Chapter 3.
Scheme Extensions

Topic: Ignore

Scheme is a public domain programming language, based on the LISP language, that uses an interpreter to run commands. ACIS provides extensions (written in C++) to the native Scheme language that can be used by an application to interact with ACIS through its Scheme Interpreter. The C++ source files for ACIS Scheme extensions are provided with the product. *Spatial*'s Scheme based demonstration application, Scheme ACIS Interface Driver Extension (Scheme AIDE), also uses these Scheme extensions and the Scheme Interpreter. Refer to the *3D ACIS Online Help User's Guide* for a description of the fields in the reference template.

abl:abh–edge–offset

Scheme Extension: Blending

Action: Creates a series of offset surfaces from an edge sequence.

Filename: abl/abl_scm/abl_scm.cxx

APIs: api_abh_edge_offset

Syntax:

```
(abl:abh–edge–offset blank radius limit–list
    [acis–opts])
```

Arg Types:

blank	body
radius	real
limit–list	edge | (edge ...)
acis–opts	acis–options

Returns: body

Errors: None

Description: Makes a long sheet body of cylinder/torus/pipe surfaces of the given radius (i.e., a series of pipe-like surfaces) around a sequence of smooth edges . The tubular sheet body consisting of the offset edge surfaces is returned.

If the edge sequence is open (which would also, in this sense, include a closed but not smoothly joined sequence), cylindrical extension surfaces are glued onto those open ends to ensure the sheet body exceeds the blank body.

blank is a tubular sheet body .

radius is radius of the sheet body of cylinder/torus/pipe .

limit–list is a sequence of smooth edges around which tubular sheet body is built.

acis–opts contains journaling and versioning information.

Limitations: The new sheet body created and returned is intended only for use in the subsequent construction of spring curves (refer to api_abh_slice and api_abh_imprint API functions), as it may not be a fully valid body.

Example:

```
; abl:abh-edge-offset
; Create a block
(define a (solid:block
    (position -25 -25 -25) (position 25 25 25)))
;; a
(define b (bool:unite a (solid:block
    (position -25 -25 -25) (position -5 25 45))))
;; b
(define e1 (list (list-ref (entity:edges a) 8)))
;; e1
(entity:set-color e1 3)
;; ()
; OUTPUT Original

(define n (abl:abh-edge-offset a 10 e1))
;; n
(entity:set-color e1 3)
;; ()
; OUTPUT Result
```

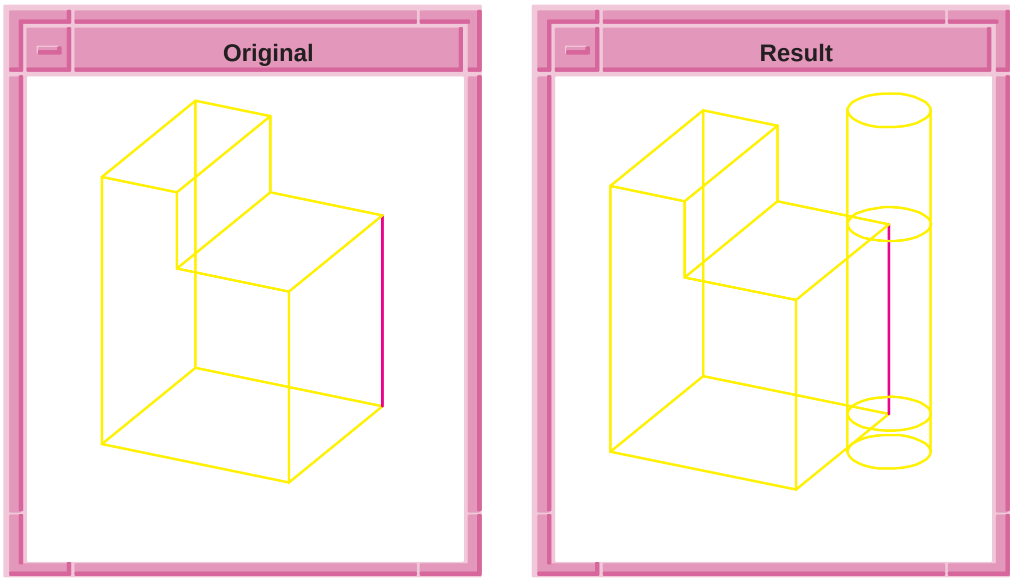


Figure 3-1. abl:abh–edge–offset

abl:abh–edge–project

Scheme Extension: Blending

Action: Creates an offset edge at a specified distance on the body parallel to a given edge.

Filename: abl/abl_scm/abl_scm.cxx

APIs: api_abh_edge_project

Syntax:
```
(abl:abh-edge-project blank edge distance direction
    left sequence [acis-opts])
```

Arg Types:

blank	body
edge	edge
distance	real
direction	gvector
left	boolean
sequence	boolean
acis–opts	acis–options

Returns: boolean

Errors: None

Description: Takes an edge on the body and imprints the edge parallel to itself on the body by the given distance. The direction of the imprint is provided by the logical left, which is the direction with respect to the underlying edge and the outward pointing vector *v*. The edge is first extended in both directions, projected upward on a plane normal to vector *v*, offset, and then reprojected on the body. If the logical sequence is specified, the smooth sequence of connected edges is handled. The modified body is returned.

blank is the body.

edge is an edge on the body.

distance is length by which offset edge is created.

direction is the direction with respect to the underlying edge and vector v.

left is a logical left of the direction.

sequence is a logical sequence of connected edges.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; abl:abh-edge-project
; Create a block
(define b
    (solid:block (position 0 0 0) (position 1 1 1)))
;; b
(zoom-all)
;; #[view 263342]
(define e (list-ref (entity:edges b) 3))
;; e
(entity:set-color e 1)
;; ()
; OUTPUT Original

(define e2 (abl:abh-edge-project b e .1
    (gvector -3 5 1) #f #f))
;; e2
; OUTPUT Result
```

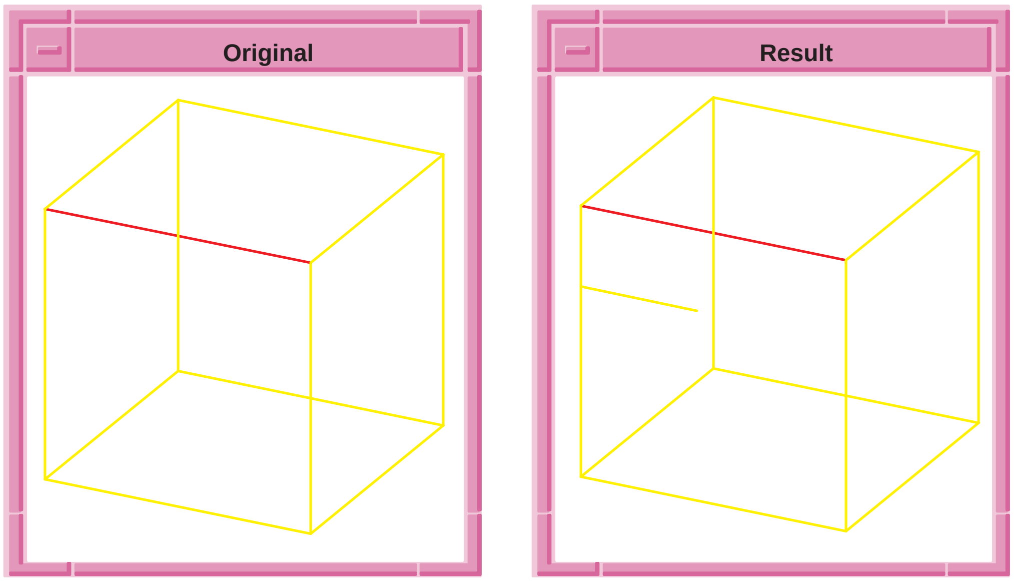


Figure 3-2. abl:abh–edge–project

abl:abh–imprint

Scheme Extension:	Blending
Action:	Creates a series of offset surfaces from an edge sequence.
Filename:	abl/abl_scm/abl_scm.cxx
APIs:	api_abh_imprint
Syntax:	`(abl:abh–imprint int–graph edge–list [acis–opts])`

Arg Types:	int–graph	body
	edge–list	edge | (edge ...)
	acis–opts	acis–options

Returns:	boolean
Errors:	None
Description:	This function performs a ”partial imprint.” It will imprint only the given edges of the intersection graph on the bodies from which the intersection graph originated.

This extension takes the listed edges of the intersection wire and imprints them on the two bodies. The intersection graph should have been made by calling api_abh_slice, and both of the original bodies must still exist.

int–graph is a body from which the intersection graph originated.

edge–list is a listed edges of the intersection wire.

acis–opts contains journaling and versioning information.

Limitations: The intersection graph should be the result of a call to api_abh_slice. The edge_list should contain edges of that intersection wire.

Example:

```
; abl:abh-imprint
; Create a block
(define a (solid:block
    (position –25 –25 –25) (position 25 25 25)))
;; a
(define b (bool:unite a (solid:block
    (position –25 –25 –25) (position –5 25 45))))
;; b
(define e1 (list (list-ref (entity:edges a) 8)))
;; e1
(entity:set–color e1 3)
;; ()
; OUTPUT Original

(define n (abl:abh–edge–offset a 10 e1))
;; n
(define n1 (abl:abh–slice a n))
;; n1
(define e2 (list–ref (entity:edges n1)2))
;; e2
(entity:set–color e1 1)
;; ()
; OUTPUT Intermediate

(abl:abh–imprint n1 e2)
;; #t
(entity:delete n1)
;; ()
(entity:delete n)
;; ()
(render:rebuild)
;; ()
; OUTPUT Result
```

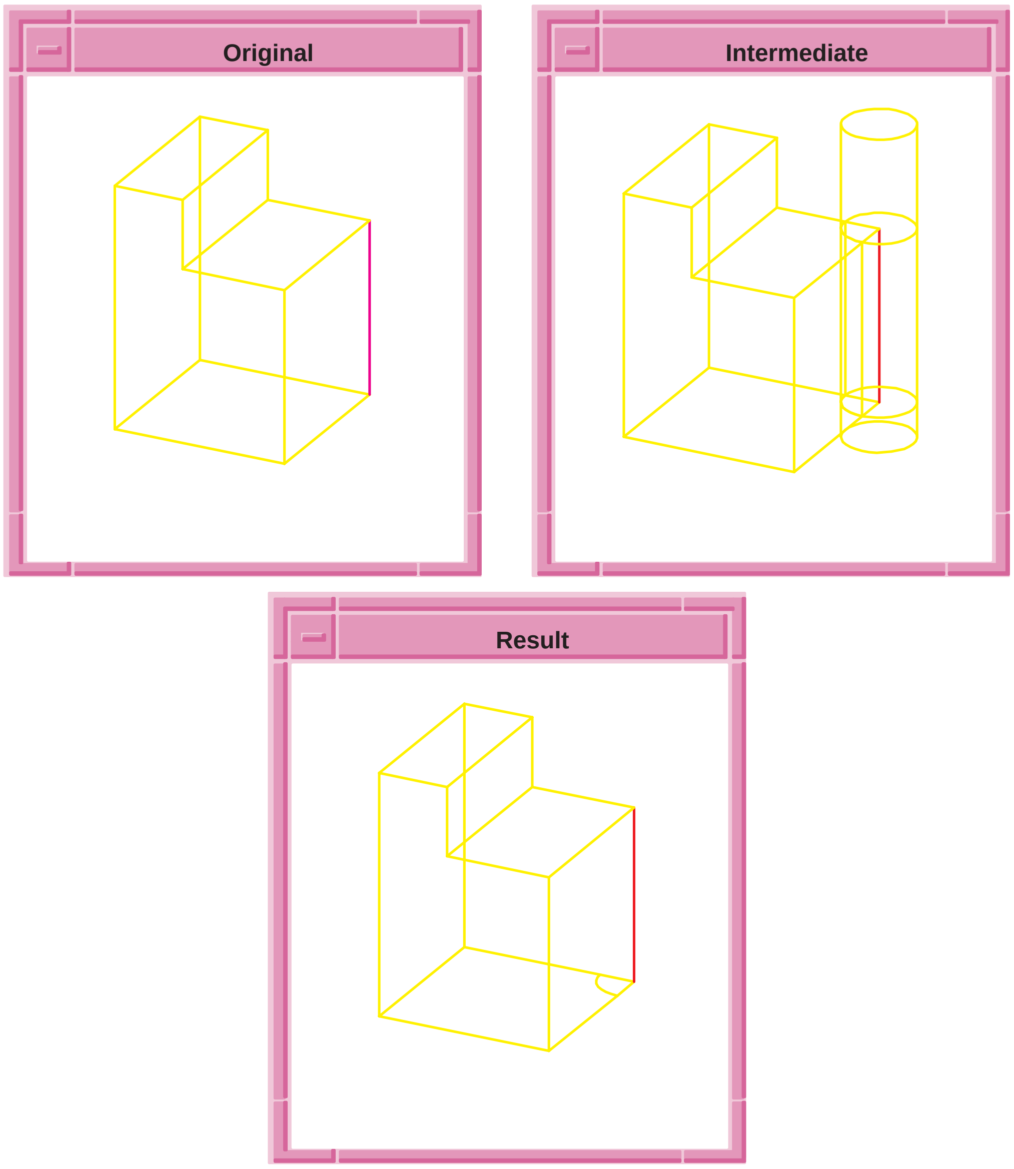


Figure 3-3. abl:abh–imprint

abl:abh–slice

Scheme Extension: Blending

Action:	Creates the intersection wire between two bodies, but does not delete the intersection graph part of it.
Filename:	abl/abl_scm/abl_scm.cxx
APIs:	api_abh_slice
Syntax:	`(`**`abl:abh–slice`**` blank tool [acis–opts])`

Arg Types:

blank	body
tool	body
acis–opts	acis–options

Returns: body

Errors: None

Description: This Scheme extension slices two given bodies and returns a special form of wire object in which each edge has two coedges (rather than one, as in the case of wires made using api_make_wire or api_make_kwire). This is similar to the "normal" slice function, api_slice, except that all the intersection graph attributes computed by the stage one Boolean operations are left in the wire object. The resulting intersection wire object is intended for use by api_abh_imprint.

blank is a input body for slicing operation.

tool is a input body for slicing operation.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; abl:abh–slice
; Create a block
(define a (solid:block
    (position –25 –25 –25) (position 25 25 25)))
;; a
(define b (bool:unite a (solid:block
    (position –25 –25 –25) (position –5 25 45))))
;; b
(define e1 (list (list–ref (entity:edges a) 8)))
;; e1
(entity:set–color e1 3)
;; ()
; OUTPUT Original
```

```
(define n (abl:abh-edge-offset a 10 e1))
;; n
(define n1 (abl:abh-slice a n))
;; n1
(entity:set-color n1 1)
;; ()
; OUTPUT Result
```

