Chapter 1.
Local Operations Component

Component: *Local Operations

The Local Operations Component (LOP), in the lop directory, provides a way to locally manipulate the geometry of a face or set of faces in a prescribed way without changing the topology of the solid model. LOP allows existing model features to be manipulated without the use of Boolean operations. The integrity of the model's topology and geometry remains intact as features are relocated or adjusted.

There are four types of local operations:

Tweak . . . . . tweaks an array of faces by calculating new geometric definitions for each edge and vertex of the tweaked faces to maintain the original ACIS body.

Move . . . . . calculates the new face surfaces by applying a transform to the old face surfaces. Move includes sweep–more, where a face is transformed along a sweep path.

Offset . . . . . calculates the new face surfaces by offsetting the old face surfaces by a given distance.

Taper . . . . . has three types: plane taper, edge taper, and shadow taper. Tapers are primarily used for modifying dies or molds to allow the work piece to be removed easily, but are also of use in general modeling.

Plane taper calculates the new face surfaces by rotating the rulings about their intersection with the supplied taper plane.

Edge taper calculates the new face surfaces by rotating the rulings about the specified face edge curves.

Shadow taper uses ruled faces to remove regions where the face is considered to be "in the shadows" (face regions that have normal vectors perpendicular to the draft direction) from the specified faces.

LOP is not usually used through tweak, but rather through the other operations, such as moving or offsetting. These automatically create the new surfaces for a face.

Use api_initialize_local_ops to access the libraries, and api_terminate _local_ops when local operations are finished.

A rich set of options for local operations can be used to set error data and validation, order of operations, whether or not to use repair functions, default behaviors, and preferences for solutions. Some options are mutually exclusive; be sure to refer to the reference material for each option.

If the lop_repair_self_int option is on, code from the Repair Body Intersections Component is called at the end of a successful local operation to fix improper intersections. It is recommended that this option is switched on for shelling and offsetting operations, to insure that self intersecting bodies are healed. However, it is recommended this option not be turned on for tweaks, where lack of geometrical information on the final model shape makes it hard to determine which of the possible solutions were intended.

Tweak

Topic: *Local Operations

Scheme Extensions: lop:tweak–faces, lop:tweak–faces–extend, lop:tweak–faces–init, lop:convert–pipes

C++ APIs: api_convert_pipes, api_tweak_faces, api_tweak_fix_edge, api_tweak_fix_vertex, api_tweak_extend_faces, api_tweak_query_edge_solutions, api_tweak_pick_edge_solution

The tweak algorithm is the foundation for all other local operations. It can change the surfaces of any number of faces to any other surfaces supplied by the user, providing that the surfaces intersect appropriately and that necessary topology changes are supported. Specific faces can be selected, or an array of faces. In this example, the top surface of the block is tweaked to the tool face.

Scheme Example

```
; lop:tweak–faces
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Create a tool face
(define face1 (face:plane (position –10 –10 0) 20 20
    (gvector 0 0 1)))
;; face1
; OUTPUT Original

; Tweak face of the body
(lop:tweak–faces (pick:face (ray (position 10 0 0)
    (gvector 0 0 1))) face1 #f)
;; #[entity 2 1]
; OUTPUT Result
```

