Chapter 1. Blending Component

Component: *Blending

The Blending Component (BLND), in the blnd directory, implements standard blending operations. The sharp edges and vertices in models must often be replaced by faces in order to improve the model, to soften sharp edges and corners, and to create smooth transitions from one surface to another. In ACIS this operation is called *blending*.

BLND provides *standard blending* functionality, which is described in this manual, and includes:

- Chamfer on an edge.
- Constant radius blend on an edge.
- Variable radius blend on an edge.

The Advanced Blending Component provides other, more advanced blending functionality, which is described in the *Advanced Blending Component Manual*.

Blending functionality is determined by three major areas of blend definition:

- Type of blend geometry constructed (the type of geometry that is added to the model to form the blend).
- Blend geometry boundary conditions (how the blend geometry fits into the existing model).
- Interaction and interference between the blend geometry and the model (how the blends interact with each other and with the model).

BLND attaches attributes to entities to be blended to store such blend information with the model.

Creating a Blend

Topic: *Blending

The following steps describe the general procedure ACIS follows in creating a blend:

1. Create a *blend sheet*, which is a sheet of surfaces that are used to form the blend.

2. Attach the blend sheet to the model at the blend location. This involves trimming or extending the original faces and replacing portions of original surfaces with surfaces from the blend sheet.

The blend sheet is generally created in a single operation. However, the application can take control of the blend process and construct the blend sheet face-by-face, allowing the blend sheet to be displayed in stages as it is constructed.

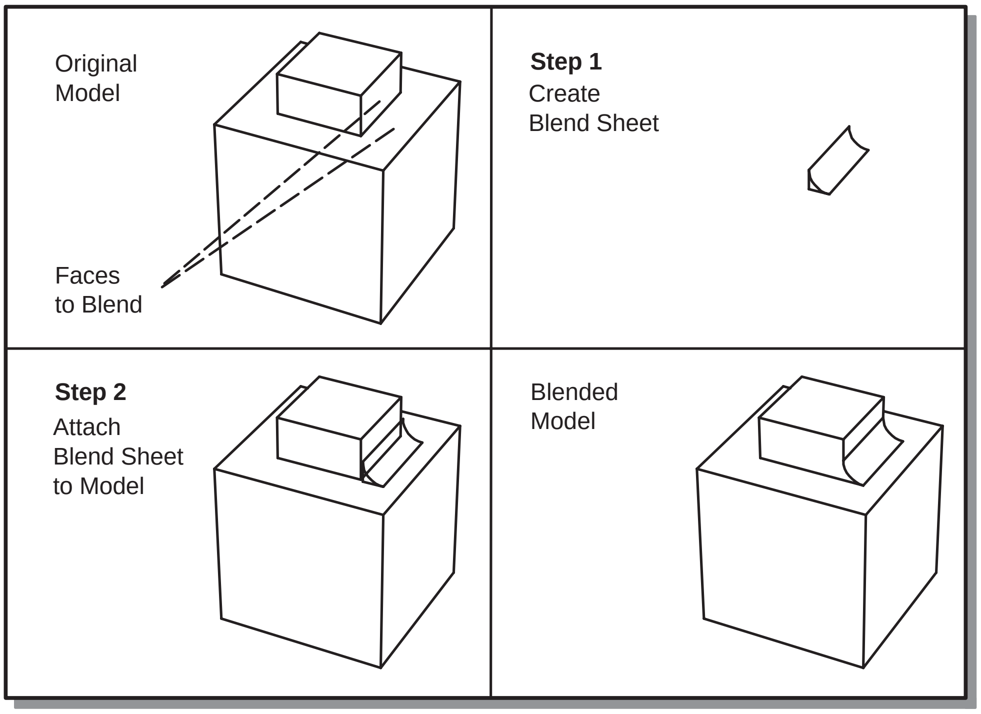


Figure 1-1. Creating a Blend

Single Stepping

Topic: *Blending

An application can provide feedback on the progress of blending by incrementally sketching the blend graph as it is being made by single stepping through the blend operation.

The basic strategy for single stepping is:

1. Attach blend attributes to the entities which are to participate in the blend, as usual.

2. Call function api_init_blend_ss with an entity list containing the blend attributes to consider during the blend operation.

3. Call function api_do_one_blend repeatedly, specifying an index into the entity list (of the attribute to process), until the list is exhausted. This list may grow longer than that initially constructed, as the blend operation may add attributes to the list itself. New geometry is added to the sheet body for each call to this API.

4. After all items on the list have been processed, call function api_concl_blend_ss to make a sheet body for the list of blend attributes in single steps.

5. Finally, make the usual function calls to complete the blend.

Adding or Removing Material

Topic: *Blending

Blending may either add material to or remove material from a model, depending on the model convexity local to the blend.

A blend on a convex edge *removes* material from the model, as shown in Figure 1-3. This rounds out an external corner, removing excess material. This type of blend is sometimes referred to as a *round*.

A blend on a concave edge *adds* material to the model, as shown in Figure 1-2. This type of blend is sometimes referred to as a *fillet*.

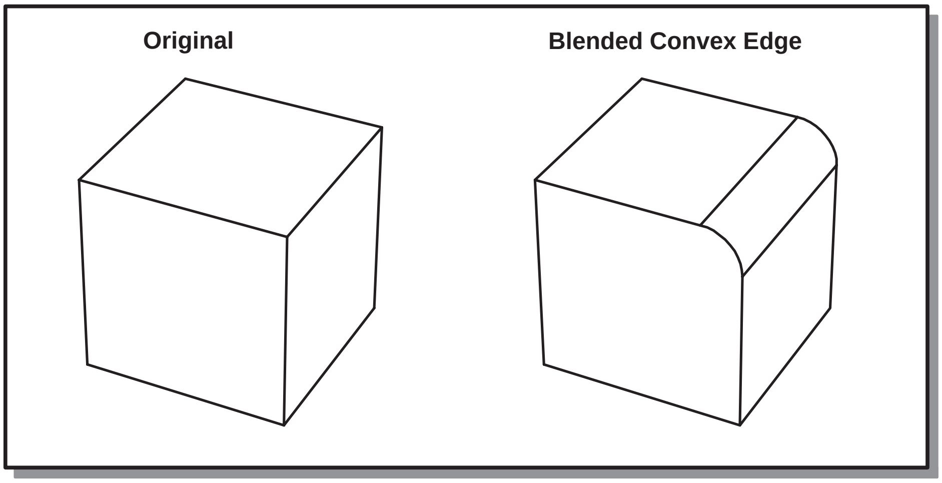


Figure 1-2. Removes Material from Convex Edge

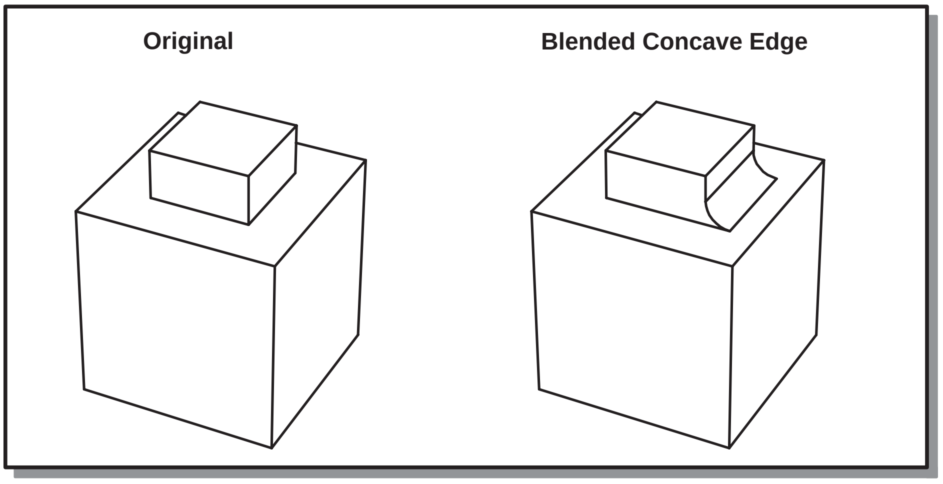


Figure 1-3. Adds Material to Concave Edge

Rolling Ball

Topic: *Blending

A figurative *rolling ball*, which maintains contact with the surfaces to be blended, is used to compute a blend. Figure 1-4 illustrates the figurative rolling ball used to blend faces from two stacked blocks.

General blends between wholly disconnected faces are not supported in the Blending Component (they are in the Advanced Blending Component). Blends between faces that are connected by one or more intervening faces are sometimes supported in the Blending Component (refer to section *Remote Blending*).

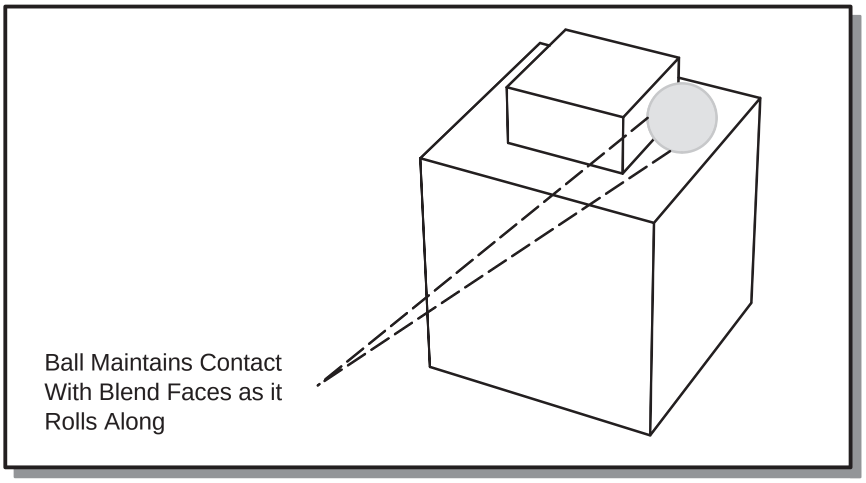


Figure 1-4. Rolling Ball

The curve swept out by the rolling ball’s center is called its *spine curve*; the two curves of contact are called *spring curves*.Spring curves are traced by the contact points between the rolling ball and the blend faces. Figure 1-5 illustrates the spine curve and a spring curve traced by a rolling ball.

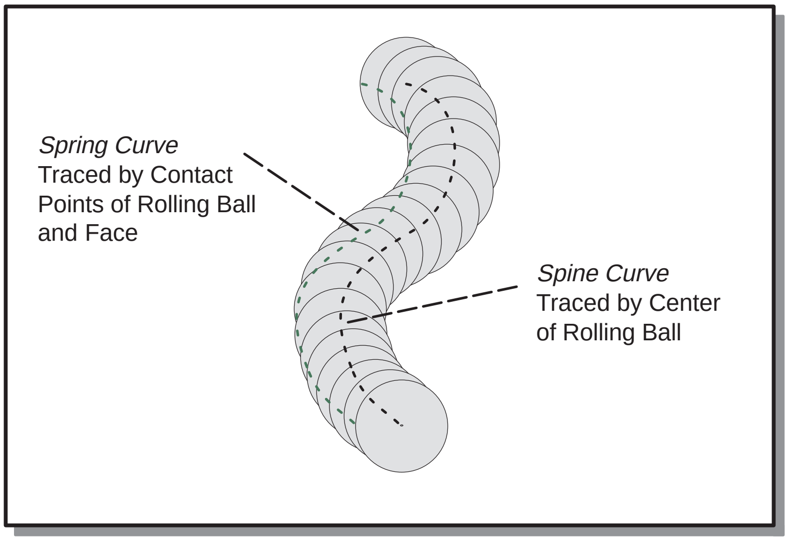


Figure 1-5. Spine and Spring Curves

Blend Geometry

Topic: *Blending

ACIS allows round blends, chamfer blends, and vertex *n*-sided patches (Gregory Polygon surfaces).

Radius

The *blend radius* refers to the radius of the figurative rolling ball. The blend radius may either be constant or variable. Any combination of variable radius and constant radius blends may exist in a sequence of edges to be blended, as long as they always meet one another with matching radius values.

Cross Section

A blend between two entities has a characteristic shape which is defined by the cross section of the blend surface. The *cross section* is the shortest line along the blend surface that connects the two faces being blended. The cross section is always perpendicular to both faces in the blend.

A circular (round) blend has a circular cross section. The cross section of a chamfer blend is a straight line.

Round Blends

Topic: *Blending

A *round blend* refers to the surface produced by either of the following:

- Rolling a ball of constant or variable radius along an edge while keeping the ball in contact with the faces to either side of the edge.
- Rolling a ball of constant radius around a vertex while keeping the ball in contact with a face and an edge adjacent to the vertex.

In general, the envelope of the rolling ball's path is represented by a spline surface. When possible, it is represented by a simple analytic surface. In either case, the surface is tangent to the two faces and has precise evaluators.

Spring curves follow the point of contact between the pipe and the two faces. The portion of a pipe surface that lies between the spring curves is used to form the round blend face.

Note *The phrase* between the spring curves *refers to the portion of the surface with smaller arc length.*

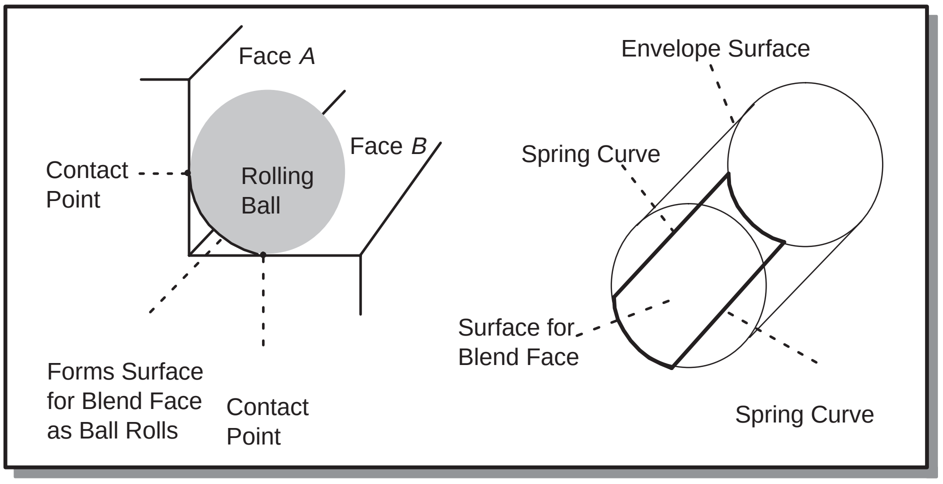


Figure 1-6. Blend Face Between Spring Curves

Constant Radius Blending

Topic: Blending

When the blend radius is constant, the spine and spring curves are obtained by:

- Offsetting the two body surfaces by the radius distance.
- Intersecting the two offset surfaces to find the blended spine.
- Projecting the spine onto each of the two body surfaces to find the spring curves.

Constant radius blends are represented by the rolling ball surface type (rb_blend_spl_sur). Older versions of ACIS used a pipe surface (pipe_spl_sur), and this behavior can be recovered by setting the option rb_replace_pipe to off (false). However, applications are strongly discouraged from changing this option.

The surface of a constant radius blend is represented by either a spline surface or a simple analytic surface. The spring curves bound the portion of the surface used for the blend.

In general, the blend surface is represented by a spline. Under certain geometric conditions, (i.e., when the spine curve is a simple curve type), the blend surface is represented by a simple analytic surface. For example:

If the spine is a:	The surface is a:
straight line	cylinder
circular arc	torus
point	sphere

The corresponding points on the spine and spring curves (i.e., the center and the two contact points of the rolling ball at a given position) determine a plane. The spine curve is perpendicular to this plane. Dropping a perpendicular from a point on the spine curve onto the other two curves provides the corresponding points.

Variable Radius Blending

Topic: Blending

Allowing the blend radius to vary introduces some fundamental differences into the algorithms. The offset-intersection algorithm used to find the spine and spring curves works only for constant-radius blends, because a variable offset is ill-defined on a surface. A figurative rolling ball must be set up at the desired position to evaluate the curves and the blend surface, and a marching algorithm is required to find the approximating curves and surface. Analytic surfaces cannot be used to represent even simple variable radius blends. The spine and spring curves are no longer perpendicular to the plane defined by corresponding points.

When a variable radius function is specified, there must be some way to indicate exactly where along the blend a particular radius value is obtained. ACIS uses the geometry of the edge being blended to calibrate this. The parameter of the radius function is the parameter of the edge curve. At a given *v*-parameter, a ball of radius $r(v)$ is set up so that its center is in the plane perpendicular to the edge curve, evaluated at *v*.

