Chapter 1.
Advanced Surfacing Component

Component: *Skinning and Lofting

The Advanced Surfacing Component (AS), in the skin directory, provides techniques for creating 2D geometry (a surface or face) by interpolating a sequence of 1D geometry (edges, coedges or wires), arbitrarily positioned in model space. Three functional variations of this are provided in the Advanced Surfacing Component:

- Lofting
- Skinning
- Net surfaces

(*Covering*, which creates a surface from a closed loop of edges, is another type of surfacing technique. This functionality is provided in the Covering Component.)

Lofting

Lofting fits a surface through a series of curves (coedges) and the surface associated with each coedge, creating a sheet body or a solid body. It provides control over the tangents of the surface.

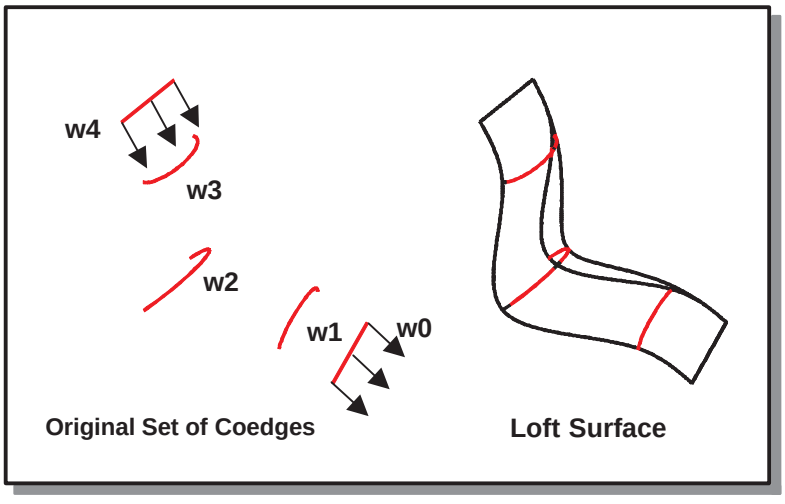

Figure 1-1. Lofting

Skinning

Skinning fits a surface through a series of disjoint curves (wire bodies), creating a sheet body or a solid body. The solid body can be open or closed, depending on the curves used as input.

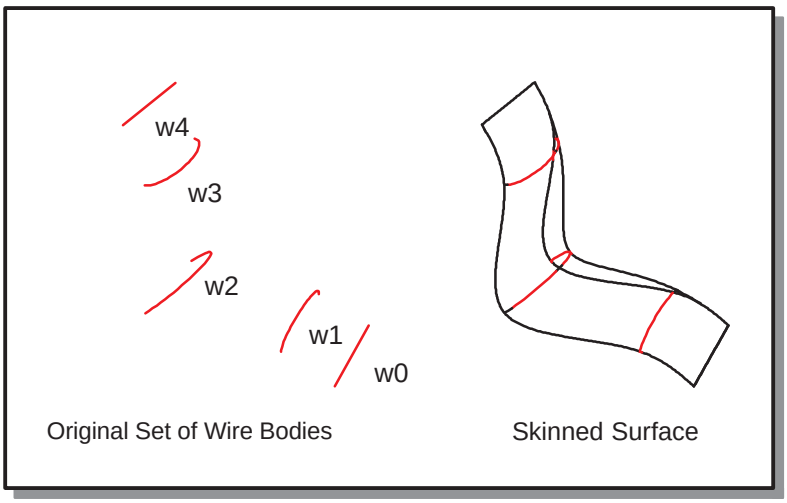

Figure 1-2. Skinning

Net Surfaces

Net surfaces stretches a surface across a “grid” of curves, and supports curve directional alignment and simplification to a plane, when appropriate. A net surface can handle wires with multiple coedges that have a G1 connection.

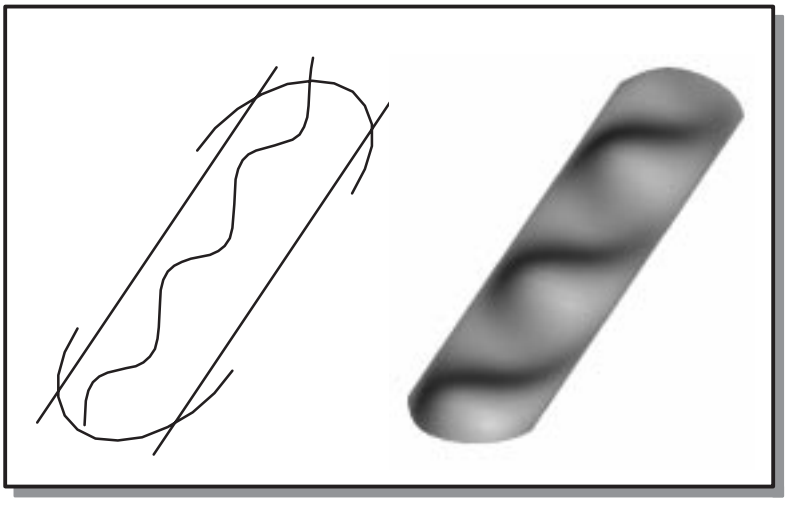

Figure 1-3. Net Surface

Lofting

Topic: *Skinning and Lofting

Lofting interpolates a surface through a given set of curves. The curves provide the *u* parameter of the resulting surface. The *v* parameter is computed and generated by the lofting algorithm. This technique allows the formation of relatively complex surfaces from a set of cross sectional curves. Lofting has been widely used for decades in the shipbuilding, automotive, and aircraft industries.

Lofting controls how the surface will pass through the set of input curves by controlling the magnitudes and directions of the tangent vectors going into and out of each curve. These tangents are controlled by coedge surface tangents or laws. Figure 1-1 shows an example of lofting five wires (w0–w4) with wire w0 and w4 having tangent constraints.

ACIS laws are supported in the lofting API and Scheme interfaces. A vector field can be specified using a law to control the take-off vector on any coedge in the profile of a loft surface. Lofting surfaces that do not contain a surface or a law can be simplified.

In general, lofting:

- Supports surface creation between coedges of wires and surfaces
- Controls the take–off vector weight factors for the loft transitions
- Supports laws to define coedge tangency constraints
- Supports surface isoparametric or arc length parameterization

- Can align the direction of the cross section curves so that they are in the same direction as the first curve
- Supports twist and non–twist minimization of the cross-section
- Supports control of the take–off vector direction, perpendicular to the coedge or in the loft direction
- Supports guide curves
- Can simplify the created surface to an analytical surface (planar, conical, spherical, or toroidal), if applicable
- Can create a closed or open solid body

Lofting Parameters

Topic: *Skinning and Lofting

Scheme Extensions: section

C++ Data Structures: Loft_Connected_Coedge_List.

A lofting operation requires two basic categories of parameters (arguments). The first category defines the profiles to be lofted plus information to control the tangents through the curves that the loft surface must follow. The second category defines optional parameters to control what type of object is formed as a result of the loft operation.

Profiles

The main API for lofting is api_loft_coedges. As input it takes a list of the C++ data structure Loft_Connected_Coedge_List. Each structure in the list corresponds to one lofting profile or cross section. This structure contains the following data items:

- The number of coedges in the lofting cross section
- An array of pointers to coedges that form a connected chain
- The lofting cross section orientation
- The take–off vector weight factor
- A list of laws associated with each coedge in the list (optional)

A minimum of two Loft_Connected_Coedge_Lists is required for each loft. If a closed solid body is required, at least three distinct Loft_Connected_Coedge_Lists must be provided. If the first and last of these lists are identical, a closed solid loft body is automatically constructed.

The ending point of any coedge in the list of connected coedges should be the starting point of the next coedge in the list, first and last coedges excepted. In other words, the coedges are ordered in the list as they appear next to each other in model space. There cannot be any gaps between adjacent coedges.

When the coedges in the list are associated with a surface, the cross section orientation uses REVBIT to set the direction “out of” or “into” the starting surface. The expression “out of” means that the lofted surface is going *out of* the given ending surface, while “into” means that the lofted surface is going *into* the ending surface. When lofting between two surfaces, the starting surface needs to have its REVBIT set to 0 (for “out of”, the default) and the ending surface needs to have its REVBIT set to 1 (for “into”). When lofting between multiple surfaces, the first surface has its REVBIT set for 0, while all others are set for 1. When lofting between a surface and an edge, REVBIT for both can be 0. The orientation bit is ignored if laws are present.

Options

In addition to the coedge list, the main lofting API takes options to control the surface parameterization, coedge twist, coedge direction, take–off direction, simplification, and creation of a solid or closed body. See *Skinning and Lofting Options* for more information.

The take–off vector can be weighted to affect the shape of the loft. Figure 1-4 illustrates the effect of the take–off vector weight factor on lofting between two surfaces. If a small value for the take–off vector factor is used, the transition from the tangent to the lofted surface happens abruptly. If a large value is used, the transition from the tangent to the lofted surface happens more gradually. Extremely high weight values could result in excessive whipping in the lofted surface, or a self–intersecting surface.

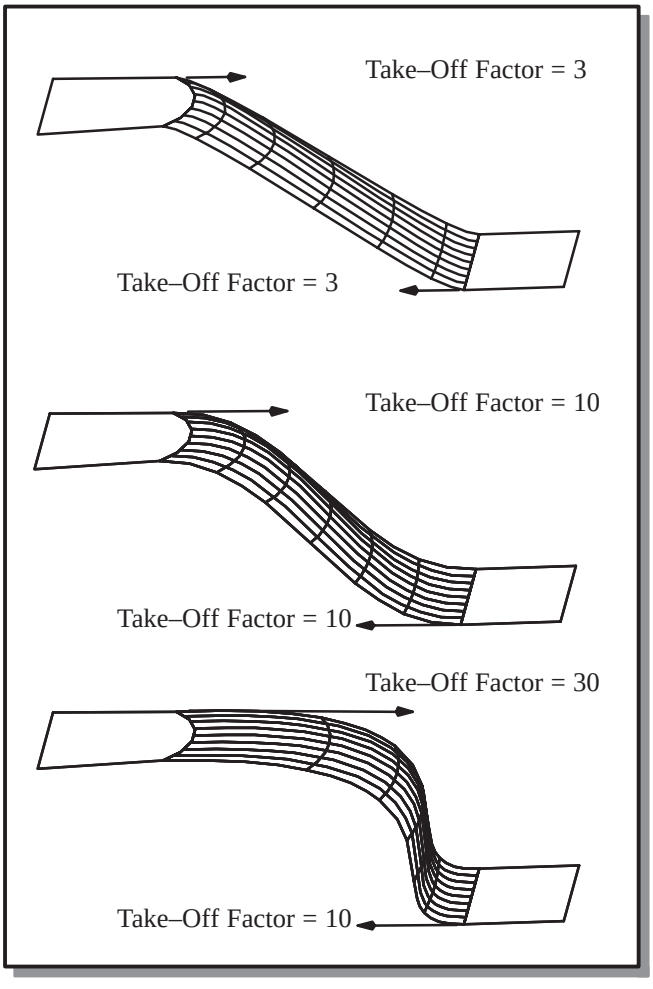

Figure 1-4. Take-Off Vector Factor

If desired, the global option loft_estimate_tanfacs can be set, which invokes an algorithm that finds an optimum factor to scale the weight for the tangent factors based on a minimum radius of curvature of the lofted body as a whole. This not only helps to create more pleasing surfaces but insures greater ability to shell and blend lofted models. If you set this option to TRUE, the user–supplied weight values in each Loft_Connected_Coedge_List are then scaled by the optimum scale factor found.

An array of law pointers is associated with each coedge in the coedge list. These vector laws are the tangency conditions of an edge. If a law isn't used to control the tangency of an edge, its pointer should be NULL. In the case in which the law is NULL and the coedge has an associated surface than the surface conditions are used to control the tangency condition. If the Loft_Connected_Coedge_List has valid laws, than the laws take precedence over surface conditions.

Lofting Surfaces

Topic: *Skinning and Lofting

Scheme Extensions: section, sheet:loft–wires

C++ APIs: api_loft_coedges

The curves used as input to the lofting function are given in terms of coedges. Since coedges can be associated with a face which contains surface information, tangent vectors from this surface can be used in defining the skin surface. These surface tangents associated with coedges are used with "tangent factors" to control magnitudes that define how the surface passes through a given input curve.

If a surface is used as a cross sectional curve, the surface determines the tangent of the skin surface when it meets the corresponding section.

The resulting sheet body consists of one or more faces linked together to form a shell. The loft surface is precise and does not have any internal C1/G1 discontinuities. Figure 1-5 shows an example of creating a lofted sheet body between two circular faces, one of which has a smaller diameter.

Scheme Example

```
(option:set ”cone_par” #t)
(option:set ”sil” #f)
(option:set ”u_par” 5)
(option:set ”v_par” 7)
(define face1 (face:cylinder (position 0 0 0)
    (position 0 20 0) 40))
(define face2 (face:cylinder (position 0 50 0)
    (position 0 70 0) 50))
(zoom–all)
(define coedges1 (entity:coedges face1))
(define coedges2 (entity:coedges face2))
(define section1 (section (list
    (list–ref coedges1 1)) #f 1))
(define section2 (section (list
    (list–ref coedges2 0)) #t 1))
(define sheet1 (sheet:loft–wires
    (list section1 section2)))
```

Example 1-1. Lofting between Surfaces

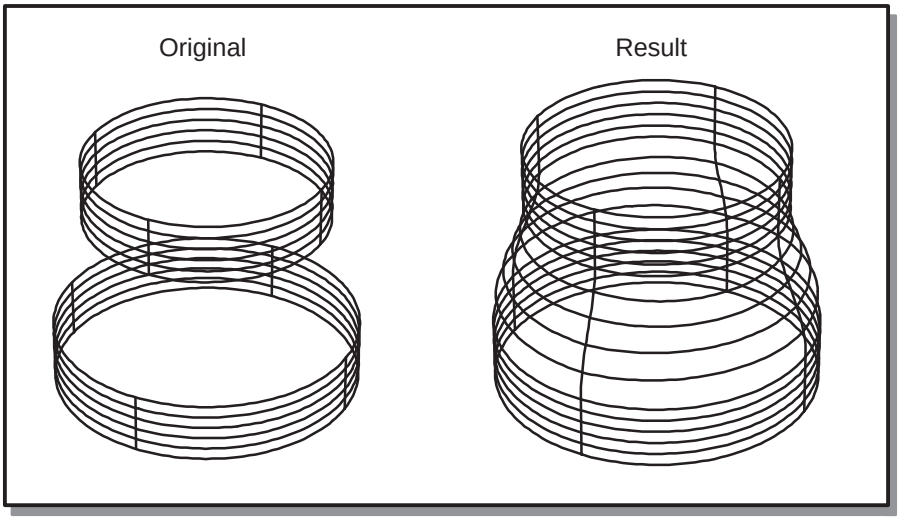

Figure 1-5. Lofting between Surfaces

Lofting with Laws

Topic: *Skinning and Lofting, *Laws

Scheme Extensions: section, law, sheet:loft–wires

C++ APIs: api_loft_coedges

An ACIS law can be associated with each of the input curves to control how the surface passes through the curve (coedge).

Laws are part of the lofting API and Scheme interfaces (because the Test Harness does not support laws, the loft command does not support this functionality). Any vector field can be specified for the take–off vector for any or all of the coedges in the loft surface profiles. This type of surface control is frequently used to build aerospace type loft surfaces (e.g., airfoils). Figure 1-6 shows an illustration of law vector fields.

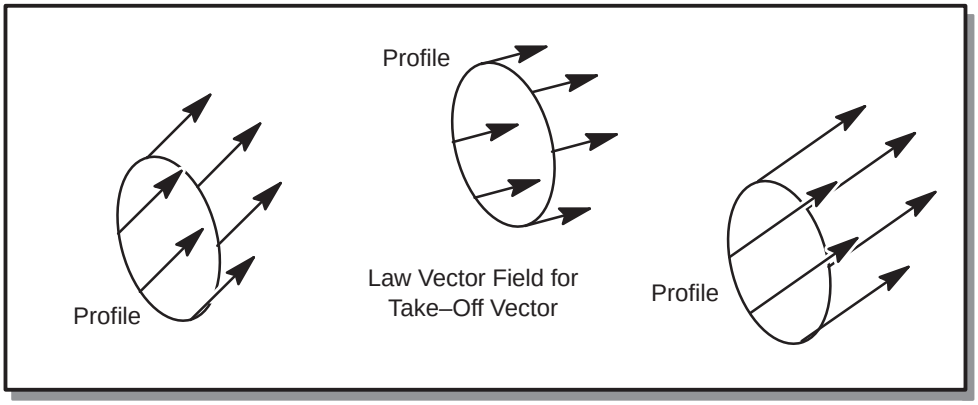

Figure 1-6. Law Vector Field

Scheme Example

```
(part:clear)
(option:set "u_par" 5)
(option:set "v_par" 7)
(define w1 (wire-body (edge:circular
    (position 0 0 0) 40 0 360)))
(define w2 (wire-body (edge:circular
    (position 0 50 100) 40 0 360)))
(define w3 (wire-body (edge:circular
    (position 0 0 200) 40 0 360)))
(define law1 (law "vec(0,-.5,.5)"))
(define dom1 (law "domain(law1,0,6.28)" law1))
(define law2 (law "vec(0,0,1)"))
(define dom2 (law "domain(law1,0,6.28)" law2))
(define law3 (law "vec(0,.5,.5)"))
(define dom3 (law "domain(law1,0,6.28)" law3))
(define c1 (entity:coedges w1))
(define sec1 (section c1 (list dom1) #f 200))
(define c2 (entity:coedges w2))
(define sec2 (section c2 (list dom2) #f 200))
(define c3 (entity:coedges w3))
(define sec3 (section c3 (list dom3) #f 200))
(define loft1 (sheet:loft-wires (list sec1 sec2 sec3)))
```

Example 1-2. Lofting with Laws

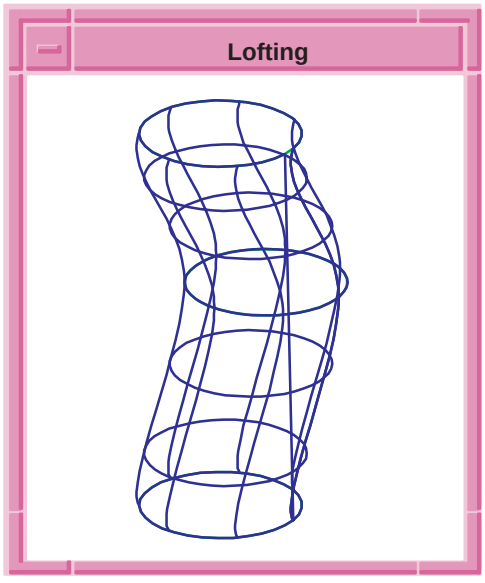

Figure 1-7. Lofting with Laws

Loft Between Faces

Topic: *Skinning and Lofting

Scheme Extensions: solid:loft–faces

C++ APIs: api_loft_faces

The api_loft_faces function is a higher–level feature for uniting two bodies using lofting. The bodies can be solid bodies or sheet bodies.

