GRASSHOPPER

PRIMER

FOR VERSION 0.6.0007 - SPANISH EDITION

BY ANDREW PAYNE & RAJAA ISSA

Traducido al español en colaboración con:
Francisco Calvo - Katherine Caceres
Miembros del grupo DUM-DUM
- www.tectonicasdigitales.com -

Manual de Grasshopper, Segunda Edición – Para versión 0.6.0007

Introducción

Bienvenido al maravilloso nuevo mundo de Grasshopper. Esta es la segunda edición de este manual y no hubiese sido posible sin la tremenda contribución de Rajaa Issa. Rajaa es un desarrollador de Robert McNeel & Associados y es el autor de varios otros plugins de Rhino incluyendo ArchCut y el siempre popular PanelingTools. Esta revisión proporciona una guía mucho más amplia que la primera edición, con más de 70 nuevas páginas dedicadas a crear tus propios componentes personalizados de secuencias de comandos.

El lanzamiento de este manual coincide con dos acontecimientos: la primera es la nueva versión de Grasshopper 0.6.0007 que resulta ser una gigante actualización de la ya sólida plataforma de Grasshopper. Los usuarios existentes se encontraran con cambios sutiles, y algunos cambios no tan sutiles de cómo se almacenan los datos en la versión actual, haciendo algunas de las definiciones anteriores obsoletas o incluso inutilizables. La esperanza de este manual es proporcionar ayuda a los usuarios nuevos y existentes en la navegación dentro de muchos de los cambios realizados en el sistema del software. El segundo evento que se superpone con esta publicación es la conferencia titulada “FLUX: Architecture in a Parametric Landscape”, que se celebra en la Facultad de Artes de California. El evento explora la arquitectura contemporánea y el diseño a través de su relación con los cambios en las tecnologías de diseño tales como el modelado paramétrico, la fabricación digital y las secuencias de comandos. Entre otras cosas, el evento constará de una serie de exposiciones y workshops dedicados a los sistemas de softwares paramétricos. Me siento honrado de estar enseñando una introducción al modelado en Grasshopper, mientras que Rajaa Issa y Gil Akos estarán manejando los workshops de modelado avanzado en Grasshopper y VB.Net.

Hemos incluido una gran cantidad de información nueva en este manual, y esperamos que siga siendo un gran recurso para aquellos que quieran aprender más sobre el plugin. Sin embargo, uno de los mayores activos que este software tiene a su favor eres tú, el usuario, ya que cuando más gente empieza a explorar y comprender el modelado paramétrico, esto ayuda a todo el grupo. Quisiera animar a cada persona que lea este manual para unirse a la línea de crecimiento de la comunidad y enviar sus preguntas al foro, ya que casi siempre hay alguien dispuesto a compartir una solución a su problema. Para obtener más información, visite:
http://www.grashopper.rhino3d.com.

Gracias y buena suerte!

Andrew Payne
LIFT architects
www.liftarchitects.com

Rajaa Issa
Robert McNeel and Associates
http://www.rhino3d.com/

La traducción de este manual de Grasshopper al español fue realizada gracias a la colaboración de: Francisco Calvo y Katherine Cáceres (miembros del grupo DUM-DUM).

Para mayor información sobre este manual y otros tutoriales, visite:

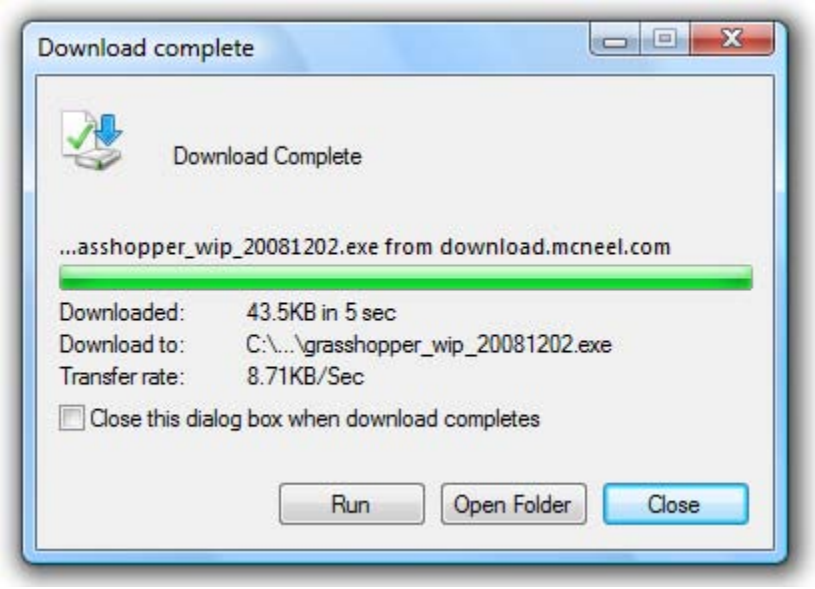
www.tectonicasdigitales.com

Tabla de contenidos

1 *Empezando*

Instalación de Grasshopper

Para descargar el plugin Grasshopper, visite **http://grasshopper.rhino3d.com/**. Haga clic en el vínculo de **descarga** en la esquina superior izquierda de la página, y cuando se le solicite en la siguiente pantalla, introduzca su dirección de correo electrónico. Ahora, haga clic derecho sobre el enlace de descarga y seleccione **"Guardar destino"** desde el menú. Seleccione una ubicación en su disco duro (nota: el archivo no se puede cargar en más de una conexión de red, por lo que el archivo debe estar guardado localmente en el disco duro del ordenador) y guardar el archivo ejecutable en esa dirección.

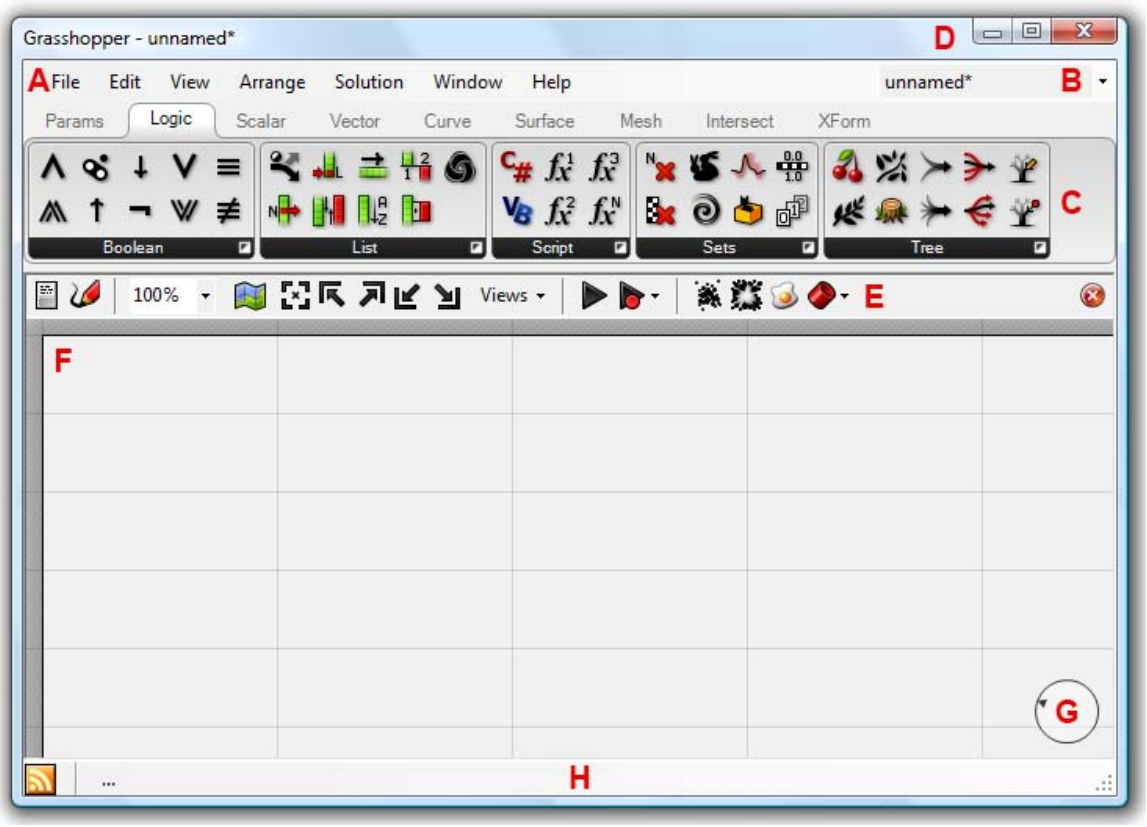


Seleccione **Ejecutar** en el cuadro de diálogo de descarga, siga las instrucciones del instalador. (nota: debe tener SR4b con Rhino 4.0 o superior ya instalado en su computadora para instalar correctamente el plugin)

2 *La interface**

El diálogo principal

Una vez que haya cargado el plugin, escriba "Grasshopper" en el comando de Rhino para mostrar la ventana principal de “Grasshopper”.



A. La barra de menú principal

El menú es similar a los típicos menús de Windows, excepto por el explorador de archivos en la derecha, B. A través de este cuadro desplegable, se puede cambiar rápidamente entre diferentes archivos cargados, mediante su selección. Tenga cuidado al usar los atajos, ya que son manejados por la ventana activa. Ésta bien podría ser Rhino, el plugin Grasshopper o cualquier otra ventana dentro de Rhino. Dado que no se puede deshacer debe ser cauteloso con los atajos Ctrl-X, Ctrl-S y Del.

B. Control explorador de archivo

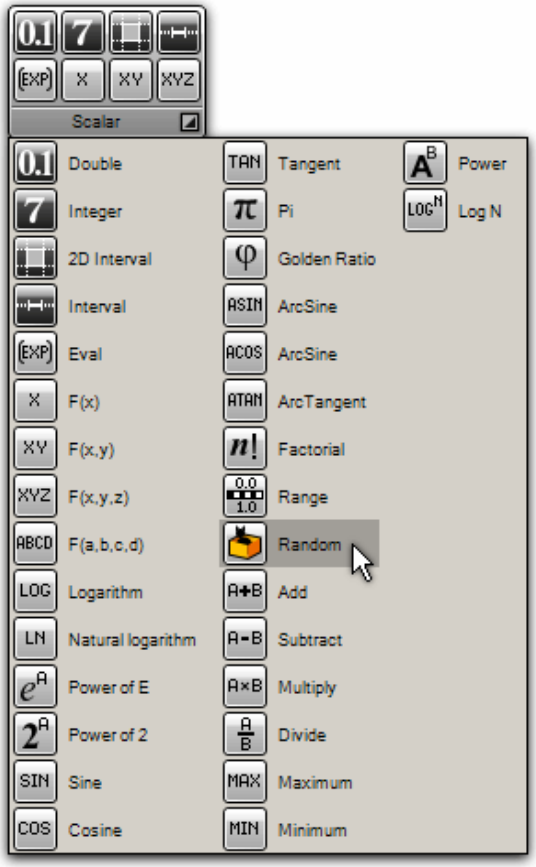
Como se discutió en la sección anterior, este menú desplegable se puede utilizar para cambiar entre los diferentes archivos cargados.

* Fuente: RhinoWiki
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryPluginInterfaceExplained.html

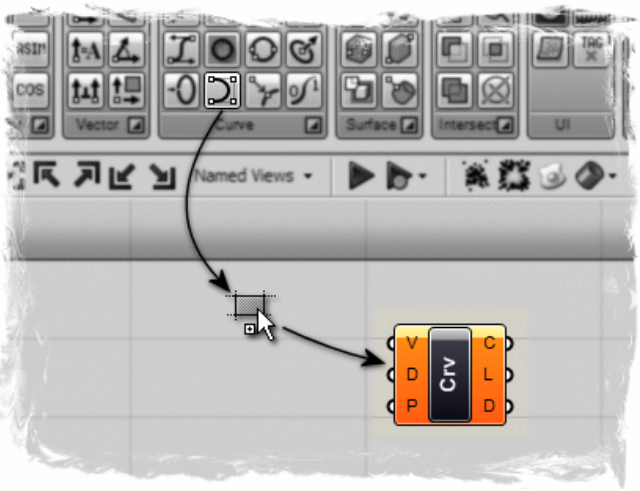
C. Paneles de Componentes

Esta área expone todas las categorías de los componentes. Todos los componentes pertenecen a una determinada categoría (como "Parámetros" para todos los tipos de datos primitivos o como "curvas" para todos los relacionados con las herramientas de la curva) y todas las categorías están disponibles en los paneles de la barra de herramientas. La altura y el ancho de las barras de herramientas pueden ser ajustados, lo que permite más o menos botones en la pantalla por cada categoría.

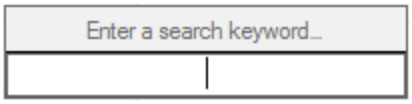
Los paneles de la barra de herramientas contienen todos los componentes que pertenecen a esa categoría. Puesto que hay un número potencialmente elevado de estos, sólo se muestra la más reciente de N artículos usados. Para ver toda la colección, tiene que hacer clic en la barra en la parte inferior del panel:



Este panel de categorías que aparece, proporciona acceso a todos los objetos. Puede hacer clic en cualquiera de los objetos de la lista desplegable o puede arrastrarlos directamente desde la lista al lienzo. Al hacer clic en los elementos del panel de categorías estos se posicionaran en la barra de herramientas para una fácil referencia futura. **Al hacer clic en los botones, los objetos no se sumaran lienzo!** Para añadirlos usted debe arrastrarlos en orden hacia el lienzo:



También puede encontrar los componentes por su nombre, haciendo doble clic en cualquier lugar del lienzo; aparecerá un cuadro de búsqueda. Escriba el nombre del componente que está buscando y verá una lista de parámetros o componentes que coinciden con su petición.



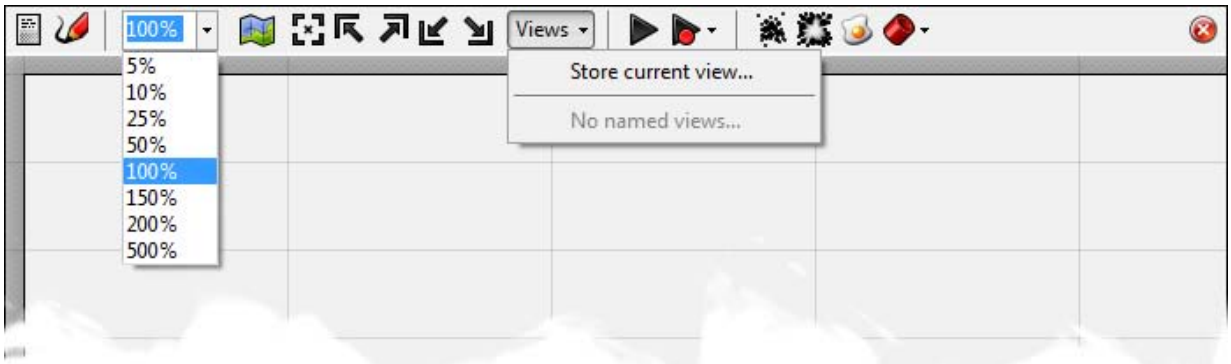
La ventana de la barra de título: D

La ventana de edición de la barra de título se comporta de forma diferente a la mayoría de los cuadros de diálogo en Microsoft Windows. Si la ventana no está minimizada o maximizada, haga doble clic en la barra de título y se plegará o desplegará el cuadro de diálogo.

Esta es una excelente manera de cambiar entre el plugin y Rhino porque reduce al mínimo el editor sin moverlo a la parte inferior de la pantalla o detrás de otras ventanas. Tenga en cuenta que si cierra el editor, las geometrías de Grasshopper en las vistas previas van a desaparecer, pero los archivos en realidad no se cerrarán. La próxima vez que ejecute el comando _Grasshopper, la ventana vuelve en el mismo estado con la mismos archivos cargados.

La barra de herramientas del lienzo: E

La barra de herramientas del lienzo proporciona acceso rápido a una serie de funciones de uso frecuente. Todas las herramientas están disponibles a través del menú, así, puede ocultar la barra de herramientas si lo desea. (Se puede volver a habilitar en el menú, Vista).



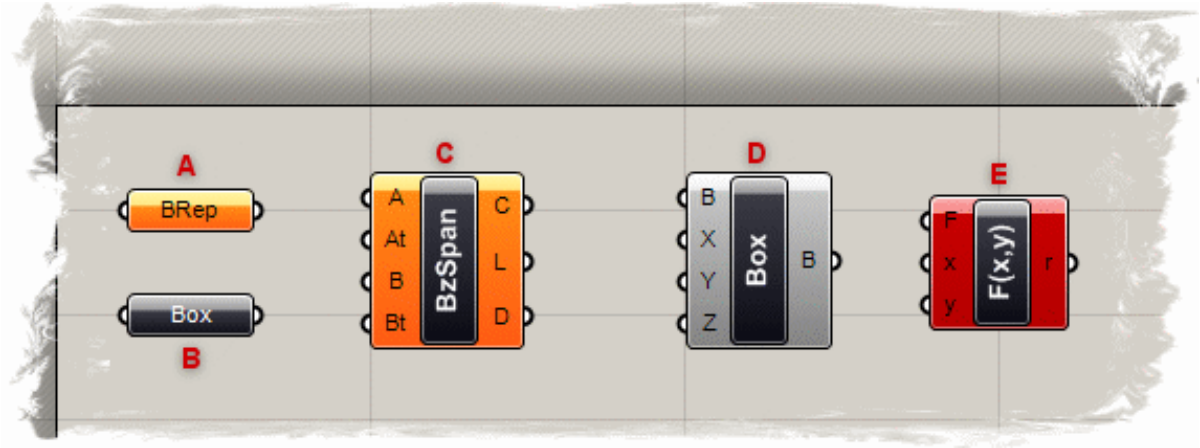
La barra de herramientas del lienzo expone las siguientes herramientas (de izquierda a derecha):

1. Editor de definición de propiedades.
2. Herramienta de bosquejo:
   La herramienta de dibujo funciona como la mayoría de las de herramientas tipo-lápiz de Photoshop o de Paint de Window. Los controles por defecto de las herramientas de dibujo permiten cambios de grosor de línea, tipo de línea y el color. Sin embargo, puede ser bastante difícil trazar líneas rectas o formas pre-definidas. Con el fin de resolver este problema, arrastra cualquier línea sobre el lienzo. Haga clic derecho sobre la línea, y seleccione "Cargar de Rhino", y seleccione cualquier forma pre-definida en la escena Rhino (Esto puede ser cualquier forma 2D como un rectángulo, círculo, estrellas... etc.). Una vez que haya seleccionado la forma de su referencia, pulse Enter, y su línea previamente elaborada se va a reconfigurar a su forma de referencia de Rhino.
3. Zoom por defecto
4. Mapa de navegación abre una pequeña ventana diagramática del lienzo que permite moverse rápidamente alrededor del lienzo sin tener que desplazarse y panear. Esta característica es similar a la ventana del Navegador en Photoshop.
5. Zoom Extensión (ajusta el factor del zoom si la definición es demasiado grande para caber en la pantalla)
6. Las esquinas de enfoque (estos 4 botones se centraran en las 4 esquinas de la definición)
7. Puntos de vista con nombre (muestra un menú para almacenar y recuperar puntos de vista con nombre)
8. Volver a generar solución (fuerza una reconstrucción completa de la definición de la historia)
9. Reconstruir los eventos (por defecto, Grasshopper responde a los cambios en Rhino y en el lienzo. Aunque este menú, puede desactivar estas respuestas)

10. Agrupador (convierte todos los objetos seleccionados en un grupo de objetos) **Los objetos agrupados aún no están terminados y es posible volver a modificarlos completamente en el futuro. Tenga cuidado al usar esto en los archivos actuales.**
11. Desagrupador (convierte todos los grupos seleccionados en objetos sueltos) **Los objetos agrupados aún no están terminados y es posible volver a modificarlos completamente en el futuro. Tenga cuidado al usar esto en los archivos actuales.**
12. Herramienta cocinar (convierte a todos los componentes seleccionados en objetos reales de Rhino).
13. Configuración previa (la geometría de Grasshopper se previsualiza por defecto. Puede deshabilitar la vista previa para cada objeto base, pero también se puede anular la vista previa para todos los objetos. La desconexión de la vista previa de sombreado acelerará algunas escenas que tienen curvas o superficies recortadas.
14. Ocultar botón. Esto oculta el botón de la barra de herramientas del lienzo, puede volver a activarlo a través del menú Vista.

F: El lienzo

Este es el editor actual donde se define y edita la red de historia. El lienzo alberga tanto a los objetos que componen la definición y algunos dispositivos de interfaz de usuario **G**. Los objetos en el lienzo usan generalmente un código de color para proporcionar información sobre su estado:



A) Parámetro. Un parámetro que contiene advertencias se muestra como un cuadro de color naranja. La mayoría de los parámetros son de color naranja cuando los añades sobre el lienzo, ya que la falta de datos es considerado como una advertencia.

B) Parámetro. Un parámetro que no contiene ni advertencias ni errores.

C) Componente. Un componente es siempre un objeto de mayor participación, ya que contiene parámetros de entrada y de salida. Este componente particular, tiene por lo menos una alerta relacionada con él. Usted puede encontrar alertas y errores a través del menú contextual de los objetos.

D) Componente. Un componente que no contiene ni advertencias ni errores.

E) Componente. Un componente que contiene al menos un error. El error puede venir bien del componente en sí mismo o de uno de sus parámetros de entrada y salida. Vamos a aprender más acerca de las estructuras de los componentes en los capítulos siguientes.

Todos los objetos seleccionados se dibujan con una cubierta verde (no mostrado).

G: Dispositivos de interfaz de usuario

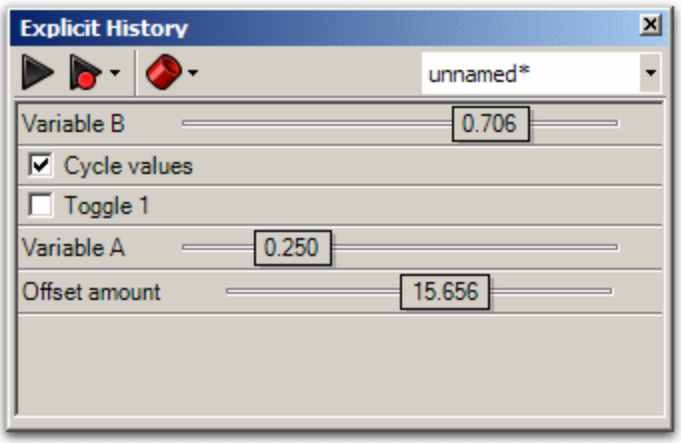
Actualmente, en único dispositivo de interfaz de usuario disponible es la brújula, que se muestra en la esquina inferior derecha del lienzo. El dispositivo de brújula ofrece una navegación gráfica que muestra su vista actual en relación con las extensiones de la definición completa. Este dispositivo puede ser activado/desactivado a través del menú Vista.

H: La barra de estado

La barra de estado proporciona información sobre el conjunto seleccionado y los principales acontecimientos que han ocurrió en el plugin. La información clave acerca de si tiene o no algún error o advertencia en su definición, se mostrará aquí.
El icono cuadrado naranjo en la parte inferior izquierda de la barra de estado, es un lector de RSS en vivo del Foro de Grasshopper. Al hacer clic en este icono, una lista de los temas más recientes, vinculados al sitio web del grupo de usuarios de Grasshopper se mostrará. Seleccionando cualquiera de los temas, lo llevará directamente a la discusión publicada por uno de los miembros del grupo de usuarios. Usted puede visitar el sitio web del Grupo de Usuarios de Grasshopper en:
http://grasshopper.rhino3d.com/

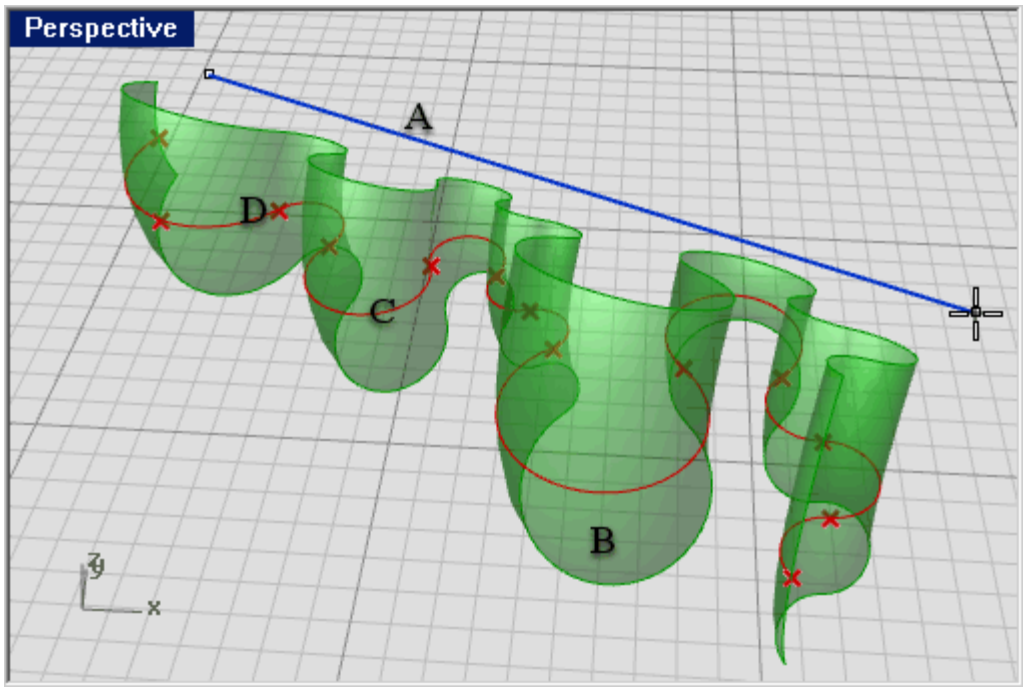
El panel de control remoto:

La ventana de Grasshopper es bastante grande, puede que no la desee sobre la pantalla todo el tiempo. Por supuesto, usted la puede minimizar o colapsar, pero entonces usted no podrá modificar más los valores. Si desea una interfaz minimizada para los valores dentro de la actual definición activada, puede habilitar el panel remoto. Se trata de un cuadro de diálogo que realiza un seguimiento de todos los reguladores e interruptores booleanos (y posiblemente otros valores, así como en futuras versiones):



El panel remoto también proporciona una vista previa básica, de los controles de eventos y archivos de conmutación. Puede habilitar el panel a través del menú Vista de la ventana principal, o a través de la Comando _GrasshopperPanel.

Vista previa:



A) La geometría **Azul** significa que está tomado actualmente con el ratón.

B) La geometría **Verde** en la vista corresponde a un componente que está seleccionado actualmente.

C) La geometría **Roja** en la vista corresponde a un componente que no se encuentra actualmente seleccionado.

D) **La geometría punto** se dibuja como una cruz en vez de un rectángulo para distinguirlo de los puntos de Rhino.

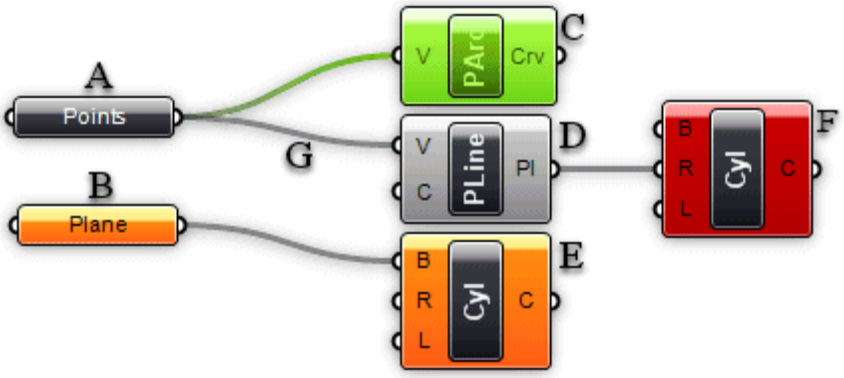
3 *Objetos de Grasshopper**

Definición de objetos de Grasshopper

Una definición de Grasshopper puede consistir en muchos tipos de objetos diferentes, pero con el fin de empezar, sólo es necesario familiarizarse con dos de ellos:

- Parámetros
- Componentes

Los parámetros contienen datos, lo que significa que **almacenan** las cosas. Los componentes contienen acciones, lo que significa que **hacen** cosas. La siguiente imagen muestra algunos de los objetos que probablemente encuentre en una definición de Grasshopper:



A) Un parámetro que contiene datos. Dado que no hay cables que salgan del lado izquierdo del objeto, este no hereda sus datos de otros lugares. Los parámetros que no contienen errores o advertencias son bloques negros con texto horizontal.

B) Un parámetro que no contiene datos. Cualquier objeto que no recoge los datos es considerado sospechoso en una definición explícita de la historia, ya que al parecer esta perdiendo el tiempo y el dinero de todos. Por lo tanto, todos los parámetros (cuando son recién añadido) son de color naranja, para indicar que no contienen ningún dato y que por lo tanto no tiene efecto funcional sobre el resultado de la solución de la historia. Una vez que un parámetro hereda o define datos, adquiere un color negro.

C) Un componente seleccionado. Todos los objetos seleccionados son verde brillante.

D) Un componente regular.

D) Un componente que contiene advertencias. Dado que es probable que un componente contenga un número de parámetros de entrada y de salida, nunca está claro qué objeto concreto genera la alerta con sólo mirar el componente. Incluso puede haber múltiples fuentes de advertencias. Tendrá que usar el menú de contexto (véase más adelante) con el fin de localizar los problemas. Tenga en cuenta que las advertencias no necesariamente tienen que ser reparadas. Pueden ser completamente legítimas.

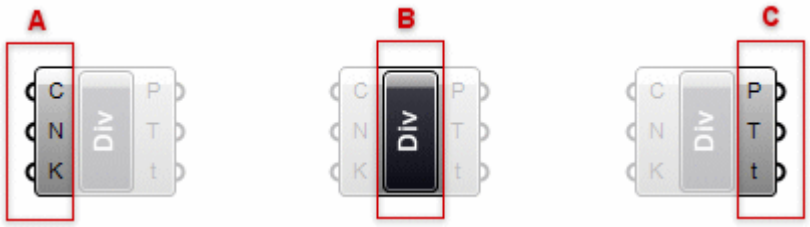
* Fuente: RhinoWiki
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryPluginObjectsExplained.html

F) Un componente que contiene errores. Similares a las advertencias, no es posible ver donde es generado el error en un componente. Tendrá que utilizar el menú de contexto (véase más adelante).Tenga en cuenta que un componente que contiene advertencias y errores aparece de color rojo, el color del error tiene prioridad sobre el color de advertencia.

G) Una conexión. Las conexiones siempre aparecen entre los parámetros de entrada y salida. No hay límite de cuántas conexiones puede contener en particular cualquier parámetro, pero no se permite crear una configuración, con conexiones cíclicas o recursivas. Tal recursividad es detectada y la solución completa está en cortocircuito cuando esto ocurre, resultando en un mensaje de error en el primer componente o parámetro que se ha detectado como recursivo. Para obtener más información sobre las conexiones, vea el capítulo sobre herencia de datos.

Partes del componente

Un componente por lo general requiere de datos para llevar a cabo sus acciones, y usualmente viene con un resultado. Por eso, la mayoría de los componentes tienen un conjunto de parámetros anidados, referido a sus parámetros de entrada y salida, respectivamente. Los parámetros de entrada están situados a lo largo del lado izquierdo y los parámetros de salida por el lado derecho:

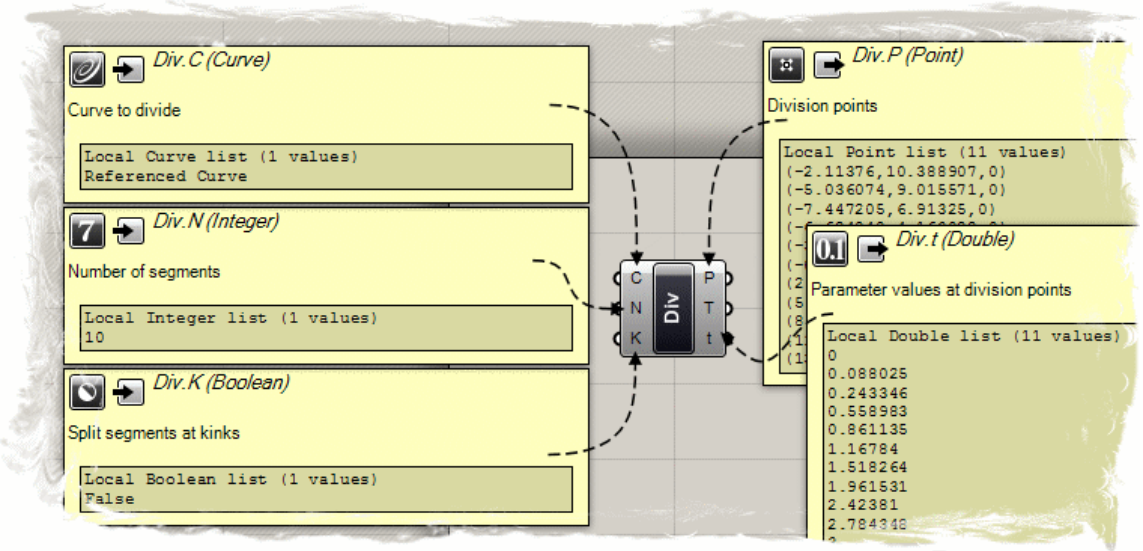


A) Los tres parámetros de entrada de la división del componente. De forma predeterminada, los nombres de parámetros son siempre extremadamente cortos. Usted puede cambiar el nombre de cada parámetro que quiera.

B) El área de la división del componente (por lo general contiene el nombre del componente)

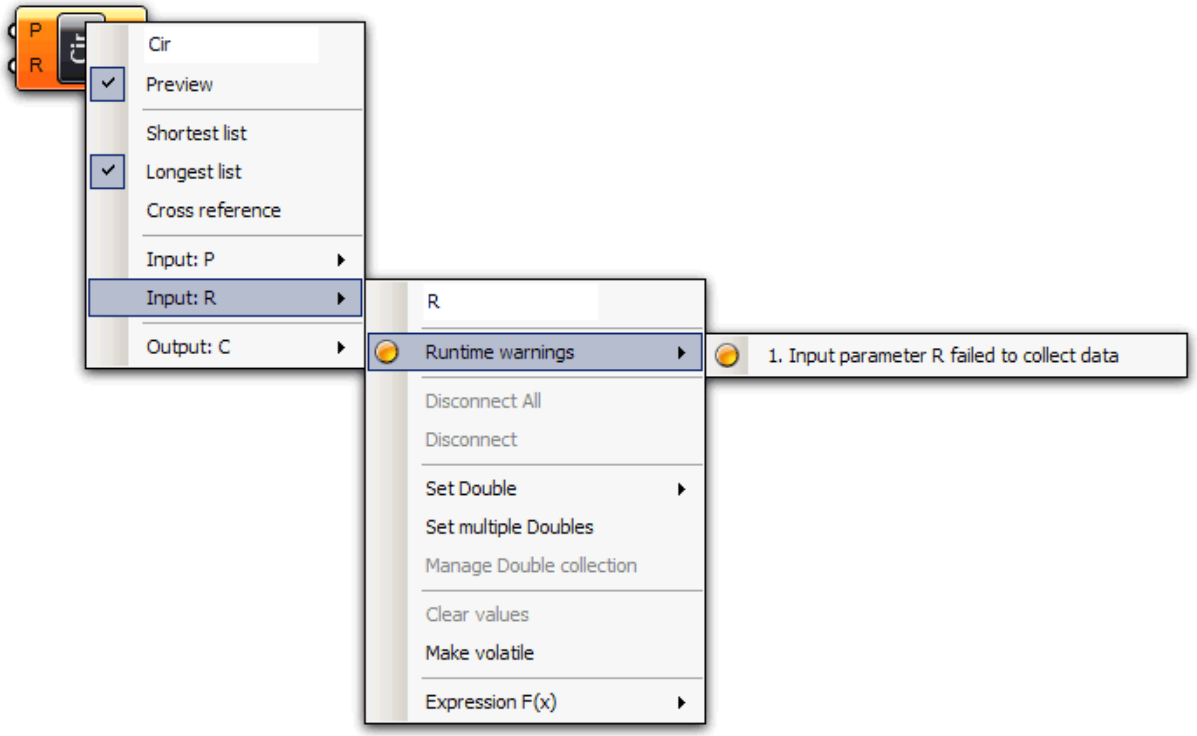
C) Los tres parámetros de salida de la división de componente.

Al pasar el ratón sobre las partes individuales de los componentes de un objeto, verá información sobre diferentes herramientas que indican el tipo particular de los (sub) objetos actualmente bajo el ratón. Las herramientas de ayuda son muy informativas, ya que dicen, tanto el tipo como los datos de los parámetros individuales:



Usando los Menús de contexto desplegables

Todos los objetos en el lienzo tienen sus propios menús contextuales que exponen la mayoría de las características de ese componente en particular. Los componentes son un poco más complicados, ya que también exponen (en estilo cascada) todos los menús de los sub-objetos que contienen. Por ejemplo, si un componente se vuelve naranja significa que quizás algún parámetro afiliado al componente genera una advertencia. Si quiere saber lo que salió mal, usted necesita utilizar el menú contextual de los componentes:



Aquí puede ver el menú principal de componentes, con el menú en cascada para el parámetro de entrada "R". El menú suele comenzar con un campo de texto editable que muestra el nombre del objeto en cuestión. Puede cambiar el nombre por algo más descriptivo, pero por defecto todos los nombres son muy cortos para reducir al mínimo el uso de pantalla del estado real. El segundo ítem en el menú (recuadro de vista previa) indica cuando o no la geometría producida/definida por este objeto será visible en las vistas de Rhino. La desconexión de la vista previa de los componentes que no contienen información vital acelerará la tasa de fotogramas de la vista de Rhino y el tiempo necesario para una "Solución de Historia" (en el caso que se trabaje con mallas). Si la vista previa de un parámetro o componente está desactivada, este aparecerá con un suave sombreado blanco. No todos los parámetros de los componentes pueden aparecer en las vistas previas (por ejemplo números) y en estos casos, el ítem "Vista Previa" suele faltar.

El menú contextual para el parámetro de entrada "R" contiene el icono de alerta naranja, que a su vez contiene una lista (sólo 1 alerta en este caso) de todas las advertencias que se han generado por este parámetro.

4 Gestión de datos persistentes*

Tipos de datos

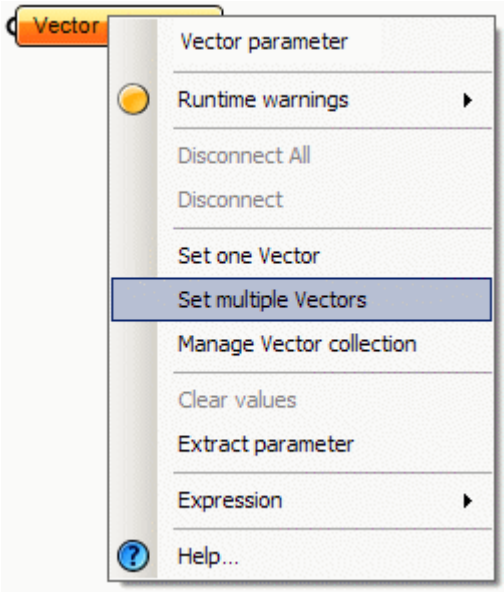
Los parámetros sólo se utilizan para almacenar información, pero la mayoría de los parámetros pueden almacenar dos tipos diferentes de datos; volátiles y persistentes. Los datos volátiles se heredan de los parámetros de una o más fuentes y son destruidos (es decir, recolectados), cuando se inicia una nueva solución. Los datos persistentes son los datos que han sido creados específicamente por el usuario. Cuando un parámetro está conectado a una fuente de objetos, los datos persistentes son ignorados, pero no destruidos.

(La excepción son los parámetros de salida que no pueden almacenar registros permanentes ni definir un conjunto de fuentes. Los parámetros de salida están totalmente bajo el control del componente que los posee.)

Los datos persistentes son accedidos mediante el menú, y dependiendo del tipo de parámetro estos cuentan con un administrador diferente. Los parámetros de vector, por ejemplo, permiten configurar los vectores de forma individual y múltiple a través del menú.

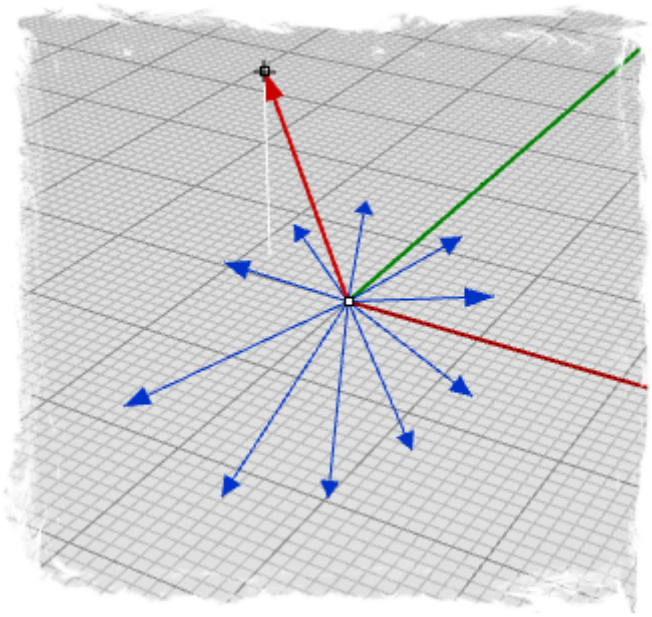
Pero, retrocedamos unos pasos y veamos cómo comporta un parámetro Vector por defecto. Una vez que lo arrastre y suelte desde el Panel de Parámetros sobre el lienzo, podrá ver lo siguiente:

El parámetro es de color naranja, indicando que se ha generado una advertencia. No es nada grave, la advertencia está simplemente allí para informarle que el parámetro está vacío (no contiene registros persistentes y no recoge los datos volátiles) y por lo tanto no tiene ningún efecto sobre el resultado de una solución de la historia. El menú de contexto del parámetro ofrece 2 modos de ajuste de datos persistentes: única y múltiple:



* Fuente: RhinoWiki
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryPersistentDataRecordManagement.html

Una vez que haga clic en cualquiera de estos elementos del menú, la ventana de Grasshopper desaparecerá y se le pedirá que elija un vector en una de las vistas de Rhino:



Una vez que haya definido todos los vectores que desea, usted puede pulsar Enter y se convertirán en parte del registro de datos persistentes de los parámetros. Esto significa que el parámetro ya no está vacío y cambia de color naranja a negro:

En este punto usted puede utilizar este parámetro para 'sembrar' cuantos objetos como desee con vectores idénticos.

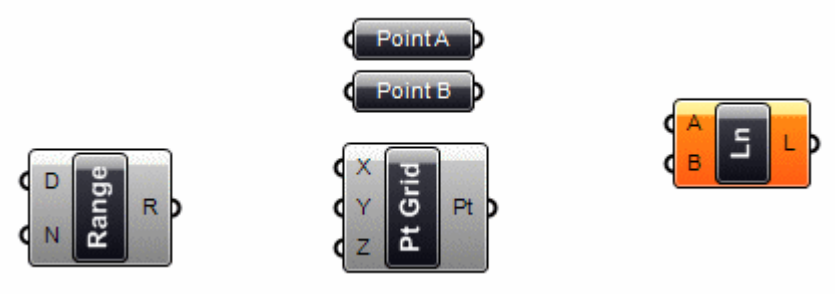
5 *Herencia de datos volátiles**

Datos de herencia

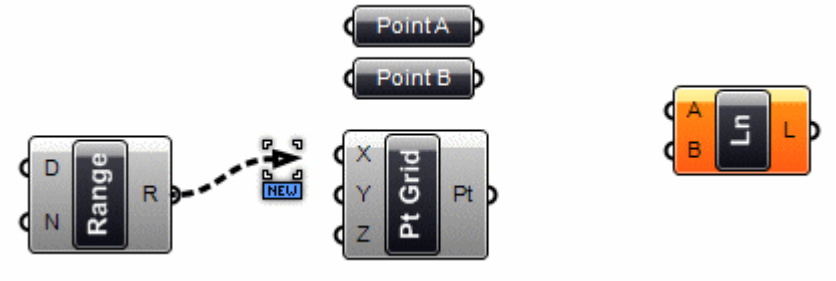
Los datos se almacenan en los parámetros (ya sea en forma volátil o persistente) y se utilizan en los componentes. Cuando los datos no se almacenan en el registro permanente de un conjunto de parámetros, deben ser heredados desde otros lugares. Todos los parámetros (excepto los parámetros de salida) definen de donde obtienen los datos y la mayoría de los parámetros no son muy particulares. Usted puede conectar un parámetro doble (que simplemente significa que es un número con un decimal) a una fuente de número entero y él se encargará de la conversión. El plug-in define muchos planes de conversión, pero si no hay un procedimiento de traducción definido, el parámetro en el extremo receptor generará un error de conversión. Por ejemplo, si usted proporciona una superficie cuando se necesita un punto, el parámetro de punto generará un mensaje de error (accesible a través del menú del parámetro en cuestión) y se volverá de color rojo. Si el parámetro pertenece a un componente, este estado de color rojo se propagará en la jerarquía y el componente se pondrá rojo también, aunque no puede contener errores de sí mismo.

Gestión de conexiones

Dado que los parámetros están a cargo de sus propias fuentes de datos, usted puede tener acceso a estos ajustes a través del parámetro en cuestión. Supongamos que tenemos una definición pequeña con tres componentes y dos parámetros:

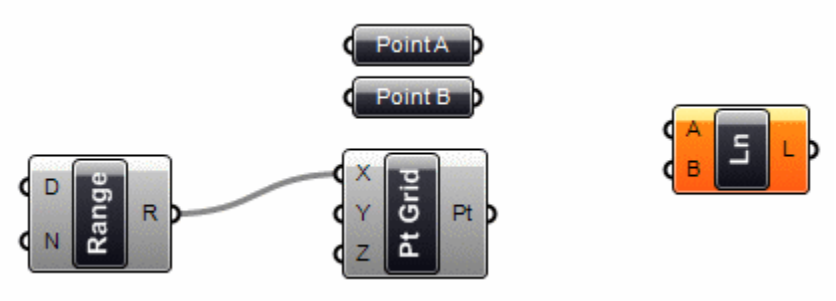


En esta etapa, todos los objetos son independientes y tenemos que empezar a conectarlos. No importa en qué orden lo hagamos, pero partamos haciéndolo de izquierda a derecha. Si usted comienza a tomarlos cerca del pequeño círculo de un parámetro, un cable de conexión se adjuntará al ratón:

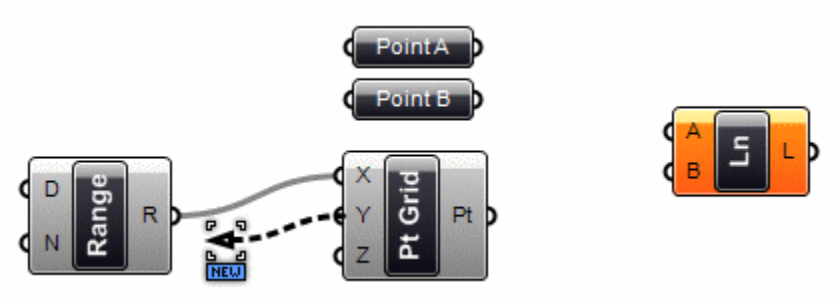


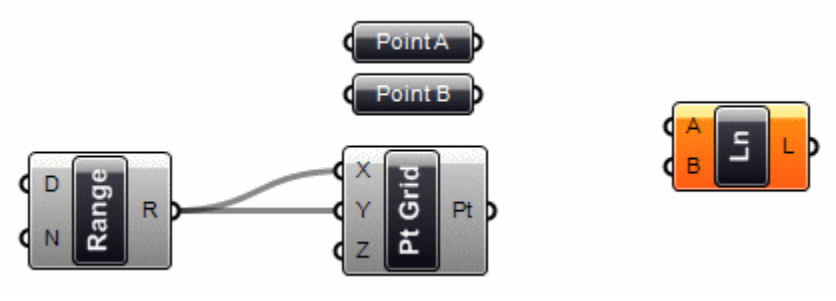
Una vez que el ratón (con el botón izquierdo todavía presionado) se pose sobre un blanco potencial de parámetro, el cable se unirá y se solidificará. Esto no es una conexión permanente hasta que suelte el botón del ratón:

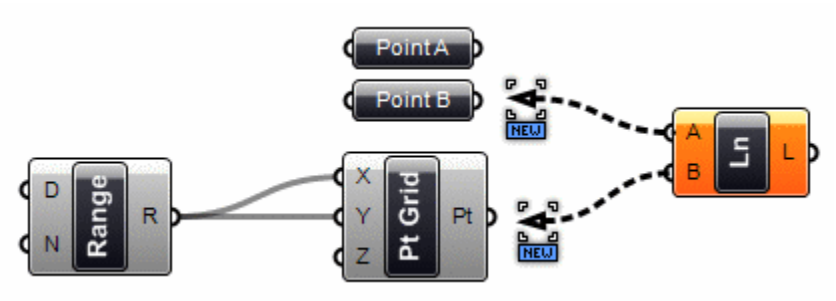
* Fuente: RhinoWiki
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryVolatileDataInheritance.html

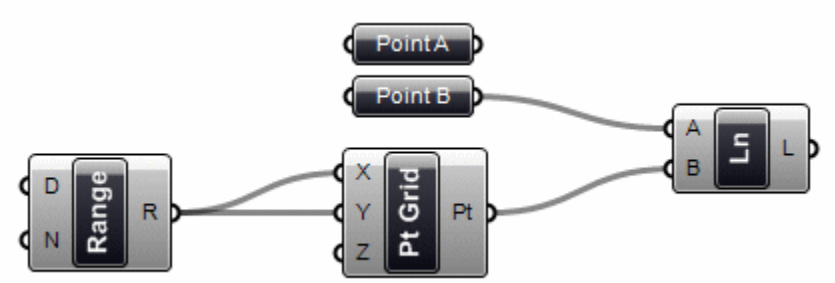


Podemos hacer lo mismo para el parámetro "Y" del componente “PtGrid” y para los parámetros "A" y "B" del componente “Line”: Clic + Arrastrar + Soltar...

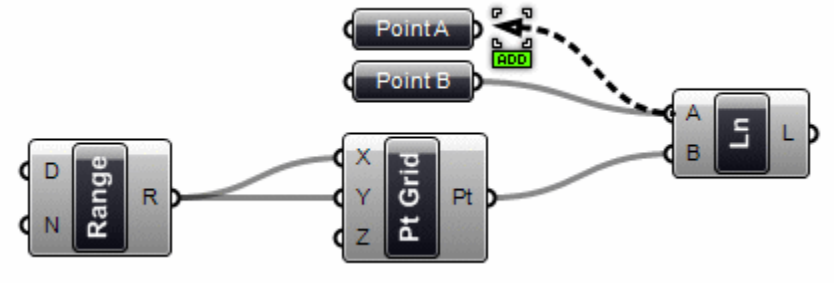


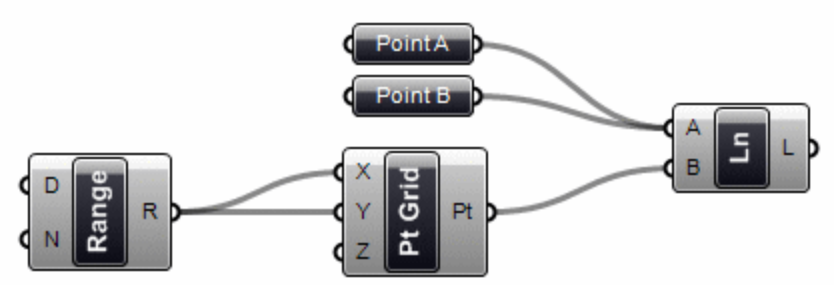






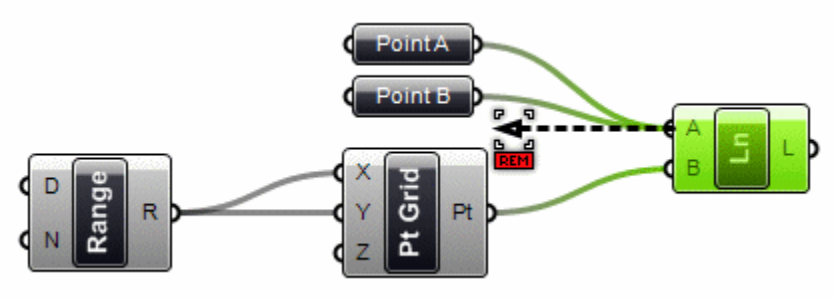
Tenga en cuenta que podemos hacer las conexiones en ambos sentidos. Pero tenga cuidado, de forma predeterminada una nueva conexión borrará las conexiones existentes. Desde que asumimos que lo más común es sólo utilizar conexiones individuales, usted tiene que hacer algo especial con el fin de definir varias fuentes. Si usted tiene apretado “Shift” mientras arrastra los cables de conexión, el puntero del ratón cambiará para indicar una nueva adición:

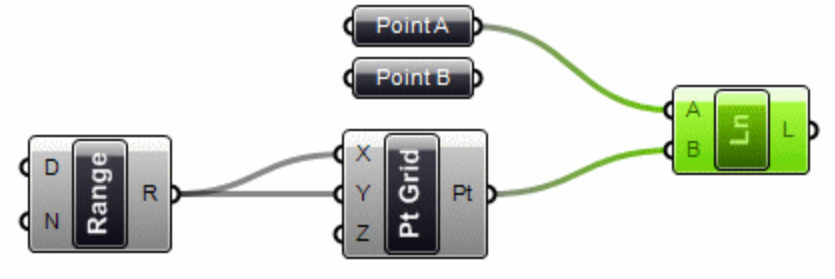




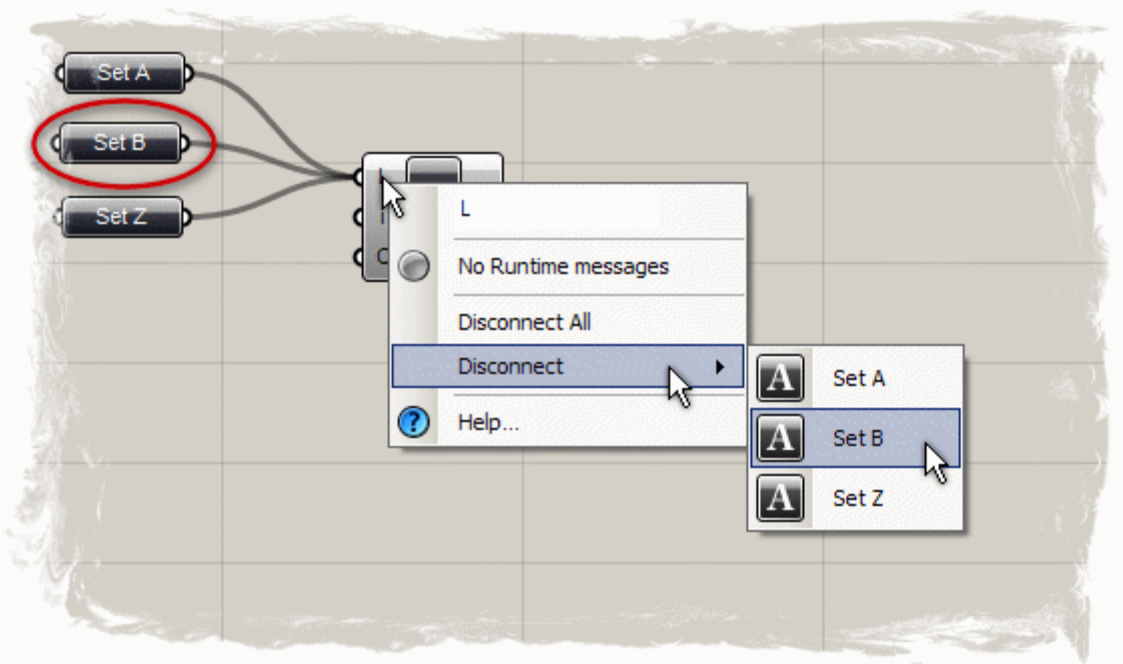
Si el cursor "ADD" está activo cuando se suelta el botón del ratón sobre un parámetro de fuente, ese parámetro se añadirá a la lista de origen. Si usted especifica un parámetro de fuente que ya ha sido definido como una fuente, no pasará nada. Usted no puede heredar de la misma fuente, más de una vez.

Por la misma razón, si mantiene presionada la tecla “Control” el cursor "REM" se hará visible, y la fuente específica se eliminará de la lista. Si el objetivo no tiene referencia, no pasará nada.

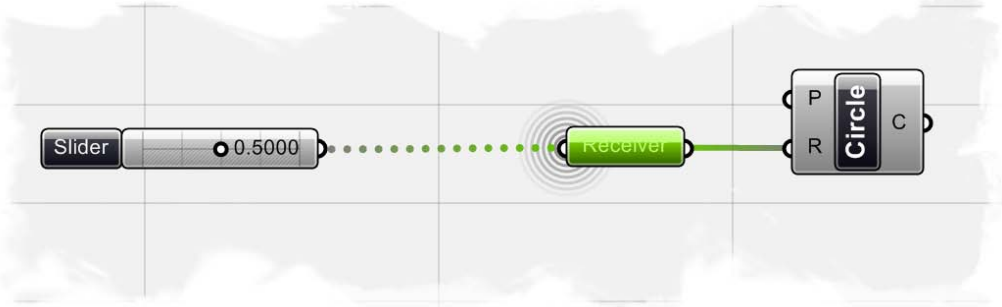




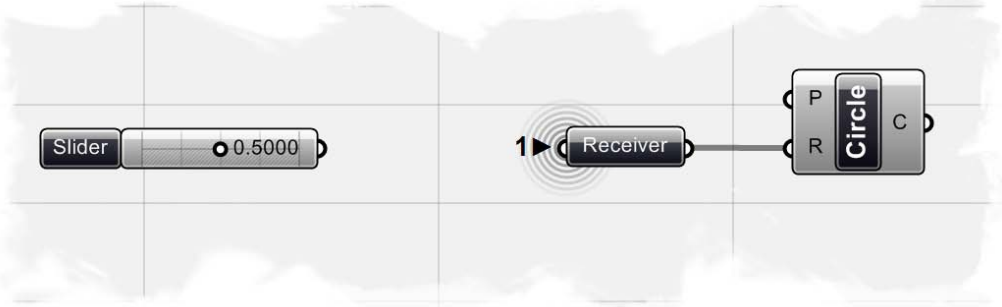
También puede desconectar (pero no conectar) las fuentes a través del menú de parámetros:



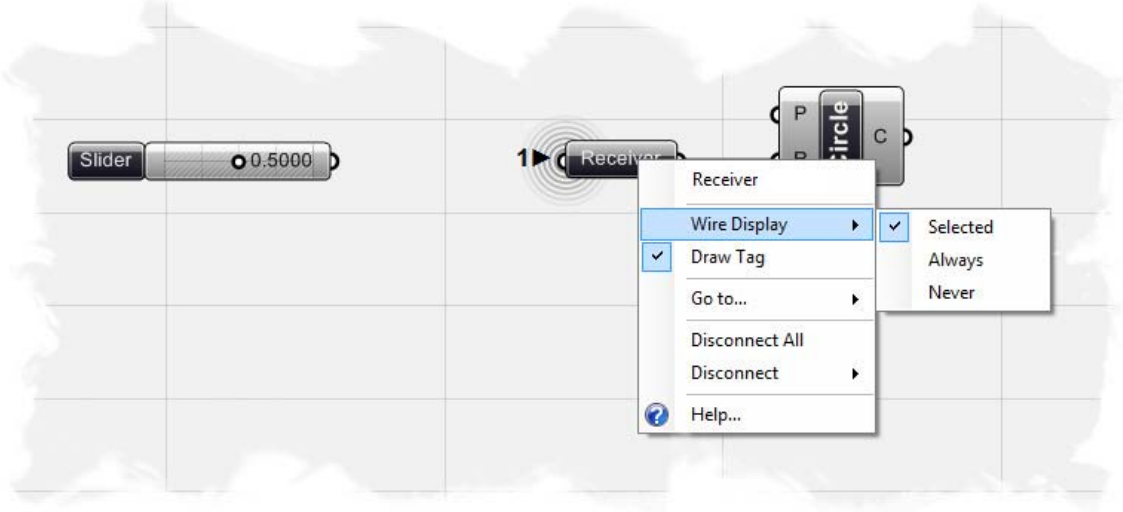
Grasshopper también tiene la capacidad de transferir información de forma inalámbrica mediante el uso de un receptor, que se encuentra bajo la subcategoría especial de la ficha “Params”. Usted puede hacer las conexiones con el receptor, tal como lo haría con cualquier otro componente. Sin embargo, tan pronto como se suelta el botón izquierdo del ratón de la conexión, el cable desaparece automáticamente. Esto ocurre porque la configuración predeterminada establece que los receptores sólo muestran sus cables de conexión con una línea punteada cuando se selecciona el receptor. Puede hacer clic en el receptor y configurar las conexiones del cable para mostrar sólo cuando el receptor está "seleccionado", o "siempre" o "nunca" mostrar los cables de conexión. Usted puede conectar la salida del receptor, con cuantos componentes como sea necesario.



Aquí, el cable de conexión se muestra con linea punteada porque se ha seleccionado el receptor del componente.



El número 1 antes de la entrada del receptor del componente indica que hay una conexión que se alimentan en la entrada. Sin embargo, dado que el componente no está seleccionado, el cable de conexión ya no aparece (pero la información sigue siendo transferida).

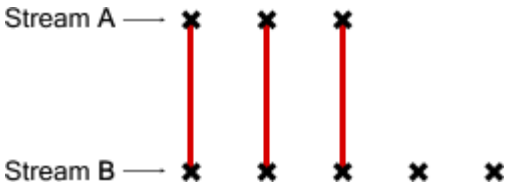


6 *Coincidencia de flujos de datos**

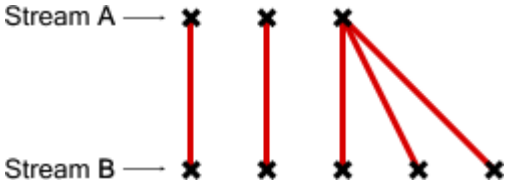
Coincidencia de datos

La coincidencia de datos es un problema sin una solución limpia. Se produce cuando un componente tiene acceso a entradas de diferente tamaño. Imagine un componente que crea segmentos de línea entre los puntos. Tendrá dos parámetros de entrada donde ambos suministran las coordenadas de los puntos ("Stream A" y "Stream B"). Es irrelevante de dónde los parámetros colectan sus datos, un componente no puede "ver" más allá de sus parámetros de entrada y salida:

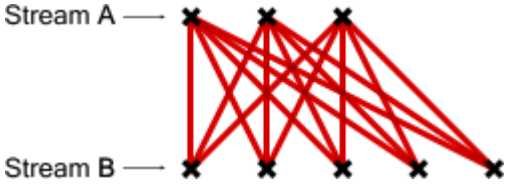
Como usted puede ver hay diferentes maneras en que podemos trazar líneas entre estos conjuntos de puntos. El plug-in Grasshopper actualmente soporta tres algoritmos de correspondencia, pero son posibles muchos más. La manera más simple es conectar las entradas uno-en-uno hasta que uno de los flujos se seque. Esto se conoce como el algoritmo "Shortest List" (Lista corta):



El algoritmo "Longest List" (Lista larga) mantiene conectando entradas hasta que todos los flujos dejen de funcionar. Este es el comportamiento por defecto de los componentes:



Por último, el metodo "Cross Reference" (Referencia cruzada) crea todas las conexiones posibles:

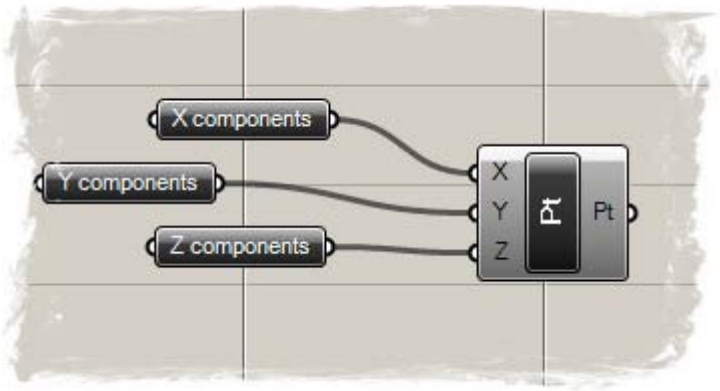


Esto es potencialmente peligroso, ya que la cantidad de salidas puede ser enorme. El problema se vuelve más intrincado cuando mas parámetros de entrada están implicados y cuando la herencia de datos volátiles comienza a multiplicar los datos, pero la lógica sigue siendo la misma.

