3_control

curves and surfaces in Grasshopper

> "...I was looking for a way to express feeling in three dimensional objects".
>
> Frank Gehry

Prior to the advent of digital tools, formal investigation was limited to traditional drawing techniques relying on geometric primitives and basic mathematical models. Emerging tools and fabrication techniques have changed the way designers conceive of architecture, opening new formal trajectories. This paradigm-shift from pencils, squares and compasses to computer-based tools has happened over the last three decades.

The computer or more specifically, 3D modeling software has pushed the boundaries of formal exploration by digitizing manually iterated design. The next epochal transition – Algorithmic Modeling – requires a robust knowledge of the principles behind form generation; specifically an understanding of NURBS geometries.

3.1 NURBS curves

NURBS or ***N**on-**U**niform **R**ational **B**-**S**plines* are *mathematical representations of 3D geometry that can accurately describe any shape from a simple 2D line, circle, arc, or curve to the most complex 3D organic free-form surface or solid.*

A NURBS curve is a mathematical curve defined by its degree, a set of weighted control points, a knots vector and an evaluation rule.

- **Degree**: a positive, integer number. For example, NURBS-lines and NURBS-polylines are degree 1, NURBS-circles are degree 2, and most free-form curves are degree 3 or 5. The value equal to degree+1 is called the ***order***.
- **Control-points**: A NURBS curve is conditioned by the position and ***weight*** of control points Weights regulate the attraction between the control-points and the curve; a weight >1 attracts the curve and a weight between 0 and 1 repels the curve. The number of control points cannot be smaller than order (degree+1).
- **Knots:** are a list of (degree+N-1) numbers, that control the smoothness of a curve.
- **Evaluation rule**: is a formula that inputs: degree, control points and knots and outputs a point location.

NURBS curves can be drawn by positioning control-points in three dimensional space or on a 2 dimensional plane. The image below is a planar NURBS-curve with a degree of 3 controlled by the position of the 7 control-points.

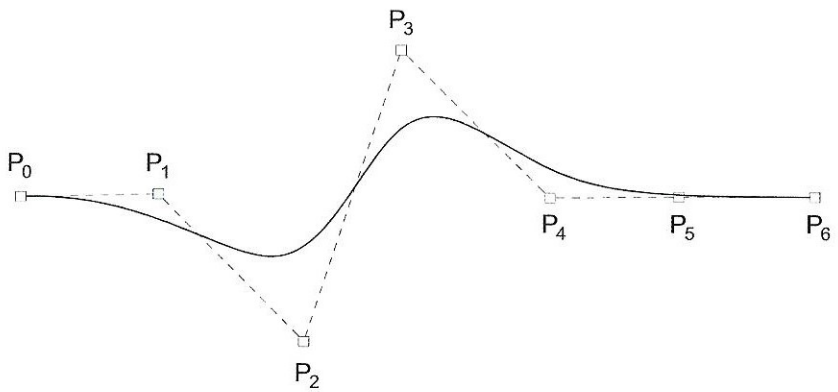


FIGURE 3.1
A NURBS curve with 7 control-points.

If the curve's degree changes different curves will be generated. A curve with the degree 1 coincides with the control-polygon.

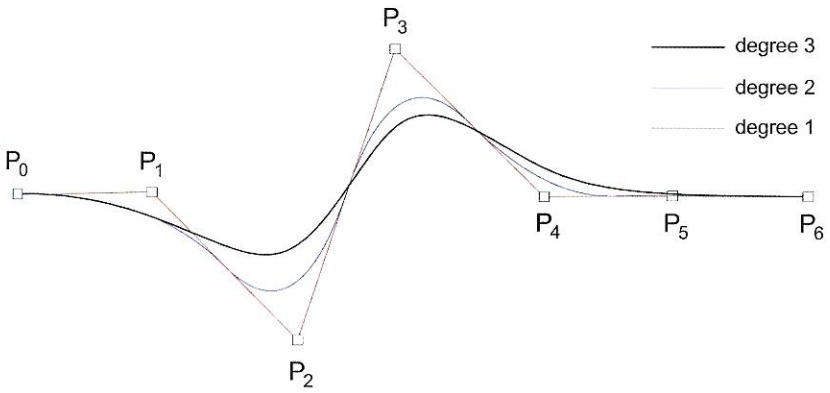


FIGURE 3.2

NURBS curves with 7 control points and different degree.

The image below shows how the *weight* of control-points affects a NURBS curve.

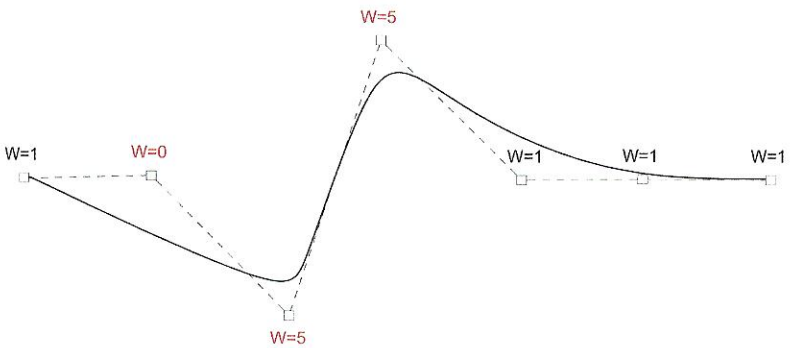


FIGURE 3.3

A control-point with a weight > 1 attracts the curve, on the contrary a control-point with a weight between 0 and 1 push off the curve.

3.2 Parametric representation of a curve

The Rhino environment is based on a **World Coordinate System (WCS)** and points on a NURBS curve are defined by a triplet of coordinates (x_i, y_i, z_i). If z=0 for each point, the curve is a *planar curve* and is defined by the coordinates (x_i, y_i).

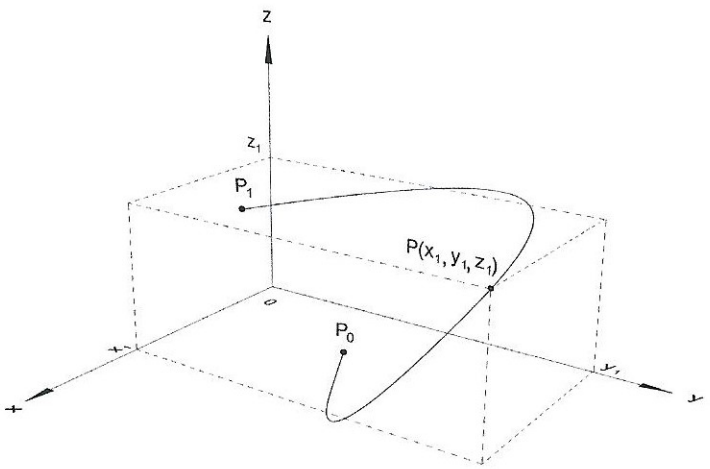
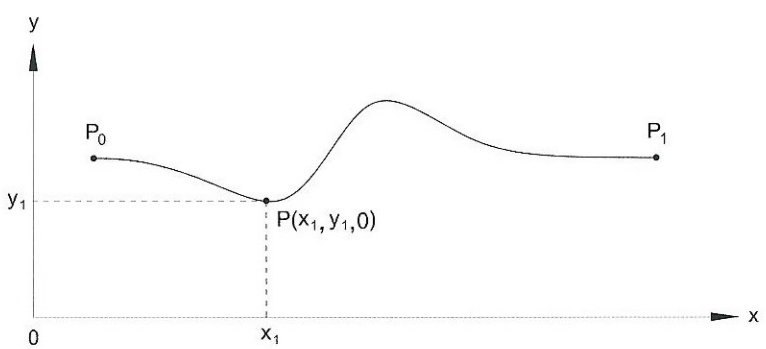
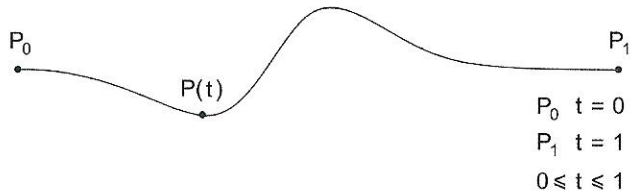


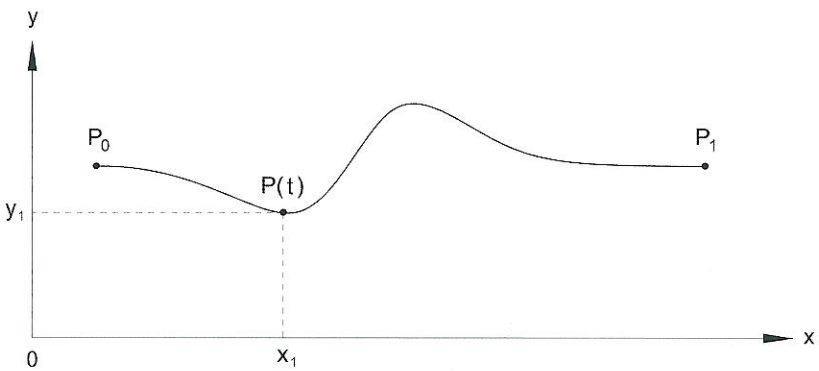
FIGURE 3.4
Each point of a NURBS curve is defined by a triplet of coordinates.



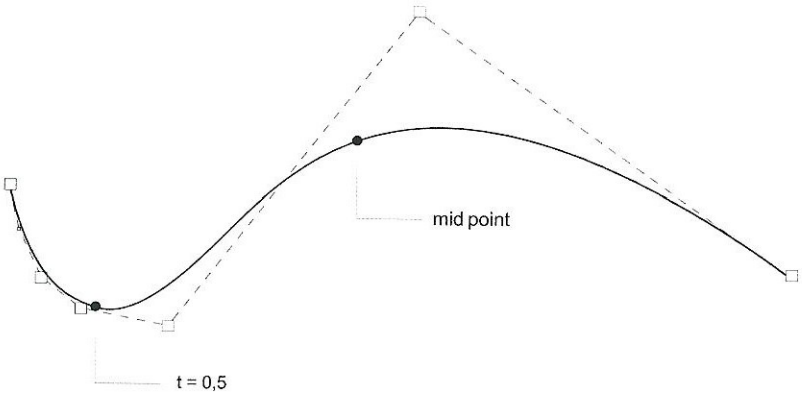
Another way to find points on a curve is based on the **Parametric representation**. The coordinates of an arbitrary point P are expressed as function of a variable ***t*** that ranges between 0 and 1. For t=0 and t=1 we get the curve's end points and for *t* ranging between 0 and 1 we describe the whole curve.

We can also say that a curve is *parameterized* between 0 and 1 or, in alternative, that its **domain** is [0,1]. The parametric representation can be considered as a **Local Coordinate System (LCS)**. Roughly speaking, it is a system set "on" the curve which have great advantages since it requires just one parameter to identify a point, which is still defined in the World Coordinate System (see images below).

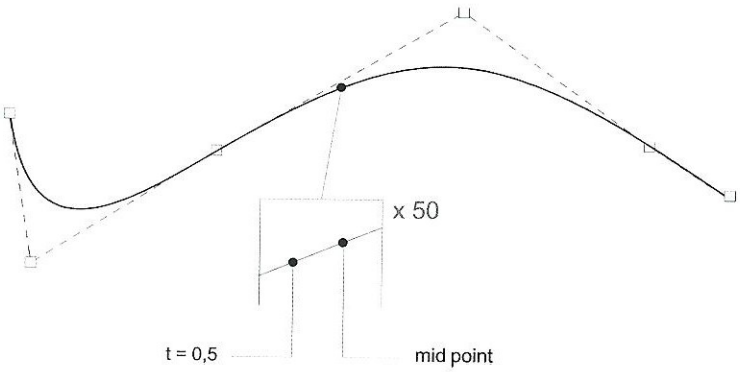




It's important to point out that *t* doesn't measure distances. We can imagine *t* as the *time* that a "particle" takes to go from t=0 to the instant position P(t). This time is affected by the position of control-points and, in particular, the motion of the "particle" slows down when it passes through a concentration of control points (see following image). For this reason t=0,5 is not the parameter that specifies the curve's mid point.



Even if the curve is *rebuilt* using Rhino, resulting in a uniform redistribution of control points, the mid point is not coincident with the point corresponding to t=0.5.

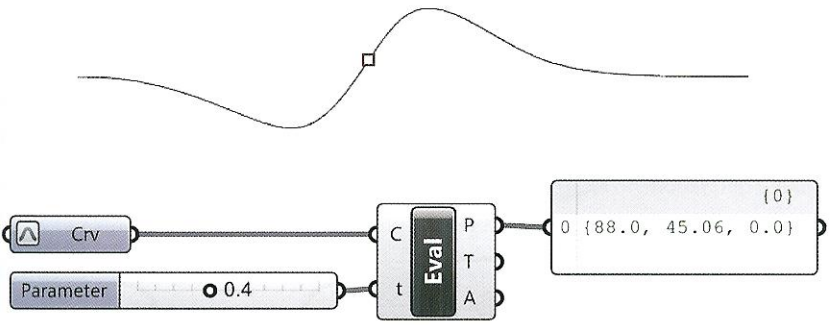


3.3 Analysis of curves in Grasshopper

The following paragraphs will introduce components, logics and strategies to control NURBS curves using *Parametric Representation.*

3.3.1 Finding points on a curve: *Evaluate Curve* component

Parametric Representation enables users to find points along a curve. The component *Evaluate Curve* (Curve > Analysis) requires a curve (C) to analyze and a parameter (t) on the curve domain to evaluate, which can be defined by a *Number Slider*.



The *Parametric Representation* assumes that a curve is parameterized between 0 and 1. However, curves in Rhino typically have different domains. The domain of a curve can be set to [0,1] by reparameterizing the curve. The ***Reparameterize*** option is available by right-clicking on the curve component and selecting *Reparameterize* from the contextual menu. If multiple curves are set with different domains, their local domain will be set to [0,1] by selecting the *Reparameterize* option.

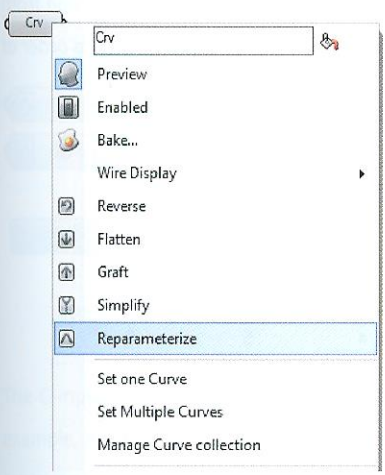
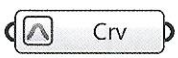
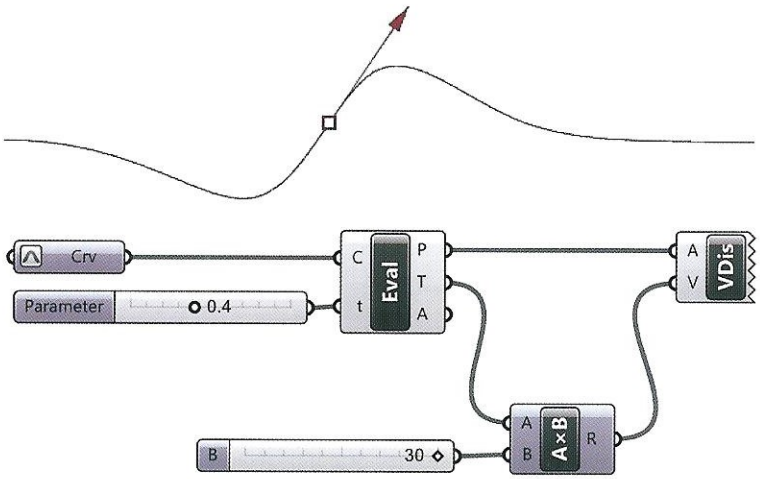


FIGURE 3.5
The Reparameterize option is available within the context menu of the Curve component.

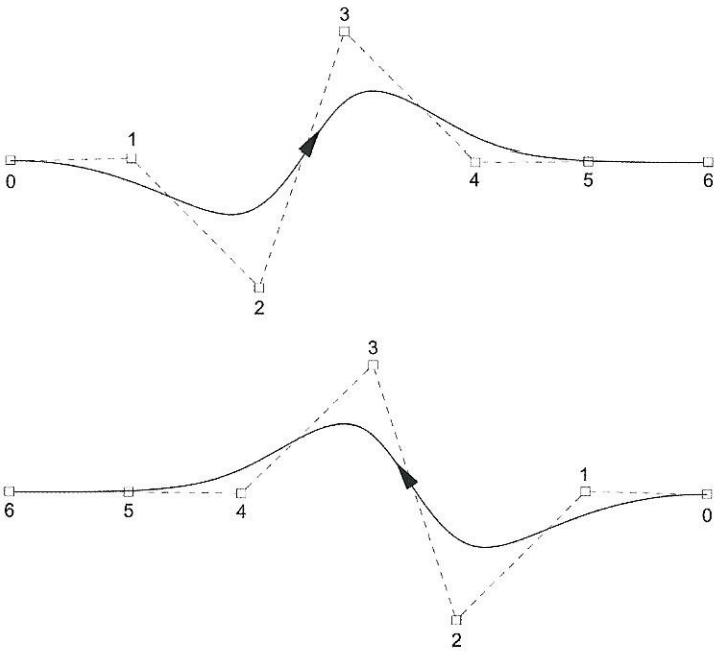


The *Evaluate Curve* output (P) expresses the points location in the World Coordinate System, the T-output is the tangent vector at *t*. The *Vector Display* component can be used to visualize the tangent vector (T). Tangent Vector (T) is a unit vector that can be amplified by scalar multiplication using the *Multiplication* component.

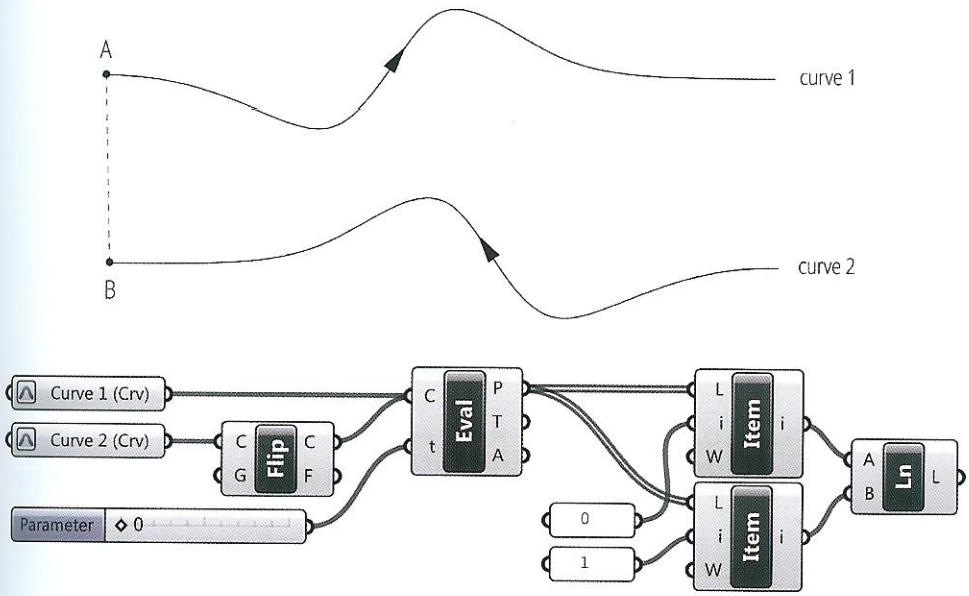


3.3.2 Inverting direction: *Flip Curve* component

Parameterized curves are described by a start point (t=0), an end point (t=1) and a *direction* from start point to the end point. The *direction* depends on how the curve was drawn, i.e. on the order of control points that define the curve.