Variable Radius Blend Surface Types

Standard blending supports two mathematical definitions of variable radius blend surfaces. The first is a *rolling ball snapshot* blend surface. The second type (introduced in Release 5.0) is a *rolling ball envelope* blend surface. Although geometrically similar, sequences of the rolling ball envelope surface are smoother, and their offsets are better behaved. Their spring curves are identical, but the cross sections have a subtly different shape. This allows the rolling ball envelope to be perfectly smooth across transitions between blend surfaces in a smooth sequence of edges.

The Advanced Blending Component provides a third mathematical definition of a variable radius blend surface called a *sliding disc*. Both the rolling ball snapshot and the sliding disc surface types have a slight crease between faces if the blend rolls across an edge that is smooth but not curvature-continuous and if the radius is varying as it crosses the edge. This can cause near-tangency problems when the surfaces are offset. The rolling ball envelope surface addresses this problem and permits robust performance in operations that use surface offsets.

The type of variable radius blend surface created (for standard blending only) is controlled by option bl_envelope_surf. If this option is on (true), a rolling ball envelope blend surface is created. If it is off (false), a rolling ball snapshot blend surface is created. However, applications are discouraged from changing this option setting.

Simplest Variable Radius Blend Surface Used

The variable radius blend algorithms construct the simplest blend surface that can represent the geometry requested. For example, if a variable radius blend is specified with a start and an end radius value, and these values are the same, then a constant blend will be made. This behavior is controlled by the option blend_make_simple, but applications are strongly discouraged from changing this option setting.

Chamfer Blends

Topic: *Blending

A *chamfer blend* refers to the ruled surface as swept out by the line drawn between the two contact points of the rolling ball. An asymmetric chamfer results from two balls of different radii rolling together, with the contact point of one ball joining with the face to one side of the blended edge and the contact point of the other ball joining with the face to the other edge side. Figure 1-7 shows an example of a chamfer blend.

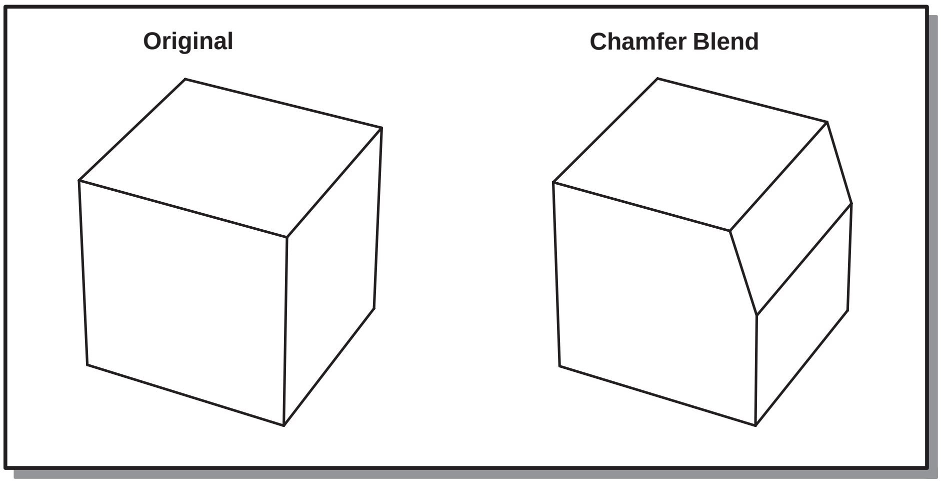


Figure 1-7. Chamfer Blend

This definition of a chamfer has the advantage that in the common case when the faces meet orthogonally in the blended edge, the radii are the same as the offsets as measured along each face. Another advantage is that chamfers and rounds behave in the same way when the angle between the faces meeting in the blended edge approaches pi: the width of the blend face shrinks to zero. One may contrast this definition with using a chamfer depth measured normal to the chamfer surface from the blended edge: the face becomes infinitely wide as the angle reaches pi. The surface types available for creating chamfers are limited to plane and cone surfaces in the Blending Component (a wider variety can be created using the rounded chamfer blend in the Advanced Blending Component).

Vertex Blends

Topic: *Blending

ACIS supports vertex blends and provides a variety of ways to control the shape of the vertex blend surface. Parameters to control the shape (e.g., setbacks, etc.) may be passed explicitly, or ACIS may be left to work them out automatically (by specifying “autosetback” or no parameters at all).

Setbacks

Topic: *Blending

When a blended edge ends in a blended vertex, the blend face made for the edge meets the blend face made for the vertex in a “cross” edge. ACIS needs to find a position and inclination for this cross edge. In such cases, start and end setbacks may be applied to each end of the blended edge. A *setback* specifies the distance along an edge back from the vertex at which the blend stops.

The setback defines a plane which is normal to the edge through a point on the edge and set back from the edge end by the given distance. The intersection of the setback plane with the blend surface defines a curve that may be used to bound part of the face for the blended vertex at the end of the edge.

The setback can be defined in one of the following ways:

- The setback can be specified at the time the implicit blend is put on each edge (it must be for the start or end of the edge, as appropriate).

  For a constant round, the cross curve will then be a circular arc. The distance of the setback is approximately the distance of the cross curve from the vertex at the end of the edge. The cross edge will not be inclined (i.e., it will not be oblique).
- Blend all the edges at the vertex, specifying blend size and shape, but no setbacks. Then, blend the vertex, at which time the setback is applied.

If the setback is to be applied on the vertex blend, it may be:

- Specified explicitly
- Calculated automatically (*autosetback*)
- Not specified at all

Setback Specified Explicitly

Start and end setbacks may be specified explicitly when the blend is set by passing the values as arguments to various blend APIs (or test harness commands or Scheme extensions). Valid values for setback depend on the size and geometry of the edges at the vertex being blended.

If the user specifies a setback that is smaller than the smallest setback practical, that value will be overridden and the smallest practical value used instead (refer to section *No Setback Specified* for information about how this minimum value is determined).

Oblique Setbacks

If the setback is specified explicitly, it may be inclined (oblique). This is a further refinement that lets the setback on the left spring curve of a blended edge differ from the setback on the right spring curve. This gives further control over the boundary shape of the blend face made for a vertex blend.

Oblique setbacks can be selected in one of two ways:

- In addition to the setback, the user can provide a signed value, which is the difference between the setbacks (left setback – right setback). This can be specified for the start and end of the blend.
- Oblique setbacks can also be selected without explicitly specifying the setback difference. Flag arguments in the APIs indicate whether or not the difference is specified. (The "oblique" option is used in the test harness.)

If the difference is not specified, when the left and right spring curves of the blended edge are intersected with the spring curves of the adjacent blended edges at the blended vertex, and when two intersection points are obtained (one for each spring curve), the cross edge is inclined so that it passes through both the intersection points. This gives the smallest and topologically simplest vertex blend face.

For a blended vertex of uniform convexity (its *n* edges are all convex or are all concave), the vertex blend face made will have *n* sides. If the blended vertex is not of uniform convexity, the blend face will have more than *n* sides; i.e., in addition to *n* cross edges, its will have one or more spring edges (edges at which the vertex blend surface springs from surfaces of faces adjacent to the blend vertex).

When a vertex is of mixed convexity or the sizes of the blends on its edges differ greatly, the oblique option can lead to poor shapes for the vertex blend boundary (angles at vertices of the boundary must be less than pi). In that case it is probably best to compute setbacks automatically.

Automatic Setback Calculation (Autosetback)

The setback can be automatically calculated, which means ACIS finds and assigns setbacks at a blended vertex (provided the edges at the vertex already carry implicit blend attributes). Autosetback is applied to a vertex blend using the api_set_vblend_auto function.

Users may want to use autosetback to get a good starting point, and then adjust the setbacks manually for further refinement of the blend shape.

Autosetback finds an average blend size for the edges at the vertex. For each edge, it considers the edge adjacent on the clockwise side and finds a setback that allows a spring curve of radius close to the average blend size, and similarly for the counterclockwise side. It then chooses the greater of the two setbacks for the current edge. This method regards the vertex as polyhedral; i.e., it looks at normals and tangents at the vertex.

Autosetback never makes oblique setbacks. However, it is possible to introduce oblique setbacks by manually refining the setback to improve vertex blend shape after the automatic setback calculation.

Because autosetback uses the sizes of the blends and the angles in which the edges meet at the vertex to estimate the setbacks, it works reasonably well if all of the following conditions are true:

- Blends on the edges are not too different in size (and not of zero size)
- Edges meet in angles not too different at the vertex (and not zero or pi)
- Edges are not too highly curved (calculations are made as if the edges were straight)

Note *A variant of autosetback exists for historical reasons. It is called "autoblend" (*api_set_vblend_autoblend*) and differs from autosetback in that it will use simple surfaces (spheres, etc.) whenever possible (autosetback does not), otherwise, it uses the same setbacks as autosetback.*

No Setback Specified

If no setback is specified, ACIS must still find the position of the cross edge. Given a blended edge with no setback specified, ACIS intersects the spring curves of the blend geometry found for the edge with the spring curves made for the adjacent blended edges at the vertex (i.e., the edges that are found by stepping to the next edge clockwise or counter–clockwise about the vertex).

Usually, two intersection points are found: one for the left and one for right spring curve. ACIS chooses the intersection point further from the blended vertex and positions the cross edge to pass through that point. If an adjacent edge meets the current edge in a reflex angle (greater than pi), no point will be found for that side, but a point must exist for the other side and that will be used to position the cross edge. In effect, this method finds the smallest setback practical for a non-oblique cross edge.

Some final adjustments are made in order to remove or improve any awkwardly-shaped curves, sometimes resulting in oblique cross curves.

As a rule of thumb, use no setbacks at all to obtain the smallest pleasing vertex blend surface, and use autosetback to obtain a larger extent for the blend surface. If no setbacks are specified, simple vertex blend surfaces will be made whenever possible (e.g., spheres, tori, etc.). Using autosetback precludes the creation of such simple surfaces, and an *n*-sided patch will always result.

Vertex Setback Example

Figure 1-8 illustrates a vertex blend with no setback, then the same blend with two different setbacks. The original block is 100 x 100 x 100. In the blend with no setback, oblique cross curves are used automatically to create a minimal blend. The first setback is 20, and the second setback is 40. With a setback of 40, the blend is larger. In this example, a setback of 200 is invalid, because it is larger than the length of the edges.

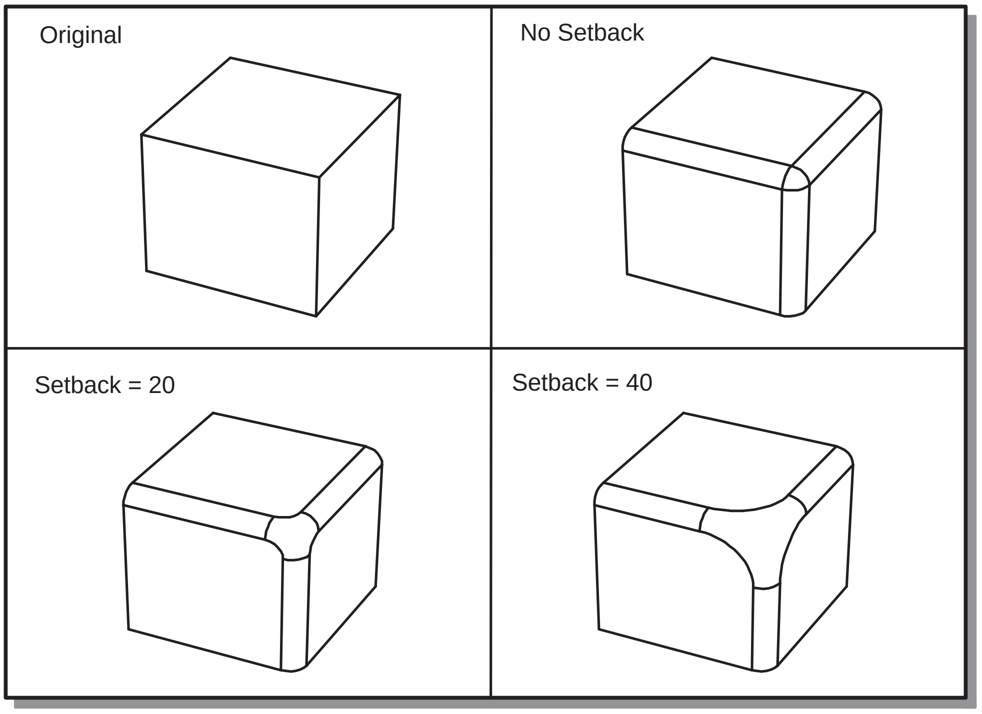


Figure 1-8. Vertex Setback

Bulge Factor

Topic: *Blending

The bulge factor is a number used to control the shape of the vertex blend. The bulge factor affects the interior shape of the vertex blend surface. It has no effect on the boundaries of the blend surface. It also affects only the *n*-sided patch surface, making no difference when spheres or tori (etc.) are used.

The bulge factor value ranges from 0.0 to 2.0. If the bulge factor is not set by the user, it is set to the default value, 1.0. A larger value increases the fullness of the blend. A value of zero is usually suitable for a blend at a mixed convexity vertex, but for a uniform convexity vertex (i.e., all of whose edges are either convex or concave), a positive bulge factor may improve the shape obtained. Figure 1-9 illustrates a vertex blend with no bulge (blend is flatter) and then the same blend with bulge = 2.0, which produces a fuller blend. The default value of 1.0 should normally be acceptable.

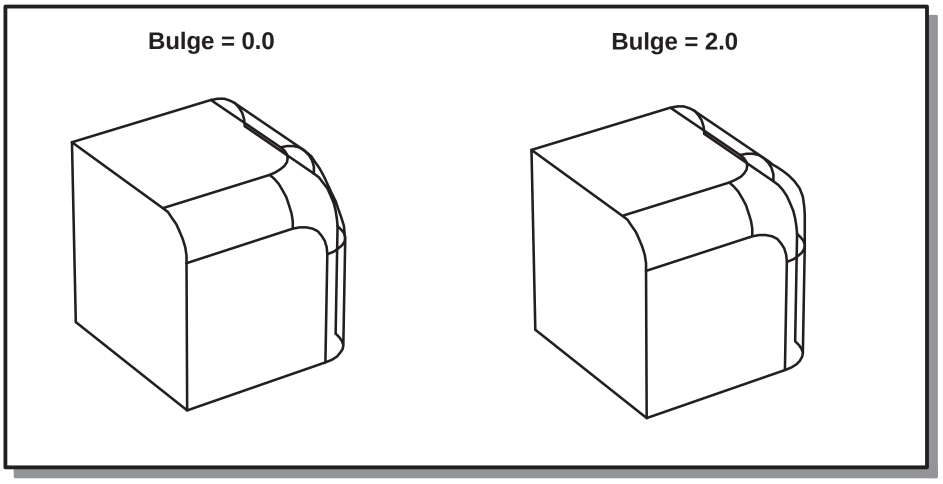


Figure 1-9. Bulge

Vertex N–Sided Gregory Polygon Surface

Topic: *Blending

A *vertex* n-*sided Gregory polygon surface* is the case in which all of the edges meeting at an internal vertex are to be blended. They do not have to have identical blend radii, nor do they have any “smoothness” requirements.

An *n*-sided Gregory polygon surface is typically produced when a sphere, torus, or rolling ball blend will not fit into the vertex region (for example, if all the blends that meet there are of different radii), or when setbacks are given (as in Figure 1-10).

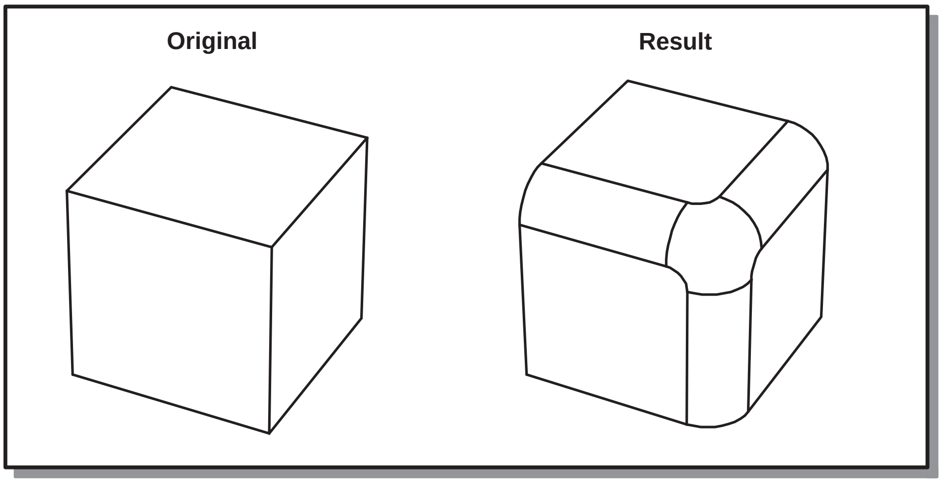


Figure 1-10. N-Sided Gregory Polygon Surface

Other cases of internal vertices that are not explicitly blended with an *n*-sided Gregory polygon surface are handled by other techniques. The vertex may be smoothed away by blending one of the edges at the vertex, leaving a smooth sequence of edges that are then blended in a second step. Another option is to simultaneously blend a pair of edges at the vertex to form a mitered joint.