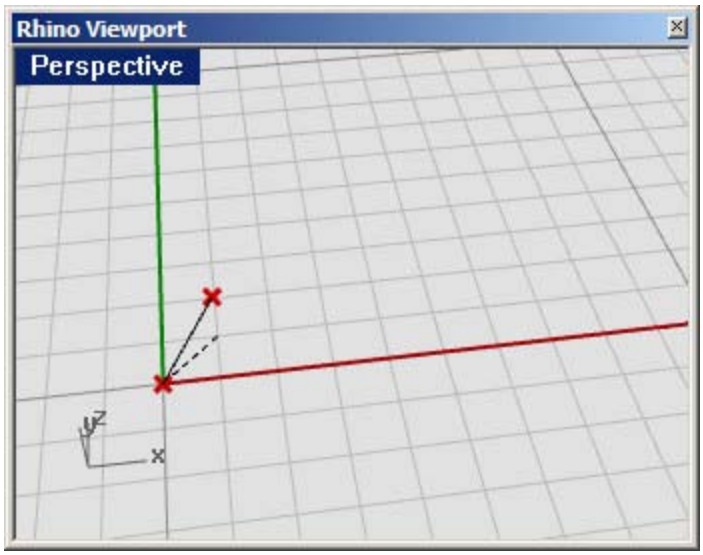
* Fuente: RhinoWiki
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryDataStreamMatchingAlgorithms.html

Imagine que tenemos un componente que hereda sus valores de X, Y y Z desde parámetros remotos que contienen los siguientes datos:

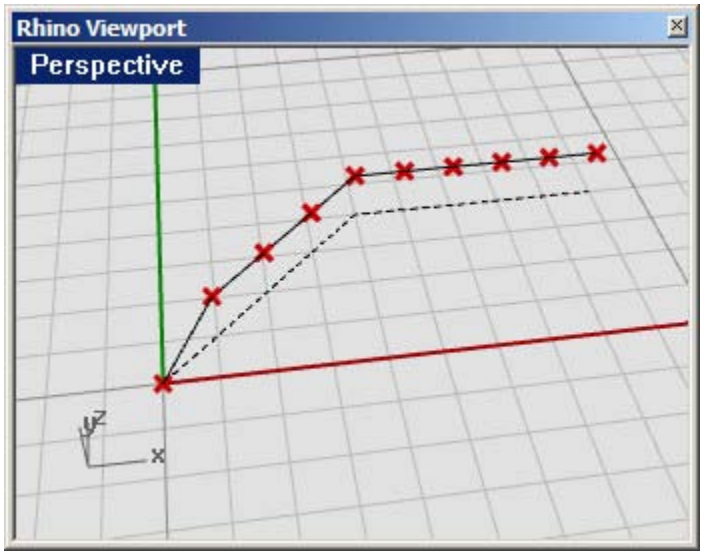
Coordenadas X: {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0}
Coordenadas Y: {0.0, 1.0, 2.0, 3.0, 4.0}
Coordenadas Z: {0.0, 1.0}



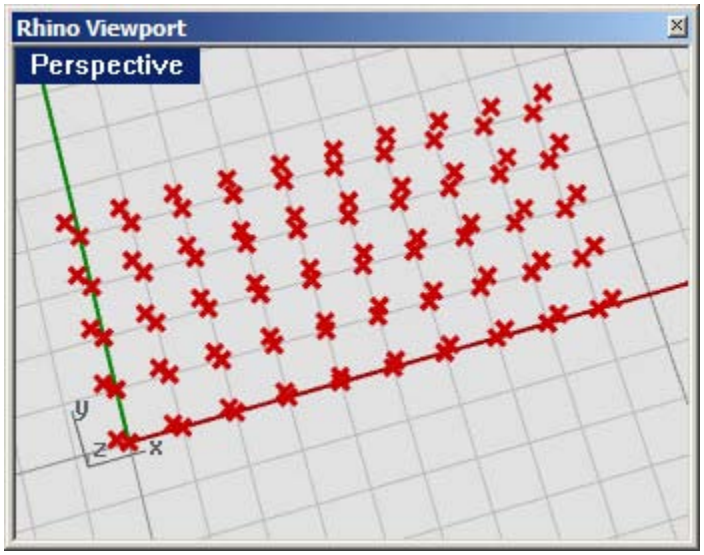
Si combinamos estos datos en el modo "lista corta", obtenemos sólo dos puntos ya que la "coordenada Z" contiene sólo dos valores. Dado que esta es la lista más corta, esta define el alcance de la solución:



El método "Lista larga" creará diez puntos, recorriendo los más altos valores posibles de los flujos Y y Z:



El método "Referencia cruzada" conectará todos los valores de X con todos los valores de Y y Z, por lo que el resultado es de 10 × 5 × 2 = un centenar de puntos:



Cada componente se puede configurar para obedecer a una de estas normas (el ajuste está disponible en el menú haciendo clic derecho sobre el icono del componente).

Tenga en cuenta la gran excepción a este comportamiento. Algunos componentes esperan obtener una lista de datos en uno o más de sus campos de entrada. El componente de polilínea, por ejemplo, crea una curva de polilínea a través de una matriz de puntos de entrada. Más puntos en el parámetro de entrada se traducirán en una polilínea más larga, pero no en más polilíneas. Los parámetros de entrada que esperan crear más de un valor se llaman parámetros de lista y estos son ignorados en los datos coincidentes.

7 *Tipos de componentes escalares*

Los tipos de componentes escalares se utilizan normalmente para diversas operaciones matemáticas y consisten en:

A) **Constantes**. Devuelve un valor constante, como Pi, Radio dorado, etc ..

C) **Intervalos**. Se utiliza para dividir los extremos numéricos (o dominios) en partes de intervalo. Hay muchos componentes en la pestaña intervalos que le permiten crear o descomponer un número en diferentes tipos de intervalo.

D) **Operadores**. Utilizados en las operaciones matemáticas como sumar, restar, multiplicar, etc ...

E) **Polinomios**. Se utiliza para elevar a un valor numérico por alguna potencia.

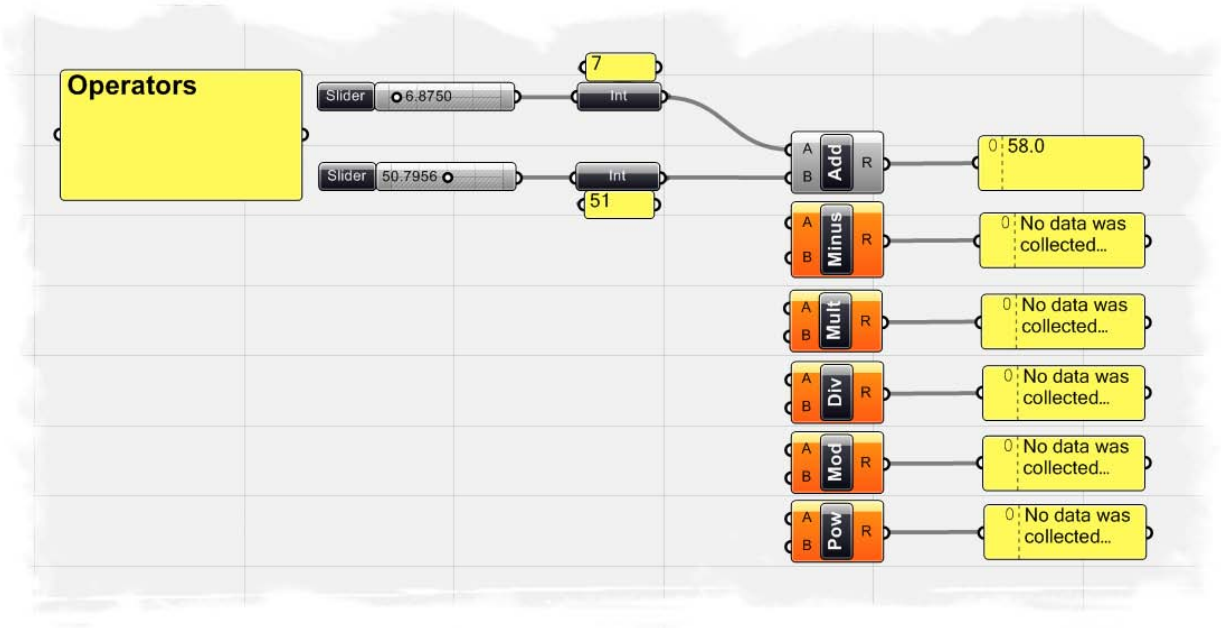
F) **Trigonometría**. Devuelve típicos valores trigonométricos tales como seno, coseno, y tangente, etc ...

G) **Utilidad** (Análisis). Se utiliza para evaluar entre dos o más valores numéricos.

7.1 Operadores

Como se mencionó anteriormente, los operadores son un conjunto de componentes que utilizan las funciones algebraicas con dos valores de entrada numérica, que se traducen en un valor de salida. Para entender mejor los operadores, vamos a crear una definición matemática simple para explorar los diferentes tipos de componentes de operador.

Nota: Para ver la versión final de esta definición, abra el archivo **Scalar_operators.ghx** que se encuentra en la carpeta “Archivos de apoyo” que acompaña a este documento. A continuación se muestra una captura de pantalla de la definición completa.



Para crear la definición desde el inicio:

- Params/Special/Numeric Slider – Arrastre y suelte un componente “Numeric Slider” sobre el lienzo.
- Doble clic en el deslizador y determine lo siguiente:
  - Lower limit: 0.0

    - Upper limit: 100.0
    - Value: 50.0 (Nota: este valor es arbitrario y puede ser modificado a cualquier valor dentro de los límites superior e inferior).
- Seleccione el deslizador y haga Ctrl+C (copiar) y Ctrl+V (pegar) para crear una copia del deslizador.
- Params/Primitive/Integer – Arrastre y suelte **dos** componentes “Integer” sobre el lienzo.
- Conecte el deslizador 1 al primer componente “Integer”.
- Conecte el deslizador 2 al segundo componente “Integer”.

    El valor por defecto del deslizador se establece en “Floating Point” (lo que resulta en un valor numérico decimal). Al conectar el deslizador en “Integer component”, se puede convertir a un número entero. Cuando conectamos panel “Post-It” (Params/Special/Panel) al valor de salida de cada componente de número entero, podemos ver la conversión en tiempo real. Mueva el control deslizante hacia la izquierda y la derecha y observe como el valor de número decimal se convierte a un número entero. Por supuesto, podríamos haber simplificado este paso sólo estableciendo el tipo de control deslizante a enteros.

- Scalar/Operators/Add – Arrastre y suelte un componente “Add” sobre el lienzo.
- Conecte el primer componente “Integer” a la entrada A del componente “Add”.
- Conecte el segundo componente “Integer” a la entrada B del componente “Add”.
- Params/Special/Panel – Arrastre y suelte un panel “Post-it” sobre el lienzo.
- Conecte la salida R de “Add” a la entrada del panel “Post-it”.

    Ahora puede ver el valor de la suma de los dos números enteros en el panel “Post-it”.

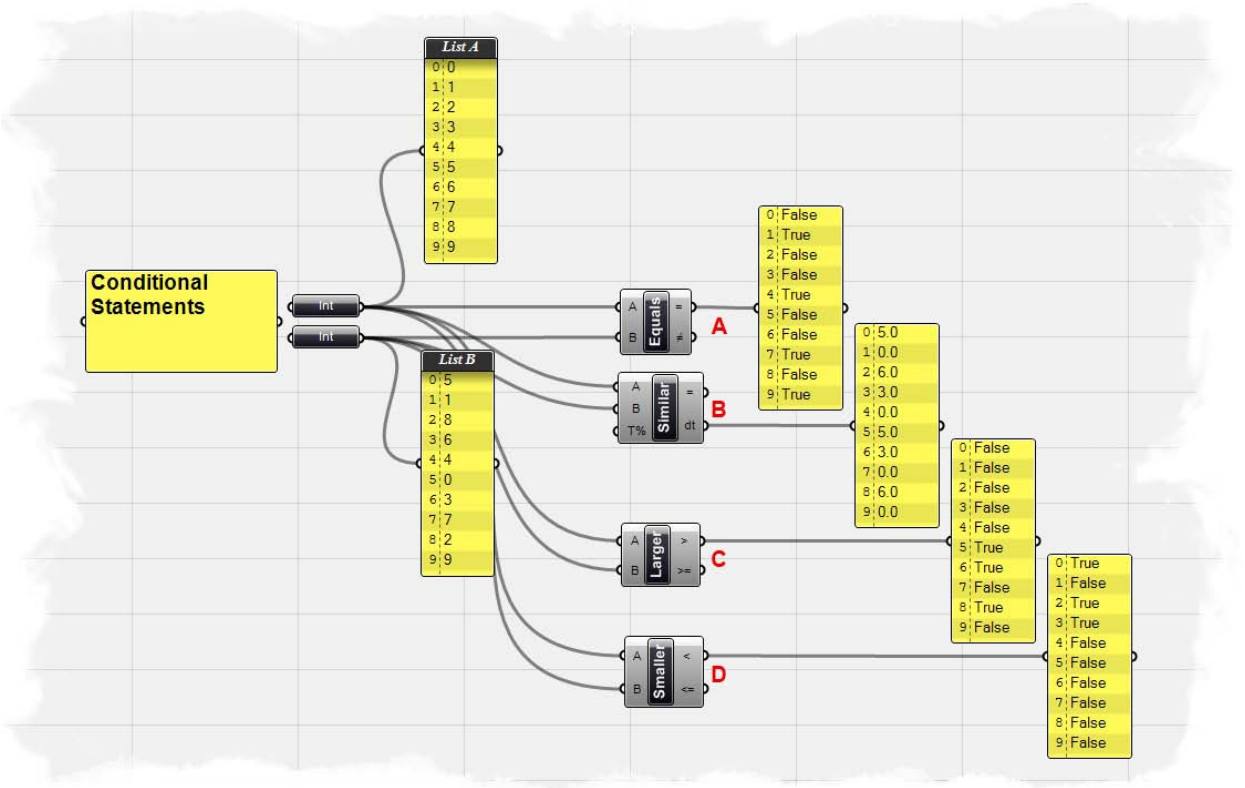
- Arrastre y suelte los operadores de escalamiento restantes sobre el lienzo:
    - Subtraction
    - Multiplication
    - Division
    - Modulus
    - Power
- Conecte el primer componente “Integer” a cada entrada A de los operadores.
- Conecte el segundo componente “Integer” a cada entrada B de los operadores.
- Arrastre y suelte **cinco** paneles “Post-it” sobre el lienzo y conecte cada panel a las salidas de los operadores.

La definición está completada y ahora cuando usted cambie cada uno de los valores de los deslizadores, podrá ver el resultado de la acción de cada operador en los paneles “Post-it”.

7.2 Sentencias condicionales

Usted probablemente ha notado que hay unos cuantos componentes en la subcategoría de los operadores escalares que no utilizamos en la última sección. Eso es porque hay 4 componentes (nuevos para la versión 0.6.0007) que actúan de forma diferente que los operadores matemáticos, dado que estos comparan dos listas de datos en lugar de realizar una expresión algebraica. Los cuatro componentes son "**Equality**", "**Similarity**", "**Larger Than**", y "**Smaller Than**" y se explican en detalle más adelante.

Nota: Para ver la versión final de esta definición, abra el archivo **Conditional Statements.ghx** que se encuentra en la carpeta "Archivos de apoyo" que acompaña a este documento.
A continuación se muestra una captura de pantalla de la definición completa.



A) El componente **Equality** (Igualdad) toma dos listas y compara el primer elemento de la lista A con el primer elemento de la lista B. Si los dos valores son los mismos, entonces un valor booleano "True" (Verdadero) es creado; de forma inversa, si los dos valores no son iguales, entonces un valor booleano "False" (Falso) se crea. El componente hace un recorrido a través de las listas de acuerdo con el algoritmo de coincidencia de datos (por defecto está en la opción "Lista larga"). Hay dos salidas para este componente. La primera devuelve una lista de valores booleanos que muestra cuales valores son iguales entre sí. La segunda devuelve una lista que muestra cuales valores no son iguales entre sí.

B) El componente **Similarity** (Similitud) evalúa las dos listas de datos y prueba la similitud entre dos números. Es casi idéntico a la forma en que el componente de igualdad compara las dos listas, con una excepción... en que posee una entrada de porcentaje que define la relación en que la lista A es desigual de la lista B, antes de que se suponga la desigualdad, además el componente de similitud tiene una salida que determina el valor de la distancia absoluta entre las dos listas de entrada.

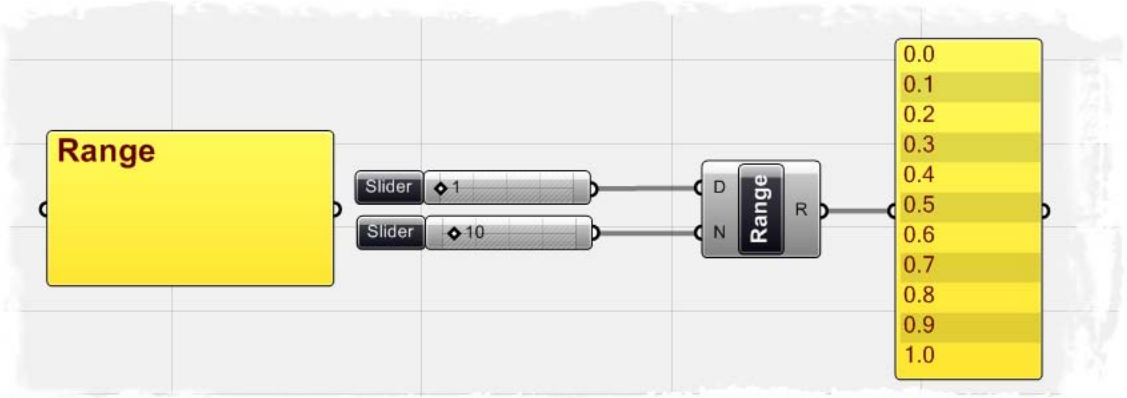
C) El componente **Larger Than** (Más grande que) tomará dos listas de datos y determinará si el primer elemento de la lista A es mayor que el primer elemento de la lista B. Las dos salidas permiten determinar si usted quiere evaluar las dos listas, con una condición mayor que (>) o superior e igual a (> =).

D) El componente **Smaller Than** (Más pequeño que) realiza la acción opuesta del componente “más grande que”. Este componente determina si es que la lista A es menor que la lista B y devuelve una lista de valores booleanos. Del mismo modo, las dos salidas le permiten determinar si desea evaluar cada lista de acuerdo a una condicion menor que (<) o inferior e igual a (<=).

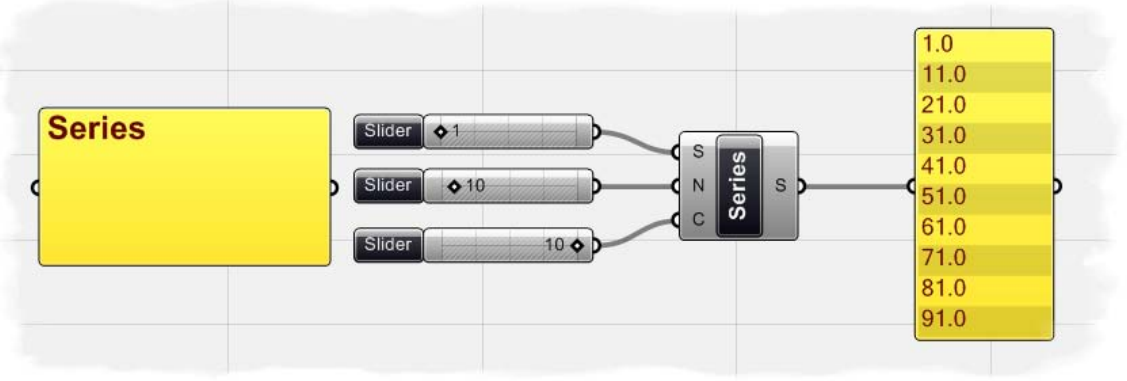
7.2 Rango vs. Series vs. Intervalos

Todos los componentes de Rango, Serie e Intervalo crean un conjunto de valores numéricos entre dos extremos, sin embargo los componentes operan de maneras diferentes.

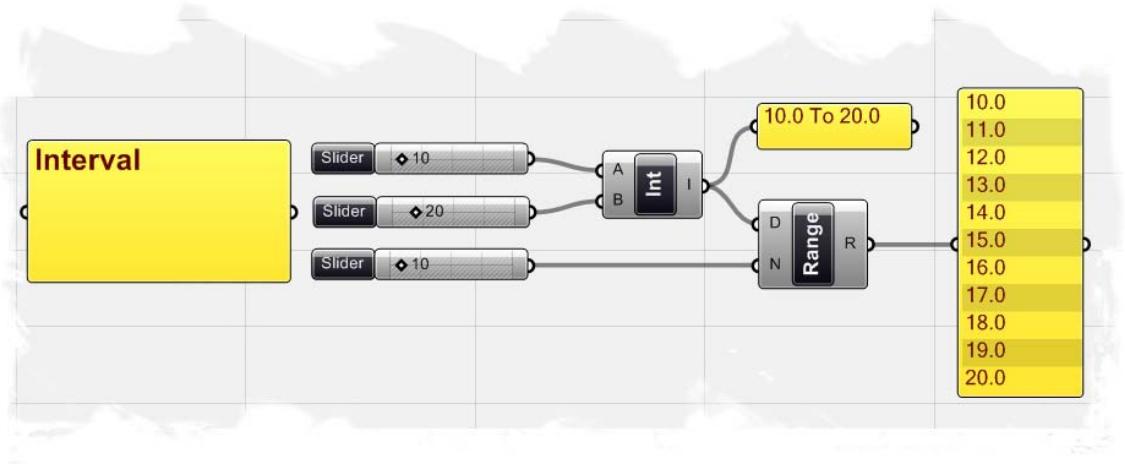
Nota: Para ver la versión final de los ejemplos siguientes, abra el archivo **Scalar_intervals.ghx** que se encuentra en la carpeta “Archivos de apoyo” que acompaña a este documento.



El componente de Rango crea una lista de números uniformemente espaciados entre un valor bajo y un valor alto llamados dominio de rango numérico. En el ejemplo anterior, dos deslizadores numéricos están conectados a los valores de entrada del componente de Rango. El primer deslizador define el dominio numérico para el rango de valores. En este ejemplo, el dominio ha sido definido a partir de cero a uno, ya que el deslizador se encuentra en 1. El segundo control deslizante define el número de pasos para dividir el dominio, que en este caso se ha establecido en 10. Así, la salida es una lista de 11 números divididos entre 0 y 1. (Nota: El segundo control deslizante, establecido en 10, está definiendo el número de divisiones entre 0 y 1, con lo cual se obtienen 11 números, no 10).



El componente de Serie crea un conjunto de números discretos sobre la base de un valor de inicio, el tamaño del espaciado, y el número de valores en la Serie. El ejemplo de la serie muestra tres deslizadores numéricos relacionados con el componente de Serie. Cuando el primer deslizante se conecta a la entrada S de la Serie, define el punto de partida para la serie de números. El segundo control deslizante, determinado en 10, define el valor del espaciado de la serie. Dado que, el valor inicial se ha fijado en 1 y el tamaño del paso se ha establecido en 10, el siguiente valor en la serie será 11. Por último, el tercer deslizante define el número de valores en la serie. Puesto que este valor ha sido ajustado en 10, el resultado final de los valores definidos en la serie muestra 10 números, que comienzan en 1 y aumentan de 10 en cada paso.



El componente de Intervalo crea un rango de todos los números posibles entre un número de bajo y otro alto. El componente de Intervalo es similar al dominio numérico que hemos definido para el componente Rango. La principal diferencia es que el componente Rango crea un dominio numérico por defecto entre 0 y cualquier otro valor de entrada definido. En el componente de Intervalo, los valores bajo y alto pueden ser definidos por las entradas A y B. En el siguiente ejemplo, hemos definido una serie de todos los valores posibles de entre 10 y 20, establecidos por los dos deslizadores numéricos. El valor de salida para el componente Intervalo ahora muestra valores desde 10,0 a 20,0 lo que refleja nuestro nuevo dominio numérico. Si ahora conectamos la salida I del Intervalo a un Rango de entrada de D, podemos crear una serie de números entre los valores del Intervalo. Como fue el caso en el ejemplo anterior del Rango, si ponemos el número de pasos para que el Rango en 10, ahora veremos los 11 valores divididos entre el valor inferior del Intervalo de 10,0 y el valor del Intervalo superior de 20,0. (Nota: Hay múltiples maneras para definir un intervalo, y usted puede ver varios otros métodos enumerados en la ficha “Scalar/Interval”. Tan sólo hemos definido un componente de Intervalo simple, pero vamos a discutir algunos otros métodos en los próximos capítulos)

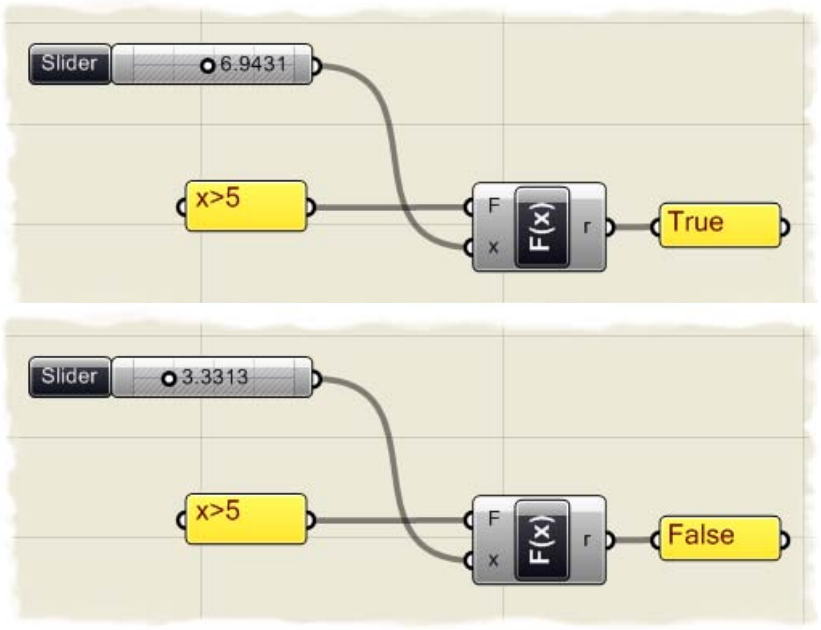
7.3 Funciones & Booleanas

Casi todos los lenguajes de programación tienen un método para la evaluación de declaraciones condicionales. En la mayoría de los casos, el programador crea un pedazo de código para hacer una simple pregunta “what if?” (¿y si?). ¿Y si el ataque terrorista del 9/11 nunca había ocurrido? ¿Qué pasa si el gas cuesta 10$/galon? Estas son preguntas importantes que representan un mayor nivel de pensamiento abstracto. Los programas de ordenador también tienen la capacidad de analizar "¿qué pasaría si?", y tomar acciones en función de la respuesta a la pregunta. Echemos un vistazo a una sentencia condicional muy simple que un programa puede interpretar:

Si el objeto es una curva, elimínelo.

El trozo de código primero mira a un objeto y determina un valor booleano simple de si es o no es una curva. No hay términos medios. El valor booleano es “**True**” (Verdadero) si el objeto es una curva, o “**False**” (Falso) si el objeto no es una curva. La segunda parte de la declaración realiza una acción según el resultado de la sentencia condicional, en este caso, si el objeto es una curva, a continuación, eliminarlo. Esta sentencia condicional se llama “**If/Else**” (Si/Entonces); Si el objeto se ajusta a ciertos criterios, entonces hacer algo, sino, hacer otra cosa.

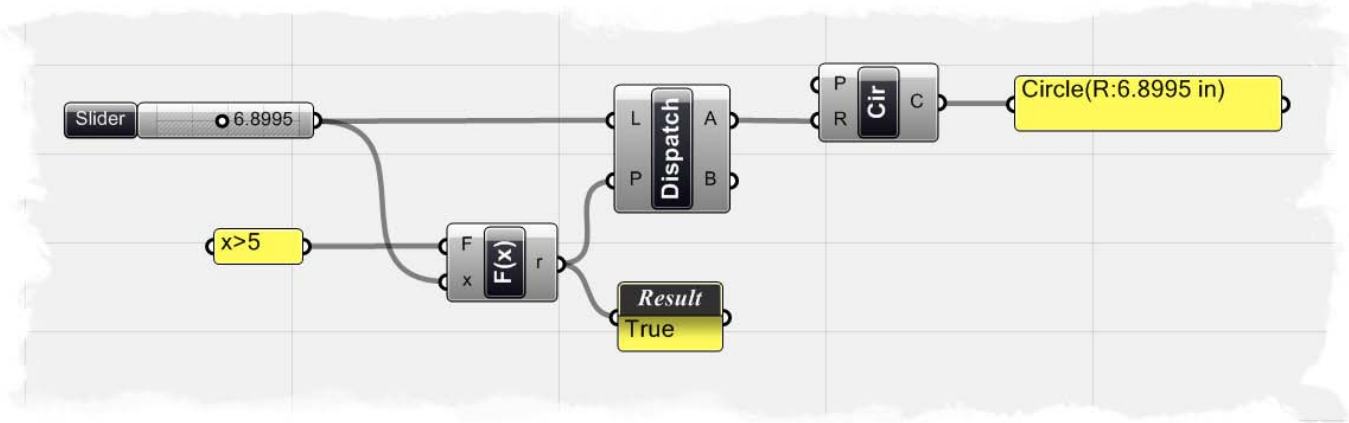
Grasshopper posee la misma capacidad para analizar las sentencias condicionales a través de la utilización de las componentes de Función.



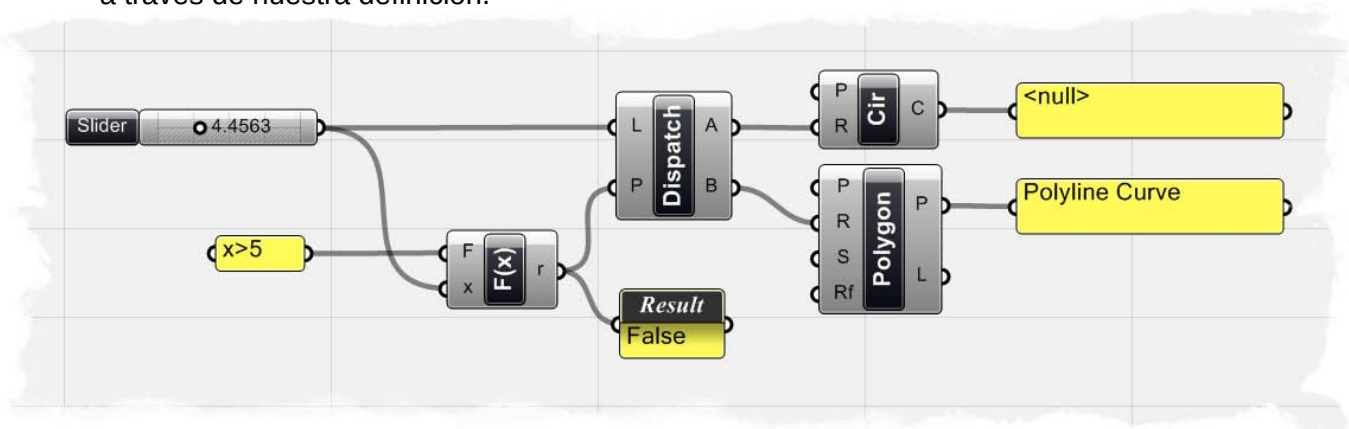
En el ejemplo anterior, hemos conectado un control deslizante con la entrada X de un componente de Función de variable única (Logic/Script/F1). Además, hemos conectado una sentencia condicional a la entrada F de la Función, definiendo la pregunta: "¿Es x mayor que 5?" Si el control deslizante está establecido por encima de 5, la salida R para la función muestra un booleano verdadero. Si el deslizante numérico es inferior a 5, a continuación, el salida R cambia a un valor falso.

Una vez que hayamos determinado el valor booleano de la función, se puede alimentar el patrón de información Verdadero/Falso en un componente de despacho (Logic/List/Dispatch) para realizar una determinada acción.

El componente “Dispatch” trabaja tomando una lista de información (en nuestro ejemplo hemos conectado el deslizador numérico a la entrada de L de “Dispatch”) y filtrando esta información basado en el resultado booleano de la función de variable única. Si el patrón muestra un valor verdadero, la información de la lista se pasará a la salida A del “Dispatch”. Si el patrón es falso, pasa la información de la lista a la salida B del “Dispatch”. Para este ejemplo, hemos decidido crear un círculo solo si el valor del control deslizante es mayor a 5. Hemos conectado un componente “Circle” (Curve/Primitive/Circle) a la salida A de “Dispatch”, de modo que un círculo con un radio determinado por el control deslizante se creará sólo si el valor booleano en el componente “Dispatch” es Verdadero. Dado que ningún elemento se ha conectado con la salida B de “Dispatch”, si el valor booleano es falso, entonces no sucederá nada y un círculo no se creará.



Podemos llevar esta definición un poco más allá, conectando una curva “N-sided Polygon” (Curve/Primitive/Polygon) a la salida B de “Dispatch”, y asegúrese de conectar la entrada R al “Polygon” para definir el radio de este. Ahora bien, si el control deslizante está por debajo de 5, luego un polígono de 5 lados con un radio definido por el deslizante numérico, será creado en el punto de origen. Si tomamos el valor del deslizador superior a 5, a continuación, un círculo será creado. Mediante este método, podemos empezar a crear tantas sentencias “**Si/Entonces**” como sea necesario para alimentar la información a través de nuestra definición.



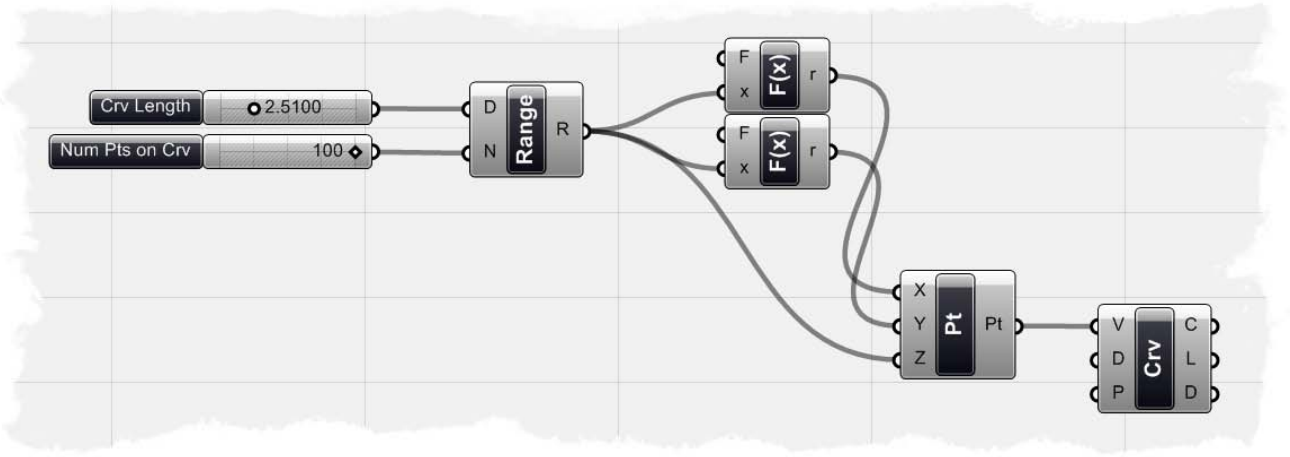
Nota: Para ver la versión final de esta prueba del círculo booleano, abra el archivo **If_Else test.ghx** encuentra en la carpeta “Archivos de apoyo” que acompaña a este documento.

7.4 Funciones & Datos numericos

El componente de Función es muy flexible; es decir, que puede ser usado en una variedad de aplicaciones diferentes. Ya hemos hablado de cómo podemos usar un componente de Función para evaluar una sentencia condicional y ofrecer una salida de valor booleano. Sin embargo, también la podemos utilizar para resolver complejos algoritmos matemáticos y visualizar en la salida los datos numéricos resultantes.

En el siguiente ejemplo, vamos a crear una espiral matemática similar al ejemplo de David Rutten encontrado en su manual de *RhinoScript 101*. Para obtener más información acerca de RhinoScript o para descargar una copia del manual, visite: http://en.wiki.mcneel.com/default.aspx/McNeel/RhinoScript101.html

Nota: Para ver la versión final de este ejmplo del espiral matemático, abra el archivo **Function_spiral.ghx** encontrado en la carpeta “Archivos de apoyo” que acompaña a este documento.
A continuación se muestra una captura de pantalla de la definición completa.

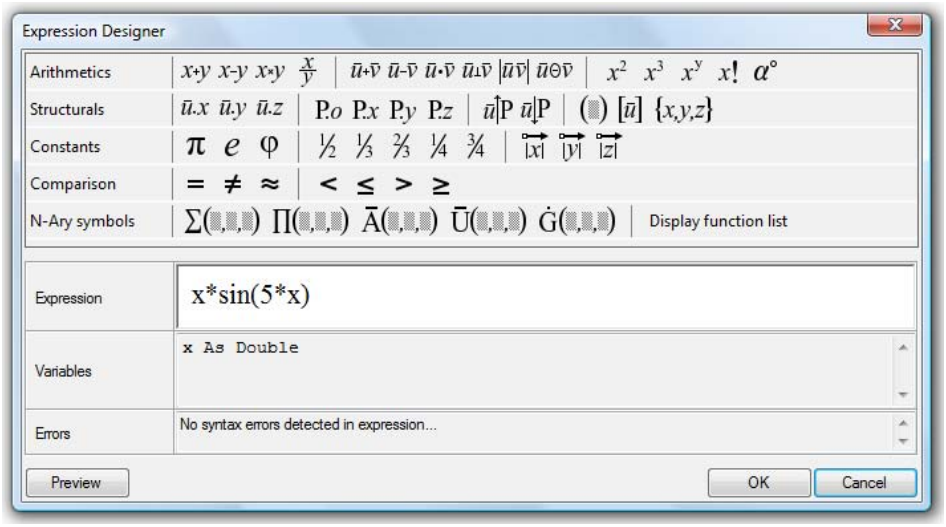


Para crear la definición desde el inicio:

- Logic/Sets/Range - Arrastre y suelte un componente “Range” sobre el lienzo.
- Params/Special/Slider - Arrastre y suelte **dos** componentes “Numeric Slider” sobre el lienzo.
- Clic derecho en el primer deslizador y determine lo siguiente:
  - Name: Crv Length
  - Slider Type: Floating Point (esto es determinado por defecto)
  - Lower Limit: 0.1
  - Upper Limit: 10.0
  - Value: 2.5
- Ahora, clic derecho en el segundo deslizador y determine lo siguiente:
  - Name: Num Pts on Crv
  - Slider Type: Integers
  - Lower Limit: 1.0
  - Upper Limit: 100.0
  - Value: 100.0
- Conecte el deslizador “Crv Length” a la entrada D del componente “Range”.
- Conecte el deslizador “Num Pts on Crv” a la entrada N del componente “Range”.

  Acabamos de crear una serie de 101 números uniformemente espaciados, que van desde 0,0 hasta 2,5, los que ahora podemos conectar a nuestros componentes de Función.

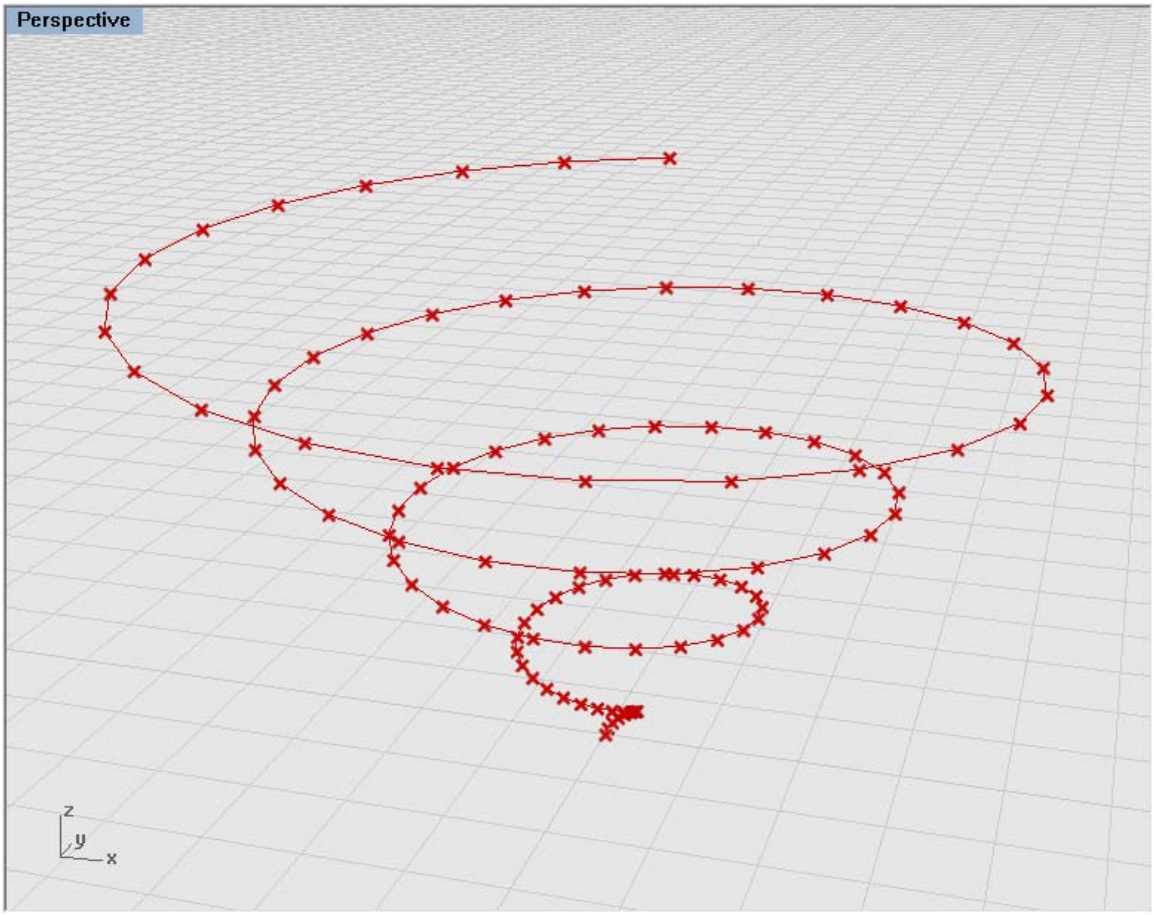
- Logic/Script/F1 - Arrastre y suelte un componente “Function” sobre el lienzo.
- Clic derecho sobre la entrada F del componente “Function” y abra “Expression Editor”.
- En la caja de dialogo del “Expression Editor”, escriba la siguiente ecuación:
  - **x*sin(5*x)**
    Si ha introducido el algoritmo en el editor correctamente, debería ver una declaración que dice: "No syntax errors detected in expression" en el marco de despliegue de errores.
  - Clic OK para aceptar el algoritmo.



- Seleccione el componente “Function” y presione Ctrl+C (copiar) y Ctrl+V (pegar) para crear un duplicado de esta función.
- Clic derecho en la entrada F del componente “Function” duplicado y abra el “Expression Editor”.
- En la caja de dialogo del “Expression Editor”, escriba la siguiente ecuación:
  - **x*cos(5*x)**
    ***Nota:** la única diferencia en esta ecuación es que hemos sustituido la función seno por la función del coseno.*
  - Clic OK para aceptar el algoritmo.
- Conecte la salida R de “Range” a la entrada X de ambos componentes “Function”.
  Hemos conectado los 101 números creados por el componente “Range” dentro del componente de “Function”, el cual resuelve un algoritmo matemático y entrega una nueva lista de datos numéricos. Usted puede colocar el ratón sobre la salida R de cada función para ver el resultado de cada ecuación.
- Vector/Point/Point XYZ - Arrastre y suelte un componente “Point XYZ” sobre el lienzo.
- Conecte la salida R del primer “Function” a la entrada X de “Point XYZ”.
- Conecte la salida R del segundo “Function” a la entrada Y de “Point XYZ”.
- Conecte la salida R de “Range” a la entrada Z del componente “Point XYZ”.
  Ahora bien, si nos fijamos en la vista de Rhino, verá una serie de puntos que forman un espiral. Usted puede cambiar valores de los dos deslizadores numéricos al comienzo de la definición para cambiar el número de puntos o la longitud de la espiral.

- Curve/Spline/Curve - Arrastre y suelte un componente “Curve” sobre el lienzo.
- Conecte la salida Pt del componente “Point” a la entrada V de “Curve”.

Hemos creado una única curva que pasa por cada punto de la espiral. Podemos hacer clic derecho sobre la entrada D de “Curve” para establecer el grado de curva, una curva de 1 grado creará segmentos de línea recta entre cada punto y asegurará que la curva pasa a través de cada punto. Una curva de grado 3 creará una curva de Bezier suave donde los puntos de la espiral actuarán como puntos de control para la curva, sin embargo la línea en realidad no toca cada punto.



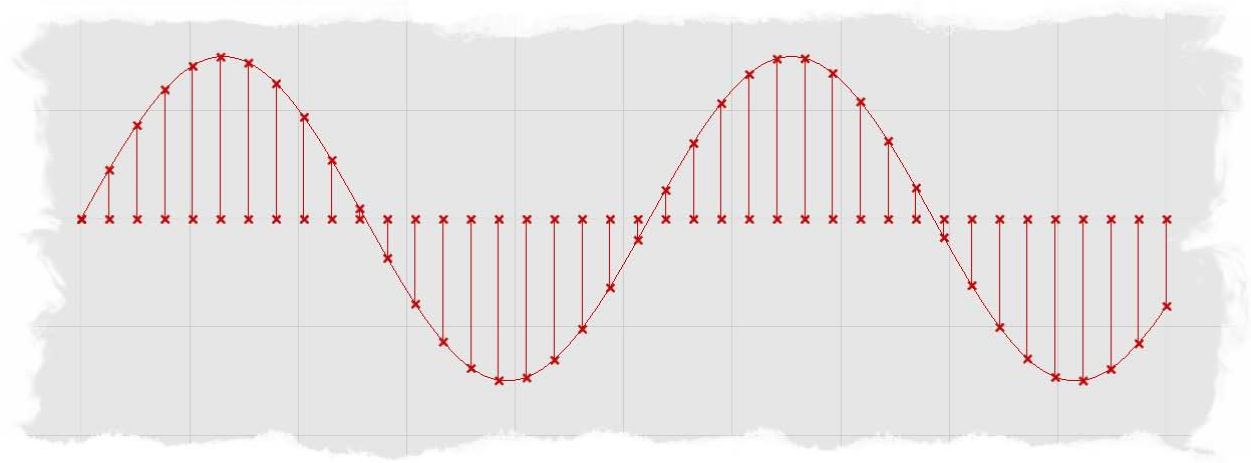
Nota: Para ver un video tutorial de este ejemplo, visite el blog de Zach Downey en: http://www.designalyze.com/2008/07/07/generating-a-spiral-in-rhinos-grasshopper-plugin/

7.5 Curvas Trigonometricas

Ya hemos demostrado que podemos usar los componentes de “Function” para evaluar fórmulas complejas para crear espirales y otras curvas matemáticas, sin embargo Grasshopper también tiene un conjunto de componentes trigonométricos integrados en la familia de los componentes escalares. Las funciones trigonométricas, como seno, coseno y tangente son herramientas importantes para los matemáticos, científicos e ingenieros, ya que definen una relación entre dos lados de un triángulo rectángulo con un ángulo específico, llamado Theta. Estas funciones son importantes en el análisis de vectores, que vamos a cubrir en capítulos posteriores. Sin embargo, usted también puede utilizar estas funciones para definir fenómenos periódicos, como funciones de onda sinusoidal encontradas en la naturaleza en forma de olas del océano, las ondas sonoras y las ondas de luz. En 1822, Joseph Fourier, matemático francés, descubrió que las ondas sinusoidales se pueden utilizar como bloques de construcción simple y describir casi cualquier forma de onda periódica. Este proceso se denomina análisis de Fourier.

En el siguiente ejemplo, vamos a crear una forma de onda sinusoidal, donde el número de puntos de la curva, la frecuencia de longitud de onda y amplitud se pueden controlar por un conjunto de deslizadores numéricos.

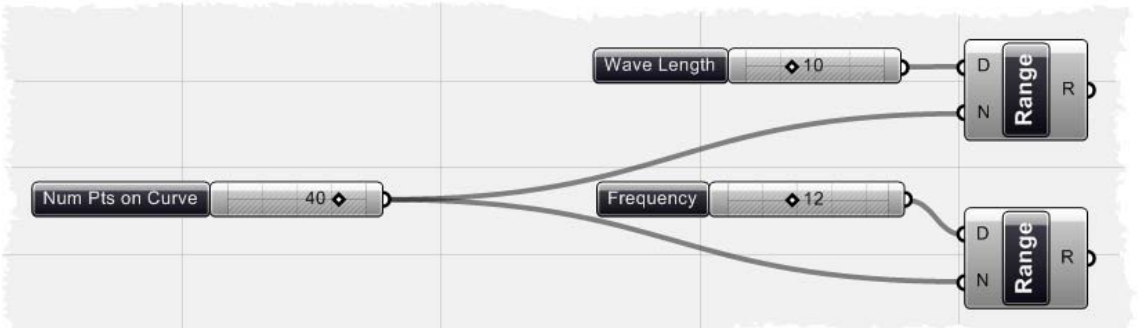
Nota: Para ver la definición final que se utilizó para crear la curva sinusoidal, abra el archivo **Trigonometric_curves.ghx** que se encuentra en la carpeta “Archivos de apoyo” que acompaña a este documento



Para comenzar la definición desde el inicio:

- Params/Special/Slider - Arrastre y suelte **tres** “Numeric Sliders” sobre el lienzo.
- Clic derecho sobre el primer deslizador y determine lo siguiente:
  - Name: Num Pts on Curve
  - Slider Type: Integers
  - Lower Limit: 1
  - Upper Limit: 50
  - Value: 40
- Clic derecho sobre el segundo deslizador y determine lo siguiente:
  - Name: Wave Length
  - Slider Type: Integers
  - Lower Limit: 0
  - Upper Limit: 30
  - Value: 10

- Clic derecho sobre el tercer deslizador y determine lo siguiente:
  - Name: Frequency
  - Slider Type: Integers
  - Lower Limit: 0
  - Upper Limit: 30
  - Value: 12
- Logic/Sets/Range - Arrastre y suelte **dos** componentes “Range” sobre el lienzo.
- Conecte el deslizador “Wave Length” a la entrada D del primer “Range”.
- Conecte el deslizador “Frequency” a la entrada D del segundo “Range”.
- Conecte el deslizador “Num Pts on Curve” a la entrada N de ambos “Range”.

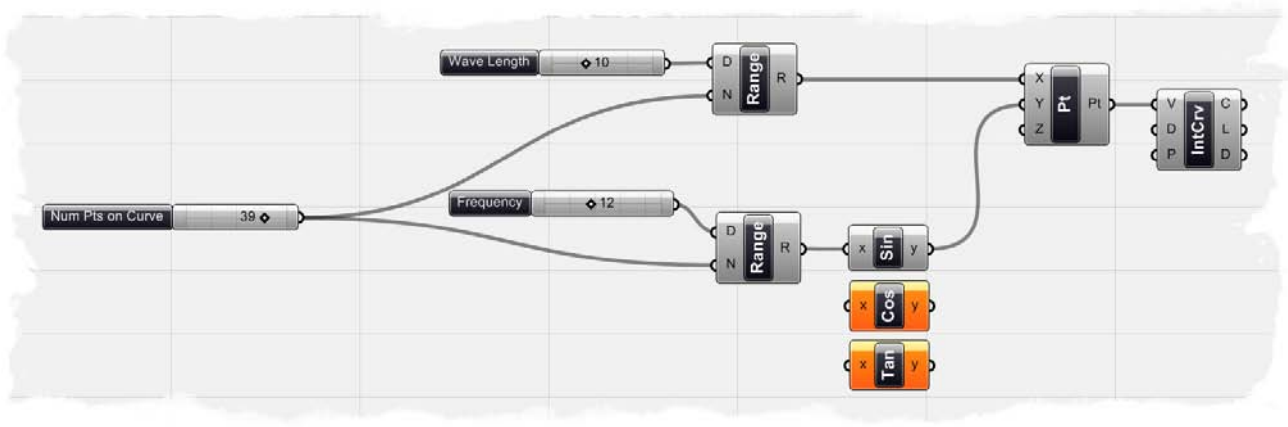


Su definición debe ser similar a la imagen de arriba. Hemos creado dos listas de datos: la primera es una serie de números divididos entre 0 y 10, y la segunda es una lista de números divididos entre 0 y 12.

- Scalar/Trigonometry/Sine - Arrastre y suelte un componente “Sine” sobre el lienzo.
- Conecte la salida R del segundo “Range” a la entrada X del componente “Sine”.
- Vector/Point/Point XYZ - Arrastre y suelte un componente “Point XYZ” sobre el lienzo.
- Conecte la salida R del primer “Range” a la entrada X de “Point XYZ”.
- Conecte la salida Y del componente “Sine” a la entrada Y del componente “Point XYZ”.

  Si nos fijamos en la vista de Rhino, debería ver una serie de puntos en forma de una onda sinusoidal. Desde que la salida del primer “Range” se conecta directamente a la entrada X del componente “Point”, sin haber sido conectado a un componente de función trigonométrica, nuestro valor x de los puntos son constantes y espaciados uniformemente. Sin embargo, nuestro componente “Sine” es conectado a la entrada Y del componente “Point”, de modo que vemos un cambio en el valor y en nuestros puntos, lo que finalmente forma un patrón de onda. Ahora usted puede cambiar cualquiera de los deslizadores numéricos para cambiar la forma del patrón de onda.
- Curve/Spline/Interpolate - Arrastre y suelte un componente “Interpolate” sobre el lienzo.
- Conecte la salida Pt del componente “Point” a la entrada V de “Curve”.

En este punto, su definición debe ser similar a la captura de pantalla a continuación. Hemos creado nuestra definición manipular el Número de puntos en la curva, la Longitud de onda, y la Frecuencia, pero podemos crear un deslizador mas para a controlar la amplitud de la curva sinusoidal.

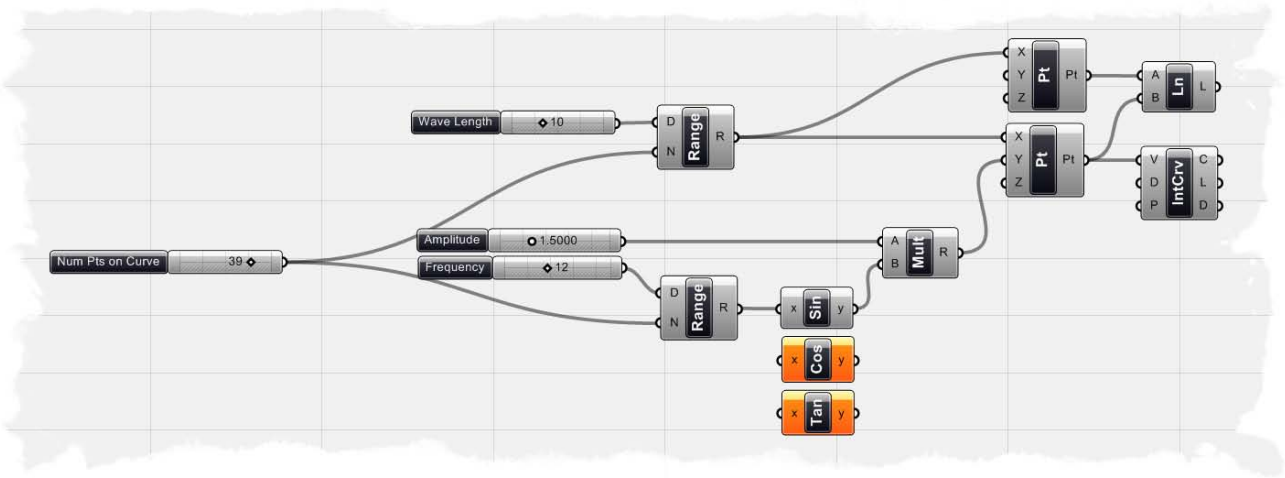


- Params/Special/Slider - Arrastre y suelte un componente “Numeric Slider” sobre el lienzo.
- Clic derecho sobre el nuevo deslizador y determine lo siguiente:
  - Name: Amplitude
  - Slider Type: Floating Point
  - Lower Limit: 0.1
  - Upper Limit: 5.0
  - Value: 2.0
- Scalar/Operators/Multiplication - Arrastre y suelte un componente “Multiplication” sobre el lienzo.
- Conecte el deslizador “Amplitude” a la entrada A del componente “Multiplication”.
- Conecte la salida Y del componente “Sine” a la entrada B de “Multiplication”.
- Conecte la salida R de “Multiplication” a la entrada Y de “Point XYZ”.

  Esta conexión debe sustituir a la conexión existente, que había sido previamente conectado a la salida de componente “Sine”. El control deslizante “Amplitude” sólo está multiplicando los valores de “Sine” por un factor de escala. Dado que el regulador está impulsando el valor Y de nuestra curva de seno, cuando aumenta el valor de “Amplitude”, también aumentará la amplitud de la curva.

- Vector/Point/Point XYZ - Arrastre y suelte otro componente “Point XYZ” sobre el lienzo.
- Conecte la salida R del primer “Range” a la entrada X del nuevo componente “Point XYZ”.
- Curve/Primitive/Line - Arrastre y suelte un componente “Line” sobre el lienzo.
- Conecte la salida Pt del primer “Point” a la entrada B del componente “Line”.
- Conecte la salida Pt del segundo “Point” a la entrada S de “Line”.

  En la última parte de la definición, hemos creado una segunda serie de puntos espaciados uniformemente a lo largo del eje X, que corresponden a las mismas coordenadas en x de la curva sinusoidal. El componente “Line” crea un segmento de línea entre la primera lista de puntos, los que crean la curva sinusoidal, y la segunda lista de puntos que definen el eje X. Las nuevas líneas darán una referencia visual del desplazamiento vertical en el patrón de forma de onda. Su definición debe ser similar a la imagen de abajo.

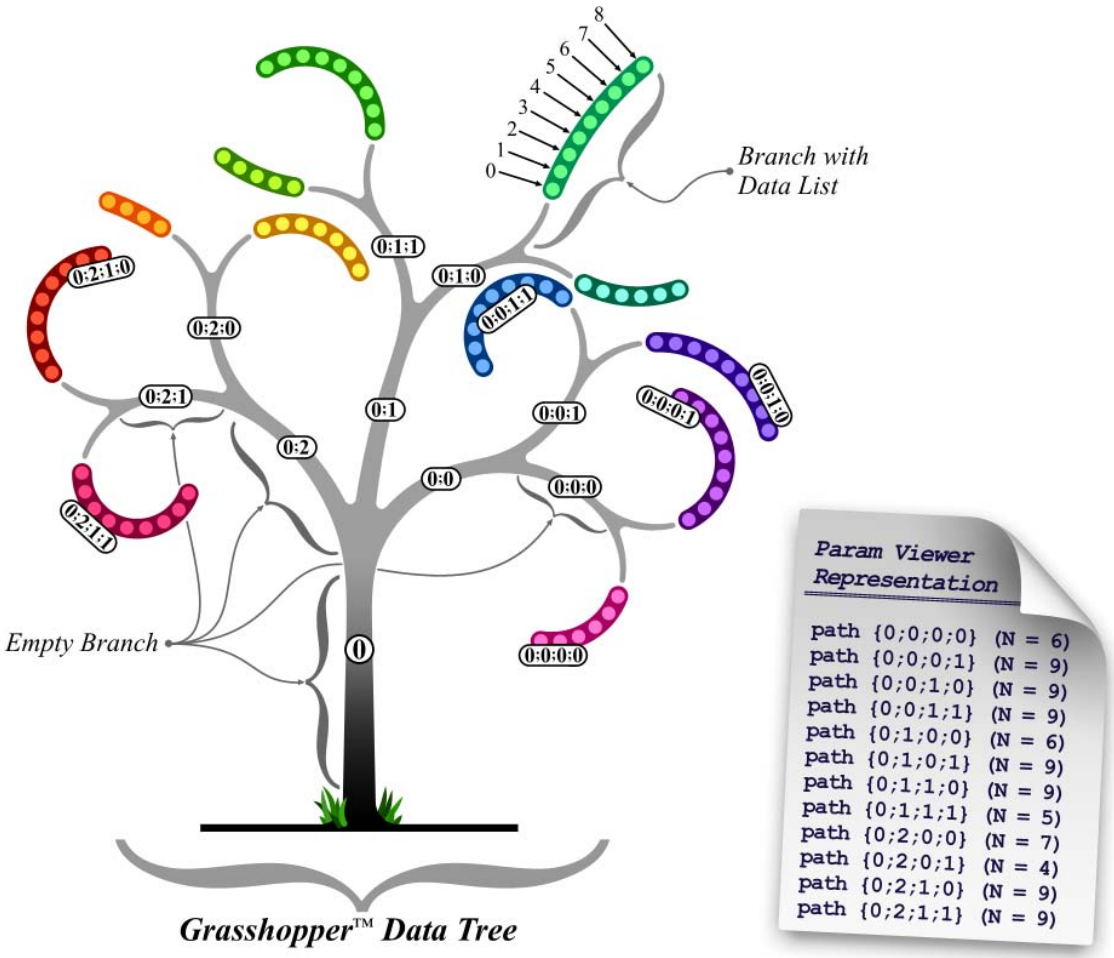


Hemos mostrado cómo crear una curva sinusoidal, sin embargo, usted puede generar otras curvas sinusoidales, como formas de onda de coseno o de tangente, sustituyendo el componente senoidal con un coseno, tangente u otro componente encontrado en la ficha Scalar/Trigonometry.

