Chapter 4.
Options

Topic: Ignore

Options may be set to modify the behavior of ACIS. An option's value may be a flag (indicating an on/off state), a number (integer or real number), or a string. Options may be set in a Scheme application (such as Scheme AIDE)using the Scheme extension option:set; in the ACIS Test Harness using the command option; or in a C++ application using one of several API functions. Refer to the *3D ACIS Online Help User's Guide* for a description of the fields in the reference template.

extend_mergeable_edges

Option: Local Operations, Modeler Control

Action: Controls whether or not mergeable edges are extended.

Name String: **`extend_mergeable_edges`**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: This option is used when mergeable edges have a NO_MERGE_ATTRIB attribute attached. Certain local operations honor these attributes. When the face on which such a mergeable edge lies grows in size, the user has a choice as to whether the mergeable edge should be extended to the boundary of the face (if the option is on)or retain its original size (if the option is off). Retaining its original size could result in edges that were connected to the boundary of the face before the local operation now becoming disconnected from the boundary.

Not all local operations honor the NO_MERGE_ATTRIB attribute. Those that do include: move faces, offset faces, offset body, sheet thickening (implemented in the Shelling Component), and shelling (implemented in the Shelling Component).

Example:

```
; extend_mergeable_edges
; Offset a body with and without the option on
(define b1 (solid:block (position –10 –10 –10)
    (position 10 10 10)))
;; b1
(define b2 (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b2
; Imprint the FACES and EDGES
(solid:imprint b1 b2)
;; #t
; Remove b2 – we don’t need it anymore
(entity:erase b2)
;; #[entity 3 1]
; Pick the new FACEs
(define f1 (pick:face (ray (position 1 1 1)
    (gvector 1 0 0))1))
;; f1
(define f2 (pick:face (ray (position 1 1 1)
    (gvector 0 1 0))1))
;; f2
(define f3 (pick:face (ray (position 1 1 1)
    (gvector 0 0 1))1))
;; f3
```

```
; Get the EDGEs of the FACEs
(define ed_lst1 (entity:edges f1))
;; ed_lst1
(define ed_lst2 (entity:edges f2))
;; ed_lst2
; Add NO_MERGE_ATTRIBs to the EDGEs
(edge:set-no-merge-attrib ed_lst1)
;; ()
(edge:set-no-merge-attrib ed_lst2)
;; ()
; Make sure the option is turned off
(option:set "extend_mergeable_edges" #f)
;; #t
(lop:offset-body b1 5)
;; #[entity 2 1]
; The mergeable edges with a NO_MERGE_ATTRIB retain
; their size, so they are no longer connected to the
; boundary of the faces
; OUTPUT Original
; Switch the option on.
(option:set "extend_mergeable_edges" #t)
;; #f
; Offset the body
(lop:offset-body b1 5)
;; #[entity 2 1]
; The mergeable edges that have a NO_MERGE_ATTRIB are
; extended, the others are merged out
```

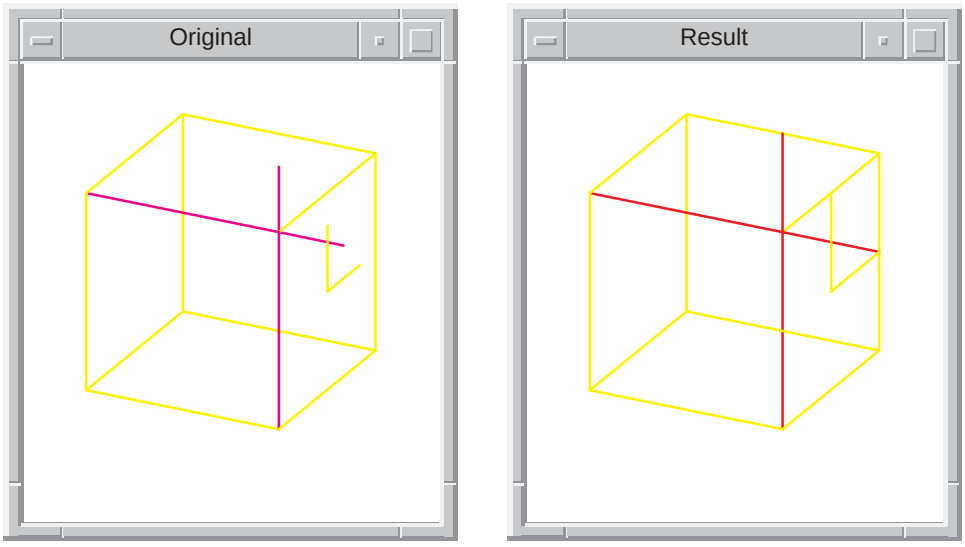

Figure 4-1. extend_mergeable_edges

lop_check_invert

Option: Local Operations, Modeler Control

Action: Controls the checking of invalid faces made in LOP.

Name String: **`lop_check_invert`**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: When this option is on, LOP checks faces and prevents invalid (inverted or inside–out)body faces from being made. If this option is off, body faces may be made and subsequently repaired using the Repair Body Intersections Component.

