Chapter 2.
Scheme Extensions

Topic: Ignore

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

rem:remove–faces

Scheme Extension: Local Operations

Action: Removes faces by growing adjacent faces to fill the gap.

Filename: `rem/rem_scm/rem_scm.cxx"`

APIs: None

Syntax:
```
(rem:remove–faces face–list [box–h box–l]
[acis–opts])
```

Arg Types:		
	face–list	face | (face ...)
	box–h	position
	box–l	position
	acis–opts	acis–options

Returns: body

Errors: Some of the following errors result in an ENTITY, which indicates where the error occurs, being highlighted. The ENTITY type follows the error message below.

At least face must be supplied or error;–
REM_NO_FACE ”no faces supplied”
Non–duplicate valid faces must be supplied from the same shell or error;–
REM_BAD_FACE ”face(s) supplied invalid, duplicate or from different shells”
Body must be manifold and solid or errors;–
REM_NON_MANIFOLD ”non–manifold edge detected” or
REM_FREE_EDGE ”free edge detected”
All faces in a shell or lump may be removed, so long as another shell remains, or error;–
REM_LAST_SHELL ”only shell in body would be lost”
Box must be valid if supplied, or error;–
REM_BAD_BOX ”invalid box supplied”
Adjacent faces must be able to combine to fill the gap left by the removed faces or error;–
REM_NO_SOLUTION ”gap cannot be filled”
Internal Algorithmic problems produce the error;–
REM_INTERNAL ”internal error”

Note that ENTITYs returned in the outcome standard_error_info object are highlighted.

Description: Removes an array of faces, growing the adjacent faces to fill the gap.

Curves and surfaces of edges and faces surrounding the removed faces, and which end in the user supplied box, are extended by a length approximately equal to twice the diagonal of the supplied box.

Similarly the intersections required to produce new edges and vertices, are limited by the size of the user supplied box.

Thus the user should supply a box likely to contain the changed faces. An overly large box will result in wasted computation. The default box is twice the body box of the original body.

Note that there may be no solution to the gap filling problem, or no solution using adjacent faces only. If the function detects these circumstances it issues an error and leaves the body unchanged.

Adjacent faces left infinite and with no loops will be deleted, e.g., the tops of cylindrical bosses having the cylindrical face removed.

If the faces to be removed are isolated from one another, they are removed in the order supplied.

The body shells and lumps are corrected at the end of the operation, should they have been split.

New shells of different solidity to the original shell are deleted, e.g., void shells made from a shell originally solid.

To have api_remove_faces heal gaps using adjacent and remote faces set option rem_use_rbi to TRUE, and api_remove_and_repair_body will be called.

Arguments

face–list identifies faces of a body to be moved.

box–h specifies one position defining a diagonal box for an intersection limit.

box–l specifies one position defining a diagonal box for an intersection limit.

acis–opts specifies options such as versioning and journaling.

Limitations: Body must be manifold and solid.

Some growing faces may shrink if necessary, but not so far as their boundaries. The healing process must be entirely within the faces adjacent to those removed.

No checks are made to see if the faces grown intersect with other non growing faces in the model. Checks that the growing faces intersect with one another is not rigorously done between unconnected regions of growing faces.

Example:

```
; rem:remove-faces
; Create a block and blend it, then remove the face
(define block1
    (solid:block (position -25 -25 -25)
    (position 25 25 25)))
;; block1
(define blend (solid:blend-edges
    (pick:edge (ray (position 0 0 0)
    (gvector 1 0 1))) 10))
;; blend
; OUTPUT Original
```

```
(define remove (rem:remove-faces
    (pick:face (ray (position 0 0 0)
    (gvector 1 0 1)))))
;; remove
; OUTPUT Result
```

