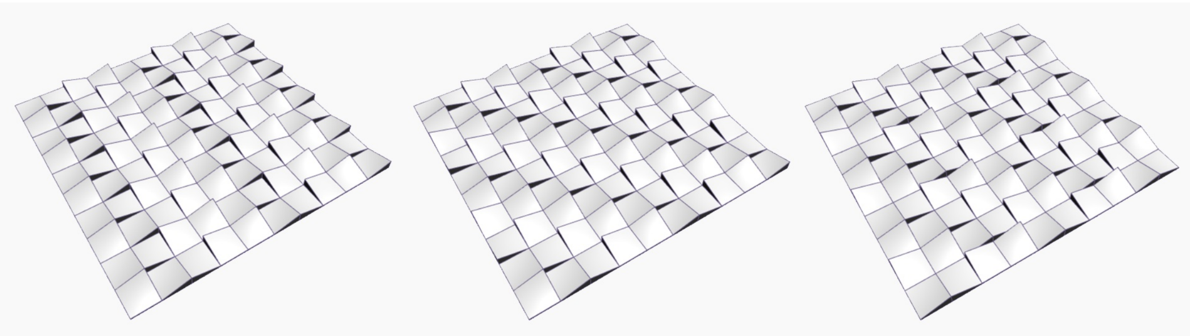
Пример узла Python в Dynamo. Код с комментариями показан ниже.

```
# Enable Python support and load DesignScript library
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.
#The solid module to be arrayed
solid = IN[0]
#A number that determines which rotation pattern to use
seed = IN[1]
#The number of solids to array in the X and Y axes
xCount = IN[2]
yCount = IN[3]

#Create an empty list for the arrayed solids
solids = []
# Create an empty list for the edge curves
crvs = []

# Place your code below this line
#Loop through edges and append corresponding curve geometry to the list
```

```
for edge in solid.Edges:
crvs.append(edge.CurveGeometry)
#Get the bounding box of the curves
bbox = BoundingBox.ByGeometry(crvs)

#Get the X and Y translation distance based on the bounding box
yDist = bbox.MaxPoint.Y-bbox.MinPoint.Y
xDist = bbox.MaxPoint.X-bbox.MinPoint.X
#get the source coordinate system
fromCoord = solid.ContextCoordinateSystem

#Loop through X and Y
for i in range(xCount):
for j in range(yCount):
#Rotate and translate the coordinate system
toCoord = fromCoord.Rotate(solid.ContextCoordinateSystem.Origin,Vector.ByCoordinates(0,0,1),(90*(i+j%seed
vec = Vector.ByCoordinates((xDist*i),(yDist*j),0)
toCoord = toCoord.Translate(vec)
#Transform the solid from the source coord system to the target coord system and append to the list
solids.append(solid.Transform(fromCoord,toCoord))

# Assign your output to the OUT variable.
OUT = solids
```

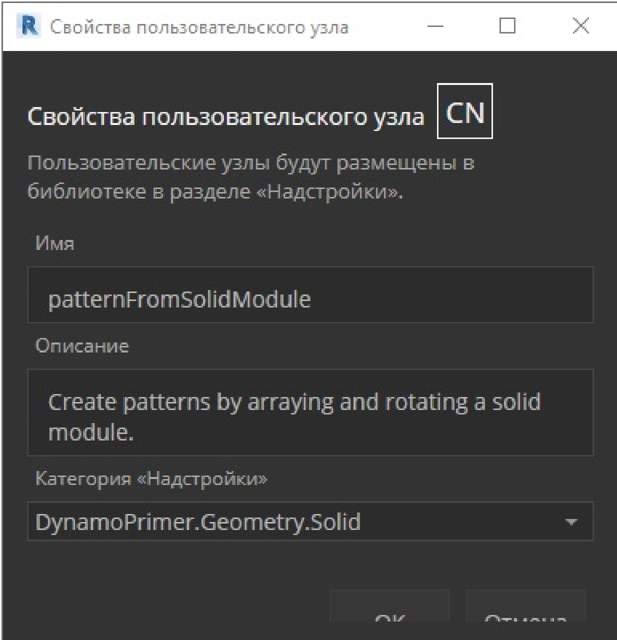


Чтобы выполнить код, щелкните кнопку Run (запуск) в узле Python.



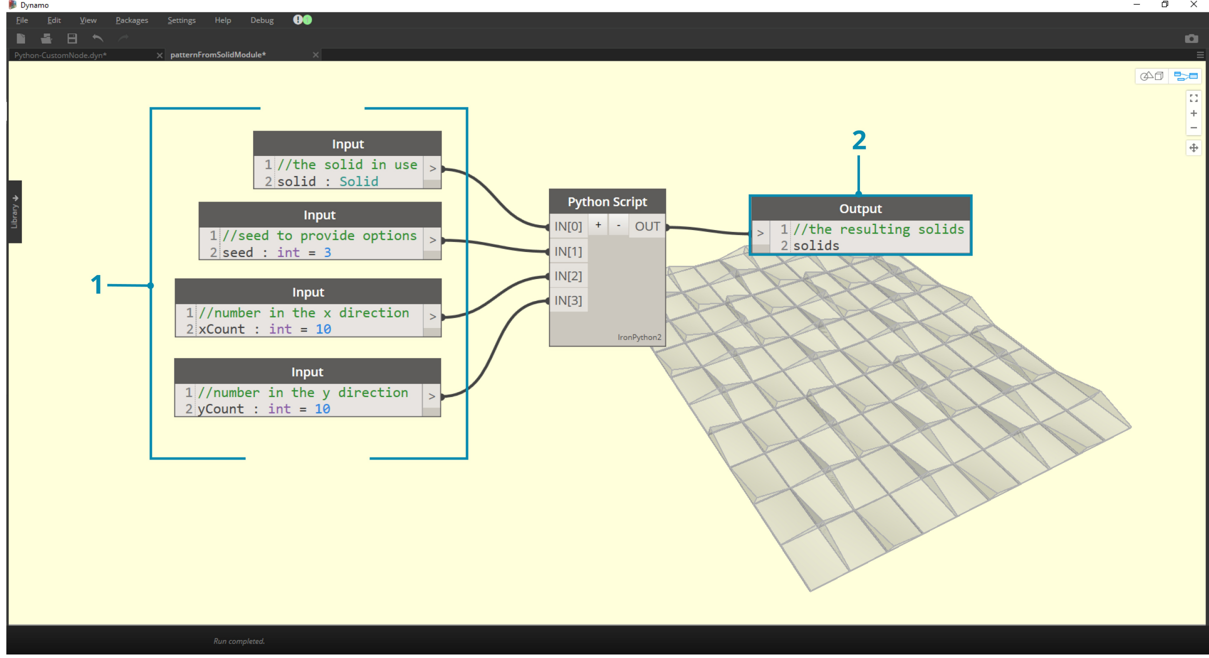
Попробуйте изменить начальное значение для создания различных образцов. Кроме того, можно изменять параметры самого твердотельного модуля для получения различных эффектов. В Dynamo 2.0 можно просто изменить прототип и запустить его, не закрывая окно Python.

Создав нужный сценарий Python, сохраним его как пользовательский узел. Выберите узел сценария Python, щелкните правой кнопкой мыши и выберите создание узла из выбранных объектов.

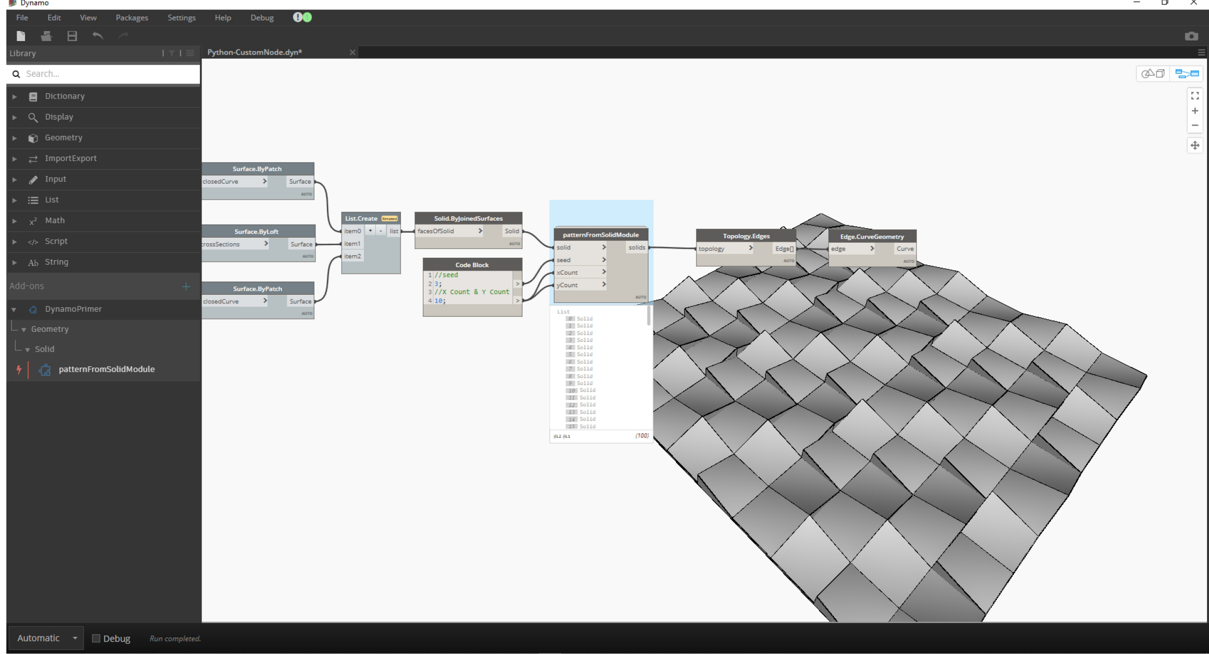


Присвойте имя, добавьте описание и категорию.

При этом откроется новое рабочее пространство для редактирования пользовательского узла.



1. **Входные параметры.** Измените имена входных параметров, сделав их более описательными, и добавьте типы данных и значения по умолчанию.
2. **Выходной параметр.** Измените имя выходного параметра и сохраните узел в виде файла DYF.



Внесенные изменения появятся в пользовательском узле.

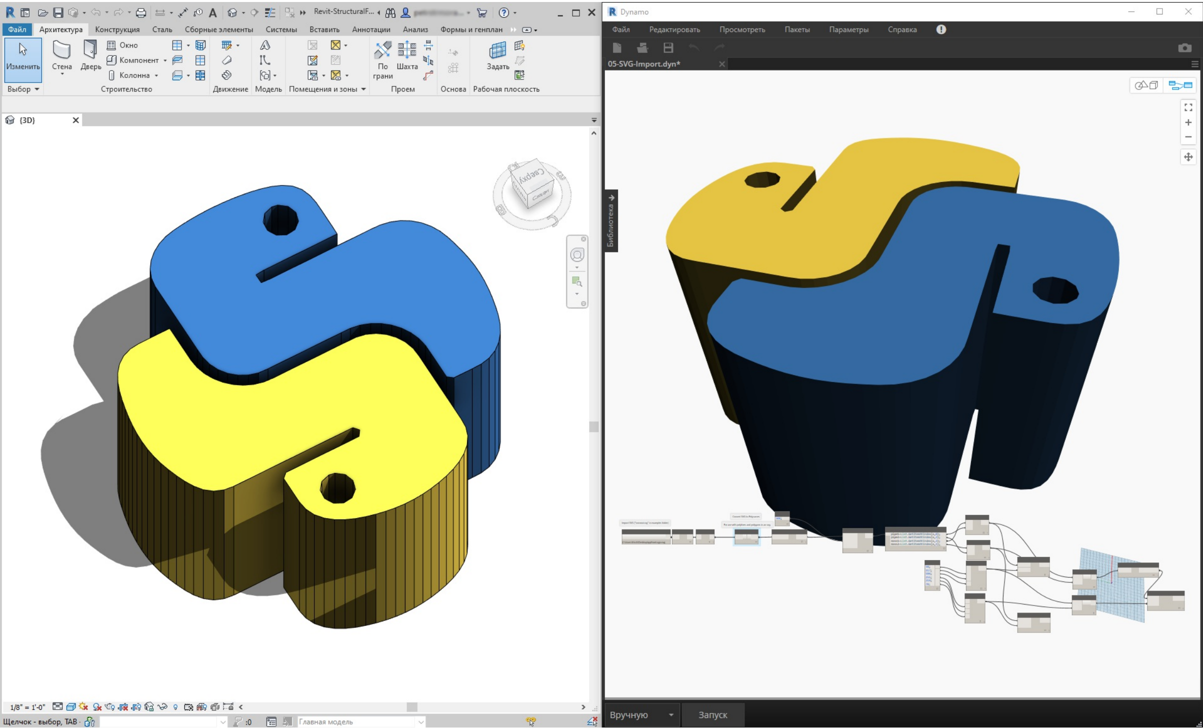
Python и Revit

Python и Revit

В предыдущем разделе был приведен пример использования сценариев Python в Dynamo. Теперь рассмотрим подключение библиотек Revit в среде сценариев. Как вы помните, базовые узлы Dynamo были импортированы с помощью первых трех строк в блоке кода, представленном ниже. Для импорта узлов Revit, элементов Revit и диспетчера документов Revit необходимо добавить еще несколько строк:

```
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# Import RevitNodes
clr.AddReference("RevitNodes")
import Revit

# Import Revit elements
from Revit.Elements import *

# Import DocumentManager
clr.AddReference("RevitServices")
import RevitServices
from RevitServices.Persistence import DocumentManager

import System
```

Это обеспечит доступ к API Revit и позволит создавать пользовательские сценарии для любых задач Revit. Благодаря объединению процесса визуального программирования с написанием сценариев в API Revit возможности совместной работы и разработки инструментов значительно увеличиваются. Например, специалист по BIM и проектировщик схем могут совместно работать над одним и тем же графиком. В результате эффективность проектирования и реализации модели повысится.



API-интерфейсы для конкретных платформ

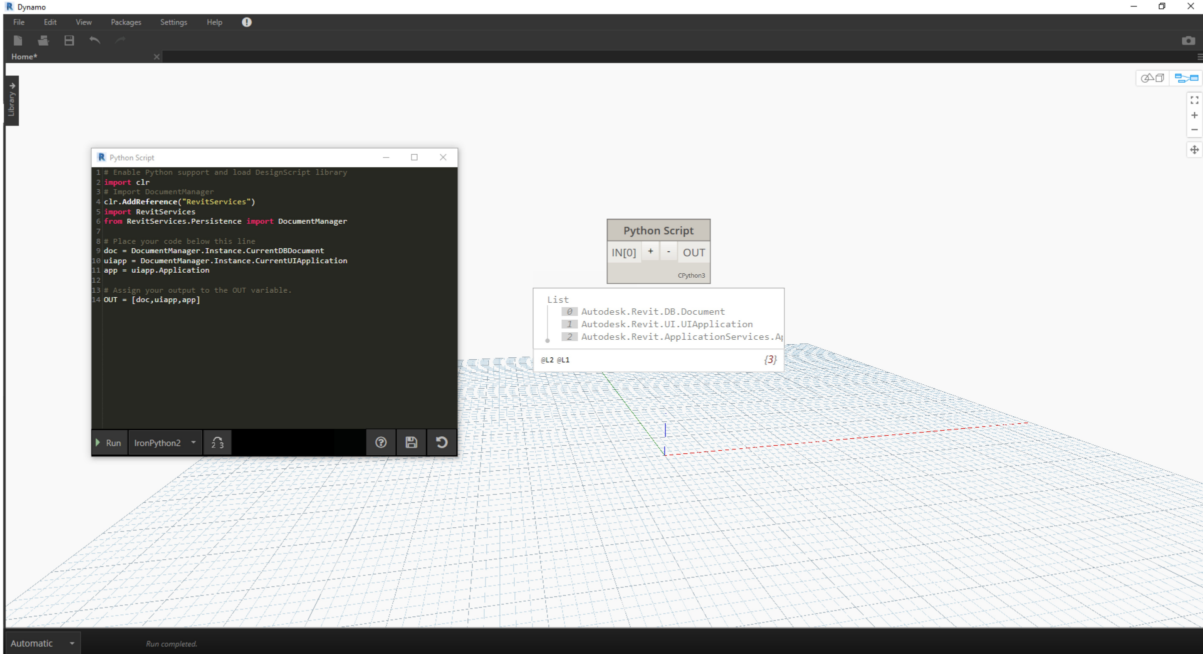
В основе проекта Dynamo лежит план по расширению масштабов внедрения платформ. По мере добавления в Dynamo поддержки новых программ пользователи получают доступ к API-интерфейсам для конкретных платформ из среды создания сценариев Python. Хотя этот раздел посвящен работе с Revit, в будущем можно ожидать появления новых разделов, содержащих учебные пособия по созданию сценариев для других платформ. Кроме того, в данный момент доступно множество библиотек IronPython, которые можно импортировать в Dynamo.

В приведенных ниже примерах иллюстрируются способы выполнения операций в Revit из модуля Dynamo с использованием языка программирования Python. Дополнительные сведения об особенностях использования Python в Dynamo и Revit см. на странице Wiki, посвященной Dynamo. Еще один полезный ресурс по Python и Revit — проект Revit Python Shell.

Упражнение 01

Создайте новый проект Revit. Скачайте файл примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. Revit-Doc.dyn

В этих упражнениях рассматриваются простейшие сценарии Python, которые можно использовать в модуле Dynamo для Revit. Данное упражнение посвящено работе с файлами и элементами Revit, а также взаимодействию между Revit и Dynamo.



Рассмотрим стандартный способ извлечения элементов *doc*, *uiapp* и *app* из файла Revit, связанного сеансом Dynamo. Программистам, которые уже работали с API Revit, могут быть знакомы элементы в списке наблюдения. Однако даже если эти элементы встречаются в первый раз, в последующих упражнениях будут и другие примеры.

Ниже описывается процедура импорта служб Revit и извлечения данных документа в Dynamo.

Python Script

```
1 # Enable Python support and load DesignScript library
2 import clr
3 # Import DocumentManager
4 clr.AddReference("RevitServices")
5 import RevitServices
6 from RevitServices.Persistence import DocumentManager
7
8 # Place your code below this line
9 doc = DocumentManager.Instance.CurrentDBDocument
10 uiapp = DocumentManager.Instance.CurrentUIApplication
11 app = uiapp.Application
12
13 # Assign your output to the OUT variable.
14 OUT = [doc,uiapp,app]
```

Run IronPython2

Пример узла Python в Dynamo. Код с комментариями показан ниже.

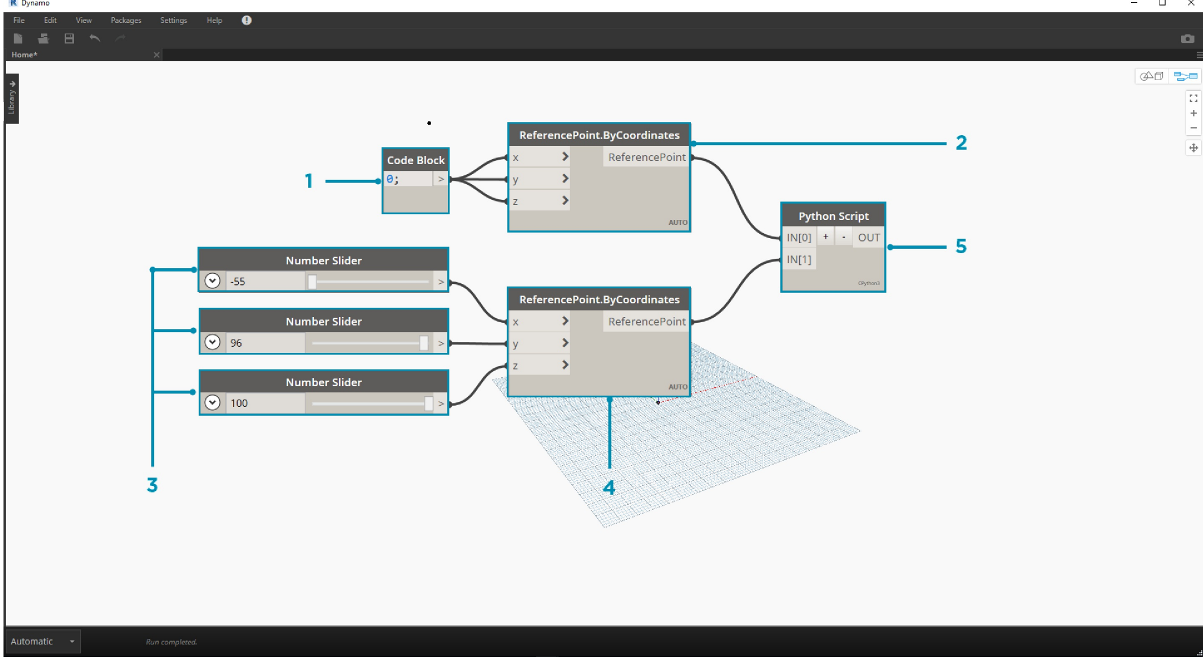
```
# Enable Python support and load DesignScript library
import clr
# Import DocumentManager
clr.AddReference("RevitServices")
```

```
import RevitServices
from RevitServices.Persistence import DocumentManager

# Place your code below this line
doc = DocumentManager.Instance.CurrentDBDocument
uiapp = DocumentManager.Instance.CurrentUIApplication
app = uiapp.Application

# Assign your output to the OUT variable.
OUT = [doc,uiapp,app]
```

Упражнение 02

> Скачайте файлы примера для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. Revit-ReferenceCurve.dyn

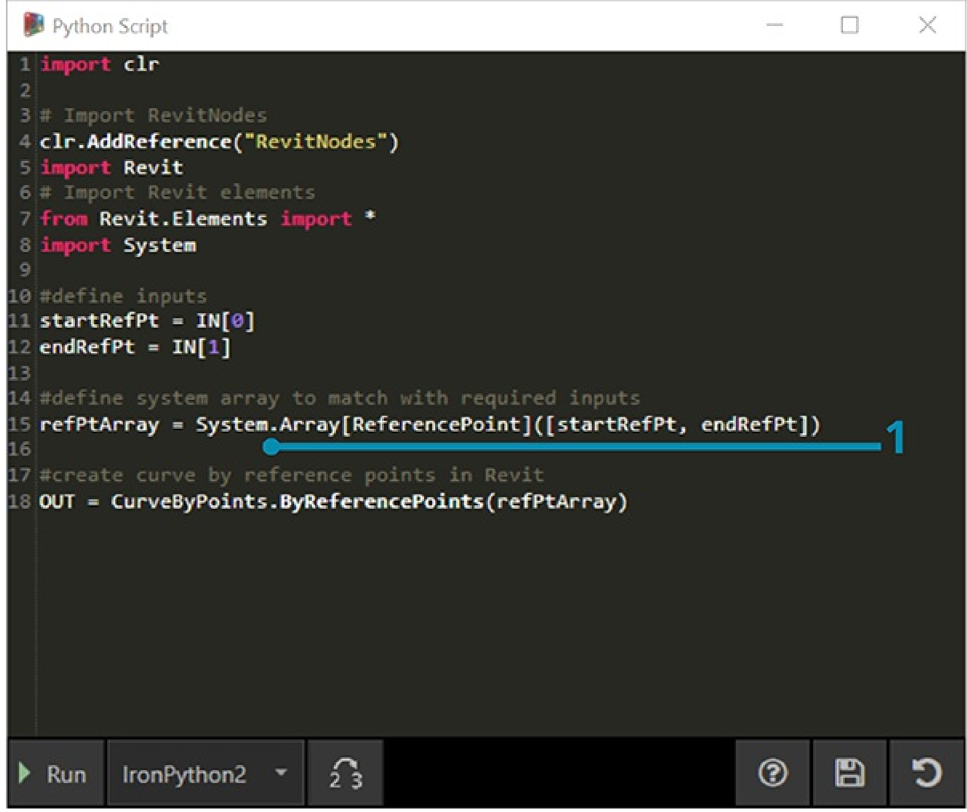
В этом упражнении будет создана простая кривая модели в Revit с помощью узла Python в Dynamo.



Начните с набора узлов, представленных на изображении выше. Сначала создайте две опорные точки в Revit с помощью узлов Dynamo.

Далее добавьте в Revit новое семейство концептуальных формообразующих элементов. Запустите Dynamo и сформируйте набор узлов, как показано на изображении выше. Сначала создайте в Revit две опорные точки с помощью узлов Dynamo.

1. Создайте блок кода и присвойте ему значение «0;».
2. Соедините это значение с входными параметрами X, Y и Z узла ReferencePoint.ByCoordinates.
3. Создайте три регулятора в диапазоне от -100 до 100 с шагом 1.
4. Соедините каждый из регуляторов с узлом ReferencePoint.ByCoordinates.
5. Добавьте в рабочее пространство узел Python, нажмите кнопку «+» в узле, чтобы добавить еще один входной параметр, и соедините опорные точки с входными параметрами. Откройте узел Python.

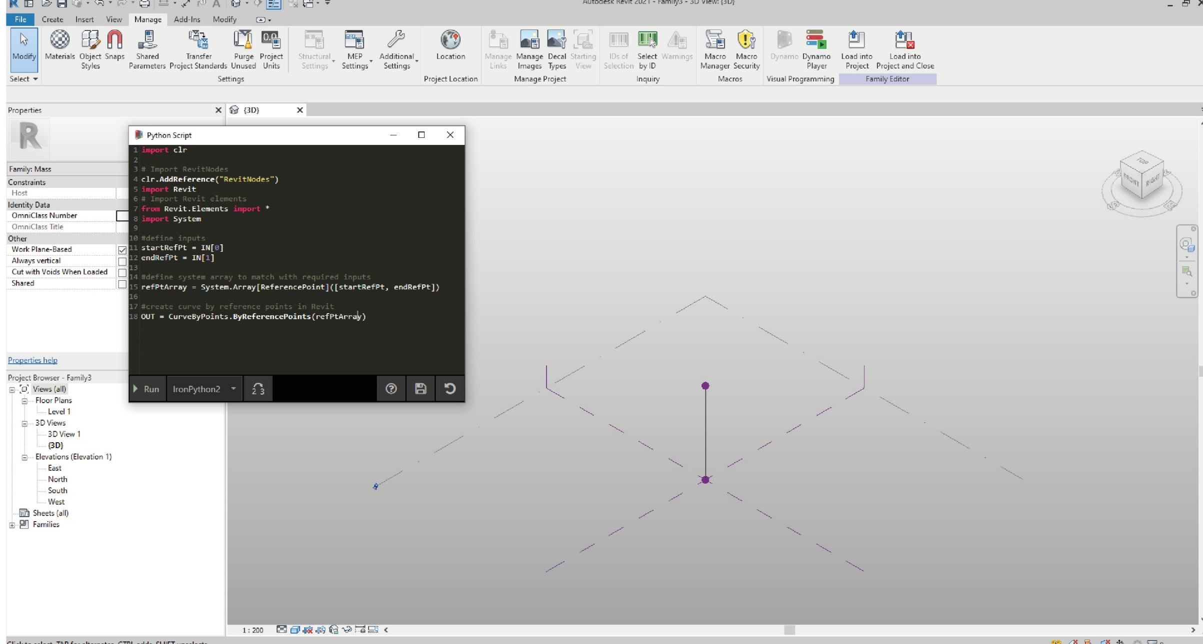


Пример узла Python в Dynamo. Код с комментариями показан ниже.

1. **System.Array.** Приложению Revit в качестве входного параметра требуется системный массив (а не список Python). Для этого необходима лишь еще одна строка кода, но следует уделить особое внимание типам аргументов, чтобы упростить программирование

на языке Python в Revit.

```
import clr

# Import RevitNodes
clr.AddReference("RevitNodes")
import Revit
# Import Revit elements
from Revit.Elements import *
import System

#define inputs
startRefPt = IN[0]
endRefPt = IN[1]

#define system array to match with required inputs
refPtArray = System.Array[ReferencePoint]([startRefPt, endRefPt])
#create curve by reference points in Revit
OUT = CurveByPoints.ByReferencePoints(refPtArray)
```

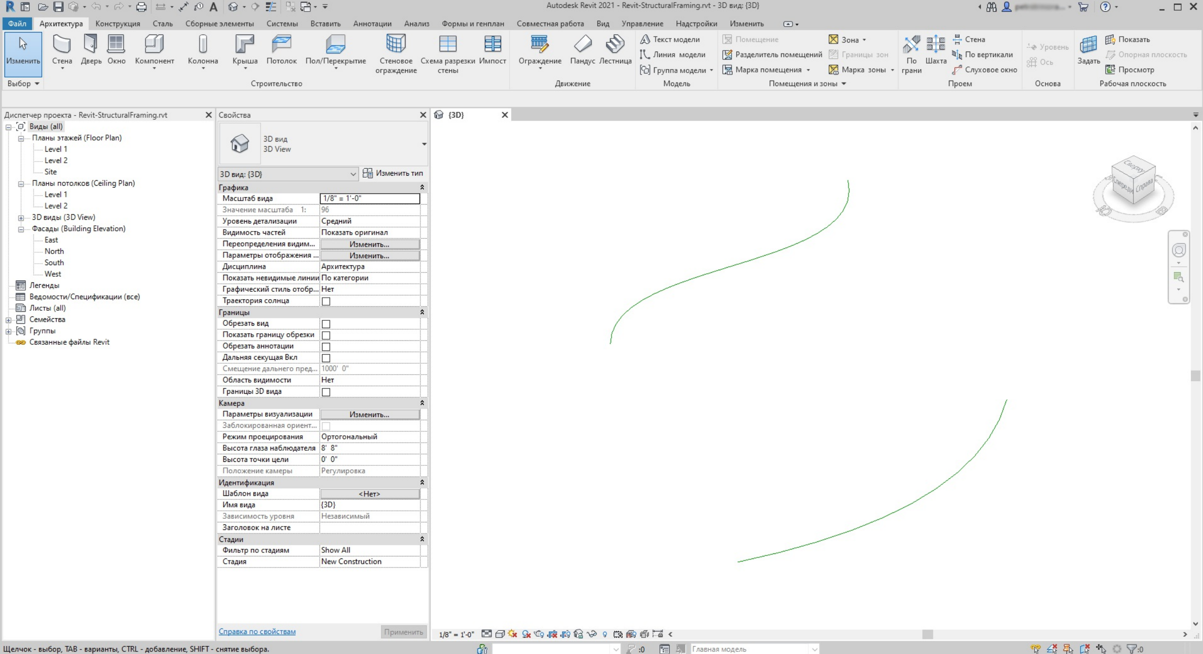


В Dynamo с помощью Python мы создали две опорные точки, соединенные линией. Продолжим работу с этим примером в следующем упражнении.

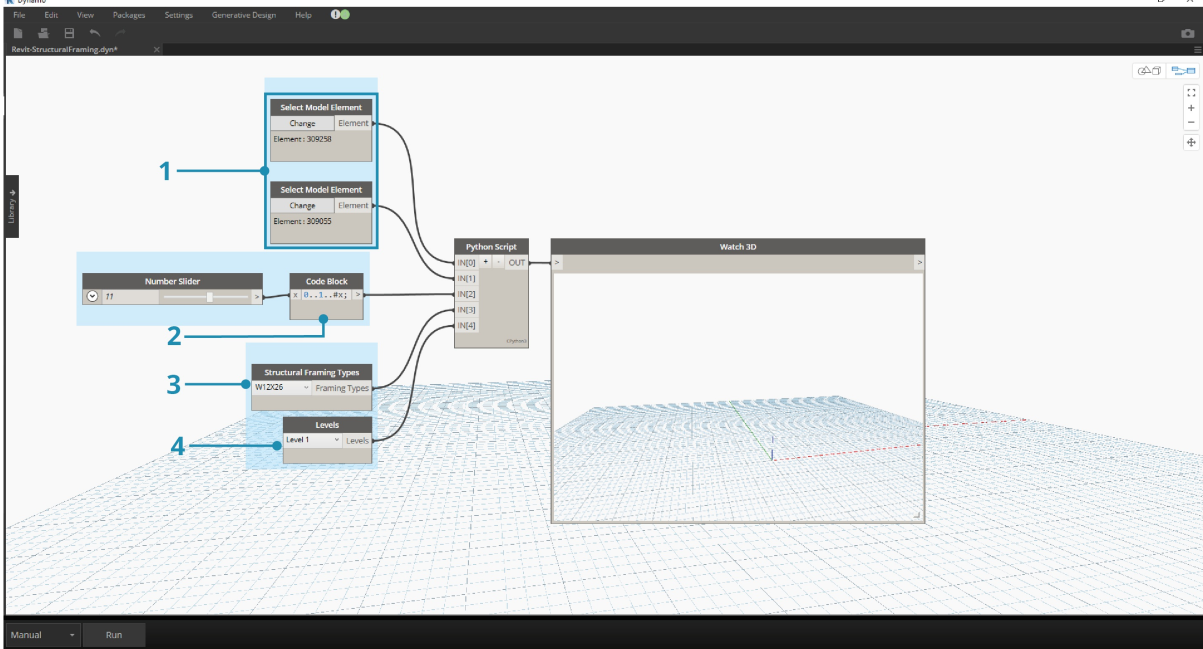
Упражнение 03

Скачайте и распакуйте файлы примеров для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. [Revit-StructuralFraming.zip]

Это упражнение довольно несложное, однако оно хорошо иллюстрирует процесс обмена данными и геометрией между Revit и Dynamo. Сначала откройте файл Revit-StructuralFraming.rvt. Затем загрузите Dynamo и откройте файл Revit-StructuralFraming.dyn.

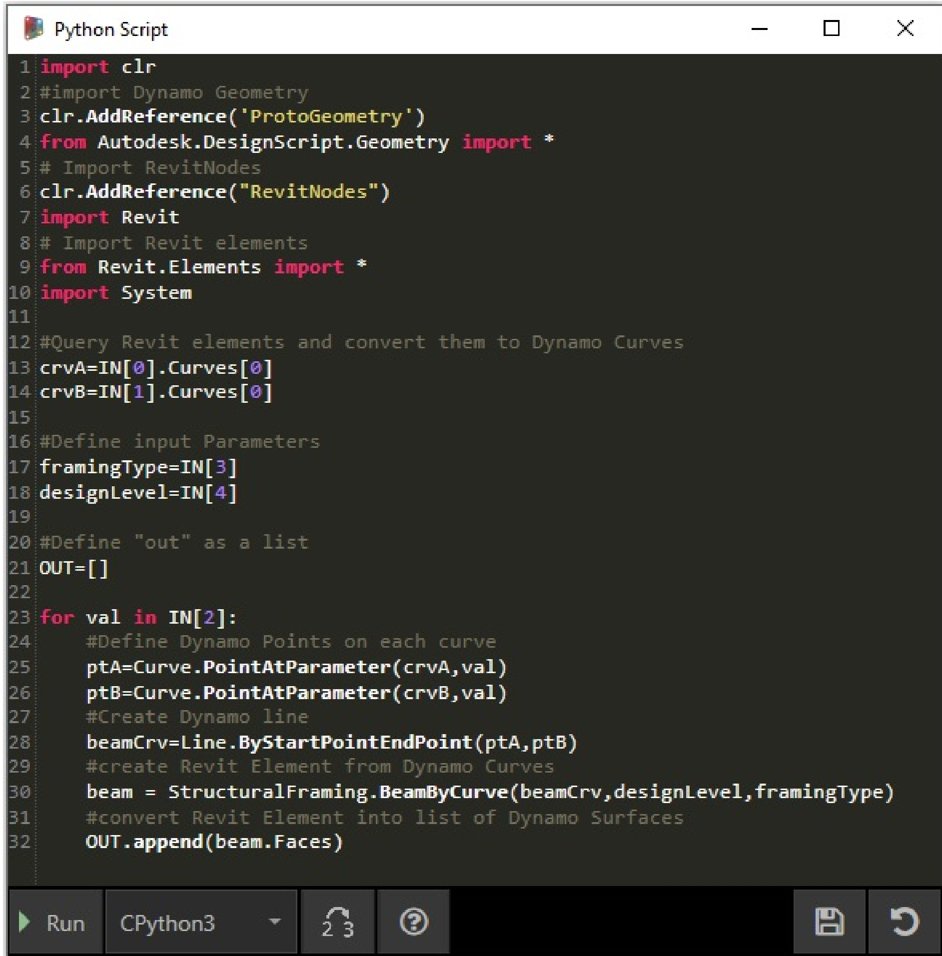


Этот файл Revit содержит лишь самые базовые данные. Имеется две опорные кривые: одна на уровне 1, другая — на уровне 2. Эти кривые необходимо добавить в Dynamo, сохранив динамическую связь.



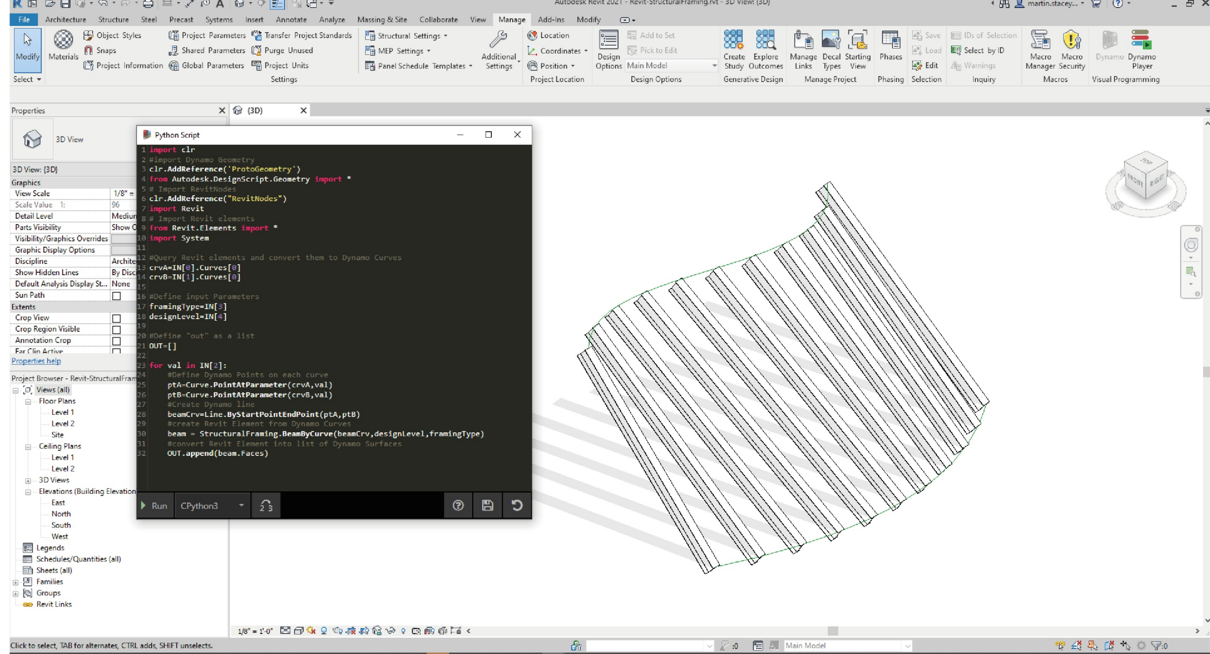
В файле имеется набор узлов, соединяемых с пятью входными параметрами узла Python.

1. **Выбор узлов для элементов модели.** Нажмите кнопку выбора для каждого узла и выберите соответствующую кривую в Revit.
2. **Блок кода.** Используя синтаксис *0..1..#x;*, соедините регулятор целых чисел от 0 до 20 с входным параметром *x*. Этот регулятор задает количество балок, которые будут построены между двумя кривыми.
3. **Типы несущих каркасов.** В раскрывающемся меню выберите балку по умолчанию W12x26.
4. **Уровни.** Выберите Level 1.

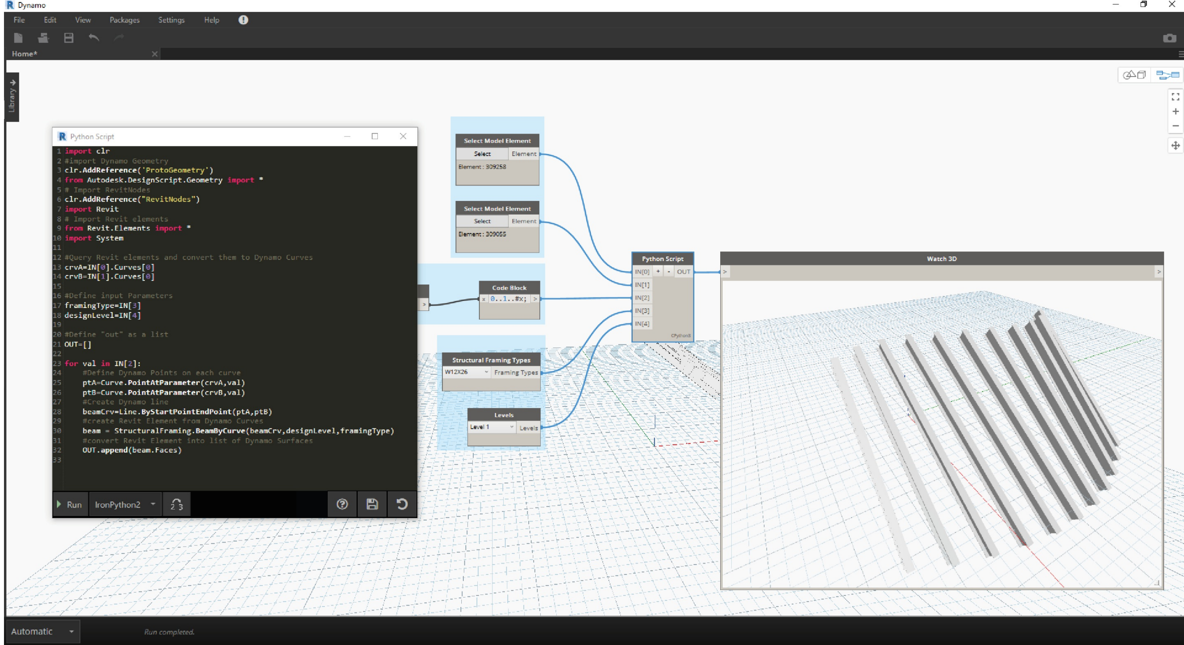


Этот код Python более сложен, но весь процесс снабжен подробными комментариями:

```
import clr
#import Dynamo Geometry
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
# Import RevitNodes
clr.AddReference("RevitNodes")
import Revit
# Import Revit elements
from Revit.Elements import *
import System

#Query Revit elements and convert them to Dynamo Curves
crvA=IN[0].Curves[0]
crvB=IN[1].Curves[0]

#Define input Parameters
framingType=IN[3]
designLevel=IN[4]

#Define "out" as a list
OUT=[]

for val in IN[2]:
#Define Dynamo Points on each curve
ptA=Curve.PointAtParameter(crvA,val)
ptB=Curve.PointAtParameter(crvB,val)
#Create Dynamo line
beamCrv=Line.ByStartPointEndPoint(ptA,ptB)
#create Revit Element from Dynamo Curves
beam = StructuralFraming.BeamByCurve(beamCrv,designLevel,framingType)
#convert Revit Element into list of Dynamo Surfaces
OUT.append(beam.Faces)
```



Итак, в Revit имеется массив балок, расположенных между двумя кривыми, служащими несущими элементами. Примечание. Данный пример не вполне реалистичен. Несущие элементы используются всего лишь в качестве примера собственных экземпляров Revit, созданных в Dynamo.

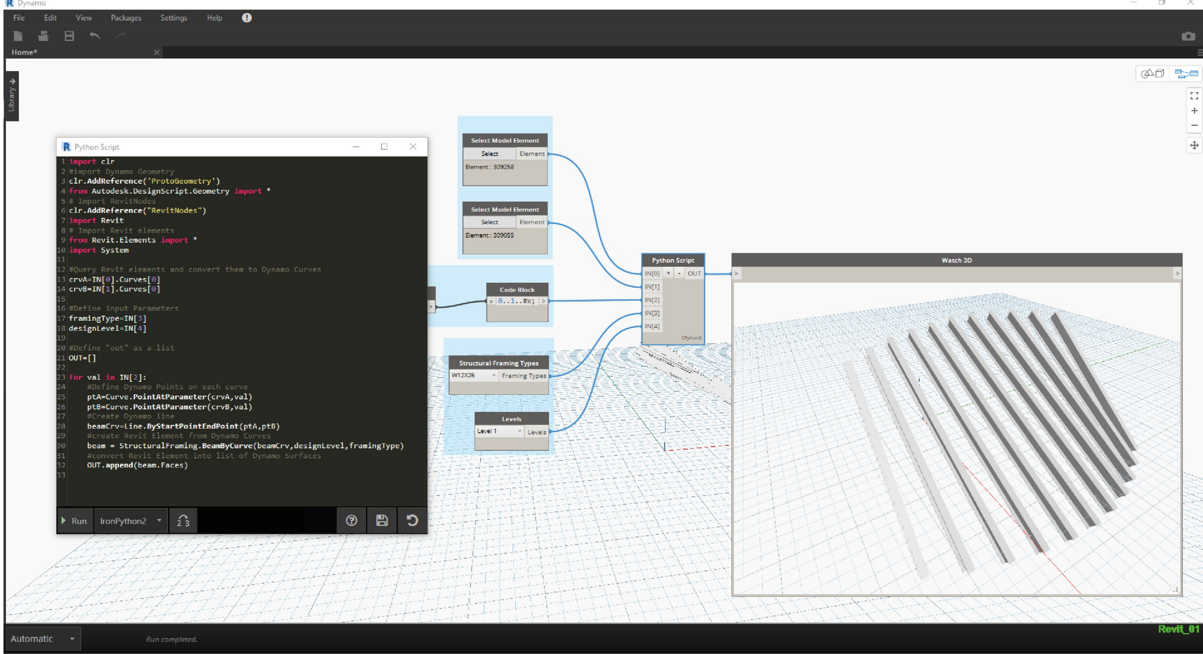


В Dynamo можно также увидеть результаты. Балки в узле Watch3D ссылаются на геометрию, запрошенную из элементов Revit.

Обратите внимание на непрерывный процесс преобразования данных из среды Revit в среду Dynamo. Вкратце этот процесс происходит следующим образом:

1. выбор элемента Revit;
2. преобразование элемента Revit в кривую Dynamo;
3. разделение кривой Dynamo на серию точек Dynamo;
4. использование точек Dynamo между двумя кривыми для создания линий Dynamo;
5. создание балок Revit на основе линий Dynamo;
6. вывод поверхностей Dynamo путем запроса геометрии балок Revit.

На вид процесс может казаться довольно сложным, но при использовании сценария достаточно всего лишь отредактировать кривую в Revit и повторно запустить решатель (хотя при этом может потребоваться удалить ранее созданные балки). *Дело в том, что когда балки размещаются в Python, связи стандартных узлов по умолчанию разрываются.*



Обновив опорные кривые в Revit, мы получим новый массив балок.

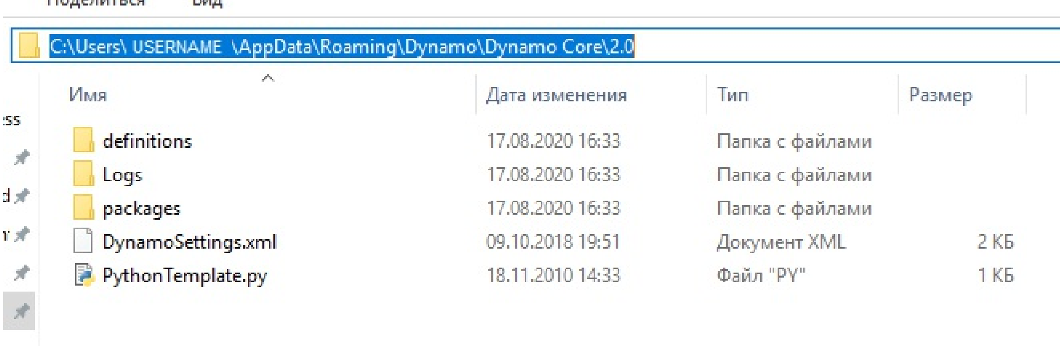
Шаблоны Python в Dynamo 2.0

Шаблоны Python

В Dynamo 2.0 появилась возможность задавать шаблон по умолчанию `(с расширением PY)` для использования при первом открытии окна Python. Эта функция, о которой нас давно просили пользователи, значительно ускоряет процесс работы со сценариями Python в Dynamo. Благодаря возможности использовать шаблон программисты получают мгновенный доступ к данным, которые требуется импортировать по умолчанию и на основе которых они могут разрабатывать пользовательские сценарии Python.

Этот шаблон размещается в подпапке APPDATA установочной папки Dynamo.

Путь к нему выглядит следующим образом: `( %appdata%\Dynamo\Dynamo Core\{версия}\ )`.



Настройка шаблона

Для использования этой функции необходимо добавить в файл `DynamoSettings.xml` следующую строку *(правки вносятся в Блокноте)*.

```
29  <CustomPackageFolders>
30    <string>C:\Users\ USERNAME \AppData\Roaming\Dynamo\Dynamo Core\2.0</string
31  </CustomPackageFolders>
32  <PackageDirectoriesToUninstall />
33  <PythonTemplateFilePath />
34  <BackupInterval>60000</BackupInterval>
35  <BackupFilesCount>1</BackupFilesCount>
36  <PackageDownloadTouAccepted>false</PackageDownloadTouAccepted>
```

Найдите строку `<PythonTemplateFilePath />` и замените ее следующим кодом:

```
<PythonTemplateFilePath>
<string>C:\Users\CURRENTUSER\AppData\Roaming\Dynamo\Dynamo Core\2.0\PythonTemplate.py</string>
</PythonTemplateFilePath>
```

Примечание. Вместо CURRENTUSER укажите имя пользователя.

Теперь нужно создать шаблон, включающий функции для встраивания. В данном случае требуется встроить функции импорта определенных данных из Revit, а также некоторые другие функции, обычно используемые при работе с Revit.

Создайте документ в Блокноте и вставьте в него следующий код:

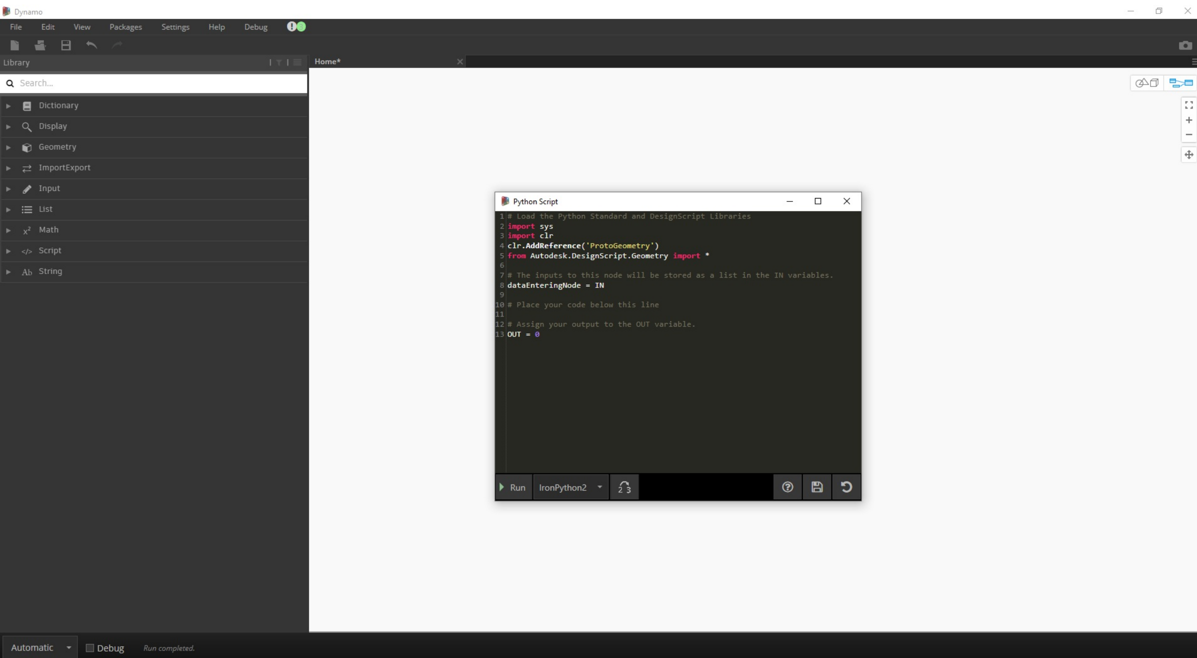
```
import clr

clr.AddReference('RevitAPI')
from Autodesk.Revit.DB import *
from Autodesk.Revit.DB.Structure import *

clr.AddReference('RevitAPIUI')
from Autodesk.Revit.UI import *

clr.AddReference('System')
from System.Collections.Generic import List

clr.AddReference('RevitNodes')
import Revit
clr.ImportExtensions(Revit.GeometryConversion)
clr.ImportExtensions(Revit.Elements)

clr.AddReference('RevitServices')
import RevitServices
from RevitServices.Persistence import DocumentManager
from RevitServices.Transactions import TransactionManager
```

```
doc = DocumentManager.Instance.CurrentDBDocument
uidoc=DocumentManager.Instance.CurrentUIApplication.ActiveUIDocument

#Preparing input from dynamo to revit
element = UnwrapElement(IN[0])

#Do some action in a Transaction
TransactionManager.Instance.EnsureInTransaction(doc)

TransactionManager.Instance.TransactionTaskDone()

OUT = element
```

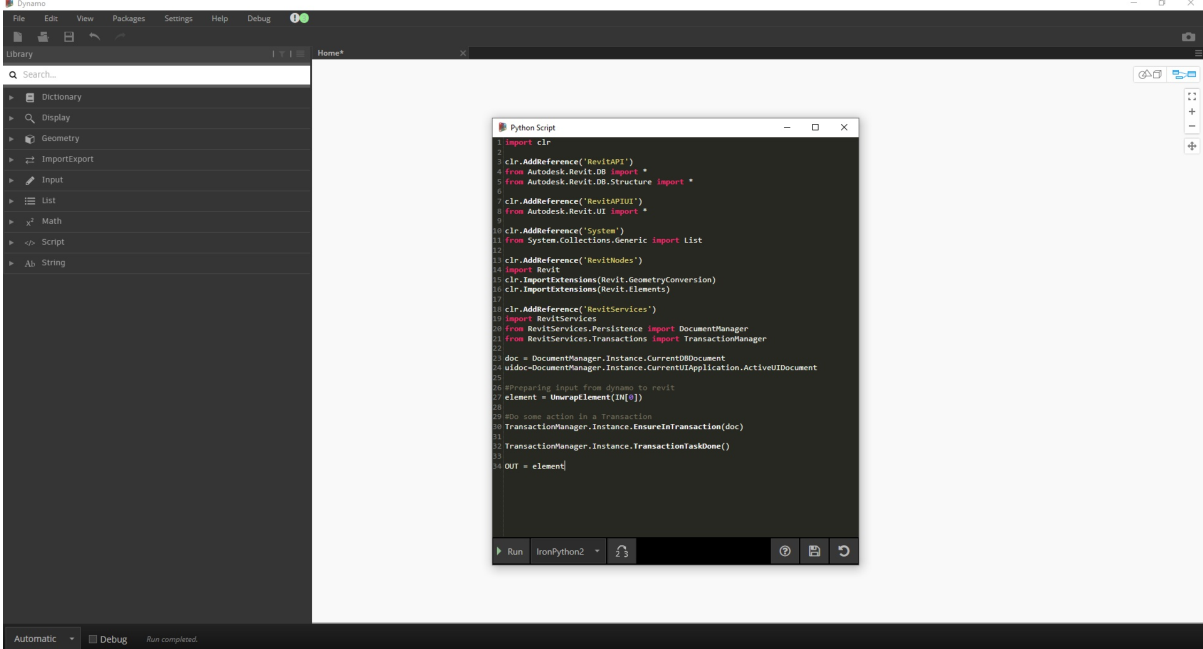
Когда все будет готово, сохраните этот файл под именем `PythonTemplate.py` в папке `APPDATA`.

Дальнейшее поведение сценария Python

После того как шаблон Python задан в приложении Dynamo, каждый раз при размещении узла Python будет выполняться поиск этого шаблона. Если шаблон не найден, отображается стандартное окно Python.



Если шаблон Python найден (как в случае с созданным шаблоном для работы с Revit), отображаются встроенные в него функции по умолчанию.

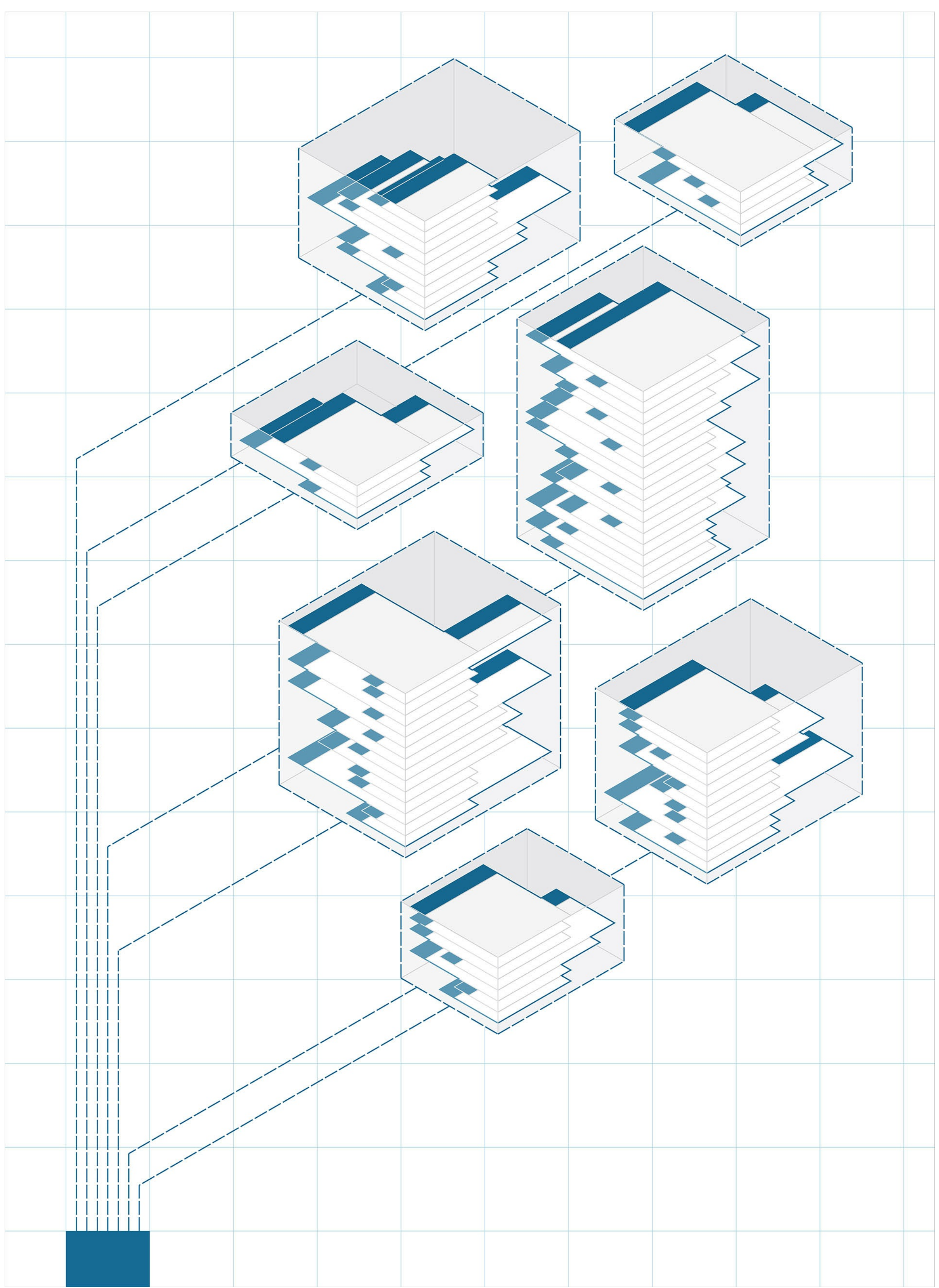


Подробные сведения об этом дополнении см. в следующем материале (автор: Раду Гидей [Radu Gidei]): https://github.com/DynamoDS/Dynamo/pull/8122.

Пакеты

Пакеты

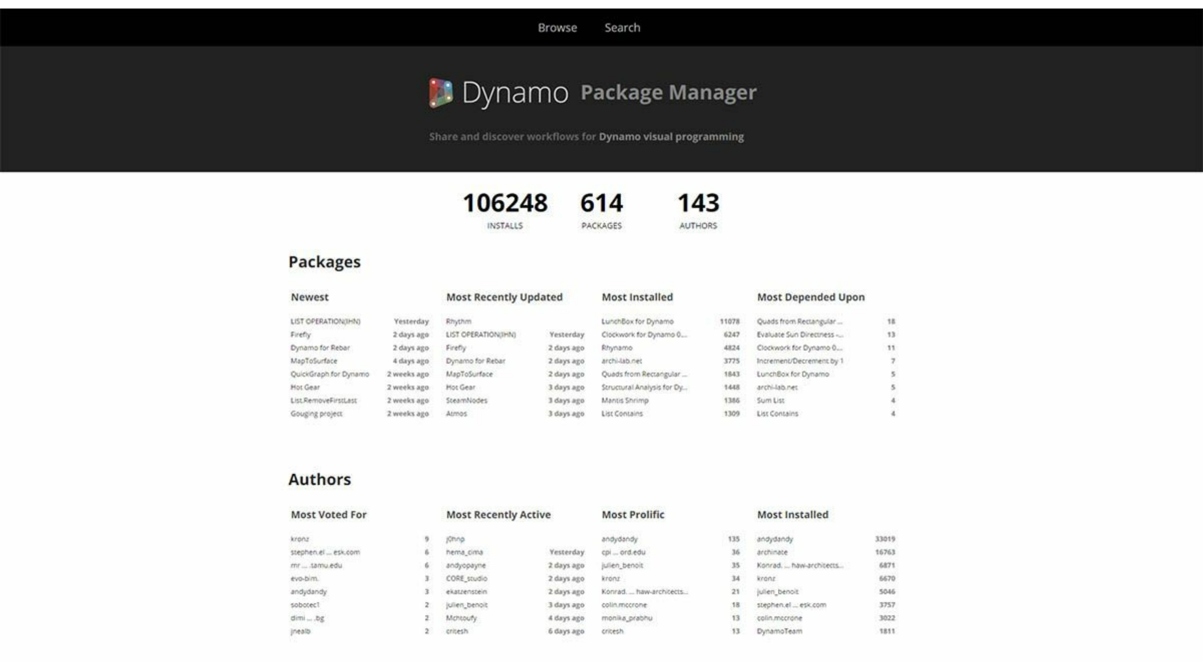
После создания пользовательских узлов необходимо упорядочить и опубликовать их в формате пакетов. Пакеты упрощают хранение узлов, а также позволяют с легкостью делиться ими с другими пользователями Dynamo.



Пакеты: введение

Пакеты

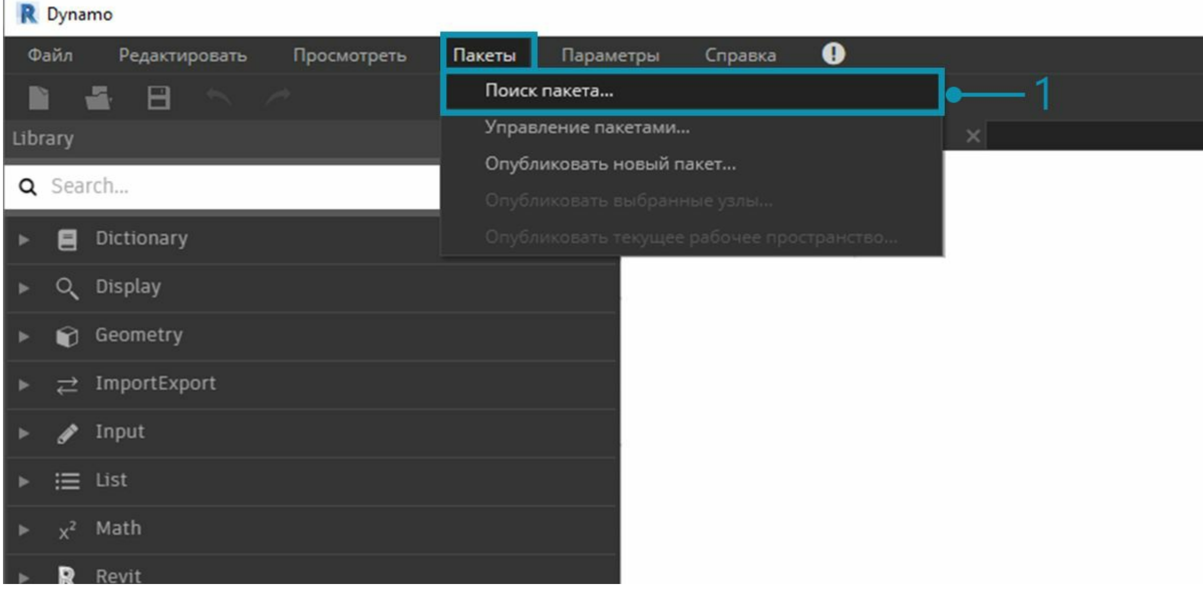
В целом пакет представляет собой набор пользовательских узлов. Dynamo Package Manager — это портал для сообщества пользователей, где можно скачать любые пакеты, которые были опубликованы в интернете. Эти инструментарии разрабатываются сторонними поставщиками и предназначены для расширения базовых функций Dynamo, доступных каждому пользователю по первому требованию.



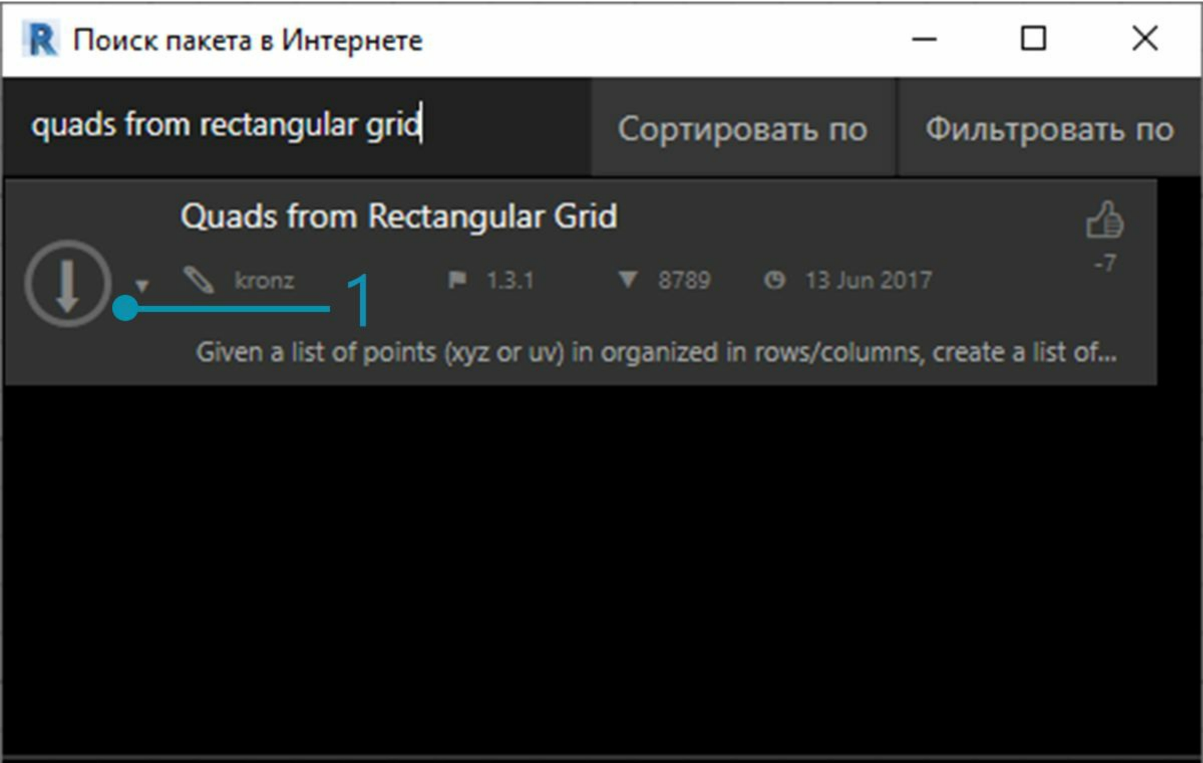
Проекты с открытым исходным кодом, такие как Dynamo, активно развиваются благодаря подобному участию сообщества. Благодаря узкоспециализированным сторонним разработчикам Dynamo может использоваться в самых различных отраслях. По этой причине команда Dynamo сконцентрировала свои усилия на оптимизации разработки и публикации пакетов (подробнее эта тема будет обсуждаться в следующих разделах).

Установка пакетов

Самый простой способ установки пакета — воспользоваться панелью инструментов «Пакеты» в интерфейсе Dynamo. Перейдем на эту панель и установим один из пакетов. В этом небольшом примере будет установлен популярный пакет для создания прямоугольных панелей на сетке.

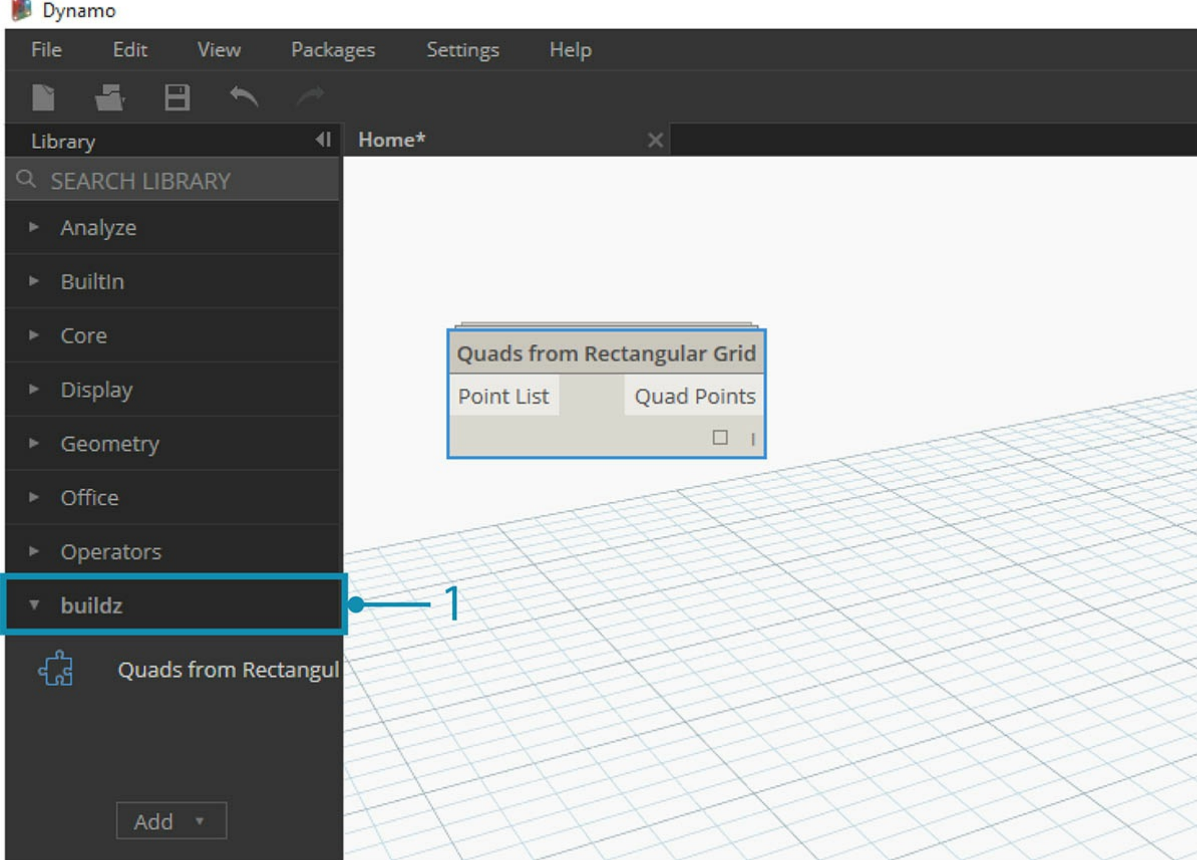


1. В Dynamo выберите *«Пакеты»* > *«Поиск пакета...»*.

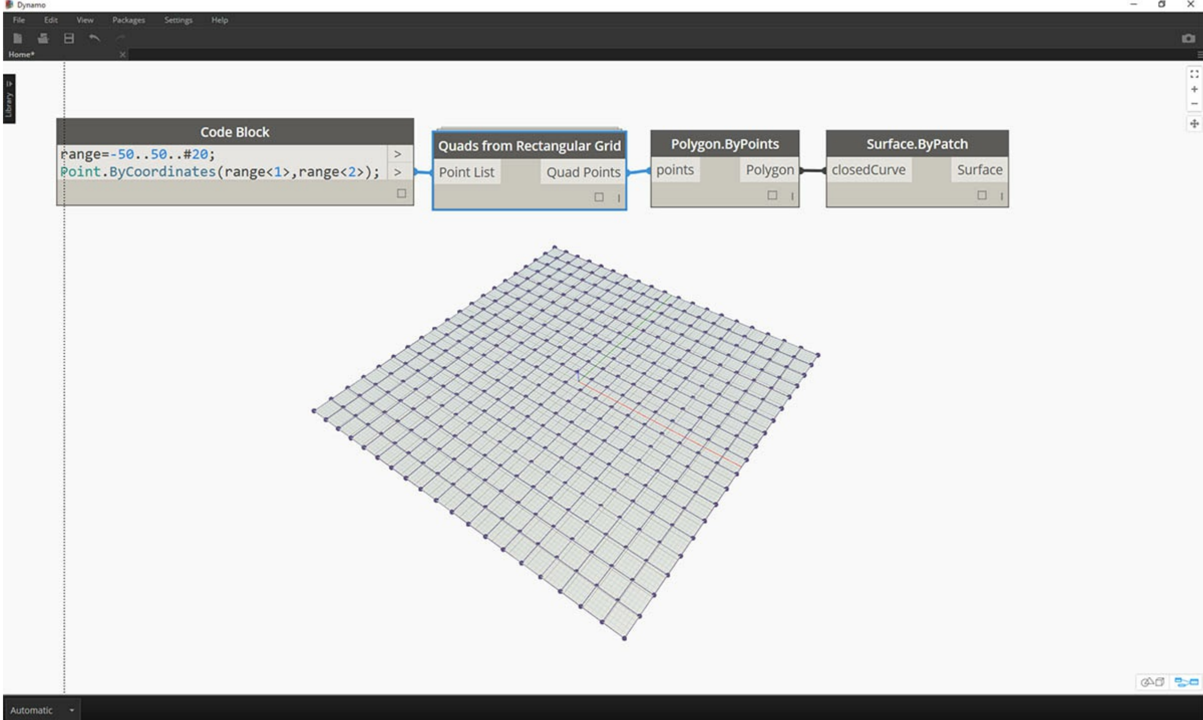


В строке поиска введите quads from rectangular grid. Через некоторое время отобразятся все пакеты, соответствующие поисковому запросу. Выберем первый пакет с соответствующим именем.

1. Щелкните стрелку скачивания слева от имени пакета, чтобы установить пакет. Готово!



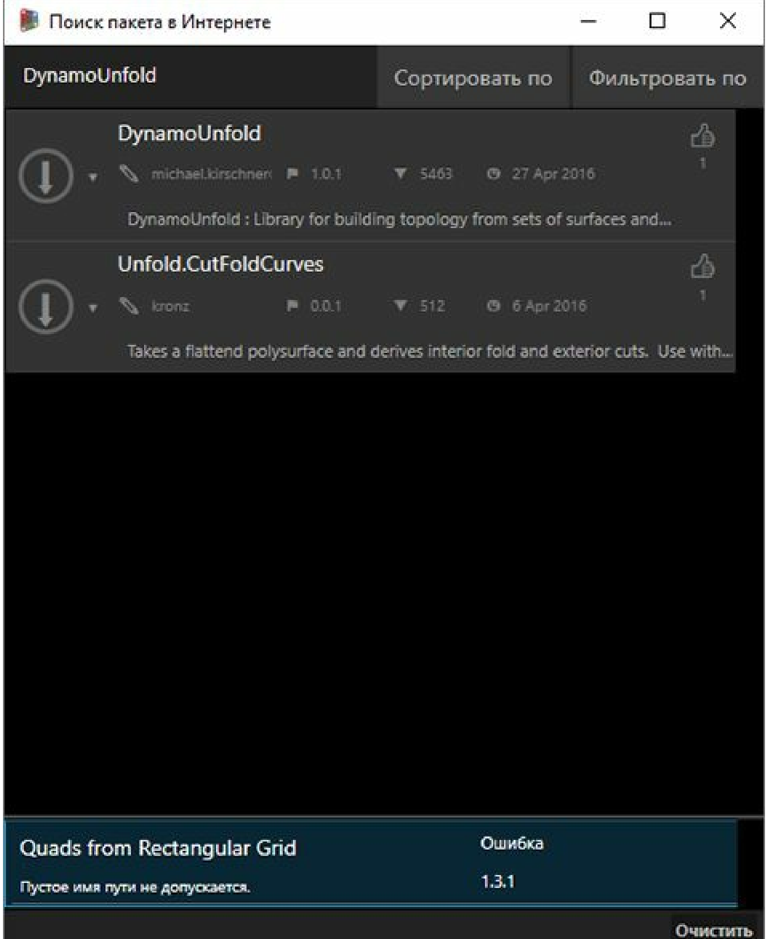
1. Обратите внимание, что в библиотеке Dynamo появилась группа с именем *buildz*. Это имя разработчика пакета, а в группе содержится пользовательский узел. Его можно сразу использовать.



