

STRUKTUR DATA

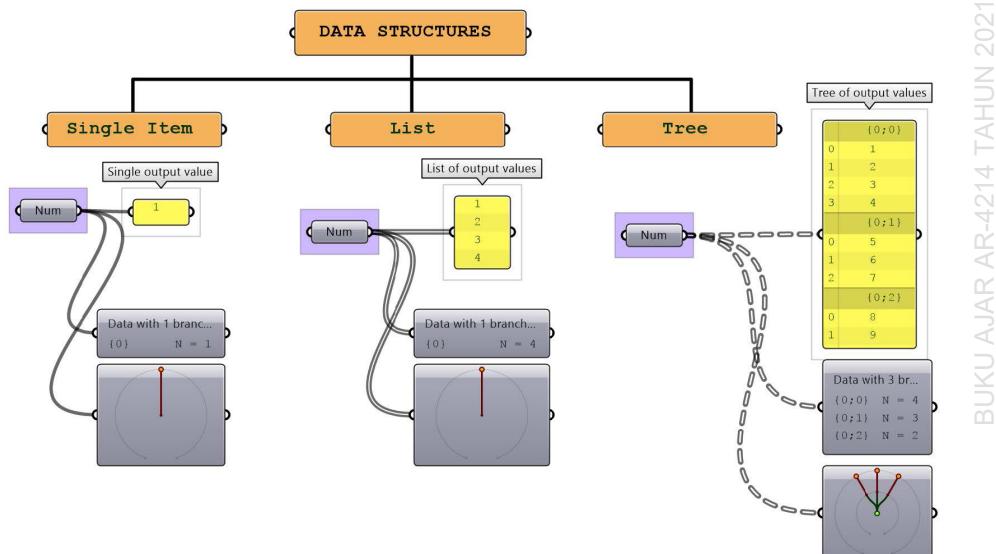
List
Tree
Data matching
Data filtering

5. STRUKTUR DATA

File latihan di bab ini dapat diunduh di:
<https://github.com/aswinindra/eskacangmerah>

Semua algoritma menyangkut pemrosesan data input untuk menghasilkan output. Data ini disimpan dengan sistem dan mekanisme tertentu yang dinamakan struktur data agar proses yang menyangkut akses dan manipulasi data dapat dilakukan secara efisien. Pemahaman atas struktur data ini menjadi sangat esensial bagi desain algoritmik.

Grasshopper memiliki tiga jenis struktur data: **Single Item**, **List**, **Tree**. Komponen dan proses di dalam GH sangat tergantung dari input data pada tiga jenis struktur data ini sehingga sangat penting kita tahu apa jenis struktur data sebelum memprosesnya karena jika tidak, maka akan terjadi error. Komponen untuk memperlihatkan struktur data adalah: **Panel** dan **Param Viewer**.



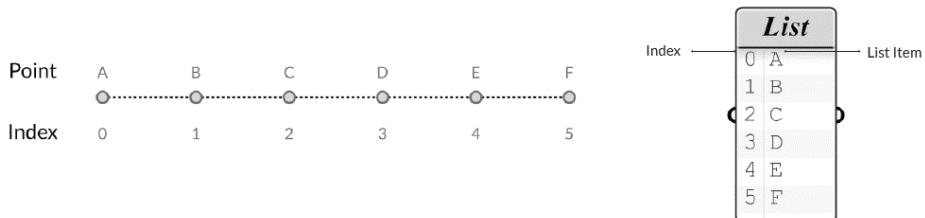
Gambar 99. Struktur Data di Grasshopper. (Sumber: Rajaa Isaa, 2020)

- Pemrosesan atas input data tergantung dari struktur datanya.
- Single dan List itu adalah data yang terdiri dari satu atau beberapa, namun hanya dalam satu himpunan/satu cabang. Sedangkan Tree adalah data yang terdiri dari satu atau beberapa dan terdiri dari beberapa himpunan data/cabang.

5.1. List

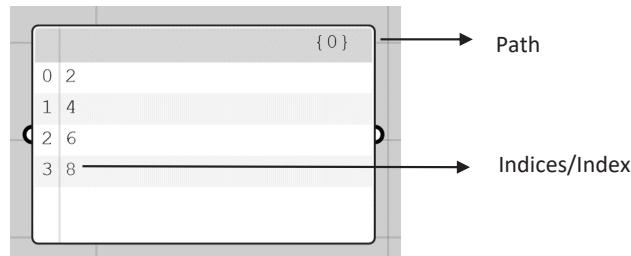
Semua data dalam GH disimpan dalam bentuk *array* atau *list*. List adalah sistem penyimpanan data dimana setiap data memiliki ‘tanda pengenal’ berupa nomor indeks. Lokasi atau posisi data dinyatakan dengan nomor indeksnya. Nomor indeks ini unik

sehingga tidak ada data yang memiliki dua nomor indeks atau satu nomor indeks memiliki dua data. Proses memanipulasi data menggunakan nomor index, bukan mengakses datanya secara langsung.



Gambar 100. Data Dalam List. (Sumber: Gil Akos & Ronie Parsons, 2017)

- Ketetapan GH, setiap nomor indeks dimulai dengan **0**, bukan **1** dan data yang disimpan dalam setiap nomor index tidak hanya angka, namun bisa Point, Curve, Surface dan sebagainya.
- Pada komponen Panel, jika kita ingin menampilkan data suatu komponen, kita bisa menampilkan nomor index: **Draw Indices** dan menampilkan struktur datanya: **Draw Path**.

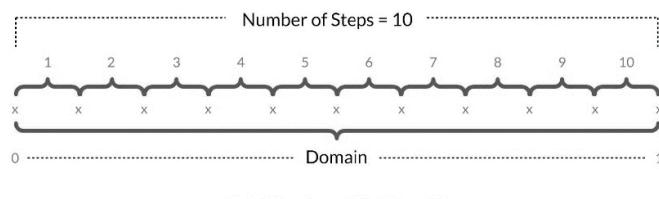


Gambar 101. Index, Data dan Path. (Sumber: Gil Akos & Ronie Parsons, 2017)

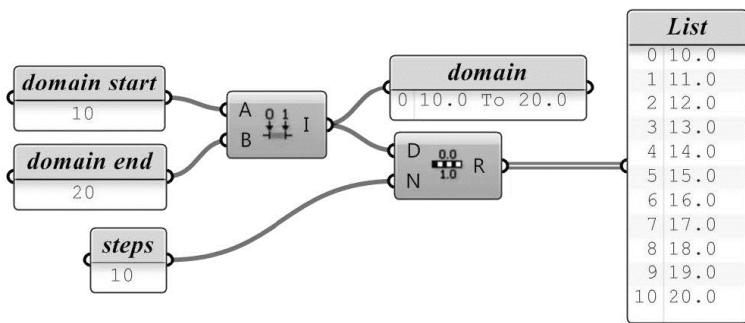
5.1.1. Membuat List

Dalam bab sebelumnya sudah dijelaskan beberapa cara membuat list mulai dari secara manual memasukkan data pada komponen **Panel** (dengan Multiline Data) atau menggunakan komponen seperti **Series**, **Range**, **Random**.

Range adalah komponen yang menghasilkan list dari sejumlah angka yang intervalnya sama, antara nilai minimum dan nilai maksimum.

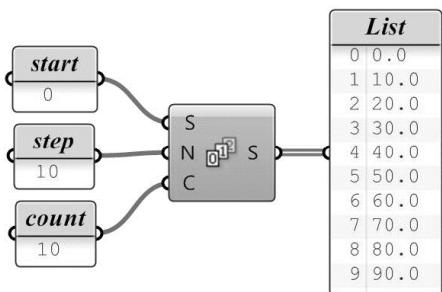


Gambar 102. Range. (Sumber: Gil Akos & Ronie Parsons, 2017)



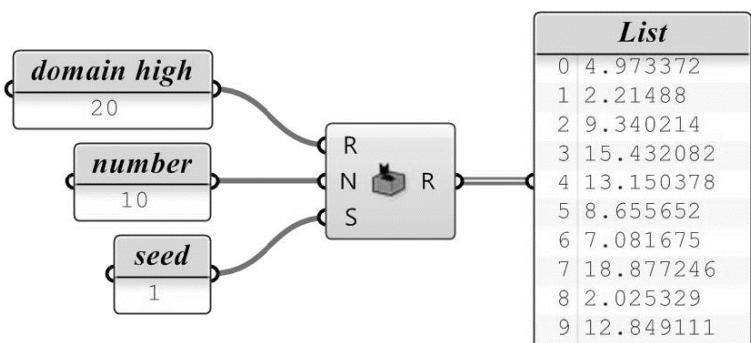
Gambar 103. Input dan Output Komponen Range.(Sumber: Gil Akos & Ronie Parsons, 2017)

Series adalah komponen yang menghasilkan list dari sejumlah angka dengan interval tertentu yang dibatasi oleh jumlah maksimum total angka yang dihasilkan.



Gambar 104. Series. (Sumber: Gil Akos & Ronie Parsons, 2017)

Random adalah komponen yang menghasilkan list sejumlah angka yang acak. Parameter *seed* menentukan ke-acakan angka yang dihasilkan. Harap diperhatikan, parameter *seed* men-generasi nilai random pada satu list, dan tidak pada path yang berbeda.



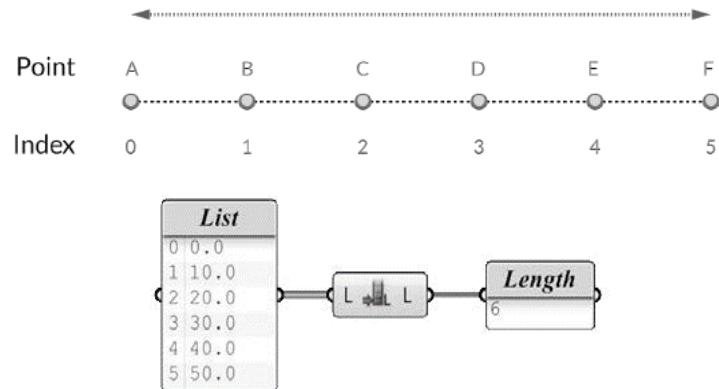
Gambar 105. Random. (Sumber: Gil Akos & Ronie Parsons, 2017)

Selain dihasilkan dari number generator, list juga didapatkan dari operasi, misalnya komponen **Divide Curve** yang menghasilkan list berupa koordinat.

5.1.2. Operasi List / Manajemen List

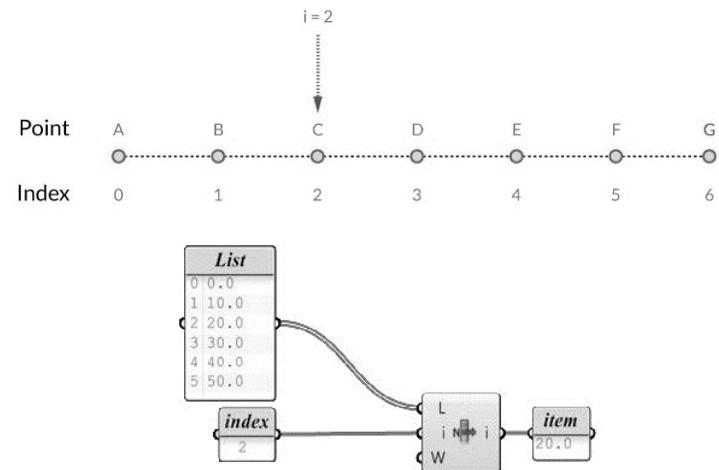
Secara default, ada beberapa komponen untuk operasi list dan manajemen list. Ingatlah bahwa kita tidak dapat mengakses data secara langsung, melainkan melalui operasi pada *list* atau nanti pada *path* (struktur data).

List Length: mengukur panjang list. Karena nomor indeks selalu dimulai dari **nol**, maka panjang suatu list adalah **nilai maksimum nomor indeks ditambah 1**. Panjang list juga dapat diartikan *berapa banyak data yang ada pada list tersebut*. Hasil dari komponen ini adalah angka.



