

SurfaceWorks 5.0 Reference: What's New

<i>SURFACEWORKS 5.0 REFERENCE: WHAT'S NEW</i>	1
ADVANCED MODELING	4
Points	4
Blend Point	4
Intersection Point.....	6
Tabulated Point.....	7
Tangent Magnet.....	8
XYZ Magnet.....	10
Proximity Points	11
Proximity Bead, Ring, Magnet	11
Curves and Snakes	14
B-Fitted Curve	14
Conic	15
Expanded Curve	18
NURBS Curve.....	19
Radius Arc	20
Tabulated Curve	22
X-Curve.....	24
Arc Snake	26
Foil Snake.....	27
Geodesic Snake	29
NURBS Snake.....	32
PolySnake.....	33
NUBFitCurve, NUBFitSnake, NUBFitSurf entities	34
NUBFitCurve	34
NUBFitSnake	36
NUBFitSurf	36
Tools and NUBFit Surf	37
Surfaces	38
Foil Lofted Surfaces	38
B-Fit Surface	41
B Surface	43
NURBS Surface	44
Projected Surface.....	47
Relative Surface	48
Tabulated Surface.....	50
X-Spline Lofted Surface.....	52
Parametric Solids	55
Additional Solid Attributes.....	55
divisions: u, v, and w divisions and subdivisions	55
Orientation.....	56

Using Solids	56
Some applications.....	56
Discussion of tank and grid examples	57
Solid Entities	57
Boundary Solid.....	57
BlockSolid	58
B-spline Lofted Solid	59
Bsolid	60
CopySolid.....	61
Ruled Solid.....	61
Triangle Mesh Entity Types	63
Triangle Mesh basics.....	63
Recursive subdivision.....	64
Locations on a Triangle Mesh	64
Common data	64
Triangle Mesh.....	65
Light Triangle Mesh.....	65
Surface Triangle Mesh	66
Copy TriMesh.....	67
PolyTriMesh.....	67
Triangle Mesh Magnet	68
Triangle Mesh Projected Magnet	68
Triangle Mesh Ring.....	69
Snakes on Triangle Meshes	69
Triangle Mesh B-spline Snake	70
Triangle Mesh Edge Snake.....	71
Triangle Mesh Intersection Snake	71
Triangle Mesh Projected Snake.....	72
Triangle Mesh SubSnake.....	73
Formulas	74
Constants, variables and formulas	74
Variables.....	75
Formulas.....	75
Basic syntax.....	75
Expression	75
Unit consistency	77
Evaluation logic.....	77
Complete syntax	78
Variables and formulas as parents	78



P.O. Box 684 / 54 Herrick Rd.
Southwest Harbor, Maine 04679 U.S.A.
voice 207-244-4100
fax 207-244-4171
email support@aerohydro.com
website www.aerohydro.com

© 2005 AeroHydro, Inc.

Advanced Modeling

Points

Blend Point

Characteristic data *point1 ... pointN* = component point names
weight1 ... weightN = corresponding weights
type = 0, 1, or 2

Description The Blended Point is a weighted sum of one or more points. Each coordinate of the blended point is the same weighted sum of the component point coordinates:

$$X = w_1 X_1 + w_2 X_2 + \dots + w_N X_N$$

$$Y = w_1 Y_1 + w_2 Y_2 + \dots + w_N Y_N$$

$$Z = w_1 Z_1 + w_2 Z_2 + \dots + w_N Z_N$$

type specifies how the *weight* values are used in the formulas:

type = 0 weights are used without modification $w_i = weight_i$

type = 1 weights sum to 1(forced) $w_i = weight_i$ EXCEPT $weight_N$ is ignored -- it is replaced by $1 - (\text{sum of the other } weights)$

type = 2 each weight is divided by the sum of all weights $w_i = weight_i / \text{sum}$

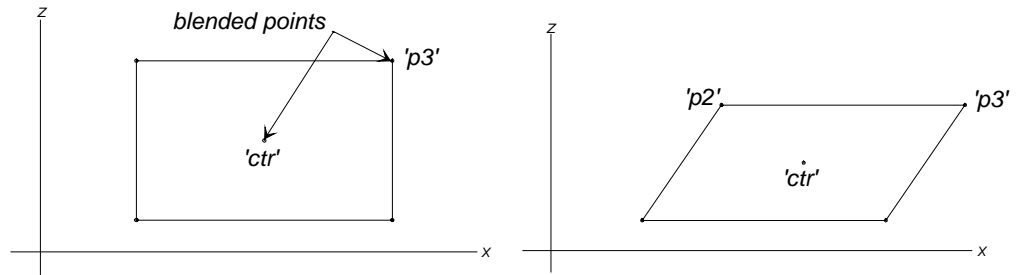
When weights sum to 1, it is generally easier to predict the result of the blend; when weights do *not* sum to one, the blend is more complex. For most applications, you probably want the weights to sum to 1. Thus, to construct the midpoint between two points, you could make a type-1 BlendPoint with weights of 0.5 and 0.5. The weights used in the formulas would be $w_1 = 0.5$, $w_2 = 1 - 0.5 = 0.5$ (in a type-1 BlendPoint $weight_N$ is ignored and 1 minus the sum of the other weights is used). You could also use a type-1 BlendPoint to construct the mirror point of *point1* across *point2*. You would use weights -1.0 and +2.0; the weights used in the formulas would then be $w_1 = -1.0$, $w_2 = 1 - (-1.0) = 2.0$.

Example 1

Blend Point2.ms2

This model has two, type-1 (weights sum to 1) BlendPoints: 'p3' and 'ctr'.

'p3' (cyan) is used to make the fourth corner of the parallelogram; its component points are 'p0', 'p1', and 'p2' (weights -1, +1, +1). Since 'p3' is a type-1 BlendPoint, the weights used in the formulas are: $w_1 = -1$, $w_2 = +1$, and $w_N = 1 - (-1 + 1) = 1$. You can drag RelPoints 'p1' and 'p2' around as you wish, but the figure formed by BCurve 'c0' is durably parallel-sided.



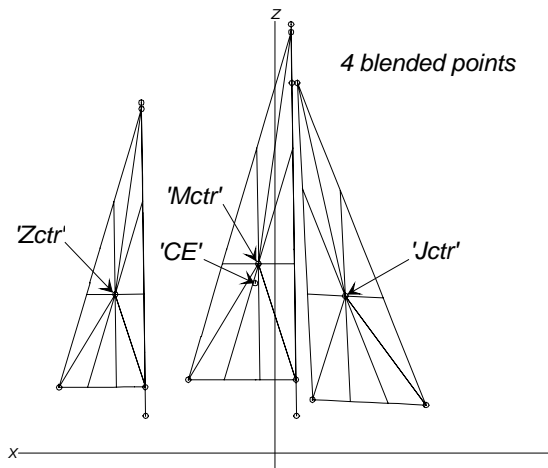
BLENDPT2.MS2 at Lat 0, Lon -90 orthographic: initial model (left); with 'p2' moved (right).

The second BlendPoint 'ctr' stays at the center of the figure; it is the average of the four corners 'p0', 'p1', 'p2', and 'p3' (weights .25, .25, .25, .25). Since 'ctr' is a type-1 BlendPoint, the weights used in the formulas are: $w_1 = .25$, $w_2 = .25$, $w_3 = .25$, and $w_N = 1 - (.25 + .25 + .25) = .25$.

Example 2

Blend Point3.ms2

This example uses three type-1 BlendPoints and one type-2 BlendPoint to make the standard yacht designer's "center of effort" calculation.



BLENDPT3.MS2 at Lat 0, Lon 90 orthographic.

The "ketch" rig has three triangular sails: 'jib', 'main' and 'mizzen'. Each sail is made from its 3 corner points. The center of each sail is located by a type-1 BlendPoint, equally weighting the 3 corners. (This is a valid way to locate the centroid of a triangle.) The boundary of each sail is a type-1 BCurve, and the surface of the sail is a RuledSurf between the centroid and the boundary. The "Center of Effort" (CE, the supposed aerodynamic center of sail force) is defined as the weighted sum of the individual sail centroids, each weighted by the area of that sail. The sail areas are obtained from Calcs/ Mass Properties:

Sail	Area (weights for the BlendPoint)	Area/sum of areas (w_1, w_2, w_3 values used in the formulas)
'Jib'	275 sq.ft.	.3681
'Main'	288 sq.ft.	.3855
'Mizzen'	184 sq.ft.	.2463

Total 747 sq.ft.

A type-2 BlendPoint 'CE' makes the center of effort calculation; the three sail areas are entered as the weights. Since 'CE' is type-2, the weights used in the formulas are $w_1 = .3681$, $w_2 = .3855$, and $w_3 = .2463$ (as shown in the table above). You can read the absolute coordinates of the center of effort 'CE' from the Edit/Attributes box or the status line.

[Variables and Formulas provide a way to represent this complete relationship durably. The critical element is the AREA function.]

Other Examples

See use of BlendPoint in *Entity Description* for ProcPtSurf - Example 1.

Intersection Point

An Intersection Point is located at the common intersection point of three planes and/or surfaces. (Actually, it solves a considerably more general intersection problem, as it allows offsets to be specified from any combination of the 3 planes and/or surfaces.)

Characteristic Data

Magnet/surface/plane/frame1

Magnet/surface/plane/frame2

Magnet/surface/plane/frame3

Offset1, Offset2, Offset3

Description

If a magnet is specified in place of a surface, the magnet location is used as a starting point in the search for an intersection point.

As usual, a frame can serve as a plane; in this case, the plane is the x-y plane of the frame.

When the intersector is a surface, the offset is positive in the direction of the positive normal (like an OffsetSurf). Bear in mind that such an offset will be reversed if the surface's orientation attribute is reversed.

Planes have an orientation, too, though this has not so far been made a visible characteristic of a plane, so you need to follow some rules if you want to make use of an offset from a plane:

*X=0, *Y=0, *Z=0 -- A positive offset is in the positive coordinate direction.

Plane2 -- A positive offset is in the direction from Point1 toward Point2

Plane3 -- A right-hand rule prevails; if you made a Frame3 from the same three points, picked in the same order, positive offset is in the direction of the positive z-axis.

OffsetPlane -- Orientation is the same as that of the parent plane.

Frame -- A positive offset is in the direction of the positive z-axis.

Associated errors

228 -- IntPoint has duplicate Parents; no solution.

In general, if the same object is used for two of the Parents, there is no unique solution for the intersection.

229 -- IntPoint: Solution failed, equations are singular.

This usually indicates a badly posed intersection, for example: two of the planes are parallel, or a surface coincides with a plane.

232 -- IntPoint: Failed to converge.

This can also result from a badly posed intersection. Bad starting locations are another cause; use a magnet near the intersection, in place of a surface.

Sample files

Intersection Point1.ms2. This has 4 IntPoints;:
ip0 is at the intersection of 3 planes.
ip1 is at the intersection of 2 planes and 1 surface.
ip2 is at the intersection of 1 plane and 2 surfaces.
ip3 is at the intersection of 3 surfaces.

Tabulated Point

Characteristic data	<i>filename</i> = filename of .3DA or .PAT file <i>point number</i> = number of points in file <i>frame/point</i> = name of a frame or point object or '*' for the default
Description	<i>filename</i> is the filename of a .3DA (or .PAT) file. If no extension is given, SurfaceWorks assumes .3DA. A .3DA file (see sample contents in example below) consists of a series of records (file lines) each having four numbers: a <i>pen</i> number and X, Y, Z coordinates. TabPoint ignores the pen value, and just counts each record as an individual point. The file must have at least as many points in it as the <i>point no.</i> specified, or a geometry error occurs. <i>point no.</i> is the sequence number of the point to be taken from the .3DA file: 1 for the first, 2 for the second, etc. <i>frame/point</i> is the name of a frame or point object or '*' for the default: When <i>frame/point</i> is a <u>point</u> , it specifies an insert origin, and <i>point's</i> coordinates are added to the TabPoint coordinates. This results in a shift (but no rotation). When <i>frame/point</i> is a <u>frame</u> , the file coordinates are interpreted as frame coordinates x,y,z. This generally results in a shift plus a rotation. When <i>frame/point</i> is <u>the default</u> '*', there is no shift or rotation.

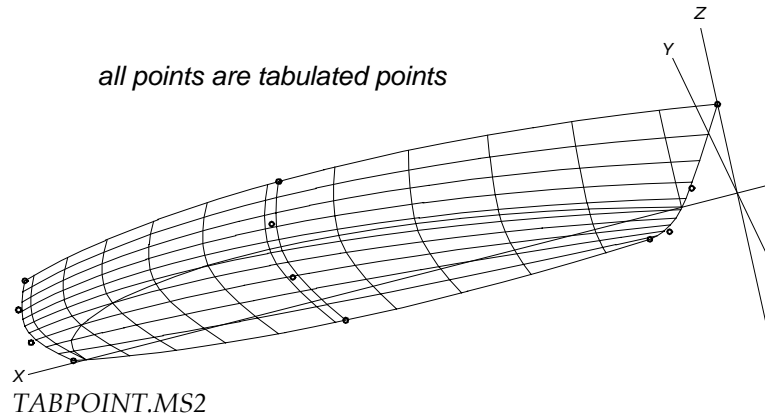
Example

tabpoint.ms2 (needs demopts.3da)

The model generated by this file is identical to the DEMO model. If you use Edit/ Model File, you can see the difference is that where the DEMO.MS2 has all of its control points "hard-wired," TABPOINT.MS2 gets its control points by reading the file DEMOPTS.3DA (see contents

below). Thus, the shape of the TABPOINT model can be controlled in many ways by changing the contents of DEMOPTS.3DA.

The 4 TabPoints defining the bow master curve are named 'P11' - 'P14' (from top to bottom), those for the middle master curve 'P21' - 'P24' (top to bottom), those for the stern master curve 'P31' - 'P34' (also top to bottom). For all the TabPoints, *filename* is DEMOPTS, and *frame/point* is the default '*' which produces no shift or rotation of the points. *point no.* for 'P11' is 1, for 'P12' is 2, for 'P13' is 3, for 'P14' is 4, for 'P21' is 5, for 'P22' is 6, etc.



Contents of DEMOPTS.3DA

0	0.000	0.000	3.600
1	1.367	0.000	0.602
1	2.324	0.000	-0.800
1	3.000	0.000	-0.900
0	15.000	4.815	2.560
1	15.000	5.046	0.628
1	15.000	3.603	-0.870
1	15.000	0.000	-1.175
0	30.000	3.500	2.760
1	30.000	3.500	1.320
1	30.000	2.500	0.120
1	30.000	0.000	0.120

See also

TabCurve, TabSurf, WireFrame

Tangent Magnet

The TanMagnet (tangent magnet) entity type constructs a point on a surface along the direction tangent to a snake. It is highly analogous to a TanPoint in 3-D (a point along the 3-D tangent to a curve, at a place on the curve specified by a bead). In the TanMagnet case, the tangency direction is really in the 2-D *u,v*-parameter space of the surface.

TanMagnet is a key to constructing snakes that are tangent to each other in both *u,v* parameter space and in 3-D.

A TanMagnet is a draggable point. The tangent direction is fixed by the ring and snake parents, so only the u,v -offset (a single degree of freedom) is controlled by dragging or nudging.

Characteristic data

Ring -- This designates both the snake to be tangent to, and the point along the snake where the tangent direction is taken. The host surface of the TanMagnet is the host surface of the ring (and therefore of the snake).

u,v -offset -- a distance in u,v parameter space from the ring parent. This is a signed distance, positive along the positive tangent direction at ring.

Construction

The program evaluates the ring parent, obtaining the host snake and a t parameter value. Then it makes a level-1 evaluation of the snake at this parameter value, obtaining the host surface, a u,v location for the ring, and the first derivative of the u,v location. It normalizes the first derivative to obtain a unit vector U in u,v space. It multiplies U by the u,v -offset and adds the resulting 2D vector to the ring's u,v coordinates, to obtain the u,v coordinates of the TanMagnet.

Error conditions

(1) The ring can fail to evaluate, or the snake can fail to evaluate at this t position (though this seems unlikely, if the ring is valid), or the surface can fail to evaluate at this u,v location. These failures produce error 284 (parent failed).

(2) The first derivative of the snake can be zero, so the normalization fails and a tangent direction can't be established. This produces a new error, 473.

Sample files

Tangent Magnet.ms2

AbsRing 'r0' (bright white) is on BSnake 'n0' (bright magenta). 'r0' is used as parent for TanMagnet 'm0' (bright red); the u,v -offset is 0.200. 'r0', 'm0', and two additional AbsMagnets 'm1' and 'm2' are used as control magnets for BSnake 'n1' (yellow). Use of the TanMagnet as the second control point of 'n1' has made 'n1' durably tangent to 'n0' at 'r0'.

SubSnake 'n2' (bright green) is the portion of 'n0' from $t=0$ to 'r0'. SubSnake 'n2' and BSnake 'n1' are assembled into PolySnake2 'n3' (bright

blue). Breakpoints command reveals only 1 degree-2 breakpoint for 'n3', at the location of 'r0'. This shows that an accurate tangency has been achieved between its component snakes 'n1' and 'n2'.

On layer 1, an image of the u,v parameter space has been constructed using a 4-point BSurf 's0' as the parameter square, and CopySnakes 'N0' and 'N1', CopyRing 'R0', and CopyMagnets 'M1' and 'M2' as images of the corresponding 3-D objects.

XYZ Magnet

Characteristic Data

Type (1 to 3)

Surface or magnet

Frame

Point

Location1, Location 2

Description

Type determines whether you are making the YZ, ZX, or XY variety

Type-1 locates the magnet by its Y and Z locations

Type-2 locates the magnet by its Z and X locations

Type-3 locates the magnet by its X and Y locations.

Note these designations are based on an "odd-man-out" rule:

Type-1 is the one that *doesn't* use X

Type-2 is the one that *doesn't* use Y

Type-3 is the one that *doesn't* use Z

Surface or magnet designates the host surface for the XYZMagnet. If a magnet is used here, it also provides a starting location for the iterative solution. Unless there is clearly only one solution (considering also the extension of the surface beyond the 0 to 1 nominal parameter range), it is always preferable to specify a magnet that is somewhere in the neighborhood of the intended intersection. If you just specify the surface, the $u = 0.5, v = 0.5$ position on the surface will be used as a starting value for the XYZMagnet. If the true intersection is far from 0.5, 0.5, the XYZMagnet may fail to converge, or may go to an unexpected location on the surface extension.

Frame specifies the coordinate frame in which to measure X, Y or Z. The default

frame '*' is the global coordinate system, in which case the coordinates used are just

the global X, Y or Z. If you choose to measure offsets or location in another *Frame* not aligned with the global coordinates, *Location1* and *Location2* will be frame coordinates, x, y or z.

Point is an optional reference point. If you choose the default point '*', the *Locations*

will be measured from the origin of *Frame*. This creates the "absolute" variety of

XYZMagnet. If you choose another point for *Point*, the *Locations* will be relative to

that *Point*. This makes the "relative" variety of XYZMagnet.

Location1, *Location2* are the two coordinate offsets.

Example

Suppose you need to locate a point 'm1' on the deck surface, exactly 3.20 ft. from centerplane and 10.46 ft. aft of the forward reference station. The most direct way to do that now is an XYZMagnet. Since X and Y are specified, the type has to be 3. Use '*' for both *Frame* and *Point* (coordinates measured in the global coordinate system), and 10.46 and 3.20 for the *Locations*.

Suppose you need another point 'm2' on the deck, exactly 2.25 ft. aft of 'm1' and the same distance from the centerplane. The simplest way to do this is an XYZMagnet, again type-3 since X and Y are specified, with *Frame* = '*', *Point* = 'm1', and *Locations* of 2.25 and 0.

Sample file: XYZ MAGNET.MS2

Proximity Points

Proximity Bead, Ring, Magnet

Characteristic Data

ProxBead

type (kind), 0 | 1 = minimum/maximum signed distance

type (kind), 0 | 1 = unconstrained/constrained

bead/curve

mirror/surface

ProxRing

type (kind), 0 | 1 = minimum/maximum signed distance

type (kind), 0 | 1 = unconstrained/constrained

ring/snake

mirror/surface

ProxMagnet

type (kind), 0 | 1 = minimum/maximum signed distance

type (kind), 0 | 1 = unconstrained/constrained

magnet/surface

mirror/surface

Description

The ProxBead, ProxRing and ProxMagnet entity types solve the following problems:

ProxBead (ProxRing):

(Type 0) Find the point on a curve (snake) that has the minimum signed distance from a mirror/surface.

(Type 1) Find the point on a curve (snake) that has the maximum signed distance from a mirror/surface.

ProxMagnet

(Type 0) Find the point on a surface that has the minimum signed distance from a mirror/surface.

(Type 1) Find the point on a surface that has the maximum signed distance from a mirror/surface.

"Signed distance" is a measure of the position of a point in relation to a mirror or surface, that takes the orientation of the mirror or surface into account. In the case of a point or line mirror, signed distance is always positive, and increases outward from the mirror. In the case of a plane mirror, signed distance is zero on the plane, positive on one side and negative on the other. In the case of a surface mirror, signed distance is zero on the surface, positive in the direction of the positive normal and negative in the other direction.

Constrained/ Unconstrained

There is an issue of whether the ProxBead is constrained to the interval $[0,1]$, or is free to go onto the extensions of the curve; similarly for a ProxMagnet, whether it can use the extension of the surface. Both alternatives appear useful in different circumstances, so both are provided by means of a second *type*: 0 for unconstrained, 1 for constrained. If a bead (ring, magnet) is given for the bead/curve

(ring/snake, magnet/surface), SurfaceWorks uses it for the starting location, and searches for a nearby local minimum or maximum. Otherwise, SurfaceWorks starts with a tabulation of the host curve (snake, surface) and identifies the closest/farthest point in the table as a starting location for the search.

Associated errors

(1) Failed to converge.

(2) No solution exists. This will be a fairly common occurrence; say, from choosing the wrong type. On a sailboat hull (unconstrained), the maximum Y point will exist and be well defined, but in general there is no minimum Y -- the extended surface goes on to minus infinity. The optimization will be trying to run to minus infinity, and we have to detect that and stop it before the numbers get outrageous.

Sample files

ProxBead1.ms2 This model contains 4 ProxBeads hosted on a circle. 'e1' is at the closest point on the circle to Point 'p3', and 'e2' is the point farthest away from 'p3'. 'e1' and 'e2' both use the circle as parent, so do not specify a starting point. 'e3' is the point closest to Point 'p4', and 'e4' is the point farthest away from Point 'p5'. 'e3' and 'e4' both use AbsBead 'e0' as a starting point.

ProxBead3.ms2 This model contains 3 ProxBeads hosted on a semicircle, each constrained to the 0-to-1 parameter interval (their second type is 1). 'e1' is the point on 'c0' that is closest to Point 'p3'; 'e2' is the point that is farthest away from 'p3'. 'e3' is also at a local maximum distance from 'p3'; its parent is AbsBead 'e0', providing a starting position for the search, which therefore finds the local rather than global maximum. If 'e0' is moved to the opposite side of 'e1', then 'e3' jumps to the t=0 end of 'c0'.

ProxRing1.ms2 This model has 2 ProxRings hosted on a BSnake. Each is at the minimum distance from a magnet. 'r1' uses the snake as parent; 'r2' uses AbsRing 'r0', which gives it a starting location for the search.

