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.

bool:chop

Scheme Extension: Booleans

Action: Simultaneously finds the intersection and difference between two bodies.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_boolean_chop_body

Syntax:
```
(bool:chop body1 body2 [keep–leftovers=#t]
    [keep–opt] [acis–opts])
```

Arg Types:		
	body1	body
	body2	body
	keep–leftovers	boolean
	keep–opt	string
	acis–opts	acis–options

Returns: entity ...

Errors: None

Description: This extension chops the blank body (body1) with the tool body (body2). The returned list contains the result of intersecting the two bodies, the result of subtracting body2 from body1, and (possibly) a body containing any “leftovers.” Either of the first two bodies returned may be empty. Leftovers can only exist if the tool body is incomplete. If there is a body containing leftovers, the keep–leftovers argument controls whether it is deleted or not: if keep–leftovers is #f, any leftovers are deleted and the returned list is always of length 2. If keep–leftovers is #t, the returned list is only of length 3 if there are some leftovers (i.e., an empty leftovers body is never returned).

The optional keep_opt specifies whether some or all input bodies should be preserved. Value ”keep_blank” preserves body1 and returns the result of the chop as a new body. ”keep_tool” preserves body2 . . . bodyn. ”keep_both” preserves all input bodies and returns the result of the chop as a new body.

body1 is a blank body.

body2 is a tool body.

keep–leftovers argument controls whether body containing leftovers deleted or not.

keep–opt specifies whether some or all input bodies should be preserved.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:chop
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 25 25)))
;; block1
; Create another solid block.
(define block2
    (solid:block (position –5 –20 –30)
    (position 30 15 35)))
;; block2
; OUTPUT Original

; Chop block1 with block2
(define chop–result (bool:chop block1 block2))
;; chop–result
; OUTPUT Result
```

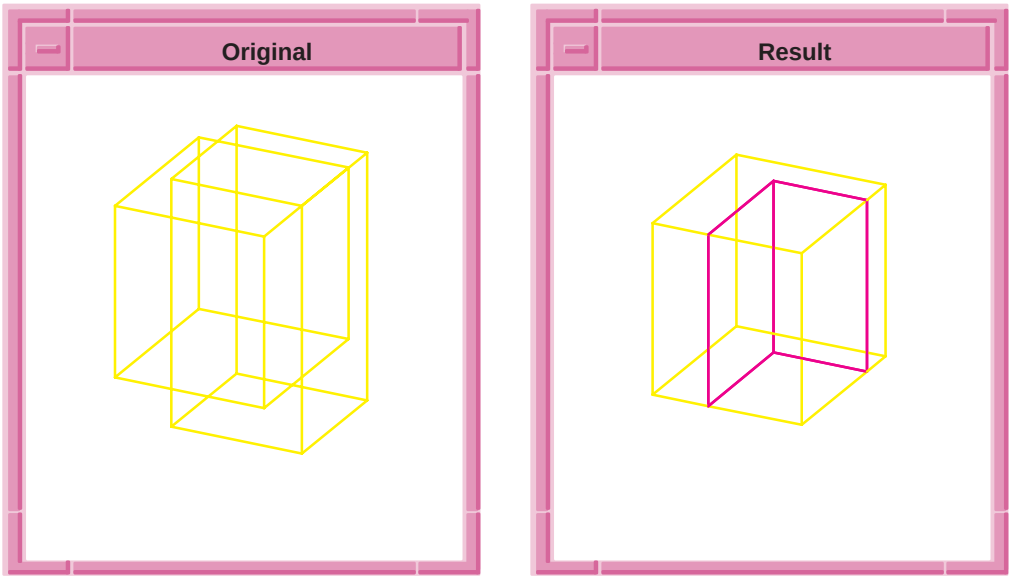

Figure 2-1. bool:chop

bool:clip

Scheme Extension: Booleans

Action: Creates a copy of a model clipped to two parallel planes.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_get_entity_box, api_slice_of_model

Syntax:
```
(bool:clip [body-list] [view=active]
    [hither=SPAresabs] [yon] [acis-opts])
```

Arg Types:		
	body–list	(body ...)
	view	view
	hither	real
	yon	real
	acis–opts	acis–options

Returns: entity ...

Errors: None

Description: This extension takes an entity list and a view, from which it extracts an eye position and a non–zero view vector (target eye). It also takes two doubles, hither and yon, which are planes orthogonal to the view vector. The model is clipped to the view planes hither and yon. It first copies the original, then trims those parts of the entity list that are outside the hither and yon planes. The trimmed copy is passed back.

As default values, it uses the active part and the active view, and sets hither to SPAresabs and yon large enough to display the whole model. The eye point is at a distance hither along the view vector from the hither plane and a distance yon along the view vector from the yon plane.

If the model itself does not need to be trimmed (i.e., it already is completely contained within the hither–yon slice), then no copy of the model is made. A pointer to the original model is passed back (instead of a pointer to a copy). Caution is advised, since the user could delete the original model thinking it is a copy.

body–list is a list of entities.

view is a view vector which defines the view.

hither and yon, are planes orthogonal to the view vector.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:clip
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Set the view eye position.
(view:set–eye (position 10 15 20))
;; #[position 100 –200 100]
; Make a copy of model clipped to the eye plane.
(define clip (bool:clip))
;; clip
; Delete original model.
(entity:delete block1)
;; ()
(define refresh (view:refresh))
;; refresh
; OUTPUT Clip1
; Notice the clipped copy.
```

```
; Set another eye position.
(view:set-eye(position 200 -300 400))
;; #[position 10 15 20]
(view:refresh)
;; #[view 1076825080]
; Note the changed eye view and the clipped corner.
; OUTPUT Clip2
```