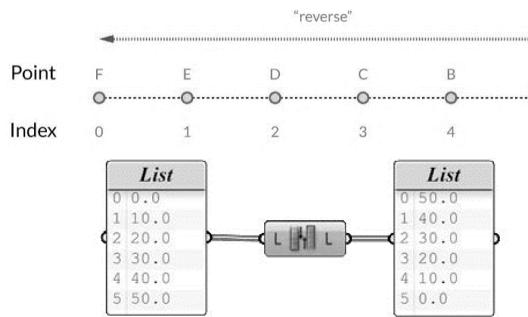
Gambar 106. List Length. (Sumber: Gil Akos & Ronie Parsons, 2017)

List Item: mengakses atau mengekstrak data pada suatu list menggunakan nomor indeksnya (*i*, input)



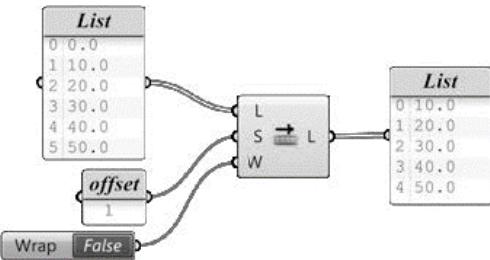
Gambar 107. List Item. (Sumber: Gil Akos & Ronie Parsons, 2017)

Reverse List: membalik urutan/ nomor indeks suatu list. Data dari nomor indeks 0 menjadi data dengan nomor indeks terakhir dan seterusnya.



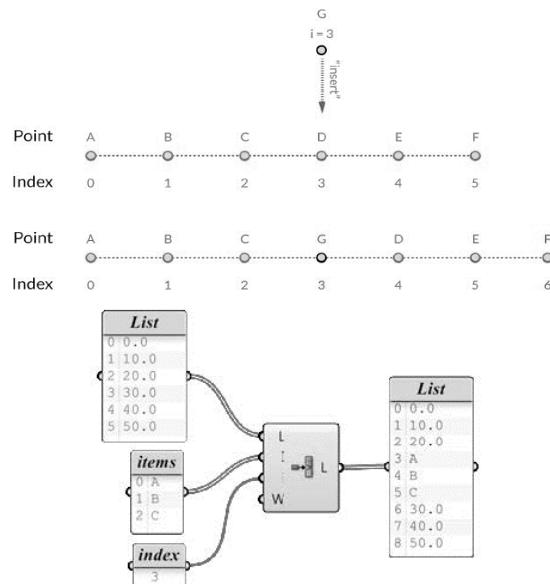
Gambar 108. Reverse List. (Sumber: Gil Akos & Ronie Parsons, 2017)

Shift List: menggeser nomor indeks maju atau mundur. Nilai shift [+] berarti nomor index akan naik sejumlah nilai shift, sedangkan bila nilai shift [-], nomor indeks akan turun sejumlah nilai shift.



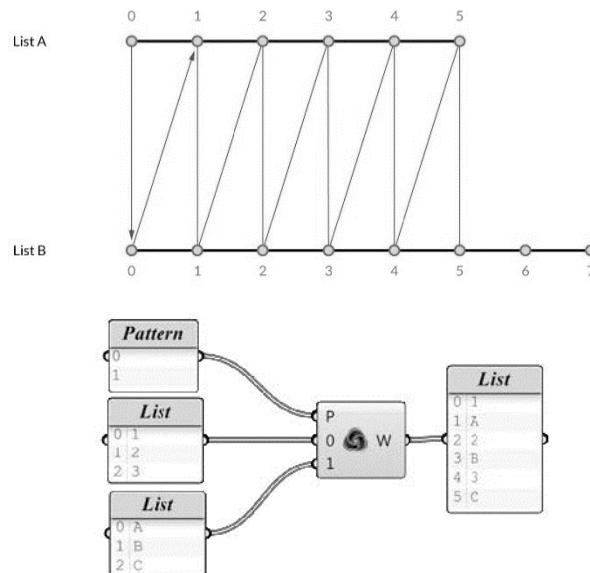
Gambar 109. Shift List. (Sumber: Gil Akos & Ronie Parsons, 2017)

Insert Item: memasukkan data pada suatu list yang ditentukan dari nomor indeks dimulainya pemasukan data tersebut.



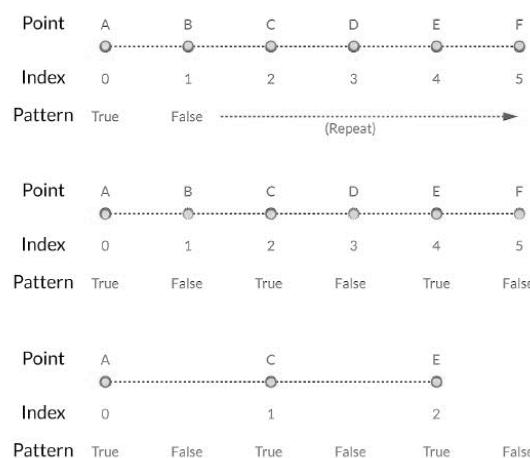
Gambar 110. Insert Item. (Sumber: Gil Akos & Ronie Parsons, 2017)

Weave: menggabungkan dua atau lebih list dengan aturan pola tertentu (parameter P)



Gambar 111. Weave. (Sumber: Gil Akos & Ronie Parsons, 2017)

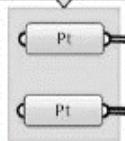
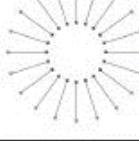
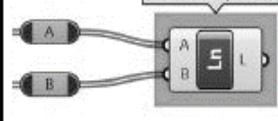
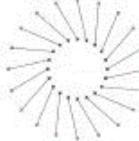
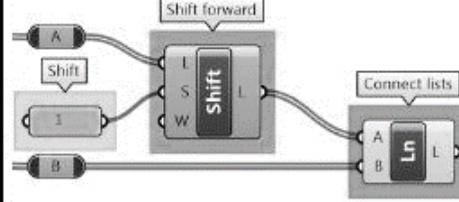
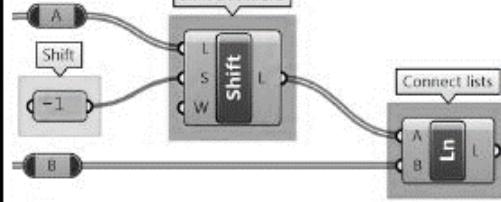
Cull Pattern: menyembunyikan/ tidak menampilkan beberapa data pada suatu list berdasarkan pola yang ditentukan menggunakan Boolean (True/ False).

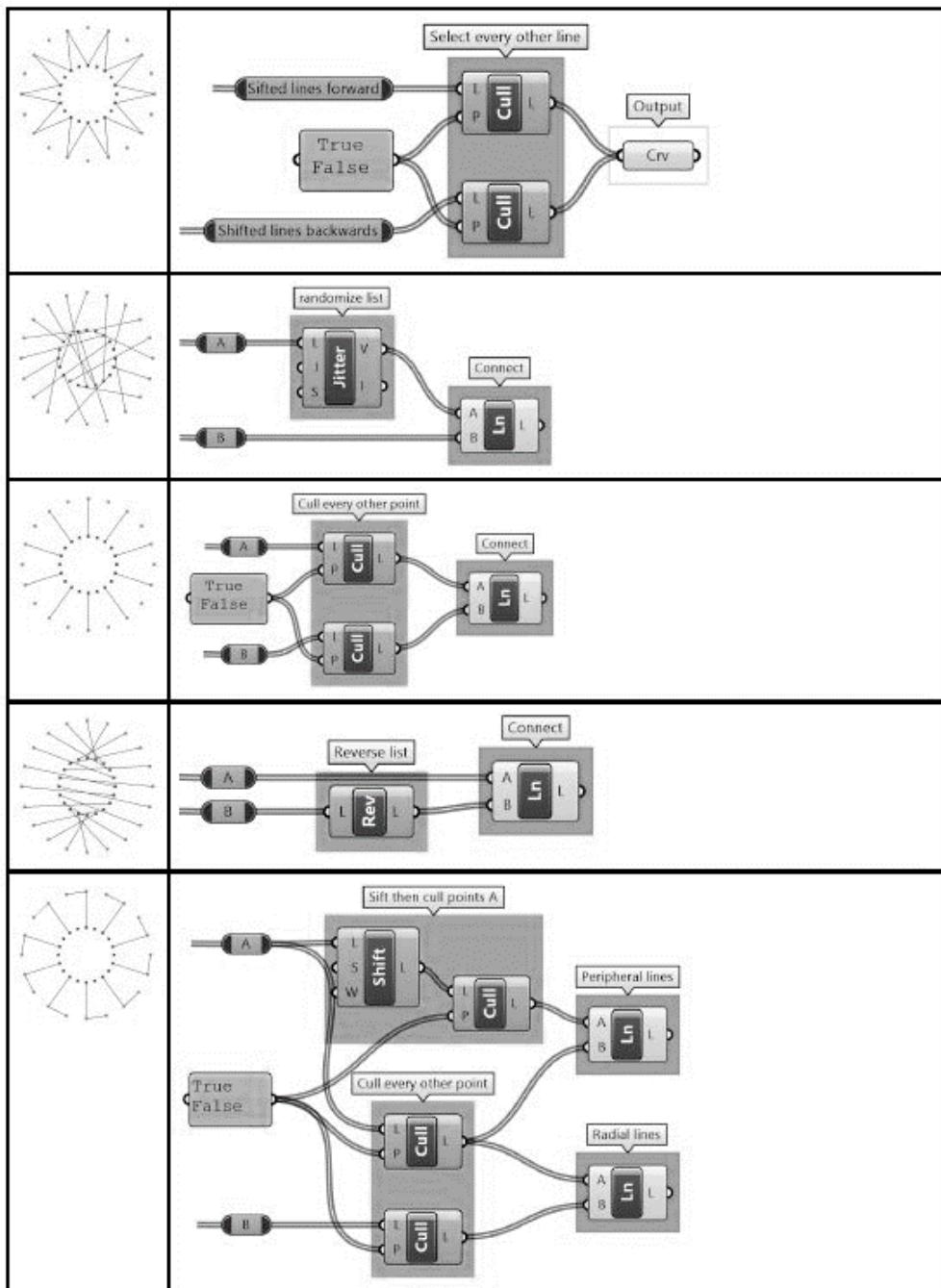


Gambar 112. Cull Pattern.(Sumber: Gil Akos & Ronie Parsons, 2017)

Latihan 7: Dasar-Dasar List

1. Buatlah beberapa definisi sesuai dengan panduan berikut untuk menghasilkan beberapa pola.

Output image	Grasshopper solution
	<p>Input lists A and B</p> 
	<p>Connect input lists</p> 
	<p>Shift forward</p>  <p>Connect lists</p>
	<p>Shift backward</p>  <p>Connect lists</p>

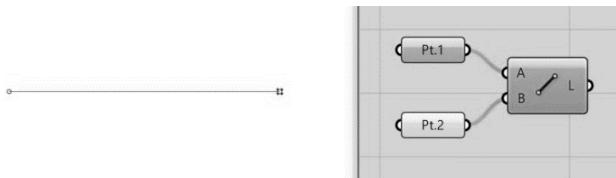


Gambar 113. Latihan Dasar-Dasar List (Sumber: Rajaa Isaa, 2020)

5.2. List Matching/ Data Stream Matching

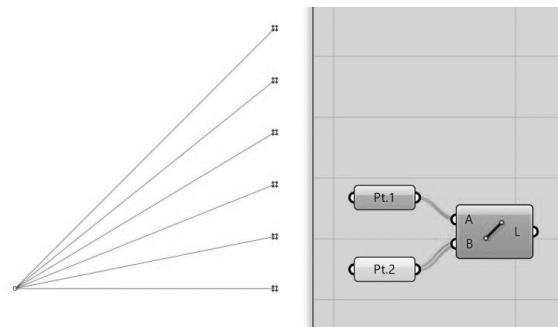
Setiap komponen menerima input data dan memprosesnya sesuai dengan fungsi komponen tersebut dan jenis data yang diproses. Dalam interaksi antara input, proses dan output ada tiga jenis interaksi yang terjadi:

- 1. Satu data ke satu data:** umumnya menghasilkan output yang diinginkan.



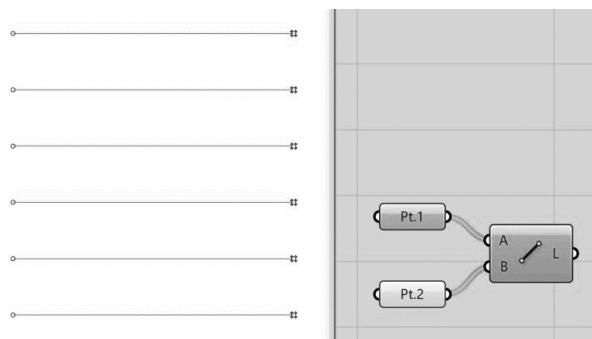
Gambar 114. Relasi Satu-Satu.