Example:

```
; lop_check_invert
; Allow inverted face, then repair body
(define b1 (solid:block
    (position –25 –25 –25) (position 25 25 25)))
;; b1
(define b2 (solid:cylinder (position 0 0 0)
    (position 0 0 50)10))
;; b2
(solid:subtract b1 b2)
;; #[entity 2 1]
(render)
;; ()
; Pull hole out of block into a boss, switch off
; lop_invert_checking so invalid inverted body face
; can be made
(option:set ”lop_check_invert” #f)
;; #t
(lop:offset–faces (pick:face (ray
    (position 0 0 0) (gvector 0 0 1))) 50)
;; #[entity 2 1]
; Now repair the body
(rbi:rep–self–int b1)
;; #[entity 2 1]
(render)
;; ()
```

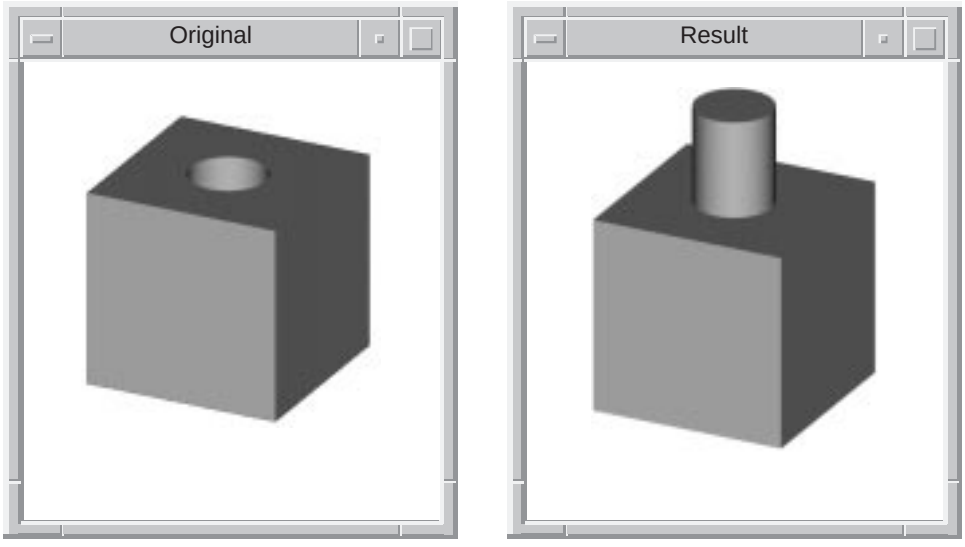

Figure 4-2. lop_check_invert

lop_convert_pipe

Option: Local Operations, Modeler Control

Action: Sets conversion of pipe spline surface.

Name String: **lop_convert_pipe**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: When this option is on, LOP converts pipe spline surfaces to rolling ball blend spline surfaces. This allows selected faces and their neighboring faces to be extended.

Example:

```
; lop_convert_pipe
; Turn off lop_convert_pipe
(option:set ”lop_convert_pipe” #f)
;; #t
```

lop_fail_on_no_part_inv_sf

Option: Local Operations, Modeler Control

Action: Controls whether or not a local operation fails when a face surface can only be partially offset.

Name String:	**lop_fail_on_no_part_inv_sf**		
Scheme:	boolean	#f, #t	#f
Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE

Description: When a face surface can only be partially offset, the geometry returns no surface at all. When this option is set to off, the local operation continues by removing the face. However, this may lead to shells that are too thin or have holes. This can be prevented by switching the option on, in which case the local operation will fail instead of removing the face.

Example:

```
; lop_fail_on_no_part_inv_sf
; Do a partial offset with the option on
(define b1 (solid:block (position –25 –25 –25)
    (position 25 25 25)))
;; b1
(define edge1 (pick:edge (ray (position 0 0 0)
    (gvector 1 0 1))))
;; edge1
(define rad1 (abl:two–ends–rad 10 1))
;; rad1
(abl:edge–blend edge1 rad1)
;; #[entity 3 1]
(blend:network edge1)
;; #[entity 2 1]
; Offset body so blend face surface partially inverts
; Presently no face surface is made but we must not
; remove the face, so switch on the option.
(option:set ”lop_fail_on_no_part_inv_sf” #t)
;; #f
(lop:offset–body b1 –4)
;; Returned edge has been highlighted.
;; *** Error lop:offset–body: no solution for an edge
```

lop_ff_int

Option: Modeler Control, Local Operations

Action: Sets additional validity checking of the body.

Name String:	**lop_ff_int**		
Scheme:	boolean	#f, #t	#f

Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE

Description: Enables additional validity checking of the body after an API call in LOP, SHL, or REM when turned on. The extra checks performed are:

- Checks for improper face/face intersections.
- Checks for improper shell containment between two shells in the same lump.
- Checks for improper lump containment (i.e. one lump containing another).

If any problems are found, the local operation rolls the body back to its original state.

These checks are not performed when this option is FALSE.