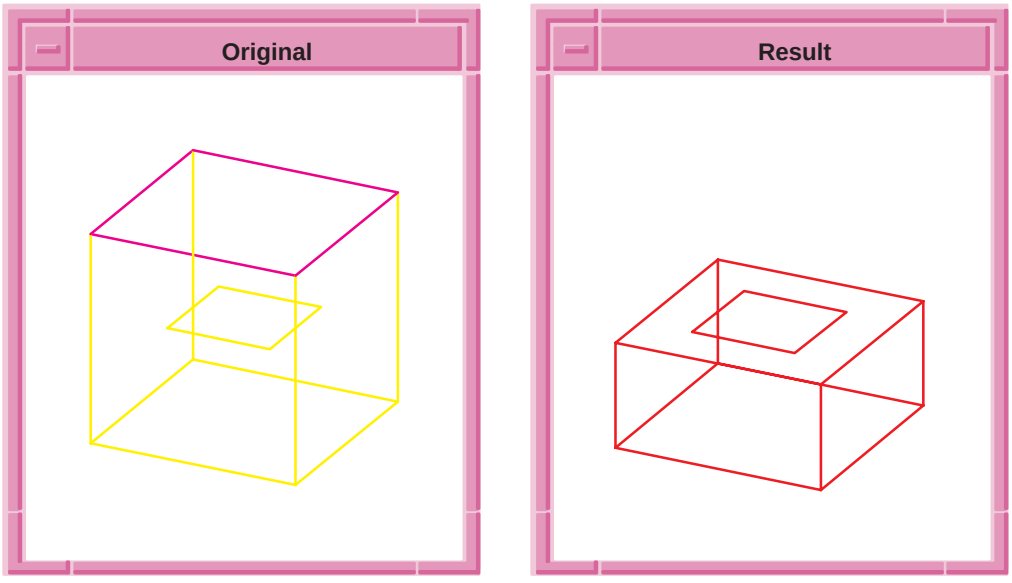

Figure 1-1. lop:tweak–faces

Move

Topic: *Local Operations

Scheme Extensions: lop:move–faces, lop:rotate–faces, lop:sweep–more

C++ APIs: api_move_faces, api_sweep_more

The move API moves an array of faces through a transform, which replaces the surfaces of the specified faces with the surfaces which are moved by the transform. The translation vector of the transform must be non–zero or an error results.

In the first example, the top surface of the block is replaced by the surface moved by the gvector transform. In the second example, lop:sweep–more extends the end face of the swept spring.

Scheme Example

```
; Move Example 1 – lop:move–faces
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; OUTPUT Original
```

```
; List the faces of the block
(entity:faces block1)
;; (#[entity 3 1] #[entity 4 1] #[entity 5 1]
;; #[entity 6 1] #[entity 7 1] #[entity 8 1])
; Move a face on the body
(lop:move–faces (entity 4) (gvector 10 10 20))
;; #[entity 2 1]
; OUTPUT Result
```