- 2. Satu data ke banyak data:** secara default GH menerapkan prinsip *longest list* artinya, keluaran proses sesuai dengan list terpanjang (yang memiliki data terbanyak). Proses ini umumnya juga menghasilkan keluaran yang diinginkan.



Gambar 115. Relasi Satu-Banyak

- 3. Banyak data ke banyak data:** jika kedua data memiliki panjang list yang sama, maka GH akan memproses berdasarkan **nomor indeks yang sama** antara kedua list tersebut. Proses ini terkadang menghasilkan keluaran yang tidak sesuai dengan yang diinginkan, terutama jika panjang kedua list tidak sama.

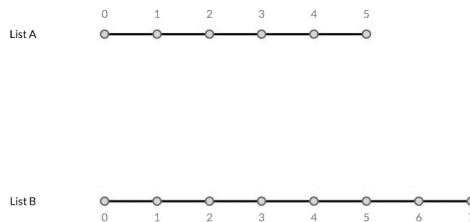


Gambar 116. Relasi Banyak-Banyak Dengan Panjang List Sama

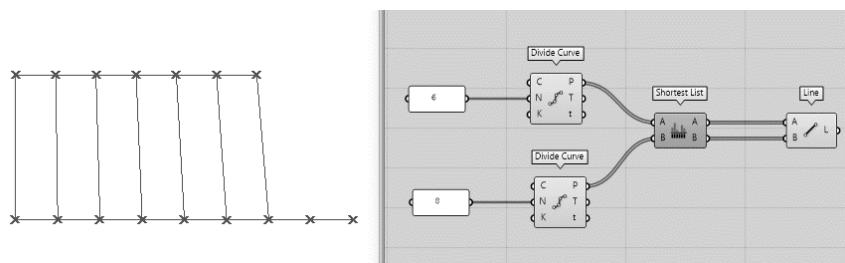
Dalam beberapa kasus, proses mencocokkan data (*data matching*) terjadi karena sebuah komponen harus mengakses data yang berbeda ukurannya (panjang list berbeda). Mengubah algoritma bagaimana sebuah komponen menngakses data akan menghasilkan output yang berbeda.

Untuk memahami konsep *Data Matching*, perhatikan contoh berikut.

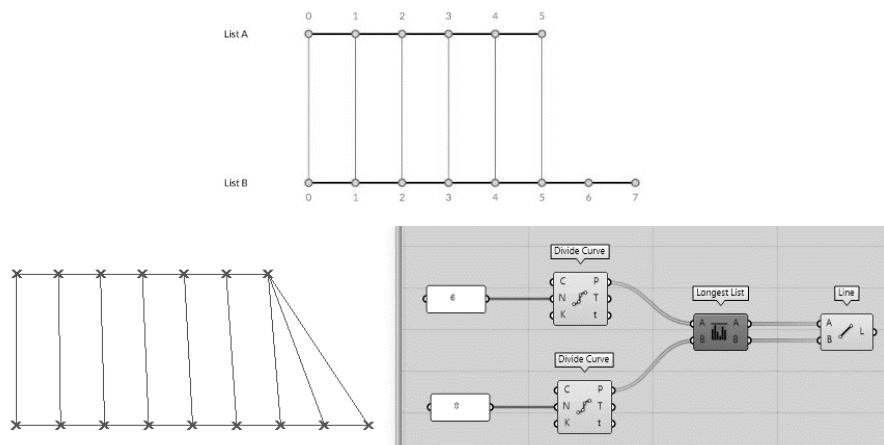
Ada dua jenis input: sebuah list dengan 6 data Point (list A) dan list dengan 8 data Point (list B).



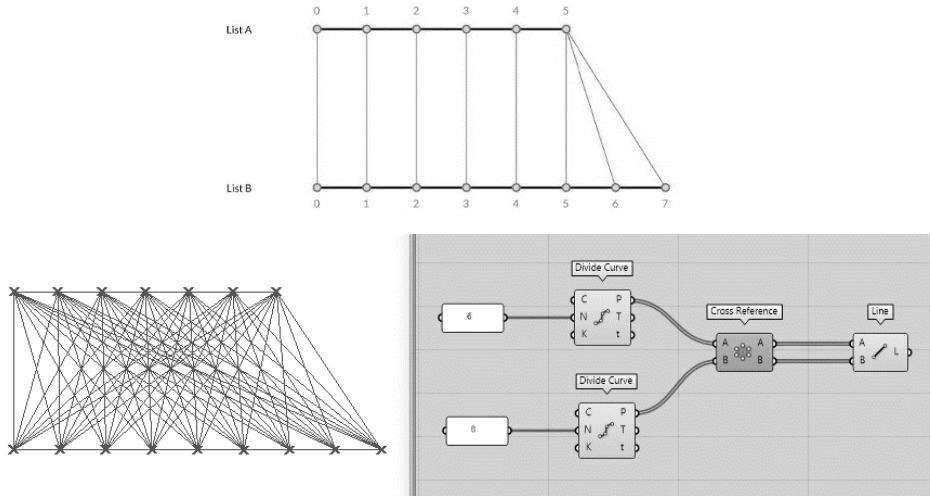
Jika kita menggunakan perintah Line untuk menghubungkan list A dan List B, maka ada tiga kemungkinan yang terjadi.



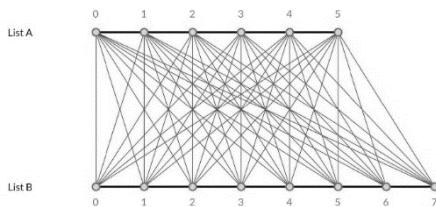
Shortest List akan menghubungkan list dengan jumlah data terkecil secara satu-satu dengan list kedua dengan tiga opsi: **Trim Start**, **Trim End** dan **Interpolate**, hingga data terakhir pada list pertama tercapai.



Longest List akan menghubungkan list dengan jumlah data terbesar secara satu-satu dengan list dengan data lebih pendek dengan lima opsi: **Repeat First, Repeat Last, Interpolate, Wrap, Flip**, hingga data terakhir pada list dengan data terbesar tercapai.

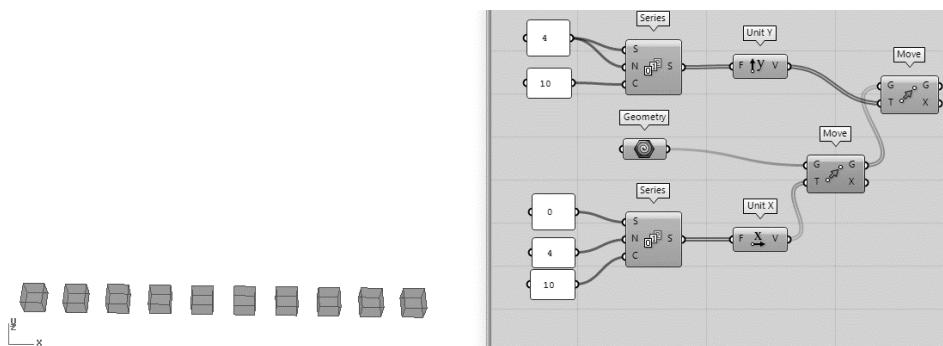


Cross Reference akan menghubungkan list pertama dan list kedua secara satu-satu hingga setiap titik pada setiap list dihubungkan satu dengan yang lain. Ada beberapa opsi dalam Cross Reference: **Holistic, Diagonal, LowerStrict, Lower Triangle, Upper Triangle, UpperStrict, Coincidence**

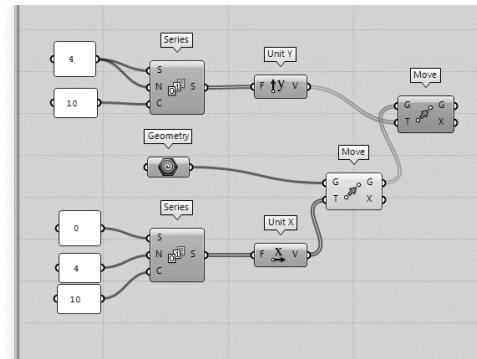
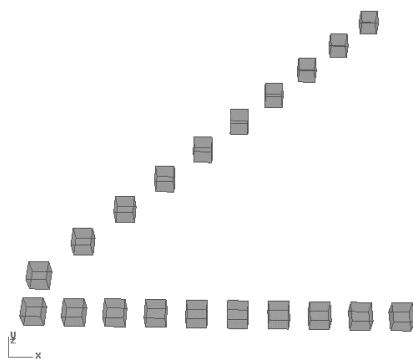


5.2.1. Latihan Data Matching

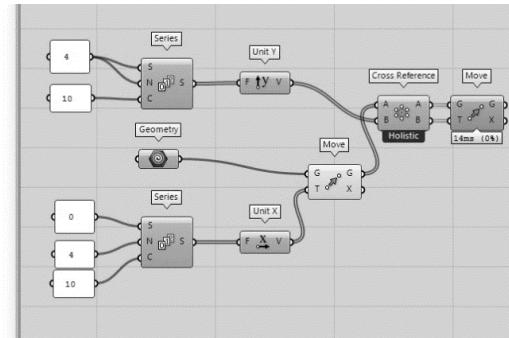
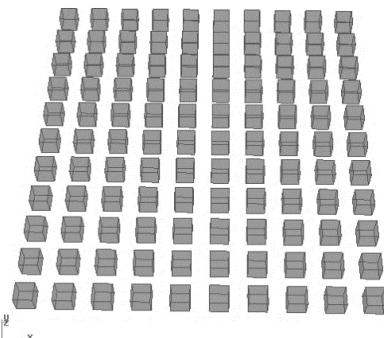
Membuat Grid dari sutau obyek



- Mulai dengan menentukan obyek, misalnya Box
- Buat array satu aksis (sumbu X) menggunakan Move dan Series.



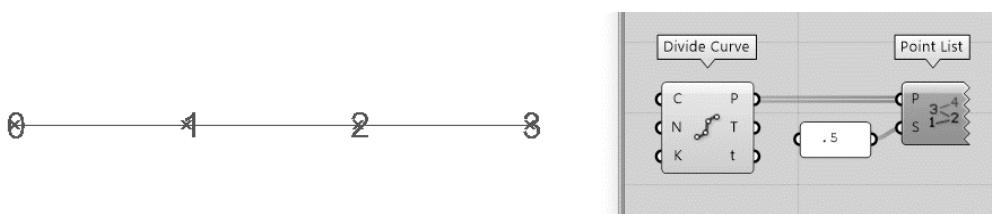
- Bagaimana array pada sumbu Y? Gunakan Series dan Unit Y. Perhatikan, hasil yang kita harapkan tidak sesuai karena, pada setiap obyek pada sumbu X, algoritma akan membuat duplikasi satu-satu dengan jarak sesuai dengan input pada Series untuk unit Y.
- Di sini kita akan gunakan Cross Reference yang menghubungkan setiap obyek pada sumbu X dan setiap obyek pada Sumbu Y.



5.3. Visualisasi List

Memahami dan membayangkan data yang disimpan di GH dalam bentuk **List** dan **Data Tree** cukup sulit jika kita tidak dapat memvisualisasikan bagaimana data tersebut bergerak dari satu komponen ke komponen lain. Beberapa komponen di GH membantu kita untuk memvisualisasikan List maupun Data Tree.

