Chapter 1.
Sweeping Component

Component: *Sweeping

The Sweeping Component (SWP), in the swp directory, creates solid or sheet bodies by sweeping a *profile*, or shape, along some *path*.

A profile can be an edge, a face, a wire body, or a planar sheet body (with non–adjacent faces). Profiles can be swept along a distance, around an axis, along a vector, or along an edge or a wire body. The best results are obtained when the profile is swept along an edge or wire body.

The sweep options allow additional control over sweeping behavior. Sweep options, explained in the following pages enable drafting, twisting, and other advanced behaviors.

Sweeping Basics

Topic: *Sweeping

The function api_sweep_with_options and the associated Scheme extension sweep:law encompass the sweeping functionality.

The api_sweep_with_options function is used to simplify sweeping in C++. Various options to control sweeping may be specified through the sweep_options class and the associated Scheme extension sweep:options. The function api_sweep_with_options has four overloaded instances:

Sweep along a path . . . . . The API arguments are: profile, a pointer to a path entity, sweep options, and a pointer to the new body (if applicable).

Sweep a distance . . . . . The API arguments are: profile, distance (in the direction of the profile face normal), sweep options, and a pointer to the new body (if applicable).

Sweep along a vector . . . . . The API arguments are: profile, vector (to sweep along), sweep options, and a pointer to the new body (if applicable).

Sweep around an axis . . . . . The API arguments are: profile, position (of the root of the sweep axis), vector (of the direction of the sweep axis), sweep options, and a pointer to the new body (if applicable).

The first argument to the api_sweep_with_options function is a profile in the form of a pointer to an entity. For example, an edge, a wire body, a face, or a planar sheet body with nonadjacent faces, which may or may not belong to a body, could form the sweeping profile.

The next argument is a path. The path may take the form of an EDGE or WIREBODY pointer. The path may be specified as a distance if the profile is a planar face or a planar sheet body with non–adjacent faces. A vector or an axis to sweep around (in the form of a position and a vector) may also be provided as the path. However, defining vector or axis as paths requires the use of the default rail law (without twist).

All other inputs are passed to the API through an instance of the sweep_options class. The sweep options are explained in detail in the next section.

The returned body depends on the profile type:

Face belonging to a body . . . . . When the profile is a face that belongs to a body, that owning body is altered, and the new body points to NULL.

Face not belonging to a body . . . . . When the profile is a face not already belonging to a body, a new body is created and returned. The original face remains separate and available for further independent manipulation or deletion.

Wire body . . . . . When the profile is a wire body, the owning body is altered and the new body points to NULL. The altered body can be a solid or a sheet body depending on whether or not the option to make a solid has been set, and whether or not the wire body is closed. It is not necessary or desirable to cover the wires. The default is to make a solid if the wire body is closed.

Edge not belonging to a body . . . . . When the profile is an edge or set of independent edges, a new body is created. The original edge remains available for further independent manipulation or deletion. If the edge is closed, by default it makes a solid body; otherwise it makes a sheet body. It is not necessary or desired that the edges be covered.

Locating the profile at the beginning of the sweep path is not always practical and is not a requirement when using sweeping. The profile's location is determined in the following manner:

Planar profile . . . . . The profile is located at the nearest intersection point on the path, within the profile plane.

Linear profile . . . . . The profile is always located at the beginning of the path.

The plane of the profile must intersect the path in the intended location. If the plane of the profile does not intersect the path at all, the relationship is ambiguous and sweeping may not complete, or may create unintended results. If the plane of the profile intersects the path at an unintended location, this location is used in determining the relationship of profile and the path.

These rules work well when the profile is perpendicular to the path. However, if the given profile is not perpendicular to the path, the path location may not be unique. Applications should move the profile to the beginning of the path and use the AS_IS argument for the portion sweep option.

The most basic sweep activities are sweeping a planar face, a solid face, a wire, along a path or vector, and revolving a face or wire body. The following examples demonstrate those capabilities.

Revolve a Face

Topic: *Sweeping, Laws

Use the Scheme extension sweep:options ("sweep_angle") to revolve a face.

Scheme Example

```
(define block1 (solid:block (position –10 –10 0)
    (position 25 25 25)))
(define entities (entity:faces block1))
(define face1 (car (cdr (cdr entities))))
(define fix (entity:fix–transform block1))
;; OUTPUT Original
```

```
(define opts (sweep:options ”sweep_angle” 60))
(define sweep1 (sweep:law face1
    (position –10 –10 –10)
    (gvector 1 0 0) opts))
;; OUTPUT Result
```

Example 1-1. Revolve a Face

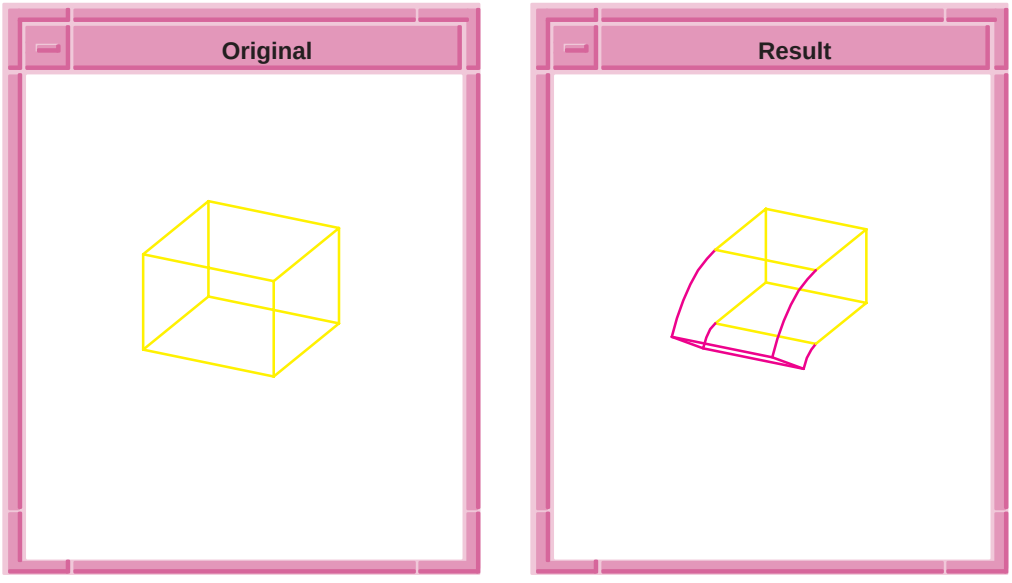


Figure 1-1. Revolve a Face

Revolve a Wire Body

Topic: *Sweeping, Laws

Use the Scheme extension sweep:options with the (“sweep_angle”) option to revolve a wire body.

Scheme Example

```
(define edge1 (edge:linear (position 0 0 0) (position 20 0 0)))
(define edge2 (edge:linear (position 20 0 0) (position 20 20 0)))
(define edge3 (edge:linear (position 20 20 0) (position 0 20 0)))
(define edge4 (edge:linear (position 0 20 0) (position 0 0 0)))
(define wirebody (wire–body (list edge1 edge2 edge3 edge4)))
; OUTPUT Original
```

```
; Create a solid by revolving the wire body.
(define sweep1 (sweep:law wirebody (position 0 0 0)
    (gvector 0 1 0) (sweep:options ”sweep_angle” 60)))
:OUTPUT Result
```

Example 1-2. Revolve a Wire Body

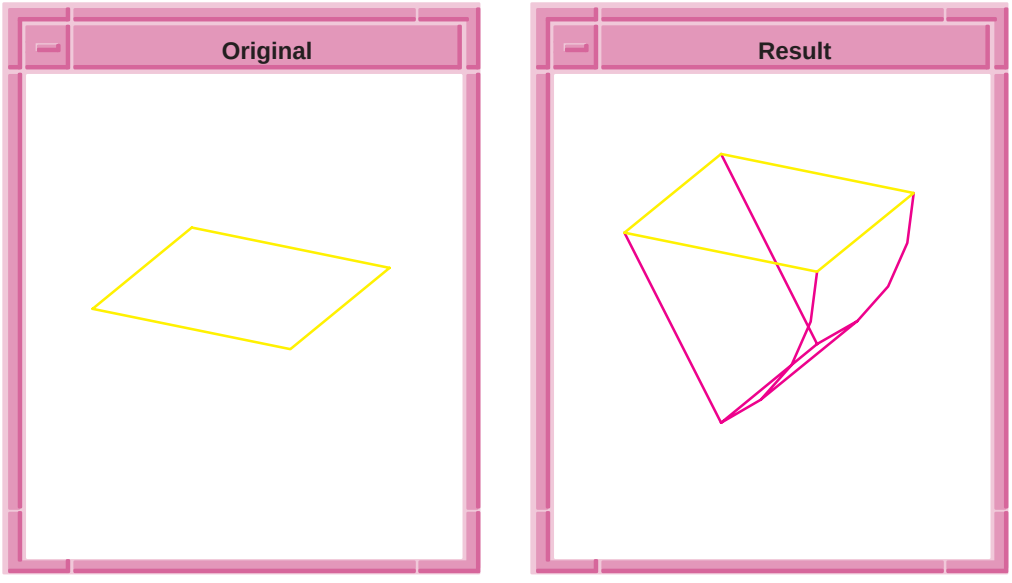


Figure 1-2. Revolve a Wire Body

Sweep a Planar Face

Topic: *Sweeping, Laws

Use the Scheme extension sweep:options with the (“draft_angle”) option to sweep a planar face. Notice the range of sweeps (in each of the three examples) when the sweep:law angle is changed. Also, notice the effects when changing the values for draft_angle from positive to negative. As a side note, this set of examples provides a perfect opportunity to demonstrate some of the uses for the journal commands.

Scheme Example

```
(journal:on ”blub.jrl”)
(define block1 (solid:block (position –10 –10 –10)
    (position 30 30 30)))
(define block1 (entity:faces block1))
(define faces (entity:faces block1))
(define face1 (car (cdr (cdr faces))))
(journal:off)
(journal:save ”blub.jrl”)
;; OUTPUT Original

; Create a new solid by sweeping the face along a gvector.
(define sweep1 (sweep:law face1 10.0
    (sweep:options ”draft_angle” –45)))
; OUTPUT Result with Negative Draft Angle

; This portion clears the entities, reloads the basic block, and
; runs a new slant on the sweeping/law example.
(part:clear)
(journal:load ”blub.jrl”)
; Note the change in distance defined to sweep along the face
; normal.
(define sweep1 (sweep:law face1 15.0
    (sweep:options ”draft_angle” –45)))
;; Modified Sweep Distance
; OUTPUT Result with Negative Draft Angle

; This portion clears the entities, reloads the basic block, and
; runs a third slant on the sweeping/law example.
(part:clear)
(journal:load ”blub.jrl”)
; This time notice the effect of a positive draft_angle.
(define sweep1 (sweep:law face1 15.0
    (sweep:options ”draft_angle” 45)))
;; Modified ”draft_angle”
```

Example 1-3. Sweep a Planar Face

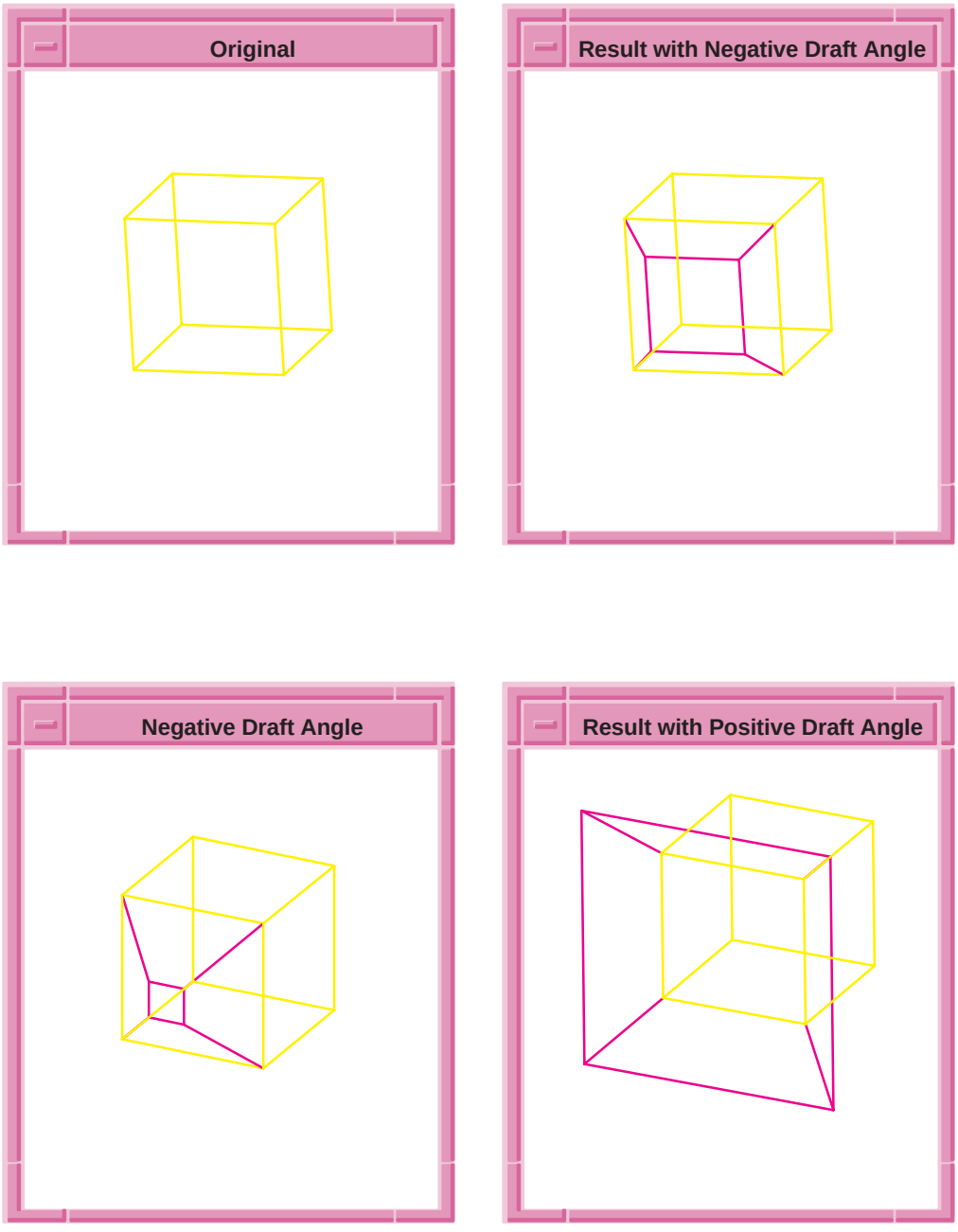


Figure 1-3. Sweep a Planar Face

Sweep a Solid Face

Topic: *Sweeping, Laws

Use the Scheme extension sweep:options with a *(positive)* “draft_angle” to sweep a solid face. This example identifies a single face of a solid block, uses a gvector to define the extent of the sweep, and further defines the sweep with a 15 degree outward angle.

Scheme Example

```
; Sweeping a solid face.
(define block1 (solid:block (position –10 –10 –10)
    (position 30 30 30)))
(define faces (entity:faces block1))
(define face2 (car faces))
;; OUTPUT Original

; Create a new solid by sweeping the face along a gvector.
(define sweep2 (sweep:law face2 (gvector 0 0 10)
    (sweep:options ”draft_angle” 15)))
;; OUTPUT Result
```

Example 1-4. Sweep a Solid Face

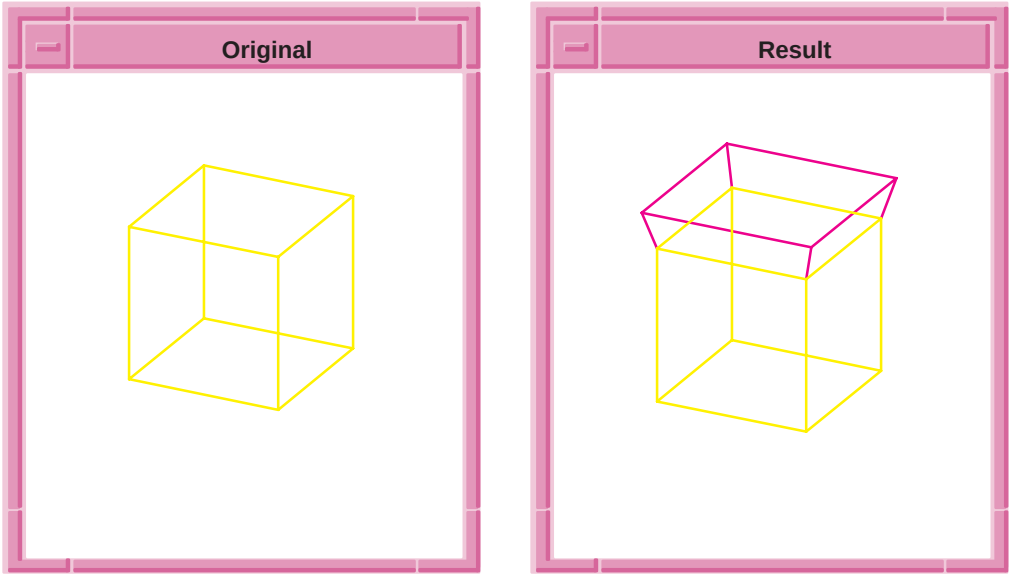


Figure 1-4. Sweeping a Face with Positive Draft Angle

Sweep a Wire

Topic: *Sweeping, Laws

Use the Scheme extension sweep:options with the “draft_angle” option to sweep a wire. This example shows converting four wires into a wire body, creating a path to sweep, and then creating the final solid by sweeping the wire body.

Scheme Example

```
(define edge_1 (edge:circular (position 0 0 0) 25 0 180))
(define edge_2 (edge:circular (position 0 0 0) 25 180 270))
(define edge_3 (edge:linear (position 0 0 0) (position 25 0 0)))
(define edge_4 (edge:linear (position 0 0 0) (position 0 –25 0)))
(define w_body (wire–body (list edge_4 edge_3 edge_1 edge_2)))
(define path1 (edge:linear (position 0 0 0) (position 0 0 7.5)))
; OUTPUT Original

; Create a solid by sweeping the wire body.
(define sweep1 (sweep:law w_body path1
    (sweep:options ”draft_angle” –10)))
; OUTPUT Result
```

Example 1-5. Sweep a Wire

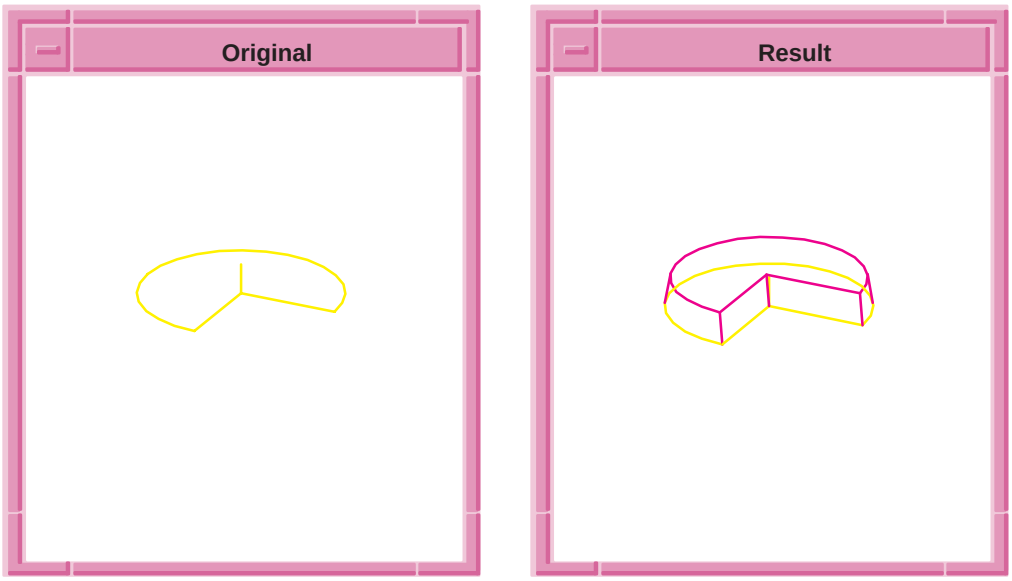


Figure 1-5. Sweeping a Wire

Sweep Along a Path

Topic: *Sweeping, Laws

Use the Scheme extension sweep:options with “draft_angle” and “to_face” options to sweep along a path. In this example, the face is swept along the path (defined by an edge) onto a surface (defined as a planar face). Watch the effect of the negative draft angle on the sweep. Also note how the sweep terminates at the planar face instead of continuing the path to the end of the linear edge.