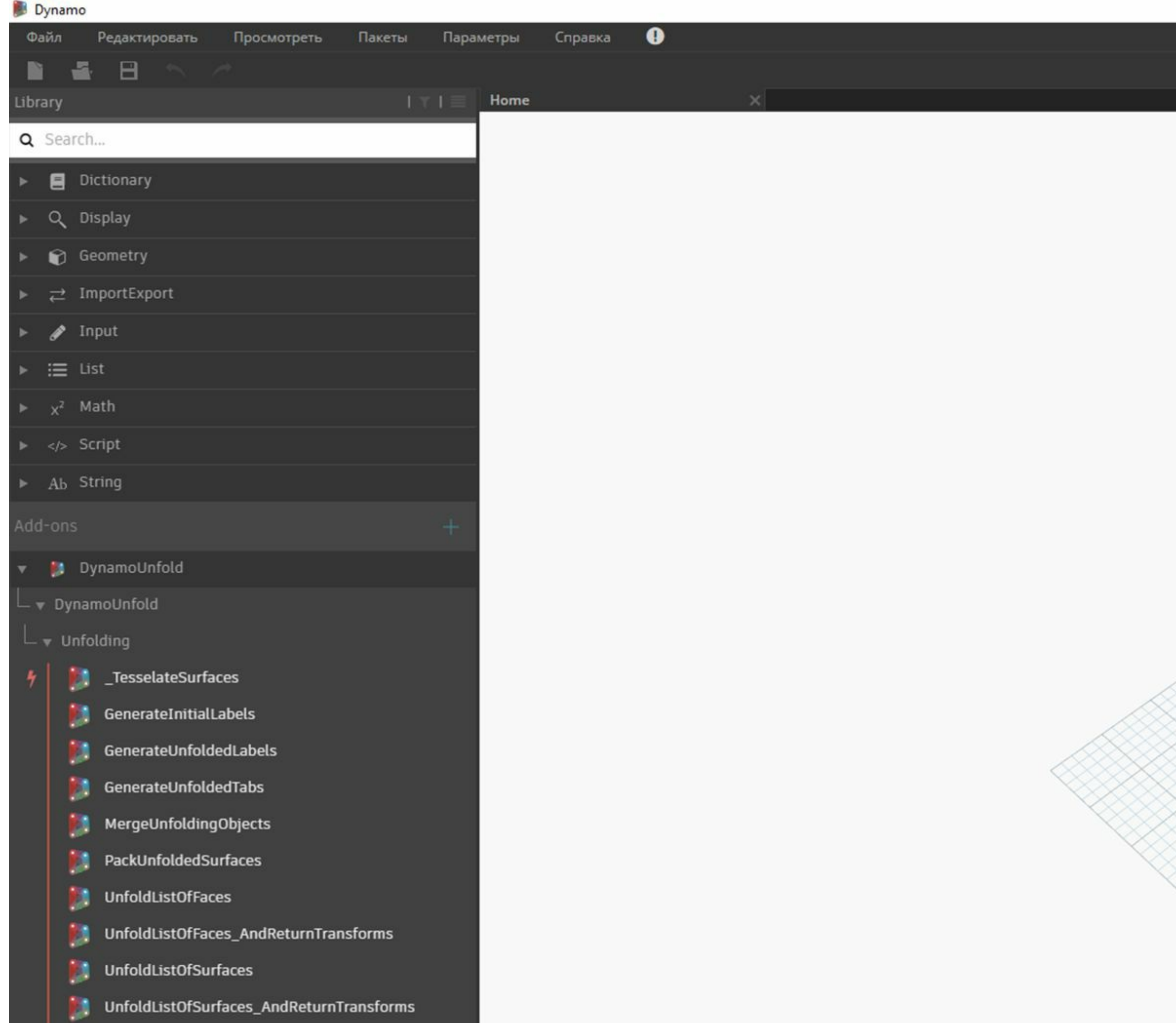
С помощью блока кода быстро определите прямоугольную сетку и создайте список прямоугольных панелей.

Папки пакетов

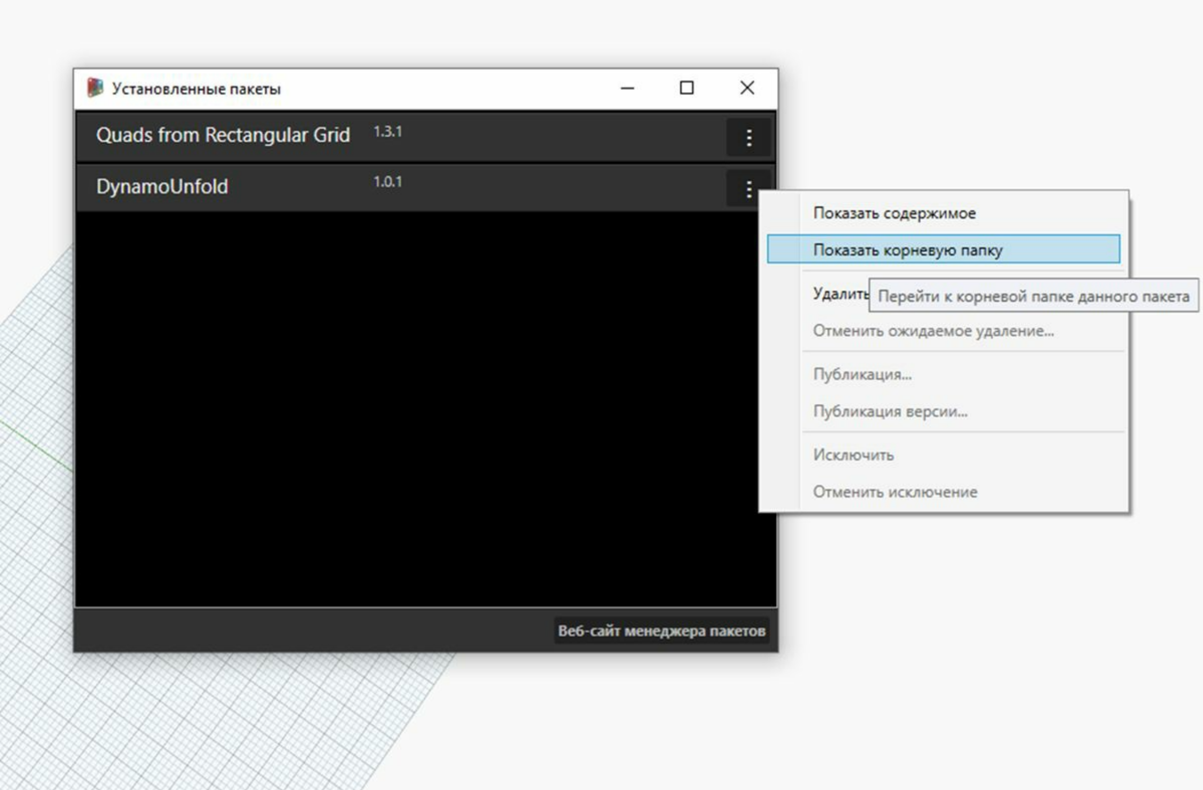
В приведенном выше примере описан пакет с одним пользовательским узлом. Та же самая процедура используется для скачивания пакетов с несколькими пользовательскими узлами и вспомогательными файлами данных. Продемонстрируем это на примере более крупного пакета: Dynamo Unfold.



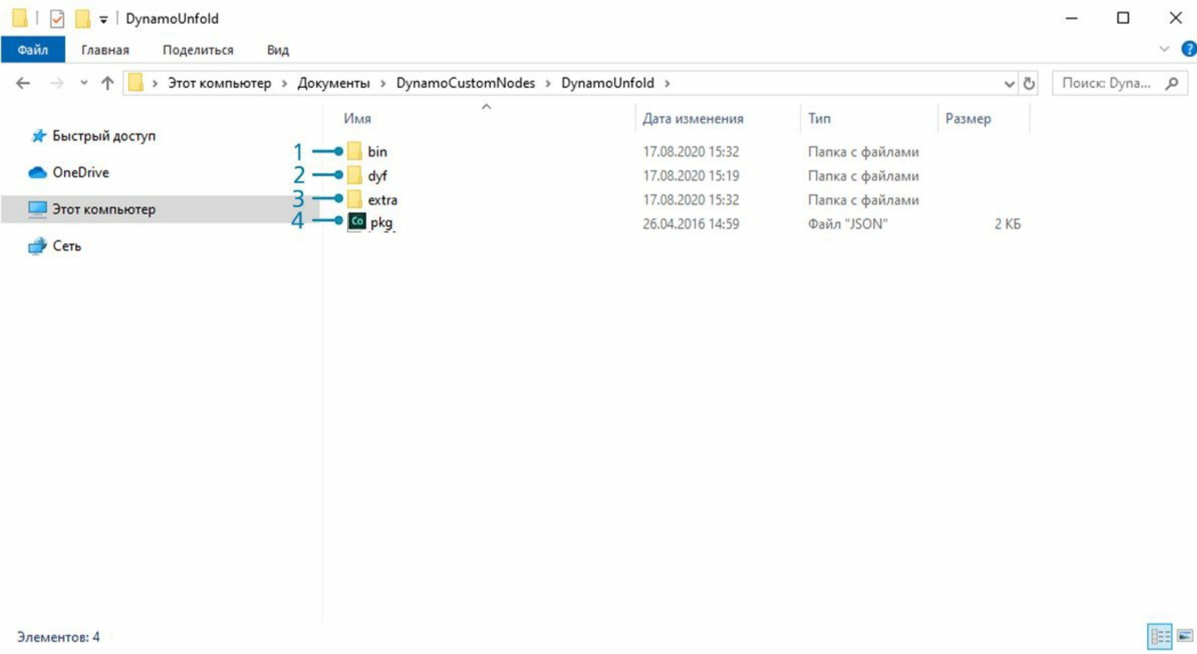
Как и в приведенном выше примере, начните с выбора команды «*Пакеты*» > «*Поиск пакета*...». На этот раз в строке поиска введите *DynamoUnfold* без пробела с учетом регистра. После отображения пакета скачайте его, щелкнув стрелку слева от имени. Пакет Dynamo Unfold будет установлен в библиотеке Dynamo.



В библиотеке Dynamo появилась группа *DynamoUnfold* с несколькими категориями и пользовательскими узлами.

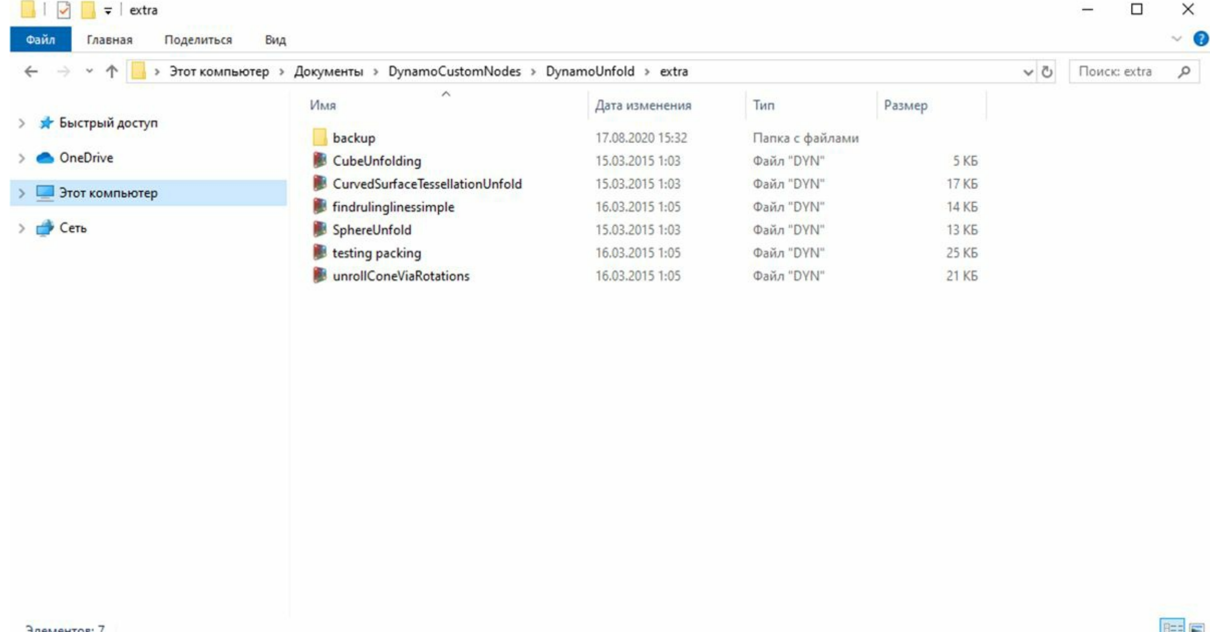


Рассмотрим структуру файлов и папок пакета. В Dynamo выберите *«Пакеты»* > *«Управление пакетами...»*. Откроется окно, в котором отображаются две установленные библиотеки. Нажмите кнопку справа от элемента *DynamoUnfold* и выберите *Показать корневую папку*.

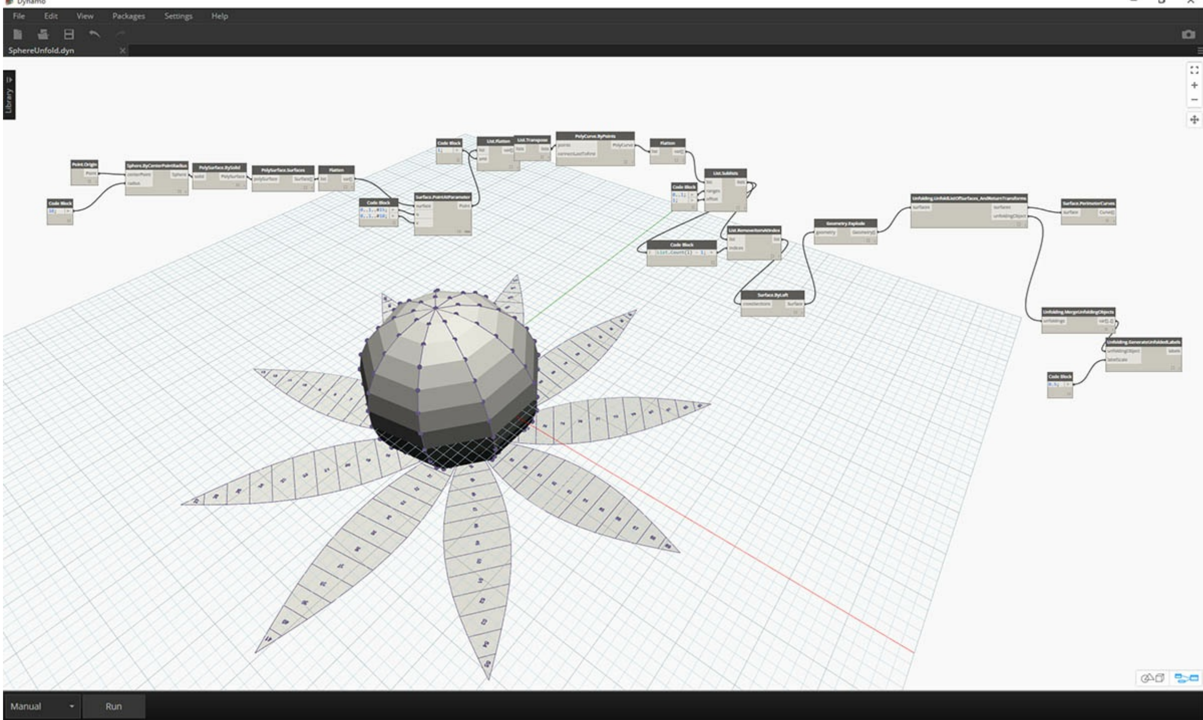


Откроется корневая папка пакета. Обратите внимание, что в ней содержится три папки и файл.

1. В папке *bin* находятся файлы DLL. Этот пакет Dynamo был разработан с помощью функции Zero-Touch, поэтому пользовательские узлы хранятся в этой папке.
2. В папке *dyf* находятся пользовательские узлы. Так как данный пакет был разработан без пользовательских узлов Dynamo, папка пуста.
3. В папке extra хранятся дополнительные файлы, включая файлы примеров.
4. Файл pkg — это базовый текстовый файл, определяющий параметры пакета. Пока мы не будем его рассматривать.



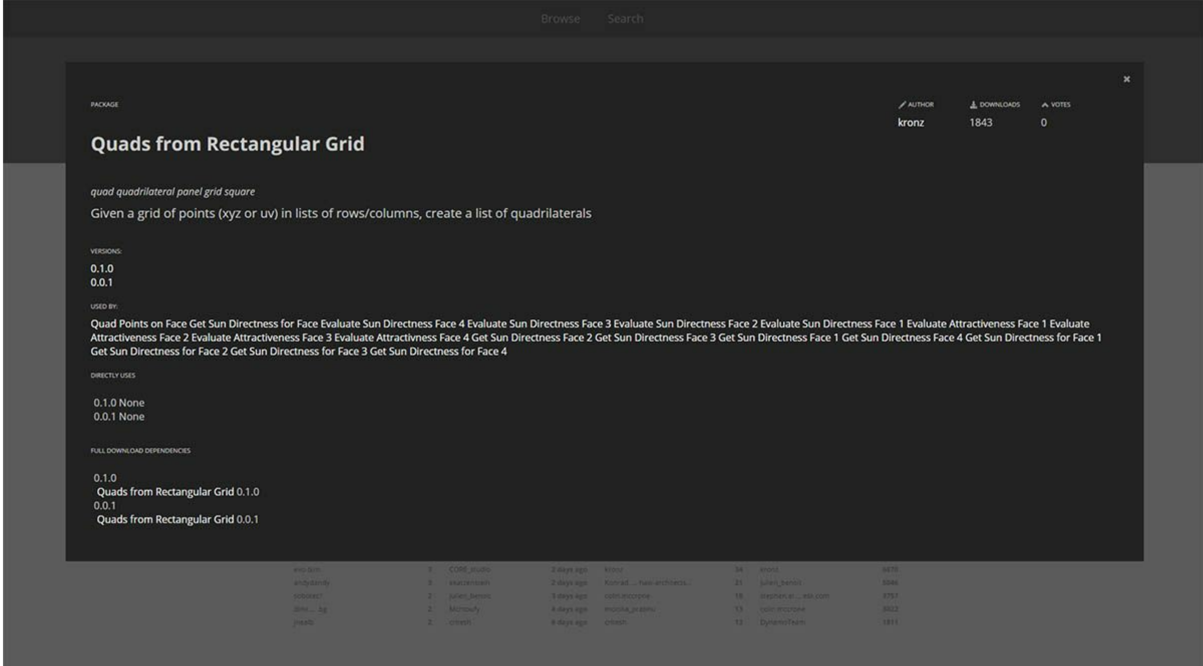
Если открыть папку *extra*, в ней можно увидеть несколько файлов примеров, которые были скачаны при установке. Если пакет сопровождается файлами примеров (что бывает не всегда), их следует искать в этой папке. Откроем файл *SphereUnfold*.



Нажав после этого кнопку *Запуск* в решателе, получим развернутую сферу. Файлы примеров, подобные этим, используются для обучения работе с новыми пакетами Dynamo.

Dynamo Package Manager

Еще один способ найти пакеты Dynamo — использовать онлайн-ресурс [Dynamo Package Manager](). Искать пакеты в этом хранилище очень удобно, так как они отсортированы по количеству скачиваний и популярности. Кроме того, здесь можно легко получить информацию о последних обновлениях пакетов, что очень важно, так как некоторые пакеты Dynamo имеют разные версии и зависимости от сборок Dynamo.



Если в Dynamo Package Manager выбрать пакет *Quads from Rectangular Grid*, откроется его описание, данные о версиях, имя разработчика, а также сведения о возможных зависимостях.

Кроме того, из Dynamo Package Manager можно скачивать файлы пакетов Dynamo, но делать это непосредственно из Dynamo проще.

Местоположение файлов пакетов на локальном компьютере

Если необходимо скачать файлы с портала Dynamo Package Manager или узнать, где хранятся все файлы пакетов, щелкните *«Параметры» > «Управление путями к узлу и пакету..»*. Щелкнув значок многоточия рядом с папкой, можно скопировать корневую папку и ознакомиться с пакетом в окне проводника. По умолчанию пакеты устанавливаются в следующей папке: *C:/Users/[имя пользователя]/AppData/Roaming/Dynamo/[версия Dynamo]*.

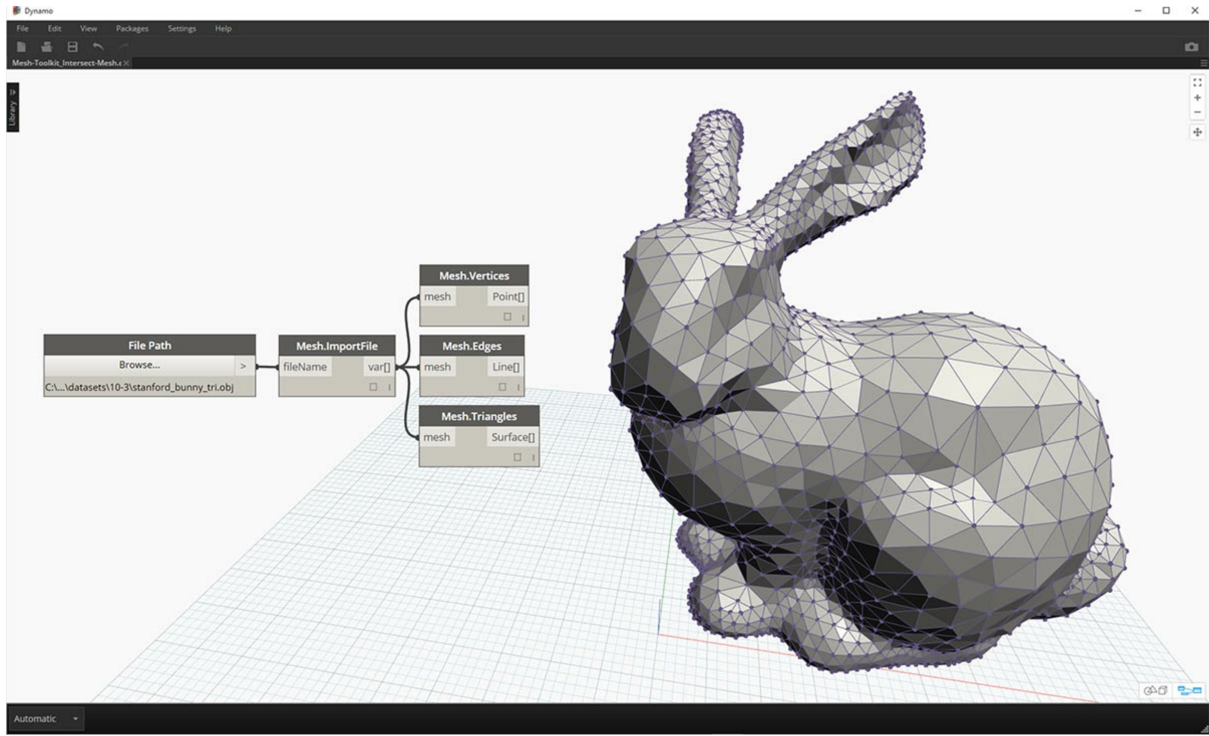
Дальнейшая работа с пакетами

Сообщество Dynamo постоянно развивается. Если периодически посещать портал Dynamo Package Manager, можно обнаружить там новые интересные разработки. В следующих разделах пакеты Dynamo будут рассматриваться более подробно и не только с точки зрения конечного пользователя, но и в контексте их самостоятельной разработки.

Практикум по работе с пакетом: Mesh Toolkit

Практикум по работе с пакетом: Mesh Toolkit

Dynamo Mesh Toolkit содержит инструменты для импорта сетей из внешних файлов других форматов, создания сетей из геометрических объектов Dynamo и построения сетей вручную по вершинам и индексам. В библиотеке также содержатся инструменты для редактирования и восстановления сетей, а также для извлечения горизонтальных срезов, используемых в ходе изготовления изделий.

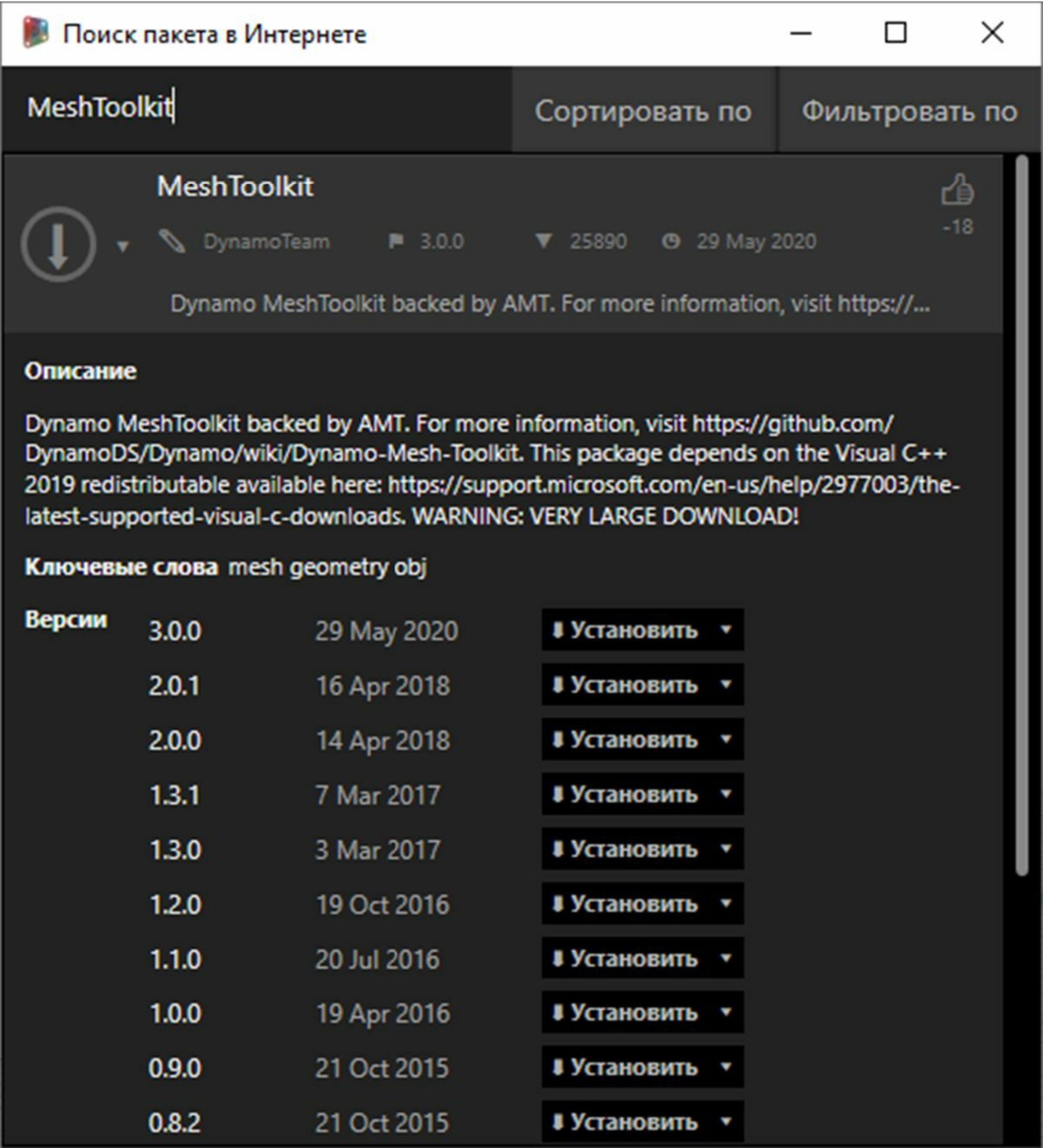


Пакет Dynamo Mesh Toolkit создан в рамках непрерывной работы специалистов Autodesk, направленной на исследование сетей, и в ближайшие годы функциональные возможности пакета будут постоянно улучшаться и пополняться. Разработчики Dynamo с нетерпением ждут ваших отзывов и предложений по новым функциям, а также сообщений об обнаруженных ошибках.

Сети и тела

В упражнении ниже демонстрируются некоторые базовые операции с сетями, выполняемые с помощью Mesh Toolkit. В этом упражнении сеть рассекается на несколько частей при помощи плоскостей, что при использовании тел потребовало бы больших вычислительных мощностей. Сеть, в отличие от тела, имеет заданное «разрешение». Кроме того, она определяется не математически, а топологически. Благодаря этому определение сети можно адаптировать в соответствии с поставленной задачей. Дополнительные сведения о взаимоотношениях сетей и тел см. в разделе Геометрия для машинного проектирования данного руководства. Подробный обзор пакета Mesh Toolkit см. в справке Wiki по Dynamo. Выполните следующее упражнение, чтобы узнать, как использовать этот пакет на практике.

Установка Mesh Toolkit

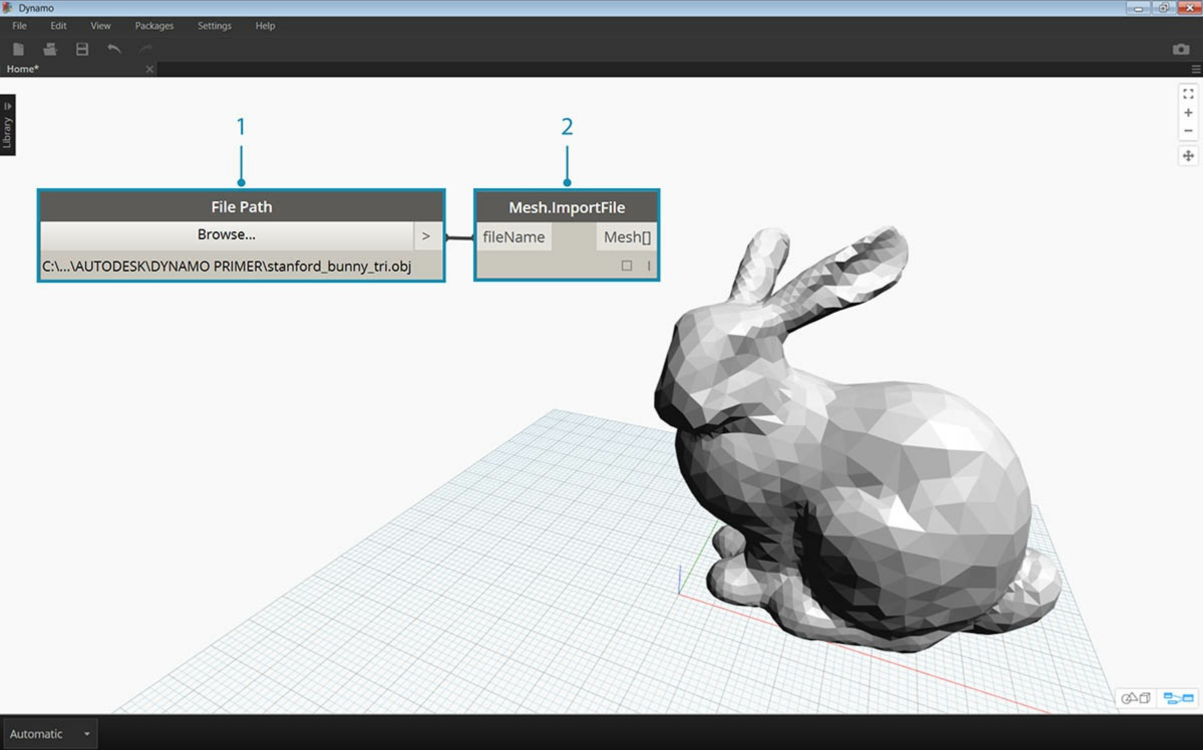


В строке меню Dynamo выберите *Пакеты* > *Поиск пакета...* В поле поиска введите *MeshToolkit* без пробелов и с соблюдением регистра. Щелкните стрелку скачивания рядом с пакетом, соответствующим установленной версии Dynamo. Проще простого.

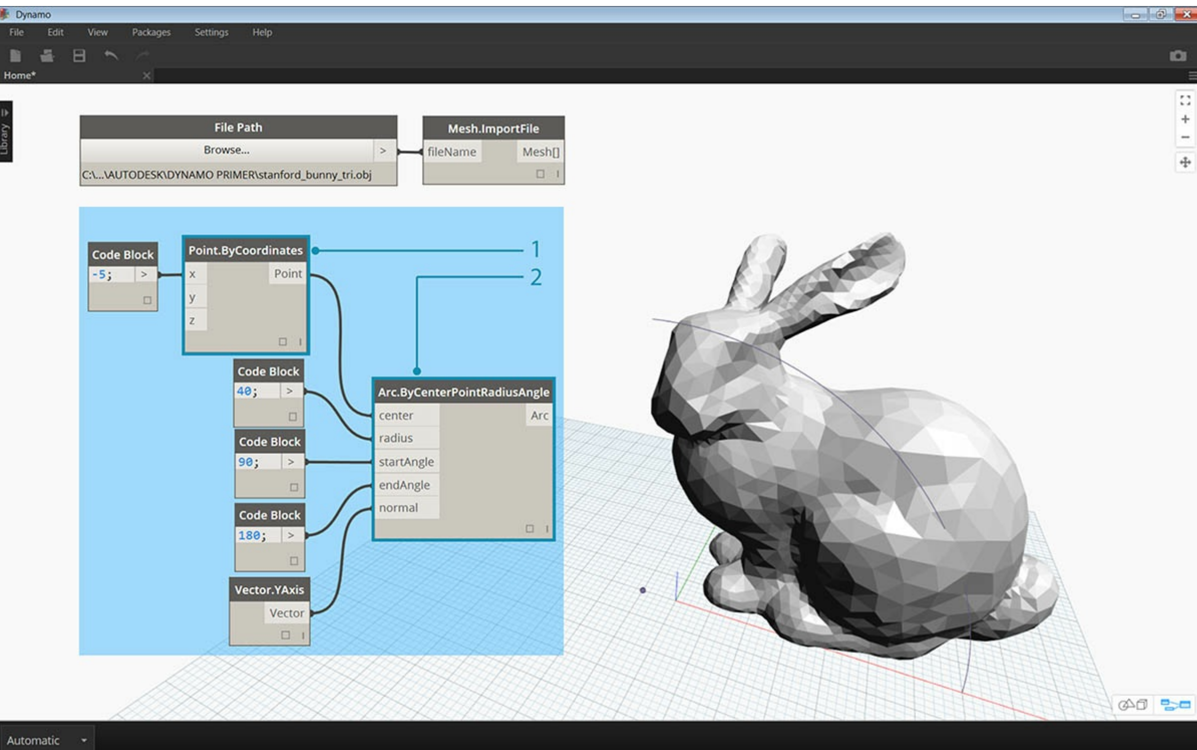
Упражнение

Скачайте и распакуйте файлы примеров для этого упражнения (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. [MeshToolkit.zip]

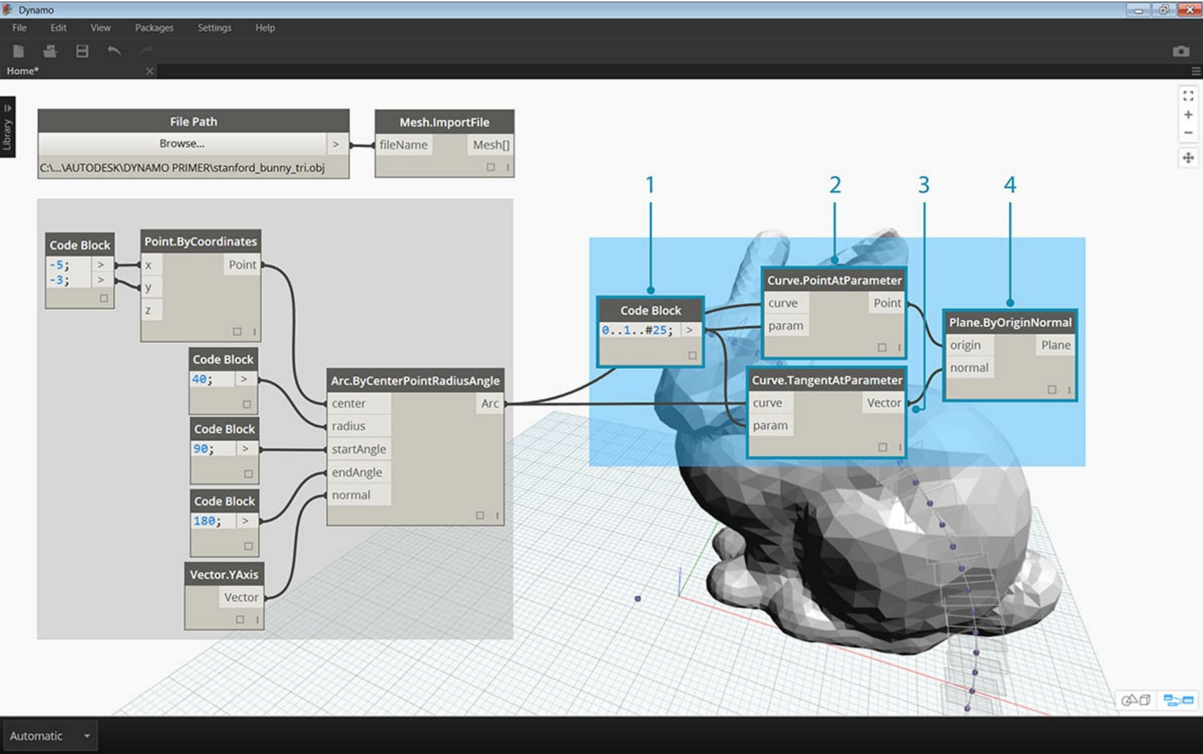
Для начала откройте в Dynamo файл *Mesh-Toolkit_Intersect-Mesh.dyn*. В этом примере рассматривается работа с узлом Intersect в составе MeshToolkit. Вам потребуется импортировать сеть, а затем рассечь ее с использованием нескольких входных плоскостей для получения срезов. Это первый этап подготовки модели изделия к изготовлению с помощью лазерной или водоструйной резки либо фрезерного станка с ЧПУ.



1. **File Path:** найдите файл сети, который требуется импортировать (*stanford_bunny_tri.obj*). Поддерживаются файлы MIX и OBJ.
2. **Mesh.ImportFile:** соедините этот узел с узлом File Path, чтобы импортировать сеть.

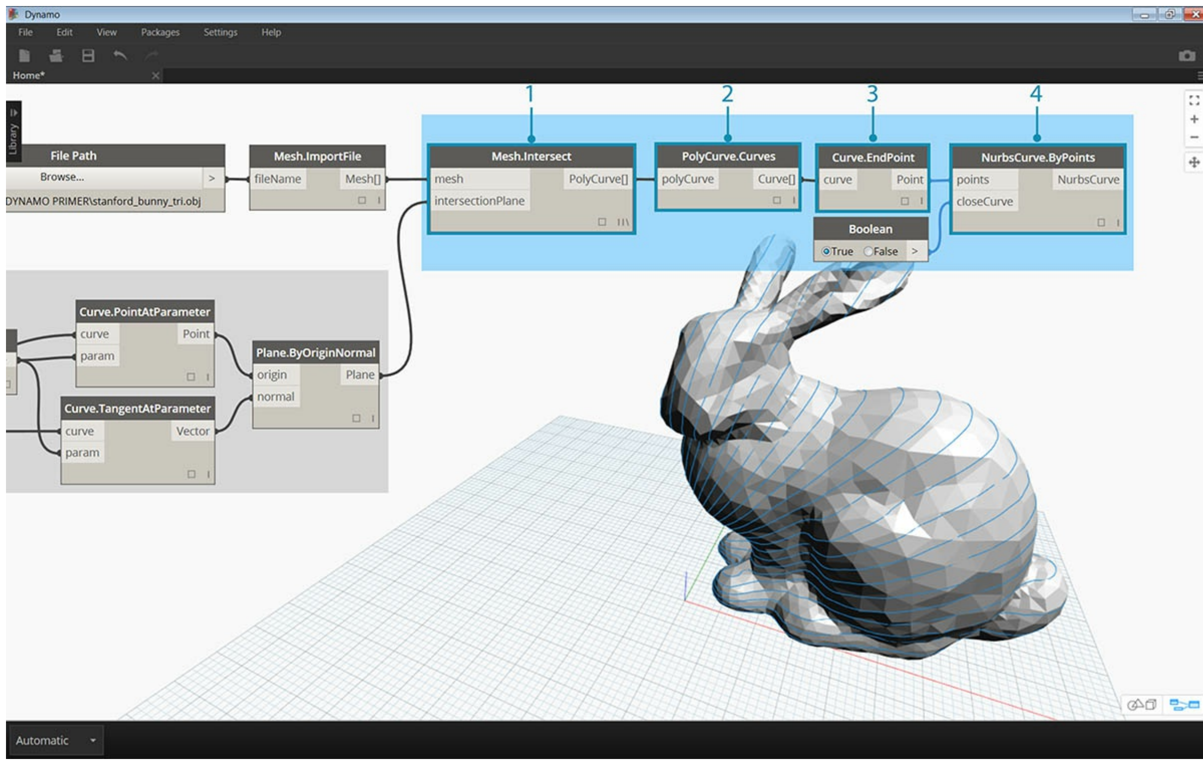


1. **Point.ByCoordinates:** создайте точку, которая станет центром дуги.
2. **Arc.ByCenterPointRadiusAngle**: создайте дугу на основе заданной точки. Эта кривая будет использована для размещения набора плоскостей.

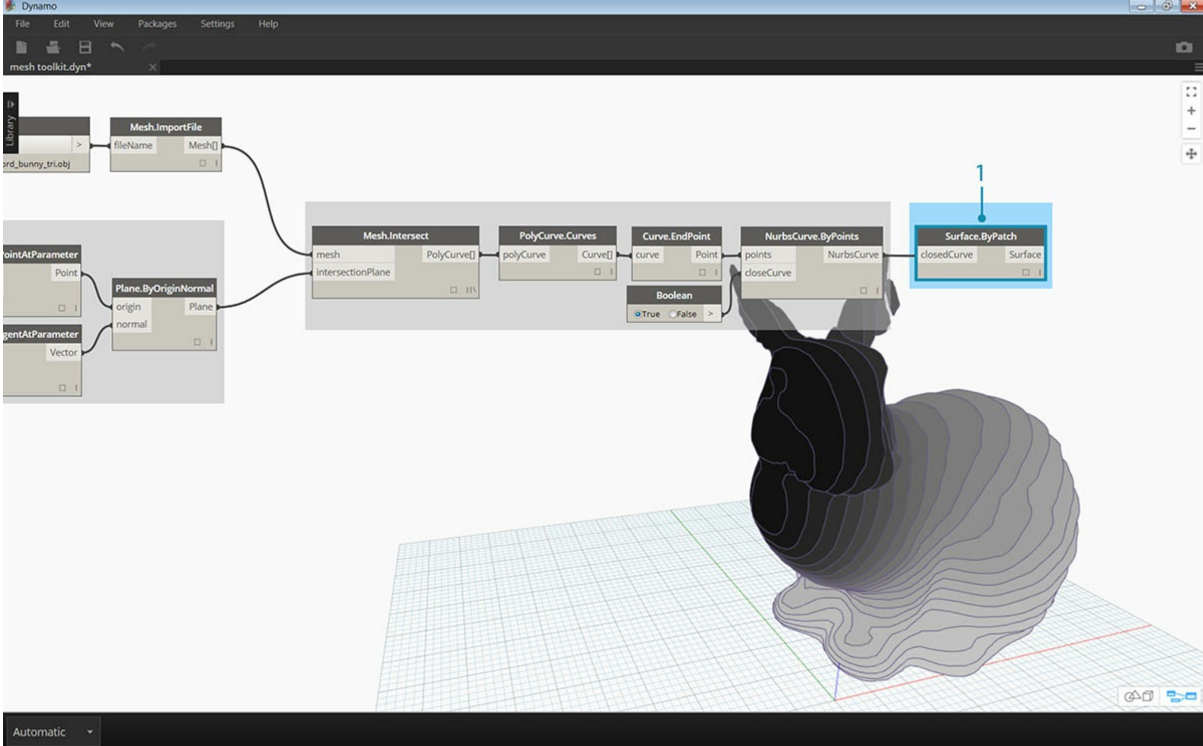


1. Code Block: задайте диапазон чисел от нуля до единицы.
2. **Curve.PointAtParameter:** соедините порт вывода Arc с портом ввода *curve*, а порт вывода Code Block — с портом ввода *param*, чтобы получить набор точек вдоль кривой.
3. **Curve.TangentAtParameter:** соедините порты этого узла аналогично портам предыдущего.
4. **Plane.ByOriginNormal:** соедините порт вывода Point с портом ввода *origin*, а порт вывода Vector — с портом ввода *normal*, чтобы создать набор плоскостей на основе полученных точек.

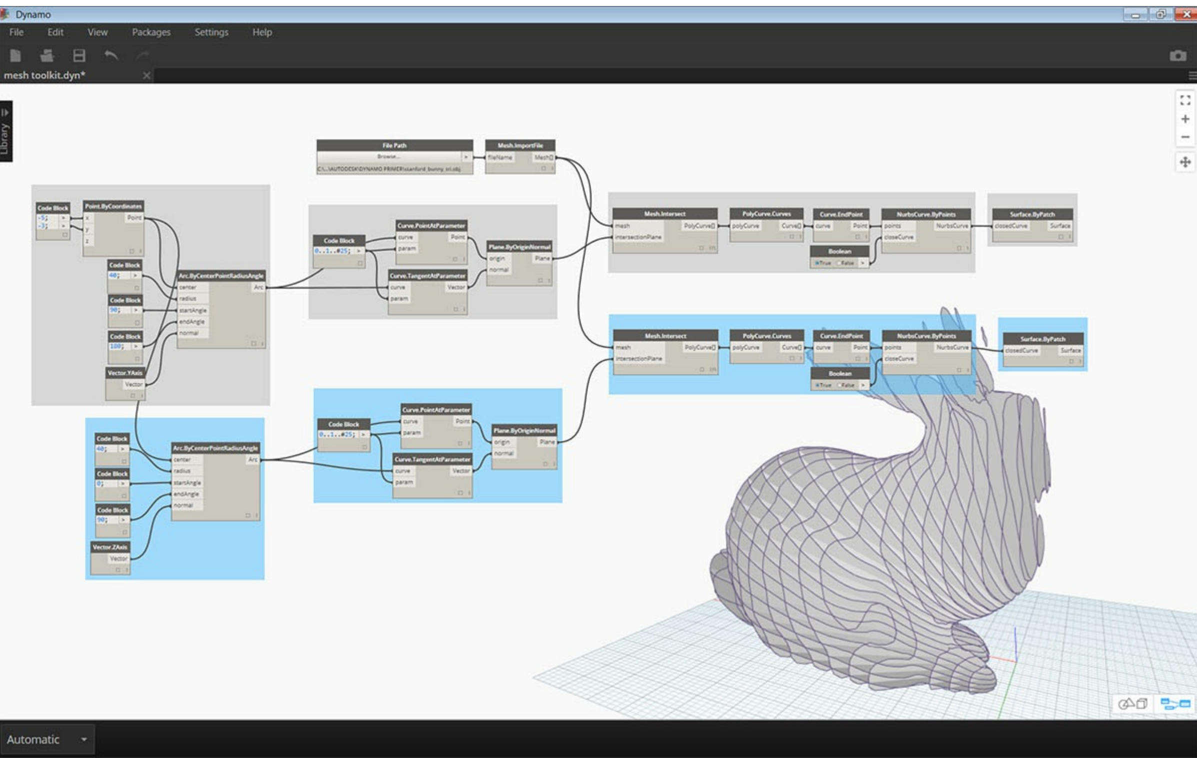
В окне должен появиться набор плоскостей, размещенных вдоль дуги. Рассеките сеть с помощью этих плоскостей.



1. **Mesh.Intersect:** плоскости рассекают импортированную сеть, в результате чего создается набор контуров, состоящих из поликривых.
2. **PolyCurve.Curves:** поликривые разбиваются на составляющие их кривые.
3. **Curve.EndPoint:** извлеките значения конечных точек для каждой кривой.
4. **NurbsCurve.ByPoints:** постройте NURBS-кривую на основе полученных точек. Добавьте узел Boolean и установите для него значение *True*, чтобы замкнуть кривые.



1. **Surface.ByPatch:** создайте участки поверхности для каждого контура, чтобы сформировать срезы сети.



Добавьте второй набор срезов для получения «вафельного» эффекта.

Как вы могли заметить, операции пересечения при работе с сетями выполняются гораздо быстрее, чем при работе с аналогичным телом. Использование сетей позволяет ускорить многие рабочие процессы, подобные представленному в этом упражнении.

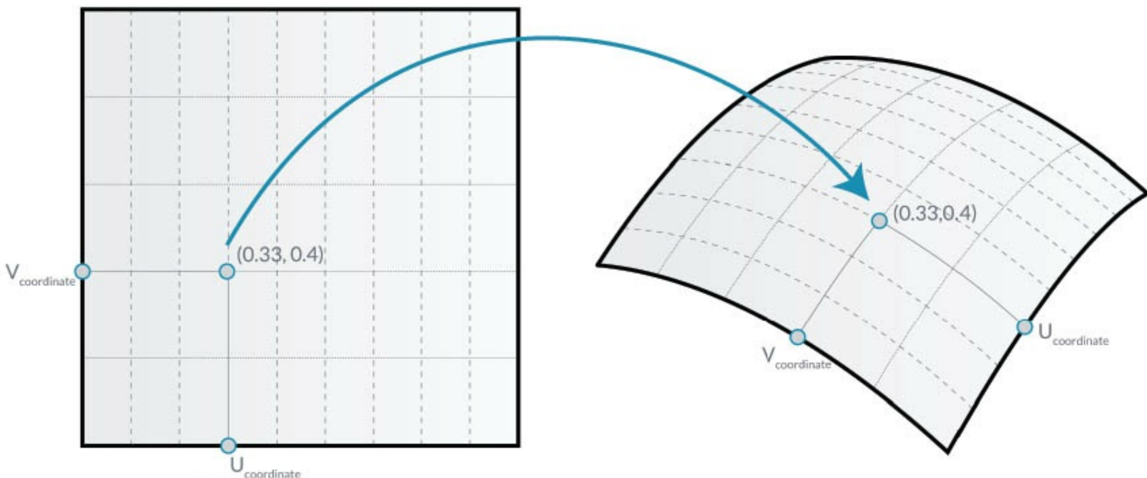
Разработка пакетов

Разработка пакетов

Dynamo поддерживает различные способы создания пакетов как с целью личного использования, так и для обмена с участниками сообщества Dynamo. Ниже рассматривается структура пакетов на примере разбора содержимого существующего пакета. Данный пример основан на упражнениях предыдущей главы, где был создан набор пользовательских узлов для сопоставления геометрии одной поверхности Dynamo с другой с помощью UV-координат.

MapToSurface

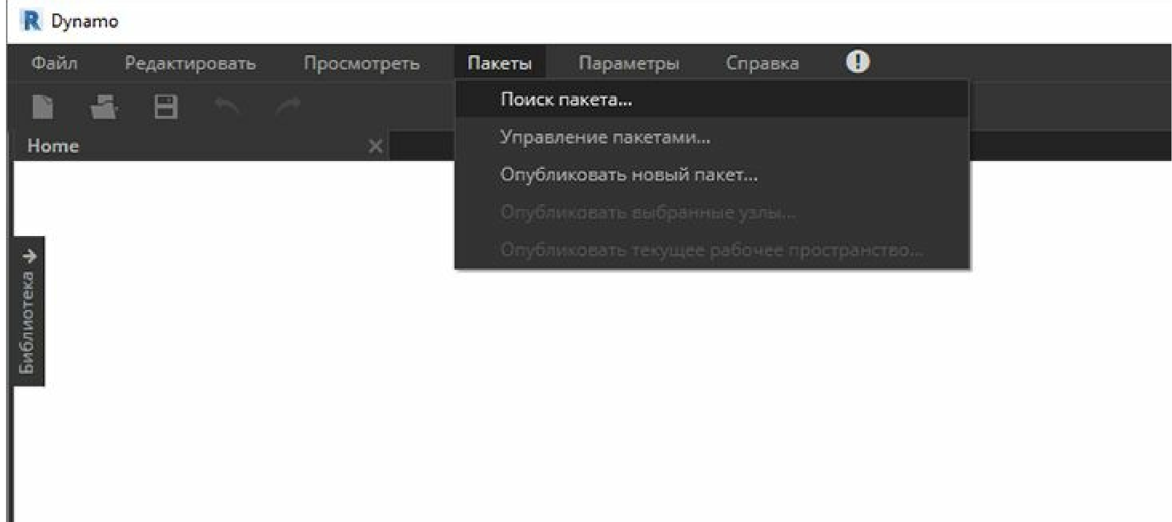
Для иллюстрации воспользуемся примером пакета для UV-наложения точек с одной поверхности на другую. Основные возможности инструмента уже рассмотрены в разделе Создание пользовательских узлов этого учебника. Приведенные ниже файлы показывают, как можно применить принцип UV-наложения к разработке набора инструментов для библиотеки с возможностью публикации.



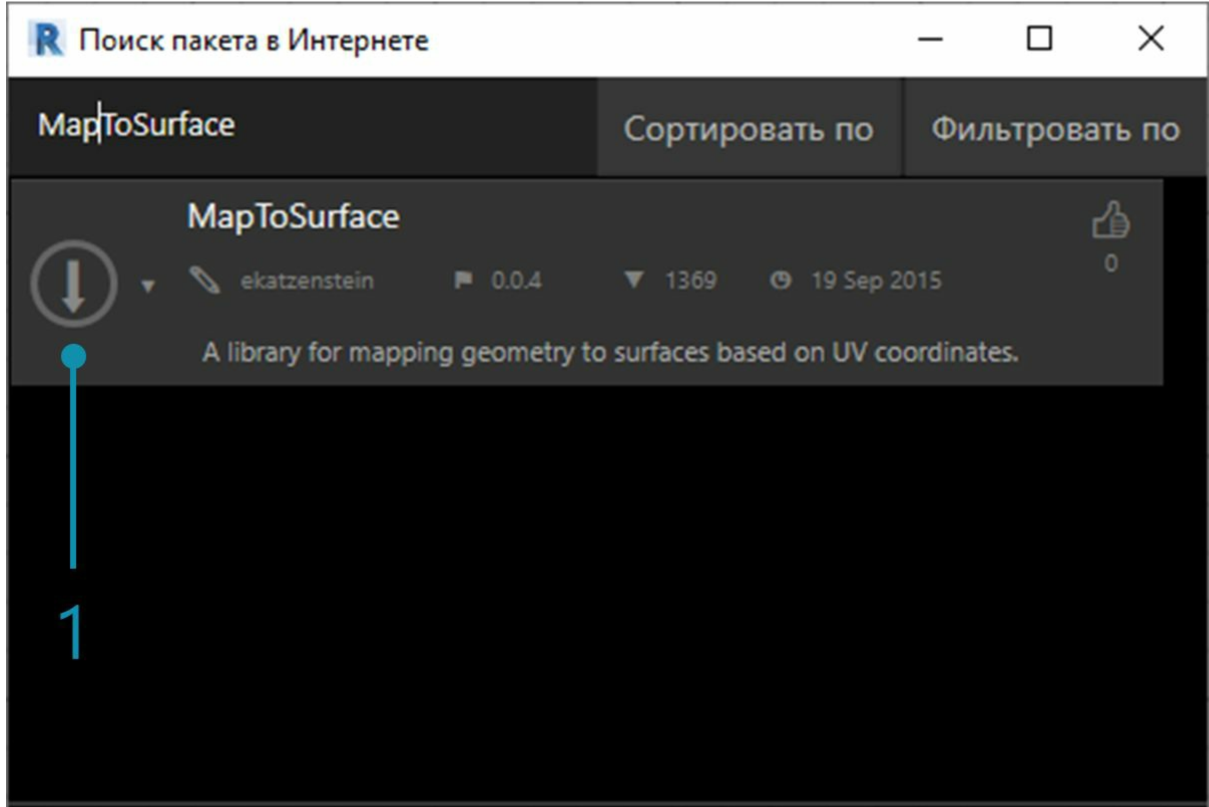
На данном изображении точка одной поверхности сопоставляется с точкой другой поверхности с помощью UV-координат. Эта же концепция лежит в основе работы данного пакета, однако он рассчитан на более сложную геометрию.

Установка пакета

В предыдущей главе рассматривались способы создания панелей поверхности в Dynamo на основе кривых, заданных в плоскости XY. В этой главе те же принципы рассматриваются более подробно, охватывая другие размеры и геометрические объекты. Чтобы показать, каким образом осуществлялась разработка пакета, он будет установлен в исходном состоянии. В следующем разделе будет рассматриваться публикация этого пакета.

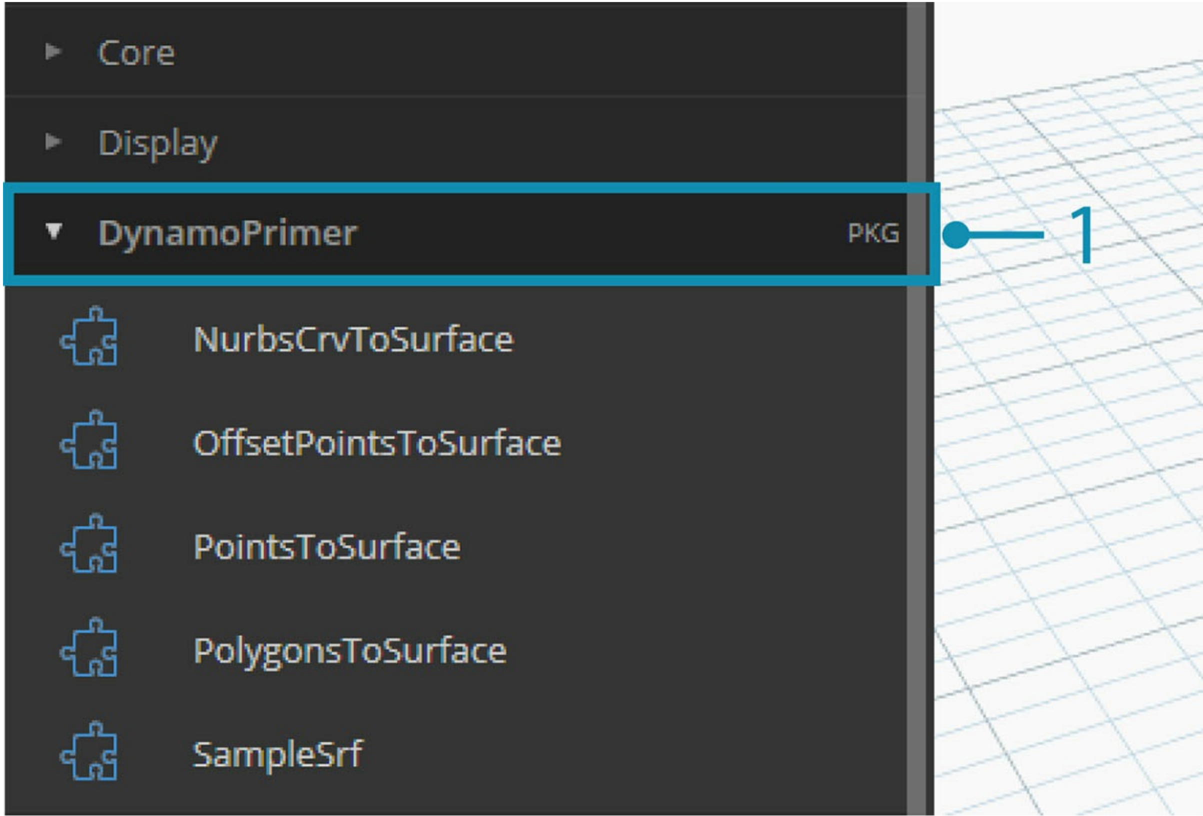


Это самая легкая часть. В Dynamo перейдите в меню *«Пакеты»* > *«Поиск пакета...»*.



В строке поиска введите *MapToSurface* (без пробелов).

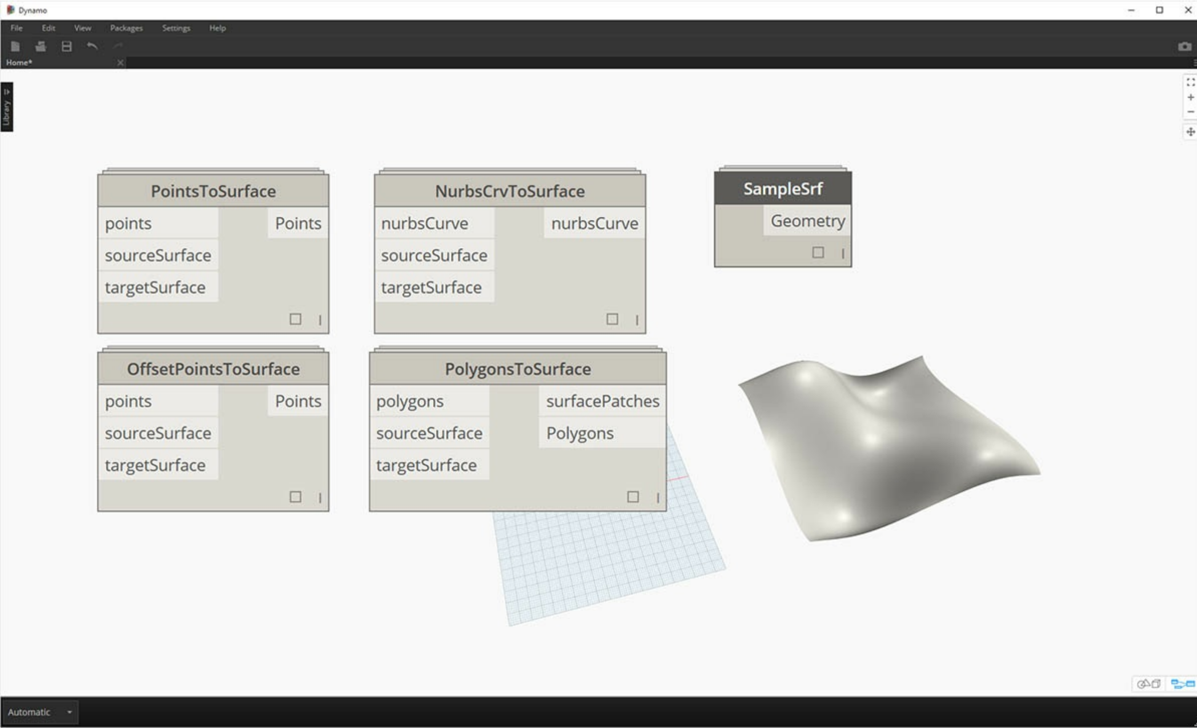
1. Когда пакет будет найден, щелкните большую стрелку скачивания слева от его имени. Пакет будет установлен в Dynamo.



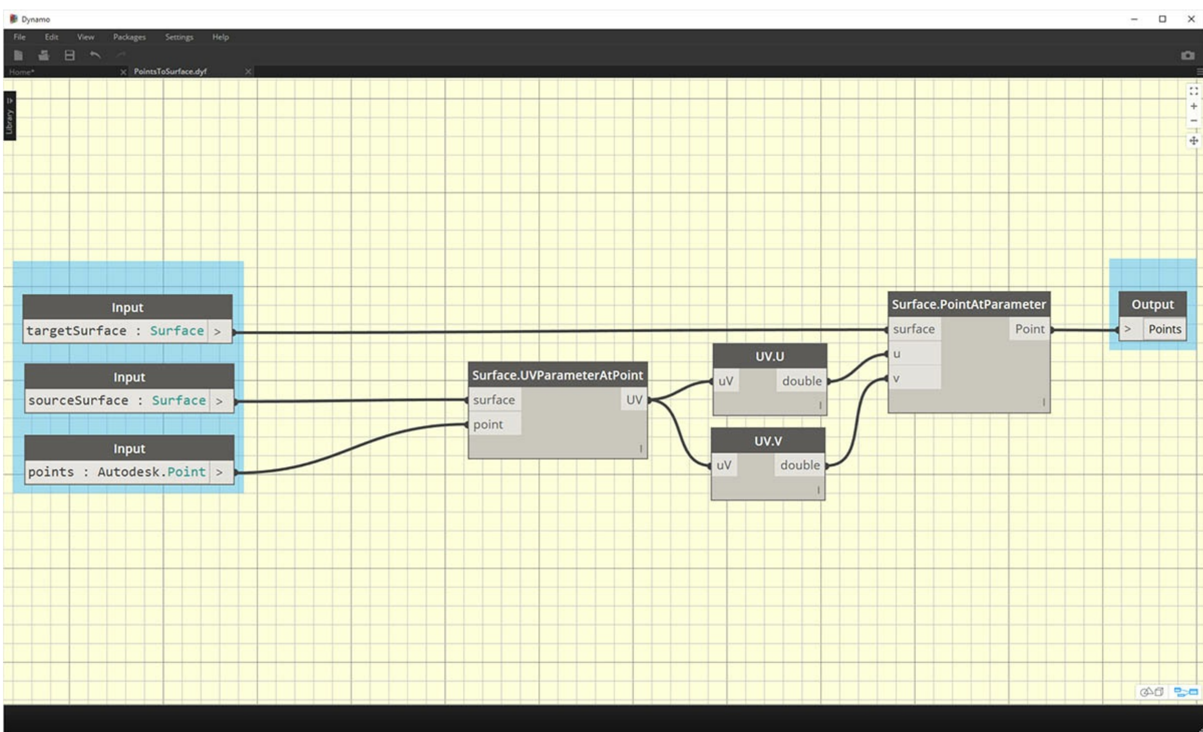
1. После установки пользовательские узлы должны быть доступны в группе DynamoPrimer или в библиотеке Dynamo. Теперь рассмотрим структуру пакета.

Пользовательские узлы

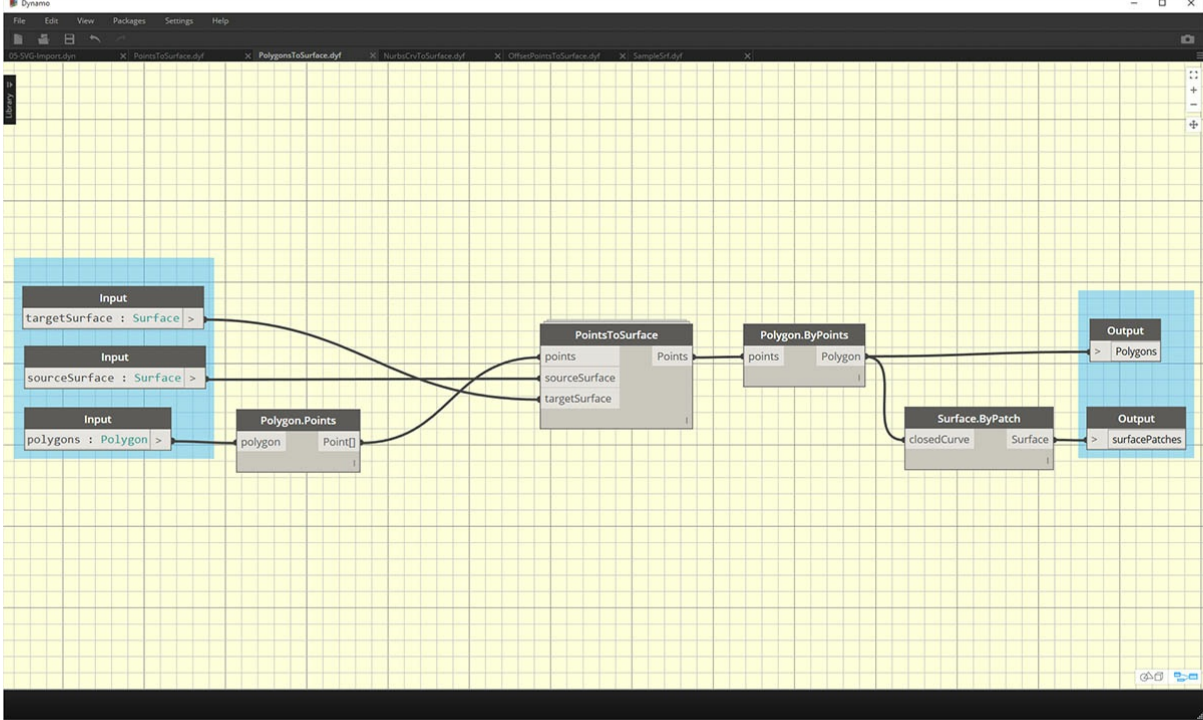
В разрабатываемом пакете есть пять пользовательских узлов, которые были созданы в качестве базовых. Ниже рассмотрим назначение каждого узла. Некоторые пользовательские узлы строятся на основе других пользовательских узлов, а графики имеют структуру, позволяющую другим пользователям легко понять их.



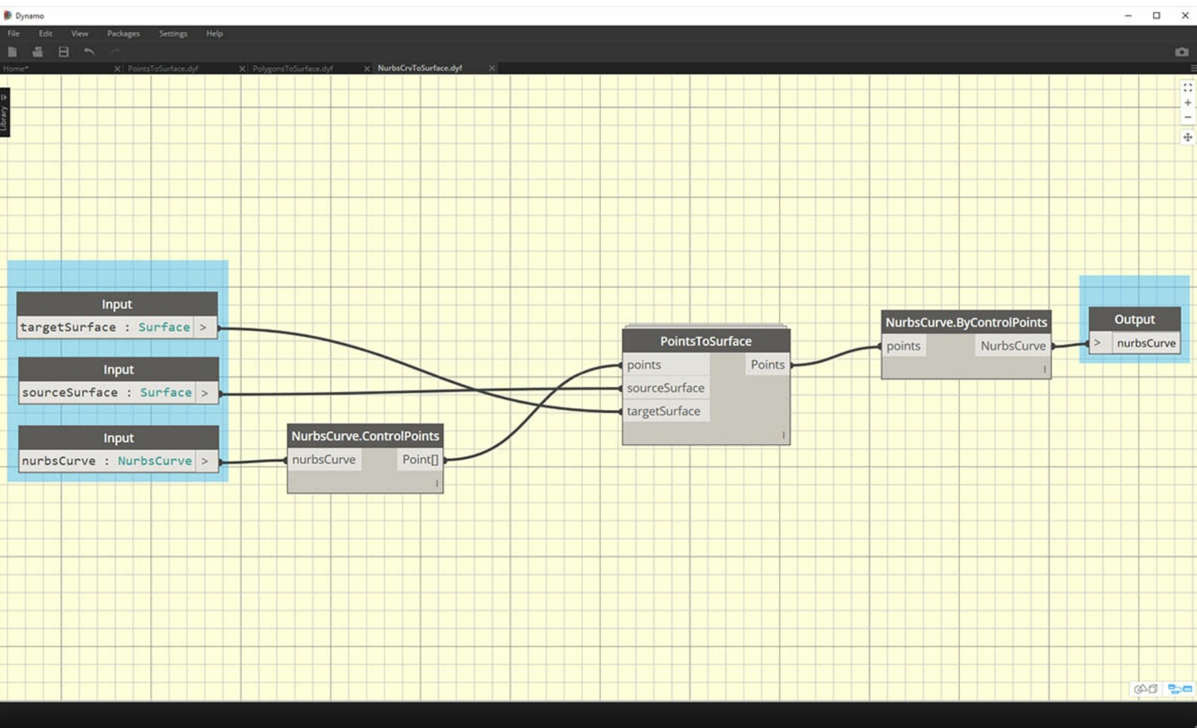
Перед нами простой пакет с пятью пользовательскими узлами. Ниже кратко рассматривается структура каждого из них.



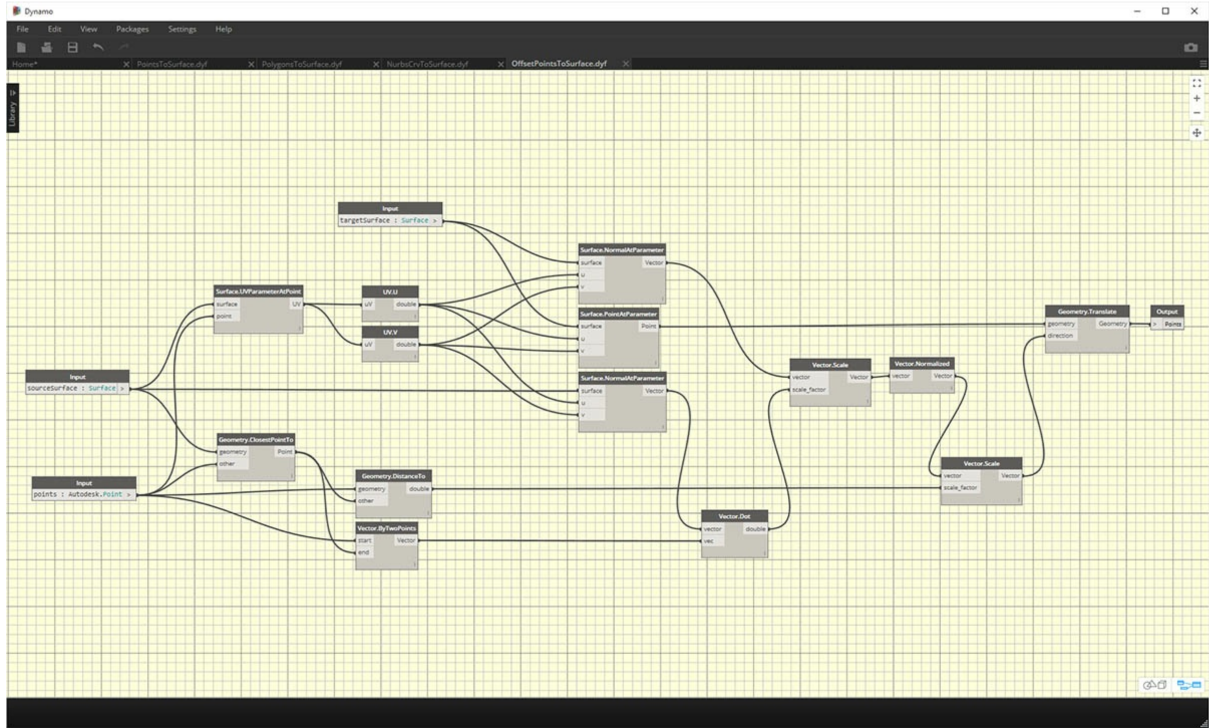
PointsToSurface. Это базовый пользовательский узел, на основе которого создаются все остальные узлы сопоставления. Говоря простым языком, данный узел сопоставляет точку UV-координаты исходной поверхности с местоположением UV-координаты целевой поверхности. Поскольку точки представляют собой простейшие геометрические объекты, на основе которых строится более сложная геометрия, этот принцип можно использовать для сопоставления 2D- и даже 3D-геометрии одной поверхности с другой.



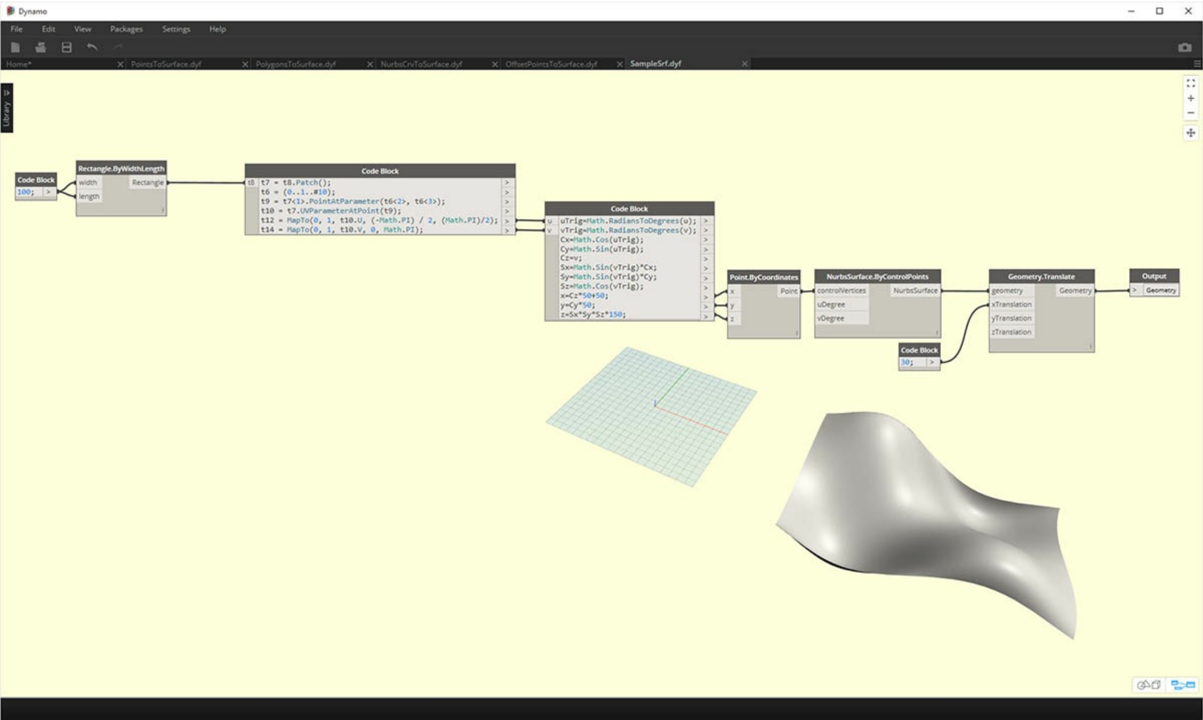
PolygonsToSurface. В данном случае для демонстрации преобразования сопоставляемых точек из одномерной в 2D-геометрию используются обычные полигоны. Обратите внимание, что в этот пользовательский узел вложен узел *PointsToSurface*. Таким образом можно сопоставить точки каждого полигона с поверхностью, а затем заново сгенерировать полигон по этим точкам. При сохранении надлежащей структуры данных (список списков точек) полигоны будут располагаться отдельно после их уменьшения до набора точек.