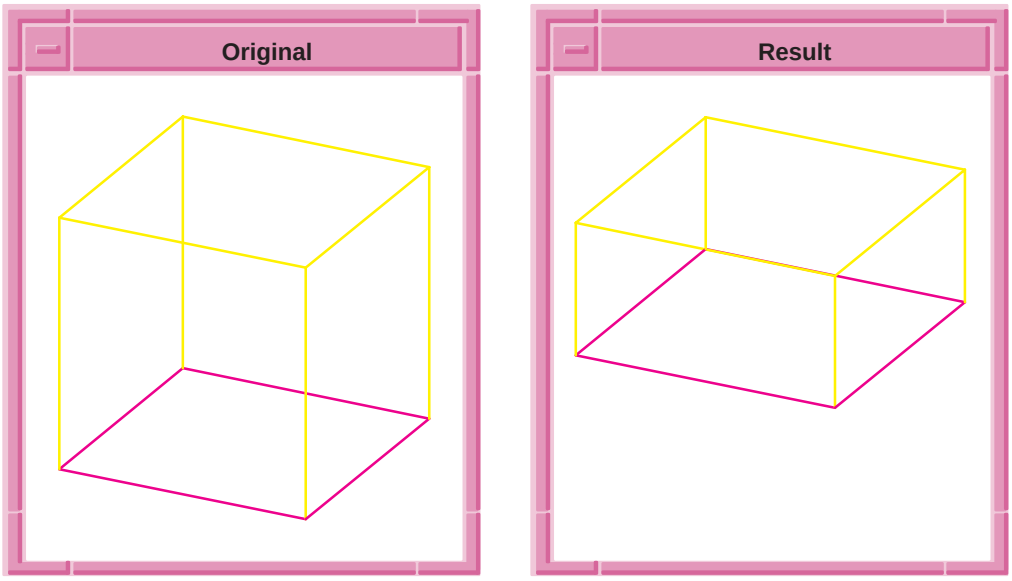

Figure 1-2. lop:move–faces

```
; Example 2 – lop:sweep–more
; Make a spring
(define path
    (edge:law ”vec(cos(x),sin(x),x/10)” 0 10))
;; path
; Define a profile
(define profile
    (edge:ellipse (position 1 0 0)
    (gvector 0 1 0) (gvector .2 0 0)))
;; profile
(define sweep (sweep:law profile path))
;; sweep
; sweep => #[entity 4 1]
; Zoom to see the model
(zoom–all)
;; #[view 1076896136]
(render:rebuild)
;; ()
; OUTPUT Original and OUTPUT Rendered Original

; Pick the face to sweep more
(define f (pick:face (ray (position 1 0 0)
    (gvector 0 –1 0))))
;; f
(lop:sweep–more f 1)
;; #[entity 4 1]
(render)
;; ()
; OUTPUT Result and OUTPUT Rendered Result
```

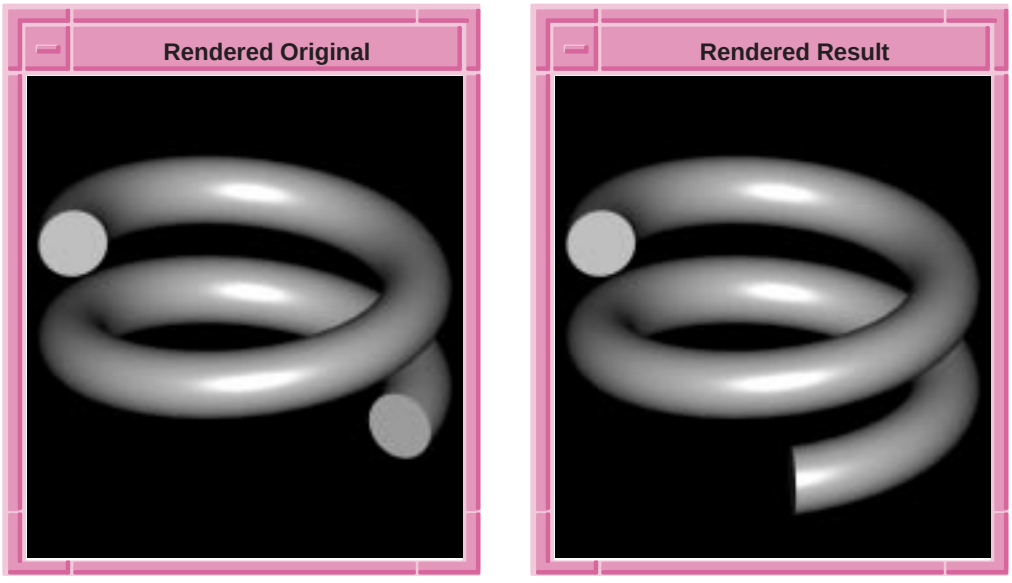

Figure 1-3. lop:sweep–more

Offset

Topic: *Local Operations

Scheme Extensions: lop:offset–body, lop:offset–faces, lop:offset_specific

C++ APIs: api_offset_body, api_offset_faces, api_offset_faces_specific, api_offset_planar_face_loops

The offset APIs offset a body or an array of faces by an offset distance. In face offsetting and body offsetting, radial faces whose geometry *would* degenerate as a result of the offset are removed prior to the offset. Thus, if a body with blended edges or vertices is offset inwards by more than the blend radius, the blends are removed prior to the offset. This uses the “remove faces” functionality (e.g., shells may be split). If some faces can not be removed, the offset fails.

In the first example, the top surface is offset vertically. In the second example, all the surfaces are offset, resulting in a larger block.

Scheme Example

```
; Offset Example 1 – lop:offset–faces
; Create a solid block.
(define block1
    (solid:block (position –10 –10 –20)
    (position 5 30 20)))
;; block1
; OUTPUT Original

; Offset a face on the body
(lop:offset–faces (pick:face (ray (position 0 0 0)
    (gvector 0 0 1))) 10)
;; #[entity 2 1]
; OUTPUT Result
```