Scheme Example

```
; sweeping a path
(define block1 (solid:block (position –45 –15 –15)
    (position –15 15 15)))
(define facelist (entity:faces block1))
(define profile1 (car (cdr (cdr (cdr (cdr (cdr facelist)))))))
(define edge1 (edge:linear (position –15 0 0)
    (position 20 0 3)))
(define planarface (face:planar–disk
    (position 10 0 0) (gvector –1 0 0) 50))
; OUTPUT Original

(define sweep1 (sweep:law profile1 edge1
    (sweep:options ”to_face” planarface ”draft_angle” –15)))
; OUTPUT Result
```

Example 1-6. Sweep Along a Path

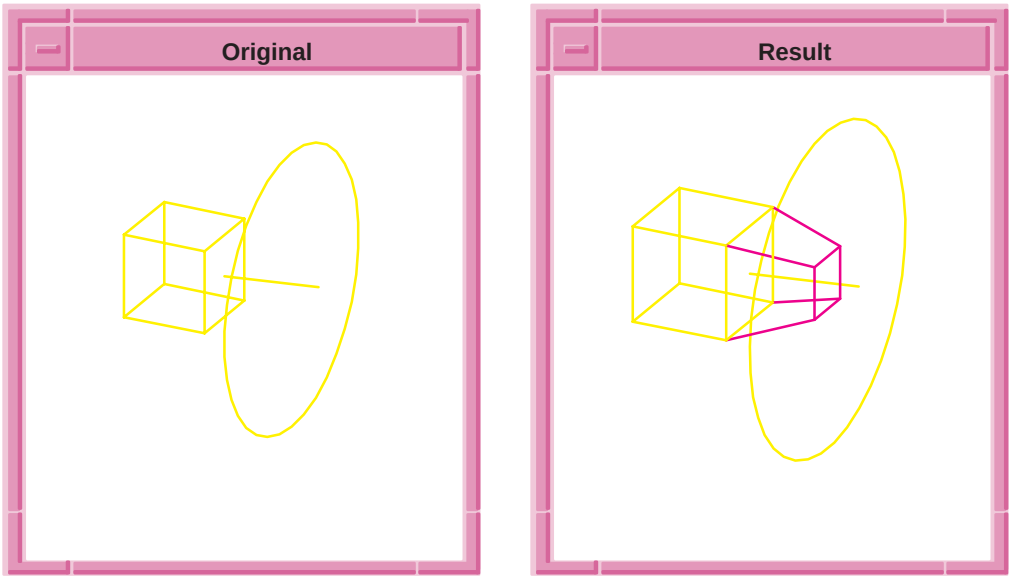


Figure 1-6. Sweep Along a Path

Sweep Along a Vector

Topic: *Sweeping, Laws

Use the Scheme extension sweep:options (“draft_angle”) to sweep along a vector.

Scheme Example

```
; Sweeping along a vector
(define face1 (face:plane (position –20 10 10) 30 30
    (gvector 10 0 0)))
;; OUTPUT Original

; Sweep a planar face along a vector with draft.
(define sweep1 (sweep:law face1 (gvector 38 0 20)
    (sweep:options ”draft_angle” 20)))
;; OUTPUT Result
```

Example 1-7. Sweep Along a Vector

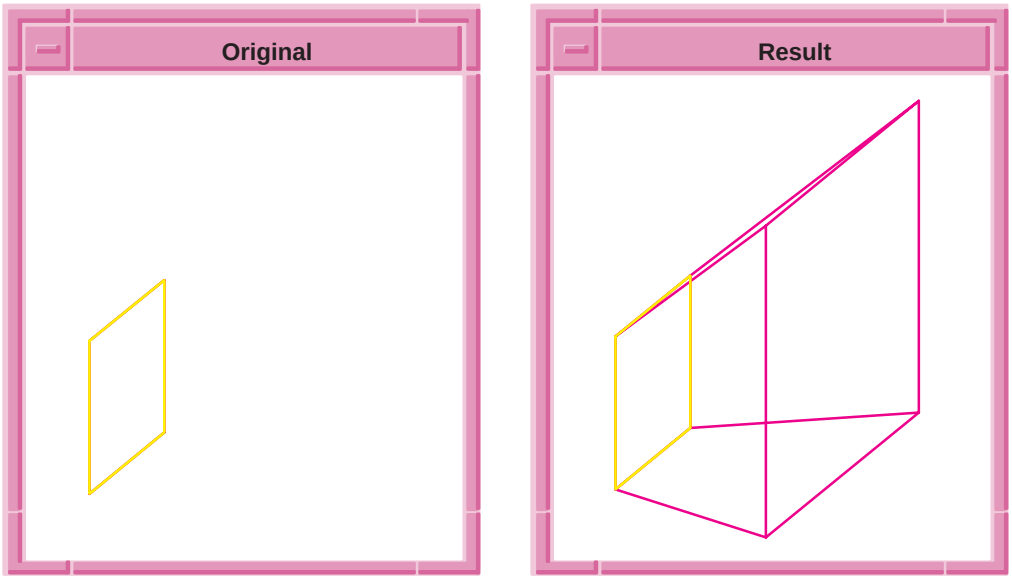


Figure 1-7. Sweep Along a Vector

Sweeping with Options

Topic: *Sweeping, Laws

The sweep_options class is used for each overloaded api_sweep_with_options call. To use anything other than the default options, an instance of this class should be created. The data structure within this class establishes default values for the options that are not set by the caller. The options are set using a set_<option name> methods present for each of the options in the class. The same names are used for the API and the Scheme extension sweep:options. In Scheme, the option names are passed into the Scheme extension within quotation marks. The following sweep options are grouped based on the functionality controlled by the options.

Table 1-1. Sweeping Options Summary

Sweeping Options		
Categories	**Options**	**Explanation**

Sweeping Options		
Categories	**Options**	**Explanation**
Sweep To:	sweep_to_body	Sweeping to a body allows the specification of the end of the sweep via the location where the sweep intersects the body specified. When these options are used, the result of the sweep is returned in the new_body argument to api_sweep_with_options.
Sweep to body results	bool_types keep_branches keep_law	This set of options controls what portions of the sweep result and body being swept to remain after the sweep operation is completed.
Object to sweep to	to_face	Used for backward compatibility. Instead of specifying a body to be used for clipping the end of the sweep, a surface underneath a face is used. The disadvantage of using this option is the surface of the face must be defined over the entire area that the sweep intersects.
Sweeping with Draft		Allows the profile to be expanded and contracted as it is swept along the path. Drafting works best when sweeping along linear paths. For non–linear paths, the amount of draft is calculated based on taking the parametric length along a straight path and applying this offset at the equivalent parameters along the non–linear path. This may cause the result to appear different than expected.
Draft amount	draft_angle end_draft_dist draft_law	The amount by which the profile is offset along the path is specified by these mutually exclusive options.

Sweeping Options		
Categories	**Options**	**Explanation**
Draft Results	draft_hole draft_repair gap_type	Additional options for further specification of draft capability.
Sweeping about an axis	sweep_angle steps close_to_axis	These options are used in conjunction with the api_sweep_options accepting the path in the form of a position and vector which define an axis.
Profile Orientation and Twisting	rail_law rigid	Controls how the profile is oriented while it is swept along the path.
Sweeping with twist	twist_angle twist_law	Allows the profile to be rotated perpendicular to the path as it moves along the path.
Path Used	portion	Allows for greater granularity to be defined when specifying an entity as the path.

There are six basic areas of consideration for sweeping (as outlined in the Sweeping Options table). Sweeping To: options define how the sweep ends. This includes how and when specific pieces considered for keeping or eliminating are handled and whether the sweep ends at an object (versus sweeping a distance, for example). The Draft options define the construction and results of a draft sweep. This also includes how specific pieces are handled. Axis Sweeping options also define basic sweep functions except to be attached to an axis. Profile Orientation and Twisting allows the Sweep functionality to include twisted profiles and rail laws. The Path Used capability allows the sweep to be applied with a finer granularity when specifying an entity path. The Sweep Results options help further define results such as the ending of the sweep, mitering, and what to do with the leftovers.

Sweep To:

Topic: Sweeping

Sweeping to a body allows the specification of the end of the sweep via the location where the sweep intersects the body specified. When these options are used, the result of the sweep is returned in the new_body argument to api_sweep_with_options.

Sweeping to a Body

The option sweep_to_body specifies a pointer to the body where the sweep is to complete. sweep_to_body allows multiple faces on a body to be used for properly clipping the result(s). The to_face option only allows a single surface to be used to clip the sweep result. Equivalent results could be obtained if a profile were swept through a body, and if then the two bodies were Boolean united together. The default is NULL.

The sweep_to_body option is mutually exclusive with to_face.

Figure 1-8 shows a circular profile swept to a bracket.

Scheme Example

```
; Sweep to a body
(define bracket1 (solid:block (position 0 0 0)
    (position 10 10 10)))
(define b2 (solid:block (position 0 0 5) (position 7 10 10)))
(define b3 (solid:block (position 3 0 2) (position 7 10 10)))
(solid:subtract bracket1 b2)
(solid:subtract bracket1 b3)
(define profile (edge:ellipse (position –5 5 5)
    (gvector 1 0 0) 2))
; OUTPUT Original

(define opts1 (sweep:options ”sweep_to_body” bracket1
    ”bool_type” ”subtract” ”keep_law” ”true” ”keep_branches” #t))
(define sweep1 (sweep:law profile (gvector 20 0 0) opts1))
; Run the rebuild command to update the visual graphics.
(render:rebuild)
;; ()
; OUTPUT Result
```

Example 1-8. Sweeping to a Body

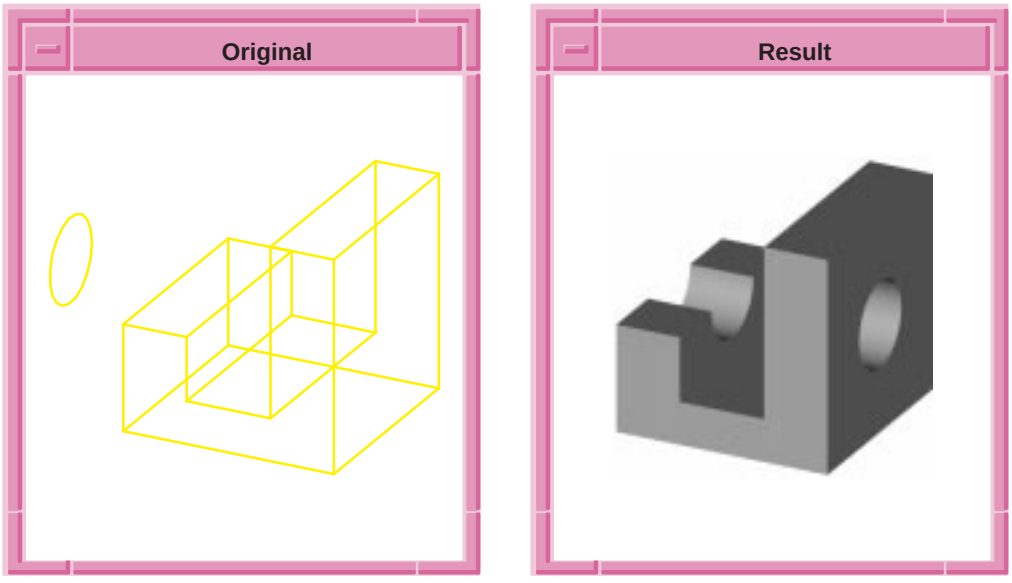


Figure 1-8. Sweep to a Body

Using Selective Booleans

Topic: *Sweeping, Laws

The option bool_type works with sweep_to_body to specify what should be done with the pieces of the profile that were removed during the sweep. This specifies how the selective Booleans are used within sweeping.

LIMIT . . . . . creates a body which is the result of the sweep up to the location of the intersection between the sweep and the fixed body. This the default setting.

UNITE . . . . . creates one body out of the fixed body and the result of the sweep up to the location of the intersection between the sweep and the fixed body.

INTERSECT . . . . . creates one or more pieces resulting from the intersection of the fixed body with the swept body.

SUBTRACT . . . . . modifies the fixed body by removing material equivalent to the swept body.

KEEP_BOTH . . . . . creates a body equivalent to the limit type without deleting the original body.

bool_type returns the created body, which is the result of the sweep up to the location of the intersection between the sweep and the fixed body, without deleting the fixed body. bool_type also controls the numbering on the cells of the tool body and the blank body. The number of cells can affect the results of a defined keep_law.

Figure 1-9 illustrates the results of defining the bool_type as ”limit”. What was thrown away in Figure 1-8, is precisely what is kept here.

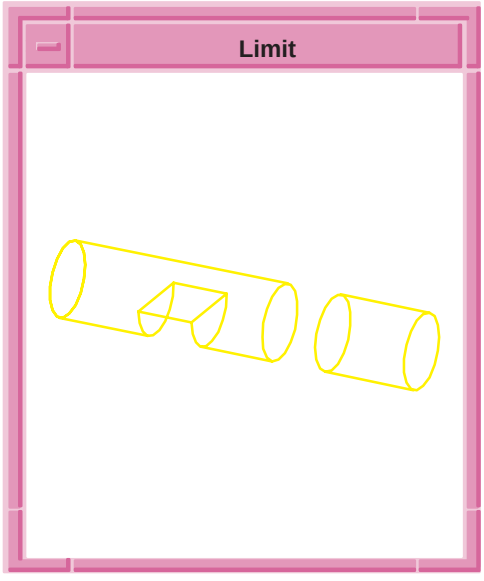


Figure 1-9. Sweeping with Boolean Types

Branches

Topic: Sweeping

The keep_branches and keep_law options work with the graph created when sweeping to a body. The keep_branches option specifies whether or not graph branches should be kept. TRUE means all branches are kept. The sweep_to_body option must be specified when using keep_branches.

Figure 1-10 shows a circular profile swept to a bracket. The sweep options used are bool_type, to_body, keep_law, and keep_branches. The to_body sweep option specifies that sweeping is to stop at the bracket body. The bool_type sweep option is set to SUBTRACT which means that any overlapping portion of the circular profile’s volume is removed from the bracket. The rendered illustration on the right shows what is desired. This example introduces some ambiguities where the swept profile only partially intersects the bracket body.

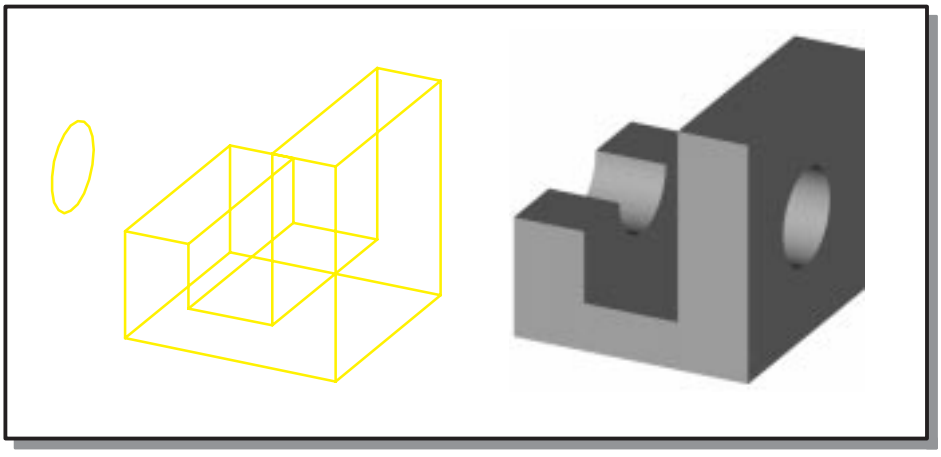


Figure 1-10. Keeping Sweep Branches

Figure 1-11 shows how the graph theory decomposes the tool (cylinder) body into graph vertices and graph edges. Most of the tool body is linear. The option keep_branches defines whether the branches are important and should be kept or should be ignored and thrown away. In this case, the branch is the half circle resulting from the tool cylinder body intersecting the bracket body. (For more information on graph theory, refer to the *Kernel Component Manual*.)

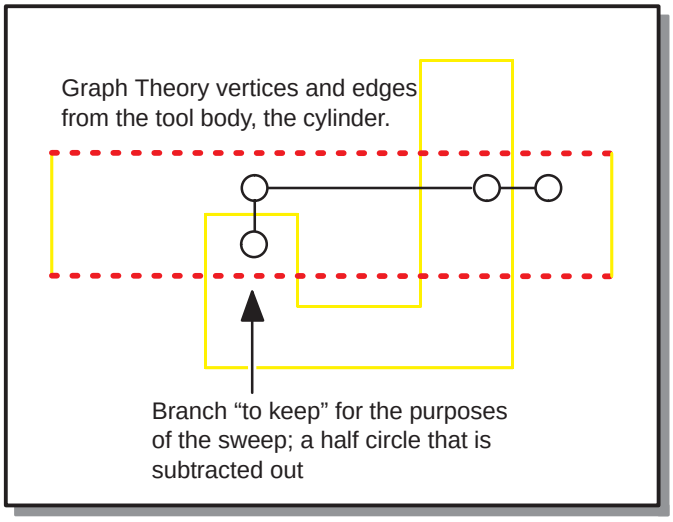


Figure 1-11. Keeping Branches Graph Theory

Figure 1-12 shows graph representing a blank body and a tool body. The tool body can be thought of as a circular profile that is swept through the blank body. If a keep_law is to be used to specify which cells are to remain after the sweep operation, the law needs to be aware of where the cell numbering begins.

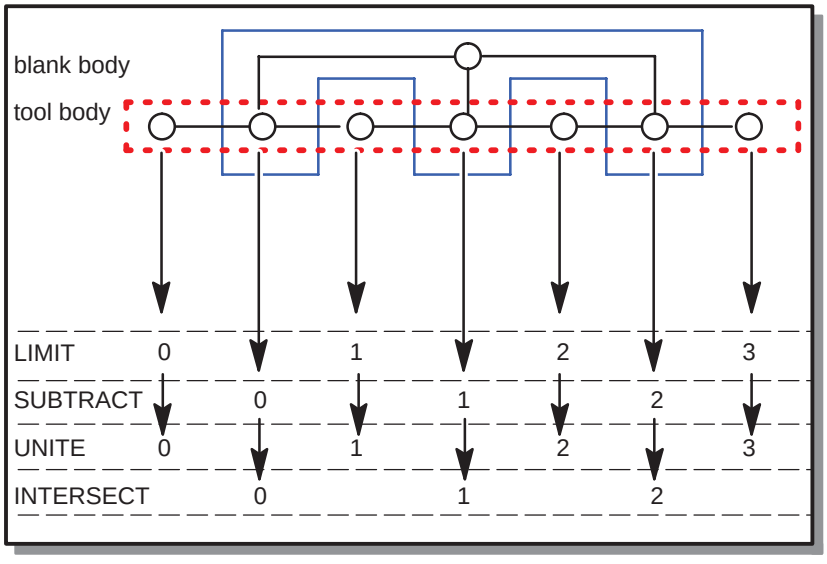


Figure 1-12. Bool_Type Affects Cell Numbering

Using the Keep Law

Topic: Sweeping, Laws

The option keep_law allows more specific definition of the portions of a graph to be kept than the sweep option sweep_to_body. Laws, primarily conditional tests, can be used to specify what is used for the sweep operation. When keep_law is TRUE, all graph pieces are saved. (For more information on graph theory, refer to the *Kernel Component Manual*.)

