

GRASSHOPPER

Interface
Alur Kerja
Pengantar Algoritma
Anatomi Program
Jenis Obyek
Bekerja dengan Data

3. DASAR-DASAR DESAIN PARAMETRIK DENGAN GRASSHOPPER

3.1. Bagaimana Desainer (dan Arsitek) Menggunakan Parameter

Judul di atas diambil dari Bab 3 buku seminal karya Robert Woodbury¹⁰ yang secara spesifik mengetengahkan adanya perubahan bagaimana desainer (arsitek) mendekati suatu persoalan desain, perubahan pada proses mendesain dan perubahan cara berpikir ketika berproses demikian. Jika bab I buku ini menjelaskan tentang sejarah dan konteks serta latar belakang berkembangnya konsep parametrik kemudian dilanjutkan dengan pemahaman atas model dan pemodelan 3D dengan Rhinoceros 3D, maka sebelum masuk ke hal-hal teknis tentang algoritma, data dan pemrograman visual dengan Grasshopper, ada beberapa rangkuman prinsip yang perlu diketahui dan dipahami.

3.1.1. Ketrampilan (*Skill*) Baru

A. Merancang Aliran Data

Desain parametrik adalah tentang mengenali relasi antar obyek. Karena obyek-obyek dapat saling bergantung (*dependent*, misalnya sebagai input dari output obyek lain, dan seterusnya), maka sangat penting untuk dapat mengamati bagaimana **pola aliran data dan aturan** (*rules*) serta **batasan** (*constraints*). Desainer (arsitek) yang merancang, menyusun, mengedit, menentukan aliran-aliran data antar obyek sehingga hasil final, dalam konteks bentuk (*form*), lebih merupakan agregat final dari kumpulan obyek penyusun dengan karakteristik (*perilaku-behavior*) tertentu. Karena hasil final ini adalah representasi dari hubungan relasional antar obyek maka sifatnya **dinamik, interaktif, dan parametrik**.

B. Membagi dan Menyelesaikan (*Divide to Conquer*)

Salah satu strategi pemrograman dasar yang selalu dikenalkan adalah, **membagi persoalan menjadi persoalan yang lebih kecil**, lebih operasional dan kemudian pemecahan persoalan dimulai dari situ. Desainer (arsitek) selalu belajar mengelola (baca: mengorganisasikan) obyek-obyek rancangan (baca: kebutuhan fungsi, kebutuhan ruang dan fasilitas) sebagai obyek-obyek yang memiliki hirarki, dimana semakin sederhana interaksi antar obyek, semakin mudah menyelesaikan persoalan desain. Desain parametrik akan selalu memerlukan strategi ini, dimana lebih mudah merancang, merakit dan menyusun obyek-obyek dengan relasi dan interaksi sederhana, lalu menambahkan *node* atau obyek baru dengan relasi baru, dibandingkan dengan merancang relasi semua obyek sekaligus.

¹⁰ Robert Woodbury (2010). *Elements of Parametric Design*. Routledge. Sangat dianjurkan untuk dibaca bagi siapapun yang akan mendalami desain parametrik dan desain komputasional.

C. Menamai

Nama atau label memfasilitasi komunikasi. Sekali lagi, dalam bidang kerja arsitek, nama atau label sangat lumrah untuk menjelaskan posisi, karakter, spesifikasi dan penjelasan atas suatu obyek. Dalam desain parametrik, penamaan atau pelabelan sangat penting untuk menjelaskan apa-apa yang sudah kita sebagai desainer atau arsitek kenal. Karena yang kita lakukan adalah merancang proses yang di dalamnya terdapat obyek yang saling berkaitan, maka sangat penting untuk memikirkan dan merancang sistem penamaan untuk obyek dan proses.

D. Berpikir Abstraktif

Istilah abstraksi merupakan tingkatan kognisi yang tinggi dimana proses melakukan abstraksi berarti **memproduksi sebuah konsep** alih-alih sebuah contoh atau suatu yang kongkrit. Abstraksi adalah **mengkonstruksi suatu struktur atau suatu sistem**. Istilah abstraksi yang dikenal dalam proses desain arsitektur adalah mengkonstruksi gagasan atau ide yang masih kasar (baca: abstrak) dan sangat mungkin terjadi interpretasi yang berbeda dan umumnya masih memungkinkan terjadinya variasi atas penyelesaian atau solusi.

Di dalam sains komputer, abstraksi memiliki makna yang menggambarkan sebuah sistem umum dan tidak detil atau sebuah hal. Abstraksi berkonotasi dengan pendekatan formal dalam bentuk bahasa formal (*code*) untuk menerjemahkan abstraksi tadi menjadi sebuah sistem yang bekerja (*program*).

Berpikir abstraksi adalah melatih untuk melihat sesuatu dalam kerangka sistem yang bekerja dalam sesuatu itu.

E. Berpikir Matematis

Kita sadari atau tidak, model CAD yang kita kenal adalah representasi dari model-model matematis. Dalam sejarahnya, profesi arsitek selalu dekat dengan matematika. Dalam istilah Robert Woodbury, ada dua jenis penggunaan matematika: *use mathematics* dan *do mathematics*. Kebanyakan para arsitek secara *inherent* menggunakan (*use*) matematika dalam keseharian berpraktek: dimensi dan ukuran, posisi dan lokasi, proporsi dan sebagainya. Beberapa arsitek bekerja dan berproses mendesain (*do*) dengan matematika untuk melakukan eksplorasi geometri berdasarkan fungsi dan rumus matematika. Beberapa contoh adalah: Leonardo Da Vinci, Antonio Gaudi, Le Corbusier dan lainnya.

Sistem parametrik membuat pemanfaatan matematika menjadi lebih dalam dan menjadi pondasi relasi antar obyek dalam program. Semua hubungan relasional, pendefinsian atas sebuah variabel, konstanta, pendefinisan atas kondisi, dan lainnya berdasarkan bahasa-bahasa matematis

F. Berpikir Algoritmik

Desain parametrik pada dasarnya adalah diagram (*graph*) yang terdiri dari obyek (*node*) yang memiliki atribut dan hubungan relasi antar obyek. Karena merupakan program, maka ada langkah atau *sequence* yang memiliki batas: ada awal dan ada akhir. Jalinan langkah-

langkah berupa aturan (rules) ini disebut sebagai algoritma. Jika kembali ke nomor D pada Ketrampilan (*skill*) Baru, maka algoritma adalah **muara dari proses berpikir abstraktif**. Definisi algoritma yang mudah untuk dipahami adalah demikian (Berlinski, 1999):

Sebuah prosedur yang punya batas, yang ditulis menggunakan simbol-simbol tertentu, dan berisi rangkaian instruksi yang akurat, tidak ambigu, berjalan dalam langkah-langkah tertentu, serta memiliki hasil tertentu.

3.2. Strategi Belajar Desain Parametrik

Bab sebelum ini adalah rangkuman dari beberapa ketrampilan baru yang akan dikuasai ketika kita belajar desain parametrik. Berikut adalah beberapa **strategi** yang perlu dilakukan dalam belajar desain parametrik sebagaimana telah distudi oleh Robert Woodbury:

A. Menggunakan Sketsa

Proses membuat sketsa adalah proses **abstraksi** yang paling dikenal oleh desainer (arsitek). Sketsa adalah hubungan interaksi dengan media paling personal yang dilakukan arsitek dalam mengkonstruksi gagasan. Dalam sketsa ada proses **eksplorasi** atas **abstraksi, eksplorasi, dan interaksi yang dinamis**. Pemodelan parametrik mengambil keuntungan dari adanya sketsa-sketsa dengan mengkonversi proses tersebut ke dalam rangkaian prosedur yang lebih formal. Serupa dengan penggunaan media digital (baca: software) dalam merancang obyek-obyek geometri 3D, adanya sketsa-sketsa membantu mengaktualisasi abstraksi dalam bentuk digital.

B. Coding Kemudian (baca: *Throw Code Away*)

Perbedaan mendasar antara desainer (arsitek) dan programmer adalah pada akhirnya keluaran dari program (baca: desain parametrik) yang dirancang oleh desainer adalah **desain atau properti atas suatu desain**, bukan programnya itu sendiri. Tentu prinsip bahwa sebuah program harus efisien dan mudah dipahami (untuk dipelajari dan digunakan kembali-*re-use*), tidak banyak *redundancy* (akan mengurangi kecepatan eksekusi program) harus dipegang oleh desainer (arsitek) yang mempelajari desain parametrik. Namun demikian, obyektif atau tujuan akhir dari merancang program tersebut adalah desain. Prioritas adalah kontrol terhadap keluaran, bukan pada proses atau programnya.

C. Copy lalu Modifikasi

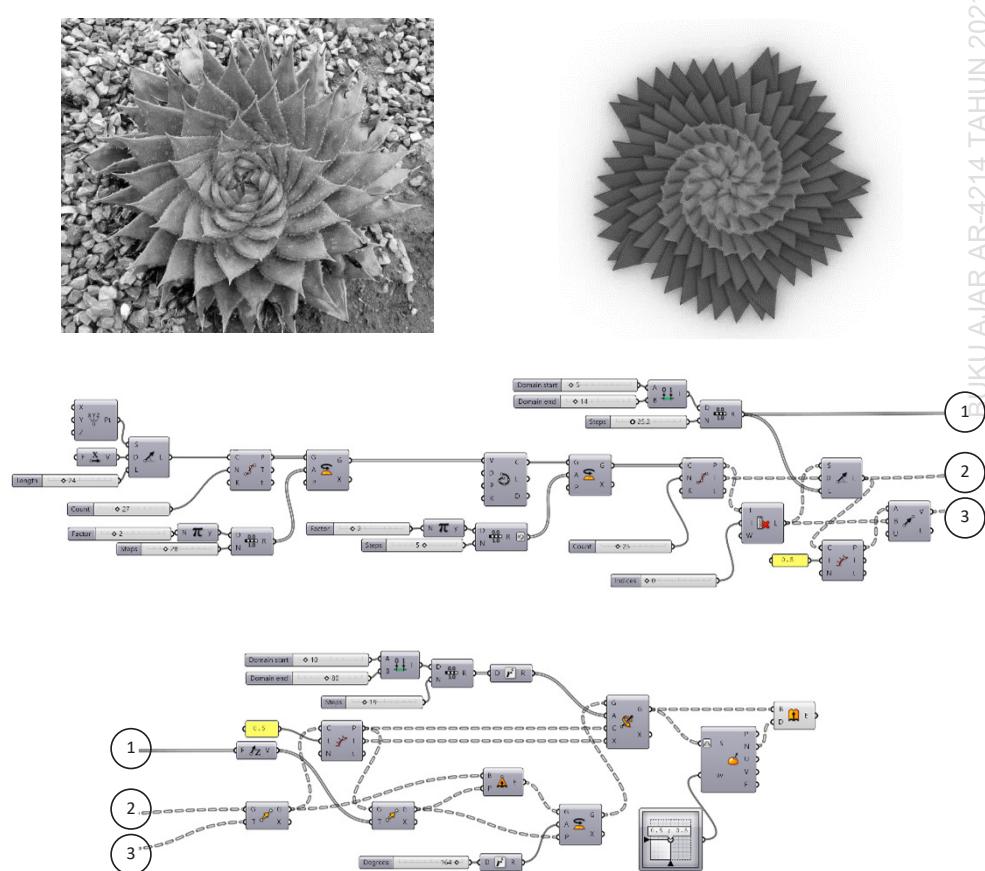
Masih berkaitan dengan bagian B, cara terbaik dalam belajar desain parametrik adalah **memahami program (*code*) yang bekerja, lalu memodifikasi sedikit demi sedikit untuk menghasilkan keluaran yang berbeda**. Karena itu penting dalam belajar desain parametrik **memahami pondasi dan cara kerja program**, alih-alih mengenali setiap fitur atau perintah yang ada dalam bahasa pemrograman.

D. Mencari Bentuk-Bentuk Baru

Desain parametrik memungkinkan kita merancang dan memanipulasi geometri menggunakan basis dan anatomis dari geometri tersebut. Jika software 3D memungkinkan kita membuat dan memanipulasi obyek menggunakan fitur-fitur yang disediakan, maka desain parametrik membuat kita mempunyai kemampuan lebih dalam. Secara analogi, desain parametrik membuat kita punya kemampuan memanipulasi obyek (*backdoor*) yang tidak disediakan oleh software 3D (*frontdoor*).

E. Memahami Desain Secara Matematis dan Komputasi

Pemahaman atas matematika, khususnya **geometri** dan **komputasi** (bagaimana memproduksi geometri tersebut) membawa konsep desain menjadi lebih fokus dan lebih kaya. Pendekatan formal ini tentunya memiliki aturan dan batasan (*constraints*) namun sejalan dengan itu, mekanisme internal (*logic*) dapat dikendalikan oleh desainer (arsitek). Pemahaman atas konsep-konsep dan prinsip matematika pada geometri khususnya akan memperkaya eksplorasi atas bentuk (Gambar 82).



Gambar 82. Spiral Aloe oleh Reni Marsinta Nahampun

F. Akurasi dan Presisi

Desain parametrik sebagai cara pandang baru dalam menyelesaikan masalah, memiliki perbedaan dalam hal keputusan-keputusan desain. Pada proses konvensional, model yang dihasilkan umumnya sudah memiliki atribut-atribut yang merepresentasikan keakuratan dan kepresisan, misalnya dimensi, posisi. Tanpa atribut-atribut tersebut, obyek belum dikatakan sebagai model. Pada desain parametrik, karena yang dirancang adalah **proses dan relasi** yang melibatkan obyek-obyek yang saling terhubung, maka atribut atas dimensi dan posisi menjadi tidak relevan dan bagian dari parameter.

G. Modular

Desain parametrik dalam bentuk diagram-diagram yang merupakan representasi program cenderung akan menjadi kompleks dan rumit dan semakin sulit dipahami. Karena hal itu, berlatih untuk **mengurangi kompleksitas dengan cara membuat modul-modul fungsi atau prosedur** (di Grasshopper dinamakan *Cluster*) menjadi penting. Setiap modul hanya memiliki input dan output minimal yang dapat direplikasi.

H. Bantu dan Ajari

Salah satu cara yang efektif dalam belajar desain parametrik dan pemrograman adalah dengan bergabung dalam **forum-forum belajar** dimana ada interaksi, saling berbagi solusi atas suatu masalah, saling berbagi kode atau program. Cara lainnya adalah dengan **mengajari orang lain**. Dengan mengajari orang lain, proses **internalisasi** dan pemahaman atas suatu konsep akan lebih terjadi.

I. Kembangkan Perangkat (*Tool*) Sendiri

Program, seperti halnya medium yang lain, adalah personal bagi banyak desainer (arsitek). Beberapa menyukai pensil yang lebih tebal dengan kertas yang agak kasar, beberapa lebih menyukai pensil yang lebih tipis, dan seterusnya. Kode-kode dan alur kerja dalam program **merefleksikan pola pikir pembuatnya**. Membuat program adalah membuat sistem untuk merancang.

3.3. Algoritma dan Data

Algoritma dan data adalah dua faktor penting dalam membuat atau mendesain suatu solusi desain parametrik. Merancang dan menuliskan algoritma memerlukan kemampuan abstraksi dan ketrampilan yang memerlukan latihan dan umumnya tidak mudah bagi para arsitek atau desainer yang sering mengandalkan intuisi. Desain suatu algoritma sangat logis dan memerlukan pernyataan-pernyataan (*statement*) yang eksplisit dari apa tujuan yang ingin dicapai dan langkah-langkah untuk mencapai hal itu. Semua proses algoritma memerlukan data dan karena itu mengetahui jenis data dan bagaimana melakukan transformasi dan manipulasi data adalah penting untuk dapat mendesain sebuah algoritma.

Data adalah perangkat untuk mendeskripsikan suatu fakta. Data ini sifatnya tak terstruktur, tidak memiliki konteks atau organisasi. Data menjadi suatu informasi jika melalui suatu proses, memiliki struktur dan konteks sehingga berguna.



Gambar 83. Hirarki Data dan Informasi (Sumber: <https://www.i-scoop.eu/big-data-action-value-context/dikw-model/>)

Perbedaan mendasar antara data dan informasi seperti pada tabel berikut:

Tabel 3.Data dan Informasi

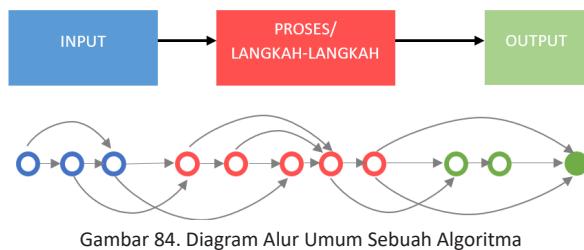
Data	Informasi
Berasal dari kata Latin: <i>Datum</i>	Berasal dari kata Latin: <i>Informare</i>
Data adalah sesuatu yang mendeskripsikan fakta	Informasi adalah data yang sudah diproses
Bisa bermanfaat atau tidak	Selalu bermanfaat
Sebagai input suatu proses, selalu diklasifikasikan sebagai data	Sebagai output suatu proses selalu diklasifikasikan sebagai informasi
Susah dipahami	Lebih mudah dipahami
Bisa jadi tidak memiliki struktur dan hirarki	Memiliki struktur dan hirarki
Contoh: data survey	Contoh: laporan survey

Pada bab 3, kita akan lebih mendalamai tentang jenis-jenis data serta propertinya untuk mendesain sebuah algoritma.