NurbsCrvtoSurface. Здесь работает тот же принцип, что и с узлом *PolygonsToSurface*. Однако вместо сопоставления полигональных точек сопоставляются управляющие точки NURBS-кривой.



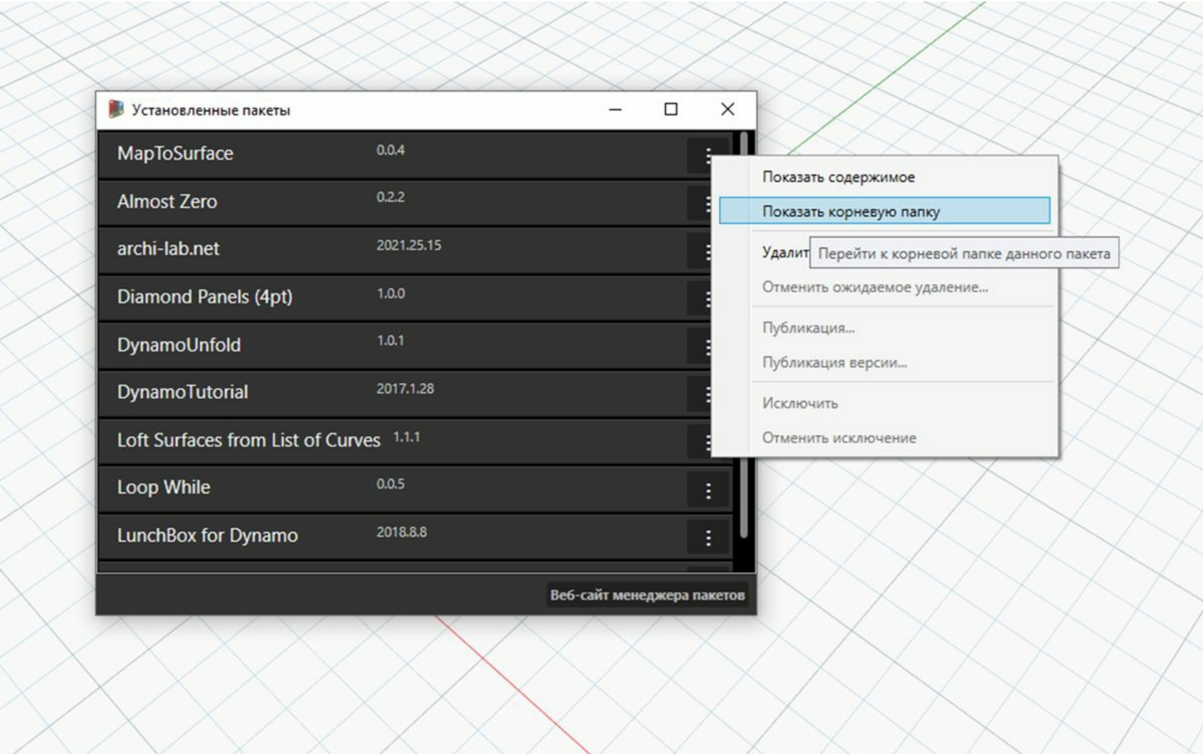
OffsetPointsToSurface. Этот узел немного сложнее, но так же, как *PointsToSurface* сопоставляет точки одной поверхности с другой. Однако в этом случае узел учитывает точки, которые отсутствуют на исходной поверхности, вычисляет расстояние от них до ближайшего UV-параметра и сопоставляет это расстояние с нормалью целевой поверхности в соответствующей UV-координате. Это будет проще объяснить с помощью файлов примеров.



SampleSrf. Это простой узел, который позволяет создать параметрическую поверхность на основе исходной сетки для сопоставления с волнистой поверхностью в файлах примеров.

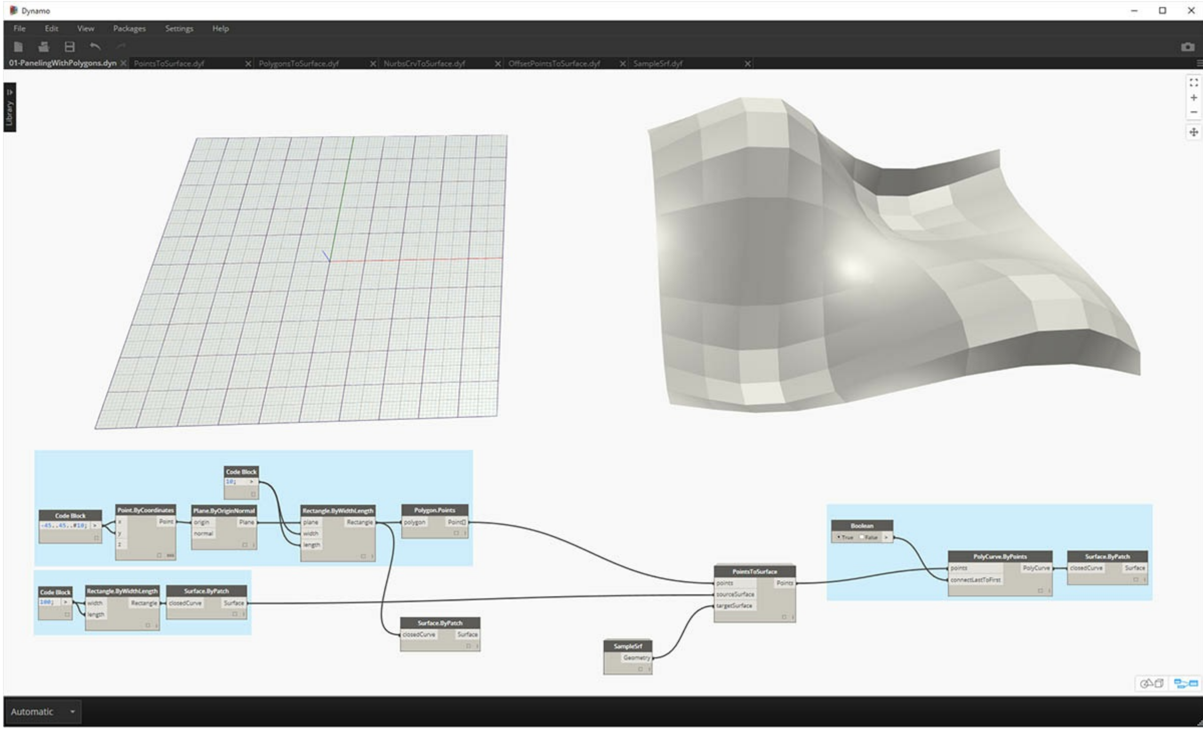
Файлы примеров

Файлы примеров можно найти в корневой папке пакета (в Dynamo перейдите в папку *«Пакеты»* > *«Управление пакетами...*)».

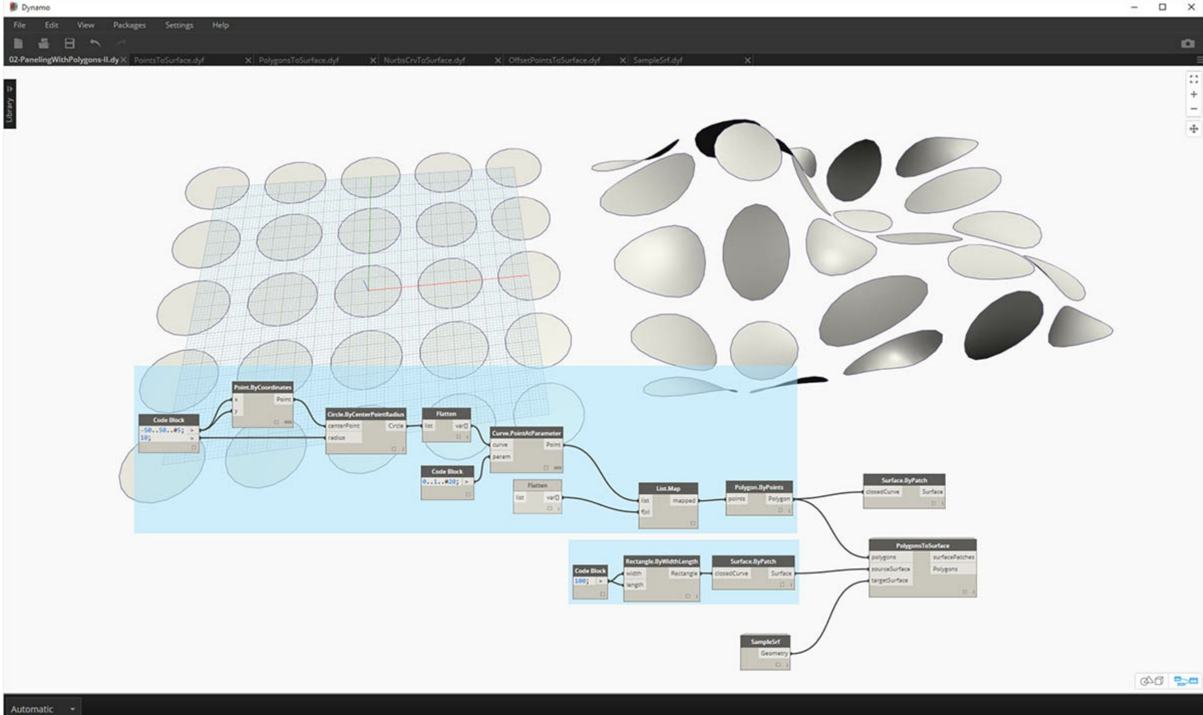


В окне «Управление пакетами» щелкните три вертикально расположенные точки справа от элемента *MapToSurface* и выберите команду*Показать корневую папку*.

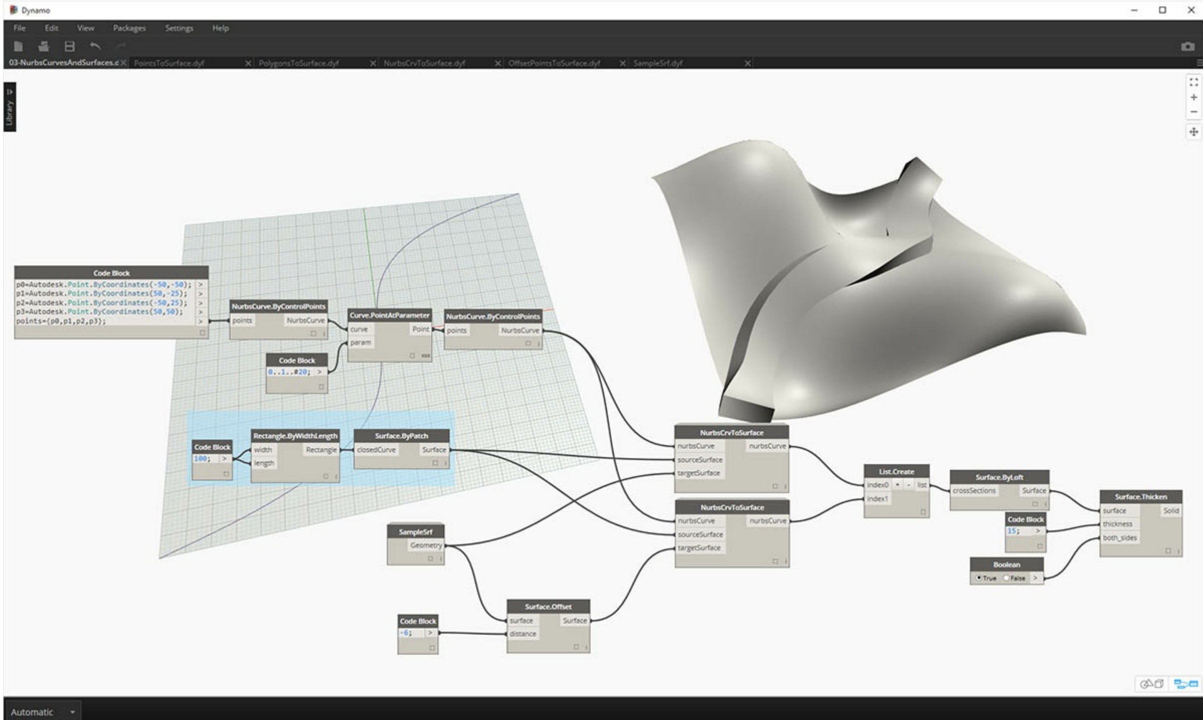
После открытия корневой папки перейдите в папку *extra*, в которой содержатся все файлы пакета, не являющиеся пользовательскими узлами. В этой папке хранятся файлы примеров (при наличии) для пакетов Dynamo. На снимках экрана ниже показаны принципы, реализованные в файлах примеров.



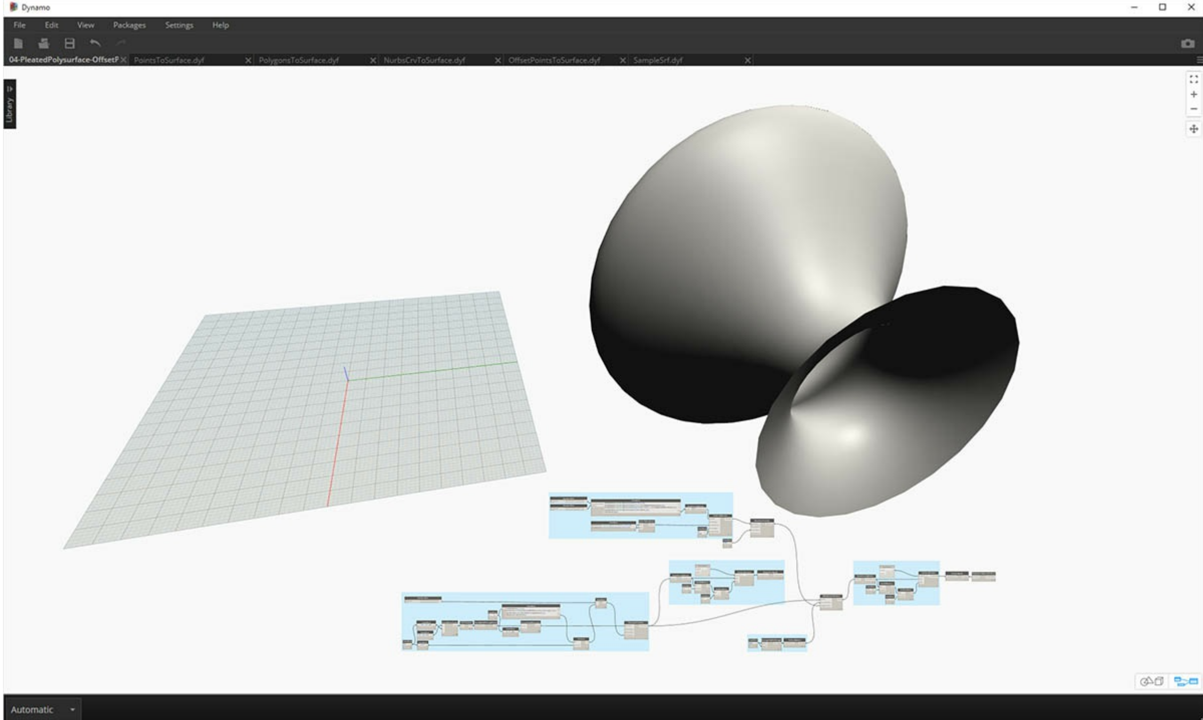
01-PanelingWithPolygons. В этом файле демонстрируется, как с помощью узла *PointsToSurface* создавать панели поверхности на основе сетки из прямоугольников. Изображение должно быть знакомым, так как аналогичный рабочий процесс был представлен в предыдущей главе.



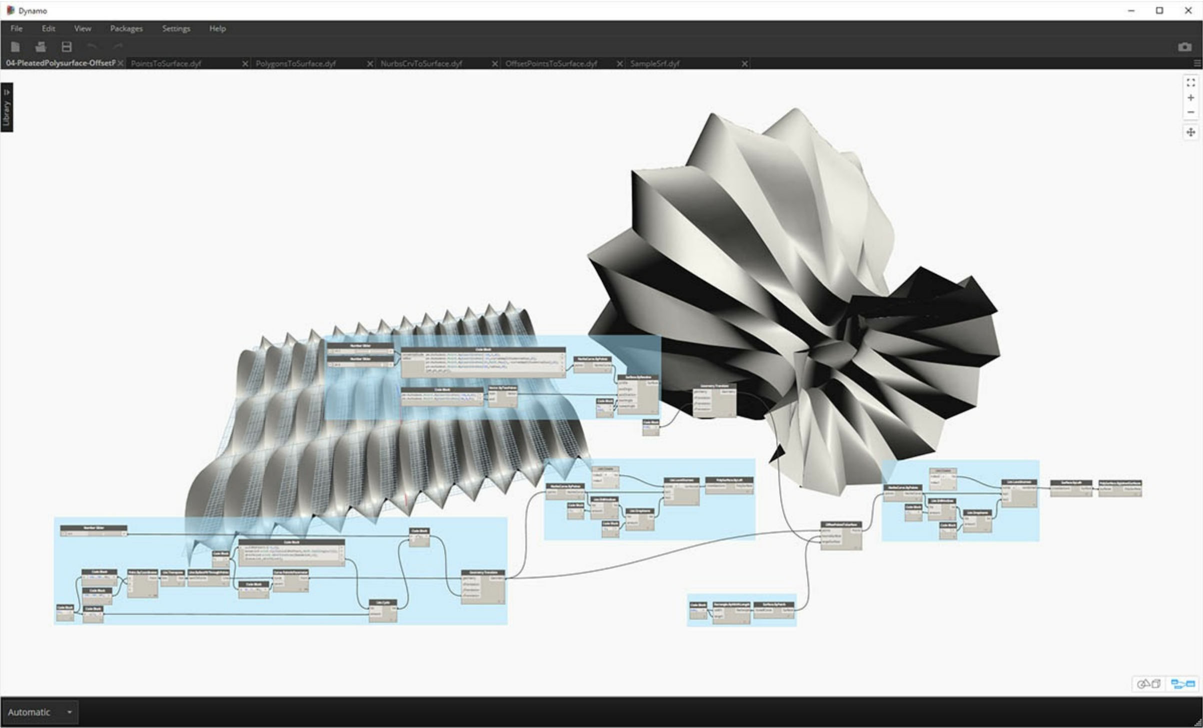
02-PanelingWithPolygons-II. В данном файле, где используется похожий рабочий процесс, показан алгоритм сопоставления окружностей (или полигонов, представляющих окружности) одной поверхности с окружностями другой. При этом используется узел *PolygonsToSurface*.



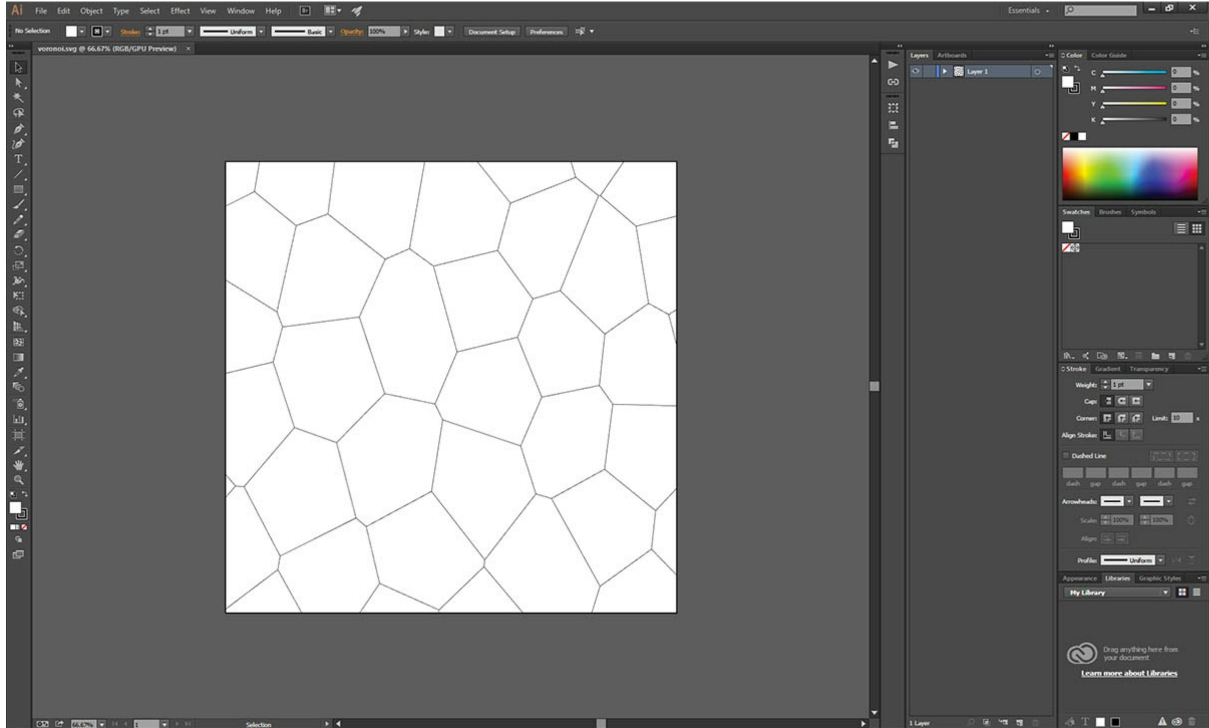
03-NurbsCrvsAndSurface. Это более сложный файл примера, так как в нем используется узел NurbsCrvToSurface. Целевая поверхность смещена на заданное расстояние, а NURBS-кривая сопоставлена с исходной целевой поверхностью и смещенной поверхностью. К двум сопоставленным кривым применяется функция лофтинга для создания поверхности, толщина которой затем увеличивается. Полученное тело имеет неровность, соответствующую нормалям целевой поверхности.



04-PleatedPolysurface-OffsetPoints. В этом файле примера показано, как сопоставить исходную гофрированную полиповерхность с целевой поверхностью. Исходная и целевая поверхности представляют собой прямоугольную поверхность, которая проходит по сетке и поверхности вращения соответственно.



04-PleatedPolysurface-OffsetPoints. Исходная полиповерхность, сопоставленная с целевой поверхностью.



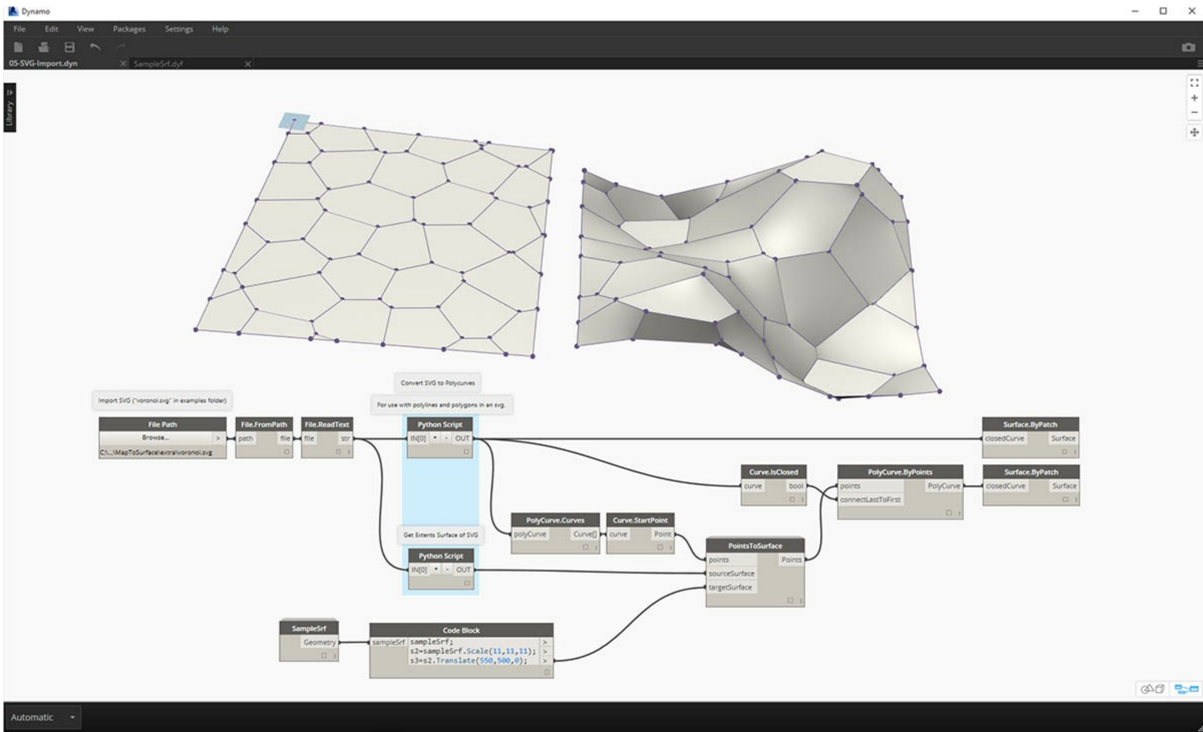
05-SVG-Import. Так как с помощью пользовательских узлов можно сопоставлять различные типы кривых, последний файл ссылается на файл SVG, экспортированный из Illustrator, и сопоставляет импортированные кривые с целевой поверхностью.

Edit Python Script...

```
import clr
import re
clr.AddReference("System.Xml")
import System.Xml

xmldoc = System.Xml.XmlDocument()
xmldoc.Load(IN[0])

OUT=[]

#for shorthanded quadratic and cubic bezier curves
svgDict={}
svgDict["et"]=[]
svgDict["ed"]=[]

def segsFromPath(data):
    subOUT=[]
    splitPath=re.findall('[A-Za-z][^A-Za-z]*',"".join(line.strip() for line in data))
    return splitPath

def viewBox():
    items = xmldoc.GetElementsByTagName("svg")
    for item in items:
        box=item.GetAttribute("viewBox")
    nums=box.split(' ')
    floats=[]
    for num in nums:
        floats.append(float(num))
    OUT.append(["viewBox",floats])


def ptsFromPoly(data):
    subOUT=[]
    pts=data.split(" ")
    for points in pts:
        ptList=points.split(",")
        if len(ptList)>1:
            numX=float(ptList[0])
            numY=float(ptList[1])
            geoPt=[numX,numY]
            subOUT.append(geoPt)
```

Accept Changes Cancel

05-SVG-Import. В результате синтаксического анализа файла SVG кривые преобразуются из формата XML в поликривые Dynamo.

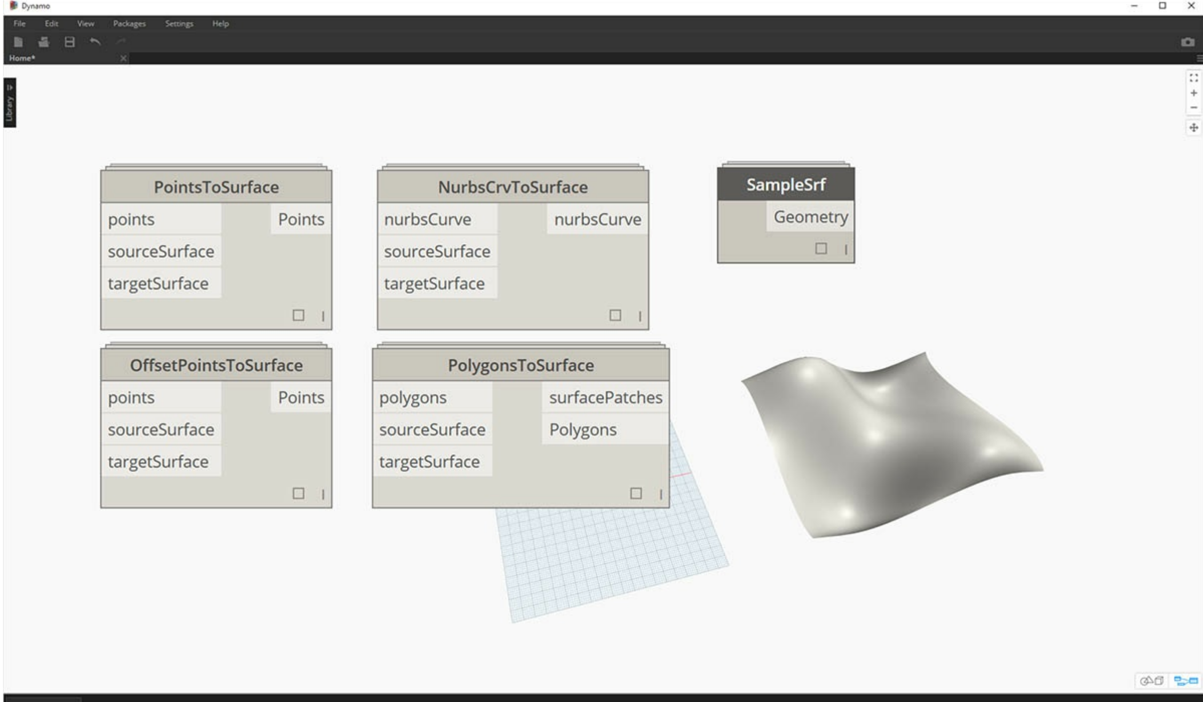


05-SVG-Import. Импортированные кривые сопоставляются с целевой поверхностью. Это позволяет явным образом (с помощью указателя) создавать панели поверхности в приложении Illustrator, импортировать их в Dynamo и применять к целевой поверхности.

Публикация пакетов

Публикация пакетов

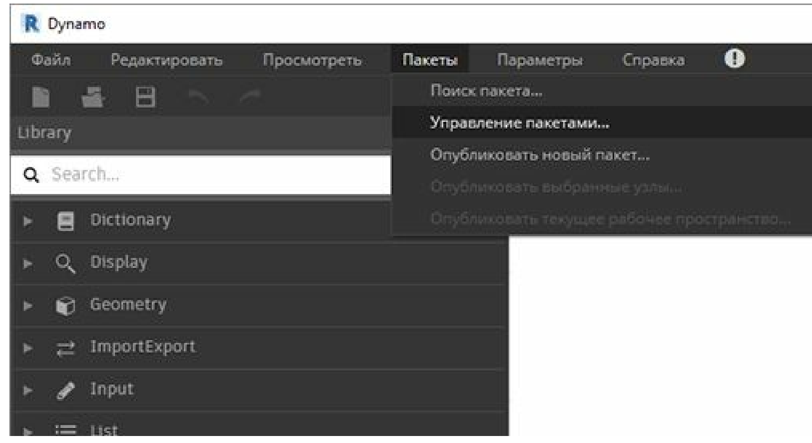
В предыдущих разделах мы подробно рассмотрели процесс настройки пакета *MapToSurface* с использованием пользовательских узлов и файлов примеров. Но как опубликовать пакет, разработка которого была выполнена на локальном компьютере? В этом примере мы рассмотрим процесс публикации пакета из набора файлов в локальной папке.



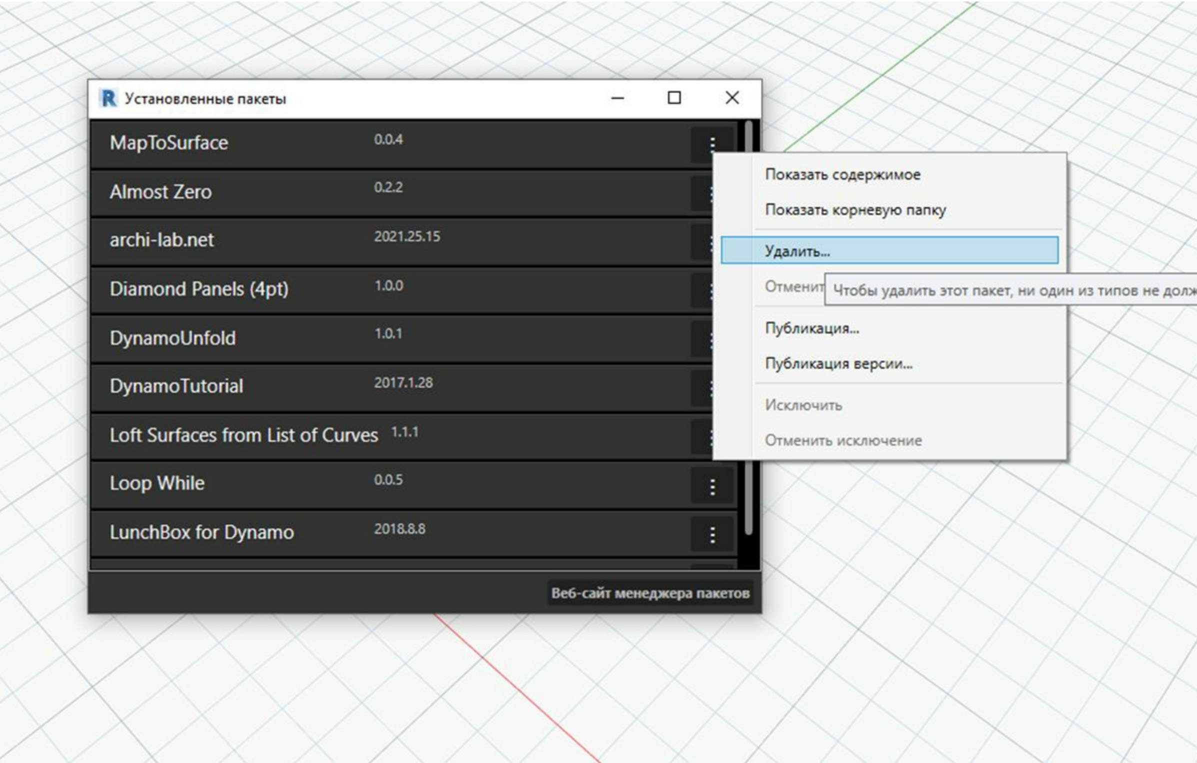
Существует много способов публикации пакета. Мы рекомендуем придерживаться следующего процесса: **сначала опубликуйте пакет на локальном компьютере, там же выполните его разработку и, наконец, опубликуйте пакет в интернете**. В рамках примера мы будем работать с папкой, содержащей все файлы пакета.

Удаление пакета

Если пакет MapToSurface уже был установлен в рамках предыдущего урока, его необходимо удалить, прежде чем приступать к публикации в рамках этого урока, чтобы избежать дублирования пакетов.



Выберите «*Пакеты*» > «*Управление пакетами...*».

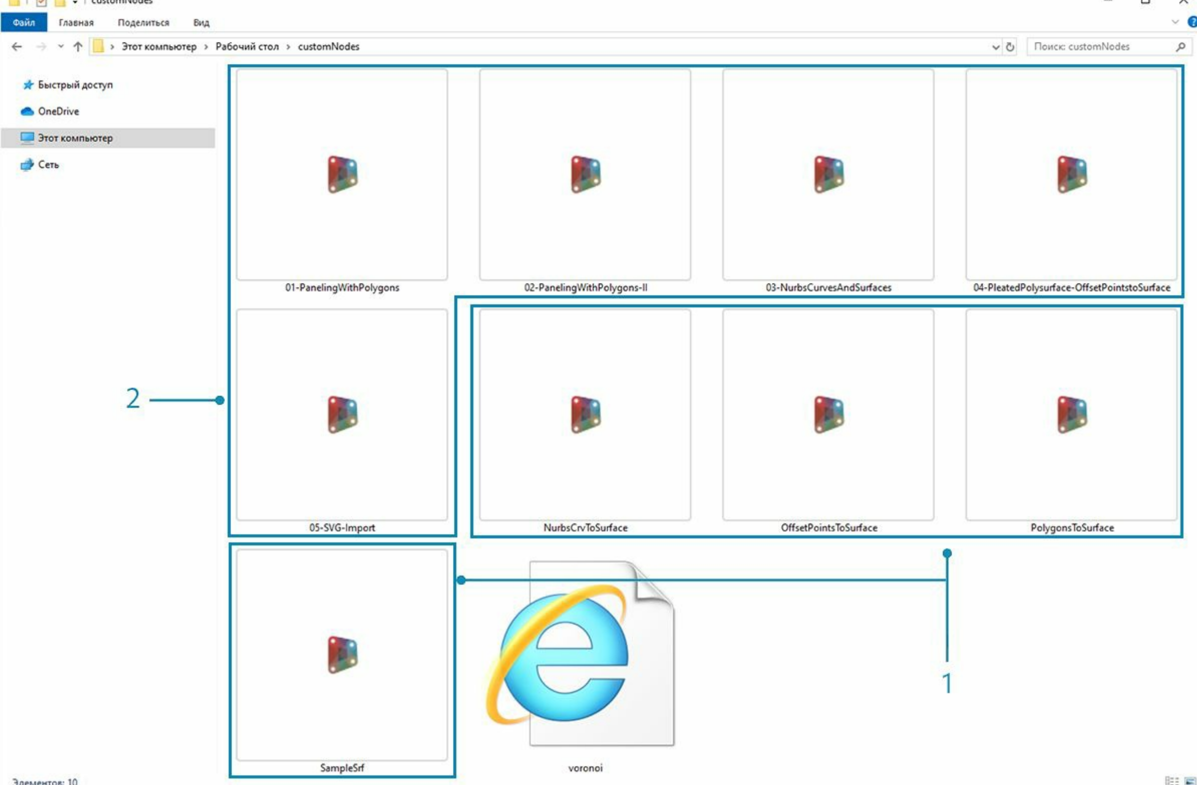


Нажмите кнопку напротив *MapToSurface* и выберите *Удалить...*. Перезапустите Dynamo. Проверьте окно *Управление пакетами* убедитесь, что пакет *MapToSurface* отсутствует. Теперь все готово к началу работы.

Публикация пакета на локальном компьютере

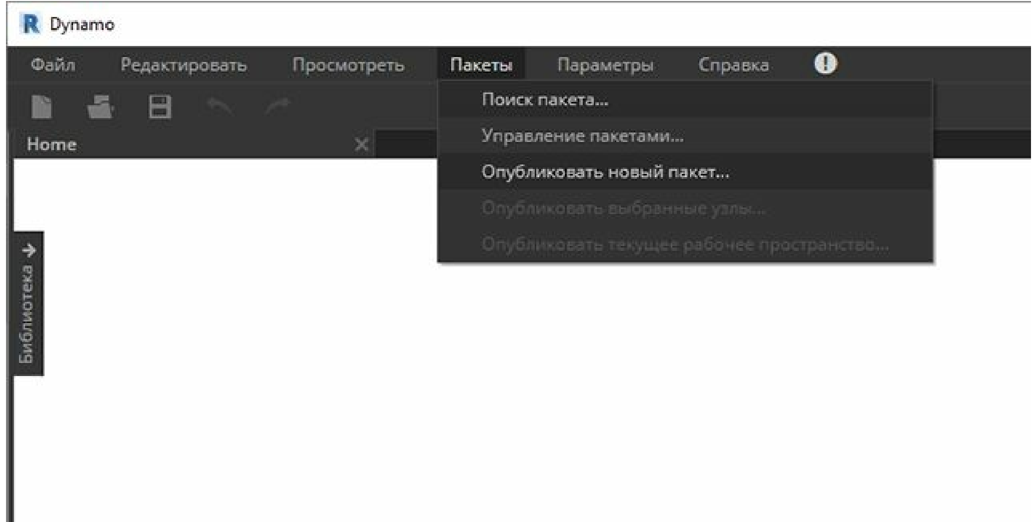
Примечание. На момент создания этого документа функция публикации пакетов Dynamo доступна только в Dynamo for Revit и Dynamo for Civil 3D. В Dynamo Sandbox функция публикации отсутствует.

Скачайте и распакуйте файлы примеров, идущие в комплекте с данным пакетом (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. MapToSurface.zip

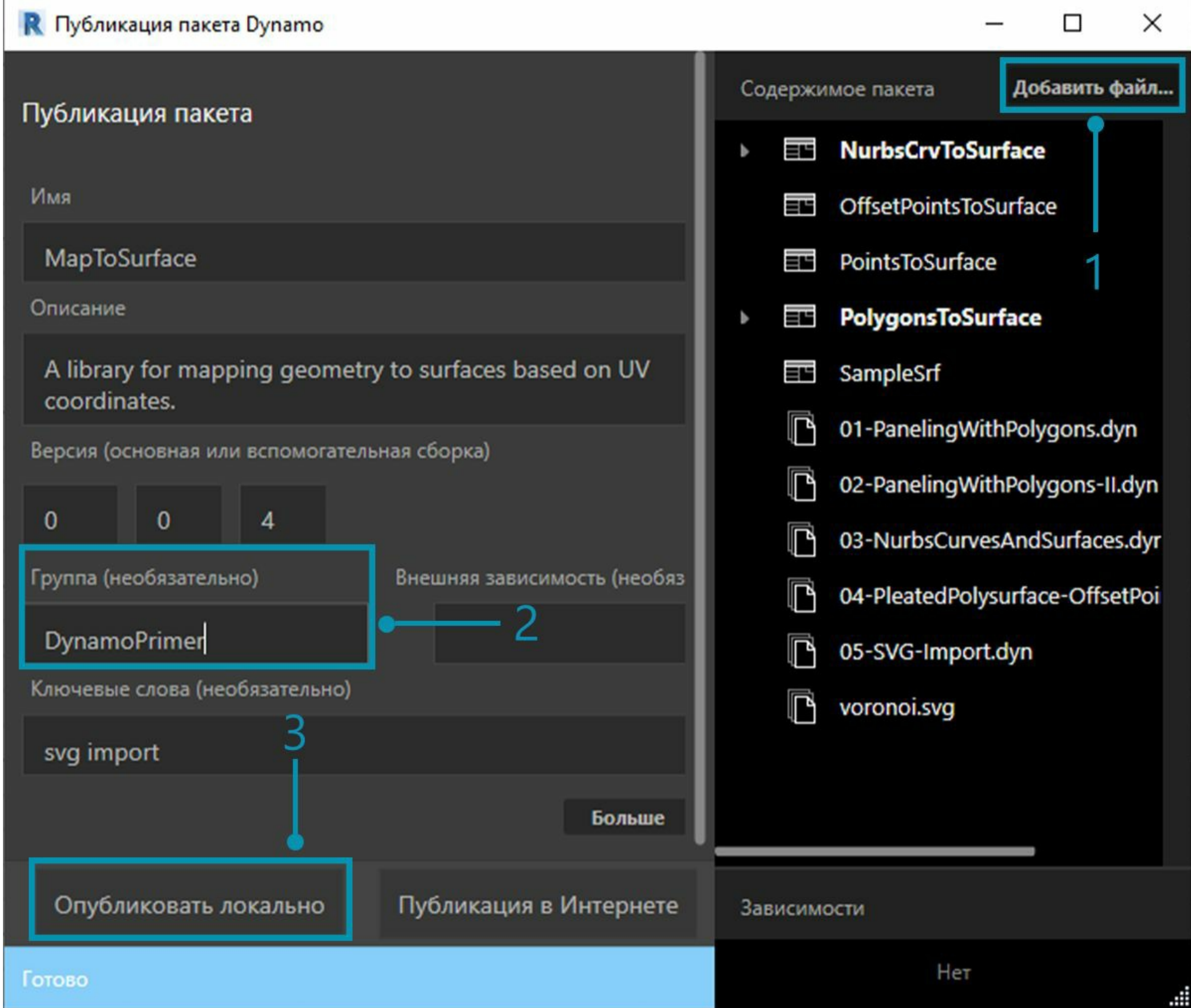


Это первая версия данного пакета. Все файлы примеров и пользовательские узлы размещены в одной папке. Поскольку папка готова к использованию, можно сразу приступать к выгрузке в менеджер пакетов Dynamo.

1. Эта папка содержит пять пользовательских узлов (DYF).
2. В ней также есть пять файлов примеров (DYN) и один импортированный файл векторов (SVG). Эти файлы будут задействованы в рамках вводных упражнений по обучению работе с пользовательскими узлами.



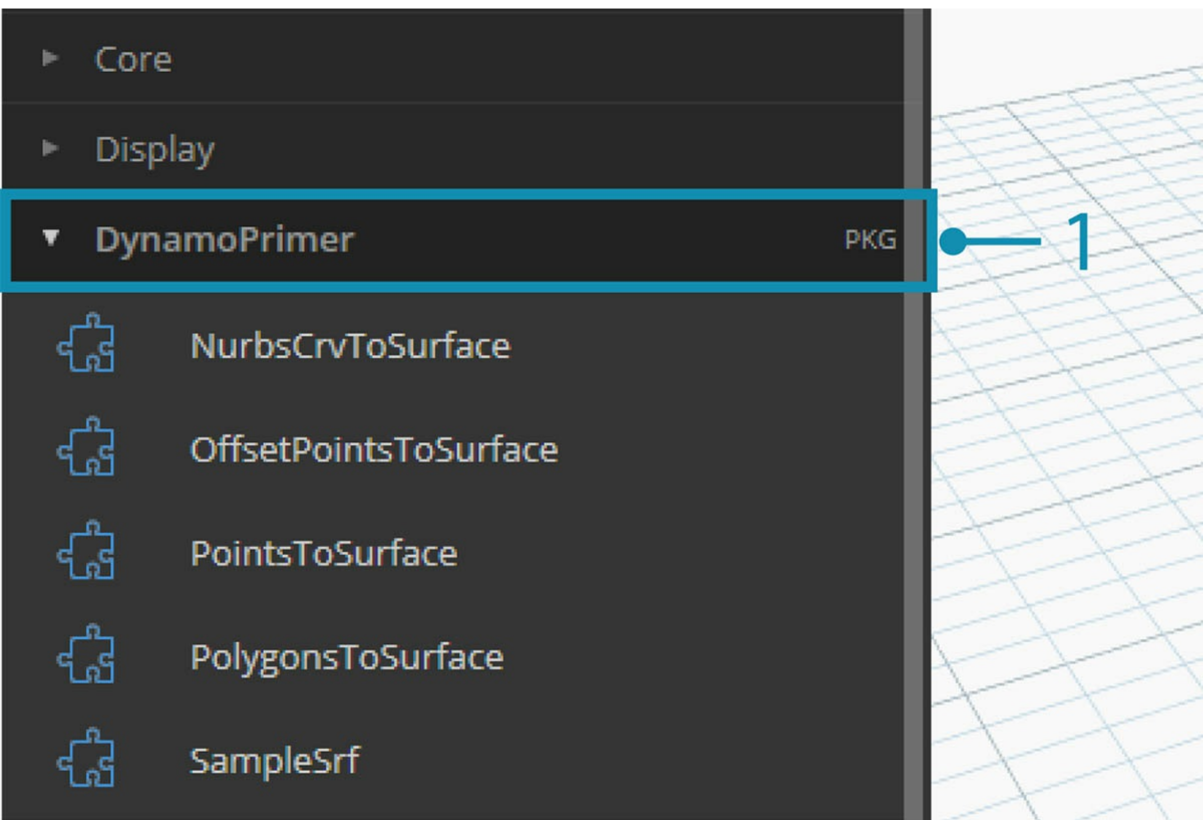
В Dynamo выберите *«Пакеты»* > *«Опубликовать новый пакет...»*.



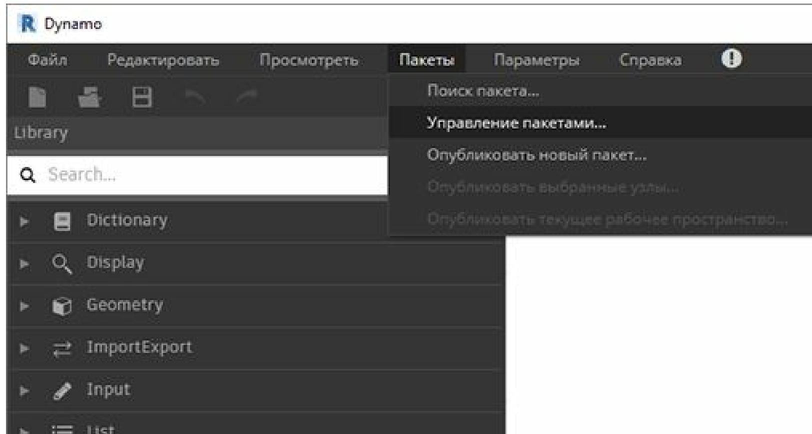
В левой части окна *Публикация пакета Dynamo* отобразятся предварительно заданные сведения о пакете.

1. В правой части экрана отображаются файлы, добавленные из структуры папок с помощью кнопки *Добавить файл*. Чтобы добавить

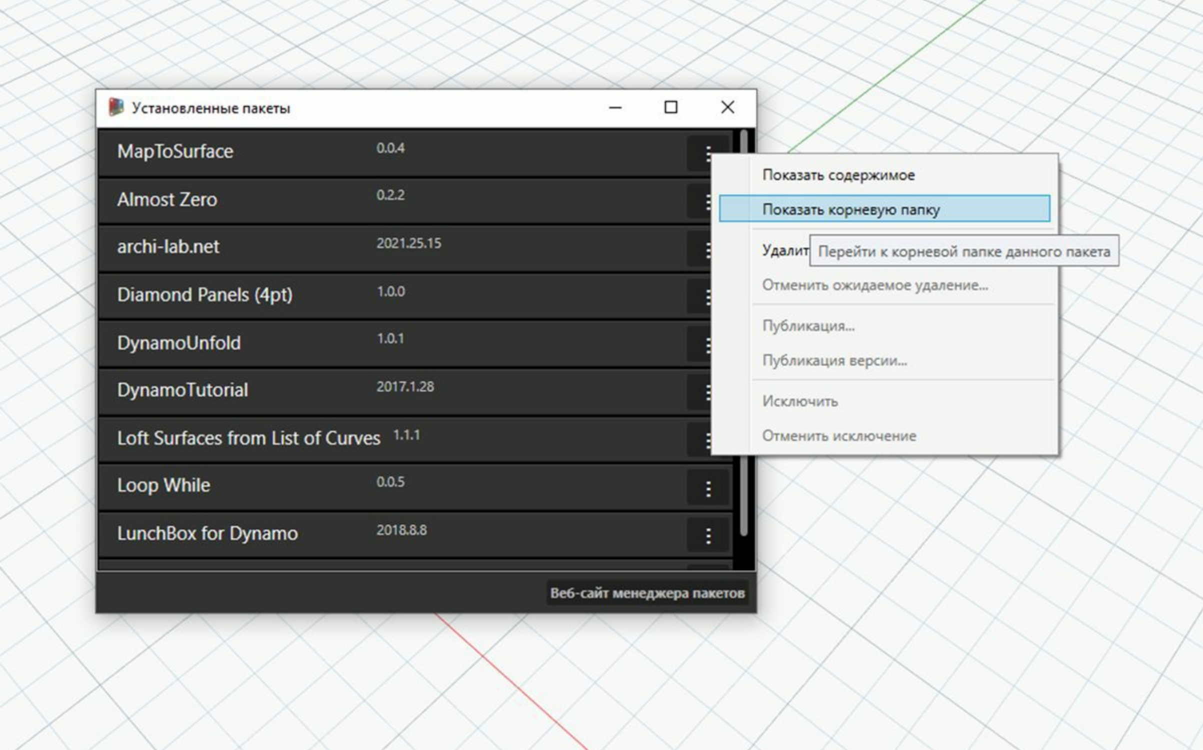
файлы, которые не являются файлами DYF, необходимо изменить тип файла, заданный в окне обозревателя, на **Все файлы(.)**". Обратите внимание, что добавлены все файлы — и файлы пользовательских узлов (DYF), и файлы примеров (DYN). При публикации пакета программа Dynamo автоматически разобьет их по категориям.
2. В поле «Группа» указывается, в какой группе можно будет найти пользовательские узлы в интерфейсе Dynamo.
3. Нажмите кнопку «Опубликовать локально» для публикации пакета. Обратите внимание, что нужно нажать именно *Опубликовать локально*, а **не** *Публикация в Интернете*, чтобы избежать появления повторяющихся пакетов в менеджере пакетов.



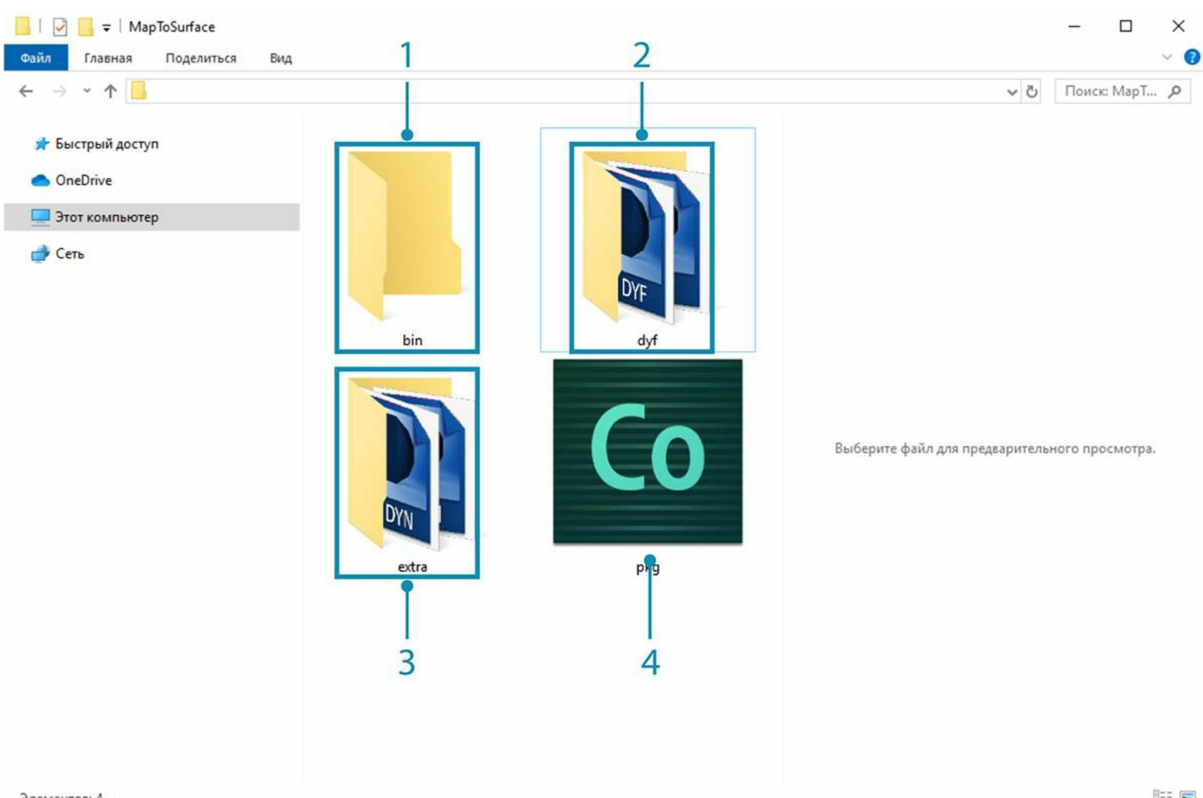
1. После публикации пользовательские узлы должны быть доступны в группе DynamoPrimer или в библиотеке Dynamo.



Теперь перейдем в корневую папку и посмотрим, как только что созданный пакет был отформатирован в Dynamo. Для этого выберите «*Пакеты*» > «*Управление пакетами...*».



В окне «Управление пакетами» щелкните три вертикально расположенные точки справа от элемента *MapToSurface* и выберите команду *Показать корневую папку*.

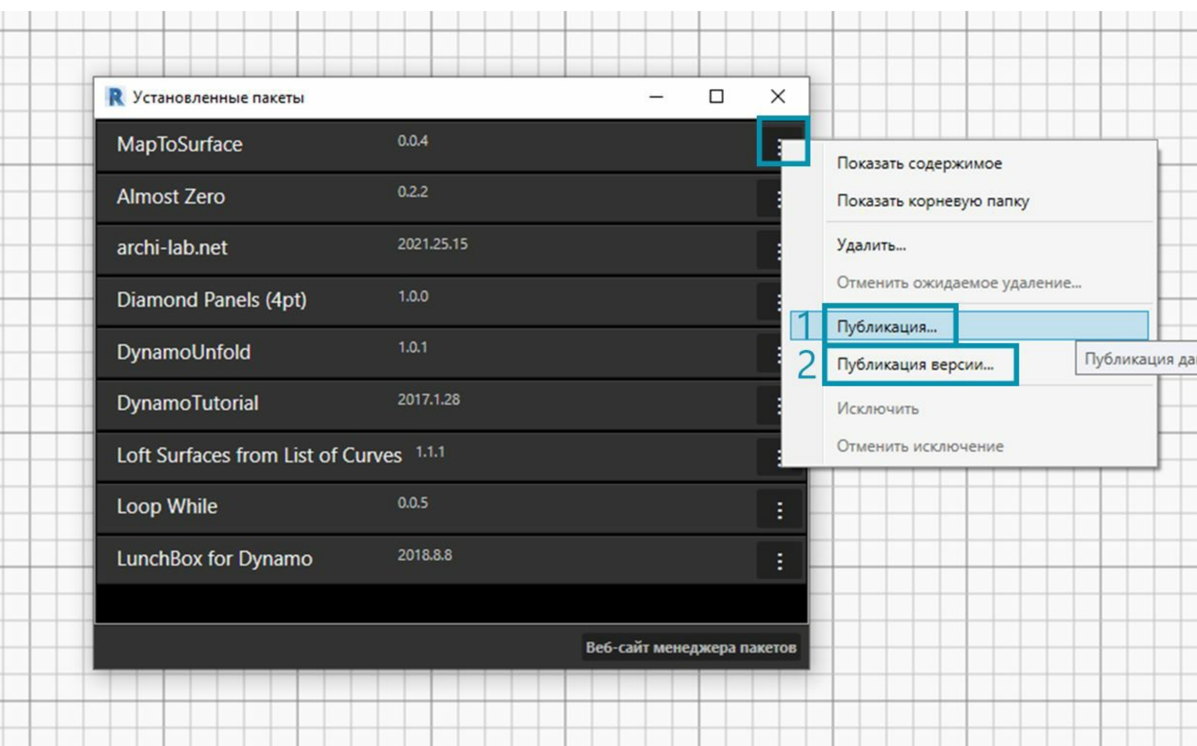


Обратите внимание, что поскольку пакет был опубликован локально, корневая папка находится в локальной папке пакета. Dynamo ссылается на эту папку для чтения пользовательских узлов. Поэтому при локальной публикации пакета важно указывать постоянную папку (а не рабочий стол, например). Структура папок пакета Dynamo выглядит следующим образом.

1. В папке *bin* хранятся файлы DLL, созданные с помощью библиотек C# или Zero-Touch. В этот пакет такие файлы не входят, поэтому данная папка пуста.
2. В папке *dyf* хранятся пользовательские узлы. Открыв ее, можно просмотреть все пользовательские узлы (файлы DYF), входящие в пакет.
3. В папке extra хранятся все дополнительные файлы. Сюда входят файлы Dynamo (DYN), а также дополнительные файлы других

формaтов (SVG, XLS, JPEG, SAT и т. д.).
4. Файл PKG — это стандартный текстовый файл, определяющий параметры пакета. Он создается в Dynamo автоматически, но если требуется подробная настройка, то параметры можно отредактировать.

Публикация пакета в интернете



Примечание. Данная процедура предназначена только для публикации пакетов, разработанных пользователями.

1. Когда пакет будет готов к публикации, откройте окно «Управление пакетами», нажмите кнопку справа от MapToSurface и выберите *Публикация*....
2. Если требуется обновить ранее опубликованный пакет, выберите «Публикация версии», и приложение Dynamo обновит пакет в интернете с учетом новых файлов в корневой папке этого пакета. Проще простого.

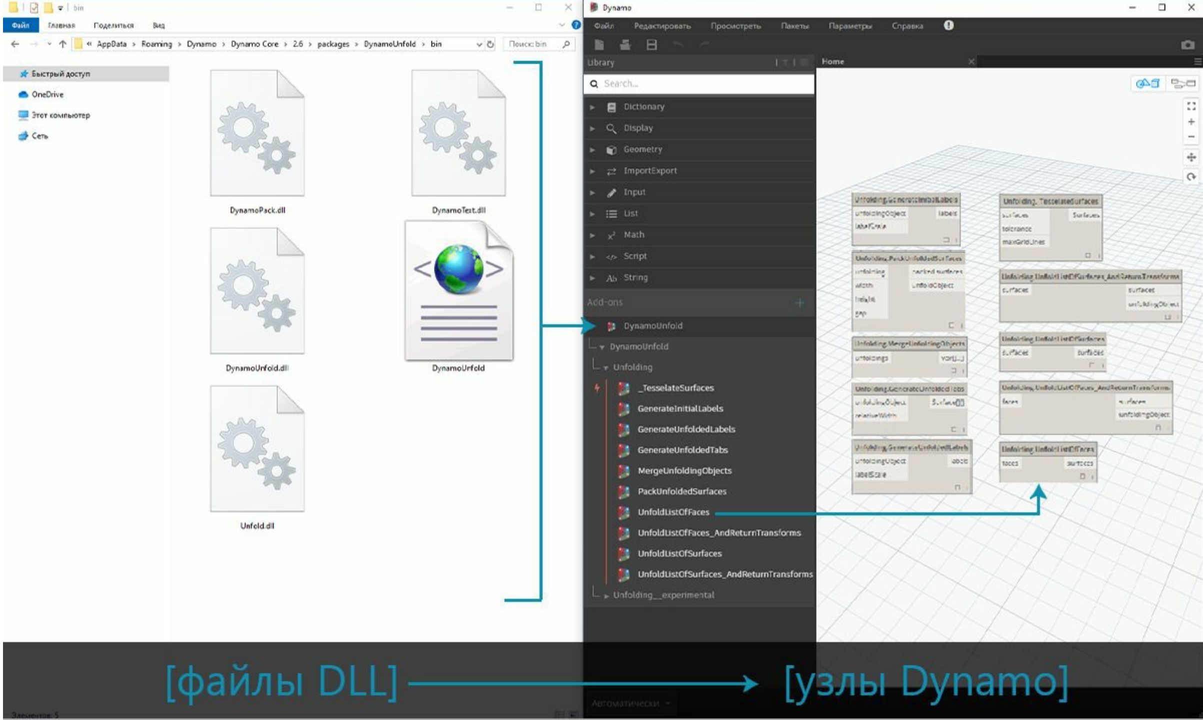
Публикация версии...

Если файлы в корневой папке опубликованного пакета были изменены, можно опубликовать новую версию этого пакета, выбрав параметр *Публикация версии...* в окне *Управление пакетами*. Эта функция позволяет с легкостью вносить в содержимое необходимые обновления и обмениваться данными с сообществом пользователей. Пользоваться функцией *Публикация версии* могут только разработчики соответствующего пакета.

Импорт Zero Touch

Что такое Zero-Touch

Импорт Zero-Touch — это метод, позволяющий легко и быстро импортировать библиотеки C# одним щелчком мыши. Приложение Dynamo считывает общие методы из файла *DLL* и преобразует их в узлы Dynamo. Функцию Zero-Touch можно использовать для разработки пользовательских узлов и пакетов, а также для импорта внешних библиотек в среду Dynamo.



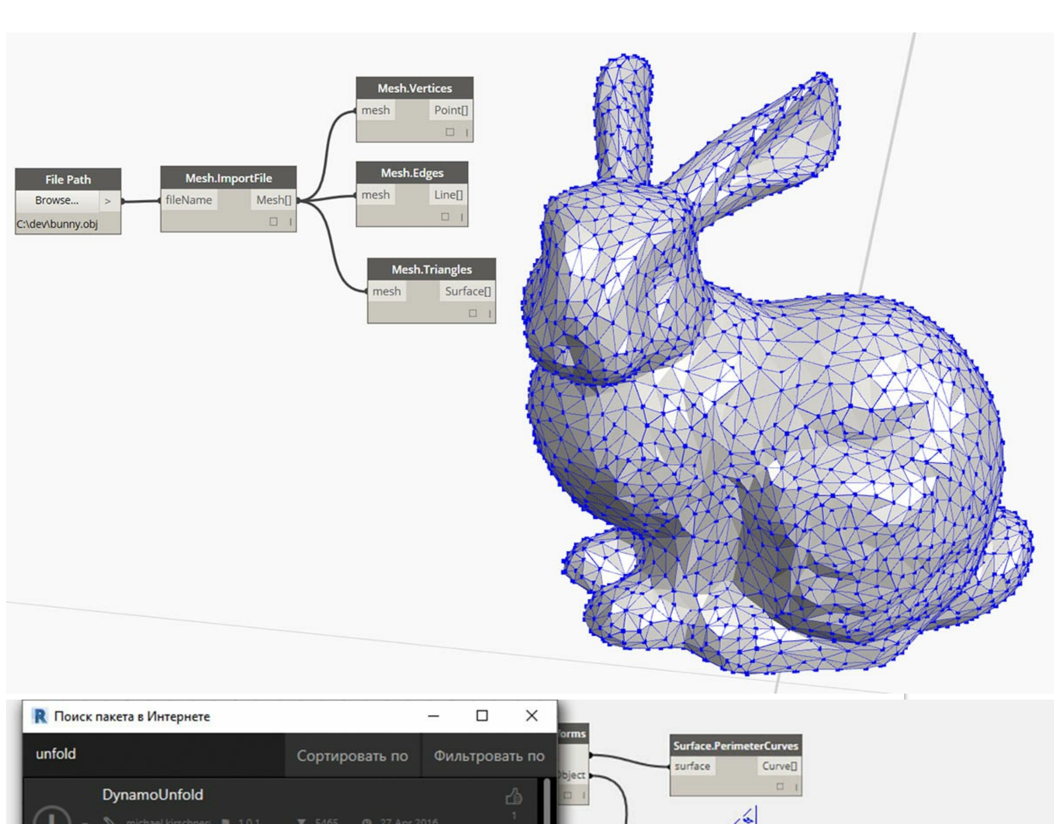
Zero-Touch позволяет импортировать библиотеки, в том числе разработанные не в Dynamo, и создавать наборы новых узлов. Эта функция является воплощением принципа кросс-платформенности, на котором основывается проект Dynamo.

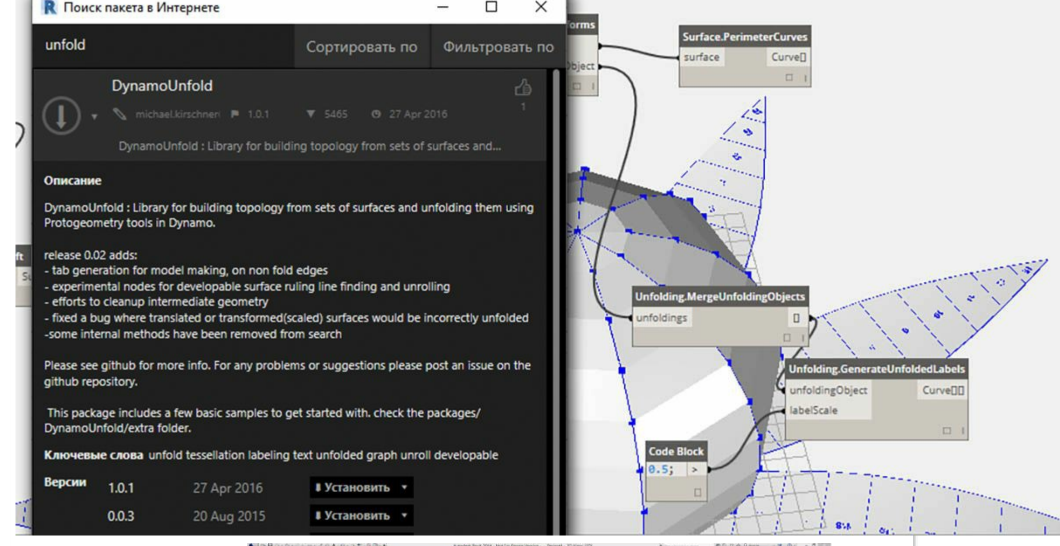
В этом разделе показан процесс импорта сторонней библиотеки с помощью функции Zero-Touch. Дополнительные сведения о разработке пользовательской библиотеки Zero-Touch см. на странице справки Wiki по работе с Dynamo.

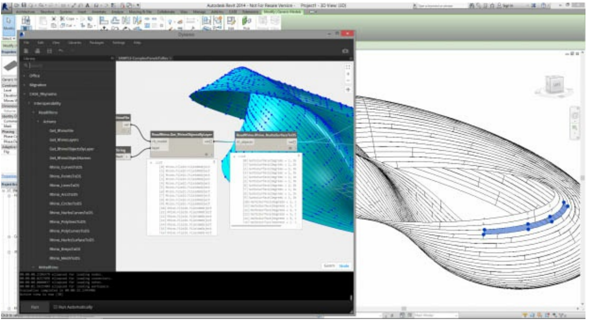
Пакеты Zero-Touch

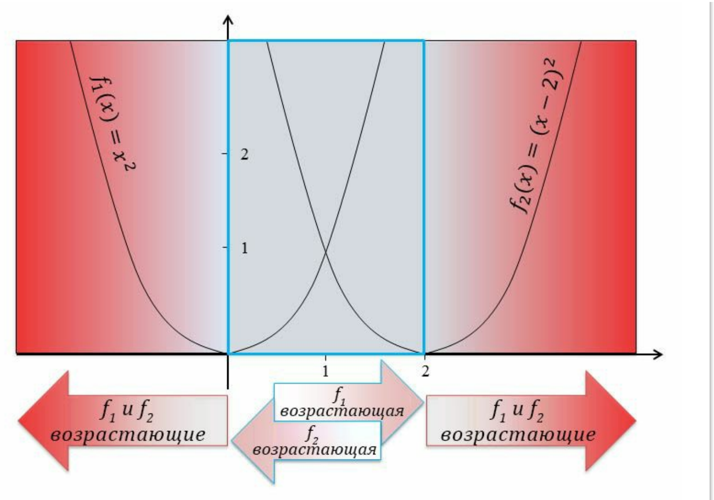
Пакеты Zero-Touch являются хорошим дополнением к пользовательским узлам. В таблице ниже приведены некоторые пакеты, в которых используются библиотеки C#. Дополнительные сведения о пакетах см. в соответствующем разделе приложения.

Логотип/изображение	


File Path
Browse...
C:\dev\bunny.obj
Mesh.ImportFile
fileName
Mesh[]
Mesh.Vertices
mesh
Point[]
Mesh.Edges
mesh
Line[]
Mesh.Triangles
mesh
Surface[]


Поиск пакета в Интернете
unfold
Сортировать по
Фильтровать по
DynamoUnfold
michael.kirschner
1.0.1
5465
27 Apr 2016
DynamoUnfold : Library for building topology from sets of surfaces and...
Описание
DynamoUnfold : Library for building topology from sets of surfaces and unfolding them using Protogeometry tools in Dynamo.
release 0.02 adds:
- tab generation for model making, on non fold edges
- experimental nodes for developable surface ruling line finding and unrolling
- efforts to cleanup intermediate geometry
- fixed a bug where translated or transformed(scaled) surfaces would be incorrectly unfolded
-some internal methods have been removed from search
Please see github for more info. For any problems or suggestions please post an issue on the github repository.
This package includes a few basic samples to get started with. check the packages/DynamoUnfold/extra folder.
Ключевые слова unfold tessellation labeling text unfolded graph unroll developable
Версии
1.0.1
27 Apr 2016
Установить
0.0.3
20 Aug 2015
Установить
Surface.PerimeterCurves
surface
Curve[]
Unfolding.MergeUnfoldingObjects
unfoldings
Unfolding.GenerateUnfoldedLabels
unfoldingObject
Curve[][]
labelScale
Code Block
0.5;



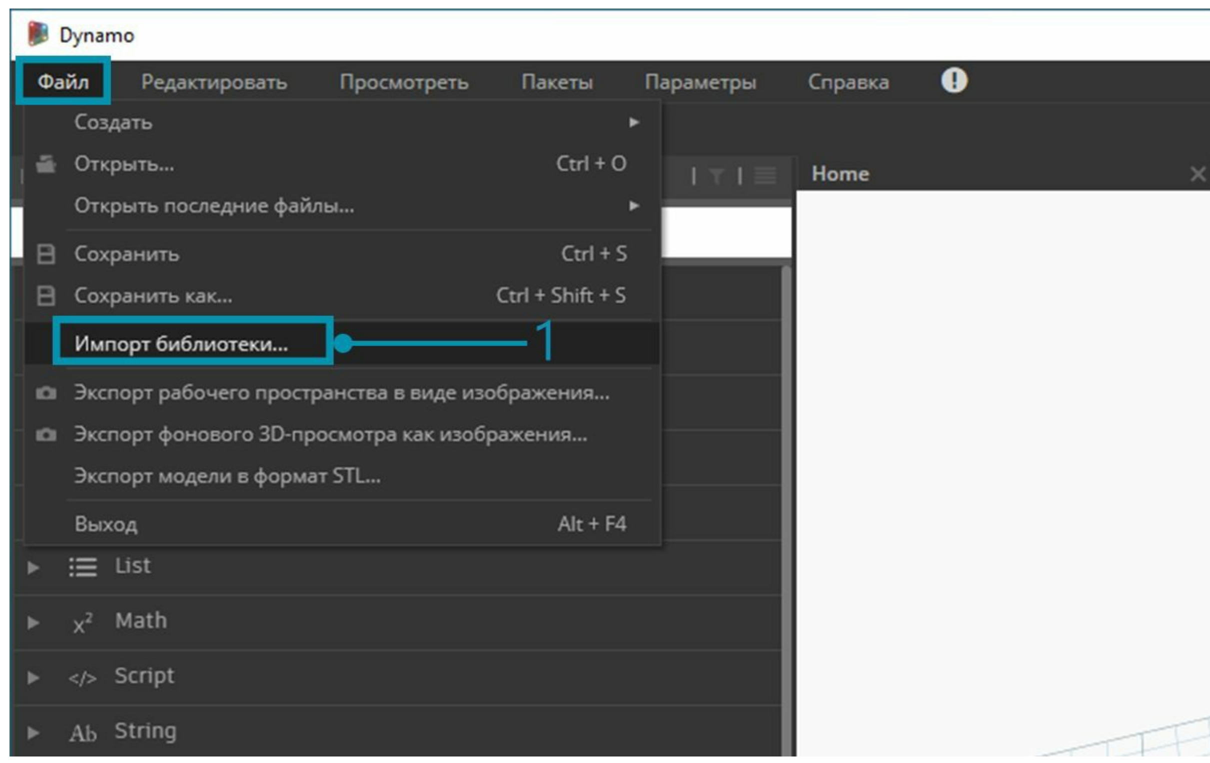


Практикум. Импорт AForge

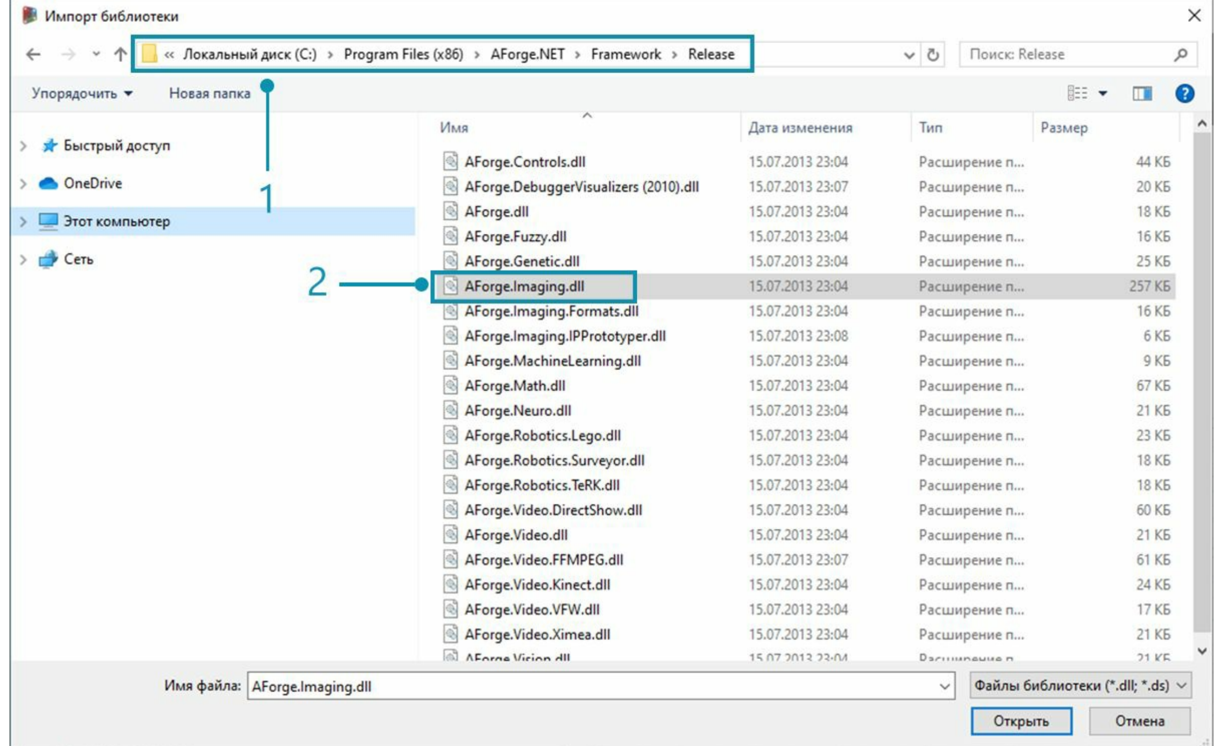
В этом примере мы рассмотрим процесс импорта внешней библиотеки AForge в формате *DLL*. AForge — это мощная библиотека, поддерживающая широкий спектр функциональных возможностей — от обработки изображений до искусственного интеллекта. При выполнении приведенных ниже упражнений по обработке изображений мы будет обращаться к классу Imaging этой библиотеки.