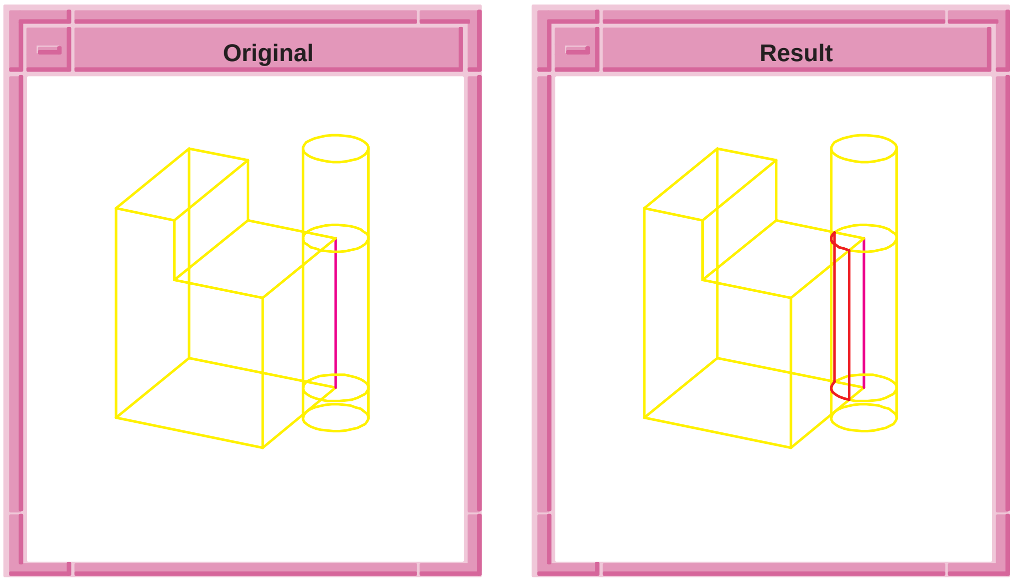


Figure 3-4. abl:abh–slice

abl:const–rad

Scheme Extension: Blending

Action: Creates a vradius object for setting advanced blending attributes for a blend with a constant radius.

Filename: abl/abl_scm/abl_scm.cxx

APIs: api_make_radius_constant

Syntax: `(`**`abl:const-rad`**` radius [acis-opts])`

Arg Types:

radius	real
acis–opts	acis–options

Returns: vradius

Errors: None

Description: This extension creates an object of vradius data type which corresponds to class var_rad_two_ends. The start and end radius values from each end of the blend are the same. In other words, a constant radius along the length of the blend edge is created and is similar to the standard blending constant radius blend.

radius is radius of an object.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; abl:const–rad
; Create a block
(define block1
    (solid:block (position –20 –20 –20)
    (position 15 20 25)))
;; block1
; OUTPUT Original

; Create a vradius
(define rad1 (abl:const–rad 20))
;; rad1
; Define an edge
(define edges1 (entity:edges block1))
;; edges1
; Put a blend attribute on the selected edge
(define blend (abl:edge–blend (car edges1) rad1))
;; blend
; Complete the blend
(blend:fix (car edges1))
;; #t
; OUTPUT Result
```

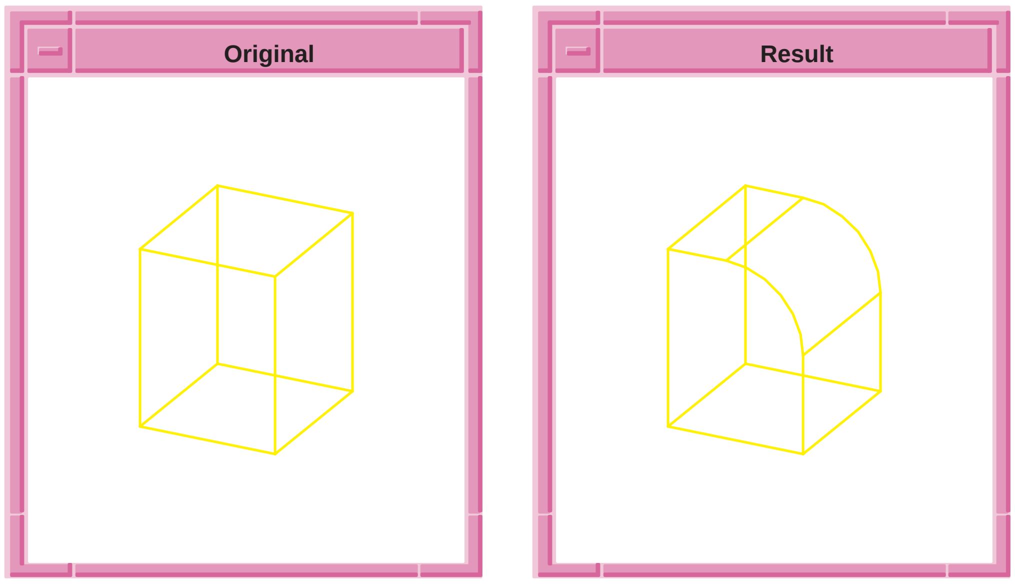


Figure 3-5. abl:const–rad

abl:create–vradius–wire

Scheme Extension:	Blending
Action:	Creates a wire body corresponding to a particular vradius object (reparameterized).
Filename:	abl/abl_scm/abl_scm.cxx
APIs:	api_make_wire
Syntax:	(**abl:create–vradius–wire** svradius min–param max–param max–iteration)
Arg Types:	svradius — vradius
	min–param — real
	max–param — real
	max–iteration — integer
Returns:	wire–body
Errors:	None

Description: Copies the vradius object, uncalibrates the copy, reparameterizes the copy between “min–param” and “max–param”, calibrates the copy, evaluates the vradius copy “max–iteration” times between “min–param” and “max–param”, uncalibrates the copy, and then returns a wire body with radii corresponding to the evaluation values.

svradius is the radius of an object.

min–param is the first position during reparameterization.

max–param is the last position during reparameterization.

max–iteration is the maximum iteration between min–param and max–param.

Limitations: None

Example:

```
; abl:create–vradius–wire
; Define a vradius
(define vrad (abl:make–radius–param–rads–tan
    (list 0.0  0.2  0.4  0.7  1.0)
    (list 3.0 0.1 0.1 3.0 3.0)))
;; vrad
(define wireb
    (abl:create–vradius–wire vrad 0.0 10.0 200))
;; wireb
; OUTPUT Original

(define sweep1 (sweep:law wireb
    (position 0 0 0) (gvector 0 0 1)
    (sweep:options ”sweep_angle” 360 ”solid” #t)))
;; sweep1
(zoom–all)
;; #[view 984568]
; OUTPUT Result
```

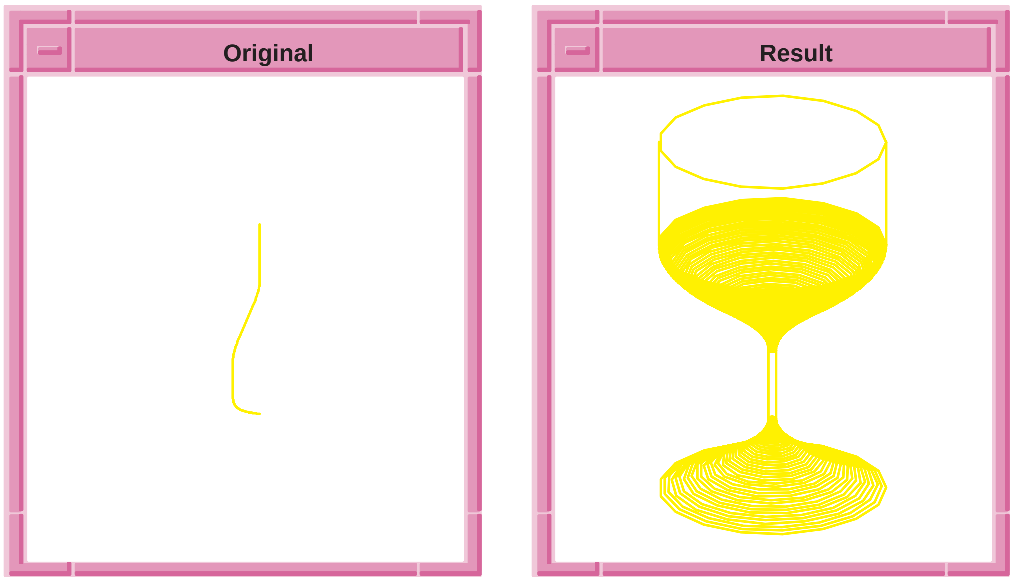


Figure 3-6. abl:create–vradius–wire

abl:create–vradius–wire–from–edge

Scheme Extension:	Blending
Action:	Creates a wire body corresponding to a particular vradius object (reparameterized).
Filename:	abl/abl_scm/abl_scm.cxx
APIs:	api_make_wire

Syntax:

```
(abl:create–vradius–wire–from–edge svradius in–edge
    max–iteration)
```

Arg Types:

svradius	vradius
in–edge	edge
max–iteration	integer

Returns: wire–body

Errors: None

Description: This Scheme extension copies the vradius object, uncalibrates the copy, reparameterizes the copy on the parameter interval of the input edge, calibrates the copy, evaluates the vradius copy “max–iteration” times on the interval of the input edge, uncalibrates the copy, and then returns a wire body with radii corresponding to the evaluation values.

svradius is the radius of an object.

in–edge is a input edge.

max–iteration is the maximum iteration.

Limitations: The input edge must have a finite parameter interval.

Example:

```
; abl:create–vradius–wire–from–edge
; create a vradius
(define blendvar (lambda (edge)
    (let ((rad2 (abl:make–radius–param–rads–tan
    (list 0.0  0.2  0.4  0.7  1.0)
    (list 3.0 0.1 0.1 3.0 3.0)))) rad2)))
;; blendvar
(define a (solid:sphere (position 0 0 0) 50))
;; a
(define b (solid:torus (position 35 0 0) 55 25))
;; b
(define union (solid:unite a b))
;; union
(define edgelist (entity:edges union))
;; edgelist
(define e0 (list–ref edgelist 0))
;; e0
(define wireb (abl:create–vradius–wire–from–edge
    (blendvar e0) e0 50))
;; wireb
(entity:set–color wireb 1)
;; ()
(zoom–all)
;; #[view 2033168]
; OUTPUT Original

(define sweep1 (sweep:law wireb
    (position 0 0 0) (gvector 0 0 1)
    (sweep:options ”sweep_angle” 360 ”solid” #t)))
;; sweep1
; OUTPUT Result
```

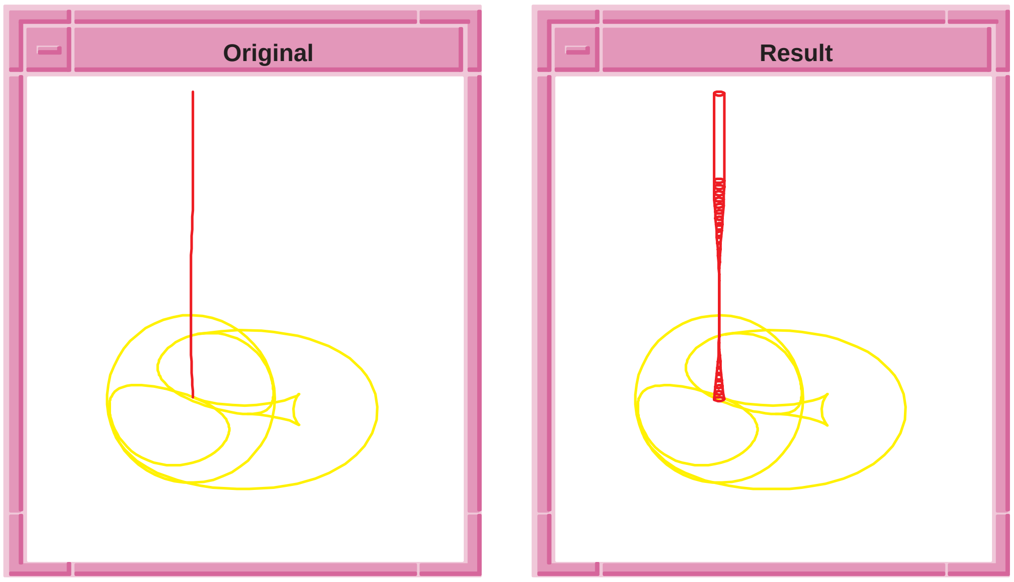


Figure 3-7. abl:create–vradius–wire–from–edge

abl:edge–blend

Scheme Extension: Blending

Action: Creates an edge sequence blend.

Filename: abl/abl_scm/abl_scm.cxx

APIs: None

Syntax:

```
(abl:edge-blend edge {vradius=null | vradius-pair
    round1=-1.0 round2=-1.0}
    [single=#f] [lofted-keyword loft-edge=null]
    [section-keyword left-thumbweight=-1.0
    right-thumbweight=-1.0]
    [side-edge1=null [tangent-flag1=#f]]
    [side-edge2=null [tangent-flag2=#f]]
    [start-setback=-1.0 end-setback=1.0])
```

Arg Types:	edge	edge
	vradius	vradius
	vradius–pair	vradius . vradius
	round1	real
	round2	real
	single	boolean
	lofted–keyword	string | symbol
	loft–edge	edge
	section–keyword	string | symbol
	left–thumbweight	real
	right–thumbweight	real
	side–edge1	edge
	tangent–flag1	boolean
	side–edge2	edge
	tangent–flag2	boolean
	start–setback	real
	end–setback	real

Returns: edge

Errors: None

Description: This extensions performs edge sequence–following blending. It requires an edge on which to place the blend attribute. The edge must belong to a solid body. Next, one vradius object or a pair of vradius objects must be specified. vradius defines the blend to be applied to the edge.

If the single flag is set to #f (default), the attribute is placed on the entire set of edges and/or vertices forming the blendable network that is attached to the given edge. Otherwise, the blend attribute is placed only on the given edge.

A lofted–edge option is available which allows specification of an edge along which the blend is to be lofted. To specify a lofted edge, either the string “lofted” or the symbol ‘lofted, followed by an edge should be entered.

A section form option is available which allows specification of thumbweights. Thumbweights provide a mechanism for custom designing a blend cross section by specifying two numbers, called left–thumbweight and right–thumbweight, which affect the shape of the cross section. For example, if both are set to unity (1), the cross section is circular. Increasing one thumbweight has the effect of pulling the section curve toward the corresponding end. The values for both thumbweights must be positive. To specify thumbweights, either the string “section” or the symbol ‘section should be entered, followed by the thumbweight values.

This extension has two options (side–edge1 and side–edge2) for side entities for the sequence blend. Up to two such side entities can be selected. A side entity must be an edge belonging to the face having another edge that is to be blended. The side entity can be tangent to the face (tangent–flag* set to #t).

Different setbacks may be applied to each end of the blended edge using start–setback and end–setback. A setback defines a plane normal to the edge through a point on the edge, set back from the edge end by the given distance. The intersection of the setback plane with the blend surface for the edge defines a curve that may be used to bound part of the face for the blended vertex at the end of the edge. If the vertex is not blended, the setback has no effect. The setback distance along the edge is calculated by finding the tangent derivative at the edge end, and hence a distance in edge parameter corresponding to the setback value (given in global body space), and from that, a setback parameter value at which the defining point of the setback plane lies.

To complete the blending operations after various entities have been marked as having blends, the extension blend:network, which fixes the body, must be issued.

edge is an input edge.

vradius is the blend to be applied to the edge.

vradius–pair is the blend to be applied to the edge.

round1 is a rounded edge.

round2 is a rounded edge.

single is a flag used to set whether the attribute is placed on the entire set of edges.

lofted–keyword is a keyword or symbol.

loft–edge is an option available which allows specification of an edge along which the blend is to be lofted.

section–keyword is a keyword or symbol.

left–thumbweight is a thumbweight for custom designing a blend cross section.

right–thumbweight is a thumbweight for custom designing a blend cross section.

side–edge1 side entity for the sequence blend.

side–edge2 side entity for the sequence blend.

tangent–flag1 flag set for entity to be tangent to the face.

tangent–flag2 flag set for entity to be tangent to the face.

start–setback setbacks applied to the start.

end–setback setbacks applied to the start.

Limitations: None

Example:

```
; abl:edge–blend
; Create a solid block
(define block1 (solid:block (position –20 –20 –20)
    (position 15 20 25)))
;; block1
; OUTPUT Original

; Create a two–ends vradius
(define rad1 (abl:two–ends–rad 5 10))
;; rad1
(define edges1 (entity:edges block1))
;; edges1
; Define an edge to put an attribute on
(define edge1 (car edges1))
;; edge1
; Put a blend attribute on the selected edge
(define blend (abl:edge–blend edge1 rad1))
;; blend
; Complete the blend
(blend:fix edge1)
;; #t
(define edge2 (car (cdr (cdr (cdr (cdr
    (cdr (cdr edges1))))))))
;; edge2
(define rad2 (abl:rnd–ch–rad 5 10 8 12))
;; rad2
; Put a blend attribute on the selected edge
; This creates a chamfer blend that is rounded.
(define chamfer (abl:edge–blend edge2 rad2 0.5 0.5))
;; chamfer
; Complete the blend
(blend:fix edge2)
;; #t
; OUTPUT Result
```

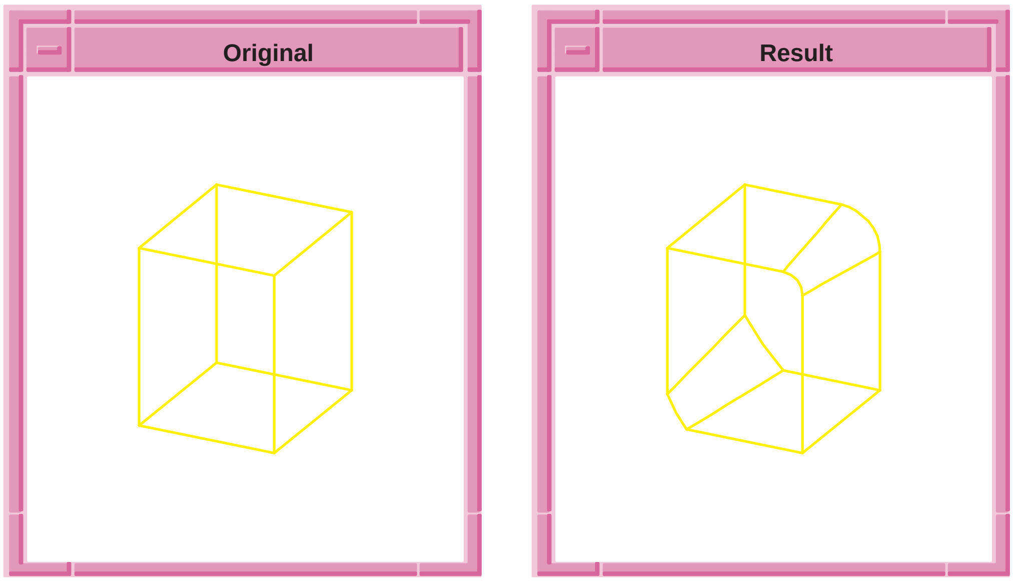


Figure 3-8. **abl:edge–blend**

abl:ell–rad

Scheme Extension:	Blending
Action:	Creates a vradius object for setting advanced blending attributes for a blend with an elliptical radius.
Filename:	abl/abl_scm/abl_scm.cxx
APIs:	api_make_radius_rot_ellipse

Syntax:

```
(abl:ell-rad ref-face-left major1 min1 angle1 angle2
    [major2=major1 min2=min1] [acis-opts])
```

Arg Types:

ref–face–left	boolean
major1	real
min1	real
angle1	real
angle2	real
major2	real
min2	real
acis–opts	acis–options

Returns: vradius

Errors: None

Description: This extension creates a vradius object which corresponds to class var_rad_rot_ellipse. The blend cross section is elliptical in nature and the cross section at the start and end of the blend are specified by the defining parameters of the ellipse at the two ends. The intermediate cross sections are internally calculated to define the radius function. If the values at the end are omitted, the same values are used as those for the start. The ref–face–left argument specifies which face is to be considered as the reference face (#t corresponds to left).

ref–face–left argument specifies which face is to be considered as the reference face.

major1 is blend cross section specification at the start.

min1 is blend cross section specification at the start.

angle1 is blend cross section specification at the start.

major2 is blend cross section specification at the end.

min2 is blend cross section specification at the end.

angle2 is blend cross section specification at the end.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; abl:ell-rad
; Create a solid block
(define block1 (solid:block (position –20 –20 –20)
    (position 15 20 25)))
;; block1
; OUTPUT Original
; Create an elliptical vradius
(define rad1 (abl:ell-rad #t 10 20 35 10 20 25))
;; rad1
(define edges1 (entity:edges block1))
;; edges1
; Define an edge to put an attribute on
(define edge1 (car edges1))
;; edge1
; Put a blend attribute on the selected edge
(define blend (abl:edge-blend edge1 rad1))
;; blend
; Complete the blend
(blend:fix edge1)
;; #t
(define edge2 (car (cdr (cdr (cdr (cdr
    (cdr (cdr edges1))))))))
;; edge2
(define rad2 (abl:ell-rad #t 10 20 35))
;; rad2
; Put a blend attribute on the selected edge
; This creates a chamfer blend that is rounded.
(define chamfer (abl:edge-blend edge2 rad2 0.5 0.5))
;; chamfer
; Complete the blend
(blend:fix edge2)
;; #t
; OUTPUT Result

; Render the result
(render)
;; ()
; OUTPUT Rendered Result
```

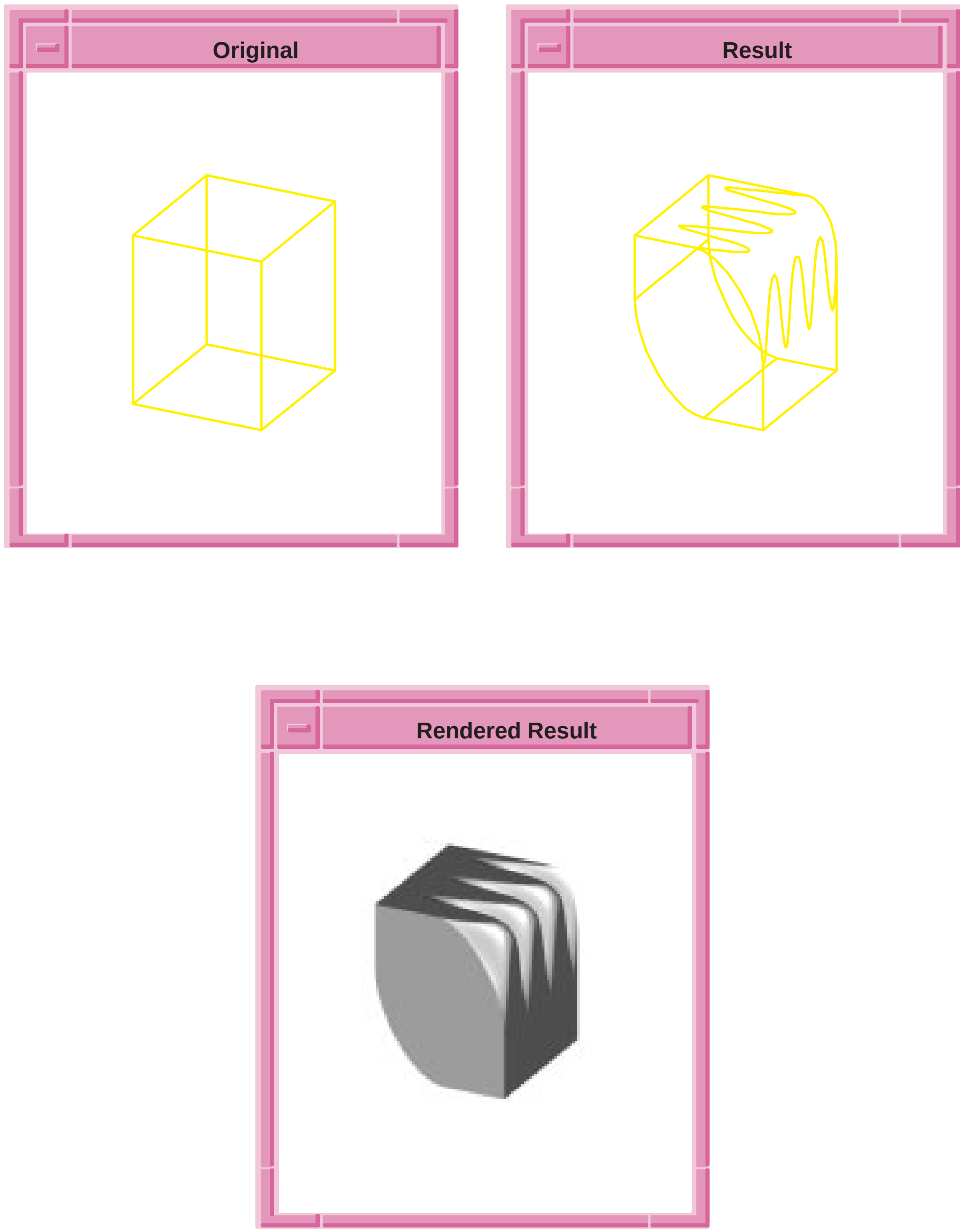


Figure 3-9. abl:ell–rad

abl:ent–ent–blend

Scheme Extension: Blending

Action:	Creates an entity–entity blend.
Filename:	abl/abl_scm/abl_scm.cxx
APIs:	api_set_ee_cr_blend

Syntax:

```
(abl:ent–ent–blend {face1 | edge1 [tangent–keyword
    tangent–face1]} {face2 | edge2
    [tangent–keyword tangent–face2]}
    {vradius | vradius–pair round1 round2}
    position convexity
    [def–edge] [section–keyword left–thumbweight
    right–thumbweight] [acis–opts])
```

Arg Types:

face1	face
edge1	edge
tangent–keyword	string | symbol
tangent–face1	face
face2	face
edge2	edge
tangent–face2	face
vradius	vradius
vradius–pair	vradius . vradius
round1	real
round2	real
position	position
convexity	boolean
def–edge	edge
section–keyword	string | symbol
left–thumbweight	real
right–thumbweight	real
acis–opts	acis–options

Returns: body

Errors: None

Description: This extension performs entity–entity blending. The side entities for the blend, which may be either edges or faces, are initially selected (edge1 or face1 and edge2 or face2). If an entity is an edge, it can be optionally tangent to a face by specifying either the string “tan” or the symbol ‘tan, followed by the tangent face. Either one vradius object or a pair of vradius objects, followed by two round values, must be specified next. The pair of vradius objects and two round values correspond to the result of abl:rnd–ch–rad.

