Chapter 2.
Advanced Blending Examples

Topic: Blending

The examples in this chapter illustrate some advanced blending concepts and the functionality available in ABL. These examples use Scheme extensions that can be entered in the Scheme based demonstration application, Scheme AIDE. Only entity–entity blends are featured in these examples.

ABL Scheme Extension Examples

Topic: *Blending, *Examples

The following examples illustrate performing advanced blending using Scheme extensions. Each example is followed by plots of the model before and after the blend is applied.

Refer to *Chapter 3, Scheme Extensions*, for a description of Scheme extension syntax and the Scheme extensions used in the following examples.

The Scheme based demonstration application, Scheme AIDE, generates and returns a numeric identifier (ID) whenever a new entity is created. The application starts the ID at one (1) when the first entity is created during the session and increments the ID whenever a new entity is created. Thus, the actual ID returned for a given entity depends on how many entities have been created before. The examples below often use these entity IDs in subsequent operations to refer to previously created entities. Each example assumes that no operations have been performed in the current session, so the IDs start at one (1). In practice, if you run these examples after performing other operations that create entities, you must adjust the entity IDs in the examples accordingly.

Remote Face–Face Blend

Topic: Blending

In this example, a face–face blend is produced. The two faces are remote, so there is no blended edge.

Scheme Example

```
; create a block
(define after (solid:block (position –30 –30 –30)
    (position 30 30 30)))
; create a wire–body
(define wireframe (wire–body:points(list (position 20 –30 30)
    (position 20 30 30) (position 30 30 20)
    (position 30 –30 20) (position 20 –30 30))))
; create a sheet from the wire–body
(sheet:cover–wires wireframe)
; create a solid from the sheet body
(define body1 (sweep:law wireframe (gvector 30 0 30)))
; subtract the solid from the block
(solid:subtract after body1)
;; OUTPUT Original
; select faces to blend
(define face1 (pick:face (ray (position 0 0 0)
    (gvector 0 0 1)) 1))
(define face2 (pick:face (ray (position 0 0 0)
    (gvector 1 0 0)) 1))
; create a vradius
(define rad1 (abl:const–rad 30))
; create an entity–entity blend
(abl:ent–ent–blend face1 face2 rad1 (position 0 0 0) #t)
; fix the blend
(blend:fix after)
;; OUTPUT Result
```

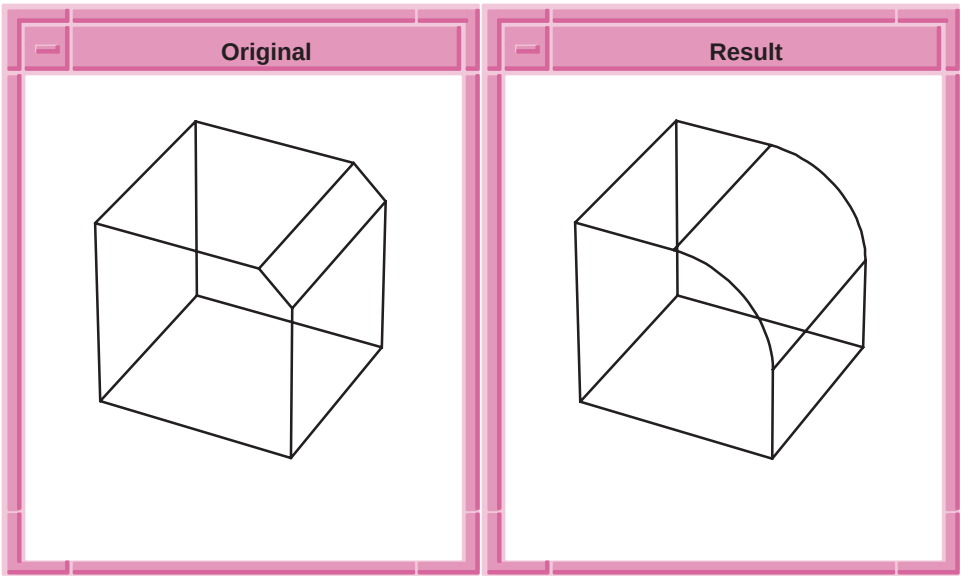


Figure 2-1. Remote Face–Face Blend

Blending Disjoint Faces

Topic: Blending

In the next example, the two faces to be blended are in disjoint shells. The blend joins the two shells into one.

Scheme Example

```
; abl:ent-ent-blend
; Create a solid block
(solid:block (position -40 -40 -5) (position 40 40 5))
;; #[entity 2 1]
; Create a sphere
(solid:sphere (position 0 0 40) 30)
;; #[entity 3 1]
; Unite the solids
(bool:unite (entity 2) (entity 3))
;; #[entity 2 1]
(zoom-all)
; OUTPUT Original
```

```
; Define vradius
(define rad (abl:const-rad 10))
;; rad
; Define side entities
(define face1 (pick:face (ray (position 0 0 0)
    (gvector 0 0 1)) 1))
;; face1
(define face2 (pick:face (ray (position 0 0 30)
    (gvector 1 0 0)) 1))
;; face2
; Do blending
(abl:ent-ent-blend face1 face2 rad
    (position 30 0 10) #f)
;; #[entity 2 1]
; Fix the body
(blend:fix (entity 2))
;; #t
; OUTPUT Result
```

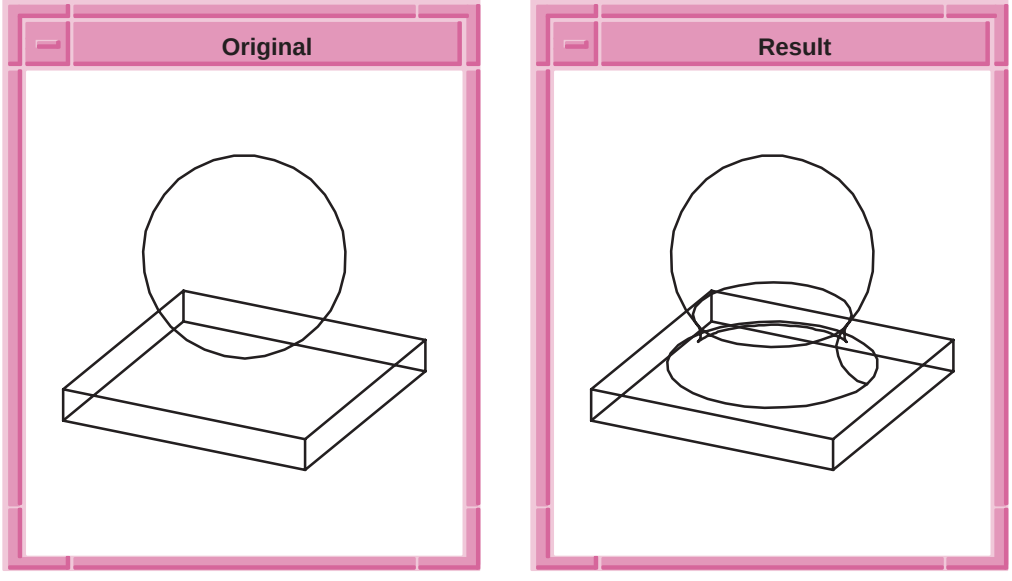


Figure 2-2. Blending Disjoint Faces

Edge–Face Blend

Topic: Blending

An edge–face blend is created by picking an edge and a face, rather than two faces. In this case, the approximate starting point must indicate on which side of the edge the rolling ball will rest. The convexity indicates on which side of the face the ball rests. The edge and the face may be picked in either order.

Scheme Example

```
(define after (solid:block (position –5 –30 –30)
    (position 5 30 30)))
; OUTPUT Original
; select the edge and face for blending
(define edge1 (pick:edge (ray (position –5 0 0)
    (gvector 0 0 1))))
(define face1 (pick:face (ray (position 0 0 0)
    (gvector 1 0 0)) 1))
; create a vradius
(define rad1 (abl:const–rad 30))
; create the blend
(abl:ent–ent–blend edge1 face1 rad1 (position 0 0 0) #t)
; fix the blend
(blend:fix after)
; OUTPUT Result
```

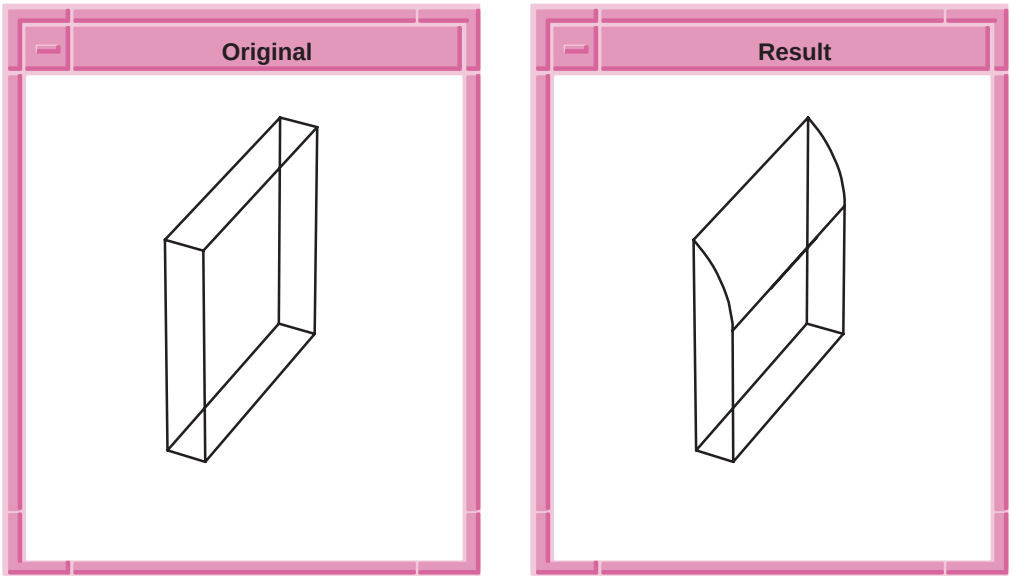


Figure 2-3. Edge–Face Blend

Blend Along Faces

Topic: Blending

In the next example, an entity–entity blend traverses smoothly from one face to the next. At the end of the sequence there is no smooth face on which to roll, so the blend terminates at that point.