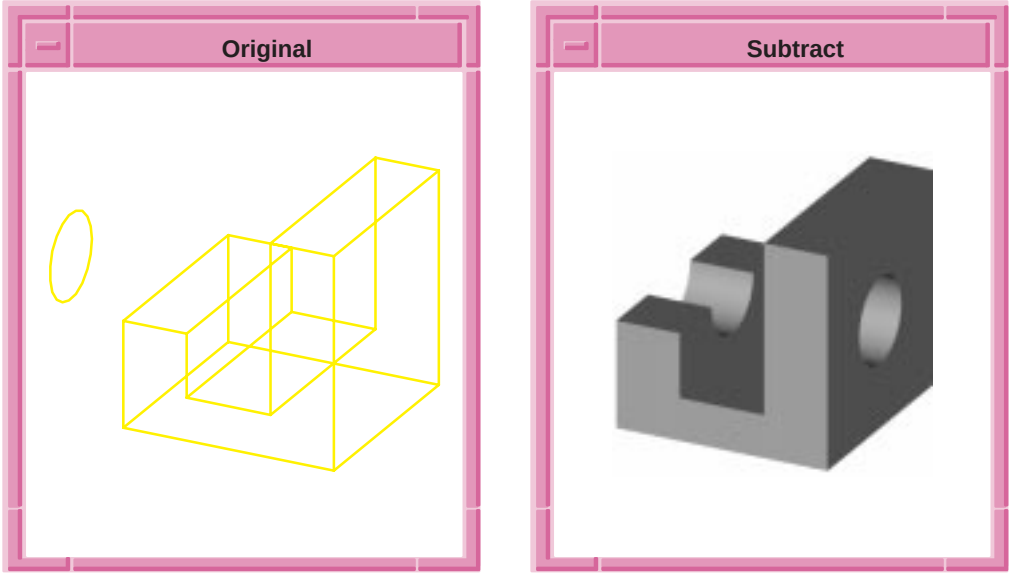
When using something other than TRUE for keep_law, the graph elements are first ordered and numbered. bool_type directly impacts the cell numbering of the tool body and the blank body. The starting cell number and the cells that are included in the number are different depending on the bool_type operation. Knowing the number associated with the specified bool_type then provides insight into a conditional law defined for the keep_law, like "x>2 AND x<5". The letter x should be used for the pieces to keep. The letter y is always the last numbered piece. The sweep_to_body option must be specified when using keep_law.

Scheme Example

```
; Sweeping with boolean types
(define bracket1 (solid:block (position 0 0 0)
    (position 10 10 10)))
(define b2 (solid:block (position 0 0 5) (position 7 10 10)))
(define b3 (solid:block (position 3 0 2) (position 7 10 10)))
(define subtract1 (solid:subtract bracket1 b2))
(define subtract2 (solid:subtract bracket1 b3))
(define profile (edge:ellipse (position –5 5 5)
    (gvector 1 0 0) 2))
; Run the rebuild command to update the visual graphics.
(render:rebuild)
;; ()
(roll:name–state ”original”)
; OUTPUT Original


(define opts1 (sweep:options ”sweep_to_body” bracket1
    ”bool_type” ”subtract” ”keep_law” ”true” ”keep_branches” #t))
(define sweep1 (sweep:law profile (gvector 20 0 0) opts1))
; Run the rebuild command to update the visual graphics.
(render:rebuild)
;; ()
; OUTPUT Subtract


(roll ”original”)
(render:rebuild)
(define opts2 (sweep:options ”sweep_to_body” bracket1
    ”bool_type” ”unite” ”keep_law” 1 ”keep_branches” #t))
(define sweep2 (sweep:law profile (gvector 20 0 0) opts2))
; Run the rebuild command to update the visual graphics.
(render:rebuild)
;; ()
; OUTPUT Unite


(roll ”original”)
(render:rebuild)
(define opts3 (sweep:options ”sweep_to_body” bracket1
    ”bool_type” ”intersect” ”keep_law” 1 ”keep_branches” #t))
(define sweep3 (sweep:law profile (gvector 20 0 0) opts3))
; Run the rebuild command to update the visual graphics.
(render:rebuild)
;; ()
; OUTPUT Intersect
```

```
(roll ”original”)
(render:rebuild)
(define opts4 (sweep:options ”sweep_to_body” bracket1
    ”bool_type” ”limit” ”keep_law” 1 ”keep_branches” #t))
(define sweep4 (sweep:law profile (gvector 20 0 0) opts4))
; Run the rebuild command to update the visual graphics.
(render:rebuild)
;; ()
; OUTPUT Limit
```

Example 1-9. Using the Keep Law



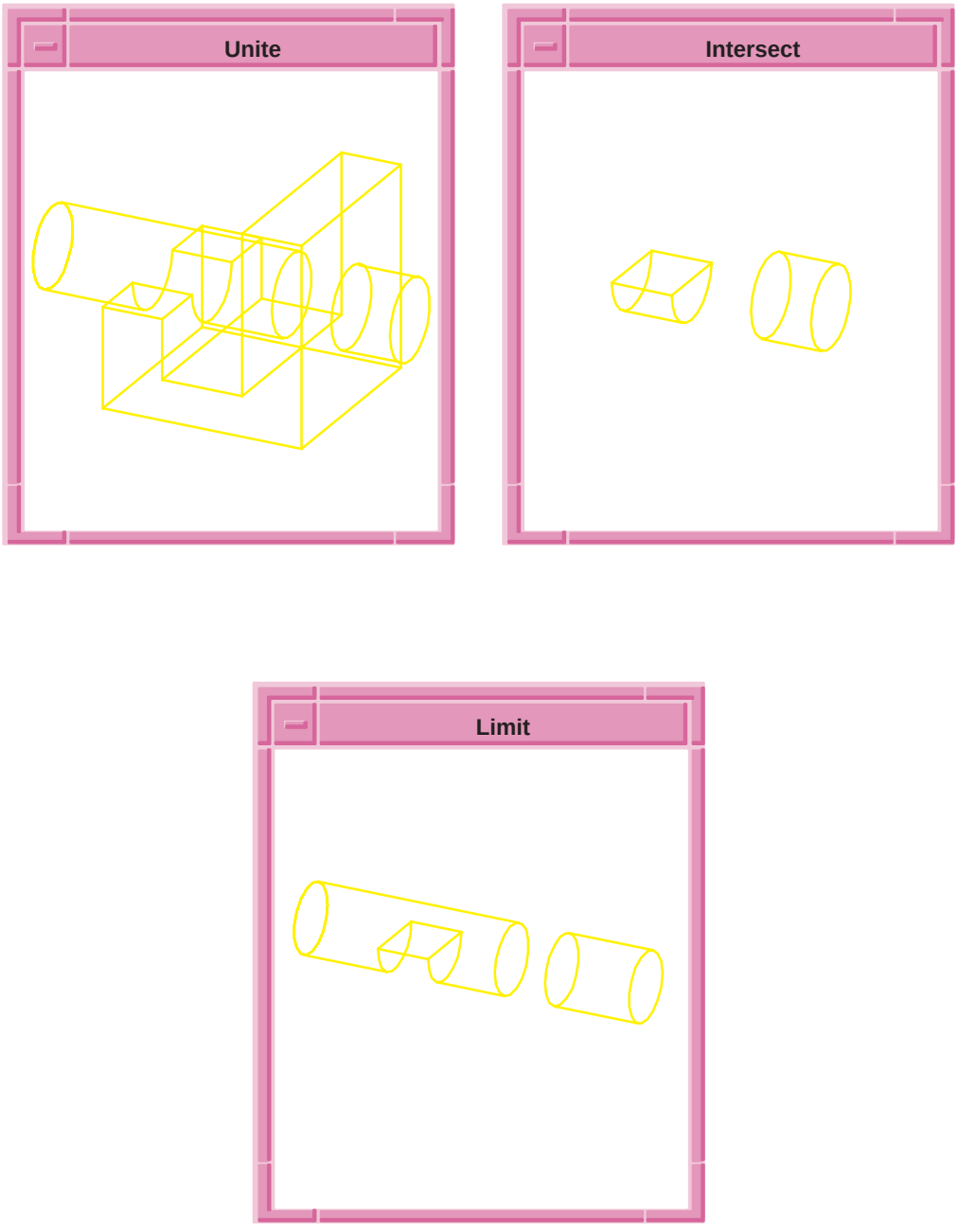


Figure 1-13. Sweep Using keep law

Surface Clipping

Topic: Sweeping

The to_face option permits clipping of the swept body at a surface. When working in Scheme, the sweep:options extension expects a pointer to a FACE entity. When using the api_sweep_with_options function with the sweep_options class, a pointer to a surface instead of a pointer to a face entity is required in the sweep_options data structure. The default is NULL. The to_face option is mutually exclusive with sweep_to_body.

Sweeping With Draft

Topic: *Sweeping, Laws

A practical use of drafting is for molded items. As the profile is swept, the ending profile has been offset an equal distance, which facilitates removal of the item from a mold. Extreme draft angles or draft laws can result in errors from self intersecting bodies, incomplete lateral faces (likely at corners), and/or unsupported topologies. Figure 1-14 represents an example offset occurring during a draft.

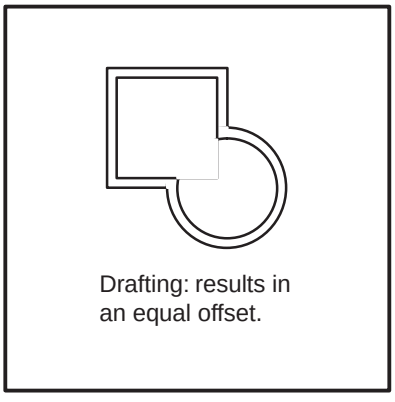


Figure 1-14. Drafting

Draft Sweeps with Angle Specifications

Topic: Sweeping

The option draft_angle is a real number that represents the angle with which the swept profile is to draft while sweeping. For closed profiles, a positive draft angle causes a draft *out*, while a negative draft angle causes a draft *in*. For example, if the profile is a circle in the *xy* plane, the path is a line along the *z*–axis, and the draft angle is 30 degrees, the result is an “upside–down” frustum. (A frustum is a cone with its tip cut off. It is “upside–down” in this case, because the draft angle *out* resulted in a top base larger than the bottom base.)

Negative draft angles can result in self–intersecting bodies, which are not permitted. However, the second graphic of Figure 1-15 illustrates a negative draft angle that does not result in a self–intersecting body. For open profiles, the positive draft angle is in the direction of the cross product of the tangent of the profile and the tangent to the path.

When a profile is provided, the start point of that profile is important. The sweeping operation moves the profile to this position and starts the sweep from there. In cases with draft angles or laws, the profile is placed at the location where the first degeneration (e.g., change of topology) occurs.

The following example illustrates how to use the Scheme extension sweep:law combined with sweep:options ("sweep_angle") to sweep a face with positive and nonself–intersecting negative draft angles.

Scheme Example

```
; draft_angle
(define path (edge:linear (position 0 0 0) (position 0 0 10)))
(define profile (edge:circular (position 0 0 0) 5))
(define opts (sweep:options "draft_angle" 30))
(define sweep (sweep:law profile path opts))
; redefine the options to create a negative draft angle.
; OUTPUT Positive Draft
(roll)
(define opts (sweep:options "draft_angle" -20))
(define sweep (sweep:law profile path opts))
; OUTPUT Negative Draft
```

Example 1-10. Sweeping with a Draft Angle

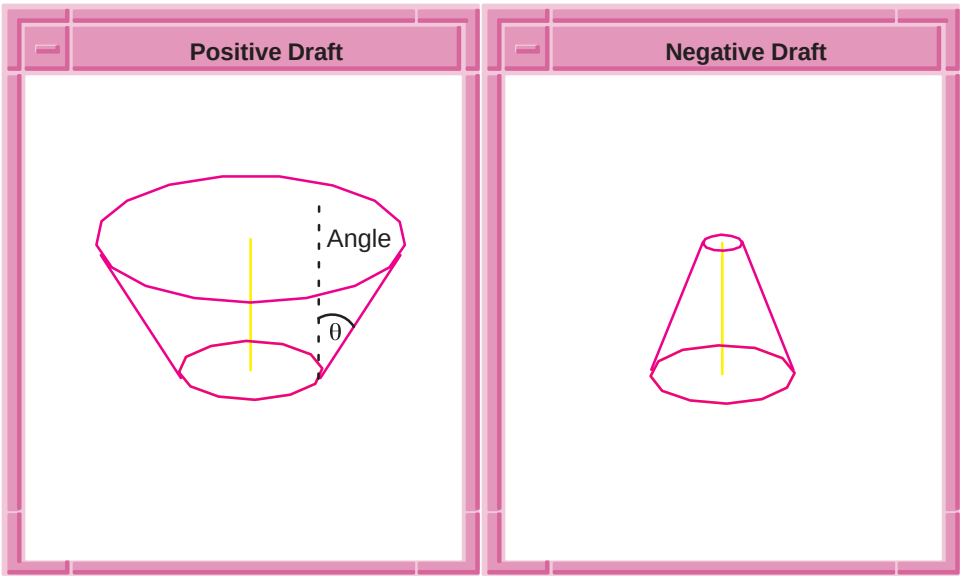


Figure 1-15. Sweeping with a Draft Angle

Scheme Example

```
; draft_angle
(define path (edge:linear (position 0 0 0) (position 0 0 20)))
(define profile (edge:linear (position 0 0 0) (position 0 30 0)))
(define opts (sweep:options ”draft_angle” 15))
(define sweep (sweep:law profile path opts))
(define color (entity:set–color sweep 3))
; OUTPUT Original
```

Example 1-11. Linear Sweep with a Positive Draft Angle

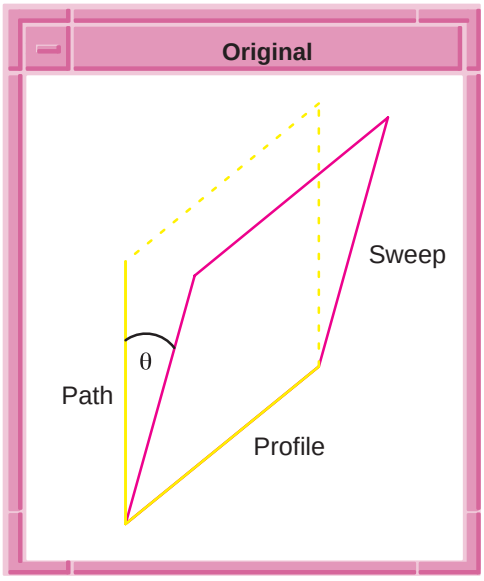


Figure 1-16. Linear Sweep with a Positive Draft Angle

The next example illustrates how to use the Scheme extension sweep:law combined with sweep:options (“sweep_angle”) to sweep a face with a positive draft angle.

Scheme Example

```
; sweep:law
; Sweep a positive draft angle.
; Create a solid block and get the faces.
(define block1 (solid:block (position –10 –10 –10)
    (position 30 30 30)))
;; block1
(define faces (entity:faces block1))
;; faces
; Select face to sweep.
(define face2 (car faces))
;; face2
; OUTPUT Original
```

```
; Create a new solid by sweeping the face along
; a gvector.
(define sweep2 (sweep:law face2 (gvector 0 0 10)
    (sweep:options ”draft_angle” 15)))
;; sweep2
; OUTPUT Result
```

Example 1-12. Sweeping a Face With Positive Draft Angle

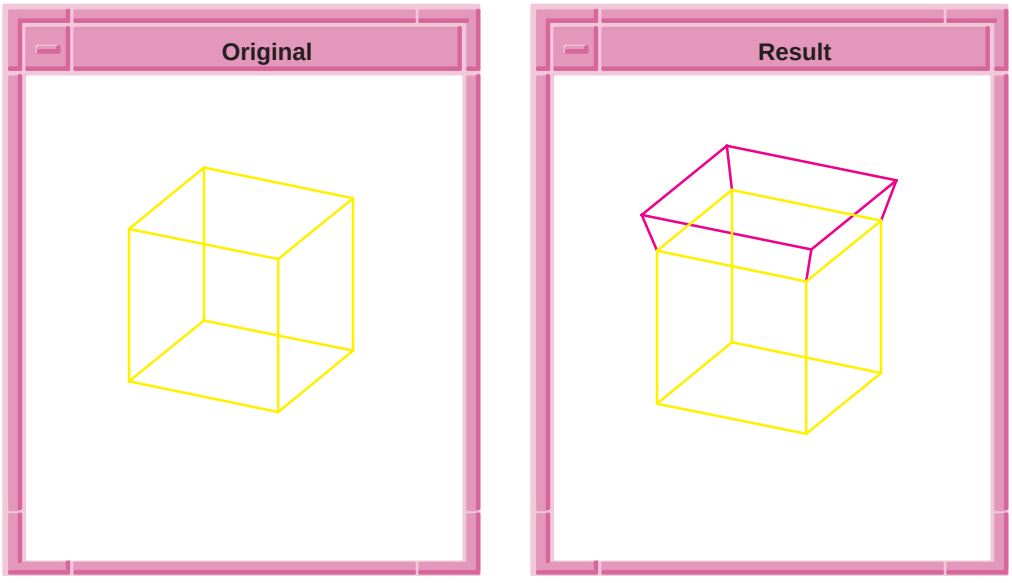


Figure 1-17. sweep:law – Sweeping a Face with Positive Draft Angle

End Offset Distance

Topic: Sweeping

The option end_draft_dist specifies an offset distance (a real number) for the swept surface in the plane where the sweep ends. It is similar to draft_angle, except that an offset distance is used instead of an angle. The larger the defined offset, the wider the draft distance. The default of 0 creates a swept surface parallel with the path. A negative offset creates a swept surface which is closer to the path at the end than at the beginning.