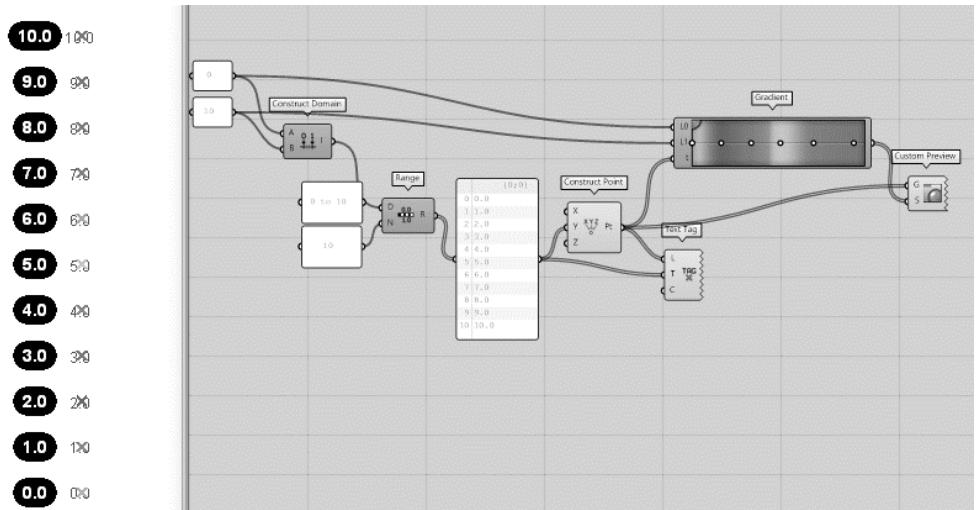
Point List: Komponen yang sangat penting dalam memberikan visualisasi berupa nomor index pada suatu obyek geometri.



Gambar 117. Point List

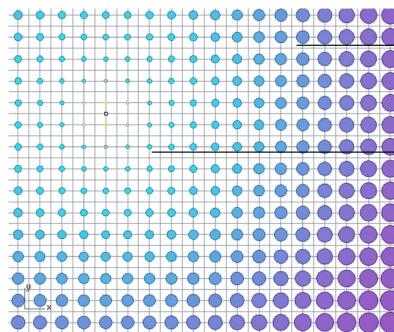
Text Tag dan Text Tag 3D: komponen yang menampilkan data pada suatu komponen berupa teks/ string. Text ini viewport independent, sehingga akan selalu menghadap kamera dan tidak terpengaruh dengan perintah zoom. Perbedaan engan Text Tag3D adalah, pada Text Tag 3D, jika di-bake, maka akan menjadi obyek Text di Rhino. Text Tag 3D juga memiliki opsi Scale.

Gradient dan Custom Preview: menampilkan warna pada data atau obyek (tidak untuk di-render).



Gambar 118. Gradient & Custom Preview

Latihan 8: Data Matching



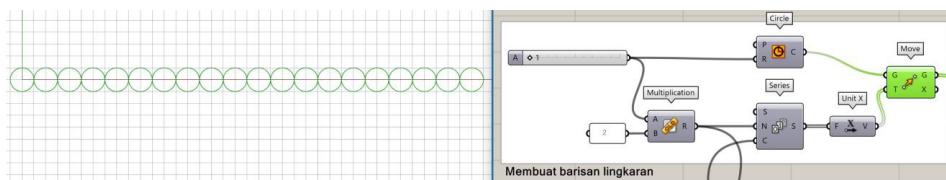
Grid yang terdiri dari obyek-obyek lingkaran (**Circle**).

Radius masing-masing Circle, responsif terhadap jaraknya terhadap suatu titik (**Point**)-Attractor.

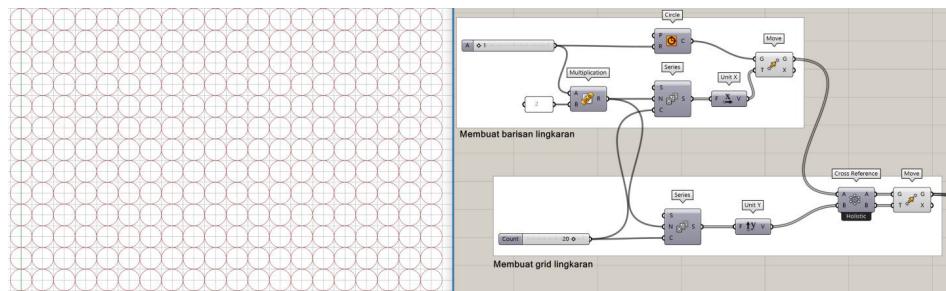
Semakin jauh jaraknya dari titik tersebut, semakin besar radiusnya.

Buat grid dari lingkaran (**Circle**), tentukan titik pusat (**Center Point**) dari masing-masing laingkaran ini.

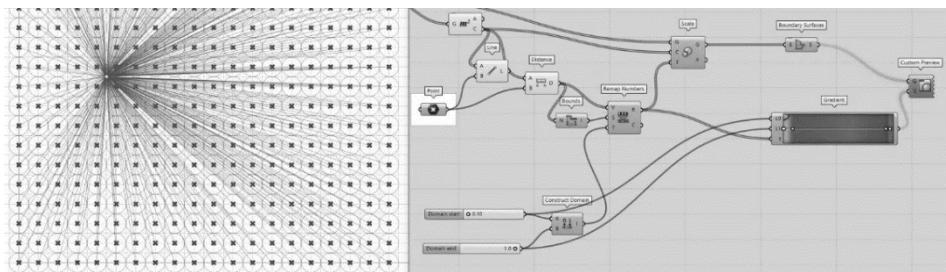
- Buat satu titik (**Point**).
- Tentukan jarak dari setiap **Center Point** lingkaran ke titik ini.
- Jarak ini akan dijadikan variabel Radius Lingkaran.



- Buat **Circle**, duplikasi dengan arah sumbu X dimana jumlah, jarak antar **Circle** ditentukan oleh komponen **Series**.

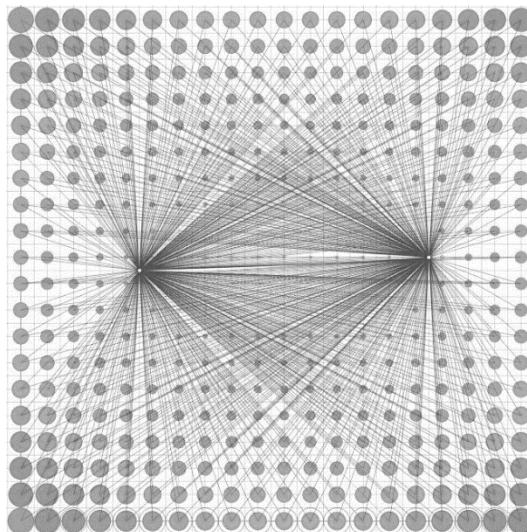


- Buat **Series** untuk menentukan grid. Step atau jarak antar lingkaran sama dengan komponen **Series** sebelumnya, Count atau jumlah adalah total baris pada grid (ke arah sumbu Y).
- Komponen data mapping **Cross Reference- Holistic** digunakan untuk memetakan obyek **Circle** yang diduplikasi pada sumbu Y (lihat bab Data Matching).



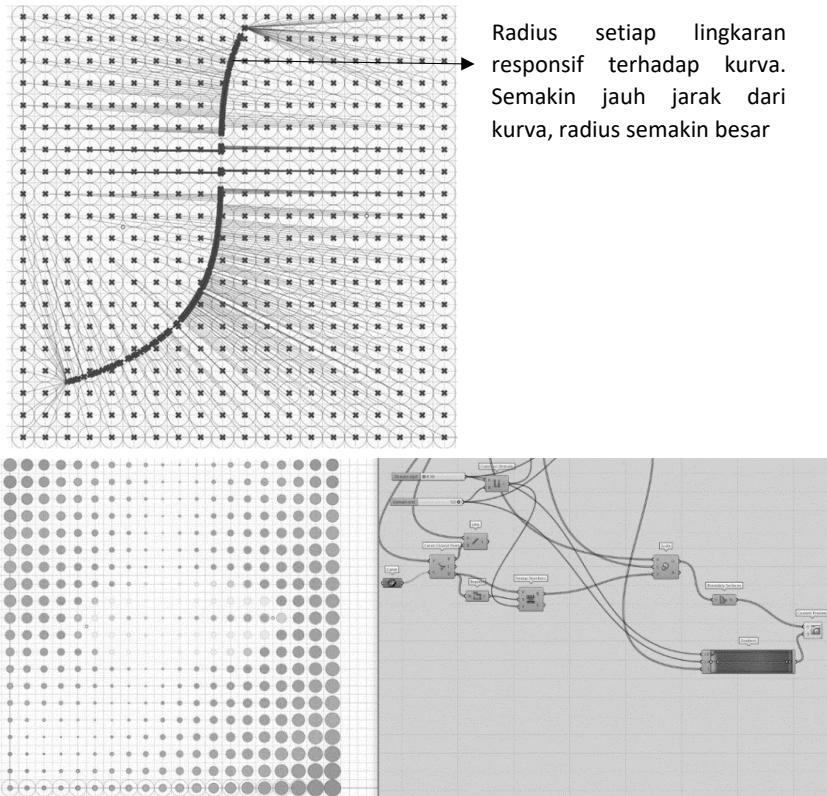
- Masukkan komponen **Area** dan hubungkan dengan komponen **Move** dari definisi sebelumnya. Ini akan menentukan semua titik pusat dari setiap lingkaran.
- Buat titik pada Rhinoceros dan tentukan melalui komponen **Point**.
- Tentukan jarak dari Point ke setiap Center Point lingkaran dengan komponen **Distance**. Di sini anda sudah dapat menentukan setiap jarak dari Point ke setiap Center Point lingkaran.
- Bagaimana caranya menentukan agar radius maksimal setiap lingkaran tetap 1 (artinya tidak lebih besar dari radius awal lingkaran)?
- Gunakan komponen **Remap** untuk melakukan konversi jarak-jarak dari titik atraktor ke titik tengah lingkaran. Gunakan komponen **Bound** untuk menentukan domain (min dan max) jarak-jarak ini. Gunakan komponen **Construct Domain** untuk membuat domain baru yang akan menentukan radius maksimal (1) dan radius minimal (0.1).

Bagaimana jika titik atraktornya lebih dari satu?



- Hitung rata-rata jarak yang sudah dihasilkan antara titik atraktor 1 dan titik atraktir 2 ke setiap titik pusat lingkaran.
- Tentukan nilai minimal dan maksimal jarak rata-rata ini dengan komponen **Bound**.
- Masukkan komponen **Remap** dan tentukan nilai **value** (v) berasal dari nilai rata-rata jarak, source dari nilai min-max yang dihasilkan oleh komponen **Bound** sebelumnya dan nilai target (T) adalah radius yang diinginkan (0.1 ke 1).

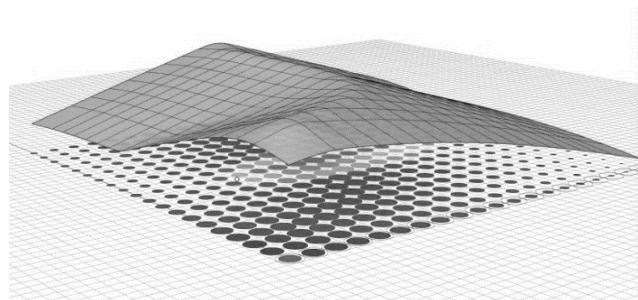
Bagaimana jika atraktornya berupa kurva?



UKM AR-4214 TAHUN 2021

- Komponen **Curve Closest Point** adalah komponen yang akan men-generasi proyeksi titik-titik (Point) pada suatu kurva. Jika kita memiliki sekumpulan titik (dalam hal ini adalah Center Point dari setiap lingkaran), maka sejumlah titik ini akan diproyeksikan menjadi sekumpulan titik pada kurva. Komponen ini menghasilkan titik-titik pada kurva dan jarak-jarak dari setiap titik proyeksi ke titik asal.

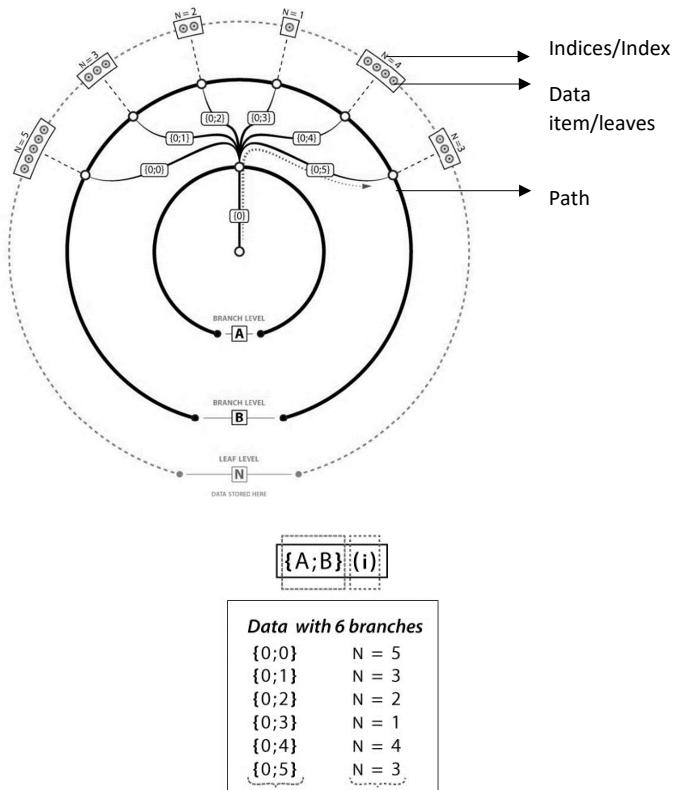
Titik-titik yang dihasilkan dari proses perhitungan jarak terhadap atraktor dapat dikembangkan misalnya dengan memindahkannya ke sumbu Z kemudian menggunakan komponen **Surface from Point**, kita dapat menghasilkan obyek Surface yang merupakan representasi 3D dari pola atraktor 2D.