3.3.1. Desain Algoritmik

Kita dapat mendefinisikan desain algoritmik sebagai sebuah metode desain dimana keluaran (*output*) hanya bisa dicapai melalui jalinan langkah-langkah (*steps*) yang terdefinisi dengan jelas. Contohnya adalah jika kita membuat kue. Hasil akhir berupa kue yang enak, empuk (*solution-output*) didapatkan dari resep yang berisi bahan-bahan (*data-input*) dan cara memprosesnya (*algorithm*). Perubahan pada komposisi bahan-bahan akan mengubah hasil akhir.

Semua algoritma, atau nanti kita namakan sebagai definisi **Grasshopper** (GH), betapapun kompleksnya, hanya terdiri dari tiga bagian utama: input, proses, dan output.



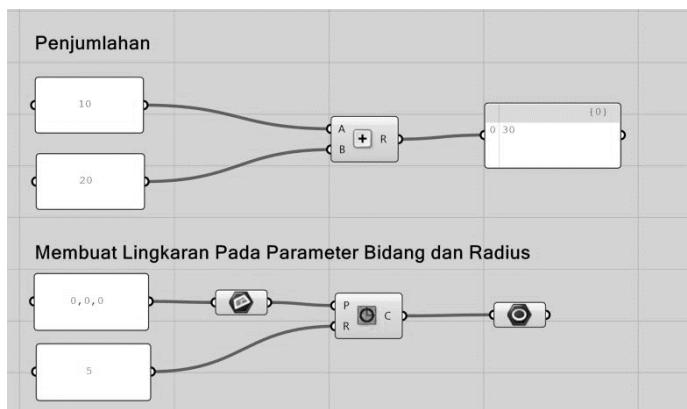
Gambar 84. Diagram Alur Umum Sebuah Algoritma

Dalam kenyataannya, seringkali dalam setiap blok komponen algoritma di atas, terdiri dari beberapa **sub-blok** dimana output satu komponen, menjadi input komponen lain dan seterusnya. Selanjutnya, proses jalannya **program tidak selalu linier**, melainkan dapat loncat, bekerja paralel, *loop*, dan lainnya. Namun yang perlu diingat adalah keluaran final selalu merupakan bagian paling kanan dari diagram alur ini sehingga aliran data (*dataflow*) selalu **dari kiri ke kanan**.

Secara umum, **membaca dan memahami** sebuah algoritma adalah langkah pertama dalam belajar desain algoritmik, setelah itu mulai mengembangkan sebuah definisi menjadi definisi yang lebih kompleks dengan hasil yang lebih kompleks. Hal yang paling penting untuk dapat menguasai pemrograman adalah mengembangkan ketrampilan (*skill*) berdasarkan pemahaman **fundamental untuk merancang algoritma** dalam menyelesaikan suatu masalah atau mencapai tujuan tertentu dari nol.

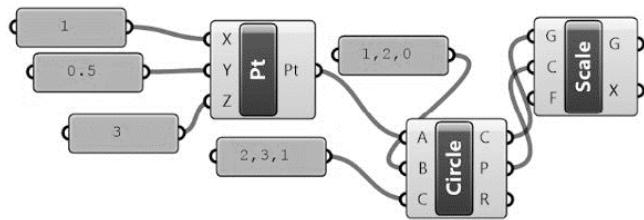
3.4. Anatomi Sebuah Definisi

Dalam GH, program atau algoritma bekerja dari kiri ke kanan (di dalam bahasa pemrograman berbasis teks, program berjalan dari atas ke bawah), dimana pada bagian paling kiri, umumnya adalah input/data dan parameter sedangkan di bagian paling kanan adalah output atau solusi. Bagian tengah umumnya berisi semua proses dan tambahan input atau output. Contohnya sebagai berikut.



Gambar 85. Algoritma (Definisi) Dasar dalam GH

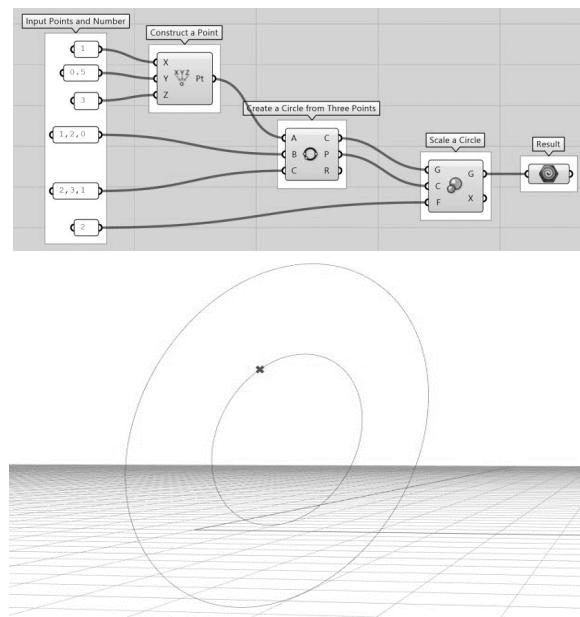
Membaca dan memberi label algoritma agar mempermudah pemahaman. Perhatikan algoritma/definisi di bawah ini.



Gambar 86. Definisi Grasshopper

Definisi terdiri dari 3 proses utama:

1. Menentukan titik: **Construct Point**.
2. Membuat lingkaran dari tiga titik: satu titik berasal dari **Construct Point**, dua titik lain ditentukan lewat **Panel**.
3. Men-skalakan obyek menggunakan komponen **Scale** dengan parameter titik pusat dan faktor skala.



Gambar 87. Definisi dan Hasil

3.5. Mendesain Algoritma: 4 Langkah Proses

Ambil contoh proses membuat kopi. Rasa sebuah kopi ditentukan oleh banyak faktor, diantaranya biji kopi, proses penyeduhan, penambahan bahan lain dan sebagainya. Ketika anda mendapatkan resep dasar membuat kopi, seperti terlihat di gambar, anda dapat mengikuti resep tersebut dan, semakin detil resepnya (artinya instruksinya jelas dari mulai kuantitas bahan, lama proses hingga peralatannya), hasil yang anda dapatkan mendekati hasil yang diharapkan oleh pembuat resep. Ketika anda semakin mahir, anda akan dapat mengubah-ubah dan mencoba hal-hal baru, misalnya teknik penyeduhan, kombinasi kopi dan lainnya yang akan menghasilkan keluaran yang berbeda juga.



Gambar 88. Analogi Algoritma

Dalam mendesain algoritma, umumnya kita bisa belajar dari solusi algoritma yang sudah ada dan perlahan-lahan mencoba mengerti dan memodifikasi algoritma tersebut untuk dapat melihat apa yang dihasilkan. Ini adalah cara umum yang dilakukan. Namun cara ini akan memiliki kelemahan. Ada kalanya setiap algoritma memiliki karakteristik dan ‘warna’ tersendiri dalam memecahkan suatu masalah. Ada ‘banyak jalan menuju Roma’, demikian pula merancang definisi/ algoritma. Memodifikasi algoritma yang kompleks terkadang jauh lebih sulit dibanding membuat algoritma dari awal karena algoritma mencerminkan pola pikir pembuat atau desainernya.

Ketika akan membuat atau mendesain algoritma baru, maka secara prinsip ada empat langkah yang harus ditetapkan:

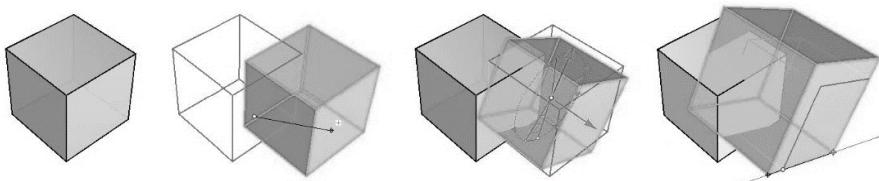
1. Menetapkan output yang jelas [**OUTPUT**].
2. Mengembangkan proses-proses utama yang diperlukan untuk menghasilkan output [**PROSES KUNCI**]
3. Menetapkan input yang diperlukan untuk proses- proses tersebut [**INPUT**].
4. Menentukan langkah-langkah dan proses diantara proses-proses utama [**INPUT DAN PROSES ANTARA**]

Mendesain sebuah algoritma adalah kemampuan/skill baru yang amat diperlukan dan ini membutuhkan kesabaran, latihan-latihan-latihan dan waktu untuk berkembang.

3.6. Berpikir Algoritmik di 3D Modeling (*Additive*) vs *Parametric Modeling (Associative)*

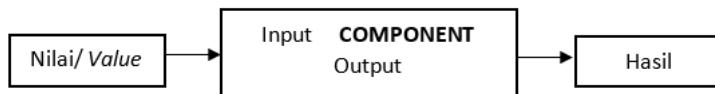
Dalam proses membuat model 3D menggunakan software pemodelan, kita sering kali secara intrinsik menerapkan berpikir algoritmik dalam langkah-langkah/ proses dan data. Proses yang umum dalam pemodelan 3D adalah:

1. Tentukan keluaran/ output (apa yang hendak dimodelkan?).
2. Tentukan perintah-perintah apa saja yang digunakan untuk menghasilkan output tersebut.
3. Dalam setiap perintah, masukkan input yang diperlukan.



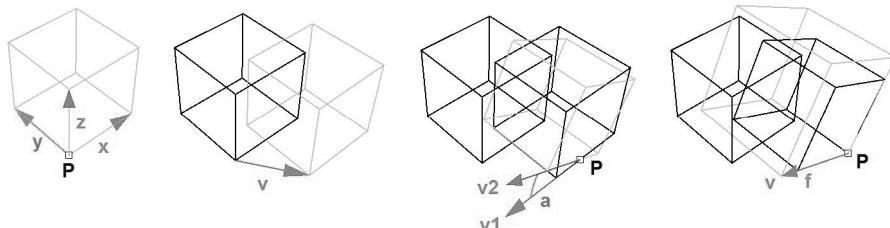
Gambar 89. Metode Aditif

Misalnya, contoh di atas ketika anda ingin menghasilkan irisan dari dua buah obyek. Pada obyek pertama, anda akan diminta memasukkan data berupa koordinat titik awal pembuatan box, dimensi box, kemudian hal yang sama untuk box kedua dan akhirnya perintah *Boolean Intersection* untuk menghitung obyek yang menjadi irisan dua obyek sebelumnya. Proses ini dilakukan secara interaktif dan langsung pada obyeknya (*direct manipulation*). Dalam proses yang dinamakan pula dengan *additive method* ini, kompleksitas pemodelan berbanding lurus dengan proses menuju keluaran final dan hasilnya sulit dimodifikasi.



Gambar 90. Alur Dasar di Grasshopper

Proses algoritmik di lain pihak, tidak interaktif dan tidak langsung. Proses ini memerlukan artikulasi eksplisit dari setiap data dan setiap proses. Pada contoh di atas misalnya, anda harus menentukan koordinat titik awal pembuatan box, orientasi dan dimensi. Ketika anda akan menduplikasi obyek, anda harus menentukan vektor perpindahan obyek, bidang perpindahan (*plane*) dan seterusnya. Pada proses yang dinamakan pula dengan *associative method* ini, semua tahapan proses secara eksplisit dibentuk oleh jalinan perintah (komponen) sehingga keluaran final dengan mudah dapat dimodifikasi sesuai dengan komponen-komponen pembentuknya.



Gambar 91. Metode Asosiatif

3.7. Pemrograman Visual dengan Grasshopper

Grasshopper (GH) adalah *Visual Programming Editor* yang dikembangkan oleh David Rutten di Robert McNeel & Associates, pengembang software Rhinoceros. Sebagai plug-in gratis di Rhinoceros, Grasshopper terintegrasi dengan semua fitur yang ada di Rhinoceros (RH). Rhinoceros dan Grasshopper menyediakan sarana untuk mengembangkan **mekanisme kontrol secara parametrik** terhadap suatu model, memiliki kapabilitas untuk melakukan eksplorasi desain secara generatif dan sebuah platform untuk mengembangkan suatu bahasa pemrograman tingkat tinggi (*high level programming logic*) dalam sebuah sistem yang intuitif dan berbasis grafis.

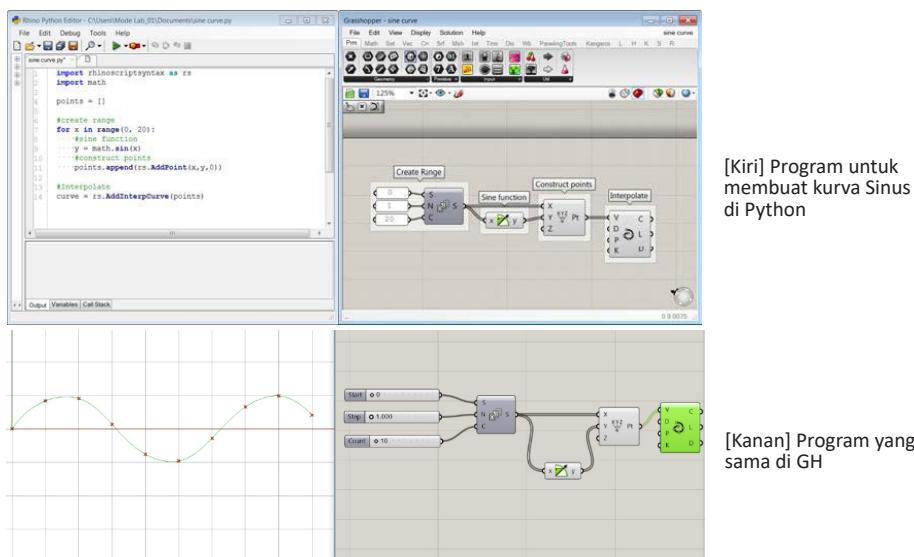
Visual programming adalah suatu paradigma dalam pemrograman komputer (*computer programming*) dimana pengguna melakukan semua operasi terhadap elemen-elemen pemrograman menggunakan **elemen-elemen grafis alih-alih teks**, yang terdapat

pada bahasa pemrograman pada umumnya misalnya: C++, C#, Visual Basic, Processing, atau yang paling dekat dengan Rhinoceros (RH) adalah Python dan Rhinoscript. Dalam Grasshopper (GH), kita menghubungkan blok-blok fungsional (*component*) dalam suatu jalinan koneksi yang sekuensial untuk memproduksi hasil yang diinginkan.

Di dalam pemrograman visual kita merepresentasikan logika komputasi dengan cara membuat atau mengkonstruksi blok-blok diagram (*graph*) dimana setiap diagram ini merupakan jalinan komponen (*Node*) dan kabel-kabel (*Wire/Edge*) sebagai penghubungnya. Setiap komponen/*Node* merupakan sebuah kompartemen (paket) yang bisa merupakan kontainer (penyimpan data/nilai) atau berupa fungsi yang melakukan operasi terhadap data. Karena itu, setiap komponen akan memerlukan input beserta parameternya, melakukan transformasi atas input dan menghasilkan output beserta parameternya.

Setiap komponen selalu memerlukan input yang dihubungkan di sebelah kiri komponen dan menghasilkan output yang letaknya di sebelah kanan komponen tersebut.

Metode pemrograman visual ini disebut juga sebagai “*black box*” karena kita tidak perlu tahu atau tidak perlu khawatir dengan bagaimana komponen melakukan fungsinya. Kita cukup memahami apa fungsi komponen tersebut, apa inputnya dan apa outputnya.



Gambar 92. [Kiri] Hasil Program di RH; [Kanan] Definisi Program di GH

3.8. Instalasi dan Add-ons

Pada RH versi 6, GH sudah otomatis menjadi salah satu **fitur standard** yang bisa diakses melalui ikon yang ada di Toolbar pada Rhinoceros. GH juga dapat dipanggil menggunakan Command Line: *Grasshopper*. Saat ini GH memiliki banyak Add-in yang umumnya gratis. Anda dapat men-download beberapa *add-ons* tersebut pada beberapa situs seperti: **Food4Rhino** dan lainnya. Proses instalasi *add-ons* ini umumnya cukup

mudah dan dijelaskan pada situs masing-masing penyedia *add-ons*. Namun secara umum, hal yang harus selalu diperhatikan adalah selalu *unlock* file-file yang akan di-instalasi.