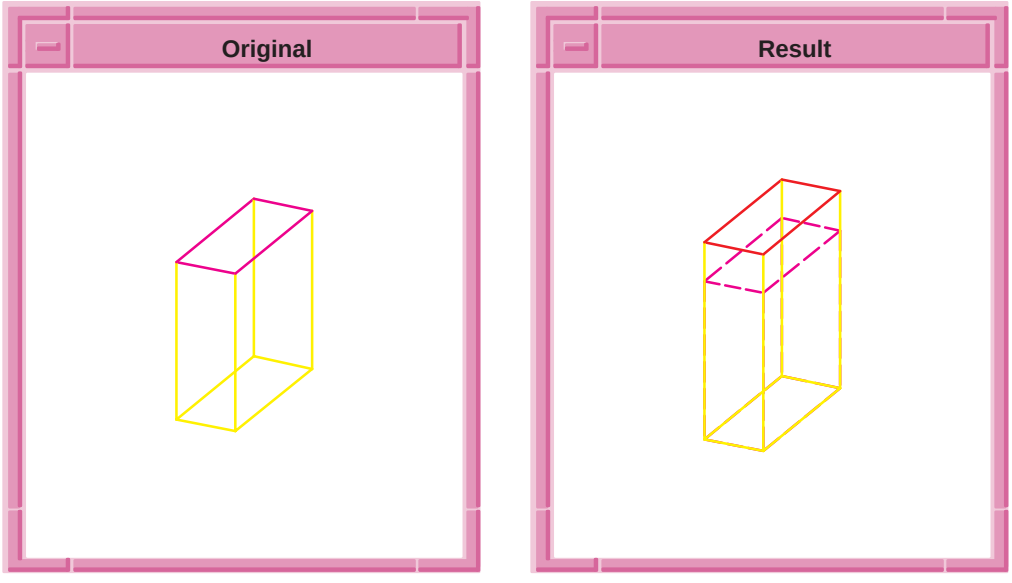

Figure 1-4. lop:offset–faces

```
; Offset Example 2 – lop:offset–body
; Create a solid block.
(define block1
    (solid:block (position –10 –10 –20)
    (position 20 15 20)))
;; block1
; OUTPUT Original
```

```
; Offset the body
(lop:offset-body block1 15)
;; #[entity 2 1]
; OUTPUT Result
```

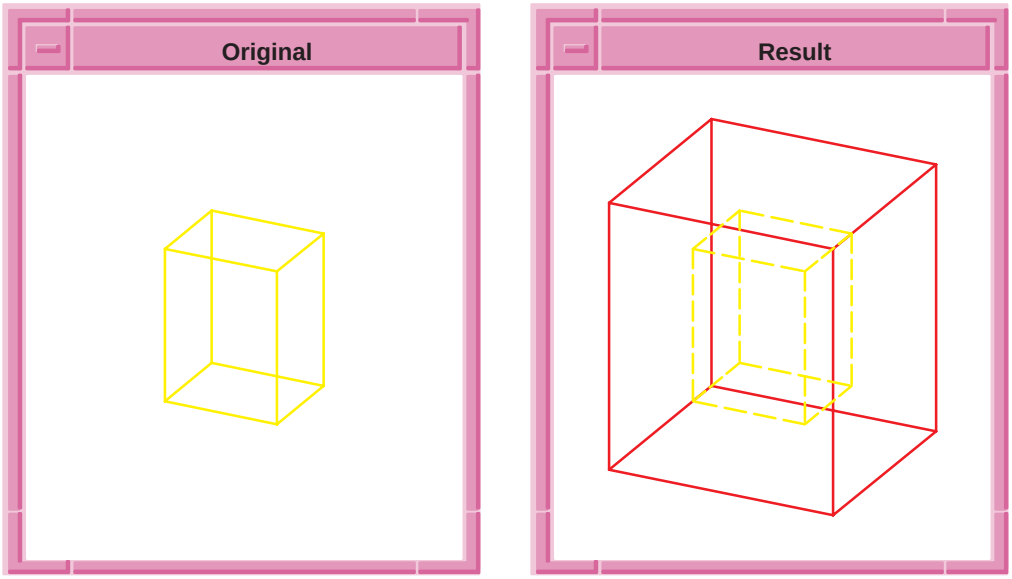

Figure 1-5. lop:offset–body

Taper

Topic: *Local Operations

Scheme Extensions: lop:taper–faces, lop:edge–taper–faces

C++ APIs: api_taper_faces, api_shadow_taper_faces, api_edge_taper_faces

A plane taper is often used to facilitate extraction of a part from a mold rather than to change the general appearance of a model. Typically the angle supplied would be very small. It replaces the surfaces of the supplied faces with surfaces tapered by a draft angle about the intersection between the draft plane and the surface. Whether the taper is executed or just validated is controlled by the option validate_lop.