5.4. Data Tree

Seperti telah dijelaskan di bab sebelumnya, data di GH disimpan dalam tiga jenis: *single data*, *list* dan *data tree*. Single data hanya berisi nomor indeks (index number) dan data, list berisi beberapa data beserta nomor indeksnya namun pada satu path/ cabang.

Apa itu pohon data/ *Data Tree*? *Data Tree* adalah struktur hirarkis untuk menyimpan data pada list bertingkat/ *nested list*, atau list dalam list. Data Tree digunakan apabila GH akan memproses sebuah dataset dan menghasilkan multiple dataset. GH akan memproses data dan memperlakukan data dalam bentuk sub list-sub list. Anda dapat membayangkan ini seperti ketika akan mengakses suatu file dalam suatu folder. Anda akan masuk ke alamat (path) folder dan masuk ke dalam folder untuk dapat mengakses file bersangkutan.



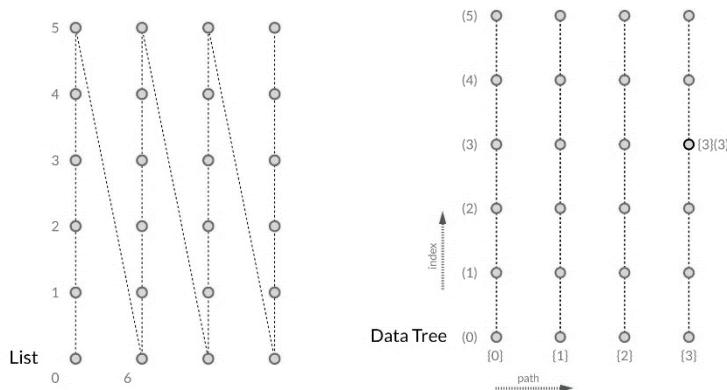
Gambar 119. Pohon Data (Data Tree) Dimana Setiap Path Merupakan Cabang
(Sumber: Gil Akos & Ronie Parsons, 2017)

- Pada diagram di atas, terdapat **satu cabang utama** yang selalu dimulai dari angka **nol**. Cabang utama ini memiliki path $\{0\}$. Path ini tidak memiliki data, tetapi memiliki enam cabang lain (sub-branches). Pada setiap sub cabang, alamat masing-masing mewarisi **alamat/path induk** cabangnya: $\{0;0\}$, $\{0;1\}$, $\{0;2\}$, $\{0;3\}$, $\{0;4\}$, $\{0;5\}$. Ingat path selalu dipisahkan oleh titik koma (;) dan dibungkus oleh kurung kurawal {}.
- Jika pada setiap path yang paling dalam, tidak ada lagi cabang lain, maka isi dari masing-masing cabang terakhir dari path adalah data (sering disebut dengan *leaves*/

daun). Sehingga, setiap data (berapapun jumlah atau banyaknya) akan selalu memiliki satu alamat path atau cabang terakhirnya.

5.4.1. Perbedaan List dan Data Tree

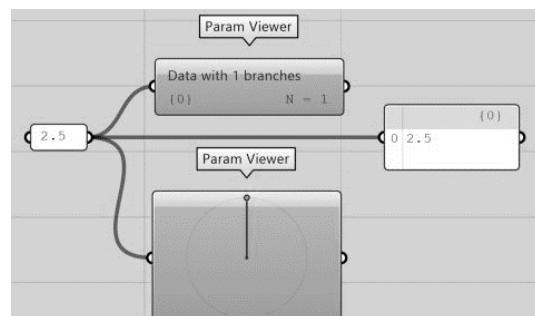
Perbedaan utama antara list dan data tree terlihat dari bagaimana mengelola data dalam jumlah besar. Contoh paling mendasar adalah antara kurva (*curve*) dan permukaan (*surface*). Data pada kurva umumnya berupa list, tetapi data pada surface **sudah pasti berupa data tree**. List tidak lebih dari sebuah himpunan yang berisi semua data dimana setiap data diidentifikasi oleh sebuah nomor indeks. Sedangkan pada Data Tree, setiap data memiliki alamat yakni path. Lebih jelasnya pada gambar berikut.



Gambar 120. List dan Path. (Sumber: Gil Akos & Ronie Parsons, 2017)

Pada list, data diakses menggunakan nomor indeks. Kolom pertama berisi nomor indeks 0-5, kolom kedua berisi nomor indeks 6-11 dan seterusnya

Pada Data Tree, ada empat kolom dan masing-masing kolom ada enam baris dimana setiap baris merupakan nomor indeks dari masing-masing kolom. Kolom (path) dan baris (index) adalah alamat data bersangkutan.

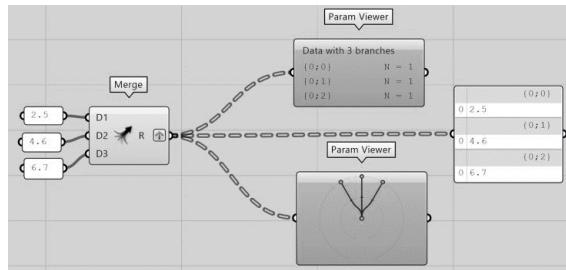


Gambar 121. Path Pada Param Viewer

Jika hanya ada satu data, maka otomatis data itu disimpan dalam path 0 {0}, sehingga bisa dilihat, path {0} memiliki satu buah data, yakni angka 2.5.

Sebuah list dengan n data umumnya disimpan dalam path {0}

Perhatikan bahwa jika datanya tidak bercabang, artinya path {0} maka kabel/connector merupakan garis continuous

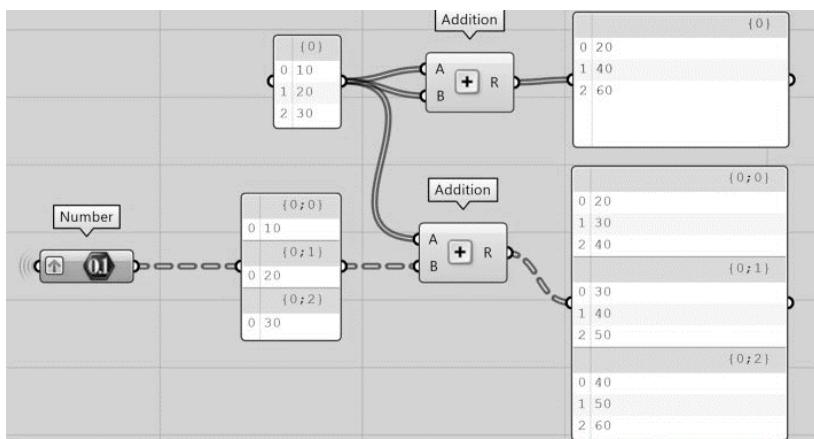


Gambar 122. Path dan Graft

Jika setiap data memiliki cabang/pathnya sendiri, maka data tersebut akan memiliki alamat/pathnya sendiri {A;B} dan nomor indeks dari data tersebut dalam setiap alamatnya masing-masing.

Setiap jumlah path >0 memiliki indikasi kabel yang berbeda(putus-putus) dan dikatakan data bersifat **Graft**.

Apakah signifikansinya, menyimpan data dalam bentuk list (*flat*) atau dalam bentuk Data Tree (*graft*)? Karena komponen akan memproses data berdasarkan struktur datanya. Data dalam bentuk list akan diproses berbeda dengan data dalam bentuk Tree, dan ini akan mempengaruhi outputnya.



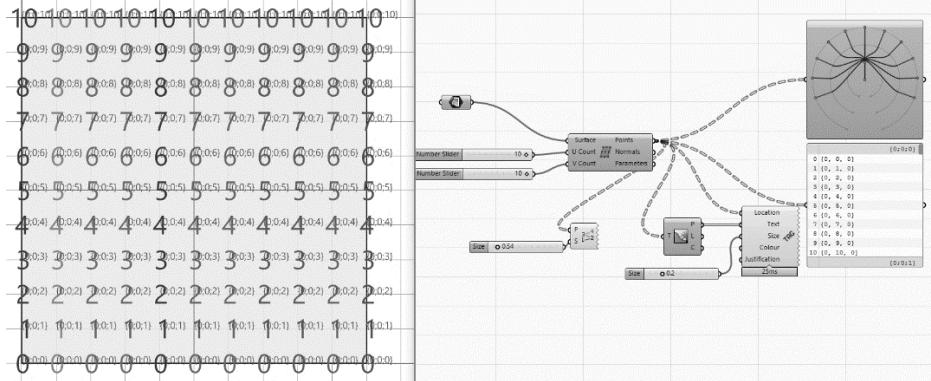
Gambar 123. Proses Pada Data Tree dan List

- Masing-masing data pada list diproses pada setiap data pada setiap path.

5.4.2. Konfigurasi Path

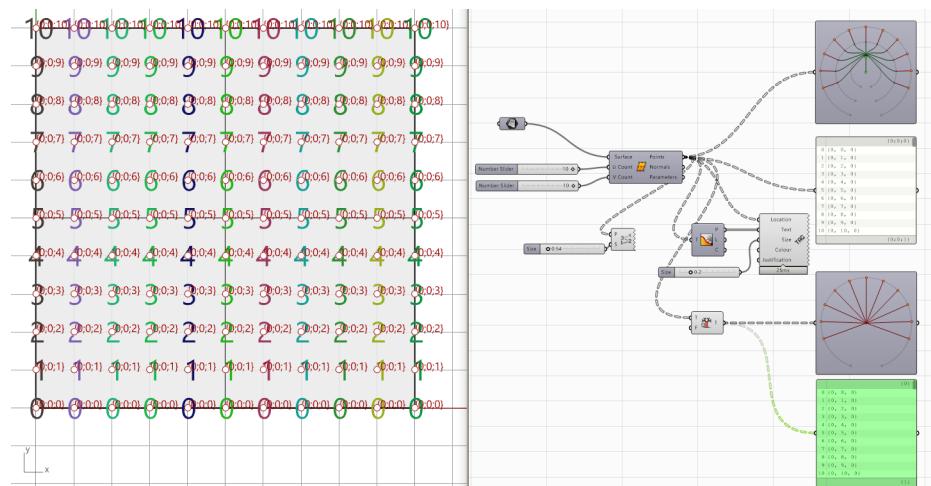
Gambar di atas memperlihatkan sebuah obyek **Surface** yang dibagi menjadi beberapa titik-titik grid ($u=10, v=10$) dengan **Divide Surface**. Pemetaan konfigurasi titik-titik pada

Surface ini sebagaimana terlihat. Setiap kolom merepresentasikan cabang, dimana setiap cabang ada 11 titik dengan indeks mulai 0 hingga 10. Perhatikan bahwa secara default, konfigurasi path ini memiliki 3 turunan {0;0;0} hingga {0;0;10}.