Example:

```
; lop_ff_int
; Create some geometry
(define b1 (solid:block (position 0 –10 –10)
    (position 10 10 10)))
;; b1
(define b2 (solid:block (position 15 –10 –10)
    (position 25 10 10)))
;; b2
(bool:unite b1 b2)
;; #[entity 2 1]
; Switch the option on.
(option:set ”lop_ff_int” #t)
;; #f
; Try to make an invalid body
(lop:move–faces (car (pick:face (ray (position 5 0 0)
    (gvector 1 0 0)))) (gvector 10 0 0))
;; entid –58248 entid –55072: Error: face
;; intersection
;; entid –58248 entid –55312: Error: face
;; intersection
;; entid –58248 entid –55552: Error: face
;; intersection
;; entid –58248 entid –56040: Error: face
;; intersection
;; entid –57528 entid –55072: Error: face
;; intersection
;; entid –57528 entid –55312: Error: face
;; intersection
```

```
;; entid –57528 entid –55552: Error: face
;; intersection
;; entid –57528 entid –55800: Error: face
;; intersection
;; entid –57288 entid –55312: Error: face
;; intersection
;; entid –57288 entid –55552: Error: face
;; intersection
;; entid –57288 entid –55800: Error: face
;; intersection
;; entid –57288 entid –56040: Error: face
;; intersection
;; entid –58008 entid –55072: Error: face
;; intersection
;; entid –58008 entid –55312: Error: face
;; intersection
;; entid –58008 entid –55800: Error: face
;; intersection
;; entid –58008 entid –56040: Error: face
;; intersection
;; entid –57048 entid –55072: Error: face
;; intersection
;; entid –57048 entid –55552: Error: face
;; intersection
;; entid –57048 entid –55800: Error: face
;; intersection
;; entid –57048 entid –56040: Error: face
;; intersection
;; entid –57000 entid –55024: Warning: shell
;; intersection
;; entid 1423104 entid 1430328: Warning: lump
;; intersection
;; *** Error lop:move–faces: improper face/face
;; intersection
```

lop_merge_vertex

Option: Local Operations, Modeler Control

Action: Sets merging of redundant vertices.

Name String: **lop_merge_vertex**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1

C++:	logical	FALSE, TRUE	TRUE

Description: The option gives the user the choice of whether to remove presently unmergeable two–edge vertices where two edges on the same geometrical curve meet, but the edge curves may not be joined together in the original model. As LOP recomputes the curves of the faces being operated on, it can make the new curve long enough for both edges and thus allow the vertex to be merged out. If the user wants the vertex to stay, it is placed at a point the same length fraction along the new edges as along the old.

Example:

```
; lop_merge_vertex
; Turn off silhouettes.
(option:set "sil" #f)
;; #t
; Turn on LOP merge vertex.
(option:set "lop_merge_vertex" #t)
;; #t
; Create a body and imprint a block that just touches
; it, to create an extra vertex.
(define cyl1 (solid:cylinder (position 0 0 -25)
    (position 0 0 25)25))
;; cyl1
(define cyl2 (solid:cylinder (position -30 0 30)
    (position 30 0 30)30))
;; cyl2
(define cyl3 (solid:subtract cyl1 cyl2))
;; cyl3
(define block1 (solid:block (position 25 -25 0)
    (position 75 25 50)))
;; block1
(solid:imprint cyl3 block1)
;; #t
(entity:delete block1)
;; ()
; Note the current state for rolling back.
(roll:name-state "before_offset")
;; "before_offset"
; Offset will remove extra vertex
(lop:offset-body cyl3 1)
;; #[entity 2 1]
; Get number of vertices in this entity; there are
; only 2, because the redundant vertex was merged.
(entity:vertices cyl3)
;; (#[entity 5 1] #[entity 6 1])
; Put vertices in the display list so they show up.
(define merge_v1 (dl-item:point (vertex:position
    (car (entity:vertices cyl3)))))
;; merge_v1
(define merge_v2 (dl-item:point (vertex:position
    (car (cdr (entity:vertices cyl3))))))
;; merge_v2
```

```
; Do again but with the lop_merge_vertex option OFF.
(roll ”before_offset”)
;; 2
(option:set ”lop_merge_vertex” #f)
;; #t
; Erase the old vertex markers from the display list
(dl–item:erase merge_v1)
;; ()
(dl–item:erase merge_v2)
;; ()
; Offset will keep the extra vertex
(lop:offset–body cyl3 1)
;; #[entity 2 1]
; Get number of vertices in this entity. There are 3,
; because the redundant vertex was NOT merged
(entity:vertices cyl3)
;; (#[entity 7 1] #[entity 8 1] #[entity 9 1])
; Put vertices in the display list so they show up.
(define nomerge_v1 (dl–item:point (vertex:position
    (car (entity:vertices cyl3)))))
;; nomerge_v1
(define nomerge_v2 (dl–item:point (vertex:position
    (car (cdr(entity:vertices cyl3))))))
;; nomerge_v2
(define nomerge_v3 (dl–item:point (vertex:position
    (car (cdr (cdr (entity:vertices cyl3)))))))
;; nomerge_v3
; Reset environment to defaults
(option:set ”sil” #t)
;; #f
(option:set ”lop_merge_vertex” #t)
;; #f
(dl–item:erase nomerge_v1)
;; ()
(dl–item:erase nomerge_v2)
;; ()
(dl–item:erase nomerge_v3)
;; ()
```