After selecting the side entities and the vradius, a position designating a rough starting point for the blend is specified, followed by a convexity flag which is #t for convex or #f for concave.

The defining edge must be specified for variable radius entity–entity blending, although it is optional for constant radius blending.

A section form option is available which allows specification of thumbweights. Thumbweights provide a mechanism for custom designing a blend cross section by specifying two numbers, called left–thumbweight and right–thumbweight, which affect the shape of the cross section. For example, if both are set to unity (1), the cross section is circular. Increasing one thumbweight has the effect of pulling the section curve toward the corresponding end. The values for both thumbweights must be positive. To specify thumbweights, either the string "section" or the symbol 'section should be entered, followed by the thumbweight values.

To complete the blending operations after various entities have been marked as having blends, the extension blend:network, which fixes the body, must be issued.

face1 side entity for the blend.

edge1 side entity for the blend.

face2 side entity for the blend.

edge2 side entity for the blend.

tangent–keyword is a string or symbol for tangential specification.

tangent–face1 is a tangential face to an edge.

tangent–face2 is a tangential face to an edge.

vradius is a radius value correspond to the result of abl:rnd–ch–rad.

vradius–pair is a radius value correspond to the result of abl:rnd–ch–rad.

round1 is a round value correspond to the result of abl:rnd–ch–rad.

round2 is a round value correspond to the result of abl:rnd–ch–rad.

position is a position designating a rough starting point for the blend.

convexity is a flag for convex or concave.

def–edge is a defining edge.

section–keyword is a keyword to specify thumbweights.

left–thumbweight is a mechanism for custom designing a blend cross section.

right–thumbweight is a mechanism for custom designing a blend cross section.

acis–opts contains journaling and versioning information.

Limitations: The variable radius forms of this extension are not currently applicable.

Example:

```
; abl:ent-ent-blend
; Create a solid block
(define block1 (solid:block (position -30 -30 -5)
    (position 30 30 5)))
;; block1
(define sphere1
    (solid:sphere (position 0 0 25) 15))
;; sphere1
(define blob1 (bool:unite sphere1 block1))
;; blob1
; Set the color for blob1
(entity:set-color sphere1 3)
;; ()
; OUTPUT Original
; Create a vradius
(define rad1 (abl:const-rad 10))
;; rad1
(define face1 (pick:face (ray (position 0 0 0)
    (gvector 0 0 1)) 1))
;; face1
(entity:set-color face1 4)
;; ()
(define face2 (pick:face (ray (position 0 0 20)
    (gvector 1 0 0)) 1))
;; face2
(entity:set-color face2 6)
;; ()
; Create an entity-entity blend.
(define blend (abl:ent-ent-blend face1 face2 rad1
    (position 25 0 10) #f))
;; blend
; Complete the blend
(blend:fix blob1)
;; #t
; OUTPUT Result

; Render the result
(render)
;; ()
; OUTPUT Rendered Result
```

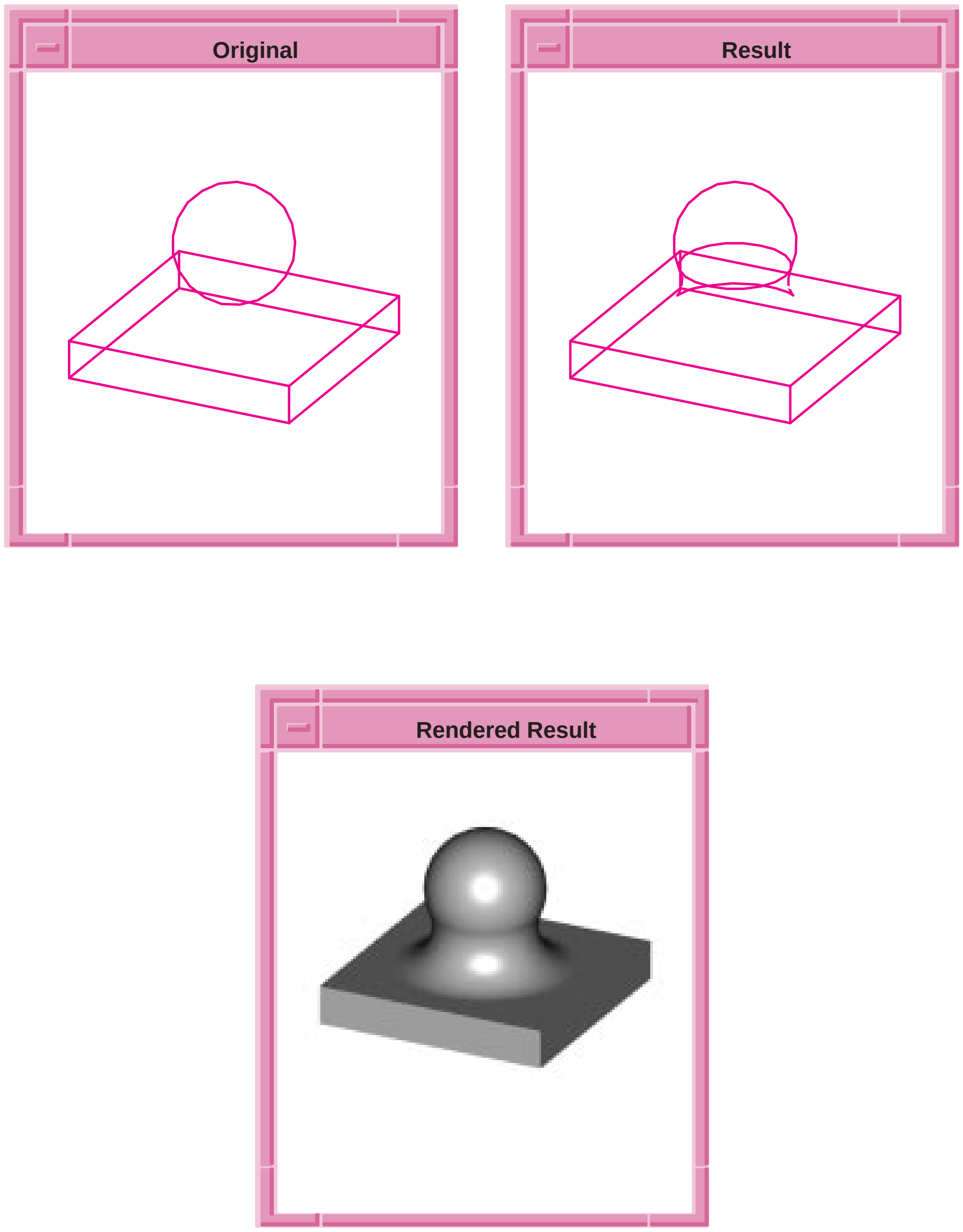


Figure 3-10. abl:ent–ent–blend

abl:eval–vradius–value

Scheme Extension: Blending

Action: Gets the evaluation value for a vradius object from a set of parameters.

Filename: abl/abl_scm/abl_scm.cxx

APIs: None

Syntax: (**abl:create–vradius–value** svradius sreal min–param max–param)

Arg Types:

svradius	vradius
sreal	real
min–param	real
max–param	real

Returns: real

Errors: None

Description: Copies the vradius object, uncalibrates the copy, reparameterizes the copy between ”min–param” and ”max–param”, calibrates the copy, evaluates the value at ”sreal”, and then uncalibrates the copy.

svradius is the radius of an object.

sreal is a real value where copy is evaluated.

min–param is the first position during reparameterization.

max–param is the last position during reparameterization.

Limitations: None

Example:

```
; abl:eval–vradius–value
; Create a vradius
(define vrad (abl:make–radius–param–rads–tan
    (list 0.0  0.2  0.4  0.7  1.0)
    (list 3 0.1 0.1 3 3)))
;; vrad
(abl:eval–vradius–value vrad 10.0 1.0 100.0)
;; 1.15890308039068
```

abl:eval–vradius–value–from–edge

Scheme Extension: Blending

Action: Gets the evaluation value for a vradius object from a set of parameters.

Filename: abl/abl_scm/abl_scm.cxx

APIs:	None

Syntax:
```
(**abl:create–vradius–value–from–edge** svradius
    sreal in–edge)
```

Arg Types:		
	svradius	vradius
	sreal	real
	in–edge	edge

Returns: real

Errors: None

Description: Copies the vradius object, uncalibrates the copy, reparameterizes the copy to the parameter space of the input edge, calibrates the copy, evaluates the copy at ”sreal”, and then uncalibrates the copy.

svradius is the radius of an object.

sreal is a real value where copy is evaluated.

in–edge is a input edge.

Limitations: The input edge must have a finite parameter interval.

Example:
```
; abl:eval–vradius–value–from–edge
; Create a vradius
(define blendvar (lambda (edge)
    (let ((rad2 (abl:make–radius–param–rads–tan
    (list 0.0  0.2  0.4  0.7  1.0)
    (list 3.0 0.1 0.1 3.0 3.0))))
    rad2)))
;; blendvar
(define a (solid:sphere (position 0 0 0) 50))
;; a
(define b (solid:torus (position 35 0 0) 55 25))
;; b
(define union (solid:unite a b))
;; union
(define edgelist (entity:edges union))
;; edgelist
(define e0 (list–ref edgelist 0))
;; e0
(abl:eval–vradius–value–from–edge
    (blendvar e0) 3.0 e0)
;; 2.67393877928773
```

abl:fixed–width–rad

Scheme Extension: Blending

Action:	Creates a vradius object for setting advanced blending attributes for a blend with a fixed width radius.
Filename:	abl/abl_scm/abl_scm.cxx
APIs:	api_make_radius_fixed_width
Syntax:	(**abl:fixed–width–rad** width [acis–opts])

Arg Types:

width	real
acis–opts	acis–options

Returns: vradius

Errors: None

Description: This extension creates a vradius object which corresponds to class var_rad_fixed_width. It specifies a radius function that varies implicitly. At any point along the blend, the radius will be calculated in such a way as to produce a blend with a fixed, or constant, cross sectional width.

width is a fixed or constant cross sectional width.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; abl:fixed–width–rad
; Create a solid block
(define block1 (solid:block (position –20 –20 –20)
    (position 15 20 25)))
;; block1
; OUTPUT Original

; Create a fixed–width vradius
(define rad1 (abl:fixed–width–rad 10))
;; rad1
(define edges1 (entity:edges block1))
;; edges1
(define edge1 (car edges1))
;; edge1
; Put a blend attribute on the selected edge
(define blend (abl:edge–blend edge1 rad1))
;; blend
; Complete the blend
(blend:fix edge1)
;; #t
; OUTPUT Result
```

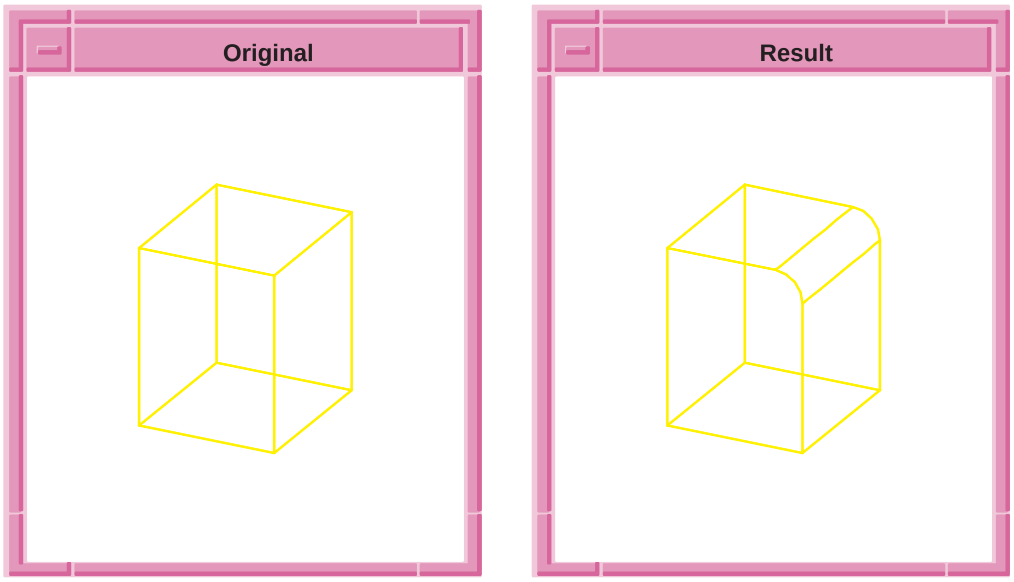


Figure 3-11. abl:fixed–width–rad

abl:make–radius–param–rads

Scheme Extension:	Blending
Action:	Creates a variable radius object from a set of parameter values along the blend and the radius values at each.
Filename:	abl/abl_scm/abl_scm.cxx
APIs:	api_make_radius_param_rads

Syntax:
```
(abl:make–radius–param–rads param–list radii–list
    [acis–opts])
```

Arg Types:

param–list	real | (real ...)
radii–list	real | (real ...)
acis–opts	acis–options

Returns: vradius

Errors: Input lists must have equal lengths.

Description: Creates a variable radius object from a set of parameter values along the blend and the radius values at each. This radius object can then be used as an input to the blend API function api_set_abh_blends or api_set_ee_vr_blend.

param–list is a set of parameter values along the blend.

radii–list is a set of radius values along the blend.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; abl:make–radius–param–rads
; Create geometry to illustrate extension.
; Create a solid block, a cylinder, and unite them.
(define block (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block
(define cylinder (solid:cylinder (position 20 0 –20)
    (position 20 0 20) 20))
;; cylinder
(define my–part (solid:unite block cylinder))
;; my–part
; Pick an edge on the body.
(define def–edge( pick:edge (ray
    (position 10 –20 100) (gvector 0 0 –1))))
;; def–edge
; Display the edge in red.
(define set–color (entity:set–color def–edge RED))
;; set–color
; OUTPUT Original

; Define a list of parameter values.
(define param–list (list 0.0 40.0 102.8318 142.8318))
;; param–list
; Define a list of radius values.
(define radii–list (list 10 4 6 8))
;; radii–list
; Create the v–radius object.
(define v–radius (abl:make–radius–param–rads
    param–list radii–list))
;; v–radius
; Complete the blend.
(define blend1 (abl:edge–blend def–edge v–radius))
;; blend1
(define blend2 (blend:network
    (blend:get–network def–edge)))
;; blend2
; OUTPUT Result
```

```
(render)
;; ()
; OUTPUT Rendered Result
```