Scheme Example

```
; end_draft_dist
(define path (edge:linear (position 0 0 0) (position 0 0 10)))
(define profile (edge:circular (position 0 0 0) 5))
(roll:name-state "original")
(define opts (sweep:options "end_draft_dist" 5))
(define sweep (sweep:law profile path opts))
(entity:set-color sweep 3)
; OUTPUT Positive Draft

(roll "original")
(render:rebuild)
(define opts (sweep:options "end_draft_dist" -5))
(define sweep (sweep:law profile path opts))
(entity:set-color sweep 3)
(render:rebuild)
; OUTPUT Negative Draft
```

Example 1-13. Defining Offset Distance

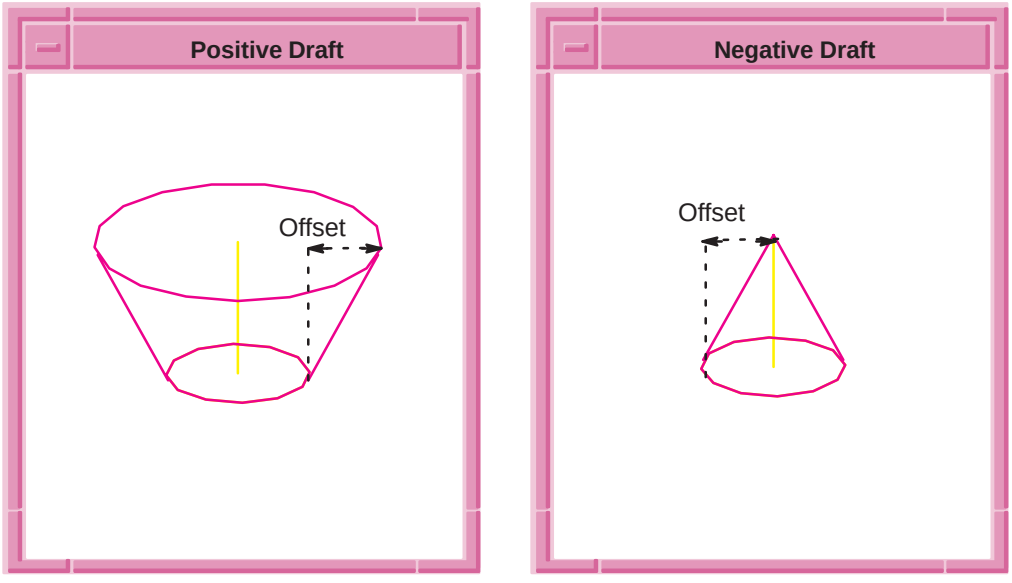


Figure 1-18. Sweeping with positive and negative Offset Distances

Draft Law

Topic: Sweeping, Laws

The option draft_law is followed by a law or a quoted law string composed of law functions. For example, if the profile is a circle in the *xy* plane, the path is a line along the *z*–axis, and the draft law is simply "sin(*x*)", the result is something resembling a vase. The value of the law function must be zero at the start. A further restriction is placed when the profile is not G1 continuous: the draft law always must be non–negative or non–positive. The default is NULL.

The following example, whose results are shown in Figure 1-19, sweeps a circular profile along a linear path. As it sweeps, it uses a draft law to create a vase–like appearance.

Scheme Example

```
; Sweeping a vase
(option:set "u_par" 10)
(option:set "v_par" 10)
(option:set "cone_par" #t)
(option:set "sil" #f)

(define edge1 (edge:linear (position 0 0 0) (position 0 0 25)))
(define path1 (wire-body edge1))
(define profile1 (edge:circular (position 0 0 0) 10))
(define vase1 (sweep:law profile1 path1
    (sweep:options "draft_law" "6*sin (x/4)")))
; OUTPUT Original
```

Example 1-14. Draft Law

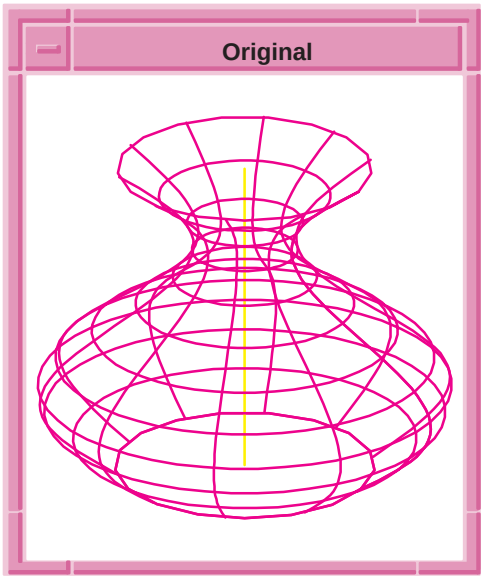


Figure 1-19. Sweeping a Vase Using draft_law

Scheme Example

```
; Sweep along a multi–edge path with a draft law of sin (x).
(define sweep_s1 (sweep:options ”draft_law” ”sin(x)”))
(define pedge1 (edge:linear (position 0 0 0)
    (position 40 0 0)))
(define pedge2 (edge:circular (position 40 20 0) 20 –90 90))
(define path1 (wire–body (list pedge1 pedge2)))
(define profile1 (edge:ellipse (position 0 0 0)
    (gvector 1 0 0) (gvector 0 5 0)))
; Use a draft law in order to get graphics faster.
(option:set ”silhouettes” #f)
(define sweep3 (sweep:law profile1 path1 sweep_s1))
;; sweep3
; OUTPUT Original
```

Example 1-15. Sweep Multi–edge Path

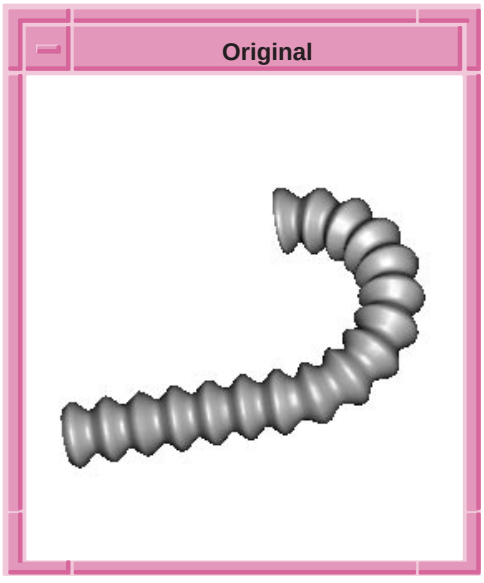


Figure 1-20. Sweep Multi–Edge Path

Draft Profile with Internal Loop Handling

Topic: Sweeping

The option draft_hole controls the offset behavior of internal loops (or holes) of a profile face when used to sweep with draft. In general, the direction of an internal loop's offset is opposite from the periphery (outer loop). Therefore, the draft direction of a hole is opposite from the periphery loop.

AGAINST_PERIPHERY . . . . . Holes are swept with the opposite draft direction as the periphery loop. This is the default setting of the draft_hole option.

ANGLE value . . . . . Holes are swept with the given draft angle. The draft angle can be different from the angle for the periphery loop.

NO_DRAFT . . . . . Holes are swept without draft.

WITH_PERIPHERY . . . . . Holes are swept with the same draft direction of the periphery loop.

The syntax in Scheme AIDE is similar to:

```
(sweep:options "draft_hole" "against_periphery"
    [... other options])
```

When using the associated Scheme code in the following example, the roll operations undo the previous sweep and permit a new sweep to be performed.

Scheme Example

```
; draft_hole
(define a_block (solid:block (position 0 0 –20)
    (position 10 10 –10)))
(define a_cyl (solid:cylinder (position 5 5 –10)
    (position 5 5 –20) 2))
(define hole (bool:subtract a_block a_cyl))
(define prof_body (face:unhook (list–ref (entity:faces hole) 1)))
(sheet:2d prof_body)
(define prof (car (entity:faces prof_body)))
(define delete (entity:delete hole))
(define path (gvector 0 0 20))
(define sw (sweep:law prof path
    (sweep:options ”draft_angle” –10)))
; OUTPUT Upper Left

(roll)
(define sw (sweep:law prof path (sweep:options ”draft_angle” 10
    ”draft_hole” ”with_periphery”)))
; OUTPUT Upper Right

(roll)
(define sw (sweep:law prof path (sweep:options ”draft_angle” –16
    ”draft_hole” ”no_draft”)))
; OUTPUT Center Left

(roll)
(define sw (sweep:law prof path (sweep:options ”draft_hole”
    ”by_angle” 30)))
; OUTPUT Center Right

(roll)
(define sw (sweep:law prof path (sweep:options ”draft_angle” 16
    ”draft_hole” ”by_angle” 30)))
; OUTPUT Lower Center
```

Example 1-16. Internal Loop Offset

Upper Left

(sweep:options ”draft_angle” –10)

Upper Right

(sweep:options ”draft_angle” 10 ”draft_hole” ”with_periphery”)

Center Left

(sweep:options ”draft_angle” –16 ”draft_hole” ”no_draft”)

Center Right

(sweep:options ”draft_hole” ”by_angle” 30)

Figure 1-21. Internal Loop Offsets

Draft Degeneracy Repair

Topic: Sweeping

In ACIS, surfaces created when sweeping with draft are defined by continuously offsetting the original profile along the path. If the offset amount is too large, an edge of the profile could be completely trimmed by adjacent edges. The draft_repair option handles this problem.

For the robustness and accuracy of sweeping with draft, the profile must be planar and the path must be tangent continuous between joints.

draft_repair controls the level of detection for degeneracy during sweeping with draft. The default setting is DEGENERACY. The levels of control include:

DEGENERACY . . . . . The sweep operation removes degenerate edges in the profile and creates a new profile to continue sweeping. For example, in Figure 1-22, the sweep operation creates a new profile made of edges e2, e3, and e4 and uses this profile after e1 degenerates. This setting checks only adjacent faces for profile changes.

FIRST_DEGENERACY . . . . . The sweep operation stops at the first degenerate location, as shown in Figure 1-22.

The syntax in Scheme is similar to:

```
(sweep:options ”draft_repair” ”first_degeneracy”
    [... other options])
```

Figure 1-22 illustrates a degenerate case where edge e1 is completely trimmed by e2 and e4. The sweep operation needs to detect the degenerate location and reset the profile to keep sweeping along the path.

Scheme Example

```
; Profile degeneracy
(define e1 (edge:linear (position 0 0 –10) (position 8 0 –10)))
(define e2 (edge:linear (position 8 0 –10) (position 10 6 –10)))
(define e3 (edge:linear (position 10 6 –10)
    (position 10 10 –10)))
(define e4 (edge:linear (position 10 10 –10) (position 0 0 –10)))
(define prof (wire–body (list e1 e2 e3 e4)))
(define path (wire–body (edge:linear (position 0 0 –10)
    (position 0 0 8))))
; OUTPUT Start

(define sw (sweep:law prof path (sweep:options ”draft_repair”
    ”first_degeneracy” ”draft_angle” –10 ”solid” #t)))
; OUTPUT Finish
```

Example 1-17. Using draft_repair

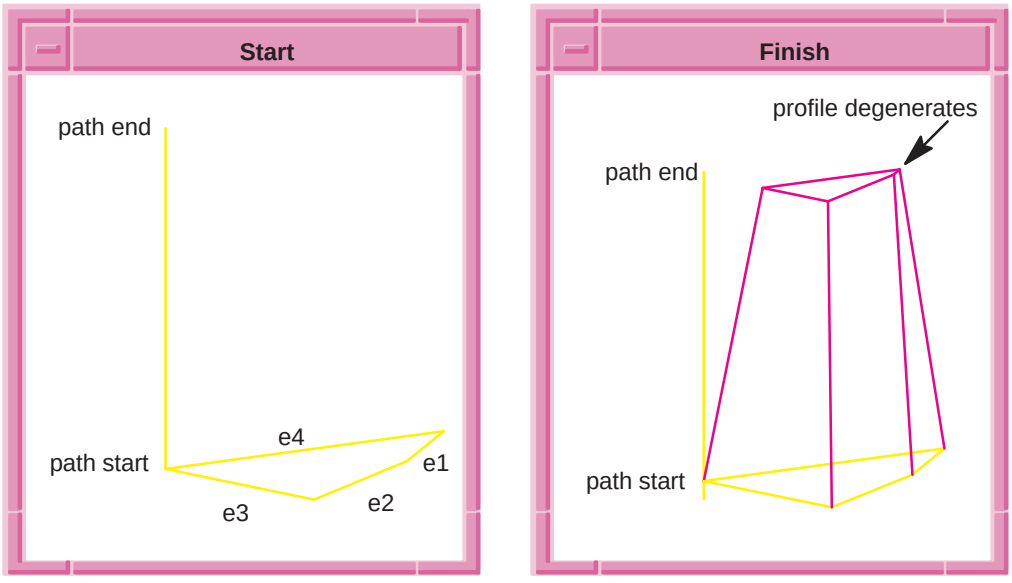


Figure 1-22. Sweeping to First Profile Degeneracy Location

Gap Repair

Topic: Sweeping

The gap_type option is an integer (enumeration) that specifies how to fill in gaps created by offsetting. The default is NATURAL. Consider the wire body made up of a square and a circle, as shown in Figure 1-23. When this wire body is swept with a draft angle (e.g., offset), a gap is created at certain junctures. Valid values are:

EXTENDED . . . . . 0: draws two straight tangent lines from the ends of each segment until they intersect.

ROUNDED . . . . . 1: creates a rounded corner between the two segments.

NATURAL . . . . . 2: extends the two shapes along their natural curves, e.g., along the circle and along the straight edge, until they intersect.

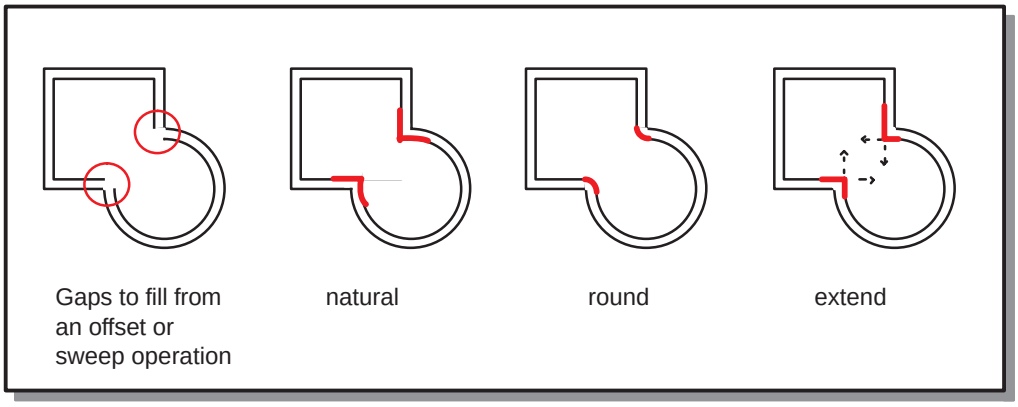


Figure 1-23. Gap Fill Types

Axis Sweeping

Topic: Sweeping

These options allow for the sweep to use a position and vector as the path. The angle of the revolution can be defined, and the geometry can be simplified (refer to Steps). A solid can be created from an open wire body.

Revolve on Sweep

Topic: Sweeping

The sweep_angle option specifies the angle to sweep around an axis, given in radians. The default is “2.0*M_PI”. Note that Scheme accepts this in degrees while C++ uses radians. Refer to Example 1-18 for a demonstration.

Scheme Example

```
; Sweep angle
; Create a profile
(define profile1 (edge:ellipse (position 0 0 0)
    (gvector 0 1 0) 5))
; Define an angle and axis to sweep about
(define sweep1 (sweep:law profile1 (position 15 0 0)
    (gvector 0 0 1) (sweep:options ”sweep_angle” 135)))
; OUTPUT Result
```

Example 1-18. Angled Sweep

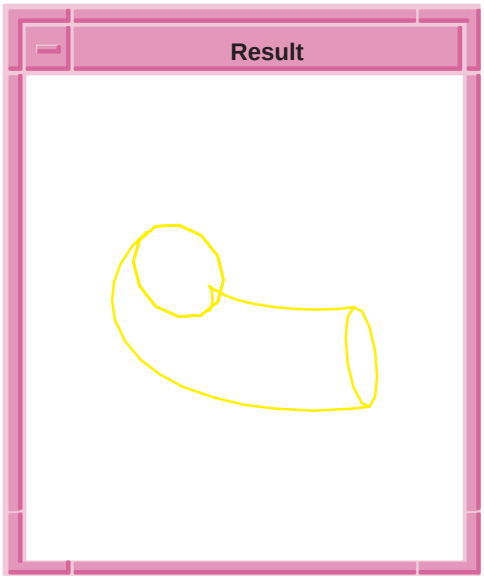


Figure 1-24. Angled Sweep

Linear Segmented Revolve

Topic: Sweeping

The steps option allows conversion of a circular sweep path into the specified number of linear segments. The results are polygons, and the intent is to create simpler geometry by keeping faces planar. The default is 0.

Because of the advantages of having the profile perpendicular to the path when an angle is less than 360 degrees is used, the beginning and ending segments of the path are half segments.

Revolved Solid Body from Open Wire Body

Topic: Sweeping

The close_to_axis option allows a solid body to be created from an open wire body profile. When TRUE, the ends are extended to the axis. Care should be taken so that all of the profile is on one side of the axis, to avoid a self–intersecting body. The default is FALSE.

Profile Orientation and Twisting

Topic: Sweeping

The rail_law and rigid options control the orientation of the profile during a sweep along a path. The twist_angle and twist_law options allows the profile to be rotated perpendicular to the path during the sweep.

Rail Laws

Topic: Sweeping, Laws

A rail law is simply a vector field along the sweep path. The field is made of unit vectors that are perpendicular to the path. An array of rail laws supplied to the sweep operation dictates how the profile is to be oriented on the path. The size of the rail law array is determined by the number of segments in the path. Each path segment has its own associated rail law. Because of this, the get function for this option requires a specific rail law to be defined. The number of rail laws can be found via get_rail_num().

The rail_laws option specifies the orientation of the profile as it is swept along the path. If not explicitly specified, the sweep functions use default rails. Care should be taken when using something other than the default rail laws. The resulting vector field must be unitized and perpendicular to the path at all times. More information on the default rail orientation and on deviating from the default is provided in this chapter and in the *Kernel Component Manual*, under the api_make_rails function.

Example 1-19 demonstrates sweeping a simple profile along a nonplanar path. In order to line the profile up with the sweep path, a rail law must be defined.

Scheme Example

```
; sweep:law
; To demonstrate the rail laws
; Create a sweep path from points.
(define plist1 (list (position 0 0 0)
    (position 10 0 0) (position 10 10 0)
    (position 10 10 10)))
(define start1 (gvector 1 0 0))
(define end1 (gvector 0 0 1))
(define path1 (edge:spline plist1
    start1 end1))
(define law1 (law "cur (edge1)" path1))
(define rail1 (law "minrot (law1,vec (0,-1,0))" law1))
(define edge1 (edge:linear (position 0 1 1)
    (position 0 1 -1)))
(define edge2 (edge:linear (position 0 1 -1)
    (position 0 -1 -1)))
(define edge3 (edge:linear (position 0 -1 -1)
    (position 0 -1 1)))
(define edge4 (edge:linear (position 0 -1 1)
    (position 0 1 1)))
(define profile1 (wire-body
    (list edge1 edge2 edge3 edge4)))
; OUTPUT Start
```

```
(define 1sweep (sweep:law profile1 path1
    (sweep:options ”rail_law” rail1)))
; OUTPUT Finish
```

Example 1-19. Using Rail Laws to Orient a Profile

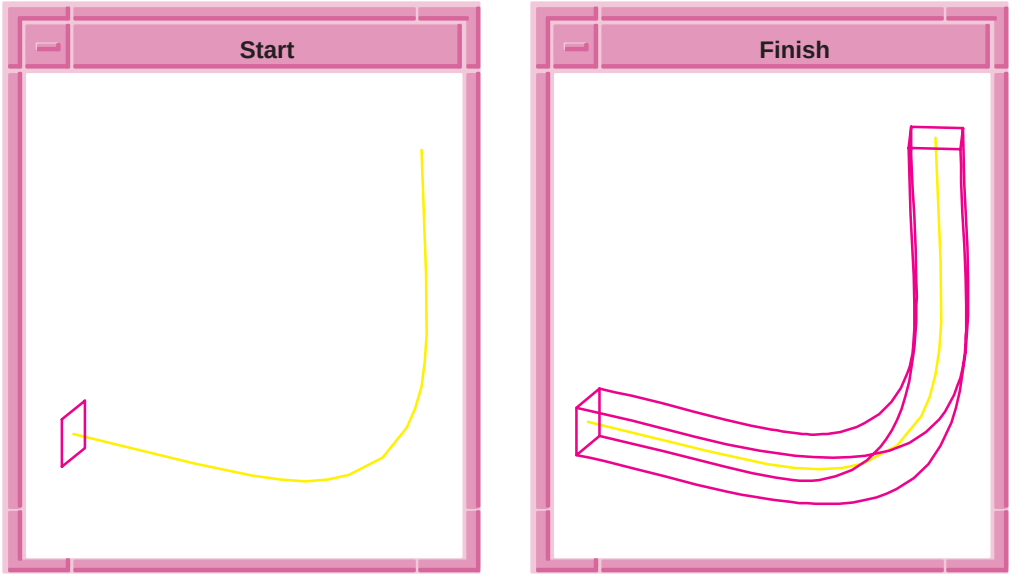


Figure 1-25. Using Rail Law to Orient a Profile

Rigid Sweep

Topic: *Sweeping, Laws

A *rigid sweep* is one in which the profile that is swept is translated—but not rotated—along the sweep path. It accomplishes a rigid extrusion of a profile along a path (Figure 1-26).