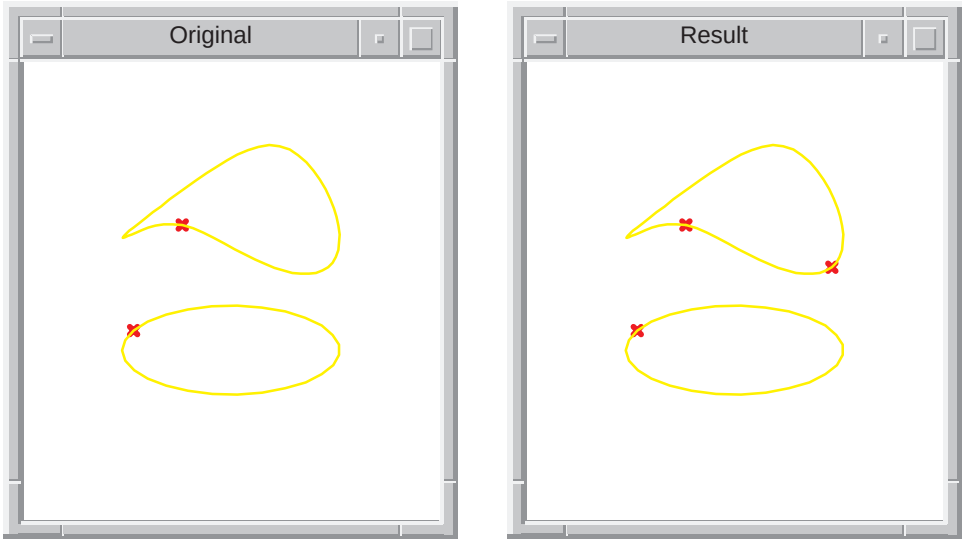

Figure 4-3. lop_merge_vertex

lop_prefer_nearest_sol

Option: Local Operations, Modeler Control

Action: Controls whether the nearest or farthest solution is preferred when picking between multiple solutions in the tweak algorithm.

Name String: **`lop_prefer_nearest_sol`**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: When there are multiple solutions for an edge during a local operation, the tweak algorithm's default behavior is to first try to match the new edge convexity as closely as possible with the old edge convexity. If there are still multiple solutions after this initial pruning, the algorithm chooses the edge solution that is closest to the original as measured with a Euclidean distance at three points along the original and new edges. If this option is on (TRUE), the nearest solution is picked. If it is off (FALSE), the farthest solution is picked.

This option allows the user to modify this default behavior. Because the tweak algorithm is the basis for every local operation (and shelling), this option modifies the behavior of all the local operations, not just tweak.

Example:

```
; lop_prefer_nearest_sol
; This example shows tweak picking the furthest
; solution from the original.
(define opts (sweep:options ”cut_end_off” #t))
;; opts
(define path(edge:law ”vec(cos(x),
    sin(x),x/5)” 0 20))
;; path
(define profile(edge:ellipse (position 1 0 0)
    (gvector 0 1 0) (gvector .2 0 0)))
;; profile
(define sweep(sweep:law profile path opts))
;; sweep
(entity:delete path)
;; ()
(define f (pick:face (ray (position 1 .01 0)
    (gvector 0 –1 0))))
;; f
(option:set ”lop_prefer_nearest_sol” #f)
;; #t
(lop:tweak–faces f f #t)
#[entity 4 1]
```

lop_prefer_same_convexity_sol

Option: Local Operations, Modeler Control

Action: Controls whether to pick a solution that most closely matches the original edge convexities, or that has the least common edge convexities.

Name String: **lop_prefer_same_convexity_sol**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: When there are multiple solutions for an edge during a local operation, the tweak algorithm’s default behavior is to first try to match the new edge convexity as closely as possible with the old edge convexity. If there are still multiple solutions after this initial pruning, the algorithm chooses the edge solution that is closest to the original as measured with a Euclidean distance at three points along the original and new edges.

This option allows the user to modify this default behavior. Because the tweak algorithm is the basis for every local operation (and shelling), this option modifies the behavior of all the local operations, not just tweak.

If this option is on (true), the algorithm picks the solution that has the same (closest) convexity.

Example:

```
; lop_prefer_same_convexity_sol
; This example shows a concave solution being picked
; over a convex solution when the original edge
; geometry is tangent convex.
; Make sure the option is turned on
(option:set "lop_prefer_same_convexity_sol" #t)
;; #t
(define cylinder1 (solid:cylinder (position 0 0 -10)
    (position 0 0 10) 10))
;; cylinder1
(define block1 (solid:block (position -20 -20 -10)
    (position 10 0 10)))
;; block1
(bool:unite block1 cylinder1)
;; #[entity 3 1]
(define f (pick:face (ray (position 0 -15 0)
    (gvector 1 0 0))))
;; f
(entity:set-highlight f #t)
;; #[entity 4 1]
(lop:move-faces f (gvector -10 0 0))
;; #[entity 3 1]
; Roll back one step
(roll)
;; -1
; Switch the option to off and repeat the two
; previous steps highlighting the same face
(option:set "lop_prefer_same_convexity_sol" #f)
;; #t
(define f (pick:face (ray (position 0 -15 0)
    (gvector 1 0 0))))
;; f
(entity:set-highlight f #t)
;; #[entity 5 1]
(lop:move-faces f (gvector -10 0 0))
;; #[entity 3 1]
```