If one of the bodies is a solid, the operation is different than if it were a sheet body. In the case of a solid body, the face provided as an argument is removed from the body. Then, that face's peripheral edges are used to skin or loft to the peripheral edges of the second face. The result is one body.

If the body is a sheet body, the face provided as an argument is kept and serves as a cap of the resulting body. Here again, the face's peripheral edges are used to skin or loft to the peripheral edges of the second face.

Note *This does not check to see if the resulting body is self–intersecting.*

In this API, lofting uses the surface information from the adjacent surfaces to determine the take-off vectors. Because this behavior is embedded inside the API it can not be changed. However, lofting between faces is simply a convenience function created from the standard lofting functionality. You could create lofting between faces functions that do permit changes to the take–off vector. Figure 1–4 illustrates lofting between two solid bodies.

Scheme Example

```
( option:set ”sil” #t)
( option:set ”u_par” 4)
( option:set ”v_par” 5)
(define block1 (solid:block (position 0 0 0)
    (position 10 10 10)))
(define face_list1 (entity:faces block1))
(define block2 (solid:block (position 30 10 0)
    (position 40 20 10)))
(define face_list2 (entity:faces block2))
(define loft1 (solid:loft–faces
    (list–ref face_list1 5)
    (list–ref face_list2 3)
    #f #t #t #f #f))
( option:set ”sil” #f)
( option:set ”u_par” 0 )
( option:set ”v_par” 0 )
```

Example 1-3. Lofting Faces between Solid Bodies

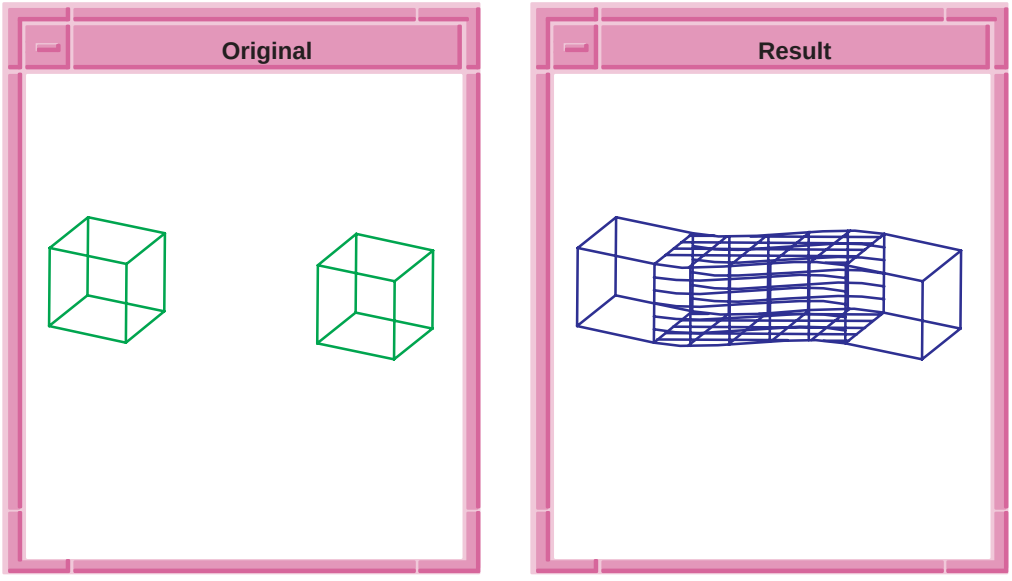

Figure 1-8. solid:loft–faces between Solid Bodies

Loft to a Point

Topic: *Skinning and Lofting

Scheme Extensions: section, law, sheet:loft_wires

C++ APIs: api_loft_coedges

This example shows lofting from a square wire to a point. The point has a radial law take–off vector field defined, which caps the top smoothly. Each side of the square has a constant law take–off vector field defined (Figure 1-9).

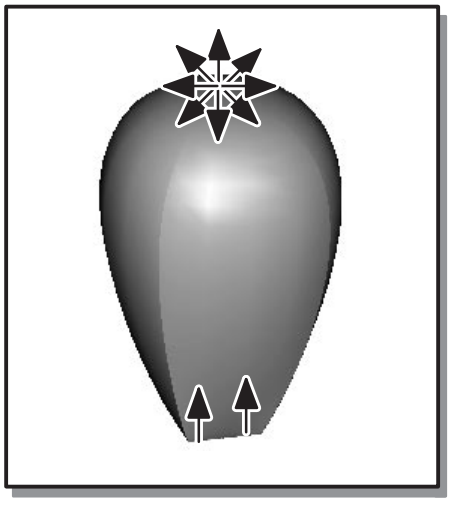

Figure 1-9. Lofting to a Point

Note *A law for the vector field is defined in general to go from R^1 to R^3. Before being applied to a section, this law needs to have its domain bounded.*

Scheme Example

```
(define v0 (wire–body:points (position 0 0 10)))
(define v1 (wire–body:points
    (list (position 1 1 0) (position –1 1 0) (position –1 –1 0)
        (position 1 –1 0) (position 1 1 0) )))
(define coedge_list1 (entity:coedges v0))
(define coedge_list2 (entity:coedges v1))
(define lawa (law "vec(cos(t),sin(t),0)"))
(define dom0 (law "domain(law1,law2,law3)" lawa (law:eval "0.0")
    (law:eval "2.0*Pi")))
(define lawb (law "vec(0,0,1)"))
(define dom1 (law "domain(law1,0,1)" lawb))
(define sec1 (section coedge_list1 (list dom0) #f 20))
(define sec2
    (section coedge_list2 (list dom1 dom1 dom1 dom1) #t 1))
(define loft1 (sheet:loft–wires (list sec1 sec2) #f #t #t #f))
(sheet:2d loft1)
```

Example 1-4. Lofting to a Point

Loft to a Circle

Topic: *Skinning and Lofting

Scheme Extensions: section, law, sheet:loft–wires

C++ APIs: api_loft_coedges

The following example shows a surface lofted from a square to a circle. The circle has a radial vector law field defined, while each side of the square has a constant law vector field defined (Figure 1-10).

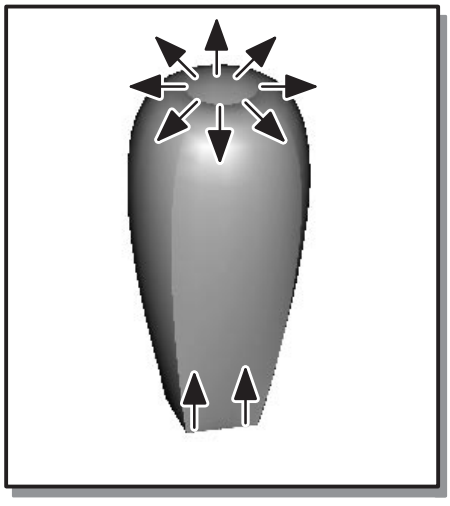

Figure 1-10. Lofting to a circle

Scheme Example

```
(define v1 (wire–body:points
    (list (position 1 1 0) (position –1 1 0) (position –1 –1 0)
        (position 1 –1 0) (position 1 1 0) )))
(define e0 (edge:circular (position 0 0 10) 1 0 360))
(define v0 (wire–body e0))
(define coedge_list1 (entity:coedges v0))
(define coedge_list2 (entity:coedges v1))
(define lawa (law "vec(cos(t),sin(t),0)"))
(define dom0 (law "domain(law1,law2,law3)" lawa (law:eval "0.0")
(law:eval "2.0*Pi")))
(define lawb (law "vec(0,0,1)"))
(define dom1 (law "domain(law1,0,1)" lawb))
(define sec1 (section coedge_list1 (list dom0) #f 10))
(define sec2
    (section coedge_list2 (list dom1 dom1 dom1 dom1) #t 1))
(define loft1 (sheet:loft–wires (list sec1 sec2) #t #t #t #f))
(sheet:2d loft1)
(render)
```

Example 1-5. Lofting to a Circle

Loft with Guides

Topic: *Skinning and Lofting

Scheme Extensions: section, law, sheet:loft–wire–guides

C++ APIs: api_loft_coedges

Loft with Guides allows the designer to perform typical lofting operations with the additional constraint of one or more guide curves. In addition to Loft with Guides, guide curves can also be added to the vector and draft skinning operations.

The rules pertaining to the guides follow those of skinning and further information can be reviewed in the Skinning with Guides section of Advanced Surfacing.

An additional skinning and lofting option – guide curve constraint – has been added to the skin options class to handle details of the Loft with Guide Curves operation. Guide curves can be placed on the profiles in any order or manner – although they must not intersect or cross. The tangent directions of the ends of the guide curves do not have to follow the loft surface tangent constraint however if they differ in direction the surface is said to be ”over–constrained”. (Figure 1–11). That is, at the exact same point, we have a guide curve directing the surface in one direction and a tangent constraint guiding the surface in a different direction.

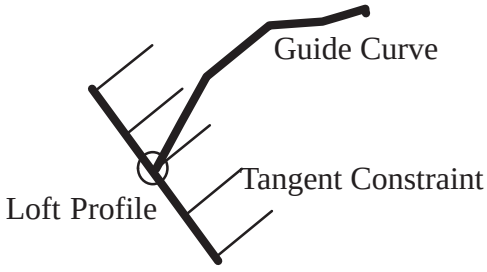

Figure 1-11. Over–constrained Surface Conditions

The ”guide curve constraint” option is used to address this problem by telling the surface to prefer one constraint to the other. The two possible values of the option are ”constrain to guide” and ”constrain to tangent”. As their names suggest the ”constrain to guide” value will force the surface to follow the guide curve and ignore the tangent constraint in the region of the guide and the ”constrain to tangent” will ignore the guide and constrain to the tangent cone in the region of the tangent cone. In order for the designer to get a surface matching exactly to the input given, he must give guide curves and tangent constraints that agree in direction.

Scheme Example

```
(part:clear)
(view:dl)
(view:gl)
(wcs (position 0 0 0) (gvector 1 0 0) (gvector 0 1 0))
(define w1 (wire–body (list (edge:linear (position 100 0 0)
    (position –100 0 0))
(edge:linear (position –100 0 0) (position –100 60 0))
(edge:circular–3pt (position –100 60 0) (position 0 110 0)
    (position 100 60 0))
(edge:linear (position 100 60 0) (position 100 0 0)))))
(define w2 (wire–body (list (edge:linear (position 120 0 140)
    (position –120 0 140))(edge:linear (position –120 0 140)
    (position –120 60 140))(edge:circular–3pt (position –120 60
    140) (position 0 150 140) (position 120 60 140))(edge:linear
    (position 120 60 140) (position 120 0 140)))))
(define s1 (sheet:skin–wires (list w1 w2)))
(entity:delete (list w1 w2))
(define w3 (wire–body (list (edge:linear (position 120 0 300)
    (position –120 0 300))(edge:linear (position –120 0 300)
    (position –120 60 300))(edge:circular–3pt (position –120 60
    300) (position 0 150 300) (position 120 60 300))(edge:linear
    (position 120 60 300) (position 120 0 300)))))

; define the guides
(define g1 (edge:spline (list (position 0 150 140) (position 0
    180 275) (position 0 150 300))))
(define coedge_list1 (list (list–ref (entity:coedges s1) 4)
    (list–ref (entity:coedges s1) 6) (list–ref (entity:coedges
    s1) 0) (list–ref (entity:coedges s1) 2)))
(define sec1 (section coedge_list1 #f 1))
(define coedge_list2 (entity:coedges w3))
(define sec2 (section coedge_list2 #t 1))
(define l1 (sheet:loft–wires–guides (list sec1 sec2) (list g1)
    (skin:options ”perpendicular” #t ”guidePreference”
    ”constrain_to_tangent” ”solid” #t)))
(entity:delete (list w3 g1))

(bool:unite l1 s1)
```

Example 1-6. Loft with Guides

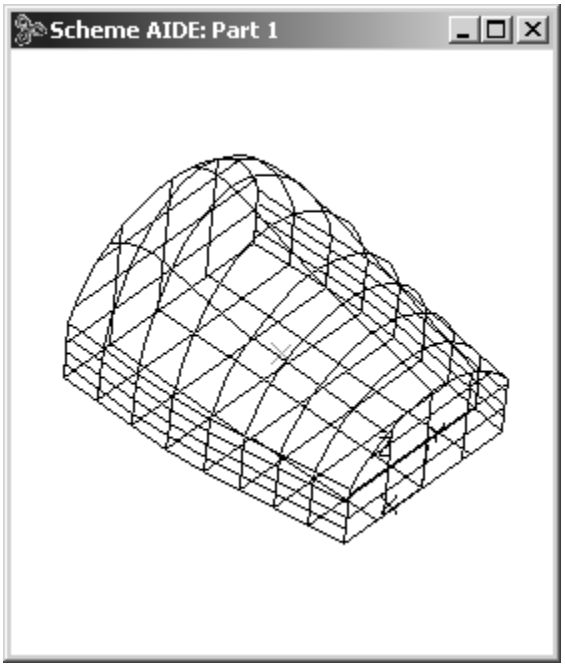

Figure 1-12. Lofting with Guide Curves

Skinning

Topic: *Skinning and Lofting

Skinning fits a surface through a series of curves. The skinned surface is a two–parameter function (u,v); the curves provide the u parameter of the surface and the skin surface algorithm defines the v parameter (Figure 1-13). It is a precise surface and does not have C1/G1 discontinuities. ACIS can make a skin surface between any numbers of curves. The surface allows a “take–off” vector field to be placed on each of the curves, forcing a tangent constraint on that surface at that curve cross section. Although tangent constraints are more often associated with lofting, skinning allows constraints to be placed on profiles in several indirect ways.

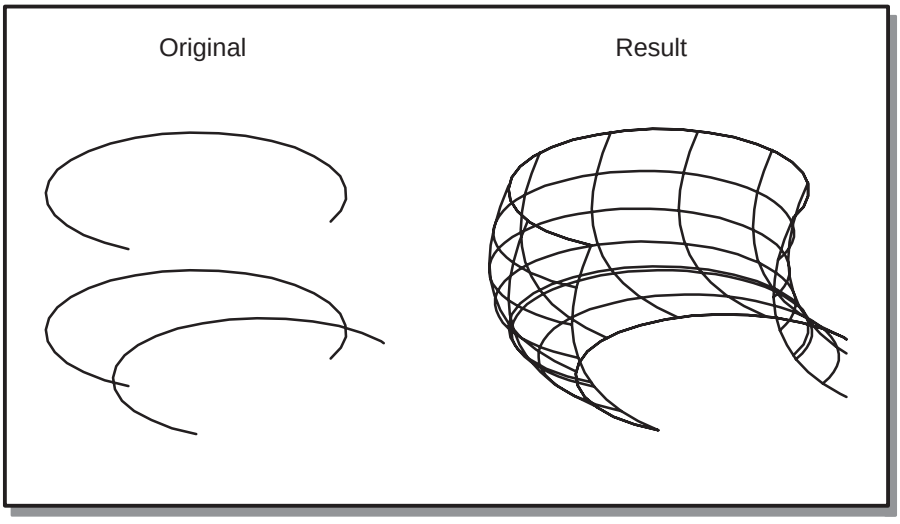

Figure 1-13. ACIS Skin Surface

The curves used as input to skinning are given in terms of wire bodies and hence may contain more than one edge (curve) per body. In this case, and for lofting, edges are aligned (with the possibility of splitting edges) and a series of surfaces are made. The resulting sheet body consists of one or more faces linked together to form a shell or solid.