ProxMagnet1.ms2 This model has a single ProxMagnet 'm1', used to locate the maximum beam location on a boat hull. The mirror is the predefined plane *Y=0, whose positive orientation is toward positive Y. Therefore the ProxMagnet's first type is 1, seeking a maximum signed distance.

MaxBeam.ms2 In this model a construction, using a Proximity Ring, is set up to find a curve showing the maximum beam on a vessel's topsides. The Proximity Ring 'Prox1' draws out a Procedural Snake caused by the action of moving its parent snake from sheer to chine. The Parents of the Procedural Snake are the Proximity Ring and the Ring 'MovingPoint', which is the parent of the ProxRing's host UV Snake.

Curves and Snakes

B-Fitted Curve

Characteristic data	<i>type</i> = B-spline type <i>nc-specifier</i> = integer; specifies choices for number of control points to use for fit (0 if free) <i>log-tolerance</i> = log (base 10) of tolerance <i>curve</i> = basis curve
Description	<p>A BFitCurve is a uniform B-spline curve, least-squares fitted to the tabulated data of its basis curve. Its principal function is for exporting models to NURBS-based systems. Creating BFitCurves before you File/Export 3D/ IGES gives you more control over the accuracy of the exported model:</p> <ul style="list-style-type: none">• you can <i>see</i> the NURBS approximation results right in SurfaceWorks, prior to exporting IGES• you can specify type and number of control points for NURBS curve approximation• you can specify different tolerances for different curves <p><i>log-tolerance</i> is the tolerance specified with a base-10 logarithm, e.g. -3 for tolerance of 10^{-3}. This notation permits a wide dynamic range. (Tolerance itself is measured as the RMS (root-mean-squared) deviation of the fitted curve from tabulated points of the basis curve.)</p> <p><i>type</i> is the type of B-spline : 1 = linear, 2 = quadratic, 3 = cubic, etc.</p> <p><i>nc-specifier</i> is an integer value which specifies choices for the number of control points to be used. There are two choices:</p> <p>When you specify a positive value (it must be greater than the corresponding <i>type</i>), the program will use exactly this number of control points for the fit.</p> <p>When you specify zero, the program will start at <i>type</i> +1 control points and cycle <i>nc</i> upward until it either achieves the fit within the tolerance or reaches the limit of 128 control points.</p> <p>If the BFitCurve does not meet the specified tolerance, the Error window will pop up and let you know the problem. Double-Clicking the Entity</p>

gives you quick access to the Property Manager, where you can try a different set of values.

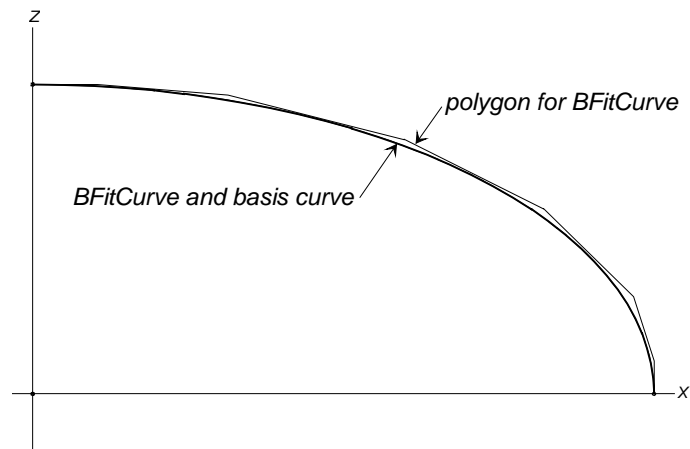
When the BFitCurve is displayed, you can see where the control points are by turning on the "polygon" (*visibility* = polygon) for the curve – the control points are located where the polygon lines meet

If you would like SurfaceWorks to report information about the quality of fit, select the BFitCurve and then choose Tools/ Clearance.

Example

bfitcv1.ms2

This example is a cubic (type-3) B-spline curve fitted to one quadrant of an ellipse basis curve. Since *nc-specifier* is 0, the program incremented the number of control points until the tolerance ($1E-4 = .0001$) was met. The fit was accomplished with 8 control points.



The deviation of the fitted curve from the basis curve, as provided by Tools/ Clearance, is: rms = 6.80e-005.

See also

Fitted Surface

Conic

Characteristic data

center = point to locate center point of complete conic section (P)

point2 = point to locate end of primary axis (P2)

point3 = point to locate end of secondary axis (P3)

s0 = conic parameter (degrees) for the start ($t = 0$)

s1 = conic parameter (degrees) for the end ($t = 1$)

type = 1, 2, 3, or 4

type-1 = ellipse

type-2 = hyperbola, positive branch

type-3 = hyperbola, negative branch

type-4 = catenary

Description

All types of conics are constructed by specifying the center (*center*), the ends of the semimajor and semiminor axes (*point2* and *point3*), and two parameter values (*s0* and *s1*). The conic curve always lies in the plane of

the three points. The end of the *primary axis* is at *point2* (P2). A temporary point Q is created for the end of the *secondary axis* by moving from P2 parallel to the primary axis as required to make the secondary axis perpendicular to the primary axis. Thus the user is freed from locating the three points such that P2-P-P3 is a right angle. The length of the secondary axis is the distance of P3 from the primary axis, also the distance from P to Q.

For a precise description of the conic curves we define the three vectors using bold face letters:

- a** is the vector from P to P2 (primary axis)
- b** is the vector from P to Q (secondary axis)
- c** is the vector from the origin to P

a, b, and c are their magnitudes, respectively.

Ellipse (type-1)

For an ellipse, the primary axis can be either the semimajor or semiminor axis, depending on the relative distances of P2 and Q from P. The full ellipse passes through points P2 (at s = 0) and Q (at s = 90 deg.). The equation for the ellipse (type-1) is:

$$\mathbf{x}(t) = \mathbf{a} \cos s + \mathbf{b} \sin s + \mathbf{c}$$

$$s = s_0 * (1 - t) + s_1 * t$$

The ellipse foci are at $\mathbf{c} \pm \mathbf{a} \sqrt{1 - b^2 / a^2}$.

Hyperbola (type-2 and type-3)

For a hyperbola, the primary axis is always the semimajor axis. The full hyperbola passes through point P2 (at s = 0).

The equation for the positive hyperbola (type-2) is:

$$\mathbf{x}(t) = \mathbf{a} \cosh s + \mathbf{b} \sinh s + \mathbf{c}$$

$$s = s_0 * (1 - t) + s_1 * t$$

The equation for the negative hyperbola branch (type-3) is:

$$\mathbf{x}(t) = -\mathbf{a} \cosh s - \mathbf{b} \sinh s + \mathbf{c}$$

$$s = s_0 * (1 - t) + s_1 * t$$

The hyperbola foci are at $\mathbf{c} \pm \mathbf{a} \sqrt{1 + b^2 / a^2}$. The asymptotes are the two lines:

$$\mathbf{x} = \mathbf{c} + r * (\mathbf{a} \pm \mathbf{b})$$

where r is a real parameter.

Catenary (type-4)

Although not actually a conic section, the catenary is a related analytic curve which is easily supported in the same context. Its equation is:

$$\mathbf{x}(t) = \mathbf{a} \cosh s + \mathbf{b} s + \mathbf{c}$$

$$s = s_0 * (1 - t) + s_1 * t$$

(s in radians). If s0 and s1 are fairly small, the curve is very similar in shape to the positive hyperbola.

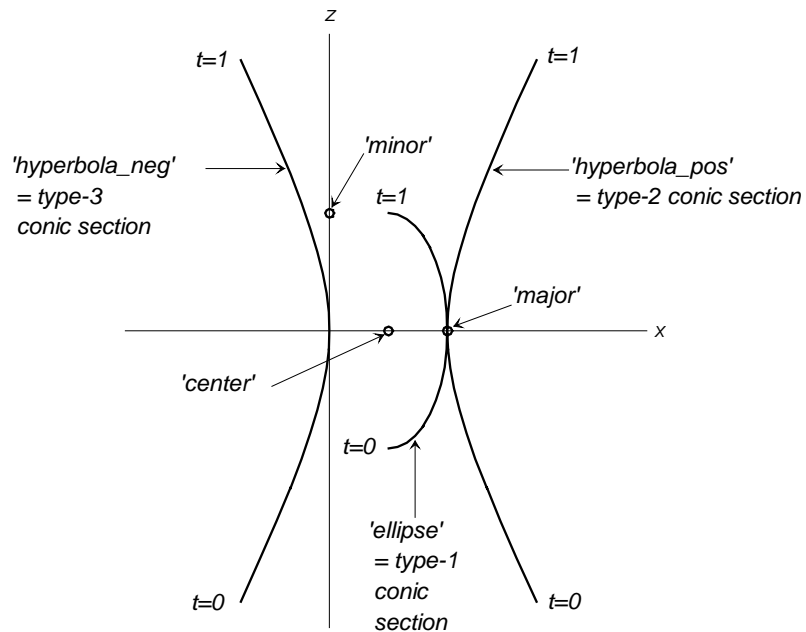
The catenary is the equilibrium curve assumed by a hanging chain or cable under its own weight, with the force of gravity acting in the -a direction. It has engineering applications in the design of bridges and arches.

For all types of Conic Sections, *relabel* is used to relabel the curve. The program's default *relabel* '*' produces the "natural" labeling, in which the parameter t is linearly related as indicated in the formulas above.

Note that the *parabola* is also a conic section. It can be regarded as the limiting case of either an ellipse or a hyperbola as the center moves to infinity. Parabolas can be created in SurfaceWorks in two ways – any type-2 B-spline or C-spline curve with three control points is a parabola.

Example 1

conics.ms2 (ellipse and hyperbola)



CONIC.MS2 in <Y> view; Conic types 1 - 3.

All three of the conic sections in this example are formed using the same control points and conic parameters:

- 'center' is the *center* point
- 'major' is *point2*, the end of the primary axis
- 'minor' is *point3*, the end of the secondary axis
- s0, the conic parameter for the start (t = 0) of the curve is -90
- s1, the conic parameter for the end (t = 1) of the curve is 90

'ellipse' is the type-1 conic section, 'hyperbola_pos' is the type-2 conic section, and 'hyperbola_neg' is the type-3 conic section.

Example 2

catenary.ms2 (catenary)

'catenary' is a type-4 conic section formed by the following three control points and two conic parameters:

'center' is the *center* point

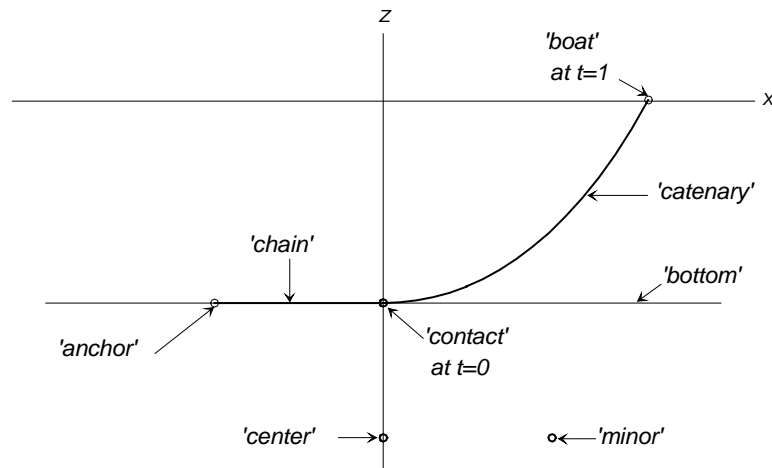
'contact' is *point2*, the end of the primary axis

'minor' is *point3*, the end of the secondary axis

$s0$, the conic parameter for the start ($t = 0$) of the curve is 0

$s1$, the conic parameter for the end ($t = 1$) of the curve is 90

In this example, 'bottom' is a line depicting the bottom of the bay (or lake or ...). 'catenary' is a type-4 conic; it is the portion of anchor chain not resting on the bottom. 'chain' is a Line; it is the portion of anchor chain that is resting on the bottom. 'boat' is a bead at the $t=1$ end of 'catenary'.



CATENARY.MS2 in <Y> view.

See also

Arc, ArcSnake, BCurve, CCurve

Expanded Curve

Characteristic data

Relabel

Type (-3 to -1, 1 to 3) -- Specifies which axis of the frame to use, and which direction along that axis (+/-) corresponds to the positive t direction.

Bead/curve -- Specifies the basis curve; if a bead, it also specifies a particular point on the curve to end up at the frame origin.

Frame

Scale factor

Description

An ExpdCurve (Expanded curve) is to a curve as a development (flattening) is to a developable surface: It is a copy of the curve, rolled

out flat to make a straight line, with preservation of arc-length of all elements.

Note that this construction implies that, in absence of a relabel, the velocity profile of an ExpdCurve will be identical to that of its basis curve.

One application is the layout of structural elements that will be formed by bending straight tubes, shapes or extrusions into curves. In this case, the 3-D basis curve should follow the neutral axis of the shape.

Sample files

ExpdCurve1.ms2

This is the forward master curve of Demo.MS2, flattened into a vertical line. Point 'p0' is used as the frame. The type is -3, therefore the positive-t direction on the ExpdCurve is the negative z axis, i.e., downward. A bead 'e0' at t = 0.257 specifies which point of the ExpdCurve should be at the origin.

NURBS Curve

Characteristic data

point1 ... pointN = control points
wt1 ... wtN = corresponding weights
knotlist = name of a KnotList object or '*' for uniform knot spacing
type = B-spline type: 1 = linear, 2 = quadratic, 3 = cubic, etc.

Description

The NURBCurve is a non-uniform rational B-spline curve. The control points guide or shape the surface in much the same way as the control points shape a B-spline Curve. In general, the curve does not interpolate any of its control points except the first and last (*point1* and *pointN*).

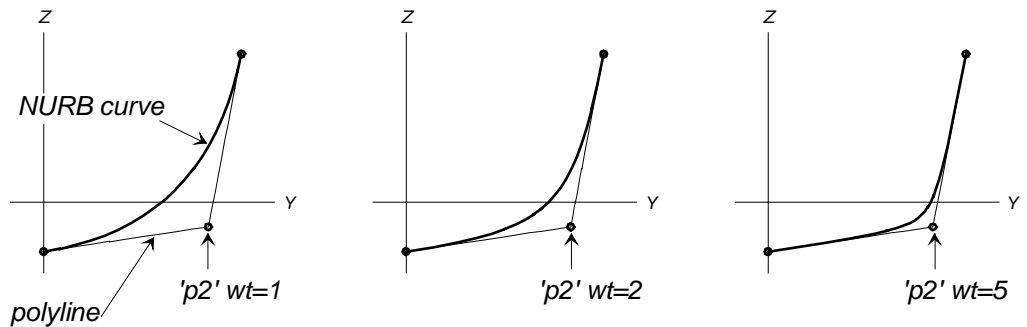
wt1, wt2, etc. are the weights associated with each control point.

knotlist controls the knot spacing. If not default '*' (uniform spacing), the *knotlist* should have (number of control points) - (type) + 1 components, and the list should begin with 0 and end with 1.

The program's default *relabel* produces the "natural" labeling, in which the parameter t is concentrated in highly curved areas formed either by closely spaced control points or by high weights.

Example

nurbcrv1.ms2



NURBCRV1.MS2 in <x> view: initial model (left); 'p2' weight edited (middle and right).

'midsection' is a NURBCurve formed by three points 'p1', 'p2', and 'p3', all of which have wt = 1 (left figure).

In the middle figure, the weight for 'p2' has been changed from 1 to 2, pulling the curve deeper. In the right-hand figure, the weight for 'p2' has been increased to 5, giving 'midsection' a tight bilge curve.

See also BCurve, NURBSnake, NURBSurf, KnotList, BSurf.

Radius Arc

Characteristic data *point1, point2, point3* = control points
radius = radius of arc
type = kind of radius arc

Description The RadiusArc is a circular arc of a specified radius. It is defined by three control points and a radius, and can be built in several different ways, as designated by *type*.

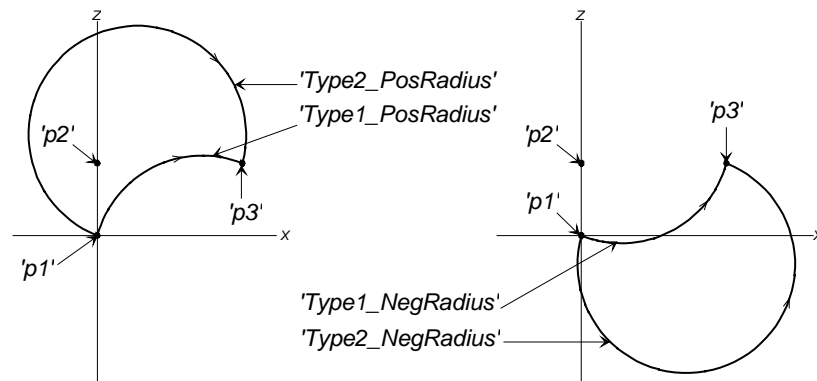
radius is a decimal number, positive or negative. For type-1 and type-2, radius must be less in absolute value than twice the distance from *point1* to *point3*.

type specifies how the radius arc is built from the control points:

- type-1 The arc runs from *point1* to *point3*, in the plane of the 3 points; type-1 is the shorter arc, with angle < 180 deg.
- type-2 The arc runs from *point1* to *point3*, in the plane of the 3 points; type-2 is the longer arc, with angle > 180 deg.
- type-3 This arc rounds the corner formed at *point2* by the directions to *point1* and *point3*. Its t=0 end is tangent to the line from *point1* to *point2*; its t=1 end is tangent to the line from *point2* to *point3*.

For type-1 and type-2 RadiusArcs, note that in a given plane, there are four possible arcs of a given radius between two points:

- type-1, positive radius
- type-1, negative radius
- type-2, positive radius
- type-2, negative radius



RadiusArc types 1 and 2: four possible arcs between two points.

The program's default *relabel* produces uniform labeling with respect to arc length or angle.

Example 1

radiusarc1.ms2

Example: radiusarc1.ms2

This is an example of a constant radius deck made by sweeping a RadiusArc to generate a ProcCvSurf (Procedural Curve Surface).

'deck_radius' (red) is a type-1 RadiusArc. It has a radius of 20 and is built between:

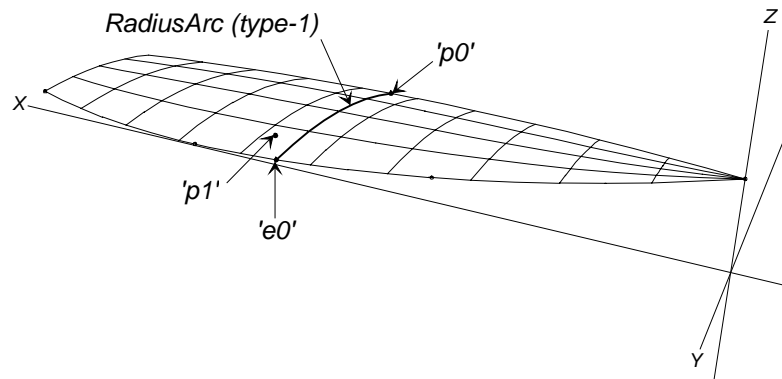
'e0' (bright green; an AbsBead on 'sheer') = *point1* for the arc

and

'p0' (cyan; a MirrPoint of 'e0' across the centerplane) = *point3* for the arc.

'p1' (yellow; a RelPoint off 'e0', at dX=0, dY=0, dZ=.1) is *point2* for the arc. It orients the arc in a plane parallel to the X=0 plane.

The deck is a ProcCvSurf for which the *moving curve* is 'deck_radius' and *bead* = 'e0'. To generate the ProcCvSurf, SurfaceWorks varies t for 'e0' from 0 to 1, and the ProcCvSurf is the surface swept out by 'deck_radius'.



RADIUSARC1.MS2 at Lat -20, Lon 30.

Example 2

radiusarc3-polycurve2.ms2

Example 2: radiusarc3-polycurve2.ms2

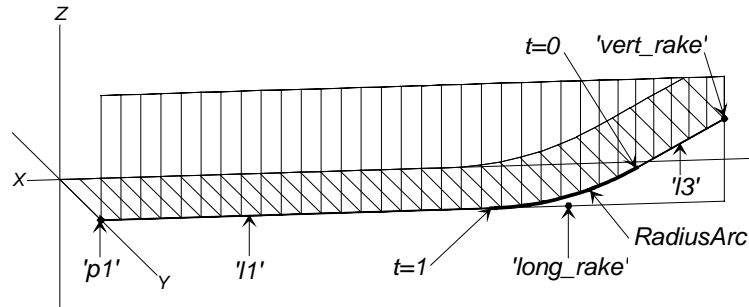
This model is a hard-chine barge with raked ends. A type-3 RadiusArc is used to form the 20-foot-radius curve in the chine.

'radiusarc' (bright green) is a type-3 RadiusArc made from 'vert_rake' (blue, a bead on 'l2'), 'long_rake' (blue, a bead on 'l0'), and 'p2' (cyan, a ProjPoint of a point at the origin). 'radiusarc' is tangent to 'l3' (yellow) at its t=0 end and to 'l1' (magenta) at its t=1 end.

'chine' (cyan) is a PolyCurve2 made from 'l1', 'radiusarc', and 'l0'.

You can vary the longitudinal and vertical extents of the rake by dragging 'long_rake' and 'vert_rake' respectively. The RadiusArc adjusts

itself to maintain its specified radius of 20 ft and it remains tangent to the lines 'l1' and 'l3'.



RADIUSARC3-POLYCURVE2.MS2 at Lat 10, Lon 100.

For discussion of the PolyCurve2 in this model, see the end of the first PolyCurve2 Entity Description example.

See also Arc

Tabulated Curve

Characteristic data *filename* = filename of .3DA or .PAT file
polyline number = number of polyline in file
frame/point = name of a frame or point object or '*' for the default

Description *filename* is the filename of a .3DA (or .PAT) file. If no extension is given, SurfaceWorks assumes .3DA.

A .3DA file (see sample contents and notes in example below) consists of a series of records (file lines) each having four numbers: a *pen* number and X, Y, Z coordinates. A zero pen means "move to this point without drawing". Therefore, if we let *nlinks* equal the number of divisions in a polyline, a polyline with *nlinks* divisions is represented by one pen = 0 record followed by *nlinks* records with nonzero pen. The product of *t-divisions* × *t-subdivisions* must be correctly set equal to *nlinks*, or a geometry error occurs. Each 0-pen record is counted as the start of a new polyline.

polyline no. is the number of the polyline to be taken from the .3DA file: 1 for the first, 2 for the second, etc.

frame/point is the name of a frame or point object or '*' for the default:

When *frame/point* is a point, it specifies an insert origin, and *point's* coordinates are added to the polyline coordinates. This results in a shift (but no rotation).

When *frame/point* is a frame, the file coordinates are interpreted as frame coordinates x,y,z. This generally results in a shift plus a rotation.

When *frame/point* is the default '*', there is no shift or rotation.