Gambar 93. Salah satu add-ins GH yakni Kangaroo untuk pemodelan dan simulasi fisik

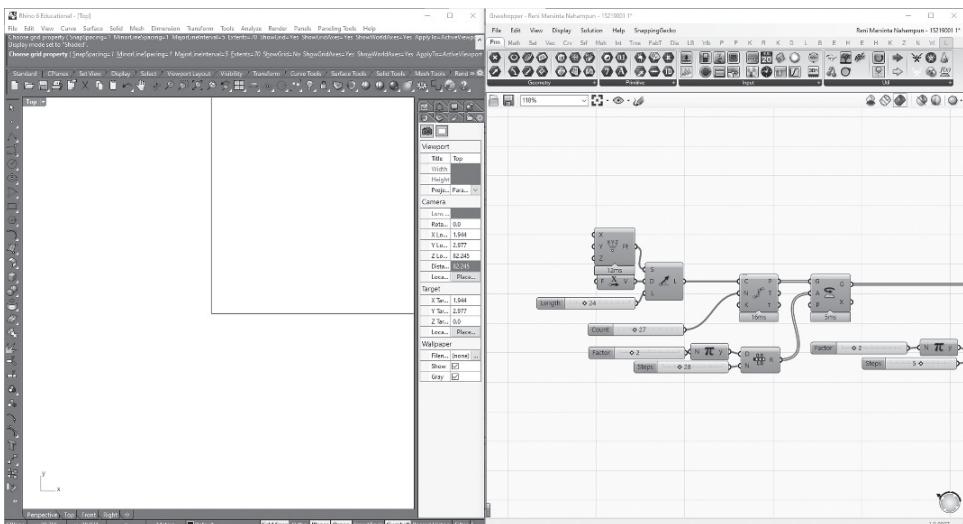
Beberapa add-ons yang esensial diantaranya:

- User Interface*
 - Bifocals: menampilkan ikon sekaligus teks pada setiap komponen.
 - Snapping Gecko: memiliki fitur *object snapping* yang berguna untuk merapikan komposisi komponen.
- Paneling & Tesselation*
 - Paneling Tools: memiliki fitur-fitur untuk membuat panel.
 - Lunchbox: memiliki beberapa fitur untuk geometri baru, panel, struktur dan juga analisis menggunakan *machine learning*.
 - Heteroptera: Memiliki koleksi komponen untuk simulasi dan generasi pola dan obyek.
- Environmental Simulation*
 - Ladybug: memiliki fitur -fitur simulasi *daylight*, *glare*, *energy*, *fluida*.
- Form-finding-Structural Simulation*
 - Kangaroo Physics: memiliki fitur-fitur untuk simulasi fisik dan *form-finding*.
- Mesh Modeling*
 - Weaverbird: memiliki fitur-fitur untuk memodifikasi mesh.
- Optimization*
 - Octopus, Wallacei: mesin evolusionari untuk optimasi multi obyektif.
- Pattern*
 - Parakeet: memiliki koleksi beberapa komponen untuk men-generasi pola
- Loop*
 - Anemone: memiliki fitur untuk melakukan operasi *loop*.

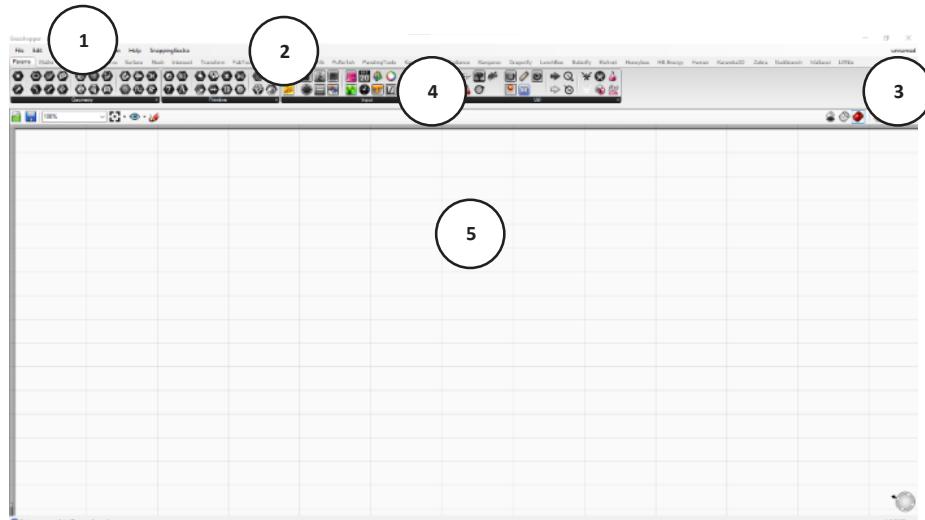
3.9. User Interface

Bekerja dengan GH pada umumnya tidak lepas dari RH karena semua proses dan hasil geometri atau simulasi akan divisualisasikan di RH. Karena itu jendela RH dan GH pada umumnya terletak berdampingan untuk memudahkan ketika bekerja dengan kedua software ini.

Navigasi di canvas GH sama persis dengan navigasi di viewport RH. *Drag + right click* untuk Pan dan *scroll up-down* untuk Zoom In dan Zoom Out.

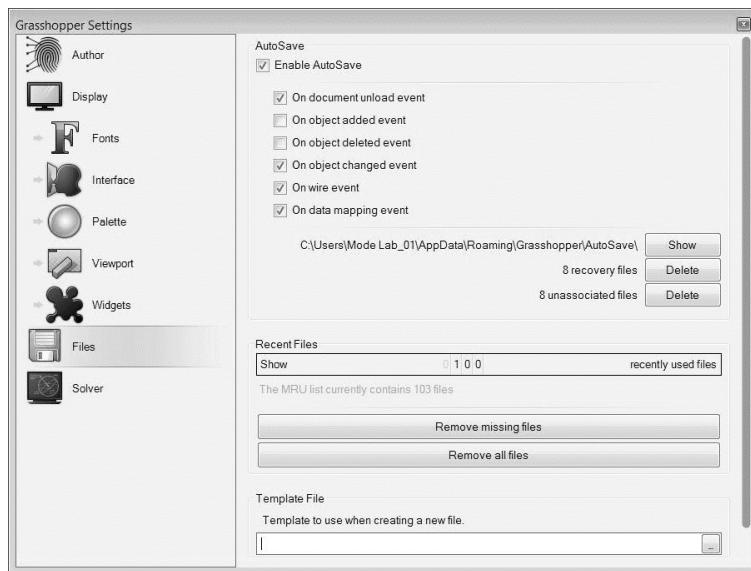


Gambar 94. [Kiri] Rhinoceros; [Kanan] Grasshopper



Gambar 95. Grasshopper User Interface

1. **Windows Title Bar:** Judul software dan nama file yang sedang dibuka
2. **Main Menu Bar:** New, Open, Save (File GH berformat.GH atau.GHX), dan lainnya.
 - **Export Quick Image & Export Hi-Res Image:** menyimpan definisi program sebagai image/ image resolusi tinggi
 - **Preferences:** Setting program



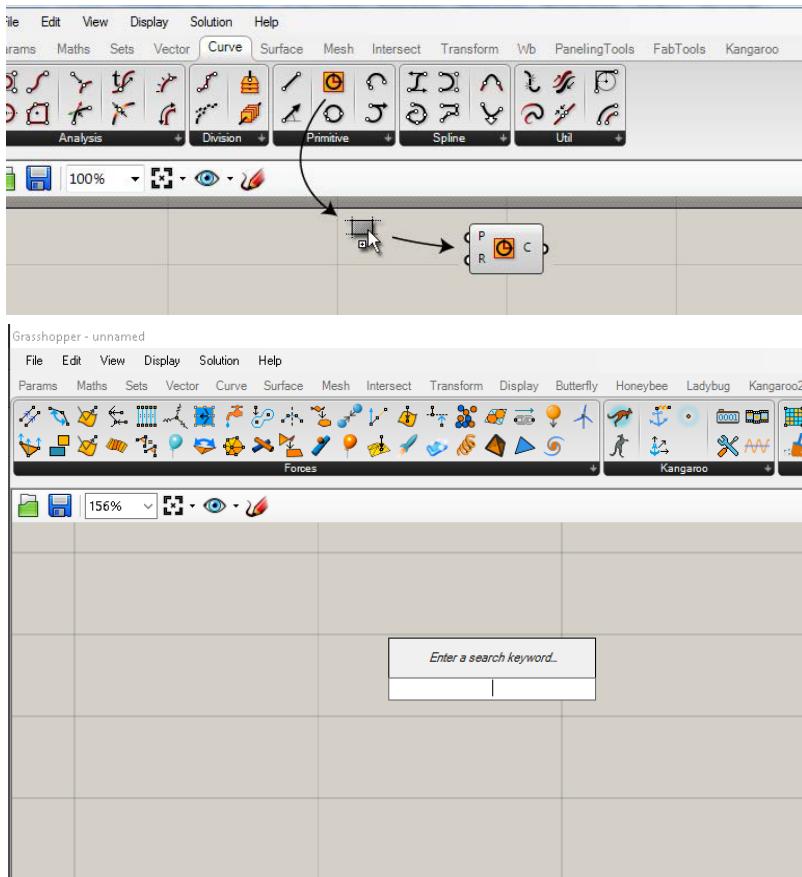
Beberapa hal penting dalam Preferences (fitur yang lain bisa dibiarkan *default*) adalah:

- Author: isi dengan nama dan identitas, termasuk misalnya alamat email. Ini akan menjadi pengenal file definisi.gh anda.
- Interface > Context Menu > Show Parameter submenus
- Interface > Overlay Graphics > Display obsolete
- Viewport > Preview Display > Display Gumballs
- Widget > Zooming widget: nilai antara 300%-600%
- Files > Autosave
- Files > Recent files : 20
- Files > Template File > Buat file yang akan menjadi template
- **Special Folder:** folder- folder dimana GH menyimpan file-file instalasi termasuk add-ins. Umumnya proses instalasi add-ins di GH dilakukan dengan meng-copy file instalasi tersebut ke dalam folder-folder ini. Dua folder utama sebagai tempat instalasi adalah:
 - **Component folder:** C:\Users\[nama komputer]\AppData\Roaming\Grasshopper\Libraries
 - **User Object folder:** C:\Users\[Nama Komputer]\AppData\Roaming\Grasshopper\UserObjects
- 3. File Browser Control: Jika anda membuka file definisi lebih dari satu, maka anda dapat memilih file apa yang anda akan buka melalui tombol ini.
- 4. Component Pallete: Semua komponen di GH diorganisasikan ke dalam palet ini. Semua ikon komponen dikategorikan dalam bentuk Tab dan sub-kategori dalam bentuk drop-down panel. Setiap Tab memiliki judul yang menggambarkan komponen-komponen yang memiliki kategori yang sama.

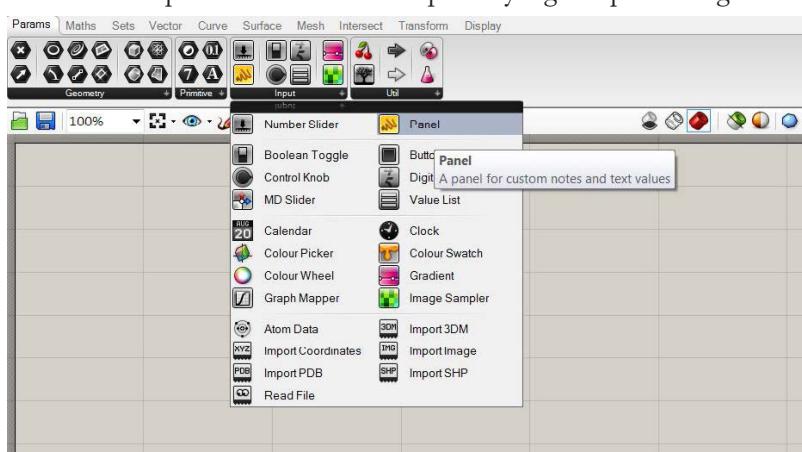
Ada dua cara untuk memasukkan ikon komponen ke dalam canvas:

- *Click+drag* komponen ke dalam canvas.

- Double-click pada canvas yang kosong dan cari komponen yang akan dimasukkan.

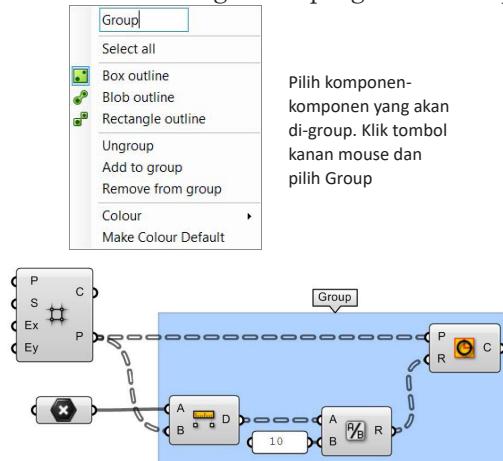


Pada setiap kategori di setiap tab, umumnya terdapat tombol panah kecil di bagian kanan untuk memperlihatkan semua komponen yang ada pada kategori tersebut.

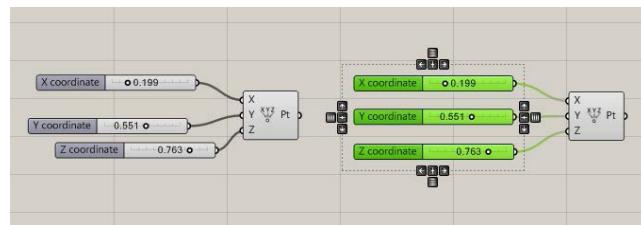


5. **Canvas:** adalah area dimana anda membuat program atau definisi menggunakan semua komponen yang ada.

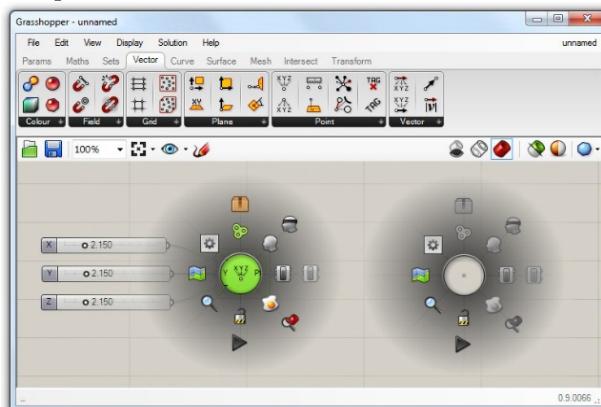
- **Group:** Beberapa jalanan komponen dan konektor dapat di-group untuk memudahkan memberi keterangan dan pengelolaan komponen.



- **Align Widget:** digunakan untuk merapihkan susunan komponen: rata kiri, rata kanan atau rata tengah baik horizontal maupun vertikal. Widget ini otomatis muncul jika anda memilih lebih dari satu komponen.



- **Menu Radial:** klik tombol tengah mouse untuk menampilkan menu radial yang berisi beberapa fitur:



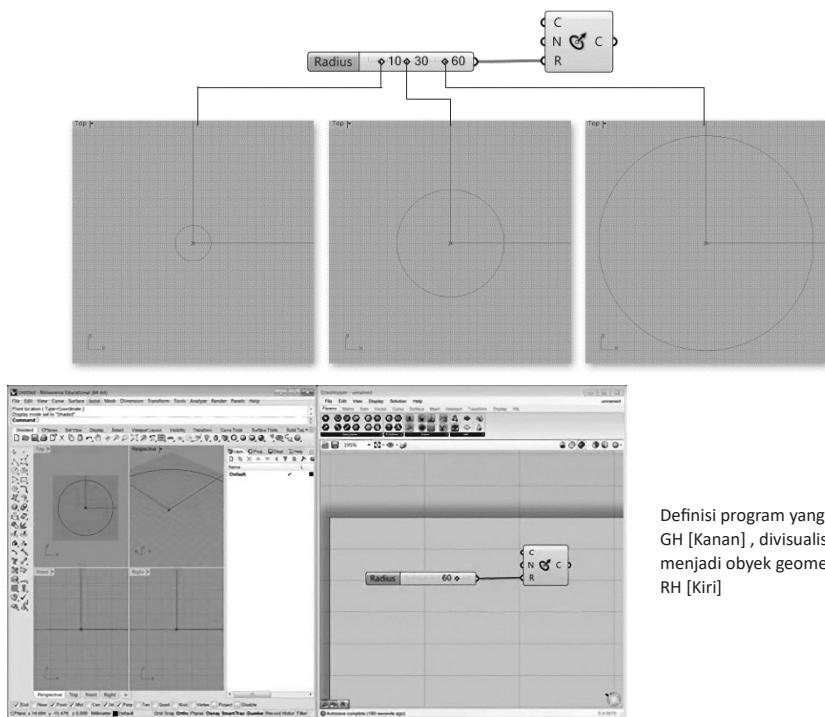
- Cluster/Group
- Hide/Preview
- Enable/Disable
- Bake

- Preference
- Find
- Disable Solver
- Recompute
- Canvas Toolbar: Toolbar pada canvas berisi tombol/ikon yang sering diakses:
 - Save File
 - Open File
 - Zoom
 - Named View
 - Sketch Tool
 - Preview Setting (wireframe, shaded)
 - Preview only selected object
 - Preview mesh quality



3.10. Koneksi dengan Rhinoceros

Hal pertama yang harus dipahami sebagai prinsip pemrograman di GH adalah secara *default* GH tidak menyimpan obyek riil atau geometri riil. Definisi GH merupakan himpunan perintah/aturan (rules) yang dijalankan oleh RH.



Gambar 96. Koneksi Live dan Auto-update

Definisi program yang ada di GH [Kanan], divisualisasikan menjadi obyek geometri di RH [Kiri]

Umpulan Balik Pada Viewport (Viewport Feedback)

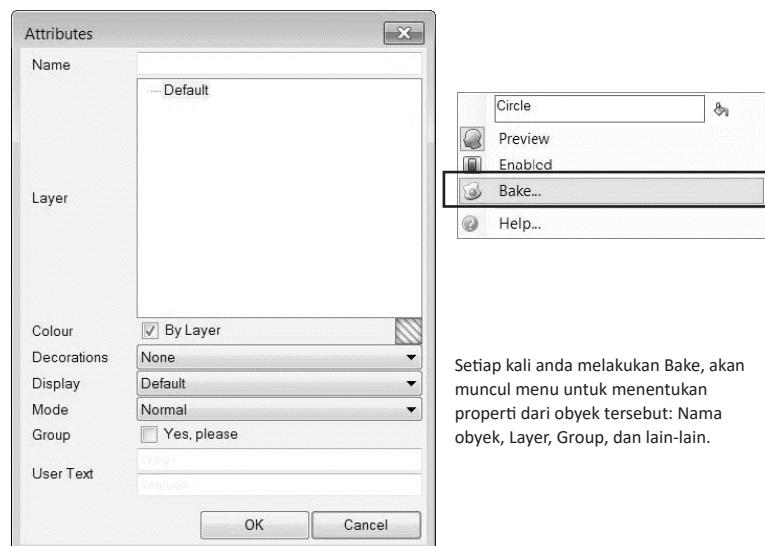
Semua geometri yang dihasilkan oleh definisi GH secara default akan ditampilkan di semua viewport di RH. Tampilan ini hanyalah preview sehingga anda tidak dapat memilih dan memodifikasi obyek ini di RH, anda mengedit dan memodifikasi obyek ini di GH.