Gambar 124. Konfigurasi Path

A. Simplify Tree

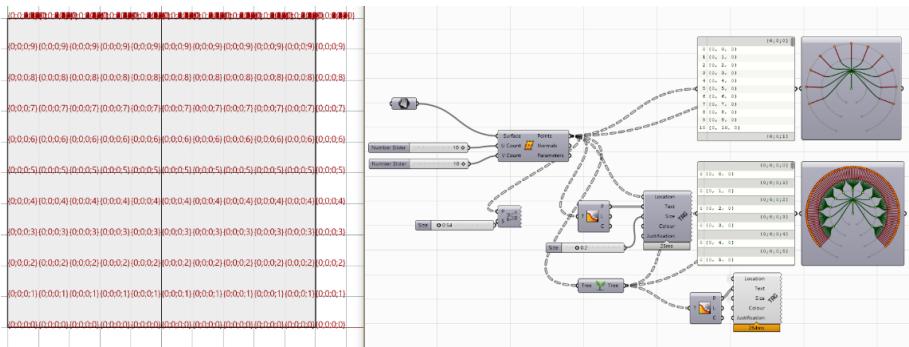


Gambar 125. Simplify Tree

Simplify Tree berfungsi untuk menghilangkan turunan yang tidak memiliki nilai. Jadi kalau sebelumnya Path bernilai {0;0;0} ke {0;0;10}, maka setelah Simplify Tree menjadi {0} ke {10}. Perhatikan bahwa informasi koordinat dan indeks dari tiap titik tidak berubah, yang berubah adalah alamat path-nya.

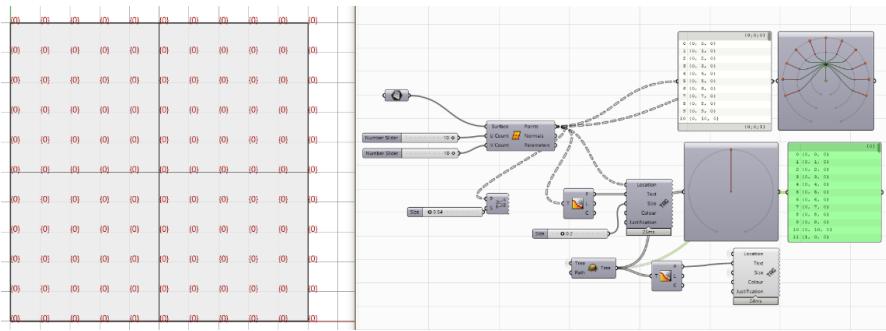
B. Graft Tree

Graft Tree akan membuat turunan dan cabang baru dari setiap data. Pada Contoh di atas, sebelumnya adalah {0;0;0} ke {0;0;10} menjadi {0;0;0;0} ke {0;0;10;10}. Setiap data memiliki alamat path sendiri (punya cabang sendiri).



Gambar 126. Graft Tree

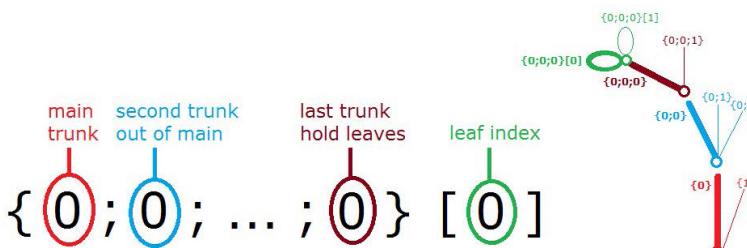
C. Flatten Tree



Gambar 127. Flatten Tree

Flatten Tree akan meniadakan cabang dan turunan, dimana semua data bara pada cabang utama yakni $\{0\}$.

5.4.3. Notasi Data Tree

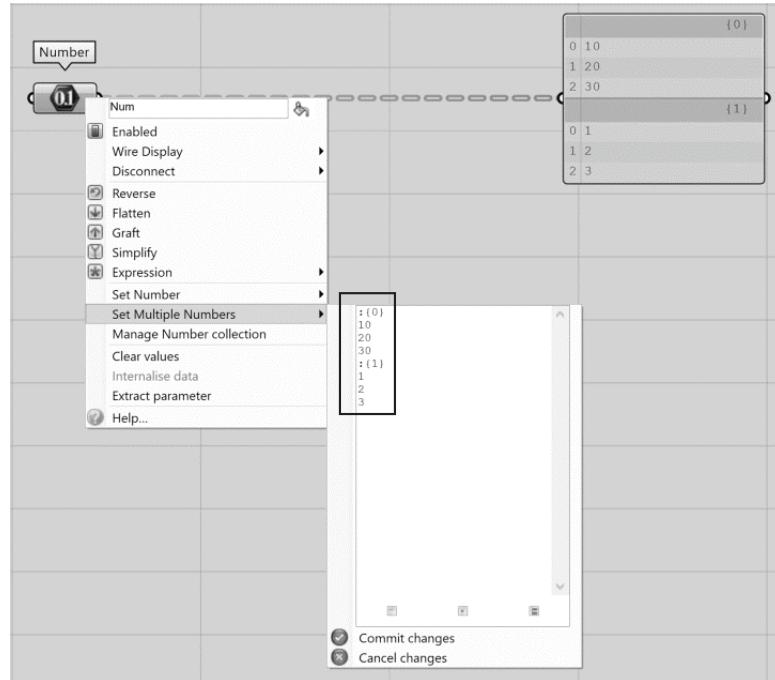


Gambar 128. Notasi Data Tree. (Sumber: Gil Akos & Ronie Parsons, 2017)

5.4.4. Menghasilkan Data Tree

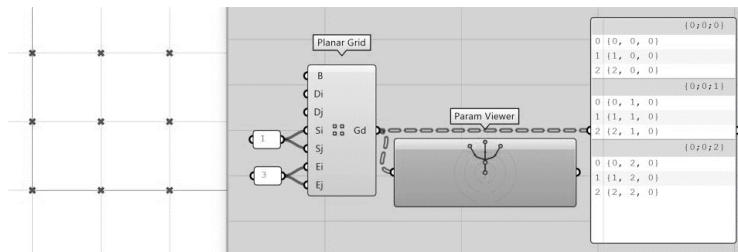
Harap diingat bahwa menghasilkan Data Tree adalah membuat cabang dari setiap data. Ada beberapa cara dalam menghasilkan Data Tree, baik pada kontainer maupun hasil suatu proses tertentu.

Set Multiple Numbers/Input: menggunakan notasi `:{N}` diikuti oleh data pada input suatu kontainer seperti contoh di bawah untuk menghasilkan multiple data dalam setiap cabang/path yang ditentukan. Cara ini juga membuat kita dapat menentukan nomor indeks cabang yang tidak harus berurutan.



Gambar 129. Produksi Path Pada Panel

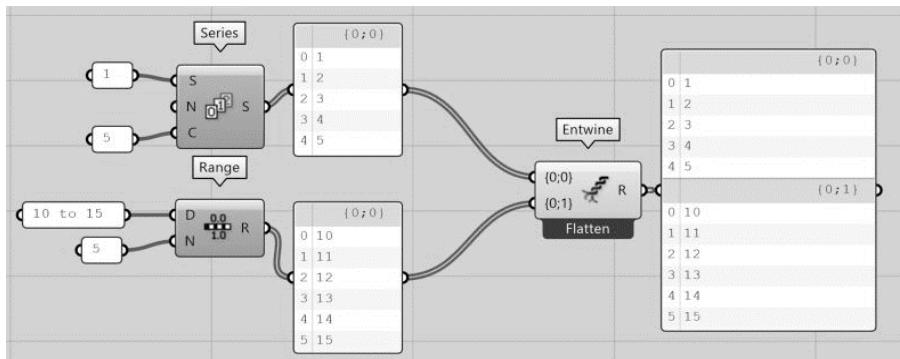
Planar Grid: menghasilkan grid berupa titik-titik yang ditentukan arah vektor pada setiap aksis, jarak-jarak setiap aksis dan jumlah titik di setiap aksis.



Gambar 130. Planar Grid

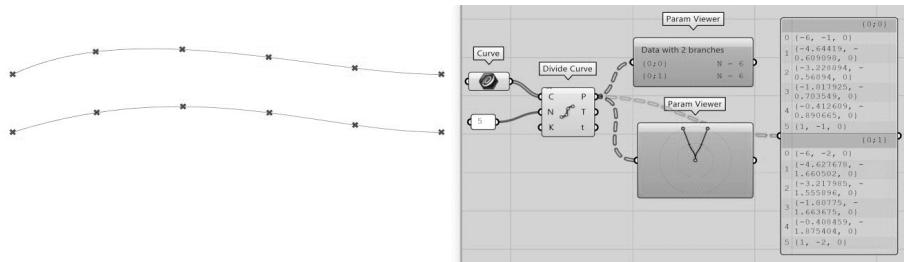
- Perhatikan bahwa secara default, Planar Grid akan menghasilkan path dalam tiga hierarki `{0;0;N}`

Series, Range dan Number Generator lain dapat menghasilkan data dalam bentuk Data Tree jika digabungkan menggunakan komponen **Entwine**.



Gambar 131. Entwine

Divide Curve juga dapat menghasilkan Data Tree jika inputnya berasal dari Multiple Curves

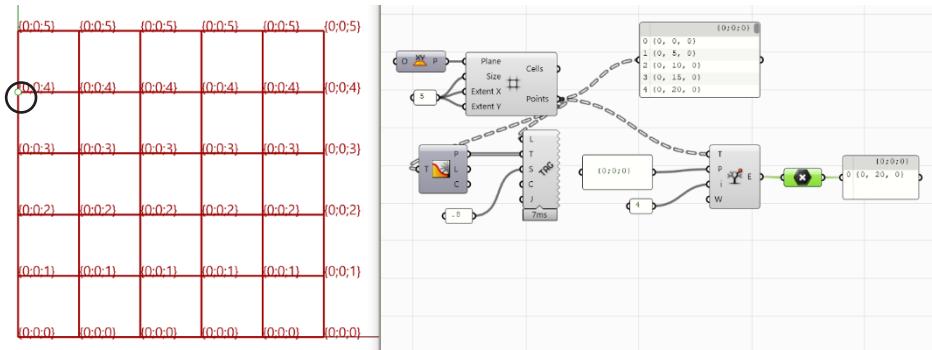


Gambar 132. Multiple Divide Curves

5.4.5. Mengakses dan Memodifikasi Data pada Tree

Seperti halnya pada List, data pada Tree dapat diakses dan dimodifikasi dengan memahami konfigurasi path.

A. Tree Item



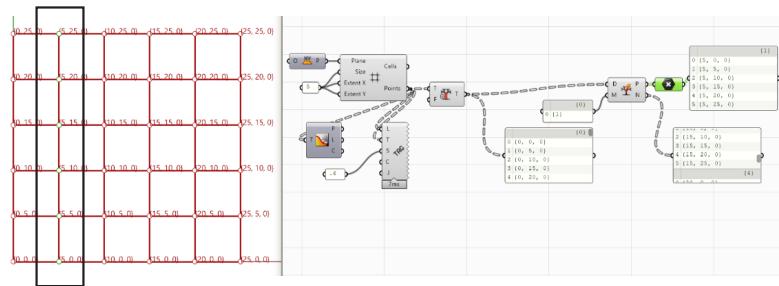
Gambar 133. Tree Item

Tree Item digunakan untuk mengakses data pada Tree. Parameter input dari komponen ini adalah:

- Tree: Tree
- Path: informasi konfigurasi path ($\{x;x;x\}$)
- Index: nomor index pada path untuk mengakses data yang diinginkan.

Pada contoh di atas, input path adalah $\{0,0,0\}$ dan index adalah 4, dan output yang dikeluarkan adalah Point dengan koordinat $(0,20,0)$. Untuk cabang path selanjutnya, input path adalah: $\{0;0;1\}$, $\{0;0;2\}$, dan seterusnya.

B. Split Tree



Gambar 134. Split Tree

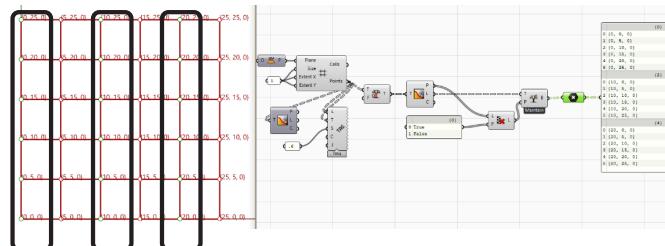
Split Tree digunakan untuk membagi Tree menjadi dua bagian: **P** adalah cabang beserta data dari input Mask yang kita masukkan, artinya cabang yang kita ekstrak, dan **N** adalah cabang sisa. Parameter input adalah:

- Data: berupa tree
- M (Mask): berupa informasi konfigurasi Path yang akan kita ekstrak.