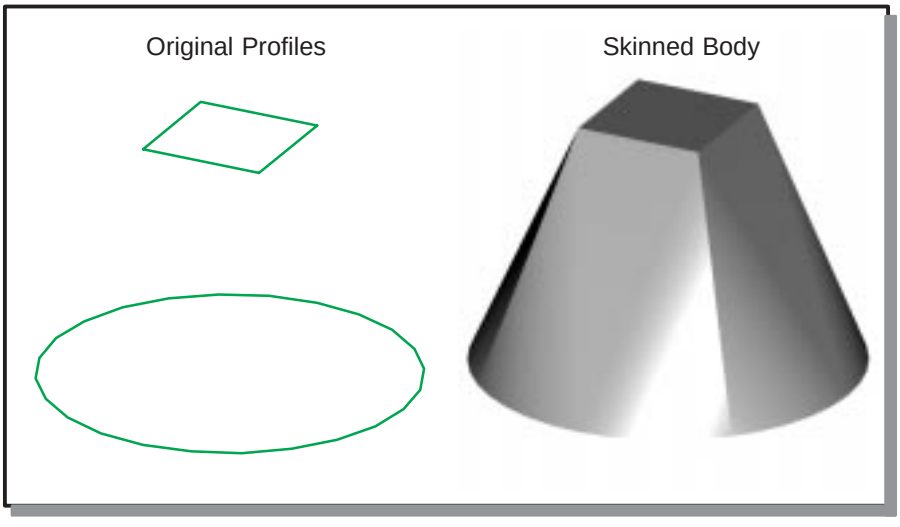

Figure 1-14. Skinning profiles with differing numbers of edges

Skinning and lofting are very similar techniques. However, in ACIS, two distinctions exist: skinning takes wire bodies as input and allows control over tangent constraints in indirect ways. Lofting takes coedges as input and allows direct control over tangent constraints. Skinning and lofting are related historically in the industry and in their ACIS implementations. Skinning is often used as a newer term for lofting, although ACIS maintains a distinction between the two operations.

The Advanced Surfacing Component provides eight functional variations of skinning, differing by methods of surface control:

- Basic skinning
- Skinning with a path
- Skinning with a draft angle
- Skinning to the planar normal
- Skinning with guide curves
- Skinning with virtual guide curves
- Skinning with ruled surfaces
- Multi–stem skinning

Each of these is explained in detail in the relevant section.

In general, skinning provides options to support the following:

- Surface isoparametric or arc length parameterization
- Alignment of the direction of the cross section curves so that they are in the same direction as the first curve
- Twist and non–twist minimization
- Simplification of the created surface to a conical surface, if applicable
- Creation of a closed body
- Creation of a solid body

Each of these is explained in detail in the relevant section.

Skinning Parameters

Topic: *Skinning and Lofting

A skinning operation requires three basic categories of parameters (arguments). The first category defines the wire body profiles to be skinned. The second category defines the type or method of the skinning operation to be performed (skinning with guides, skinning with a path, etc.) The third category contains optional parameters to control what type of object is formed as a result of the skinning operation (closed, solid, etc.).

```
api_skin_wires (profiles, guides or paths, skinning options)
```

Profiles

The profiles used for skinning are given as an array of wire bodies. There must be at least two wires. If a closed solid body is required, at least three distinct wire profiles must be provided. If the first and last profiles are identical, a closed solid skinned body is automatically constructed. Wires can be open or closed but the entire wire list must be all open or all closed. They must be simple loops if closed and in all cases, not self–intersecting. Additionally, the first and last wires in a skin surface may be points (a point in ACIS is represented by a vertex wire). For example, skinning between a point and a circular arc will give a spline cone with the point being the apex.

Guides, Paths, Draft Conditions, etc.

C++ Data Structures: skinning_normals

The second category defines the type of skinning operation to be performed. In the case of simple skinning there are no arguments for this category. However for skinning with guides, a list of one or more edges is required. For skinning with a path, an additional wire body is required. For skinning with a draft angle, two angles and magnitudes are required. For skinning with planar normal constraints, the enumerated type skinning_normals is required. api_skin_wires is overloaded to handle all the different permutations of input.

Options

C++ Data Structures: skinning_options

The main skinning API also takes options to control surface parameterization, wire twist, wire direction, simplification, and creation of a solid or closed body. Although different types of skinning accept different options, all options maintain the same principle and purpose across the type variations. The options can be passed in individually as a series of flags or consolidated in the class skinning_options. See *Skinning and Lofting Options* for more information.

Functional Variations of Skinning

Topic: *Skinning and Lofting

The following examples demonstrate the functional variations of skinning. Starting with the same original profiles, applying the different methods of surface control produces very different results.

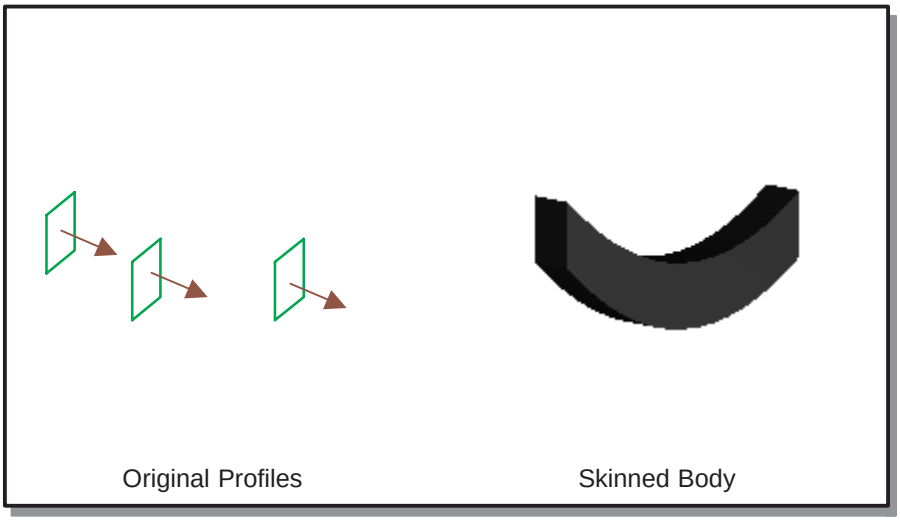

Figure 1-15. Basic Skinning

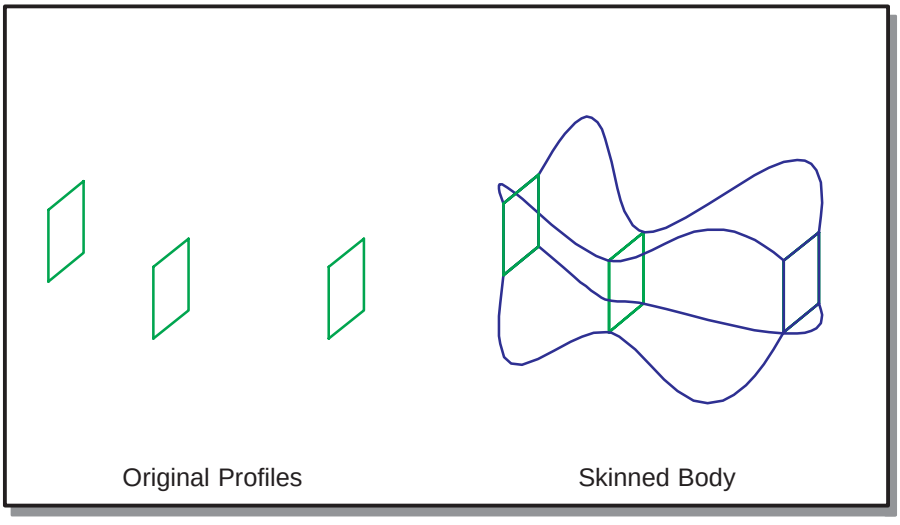

Figure 1-16. Skinning with Draft Angles

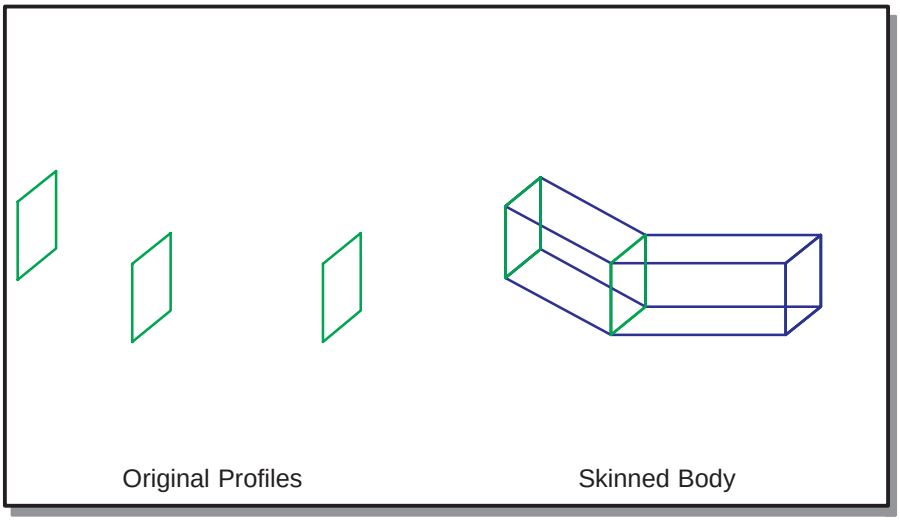

Figure 1-17. Ruled Skinning

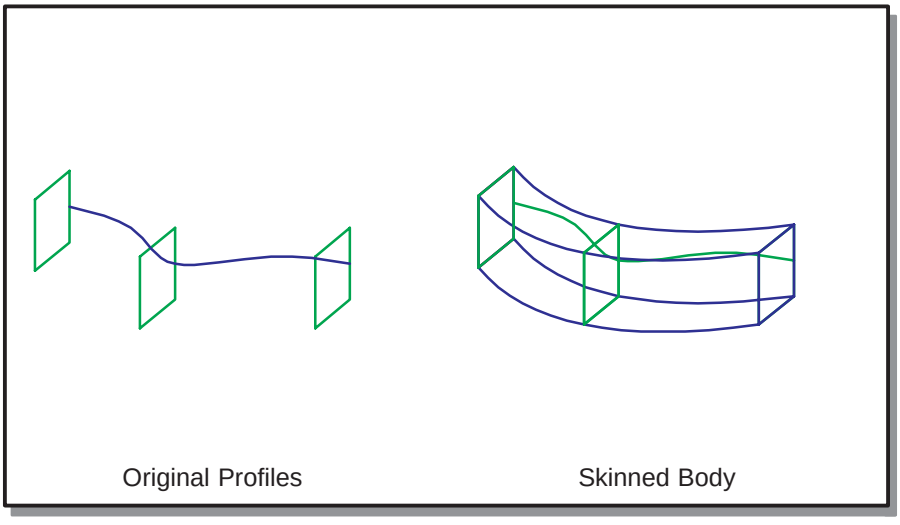

Figure 1-18. Skinning with Guide Curves

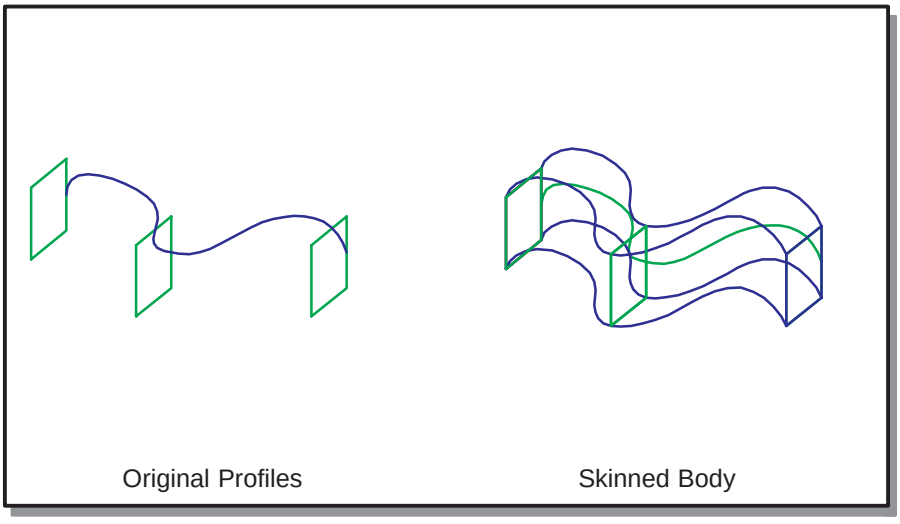

Figure 1-19. Skinning with Virtual Guide Curves

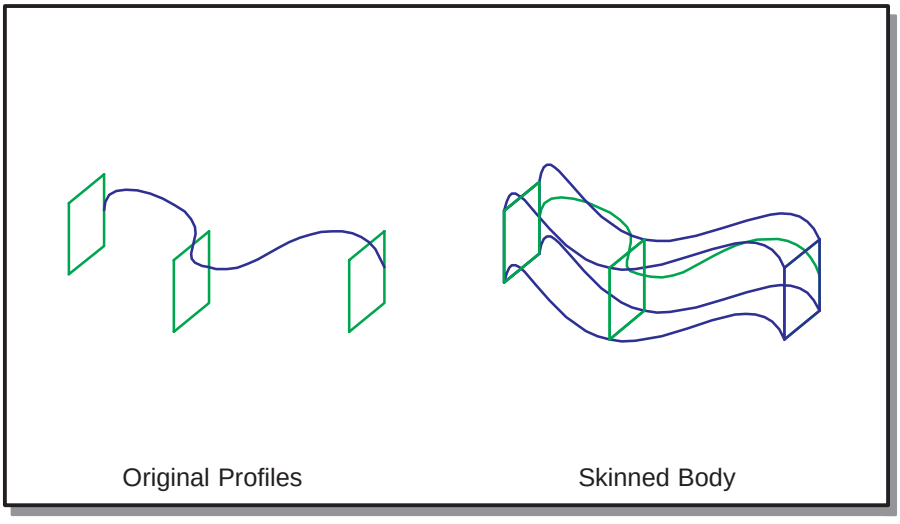

Figure 1-20. Skinning with Paths

Basic Skinning

Topic: *Skinning and Lofting

Scheme Extensions: sheet:skin–wires

C++ APIs: api_skin_wires

Skinning Options: arc length, no twist, minimize twist, simplify, closed, solid, periodic

Basic skinning fits a surface through a series of curves (Figure 1-21). In this case the skinning algorithm computes the take–off vector field going into and out of the wire bodies. If only two wires are given, the resulting surface is ruled.

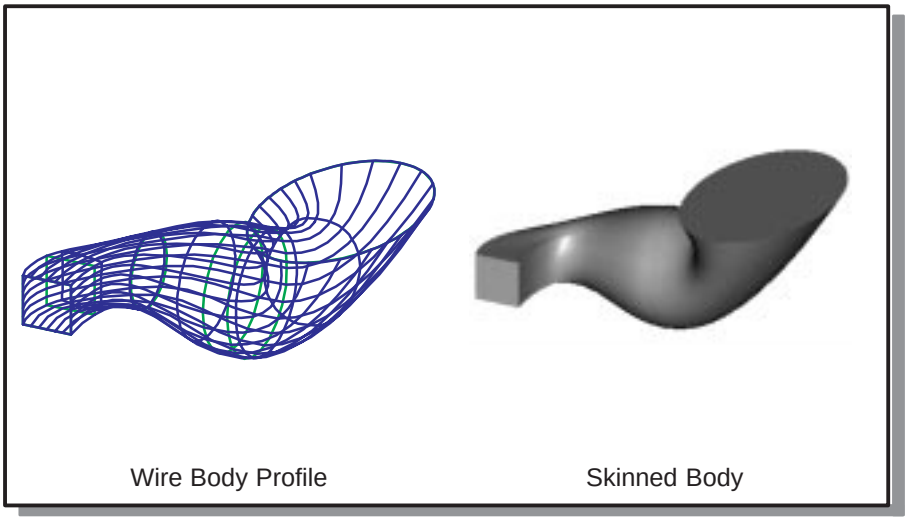

Figure 1-21. Basic Skinning

Skinning with Guide Curves

Topic: *Skinning and Lofting

Scheme Extensions: sheet:skin–wires–guides

C++ APIs: api_skin_wires

Skinning Options: arc length, no twist, minimize twist, closed, solid, periodic, simplify, virtual guides

Skinning with guide curves is a method of locally controlling the shape of the v parameter direction of the skin surface in between the input wires. In a typical skinning operation, the user creates two or more wire bodies and a surface is fitted between them. When skinning with a guide curve, one or more 3–space curves can be specified between the skinning profiles, and the surface will follow the guide curves. The surface will be C1 continuous at every point, even across the guide itself. (Figure 1-22).

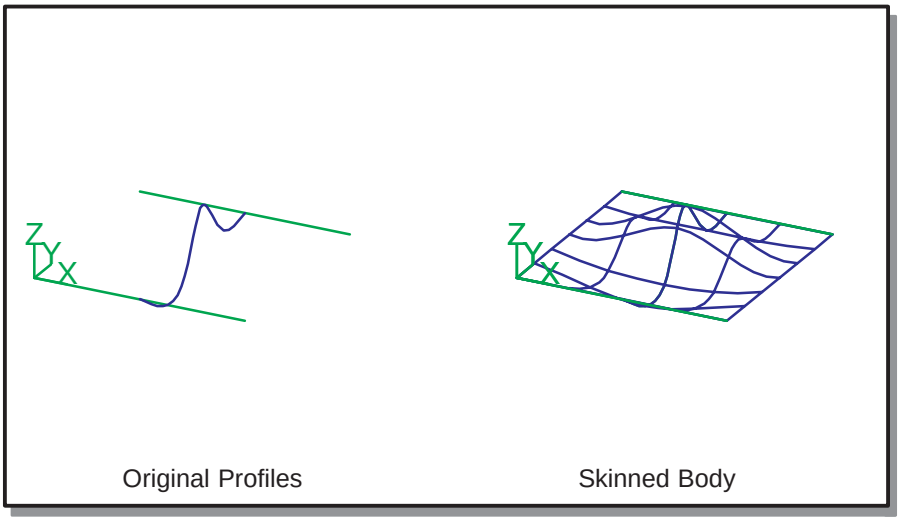

Figure 1-22. Skinning with Guide Curve

One or more guide curves can be placed on the skinning profiles (Figure 1-23).