Nota: Para ver el video tutorial de este ejemplo, visite blog de David Fano, en:
http://designreform.net/2008/06/01/rhino-3d-sine-curve-explicit-history/

8 El jardín de senderos que bifurcan

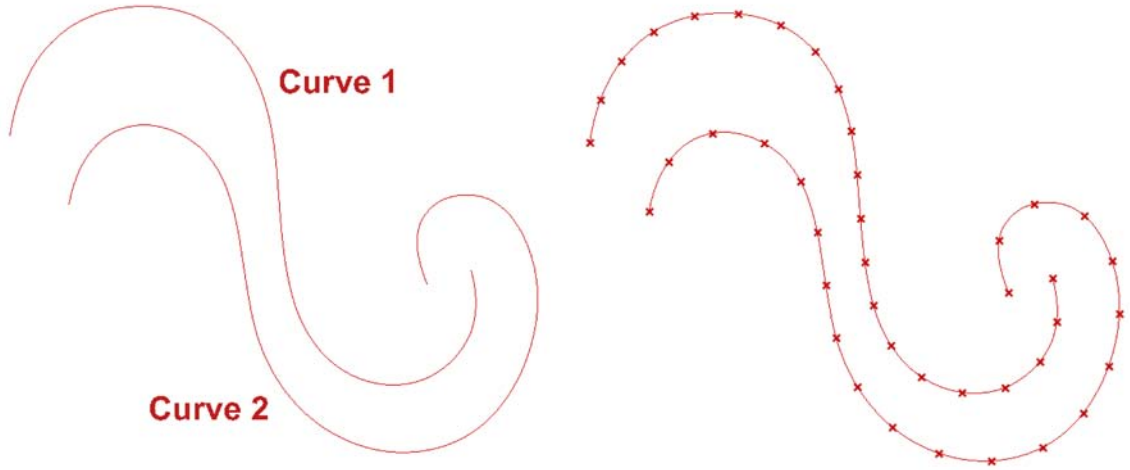
En todas las versiones de Grasshopper antes de la versión 0.6.00xx, los datos dentro de un parámetro se almacenaban en una lista única, y como tal, no necesitaba un índice para esta la lista. Sin embargo, ha habido una revisión completa de los datos que se almacenan en Grasshopper, y ahora es posible tener varias listas de datos dentro de un único parámetro. Dado que varias listas están disponibles, es necesaria una manera de identificar a cada lista individual. A continuación se muestra una imagen, creada por David Rutten, que representa una complejidad razonable de un diagrama de árbol bien estructurado.



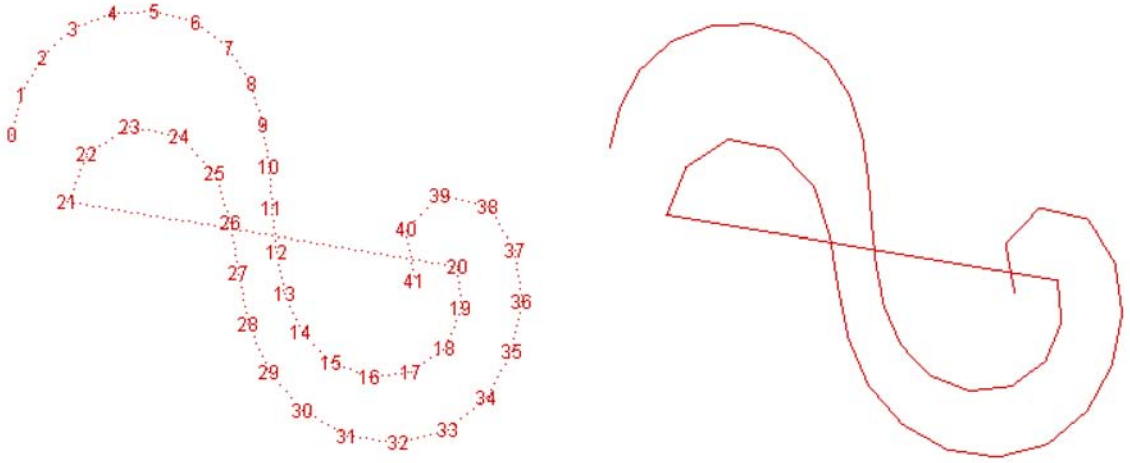
En la imagen de arriba, hay una rama maestra única (que podríamos llamar un baúl, pero como es posible tener varias ramas maestras, puede ser un poco eufemismo) en la ruta (0). Este camino no contiene datos, pero tiene 3 sub-ramas. Cada una de estas sub-ramas hereda el índice de la rama madre (0) y añade su propio sub-índice (0, 1 y 2, respectivamente). No sería correcto llamar a esto un "índice", porque implica que sólo un número único. Probablemente es mejor referirse a esto como un "camino". Cada una de estas sub-ramas se volverá a acoger a 2 sub-su-bramas y de nuevo no contienen datos en sí. Lo mismo es cierto para las sub-sub-ramas. Una vez que alcanzan el nivel de anidación 4, finalmente nos encontramos con algunos datos (las listas de colores se representan como líneas gruesas, los elementos de datos son círculos brillantes). Cada sub-sub-sub-sucursales (o nivel 4 de rama) es una rama de la terminal, lo que significa que no se subdividirá más.

Cada elemento de datos es, pues, parte de uno (y sólo uno) en la rama del árbol, y cada elemento tiene un índice que especifica su ubicación dentro de la rama. Cada rama tiene una ruta que especifica su ubicación en el árbol.

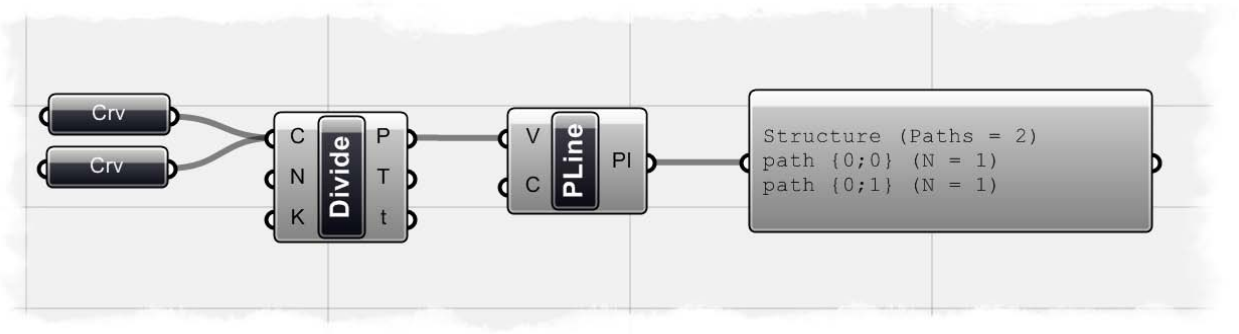
Para examinar más a fondo las estructuras del árbol, echemos un vistazo a un ejemplo muy simple. Empezaremos con dos curvas que se referenciaremos dentro de Grasshopper. A continuación, deberemos utilizar un componente “Divide Curve” (Curve/Division/Divide Curve), para dividir las dos curvas en 20 segmentos, en última instancia, nos darán 21 puntos en cada curva (pero una lista de 42 puntos). Ahora conectemos todos los puntos a un componente Polilínea (Curve/Spline/Polyline), que creará una nueva línea a través de los puntos de división.



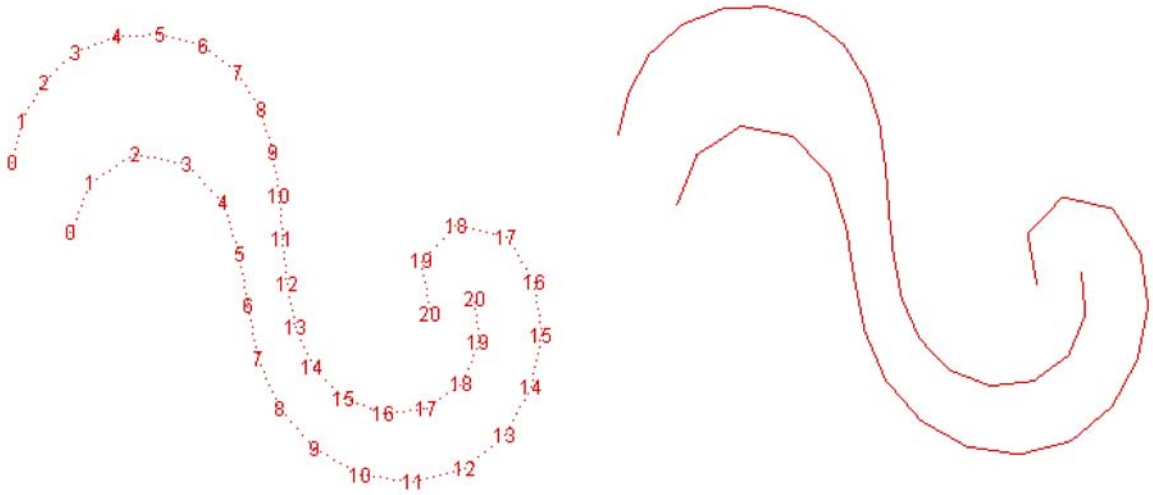
Si hubiéramos estado utilizando la versión de Grasshopper 0.5 o anterior, la curva de la polilínea se basaría sólo en 1 línea a través de nuestra lista de puntos (que tiene 42 puntos como resultado del componente “Divide Curve”). Esto es así porque todos los puntos se almacenan en la lista 1 (sin ramas o caminos), de modo que el componente está programado para dibujar una polilínea a través de todos los puntos de la lista. Si se utiliza la versión 0.5 de Grasshopper, el índice del punto y la polilínea resultante tendría este aspecto:



Sin embargo, desde ahora sabemos que Grasshopper tiene la capacidad de incorporar los caminos y las ramas, y ahora podemos utilizar dicho índice para controlar cómo se comporta el componente Polilínea. Si seguimos los mismos pasos que antes, sólo utilizando la versión Grasshopper 0.6.00xx o superior, nuestro componente Polilínea creará 2 polilíneas porque reconoce que hay 2 caminos y que cada uno ha sido dividido en 20 segmentos. Podemos comprobar la estructura del árbol mediante el Visor de parámetros (Params/Special/Param Viewer). A continuación se muestra una captura de pantalla de este ejemplo, lo que demuestra que nuestra estructura cuenta con 2 vías (0, 0) y (0, 1), y cada uno tiene 1 curva resultante en su terminal.

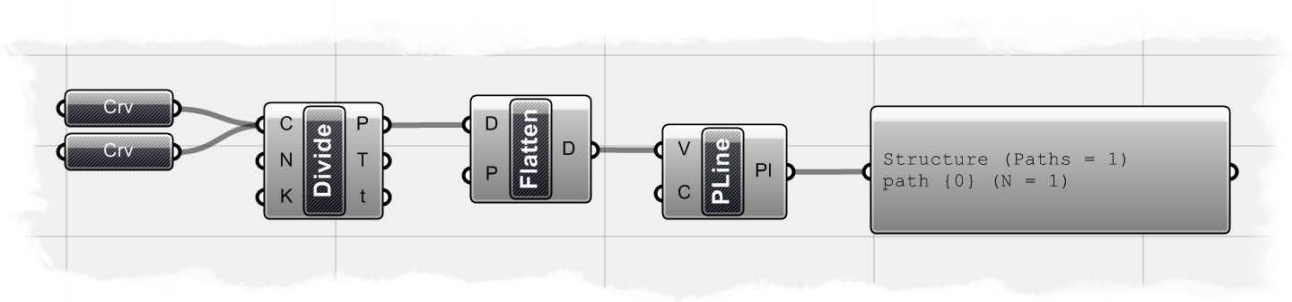


El índice del punto de las 2 polilíneas y las curvas resultantes deben parecerse a esto:

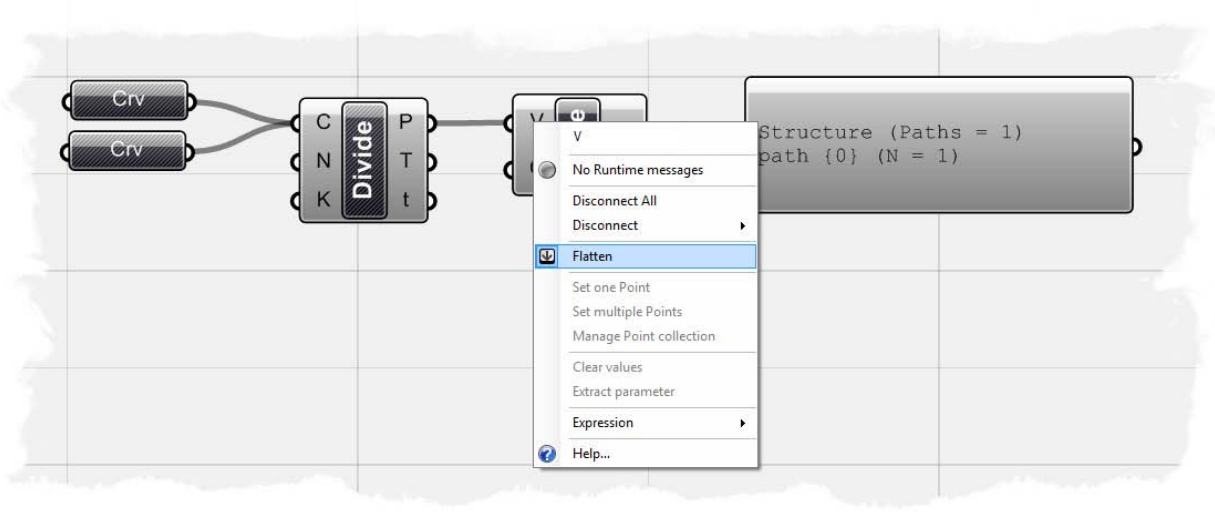


Ahora sabemos que nuestra definición tiene 2 caminos, pero sucedería si realmente sólo queremos una vía y por lo que una polilínea resultante. Grasshopper tiene un número de componentes de la nueva estructura de árbol, encontrado en la subcategoría de la ficha Árbol de “Logics”, que puede ayudarle a controlar su estructura de árbol. Podemos usar un componente “Flatten Tree” (Logic/Tree/Flatten Tree) para cerrar una estructura de árbol en una sola lista, al igual que los datos que han sido expuestas en la versión 0.5 o anterior.

Usted puede ver en la imagen, que cuando se aplasta nuestro árbol, nuestra estructura sólo tiene un camino, con un curva resultante almacenada en la terminal de... en última instancia, nos da la misma curva de polilínea única que habría tenido si se hubiera utilizado una versión anterior de Grasshopper.



Usted también puede simplificar esta definición mediante el aplanamiento de su estructura dentro del componente Polilínea, y pasando por alto el componente de aplanar “Flatten”. Para hacer esto, simplemente haga clic derecho en la entrada V del componente “Polyline” y seleccione la opción " Flatten ". Al hacer esto, usted debe notar que la estructura de árbol (que podemos ver mediante el Visor de Parámetros) cambia de 2 vías a 1.



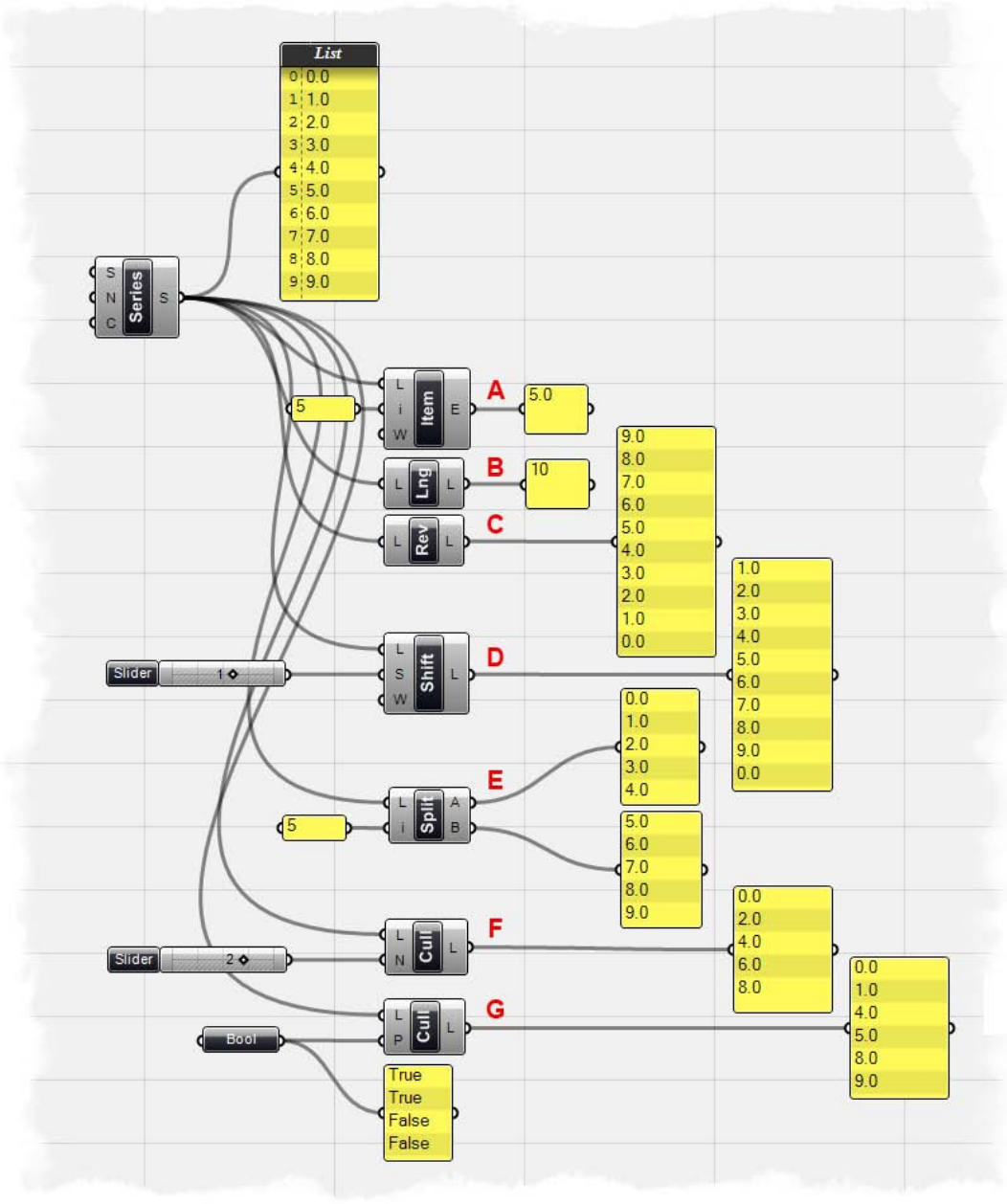
Nota: Para ver la definición final utilizada para crear este ejemplo, abra el archivo **Curve Paths.ghx** encuentrado en la carpeta “Archivos de apoyo” que acompaña a este documento. Usted también tendrá que abrir el archivo para Rhino que se encuentra en la misma carpeta llamado **Curve Paths_base file.3dm.**

8.1 Listas & Gestión de datos

Es útil pensar en Grasshopper en términos de flujos de datos, ya que la interfaz gráfica está diseñada para contener flujos de información dentro y fuera de determinados tipos de componentes. Sin embargo, son los datos (como las listas de puntos, curvas, superficies, cadenas, booleanos, números, etc.) los que definen la información que entra y sale de los componentes, de tal manera que la comprensión de cómo manipular los datos de la lista es fundamental para la comprensión del plug-in Grasshopper.

A continuación se muestra un ejemplo de cómo se puede controlar una lista de datos numéricos utilizando una variedad de componentes de la lista encontrado en la subcategoría de la ficha lógicas.

Nota: Para ver la definición final del ejemplo a continuación, abra el archivo **List Management.ghx** encontrado en la carpeta “Archivos de apoyo” que acompaña a este documento.



Para empezar, hemos creado un componente “Serie”, con un valor inicial de 0,0, un valor de espaciado de 1,0, y un recuento de 10. Así pues, el panel “Post-it” conectado a la salida S de “Serie” muestra la siguiente lista de números: 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0 y 9.0.

Nota: La configuración por defecto del panel “Post-it” muestra el número índice de la lista antes que los datos numéricos, de modo que cada entrada del panel muestra lo siguiente:

Estos se pueden activar o desactivar haciendo clic derecho en el panel “Post-it” y cambiando la vista previa, a encendido o apagado. Sin embargo, dejaremos los números de entrada activada para este ejemplo, ya que nos permitirá ver exactamente cual numero índice de la lista está asignado a cada valor.

A) Los datos numéricos se conectan dentro de un componente “**List Item**” (Logic/Lista/Lista de objetos), que es utilizado para recuperar una entrada específica en la lista. Al acceder a los elementos individuales en una lista, uno tiene que usar un valor de índice. El primer elemento de cualquier lista siempre se almacena en la posición cero, el segundo elemento en la posición 1 y así sucesivamente. Normalmente, si usted comienza a acceder a una lista en el índice -5, se producirá un error, ya que no existe tal lugar. Hemos conectado la salida S de “Serie” dentro de la entrada L de “List Item”. Adicionalmente, hemos conectado un número entero en la entrada I de “List Item”, con el cual definimos que número índices de la lista queremos recuperar. Dado que hemos fijado este valor a 5.0, la salida de “List Item” muestra los datos numéricos asociados con el número 5 de la entrada, que en este caso es también 5.0.

B) El componente “**List Length**” (Logic/List/List Length), esencialmente evalúa el número de entradas en la lista y entrega el último número de la entrada, o la longitud de la lista. En este ejemplo, hemos conectado la salida de S de “Serie” a la entrada L de “List Length”, lo que muestra que hay 10 valores en la lista.

C) Podemos invertir el orden de la lista mediante un componente “**Reverse List**” (Logic/List/Reverse List). Hemos de ingresado una lista ascendente de números en el componente “Reverse List”, con lo cual la salida muestra una lista descendente desde 9,0 hasta 0,0.

D) El componente “**Shift List**” (Logic/List/Shift List) o bien mueve la lista hacia arriba o hacia abajo por una serie de incrementos que dependen del valor que le demos al componente. Hemos conectado la salida S de “Serie” a la salida L “Shift List”, mientras que

conectamos un control deslizante numérico para la entrada S de “Shift List”. Hemos fijado el tipo de control deslizante para números enteros de modo que el cambio del desplazamiento se producirá en números enteros. Si se establece el control deslizante a -1, todos los valores de la lista se moverán hacia abajo por un número de entrada. Del mismo modo, si se cambia el valor del control deslizante a +1, todos los valores de la lista se desplazarán hacia arriba por un número de entrada. También se puede establecer el valor de ajuste, o la entrada W de “Shift List”, ya sea Verdadero o Falso haciendo clic derecho en la entrada y escogiendo “Set Boolean”. En este ejemplo, tenemos un valor de cambio establecido en 1, así que tenemos que tomar una decisión sobre cómo queremos tratar el primer valor. Si nos fijamos el valor de ajuste en Verdadero, la primera entrada se moverá a la parte inferior de la lista. Sin embargo, si nos fijamos el valor de ajuste en Falso, la primera entrada se desplazará hacia arriba y fuera de la lista, en esencia, eliminando este valor del conjunto de datos.

E) El componente “**Split List**” divide una lista en dos listas más pequeñas. Tiene un índice de división, que es simplemente un número entero por el cual la lista es dividida. En este caso, hemos ajustado al componente “Split List” para dividir la lista después del elemento de índice 5, con lo que nuestra lista se verá así: Lista A - 0.0, 1.0, 2.0, 3.0, 4.0 Lista B - 5.0, 6.0, 7.0, 8.0, 9.0.

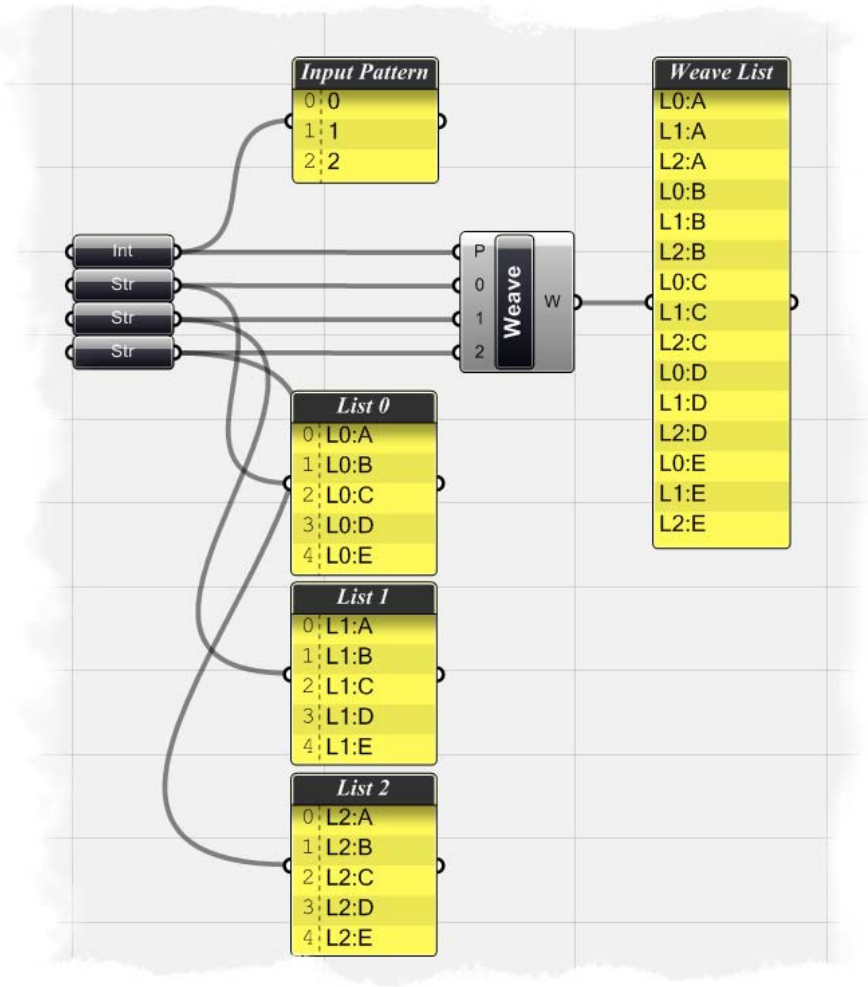
F) El componente “**Cull Nth**” (Logic/Sets/Cull Nth) eliminará todos los datos de entradas enésimas de la lista, donde N es definido por un número entero. En este ejemplo, hemos conectado un control deslizante numérico a la entrada N de “Cull Nth”. Hemos establecido nuestro control deslizante a 2,0, de manera que el componente “Cull Nth” eliminará todas las otras entradas de la lista. La salida L de “Cull Nth” muestra una lista de desecho en el que cada nueva entrada impar se ha suprimido: 0,0, 2,0, 4,0, 6,0 y 8,0. Si cambiamos el regulador numérico a 3,0, el componente “Cull Nth” eliminará cada tercer número de la lista para lo cual el resultado sería: 0.0, 1.0, 3.0, 4.0, 6.0, 7.0 y 9.0.

G) El componente “**Cull Pattern**” (Logic/Sets/Cull Pattern) es similar al componente “Cull Nth”, ya que elimina los elementos de una lista basada en un valor determinado. Sin embargo, en este caso, se utiliza un conjunto de valores booleanos que forman un patrón, en lugar de valores numéricos. Si el valor booleano se establece en Verdadero, la entrada de datos se mantendrá en la lista, mientras que un valor Falso eliminará la entrada de datos del conjunto. En este ejemplo, hemos establecido el patrón booleano a: Verdadero, Verdadero, Falso, Falso. Dado que hay sólo 4 valores booleanos y nuestra lista tiene 10 entradas, el patrón se repetirá hasta llegar al final de la lista. Con este patrón, la lista de salida será: 0.0, 1.0, 4.0, 5.0, 8.0 y 9.0. El componente “Cull Pattern” ha mantenido las dos primeras entradas (0,0 y 1,0) y luego remueve los próximos dos valores (2,0 y 3,0). El componente ha seguido este patrón hasta llegar a la final de la lista.

8.2 Datos entrelazados

En la sección pasada, explicamos cómo se puede utilizar una serie de diferentes componentes para controlar cómo se gestionan las listas de Grasshopper, pero también podemos utilizar el componente "**Weave**" para controlar el orden de una lista. El componente "Weave" se puede encontrar bajo la subcategoría de las listas de la Ficha de Lógicas (Logics/Lists/Weave).

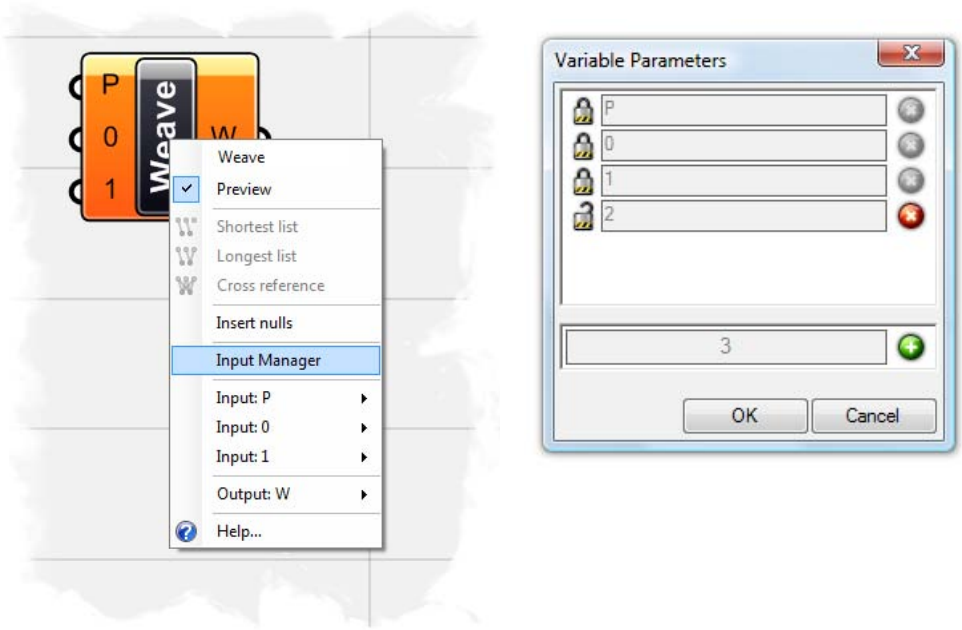
Nota: Para ver la definición final a continuación, abra el archivo **Weave Pattern.ghx** encontrado en la carpeta "Archivos de apoyo" que acompaña a este documento.



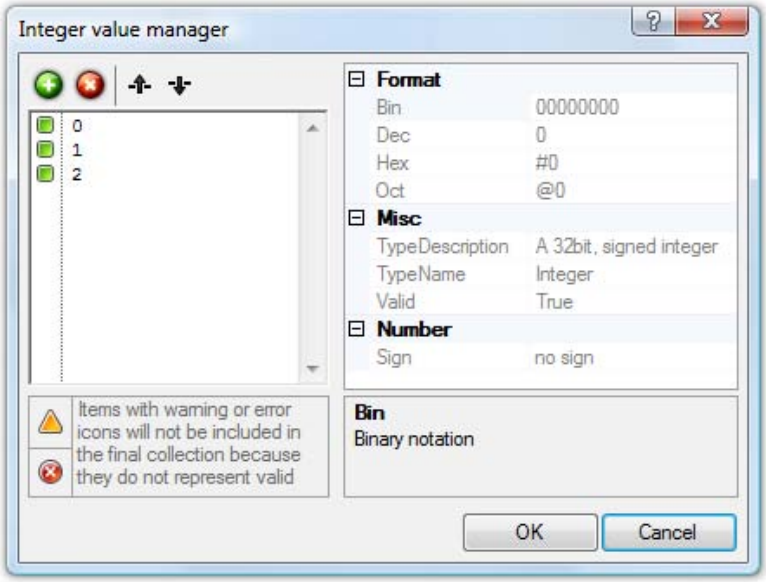
Para crear la definición desde el inicio:

- Logic/List/Weave - Arrastre y suelte un componente "Weave" sobre el lienzo.

  Usted debe ver inmediatamente a 3 puntos de entrada. El primero, o la entrada P, es el patrón de tejido y esto determinará el orden en el que tejemos nuestros datos. Los siguientes dos elementos de entrada, la etiqueta 0 y 1, respectivamente, permitirán la entrada de dos listas para tejerlas juntas. Pero, ¿y si queremos tejer más de dos listas? Bueno, si pulsa el botón derecho en el medio del componente "Weave", puede abrir el "Input Manager" y añadir tantas listas como sea necesario. Cuando el "Input Manager" se abre, haga clic en el botón "+" verde para agregar otra lista. Del mismo modo, la "X" roja al lado de cada lista eliminará de la lista del componente.



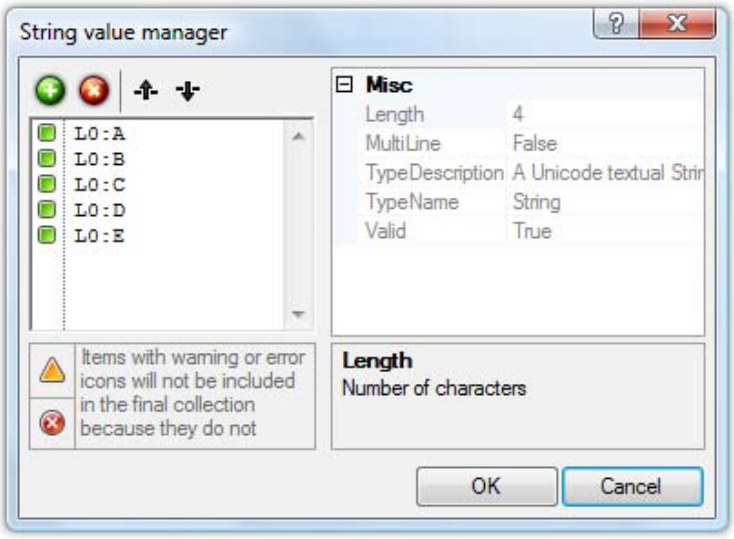
- Abra el “Input Manager” y seleccione el botón verde “+” para agregar una lista adicional para el componente. Su “Input Manager” debe parecerse a la imagen de arriba.
- Params/Primitive/Integer - Arrastre y suelte un componente “Integer” sobre el lienzo.
- Clic derecho sobre el “Integer” y seleccione "Manage Integer Collection"
- Al hacer clic en el botón verde “+” en la parte superior, se puede añadir un número entero en la colección. Añada tres números a la colección y cambie de cada valor, de modo que su lista sea así: 0, 1, 2. Su administrador debería ser similar a la imagen de abajo.



Esta colección de números enteros determinará nuestro patrón de tejido. Al cambiar el orden de estos números, podemos cambiar rápidamente el orden de nuestro conjunto de datos.

- Params/Primitive/String - Arrastre y suelte un componente “String” sobre el lienzo
- Clic derecho sobre “String” y seleccione "Manage String Collection"

Al igual que hicimos con la colección de números enteros, vamos a añadir una lista de cadenas que queremos tejer juntas. Esta será nuestra primera lista de datos, pero vamos a copiar y pegar esta lista dos veces más, de modo que tenemos tres listas para tejer. Añada 5 “strings” a la colección de manera que la lista se vea así: L0: A, L0: B, L0: C, L0: D, L0: E. El prefijo L0: sólo nos dice que estas cadenas se encuentran en la lista cero y nos ayudará a ser capaces de rastrear los datos una vez que se tejen juntos. Su “String Manager” debe ser similar a la imagen de abajo.



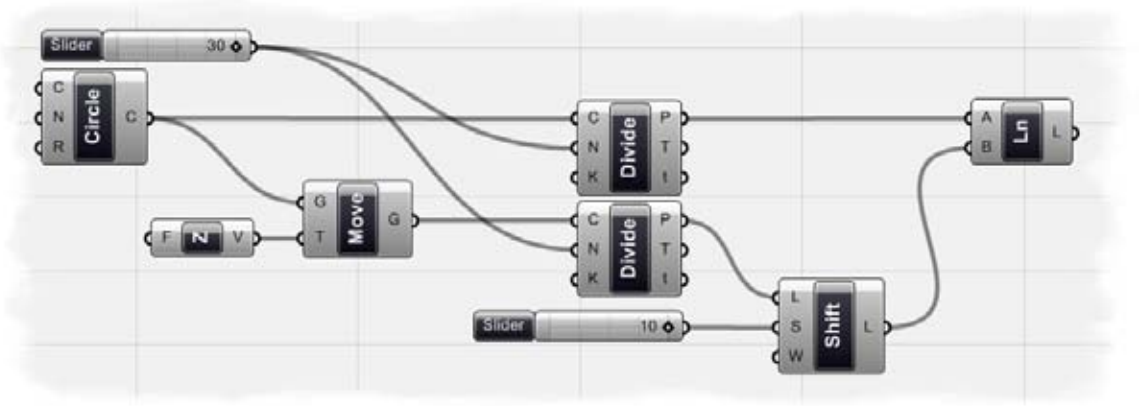
- Seleccione el parámetro “string” y pulse Ctrl+C (copiar) y Ctrl+V (pegar) para duplicar **dos** parámetros “string” más al lienzo.
- Haga clic derecho sobre el segundo parámetro “String” y abria el “String Collection Manager”. Cambie su colección de cadena para que la lista se vea así: L1:A, L1:B, L1:C, L1:D, L1:E.
- Haga clic derecho sobre el tercer parámetro “String” y abria el “String Collection Manager”. Cambie su colección de cadena para que la lista se vea así: L2:A, L2:B, L2:C, L2:D, L2:E.
- Ahora, conecte el parámetro “Integer” - el que va a definir nuestro modelo de tejido - a la entrada P del componente “Weave”.
- Conecte el primer parámetro “String” a la entrada 0 del componente de “Weave”.
- Conecte el segundo parámetro “String” a la entrada 1 del componente de “Weave”.
- Conecte el tercer parámetro “String” a la entrada 2 del componente de “Weave”.
- Params/Special/Post-it Panel - Arrastre y suelte un panel “Post-it” sobre el lienzo.
- Conecte la salida W de “Weave” al panel “Post-it” de modo que podamos ver los datos.

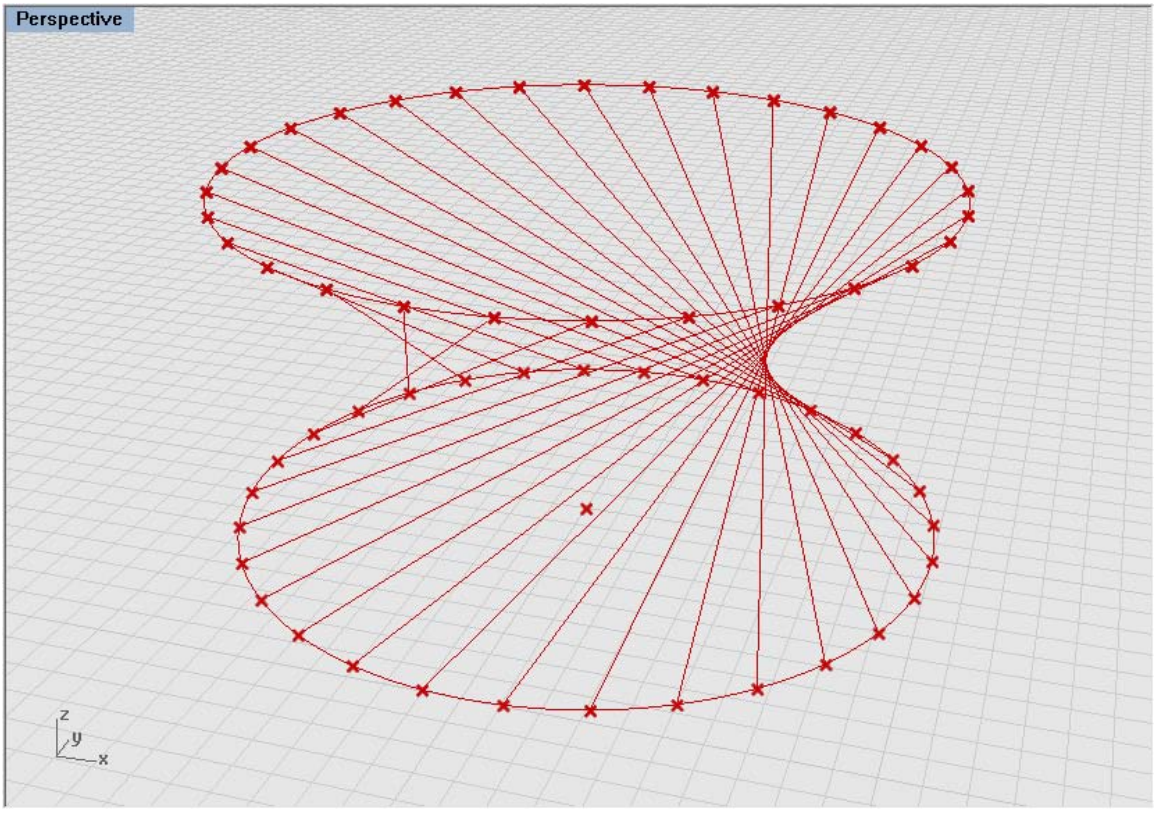
El panel “Post-it” debe ahora mostrar una lista de datos que se han tejido en conjunto de acuerdo a nuestra colección de números enteros. El primer elemento de la lista debería ser el primer elemento de la lista 0. En consecuencia, el segundo elemento de la lista será el primer elemento de la lista 1. Y así sucesivamente, hasta llegar al final de la lista. Pruebe a cambiar el orden de la colección para cambiar el orden de la lista de tejido.

8.3 Datos variables

Hemos discutido en la sección 8.1 cómo podemos utilizar el componente “Shift” para mover todos los valores de una lista hacia arriba o hacia abajo dependiendo de un cambio en el desfase de valores. A continuación se muestra un ejemplo, creado por David Rutten, para demostrar cómo podemos usar el componente “Shift” en dos series de listas de puntos de un círculo. Usted puede encontrar más información sobre este ejemplo aquí:
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryShiftExample.html

Nota: Para ver la versión final de esta definición, abra el archivo **Shift Circle.ghx** que se encuentra en la carpeta “Archivos de apoyo” que acompaña a este documento. A continuación se muestra una captura de pantalla de la definición completa.





Para crear la definición desde el inicio:

- Curve/Primitive/Circle CNR - Arrastre y suelte un “Circle CNR” (Centro, Normal, y Radio) sobre el lienzo.
- Clicl derecho sobre la entrada C de “Circle” y seleccione “Set One Point”.
- En el cuadro de diálogo de Rhino, escriba "0,0,0" y pulse enter.
- Clicl derecho sobre la entrada C de “Circle” y determine el punto en 10.0
- Vector/Constants/Unit Z - Arrastre y suelte un componente “Unit Z” sobre el lienzo.
- Clicl derecho sobre la entrada F de “Unit Z” y determine el punto en 10.0
- X Form/Euclidean/Move - Arrastre y suelte un componente “Move” sobre el lienzo.
- Conecte la salida V de “Unit Z” a la entrada T de “Move”.
- Conecte la salida C de “Circle” a la entrada G de “Move”.

  Acabamos de crear un círculo cuyo centro es el punto 0,0,0 y tiene un radio de 10,0 unidades. A continuación, utilizamos el componente “Move” para copiar este círculo y mover el círculo duplicado en 10,0 unidades en el eje Z.
- Curve/Division/Divide Curve - Arrastre y suelte **dos** componentes “Divide Curve” sobre el lienzo.
- Conecte la salida C de “Circle” a la entrada C del primer “Divide Curve”.
- Conecte la salida G de “Move” a la entrada C del segundo “Divide Curve”.
- Params/Special/Slider - Arrastre y suelte un “Numeric slider” sobre el lienzo.
- Seleccione el deslizador y determine lo siguiente:
  - Slider Type: Integers
  - Lower Limit: 1.0
  - Upper Limit: 30.0
  - Value: 30.0
- Conecte el deslizador a la entrada N de ambos componentes “Divide Curve”.
  Debería ver ahora 30 puntos uniformemente espaciados a lo largo de cada círculo
- Logic/List/Shift List - Arrastre y suelte un componente “Shift List” sobre el lienzo.
- Conecte la salida P del segundo “Divide Curve” a la enrtada L de “Shift List”.
- Params/Special/Slider - Arrastre y suelte otro “Numeric slider” sobre el lienzo.
- Seleccione el nuevo deslizador y determine lo siguiente:
  - Slider Type: Integers
  - Lower Limit: -10.0
  - Upper Limit: 10.0
  - Value: 10.0
- Conecte la salida del deslizador a la entrada S de “Shift List”.
- Clic derecho sobre la entrada W de “Shift List” y ajuste el “boolean value” a True.

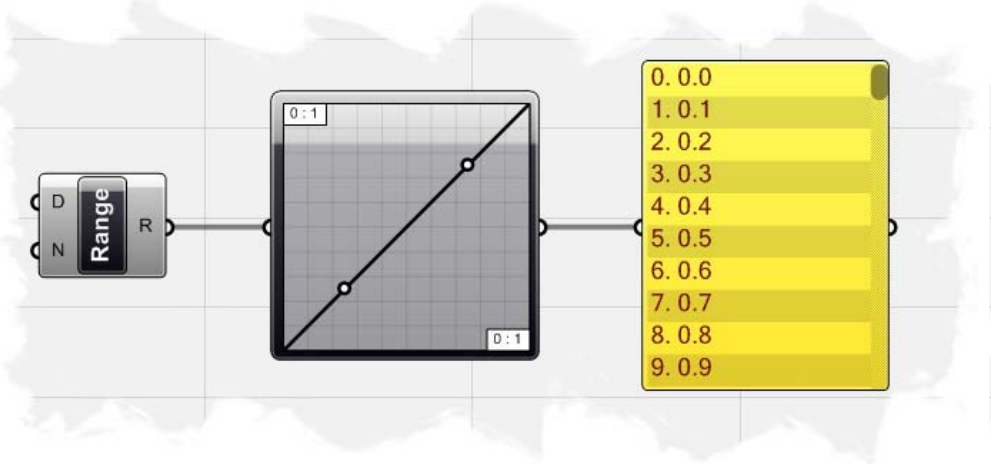
  Hemos cambiado los puntos en el círculo superior en 10 unidades en su lista de índices. Al establecer el valor de ajuste en True, hemos creado un circuito de entradas de datos cerrado.
- Curve/Primitive/Line - Arrastre y suelte un componente “Line” sobre el lienzo.
- Conecte la salida P del primer “Divide Curve”a la entrada A de “Line”.
- Conecte la salida L de “Shift” a la entrada B de “Line”.

  Hemos creado una serie de segmentos de línea que conectan las listas de puntos en el círculo de abajo con el conjunto de puntos desplazado en el círculo superior. Podemos cambiar el valor del deslizador numérico que controla el desplazamiento para ver el cambio de línea entre los segmentos de la lista de puntos.

8.4 Exportación de datos a Excel

Hay muchos casos en que puede ser necesario exportar los datos de Grasshopper para importar la información a otro software para posteriores análisis.

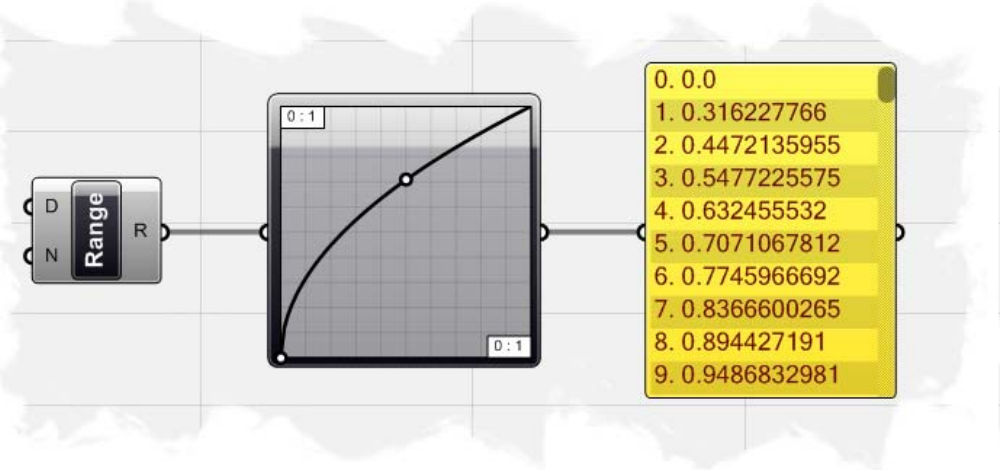
Nota: Para ver la versión final de este ejemplo, abra el archivo **Stream Contents_Excel.ghx** que se encuentra en la carpeta “Archivos de apoyo” que acompaña a este documento.



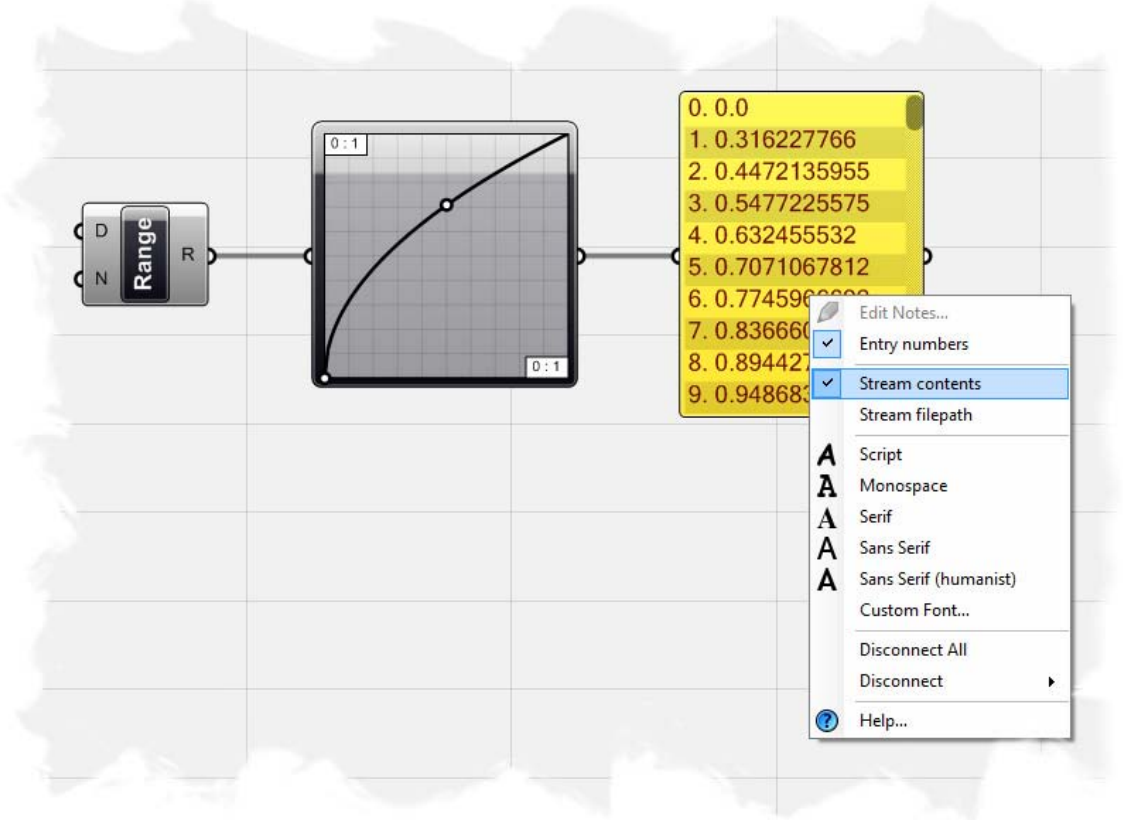
Para empezar, hemos añadido un componente “**Range**” (Logic/Sets/Range), sobre el lienzo, y hemos establecido el dominio numérico de 0,0 a 10,0. Haciendo clic derecho sobre la entrada N de “Range”, hemos fijado el número de pasos a 100, para que nuestra lista de salida muestre 101 valores equidistantes entre 0,0 y 10,0.

A continuación, arrastre y suelte un componente “**Graph Mapper**” (Params/Special/Graph Mapper) sobre el lienzo. Haga clic en el componente “Graph Mapper” y establezca el tipo de gráfico a “**Linear**”. Ahora conecte la salida R de “Range” a la entrada del “Graph Mapper”. Para terminar la definición, arrastre y suelte un panel “**Post-it**” sobre el lienzo y conecte la salida del “Graph Mapper” a la entrada del panel “Post-it”.

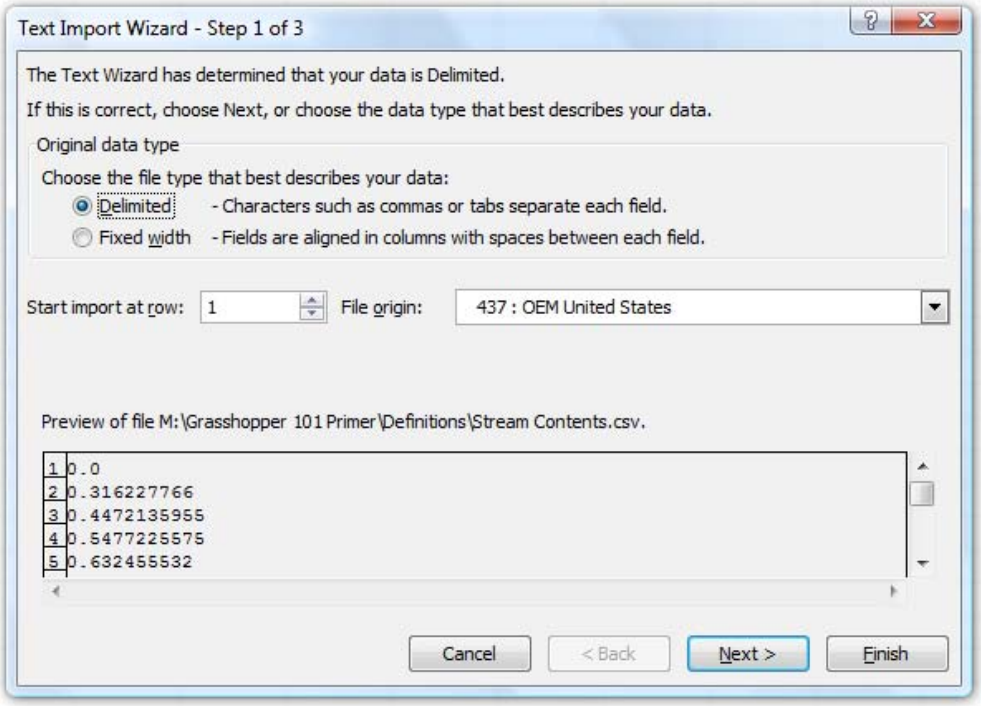
Dado que el tipo de “Graph Mapper” se estableció “Linear”, la lista de salida (que se muestra en el panel “Post-it”) muestra un conjunto de datos numéricos, que sube de 0,0 a 10,0 en forma lineal. Sin embargo, si hacemos clic derecho en el componente de “Graph Mapper” y establecemos el tipo de gráfico a “**Square Root**” (Raíz Cuadrada), deberíamos ver una nueva lista de datos que representa una función logarítmica.



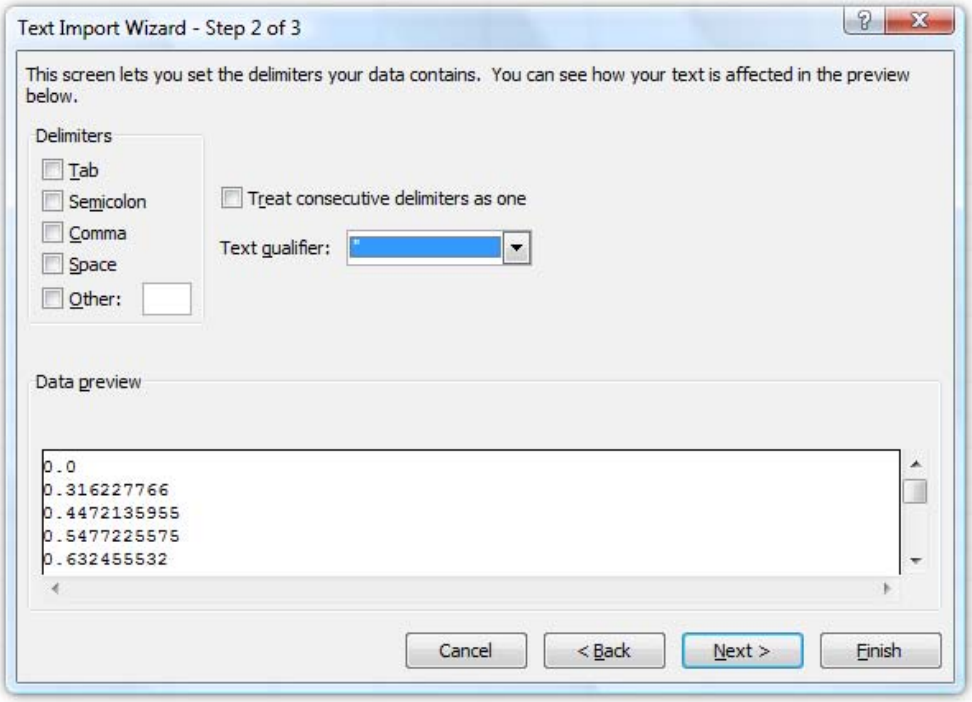
Para exportar la lista de datos desde el panel “Post-it”, simplemente haga clic derecho en el panel y seleccione “**Stream Contents**”. Cuando se le solicite, encuentre una ubicación en su disco duro para guardar el archivo con un nombre de archivo único. En nuestro ejemplo, vamos a guardar el archivo en la siguiente ubicación: C:/Tutorials/Exporting Data/Stream_Contents.csv. Hay una gran variedad de tipos de archivo que se puede utilizar para guardar los datos, incluyendo archivos de texto (.txt), valores separados por comas (.csv), y archivos de datos (.dat), por nombrar unos pocos. Yo tiendo a usar el formato de archivo de valores separados por comas porque fueron diseñados para el almacenamiento de datos estructurados en forma de tabla, aunque muchos de los formatos de archivo se pueden importar en Excel. Cada línea en el archivo CSV corresponde a una fila en la tabla. Dentro de una línea, los campos están separados por comas, cada campo perteneciente a una columna de la tabla. Nuestro ejemplo sólo tiene un valor por línea, por lo que no utilizará el aspecto de mutli-columna disponible con este formato de archivo, pero es posible crear hojas de cálculo complejas mediante la exportación de los datos correctamente.



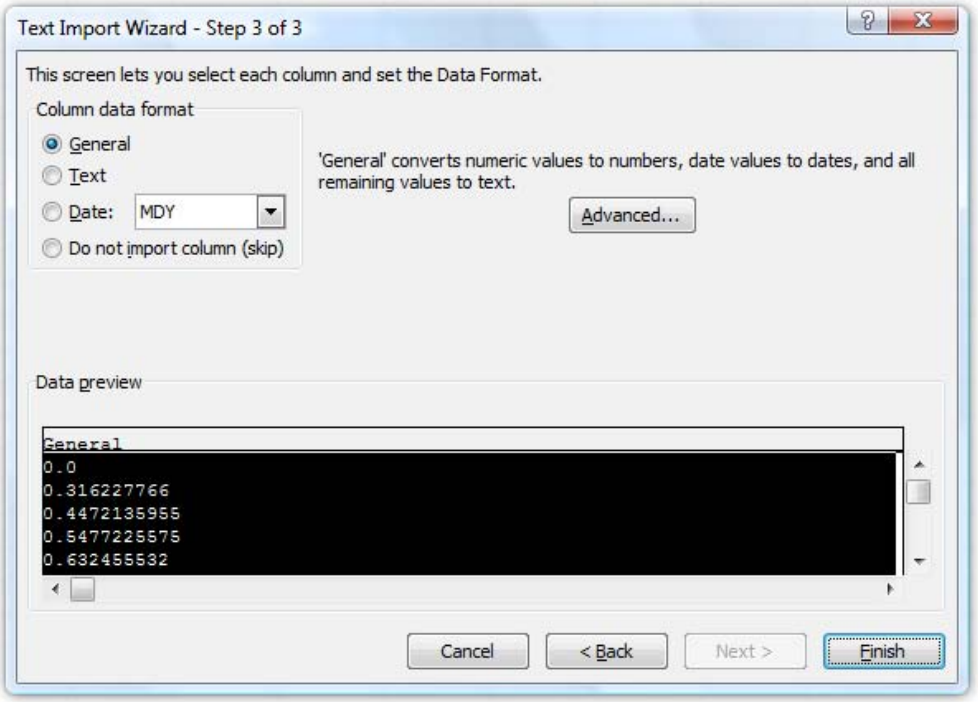
Ahora podemos importar los datos en Microsoft Excel 2007 o superior. Primero abra la aplicación y seleccione la ficha Datos. Seleccione “**Get External Data from Text**” en esta ficha y encuentre el archivo Stream_Contents. que guardó en su disco duro. Ahora, será guiado por el “**Text Import Wizard**”, donde se le pedirá una serie de preguntas sobre cómo desea importar los datos. Asegúrese de que el botón radial “**Delimited**” está marcado y seleccione “Next” para proceder con el Paso 2.



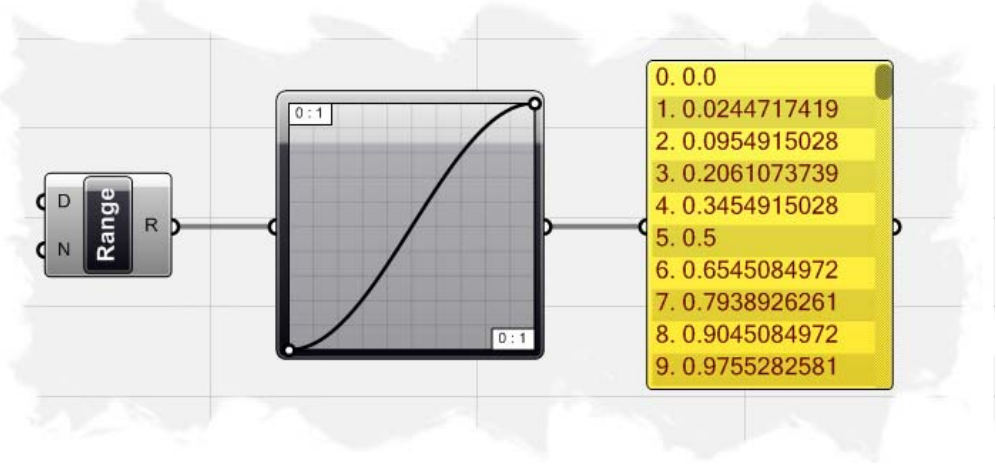
El paso 2 del “Text Import Wizard” le permite definir el tipo de delimitadores que definirá cómo se separan los datos. Un delimitador es un personaje (como un punto y coma, coma o espacio) almacenados en el archivo CSV que indica que los datos deben dividirse en otra columna. Dado que sólo disponemos de datos numéricos almacenados en cada línea del archivo CSV, que no es necesario seleccionar un delimitador específico. Seleccione “Next” para continuar con el paso 3.



El paso 3 del “Text Import Wizard” le permite indicar a Excel cómo le gustaría dar formato a los datos en Excel. “General” convierte los datos numéricos a números; “Date” convierte todos los valores a fechas (Día/Mes/Año), y todos los valores restantes tienen formato de datos de texto. Para nuestro ejemplo, seleccione “General” y pulse Finalizar.