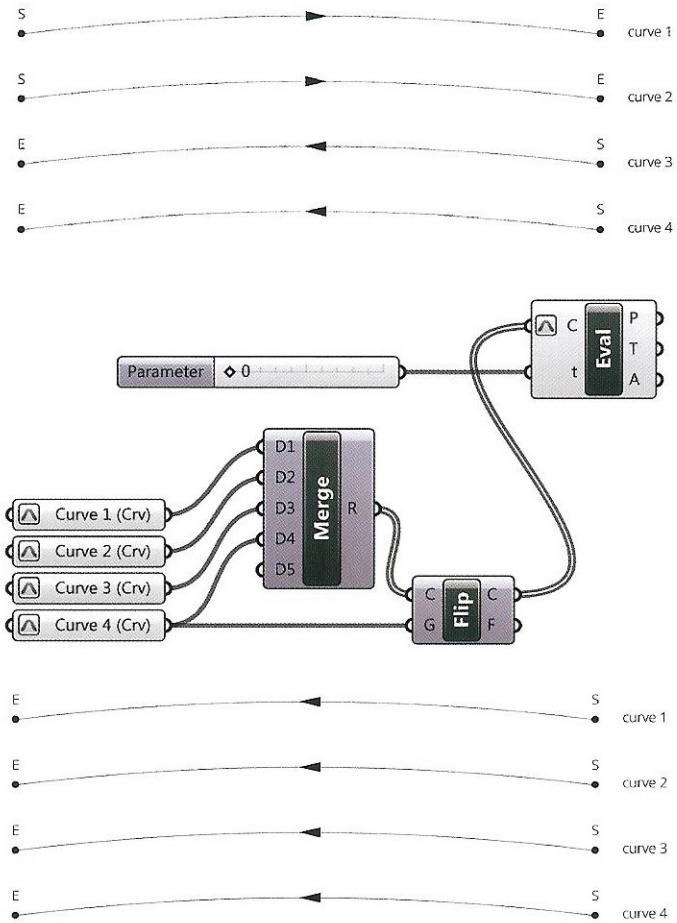


Reparameterizing a curve does not change the direction. The direction of a curve can be changed using the component *Flip Curve* (Curve > Util). The following image displays 2 curves with opposite directions: Curve 1 has left-to-right direction while Curve 2 has right-to-left direction. A is the start-point of Curve 1 while B is the end-point of Curve 2. Our aim is to connect the points A and B by a line. If we use an *Evaluate Curve* component to extract points, we must previously flip the direction of either Curve 1 or Curve 2 by *Flip Curve*. In this way, the point B will become the start-point of Curve 2 and we will be able to create the line.



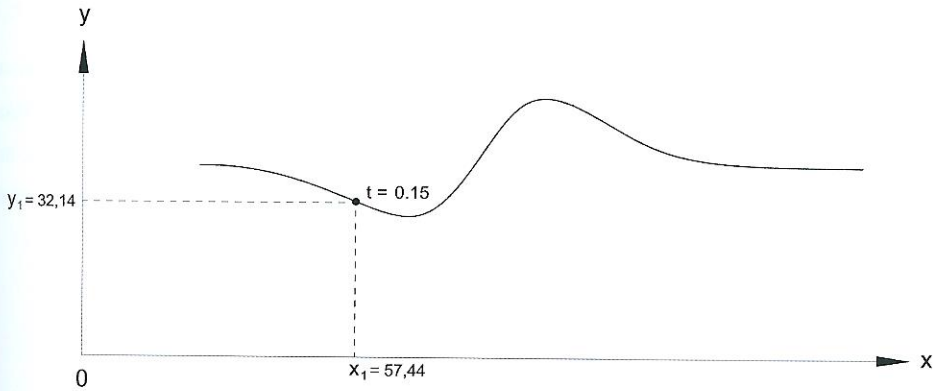
The G-input of *Flip* specifies a reference curve used to unify the direction of set of curves. For example, a set of curves with different directions – curves 1 and 2 are left-right oriented and the remaining are right-left oriented – are set and merged (see following image).

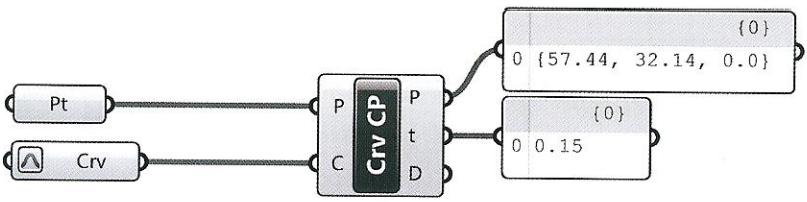
The merged list is connected to the C-input of a *Flip* component and Curve 4 is set as the guide curve in input (G), resulting in a set of right-left oriented curves. The *Evaluate Curve* component can be used to specify points along the curves (it is important to reparameterize the C-input).



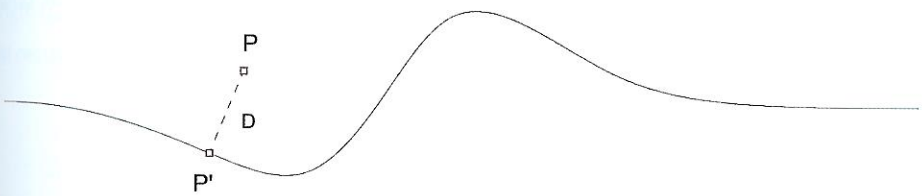
3.3.3 Finding points on a curve: *World to Local* conversion

By default, set points from Rhino are expressed in the World Coordinate System. The component *Curve CP* (Curve > Analysis) is used to convert a point expressed in the WCS to the *t* local parameter of a given curve. The *t*-output of *Curve CP* is the parameter on the curve's domain expressed in the Local Coordinate System.



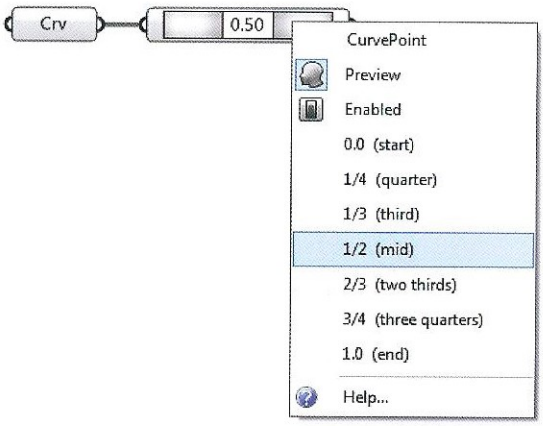


The component *Curve CP* enables users to find the closest point P' on a curve, given an external point P. The displacement distance between points P and P' is provided by the D-output. When *Curve CP* is used to convert from WCS to LCS: P=P' and D=0.



3.3.4 Finding points on a curve: *Point on Curve* component

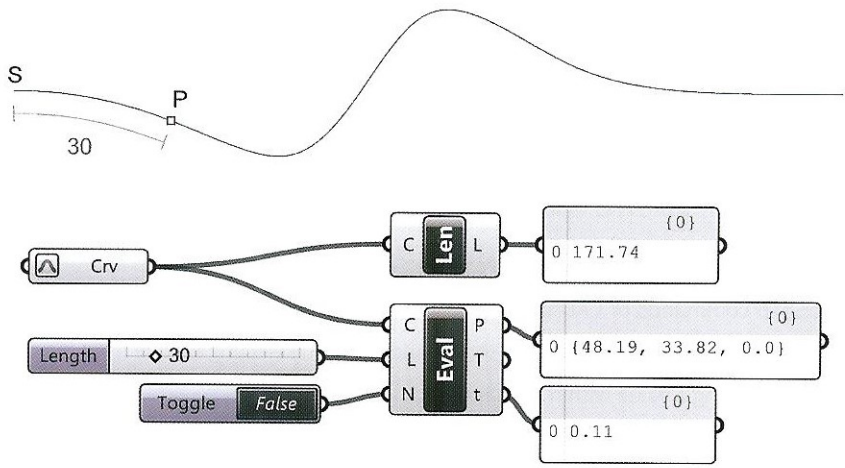
As discussed in chapter 1 the component *Point On Curve* (Curve > Analysis) can operate similarly to the Object Snap commands of CAD packages. The component can be used to find points such as start, quarter, mid etc., by right-clicking on the component and selecting a value from the contextual menu, or by specifying a value within the slider.



The component *Point On Curve* is based on the curve's length not on the curve's domain. As follows, a value of 0.5 expressed by the components slider corresponds to the mid point of a curve, unlike the *Evaluate Curve* component. The *Point On Curve* component does not require the input curve to be reparameterized.

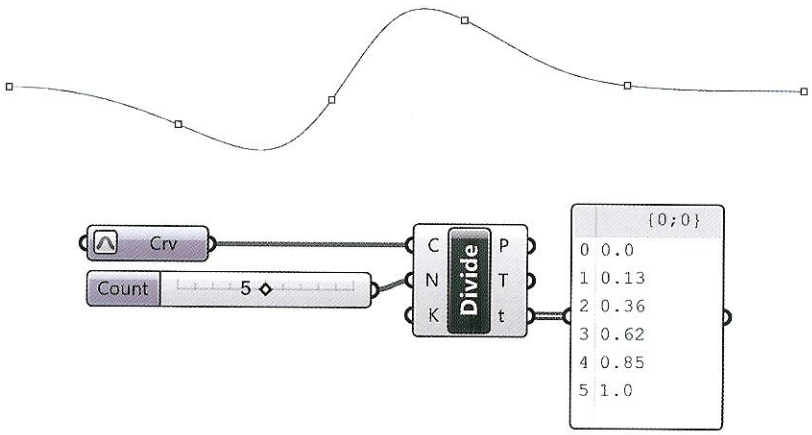
3.3.5 Finding points on a curve: *Evaluate Lenght* component

The component *Evaluate Length* (Curve > Analysis), finds a point (P) a distance measured as an arc-length (L) from the start point t=0. The input (L) is a number between 0 and the curve's length; which can be calculated using the component *Curve Length* (Curve > Analysis). The *Evaluate Length* component outputs coordinates in the WCS (P-output) as well as in the LCS (t-output).



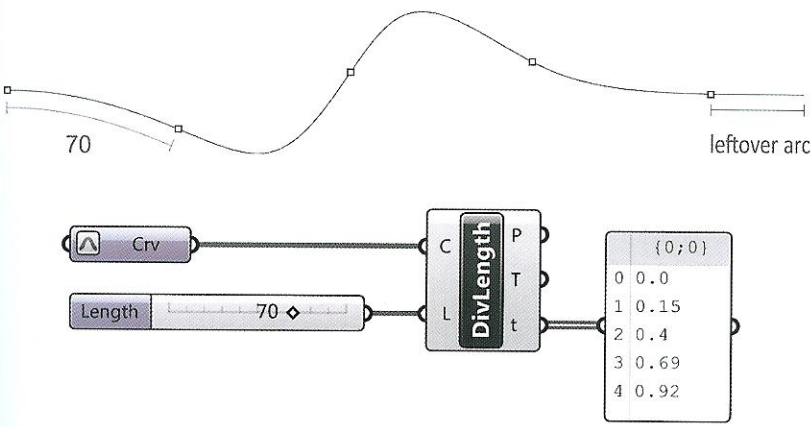
3.3.6 Dividing a curve: *Divide Curve* component

As discussed in chapter 1 (1.7.3), the component *Divide Curve* (Curve > Division) generates a set of points (P) by dividing a curve into (N) equal length arcs. As a result, N+1 points are generated in the case of open curves, and N in the case of closed curves. The component calculates the tangent vectors (T) at division points as well as the LCS (t) parameters.



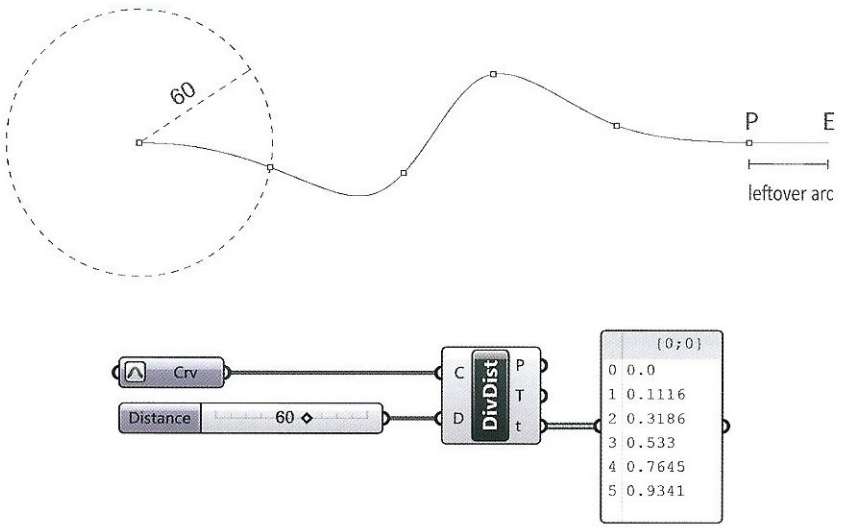
3.3.7 Dividing a curve: *Divide Length* component

The component *Divide Length* (Curve > Division) divides a curve into segments specified by the arc-length (L). If the curve's length is not a multiple of (L), a "leftover-arc" with a length different from (L) will result.



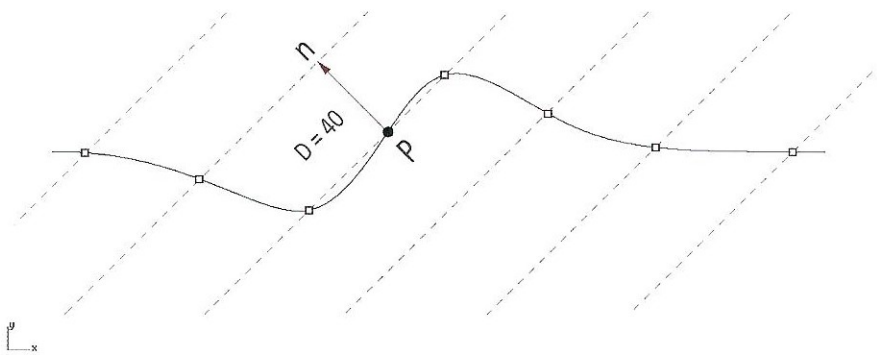
3.3.8 Dividing a curve: *Divide Distance* component

The component *Divide Distance* (Curve > Division), divides a curve into segments by calculating sequential intersections of circles and the curve. Depending on the value of the radial distance (D) a "leftover-arc" will result having a distance PE different from (D).



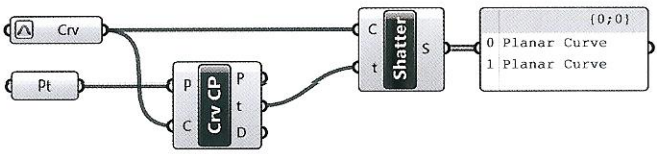
3.3.9 Dividing a curve: *Contour* component

The component *Contour* (Curve > Division), creates a set of curve contours given a curve (C), a start point (P), a vector (N) and a distance (D). The resulting intersections are output as point coordinates in the WCS (P) as well as in the LCS (t).

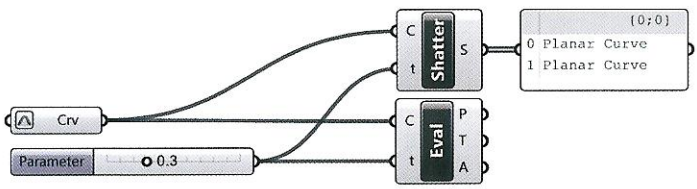


3.3.10 Splitting a curve: *Shatter* component

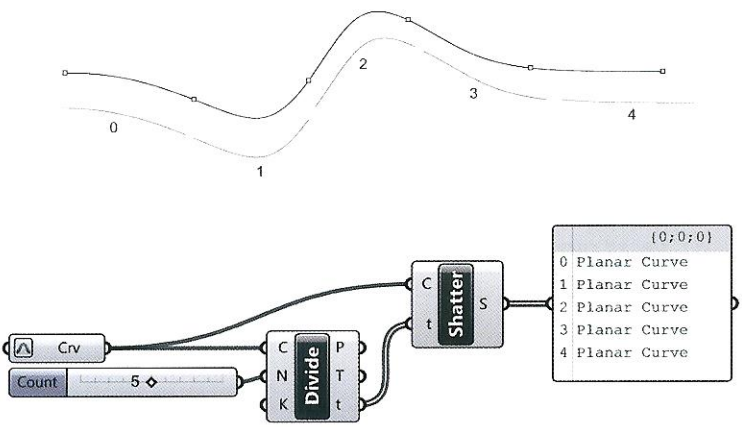
The component *Shatter* (Curve > Division) splits a curve into segments at (t) input parameters outputting segments, instead of points. The parameters to split at (t) can be set by specifying a value within the LCS domain [0,1] or supplied by the output of other components. For example, a curve set from Rhino can be split into two parts using a splitting point (set from Rhino) by connecting the *t*-output of the Curve CP to the *t*-input of the *Shatter* component, resulting in two curves as observed in a panel.



The *t* parameter can also be satisfied by using a slider to specify a value within the numeric domain [0,1].



Similarly, the *t*-output of the *Divide Curve* component can be used as a splitting parameter.



3.4 Notion of Curvature for planar curves

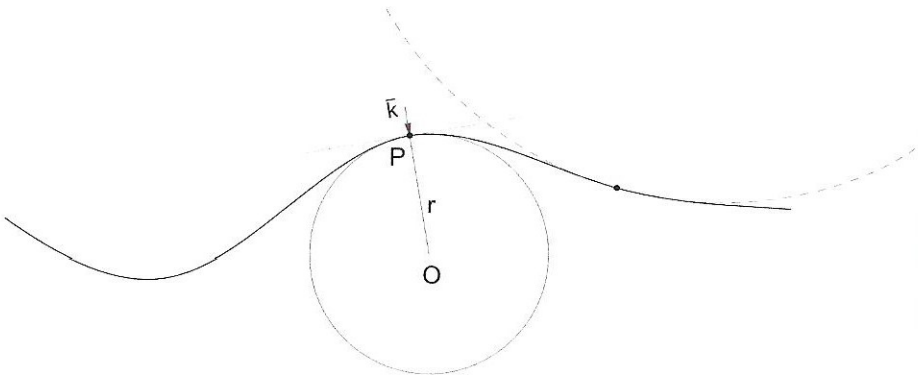
The curvature of a planar curve *c* measured at a point P calculates the deviation of *c* from its tangent-line near P. The curvature of a generic curve varies from point to point. In order to mathematically define curvature there are two basic assumptions:

- The curvature of a straight line is zero at every point;
- The curvature of a circle is constant. In particular, curvature is equal to zero when the radius approaches infinity (i.e. a straight line is a circle with an infinite radius).