The rigid option specifies whether or not to make the cross–sections of a sweep parallel to one another. The default is FALSE, which means the cross–sections are perpendicular to the sweep path.

No checks are made when a rigid sweep is performed; consequently, the resulting surface is self–intersecting if the given path is closed, or if the direction of the path changes more than 180 degrees.

The following examples demonstrate how to use the Scheme extension sweep:options (“rigid”) to do a rigid sweep. In the first example, three linear edges are converted into a wire body. The rigid sweep is done on the wire body to create the solid.

Scheme Example

```
; Rigid sweep
; define a profile
(define e1 (edge:linear (position 0 0 0) (position 0 10 0)))
(define e2 (edge:linear (position 0 10 0) (position 10 10 0)))
(define e3 (edge:linear (position 10 10 0) (position 10 0 0)))
(define e4 (edge:linear (position 10 0 0) (position 0 0 0)))
(define profile1 (wire–body (list e1 e2 e3 e4)))
; define a path
(define e5 (edge:linear (position 5 5 0) (position 5 5 10)))
(define e6 (edge:linear (position 5 5 10) (position 35 5 30)))
(define e7 (edge:linear (position 35 5 30) (position 35 5 45)))
(define path1 (wire–body (list e5 e6 e7)))
; OUTPUT Start

; define a rigid sweep
(define sweep1 (sweep:law profile1 path1
    (sweep:options ”rigid” #t)))
; OUTPUT Finish
```

Example 1-20. Rigid Sweep

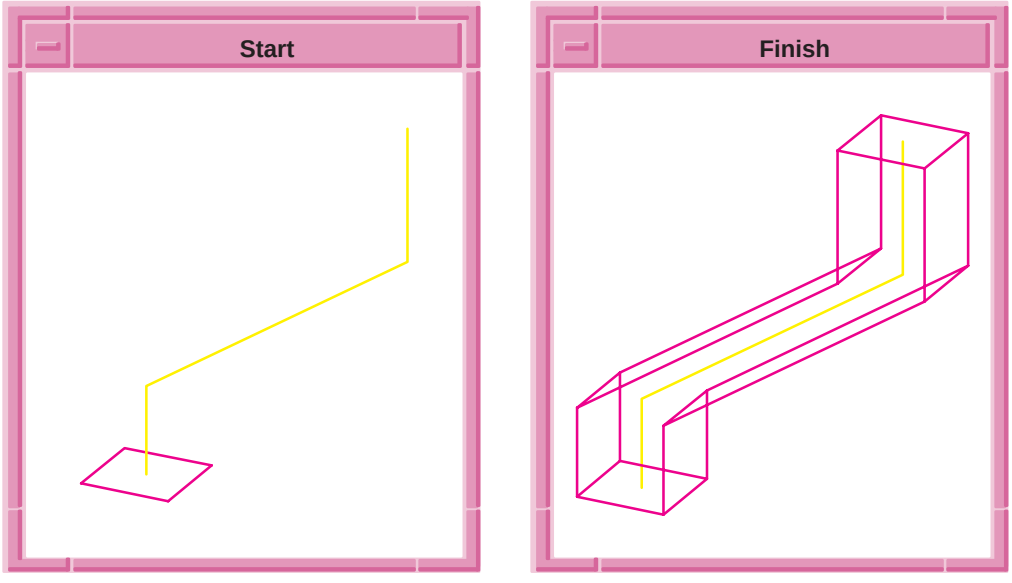


Figure 1-26. Rigid Sweep

Sweep With Twist

Topic: *Sweeping, Laws

Allows the profile to be rotated perpendicular to the path as it moves along the path.

The following examples demonstrate how to use sweep:options (“twist_angle”) to sweep with a twist. Notice the third and fourth figures where the twist has been changed to a negative value to cause the twist to turn in the opposite direction.

Scheme Example

```
; sweep:options
; Conducting a Sweep with Twist and negative Twist.
; Create a solid block.
(define block1 (solid:block (position –15 –15 –15)
    (position 15 15 15)))
; Get a list of the solid block’s faces and get
; one face.
(define faces (entity:faces block1))
(define entity1 (car (cdr (cdr (cdr (cdr
    (cdr faces)))))))
; Create a linear edge.
(define edge1 (edge:linear (position 15 0 0)
    (position 100 0 0)))
(roll:name–state ”original”)
; OUTPUT Original

; Sweep a face along the linear edge path.
(define sweep1 (sweep:law entity1 edge1
    (sweep:options ”twist_angle” 30)))
; OUTPUT Result

; Roll back one step in order to do a negative
; twist.
(roll “original”)
(define sweep1 (sweep:law entity1 edge1
    (sweep:options ”twist_angle” –85)))
; Negative Twist

(render)
; Rendered View – Negative Twist
```

Example 1-21. Sweep with Twist

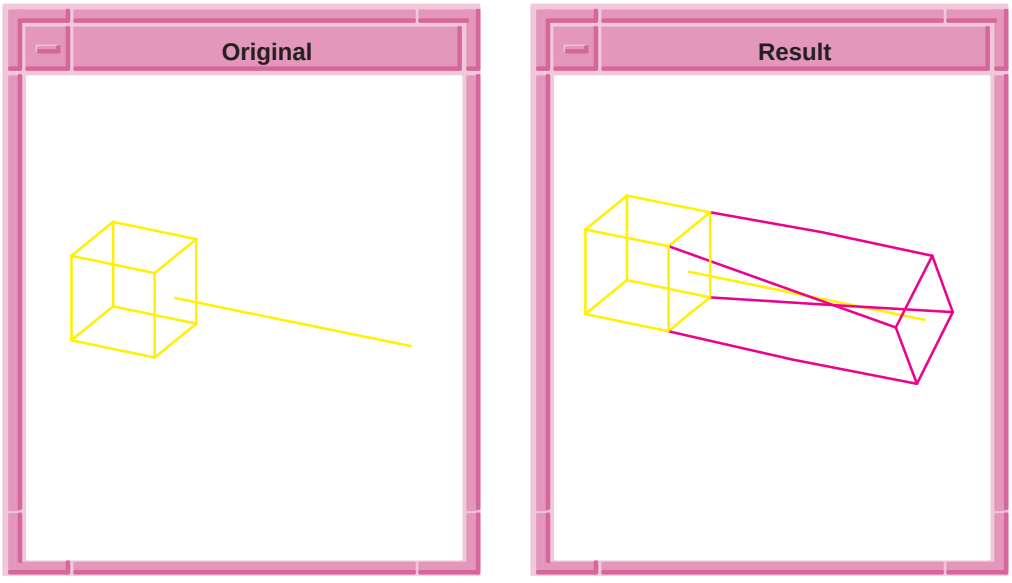


Figure 1-27. Sweep with Twist

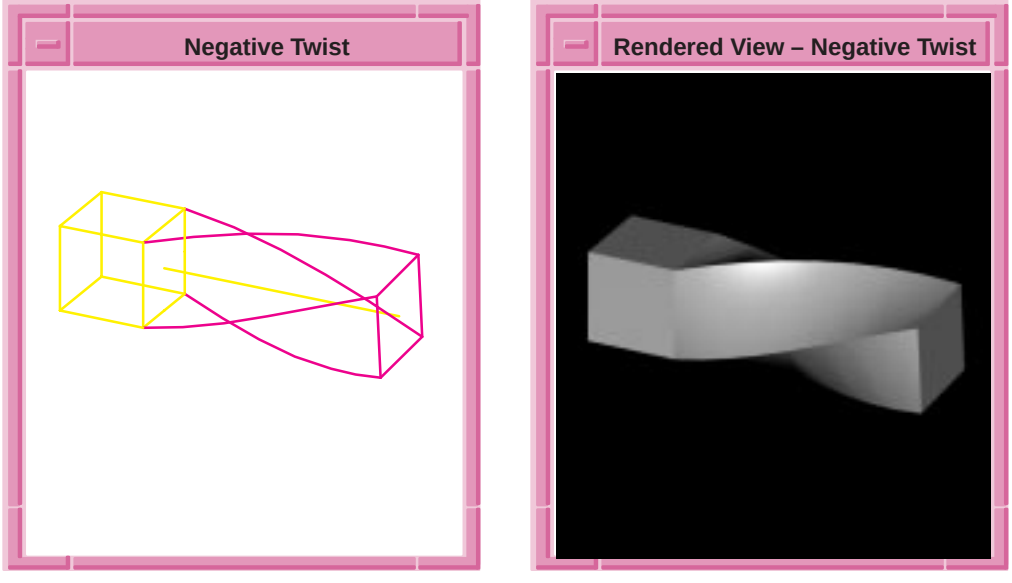


Figure 1-28. Sweep with Negative Twist

Angle Specified Twist

Topic: Sweeping

The twist_angle option is the default between twist_angle and twist_law. The option is followed by a real number for the angle. The twist angle represents how much the profile twists in total as the profile is swept along the path, regardless of the length of the path. The sweeping must be along a line segment and not a vector. Scheme accepts the angle in degrees while C++ expects the angle in radians. The default is 0.

Twisting has two mutually exclusive options: twist_angle and twist_law. Only one can be set by the sweep_options data structure. When one is set, the other is cleared out. The default is twist_angle set to 0. If the rail_law array is set up using api_make_rails and includes a twist law, then rail_law is mutually exclusive with the twist_law option. In other words, don't use a twist law more than once on a sweep.

Law Specified Twist

Topic: Sweeping, Laws

The twist_law option returns the angle of twist in radians as a function of the length of the wire, starting at wire length equal to zero for the beginning of the wire. The default is NULL.

Twisting has two mutually exclusive options: twist_angle and twist_law. Only one can be set by the sweep_options data structure. When one is set, the other is cleared out. The default is twist_angle set to 0. If the rail_law array was set up using api_make_rails and it included a twist law, then rail_law would be mutually exclusive with the twist_law option. In other words, don't use a twist law more than once.

Refer to Figure 1-29 for an illustration.

Scheme Example

```
; twist_law
(define profile (face:plane (position –5 –5 0) 10 10))
(define path (edge:linear (position 0 0 0) (position 0 0 20)))
(define opt (sweep:options "twist_law" "x^2*0.0125"))
(define sw (sweep:law profile path opt))
;; OUTPUT Result
```

Example 1-22. Sweep with twist_law

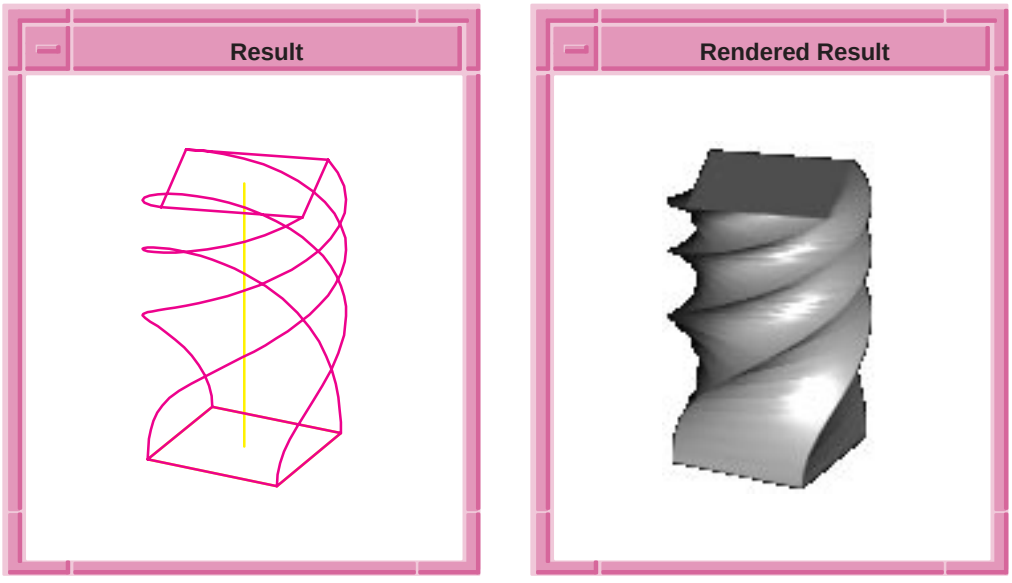


Figure 1-29. Sweep with Twist Law

Partial Path Sweep

Topic: *Sweeping, Laws

The portion option permits subsetting how much of the path is used for sweeping the profile. The default is ENTIRE_PATH. When positions are specified to localize the region for sweeping, the positions do not have to lie on the path. The closest point on the path from the specified position is used as the sweep limit.

BETWEEN_POINTS . . . . . sweeps the profile along the path between two specified locations. The locations do not have to be on the path. The closest positions on the path to the specified points are used as the region for sweeping.

ENTIRE_PATH . . . . . sweeps the profile along the entire path. The profile does not have to touch the path.

FROM_PROFILE . . . . . sweeps the profile starting from the profile and going to the end of the sweep path.

TO_PROFILE . . . . . sweeps the profile from the starting point of the path and going to the profile.

SWEEP_TO ..... is similar to BETWEEN_POINTS except that only one location is specified. The location does not have to be on the path. The closest point on the path to the specified location is used as the limit for sweeping. The second location is assumed to be the profile.

AS_IS ..... is not recommended for use, but is present for backward compatibility. It assumes the profile is at the starting point of the path and that the normal to the profile matches the parameterization at the beginning of the path. It sweeps the whole path. It does no checking to see where the profile is with respect to the path. Unexpected results are caused by discontinuities in the second derivative of the path generating discontinuities in the first derivatives of the swept surface.

Figure 1-30 shows a bracket containing a circular profile that is to be swept along a nonplanar path. Figure 1-30 shows three (of many) possibilities for sweeping using the portion sweep option. The example contains the associated Scheme code for these variations, as well as a few others that are not illustrated.

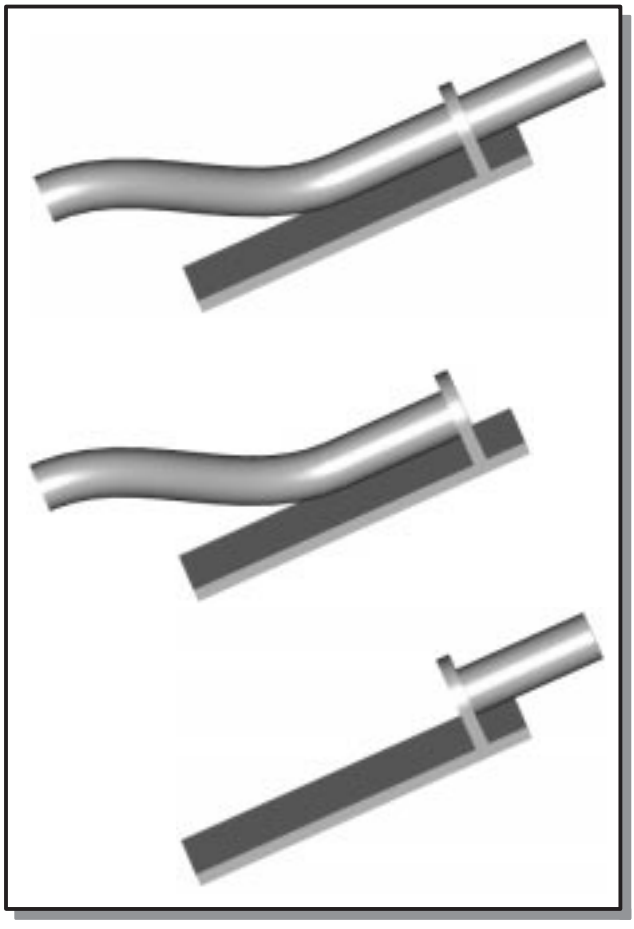


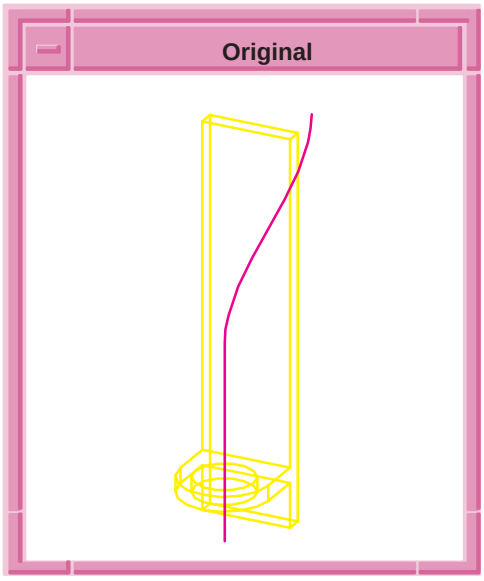
Figure 1-30. Variations of the Portion Sweep Option

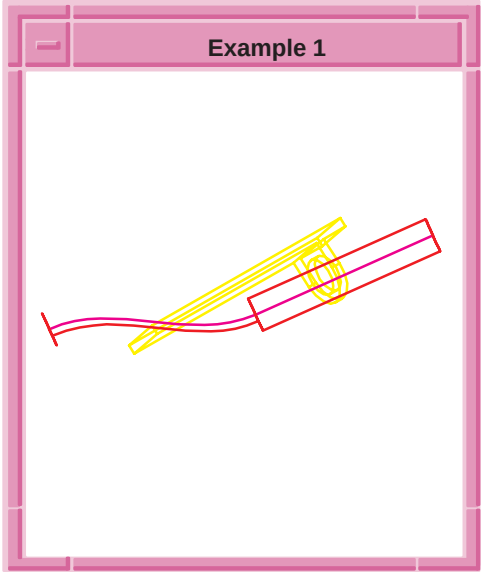
Scheme Example

```
; Portion sweep
; Define common elements
(journal:on "portion")
(define block (solid:block (position –1.5 –1.5 –6.5)
    (position 1.5 –2 6.5)))
(define arc (edge:ellipse (position 0 0 –5)
    (gvector 0 0 1) 1.5 1.0 0 180))
(define e1 (edge:linear (position 1.5 0 –5)
    (position 1.5 –2 –5)))
(define e2 (edge:linear (position 1.5 –2 –5)
    (position –1.5 –2 –5)))
(define e3 (edge:linear (position –1.5 –2 –5)
    (position –1.5 0 –5)))
(define prof (wire–body (list arc e1 e2 e3)))
(define through (sweep:along–vector prof
    (position 0 0 –5) (gvector 0 0 .5)))
(define base (solid:unite through block))
(define hole (solid:cone (position 0 0 –6)
    (position 0 0 –4) 1 1))
(define body (list prof through base hole))
(define rotate (entity:rotate body 0 0 1 180))
(define subtract (solid:subtract base hole))
(define zoom (begin (for–each zoom–all (env:views))))
(refresh–all)
; OUTPUT Subtract

(define e1 (edge:spline (list (position 0 0 0)
    (position 2 5 10)) (gvector 0 0 1) (gvector 0 0 1)))
(define e2 (edge:linear (position 0 0 –10) (position 0 0 0)))
(define path (wire–body (list e1 e2)))
(define reverse (wire:reverse path))
(define prof (wire–body (edge:ellipse (position 0 0 –4.5)
    (gvector 0 0 1) 0.9)))
; OUTPUT Common
; End of common elements
```

```
; Define the view. Uses a law trick which reads in a vector,
; manipulates it, and outputs a position.
(view:set–eye (law:eval–position (law ”100*law1” (view:right))))
(view:set–up (view:right))
(define rotate (entity:rotate subtract 0 1 1 45))
(render:rebuild)
(refresh–all)
(journal:off)
(journal:save ”portion”)
; OUTPUT Original

; Example 1
; default: move the profile to start and sweep entire path
(journal:load ”portion”)
(define sw (sweep:law prof path))
(entity:check sw)
; OUTPUT Example 1

; Example 2
(journal:load ”portion”)
(define opts (sweep:options ”portion” ”from_profile”))
(define sw (sweep:law prof path opts))
(entity:check sw)
; OUTPUT Example 2
```