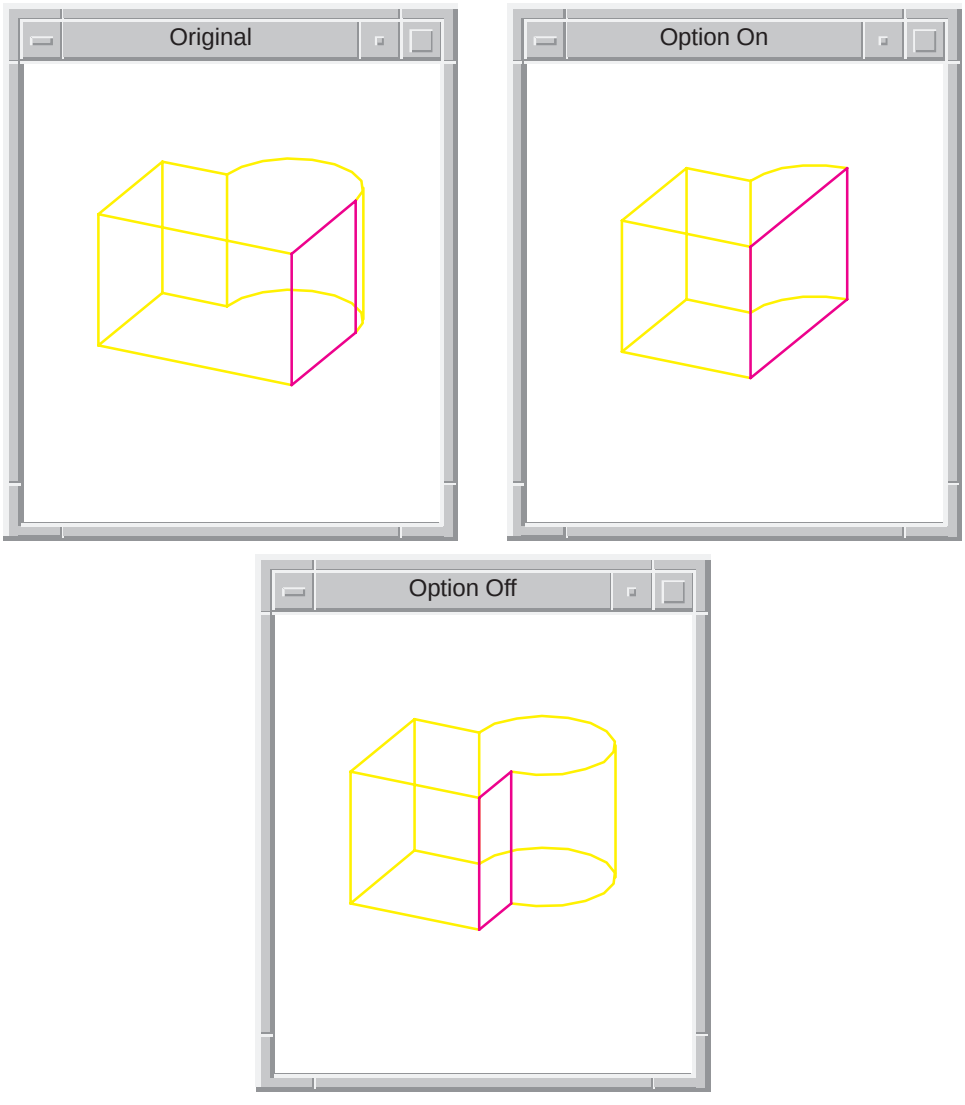

Figure 4-4. lop_prefer_same_convexity_sol

lop_repair_self_int

Option: Local Operations, Modeler Control

Action: Controls whether or not self–intersections are repaired on the bodies that result from local operations and shelling.

Name String:	**lop_repair_self_int**		
Scheme:	boolean	#f, #t	#f
Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE

Description: This option controls whether or not self–intersections are repaired on the bodies that result from local operations and shelling. If this option is on, api_repair_body_self_ints is called at the end of a successful local operation and supplied with the changed body faces. It is therefore also called during shelling when the LOP API api_offset_faces is called.

If repairing self–intersections is required, option lop_check_invert must be turned off, so that the elementary LOP self–intersection checks are disabled and the operation produces a possibly invalid body to pass to API api_repair_body_self_ints.