Scheme Example

```
; abl:ent-ent-blend
; Create a solid block
(solid:block (position -20 -20 -20)
    (position 20 20 20))
;; #[entity 2 1]
; Create a cylinder
(solid:cylinder (position 20 0 -20)
    (position 20 0 20) 20)
;; #[entity 3 1]
; Unite the solids
(bool:unite (entity 2) (entity 3))
;; #[entity 2 1]
; OUTPUT Original
```

```
; Define vradius
(define rad (abl:const–rad 10))
;; rad
; Define side entities
(define face1 (pick:face (ray (position 0 0 0)
    (gvector 0 0 1)) 1))
;; face1
(define face2 (pick:face (ray (position 0 0 0)
    (gvector 0 –1 0)) 1))
;; face2
; Do blending
(abl:ent–ent–blend face1 face2 rad (position 0 0 0) #t)
;; #[entity 2 1]
; Fix the body
(blend:fix (entity 2))
;; #t
; OUTPUT Result
```

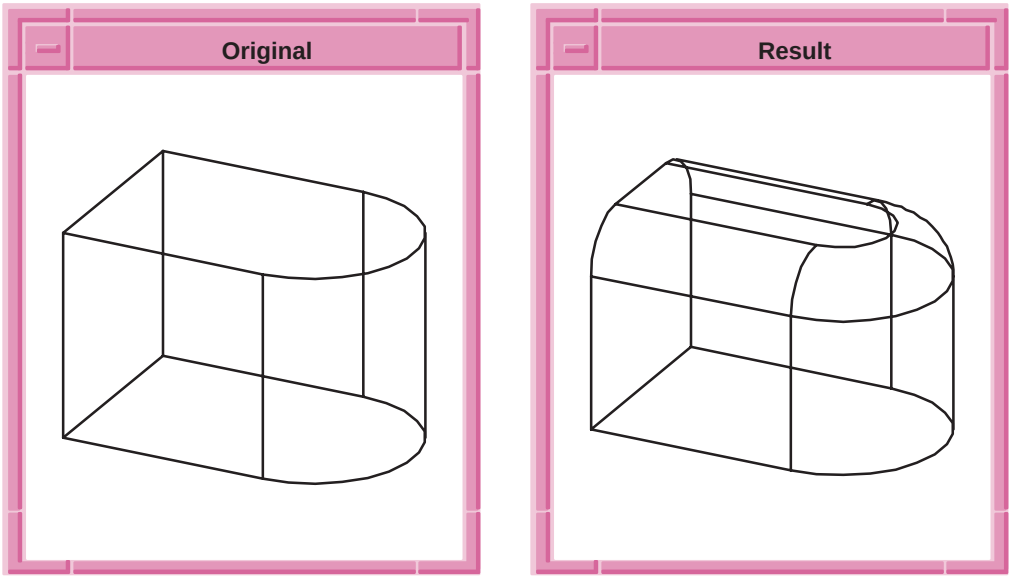


Figure 2-4. Blend Along Faces

Blend Along Edges

Topic: Blending

For edges, the rule is that when the end of an edge is reached and there is a tangent edge at that vertex, then the blend sequence will continue onto that tangent edge. If there is not, then the blend will terminate. A blend rolls from one tangent edge onto the next.

Scheme Example

```
(define block1 (solid:block (position –20 –20 –2.5)
    (position 20 20 2.5)))
(define cylinder1 (solid:cylinder  (position 20 0 –2.5)
    (position 20 0 2.5) 20))
(define after (solid:unite block1 cylinder1))
; OUTPUT Original
; select the face and edge for blending
(define face1 (pick:face (ray (position 0 0 0)
    (gvector 0 0 1)) 1))
(define edge1 (pick:edge (ray (position 0 –20 0)
    (gvector 0 0 –1))))
; create a vradius
(define rad1 (abl:const–rad 10))
; create the blend
(abl:ent–ent–blend face1 edge1 rad1 (position 0 0 0) #t)
; fix the blend
(blend:fix after)
; OUTPUT Result
```

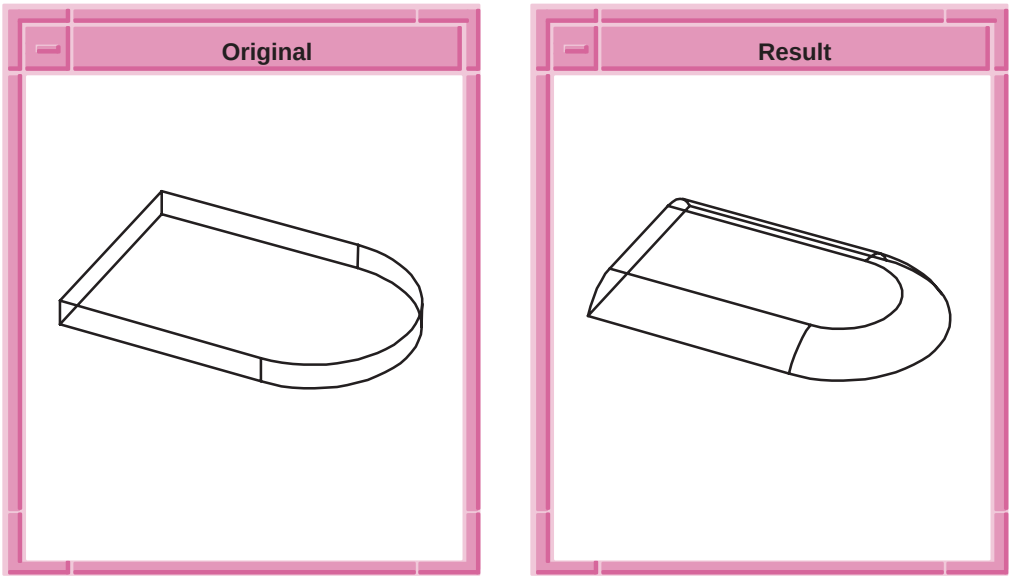


Figure 2-5. Blend Along Edges

A complication with blends running along edges is that they sometimes come away from the edge and roll onto the adjoining face. When this happens, that adjoining face is automatically included into the sequence. In the following example, a blend comes off an edge onto the adjoining face. It does not make sense to try and stop the blend at that point, as there is no way to close it off sensibly. In addition to the original edge–face blend, a face–face blend is also included in the sequence. Compare this example, in which the initial starting blend is edge–face, with the ”Forcing End Cap on Edge–Face” example, in which the blend starts as face–face and rolls on to the edge. The two examples produce the same results.

Scheme Example

```
(define before (solid:block (position –30 –30 –10)
    (position 30 30 10)))
(entity:transform before (transform:translation
    (gvector 0 0 –10)))
(define wb (wire–body:points (list (position –30 –30 0)
    (position –30 –30 10) (position –30 30 30)
    (position –30 30 0) (position –30 –30 0))))
(sheet:cover–wires wb)
(define body1 (sweep:law wb (gvector 30 0 0)))
(solid:unite before body1)
; OUTPUT Original
; select the edge and face for the blend
(define edge1 (pick:edge (ray (position 0 0 1)
    (gvector 0 0 1))))
(define face1 (pick:face (ray (position 1 0 1)
    (gvector 0 0 –1)) 1))
; create the vradius
(define rad1 (abl:const–rad 20))
; create the blend
(abl:ent–ent–blend edge1 face1 rad1 (position 20 0 20) #f)
; fix the blend
(blend:fix before)
; OUTPUT Result
```