Скачайте и распакуйте файлы примеров, идущие в комплекте с данным пакетом (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. Zero-Touch-Examples.zip.

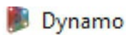
1. Для начала скачайте AForge. На странице загрузки AForge нажмите *[Download Installer]*, дождитесь завершения загрузки и выполните установку.

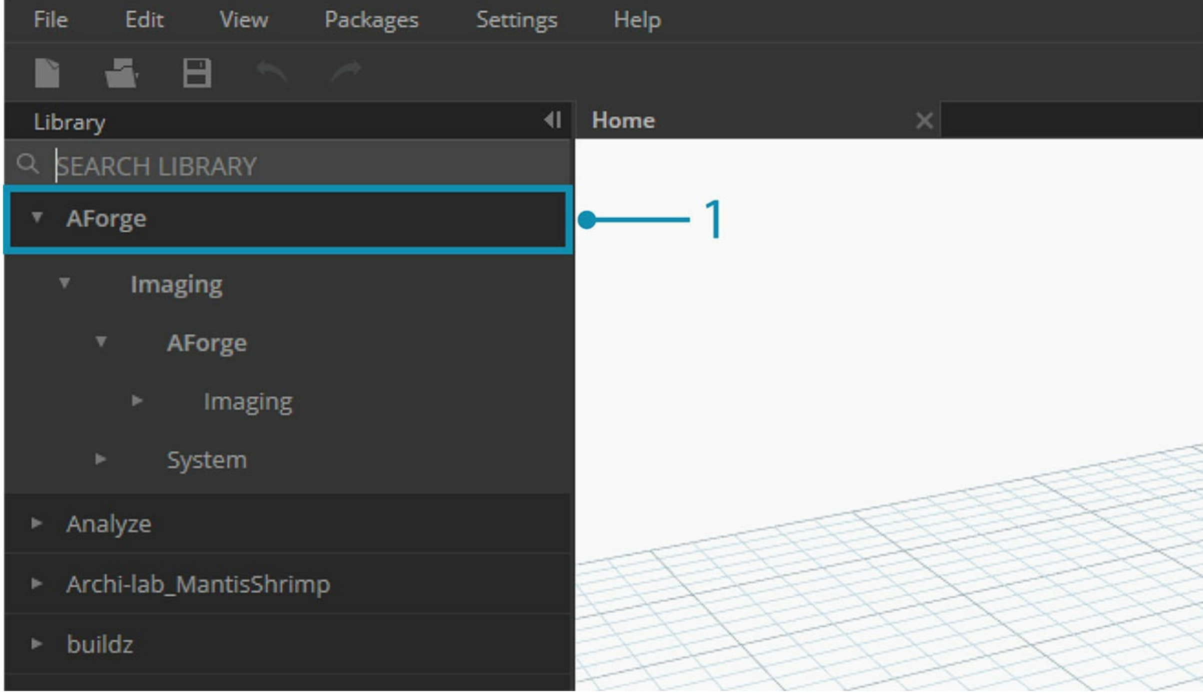


1. Создайте новый файл в Dynamo и выберите *«Файл» > «Импорт библиотеки...»*.



1. В появившемся окне перейдите к подпапке Release в папке установки AForge. Путь к папке, скорее всего, будет выглядеть таким образом: *C:\Program Files (x86)\AForge.NET\Framework\Release*.
2. **AForge.Imaging.dll:** в рамках данного примера нам требуется только этот файл библиотеки AForge. Выберите этот файл *DLL* и нажмите *Открыть*.





1. В Dynamo на панели инструментов «Библиотека» должна появиться группа узлов *AForge*. Теперь библиотека для работы с изображениями AForge доступна непосредственно в приложении для визуального программирования.

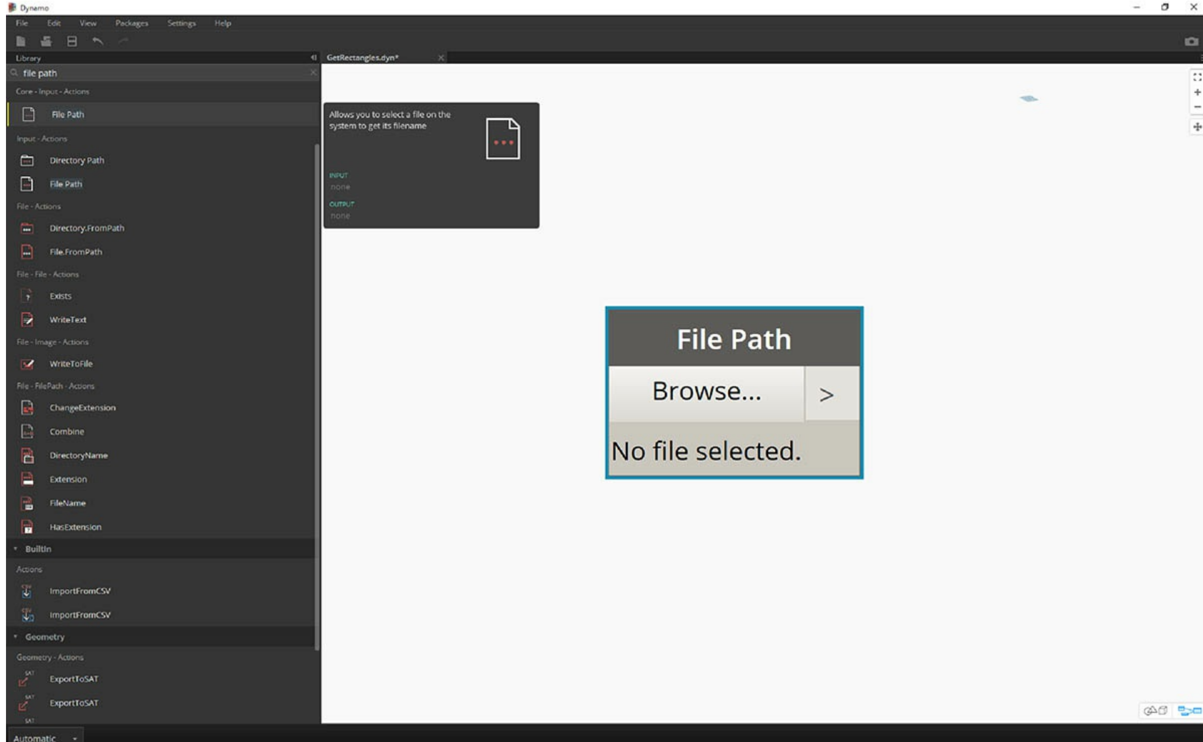
Упражнение 1. Выделение границ

Выполнив импорт библиотеки, можно приступать к первому несложному упражнению. Сначала мы выполним базовую обработку стандартного изображения и посмотрим, как AForge осуществляет фильтрацию изображений. Затем мы воспользуемся узлом *Watch Image* для отображения результатов и применим к изображению фильтры Dynamo, аналогичные фильтрам приложения Photoshop.

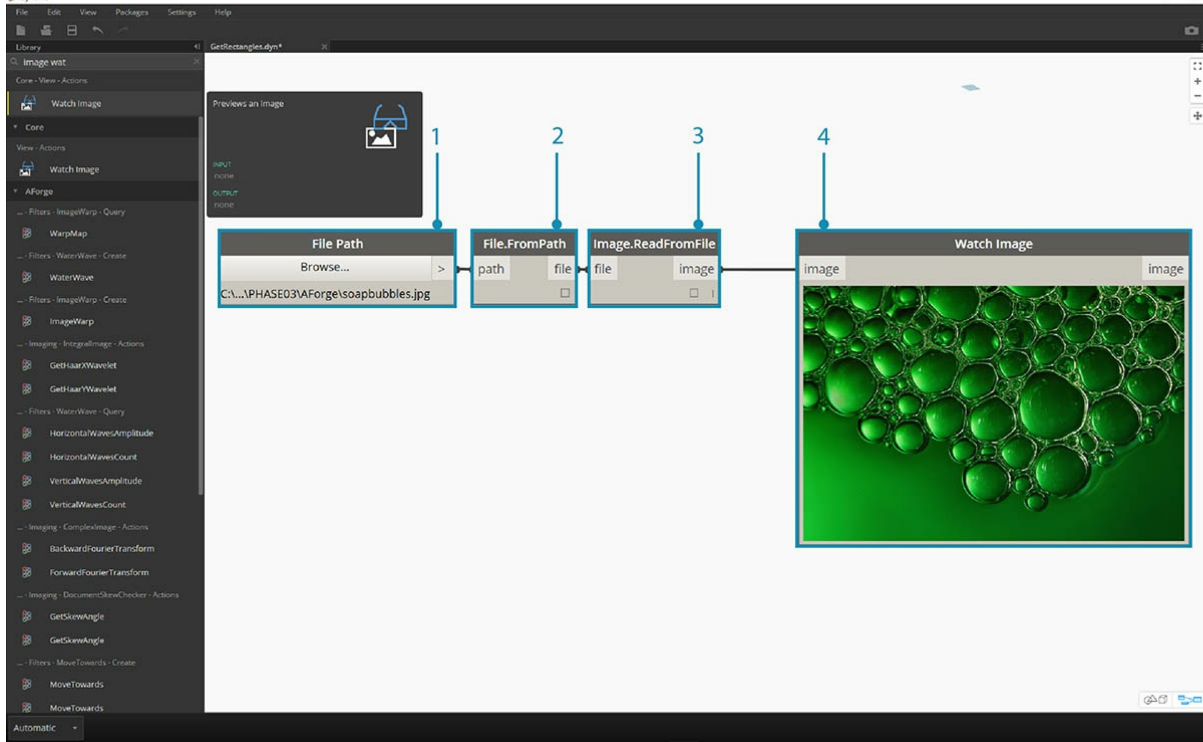
Скачайте и распакуйте файлы примеров, идущие в комплекте с данным пакетом (щелкните правой кнопкой мыши и выберите «Сохранить

ссылку как...»). Полный список файлов примеров можно найти в приложении. [ZeroTouchImages.zip]

Выполнив импорт библиотеки, можно приступать к первому несложному упражнению (*01-EdgeDetection.dyn*). Сначала мы выполним базовую обработку стандартного изображения и посмотрим, как AForge осуществляет фильтрацию изображений. Затем мы воспользуемся узлом *Watch Image* для отображения результатов и применим к изображению фильтры Dynamo, аналогичные фильтрам приложения Photoshop.

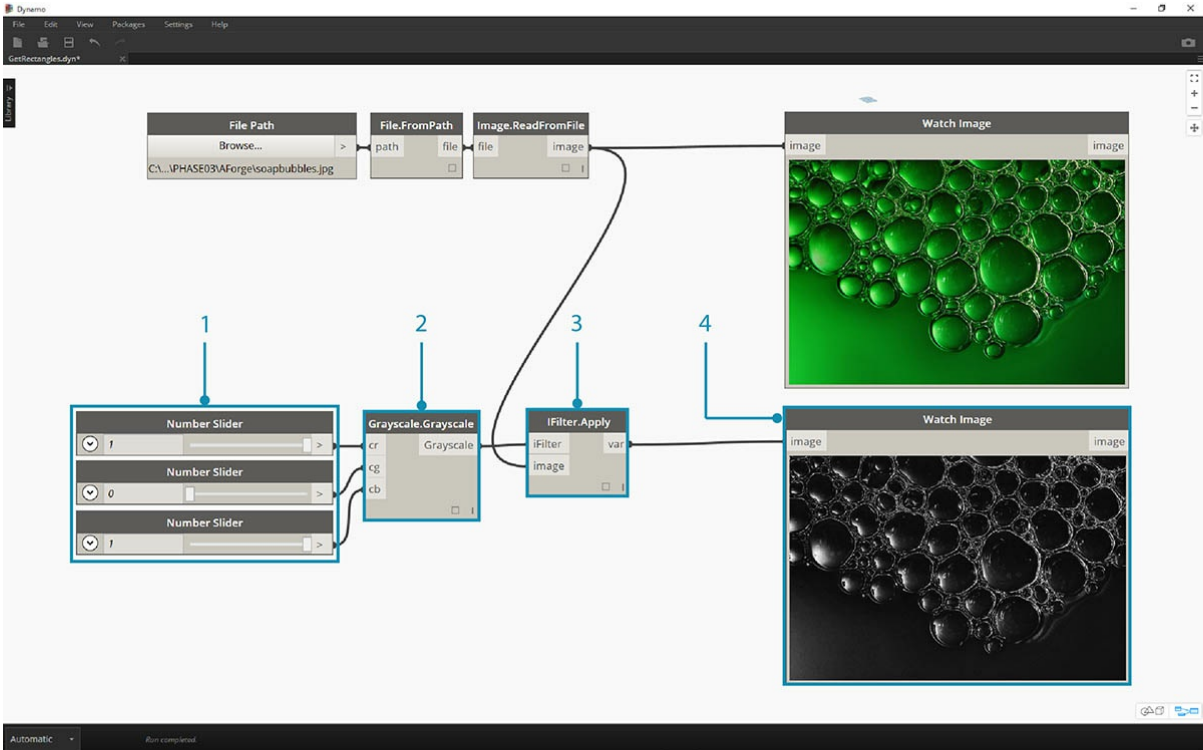


Для начала нужно импортировать изображение, с которым мы будем работать. Добавьте узел *File Path* в рабочую область и выберите файл soapbubbles.jpg в папке загруженных материалов для упражнения (источник изображения: [flickr]).



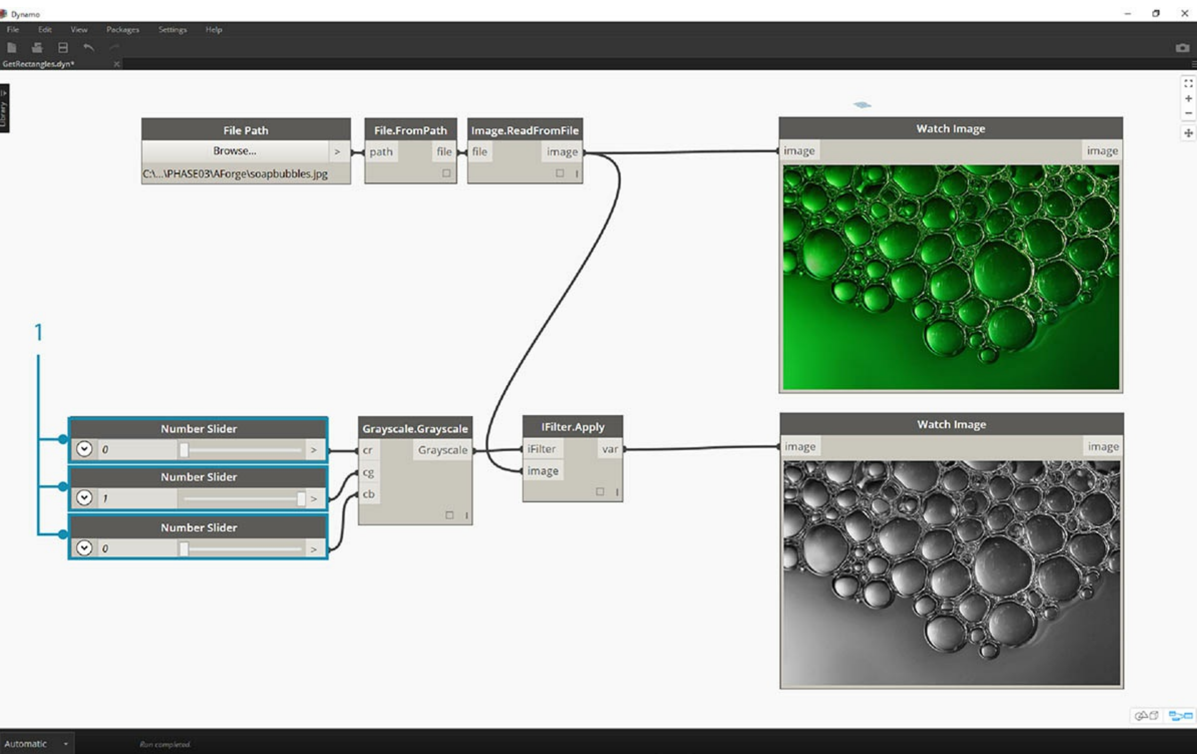
1. Узел File Path предоставляет путь к выбранному изображению в виде строки. Необходимо преобразовать эту строку в изображение в среде Dynamo.
2. Соедините узел File Path с узлом File.FromPath.
3. Чтобы преобразовать файл в изображение, используйте узел Image.ReadFromFile.
4. Наконец, чтобы увидеть результат, перетащите узел Watch Image в рабочую область и соедините его с Image.ReadFromFile. Мы еще

не воспользовались библиотекой AForge, но уже успешно импортировали изображение в Dynamo.



В разделе AForge.Imaging.AForge.Filters (в меню навигации) доступен широкий выбор фильтров. Мы воспользуемся одним из них, чтобы обесцветить изображение в соответствии с пороговыми значениями.

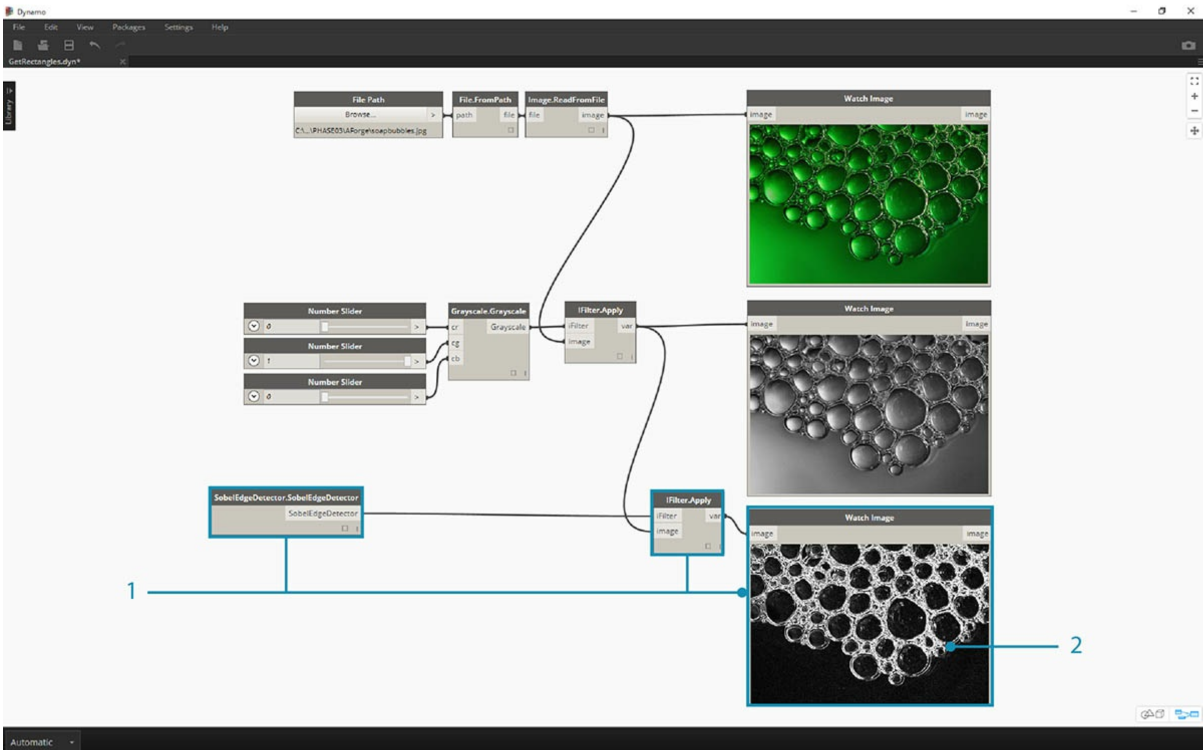
1. Перетащите в рабочую область три регулятора и задайте для них диапазоны от 0 до 1 с шагом 0,01.
2. Добавьте в рабочую область узел Grayscale.Grayscale. Это фильтр AForge, который позволяет применить к изображению оттенки серого. Соедините три регулятора, добавленные в шаге 1, с элементами cr, cg и cb. Задайте для верхнего и нижнего регуляторов значение 1, а для среднего — 0.
3. Чтобы применить оттенки серого, нам нужно задать действие, которое будет выполняться с изображением. Для этого мы используем узел IFilter.Apply. Соедините порт ввода image этого узла с узлом Image, а порт ввода iFilter — с узлом Grayscale.Grayscale.
4. Соедините этот узел с новым узлом Watch Image, и вы получите обесцвеченное изображение.



Путем задания пороговых значений для красного, зеленого и синего цветов можно управлять тем, как именно будет обесцвечиваться

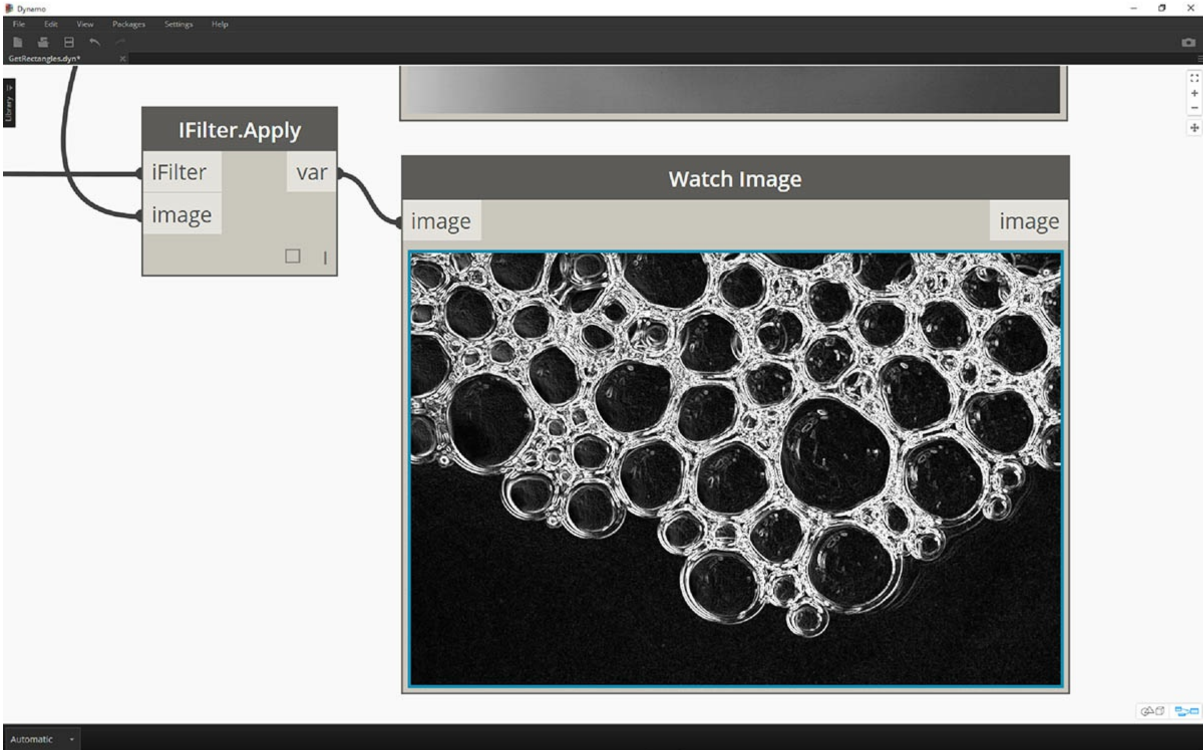
изображение. Пороговые значения задаются через порты ввода узла Grayscale.Grayscale. Обратите внимание, что изображение выглядит довольно тусклым. Это вызвано тем, что для регулятора зеленого цвета задано значение 0.

1. Задайте для верхнего и нижнего регуляторов значение 0, а для среднего — 1. В результате полученное обесцвеченное изображение становится более контрастным.



Теперь применим к полученному обесцвеченному изображению еще один фильтр. Поскольку изображение обладает определенной контрастностью, мы попробуем применить к нему функцию выделения границ.

1. Добавьте узел SobelEdgeDetector.SobelEdgeDetector в рабочую область. Добавьте новый узел IFilter и соедините его порт ввода IFilter с узлом SobelEdgeDetector, а порт ввода image — с узлом IFilter обесцвеченного изображения.
2. В результате мы получаем новое изображение, в котором оператор Собеля выделил все обнаруженные границы.



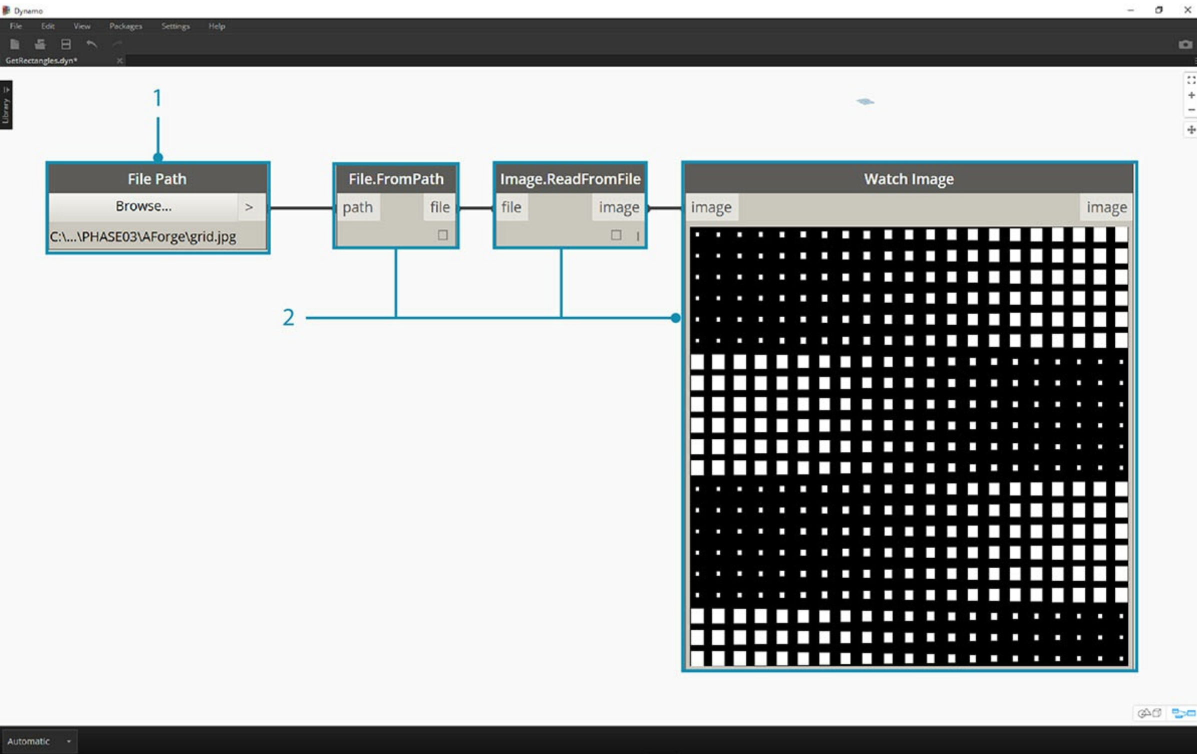
В результате применения инструмента выделения границ мы получили увеличенное изображение пузырьков, границы которых выделены с

помощью пикселей. В библиотеке AForge есть инструменты, которые позволяют использовать подобные результаты для создания геометрии Dynamo. Мы рассмотрим их в следующем упражнении.

Упражнение 2. Создание прямоугольников

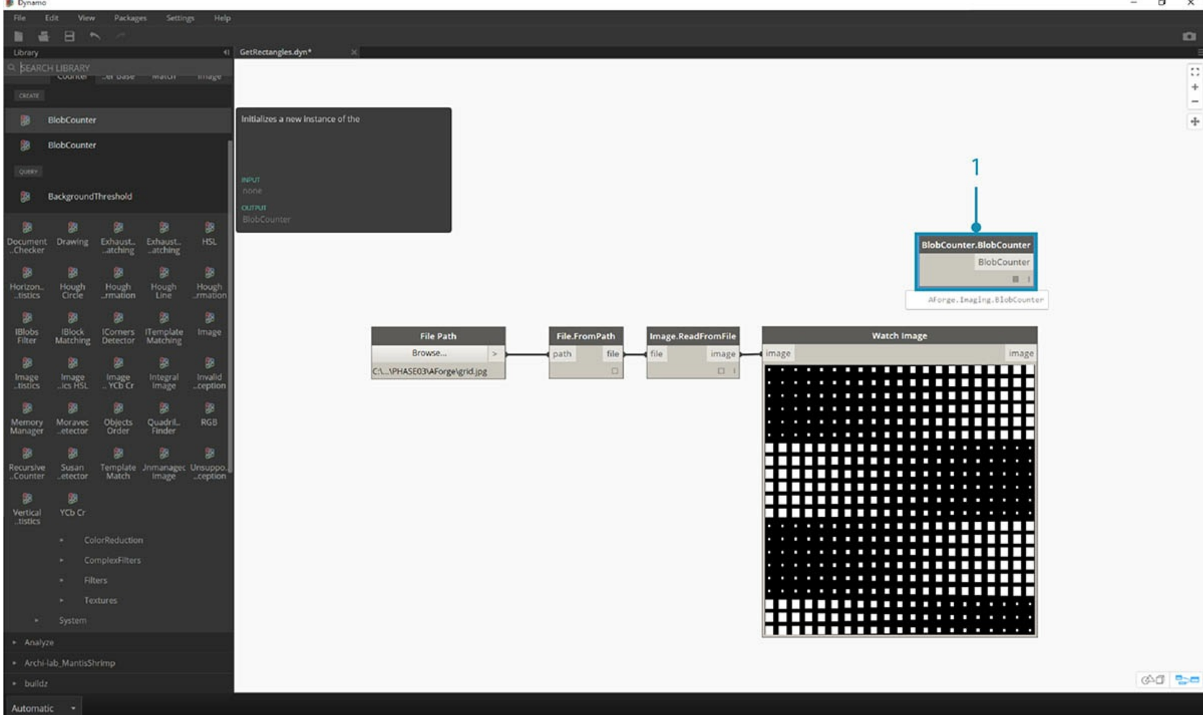
Скачайте и распакуйте файлы примеров, идущие в комплекте с данным пакетом (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. ZeroTouchImages.zip

Теперь, когда мы ознакомились с базовыми возможностями обработки изображений, можно приступить к использованию изображений для создания геометрии Dynamo. Ваша минимальная задача в рамках этого упражнения — выполнить так называемую *быструю трассировку* изображения с помощью AForge и Dynamo. Пока что в целях простоты мы ограничимся извлечением прямоугольников из опорного изображения, однако в AForge доступны инструменты и для более сложных операций. В этом упражнении мы используем файл *02-RectangleCreation.dyn* из загруженного набора материалов для упражнения.

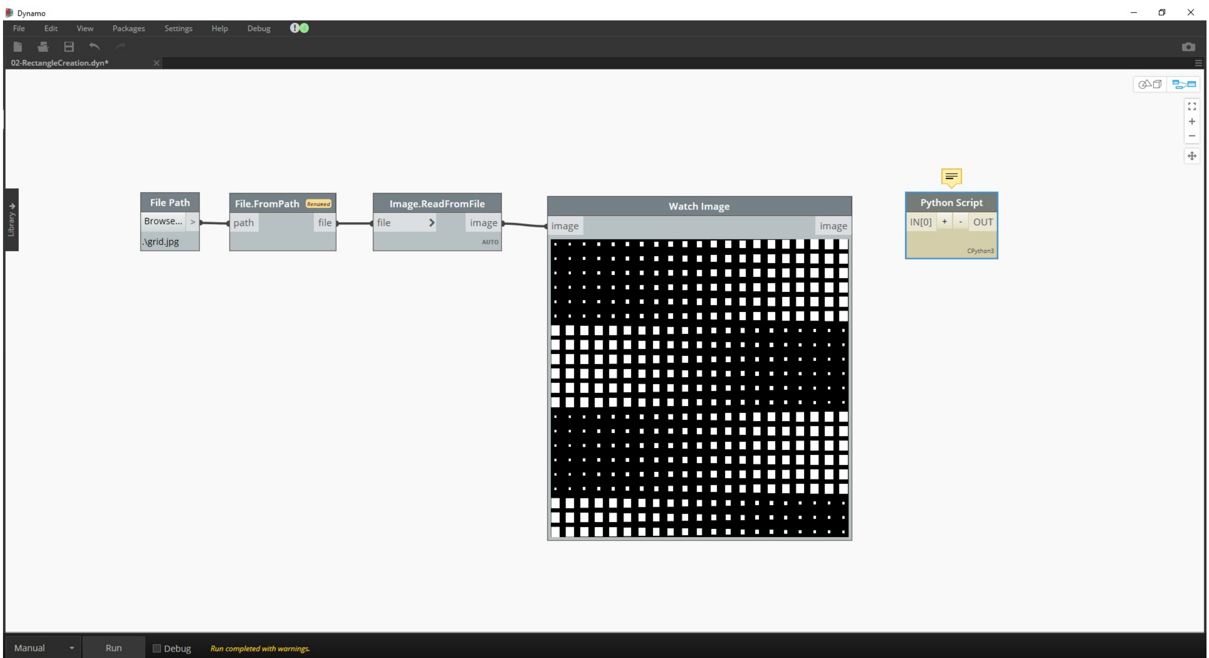


1. С помощью узла File Path задайте путь к файлу grid.jpg в папке материалов для упражнения.
2. Соедините последовательно оставшиеся узлы, как показано выше, чтобы отобразить грубую параметрическую сетку.

На следующем шаге мы зададим белые прямоугольники из этого изображения в качестве опорных объектов и преобразуем их в геометрию Dynamo. Библиотека AForge включает множество мощных инструментов компьютерного распознавания образов. В этом упражнении будет использован один из ключевых инструментов под названием BlobCounter.



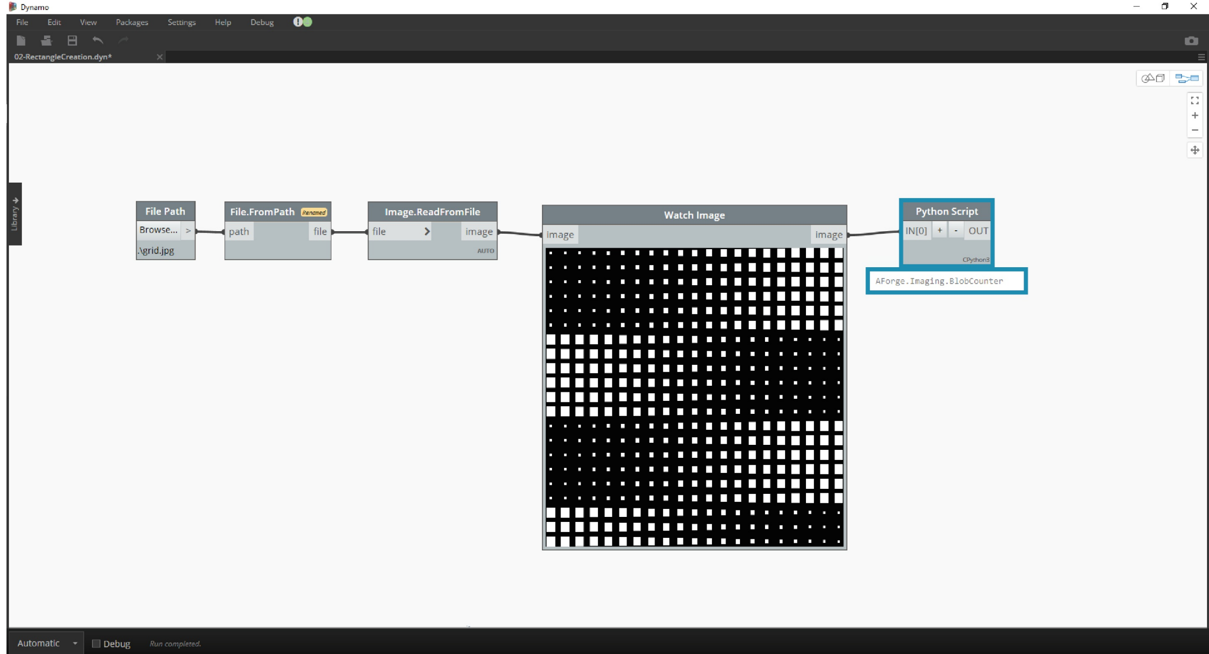
1. После добавления узла BlobCounter в рабочую область нам нужно выполнить обработку изображения (аналогично использованию инструмента IFilter в предыдущем упражнении). К сожалению, найти узел обработки изображений Process Image в библиотеке Dynamo может быть затруднительно. Это связано с тем, что эта функция может быть не видна в исходном коде библиотеки AForge. Чтобы обойти эту проблему, потребуется временное решение.



1. Добавьте узел Python в рабочую область.

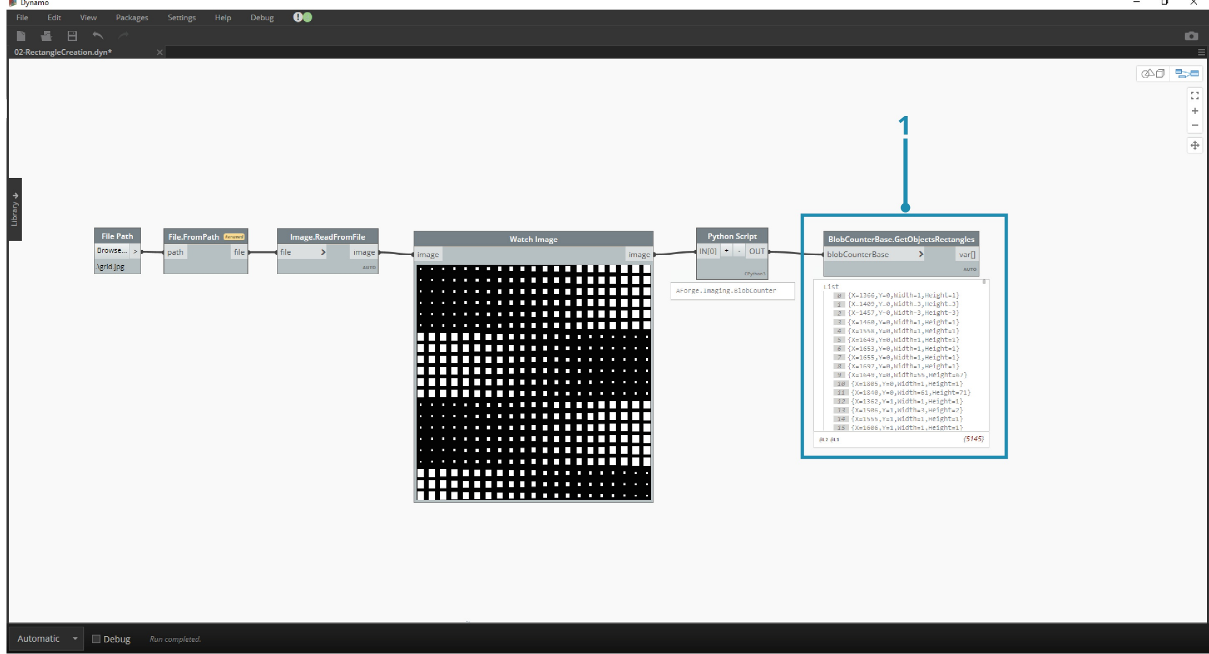
```
import clr
clr.AddReference('AForge.Imaging')
from AForge.Imaging import *

bc= BlobCounter()
bc.ProcessImage(IN[0])
OUT=bc
```

Добавьте в узел Python приведенный выше код. Этот код позволяет импортировать библиотеку AForge, а затем обработать импортированное изображение.

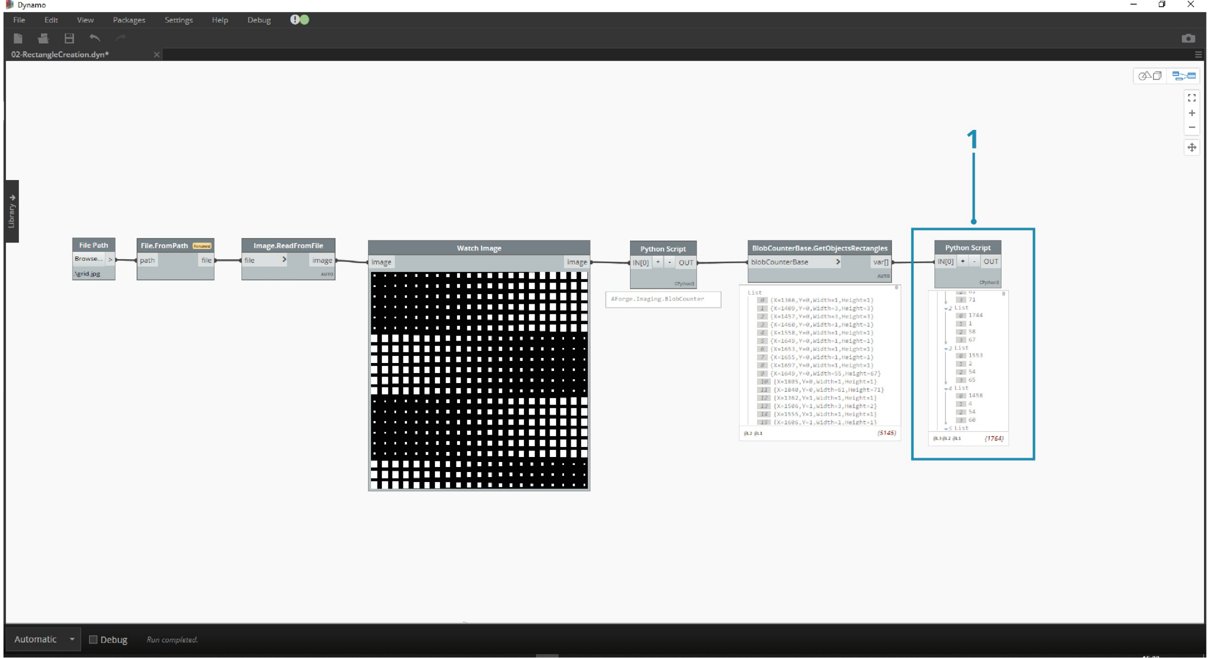


При соединении порта вывода image с портом ввода узла Python последний выдает результат AForge.Imaging.BlobCounter.

Следующие шаги включают в себя операции, требующие определенного опыта работы с API-интерфейсом обработки изображений AForge. Это не значит, что для работы с Dynamo обязательно нужно обладать этими знаниями. Мы сделали это в целях демонстрации гибких возможностей работы с внешними библиотеками в среде Dynamo.

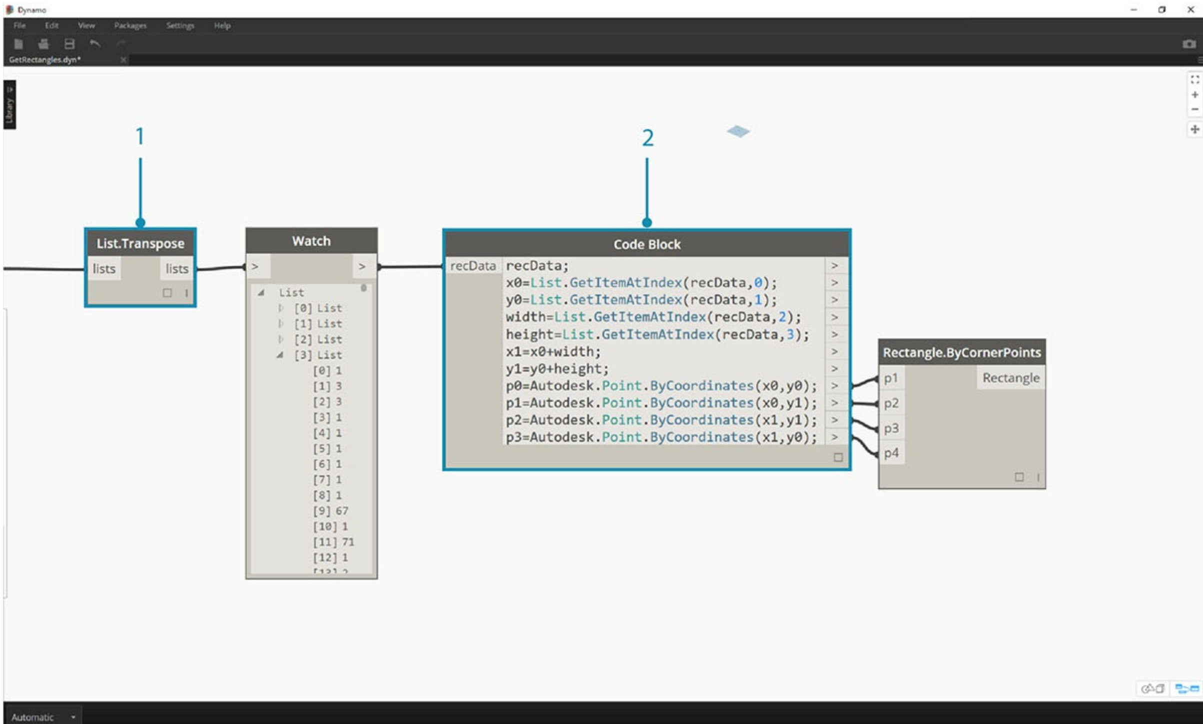


1. Соедините порт вывода узла сценария Python с узлом BlobCounterBase.GetObjectRectangles. При этом объекты изображения считываются в соответствии с заданным пороговым значением, после чего квантифицированные прямоугольники извлекаются из пиксельного пространства.



1. Добавьте еще один узел Python в рабочую область, соедините его с узлом GetObjectRectangles и введите в него код, указанный ниже. В результате создается упорядоченный список объектов Dynamo.

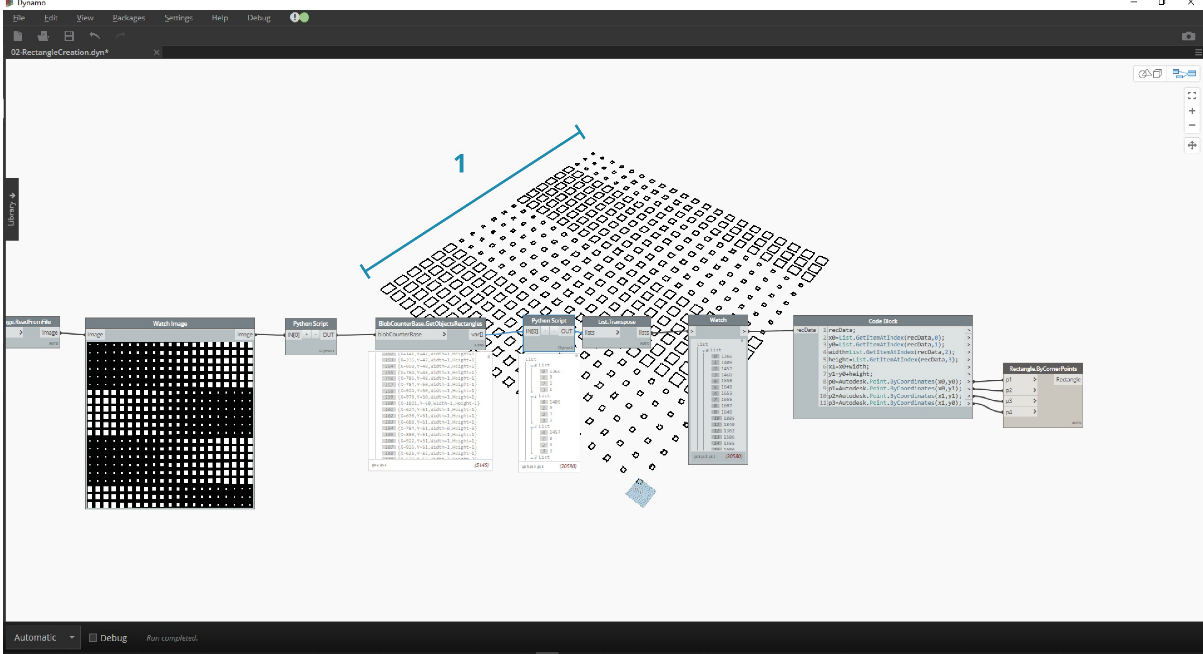
```
OUT = []
for rec in IN[0]:
subOUT=[]
subOUT.append(rec.X)
subOUT.append(rec.Y)
subOUT.append(rec.Width)
subOUT.append(rec.Height)
OUT.append(subOUT)
```



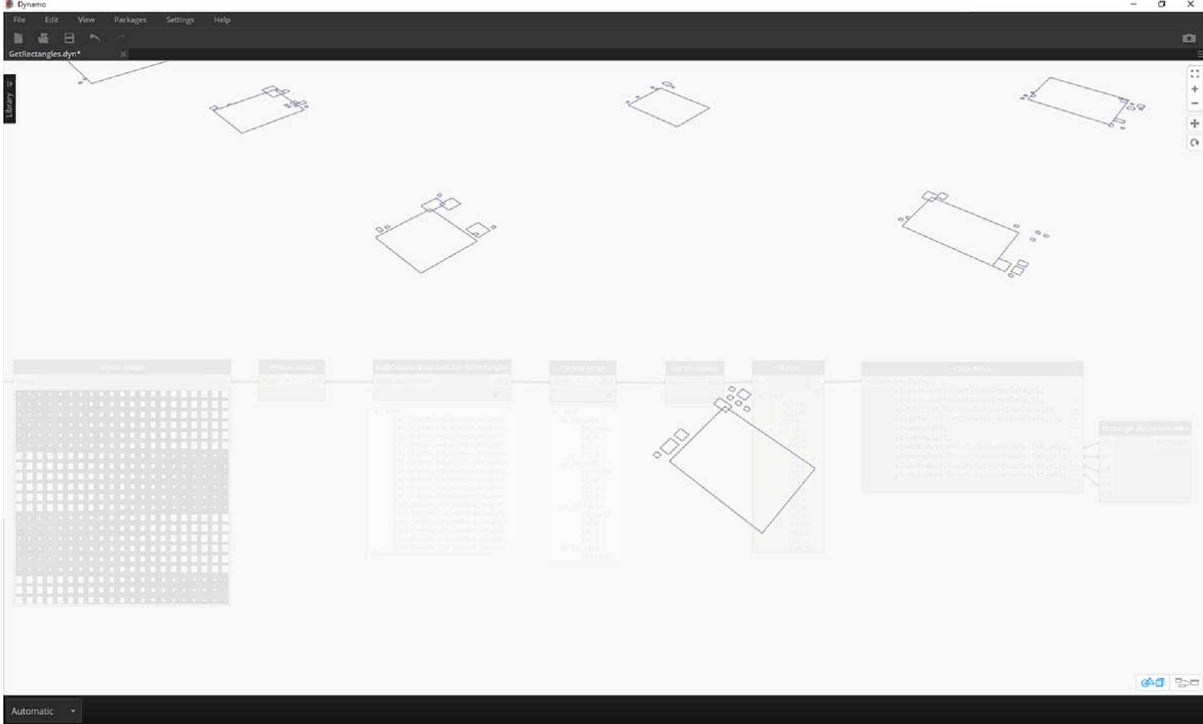
1. Добавьте узел Transpose к порту вывода узла Python из предыдущего шага. Создаются четыре списка, содержащие значения координат X и Y, а также ширины и высоты для каждого прямоугольника.
2. С помощью узла Code Block упорядочим данные таким образом, чтобы их можно было использовать в узле Rectangle.ByCornerPoints (см. код ниже).

```
recData;
x0=List.GetItemAtIndex(recData,0);
y0=List.GetItemAtIndex(recData,1);
```

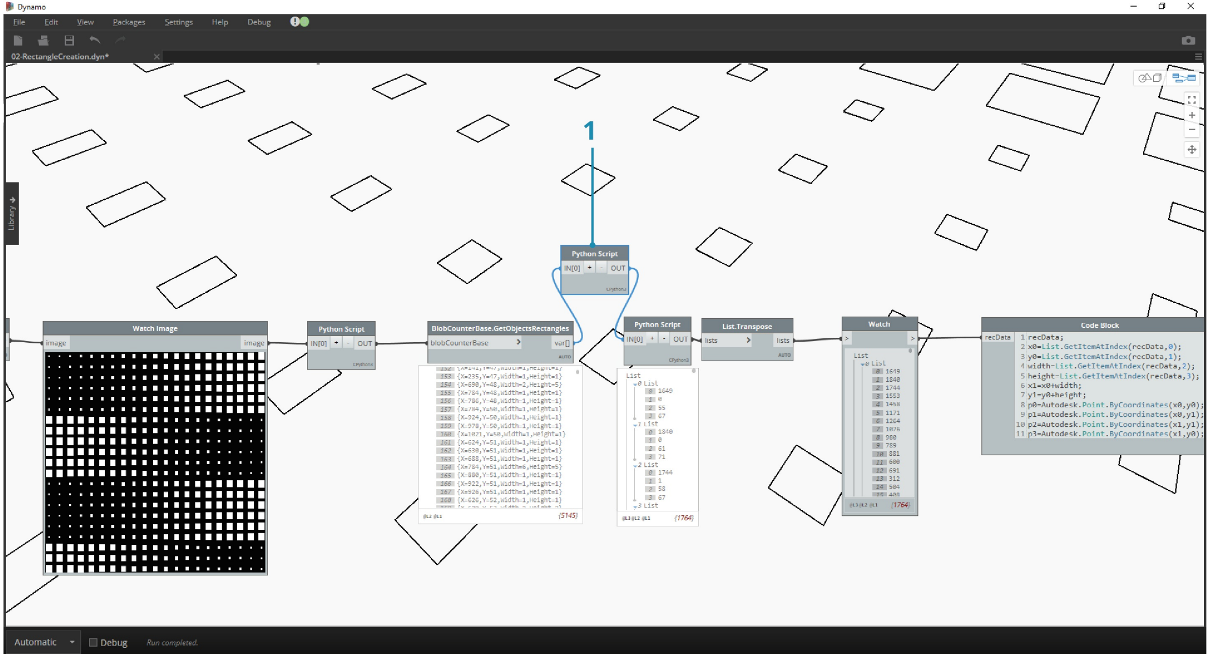
```
width=List.GetItemAtIndex(recData,2);
height=List.GetItemAtIndex(recData,3);
x1=x0+width;
y1=y0+height;
p0=Autodesk.Point.ByCoordinates(x0,y0);
p1=Autodesk.Point.ByCoordinates(x0,y1);
p2=Autodesk.Point.ByCoordinates(x1,y1);
p3=Autodesk.Point.ByCoordinates(x1,y0);
```



Уменьшив масштаб отображения, вы увидите массив прямоугольников, представляющих белые квадраты на изображении. Таким образом, с помощью программирования мы смогли выполнить операцию, аналогичную быстрой трассировке в программе Adobe Illustrator.

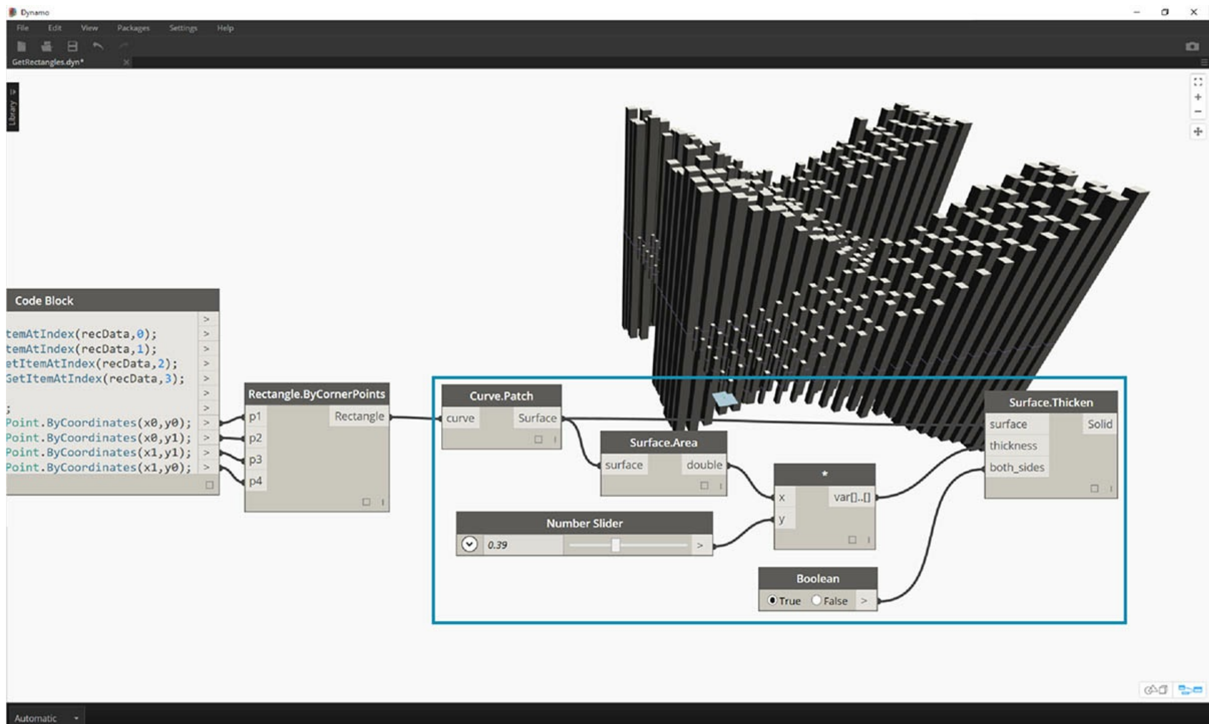


Теперь нам нужно удалить из изображения все лишнее. Увеличив масштаб, вы увидите маленькие прямоугольники, которые требуется удалить.

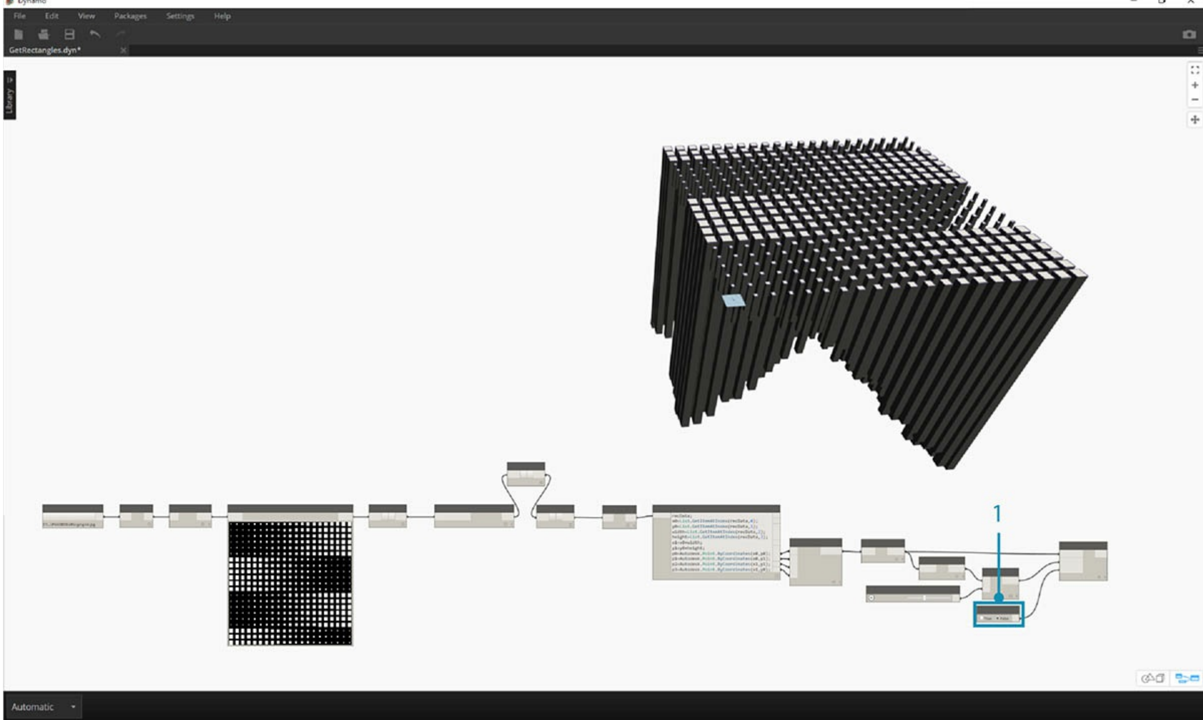


1. Чтобы удалить лишние прямоугольники, вставьте новый узел Python между узлом GetObjectRectangles и узлом Python, следующим за ним. Приведенный ниже код для этого узла позволяет удалить все прямоугольники, размер которых меньше заданного значения.

```
rectangles=IN[0]
OUT=[]
for rec in rectangles:
if rec.Width>8 and rec.Height>8:
OUT.append(rec)
```



Удалив лишние прямоугольники, мы можем поэкспериментировать и создать поверхность из прямоугольников, а затем выдавить их на глубину, соответствующую их площади.



1. Наконец, измените значение both_sides на false, чтобы получить выдавливание в одном направлении. Если залить то, что у нас здесь получилось, эпоксидной смолой, то у нас был бы модный столик в стиле хай-тек.

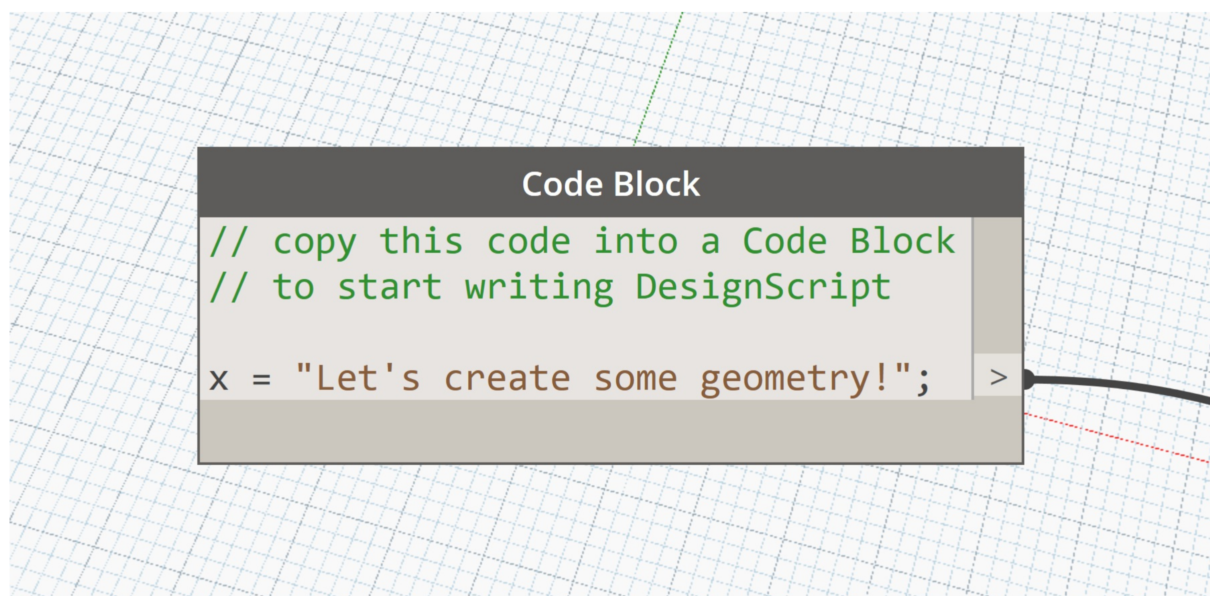
Мы выполнили несколько простых упражнений, однако процедуры, которые здесь рассматривались, можно использовать гораздо более интересным образом для самых разных целей. Возможности компьютерного распознавания образов применимы в широчайшем спектре процессов, таких как сканирование штрихкодов, подгонка перспективы, наложение данных проекции, дополненная реальность и многое другое. Дополнительные темы по работе с библиотекой AForge, связанные с этим упражнением, см. в данной статье.

Создание геометрии с помощью DesignScript

Создание геометрии с помощью DesignScript

В этом разделе представлены упражнения по созданию геометрии с помощью DesignScript. Для их выполнения следует просто скопировать примеры кода DesignScript в узлы Code Block в Dynamo.

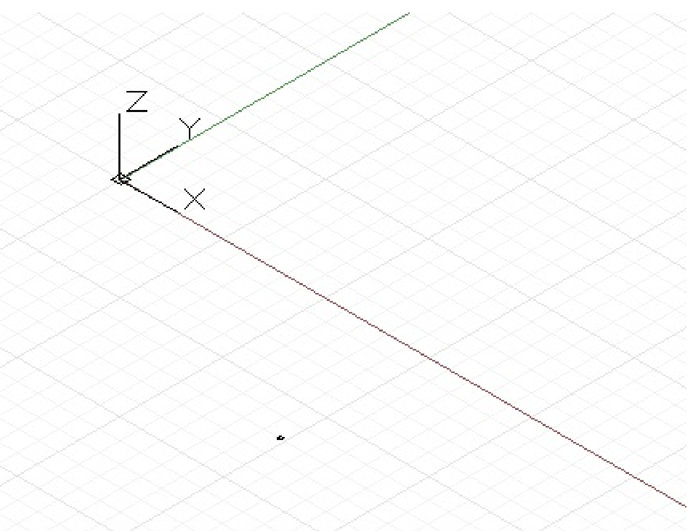
```
// copy this code into a Code Block
// to start writing DesignScript

x = "Let's create some geometry!";
```



Основы работы с геометрией посредством DesignScript

Основы работы с геометрией посредством DesignScript

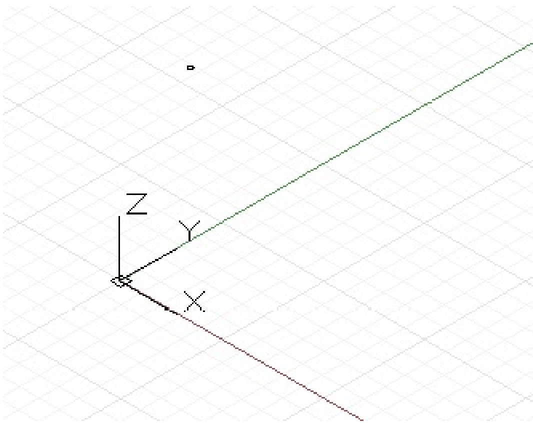
Простейший геометрический объект, доступный в библиотеке стандартных геометрических объектов Dynamo, — это точка. Вся геометрия создается с помощью специальных функций (конструкторов), каждая из которых возвращает новый экземпляр геометрического объекта определенного типа. В Dynamo название каждого конструктора начинается с наименования типа объекта (в данном случае Point, точка), за которым следует метод построения объекта. Чтобы построить трехмерную точку на основе значений X, Y и Z в прямоугольной системе координат, используйте конструктор *ByCoordinates*:




```
// create a point with the following x, y, and z
// coordinates:
x = 10;
y = 2.5;
z = -6;

p = Point.ByCoordinates(x, y, z);
```

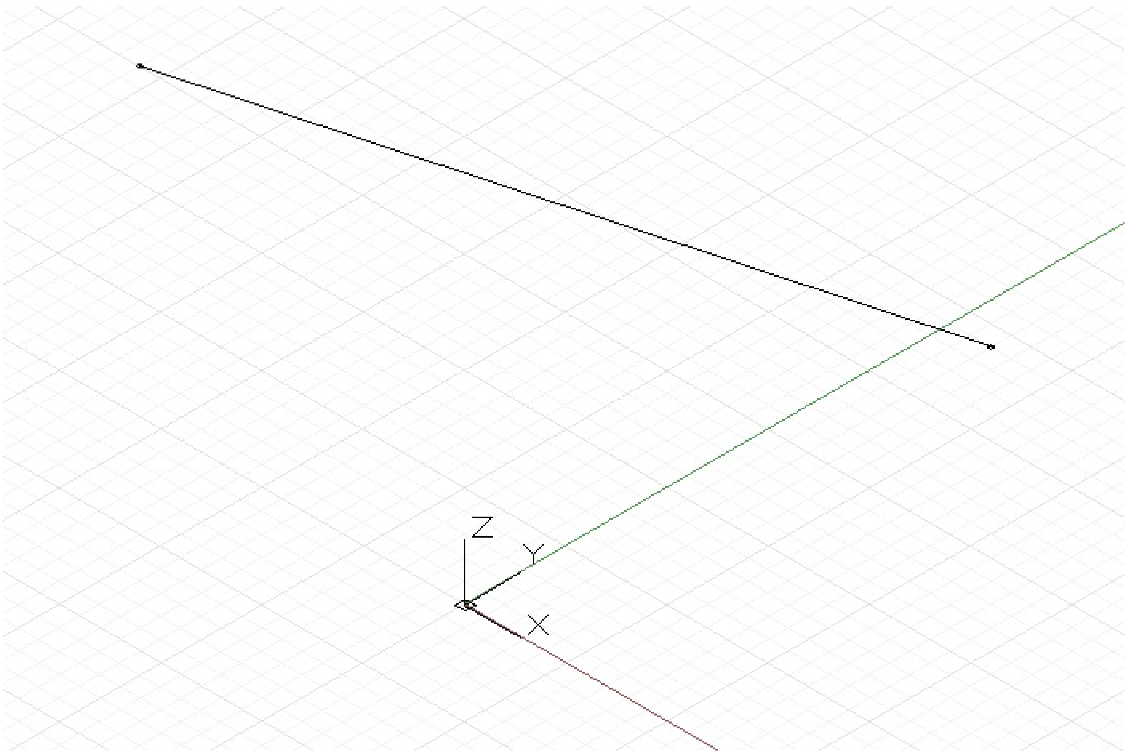
Конструкторы в Dynamo обычно обозначаются приставкой *By*, и при вызове эти функции возвращают новый объект соответствующего типа. Этот объект сохраняется в переменной, имя которой указано слева от знака равенства.

Для создания большинства объектов можно использовать множество различных конструкторов. Используя конструктор *BySphericalCoordinates*, можно задать точку на сфере, определяемой по радиусу, а также первому и второму углам поворота (заданным в градусах):



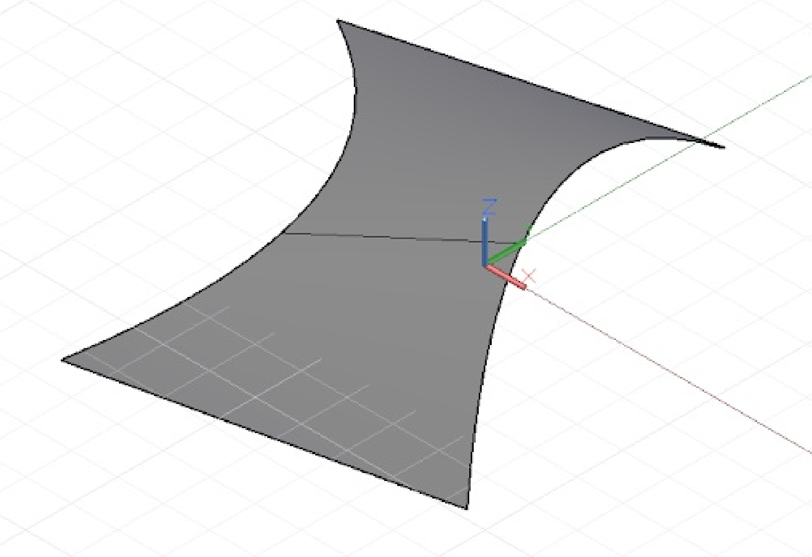

```
// create a point on a sphere with the following radius,
// theta, and phi rotation angles (specified in degrees)
radius = 5;
theta = 75.5;
phi = 120.3;
cs = CoordinateSystem.Identity();

p = Point.BySphericalCoordinates(cs, radius, theta,
    phi);
```

Используя точки, можно создавать геометрические объекты более высокого уровня, например отрезки. С помощью конструктора *ByStartPointEndPoint* создайте объект Line (отрезок) между двумя точками:




```
// create two points:
p1 = Point.ByCoordinates(3, 10, 2);
p2 = Point.ByCoordinates(-15, 7, 0.5);

// construct a line between p1 and p2
l = Line.ByStartPointEndPoint(p1, p2);
```

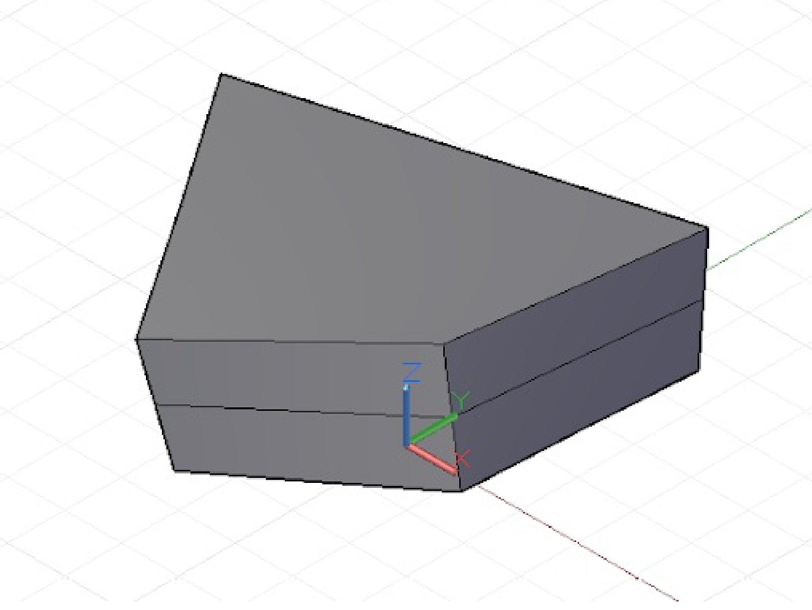
Аналогичным образом из отрезков можно создавать геометрические объекты следующего уровня — поверхности. Для этого можно использовать, например, конструктор *Loft*, который выполняет интерполяцию поверхности между заданными отрезками или кривыми.



```
// create points:
p1 = Point.ByCoordinates(3, 10, 2);
p2 = Point.ByCoordinates(-15, 7, 0.5);

p3 = Point.ByCoordinates(5, -3, 5);
p4 = Point.ByCoordinates(-5, -6, 2);

p5 = Point.ByCoordinates(9, -10, -2);
p6 = Point.ByCoordinates(-11, -12, -4);

// create lines:
```

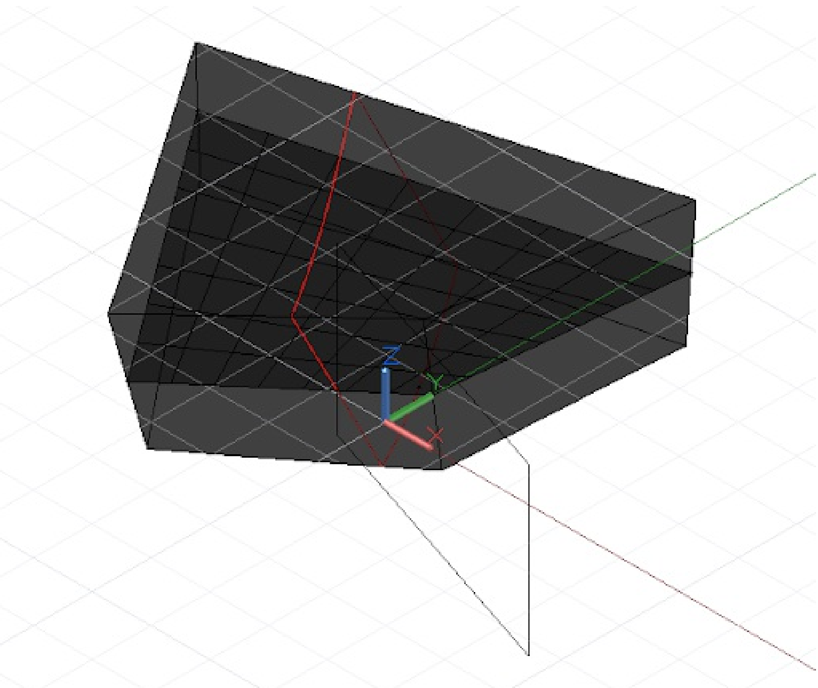
```
l1 = Line.ByStartPointEndPoint(p1, p2);
l2 = Line.ByStartPointEndPoint(p3, p4);
l3 = Line.ByStartPointEndPoint(p5, p6);

// loft between cross section lines:
surf = Surface.ByLoft([l1, l2, l3]);
```

Поверхности также можно использовать для создания геометрии более высокого уровня, а именно тел. Сделать это можно, например, путем увеличения толщины поверхности на заданную величину. Многим объектам изначально назначены определенные функции, называемые методами, которые позволяют программистам выполнять действия с определенным объектом. К методам, доступным для всех геометрических объектов, относятся операции *Translate* и *Rotate*, которые позволяют переносить и поворачивать геометрию в соответствии с заданной величиной переноса или поворота. Для поверхностей доступен метод *Thicken*, для использования которого требуется одно входное значение — число, определяющее новую толщину поверхности.



```
p1 = Point.ByCoordinates(3, 10, 2);
p2 = Point.ByCoordinates(-15, 7, 0.5);

p3 = Point.ByCoordinates(5, -3, 5);
p4 = Point.ByCoordinates(-5, -6, 2);

l1 = Line.ByStartPointEndPoint(p1, p2);
l2 = Line.ByStartPointEndPoint(p3, p4);

surf = Surface.ByLoft([l1, l2]);

// true indicates to thicken both sides of the Surface:
solid = surf.Thicken(4.75, true);
```

Команды *Intersect* позволяют извлекать из геометрии высокого уровня более простые геометрические объекты. Такие объекты могут затем послужить основой для создания другой сложной геометрии. В результате получается циклический процесс создания, извлечения и повторного построения геометрии. В этом примере объект поверхности Surface, извлеченный из объекта тела Solid, используется для создания объекта кривой Curve.




```
p1 = Point.ByCoordinates(3, 10, 2);
p2 = Point.ByCoordinates(-15, 7, 0.5);

p3 = Point.ByCoordinates(5, -3, 5);
p4 = Point.ByCoordinates(-5, -6, 2);

l1 = Line.ByStartPointEndPoint(p1, p2);
l2 = Line.ByStartPointEndPoint(p3, p4);

surf = Surface.ByLoft([l1, l2]);

solid = surf.Thicken(4.75, true);

p = Plane.ByOriginNormal(Point.ByCoordinates(2, 0, 0),
    Vector.ByCoordinates(1, 1, 1));

int_surf = solid.Intersect(p);

int_line = int_surf.Intersect(Plane.ByOriginNormal(
    Point.ByCoordinates(0, 0, 0),
    Vector.ByCoordinates(1, 0, 0)));
```

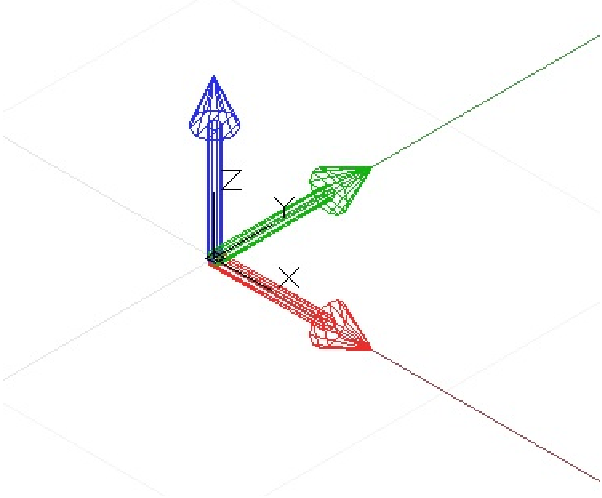
Геометрические примитивы

Геометрические примитивы

Несмотря на то что в Dynamo можно создавать разнообразные сложные геометрические формы, основу машинного проектирования составляют простые геометрические примитивы. Они либо задают итоговую форму спроектированной конструкции, либо играют роль каркаса, на котором достраивается более сложная геометрия.