The taper operation can include *vent* faces. If one face of a mergeable edge is to be tapered and the edge does not lie in the draft plane, the taper algorithm adds a vent face. The vent face lies between the edge on the face that is not tapered and the corresponding edge on the tapered face. If the mergeable edge intersects the draft plane, the edge is split at the intersection point and two vent faces are inserted. The simplest surface definition possible is used for the vent face. All curves on planar or conical faces are supported.

In a shadow taper, the shadow is caused by a beam of light traveling with the draft direction and angle. The shadow taper functionality fills in gaps that are not visible from a specified direction. It detects special geometry cases on spheres, cones, and tori, and makes a cone or plane for the taper surface rather than a spline.

Scheme Example

```
; Taper Example 1 – lop:taper–faces
(define block2
    (solid:block (position –25 –25 –10)
    (position 25 25 10)))
;; block2
(solid:blend–edges (pick:edge (ray
    (position 0 0 0) (gvector 1 –1 0))) 20)
;; (#[entity 4 1])
; OUTPUT Original

(lop:taper–faces (list (pick:face
    (ray (position 0 0 0) (gvector 1 0 0)))
    (pick:face (ray (position 0 0 0)
    (gvector 0 –1 0))) (pick:face
    (ray (position 0 0 0) (gvector 1 –1 0))))
    (position 0 0 –10) (gvector 0 0 1) 45)
;; #[entity 4 1]
; OUTPUT Result
```

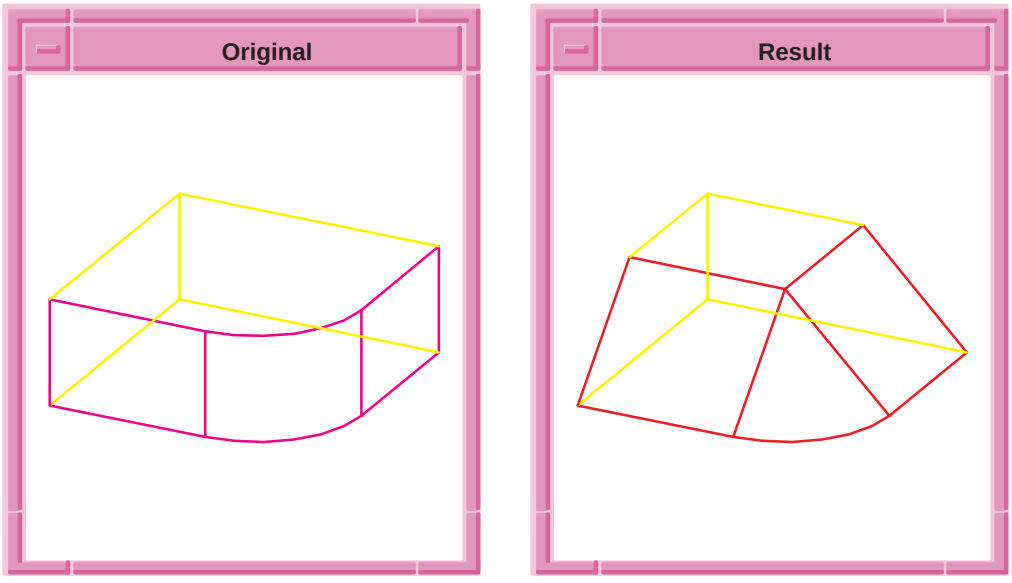

Figure 1-6. lop:taper–faces

```
; Taper Example 2 – lop:shadow–taper–faces
; Create a solid block
(define block1
    (solid:block (position –25 –25 –10)
    (position 25 25 10)))
;; block1
; Create a solid cone
(define cone1
    (solid:cone (position –20 0 10)
    (position 20 0 10) 20 10))
;; cone1
; Unite the cone and block
(bool:unite block1 cone1)
;; #[entity 2 1]
; OUTPUT Original

; Shadow taper a face on the body
(lop:shadow–taper–faces
    (pick:face (ray (position 0 0 0)
    (gvector 0 0 1))) (gvector 0 0 1) 20)
;; #[entity 2 1]
; OUTPUT Result
```