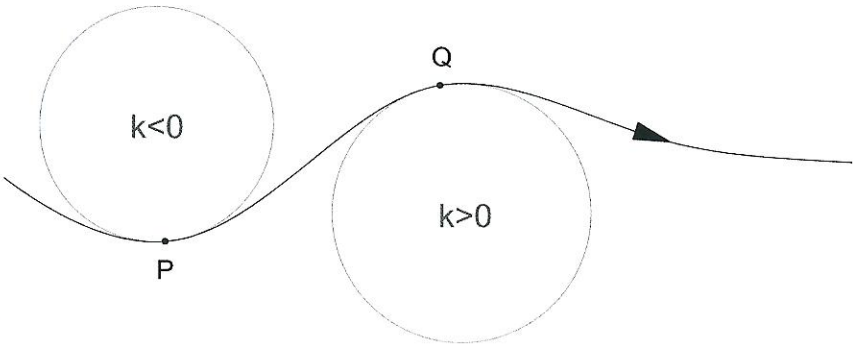
As follows, the curvature (k) can be defined as the reciprocal of the radius (r):

$$k = 1/r \quad [1]$$

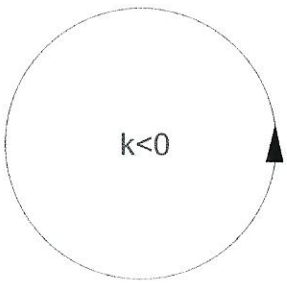
In order to extend this concept to generic curves a geometric construction is used: the *osculating circle*. Given a point P of a generic curve *c*, the circle which most closely approximates the curve near P is defined as the *osculating circle* at P. The curvature's value *k* of curve *c* at point P is defined as the reciprocal of *osculating circles* radius.



Curvature can also be defined as the vector $\bar{k}$ with direction PO and a magnitude equal to the reciprocal of the *osculating circle's* radius. Since curvature is the quotient of two positive numbers - and by definition the radius is a positive number – the curvature of a planar curve is always a positive number. However, ***signed curvature*** convention denotes a positive sign if the *osculating circle* lies to the left according to the default direction and has negative sign if the *osculating circle* lies to the right of the curve.

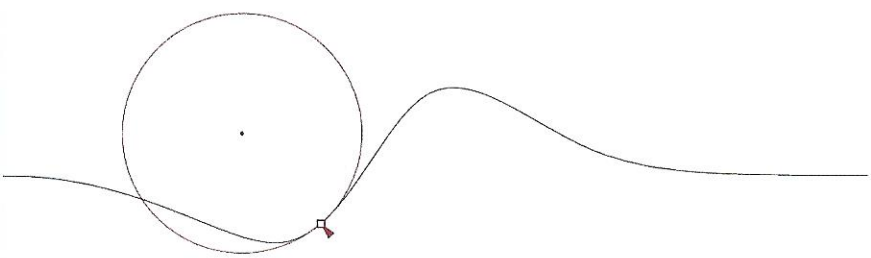


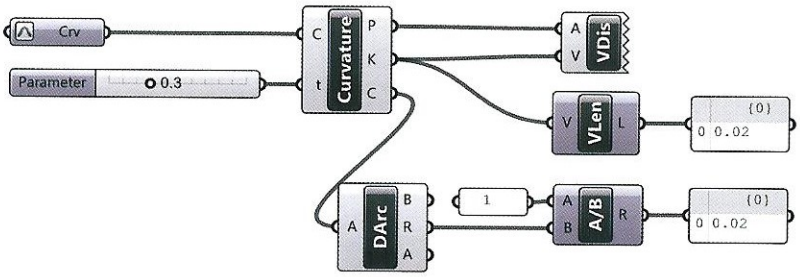
According to the default counter clockwise direction, the circle has a negative curvature sign.



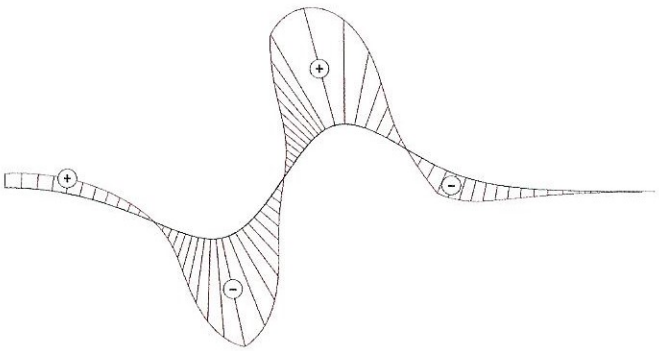
The component *Curvature* (Curve > Analysis) graphically displays the curvature vector $\bar{k}$ and the *osculating circle* at a generic point P expressed in the Local Coordinate System (LCS).

The curve's curvature value can be calculated in one of two ways: as the reciprocal of the radius of *osculating circle* (C), which can be extracted using the R-output of the component *Deconstruct Arc-DArc* (Curve > Analysis), or as the magnitude of the vector $\bar{k}$ which can be calculated using the component *Vector Length* (Vector > Vector).



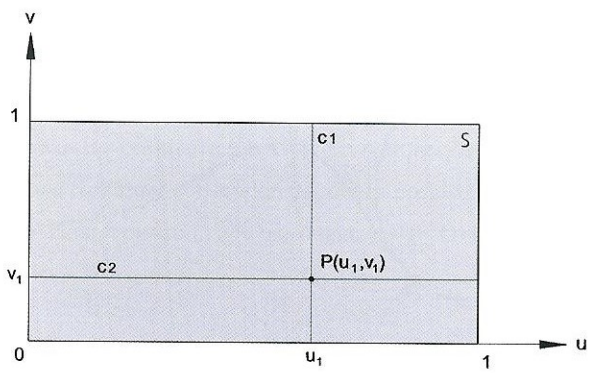


The component ***Curvature Graph*** (Curve > Analysis) visually displays curvature as a graph.



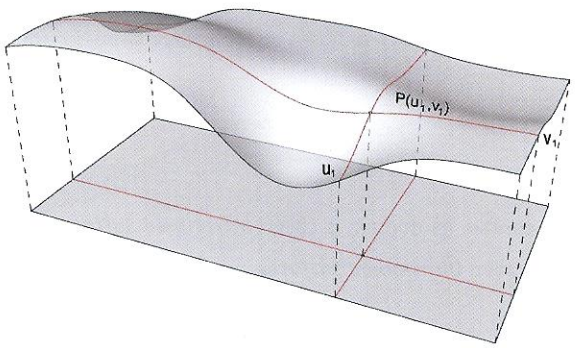
3.5 Parametric representation of a surface

Similar to curves, surfaces can be associated to a LCS. The parameters *u* and *v* (ranging between 0 and 1) operate similarly for surfaces as the parameter *t* operates for curves.

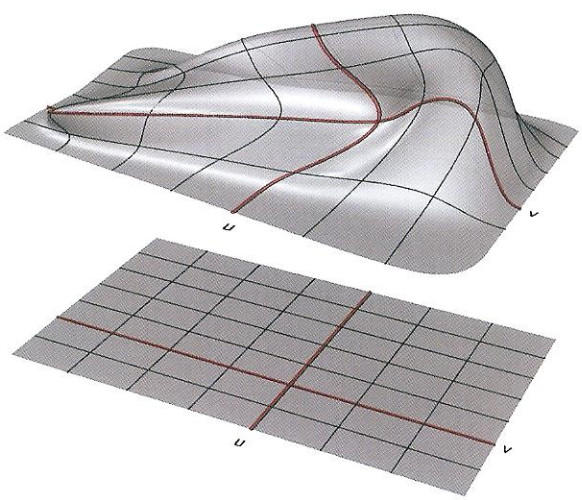


For each value of {u}, points P(u1,v) can be found to constitute the "sectional curve" C1. As follows, for each value of {v} points P(u,v1) can be found to constitute the "sectional curve" C2. Curves C1 and C2 are called ***Isocurves*** or *Isoparametric Curves*. In particular, C1 and C2 are the isocurves of the surface (S) at P(u1,v1). An isocurve is a curve with constant *u* or *v* values on a surface.

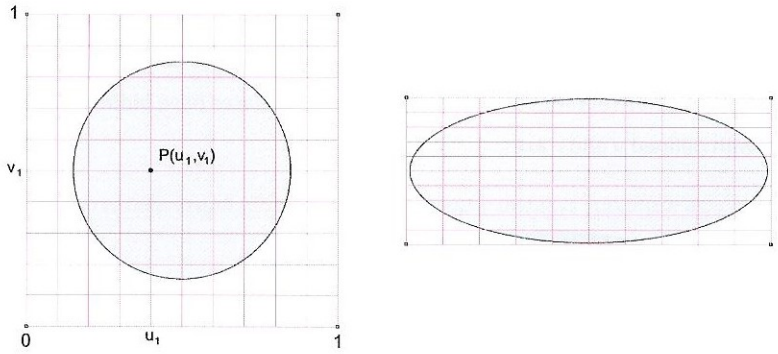
Isocurves create a rectangular grid generalizing the notion of the *Cartesian Grid* on a curved surface. This concept is valid for every freeform surface since a **NURBS-surface can always be imagined as the deformation of a flat-rectangular surface**. Moreover, surfaces have **bidimensional domains** (or **domain²**) with defined *u* and *v* axes.



As shown below, we can get a complex, freeform surface starting from a rectangular flat surface. The rectangular-orthogonal grid of isocurves becomes a rectangular-distorted grid.



Non-rectangular surfaces are however based on rectangular grids. This can be visualized by drawing a surface from a circle or from an ellipse and activating the rectangular grid of control-points. Rhino hides the trimmed area, but the actual domain is always rectangular.



A surface that does not contain its domain is called a ***Trimmed Surface***. Conversely, surfaces that contain their domain are called ***Untrimmed Surfaces***.

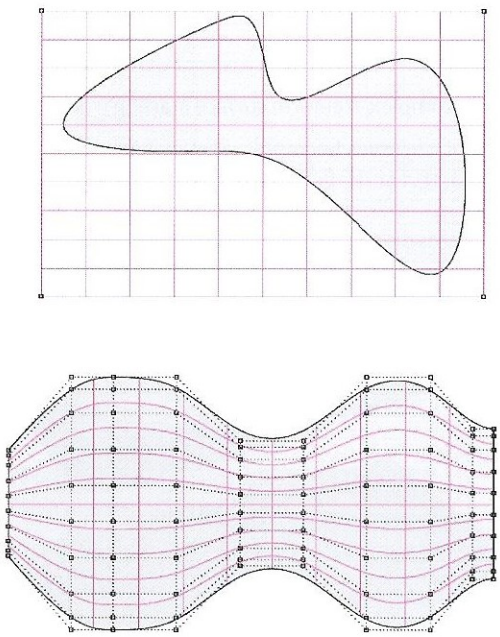
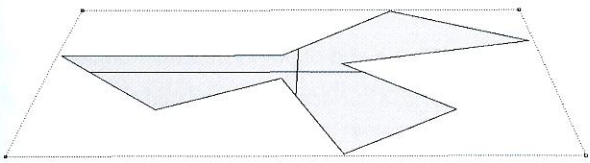


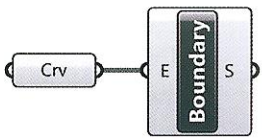
FIGURE 3.6
A trimmed (above) and an untrimmed (below).

3.6 Surface creation

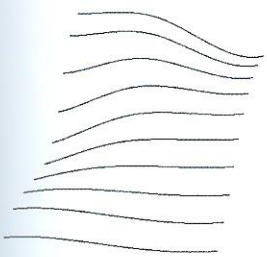
Grasshopper provides several components to create surfaces; they are hosted within the *Freeform* panel of the *Surface* tab:

- **Extrude:** is the easiest way to create a surface. The component *Extrude* (Freeform > Surfaces) requires a profile curve (open or closed) or a surface to extrude according to a vector. Extrude yields a surface with a constant section.
- **Boundary surface**: creates a trimmed or untrimmed **planar** surface from a collection of boundary edge curves. The component *Boundary Surfaces* (Freeform > Surfaces) requires planar and continuous closed curves to result in a surface.

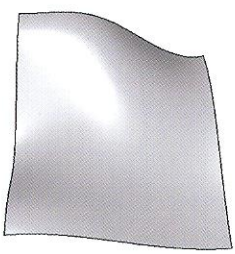




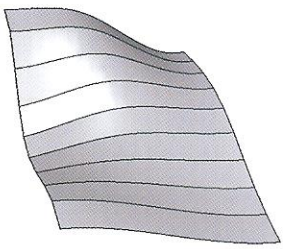
- **Loft**: creates a surface through a set of ordered section curves. The component *Loft* (Freeform > Surfaces) requires at least two curves to generate a surface. The component *Loft Options* (Freeform > Surfaces) controls the method of surface creation, or the default method will operate. *Loft Options* specifies whether a surface is closed, if the seams should to be adjusted, if surfaces should be rebuilt, a refit tolerance, as well as one of six loft types specified by an integer: 0=Normal, 1=Loose, 2=Tight, 3=Straight, 4=Developable, 5=Uniform.



section curves

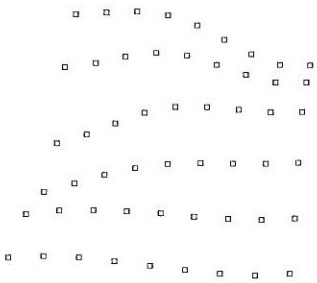


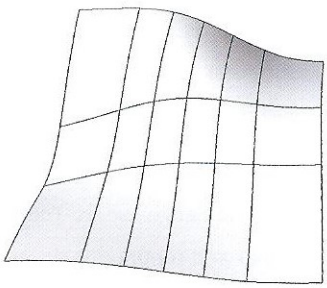
regular loft



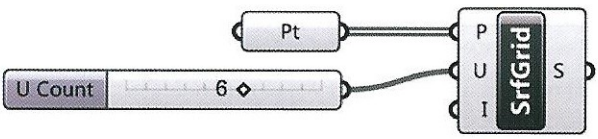
straight loft

- **Surface from points**: creates a surface from a grid of points. The component *Surface from Points* (Freeform > Surfaces) is useful for creating surfaces from other surfaces, or as discussed in 2.3.2 from mathematical functions.

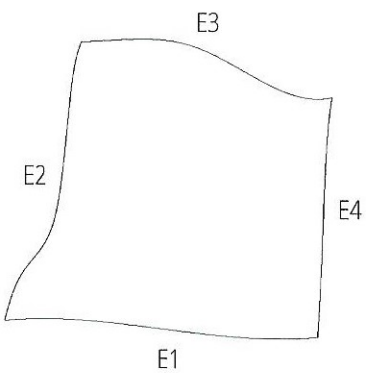


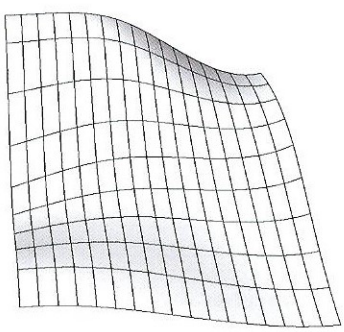


The component *Surface from Points* requires a grid of points and the number of points in the *u* direction to be declared. For instance, a surface can be created from a 9 x 6 grid of points by specifying a U-count of 6.

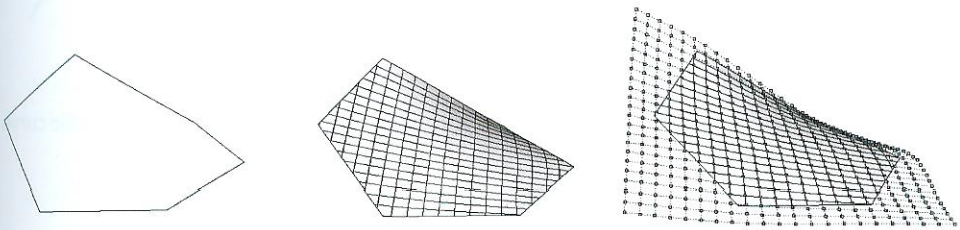


- **Edge surface** creates a surface from two, three or four curves. The component *Edge Surface (Freeform* > Surfaces) interpolates a surface from ordered edge curves.

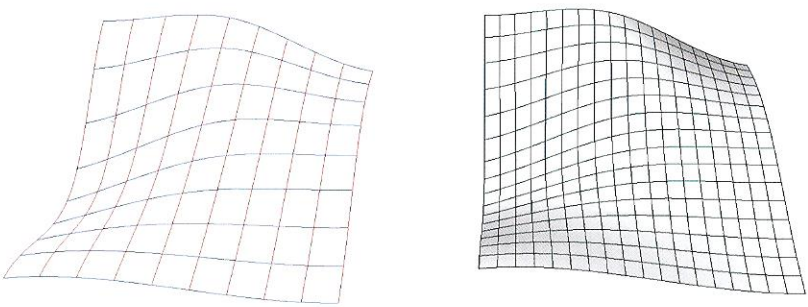




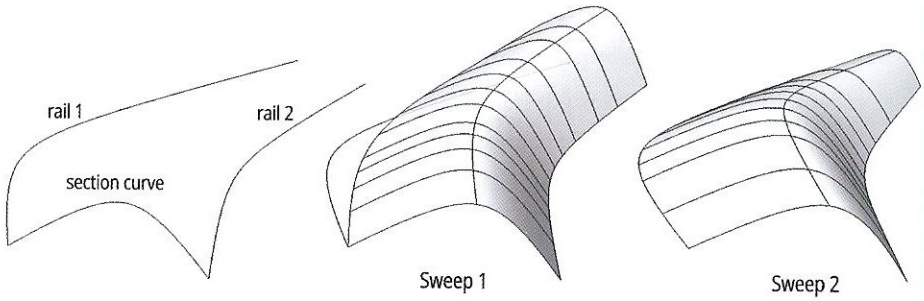
- **Patch:** fits a surface through a set of points, or curves that are open or closed. The component *Patch* (Freeform > Surfaces) generates a series of perpendicular curves, called spans, from input curves which compose a trimmed or untrimmed surface. Span curves are rarely parallel to any edge of the output surface so the *Patch* component is used when it's not possible to perform other methods.