Pada contoh di atas, setelah konfigurasi Tree dari data dikenai **SimplifyTree** (konfigurasi cabang menjadi $\{0\}$, $\{1\}$, dan seterusnya), maka kita bisa gunakan informasi Mask $\{1\}$ untuk mengekstrak semua data pada path $\{1\}$. Jika tanpa **Simplyfy Tree**, maka kita harus cek konfigurasi path dengan **Tree Statistic** sebelum menentukan konfigurasi path yang akan kita ekstrak pada parameter Mask.

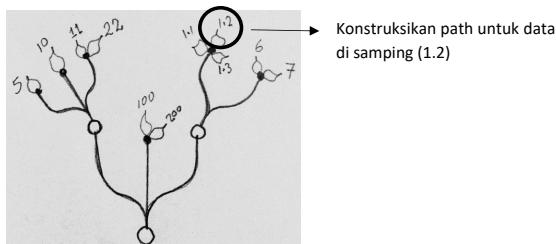
C. Tree Branch

Tree Branch digunakan untuk mengekstrak lebih dari satu cabang dengan fleksibilitas lebih tinggi, misalnya kita bisa gunakan masking berupa **Cull Pattern**. Pada contoh, **Cull Pattern** digunakan dengan parameter *True*, *False* digunakan untuk mengekstrak cabang $\{0\}$, $\{2\}$, $\{4\}$.



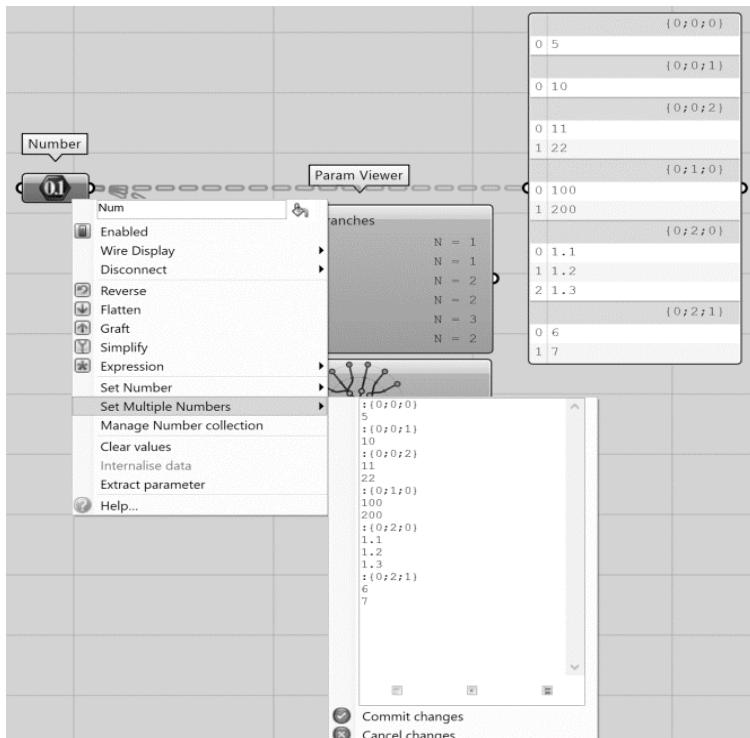
Gambar 135. Tree Branch

Latihan 9: Pengertian Data Tree



Gambar 136. Ilustrasi Data Tree. (Sumber: Rajaa Isaa, 2020)

- Cabang utama=0, cabang selanjutnya adalah 0,1,2. Data berada pada cabang nomor 2 sehingga path= {0;2;X}[N] dimana X adalah cabang ketiga dan N adalah indeks dimana data berada.
 - Data berada pada cabang pertama pada turunan cabang ketiga sehingga path= {0;2;0} dan data berada pada nomor indeks 1 sehingga full path untuk data 1.2 adalah {0;2;0}[1].
 - Cara menuliskan di Grasshopper:



Gambar 137. Membuat Path dan Data Dalam Path Dengan Panel

Notes:

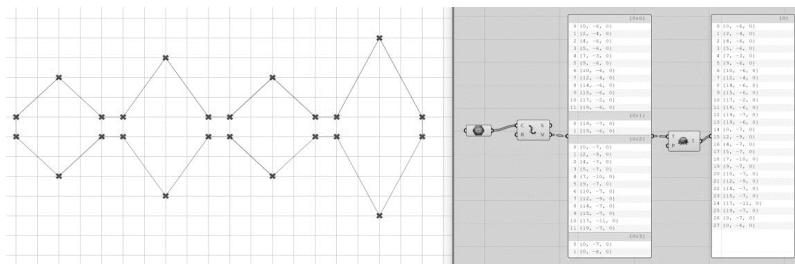
- Data Tree menjadi ‘invisible’ jika komponen tidak melakukan operasi terhadap *multiple data* secara bersamaan. Pada kasus ini, komponen akan memperlakukan data (baik single maupun list) sebagai data yang hanya memiliki satu cabang [0] [0,...n].

5.4.6. Memanipulasi Data Tree

Beberapa komponen digunakan untuk memanipulasi susunan atau struktur data pada sebuah Data Tree untuk tujuan tertentu dan menghasilkan output tertentu. Beberapa komponen tersebut adalah: **Flatten**, **Graft**, dan **Flip Matrix**.

1. Flatten Tree

Dari Namanya, komponen ini berfungsi untuk menghilangkan semua cabang/branch dan menyimpan semua data yang ada pada setiap cabang pada cabang utama {0}. Jika ada kasus data yang didapatkan berupa Data Tree dengan kabel koneksi putus-putus, maka jika dilakukan *flatten*, maka kabel koneksi menjadi garis kontinyu yang artinya, tidak ada cabang pada data tersebut.

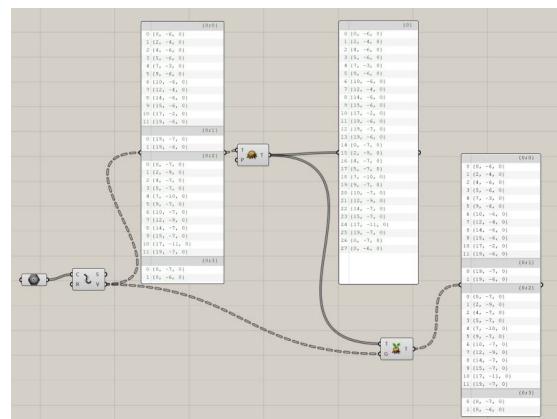


Gambar 138. Flatten

- Secara internal, setiap komponen memiliki menu untuk meng-konversi data input atau data output menjadi flat, atau fitur flatten yakni dengan klik-kanan dan pilih Flatten (ikon tanda panah mengarah ke bawah).

2. Unflatten Tree

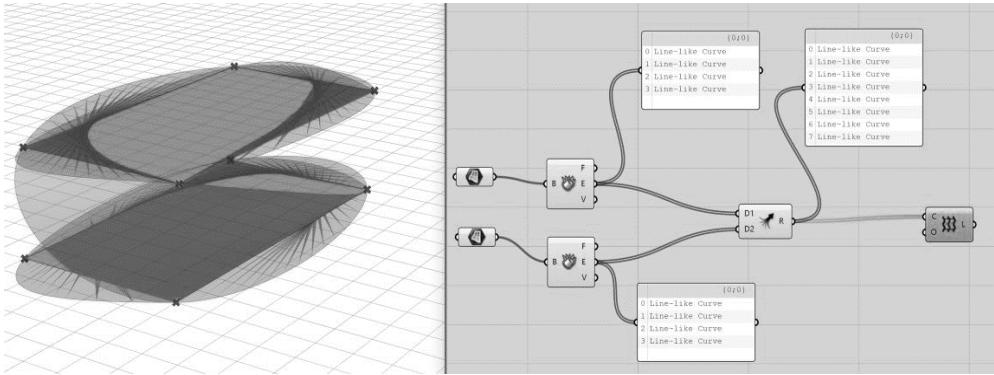
Kebalikan dari Flatten, komponen ini akan memindahkan data pada cabang-cabang/branches. Komponen ini memerlukan input berupa Guide DataTree yang menginformasikan jumlah turunan dan cabang serta data -data yang masuk dalam cabang tersebut.



Gambar 139. Unflatten

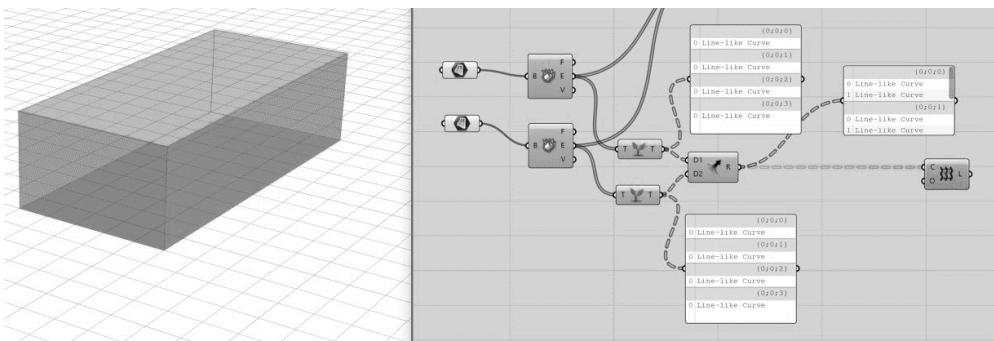
3. Graft Tree

Graft Tree akan membuat cabang baru dari setiap data dalam suatu list. Jika ada n data dalam satu list, maka komponen Graft Tree akan membuat cabang sebanyak n dimana pada masing-masing cabang akan ada satu data pada indeks 0.



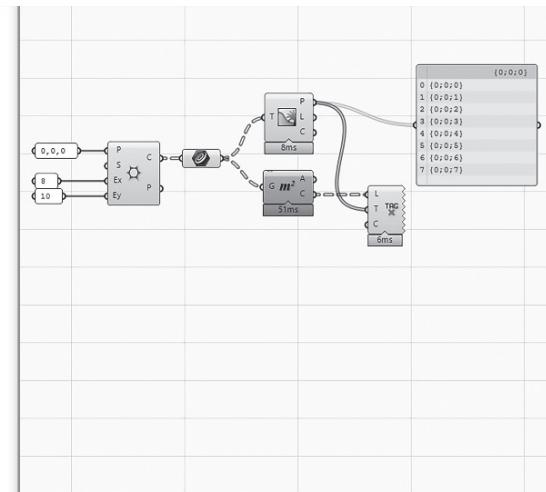
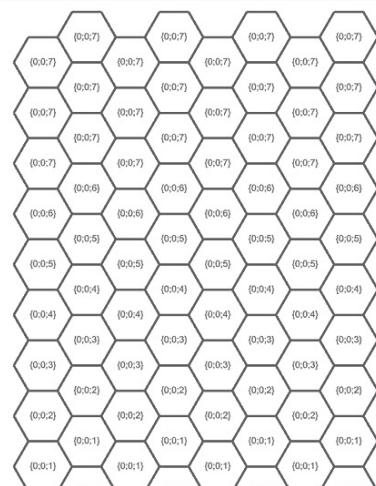
Gambar 140. Graft

- Jika ada dua surface masing-masing sisinya adalah edge: a,b,c,d dan A,B,C,D. Komponen **Deconstruct Brep** akan meng-ekstrak *Face*, *Edge* dan *Vertices* dari masing-masing *Surface*.
- Jika kita ingin melakukan **Loft** untuk menghubungkan antara *edge*: a-A, b-B, c-C dan d-D kita tidak dapat melakukannya tanpa mengkonversi data edge dari masing-masing surface dengan Graft Tree karena hasilnya seperti di atas. Komponen loft akan melakukan loft a-b-c-d-A-B-C-D. Tidak seperti yang kita harapkan.