Approximated Vertex Blend Surface

Topic: *Blending

The API function api_make_VBL_output_surfaces approximates an *n*–sided VBL_SURF surface by *n* four–sided bs3_surfaces. This enables ACIS to output vertex blend surfaces in a form that can be read and used by most other systems.

Blend Topology

Topic: *Blending

The ACIS blending algorithm works on a network of edges and vertices. The edges and vertices presented for blending should not form disjoint networks. Each of the edges and vertices has an associated blend radius or chamfer distance. The radii in the network do not have to be the same. Blends are created, evaluated, and attached with a high degree of precision. Wherever possible, analytic surfaces are used.

Edges are characterized by the continuity between the two faces meeting at the edge and by the convexity of the faces along the edge. They are *smooth* if their two faces meet with tangent plane (G1) continuity; else, they are *nonsmooth*. Blending operations may fail on non-G1 surfaces. Edges are further characterized as being *convex* or *concave*. Edges must have the same continuity and convexity along their entire domain of definition.

Vertices are characterized by the number of blended edges meeting at the vertex. A vertex is a *terminal vertex* if exactly one blended edge ends at the vertex. Figure 1-11 shows terminal vertices.

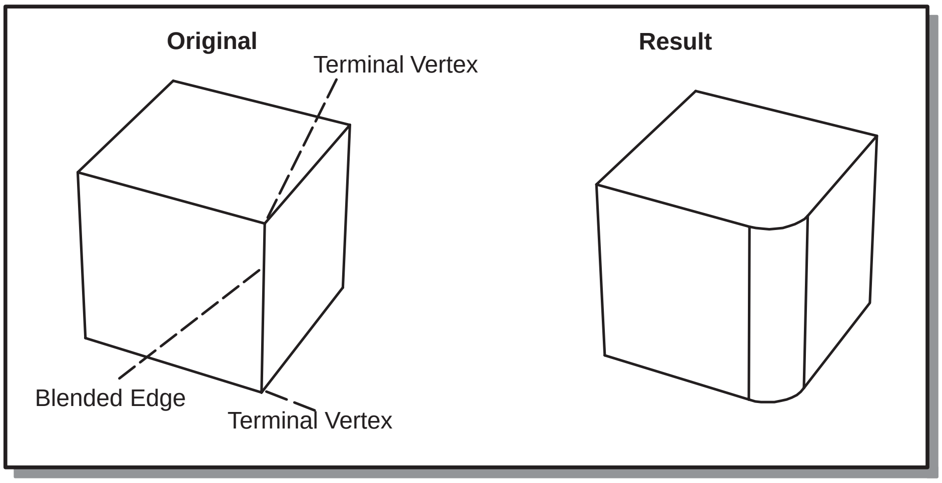


Figure 1-11. Terminal Vertex

A vertex is an *internal vertex* if two or more edges coming into the vertex are blended. Figure 1-12 shows internal vertices. In this example, blending one edge at the vertex creates a smooth connection between the other two edges, which are then blended as a smooth sequence.

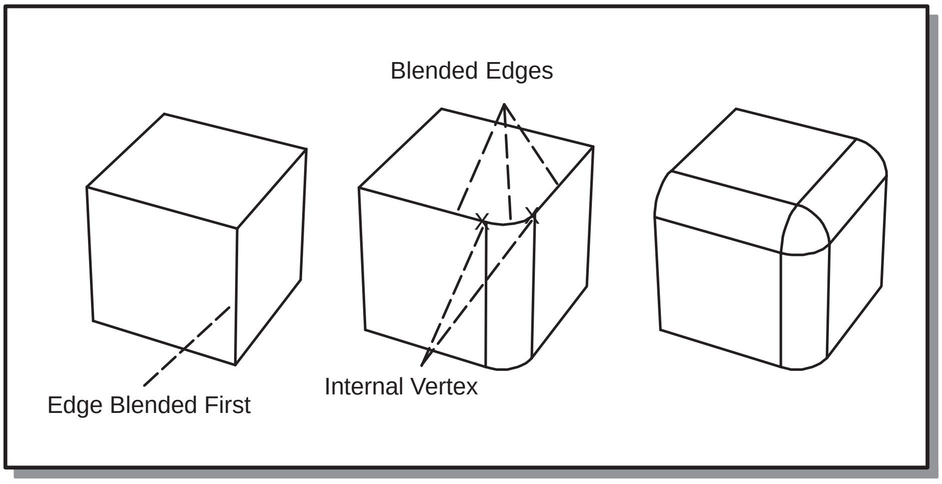


Figure 1-12. Internal Vertex

Capping

Topic: *Blending

A blend terminates when the figurative rolling ball does not roll onto any of the new entities it encounters, but attempts to trim the blend face to the body. Normally, this involves intersecting the blend surface with existing body faces, or possibly the extension of existing body faces. This process is referred to as *capping*. When a cap terminates a blend, it is referred to as an *end cap*.

Capping may not always terminate the blend. The cap may not cover the end of the face, but merely a side of it, with the original blend face continuing afterwards. Such a cap is called a *side cap*.

Special Cases

Topic: Blending

The capping algorithm is robust and can handle a large number of cases. However, for a few simple blend cases, the capping algorithm is unnecessarily complex and therefore slower than necessary. With performance in mind, a special-case capping algorithm is used to handle the following two cases differently:

- The most simple mixed convexity capping case (refer to Figure 1-13)
- The mixed convexity case with the capping intercept *not* on a body edge (refer to Figure 1-14)

The special-case algorithm first identifies these two cases and then calculates the capping surface and the edges that are to be used to close off the blend. The capping face topology is then built and appended to the blend sheet. Attributes that aid the Booleans in blend stage two (by avoiding the recalculation of the intersections) are then added to this sheet body. On average, this algorithm results in a two-fold (2X) time improvement—though only over the time spent in capping.

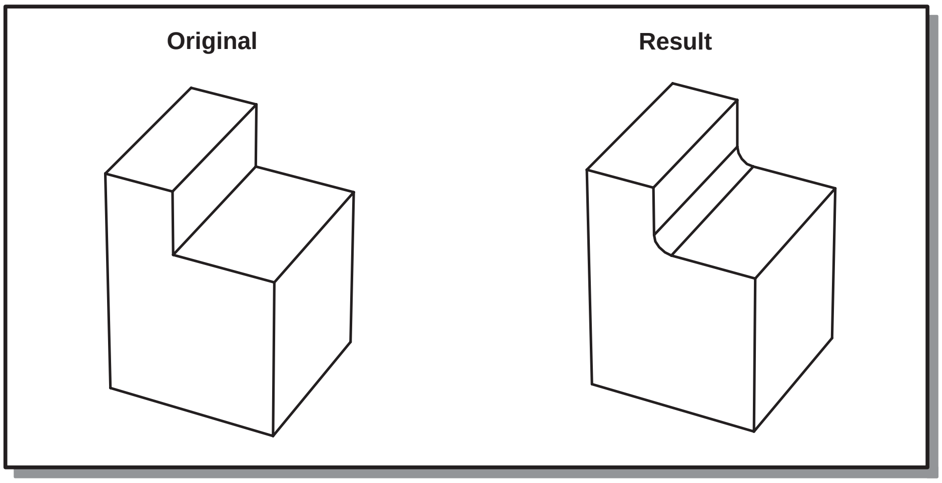


Figure 1-13. Simple Mixed Convexity

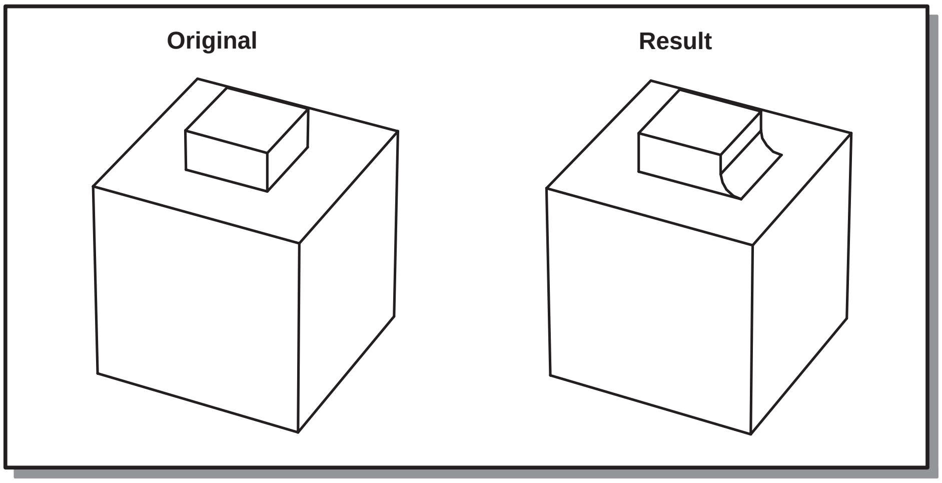


Figure 1-14. Capping Intercept Not on Edge

Edge–Face Blend Instead of Side Cap

Topic: Blending

Standard blending can sometimes use edge-face blends as an alternative to creating blend side caps. The preference for the edge-face blend alternative can be specified by supplying the extra final argument to api_set_const_rounds as bl_how_roll_on. The default value for this argument prefers side caps, thereby maintaining the behavior of previous versions of ACIS.

Figures 1-15 and 1-16 illustrate the difference between a blend with side caps and a blend in which edge-face blends are used. In Figure 1-15, the blend has side caps. In Figure 1-16, edge-face blends are used instead of side caps, and the straight edges that bound the planar face on which the blend rests are preserved.

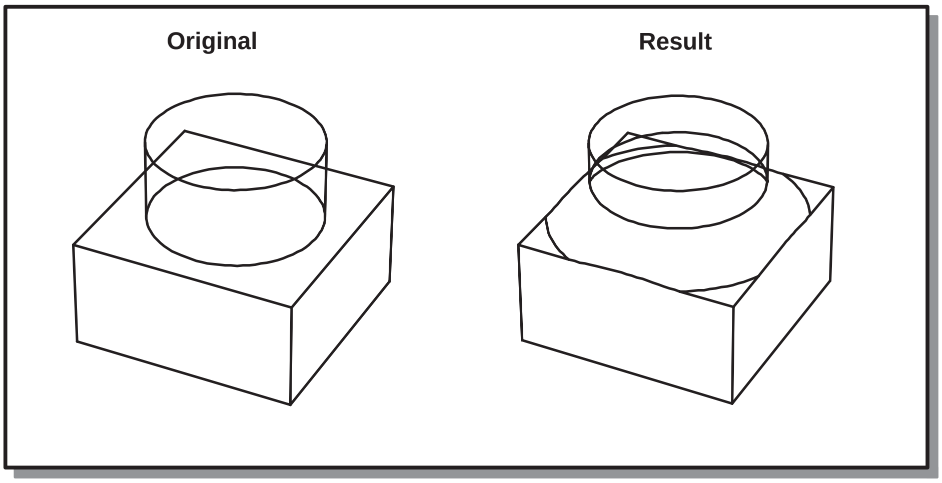


Figure 1-15. Side Caps

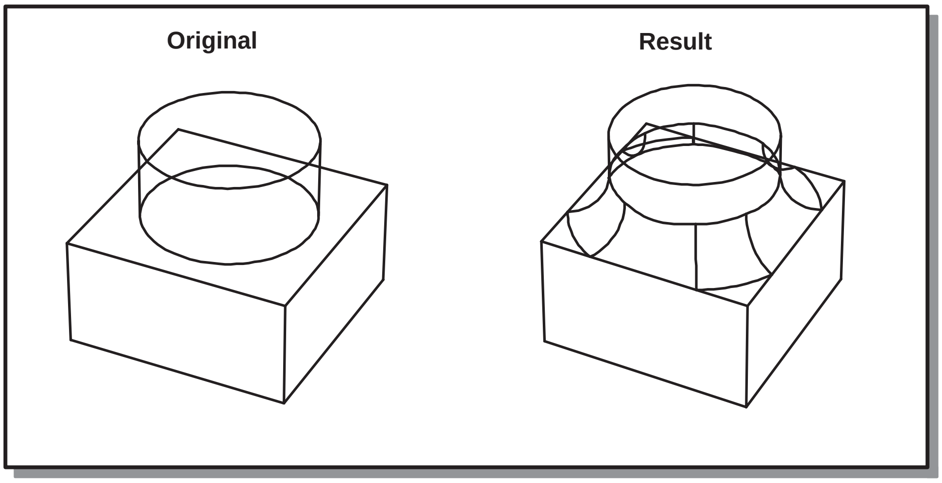


Figure 1-16. Edge–Face Blends Instead of Side Caps

If the algorithms determine that the edge-face alternative may not work, a side cap is used, regardless of the setting of the extra argument to api_set_const_rounds. For example, if the sequence of edges where the side cap would normally occur contains nontangent meeting edges, the side cap is used anyway. If an edge-face blend is used, and the other side (where the blend still rests on a face) requires a side cap, the blending algorithms consider using an edge-edge blend. This feature currently has limited capability.

In Figure 1-17, edge-face blends are used wherever possible, and a side cap is produced where edge-face blends were not.

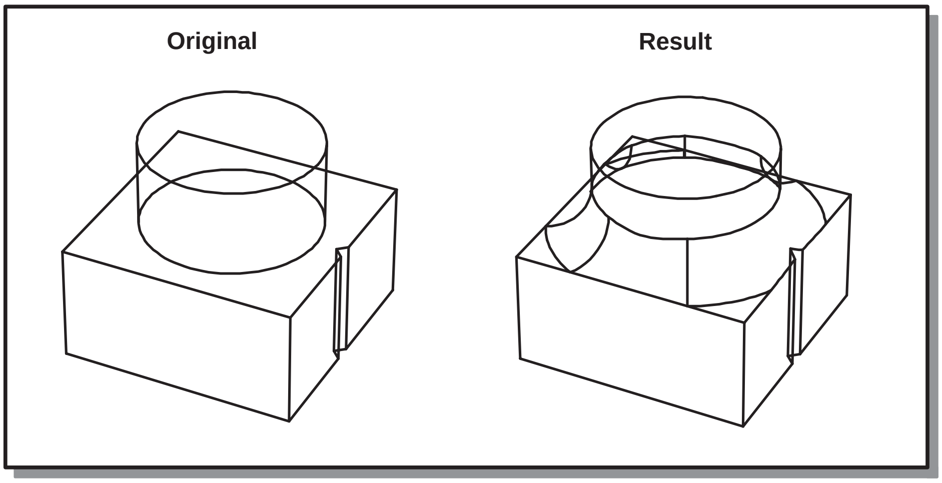


Figure 1-17. Edge–Face Blends and Side Cap

Sequences of Blended Edges

Topic: *Blending

A blend sequence is a series of connected edges that are to be blended in a single operation. The vertices where two blended edges meet are categorized as follows:

Smooth	Two blended edges meet tangentially at a vertex, and any number of nonblended edges may meet as long as they are smooth edges. A vertex with this configuration is called a *bi-blend.*
Nonsmooth . . .	Blended edges do not meet tangentially, or there are nonsmooth nonblended edges at the vertex. This results in a *mitered blend*.
Closed/Open . .	Either type of sequence may be open or closed. A closed sequence consists of a closed loop of blended edges. In a closed sequence, there is no need to find end capping faces for the blend surfaces (side caps can still occur).

Bi–Blend

Topic: Blending

A bi-blend is an internal vertex at which two edges meet tangentially and have compatible blend attributes. Any number of smooth, nonblended edges may meet at the vertex. At a bi-blend vertex, the blend surfaces from the two blended edges mate exactly to form a smooth continuous blend across the vertex.