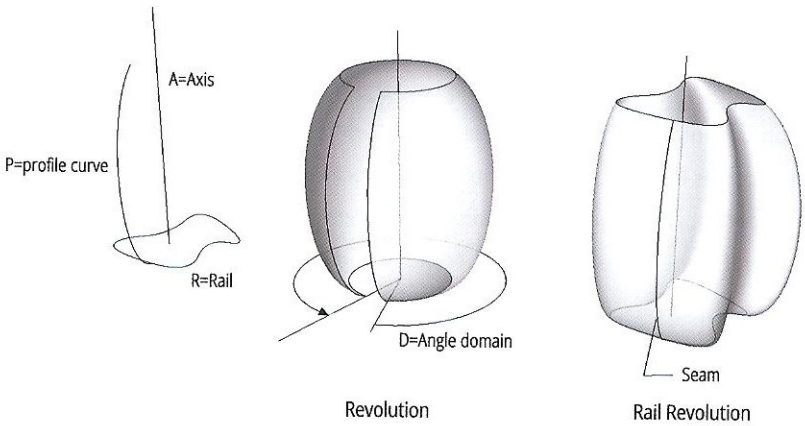
- **Network surface** operates on two sets of ordered curves to create a single surface. The component *Network Surface* (Freeform > Surfaces) requires curves in the *u* direction and the curves in the *v* direction. The C-input specifies the type of surface continuity (0=*Loose*, 1=*Position*, 2=*Tangency*, 3=*Curvature*. The *Network Surface* component allows for more control of the surfaces edges, when compared to the *Loft* component.



- **Sweep 1 and Sweep 2:** generate a surface using a section curve S swept along one or two rail curves R. *Sweep1* (Freeform > Surfaces) should be used with one rail curve and *Sweep2* (Freeform > Surfaces) with two rail curves.



- **Revolution and Rail Revolution:** generate surfaces from the revolution of a profile curve P around an axis or using a sweep rail respectively.



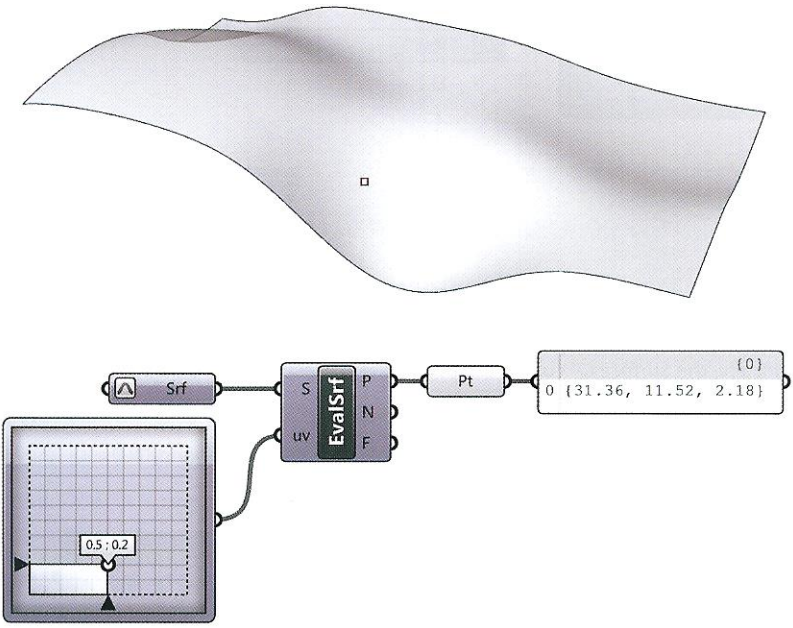
3.7 Analysis of surfaces using Grasshopper

3.7.1 Finding points on a surface: *Evaluate Surface* component

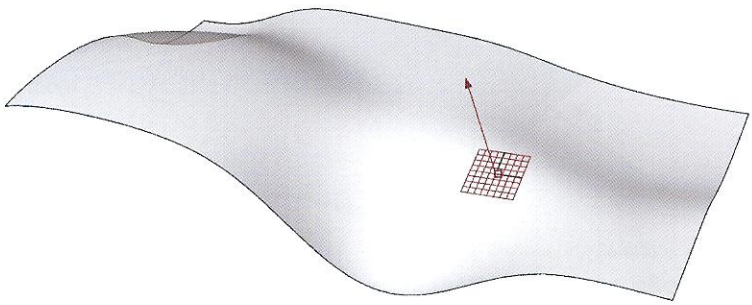
Parametric Representation is used to find points on a surface. The component *Evaluate Surface* (Surface > Analysis) requires a surface (S) to analyze and two LCS coordinates *(u and v)*. The component *MD Slider* (Params > Input) can be used to assign *u* and *v* coordinates.

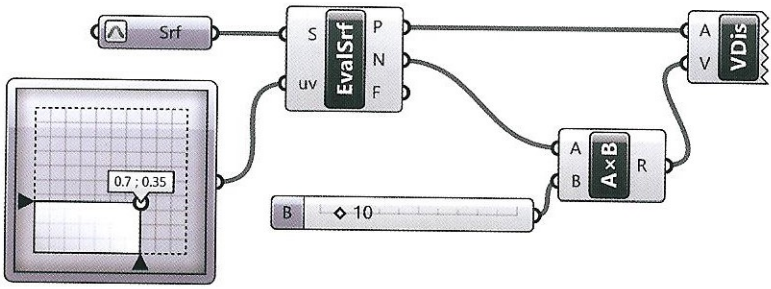
The *MD Slider* (or multi dimensional slider) is a bidimensional extension of the *Number Slider*. By default its values range between 0 and 1 in *u* and *v directions,* assuming the input surface is parameterized between the same interval.

As for curves, the *Surface* component and the S-input of *Evaluate Surface* allow us to reparameterize the input surface by right-clicking the *Surface* component or the S-input of *Evaluate Surface* and selecting *Reparameterize*.



The *Evaluate Surface* component outputs: a point (P) expressed in the WCS, a normal vector (N) at P and a tangent plane (F) at P. The displayed dimension of planes can be set from *Display* > *Preview Plane*.

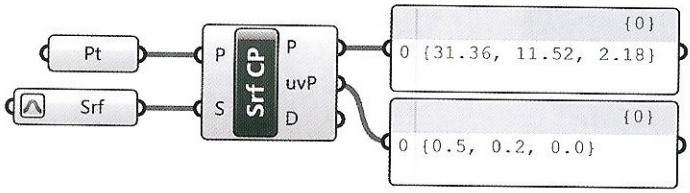


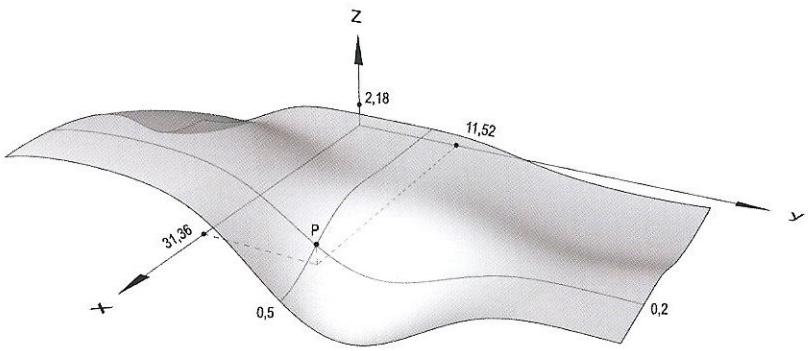


3.7.2 Finding points on a surface: *World to Local* conversion

The component *Surface CP* (analysis > surface) can be used to convert a point P on a surface expressed in the WCS to a point expressed in the LCS.

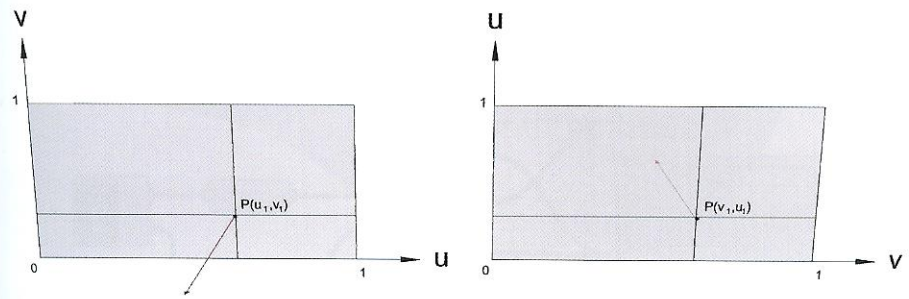
More specifically, the *Surface CP* component finds the closest point P' on a surface, given an external point P. The displacement distance between points P' and P is provided by the output (D). When the component *Surface CP* is used to convert between WCS and LCS: P=P' and D=0.



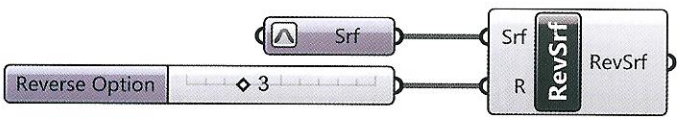


3.7.3 Inverting direction: *Reverse Surface Direction* component

The component *Surface Direction* (LunchBox > Util) inverts the *uv* direction of a surface. The Tab **Lunch Box**, developed by Nathan Miller[9] is available as a plug-in for Grasshopper and includes utilities for mathematical forms, paneling systems, structures, and workflow.



The R-input of *Surface Direction* specifies the reverse option: if R=0 the directions will not be inverted, if R=1 the *u* direction will be inverted, if R=2 the *v* direction will be inverted and if R=3 both *u* and *v* will be inverted. Often it is required to *reparameterize* the RevSrf-output.

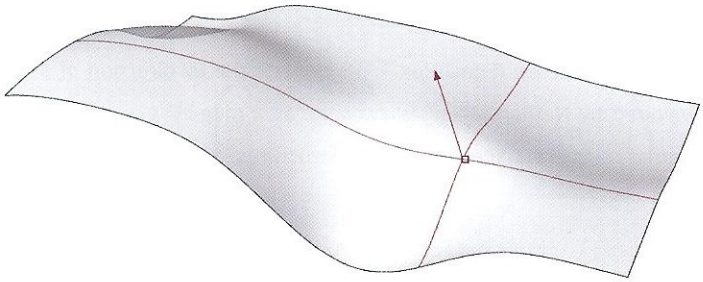


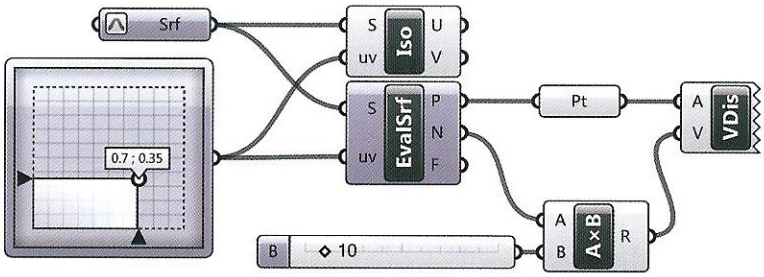
3.7.4 Extracting Isocurves: *Isocurve* component

The component ***Isocurve*** (Curve > Spline) extracts the isocurves of a surface at a specific point (P), given its local coordinates (uv). Isocurves U (parallel to *u* direction) and V (parallel to *v* direction) are extracted separately. The U or V isocurves can be visualized independently by connecting a curve component to the U or V output and hiding the *Isocurve* component.

NOTE 9

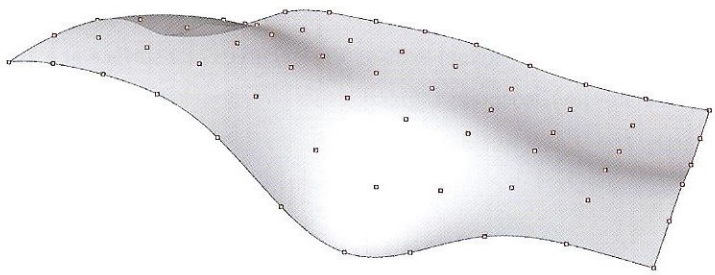
Nathan Miller is an Associate Partner and Director of Architecture & Engineering Solutions at CASE where he is responsible for leading the efforts on computational design strategy and complex modeling and geometric rationalization. His expertise in 3D modeling programs and fluency in several scripting and programming languages provides leading-edge solutions and capabilities for significant AECO clients. Explorations on computational design are published on his blog: http://www.theprovingground.org/

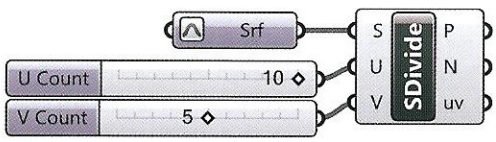




3.7.5 Dividing a surface: *Divide Surface* component

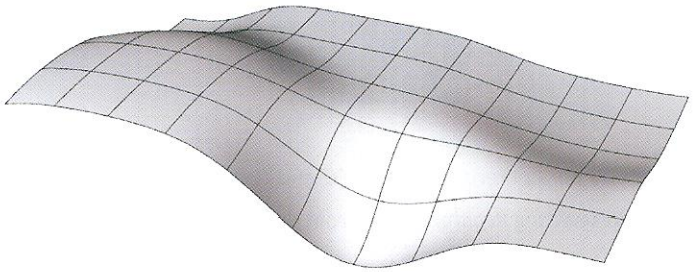
The component *Divide Surface* (Surface > Util) generates a grid of points on a surface. The points (P) are the intersection vertices of a grid of isocurves calculated by dividing the *u* and *v* axes evenly by a positive integer input. The *Divide Surface* component outputs: normal vectors (N) at points (P) as well as their local coordinate (uv).



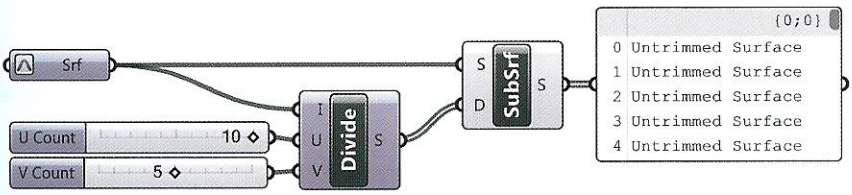


3.7.6 Splitting a surface: *Isotrim* component

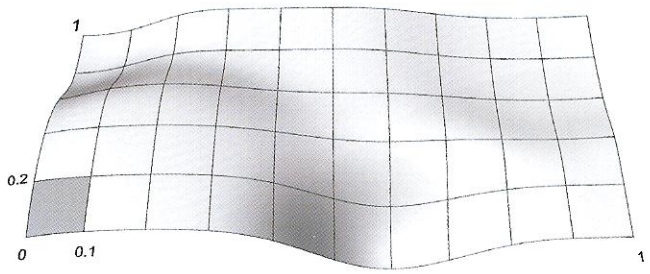
The component *Isotrim-SubSrf* (Surface > Util) splits a surface into a set of contiguous sub-surfaces based on a grid of splitting isocurves. For example, the image below is a surface split into 50 sub-surfaces.

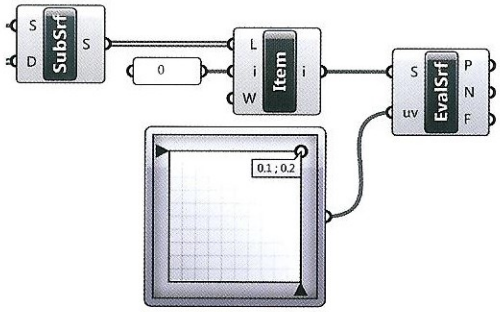


The *Isotrim* component is used in conjunction with the component *Divide Domain²* (Maths > Domain). The *Divide Domain²* component sets the number of divisions in both *u* and *v* directions generating a cutting-grid on the surface for the *Isotrim* component to operate on.

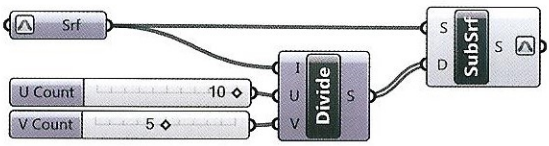


The *Isotrim* component outputs a set of *untrimmed* surfaces parameterized according to the domain of the parent-surface. For example, the domain of the bottom-left sub-surface is [0;0.1] in the *u* direction and [0;0.2] in the *v* direction.

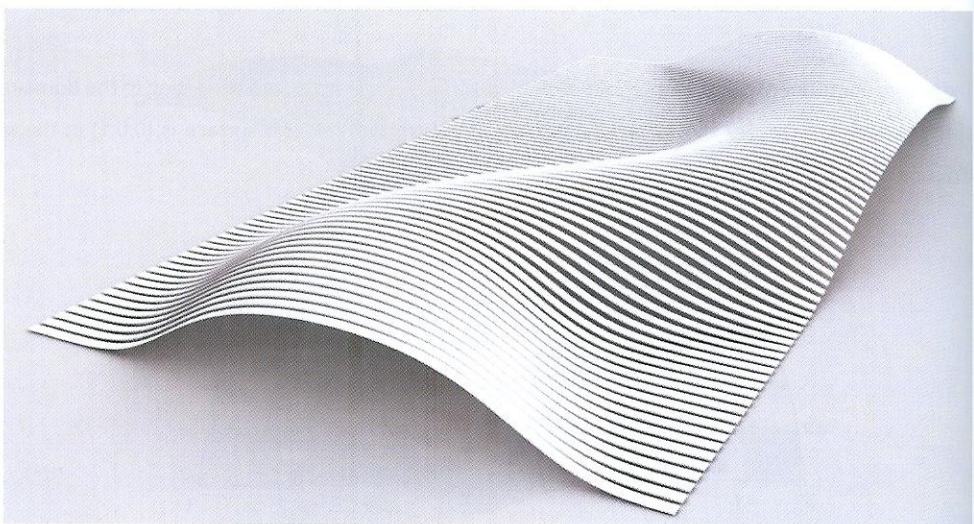


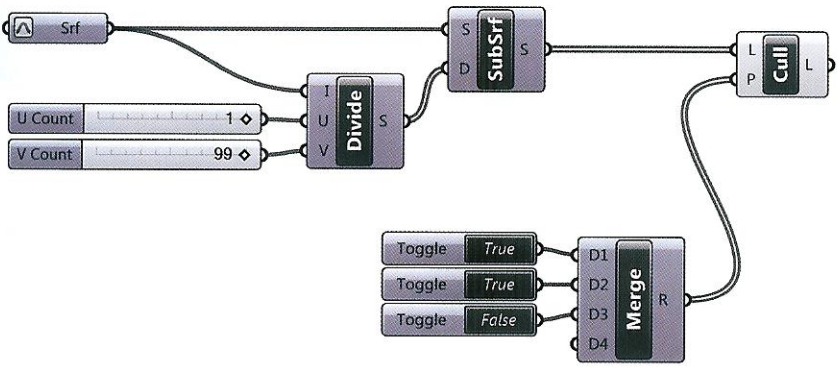


Each sub-surface can be parameterized between 0 and 1, by right-clicking on the S-output of the *Isotrim* component and selecting *Reparameterize* from the context menu.



If 1 is set as the U-input or the V-input of *Divide Domain²* component, a series of contiguous stripes that follows the parent-surface will be generated. Patterns can be created using the component *Cull Pattern* (Sets > Sequence) with a list, for example, of (True, True, False) Boolean inputs.





3.7.7 Strategy: uneven splitting

The cutting-grid generated by *Divide Domain²* is evenly spaced because the *u* and *v* axes are divided into equal lengths by the U and V input values respectively. To generate an unevenly spaced grid the distribution of points within *u* and *v* are required to be adjusted.