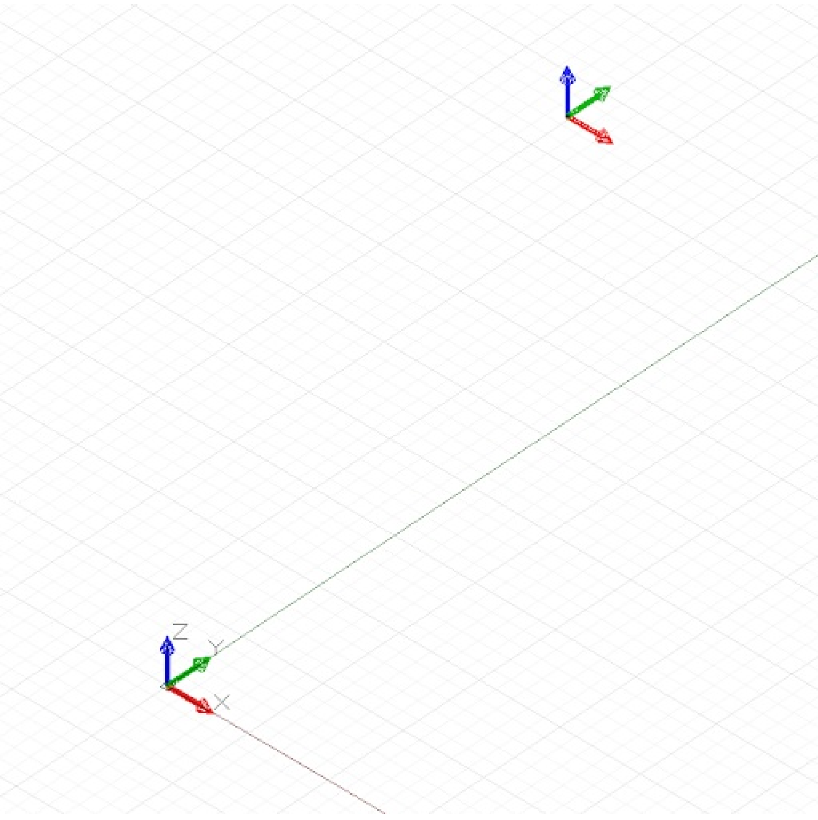
Объект CoordinateSystem не является геометрическим объектом в строгом смысле, однако он играет важную роль при построении геометрии. Объект CoordinateSystem позволяет отслеживать как положение, так и геометрические преобразования, такие как поворот, сдвиг и масштабирование.

Для создания объекта CoordinateSystem, центрованного по точке с координатами x = 0, y = 0, z = 0 без поворота, масштабирования или сдвига, достаточно вызвать конструктор Identity:




```
// create a CoordinateSystem at x = 0, y = 0, z = 0,
// no rotations, scaling, or sheering transformations

cs = CoordinateSystem.Identity();
```

Объекты CoordinateSystem, к которым были применены геометрические преобразования, в этой главе не рассматриваются, однако стоит упомянуть один конструктор, который позволяет создать систему координат в определенной точке, — *CoordinateSystem.ByOriginVectors*:



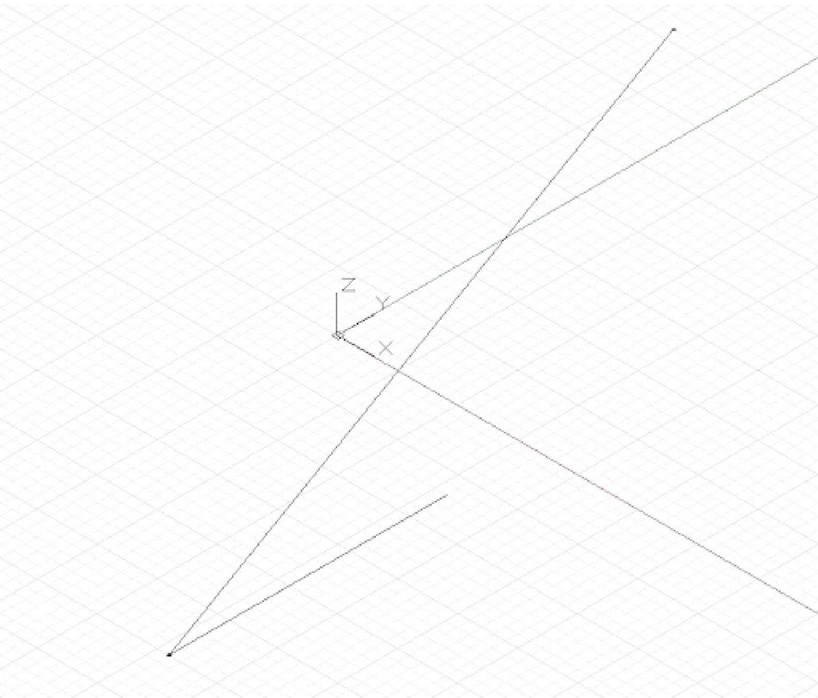

```
// create a CoordinateSystem at a specific location,
// no rotations, scaling, or sheering transformations
x_pos = 3.6;
y_pos = 9.4;
z_pos = 13.0;

origin = Point.ByCoordinates(x_pos, y_pos, z_pos);
identity = CoordinateSystem.Identity();

cs = CoordinateSystem.ByOriginVectors(origin,
    identity.XAxis, identity.YAxis, identity.ZAxis);
```

Простейшим геометрическим примитивом является Point (точка), обозначающий расположение, у которого отсутствуют измерения, в трехмерном пространстве. Как уже упоминалось ранее, создать точку в определенной системе координат можно несколькими способами: с помощью *Point.ByCoordinates* по заданным координатам X, Y и Z; с помощью *Point.ByCartesianCoordinates* по заданным координатам X, Y и Z в определенной системе координат; с помощью *Point.ByCylindricalCoordinates* на цилиндре, заданном по радиусу, углу поворота и высоте; и, наконец, с помощью *Point.BySphericalCoordinates* на сфере, заданной по радиусу и двум углам поворота.

В этом примере показаны точки, созданные в разных системах координат:

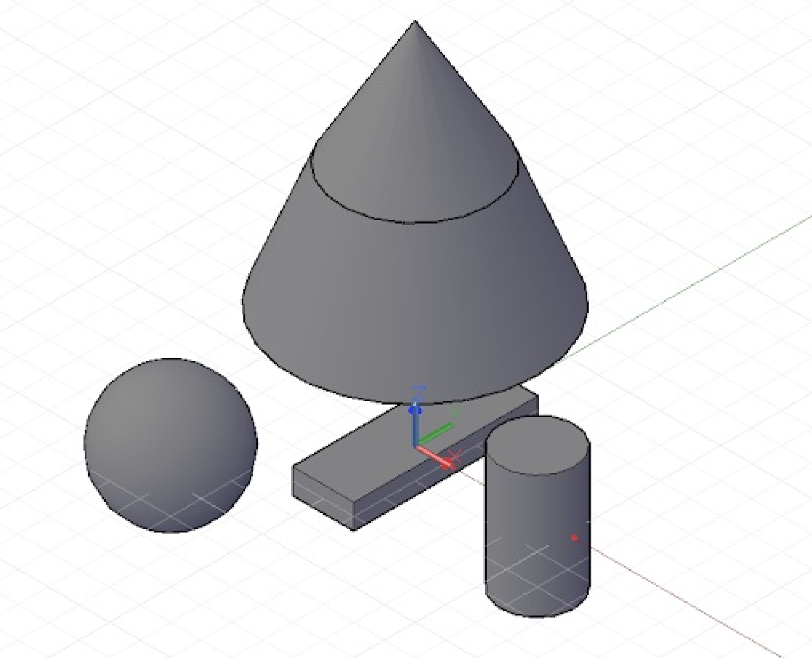
```
// create a point with x, y, and z coordinates
x_pos = 1;
y_pos = 2;
z_pos = 3;

pCoord = Point.ByCoordinates(x_pos, y_pos, z_pos);

// create a point in a specific coordinate system
cs = CoordinateSystem.Identity();
pCoordSystem = Point.ByCartesianCoordinates(cs, x_pos,
    y_pos, z_pos);

// create a point on a cylinder with the following
// radius and height
radius = 5;
height = 15;
theta = 75.5;

pCyl = Point.ByCylindricalCoordinates(cs, radius, theta,
    height);

// create a point on a sphere with radius and two angles

phi = 120.3;

pSphere = Point.BySphericalCoordinates(cs, radius,
    theta, phi);
```

Следующим по сложности примитивом Dynamo является отрезок, который представляет собой бесконечное количество точек, лежащее между двумя конечными точками. Чтобы создать отрезок, можно либо явным образом задать две граничные точки с помощью конструктора *Line.ByStartPointEndPoint*, либо задать начальную точку, направление и длину с помощью конструктора *Line.ByStartPointDirectionLength*.




```
p1 = Point.ByCoordinates(-2, -5, -10);
p2 = Point.ByCoordinates(6, 8, 10);

// a line segment between two points
l2pts = Line.ByStartPointEndPoint(p1, p2);

// a line segment at p1 in direction 1, 1, 1 with
// length 10
lDir = Line.ByStartPointDirectionLength(p1,
    Vector.ByCoordinates(1, 1, 1), 10);
```

В Dynamo доступны объекты, представляющие базовые типы геометрических трехмерных примитивов: кубоиды, для создания которых используется *Cuboid.ByLength*; конусы, создаваемые с помощью *Cone.ByPointsRadius* и *Cone.ByPointsRadii*; цилиндры, получаемые при помощи *Cylinder.ByRadiusHeight*; и сферы, создаваемые с помощью *Sphere.ByCenterPointRadius*.



```
// create a cuboid with specified lengths
cs = CoordinateSystem.Identity();

cub = Cuboid.ByLengths(cs, 5, 15, 2);
```

```
// create several cones
p1 = Point.ByCoordinates(0, 0, 10);
p2 = Point.ByCoordinates(0, 0, 20);
p3 = Point.ByCoordinates(0, 0, 30);

cone1 = Cone.ByPointsRadii(p1, p2, 10, 6);
cone2 = Cone.ByPointsRadii(p2, p3, 6, 0);

// make a cylinder
cylCS = cs.Translate(10, 0, 0);

cyl = Cylinder.ByRadiusHeight(cylCS, 3, 10);

// make a sphere
centerP = Point.ByCoordinates(-10, -10, 0);

sph = Sphere.ByCenterPointRadius(centerP, 5);
```

Векторная математика

Векторная математика

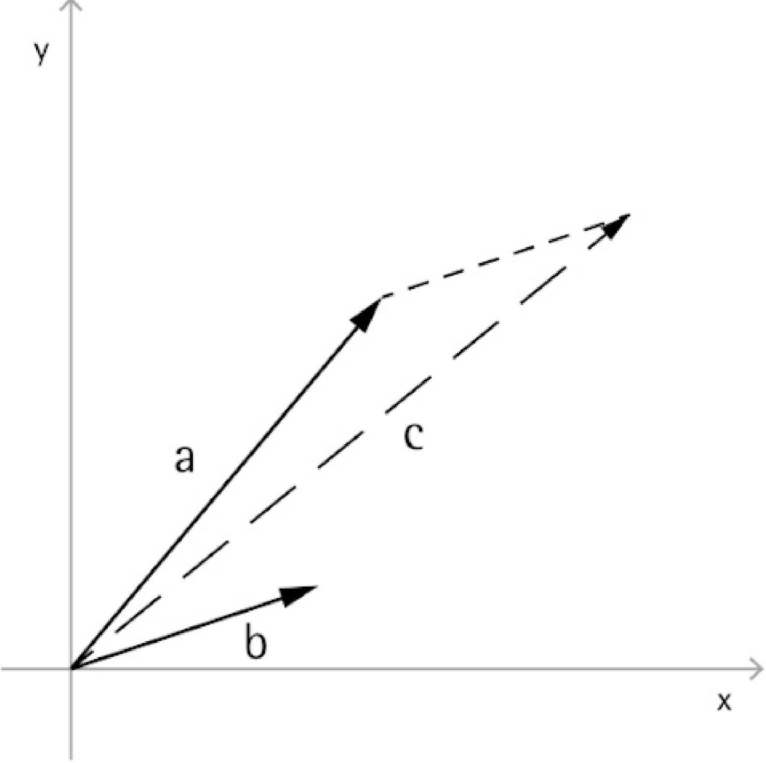
В машинном проектировании редко бывает так, что объекты создаются сразу в своей конечной форме и конечном положении. Чаще всего созданные объекты приходится переносить, поворачивать и иным образом позиционировать относительно существующей геометрии. Векторная математика играет роль своего рода геометрического каркаса, определяющего направление и ориентацию геометрии, а также позволяющего осмыслять перемещения геометрии по 3D-пространству без их визуального представления.

Если отбросить все лишнее, можно сказать, что вектор представляет собой то или иное положение в 3D-пространстве. Под вектором чаще всего подразумевают конечную точку стрелки, исходящей из начала координат (0, 0, 0) и указывающей на это местоположение. Векторы создаются с помощью конструктора *ByCoordinates*, в котором используются координаты X, Y и Z для нового объекта Vector. Обратите внимание, что объекты Vector не являются геометрическими объектами и не отображаются в окне Dynamo. При этом информацию о созданном или измененном векторе можно вывести на печать в окне консоли:

```
// construct a Vector object
v = Vector.ByCoordinates(1, 2, 3);

s = v.X + " " + v.Y + " " + v.Z;
```

Объекты Vector поддерживают определенный набор математических операций, которые позволяют складывать, вычитать, умножать и иным образом преобразовывать объекты в 3D-пространстве точно так же, как и при работе с вещественными числами в одномерном пространстве числовой оси.

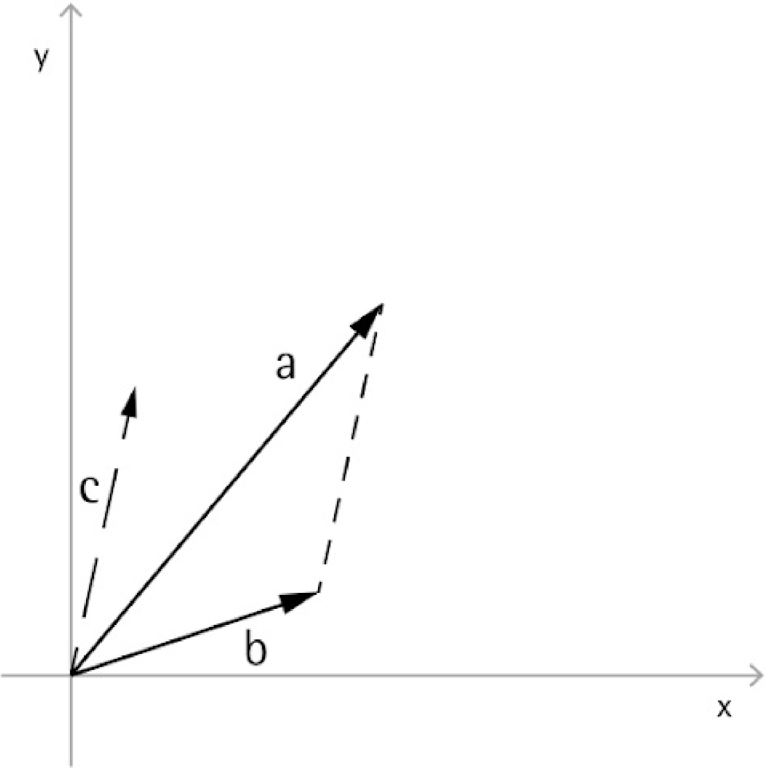
Операция сложения объектов Vector позволяет получить сумму компонентов двух векторов. Визуально представить себе результат сложения векторов можно, поместив начало второго вектора в конечную точку первого. Операция сложения объектов Vector выполняется с использованием метода *Add* и представлена на схеме слева.




```
a = Vector.ByCoordinates(5, 5, 0);
b = Vector.ByCoordinates(4, 1, 0);
```

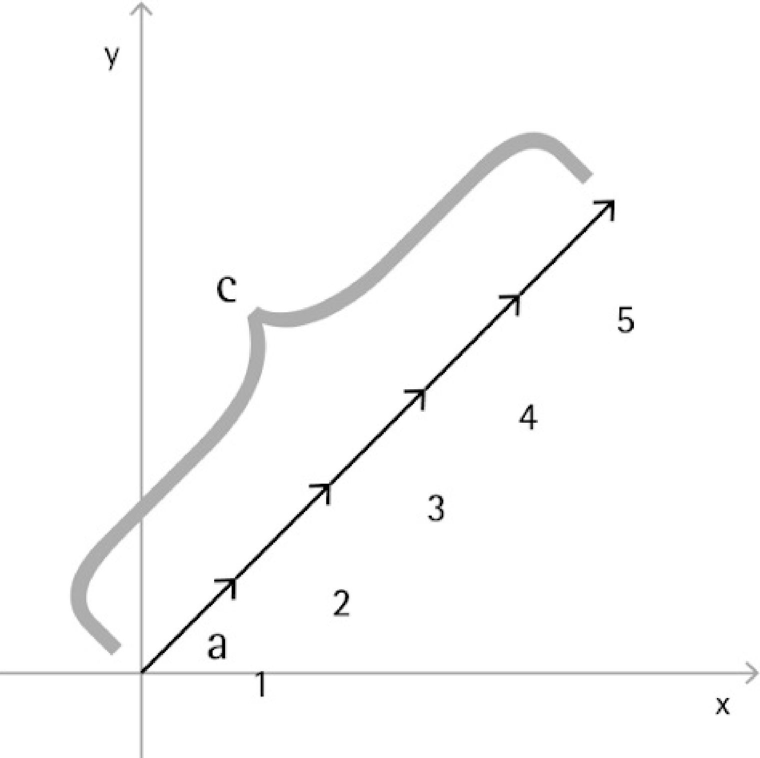
```
// c has value x = 9, y = 6, z = 0
c = a.Add(b);
```

Аналогичным образом можно вычесть один объект Vector из другого, используя метод *Subtract*. Результат вычитания двух векторов можно представить как расстояние от конечной точки одного вектора до конечной точки другого.



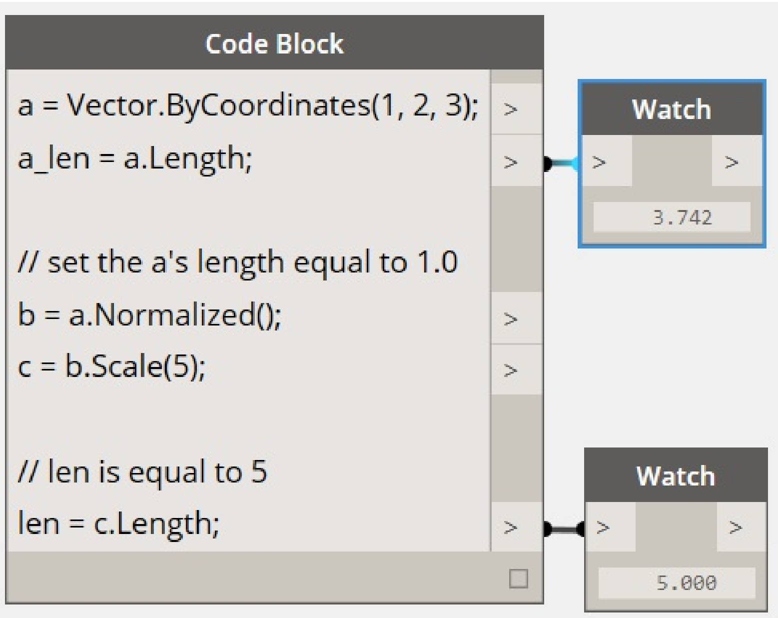

```
a = Vector.ByCoordinates(5, 5, 0);
b = Vector.ByCoordinates(4, 1, 0);

// c has value x = 1, y = 4, z = 0
c = a.Subtract(b);
```

При умножении объекта Vector на какое-либо число его конечная точка перемещается в направлении, соответствующем направлению вектора, на расстояние, соответствующее длине вектора и множителю.




```
a = Vector.ByCoordinates(4, 4, 0);

// c has value x = 20, y = 20, z = 0
c = a.Scale(5);
```

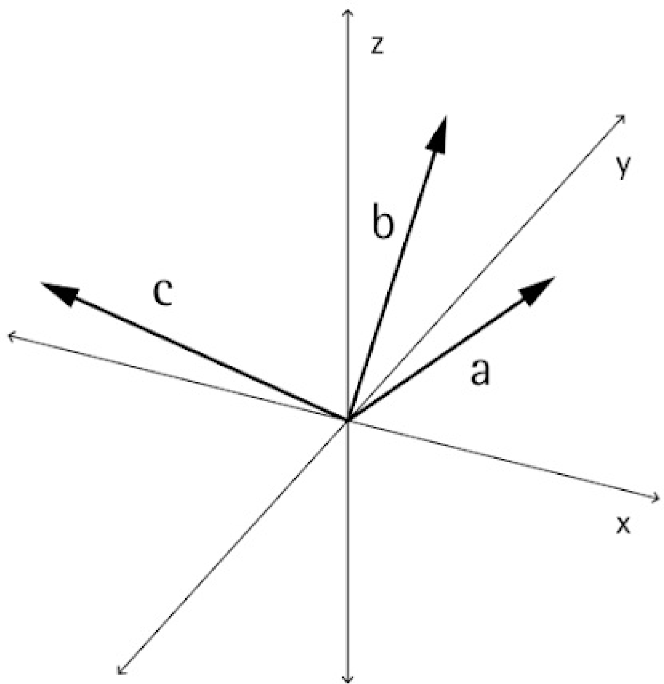
Зачастую при масштабировании вектора требуется, чтобы его длина стала равна значению коэффициента масштабирования. Этого можно легко добиться, выполнив нормализацию вектора, то есть приведя значение его длины к единице.




```
a = Vector.ByCoordinates(1, 2, 3);
a_len = a.Length;

// set the a's length equal to 1.0
b = a.Normalized();
c = b.Scale(5);

// len is equal to 5
len = c.Length;
```

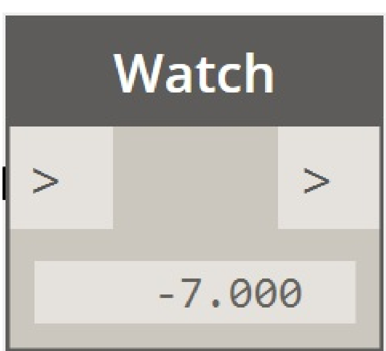
Вектор С указывает в том же направлении, что и вектор А (1, 2, 3), но его длина теперь равна 5.

Векторная материка включает два дополнительных метода, не имеющих точных аналогов в одномерной математике: это векторное произведение и скалярное произведение. При векторном произведении создается объект Vector, который ортогонален (т. е. находится под углом 90 градусов) к двум существующим объектам Vector. В качестве примера можно привести ось Z, которая является векторным произведением осей X и Y (хотя исходные объекты Vector, на основе которых вычисляется векторное произведение, не обязательно должны быть ортогональны друг другу). Для вычисления векторного произведения используется метод *Cross*.




```
a = Vector.ByCoordinates(1, 0, 1);
b = Vector.ByCoordinates(0, 1, 1);

// c has value x = -1, y = -1, z = 1
c = a.Cross(b);
```

Скалярное произведение — это еще одна дополнительная и чуть более сложная функция векторной математики. Результатом скалярного произведения двух векторов является вещественное число (а не объект Vector), соответствующее углу между этими векторами (но не равняющееся ему). Одним из полезных свойств этой функции является возможность определить, являются ли векторы перпендикулярными, поскольку только в этом случае их скалярное произведение равняется нулю. Для вычисления скалярного произведения используется метод *Dot*.



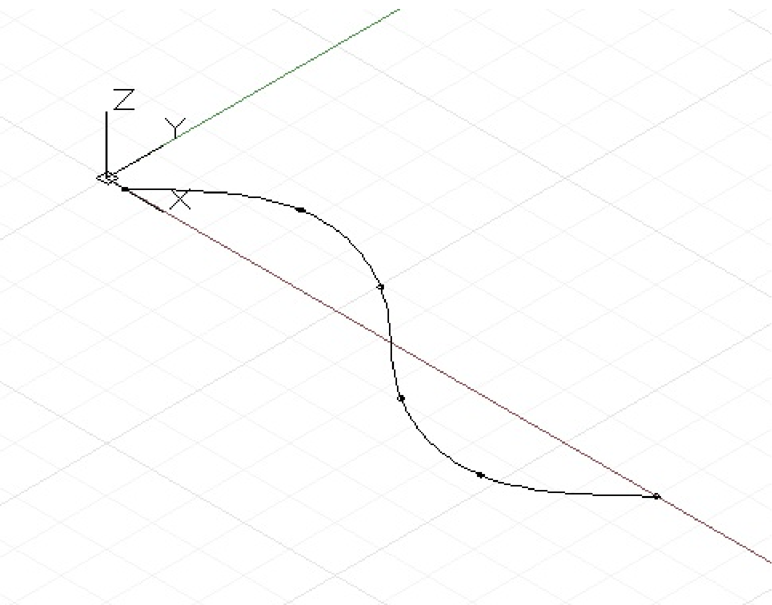

```
a = Vector.ByCoordinates(1, 2, 1);
b = Vector.ByCoordinates(5, -8, 4);

// d has value -7
d = a.Dot(b);
```

Кривые: интерполяционные и по управляющим точкам

Кривые: интерполяционные и по управляющим точкам

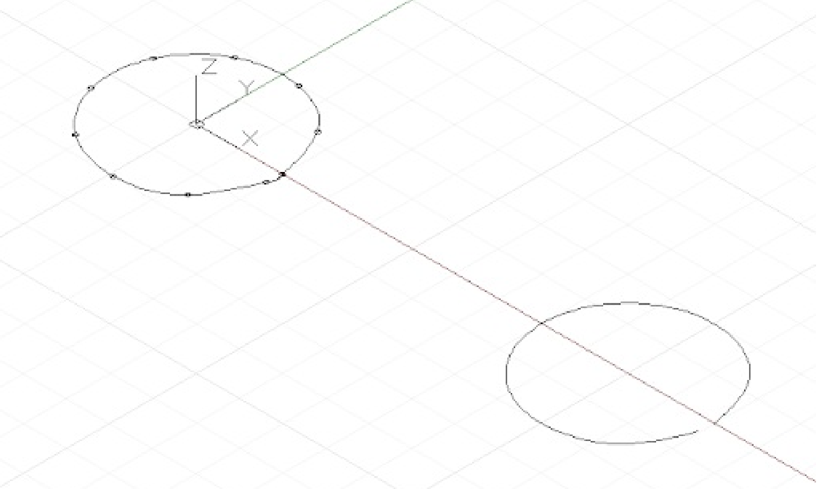
Создавать кривые произвольной формы в Dynamo можно двумя основными способами: задать набор точек и создать сглаженную кривую между ними путем интерполяции либо, если требуется получить кривую с конкретной степенью сглаживания, задать построение по управляющим точкам. Интерполяционные кривые подходят для случаев, когда проектировщик точно знает, какой формы должна быть линия, или когда в проекте заданы конкретные зависимости, определяющие то, где кривая может проходить, а где не может. Кривые по управляющим точкам представляют собой набор прямолинейных сегментов, который путем применения алгоритма сглаживается до получения требуемой кривой. Построение кривой по управляющим точкам позволяет сравнивать варианты с разными степенями сглаживания, а также обеспечивать последовательное применение сглаживания к криволинейным сегментам.

Для построения интерполяционной кривой достаточно задать набор точек и использовать его в качестве входных данных для метода *NurbsCurve.ByPoints*.




```
num_pts = 6;

s = Math.Sin(0..360..#num_pts) * 4;

pts = Point.ByCoordinates(1..30..#num_pts, s, 0);

int_curve = NurbsCurve.ByPoints(pts);
```

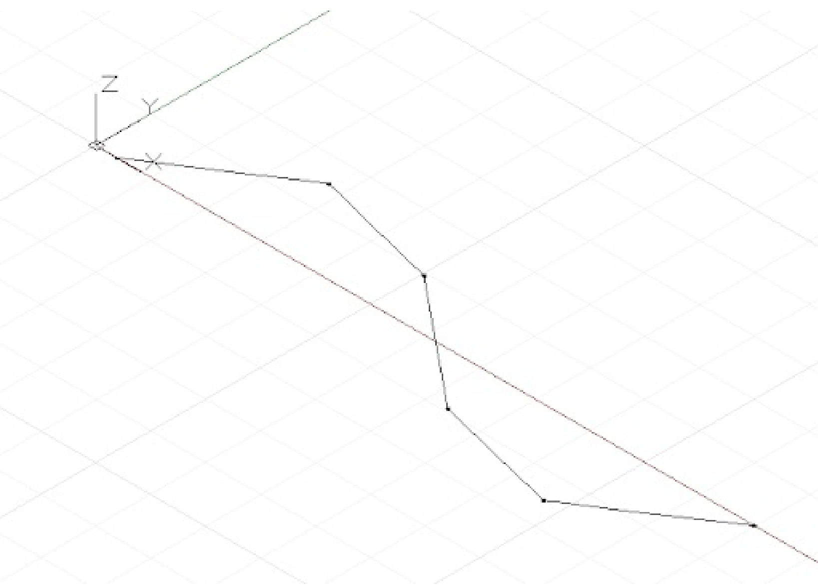
Полученная кривая пересекает каждую из заданных точек, а ее начало и конец находятся в первой и последней точках заданного набора соответственно. Задав дополнительный параметр периодичности, можно получить замкнутую периодическую кривую. При этом Dynamo автоматически подставит отсутствующий сегмент, так что отдельно задавать конечную точку, совпадающую с начальной точкой, не требуется.




```
pts = Point.ByCoordinates(Math.Cos(0..350..#10),
```

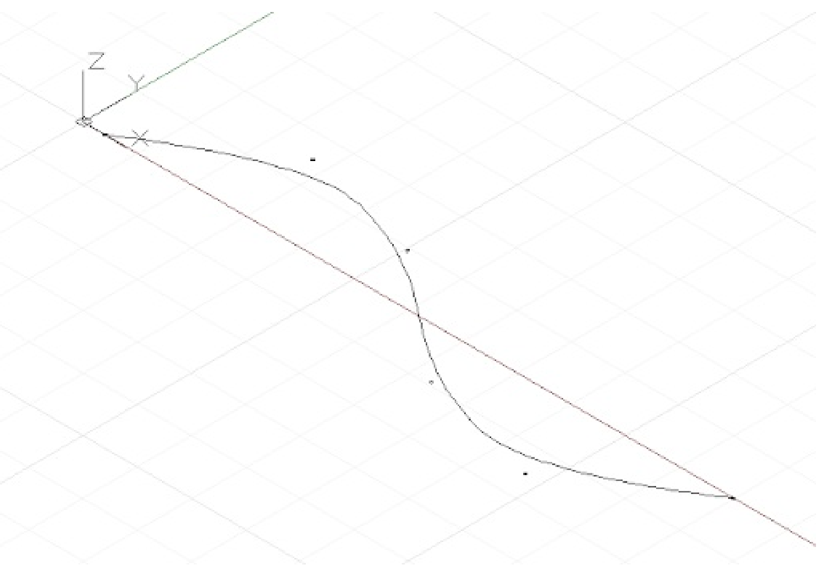
```
    Math.Sin(0..350..#10), 0);

// create an closed curve
crv = NurbsCurve.ByPoints(pts, true);

// the same curve, if left open:
crv2 = NurbsCurve.ByPoints(pts.Translate(5, 0, 0),
    false);
```

Построение объектов NurbsCurve выполняется схожим образом. В качестве первого параметра задается набор точек (а именно конечных точек прямолинейных сегментов), а в качестве второго — величина и тип (т. е. степень) сглаживания кривой.* Кривая со степенью сглаживания 1 не сглаживается и представляет собой полилинию.



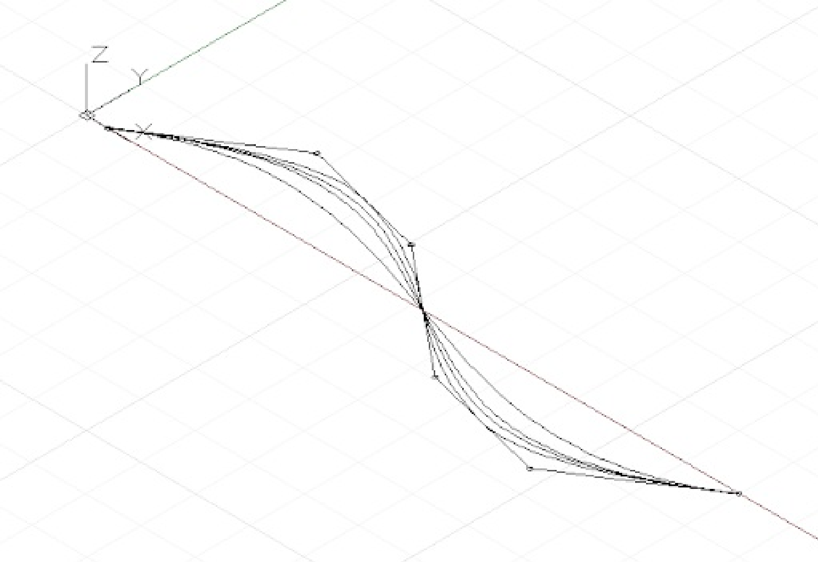

```
num_pts = 6;

pts = Point.ByCoordinates(1..30..#num_pts,
    Math.Sin(0..360..#num_pts) * 4, 0);

// a B-Spline curve with degree 1 is a polyline
ctrl_curve = NurbsCurve.ByControlPoints(pts, 1);
```

Кривая со степенью сглаживания 2 сглаживается таким образом, чтобы она проходила сквозь и по касательной к средним точкам сегментов полилинии:




```
num_pts = 6;

pts = Point.ByCoordinates(1..30..#num_pts,
    Math.Sin(0..360..#num_pts) * 4, 0);

// a B-Spline curve with degree 2 is smooth
ctrl_curve = NurbsCurve.ByControlPoints(pts, 2);
```

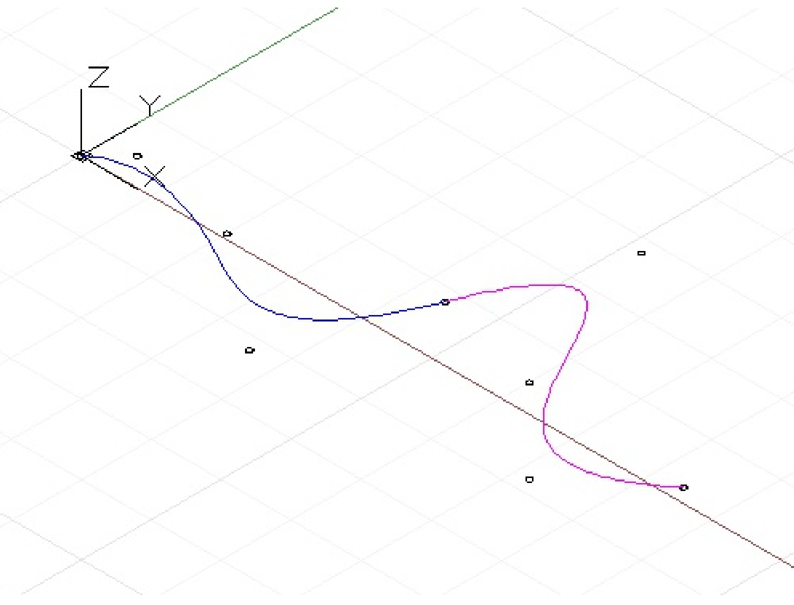
В Dynamo поддерживаются NURBS-кривые (неоднородные рациональные B-сплайны) со степенью сглаживания от 1 до 20. Приведенный ниже сценарий демонстрирует, как повышение степени сглаживания влияет на форму кривой.




```
num_pts = 6;

pts = Point.ByCoordinates(1..30..#num_pts,
    Math.Sin(0..360..#num_pts) * 4, 0);

def create_curve(pts : Point[], degree : int)
{
    return = NurbsCurve.ByControlPoints(pts,
        degree);
}

ctrl_crvs = create_curve(pts, 1..11);
```

Обратите внимание, что число управляющих точек должно превышать значение степени сглаживания как минимум на единицу.

Еще одно преимущество построения кривых по управляющим вершинам — возможность сохранения касательности между отдельными криволинейными сегментами. Для этого программа определяет направление при движении от предпоследней управляющей точки сегмента к последней, а затем размещает две первые управляющие точки следующего сегмента в соответствии с этим направлением. В следующем примере показаны две отдельные NURBS-кривые, которые при этом выглядят как единая сглаженная кривая.




```
pts_1 = {};

pts_1[0] = Point.ByCoordinates(0, 0, 0);
pts_1[1] = Point.ByCoordinates(1, 1, 0);
pts_1[2] = Point.ByCoordinates(5, 0.2, 0);
pts_1[3] = Point.ByCoordinates(9, -3, 0);
pts_1[4] = Point.ByCoordinates(11, 2, 0);
```

```
crv_1 = NurbsCurve.ByControlPoints(pts_1, 3);

pts_2 = {};

pts_2[0] = pts_1[4];
end_dir = pts_1[4].Subtract(pts_1[3].AsVector());

pts_2[1] = Point.ByCoordinates(pts_2[0].X + end_dir.X,
    pts_2[0].Y + end_dir.Y, pts_2[0].Z + end_dir.Z);

pts_2[2] = Point.ByCoordinates(15, 1, 0);
pts_2[3] = Point.ByCoordinates(18, -2, 0);
pts_2[4] = Point.ByCoordinates(21, 0.5, 0);

crv_2 = NurbsCurve.ByControlPoints(pts_2, 3);
```

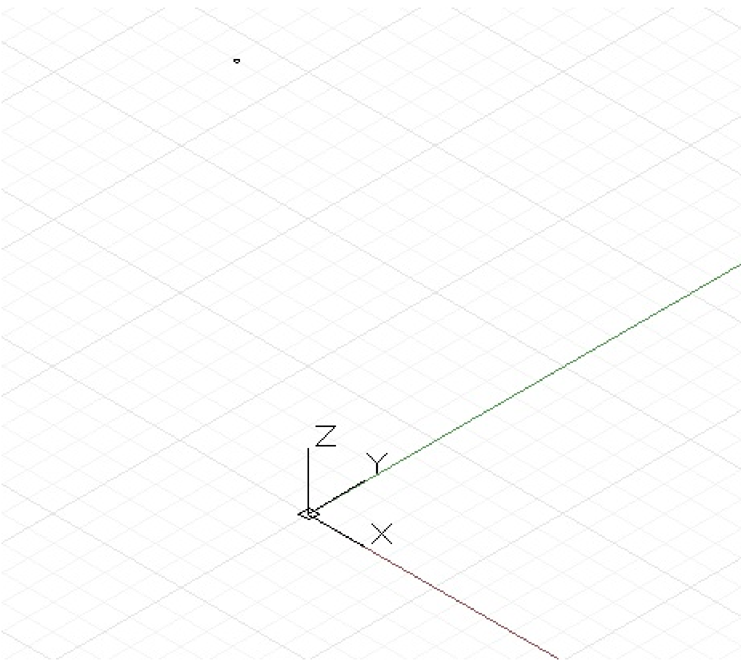
- В данной главе приведено упрощенное описание геометрии NURBS-кривых. Для получения подробных сведений см. Pottmann, et al, 2007 г., в списке литературы.

Перенос, поворот и другие преобразования

Перенос, поворот и другие преобразования

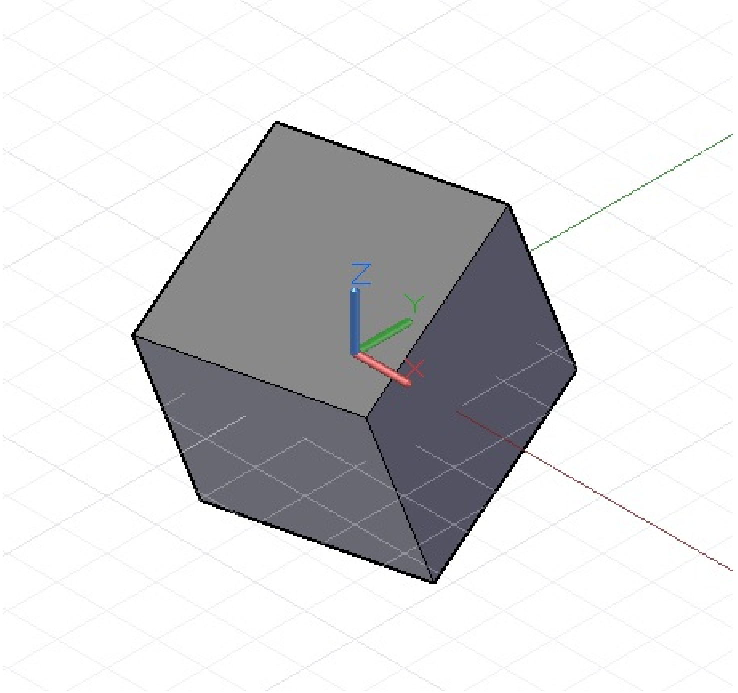
Определенные геометрические объекты можно создавать путем непосредственного указания их координат по осям X, Y и Z в трехмерном пространстве. Однако в большинстве случаев итоговое положение геометрии задается путем преобразований, применяемых либо к самому объекту, либо к его базовой системе координат (CoordinateSystem).

Простейшим геометрическим преобразованием является перенос, в результате которого объект перемещается на заданное число делений вдоль осей X, Y и Z.



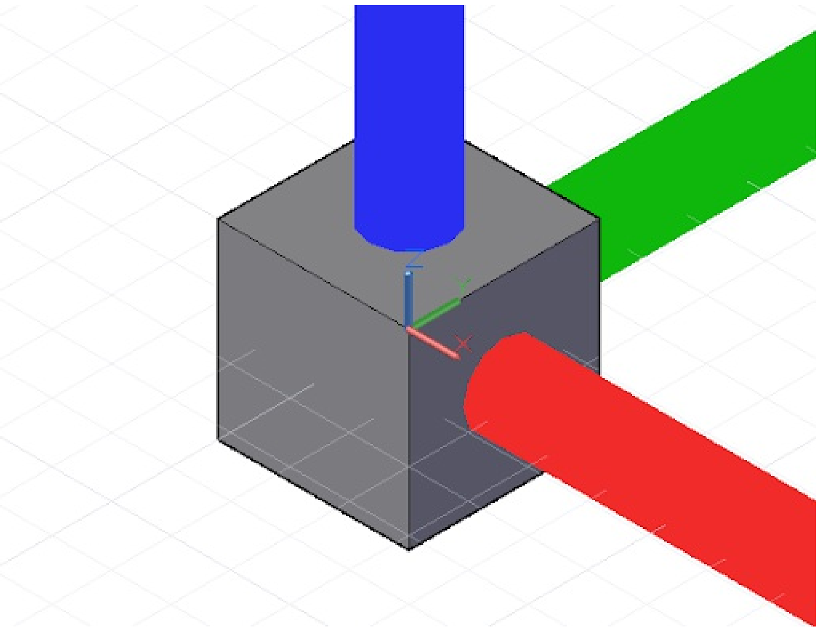

```
// create a point at x = 1, y = 2, z = 3
p = Point.ByCoordinates(1, 2, 3);

// translate the point 10 units in the x direction,
// -20 in y, and 50 in z
// p2’s new position is x = 11, y = -18, z = 53
p2 = p.Translate(10, -20, 50);
```

Любой объект в Dynamo можно перенести, просто добавив метод .*Translate* после имени объекта, однако более сложные преобразования требуют изменения базовой системы координат объекта (CoordinateSystem). Например, чтобы повернуть объект на 45 градусов вокруг оси X, потребуется заменить существующую систему координат (без поворота) на систему координат, повернутую на 45 градусов вокруг оси X с помощью метода .*Transform*.




```
cube = Cuboid.ByLengths(CoordinateSystem.Identity(),
    10, 10, 10);

new_cs = CoordinateSystem.Identity();
new_cs2 = new_cs.Rotate(Point.ByCoordinates(0, 0),
    Vector.ByCoordinates(1,0,0.5), 25);

// get the existing coordinate system of the cube
old_cs = CoordinateSystem.Identity();

cube2 = cube.Transform(old_cs, new_cs2);
```

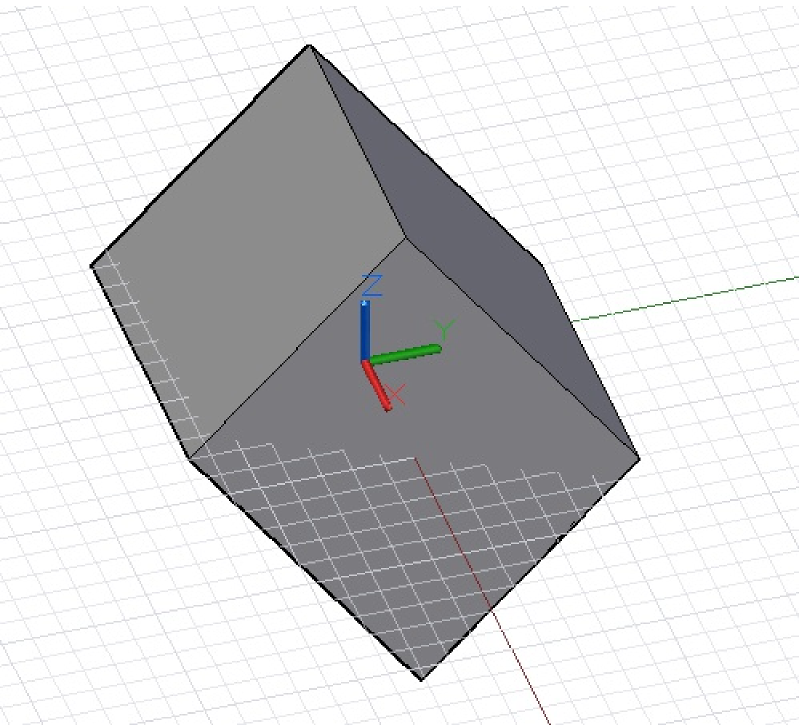
Помимо операций переноса и поворота к объектам CoordinateSystem также можно применять операции масштабирования и сдвига. Для масштабирования объектов CoordinateSystem используется метод *.Scale*.



```
cube = Cuboid.ByLengths(CoordinateSystem.Identity(),
    10, 10, 10);

new_cs = CoordinateSystem.Identity();
new_cs2 = new_cs.Scale(20);

old_cs = CoordinateSystem.Identity();
```

```
cube2 = cube.Transform(old_cs, new_cs2);
```

Для сдвига объекта CoordinateSystem используется конструктор CoordinateSystem, в который нужно ввести неортогональные векторы.




```
new_cs = CoordinateSystem.ByOriginVectors(
    Point.ByCoordinates(0, 0, 0),
    Vector.ByCoordinates(-1, -1, 1),
    Vector.ByCoordinates(-0.4, 0, 0));

old_cs = CoordinateSystem.Identity();

cube = Cuboid.ByLengths(CoordinateSystem.Identity(),
    5, 5, 5);

new_curves = cube.Transform(old_cs, new_cs);
```

По сравнению с поворотом и переносом операции масштабирования и сдвига являются более сложными и поддерживаются не всеми объектами Dynamo. Объекты Dynamo, которые поддерживают операции изменения масштаба и сдвига системы координат, перечислены в следующей таблице.

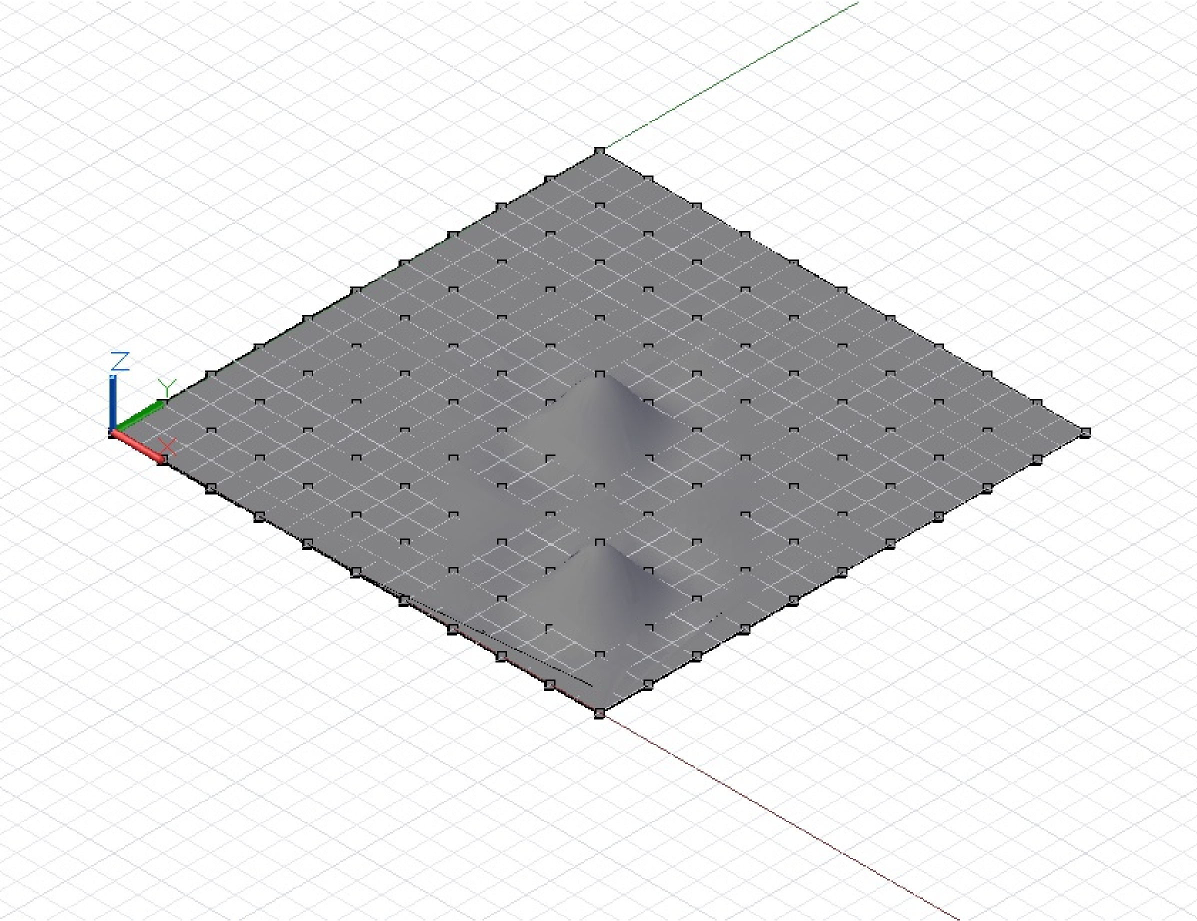
Класс	Объект CoordinateSystem с измененным масштабом	Объект CoordinateSystem со сдвигом
Дуга	Нет	Нет
Объект NurbsCurve	Да	Да
Объект NurbsSurface	Нет	Нет
Окружность	Нет	Нет
Отрезок	Да	Да
Плоскость	Нет	Нет
Точка	Да	Да
Полигон	Нет	Нет
Тело	Нет	Нет
Поверхность	Нет	Нет
Текст	Нет	Нет

Поверхности: интерполяционные, лофтированные, по управляющим точкам и поверхности вращения

Поверхности: интерполяционные, лофтированные, по управляющим точкам и поверхности вращения

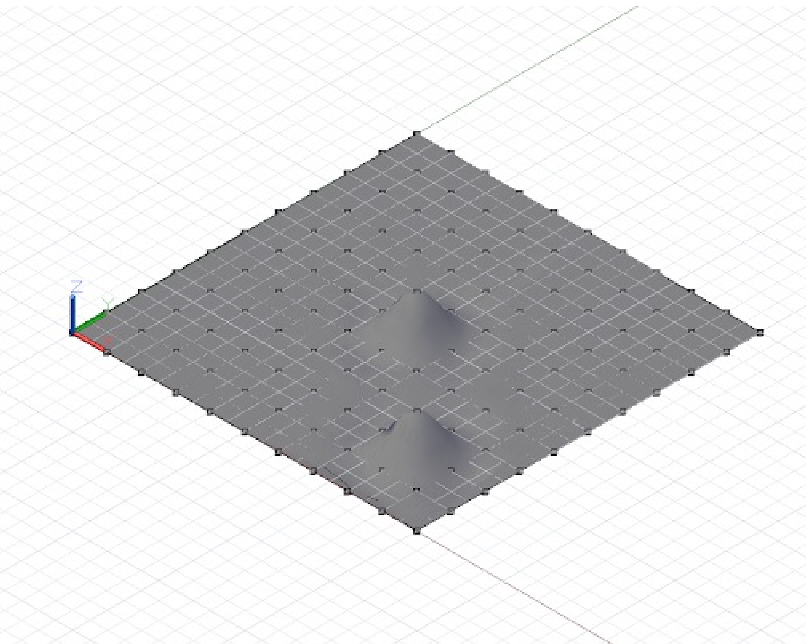
Объект NurbsSurface является двумерным аналогом объекта NurbsCurve. Как и объекты NurbsCurve, объекты NurbsSurfaces создаются двумя основными способами: путем интерполяции набора базовых точек либо указания управляющих точек поверхности. Аналогично кривым произвольной формы интерполяционные поверхности подходят для случаев, когда проектировщик точно знает, какой формы должна быть поверхность, или когда в проекте заданы конкретные точки зависимости, через которые плоскость должна проходить. Поверхности по управляющим точкам подходят для случаев, требующих анализа поведения поверхности при различных уровнях сглаживания.

Для создания интерполяционной поверхности достаточно задать двумерный набор точек, приблизительно соответствующий форме поверхности. Набор должен быть прямоугольным, без изломов. Чтобы создать поверхность из этих точек, используйте метод *NurbsSurface.ByPoints*.



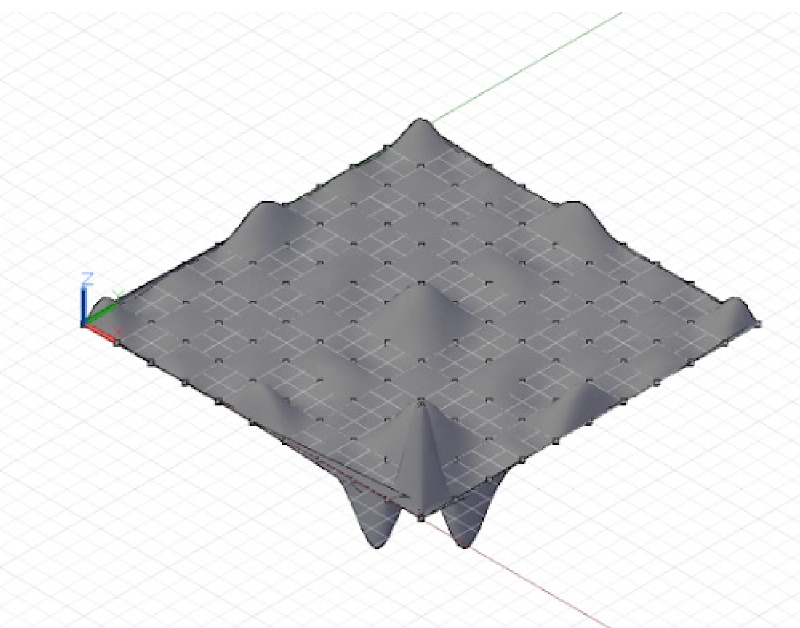

```
// python_points_1 is a set of Points generated with
// a Python script found in Chapter 12, Section 10

surf = NurbsSurface.ByPoints(python_points_1);
```

Кроме того, объекты NurbsSurface произвольной формы можно создавать путем задания базовых управляющих точек. Как и в случае с объектами NurbsCurve, управляющие точки образуют четырехугольную сеть, состоящую из прямолинейных сегментов, к которой применена та или иная степень сглаживания. Для создания объекта NurbsSurface по управляющим точкам в метод *NurbsSurface.ByPoints* необходимо добавить два дополнительных параметра, которые позволяют указать степень сглаживания базовых кривых по обеим сторонам поверхности.



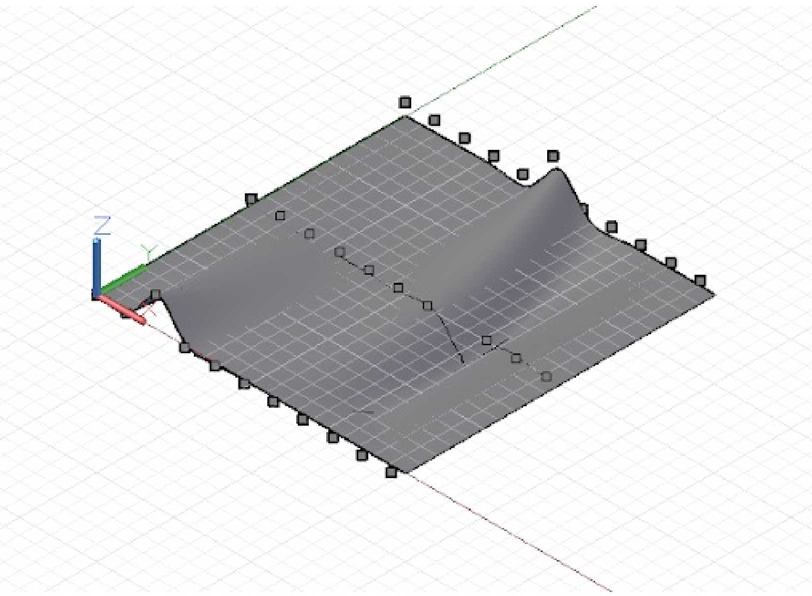
```
// python_points_1 is a set of Points generated with
// a Python script found in Chapter 12, Section 10

// create a surface of degree 2 with smooth segments
surf = NurbsSurface.ByPoints(python_points_1, 2, 2);
```

Увеличив степень сглаживания объекта NurbsSurface, можно изменить итоговую геометрию поверхности.



```
// python_points_1 is a set of Points generated with
// a Python script found in Chapter 12, Section 10

// create a surface of degree 6
surf = NurbsSurface.ByPoints(python_points_1, 6, 6);
```

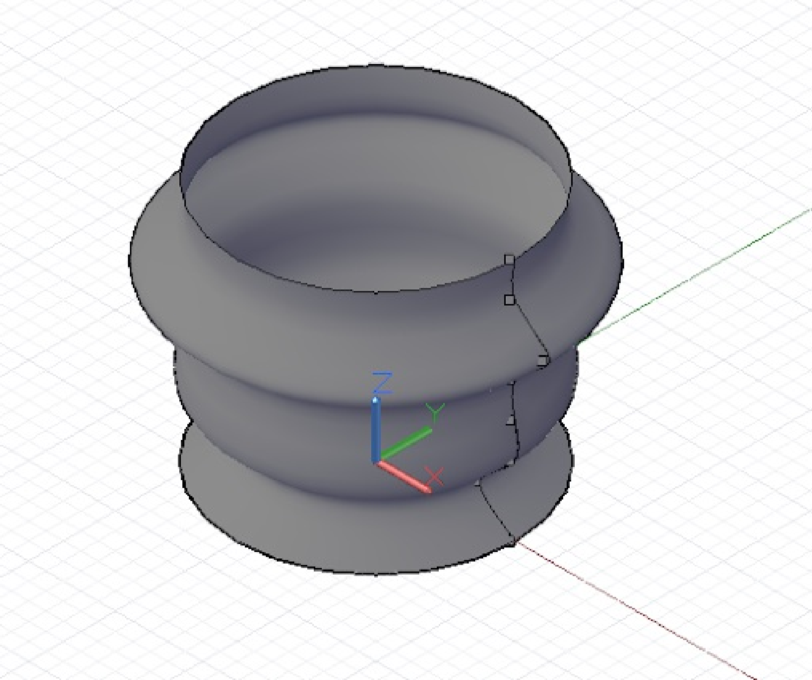
Интерполяционные поверхности можно создавать не только на основе наборов точек, но и на основе наборов кривых. Это называется лофтингом. Для построения лофтированных поверхностей используется конструктор *Surface.ByLoft*, где в качестве единственного входного параметра указывается набор кривых.




```
// python_points_2, 3, and 4 are generated with
// Python scripts found in Chapter 12, Section 10

c1 = NurbsCurve.ByPoints(python_points_2);
c2 = NurbsCurve.ByPoints(python_points_3);
c3 = NurbsCurve.ByPoints(python_points_4);

loft = Surface.ByLoft([c1, c2, c3]);
```

Поверхности вращения — это дополнительный тип поверхностей, который создается путем сдвига базовой кривой относительно центральной оси. Такие поверхности являются двумерным аналогом окружностей и дуг, точно так же как интерполяционные поверхности являются двумерным аналогом интерполяционных кривых.

Для построения поверхности вращения необходимо задать базовую кривую («кромку» поверхности), начало координат оси (базовую точку поверхности), направление оси (направление «центра», вокруг которого строится поверхность), начальный и конечный углы сдвига. Все эти данные используются в качестве входных параметров конструктора *Surface.Revolve*.




```
pts = {};
pts[0] = Point.ByCoordinates(4, 0, 0);
pts[1] = Point.ByCoordinates(3, 0, 1);
pts[2] = Point.ByCoordinates(4, 0, 2);
pts[3] = Point.ByCoordinates(4, 0, 3);
pts[4] = Point.ByCoordinates(4, 0, 4);
pts[5] = Point.ByCoordinates(5, 0, 5);
```

```
pts[6] = Point.ByCoordinates(4, 0, 6);
pts[7] = Point.ByCoordinates(4, 0, 7);

crv = NurbsCurve.ByPoints(pts);

axis_origin = Point.ByCoordinates(0, 0, 0);
axis = Vector.ByCoordinates(0, 0, 1);

surf = Surface.ByRevolve(crv, axis_origin, axis, 0,
    360);
```

Параметризация геометрических объектов

Параметризация геометрических объектов

В машинном проектировании кривые и поверхности часто используются в качестве каркаса, поверх которого затем надстраивается более сложная геометрия. Чтобы эти базовые геометрические объекты можно было использовать в качестве основы для других объектов, необходимо написать сценарий для извлечения количественных характеристик, таких как положение и ориентация, по всей площади такого объекта. И кривые, и поверхности поддерживают этот процесс, который называется параметризацией.

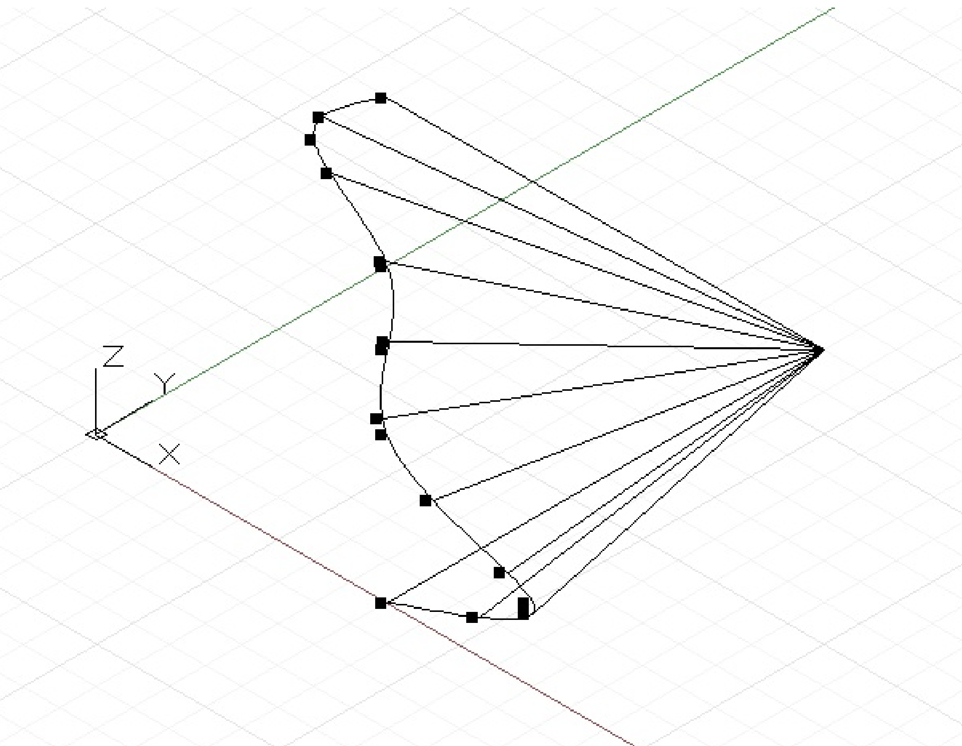
Представим, что каждой точке на кривой назначен уникальный параметр в диапазоне от 0 до 1. Если мы создаем объект NurbsCurve путем интерполяции или по нескольким управляющим точкам, то первой точке назначается параметр со значением 0, а последней — параметр со значением 1. Заранее узнать, какой именно параметр будет назначен любой из промежуточных точек, невозможно. Эта проблему можно свести к минимуму путем использования нескольких вспомогательных функций. Процесс параметризации поверхностей аналогичен процессу для кривых, за исключением того, что вместо одного параметра назначаются два — u и v. Предположим, что требуется создать поверхность по следующим точкам:

```
pts = [ [p1, p2, p3],
        [p4, p5, p6],
        [p7, p8, p9] ];
```

В этом случае точке p1 будут назначены параметры u = 0, v = 0, а точке p9 — u = 1, v = 1.

Для определения точек, на основе которых строится кривая, параметризация не слишком эффективна. Ее основное назначение — определение местоположений промежуточных точек, создаваемых конструкторами NurbsCurve и NurbsSurface.

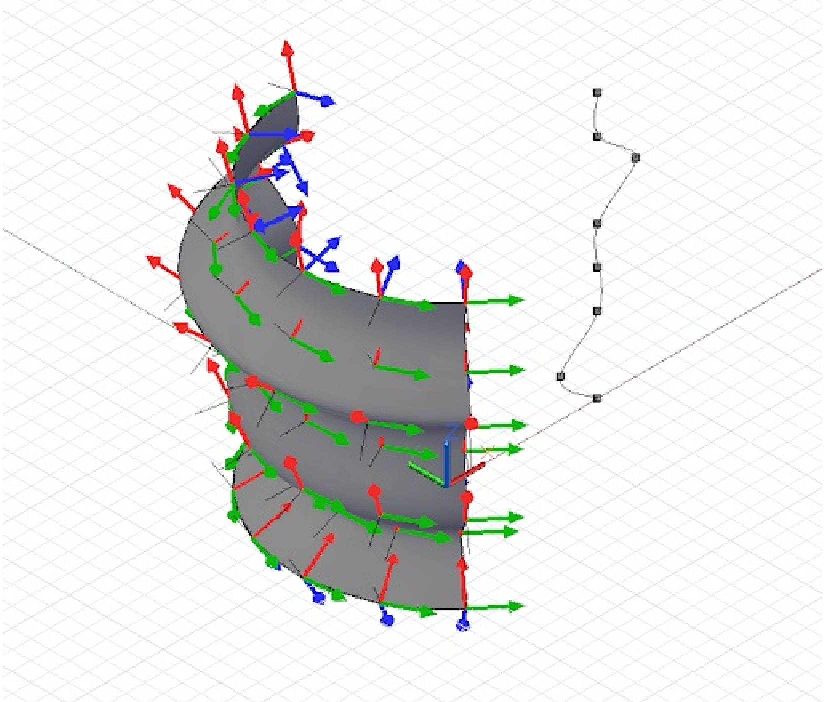
Для работы с кривыми доступен метод *PointAtParameter*, который в качестве входных данных использует один двойной аргумент в диапазоне между 0 и 1 и возвращает объект Point, соответствующий этому параметру. Например, при использовании этого сценария были определены точки с параметрами 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 и 1:




```
pts = {};
pts[0] = Point.ByCoordinates(4, 0, 0);
pts[1] = Point.ByCoordinates(6, 0, 1);
pts[2] = Point.ByCoordinates(4, 0, 2);
pts[3] = Point.ByCoordinates(4, 0, 3);
pts[4] = Point.ByCoordinates(4, 0, 4);
pts[5] = Point.ByCoordinates(3, 0, 5);
pts[6] = Point.ByCoordinates(4, 0, 6);

crv = NurbsCurve.ByPoints(pts);

pts_at_param = crv.PointAtParameter(0..1..#11);

// draw Lines to help visualize the points
lines = Line.ByStartPointEndPoint(pts_at_param,
    Point.ByCoordinates(4, 6, 0));
```

Для работы с поверхностями доступен аналогичный метод *PointAtParameter*, который в качестве входных данных использует два аргумента, а именно параметры u и v созданного объекта Point.

Хотя извлечение отдельных точек кривой или поверхности может быть само по себе полезно, для выполнения многих сценариев требуются определенные геометрические характеристики конкретного параметра, например направление кривой или поверхности. Метод *CoordinateSystemAtParameter* позволяет определить не только положение конкретного параметра кривой или поверхности, но и ориентацию соответствующего объекта CoordinateSystem. Например, следующий сценарий извлекает сведения об ориентации объектов CoordinateSystem относительно поверхности вращения и использует эти сведения для построения отрезков перпендикулярно этой поверхности:



```
pts = {};
pts[0] = Point.ByCoordinates(4, 0, 0);
pts[1] = Point.ByCoordinates(3, 0, 1);
pts[2] = Point.ByCoordinates(4, 0, 2);
pts[3] = Point.ByCoordinates(4, 0, 3);
pts[4] = Point.ByCoordinates(4, 0, 4);
pts[5] = Point.ByCoordinates(5, 0, 5);
pts[6] = Point.ByCoordinates(4, 0, 6);
pts[7] = Point.ByCoordinates(4, 0, 7);

crv = NurbsCurve.ByPoints(pts);

axis_origin = Point.ByCoordinates(0, 0, 0);
axis = Vector.ByCoordinates(0, 0, 1);

surf = Surface.ByRevolve(crv, axis_origin, axis, 90,
    140);

cs_array = surf.CoordinateSystemAtParameter(
    (0..1..#7)<1>, (0..1..#7)<2>);

def make_line(cs : CoordinateSystem) {
    lines_start = cs.Origin;
    lines_end = cs.Origin.Translate(cs.ZAxis, -0.75);

    return = Line.ByStartPointEndPoint(lines_start,
        lines_end);
}

lines = make_line(Flatten(cs_array));
```

Как уже упоминалось, параметризация не всегда выполняется равномерно по всей длине кривой или поверхности. Это значит, что параметр 0.5 не всегда соответствует средней точке, а параметр 0.25 — точке на отметке в одну четвертую длины кривой или поверхности. Чтобы обойти это ограничение, можно воспользоваться дополнительным набором команд параметризации, доступным для объектов Curve. Эти команды позволяют найти точки, расположенные в определенном месте на кривой.

Пересечение и обрезка

Пересечение и обрезка

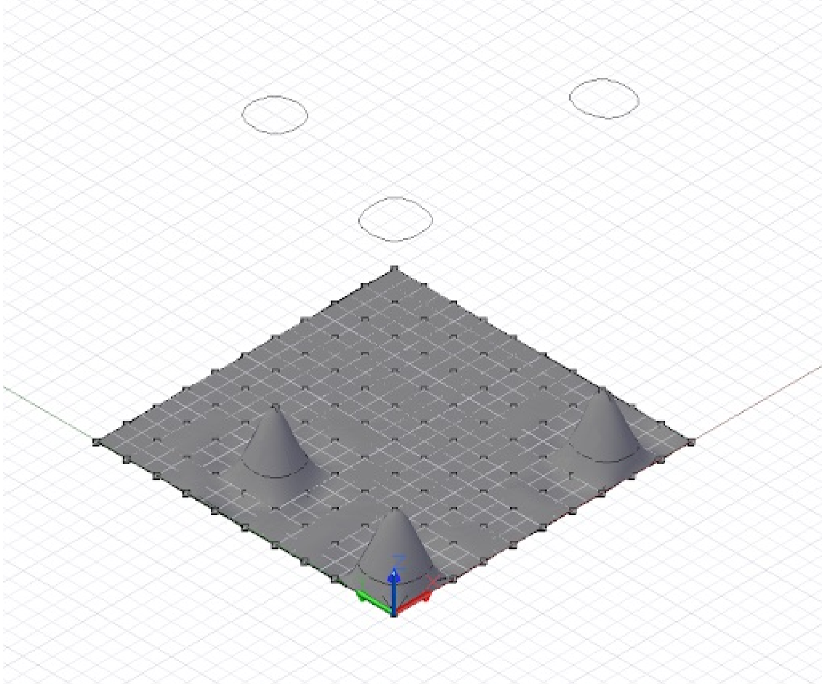
Во многих рассмотренных примерах основное внимание было уделено построению более сложных геометрических объектов на основе более простых объектов. Методы Intersect позволяют делать обратное, а именно получать простые геометрические объекты из сложных. Команды Trim и SelectTrim позволяют использовать сценарии для масштабных преобразований геометрических форм после их создания.