Koneksi Live

GH adalah lingkungan pemrograman dinamis dimana setiap perubahan yang dilakukan di GH akan otomatis berdampak pada preview obyek tersebut di RH.

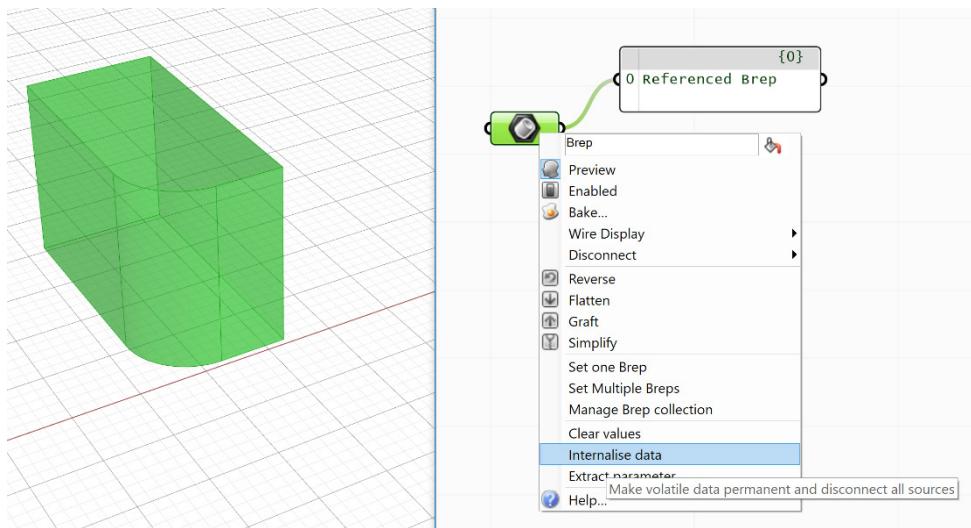
Baking Geometry

Untuk dapat memodifikasi obyek yang dihasilkan oleh GH, maka obyek tersebut harus *di-bake*. Fitur Bake ini ada di setiap komponen di GH. Komponen yang *di-bake* artinya akan menjadi obyek RH dan tidak lagi memiliki koneksi dengan GH. Klik kanan tombol mouse pada bagian output port dari setiap komponen untuk menemukan fitur Bake.



Manajemen File

Jika definisi GH anda memerlukan obyek referensi dari RH, maka anda harus membuka file GH dan file RH agar definisi tersebut dapat berjalan baik. Jika anda ingin semua geometri referensi di Rhinoceros disimpan dalam file Grasshopper, maka anda harus melakukan *internalisasi* obyek-obyek tersebut ke dalam komponen yang sesuai di Grasshopper. Internalize ini maksudnya adalah obyek referensi dari Rhinoceros disimpan di dalam komponen Grasshopper sehingga anda tidak perlu membuka file Rhinoceros yang berisi geometri yang diperlukan.



Gambar 97. Internalize Data

3.11. Beberapa Hotkeys atau Shortcut

Berikut adalah beberapa *shortcut* untuk mempercepat proses pembuatan definisi dalam kanvas GH:

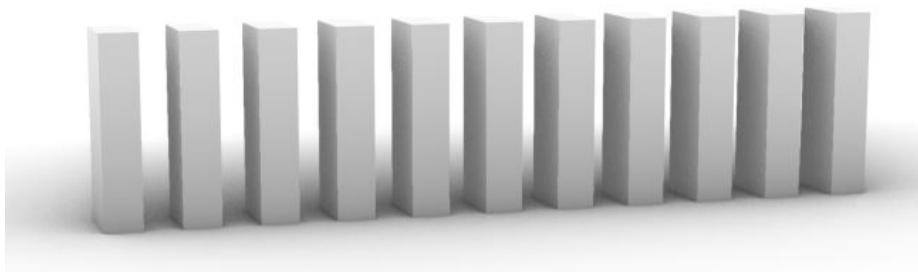
- Memunculkan komponen Panel: “ // ” atau //
- Memunculkan komponen Number Slider dengan rentang nilai tertentu: x<y (ganti x dan y dengan angka)
- Duplikasi komponen: Shift + Alt + drag komponen bersangkutan
- Komponen Preview/Hide : CTRL + Q
- Komponen Enabled/ Disabled: CTRL+E
- Grouping: CTRL + G
- Komponen Bake: Insert
- Zoom: CTRL + drag tombol kanan mouse
- *Split canvas*: ALT + drag tombol kiri mouse pada area kosong kanvas, tap tombol ALT untuk mengganti aksis *split canvas*: vertikal atau horizontal.

Latihan 4: Konsep Dasar Grasshopper

1. Apa keunggulan metode asosiatif di GH dan metode aditif di RH?
2. Mengapa anda harus merencanakan algoritma sebelum bekerja menggunakan algoritma?
3. Bagaimana aliran data bekerja di GH?
4. Apa salah satu algoritma dasar yang tidak tersedia komponennya secara default di GH?

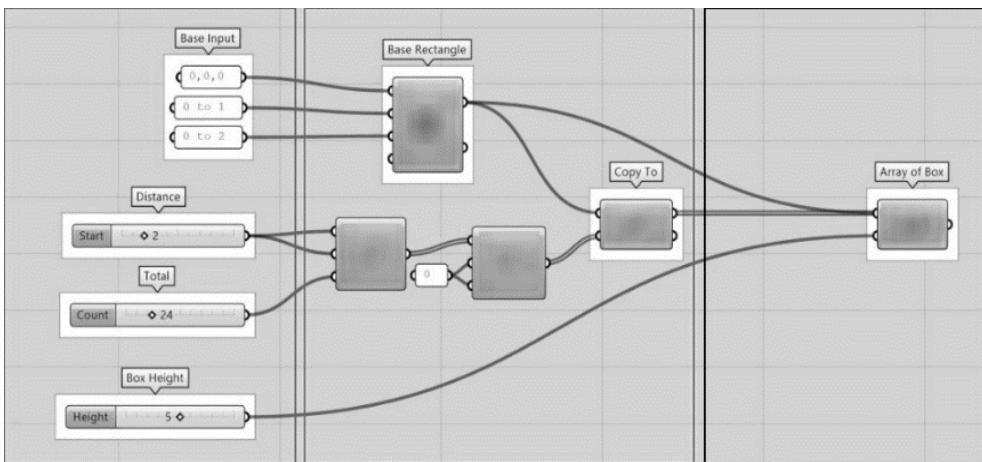
Buatlah definisi sederhana di GH untuk sketsa berikut. Jangan lupa, dalam pemrograman, yang ditentukan paling awal adalah outputnya, lalu perkiraan prosesnya, baru kemudian input-inputnya.

1. Penjumlahan
 - Input: 2 angka
 - Proses: **Addition**
2. Membuat Lingkaran
 - Input: koordinat *Plane*
 - Input: radius lingkaran
 - Proses: **Circle** (Perhatikan ada beberapa jenis komponen dengan nama yang sama)
3. Membuat garis
 - Input: koordinat titik 1, koordinat titik 2
 - (Proses antara dan input): titik, titik 2. Buat titik menggunakan data koordinat
 - Proses: **Line**
4. Dari gambar berikut, perkirakan bagaimana tahapan membuatnya:

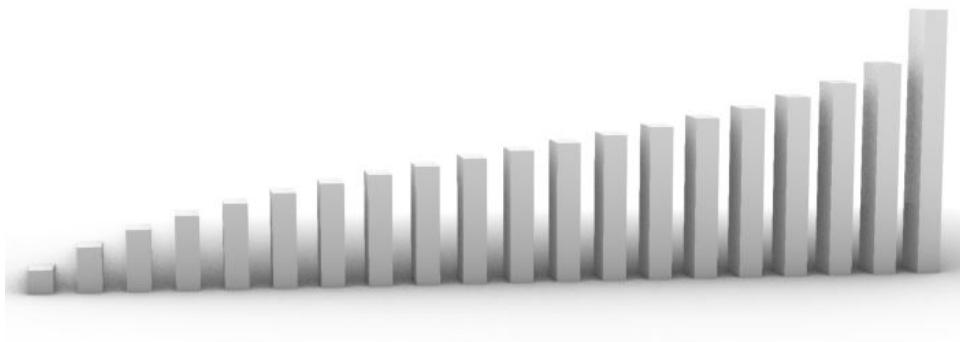


Hints:

- Hasil akhir berupa rangkaian box dengan tinggi sama.
- Proses dimulai dari membuat base rectangle yang diduplikasi.



5. Pengembangan dari no.4, perkirakan bagaimana definisi berikut:



Hints:

- Tinggi hasil akhir merupakan fungsi grafik
- Input fungsi grafik adalah posisi dari setiap base rectangle.

3.12. Anatomi Sebuah Definisi

Bagian sebelumnya kita berkenalan dengan UI Grasshopper, sebuah editor algoritma visual yang memungkinkan anda merancang algoritma untuk memecahkan masalah dan mencari solusi. Kita juga berkenalan dengan prinsip dan langkah-langkah utama dalam merancang sebuah definisi/ algoritma.

Di dalam GH, anda akan merancang definisi/ program visual yang terdiri dari jalinan komponen (*node*) dan kabel-kabel (*edge/wire*).

3.12.1. Jenis Obyek

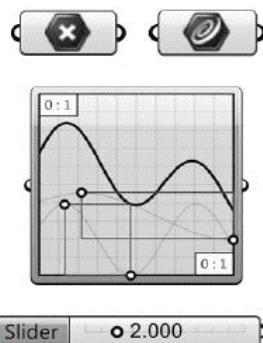
Secara umum ada dua jenis obyek:

1. Parameter/ kontainer: adalah obyek yang menyimpan data.
2. Komponen: adalah obyek yang memproses data.

Parameter/ Kontainer

Parameter/kontainer menyimpan data. Anggap ini adalah sebuah laci dalam sebuah lemari untuk menyimpan beberapa jenis data: angka, warna, geometri, teks yang bisa didapatkan dari beberapa cara: output dari proses sebelumnya, dari RH atau dari file lain.

Umumnya parameter atau kontainer memiliki **satu port input dan satu port output**. Ikon parameter umumnya berupa hexagonal hitam dengan lambang jenis data yang disimpan.

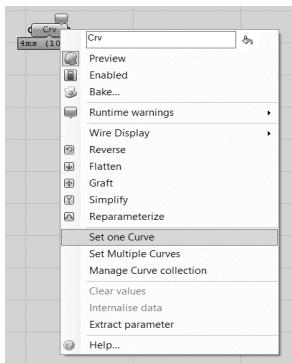


Kontainer point, geometry menyimpan geometri dari RH.

Parameter lain berupa input parameter: **Graph Mapper** dan **Number Slider** menyimpan data dinamis karena pada data yang disimpan, masing-masing dari data tersebut dapat digunakan untuk eksekusi program.

Contoh cara untuk mendefinisikan sebuah objek dari RH ke dalam salah satu container GH adalah:

- 1) Klik kanan pada container **Curve** di GH dan pilih *Set One Curve*, atau jenis objek lain tergantung container masing-masing.



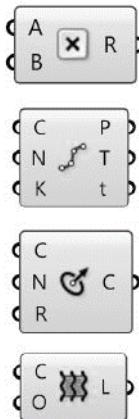
- 2) Klik objek yang ingin didefinisikan di viewport RH.
- 3) Langkah ini juga dapat dibalik urutannya, silahkan lakukan yang memudahkan pekerjaan.

Selain mendefinisikan 1 buah objek saja, GH juga bisa mendefinisikan banyak objek sekaligus. Pilihan *Set Multiple Curve*, atau jenis objek lain dapat dipakai untuk pendefinisian ini.

Komponen

Komponen merupakan obyek untuk **memproses input** dan melakukan aksi (*action*). Ada banyak komponen dan fungsinya masing-masing.

- **Multiplication:** perkalian dua angka.
- **Divide Curve:** Membagi kurva menjadi segmen yang sama panjang.

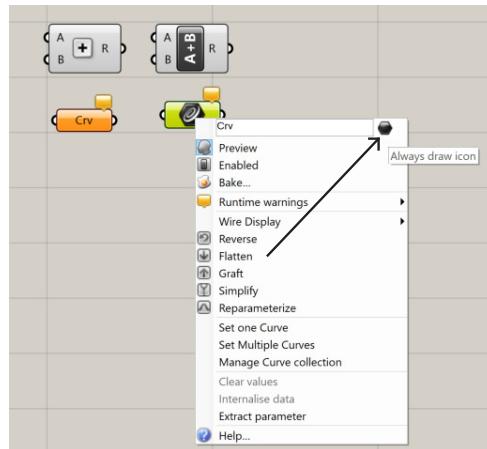


- **Circle CNR:** membuat lingkaran dari tiga input: *Center*, *Normal Vector* dan *Radius*.
- **Loft:** membuat surface dari beberapa kurva profil.

3.12.2. Label

Setiap komponen pada GH dapat ditampilkan menggunakan ikon atau label nama berupa teks. Cara mengubah setting ini adalah dengan klik kanan pada bagian tengah

komponen (bagian teks/icon) dan pilih *Always draw icon* atau *Always draw name* atau *Use application setting*.



- Klik kanan pada bagian tengah suatu komponen.
- Pilih setting label.

3.12.3. Warna Obyek

Beberapa kode warna pada komponen dan parameter memiliki maksud tertentu sebagai berikut.



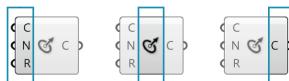
- [ABU MUDA dengan TITLE PUTIH] Komponen memiliki data di dalamnya dan tidak error.
- [ABUTUA dengan TITLE PUTIH] Komponen memiliki data dan disembunyikan (hide).
- [HIJAU] Komponen diseleksi.



- [ABU TUA dengan TITLE HITAM] Komponen memiliki data di dalamnya dan DI-DISABLE/ tidak dieksekusi.
- [ORANYE] Komponen tidak memiliki data di dalamnya/ tidak ada operasi yang dilakukan. Perhatikan tooltip yang ada pada pojok kanan atas komponen untuk mencari tahu jenis data apa yang diperlukan.
- [MERAH] Komponen error. Perhatikan tooltip yang ada pada pojok kanan atas komponen untuk mencari tahu apa error-nya.

3.12.4. Bagian-Bagian Komponen

Sebuah definisi program hanya bisa berjalan atau dieksekusi jika jalinan antara komponen yang dihubungkan dengan kabel dapat bekerja. Setiap komponen memiliki **port input (kiri)** dan **port output (kanan)**.



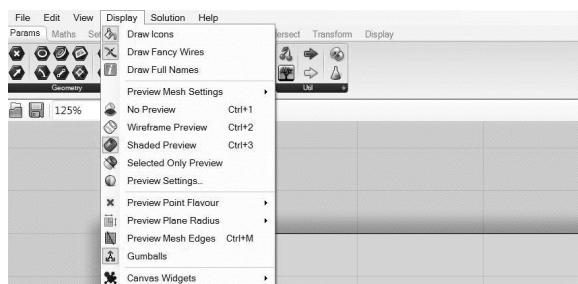
- Tiga input port pada komponen Circle CNR menentukan jenis input apa yang diperlukan.
- Bagian tengah adalah nama komponen. Anda dapat mengaktifkan contextual menu dengan jalan klik kanan pada bagian ini.
- Output port berupa circle/ lingkaran



- Ada beberapa komponen yang tidak memiliki output port, biasanya bagian kanan berupa sisi bergerigi/ jagged.
- Umumnya komponen ini tidak menghasilkan obyek tetapi hanya membantu memvisualisasikan.

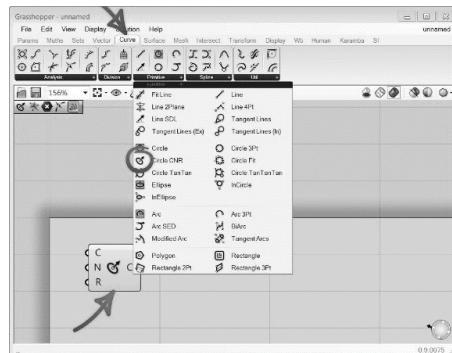
3.12.5. Display Label dan Ikon

- Setiap komponen dapat ditampilkan menggunakan dua cara: menggunakan ikon atau menggunakan teks. Anda dapat mengakses setting ini pada **Display → Draw Icon**, atau dengan klik kanan pada komponen bersangkutan dan pilih **Always Draw Icon** atau **Always Draw Name** pada setting di sebelah nama komponen.





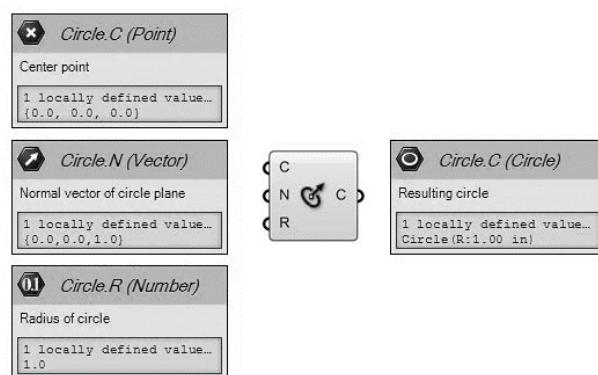
- Beberapa komponen memiliki nama yang sama tapi ikon berbeda. Contohnya adalah komponen **Circle CNR** dan **Circle 3pt** di atas.