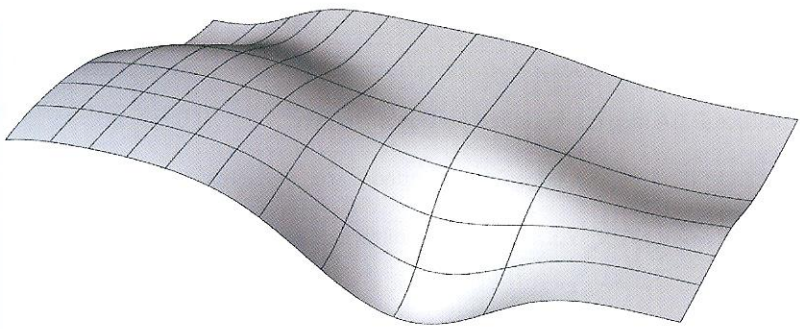
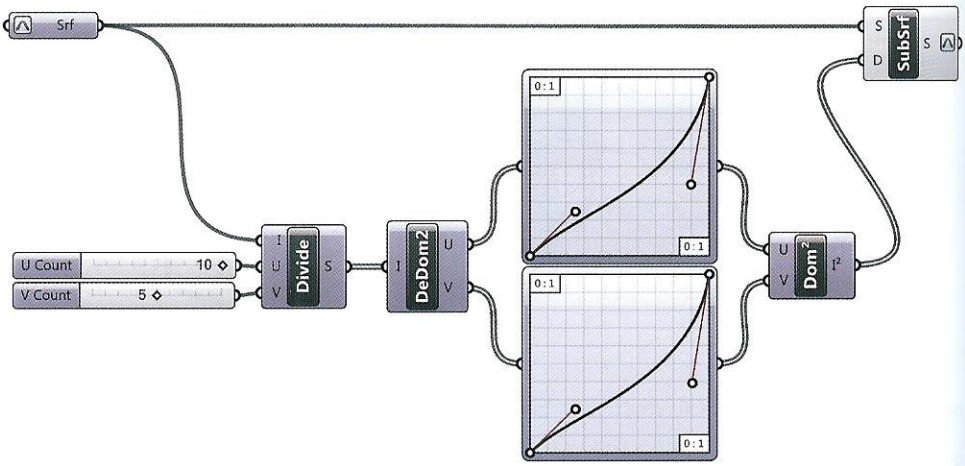


FIGURE 3.7
Uneven splitting of a NURBS surface.

To construct an unevenly spaced grid: first, the bidimensional domain must be divided into its mono-dimensional components U and V using the component *Deconstruct Domain²* (Maths > Domain), then the distribution of points within the two domains (U and V) are changed by the component *Graph Mapper* (Params > Input) set to *Bezier*, by right-clicking on the component and specifying *Bezier* from the context menu; next, the component *Construct Domain²* (Maths > Domain) reassembles the mono-dimensional domains into a bidimensional domain, and finally the surface is split using the *Isotrim* component. Acting on graph's grips will change the grid spacing.



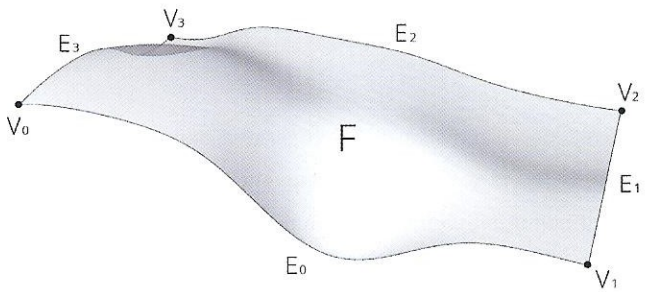
3.7.8 Decomposing a surface: *Deconstruct Brep* component

NURBS-surfaces can be imagined as a geometrical object composed of three geometric entities: ***faces***, ***edges***, and ***vertices***.

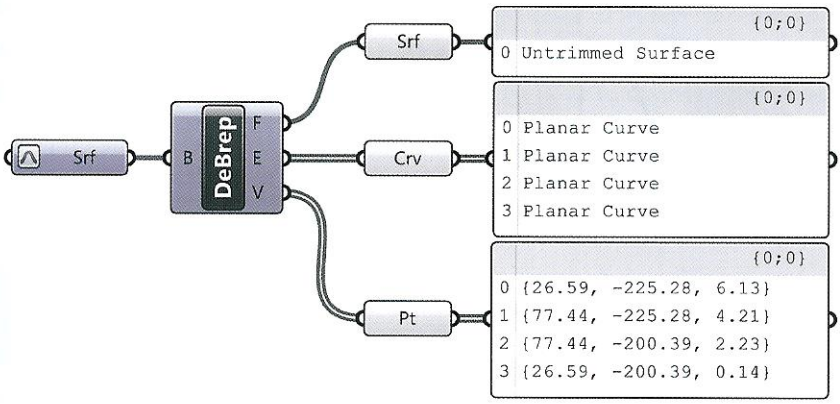
For every surface:

- One *Face* (F_0). A face is a bounded portion of surface;
- Four *Edges* (E_0, E_1, E_2, E_3). Edges are bounded pieces of curves;
- Four *Vertices* (V_0, V_1, V_2, V_3). Vertices are points that lie at corners;

can be extracted.

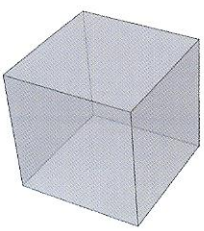


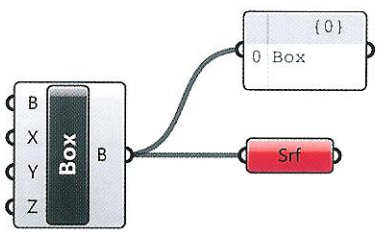
The component *Deconstruct Brep* (Surface > Analysis) is used to extract *faces*, *edges*, or *vertices* from a surface.



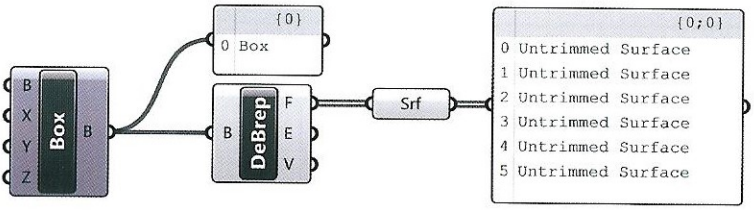
The term **Brep** (or **boundary representation**) describes a way of defining geometric forms using boundaries. Primitives such as Boxes, Cylinders, Cones as well as other geometric solids are defined by boundary representation. Meaning, that a solid or a polysurface is composed of a collection of edges and faces "stitched" together to form a solid object. For instance, a NURBS surface can be considered as a collection of one face.

For example, if the component *Center Box* (Surface > Primitive) is used to define a box with a specified domain and the *Surface* component is connected to the box output (B) the *Surface* component will turn red to display a data mismatch error.



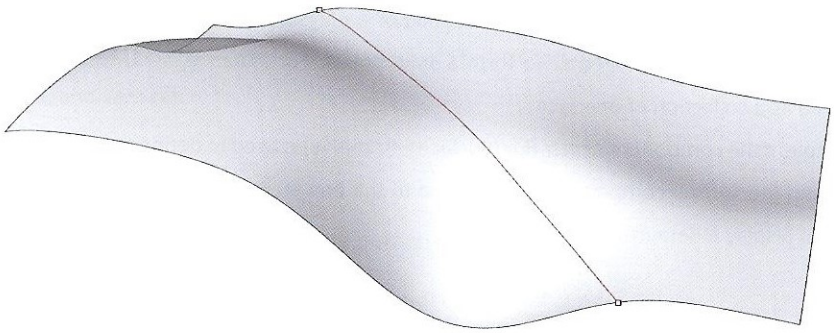


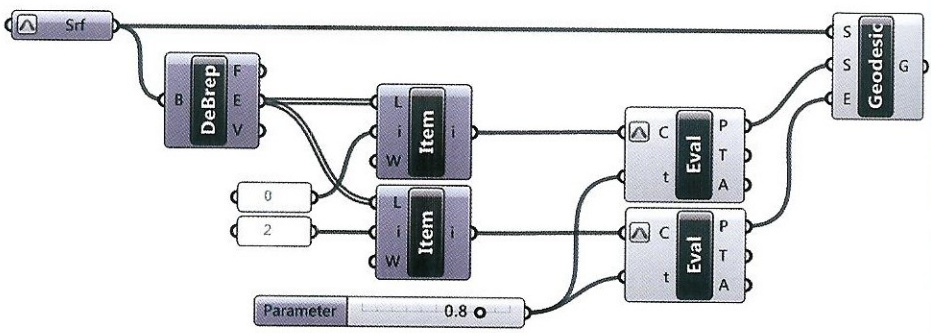
The error occurs because the box is constituted by six surfaces joined together into a Brep. In order to extract surfaces, the box or Brep has to be deconstructed into surfaces using the component *Deconstruct Brep*.



3.7.9 Splitting a surface using generic curves: *Surface Split* component

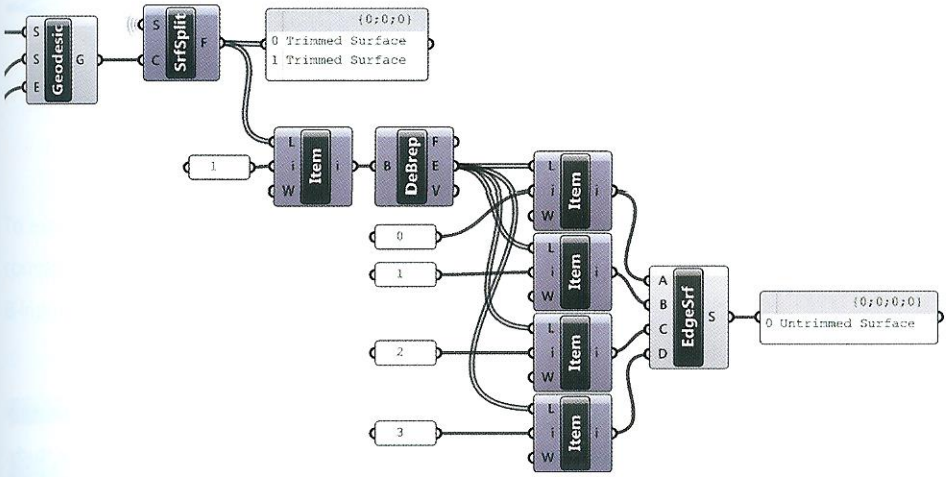
A surface can be split by a set of curves that are coincident to the surface. For example, a ***geodesic*** curve can be used as a cutting curve to split a surface into two parts. Given a surface (S) and two points (S) and (E) placed on opposite edges, a geodesic (or a shortest path curve) can be constructed between points (S) and (E) on the surface. A geodesic line can be considered a straight line in curved space.

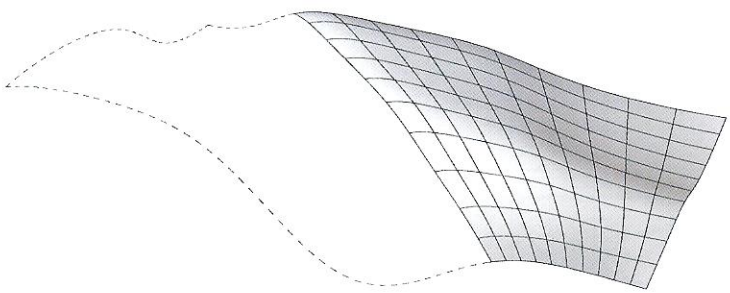




The component *Surface Split* (Intersect > Physical) splits a surface (S) at splitting curve (C) which in this case is defined by the geodesic curve.

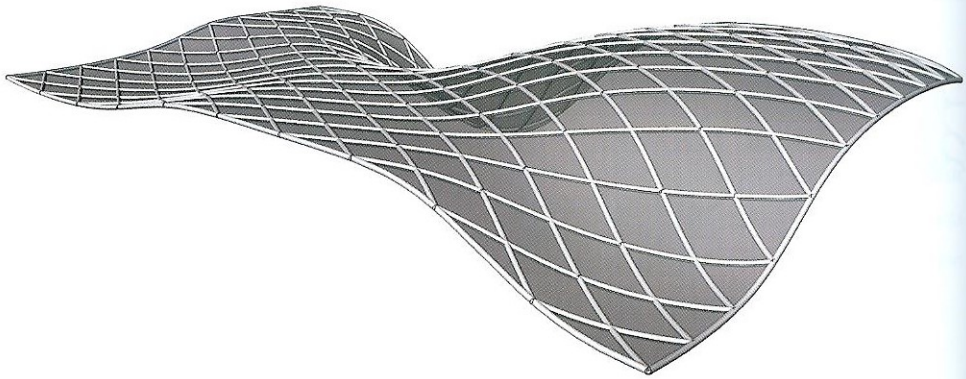
The result of the *Surface Split* component is two *trimmed* surfaces. To create an *untrimmed* surface from a *trimmed* surface the *Deconstruct Brep* component can be used to extract the edges and create a new surface using the component *Edge Surface* (Surface > Freeform).



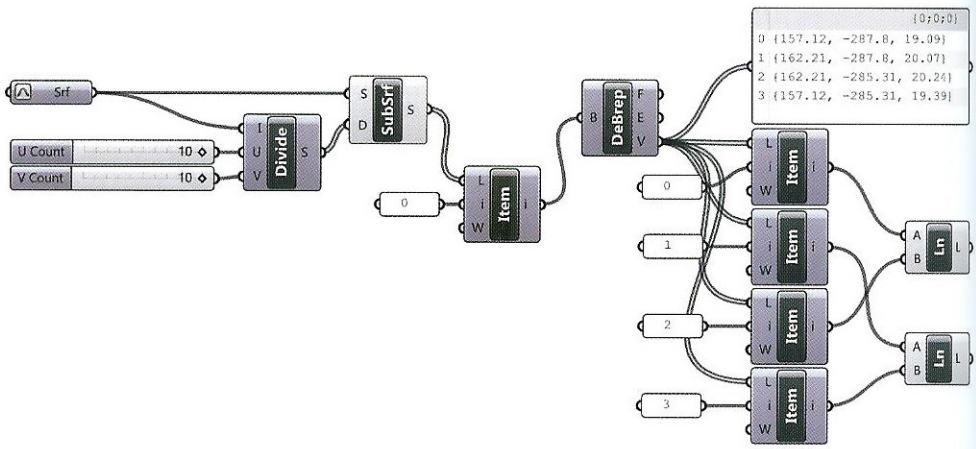


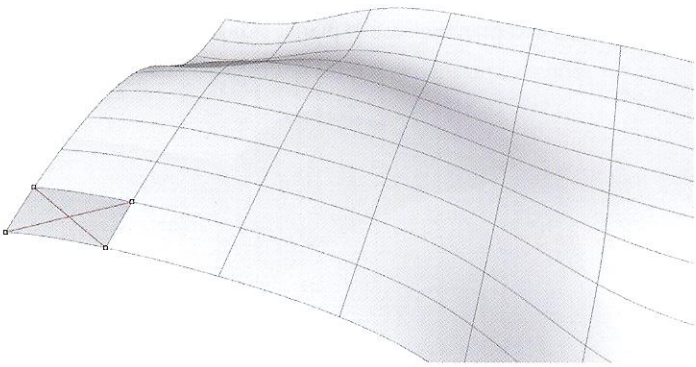
3.7.10 Example: diagrid

A diagonal grid or a *diagrid* can be created on a target surface by using the decomposition logic of Brep geometry.

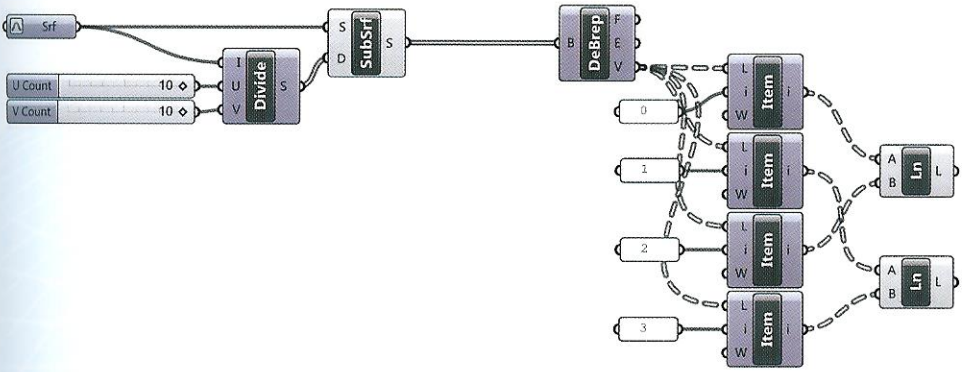


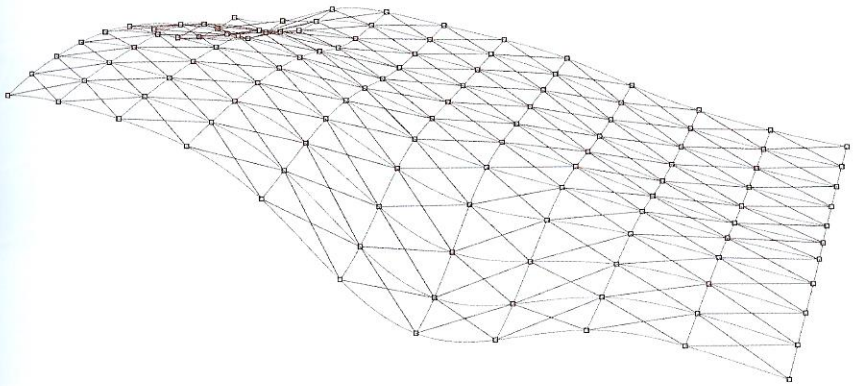
A surface set from Rhino is evenly divided into a set of sub-surfaces using the *Isotrim* component. To understand the logic of creating a diagrid, one surface item (0) will be examined as a case study. A diagrid is created by connecting opposite vertices for each sub-surface. The vertices are extracted using the *Deconstruct Brep* component and itemized separately by four *List Item* components. A line is created between the items (0 to 1) and (1 to 3), creating one module of the diagrid pattern.



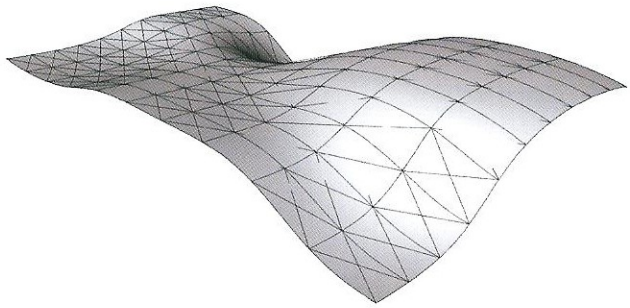


To extend the diagrid logic to the overall surface the *List Item* component, placed after the *SubSrf* component, is deleted. Then a direct connection is made between the S-output of *SubSrf* and the B-input of *Deconstruct Brep*.

