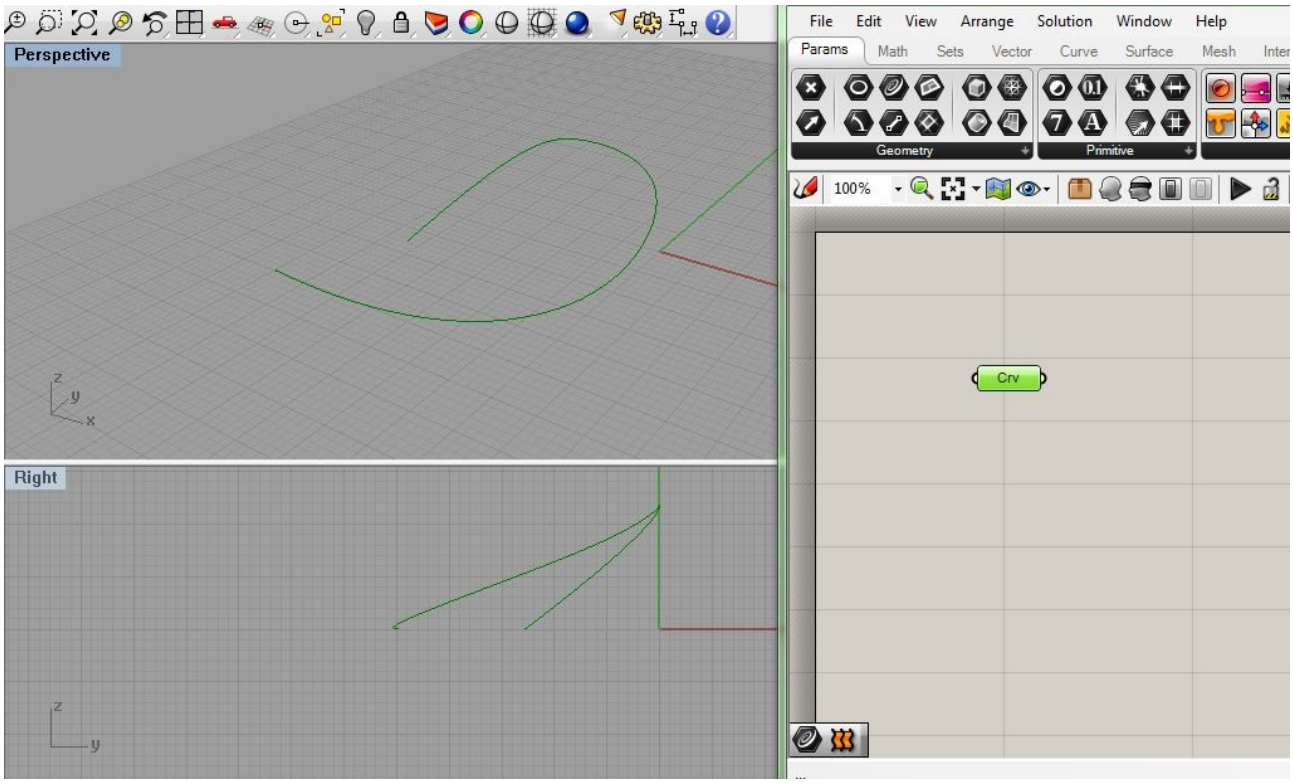
Tutorial e lavoro su altre caratteristiche di Grasshopper
Arch. A. Di Raimo

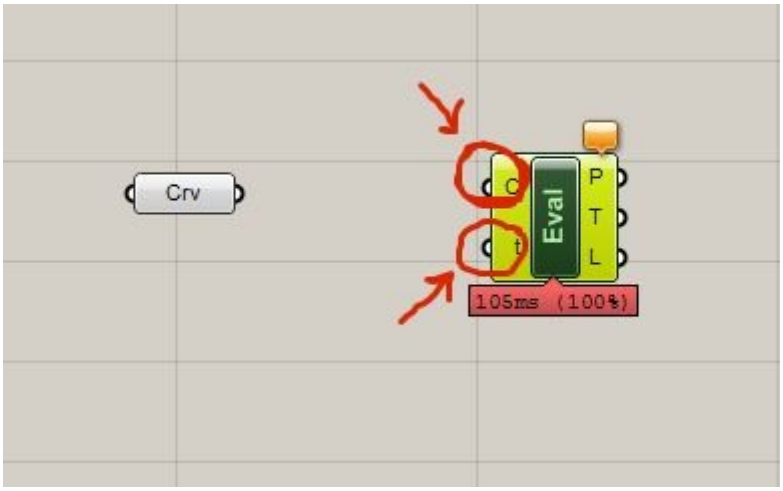
Abbiamo già introdotto il concetto di ***parametro****:* come abbiamo osservato si tratta ancora di una variabile che ci permette di considerare gli enti geometrici con i quali lavoriamo in maniera unitaria. In sostanza, nel caso di una curva ad esempio se noi diciamo che la sua lunghezza totale è unitaria, possiamo descriverla immaginando che una variabile possa descriverla completamente al variare dei valori che questa assume dallo 0 fino al valore 1.

Per capire meglio il senso del parametro disegniamo una curva in Rhino, prendiamo il component Curve, trasciniamolo nel canvas e col comando *Set One Curve,* selezioniamo la curva sulla scena:



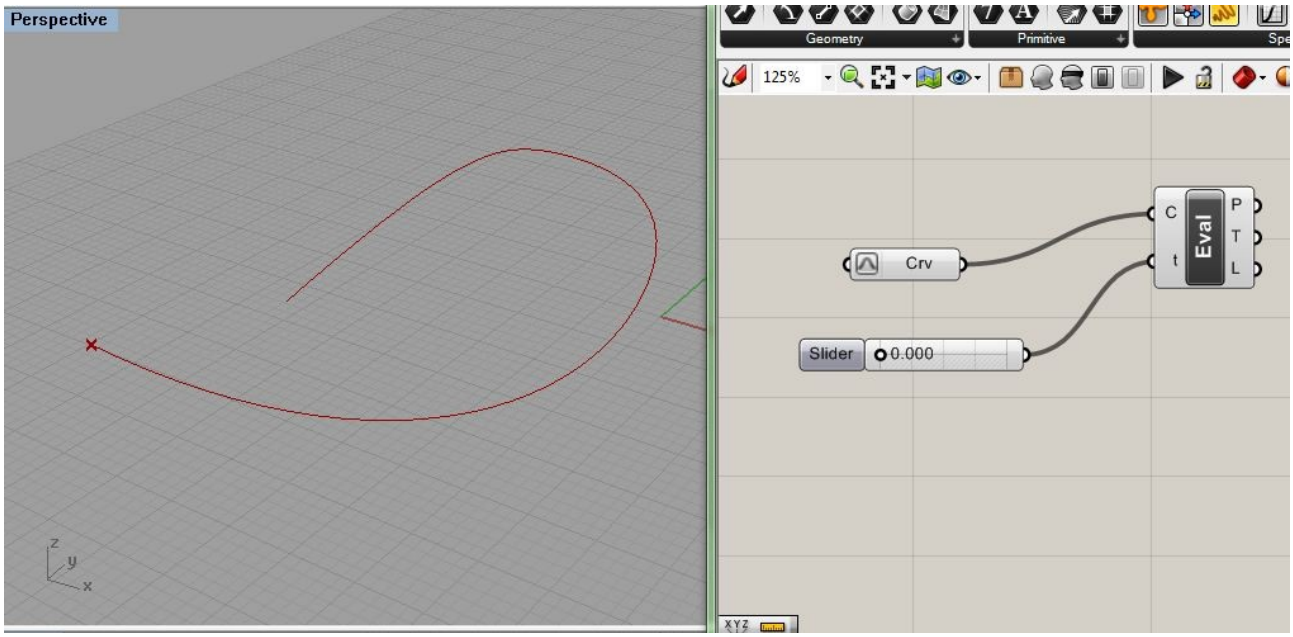
Grasshopper ora, ha immagazzinato per così dire, la curva sulla scena; cerchiamo di capire quali sono gli strumenti che ci offre, per lavorare con essa in maniera proficua.