- Jika anda ingin mengetahui lokasi suatu komponen pada toolbar, anda dapat menekan **CTRL + ALT +klik** komponen bersangkutan.

3.12.6. Tooltips

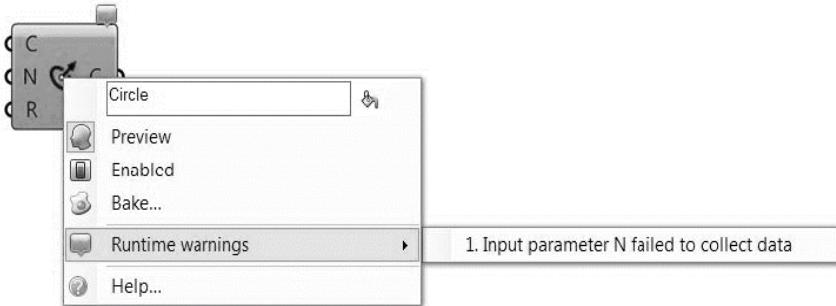
Bagian input dan bagian output dari suatu komponen memiliki fungsi masing-masing. Pada bagian input, setiap port mengharapkan anda untuk memberikan input berupa **jenis data tertentu**. Bagian output memberikan infomasi hasil proses yang terjadi dalam komponen tersebut. Anda dapat mengetahui apa-apa yang diharapkan oleh masing-masing port melalui Tooltips. Tooltips ini akan muncul ketika mouse berada di atas port bersangkutan. Tooltips ini sangat berguna karena memberikan informasi jenis data dan jumlah data yang ada di setiap port.



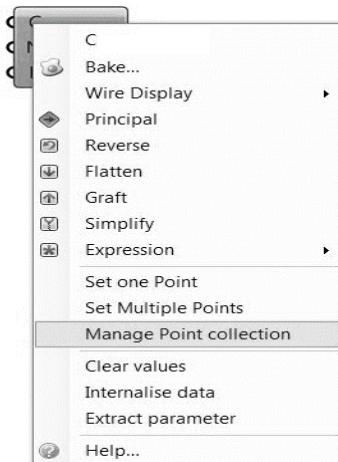
- Tooltips pada masing-masing input port memberikan informasi jenis data apa yang harus dimasukkan.
- Tooltips pada output port memberikan informasi keluaran komponen bersangkutan.

3.12.7. Contextual Popup Menu

- Klik kanan pada bagian tengah/nama komponen untuk mengakses menu kontekstual.
- Ini adalah cara cepat untuk: Show/hide, Enabled/disabled, Bake, dan lainnya.



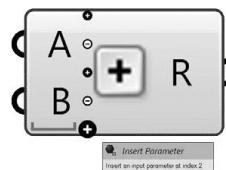
- *Wire Display*: setting untuk menampilkan kabel: default, tipis/ faint, atau tidak ditampilkan/ hidden.
- Bagian atas dari kontekstual menu terkait dengan obyek, koneksi dan perlakuan terhadap struktur data dalam komponen.
- Bagian kedua terkait dengan input data yang disimpan



- Bagian ketiga terkait dengan modifikasi data dalam komponen: dihapuskan, diinternalisasi atau dieksternalisasi/ekstrak.

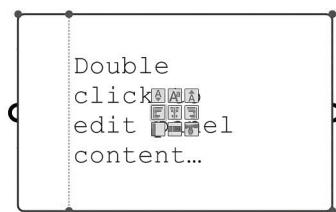
3.12.8. Zoomable User Interface

Setiap komponen dapat di-zoom untuk melihat apakah input port dapat ditambahkan atau dikurangi. Pada beberapa komponen kita dapat menambahkan atau mengurangi parameter pada input port.

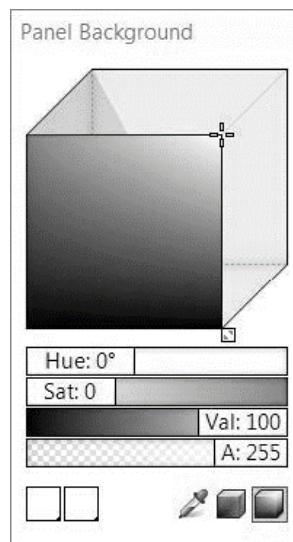


- Tekan tanda (+) untuk menambahkan parameter input atau sebaliknya, tekan tanda (-) untuk mengurangi parameter input.

Komponen panel juga memiliki beberapa setting, diantaranya, kita dapat mengubah warna latar, mengganti jenis dan besar huruf.



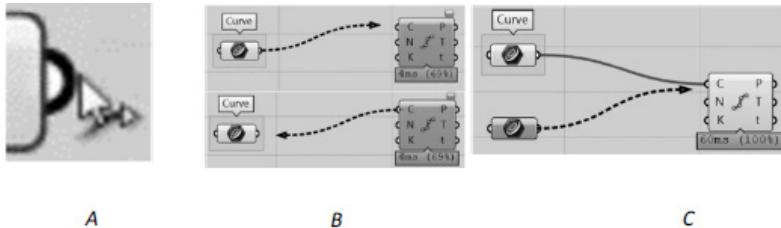
- Tekan ujung-ujung komponen panel untuk mengatur posisi teks.
- Klik kanan untuk munculkan menu kontekstual, pilih *Color*.



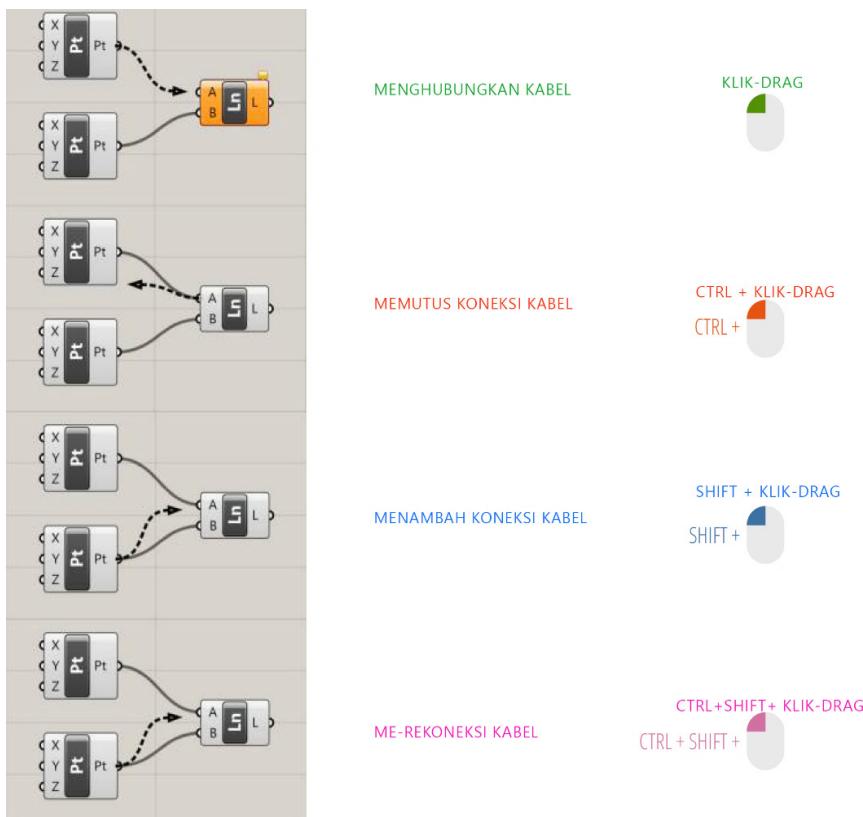
3.12.9. Menghubungkan Kontainer/Komponen dengan Kabel

Untuk menghubungkan kontainer/komponen dengan kabel, arahkan kursor menuju salah satu port dari komponen sampai kursor berubah bentuk menjadi seperti Gambar A. Kemudian, click dan drag ke arah port yang akan dituju dan lepaskan. Saat kursor sedang men-drag, kabel dengan garis putus-putus akan muncul seperti pada Gambar B.

Dalam beberapa kasus, kita perlu menghubungkan lebih dari 1 kabel ke dalam port yang sama. Untuk melakukan hal tersebut, tahan **Shift** saat sedang men-drag kabel ke port. Jika kita tidak menekan tombol **Shift**, kabel yang sebelumnya sudah terhubung ke port tersebut akan lepas dan digantikan dengan kabel yang baru (Gambar C).



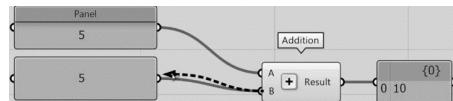
Berikut adalah rangkuman teknik menghubungkan kabel, memutus koneksi kabel, menambah koneksi kabel dan me-rekoneksi kabel.



Gambar 98. Menghubungkan Komponen dengan Kabel

Latihan 5: Bekerja dengan Grasshopper

1. Tombol apakah yang digunakan untuk memutus relasi kabel pada suatu komponen?

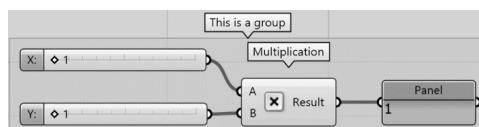


- a. Alt;
- b. Shift;
- c. Alt+Shift;
- d. Ctrl;
- e. Tab

2. Apakah arti ketika suatu komponen berwarna oranye?

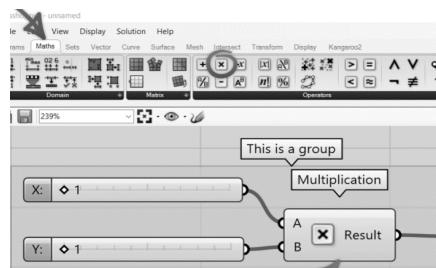
- a. Error;
- b. Disable;
- c. Empty;
- d. Memiliki bilangan genap;
- e. Memiliki bilangan ganjil

3. Apakah shortcut untuk membuat Group?



- a. Tab+Shift;
- b. Shift+Alt;
- c. Ctrl+G;
- d. Shift+Ctrl;
- e. Bukan yang di atas

4. Apakah shortcut untuk menunjukkan lokasi komponen?



- a. Tab+Spacebar;
- b. Shift+Tab;
- c. Shift+Esc;
- d. Ctrl+Alt+klik tombol kiri mouse;
- e. Ctrl+Esc

4. DATA

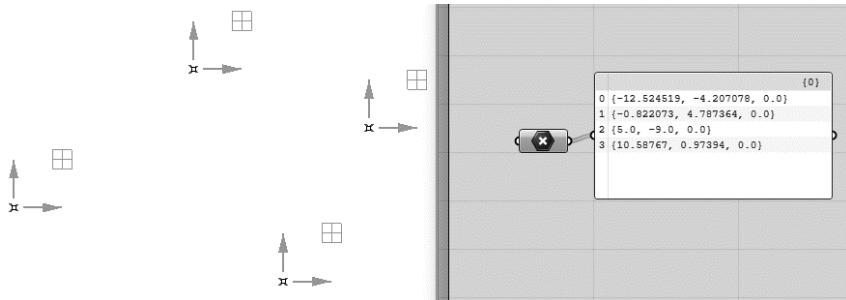
File latihan di bab ini dapat diunduh di:
<https://github.com/aswinindra/eskacangmerah>

4.1. Konsep Persistent dan Volatile Data

Dalam GH, secara umum ada dua cara data dimasukkan dalam suatu komponen: baik itu komponen sebagai kontainer (penyimpanan data) maupun komponen sebagai pemroses data.

1. Internally set data

Data yang dimasukkan ke dalam komponen secara **manual**. Data ini juga menjadi **konstan** atau tidak berubah pada komponen bersangkutan dan disimpan dalam file GH/GHX. Ini dinamakan *Persistent Data*.

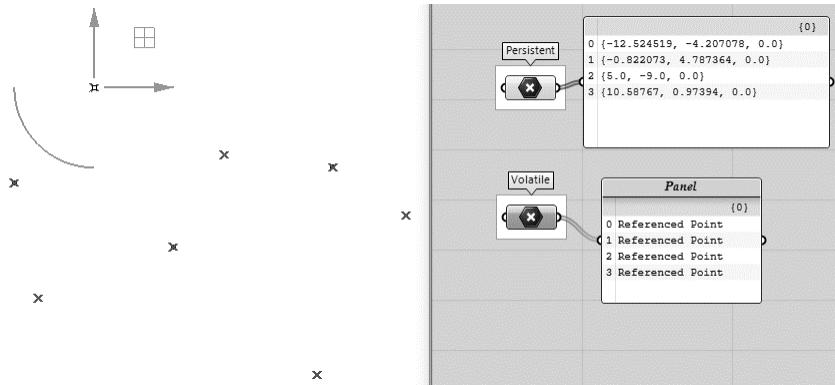


- Ketika anda memasukkan data dengan membuat beberapa Point pada komponen **Point** (melalui *Set Multiple Point*)
- Maka pada komponen tersebut tersimpan data berupa koordinat Point yang persistent. Jika ada pilih komponen Point, kemudian anda pilih salah satu **Point** di Rhinoceros dan anda pindahkan, informasi koordinat Point tersebut langsung ter-update di definisi GH.
- Obyek **Point** yang dimasukkan dalam komponen GH tersebut jika anda set menjadi *internalized data*, telah menjadi bagian dari komponen Point. Anda tidak harus menyimpan file Rhino untuk membuat komponen ini bekerja, karena secara internal, sudah tersimpatis obyek Point di dalamnya.

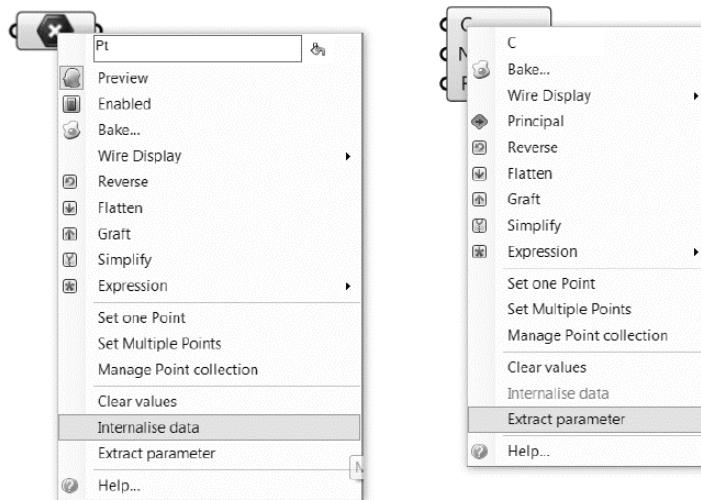
2. Referenced data

Data yang dimasukkan ke dalam komponen dengan mengacu/ me-refer dari sumber lain, misalnya dari Rhino atau dari file lain. Misalnya ketika anda menggunakan obyek Point di Rhino sebagai obyek referensi untuk komponen Point di GH. Jika obyek Point ini berubah posisinya di Rhino, maka informasi posisi ini akan otomatis ter-update di GH. Demikian juga sebaliknya. Jadi data referensi ini bersifat **dinamis**.

Namun demikian, karena data di GH disimpan terpisah dengan obyek referensinya di Rhino, maka ketika anda membuka definisi tersebut di GH, file Rhinoceros harus dibuka juga. Ini yang dinamakan *Volatile Data*.



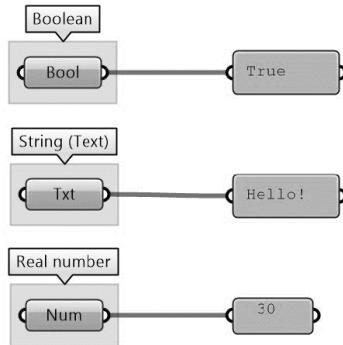
- **Volatile data** adalah data yang tersimpan dalam suatu komponen, yang berasal dari obyek di luar komponen tersebut, dimana jika obyek yang menjadi referensi itu hilang, maka data yang tersimpan di komponen juga akan hilang.
- Terlihat perbedaan dari dua data berupa Point pada Komponen Point di atas. Yang bawah adalah Volatile Data karena me-refer pada obyek Point di Rhinoceros. Untuk Volatile data, anda harus membuka file Rhino bersama file GH untuk memastikan data yang tersimpan dalam suatu komponen tidak hilang atau rusak.
- Anda dapat mengubah jenis data dari Volatile menjadi **Persistent** dengan jalan membuat data tersebut menjadi: **Internal (Internalized Data)** dan sebaliknya, jika ingin membuatnya volatile, gunakan: *Extract parameter*.