Метод *Intersect* поддерживается всеми геометрическими объектами в Dynamo. В теории это значит, что любой геометрический объект может пересекаться с любым другим геометрическим объектом. Разумеется, построение определенных пересечений, таких как пересечение объектов Point, не имеет смысла, поскольку результатом пересечения будет тот же объект Point, который был задан на входе. Другие возможные комбинации пересекающихся объектов приведены в следующей таблице. В ней указаны возможные результаты различных операций пересечения.

Пересечение

Пересекающиеся объекты	**Поверхность**	**Кривая**	**Плоскость**	**Тело**
Поверхность	Кривая	Точка	Точка, кривая	Поверхность
Кривая	Точка	Точка	Точка	Кривая
Плоскость	Кривая	Точка	Кривая	Кривая
Тело	Поверхность	Кривая	Кривая	Тело

Приведенный ниже пример иллюстрирует пересечение плоскости с поверхностью NurbsSurface. В результате пересечения получается массив NurbsCurve, который можно использовать как любой другой объект NurbsCurve.



```
// python_points_5 is a set of Points generated with
// a Python script found in Chapter 12, Section 10

surf = NurbsSurface.ByPoints(python_points_5, 3, 3);

WCS = CoordinateSystem.Identity();

pl = Plane.ByOriginNormal(WCS.Origin.Translate(0, 0,
    0.5), WCS.ZAxis);

// intersect surface, generating three closed curves
crvs = surf.Intersect(pl);

crvs_moved = crvs.Translate(0, 0, 10);
```

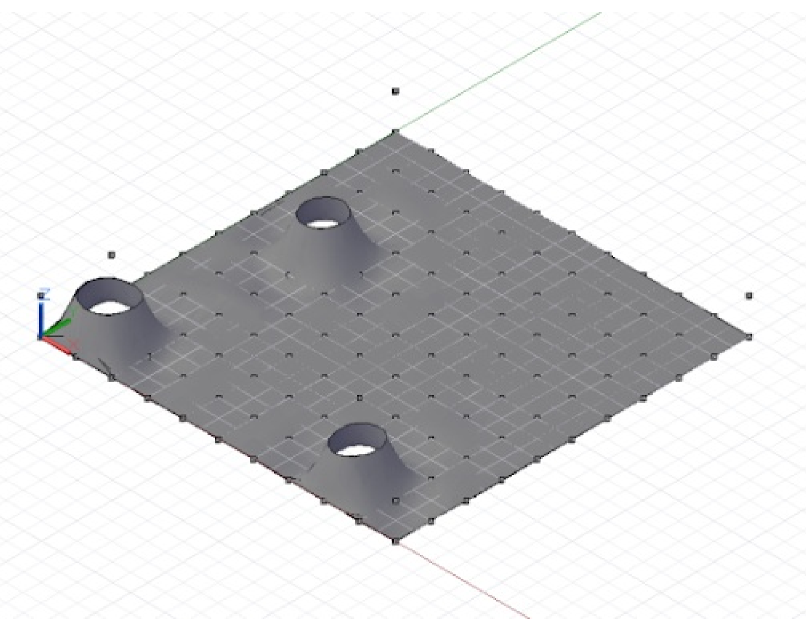
Метод *Trim* очень похож на метод Intersect. Он поддерживается почти для всех геометрических объектов. Однако в отношении метода *Trim* действует больше ограничений, чем в отношении метода *Intersect*.

Обрезка

Объект, с помощью которого выполняется обрезка:

	точка				
Объект, на котором выполняется обрезка: кривая	Да	Нет	Нет	Нет	Нет
Полигон	-	Нет	Да	Нет	Нет
Поверхность	-	Да	Да	Да	Да
Тело	-	-	Да	Да	Да

Особенностью методов *Trim* является обязательное наличие точки «выбора», определяющей, какая геометрия будет обрезана, а какая сохранена. Dynamo использует точку выбора для выявления геометрии, находящейся ближе всего к этой точке, и затем обрезает эту геометрию.



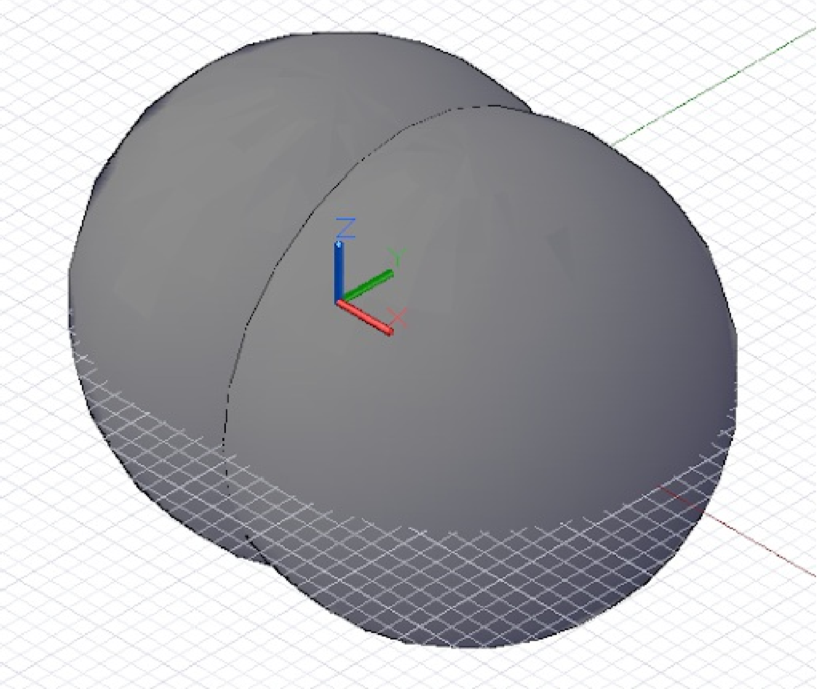
```
// python_points_5 is a set of Points generated with
// a Python script found in Chapter 12, Section 10

surf = NurbsSurface.ByPoints(python_points_5, 3, 3);

tool_pts = Point.ByCoordinates((-10..20..10)<1>,
    (-10..20..10)<2>, 1);

tool = NurbsSurface.ByPoints(tool_pts);

pick_point = Point.ByCoordinates(8, 1, 3);

result = surf.Trim(tool, pick_point);
```

Логические операции с геометрическими объектами

Логические операции с геометрическими объектами

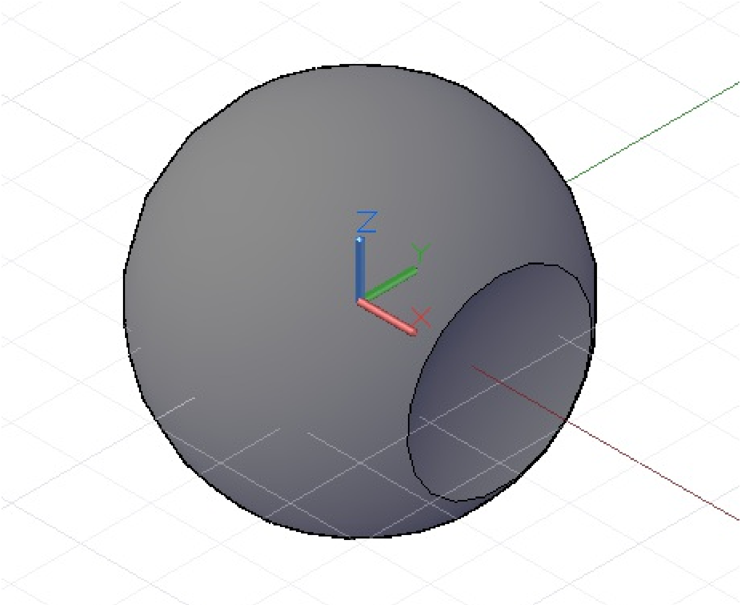
Методы *Intersect*, *Trim* и *SelectTrim* в основном используются при работе с простыми геометрическими объектами, такими как точки, кривые и поверхности. Для твердотельных геометрических объектов доступны дополнительные методы изменения формы после ее построения. Эти методы включают как удаление материала аналогично методу *Trim*, так и объединение нескольких элементов для получения единого большого элемента.

Метод *Union* позволяет создать новый твердотельный объект на основе двух исходных объектов. Итоговый объект занимает в пространстве столько же места, сколько занимали оба исходных. Если объекты накладываются друг на друга в пространстве, то в итоговой форме накладывающиеся участки объединяются. В этом примере из сферы и кубоида путем объединения была получена единая кубо-сферическая твердотельная форма:



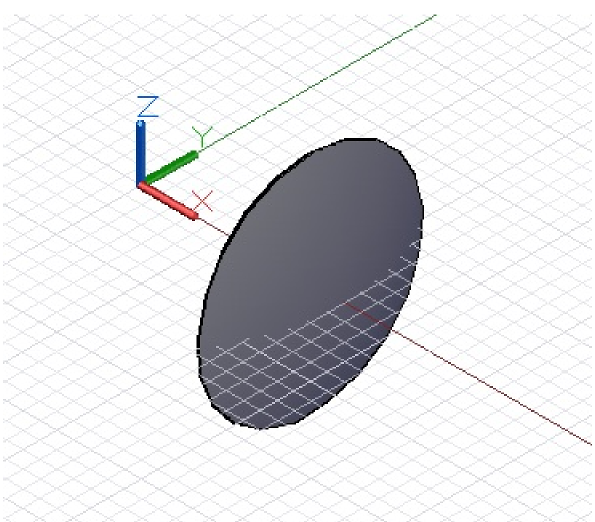

```
s1 = Sphere.ByCenterPointRadius(
    CoordinateSystem.Identity().Origin, 6);

s2 = Sphere.ByCenterPointRadius(
    CoordinateSystem.Identity().Origin.Translate(4, 0,
    0), 6);

combined = s1.Union(s2);
```

Метод *Difference*, аналогично методу *Trim*, позволяет удалить из базового тела материал, объем которого соответствует используемому на входе твердотельному инструменту. В этом примере в сфере был создан небольшой вырез:




```
s = Sphere.ByCenterPointRadius(
    CoordinateSystem.Identity().Origin, 6);

tool = Sphere.ByCenterPointRadius(
    CoordinateSystem.Identity().Origin.Translate(10, 0,
    0), 6);

result = s.Difference(tool);
```

Результатом использования метода *Intersect* является тело, образованное наложением двух других тел. В следующем примере вместо метода *Difference* был использован метод *Intersect*, в результате чего было получено тело, объем которого соответствует вырезу в предыдущем примере:

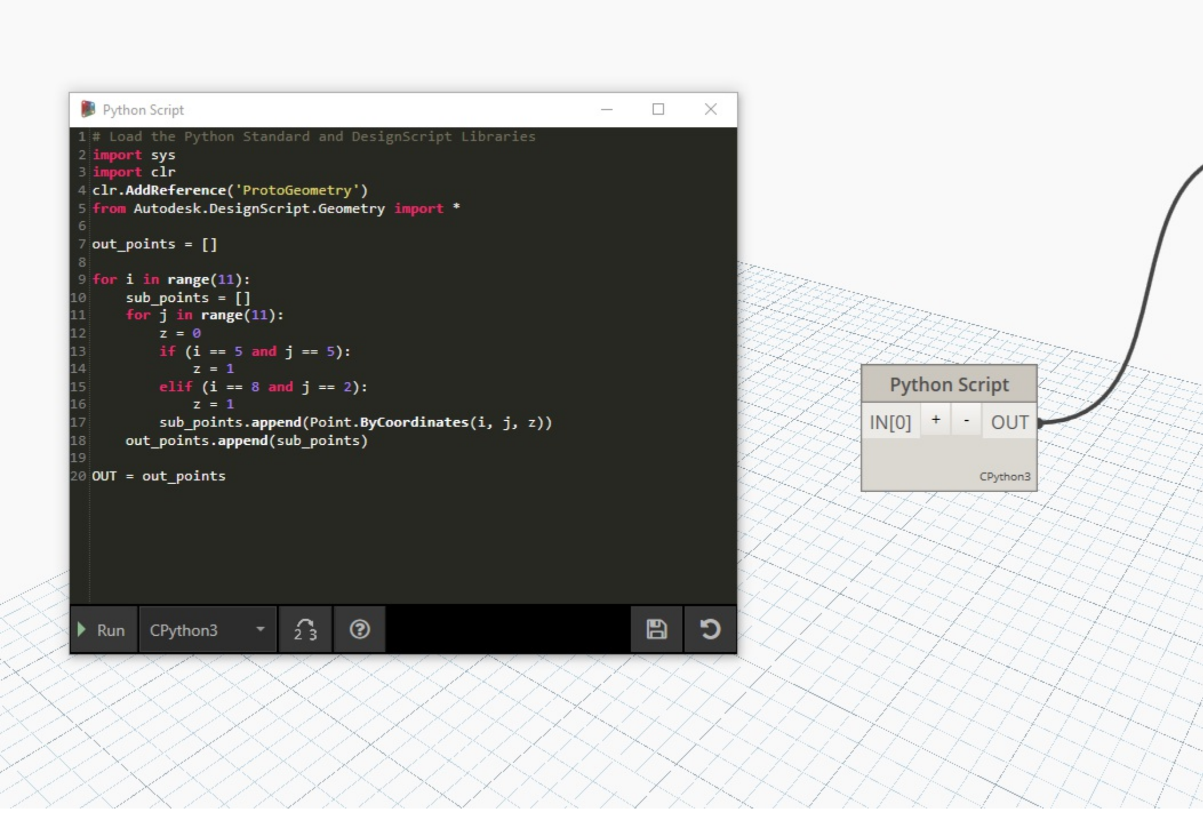



```
s = Sphere.ByCenterPointRadius(
    CoordinateSystem.Identity().Origin, 6);

tool = Sphere.ByCenterPointRadius(
    CoordinateSystem.Identity().Origin.Translate(10, 0,
    0), 6);

result = s.Intersect(tool);
```

Генераторы точек Python

Генераторы точек Python

Приведенные ниже сценарии Python позволяют создать массивы точек, которые будут использованы в нескольких примерах. Их требуется вставить в узел Python Script следующим образом:



python_points_1

```
out_points = []

for i in range(11):
    sub_points = []
    for j in range(11):
        z = 0
        if (i == 5 and j == 5):
            z = 1
        elif (i == 8 and j == 2):
            z = 1
        sub_points.append(Point.ByCoordinates(i, j, z))
    out_points.append(sub_points)

OUT = out_points
```

python_points_2

```
out_points = []

for i in range(11):
    z = 0
    if (i == 2):
        z = 1
    out_points.append(Point.ByCoordinates(i, 0, z))

OUT = out_points
```

python_points_3

```
out_points = []

for i in range(11):
    z = 0
    if (i == 7):
```

```
        z = -1
    out_points.append(Point.ByCoordinates(i, 5, z))

OUT = out_points
```

python_points_4

```
out_points = []

for i in range(11):
    z = 0
    if (i == 5):
        z = 1
    out_points.append(Point.ByCoordinates(i, 10, z))

OUT = out_points
```

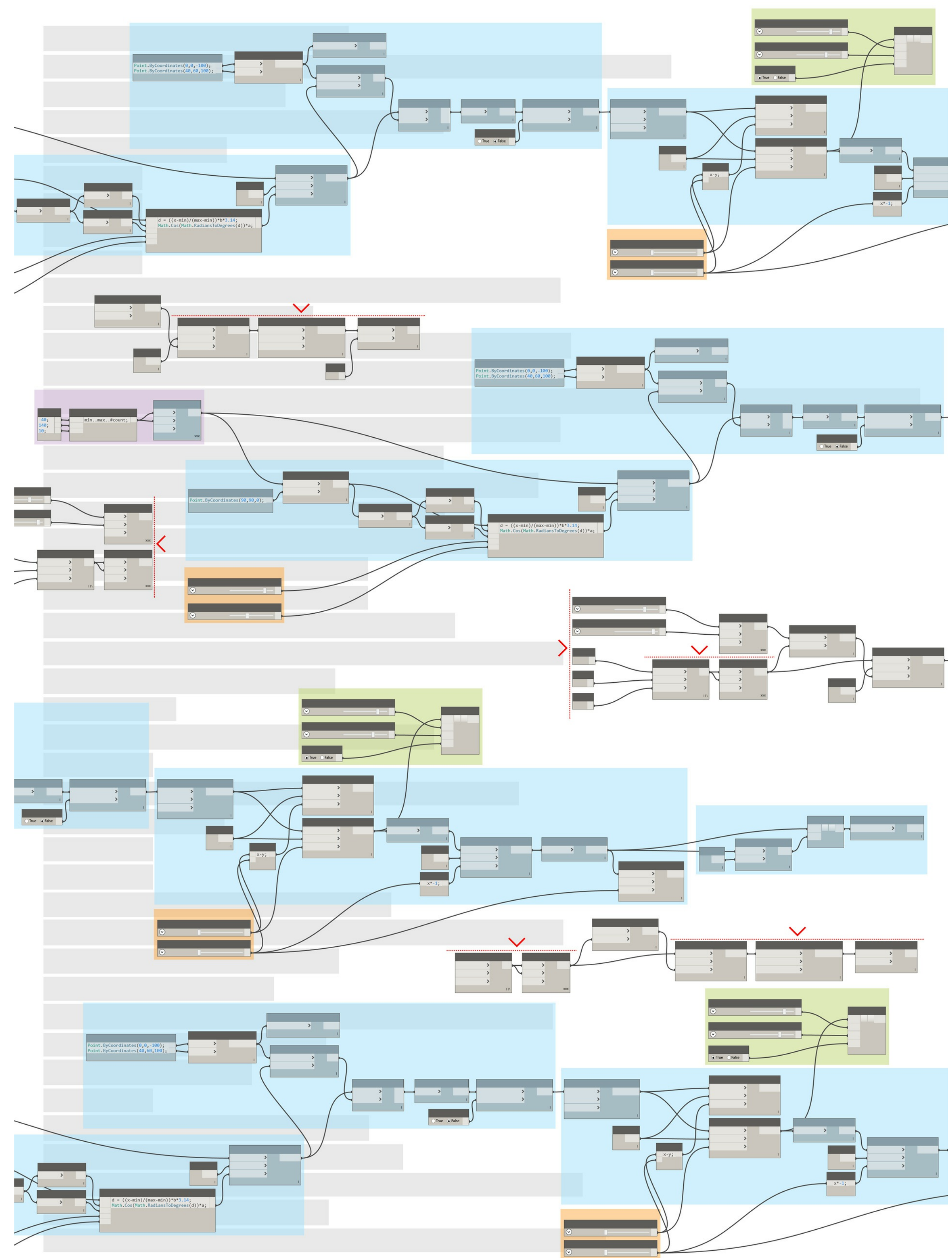
python_points_5

```
out_points = []

for i in range(11):
    sub_points = []
    for j in range(11):
        z = 0
        if (i == 1 and j == 1):
            z = 2
        elif (i == 8 and j == 1):
            z = 2
        elif (i == 2 and j == 6):
            z = 2
        sub_points.append(Point.ByCoordinates(i, j, z))
    out_points.append(sub_points)

OUT = out_points
```

Рекомендуемые практические приемы

Практические рекомендации

Этот раздел данного руководства является своеобразным сборником полезных советов. В нем рассказывается о разных стратегиях, разработанных на основе опыта и результатов исследований и позволяющих повысить качество параметрических рабочих процессов. Как проектировщики и программисты, мы измеряем качество наших инструментов их стабильностью, надежностью, удобством и эффективностью. В этом разделе вы найдете отдельные примеры для визуальных и текстовых сценариев, однако основополагающие принципы являются универсальными для всех сред программирования и могут использоваться в самых разных вычислительных рабочих процессах.

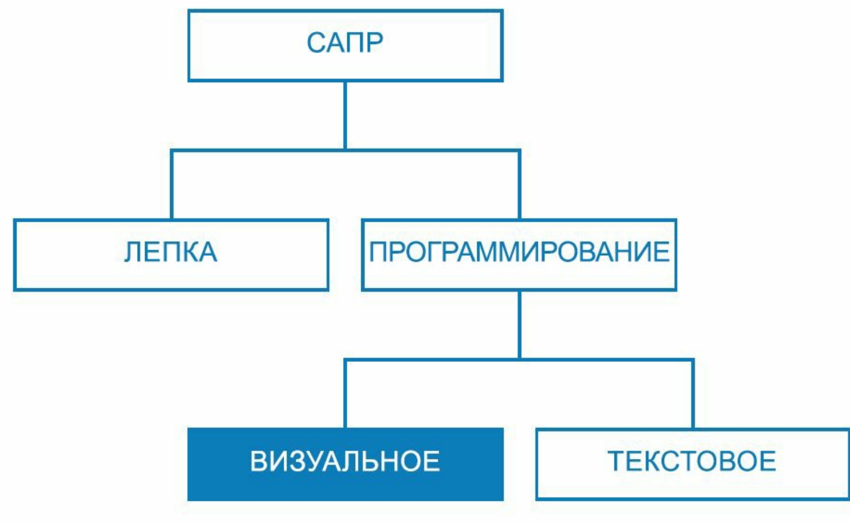


Point.ByCoordinates(0,0,-100);
Point.ByCoordinates(40,60,100);
True False
True False
x-y;
x*-1;
d = ((x-min)/(max-min))*b*3.14;
Math.Cos(Math.RadiansToDegrees(d))*a;
Point.ByCoordinates(0,0,-100);
Point.ByCoordinates(40,60,100);
-40;
140;
10;
min..max..#count;
True False
Point.ByCoordinates(90,90,0);
d = ((x-min)/(max-min))*b*3.14;
Math.Cos(Math.RadiansToDegrees(d))*a;
True False
True False
x-y;
x*-1;
Point.ByCoordinates(0,0,-100);
Point.ByCoordinates(40,60,100);
True False
True False
x-y;
x*-1;
d = ((x-min)/(max-min))*b*3.14;
Math.Cos(Math.RadiansToDegrees(d))*a;

Методы создания графиков

Методы создания графиков

В предыдущих главах было описано, как пользоваться мощными возможностями создания визуальных сценариев в Dynamo. Понимание этих возможностей является основой и первым шагом к созданию надежных визуальных программ. При работе с визуальными программами в полевых условиях, обмене ими с коллегами, устранении неполадок или проверке ограничений приходится сталкиваться и с другими проблемами. Если программа рассчитана на другого пользователя или предполагается открыть ее только через полгода, она должна обладать абсолютно понятной графикой и логикой. В Dynamo есть множество инструментов для работы со сложными программами. В этой главе приводятся рекомендации по их своевременному использованию.

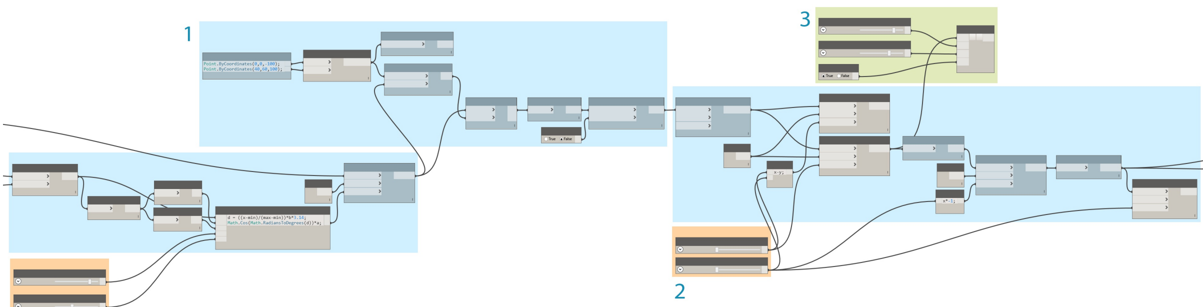


Упрощение

По мере разработки графика Dynamo и проверки различных идей он увеличивается в размере и становится сложнее. Несомненно, очень важно создать работающую программу, однако столь же важно сделать ее максимально простой. Благодаря этому работа графика будет более быстрой и предсказуемой, а пользователи вместе с разработчиком смогут понять его логику по прошествии времени. Ниже представлены варианты того, как можно упорядочить логику графиков.

Модульная организация за счет групп

- Благодаря группам можно **создавать функционально автономные части** при разработке программы.
- Группы позволяют **перемещать крупные части программы**, соблюдая при этом модульность и выравнивание.
- Можно менять **цвета групп, чтобы различать** их предназначение (входные данные или функции).
- С помощью групп можно **структурировать график таким образом, чтобы упростить создание пользовательских узлов**.



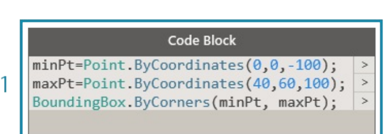
Цвета в этой программе обозначают назначение каждой группы. Этот метод может использоваться для создания иерархии в любых разрабатываемых графических стандартах или шаблонах.

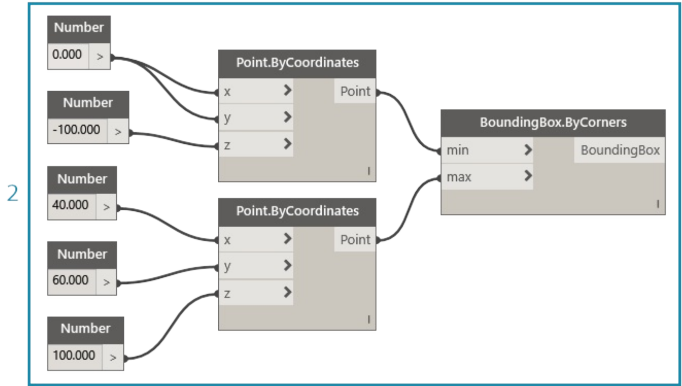
1. Группа функций (синий)
2. Группа входных данных (оранжевый)
3. Группа сценариев (зеленый) Сведения об использовании групп см. в разделе Управление программой.

Эффективная разработка с помощью блоков кода

- Иногда с помощью блока кода можно **быстрее ввести число или метод узла, чем при поиске** (Point.ByCoordinates, Number, String, Formula).

- Блоки кода можно использовать **для настройки пользовательских функций в DesignScript, уменьшающих количество узлов в графике**.





Примеры 1 и 2 выполняют одну и ту же функцию. Получилось значительно быстрее написать несколько строк кода, чем прибегать к функции поиска и добавлять каждый узел по отдельности. Кроме того, блок кода значительно меньше по объему.

1. Сценарий DesignScript, написанный с помощью блока кода
2. Аналогичная программа с использованием узлов Сведения об использовании блоков кода см. в разделе Определение блока кода.

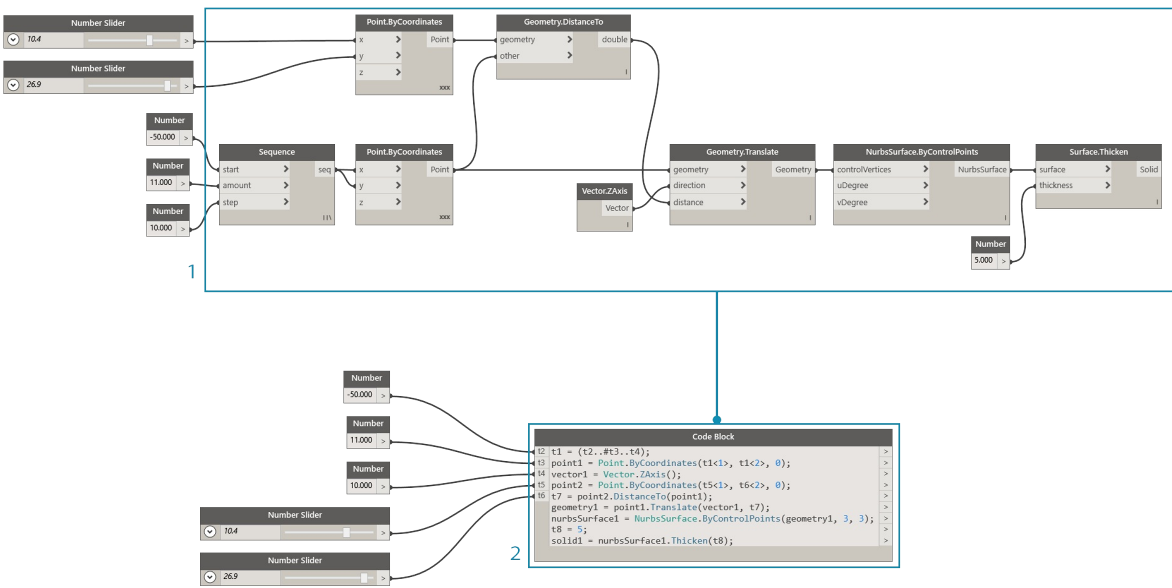
Сжатие узла в код

- **Сложность графика можно уменьшить с помощью преобразования узла в код (Node to Code)**. При этом для набора простых узлов будет создан соответствующий сценарий DesignScript, состоящий из одного блока кода.
- Функция Node to Code позволяет **сжать код, не усложнив восприятие программы**.
- Далее перечислены **преимущества** использования функции Node to Code.
- Простое сжатие кода в один редактируемый компонент.
- Упрощение значительной части графика.
- Удобство применения к мини-программам, которые редко редактируются.

- Возможность встраивания других типов блоков кода, таких как функции.

- Ниже представлены **недостатки** использования функции Node to Code.

- Типовые имена ухудшают удобочитаемость.
- Сложность восприятия для других пользователей.
- Нет простого способа вернуться к визуальной версии программы.

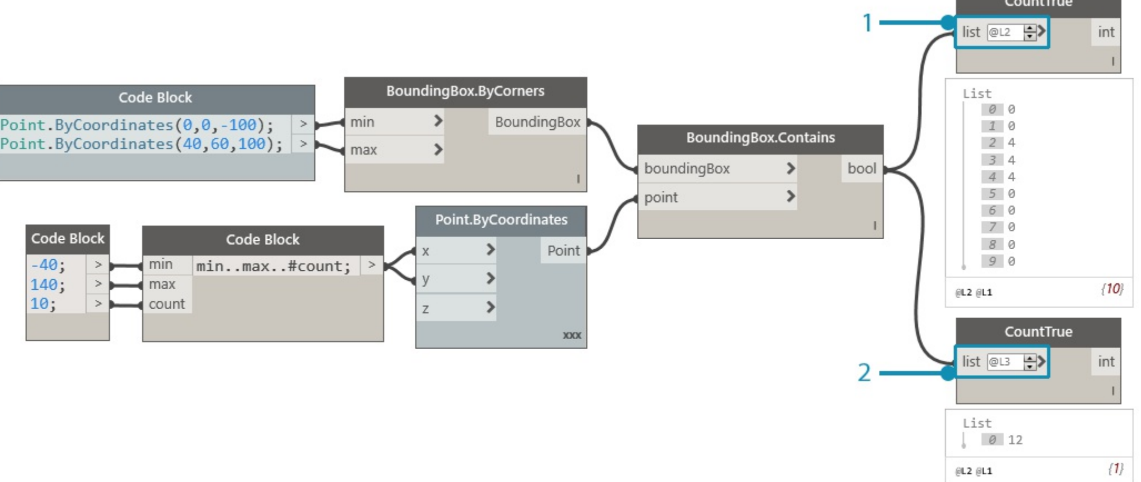


1. Существующая программа
2. Блок кода, созданный с помощью функции Node to Code Сведения об использовании функции Node to Code см. в разделе Синтаксис DesignScript.

Гибкий доступ к данным с помощью функции List@Level

- С помощью функции List@Level можно **упростить график, заменив узлы List.Map и List.Combine**, которые могут занимать значительное место рабочей области.

- При построении логики узла List@Level работает **быстрее, чем List.Map/List.Combine**, так как предоставляет доступ к данным любого уровня в списке непосредственно с входного порта узла.



Активировав функцию List@Level для входных данных списка CountTrue, можно проверить, сколько истинных значений и в каких списках возвращает функция BoundingBox.Contains. List@Level позволяет определить, с какого уровня данные будут подаваться на ввод. Работа с List@Level отличается гибкостью, эффективностью и более предпочтительна по сравнению с другими методами, где используются функции List.Map и List.Combine.

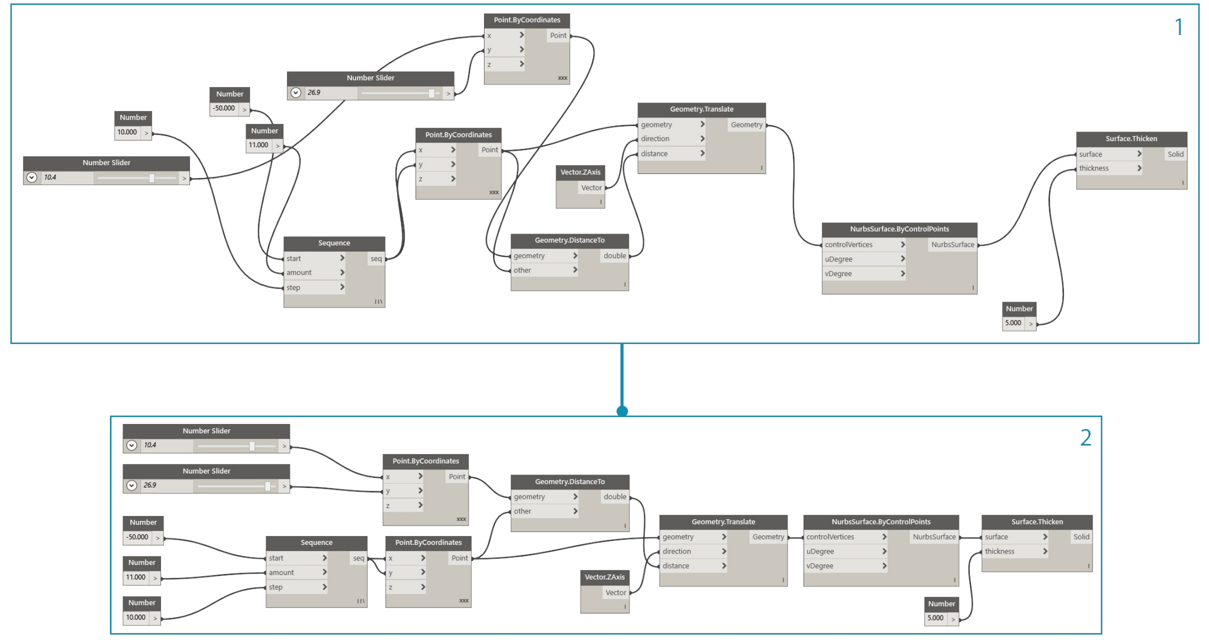
1. Подсчет истинных значений на 2 уровне списка.
2. Подсчет истинных значений на 3 уровне списка. Сведения о работе с функцией List@Level см. в разделе Списки списков.

Обеспечение наглядности

Помимо простоты и эффективности графиков, необходимо позаботится об их максимальной наглядности. Несмотря на все усилия по созданию интуитивного графика за счет логических группировок, взаимосвязи могут быть видны недостаточно хорошо. Лишних поисков и неопределенности можно избежать благодаря простому примечанию внутри группы или переименованному регулятору. Описанные ниже способы помогут обеспечить визуальное единообразие в одном или нескольких графиках.

Достижение визуальной целостности посредством выравнивания узлов

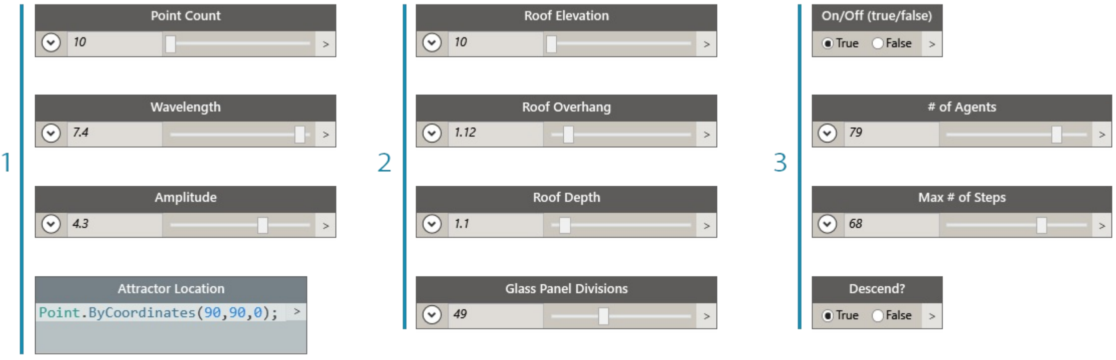
- Чтобы уменьшить количество доработок после построения графика, попытайтесь сделать компоновку узлов удобочитаемой, **периодически выравнивая их**.
- Если с графиком будут работать другие пользователи, **убедитесь, что компоновка проводов и узлов имеет четкую логику**.
- Для упрощения выравнивания **используйте функцию «Очистить компоновку узла», чтобы автоматически выровнять** график (однако в таком случае точность будет меньше, чем при выравнивании вручную).



1. Неупорядоченный график
2. Выровненный график Сведения об использовании функции выравнивания узлов см. в разделе Управление программой.

Использование описательных меток при переименовании

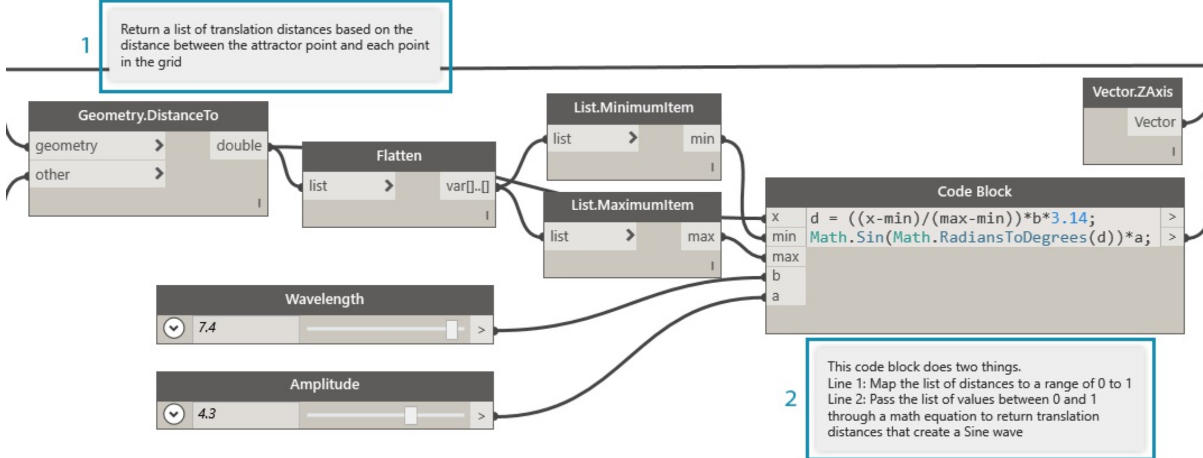
- Переименование входных данных сделает график более понятным другим пользователям, **особенно если требуется подсоединиться к узлу, который не будет виден на экране**.
- **При переименовании любых узлов, кроме узлов входных данных, будьте максимально осторожны.** Можно также создать пользовательский узел из кластера узлов и переименовать его: при этом будет понятно, что в нем содержится нечто другое.



1. Входные данные для управления поверхностью
2. Входные данные архитектурных параметров
3. Входные данные в сценарии моделирования водоспуска Чтобы переименовать узел, щелкните его имя правой кнопкой мыши и выберите «Переименовать узел...».

Разъяснения в примечаниях

- Примечание добавляется, если для какой-либо части **графика требуется пояснение на простом языке**, которое не может быть выражено с помощью узла.
- Примечание добавляется, если набор **узлов или группа имеют слишком большой размер, сложную структуру или логику**.



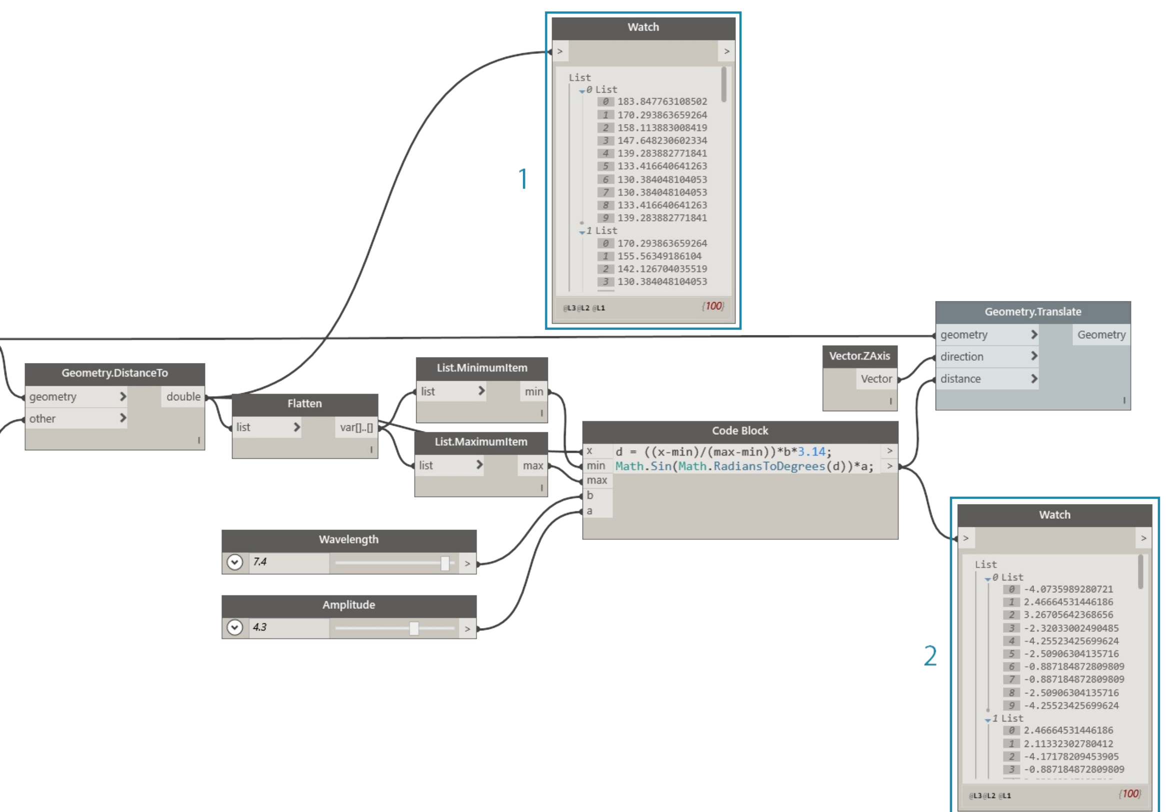
1. Примечание, описывающее часть программы, которая возвращает примерные расстояния переноса.
2. Примечание, описывающее код, который сопоставляет эти значения с синусоидальной волной. Сведения о том, как добавить примечание, см. в разделе Управление программой.

Зондирование на всех этапах работы

При создании визуального сценария важно убедиться в том, что возвращаемые результаты соответствуют ожидаемым. Не все ошибки или проблемы ведут к немедленному сбою в работе программы. Особенно это касается нулевых значений, которые могут повлиять на работу значительно позже. Этот метод также применяется к текстовым сценариям. См. раздел Методы создания сценариев. Следующие рекомендации помогут получить ожидаемые результаты.

Мониторинг данных с помощью марок наблюдения (Watch) и предварительного просмотра (Preview)

- При создании программы используйте марки Watch и Preview, чтобы **убедиться в правильности ключевых выходных данных**.



Узлы Watch используются для сравнения следующих данных:

1. примерные расстояния переноса;
2. значения, проходящие через уравнение синусоиды. Инструкции по использованию узла Watch см. в разделе Библиотека.

Повторное использование

Весьма вероятно, что рано или поздно вашу программу откроет другой пользователь, даже если вы работаете самостоятельно. На основе входных и выходных данных этому пользователю нужно будет быстро понять, что требуется для работы программы и каковы результаты этой работы. Это особенно важно при разработке пользовательских узлов, которые будут применяться сообществом Dynamo и добавляться в программы других разработчиков. Следующие рекомендации помогут создавать надежные, многократно используемые программы и узлы.

Управление вводом/выводом

- Чтобы обеспечить удобочитаемость и масштабируемость, попробуйте **минимизировать входные и выходные данные**.
- Перед тем, как добавить первый узел в рабочую область, необходимо **определить метод построения логики, создав примерный план** ее действия. При создании примерного плана отслеживайте, какие входные и выходные данные войдут в сценарии.

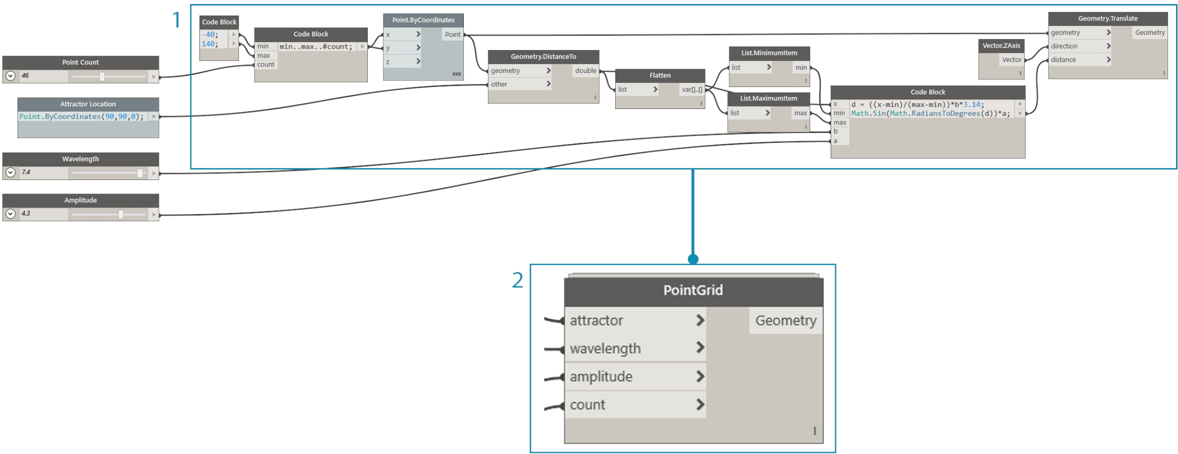
Использование наборов параметров при добавлении входных значений

- При наличии **определенных вариантов или условий, которые требуется включить в график**, для быстрого доступа к ним следует использовать наборы параметров.
- Наборы параметров можно также использовать для **уменьшения сложности путем кэширования определенных значений регулятора** на графике с длительным временем выполнения.

Сведения об использовании наборов параметров см. в разделе Управление данными с помощью наборов параметров.

Упаковка программ с пользовательскими узлами в контейнеры

- Пользовательский узел применяется, если **можно объединить программу в одном контейнере**.
- Еще пользовательский узел применяется, если **часть графика будет повторно использоваться** в других программах.
- И, наконец, пользовательский узел применяется, если **необходимо сделать функцию доступной сообществу Dynamo**.

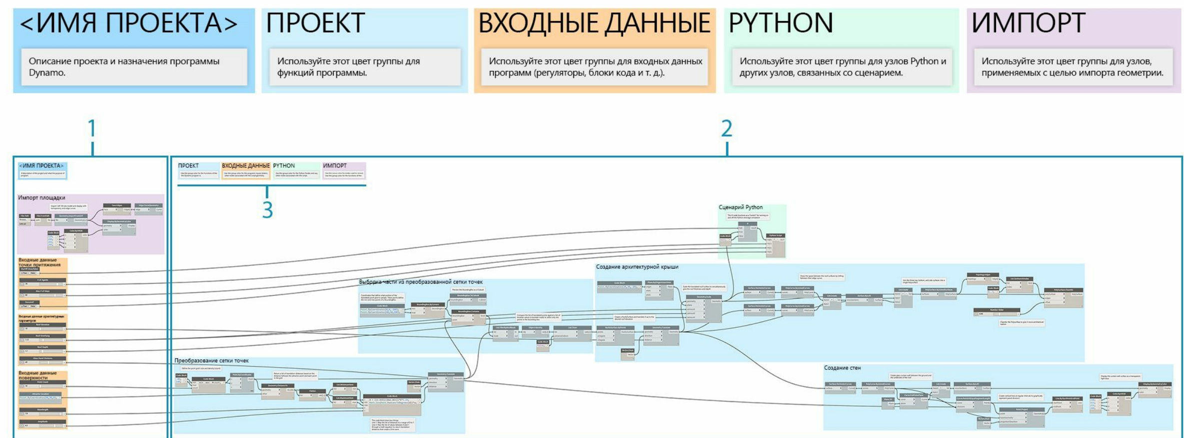


Если собрать программу преобразования точек в пользовательский узел, получится более надежная и оригинальная переносная программа, в которой легко разобраться. Правильно обозначенные порты ввода помогут другим пользователям понять, как применять узел. Не забудьте добавить описания и требуемые типы данных для каждого входного элемента.

1. Существующая программа точки притяжения
2. Пользовательский узел для размещения программы, PointGrid Сведения о работе с пользовательскими узлами см. в разделе Введение в пользовательские узлы.

Создание шаблонов

- Шаблоны используются в качестве **визуальных стандартов, чтобы обеспечить общий подход к построению графиков среди пользователей, осуществляющих совместную работу**.
- При создании шаблона можно стандартизировать **цвета и размеры шрифтов группы**, чтобы отнести типы рабочих процессов или операции с данными к определенным категориям.
- При создании шаблона можно даже стандартизировать **метки, цвета или стили для обозначения различий между внешними и внутренними рабочими процессами** на графике.

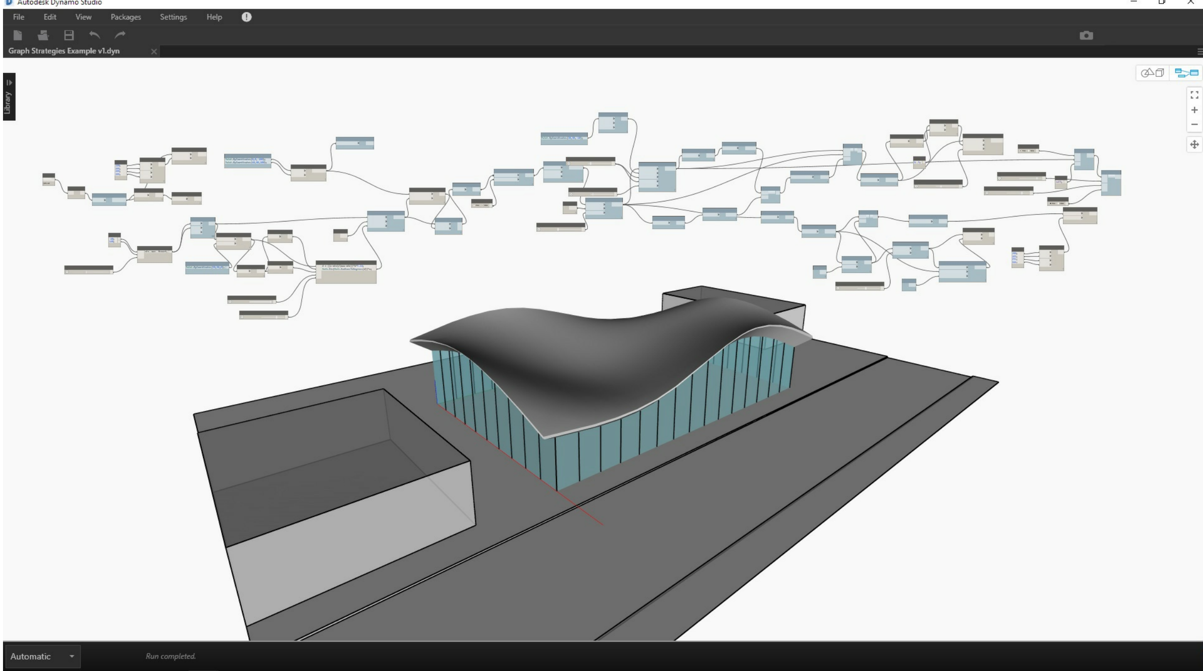


1. Пользовательский интерфейс (внешняя часть программы). Включает в себя имя проекта, регуляторы ввода и импортируемую геометрию.
2. Внутренняя часть программы.
3. Цветовые категории групп (проект в целом, входные данные, сценарии на языке Python, импортированная геометрия).

Упражнение «Архитектурная крыша»

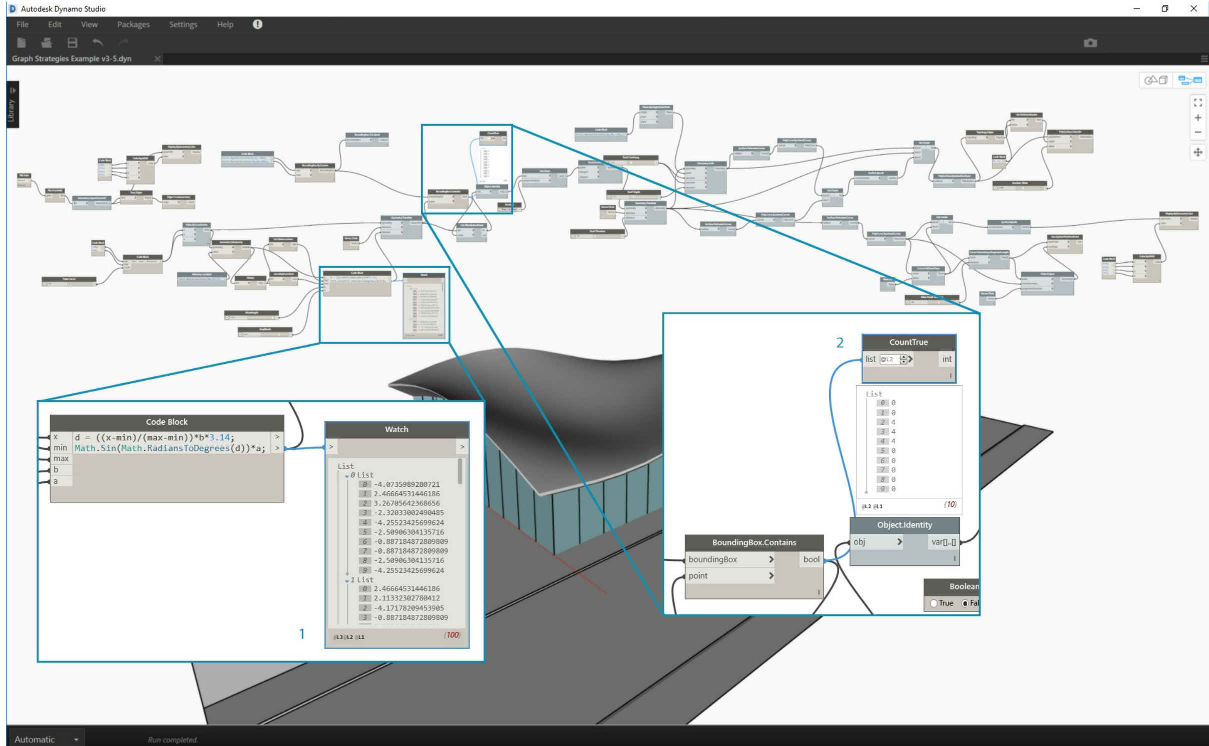
Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. RoofDrainageSim.zip

Ознакомившись с некоторыми практическими советами, попробуйте применить их к быстро составленной программе. Несмотря на то, что программа успешно создает крышу, график отражает «поток сознания» автора. Отсутствует какая-либо структура и руководство по использованию. Применив практические советы по организации, описанию и анализу программы, мы поможем понять другим пользователям, как ее использовать.



Программа работает, но график не структурирован.

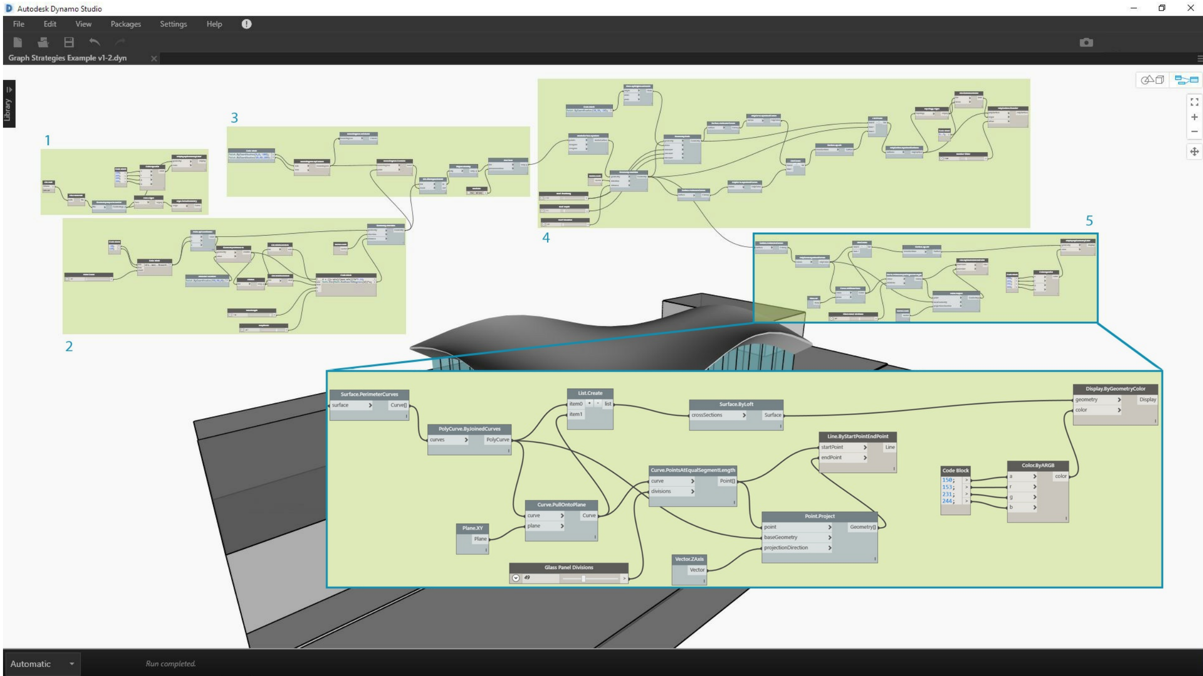
Начните с определения данных и геометрии, возвращаемых программой.



Чтобы создать логические разделы или модули, очень важно понимать, когда данные подвергаются наибольшим изменениям. Перед переходом к следующему шагу попробуйте проверить остальную часть программы с помощью узлов Watch, чтобы убедиться в возможности определить группы.

1. Этот блок кода с математическим уравнением выглядит как ключевая часть программы. Узел Watch указывает, что он возвращает списки расстояний переноса.
2. Назначение этой области не очевидно. Расположение истинных значений на уровне списка L2 из BoundingBox.Contains и наличие List.FilterByBoolMask говорит о том, что это выборка части из сетки точек.

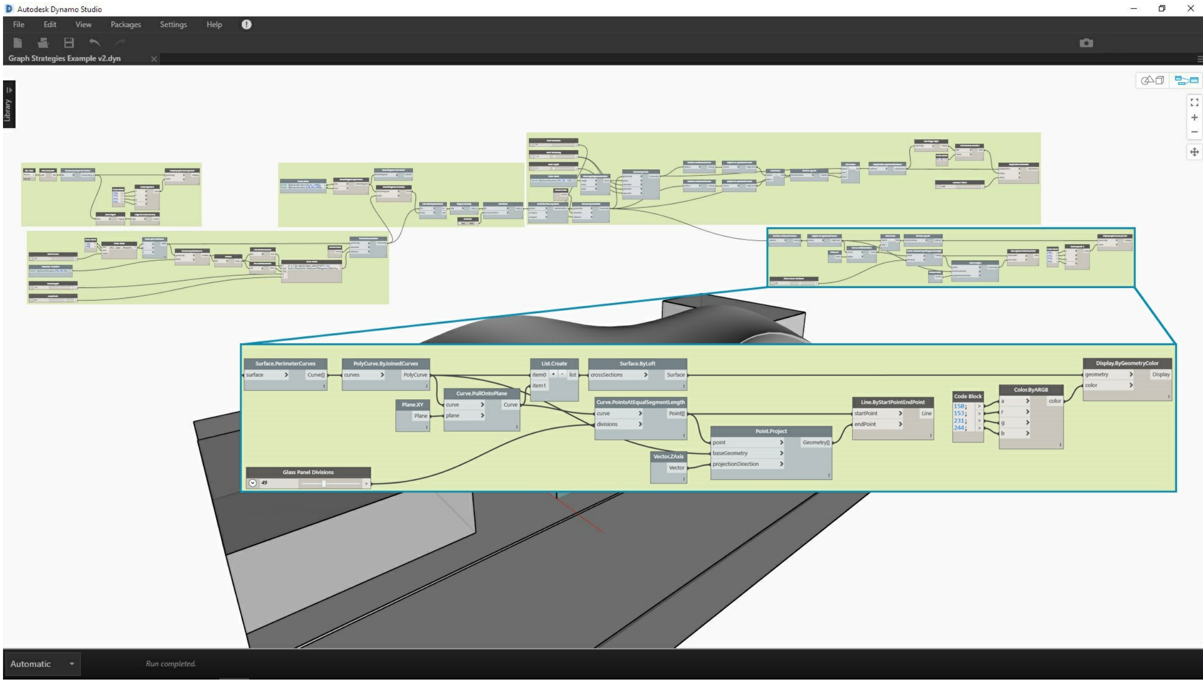
Разобравшись в компонентах программы, разделите их на группы.



Группы позволяют визуально дифференцировать компоненты программы.

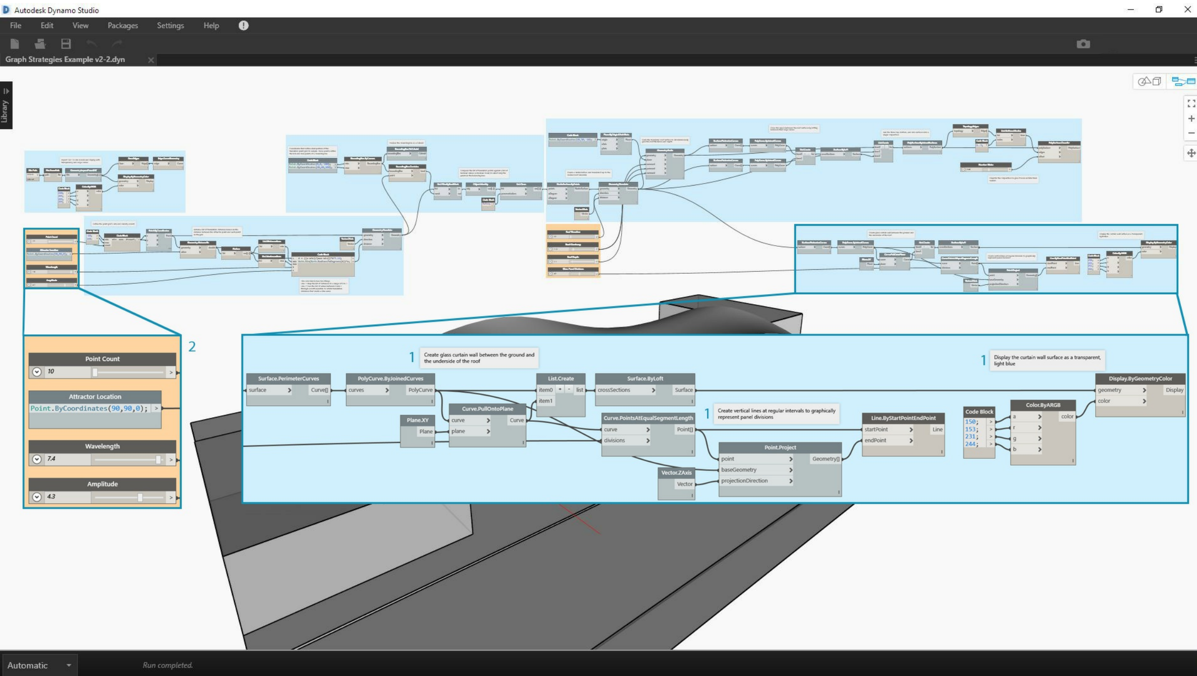
1. Импорт 3D-модели площадки
2. Преобразование сетки точек на основе уравнения синусоиды
3. Выборка части из сетки точек
4. Создание поверхности архитектурной крыши
5. Создание стеклянного витража

После определения групп выровняйте узлы, чтобы обеспечить визуальную целостность графика.



Визуальная целостность позволяет видеть ход выполнения программы и скрытые взаимосвязи между узлами.

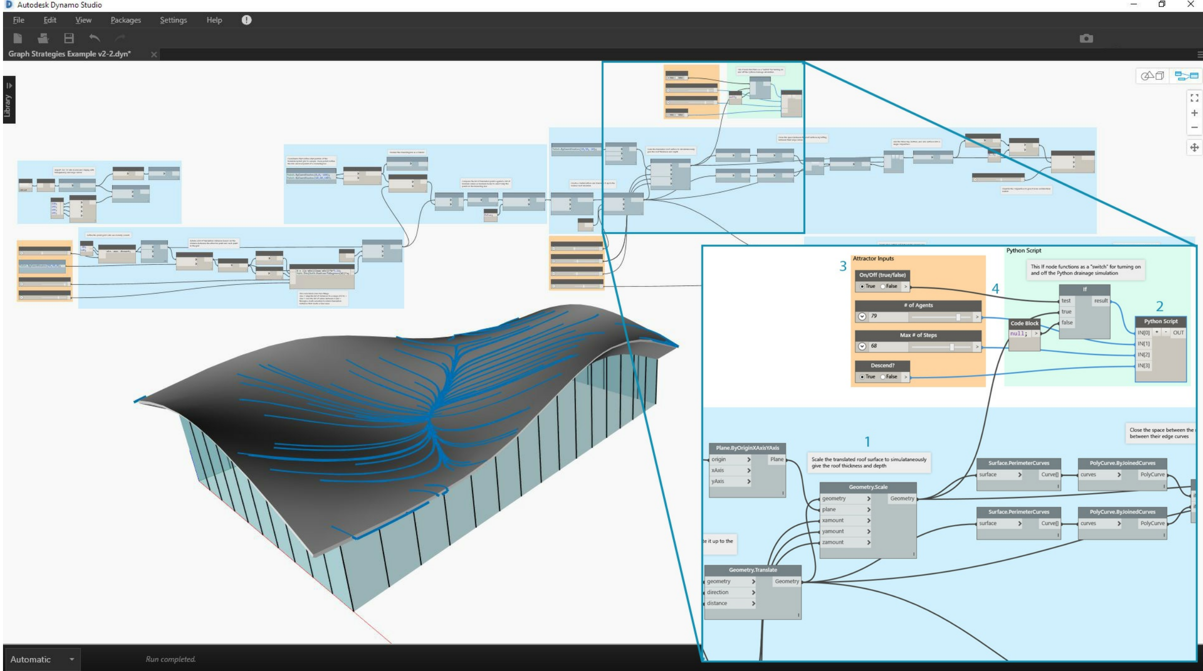
Сделайте программу более понятной, добавив еще один слой улучшений графического интерфейса. Добавьте примечания, чтобы пояснить, как работает та или иная часть программы, укажите пользовательские имена входных данных и назначьте цвета различным типам групп.



Благодаря этим улучшениям графического интерфейса пользователи лучше поймут назначение этой программы. Различные цвета групп позволяют отличать входные данные от функций.

1. Примечания
2. Входные данные с описательными именами

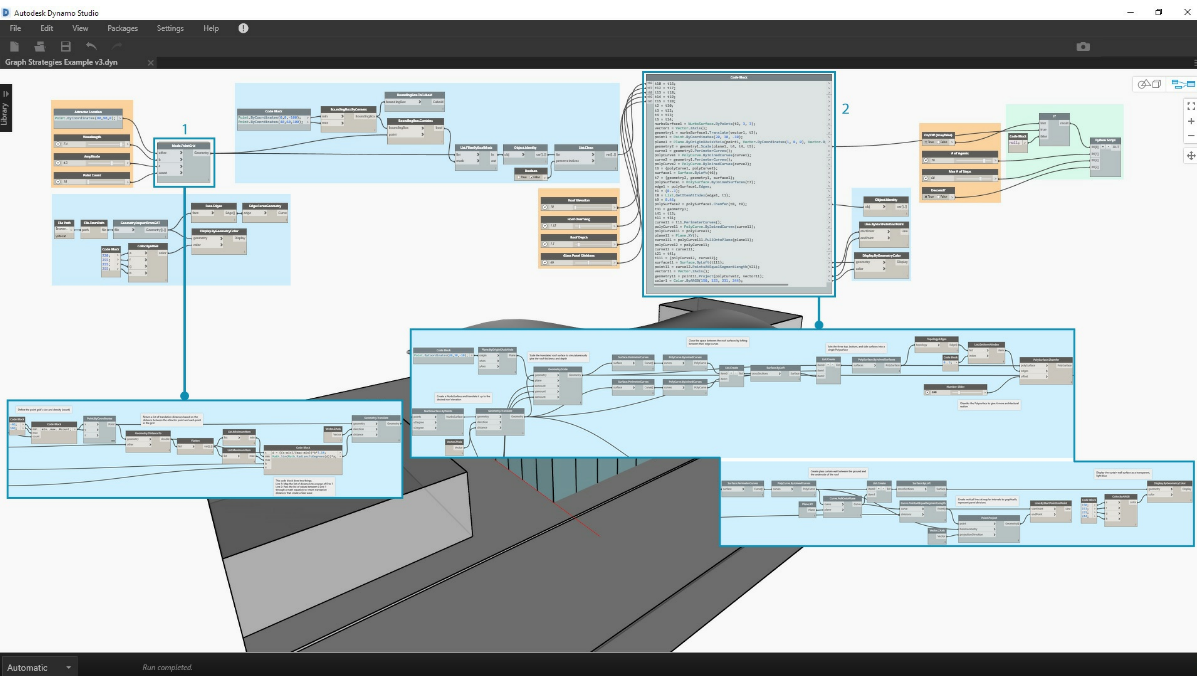
Перед тем как приступить к сжатию программы, определим предполагаемое место вставки сценария Python для моделирования водоспуска. Разместите выходные данные первой масштабированной поверхности крыши в соответствующих входных данных сценария.



Сценарий встраивается в эту часть программы, чтобы можно было запустить моделирование водоспуска на одной исходной поверхности крыши. Данная поверхность не отображается в области предварительного просмотра, но при этом не нужно выбирать верхнюю поверхность на сложной поверхности с фаской.

1. Исходная геометрия для входных данных сценария
2. Узел Python
3. Регуляторы ввода
4. Переключатель вкл./откл.

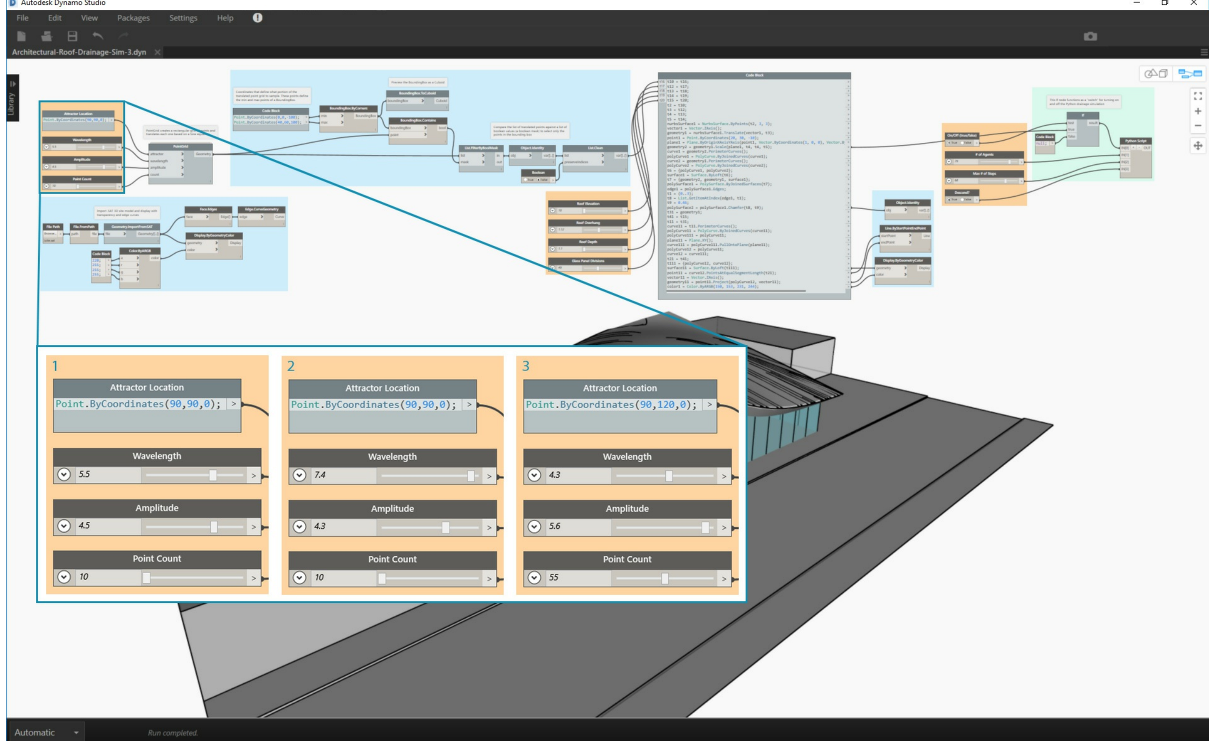
Упростите график, чтобы расставить все по местам.



Сжатие программы с помощью функций Node to Code и Custom Node привело к значительному уменьшению размера графика. Группы, отвечающие за создание поверхности крыши и стен, преобразованы в код, так как характерны только для данной программы. Группа преобразования точек содержится в пользовательском узле, так как ее можно использовать в другой программе. Создайте в файле примера собственный пользовательский узел из группы преобразования точек.

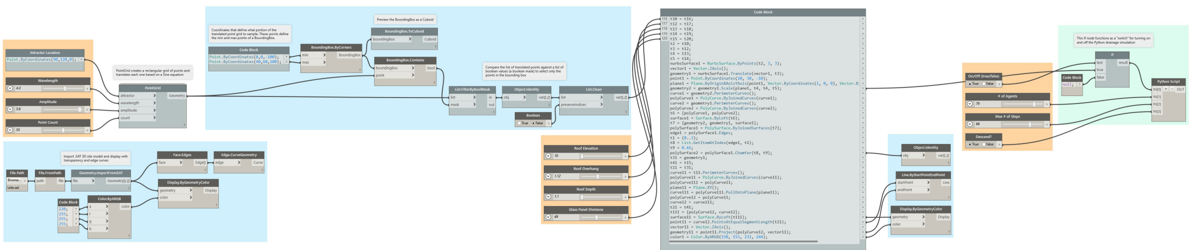
1. Пользовательский узел для размещения группы «преобразование сетки точек»
2. Функция Node to Code для сжатия групп «Создание поверхности архитектурной крыши и виража»

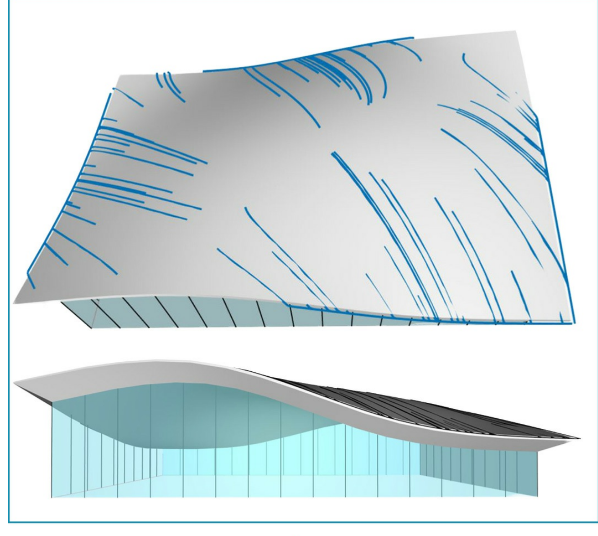
В завершение создайте наборы параметров для образцов формы крыши.

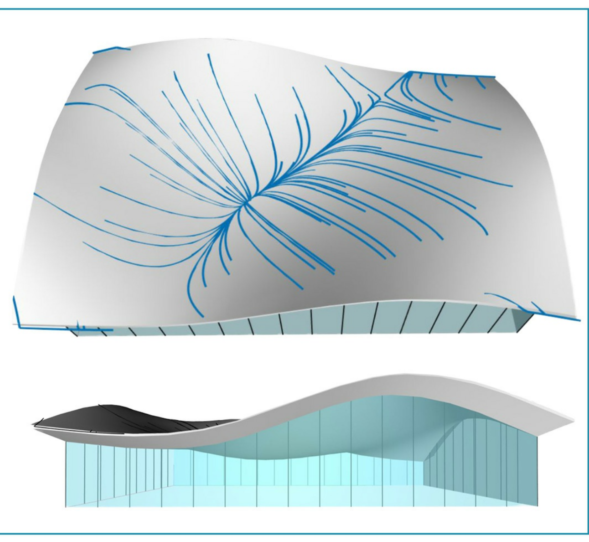


Эти входные данные в существенной мере определяют форму крыши и помогут пользователям увидеть возможности программы.

Программа, где видны два набора параметров.