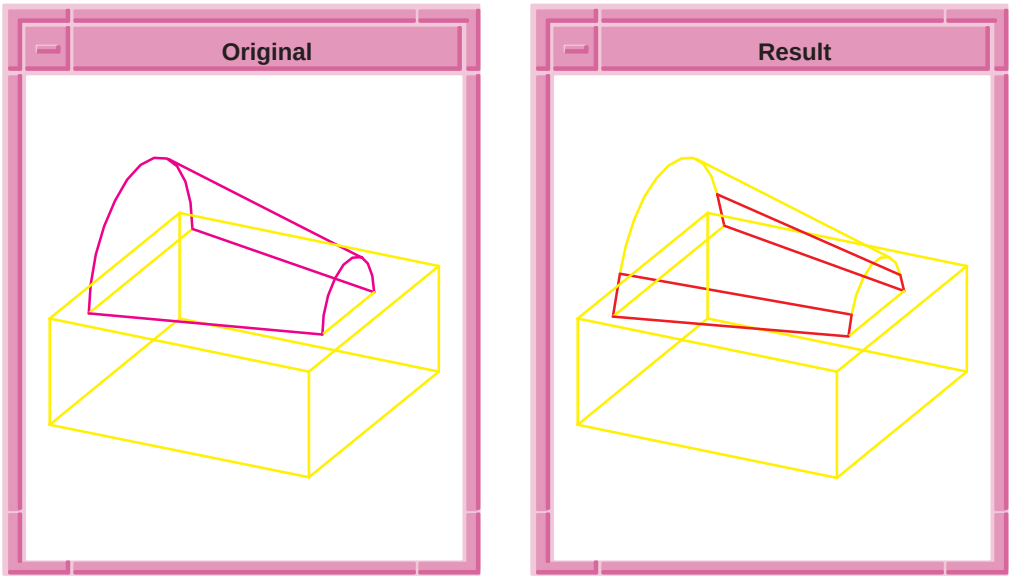

Figure 1-7. lop:shadow–taper–faces

```
; Example 3 – edge taper merging vent faces.
; Define some entities
(part:clear)
;; #t
(define block1 (solid:block
    (position –25 –25 –25) (position 25 25 25)))
;; block1
(define block2 (solid:block
    (position –50 –50 0) (position 50 50 100)))
;; block2
(solid:imprint block1 block2)
;; #t
(entity:delete block2)
;; ()
(option:set ”annotation” #t)
;; #f
(option:set ”unhook_annotations” #f)
;; #t
(define e1 (pick:edge
    (ray (position 0 0 –25) (gvector 1 0 0))))
;; e1
(define f1 (pick:face
    (ray (position 0 0 –20) (gvector 1 0 0))))
;; f1
(define e2 (pick:edge (ray
    (position 0 0 –25) (gvector –1 0 0))))
;; e2
(define f2 (pick:face (ray
    (position 0 0 –20) (gvector –1 0 0))))
;; f2
(define e3 (pick:edge
    (ray (position 0 0 –25) (gvector 0 1 0))))
;; e3
(define f3 (pick:face
    (ray (position 0 0 –20) (gvector 0 1 0))))
;; f3
(define e4 (pick:edge
    (ray (position 0 0 –25) (gvector 0 –1 0))))
;; e4
(define f4 (pick:face
    (ray (position 0 0 –20) (gvector 0 –1 0))))
;; f4
; OUTPUT Original
```

```
(lop:edge-taper-faces (list f1 f2 f3 f4)
    (list e1 e2 e3 e4) (gvector 0 0 1) 20)
;; #[entity 24 1]
; OUTPUT Result

; Render to see the final result.
(render)
;; ()
; OUTPUT Rendered
```