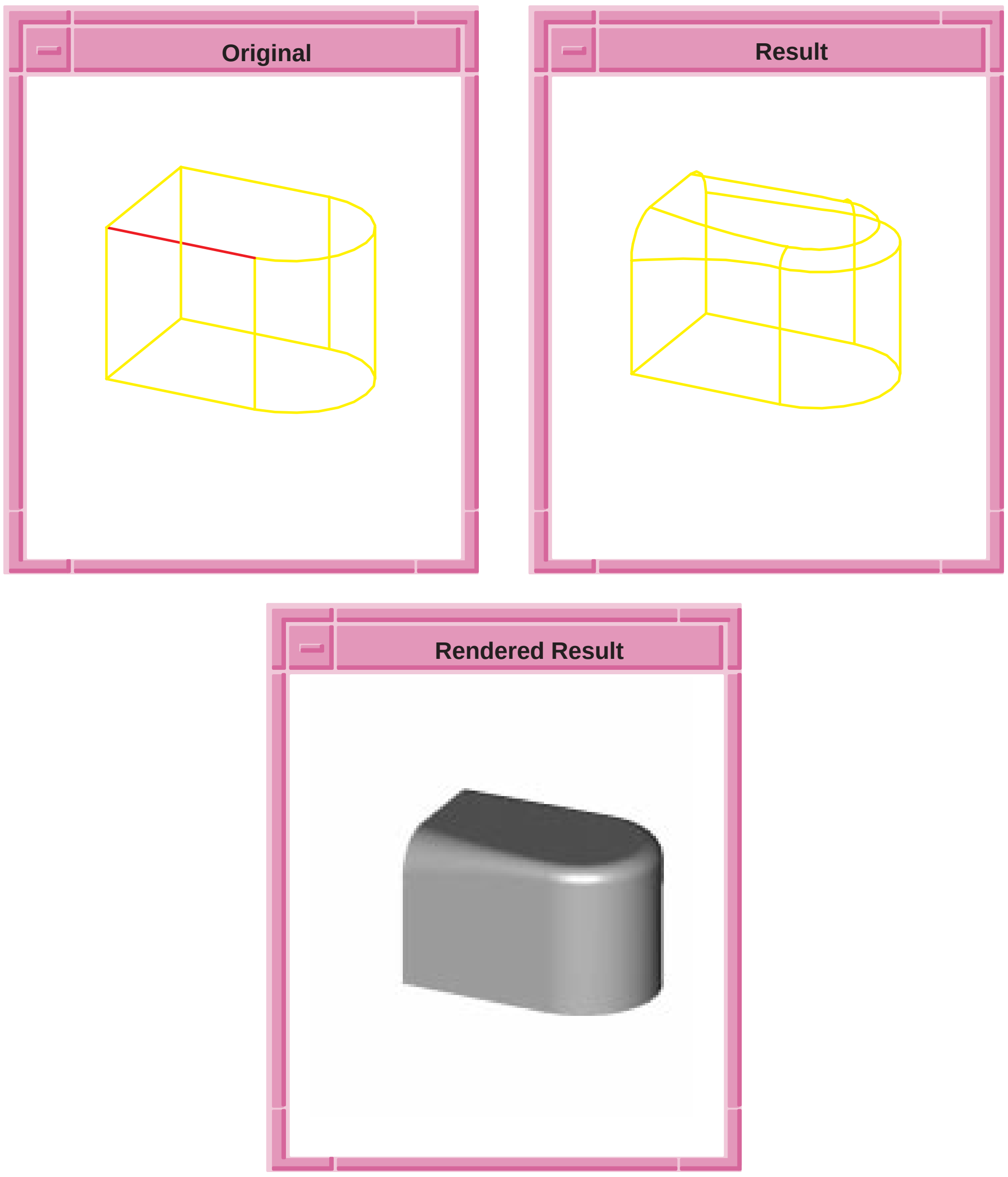


Figure 3-12. abl:make–radius–param–rad

abl:make–radius–param–rads–tan

Scheme Extension: Blending

Action: Creates a variable radius object from a set of parameter values along the blend and radius values at each parameter.

Filename: abl/abl_scm/abl_scm.cxx

APIs: api_make_radius_param_rads_tan

Syntax:

```
(abl:make-radius-param-rads-tan param-list
    radii-list [slope-keyword slope]
    [slope-keyword slope] [acis-opts])
```

Arg Types:

param–list	real | (real ...)
radii–list	real | (real ...)
slope–keyword	string
slope	real
acis–opts	acis–options

Returns: vradius

Errors: Input lists must have equal lengths.

Description: Creates a vradius object which corresponds to class var_rad_functional. It uses parameter–radius pairs plus start and end slope information to create a var_rad_functional object. This allows the radius function to be specified at various positions along the blend.

This extension allows specification of the slope of the calibration curve at the start and end. The radius object created can be used as an input to the blend functions api_set_aph_blends or api_set_ee_vr_blend.

param–list is a list of parameter values.

radii–list should contain the same number of arguments as param–list. It represents the intended blend radius at those parameter values.

slope–keyword must be “start–slope” or “end–slope” string followed by slope.

slope is interpreted as the slope (dy/dx) in 2D coordinates, where 0 is horizontal, and 1 is 45 degrees. Pass in NULL for unspecified.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; abl:make-radius-param-rads-tan
; Create geometry to illustrate extension.
; Create a solid block, a cylinder, and unite them.
(define block (solid:block (position -20 -20 -20)
     (position 20 20 20)))
;; block
(define cylinder (solid:cylinder (position 20 0 -20)
     (position 20 0 20) 20))
;; cylinder
(define my-part (solid:unite block cylinder))
;; my-part
; Pick an edge on the body.
(define def-edge( pick:edge (ray
    (position 10 -20 100) (gvector 0 0 -1))))
;; def-edge
; Display the edge in red.
(define set-color (entity:set-color def-edge RED))
;; set-color
; Define a list of parameter values.
(define param-list (list 0.0 40.0 102.8318 142.8318))
;; param-list
; Define a list of radius values.
(define radii-list (list 10 4 6 8))
;; radii-list
; Create the v-radius object.
(define v-radius (abl:make-radius-param-rads-tan
    param-list radii-list))
;; v-radius
; OUTPUT Original

; Complete the blend.
(define blend1 (abl:edge-blend def-edge v-radius))
;; blend1
(define blend2 (blend:network
    (blend:get-network def-edge)))
;; blend2
; OUTPUT Result

(render)
;; ()
; OUTPUT Rendered
```

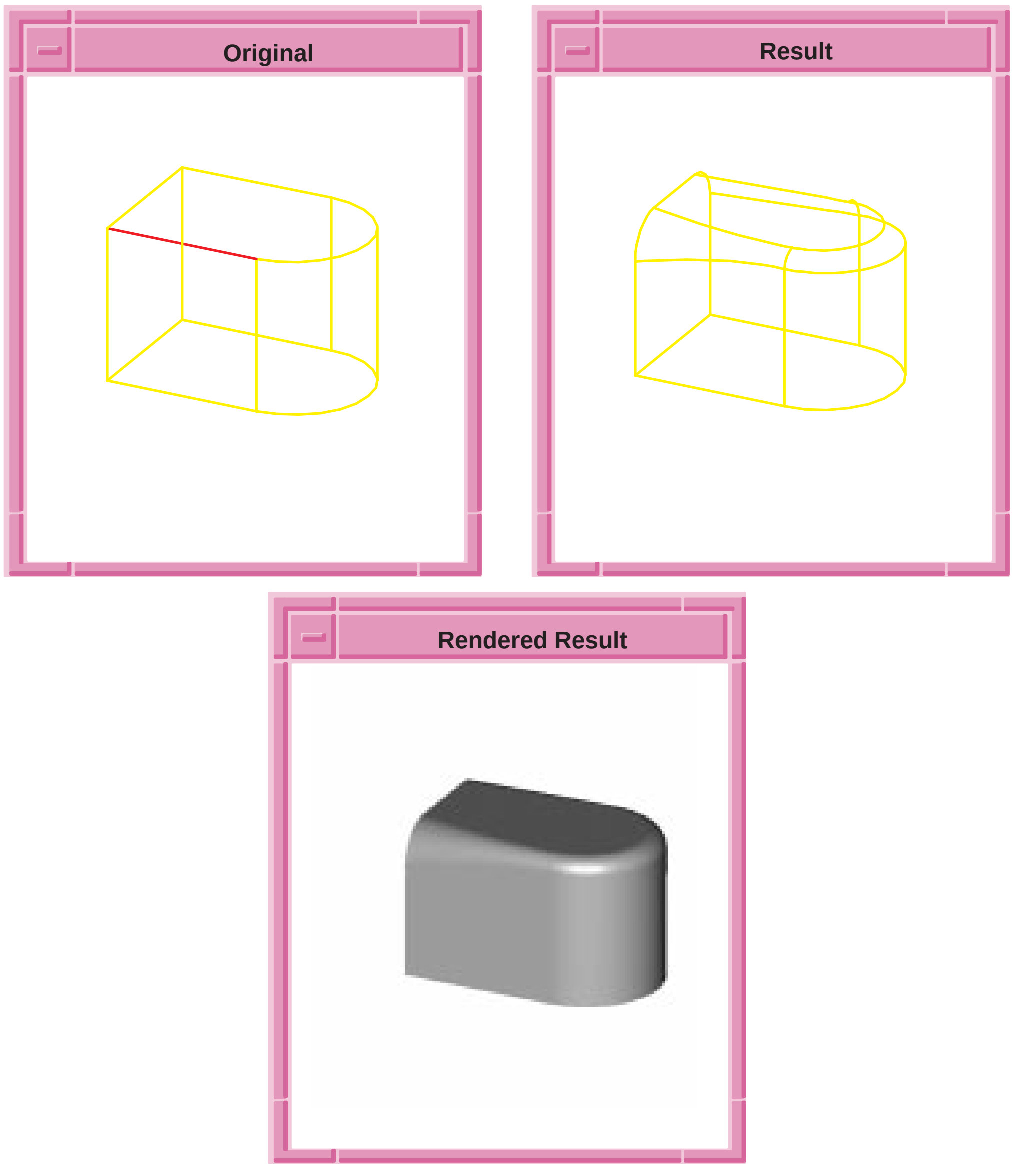


Figure 3-13. abl:make–radius–param–rads–tan

abl:pos–rad

Scheme Extension: Blending

Action:	Creates a vradius object for setting advanced blending attributes for a blend whose radius is specified by position–radius pairs.
Filename:	abl/abl_scm/abl_scm.cxx
APIs:	api_make_radius_pos_rads

Syntax:

```
(abl:pos–rad position–list radii–list curve–edge
   [”start–tan”][”end–tan”] [transform] [acis–opts])
```

Arg Types:

position–list	position | (position ...)
radii–list	real | (real ...)
curve–edge	edge | curve
”start–tan”	string
”end–tan”	string
transform	transform
acis–opts	acis–options

Returns: vradius

Errors: Input lists must have equal lengths.

Description: This extension creates a vradius object which corresponds to class var_rad_functional. It uses position–radius pairs and a defining edge to create a 2D B–spline curve, which is then used to create a var_rad_functional object. This allows the radius function to be specified at various positions along the blend.

position–list defines the positions on or close to the edge to be blended. Ideally, the first and last positions should correspond to the starting and ending positions of the edge.

radii–list defines the intended blend radius at the position points defined in position–list. radii–list should contain the same number of arguments as the position–list.

curve–edge is a curve or edge. The positions and curve (or edge) must be represented in the same coordinate system. For blending, the coordinate system should be that of the body being blended, not model space. If the calibration curve is created by api_smooth_edges_to_curve, it is in the coordinate system of the body (i.e., that of the edges passed.) Therefore, if the positions are given in model space, they have to be transformed by the inverse of the blended body’s transform. This may be easily accomplished by passing the optional transform argument to this scheme extension where transform is obtained as the transform of the body being blended.

The valid strings ”start_tan” and ”end–tan” are interpreted as slope (dy/dx) in 2D coordinates. A zero (”0”) defines horizontal and a one (”1”) is 45 degrees. ”start–tan” and ”end–tan” are optional and independent of each other.

start_tan is interpreted as slope (dy/dx) in 2D coordinates.

end–tan is interpreted as slope (dy/dx) in 2D coordinates.

transform specifies any valid transform.

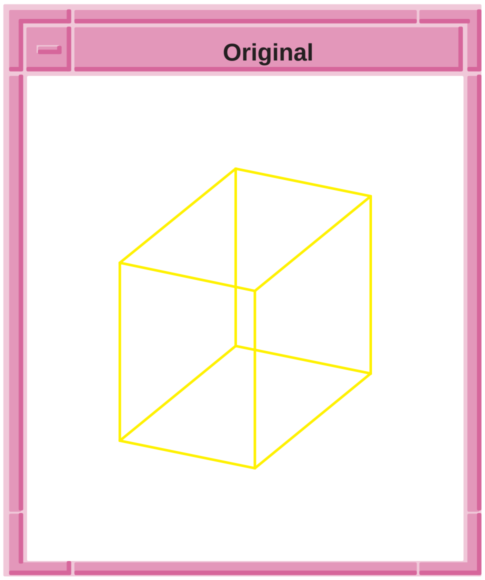
acis–opts contains journaling and versioning information.

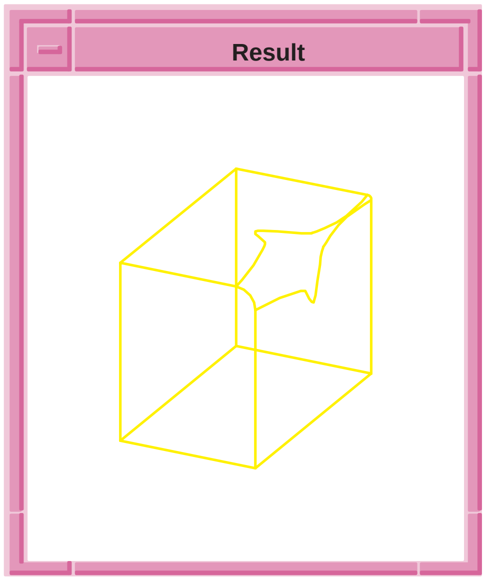
Limitations: None

Example:

```
; abl:pos–rad
; Create a solid block.
(define block1 (solid:block (position –20 –30 –20)
    (position 15 30 25)))
;; block1
; OUTPUT Original

; Define the edges.
(define edges1 (entity:edges block1))
;; edges1
; Define which edge to attach the attribute.
(define edge1 (car edges1))
;; edge1
; Define the vertex.
(define vert1 (entity:vertices edge1))
;; vert1
(define start1 (vertex:position (car vert1)))
;; start1
(define end1 (vertex:position (car (cdr vert1))))
;; end1
(define rad1 (abl:pos–rad (list start1
    (position 15 –5 25) (position 15 0 25)
    (position 15 5 25) end1)
    (list 5 10 15 3 1) edge1))
;; rad1
; Put a blend attribute on the selected edge.
(define blend (abl:edge–blend edge1 rad1))
;; blend
; Complete the blend.
(blend:fix edge1)
;; #t
; OUTPUT Result
; Render the result.
(render)
;; ()
; OUTPUT Rendered Result
```





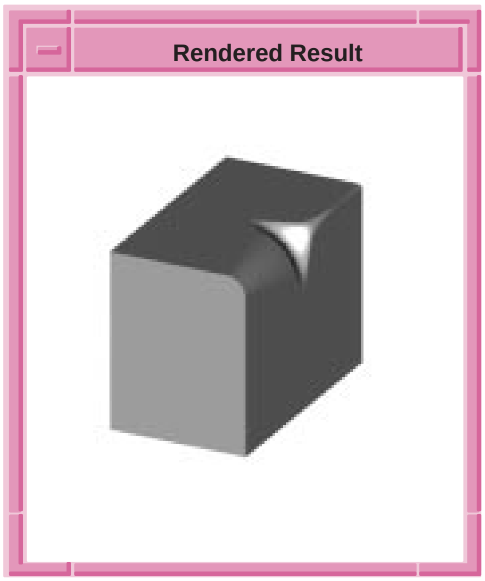


Figure 3-14. abl:pos–rad

```
; example 2:abl:pos–rad.
; Create the geometry to illustrate command. Create
; a block and a cylinder and unite them.
(part:clear)
;; #t
(define block (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block
(define cylinder (solid:cylinder (position 20 0 –20)
    (position 20 0 20) 20))
;; cylinder
(define united (solid:unite block cylinder))
;; united
; Create a list of all entities.
(define entity–list (part:entities))
;; entity–list
; Get the first entity from the entity list.
(define first–entity (car entity–list))
;; first–entity
; Transform the body.
(define transform (body:get–transform united))
;; transform
; Pick an edge on the body.
```

```
(define chosen-one (pick:edge
    (ray (position 10 -20 100) (gvector 0 0 -1))))
;; chosen-one
; Display it in red.
(define redshoes (entity:set-color chosen-one RED))
;; redshoes
; Get all of the edges that are smooth to the
; selected edge.
(define smooth (blend:get-smooth-edges chosen-one))
;; smooth
; Display them in blue.
(define bluemonday
    (entity:set-color smooth BLUE))
;; bluemonday
; Construct a curve approximating the smooth edges.
(define smooth-curve-list
    (blend:smooth-edges-to-curve smooth))
;; smooth-curve-list
; Find the calibration curve.
(define smooth-curves (car smooth-curve-list))
;; smooth-curves
; Define a position list.
(define positions (list (position -20 -20 -20)
    (position 20 -20 20) (position 20 20 20)
    (position -20 20 20)))
;; positions
; Define a radii list.
(define radii-list (list 8 5 5 8))
;; radii-list
; Build the v-radius object.
(if transform (define v-radius (abl:pos-rad positions
    radii-list smooth-curves transform))
    (define v-radius (abl:pos-rad positions
    radii-list smooth-curves)))
;; v-radius
; OUTPUT Original

; Complete the blend.
(define blend (abl:edge-blend chosen-one v-radius))
;; blend
(define complete (blend:network
    (blend:get-network chosen-one)))
;; complete
; OUTPUT Result
```

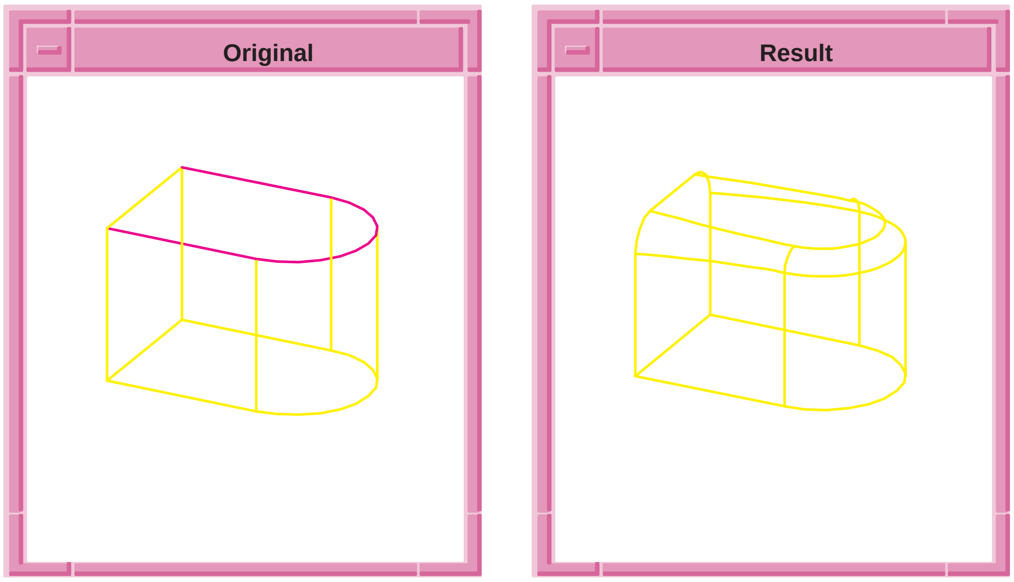


Figure 3-15. abl:pos–rad

abl:rnd–ch–rad

Scheme Extension:	Blending
Action:	Creates a pair of vradius objects for setting advanced blending attributes for blend with a rounded chamfer cross section.
Filename:	abl/abl_scm/abl_scm.cxx
APIs:	api_make_radius_rnd_chamfer

Syntax:

```
(abl:rnd–ch–rad left1 right1
    [left2=left1 right2=right1] [acis–opts])
```

Arg Types:

left1	real
right1	real
left2	real
right2	real
acis–opts	acis–options

Returns: vradius . vradius

Errors: None

Description: This extension creates a pair of vradius objects which correspond to class var_rad_two_ends. The first set of arguments, left1 and right1, specify the size of the chamfer on one end of blend. If the direction of the blend is considered to be in the *z*–direction, then left1 and left2 more or less specify the cut of the chamfer from the blend edge in the *x*–direction, while right1 and right2 more or less specify the cut of the chamfer from the blend edge in the *y*–direction.

If the optional second set of arguments, left2 and right2, is not specified, the chamfer on the other end of the blend is the same. Otherwise, the dimensions of the chamfer changes from the first pair to the dimensions specified from the second pair.

Using options on abl:edge–blend, a rounded chamfer can be created. Likewise, a cross section in terms of left and right ranges can be defined with respect to the defining edge and the bulge, or round, value. The round value is the normal distance from a simple planar chamfer with the same ranges to the rounded chamfer surface. These defining parameters for the rounded chamfer are specified at the two ends of the edge. If the values at the end are omitted, the same values are used as those for the start.

left1 more or less specify the cut of the chamfer from the blend edge in the *x*–direction.

left2 more or less specify the cut of the chamfer from the blend edge in the *x*–direction.

right1 more or less specify the cut of the chamfer from the blend edge in the *y*–direction.

right2 more or less specify the cut of the chamfer from the blend edge in the *y*–direction.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; abl:rnd–ch–rad
; Create a solid block
(define block1 (solid:block (position –20 –20 –20)
    (position 15 20 25)))
;; block1
; OUTPUT Original
; Define an edge for the blend.
(define edges1 (entity:edges block1))
;; edges1
; Define an edge to put an attribute on
(define edge1 (car edges1))
;; edge1
; Define the radius.
(define rad1 (abl:rnd-ch-rad 5 30 10 15))
;; rad1
; Put a blend attribute on the selected edge
(define blend (abl:edge-blend edge1 rad1 1 1))
;; blend
; Complete the blend
(blend:fix edge1)
;; #t
; OUTPUT Result
```

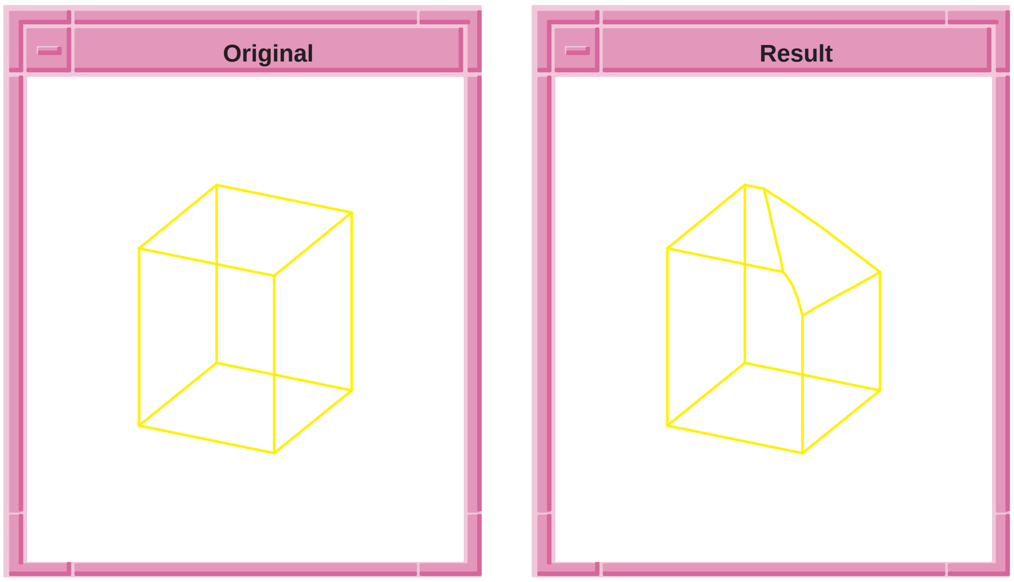


Figure 3-16. abl:rnd–ch–rad

abl:set–abh–blends

Scheme Extension: Blending

Action: Creates an edge sequence blend by calling api_set_abh_blends.

Filename: abl/abl_scm/abl_scm.cxx

APIs:	api_set_abh_blends

Syntax:

```
(abl:set-abh-blends edge-list {vradius=null |
vradius-pair}
    [section-keyword value1 value2]
    [calibration-keyword
    {calibration-curve start-edge end-edge} |
    {(calibration-curve start-edge end-edge)}]
    [side-edge1=null [tangent-flag1=#f]]
    [side-edge2=null [tangent-flag2=#f]]
    [start-setback=0 end-setback=0]
    [lofted-keyword loft-edge] [acis-opts])
```

Arg Types:

edge–list	entity **|** (entity ...)
vradius	vradius
vradius–pair	vradius **|** vradius
section–keyword	string
value1	real
value2	real
calibration–keyword	string
calibration–curve	curve
start–edge	edge
end–edge	edge
side–edge1	edge
tangent–flag1	boolean
side–edge2	edge
tangent–flag2	boolean
start–setback	real
end–setback	real
lofted–keyword	string
loft–edge	edge
acis–opts	acis–options

Returns: (entity ...)

Errors: None

Description: This extension calls the API api_set_abh_blends after setting the API's arguments.

This extension performs edge sequence following blending. It requires a list of edges on which to place the blend attributes. The edges must belong to a solid body. A vradius object or a pair of vradius objects must be specified. The vradius object(s) define the blends to be applied to the edges.

The section–keyword argument allows specification of thumbweight or rounded–chamfer blends.

Thumbweights provide a mechanism for custom designing a blend cross section by specifying two numbers which affect the shape of the cross section. For example, if both are set to unity (1), the cross section is circular. Increasing one thumbweight has the effect of pulling the section curve toward the corresponding end. The values for both thumbweights must be positive. To specify thumbweights, define the string "thumbweights" or the symbol 'thumbweights, followed by the thumbweight values.