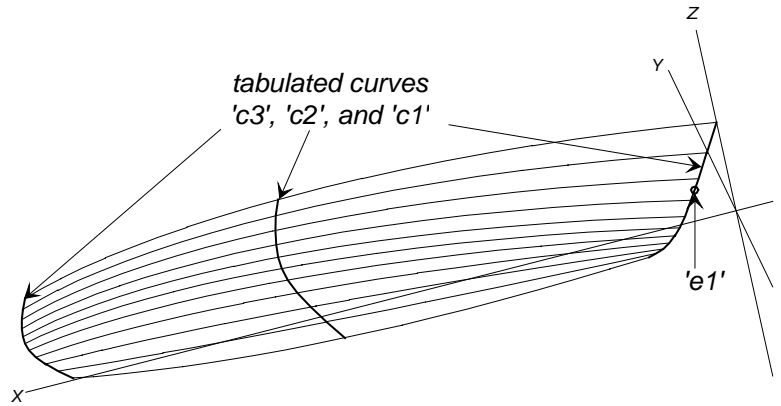
relabel is used to relabel the curve. The program's default *relabel* '*' produces a labeling which corresponds to the labeling of the original polyline points; i.e., the first point is at $t = 0$; the second is at $t = 1/nlinks$, the third is at $2/nlinks$... and the last is at $t = 1$.

Note: A TabCurve has less inherent smoothness than most native SurfaceWorks curves. It is actually a polyline, like a type-1 BCurve. You cannot subdivide a TabCurve to produce a smoother version of the same curve.

Example

tabcurve.ms2 (needs demomcs.3da)

This example produces a model identical to the DEMO model. However, in this case, instead of generating the master curves itself, the model reads them in from a file DEMOMCS.3DA (see contents below). Therefore, the shape of the hull depends entirely on the contents of that file.



TABCURVE.MS2 at Lat -30, Lon 60.

The three TabCurves are named 'c1', 'c2', and 'c3'. For each of them, *filename* is DEMOMCS, and *frame/point* is the default '*' which produces no shift or rotation of the curves. *polyline no.* for 'c1' is 1, for 'c2' is 2, and for 'c3' is 3.

The bead 'e1' is included to show that a TabCurve, like any other curve, will support beads and other objects depending on curves.

Contents of DEMOMCS.3DA

0	0.000	0.000	3.600	pen=0 point (1st point) on 1st polyline 'c1'
11	0.511	0.000	2.493	1st of 10 records with nonzero pen for 1st polyline; <i>nlinks</i> for 'c1' = 10
11	0.951	0.000	1.569	
11	1.321	0.000	0.829	
11	1.619	0.000	0.273	
11	1.845	0.000	-0.099	
11	2.045	0.000	-0.355	
11	2.260	0.000	-0.564	
11	2.491	0.000	-0.724	
11	2.738	0.000	-0.836	
11	3.000	0.000	-0.900	
0	15.000	4.815	2.560	pen=0 point (1st point) on 2nd polyline 'c2'
11	15.000	4.869	1.835	

11	15.000	4.847	1.204	
11	15.000	4.749	0.667	
11	15.000	4.575	0.226	
11	15.000	4.325	-0.121	
11	15.000	3.921	-0.403	
11	15.000	3.286	-0.649	
11	15.000	2.421	-0.860	
11	15.000	1.326	-1.035	
11	15.000	0.000	-1.175	
0	30.000	3.500	2.760	pen=0 point (1st point) on 3rd polyline 'c3'
11	30.000	3.480	2.218	
11	30.000	3.420	1.742	
11	30.000	3.320	1.334	
11	30.000	3.180	0.994	
11	30.000	3.000	0.720	
11	30.000	2.720	0.504	
11	30.000	2.280	0.336	
11	30.000	1.680	0.216	
11	30.000	0.920	0.144	
11	30.000	0.000	0.120	

See also TabPoint, TabSurf, WireFrame

X-Curve

Characteristic data *point1, point2 ... pointN* = control points
type = direction of orientation of XCurve
ecc = end condition code for the two directions other than the direction of orientation (specified by *type*)
s/m1, s/m2, s/m3, s/m4 = slope or moment at each end of the curve for the two directions other than the direction of orientation (specified by *type*)

Description The XCurve was created to add 100% forward compatibility from FL/2B to SurfaceWorks. The XCurve, "explicit spline" or X-spline, is a cubic spline passing through its data points, with knots at the interior data points. In this respect, it is like a type-3 CCurve. However, there are several important differences from a CCurve:

- (1) The XCurve is an explicit function of either X, Y, or Z, according to its *type*. Its natural or default t parameter is distributed in proportion to distance along the specified axis.
- (2) A consequence of the above is that an XCurve must be oriented with respect to its selected axis, and must have that coordinate either strictly increasing or strictly decreasing along its length; it isn't allowed to double back on itself.
- (3) If you rotate the set of points that parent an XCurve about any line not parallel to the selected axis, the curve changes shape, as well as position. If you rotate the points far enough, the XCurve will cause a geometry error.

- (4) At each end of an XCurve, you have explicit control over either the slope or the curvature (bending moment) in two directions. To be more specific, suppose we have a type-1 XCurve, i.e., oriented with respect to the X-axis. At each end of this curve, you will explicitly specify either the slopes Y', Z' or the moments Y'', Z'' .
- (5) An XCurve will not in general lie in a plane, even if all its control points do. There are somewhat intricate constraints on the end condition values that are required to make the curve lie in a plane.
- (6) The Y and Z projections of a type-1 XCurve are completely independent. That is, the Y-projection depends only on the Z-coordinates and the Z' or Z'' end condition values, and vice versa.

type specifies the direction of orientation of the XCurve:

- 1 for X orientation
- 2 for Y orientation
- 3 for Z orientation

Note: this usage of *type* has nothing to do with "spline" *type* as used in BCurve, CCurve entity specifications. XCurses are always cubic splines.

ecc is the end condition code for the two directions other than the direction of orientation (specified by *type*):

- 0 = moment both ends
- 1 = slope at t=0 end, moment at t=1 end
- 2 = moment at t=0 end, slope at t=1 end
- 3 = slope at both ends

s/m1, s/m2, s/m3, s/m4 are the slope or moment at each end of the curve for the two directions other than the direction of orientation (specified by *type*):

	type-1 (X orientation)	type-2 (Y orientation)	type-3 (Z orientation)	
<i>s/m1</i>	Y' or Y''	Z' or Z''	X' or X''	at t = 0 end
<i>s/m2</i>	Z' or Z''	X' or X''	Y' or Y''	at t = 0 end
<i>s/m3</i>	Y' or Y''	Z' or Z''	X' or X''	at t = 1 end
<i>s/m4</i>	Z' or Z''	X' or X''	Y' or Y''	at t = 1 end

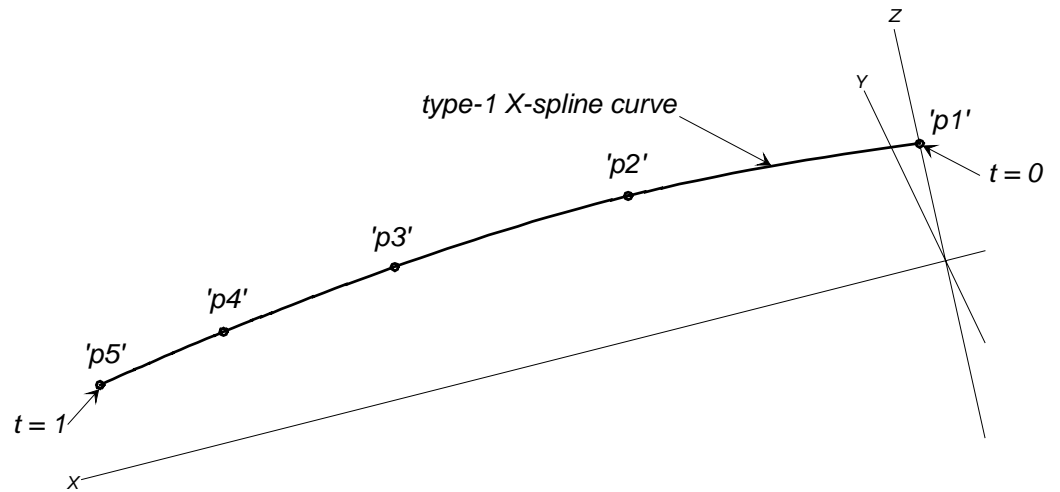
While inclusion of the XCurve was motivated primarily by the desire for full compatibility with FAIRLINE/2, it certainly has some other worthwhile uses. For example... boat hull with straight frames... cambered deck as ALoftSurf... accurately flat spray chines...

Example

xcurve.ms2

This example is a candidate sheerline for an 8-meter sailing yacht. You'll probably want to use Edit/ Model File to follow the discussion. The XCurve is type-1 (oriented to the x-axis), and has end condition code = 1 (slope at t=0 end, moment at t=1 end). It passes through the five points in sequence. At t=0, the Y-slope (dY/dX) is 0.5, and the Z-slope (dZ/dX) is

-0.13. At $t=1$, the Y-moment (d^2Y/dX^2) is -.02, and the Z-moment (d^2Z/dX^2) is .005.



XCURVE.MS2 at Lat -30, Lon 60.

See also

XLoftSurf

Arc Snake

Characteristic data

$magnet1, magnet2, magnet3$ = magnet or ring control points
 $type = 1 - 6$

Description

An Arc Snake is a continuous curve lying in a surface and defined by 3 magnets, which must all lie in the same surface. In the u-v parameter space, the Arc Snake is a circular arc. There are 6 types:

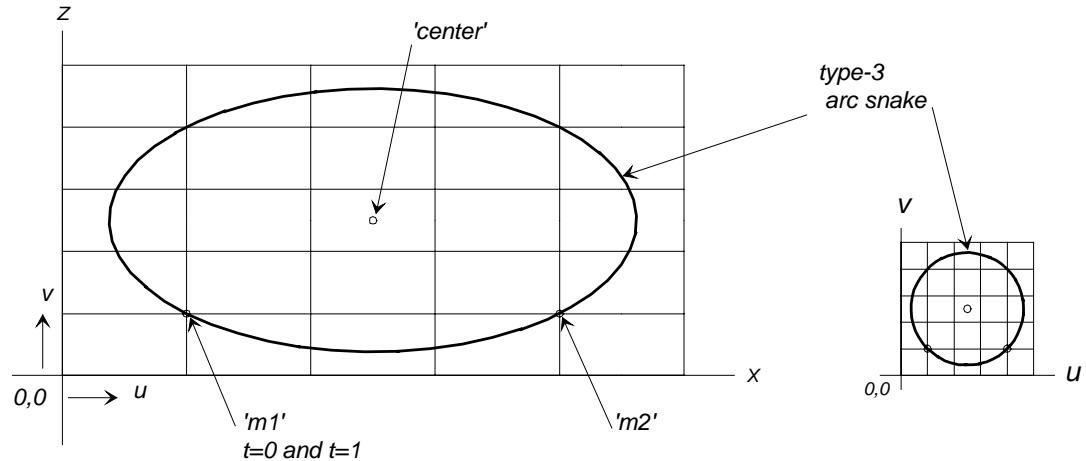
- type-1 arc (through 3 magnets) – the arc passes through all three magnets
- type-2 arc (start-center-guide) – starts at $magnet1$, uses $magnet2$ for its center, and ends (at the correct radius) on a line from the center through $magnet3$
- type-3 circle – the arc is a complete circle starting at $magnet1$ and using $magnet2$ for its center; $magnet3$ has no role except to set the direction of the circle; $magnet3$ is used to set the direction of the circle – the closest point on the circle to $magnet3$ is at a parameter value t that is less than 0.5
- type-4 arc (tangent at start) – the arc starts at $magnet1$, uses $magnet2$ for the middle point of its polygon, ends at $magnet3$, and is tangent to its polygon at the $t=0$ end
- type-5 arc (tangent at end) – the arc starts at $magnet1$, uses $magnet2$ for the middle point of its polygon, ends at $magnet3$, and is tangent to its polygon at the $t=1$ end
- type-6 semi-circle – the arc is a semi-circle starting at $magnet1$ and using $magnet2$ as its center. $magnet3$ determines which of the two possible semi-circles you get – the one that passes closest to $magnet3$

The program's default *relabel* produces the "natural" labeling, in which the parameter *t* is uniformly distributed with respect to arc length or angle, in the *u-v* parameter space.

Example

arcsnake.ms2

'garter' (red), a type-3 arc snake, is shaped by the three magnets 'm1', 'm2', and 'center' (all cyan). 'garter' is a circle in the *u-v* parameter space mapped onto a rectangle 'patch'. This produces a true ellipse. If the shape of 'patch' is changed, 'garter' will change its position in space but will remain always on 'patch', in a corresponding position. 'garter' will remain an ellipse as long as 'patch' remains a rectangle.



ARCSNAKE.MS2 in <Y> view.

See also Arc

Foil Snake

Characteristic data *magnet1* ... *magnetN* = control points
type = type of airfoil family, 1 to 5; types 101 - 255 reserved for user-defined foils

Description A Foil Snake is a true NACA airfoil section or user-defined foil section in the *u-v* parameter plane. This maps into a foil shape lying in the surface its 3 or 5 control point magnets are attached to. If the net of *u-v* parameter lines in the vicinity is roughly rectangular, the Foil Snake will be close to a true NACA section or user-defined foil in space.

Foils can be constructed with either 3, 4, or 5 control magnets — 3 for a half section, 4 for a symmetric full section, or 5 for a cambered or symmetric full section. In all cases:

- The snake lies in the same surface as its magnets.
- *magnet1* is the trailing edge.
- *magnet3* is the leading edge.
- The "chord line" is the Line Snake joining *magnet1* and *magnet3*.
- *point2*, and *point4* of a 5-point foil, determine thickness and camber.

- The last control point in a 4- or 5-point foil nominally closes the foil (usually same point as *point1*).

type:

type-1	NACA 4-digit series; max camber at 40%
type-2	NACA 63 series with a=0.3 mean line
type-3	NACA 64 series with a=0.4 mean line
type-4	NACA 65 series with a=0.5 mean line
type-5	NACA 0010-34; max camber at 40%
types 101-255	user-defined foils

relabel is used to relabel the snake. The default *relabel* '*' produces a labeling in which the parameter t is concentrated in the highly curved leading edge area, in the u-v parameter space. Subdivision control may be important if you are creating panels for a potential-flow code.

Notice

The implementation of FSnake may be changed in a future version. We have found FSnakes to be of marginal value because of the distortion of the foil shape that occurs when the true foil outline is expressed in u, v parameters and mapped onto the distorted mesh of u, v grid lines. We now see the possibility of compensating for this distortion, producing a much truer rendition of the foil.

The ProjSnake entity provides a way to overcome this problem: instead of using an FSnake, use a ProjSnake made from an FCurve. For example, in the FSNAKE2.MS2 example below, you could make an FCurve lying in a horizontal plane such as Z = 0, then project it vertically onto the hull surface.

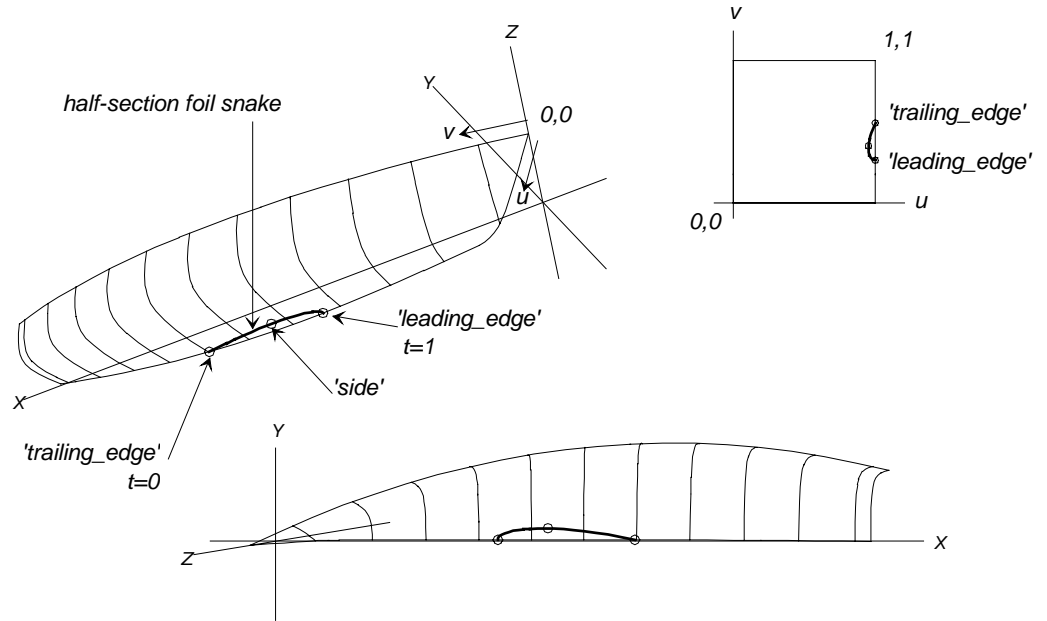
Example

fsnake2.ms2

'hull' is a C-lofted surface made from 5 master curves 'MC1' to 'MC5'. u = 0 is the sheerline, and v = 0 is 'MC1'.

'trailing_edge', 'side', and 'leading_edge' are three magnets on 'hull'. 'trailing_edge' and 'leading_edge' are located on the u = 1 edge of the surface.

'keel_root' is a type-1 half-section foil curve, defined by the three magnets and lying on 'hull'.



FSNAKE2.MS2 at Lat -30, Lon 50 (top) and Lat 90, Lon -90 (bottom).

If 'hull' is changed by modification of one or more of the master curves, 'keel_root' will change its position in space but will remain always on 'hull', in a corresponding position.

See also FCurve

Geodesic Snake

Characteristic data

- curvature* = a geodesic curvature value (defined below)
- graph* = name of a BGraph object or '*' for default graph
- log-tolerance* = logarithm (base 10) of tolerance for matching prescribed curvature
- magnet1* locates the t=0 end of the snake
- magnetN* locates the t=1 end of the snake

Description

A Geodesic Snake is a curve embedded in a surface, with a specified distribution of geodesic curvature along its length (for definition of geodesic curvature, see "Geodesic Curvature" below). The geodesic curvature can be zero (constant) in which case the GeoSnake is a geodesic on the host surface.

Note that calculation of a GeoSnake is a computationally intensive searching process, without guarantee of success (details below in "Failure to converge"). Therefore, unless your task really requires accurate control over geodesic curvature, we recommend that you use a simpler snake entity, such as a LineSnake or BSnake.

A GeoSnake is defined by 2 or more magnets which all must lie in the same surface. The program uses a type-2 BSnake made from the magnets as its starting path for computing the GeoSnake. If you give only 2

magnets, start and end, the type-2 BSnake is just the LineSnake from *magnet1* to *magnet2*. For many GeoSnake applications, especially those with zero *curvature*, two magnets will be sufficient. If the resulting GeoSnake fails to converge, you can add one or more intermediate magnets to get a better starting configuration.

A GeoSnake can have zero (constant) geodesic curvature for its entire length, or it can have a specified distribution of geodesic curvature. *curvature* and *graph* are the controls:

curvature is a geodesic curvature value, units = 1/length. 0 curvature makes an actual geodesic satisfying Geodesic conditions (1), (2) below.

graph modulates the geodesic curvature; i.e., multiplies the curvature value. The default *graph* '*' is interpreted as a constant, 1.

log-tolerance allows you to specify a tolerance for the convergence of the iterative search SurfaceWorks uses to compute the GeoSnake. The value is specified as the log (base 10) of tolerance (using a logarithm here allows a wide dynamic range). Example: *log-tolerance* = -4 means tolerance = 10^{-4} = .0001. *log-tolerance* is provided as a control because practical accuracy requirements for geodesics vary widely between applications.

Tolerance itself is the largest tolerable deviation of calculated geodesic curvature from specified geodesic curvature, multiplied by the total arc length of the GeoSnake (this makes it non-dimensional). Example: a curve with 10-meter arc length and maximum curvature of 10^{-3} /meter (curvature x arc length = 10^{-4}) could deviate from a straight line by no more than 0.125 mm.

relabel is used to relabel the snake. The default *relabel* '*' produces the "natural" relabeling of the snake which is uniform with respect to arc length.

Geodesic curvature *Geodesic curvature* is a pointwise scalar property of a curve lying in a surface, determined as follows:

- (1) At any point along the curve, construct the plane tangent to the surface.
- (2) Project the local portion of the curve normally onto the tangent plane.
- (3) Geodesic curvature is the curvature of the projection.

Geodesic A *geodesic* is a curve embedded in a surface, with a special property which can be expressed in either of two mathematically equivalent ways:

- (1) A geodesic has zero geodesic curvature throughout its length.
- (2) A geodesic is the shortest curve in the surface joining its endpoints.

Failure to converge The typical way a GeoSnake fails to evaluate is "failure to converge." Features which can promote this problem are:

- coarse subdivisions of the host surface
- irregular mesh on the host surface
- lack of smoothness of the host surface
- passing through or close to a coordinate singularity
- starting path far from the geodesic path

- tight tolerance

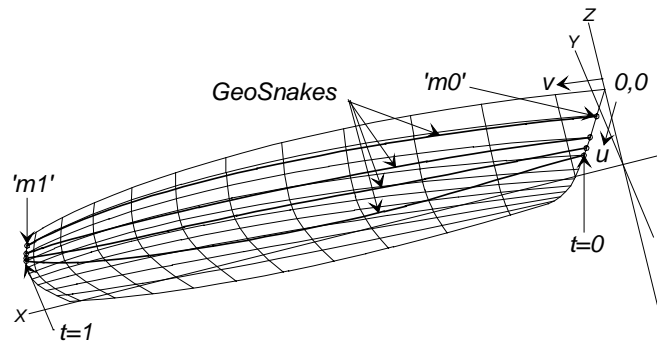
Usage

Geodesics have significant applications in manufacturing:

- (1) Ribbands — they must lie along geodesic paths if they are to conform to the hull surface between frames (see “Example”).
- (2) Planked wooden boat construction — geodesic seams produce straight planks.
- (3) Cold-molded plywood construction — veneers have to lie along geodesic paths. (Take a look at VENEERS.MS2 which shows a hull in the process of veneer layout. After making the first veneer-edge GeoSnake, we saved it as a component, then loaded it successively, moving each new GeoSnake to its proper location by moving its AbsRing control. Then we’d make DevSurfs (the veneers) between adjacent pairs of GeoSnakes.)
- (4) Sail making — geodesic seams are key to efficient use of cloth and flat panel development. (Take a look at SAILSEAM.MS2 which shows a jib in the process of being seamed; eventually, DevSurfs would be created between each adjacent pair of seams.)
- (5) Plate expansion — a mesh of geodesic triangles is key step in the process.
- (6) Compounding by line heating — using geodesics for heat lines minimizes residual stress.
- (7) Automated lamination — reinforcing tape has to be laid along geodesic paths, or it wrinkles and buckles.

Example

ribbands.ms2



In this example, GeoSnakes are used to layout the paths for ribbands. The use of geodesics makes ribbands that will naturally lie flat on the frames and conform to the hull surface between frames.

'm0' is an AbsMagnet on the $v = 0$ edge of the hull; 'm1' is a RelMagnet on the $v = 1$ edge of the hull (at $du = 0, dv = 1$ from 'm0'). 'n0' is the GeoSnake between 'm0' and 'm1'.