Example 1-23. Sweep with portion option





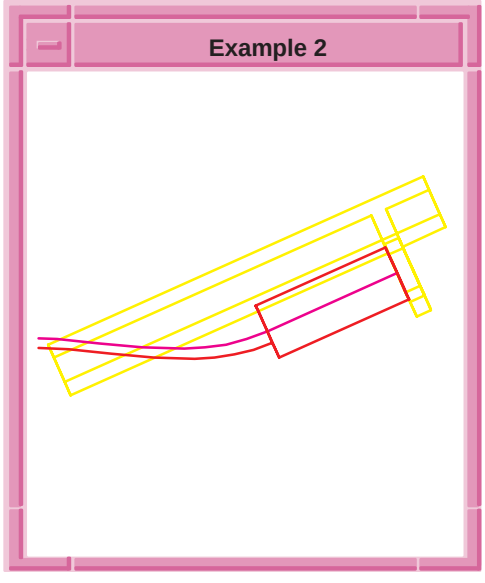


Figure 1-31. Sweep with Portion

```
; Example 3
(journal:load ”portion”)
(define opts (sweep:options ”portion” ”to_profile”))
(define sw (sweep:law prof path opts))
(zoom–all)
(entity:check sw)
; OUTPUT Example 3

; Example 4
(journal:load ”portion”)
(define start (position –1.5 –1.5 6.5))
(define start_mark (solid:sphere start 0.15))
(entity:set–color start_mark 1)
(define proj_start
    (position 0.59553578834011 1.4888394708503 3.7032127719460))
(define proj_mark (solid:sphere proj_start 0.1))
(entity:set–color proj_mark 1)
; OUTPUT Example 4a

(define limit (edge:linear start proj_start))
(define opts (sweep:options ”portion” ”sweep_to” start))
(define sw (sweep:law prof path opts))
(entity:check sw)
; OUTPUT Example 4b
```

Example 1-24. Sweep with Portion

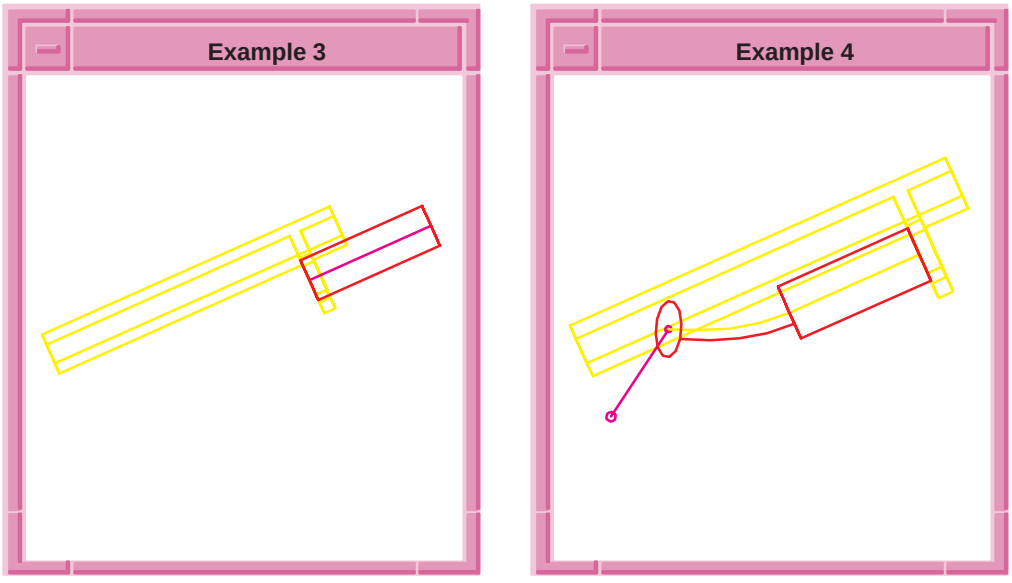


Figure 1-32. Sweep with Portions

```
; Example 5
(journal:load ”portion”)
(define end (position –1.5 –1.5 –6.5))
(define end_mark (solid:sphere end 0.15))
(entity:set–color end_mark 1)
(define proj_end (position 0 0 –6.5))
(define proj_mark (solid:sphere proj_end 0.1))
(entity:set–color proj_mark 1)
(define limit (edge:linear end proj_end))
(define opts (sweep:options ”portion” ”sweep_to” end))
(define sw (sweep:law prof path opts))
(entity:check sw)
; OUTPUT Example 5

; Example 6
(journal:load ”portion”)
(entity:set–color end_mark 1)
(define opts (sweep:options ”portion” ”between_points”
    start end))
(define sw (sweep:law prof path opts))
(entity:check sw)
; OUTPUT Example 6
```

Example 1-25. Sweep with Portion

Sweep Results

Topic: Sweeping

The options in this category allow for additional specification of the resulting body from the sweep operation. solid determines whether a wire body sweep results in a solid or sheet body. two_sided defines a resulting body as one or two–sided. cut_end_off establishes the end type. keep_start_face retains the original face profile. miter dictates the handling of corners.

Defining a 2D or 3D Body

Topic: Sweeping

The solid option is a logical flag which specifies whether the result of sweeping a wire body profile is intended to be a solid or a sheet body. The default is TRUE, meaning the result should be a solid, unless the profile is a non–closed wire or edge. If the intended result is a one–sided sheet body, the solid and two_sided options should be set to FALSE.

Scheme Example

```
; sweep with solid = true
; Define a solid cylinder
(define circ1 (edge:circular (position –20 0 0) 7.5))
(define path1 (edge:linear (position –20 0 0)
    (position –20 0 20)))
(define cyl (sweep:law circ1 path1 (sweep:options ”solid” #t)))
```

Example 1-26. Defining Resulting Body

One– or Two–Sided Bodies

Topic: Sweeping

The two_sided option is a convenience flag. If solid is set to FALSE, two_sided is set to TRUE, and the sweep operation creates a two–sided sheet body. If the intended result is a one–sided sheet body, then the solid and two_sided options should be set to FALSE.

The return is –1 if the option has not yet been set, 0 if the option has been set to FALSE, and 1 if the option has been set to TRUE.

Scheme Example

```
; Define a two sided straw
(define circ2 (edge:circular (position –10 20 0) 7.5))
(define path2 (edge:linear (position –10 20 0)
    (position –10 20 20)))
(define straw (sweep:law circ2 path2
    (sweep:options ”solid” #f ”two_sided” #t)))
```

Example 1-27. One– or Two–Sided Bodies

Defining an End Type

Topic: Sweeping

The cut_end_off option determines whether or not the end of the swept surface should be cut off squarely. When TRUE, the end of sweeping is planar and perpendicular to the path. When FALSE, the sweeping operation is faster, and a nonplanar profile is not cut off at the end of the sweep path. The default is FALSE.

Scheme Example

```
; Sweeping with cut_end_off = false
(define path1 (edge:linear (position 15 0 0)
    (position 15 20 20)))
(define profile1 (edge:circular (position 15 0 0) 5))
(define opts1 (sweep:options "cut_end_off" #f))
(define sweep1 (sweep:law profile1 path1 opts1))
(entity:set-color sweep1 3)
; OUTPUT Angled End Cut

; Sweeping with cut_end_off = true
(define path2 (edge:linear (position 0 0 0)
    (position 0 20 20)))
(define profile2 (edge:circular (position 0 0 0) 5))
(define opts2 (sweep:options "cut_end_off" #t))
(define sweep2 (sweep:law profile2 path2 opts2))
(entity:set-color sweep2 1)
; OUTPUT Flat End Cut
```

Example 1-28. Defining End Type

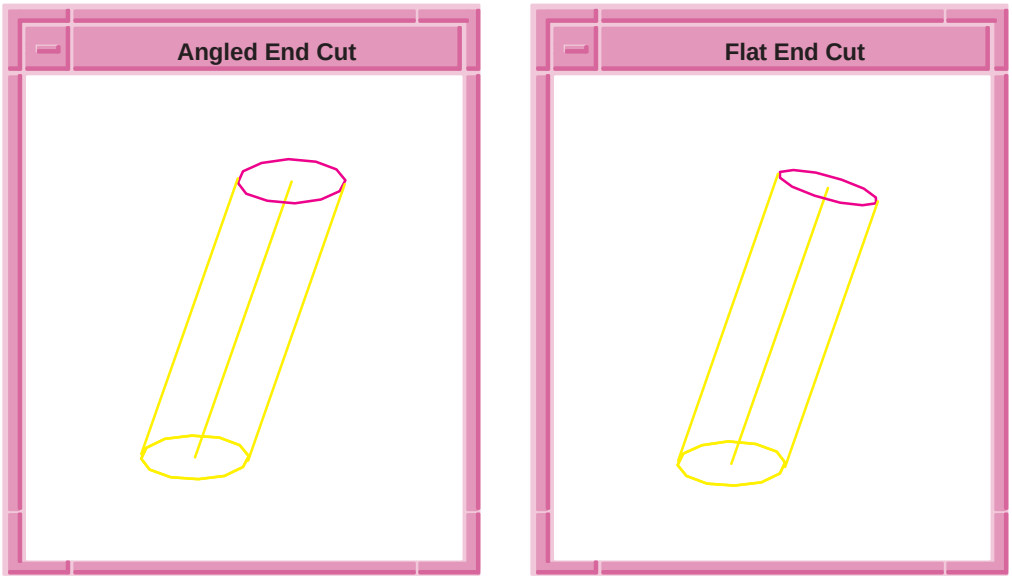


Figure 1-33. Defining End Type

Keep Profile Face on Closed Sweep

Topic: Sweeping

The keep_start_face option specifies the profile face used to sweep remains available after the sweep operation. This is useful when the sweep path is a closed loop, or when selective Booleans will be used after the sweep.

For example, a circular profile can be swept around a circle, resulting in a torus. By default, once the torus is created, the start face is removed, along with information about where the torus surface starts and ends. When keep_start_face is turned on, the start face is left as part of the resulting model, as demonstrated in Example 1-29. This acts as a “handle” for the start and end location. If the swept torus were to be used in a selective boolean process, the handle would be necessary. For example, the API function api_boolean_tube_body makes use of the start face.

The sweep_to_body option should not be used with keep_start_face.

Scheme Example

```
; Keep start face
(option:set ”sil” #t)
(define profile (edge:ellipse (position 10 0 0)
    (gvector 0 0 1) 2.5))
(entity:set–color profile 2)
(define path (edge:ellipse (position 0 0 0) (gvector 0 1 0) 10))
(define torus1 (sweep:law profile path))
; OUTPUT Sweep to Body

(entity:check torus1)
;; checked:
    1 lumps
    1 shells
    0 wires
    1 faces
    0 loops
    0 coedges
    0 edges
    0 vertices

(part:clear)
(option:set ”sil” #t)
(define profile2 (edge:ellipse (position 10 0 0)
    (gvector 0 0 1) 2.5))
(entity:set–color profile2 2)
(define path2 (edge:ellipse (position 0 0 0) (gvector 0 1 0) 10))
(define torus2 (sweep:law profile2 path2 (sweep:options
    ”keep_start_face” #t)))
; OUTPUT Keep Start Face

(entity:check torus2)
;; checked:
    1 lumps
    1 shells
    0 wires
    2 faces
    3 loops
    3 coedges
    1 edges
    1 vertices
```

Example 1-29. Sweep with keep_start_face option

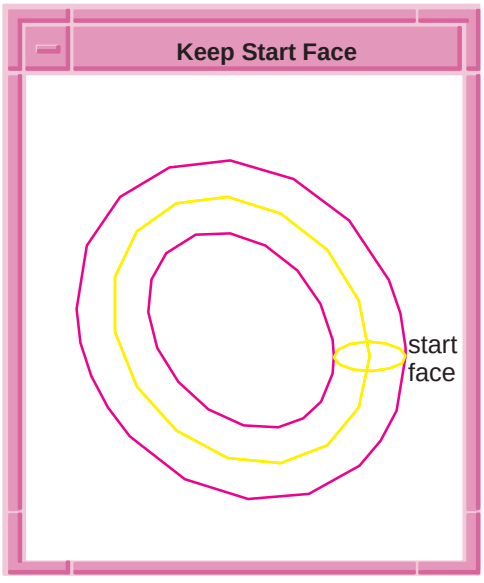


Figure 1-34. Keep Start Face

Mitering

Topic: Sweeping

The miter option specifies how mitering is performed. The default is the enumeration value default_miter, which takes the system default. Other options include new_miter, old_miter, crimp_miter and bend_miter. Refer to Example 1-30 for some demonstrations of this option.

new_miter . . . . . reflects the profile to the other side of the corner up to the discontinuous point. The two sides are then extended, intersected, and new edges formed as necessary. The "ending" profile is the same as the starting profile.

old_miter . . . . . intersects the plane which is perpendicular to the path at the half angle of the corner. The resulting profile on the plane is then swept continuing along the path.

crimp_miter . . . . . reflects the profile to the other side of the corner up to the discontinuous point. The portions of the two sides which are not intersecting are connected using a smooth rotation about the discontinuous point.

bend_miter . . . . . requires specification of a minimum radius to fillet the path. The result is a smooth curved junction. The minimum radius must be positive. A similar result could be obtained by using api_fillet_vertex to fillet the path before sweeping.

Scheme Example

```
; Sweep miter options
; Define a profile and path
(define profile (edge:ellipse (position 0 0 0)
    (gvector 0 1 0) 5))
(define edge1 (edge:linear (position 0 0 0) (position 0 10 0)))
(define edge2 (edge:linear (position 0 10 0) (position 10 10 0)))
(define path (wire–body (list edge1 edge2)))

; Example 1
(define sweep (sweep:law profile path
    (sweep:options ”miter_type” ”new_miter”)))
(define sweep (sweep:law profile path
    (sweep:options ”miter_type” ”new_miter”)))
;; OUTPUT New Miter

; Example 2
(roll –2)
(define sweep (sweep:law profile path
    (sweep:options ”miter_type” ”old_miter”)))
;; OUTPUT Old Miter

; Example 3
(roll)
(define sweep (sweep:law profile path
    (sweep:options ”miter_type” ”crimp_miter”)))
;; OUTPUT Crimp Miter

; Example 4
(roll)
(define sweep (sweep:law profile path
    (sweep:options ”miter_type” ”bend_miter” 1 )))
;; OUTPUT Bend Miter
```

Example 1-30. Sweep with miter option

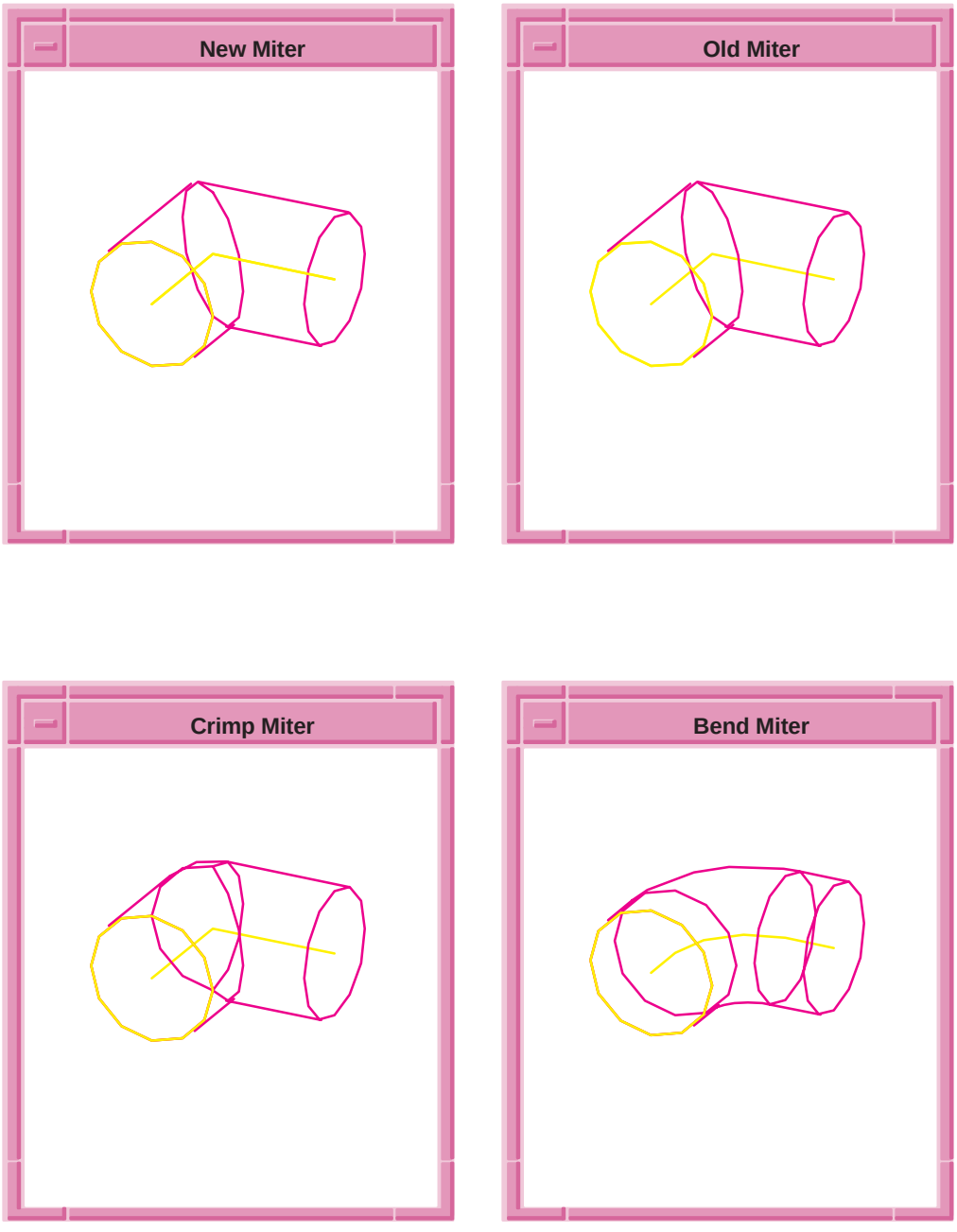


Figure 1-35. Sweep Miter Options

Self–intersecting checks in sweeping

Topic: *Sweeping

Potential self–intersecting sweeps can be caused by paths that bring the profile cross–sections to intersecting positions. Another method of creating a potential self–intersecting sweep is using a draft that will cause faces from different edges of the profile to intersect each other. Because self–intersecting bodies are not valid in ACIS, sweeping provides some methods to reduce the chances of creating a self–intersecting sweep.

When the path specified is a helix, sweeping automatically checks to see if the profile will intersect itself at successive turns of the helix. If the height of the profile relative to the axis direction is equal to or greater than the pitch of the helix, sweeping prevents the creation of a self–intersecting body. It is recommended that at least SPAresfit clearance between successive turns of the helix be allowed. This will improve the robustness of the resulting body.