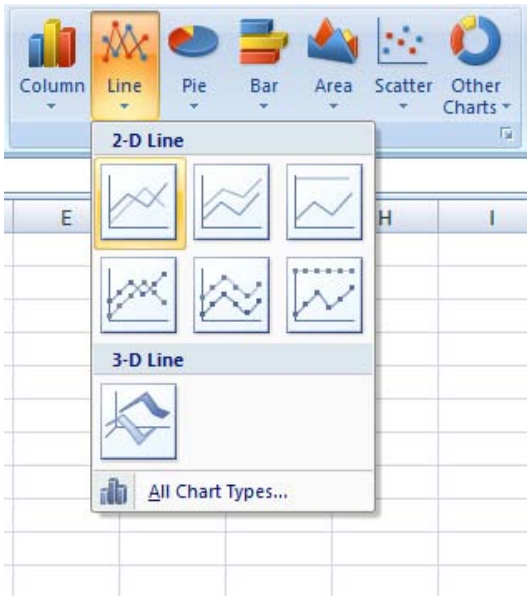


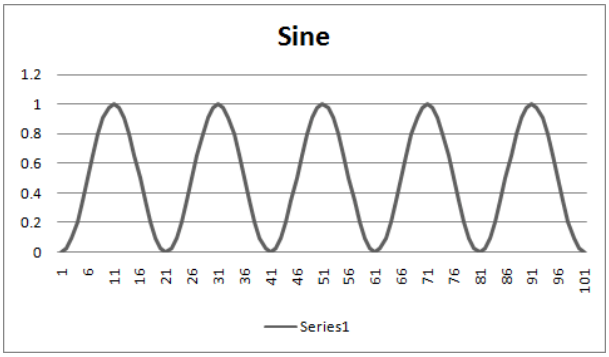
Ahora se le preguntará a qué celda desea utilizar para empezar a importar sus datos. Vamos a utilizar el valor predeterminado de la celda A1. Ahora verá los 101 valores de la columna A, que corresponden a los valores dentro del panel “Post-it” de Grasshopper. La definición de Grasshopper está constantemente transfiriendo los datos, de modo que cualquier cambio que hagamos a los datos de la lista se actualizará automáticamente en el archivo CSV. Vuelva a Grasshopper y cambie el tipo de gráfico a “**Sine**”. Note el cambio de datos en el panel “Post-it”.



Cambie de nuevo a Microsoft Excel y en la ficha de datos, podrá ver otro botón que dice “**Refresh All**”. Seleccione este botón, y cuando se le pida, seleccione el archivo CSV que previamente se cargó en Excel. Los datos de la lista en la columna A serán actualizados.

Ahora seleccione todas las celdas desde la A1 hasta la A101 (seleccione la A1 y mientras mantiene apretado el botón “Shift”, seleccione la A101) y haga clic en la ficha Insertar en la parte superior. Seleccione el tipo de de gráfico “**Line Chart**” y seleccione el icono de gráfico “2D line”.

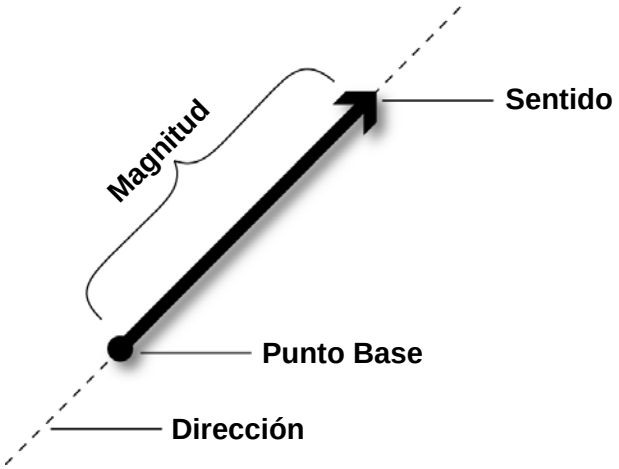




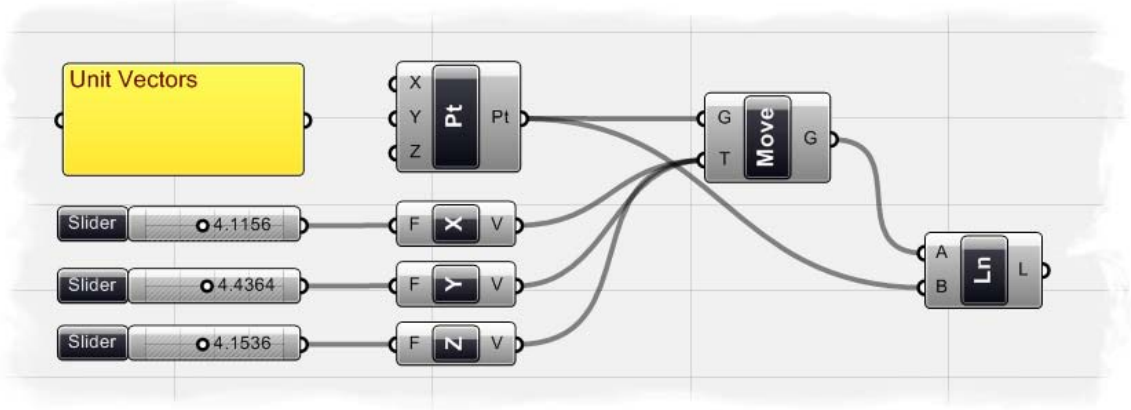
Usted verá un gráfico de línea que refleja la misma forma que se muestra en el componente del gráfico “Graph Mapper” en Grasshopper. Usted puede hacer tantos cambios como desee dentro de Grasshopper, y después actualizarlos, podrá ver los cambios reflejados en Excel.

9 *Vectores basicos*

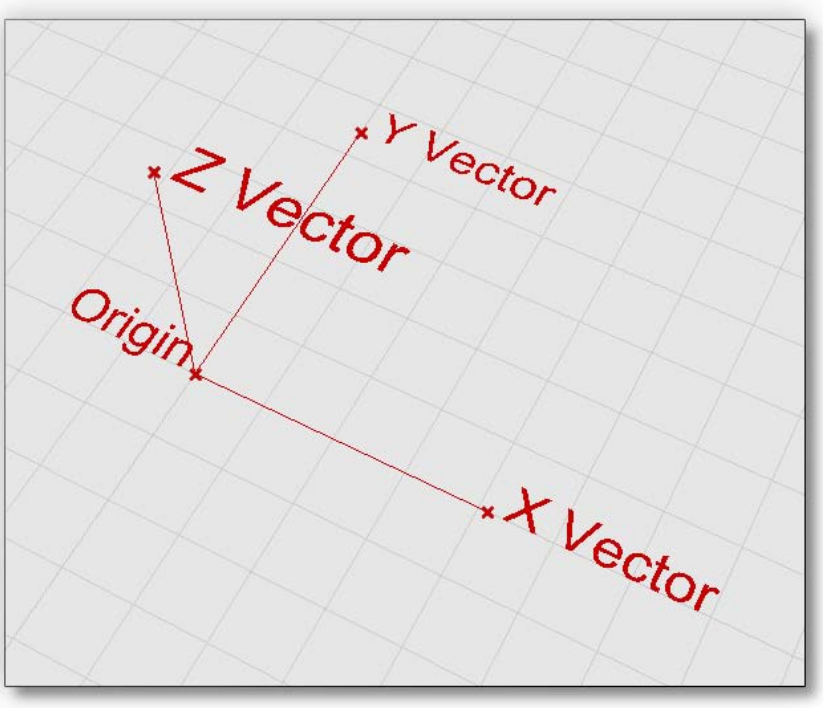
Desde la física, sabemos que un vector es un objeto geométrico que tiene una magnitud (o longitud), dirección y sentido (o la orientación a lo largo de la dirección dada). Un vector es a menudo representado por un segmento de línea con una dirección definida (a menudo representado como una flecha), conectando un punto base A con un punto terminal B. La magnitud (o amplitud) del vector es la longitud del segmento y la dirección caracteriza el desplazamiento de B respecto a A: ¿Cuánto se debe mover el punto A para llegar hasta el punto B?.



En Rhino, los vectores son indistinguibles de los puntos. Donde cada punto (o variable numérica que puede almacenar números con decimales) representando las coordenadas X, Y y Z. en el espacio cartesiano. La diferencia es que los puntos son tratados como absolutos, mientras que los vectores son relativos. Cuando tratamos con estas tres variables como un punto, estas representan una cierta coordinada en el espacio. Cuando tratamos a la matriz como un vector, esta representa una determinada dirección. Los vectores son considerados relativos, ya que sólo indican la diferencia entre los puntos inicial y final de la flecha, es decir que **los vectores no son entidades geométricas, son sólo información**. Esto significa que no hay ningún indicador visual del vector en Rhino, sin embargo, podemos utilizar la información vectorial para informar acciones específicas como la traducción geométrica, la rotación y la orientación.



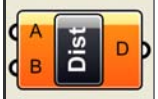
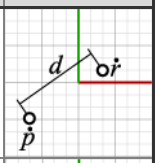
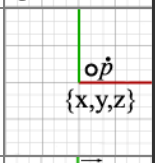
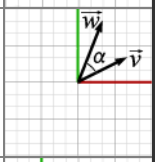
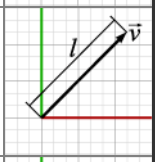
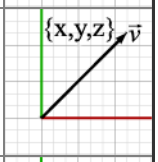
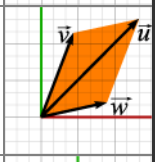
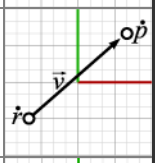
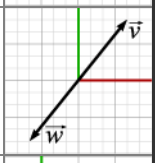
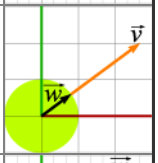
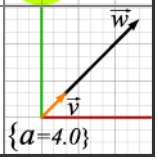
En el ejemplo siguiente, empezaremos por la creación de un punto en el origen 0,0,0 utilizando el componente de punto XYZ (Vector/Point/Point XYZ). A continuación, conectaremos la salida Pt de “Point” a la entrada G de “Move” para trasladar una copia del punto en una dirección del vector. Para ello, debemos de arrastrar y soltar los componentes “Unit X”, “Unit Y”, y “Unit Z” sobre el lienzo (Vector/Constants). Estos componentes especifican una dirección de vector en una de las direcciones ortogonales de los ejes X, Y, o Z. Podemos especificar la magnitud del vector mediante la conexión de un control deslizante numérico con la entrada de cada componente de vector unitario. Manteniendo pulsada la tecla “Shift” mientras se conectan las salidas de los “Unit Vector” con la entrada T de “Move”, así seremos capaces de conectar más de un componente. Ahora bien, si nos fijamos en la vista de Rhino, verá un punto en el origen, y tres nuevos puntos que se han movido en cada uno de los ejes X, Y, y Z. Usted podrá cambiar libremente el valor de cualquiera de los deslizadores numéricos para ver el cambio en la magnitud de cada vector. Para obtener un indicador visual de los vectores, de manera similar a la elaboración de una flecha, podemos crear un segmento de línea desde el punto de origen de cada uno de los puntos a trasladar. Para ello, arrastre y suelte un componente “Line” (Curve/Primitive/Line) en el lienzo. Conecte la salida de G de “Move” a la entrada A de “Line” y conecte la salida Pt de “Point” a la entrada B de “Line”. A continuación se muestra una captura de pantalla de la definición.



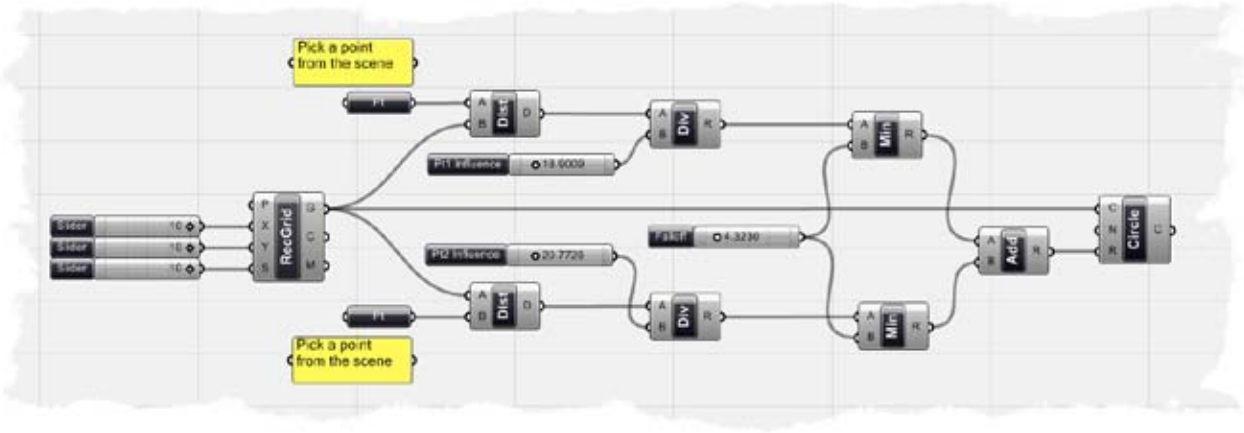
Nota: Para ver la versión final de esta definición, abra el archivo **Unit Vectors.ghx** que se encuentra en la carpeta “Archivos de apoyo” que acompaña a este documento.

9.1 Manipulacion Punto/Vector

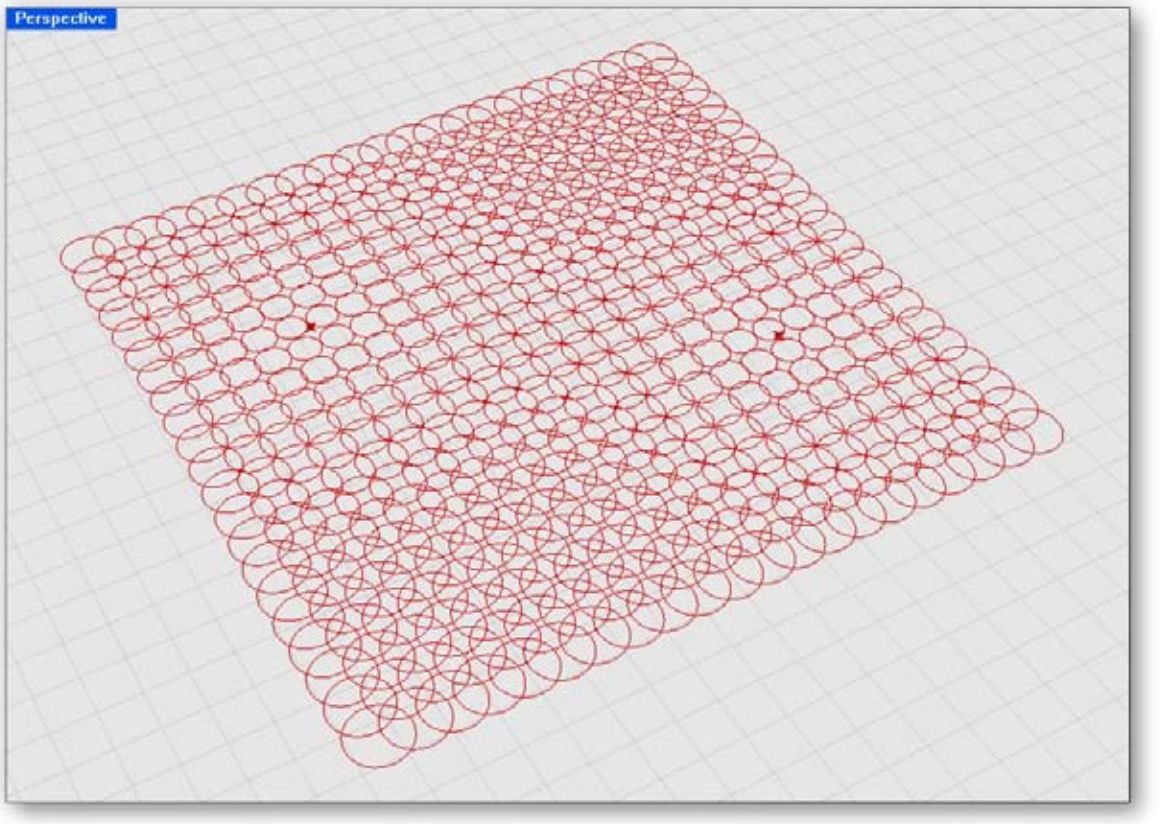
Grasshopper tiene todo un grupo de componentes “Point/Vector” que realizan las operaciones básicas de la "matemática de vectores". A continuación se muestra una tabla de los componentes más utilizados y sus funciones.

Componente	Ubicación	Descripción	Ejemplo
	Vector/Point/**Distance**	Calcula la distancia entre dos puntos (A y B como entradas)	
	Vector/Point/**Decompose**		
	Vector/Vector/**Angle**	Calcula el ángulo entre dos vectores Resultado es en Radianes	
	Vector/Vector/**Length**	Calcula la longitud (amplitud) de un vector	
	Vector/Vector/**Decompose**	Descompone un vector en sus componentes	
	Vector/Vector/**Summation**	Entrega un vector resultante sumando los componentes del vector 1 con los componentes del vector 2	
	Vector/Vector/**Vector2pt**	Crea un vector desde dos puntos definidos	
	Vector/Vector/**Reverse**	Invierte el sentido de un vector, manteniendo la misma longitud.	
	Vector/Vector/**Unit Vector**	Divide todos los componentes por la longitud inversa del vector. El vector resultante tiene una longitud de 1,0 y se denomina vector unidad. Algunas veces conocido como "normalización"	
	Vector/Vector/**Multiply**	Multiplica los componentes del vector por un factor especificado	

9.2 Usando Vectores/Escalamiento con puntos atractores (Circulos escalados)
Ahora que conocemos algunos de los fundamentos detrás de “Scalar” y “Vector mathematics”, Hechemos un vistazo a un ejemplo que escala una red de círculos de acuerdo a la distancia desde el centro del círculo a un punto determinado.



Nota: Para ver la versión final de esta ejemplo, abra el archivo **Attractor_2pt_circles.ghx** que se encuentra en la carpeta “Archivos de apoyo” que acompaña a este documento.
A continuación se muestra una captura de pantalla de la definición completa.



Para crear la definición desde el inicio:

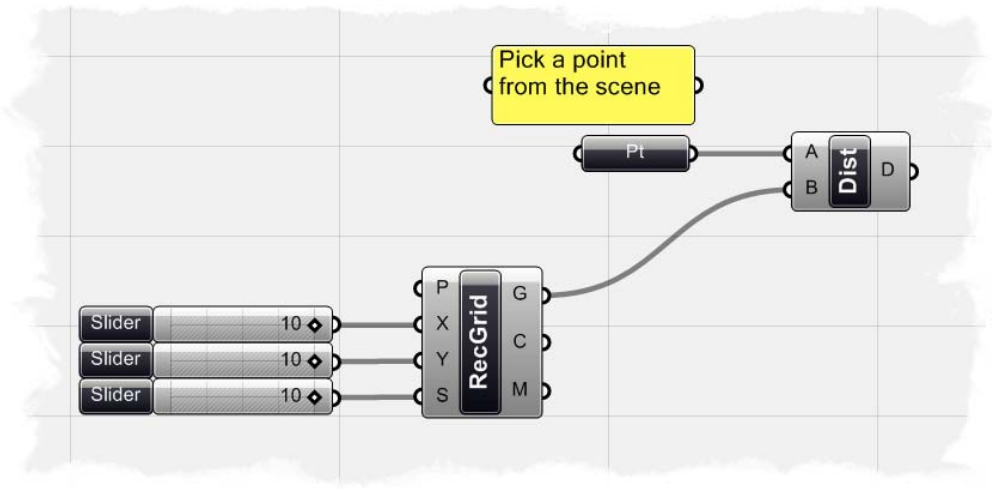
- Params/Special/Numeric Slider - Arrastre y suelte **tres** componentes “Numeric Slider” sobre el lienzo.
- Clic derecho sobre los tres deslizadores y determine lo siguiente:
  - Slider Type: Integers
  - Lower Limit: 0.0
  - Upper Limit: 10.0
  - Value: 10.0
- Vector/Point/Grid Rectangular - Arrastre y suelte un componente “Grid Rectangular” sobre el lienzo.
- Conecte el primer deslizador a la entrada X de “Pt Grid”.
- Conecte el segundo deslizador a la entrada Y de “Pt Grid”.
- Conecte el tercer deslizador a la entrada S de “Pt Grid”.

  El componente “Rectangular Point Grid” crea una red de puntos, donde la entrada P es el origen de la red (en nuestro caso usaremos 0,0,0). El componente crea una serie de puntos en los ejes X e Y especificados por los deslizadores numéricos. Sin embargo, note que hemos fijado ambos a 10,0. Si realmente contamos el número de filas y columnas, usted encontrará que hay 20 en cada dirección. Esto es debido a que la red se crea desde un punto central, y compensa el número de filas y columnas en cada dirección desde este punto. Así pues, en esencia, recibe el doble del número de puntos X e Y. La entrada S especifica el espacio entre cada punto.
- Params/Geometry/Point - Arrastre y suelte un componente “Point” sobre el lienzo

  Este componente se considera un componente implícito, ya que utiliza datos persistentes como su valor de entrada (véase el Capítulo 4 para obtener más información acerca de los tipos de datos persistentes). Este componente es diferente al otro componente “Point XYZ” que hemos usado antes, ya que no crea un punto hasta que haya elegido un punto de la escena. Para hacer esto, por supuesto, debe existir ya un punto en su escena de Rhino. **Este será nuestro punto de atracción.**
- En la escena de Rhino, escriba "Point" en el cuadro de diálogo y conloque un punto en cualquier lugar de la escena. En nuestro caso, vamos a colocar el punto en el visor Superior de modo que el punto esté en el plano XY.

  Ya que hemos creado un punto de atracción en la escena, podemos asignarlo al componente “Point” que acabamos de crear en Grasshopper.
- Clic derecho en el componente "Point" y seleccione "Set One Point".
- Cuando se le pida, seleccione el punto de atracción que ha creado en la escena de Rhino.

  Hemos creado una red de puntos y también los hemos referenciado en un punto de atracción de nuestra escena. Podemos usar un poco de matemáticas de vectores para determinar la distancia desde cada punto de la cuadrícula hasta el punto de atracción.
- Vector/Point/Distance - Arrastre y suelte un componente "Distance" sobre el lienzo.
- Conecte la salida del punto atractor a la entrada A de "Distance".
- Conecte la salida G de “Grid Point” a la entrada B de "Distance".



El primer paso de nuestra definición debe ser similar a la captura de pantalla que se muestra arriba. Si posamos el ratón sobre la salida D de "Distance", veremos una lista de números que corresponden a la distancia desde cada punto de la red hasta el punto de atracción. Vamos a utilizar estos valores para determinar el radio de cada círculo, pero primero debemos escalar estos números hasta una dimensione máxima de radio apropiada.

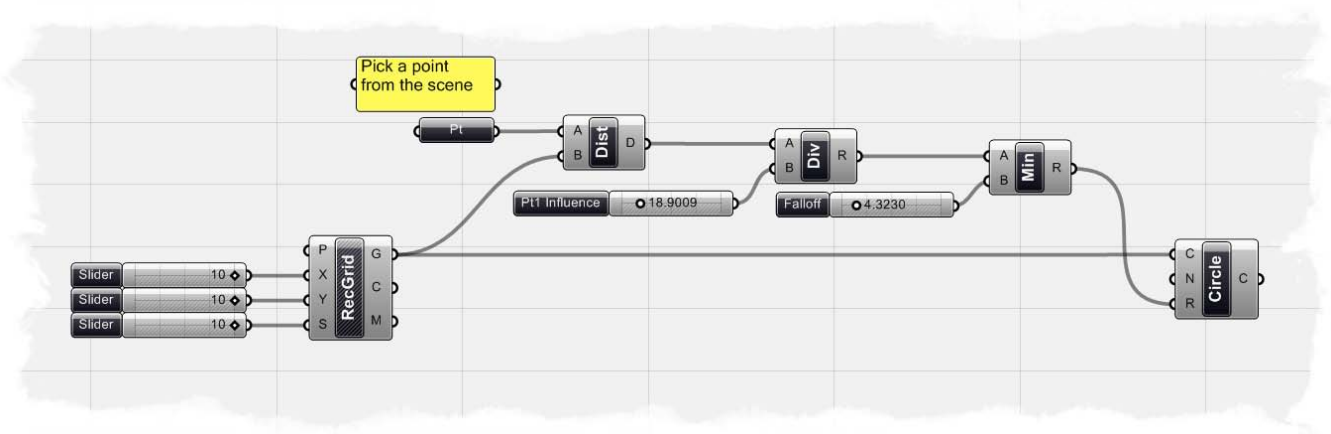
- Scalar/Operators/Division - Arrastre y suelte un componente "Division" sobre el lienzo.
- Conecte la salida D de "Distance" a la entrada A de "Division".
- Params/Special/Numeric Slider - Arrastre y suelte un "Numeric Slider" sobre el lienzo.
- Clic derecho sobre el deslizador y determine lo siguiente:
  - Name: Pt1 Influence
  - Slider Type: Floating Point
  - Lower Limit: 0.0
  - Upper Limit: 100.0
  - Value: 25.0
- Conecte el deslizador "Pt1 Influence" a la entrada B de "Division".

  Dado que los valores de distancia son bastante grandes, necesitamos un factor de escala para llevar los números a un valor más manejable. Hemos utilizado el control deslizante numérico como factor de división que nos dé un nuevo conjunto de valores de salida que podamos utilizar para nuestro radio de círculo. Podemos ingresar directamente estos valores en un componente de círculo, pero podemos usar un poco de matemáticas escalares para determinar una distancia de máxima. Lo que esto significa, es que podemos limitar lo grande de nuestros círculos se si están demasiado lejos desde el punto de atracción.
- Scalar/Utility/Minimum - Arrastre y suelte un componente "Minimum" sobre el lienzo.
- Conecte la salida R de "Division" a la entrada A de "Minimum".
- Params/Special/Numeric Slider - Arrastre y suelte un "Numeric Slider" sobre el lienzo.
- Clic derecho sobre el nuevo deslizador y determine lo siguiente:
  - Name: Falloff
  - Slider Type: Floating Point

- Lower Limit: 0.0
- Upper Limit: 30.0
- Value: 5.0

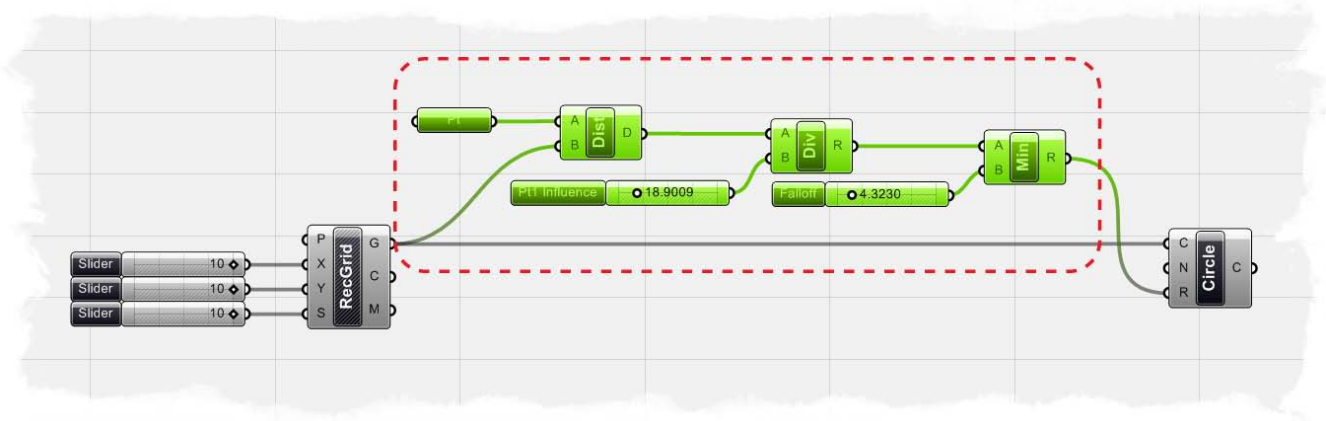
- Conecte el deslizador “Falloff” a la entrada B de “Minimum”.
- Curve/Primitive/Circle CNR - Arrastre y suelte un “Circle CNR” (Centro, Normal, y Radio) sobre el lienzo.

  Nos gustaría que el punto central de cada círculo se localice en uno de los puntos de la red que hemos creado al comienzo de la definición.
- Conecte la salida G de “Rectangular Point Grid” a la entrada C de “Circle”.
- Conecte la salida R de “Minimum” a la entrada R de “Circle”.
- Clic derecho sobre el componente “Rectangular Point Grid” y apague el “Preview”.



Su definición debe ser similar a la imagen de arriba. Ahora hemos creado un conjunto de círculos que se escalan hacia arriba y abajo dependiendo de lo lejos que cada círculo este del punto de atracción. Pero, ¿y si queremos añadir un punto de atracción adicional? Dado que ya tenemos la definición arriba, sólo necesitamos copiar y pegar algunos de nuestros componentes para que esto suceda.

- Seleccione los siguientes componentes: “Attractor Point”, “Distance”, deslizador “Pt 1 Influence”, “Division”, deslizador “Falloff”, y “Minimum”, luego presione Ctrl-c (copiar) y Ctrl-v (pegar) para duplicar estos componentes.

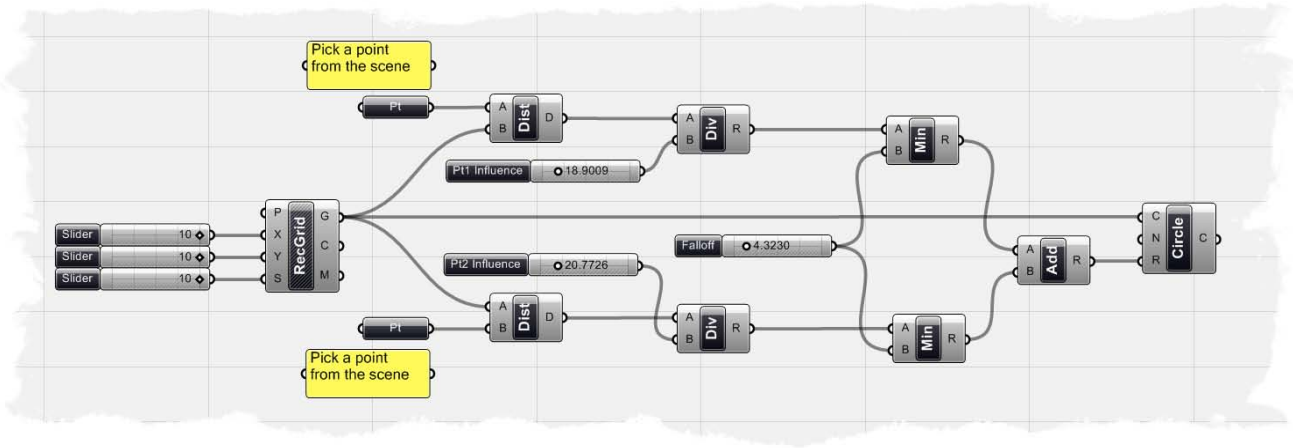


- Mueva los componentes duplicados un poco, de modo que sus componentes no se superpongan unos a otros.

Tenemos que asignar otro punto de atracción a nuestra definición... pero, por supuesto, para ello tenemos que crear otro punto en nuestra escena de Rhino primero.

- En la escena de Rhino, escriba "Point" en el cuadro de diálogo y coloque un punto en cualquier lugar de la escena. Como lo hicimos antes, coloque el punto en la vista Superior para que el punto esté en el plano XY.
- Clic derecho sobre el “Point” duplicado y seleccione "Set One Point".
- Cuando se le pida, seleccione el punto de atracción que acaba de crear en la escena de Rhino.

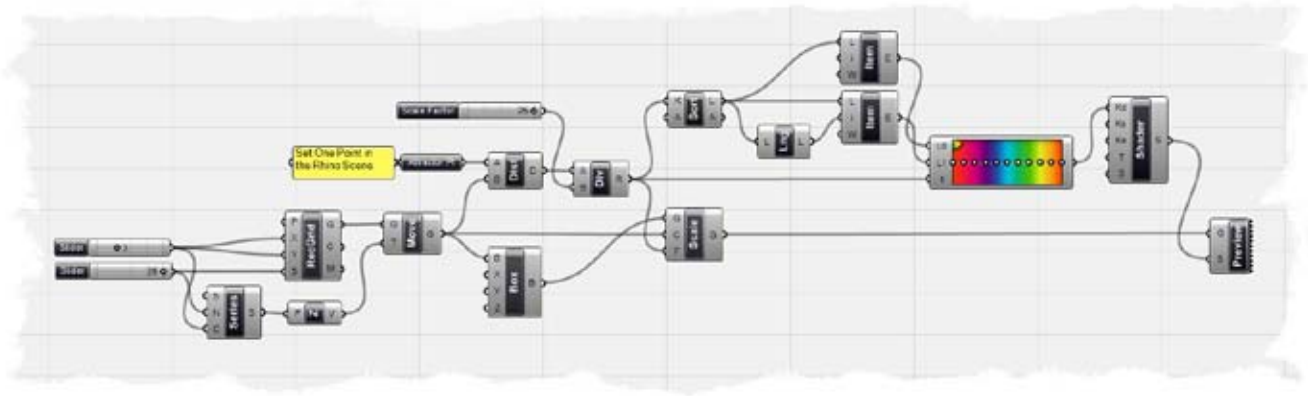
  Ahora disponemos de 2 filas de componentes que están evaluando la distancia de la cuadrícula rectangular de puntos y nos dan información que podemos usar para controlar el radio de los círculos. Sin embargo, tenemos que combinar las dos listas que son el resultado del componente mínimo en una sola lista. Para ello, vamos a utilizar el componente de suma escalar.
- Scalar/Operators/Addition - Arrastre y suelte un componente "Addition" sobre el lienzo.
- Conecte la salida R del primer "Minimum" a la entrada A de "Addition".
- Conecte la salida R del segundo "Minimum" a la entrada B de "Addition".
- Ahora, conecte la salida R de "Addition" a la entrada R de "Circle"
  *(**Nota:** Esto reemplazará la conexión existente.)*



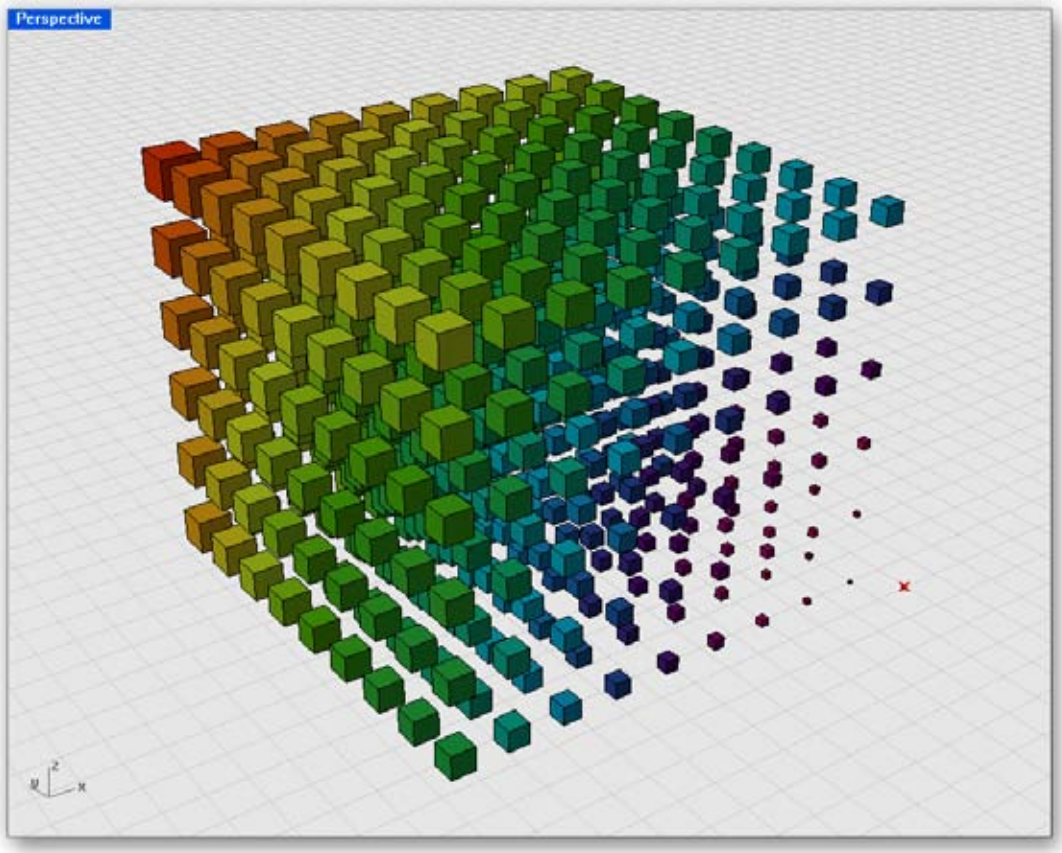
Si se completa correctamente, su definición debe ser similar a la captura de pantalla anterior. Usted puede notar que he suprimido uno de los deslizadores de “Falloff” y estoy controlando ambos componentes mínimos con un control deslizante único. Trate de cambiar los deslizadores “Pt Influence” y “Falloff” para ajustar la escala de sus círculos de acuerdo con la información de la distancia.

Nota: Ver el video tutorial de este ejemplo, visite el blog de David Fano en:
http://designreform.net/2008/07/08/grasshopper-patterning-with-2-attractor-points/

9.3 Usando Vectores/Escalamiento con puntos atractores (Cajas escaladas)
Ya hemos demostrado cómo podemos usar las matemáticas vectoriales y escalares para determinar el radio de los círculos basada en las distancias de los objetos a otro punto, pero también puede utilizar los componentes del núcleo mismo de la escala de objetos y colores, con objeto de determinar los componentes de sombreado de Grasshopper. La definición siguiente se ha utilizado para generar la siguiente captura de pantalla, pero vamos a empezar desde el principio y trabajar a través de la definición paso a paso.



Nota: Para ver la versión final de esta definición, abra el archivo **Color Boxes.ghx** que se encuentra en la carpeta “Archivos de apoyo” que acompaña a este documento.



Paso 1: Comience por crear una grilla de puntos en 3 dimensiones.

- Params/Special/Numeric Slider – Arrastre y suelte **dos** deslizadores en el lienzo.
- Clic derecho sobre el primer deslizador y determine lo siguiente:
  - Slider Type: Integers
  - Lower Limit: 0.0
  - Upper Limit: 10.0
  - Value: 3.0
- Clic derecho sobre el segundo deslizador y determine lo siguiente:
  - Slider Type: Integers
  - Lower Limit: 0.0
  - Upper Limit: 25.0
  - Value: 25.0
- Vector/Point/Grid Rectangular – Arrastre y suelte un componente “Point Grid” sobre el lienzo.
- Conecte el primer deslizador a las entradas X e Y del componente “Point Grid”.
- Conecte el segundo deslizador a la entrada S del componente “Point Grid”.

  Usted debe ver una cuadrícula de puntos en la escena, donde el primer deslizador controla el número de puntos en los ejes X e Y (recuerde que si duplica los puntos desde un punto central, siempre habrá el doble de la cantidad de filas y columnas que la entrada en el deslizante numérico). El interlineado será controlado por el segundo control deslizante. Ahora tenemos que copiar esta grilla de puntos en el eje Z para formar un volumen tridimensional.
- Logic/Sets/Series – Arrastre y suelte un componente “Series” sobre el lienzo.
- Conecte el segundo deslizador a la entrada N del componente “Series”.
- Conecte el primer deslizador a la entrada C del componente “Series”.

  Nuestro componente “Series” contará el número de copias de la grilla de puntos que vamos a hacer en la dirección z, pero es posible que ya haya notado un pequeño error en nuestras matemáticas. Como se mencionó anteriormente, nuestra grlilla de puntos se creó a partir de un punto central, así que aunque nos hemos fijado el número de puntos tanto en el eje X e Y a 3, en realidad tenemos 7 puntos en cada dirección. Puesto que en última instancia, nos gustaría tener un cubo tridimensional de puntos, tenemos que crear 7 copias en el eje Z. A fin de mantener la cuenta uniforme, tendremos que escribir una expresión simple para duplicar la cuenta de serie.
- Clic derecho en la entrada C de “Series” y seleccione el “Expression Tab”.
- En el “Expression editor”, escriba la siguiente ecuación: (**C*2)+1**

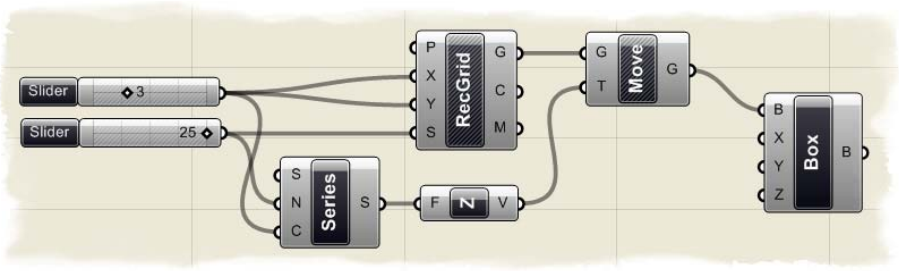
  Ahora le hemos dicho al componente "Series” que multiplique su entrada C por un factor de 2 y luego le añada 1. Ya que nuestra entrada original se establece en 3, nuestra nueva serie será 7,0.
- Vector/Constants/Unit Z – Arraste y suelte un componente “Unit Z” sobre el lienzo.
- Conecte la salida S de “Series” a la entrada F de “Unit Z”.

  Si usted pasa el ratón sobre la salida V de “Unit Z” debería ver que hemos definido 7 valores localmente definidos donde el valor Z de cada entrada aumenta en 25,0, o el valor del segundo control deslizante. Vamos a utilizar este valor de vector para definir la distancia que nos gustaría que cada copia de nuestra grilla tenga.
- X Form/Euclidean/Move – Arraste y suelte un componente “Move” en el lienzo.
- Conecte la salida G de “Point Grid” a la entrada G de “Move”.
- Conecte la salida V de “Unit Z” a la entrada T de “Move”.

*Si nos fijamos en la escena de Rhino, se dará cuenta de que nuestros puntos no necesariamente se parecen mucho a un cubo tridimensional de puntos. En todo caso, parece una rampa que conduce a un plano rectangular de puntos. Esto es porque el algoritmo de coincidencia de datos por defecto está en "**Longest List**". Si hace clic en el componente, puede cambiar el algoritmo a "**Cross Reference**". Ahora, cuando usted mira la escena debería ver un cubo de puntos en tres dimensiones. (Vea el Capítulo 6 para obtener más información acerca de "Coincidencia de flujos de datos").*

- Surface/Primitive/Center Box – Arrastre y suelte un componente "Center Box" sobre el lienzo.
- Conecte la salida G de "Move" a la entrada B de "Center Box".
- Clic derecho en los componentes "Point Grid" y "Move" y apague el "Preview".

A continuación se muestra una captura de pantalla de nuestro primer paso de la definición.



Paso 2: Resolver las matemáticas escalares y vectoriales.

- Params/Geometry/Point – Arrastre y suelte un componente "Point" en el lienzo.
- Clic derecho en el componente "Point" y renómbrelo como: "Attractor Pt".

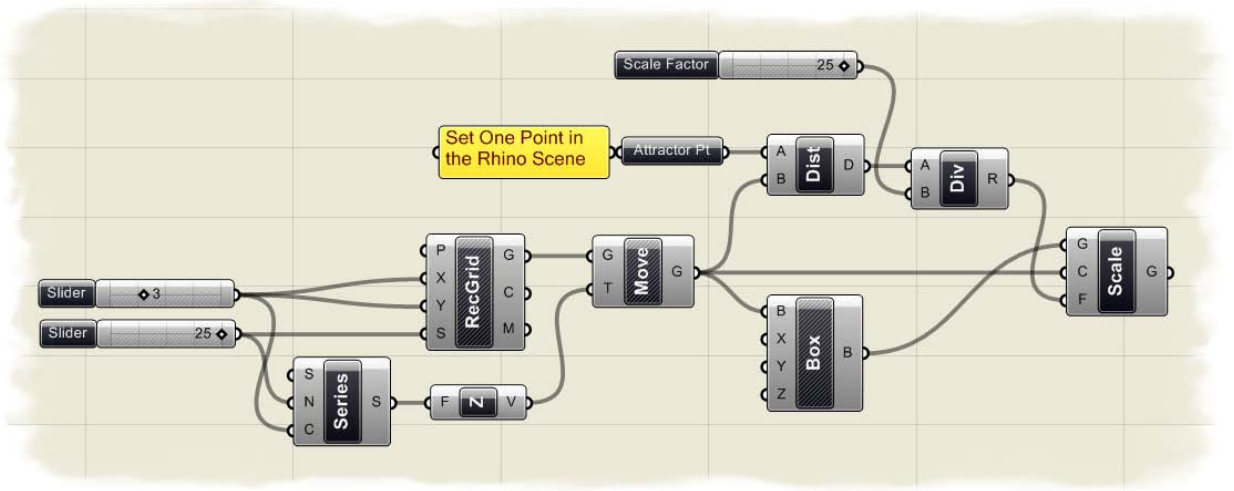
  Al igual que en el ejemplo de los círculos escalados, habrá que asignar el componente "Atractor Pt" a un punto creado en la escena de Rhino. Para ello, primero tendrá que crear un punto.
- En la escena de Rhino, escriba "Point" en el cuadro de diálogo y coloque un punto en cualquier lugar de la escena.
- Vuelva a Grasshopper, haga clic derecho en el componente de "Pt Atractor" y seleccione "Set One Point".
- Cuando se le pida, seleccione el punto recién creado en la escena de Rhino.

  Ahora debería ver una X roja sobre el punto que indica que el componente "Atractor PT" ha sido asignado al punto en la escena. Si mueve el punto en cualquier lugar de la escena, el componente de Grasshopper actualizará automáticamente su posición.
- Vector/Point/Distance – Arrastre y suelte un componente "Distance" sobre el lienzo.
- Conecte al salida del "Attractor Pt" a la entrada A de "Distance".
- Conecte la salida G de "Move" a la entrada B de "Distance".

  Si posa su ratón sobre la salida D de "Distance", podremos ver una lista de valores numéricos que indican la distancia entre cada punto desde el "Atractor Pt". Para poder utilizar estos valores como un factor de escala de nuestras cajas, tendremos que dividirlos por un número para que los valores máximos sean apropiados para nuestro ejemplo.

- Scalar/Operators/Division – Arrastre y suelte un componente “Division” sobre el lienzo.
- Conecte la salida D de “Distance” a la entrada A de “Division”.
- Params/Special/Numeric Slider – Arrastre y suelte un deslizador en el lienzo.
- Clic derecho sobre el deslizador y determine lo siguiente:
  - Name: Scale Factor
  - Slider Type: Integers
  - Lower Limit: 0.0
  - Upper Limit: 25.0
  - Value: 25.0
- Conecte el deslizador “Scale Factor” a la entrada B de “Division”.
- X Form/Affine/Scale – Arrastre y suelte un componente “Scale” en el lienzo.
- Conecte la salida B de “Center Box” a la entrada G de “Scale”.
- Conecte la salida R de “Division” a la entrada F de “Scale”.
- Clic derecho sobre el componente “Center Box” y apague la opción “Preview”.

A continuación se muestra una captura de pantalla de la forma en que la definición se ha creado hasta ahora. Si nos fijamos en la escena Rhino ahora, usted debe notar que todos los cuadros se han reducido en función de su distancia desde el punto de atracción. Podemos llevar nuestra definición un paso más allá mediante la adición de color a nuestras cajas para darnos una representación visual del factor de escala.



Paso 3: Asignar un valor de color a cada caja escalada.

- Logic/List/Sort List – Arrastre y suelte un componente “Sort List” en el lienzo.

A fin de asignar un valor de color basado en la distancia de cada caja hasta el punto de atracción, será necesario conocer dos valores numéricos, el punto más cercano y el punto que más lejano. Para ello, tenemos que ordenar nuestra lista de valores de distancia, y retornar la primera y la última entrada de la lista.

- Logic/List/List Item – Arrastre y suelte un componente “List Item” en el lienzo.
- Conecte la salida L de “Sort List” a la entrada L de “List Item”.
- Clic derecho en la entrada i de “List Item” y ajuste el “Integer value” en 0.0

Esto entregará la primera entrada en nuestra lista que será el valor de distancia menor .

- Logic/List/List Length – Arrastre y suelte un componente “List Length” sobre el lienzo.
- Conecte la salida L de “Sort List” a la entrada L de “List Length”.

El componente “List Length” nos dirá cuántas entradas existen en nuestra lista. Podemos ingresar esta información en otro componente “List Item” para obtener el último valor de la lista.

- Logic/List/List Item- Arrastre y suelte otro componente “List Item” sobre el lienzo.
- Conecte la salida L de “Sort List” a la entrada L del segundo “List Item”.
- Conecte la salida L de “List Length” a la entrada i del segundo “List Item”.

  Si posa su ratón sobre la salida E del segundo “List Item”, verá que el componente no ha recuperado el último valor de la lista. Esto es porque los datos de la lista en el Grasshopper siempre almacenan el primer valor de entrada como 0. Entonces, si nuestra longitud de lista muestra que hay 100 valores y nuestro primer número comienza en 0, nuestro número en la última entrada en realidad será el 99. Debemos agregar una expresión a la entrada i del segundo “List Item” para restar 1 valor en la longitud de la lista para realmente recuperar el último elemento.
- Clic derecho en la entrada i del segundo “List Item” y vaya a “Expression Editor”
- En el cuadro de diálogo, escriba la siguiente ecuación: **i-1**

  Ahora, si usted posa el ratón sobre la salida E del segundo “List Item “, debería ver un valor numérico que corresponde al valor de la distancia más lejana desde el punto de atracción.
- Params/Special/Gradient – Arrastre y suelte un componente “Gradient” sobre el lienzo.
- Conecte la salida E del primer “List Item” (el que está asociado con el menor valor de distancia) con la entrada L0 de “Gradient”.
- Conecte la salida E del primer “List Item” (el que está asociado con el mayor valor de distancia) con la entrada L1 de “Gradient”.
- Conecte la salida R de “Division” a la entrada t de “Gradient”.

  La entrada L0 define cual valor numérico representará el lado izquierdo de “Gradient”, y en nuestro ejemplo, el lado izquierdo de la gradación se indica la caja más cercana al punto de atracción. La entrada L1 del “Gradient” define cual valor numérico representará la parte derecha de la gradiente, y tenemos que establecer esto para representar el valor del punto más lejano desde el punto de atracción. El valor de entrada t representa la lista de los valores que desea trazar a lo largo del intervalo de la gradiente. Hemos elegido a ingresar todos nuestros valores de factor de escala, de modo que la escala de cada caja, se asociará con un color a lo largo del rango de gradiente.
- Vector/Color/Create Shader – Arrastre y suelte un componente “Create Shader” sobre el lienzo.
- Conecte la salida de “Gradient” a la entrada Kd de “Shader”.

  El componente de “Shader” posee varias entradas para ayudarnos a definir cómo queremos que sea su vista previa. A continuación se muestra una descripción de cómo cada entrada afecta a los colores resultantes.

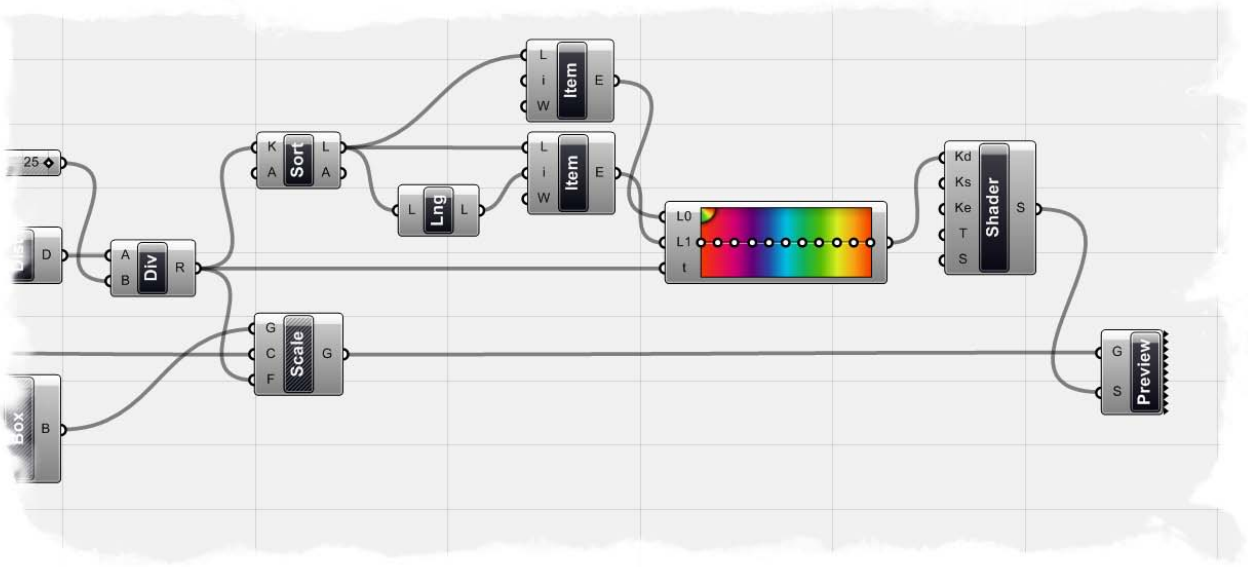
  ***Kd:** Define un color difuso. Esto va a definir el color principal de cada objeto. El color difuso es definido por tres números enteros que van desde 0 hasta 255, y representan los niveles de Rojo, Verde y Azul de un color.*
  ***Ks:** Define el color de la iluminación especular y requiere una entrada de tres valores enteros para definir su color RGB.*
  ***Ke:** Define la iluminación propia de cada color.*
  ***T:** Define la transparencia de cada color.*

S: *Define el brillo del color, donde un valor 0 significa que el color es opaco, y el valor 100 implica el brillo máximo.*

*Hemos conectado el control deslizante a la entrada Kd (degradado) del componente "Shader" para que los colores primarios de nuestra caja estén representados por el patrón de gradiente. Usted puede cambiar los colores del degradado mediante la selección de uno de los pequeños puntos blancos en el patrón de degradado. También puede arrastrar el punto de la pendiente de arriba a abajo para controlar el cambio de color. Además, hay un número preestablecido de patrones de degradado cargado en el componente, y se puede establecer al hacer clic derecho en el patrón de la pendiente y escogiendo uno de los cuatro patrones preestablecidos. Nuestro ejemplo tiene el patrón de gradiente establecido en "**Spectrum**".*

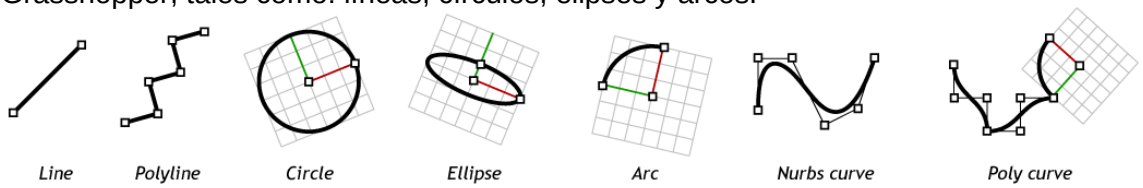
- Params/Special/Custom Preview – Arrastre y suelte un componente "Custom Preview" sobre el lienzo.
- Conecte la salida G de "Scale" a la entrada G de "Custom Preview".
- Conecte la salida S de "Shader" a la entrada S de "Custom Preview".
- Clic derecho sobre el componente "Scale" y apague la opción "Preview".

A continuación se muestra una captura de pantalla de la tercera etapa de esta definición. Si mueve el punto de atracción en la escena de Rhino, las cajas escaladas y la información de color se actualizarán automáticamente.



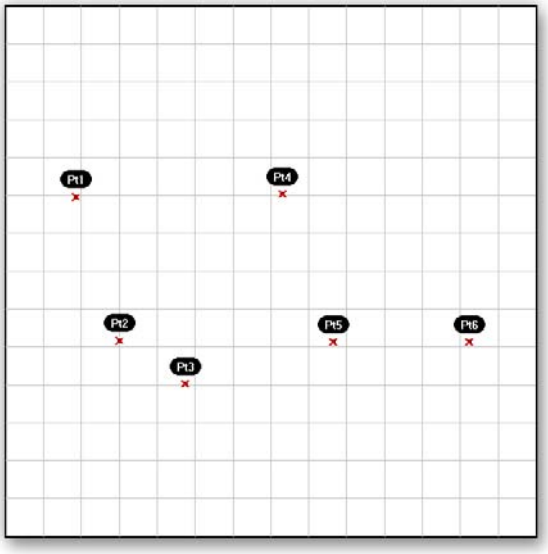
10 *Tipos de curvas*

Dado que las curvas son objetos geométricos, ellos poseen un número de propiedades o características que pueden ser utilizadas para describir o analizar. Por ejemplo, cada curva tiene una coordenada inicial y una coordenada final.Cuando la distancia entre estas dos coordenadas es cero, la curva se cierra. Además, cada curva tiene un número de puntos de control, si todos estos puntos están ubicados en el mismo plano, la curva es completamente plana. Algunas propiedades se aplican a la curva en su conjunto, mientras que otras sólo se aplican a puntos específicos de la curva. Por ejemplo, la planaridad es una propiedad global, mientras que los vectores tangentes son una propiedad local. Además, algunas de las propiedades sólo se aplican a algunos tipos de curva. Hasta ahora hemos discutido algunos de los **componentes primitivos de las curvas** de Grasshopper, tales como: líneas, círculos, elipses y arcos.

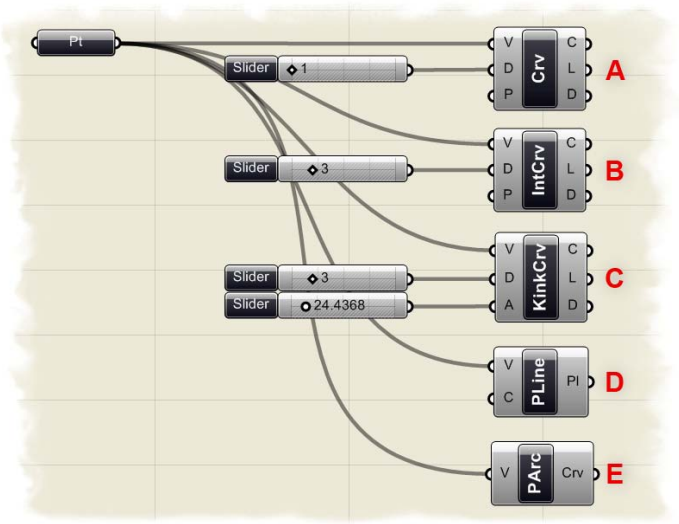


Grasshopper también tiene un conjunto de herramientas para expresar curvas de Rhino de tipos más avanzados, como las curvas NURBS y las policurvas . A continuación, un ejemplo que nos llevara a través de algunos de los componentes de la Spline de Grasshopper, pero primero tendrá que crear un conjunto de puntos que definirán cómo nuestras curvas van a trabajar.

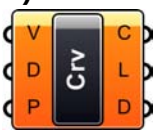
En la carpeta “Archivos de apoyo” que acompaña a este manual, abra el archivo **Curve Types.3dm**. En la escena, se encuentran 6 puntos colocados en el plano XY. Están etiquetado de izquierda a derecha, como la imagen de la derecha, pues este es el orden en que serán tomados dentro de Grasshopper.



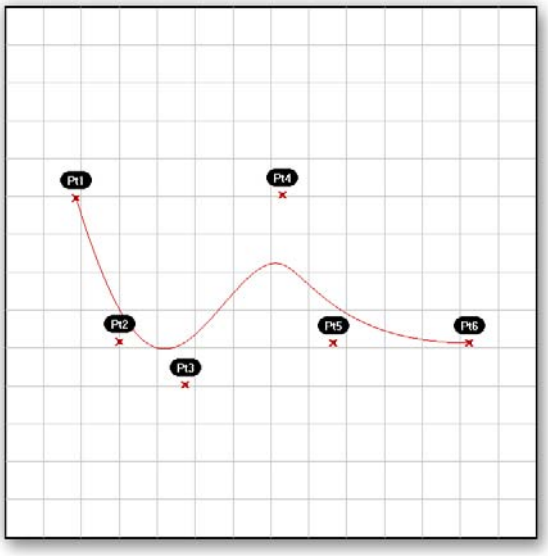
Ahora, en Grasshopper, abra el archivo **Curve Types.ghx** de la carpeta “Archivos de apoyo” que acompaña a este manual. Usted verá un componente “Point” conectado a varios componentes “Curve”, cada uno definiendo una curva usando un metodo distinto. Iremos a través de cada componente por separado, pero primero tenemos que asignar los puntos en la escena de Rhino para el componente **“Multiple Points”**. Cuando se le pida, seleccione cada uno de los 6 puntos, asegurándose de seleccionar los puntos en el orden correcto, de izquierda a derecha. Como ha seleccionado los puntos, una línea de conexión implícita se dibuja en la pantalla en color azul para indicar sus selecciones. Cuando haya seleccionado los 6 puntos, pulse **Intro** para volver a Grasshopper. Los 6 puntos deben tener ahora una pequeña X roja en la parte superior de ellos, lo que indica, que este punto en particular se le ha asignado al componente “Point” de Grasshopper.



A) NURBS Curves Curve/Spline/Curve)



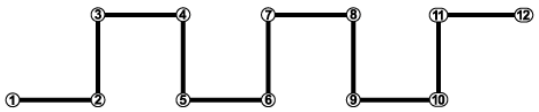
Non-**U**niform **R**ational **B**asic **S**plines (Curvas de Base Racional No-Uniforme), o curvas NURBS ,son una de las tantas definiciones de curvas que están disponibles en Rhino. Además de los puntos de control que ayudan a definir la ubicación de las curvas (estos son los 6 puntos que acaba de seleccionar en Rhino), las curvas de NURBS también poseen propiedades específicas, como grado, vectores nudo y pesos. Libros enteros (o al menos ensayos muy largos), han escrito las matemáticas detrás de las curvas NURBS, y no voy a referirme a esto aquí. Sin embargo, si desea un poco más de información acerca de este tema, porfavor visite:



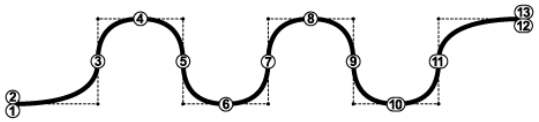
http://en.wikipedia.org/wiki/NURBS.

La entrada V de **"NURBS Curve"** define lo puntos de control la curva y puede ser descrita implícitamente por la selección de puntos dentro de la escena Rhino, o por la herencia de datos variables de otros componentes. La entrada D de **"NURBS Curve"** establece el grado de curvatura. El grado de curvatura es siempre un entero positivo entre 1 y 11 incluyéndolos. Básicamente, el grado de curvatura determina el rango de influencia que los puntos de control tienen sobre ella, cuanto más alto el grado, más largo el intervalo. La tabla en la página siguiente es del manual RhinoScript 101 de David Rutten, e ilustra cómo los diferentes grados definen el resultado de una curva NURBS .

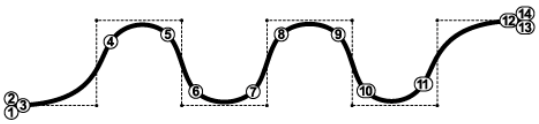
NURBS curve knot vectors as a result of varying degree



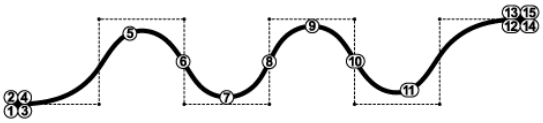
A D^1 nurbs curve behaves the same as a polyline. It follows from the knotcount formula that a D^1 curve has a knot for every control point. Thus, there is a one-to-one relationship.