To make the set of ribbands, we saved this first ribband as a component (objects 'm0', 'm1', and 'n0'; RIBBAND.MC2), then loaded the component successively to make the other ribbands. Since we made the initial ribband so that it can be moved by moving just the magnet 'm0', it is easy to move each component ribband to its own position on the hull.

Note that where the hull has relatively high curvature at the turn of the bilge, we had to play with the position of the control magnet to get the geodesic we wanted. In this region, a small movement up or down of 'm0' makes a big difference in the geodesic path.

See also LineSnake

NURBS Snake

Characteristic data *magnet1 ... magnetN* = control points and weights
wt1 ... wtN = corresponding weights
knotlist = name of a KnotList object or '*' for uniform knot spacing
type = B-spline type: 1 = linear, 2 = quadratic, 3 = cubic, etc.

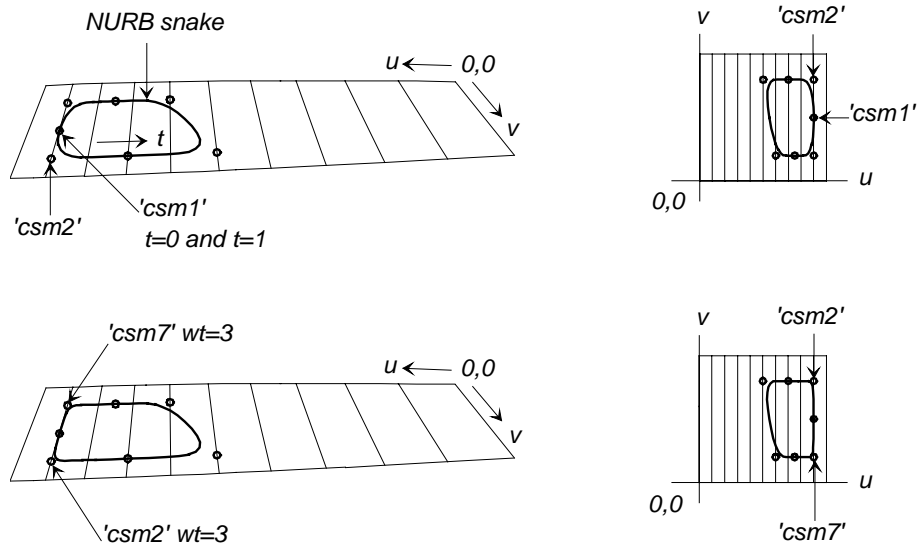
Description A NURBSnake is a continuous curve lying in a surface and defined by a series of magnets for control points. The magnets must all lie in the same surface. The curve ends at the two end control points (*magnet1* and *magnetN*), but in general does not pass through the others (unless snake is type-1). Instead, it is "molded" or "shaped" by them. In the u-v parameter space, the NURBSnake is a true NURBCurve.

wt1, wt2, etc. are the weights associated with each control point.

knotlist controls the knot spacing. If not default '*' (uniform spacing), the *knotlist* should have (number of control points) - (type) + 1 components, and the list should begin with 0 and end with 1.

The program's default *relabel* '*' produces the "natural" labeling, in which the parameter *t* is concentrated in highly curved areas formed either by closely spaced control points or by high weights, in the u-v parameter space.

Example **nurbsnk1.ms2**



NURBSNK1.MS2 in Lat 0, Lon 60 orthographic view: initial model (top); with two weights edited (bottom).

'cabin_side' is the side of a boat cabin. 'portlight' is a type-2 NURBSnake formed by the seven magnets 'csm1' to 'csm7' which all lie on 'cabin_side' and all have $wt = 1$ (top figure). If the shape of 'cabin_side' is modified, 'portlight' will change its position in space but will remain always on 'cabin_side', in a corresponding position.

The lower figure shows the effect of increasing the weights of 'csm2' and 'csm7' to 3.

See also

BCurve, NURBCurve, NURBSurf, KnotList, BSurf

PolySnake

Characteristic data

snake1, snake2 ... snakeN = component snakes (t-orientation doesn't matter)

Description

A PolySnake is a single curve made by joining two or more other snakes (called the "component snakes") end-to-end. The component snakes must all lie in the same surface and be selected in the order in which they are to be assembled.

It is anticipated that the end of *snake1* is actually the same point as one of the ends of *snake2*, etc. However, this is not enforced or checked. If you specify snakes that do not share endpoints, the moving point will jump discontinuously from the end of *snake1* to the start of *snake2* as the parameter t passes t_1 , etc. (see next paragraph).

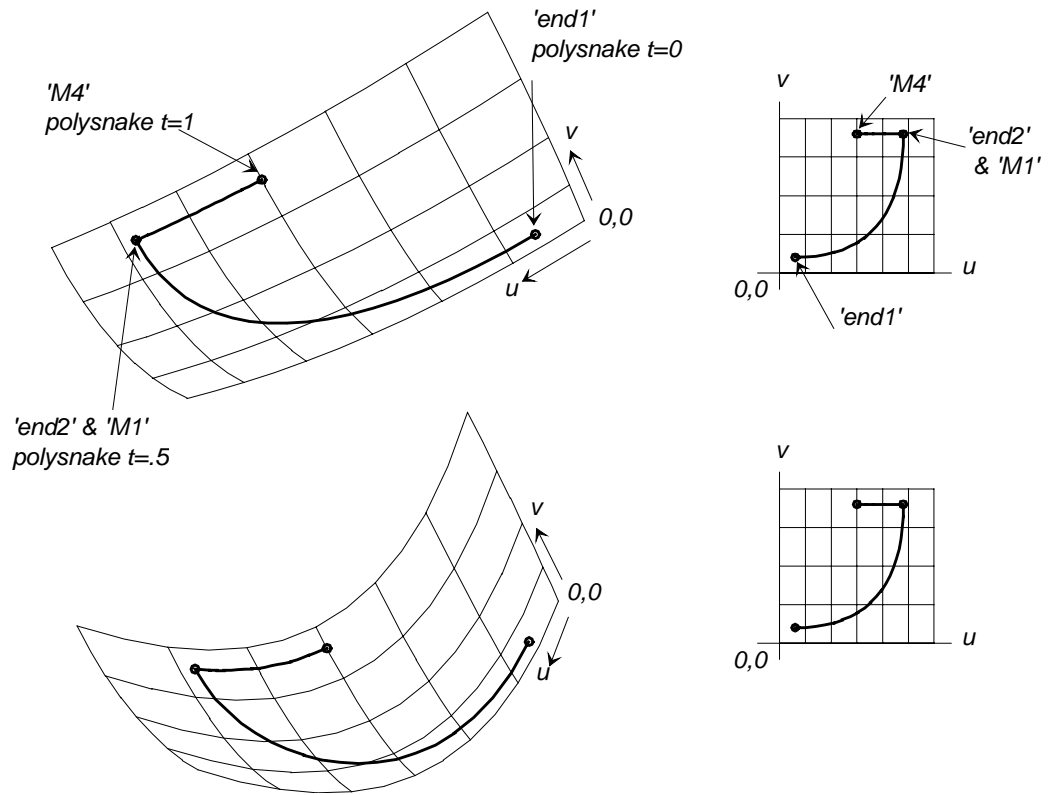
relabel is used to relabel the snake. The program's default *relabel* '*' produces the "natural" labeling, in which the parameter t is t_1 at the end of the first component snake, t_2 at the end of the second component snake, etc., in the u - v parameter space.

Example

polysnk1.ms2

't_reversed' (cyan) is the same subsnake we created in the second example for the "SubSnake" entity description. 't_reversed' begins ($t = 0$) at 'end1' and ends ($t = 1$) at 'end2'. 'lsnake' (yellow) is a line snake that begins at magnet 'M1' and ends at magnet 'M4'. Both 't_reversed' and 'lsnake' lie in the surface 'patch'. 'end2' ($t = 1$ for 't_reversed') and 'M1' ($t = 0$ for 'lsnake') are at the same position. 'polysnake' (magenta) is formed by joining 't_reversed' and 'lsnake'. $t = 0$ for 'polysnake' is at 'end1', $t = .5$ is at the end of 't_reversed' ('end2'), and $t = 1$ is at the end of 'lsnake' ('M4').

In the lower figure, the shape of 'patch' has been changed, but you can see that 'polysnake' still lies in the surface.



POLYSNK1.MS2 at Lat -60, Lon 60: initial model (top); with 'patch' surface edited (bottom).

See also PolyCurve, SubSnake

NUBFitCurve, NUBFitSnake, NUBFitSurf entities

The "NUBFit" entities are essentially replacements for the "BFit" entities. BFitCurve, BFitSnake and BFitSurf – with substantially greater flexibility. You can think of the "NU" as standing for "non-uniform", or for "New". The BFits still exist for backwards compatibility, but the NUBFits do everything the BFits did, and much more.

NUBFitCurve

Characteristic data	Type
	Knotlist
	nc-specifier
	Log-tolerance
	Curve

Description

Compared with a BFitCurve, the only thing new here is the Knotlist. There are 3 alternatives for this entry:

1. '*' (the "default" knotlist) - This allows the fitting to use the best set of nonuniform knots it can come up with.
2. '*UNIFORM' - a new "predefined" knotlist, available in the System Objects: this forces the fitting to use uniform knots, essentially duplicating the function of BFitCurve.
3. You can choose a KnotList or KnotList2 object containing a suitable specified list of knots.

As in the BFitCurve, for nc-specifier, there are two basic choices:

A nc-specifier can be a positive integer (greater than *type*), specifying a definite number of control points to use for the fit.

B nc-specifier can be 0, which signifies "increase the number of control points until tolerance is met".

Since there are 3 alternatives for Knotlist and 2 alternatives for nc-specifier, altogether there are 6 combinations which produce different behaviors for the

NUBFitCurve entity:

1A. '*' knotlist, combined with positive nc-specifier: The fitting will use exactly nc-specifier control points, and will use the best non-uniform knots it can generate.

1B. '*' knotlist, combined with nc-specifier = 0: The fitting will increase the number of control points (from $type+1$ up to the number of data points), until tolerance is met. At each step, it will optimize the non-uniform knotlist.

2A. '*UNIFORM' knotlist, combined with positive nc-specifier: The fitting will use exactly nc-specifier control points and uniform knots.

2B. '*UNIFORM' knotlist, combined with nc-specifier = 0: The fitting will increase the number of control points (from $type+1$ up to the number of data points), until tolerance is met. At each step, it will use uniform knots.

3A. KnotList object, combined with positive nc-specifier: The fitting will use exactly nc-specifier control points, along with the specified KnotList. In this case, there is a required relationship between nc-specifier, *type*, and number of knots: the number of interior knots (i.e., knots which are > 0 and < 1) = number of control points + *type* - 1.

3B. KnotList object, combined with nc-specifier = 0: This case results in an error

condition; because of the relationship noted under (3A); the number of control points is not free to escalate when the *type* and number of knots are

both specified.

Tools/Clearance works with NUBFits (as with BFits); with a single NUBFit object

selected, it reports the number of control points used and the goodness-of-fit RMS

and worst-point statistics.

Tools/Special/Freeze Fit (and the BFreeze command) work with NUBFits just as for

BFits. They will generate a NURBCurve and its control points (or NURBSnake and

its control magnets) as freestanding SurfaceWorks objects. When the fit is non-uniform, it

generates a KnotList in addition.

NUBFitSnake

The NUBFitSnake entity is very similar to the NUBFitCurve. Its parent must be a snake, and the resulting approximation is in the u,v , space of the supporting surface.

Its control points are u,v locations on the supporting surface. The “tolerance” in $\log\text{tolerance}$ is a dimensionless value in u,v -space, rather than a distance.

NUBFitSurf

The NUBFitSurf entity works similarly to NUBFitCurve, but it approximates a surface.

Characteristic data

u-type

u-knotlist

ncu-specifier

v-type

v-knotlist

ncv-specifier

log-tolerance

Surface

Description

The fittings in the u - and v -directions are relatively independent. As with NUBFitCurve, there are 3 alternatives for each knotlist entry:

1. ‘*’ (the “default” knotlist) - This allows the fitting to use the best set of nonuniform knots it can come up with.
2. ‘*UNIFORM’ - a new “predefined” knotlist, available in the System Entities; this forces the fitting to use uniform knots, essentially duplicating the function of BFitCurve.
3. You can choose a KnotList or KnotList2 object containing a suitable specified list of knots, and two alternatives for each nc -specifier:

A. nc-specifier can be a positive integer (greater than *type*), specifying a definite number of control points to use for the fit.

B. nc-specifier can be 0, which signifies “increase the number of control points until tolerance is met”.

These can be used in any combination, except 3B. (nc-specifier = 0 can't be used in combination with a specified KnotList object, because of the fixed number of knots.)

One common reason to use a NUBFitSurf prior to exporting IGES is to control the characteristics and quality of NURBS approximation on a surface-by-surface basis.

Another reason is to achieve exact matching along shared surface edges in the exported file. For example, suppose we have 'topside' and 'bottom' surfaces on a single-chine hull, joining each other exactly along the chine; suppose the u-direction is longitudinal and the v-direction is transverse on each surface. If we just export the 'topside' and 'bottom' surfaces through IGES with a global tolerance, we are very likely to come out with a different number of control points on the two exported surfaces. For example, the 'topside' might meet the tolerance with 12 x 5 control points, while 'bottom' gets 10 x 7. Two NURBS curves with different numbers of control points and different knots are identical only under extremely unlikely conditions, so there will almost certainly be gaps and overlaps between the two IGES surfaces that result.

However, if we first make two NUBFitSurfs 'nub_topside' and 'nub_bottom' using the following conditions:

- same ncu-specifier (say, 12 for this example)
- same knots for the u direction
- same number of u-divisions x u-subdivisions on the base surfaces

then we are guaranteed that the two fitted surfaces will join exactly along their common edge. Since there is no further approximation in the export of the NUBFitSurfs, the two exported surfaces will join exactly in the IGES file, and in the receiving application.

Tools and NUBFit Surf

Tools/Clearance works with NUBFitSurf (as with BFitSurf). With a single NUBFitSurf selected, it reports the number of control points used and the goodness-of-fit RMS and worst-point statistics.

Surfaces

Foil Lofted Surfaces

Characteristic data	<i>curve1 ... curveN</i> = master curves (t-orientation doesn't matter) <i>type</i> = type of airfoil family, 1 to 5; types 101 - 255 reserved for user-defined foils <i>relabel</i> = name of a Relabel object or '*' for default labeling
Description	<p>In the Foil-Lofted Surface, the lofting curves are Foil Curves. This means the Foil-Lofted Surface passes through its first (trailing edge), third (leading edge), and fourth or fifth (trailing edge; if present) master curves, but in general not the others, which control the thickness and camber much as the second and fourth (if present) control points of a Foil Curve do.</p> <p>Similarly to a Foil Curve, the number of master curves (3, 4, or 5) controls whether you generate a half or full profile.</p>
<i>Half-section surface</i>	<p>With 3 master curves, you get one half of a symmetric foil-lofted surface, with the parameter <i>v</i> running from 0 at the trailing edge to 1 at the leading edge. <i>curve2</i> controls the half-thickness of the symmetric foil; this is the distance of <i>curve2</i> from the ruled surface consisting of all the chord lines joining the leading and trailing edges.</p>
<i>Full-section surface</i>	<p>With 4 or 5 master curves, you get a full-section foil-lofted surface – that is, if <i>curve1</i> and the last master curve coincide, or are the same curve. When <i>curve1</i> and the last master curve are not the same, you still get a more-or-less foil-shaped surface, but the trailing edge is open.</p> <p>5 master curves. This creates a full section surface:</p> <ul style="list-style-type: none">• <u>symmetric</u> – when <i>curve2</i> and <i>curve4</i> are symmetrically placed (mirror images) with respect to the chord line• <u>cambered</u> – when <i>curve2</i> and <i>curve4</i> are NOT symmetrically placed with respect to the chord line <p><i>curve2</i> and <i>curve4</i> act together in determining the thickness and camber. The full-section foil will be cambered if <i>curve2</i> and <i>curve4</i> are not symmetrically placed with respect to the chord line.</p> <p>4 master curves. This creates a <u>symmetric</u> full section. A 4-point foil behaves just like a 5-point foil in which the 4th parent is the mirror image of the 2nd parent across the chord line.</p> <p>u,v parameters. The <i>u</i> parameter on the surface is the same as the <i>t</i> parameter for the first master curve. The <i>v</i> parameter is the parameter along the lofting foils, running from 0 at the trailing edge (upper surface)</p>

to 0.5 at the leading edge, then to 1.0 at the trailing edge (lower surface).
The lofting foils are the $u = \text{constant}$ lines.

type:

- type-1 NACA 4-digit series; max camber at 40%
- type-2 NACA 63 series with a=0.3 mean line
- type-3 NACA 64 series with a=0.4 mean line
- type-4 NACA 65 series with a=0.5 mean line
- type-5 NACA 0010-34; max camber at 40%
- types 101-255 user-defined foils

relabel is used to control the labeling of the v-direction (the lofting curves) of the surface. See *Reference* section "Relabeling Curves and Snakes."

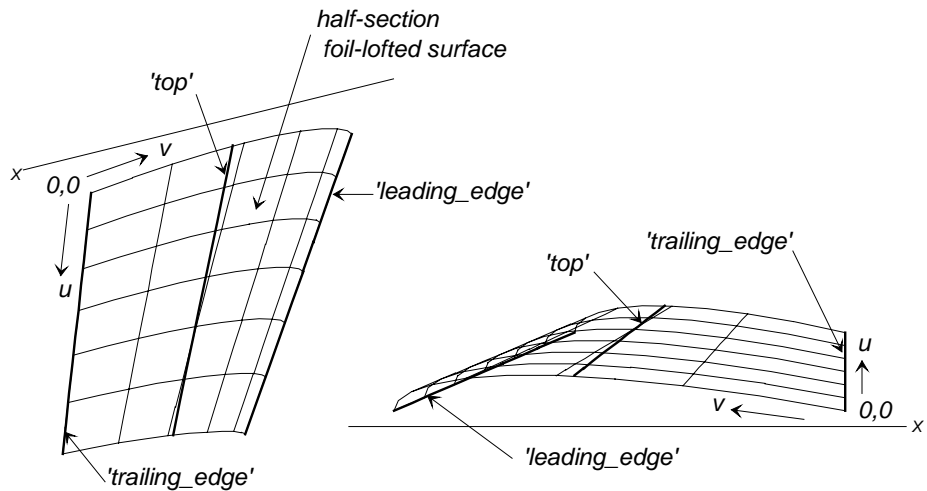
Usage

The Foil-Lofted Surface can be used to create complex wing, keel, rudder, and hydrofoil geometries having true NACA sections or user-defined foil sections.

Example 1

floft1.ms2 (half-section)

'keel' is a type-1 half-section foil-lofted surface defined by 3 master curves: 'trailing_edge', 'top', and 'leading_edge'. 'keel' passes through 'trailing_edge' and 'leading_edge' and uses 'top' to establish the half-thickness of the foil surface.



FLOFT1.MS2 at Lat -30, Lon 70 (left).

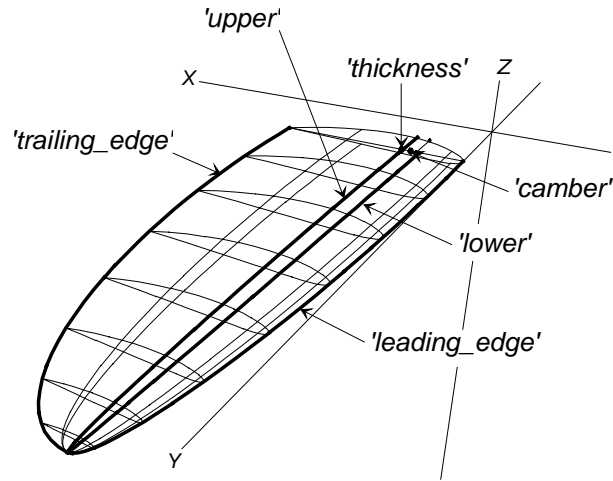
Example 2

floft2.ms2 (full-section)

FLOFT2.MS2 is a cambered elliptic wing made from 5 master curves that are full foil sections: 'trailing_edge', 'upper', 'leading_edge', 'lower', and 'trailing_edge' (again). Note that 'trailing_edge' is named twice in order to close the surface. In the model (but not the figure below), tickmarks are turned on for the master curves so you can see them more easily.

Camber and thickness can be adjusted independently, by sliding RelBeads 'camber' (cyan) and 'thickness' (red) respectively. (The model is constructed so that the t-offset of RelBead 'camber' is the

camber/chord ratio, and the t-offset of RelBead 'thickness' is half the thickness/chord ratio.)



FLOFT2.MS2
at Lat 30, Lon 60
(tickmarks turned off).

See also

FCurve, FSnake

B-Fit Surface

Characteristic data *u*type, *v*type = B-spline types for u and v directions: 1, 2, 3, etc.
ncu-specifier, *ncv*-specifier = integers; specify choices for number of control points to use for fit (0 if free)
log-tolerance = log (base 10) of tolerance
surface = basis surface

Description

A BFitSurf is a uniform B-spline surface, least-squares fitted to the tabulated data of its basis surface. Its principal function is for exporting surfaces to NURBS-based systems.

- you can see the NURBS approximation results right in SurfaceWorks, prior to exporting IGES
- you can assure accurate joins between NURBS surfaces (by forcing the same number of control points along adjoining surface edges)
- you can specify different tolerances for different surfaces

log-tolerance is the tolerance specified with a base-10 logarithm, e.g. -3 for tolerance of 10^{-3} . This notation permits a wide dynamic range. (Tolerance itself is measured as the RMS (root-mean-squared) deviation of the fitted surface from tabulated points of the basis surface.)

*u*type and *v*type are the spline type for the u and v directions respectively: 1 = linear, 2 = quadratic, 3 = cubic, etc.

ncu-specifier and *ncv*-specifier are integer values which specify choices for the number of control points to be used in the u- and v-directions. For each specifier there are two choices:

When you specify a positive value (it must be greater than the corresponding *type*), the program will use exactly this number of control points for the fit.

When you specify zero, the program will start at *type* +1 control points and cycle the number upward until it either achieves the fit within the tolerance or reaches the limit of 128 control points.

If the BFitSurf does not meet the specified tolerance, the Error window will pop up and let you know the problem. Clicking the <Edit> button gives you quick access to the Edit dialog where you can try a different set of values.

When the BFitSurf is displayed, you can see where the control points are by turning on the "net" (*visibility* = net) for the surface – the control points are located where the net lines meet.

If you would like SurfaceWorks to report information about the quality of fit, select the BFitSurf and then choose Tools/ Clearance.

Example **bfitsrf1.ms2**

This is the demo model plus a BFitSurf. For the first run, we used the default values: *u-type* and *v-type* both = 3, *ncu-specifier* and *ncv-specifier* both = 0 (which lets the program set the number of control points), and *log-tolerance* = 0. The discrepancies between the XContours on the 2 surfaces visually shows that the fit is not great (Fig. 2, left). Checking Tools/ Clearance with the BFitSurf selected, reports that SurfaceWorks used *ncu* = 4 and *ncv* = 4 to achieve the fit.