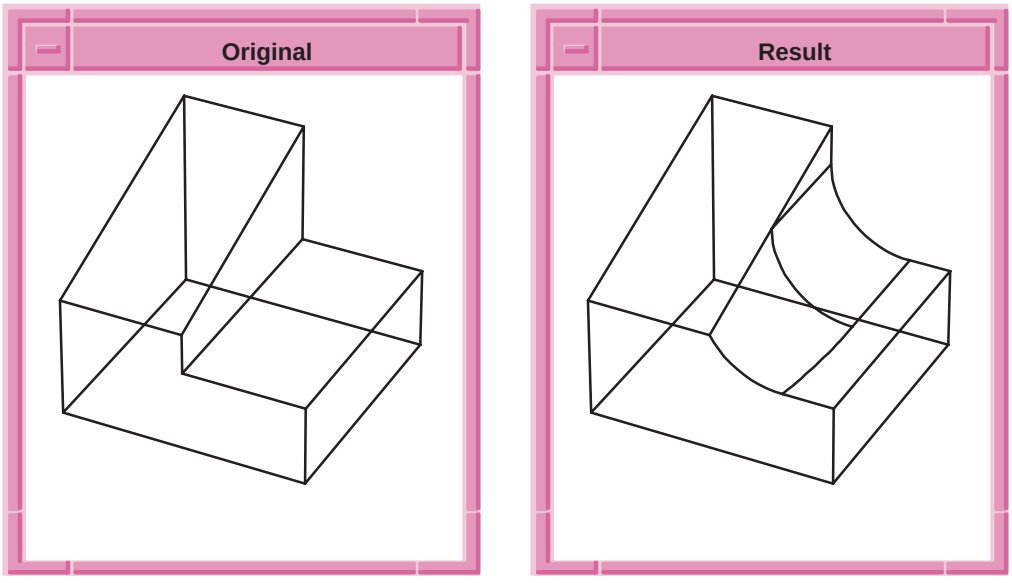


Figure 2-6. Rolls Onto Adjoining Face

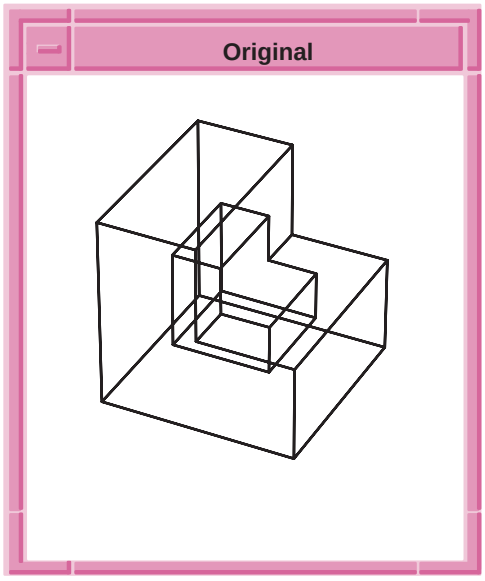
Ambiguities in a Blend Sequence

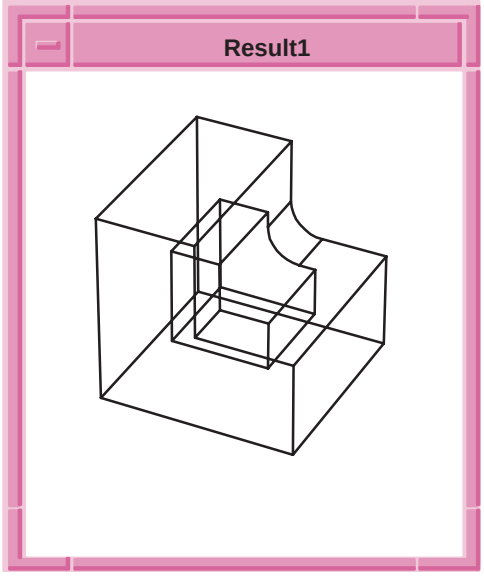
Topic: Blending

Occasionally, specifying a pair of entities does not define a unique set of faces to be made. In these cases the starting point is used to select the desired one. There are two distinct blends that could exist between the two faces in the following example. By placing the approximate starting point of the rolling ball's center in the region where it sweeps out the desired blend face, we can specify which blend is produced. Compare bodies `b1` and `b2`, which represent the same face–face blend on the original body, but with different starting positions for the rolling ball.

Scheme Example

```
(define after (solid:block (position –30 –30 –30)
    (position 30 30 30)))
(define cube2 (solid:block (position –30 –30 –30)
    (position 30 30 30)))
(entity:transform cube2
    (transform:translation (gvector 30 0 30)))
(solid:subtract after cube2)
(define cube3 (solid:block (position –15 –15 –15)
    (position 15 15 15)))
(solid:subtract after cube3)
; OUTPUT Original
; select faces for blending
(define face1 (pick:face (ray (position 25 0 25)
    (gvector 0 0 –1)) 1))
(define face2 (pick:face (ray (position 25 0 25)
    (gvector –1 0 0)) 1))
; create the vradius
(define rad1 (abl:const–rad 10))
; make a copy of the entity
(define step2 (entity:copy after))
; create the entity–entity blend
(abl:ent–ent–blend face1 face2 rad1 (position 0 25 0) #f)
; fix the blend
(blend:fix after)
; OUTPUT Result1
; delete the modified entity
(entity:delete after)
; select the faces for blending
(set! face1 (pick:face (ray (position 25 0 25)
    gvector 0 0 –1)) 1))
(set! face2 (pick:face (ray (position 25 0 25)
    gvector –1 0 0)) 1))
; create the entity–entity blend
(abl:ent–ent–blend face1 face2 rad1 (position 0 –25 0) #f)
; fix the blend
(blend:fix step2)
; OUTPUT Result2
```





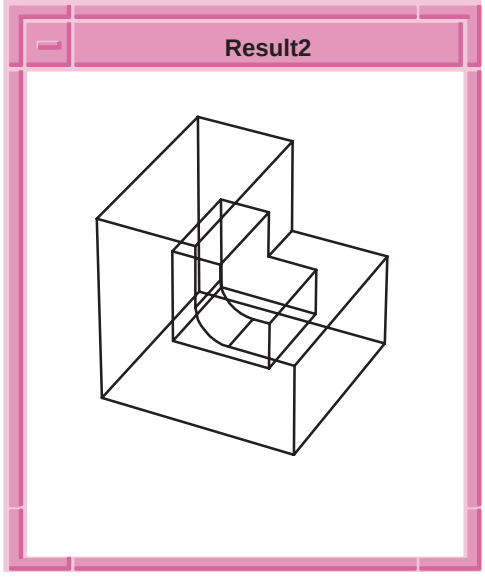


Figure 2-7. Ambiguities in a Blend

If the starting point does not lie within one of the two possible regions (e.g., if 0 0 0 had been given as the starting point in the previous example), then it is undefined which of the two blends would be produced. If the topologies and geometries involved allow only one possible blend, however, then the correct regions should be made, whether or not the starting point lies within it.

Blend Termination and Capping

Topic: Blending

Blend termination means that the rolling ball does not roll onto any of the new entities it may have encountered, but rather attempts to trim the blend face to the body. Normally this involves intersecting the blend surface with existing body faces, or possibly the extensions of existing body faces. This process is referred to as capping. When capping terminates a blend, it is referred to as an end cap. End caps are seen in all the previous examples, except in the blending disjoint faces example.

Capping may not always terminate the blend. The cap may not cover the end of the face but merely a side of it. Such caps are called side caps. Whether a cap is an end cap or side cap is a property of the geometries and topology in that region; it is not a user choice. The other common way that a blend may terminate is by meeting itself, such as in the blending disjoint faces example.

The rule governing how a blend progresses across a body may be explained by saying that whenever the ball leaves the current entity, the blend must either roll onto one of the new entities encountered at that point, or it must start capping itself there. This latter action may or may not terminate the blend, depending on whether the cap is an end cap or a side cap.

The following example illustrates an edge–face blend with side caps. There are four side caps where the ball rolls off the face, and one further side cap where it rolls off the edge. None of the caps are end caps, and the blend eventually terminates by meeting itself again.

Scheme Example

```
(define tube1 (solid:cylinder
(position 0 0 –10)
               (position 0 0 10) 20))
(define after (solid:block (position –30 –30 –5)
               (position 30 30 5)))
(entity:transform after
    (transform:translation (gvector 0 0 –5)))
(solid:unite after tube1)
(define block1 (solid:block (position –30 –10 –10)
               (position 30 10 10)))
(entity:transform block1
    (transform:translation (gvector 30 0 15)))
(solid:subtract after block1)
; OUTPUT Original
; select face and edge for blending
(define face1 (pick:face (ray (position 25 0 5)
               (gvector 0 0 –1)) 1))
(define edge1 (pick:edge (ray (position –10 0 10)
               (gvector –1 0 0))))
; create the vradius
(define rad1 (abl:const–rad 15))
; create the entity–entity blend
(abl:ent–ent–blend face1 edge1 rad1 (position 30 0 10) #f)
; fix the blend
(blend:fix after)
; OUTPUT Result
```

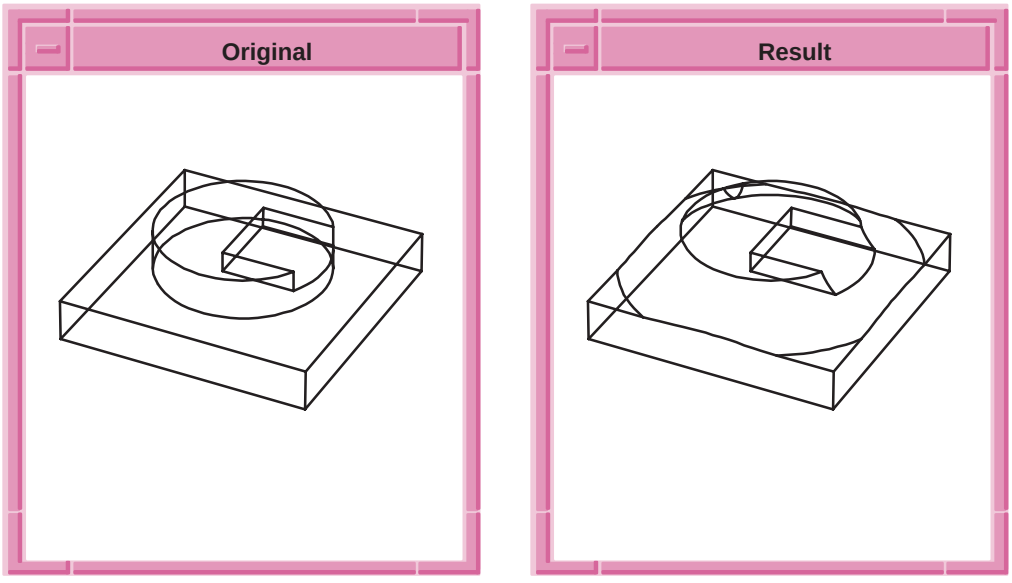


Figure 2-8. Edge–Face Blend with Side Caps

Scheme Example

```
(define after (solid:block (position –30 –30 –30)
               (position 30 30 30)))
(define cube2 (solid:block (position –30 –30 –30)
               (position 30 30 30)))
(entity:transform cube2
    (transform:translation (gvector 30 0 30)))
(solid:subtract after cube2)
(define cube3 (solid:block (position –5 –15 –5)
               (position 5 15 5)))
(entity:transform cube3
    (transform:translation (gvector 0 0 10)))
(solid:subtract after cube3)
(define cube4 (solid:block (position –5 –15 –5)
               (position 5 15 5)))
(entity:transform cube4
    (transform:translation (gvector 10 0 0)))
(solid:subtract after cube4)
; OUTPUT Original
;select the faces for blending
(define face1 (pick:face (ray (position 25 0 25)
               (gvector 0 0 –1)) 1))
(define face2 (pick:face (ray (position 25 0 25)
               (gvector –1 0 0)) 1))
; create the vradius
(define rad1 (abl:const–rad 10))
; create the entity–entity blend
(abl:ent–ent–blend face1 face2 rad1 (position 0 0 0) #f)
; fix the blend
(blend:fix after)
; OUTPUT Result
```