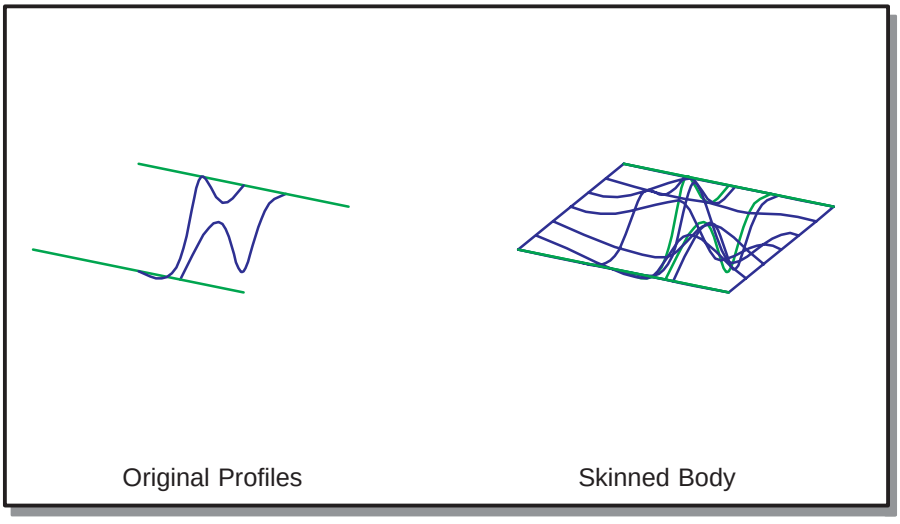

Figure 1-23. Skinning with two Guide Curves

The guide curve is used for local surface control only. It affects the geometry of the surface created between the series of edges to which the curve is attached. No other surfaces are effected (Figure 1-24).

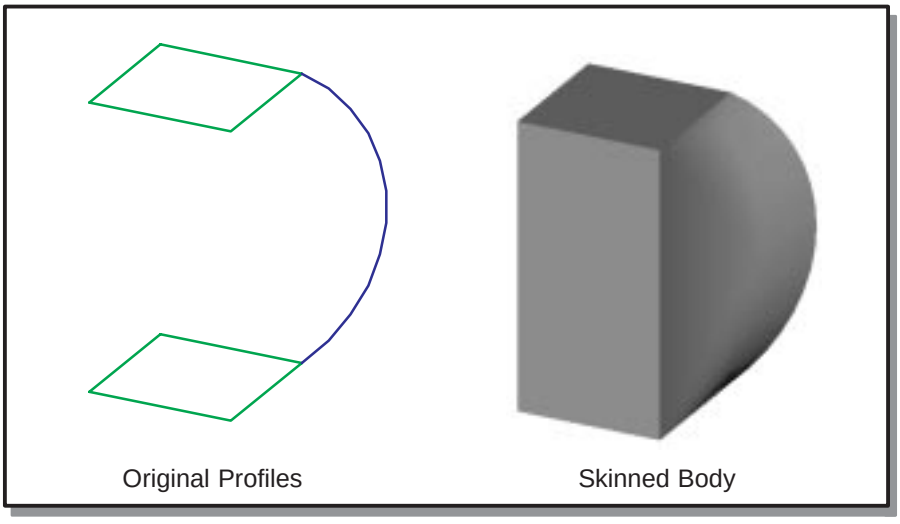

Figure 1-24. Guide curve affects only the right–most surface

If the guide curve passes through the vertices, the surfaces on the two adjoining faces follow the curve profile. In this case, the boundary of the surface extends along the guide and we do not maintain C1 continuity (Figure 1-25).

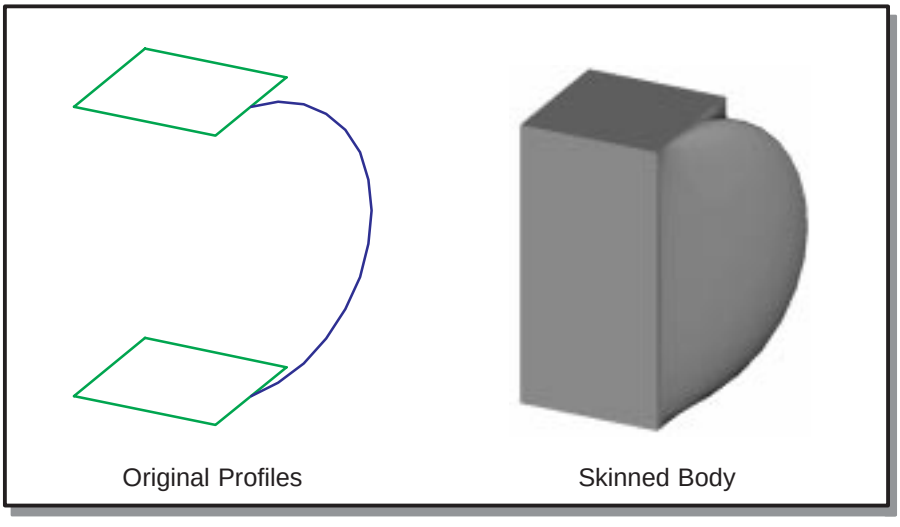

Figure 1-25. Guide curve affects the right and front surfaces

Considerations on input of guide curves:

- The guide curves must contact (within SPAresabs) each profile in the skinning list
- The guide curves must start and end on the first and last guide curves respectively
- Any number of guide curves can be added
- The guide curves need not be consistent in the v direction
- The guide curves must be non–looping and well behaved

In addition to surface control, guide curves also implicitly control the breakup of the coedges. If the guide curves intersect vertices on the wire it is guaranteed that those vertices will align. If a guide curve intersects coedges at positions other than vertices, then the adjacent vertices will be aligned (A and B in Figure 1-26).

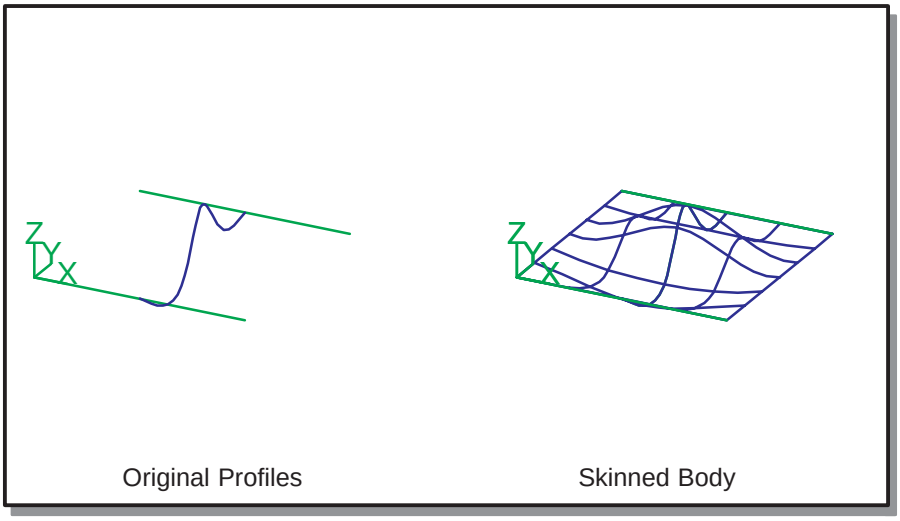

Figure 1-26. Vertices are automatically aligned

Skinning with Virtual Guide Curves

Topic: *Skinning and Lofting

Scheme Extensions: sheet:skin–wires–guides

C++ APIs: api_skin_wires

Skinning Options: arc length, no twist, minimize twist, closed, solid, periodic, simplify, virtual guides

Virtual guide curves provide the ability to make one guide curve globally affect the skinned surfaces (Figure 1-27). If one guide curve is added to a list of profiles, ACIS takes that guide curve and propagates it across the profiles to each edge vertex. This effectively changes guide curves from local surface control to global surface control. This flag is false by default, but can be set to true in the call to the function api_skin_wires.

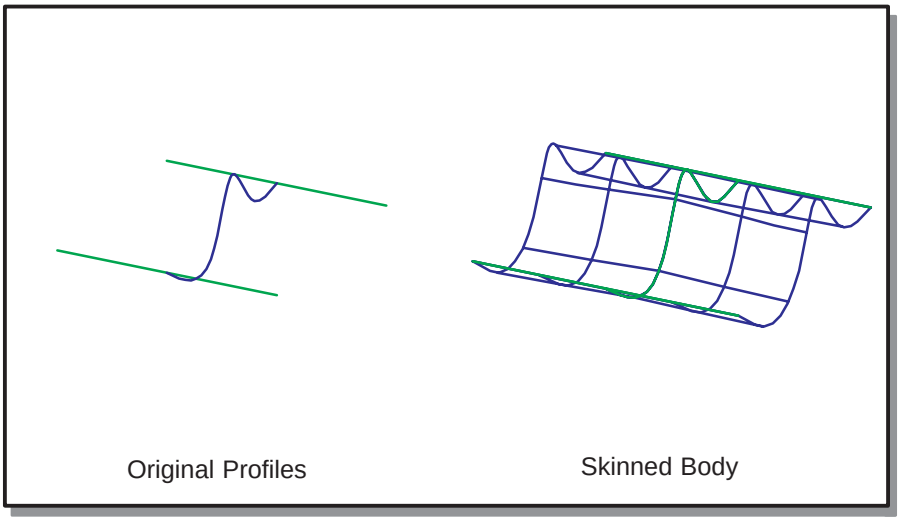

Figure 1-27. Virtual Guide Curves

One or more guide curves can be placed on the wire profiles. In this case, the guide curve closest to the edge is propagated to each edge vertex (Figure 1-28).

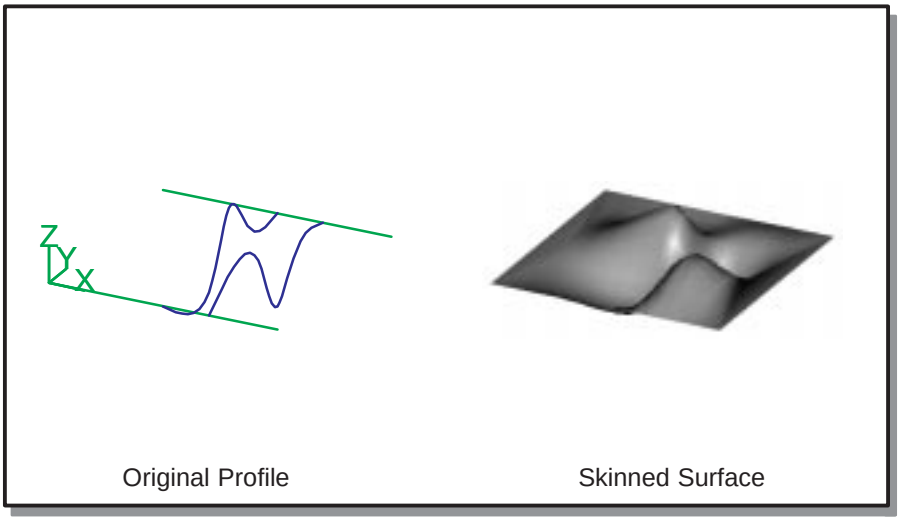

Figure 1-28. Virtual guide curves with two user–defined curves

If a wire vertex has two neighboring user–defined guide curves, the two curves are blended together at the vertex (Figure 1-29).

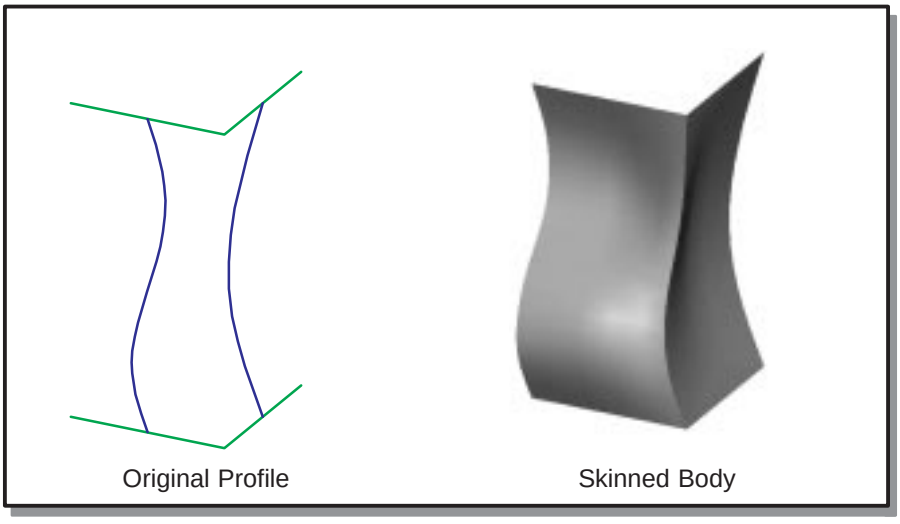

Figure 1-29. Virtual guide curves with two user–defined curves

Skinning with Draft Angles

Topic: *Skinning and Lofting

Scheme Extensions: sheet:skin–wires–draft

C++ APIs: api_skin_wires

Skinning Options: arc length, no twist, minimize twist, solid

Skinning with draft angles provides the ability to control the take–off vectors of the two outer skinning profiles. The “draft” angle is defined as an angle off the plane of the wire at every point along the skinning profile. In addition to the user supplying the angle itself, one may also supply a magnitude for the take–off vector.

The draft angle and magnitude is constant for the entire profile. However, one may apply different draft angles to the two outer profiles. In addition skinning with draft angles supports open and closed profiles and skinning to a point. When skinning to a point, the algorithm constructs its own normal vector. The outer profiles must be planar when not degenerate.

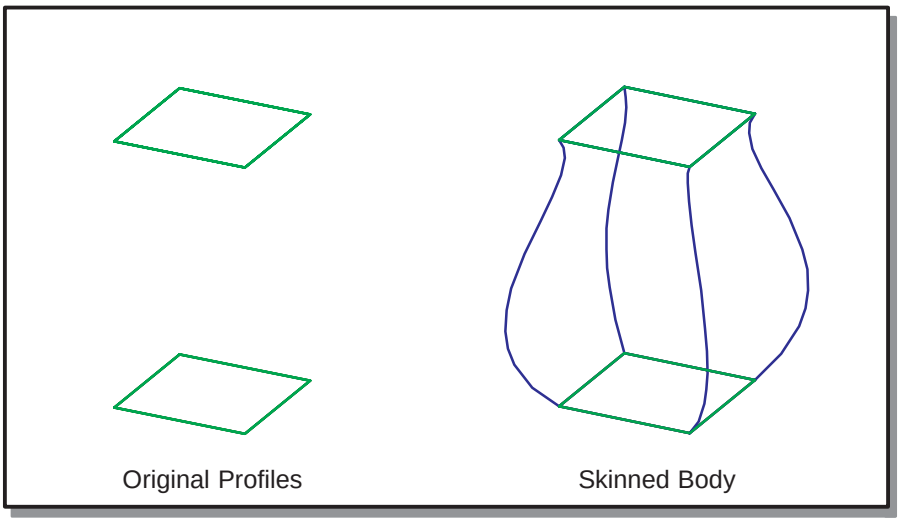

Figure 1-30. Skinning with Draft Angles

Skinning to the Planar Normal

Topic: *Skinning and Lofting

Scheme Extensions: sheet:skin–wires–normal

C++ APIs: api_skin_wires

C++ Data Structures: skinning_normals

Skinning Options: arc length, no twist, minimize twist, closed, solid, periodic

Skinning to the planar normal provides the ability to constrain the take–off vectors on each profile to the profile's normal. All profiles must be planar and non–degenerate.

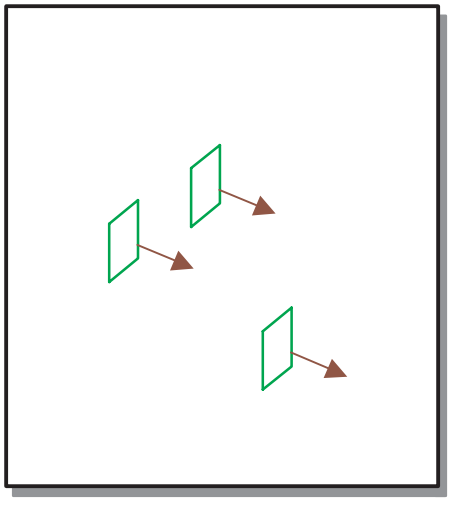

Figure 1-31. Skinning to the Planar Normal

As input, api_skin_wires takes the enumerated type skinning_normals containing the following constants:

LAST_NORMAL . . . . . constrain only the last profile

FIRST_NORMAL . . . . . constrain only the first profile

ENDS_NORMAL . . . . . constrain the first and last profile

ALL_NORMAL . . . . . constrain all the profiles

Skinning to the planar normal does not allow control of the magnitude of the take–off vectors; rather, it determines the magnitudes that yield surfaces with the maximum minimum radius of curvature.

Ruled Skinning

Topic: *Skinning and Lofting

Scheme Extensions: sheet:skin–wires–ruled

C++ APIs: api_skin_wires

Skinning Options: arc length, no twist, minimize twist, simplify, closed, solid

Ruled skinning provides the ability to skin a series of three or more profiles by placing ruled surfaces in between each section of profiles. If only two profiles are provided, this command defaults to basic skinning; and generates a ruled surface in between the two profiles.

Ruled skinning works with both closed and open profiles, degenerate end points, and the solid and closed options. However, it is possible in many “closed” situations to have surfaces that intersect each other. No check is made for that case.