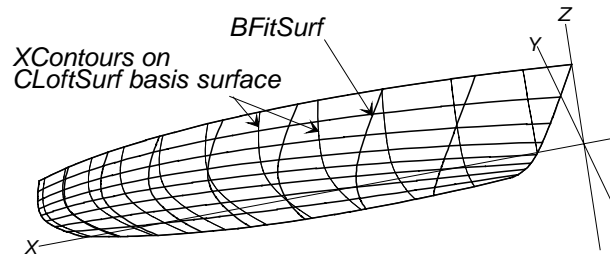


Fig. 1

With *log-tolerance* = -2 (the value it has in the final sample model file), the fit is better (Fig. 2, right). Tools/ Clearance shows that SurfaceWorks used *ncu* = 6 and *ncv* = 4 to achieve this fit.

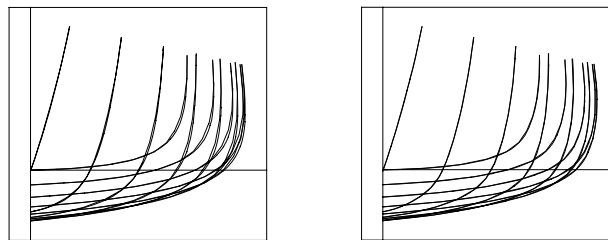
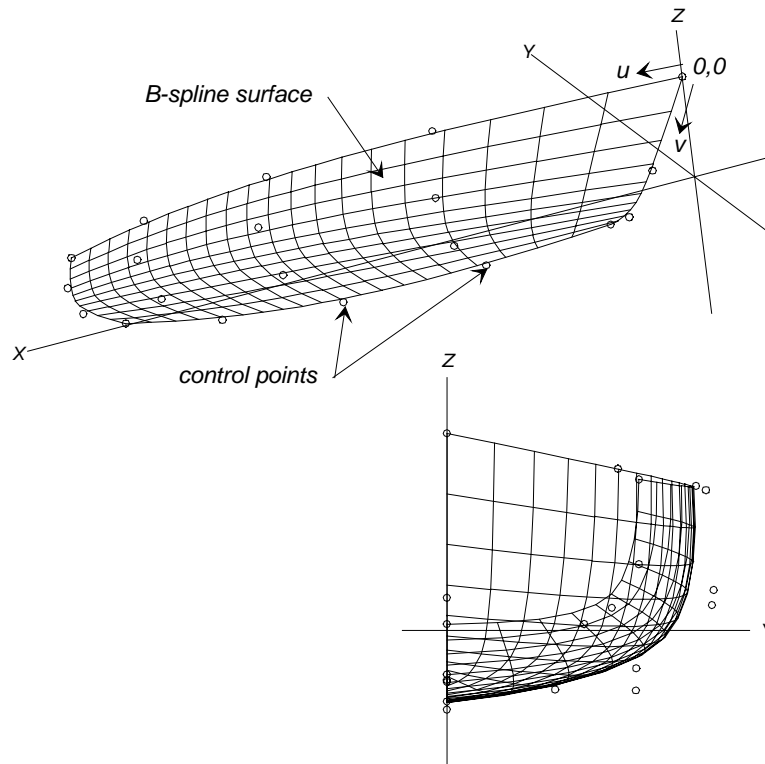


Fig. 2. Ship lines (body plan lifts) of 2 variations the BFitSurf: with *log-tolerance* = 0.0 (left); with *log-tolerance* = -2 (right).

See also BFitCurve

B Surface

Characteristic data	<i>point1 ... pointMN</i> = control points <i>utype, vtype</i> = B-spline types for u and v directions: 1, 2, 3, etc. <i>ncu</i> = no. of control pts. in u-direction (<i>ncv</i> = in documentation below, no. of control points in v-direction)
Description	<p>BSurf is a uniform B-spline surface. The control points form a quadrilateral net which guides or shapes the surface in much the same way as the control points shape a B-spline curve. In general, the surface does not interpolate any of its control points, except the four at the corners of the net. You can think of the surface as being connected to the interior control points by springs.</p> <p>The edges are all B-spline curves using control points from the four edges of the net:</p> <ul style="list-style-type: none">v = 0 uses the first <i>ncu</i> control pointsv = 1 uses the last <i>ncu</i> control points <p>A B-spline Surface is a simplified case of a NURB Surface — the knots are uniform in both directions, and you don't need to use weights because they are all set to one. Technically, a B-spline should have at least one more control point than its type, e.g. at least 2 for type-1, 3 for type-2, etc. However, SurfaceWorks will automatically “demote” a spline to lower type as necessary to fit the number of control points. For example, a type-3 spline specified with only 2 points will be treated as type-1; with 3 points it will be treated as type-2; only with 4 or more points will it actually be a cubic spline.</p>
Example	<p>bsurf.ms2</p> <p>In this example, 'hull' is a B-spline surface which uses the same data points as the models:</p> <ul style="list-style-type: none">CLFT5X4.MS2 in which 'hull_c' is a C-lofted surface (“CLOftSurf” example)BLFT5X4.MS2 in which 'hull_b' is a B-lofted surface (“BLOftSurf” example 1) <p>Generally, a B-spline surface will lie inside a C-lofted surface defined by the same set of control points. A B-spline surface and a B-lofted surface defined by the same set of control points and using the same types of B-splines will be identical, but the BLOftSurf entity is much more flexible than the BSurf entity when it comes to refinement of surfaces.</p> <p>For the surface 'hull' in this example, the u-direction (longitudinal, in this case) B-splines are type-3 (cubic), the v-direction B-splines are type-2 (quadratic), and the number of control points in the u-direction is 5.</p>



BSURF.MS2 at Lat -20, Lon 50 (top) and in <x> view (bottom).

See also BLoftSurf, CLoftSurf, NURBSurf

NURBS Surface

Characteristic data *point11, point21 ... pointMN* = control points
wt1, wt2 ... wtMN = corresponding weights
knotlist1 and *knotlist2* = names of two KnotList objects or '*' for uniform knot spacing
utype, vtype = B-spline types for u and v directions: 1, 2, 3, etc.
ncu = number of control points in u-direction
 (ncv = in documentation below, number of control points in v-direction)

Description The NURBSurf is a rational B-spline surface. The control points form a quadrilateral *net* which guides or shapes the surface in much the same way as the control points shape a B-spline Curve (Fig. 1). In general, the surface does not interpolate any of its control points except the four at the corners of the net.

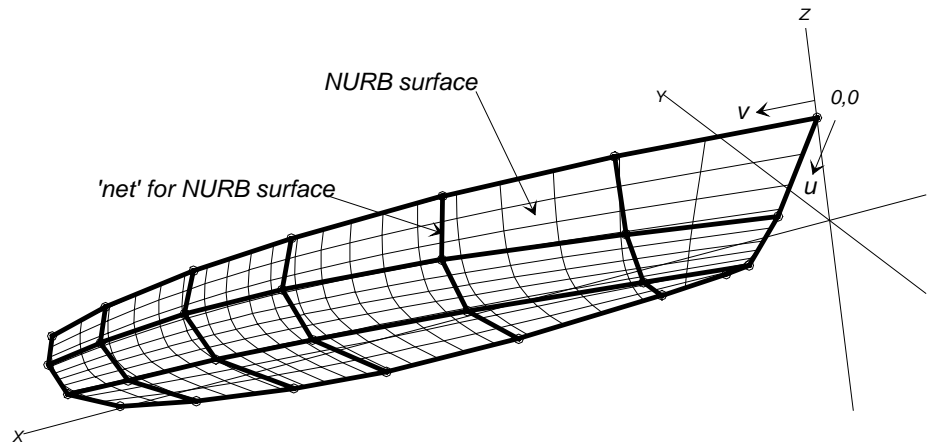


Fig. 1
A NURBSurf and its net.

The edges are all rational B-spline curves using control points from the four edges of the net:

$v = 0$ uses the first ncu control points

$v = 1$ uses the last ncu control points

$wt1, wt2$, etc. are the weights associated with each control point. If wt 's are all 1 you get the plain vanilla B-spline Surface. Increasing wt on one control point draws the surface (and the parameter lines) toward that control point. In Figure 2, point 'P32' has been given a weight of 3, while all others remain at 1. This pulls the surface (and the parameter lines) toward point 'P32'. Below the profile view, the body plan view compares the $X = 10$ contour of the original, evenly-weighted model to that of the model with 'P32' weight = 3.

$knotlist1$ and $knotlist2$ control knot spacing. If not default '*' (uniform spacing) for each list, the knotlists should have $(ncu - utype + 1)$ and $(ncv - vtype + 1)$ components respectively, and each list should begin with 0 and end with 1.

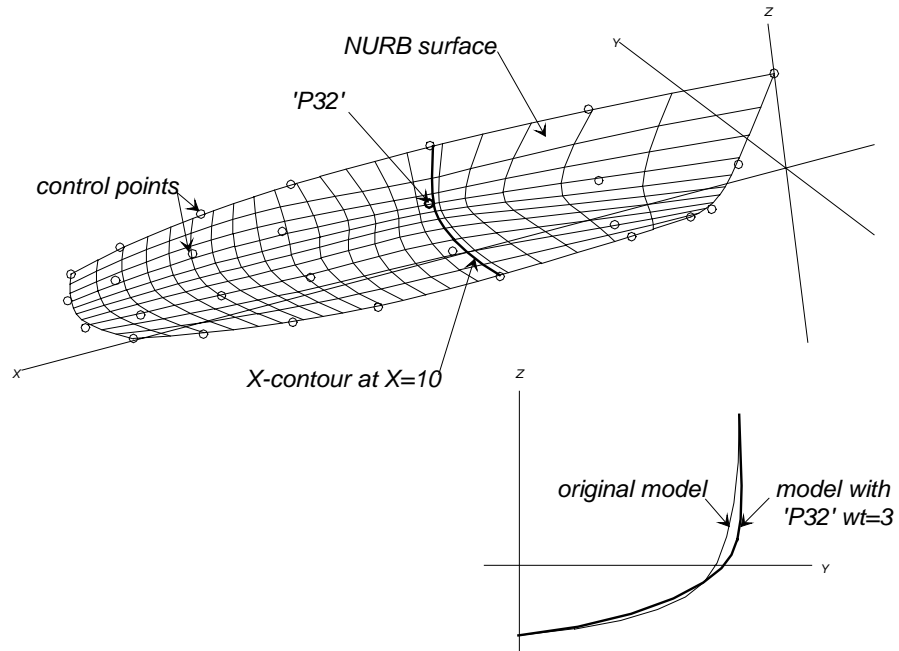


Fig. 2
A NURBSurf with the weight on one point ('P32') increased to 3.

Examples

nurbs7a.ms2, nurbs7c.ms2.

'hull' is a NURBSurf formed by 28 points ('P11' to 'P47') with all weights equal to 1. This is the same hull as in Figure 1 above, but to make that figure, we used visibility = net plus u-constant and v-constant lines. The u-direction B-splines are type-2 (quadratic) and the v-direction B-splines are type-3 (cubic). The two default knotlists distribute the knots uniformly in both the u and v directions. There are 4 control points in the u-direction (and 7 in the v-direction).

Now let's see what happens when we change knot spacing in the v-direction. This means we need to add the definition of a knotlist to the model. Since in the longitudinal direction on this hull surface we use cubic splines (vtype = 3) with ncv = 7 columns of control points, then the knotlist for the v-direction (which we named 'knotsv') needs $7 - 3 + 1 = 5$ entries, for example { 0 .25 .5 .75 1 }.

This particular knotlist would actually produce uniform B-splines; i.e., it would have the same effect as using the default '*' for knotlist2 in the NURBSurf description (Fig. 3, top).

But using:

{ 0 .45 .50 .55 1.00 } would allow a region of high longitudinal curvature near the middle ($v = .5$) of the hull

{ 0 .50 .50 .50 1.00 } would actually permit a slope discontinuity (in this case, a crease or bend) to run around the hull at $v = .5$, as in the model nurbs7c.ms2 (Fig. 3, bottom). Note that at most $type + 1$ sequential knots can have identical values.

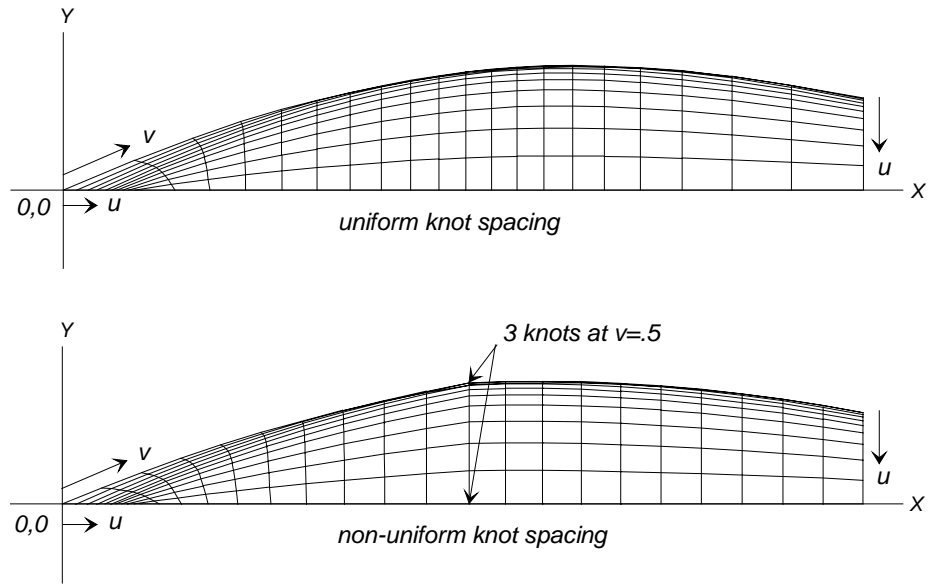


Fig. 3. Uniform (top; NURBS7A.MS2) and non-uniform (bottom; NURBS7C.MS2) knot spacing on a NURBSurf; view here is Lat 90, Lon -90 orthographic.

See also BCurve, BSurf, BLoftSurf, CLoftSurf, KnotList.

Projected Surface

Characteristic data *surface* = basis surface
mirror = a plane, Line, or point object

Description A Projected Surface is a surface formed by projecting a specified *surface* (called the “basis surface”) onto a *mirror*. Each point of *surface* is projected normally (i.e., perpendicularly) onto *mirror*. The Projected Surface therefore lies entirely in *mirror*. If the basis surface is later changed, the Projected Surface will be automatically relocated to remain in the perpendicularly-projected position.

mirror can be a plane, Line, or point object:

When *mirror* is a plane, each projection line is perpendicular to the plane.

When *mirror* is a Line, each projection line is perpendicular to the Line.

When *mirror* is a point, each projection line ends at that point.

(Note: When *mirror* is a point, each projection line passes through that point. This Projected Surface would be doubly degenerate, i.e. the entire surface would be at a single point. We do not know of any present utility in making these degenerate surfaces.)

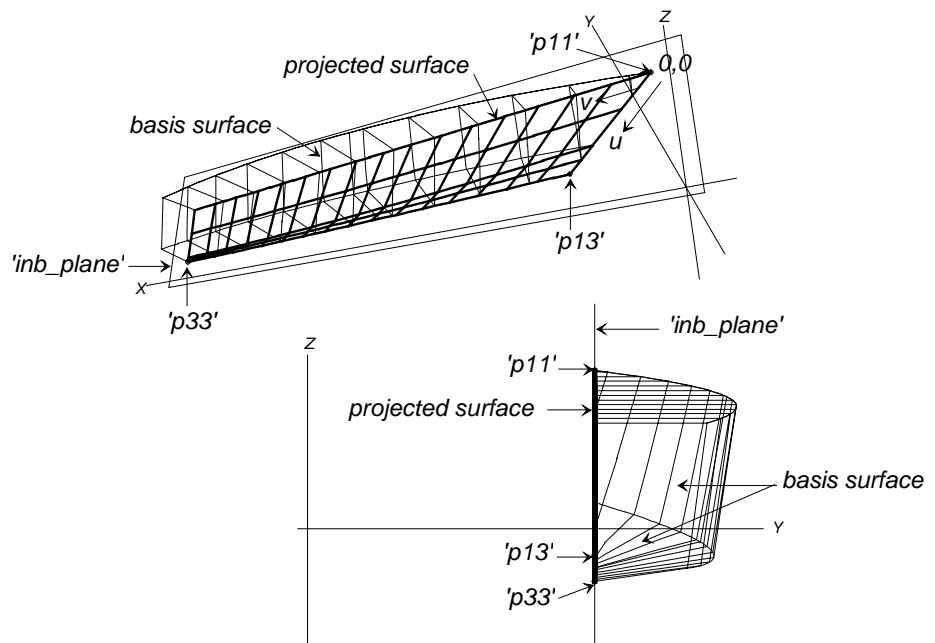
Example

projsurf.ms2

'outboard' (the basis surface) is the outboard surface for the starboard hull of a planing catamaran design. It is like half of a conventional V-bottom hull. 'inb_plane' is a plane defined by three points: 'p11' at the top of 'MC1' (the point of the stem), 'p13' at the bottom of 'MC1', and 'p33' at the bottom of 'MC3'. (Since its three control points are all at $Y = 10$, 'inb_plane' currently is the same as the YPlane at $Y = 0$.)

'inboard' (the projected surface) is the inboard surface for the starboard catamaran hull, obtained by projecting 'outboard' onto 'inb_plane'. 'inboard' is flat and parallel to the $Y = 0$ plane (the centerplane), at $Y = 10$.

Note that we could create the port hull for this catamaran by turning on Y-symmetry (Settings/ Model), as we did in the "MirrSurf" entity description.



PROJSURF.MS2 at Lat -20, Lon 60 (top) and <x> view (bottom).

See also MirrSurf, ProjPoint, ProjCurve

Relative Surface

Characteristic data *surface* = basis surface
point1, (*point2*, *point3*, *point4*) = offsets at corners of *surface* (1 or 4 required)

Description A Relative Surface is formed by offsetting the points of the basis surface, in a way similar to making a Relative Curve from a basis curve. You specify a Relative Surface by naming the basis *surface* and new positions for *either*:

- **one corner** (*point1*) – Each point of the RelSurf is offset from the corresponding point of the basis surface by a constant distance and direction – the offset of *point1* from the $u=0, v=0$ corner of the basis surface. The $u=0, v=0$ corner of the RelSurf is at *point1*.

Note that a RelSurf using a constant offset is NOT the same as creating an OffsetSurf (in which the offsets are applied *perpendicularly to the basis surface*). The perpendicular distance between a RelSurf and its basis surface would not in general be constant.

- **each of the four corners** (*point1* to *point4*) – The surface is stretched out to meet these four corner points. Intermediate points are offset by amounts linearly related to their u, v coordinates.

The order of naming the four corner points is important. The corners are identified as follows:

point1 is at $u = 0, v = 0$

point3 is at $u = 1, v = 1$

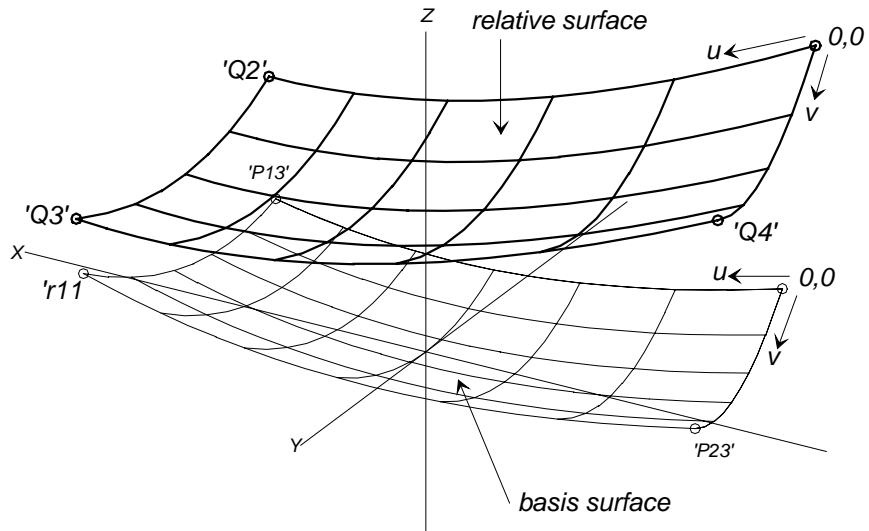
point2 is at $u = 1, v = 0$

point4 is at $u = 0, v = 1$

For most accurate results, give the RelSurf the same divisions as the basis surface.

Example

relsurf2.ms2



RELSURF2.MS2 at Lat -20, Lon 60.

'patch' is the basis surface for the relative surface 'relsurf'. 'Q1', 'Q2', 'Q3', and 'Q4' are the four corners of 'relsurf'. They are RelPoints offset from the four corner points of 'patch'. 'r11' is an AbsMagnet at the $u = 1, v = 1$ corner of 'patch'. The translation surface has no point object defined at that corner, yet one is needed in order to define the offset of the relative surface from that corner, hence the need for 'r11'.

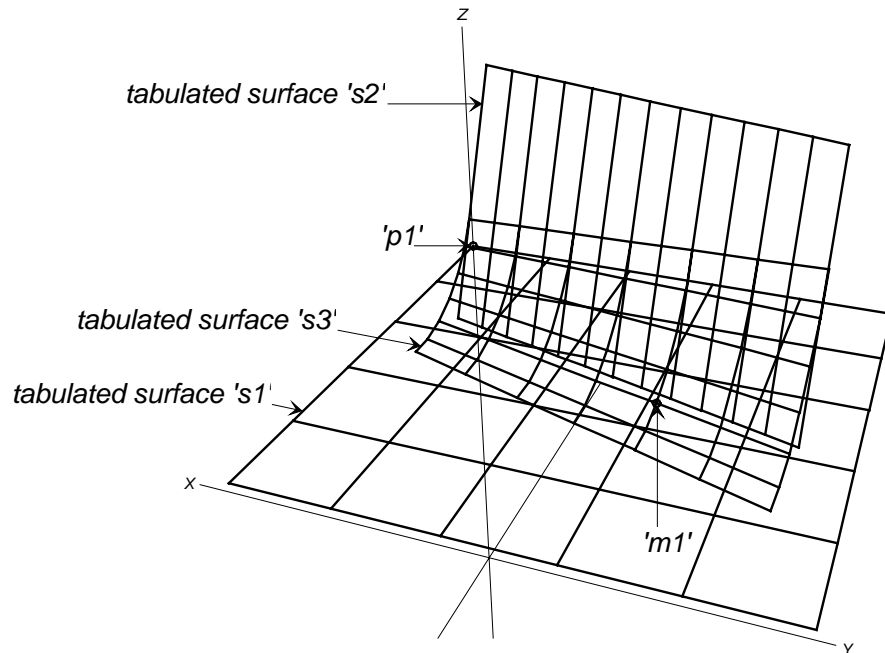
Note that in this example the surface divisions are not coordinated: 'patch' has $6 \times 3 \ 5 \times 2$, while 'relsurf' has $5 \times 3 \ 4 \times 2$ – we did this just to make the two surfaces easier to identify in black-and-white figures.