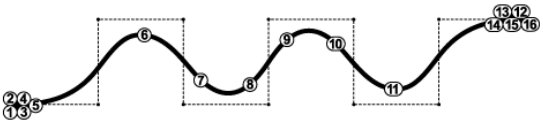
A D^2 nurbs curve is in fact a rare sighting. It always looks like it is over-stressed, but the knots are at least in straightforward locations. The spline intersects with the control polygon halfway each segment. D^2 nurbs curves are typically only used to approximate arcs and circles.



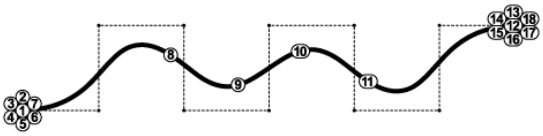
D^3 is the most common type of nurbs curve and -indeed- the default in Rhino. You are probably very familiar with the visual progression of the spline, eventhough the knots appear to be in odd locations.

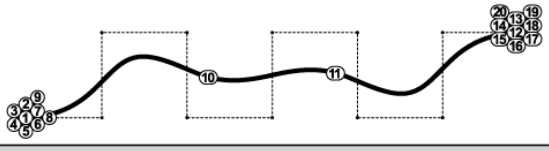


D^4 is technically possible in Rhino, but the math for nurbs curves doesn't work as well with even degrees. Odd numbers are usually preferred.



D^5 is also quite a common degree. Like the D^3 curves it has a natural, but smoother appearance. Because of the higher degree, control points have a larger range of influence.





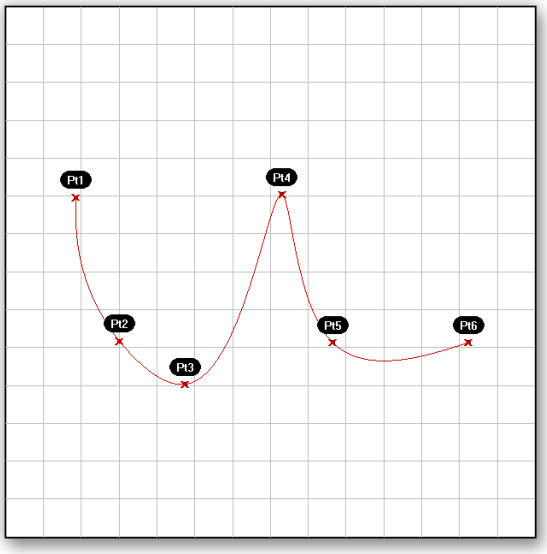
D^7 and D^9 are pretty much hypothetical degrees. Rhino goes all the way up to D^{11}, but these high-degree-splines bear so little resemblance to the shape of the control polygon that they are unlikely to be of use in typical modeling applications.

En nuestro ejemplo, hemos conectado un control deslizante a la entrada D de “Curve” para definir el grado curvatura de nuestra curva NURBS. Arrastrando nuestro control deslizante hacia la izquierda y la derecha, se puede ver visualmente el cambio en la influencia de cada punto de control. La **entrada P de la curva NURBS** utiliza un valor booleano para definir si la curva debe ser periódica o no. Una entrada Falso creará una curva NURBS abierta, mientras que un valor Verdadero creará una curva NURBS cerrada. Los tres valores de salida para el componente son bastante explícitos, donde la **salida C** define la curva resultante, la **salida L** proporciona un valor numérico de la longitud de la curva, y la **salida D** define el dominio de la curva (o el intervalo desde 0 hasta el valor numérico del grado de la curva).

B) Curvas Interpoladas (Curve/Spline/**Interpolate**)

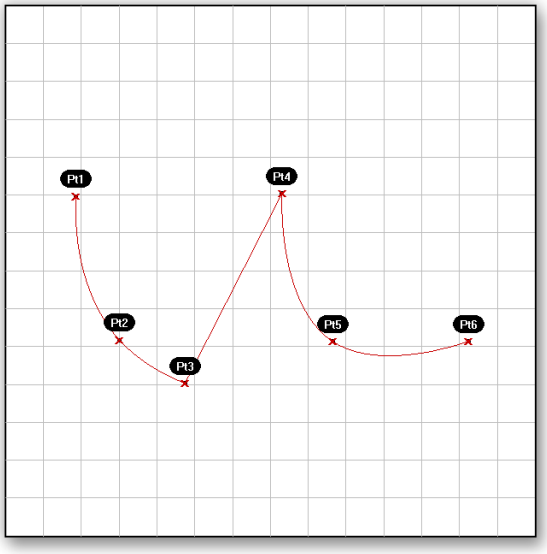
Las curvas interpoladas se comportan de forma ligeramente diferente que las curvas NURBS, en que las curvas interpoladas pasan por los puntos de control. Como sabemos, es muy difícil hacer que las curvas NURBS pasen por unas determinadas coordenadas. Incluso si tuviéramos que ajustar los puntos de control individualmente, sería una tarea muy ardua para que la curva pasara a través de un punto específico. Dentro de las curvas interpoladas, la **entrada V** es similar al componente de NURBS, en que utiliza un conjunto específico de puntos para crear la curva. Sin embargo, con el método de la curva interpolada, la curva resultante de hecho

pasará a través de estos puntos, independientemente del grado de curvatura. En el componente de curva de NURBS, sólo se podría lograr esto cuando el grado de curvatura se establece en 1. Además, como el componente de curva de NURBS, la **entrada D** define el grado de la curva resultante. Sin embargo, con este método, sólo tomará valores numéricos de grados singulares, por lo que es imposible crear una curva interpolada de 2 grados. Una vez más, la **entrada P** determina si la curva es periódica. Las **salidas C, L y D**, suelen especificar la curva resultante, la longitud, y el dominio de la curva, respectivamente.



C) Curvas Kinky

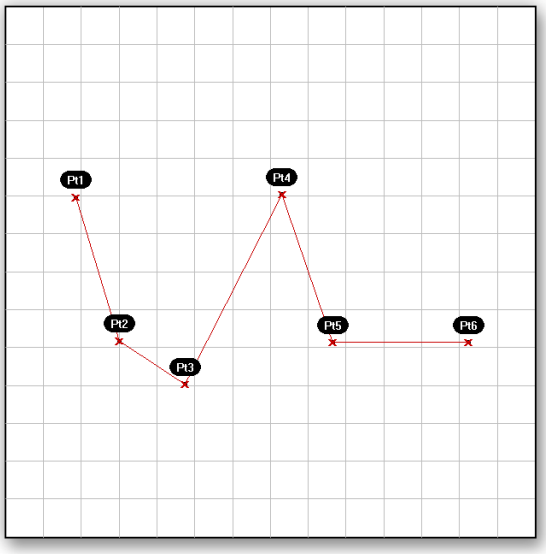
A pesar de su nombre, una curva Kinky no es más que una curva interpolada glorificada. Tiene muchos de los atributos de la curva mencionada en el sección B, con una pequeña diferencia. El componente de la curva Kinky posee la posibilidad de controlar un umbral de ángulo determinado, donde la curva pasa de una línea quebrada a una curva suave interpolada. Hemos conectado un control deslizante numérico para la entrada de un componente de curva Kinky para ver el cambio de umbral en tiempo real. Cabe señalar que la entrada A requiere un valor en radianes. En nuestro ejemplo, hay una expresión en la entrada A para convertir nuestro control deslizante de un ángulo en grados a radianes.



D) Curvas Polilíneas (Curve/Spline/Polyline)

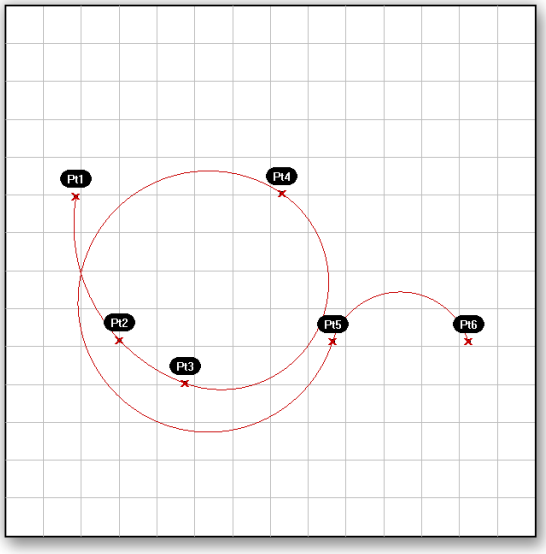
Una polilínea es uno de los tipos de curva más flexible disponible en Rhino. Esto se debe a que una curva polilínea puede estar compuesta de segmentos de línea, segmentos de polilínea, curvas NURBS de grado 1, o cualquier combinación de los anteriores. Pero, vamos a empezar con los fundamentos detrás de la polilínea. En esencia, una polilínea es igual que una matriz de puntos. La única diferencia es que tratamos a los puntos de una polilínea como una serie, que nos permite trazar una línea secuencial entre ellas. Como se mencionó anteriormente, una curva NURBS de grado 1, actúa, para todos los efectos, de forma idéntica a una polilínea. Ya que una polilínea es una colección de segmentos de línea que conecta dos o más puntos, la línea resultante pasará siempre a través de sus puntos de control, por lo

que es similar en algunos aspectos a una curva interpolada. Al igual que los tipos de curva antes mencionados, la entrada V del componente Polilínea especifica un conjunto de puntos que definirán los límites de cada segmento de línea que conforman la polilínea. La entrada C del componente define si la polilínea es una curva abierta o cerrada. Si la ubicación del primer punto no coincide con la ubicación del último punto, un segmento de línea se creará para cerrar el bucle. La salida para el componente de Polilínea es ligeramente diferente a la de los ejemplos anteriores, en que, la única resultante es la curva en sí. Usted tendría que utilizar uno de los otros componentes de la curva de análisis dentro de Grasshopper para determinar los otros atributos de la curva.



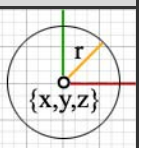
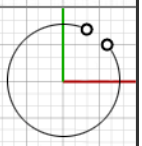
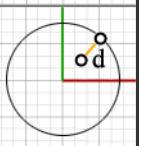
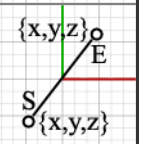
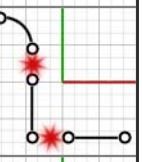
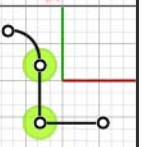
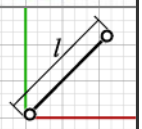
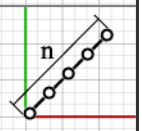
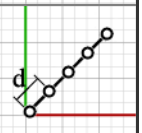
E) Poli-arcos (Curve/Spline/Poly Arc)

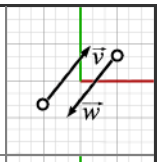
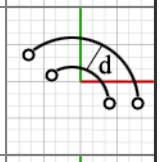
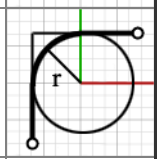
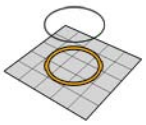
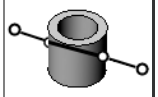
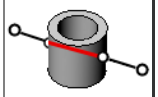
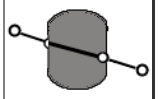
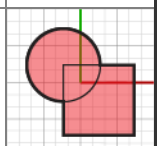
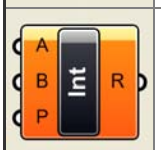
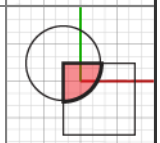
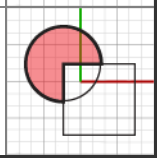
Un poli-arco es casi idéntico en naturaleza a la polilínea, excepto que en lugar de segmentos de línea recta, el poli-arco utiliza una serie de arcos que conectan cada punto. El poli-arco es el único, que calcula la tangencia requerida en cada punto de control a fin de crear una curva de fluida en la transición entre cada arco. No existen otras entradas más que la matriz de punto inicial, y la única salida es la curva resultante.



10.1 Analisis de Curva

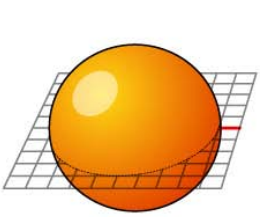
Sería muy difícil crear un tutorial que utulice todas las herramientas analíticas disponibles en Grasshopper, así que he incluido una tabla para explicar muchos de los componentes más usados.

Componente	Ubicación	Descripción	Ejemplo
	Curve/Analysis/**Center**	Encuentra el punto central y el radio de arcos y círculos	
	Curve/Analysis/**Closed**	Prueba si una curva es cerrada o abierta	
	Curve/Analysis/**Closest Point**	Encuentra el punto más cercano de una curva a cualquier otro punto en el espacio	
	Curve/Analysis/**End Points**	Extrae los puntos extremos de una curva	
	Curve/Analysis/**Explode**	Descompone una curva en sus componentes	
	Curve/Utility/**Join Curves**	Une cuantos segmentos de curva sea posible	
	Curve/Analysis/**Length**	Mide la longitud de la curva	
	Curve/Division/**Divide Curve**	Divide la curva en segmentos de igual longitud	
	Curve/Division/**Divide Distance**	Divide una curva con una distancia preestablecida entre puntos	
	Curve/Division/**Divide Length**	Dividir una curva en segmentos con una longitud preestablecida	

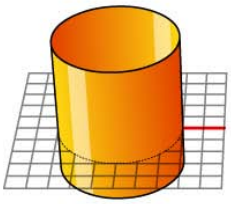
	Curve/Utility/**Flip**	Invierte la dirección de una curva con una curva guía opcional	
	Curve/Utility/**Offset**	Desplaza una curva con una distancia especifica	
	Curve/Utility/**Fillet**	redondea la esquina de una curva con un radio establecido	
	Curve/Utility/**Project**	Proyecta una curva en un Brep (un Brep es un conjunto de superficies unidas como una polisuperficie en Rhino)	
	Intersect/Region/**Split with Brep(s)**	Divide una curva con uno o mas Brips	
	Intersect/Region/**Trim with Brep(s)**	Recortar una curva con uno o más Breps. El **Ci** (curvas interiores) y **Co** (curvas exteriores) indican la dirección en que decea que el ajuste se realice.	
	Intersect/Region/**Trim with Region(s)**	Recorta una curva con una o más regiones. El **Ci** (curvas interiores) y **Co** (curvas exteriores) indican la dirección en que decea que el ajuste se realice.	
	Intersect/Boolean/**Region Union**	Encuentra el contorno (o unión) de dos planos de curvas cerradas	
	Intersect/Boolean/**Region Intersection**	Encuentra la intersección de dos planos de curvas cerradas	
	Intersect/Boolean/**Region Difference**	Encuentra la diferencia entre dos planos de curvas cerradas	

11 *Tipos de superficie**

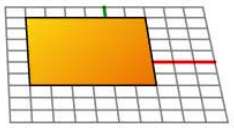
Aparte de unos pocos tipos de superficies primitivas, tales como esferas, conos, planos y cilindros, Rhino es compatible con tres tipos de tipos de superficies de forma libre, de las cuales la más útil es la superficie NURBS. Similares a las curvas, todas las formas de superficies posibles se pueden representar por una superficie NURBS, y esta es el último recurso por defecto del nuevo Rhino. También es lejos la más útil definición de superficie y en la cual nos centraremos de aquí en adelante.



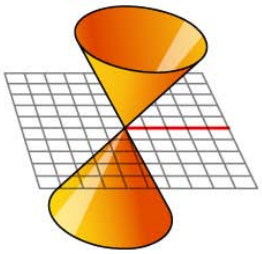
Esfera primitiva
(plano; radio)



Cilindro primitivo
(plano; radio; alto)

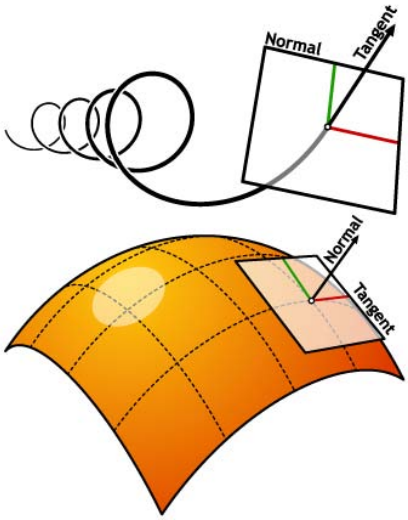


Plano primitivo
(plano; ancho; alto)



Cono primitivo
(plano; radio; alto)

Las superficies NURBS son muy similares a las curvas NURBS. Los mismos algoritmos se utilizan para calcular la forma, normales, tangentes, curvaturas y otras propiedades, pero hay algunas diferencias. Por ejemplo, las curvas poseen vectores tangentes y planos normales, mientras que las superficies poseen vectores normales y planos tangentes. Esto significa que las curvas carecen de orientación, mientras que las superficies carecen de dirección. Esto es, por supuesto cierto para todos los tipos de curva y superficie y es algo con lo cual tendrá que aprender a vivir. A menudo cuando el código escrito consiste en curvas o superficies, usted tendrá que hacer suposiciones acerca de la dirección y orientación, y estos supuestos a veces serán erróneos.



En el caso de superficies NURBS existen en efecto dos direcciones implícitas por la geometría, porque las superficies NURBS son grillas rectangulares de curvas (u) y (v). Y a pesar de que estas direcciones son a menudo arbitrarias, terminamos utilizándolas de todos modos porque nos hacen la vida mucho más fácil para nosotros.

Grasshopper maneja superficies NURBS de forma similar a la forma en que lo hace Rhino porque es construido con el mismo núcleo de las operaciones necesarias para generar la superficie. Sin embargo, debido a que Grasshopper muestra la superficie sobre la vista de Rhino (por lo que en realidad no se puede seleccionar ninguna de las geometrías creadas a través de Grasshopper en el visor, hasta que los resultados en la escena sean terminados), algunas de las opciones de malla son ligeramente menores con el fin de mantener la velocidad de los resultados de Grasshopper bastante altos. Usted puede notar algunas imperfecciones sobre la malla de su superficie, pero esto es de esperarse, y es sólo una consecuencia de los ajustes de dibujo de Grasshopper. Cualquier geometría finalizada utilizará las configuraciones más altas de la malla.

* Fuente: Rhinoscript 101 por David Rutten
http://en.wiki.mcneel.com/default.aspx/McNeel/RhinoScript101

Grasshopper permite manejar las superficies de dos maneras. La primera, como ya hemos discutido, es a través del uso de superficies NURBS. En general, todos los componentes de análisis de superficies pueden ser utilizados en superficies NURBS, como encontrar el área o la curvatura
de una determinada superficie. Si bien hay una buena cantidad de cálculos complicados, esto todavía sería bastante fácil de resolver porque el ordenador no tiene que tener en cuenta la tercera dimensión de un volumen, como la profundidad o espesor. Pero, ¿cómo interpreta Grasshopper las superficies en tres dimensiones? Bueno, los desarrolladores de McNeel decidieron darnos una mano, y crear un segundo método desde el que podemos controlar objetos sólidos como lo hacemos normalmente en la interfaz de Rhino. Introdúzcase a la representación de límites “BREP” (Boundary Representation).

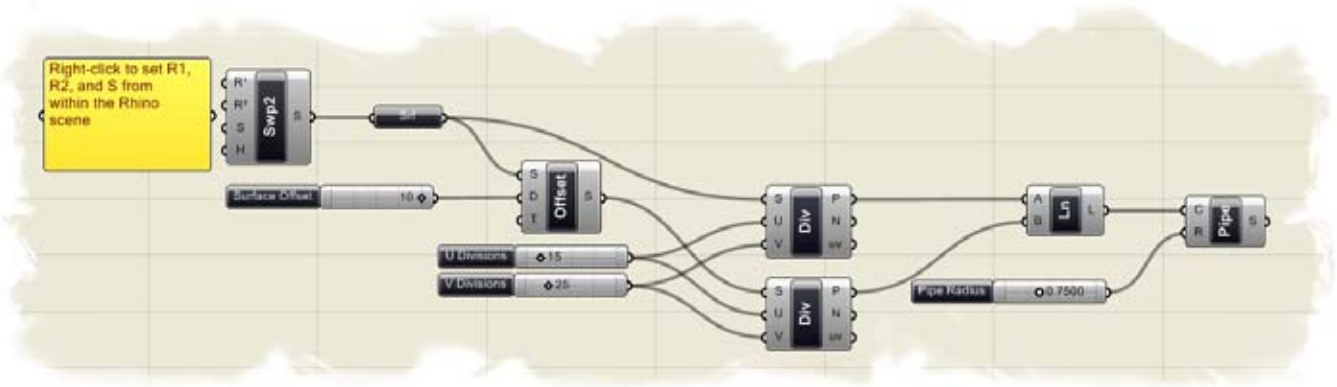
El BREP puede ser considerado como un sólido de tres dimensiones o como un sólido similar a la de una polisuperficie en Rhino. Todavía se compone de una colección de superficie NURBS, sin embargo se unen juntas para crear un objeto sólido con un espesor, mientras que una única superficie NURBS teóricamente no tiene espesor. Desde que BREP se compone esencialmente de superficies unidas, algunos de los componentes de análisis estándar de superficies NURBS todavía trabajan en un BREP, mientras que otros no. Esto sucede porque Grasshopper se ha construido en un la lógica de traducción, que trata de convertir los objetos en la entrada deseada. Si un componente quiere un BREP para la entrada y le das una superficie, la superficie se convierte en un BREP sobre la marcha. Lo mismo es cierto para los números y números enteros, Colores y Vectores, Arcos y círculos. Hay incluso algunas traducciones bastante exóticas definidas, por ejemplo:

- Curva → Numero (entrega la longitud de la curva)
- Curva → Intervalo (entrega el dominio de la curva)
- Superficie → Intervalo2D (entraga el dominio uv de la superficie)
- Cadena → Numero (evaluara la cadena, incluso si es una expresión completa)
- Intervalo2D → Numero (entrega el área del intervalo)

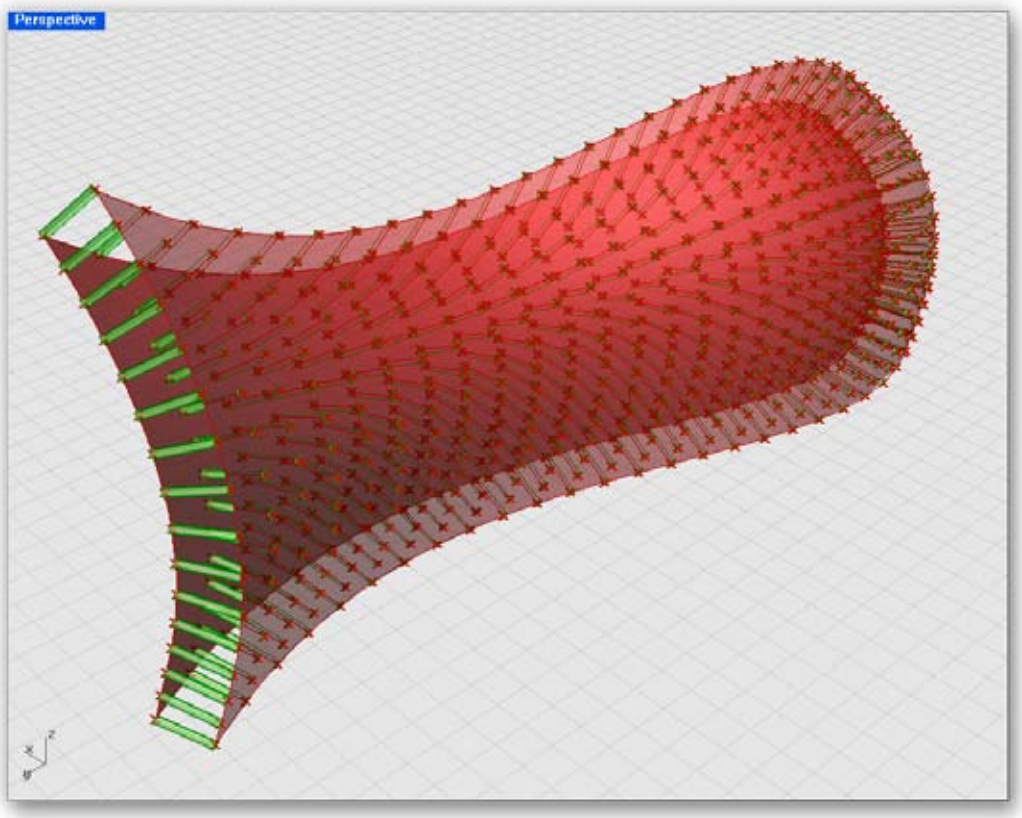
Hay más traducciones de auto-transformación y cada vez que se añaden más tipos de datos, la lista crece. Estos deberían ser suficientes antecedentes sobre los tipos de superficie para comenzar a ver unos ejemplos diferentes.

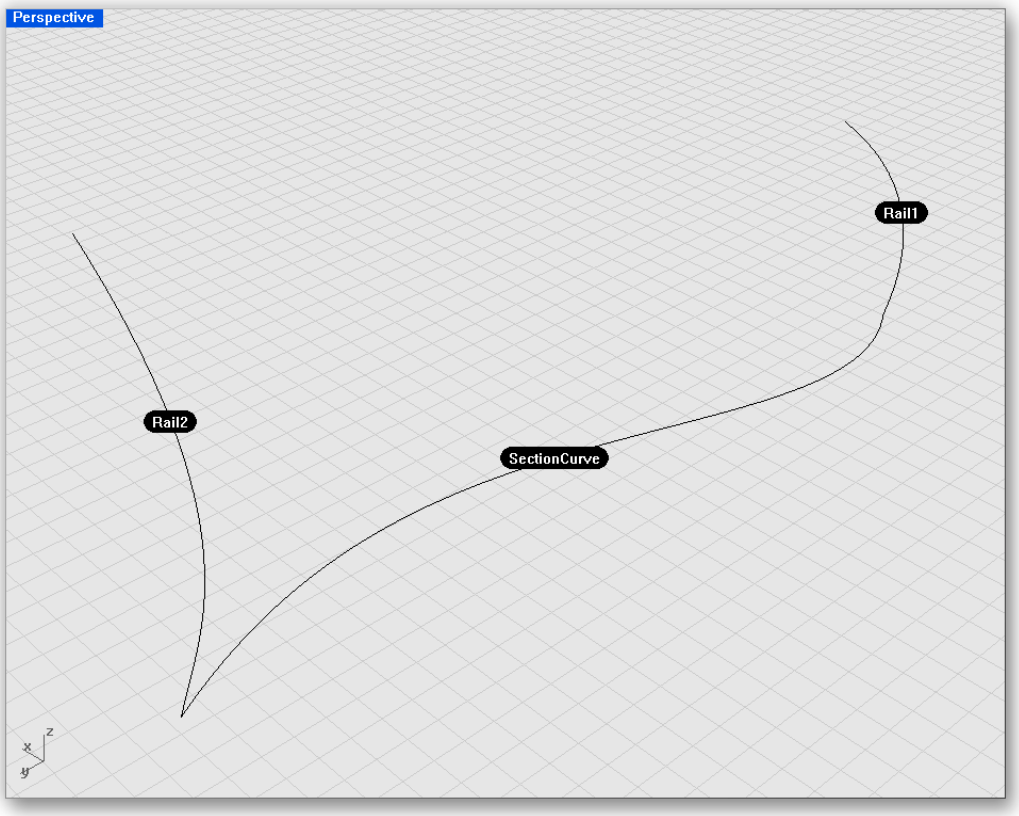
11.1 Conexión de Superficies.

El siguiente ejemplo, creado por David Fano de Design Reform, es un excelente tutorial que muestra una gama de técnicas de manipulación de superficies. En este ejemplo vamos a ocupar los componentes Sweep2Rail, Offset Surface y Surface Division mediante la creación del modelo que aparece a continuación. Para empezar, en Rhino abra el archivo SurfaceConnect.3dm que se encuentra en la carpeta de archivos que acompaña a este manual. En este archivo encontrará 3 curvas (2 rail curves y una section curve), que proporcionarán el marco necesario para iniciar este ejemplo.



Nota: Para ver la definición final de este ejemplo, abra en Grasshopper el archivo **SurfaceConnect.ghx** también encontrado en la carpeta “Archivos de apoyo”.





Para crear la definición desde el inicio:

- Surface/Freeform/Sweep2Rail - Arrastrar y soltar un componente de Sweep 2 Rails en el lienzo.
- Haga clic derecho en la entrada R1 del Sweep2Rail y seleccione "Set one Curve".
- Cuando se le pida, seleccione la primera “rail curve” en la escena (ver imagen).
- Haga clic derecho en la entrada R2 del Sweep2Rail y seleccione "Set one Curve".
- Cuando se le pida, seleccione la segunda “rail curve” en la escena (ver imagen).
- Haga clic derecho en la entrada S del Sweep2Rail y seleccione " Set one Curve".
- Una vez más, cuando se le pida, seleccione la “section curve” de la escena.

  Si todas las curvas fueron bien seleccionadas, ahora debería ver una superficie entre las “rail curves”.

- Superficie/Freeform/Offset - Arrastrar y soltar un componente Surface Offset en el lienzo.
- Conecte la salida S del Sweep2Rail a la entrada S del Offset.
- Params/Special/Slider - Arrastre y suelte el “Numeric Slider” sobre el lienzo.
- Haga clic en el deslizador y establezca los siguientes parámetros:
  - Name: Surface Offset
  - Slider Type: Floating Point
  - Lower Limit: 0.0
  - Upper Limit: 10.0
  - Value: 10.0
- Conecte el “Numeric Slider” con la entrada D del “Surface Offset”.

Ahora debería ver una nueva superficie distanciada en 10 unidades (o cualquier valor que hayamos fijado para la superficie) de la superficie original.

- Surface/Utility/Divide Surface - Arrastre y suelte dos componentes “Divide Surface” en el lienzo.
- Conecte la salida S del “Sweep2Rail” a la primera entrada S de “Divide Surface”.

  Usted debería ver inmediatamente una serie de puntos sobre su primera superficie Sweep2Rail. Esto es porque el valor por defecto en U y V para el componente Divide Surface se establece en 10. Básicamente, el componente Divide Surface está creando 10 divisiones en cada dirección de la superficie, en última instancia, creando una red de puntos sobre su superficie. Si usted fuera a conectar los puntos a lo largo de cada línea de división, se obtendrían las "isocurvas" que forman el marco interno de la superficie.
- Conecte la salida S de “Surface Offset” a la entrada S de “Divide Surface”.

  Nuevamente, una red de puntos ha sido creada en la superficie offset.
- Params/Special/Slider - Arrastre y suelte dos “Numeric Slider” sobre el lienzo.
- Clic derecho sobre el primer deslizador y determine lo siguiente:
  - Name: U Divisions
  - Slider Type: Integers
  - Lower Limit: 0.0
  - Upper Limit: 100.0
  - Value: 15.0
- Clic derecho sobre el segundo deslizador y determine lo siguiente:
  - Name: V Divisions
  - Slider Type: Integers
  - Lower Limit: 0.0
  - Upper Limit: 100.0
  - Value: 25.0
- Conecte el deslizador “U Divisions” a ambas entradas U de los “Divide Surface”.
- Conecte el deslizador “V Divisions” a ambas entradas V de los “Divide Surface”.

  Los dos deslizadores ahora controlan el numero de divisiones en cada dirección de ambas superficies. Dado que ambas superficies tienen exactamente el mismo número de puntos, y por lo tanto cada punto tiene el mismo número de índice, fácilmente seremos capaces de conectar la superficie interior con la superficie exterior mediante una línea simple.
- Curve/Primitive/Line - Arrastre y suelte una componente “Line” sobre el lienzo.
- Conecte la primera salida P de “Divide Surface” a la entrada A de “Line”.
- Conecte la segunda salida P de “Divide Surface” a la entrada B de “Line”.

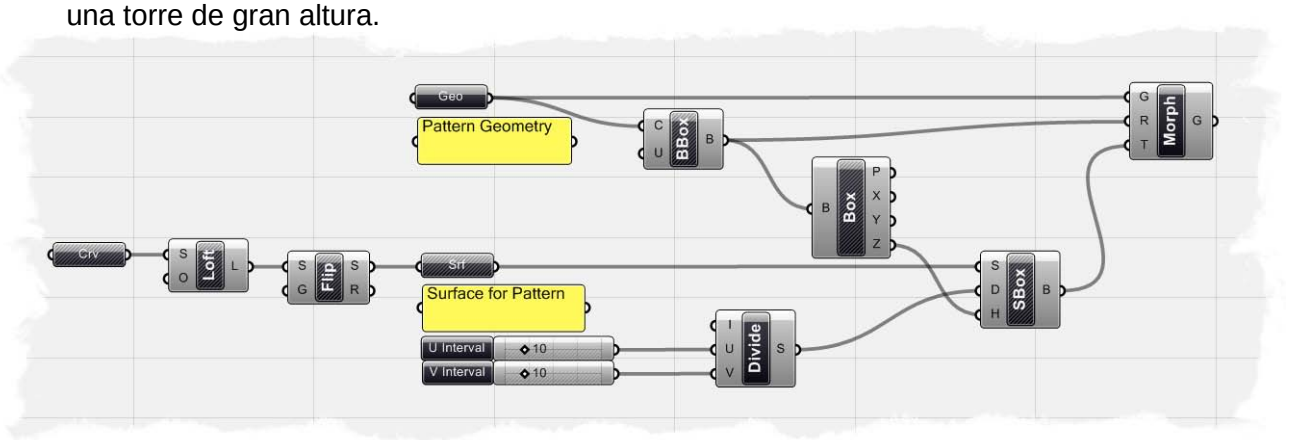
  Es tan fácil como eso. Usted debe ahora ver una serie de líneas de conexión entre cada punto de la superficie interna hasta el punto correspondiente en la superficie exterior. Podemos llevar la definición un paso más allá, dando a cada línea un espesor.
- Surface/Freeform/Pipe - Arrastre y suelte un componente “Pipe” sobre el lienzo.
- Conecte la salida L de “Line” a la entrada C de “Pipe”.
- Params/Special/Slider - Arrastre y suelte un “Numeric Slider” sobre el lienzo.
- Clic derecho sobre el deslizador y determine lo siguiente:
  - Name: Pipe Radius
  - Slider Type: Floating Point
  - Lower Limit: 0.0
  - Upper Limit: 2.0
  - Value: 0.75
- Conecte el deslizador “Pipe Radius” a la entrada R de “Pipe.

11.2 Herramientas de panelizado.

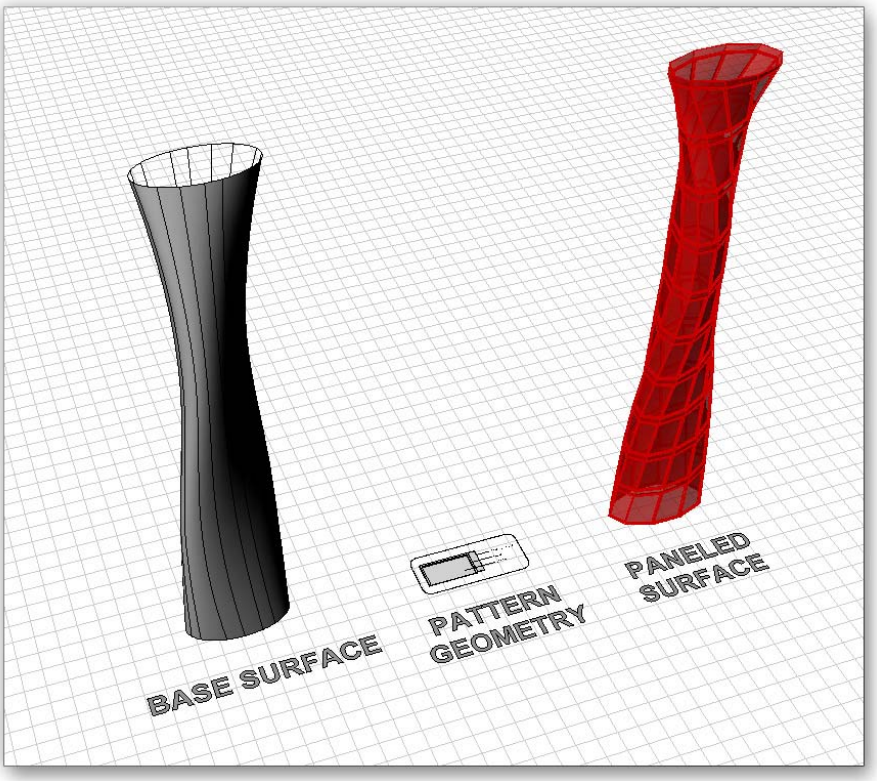
McNeel ha publicado recientemente un excelente plug-in libre para Rhino llamado "**Paneling Tools**". El plug-in, entre otras cosas, permite la capacidad de propagar módulos geométricos específicos sobre una superficie determinada. Grasshopper también tiene algunos componentes que pueden ser utilizados para re-crear el método de penalizado y en el siguiente tutorial vamos a ver la manera de subdividir una superficie mediante el uso de un componente de intervalo, y vamos a utilizar algunos de los componentes "Morphing" incluidos en Grasshopper.

Para obtener más información acerca del plug-in "Paneling Tools" , por favor visite:
http://en.wiki.mcneel.com/default.aspx/McNeel/PanelingTools.html

A continuación se muestra una captura de pantalla de la definición necesaria para copiar un patrón geométrico, como un sistema de ventanas torcidas, a través de la superficie de una torre de gran altura.



Nota: Para ver la definición final de este ejemplo, abra en Grasshopper el archivo **Paneling Tool.ghx** encontrado en la carpeta "Archivos de apoyo" que acompaña a este documento.



Para crear la definición desde el inicio, lo primero que se necesita es un conjunto de curvas para interpolar la superficie de la torre. En Rhino, abra el archivo **Panel Tool_base.3dm** también se encuentra en la carpeta “Archivos de apoyo”. Usted debe encontrar 4 elipses, que vamos a utilizar para ayudarnos a definir nuestra superficie, y una geometría base preestablecida en la escena.

Vamos a empezar la definición mediante la creación de una superficie “loft”:

- Params/Geometry/Curve – Arrastre y suelte un componente “curve” sobre el lienzo.
- Clic derecho sobre el componente “Curve” y seleccione “Set Multiple Curves”.
- Cuando se le pida, seleccione las 4 elipses en la escena, uno por uno, a partir de la curva del fondo hasta la cima.
- Pulse enter después de haber seleccionado la última elipse.

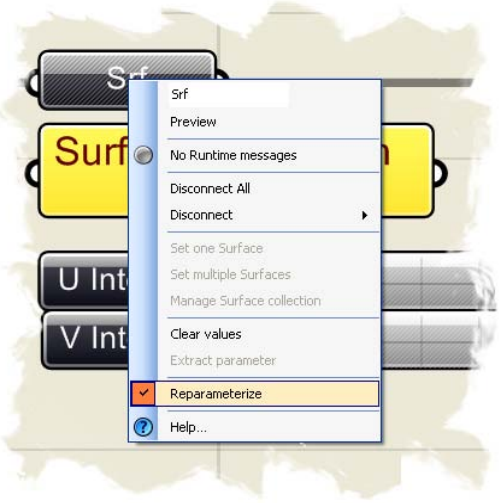
  Como habremos hecho en algunos de los ejemplos anteriores, hemos definido implícitamente las geometría de Rhino dentro de Grasshopper.
- Surface/Freeform/Loft – Arrastre y suelte un componente “Loft” sobre el lienzo.
- Conecte la salida de “Curve” a la entrada S de “Loft”.

  Si hace clic derecho sobre la entrada O “Loft”, usted encontrará las opciones típicas de “Loft” asociadas con el comando de Rhino. En nuestro caso, la configuración por defecto será suficiente, pero puede haber un momento en el cual estos parámetros tendrían que ser ajustados para adaptarse a su necesidad particular.

* **Paso opcional:** Es casi imposible decir sí o no su superficie “loft” se encuentra en la dirección correcta. Sin embargo, por ensayo y error en la creación de este tutorial, me di cuenta de que mi superficie “loft” por defecto se había creado hacia adentro, causando que todos mis paneles fueran hacia adentro también. Así que, podemos utilizar el componente “Flip” para revertir las normales de nuestra superficie para volver sus caras hacia el exterior.

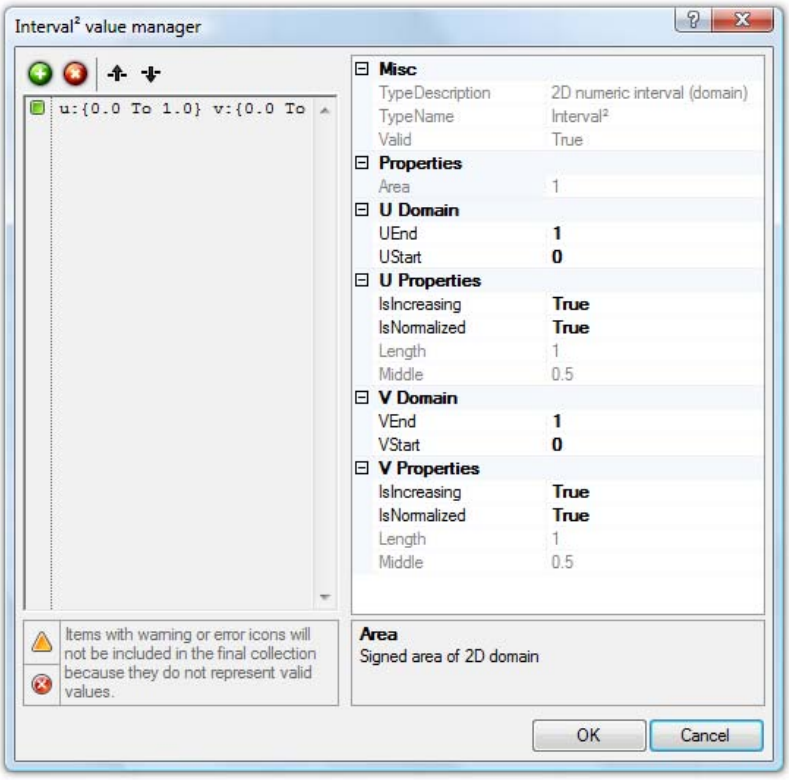
- Surface/Utility/Flip – Arrastre y suelte un componente “Flip” sobre el lienzo.
- Conecte la salida L de “Loft” a la entrada S de “Flip”.
- Params/Geometry/Surface – Arrastre y suelte un componente “Surface” sobre el lienzo.
- Conecte la salida S de “Flip” a la entrada de “Surface”.
- Clic derecho sobre el component “Surface” y selecione “**Reparameterize**”.

Actualmente, nuestra superficie posee un intervalo de dominio que va de cero (en la base de la superficie) a un número que representa la parte superior de la superficie. Realmente no importa lo cual límite superior sea, porque podemos re-parametrizar la superficie. Esto significa, que vamos a restablecer el dominio de la superficie a un intervalo de cero a uno, donde cero representa la base de la superficie y una de ellas representa el límite superior de la superficie. Este es un paso crucial, porque nuestra superficie



no se dividirá correctamente si no es re-parametrizada en un intervalo entre cero y uno.

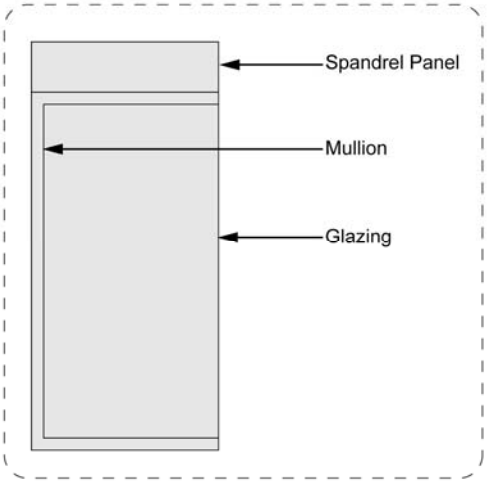
- Scalar/Interval/Divide Interval2 – Arrastre y suelte un componente “Divide Two Dimensional Interval” sobre el lienzo.
  En primer lugar, tendrá que configurar el intervalo, antes de subdividir la superficie.
- Clic derecho sobre la entrada I del componente “Divide Interval” y seleccione "Manage Interval Collection".
- Clic sobre el botón verde para añadir un intervalo a la colección.
  De forma predeterminada, el intervalo se establece como: u: (0,0 a 0,0 v): (0,0 a 0,0), pero necesitamos que el intervalo sea de 0 a 1 en ambas direcciones U y V.
- Cambie el valor de “U-End” a 1.0
- Cambie el valor de “V-End” a 1.0



Su Administrador de colección de intervalos debe ser similar a la imagen de arriba. Haga clic en OK para aceptar el intervalo. Ahora, si usted pasa el ratón sobre la entrada I de “Divide Interval”, ahora verá que tanto nuestra base de intervalos de U y V van de cero a uno, al igual que nuestra superficie re-parametrizada.

- Params/Special/Slider - Arrastre y suelte dos componentes “numeric sliders” sobre el lienzo.
- Clic derecho sobre el primer deslizador y determine lo siguiente:
  - Name: U Interval
  - Slider Type: Integers
  - Lower Limit: 5.0
  - Upper Limit: 30.0
  - Value: 10.0
- Clic derecho sobre el segundo deslizador y determine lo siguiente:

    - Name: V Interval
    - Slider Type: Integers
    - Lower Limit: 5.0
    - Upper Limit: 30.0
    - Value: 10.0
- Conecte el deslizador “U Interval” a la entrada U de “Divide Interval”.
- Conecte el deslizador “V Interval” a la entrada V de “Divide Interval”.
- Xform/Morph/Surface Box - Arrastre y suelte un componente “Surface Box” sobre el lienzo.
- Conecte la salida del componente “Surface” a la entrada S de “Surface Box”.
- Conecte la salida S “Divide Interval” a la entrada D de “Surface Box”.
- Clic derecho en los components “Curve”, “Loft”, and “Surface” y apague la función 'Preview'.



Hemos dividido nuestra superficie en 100 áreas en base a nuestros deslizadores de intervalo U y V. Lo que ocurre es que, originalmente hemos creado un intervalo que oscila entre cero a uno, que coincide con el mismo valor de intervalo de nuestra superficie. Entonces, hemos dividido ese intervalo 10 veces en la dirección U y 10 veces en la dirección V, que en última instancia han creado 100 combinaciones únicas de intervalo. Usted puede cambiar los valores de los deslizadores U y V para controlar la cantidad de subdivisiones en cada dirección de su superficie. Ahora, vamos a dar un paso atrás y crear un patrón geométrico que se puede propagar a través de cada nueva subdivisión. En la escena, se encuentra un sistema de ventanas que consiste en un panel de separación, un marco de ventana, y un panel de vidrio.
Vamos a utilizar estos tres componentes geométricos para crear un sistema de fachada en nuestra superficie.

- Params/Geometry/Geometry - Arrastre y suelte un componente “Geometry” sobre el lienzo.
- Clic derecho sobre “Geometry” y seleccione "Set Multiple Geometries".
- Cuando se le pida, seleccione el “spandrel panel”, “mullion”, y el “glazing panel” de la escena.
- Pulse enter después de seleccionar los tres componentes.
- Surface/Primitive/Bounding Box - Arrastre y suelte un componente “Bounding Box” sobre el lienzo.
- Conecte la salida de “Geometry” a la entrada C de “Bounding Box”.

El componente “Bounding Box “ nos ayudará a determinar la altura de nuestros patrones geométricos. Puesto que sólo hemos usado cajas rectangulares como nuestro patrón, la altura sería muy fácil de determinar. Sin embargo, si eligiéramos una forma más orgánica para copiar toda su superficie, la altura sería mucho más difícil de resolver. Usaremos esta información de la altura como un valor de entrada para el componente “Surface Box “. En segundo lugar, vamos a utilizar el “Bounding Box” como referencia para el component “BoxMorph” que vamos a discutir en un momento.

- Clic derecho en la entrada U del componente “Bounding Box” y determinar el “boolean value” como “True”.

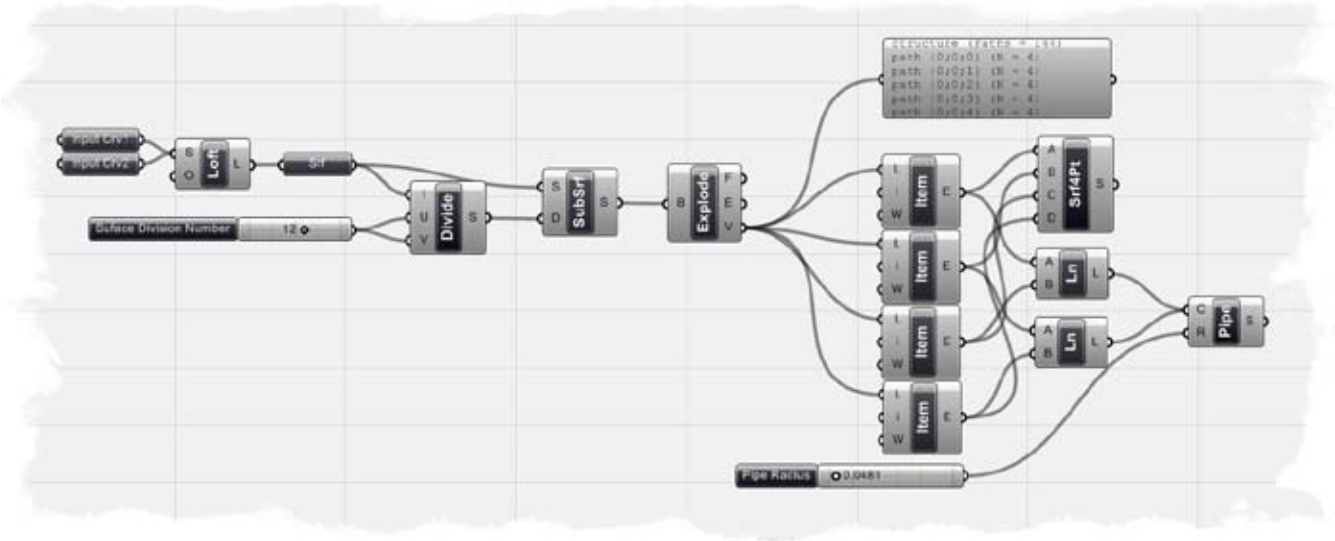
  Este es un paso importante, ya que esto hará que nuestro “Bounding Box” cree una caja única para los 3 objetos.
- Surface/Analysis/Box Components - Arrastre y suelte un componente “Box” sobre el lienzo.
- Conecte la salida B de “Bounding Box” a la entrada B del componente “Box”.
- Conecte la salida Z del componente “Box” a la entrada H de “Surface Box”.

  Ahora estamos en la recta final.
- Xform/Morph/Box Morph - Arrastre y suelte un componente “Box Morph” sobre el lienzo.
- Conecte la salida de “Pattern Geometry” a la entrada G del “Box Morph”.
- Conecte la salida B de “Bounding Box” a la entrada R de “Box Morph”.
- Conecte la salida B de “Surface Box”-B a la entrada T de “Box Morph”.
- Clic derecho en el componente “Morph Box” y determine la “data matching algorithm” en “**Cross Reference**”.
- Clic derecho en el component “Surface Box” y desactive la opción '**Preview**'.

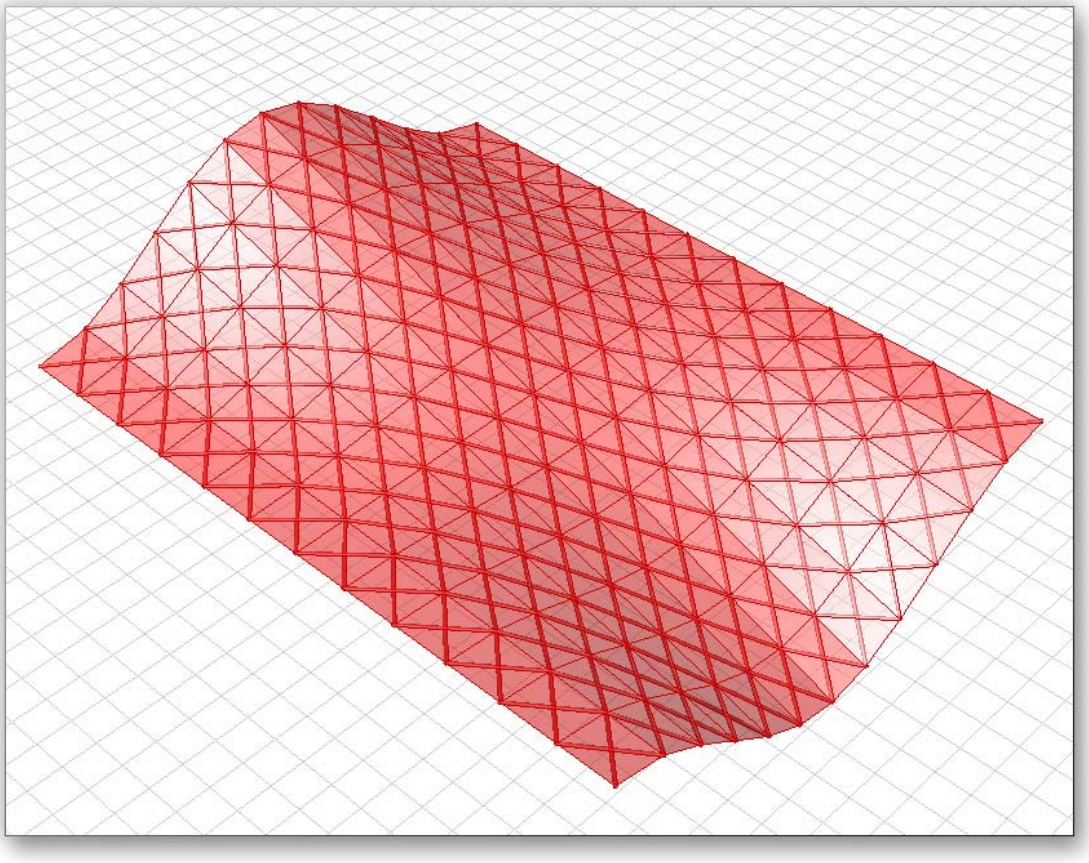
  Uff. Si su cerebro no está a punto de estallar, voy a tratar de explicar la última parte de la la definición. Tenemos la geometría del modelo ingresada en el componente “Morph Box“, que reproduce ese modelo en cada subdivisión. Hemos utilizado el “Bounding Box” como una referencia para nuestro sistema de ventanas, y hemos ingresado las 100 subdivisiones como nuestro objetivo para replicar nuestra geometría. Si ha seguido todos los pasos correctamente, usted debería ser capaz de cambiar la superficie base, la geometría de patrones, y las subdivisiones en U y V para controlar cualquier número de paneles en una superficie.

11.3 Superficies de grilla diagonal

Ya hemos demostrado cómo podemos utilizar la definición de herramientas de panelizado para crear elementos de fachada en una superficie, pero el siguiente ejemplo muestran realmente cómo podemos manipular el el flujo de información para crear una cuadrícula de diamantes estructurales, o “Diagrid” sobre cualquier superficie. Para empezar, vamos a abrir el archivo **Surface Diagrid.3dm** en Rhino. En la escena, usted encontrará dos curvas de coseno reflejadas, las cuales serán los límites de la superficie “loft”.



Nota: Para ver la definición final de este ejemplo, en Grasshopper abra el archivo **Surface Diagrid.ghx** encontrado en la carpeta “Archivos de apoyo” que acompaña a este documento.



Vamos a comenzar la definición desde el inicio:

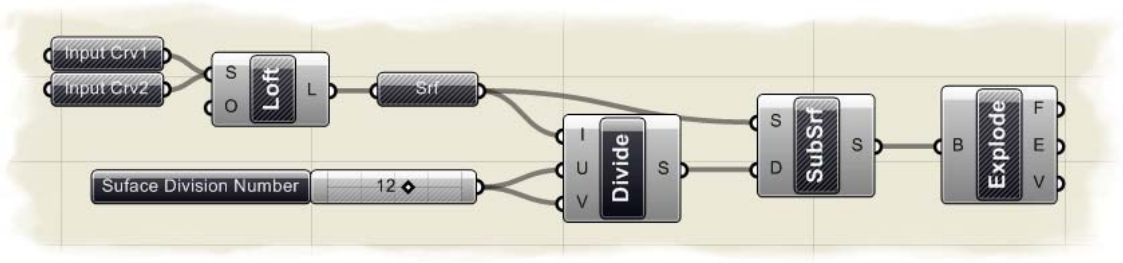
- Params/Geometry/Curve – Arrastre y suelte **dos** componentes “Curve” sobre el lienzo.
- Clic derecho sobre el primer “Curve” y renombrarlo como “Input Crv1”.
- Clic derecho sobre el “Input Crv1” y seleccione “Set One Curve”.
- Cuando se le pida, seleccione una de las curvas en la escena de Rhino.
- Clic derecho sobre el segundo “Curve” y renombrarlo como “Input Crv2”.
- Clic derecho sobre el “Input Crv2” y seleccione “Set One Curve”.
- Cuando se le pida, seleccione la otra curva en la escena de Rhino.
- Surface/Freeform/Loft – Arrastre y suelte un componente “Loft” sobre el lienzo.
- Conecte el componente “Input Crv1” a la entrada S de “Loft”.
- Mientras mantiene presionada la tecla Shift, conecte el componente “Input Crv2” a la entrada S de “Loft”.

  Ahora debería ver una superficie “loft” entre las dos curvas de entrada en la escena de Rhino.
- Params/Geometry/Surface – Arrastre y suelte un componente “Surface” sobre el lienzo.
- Conecte la salida L de “Loft” a la entrada del componente “Surface”.
- Scalar/Interval/ Divide Interval^2 – Arrastre y suelte un componente “Divide Two Dimensional” sobre el lienzo.

  Al igual que en el último ejemplo, vamos a subdividir nuestra superficie en pequeñas superficies. Para ello, debemos crear un intervalo en la dirección U y V desde los que subdividiremos la superficie.
- Conecte la salida del componente “Surface” a la entrada I de “Divide Interval”.
- Params/Special/Numeric Slider – Arrastre y suelte un deslizador sobre el lienzo.
- Clic derecho sobre el deslizador y determine lo siguiente:
  - Name: Surface Division Number
  - Slider Type: Integers
  - Lower Limit: 0.0
  - Upper Limit: 20.0
  - Value: 12.0
- Conecte el deslizador a ambas entradas U y V del componente “Divide Interval”.
- Surface/Utility/Isotrim – Arrastre y suelte un componente “Isotrim” sobre el lienzo.
- Conecte la salida de “Surface” a la entrada S de “Isotrim”.
- Conecte la salida S de “Divide Interval” a la entrada D de “Isotrim”.
- Clic derecho sobre “Loft” y “Surface” y apague la opción de ‘Preview’.

  Ahora debería ver un conjunto de superficies subdivididas correspondiente al valor de la subdivisión establecida por el control deslizante numérico. Debido a que hemos conectado un solo regulador para ambos intervalos en U y V, usted debería ver cambiar las subdivisiones igualmente en ambas direcciones. Sin embargo, usted puede agregar un control deslizante adicional si desea controlar las divisiones por separado.
- Surface/Analysis/Brep Components – Drag and drop a Brep Components onto the canvas

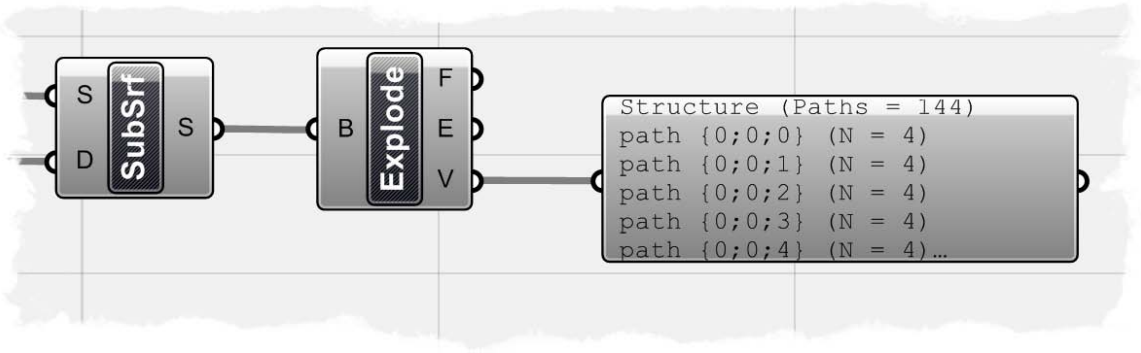
  Este componente romperá cualquier bloque agrupado y lo descompondrá en sus elementos por separado, como caras, bordes y vértices. En este caso, queremos conocer las posiciones de cada punto de vértice para que podamos conectar diagonalmente cada subdivisión.
- Conecte la salida S de “Isotrim” con la entrada B de “Brep Components”.



Nuestra definición, hasta ahora, debería ser similar a la imagen de arriba. En esencia, hemos subdividido nuestra superficie en sub-superficies y explotado cada sub-superficie para obtener la posición de cada punto vértice. En nuestro siguiente paso, vamos a ver en nuestra estructura de árbol el índice de cada uno de los puntos vértices.

- Param/Special/Parameter Viewer - Drag and drop a Parameter Viewer component onto the canvas
- Connect the Brep Components-V output to the Parameter Viewer input

El Visor de parámetros nos ayudará a examinar nuestra estructura de árbol. La estructura para este ejemplo particular, nos dice que tenemos 144 rutas, con cada rama terminal con 4 entradas de datos (en nuestro caso, estos son los cuatro puntos vértices de cada superficie subdividida). Podemos utilizar el componente "List Item" para entrar en cada ruta, o sub-ruta de acceso y recuperar una base de datos de entrada específica. Para crear nuestra grilla diagonal, queremos saber dónde está el primero, segundo, tercer, y cuarto punto de cada superficie subdividida en el espacio cartesiano.



- Logic/List/List Item - Arrastre y suelte **cuatro** "Parameter Viewer" sobre el lienzo.
- Conecte la salida V de "Brep Components" a las cuatro entradas L de "List Item".
- Clic derecho sobre el input i del primer "List Item" y coloque el "Integer" en 0.
- Clic derecho sobre el input i del segundo "List Item" y coloque el "Integer" en 1.
- Clic derecho sobre el input i del tercer "List Item" y coloque el "Integer" en 2.
- Clic derecho sobre el input i del cuarto "List Item" y coloque el "Integer" en 3.

Usted probablemente ha notado que hemos aumentado el número índice para cada "List Item" en 1 cada vez. Ya que nuestra estructura de árbol ha sido configurada y Grasshopper entiende que tenemos 144 rutas diferentes, cada una con 4 ítems al final de cada ruta... todo lo que necesitamos hacer es entrar en cada ruta y recuperar el primer punto, luego recuperar el segundo, y así sucesivamente. Así que los 4 componentes "List Item" van a definir cada punto vértice de la sub-superficie, y ahora que los tenemos separados, podemos conectar los puntos opuestos con una línea para crear nuestra grilla diagonal.