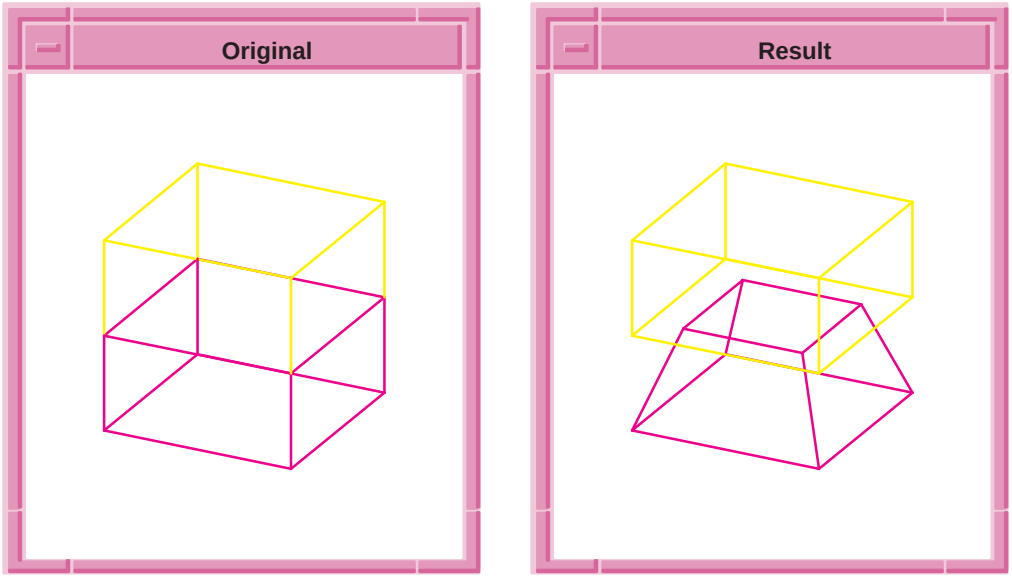

Figure 1-8. lop:edge–taper–faces

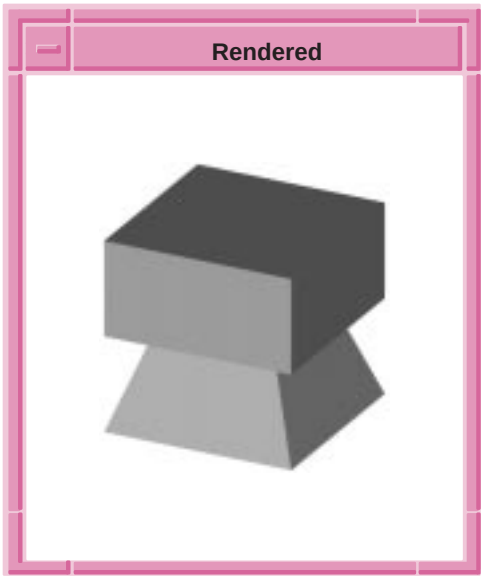

Figure 1-9. lop:edge–taper–faces, rendered

Topology

Topic: *Local Operations

The core LOP capability is to be able to intersect unbounded surfaces and curves to create ACIS edges and vertex topology. Two types of intersections are used to support this:

- Intersection of two unbounded surfaces to get a single curve.
- Intersection of an unbounded edge with an unbounded surface to get a single point. In some cases curve–curve intersections are preferred, but not typically.

LOP expects to see edges and vertices from the intersections at all times. If more than one edge or vertex position is possible, LOP will test each to determine which is best. It fails whenever this condition is violated, as when the unbounded surfaces (or unbounded curves) never intersect (e.g., two parallel planes).

For example, the operation for moving faces requires that the underlying curves be recomputed for the edges of each face being moved, as well as the face's vertex coordinates. If the top face of a cube is moved, each of the four top edges' curves are recomputed by intersecting the moved top plane with the four side planes. The four vertices are recomputed by either intersecting the four side straights with the top face plane or by intersecting the new four top straights with the side planes. This new geometry is then copied to the old edge and vertex topology of the moved face. If the new geometry for a moved face and its edges interferes with any other part of the block, no check is performed and a self–intersecting body is the result, unless the option lop_repair_self_int is TRUE, in which case the problem will be fixed in the Repair Body Intersections Component.

The other local operations are quite similar. The operation for offsetting a body copies and offsets all of the surfaces in the body. Then, based on the old body's coedges, it intersects each offset with its neighbors to get new curves. The tweak operation simply replaces a surface instead of moving one.