There is an option, ”sweep_selfint” that will perform some additional checks on surfaces created in sweeping. These checks will find if a single surface created in sweeping intersects itself. Because this check decreases performance, it is off by default.

Another option, ”careful” performs even more stringent checks for self–intersecting sweeps. These checks also look for combinations of faces created during sweeping that are intersecting. This check is another decrease in performance and is off by default.

Rails, Vector Fields, and Hedgehogs

Topic: *Laws, *Sweeping

Rails are unit length vector fields which are perpendicular to the path. They serve to align the profile correctly to the path as it is swept. The sweep_options data structure accepts an array of rails in its set_rail_laws number function.. The dimension of the array is the number of segments in the sweep path. For each segment that is not defined by the user, default rail definitions are used.

The array of rails permits piecewise planar sweeping to be handled in an expected, predictable manner. The function api_make_rails and the associated Scheme extension law:make–rails may be used to define new rails, and perform automatic creation of the rail law array. If the default operation is acceptable, api_make_rails doesn’t have to be called explicitly. The api_make_rails function is a quick way of overriding one or all of the default rail definitions in the array.

- If the path is planar, the rail law is the planar normal.
- If the path is piecewise planar, the rail law is an array of planar normals.
- If the path is a helix, the rail law points towards the axis.

- If the path is an edge or a wire body, the rail law will have perpendicular segments for each section. The direction will be ambiguous; the developer can only be assured that it will be perpendicular.
- If the path is a curve of some other form, the minimum rotation rail law will be used. This returns a vector field with the minimum amount of twist along the path.

Refer to Figure 1-36 for illustrations of using default rail laws. The figures in this chapter make use of the law:hedgehog command which allows for the viewing of the created rails.

Scheme Example

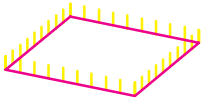
```
; Planar default – plane normal
; Define a plane
(define edge1 (edge:linear (position 10 10 0)
    (position 20 10 0)))
(define edge2 (edge:linear (position 20 10 0)
    (position 20 20 0)))
(define edge3 (edge:linear (position 20 20 0)
    (position 10 20 0)))
(define edge4 (edge:linear (position 10 20 0)
    (position 10 10 0)))
(define planar1 (wire–body (list edge1 edge2 edge3 edge4)))
(view:compute–extrema)
(entity:scale planar1 .9)
(render:rebuild)
(refresh–all)
(entity:set–color planar1 2)
; Define the rail laws
(define law1 (law:make–rails planar1))
(define default1 (law:hedgehog planar1 law1))
(define color (entity:set–color default1 3))
; OUTPUT Planar
```

```
; Piecewise planar default – planar normals
; Define a plane
(part:clear)
(define edge5 (edge:linear (position 11 11 –10)
    (position 20 11 –10)))
(define edge6 (edge:linear (position 20 11 –10)
    (position 20 20 –10)))
(define edge7 (edge:linear (position 20 20 –10)
    (position 10 20 –10)))
(define edge8 (edge:linear (position 10 20 –10)
    (position 10 10 –10)))
; Define a connection
(define connect (edge:linear (position 10 10 –10)
    (position 10 10 –20)))
; Define a plane
(define edge9 (edge:linear (position 10 10 –20)
    (position 20 10 –20)))
(define edge10 (edge:linear (position 20 10 –20)
    (position 20 20 –20)))
(define edge11 (edge:linear (position 20 20 –20)
    (position 11 20 –20)))
(define edge12 (edge:linear (position 11 20 –20)
    (position 11 11 –20)))
; Define the piecewise
(define piece_planar1 (wire–body (list edge5 edge6 edge7 edge8
     connect edge9 edge10 edge11 edge12)))
(entity:set–color piece_planar1 2)
(define law2 (law:make–rails piece_planar1))
(define default2 (law:hedgehog piece_planar1 law2))
(entity:set–color default2 3)
(view:compute–extrema)
(render:rebuild)
(refresh–all)
; OUTPUT Piecewise Planar

; Helix default – axis centric
(part:clear)
(define path1 (edge:helix (position –10 5 0)
    (position –10 0 0) (gvector 0.1 1 0)5 2 #t))
(entity:set–color path1 3)
(define law3 (law:make–rails path1))
(define default3 (law:hedgehog path1 law3 20))
(zoom–all)
; OUTPUT Helix
```

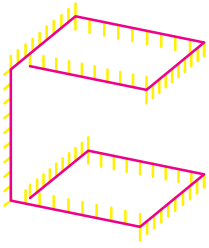
```
; Straight edge default – ambiguous perpendicular
(part:clear)
(define straight1 (edge:linear (position –10 20 0)
    (position –10 30 0)))
(entity:set–color straight1 2)
(define law4 (law:make–rails straight1))
(define default4 (law:hedgehog straight1 law4))
(entity:set–color default4 3)
(define straight2 (edge:linear (position –10 30 5)
    (position –10 20 3)))
(entity:set–color straight2 2)
(define law5 (law:make–rails straight2))
(define default5 (law:hedgehog straight2 law5))
(entity:set–color default5 3)
(zoom–all)
; OUTPUT Straight Edge

; Curve default – minimum rotation
(part:clear)
(define plist1 (list (position –10 0 –20)
    (position –20 0 –20) (position –20 10 –20)
    (position –10 10 –10)))
(define start1 (gvector 1 0 0))
(define end1 (gvector 0 0 1))
(define curve1 (edge:spline plist1 start1 end1))
(entity:set–color curve1 2)
(define law6 (law:make–rails curve1))
(define default6 (law:hedgehog curve1 law6 30))
(entity:set–color default6 3)
(zoom–all)
; OUTPUT Curve
```

Example 1-31. Using Rails, Vectors, and Hedgehogs

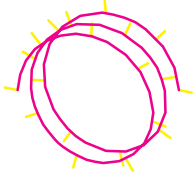
Planar



Piecewise Planar



Helix



Straight Edge

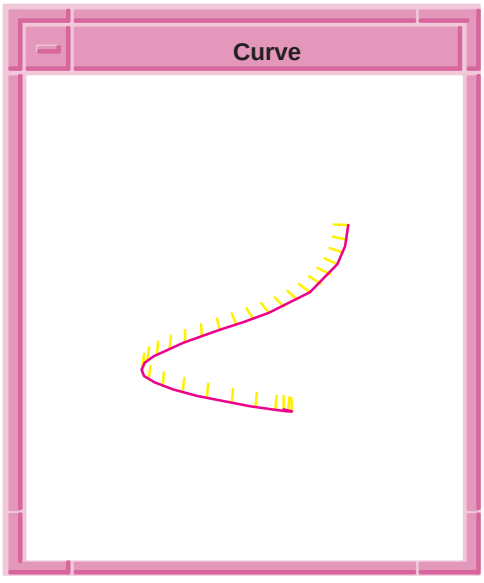


Figure 1-36. Using the Rail Laws

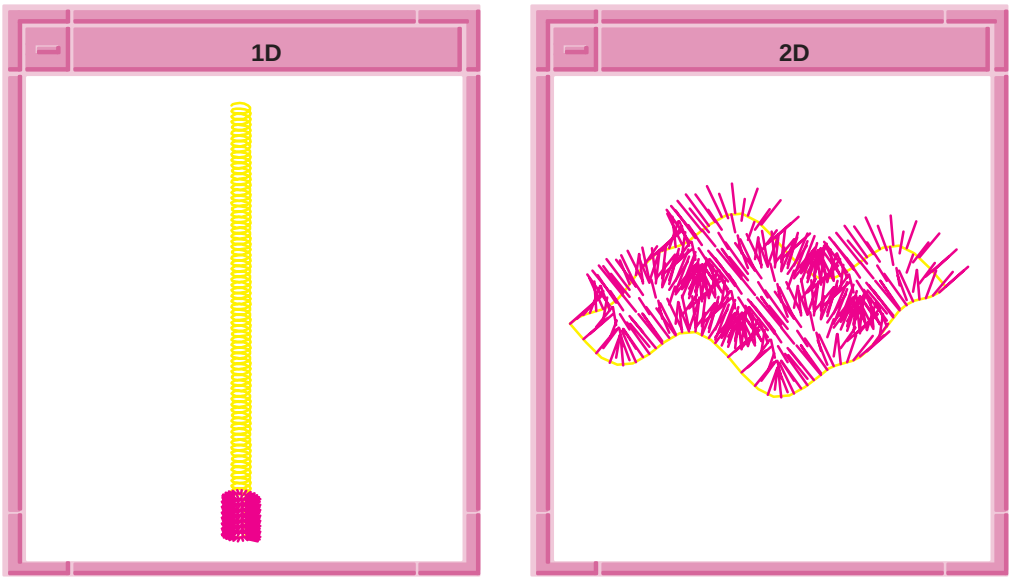
Law functions are translatable from one dimension to another. For example, a law used to make a surface may go from 2D to 3D. 1D–to–3D laws may be viewed as vector fields on a curve. 2D–to–3D laws may also be viewed as vector fields on a surface. 3D–to–3D laws may be viewed as a vector field in space. Figure 1-37 illustrates “hedgehogs” formed from the vector field representations.

Scheme Example

```
; Create a 1D Hedgehog
(define edge1 (edge:law ”vec (cos(x),sin (x),x/10)” 0 500))
(zoom–all)
(define base_law1 (law ”vec (cos (x),sin (x),x/10)”))
(define hair_law1 (law ”vec (cos (x),sin (x),0)”))
(define h1 (law:hedgehog hair_law1 base_law1 0 50 200))
; OUTPUT 1D
```

```
; Create a 2D Hedgehog
(part:clear)
(define face1 (face:law "vec (x,y,cos (x)*sin (y))" -5 5 -5 5))
(zoom-all)
(define base_law2 (law "vec (x,y,cos (x)*sin (y))"))
(define dx (law:derivative base_law2))
(define dy (law:derivative base_law2 "y"))
(define norm (law "cross (law1,law2)" dx dy))
(define h1 (law:hedgehog norm base_law2 -5 5 -5 5 20 ))
; OUTPUT 2D

; Create a 3D Hedgehog
(part:clear)
(define base_law3 (law "vec (x,y,z)"))
(define hair_law3 (law "norm (vec (x,y,z))"))
(define h1
    (law:hedgehog hair_law3 base_law3 -10 10 -10 10 -10 10))
; OUTPUT 3D
```

Example 1-32. Demonstrating Hedgehogs



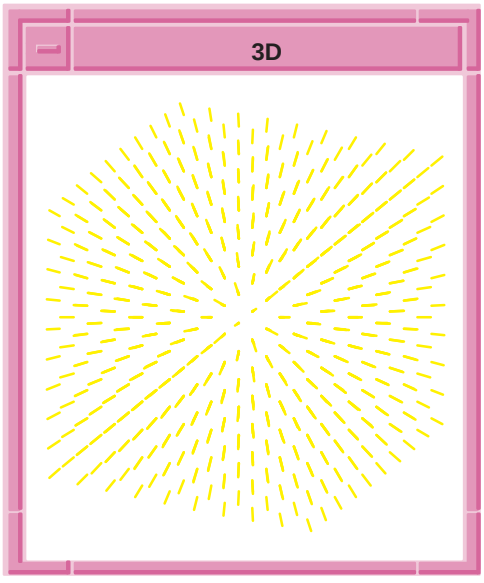


Figure 1-37. 1D, 2D, and 3D Hedgehogs

Feature Naming

Topic: Sweeping, *Feature Naming

Sweep annotations allow features, created during sweeping, to be named. With feature naming, the underlying geometry might be altered, but the feature can still be referenced through the name. To use the annotations, the annotation option must be set to TRUE. (For more information about feature naming, refer to the *Kernel Component Manual*.)

Sweep annotations contain information identifying the inputs and outputs of a sweep operation. The inputs to the sweep operation include the edges and vertices of the profile. The inputs also include the edges of the path, when a path is specified. If sweeping is called using a vector, distance, or axis in place of the path, no path input is present in the annotation. In some miter situations the path vertex may also be included in the inputs. Use the path edge entities instead of path vertices. Annotations on faces should be used before annotations on edges or vertices, since the face annotations are the least subject to algorithmic changes. The following section presents a series of sweep examples illustrating some of the various annotations scenarios.

The following classes are derived from the base class SWEEP_ANNOTATION:

SWEEP_ANNO_EDGE_LAT . . . . . Identifies the lateral (side) topology created from a profile edge during sweeping. If faces have been merged, there are two path input edges.

Inputs:

Profile – edge

Path – edge

Outputs:

Lateral_face – face

SWEEP_ANNO_EDGE_TOP . . . . . Defines the top (ending) topology created from a profile edge during sweeping.

Inputs:

Profile – edge

Path – edge

Outputs:

Top_edge – edge

SWEEP_ANNO_END_CAPS . . . . . Identifies the faces at the start and end of the sweep. Does not have the inputs and outputs in the same style as the other sweep annotations. An annotation of this kind can have a pointer to a start and an end, or just one or the other.

Inputs:

none

Outputs:

Start_face – face

End_face – face

SWEEP_ANNO_VERTEX_LAT . . . . . Identifies the lateral (side) topology created from a profile vertex during sweeping.

Inputs:

Profile – vertex

Path – edge

Outputs:

Right_lateral_face – face

Left_lateral_face – face

Right_lateral_edge – edge

Left_lateral_edge – edge

Mid_lateral_edge – edge

SWEEP_ANNO_VERTEX_TOP ..... Identifies the top (ending) topology created from a profile vertex during sweeping. The vertex is the input profile vertex and the output mid_top_vertex

Inputs:

Profile – vertex

Path – edge

Outputs:

Right_top_edge – edge

Left_top_edge – edge

Right_top_vertex – vertex

Left_top_vertex – vertex

Mid_top_vertex – vertex

Sweeping an Open Path Without Draft

Topic: *Sweeping

This example demonstrates the most common annotations scenario. Subsequent examples only demonstrate annotations particular to that scenario.

Scheme Example

```
(define profile (wire-body:points (list (position 0 0 0)
    (position 3 0 0) (position 3 3 0) (position 0 3 0)
    (position 0 0 0))))
(define path (wire-body:points (list
    (position 1.5 1.5 0) (position 1.5 1.5 8.5)
    (position 10 1.5 8.5))))
(zoom-all)
(define swp (sweep:law profile path))
```

Example 1-33. Sweep Open Path Without Draft

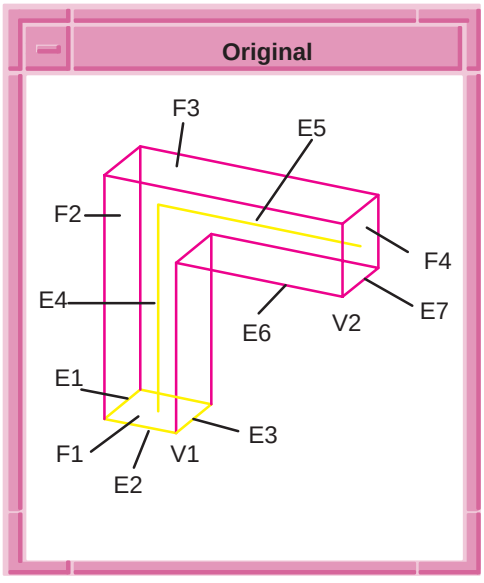


Figure 1-38. Sweep Open Path Without Draft

A SWEEP_ANNO_EDGE_LAT annotation has an input profile edge (E1), an input path edge (E5), and an output lateral_face (F3). Another SWEEP_ANNO_EDGE_LAT annotation has an output lateral_face (F2) and an input profile edge (E2), but because faces have been merged there are two path input edges (E4 and E5).

A SWEEP_ANNO_EDGE_TOP annotation has an input profile edge (E3), an input path edge (E5), and an output top_edge (E7).

A SWEEP_ANNO_VERTEX_LAT annotation has an input profile vertex (V1), an input path edge (E5), and an output mid_lateral_edge (E6). E6 is labeled as a mid_lateral_edge because in later examples with draft other identifiers are used.

A SWEEP_ANNO_VERTEX_TOP annotation has an input profile vertex (V1), an input path edge (E5), and an output mid_top_vertex (V2).

SWEEP_ANNO_END_CAPS annotations do not have the inputs and outputs in the same style as the other sweep annotations. An annotation of this kind can have a pointer to a start_face (F1) and an end_face (F4), or just one or the other.

Sweeping a Closed Path Without Draft

Topic: *Sweeping

In this case the profile edges and vertices are equivalent to the top edges and vertices.

This sweep would not have a SWEEP_ANNO_END_CAPS annotation, unless the sweep options were set to keep the start face. If the option is set, face (F1) is the start_face and the end_face.

Scheme Example

```
(define profile (wire-body (list (edge:linear
    (position 0 0 0)(position 3 0 0))
    (edge:linear (position 3 0 0) (position 1.5 1.5 0))
    (edge:linear (position 1.5 1.5 0) (position 0 0 0)))))
(define path (wire-body (list (edge:linear
    (position 1.5 0.75 0) (position 1.5 0.75 5))
    (edge:ellipse (position 1.5 -2.5 5) (gvector 1 0 0)
    (gvector 0 3.25 0) 1 0 180)
    (edge:linear (position 1.5 -5.75 5) (position 1.5 -5.75 0))
    (edge:ellipse (position 1.5 -2.5 0) (gvector -1 0 0)
    (gvector 0 3.25 0) 1 0 180))))
(define swp (sweep:law profile path))
(define zoom (zoom-all))
(define torus (list swp path))
(define rotate (entity:rotate torus 0 1 0 105))
(define rotate (entity:rotate torus 1 0 0 70))
```

Example 1-34. Sweep Closed Path Without Draft

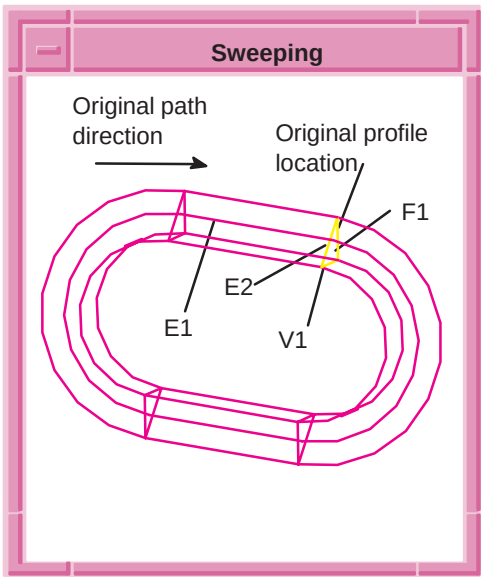


Figure 1-39. Sweep Closed Path Without Draft

A SWEEP_ANNO_EDGE_TOP annotation has an input path edge (E1). The edge (E2) is both an input profile edge and an output mid_top_edge.

A similar SWEEP_ANNO_VERTEX_TOP annotation has an input path edge (E1). The vertex (V1) is both the input profile vertex and the output mid_top_vertex.

This sweep would not have a SWEEP_ANNO_END_CAPS annotation, unless the sweep options were set to keep the start face. If the option is set, face (F1) is the start_face and the end_face.

Sweeping With Draft Resulting in No Gap Faces

Topic: *Sweeping

Here the annotations are the same as 'without draft'.