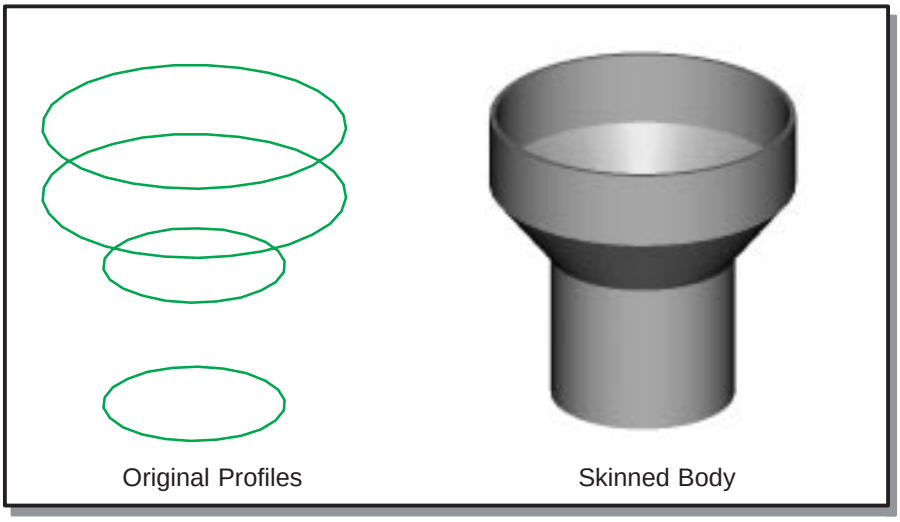

Figure 1-32. Ruled Skinning

Skinning with Paths

Topic: *Skinning and Lofting

Scheme Extensions: sheet:skin–wires

C++ APIs: api_skin_wires

Skinning Options: arc length, no twist, minimize twist, simplify, closed, solid

Skinning with a path provides the ability to constrain the take–off vectors on each profile based on a “path” curve. The resulting surface does not follow the path exactly; rather a constant vector field is placed on each profile. The vector is defined as the tangent vector of the path curve at the point in which the curve intersects the profile’s plane. (Figure 1-33).

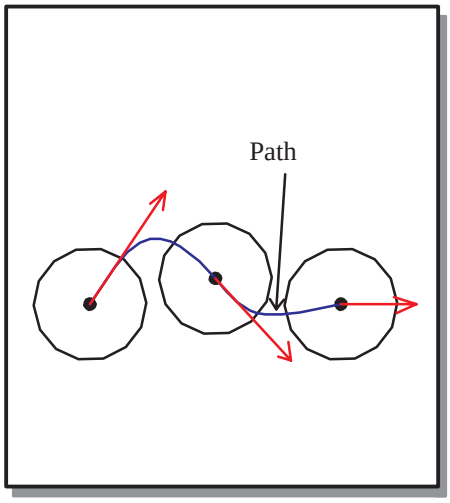

Figure 1-33. Take–Off Vector used in Skin with Path

Only one path can be used and it must be a well behaved curve.

Branched Skinning

Topic: *Skinning and Lofting

Scheme Extensions: sheet:skin–wires–branch

C++ APIs: api_skin–wires

C++ Data Structures: skinning_normals

Skinning Options: arc_length, align direction, minimize twist, solid

Branched skinning provides the ability to skin multiple profile paths. The user first specifies a profile path for the trunk of the skin, consisting of one or more wires.They can then specify two or more branches (each of one or more wires) which the skin body will follow.

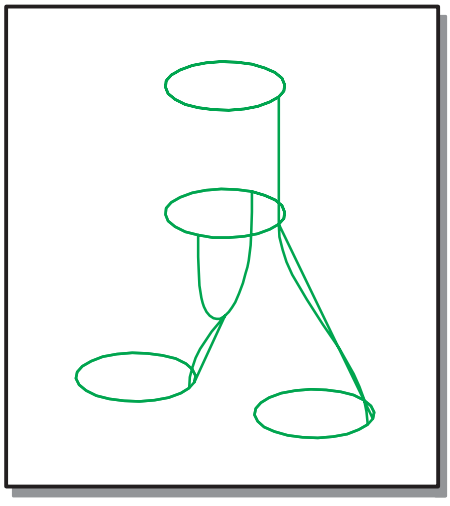

Figure 1-34. Branched Skinning

Unlike regular skinning, the order of the wires in the profile list(s) is important. The last profile in the trunk will be the end to which the branches attach. The first profile of each of the branches will be the end which they attach to the trunk.

In addition to the standard options, the user may also specify “normal” conditions similar to the command “skinning to the planar normal”. However, in all cases, the normal condition will always be imposed on the last wire of the trunk. This guarantees a C1 continuity between the trunk and the branches.

The wire profiles may be open or closed and must be planar.

Scheme Example

```
(define wire1 (wire-body (list (edge:circular (position 0 0 70) 30 0 360))))
(define wire2 (wire-body (list (edge:circular (position 0 0 0) 30 0 360))))
(define wire3 (wire-body (list (edge:circular (position -50 0 -100)
    30 0 360))))
(define wire4 (wire-body (list (edge:circular (position 50 0 -100)
    30 0 360))))
(define body (sheet:skin-wires-tree (list wire1 wire2) (list wire3)
    (list wire4)))
```

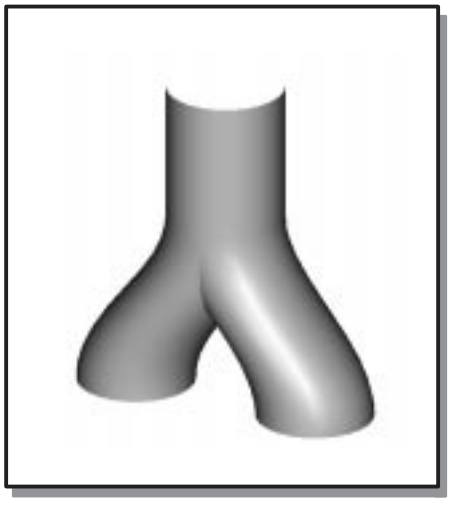

Figure 1-35. Branched Skinning

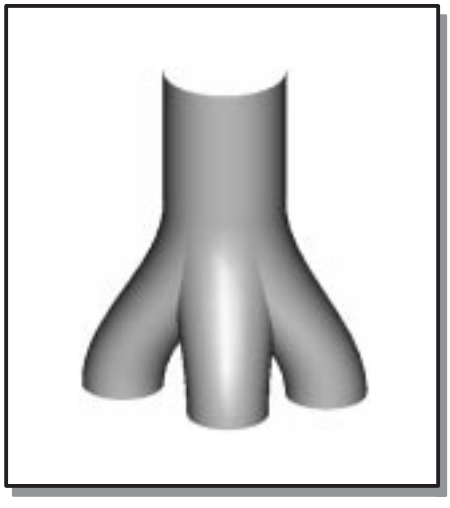

Figure 1-36. Branched Skinning

Skinning and Lofting Options

Topic: *Skinning and Lofting

A number of options are available with lofting to control the edge parameterization, twist minimization, coedge direction alignment, take–off vector direction, conic simplification and creation of a solid body. The skinning and lofting optional arguments have the following default values:

Optional Argument	Default
arc length	false
twist	true
align direction	true
perpendicular	false
simplify	true
closed	false
solid	true
periodic	false
virtual guides	false
merge wire coedges	true
estimate loft tanfacs	true
match vertices	true
guide pref	FOLLOW_GUIDE_CONSTRAINT

Arc Length Option

Topic: Skinning and Lofting

The *arc length* option is used to choose arc length or isoparametric parameterization of the skinning/lofting surfaces. In isoparametric parameterization, the surface parameter in the v direction follows the cross section curves. In arc length parameterization, the surface parameter follows lines of constant length. The default is isoparametric parameterization.

Scheme Example

```
; Turn on u_param lines to see different parameterization.
(option:set 'u_param 7)
(zoom-all)
(define wire_1 (wire-body (list (edge:linear (position 0 0 0)
    (position 50 0 0))(edge:linear (position 50 0 0)
    (position 50 50 0)) (edge:linear (position 50 50 0)
    (position 0 50 0))(edge:linear (position 0 50 0)
    (position 0 0 0)))))
(define wire_2 (wire-body (list (edge:linear (position 0 0 60)
    (position 50 0 40))(edge:linear (position 50 0 40)
    (position 50 50 40))(edge:linear (position 50 50 40)
    (position 0 50 60))(edge:linear (position 0 50 60)
    (position 0 0 60)))))
(define wire_3 (wire-body (list (edge:linear (position 0 0 100)
    (position 50 0 100))(edge:linear (position 50 0 100)
    (position 50 50 100))(edge:linear (position 50 50 100)
    (position 0 50 100))(edge:linear (position 0 50 100)
    (position 0 0 100)))))
; --- Arc length
(define arc_body (sheet:skin-wires
    (list wire_1 wire_2 wire_3) #t #f #f #f))
(define move1 (entity:move arc_body 100 0 0))
; --- Iso
(define iso_body (sheet:skin-wires
    (list wire_1 wire_2 wire_3) #f #f #f #f))
(define move2 (entity:move iso_body -100 0 0))
(iso)
(zoom-all)
```

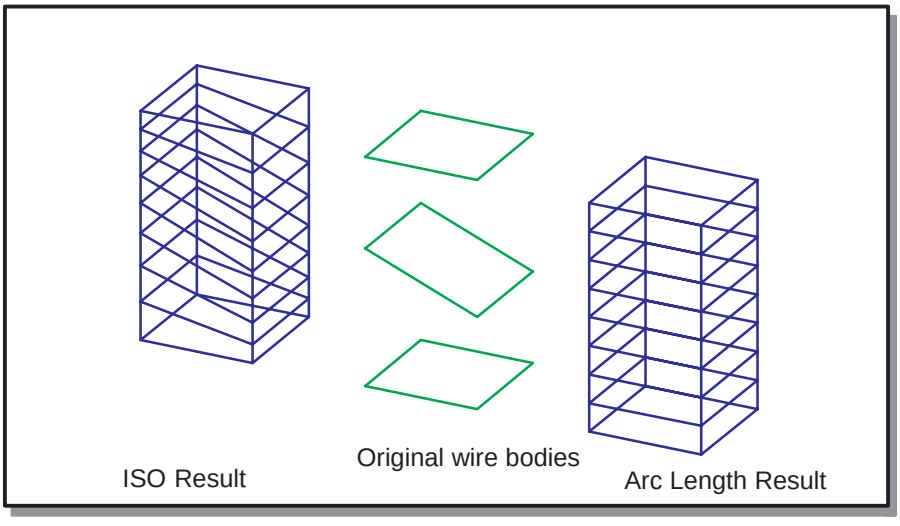

Figure 1-37.

Example 1-7. Using the Arc Length Option

Twist Option

Topic: Skinning and Lofting

The *twist* option may be used to minimize the twist of the surface produced. Twist minimization aligns closed curves such that the start of the second curve is aligned to the start of the first curve. Even if a body's shape is unaffected by twisting, a surface with a twist could produce unexpected results when faceting and rendering. Figure 1-38 and Figure 1-39 show illustrations of twist minimization.

The default is that twist minimization is off, because otherwise the geometry definition of the underlying curves is changed to align the starting positions. Twist minimization is also an involved calculation that some users may not want to have carried out.

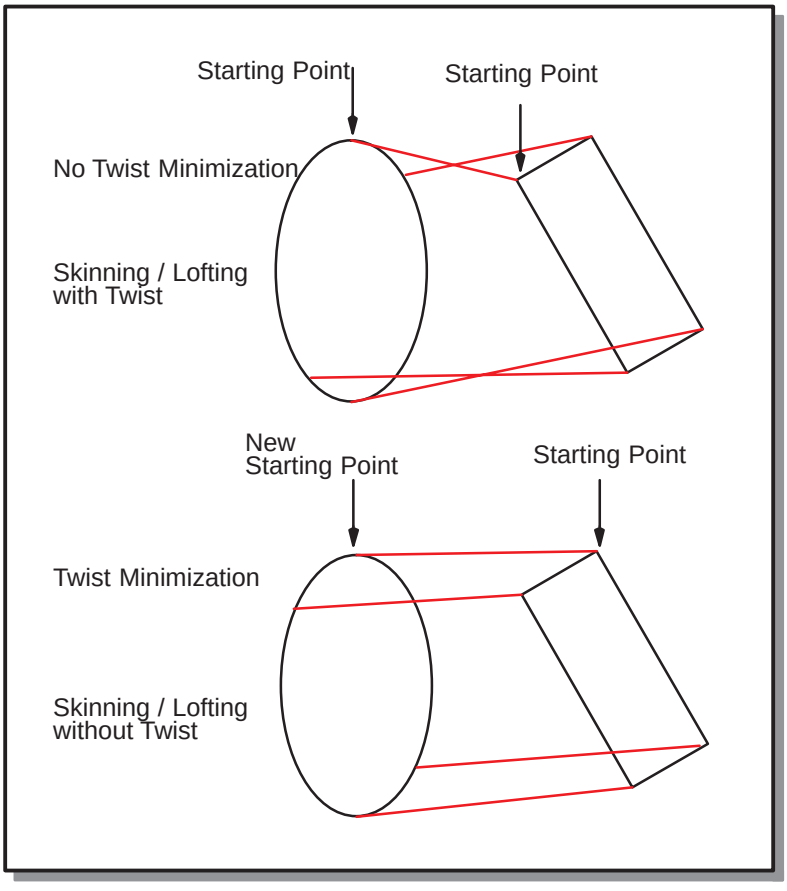

Figure 1-38. Conceptual Illustration of Twist Minimization

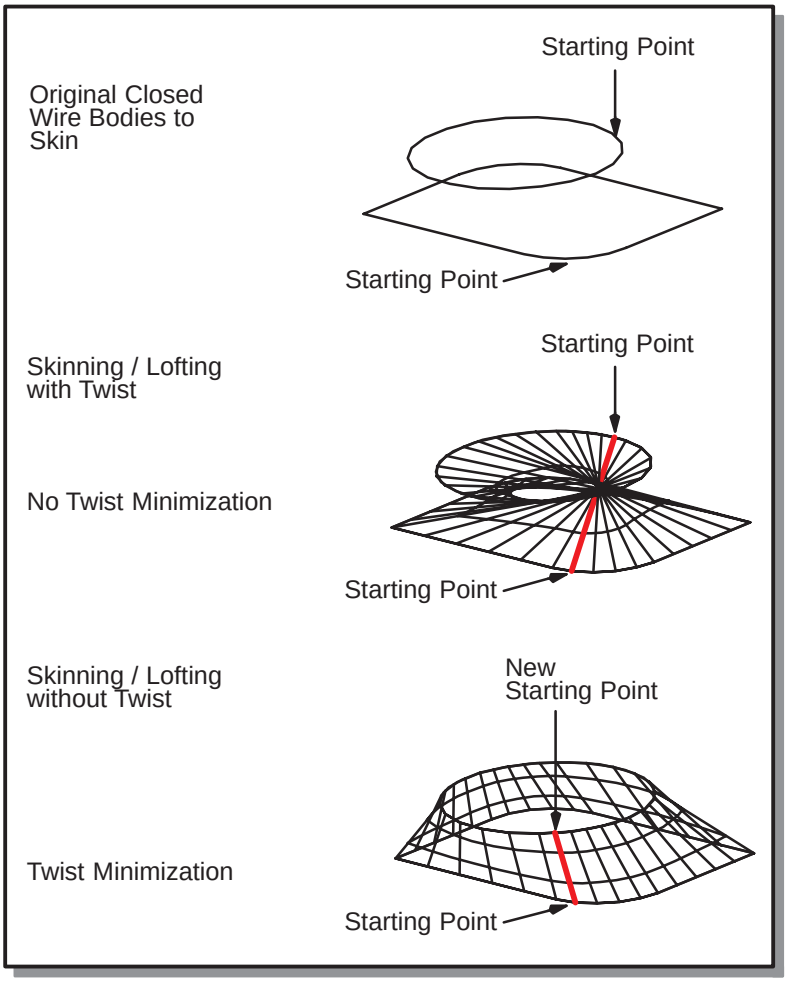

Figure 1-39. Twist Minimization, Faceted

Figure 1-39 shows an example of skinning between circular and square profiles without twist minimization (upper left) and with twist minimization (lower right).

Scheme Example

```
(define loop1 (wire-body (list
    (edge:circular (position 0 0 0) 10) )))
```

```
(define loop2 (wire-body (list
    (edge:linear (position -10 -10 -5) (position -10 10 -5))
    (edge:linear (position -10 10 -5) (position 10 10 -5))
    (edge:linear (position 10 10 -5) (position 10 -10 -5))
    (edge:linear (position 10 -10 -5) (position -10 -10 -5)) )))

(define skin (sheet:skin-wires (list loop1 loop2) #t #f #t))
(define skin2 (sheet:2d skin))
; OUTPUT No Twist

(roll -2)

(define skin (sheet:skin-wires (list loop1 loop2) #t #t #t))
(define skin2 (sheet:2d skin))
; OUTPUT Twist
```

Example 1-8. Twist Minimization, Rendered

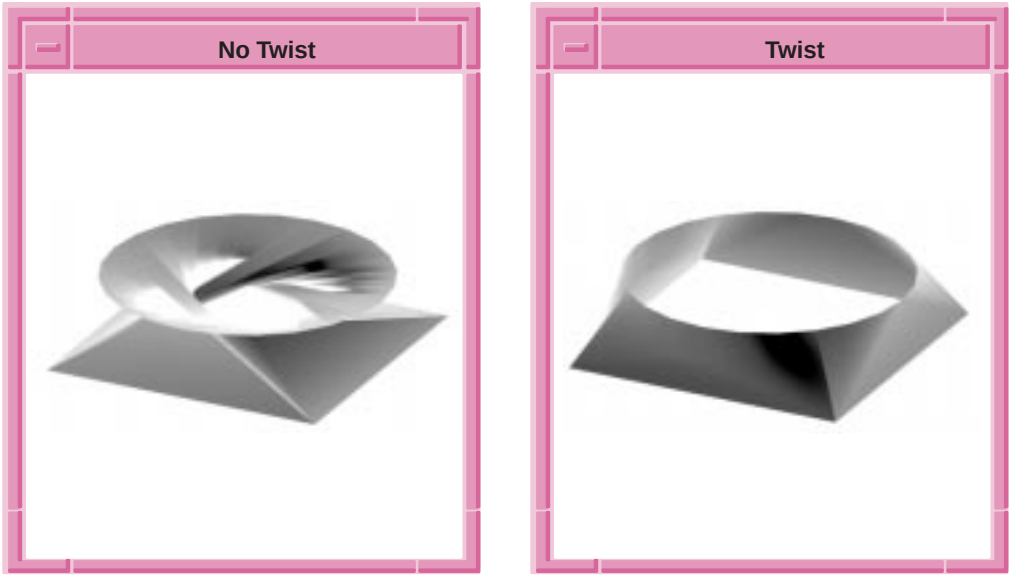

Figure 1-40. Twist Minimization, Rendered

Align Direction Option

Topic: Skinning and Lofting

The *align direction* option may be used to allow the lofting algorithm to align the direction of the wires or coedges in the input list. Refer to Figure 1-41 for an example of alignment of open coedges. Closed loops of wires can also be aligned. The default is aligned.