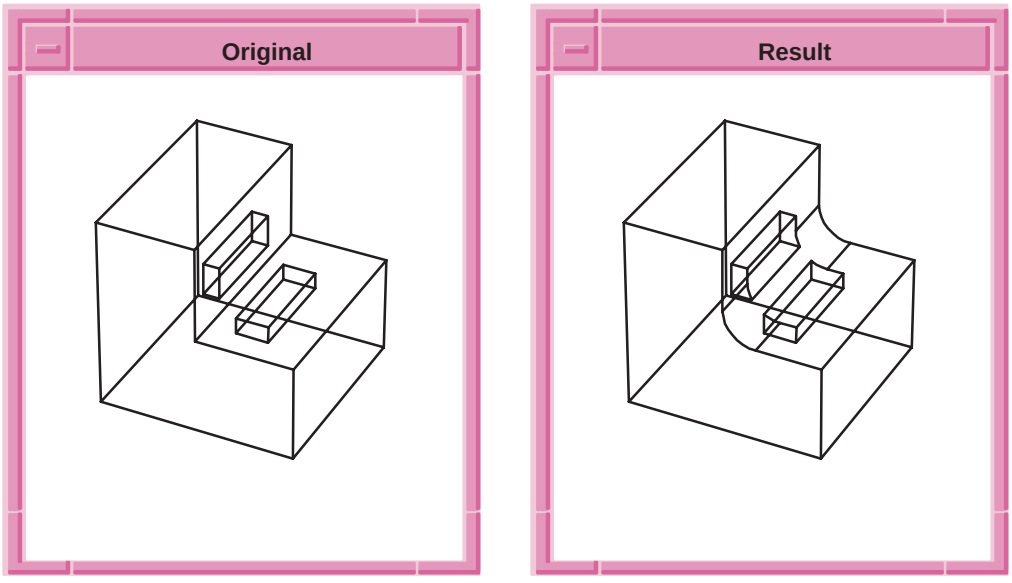


Figure 2-9. Side Caps in the Middle of the Body

Blend Instruction

Topic: Blending

There are occasions when the default behavior of the rolling ball is not desired. The default behavior may be overridden by setting blend instructions on the body. Because an entity–entity blend has no sequence of blended edges, the only place from which instructions can be read is the entities into which the contact points of the ball actually run. Which of the two available instructions to use depends on the behavior choices the blend has at any point. The behaviors are either to roll onto the newly encountered entities, or to begin capping the existing blend instead. The available instructions are:

Roll On . . . . . Is used to tell the blend to roll onto the encountered entity when the default would have been to cap the blend at that point

Cap . . . . . Is used to tell the blend to begin capping the existing blend face rather than to roll onto the new entities. Whenever an entity would normally have been rolled onto, placing a cap instruction there forces the blend to cap instead, forming either side or end caps.

A roll on instruction may be placed by the scheme command:

abl:set–instruction `<entity type> 1`

and a cap instruction may be placed by the scheme command:

abl:set–instruction <entity type> 2

where <entity type> could be one of the keywords face, edge, or vertex.

Using Blend Instructions

Topic: Blending

The following example illustrates the use of blend instructions. In this example, a cap instruction (instruction action value = 2) is placed on the first smoothly connected face to prevent the rolling ball from rolling on. Compare this example with example for abl:ent–ent–blend.

Scheme Example

```
; Forcing a blend to cap
; Create a solid block
(solid:block (position –20 –20 –20)
    (position 20 20 20))
;; #[entity 2 1]
; Create a cylinder
(solid:cylinder (position 20 0 –20)
    (position 20 0 20) 20)
;; #[entity 3 1]
; Unite the solids
(bool:unite (entity 2) (entity 3))
;; #[entity 2 1]
; OUTPUT Original
```

```
; Define vradius
(define rad (abl:const–rad 10))
;; rad
; Define side entities
(define face1 (pick:face (ray (position 0 0 0)
    (gvector 0 0 1)) 1))
;; face1
(define face2 (pick:face (ray (position 0 0 0)
    (gvector 0 –1 0)) 1))
;; face2
; Define a face to put instruction on
(define face_instr (pick:face (ray (position 0 0 0)
    (gvector 1 0 0)) 1))
;; face_instr
; Put a cap instruction (action = 2) on the face
(abl:set–instruction face_instr 2)
;; ”cap”
; Do blending
(abl:ent–ent–blend face1 face2 rad (position 0 0 0) #t)
;; #[entity 2 1]
; Fix the blend
(blend:fix (entity 2))
;; #[entity 2 1]
; OUTPUT Result
```

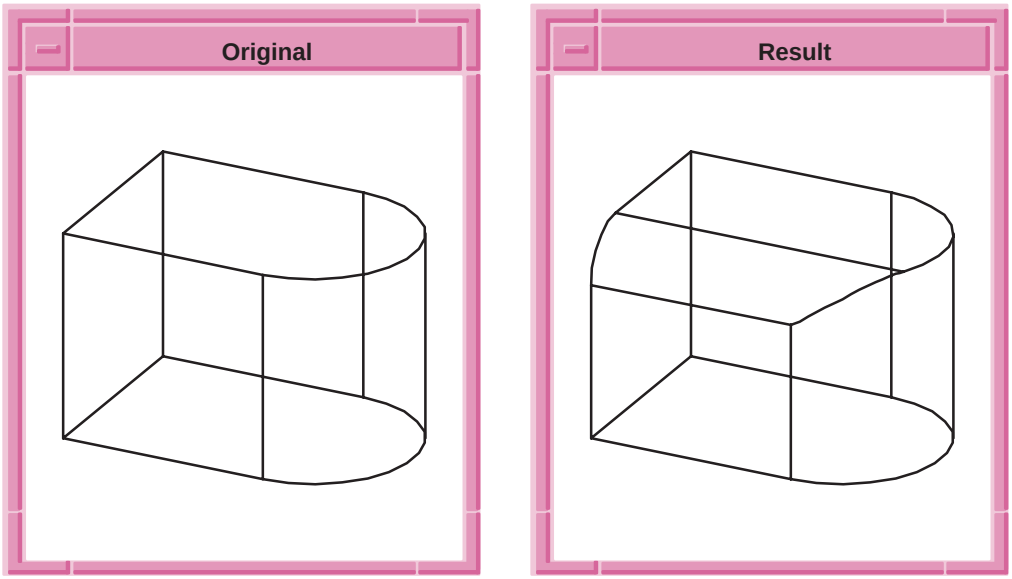


Figure 2-10. Forcing a Blend to Cap

Compare the following example with the previous example. A cap instruction is placed on the first tangent edge to prevent the blend from rolling onto it.

Scheme Example

```
(define after (solid:block (position –20 –20 –2.5)
    (position 20 20 2.5)))
(define tube1 (solid:cylinder (position 0 0 –2.5)
    (position 0 0 2.5) 20))
(entity:transform tube1
    (transform:translation (gvector 20 0 0)))
(solid:unite after tube1)
; OUTPUT Original
; select edge for placing blend instruction
(define edge1 (pick:edge (ray (position 0 0 –2.5)
    (gvector 1 0 0))))
; select the face and edge for blending
(define face1 (pick:face (ray (position 0 0 0)
    (gvector 0 0 1)) 1))
(define edge2 (pick:edge (ray (position 0 –20 0)
    (gvector 0 0 –1))))
; create the vradius
(define rad1 (abl:const–rad 10))
; setting capping instruction on edge1
(abl:set–instruction edge1 2)
; create the entity–entity blend
(abl:ent–ent–blend face1 edge2 rad1 (position 0 0 0) #t)
; fix the blend
(blend:fix after)
; OUTPUT Result
```