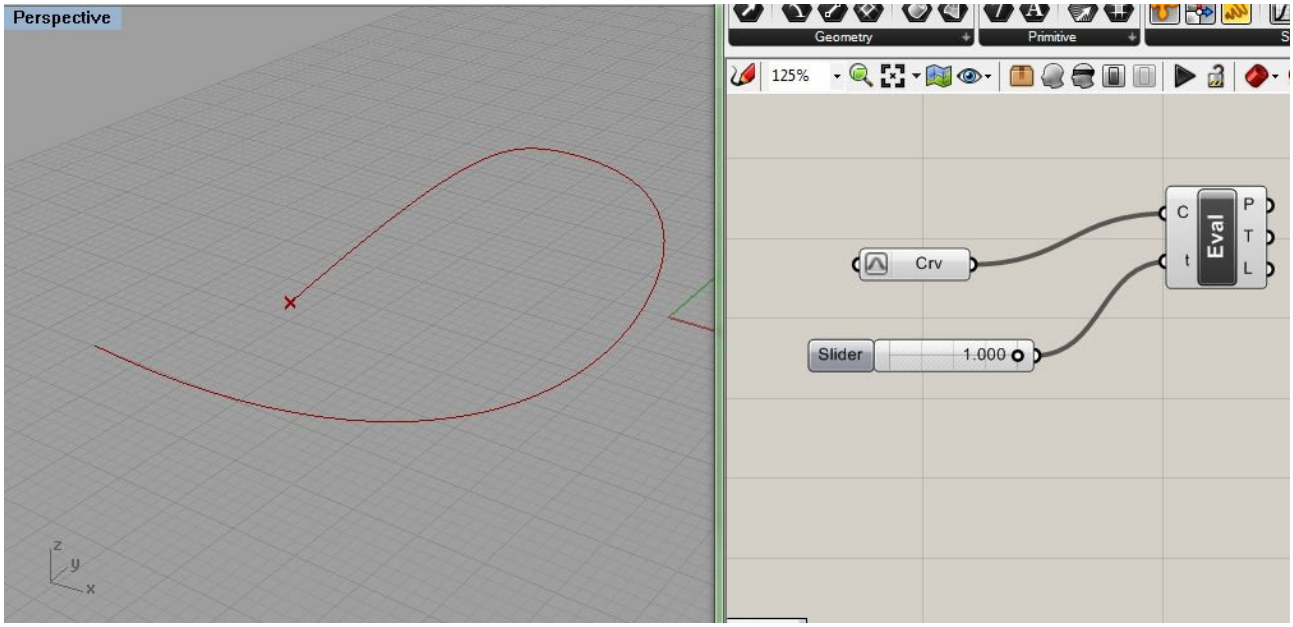
Come già abbiamo visto negli altri tutorial, un componente chiave in questo senso è *Evaluate Curve,* che troviamo naturalmente, nel set relativo alle curve (*Curve),* sotto la voce *Analysis.* Il che è quasi ovvio. Osserviamo il component, sia negli input che negli output; ne vale la pena:
A sinistra (input che arrivano al component) troviamo due ingressi; uno che rappresenta la curva da analizzare (C) e l'altro relativo al parametro.



Quest'ultimo viene chiamato *t,* il che è un preciso riferimento alla rappresentazione di curve parametriche, come sarà sicuramente capitato nei corsi di Matematica, dove invece di rappresentare le curve in funzione x, e y, viene utilizzato il parametro t, che viene fatto variare fra 0 e 1.
Allora, a questo punto possiamo connettere il *component* Crv relativo alla curva, con *Evaluate Curve;* allo stesso tempo secondo voi come possiamo gestire l'ingresso relativo al parametro *t*? Se ci pensate abbiamo bisogno di un intervallo di numeri variabile fra 0 e 1. Allora, se ricordate quanto abbiamo detto sullo *slider,* ciò di cui abbiamo bisogno è proprio uno slider che ci fa variare il parametro *t,* fra 0 e 1, considerando tutti i valori *float* che può assumere (non proprio tutti ovviamente e già sapete

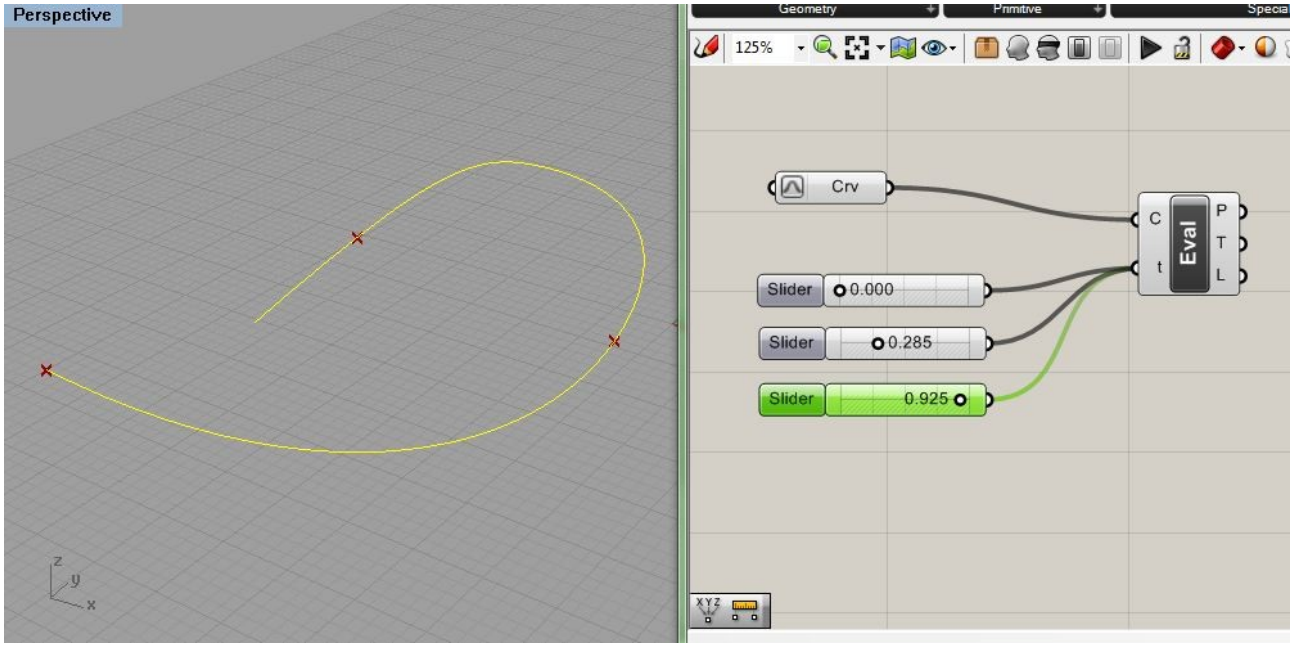
perchè).
Ora, inserite il component slider, dal pannello params, (con i valori che ha di default va benissimo), e, abbiate cura di *riparametrizzare la* curva (selezionando il relativo component, è cliccando col tasto dx alla relativa voce), e notate che succede se ponente lo slider sul valore 0, o sul valore 1: stiamo considerando appunto cosa succede sulla curva di lunghezza unitaria:





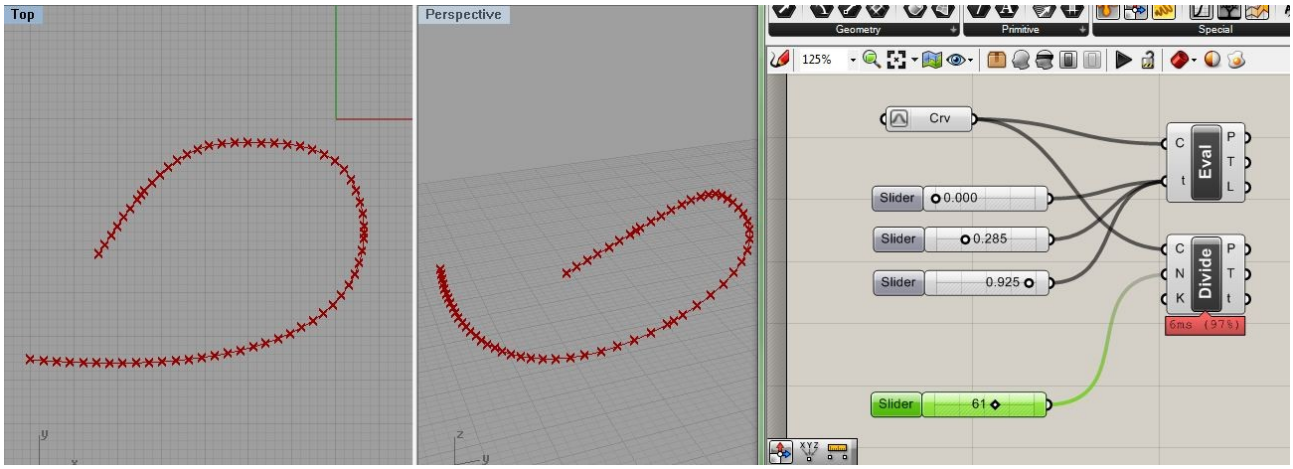
Naturalmente, ragionando in questo modo, se voglio trovare 'la posizione a metà della curva', mi basta porre il valore dello slider a ½, ovvero a 0.5.

Può sorgere l'esigenza di visualizzare più punti sulla stessa curva. Un modo per ottenere contemporaneamente queste diverse posizioni, è quello di associare più slider al parametro *t.* Qui sotto ad esempio ne abbiamo messi 3 per avere contemporaneamente il punto iniziale, quello a metà, e infine quello finale:



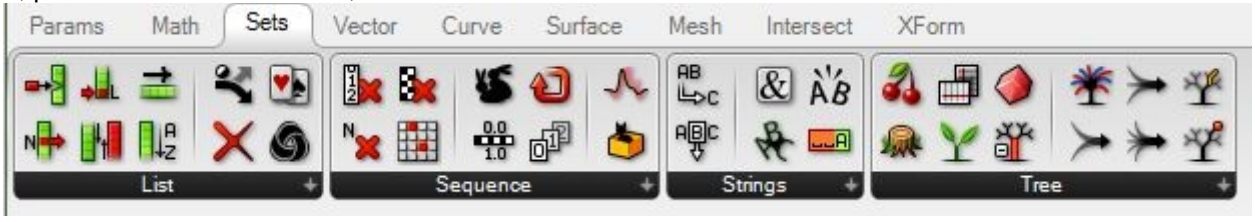
Diamo uno sguardo ora, ai valori *P, T, L,* in uscita dal nostro component. Il primo, *P,* rappresenta il punto o i punti, che abbiamo definito nello slider, attraverso le rispettive triplette di valori; il second *T,* rappresenta le tangenti ai punti, ed infine il terzo *L,* ci rappresenta la posizione lungo la curva in termini di lunghezza. (Se ci avviciniamo col mouse notiamo che il primo punto è collocato alla lunghezza 0.0, com'è lecito aspettarsi...)

utilizziamo ancora un altro component per lavorare sulla nostra curva; il component *Divide Curve,* può agevolarci a lavorare sulla curva stessa; abbiamo infatti in ingresso la curva da dividere (C) che possiamo associare al component, semplicemente connettendola. Abbiamo poi N, che sul component ci rappresenta il numero di segmenti nei quali vogliamo dividere la curva stessa; viene quasi spontaneo associare uno slider per la divisione fra un valore min. ed un max (definiamo lo slider operante secondo una variabile *integer* ovviamente):



Otteniamo quindi i punti di suddivisione sulla curva; comodo no?