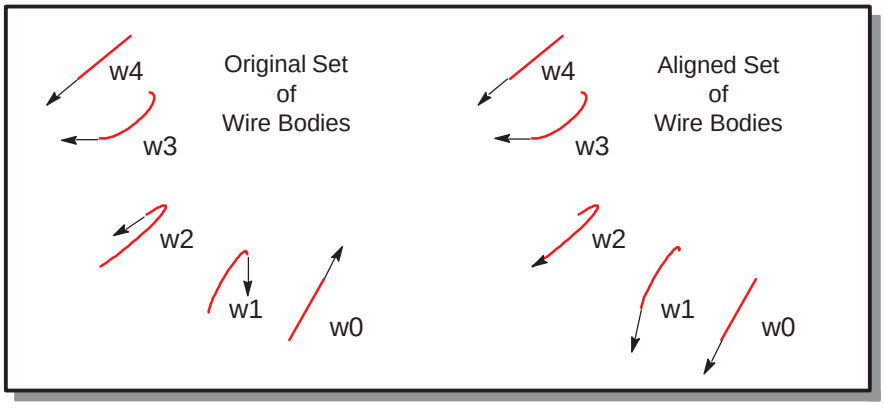

Figure 1-41. Align Direction Option

Perpendicular Option

Topic: Skinning and Lofting

The take–off vector is a tangent vector going out of the starting edge or surface and into the skinned or lofted surface. The *perpendicular* option (for lofting only) is used to specify the direction of the take–off vector, perpendicular to the coedge or in the loft direction. (This removes any restriction that the take–off vector for the loft has to be determined by the cross–product of the coedge tangent vector and the surface normal times the tangent factor.) The default is in the loft direction, because a perpendicular take–off vector can cause self–intersections to the surface.

Scheme Example

In the following example, body0 is made with the perpendicular option and body1 is made without the perpendicular option.

```
(define wire_1 (wire-body (list (edge:linear (position 0 0 0)
     (position 50 0 0))
    (edge:linear (position 50 0 0) (position 50 50 0))
    (edge:linear (position 50 50 0) (position 0 50 0))
    (edge:linear (position 0 50 0) (position 0 0 0)))))
(define wire_2 (wire-body (list (edge:linear (position 100 0 0)
     (position 150 0 0))
    (edge:linear (position 150 0 0) (position 150 50 0))
    (edge:linear (position 150 50 0) (position 100 50 0))
    (edge:circular-3pt (position 100 50 0) (position 90 25 0)
     (position 100 0 0)))))
(zoom-all)
```

```
(sheet:planar–wire wire_1)
(sheet:planar–wire wire_2)
(define coedge1 (list–ref (entity:coedges wire_1) 1))
(define coedge2 (list–ref (entity:coedges wire_2) 3))
(define loft_wire1 (section coedge1 #f 100))
(define loft_wire2 (section coedge2 #t 100))
(define Sect (list loft_wire1 loft_wire2))

(define body0 (sheet:loft–wires Sect #f #t #t #t))
(define body1 (sheet:loft–wires Sect #f #t #t #f))
```

Example 1-9. Sheet:loft–wires showing perpendicular –vs– no perpendicular

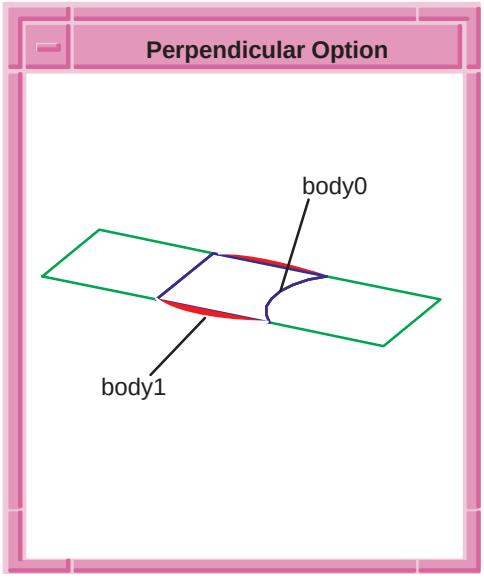

Figure 1-42. Sheet:loft–wires showing perpendicular –vs– no perpendicular

Simplify Option

Topic: Skinning and Lofting

The *simplify* option simplifies the created surface to a conical surface, if applicable. If all of the cross sections lie on a conical surface (plane, cylinder, cone, sphere, or torus), the conical surface is created instead. The SPAresabs variable is used to determine whether or not the cross section lies on an analytical surface (planar, conical, spherical, or toroidal). The default is not simplified.

Closed Option

The *closed* option may be used when the user needs to construct a solid body closed in *v.* A solid body is constructed only when all the wires supplied are closed; otherwise this option is ignored. The default is an open (not closed) body. Figure 1-43 shows an example of a closed skin constructed from four profiles. The surface is continuous at each profile.

If the user provides a set of closed profiles, the face normals of the skin or loft body point outside, away from the body material. When the user provides a set of open profiles, the face normals of the skin or loft face are oriented along the surface normals, and no attempt is made to change the face normal orientation.

Scheme Example

```
(define loop1 (wire-body (list
    (edge:linear (position -20 0 0)(position -20 0 5))
    (edge:linear (position -20 0 5)(position -15 0 5))
    (edge:linear (position -15 0 5)(position -15 0 0))
    (edge:linear (position -15 0 0)(position -20 0 0)))))
(define loop2 (wire-body (list
    (edge:linear (position 0 -20 0)(position 0 -20 5))
    (edge:linear (position 0 -20 5)(position 0 -15 5))
    (edge:linear (position 0 -15 5)(position 0 -15 0))
    (edge:linear (position 0 -15 0)(position 0 -20 0)))))
(define loop3 (wire-body (list
    (edge:linear (position 20 0 0)(position 17.5 0 5))
    (edge:linear (position 17.5 0 5)(position 15 0 0))
    (edge:linear (position 15 0 0)(position 20 0 0)))))
(define loop4 (wire-body (list
    (edge:linear (position 0 20 0)(position 0 17.5 5))
    (edge:linear (position 0 17.5 5)(position 0 15 0))
    (edge:linear (position 0 15 0)(position 0 20 0)))))
; OUTPUT Original

(define skin (sheet:skin-wires
    (list loop1 loop2 loop3 loop4) #t #t #t #f #t #t))
; OUTPUT Result
```

Example 1-10. Skinned Closed Body

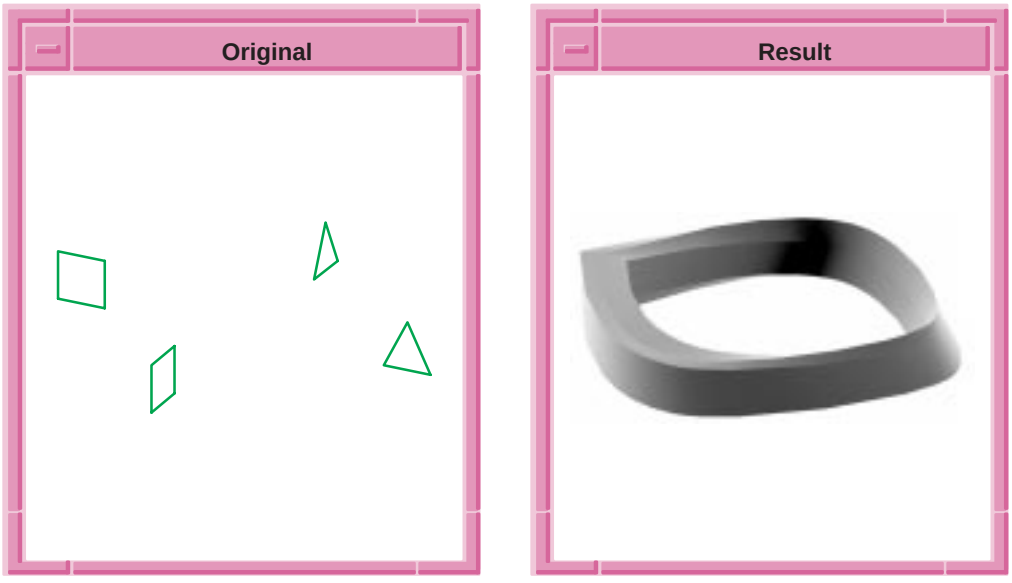

Figure 1-43. Skinned Closed Body

Periodic Option

Topic: Skinning and Lofting

The *periodic* option allows constructing loft bodies that are periodic in *v*. This implies that the loft bodies close back on themselves smoothly (continuously) at the start and end profiles. Setting the closed flag with a value equal to 2 in the skinning APIs activates the periodic option.

In Scheme, the periodic flag is set to #t to achieve the same effect. At least three profiles must be supplied to create a periodic loft body. For example, the periodic skin surface may be constructed from the same set of four profiles used in the preceding example.

Scheme Example

```
(define loop1 (wire-body (list
    (edge:linear (position -20 0 0)(position -20 0 5))
    (edge:linear (position -20 0 5)(position -15 0 5))
    (edge:linear (position -15 0 5)(position -15 0 0))
    (edge:linear (position -15 0 0)(position -20 0 0)))))
(define loop2 (wire-body (list
    (edge:linear (position 0 -20 0)(position 0 -20 5))
    (edge:linear (position 0 -20 5)(position 0 -15 5))
    (edge:linear (position 0 -15 5)(position 0 -15 0))
    (edge:linear (position 0 -15 0)(position 0 -20 0)))))
(define loop3 (wire-body (list
    (edge:linear (position 20 0 0)(position 17.5 0 5))
    (edge:linear (position 17.5 0 5)(position 15 0 0))
    (edge:linear (position 15 0 0)(position 20 0 0)) )))
(define loop4 (wire-body (list
    (edge:linear (position 0 20 0)(position 0 17.5 5))
    (edge:linear (position 0 17.5 5)(position 0 15 0))
    (edge:linear (position 0 15 0)(position 0 20 0)))))
; Options: arcwise #t; minimize_twist #t; align_directions #t;
;   simplify #t; closed #f; solid #f; periodic #t
(define skin (sheet:skin-wires
    (list loop1 loop2 loop3 loop4) #t #t #t #f #f #f #t))
```

Example 1-11. Using the Periodic Option

Virtual Guides

Topic: Skinning and Lofting

Refer to the topic *Skinning with Virtual Guide Curves*.

Solid Option

Topic: Skinning and Lofting

The *solid* option may be used when a solid skin or loft must be constructed but a closed body is not desired. When a closed body is not desired, the end wires are capped with planar faces. The end face normals are oriented away from (outside) the resulting body. The default is a solid body. Figure 1-44 shows an example of a set of profiles using the solid option instead of the closed option.

If the user provides a set of closed profiles, the face normals of the skin or loft body point outside, away from the body material. When the user provides a set of open profiles, the face normals of the skin or loft face are oriented along the surface normals, and no attempt is made to change the face normal orientation.

Scheme Example

```
(define loop1 (wire-body (list
    (edge:linear (position -20 0 0)(position -20 0 5))
    (edge:linear (position -20 0 5)(position -15 0 5))
    (edge:linear (position -15 0 5)(position -15 0 0))
    (edge:linear (position -15 0 0)(position -20 0 0)))))
(define loop2 (wire-body (list
    (edge:linear (position 0 -20 0)(position 0 -20 5))
    (edge:linear (position 0 -20 5)(position 0 -15 5))
    (edge:linear (position 0 -15 5)(position 0 -15 0))
    (edge:linear (position 0 -15 0)(position 0 -20 0)))))
(define loop3 (wire-body (list
    (edge:linear (position 20 0 0)(position 17.5 0 5))
    (edge:linear (position 17.5 0 5)(position 15 0 0))
    (edge:linear (position 15 0 0)(position 20 0 0)))))
(define loop4 (wire-body (list
    (edge:linear (position 0 20 0)(position 0 17.5 5))
    (edge:linear (position 0 17.5 5)(position 0 15 0))
    (edge:linear (position 0 15 0)(position 0 20 0)))))
; OUTPUT Original

(define skin (sheet:skin-wires
    (list loop1 loop2 loop3 loop4) #t #t #t #f #f #t))
; OUTPUT Result
```

Example 1-12. Skinned Solid Body

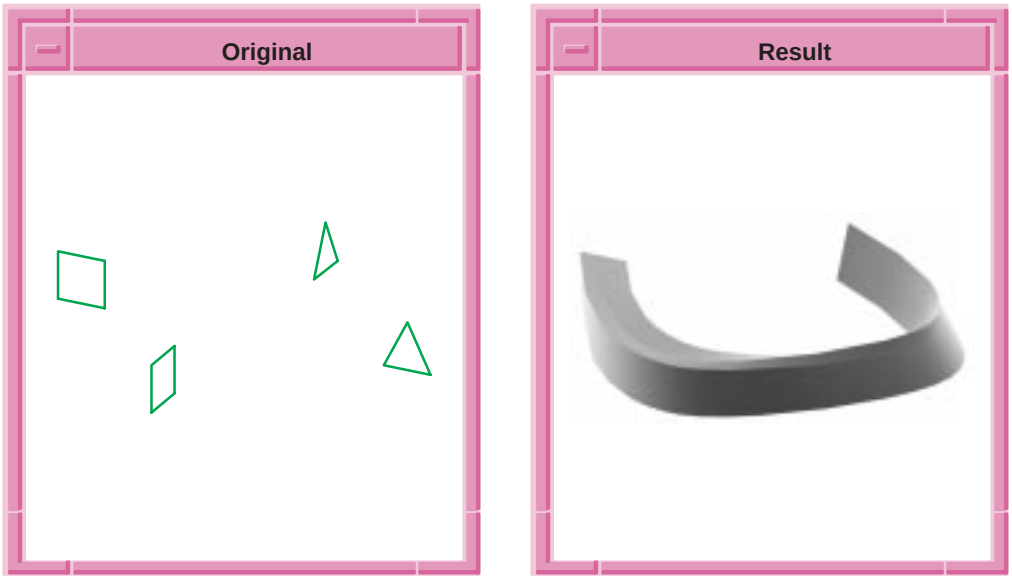

Figure 1-44. Skinned Solid Body

Merge Wire Coedges Option

Topic: Skinning and Lofting

The merge_wirecoedges option may be used to merge G1 vertices of the skinning and lofting wire profiles. This improves operations such as blending and shelling since it reduces the coedge/edge count, which reduces the number of surfaces and eliminates near tangent edges. (Note: this option replaces the global option of the same name in versions 7.0 and latter)

Match Vertices Option

Topic: Skinning and Lofting

The match_vertices option (when FALSE) suppresses the vertex matching algorithm which ensures that all profiles consist of the same number of coedges. A heuristic approach is used to determine which vertex pairs are good matches. Profile coedges are then split where additional vertices are needed. If you have an equal number of edges per skinning profile and do not wish to have any change in topology than set this option to FALSE. However, this option will be forced to TRUE if the coedge numbers of the profiles are not equal. Its default is TRUE. (Note: this option replaces the global options ”match_corners” and ”align_corners” in versions 7.0 and latter)

Loft Estimate Tangent Factors Option

Topic: Skinning and Lofting

If desired, the option loft_estimate_tanfacs can be set, which invokes an algorithm that finds an optimum factor to scale the weight for the tangent factors based on a minimum radius of curvature of the lofted body as a whole. This not only helps to create more pleasing surfaces but insures greater ability to shell and blend lofted models. If you set this option to TRUE, the user–supplied weight values in each Loft_Connected_Coedge_List are then scaled by the optimum scale factor found. (Note: this option replaces the global option of the same name in versions 7.0 and latter)

Guide Pref Option

Topic: Skinning and Lofting

The guide preference option is used in Lofting with Guides and in the Skin with a Draft and Skin with Vectors operations when guides are present. The two possible values of the option are ”constrain to guide” and ”constrain to tangent”. As their names suggest the ”constrain to guide” value will force the surface to follow the guide curve and ignore the tangent constraint in the region of the guide and the ”constrain to tangent” will ignore the guide and constrain to the tangent cone in the region of the tangent cone. The Loft with Guide Curve section contains more details.

Interactive Interface

Topic: *Skinning and Lofting

The interactive programming interface for skinning and lofting allows the application developer to call the individual steps performed in skinning and lofting one at a time to build an interactive skinning or lofting application. An interactive skinning or lofting application allows the end user more control of the skinned or lofted body by using guide curves, mapping curves, preview functions, tangent factors, and local control of profile vertices.