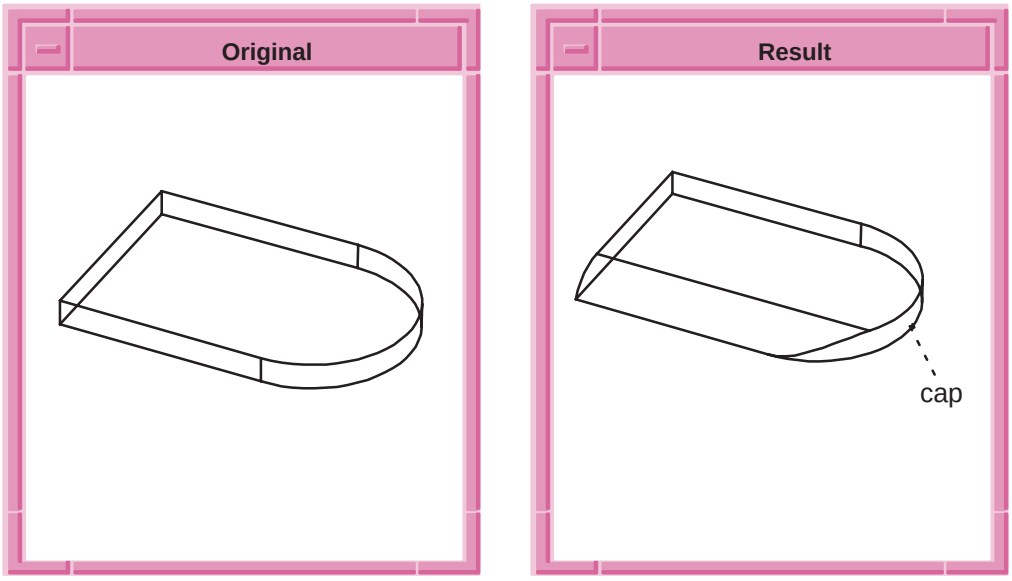


Figure 2-11. Capping at First Tangent Edge

Forcing a Blend to Roll On

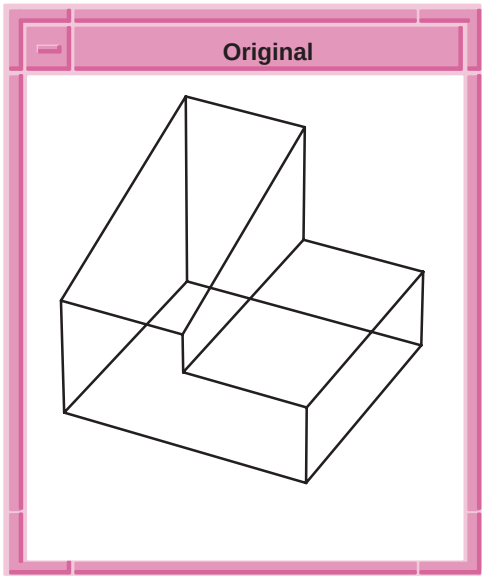
Topic: Blending

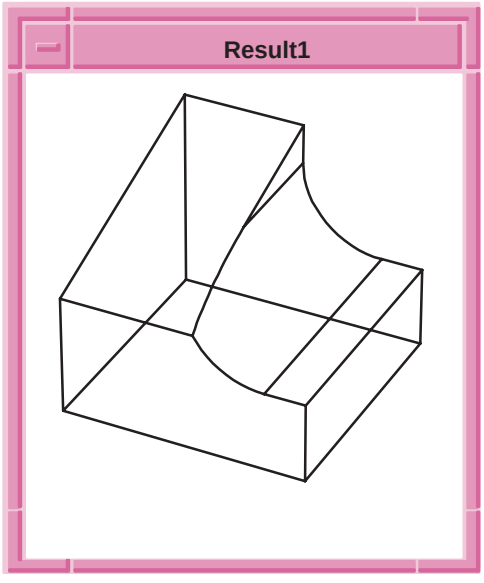
Whenever a blend running along a face runs into an edge, it will roll onto any smooth face it finds there, otherwise it will cap. Sometimes it is desirable to roll onto the edge of the face rather than to cap the blend.

Compare the two results in the following example. In the default case, the face–face blend is end capped. The addition of the roll on instruction to the edge in the second case causes the face–face blend to become an edge–face blend. It is this edge–face blend which is subsequently end capped.

Scheme Example

```
(define after (solid:block (position –30 –30 –10)
    (position 30 30 10)))
(entity:transform after
    (transform:translation (gvector 0 0 –10)))
(define wireframe (wire–body:points (list (position –30 –30 0)
    (position –30 –30 10) (position –30 30 30)
    (position –30 30 0) (position –30 –30 0))))
(sheet:cover–wires wireframe)
(sheet:cover wireframe)
(define fac (pick:face (ray (position 0 0 10)
    (gvector –1 0 0)) 1))
(define body1 (sweep:law fac 30))
(solid:unite after body1)
(define face1 (pick:face (ray (position 1 0 1)
    (gvector –1 0 0)) 1))
(define face2 (pick:face (ray (position 1 0 1)
    (gvector 0 0 –1)) 1))
(define rad1 (abl:const–rad 20))
; OUTPUT Original
(define step2 (entity:copy after))
(abl:ent–ent–blend face1 face2 rad1 (position 20 0 20) #f)
(blend:fix after)
; OUTPUT Result1
(entity:delete after)
(define face1 (pick:face (ray (position 1 0 1)
    (gvector –1 0 0)) 1))
(define face2 (pick:face (ray (position 1 0 1)
    (gvector 0 0 –1)) 1))
(define edge1 (pick:edge (ray (position 0 0 1)
    (gvector 0 0 1))))
(abl:set–instruction edge1 1)
(abl:ent–ent–blend face1 face2 rad1 (position 20 0 20) #f)
(blend:fix step2)
; OUTPUT Result2
```





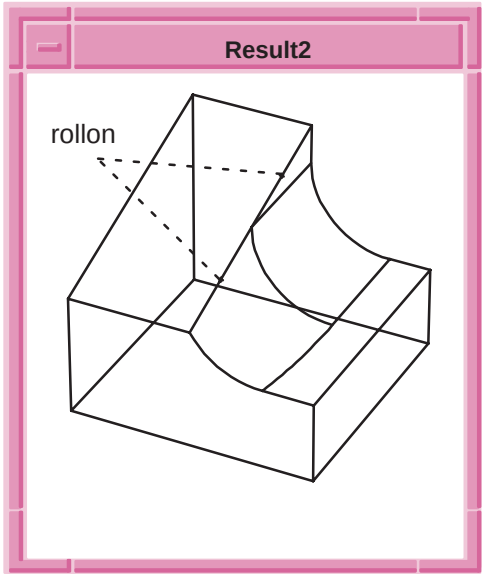


Figure 2-12. Forcing End Cap on Edge–Face

Scheme Example

```
(define after (solid:block (position –40 –40 –10)
    (position 40 40 10)))
(entity:transform after
    (transform:translation (gvector 0 0 –10)))
(define cube1 (solid:block (position –20 –20 –10)
    (position 20 20 10)))
(solid:unite after cube1)
; OUTPUT Original
; select the vertex for blend instruction
(define v1 (pick:vertex (ray (position 0 0 10)
    (gvector 1 –1 0))))
(define edge1 (pick:edge (ray (position 0 0 10)
    (gvector 0 –1 0))))
(define face1 (pick:face (ray (position 0 –30 10)
    (gvector 0 0 –1)) 1))
; create the vradius
(define rad1 (abl:const–rad 20))
; set the rollon instruction at the vertex
(abl:set–instruction v1 1)
; create the blend
(abl:ent–ent–blend edge1 face1 rad1 (position 0 –30 10) #f)
; fix the blend
(blend:fix after)
; OUTPUT Result
```

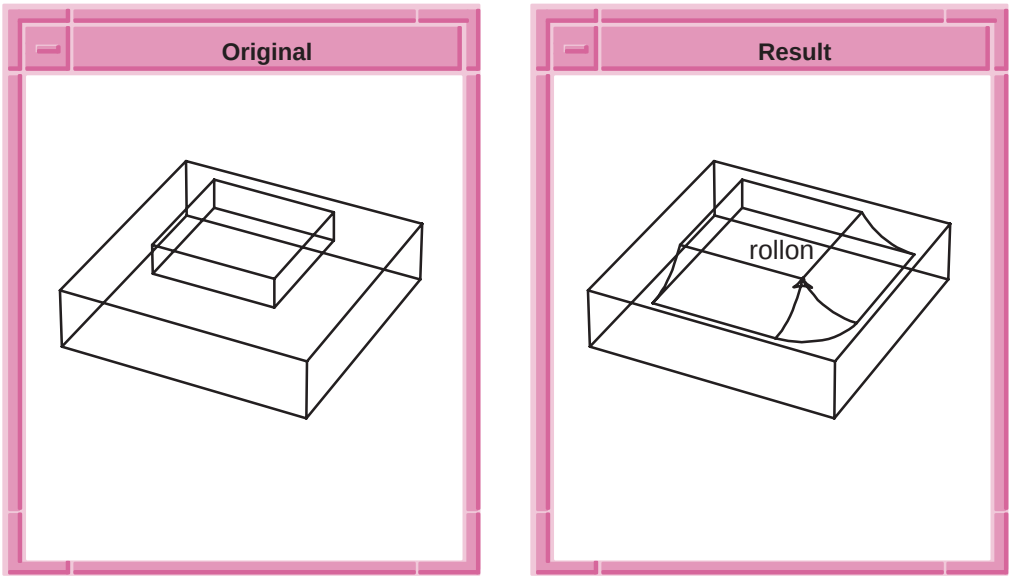


Figure 2-13. Face–Vertex Blend

Specifying Instruction Positions

Topic: Blending

At one end of the initial face–face blend in the following example, we wish to roll onto the edge (rather than cap as usual), so a roll on instruction is placed. However, at the other end we wish to cap as usual, so we must place a cap instruction to override the previously established roll on instruction that would now apply. Since there are two instructions for the same edge, we must distinguish them by associating positions for the instructions. Whenever there is more than one instruction on a blend, the nearest instruction on the entity in question applies.