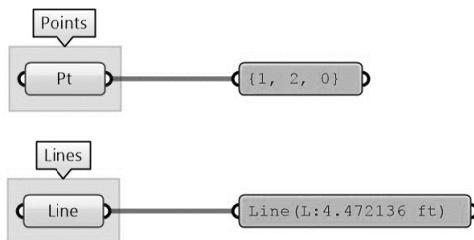
4.2. Jenis Data/ Data Types

Semua bahasa pemrograman mengidentifikasi jenis data yang digunakan untuk proses tertentu dan seperti halnya di semua bahasa pemrograman, semua fungsi atau operasi

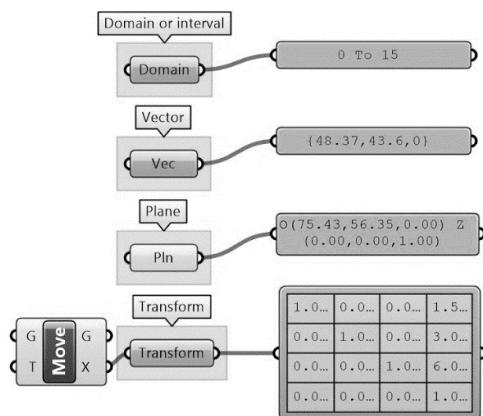
sangat sensitif terhadap jenis data. Beberapa jenis data yang umum adalah: **Integer**, **Number**, **Text**, **Bool** (True or False) dan lainnya. Semua jenis data di GH ada dalam **Params**→ **Primitives**.



GH juga mengenali jenis data berupa elemen geometri dalam konteks pemodelan 3D seperti: Point, Line, NURBS Curve, NURBS Surface, Brep dan lainnya. Semua jenis data geometri ada di **Params**→ **Geometry**.

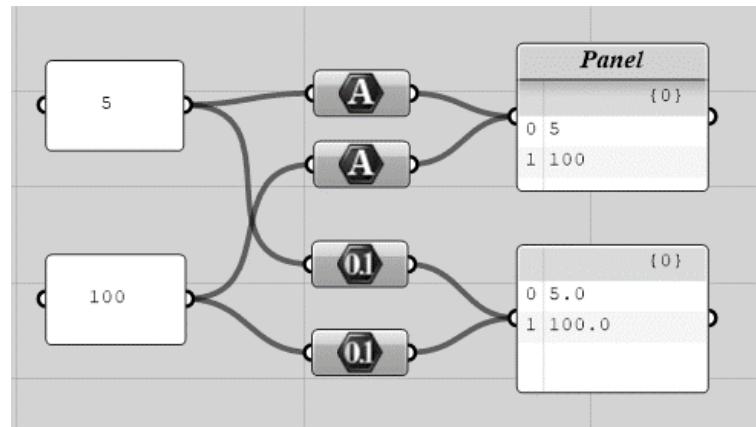


Selain dua jenis data itu, terdapat jenis data lain yang umumnya tidak digunakan di pemodelan 3D tapi sering digunakan dalam pemrograman parametrik: **Domains**, **Vectors**, **Planes**, **Transformation Matrices**.



Komponen di GH (misalnya **Panel**) dapat digunakan untuk melakukan konversi antara jenis data yang satu ke jenis data yang lain. Proses konversi ini dinamakan: **Casting**.

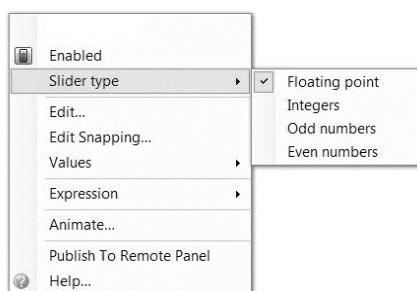
Misalnya, jika anda ingin mengganti suatu Text menjadi Number, anda masukkan data tersebut ke dalam komponen Number. Harap diingat, cara Casting ini tidak selalu berhasil dan selalu cek jenis data keluaran dari suatu komponen.



- Anda musti hati-hati. Angka dapat diidentifikasi/casting sebagai teks dalam komponen Text.

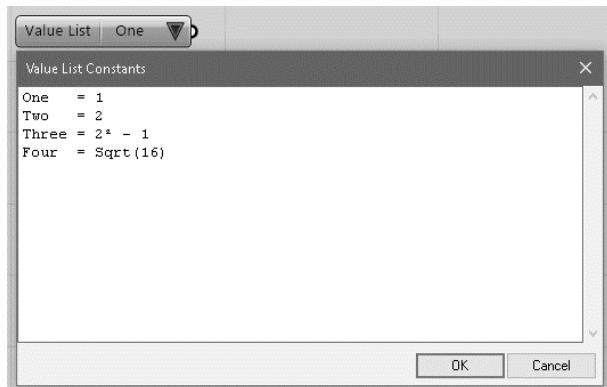
4.3. Number Generator

Ada beberapa jenis komponen yang menghasilkan angka (*Number Generator*) baik satuan ataupun dalam bentuk list dengan aturan tertentu. Komponen seperti: Panel, Number Slider, Number, Value List, menghasilkan satu angka dalam setiap eksekusi. Komponen seperti: Series, Graph Mapper, Range, Domain, menghasilkan beberapa angka dalam bentuk list dalam sekali eksekusi.

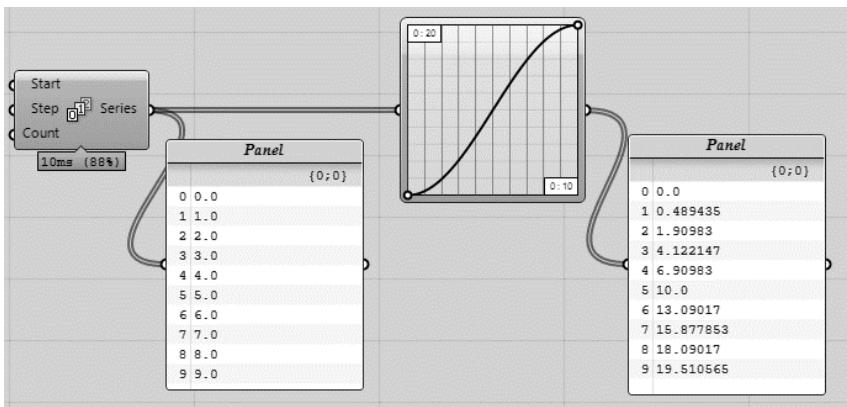


- Klik kanan pada judul slider untuk menentukan *setting*.
- Jenis angka yang dihasilkan ada empat jenis: R (Desimal), N(Bulat), E (Genap), O (Ganjil).
- Tentukan batas minimum dan batas maksimum nilai.

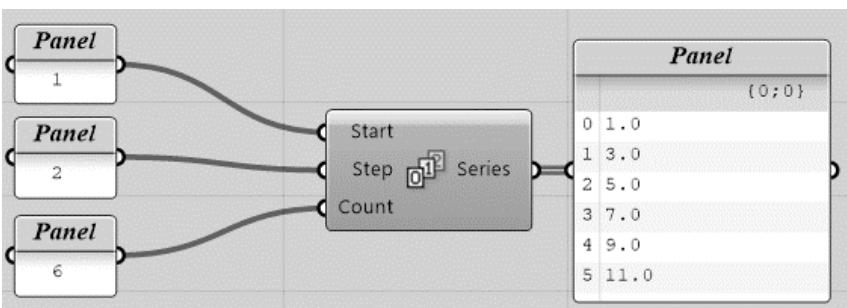
- **Value List** adalah interface yang mana input yang kita masukkan merupakan pilihan dari list yang kita tetapkan sendiri.



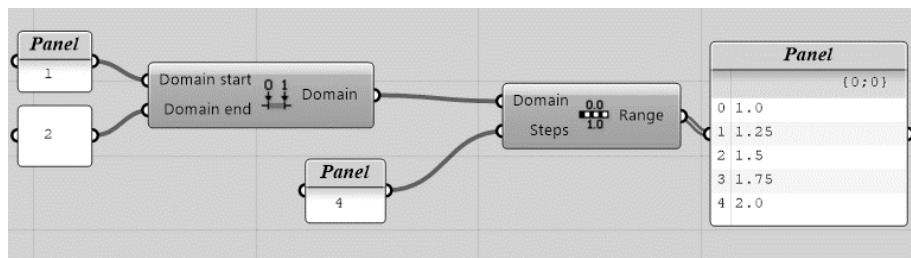
- Graph Mapper adalah interface 2D untuk memetakan nilai Y sesuai dengan grafik yang kita pilih-atau modifikasi, berdasarkan input X yang kita masukkan. Jadi keluaran dari Graph Mapper adalah nilai Y dan data berupa nilai X dimana nilai Y tersebut merupakan hasil dari fungsi yang tergambar pada grafik tersebut.



- Series adalah komponen yang menghasilkan beberapa angka sekaligus, dengan aturan: *Start* (*S*), *Step* (*N*) dan *Count* (*C*) jumlah total angka yang dihasilkan.



- Range adalah komponen yang menghasilkan beberapa angka sekaligus dengan aturan: *Domain* (batas bawah dan batas atas) dan *Step*: jumlah pembagian diantara batas domain tersebut.

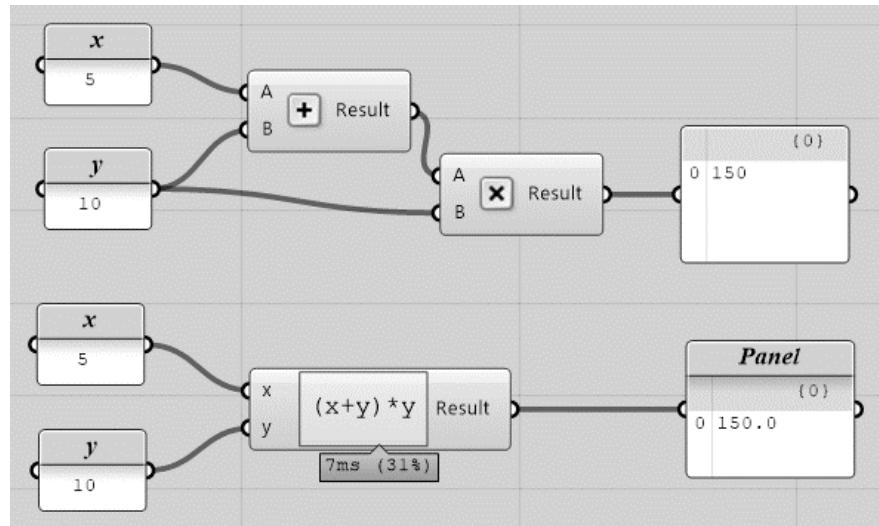


4.4. Pemrosesan Data

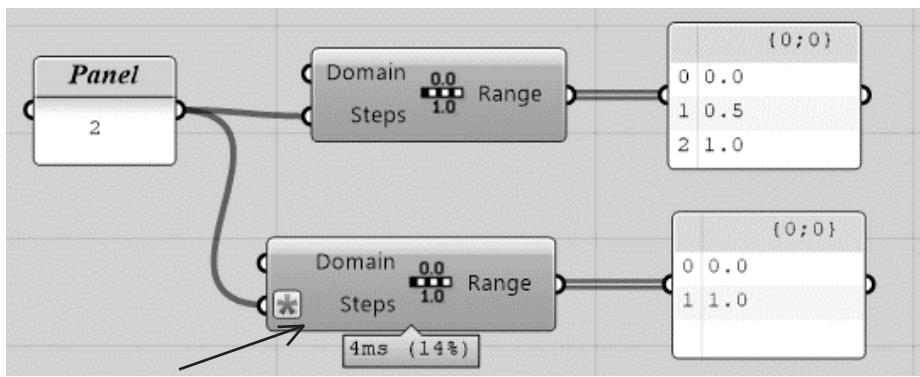
Ada beberapa jenis pemrosesan data yang sering digunakan dalam desain parametrik: numeric, logical, analysis, sorting dan selection.

4.4.1. Numeric Operation

- Termasuk di dalamnya: aritmatika, trigonometri, polynomial dan bilangan kompleks.
- Semua operasi matematika ada di bagian tab: **Math**
- Ada dua cara untuk melakukan operasi numerik: 1) dengan komponen operasi: **Addition**, **Multiplication**, **Subtraction**, **Division**, dan lainnya; 2) dengan menggunakan **Expression** dimana anda dapat melakukan serangkaian perhitungan numerik sekaligus dalam satu komponen.

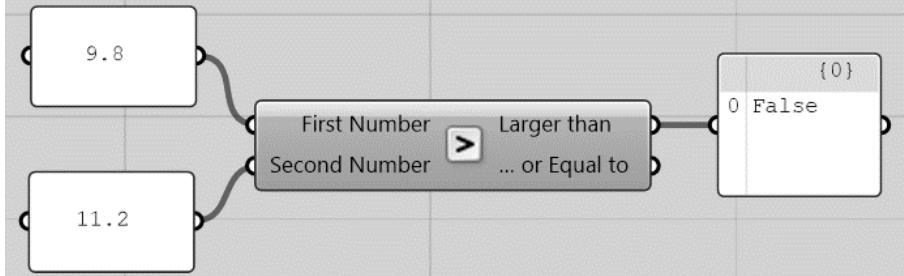


- Kedua jenis definisi di atas memiliki input dan output yang sama tapi prosesnya berbeda. Dengan menggunakan **Expression**, proses operasi lebih mudah dibaca dan lebih ringkas.
- Cara lain memasukkan **expression** adalah dengan memasukkannya secara internal yakni, dengan klik kanan pada input port suatu komponen dan pilih **Expression**.

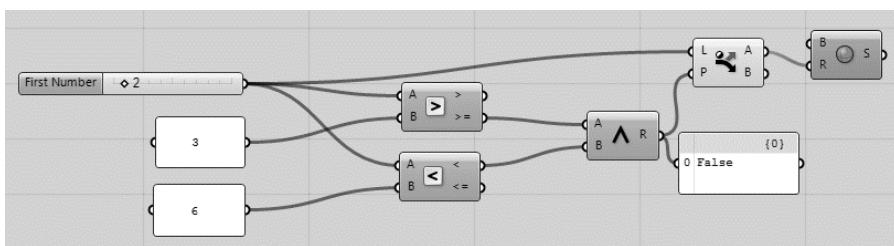


4.4.2. Logical Operation

Beberapa operasi logic yang ada di GH diantaranya: *equalities*, *set* dan operasi *logic*. Operasi logika ini umumnya digunakan untuk menyaring data atau mengatur aliran data.



Misalnya anda akan membuat sebuah bola yang radiusnya harus berada diantara dua angka/ dua nilai yang anda tentukan (misalnya, diantara 3 dan 6). Anda memerlukan gerbang logika (*logic gate*) untuk menyaring hanya nilai yang masuk dalam kategori yang anda tentukan yang dapat digunakan untuk membuat bola tersebut.



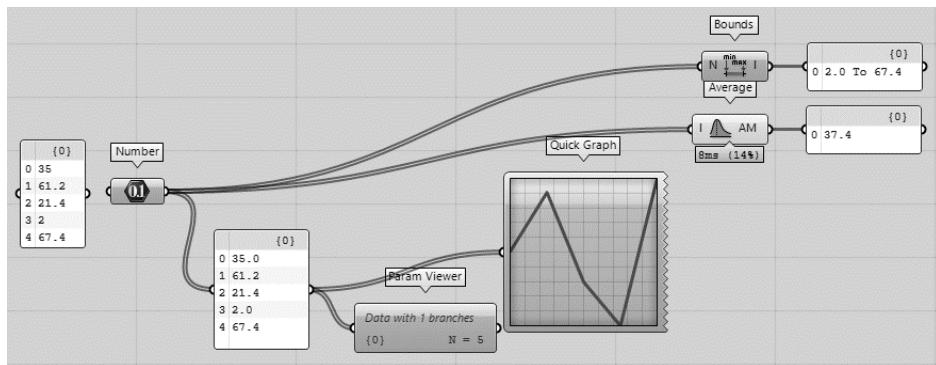
- Anda harus punya number generator sebagai pembanding, misalnya dengan **Number Slider**
- Gate **Larger Than** berisi nilai minimim (3) dan gate **Smaller Than** berisi nilai maksimum (6)
- Bandingkan input dari Number Slider dengan kedua gate tersebut.
- Gate **AND** digunakan untuk menyaring **HANYA** nilai yang memenuhi keduanya (artinya, <6 DAN >3) yang akan lolos atau bernilai **TRUE**.

- Komponen **Dispatch** adalah komponen terakhir yang memetakan nilai TRUE hasil dari gate AND dengan input awal. Ingat, semua Logic Gate hanya bernilai TRUE atau FALSE.

4.4.3. Data Analysis

Beberapa komponen digunakan untuk menampilkan data, struktur data, menganalisis data.

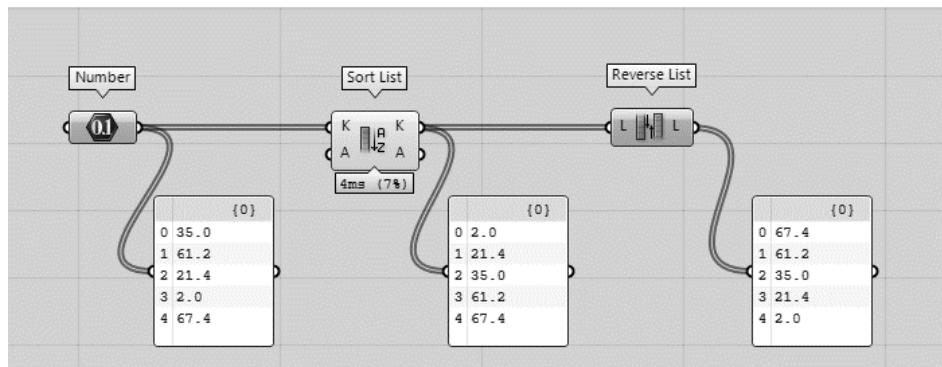
- **Panel:** input data, menampilkan isi komponen
- **Param Viewer:** menampilkan struktur data berupa diagram pohon maupun keterangan angka
- **Quick Graph:** menampilkan data dalam diagram Cartesian
- **Bound:** menampilkan nilai max dan min dari suatu data
- **Average:** menampilkan rata-rata dari suatu data.