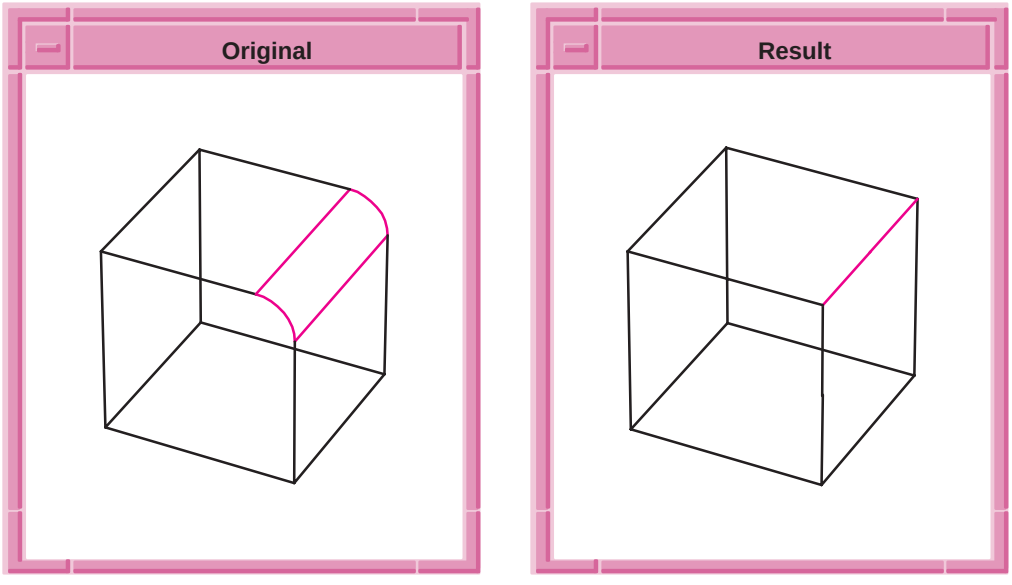

Figure 2-1. rem:remove–faces

```
; Example 2: Remove and extend
; Clear the previous part
(part:clear)
;; #t
(define surf1 (splsurf))
;; surf1
(define ctrlpoints1 (list
    (position –25 25 20)
    (position 0 25 50)
    (position 25 25 30)
    (position –25 0 10)
    (position 0 0 50)
    (position 25 0 25)
    (position –25 –25 30)
    (position 0 –25 50)
    (position 25 –25 20)))
;; ctrlpoints1
(splsurf:set–ctrlpt–list surf1 ctrlpoints1 3 3)
;; #[splsurf 40255540]
(splsurf:set–u–param surf1 2 0 0 0)
;; #[splsurf 40255540]
(splsurf:set–v–param surf1 2 0 0 0)
;; #[splsurf 40255540]
(define uknots (list 0 0 1 1))
;; uknots
(splsurf:set–u–knot–list surf1 uknots 4)
;; #[splsurf 40255540]
(define vknots (list 0 0 1 1))
;; vknots
(splsurf:set–v–knot–list surf1 vknots 4)
;; #[splsurf 40255540]
(define face1 (face:spline–ctrlpts surf1))
;; face1
(define block2
    (solid:block (position –25 –25 –25)
    (position 25 25 25)))
;; block2
(define tweak (lop:tweak–faces (car (pick:face
    (ray (position 0 0 0) (gvector 0 0 1))))
    face1 #f))
;; tweak
; OUTPUT Original
```

```
(define remove (rem:remove-faces (pick:face
    (ray (position 0 0 0) (gvector 1 0 0)))
    (position -500 -100 -100)
    (position 500 100 100)))
;; remove
; OUTPUT Result
```