To facilitate this multi–function API, two classes are provided to maintain a “context” or state of the lofting/skinning operation. This state is passed as an object to each of the APIs. More specifically, the first API called is api_create_si for skinning, or api_create_li for lofting, which returns an AcisSLInterface object. This object maintains all the information and the state of the lofting or skinning operation. It is passed in as input to the other APIs and is deleted in the end with a call to api_delete_sli.

Interactive Interface Functions

Topic: *Skinning and Lofting

The interactive interface is essentially the surfacing of more than 20 functions, which perform required or optional skinning and lofting steps.

The following list shows the required steps and their corresponding APIs.

1. Accept as input a list of wire bodies (in the case of skinning) or a loft connected coedge list (in the case of lofting) and build a temporary set of skinning/lofting wires.

   api_create_si or api_create_li

2. Align this set of temp wires so that the wire normals are consistent. (This is explained in detail with the align option.)

   api_align_wires_sli

3. Perform a minimize twist on this set of temp wires so that a non–twisted surface is created. (This is explained in detail with the minimize twist option.)

   api_minimize_twist_wires_sli

4. (Optional) Add a mapping curve.

   api_add_mapping_curve_sli

5. (Optional) Add a guide curve.

   api_add_guide_curve_si

6. Perform a breakup of the coedges of the set of temporary wires so that they all have an equal number of coedges.

   api_breakup_wires_sli

7. (Optional) Preview the edges of the lofted body.

   api_build_edges_sli

8. (Optional) Locally modify the location of a non-corner vertex to control the surface extents.

   api_modify_wire_sli

9. Create surfaces stretched from coedge to coedge and stitch them to make a sheet or solid body.

   api_build_body_sli

10. Delete excess data.

    api_delete_sli

The following group of APIs perform non–incremental optional functions such as adding guide curves, adding mapping curves, computing the tangent factors, and previewing edges.

The following APIs must be called from the interactive interface.

api_align_wires_sli . . . . . Aligns the directions of the wires in the skinning or lofting profiles.

api_build_body_sli . . . . . Builds the sheet body from the data in the lofting interface.

api_breakup_wires_sli . . . . . Creates an equal number of coedges in each wire of the skinning or lofting profiles.

api_create_si . . . . . Creates an AcisSkinningInterface object.

api_create_li . . . . . Creates an AcisLoftingInterface object.

api_delete_sli . . . . . Deletes an AcisSLInterface object.

api_make_wires_sli . . . . . Creates a set of broken up wires used for skinning or lofting.

api_minimize_twist_wires_sli . . . . . Aligns the start vertices of the wires in the skinning/lofting profiles.

The following APIs do not need to be called from the interactive interface.

api_add_guide_curve_si . . . . . Adds a guide curve to a set of skinning profiles.

api_add_mapping_curve_sli . . . . . Adds a guide curve to a set of skinning profiles.

api_add_vertex_sli . . . . . Adds a vertex to each wire in a list of wires.

api_build_edges_sli . . . . . Builds a list of edges that represent the extents of the surfaces if the wires or coedges were to be lofted or skinned.

api_build_faces_sli . . . . . Builds a list of skinning or lofting faces.

api_clear_guide_curves_sli . . . . . Clears the guide curves in the AcisSkinningInterface.

api_clear_mapping_curves_sli . . . . . Removes all the mapping curves from the AcisSLInterface.

api_collapse_wires_sli . . . . . Deletes a degenerate coedge in each wire of a list of wires.

api_estimate_min_rad_curvature_skin . . . Estimates the magnitude of the tangent vector field to build surfaces with a minimum radius of curvature.

api_estimate_tangent_factor_scale_li ... Estimates the optimal magnitude to scale the takeoff vectors on the loft profile cross section.

api_get_tangent_factors_li ..... Gets the current set of tangent factors on the loft profiles.

api_lose_surface_conditions_li ..... Removes the surface conditions from the wires in the lofting profiles.

api_make_mapping_curves_sli ..... Gets a list of the mapping curves that currently exist in the AcisSLInterface.

api_modify_wire_sli ..... Modifies the position of a vertex on a coedge of a wire.

api_remove_mapping_curve_sli ..... Removes a mapping curve from the AcisSLInterface.

api_remove_vertex_sli ..... Removes a vertex from each wire in a list of wires.

api_set_tangent_factors_li ..... Sets the scale factors of the takeoff vectors for the lofting operation.

api_start_vertex_sli ..... Modifies which vertex in a loop of coedges forming a wire is the starting vertex for traversing the loop.

api_valid_start_vertices_sli ..... Gets a list of valid starting vertices for skinning or lofting.

Interactive Interface Examples

Topic: *Skinning and Lofting

The following examples show skinning and lofting applications that can be written with the Interactive Interface. Each requires some form of user input in an intermediate step of skinning or lofting, hence requiring this multi–function (interactive) API.

Interactive Skinning with Guide Curves

Topic: *Skinning and Lofting

Scheme Extensions: slinterface:skin–guide

C++ APIs: api_add_guide_curve_sli

Scheme Example

```
(part:clear)
(define frame1 (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 1 0)))
(define wire_0 (wire–body (list
    (edge:linear (position –60 0 0) (position 60 0 0))
    (edge:linear (position 60 0 0) (position 60 50 0))
    (edge:linear (position 60 50 0) (position –60 50 0))
    (edge:linear (position –60 50 0) (position –60 0 0)))))
(define wire_1 (wire–body (list
    (edge:linear (position –30 0 100) (position 30 0 100))
    (edge:linear (position 30 0 100) (position 30 40 100))
    (edge:linear (position 30 40 100)(position –30 40 100))
    (edge:linear (position –30 40 100) (position –30 0 100)))))
(define wire_3 (wire–body (list (edge:circular
    (position 0 –85 150) 25 0 360))))
(define guide1 (edge:spline (list (position 0 0 0)
    (position 0 20 50) (position 0 0 100) (position 0 –20 160)
    (position 0 –50 160) (position 0 –60 150))))
(zoom–all)
(define interface (slinterface:skinning
    (list wire_0 wire_1 wire_3) #f #t #t #f #t #f #f #t))
(define skinningWires (slinterface:wires interface))
(entity:set–color skinningWires 1)
(slinterface:align–wires interface)
(slinterface:minimizetwist–wires interface)
(slinterface:breakup–wires interface)
(body:reverse (list–ref skinningWires 2))
(slinterface:skin–guide interface guide1)
(define body (slinterface:build–body interface))
(slinterface:delete–interface interface)
(entity:delete (list wire_0 wire_1 wire_3))
(solid:blend–edges (list–ref (entity:edges body) 8) 8)
(solid:blend–edges (list–ref (entity:edges body) 4) 8)
(solid:blend–edges (list–ref (entity:edges body) 13) 8)
(solid:blend–edges (list–ref (entity:edges body) 7) 8)
(shell:sheet–thicken body 3)
```

Example 1-13. Skinning with Guide Curves

Figure 1-45. Skinning with Guide Curves

Mapping Curves with Skinning and Lofting

Topic: *Skinning and Lofting

Scheme Extensions: slinterface:mapping–curve

C++ APIs: api_add_mapping_curve_sli

Mapping curves give the user the ability to control the breakup of skinning and lofting wires without constraining the surfaces. Figure 1-46 shows a set of vertices selected by the end user. After selection, the breakup algorithm guarantees alignment of those vertices.

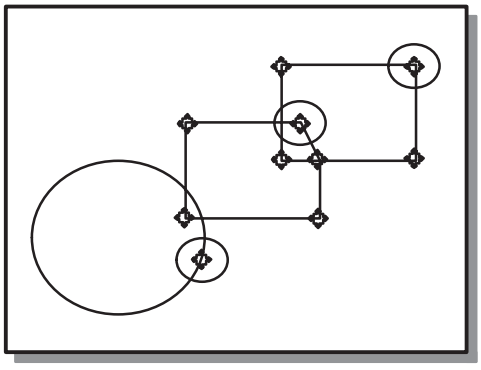

Figure 1-46. The circled vertices were selected

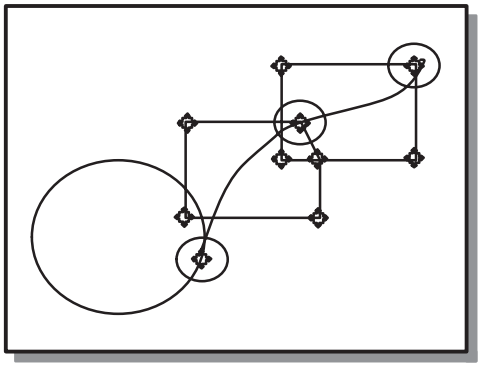

Figure 1-47. Edge created from the mapping curve

The user may select additional vertices to be aligned or may simply start the breakup code at this point. The breakup control of the mapping curves will be identical to the breakup control of the guide curves. For skinning, it is possible to add both mapping curves and guide curves to the same set of wire profiles.

A large degree of preprocessing is needed to validate the input of the mapping curves. The following limitations apply:

- A vertex can only be included in one mapping curve.
- Mapping curves cannot cross.
- A vertex on each wire profile must be designated.
- Vertices must be given in order, from the first wire to the last, or from the last to the first.
- No more than one vertex per wire per guide curve may be given.

Figure 1-48 shows the flow of control from user input to breakup. The mapping attributes are entity attributes that are placed on the vertices of the coedges and read by the breakup code.

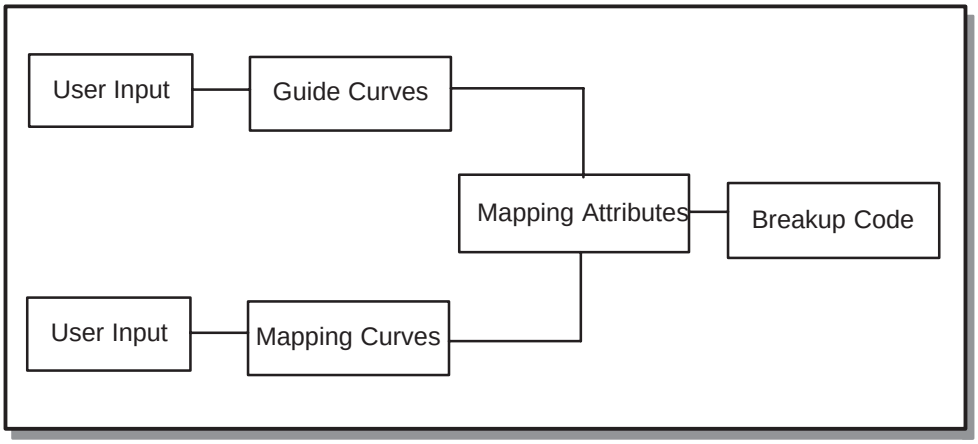

Figure 1-48. Flow of control from user input to breakup

Scheme Example

```
; Wire–body 1
(part:clear)
(define frame1 (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 1 0)))
(define wire1 (edge:circular (position 0 0 0) 25 0 360))
(define wire2 (edge:circular (position 20 –20 0) 25 0 90))
(define wire3 (edge:circular (position 20 –20 0) 20 0 90))
(define twoEdges (edge:split wire2 (car
    (curve:intersect wire1 wire2))))
(entity:delete (cadr twoEdges))
(define twoEdges (edge:split wire3 (car
    (curve:intersect wire1 wire3))))
(entity:delete (cadr twoEdges))
(define twoEdges (edge:split wire1 (car
    (curve:intersect wire1 wire2))))
(entity:delete (car twoEdges))
(define wire1 (cadr twoEdges))
(define twoEdges (edge:split wire1 (car
    (curve:intersect wire1 wire3))))
(entity:delete (cadr twoEdges))
(define twoEdges (edge:split wire1 (car
    (curve:intersect wire1 wire3))))
(entity:delete (cadr twoEdges))
(define wire4 (edge:linear (edge:start wire2)
    (edge:start wire3)))
(define wireBody1 (wire–body (list wire1 wire2 wire4 wire3)))
```

```
; Wire–body 2
(define wire1 (edge:circular (position 0 0 –60) 10 0 360))
(define wire2 (edge:circular (position 5 –23.5 –60) 25 65 90))
(define wire3 (edge:circular (position 5 –23.5 –60) 21 65 90))
(zoom–all)
(define twoEdges (edge:split wire2 (car
    (curve:intersect wire1 wire2))))
(entity:delete (cadr twoEdges))
(define twoEdges (edge:split wire3 (car
    (curve:intersect wire1 wire3))))
(entity:delete (cadr twoEdges))
(define twoEdges (edge:split wire1 (car
    (curve:intersect wire1 wire2))))
(entity:delete (car twoEdges))
(define wire1 (cadr twoEdges))
(define twoEdges (edge:split wire1 (car
    (curve:intersect wire1 wire3))))
(entity:delete (cadr twoEdges))
(define twoEdges (edge:split wire1 (car
    (curve:intersect wire1 wire3))))
(entity:delete (cadr twoEdges))
(define wire4 (edge:linear (edge:start wire2)
    (edge:start wire3)))
(define wireBody2 (wire–body (list wire1 wire2 wire4 wire3)))
; Start the skinning operation.
(define interface (slinterface:skinning
    (list wireBody1 wireBody2)))
(define skinningWires (slinterface:wires interface))
(entity:set–color skinningWires 0)
(slinterface:align–wires interface)
(slinterface:minimizetwist–wires interface)
(define positionList (list (position 40 –20 0)
    (position 13.8749834965547 –4.46753647223035 –60)))
(slinterface:mapping–curve interface positionList)
(define positionList (list (position 45 –20 0)
    (position 15.5654565435175 –0.842305324083753 –60)))
(slinterface:mapping–curve interface positionList)
(slinterface:breakup–wires interface)
(define body (slinterface:build–body interface))
(slinterface:delete–interface interface)
```

Example 1-14. Mapping Curves

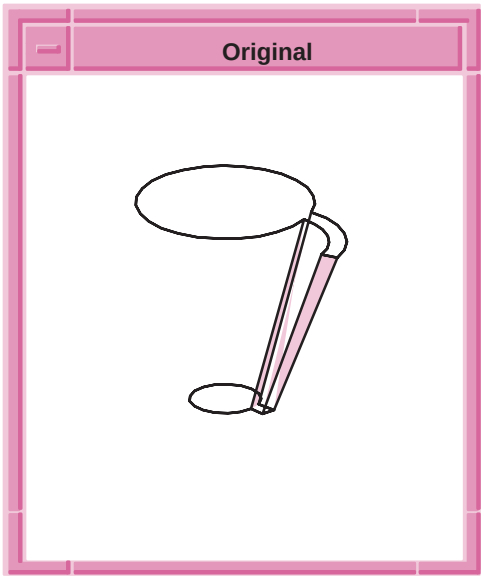

Figure 1-49. Mapping Curves

Estimating Tangent Factors for Lofting

Topic: *Skinning and Lofting

Scheme Extensions: slinterface:get–tanfac–scale, slinterface:set–tan–facs

C++ APIs: api_estimate_tangent_factor_scale_li, api_set_tangent_factors_li

The Advanced Surfacing Component uses APIs and Scheme extensions to estimate the magnitude of weight factors.

Using APIs

Loft bodies can be shelled only to a thickness smaller than their minimum radius of curvature, api_estimate_min_rad_curvature_skin allows users to estimate the maximum possible shelling thickness. Also, edges of loft bodies cannot be filleted by blend radii that are larger than the minimum radius of curvature of the loft body. This API should also help users in selecting the appropriate blend radius for edges of loft bodies.

The api_estimate_tangent_factor_scale_li allows the user to estimate the magnitude of the take-off vectors on each of the loft profile cross–sections so that the resulting loft body has as large a minimum radius of curvature as possible. Maximizing the minimum radius of curvature should enable the loft bodies to be shelled to a greater thickness and their edges to be filleted with greater blend radii. It is important to note that this API provides a *single* scale factor that should be applied simultaneously to *all* take-off vectors.

There are four arguments to this API—one input and three outputs, the last two being optional. The input argument to this API is a pointer to an AcisLoftingInterface object. The outputs from the API are: (a) a range of reasonable values by which the take–off vectors may be scaled; (b) the optimum scale factor value in the range which maximizes the minimum radius of curvature and; (c) the minimum radius of curvature of the loft corresponding to the optimum scale factor. The typical use of this API is as follows:

1. Create an instance of AcisLoftingInterface using api_create_li. The inputs to this API are the same as inputs to api_loft_coedges. A populated AcisLoftingInterface object is created as a result.

2. Use api_estimate_min_rad_curvature_skin to estimate the minimum radius of curvature of the loft body.

3. If the minimum radius of curvature of the loft body is satisfactory then there is no need to re–scale the magnitudes of the take-off vectors.

4. Otherwise, use api_estimate_tangent_factor_scale_li to estimate a reasonable range for the scale factor if that is sufficient, or estimate an *optimal* value for the scale factor and the corresponding minimum radius of curvature of the loft body.

5. Re–scale the tangent factors of the individual profiles by the scale factor determined in Step 4.

6. Reset the new tangent factors of the wire profiles in the AcisLoftingInterface object using api_set_tangent_factors_li.

7. Build the loft body using api_build_body_sli. The resulting loft body will have close to the largest minimum radius of curvature that is possible under the given law/surface constraints.

The API api_estimate_tangent_factor_scale_li is interfaced into Scheme through the Scheme extension slinterface:get–tanfac–scale. This Scheme extension takes as argument a slinterface object that describes the loft and a Boolean value. If the Boolean is true the API estimates a range of scale factor values and an optimal value for the scale factor in this range, otherwise (Boolean = false) it returns only a range of scale factor values. Consequently the API works faster when the Boolean is set to false.

The API api_estimate_min_rad_curvature_skin is interfaced into Scheme through the Scheme extension slinterface:min–rad. This Scheme extension takes as argument a slinterface object that describes the loft and returns the minimum radius of curvature of the loft body.