See also

OffsetSurf, RelCurve, RelSnake

Tabulated Surface

Characteristic data	<i>filename</i> = filename of .PAT file <i>patch number</i> = number of patch in file <i>frame/point</i> = name of a frame or point object or '*' for the default
Description	<p><i>filename</i> is the filename of a .PAT file. If no extension is given, SurfaceWorks assumes .PAT.</p> <p>A .PAT file (see sample contents and notes in example below) is a special case of a .3DA file, in which one or more rectangular meshes (patches) is represented by a set of $nv + 1$ polylines, each having $nu + 1$ points. Individual patches are separated by an extra record having zero pen. A polyline of $nu + 1$ points consists of one pen = 0 record followed by nu records with nonzero pen. $u\text{-divisions} \times u\text{-subdivisions}$ is required to be equal to nu, and $v\text{-divisions} \times v\text{-subdivisions}$ is required to be equal to nv, or a geometry error occurs.</p> <p><i>patch no.</i> is the number of the patch to be taken from the .PAT file: 1 for the first, 2 for the second, etc.</p> <p><i>frame/point</i> is the name of a frame or point object or '*' for the default:</p> <ul style="list-style-type: none">When <i>frame/point</i> is a <u>point</u>, it specifies an insert origin, and <i>point's</i> coordinates are added to the surface coordinates. This results in a shift (but no rotation).When <i>frame/point</i> is a <u>frame</u>, the file coordinates are interpreted as frame coordinates x,y,z. This generally results in a shift plus a rotation.When <i>frame/point</i> is <u>the default</u> '*', there is no shift or rotation. <p>The u-direction of the TabSurf is in the direction of the polylines.</p> <p>Note: A TabSurf has less inherent smoothness than most native SurfaceWorks surfaces. A TabSurf is a faceted model of the original surface and cannot be further subdivided inside SurfaceWorks.</p>
Example	<p>tabsurf.ms2 (needs fillet.pat)</p> <p>FILLET.PAT is a patch file (see partial contents below) made from a model having three surfaces: two plane patches, with a fillet surface between them. The TabSurf 's1' is the first of these patches, 's2' is the second, and 's3' is the third. The three patches use for insert location (<i>frame/point</i>) the Point 'p1' (at $X=0, Y=0, Z=1$), which has the effect of adding 1.0000 to the Z-coordinate of each point read from the file. Since <i>frame/point</i> is a point, there is no rotation of the patches, just a 1-unit shift in Z. Moving 'p1' would shift the insert location of the entire set of TabSurfs, again without rotating them.</p> <p>The magnet 'm1' is added to show that TabSurfs, like other surfaces, can parent magnets.</p>



TABSURF.MS2 at Lat 30, Lon 160.

Partial contents of fillet.pat

0	0.000	0.000	0.000	pen=0 point (1st point) on 1st polyline of 1st patch ('s1')
10	0.200	0.000	0.000	
10	0.400	0.000	0.000	
10	0.600	0.000	0.000	
10	0.800	0.000	0.000	
10	1.000	0.000	0.000	6th (last) point of 1st polyline; <i>u</i> -divisions x <i>u</i> -subdivisions for 's1' = 5; nu + 1 = 5 + 1 = 6
0	0.000	0.200	0.000	pen=0 point (1st point) on 2nd polyline
10	0.200	0.200	0.000	
10	0.400	0.200	0.000	
10	0.600	0.200	0.000	
10	0.800	0.200	0.000	
10	1.000	0.200	0.000	
0	0.000	0.400	0.000	pen=0 point (1st point) on 3rd polyline
10	0.200	0.400	0.000	
10	0.400	0.400	0.000	
10	0.600	0.400	0.000	
10	0.800	0.400	0.000	
10	1.000	0.400	0.000	
0	0.000	0.600	0.000	pen=0 point (1st point) on 4th polyline
10	0.200	0.600	0.000	
10	0.400	0.600	0.000	
10	0.600	0.600	0.000	
10	0.800	0.600	0.000	
10	1.000	0.600	0.000	

0	0.000	0.800	0.000	pen=0 point (1st point) on 5th polyline
10	0.200	0.800	0.000	
10	0.400	0.800	0.000	
10	0.600	0.800	0.000	
10	0.800	0.800	0.000	
10	1.000	0.800	0.000	
0	0.000	1.000	0.000	pen=0 point (1st point) on 6th (and last) polyline; <i>v-divisions</i> x <i>v-subdivisions</i> for 's1' = 6; $nv + 1 = 5 + 1 = 6$
10	0.200	1.000	0.000	
10	0.400	1.000	0.000	
10	0.600	1.000	0.000	
10	0.800	1.000	0.000	
10	1.000	1.000	0.000	
0	0.000	0.000	0.000	extra pen=0 record = end of patch
0	0.342	0.121	0.000	pen=0 (first) point of 1st polyline on 2nd patch ('s2')
11	0.360	0.186	0.000	
...				

See also TabPoint, TabCurve, WireFrame

X-Spline Lofted Surface

Characteristic data *curve1, curve2 ... curveN* = master curves (t-orientation doesn't matter)
type = direction of orientation of XLoftSurf
ecc = end condition code for the two directions other than the direction of orientation (specified by *type*)
graph1, graph2, graph3, graph4 = names of BGraph objects or '*' for default graph = end condition data for the two directions other than the direction of orientation (specified by *type*)

Description The XLoftSurf, like other lofted surfaces, is parented by two or more master curves. For the XLoftSurf, the number of master curves is arbitrary. The XLoftSurf is lofted with X-splines. Like the XCurve, the XLoftSurf needs end condition data, which is supplied by the four *graphs*.

type specifies the direction of orientation of the lofting X-splines:

- 1 for X orientation
- 2 for Y orientation
- 3 for Z orientation

Note: this usage of *type* has NOTHING to do with "spline" *type* as used in BLoftSurf, CLoftSurf, etc. entity specifications. XLoftSurfs use cubic splines.

ecc is the end condition code for the lofting X-splines, for the two directions other than the direction of orientation (specified by *type*):

- 0 = moment at both edges
- 1 = slope at v=0 edge, moment at v=1 edge

2 = moment at v=0 edge, slope at v=1 edge
 3 = slope at both edges

graph is the name of a BGraph object. The four BGraphs supply the end condition data for the lofting X-splines, for the two directions other than the direction of orientation (specified by *type*)

	type-1 (X orientation)	type-2 (Y orientation)	type-3 (Z orientation)	
graph1	Y' or Y''	Z' or Z''	X' or X''	at first MC
graph2	Z' or Z''	X' or X''	Y' or Y''	at first MC
graph3	Y' or Y''	Z' or Z''	X' or X''	at last MC
graph4	Z' or Z''	X' or X''	Y' or Y''	at last MC

relabel is used to control the labeling of the v-direction (the lofting curves) of the surface. See *Reference* section "Relabeling Curves and Snakes."

The type-1 XLoftSurf can reproduce exactly any FAIRLINE/2 surface. In this case, the BGraphs must be the same *type* as the end master curves and must have one component for each master curve vertex. The slope or moment columns in the FAIRLINE/2 Rep can be copied directly into the BGraphs. To *automatically* perform the translation of a FAIRLINE/2 Rep file into Points, BCurves, BGraphs, and an XLoftSurf, you can use the utility program FL2MSF (see *Appendix B*).

Crossed Master Curves

Users of XLoftSurfs should be aware that they are vulnerable to a special error condition that can cause unexpected and puzzling results: the dreaded "Crossed Master Curves" error (Error 247 or 248). To locate a point at parameters u,v on an XLoftSurf, SurfaceWorks first evaluates each master curve (and each end condition graph) at t = u, then it passes an XCurve through the points in order, then it evaluates the XCurve at t = v. The problem arises in the second stage if the points obtained from the master curves don't form a strictly ascending or descending sequence in the coordinate corresponding to *type*, e.g. the X-coordinate for type-1. The construction of the XCurve fails, and the surface is marked as in error; any descendants of the XLoftSurf get error 284.

This error can occur even when the master curves don't actually cross within the nominal u-range of 0 to 1. One way it happens is a consequence of SurfaceWorks's extending the tabulation of curves and surfaces for one subdivision beyond 0 and 1. Suppose you have specified 10 divisions and 1 subdivision in the u-direction; then SurfaceWorks actually tabulates the surface from u = -0.1 to u = 1.1. If master curves are approaching each other near the 0 (or 1) end of the range, their extensions can actually cross a little ways beyond 0 (or 1), and this "invisible" crossing will trigger error 248. Two workarounds are possible:

- (1) try increasing the subdivisions for the u-direction, since a smaller subdivision means the tabulated surface extends a shorter distance beyond u = 0 (or 1)

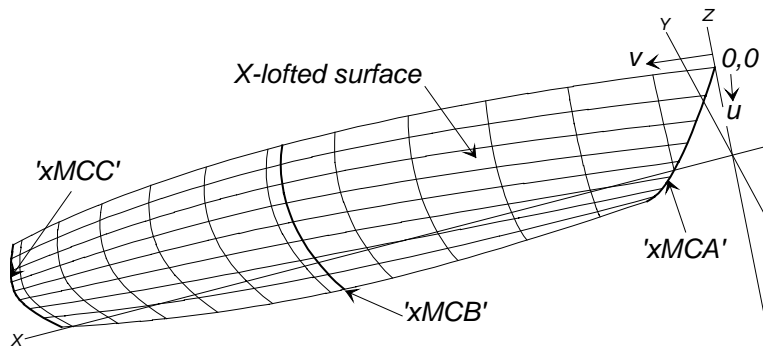
(2) change the master curves so they are farther apart at $u = 0$ (or 1)

Another way this error can be triggered is by dragging a magnet near or beyond the $u = 0$ or $u = 1$ edge of an XLoftSurf, where master curves are converging. As the program performs its search for the u, v coordinates on the surface corresponding to the visual position of the cursor, the surface might be evaluated at points outside the range $u = 0$ to $u = 1$. If the error occurs while dragging a magnet, you can usually clear it by choosing Edit/ Model File, then just <Ok> without making any changes.

Likewise, the error can occur during the solution for position of a ProjMagnet, IntMagnet, ProjSnake, or IntSnake on an XLoftSurf. It is especially important on an XLoftSurf to use a magnet to specify the starting position of one of these objects. This helps a great deal to limit the area that the program has to search in, reducing the chance that it will look beyond the $u = 0$ or $u = 1$ edges.

Example

defaultx.ms2



DEFAULTX.MS2 at Lat -30, Lon 60.

DEFAULTX.MS2 is the FL/2B Default Rep converted to a SurfaceWorks model file by the utility program FL2MSF (for FL2MSF details, see *Appendix B*). The SurfaceWorks model is formed by three type-2 B-spline master curves and an XLoftSurf. You'll probably want to use Edit/ Model File to follow the discussion. There are four BGraphs which distribute bending moment along the end master curves of the XLoftSurf. The BGraphs are type-2, the same *type* as the B-spline master curves, and they have the same number of *values* as the number of vertices in each of the master curves – 3 for xMCA and 4 for xMCC.

See also

Xcurve

Parametric Solids

All of the solids you can form with SurfaceWorks are defined with reference to a trio of parameters — u , v , and w — each of which, has a nominal range of 0 to 1. The rule that defines the solid in terms of its basis surfaces and/or curves is written in terms of these three parameters. You can think of every point of a solid as being labeled with a particular trio of values for u , v , and w . Generation of the solid is a continuous mapping of points in the unit cube to points in 3D space. We refer to the unit cube as the “ u - v - w parameter space” of the solid. It is a 3D space, each of whose points correspond to a point in the solid.

At any point in a parametric solid there is a direction (called the “ u -direction”) in which u increases while v and w are constant. Similarly there is a v -direction in which v increases while u and w are constant and a w -direction in which w increases while u and v are constant. The only reason you usually need to be conscious of which direction is which is when deciding how finely the solid should be subdivided in the three directions for tabulation and display.

A parametric solid nominally has six faces and eight corners. The eight corners have (u,v,w) values of $(0., 0., 0.)$, $(0., 1., 0.)$, $(0., 0., 1.)$, $(0., 1., 1.)$, $(1., 0., 0.)$, $(1., 1., 0.)$, $(1., 0., 1.)$, and $(1., 1., 1.)$. Some of the faces can be degenerate surfaces, i.e. curves or even points.

The distribution of the parameter t along the curves used to build surfaces supporting a solid affect the shape of the solid. In most cases, the “default” labeling works just fine. For a discussion about changing the labeling, see *Reference* topic “Relabeling Curves and Snakes”.

For some applications and examples of using solids, including how to export 3D meshes, see *Reference* topic “Using Solids”.

Note: SurfaceWorks users, please understand that support of parametric solids does NOT turn SurfaceWorks into a “solid modeler” comparable to the design programs in that category such as Pro-Engineer, SolidWorks, and Solid Edge. These solid modeling applications are based on B-rep (“boundary representation”) solids, a very versatile way to represent solid objects of arbitrary complexity. Parametric solids are a more restricted solid representation in that they must have the basic 6-face topology of a cube (just as parametric surfaces always have the 4-sided topology of a square). On the other hand, the solid shapes you can make with SurfaceWorks’s parametric solids are comparatively diverse because SurfaceWorks’s surface modeling capability is much richer than that of most solid modelers.

Additional Solid Attributes

divisions: u , v , and w divisions and subdivisions

All solids have u -divisions, u -subdivisions, v -divisions, v -subdivisions, w -divisions, w -subdivisions attributes. The number of u (or v or w) divisions \times subdivisions = the number of line segments the program uses to represent the surface in the u (or v or w) direction for display and internal tabulation. u -divisions, u -subdivisions, v -divisions, v -subdivisions, w -divisions, and w -subdivisions are all integers (1-255). The greater the *total* number of u , v , and/or w divisions, the more closely the solid screen display will approximate the solid itself (which is composed of an infinite number of points).

In addition, the number of u -divisions (or v - or w -divisions) establishes the spacing of the u -constant (or v -constant or w -constant) lines drawn in a wireframe display.

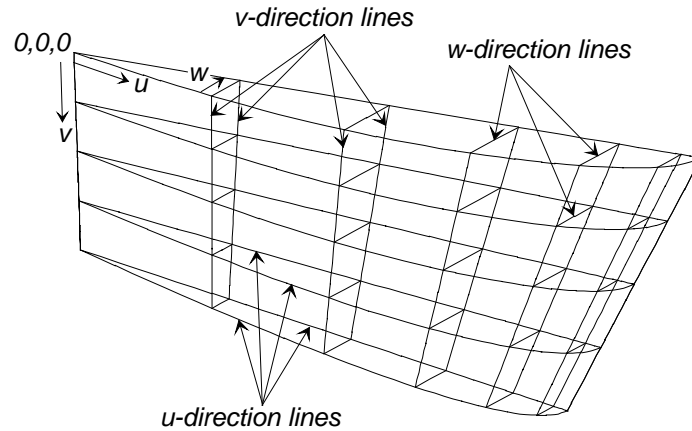


Fig. 4. u, v, and w-direction lines on a solid.

For instance, if a solid is defined with 6x4 4x2 2x2 divisions, SurfaceWorks will tabulate the solid internally with 24 segments in the u-direction, 8 segments in the v-direction, and 4 segments in the w-direction (Fig. 4).

Note: The visibility options for solids do not “parallel” those for surfaces. Solid display options show the u-direction lines (not the u= constant lines), etc. These display lines run in the same direction as the orientation marks for u, v, and w.

Orientation

Like surfaces, solids have an *orientation* attribute which can be either 0 (normal) or 1 (reversed). Solid orientation is different from surface orientation — currently its only effect is to reverse the sign of volume and weight.

Using Solids

Some applications

- Accurate weight analysis and CG for sailboat ballast (see RuledSolid examples 1 and 2)
- Tank volumes (for an example, look at the model Tanks.ms2)
- Exterior volume meshes for CFD analysis (for an example, look at the model Exterior-Grid.ms2.
- Interior volume meshes for finite element analysis (for an example, look at the model Interior-Grid.ms2.

Discussion of tank and grid examples

Tanks.ms2

This is a model for an integral tank that can be dropped into almost any hull with a single-surface bottom. It starts with 'p0', which sets the positions of the planes forming the flat forward, top, and inboard surfaces, and 'p1' which controls the length of the tank. The line 'l0' between these points is projected horizontally and vertically onto 'hull'. Each of the resulting ProjSnakes gets a ring at each end ('r00', 'r01', 'r10', and 'r11'). Lines between the rings are projected onto 'hull', then a SubSurf 's4' is made between the four ProjSnakes.

To make the RuledSolid 'fwd_tank', we need two surfaces. One is the SubSurf 's4' and the other is a degenerate RuledSurf between the line 'l0' and itself.

Tools/ Mass Properties gives the tank volume = 7.68 cubic ft.

Tools/ Weight Schedule gives the weight (and c.g.) of its contents = 958 lbs including the port-side image.

Exterior-Grid.ms2

'mesh1' is a BLoftSolid forming a body-fitted 3D mesh for CFD analysis of flow past a streamlined body. The 'body' (bright green) is a BLoftSurf made from elliptic cross-sections. The BLoftSolid has three control surfaces: 'body' (bright green; an OffsetSurf), and 'outer' (dark red; a cylindrical outer-boundary surface). Using an OffsetSurf for 'middle' ensures that the mesh is orthogonal to 'body'. The mesh is also relabeled in the w direction, creating thin cells at the 'body' surface for viscous flow analysis.

'mesh1' has $8 \times 4 \times 8 \times 4 \times 8 \times 2 = 16,384$ cells in each quadrant.

Interior-Grid.ms2

This model is a volume grid for finite-element analysis of a pulley wheel. (Turn on symmetry or View/ Render to see all the images.) 'grid' (gray; on Layer 2) is a RuledSolid between the inner and outer RevSurfs (bright green; on Layer 1).

If you turn layers 1 and 2 off and take the <y> view, you can make many parametric variations by dragging points 'p0', 'p1', 'e0', and 'p3'. The analysis grid automatically adapts.

Solid Entities

Boundary Solid

A Boundary Solid is parented by six surrounding surfaces, analogous to the way a Tangent Boundary Surface is parented by four surrounding curves. If the surfaces meet accurately along their junctions, the Boundary Solid will exactly fill the volume they bound. In the language of grid generation, Boundary Solid implements "transfinite interpolation" between its six bounding surfaces, although our implementation, using three relabels as blending functions, is more general than the standard method.

Characteristic Data	<i>u-graph</i> <i>v-graph</i> <i>w-graph</i> <i>Bounding surfaces</i>
Description	<p>Exactly six bounding surfaces are required in all cases. Some of the surfaces can be degenerate, i.e., collapsed to a curve or a single point. This allows the Boundary Solid to have the topology of a prism (one face degenerated to a curve) or a pyramid (one face degenerated to a point), for example.</p> <p>The <i>u,v,w</i> directions of the Boundary Solid depend entirely on which surface parent is first in the list of "Bounding surfaces". The first surface parent becomes the $w = 0$ face, and its <i>u</i> and <i>v</i> parameter directions become the <i>u</i> and <i>v</i> parameter directions for the solid. A non-degenerate surface should be chosen for the first parent, otherwise the orientation of the solid may be unstable, or it may not fill the volume enclosed by the other bounding surfaces.</p>

Associated errors

433. Boundary Solid: Wrong number of bounding surfaces (6 required).

Sample files

Boundary Solid1.ms2 is a simple rectangular block whose faces are 6 BSurfs.

Boundary Solid-Prism.ms2 is a prismatic block with a degenerate BSurf 's5' as one of its faces. If you move 's5' to the top of the list of parents, the resulting solid does not fill the volume enclosed by the surface parents.

BlockSolid

A BlockSolid is a very simple solid requiring minimal parents. It is a rectangular solid, oriented by a frame and otherwise supported by just two points at two opposite corners.

Characteristic Data

u-Relabel, v-Relabel, w-Relabel -- optional relabels for the 3 parametric directions

Frame

Point1

Point2

Description

The block is oriented parallel to the Frame coordinates. Its *u*-, *v*- and *w*-directions are along the Frame's *x*-, *y*- and *z*-axes respectively. Its $u=0, v=0, w=0$ corner is at Point1, and its $u=1, v=1, w=1$ corner is at Point2. The three Relabels apply to the three coordinate directions.

Associated errors

None

Sample files

BlockSolid2.ms2 is the simplest example of a block. It uses the default frame '*', so its faces are parallel to the global coordinate system.

B-spline Lofted Solid

Characteristic data

surfaces = control surfaces

type = type for the lofting B-splines; 1, 2, 3, etc.

relabel = name of Relabel object or '*' for default labeling

Description

The BLoftSolid is a solid between two or more supporting surfaces. It is generated by evaluating the point on each of the surfaces at each given u,v and constructing a B-spline curve from the points. The BLoftSolid interpolates its first surface (*surface1*) and last surface (*surfaceN*), but in general not the others, which have a "guiding" or "shaping" effect like the interior control points of a B-spline Curve

Technically, a BLoftSolid should have at least one more control surface than its *type*, e.g. at least 2 for type-1, 3 for type-2, etc; therefore, a BLoftSolid should have at least one more supporting surface than its *type*. However, SurfaceWorks will automatically "demote" a spline to lower type as necessary to fit the number of control points. For example, a type-3 BLoftSolid specified with only 2 surfaces will be treated as type-1; with 3 surfaces it will be treated as type-2; only with 4 or more surfaces will it actually be a cubic solid.

The supporting surfaces must have similar u,v orientation — the 0,0 corners need to be in a relatively similar place, and u and v must run in similar directions (otherwise the solid could wind up with an unexpected twist).

u and v for the solid run in the same directions as u and v for each supporting surface. w is the parameter along the lofting B-splines.

relabel is used to control the labeling of the w -direction (the lofting curves) of the solid. The default *relabel* '*' produces the "natural" labeling, in which the parameter w is concentrated in highly curved areas formed by closely-spaced control surfaces. See *Reference* section "Relabeling Curves and Snakes."

Example

bloftsolid1.ms2

This BLoftSolid is made from three supporting surfaces:

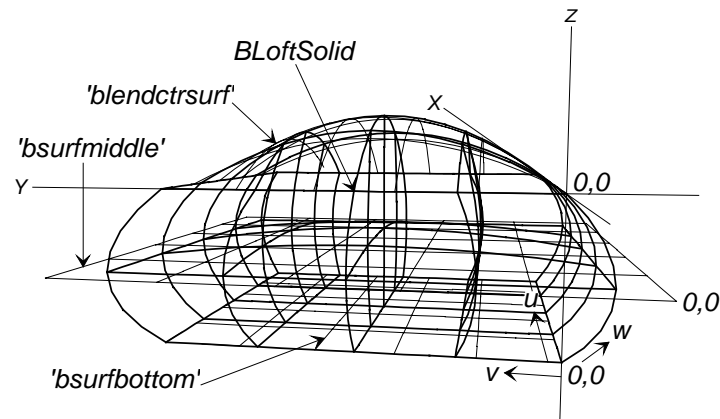
surface1 = 'blendctrsurf' (cyan), a Centerpoint Boundary Surface

surface2 = 'bsurfmiddle' (green), a flat BSurf

surface3 = 'bsurfbottom' (magenta), another flat BSurf

Each of the lofting B-splines has three control points, taken from the point at the same u,v on each of the three surfaces. These lofting B-splines are type-2. As you can see, the BLoftSolid passes through its first and last supporting surfaces, but not its middle one. The w parameter for

the solid runs along these B-splines; $w = 0$ is at the first supporting surface and $w = 1$ at the last one.