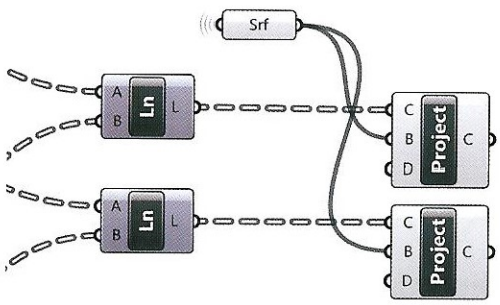


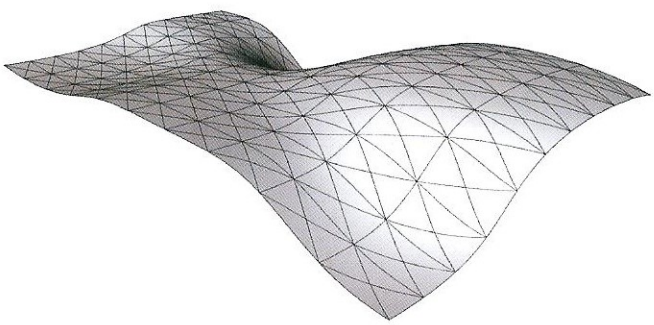


Since the algorithm is defined by connecting the vertices of sub-surfaces, the diagrid is not coincident to the surface, except in the case of planar sub-surfaces.

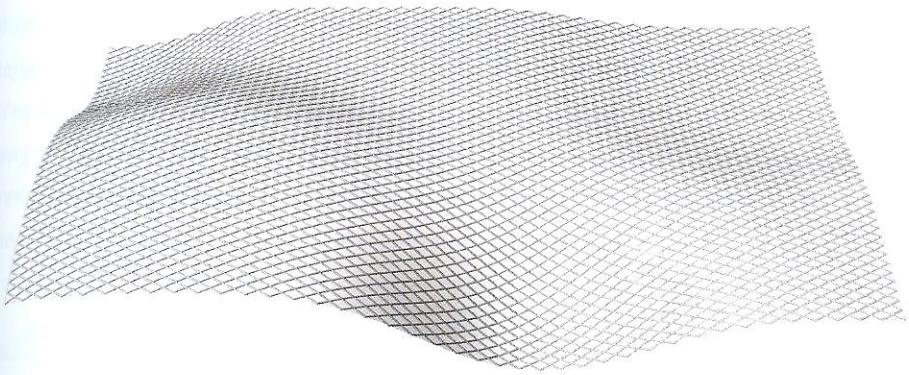


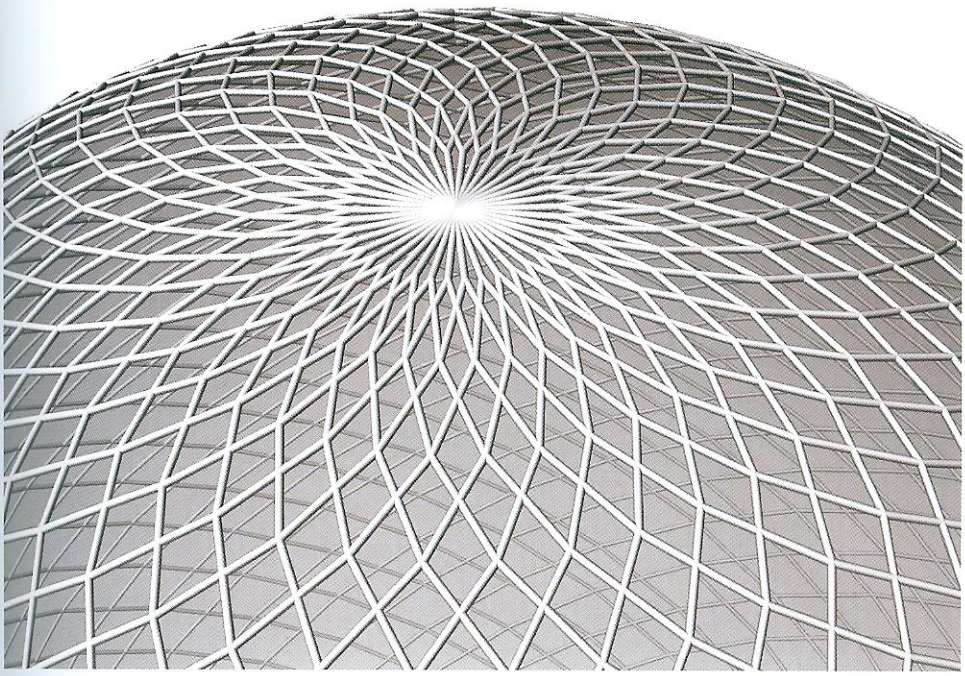
In order to generate a diagrid that is coincident with the target surface, the component *Project* (Curve > Util) is used to project the curves onto the target surface. The output of the *Line* component (L) is connected to the C-input of the *Project* component, and a receiver (wireless connection) of the original surface is connected to the B-input of the *Project* component, projecting the diagrid onto the original surface. Line segments will be replaced by curves as the output (C) of the *Project* component.





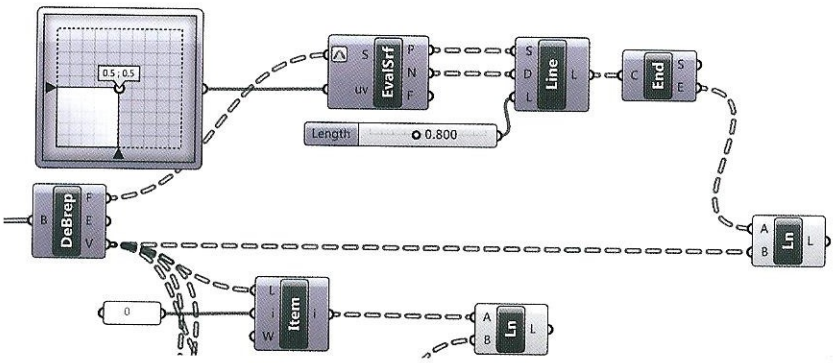
The diagrid algorithm is not bound to a specific surface, the algorithm can be used as a diagrid generator for any surface. The component *Pipe* (Surface > Freeform) will sweep a circle of a specified radius along the diagrid curves (used as rail curves) creating a three dimensional diagrid.

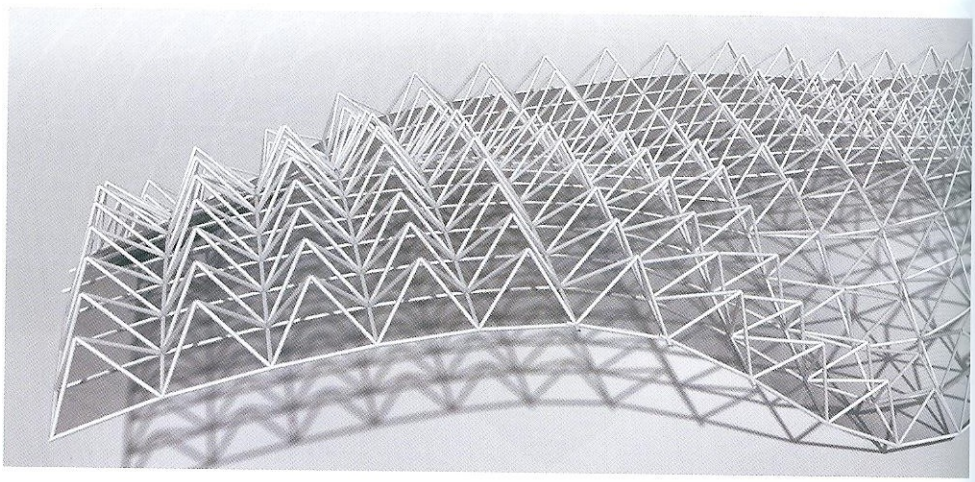




3.7.11 Example: space frame

The diagrid-algorithm can be extended to create a ***space frame***. The F-output of the *Deconstruct Brep* component is a set of sub-surfaces, so the approximate center point of each sub-surface can be calculated using the *Evaluate Surface* component connected to an *MD Slider* set to (0.5;0.5). Since the surfaces returned from the F-output of *Deconstruct Brep* are parameterized according to the domain of the parent surface they must be **reparameterized**. The P-output of the *Evaluate Surface* component is a set of points approximating the centers, and N-output the vector normals to the surfaces. The component *Line SDL* (Curve > Primitive) is used to create a set of line segments given: the start points (S) declared by the P-output, the directions (D) declared by N-output and lengths (L) specified by a slider. The component *Endpoint* is used to extract the end points of the lines which are finally connected to the V-output of *Deconstruct Brep*, generating the remaining components of the space frame.





3.7.12 Grid of equidistant points on a generic surface

Parametric Space can be imagined as a deformed *Cartesian Grid* which perfectly fits an arbitrary freeform surface. The dimensions of the grid are dependent on the initial domain as well as its subdivision. As a consequence, **edges do not have the same length** unless the surface curvature is equal to zero.

To generate a grid with equal-length edges, a grid of equidistant points on a surface is required to be declared. The Russian mathematician **Pafnuty Lvovich Chebyshev** (1821-1894) developed a method called the **Chebyshev-net** to clothe curved surfaces by cutting and sewing flat pieces of fabric. The method is published in the book *On the cutting of our clothes* (1878).

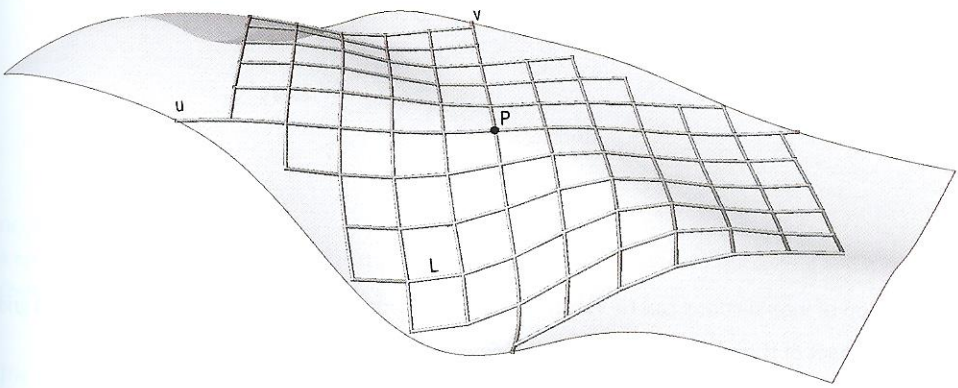


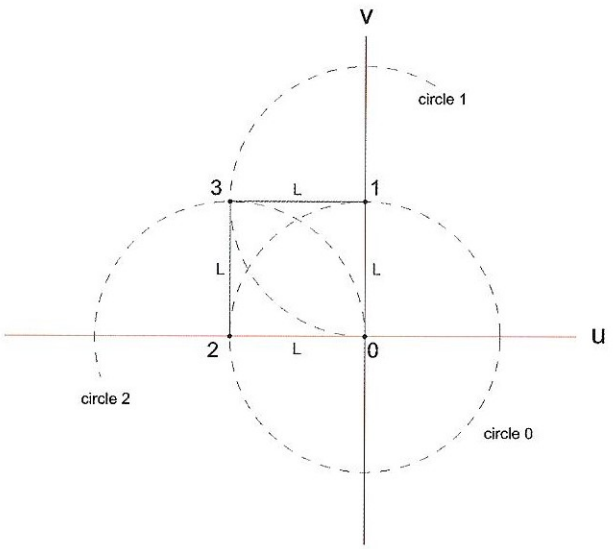
FIGURE 3.8
A grid of equal-length edges generated on an arbitrary freeform surface.

Given two arbitrary transverse curves *u* and *v* starting from an arbitrary point (P) of a surface, and a specified edge-length (L) a unique Chebyshev-net can be found. The Chebyshev-net method can be applied to the plane as well as any freeform surface. The geometric approach[10] to the Chebyshev-net can be imagined as a tridimensional extension of the *Divide Distance* logic (see 3.3.8).

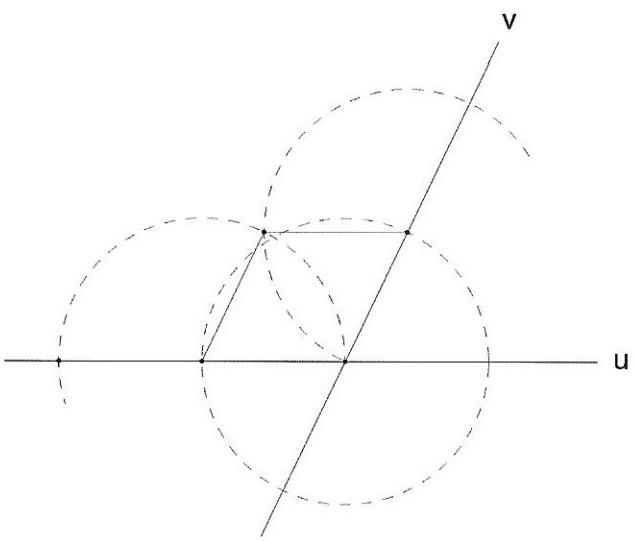
To find a grid of equidistant points in a bidimensional space a generic point 0 is used to draw two orthogonal curves *u* and *v*. The desired edge-length (L) is defined by drawing a circle *(circle 0)* with radius (L) originating from *point 0*; then intersecting the circle with the isocurves *u* and *v* to define *point 1* and *point 2*. At the intersection points circles (*circle 1* and *circle 2*) are drawn with the same

NOTE 10
E.V. Popov, 2002, *Geometric Approach to Chebyshev Net Generation Along an Arbitrary Surface Represented by NURBS.*

radius as *circle 0*. The circles intersection defines *point 3*. The procedure is repeated for the remaining three quadrants to create a grid.



The method of intersections can be extended to non-orthogonal curves *u* and *v*. Resulting in a grid formed by a set of rhombi with equal-length sides.



Moreover, curves *u* and *v* are not required to be straight. The following image displays a grid of equidistant points calculated from two isocurves with a degree of 3 on a flat surface originating from an arbitrary point 0. An equidistant grid generates a leftover area. Leftover areas do not happen in the case of domain-based grids since the segment length is dependent on equally dividing the domain.

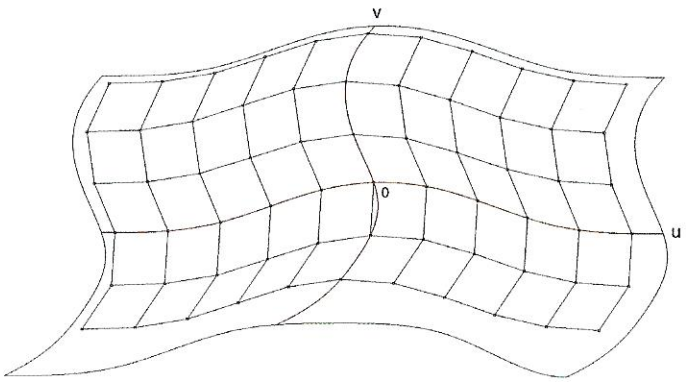
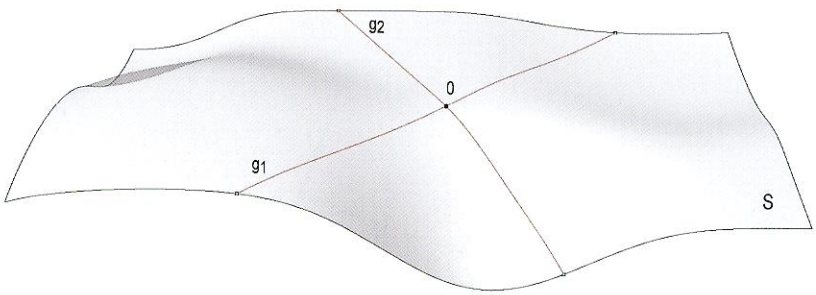


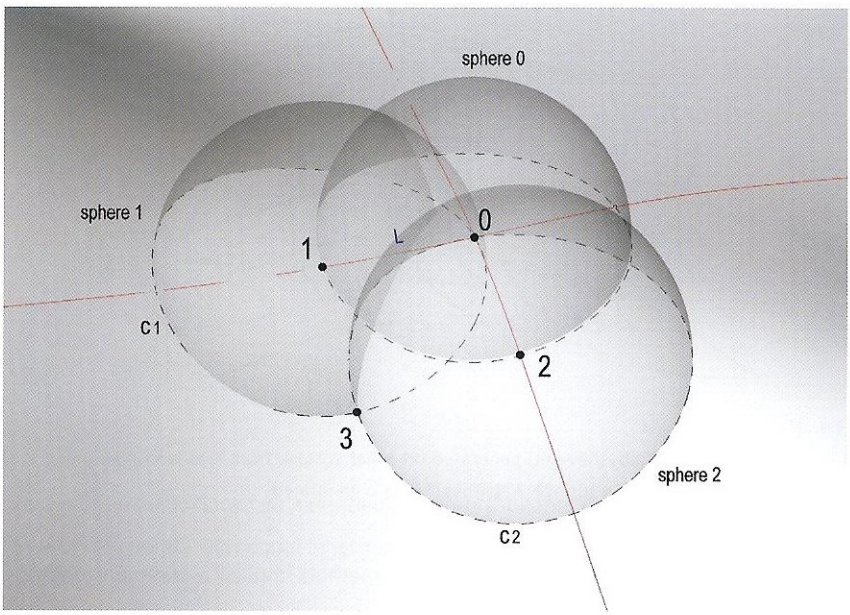
FIGURE 3.9
A net of equidistant points (Chebyshev-net) generated on a flat surface from two isocurves *u* and *v*. Usually a Chebyshev-net cannot perfectly fit a surface. A "leftover" area is generated.

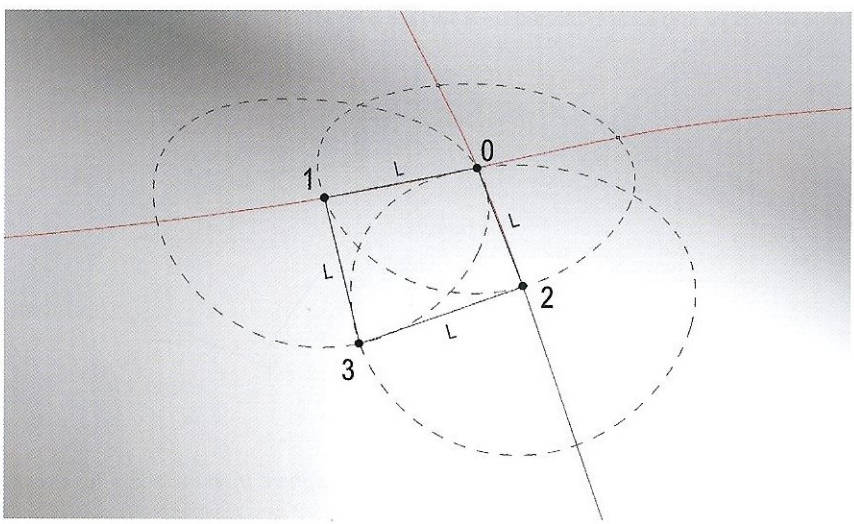
The tridimensional extension of the bidimensional method implies a tridimensional geometric construction. Meaning, a set of spheres are used instead of circles. To construct a tridimensional net, two generic curves *g1* and *g2* are defined on a surface S and their intersection (*point 0*) is found.