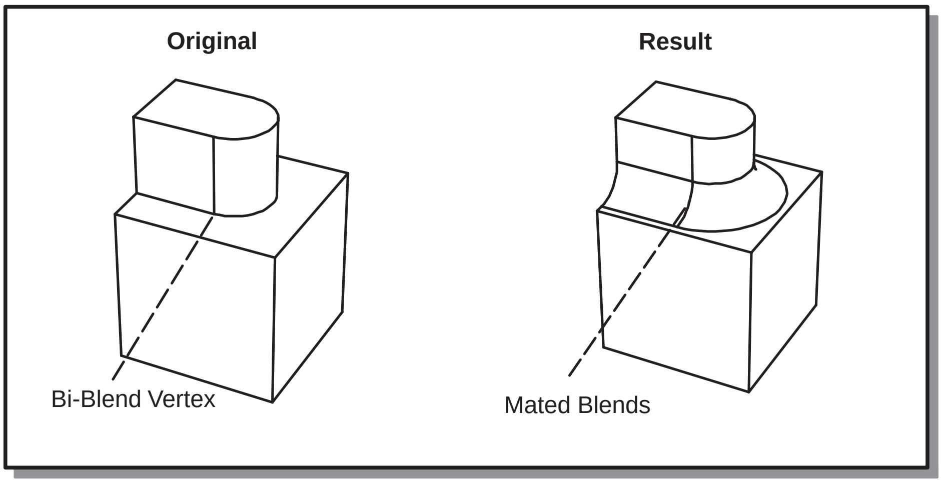


Figure 1-18. Smooth Sequence Bi-Blend

Mitered Blends

Topic: Blending

A mitered blend is the case in which the vertex is an internal vertex at which two edges do not meet each other tangentially or there are nonsmooth nonblended edges present at the vertex.

A miter surface is used to calculate the cross curves when blending two edges that do not meet tangentially. In the special case of constant rounded blends on coplanar straight or circular edges, a curved miter surface is constructed instead of a plane.

The two blended edges need not have the same blend attributes, but they must have the same convexity. At most, two other edges may be incident at the vertex, and they must be nonsmooth. If two other edges are present, they must be separated by the two edges being blended. Figure 1-19 shows a concave mitered blend.

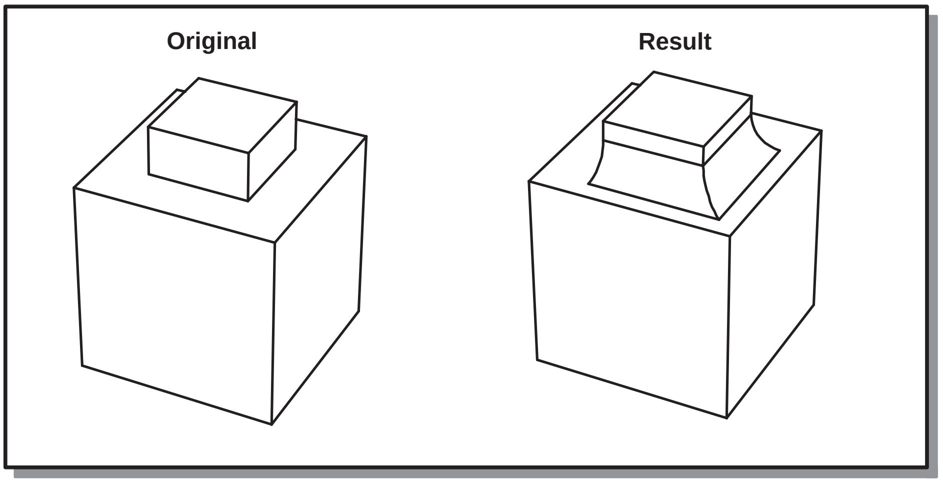


Figure 1-19. Concave Mitered Blend

Complex Miter

Topic: Blending

If two blend sheets do not meet smoothly and do not match at their ends, this condition requires special processing. One of the side surfaces must be extended as a partial end cap. This may happen if the two blends have different radii. Figure 1-20 shows a concave, complex miter.

Complex miters might also be required if a common side surface is not perpendicular to the intervening edge. This condition also requires that one of the side surfaces be extended as a partial end cap. Figure 1-21 shows such a mitered blend. The blends have the same radii, but the miter is complex, because the two upper spring curves intersect the vertical edge at different points.

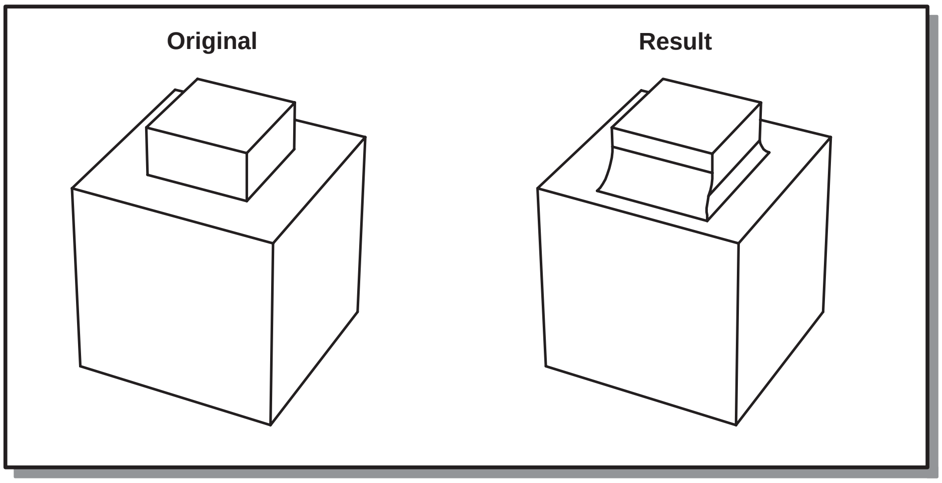


Figure 1-20. Concave Complex Miter

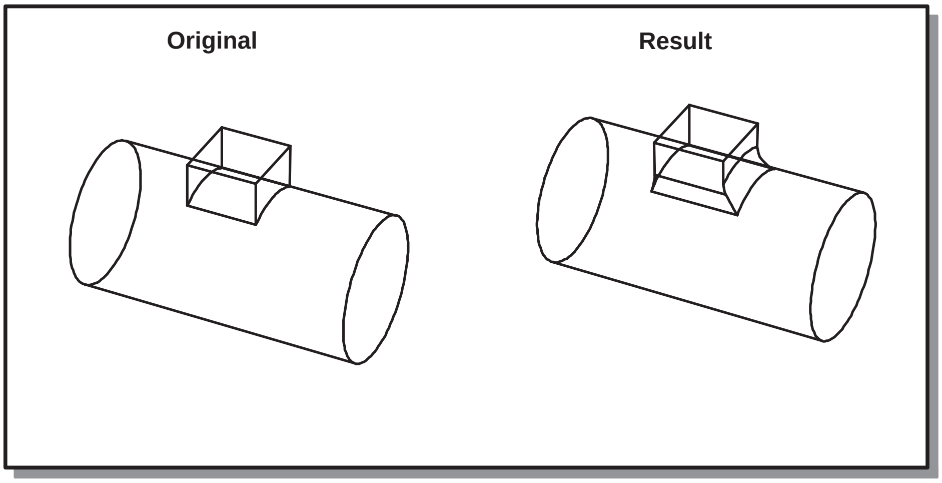


Figure 1-21. Complex Mitered Blend

Closed Sequences

Topic: Blending

A closed sequence of blended edges and bi-blend vertices has no open ends and no end caps. Internal vertices in the sequence may be smooth (a bi-blend) or nonsmooth (a mitered joint). Figure 1-22 shows a closed sequence with both bi-blends and miters. The term *periodic sequence* is sometimes used to refer to a closed sequence with no miters or vertex blends.

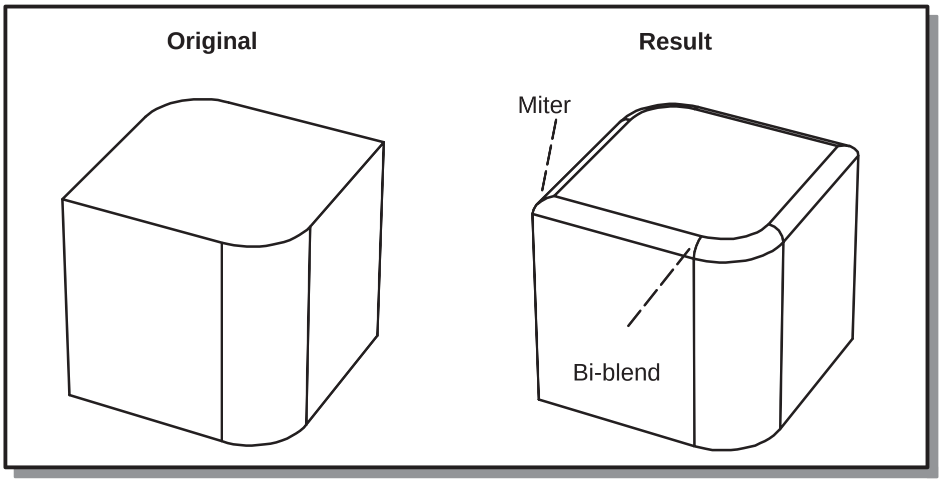


Figure 1-22. Closed Sequence

Open Ends and End Caps

Topic: *Blending

At the end of a blend sequence (an “open end” of the sequence), the blend sheet must be fitted exactly to the original model. When possible, a face (or faces) in the model will be intersected with the blend sheet to trim the sheet exactly to the model. When no model faces intersect the blend sheet, ACIS will attempt to extend existing model faces to close off the end of the blend. Figure 1-23 shows an example of such an end cap.

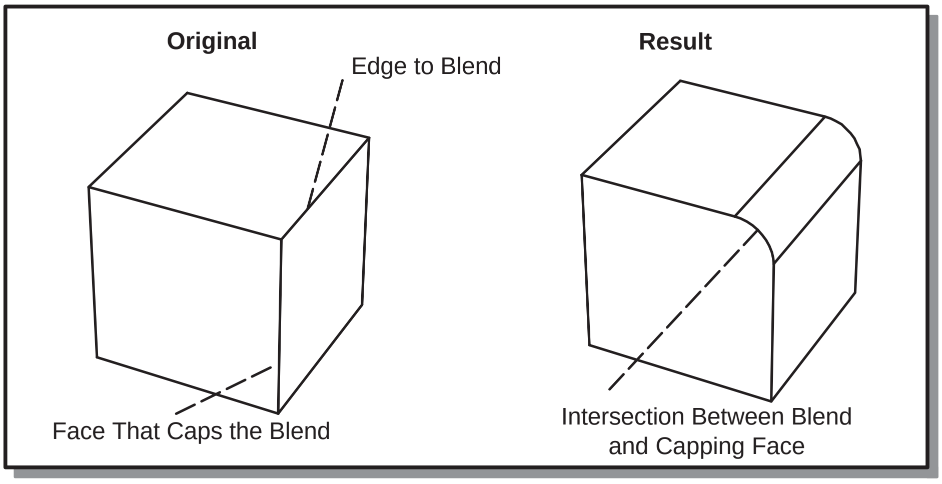


Figure 1-23. Identifying End Caps

Multiple Capping Faces

Topic: Blending

The blend sheet may intersect multiple capping faces in the body. Figure 1-24 shows a blend sheet that is intersected with two of the faces in order to trim the sheet to the correct size.

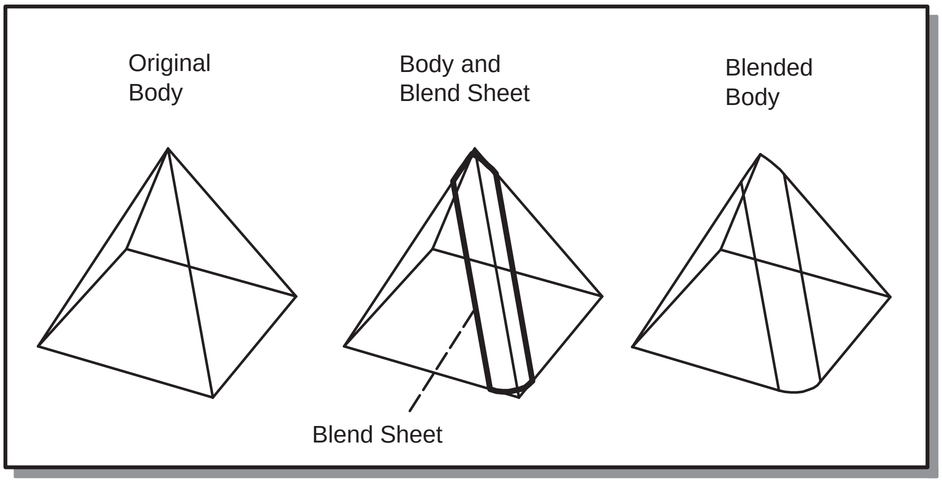


Figure 1-24. Multiple Capping Faces

Extending Capping Faces

Topic: Blending

If a blend has an open end at a mixed convexity vertex, a face in the body may need to be extended to create the blend's end cap. A *mixed convexity vertex* is a vertex at which the blended edge meets other edges whose convexity is not the same as the blended edge's. If the capping faces are spline surfaces, then an extension of the capping face is created so that the blend surface is guaranteed to intersect with the capping surface.

In Figure 1-25, two faces, *A* and *B*, are extended in order to close off the end of the blend.

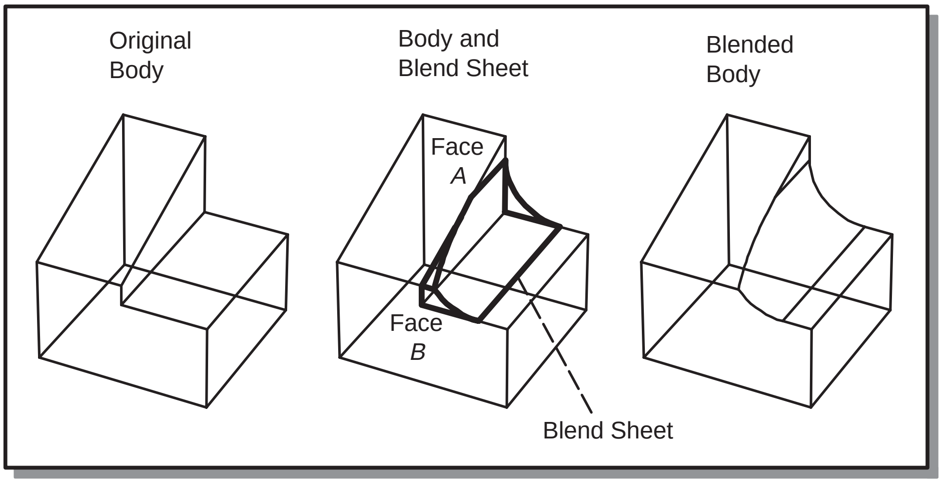


Figure 1-25. Extended Capping Faces

Remote Blending

Topic: *Blending

A *remote blend* is a blend that is not traveling on the two faces immediately adjacent to the blended edges. ACIS has various facilities for handling remote blends.

Only Some Blends Are Remote

Topic: *Blending

ACIS can compute remote face-face blends or edge-face blends when they are adjacent to blends that are not remote. Figure 1-26 demonstrates a blend that has traveled onto a cylindrical pre-existing blend face, which is a remote face.

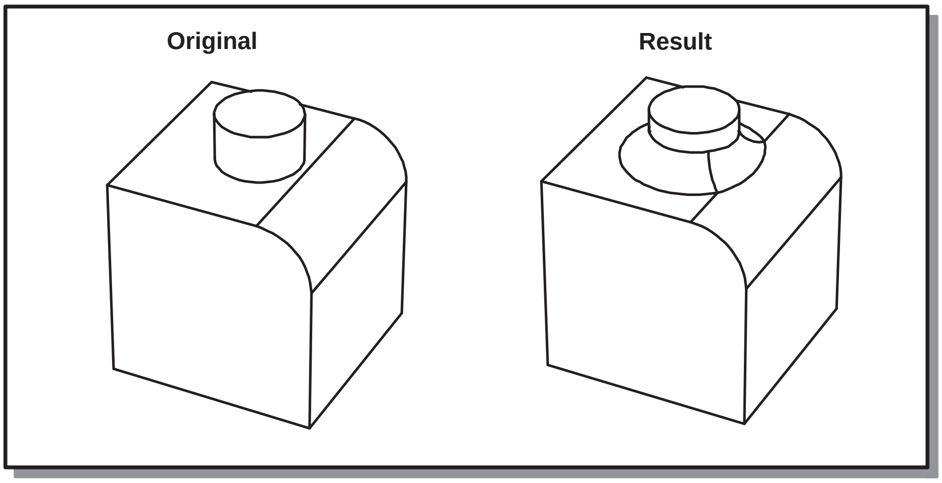


Figure 1-26. Remote Face Blend

Remote edge-face blends may also sometimes be used in a similar situation (refer to section *Edge–Face Blend Instead of Side Cap*, Figures 1-16 and 1-17).