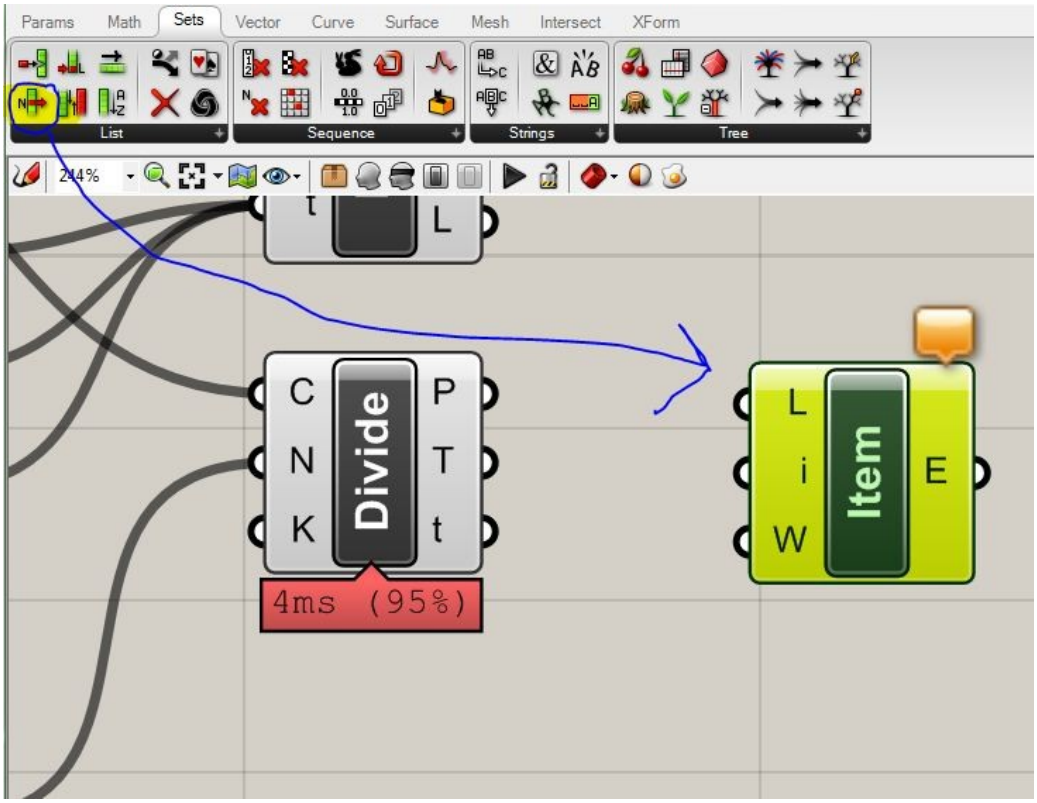
Ora però proviamo ad utilizzare questi dati *output* che abbiamo fin qui ottenuto. Se proviamo a strisciare con il mouse sul punto P, possiamo notare che a seconda del numero di divisione otteniamo altrettante triplette di valori +1, (poiché se divido per 1 avrò ad esempio due valori per ogni estremo; se divido per 2 avrò 3 valori dei punti e così via); quindi utilizziamo questi valori immagazzinandoli nel component *List Item* dal pannello *Sets.* Questo è un pannello molto importante, poiché è qui che troviamo tutti i comandi che ci permettono di operare con i diversi dati via via ottenuti, al pari di un database nel quale possiamo ottenere liste di valori, possiamo ordinare tali liste, scambiarne l'ordine e così via.



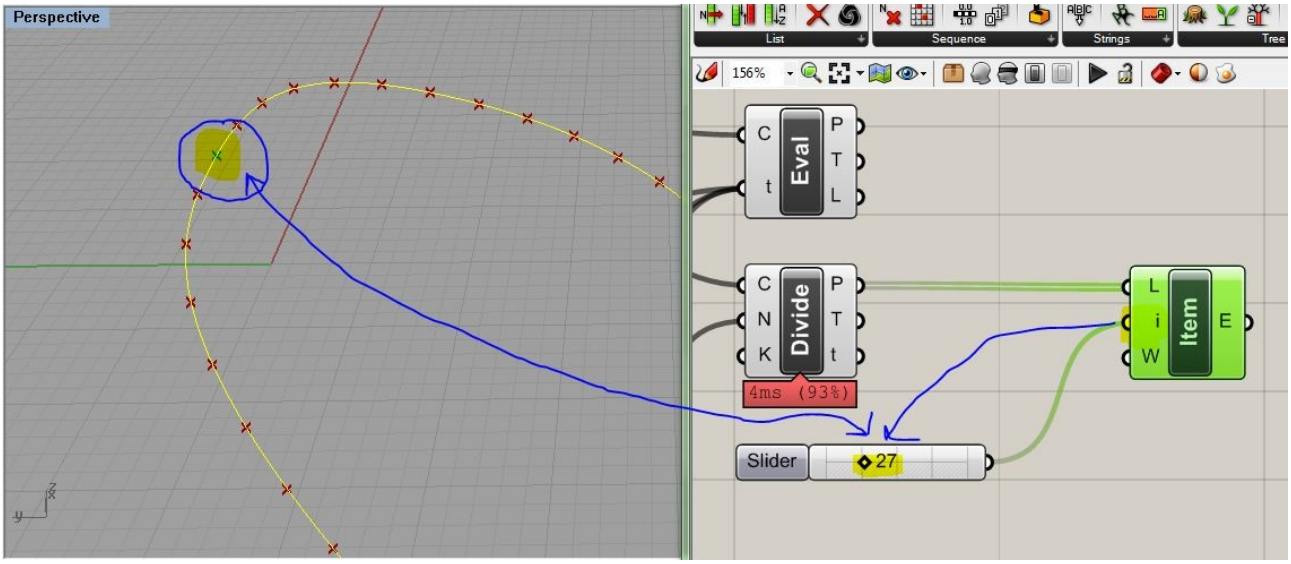
Notiamo anche a destra i component del tipo *Tree,* che ci permettono di manipolare i dati secondo delle strutture progressivamente arborescenti; ne parleremo.
Ora una cosa da capire è che ogni dato che abbiamo a disposizione comporta diverse informazioni, e tali informazioni proprio perché sono tali, ci permettono di volta in volta di eseguire operazioni diverse. Ad esempio un punto sulla nostra curva può comportare diversi dati: quelli relativi alla sua posizione espressa in coordinate cartesiane dello spazio di lavoro (queste sono informazioni di posizione, ottenute secondo una convenzione determinata, che è quella appunto dello spazio cartesiano);

possiamo però anche leggere l'informazione della posizione del punto in relazione alla posizione del punto rispetto alla lunghezza unitaria della curva (qui la convenzione è che la curva viene assunta di lunghezza pari a 1, quindi un punto all'inizio sarà relativo alla lunghezza 0, uno a metà alla lunghezza ½ = 0.5, ed uno alla fine sarà alla lunghezza 1). Infine altra informazione utile è l'indice (*Index)* di un elemento considerato facente parte di un oggetto più grande. Ad esempio i vertici di un oggetto mesh, (chi usa 3dsMax lo sa), sono caratterizzati dall'avere un indice che è un numero intero, così come i vertici di una spline.