A rounded chamfer blend can be created by using the string "rounded–chamfer" or the symbol 'rounded–chamfer. The string must be followed by start and end round, or bulge, values. These values correspond to the normal distance from a simple chamfer to the rounded chamfer surface.

The calibration–keyword argument allows you to define the start and end edges by using the string "calibration–curve" or the symbol 'calibration–curve. The string must be followed by a curve and two edges.

This extension has two options (side–edge1 and side–edge2) for side entities for the sequence blend. Up to two such side entities can be selected. A side entity must be an edge belonging to the face having another edge that is to be blended. The side entity can be tangent to the face (tangent–flag* set to #t).

The lofted–keyword argument allows you to specify which edge of the blend is to be lofted. To specify a lofted edge, define the string "lofted" or the symbol 'lofted. The string must be followed by an edge.

Setbacks can be applied to each end of the blended edge using start–setback and end–setback. A setback defines a plane normal to the edge through a point on the edge, set back from the edge end by the given distance. The intersection of the setback plane with the blend surface for the edge defines a curve that may be used to bound part of the face for the blended vertex at the end of the edge. If the vertex is not blended, the setback has no effect. The setback distance along the edge is calculated by finding the tangent derivative at the edge end, and hence a distance in edge parameter corresponding to the setback value (given in global body space). From that, a setback parameter value at which the defining point of the setback plane lies.

To complete the blending operations after various entities have been marked as having blends, the extension blend:network, which fixes the body, must be issued.

edge–list is a list of edges on which to place the blend attributes.

vradius is the blends to be applied to the edges.

vradius–pair is the blends to be applied to the edges.

section–keyword allows specification of thumbweight or rounded–chamfer blends.

value1 for custom designing a blend cross section.

value2 for custom designing a blend cross section.

calibration–keyword allows you to define the start and end edges.

calibration–curve is a curve used for calibration.

start–edge is a starting edge.

end–edge is an end edge.

side–edge1 is an edge belonging to the face having another edge that is to be blended.

side–edge2 is an edge belonging to the face having another edge that is to be blended.

tangent–flag1 is an entity tangent to the face .

tangent–flag2 is an entity tangent to the face .

start–setback defines a plane normal to the edge through a point on the edge, set back from the edge end by the given distance.

end–setback defines a plane normal to the edge through a point on the edge, set back from the edge end by the given distance.

lofted–keyword allows you to specify which edge of the blend is to be lofted.

loft–edge is the lofted edge.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; abl:set–abh–blends
; Create geometry to demonstrate command.
; Create a solid block.
(define block1 (solid:block (position –20 –20 –20)
    (position 15 20 25)))
;; block1
; Define an edge for the blend.
(define edges1 (entity:edges block1))
;; edges1
; Define an edge to put an attribute on.
(define att–edge (car edges1))
;; att–edge
; Define the calibration curve.
(define curve (car
    (blend:smooth–edges–to–curve att–edge)))
;; curve
; Define the radius.
(define rad1 (abl:rnd–ch–rad 5 30 10 15))
;; rad1
; Put a blend attribute on the selected edge.
(define blend (abl:set–abh–blends att–edge rad1
    'calibration–curve curve att–edge att–edge
    'rounded–chamfer 0 0))
;; blend
; OUTPUT Original

; Complete the blend.
(define finish (blend:fix att–edge))
;; finish
; OUTPUT Result
```

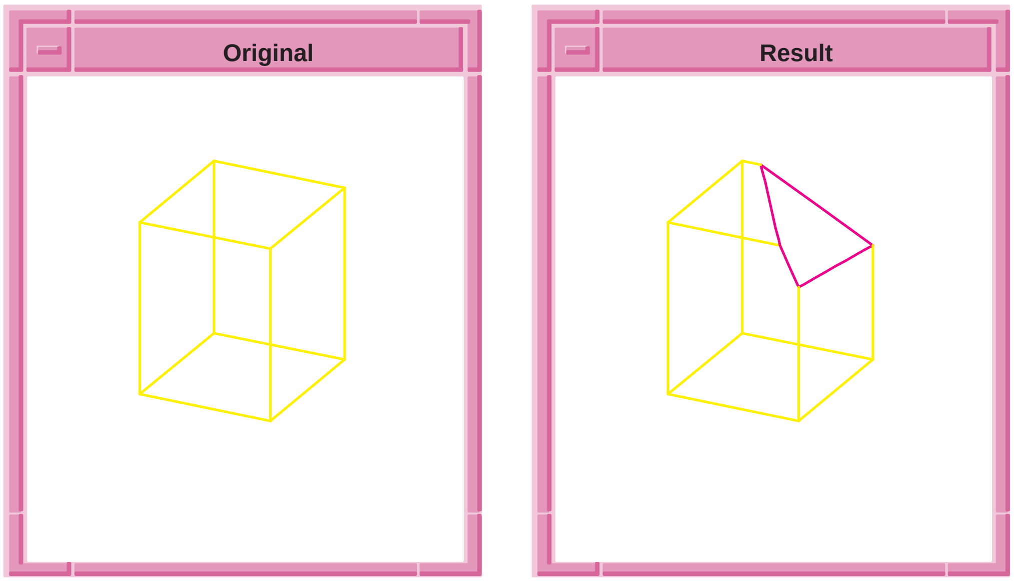


Figure 3-17. abl:set–abh–blends

abl:set–instruction

Scheme Extension:	Blending
Action:	Attaches a blend instruction to an entity.
Filename:	abl/abl_scm/abl_scm.cxx
APIs:	api_set_inst_blend

Syntax:
```
(abl:set-instruction entity action=2 [position]
    [acis-opts])
```

Arg Types:

entity	face | edge | vertex
action	integer
position	position
acis–opts	acis–options

Returns: boolean

Errors: None

Description: This extension attaches a blend instruction, which could be later used for an entity–entity blend, to the specified entity. The type of entity could be face, edge, or vertex. The instruction is specified by the value of the action argument, as follows:

Value	Instruction
1	roll on
2	cap
other	unknown

The *roll on* instruction (action=1) is used to tell the blend to roll on to the encountered entity, when the normal default would have been to cap the blend at that point. The *cap* instruction (action=2) is used to tell the blend to begin capping the existing blend face rather than to roll on to the new entities. If the action passed is *unknown*, all instructions on the entity are deleted.

The position argument may be provided to associate a position with the instruction to resolve ambiguities. Only one instruction per owner is allowed to refer to any particular position, and likewise, only one instruction per owner may be specified without a position.

entity is face, edge, or vertex.

action is to specify the instruction.

position associates a position with the instruction to resolve ambiguities.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; abl:set–instruction
; Create a solid block
(define block1 (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
(define cylinder1 (solid:cylinder (position 20 0 –20)
    (position 20 0 20) 20))
;; cylinder1
(define blob1 (solid:unite block1 cylinder1))
;; blob1
(define topface1 (pick:face (ray (position 0 0 0)
    (gvector 0 0 1)) 1))
;; topface1
(define planarface1 (pick:face
    (ray (position 0 0 0) (gvector 0 –1 0)) 1))
;; planarface1
(define curveface1 (pick:face (ray (position 0 0 0)
    (gvector 0 –1 0)) 1))
;; curveface1
; OUTPUT Original
; Define the radius.
(define rad1 (abl:const-rad 10))
;; rad1
(abl:set-instruction curveface1 2)
;; ”cap”
; Put a blend attribute on the selected edge
(define blend (abl:ent-ent-blend topface1 planarface1
    rad1 (position 0 0 0) #t))
;; blend
; Complete the blend
(blend:fix blob1)
;; #t
; OUTPUT Result
```

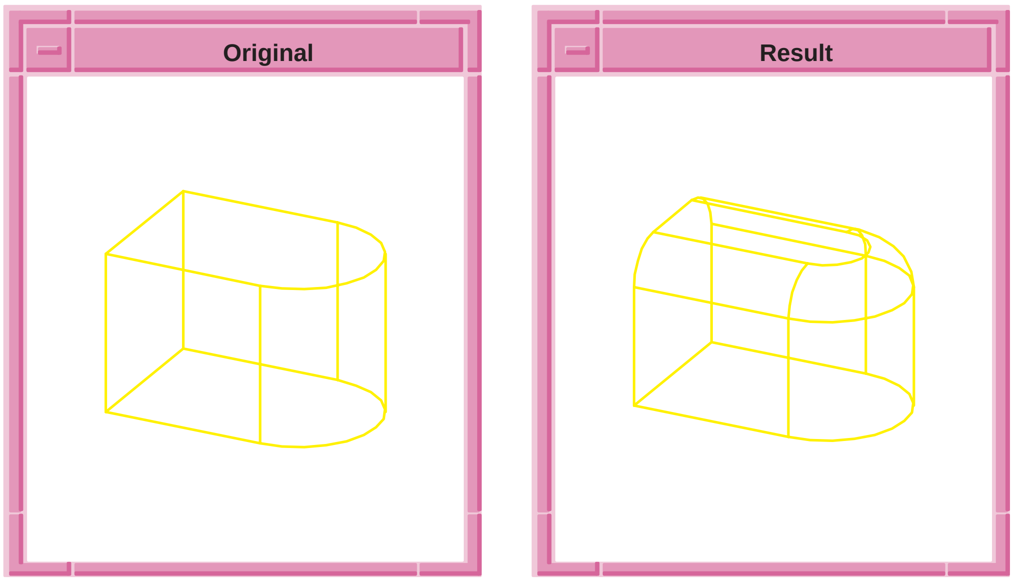


Figure 3-18. abl:set–instruction

abl:spline–rad

Scheme Extension: Blending

Action: Creates a vradius object for setting advanced blending attributes for a blend whose radius function is specified by a 3D spline curve.

Filename: abl/abl_scm/abl_scm.cxx

APIs:	api_make_radius_spline_rad	
Syntax:	(**abl:spline–rad** edge [acis–opts])	
Arg Types:	edge	edge
	acis–opts	acis–options
Returns:	vradius	
Errors:	None	

Description: This extension creates a vradius object which corresponds to class var_rad_functional. It uses bs3_edge to create a bs2_curve (2D B–spline curve) which is then used to create a var_rad_functional object. The radius values are taken from the *y*–coordinates of the 3D spline. This allows the input spline to be inspected visually, as a graph of *y* = *f(x)*, if appropriate values are used for the *x*–coordinates. The *x*– and *z*–coordinates are ignored internally. The parameter range of the input spline is unimportant, since it will be mapped to match the edge being blended.

edge is an input spline.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; abl:spline–rad
; Create a bs3_edge
(define edge1
    (edge:bezier (position 5 0 0) (position 15 0 20)
    (position 25 0 20) (position 35 0 0)))
;; edge1
; Create a vradius
(abl:spline–rad edge1)
;; #[functional vradius 595a2a0]
```

abl:two–ends–rad

Scheme Extension: Blending

Action: Creates a vradius object for setting advanced blending attributes for a blend with the radii specified at the two ends of the blend.

Filename: abl/abl_scm/abl_scm.cxx

APIs: api_make_radius_two_ends

Syntax:	(**abl:two–ends–rad** radius1 radius2 [acis–opts])

Arg Types:

radius1	real
radius2	real
acis–opts	acis–options

Returns: vradius

Errors: None

Description: This extension creates a vradius object which corresponds to class var_rad_two_ends. This allows specification of the radius values for the two ends of the edge to be blended. The blend starts with a radius of radius1 and then tapers or expands to a radius of radius2. Intermediate linearly varying radius values are implicitly calculated internally.

radius1 is the radius value at first edge.

radius2 is the radius value at last edge.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; abl:two–ends–rad
; Create a solid block
(define block1
    (solid:block (position –20 –20 –20)
    (position 15 20 25)))
;; block1
; OUTPUT Original

; Create a two–ends vradius
(define rad1 (abl:two–ends–rad 10 25))
;; rad1
(define edges1 (entity:edges block1))
;; edges1
; Define an edge to put an attribute on
(define edge1 (car edges1))
;; edge1
; Put a blend attribute on the selected edge
(define blend (abl:edge–blend edge1 rad1))
;; blend
; Complete the blend
(blend:fix edge1)
;; #t
; OUTPUT Result
```

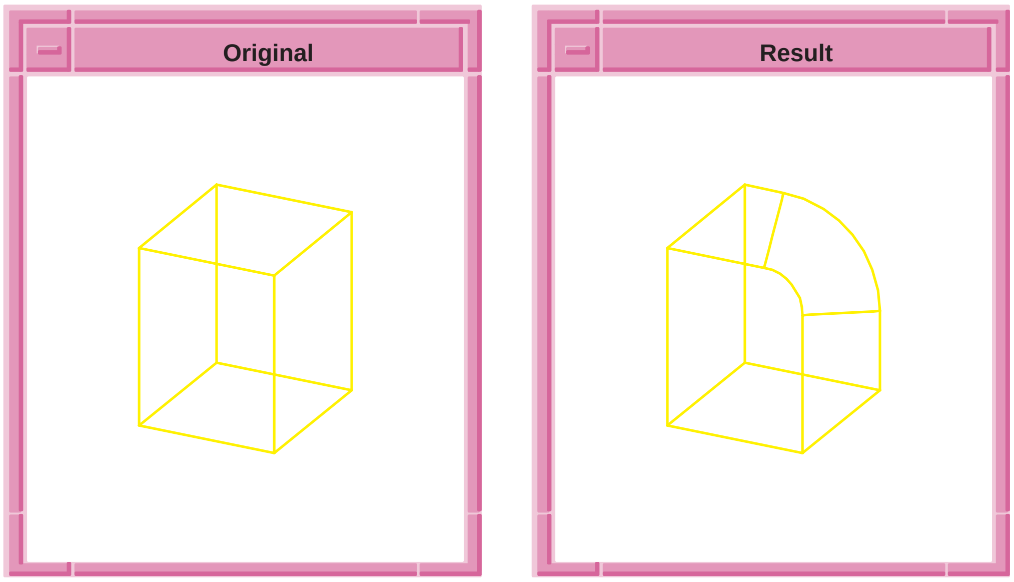


Figure 3-19. **`abl:two-ends-rad`**

abl:var–round

Scheme Extension: Blending

Action:	Attaches a smooth variable radius blend instruction to an entity.
Filename:	abl/abl_scm/abl_scm.cxx
APIs:	`api_abh_vblend, api_fix_blends, api_get_owner, api_smooth_edge_seq`