All Blends Are Remote

Topic: *Blending

When the blends in a sequence are all remote, then blending must determine an alternative pair of entities against which to start blending. These may result in face-face, edge-face, or even edge-edge blends. This facility is currently restricted to blend networks without vertex blends or miters.

Without remote blending, the blend shown in Figure 1-27, would fail if the blend radius was greater than the height of the step. With remote blending, an edge-face blend is used instead.

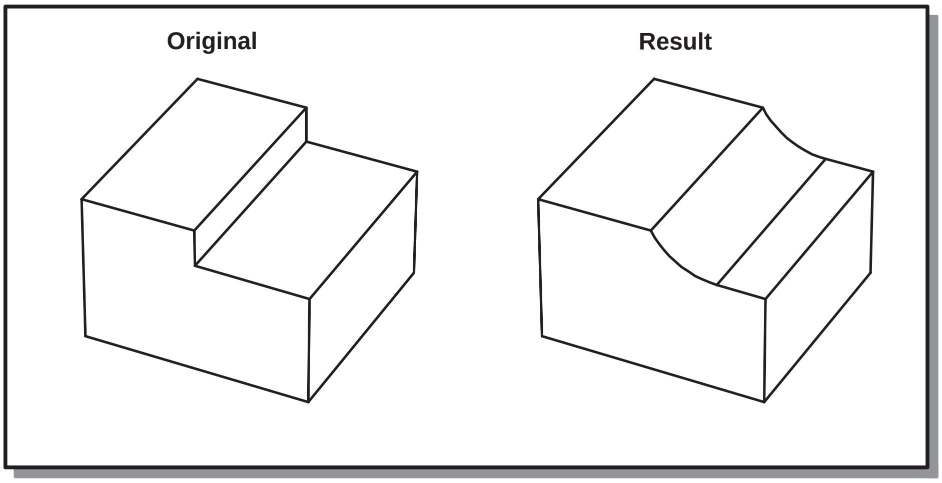


Figure 1-27. Edge–Face Blending

In Figure 1-28, a sequence of 3 edge-face blends is created, as the blend radius is greater than the height of the small protrusion.

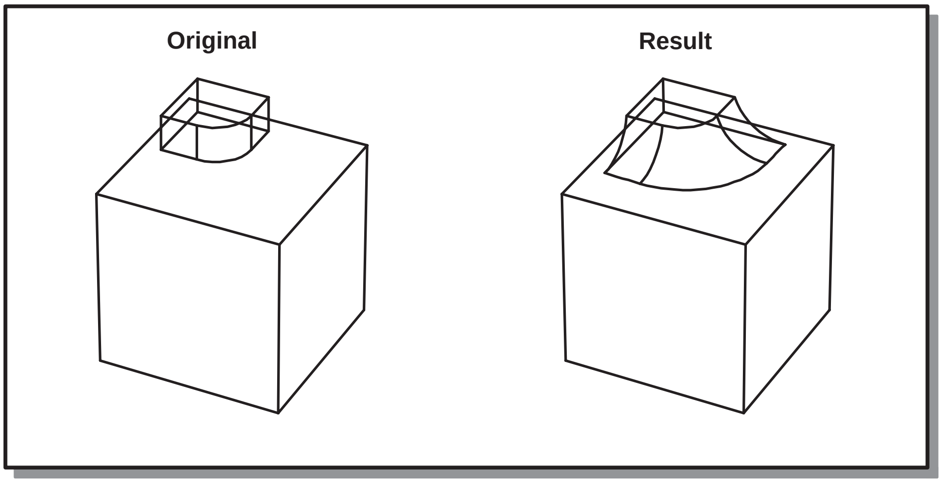


Figure 1-28. Remote Edge–Face Blending

In Figure 1-29, a face-face blend will not fit, so the blending algorithm switches to a remote edge-edge blend.

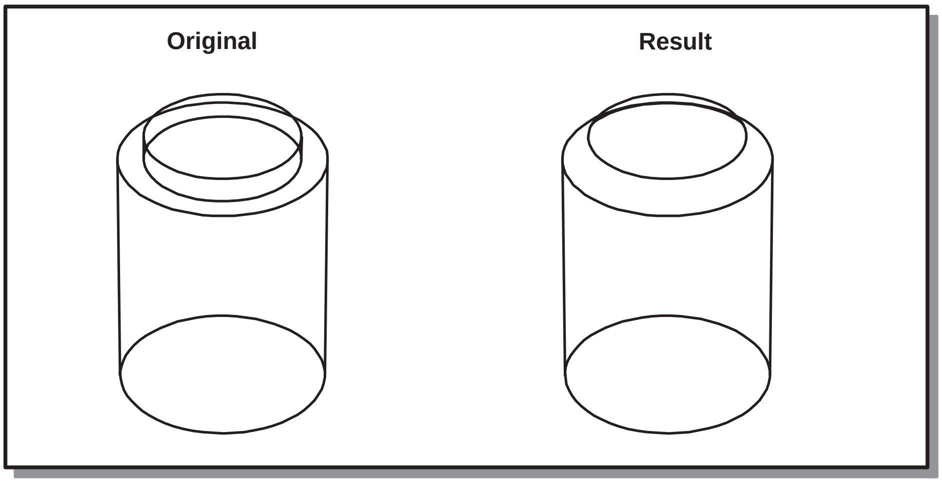


Figure 1-29. Remote Edge–Edge Blend

Vertex Blend Order

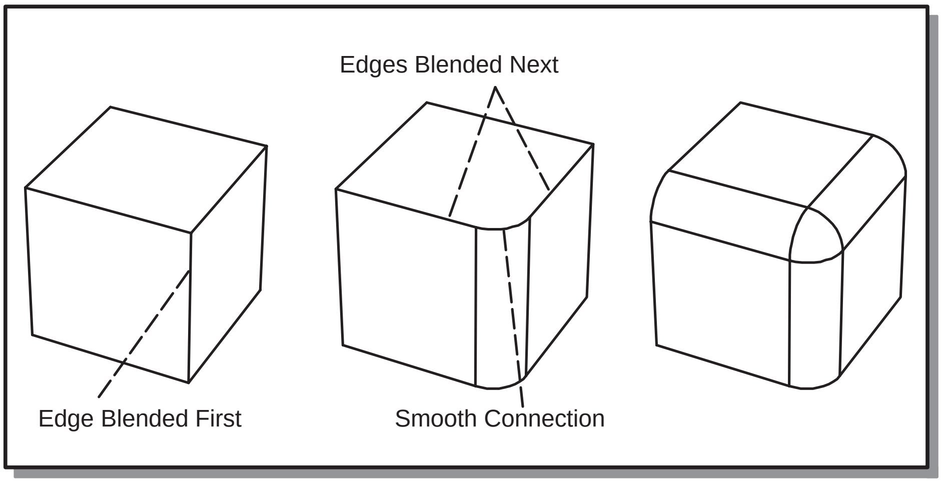
Topic: *Blending

ACIS provides a variety of ways to blend vertices. The order in which the edges incident on a vertex are blended controls the result.

Smooth Continuity Blend

Topic: Blending

First, blend one edge at the vertex. This creates a smooth connection between the other two, that are then blended at the same time as a smooth sequence. Figure 1-30 shows two examples of smooth continuity blending.



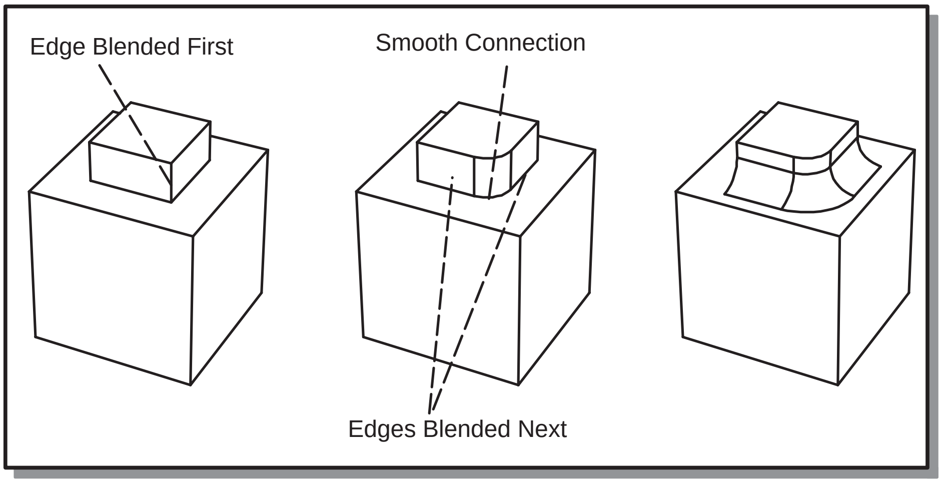


Figure 1-30. Smooth Continuity Blends

Mitered Blend

Topic: Blending

Miter two edges first, which creates a smooth path for blending the third edge. Decrease the radius of the third blend to roll through the *miter arc* (the arc that is produced by intersecting, or mitering, the two blend surfaces). Figure 1-31 illustrates this.

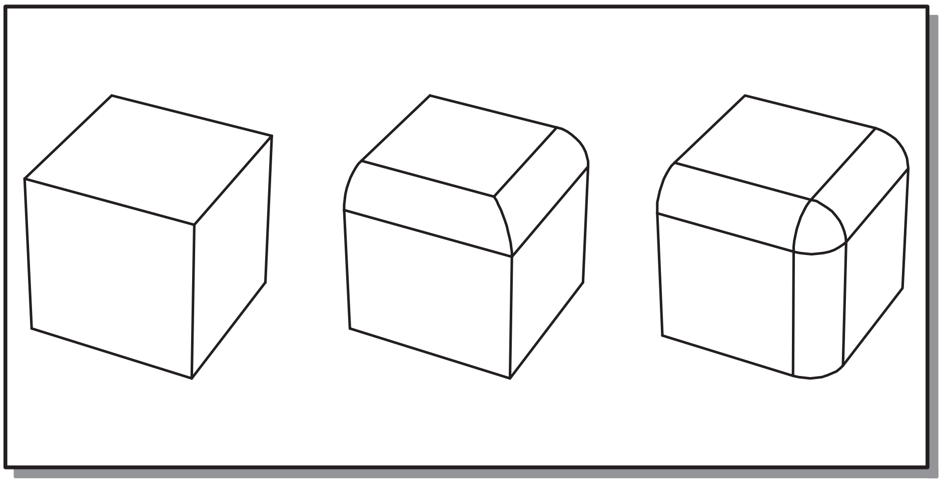


Figure 1-31. Mitered Blends

Mixed–Convexity Mitered Blend

Topic: Blending

A mixed–convexity mitered blend is a blend on multiple edges that do not all have the same convexity, which meet at a common vertex, and which are intended to be mitered. Figure 1-32 shows an example of a mixed-convexity mitered blend in which the two concave edges are mitered first to create a smooth path for blending the third, convex, edge. In this example, there is no problem with a ball of the same radius rolling across the outside of the miter arc.

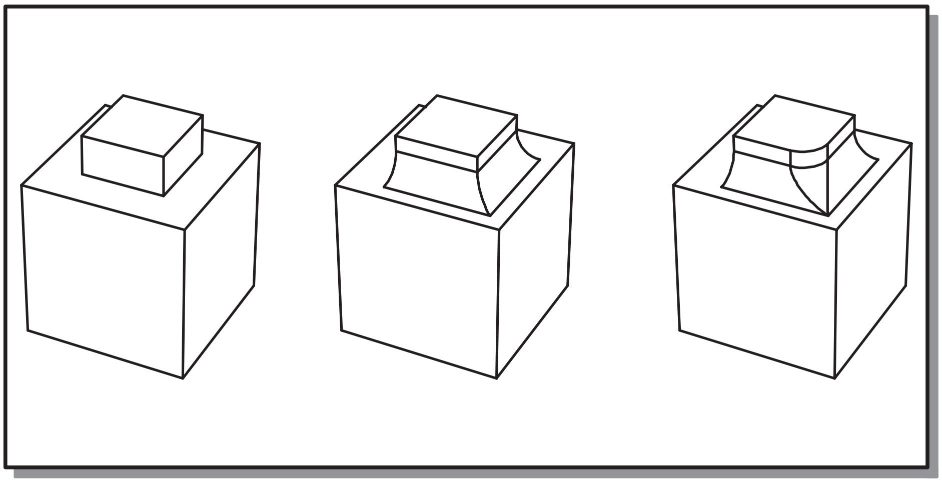


Figure 1-32. Mixed-Convexity Mitered Blend

Mixed–convexity mitered blends may not produce a unique valid solution. Therefore, ACIS disallows a single blend to be fixed with mixed–convexity mitered blends. The user must blend the edges separately to specify which solution is desired. In other words, the user must create the blend in multiple steps, blending the edges in the desired order.

Figures 1-33 and 1-34 illustrate that blending edges in different orders can produce different valid solutions. Each figure shows the original body (a small block sitting on a larger block), the result after the first edge is blended, and the final blend solution.

In Figure 1-33, the blend is done by starting with a blend on the concave horizontal edge and then blending the convex vertical edge.

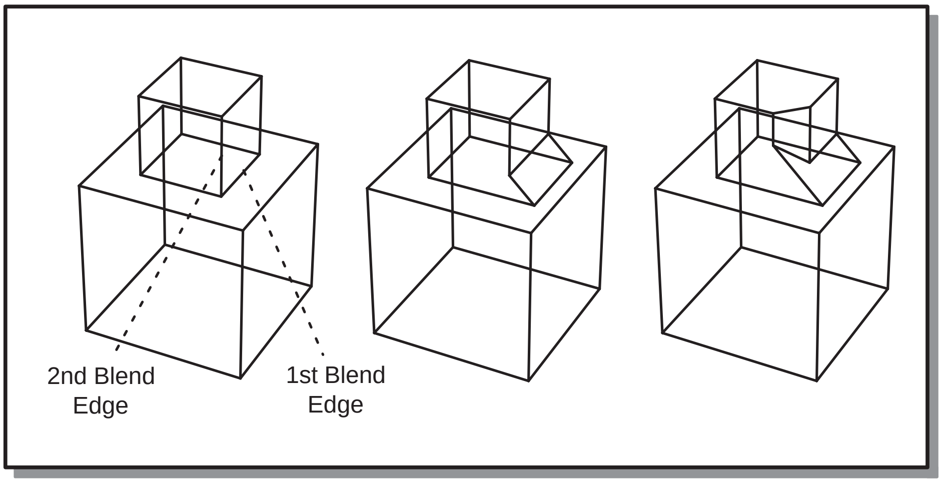


Figure 1-33.   First Solution: Blend Concave Edge First

In figure 1-34, the blend is done by starting with a blend on the convex vertical edge and then blending the concave horizontal edge.

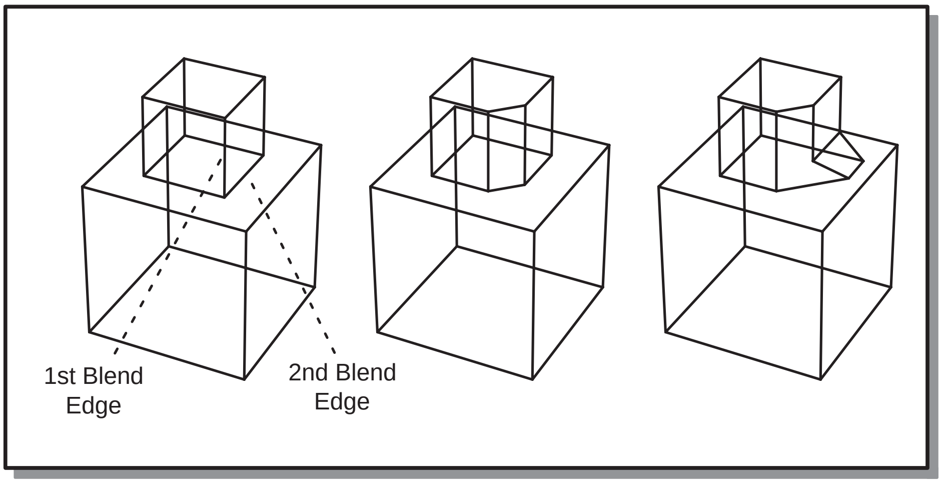


Figure 1-34.   Second Solution: Blend Convex Edge First

Single Blend

Topic: Blending

If an edge joins smoothly with any other edges, any blend that is applied to the edge will also automatically be applied to every edge in the smooth sequence. This behavior may be overridden by specifying a *single* blend. A blend surface will then be calculated for the single edge, and be extended, intersected and capped with nearby faces.

As an example, first miter a pair of edges, then blend the third edge as a single blend to force it to intersect with the existing miter, instead of following the miter arc. Figure 1-35 shows an example of a single blend on a miter.