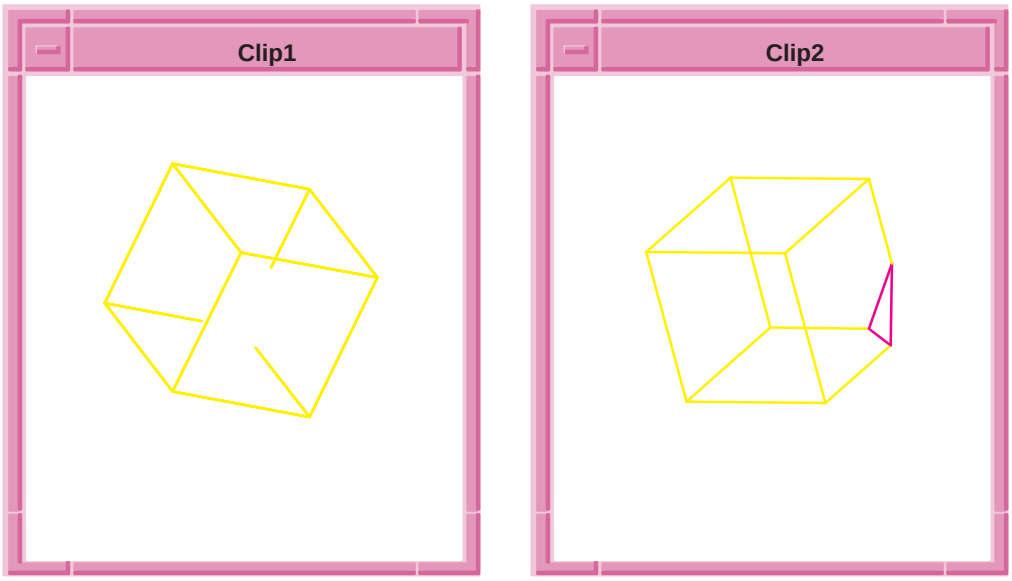

Figure 2-2. bool:clip

bool:glue–subtract

Scheme Extension: Booleans

Action: Performs a subtraction on two bodies that have a set of overlapping, coincident faces and no penetrating face–face intersections.

Filename: bool/bool_scm/glue_scm.cxx

APIs: None

Syntax:
```
(bool:glue-subtract body1 body2 faces1 faces2
    [glue-opts | "name-of-option" {value} ...]
    [keep-opt] [acis-opts])
```

Arg Types:	body1	body
	body2	body
	faces1	face | (face ...)
	faces2	face | (face ...)
	glue–opts	glue–options
	“name–of–option”	string
	value	boolean
	keep–opt	string
	acis–opts	acis–options

Returns: body

Errors: None

Description: This extension performs a Boolean subtract operation on two bodies in the special case where the intersection graph (see solid:slice) lies precisely on a set of overlapping, coincident faces, and neither body penetrates the faces of the other body.

This operation is designed to increase performance over the general Boolean operation, bool:subtract.

Two faces are coincident if the intersection of their interior point sets is non–empty and bounded by the edges of either face, and on this overlap, their surface geometries are coincident.

The glue operation performs only those face–face intersections deemed necessary by the list of pairwise coincident faces, faces1 and faces2, of body1 and body2 respectively. There is no verification that each pair of faces is, indeed, coincident, so it is *essential* that these lists are accurate and complete.

Refer to glue:options for information on the options designed to enhance performance. The options may be specified directly or in a Scheme object.

body1 and body2 are two bodies, where the intersection graph lies precisely on a set of overlapping, coincident faces, and neither body penetrates the faces of the other body.

faces1 and faces2 are the faces of body1 and body2 respectively.

glue–opts options designed to enhance performance.

name–of–option is the alternative for glue option. It can take the values ”patch_and_face_cover”, blank_patches_strict_cover” and ”non–trivial”

value is set for name–of–option as true or false.