Scheme Example

```
(define after (solid:block (position –40 –40 –20)
    (position 40 40 20)))
(define tube1 (solid:cylinder (position 0 0 –20)
    (position 0 0 20) 30))
(entity:transform tube1
    (transform:rotation (position 0 0 0) (gvector 1 0 0) 90))
(entity:transform tube1
    (transform:translation (gvector 0 0 15)))
(solid:unite after tube1)
; OUTPUT Original
; select the edge for blend instruction
(define edge1 (pick:edge (ray (position 0 0 45)
    (gvector 0 –1 0))))
; select the faces for blending
(define face1 (pick:face (ray (position 0 0 25)
    (gvector 0 –1 0)) 1))
(define face2 (pick:face (ray (position 0 –30 0)
    (gvector 0 0 1)) 1))
; create the vradius
(define rad1 (abl:const–rad 5))
; apply rollon instruction to edge1 at specific position
(abl:set–instruction edge1 1 (position 45 0 0))
; apply capping instruction to edge1 at specific position
(abl:set–instruction edge1 2 (position –45 0 0))
; create the blend
(abl:ent–ent–blend face1 face2 rad1 (position 0 –30 30) #f)
; fix the blend
(blend:fix after)
; OUTPUT Result
```

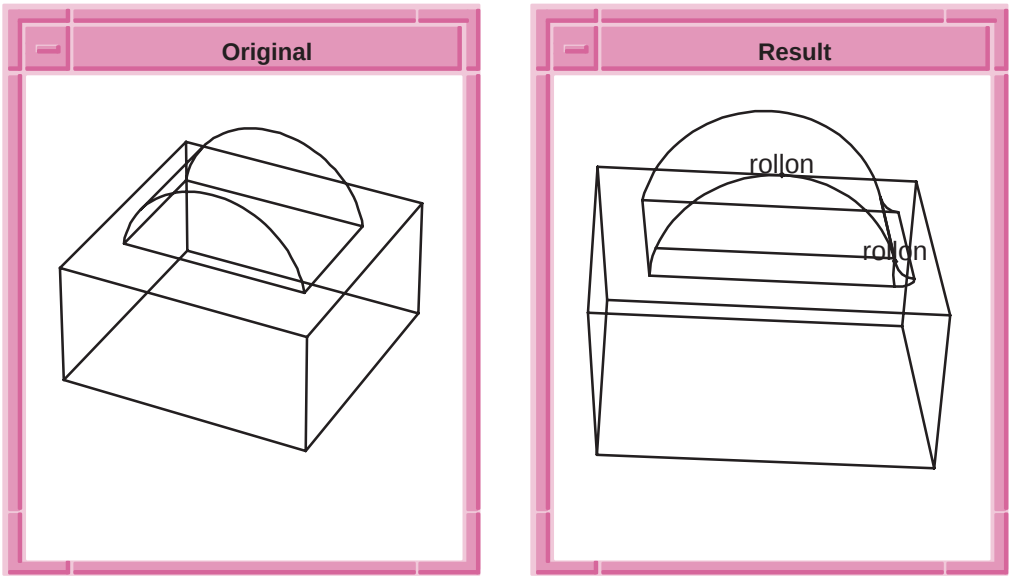


Figure 2-14. Instructions at Specified Positions

Edge–Face Sequence

Topic: Blending

The following is an example of a blend of an edge–face sequence. Unfortunately, the rolling ball falls off the face as it follows the series of tangent edges. With no roll on instruction, the blend starts capping, giving rise to a side cap and then an end cap. The effect of the roll on instruction, terminating with two edge–edge blend faces, is probably more desirable.

Scheme Example

```
(define after (solid:block (position –40 –40 –10)
    (position 40 40 10)))
(entity:transform after (transform:translation
    (gvector 0 0 –10)))
(define cube1 (solid:block (position –30 –35 –10)
    (position 30 35 10)))
(entity:transform cube1
    (transform:translation (gvector 10 5 0)))
(solid:unite after cube1)
(solid:blend–edges (pick:edge  (ray (position –20 0 5)
    (gvector 0 –1 0))) 20)
; OUTPUT Original
; select face and edge for blending
(define face1 (pick:face (ray (position –30 0 10)
    (gvector 0 0 –1)) 1))
(define edge1 (pick:edge (ray (position 0 0 10)
    (gvector –1 0 0))))
; create the vradius
(define rad1 (abl:const–rad 20))
; make a copy of the original entity
(define step1 (entity:copy after))
; create the blend
(abl:ent–ent–blend face1 edge1 rad1 (position –30 0 10) #f)
(blend:fix after)
; OUTPUT Result1
(entity:delete after)
(define face1 (pick:face (ray (position –30 0 10)
    (gvector 0 0 –1)) 1))
(define edge1 (pick:edge (ray (position –30 0 0)
    (gvector 0 –1 0))))
(define edge2 (pick:edge (ray (position 0 0 10)
    (gvector –1 0 0))))
(abl:set–instruction edge1 1)
(abl:ent–ent–blend face1 edge2 rad1 (position –30 0 10) #f)
(blend:fix step1)
; OUTPUT Result2
```

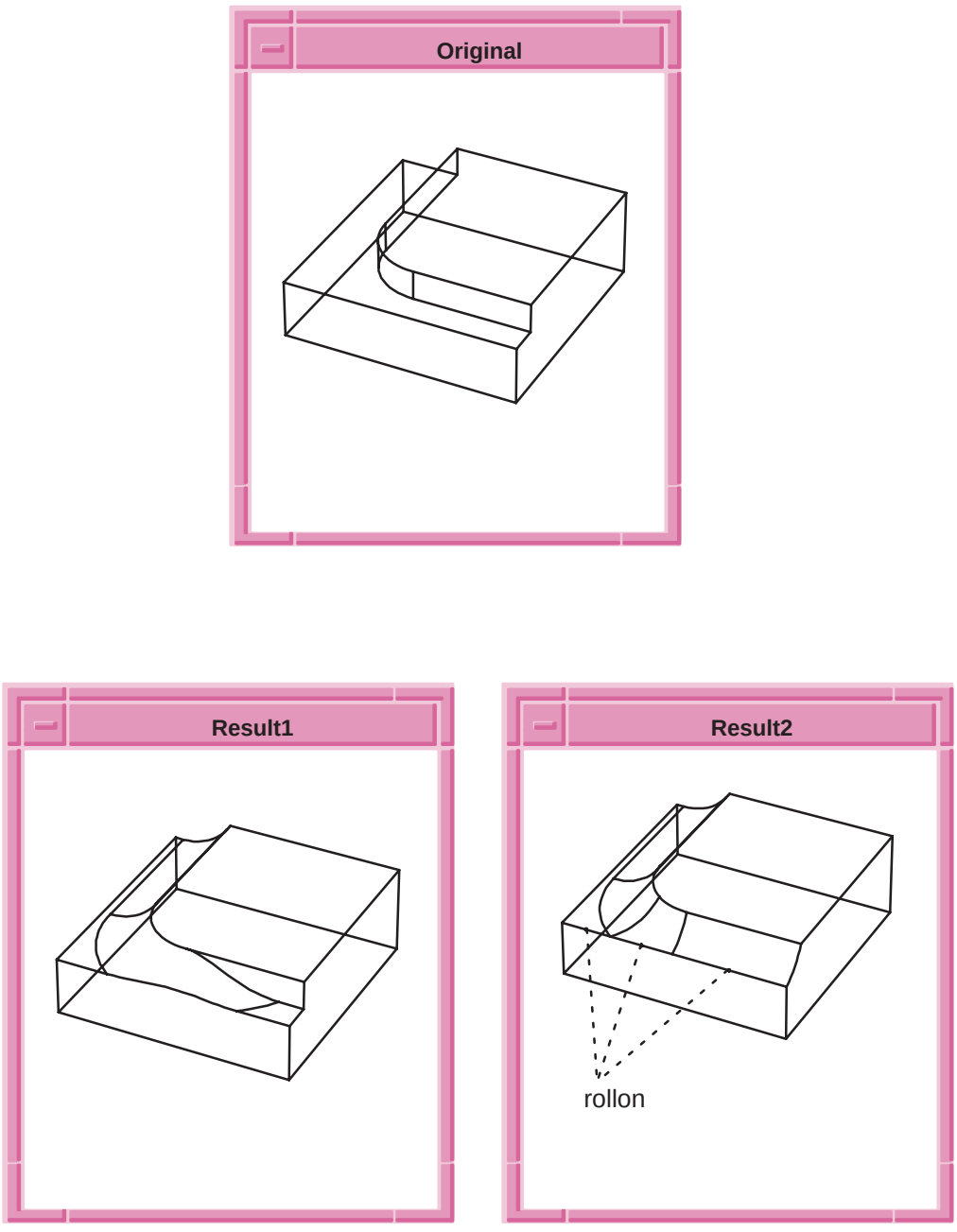


Figure 2-15. Edge–Face Sequence

Tapering Blends

Topic: Blending

Sometimes it is desirable for the ends of a blend to taper out, rather than get capped. In the following example, the blend will only taper out to nothing if the edge which meets the bottom face does so tangentially. Otherwise, the rolling ball would roll over the edge onto the next face. Although only one roll on instruction is required, all the edges in the series of tangent edges used could have been marked with roll on. This is useful if one does not know which of the tangent edges will actually be intercepted by the face–face blend.

Scheme Example

```
(define tube1 (solid:cylinder (position 0 0 –30)
    (position 0 0 30) 30))
(entity:transform tube1
    (transform:rotation (position 0 0 0) (gvector 0 1 0) 90))
(entity:transform tube1
    (transform:translation (gvector 0 –15 30)))
(define cube1 (solid:block (position –30 –7.5 –10)
    (position 30 7.5 10)))
(entity:transform cube1
    (transform:translation (gvector 0 –7.5 10)))
(solid:subtract cube1 tube1)
(define tube2 (solid:cylinder (position 0 0 –30)
    (position 0 0 30) 30))
(entity:transform tube2
    (transform:rotation (position 0 0 0) (gvector 0 1 0) 90))
(entity:transform tube2
    (transform:translation (gvector 0 15 –30)))
(entity:transform tube2
    (transform:translation (gvector 0 0 8.0384757729336833)))
(define cube2 (solid:block (position –30 –7.5 –10)
    (position 30 7.5 10)))
(entity:transform cube2
    (transform:translation (gvector 0 7.5 10)))
(solid:intersect cube2 tube2)
(define after (solid:block (position –30 –5 –4.0192378864668417)
    (position 30 5 4.0192378864668417)))
(entity:transform after
    (transform:translation (gvector 0 20 4.0192378864668417)))
(solid:unite cube2 cube1)
(solid:unite after cube2)
(define cube3 (solid:block (position –40 –30 –2.5)
    (position 40 30 2.5)))
(entity:transform cube3
    (transform:translation (gvector 10 –5 –2.5)))
```

```
(solid:unite after cube3)
; OUTPUT Original
; select edge for rollon instruction
(define edge1 (pick:edge (ray (position 30 5 1)
    (gvector 0 0 1))))
; select faces for blending
(define face1 (pick:face (ray (position 40 0 10)
    (gvector 0 0 –1)) 1))
(define face2 (pick:face (ray (position 40 0 1)
    (gvector –1 0 0)) 1))
; create the vradius
(define rad1 (abl:const–rad 7))
; set the blend instruction
(abl:set–instruction edge1 1)
; create the blend
(abl:ent–ent–blend face1 face2 rad1 (position 40 10 10) #f)
(blend:fix after)
; OUTPUT Result
```