Single blends are not always geometrically possible. This is illustrated in the examples of Figures 1-33 and 1-34, if round blends are used in place of chamfers.

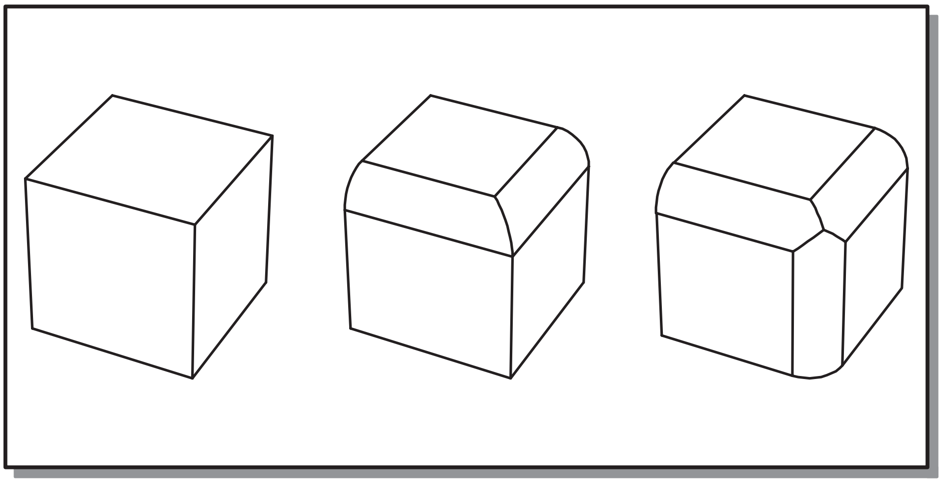


Figure 1-35. Single Blend

Simple Vertex Blend

Topic: Blending

Figure 1-36 shows an example of a simple vertex blend. The three edges and the vertex are blended simultaneously with the same radius. A setback is used on this vertex blend.

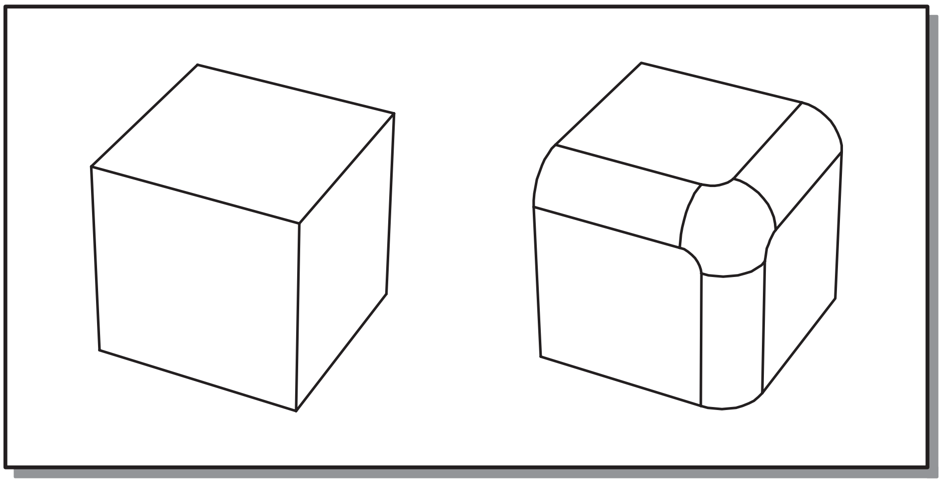


Figure 1-36. Vertex Blend with Setback

Figure 1-37 also shows a vertex blend, but the edges being blended are of mixed convexity. The three edges and the vertex are blended simultaneously with the same radius. A setback is used on the vertex blend.

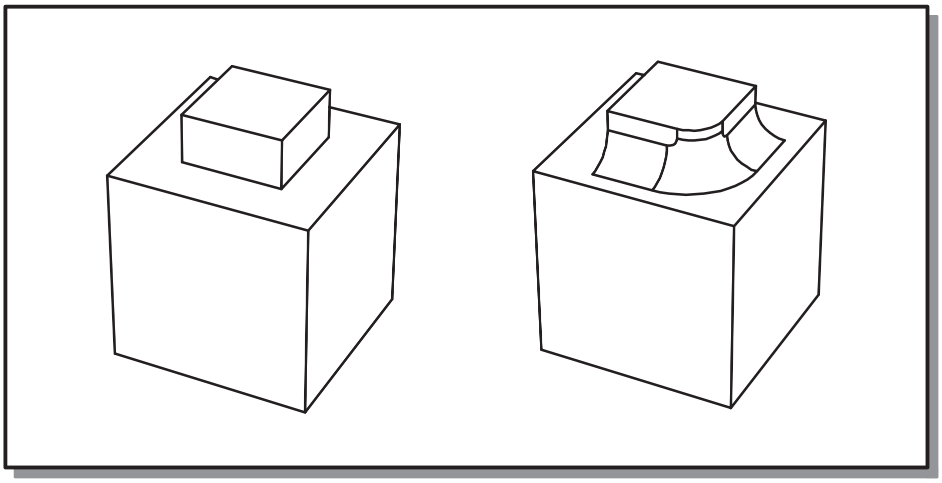


Figure 1-37. Mixed-Convexity Vertex Blend

Blend Reordering

Topic: *Blending

The blend reordering algorithm enables the blending of edges between two existing blend faces, when the radii of the existing blends is smaller than that of the new blend. The new blend surfaces are made up as though the larger radius blend had been done before the smaller radius blends. The blend reordering algorithm can blend edges between two blends of different radii, whether both—or just one—of these blends has a smaller radius than the new blend.

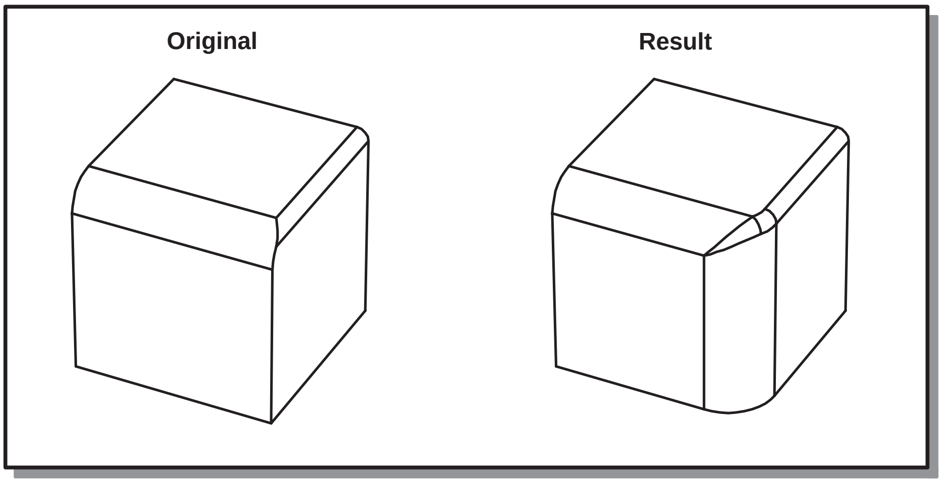


Figure 1-38. Blend Reorder

Local and Global Interference Checking

Topic: *Blending

A critical portion of the blend algorithm is to detect the interference of the blend sheet with the geometry of the part being blended. Usually, interfering geometry can be detected by intersecting the spring curves of the sheet with geometry local to the blend. Occasionally, the surfaces of the sheet must be intersected with all of the geometry of the body to detect such global interferences.

The option bl_remote_ints directs the blending algorithm to perform global or local interference checking. This option is off by default, indicating that the algorithm is to perform local interference checking only. This speeds up blending dramatically in some cases (especially thin-walled parts). Although not common, cases do sometimes arise which require global interference checking. If desired, option bl_remote_ints can be turned on, indicating that global interference checking is to be performed.

This section provides examples that illustrate some unusual cases that require global interference checking.

Vertex Blend with Univex Side Capping

Vertex blends that need univex side capping need global interference checking.

Scheme Example

```
; The example would fail if this option is not set
(option:set ”bl_remote_ints” #t)
; #f
(define cube (solid:block
    (position –30 –30 –30)
    (position 30 30 30)))
; cube
(define wb (wire–body:points
    (list (position 30 –30 30)
    (position 30 –10 30)
    (position 30 20 10)
    (position 30 –30 10)
    (position 30 –30 30))))
; wb
(define sht (sheet:cover–wires wb))
; sht
(sweep:law sht 60)
; #[entity 16 1]
(solid:subtract cube sht)
; #[entity 15 1]
;; OUTPUT Original
```

```
(define edge1 (pick:edge
    (ray (position 0 0 10)
    (gvector 1 0 0))))
; edge1
(blend:entities edge1
    ”round” 13
    ”setback–start” 2
    ”setback–end” 2)
; #[entity 17 1]
(define edge2 (pick:edge
    (ray (position 30 29 29)
    (gvector 0 –1 0))))
; edge2
(blend:entities edge2
    ”round” 7
    ”setback–start” 5
    ”setback–end” 5)
; #[entity 18 1]
(define edge3 (pick:edge
    (ray (position 0 0 10)
    (gvector 0 1 0))))
; edge3
(blend:entities edge3
    ”round” 2
    ”setback–start” 3
    ”setback–end” 3)
; #[entity 19 1]
(define vertex1 (pick:vertex
    (ray (position 0 20 10)
    (gvector 1 0 0))))
; vertex1
(blend:entities vertex1 ”setback” 3 ”fix”)
; (#[entity 21 1] #[entity 22 1] #[entity 23 1]
; #[entity 24 1] #[entity 25 1] #[entity 26 1]
; #[entity 27 1] #[entity 28 1])
;; OUTPUT Result
```

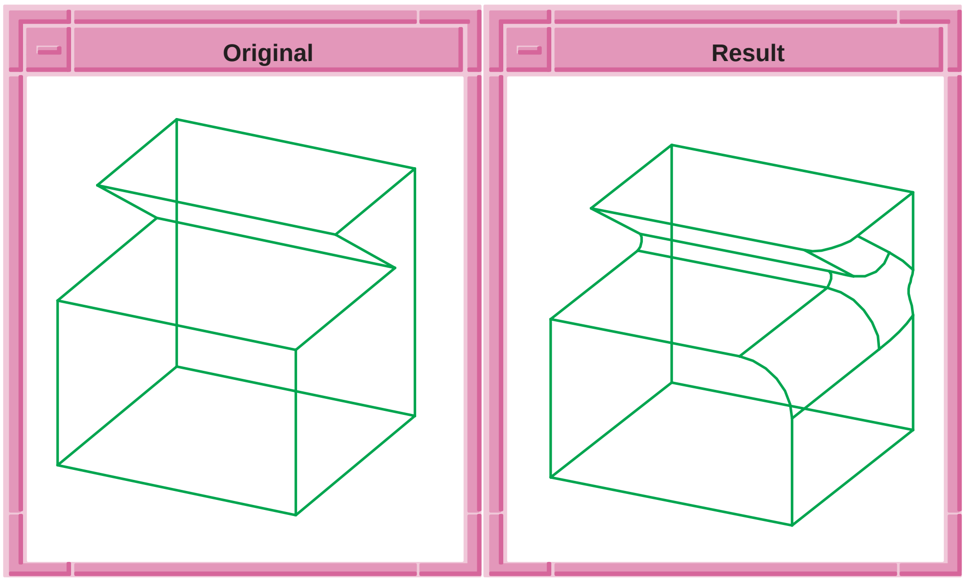


Figure 1-39. Vertex Blend with Univex Side Capping

Convex Block with Hole

Some blends really need global intersections, such as a convex block with a hole.

Scheme Example

```
(option:set ”bl_remote_ints” #t)
; #f
(define cube (solid:block
    (position –30 –30 –30)
    (position 30 30 30)))
; cube
(define cyl (solid:cylinder
    (position 0 0 –30)
    (position 0 0 30)
    5))
; cyl
(entity:move cyl 20 0 0)
; #[entity 3 1]
(solid:subtract cube cyl)
; #[entity 2 1]
;; OUTPUT Original
(define edge1 (pick:edge
    (ray (position 30 0 0) (gvector 0 0 1))))
; edge1
(blend:entities edge1 ”round” 20 ”fix”)
; (#[entity 5 1] #[entity 6 1]
; #[entity 7 1] #[entity 8 1])
;; OUTPUT Result
```

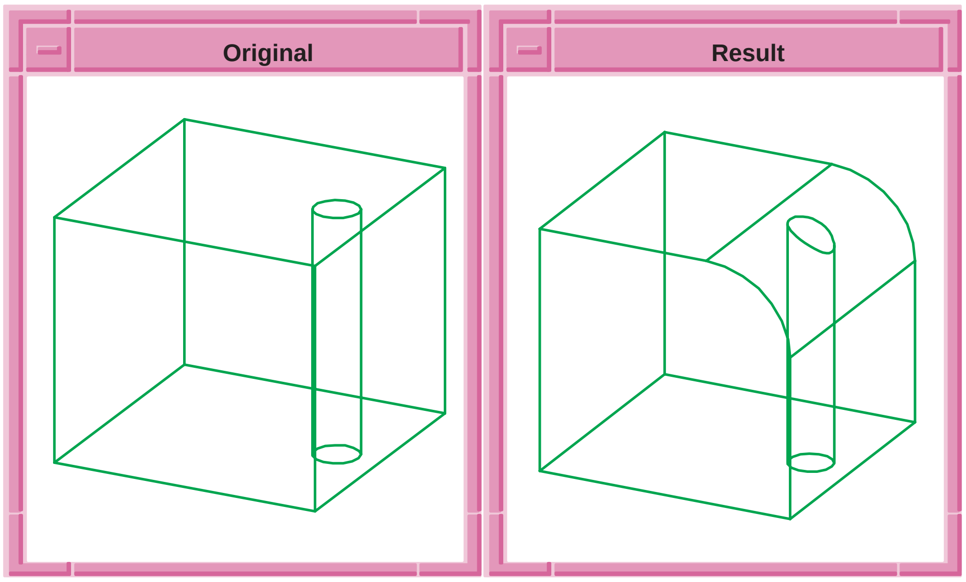


Figure 1-40. Convex Block with Hole

Pocket Loss with Checking

Occasionally, a blend may work without global interference checking, but the result could be slightly different than expected. In this example the cylindrical pocket gets lost if option bl_remote_ints is off, However, if it is on, the blend works as desired.

Scheme Example

```
(option:set ”bl_remote_ints” #t)
; #f
(define cube (solid:block
    (position –30 –30 –30)
    (position 30 30 30)))
; cube
(define cyl (solid:cylinder
    (position 0 0 –10)
    (position 0 0 10)
    5))
; cyl
(entity:move cyl 20 0 20)
; #[entity 3 1]
(solid:subtract cube cyl)
; #[entity 2 1]
;; OUTPUT Original
(define edge1 (pick:edge
    (ray (position 30 0 0) (gvector 0 0 1))))
; edge1
(blend:entities edge1 ”round” 20 ”fix”)
; (#[entity 5 1] #[entity 6 1]
; #[entity 7 1] #[entity 8 1])
;; OUTPUT Result
```

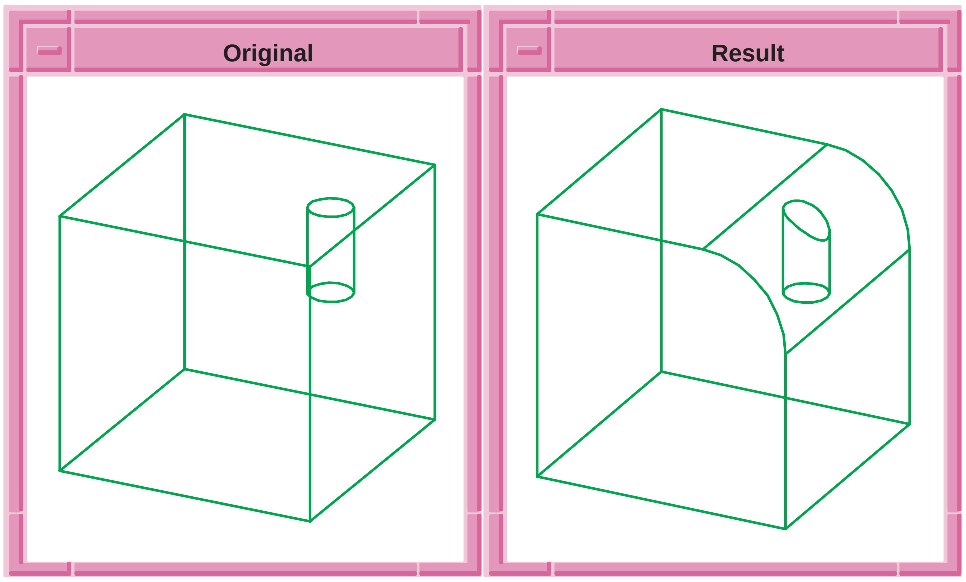


Figure 1-41. Pocket Lost without Global Checking

Different Edge and Face Topology

If blend edge topology and sheet face topology are different, the scope of blending as a kind of “local operation” can occasionally be exceeded in some circumstances where it may have worked with global interference checking. In this example, the “tunnel” that “takes” the edge through to the far side of the block misleads the algorithms and the blend fails without global interference checking.