Example:

```
; lop_repair_self_int
; Repair self–intersections during a local operation
; Create a block
; Make sure the option is off
(option:set ”lop_repair_self_int” #f)
;; #f
(define b1 (solid:block (position –25 –25 –25)
    (position 25 25 25)))
;; b1
; Create a cylinder
(define b2 (solid:cylinder (position 0 0 –25)
    (position 0 0 25) 10))
;; b2
; Subtract the cylinder from the block
(solid:subtract b1 b2)
;; #[entity 2 1]
; OUTPUT Original

; Turn off the lop_check_invert option
(option:set ”lop_check_invert” #f)
;; #t
(lop:move–faces (pick:face (ray (position 0 0 0)
    (gvector 1 0 0)) 1) (gvector 25 0 0))
;; #[entity 2 1]
; OUTPUT Option Off
```

```
; Roll back one step
(roll)
;; –1
(option:set ”lop_repair_self_int” #t)
;; #f
; Make a self–intersecting body with
; self–intersecting faces
(lop:move–faces (pick:face (ray (position 0 0 0)
    (gvector 1 0 0)) 1) (gvector 25 0 0))
;; #[entity 2 1]
; OUTPUT Option On
```

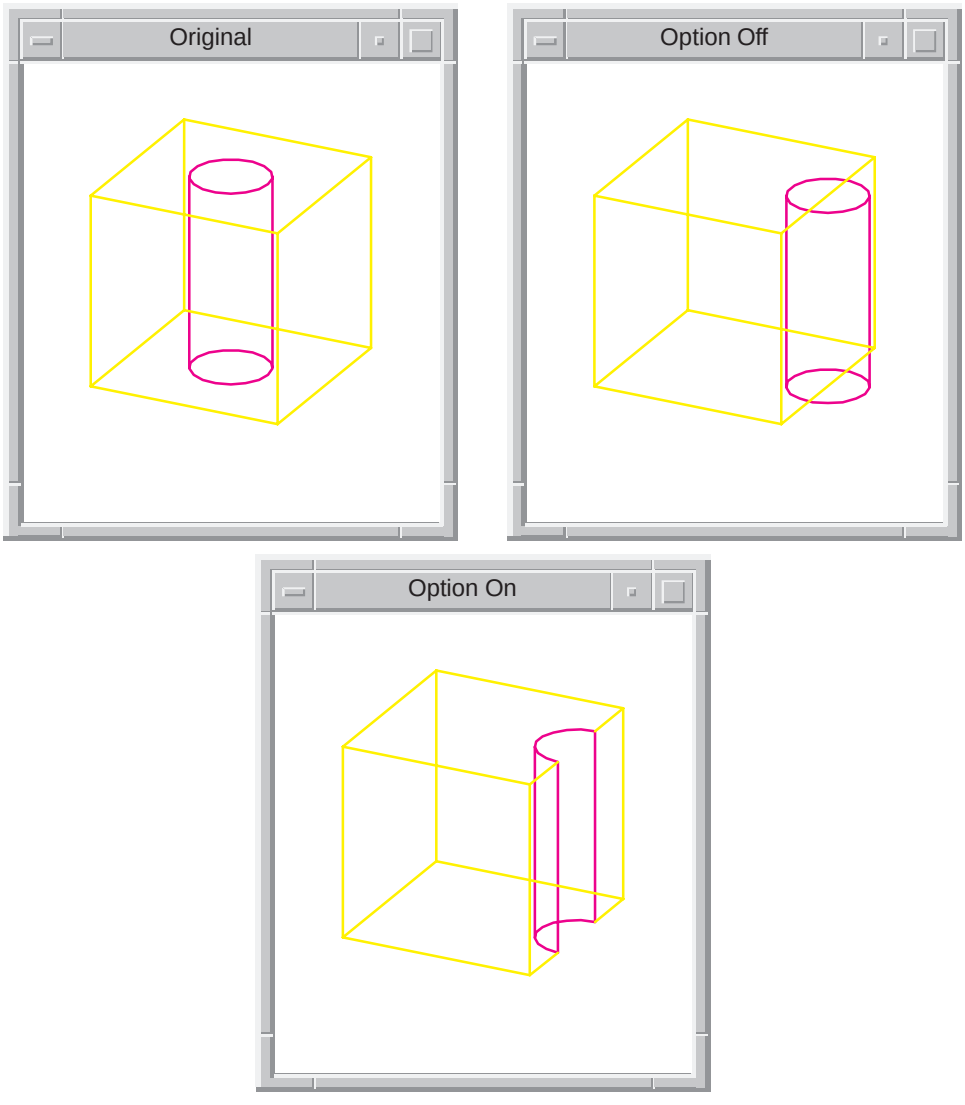

Figure 4-5. lop_repair_self_int

lop_sort_on_convexity

Option: Local Operations, Modeler Control

Action: Controls whether sorting of solutions proceeds on convexity and then distance or on distance and then on convexity.

Name String:	**lop_sort_on_convexity**		
Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: When there are multiple solutions for an edge during a local operation, the tweak algorithm's default behavior is to first try to match the new edge convexity as closely as possible with the old edge convexity. If there are still multiple solutions after this initial pruning, the algorithm chooses the edge solution that is closest to the original as measured with a Euclidean distance at three points along the original and new edges.

This option allows the user to modify this default behavior. Because the tweak algorithm is the basis for every local operation (and shelling), this option modifies the behavior of all the local operations, not just tweak.