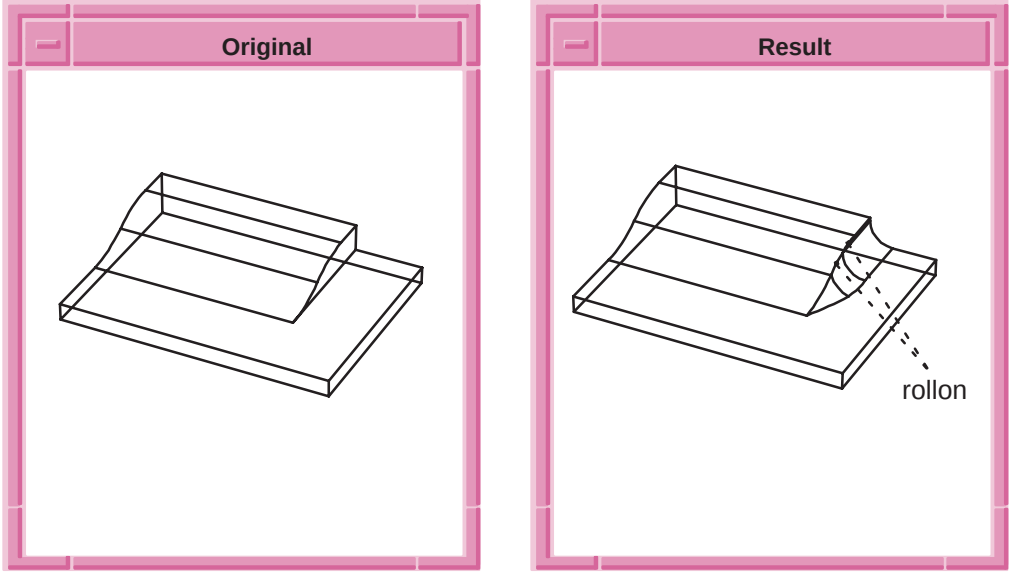


Figure 2-16. Tapering Blends

A tapering end to a blend is more difficult if the edge down which the blend rolls does not approach the face tangentially. In some cases, an edge–edge blend can provide a solution to this problem. In this example, the face–face blend first becomes edge–face and the final triangle of blend surface is edge–edge. The second roll on instruction is important, as the edge–face surface is otherwise difficult to end cap.

Scheme Example

```
(define after (solid:block (position –30 –30 –15)
    (position 30 30 15)))
(entity:transform after
    (transform:translation (gvector 0 0 –15)))
(define wb (wire–body:points (list (position 0 –30 0)
    (position 0 30 0) (position 0 30 30) (position 0 –30 0))))
(sheet:cover–wires wb)
(sheet:cover wb)
(define fac (pick:face (ray (position –1 0 1)
    (gvector 1 0 0)) 1))
(define body1 (sweep:law fac 30))
(solid:unite after body1)
; OUTPUT Original
; select edges for blend instruction
(define edge1 (pick:edge (ray (position 0 0 1)
    (gvector 0 0 1))))
(define edge2 (pick:edge (ray (position 1 0 0)
    (gvector 0 –1 0))))
; select faces for blending
(define face1 (pick:face (ray (position 1 0 1)
    (gvector –1 0 0)) 1))
(define face2 (pick:face (ray (position 1 0 1)
    (gvector 0 0 –1)) 1))
; create the vradius
(define rad1 (abl:const–rad 20))
; set rollon instruction on edge1 and edge2
(abl:set–instruction edge1 1)
(abl:set–instruction edge2 1)
; create the blend
(abl:ent–ent–blend face1 face2 rad1 (position 0 0 0) #f)
; fix the blend
(blend:fix after)
; OUTPUT Result
```

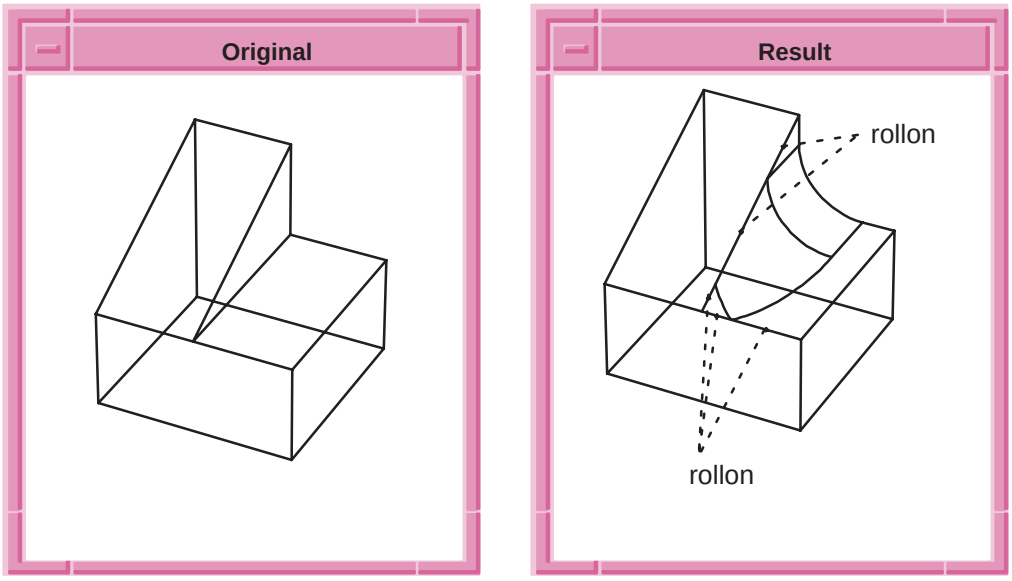


Figure 2-17. Tapering with Edge–Edge Blend

In this example, tapering the blend away with an edge–face blend is a reasonable solution to an otherwise difficult end capping problem. The straight edges approach the face tangentially, so the blend tapers to points.

Scheme Example

```
(define after (solid:block (position –40 –40 –20)
    (position 40 40 20)))
(entity:transform after
    (transform:translation (gvector 0 0 –20)))
(define tube1 (solid:cylinder (position 0 0 –40)
    (position 0 0 40) 40))
(solid:unite after tube1)
; OUTPUT Original
; select edges for blend instruction
(define edge1 (pick:edge (ray (position 38 –38 0)
    (gvector 1 0 0))))
(define edge2 (pick:edge (ray (position 38 –38 0)
    (gvector 0 –1 0))))
; select faces for blending
(define face1 (pick:face (ray (position 38 –38 10)
    (gvector –1 1 0)) 1))
(define face2 (pick:face (ray (position 38 –38 –1)
    (gvector 0 0 1)) 1))
; create the vradius
(define rad1 (abl:const–rad 5))
; set the rollon instruction for the edges
(abl:set–instruction edge1 1)
(abl:set–instruction edge2 1)
; create the blend
(abl:ent–ent–blend face1 face2 rad1 (position 40 –40 0) #f)
; fix the blend
(blend:fix after)
; OUTPUT Result
```

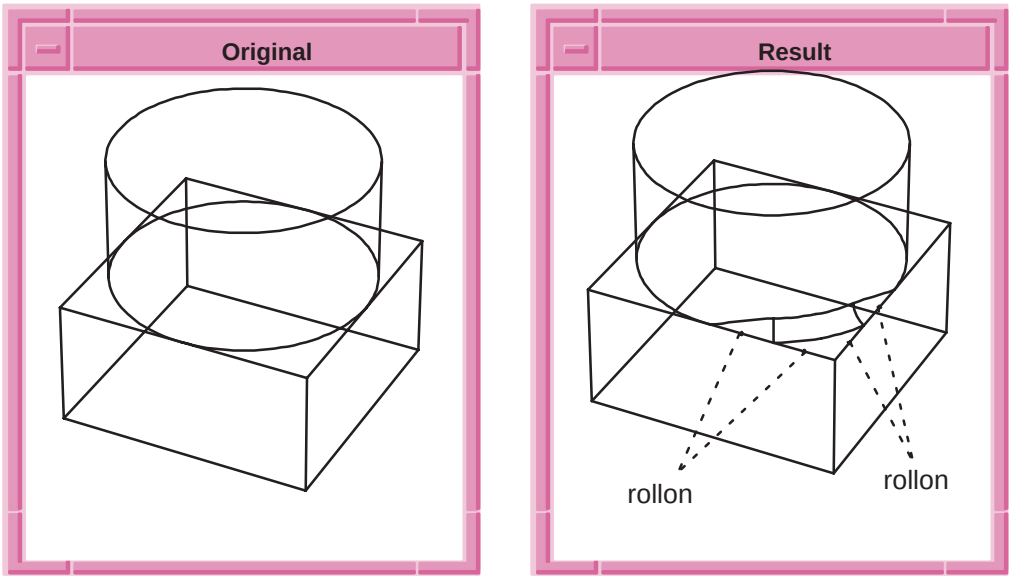


Figure 2-18. Tapering Away with Edge–Face Blend

Blends at Complex Vertices

Topic: Blending

Entity–entity blends provide a means of rolling around or over edges and vertices to join together into a single sequence. This example can be compared with what one would obtain with ordinary (i.e., non entity–entity) edge blends, mitered together. In this case the miter would actually extend so far as to run off the bottom face.