Scheme Example

```
; The example would fail if this option is not set
(option:set ”bl_remote_ints” #t)
; #f
(define cube1 (solid:block
    (position –30 –30 –30)
    (position 30 30 30)))
; cube1
(define cube2 (solid:block
    (position –30 –30 –30)
    (position 30 30 30)))
; cube2
(entity:move cube2 30 0 30)
; #[entity 2 1]
(solid:subtract cube1 cube2)
; #[entity 1 1]
(define cube3 (solid:block
    (position –20 –10 –20)
    (position 20 10 20)))
; cube3
(define cube4 (solid:block
    (position –1 –10 –1)
    (position 1 10 1)))
; cube4
(solid:subtract cube3 cube4)
; #[entity 3 1]
(solid:unite cube1 cube3)
; #[entity 1 1]
;; OUTPUT Original
(define edge1 (pick:edge
    (ray (position –5 0 0) (gvector 1 0 0))))
; edge1
(blend:entities edge1 ”round” 15 ”fix”)
; (#[entity 6 1] #[entity 7 1] #[entity 8 1])
;; OUTPUT Result
```

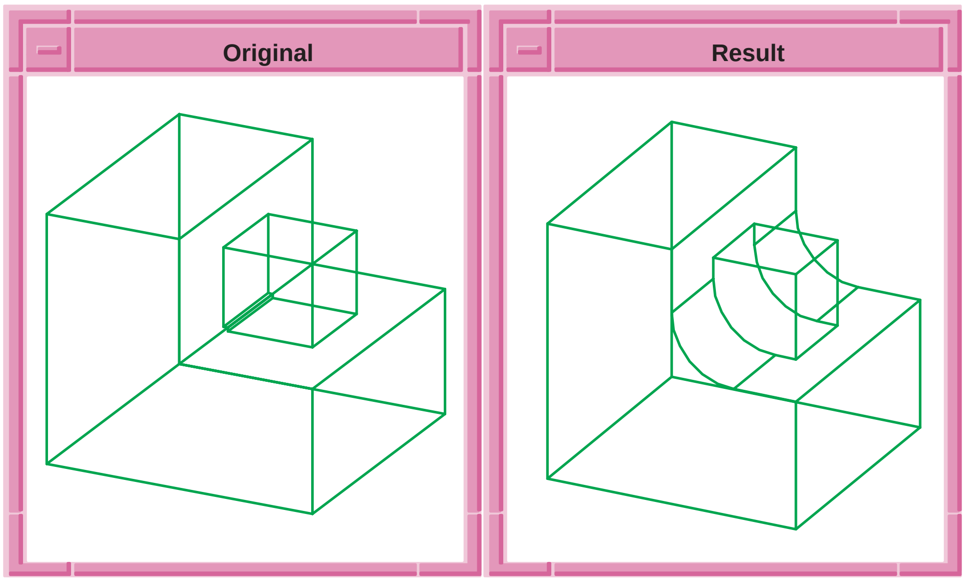


Figure 1-42. Different Edge and Face Topology

Blends on Sheet Bodies

Topic: *Blending

ACIS handles blends on sheet bodies (*sheet blends*). This allows blending between two surfaces, rather than just blending faces that reside in a solid. A sheet body is a free standing face that may belong to an open solid.

Note *A* sheet blend *is not the same as a* blend sheet. *A blend sheet is just a temporary sheet with single-sided faces that is created as part of the blend process on solids. The blend sheet may been seen during single-step blending, but it is* not *part of the completed blend.*

Compared to blends on solids, blends on sheet bodies differ only where they meet the sheet boundary. If an open end of a sequence of blended edges lies on the boundary of the sheet, there is no adjacent face available to intersect with the blend surface in order to trim that surface at the open end. However, the end points of the trimming curve are known; these are the points at which the spring curves of the blended edge meet the sheet boundary. The two points are joined with a curve defined in the parameter space of the blend surface.

For chamfers, the curve will be a straight line in parameter space. For rounds, the curve is one that at it its ends is tangent to the edges of the sheet at the two points. Special cases where a 3D straight line or circle can be used are recognized. Figure 1-43 shows a simple round sheet blend on two double-sided planar sheets.

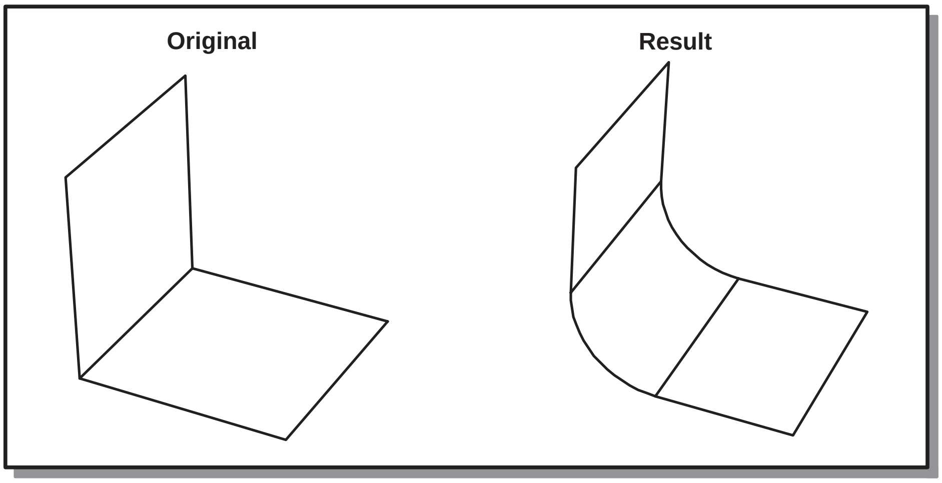


Figure 1-43. Simple Sheet Blend

Where an end point of the joining curve lies at a vertex of the sheet body boundary, the end tangent is not defined, and so a curve is made in parameter space that meets the end point in position only.

Figure 1-44 shows sheet blends in which one of the double-sided planar sheets contains a V-notch. The first sheet blend goes through the center of the notch. The second sheet blend goes through the notch, avoiding the vertices. This is an example of an end point at or away from the sheet boundary vertex.

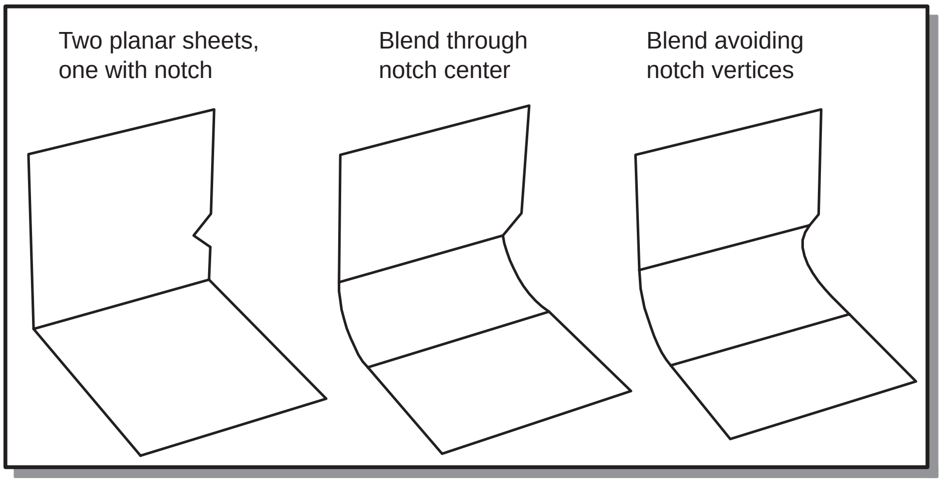

Two planar sheets, one with notch
Blend through notch center
Blend avoiding notch vertices

Figure 1-44. Blend Through Notch

The faces being blended must be all single-sided or all double-sided. Blends at nonmanifold edges (edges meeting one or more than two faces; e.g., an edge of a cellular body where an internal face meets two external faces) are not permitted. Equally, the spring curves of the blend at an end of a sheet blend must not end at nonmanifold edges. This rules out blends such as a blend along a swept L-shaped sheet that is built into a solid. Away from the boundary of the sheet, a blend on a sheet body behaves like a blend on a solid, and can contain vertex blends, miters and so on. An error will be signaled if there is an attempt to set a blend on a vertex or edge of a sheet boundary.

Sheet Blends Between Disconnected Faces

Topic: Blending

Standard blending allows blends between sheet faces when there are edges connecting them. Blends between disconnected sheet faces are enabled through entity–entity blends using the Advanced Blending Component.

Complex Sheet Blends

Topic: Blending

Figure 1-45 shows two examples of complex sheet blends. The first is a wiggle blended with part of a cylindrical face. The second is a wiggle blended with two free standing faces, which were originally from a cube.

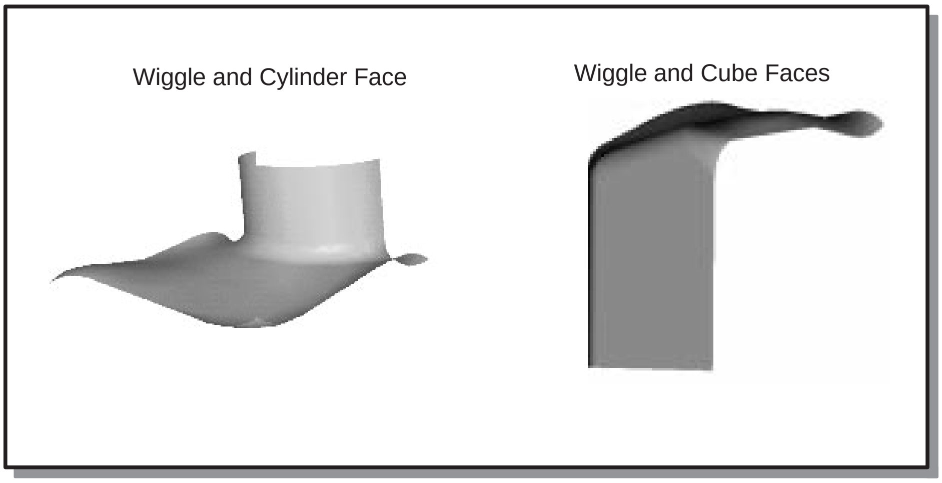


Figure 1-45. Complex Sheet Blends with Wiggles

Figure 1-46 illustrates a complex sheet blend. This example shows round blends, a vertex blend, and a miter blend in a sheet blend on three faces and a wiggle. The second image in the figure shows that the back faces were removed to create a single-sided sheet. The blends to be completed are marked in the third image of the figure.

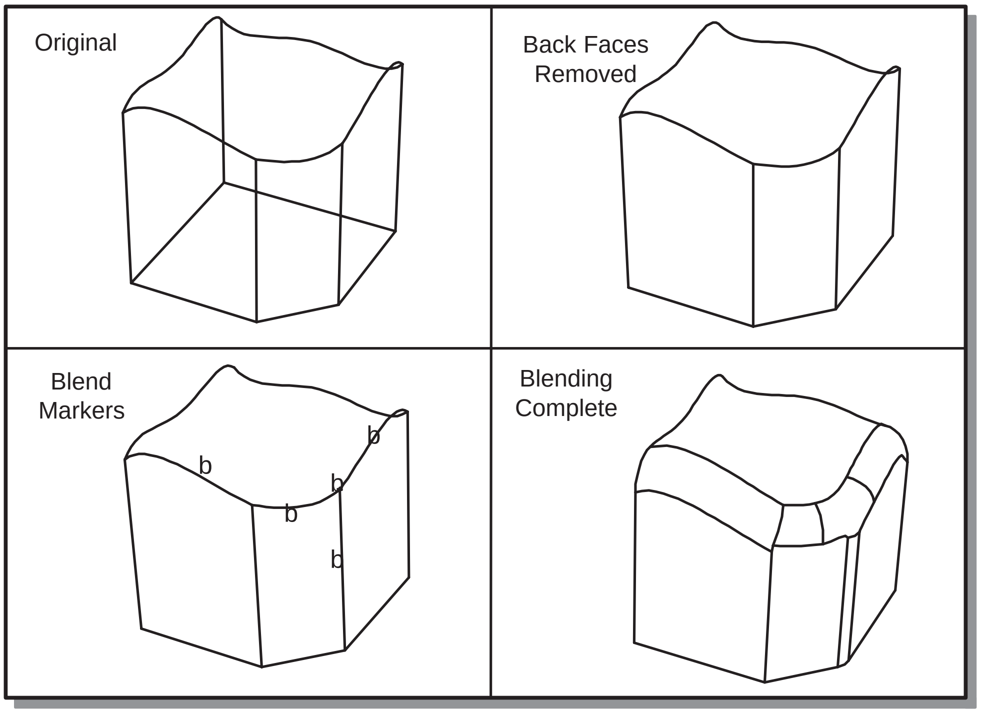


Figure 1-46. Complex Sheet Blend on a Wiggle

Blend Attributes

Topic: *Blending

Attributes are used to set up blends and blend sequences on edges and vertices. A blend is assigned by picking an edge or vertex and describing the blend characteristics, such as its radius. Attributes of class ATTRIB_BLEND are attached to the picked entities. When an entity with a blend attribute is picked to have its blend fixed, the entire set of connected entities with blend attributes is fixed in one operation. This mechanism gives added control and is used to implement various blend construction techniques.

Blend attribute methods test for equality of blends, continuity across a blend (i.e., position-continuous for chamfers, and slope-continuous for rounds), and indicate the size of a blend (in particular, whether it is zero).

Rolling ball blends that have simple curves (straights or ellipses) as their spines or spring curves are recognized as simple. This leads to economies in storage space and compute times, and to improved numerical stability.

There is a general routine for finding blend geometry (spring curves plus auxiliary surfaces and blend surface). There are other more specific routines that find a spine curve, spring curves, the blend surface, or a cross curve.

Derived classes for new blend geometries need supply only as much new geometric code as is needed. For example, one might adopt the standard method of finding spring curves but construct a different blend surface through them.

The basic blend attribute class hierarchy is shown in Figure1-47 (not all blend attribute classes are shown in this figure).

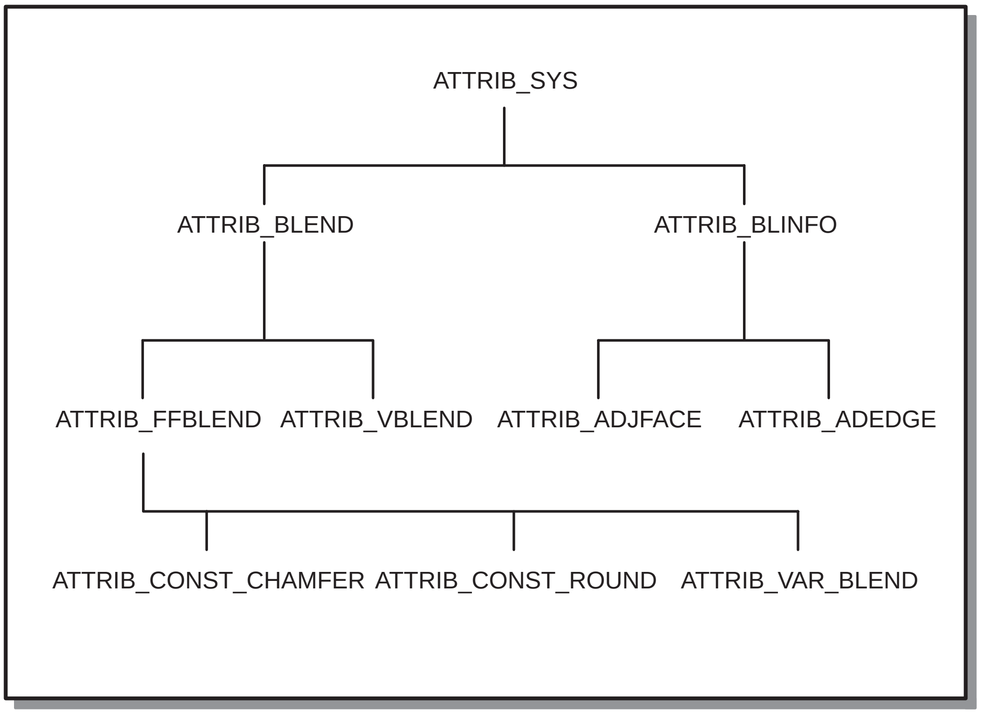


Figure 1-47. Blend Attribute Class Hierarchy

The two types of blend attribute classes are:

- Those derived from ATTRIB_BLEND, which are attached to the body to be blended and indicate the blend operations requested.
- Those derived from ATTRIB_BLINFO, which are attached to the blend sheet during blending to record its relationship with the blank body.