Quando inseriamo il component *List Item,* abbiamo la possibilità di specificare tali valori:

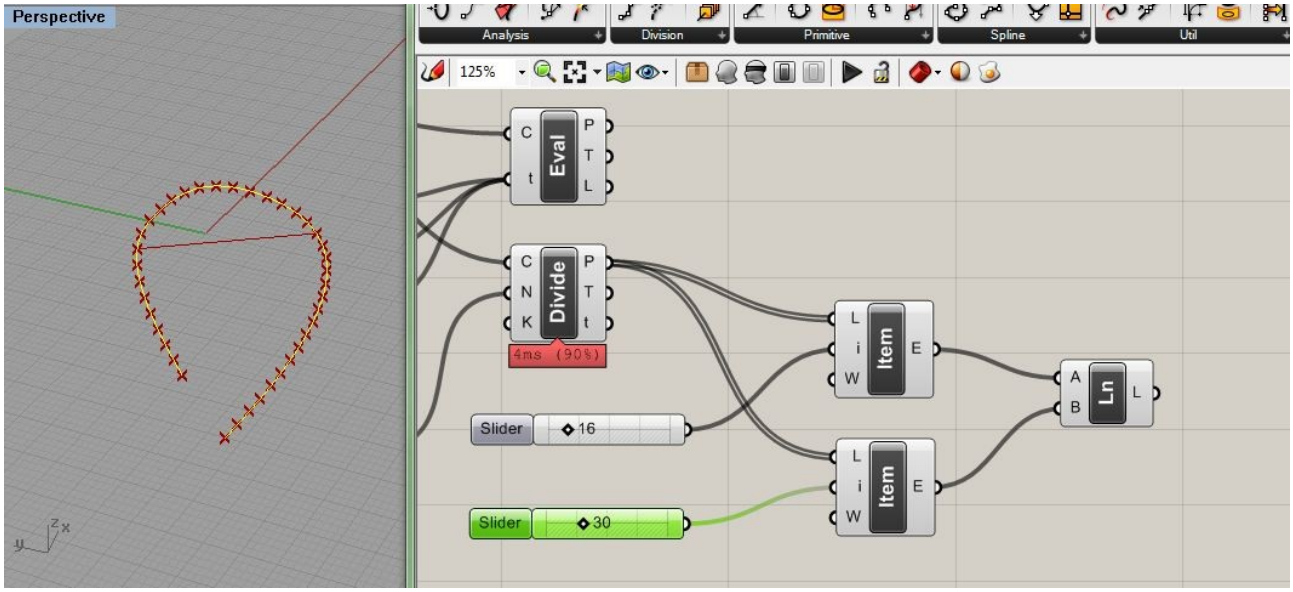


Notiamo che per valori in input la L, si riferisce proprio alla lista di valori, quindi è quasi spontaneo associare la L con l'Output P del component *Divide.* E così il component contiene la lista immagazzinata. Ora però io voglio un valore solo della lista. Se notiamo sempre a sinistra abbiamo in input la *i* che rappresenta appunto l'indice di un elemento della lista con il quale voglio lavorare. Siccome questo indice nella nostra curva dipende direttamente dal numero di suddivisioni che ho effettuato precedentemente tramite lo slider, possiamo copiare e incollare lo slider stesso e scegliere così l'elemento di indice *i.* Notiamo che cosa succede per i punti della nostra curva, quando inseriamo un certo indice:

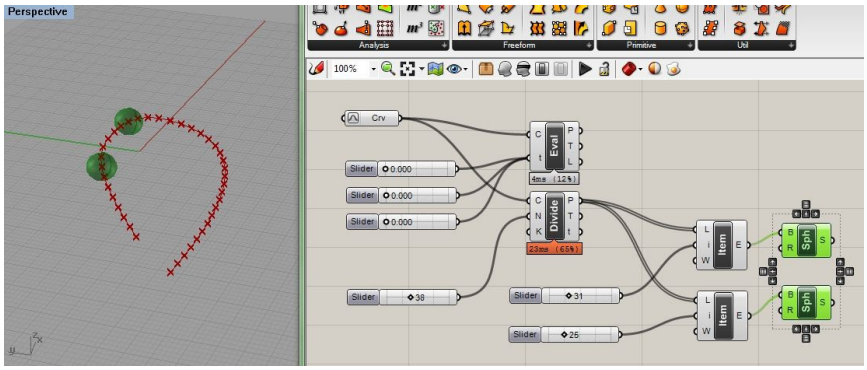


Proviamo ora a fare lo stesso procedimento per operare con 2 punti sulla curva, e giusto come esempio a connetterli attraverso una linea. Operando con due punti prenderemo 2 component del tipo *List Item,* avendo cura di connettere all'indice i degli slider del tipo *integer,* e di connettere le uscite con un component *line.*

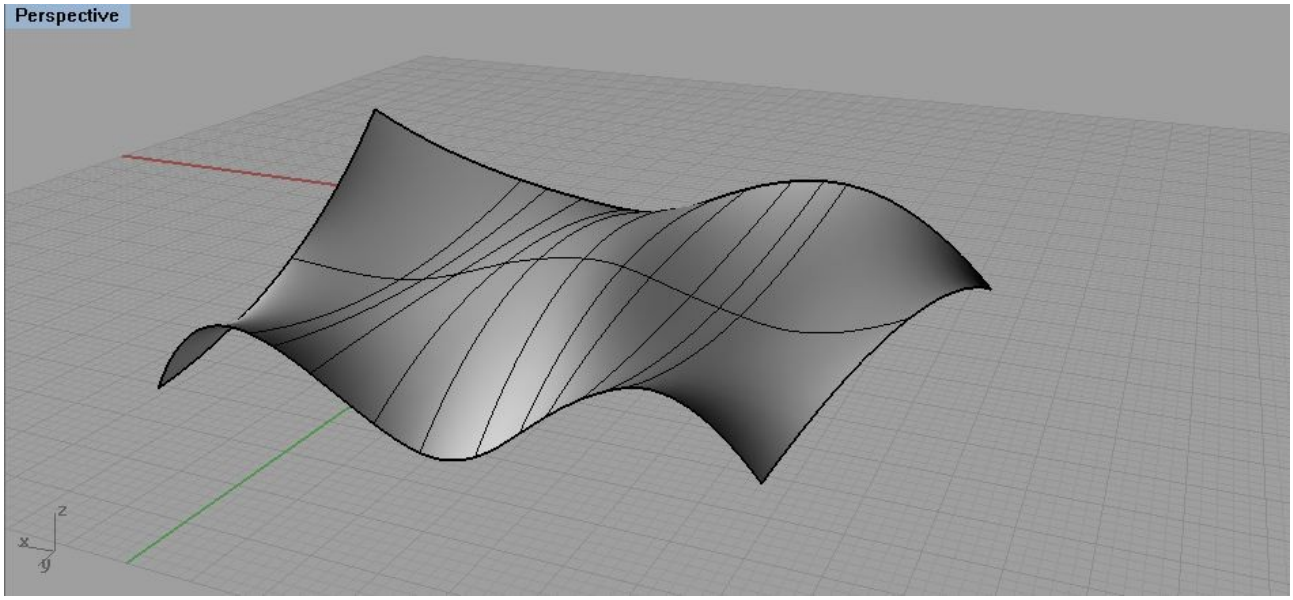
Non è detto che dobbiamo per forza utilizzare il component line, possiamo provarne altri, o ancora se usiamo *line* possiamo procedere associando altri elementi. Proviamo a far scorrere gli slider, per osservare come cambiano gli estremi della linea lungo la curva stessa.



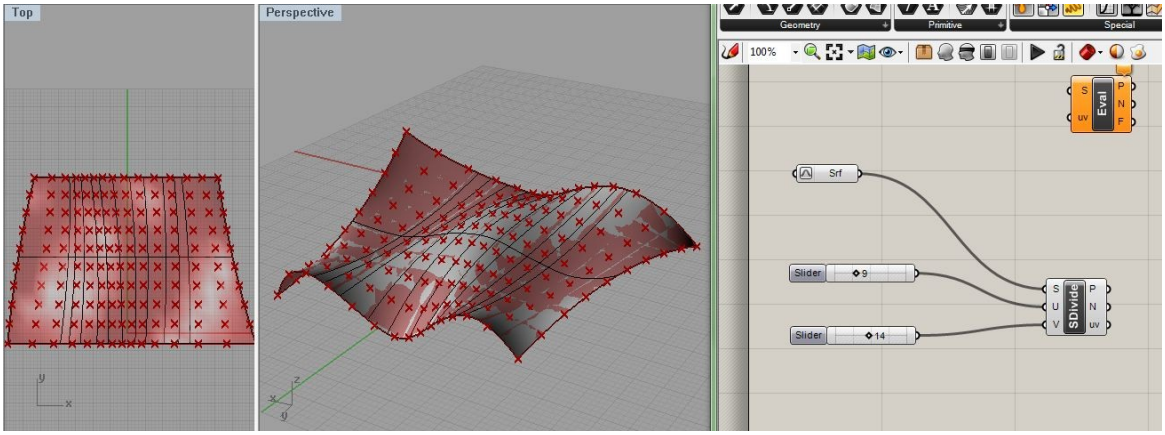
Ovviamente se associo alle uscite dei component *sphere* (non dal panel mesh, ma da quello surface), posso posizionare degli oggetti sulla curva in maniera piuttosto precisa, e muovendomi con gli slider posso collocarli dove voglio sulla curva:



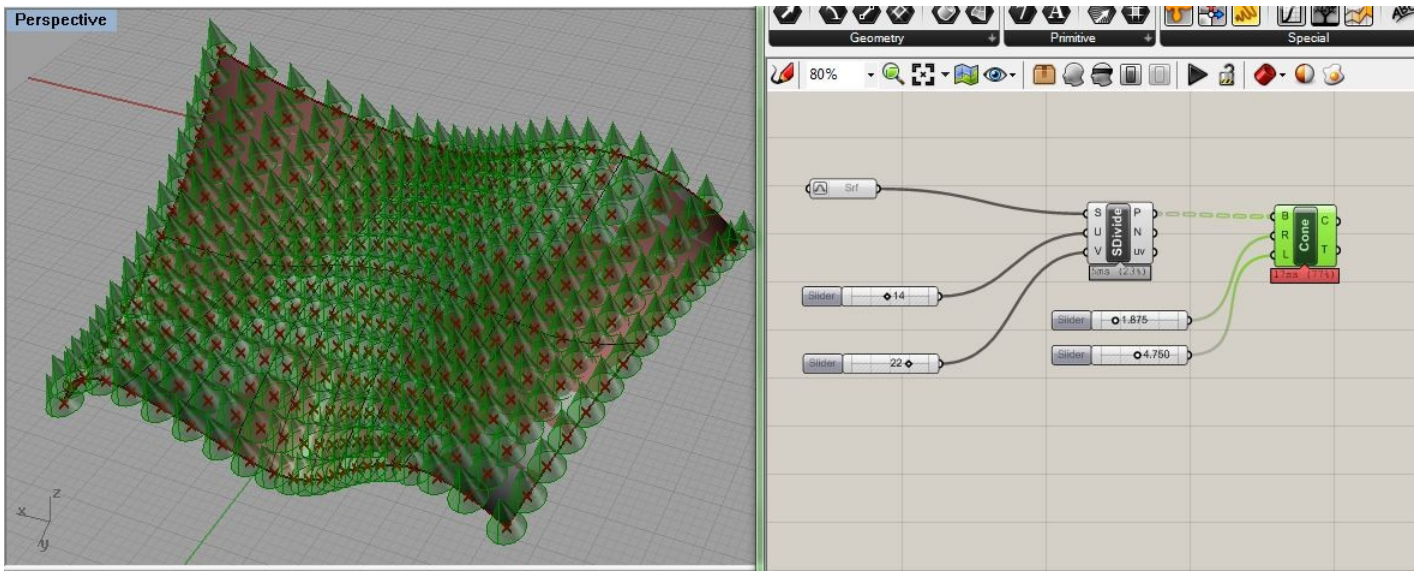
Cerchiamo ora di estendere queste semplicissimi considerazioni al caso della superficie che possiamo sempre vedere come un caso più generale della curva, dove invece di un unico parametro *t,* ne abbiamo due, solitamente indicati con *u* e *v.*
Immaginiamo di avere una superficie sulla scena, magari ottenuta con un loft, o come si vuole:

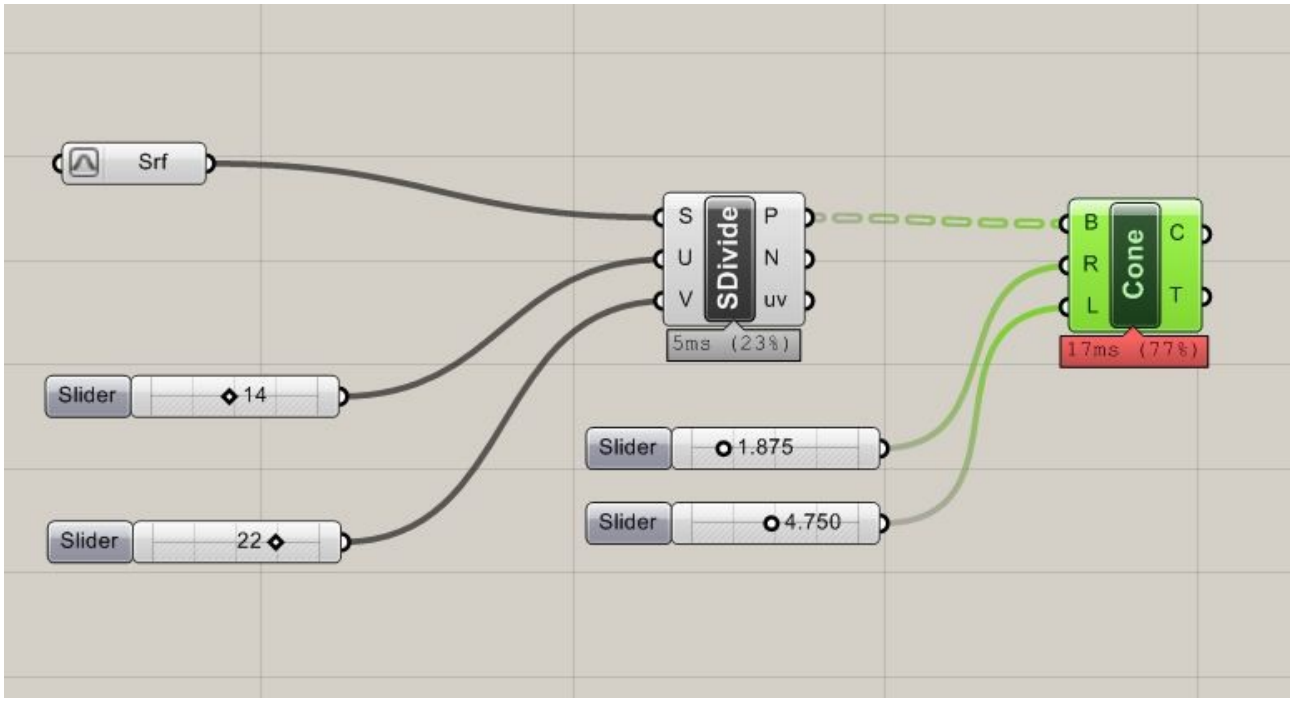


In Grasshopper portiamo il component *Surface* sul canvas, prendendolo dal tab *Params,* e aggiungiamo la superficie nella scena ricordando di ri-parametrizzare.
Fra i diversi component, che possono esserci utili (si trovano nel tab Surface, sotto *Analysis* e *Util*), possiamo ad esempio utilizzare il component *SDivide,* che opportunamente associato a 2 slider in ingresso (del tipo *Integer)* mi permette di suddividere la superficie in coordinate u e v:

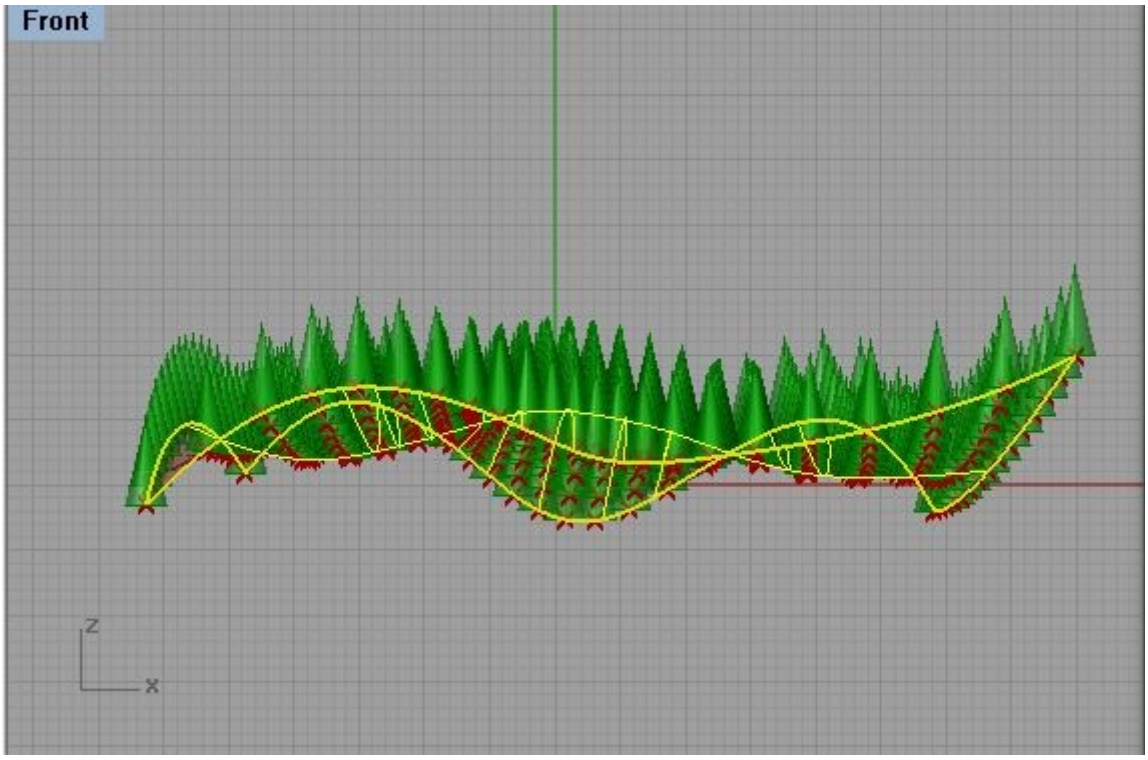


Da questo punto in poi è facile associare un component relativo ad un oggetto (o farsene uno proprio), e popolare la superficie. Nel caso abbiamo utilizzato il component *cone* e con due slider ne abbiamo regolato raggio e altezza:



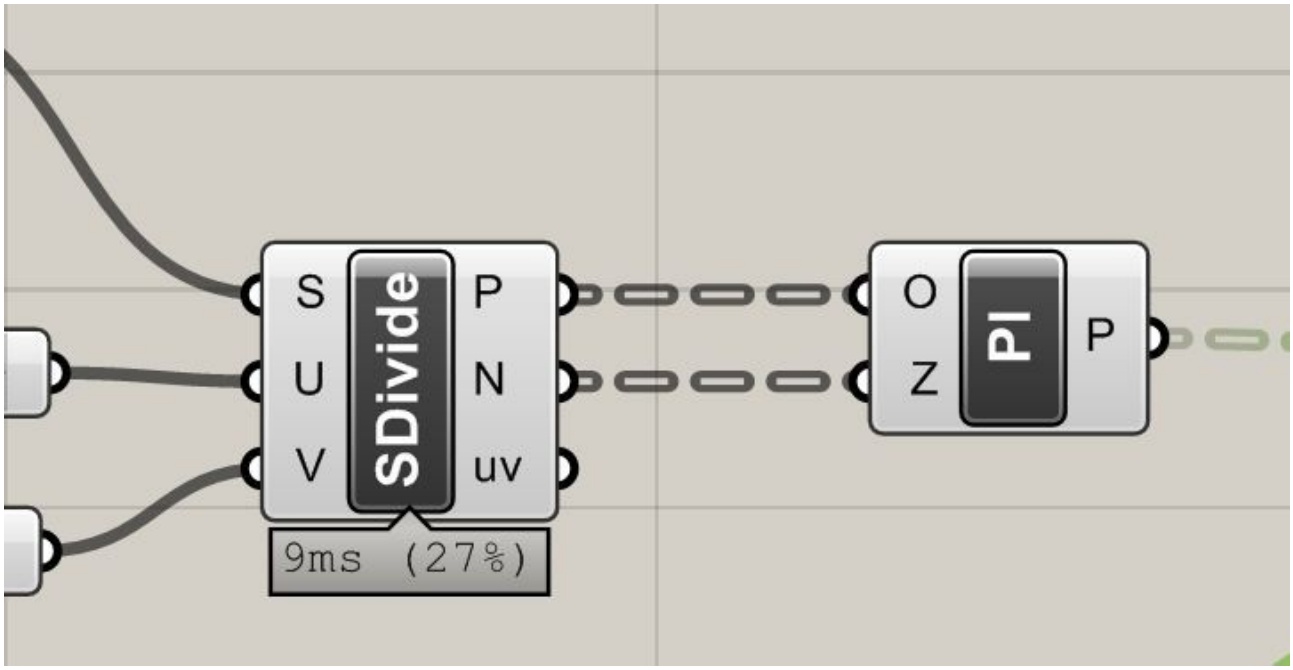


Ora però, proprio questo esempio così semplice ci permette di capire alcune cose:

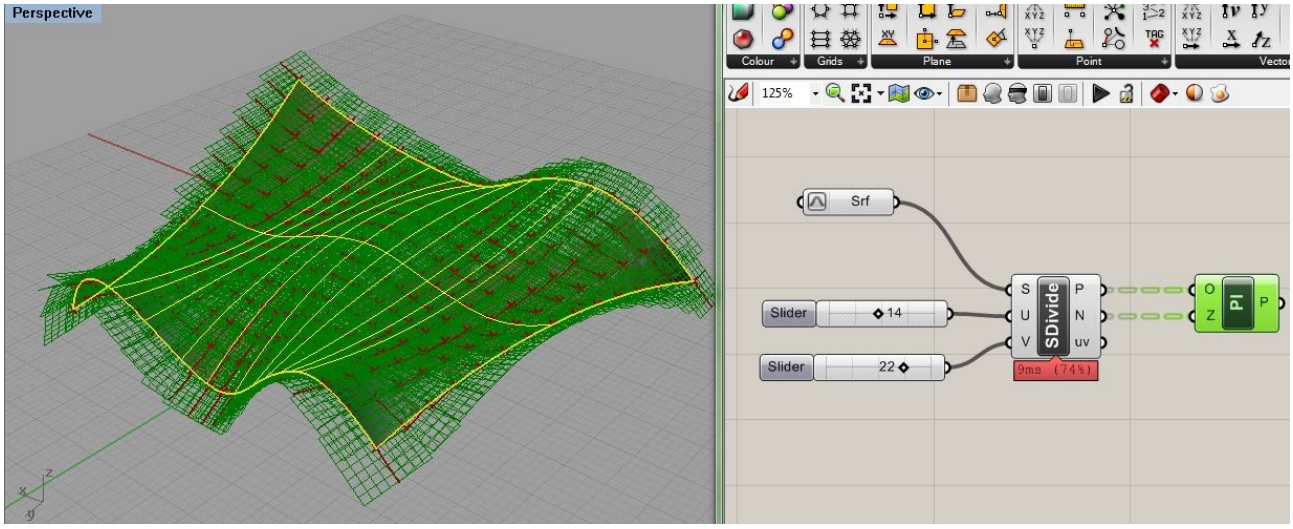


Come vediamo sebbene abbiamo posizionato i coni, o altri oggetti, lungo la superficie nelle direzioni *u* e *v,* e il centro della base coincide con questi punti, nulla accade per la direzione dei coni: in altre parole tale direzione è orientata secondo l'asse z, senza nessuna relazione con la superficie stessa. Sorge quindi spontaneo provare a vedere se possiamo orientare i coni secondo la direzione perpendicolare al piano tangente alla superficie nel punto, in modo tale che i coni possano seguire la curvatura.

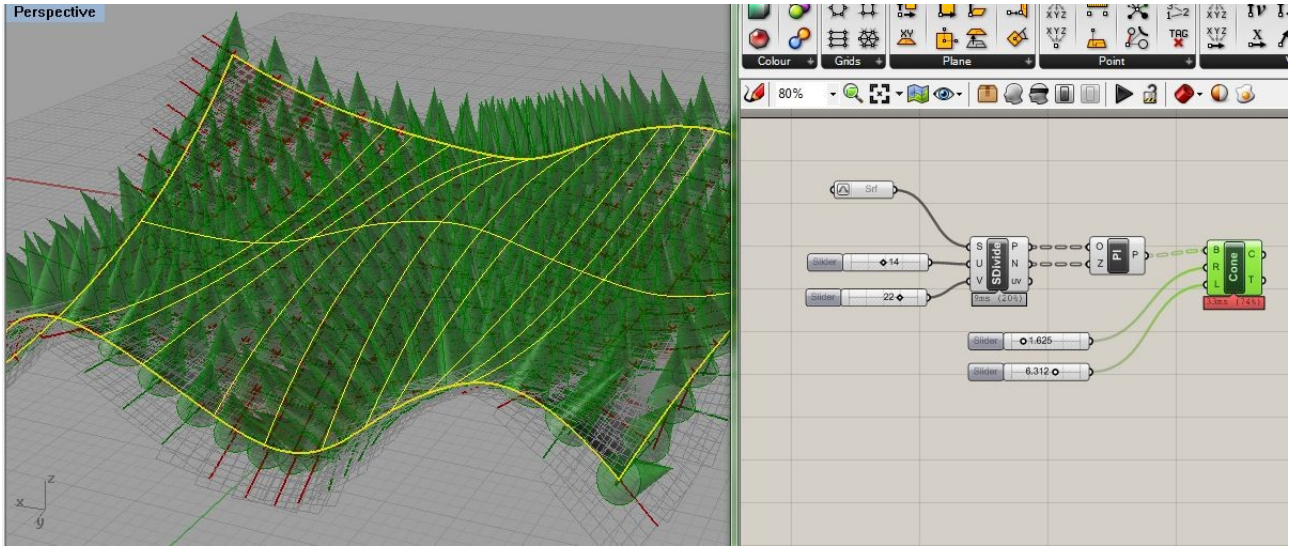
Se noto che il component *SDivide* mi fornisce già in uscita il valore N, che mi fornisce i vettori orientati secondo la Normale alla superficie nel punto. Allora posso considerare il piano alla normale di questi vettori e utilizzarlo per alloggiare i coni. Il component che mi permette di ottenere questi valori è *PlaneNormal,* che prendo dal tab Vector e che mi fornisce appunto un piano perpendicolare ad una direzione data.