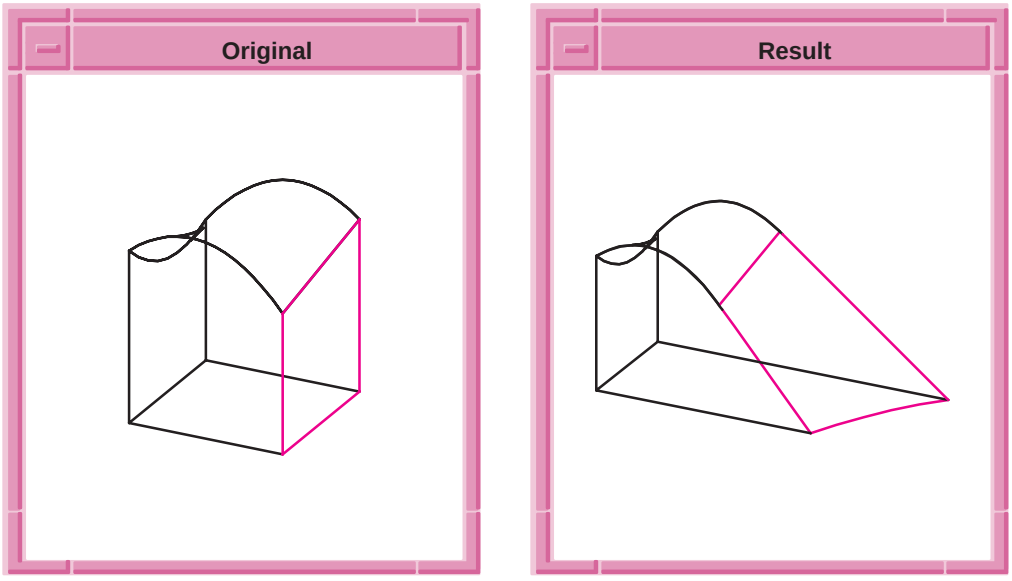

Figure 2-2. rem:remove–faces (Remove and Extend Faces)

```
; Example 3: Remove and split
; Clear the previous part
(part:clear)
;; #t
; Create a pyramid.
(define pyr1 (solid:pyramid 100 25 25 6 25))
;; pyr1
; Rotate pyr1.
(define rotate (entity:transform pyr1
    (transform:rotation (position 0 0 0)
    (gvector 0 1 0) 90)))
;; rotate
; Create a sphere.
(define sphere1 (solid:sphere (position 0 0 40) 25))
;; sphere1
; Unite the sphere with pyr1.
(define unite (solid:unite pyr1 sphere1))
;; unite
; OUTPUT United pyr1 and sphere1

; Create another pyramid.
(define pyr2 (solid:pyramid 100 25 25 6 25))
;; pyr2
; Rotate pyr2.
(define rotate2 (entity:transform pyr2
    (transform:rotation
    (position 0 0 0) (gvector 1 0 0) 90)))
;; rotate2
; Translate pyr2.
(define translate (entity:transform pyr2
    (transform:translation
    (gvector 0 0 80))))
;; translate
(define unite (solid:unite pyr1 pyr2))
;; unite
; OUTPUT United pyr1 and pyr2

(define remove (rem:remove–faces (pick:face
    (ray (position 0 0 40) (gvector 1 0 0)))))
;; remove
; OUTPUT Result
```