The tangent factors on the loft body may be reset at any time using the API api_set_tangent_factors_li or its equivalent Scheme extension slinterface:set–tan–facs. This Scheme extension takes as argument a slinterface object that describes the loft and the factor by which the weights on all the individual sections need to be scaled.

Scheme Example

The following Scheme example demonstrates estimating the magnitude of the take–off vectors on two profiles.

```
; Construct a rectangular wire body.
(define w1 (wire–body:points (list (position 0 0 0)
    (position 50 0 0) (position 50 50 0)
    (position 0 50 0) (position 0 0 0))))
; Make a wire body from a single position.
(define w2 (wire–body:points (list (position 0 0 75))))
; Specify a normalized vector field along the z–axis.
(define dom1 (law "DOMAIN (VEC (0,0,1),0,50)"))
; Get the list of coedges of wire 1.
(define coedge1 (entity:coedges w1))
; Apply the vector field to the edges of wire1.
(define loft_wire1 (section coedge1
    (list dom1 dom1 dom1 dom1) #t 1))
; Get the list of coedges of wire 2.
(define coedge2 (entity:coedges w2))
; Define a normalized radial vector field in the xy – plane.
(define dom2 (law
    "DOMAIN (–VEC (COS (X),SIN (X),0),0,6.28318530717959)"))
; Apply the vector field to the edges of wire 2.
(define loft_wire2 (section coedge2 (list dom2 ) #t 1))
; Combine the coedge lists together in one list.
(define sect (list loft_wire1 loft_wire2 ))
; Set the following lofting options:
; arc_length = TRUE
; minimize_twist = TRUE
; align_directions = TRUE
; perpendicular = FALSE
; simplify = FALSE
; solid = FALSE
; closed = TRUE
; Define the lofting interface object using the coedge lists
; and the lofting options.
(define interface (slinterface:lofting
    sect #t #t #t #f #f #t #f))
; Create the loft body.
(define loftBody (slinterface:build–body interface))
; Estimate the minimum radius of curvature of the loft body.
(define min_rad1 (slinterface:min–rad interface))
; Display value of minimum radius of curvature.
```

```
; min_rad1
; Obtain a reasonable range for the tangent factor magnitude
; and get the optimal magnitude.
(define reallist (slinterface:get–tanfac–scale interface #t))
(define scale (list–ref reallist 2))
; Apply the optimal tangent factor magnitude.
(slinterface:set–tan–facs interface scale)
; Build the loft body with the newly estimated tangent
; factor scale.
(define loftBody (slinterface:build–body interface))
; Get the radius of curvature of the new body.
(define min_rad2 (slinterface:min–rad interface))
; Display value of minimum radius of curvature min_rad2.
; Delete the lofting interface object.
(slinterface:delete–interface  interface)
```

Example 1-15. Take–off Vector Changes

Figure 1-50. Take-off Vector Changes

Local Vertex Modification with Skinning and Lofting

Topic: *Skinning and Lofting

Scheme Extensions: slinterface:modify–vertex, slinterface:build–edges

C++ APIs: api_modify_wire_sli, api_build_edges_sli

Local vertex modification allows the user to move selected vertices along the coedges they separate. Generally speaking, after the call to slinterface:breakup-wires, additional vertices and coedges will be generated on one or more of the skinning or lofting profiles. With a call to slinterface:modify-vertex, a vertex can be repositioned along its neighboring coedges. In Figure 1-51, which shows the use of the skinning and lofting modify vertex command, the circled vertex can be moved along the wire profile in either direction.

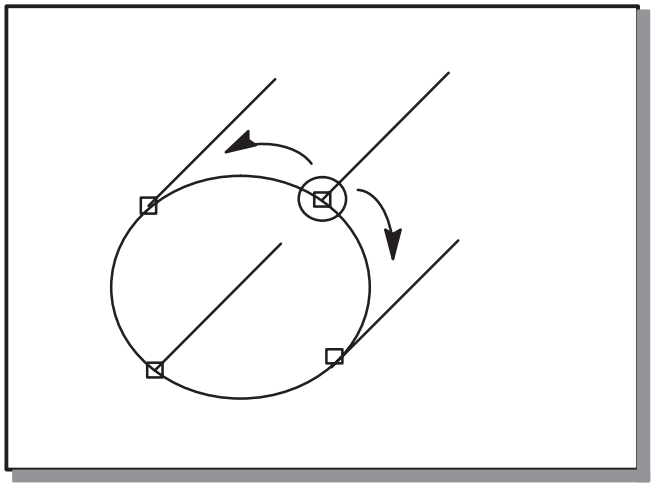

Figure 1-51. Local Vertex Modification

Scheme Example

```
; Build the necessary geometry for a sweep.
(define frame1 (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 1 0)))
(entity:set-color frame1 1)

(define frame2 (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 0 1)))
(wcs:set-active frame2)
(entity:set-color frame2 1)
(define wire_5 (wire-body (list (edge:circular
    (position 50 50 60) 20 0 360))))
(define wire_6 (wire-body (list (edge:circular
    (position 50 50 90) 20 0 360))))
(zoom-all)
```

```
(wcs:set–active frame1)
(define wire_1 (wire–body (list (edge:linear
    (position –20 30 0) (position 120 30 0))
    (edge:linear (position 120 30 0) (position 120 20 0))
    (edge:linear (position 120 20 0) (position –20 20 0))
    (edge:linear (position –20 20 0) (position –20 30 0)))))
(define dom1 (law "domain (vec (0, 0, 1), 0, 140)"))
(define dom2 (law "domain (vec (0, 0, 1), 0, 10)"))
(define dom3 (law "domain (vec (0, 0, 1), 0, 140)"))
(define dom4 (law "domain (vec (0, 0, 1), 0, 10)"))
(define laws1 (list dom1 dom2 dom3 dom4))

(define wire_2 (wire–body (list (edge:linear (position 10 30 30)
     (position 90 30 30))
    (edge:linear (position 90 30 30) (position 90 20 30))
    (edge:linear (position 90 20 30) (position 10 20 30))
    (edge:linear (position 10 20 30) (position 10 30 30)))))
(define dom1 (law "domain (vec (0, 0, 1), 0, 80)"))
(define dom2 (law "domain (vec (0, 0, 1), 0, 10)"))
(define dom3 (law "domain (vec (0, 0, 1), 0, 80)"))
(define dom4 (law "domain (vec (0, 0, 1), 0, 10)"))
(define laws2 (list dom1 dom2 dom3 dom4))
(zoom–all)

(define wire_3 (wire–body (list (edge:linear (position 20 20 50)
     (position 80 20 50))
    (edge:linear (position 80 20 50) (position 80 13 43))
    (edge:linear (position 80 13 43) (position 20 13 43))
    (edge:linear (position 20 13 43) (position 20 20 50)))))
(define dom1 (law "domain (vec (0, –0.7, 0.7), 0, 40)"))
(define dom2 (law "domain (vec (0, –0.7, 0.7), 0, 10)"))
(define dom3 (law "domain (vec (0, –0.7, 0.7), 0, 40)"))
(define dom4 (law "domain (vec (0, –0.7, 0.7), 0, 10)"))
(define laws3 (list dom1 dom2 dom3 dom4))
(zoom–all)

(define coedges1 (entity:coedges wire_1))
(define coedges2 (entity:coedges wire_2))
(define coedges3 (entity:coedges wire_3))
(define coedges5 (entity:coedges wire_5))
(define coedges6 (entity:coedges wire_6))

(define section1 (section coedges1 laws1 #f 100))
(define section2 (section coedges2 laws2 #f 50))
(define section3 (section coedges3))
(define section5 (section coedges5))
(define section6 (section coedges6))
```

```
; Start the lofting step.
(define sect (list section2 section3 section5 section6))
(define interface (slinterface:lofting sect))
(define tempWires (slinterface:wires interface))
(entity:set-color tempWires 1)

; Align, breakup and minimize the temporary lofting wires.
(slinterface:align-wires interface)
(slinterface:minimizetwist-wires interface)
(slinterface:breakup-wires interface)
; Spend some time looking at the edges.
(define previewEdges (slinterface:build-edges interface))
; Delete the edges.
(entity:delete previewEdges)

; Move some vertices around
(slinterface:modify-vertex interface (list-ref tempWires 3)
    (position 30.2667875017707 -90 53.255813953488)
    (position 40.288231 -90 67.483751))
(slinterface:modify-vertex interface (list-ref tempWires 3)
    (position 69.7332124982292 -90 53.255813953488)
    (position 59.71176899999 -90 67.483751))
(slinterface:modify-vertex interface (list-ref tempWires 3)
    (position 69.7332124982292 -90 46.744186046511)
    (position 66.098883 -90 61.867012))
(slinterface:modify-vertex interface (list-ref tempWires 3)
    (position 30.2667875017708 -90 46.744186046511)
    (position 33.901117 -90 61.867012))

(slinterface:modify-vertex interface (list-ref tempWires 2)
    (position 30.2667875017707 -60 53.255813953488)
    (position 40.288231 -60 67.483751))
(slinterface:modify-vertex interface (list-ref tempWires 2)
    (position 69.7332124982292 -60 53.255813953488)
    (position 59.71176899999 -60 67.483751))
(slinterface:modify-vertex interface (list-ref tempWires 2)
    (position 69.7332124982292 -60 46.744186046511)
    (position 66.098883 -60 61.867012))
(slinterface:modify-vertex interface (list-ref tempWires 2)
    (position 30.2667875017708 -60 46.744186046511)
    (position 33.901117 -60 61.867012))
```

```
; Spend some time looking at the edges.
(define previewEdges (slinterface:build-edges interface))
; Delete the edges.
(entity:delete previewEdges)
; We like them now, so build the body.
(define body (slinterface:build-body interface))
(slinterface:delete-interface interface)
```

Example 1-16. Modify Vertex

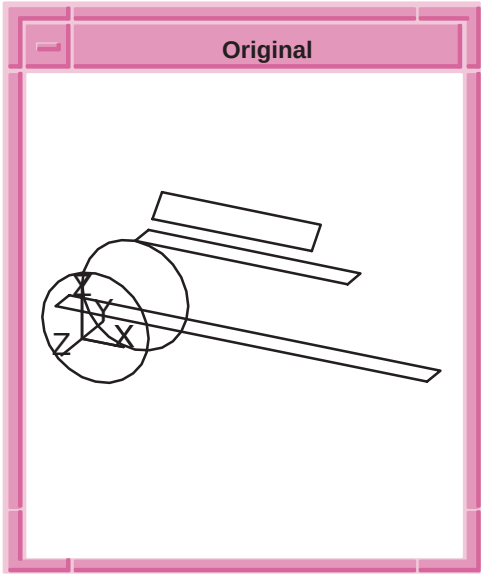

Figure 1-52. Modify Vertex

Journaling for Skinning and Lofting

Topic: *Skinning and Lofting

The Advanced Surfacing Component contains its own specific version of journaling to support bug reporting. Each C++ API is journaled, via Scheme, when the feature is turned on. Each specific skinning or lofting operation should be performed in its own journaling stream.

Set the global option skinning_journal to TRUE to start journaling. AS will create a Scheme journaling script replicating the exact skinning and lofting APIs called in an ACIS C++ application. A Scheme file, skin#.scm, and a SAT file, skin#.sat, will be created in the working directory. The generated Scheme script(s) and SAT file(s) can be used in Scheme AIDE to debug problems with the skinning. Set the global option skinning_journal to FALSE to turn off journaling.

Net Surfaces

Topic: *Skinning and Lofting

Net surfacing is an advanced surfacing technique that uses a network of curves defining both the *u* and *v* surface parameter directions (often called “bi–directional” curves because of this), unlike skinning and lofting which only take curves defining the *u* parameter. This allows a large degree of control over the final shape of the surface.

For net surfacing, the input curves are provided in the form of wire bodies. There must be at least four wire bodies, two in each direction. These wires define the cross sections to be interpolated by the resulting sheet body (Figure 1-53).

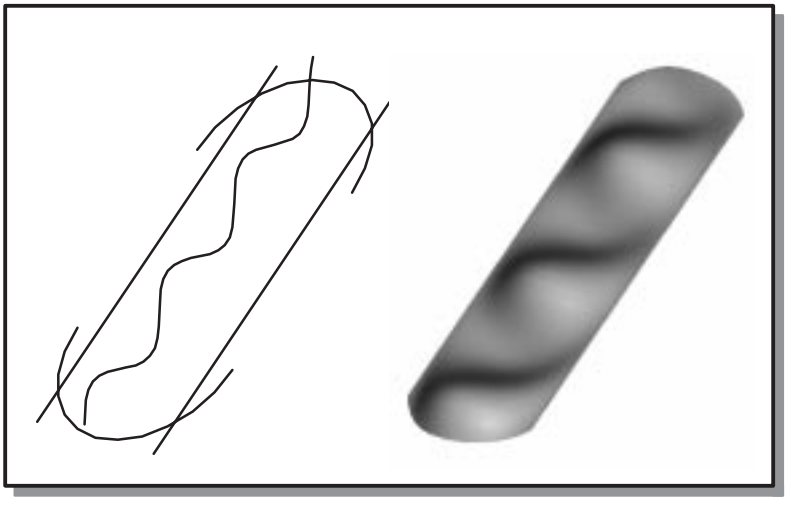

Figure 1-53. Net Surface

If all of the curves intersect, then the surface passes through the curves and their intersections. If any of the *u* curves of the net do not intersect all of *v* curves at some point, the intersection is interpolated. The maximum distance for the interpolation is governed by a tolerance argument to the api_net_wires. The default for this tolerance value is SPAresfit. If the tolerance is changed, a net surface can be created as long as the distance between *u* curves or between *v* curves (e.g., in the skin direction) is larger than the tolerance intersection distance between a *u* curve and a *v* curve. Thus, the curve interpolating accuracy of the net surface is controlled by the user's accuracy of the intersections of the cross sections. Refer to Figure 1-54 for an illustration of net wires.

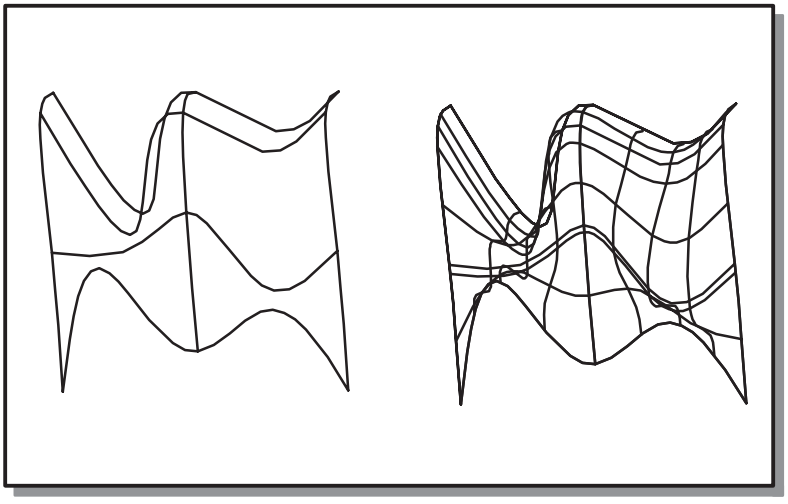

Figure 1-54. Net Wires

Curves in the resulting surface that extend beyond the intersections are trimmed. However, shorter curves are not extended to meet the intersection points. The original curves are not altered, but the surface defined from the net surface is bounded by the trimmed curves, as shown in Figure 1-55. This also illustrates interpolation. In this example, line 4 is not in the same plane as lines 1, 2, and 3 and does not intersect them, so the resulting surface is placed in between the two planes. (The offset is exaggerated for effect.)

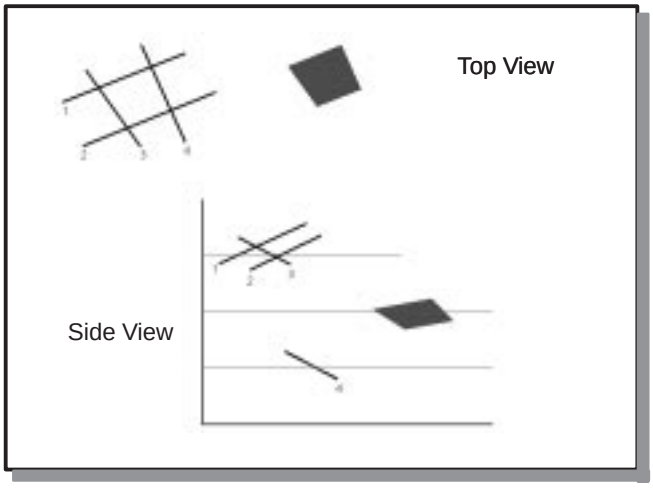

Figure 1-55. Net Surface Interpolation and Trimming

The api_net_wires function creates the network surface. It requires the number of wires in the *u* direction, a pointer to the first wire body in the *u* list, the number of wires in the *v* direction, a pointer to the first wire body in the *v* list, an address for the resulting surface body, and flags for alignment and simplification.

The optional argument for align_directions specifies whether or not to line up the directions of the curves in both the *u* and *v* directions listed in the body_list. When this option is on, the directions of open and closed curves in the *u* or *v* direction are changed to line up with the first curves in the given direction. After all of the curves in the *u* and v directions are aligned, the order and directions of the *v* curves are modified to insure the proper surface parameter orientation. This insures that the start of the first *v* curve intersects the start of the first *u* curve. The start of the curve in the *v* direction is changed to be the start of the curve in the *u* direction.

The optional argument simplify specifies whether or not to simplify the resulting surface geometry. If appropriate, the result is a planar surface bounded by the first and last *uv* wires.