Scheme Example

```
(define profile (wire-body:points (list (position 0 0 0)
    (position 2 -2 0) (position 4 0 0) (position 0 4 0)
    (position -4 0 0) (position -2 -2 0) (position 0 0 0))))
(define path (edge:linear (position 6 0 0) (position 6 0 10)))
(define opts (sweep:options "draft_angle" 5))
(define swp (sweep:law profile path opts))
```

Example 1-35. Sweep with Draft for No Gap Face

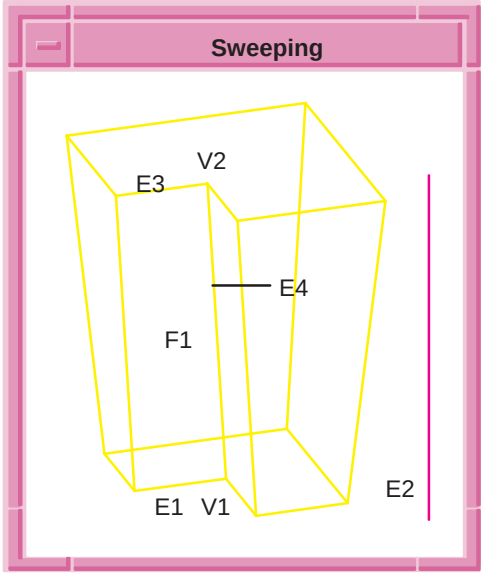


Figure 1-40. Sweep with Draft for No Gap Face

A SWEEP_ANNO_EDGE_LAT annotation has an input profile edge (E1), an input path edge (E2), and an output lateral_face (F1).

A SWEEP_ANNO_EDGE_TOP annotation has an input profile edge (E1), an input path edge (E2), and an output top_edge (E3).

A SWEEP_ANNO_VERTEX_LAT annotation has an input profile vertex (V1), an input path edge (E2), and an out put mid_lateral_edge (E4).

A SWEEP_ANNO_VERTEX_TOP annotation has an input profile vertex (V1), an input path edge (E2), and an output mid_top_vertex (V2).

Sweeping With Draft Resulting in One Gap Face

Topic: *Sweeping

With the introduction of gap faces, the right, left, and mid modifiers take on significance.

Scheme Example

```
(define profile (wire-body:points (list (position 0 0 0)
    (position 2 -2 0) (position 4 0 0) (position 0 4 0)
    (position -4 0 0) (position -2 -2 0) (position 0 0 0))))
(define path (edge:linear (position 6 0 0) (position 6 0 10)))
(define opts (sweep:options "draft_angle" -3 "gap_type"
     "rounded"))
(define swp (sweep:law profile path opts))
```

Example 1-36. Sweep With Draft – One Gap Face

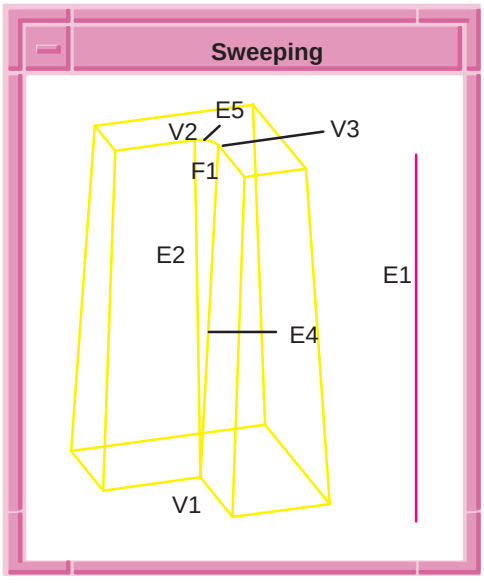


Figure 1-41. Sweep With Draft – One Gap Face

A SWEEP_ANNO_VERTEX_LAT annotation has an input profile vertex (V1), an input path edge (E1), and an output left_lateral_edge (E2).

A SWEEP_ANNO_VERTEX_LAT annotation has the input profile vertex (V1), the input path edge (E1), and an output right_lateral_edge (E4).

A SWEEP_ANNO_VERTEX_LAT annotation has the input profile vertex (V1), the input path edge (E1), and an output right_lateral_face (F1). When a single gap face results from the sweep, the output face (F1) is labeled as a right_lateral_face.

A SWEEP_ANNO_VERTEX_TOP annotation has an input profile vertex (V1), an input path edge (E2), and an output left_top_vertex (V2).

A SWEEP_ANNO_VERTEX_TOP annotation has the input profile vertex (V1), the input path edge (E1), and an output right_top_vertex (V3).

A SWEEP_ANNO_VERTEX_TOP annotation has the input profile vertex (V1), the input path edge (E1), and an output right_top_edge (E5). When a single gap face results from the sweep, the output edge (E5) is labeled as a right_top_edge.

Sweep With Draft Results – Two Gap Faces

Topic: *Sweeping

This case shows how sweep annotations identify two faces resulting from one profile vertex.

Scheme Example

```
(define edge1 (edge:circular–3pt (position 0 0 0)
    (position 1 –1.5 0) (position 2 –2 0)))
(define edge2 (edge:linear (position 2 –2 0) (position 4 0 0)))
(define edge3 (edge:circular–3pt (position 4 0 0)
    (position 0 4 0) (position –4 0 0)))
(define edge4 (edge:linear (position –4 0 0) (position –2 –2 0)))
(define edge5 (edge:circular–3pt (position –2 –2 0)
    (position –1 –1.5 0) (position 0 0 0)))
(define profile (wire–body (list edge1 edge2 edge3 edge4 edge5)))
(define path (edge:linear (position 6 0 0) (position 6 0 10)))
(define opts (sweep:options ”draft_angle” –3 ”gap_type”
     ”extended”))
(define swp (sweep:law profile path opts))
(zoom–all)
(entity:rotate swp 0 0 1 20)
```

Example 1-37. Sweep With Draft – Two–Gap Face

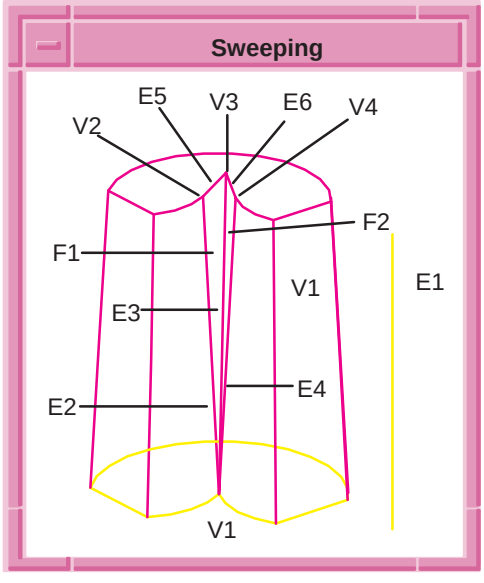


Figure 1-42. Sweep With Draft – Two–Gap Face

The annotations particular to this sweeping case all have an input profile vertex (V1) and an input path edge (E1).

SWEEP_ANNO_VERTEX_LAT annotations are each present with the following outputs: left_lateral_edge (E2), mid_lateral_edge (E3), right_lateral_edge (E4), left_lateral_face (F1), and right_lateral_face (F2).

SWEEP_ANNO_VERTEX_TOP annotations are each present with the following outputs: left_top_vertex (V2), mid_top_vertex (V3), right_top_vertex (V4), left_top_edge (E5), and right_top_edge (E6).

Sweep With Draft Resulting in Degenerate Topology

Topic: *Sweeping

This example demonstrates when the sweep profile degenerates because of sweeping with draft.

Scheme Example

```
(define profile (wire-body:points (list (position 0 0 0)
    (position 3.75 0 0) (position 4 0.25 0) (position 4 4 0)
     (position 0 4 0) (position 0 0 0))))
(define opts (sweep:options "draft_angle" -15))
(define swp (sweep:law profile (gvector 0 0 5) opts))
```

Example 1-38. Demonstrating Degenerate Topology

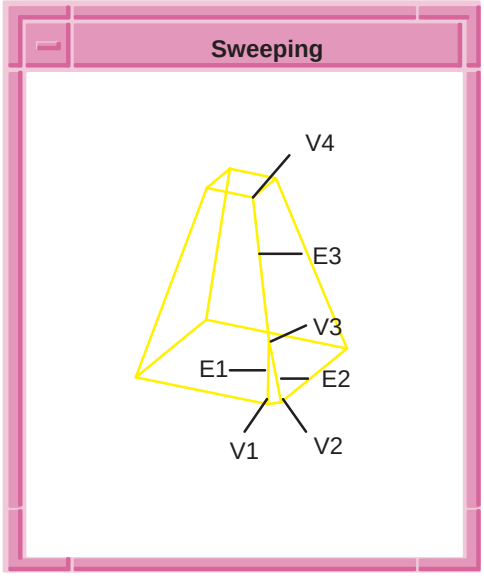


Figure 1-43. Demonstrating Degenerate Topology

Two SWEEP_ANNO_VERTEX_LAT annotations with mid_lateral_edges (E1 and E2, respectively) are present with input profile vertices (V1 and V2, respectively).
The SWEEP_ANNO_VERTEX_LAT annotation with an output mid_lateral_edge (E3) has two input profile vertices (V1 and V2).

A SWEEP_ANNO_VERTEX_TOP annotation with an output mid_top_vertex (V3) has two input profile vertices (V1 and V2).

The SWEEP_ANNO_VERTEX_TOP annotation with an output mid_top_vertex (V4) has two input profile vertices (V1 and V2).

Here is another sweep example demonstrating degeneracy resulting from draft.

Scheme Example

```
(define profile (wire-body:points (list (position 0 0 0)
    (position 4 0 0) (position 4 2 0) (position 0 2 0)
    (position 0 0 0))))
(define opts (sweep:options "draft_angle" -30))
(define swp (sweep:law profile (gvector 0 0 10) opts))
```

Example 1-39. Degenerate Topology

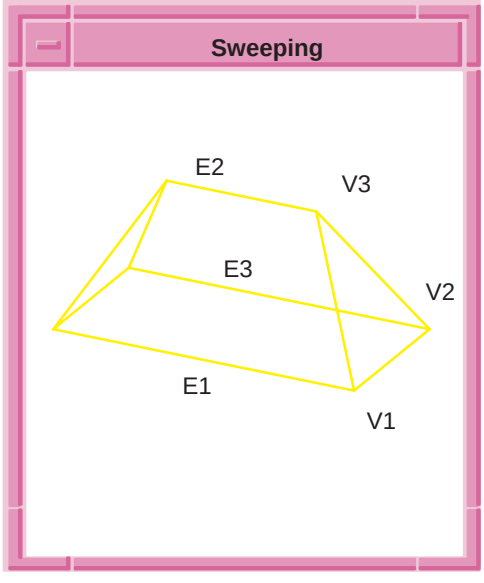


Figure 1-44. Degenerate Topology

The SWEEP_ANNO_EDGE_TOP annotation with an output top_edge (E2) has two input profile edges (E1 and E3).

The SWEEP_ANNO_VERTEX_TOP annotation with an output mid_top_vertex (V3) has two input profile vertices (V1 and V2).

This sweep degenerates to just one vertex.

Scheme Example

```
(define profile (wire-body:points (list (position 0 0 0)
    (position 4 0 0) (position 4 4 0) (position 0 4 0)
    (position 0 0 0))))
(define opts (sweep:options "draft_angle" -30))
(define swp (sweep:law profile (gvector 0 0 10) opts))
```

Example 1-40. Degenerate Topology

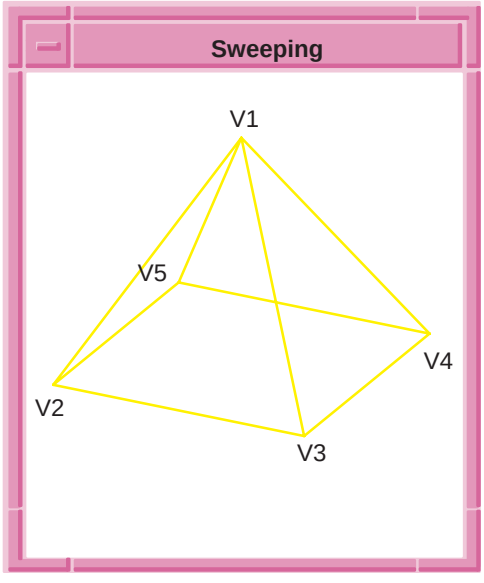


Figure 1-45. Degenerate Topology

The SWEEP_ANNO_VERTEX_TOP annotation with an output mid_top_vertex (V1) has four input profile vertices (V2, V3, V4, and V5).

Sweep With Miter

Topic: *Sweeping

Topology that results from miters will generally not have sweep annotations.

Scheme Example

```
(define profile (wire-body:points (list (position 0 0 0)
    (position 3 0 0) (position 3 3 0) (position 0 3 0)
    (position 0 0 0))))
(define path (wire-body:points (list
    (position 1.5 1.5 0) (position 1.5 1.5 8.5)
    (position 10 1.5 8.5))))
(define opts (sweep:options "miter" "crimp"))
(define swp (sweep:law profile path opts))
(define model (list swp path))
(entity:rotate model 0 0 1 35)
```

Example 1-41. Sweep with Miter

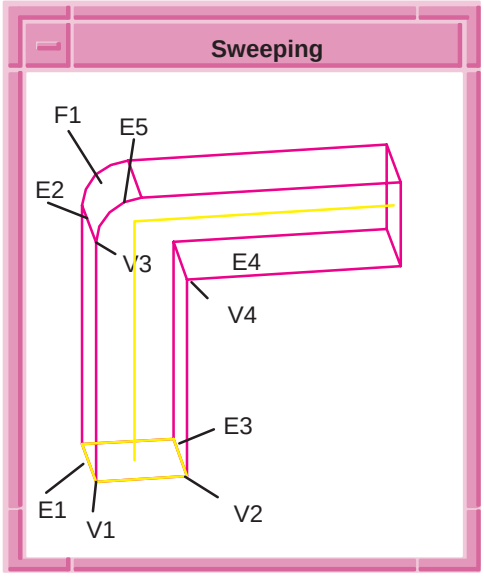


Figure 1-46. Sweep With Miter

Although E2 and E4 are resulting from E1 and E3, there are no annotations on E2 and E4. Similarly there are no annotations on V3 and V4. Edges and vertices such as these that are a result of the sweep operation proceeding through a miter are not annotated.

There is a SWEEP_ANNO_EDGE_LAT annotation with an input profile edge (E1) and an output lateral_face (F1) in this sweep miter that results in the creation of a face. There is also a SWEEP_ANNO_VERTEX_LAT annotation with an input profile vertex (V1) and an output mid_lateral_edge (E5). Most miters do not produce lateral topology like this situation. It should also be noted that the two annotations described here do not have an input path edge.

Sweeps With End Caps Merge

Topic: *Sweeping

The end caps merge when the sweep results in the merging of the start_face and the end_face.

Scheme Example

```
(define profile (wire-body:points (list (position 0 0 0)
    (position 3 0 0) (position 3 3 0) (position 0 3 0)
    (position 0 0 0))))
;; define a sweep angle of 180 degrees.
(define opts (sweep:options "sweep_angle" 180))
(define swp (sweep:law profile (position 0 0 0) (gvector 0 1 0)
     opts))
(zoom-all)
; OUTPUT 180 degree sweep
(define profile2 (wire-body:points (list
    (position 0 5 0) (position 3 5 0) (position 3 8 0)
    (position 0 8 0) (position 0 5 0))))
;; define a sweep angle of 120 degrees.
(define opts2 (sweep:options "sweep_angle" 120))
(define swp2 (sweep:law profile2 (position 0 0 0)
    (gvector 0 1 0) opts2))
;; Size model again for better viewing.
(define body (list swp swp2))
(view:compute-extrema)
(entity:scale body .9)
(refresh-all)
(entity:set-color swp2 3)
; OUTPUT 120 degree sweep
```

Example 1-42. Sweeps With End Caps Merge

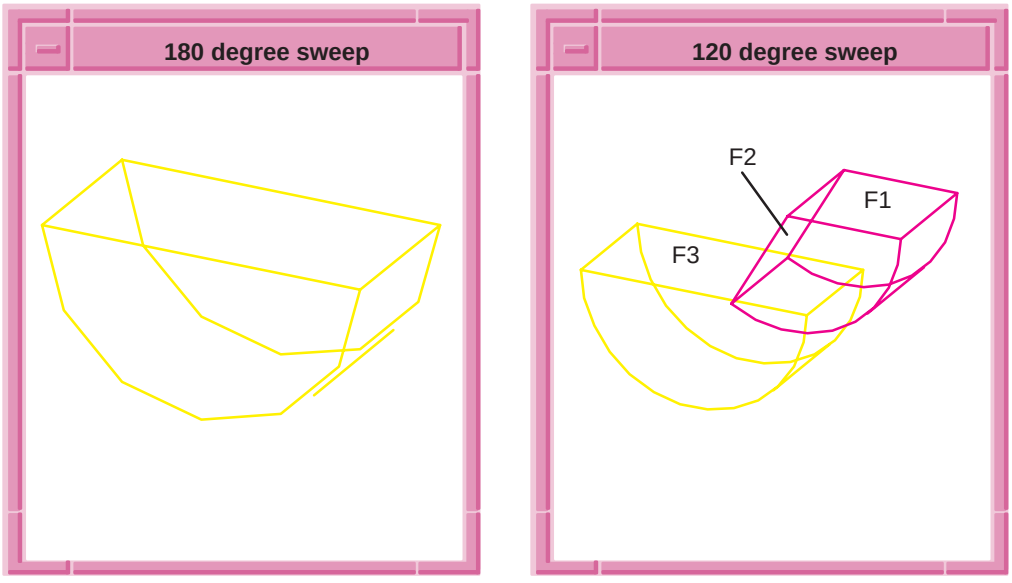


Figure 1-47. Sweeps With End Caps Merge

The first (lower) body in Figure 1-47 is created by sweeping 180 degrees. Here F3 is a start_face and an end_face.

The second (upper) body in Figure 1-47 is created by sweeping 120 degrees. Here there are two end cap faces, F1 being the start_face and F2 being the end_face.

C++ Examples Using Sweeping

Topic: *Sweeping, Examples

This section includes examples to help you get started working with sweeping. These examples show first how to call the sweep API function doing a simple extrude, and then an example for how to use the sweep options to perform more complicated sweeping operations. The examples are C++ **code snippets** for illustrating how to use the functionality. They are not intended to be complete, compilable, or runnable programs.

Using Function api_sweep_with_options

Topic: *Sweeping, Examples

Suppose you have a planar FACE named “prof”, to be used as the profile for the sweep. Define the objects for the API arguments:

```
sweep_options *options = ACIS_NEW sweep_options();
BODY *new_body = NULL;
```

Specify the path as a SPAvector:

```
SPAvector path_dir(¿,¿,¿);
```

The call to the sweeping functionality can be performed with the following:

```
outcome result =
api_sweep_with_options(prof,path_dir,options,new_body);
```

The profile sent into the sweep determines how the resulting body is returned. One way to handle the behavior is to check if the new_body argument is NULL. If it is not NULL, it is the result of the sweep, otherwise the owner of the profile is the result of the sweep.

```
ENTITY *sweep_result = NULL;
if(new_body)
{
    sweep_result = new_body;
}
else
{
    check_outcome(  api_get_owner(prof,sweep_result));
}
```

Using the Sweep Options

Topic: *Sweeping, Examples

Suppose you have a wire–body profile, an EDGE path, and a BODY new_body. There is also an e–shaped BODY, ebody, that the sweep will intersect. The following example sets a few options to demonstrate how selective booleans can be used with sweeping.

```
sweep_options *options = ACIS_NEW sweep_options();
options->set_sweep_to_body(ebody);
options->set_bool_type(LIMIT);
char keep_str[11] = "X<2 OR X=Y";
law *keep_law = NULL;
check_outcome(api_str_to_law(keep_str,&keep_law));
options->set_keep_law(keep_law);
outcome result =
api_sweep_with_options(profile,path,options,new_body);
```