If this option is on (true), sorting is first done on convexity, and then on distance. If the option is off (false), sorting is done on distance first and then on convexity.

Example:

```
; lop_sort_on_convexity
; The following example shows tweak picking a nearer
; but concave solution over a convex solution for an
; edge. (The option is off.)
(define cylinder (solid:cylinder (position 0 0 –10)
    (position 0 0 10) 10))
;; cylinder
(define block (solid:block (position –20 –20 –10)
    (position 10 0 10)))
;; block
(bool:unite block cylinder)
;; #[entity 6 1]
(define f (pick:face (ray (position 0 –15 0)
    (gvector 1 0 0))))
;; f
(entity:set–highlight f #t)
;; #[entity 7 1]
```

```
(define block1 (solid:block (position 9.8 –22 –11)
    (position 10.2 15 11)))
;; block1
(entity:transform block1 (transform:rotation
    (position 10 –2 0) (gvector 0 0 1) 45))
;; #[entity 8 1]
(entity:fix–transform block1)
;; #[entity 8 1]
(define t (car (pick:face (ray (position 12 –10 0)
    (gvector 0 1 0)))))
;; t
(entity:set–highlight t #t)
;; #[entity 9 1]


; Make sure the option is off
(option:set ”lop_sort_on_convexity” #f)
;; #t
(lop:tweak–faces f t #f)
;; #[entity 6 1]
(entity:delete block1)
;; ()
```

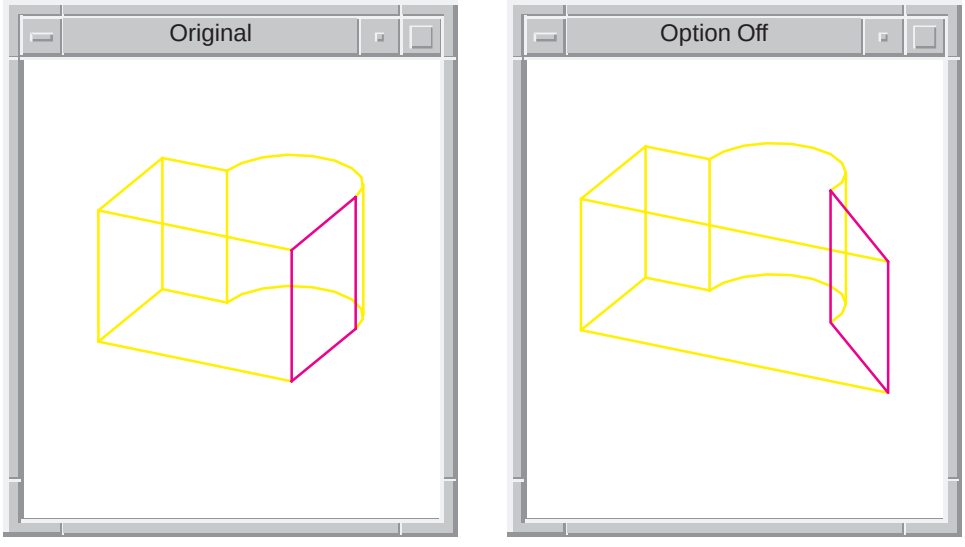

Figure 4-6. lop_sort_on_convexity off

```
; example 2: option on
; RESTART SCHEME AIDE AND REDO EXAMPLE WITH OPTION ON
; This example shows tweak picking a farther
; but convex solution.
(define cylinder (solid:cylinder (position 0 0 –10)
    (position 0 0 10) 10))
;; cylinder
(define block (solid:block (position –20 –20 –10)
    (position 10 0 10)))
;; block
(bool:unite block cylinder)
;; #[entity 3 1]
(define f (pick:face (ray (position 0 –15 0)
    (gvector 1 0 0))))
;; f
(entity:set–highlight f #t)
;; #[entity 4 1]

(define block1 (solid:block (position 9.8 –22 –11)
    (position 10.2 15 11)))
;; block1
(entity:transform block1 (transform:rotation
    (position 10 –2 0) (gvector 0 0 1) 45))
;; #[entity 5 1]
(entity:fix–transform block1)
;; #[entity 5 1]
(define t (car (pick:face (ray (position 12 –10 0)
    (gvector 0 1 0)))))
;; t
(entity:set–highlight t #t)
;; #[entity 6 1]

; Make sure the option is on
(option:set ”lop_sort_on_convexity” #t)
;; #t
(lop:tweak–faces f t #f)
;; #[entity 3 1]
(entity:delete block1)
;; ()
```