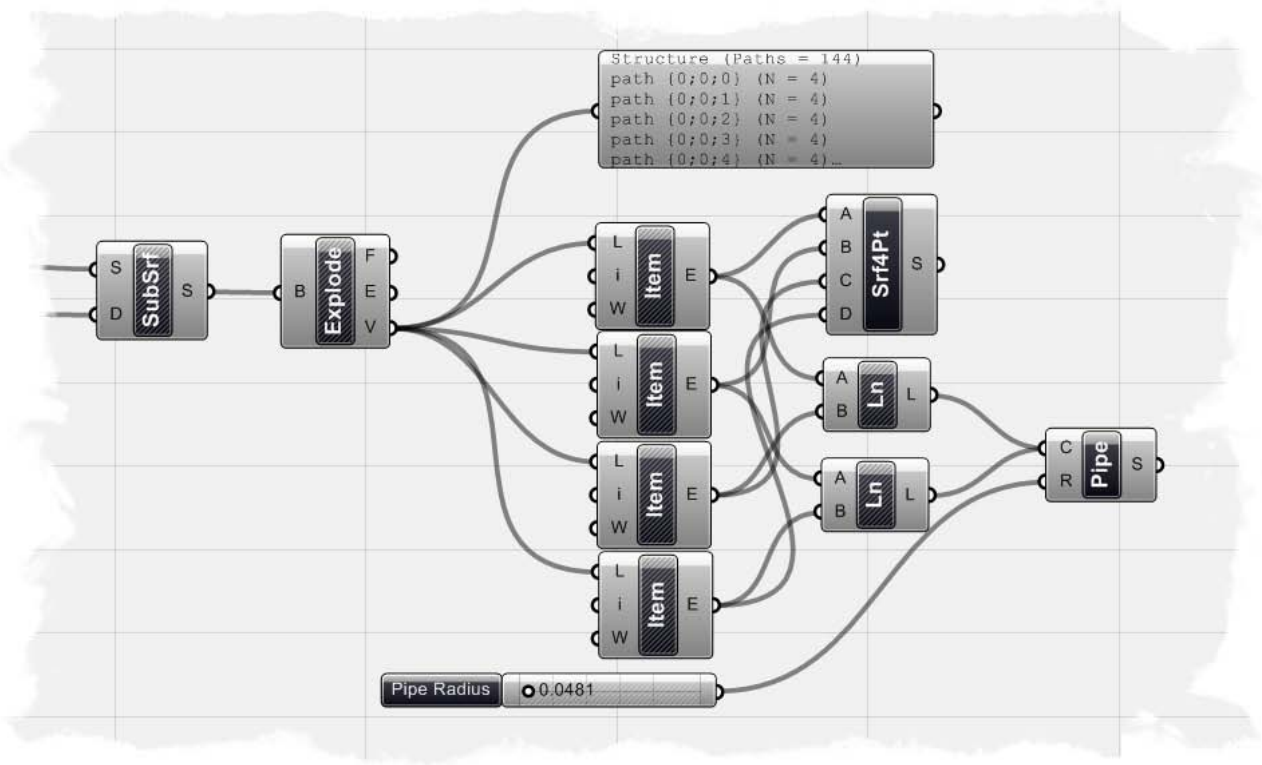
- Curve/Primitive/Line - Arrastre y suelte **dos** componentes “Line” sobre el lienzo.
- Conecte la salida E del primer “List Item” a la entrada A del primer “Line”.
- Conecte la salida E del tercer “List Item” a la entrada B del primer “Line”.
- Conecte la salida E del segundo “List Item” a la entrada A del segundo “Line”.
- Conecte la salida E del cuarto “List Item” a la entrada B del segundo “Line”.

  En este punto, usted debe ver un conjunto de líneas en su superficie que representan la estructura de la grilla diagonal.
- Surface/Freeform/Pipe - Arrastre y suelte un componente “Pipe” sobre el lienzo.
- Conecte la salida L del primer “Line” a la entrada C de “Pipe”.
- Mientras mantiene presionada la tecla Shift, conecte la salida L de la segunda “Line” a la entrada C de “Pipe”.

  Para crear una estructura de espesor variable usaremos un deslizador para controlar el radio del “Pipe”.
- Params/Special/Number Slider – Arrastre y suelte un deslizador sobre el lienzo.
- Clic derecho sobre el deslizador y determine lo siguiente:
  - Name: Pipe Radius
  - Slider Type: Floating Point
  - Lower Limit: 0.0
  - Upper Limit: 1.0
  - Value: 0.05
- Conecte la salida del “Pipe Radius” a la entrada R de “Pipe”.

  Por último, vamos a crear una superficie plana entre cada uno de los 4 puntos vértices. Estamos haciendo esto, porque la sub-superficie que originalmente se creó es de doble curvatura y para este ejercicio nos gustaría crear una superficie panelizada como un sistema estructural de grilla diagonal.
- Surface/Freeform/4Point Surface – Arrastre y suelte un componente “4 Point Surface” sobre el lienzo.
- Conecte la salida E del primer “List Item” a la entrada A de “4 Point Surface”.
- Conecte la salida E del segundo “List Item” a la entrada B de “4 Point Surface”.
- Conecte la salida E del tercer “List Item” a la entrada C de “4 Point Surface”.
- Conecte la salida E del cuarto “List Item” a la entrada D de “4 Point Surface”.

  Asegúrese de haber apagado la vista previa de todos los componentes, excepto para el “4 Point Surface” y el componente “Pipe”.

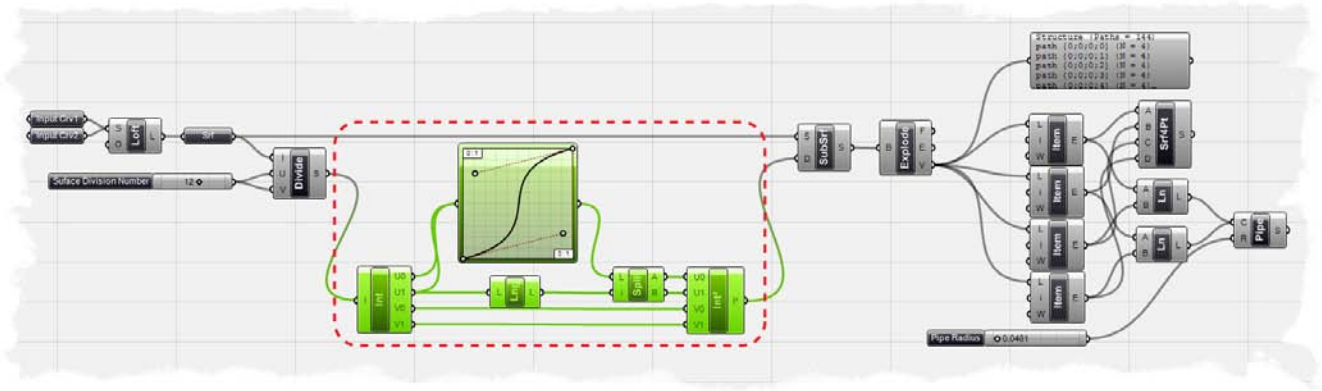


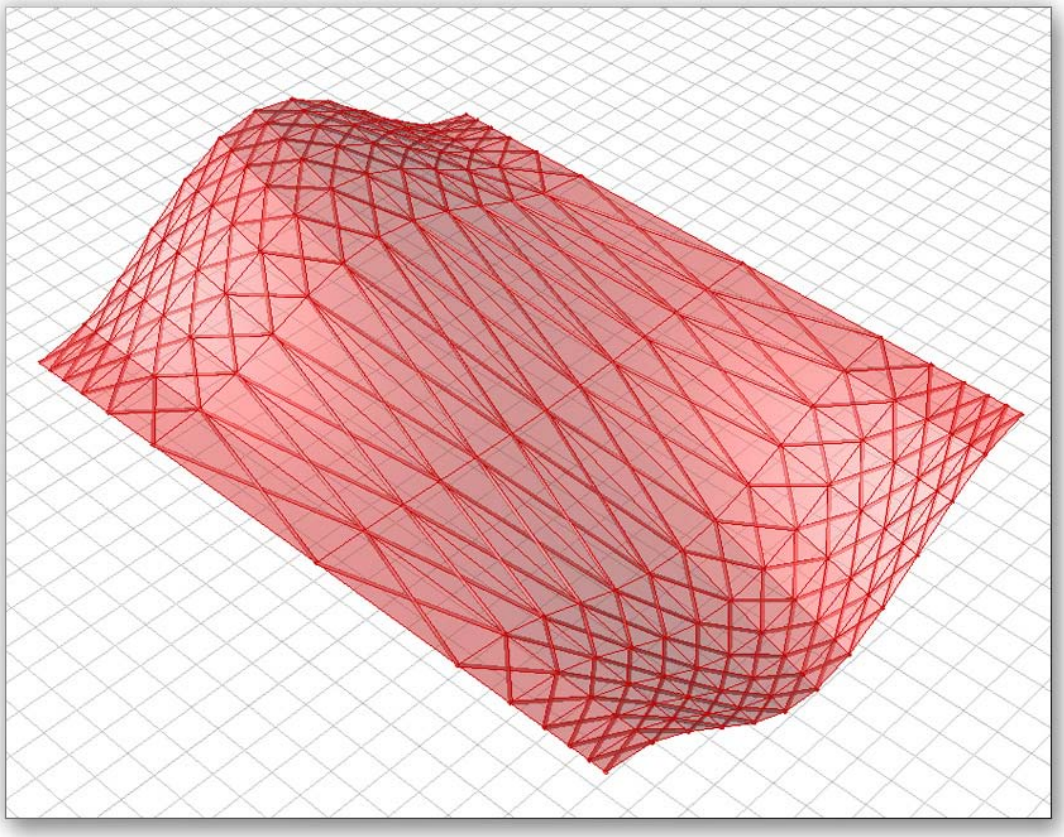
La última parte de su definición debe ser similar a la imagen de arriba. Esta definición funcionará en cualquier tipo de superficie y usted puede reemplazar la parte de "Loft"de la definición en el principio, por otros métodos más complejos de generación de superficies.

11.4 Superficies de grilla diagonal no-uniforme

Hemos mostrado en el último ejemplo cómo podemos subdividir una superficie para crear una estructura diagonal uniformemente espaciada. Esta uniformidad ha sido creada por intervalos iguales... Sin embargo, podemos utilizar algunos componentes nuevos, y un “Graph Mapper” para controlar nuestra colección de intervalo, y también el espacio de la grilla. Este tutorial se basará en el ejemplo anterior, así que asumiré que ya han construido la definición de la grilla uniforme. A continuación se muestra una captura de pantalla de la definición acabada para la grilla diagonal no-uniforme, y los componentes que vamos a añadir a la definición que se muestran en verde.

Nota: Para ver la definición final de este ejemplo, en Grasshopper abra el archivo **Uneven Surface Diagrid.ghx** que se encuentra en la carpeta “Archivos de apoyo” que acompaña a este documento.

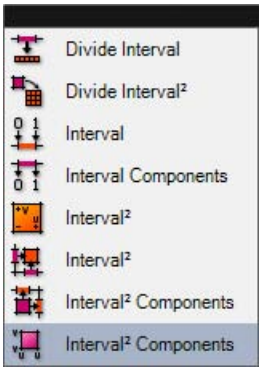




En la última definición, conectamos el componente “Divide Interval” al componente “Isotrim”. Empezaremos nuestra definición desactivando dicha conexión.

- Clic derecho en la entrada D de “Isotrim” y seleccione "Disconnect All".

  Debido a que se van a introducir una serie de nuevos componentes, es probablemente una buena idea seleccionar todos los componentes detrás de “Isotrim” y moverlos más hacia el lado derecho del espacio de trabajo.



- Scalar/Interval/Interval Components - Arrastre y suelte un componente “Interval Components” sobre el lienzo. (El que descompone el intervalo en 4 números).
- Conecte la salida S de “Divide Interval” a la entrada I de “Interval Components”.
- Params/Special/Graph Mapper - Arrastre y suelte un componente “Graph Mapper” sobre el lienzo.
- Conecte la salida U0 a la entrada del “Graph Mapper”.
- Manteniendo presionada la tecla Shift, conecte la salida U1 a la entrada de “Graph Mapper”.
- Clic derecho sobre el “Graph Mapper” y seleccione un tipo de grafico.

  Creo que el tipo de grafico “bezier” es el que mejor funciona en esta situación. Usted puede ajustar el gráfico moviendo el “Bézier” en todo el “Graph Mapper”. Hasta ahora básicamente lo que hemos hecho es descomponer nuestros intervalos espaciados uniformemente en una lista donde se puede acceder a cada uno de nuestros valores de U y V. Hemos alimentado nuestra lista de datos U en el “Graph Mapper”, el cual reorganizará la lista. En última instancia vamos a tomar estos datos reorganizados y montarlos de nuevo en un intervalo que pode-mos volver a alimentar dentro del componente de “Isotrim”.

- Logic/List/List Length - Arrastre y suelte un componente “List Length” sobre el lienzo.
- Conecte la salida U1 a la entrada L de “List Length”.
- Logic/List/Split List - Arrastre y suelte un “Split List” sobre el lienzo.
- Conecte la salida de “Graph Mapper” a la entrada L de “Split List”.
- Conecte la salida L de “List Length” a la entrada i de “Split List”.
- Scalar/Interval/Interval 2d - Arrastre y suelte un componente “Interval 2d” sobre el lienzo.
- Conecte la salida A de “Split List” a la entrada U0 de “Interval 2d”.
- Conecte la salida B de “Split List” a la entrada U1 de “Interval 2d”.
- Conecte las salidas V0 y V1 del interval descompuesto a las entradas V0 y V1 (respectivamente) del componente “Interval 2d”.

  Ya que conectamos las dos listas en la entrada del “Graph Mapper”, nece-sitábamos dividir la lista en dos, de modo que podríamos alimentar a los datos de la lista correcta de nuevo en el intervalo de dos dimensiones. Hemos conectado todos los valores de U en el “Graph Mapper”, así que nuestra subdivisión en U será controlada por la curva de “Bezier”.

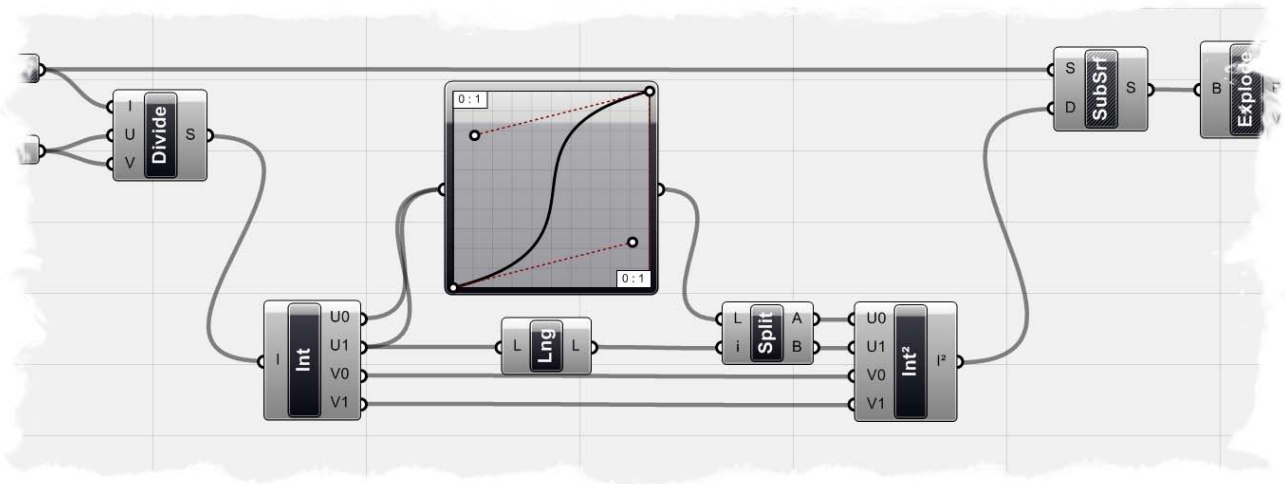
  Ya que hemos conectado el V0 y V1, directamente desde intervalo descompuesto en el intervalo de dos dimensiones, no habrán cambios en esa dirección.

  Sin embargo, usted podría seguir los mismos pasos y conectar otro

“Graph Mapper” a las listas V0 y V1 y controlar las subdivisiones en V de la misma manera. Ahora, todo lo que tenemos que hacer es conectar nuestra nueva colección de intervalo creada con el componente “Isotrim”.

- Conecte la salida I^2 del “Interval 2d” a la entrada D de “Isotrim”.

  Ya que la parte final de la definición es la misma, nuestro tipo de gráfico ahora controlará el espaciado de nuestras subdivisiones de nuestra grilla diagonal. Usted puede ajustar la curva de “Bezier” para aumentar la cantidad de estructura necesaria para apoyar su superficie. La parte central de la definición debería parecerse a la imagen de abajo.



12 *Introducción al scripting*

La funcionalidad de Grasshopper se puede ampliar usando los componentes de scripting para escribir códigos usando los lenguajes de programación VB DotNet o C#. Probablemente habrá soporte para más lenguajes en el futuro. Los códigos de usuario se colocan dentro de una clase generada dinámicamente mediante plantillas que se ensamblan utilizando el compilador de CLR que funciona con el marco DotNET. Esta función existe sólo en la memoria de la computadora y no se descarga hasta que sale de Rhino.

El componente script en Grasshopper tiene acceso a clases de Rhino DotNET SDK y funciones que son lo que los desarrolladores utilizan para construir sus plug-ins para Rhino. Como de hecho, Grasshopper es un plugin de Rhino que está completamente escrito como un plugin DotNET usando el mismo SDK que permite acceder a estos componentes de secuencias de comandos.

Pero ¿por qué molestarse utilizando los componentes de secuencia de comandos para empezar? De hecho, puede que nunca necesite utilizar uno, pero hay algunos casos en los que es posible que lo necesite. Un caso sería para lograr una funcionalidad que no es posible con otros componentes de Grasshopper. O si usted está escribiendo un sistema generativo que utiliza funciones recursivas como los fractales.

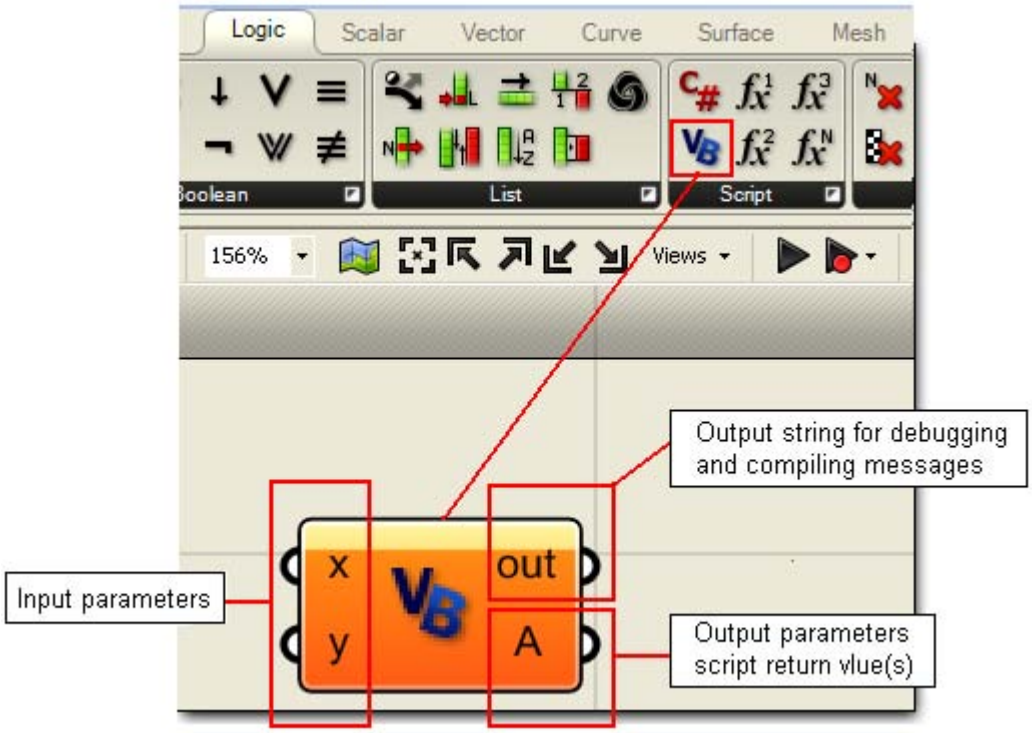
Este manual ofrece una visión general de cómo utilizar los componentes de secuencias de comandos en Grasshopper utilizando el lenguaje de programación VB dotNET. Incluye tres secciones, la primera se refiere a la interfaz del componente de script,el segundo incluye una revisión rápida del lenguaje VB DotNET. La siguiente sección se trata de Rhino DotNET SDK, clases de geometría y funciones de utilidad. Al final, hay una lista de donde se puede ir para una ayuda adicional.

13 Interface del scripting

13.1 Dónde encontrar los componentes de Script

El componente de script VB DotNet se encuentra bajo la pestaña de lógicas. Actualmente hay dos componentes de script. Uno es para escribir códigos de Visual Basic y el otro para C#. No hay duda que en el futuro habrá más lenguajes de secuencias de comandos soportados.

Para añadir un componente de secuencia de comandos sobre el lienzo, arrastrar y soltar el icono del componente.



El componente de secuencia de comandos por defecto dispone de dos entradas y dos salidas. El usuario puede cambiar nombres, tipos y número de entradas y salidas.

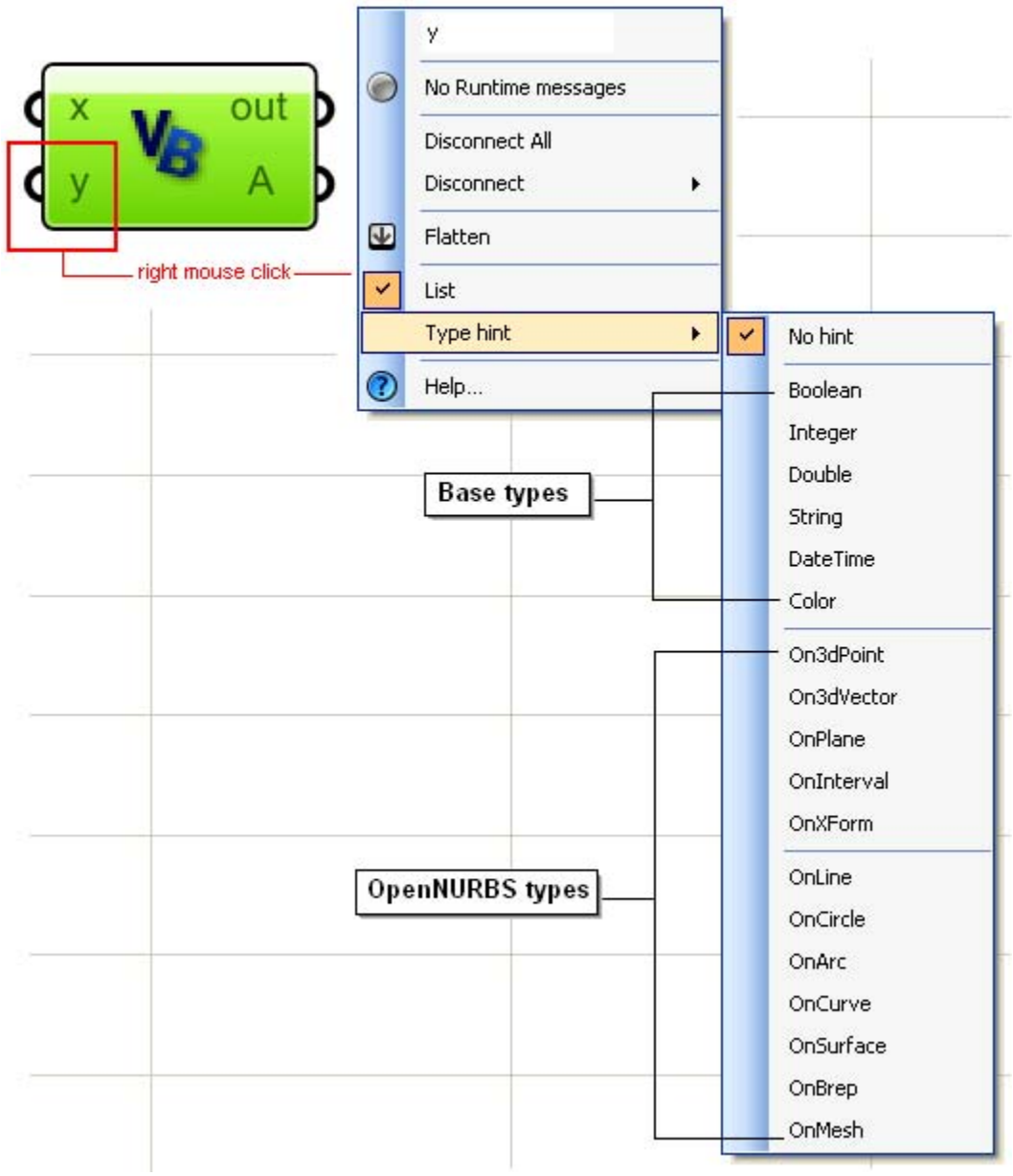
- **X**: primera entrada de un tipo genérico (objeto).
- **Y**: segunda entrada de un tipo genérico (objeto).
- **Out**: cadena de salida con los mensajes de compilación.
- **A**: salida de retorno del tipo de objeto.

13.2 Parámetros de entrada

Por defecto, hay dos parámetros de entrada: X e Y. Es posible editar los parámetros de nombres, eliminar o agregarlos y también asignar un tipo. Si hace clic derecho sobre cualquiera de los parámetros de entrada, verá un menú que tiene el siguiente texto:

- **Nombre del parámetro**: usted puede hacer clic en él y escribir un nuevo nombre.

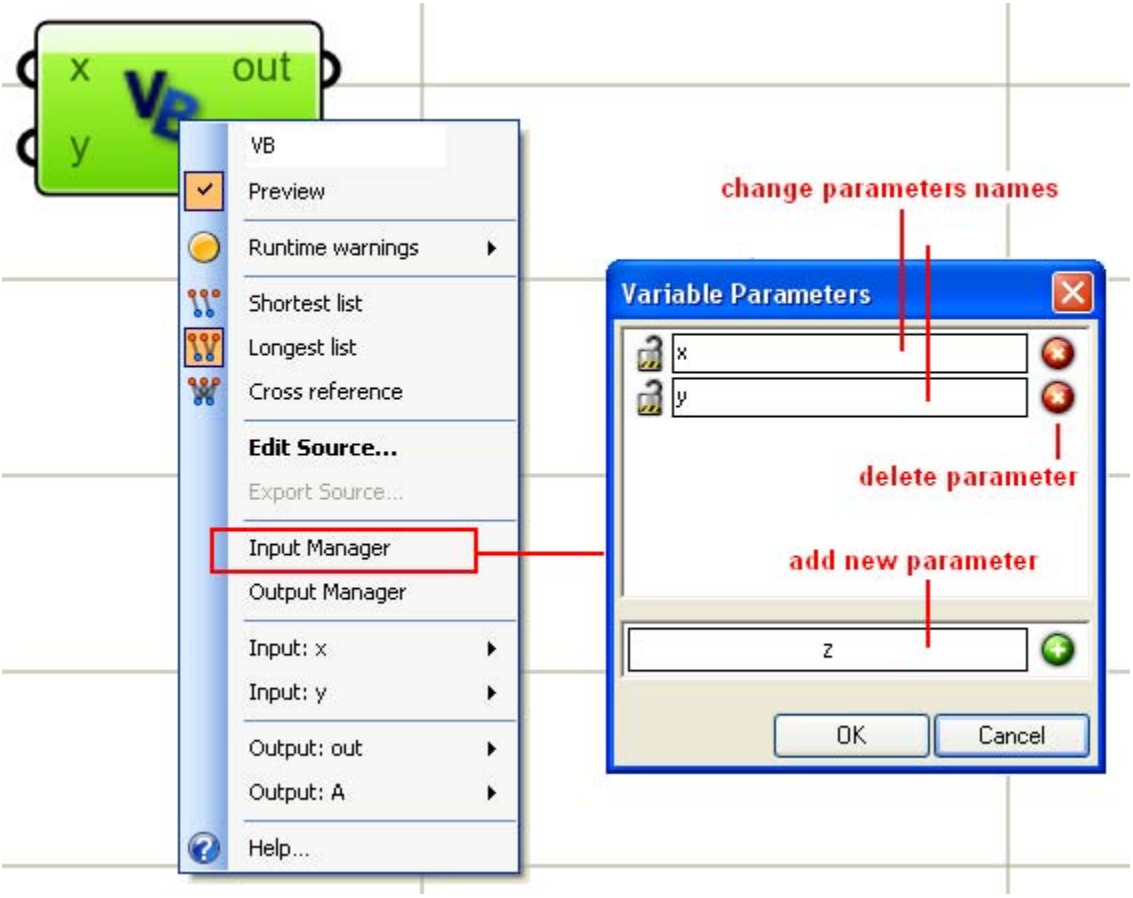
- **Run time message**: para los errores y advertencias.
- **Disconnect** and **Disconnect All**: trabaja del mismo modo que en otros componentes de Grasshopper.
- **Flatten**: para aplanar los datos. En el caso de una lista jerarquizada de datos, la convierte en una serie única de elementos.
- **List**: para indicar si la entrada es una lista de datos.
- **Type hint**: Los parámetros de entrada se establecen por defecto como tipo genérico "objeto". Es mejor especificar un tipo para hacer el código más legible y eficiente. Los tipos que comienzan con "On" son tipos de OpenNURBS.



Los parámetros de entrada también se pueden gestionar desde el menú principal del componente. Haciendo clic derecho en el centro del componente, se obtiene un menú que contiene los detalles de las entradas y salidas.

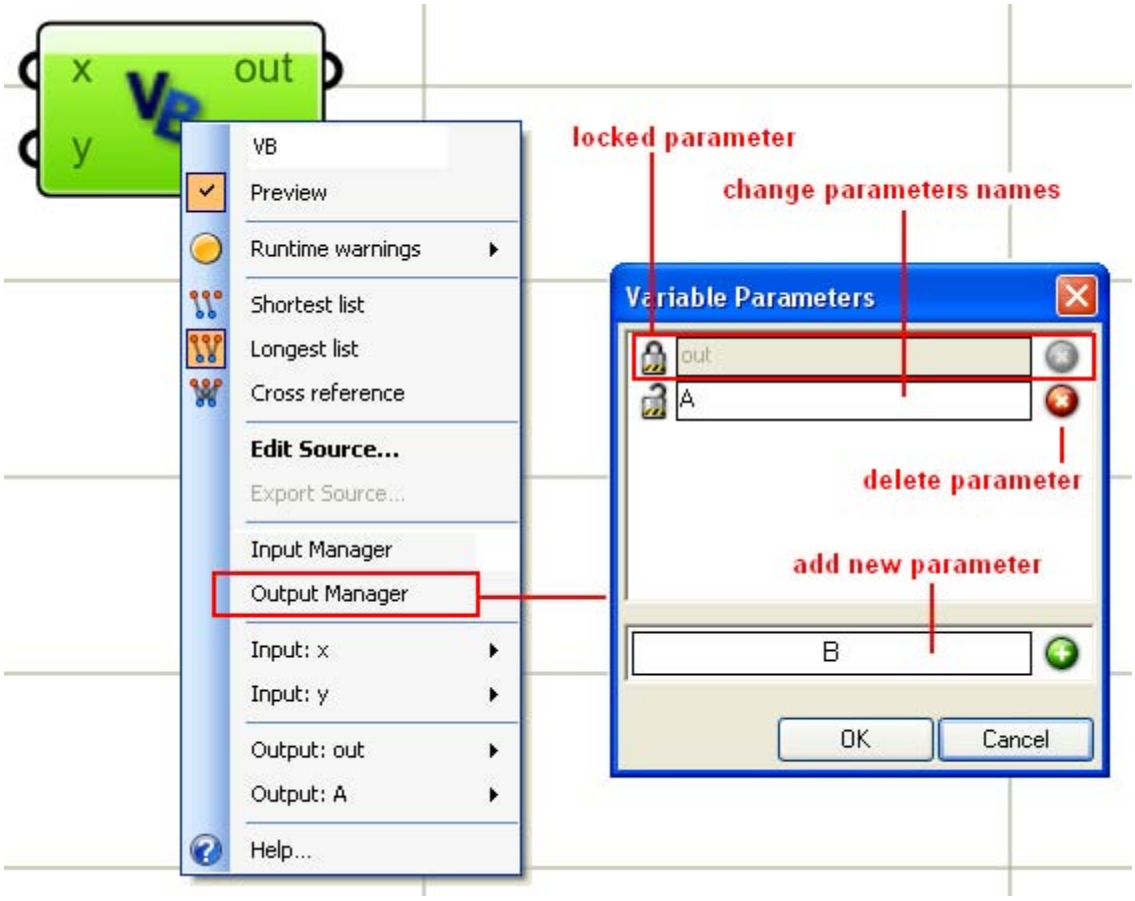
Usted puede usar este menú para abrir el administrador de entrada y cambiar los nombres de los parámetros, añadir otras entradas nuevas o eliminarlas, como se muestra en la imagen.

Tenga en cuenta que su firma de función de secuencias de comandos (parámetros de entrada y sus tipos) sólo puede ser cambiada a través de este menú. Una vez que comience la edición de la fuente, sólo el cuerpo de la función puede ser cambiado y no los parámetros.



13.3 Parámetros de salida

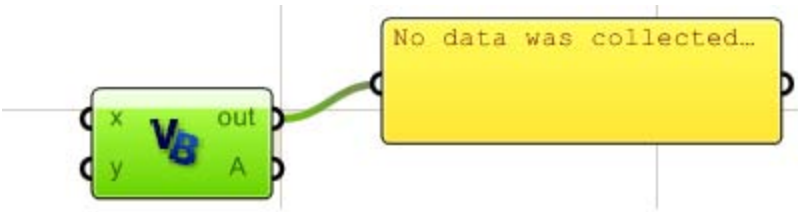
También se pueden definir tantas salidas o retornos como desee usando el menú principal del componente. A diferencia de los parámetros de entrada, no hay tipos asociados con la salida. Son definidos como el tipo de sistema genérico de "objeto" y la función puede asignar cualquier tipo, variedad o no a alguna de las salidas. La siguiente imagen muestra cómo establecer las salidas usando el administrador de salidas. Tenga en cuenta que el parámetro "out" no se puede eliminar. Posee cadenas de depuración y cadenas de depuración de usuario.



13.4 Ventana de salida y depuración de información

La ventana de salida que se llama "out", por defecto está ahí para proporcionar la información de depuración. Ella elabora una lista de todos los errores y advertencias. El usuario también puede imprimir valores a ella desde dentro del código para ayudar a depurarlos. Es muy útil para leer los mensajes compilados cuidadosamente cuando el código no se ejecuta correctamente.

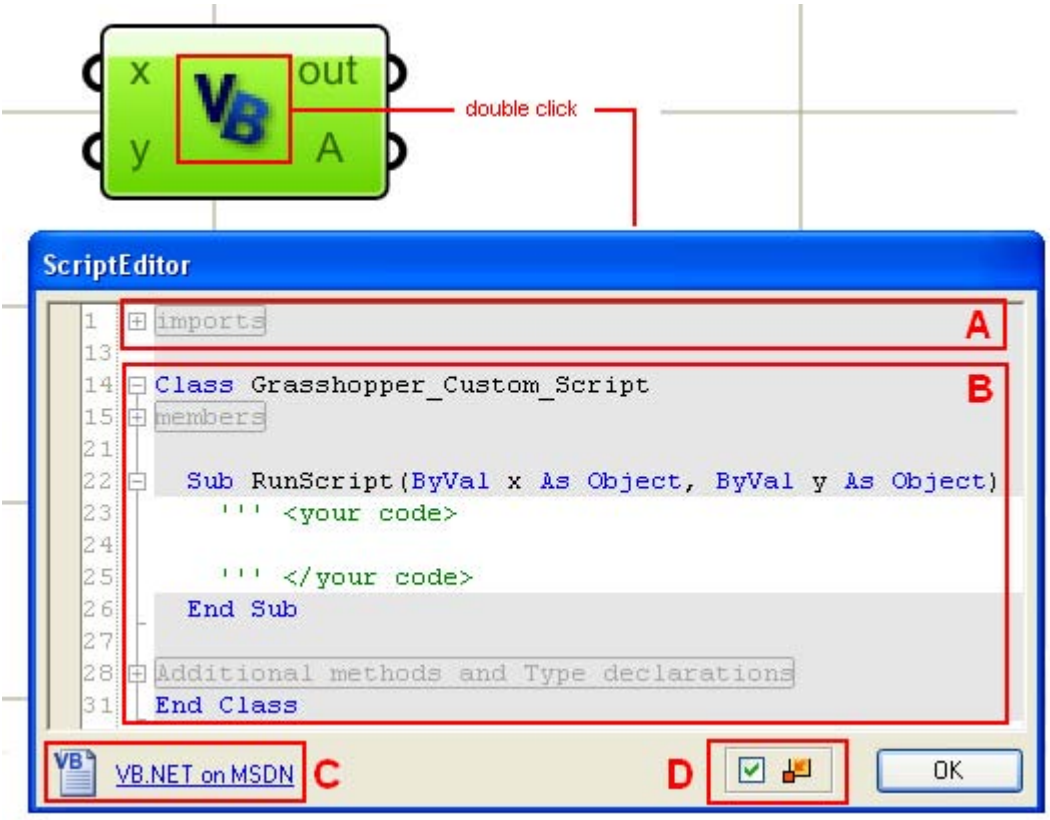
Es una buena idea el conectar la cadena de salida a un componente de texto para ver los mensajes de compilación y depurar la información directamente.



13.5 Dentro del Componente script

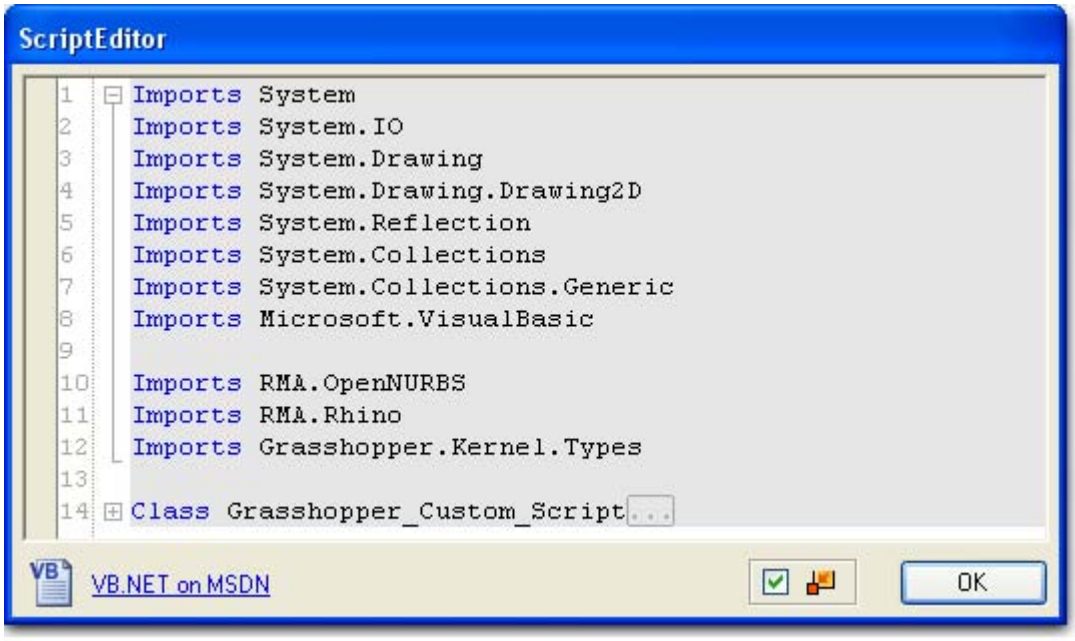
Para abrir el componente de script , haga doble clic en el centro de la componente de script o seleccione " Edit Source…" del menú de componentes. El componente de script consta de dos partes. Estas son:

A: Imports
B: Grasshopper_Custom_Script class.
C: Link to Microsoft developer network help on VB.NET.
D: Check box to activate the out of focus feature of scripting component.



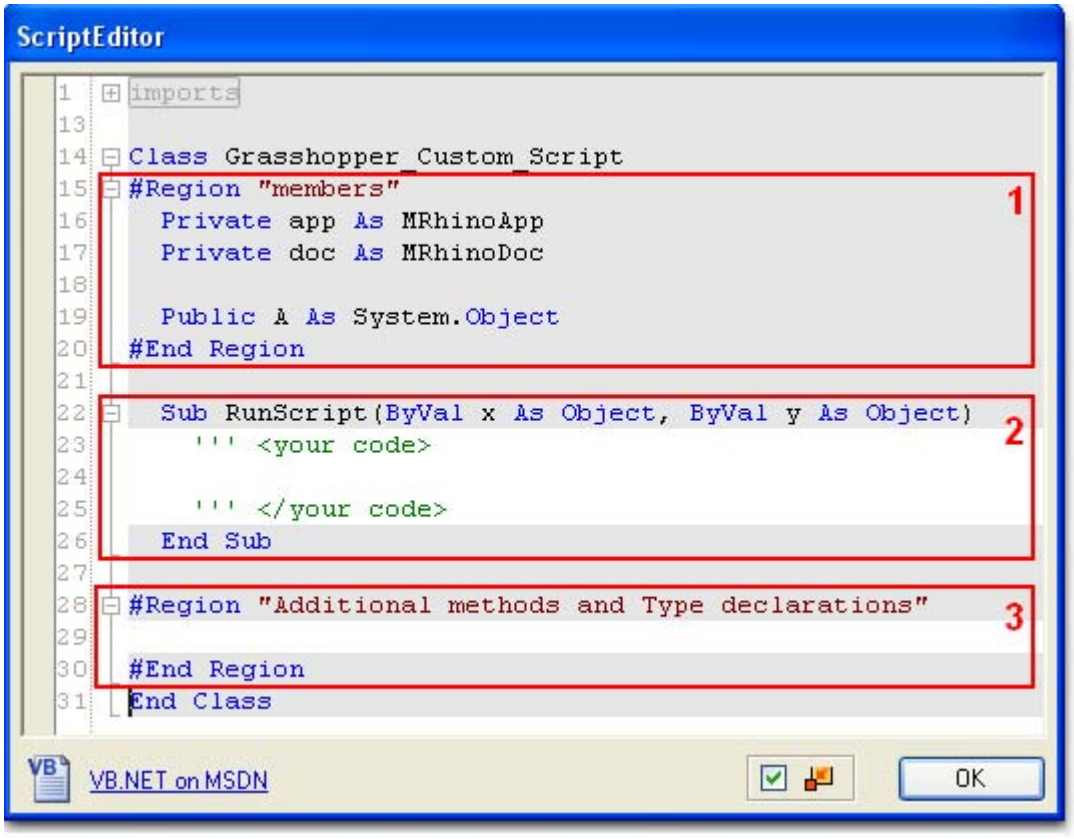
A: Imports

Las importaciones son dlls externos que puedes utilizar en tu código. La mayoría de ellos son sistemas de importaciones DotNET, pero también existen dos dlls en Rhino: RMA.openNURBS y RMA.Rhino.
Estos incluyen toda la geometría y funciones de utilidad de Rhino. Hay también tipos específicos de Grasshopper.



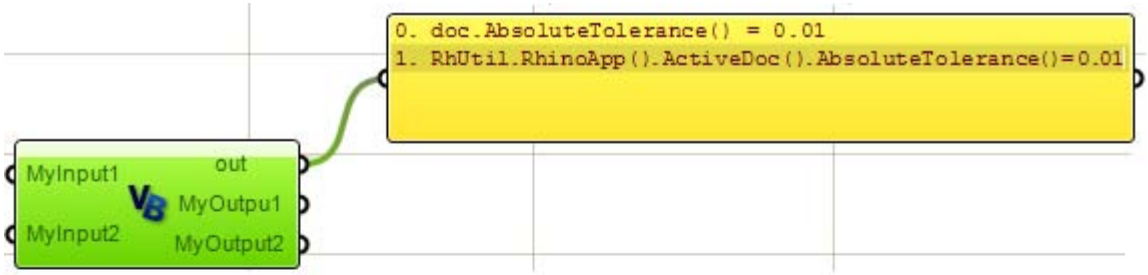
B: Grasshopper_Custom_Script Class

Grasshopper_Custom_Script class consiste de tres partes:



1. **Members:** Esto incluye dos referencias, una a la aplicación actual de rhino (APP) y la otra para el documento activo (doc). La aplicación y el documento de Rhino también se pueden acceder directamente a través de RhinoUtil. La región “Members” también incluye los valores de retorno o la salida de la función de secuencia de comandos. El retorno se define como un tipo de sistema de genérico y el usuario no puede cambiar ese tipo.
2. **RunScript**: Función principal dentro de la cual el usuario escribe su código.
3. **Additional methods and type declarations**: Aquí es donde usted puede poner sus funciones y tipos.

El siguiente ejemplo muestra dos formas de acceder a la toleracncia absoluta del documento. La primera es mediante el uso de la referencia del documento que viene con el componente de secuencia de comandos (doc) y la segunda es a través de RhUtil (Rhino funciones de utilidad). Observe que cuando se imprime el valor de la tolerancia a la ventana de salida, ambas funciones de ceden el mismo resultado. También tenga en cuenta que en este ejemplo hay dos salidas (MyOutput1 y MyOutput2). Estos se enumeran en la region “Members” de la clase de script.




```
Class Grasshopper_Custom_Script
#Region "members"
  Private app As MRhinoApp
  Private doc As MRhinoDoc

  Public MyOutput1, MyOutput2 As System.Object
#End Region

  Sub RunScript(ByVal MyInput1 As Object, ByVal MyInput2 As Object)
    ''' <your code>
    Dim tol As Double

    tol = doc.AbsoluteTolerance()
    Print(" doc.AbsoluteTolerance() = " & tol)

    tol = RhUtil.RhinoApp().ActiveDoc().AbsoluteTolerance()
    Print(" RhUtil.RhinoApp().ActiveDoc().AbsoluteTolerance()=" & tol)

    ''' </your code>
  End Sub
```

14 Visual Basic DotNET

14.1 Introducción

Hay un montón de referencias acerca de VB.NET disponibles en Internet y e impresas. La siguiente pretende ser una revisión rápida de los elementos esenciales que necesitará para usar en su código.

14.2 Comentarios

Es una práctica muy buena el comentar su código tanto como sea posible. Se sorprenderá de lo rápido que se le olvida lo que hiciste en VB.NET, puede utilizar un apóstrofe para indicar que el resto de la línea es un comentario y el compilador lo ignorará. En Grasshopper, los comentarios son de color gris. Por ejemplo:

```
'This is a comment… I can write anything I like!
'Really… anything
```

14.3 Variables

Usted puede pensar en variables como contenedores de datos. Diferentes variables tienen diferentes tamaños dependiendo del tipo de datos que acomodan. Por ejemplo, una variable int32 reservas 32 bits en la memoria y el nombre de la variable en el nombre de ese contenedor.
Una vez que se define una variable, el resto del código puede recuperar los datos del contenedor utilizando el nombre de esa variable.

Vamos a definir un contenedor o una variable llamada "x" de tipo Int32 e inicializarla a "10". Después de esto, vamos a asignar el nuevo valor entero "20" a x. Así es como se ve en VB DotNet:

```
Dim x as Int32 = 10
'If you print the value of x at the point, then you will get 10
x = 20
'From now on, x will return 20
```

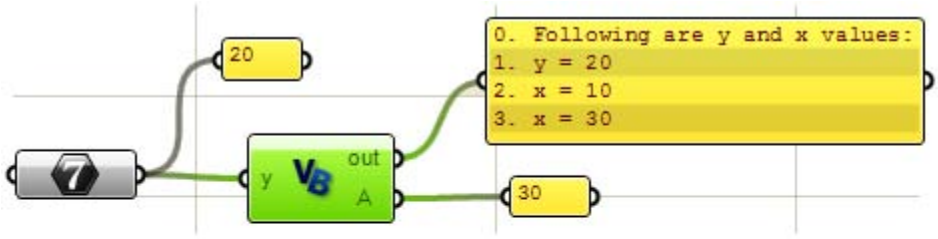
Éstos son otros ejemplos de tipos de uso común:

```
Dim x as Double = 20.4          'Dim keyword means that we are about to define a variable
Dim b as Boolean = True         'Double, Boolean and String are all examples of base or
Dim name as String = "Joe"      ' system defined types
```

El ejemplo siguiente de Grasshopper utiliza tres variables:

x: es una variable entera que se define en el código.
y:s una variable entera ingresada a la función como una entrada.
A: es la variable de salida.

El ejemplo imprime los valores de las variables a la ventana de salida a través del código. Ya que se mencionó antes, esta es una buena manera de ver lo que está sucediendo dentro de su código y depuración con la esperanza de reducir al mínimo la necesidad de un editor externo.




```
Sub RunScript(ByVal y As Integer)
  'Print variables values
  Print("Following are y and x values:")

  'Print input value (y)
  Print("y = " & y)

  'Declare x as an integer variable
  Dim x As int32 = 10

  'Print x initial value
  Print("x = " & x)

  'Set x value to be whatever was there plus input y
  x = x + y
  Print("x = " & x)

  'Assign x to output
  A = x
End Sub
```

Asignando nombres variables significativos que puedes reconocer rápidamente hará que el código sea más legible y más fácil de depurar. Vamos a tratar de atenernos a algunas buenas prácticas de codificación en los ejemplos en este capítulo.

14.4 Matrices y listas

Hay muchas maneras de definir matrices en VB.NET. Existen matrices unidimensionales y múltiples. Usted puede definir el tamaño de las matrices o el uso de matrices dinámicas. Si usted sabe de antemano el número de elementos de la matriz, puede declarar matrices de tamaño definido de esta manera:

```
'One dimensional array
Dim myArray(1) As Integer
myArray (0) = 10
myArray (1) = 20
```

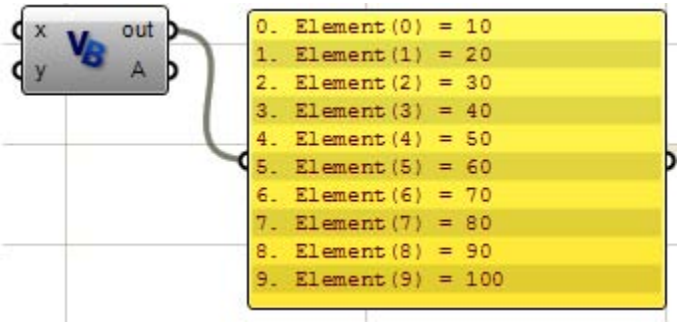
```
'Two-Dimensional array
Dim my2DArray (1,2) As Integer
my2DArray (0, 0) = 10
my2DArray (0, 1) = 20
my2DArray (0, 2) = 30
my2DArray (1, 0) = 50
my2DArray (1, 1) = 60
my2DArray (1, 2) = 70
```

```
'Declare and assign values
Dim myArray() As Integer  = {10,20}
```

```
'Declare and assign values
Dim my2DArray(,)As Integer = {{10,20,30},{40,50,60}}
```

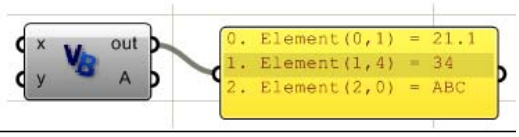
Tenga en cuenta que las matrices en VB.NET son de base cero, así que cuando se declara el tamaño de una matriz en (9) significa que la matriz tiene 10 elementos. Lo mismo va para las matrices multi-dimensionales.

```
Dim myList As New List(Of Integer)
For i As Integer = 1 To 10
  myList.Add(10 * i)
  Print("Element(" & i - 1 & ") = " & myList(i - 1))
Next
```

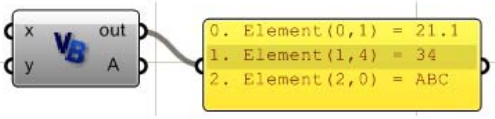


Puede utilizar la lista anidada o ArrayList para declarar matrices multi-dimensionales dinámicas de mismo tipo u otro mixto. Observe el siguiente ejemplo:

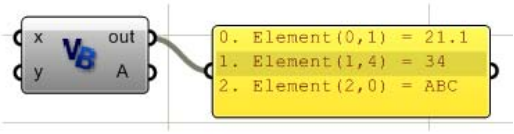
```
Dim myList As New ArrayList()
Dim myRow1() As Double = {20.1,21.1,22.1,23.1}
Dim myRow2() As Integer = {30,31,32,33,34}
Dim myRow3() As String = {"ABC", "DEF"}
```



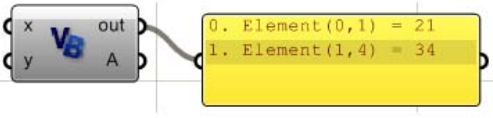

```
Dim myList As New ArrayList()
Dim myRow1 As New List( Of Double )
MyRow1.Add(20.1): MyRow1.Add(21.1): MyRow1.Add(22.1): MyRow1.Add(23.1)
Dim myRow2 As New List( Of Integer )
MyRow2.Add(30): MyRow2.Add(31): MyRow2.Add(32): MyRow2.Add(33): MyRow2.Add(34)
Dim myRow3 As New List( Of String )
MyRow3.Add("ABC"): MyRow3.Add("DEF")
```




```
Dim myList As New List(Of Object)
Dim myRow1 As New List( Of Double )
MyRow1.Add(20.1): MyRow1.Add(21.1): MyRow1.Add(22.1): MyRow1.Add(23.1)
Dim myRow2 As New List( Of Integer )
MyRow2.Add(30): MyRow2.Add(31): MyRow2.Add(32): MyRow2.Add(33): MyRow2.Add(34)
Dim myRow3 As New List( Of String )
MyRow3.Add("ABC"): MyRow3.Add("DEF")
```




```
Dim myList As New List(Of List(Of Integer))
Dim myRow1 As New List( Of Integer )
MyRow1.Add(20): MyRow1.Add(21): MyRow1.Add(22): MyRow1.Add(23)
Dim myRow2 As New List( Of Integer )
MyRow2.Add(30): MyRow2.Add(31): MyRow2.Add(32): MyRow2.Add(33): MyRow2.Add(34)
```



Add Lists and Print:

```
myList.Add(myRow1)
myList.Add(myRow2)
myList.Add(myRow3)

Print("Element(0,1) = " & myList(0)(1))
Print("Element(1,4) = " & myList(1)(4))
Print("Element(2,0) = " & myList(2)(0))
```

14.5 Operadores

Hay muchos operadores integrados en VB.NET. Ellos operan en uno o más niveles. Aquí hay una tabla de los operadores más comunes para una referencia rápida:

Tipo	Operador	Descripción
Operadores Aritmeticos	**^**	Eleva un número a la potencia de otro número.
	*****	Multiplica dos números.
	/	Divide dos números y devuelve un resultado de punto flotante.
	****	Divide dos números y devuelve un resultado entero
	Mod	Divide dos números y devuelve sólo el resto.
	+	Suma dos números y devuelve el valor positivo de una expresión numérica.
	-	Devuelve la diferencia entre dos expresiones numéricas o el valor negativo de una expresión numérica.
Operadores de asignación	**=**	Asigna un valor a una variable.
	^=	Aumenta el valor de una variable a la potencia de una expresión y asigna el resultado de nuevo a la variable.
	***=**	Multiplica el valor de una variable por el valor de una expresión y asigna el resultado a la variable.
	/=	Divide el valor de una variable por el valor de una expresión y asigna el resultado de punto flotante a la variable.
	\=	Divide el valor de una variable por el valor de una expresión y asigna el resultado entero a la variable.
	+=	Agrega el valor de una expresión numérica al valor de una variable numérica y asigna el resultado a la variable. También puede ser utilizado para concatenar una expresión de cadena a una variable de cadena y asignar el resultado a la variable.
	-=	Resta el valor de una expresión desde el valor de una variable y le asigna el resultado a la variable.
	&=	Concatena una expresión de cadena a una variable de cadena o a las propiedades y asigna el resultado a la variable o propiedad.
Operadores de comparación	**<**	Menor a
	<=	Menor o igual
	>	Mayor a
	>=	Mayor o igual
	=	Igual
	<>	No igual
Operadores de concatenación	**&**	Genera una concatenación de cadenas de dos expresiones.
	+	Concatena dos expresiones de cadena.
Operadores logicos	**And**	Realiza una conjunción lógica de dos expresiones booleanas
	Not	Realiza la negación lógica en una expresión booleana
	Or	Realiza una disyunción lógica de dos expresiones booleanas
	Xor	Realiza una exclusión lógica de dos expresiones Boolean

14.6 Sentencias condicionales

Usted puede pensar en sentencias condicionales como bloques de código con puertas que se abren sólo cuando se cumple la condición requerida. La sentencia condicional que se utiliza mayormente es la condición "If" con el siguiente formato **“IF<condition> Then <code> End IF”.**

```
'Single line if statement doesn't need End If: condition=(x<y), code=(x=x+y)
If x < y Then x = x + y

'Multiple line needs End If to close the block of code
If x < y Then
   x = x + y
End If
```

También es posible utilizar un **"Else If ... Then"** y **"Else"** para elegir el bloque de código alternativo n a ejecutar. Por ejemplo:

```
If x < y Then
   x = x + y        'execute this line then go to the step after "End If"
Else If x > y Then
   x = x – y        'execute this line then go to the step after "End If"
Else
   x = 2*x          'execute this line then go to the step after "End If"
End If
```

También está la sentencia "Select Case". Se utiliza para ejecutar diferentes bloques de código basados en el valor de una expresión ("index" en el ejemplo). Por ejemplo:

```
Select Case index
   Case 0          'If index=0 the execute next line, otherwise go directly to the next case
      x = x * x
   Case 1
      x = x ^ 2
   Case 2
      x = x ^ (0.5)
End Select
```

14.7 Loops

Los loops permiten repetir la ejecución del bloque de código dentro del cuerpo del loop una y otra vez, siempre y cuando se cumpla la condición de loop. Existen diferentes tipos de loops. Vamos a explicar dos de ellos que son los más comúnmente utilizados.

“For … Next” Loop

Esta es la forma más común de crear un loop. La estructura del loop se ve así:

```
For < index = start_value> To <end_value> [Step <step_value>]

   'For loop body starts here
    [ statements/code to be executed inside the loop]

  [ Exit For ]   'Optional: to exit the loop at any point

    [ other statements]
```

```
    [ Continue For ]     'optional: to skip executing the remaining of the loop statements.

    [ other statements]

    'For loop body ends here (just before the "Next")
    'Next means: go back to the start of the for loop to check if index has passed end_value
    'If index passed end_value, then exit loop and execute statements following "Next"
    'Otherwise, increment the index by "step_value"
Next
[ statements following the For Loop ]
```

El siguiente ejemplo utiliza un loop para iterar a través de una serie de funciones:

```
'Array of places
Dim places_list As New List( of String )
places_list.Add( "Paris" )
places_list.Add( "NY" )
places_list.Add( "Beijing" )

'Loop index
Dim i As Integer
Dim place As String
Dim count As Integer = places_list.Count()

'Loop starting from 0 to count -1 (count = 3, but last index of the places_list = 2)
For i=0 To count-1
   place = places_list(i)
   Print( place )
Next
```

Si utiliza el loop a través de los objetos en una matriz, también puede utilizar el loop "For ...Next" para iterar a través de los elementos de la matriz sin necesidad de utilizar un índice. El ejemplo de arriba puede ser re-escrito de la siguiente forma:

```
Dim places_list As New List( of String )
places_list.Add( "Paris" )
places_list.Add( "NY" )
places_list.Add( "Beijing" )

For Each place As String In places_list
   Print( place )
Next
```

"While ... End While" Loop

Esto también es un loop muy utilizado. La estructura del loop es así:

```
While < some condition is True >

    'While loop body starts here
    [ statements to be executed inside the loop]

   [ Exit While ]        'Optional to exit the loop

    [ other statements]

    [ Continue While ]  'optional to skip executing the remaining of the loop statements.
```

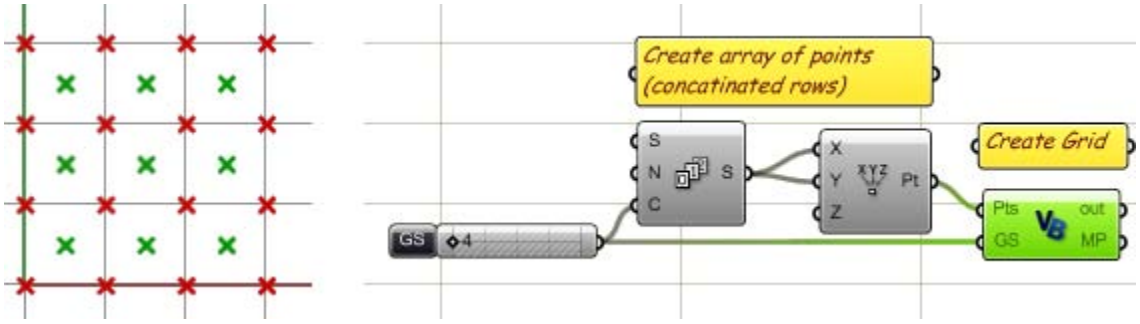
```
    [ other statements]

    'While loop body ends here
    'Go back to the start of the loop to check if the condition is still true, then execute the body
    'If condition not true, then exit loop and execute statements following "End While"

End While
[ statements following the While Loop ]
```

Este es el ejemplo de las funciones anteriores re-escrita usando el loop "while":

```
Dim places_list As New List( of String )
places_list.Add( "Paris" )
places_list.Add( "NY" )
places_list.Add( "Beijing" )

Dim i As Integer
Dim place As String
Dim count As Integer = places_list.Count()

i = 0
While i < count            '(i<count) evaluates to true or false
   place = places_list(i)
   Print( place )
   i = i + 1
End While
```

14.8 Loops Anidados

Los loops anidados son loops que incluye otro loop en su cuerpo. Por ejemplo, si tenemos una rejilla de puntos, luego con el fin de llegar a cada punto de la lista, necesitamos utilizar un loop anidado. El siguiente ejemplo muestra cómo convertir una matriz unidimensional de puntos en una grilla de 2 dimensiones. Luego de esto procesaremos la grilla para encontrar los puntos medios de las células.



El script tiene dos partes:

- Primero convertiremos una matriz unidimensional en una de 2 dimensiones que llamaremos "Grid".
- Segundo procesamos la grilla para encontrar los puntos medios de las células.

En ambas partes hemos utilizado loops anidados. Aquí está la definición de secuencia de comandos en Grasshopper:

```
Sub RunScript(ByVal Pts As List(Of On3dPoint), ByVal GS As Integer)

    'Create a grid of points
    Dim Grid As New ArrayList()

    Dim i As Integer
    Dim j As Integer

    'Nested loop to covert 1D array to 2D grid
L1  For i = 0 To Pts.Count() - 1 Step GS
      'Declare a row of points
      Dim Row As New List( Of On3dPoint )
L2    For j = i To i + GS - 1
        'Get a reference od the point
        Dim pt As On3dPoint
        pt = Pts(j)

        'Add point to the row
        Row.Add(pt)
      Next
      'Add row to the grid
      Grid.Add(Row)
    Next

    'Process the grid to find mid points of cells
    Dim mid_points As New List( Of On3dPoint )
L1  For i = 1 To Grid.Count() - 1
      'Get first and second rows
      Dim Row0 As List( Of On3dPoint )
      Row0 = Grid(i - 1)
      Dim Row1 As List( Of On3dPoint )
      Row1 = Grid(i)

L2    For j = 1 To Row0.Count() - 1
        Dim mid_pt As New On3dPoint
        mid_pt = (Row0(j-1) + Row0(j) + Row1(j-1) + Row1(j)) / 4
        mid_points.Add(mid_pt)
      Next
    Next

    'Assign mid point to output
    MP = mid_points
End Sub
```

14.9 Subs y Funciones

RunScript es la función principal que todos los componentes script usan. Esto es lo que usted ve cuando abre un componente de script por defecto en Grasshopper:

```
Sub RunScript(ByVal x As Object, ByVal y As Object)
        '"<your code…>
End Sub
```

Sub… End Sub: Son Palabras clave que cierran los bloques de código.
"**RunScript**": es el nombre del "Sub".
"**(…)**": Paréntesis después del nombre "Sub" encierran los parámetros de entrada
"**ByVal x As Object,…**": Son los llamados parámetros de entrada.