BLOFTSOLID1.MS2 at Lat 6, Lon -10.

Bsolid

A BSolid (or B-spline solid) is constructed from control points in a way that is very parallel to BCurve and BSurf entity types, but extended to one more dimension.

Characteristic Data

u-Relabel, v-Relabel, w-Relabel -- optional relabels for the 3 parametric directions

u-type, v-type, w-type -- B-spline types for the 3 parametric directions

ncu -- number of control points in the u direction

ncv -- number of control points in the v direction

Control points -- (ncu x ncv x ncw of them)

(BSolid has a "Net" visibility option, like the "Polygon" option for a BCurve, or the "Net" option for a BSurf.)

Associated errors

436 BSolid: ncu, ncv, ncw must each be 2 or greater

437 BSolid: The number of control points is not evenly divisible by ncu x ncv.

Sample models

BSolid1.ms2 is a minimal example of a BSolid, with just 8 control points. Its B-spline type is specified as 3 (cubic) in all 3 directions, but because there are only 2 control points in each of the 3 directions, all the B-splines are automatically demoted to degree-1 (linear).

CopySolid

The CopySolid entity is the extension of the concepts of CopyPoint, CopyCurve, and CopySurf to one higher dimension.

Characteristic Data

Solid

Frame0 -- source frame

Frame1 -- destination frame

x-scale, y-scale, z-scale -- scale factors for the 3 frame axes

Associated errors

None

Sample files

CopySolid1.ms2 is an example of a CopySolid 'o1' made as a mirror image of another solid 'o0', reflected in the X=0 plane. RPYFrame 'F0' is located at MirrPoint 'p9', is left-handed (Orientation = 1) and has Pitch of 180 degrees, so its three axes are all parallel to the global X,Y,Z axes, but the x-axis is reversed.

Ruled Solid

Characteristic data

surface1 and surface2 = surface parents

relabel = name of Relabel object or '*' for default labeling

Description

The RuledSolid is a solid between two supporting surfaces. It is generated by taking the point on each of the surfaces at each given u,v and constructing the line ("ruling") between. w is the parameter along the line.

The supporting surfaces must have similar u,v orientation — the 0,0 corners need to be in a relatively similar place, and u and v must run in similar directions (otherwise the solid could wind up with an unexpected twist).

u and v for the solid run in the same directions as u and v for each supporting surface. w is along the rulings. from *surface1* to *surface2*.

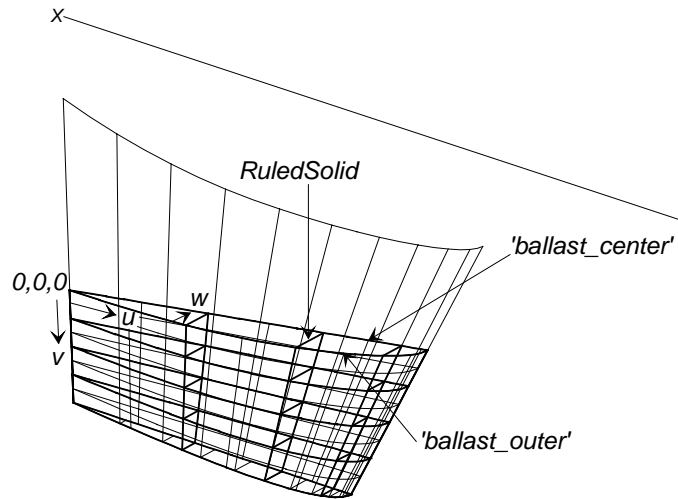
relabel is used to control the labeling of the w-direction of the solid. The default *relabel* '*' produces linear labeling with respect to w.

Example 1

ruledsolidballast1.ms2

This is the outside lead ballast casting for a trapezoidal fin keel. The lead line is adjustable by dragging rings 'r0', 'r1'. You can have the weight schedule open and watch weight and C.G. change as you adjust the lead line.

The lead line at the keel surface is an IntSnake. The SubSurf 's0' below the leadline is projected to centerplane (ProjSurf 's1'), then 's0' and 's1' are the parents for the RuledSolid.



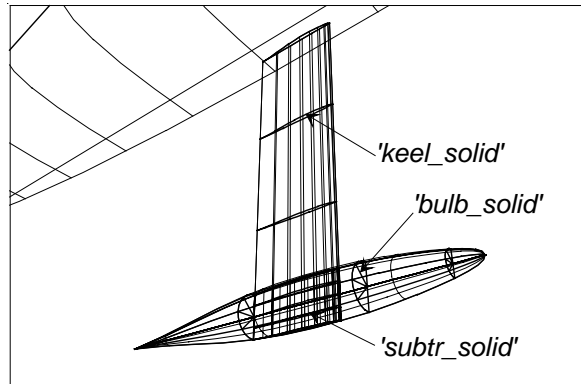
RULEDSOLIDBALLAST1.MS2 at Lat 25, Lon 52.

Example 2 ruledsolidballast2.ms2

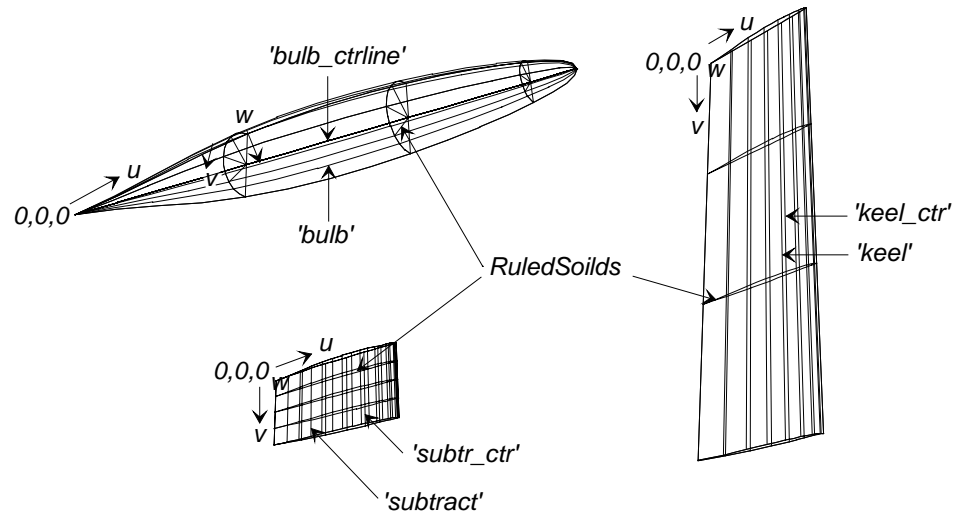
Example 2: ruledsolidballast2.ms2

This is a bulb keel using solids to get weight and C.G. of the fin and bulb combination. 'bulb_solid' is a RuledSolid between the bulb surface and a deliberately degenerate RuledSurf that lies along the bulb axis (projection of 'meridian' on axis line). 'keel_solid' is a RuledSolid between the fin surface and a ProjSurf on the centerplane. Note that 'keel_solid' has orientation = 1 to give it a positive volume and weight.

The missing lead where the fin passes through the bulb (we assume here the fin goes all the way through the lead) is accounted for by 'subtr_solid'. The weight schedule gives the correct weight and C.G. for the combination.



RULEDSOLIDBALLAST2.MS2 at Lat -30, Lon 50.



The three solids in RULEDSOLIDBALLAST2.MS2. The w parameter on the keel and the subtraction solids are transverse lines from 'keel' and 'subtract', respectively, to the centerplane.

See also [BLoftSolid](#), [RuledSurf](#)

Triangle Mesh Entity Types

Surface Works 5.0 has partial support for a new class of entity types called Triangle Meshes. A Triangle Mesh is a form of surface, fundamentally consisting of an assembly of triangles that join edge-to-edge. A Triangle Mesh can also be subjected to an optional smoothing/refining operation known as subdivision, which produces successively finer meshes. Points and curves can be embedded in a Triangle Mesh, analogous to magnets, snakes and rings on a surface.

Triangle meshes are much less constrained topologically than parametric surface patches. They can have holes, and any number of sides, or even no sides -- seamless closed bodies are easy to make.

Triangle meshes are included in Surface Works 5.0 primarily for the possibilities they offer in flattening with the add-on Flattener product. An STL or RAW file can be imported, resulting in a Light Triangle Mesh. So far, they have no other import and export options.

Triangle Mesh basics

The basic constituents of a Triangle Mesh are a set of points, called *nodes*, and a set of triangles, each made from 3 distinct nodes. The nodes are numbered with an index, 1 to $nnodes$. The triangles are numbered with another index, 1 to $ntris$.

Each triangle has 3 *edges* -- lines connecting 2 nodes. Each edge can be shared with at most one other triangle. An edge that is not shared with another triangle is a *boundary edge*, and is part of the boundary of the Triangle Mesh.

The triangles must be sufficiently connected by shared edges so you can reach any triangle in the set from any other by stepping from one triangle to the next, always crossing shared edges. For example, two triangles sharing just a single node do not make a valid Triangle Mesh, nor do two triangles that don't share any nodes.

Unlike a surface, which has an invisible extension outside the 0-to-1 parameter range, a Triangle Mesh has no extension; when you come to the boundary, you're really at the boundary.

Recursive subdivision

Subdivision is an operation applied to a Triangle Mesh, that results in another Triangle Mesh with 4 times the number of triangles. The nodes for the subdivided mesh are derived from the original nodes by an averaging or smoothing operation

In each stage of subdivision, a new node is generated near the middle of each existing edge, and each original triangle is replaced by 4 triangles.

Recursive subdivision is the repeated application of the subdivision operation. Any Triangle Mesh entity has a *degree* property that controls its degree of recursive subdivision. *degree* is a non-negative whole number. The default value is 0, which means "no subdivision". The unsubdivided mesh is often referred to as the "coarse mesh".

Locations on a Triangle Mesh

A triangle mesh does not have an overall coordinate system like the u,v -parameters of a parametric surface. However, within each triangle it has a local coordinate system that can be used to locate points on the surface. Locations in a triangle are specified by so-called *barycentric coordinates* u, v, w which act as weights applied to the 3 corners of the triangle. In the triangle interior each of u, v and w is in the range of 0 to 1, and they satisfy the constraint: $u + v + w = 1$, or (equivalently) $w = 1 - u - v$.

Thus the combination {triangle index, u, v } is sufficient to uniquely identify a point located on a Triangle Mesh, because w can be expressed as a function of u and v .

Figure: Barycentric coordinates in a single triangle

Common data

Orientation. Any Triangle Mesh entity has a normal orientation property with one of the two values: Normal and Reversed. This is very similar to the orientation property of a surface; it just defines which direction will be considered the positive normal.

Weight/unit area. A Triangle Mesh has a weight/unit area property that is completely analogous to weight/unit area for a surface. It is multiplied by the Triangle Mesh area for inclusion in the Weight Schedule.

Visibility options. The edges of component triangles can be made visible or hidden.

Kind. A Triangle Mesh has a *kind*, which is not yet used for anything. The only valid value for *kind* is 0.

Degree. The degree of subdivision, a nonnegative whole number.

Triangle Mesh

Characteristic data	<i>degree</i> = degree of subdivision <i>Points</i> = A list of points that are the nodes. <i>Triangles</i> = A list of integers, which are node indices in groups of three; each group of three is a triangle. <i>Breaks</i> = A list of node indices that identify breakpoints and breaklines.
Description	This is the most basic form of Triangle Mesh, where the nodes are explicit point entities, and the connection into triangles is explicitly specified. A subdivided TriMesh (degree > 0) is related in shape to its coarse-mesh nodes in a way that is similar to the relationship of a B-spline Surface or NURBS surface to its net of control points. The surface generally doesn't pass through its coarse-mesh nodes, but instead acts as if it is attracted to them. TriMesh has a "net" visibility option which just shows the coarse triangulation, to help visualize this relationship. A major difference is that, while the control net of a B-Spline surface has to be organized into orderly rows and columns of control points, the TriMesh control net has a much freer topology -- it can be any valid triangle mesh.
Example	TriMesh1.ms2 Triangle Mesh 'tm' is made from 5 points, named 'p1' to 'p5' (turn on point nametags). The points are nodes 1 to 5. As you can see in the Triangles data, there are 4 triangles, with node indices: 1, 2, 3; 2, 5, 3; 5, 4, 3; 4, 1, 3. The <i>Degree</i> is 0, i.e., no subdivision is being applied. Joint.ms2 A Trimesh model of a human leg. Change the degree to get better definition.

Light Triangle Mesh

Characteristic data	<i>degree</i> = sub-division amount <i>Frame</i> = Parent location
----------------------------	---

Scale Factors, X, Y, and Z

Nodes = A list of real values which are node coordinates in groups of three; each group of three is the x, y, z coordinates of one node.

Triangles = A list of integers which are node indices in groups of three; each group of three is a triangle.

Breaks = A list of node indices that identify breakpoints and breaklines.

Description

This is another basic way to represent a Triangle Mesh as a list of nodes and a list of triangles. Rather than depending on point entities for its nodes, a LtTriMesh is a self-contained single entity. This makes it relatively compact. It is implemented primarily for purposes of importing large triangle meshes from other systems.

Also see: Triangle Mesh; Light NURBS Surface

Example

LtTriMesh1.ms2

This is the same example as TriMesh1.ms2, but cast in the form of a Light Triangle Mesh. Frame coordinates for the five supporting points are in the Nodes section. The Triangles section is the same as in TriMesh1.ms2 - - node indices for 4 triangles.

Surface Triangle Mesh

Characteristic data

degree = degree of subdivision

surface = Surface, Trimmed Surface or Composite Surface

nu = no. of triangles in u-direction

nv = no. of triangles in v-direction

style = 0 - Forward 1 - Reverse 2 - Alternating

Description

The SurfaceTriangle Mesh entity type, parented by a parametric or trimmed surface, or Composite Surface. This creates a Triangle Mesh from a surface. It has 3 integer parameters: *nu*, *nv*, *style*. These parameters have no effect when the parent is a TrimSurf or CompSurf -- in those cases, you just get the object's triangulation. When the parent is a parametric surface, it is divided first into rectangles (in u,v parameter space) along u=constant and v=constant parameter lines, then each rectangle is split into two triangles.

nu controls the number of triangles in the u direction, with these options:
nu < 0 -- uniform subdivision into -*nu* strips, each covering 1/(-*nu*) of the u parameter space.

nu = 0 -- similar uniform subdivision as *nu*<0, but taking *nu* from the basis surface u-divisions x subdivisions

nu > 0 -- the surface is first divided at any degree-1 breaklines, then the intervals between breaklines are uniformly subdivided into a total of *nu* strips

nv has similar meaning for the v-direction subdivision

style selects forward diagonals, reverse diagonals, or alternating diagonals for the division of each rectangle into two triangles.

Example**SurfTriMesh1.ms2**

'patch' (the basis surface) is a Translation Surface. Surface Triangle Mesh 'I0' has $nu = 8$, $nv = 4$, and $style = 0$. It therefore divides 'patch' into $8 \times 4 \times 2 = 64$ triangles. These are uniformly distributed in both u and v directions, because 'patch' has no degree-1 breaklines.

Copy TriMesh**Characteristic data**

degree = degree of subdivision
Triangle mesh = Parent Triangle Mesh entity
Frame1 = Parent location
Frame2 = Copy Location
Scale Factors, X, Y, and Z

Description

Copy entities takes a parent of the same class (the parent entity); two frames; and three scale factors. The two "environments" are two frames, designated *Frame0* (the "source" frame) and *Frame1* (the "destination" frame). The process of constructing the Copy object has three steps:

- 1 transform the basis object into frame coordinates in *Frame0*;
- 2 apply the three stretching factors to the x, y, z coordinates in *Frame0*;
- 3 create the Copy at those same x, y, z coordinates in *Frame1*.

All this sounds like a lot of Frames. Although Frames are versatile and not hard to create, remember that any point can serve as a frame; its x, y, z axes are parallel to the global X, Y, Z axes respectively. In practice, some 90% of Copy entities will use points for one or both frame parents.

Also see: Copy Curve, Copy Surface

Example**CopyTriMesh1.ms2**

'tm' (the parent Triangle Mesh) is a 5-node, 4-triangle Triangle Mesh entity similar to TriMesh1.ms2. 'I0' is a Copy Triangle Mesh made from 'tm', with the two points 'p3' and 'p6' as the two frames. 'I0' has an x -scale factor of 0.5, so it is only half as wide in the X -direction as its parent.

PolyTriMesh**Characteristic data**

degree = sub-division amount
Log-tolerance = Base-10 logarithm of distance tolerance
Triangle meshes = Parent list

Description

The Poly Triangle Mesh entity type is two or more Triangle Mesh objects assembled into a single Triangle Mesh. The component Triangle Meshes currently need to have compatible triangulations, with nodes agreeing within a tolerance specified by the *log-tolerance* parameter.

Also see: Poly Surface

Example

PolyTriMesh1.ms2

Poly Triangle Mesh 'I0' is assembled from two parent Triangular Meshes: 'tm1' (6 nodes, 4 triangles) and 'tm2' (9 nodes, 8 triangles). Since the two parents share 3 point entities as their common nodes, the value of *Log-tolerance* is not significant in this example.

Triangle Mesh Magnet

Characteristic data

Triangle mesh = Host Triangle Mesh entity
Triangle no. = A triangle index in the host Triangle Mesh
u,v = Barycentric coordinates in the selected triangle

Description

A Triangle Mesh Magnet is a point located on a Triangle mesh by {triangle index, u, v} location in the coarse mesh. It stays in the same relative position when the host Triangle Mesh is subdivided. Dragging is implemented in SurfaceWorks.

Example

TMBSnake2.ms2

This model has 7 Triangle Mesh Magnets supporting two Triangle Mesh B-spline Snakes.

Triangle Mesh Projected Magnet

Characteristic data

Point = Point entity to project
TMMagnet/TriMesh = designates the host Triangle Mesh entity that the Triangle Mesh Projected Magnet will lie on.
Mirror/TriMesh. = Plane, line, point or Triangle Mesh entity.

Description

A Triangle Mesh Projected Magnet is a point located on a Triangle mesh by projection of another *Point* along a straight line.

When *Mirror/TriMesh* is a plane, the projection line is normal (perpendicular) to the plane.

When *Mirror/TriMesh* is a line, the projection line is normal (perpendicular) to the line.

When *Mirror/TriMesh* is a point, the projection line radiates from the point.

When *Mirror/TriMesh* is a Triangle Mesh (it has to be the same Triangle Mesh that the Triangle Mesh Projected Magnet is to lie on) plane, the projection line is along the local normal direction to the Triangle Mesh; the Triangle Mesh Projected Magnet will be at the closest point of the Triangle Mesh to *Point*.

TMMagnet/TriMesh can be either a Triangle Mesh Magnet or Ring, or a Triangle Mesh entity.

When *TMMagnet/TriMesh* is a magnet or ring, it serves two purposes:

(1) It specifies which Triangle Mesh the Triangle Mesh Projected Magnet is to lie on -- the host Triangle Mesh of the magnet or ring.

(2) It gives a starting location for the iterative search for the projection point. In the case of multiple intersections of the host Triangle Mesh with the line of projection, the Triangle Mesh Projected Magnet will (usually) be made at the intersection nearest to the starting location (in terms of distance measured within the Triangle Mesh, not 3-D distance).

When *TMMagnet/TriMesh* is a Triangle Mesh, it specifies directly which Triangle Mesh the Triangle Mesh Projected Magnet is to be hosted on. In case of multiple intersections of the projection line, the starting location for the search will be at triangle 1 of the host Triangle Mesh, and the search will usually converge to the intersection closest to triangle 1.

Example

TMProjMagnet1.ms2

Triangle Mesh Projected Magnet 'G1' (bright red) is the vertical projection of Point 'P0' onto the Surface Triangle Mesh 'I0', using the system plane *X=0 for the mirror.

Triangle Mesh Projected Magnet 'G2' (bright white) is the normal projection of Point 'P0' onto the Surface Triangle Mesh 'I0', by using 'I0' as the *Mirror/TriMesh* parent.

Triangle Mesh Ring

Characteristic data *TMSnake* = A Triangle Mesh Snake entity
 t position = location along snake

Description A Triangle Mesh Ring is a point located by *t* parameter value along a Triangle Mesh Snake. It can serve to mark a location along the snake; it also serves as a location on the host Triangle Mesh.

Dragging of Triangle Mesh Rings is implemented in Surface Works.

Example

TMBSnake1.ms2

This model has four Triangle Mesh Rings (bright blue) marking locations on two different Triangle Mesh B-spline Snakes (magenta) for the ends of SubSnakes (yellow and bright cyan).

Snakes on Triangle Meshes

There is a class of Triangle Mesh Snakes representing curves embedded in a Triangle Mesh. These are highly analogous to snakes on surfaces. There is a variety of possible ways to specify them, represented by 5 different entity types.

NOTE: Triangle Mesh snakes are curves, but for technical reasons in Surface Works 5.0 they can't serve as curve parents for other entities (for example, as master curves for a surface).

Triangle Mesh B-spline Snake

Characteristic data *Type* = B-spline degree: 1 = linear, 2 = quadratic, 3 = cubic
Periodic = Switch controlling periodic property; 0 or 1
TMMagnets/TMRings = Control points; Triangle mesh magnets or rings.
magnet1, magnet2,...magnetN

Description A Triangle Mesh B-spline Snake is a continuous curve embedded in a Triangle Mesh entity and governed by a set of Triangle Mesh magnets as control points. The magnets must all be hosted by the same Triangle Mesh. Except for degree-1, the snake does not generally pass through its control points, but instead behaves as if it's attracted to them. Moving one control point produces a local modification of the curve shape that is strongest in the vicinity of that control point.

At the time of Surface Works 5.0 release, there is no difference in the behavior of degree 2 and degree 3 for this entity type.

Setting the periodic property makes the snake reuse its control points in a cyclic fashion, so the snake is a closed curve ($t = 0$ and $t = 1$ are at the same location), and in fact is periodic (t and $t+N$ are at the same location, where N is a positive or negative integer). The $t=0$ location on a periodic snake is generally close to magnet1, but the snake does not generally pass through magnet1.

A non-periodic Triangle Mesh B-spline Snake does begin at magnet1 and end at magnetN, similar to a B-spline Curve or Snake.

For details about B-spline degrees and how they behave, see "B-spline entities -- Degrees".

See also: B-spline Curve, B-spline Snake

Example

TMBSnake2.ms2

This model has a total of four Triangle Mesh B-spline snake examples. The host Triangle Mesh 'tm' is similar to LtTriMesh1.ms2, but has a subdivision degree of 2.

'N2' (yellow) is a degree-1 snake with 3 control magnets, 'G0', 'G1', and 'G2'. 'N0' (magenta) is the degree-2 snake made from the same 3 magnets.

'N3' (yellow) is a degree-1 periodic snake made from 4 control magnets 'G3' to 'G6'. 'N1' (magenta) is the degree-2 periodic snake made from the same 4 magnets.