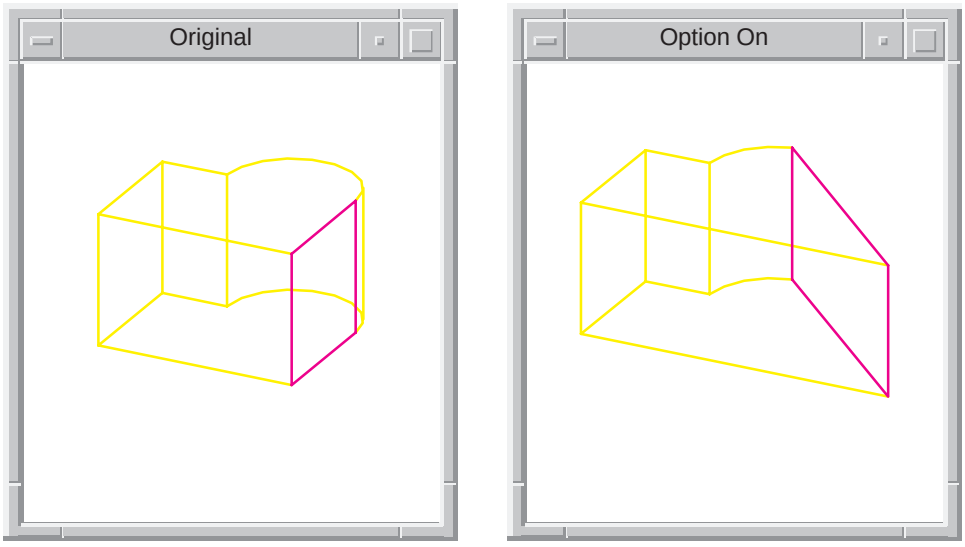

Figure 4-7. lop_sort_on_convexity on

lop_stack_limit

Option: Local Operations, Modeler Control

Action: Controls how much recursion can be done in the LOP code.

Name String: **`lop_stack_limit`**

Scheme:	integer	> 0	512
Test Harness:	integer	> 0	512
C++:	int	> 0	512

Description: This option limits how much recursion can be done in the LOP code in order to prevent crashes due to stack overflows. The value given by the option will relate to the machine and operating system being used. The range of values for the option is any positive integer value. However, it is not intended for the option to be set very low, because this will cause many simple examples (such as the following one) to fail unnecessarily.

Example:

```
; lop_stack_limit
; This will work with the default stack limit
(option:set ”lop_repair_self_int” #t)
;; #f
(define b1 (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; b1
(define cface (face:cone (position 0 0 0)
    (position 0 30 0) 1 25 ))
;; cface
(lop:tweak–faces (pick:face (ray (position 19 19 19)
    (gvector 0 1 0 ))) cface #f)
;; #[entity 11 1]
; Set the stack limit very low, and it fails
; unnecessarily
(option:set ”lop_repair_self_int” #t)
;; #t
(option:set ”lop_stack_limit” 6)
;; 512
(define b1 (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; b1
(define cface (face:cone (position 0 0 0)
    (position 0 30 0) 1 25 ))
;; cface
(lop:tweak–faces (pick:face (ray (position 19 19 19)
    (gvector 0 1 0 ))) cface #f)
;; #[entity 14 1]
```

lop_use_euclidean_dist_score

Option: Local Operations, Modeler Control

Action: Controls whether Euclidean or parametric distance is used when scoring different solutions.

Name String: **lop_use_euclidean_dist_score**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: When there are multiple solutions for an edge during a local operation, the tweak algorithm's default behavior is to first try to match the new edge convexity as closely as possible with the old edge convexity. If there are still multiple solutions after this initial pruning, the algorithm chooses the edge solution that is closest to the original as measured with a Euclidean distance at three points along the original and new edges.

This option allows the user to modify this default behavior. Because the tweak algorithm is the basis for every local operation (and shelling), this option modifies the behavior of all the local operations, not just tweak.

If this option is on (true), Euclidean distance is used to score the different possible solutions.

Example:

```
; lop_use_euclidean_dist_score
; This example shows tweak taking the nearer
; parametric solution, which is different for the
; closest solution measured with Euclidean distance.
(define plist(list (position 0 –8 0)
    (position 0 0 0 ) (position 3 4 0)
    (position 3.5 3.8 0) (position 6 0 0)
    (position 5.5 –7.8 0) (position 5 –8 0)
    (position 4.5 –7.8 0) (position 4 0 0)
    (position 3.5 1.8 0) (position 3 2 0)
    (position 2.5 1.8 0) (position 2 0 0)
    (position 2 –8 0)))
;; plist
(define start(gvector 0 1 0))
;; start
(define end(gvector 0 –1 0))
;; end
(define path(edge:spline plist start end))
;; path
```

```
(define profile(edge:ellipse
    (position 0 –8 0) (gvector 0 1 0).25))
;; profile
(define sweep(sweep:law profile path))
;; sweep
; Zoom to see the whole model
(zoom–all)
;; #[view 1076895352]
(define f (pick:face (ray (position 0 –7.8 0)
    (gvector 0 –1 0))))
;; f
(define block (solid:block (position 0 2 –2)
    (position 12 2.1 2)))
;; block
(entity:transform block (transform:rotation
    (position 1 2 0) (gvector 0 0 1) –60))
;; #[entity 6 1]
(entity:fix–transform block)
;; #[entity 6 1]
(define t (car (pick:face (ray (position 1 1.9 0)
    (gvector 0 1 0)))))
;; t
; Make sure that option is off
(option:set ”lop_use_euclidean_dist_score” #f)
;; #t
(lop:tweak–faces f t #t)
;; #[entity 4 1]
```