Syntax:

```
(abl:var–round edge radius–1 radius–2
    [”fix” | ”single”] [acis–opts])
```

Arg Types:

edge	edge
radius–1	real
radius–2	real
“fix”	string
“single”	string
acis–opts	acis–options

Returns: boolean

Errors: None

Description: This extension attaches a smooth variable radius blend instruction. The radius function is a smooth, cubic transition.

edge is an input edge to which blend is applied.

radius–1 is a radius value at first edge.

radius–2 is a radius value at last edge.

fix is an input string value.

single is an input string value.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; abl:var-round
; Create block topology to illustrate this command.
(define block1 (solid:block
    (position –20 –10 –30) (position 15 25 10)))
;; block1
; OUTPUT Original

(define eds1 (entity:edges block1))
;; eds1
(define blend1 (abl:var–round
    (list–ref eds1 9) 2 3 ”fix”))
;; blend1
(define eds2 (entity:edges block1))
;; eds2
(define blend2 (abl:var–round
    (list–ref eds2 8) 2 3 ”fix”))
;; blend2
; OUTPUT Result
```

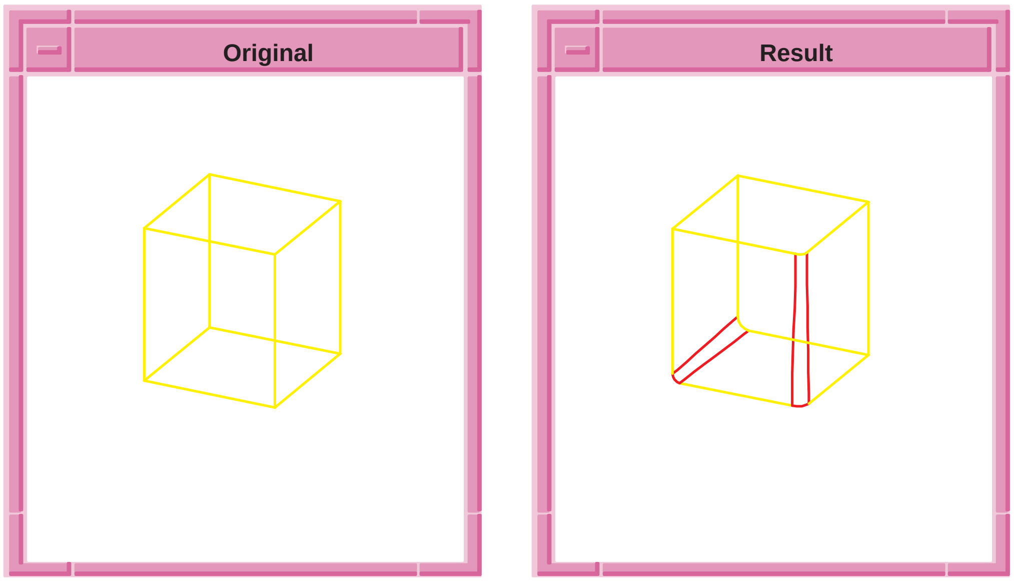


Figure 3-20. **`abl:var-round`**

solid:blend–edges–pos–rads

Scheme Extension:	Blending
Action:	Blend edges of a solid by defining position–radius pairs.
Filename:	abl/abl_scm/abl_scm.cxx
APIs:	api_blend_edges_pos_rad

Syntax:

```
(solid:blend-edges-pos-rads edge-list
    position-list radii-list [start-slope
[end-slope]]
    [acis-opts])
```

Arg Types:

edge–list	edge | (edge ...)
position–list	position | (position ...)
radii–list	real | (real ...)
start–slope	real
end–slope	real
acis–opts	acis–options

Returns: body

Errors: Not applicable

Description: This Scheme extension creates a calibration curve and a radius object, applies variable radius attributes, and fixes the blend. It also attempts to simplify the blend by substituting the variable radius attribute with the constant radius blend for the edges. The radius function is constant. When simplification fails, a regular variable radius blend is conducted.

edge–list defines one or more edges to be blended. position–list defines the positions (specified in the global coordinate system) to be projected on the calibration curve, composed of the given edges, to find the calibration curve parameter values corresponding to the given radii. radii–list defines one or more radii to be paired with each of the defined positions in the position–list.

The optional start_slope and end_slope arguments specify the derivative of the radius function with respect to the parameter along the calibration curve. They make sense only for open calibration curves. As the parameter speed is set arbitrarily (or, more accurately, depending on the first edge in the sequence), only zero slope value is safe and meaningful.

edge–list defines one or more edges to be blended.

position–list defines the positions to be projected on the calibration curve.

radii–list defines one or more radii to be paired with each of the defined positions in the pos–list.

start–slope specify the derivative of the radius function with respect to the parameter along the calibration curve.

end–slope specify the derivative of the radius function with respect to the parameter along the calibration curve.

acis–opts contains journaling and versioning information.

Limitations: None

Example:
```
; solid:blend-edges-pos-rads
; Define an xyz plus radius.
(define N 9)
;; N
(define H 10)
;; H
(define R (* 0.4 H))
;; R
; Define a solid prism.
(define p (solid:prism (* 4 H) H H N))
;; p
; Define the edges for the prism.
(define all_edges (entity:edges p))
;; all_edges
(define blend1 (solid:blend-edges all_edges R))
;; blend1
; Define a solid block.
(define b (solid:block (position (- 0 H) (- 0 H)
    (- 0 H)) (position H H 0)))
;; b
(define intersect (solid:intersect p b))
;; intersect
; Blend bottom edges with some CONST radius sections.
(define face (pick:face (ray
    (position 0 0 (* H -0.5)) (gvector 0 0 -1))))
;; face
(define bottom_edges (entity:edges face))
;; bottom_edges
(define z0 (- 0 H))
;; z0
; Define list of positions.
(define pos_list (list (position H 0 z0)
    (position 0 H z0) (position (- 0 H) 0 z0)
    (position 0 (- 0 H) z0)))
;; pos_list
; Define the radius list.
(define rad_list (list (* 0.7 R) (* 0.7 R)
    (* 0.1 R) (* 0.1 R)))
;; rad_list
; OUTPUT Original
; Blend the edges using the defined pairs.
(define blend2 (solid:blend-edges-pos-rads
    bottom_edges pos_list rad_list))
;; blend2
; OUTPUT Result
```

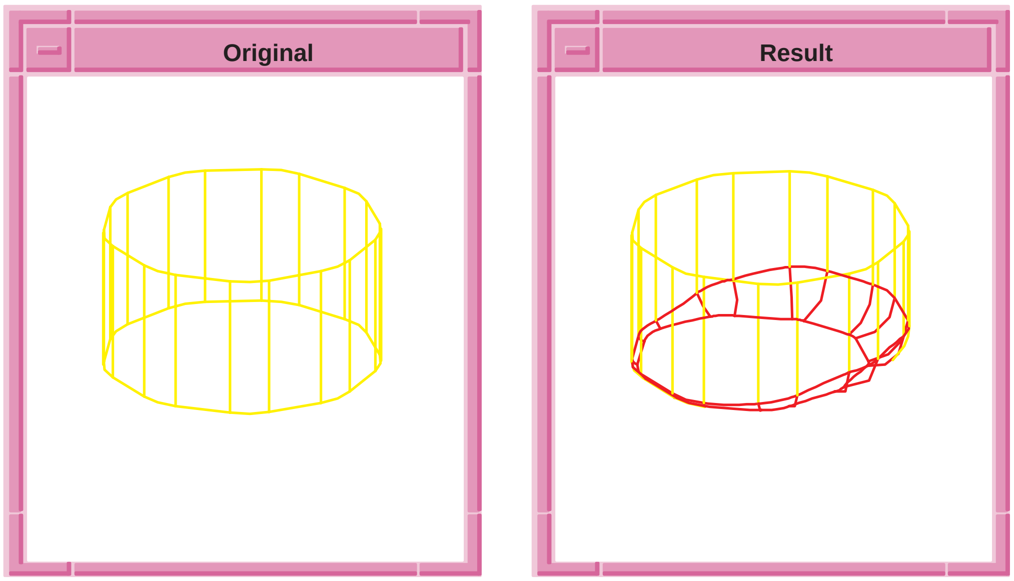


Figure 3-21. solid:blend–edges–pos–rads

solid:var–chamfer–edges

Scheme Extension: Blending

Action: Creates a variable chamfer on the defined edges.

Filename: abl/abl_scm/abl_scm.cxx

APIs: api_abh_chamfer_edges

Syntax:
```
(solid:var-chamfer-edges entity-list
    left-beginning-range [right-beginning-range]
    [left-end-range] [right-end-range]
    [left-round-chamfer-radius]
    [right-round-chamfer-radius] [acis-opts])
```

Arg Types:	entity–list	entity | (entity ...)
	left–beginning–range	real
	right–beginning–range	real
	left–end–range	real
	right–end–range	real
	left–round–chamfer–radius	real
	right–round–chamfer–radius	real
	acis–opts	acis–options

Returns: (entity ...)

Errors: None

Description: This extension creates variable chamfers on the defined edges using the supplied left and right ranges. Left and right are with respect to the edge direction. For a chamfer with equal left and right ranges, the points where the chamfer surfaces meet the faces of the chamfered edge are the same as those for a round with radius equal to the chamfer range. If the ranges for left and right differ, the meeting points on the left face are the same as those for a round with radius equal to the left range. Likewise, the meeting points on the right face are the same as those for a round with radius equal to the right range.

The argument entity–list specifies a list of solid body edges to be chamfered. If right–beginning–range is omitted, a range equal to left–beginning–range is used. The same is true for the range values at the end of the edge: if the range value for the edge end is omitted, the value for the beginning of the edge is used. The round chamfer radii (R1 and R2) are assumed to be zero when not defined, which corresponds to the planar chamfer. The negative round chamfer radii result in a concave chamfer, and the positive values result in a convex chamfer surface.

This extension returns a list of entity owners of all the edges in the entity list. If all the edges in the entity list belong to a single solid, this extension returns a list with only one element (the solid).

entity–list specifies a list of solid body edges to be chamfered.

left–beginning–range is the left beginning range for chamfering.

right–beginning–range is the right beginning range for chamfering.

left–end–range is the left end range for chamfering.

right–end–range is the right end range for chamfering.

left–round–chamfer–radius is the chamfer radius at the left end.

right–round–chamfer–radius is the chamfer radius at the right end.

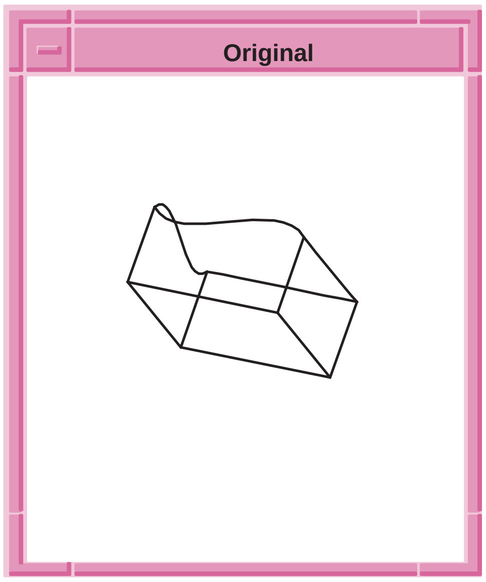
acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; solid:var–chamfer–edges
; Create a wiggle to illustrate command.
(define wiggle1 (solid:wiggle 40 40 20 –1 0 1 0))
;; wiggle1
; Chamfer a top edge.
(define edge (pick:edge (ray (position 0 0 0)
    (gvector 0 20 10))))
;; edge
; Define details of rotation.
(define rotate (transform:rotation
    (position 0 0 0) (gvector –5 0 0) 45))
;; rotate
; Apply the rotation to the wiggle1 entity.
(define apply (entity:transform wiggle1 rotate))
;; apply
; OUTPUT Original

; Execute the chamfer command.
(define chamfer (solid:var–chamfer–edges
    (list edge) 10 5 0.5 0.5))
;; chamfer
; OUTPUT Result
```



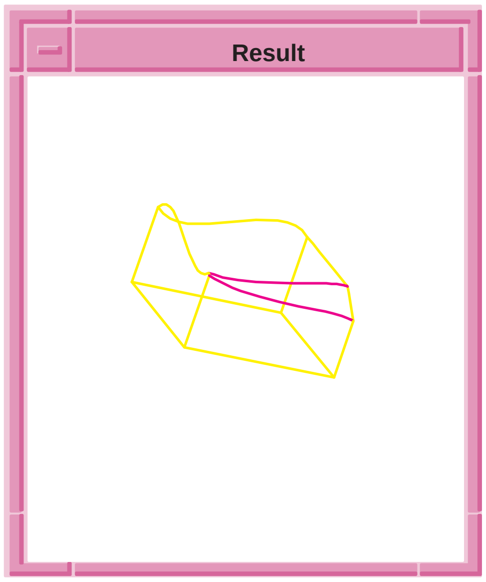


Figure 3-22. **solid:`var-chamfer-edges`**