4.4.4. Menyortir Data/ Sorting

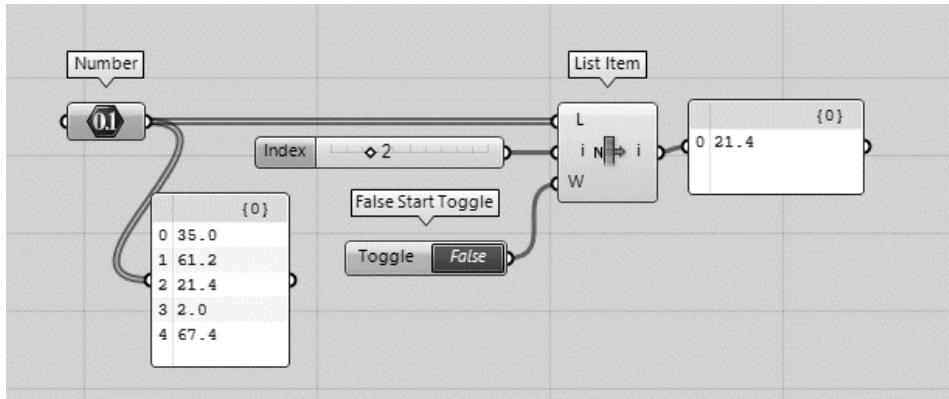
Beberapa komponen digunakan untuk menyortir data secara numerik dan secara geometrik.

- **Sort List:** menyortir data/ mengurutkan data dari mulai yang terkecil atau mulai yang terbesar.

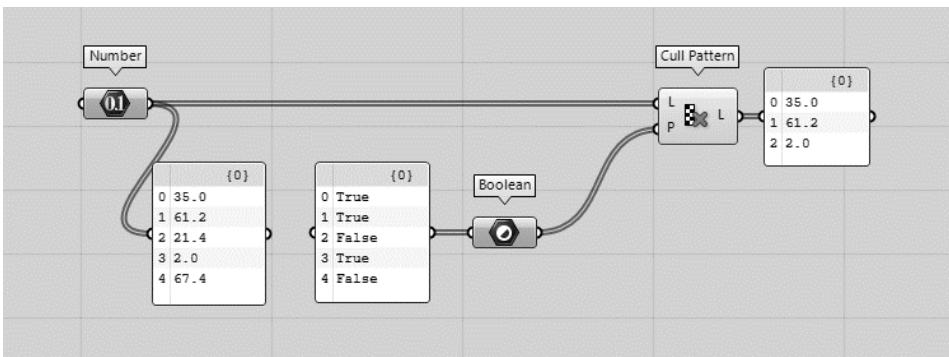


4.4.5. Memilih Data/ Data Selection

Proses di pemodelan 3D membuat kita dapat memilih dan mengedit obyek/ bagian dari obyek secara interaktif. Tidak demikian dengan pemrograman parametrik. Obyek/ bagian dari obyek dipilih berdasarkan bagian dari data yang menyimpan elemen, atau lokasi obyek bersangkutan. Setiap data yang disimpan dalam suatu komponen diatur menurut **struktur data**. Salah satu mekanisme untuk memilih data yang tersimpan dalam suatu komponen adalah melalui **nomor index-nya**. Setiap data yang tersimpan memiliki nomor indeks. Nomor indeks selalu dimulai dari 0 (Nol).



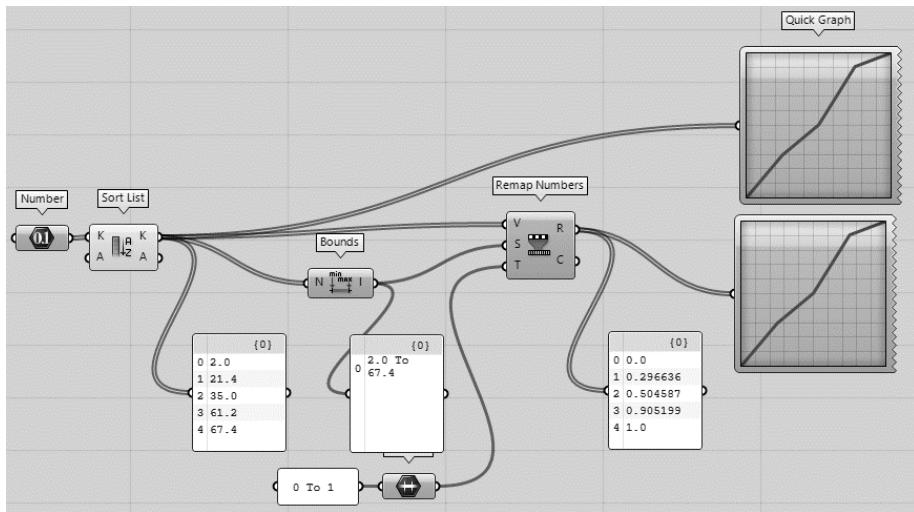
- **List Item:** Mengekstrak data menggunakan index number.



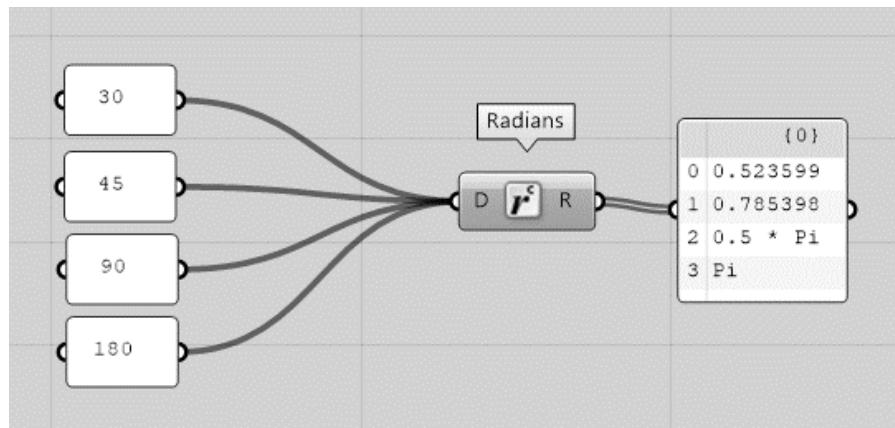
- **Cull Pattern:** Mengekstrak data menggunakan pola Boolean: *True*= data di-ekstrak, *False*=data tidak di-ekstrak.

4.4.6. Memetakan Data/ Data Mapping

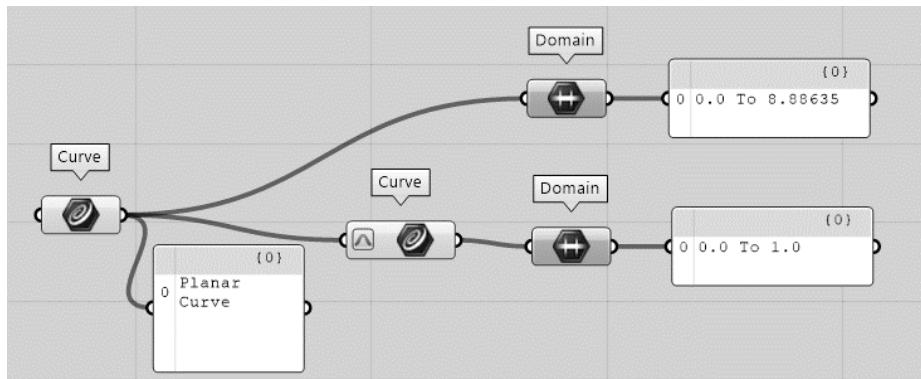
Dalam banyak kasus, kita akan memetakan secara linier sekumpulan angka dalam domain tertentu ke suatu domain yang lain. Anda dapat membayangkan sekumpulan angka sebagai suatu obyek yang akan di-skalakan. Komponen ini dinamakan **ReMap**. Komponen ReMap digunakan untuk menskalakan sekumpulan angka dalam suatu domain ke domain lain yang sesuai dengan algoritma yang kita buat.



- **ReMap:** memetakan satu set data berdasarkan domainnya (max-min) ke domain baru.



- **Radians:** memetakan angka berupa sudut desimal ke radian.



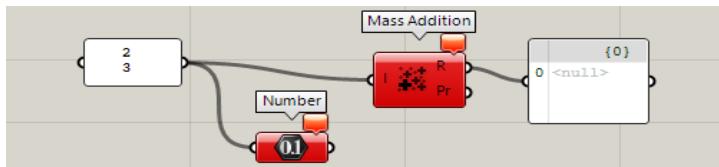
- **Reparameterize:** memetakan domain suatu data ke domain 0-1 ("0 To 1")

4.5. Kesalahan dalam Desain Algoritmik

Beberapa isu dan kesalahan dalam pembuatan definisi atau algoritma yang menyebabkan error, kesalahan output atau output yang tidak terduga.

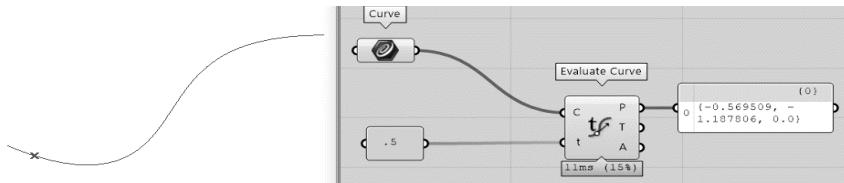
1. Kesalahan jenis input data.

Kesalahan jenis data yang dimasukkan dalam komponen menyebabkan error (warna merah pada komponen). Namun kadangkala ini tidak terdeteksi sehingga hasil eksekusi akhir tidak seperti yang diinginkan. Karena itu, selalu cek jenis data pada input dan jika perlu, selalu lakukan ‘casting’ sesuai dengan jenis data bersangkutan.

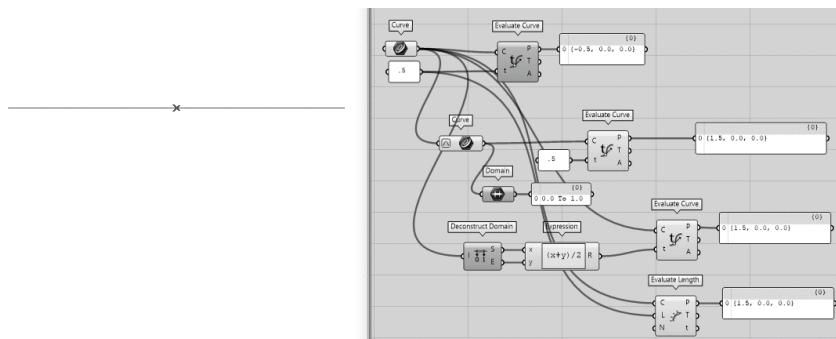


2. Kesalahan logika pada input data

Kesalahan jenis ini tidak menyebabkan komponen error, tetapi hasil eksekusi definisi menjadi salah.



- Definisi di atas salah dalam mengidentifikasi titik tengah ($t=0.5$) suatu kurva. Mengapa? Karena definisi mengasumsikan domain kurva adalah 0-1 yang sebenarnya arbitrerai (kita tidak tahu nilai min dan max kurva tersebut).
- Bagaimana memperbaikinya? Ada tiga cara:



- Menggunakan Reparametrize untuk remap domain menjadi 0-1.
- Menggunakan Deconstruct Domain untuk mengetahui domain kurva, dan menghitung titik tengah dengan Expression.

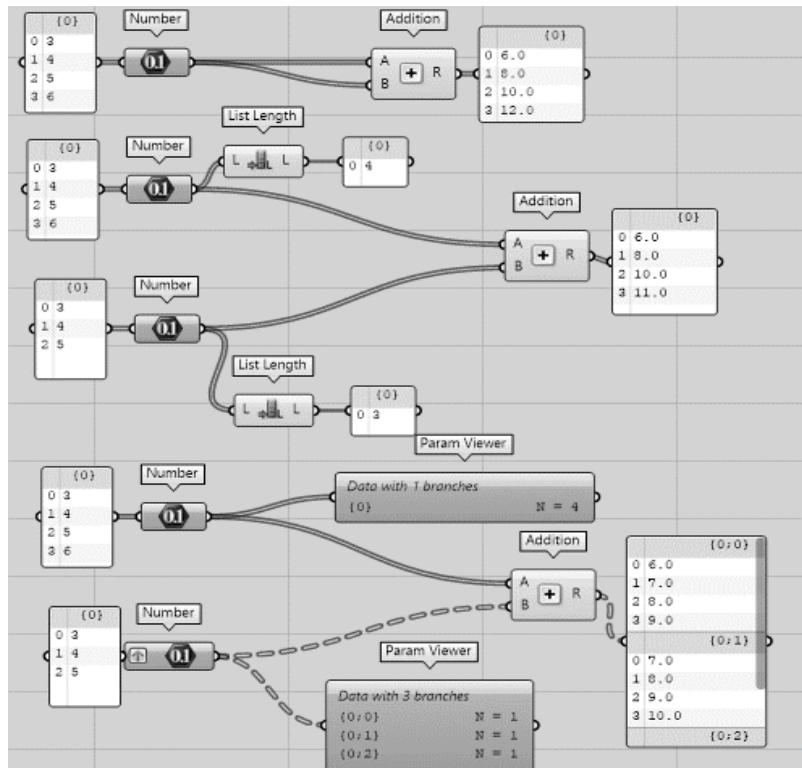
- Menggunakan **Evaluate Length** untuk menentukan titik tengah kurva (Ini akan menghasilkan hitungan yang berbeda dengan dua cara di atas jika obyek kurvalinier).

3. Kesalahan Urutan Proses Eksekusi dan Organisasi Komponen

Anda harus berlatih untuk mengorganisasikan komponen-komponen beserta kabel-kabel secara horizontal maupun vertikal agar terlihat jelas urutan (*sequence*) dari operasional definisi tersebut. Proses mencari apa yang salah pada program/definisi (*debugging*) akan lebih mudah dilakukan jika sebuah definisi lebih teratur dan tidak acak-acakan. Untuk perhitungan aritmatika yang kompleks, dapat gunakan komponen **Expression** alih-alih menggunakan beberapa komponen.

4. Kesalahan Struktur Data

Kesalahan ini pun umumnya tidak tampak pada komponen tetapi menyebabkan hasil eksekusi menjadi salah.

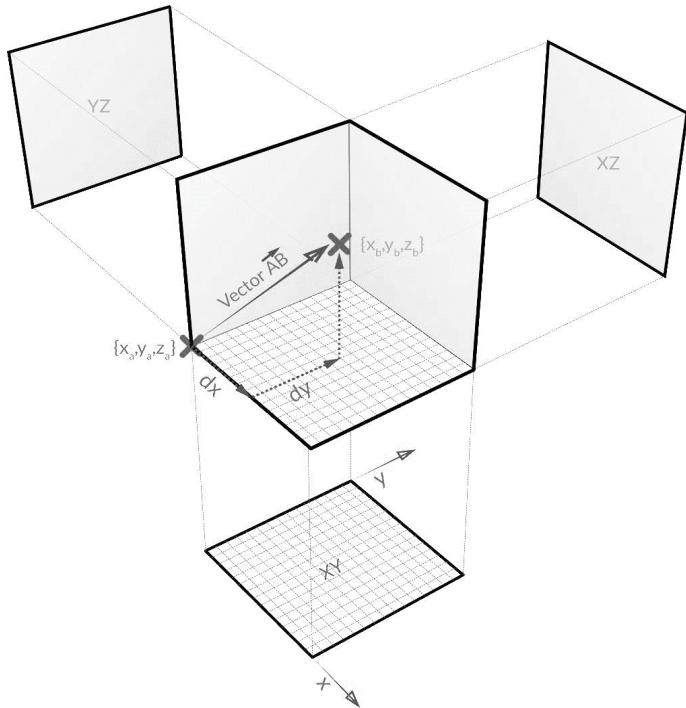


5. Pemrosesan yang Lama

Beberapa definisi atau algoritma akan memakan waktu cukup lama eksekusinya. Selalu berlatih untuk membuat algoritma yang efisien namun tetap mendapatkan hasil yang sesuai. Pada tahap-tahap latihan selalu gunakan data set yang minim sebelum menggunakan data set yang besar untuk yakin dengan algoritma yang dibuat.

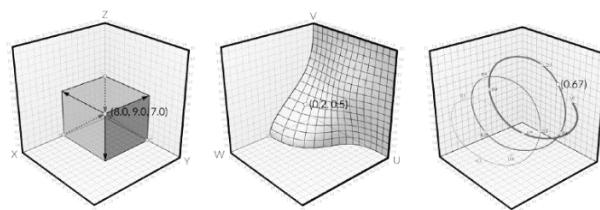
4.6. Points, Vector, Plane

Sebelumnya jenis data yang kita kenal adalah angka. Unit informasi terkecil dari data geometri adalah *Point* yang merupakan sebuah obyek yang hanya memiliki informasi koordinat. Nilai dari koordinat ini merupakan besaran-besaran yang akan digunakan untuk menentukan dimensi dan transformasi. *Point*, *Plane* dan *Vector* adalah dasar untuk membuat dan melakukan transformasi pada geometri di Grasshopper.