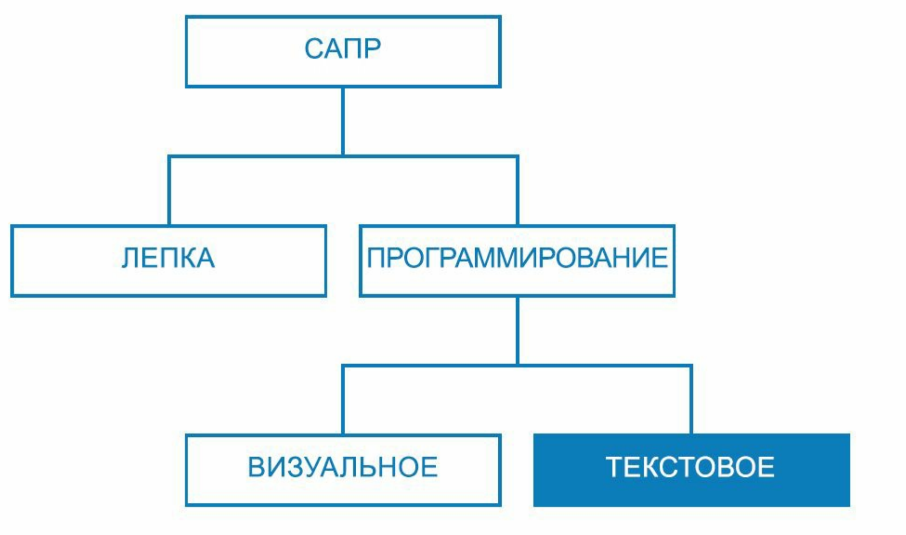
1 2

Аналитический вид наборов параметров, соответствующих образцам водостока крыши.

Методы создания сценариев

Методы создания сценариев

Создание сценариев на основе текста в среде разработки визуальных сценариев обеспечивает возможность построения эффективных наглядных взаимосвязей с использованием языков DesignScript, Python и ZeroTouch (C#). В DesignScript можно отображать такие элементы, как регуляторы ввода, и сжимать сложные операции. В том же рабочем пространстве с помощью Python или C# можно получить доступ к мощным инструментам и библиотекам. При грамотном использовании сочетание этих методов может обеспечить высокую степень адаптированности, ясности и эффективности всей программы. Ниже приводится набор рекомендаций по дополнению визуальных сценариев текстовыми.



Своевременное использование сценариев

Текстовые сценарии позволяют устанавливать более сложные взаимосвязи, чем визуальное программирование, хотя их возможности во многом пересекаются. Это следует учитывать, так как узлы представляют собой эффективные пакеты кода, что позволяет написать целую программу для Dynamo в DesignScript или Python. Однако визуальные сценарии используются по причине того, что интерфейс, состоящий из узлов и проводов, позволяет создать интуитивно понятный поток графической информации. Информация о том, в каких случаях возможности текстовых сценариев превосходят возможности визуальных сценариев, — ключ к их использованию без отказа от работы с интуитивно понятными узлами и проводами. Ниже приводятся рекомендации в отношении того, когда следует использовать сценарии и какой при этом выбрать язык программирования.

Текстовые сценарии используются в следующих случаях:

- организация циклов;
- использование рекурсии;
- доступ к внешним библиотекам.

Выбор языка

	Циклы	**Рекурсия**	**Сжатие узлов**	**Внешн. библиотеки**	**Сокращение**
DesignScript	Да	Да	Да	Нет	Да
Python	Да	Да	Частично	Да	Нет
ZeroTouch (C#)	Нет	Нет	Нет	Да	Нет

Список ресурсов, доступных в библиотеках Dynamo, см. в Справке по сценариям.

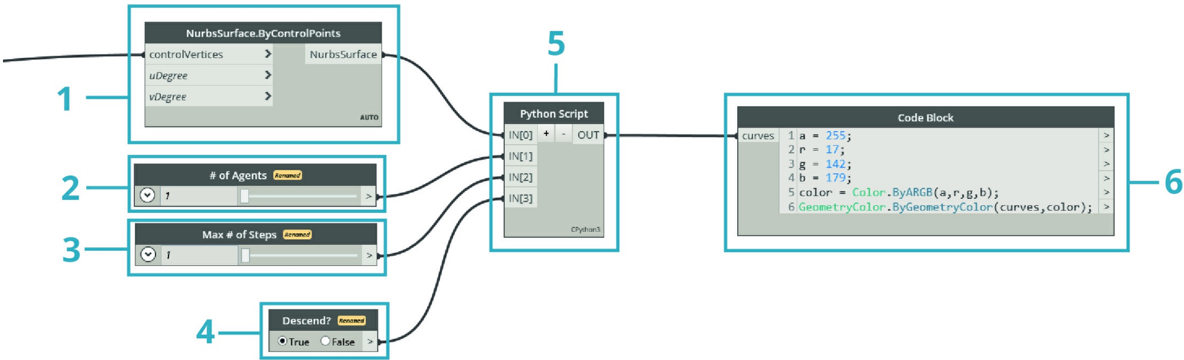
Параметрическое мышление

При написании сценариев в Dynamo — параметрической среде — имеет смысл структурировать код с учетом системы узлов и проводов, в которой он будет размещен. Рассматривайте узел, содержащий текстовый сценарий, как любой другой узел в программе с некоторыми определенными входными данными, функцией и ожидаемыми выходными данными. При этом код внутри узла получает небольшой набор переменных, на основе которых можно строить работу, а это — ключ к безупречной параметрической системе. Далее приводятся некоторые рекомендации, позволяющие успешнее встроить код в визуальную программу.

Определите внешние переменные.

- Попробуйте определить заданные параметры в проектной задаче, чтобы можно было построить модель непосредственно на этих данных.
- Перед написанием кода необходимо указать следующие переменные:
  - минимальный набор входных данных;

- ожидаемые выходные данные;
- константы.

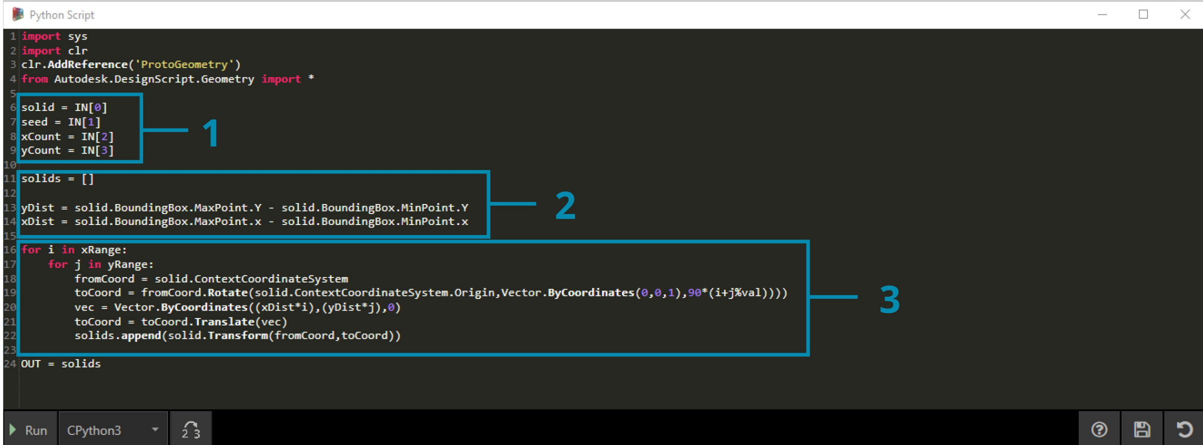


Перед написанием кода было определено несколько переменных.

1. Поверхность, на которую будет моделироваться выпадение осадков.
2. Желаемое количество капель дождя (агентов).
3. Расстояние перемещения капель дождя.
4. Переключение между режимом спуска по траектории с наибольшей крутизной и режимом пересечения поверхности.
5. Узел Python с соответствующим количеством входных данных.
6. Блок кода для окрашивания возвращаемых кривых синим цветом.

Разработайте внутренние взаимосвязи.

- Параметрическая архитектура позволяет редактировать определенные параметры или переменные для настройки или изменения конечного результата уравнения или работы системы.
- Если объекты в сценарии логически связаны между собой, следует указать, что один является функцией другого и наоборот. Таким образом, при изменении одного объекта другой также будет обновлен.
- Сократите количество входных данных, оставив только ключевые параметры.
  - Если набор параметров может быть сформирован из дополнительных родительских параметров, в качестве входных данных сценария оставьте только эти родительские параметры. Это повышает удобство работы со сценарием за счет некоторого упрощения его интерфейса.



«Модули» кода из примера в узле Python.

1. Входные данные.
2. Внутренние переменные сценария.
3. Цикл, для выполнения функции которого используются эти входные данные и переменные. Совет. Уделите столько же внимания процессу, сколько и самому решению.

Не повторяйтесь (следуйте принципу DRY — Don't repeat yourself).

- Если один и тот же фрагмент в сценарии выражается несколькими способами, рано или поздно произойдет десинхронизация этих совпадений, что потребует значительных исправлений, приведет к ошибкам в расчетах и внутренним противоречиям.

- Принцип DRY звучит следующим образом: «Информация, вводимая в компьютер должна быть конкретной и однозначной».

    - При успешном применении этого принципа все взаимосвязанные элементы в сценарии будут изменяться предсказуемо и единообразно, а все несвязанные элементы не будут иметь логических последствий друг для друга.

```
### BAD
for i in range(4):
for j in range(4):
point = Point.ByCoordinates(3*i, 3*j, 0)
points.append(point)

### GOOD
count = IN[0]
pDist = IN[1]

for i in range(count):
for j in range(count):
point = Point.ByCoordinates(pDist*i, pDist*j, 0)
points.append(point)
```

Совет. Перед дублированием объектов в сценарии (например, константы в примере выше) подумайте, можно ли вместо этого установить связь с источником.

Модульная структура

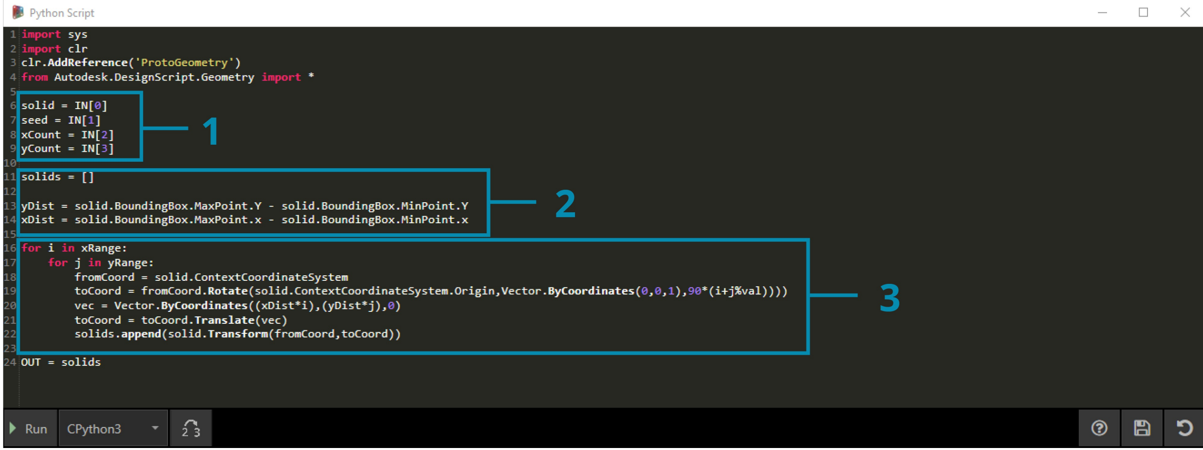
По мере расширения и усложнения кода основная идея (ключевой алгоритм) становится все менее и менее читаемой. При этом все сложнее отслеживать, что и где может произойти, выявлять ошибки при возникновении проблем, встраивать другой код и назначать задачи по разработке. Чтобы избежать этих сложностей, для написания кода рекомендуется использовать модули. При этой организационной методике код разбивается на части в зависимости от выполняемой задачи. Далее представлены некоторые советы по расширению возможностей управления сценариями благодаря модульности.

Записывайте код в виде модулей.

- «Модуль» — это группа данных кода, выполняющая определенную задачу (аналогично узлу Dynamo в рабочем пространстве).

- Это может быть любой объект, который визуально отделен от близлежащего кода (функция, класс, группа входных данных или импортируемые библиотеки).

- Разработка кода в форме модулей повышает визуальное и интуитивное качество узлов, позволяя строить сложные взаимосвязи, реализуемые только с помощью текстовых сценариев.



Эти циклы вызывают класс с именем agent (агент), который мы будем разрабатывать в упражнении.

1. Модуль кода, определяющий начальную точку каждого агента.
2. Модуль кода, обновляющий агента.
3. Модуль кода, который строит маршрут траектории агента.

Ищите повторяющийся код.

- Если выясняется, что код выполняет одно и то же (или почти одно и то же) действие в нескольких местах, рекомендуется кластеризовать его в функцию, доступную для вызова.

- Функции Manager управляют ходом выполнения программы и содержат вызовы функций Worker для обработки низкоуровневых задач, например для перемещения данных между конструкциями.

В этом примере создаются сферы с радиусами и цветом, зависящими от значения Z центров.

1. Имеется две родительские функции Worker: одна из них создает сферы с радиусами и отображает цвета в зависимости от значения Z центра.
2. В родительской функции Manager объединены две функции Worker. При ее вызове вызываются и обе находящиеся в ней функции.

Отображайте только нужные элементы.

- В интерфейсе модуля отображаются элементы, как предоставляемые самим модулем, так и необходимые ему.

- После того как интерфейсы между блоками определены, детальная разработка каждого блока может выполняться отдельно.

Разделимость и заменяемость.

- Модули никаким образом не зависимы друг от друга.

Основные формы модульной организации.

- Группировка кодов

```
# IMPORT LIBRARIES
import random
import math
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# DEFINE PARAMETER INPUTS
surfIn = IN[0]
maxSteps = IN[1]
```

- Функции

```
def get_step_size():
area = surfIn.Area
stepSize = math.sqrt(area)/100
return stepSize

stepSize = get_step_size()
```

- Классы

```
class MyClass:
i = 12345

def f(self):
return 'hello world'
```

```
numbers = MyClass.i
greeting = MyClass.f
```

Зондирование на всех этапах работы

При написании текстовых сценариев в Dynamo важно всегда быть уверенным, что создаваемое соответствует ожидаемому. Благодаря этому непредвиденные события — синтаксические ошибки, логические несоответствия, неточности значений, непредвиденные выходные данные и т. д. —можно быстро выявлять и устранять отдельно по мере появления, а не общей массой по завершении работы. Так как текстовые сценарии размещаются в узлах рабочей области, они автоматически встроены в поток данных визуальной программы. Благодаря этому последующий мониторинг сценария ограничивается лишь назначением данных для вывода, запуском программы и оценкой результатов, выводимых из сценария с помощью узла наблюдения (Watch). Ниже приводятся советы по непрерывной проверке сценариев в процессе их создания.

Проверка во время работы.

- Каждый раз по завершении работы над группой функций рекомендуется выполнять следующие действия.
    - Сделайте паузу и уделите время проверке кода.
    - Проявляйте критичность. Ваши коллеги смогут понять, для чего он предназначен? Эта функция действительно необходима? Можно ли повысить ее эффективность? Не создаю ли я лишних копий или зависимостей?
    - Проведите экспресс-проверку данных на целесообразность.
- В качестве выходных данных назначайте наиболее актуальную информацию, обрабатываемую в сценарии, чтобы при обновлении сценария узел всегда выводил релевантные данные.

Зондирование образца кода из узла Python.

1. Убедитесь, что все кромки тела возвращаются в виде кривых, в результате чего вокруг него создается ограничивающая рамка.
2. Проверьте, чтобы входные данные количества успешно преобразовывались в диапазоны.
3. Убедитесь, что системы координат в данном цикле правильно преобразованы и повернуты.

Учитывайте пограничные случаи.

- При написании сценариев присвойте входным параметрам минимальные и максимальные значения в отведенной им области, чтобы проверить, будет ли программа функционировать при экстремальных условиях.
- Даже если программа работает с предельными установками, проверьте, не возвращает ли она нежелательные нулевые или пустые значения.
- Иногда неисправности и ошибки, позволяющие обнаружить скрытую проблему в сценарии, проявляются только в таких пограничных случаях.
    - Определите, что вызывает ошибку, а затем решите, следует ли исправить ее изнутри или перенастроить всю область параметров, чтобы избежать проблемы.

Совет. Всегда исходите из того, что пользователь может выбрать любую комбинацию с любым доступным входным значением. Это поможет избежать неприятных сюрпризов.

Эффективная отладка

Отладкой называется процесс устранения ошибок в сценарии. Под ошибками подразумеваются неполадки, недоработки, неточности и любые нежелательные результаты. Иногда чтобы исправить ошибку, достаточно скорректировать неправильно написанное имя переменной. В других случаях речь может идти о более глобальной проблеме, связанной со структурой сценария. В идеале зондирование сценария в процессе его создания поможет сразу выявить потенциальные проблемы, хотя это и не гарантирует полное отсутствие ошибок. Ниже приведены некоторые практические советы, которые помогут устранять ошибки систематически.

Используйте марки наблюдения.

- Проверяйте данные, возвращаемые в различных местах кода, назначая их переменной OUT (аналогично методу зондирования программы).

Оставляйте подробные комментарии.

- Модуль кода будет намного проще отладить, если предполагаемый результат его работы будет точно описан.

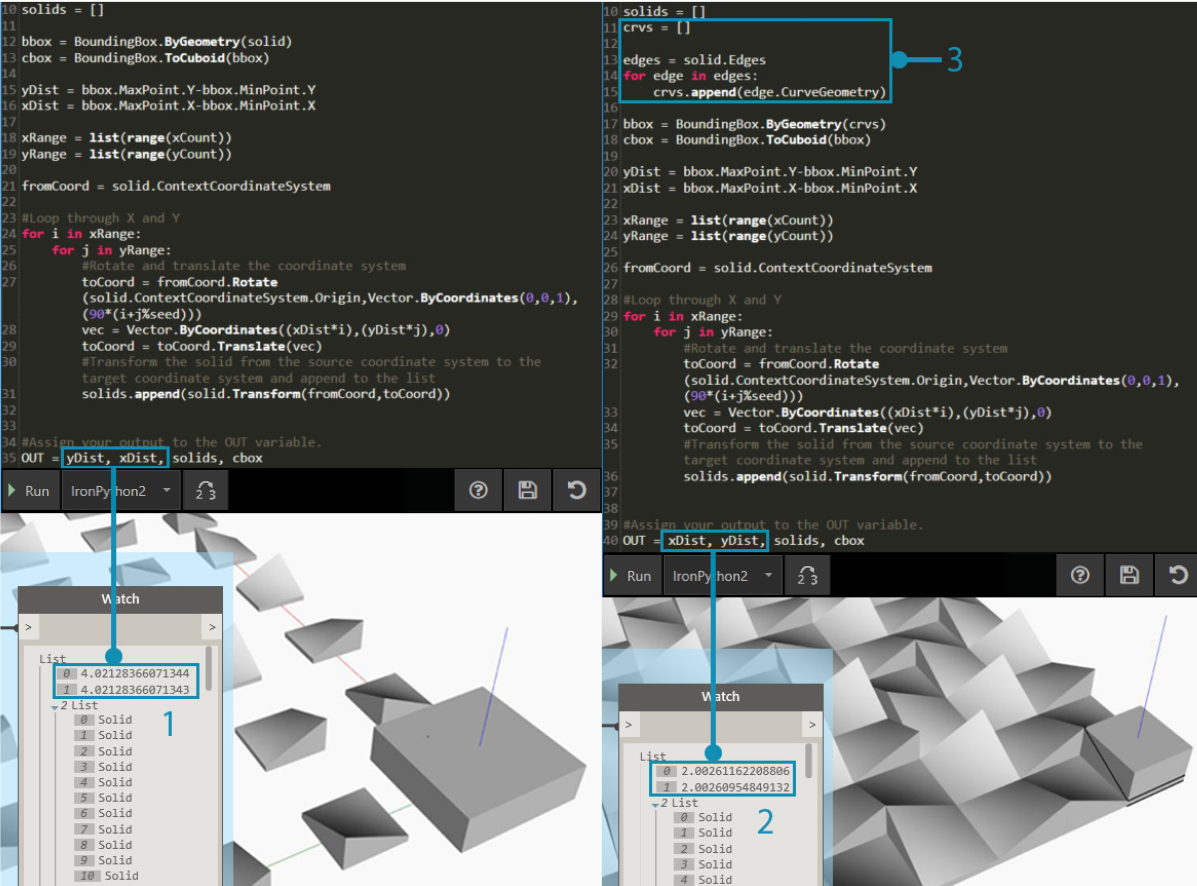
```
# Loop through X and Y
for i in range(xCount):
for j in range(yCount):

# Rotate and translate the coordinate system
```

```
toCoord = fromCoord.Rotate(solid.ContextCoordinateSystem.Origin,Vector.ByCoordinates(0,0,1),(90*(i+j%seec
vec = Vector.ByCoordinates((xDist*i),(yDist*j),0)
toCoord = toCoord.Translate(vec)

# Transform the solid from the source coord system to the target coord system and append to the list
solids.append(solid.Transform(fromCoord,toCoord))
```

Обычно это приводит к увеличению количества информации и пустых строк, но при отладке позволяет анализировать данные, разбив их на отдельные части.

Используйте модульный принцип организации кода.

- Источник проблемы можно свести к определенным модулям.
- После того как проблемный модуль определен, исправить проблему будет значительно проще.
- Если необходимо изменить программу, код, который был разработан в отдельных модулях, будет намного проще скорректировать.
    - Можно вставить в существующую программу новый или отлаженный модуль с уверенностью в том, что остальная часть программы не изменится.



Отладка файла примера из узла Python.

1. Входная геометрия возвращает ограничивающую рамку слишком большого размера. Это можно видеть, назначив переменной OUT значения xDist и yDist.
2. Кривые ребра входной геометрии имеют подходящую ограничивающую рамку и правильные расстояния xDist и yDist.
3. Вставлен модуль кода, позволяющий решить проблему со значениями xDist и yDist.

Упражнение «Траектория с наибольшей крутизной»

Скачайте файл примера, прилагаемый к этому упражнению (щелкните правой кнопкой мыши и выберите «Сохранить ссылку как...»). Полный список файлов примеров можно найти в приложении. SteepestPath.dyn

Напишите сценарий для моделирования дождевых осадков с учетом представленных здесь практических советов по созданию текстовых сценариев. Несмотря на то, что практические советы были успешно применены к плохо организованной визуальной программе в разделе «Методы создания графиков», данные операции значительно сложнее выполнять с текстовыми сценариями. Логические связи, существующие в текстовых сценариях, сложнее отслеживаются и почти не распознаются в запутанном коде. Вместе с возможностями, которые дают текстовые сценарии, повышаются и требования к организации кода. Рассмотрим каждый из этапов, применяя на деле практические советы.

Наш сценарий применялся к поверхности, деформированной точкой притяжения.

Сначала следует импортировать необходимые библиотеки Dynamo. Сделав это в первую очередь, мы предоставим глобальный доступ к функциональным возможностям Dynamo в Python.

Сюда необходимо импортировать все библиотеки, которые планируется использовать.

Затем следует определить входные и выходные данные сценария, которые будут отображаться в качестве входных портов узла. Эти внешние входные данные являются основой сценария и ключом к созданию параметрической среды.

Необходимо определить входные данные, соответствующие переменным в сценарии Python, и определить желаемый результат.

1. Поверхность, по которой будут спускаться агенты.
2. Количество движущихся агентов.
3. Максимальное количество шагов, которые могут сделать агенты.
4. Возможность использования кратчайшего пути вниз по поверхности или ее пересечения.
5. Узел Python с идентификаторами ввода, соответствующими входным данным в сценарии (IN[0], IN[1]).
6. Выходные кривые, которые могут отображаться другим цветом.

Теперь воспользуемся принципом модульной организации и создадим основную часть сценария. Важной задачей является моделирование кратчайшей траектории вниз по поверхности для нескольких начальных точек. Для этого требуется использование нескольких функций. Вместо того чтобы вызывать различные функции в сценарии, можно создать модули кода, собрав функции в одном классе (агенте). Различные функции этого класса (или модуля) можно вызывать с помощью различных переменных или даже использовать в других сценариях.

Для агента необходимо определить класс (макет), который позволит спускаться по поверхности, выбирая перемещения в направлении с наибольшей крутизной при каждом шаге.

1. Имя.
2. Глобальные атрибуты, общие для всех агентов.
3. Атрибуты экземпляра, уникальные для каждого агента.
4. Функция для выполнения шага.
5. Функция каталогизации положения каждого шага в списке маршрутов.

Теперь инициализируйте агентов, определив их начальное положение. Это хорошая возможность для зондирования сценария с целью проверки работоспособности класса агентов.

Необходимо создать экземпляры всех агентов, спуск которых по поверхности будет отслеживаться, и определить их исходные атрибуты.

1. Новый пустой список маршрутов.
2. Место начала движения по поверхности.
3. Мы назначили список агентов в качестве выходных данных, чтобы проверить результат, возвращаемый сценарием. Возвращается правильное количество агентов, но позднее потребуется снова выполнить зондирование сценария для проверки полученной с помощью него геометрии.

Обновляйте агентов при каждом шаге.

Затем необходимо вставить вложенный цикл, где положение каждого агента и шага будет обновляться и записываться в списке маршрутов. Кроме того, при выполнении каждого шага нужно проверять, не достиг ли агент точки на поверхности, где невозможно выполнить следующий шаг с направлением вниз. Если это условие выполняется, движение агента прекращается.

Теперь, когда все агенты полностью обновлены, получим соответствующую им геометрию.

После того как все агенты достигнут предельного значения спуска или максимального количества шагов, следует создать сложную кривую (PolyCurve), проходящую через точки в списке их маршрутов, и выведем маршруты сложной кривой.

Сценарий для поиска траекторий с наибольшей крутизной.

1. Набор параметров для моделирования осадков на базовой поверхности.
2. Вместо поиска траектории с наибольшей крутизной можно переключить агентов на пересечение базовой поверхности.

Полный текстовый сценарий Python.

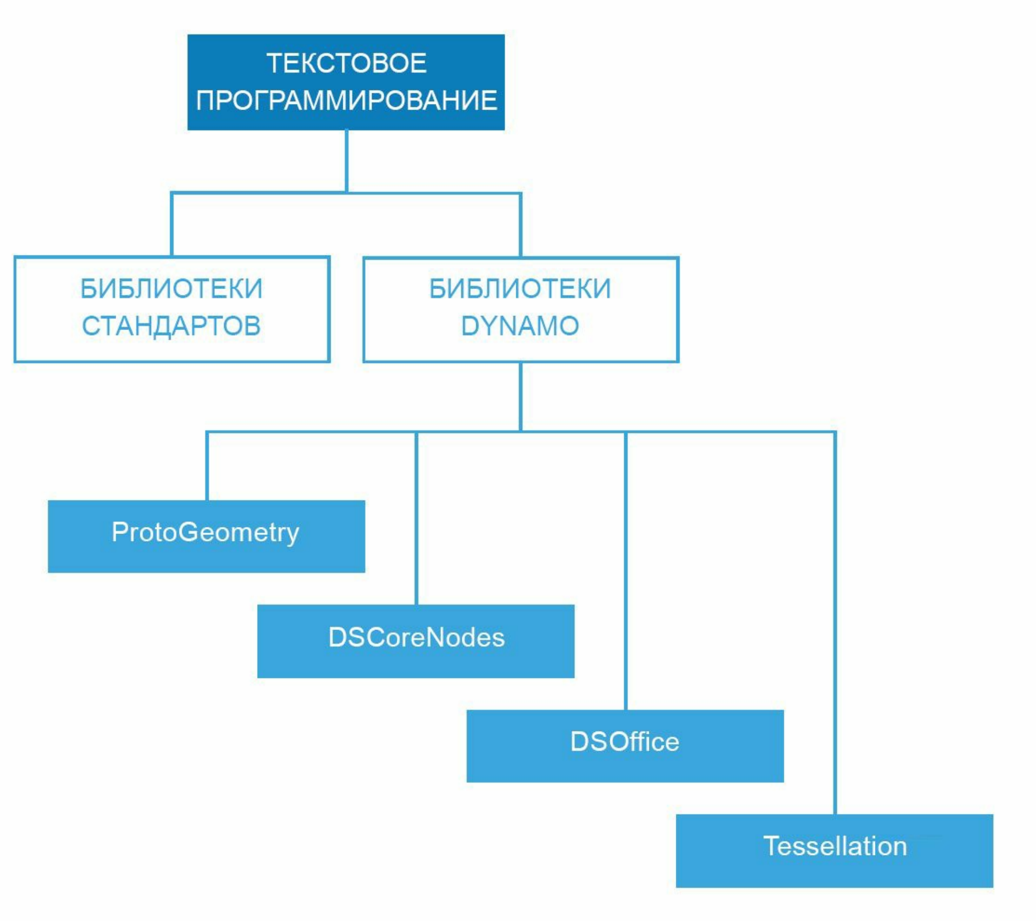
Справочник по созданию сценариев

Справочник по созданию сценариев

Данный справочник представляет собой дополнение к практическим рекомендациям, рассмотренным в главе «Методы создания сценариев». В нем содержатся дополнительные сведения о библиотеках кодов, метках и стиле. Для иллюстрации будет использоваться язык Python, однако принципы являются общими для Python и C# (Zero Touch) с учетом различий в синтаксисе.

Используемые библиотеки

Стандартные библиотеки не входят в состав Dynamo и написаны на языках программирования Python и C# (Zero Touch). В Dynamo также имеется собственный набор библиотек, которые точно отражают иерархию узлов модуля. Благодаря этому пользователи могут создавать программы с помощью узлов и проводов в форме кода. В представленном ниже руководстве описано содержимое каждой из библиотек Dynamo и случаи использования стандартных библиотек.



Стандартные библиотеки и библиотеки Dynamo

- Для формирования данных и процессов со сложной структурой в среде Dynamo можно использовать стандартные библиотеки Python и C#.
- Библиотеки Dynamo в точности следуют иерархии узлов, что удобно при создании геометрических и других объектов Dynamo.

Библиотеки Dynamo

- ProtoGeometry
    - Функции: дуга, ограничивающая рамка, окружность, конус, система координат, кубоид, кривая, цилиндр, ребро, эллипс, дуга эллипса, грань, геометрия, спираль, группа индексов, линия, сеть, NURBS-кривая, NURBS-поверхность, плоскость, точка, полигон, прямоугольник, тело, сфера, поверхность, топология, T-сплайн, UV, вектор, вершина.
    - Импорт: `import Autodesk.DesignScript.Geometry`
    - **Обратите внимание, что при работе с библиотекой ProtoGeometry в Python или C#** создаются неуправляемые объекты, память которых можно освободить только вручную. Дополнительные сведения см. в разделе **Неуправляемые объекты** ниже.
- DSCoreNodes
    - Функции: цвет, диапазон цветов 2D, дата и время, интервал времени, ввод/вывод, формула, логика, список, математическое вычисление, квадрадерево, строка, поток.
    - Импорт: `import DSCore`
- Tessellation

    - Функции: выпуклая оболочка, Делоне, Вороной.
    - Импорт `import Tessellation`
- DSOffice
    - Функции: Excel.
    - Импорт: `import DSOffice`

Осторожное использование меток

При написании сценариев постоянно используются идентификаторы, которые служат для обозначения переменных, типов, функций и других элементов. Благодаря этой системе условных обозначений можно строить алгоритмы, ссылаясь на данные посредством меток, которые обычно представляют собой последовательность символов. Правильное присвоение имен очень важно при написании кода, так как позволяет сделать его понятным не только другим пользователям, но и самому автору в будущем. Ниже приводятся рекомендации по присвоению имен элементам сценария.

Использование сокращений допускается, но с поясняющим комментарием:

```
### BAD
csfX = 1.6
csfY= 1.3
csfZ = 1.0
```

```
### GOOD
# column scale factor (csf)
csfX = 1.6
csfY= 1.3
csfZ = 1.0
```

Избегайте избыточных меток:

```
### BAD
import car
seat = car.CarSeat()
tire = car.CarTire()
```

```
### GOOD
import car
seat = car.Seat()
tire = car.Tire()
```

В именах переменных используйте положительную, а не отрицательную логику:

```
### BAD
if 'mystring' not in text:
print 'not found'
else:
print 'found'
print 'processing'
```

```
### GOOD
if 'mystring' in text:
print 'found'
print 'processing'
else:
print 'not found'
```

Старайтесь использовать обратный порядок слов в обозначениях:

```
### BAD
agents = …
active_agents = …
dead_agents ...
```

```
### GOOD
agents = …
agents_active = …
agents_dead = ...
```

Это более целесообразно с точки зрения структуры.

Для сокращения длинных или часто повторяющихся цепочек наименований используйте псевдонимы:

```
### BAD
from RevitServices.Persistence import DocumentManager

DocumentManager = DM

doc = DM.Instance.CurrentDBDocument
uiapp = DM.Instance.CurrentUIApplication
```

```
### GOOD
from RevitServices.Persistence import DocumentManager as DM

doc = DM.Instance.CurrentDBDocument
uiapp = DM.Instance.CurrentUIApplication
```

Однако помните, что использование псевдонимов может сделать программу непонятной и нестандартной.

Используйте только необходимые слова:

```
### BAD
rotateToCoord = rotateFromCoord.Rotate(solid.ContextCoordinateSystem.Origin,Vector.ByCoordinates(0,0,1),5

### GOOD
toCoord = fromCoord.Rotate(solid.ContextCoordinateSystem.Origin,Vector.ByCoordinates(0,0,1),5)
```

«Все должно быть изложено так просто, как только возможно, но не проще». — Альберт Эйнштейн

Единообразие стиля

Любую программу можно написать несколькими способами, то есть персональный стиль создания сценариев формируется в результате принятия (или непринятия) бесчисленных мелких решений по ходу работы. Это означает, что читаемость и возможность доработки кода — прямой результат внутренней согласованности и соблюдения общих стилистических правил. Главное правило: одинаковый код в двух разных местах должен работать одинаково. Ниже приводятся советы по созданию понятного единообразного кода.

Правила именования (выберите одно из следующих правил для каждого типа объекта в коде и придерживайтесь его)

- Переменные, функции, методы, пакеты, модули: `нижний_регистр_с_нижним_подчеркиванием`
- Классы и исключения: `СловаСЗаглавнойБуквы`
- Защищенные методы и внутренние функции: `_одно_нижнее_подчеркивание_в_начале(self, ...)`
- Собственные методы: `__двойное_нижнее_подчеркивание_в_начале(self, ...)`
- Константы: `ВСЕ_ПРОПИСНЫЕ_С_НИЖНИМИ_ПОДЧЕРКИВАНИЯМИ`

Совет. Следует избегать использования переменных из одной буквы (особенно l, O, I). Исключением могут быть очень короткие блоки, когда значение легко понять из ближайшего контекста.

Использование пустых строк

- До и после определения функции верхнего уровня или класса следует оставлять две пустые строки.
    - До и после определения метода внутри класса следует оставлять одну пустую строку.
    - Для разделения групп связанных функций можно использовать дополнительные пустые строки (в разумных количествах).

Избегайте ненужных пробелов

- После открывающейся или перед закрывающейся круглой скобкой, квадратной скобкой или фигурной скобкой:

```
### BAD
function( apples[ 1 ], { oranges: 2 } )

### GOOD:
function(apples[1], {oranges: 2})
```

- Перед запятой, точкой с запятой или двоеточием:

```
### BAD
if x == 2 : print x , y ; x , y = y , x

### GOOD
if x == 2: print x, y; x, y = y, x
```

- Перед открывающейся скобкой, за которой следует список аргументов вызываемой функции:

```
### BAD
function (1)

### GOOD
function(1)
```

- Перед открывающейся скобкой, за которой следует индексирование или членение:

```
### BAD
dict ['key'] = list [index]

### GOOD
dict['key'] = list[index]
```

- До и после следующих двоичных операторов всегда вставляйте одиночный пробел:

```
assignment ( = )
augmented assignment ( += , -= etc.)
comparisons ( == , < , > , != , <> , <= , >= , in , not in , is , is not )
Booleans ( and , or , not )
```

Следите за длиной строки

- Она не должна превышать 79 символов.

- Если ограничить ширину окна редактора, можно открыть несколько файлов рядом. Это также удобно при использовании инструментов проверки кода, когда обе версии кода представлены в соседних столбцах.

- Длинные строки можно разбить на несколько строк, заключив выражения в круглые скобки:

Избегайте очевидных и лишних комментариев

- Иногда меньшее количество комментариев делает код более читабельным, особенно если вместо них используются понятные идентификаторы.

- Хороший стиль написания кода уменьшает зависимость от комментариев:

```
### BAD
# get the country code
country_code = get_country_code(address)

# if country code is US
if (country_code == 'US'):
# display the form input for state
print form_input_state()

### GOOD
# display state selection for US users
country_code = get_country_code(address)
if (country_code == 'US'):
print form_input_state()
```

Совет. Комментарии отвечают на вопрос «Почему?», код — на вопрос «Как?».

Просматривайте открытый исходный код

- Проекты с открытым исходным кодом создаются совместными усилиями многих разработчиков. Код, на котором пишутся эти проекты, должен быть максимально понятным, чтобы обеспечить эффективную работу всей группы. Поэтому рекомендуется просматривать исходный код подобных проектов, чтобы понять ход мыслей разработчиков.

- Совершенствуйте правила:

  - Задавайте вопросы о том, срабатывает ли то или иное правило в отношении текущих задач.

  - Не ухудшается ли функциональность или эффективность?

Стандарты C# (Zero Touch)

Посетите эти страницы Wiki, чтобы узнать об особенностях написания кода на C# для Zero Touch и его вставки в Dynamo.

- На этой странице рассматриваются некоторые общие стандарты, касающиеся документирования и проверки кода: https://github.com/DynamoDS/Dynamo/wiki/Coding-Standards

- На этой странице рассматриваются стандарты именования для библиотек, категорий, имен узлов, имен портов и сокращений: https://github.com/DynamoDS/Dynamo/wiki/Naming-Standards

  Неуправляемые объекты

  При использовании библиотеки геометрических объектов Dynamo *(ProtoGeometry)* в Python или C# управлять геометрическими объектами с помощью виртуальной машины будет невозможно, и память многих из этих объектов необходимо освобождать вручную. Для удаления локальных или неуправляемых объектов можно использовать метод **Dispose** или ключевое слово **using**. Дополнительные сведения см. на следующей странице Wiki: https://github.com/DynamoDS/Dynamo/wiki/Zero-Touch-Plugin-Development#dispose--using-statement.

  Удалять следует только неуправляемые ресурсы, которые не возвращаются в график или на которые не указывают ссылки. Далее в этом разделе подобные объекты будут называться *промежуточной геометрией*. Этот класс объектов используется в примере кода, приведенном ниже. Функция C# Zero Touch с именем **singleCube** возвращает один куб, но в процессе выполнения создает еще 10 000 кубов. Предположим, что оставшиеся геометрические объекты были использованы в качестве некой промежуточной вспомогательной геометрии.

  Скорее всего, эта функция вызовет аварийное завершение работы Dynamo.В результате ее использования было создано 10 000 тел, а сохранено и возвращено одно из них. В этом случае необходимо удалить все промежуточные кубы, кроме возвращаемого. Возвращаемый объект удалять не нужно, так как он будет распространен по графику и использован в других узлах.

```
public Cuboid singleCube(){

var output = Cuboid.ByLengths(1,1,1);

for(int i = 0; i<10000;i++){
output = Cuboid.ByLengths(1,1,1);
}
return output;
}
```

Постоянный код должен выглядеть примерно так:

```
public Cuboid singleCube(){

var output = Cuboid.ByLengths(1,1,1);
var toDispose = new List<Geometry>();
```

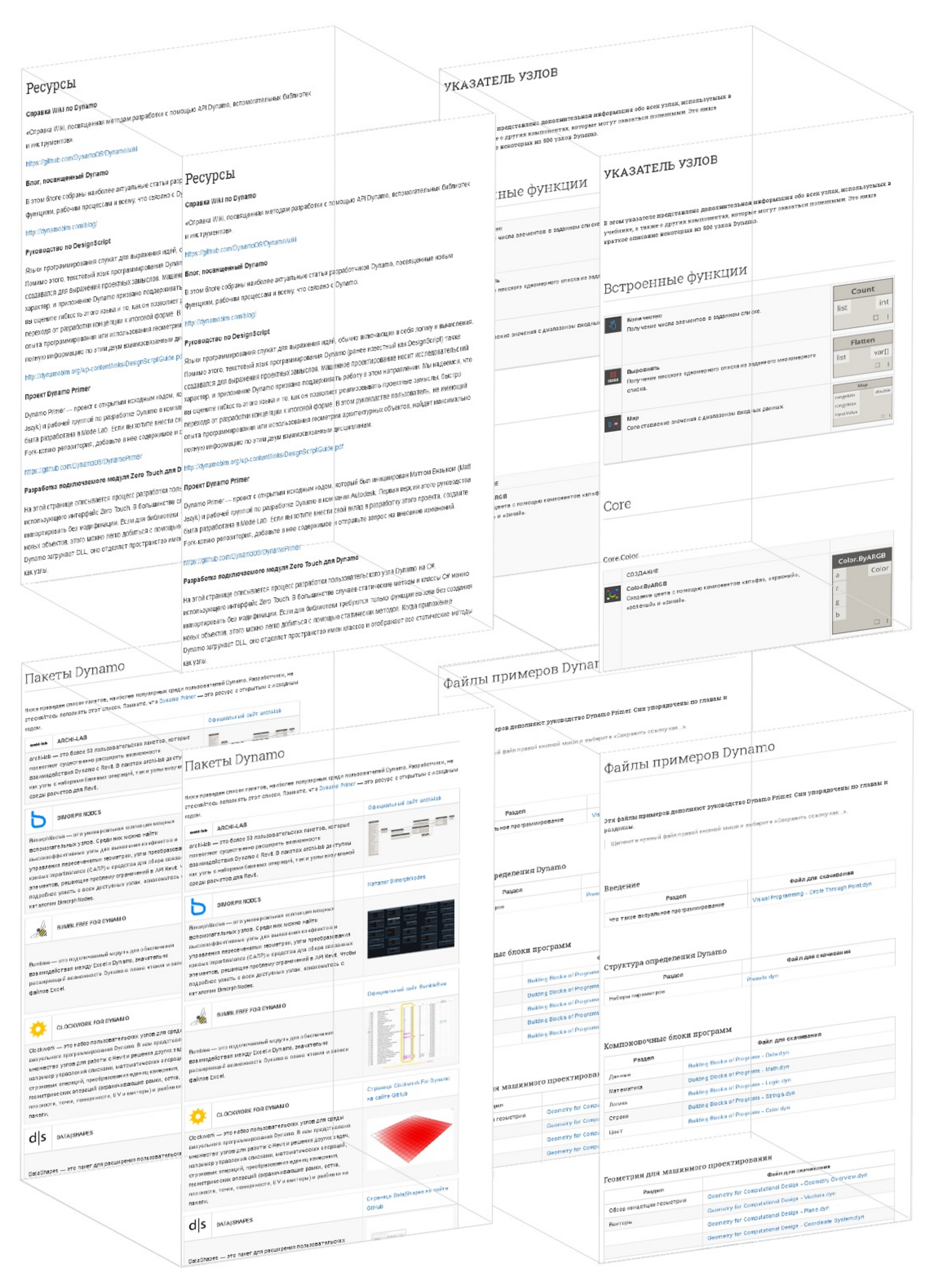
```
for(int i = 0; i<10000;i++){
toDispose.Add(Cuboid.ByLengths(1,1,1));
}

foreach(IDisposable item in toDispose ){
item.Dispose();
}

return output;
}
```

Обычно требуется удалять только такие геометрические объекты, как поверхности, кривые и тела. Однако в целях безопасности можно удалить все типы геометрии (векторы, точки, системы координат).

Приложение

Приложение A. Ресурсы

В этом разделе указаны дополнительные ресурсы, которые позволят перейти на новый уровень владения Dynamo. Кроме того, мы добавили в это руководство индекс важных узлов, набор полезных пакетов и хранилище файлов примеров. Мы будем рады, если вы примете участие в пополнении этого раздела. Портал Dynamo Primer открыт для всех.



Ресурсы
Справка Wiki по Dynamo
Блог, посвященный Dynamo
Руководство по DesignScript
Проект Dynamo Primer
Разработка подключаемого модуля Zero Touch для Dynamo
УКАЗАТЕЛЬ УЗЛОВ
Встроенные функции
Count
Flatten
Core
Color.ByARGB
Пакеты Dynamo
ARCHI-LAB
BIMORPH NODES
BUMBLEBEE FOR DYNAMO
CLOCKWORK FOR DYNAMO
DATA|SHAPES
Файлы примеров Dynamo
Введение
Структура определения Dynamo
Компоновочные блоки программ
Геометрия для машинного проектирования

Ресурсы

Ресурсы

Справка Wiki по Dynamo

«Справка Wiki, посвященная методам разработки с помощью API Dynamo, вспомогательных библиотек и инструментов».

https://github.com/DynamoDS/Dynamo/wiki

Блог, посвященный Dynamo

В этом блоге собраны наиболее актуальные статьи разработчиков Dynamo, посвященные новым функциям, рабочим процессам и всему, что связано с Dynamo.

http://dynamobim.com/blog/

Руководство по DesignScript

Языки программирования служат для выражения идей, обычно включающих в себя логику и вычисления. Помимо этого, текстовый язык программирования Dynamo (ранее известный как DesignScript) также создавался для выражения проектных замыслов. Машинное проектирование носит исследовательский характер, и приложение Dynamo призвано поддерживать работу в этом направлении. Мы надеемся, что вы оцените гибкость этого языка и то, как он позволяет реализовывать проектные замыслы, быстро переходя от разработки концепции к итоговой форме. В этом руководстве пользователь, не имеющий опыта программирования или использования геометрии архитектурных объектов, найдет максимально полную информацию по этим двум взаимосвязанным дисциплинам.

http://dynamobim.org/wp-content/links/DesignScriptGuide.pdf

Проект Dynamo Primer

Dynamo Primer — проект с открытым исходным кодом, который был инициирован Мэттом Ензыком (Matt Jezyk) и рабочей группой по разработке Dynamo в компании Autodesk. Первая версия этого руководства была разработана в Mode Lab. Если вы хотите внести свой вклад в разработку этого проекта, создайте Fork-копию репозитория, добавьте в нее содержимое и отправьте запрос на внесение изменений.

https://github.com/DynamoDS/DynamoPrimer

Разработка подключаемого модуля Zero Touch для Dynamo

На этой странице описывается процесс разработки пользовательского узла Dynamo на C#, использующего интерфейс Zero Touch. В большинстве случаев статические методы и классы C# можно импортировать без модификации. Если для библиотеки требуются только функции вызова без создания новых объектов, этого можно легко добиться с помощью статических методов. Когда приложение Dynamo загружает DLL, оно отделяет пространство имен классов и отображает все статические методы как узлы.

https://github.com/DynamoDS/Dynamo/wiki/Zero-Touch-Plugin-Development

Python для начинающих

Python — это интерпретируемый, интерактивный, объектно-ориентированный язык программирования. Он включает в себя модули, исключения, динамический ввод, динамические типы данных очень высокого уровня и классы. Python сочетает в себе высокую мощность и понятный синтаксис. Он включает интерфейсы для взаимодействия с различными системными вызовами и библиотеками, а также с различными оконными системами. Кроме того, он поддерживает расширение с использованием C или C++. Его можно использовать как язык расширения для приложений, которым требуется программируемый интерфейс. Наконец, язык Python является переносимым: он работает на множестве вариантов Unix, компьютерах Mac, на платформах Windows 2000 и более поздних версий. В руководстве по Python для начинающих приведены ссылки на другие ознакомительные учебные пособия и ресурсы для обучения программированию на Python.

https://www.python.org/about/gettingstarted

AForge

AForge.NET — это платформа C# с открытым исходным кодом, предназначенная для разработчиков и исследователей в сферах компьютерного зрения и искусственного интеллекта: обработка изображений, нейронные сети, генетические алгоритмы, нечеткая логика, машинное обучение, робототехника и т. д.

http://www.aforgenet.com/framework/

Wolfram MathWorld

MathWorld — это математический онлайн-ресурс, составленный Эриком В. Вайсстайном с помощью тысяч соавторов. С момента первой публикации в 1995 г. MathWorld стал лидирующим информационным ресурсом по математике как в математическом, так и в образовательном сообществах. На публикации MathWorld ссылается огромное количество журналов и книг разных степеней научности.

http://mathworld.wolfram.com/

Ресурсы по Revit

buildz

«Эти публикации в основном посвящены платформе Revit и помогают пользователям работать с ней с удовольствием».

http://buildz.blogspot.com/

Nathan's Revit API Notebook

«Эти записи призваны устранить ряд пробелов в ресурсах для изучения и применения API Revit в контексте рабочего процесса проектирования».

http://wiki.theprovingground.org/revit-api

Оболочка Python в Revit

«RevitPythonShell добавляет в Autodesk Revit и Vasari модуль IronPython, интерпретирующий данные». Этот проект возник до появления Dynamo и является отличным источником информации по разработке на Python. Проект RPS: https://github.com/architecture-building-systems/revitpythonshell Блог разработчика: http://darenatwork.blogspot.com/

The Building Coder

Исчерпывающий каталог рабочих процессов на основе API Revit от одного из ведущих специалистов по BIM.

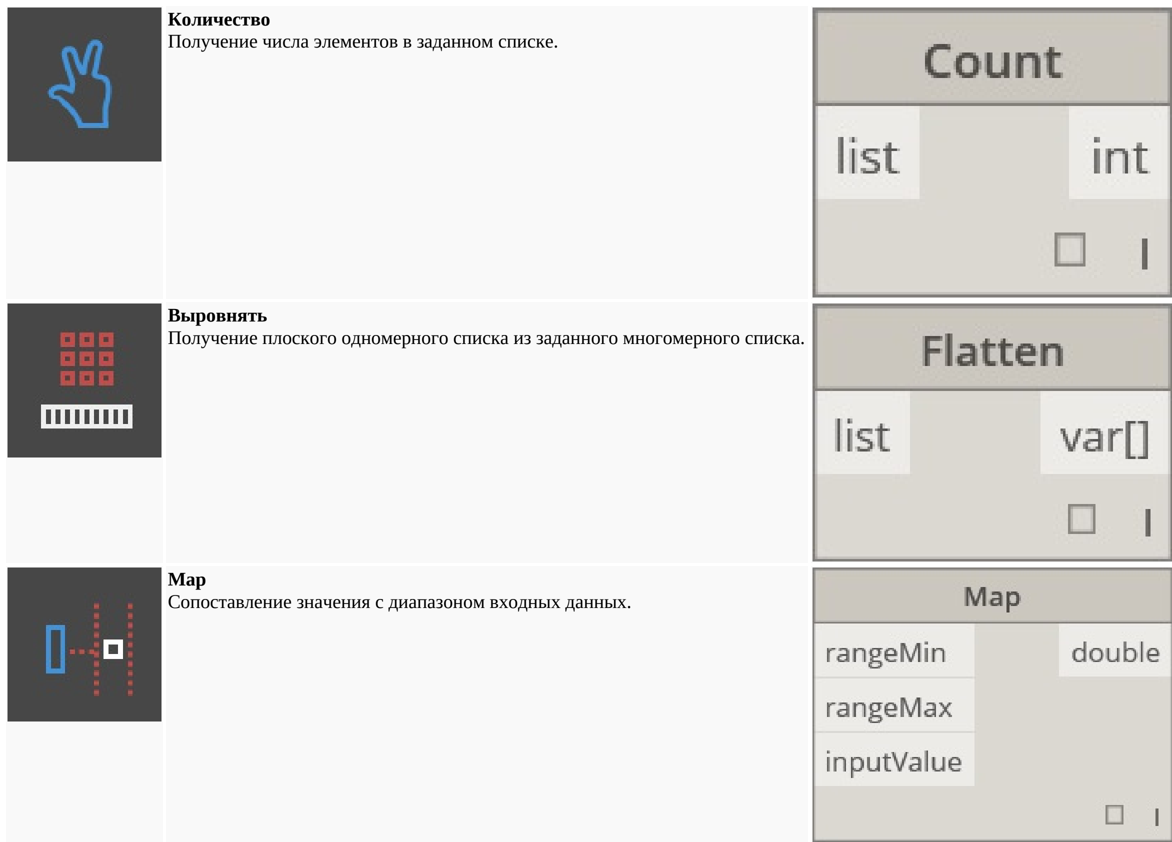
http://thebuildingcoder.typepad.com/

Указатель узлов

УКАЗАТЕЛЬ УЗЛОВ

В этом указателе представлена дополнительная информация обо всех узлах, используемых в учебнике, а также о других компонентах, которые могут оказаться полезными. Это лишь краткое описание некоторых из 500 узлов Dynamo.

Встроенные функции

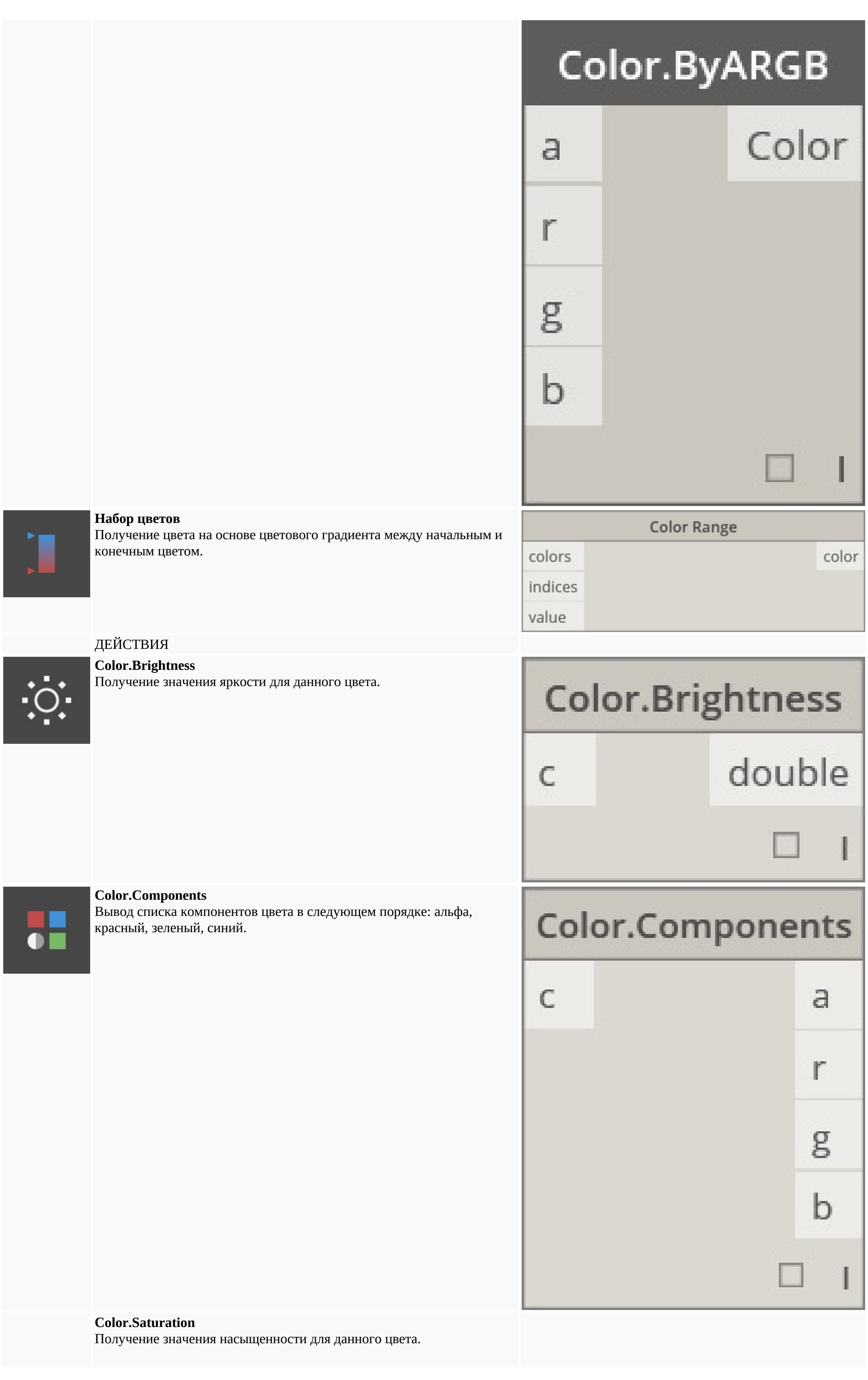


	Количество Получение числа элементов в заданном списке.	Count list int
	Выровнять Получение плоского одномерного списка из заданного многомерного списка.	Flatten list var[]
	Map Сопоставление значения с диапазоном входных данных.	Map rangeMin double rangeMax inputValue

Core

Core.Color

	СОЗДАНИЕ	
	Color.ByARGB Создание цвета с помощью компонентов «альфа», «красный», «зеленый» и «синий».	



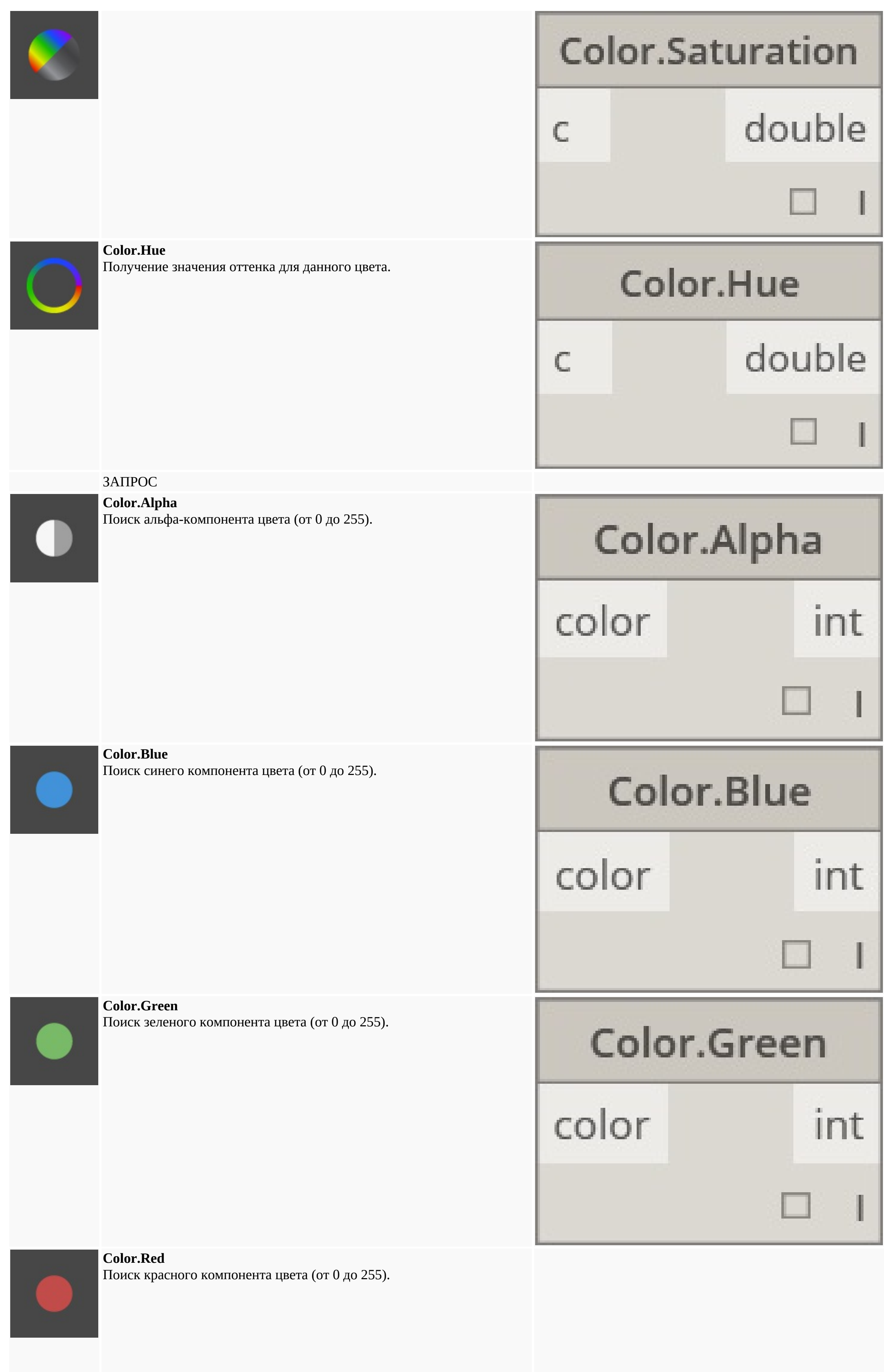
Набор цветов
Получение цвета на основе цветового градиента между начальным и конечным цветом.

ДЕЙСТВИЯ

Color.Brightness
Получение значения яркости для данного цвета.

Color.Components
Вывод списка компонентов цвета в следующем порядке: альфа, красный, зеленый, синий.

Color.Saturation
Получение значения насыщенности для данного цвета.



Color.Hue
Получение значения оттенка для данного цвета.

ЗАПРОС

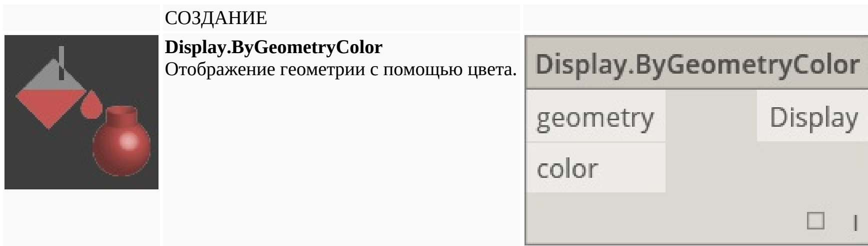
Color.Alpha
Поиск альфа-компонента цвета (от 0 до 255).

Color.Blue
Поиск синего компонента цвета (от 0 до 255).

Color.Green
Поиск зеленого компонента цвета (от 0 до 255).

Color.Red
Поиск красного компонента цвета (от 0 до 255).

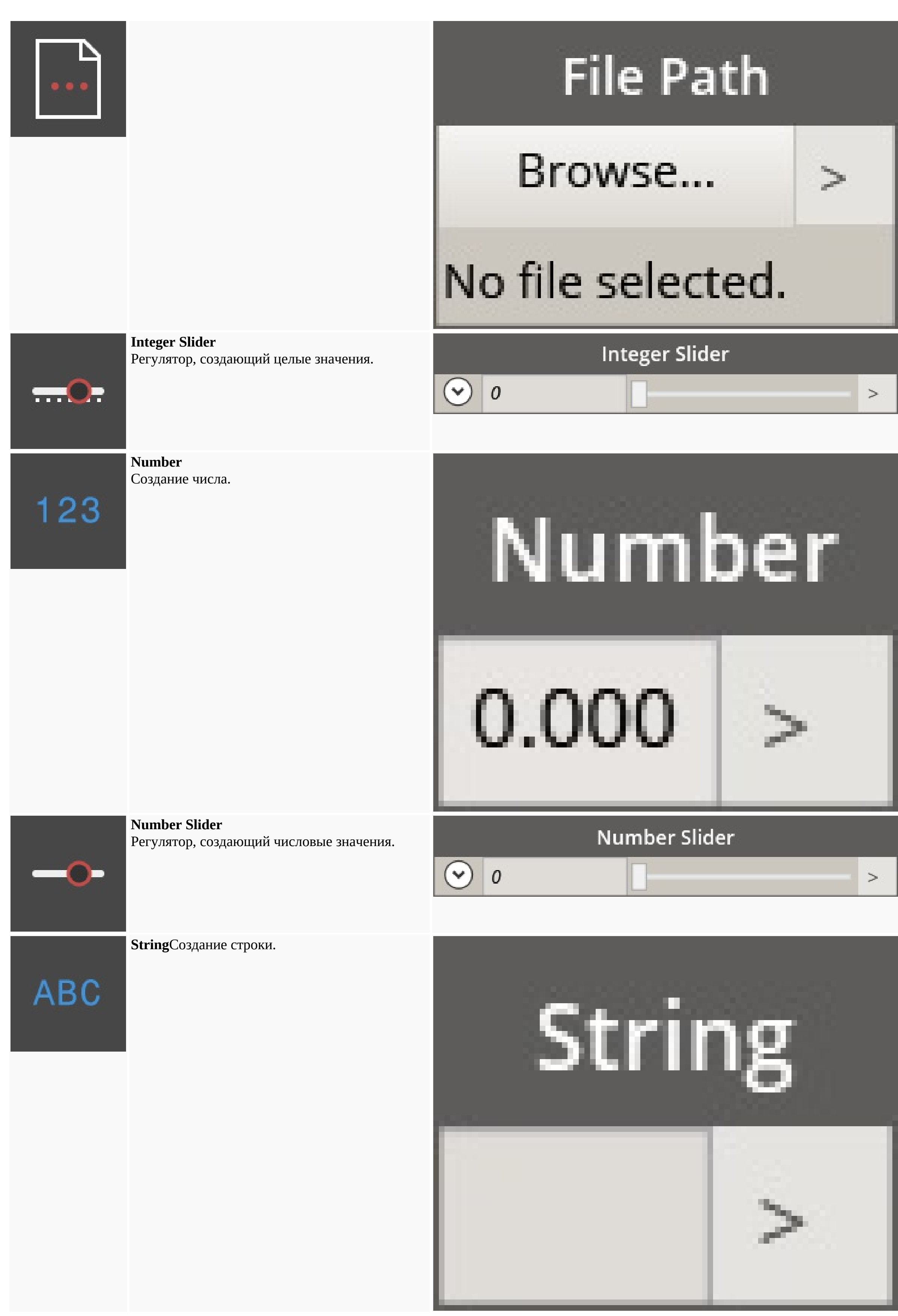
Core.Display



	СОЗДАНИЕ	
	Display.ByGeometryColor Отображение геометрии с помощью цвета.	

Core.Input

	ДЕЙСТВИЯ	
	Boolean Выбор между значениями True и False.	
	Code Block Непосредственная разработка кода DesignScript.	
	Directory Path Выбор папки в системе и получение пути к ней.	
	File Path Выбор файла в системе и получение его имени.	



Integer Slider
Регулятор, создающий целые значения.

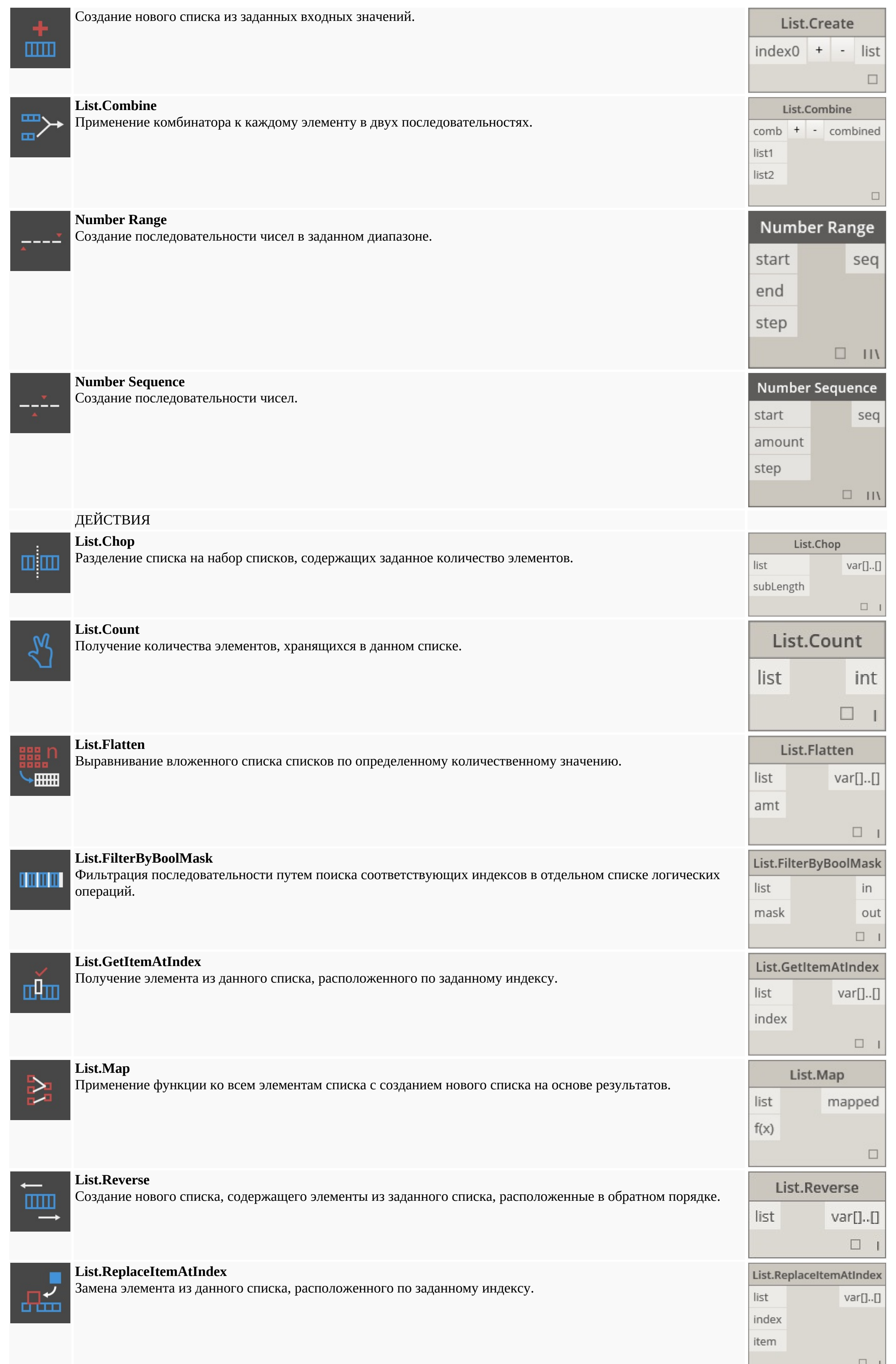
Number
Создание числа.

Number Slider
Регулятор, создающий числовые значения.

StringСоздание строки.

Core.List

	СОЗДАНИЕ	
	List.Create	



Создание нового списка из заданных входных значений.

List.Combine
Применение комбинатора к каждому элементу в двух последовательностях.

Number Range
Создание последовательности чисел в заданном диапазоне.

Number Sequence
Создание последовательности чисел.

ДЕЙСТВИЯ

List.Chop
Разделение списка на набор списков, содержащих заданное количество элементов.

List.Count
Получение количества элементов, хранящихся в данном списке.

List.Flatten
Выравнивание вложенного списка списков по определенному количественному значению.

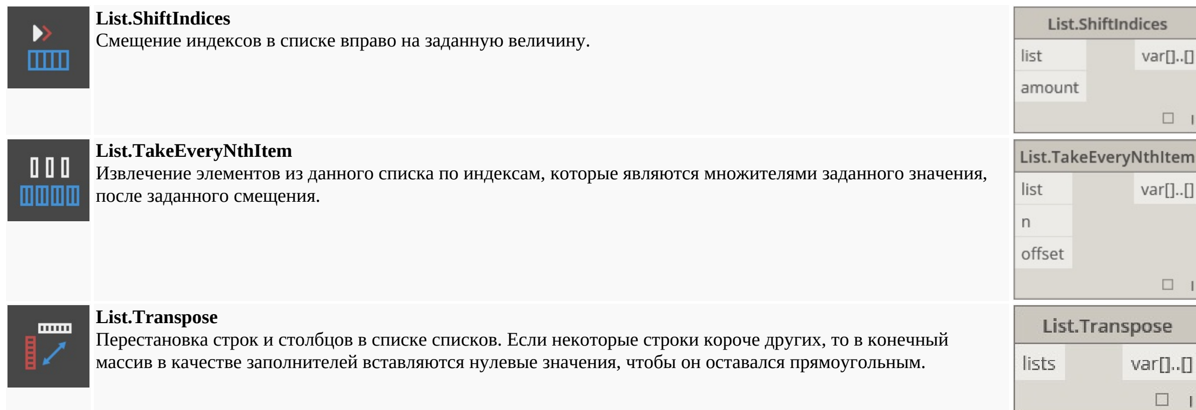
List.FilterByBoolMask
Фильтрация последовательности путем поиска соответствующих индексов в отдельном списке логических операций.

List.GetItemAtIndex
Получение элемента из данного списка, расположенного по заданному индексу.

List.Map
Применение функции ко всем элементам списка с созданием нового списка на основе результатов.

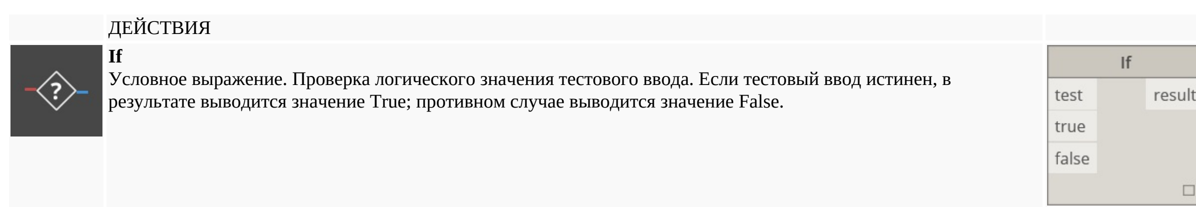
List.Reverse
Создание нового списка, содержащего элементы из заданного списка, расположенные в обратном порядке.

List.ReplaceItemAtIndex
Замена элемента из данного списка, расположенного по заданному индексу.



	List.ShiftIndices Смещение индексов в списке вправо на заданную величину.	List.ShiftIndices list — var[]..[] amount
	List.TakeEveryNthItem Извлечение элементов из данного списка по индексам, которые являются множителями заданного значения, после заданного смещения.	List.TakeEveryNthItem list — var[]..[] n offset
	List.Transpose Перестановка строк и столбцов в списке списков. Если некоторые строки короче других, то в конечный массив в качестве заполнителей вставляются нулевые значения, чтобы он оставался прямоугольным.	List.Transpose lists — var[]..[]

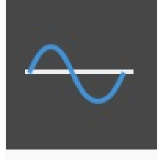
Core.Logic

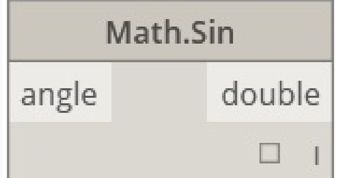


	ДЕЙСТВИЯ	
	If Условное выражение. Проверка логического значения тестового ввода. Если тестовый ввод истинен, в результате выводится значение True; противном случае выводится значение False.	If test — result true false

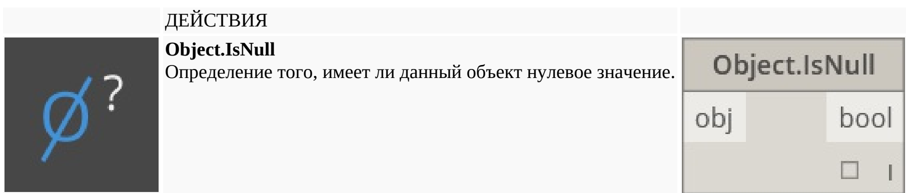
Core.Math

	ДЕЙСТВИЯ	
	Math.Cos Нахождение косинуса угла.	Math.Cos angle — double
	Math.DegreesToRadians Преобразование единиц угла из градусов радианы.	Math.DegreesToRadians degrees — double
	Math.Pow Возведение числа в заданную степень.	Math.Pow number — double power
	Math.RadiansToDegrees Преобразование единиц угла из радианов в градусы.	Math.RadiansToDegrees radians — double
	Math.RemapRange Корректировка диапазона списка чисел при сохранении коэффициента распределения.	Math.RemapRange numbers — var[]..[] newMin newMax
	Math.Sin Поиск синуса угла.	





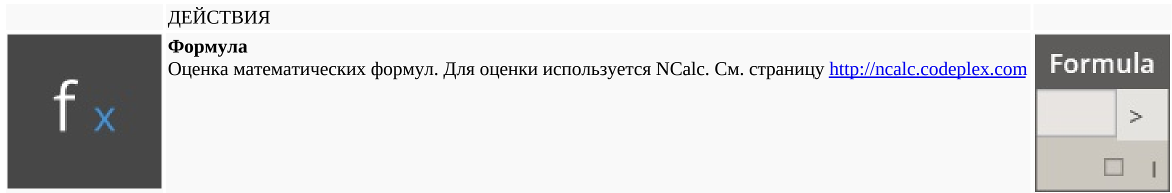
Core.Object



ДЕЙСТВИЯ

Object.IsNull
Определение того, имеет ли данный объект нулевое значение.