Standard Blending Error Messages

Topic: *Blending, *Debugging

The table below lists the standard blending error messages. This includes the mnemonic code and the message text for each error message.

Mnemonic Code	Message Text
BL_BAD_INTERCEPT	cannot find body edge to trim spring curve during capping.
BL_BAD_REVERSE	bad surface surface intersections during capping
BL_BAD_SLICES	error during var radius marching
BL_BAD_SPINE	cannot iterate to spine pt.
BL_BAD_VBL_BNDRY	vertex blend boundary is malformed or unsuitable
BL_BAD_ZERO_BL_ED	incompatibly blended edge at bi–blend vertex
BL_BLEND_TOO_BIG	blend radius too big for adjacent face, or edge curvature
BL_CHAMFER_NOT_IMPL	chamfer case not implemented
BL_CHAMF_ERR	unable to find spine (blend too big ?)
BL_CI_NOT_FINITE	curve–interval must be finite
BL_CURVES_DIFFER	operator requires curve–intervals on the same curve
BL_CUSP_GEOM_TOO_CMPLX	cuspate geometry too complex to blend
BL_EMPTY_INT_GRAPH	bad wire body present
BL_EMPTY_MARCH_INT	interval of working curve to be marched is empty
BL_EMPTY_SEQUENCE	no edges in sequence
BL_EMPTY_SHEET	empty sheet body made
BL_END_TOO_CMPLX	geometry or topology at end of blend too complex
BL_GEOM_CONSTRUCTION_FAILED	unable to construct Acis geometry for blend
BL_GEOM_TOO_CMPLX	geometry too complex to blend (blend may be too big)
BL_INT_OFF_CURVE	start or end intersection not on intersection curve
BL_MARCH_FAILED	marching failed for variable radius blend
BL_MITRE_AWRY	cannot find mitre details where open sheet ends meet

Mnemonic Code	Message Text
BL_MITRE_TOO_CMPLX	mitred vertex too complex to process
BL_NON_EDVERT	blended entities must be edges or vertices
BL_NON_MAN_VERT	vertex at end of blend sequence too complex
BL_NON_U_CVXTY	cannot blend a partly–convex, partly–concave edge
BL_NOT_IMPLEM	this blend not yet implemented
BL_NO_BLEND_EDGE	unable to find blended edge for variable radius blend
BL_NO_CAP	no capping faces could be found
BL_NO_CAP_EXTN	geometry extension failure in capping
BL_NO_CLEAN_INT	no clean intersection between capping face and sheet face
BL_NO_CUTS_MIXED	cannot adjust lateral edge end with mixed convex-ity
BL_NO_DEF_CURVE	unable to find defining curve for variable radius blend
BL_NO_FACE_AT_VERT	cannot find lateral face at end vertex
BL_NO_INIT	call to initialise_blend_one omitted
BL_NO_MATE	mating topology or geometry for vertex blend not found
BL_NO_MITRE_MIXED	can only mitre edges of the same convexity
BL_NO_PROJ_CUR	no projection curve during imprint
BL_NO_SHEET_SURF	sheet surface is missing
BL_NO_SPR_CUR_INT	spring curves do not intersect
BL_NO_VTX_GEOM	no vertex blend geom created
BL_NO_X_CURVE	cannot find cross curve (blend too big?)
BL_OBLIQUECRVS_NOT_SAVED	cannot save oblique cross curve data on blend
BL_POINT_CUR	error in point_cur
BL_POINT_CUR_SURF	error in point_cur_surf
BL_RADIUS_CALIBRATION	unable to calibrate radius function
BL_SETBACK_TOO_LARGE	combination of edge setbacks is longer than the edge

Mnemonic Code	Message Text
BL_TOO_MANY_SM_EDS	too many smooth edges at bi–blend vertex
BL_UNBL_VERTEX	cannot assign vertex blend to a bi–blend vertex
BL_UNFIN_SHEET	unable to complete sheet
BL_UNKNOWN_BLEND	blend of unknown type or on unexpected entity
BL_UNKNOWN_CVXTY	unknown convexity
BL_VTX_EDGE_UNBL	an edge at a general vertex blend should be blended

Error Message Explanations

Topic: Blending, Debugging

Each error message is classified by one or more error types and problem areas. This classification is listed in italics, using the following codes (where the error type is given, followed by a colon and then the problem area):

Code	Error Type
PROG	Program error
APPL	Application error
USER	User error
LIMT	Current limitation
IMPS	Impossible specification

Code	Problem Area
CAPP	Capping
MITR	Mitering
GEOM	Geometry creation
BLN1	Blend stage 1
DEFN	Definition of blend
GENR	General
COMP	Blend complete
VTBL	Vertex blend
SVRS	Save and restore

For example, if an error is classified as *PROG: CAPP*, it is a program error involving capping.

BL_BAD_INTERCEPT

PROG: CAPP

This message occurs during capping when the algorithm is unable to find a body edge to trim the open spring curve. This could occur if the sheet geometry is not long enough, or if the geometry is badly made so that capping is not possible. Reducing the blend size, making a sheet that can be capped in the given configuration, may help.

Suggestion:

- Try with a smaller blend radius.

BL_BAD_REVERSE

PROG: CAPP

This message occurs if there are any problems with the surface-surface intersections when the sheet edges are being made during capping. This could arise due to bad geometry or due to failure of the intersectors.

BL_BAD_SLICES

PROG: GEOM

This is a variable radius error message. This designates a general failure during the calculation of the variable radius blend surface approximation. It is most likely caused by a failure in calculation of a slice cross section. This is usually caused by a failure in the variable radius evaluators, or sometimes if there is an error in calibrating the radius function, or in many cases if the blend specification is too large.

Suggestion:

- Try with a smaller blend specification.

BL_BAD_SPINE

PROG: GEOM

This message occurs during the calculation of the spine or the path of the center of the ball when the iteration fails to converge. Typical causes could be a bad blend specification, such as a large blend radius, or bad configuration for the faces forming the edge.

Suggestion:

- Try with a smaller blend radius.

BL_BAD_VBL_BNDRY

USER/PROG: GEOM

This message occurs when the algorithm is unable to process the vertex blend due to unavailability of the cross curves. This could be a geometric limitation, a configuration not handled, or a case of improper setbacks applied for the vertex blends.

Suggestion:

- Readjust the setbacks.
- Try changing the size of the blends.

BL_BAD_ZERO_BL_ED

IMPS: DEFN

This message occurs if there is a zero radius blend specification at a smooth bi–blend vertex.

BL_BLEND_TOO_BIG

USER: GEOM

This message occurs if the user is trying to make a blend that is too big for the geometric configuration of the object or is too large compared to the edge curvature.

Suggestion:

- Try with a smaller blend radius.

BL_CHAMFER_NOT_IMPL

LIMT: DEFN

This message occurs if a chamfer is specified that is not implemented or supported.

Suggestion:

- If the Advanced Blending Component is available, try a rounded chamfer blend with a zero bulge

BL_CHAMF_ERR

PROG: GEOM

This message occurs during the chamfer geometry creation if there is trouble creating the spine. This is usually caused by a blend specification that is too big.

Suggestion:

- Try with a smaller chamfer.

BL_CI_NOT_FINITE

PROG: GENR

This is an internal programming error and should not occur under normal circumstances.

BL_CURVES_DIFFER

PROG: GENR

This is an internal programming error and should not occur under normal circumstances.

BL_CUSP_GEOM_TOO_CMPLX

LIMT: BLN1

This message reports a complicated geometry near a cuspate vertex, which the algorithm is unable to handle.

BL_EMPTY_INT_GRAPH

USER/PROG : COMP

This message occurs if the blend wire body created is not continuous and contains gaps, or if the blend sheet and wire are passed into the blend completion process in incorrect order.

Suggestion:

- ○ Check the order of wire and sheet body in the complete command.
- ○ Check the sheet body before the blend wire is created.

BL_EMPTY_MARCH_INT

PROG:GEOM

This is a variable radius error message. This error message occurs while calculating the marching range during geometry creation and there are problems in obtaining the start or end slices, in which case the marching interval is set to a null value. Usually, this problem should be trapped earlier on in the blend processing.

BL_EMPTY_SEQUENCE

USER: DEFN

This is reported when there are no blend attributes attached to the edges in the blend sequence, or no edges in the sequence.

Suggestion:

- ○ Check the blend definition.

BL_EMPTY_SHEET

PROG: BLN1

This message is reported at the end of the blend stage one (sheet creation) processing if an invalid sheet body has been created without a lump, shell or face.

BL_END_TOO_CMPLX

LIMT: BLN1/CAPP

This error is reported when there are problems blending the ends of a blend. A capping failure, a detection of a complex end case, or bad intersections during capping are some of the circumstances that may give rise to this error.

BL_GEOM_CONSTRUCTION_FAILED

PROG: GEOM

This is a variable radius error message. This message is reported if there are problems constructing the geometry of the sheet. This could be caused by an incorrect slice list after marching, which cause the resulting surface to be self-intersecting.

BL_GEOM_TOO_CMPLX

PROG: BLN1

This message occurs due to complicated geometry arising as a result of cuspate vertices, nonmanifold topology, or if the blend is too large to fit the body geometry. Usually, reducing the blend radius results in a possible blend.

Suggestion:

- Try with a smaller blend radius.

BL_INT_OFF_CURVE

PROG:CAPP

This is reported when there are problems with capping the blend sheet. This could occur in the case of complicated end geometry in which there are a number of capping options.

BL_MARCH_FAILED

PROG: GEOM

This is a variable radius error message. Marching fills up intermediate slices along the blend sequence after the end slices have been calculated. An error in marching could occur if the evaluators fail while evaluating the surfaces forming the blend.

BL_MITRE_AWRY

PROG: MITR

This indicates a problem in the mitering process. This could be due to a surface extension problem during mitering, a problem with miter intersections, or complications in obtaining the sheet segment ends for mitering.

Suggestion:

- Try with smaller blends.

BL_MITRE_TOO_CMPLX

LIMT:MITR

This message is reported when the algorithm detects that the given mitering problem is topologically too complicated to handle. Conditions such as too many edges at the vertex, a small edge present near the mitering vertex, or other topologically complex conditions near the vertex lead to this message.

BL_NON_EDVERT

USER: DEFN

This message indicates a user error in the assignment of blend attributes on entities other than bodies, edges, and vertices.

BL_NON_MAN_VERT

LIMT: BLN1

This signals the detection of a nonmanifold vertex in the blend sequence processing, which the blend processing algorithm cannot handle.

BL_NON_U_CVXTY

USER: DEFN

This error is reported when the user tries to blend an edge that transitions from convex to concave, or vice-versa.

BL_NOT_IMPLEM

LIMT: GEN

This message is reported if a case that is not implemented, such as a vertex blend on the apex of a cone or a complicated case of a blend on a cuspate vertex, is encountered.

BL_NO_BLEND_EDGE

PROG: GEOM

This is a variable radius error message. This error is reported during the creation of the defining curve if the first and last edges in the sequence are not uniquely determined. This could mean that the created defining curve is also bad.

BL_NO_CAP

PROG:CAPP

This designates a general capping failure and typically arises in a complicated mixed convexity capping case. Inability to find a suitable capping face to cap the blend also results in this error.

BL_NO_CAP_EXTN

PROG: CAPP

This error is generated when there is a failure to extend a particular capping face.

BL_NO_CLEAN_INT

PROG:CAPP

This message indicates a capping failure occurring at the open end of the sheet. This could arise due to bad geometry or failure of the intersectors.

BL_NO_CUTS_MIXED

PROG: BLN1

This occurs if there are problems with imprinting the sheet geometry onto the body. It could arise due to improper sheet geometry created, or due to difficulties in intersecting the spring curves with the near faces at a mixed convexity end.

BL_NO_DEF_CURVE

PROG: GEOM

This is a variable radius error message. Integral to the creation of a variable radius blend is the defining curve, which determines the parameterization for the blend. This is a composite curve created from the edge geometry making up the sequence of edges. If the algorithm detects that the defining curve is missing, this error is generated. This should not occur under normal circumstances.

BL_NO_FACE_AT_VERT

PROG: CAPP

This error occurs during capping when the algorithm has problems trimming the spring curve by a body edge. This is usually when the spring curve degenerates to a point.

BL_NO_INIT

APPL: GENR

This is an initialization error caused when the application fails to call the initialise_blend_one function before doing any blend operation.

BL_NO_MATE

PROG: VTBL

This error occurs during a vertex blend when the algorithm has problems finding the end geometry for an edge with which the vertex blend connects. This may be due to use of a blend radius that is too large, or due to improper specification of the setbacks at the vertex blend.

BL_NO_MITRE_MIXED

IMPS: MITR

This indicates that opposite convexity edges are specified for a mitering.

BL_NO_PROJ_CUR

PROG: BLN1

This error occurs during the imprinting of the sheet on the body in cases of degeneracy in which the spring curves degenerate to points and the algorithm is unable to determine a suitable projection curve for the imprint.

BL_NO_SHEET_SURF

PROG: BLN2

This message is reported whenever the algorithm detects that the sheet underlying face geometry has not been made, or in cases of parametric surfaces if the bs3_surface is absent.

BL_NO_SPR_CUR_INT

PROG: VTBL

This denotes a failure in intersecting the spring curves during the creation of a vertex blend surface. These intersection points are used to limit the extent of the vertex blend surface.

BL_NO_VTX_GEOM

PROG: VTBL

This is reported when the program is unable to create the vertex blend surface for any reason.

BL_NO_X_CURVE

PROG: BLN1

This error occurs when the algorithm is unable to find a cross-curve at an open end of a blend sequence. Because this curve is obtained by intersecting surfaces, a failure in the same would trigger the error. It could also occur if the blend radius is too big or if a cross-curve is found to be missing during any part of the processing.

BL_OBLIQUECRVS_NOT_SAVED

USER: SVRS

This warning is reported when a body containing oblique cross-curve vertex blends is saved in an older save file format (pre–1.7) in which these blends were not supported.

BL_POINT_CUR

PROG: BLN1

This is an internal programming error that should not occur under normal circumstances.

BL_POINT_CUR_SURF

PROG: BLN1

This is an internal programming error that should not occur under normal circumstances.

BL_RADIUS_CALIBRATION

PROG: GEOM

This is a variable radius error message. This denotes an error in the radius calibration; i.e., the assignment of the radius function over the sequence of edges making up the blend. Possible reasons could be the inability to make the procedurally defined blend surface or failure to determine the terminal edges in the edge sequence.

BL_SETBACK_TOO_LARGE

USER: DEFN

This indicates that a user tried to assign setbacks at a vertex along an edge that are larger than the length of the edge or that the combination of all the setbacks on the edge is larger than the edge length.

Suggestion:

- Reduce the currently assigned setbacks.

BL_TOO_MANY_SM_EDS

USER: DEFN

This is a warning issued when there are too many smooth edges at a bi-blend vertex, which could create subsequent problems.

BL_UNBL_VERTEX

USER: DEFN

This error message is reported when a vertex blend is assigned to a bi-blend smooth vertex.

BL_UNFIN_SHEET

PROG: BLN1

This occurs when the algorithm is unable to create a sheet segment for a particular attribute. This results in an incomplete sheet if the other attributes have been processed. This could be due to complicated geometry or numerical problems in the blend evaluations.

Suggestion:

- Reducing the blend radius might help in cases of numerical problems.

BL_UNKNOWN_BLEND

PROG: GENR

This error is generated by the base class blend attribute sheet creation method. All the blend attributes have their own derived methods and hence the base class method should not get called under normal circumstances.

BL_UNKNOWN_CVXTY

PROG: BLN1

This error is reported if the algorithm is unable to determine the convexity of an edge undergoing blending.

BL_VTX_EDGE_UNBL

USER: DEFN

This indicates the user failed to assign a blend on an edge starting on a vertex on which a vertex blend is placed. All the edges meeting at the vertex must have blend attributes assigned to them.

Suggestion:

- Assign blend attributes to the unblended edge(s).