Scheme Example

```
(define after (solid:block (position –50 –50 –10)
    (position 50 50 10)))
(define body1 (solid:pyramid 60  40 40 3))
(entity:transform after (transform:translation
    (gvector 0 0 –20)))
(solid:unite after body1)
; OUTPUT Original
; select edge for blend instruction
(define edge1 (pick:edge (ray (position 0 0 0)
    (gvector –1 0 1))))
; select faces for blending
(define face1 (pick:face (ray (position 0 0 0)
    (gvector 0 1 0)) 1))
(define face2 (pick:face (ray (position 30 0 0)
    (gvector 0 0 –1)) 1))
; create the vradius
(define rad1 (abl:const–rad 20))
; set the rollon instruction
(abl:set–instruction edge1 1)
; create the blend
(abl:ent–ent–blend face1 face2 rad1 (position 30 0 –10) #f)
; fix the blend
(blend:fix after)
; OUTPUT Result
```

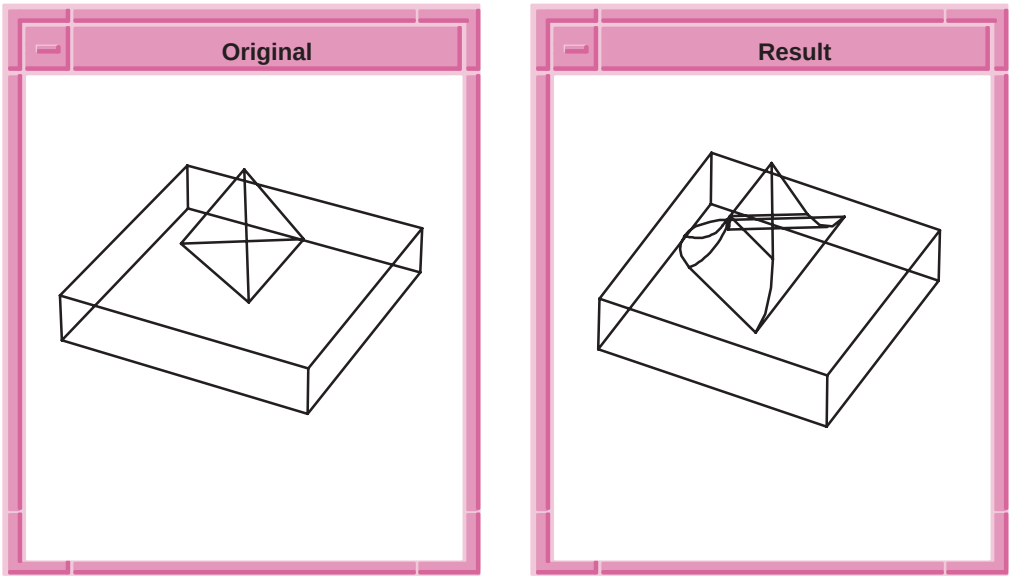


Figure 2-19. Joining Edges and Vertices into Single Sequence

This is another example in which the rolling ball rolls around a complex vertex. As it works its way around, two extra edge–face and one extra face–face blends are produced. Both roll on instructions are required to guide the ball around.

Scheme Example

```
(define after (solid:block (position –40 –40 –40)
    (position 40 40 40)))
(define tube1 (solid:cylinder (position 0 0 –50)
    (position 0 0 50) 100))
(entity:transform tube1
    (transform:translation (gvector 0 65 0)))
; imprint the intersection curves on the body
(solid:imprint after tube1)
(entity:delete tube1)
(define tube2 (solid:cylinder (position 0 0 –50)
    (position 0 0 50) 100))
(entity:transform tube2
    (transform:rotation (position 0 0 0) (gvector 1 0 0) 90))
(entity:transform tube2
    (transform:translation (gvector 0 0 –65)))
; imprint the intersection curves on the body
(solid:imprint after tube2)
(entity:delete tube2)
; OUTPUT Original
; select the imprinted edges for blending
(define edge1 (pick:edge (ray (position 0 0 40)
    (gvector 0 –1 0))))
(define edge2 (pick:edge (ray (position 0 –40 0)
    (gvector 0 0 1))))
; create the vradius
(define rad1 (abl:const–rad 20))
; create the blend
(abl:ent–ent–blend edge1 edge2 rad1 (position 0 –20 20) #t)
; fix the blend
(blend:fix after)
; OUTPUT Result
```

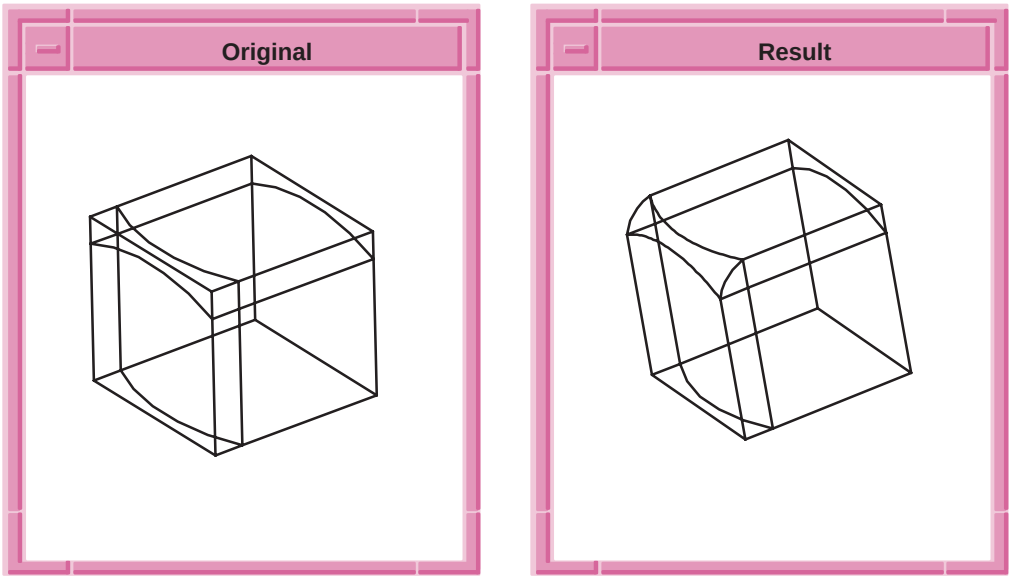


Figure 2-20. Using Spring Curves

High Degree of Behavior Control

Topic: Blending

Advanced blending offers a high degree of control over where and how to roll onto new entities or to cap. If there is more than one location at which side caps could be produced, a face–face blend can be used to roll on at some, but not all, of these places. In the following example there are four places at which side caps could be formed, but we wish to roll on at just one of these locations. Therefore, we use the roll on instruction to force the rolling ball to roll along just one intercepted edge.

Scheme Example

```
(define after (solid:block (position –30 –30 –15)
    (position 30 30 15)))
(entity:transform after
    (transform:translation (gvector 0 0 –15)))
(define tube1 (solid:cylinder (position 0 0 –30)
    (position 0 0 30) 20))
(solid:unite after tube1)
; OUTPUT Original
; select edge for blend instruction
(define edge1 (pick:edge (ray (position 25 0 0)
    (gvector 1 0 0))))
; select faces for blending
(define face1 (pick:face (ray (position 25 0 1)
    (gvector 0 0 –1)) 1))
(define face2 (pick:face (ray (position 0 0 5)
    (gvector 1 0 0)) 1))
; create the vradius
(define rad1 (abl:const–rad 15))
; create the instruction on the edge
(abl:set–instruction edge1 1)
; create the blend
(abl:ent–ent–blend face1 face2 rad1 (position 30 0 10) #f)
; fix the blend
(blend:fix after)
; OUTPUT Result
```

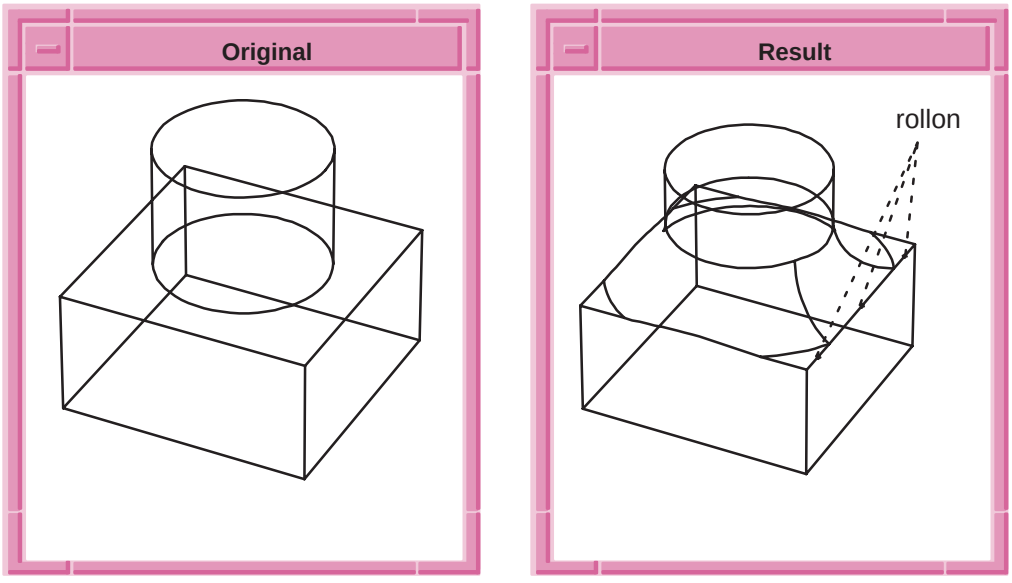


Figure 2-21. Controlling Capping