Cada parámetro de entrada necesita definir los siguiente:

- **ByRef** o **ByVal**: Especifica si el parámetro es pasado por valor o por referencia.
- **Name**: nombre del parámetro.
- El tipo de parámetro es definido luego de la palabra clave **"As"**.

Tenga en cuenta que los parámetros de entrada en "RunScript sub" de Grasshopper son todos pasados por valor (palabra clave **ByVal**). Eso significa que son copias de la entrada original y cualquier modificación en estos parámetros dentro del script no afectará a la entrada original. Sin embargo, si se define subs/funciones adicionales dentro de su componente script, puede pasar parámetros por referencia (**ByRef**). Pasar parámetros por referencia significa que cualquier modificación de estos parámetros dentro de la función va a cambiar el valor original que fue pasado cuando la función exista.

Usted tal vez pueda incluir todo el código dentro del cuerpo de RunScript, sin embargo, también puede definir subs/funciones externas si es necesario. Pero ¿por qué utilizar las funciones externas?

- Para simplificar el código de la función principal.
- Para hacer el código mas legible.
- Para aislar y reutilizar ciertas funcionalidades comunes.
- Para definir funciones especializadas, como la recursividad.

¿Cuál es la diferencia entre **Sub** y **Función** de todos modos? Se define un Sub si no necesita un valor de retorno. Las funciones en cambio, le permiten devolver un resultado. Es, básicamente, asignar un valor al nombre de aquella función. Por ejemplo:

```
Function AddFunction( ByVal x As Double, ByVal y As Double )
        AddFunction = x + y
End Function
```

Esto significa, que no necesita tener una función para obtener un valor. Subs puede hacerlo a través de los parámetros de entrada que se pasan "ByRef". En el siguiente, "**rc**" se utiliza para devolver resultados:

```
Sub AddSub( ByVal x As Double, ByVal y As Double, ByRef rc As Double )
        rc = x + y
End Sub
```

He aquí cómo la función de llamada se ve usando "Function" y "Sub":

```
Dim x As Double = 5.34
Dim y As Double = 3.20
Dim rc As Double = 0.0
'Can use either of the following to get result
rc = AddFunction( x, y )    'Assign function result to "rc"
AddSub( x, y, rc )          'rc is passed by refernce and will have addition result
```

En la sección de loops anidados, ilustramos un ejemplo que ha creado una grilla desde una lista de puntos y luego, calculamos los puntos medios. Cada una de estas dos

funciones es suficiente distinta para separar en una subestación externa y probablemente reutilizarla en el código futuro. Aquí hay una re-escritura del ejemplo de la red utilizando las funciones externas.

```
  Sub RunScript(ByVal Pts As List(Of On3dPoint), ByVal GS As Integer)

    'Create a grid of points
    Dim Grid As New ArrayList()

    'Call grid function
1   Call CreateGrid(Pts, Grid, GS)

    'Call mid points function
    Dim mid_points As New List( Of On3dPoint )
2   Call FindMidPoints(Grid, mid_points)

    'Assign mid point to output
    MP = mid_points
  End Sub

#Region "Additional methods and Type declarations"

  'Function to convert 1d array to 2d array
1 Sub CreateGrid( ByVal Pts As List(...

  'Function to find grid mid points
2 Sub FindMidPoints(ByVal Grid As ArrayList, mid_points As List(...

#End Region
End Class
```

Así es cómo cada uno de los dos Subs se ve una vez ampliado:

```
#Region "Additional methods and Type declarations"

  'Function to convert 1d array to 2d array
  Sub CreateGrid( ByVal Pts As List(Of On3dPoint), ByRef Grid As ArrayList, ByVal GS As Integer )

    Dim i As Integer
    Dim j As Integer

    For i = 0 To Pts.Count() - 1 Step GS
      'Declare a row of points
      Dim Row As New List( Of On3dPoint )
      For j = i To i + GS - 1
        'Get a reference od the point
        Dim pt As On3dPoint
        pt = Pts(j)

        'Add point to the row
        Row.Add(pt)
      Next
      'Add row to the grid
      Grid.Add(Row)
    Next

  End Sub
```

```
'Function to find grid mid points
Sub FindMidPoints(ByVal Grid As ArrayList, mid_points As List(Of On3dPoint ))

  Dim i As Integer
  Dim j As Integer

  For i = 1 To Grid.Count() - 1
    'Get first and second rows
    Dim Row0 As List( Of On3dPoint )
    Row0 = Grid(i - 1)
    Dim Row1 As List( Of On3dPoint )
    Row1 = Grid(i)

    For j = 1 To Row0.Count() - 1
      Dim mid_pt As New On3dPoint
      mid_pt = (Row0(j - 1) + Row0(j) + Row1(j - 1) + Row1(j)) / 4
      mid_points.Add(mid_pt)
    Next
  Next

End Sub
#End Region
```

14.10 Recursividad

Las funciones recursivas son un tipo especial de funciones que se llaman a sí mismas hasta que alguna condición de parada se cumpla. La recursividad es comúnmente utilizada para la búsqueda de datos, la subdivisión y los sistemas generativos. Vamos a discutir un ejemplo que muestra cómo funciona la recursividad. Para obtener más ejemplos de recursividad, vea la wiki de Grasshopper y la página de las Galerías.

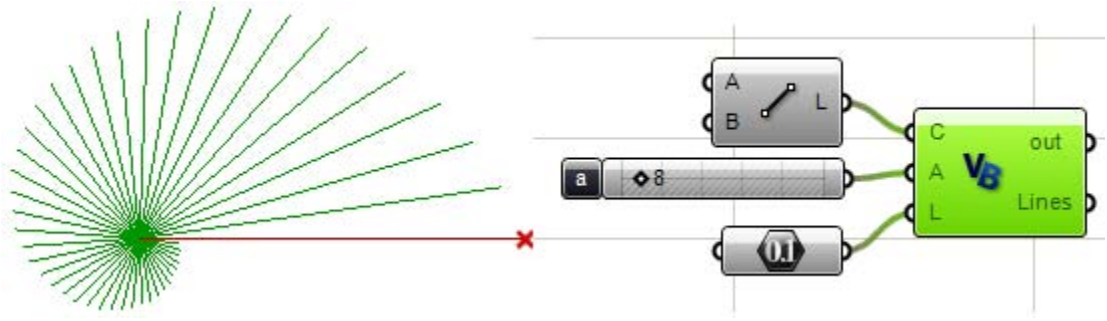
El siguiente ejemplo toma la menor parte de una línea de entrada y la rota en un ángulo determinado. Esto lo sigue haciendo hasta que la longitud de línea es menor que una longitud mínima determinada.

Parametros de entrada:

- Linea inicial (C).
- Anglulo en radianes (A). El deslizador muestra los angulos en grados, pero los convierte en radianes
- Longitud minima (L) – condición de parada.

Salida:

- Serie de lineas.



Vamos a resolver el mismo ejemplo de forma iterativa, así como recursiva para poder comparlos.

Solución recursiva. Tenga en cuenta que dentro de la sub "DivideAndRotate" hay:

- Una condicion de parada para salir del “sub”.
- Una llamada para la misma funcion (la funcion recusiva se llama a si misma).
- “AllLines” (serie de lineas) es pasada como referencia para seguir añadiendo nuevas lineas a la lista.

```
Sub RunScript(ByVal C As OnLine, ByVal A As Double, ByVal L As Double)

  'Declare all lines
  Dim AllLines As New List( Of OnLine )

  'Call recursive function
  Call DivideAndRotate(Line, AllLines, A, L)

  'Assign return value
  Lines = AllLines
End Sub

#Region "Additional methods and Type declarations"

  Sub DivideAndRotate(ByVal Line As OnLine,
                      ByRef AllLines As List(Of OnLine),
                      ByVal angle As Double,
                      ByVal MinLength As Double)

    'Check the stopping condition
    If Line.Length() < MinLength Then Exit Sub

    'Take a portion of the line
    Dim new_line As New OnLine(Line)
    Dim end_pt As New On3dPoint
    end_pt = new_line.PointAt(0.95)

    new_line.To = end_pt

    'Rotate
    new_line.Rotate(angle, OnUtil.On_zaxis, Line.from)

    AllLines.Add(new_line)

    'Call self
    Call DivideAndRotate(new_line, AllLines, angle, MinLength)

  End Sub

#End Region
```

Esta es la misma funcionalidad con la solución iterativa mediante un loop "while":

```
Sub RunScript(ByVal C As OnLine, ByVal A As Double, ByVal L As Double)

  'Declare all lines
  Dim AllLines As New List( Of OnLine )

  'Find current length
  Dim current_L As Double = C.Length()

  Dim new_line As OnLine
  new_line = C

  'Loop until length is less than min length
  While current_L > L
    'Generate the new line
    new_line = DivideAndRotate(new_line, A)

    'Add to list
    AllLines.Add(new_line)

    'Stopping condition
    current_L = new_line.Length()
  End While

  'Assign return value
  Lines = AllLines
End Sub

#Region "Additional methods and Type declarations"

Function DivideAndRotate(ByVal L As OnLine, ByVal A As Double) As OnLine

  'Take a portion of the line
  Dim new_line As New OnLine(L)
  Dim end_pt As New On3dPoint
  end_pt = new_line.PointAt(0.95)

  new_line.To = end_pt

  'Rotate
  new_line.Rotate(A, OnUtil.On_zaxis, L.from)

  'Function return
  DivideAndRotate = new_line

End Function

#End Region
```

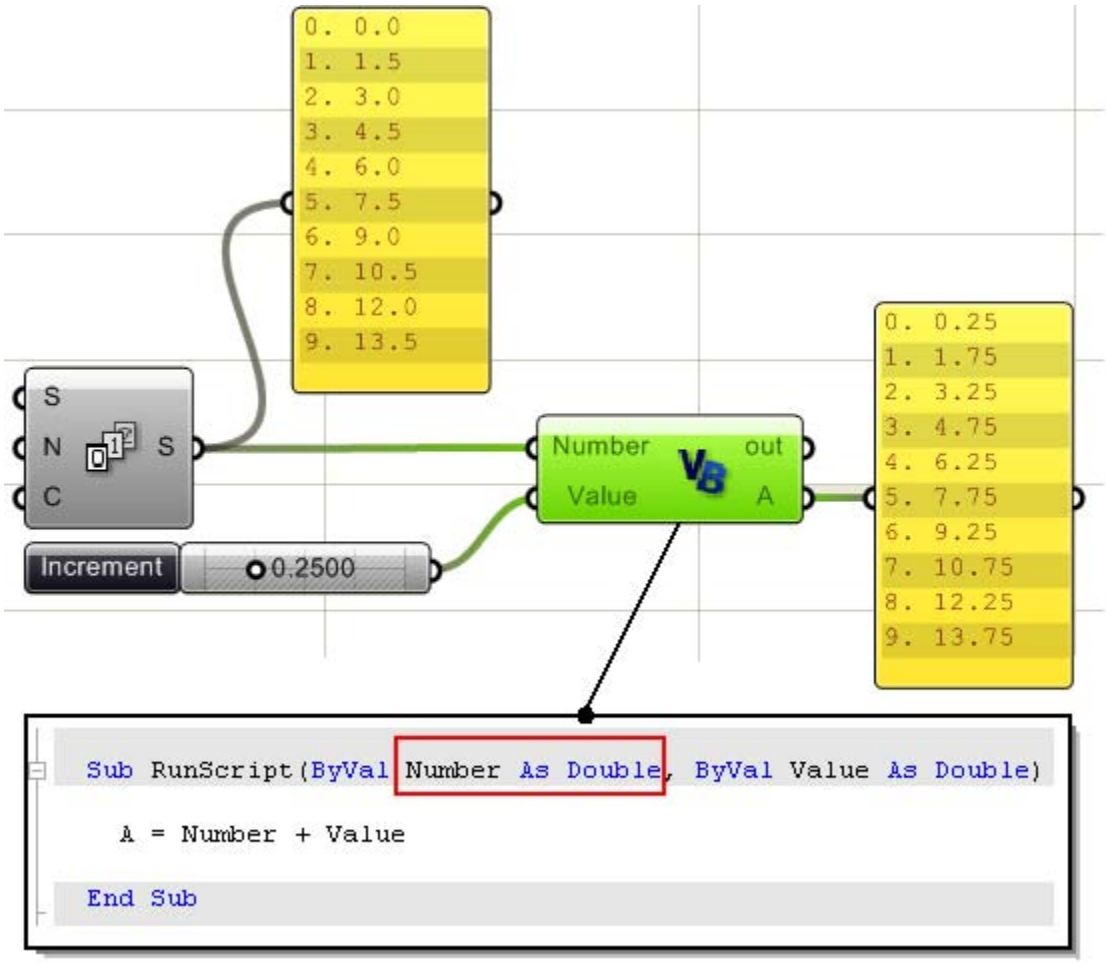
14.11 Procesando listas en Grasshopper

El componente script de Grasshopper puede procesar la lista de entrada de dos formas:

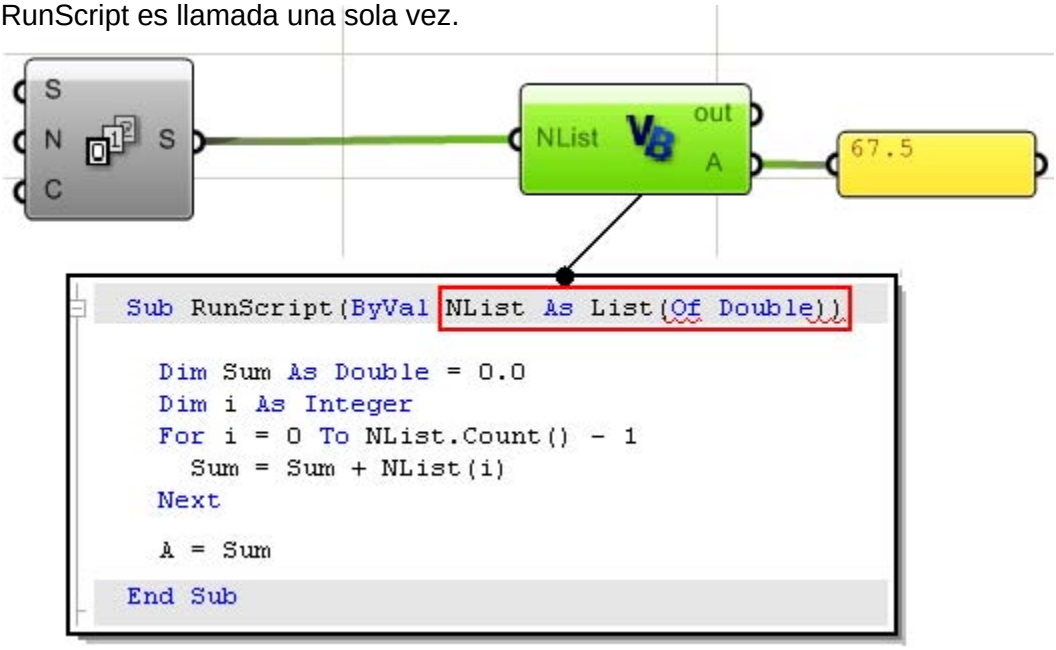
1. Proceso de un valor de entrada a la vez (el componente es llamado el número de veces igual la número de valores en la matriz de entrada).
2. Proceso de todos los valores de entrada entre sí (el componente es llamado una sola vez).

Si usted necesita procesar cada elemento en una lista independientemente del resto de los elementos, entonces es más fácil usando el primer método. Por ejemplo, si usted tiene una lista de números a los que desea incrementar cada uno de ellos por "10", entonces usted puede utilizar el primer método. Pero si usted necesita una función de suma para añadir todos los elementos, entonces usted tendrá que utilizar el segundo método y pasar toda la lista como una entrada.

El siguiente ejemplo muestra cómo procesar una lista de datos mediante el primer método de un valor de entrada a la vez. En este caso, la función RunScript es llamada 10 veces (una por cada número). Lo esencial a destacar es que el parámetro de entrada "Número es pasado como un "Doble" en contraposición a la "Lista (de doble)", como veremos en el siguiente ejemplo.



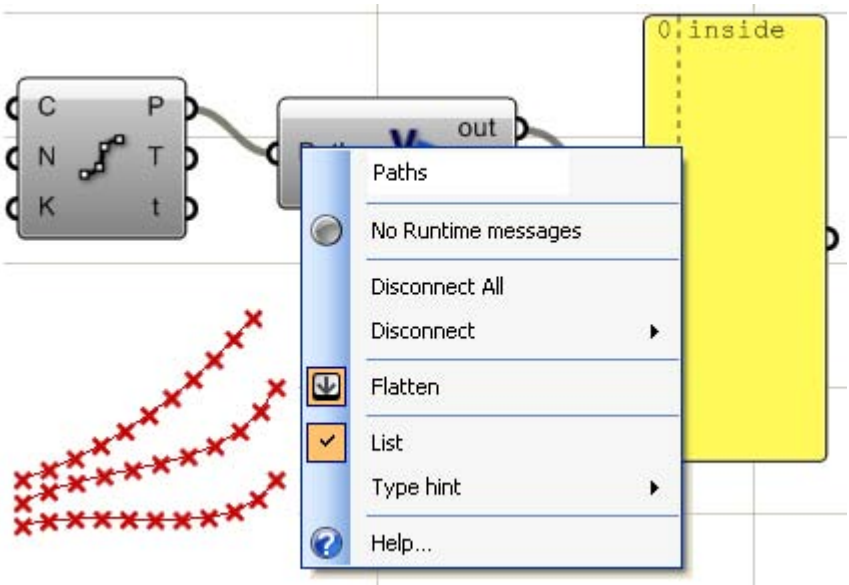
En el siguiente, ingresamos la lista de números. Usted puede hacer esto, mediante un clic derecho sobre el parámetro de entrada y de pulsando "Lista". La función RunScript es llamada una sola vez.



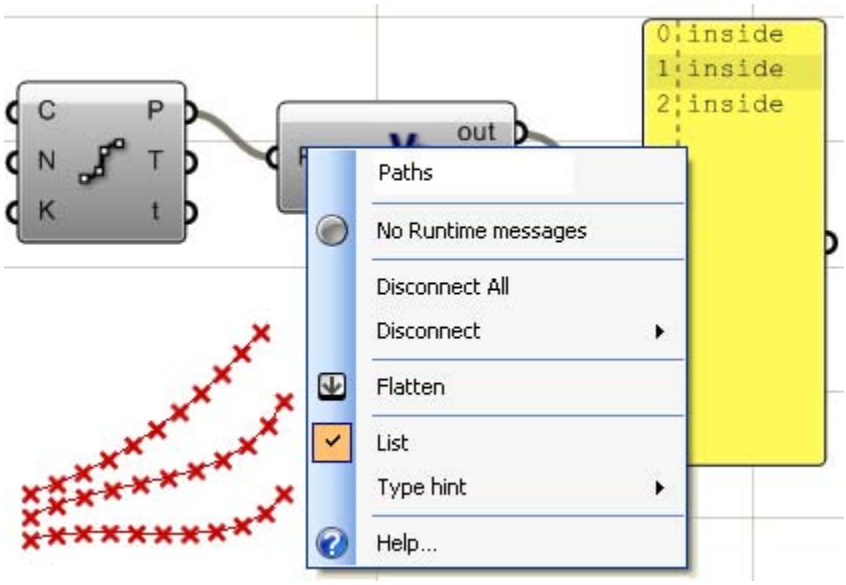
14.12 Procesando árboles en Grasshopper

Los árboles (o datos multi-dimensionales) se pueden procesar de un elemento a la vez o todos a la vez. Por ejemplo, si dividimos en tres curvas en diez segmentos cada una, entonces tenemos una estructura de tres rutas, cada una con once puntos. Si usamos esto como una entrada entonces obtenemos lo siguiente:

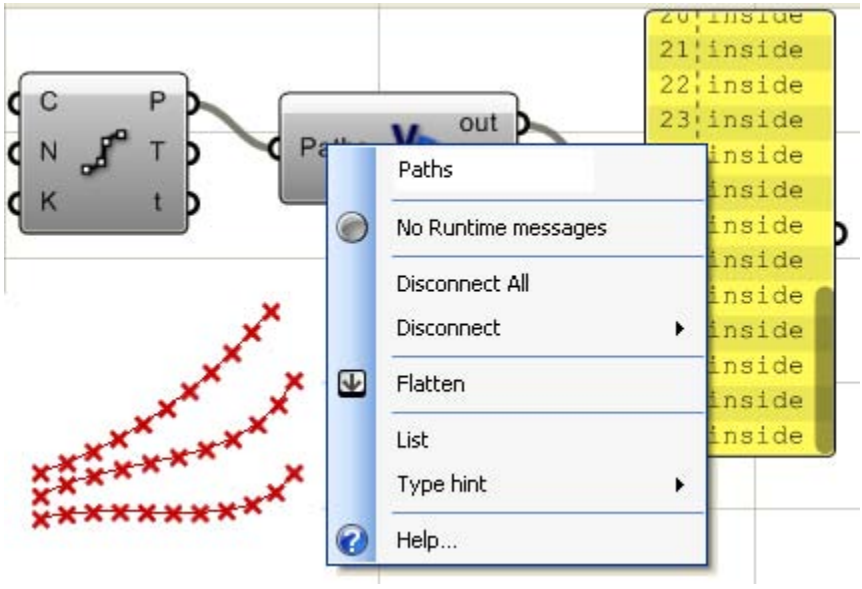
A: Si ambos “Flatten” y “List” estan activos, entonces el componente es llamado una sola vez y la “flat list” de todos los puntos son pasados:



B: Si "List" está seleccionado, entonces el componente es llamado tres veces en cada tiempo, pasando a una lista de una curva con puntos de división:



C: Cuando no esta seleccionado “List”, entonces la función se llama una vez para cada punto de división (33 veces en este caso). Lo mismo ocurre si sólo "Flatten" está marcado:



Este es el código en el componente de VB. Básicamente, sólo se imprima la palabra "dentro" para indicar que se llamó el componente:

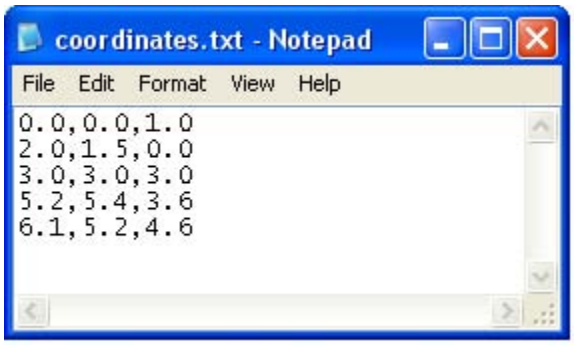
```
Sub RunScript(ByVal Paths As Object)

  Print("inside")

End Sub
```

14.13 Archivo I/O

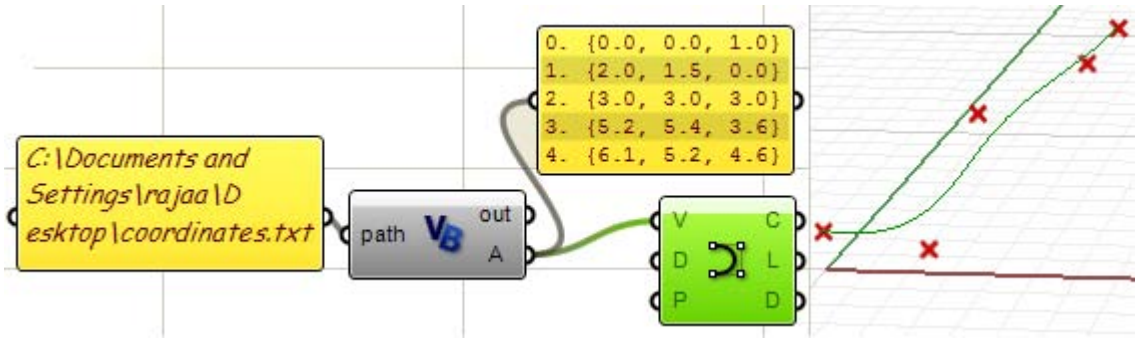
Hay muchas maneras de leer y escribir archivos en VB.NET y muchos tutoriales y documentos están disponibles en Internet. En general, la lectura de un archivo consiste en lo siguiente:

- Abra el archivo. Generalmente se necesita de una ruta de accseo para hacerlo.
- Leer la cadena (cadena entera o línea por línea).
- Tokenize la cadena usando algunos caracteres delimitadores.
- Cast de cada token (el doble en este caso).
- Resultado de la funcion.

Vamos a ilustrar un ejemplo sencillo que analiza los puntos de un archivo de texto. Utilizando el siguiente formato de archivo de texto, vamos a leer cada línea como un solo punto y utilizar el primer valor como coordenada “x”, el segundo como “y” y el tercero como “z” del punto. Podremos entonces usar estos puntos como puntos de control de una curva.



El componente de VB acepta como entrada una cadena que es la ruta del archivo a leer y de salida “On3dPoints”.



Aquí está el código dentro del componente de secuencia de comandos.

```
Sub RunScript(ByVal path As String)

  'Check if file exists
  If (Not IO.File.Exists(path)) Then
    Print("Exit without reading")
    Return
  End If

  'Read the file
  Dim lines As String() = IO.File.ReadAllLines(path)

  'Check that file is not empty
  If (lines Is Nothing) Then
    Print("File has no content")
    Return
  End If

  'Declare list of points
  Dim pts As New List(Of On3dPoint)

  'Loop through lines
  For Each line As String In lines
    'Tokenize line into array of strings separated by ","
    Dim parts As String() = line.Split(",".ToCharArray())

    'Make sure that each line has exactly 3 values
    If UBound(parts) <> 2 Then Continue For

    'Convert each coordinate from string to double
    Dim x As Double = Convert.ToDouble(parts(0))
    Dim y As Double = Convert.ToDouble(parts(1))
    Dim z As Double = Convert.ToDouble(parts(2))

    pts.Add(New On3dPoint(x, y, z))
  Next

  A = pts

End Sub
```

15 *Rhino .NET SDK*

15.1 Descripción general

Rhino. NET SDK proporciona acceso a la geometría y funciones de utilidad OpenNURBS Hay un archivo de ayuda que viene cuando se descarga. NET SDK. Es un recurso de gran ayuda. Aquí es donde usted puede conseguirlo:
http://en.wiki.mcneel.com/default.aspx/McNeel/Rhino4DotNetPlugIns.html

En esta sección, nos centraremos en la parte de la relacion del SDK con las clases de geometría Rhino y funciones de utilidad. Le mostraremos ejemplos acerca de cómo crear y manipular la geometría utilizando el componente de escritura de VB en Grasshopper .

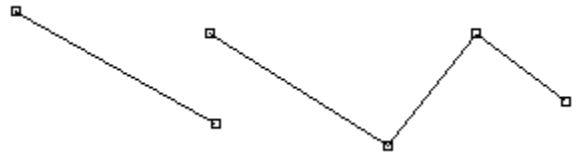
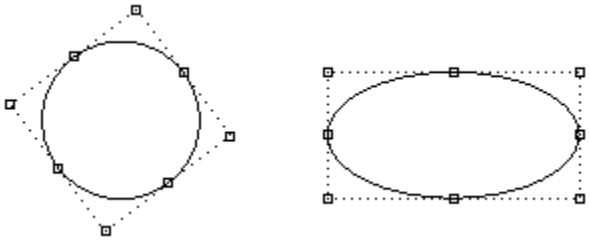
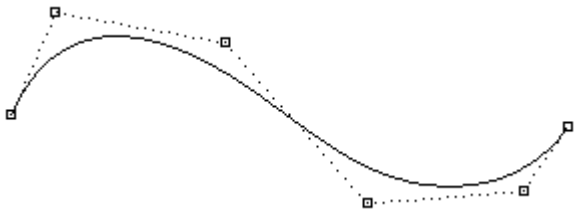
15.2 Entendimiendo las NURBS

Rhino es un modelador NURBS que define las curvas y superficies de geometría utilizando curvas de base racional no-uniforme “Non-uniform Rational Base Spline” (o NURBS, para abreviar). NURBS es una representación matemática precisa de las curvas y superficies que es muy intuitivo para editar.

Hay muchos libros y referencias para aquellos interesados en una lectura en profundidad sobre NURBS (http://en.wikipedia.org/wiki/NURBS). Una comprensión básica de NURBS es necesaria para ayudarle a utilizar las clases de SDK y las funciones más eficazmente.

Hay cuatro cosas que definen una curva nurbs. Los grados, puntos de control, nodos y las normas de evaluación:

Grados
Es un número entero positivo que es generalmente igual a 1,2,3 o 5. Los ejemplos siguientes son algunas curvas y sus respectivos grados:

	**Lineas** y **polylineas** son curvas nurbs de grado 1. Orden = 2 (orden = grado + 1)
	**Circulos** y **elipses** son ejemplos de curvas nurbs de grado 2. Orden = 3.
	Curvas de forma libre son usualmente repre sentadas como curvas nurbs de grado 3 Orden = 4

Puntos de control

Los puntos de control de una curva NURBS es una lista de al menos (grado+1) puntos. La manera más común de cambiar la forma de una curva nurbs es a través de la manipulación de sus puntos de control. Los puntos de control tienen asociados un número llamado “weight”. Salvo algunas excepciones, los “weights” corresponden a números positivos. Cuando los puntos de control de una curva tienen el mismo “weight” (usualmente 1), la curva es llamada no-racional. En adelante veremos cómo cambiar interactivamente los weights” de los puntos de control en Grasshopper.

Nudos o vector de nudo

Cada curva NURBS tiene una lista de números asociados a ella que se llama “vector de nudo”. Los nudos son un poco más difíciles de entender y configurar, pero por suerte hay funciones SDK que hacen el trabajo por usted. Sin embargo, hay un par de cosas que serán útiles para aprender sobre el vector de nudos:

Multiplicidad de nudo

El número de veces que un valor de nudo es duplicado se denomina multiplicidad del nudo. Cualquier valor de nudo no puede ser duplicado más veces que el grado de curva. Aquí hay algunas cosas que sería bueno saber acerca de los nudos.

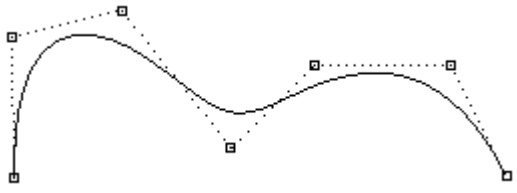
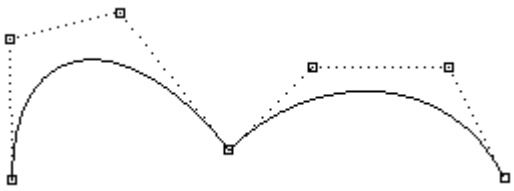
Nudo de multiplicidad completa es un nudo duplicado el número de veces igual al grado de la curva. Las curvas sujetas al suelo tienen nudos con multiplicidad completa en los dos extremos de la curva y es por eso que los puntos de control final coinciden con puntos finales de la curva. Si hubiera un nudo de multiplicidad completa en el medio del vector de nudo, entonces la curva pasará por el punto de control y habría un retorcimiento.

Nudo simple: es un nudo con un valor que aparece sólo una vez.

Un vector nudo uniforme cumple 2 condiciones:

1. Número de nudos = número de puntos de control + grado - 1.
2. Los nudos comienzan con un nudo de multiplicidad completa, es seguido por dos nudos simples, termina con un nudo de multiplicidad completa, y los valores van en aumento y equidistantes. Esto es típico de las curvas sujetas al suelo. Las curvas periódicas funcionan de manera diferente, como veremos más adelante.

Éstas son dos curvas con puntos de control idénticos pero de vectores de nudos diferentes:

	Grado = 3 Número de puntos de control = 7 vector nudo = (0,0,0,1,2,3,5,5,5)
	Grado = 3 Número de puntos de control = 7 vector nudo = (0,0,0,1,1,1,4,4,4) **Nota:***Multiplicidad nudo completo en el medio crea un pliegue y la curva se ve obligada a pasar por el punto de control asociado.*

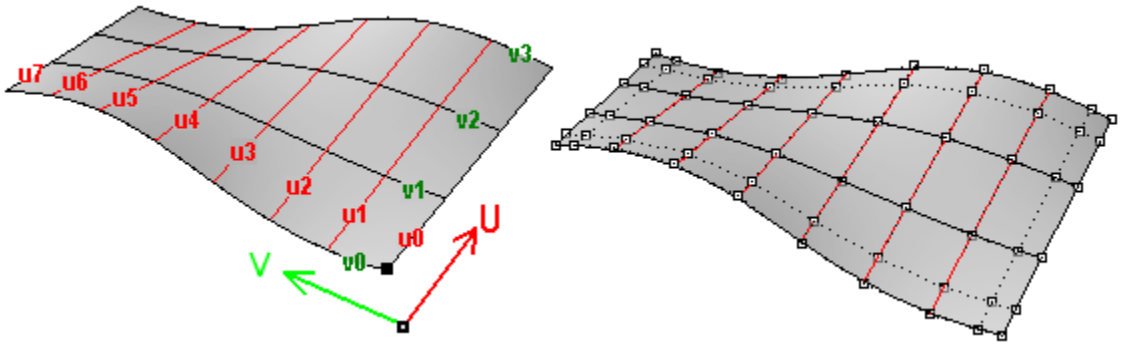
Regla de evaluación

La regla de evaluación utiliza una fórmula matemática que toma un número y asigna un punto. La fórmula consiste en el grado, los puntos de control, y los nudos.

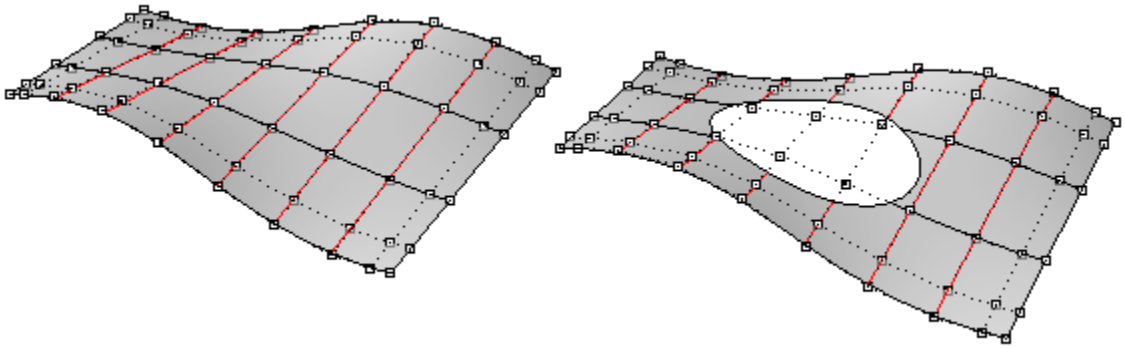
Usando esta fórmula, hay funciones SDK que toman un parámetro de la curva, y producen el punto correspondiente en la curva. Un parámetro es un número que se encuentra dentro del dominio de la curva. Los dominios son por lo general en aumento y se componen de dos números:
parámetro de dominio mínimo (m_t (0)) que por lo general es el inicio de la curva y máximo (m_t (1)) al final de la curva.

Surperficies NURBS

Usted puede pensar en superficies NURBS como una cuadrícula de curvas NURBS que van en dos direcciones. La forma de una superficie NURBS se define mediante un número de puntos de control y el grado que la superficie posee en cada una de las dos direcciones (direcciones **u** y **v**). Consulte el *Glosario de Ayuda de Rhino* para más detalles.



Las superficies NURBS se pueden recortar o des-recortar. Piense en las superficies recortadas como el uso de una superficie NURBS subyacentes y con curvas cerradas para cortar una forma específica de dicha superficie. Cada superficie tiene una curva cerrada que define el borde externo (*loop externo*) y posee muchas curvas interiores no-intersectadas para definir vacios (*loops interiores*). Una superficie con loop externo que es la misma que la de su superficie NURBS subyacente y que no tiene agujeros es lo que llamamos una superficie sin recortar.

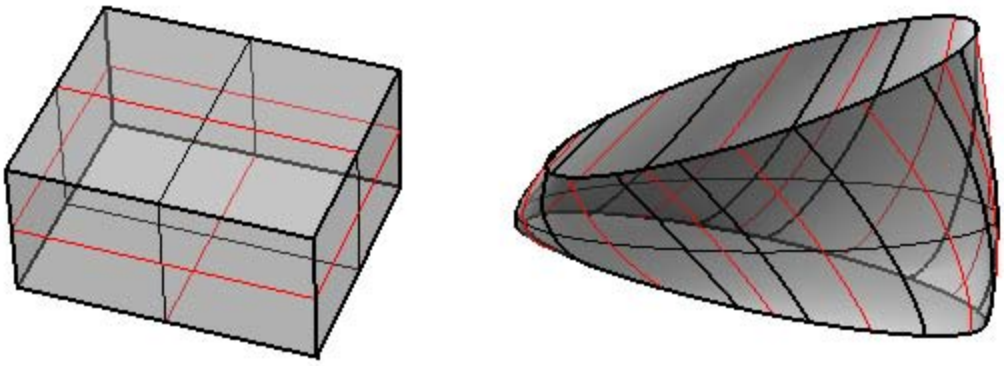


La superficie de la izquierda está sin recortar. La de la derecha es la misma superficie con un agujero elípitco recortado. Tenga en cuenta que la estructura NURBS de la superficie no cambia cuando es recortada.

Polisuperficies

Una polisuperficie consta de más de una superficie (generalmente recortadas) unidas entre ellas. Cada una de las superficies tiene su propia parametrización y sus direcciones UV no tienen que coincidir. Las polisuperficies y las superficies recortadas se representan usando lo que se llama la representación de contorno (BREP para abreviar). Básicamente describe superficies, bordes y vértices de la geometría con datos recortados y relaciones entre las diferentes partes. Por ejemplo, se describe cada cara, alrededor de sus bordes y recortes, dirección normal respecto a la superficie, la relación con las caras vecinas y así sucesivamente. Vamos a describir las variables de BReps y cómo están conectadas en algún detalle más adelante.

OnBrep es, probablemente, la estructura de datos más complejos en OpenNURBS y tal vez no sea fácil de digerir, pero por suerte hay un montón de herramientas y funciones globales que vienen con el SDK de Rhino para ayudar a crear y manipular BReps.



15.3 Jerarquía de objetos en OpenNURBS

La ayuda del archivo SDK muestra todas las clases de jerarquía. Aquí hay un subconjunto diseccionado de las clases en relación con la creación y la manipulación de la geometría que es probable que usted utilice cuando escriba los scripts. Tengo que advertir que, esta lista es muy incompleta. Por favor, consulte el archivo de ayuda para más detalles.

OnObject (*todas las clases de Rhino derivan de OnObject*)

- **OnGeometry** (*la clase se deriva de o hereda de OnObject*
  - OnPoint
    - OnBrepVertex
    - OnAnnotationTxtDot
  - OnPointGrid
  - OnPointCloud
  - OnCurve (*abstaract class)*
    - OnLineCurve

            - OnPolylineCurve
            - OnArcCurve
            - OnNurbsCurve
            - OnCurveOnSurface
            - OnCurveProxy
                - OnBrepTrim
                - OnBrepEdge
        - OnSurface (*clase resumida*)
            - OnPlaneSurface
            - OnRevSurface
            - OnSumSurface
            - OnNurbsSurface
            - OnProxySurface
                - OnBrepFace
                - OnOffsetSurface
        - OnBrep
        - OnMesh
        - OnAnnotation

- **Puntos y vectores** *(no derivada de OnGeometry)*
    - On2dPoint (bueno para los puntos de espacio de los parámetros)
    - On3dPoint
    - On4dPoint (bueno para representar los puntos de control con **x,y,z** y **w** para “weight”)
    - On3dVector

- **Curvas** *(no derivada de OnGeometry)*
    - OnLine
    - OnPolyline (en realidad se deriva de OnPointArray)
    - OnCircle
    - OnArc
    - OnEllipse
    - OnBezierCurve

- **Superficies** *(no deriva de OnGeometry)*
    - OnPlane
    - OnSphere
    - OnCylinder
    - OnCone
    - OnBox
    - OnBezierSurface

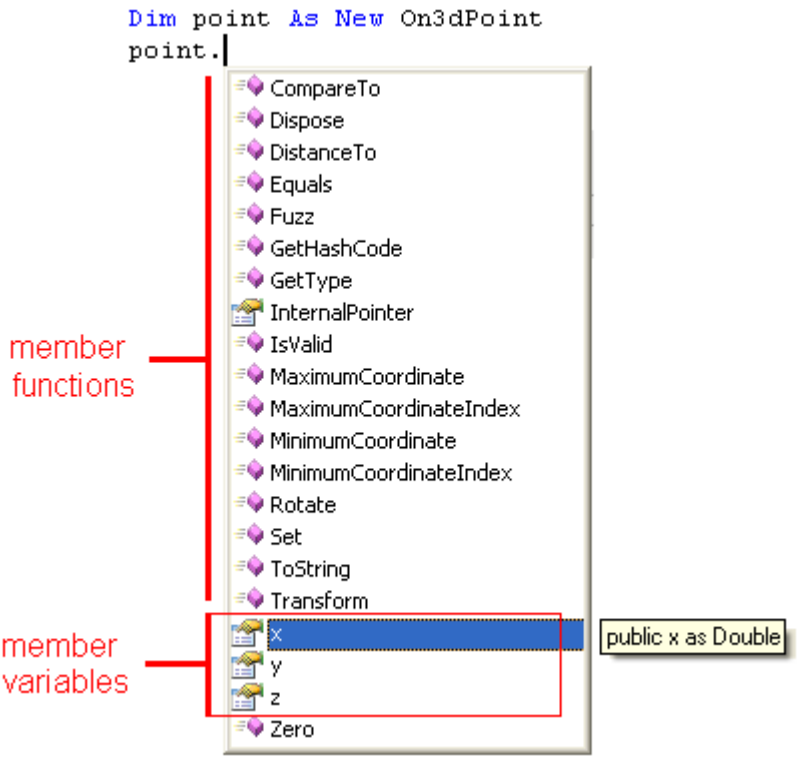
- **Misceláneos**
    - OnBoundingBox (Para cálcular la caja delimitadora de objetos)
    - OnInterval (Utilizado para los dominios de curvas y la superficies)
    - OnXform (Transformación geometrica de objetos: mover, rotar, escalar, etc.)
    - OnMassProperties (para calcular el volumen, área, centroide, etc)

15.4 Estructura de clases

Una clase típica (que es la estructura de datos definida por el usuario) tiene cuatro partes principales:

- **Constructor**: Se utiliza para crear una instancia de la clase.
- **Variables de miembro publica**: Aquí es donde se almacenan los datos de la clase. Las variables de miembro OpenNURBS suelen comenzar con "**m**_" para aislarse rápidamente.
- **Funciones de miembro público**: Esto incluye todas las funciones de clase para crear, actualizar y manipular las variables de miembro de la clase o realizar determinadas funciones.
- **Miembros privados**: se trata de funciones y variables de utilidad de clase de uso interno.

Una vez que usted inicie una clase, usted será capaz de ver todas las funciones de miembro de clase y variables de miembro a través de la función de autocompletado. Tenga en cuenta que cuando se despliegue sobre cualquiera de las funciones o variables, se muestra la firma de esa función. Una vez que usted comience a llenar los parámetros de la función, la función de autocompletar le mostrará en qué parámetro está y su tipo. Esta es una gran manera de navegar en funciones disponibles para cada clase. Aquí hay un ejemplo de la clase On3dPoint:



Copiar datos desde una clase existente a una nueva se puede hacer en una o varias formas, dependiendo de la clase. Por ejemplo, vamos a crear un On3dPoint nuevo y copiar el contenido de un punto existente en ella. Asi es como podemos hacerlo:

```
Use the constructor when you instantiate an instance of the point class
Dim new_pt as New On3dPoint( input_pt )

Use the "= operator" if the class provides one
Dim new_pt as New On3dPoint
new_pt = input_pt
```

```
You can use the “New” function if available
Dim new_pt as New On3dPoint
new_pt.New( input_pt )

There is also a  “Set” function sometimes
Dim new_pt as New On3dPoint
new_pt.Set( input_pt )

Copy member variables one by one.  A bit exhaustive method
Dim new_pt as New On3dPoint
new_pt.x =  input_pt.x
new_pt.y =  input_pt.y
new_pt.z =  input_pt.z

OpenNURBS geometry  classes provide “Duplicate” function that is very efficient to use
Dim new_crv as New OnNurbsCurve
new_crv = input_crv.DuplicateCurve()
```

15.5 Instancias constantes vs no-constantes

Rhino. NET SDK proporciona dos conjuntos de clases. La primera es constante, y sus nombres están precedidos por un "I", por ejemplo *IOn3dPoint*. La clase no-constante correspondiente tiene el mismo nombre sin el "I", por ejemplo *On3dPoint*. Puede duplicar una clase constante o ver sus variables de miembro y algunas de las funciones, pero no puede cambiar sus variables.

Rhino. NET SDK se basa en Rhino C++ SDK. El lenguaje de programación C++ ofrece la posibilidad de pasar instancias de clase constante y se utiliza en todas las funciones de SDK. Por otra parte, DotNET no tiene ese concepto y, por tanto no posee las dos versiones para cada clase.

15.6 Puntos y Vectores

Hay muchas clases que se podrían utilizar para almacenar y manipular puntos y vectores. Tomemos por ejemplo los puntos de doble precisión. Hay tres tipos de puntos:

Nombre clase	Variables de miembro	Notas
On2dPoint	**x** as Double **y** as Double	Se utilizan sobre todo para los puntos de espacio de parámetros. La "**d**" en el nombre de la clase implica doble precisión. Hay otras clases de puntos que tienen una "**f**" en el nombre que implican precisión única.
On3dPoint	**x** as Double **y** as Double	Más comúnmente utilizado para representar puntos en el espacio de coordenadas de tres dimensiones.
On4dPoint	**x** as Double **y** as Double **z** as Double **w** as Double	Se utiliza para los puntos de agarre. Posee información del “weight”, además de las tres coordenadas.

Las operaciones de puntos y vectores incluyen:

Vector Adición:

```
Dim add_v As New On3dVector = v0 + v1
```

Vector Substracción:

Dim subtract_vector As New On3dVector = v0 – v1

Vector entre dos puntos:

Dim dir_vector As New On3dVector = p1 – p0

Vector producto (si el resultado es positivo, los vectores van en la misma dirección):

Dim dot_product As Double = v0 * v1

Vector producto cruzado (resultado es un vector normal a los 2 vectores de entrada)

Dim normal_v As New On3dVector = OnUtil.ON_CrossProduct(v0, v1)

Escalar un vector:

Dim scaled_v As New On3dVector = factor * v0

Mover un punto por un vector:

Dim moved_point As New On3dPoint = org_point + dir_vector

Distancia entre 2 puntos:

Dim distance As Double = pt0.DistanceTo(pt1)

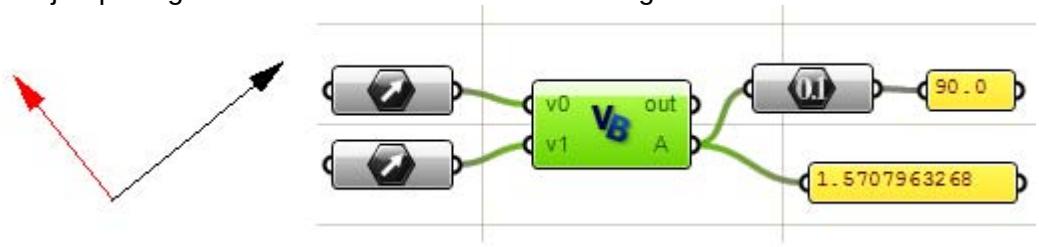
Obtener vector unitario (determina la longitud del vector a 1):

v0.Unitize()

Obtener longitud del vector:

Dim length As Double = v0.Length()

El ejemplo siguiente muestra cómo calcular el ángulo entre dos vectores.



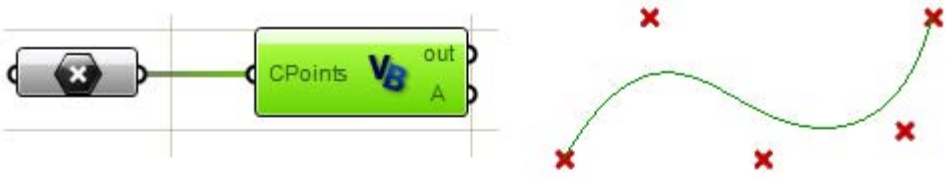

```
Sub RunScript(ByVal v0 As On3dVector, ByVal v1 As On3dVector)

  ' Unitize the input vectors
  v0.Unitize()
  v1.Unitize()
  Dim dot As Double = OnUtil.ON_DotProduct(v0, v1)

  ' Force the dot product of the two input vectors to
  ' fall within the domain for inverse cosine, which
  ' is -1 <= x <= 1. This will prevent runtime
  ' "domain error" math exceptions.
  If (dot < -1.0) Then dot = -1.0
  If (dot > 1.0) Then dot = 1.0

  A = System.Math.Acos(dot)

End Sub
```

15.7 Curvas OnNurbs

Con el fin de crear una curva NURBS, usted tendrá que proporcionar lo siguiente:

- Dimensión, que es típicamente = 3.
- Orden: grado de curva + 1.
- Los puntos de control (matriz de puntos).
- Vector nudo (matriz de números).
- De tipo de curva (anclada o periódica).

Hay funciones para ayudar a crear el vector de nudos, como veremos en breve, así que usted básicamente tiene que decidir sobre el grado y tiene una lista de puntos de control y usted es listo para empezar. El siguiente ejemplo crea una curva anclada.




```
Sub RunScript(ByVal CPoints As List(Of On3dPoint))

  'Create nurbs curve
  Dim dimension As Integer = 3
  Dim order As Integer = 4
  Dim nc As New OnNurbsCurve

  'Create open (Clamped) Nurbs Curve
  nc.CreateClampedUniformNurbs(dimension, order, CPoints.ToArray())

  'Assign curve to the output value A
  If( nc.IsValid() ) Then
    A = nc
  End If
End Sub
```

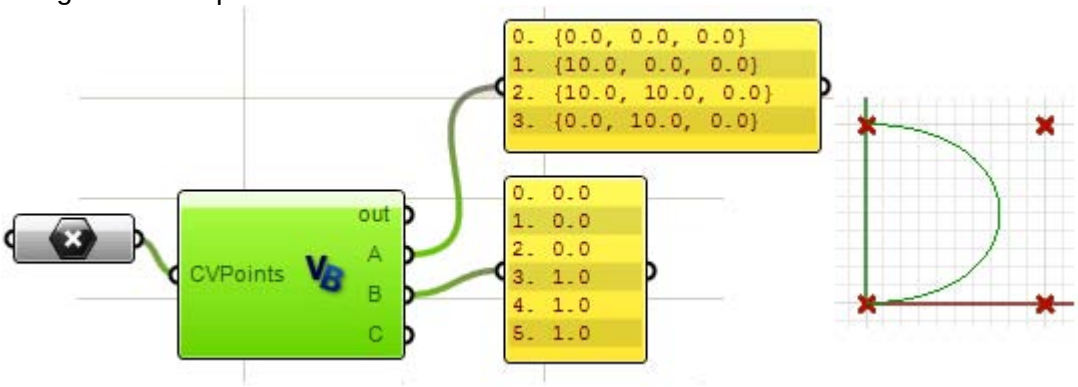
Para suavizar las curvas cerradas, debe crear las curvas de periódicas. Utilizando los mismos puntos de control de entrada y el grado de la curva, el siguiente ejemplo muestra cómo crear una curva periódica (continua).

```
Sub RunScript(ByVal CPoints As List(Of On3dPoint))

  'Create nurbs curve
  Dim dimension As Integer = 3
  Dim order As Integer = 4
  Dim nc As New OnNurbsCurve

  'Create closed (Periodic) Nurbs Curve
  nc.CreatePeriodicUniformNurbs(dimension, order, CPoints.ToArray())

  'Assign curve to the output value A
  If( nc.IsValid() ) Then
    A = nc
  End If
End Sub
```

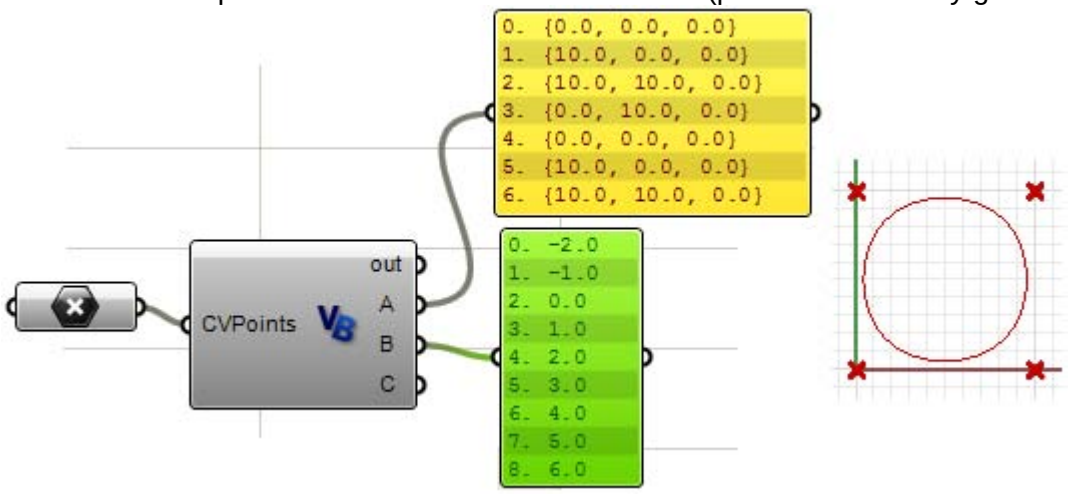
Curvas NURBS ancladas vs periódicas

Las curvas ancladas suelen ser curvas abiertas donde sus extremos coinciden con los puntos de control finales. Las curvas periódicas son curvas cerradas suaves. La mejor manera de comprender las diferencias entre ambas es a través de la comparación de sus puntos de control

El siguiente componente crea una curva NURBS anclada:

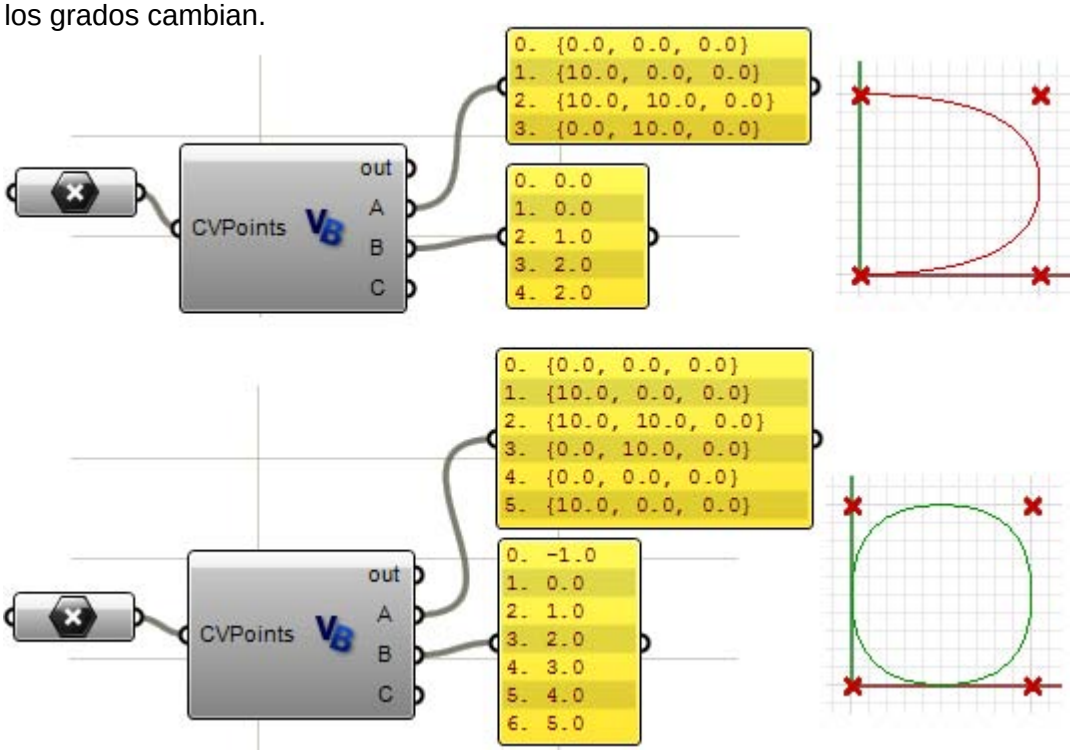


Esta es la curva periódica mediante la misma entrada (puntos de control y grado de curva):



Tenga en cuenta que la curva periódica volvió los cuatro puntos de entrada en siete puntos de control (4 + grado) "y mientras que la curva anclada utiliza sólo cuatro puntos de control. El vector de nudos de la curva periódica utiliza sólo dos nudos simples, mientras que la curva anclada comienza y termina con nudos de multiplicidad total.

Éstos son ejemplos de lo mismo pero con las curvas de grado 2. Como usted habrá notado, el número de puntos de control y nudos de las curvas periódicas cambia cuando los grados cambian.

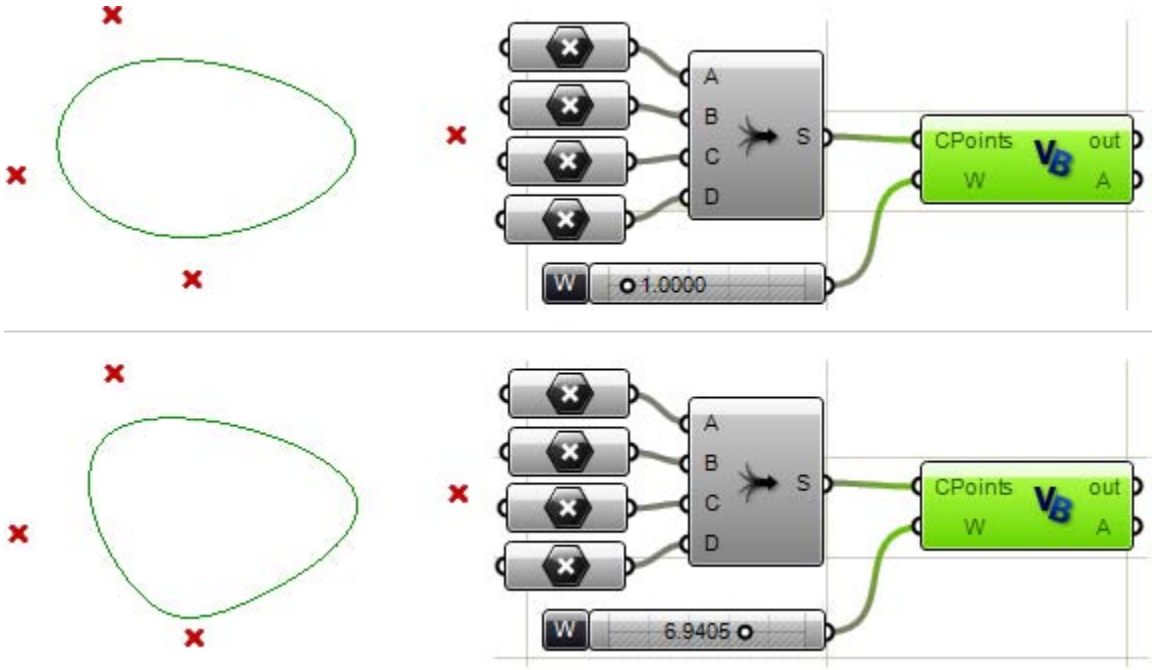


Este es el código utilizado para navegar a través de los puntos CV y los nudos en los ejemplos anteriores:

```
'Output control points
Dim count As Double = nc.CVCount()
Dim i As Integer
Dim cvs As New List( Of On3dPoint )
For i = 0 To count - 1
  Dim cv As New On3dPoint(0, 0, 0)
  nc.GetCV(i, cv)
  cvs.Add(cv)
Next

'Output knots
Dim knots As New List( Of Double )
count = nc.KnotCount()
For i = 0 To count - 1
  knots.Add(nc.Knot(i))
Next
```

Pesos (Weights)

Los pesos de los puntos de control en una curva NURBS uniforme se establece en 1, pero este número puede variar en las curvas NURBS racionales. El siguiente ejemplo muestra cómo modificar los pesos de los puntos de control de forma interactiva en Grasshopper.



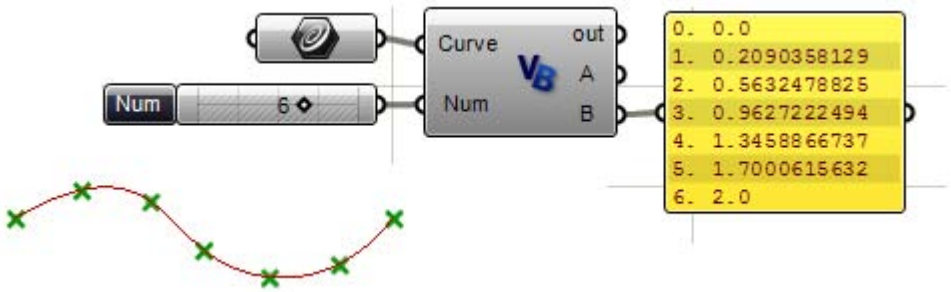

```
Sub RunScript(ByVal CPoints As List(Of On3dPoint), ByVal W As Double)

  Dim i As Integer

  'Create nurbs curve
  Dim dimension As Integer = 3
  Dim order As Integer = 4
  Dim cv_count As Integer = CPoints.Count
  Dim nc As New OnNurbsCurve
  nc.CreatePeriodicUniformNurbs(dimension, order, CPoints.ToArray())
  nc.MakeRational()

  'Assign weights
  Dim cv As New On3dPoint
  For i = 0 To cv_count - 1
    nc.GetCV(i, cv)
    cv = cv * W
    nc.SetCV(i, cv)
    nc.SetWeight(i, W)
  Next


  'Assign curve to the output value A
  If( nc.IsValid() ) Then
    A = nc
  End If
End Sub
```

Dividir curvas NURBS

La división de una curva en una serie de segmentos incluye los siguientes pasos:

- Encontrar el dominio de la curva que es el intervalo de espacio de los parámetros.
- Hacer una lista de los parámetros que dividen a la curva en segmentos iguales.
- Encontrar los puntos de la curva 3D.