Triangle Mesh Edge Snake

Characteristic data	<i>Type</i> = Counterclockwise (0) or Clockwise (1) <i>TMMagnet1</i> = Triangle Mesh Magnet/Ring for the $t = 0$ end <i>TMMagnet2</i> = Triangle Mesh Magnet/Ring for the $t = 1$ end
Description	<p>A Triangle Mesh Edge Snake is a curve that follows the boundary of a Triangle Mesh, between two Triangle Mesh Magnets. The two magnets must be hosted by the same Triangle Mesh entity; they must both lie on the boundary; and if the Triangle Mesh has multiple boundaries, the two magnets must be on the same boundary.</p> <p>Two boundary magnets divide a boundary loop into two parts. <i>Type</i> determines which of those parts becomes the Triangle Mesh Edge Snake.</p>
Example	<p>TMEdgeSnake1.ms2</p> <p>This model has two Triangle Mesh Edge Snake examples. The host Triangle Mesh 'tm' is similar to the Trimesh1.ms2 example, but has a subdivision degree of 2. The two Triangle Mesh magnets 'G0' and 'G1' (bright red) lie on the boundary of 'tm'. Edge Snake 'N0' (magenta) goes from 'G0' to 'G1' in the counterclockwise direction. Edge Snake 'N1' (yellow) is made from the same two magnets, in the same order, but, being type-1, goes in the clockwise direction.</p>

Triangle Mesh Intersection Snake

Characteristic data	<i>Type</i> = 0 to 1 <i>TMMagnet</i> = a Triangle Mesh magnet or ring <i>Mirror/Surface</i> = plane, frame, surface, Line or point. Defines a family of potential cutting planes or surfaces. <i>Point</i> = point entity. Specifies the actual cutting plane or surface.
Description	<p>A Triangle Mesh Intersection Snake is a curve on a Triangle Mesh, located where a real or virtual cutting plane or surface intersects the Triangle Mesh.</p> <p>The TMMagnet parent serves three roles:</p> <ul style="list-style-type: none">It specifies which Triangle Mesh the Intersection Snake is cut on (i.e., the Triangle Mesh that is host to the TMMagnet)If there are multiple intersections, the one closest to the TMMagnet is selected.The end of the selected intersection that is nearest to the TMMagnet becomes the $t = 0$ end of the Intersection Snake.

The surface doing the cutting can be the actual plane, frame (its x,y-plane) or surface specified by *Mirror/Surface*, if *Point* lies in that plane or surface; or it can be a virtual cutting surface offset from *Mirror/Surface* and passing through *Point*. (For details, see "Specifying the cutting surface".)

The *Type* property is significant only when the intersection is a closed curve. Then it is used to determine the clockwise/ counterclockwise orientation of the Intersection Snake.

Example

TMIntSnake1.ms2

'tm' is a Triangle Mesh similar to TriMesh1.ms2. The parents of Triangle Mesh Intersection Snake 'N0' are:

Triangle Mesh Magnet 'G0' (red)

System plane *Y=0

Bead 'e0' (bright white)

'G0' specifies the host Triangle Mesh for the snake ('tm', host of 'G0'), and selects which end of the cut will be the $t = 0$ end of the snake. The cut is with a plane parallel to *Y=0, passing through 'e0'. (2-point Plane 'a0' is included in the model to show the cutting plane. but is not directly involved in the Intersection Snake.)

Triangle Mesh Projected Snake

Characteristic data

Curve = curve entity

TMMagnet/TriMesh = Triangle Mesh magnet or ring, or Triangle mesh entity. Designates the Triangle Mesh on which the Projected Snake will lie.

Mirror/TriMesh = plane, frame, line, point, or Triangle mesh entity

Draft Angle = Draft Angle (degrees) -- Not currently implemented

Description

A Triangle Mesh Projected Snake is a projection of *Curve* onto a Triangle Mesh. Each point of *Curve* is projected along a straight line to locate the corresponding point on the Projected Snake. The projection can be done with any valid type of mirror, or can be normal to the Triangle mesh host:

When *Mirror/TriMesh* is a plane, the projection lines are normal (perpendicular) to the plane.

When *Mirror/TriMesh* is a Line, the projection lines radiate from the Line and are normal (Perpendicular) to the Line.

When *Mirror/TriMesh* is a point, the projection lines radiate from the point.

When *Mirror/TriMesh* is a Triangle Mesh (it must be the same Triangle Mesh the projected Snake is to lie on), the projection is along the local normal direction to the Triangle Mesh. Each point of the Projected Snake will be at the foot of a line from the corresponding point on Curve

dropped normally to the Triangle Mesh. (Usually this is the closest point on the Triangle Mesh.)

TMMagnet/TriMesh specifies the Triangle Mesh the Projected Snake is to lie on. When *TMMagnet/TriMesh* is a Triangle Mesh magnet or ring, it serves two purposes:

It specifies which Triangle Mesh the Projected Snake is to lie on: the Triangle Mesh that is host to *TMMagnet*.

It provides a starting location on the Triangle Mesh for the iterative search that locates the first point (the $t=0$ end) of the Projected Snake. This is especially important in case of multiple intersections of the Triangle Mesh with the projection line; then the location of *TMMagnet* will help to select the correct one of two or more possible Projected Snakes.

Whenever you create a Triangle Mesh Projected Snake, we recommend you use a magnet or ring for the *TMMagnet/TriMesh* parent, locating it reasonably near where the Projected Snake is to begin.

When *TMMagnet/TriMesh* is a Triangle Mesh, it specifies directly the host triangle mesh for the Projected Snake. The search for the intersection point at the $t = 0$ end of the snake will start at the first triangle of the Triangle Mesh.

Example

TMProjSnake2.ms2

'tm' is a dome-shaped Triangle Mesh entity with 7 nodes, 6 triangles, and 2 degrees of subdivision. *Curve* is a B-spline Snake 'n0' (magenta) drawn on the B-spline Surface 's0'. Triangle Mesh Projected Snake 'N0' (yellow) is the vertical projection of 'n0' onto 'tm', using the system plane *Z=0 as mirror. Triangle Mesh Magnet 'G1' (bright red) identifies the host Triangle Mesh for the Projected Snake (its host, 'tm'), and provides a starting location for the projection of its $t = 0$ end.

Triangle Mesh SubSnake

Characteristic data

Degree = blending function B-spline degree, 1 to --

Direction = Normal or Reversed

TMRings = ring1, ring2, ... ringN

Description

A Triangle Mesh SubSnake is a portion of a parent Triangle Mesh snake, between ring1 ($t = 0$ on the SubSnake) and ringN ($t = 1$ on the SubSnake). In most cases, 2 TMRings will be sufficient ($N = 2$). When more than 2 rings are used as parents ($N > 2$), the intermediate rings control the labeling of the SubSnake, using B-spline blending in a fashion similar to SubCurve.

Notice that if ring1 is at a larger t -value on the parent snake than ringN, the t parameter on the SubSnake will run in the opposite direction from the t parameter on the parent snake.

The Triangle Mesh SubSnake is hosted on the Triangle Mesh that is host to its parent *TMRings*. The *TMRings* must all be on the same Triangle Mesh snake (the parent snake). Predefined rings *0 and *1 can be used for any of the *TMRings*, **except** at least one of the *TMRings* has to be an actual bead or ring, in order to specify the parent snake.

Degree is the B-spline degree for the blending functions: 1 = linear, 2 = quadratic, 3 = cubic, etc. As usual, if the number of parents N is less than *Degree*+1, the splines are automatically demoted to degree N+1.

The *Direction* property is active only when the parent snake is periodic. Then there are two possible ways to get from ring1 to ringN -- clockwise and counterclockwise. When *Direction* is Normal, the subsnake will use the part that goes around in the host snake's positive t direction; when *Direction* is Reverse, it will use the other part.

Example

TMBSubSnake1.ms2

This model has three example Triangle Mesh Subsnakes.

'N2' (yellow) is between Triangle Mesh Rings 'W0' and 'W1' (bright blue) on Triangle Mesh B-spline Snake 'N0' (magenta).

'N3' (yellow) is between Triangle Mesh Rings 'W2' and 'W3' in the Normal direction, i.e. its t parameter runs in the same direction as the periodic parent snake, Triangle Mesh B-spline Snake 'N1' (magenta).

'N4' (bright cyan) is also between Triangle Mesh Rings 'W2' and 'W3', but is in the Reverse direction, i.e. its t parameter runs counter to the t parameter of the parent snake 'N1'.

Formulas

SurfaceWorks 5.x has a profound new capability for generation of parametric families of geometric models, in a new class of *real-valued objects*: the Variable and Formula entity types. This is a complex topic, full of new information and rich with new possibilities.

Constants, variables and formulas

There are three ways to specify a real number in the syntactic definition of an object: with a constant value, a variable or a formula. Variables and formulas are objects. A variable carries a value that lies within a specified range; a formula involves an expression made of variables, constants, functions and formulas.

Constants

Constants are 1.23, -0.7, 12e-8, +98, etc. Although they have no units they are always considered to carry the adequate unit needed at the particular location where they are used.

For instance in

```
Point P 1 1 / 1 2 3 ;
```

1, 2 and 3 are considered as having length units.

In

```
AbsMagnet M 1 1 / S 0.123 0.456 ;
```

the u and v parameter values 0.123 and 0.456 are considered to be unitless.

Variables

Please see the document “What’s New in SurfaceWorks 5.0” for complete Variable documentation.

Formulas

A formula expresses a computation from constants, variables and other formulas with operators and functions; it carries units resulting from its expression.

Basic syntax

The simplest syntax for a formula is:

Formula *name* / { *expression* } ;

For instance

Formula *c* / { 177.69 } ;

Formula *f* / { *x* } ;

Formula *s* / { *x* + *y* } ;

Formula *s* / { *x* + 5 * *y* } ;

Expression

Operators

The expression of a formula can be made with the usual operators ‘+’ (addition), ‘-’ (subtraction), ‘*’ (multiplication), ‘/’ division. Exponentiation is symbolized with ‘^’ and is evaluated right to left, i.e. x^y^z is $x^{(y^z)}$ and not $(x^y)^z$. Parentheses can be used to prioritize the evaluation.

Formula *d* / { $b^2 - 4*a*c$ } ;

Formula *P* / { $R*I^2$ } ;

Formula *V* / { $n * R * T / P$ } ;

Formula *s* / { $(1-t^2)/(1+t^2)$ } ;

Functions

A set of mathematical functions and object accessors is available:

Name	Argument(s)	Result	Synopsis
SIN	1, radian (unitless)	Unitless	<i>Sine</i>
SIND	1, degree (unitless)	Unitless	<i>Sine</i>
COS	1, radian (unitless)	Unitless	<i>Cosine</i>
COSD	1, degree (unitless)	Unitless	<i>Cosine</i>
TAN	1, radian (unitless)	Unitless	<i>Tangent</i>

TAND	1, degree (unitless)	Unitless	<i>Tangent</i>
ATN	1, unitless	Radian (unitless)	<i>Arc tangent</i>
ATND	1, unitless	Degree (unitless)	<i>Arc tangent</i>
ATN2	2, both with same any units	Radian (unitless)	<i>Arc tangent(y/x)</i>
ATN2D	2, both with same any units	Degree (unitless)	<i>Arc tangent(y/x)</i>
LOG	1, unitless	Unitless	<i>Natural logarithm</i>
LOG10	1, unitless	Unitless	<i>Decimal logarithm</i>
EXP	1, unitless	Unitless	<i>Exponential</i>
SQRT	1, unit dimensions multiple of 2	Units dimensions of argument divided by 2.	<i>Square root</i>
ABS	1, any units	Same units as argument	<i>Absolute value</i>
ROUND	1, any units	Same units as argument	<i>Rounding</i>
MIN	2, both with same any units	Same units as arguments	<i>Minimum</i>
MAX	2, both with same any units	Same units as arguments	<i>Maximum</i>
TPOS	1, bead	Unitless	<i>T parameter</i>
UPOS	1, magnet	Unitless	<i>U parameter</i>
VPOS	1, magnet	Unitless	<i>V parameter</i>
XPOS	1, point	Length	<i>X coordinate</i>
YPOS	1, point	Length	<i>Y coordinate</i>
ZPOS	1, point	Length	<i>Z coordinate</i>
DIST	2, point, point	Length	<i>Distance between points</i>
CLEAR	2, point, graphic object	Length	<i>Clearance</i>
ANGLE	3: point, point, point	Unitless (degree)	<i>Angle of three points</i>
ARCLEN	3: curve, unitless, unitless	Length	<i>Arc distance along curve</i>
GRAPH	2: graph, unitless	Unitless	<i>Evaluation of graph</i>
FRAMEPOS	3: point, frame, index (1-3, for x,y,z coordinate)	Length	<i>Coordinates in frame</i>
AREA	2; surface, use_sym (0 or 1)	Area = L ²	<i>Area of surface</i>
VOLUME	2; solid, use_sym (0 or 1)	Volume = L ³	<i>Volume of solid</i>
CENTROID	3; object, use_sym (0 or 1), index (1-3, for X,Y,Z coordinate)	Length	<i>Coordinates of centroid</i>
IF	3: any units	Same as units of selected argument	<i>If arg1 <=0, arg2, else arg3</i>

Examples

Formula I / { A * SIN(omega * t + phi) } ;

Formula x / { EXP(COS(phi)) } ;

Formula y / { $\text{SQRT}(\text{ABS}(b^2 - 4 * a * c))$ } ;

Formula z / { $\text{TAN}(c * \text{ANGLE}(p1, p2, p3) + \text{phi}^2) / b$ } ;

Unit consistency

Rules

Variables or formulas cannot appear anywhere inside a formula: unit consistency must be observed. The units of the result of a formula must be specified and must match the units generated by the expression. The rules are:

1. Only variables and formulas of same units can be added or subtracted.
2. Exponentiation of an expression with a rational exponent (this includes the *SQRT* function) is only possible if the result has integral unit dimensions, e.g. $\text{SQRT}(x^2 + y^2)$ is valid, $(x^3)^{(1/4)}$
3. Constants are considered to have the same units as the other operand in a sum and no units in a product, e.g. in $1 - x$, 1 is considered to have the same units as x . *This is offered as a facility: no physically meaningful formula has unit-less additive constants, since the formula would become inconsistent in case of unit scaling.*
4. Function arguments and results have unit dimensions as specified in the preceding chapter.

Unit mismatch is an error. Unit consistency is checked at parse time when possible (the unit dimensions of x^y cannot be determined without knowing y 's value). It is however completely checked at evaluation time.

Examples

Point V1 1 1 / 1 2 3 ;

Point V2 1 1 / 4 5 6 ;

Formula $V1 \cdot V2$ / { $\text{XPOS}(V1) * \text{XPOS}(V2)$
 $+ \text{YPOS}(V1) * \text{YPOS}(V2)$
 $+ \text{ZPOS}(V1) * \text{ZPOS}(V2)$
} L^2 ;

Variable Intensity / 2.5 I (0,10) ;

Variable Resistance / 100 $L^2 M T^{-3} I^{-2}(0,500)$;

Formula DPotential / { Resistance * Intensity² } $M T^{-3} I^{-2}$;

Evaluation logic

No simplification

Expressions are not simplified, therefore

Formula f / { $x - x$ } ;

depends upon x (twice!), which gets evaluated when f needs to be.

No optimization

No optimization is performed which would prevent parents from being evaluated, e.g. in

Formula f / { $x * y$ } ;

y will be evaluated even if x is 0.

Complete syntax

```
'Formula' name { attr, } { 'R:1' } '/' '{' expr '}' { units } ';' 
```

Due to the fact that numeric constants are needed in formulas, and that MS2 syntax allows object names such as 12, 4e3, etc., objects with such names will not be recognized inside the expression of a formula, between { and }; these symbols will be considered as numeric constants instead. Likewise, objects having the same name as a function (*SIN*, *SQRT*, *XPOS*...) will not be recognized.

Variables and formulas as parents

Variables and formulas can of course be parents of formulas. But they can be parents of all objects depending on a real value, and wherever an object regardless of its type can be a parent, such as in an object list. Therefore they can be parents of points (as coordinates, offsets, angles, t-, u- and v-parameter), curves and surfaces (via knot lists and weights), etc.

```
Variable A / 10 L ;  
Variable B / 5 L ;  
Variable C / 30 L ;  
Variable D / 25 L ;  
Point P1 14 1 / 0 0 0 ;  
Point P2 14 1 / A 0 0 ;  
Point P3 14 1 / A B B ;  
Point P4 14 1 / C B B ;  
Point P5 14 1 / C 0 0 ;  
Point P6 14 1 / D 0 0 ;  
Variable k0 / 0.25 ;  
Variable k1 / 0.75 ;  
KnotList k1 / { 0 k0 0.5 k1 1.0 } ;  
Variable w0 / 1 ;  
Variable w1 / 3 ;  
NURBCurve spline0 11 5 64x1 / * 2 k1 { P1 1 P2 w0 P3 w1 P4 1 P5 1 P6 w0 }  
;
```

Note that since a variable is defined by three real numbers, these can be replaced by variables or formulas: you can for instance have a variable with a variable range:

```
Variable min / 0 ;  
Variable max / 24 ;  
Variable x / 7 (min,max) ;
```

In this case the objects used as value and range bounds must all carry the same units (as always, constant are assigned the units that match).

User interface

Figure 2 shows the properties for the newly created Formula 'f1'. Note the Unit dimensions are specified as "Length". Figure 3 is the Expressions dialog. The DIST expression was chosen from the "Functions" drop-down and the two points were added from the "Entities" drop-down. The expression was completed, by adding the closed parenthesis from the dialog, or your keyboard.

When you are editing a real value in the data for any object, you can use an appropriately dimensioned real object (Variable or Formula) instead of a constant. For example, creating or editing a RadiusArc, one required data element is the Radius. When you select this item a constant or a real object can be entered. Figure 4 shows a Radius Arc with the Radius field active and a radius of 1. To enter a Formula, go to the Surfer View and choose the appropriate Formula. (Figure 5) Figure 6 shows Formula 'f1' as a parent, which has a value of 0.500.

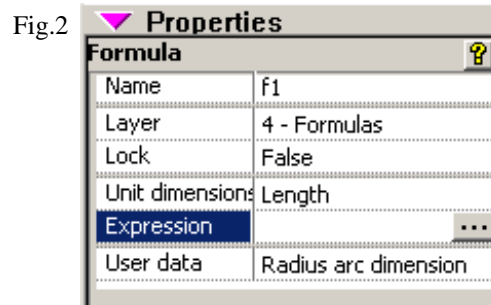


Fig. 3

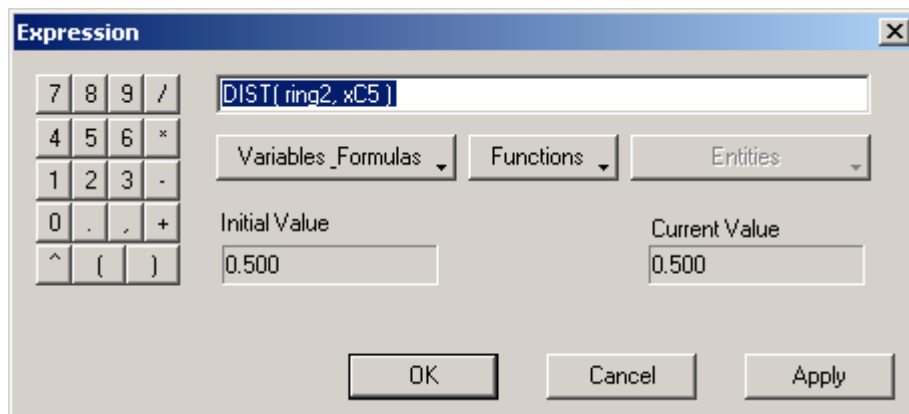


Fig. 4



▼ Properties	
Radius Arc 	
Name	Arc1
Color	
Visible	True
Layer	0
Lock	False
Relabel	*
Type	Tangent to lines Point1-
Radius	1.000
Point1	Bead9
Point2	q6
Point3	Bead4
Divisions	8
Subdivisions	4
Show tickmark	False
Show polyline	False
Weight/unit len	0.000
User data	

Fig. 5

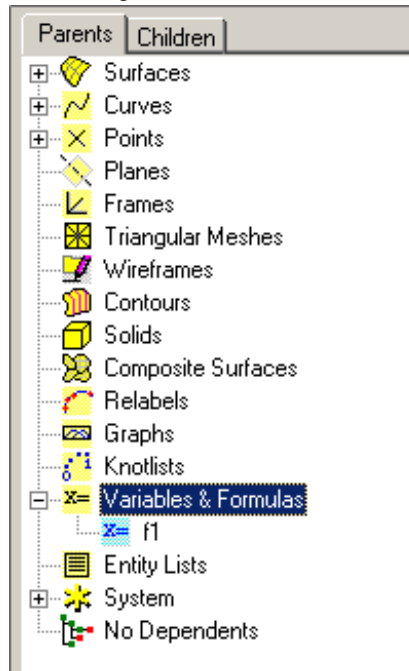




Fig. 6

▼ Properties	
Radius Arc 	
Name	Arc1
Color	
Visible	True
Layer	0
Lock	False
Relabel	*
Type	Tangent to lines Point1-
Radius	f1 (0.500)
Point1	Bead9
Point2	q6
Point3	Bead4
Divisions	8
Subdivisions	4
Show tickmark	False
Show polyline	False
Weight/unit len	0.000
User data	

Example Application

The following syntax generates the profile of an I-beam given four parameters: *Height*, *Width*, *FlgThk*, *WebThk* (note how *FlgThk*, *WebThk* are upper-bounded).

```
Variable Height / 100.000 L ( 10.000, 200 ) ;
Variable Width / 70.000 L ( 10.000, 150 ) ;
Variable FlngThk "flange thickness" / 20.000 L ( 0.000, Height ) ;
Variable WebThk / 20.000 L ( 0.000, Width ) ;
Formula d1 / { ( Width + WebThk ) / 2 } L ;
Formula d2 / { Height - FlngThk } L ;
Formula d3 / { ( Width - WebThk ) / 2 } L ;
FrameAbsPt BotFlgLLC 14 1 / * 0.000 0.000 0.000 ;
FrameAbsPt BotFlgLRC 14 1 / * Width 0.000 0.000 ;
FrameAbsPt BotFlgURC 14 1 / * Width FlngThk 0.000 ;
FrameAbsPt WebLRC 14 1 / * d1 FlngThk 0.000 ;
FrameAbsPt WebURC 14 1 / * d1 d2 0.000 ;
FrameAbsPt TopFlgLRC 14 1 / * Width d2 0.000 ;
FrameAbsPt TopFlgURC 14 1 / * Width Height 0.000 ;
FrameAbsPt TopFlgULC 14 1 / * 0.000 Height 0.000 ;
FrameAbsPt TopFlgLLC 14 1 / * 0.000 d2 0.000 ;
FrameAbsPt WebULC 14 1 / * d3 d2 0.000 ;
FrameAbsPt WebLLC 14 1 / * d3 FlngThk 0.000 ;
FrameAbsPt BotFlgULC 14 1 / * 0.000 FlngThk 0.000 ;
BCurve I_profile 11 1 12x1 / * 1
    { BotFlgLLC BotFlgLRC BotFlgURC WebLRC WebURC TopFlgLRC
    TopFlgURC
    TopFlgULC TopFlgLLC WebULC WebLLC BotFlgULC BotFlgLLC } ;
```