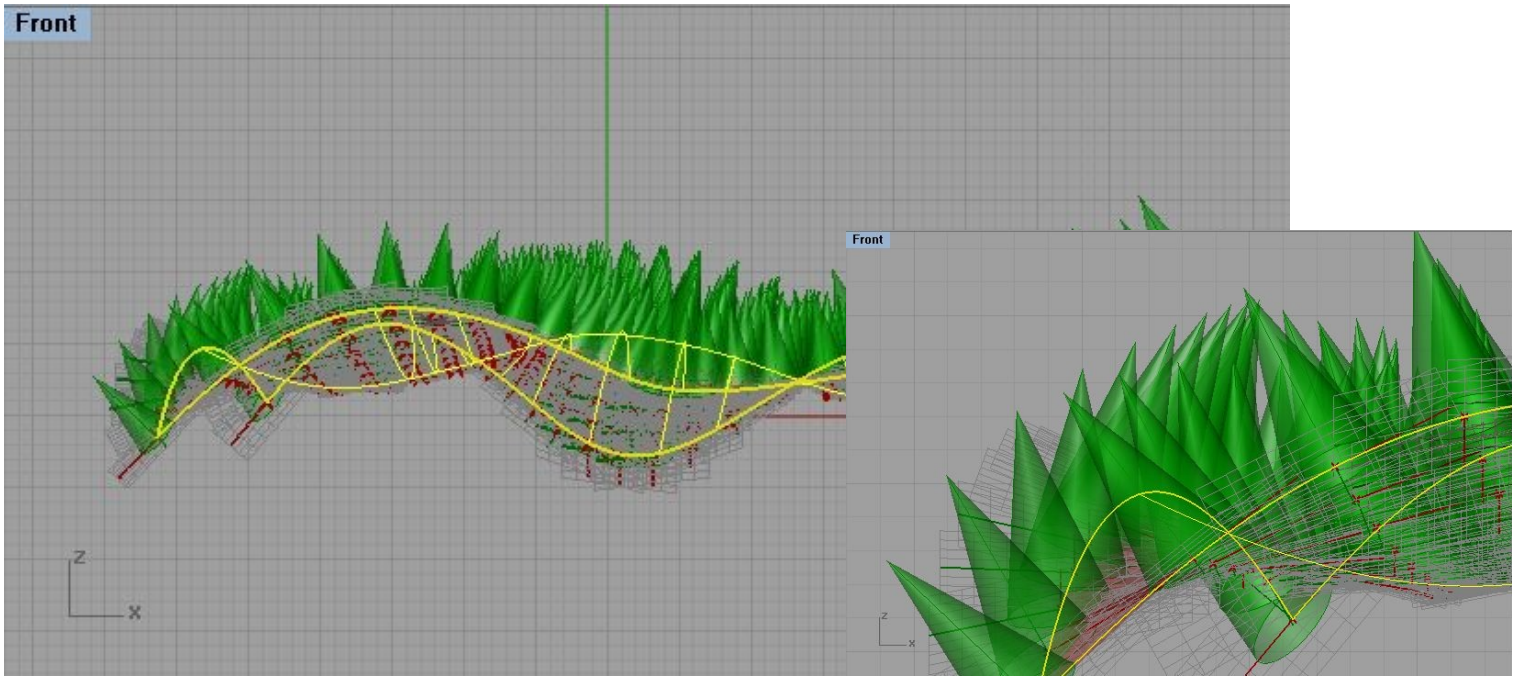


Ora connettiamo i valori P del component *SDivide* con quelli in entrata O del component *NormalPlane;* con questa operazione stiamo associando i punti di suddivisione della superficie con l'origine dei piani perpendicolari che vogliamo creare.
Poi connettiamo i valori di N, che come abbiamo detto rappresentano le direzioni perpendicolari alla superficie nei punti di suddivisione, con i valori in entrata Z; stiamo dicendo cioè che ogni asse Z dei piani creati deve corrispondere al vettore Z stabilito dal component *SDivide.* In questo modo abbiamo i piani cercati.



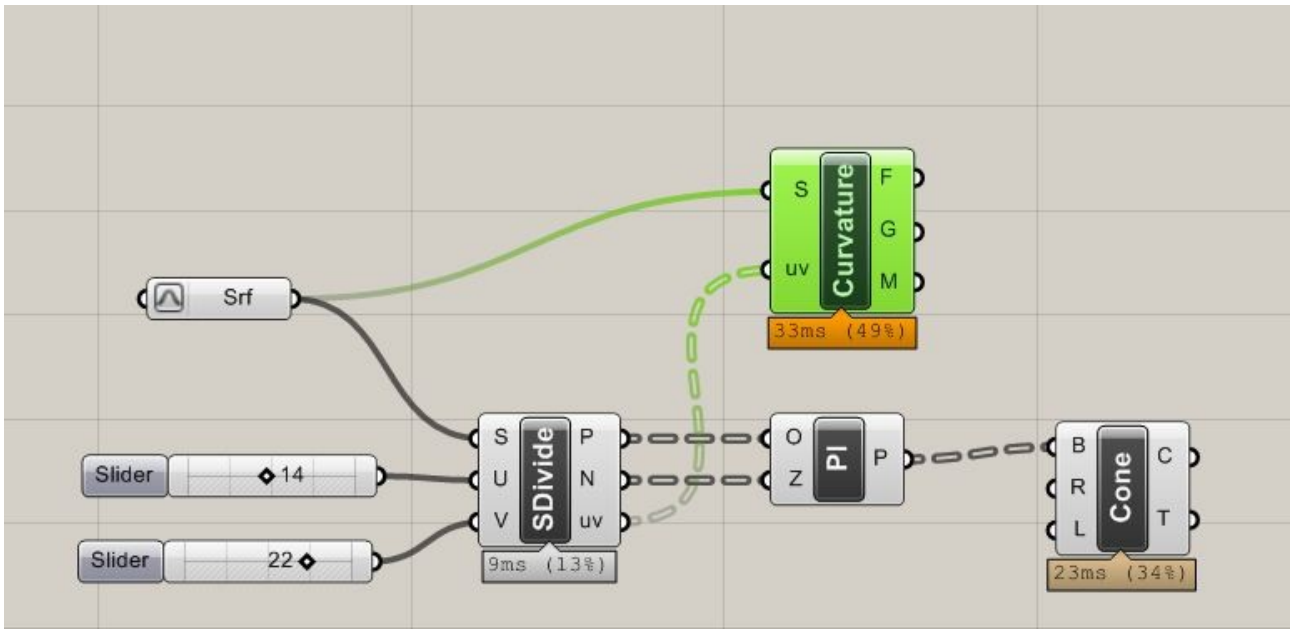
A questo punto se connettiamo i valori in uscita del component *normalPlane,* con quelli B relativi al piano di base sul quale il component *Cone,* alloggia i nostri coni, possiamo osservare che l'orientamento di questi segue la perpendicolare alla superficie in quel punto:



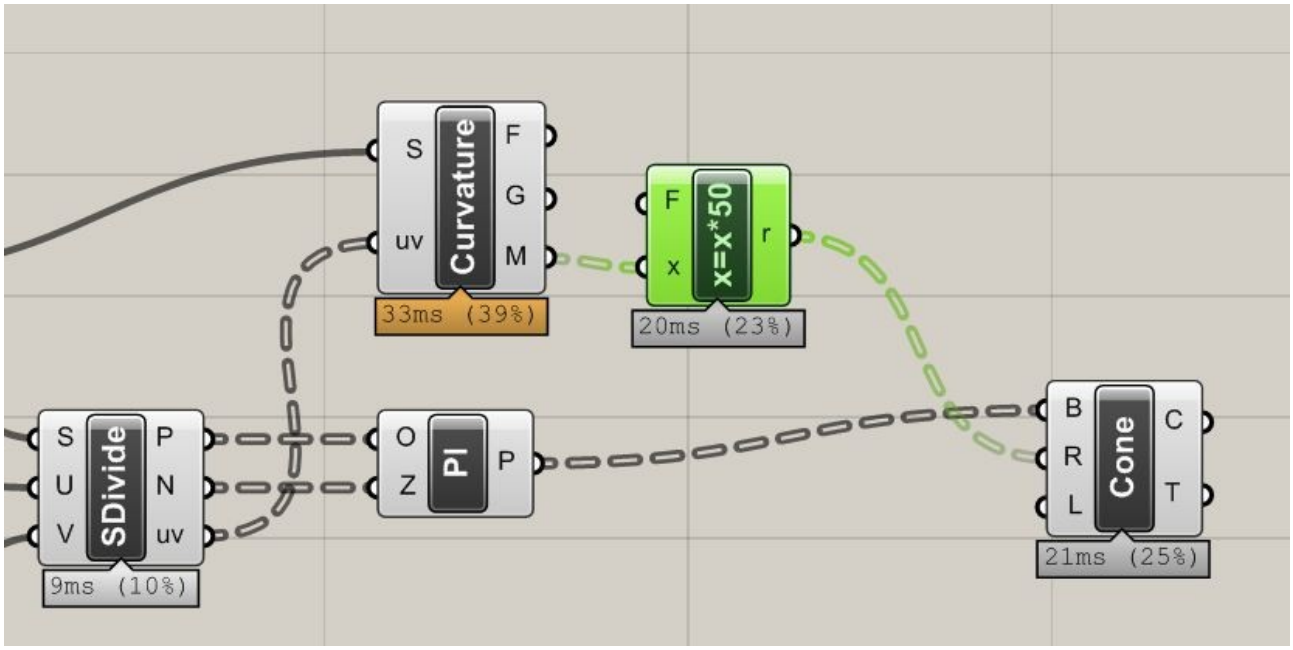


Ora però, se abbiamo creato una relazione piuttosto ovvia, ma importante da sottolineare, fra direzione dei coni e perpendicolari ai punti sulla superficie, nulla ci vieta di crearne altre, relative ad esempio alle basi dei coni.
Questi ragionamenti che facciamo sul caso semplicissimo dei coni, possono essere estesi con la stessa logica a degli elementi che creiamo noi e che in diverse maniere e finalità possono popolare una superficie. Tornando ai coni, osserviamo che finora abbiamo controllato la dimensione in altezza e quella del raggio con dei semplici slider. Nulla ci vieta anche qui di istituire delle relazioni con la superficie stessa, magari con la curvatura in un punto.