4.6.1. Point

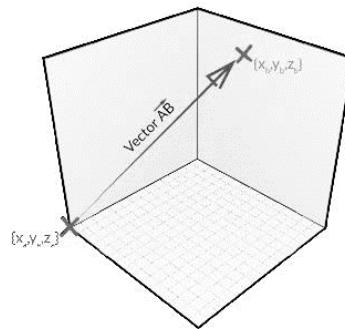
- Point pada 3D memiliki koordinat pada sumbu x,y,z
- Point pada 2D memiliki koordinat pada bidang yang ditentukan oleh dua sumbu: [x,y],[x,z],[y,z], atau [u,v].
- Point pada suatu *Surface*: dalam aturan GH, meskipun secara hipotesis ada tak terhingga Point pada suatu bidang (surface) namun akan selalu ada nilai maximum dan nilai minimum yang membatasi dimensi bidang tersebut. Jika ada suatu obyek Point yang berada di luar bidang tersebut, maka obyek Point ini menjadi invalid.
- Translasi bidang pada ruang: dalam GH, setiap bidang (surface) pasti berada pada suatu ruang [x,y,z] sehingga jika suatu Point didefinisikan pada koordinat bidang [x,y] atau [u,v], maka Point tersebut dapat ditranslasi pada koordinat ruang [x,y,z].



4.7. Vector

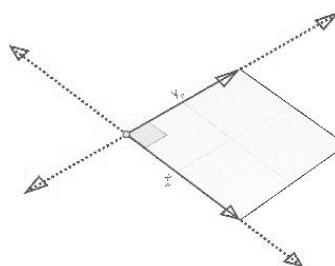
Vector adalah kuantitas geometri untuk menggambarkan arah dan besar pada suatu pergerakan. Vector ini abstrak, memiliki besar/ nilai tapi tidak memiliki bentuk geometri sehingga obyek Vector tidak bisa dibedakan dengan Point. Pada GH, anda mendefinisikan Vector sama dengan mendefinisikan Point, menggunakan koordinat [x,y,z].

Perbedaannya adalah, pada Point, posisi koordinat adalah absolut sedangkan koordinat yang sama, jika kita tentukan sebagai Vector merepresentasikan pergerakan dari titik origin [0,0,0]. Vector adalah panah pada ruang yang bergerak dari 0,0,0.

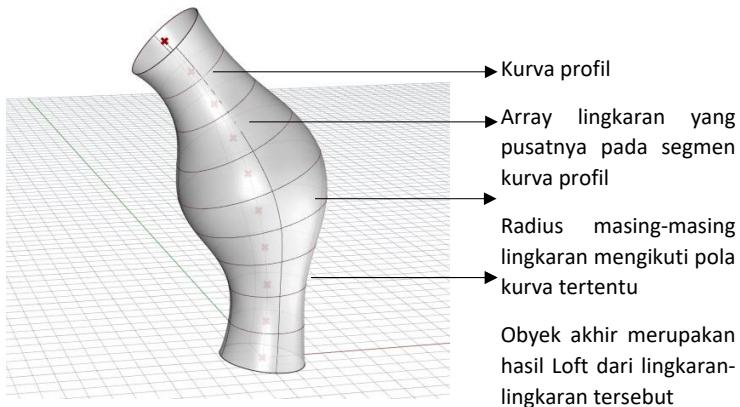


4.8. Plane

Plane adalah obyek bidang yang rata/flat yang tidak memiliki batas pada dua arah. Plane digunakan untuk mendefinisikan koordinat pada ruang 3D.

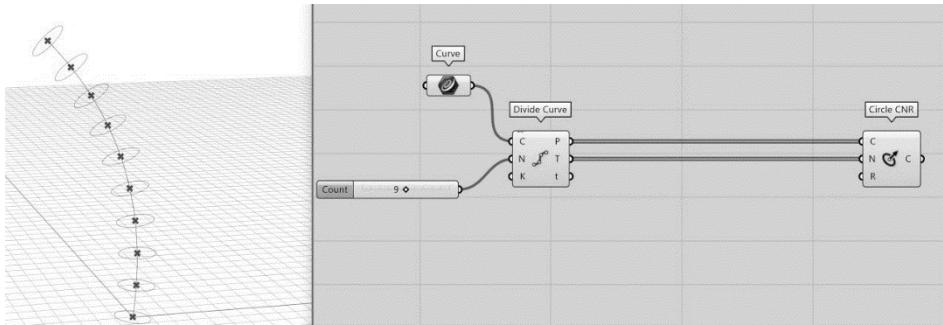


Latihan 6: Anatomi Definisi



Latihan 1

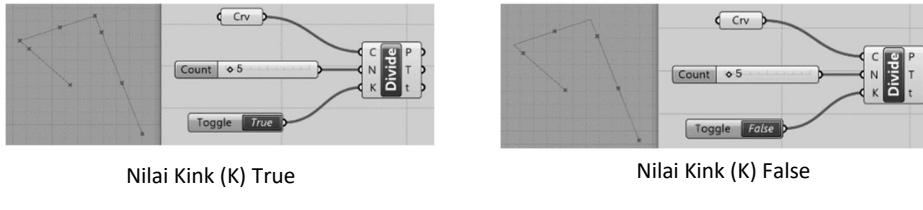
Langkah-langkah



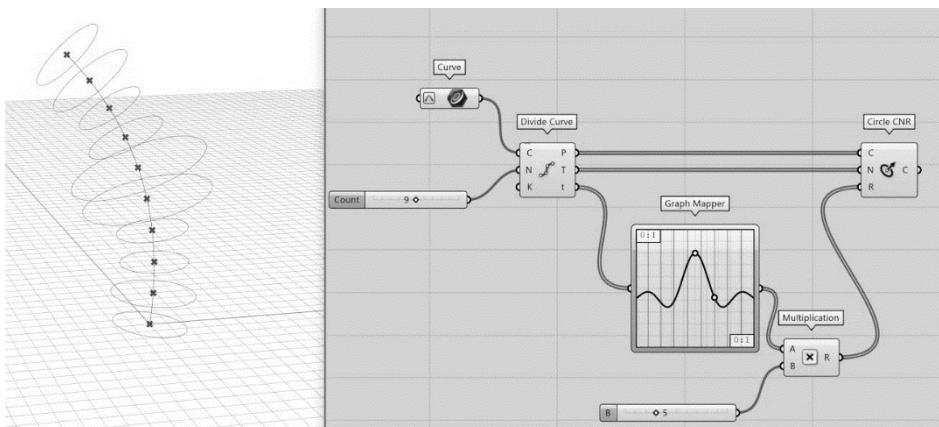
BUKU AJAR AR-4214 TAHUN 2021

- Buat sebuah **Control Point Curve** di RH, kemudian definisikan objek curve tersebut ke dalam GH dengan cara:
 - Buka GH, dan masukkan komponen **Curve** di GH dengan double-click daerah kanvas yang kosong dan ketik “Curve”, lalu klik container “Curve” yang muncul di pop up.
 - Klik kanan pada container “Curve” tersebut, kemudian pilih **Set One Curve**. Lalu klik objek curve yang sebelumnya dibuat di RH.
- Bagi (*Divide*) curve tersebut menjadi N Segmen menggunakan **Divide Curve**:
 - Masukkan komponen **Divide Curve**
 - Hubungkan container **Curve** sebelumnya ke komponen **Divide Curve** dengan cara click-drag dari port output container **Curve** menuju port input **Curve (C)** pada komponen **Divide Curve**.
 - Untuk nilai Segment (N), secara default bernilai 10, namun kita dapat memasukkan nilai berapapun. Buat sebuah **Number Slider** dengan cara double-click kanvas kemudian ketik angka yang diinginkan, misalnya “12”, kemudian **Enter**.

- Nilai *Kinks* (K) merupakan data Boolean (True/False). Bila bernilai True, maka komponen Divide Curve akan menganggap sudut sebagai salah satu titik segmen. Bila bernilai False, maka komponen Divide Curve akan tetap mengikuti prosedur default yaitu membagi curve dengan segmen yang sama panjang.



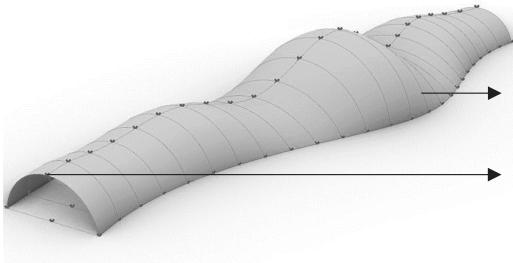
- Dalam Latihan ini, kita biarkan nilai Kink (K) sebagai default yaitu False. Tidak perlu memberi input pada port Kink (K). Gambar di bawah ini hanya untuk memberikan ilustrasi perbedaan nilai Kink yang memengaruhi hasil Divide Curve.
- Untuk membuat lingkaran pada titik-titik segmen tersebut, masukkan komponen Circle CNR, kemudian hubungkan port output Point (P) pada komponen Divide Curve menuju port input Center (C) pada komponen Circle CNR.
- Agar setiap lingkaran orientasinya tegak lurus/ normal dengan segmen, maka parameter Normal pada Circle CNR harus merupakan nilai tangensial dari setiap titik segmen (T).
- Selanjutnya, yang harus kita tentukan adalah variabel radius yang mengikuti pola tertentu. Kita akan pakai Graph Mapper untuk ini.
 - Masukkan komponen Graph Mapper
 - Hubungkan port output Parameter (t) pada Divide Curve ke port input Graph Mapper.
 - Kita akan memilih Grafik Sinus sebagai pola radius dari lingkaran-lingkaran. Klik Kanan pada Graph Mapper > Graph types > Sinc. Ubah bentuk grafik dengan men-drag titik putih yang ada di dalam Graph Mapper sesuai keinginan.



- Gunakan **Multiplication** sebagai amplifier parametrik menentukan besar radius yang berkorespondensi dengan pola hasil Graph Mapper
 - Masukkan komponen Multiplication

- Hubungkan port output Graph Mapper ke port input A pada Multiplication
- Masukkan number slider baru dan hubungkan ke port input B pada Multiplication
- Hubungkan port output Multiplication ke port input Radius (R) pada Circle CNR
- Profil kurva harus di-set Reparameterize karena nilai domain pada Graph Mapper adalah 0-1. Klik kanan di container *Curve > Reparameterize*
- Gunakan Loft untuk finalisasi hasil akhir.
 - Masukkan komponen Loft
 - Hubungkan port output *Curve (C)* pada Circle CNR ke port input pada Loft

Latihan 2



Dua kurva profil,
masing-masing dibagi
menjadi N segmen

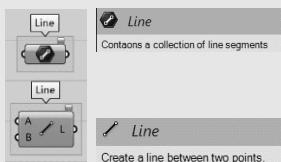
Tentukan titik tengah
dari jarak antara dua
titik segmen

Tentukan ketinggian
titik tengah tersebut
berdasarkan pola kurva
tertentu.

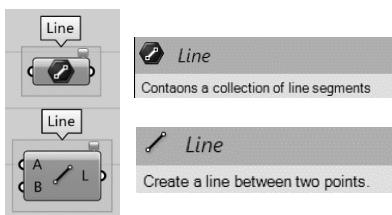
Langkah-langkah

Jangan Lupa !

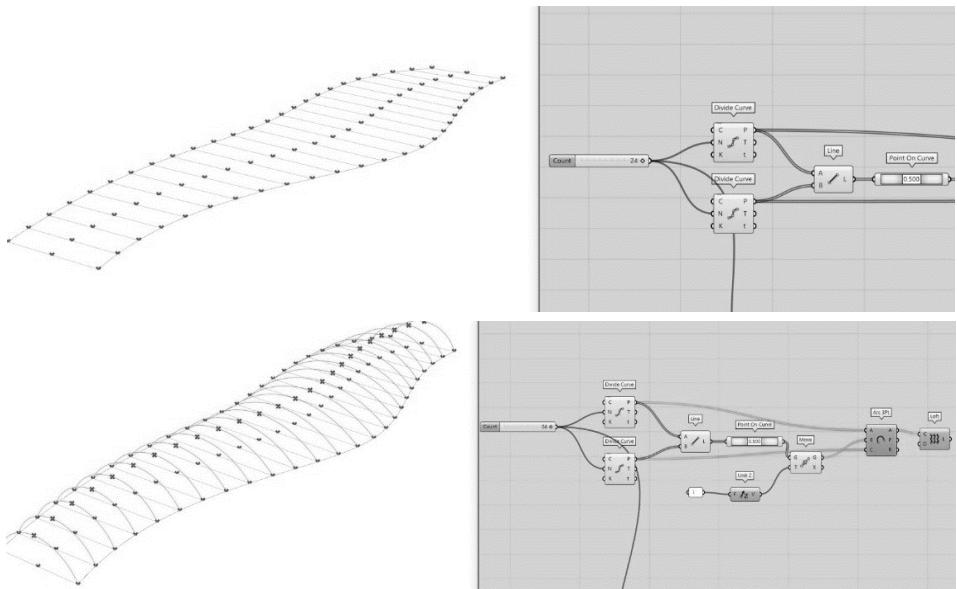
Beberapa komponen GH memiliki nama yang sama namun fungsi yang berbeda. Pastikan kita memasukkan komponen yang benar dengan melihat simbol dan/atau membaca keterangan dari pop-up yang muncul



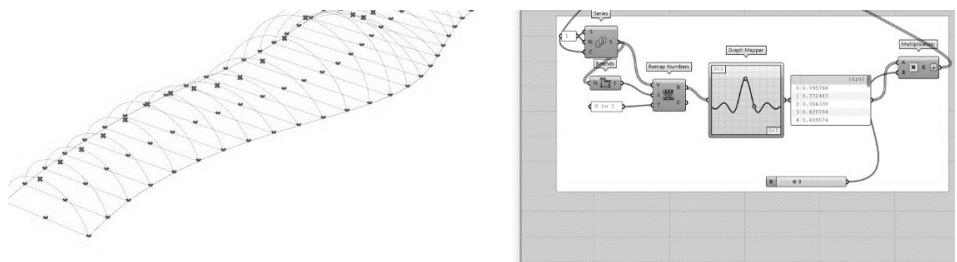
Buat dua buah **Control Point Curve** di Rhino, masing-masing dibagi menjadi N Segmen menggunakan **Divide Curve** (ikuti panduan pada Latihan 1 untuk melakukan ini)



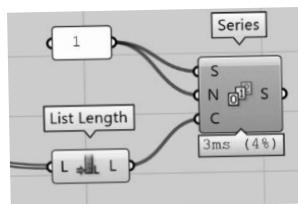
- Buat Line yang menghubungkan masing-masing titik segmen
 - Masukkan komponen Line
 - Hubungkan port output dari masing-masing Divide Curve menuju port input A dan port input B (seperti gambar)
- Tentukan titik tengah setiap garis dengan **Point on Curve (0.5)**. Masukkan komponen **Point on Curve** dan hubungkan port output *Line* menuju port input *Point on Curve*



- Pindahkan titik tengah ke arah Z:
 - Masukkan komponen Move.
 - Hubungkan port output Point on Curve ke port input Geometry (G) pada Move.
 - Untuk parameter Motion (T) pada Move, kita perlu memasukkan data Vector ke arah sumbu Z. Caranya, masukkan komponen Unit Z. Lalu, hubungkan number slider ke port input Unit Z. Kemudian, hubungkan port output Unit Z ke port input Motion (T) pada Move (seperti gambar 10)
- Buat Arc 3Pt agar menghasilkan lengkungan Arc yang dibentuk oleh tiga titik: titik pada masing-masing kurva dan titik tengah yang dipindahkan ke atas.
 - Masukkan komponen Arc 3pt
- Perhatikan bahwa Arc 3pt memiliki tiga input yaitu A, B dan C. Input A dan C menentukan ujung-ujung dari arkus sedangkan input B menentukan titik tengah dari arkus. Dengan begitu, untuk input A dan C kita masukkan output dari Divide Curve, sedangkan untuk input B kita masukkan output dari Move.
- Bagaimana jika ketinggian (Z) tidak sama? Tetapi mengikuti pola kurva tertentu? Yang diubah adalah nilai Factor Z: alih-alih satu angka, kita akan gunakan deretan angka menggunakan Graph Mapper.



- generasi angka dengan Series dengan jumlah total angka sejumlah segmen.



- Masukkan komponen Series
- Untuk parameter Start (S) dan (Step) kita masukkan nilai "1" dengan panel.
- Untuk parameter Count (C) kita perlu memasukkan jumlah arkus yang akan dibuat di Arc3pt. Untuk itu, masukkan komponen List Length yang berfungsi menghitung panjang sebuah list. Lalu, hubungkan salah satu output Point (P) pada Divide Curve menuju input List Length. Kemudian, hubungkan output List Length ke input Count (C) dari Series.
- Output dari Series adalah berupa nilai "1, 2, 3.. dst". Sedangkan, yang kita inginkan adalah rentang angka 0-1. Untuk itu, Tentukan domain angka tersebut dengan Bound dan gunakan Remap agar rentang angka berada pada 0-1
 - Masukkan komponen Bound dan Remap
 - Hubungkan output Series (S) ke input pada Bound dan input Value (V) pada Remap.
 - Hubungkan output Bound ke input Source (S) pada Remap, ini artinya kita memberikan domain asal berupa rentang data awal (1,2,3...dst)
 - Kita ingin membuat rentang 0-1 sebagai domain tujuan, untuk itu double click kanvas, dan ketik "0 to 1" kemudian Enter, untuk membuat sebuah Panel berisi nilai rentang 0-1. Kemudian hubungkan Panel tersebut ke input Target (T) pada Remap.
- Gunakan Graph Mapper dan Multiplication untuk amplifikasi parameter ketinggian kurva.
- Hubungkan hasil ini ke Factor Z untuk posisi titik-titik tengah.
- Gunakan Loft untuk membuat Surface final