LOP multiple solutions algorithm handles the general multiple solution problem. Points are calculated by intersecting the corresponding curve and surface. Curves are calculated by intersecting corresponding surfaces. Intersections that do not preserve the original topology are eliminated with the default settings. Any solutions that do not maintain the original convexity are also eliminated. Of the remaining solutions, one is chosen using a distance criteria that varies depending on the particular operation. The way this algorithm works will change depending on the setting for various options, including lop_repair_self_int. This, together with the curve and surface extension capability, enables the solution of local operation problems not involving unsupported changes in topology.

Blended Faces

Topic: *Local Operations

If LOP is supplied with faces that have been blended, the faces can be treated as blends or as regular faces.

If treating the faces as blends, the properties of the blend are preserved. LOP recalculates the blend geometry so that it remains tangent to its supporting surfaces. As with most local operations, the topology of the blend cannot change.

If treating the blend as a regular face, the edge between the blend and its supporting face is recalculated by intersecting the corresponding surfaces. An intersection must exist for the operation to work. The new edge is unlikely to be a tangent one in taper and move cases, although offsetting two adjacent tangent faces preserves tangency. In general, the new edge will bite into the blend surface.

LOP relies on standard blending (using the Blending Component)to perform the blending again. It cannot produce blends that are outside the scope of standard blending. In addition, LOP blending does not presently work on variable radius blends, blends on blends, vertex blends or entity–entity blends. All blends in a sequence need to be treated consistently.

In order for a face to be treated as a blend, it must have a blend attribute attached. Blending attaches suitable attributes to faces if the option add_bl_atts is switched on. Alternatively, the attributes may be added to existing surfaces (made by blending or otherwise) using the appropriate API call.

Limitations

Topic: *Local Operations

Intersections may yield several geometries. LOP chooses geometry that leaves the body locally valid and with the same edge convexity as the original body if possible. The body is only checked to be locally valid. If non–neighboring faces interpenetrate they are not detected, and a self–intersecting invalid body will result unless lop_repair_self_int is on, in which case these bodies will be healed.

If any intersection yields no geometry, local operations fails unless a suitable topology change to the model is supported. Topology changes are not permitted except for those described below.

Model surfaces and curves are extended by all local operations except tweak if they can provide good estimates for the required extension. All local operations are tolerant modeling friendly. This means that local operations behave sensibly when tolerant geometry is encountered. When lop_repair_self_int is on, LOP can make tolerant edges as necessary to successfully complete the operation.

LOP has the following limitations:

- The original and final body must be manifold and solid.
- No other topology changes than those documented are permitted.
- Tweak faces does not extend the geometry of the tool surfaces or their neighbors, unlike the other local operations, which do. (However, api_tweak_extend_faces can be used to extend surfaces and curves before a tweak operation.)
- Some options may be mutually exclusive. Be sure to check the reference documentation.

Geometry Changes

- pipe_spl_sur spline surfaces are converted to rb_blend_spl_sur spline surfaces in the supplied faces and their neighbors if the option lop_convert_pipe is set on, which is the default setting.

Topology Changes

- Mergeable edges and vertices on the supplied faces are merged out.
- Edges and vertices between faces that are tweaked to the same geometry are merged out.

- Two edge vertices on the supplied faces between edges on the same geometrical nonanalytic intersection curve are merged out by lop:tweak–faces if the option lop_merge_vertex is set on and if not presently merged elsewhere. This is the default setting.
- Vertices with more than three edges may split into two or more vertices during a local operation. Single inverted faces are created if needed to fill any gap in the body. If two or more inverted faces are needed, the operation fails.
- Self intersecting bodies that result from LOP can be healed by turning the option lop_repair_self_int on.
- Edges in multi–edge loops may degenerate exactly to a point and are removed.
- Isolated single edge loops may degenerate exactly to a point or be removed altogether. Their faces are deleted when appropriate.
- If an edge exactly degenerates to a point in a multi–edge loop, the edges on either side are detected and become fully or partially coincident. The loop is repaired by splitting it into valid, degenerate loops. Degenerate loops and their faces are deleted, when appropriate.
- When faces have been deleted, their shells and lumps are repartitioned and split if necessary.