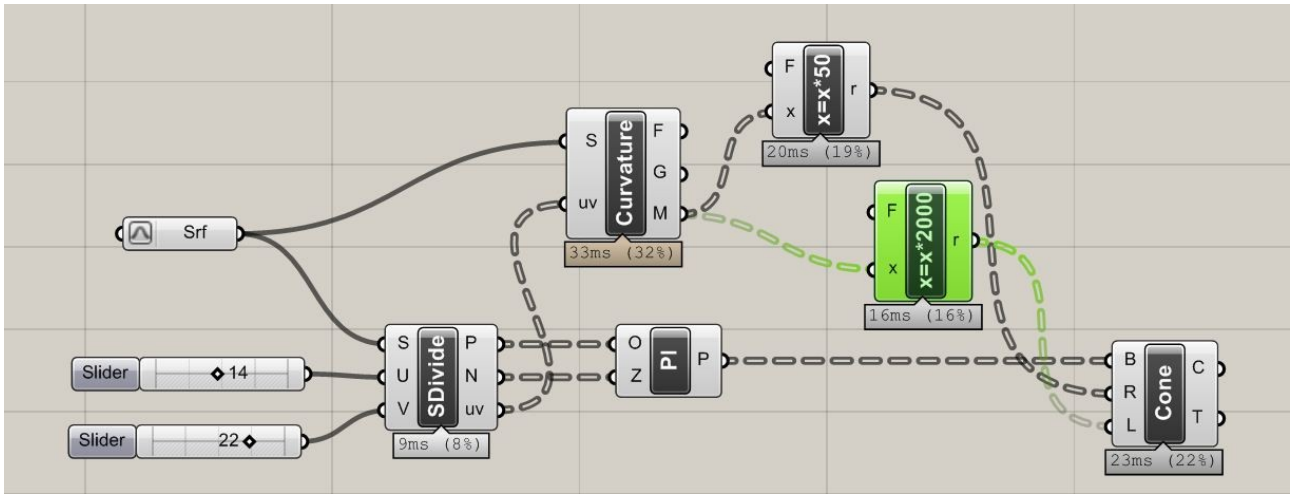
Se prendo il component *Surface Curvature,* dal tab Analysisi delle superfici, posso connettendolo alla Superficie, e alle coordinate *u* e *v,* ottenere la curvatura media in un punto. Questo valore è dato da M;



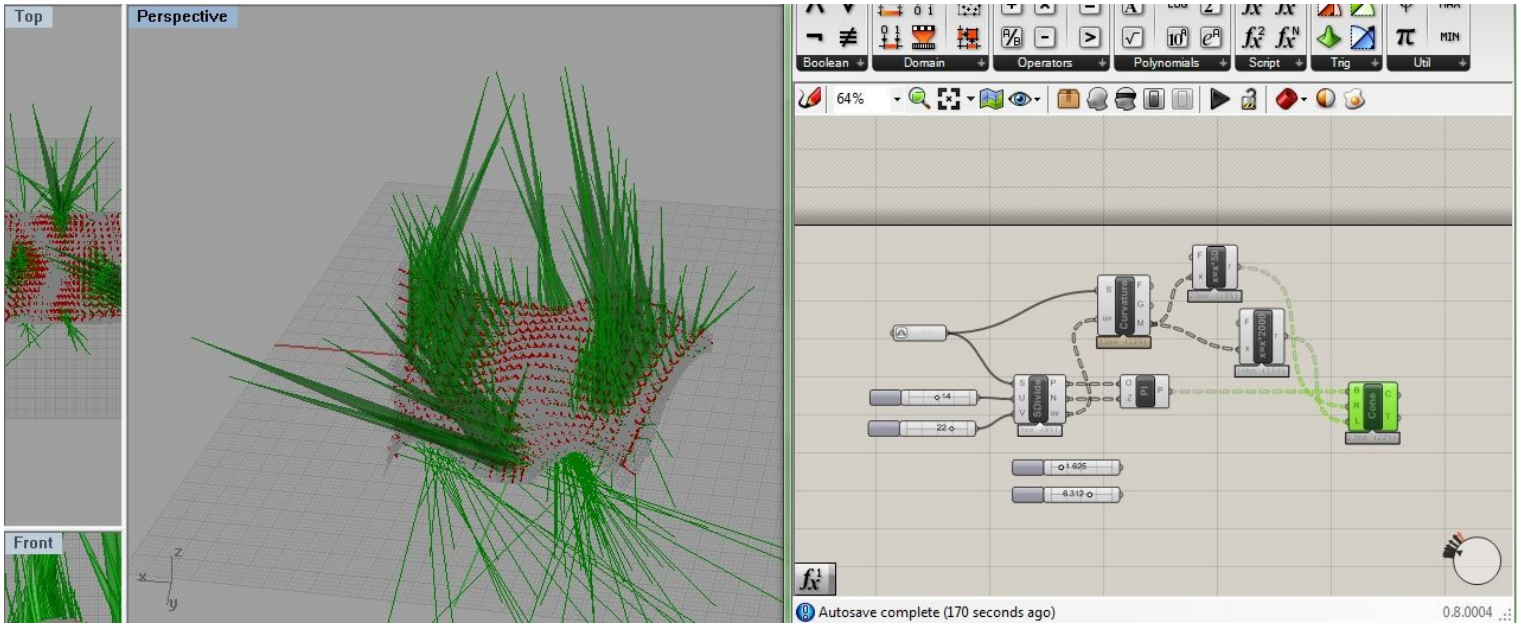
Ora però siccome si tratta di valori molto piccoli, moltiplichiamoli per un conveniente numero, ad esempio 50, con una semplice funzione del tipo x = x * 50; questa la inserisco attraverso il component *F(x)* dal tab Script. E lo associo a M. I valori in uscita invece li connetto al raggio *R,* del component *Cone.*

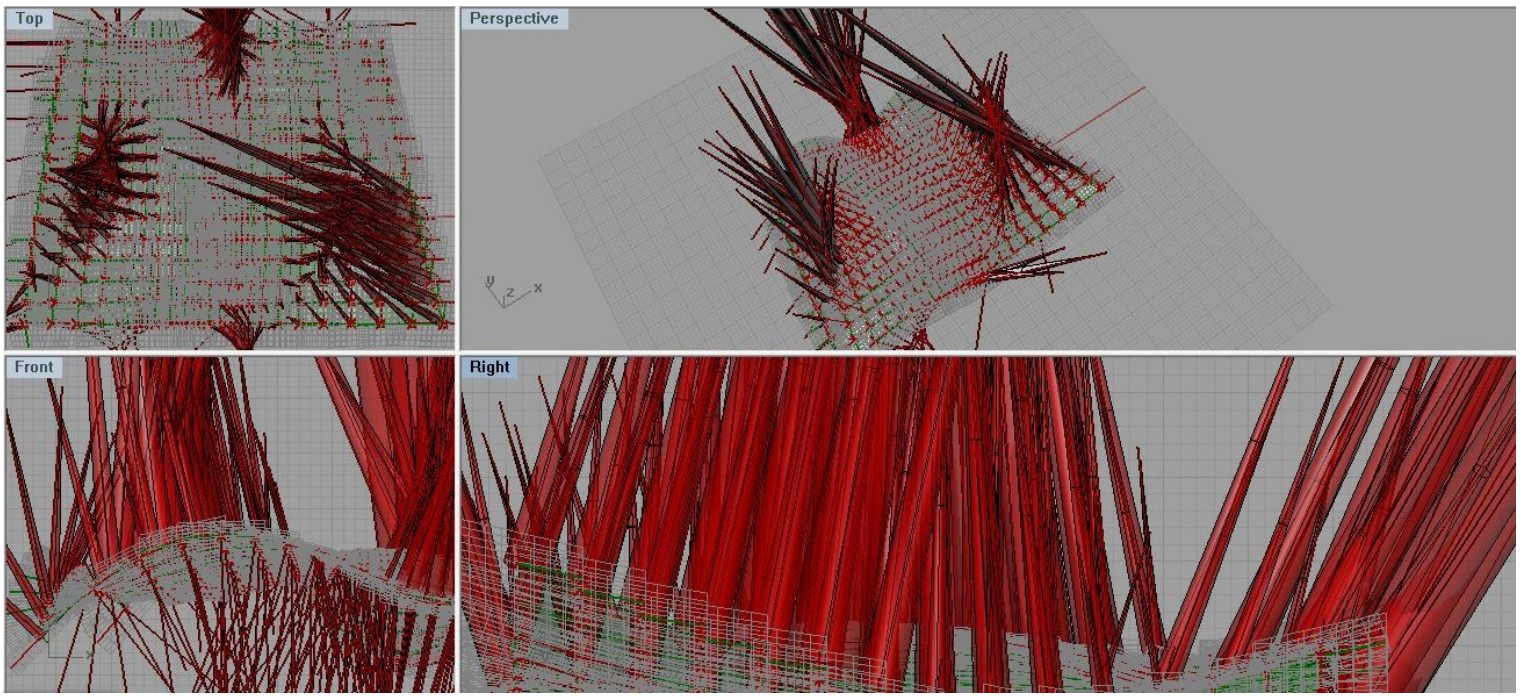


Questo vuol dire che nei punti a curvatura maggiore il raggio sarà maggiore nelle dimensioni.
Voglio avere anche una relazione simile per l'altezza dei coni, e allora provo ad inserire un altro component *F(x),* con una relazione del tipo x = x* 2000. Connetto l'input con il valore M della curvatura e l'output con la L del component *Cone.*



Come vediamo l'aver moltiplicato per 2000, mi fornisce altezze dei coni elevate nei punti a maggiore curvatura.





LINK alla SECONDA PARTE >>>