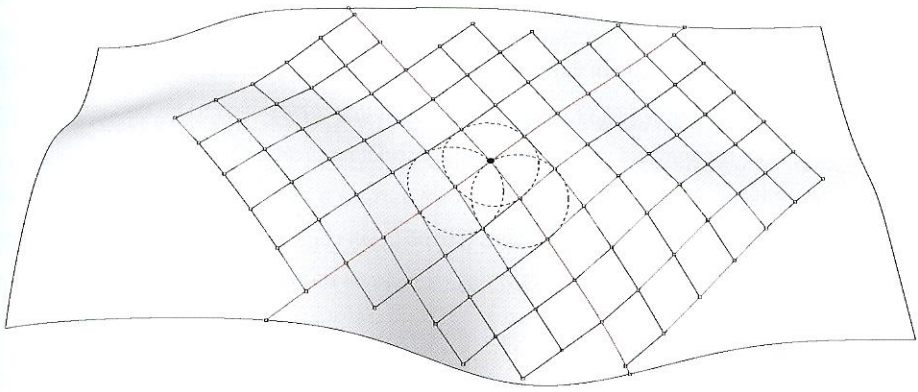


Then - expanding the bidimensional method - a sphere (*sphere 0*) with radius (L) is drawn and intersected with curves *g1* and *g2* to define *point 1* and *point 2. Point 3* is found by intersecting two

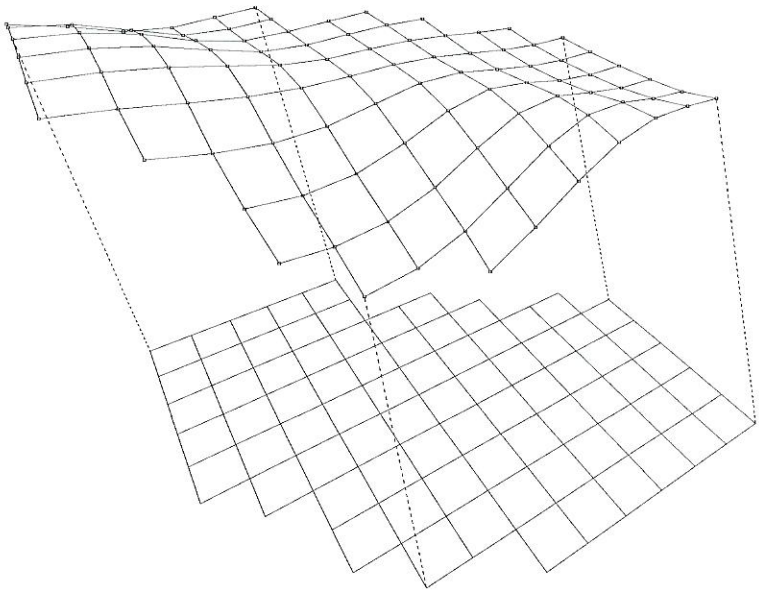
spheres (*sphere 1* and *sphere 2*) wih the same radius (L) with the surface, generating curves which are intersected to define *point 3*. The procedure must be iterated and repeated for the remaining three quadrants to generate the complete grid. Grasshopper does not provide a built-in component that allows users to perform iterations and create a Chebyshev-net. Anyhow, as we will see in chapter 7, a specific add-on is available to perform iterative procedures.







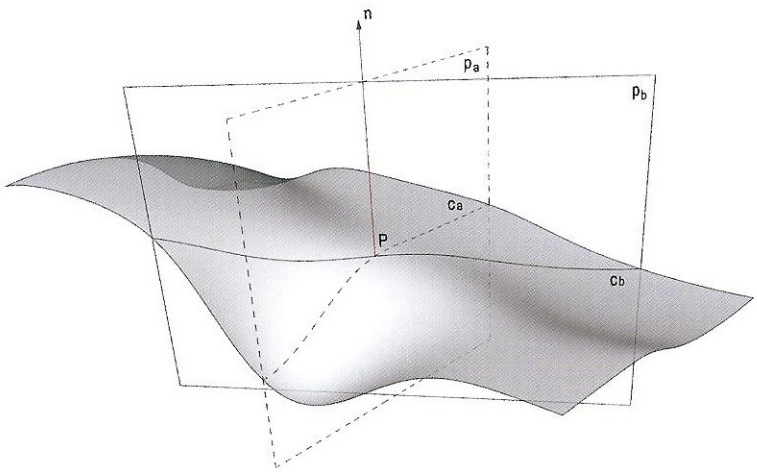
It is important to point out that an equidistant point grid can be flatten in a regular square grid as displayed below. This characteristic is crucial to form freeform structures (gridshells) starting from planar and deformable elements.



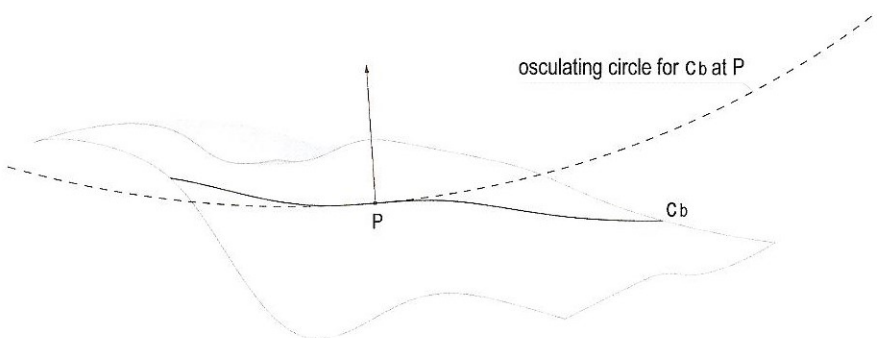
3.8 Notion of Curvature for surfaces

The curvature of a surface at a point P measures how the surface deviates from the tangent plane at P. In order to define the curvature of a surface the *osculating circle* needs to be defined following a procedure similar to the method used to define the *osculating circle* for planar curves.

For each point P belonging to an arbitrary freeform surface a normal vector *n* can be calculated. A infinite number of planes or ***sheaf of planes*** (*pa, pb,..., pi*) containing the normal vector *n* can be found. As follows, an infinite set of curves (*ca, cb,..., ci*) can be found as the intersection between the planes and the surface.

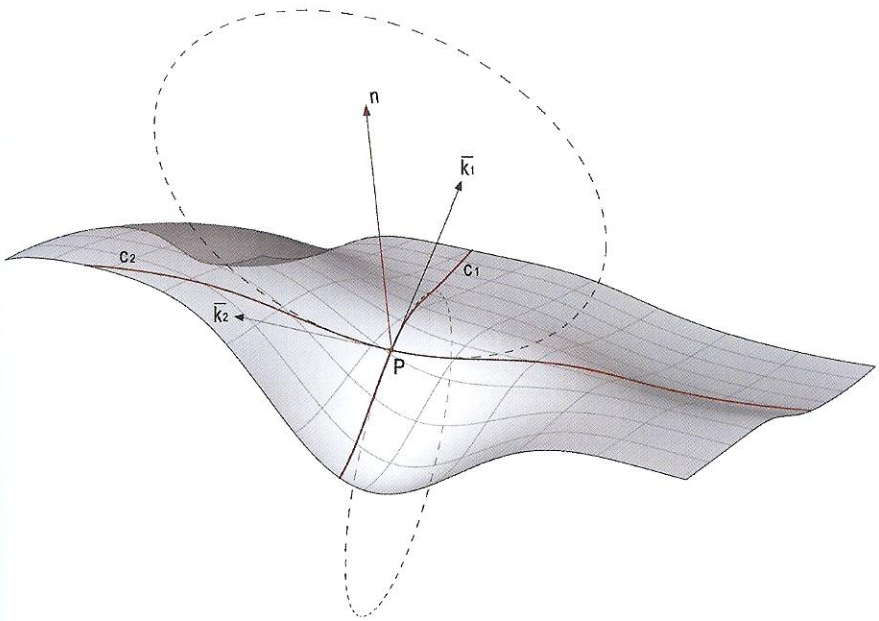


For each curve (*ca, cb,..., ci*) the curvature can be calculated at P as the reciprocal of the radius of the *osculating circle*, resulting in a set of curvature values: ka, kb,..., ki.



Within the set of curvatures (ka, kb,...ki) a *Minimum Principal Curvature* k_1 and a *Maximum Principal Curvature* K_2 can be found. The curve in which curvature K_1 is calculated is named C_1. Similarly curvature K_2 is calculated for curve C_2.

The tangent vector to curve C_1 at P is called the ***Minimum Principal Curvature Direction*** $\overline{k1}$. Similarly, the tangent vector to curve C_2 at P is called the ***Maximum Principal Curvature Direction*** $\overline{k2}$.



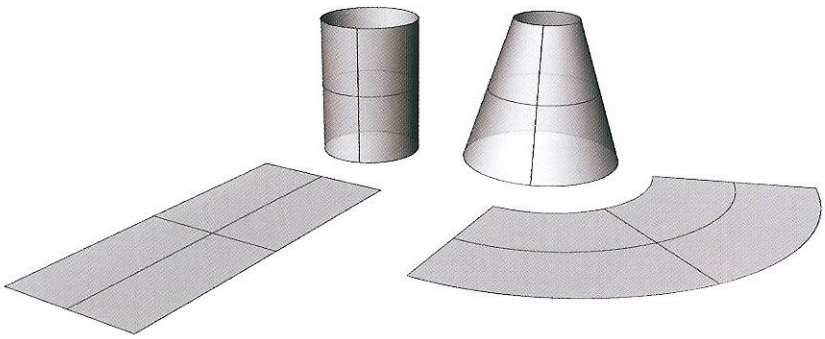
The two primary methods of defining curvature, ***Gaussian Curvature*** and ***Mean Curvature***, can be defined by the equations:

$$G = \text{Gaussian Curvature} = K_1 \cdot K_2 \quad [2]$$

$$M = \text{Mean Curvature} = (K_1+K_2)/2 \quad [3]$$

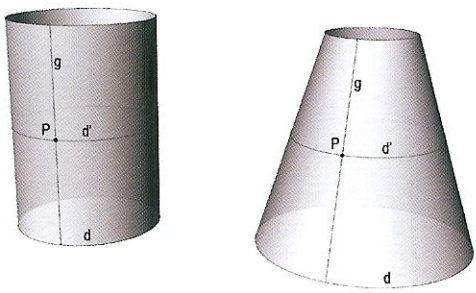
3.8.1 *Gaussian Curvature*

Gaussian and *Mean* curvature are significant to the theory of surfaces. **Surfaces with null *Gaussian Curvature* at all points are *Developable Surfaces***; meaning the surfaces can be flattened onto a plane without deformation. Developable surfaces are practical in manufacturing and fabrication since forms can be constructed by bending flat sheets of flexible materials such as: metal, cardboard, plywood etc. Additionally, these surfaces contain a set of straight lines which also simplify the structure's construction (linear beams).



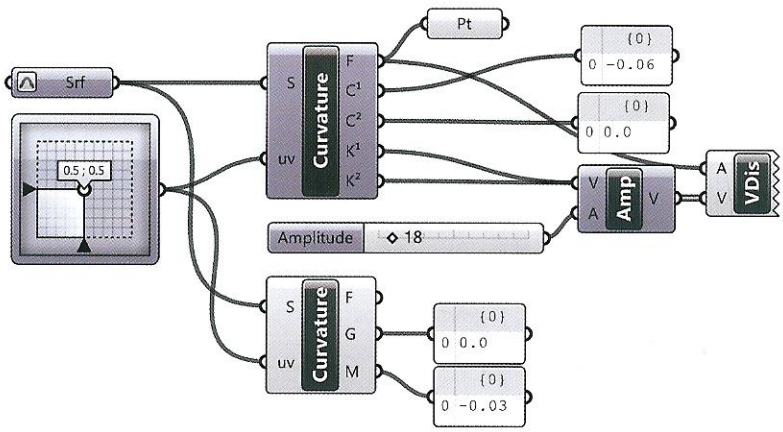
A surface is *developable* if the ***Gaussian Curvature* is equal to zero (G=0)** at each point.
The condition G=0 is satisfied when $\mathbf{k_1=0}$ or $\mathbf{k_2=0}$, since the product of zero is always zero.
For Example, a cylinder's curvature is measured at a point (P):

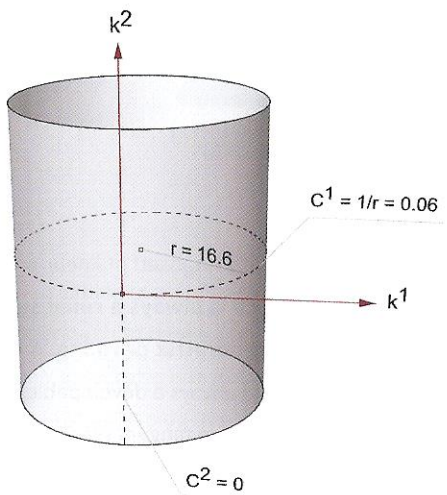
- *Minimum Principal Curvature* $\mathbf{k_1}$ corresponds to the generatrix *g*. Since generatrix *g* is a ruled line $k_1=0$.
- *Maximum Principal Curvature* $\mathbf{k_2}$ corresponds to *d'* which is parallel to the directrix *d*. Since *d'* is a circle its curvature k_2 has a constant value equal to the reciprocal of cylinder's radius.



The product of K_1=0 and K_2=(1/r) is equal to zero. Thus, the curvature of a cylindrical surface is zero at each point meaning *Gaussian Curvature* is null. Similarly, the cone's Gaussian Curvature is null since K_1=0 and K_2=(1/r_n).

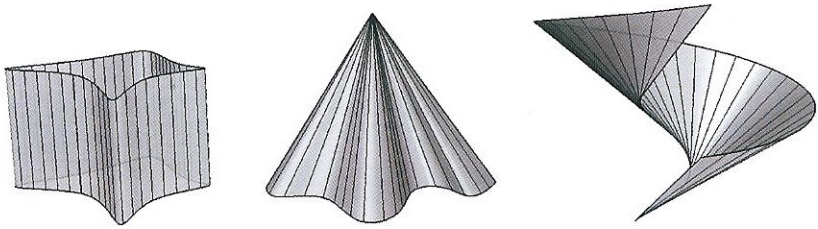
The components ***Principal Curvature*** and ***Surface Curvature*** (Analysis > Surface) calculate curvature for surfaces. The *Principal Curvature* component outputs: K^1 and K^2 measuring the principal curvature directions as vectors, as well as C^1 and C^2 which calculate principal curvature values by definition. The *Surface Curvature* component calculates *Gaussian Curvature* as G-output and the *Mean Curvature* as M-output. For instance, for each point of the cylinder the *Gaussian Curvature* is zero whereas the *Mean Curvature* is the average of C^1 and C^2: resulting in a negative value since C^1 – a circle – has a negative curvature according to the *signed curvature* convention.



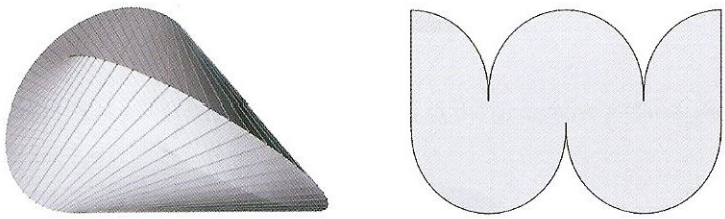


Developable surfaces include:

- ***Planes;***
- ***Cylinders;***
- ***Generalized Cylinders:*** surfaces that have the cross-section of an ellipse, a parabola or, in general, any smooth curve;
- ***Cones;***
- ***Generalized Cones:*** surfaces created by the set of lines passing through a vertex and every point on a smooth curve;
- ***The Oloid.*** A geometric object that was discovered by Paul Schatz in 1929;
- ***Tangent developable surfaces***: surfaces formed by the union of the tangent lines of a free-form curve.



generalized cylinder generalized cone tangent developable

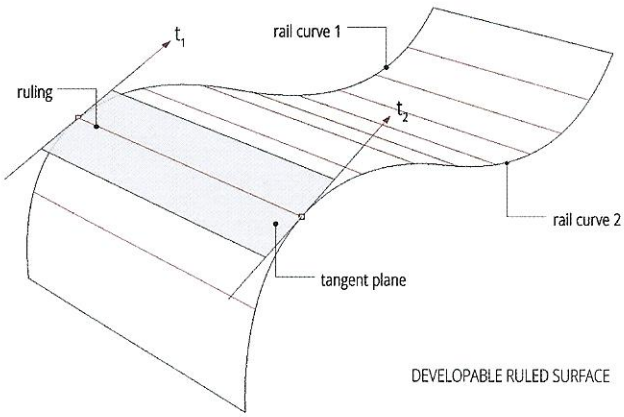


Oloid Developed oloid

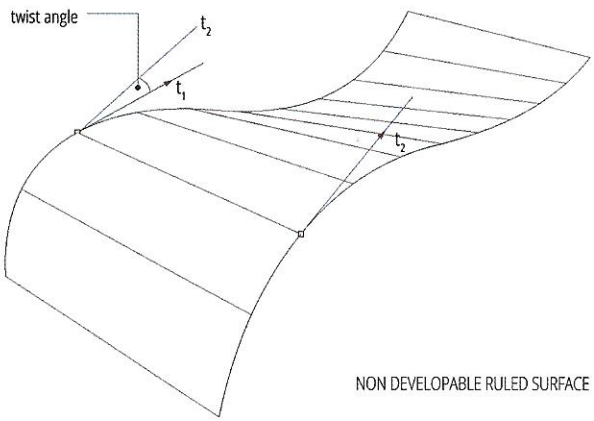
By definition developable surfaces are composed of linear cross sections in one primary direction. In other words, a **developable surface is always a ruled surface**, i.e. a surface generated by the motion of a straight line called the **generatrix** or **ruling**. A developable surface is always a ruled surface; however a ruled surface is not always a developable surface.

A ruled surface is developable when unique tangent planes can be calculated along a ruling. In other words, the **twist angle** (see following images) between tangents at rulings' end-points is null (in this

case a ruling is called a ***torsal ruling***). For example, to create a *developable ruled surface* starting from an arbitrary curve (*rail curve 1*), the geometry of the second curve (*rail curve 2*) is dependent upon outputting a null twist angle.

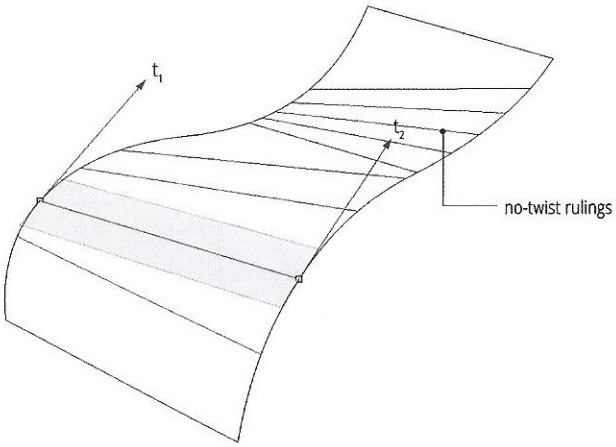


DEVELOPABLE RULED SURFACE



NON DEVELOPABLE RULED SURFACE

A different strategy must be applied when aiming to get a *developable ruled surface* from two arbitrary and fixed rail curves. Of course, it's not enough to divide both curves into equal parts and connect the resulting points through "ruling" lines. In fact, it is not guaranteed that twist angle is null for each ruling. In this case, specific algorithms (available as plug-in software) must be used in order to search for the *developable ruling lines* which have no "twist".