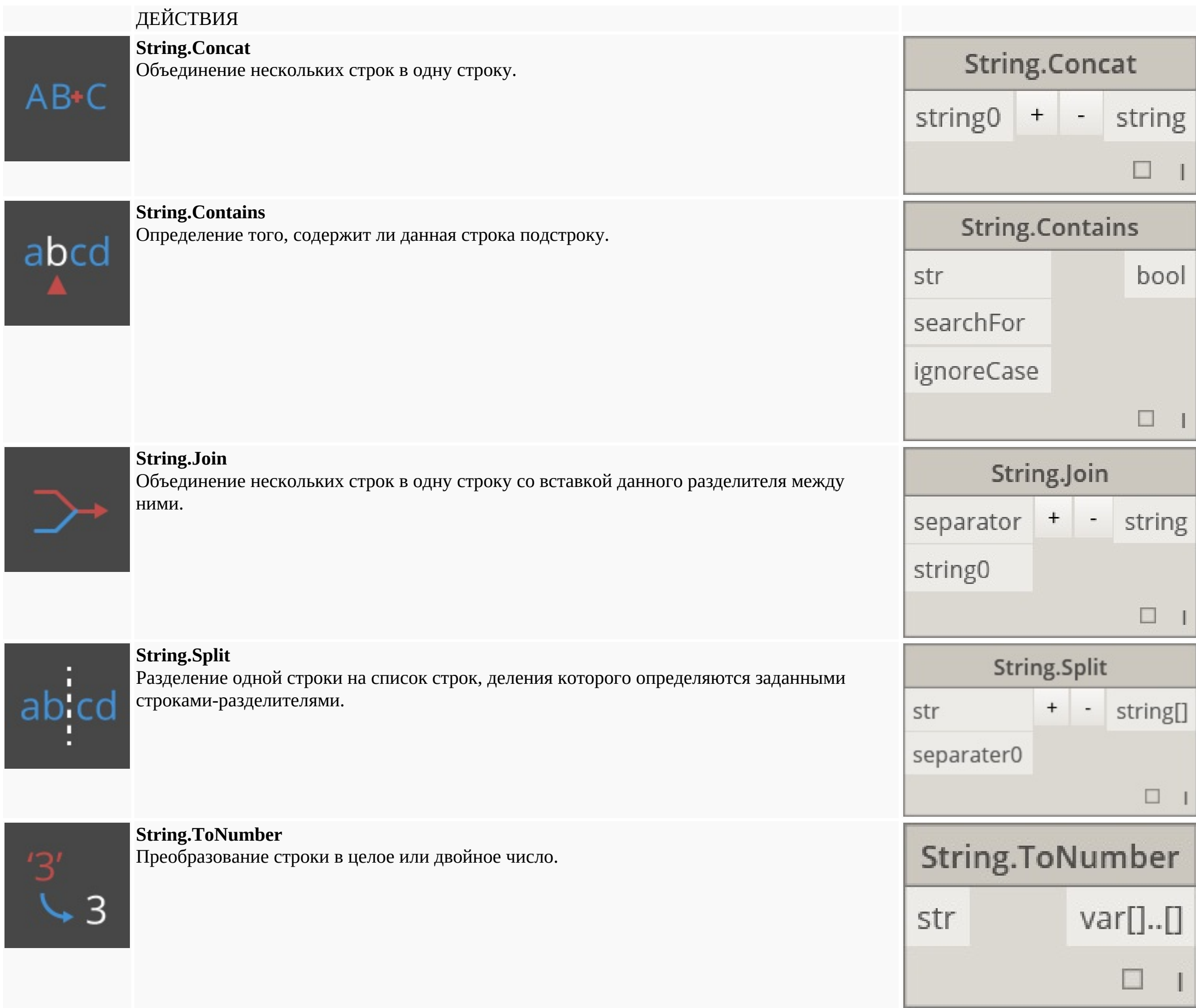
Core.Scripting



ДЕЙСТВИЯ

Формула
Оценка математических формул. Для оценки используется NCalc. См. страницу http://ncalc.codeplex.com

Core.String



ДЕЙСТВИЯ

String.Concat
Объединение нескольких строк в одну строку.

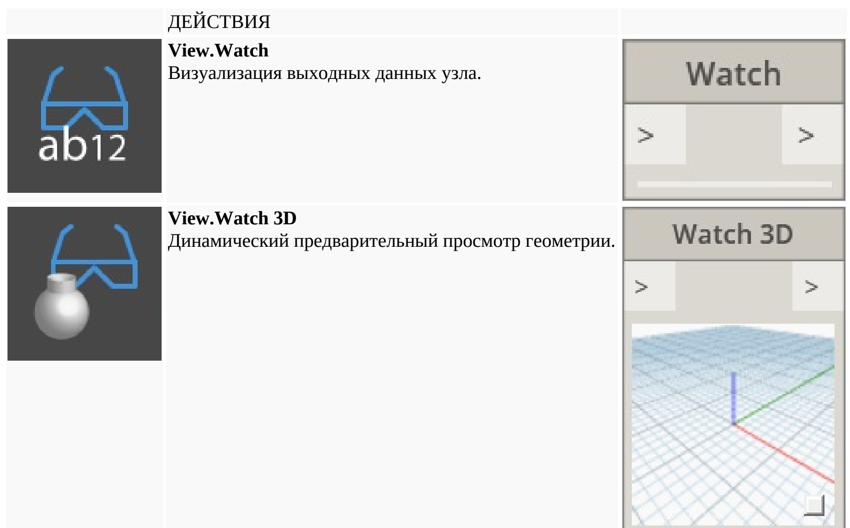
String.Contains
Определение того, содержит ли данная строка подстроку.

String.Join
Объединение нескольких строк в одну строку со вставкой данного разделителя между ними.

String.Split
Разделение одной строки на список строк, деления которого определяются заданными строками-разделителями.

String.ToNumber
Преобразование строки в целое или двойное число.

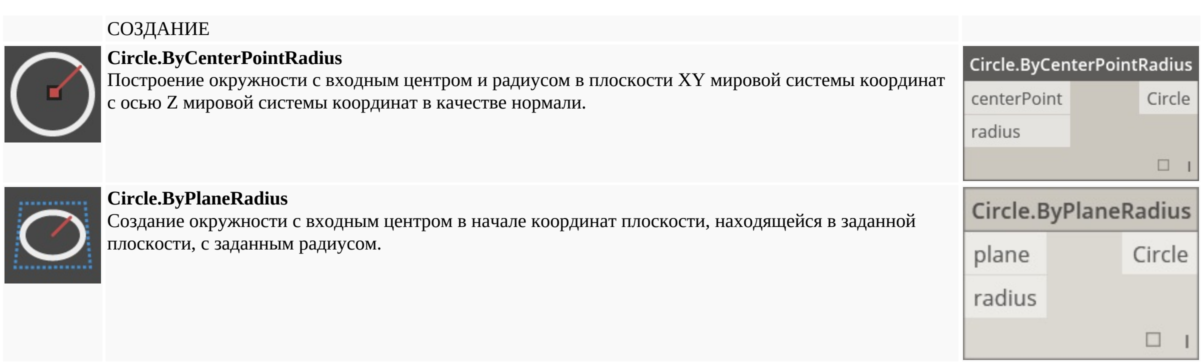
Core.View



	ДЕЙСТВИЯ
	View.Watch Визуализация выходных данных узла.
	View.Watch 3D Динамический предварительный просмотр геометрии.

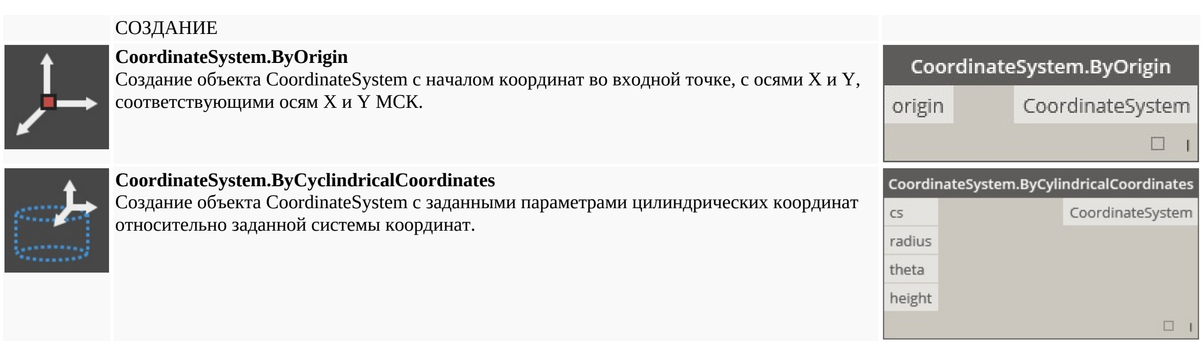
Геометрия

Geometry.Circle



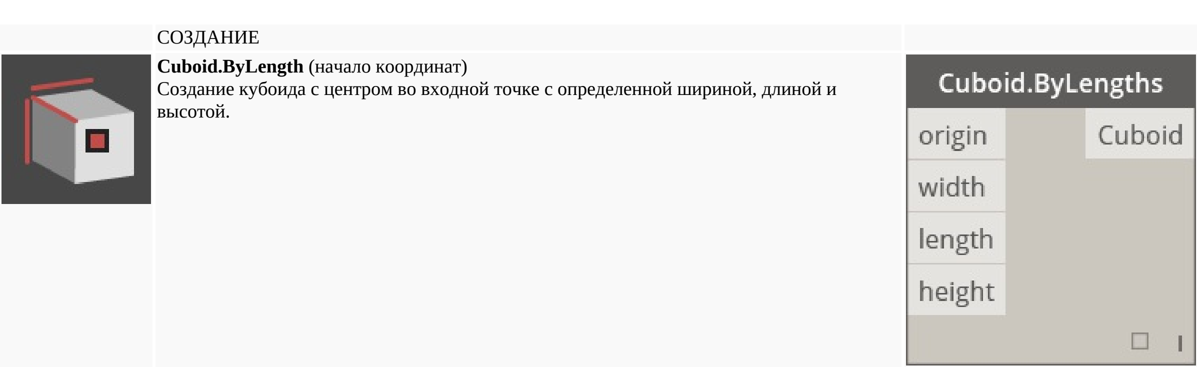
	СОЗДАНИЕ
	Circle.ByCenterPointRadius Построение окружности с входным центром и радиусом в плоскости XY мировой системы координат с осью Z мировой системы координат в качестве нормали.
	Circle.ByPlaneRadius Создание окружности с входным центром в начале координат плоскости, находящейся в заданной плоскости, с заданным радиусом.

Geometry.CoordinateSystem



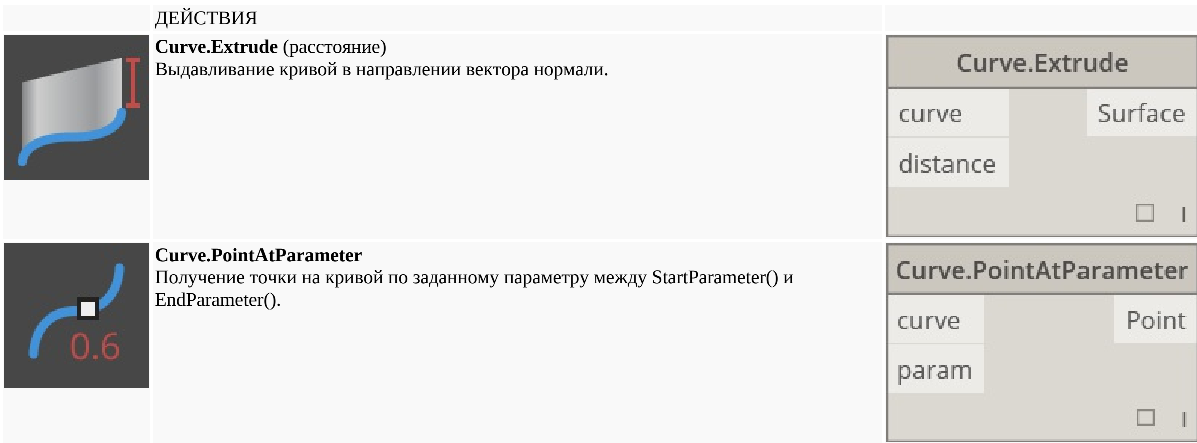
	СОЗДАНИЕ
	CoordinateSystem.ByOrigin Создание объекта CoordinateSystem с началом координат во входной точке, с осями X и Y, соответствующими осям X и Y МСК.
	CoordinateSystem.ByCyclindricalCoordinates Создание объекта CoordinateSystem с заданными параметрами цилиндрических координат относительно заданной системы координат.

Geometry.Cuboid



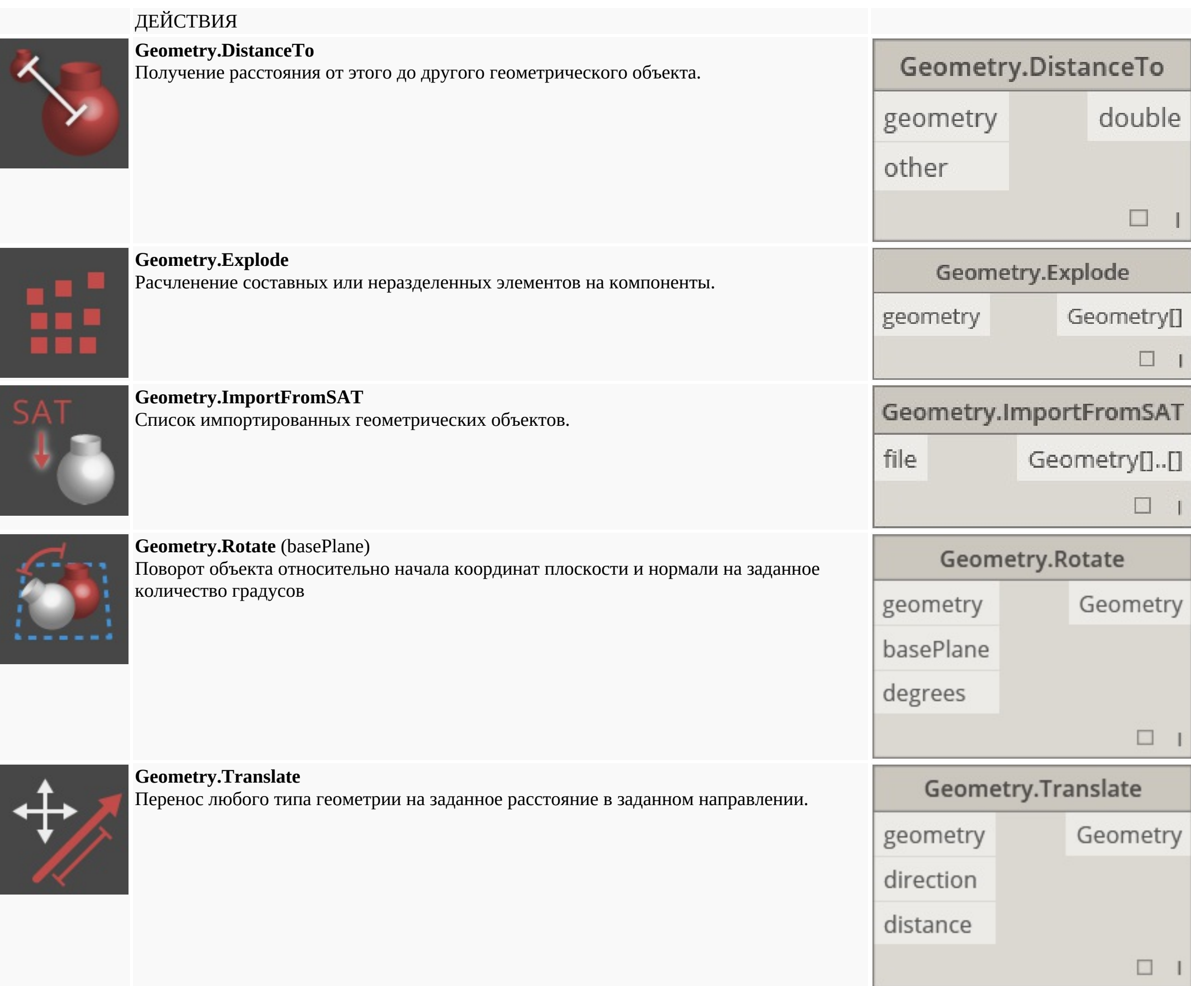
	СОЗДАНИЕ
	Cuboid.ByLength (начало координат) Создание кубоида с центром во входной точке с определенной шириной, длиной и высотой.

Geometry.Curve



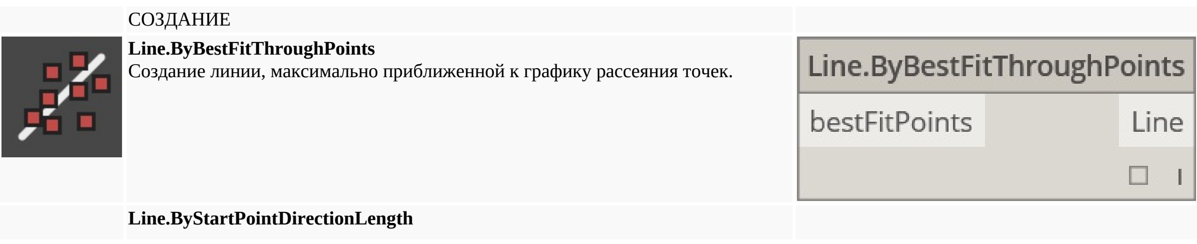
	ДЕЙСТВИЯ	
	Curve.Extrude (расстояние) Выдавливание кривой в направлении вектора нормали.	Curve.Extrude curve → Surface distance
	Curve.PointAtParameter Получение точки на кривой по заданному параметру между StartParameter() и EndParameter().	Curve.PointAtParameter curve → Point param

Geometry.Geometry



	ДЕЙСТВИЯ	
	Geometry.DistanceTo Получение расстояния от этого до другого геометрического объекта.	Geometry.DistanceTo geometry → double other
	Geometry.Explode Расчленение составных или неразделенных элементов на компоненты.	Geometry.Explode geometry → Geometry[]
	Geometry.ImportFromSAT Список импортированных геометрических объектов.	Geometry.ImportFromSAT file → Geometry[]..[]
	Geometry.Rotate (basePlane) Поворот объекта относительно начала координат плоскости и нормали на заданное количество градусов	Geometry.Rotate geometry → Geometry basePlane degrees
	Geometry.Translate Перенос любого типа геометрии на заданное расстояние в заданном направлении.	Geometry.Translate geometry → Geometry direction distance

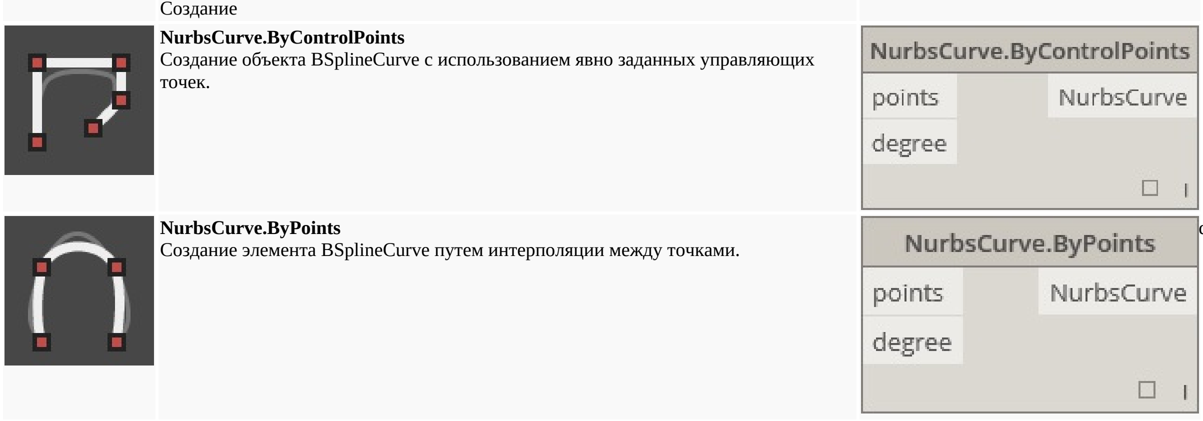
Geometry.Line



	СОЗДАНИЕ	
	Line.ByBestFitThroughPoints Создание линии, максимально приближенной к графику рассеяния точек.	Line.ByBestFitThroughPoints bestFitPoints → Line
	Line.ByStartPointDirectionLength	

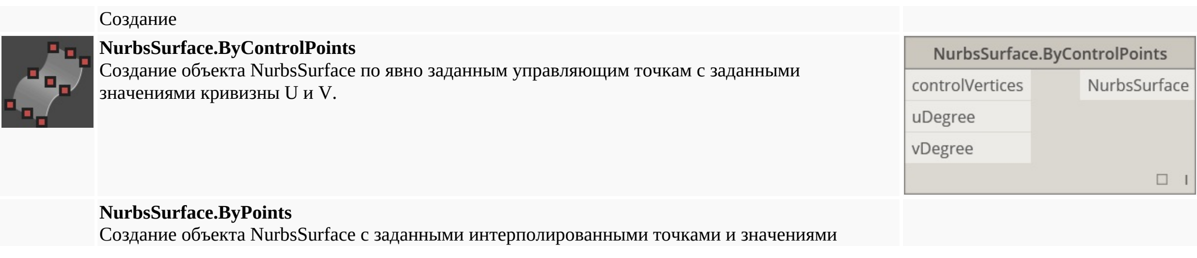
	Построение прямой линии от начальной точки в направлении вектора на заданную длину.	Line.ByStartPointDirectionLength: startPoint, direction, length → Line
	Line.ByStartPointEndPoint Построение прямой линии между двумя заданными точками.	Line.ByStartPointEndPoint: startPoint, endPoint → Line
	Line.ByTangency Создание линии, касательной к исходной кривой, расположенной в точке параметра исходной кривой.	Line.ByTangency: curve, parameter → Line
	ЗАПРОС	
	Line.Direction Направление кривой.	Line.Direction: line → Vector

Geometry.NurbsCurve

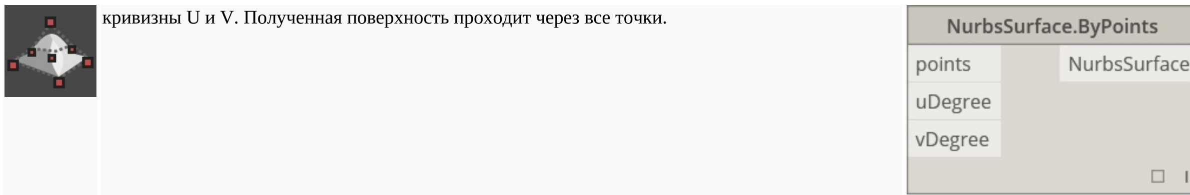


	Создание	
	NurbsCurve.ByControlPoints Создание объекта BSplineCurve с использованием явно заданных управляющих точек.	NurbsCurve.ByControlPoints: points, degree → NurbsCurve
	NurbsCurve.ByPoints Создание элемента BSplineCurve путем интерполяции между точками.	NurbsCurve.ByPoints: points, degree → NurbsCurve

Geometry.NurbsSurface

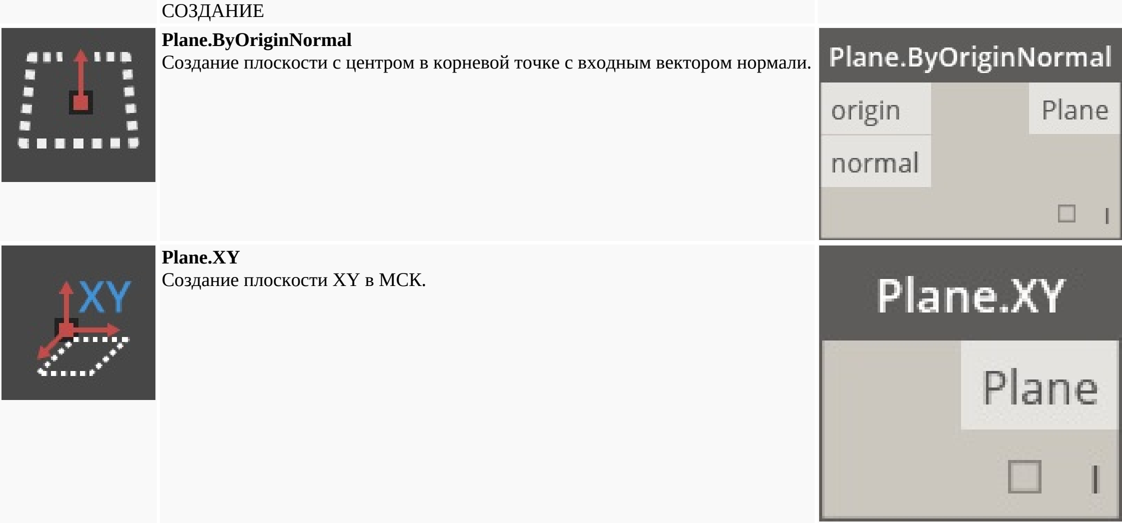


	Создание	
	NurbsSurface.ByControlPoints Создание объекта NurbsSurface по явно заданным управляющим точкам с заданными значениями кривизны U и V.	NurbsSurface.ByControlPoints: controlVertices, uDegree, vDegree → NurbsSurface
	NurbsSurface.ByPoints Создание объекта NurbsSurface с заданными интерполированными точками и значениями	



кривизны U и V. Полученная поверхность проходит через все точки.

Geometry.Plane

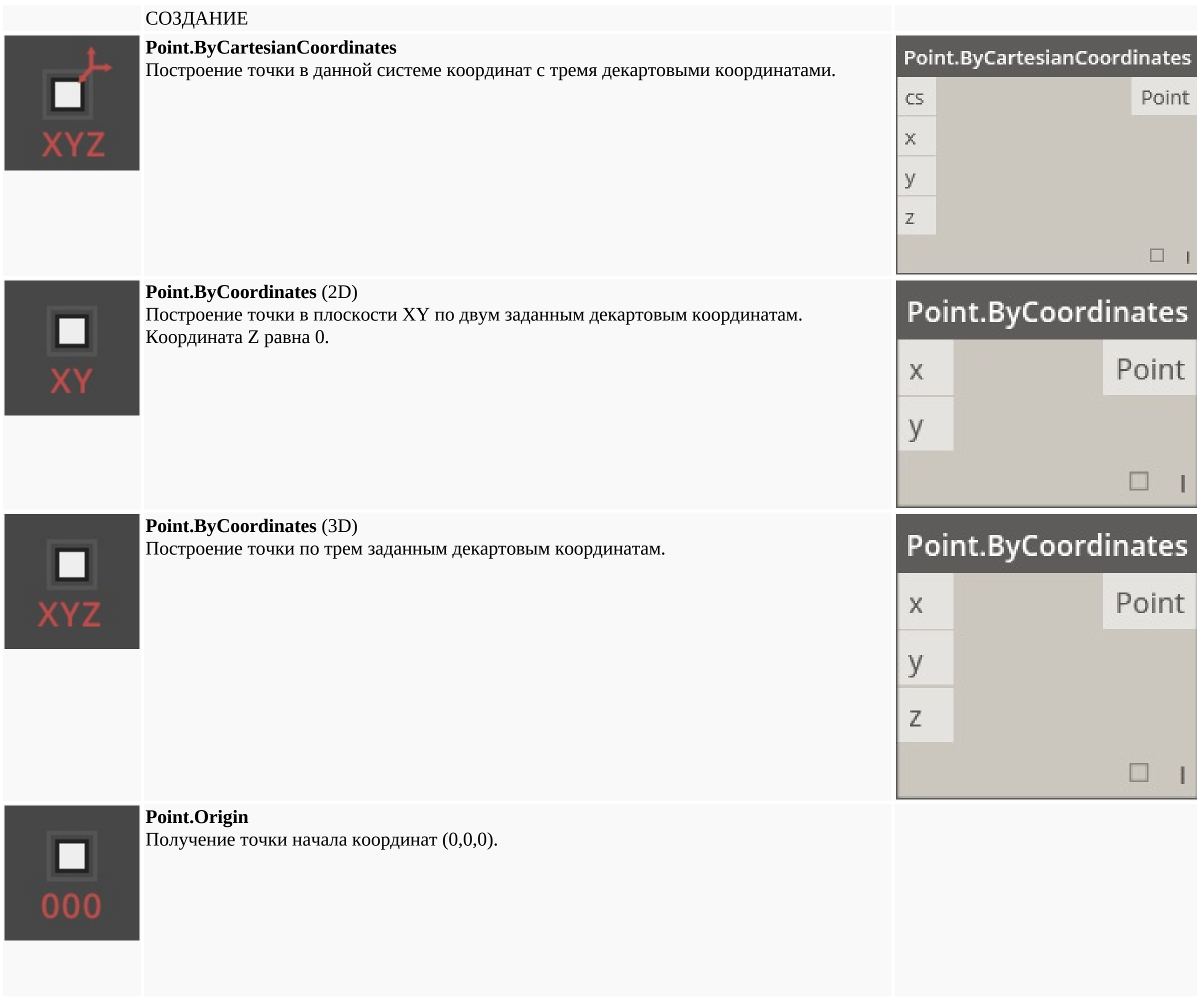


СОЗДАНИЕ

Plane.ByOriginNormal
Создание плоскости с центром в корневой точке с входным вектором нормали.

Plane.XY
Создание плоскости XY в МСК.

Geometry.Point



СОЗДАНИЕ

Point.ByCartesianCoordinates
Построение точки в данной системе координат с тремя декартовыми координатами.

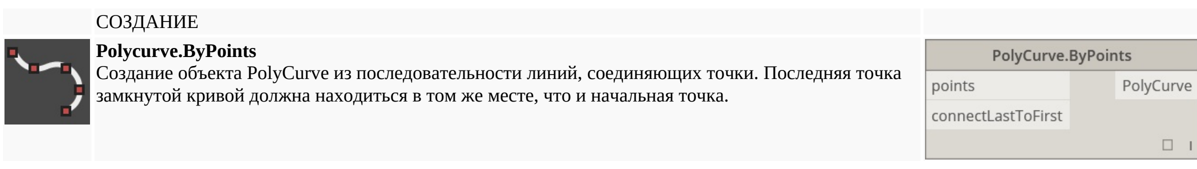
Point.ByCoordinates (2D)
Построение точки в плоскости XY по двум заданным декартовым координатам.
Координата Z равна 0.

Point.ByCoordinates (3D)
Построение точки по трем заданным декартовым координатам.

Point.Origin
Получение точки начала координат (0,0,0).

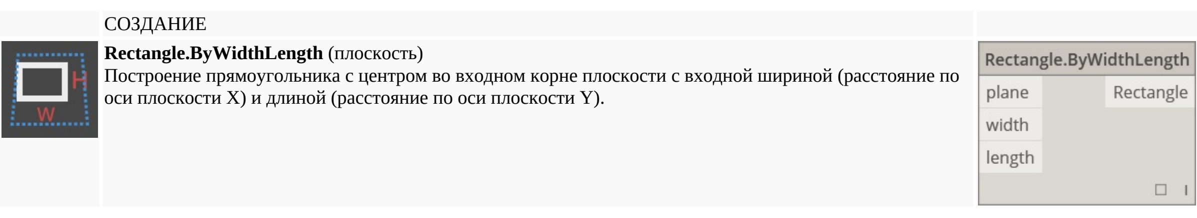
		Point.Origin Point
	ДЕЙСТВИЯ	
	Point.Add Добавление вектора к точке. Аналогично Translate (вектор).	Point.Add point, vectorToAdd → Point
	ЗАПРОС	
X	**Point.X** Получение координаты точки по оси X.	Point.X point → double
Y	**Point.Y** Получение координаты точки по оси Y.	Point.Y point → double
X	**Point.Z** Получение координаты точки по оси Z.	Point.Z point → double

Geometry.Polycurve


СОЗДАНИЕ

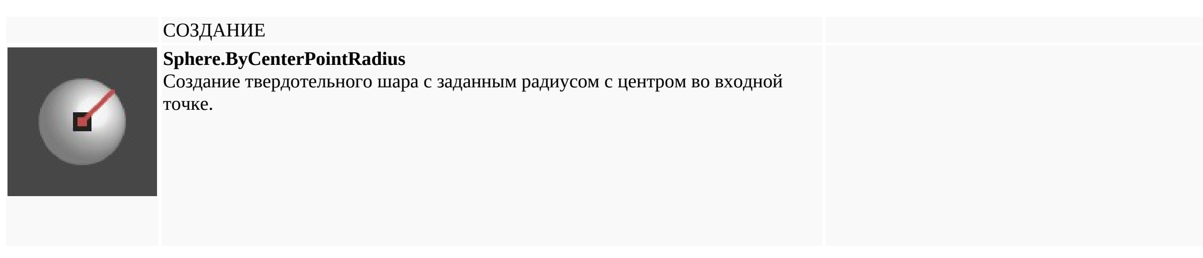
Polycurve.ByPoints
Создание объекта PolyCurve из последовательности линий, соединяющих точки. Последняя точка замкнутой кривой должна находиться в том же месте, что и начальная точка.

Geometry.Rectangle

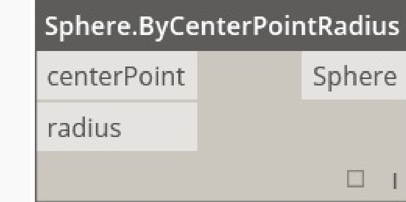

СОЗДАНИЕ

Rectangle.ByWidthLength (плоскость)
Построение прямоугольника с центром во входном корне плоскости с входной шириной (расстояние по оси плоскости X) и длиной (расстояние по оси плоскости Y).

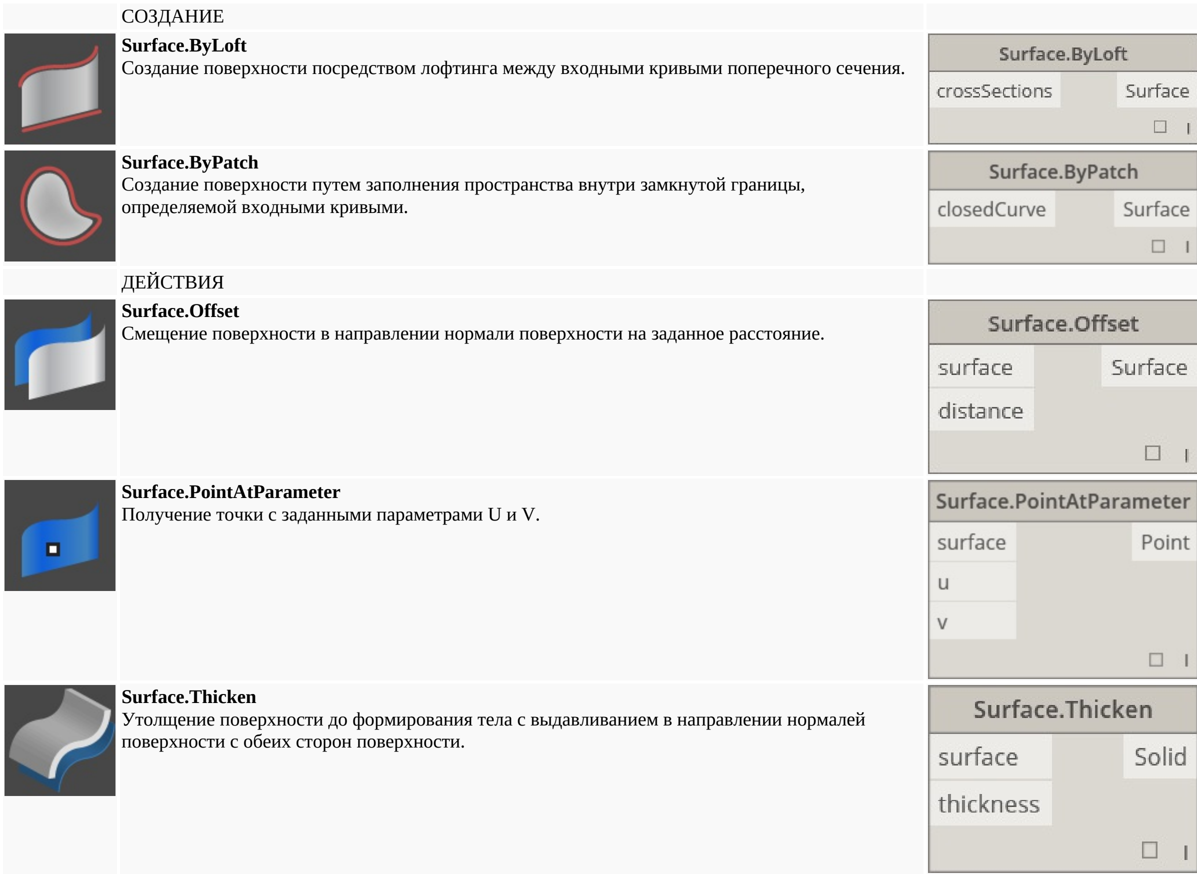
Geometry.Sphere


СОЗДАНИЕ

Sphere.ByCenterPointRadius
Создание твердотельного шара с заданным радиусом с центром во входной точке.

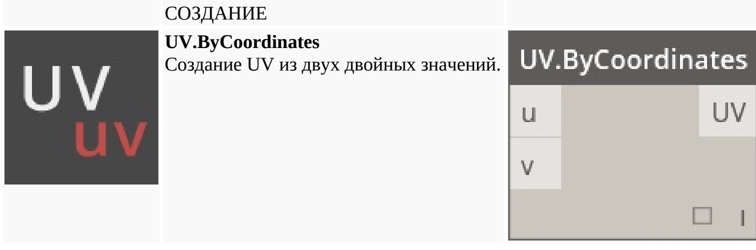


Geometry.Surface



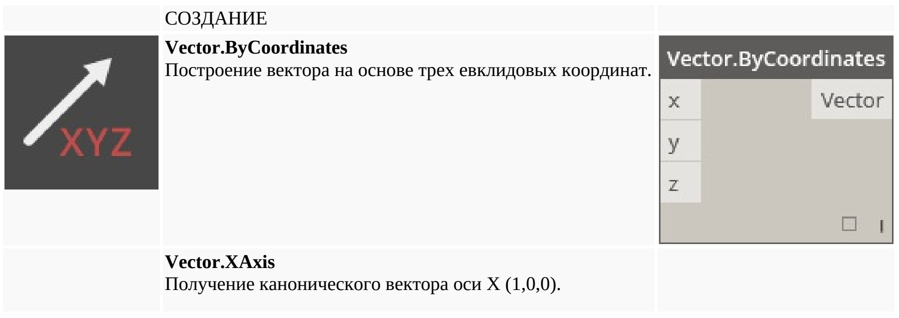
	СОЗДАНИЕ	
	Surface.ByLoft Создание поверхности посредством лофтинга между входными кривыми поперечного сечения.	Surface.ByLoft crossSections → Surface
	Surface.ByPatch Создание поверхности путем заполнения пространства внутри замкнутой границы, определяемой входными кривыми.	Surface.ByPatch closedCurve → Surface
	ДЕЙСТВИЯ	
	Surface.Offset Смещение поверхности в направлении нормали поверхности на заданное расстояние.	Surface.Offset surface, distance → Surface
	Surface.PointAtParameter Получение точки с заданными параметрами U и V.	Surface.PointAtParameter surface, u, v → Point
	Surface.Thicken Утолщение поверхности до формирования тела с выдавливанием в направлении нормалей поверхности с обеих сторон поверхности.	Surface.Thicken surface, thickness → Solid

Geometry.UV

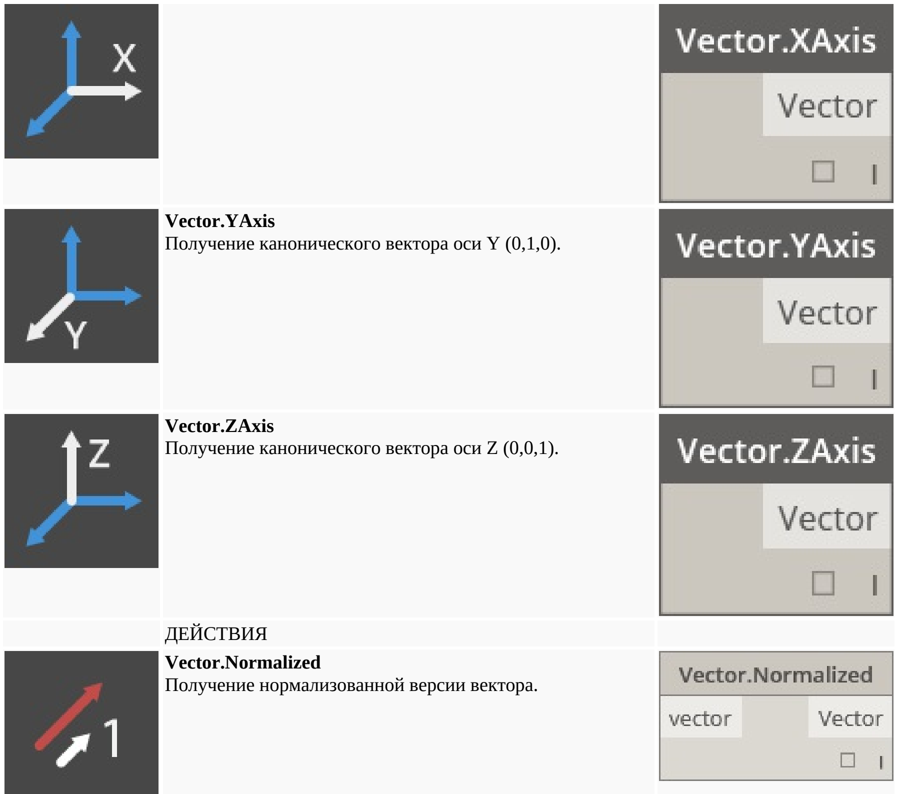


	СОЗДАНИЕ	
UV uv	**UV.ByCoordinates** Создание UV из двух двойных значений.	UV.ByCoordinates u, v → UV

Geometry.Vector



	СОЗДАНИЕ	
XYZ	**Vector.ByCoordinates** Построение вектора на основе трех евклидовых координат.	Vector.ByCoordinates x, y, z → Vector
	Vector.XAxis Получение канонического вектора оси X (1,0,0).	



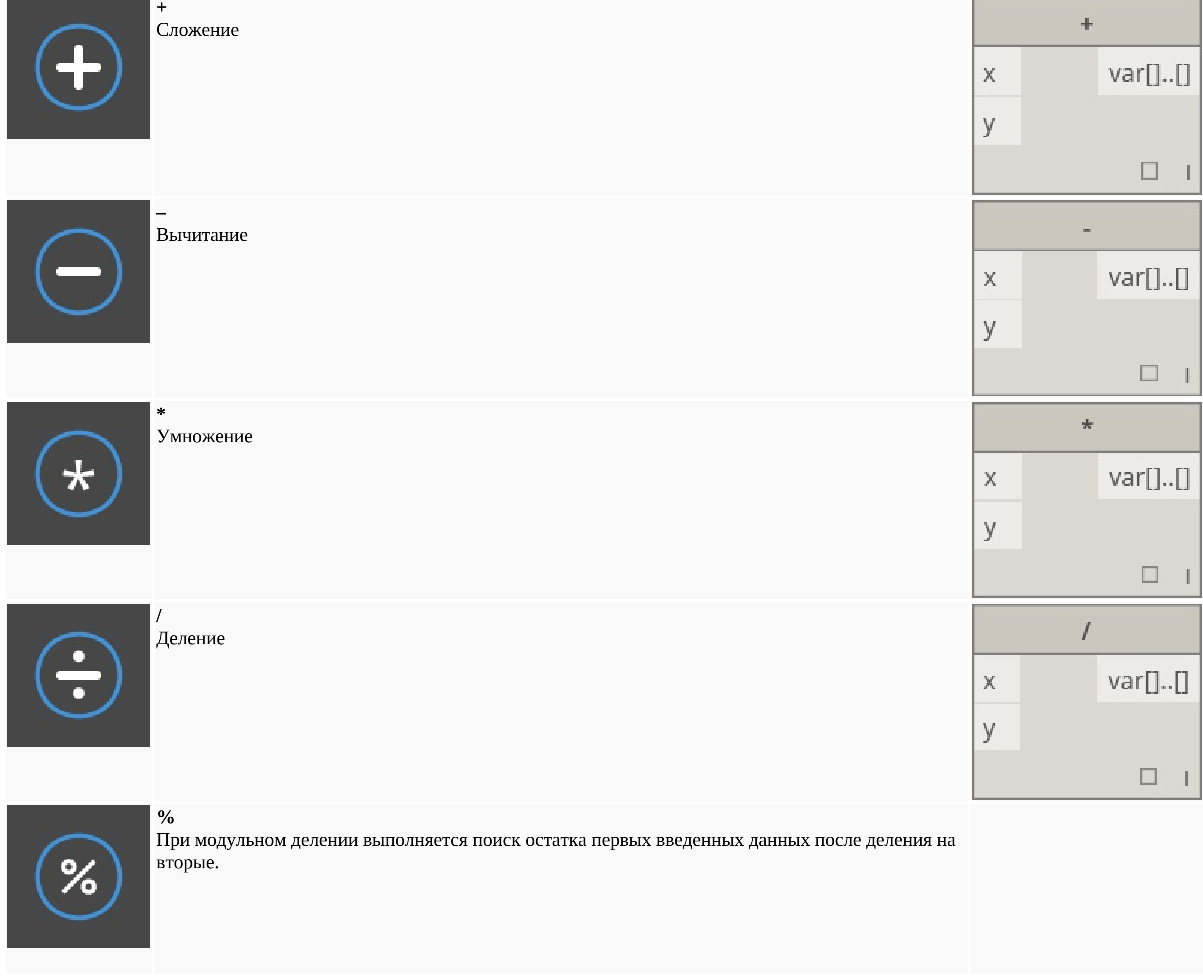
Vector.YAxis
Получение канонического вектора оси Y (0,1,0).

Vector.ZAxis
Получение канонического вектора оси Z (0,0,1).

ДЕЙСТВИЯ

Vector.Normalized
Получение нормализованной версии вектора.

Операторы



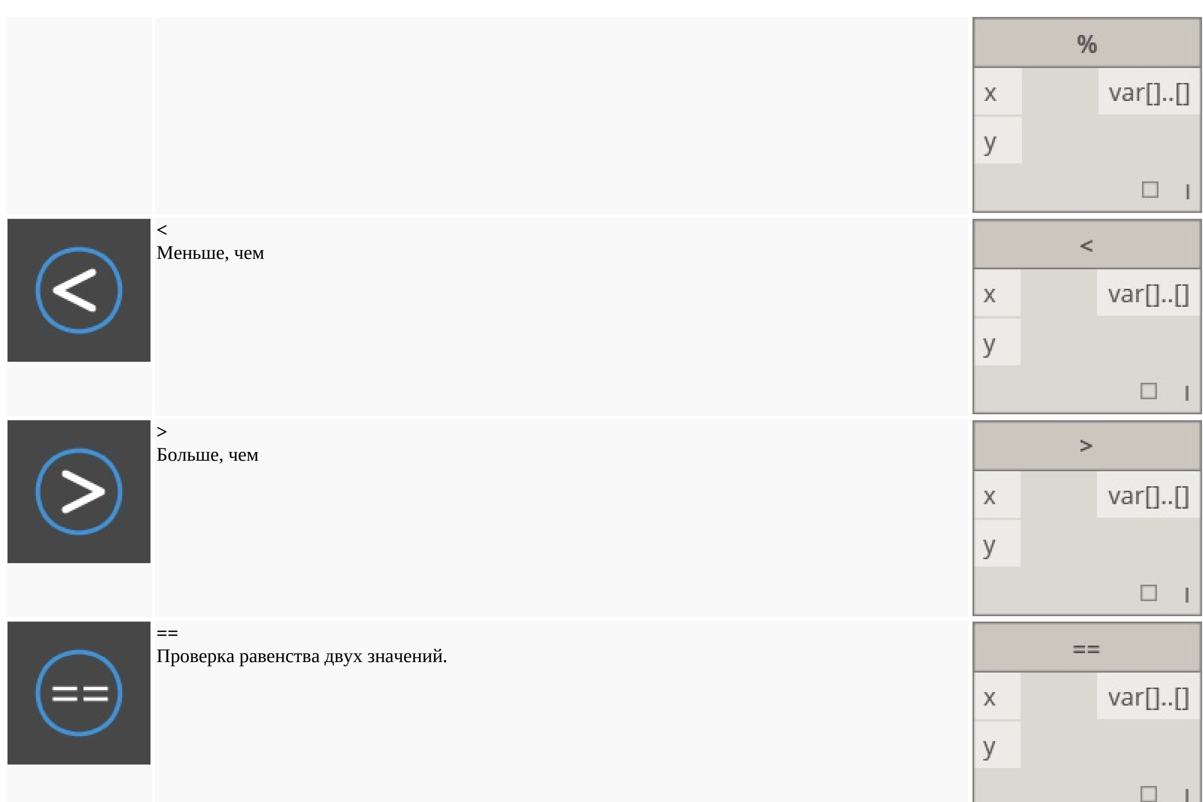
+
Сложение

–
Вычитание

*****
Умножение

/
Деление

%
При модульном делении выполняется поиск остатка первых введенных данных после деления на вторые.



		% x y var[]..[]
	< Меньше, чем	< x y var[]..[]
	> Больше, чем	> x y var[]..[]
	== Проверка равенства двух значений.	== x y var[]..[]

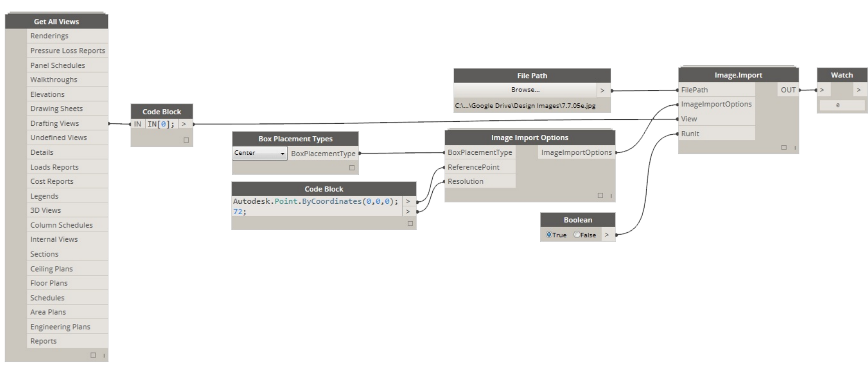
Полезные пакеты

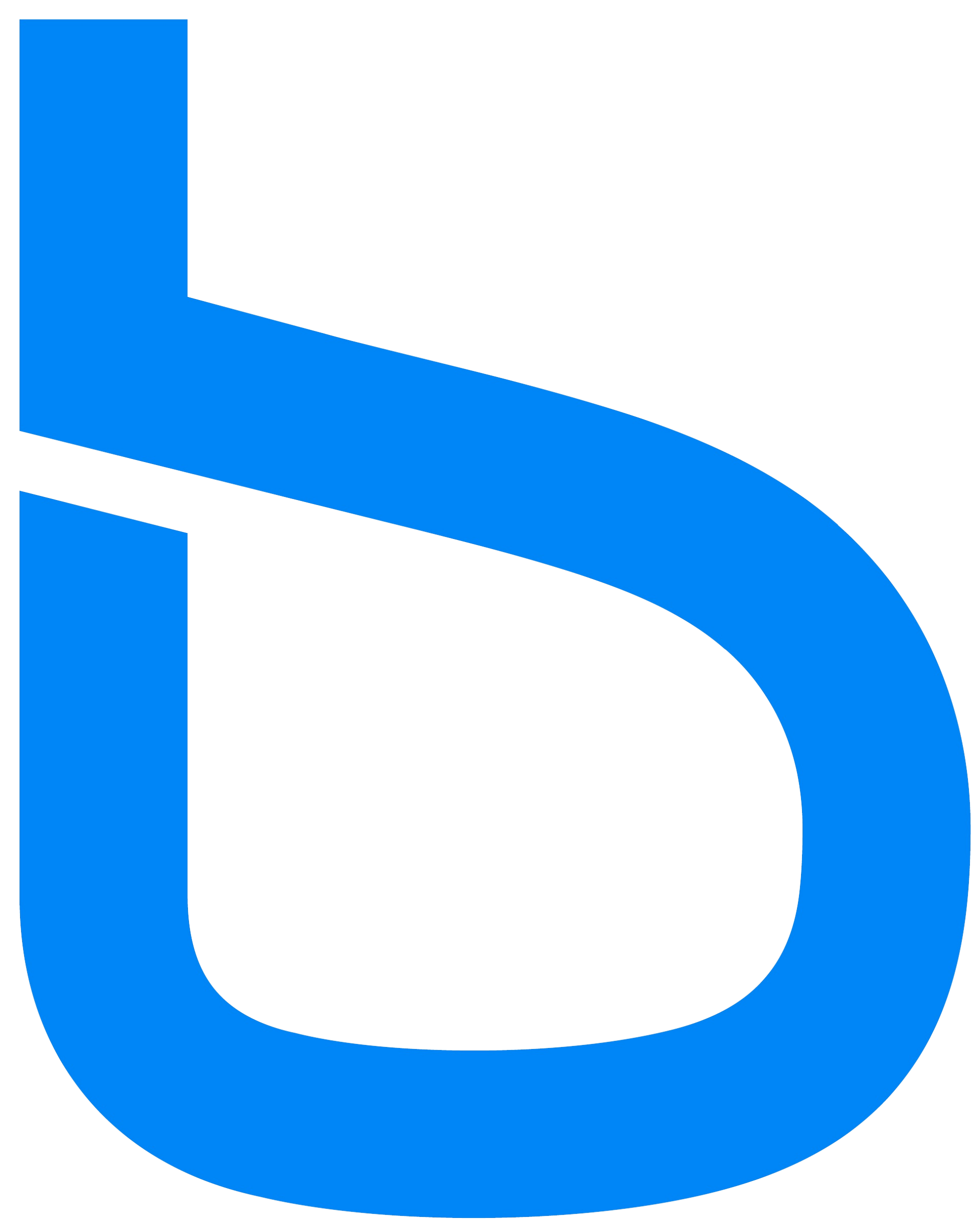
Пакеты Dynamo

Ниже приведен список пакетов, наиболее популярных среди пользователей Dynamo. Разработчики, не стесняйтесь пополнять этот список. Помните, что Dynamo Primer — это ресурс с открытым с исходным кодом.

ARCHI-LAB Официальный сайт archi-lab

archi-lab — это более 50 пользовательских пакетов, которые позволяют существенно расширить возможности взаимодействия Dynamo с Revit. В пакетах archi-lab доступны как узлы с наборами базовых операций, так и узлы визуальной среды расчетов для Revit.

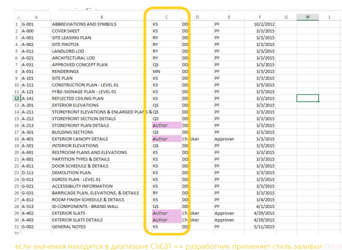




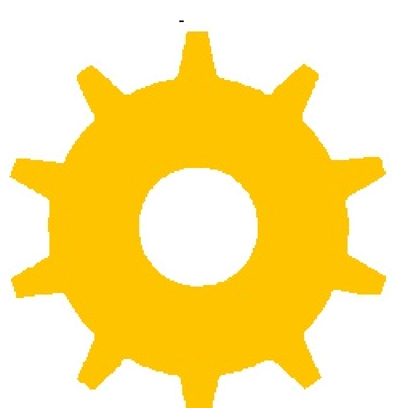
BimorphNodes — это универсальная коллекция мощных вспомогательных узлов. Среди них можно найти высокоэффективные узлы для выявлени
конфликтов и управления пересечениями геометрии, узлы преобразования кривых ImportInstance (САПР) и средства для сбора связанных элемен
решающие проблему ограничений в API Revit. Чтобы подробнее узнать о всех доступных узлах, ознакомьтесь с каталогом BimorphNodes.

BUMBLEBEE FOR DYNAMO [Официальный сайт BumbleBee]

Bumblee — это подключаемый модуль для обеспечения взаимодействия между Excel и Dynamo, значительно расширяющий возможности Dynamo в плане чтения и записи файлов Excel.

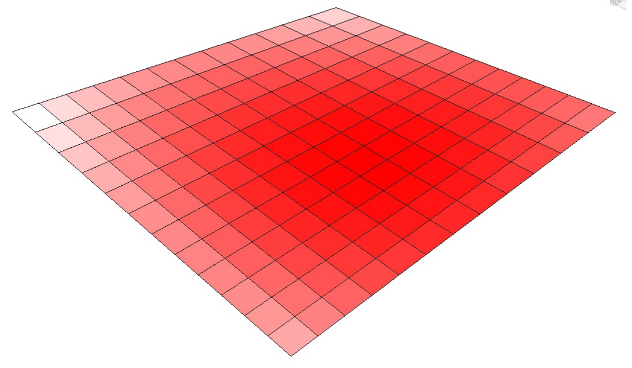


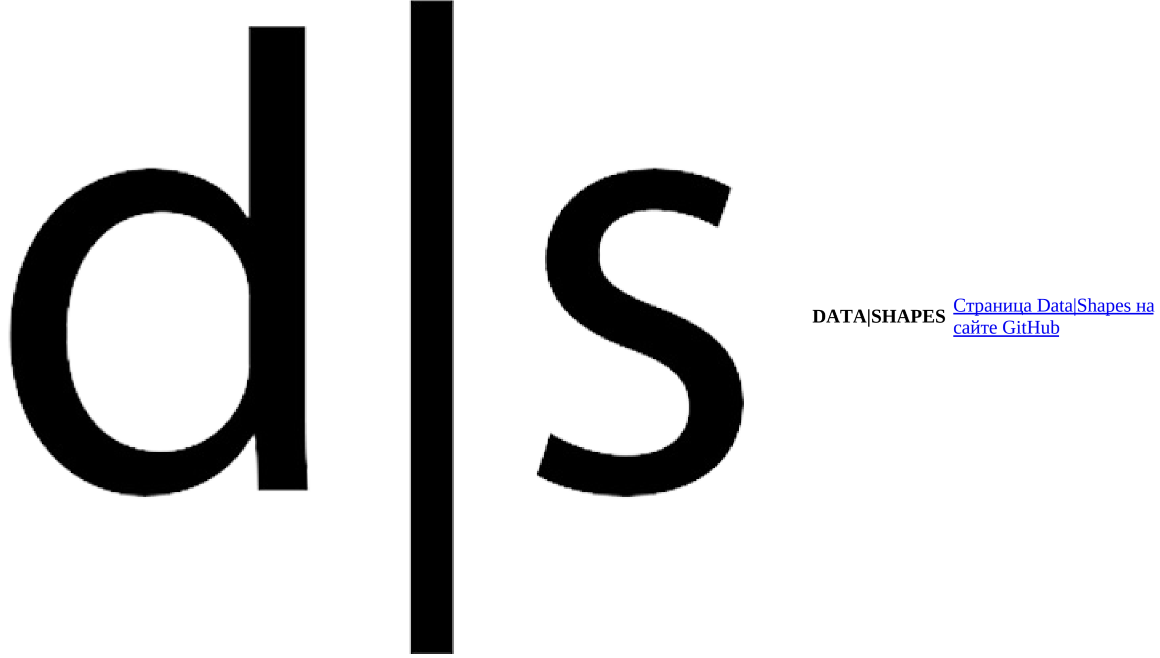
	A	B	C	D	E	F	G	H	I
1	G-001	ABBREVIATIONS AND SYMBOLS	KS	DD	PF	10/2/2012			
2	A-000	COVER SHEET	KS	DD	PF	3/3/2015			
3	A-001	SITE LEASING PLAN	RY	DD	PF	3/3/2015			
4	A-002	SITE PHOTOS	RY	DD	PF	3/3/2015			
5	A-011	LANDLORD LOD	RY	DD	PF	3/3/2015			
6	A-021	ARCHITECTURAL LOD	RY	DD	PF	3/3/2015			
7	A-031	APPROVED CONCEPT PLAN	QS	DD	PF	3/3/2015			
8	A-051	RENDERINGS	MN	DD	PF	3/3/2015			
9	A-101	SITE PLAN	KS	DD	PF	3/3/2015			
10	A-111	CONSTRUCTION PLAN - LEVEL 01	KS	DD	PF	3/3/2015			
11	A-121	FF&E-SIGNAGE PLAN - LEVEL 01	KS	DD	PF	3/3/2015			
12	A-141	REFLECTED CEILING PLAN	KS	DD	PF	3/3/2015			
13	A-201	EXTERIOR ELEVATIONS	QS	DD	PF	3/3/2015			
14	A-211	STOREFRONT ELEVATIONS & ENLARGED PLANS &	QS	DD	PF	3/3/2015			
15	A-212	STOREFRONT SECTION DETIALS	QS	DD	PF	3/3/2015			
16	A-213	STOREFRONT PLAN DETAILS	Author	DD	PF	3/3/2015			
17	A-301	BUILDING SECTIONS	QS	DD	PF	3/3/2015			
18	A-401	EXTERIOR CANOPY DETAILS	Author	Checker	Approver	3/3/2015			
19	A-501	INTERIOR ELEVATIONS	QS	DD	PF	3/3/2015			
20	A-691	RESTROOM PLANS AND ELEVATIONS	KS	DD	PF	3/3/2015			
21	A-801	PARTITION TYPES & DETAILS	KS	DD	PF	3/3/2015			
22	A-811	DOOR SCHEDULE & DETAILS	KS	DD	PF	3/3/2015			
23	D-111	DEMOLITION PLAN	KS	DD	PF	3/3/2015			
24	G-011	EGRESS PLAN - LEVEL 01	KS	DD	PF	3/3/2015			
25	G-021	ACCESSIBILITY INFORMATION	KS	DD	PF	3/3/2015			
26	G-031	BARRICADE PLAN, ELEVATIONS, & DETAILS	RY	DD	PF	3/3/2015			
27	A-812	ROOM FINISH SCHEDULE & DETAILS	KS	DD	PF	3/4/2015			
28	A-513	ID COMPONENTS - BRAND WALL	QS	DD	PF	4/1/2015			
29	A-402	EXTERIOR SLATS	Author	Checker	Approver	4/29/2015			
30	A-403	EXTERIOR SLATS DETAILS	Author	Checker	Approver	4/29/2015			
31	G-002	GENERAL NOTES	KS	DD	PF	5/11/2015			
32									



CLOCKWORK FOR DYNAMO [Страница Clockwork For Dynamo на сайте GitHub]

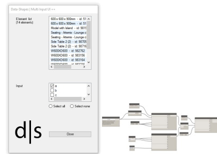
Clockwork — это набор пользовательских узлов для среды визуального программирования Dynamo. В нем представлено множество узлов для работы с Revit и решения других задач, например управления списками, математических операций, строковых операций, преобразования единиц измерения, геометрических операций (ограничивающие рамки, сетки, плоскости, точки, поверхности, UV и векторы) и разбивки на панели.





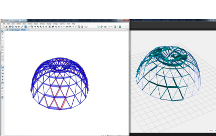
DATA|SHAPES [Страница Data|Shapes на сайте GitHub]

DataShapes — это пакет для расширения пользовательских функций в сценариях Dynamo. Основной целью пакета является увеличение спектра функциональных возможностей проигрывателя Dynamo. Дополнительные сведения см. на странице https://data-shapes.net/. Если вам нужно создать мощные рабочие процессы для проигрывателя Dynamo, обратите внимание на этот пакет.



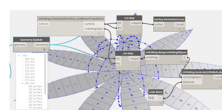
DYNAMO SAP	[Проект DynamoSAP на сайте Core Studio]

DynamoSAP — это параметрический интерфейс для SAP2000, встраиваемый в Dynamo. Этот проект позволяет инженерам осуществлять генеративное проектирование и анализ строительных систем в SAP, используя Dynamo для управления моделью SAP. Проект содержит несколько типовых рабочих процессов, описанных в прилагаемых файлах примеров, а также предоставляет возможности для автоматизации типовых задач в SAP.



DYNAMO UNFOLD [Страница DynamoUnfold на сайте GitHub]

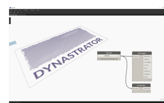
Эта библиотека позволяет расширить функциональные возможности Dynamo/Revit за счет развертывания геометрии поверхностей и сложных поверхностей. Библиотека позволяет пользователям сначала преобразовывать поверхности в плоскую мозаичную топологию, а затем выполнить их развертку с помощью инструментов Protogeometry в Dynamo. В этом пакете также содержится несколько экспериментальных узлов и файлов с простыми примерами.



[Страница загрузки]

DYNASTRATOR [сайте Package Manager]

Импортируйте векторные изображения из Illustrator или из интернета в формате SVG. Этот инструмент позволяет импортировать созданные вручную чертежи в модуль Dynamo для параметрических операций.



ENERGY ANALYSIS [Проект Energy Analysis]

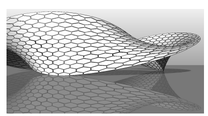
FOR DYNAMO for Dynamo на сайте GitHub

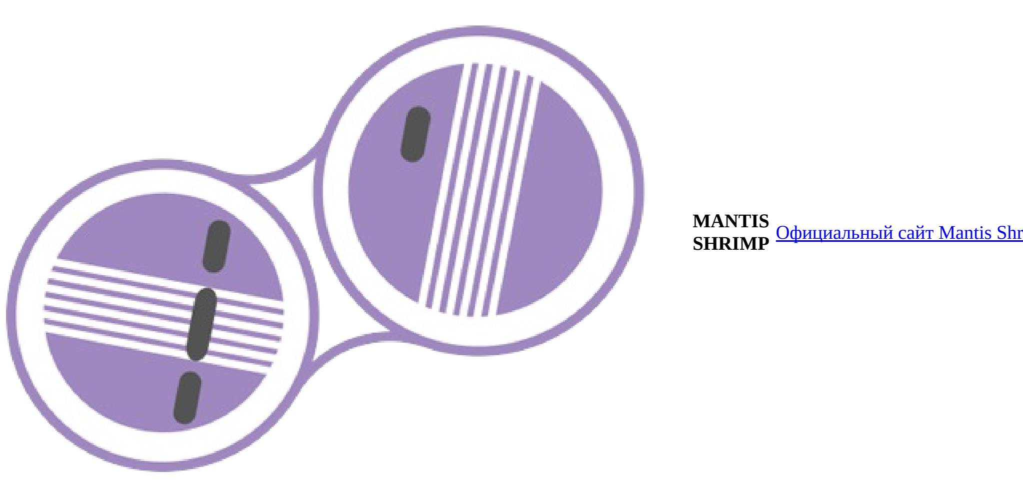
Energy Analysis for Dynamo позволяет выполнять параметрическое моделирование энергопотребления и создавать рабочие процессы для расчета энергопотребления всего здания в Dynamo 0.8. Пакет Energy Analysis for Dynamo позволяет настроить модель энергопотребления в Autodesk Revit, отправить ее в Green Building Studio для расчета энергопотребления DOE2 и изучить результаты, полученные после расчета. Пакет разрабатывается компанией Thornton Tomasetti в рамках проекта CORE Studio.

Firefly — это набор узлов, позволяющих Dynamo обмениваться данными с устройствами ввода/вывода, такими как микроконтроллер Arduino. По поток данных передается в режиме реального времени, Firefly открывает множество возможностей для интерактивного создания прототипов на с между цифровыми и физическими системами с помощью веб-камер, мобильных телефонов, игровых контроллеров, датчиков и т. д.

LUNCHBOX FOR DYNAMO [Страница загрузки Lunchbox for Dynamo на сайте Proving Ground]

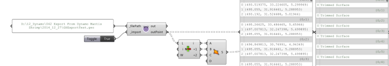
LunchBox — это набор узлов для управления повторно используемыми геометрическими объектами и данными. Инструменты были протестированные в Dynamo 0.8.1 и Revit 2016. В пакет входят узлы для разбивки поверхности на панели, работы с геометрией, сбора данных Revit и многого другого.

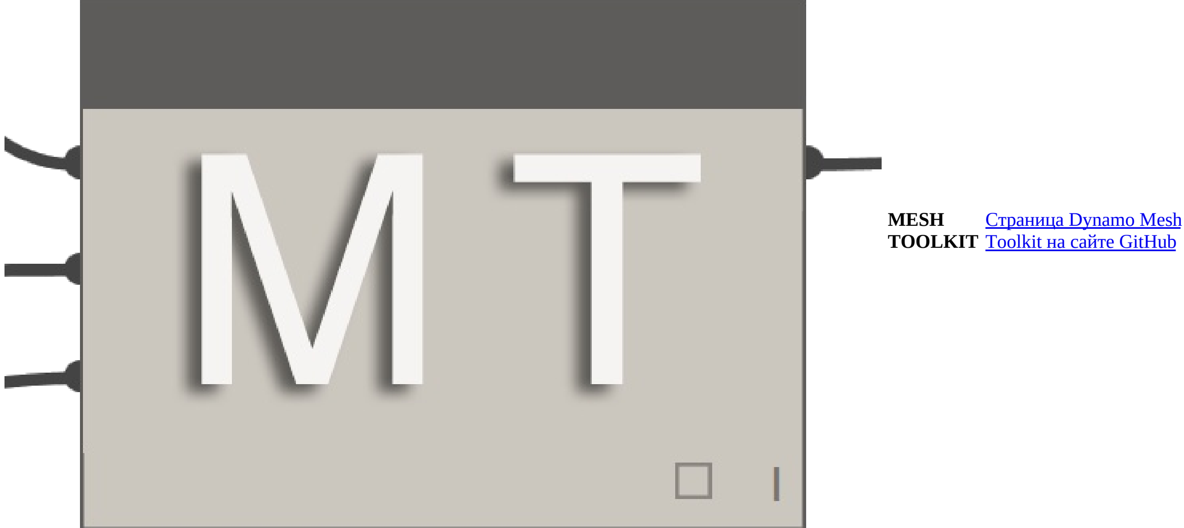




MANTIS SHRIMP [Официальный сайт Mantis Shrimp]

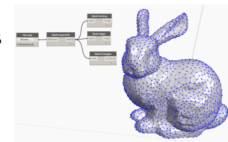
Mantis Shrimp — это проект по поддержке совместимости, который позволяет легко импортировать геометрию Grasshopper и/или Rhino в Dynamo.





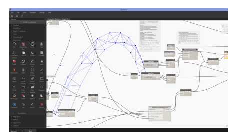
MESH TOOLKIT [Страница Dynamo Mesh Toolkit на сайте GitHub]

Инструментарий Dynamo Mesh Toolkit содержит множество полезных инструментов для работы с геометрией сети. В этом пакете имеются возможности для импорта сетей из внешних файлов в других форматах, формирования сетей из существующих геометрических объектов Dynamo и построения сетей вручную на основе данных о вершинах и соединениях. Кроме этого, в пакет входят инструменты для изменения и восстановления геометрии сети.



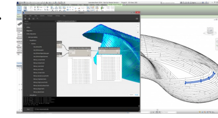
OPTIMO [Страница Optimo на сайте GitHub]

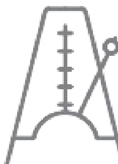
Optimo обеспечивает пользователей Dynamo возможностями для оптимизации решения самостоятельно поставленных проектных задач с помощью различных адаптируемых алгоритмов. Пользователи могут определять не только одну задачу или несколько задач, но и отдельные функции пригодности.



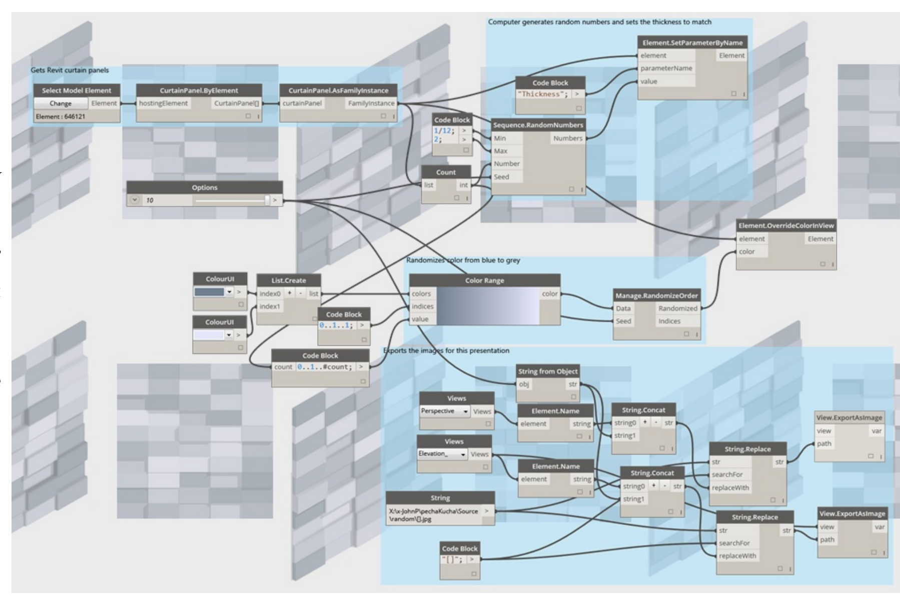
RHYNAMO [Страница Rhynamo на сайте Bitbucket]

Библиотека узлов Rhynamo предоставляет пользователям возможность чтения и записи файлов Rhino 3DM из Dynamo. Rhynamo преобразует геометрию Rhino в пригодную к использованию геометрию Dynamo с помощью библиотеки OpenNURBS от компании McNeel, позволяя создавать новые рабочие процессы для беспрепятственного обмена геометрией и данными между Rhino и Revit. Этот пакет также содержит несколько экспериментальных узлов, которые обеспечивают прямой доступ к командной строке Rhino.





RHYTHM [Страница Rhythm на сайте GitHub]



На первый взгляд Rhythm ничем не выделяется среди других пакетов. У него нет ни сверхсложного кода, ни других подобных свойств. Однако Rhythm — это плод упорного труда, направленного на достижение практических целей. Идея, лежащая в основе этого пакета, — позволить пользователям работать с Rhythm в Revit с помощью модуля Dynamo. Основу Rhythm составляют готовые узлы Dynamo, используемые особым образом в процессе применения к среде Revit.

Файлы примеров

Файлы примеров Dynamo

Эти файлы примеров дополняют руководство Dynamo Primer. Они упорядочены по главам и разделам.

Щелкните нужный файл правой кнопкой мыши и выберите «Сохранить ссылку как...».

Введение

Раздел	Файл для скачивания
Что такое визуальное программирование	Visual Programming - Circle Through Point.dyn

Структура определения Dynamo

Раздел	Файл для скачивания
Наборы параметров	Presets.dyn

Компоновочные блоки программ

Раздел	Файл для скачивания
Данные	Building Blocks of Programs - Data.dyn
Математика	Building Blocks of Programs - Math.dyn
Логика	Building Blocks of Programs - Logic.dyn
Строки	Building Blocks of Programs - Strings.dyn
Цвет	Building Blocks of Programs - Color.dyn

Геометрия для машинного проектирования

Раздел	Файл для скачивания
Обзор концепции геометрии	Geometry for Computational Design - Geometry Overview.dyn
Векторы	Geometry for Computational Design - Vectors.dyn
	Geometry for Computational Design - Plane.dyn
	Geometry for Computational Design - Coordinate System.dyn
Точки	Geometry for Computational Design - Points.dyn
Кривые	Geometry for Computational Design - Curves.dyn
Поверхности	Geometry for Computational Design - Surfaces.dyn
	Surface.sat

Проектирование на основе списков

Раздел	Файл для скачивания
Что такое список	Lacing.dyn
Работа со списками	List-Count.dyn
	List-FilterByBooleanMask.dyn
	List-GetItemAtIndex.dyn
	List-Operations.dyn
	List-Reverse.dyn
	List-ShiftIndices.dyn
Списки списков	Chop.dyn
	Combine.dyn
	Flatten.dyn
	Map.dyn
	ReplaceItems.dyn
	Top-Down-Hierarchy.dyn
	Transpose.dyn
Многомерные списки	n-Dimensional-Lists.dyn
	n-Dimensional-Lists.sat

Узлы Code Block и DesignScript

Раздел	Файл для скачивания
Синтаксис DesignScript	Dynamo-Syntax_Attractor-Surface.dyn
Сокращение	Obsolete-Nodes_Sine-Surface.dyn
Функции	Functions_SphereByZ.dyn

Dynamo для Revit

Раздел	Файл для скачивания
Выбор	Selecting.dyn
	ARCH-Selecing-BaseFile.rvt
Редактирование	Editing.dyn
	ARCH-Editing-BaseFile.rvt
Создание	Creating.dyn
	ARCH-Creating-BaseFile.rvt
Адаптация	Customizing.dyn
	ARCH-Customizing-BaseFile.rvt
Выпуск документации	Documenting.dyn
	ARCH-Documenting-BaseFile.rvt

Словари в Dynamo

Раздел	Файл для скачивания
Словарь помещений	RoomDictionary.dyn

Пользовательские узлы

Раздел	Файл для скачивания
Создание пользовательских узлов	UV-CustomNode.zip
Публикация узлов в библиотеку	PointsToSurface.dyf
Узлы Python	Python-CustomNode.dyn
Python и Revit	Revit-Doc.dyn
Python и Revit	Revit-ReferenceCurve.dyn
Python и Revit	Revit-StructuralFraming.zip

Пакеты

Раздел	Файл для скачивания
Практикум по работе с пакетом: Mesh Toolkit	MeshToolkit.zip
Публикация пакетов	MapToSurface.zip
Импорт Zero Touch	ZeroTouchImages.zip