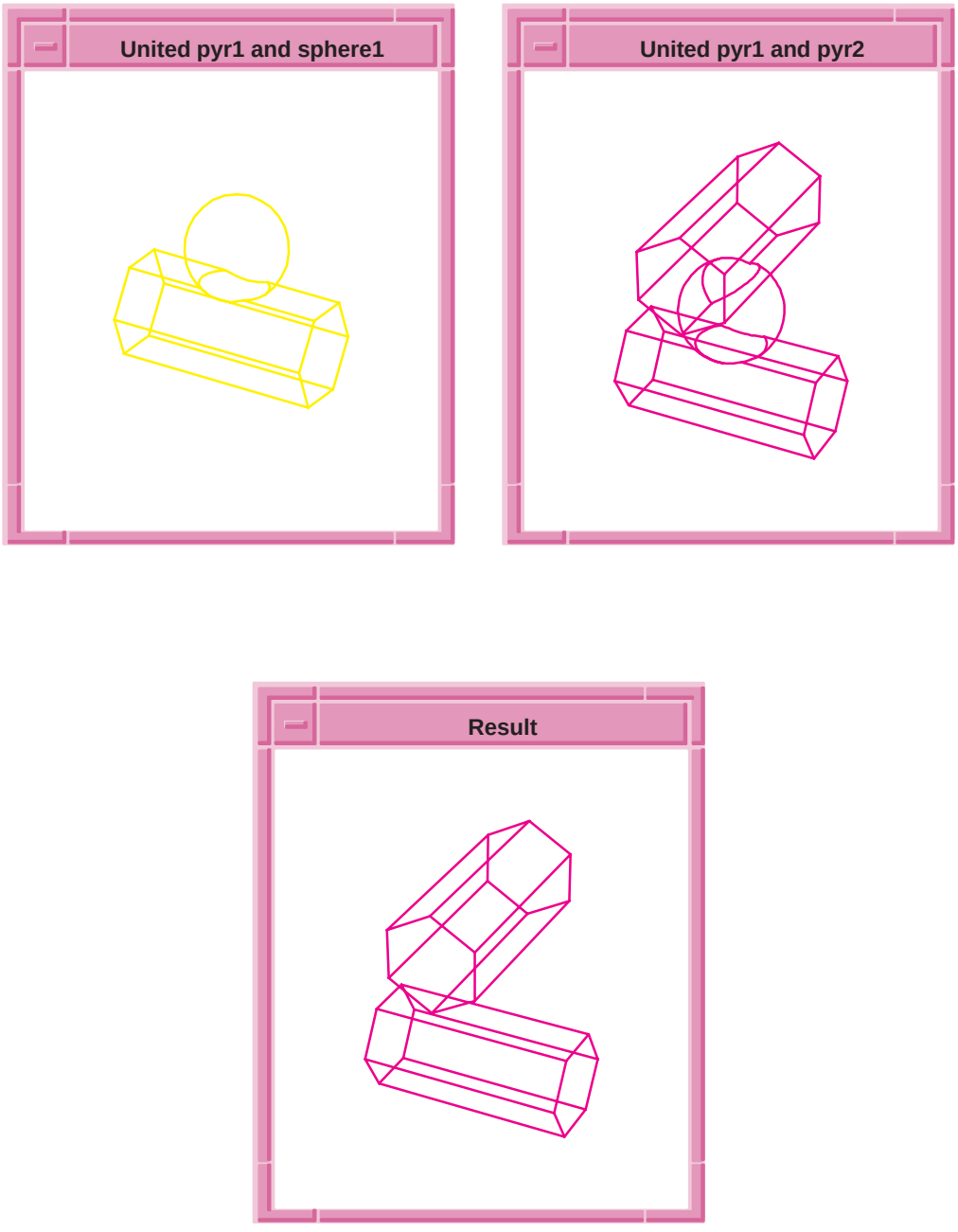

Figure 2-3. rem:remove–faces (Remove and Split Faces)

```
; Example 4: Remove and delete
; Clear the previous part
(part:clear)
;; #t
; Create a block.
(define block1
    (solid:block (position –25 –25 –25)
    (position 25 25 25)))
;; block1
(define cyl1
    (solid:cylinder (position 0 0 –5)
    (position 0 0 45) 20))
;; cyl1
(define unite (solid:unite block1 cyl1))
;; unite
; OUTPUT Original

(define remove (rem:remove–faces (pick:face (ray
    (position 0 0 30) (gvector 1 0 0)))))
;; remove
; OUTPUT Result
```

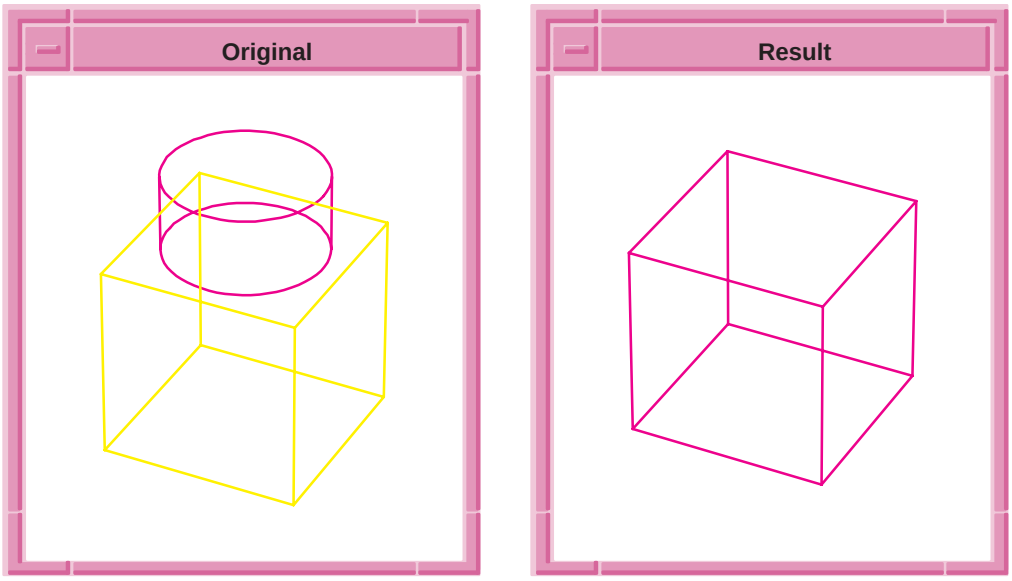

Figure 2-4. rem:remove–faces (Remove and Delete Faces)