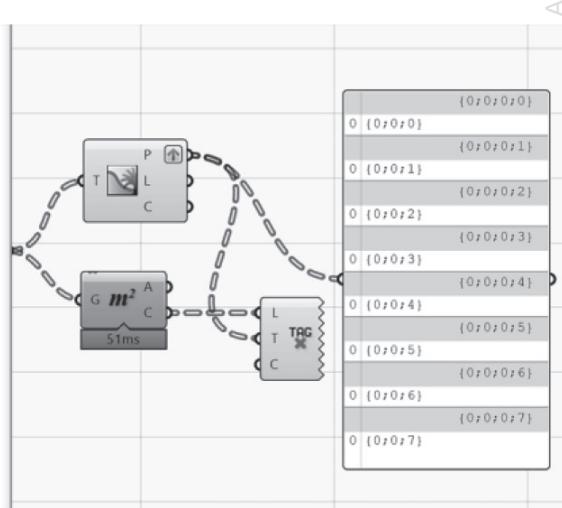
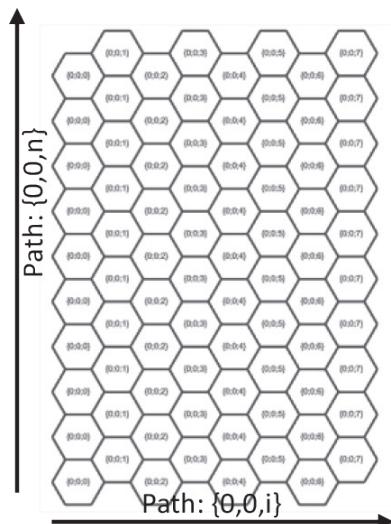


- Jika data dari edge pada masing-masing surface di-graft, maka ketika dilakukan merge, data edge pada masing-masing surface akan otomatis dikelompokkan pada satu cabang/branch sehingga perintah loft akan mengeksekusi obyek yang berada pada masing-masing cabang.
- Komponen Graft Tree dapat diaktifkan dengan klik-kanan pada port input atau output pada komponen dan pilih Graft (ikon tanda panah mengarah ke atas).

Latihan 10: Data Tree

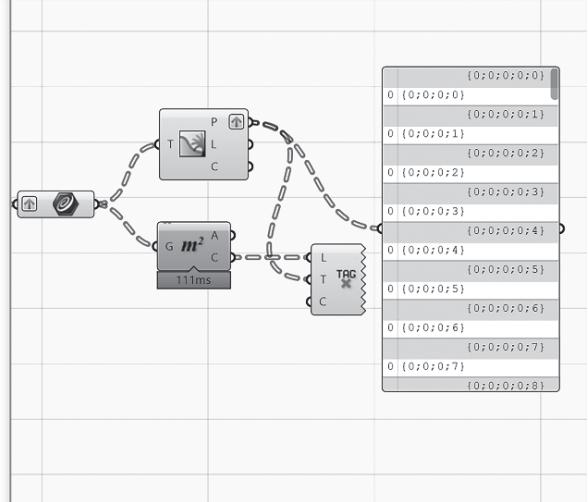
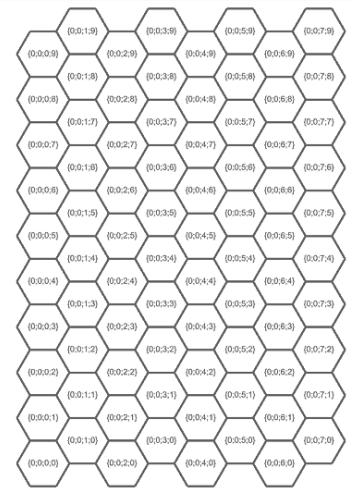


- Komponen **Tree Statistic** menunjukkan jumlah *Path* (cabang), jumlah *list* dari setiap *path* dan jumlah *path*.
- Visualisasi teks dengan **Text Tag** pada setiap sel hexagonal menunjukkan alamat path pada setiap sel.
- Pertanyaan:
 - Apa yang salah dari definisi di atas?*
 - Mengapa setiap sel memiliki 3 kedalaman path?*

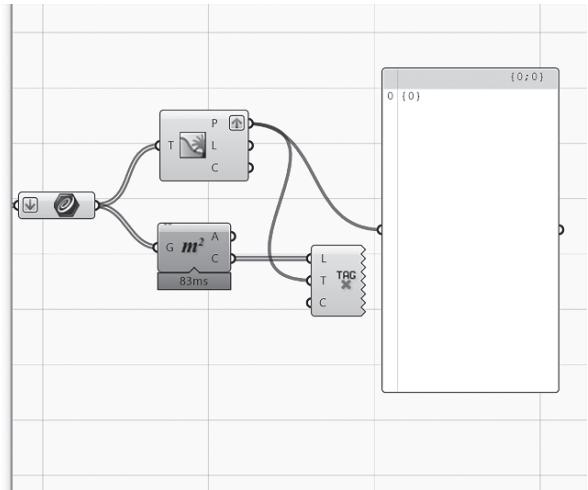
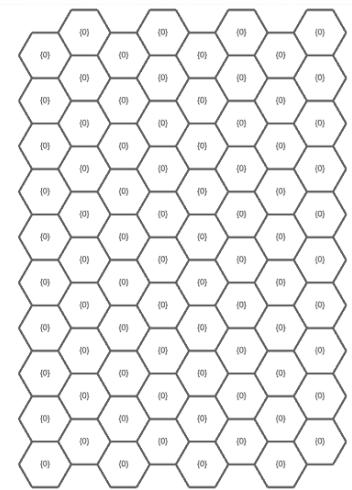


- Dengan melakukan **Graft** pada keluaran *P* (*path*) dari **Tree Statistic**, maka struktur penomoran alamat setiap sel berubah.
- Satu *path* (satu kolom) terdiri dari 10 hexagon. Setiap baris terdiri dari 8 hexagon (indeks 0-7).

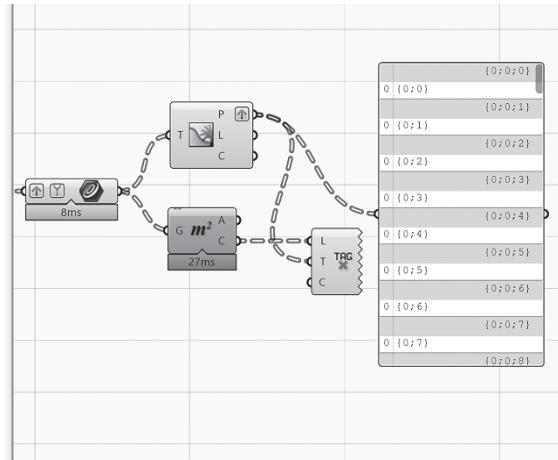
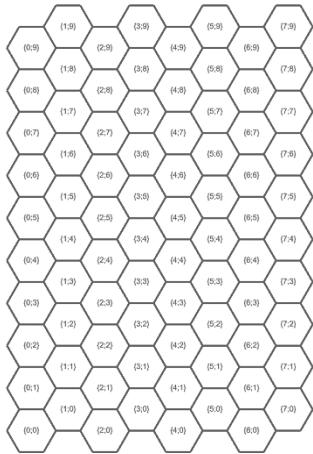
- Pada tahap ini pada setiap kolom, setiap hexagon masih memiliki alamat *path* yang sama. Bagaimana caranya untuk menunjukkan agar setiap hexagon memiliki *pathnya sendiri*?



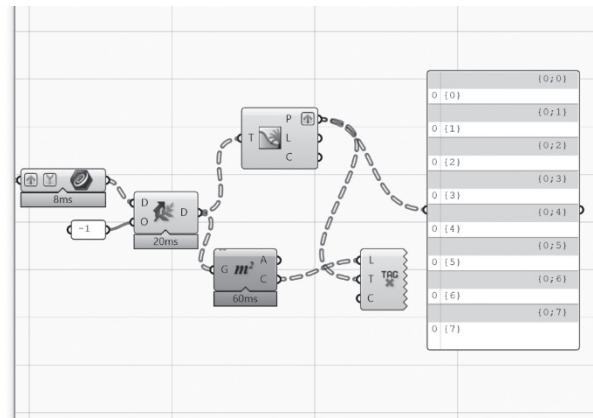
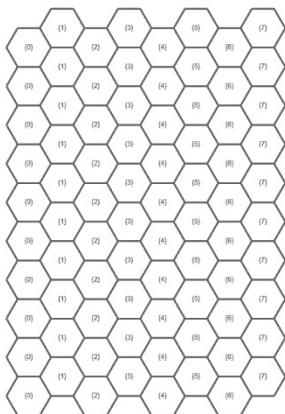
- Dengan melakukan **Graft** pada keluaran *Cell* dari komponen Hexagon maka struktur data berubah.
- Setiap kolom memiliki urutan sel hexagon dan setiap baris memiliki urutan sel hexagon. Perhatikan bahwa dengan melakukan **Graft** pada Cell, kita menambah cabang di kedalaman keempat.
- Jadi cabang ketiga adalah urutan baris dari sel, cabang keempat adalah urutan kolom dari setiap baris.



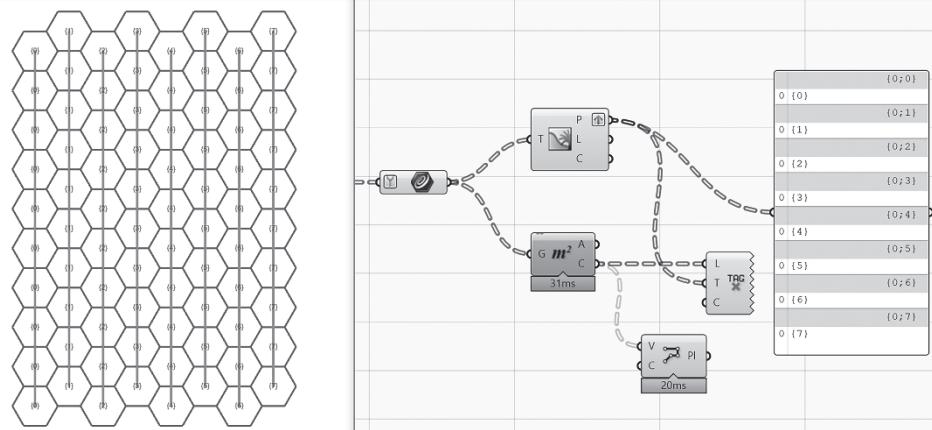
- **Flatten** pada Cell menghilangkan semua cabang (*path*), sehingga semua sel hanya berada pada cabang utama {0}, namun Flatten tetap mempertahankan urutan list dari setiap kolom.



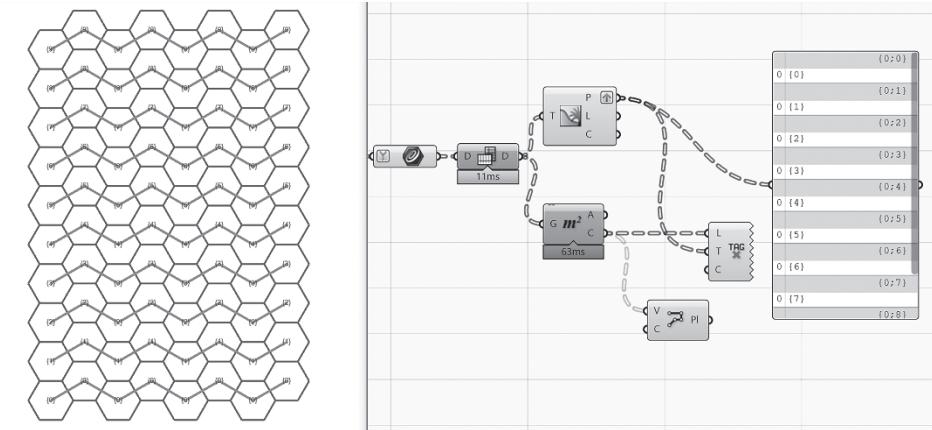
- **Simplify** pada *Cell* akan menghilangkan semua cabang yang tidak berubah. Jika diperhatikan, digit pertama dan kedua (cabang utama dan cabang kedua) memiliki nilai nol sehingga ini dihilangkan.
- Dengan **Simplify** lebih jelas terlihat struktur data dari obyek tersebut.



- **Shift Path** menghilangkan cabang sesuai dengan parameter Offset (O). Pada contoh di atas, jika kita ingin menghilangkan semua cabang kedua (dari data yang sudah di-Simplify sebelumnya), maka nilai O adalah -1 (bergeser satu ke kiri). Jika nilai O adalah 1 maka yang dihilangkan adalah cabang pertama. Jika nilai O adalah 0 (nol), maka tidak ada perubahan.



- **Polyline** digunakan untuk membuat garis dengan input titik-titik pusat setiap sel. Ingat bahwa proses secara default menghubungkan data dengan indeks yang sama. Terlihat di contoh bahwa garis-garis Polyline menghubungkan setiap titik pusat sel dengan indeks yang sama.



- **Flipmatrix** mengubah membalik susunan struktur data (kolom → baris, baris → kolom).