Many architects, designers and artists use developable surfaces in their work. For example, Frank Gehry often studies geometry using physical models composed of flexible sheet materials to developable geometries for his buildings.

FIGURE 3.10
F. Gehry, MARTa Herford Museum, 2005. Image by Oliver S. Wittekind.

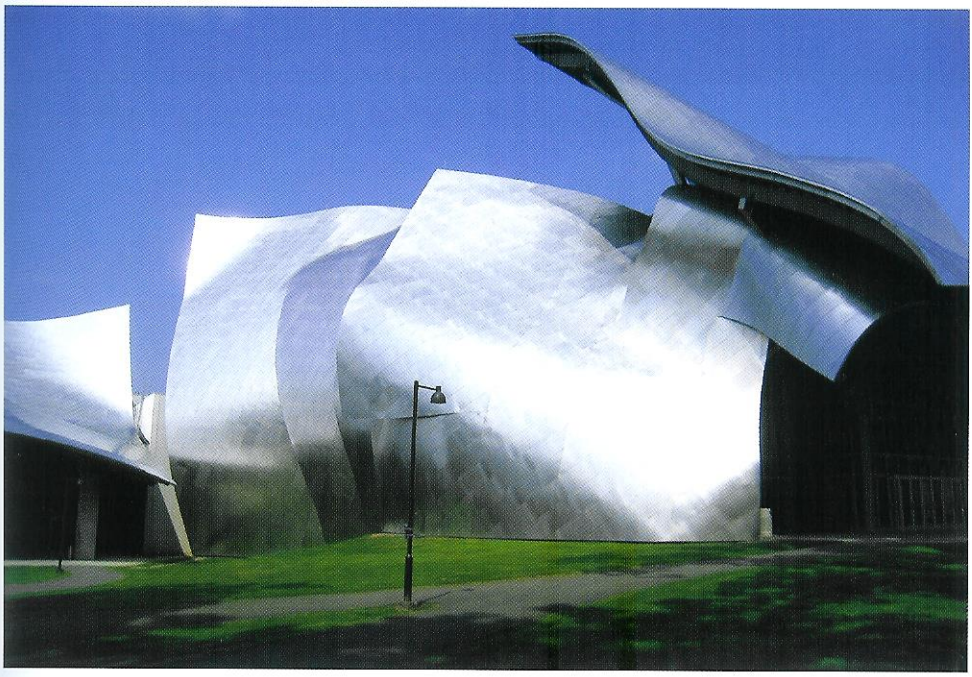
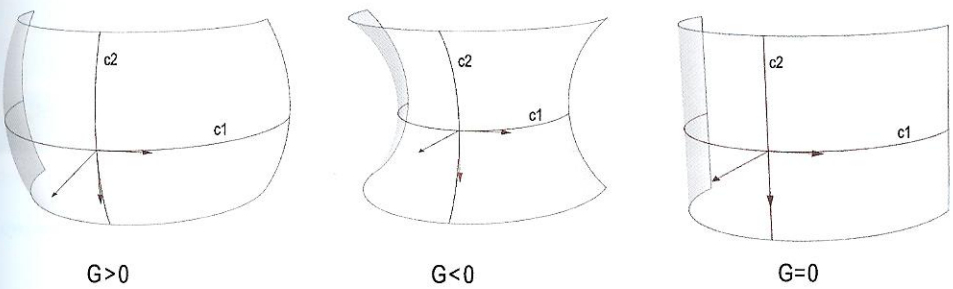


FIGURE 3.11
F. Gehry, Richard B. Fisher Center for the Performing Arts, Bard College, Annandale-on-Hudson, New York, USA. Image by Daderot.

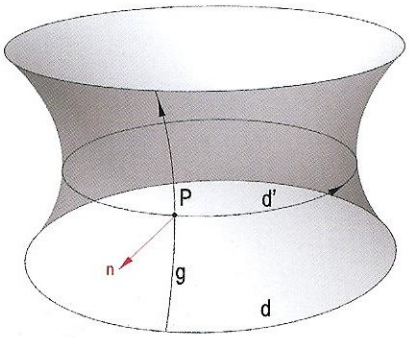
3.8.2 Sign of *Gaussian Curvature*

Gaussian Curvature is null for developable surfaces. *Un-developable surfaces* can have positive or negative *Gaussian Curvature*. *Gaussian Curvature* is positive when K_1 and K_2 are both negative or positive meaning c1 and c2 have their osculating circles lying on the same side of the generatrix. *Gaussian Curvature* is negative when K_1 and K_2 have different signs, meaning c1 and c2 have their osculating circle lying on different sides of the generatrix. If *Gaussian Curvature* is null, c2 is a straight line.



3.8.3 *Mean Curvature*

Surfaces that have a null *Mean Curvature* at all points are called ***Minimal Surfaces***. For Instance, a ***catenoid*** *minimal surface is* formed by the revolution of a catenary-curve *g* about its directrix *d*.



By definition a ***minimal surface*** has **M=0**, meaning $\mathbf{K_1=-K_2}$. In other words, the *Minimum Principal Curvature* and the *Maximum Principal Curvature* have the same value and opposite signs at each point. For instance, the catenoids curve *g* has its osculating circle on the positive side of generatrix and the curve *d'*, a circle, has negative curvature by definition.

A soap film enclosing no volume and formed between boundary constraints is a minimal surface. Soap films have a vanishing *mean curvature* because the surface area is constantly approaching a minimal surface area.

FIGURE 3.12

The mathematical shape "the catenoid" made of a soap film. In every point of its surface the *Mean Curvature* is zero. Image courtesy of soapbubbledk (www.soapbubble.dk).

3.8.4 Strategy: developable test

Developable primitives can be used to isolate developable sub-surfaces, however this method can lead to false results if the geometry is not trimmed from a primitive.

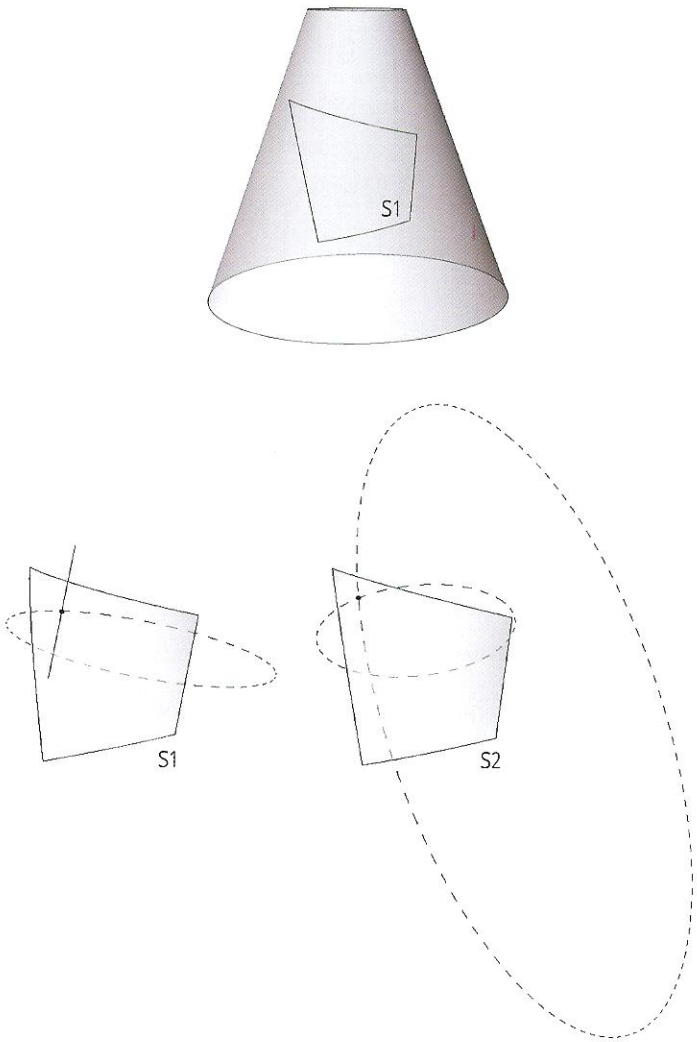
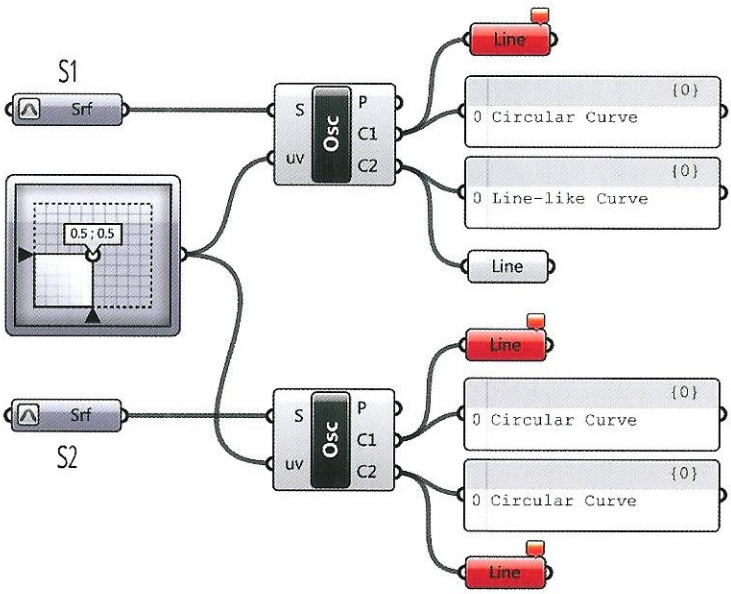


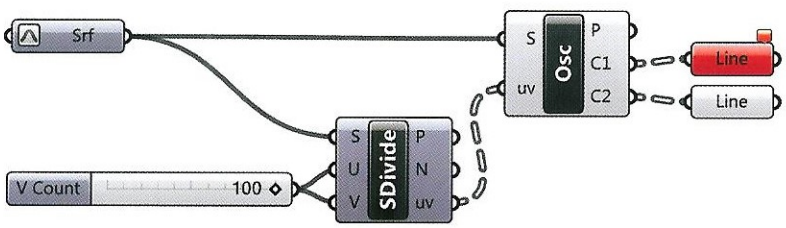
FIGURE 3.13

In case of a developable surface S1, at least one of the osculating circles becomes a straight line and Grasshopper draws a short segment. In a similar (not developable) surface S2 both osculating circles are represented as curves.

To indentify developable surfaces the component *Osculating Circles* (Surface > Analysis) can be used to draw the two *osculating circles* at a generic point P expressed in the LCS. If the surface is developable at least one of the *osculating circles* will be a straight line at each point. In other words, at least one of the *Line* components connected to C1 and C2 will be in "correct" status.

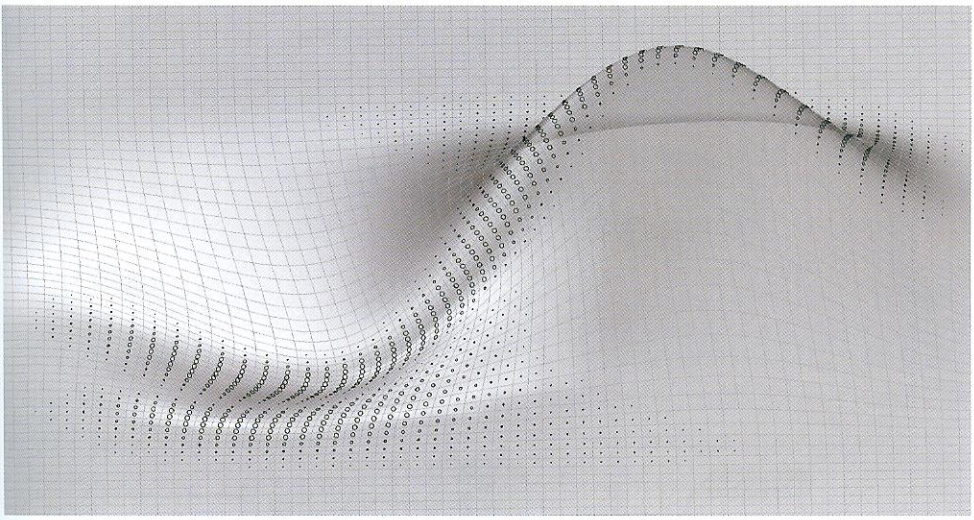


To generalize the method we can generate a grid of points on an input surface by the *Divide Surface* component and calculate the *osculating circles* at resulting points expressed in the LCS.



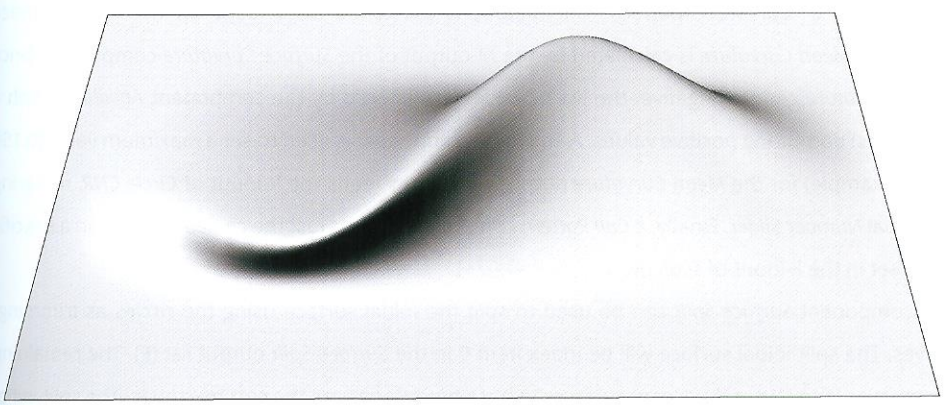
3.8.5 Example: curvature pattern

Mean Curvature analysis can be used to generate complex patterns on surfaces by using measured curvature to inform a pattern.

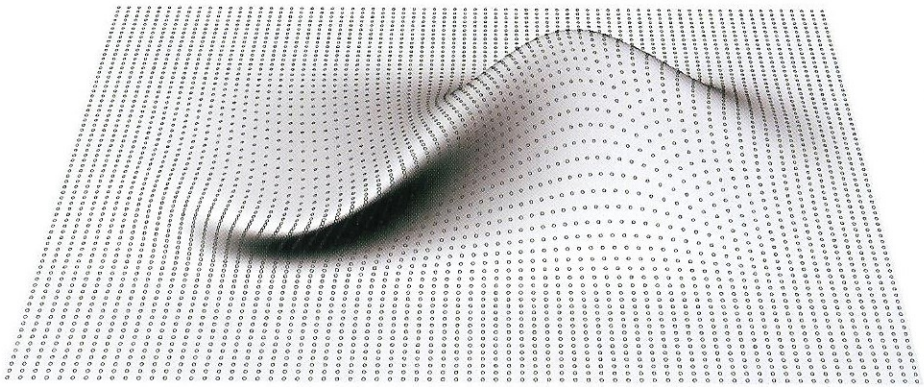


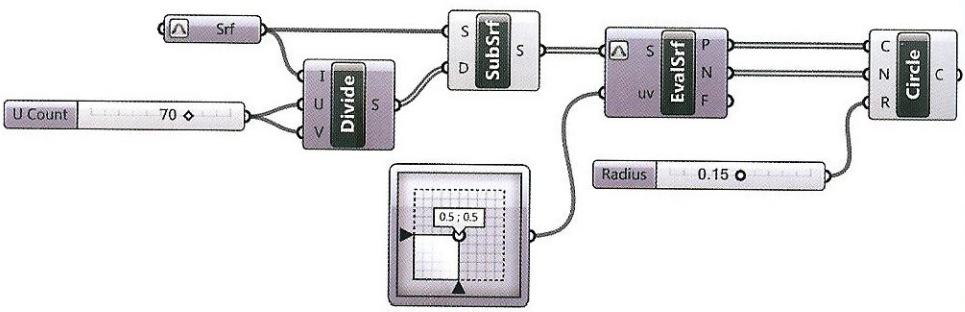
A surface manipulated by adjusting points using the *Gumball* tool, is set from Rhino. In order to generate relevant changes in curvature, surface ridges are created.

To create the pattern the set surface is reparameterized, and divided into a set of N sub-surfaces using the *Divide Domain* and *Isotrim-SubSrf* components. The *Evaluate Surface* is used in conjunction with a *MD Slider* set to (0,5;0,5), to create a point approximating the center of each surface.



To generate the pattern of circles the C-input and the N-input of a *Circle CNR* component are connected to the P-output and N-output of *Evaluate Surface*, respectively. The radius of circles (R) is set to an arbitrary value through a slider.

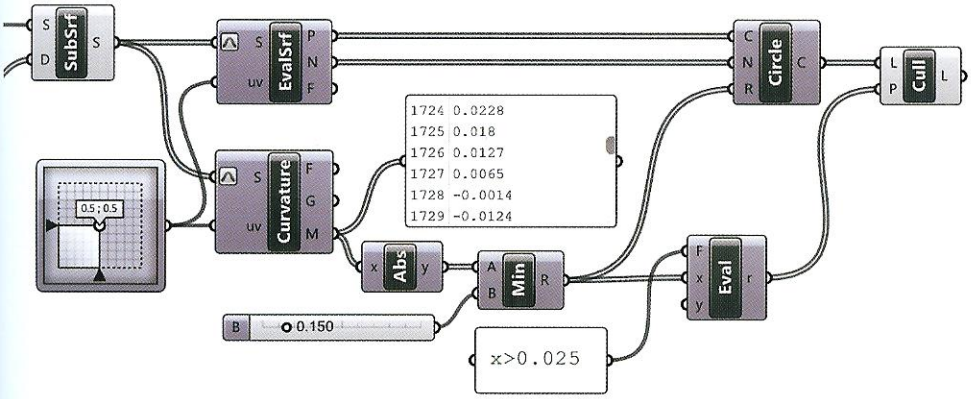




To generate a "curvature pattern" the R-input is set to be dependent on the mean surface curvature. *Mean Curvature* is calculated by the M-output of the *Surface Curvature* component. Since *Mean Curvature* can be negative, the list of values are filtered by the component *Absolute* (Math > Operators) outputting positive values. A *Minimum* component is used to set a maximum value (0.150 in the example) for the *Mean Curvature* output which now feeds the R-input of *Circle CNR*, replacing the initial *Number Slider*. Finally, a *Cull Pattern* component selects just the circles larger than a specific value set in the F-input of *Evaluate* (we set x>0.025).

The component *Surface Split* can be used to split the initial surface using the circles as trimming-curves. The split initial surface will be index item 0 in the *Surface Split* output list (F). The remaining geometries (1 to N) are the "scrap surfaces" whose boundaries are the cutting curves. The split initial

surface can be visualized by using the *List Item* component set to 0.



In order to visualize just the "scraps" index 0 – the split initial surface – can be removed from the list using the *Cull Index* component.