El siguiente ejemplo muestra cómo lograrlo. Tenga en cuenta que hay una función global en el espacio de nombre RhUtil que divide una curva por el número de segmentos o longitud de arco que se puede utilizar directamente como se ilustrará más adelante.




```
Sub RunScript(ByVal Curve As OnCurve, ByVal Num As Integer)

  Dim min As Double = Curve.Domain().Min()
  Dim max As Double = Curve.Domain().Max()

  'Find the step value
  Dim step_value As Double = (max - min) / (Num - 1)

  Dim Points As New List( Of on3dPoint )

  Dim t_list(Num) As Double
  For i As Integer = 0 To Num
    t_list(i) = i / Num
  Next

  If (Curve.GetNormalizedArcLengthPoints(t_list, t_list)) Then
    For i As Integer = 0 To Num
      Dim pt As On3dPoint = Curve.PointAt(t_list(i))
      Points.Add(pt)
    Next
  End If


  A = Points
  B = t_list

End Sub
```

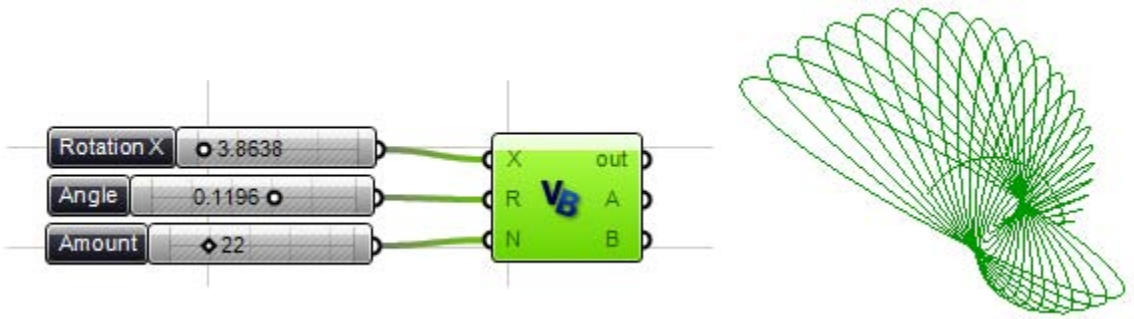
15.8 Clases de curvas no derivadas de OnCurve

A pesar de que todas las curvas se pueden representar como curvas NURBS, es útil a veces trabajar con otros tipos de geometrías de curvas. Una de las razones es porque

su representación matemática es más fácil de entender que NURBS y son típicamente más livianas. Es relativamente fácil conseguir la forma NURBS de esas curvas no derivadas de OnCurve cuando se requiere. Básicamente, usted necesita convertirlas a una clase correspondiente.
La siguiente tabla muestra la correspondencia:

Tipos de curvas	Tipos derivados de OnCurve
OnLine	OnLineCurve
OnPolyline	OnPolylineCurve
OnCircle	OnArcCurve or OnNurbsCurve (use GetNurbsForm() member function)
OnArc	OnArcCurve or OnNurbsCurve (use GetNurbsForm() member function)
OnEllipse	OnNurbsCurve (use GetNurbsForm() member function)
OnBezierCurve	OnNurbsCurve (use GetNurbsForm() member function)

Este es un ejemplo que usa las clases OnEllipse y OnPolyline:



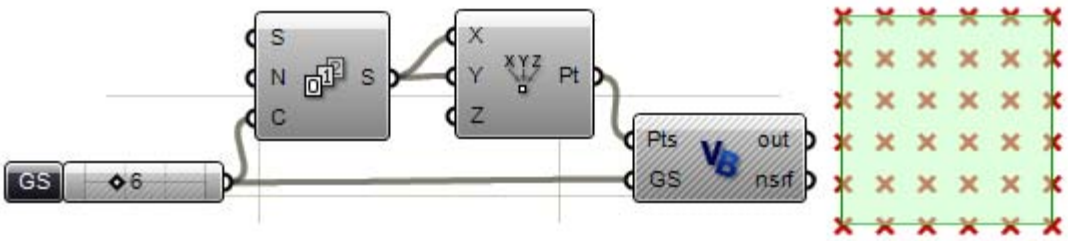

```
Sub RunScript(ByVal X As Object, ByVal R As Object, ByVal N As Object)
  'Declare a new list of OpenNURBS circles
  Dim c_list As New List(Of OnEllipse)

  'Declare list of lines
  Dim p_list As New On3dPointArray

  For i As Int32 = 1 To N
    'Declare a new circle
    Dim c As New OnEllipse(OnUtil.On_xy_plane, i / 2, i)
    'Rotate the circle
    C.Rotate(R * i, New On3dVector(0, 1, 0), New On3dPoint(X, 0, 0))
    'Add the circle to the list
    c_list.Add(c)
    'Add center point
    p_list.Append(C.Center())
  Next

  Dim polyline As New OnPolyline(p_list)
  'Assign the list to the output value A
  A = c_list
  'Assign polyline to output value B
  B = polyline
End Sub
```

15.9 Superficie OnNurbs

Similar a lo que hemos discutido en la clase de curvas OnNurbs, para crear una superficie OnNurbs usted necesita saber:

- Dimensión, que es típicamente = 3.
- Orden en la dirección U y V: grado + 1.
- Puntos de control.
- Vector nudo en la dirección U y V.
- Tipo de superficie (ancladas o periódicas).

El ejemplo siguiente crea una superficie NURBS desde una grilla de puntos de control:



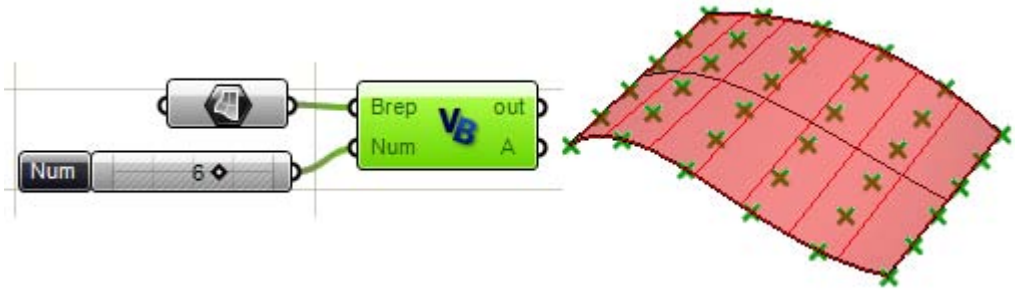

```
Sub RunScript(ByVal Pts As List(Of On3dPoint), ByVal GS As Integer)
  'Create a grid of points
  Dim Grid As New ArrayList()
  'Call grid function
  Call CreateGrid(Pts, Grid, GS)
  'Call create nurbs surface function
  Dim ns As OnNurbsSurface
  ns = CreateNS(Grid, GS)

  'Assign mid point to output
  nsrf = ns

End Sub
```

```
Sub CreateGrid( ByVal Pts As List(Of On3dPoint),
                ByRef Grid As ArrayList, ByVal GS As Integer )
  Dim i As Integer
  Dim j As Integer
  For i = 0 To Pts.Count() - 1 Step GS
    'Declare a row of points
    Dim Row As New List( Of On3dPoint )
    For j = i To i + GS - 1
      'Get a reference od the point
      Dim pt As On3dPoint
      pt = Pts(j)
      'Add point to the row
      Row.Add(pt)
    Next
    'Add row to the grid
    Grid.Add(Row)
  Next
End Sub
```

```
Function CreateNS(ByVal cvpoints As ArrayList,
                  ByVal GS As Integer) As OnNurbsSurface
  Const Degree As Integer = 3

  'Make the surface
  Dim orderU As Integer = Degree + 1
  Dim orderV As Integer = Degree + 1

  Dim ns As New OnNurbsSurface
  ns.Create(3, False, orderU, orderV, GS, GS)

  'Add cv points
  Dim i As Integer
  Dim j As Integer
  Dim pt As On3dPoint
  For i = 0 To GS - 1
    For j = 0 To GS - 1
      pt = cvpoints(i)(j)
      ns.SetCV(i, j, pt)
    Next
  Next

  'Set knots for open surface
  ns.MakeClampedUniformKnotVector(0)
  ns.MakeClampedUniformKnotVector(1)

  CreateNS = ns
End Function
```

Otro ejemplo común es dividir el dominio de una superficie. El ejemplo siguiente divide el dominio de la superficieio en un número igual de puntos en ambas direcciones (el número de puntos debe ser mayor que 1, para que tenga sentido) y se hace lo siguiente:

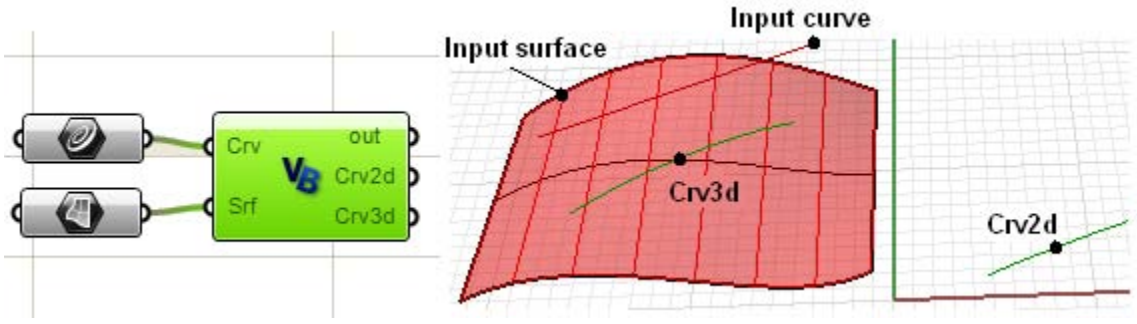
- Normalizar el dominio de la superficie (determinar el intervalo del dominio a 0-1).
- Calcular el valor del paso utilizando el número de puntos.
- Utiliza un loop anidado para calcular los puntos de la superficie usando los parámetros u y v.



```
Sub RunScript(ByVal Brep As OnBrep, ByVal Num As Integer)
  'Find step - Num must be > 1
  Dim StepValue As Double = 1 / (Num - 1)

  Dim nSrf As New OnNurbsSurface
  nSrf = Brep.Face(0).NurbsSurface

  'Normalize domain in u and v directions
  nSrf.SetDomain(0, 0, 1)
  nSrf.SetDomain(1, 0, 1)

  Dim Points As New List( Of on3dPoint )
  Dim i As Double = 0
  Dim j As Double = 0
  For i = 0 To 1 Step StepValue
    For j = 0 To 1 Step StepValue
      Dim Pt As New On3dPoint
      Pt = nSrf.PointAt(i, j)
      Points.Add(Pt)
    Next
  Next

  A = Points
End Sub
```

La clase OnSurface tiene muchas funciones que son útiles para manipular y trabajar con superficies. El siguiente ejemplo muestra cómo tirar una curva a una superficie.

Hay 2 salidas en el componente script de Grasshopper. La primera es el parámetro espacio de la curva (representación plana de la curva 3D en el plano xy) respecto al dominio de la superficie. El segundo es la curva en el espacio 3D. Tenemos la curva 3D a través de "empujar" el parámetro espacio de la curva en 2D sobre la superficie.



```
Sub RunScript(ByVal Crv As OnCurve, ByVal Srf As OnBrep)

  'Get pulled curve in 2D parameter space
  Dim pull_crv As OnCurve
  pull_crv = Srf.m_S(0).Pullback(Crv, doc.AbsoluteTolerance())

  'Get the pulled curve in 3D space
  Dim push_crv As OnCurve
  push_crv = Srf.m_S(0).Pushup(pull_crv, doc.AbsoluteTolerance())

  'Output both curves
  Crv2d = pull_crv
  Crv3d = push_crv

End Sub
```

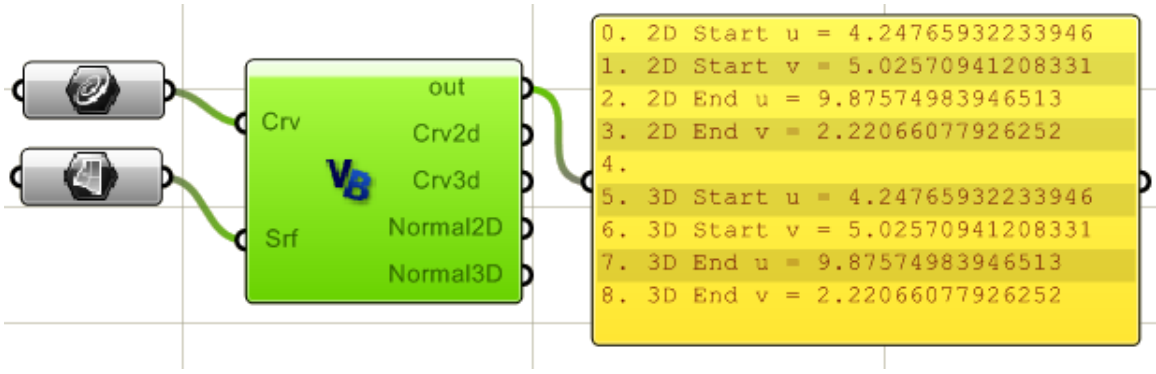
Utilizando el ejemplo anterior, vamos a calcular el vector normal de los puntos inicial y final de la curva ingresada. Aquí hay 2 formas de hacerlo:

- Utilice los puntos 2D de inicio y final de la curva ingresada y estos serán los puntos de inicio y final de los parámetros del espacio de la superficie.
- O utilice los puntos finales de la curva 3d, encuentre el punto más cercano a la superficie y utilice los parámetros resultantes para encontrar la superficie normal.

```
Sub GetEndNormals2D(ByVal crv2d As OnCurve,
                    ByVal srf As OnSurface,
                    ByRef EndVectors2D As List(Of On3dVector ))
  Dim start_normal As On3dVector
  Dim end_normal As On3dVector

  'find start and end points in parameter space
  Dim start2d As New On2dPoint
  start2d = crv2d.PointAtStart()
  Dim end2d As New On2dPoint
  end2d = crv2d.PointAtEnd()

  'Output parameters
  'Surface parameters are the x and y of the 2d curve end points
  Print("2D Start u = " & start2d.x)
  Print("2D Start v = " & start2d.y)
  Print("2D End u = " & end2d.x)
  Print("2D End v = " & end2d.y)
  Print("")

  'Call surface normal function
  start_normal = srf.NormalAt(start2d.x, start2d.y)
  end_normal = srf.NormalAt(end2d.x, end2d.y)

  EndVectors2D.Add(start_normal)
  EndVectors2D.Add(end_normal)
End Sub
```

```
Sub GetEndNormals3D(ByVal crv3d As OnCurve,
                    ByVal srf As OnSurface,
                    ByRef EndVectors3D As List(Of On3dVector ))

  'Declare start and end normal
  Dim start_normal As On3dVector
  Dim end_normal As On3dVector

  'Find start and end points in parameter space
  Dim start3d As New On3dPoint
  start3d = crv3d.PointAtStart()
  Dim end3d As New On3dPoint
  end3d = crv3d.PointAtEnd()

  'Declare parameters
  Dim u As Double
  Dim v As Double

  'Get surface closest point
  srf.GetClosestPoint(start3d, u, v)
  start_normal = srf.NormalAt(u, v)

  'Output start parameters
  Print("3D Start u = " & u)
  Print("3D Start v = " & v)

  srf.GetClosestPoint(end3d, u, v)
  end_normal = srf.NormalAt(u, v)

  'Output end parameters
  Print("3D End u = " & u)
  Print("3D End v = " & v)

  EndVectors3D.Add(start_normal)
  EndVectors3D.Add(end_normal)
End Sub
```

Esta es la imagen que muestra la salida del valor del parámetro en los puntos finales utilizando ambas funciones. Tenga en cuenta que ambos métodos no dan los mismos parámetros como se esperaba.

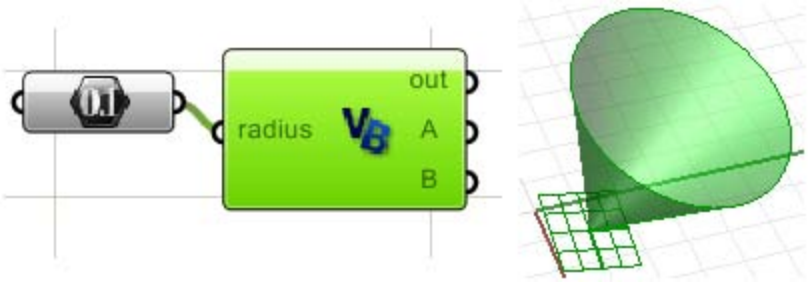


15.10 Clases de superficie no derivadas de OnSurface

OpenNURBS ofrece clases de superficie no derivas de OnSurface. Estas son definiciones matemáticas validas de superficies y pueden ser convertidas en tipos derivados de OnSurface para su uso en funciones que necesitan OnSurface. Aquí está una lista de clases de superficie y su correspondiente derivada de OnSurface:

Tipos basicos de superficies	Tipos derivados de OnSurface
OnPlane	OnPlaneSurface or OnNurbsSurface (use OnPlane.GetNurbsForm() function)
OnShpere	OnRevSurface or OnNurbsSurface (use OnShpere.GetNurbsForm() function)
OnCylinder	OnRevSurface or OnNurbsSurface (use OnCylinder.GetNurbsForm() function)
OnCone	OnRevSurface or OnNurbsSurface (use OnCone.GetNurbsForm() function)
OnBezierSurface	OnNurbsSurface (use GetNurbsForm() member function)

Este es un ejemplo que utiliza las clases OnPlane y OnCone:



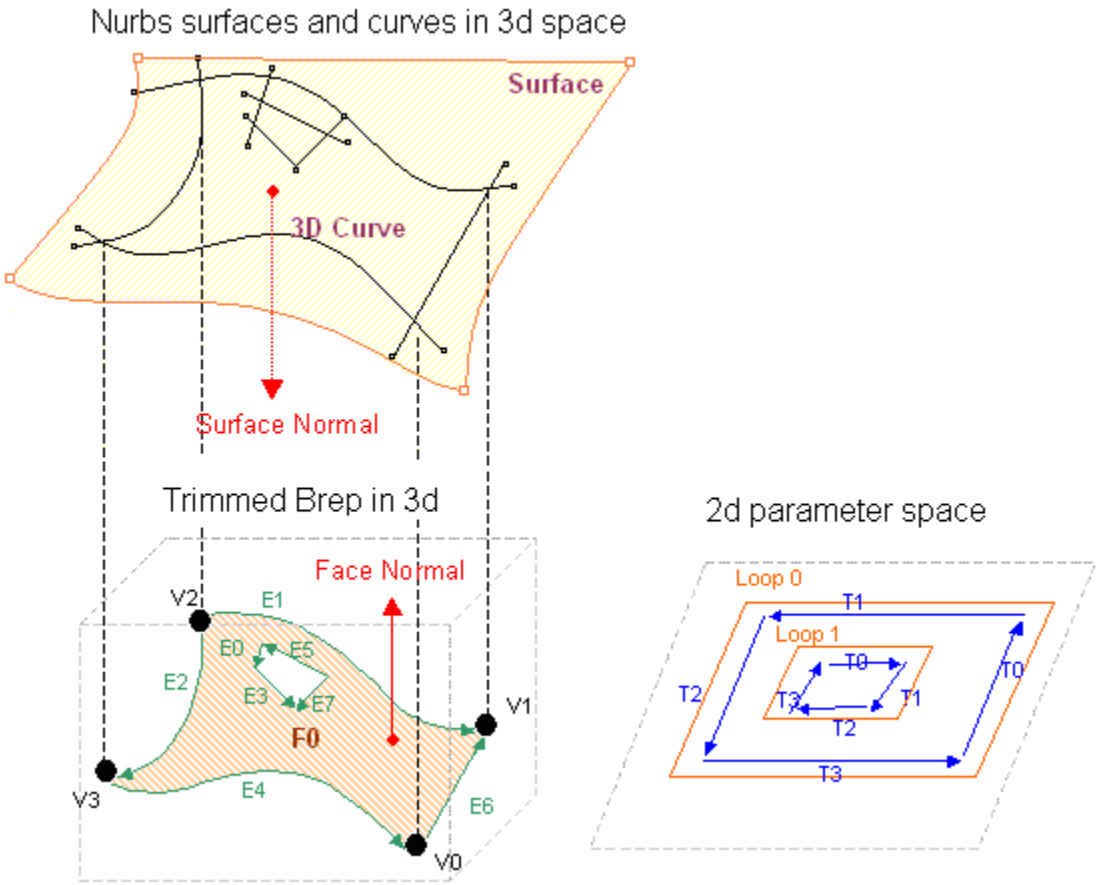

```
Sub RunScript(ByVal radius As Double)
  'Create a plane from origin and normal
  Dim plane As New OnPlane
  Dim origin As New On3dPoint(1, 1, 0)
  Dim normal As New On3dVector(1, 1, 3)
  plane.CreateFromNormal(origin, normal)

  'Define height value
  Dim height As Double = 5
  'Create cone
  Dim cone As New OnCone(plane, height, radius)

  'Assign output parameter
  A = cone
  B = plane
End Sub
```

15.11 Representación de Límites (OnBrep)

La representación de Fronteras (B-Rep) se utiliza para representar los objetos de forma inequívoca en cuanto a sus superficies límites. Usted puede pensar en OnBrep como tres partes diferenciadas:

- Geometría: la geometría 3D de curvas y superficies NURBS. También las curvas 2D del espacio paramétrico o curvas de corte.
- Topología 3D: caras, bordes y vértices. Cada una de las caras referencia a una superficie NURBS. La cara también reconoce todos los loops que pertenecen a esa cara. Los bordes referencian a curvas 3D. Cada borde tiene una lista de los cortes que usa el borde y también los dos vértices finales. Los vértices referencian puntos 3D en el espacio. Cada vértice también tiene una lista de las aristas que se encuentran en uno de sus extremos.
- Topología 2D: La representación 2D del espacio paramétrico de caras y aristas. En el espacio paramétrico, las curvas de corte 2D van o bien hacia la derecha o hacia la izquierda, dependiendo de si son parte de un loop exterior o interior de la cara. Cada cara válida tendrá que tener exactamente un lazo externo, pero pueden tener tantos loops internos como necesite (agujeros). Cada recorte referencia a uno de los bordes, 2 vértices finales, un loop y la curva en 2D.

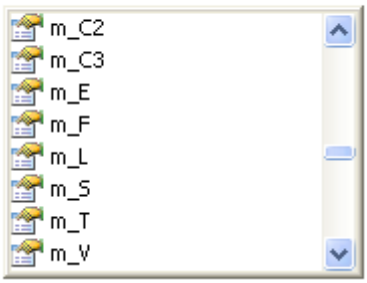
El siguiente diagrama muestra las tres partes y cómo se relacionan entre sí. La parte superior muestra la geometría subyacente de la superficie y curva nurbs 3D que definen una sola cara con un agujero. El centro es la topología 3D que incluye la cara “BREP”, bordes externos, bordes internos (cerrando un agujero) y los vértices. Luego está el espacio de los parámetros con los recortes y loops.



Variables de miembro OnBrep

Las variables de miembro OnBrep incluyen a todas las geometrías y topologias en 3D y 2D. Una vez que se crea una instancia de una clase BREP, usted puede ver todas las funciones y variables de miembro. La imagen siguiente muestra las funciones de miembro y la tabla enumera los tipos de datos con la descripción.

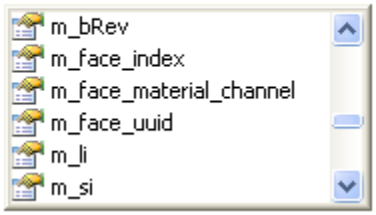
```
Dim brep As New OnBrep
brep.
```



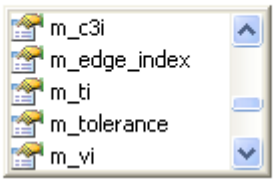
Miembros de la topología: describe las relaciones entre las partes del brep.	
OnBrepVertexArray **m_V**	Array of brep vertices (OnBrepVertex)
OnBrepEdgeArray **m_E**	Array of brep edges (OnBrepEdge)
OnBrepTrimArray **m_T**	Array of brep trims (OnBrepTrim)
OnBrepFaceArray **m_F**	Array of brep faces (OnBrepFace)
OnBrepLoopArray **m_L**	Array of loops (OnBrepLoop)
Miembros de la Geometría: datos de la geometría de las curvas y superficies 3d.	
OnCurveArray **m_C2**	Array of trim curves (2D curves)
CnCurveArray **m_C3**	Array of edge curve (3D curves)
ONSurfaceArray **m_S**	Array of surfaces

Observe que cada una de las funciones de miembro OnBrep son básicamente un conjunto de otras clases. Por ejemplo m_F es un conjunto de referencias a OnBrepFace. OnBrepFace es una clase derivada de OnSurfaceProxy y tiene variables y funciones de miembro propias. Aquí están las variables miembro de las clases OnBrepFace, OnBrepEdge, OnBrepVertex, OnBrepTrim y OnBrepLoop:

```
Dim brep_face As New OnBrepFace
brep_face.
```

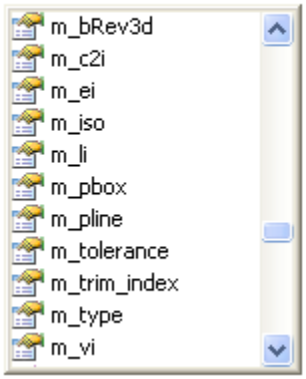


```
Dim brep_edge As New OnBrepEdge
brep_edge.
```

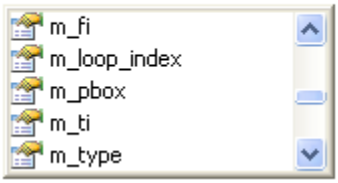



```
Dim brep_vertex As New OnBrepVertex
brep_vertex.
```

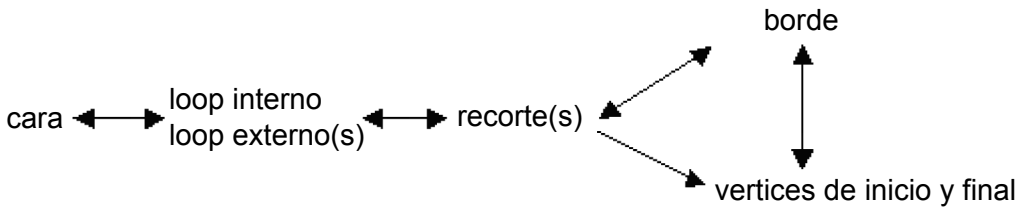

```
Dim brep_trim As New OnBrepTrim
brep_trim.
```



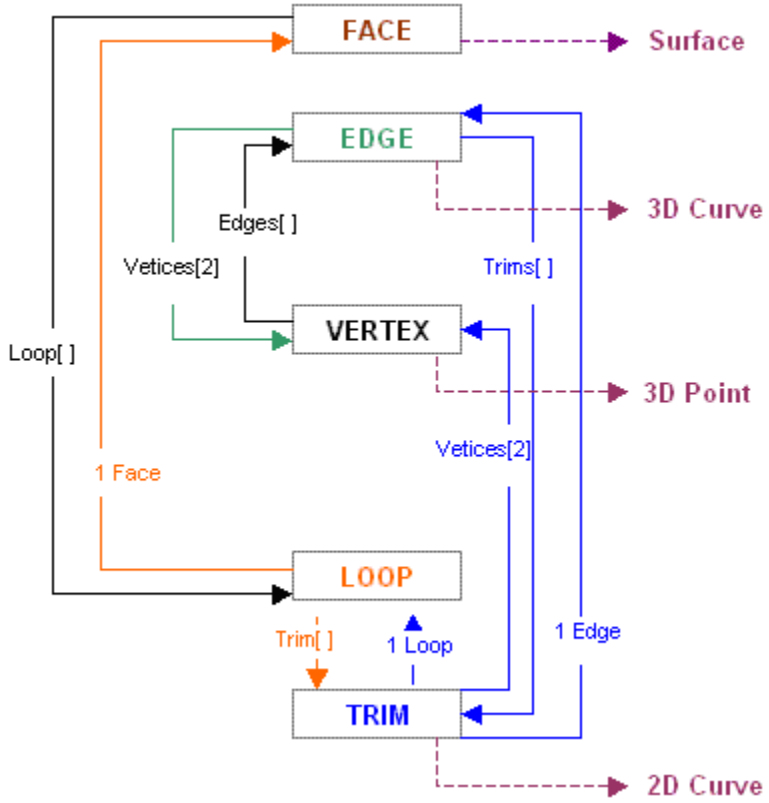

```
Dim brep_loop As New OnBrepLoop
brep_loop.
```



El siguiente diagrama muestra las variables de miembro OnBrep y como se referencian unas a otras. Usted puede utilizar esta información para llegar a cualquier parte de la BREP. Por ejemplo, cada cara conoce sus loops y cada ciclo tiene una lista de los recortes y desde estos se puede llegar a las orillas que está conectado y los 2 vértices finales y así sucesivamente.



Aquí hay otro diagrama más detallado de cómo las partes del BREP están conectadas entre sí, y cómo ir de una parte o la otra.



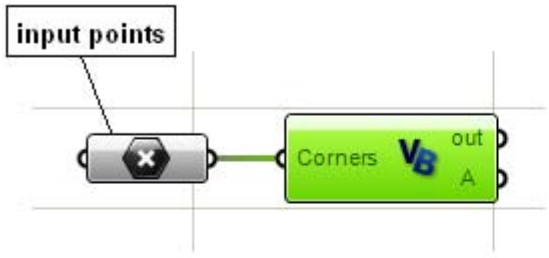
En los ejemplos próximos presentaremos cómo crear un OnBrep, navegar por las diferentes partes y extraer información del BREP. También le mostraremos cómo utilizar algunas de las funciones que vienen con la clase, así como funciones globales.

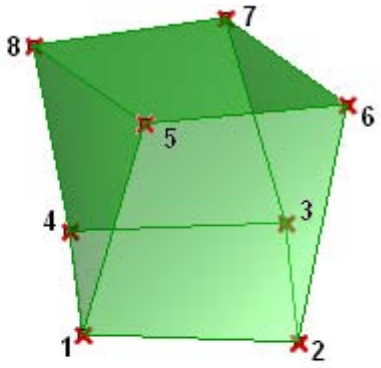
Crear un OnBrep

Existen distintas maneras de crear una nueva instancia de una clase de OnBrep:

- Duplicando un brep existente.
- Duplicando o extrayendo una cara en un brep existente.
- Usando la función que toma el OnSurface como un parámetro de entrada. Hay 5 funciones que utilizan distintos tipos de superficies:
  - desde SumSurface
  - desde RevSurface
  - desde PlanarSurface
  - desde OnSurface
- Las funciones de utilidad de uso global.
  - Desde OnUtil como ON_BrepBox, ON_BrepCone, etc.
  - Desde RhUtil como RhinoCreatEdgeurface o RhinoSweep1 entre otras.

Este es un ejemplo para crear una caja BREP desde los vértices.





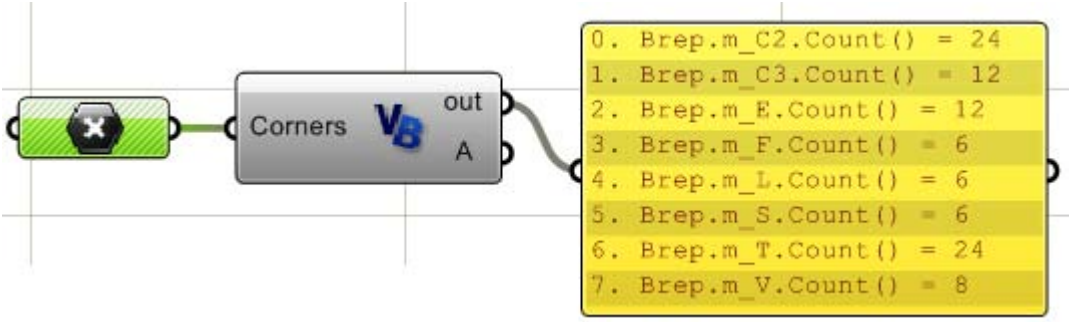

```
Sub RunScript(ByVal Corners As List(Of On3dPoint))

  ' Build the brep from corners
  Dim Brep As OnBrep = OnUtil.ON_BrepBox(Corners.ToArray())

  A = Brep
End Sub
```

Navegando en los datos de OnBrep

El siguiente ejemplo muestra cómo extraer los puntos vértices de una caja BREP

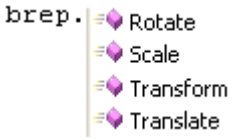
```
Sub RunScript(ByVal Corners As List(Of On3dPoint))

  ' Build the brep from corners
  Dim Brep As OnBrep = OnUtil.ON_BrepBox(Corners.ToArray())

  Dim myCorners As New List( Of On3dPoint )
  Dim v As OnBrepVertex
  Dim i As Integer

  For i = 0 To Brep.m_V.Count() - 1
    'get reference to OnBrepVertex
    v = Brep.m_V(i)

    'Get vertex point (loction)
    Dim pt As New On3dPoint
    pt = v.point

    'Add point to array
    myCorners.Add(pt)
  Next

  A = Brep
  B = myCorners
End Sub
```

El siguiente ejemplo muestra cómo obtener el número de la geometría y la topología de las piezas en una caja de BREP (caras, bordes, recortes, vértices, etc.)



Transformar OnBreps

Todas las clases derivadas de OnGeometry heredan cuatro funciones de transformación. Las tres primeras son probablemente las más utilizadas que son Rotar, Escalar y Transformar. Pero también hay una función genérica "Trabsform" que toma una matriz de transformación 4x4 definida con la clase OnXform. Vamos a discutir el OnXform en la siguiente sección.

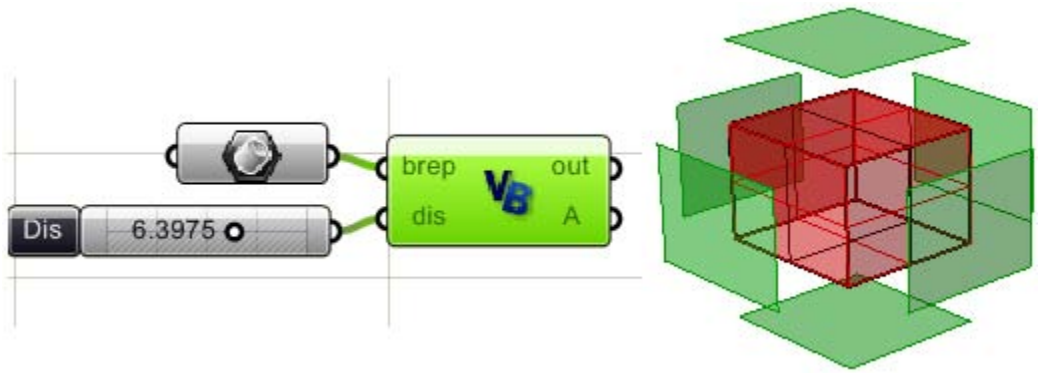


Editar OnBrep

La mayoría de las funciones de clases de miembro en OnBrep son las herramientas de usuario experto para crear y editar breps. Sin embargo, hay muchos funciones globales para Booleanas, intersectar o separa breps como se ilustra en una sección separada.

Existen ejemplos detallados disponibles en las muestras de DotNET de McNeel Wiki que crea un BREP desde el inicio y debería darle una buena idea de lo que se necesita para crear una OnBrep válido desde el principio.

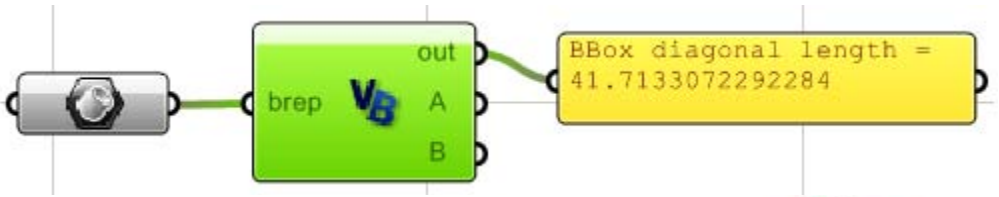
Aquí hay un ejemplo que extrae las caras de OnBrep y las desplaza del centro de la BREP. El ejemplo utiliza el centro de la caja delimitadora.



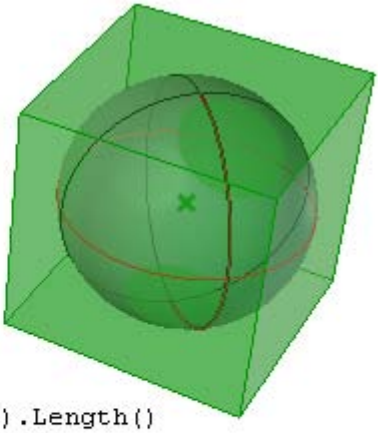
```
Sub RunScript(ByVal brep As OnBrep, ByVal dis As Double)

  Dim faces As New List(Of OnBrep)

  'Loop through brep faces to extract them
  For fi As Integer = 0 To brep.m_F.Count() - 1
    'Decalre new brep
    Dim face As New OnBrep
    face = brep.DuplicateFace(fi, False)

    'Add to faces array
    faces.Add(face)
  Next

  'Find brep bounding box center
  Dim center As New On3dPoint
  center = brep.BoundingBox().Center()

  'Loop through faces and move away from center by dis
  Dim dir As New On3dVector
  For i As Integer = 0 To faces.Count() - 1
    Dim face As OnBrep
    face = faces(i)

    'Find ceneter of each extracted face
    Dim face_center As On3dPoint
    face_center = face.BoundingBox().Center()

    'Find translation vector
    dir = face_center - center
    dir.Unitize()
    dir *= dis

    'Move face away from center
    face.Translate(dir)
  Next

  'Assign output
  A = faces


End Sub
```

Otras funciones de miembro OnBrep

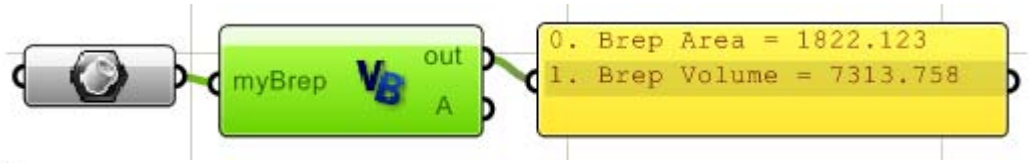
La clase OnBrep tiene muchas otras funciones que son heredadas de una clase padre o son específicas de la clase OnBrep. Todas las clases de geometría, incluyendo OnBrep, tienen un función de miembro llamada "BoundingBox()". Una de las clases de OpenNURBS es OnBoundingBox la cual entrega la geometría útil de delimitación de la información. Vea el siguiente ejemplo que encuentra un cuadro delimitador brep, su centro y la longitud de la diagonal.



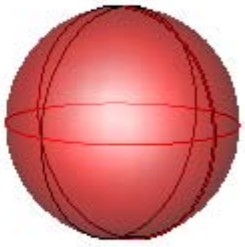

```
Sub RunScript(ByVal brep As OnBrep)

  'Find brep bounding box
  Dim bbox As New OnBoundingBox
  bbox = brep.BoundingBox()

  'Find bounding box ceneter
  Dim center As New On3dPoint
  center = bbox.Center()

  'Print bounding box diagonal length
  Dim length As Double = bbox.Diagonal().Length()
  Print("BBox diagonal length = " & length)

  A = bbox
  B = center

End Sub
```



Otra área que es útil conocer es la de las propiedades de masa. La clase OnMassProperties y algunas de sus funciones se ilustran en el siguiente ejemplo:

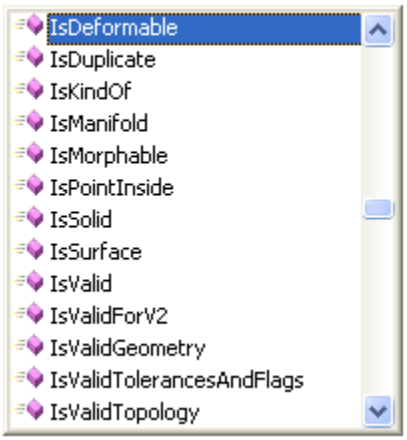



```
Sub RunScript(ByVal myBrep As Object)

  'Find and print brep area
  Dim a_mass As New OnMassProperties
  myBrep.AreaMassProperties(a_mass)
  Dim area As Double = a_mass.Area()
  Print("Brep Area = " & area)

  'Find and print brep volume
  Dim v_mass As New OnMassProperties
  myBrep.VolumeMassProperties(v_mass)
  Dim vol As Double = v_mass.Volume()
  Print("Brep Volume = " & vol)

End Sub
```

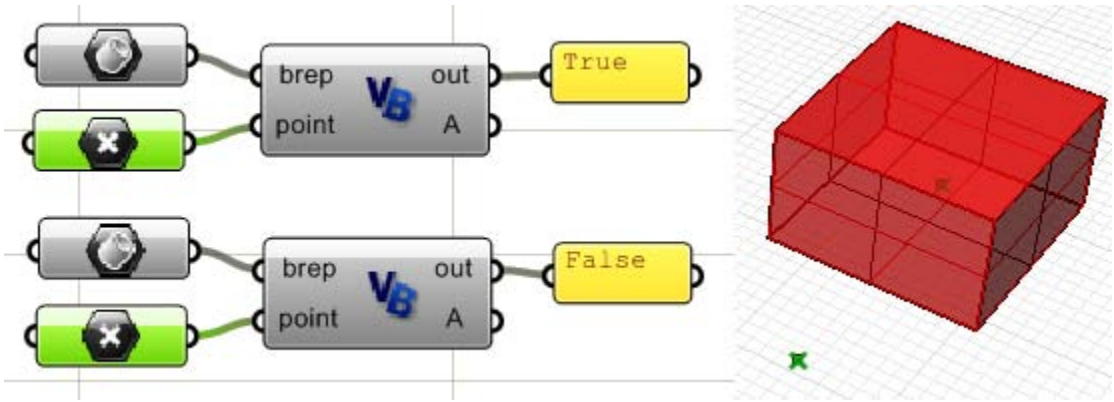


Hay unas pocas funciones que comienzan con "Is" las que normalmente retornan un valor booleano (verdadero o falso). Estos investigan acerca de la instancia BREP que usted está trabajando. Por ejemplo, si quieres saber si el BREP es una polisuperficie cerrada, a continuación, utilice la función OnBrep.IsSolid (). También es útil para comprobar si el BREP es válido o posee una geometría válida. Aquí está una lista de estas funciones de indagación en la clase OnBrep:

```
Dim brep As New OnBrep
brep.Is…
```



El siguiente ejemplo determina, si un punto está en el interior de un BREP:

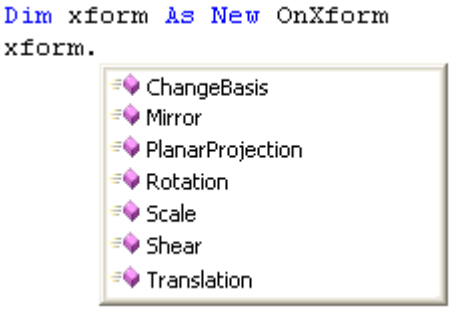


Aquí está el código para comprobar si un punto está dentro:

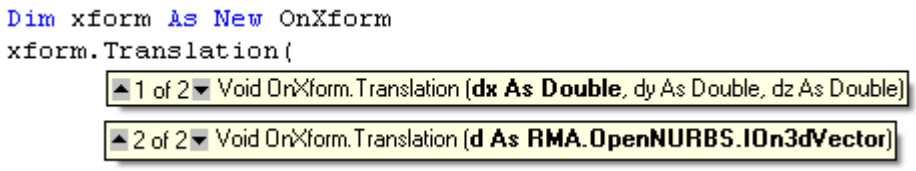
```
Sub RunScript(ByVal brep As OnBrep, ByVal point As On3dPoint)
  'Test if input point is inside brep
  Dim tol As Double = doc.AbsoluteTolerance()
  Dim stricktly_inside As Boolean = True
  Dim is_inside As Boolean

  'Call brep function to test the point
  is_inside = brep.IsPointInside(point, tol, stricktly_inside)

  Print(is_inside)

End Sub
```

15.12 Transformaciones de geometría

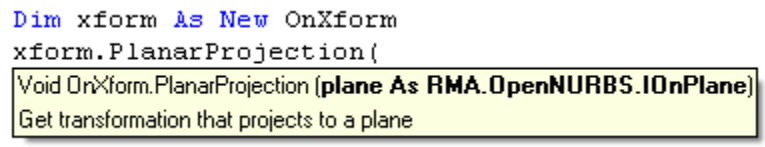
OnXform es una clase para almacenar y manipular la matriz de transformación. Esto incluye, pero no limitado a, la definición de una matriz para mover, rotar, escalar o cortar objetos. OnXform's m_xform es una matriz 4x4 de números de doble precisión. La clase también tiene funciones que soportan las operaciones de la matriz como la inversa y la transpuesta. He aquí algunas de las funciones de miembros en relación con la creación de diferentes transformaciones.

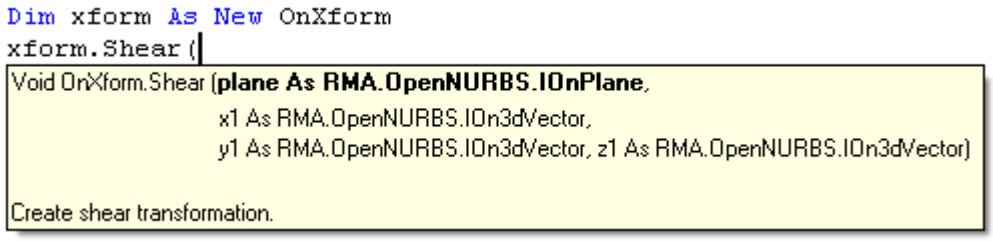


Una buena característica de auto-completado (disponible para todas las funciones) es que una vez que una función está seleccionada, el auto-completado muestra todas las funciones sobrecargadas. Por ejemplo, la traducción o bien acepta tres números o un vector, como se muestra en la imagen.

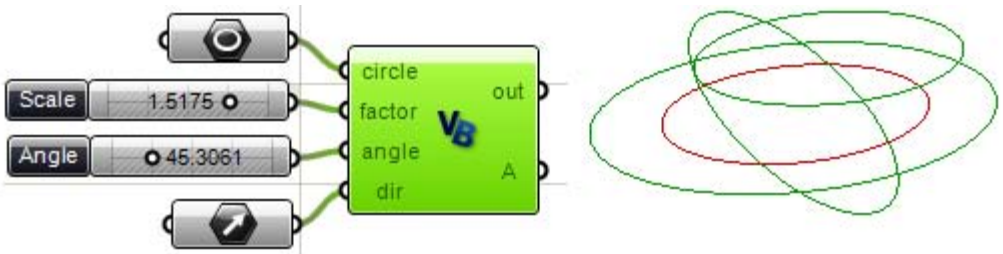


Aquí están algunas funciones OnXform:





El ejemplo siguiente tiene un círculo de entrada y tres círculos de salida. El primero es una copia escala del círculo original, el segundo es un círculo rotado y el tercero es uno trasladado.



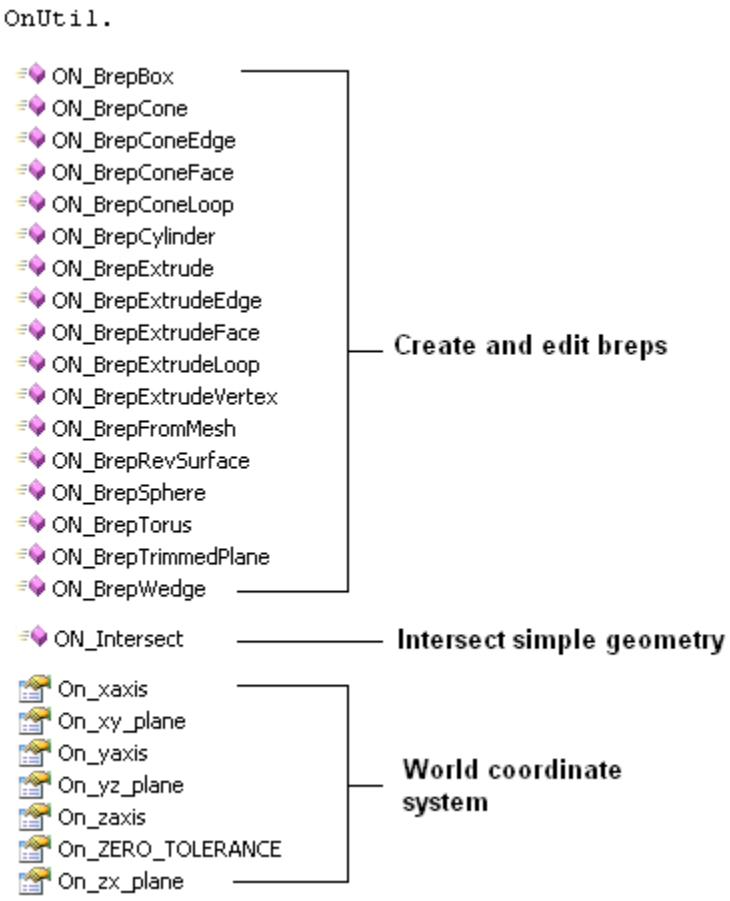

```
Sub RunScript(ByVal circle As OnCircle,
              ByVal factor As Double,
              ByVal angle As Double, ByVal dir As On3dVector)

  Dim circles As New List( Of OnCircle)

  'Scaled circle
  Dim scale As New OnXform
  scale.Scale(OnUtil.On_origin, factor)
  Dim s_circle As New OnCircle(circle)
  s_circle.Transform(scale)
  circles.Add(s_circle)

  'Rotated circle
  Dim rotate As New OnXform
  rotate.Rotation(angle, OnUtil.On_yaxis, OnUtil.On_origin)
  Dim r_circle As New OnCircle(circle)
  r_circle.Transform(rotate)
  circles.Add(r_circle)

  'Moved circle
  Dim move As New OnXform
  move.Translation(dir)
  Dim m_circle As New OnCircle(circle)
  m_circle.Transform(move)
  circles.Add(m_circle)

  'Assign output
  A = circles

End Sub
```

15.13 Funciones de utilidad global

Aparte de las funciones de miembro que vienen dentro de cada clase, Rhino .NET SDK proporciona las funciones globales bajo los espacios de nombre OnUtil y RhUtil. Veremos algunos ejemplos que utilizan algunas de estas funciones.

OnUtil

Aquí hay algunas de las funciones disponibles en OnUtil que están relacionadas con la geometría:



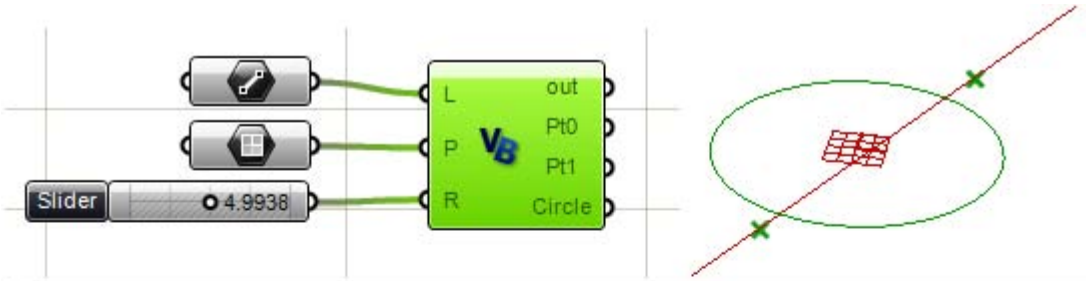
Intersecciones en OnUtil

La función de utilidad ON_Intersect tiene 11 funciones sobrecargadas. Aquí está una lista de geometrías intersectadas y los valores de retorno (la anteposición de "I" como en "IOnLine", significa que una instancia constante es pasada):

Geometría intersectada	salida
IOnLine with IOnArc	Line parameters (t0 & t1) and Arc points (p0 & p1)
IOnLine with IOnCircle	Line parameters (t0 & t1) and circle points (p0 & p1)
IOnSphere with IOnShere	OnCircle
IOnBoundingBoc with IOnLine	Line parameters (OnInterval)
IOnLine with IOnCylinder	2 points (On3dPoint)
IOnLine with IOnSphere	2 points (On3dPoint)
IOnPlane with IOnSphere	OnCircle
IOnPlane with IOnPlane with IOnPlane	On3dPoint

IOnPlane with IOnPlane	OnLine
IOnLine with IOnPlane	Parameter t (Double)
IOnLine with IOnLine	Parameters a & b (on first and second line as Double)

Aquí hay un ejemplo del resultado de la intersección de una línea y un plano con una esfera:




```
Sub RunScript(ByVal L As OnLine, ByVal P As OnPlane, ByVal R As Double)

  Dim point0 As New On3dPoint
  Dim point1 As New On3dPoint
  Dim circle0 As New OnCircle

  'Declare the sphere
  Dim sphere As New OnSphere(OnUtil.On_origin, R)

  'Intersect line with sphere
  OnUtil.ON_Intersect(L, sphere, point0, point1)

  'Intersect plane with sphere
  OnUtil.ON_Intersect(P, sphere, circle0)

  'Assign output
  Pt0 = point0
  Pt1 = point1
  Circle = circle0
End Sub
```

RhUtil

La Utilidad de Rhino (RhUtil) tiene muchas más funciones relacionadas con la geometría. La lista se amplía con cada nueva versión basada en las solicitudes del usuario. Esta es una imagen instantánea de las funciones relacionadas con la geometría:

RhUtil.

Points

- RhinoArePointsCoplanar
- RhinoPointInPlanarClosedCurve
- RhinoProjectPointsToBreps
- RhinoIsPointInBrep
- RhinoIsPointOnFace

Curve

- RhinoConvertCurveToPolyline
- RhinoCurveBrepIntersect
- RhinoCurveFaceIntersect
- RhinoDivideCurve
- RhinoDoCurveDirectionsMatch
- RhinoExtendCrvOnSrf
- RhinoExtendCurve
- RhinoExtendLineThroughBox
- RhinoExtrudeCurveStraight
- RhinoExtrudeCurveToPoint
- RhinoFairCurve
- RhinoFitCurve
- RhinoFitLineToPoints
- RhinoInterpCurve
- RhinoInterpolatePointsOnSurface
- RhinoMakeCubicBeziers
- RhinoMakeCurveClosed
- RhinoMakeCurveEndsMeet
- RhinoMergeCurves
- RhinoOffsetCurve
- RhinoOffsetCurveOnSrf
- RhinoPlanarClosedCurveContainmentTest
- RhinoPlanarCurveCollisionTest
- RhinoProjectCurvesToBreps
- RhinoPullCurveToMesh
- RhinoRebuildCurve
- RhinoRemoveShortSegments
- RhinoRepairCurve
- RhinoShortPath
- RhinoSimplifyCurve
- RhinoSimplifyCurveEnd

Surface

- RhinoChangeSeam
- RhinoCreateSurfaceFromCorners
- RhinoExtendSurface
- RhinoFitSurface
- RhinoIntersectSurfaces
- RhinoMakeG1Surface
- RhinoRailRevolve
- RhinoRebuildSurface
- RhinoRepairSurface
- RhinoRetrimSurface
- RhinoRevolve
- RhinoSrfControlPtGrid
- RhinoSrfPtGrid

Mesh

- RhinoMeshBooleanDifference
- RhinoMeshBooleanIntersection
- RhinoMeshBooleanSplit
- RhinoMeshBooleanUnion
- RhinoMeshBox
- RhinoMeshCone
- RhinoMeshCylinder
- RhinoMeshObjects
- RhinoMeshOffset
- RhinoMeshPlane
- RhinoMeshSphere
- RhinoRepairMesh
- RhinoSplitDisjointMesh
- RhinoUnifyMeshNormals

Brep

- RhinoBooleanDifference
- RhinoBooleanIntersection
- RhinoBooleanUnion
- RhinoBrepCapPlanarHoles
- RhinoBrepClosestPoint
- RhinoBrepGet2dProjection
- RhinoBrepGet2dSection
- RhinoBrepSplit
- RhinoCreate1FaceBrepFromPoints
- RhinoCreateEdgeSrf
- RhinoIntersectBreps
- RhinoJoinBrepNakedEdges
- RhinoJoinBreps
- RhinoMakePlanarBreps
- RhinoMergeAdjoiningEdges
- RhinoMergeBrepCoplanarFaces
- RhinoMergeBreps
- RhinoRepairBrep
- RhinoSplitBrepFace
- RhinoStraightenBrep
- RhPlanarRegionBoolean
- RhPlanarRegionDifference
- RhPlanarRegionIntersection
- RhPlanarRegionUnion
- RhinoSdkLoft
- RhinoSdkLoftSurface
- RhinoSweep1
- RhinoSweep2

Utility

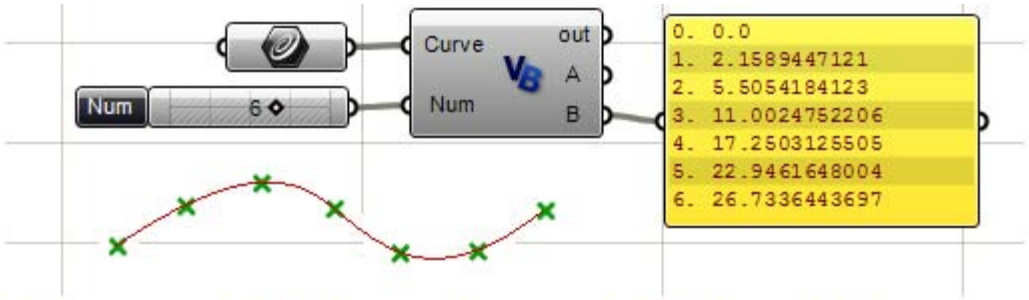
- RhinoActiveCPlane
- RhinoApp
- RhinoFitPlaneToPoints
- RhinoPlaneThroughBox
- RhinoProjectToPlane
- RhinoTriangulate3dPolygon

División de curva RhUtil

Es posible dividir una curva por el número de segmentos o la longitud de la curva utilizando la función de utilidad RhUtil.RhinoDivideCurve. Esta es un desglose de los parámetros de la función:

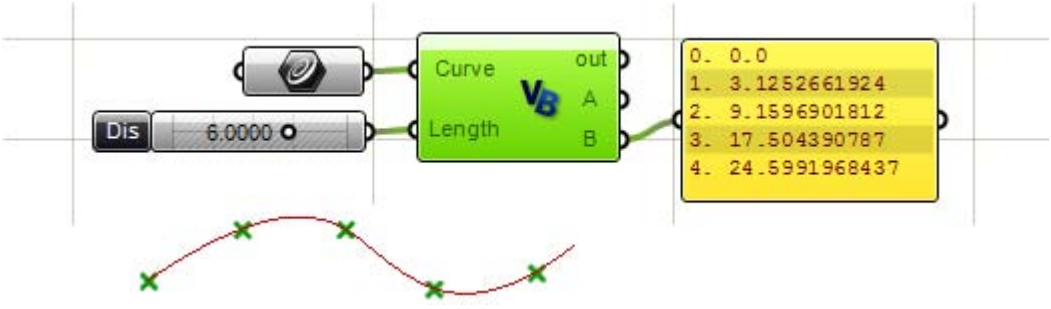
RhinoDivideCurve: Nombre de función.
Curve: Curva constante a dividir.
Num: Número de segmentos.
Len: Longitud de la curva para dividir por.
False: Opción para invertir la curva (se puede establecer en Verdadero o Falso).
True: Incluir el punto final (se puede establecer en Verdadero o Falso).
crv_p: Lista de los puntos de división.
crv_t: Lista de los parámetros de los puntos de división en la curva.

Ejemplo de división de la curva por el número de segmentos:




```
Sub RunScript(ByVal Curve As OnCurve, ByVal Num As Integer)

  Dim crv_p As New On3dPointArray
  Dim crv_t As New Arraydouble

  'A utility function to divide by number or curve length
  RhUtil.RhinoDivideCurve(Curve, Num, 0, False, True, crv_p, crv_t)

  A = crv_p
  B = crv_t

End Sub
```

Ejemplo de división por la longitud de la curva:



```
Sub RunScript(ByVal Curve As OnCurve, ByVal Len As Double)

  Dim crv_p As New On3dPointArray
  Dim crv_t As New Arraydouble

  'A utility function to divide by number or curve length
  RhUtil.RhinoDivideCurve(Curve, 0, Len, False, True, crv_p, crv_t)

  A = crv_p
  B = crv_t

End Sub
```

Curva a través de puntos RhUtil (curva de interpolación)

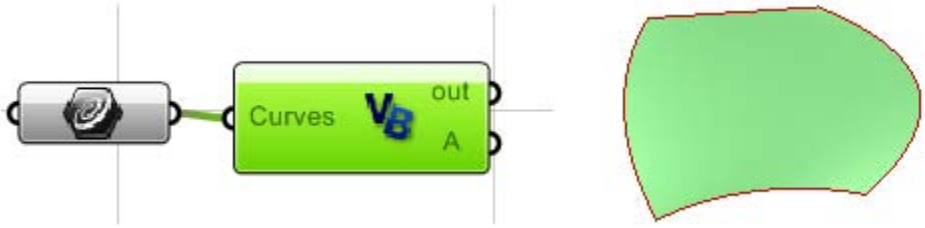
RhinoInterpCurve: Nombre de la función.
3: Grado de la curva.
pt_array: Puntos para crear una curva.
Nothing: Tangente de inicio.
Nothng: Tangente final.
0: Nudos uniformes.

El siguiente ejemplo toma como entrada una lista de On3dPoints y entrega una curva NURBS que pasa por estos puntos.

```
Sub RunScript(ByVal Points As List(Of On3dPoint))

  Dim pt_array As New ArrayOn3dPoint
  Dim i As Integer

  For i = 0 To Points.Count() - 1
    pt_array.Append(Points(i))
  Next

  'Create an interpolated nurbs curve
  Dim crv As New OnNurbsCurve
  crv = RhUtil.RhinoInterpCurve(3, pt_array, Nothing, Nothing, 0)

  If( crv.IsValid() ) Then
    'Set return vlue to list
    A = crv
  End If

End Sub
```

Creación de superficie desde bordes RhUtil

La entrada en el ejemplo siguiente es una lista de cuatro curvas y el resultado es una superficie con sus respectivos bordes.




```
Sub RunScript(ByVal Curves As List(Of OnCurve))

  Dim nc_list(3) As OnNurbsCurve
  For i As Integer = 0 To 3
    nc_list(i) = New OnNurbsCurve()
  Next

  ' Get nurb form of each curve
  For i As Integer = 0 To 3
    Curves(i).GetNurbForm(nc_list(i))
  Next

  ' Create the edgesurface
  Dim Brep As OnBrep = RhUtil.RhinoCreateEdgeSrf(nc_list)

  A = Brep
End Sub
```

16 *Ayuda*

Dónde encontrar más información acerca de Rhino DotNET SDK

Hay una gran cantidad de información y muestras en McNeel Wiki. Activamente los desarrolladores añaden nuevos materiales y ejemplos todo el tiempo. Este es un gran recurso y lo puedes encontrar aquí:

http://en.wiki.mcneel.com/default.aspx/McNeel/Rhino4DotNetPlugIns.html

Foros y grupos de discusión

La comunidad de Rhino es bastante activa y el foro de debate Grasshopper es un buen lugar para comenzar.

http://grasshopper.rhino3d.com/

También puedes enviar preguntas al grupo de noticias para desarrolladores de Rhino. Puedes acceder al grupo de noticias de desarrolladores desde la página del desarrollador de McNeel:
http://www.rhino3d.com/developer.htm

Depurando con Visual Studio

Para obtener un código más complejo que es difícil de depurar dentro del componente de secuencia de comandos de Grasshopper, puedes utilizar Visual Studio Express que Microsoft proporciona de forma gratuita o la versión completa de “Developer Studio”.

Los detalles de dónde obtener el Visual Studio Express y cómo utilizarlo estan aquí:
http://en.wiki.mcneel.com/default.aspx/McNeel/DotNetExpressEditions

Si usted tiene acceso a la versión completa de Visual Studio, entonces es incluso mejor. Compruebe la página siguiente para ayudarle a conseguir las opciones:
http://en.wiki.mcneel.com/default.aspx/McNeel/Rhino4DotNetPlugIns.html

Muestras de secuencias de comandos de Grasshopper

La galeria y foro de discusión de Grasshopper tiene muchos ejemplos de los componentes de secuencias de comandos que usted probablemente encontrará útil.

Notas