keep–opt specifies whether some or all input bodies should be preserved.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; bool:glue–subtract
; Create entities to demonstrate command
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 25 25)))
;; block1
(define block2
    (solid:block (position –10 –20 15)
    (position 10 15 25)))
;; block2
; Retrieve the list of faces for each body
(define faces1 (entity:faces block1))
;; faces1
(define faces2 (entity:faces block2))
;; faces2
; Pick out the coincident faces on each body
(define f10 (list–ref faces1 0))
;; f10
(define f12 (list–ref faces1 2))
;; f12
(define f20 (list–ref faces2 0))
;; f20
(define f22 (list–ref faces2 2))
;; f22
; change highlight color for images
(env:set–highlight–color 1)
;; ()
; Highlight the first pair to see they are coincident
(entity:set–highlight (list f10 f20) #t)
;; (#[entity 4 1] #[entity 10 1])
; Highlight the second pair to see they are
; coincident
(entity:set–highlight (list f12 f22) #t)
;; (#[entity 6 1] #[entity 12 1])
; OUTPUT Original
```

```
; Now create a glue options object
(define glue-opts (glue:options "face_pair_cover" #t
    "blank_patches_strict_cover" #t
    "non_trivial" #t))
;; glue-opts
(define glue-result (bool:glue-subtract block1 block2
    (list f10 f12) (list f20 f22) glue-opts))
;; glue-result
; OUTPUT Result
```

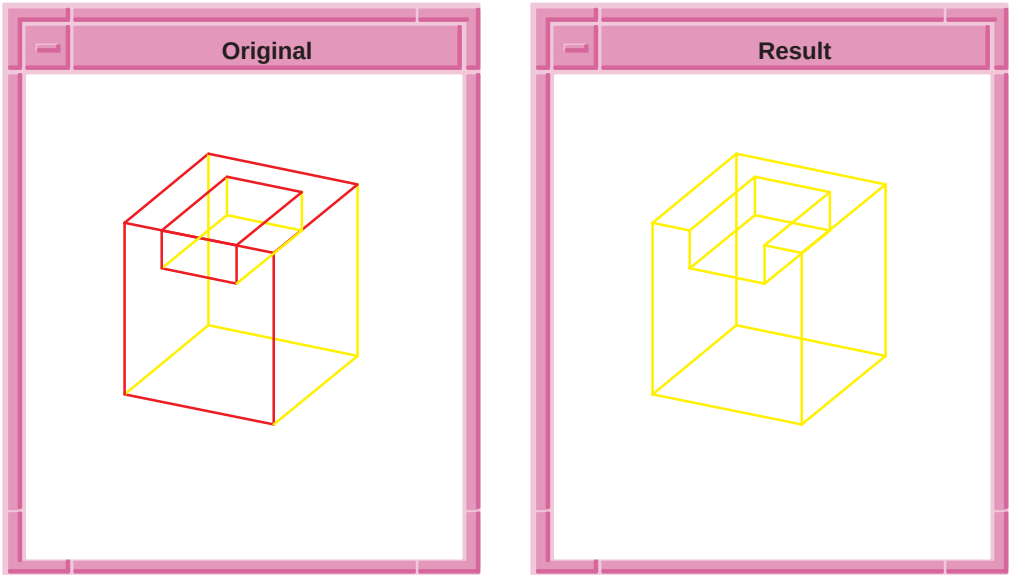

Figure 2-3. bool:glue–subtract

```
(part:clear)
;; #t
(define block1 (solid:block
    (position –20 –20 –20) (position 20 25 25)))
;; block1
(define block2 (solid:block
    (position 0 –20 –10) (position 20 0 10)))
;; block2
(define faces1 (entity:faces block1))
;; faces1
(define faces2 (entity:faces block2))
;; faces2
; Pick out the coincident faces
(define f12 (list–ref faces1 2))
;; f12
(define f15 (list–ref faces1 5))
;; f15
(define f22 (list–ref faces2 2))
;; f22
(define f25 (list–ref faces2 5))
;; f25
; change highlight color for images
(env:set–highlight–color 1)
;; ()
; Highlight each face pair to see they are coincident
(entity:set–highlight (list f12 f22) #t)
;; (#[entity 20 1] #[entity 26 1])
(entity:set–highlight (list f15 f25) #t)
;; (#[entity 23 1] #[entity 29 1])
; OUTPUT Original
```

```
; Define a glue options object for a subtract
; operation.
; face_pair_cover can be set to #t because f12 covers
; f22 and f15 covers f25.
; blank_patches_strict_cover can be set to #t because
; the patch consisting of f12 and f15 strictly covers
; the patch consisting of f22 and f25.  The boundary
; edges of the smaller patch (f22 and f25) do not
; touch the boundary edges of the larger patch
; (f12 and f15). non_trivial can be set to #t because
; block2 lies inside block1.
(define g (glue:options "face_pair_cover" #t
    "blank_patches_strict_cover" #t
    "non_trivial" #t))
;; g
; "blank_patches_strict_cover" 1 "non_trivial" 1]
(define glue-result (bool:glue-subtract
    block1 block2 (list f12 f15) (list f22 f25) g))
;; glue-result
(entity:check glue-result)
;checked:
;        1 lumps
;        1 shells
;        0 wires
;        10 faces
;        10 loops
;        48 coedges
;        24 edges
;        16 vertices
;; ()
; OUTOUT Result
```

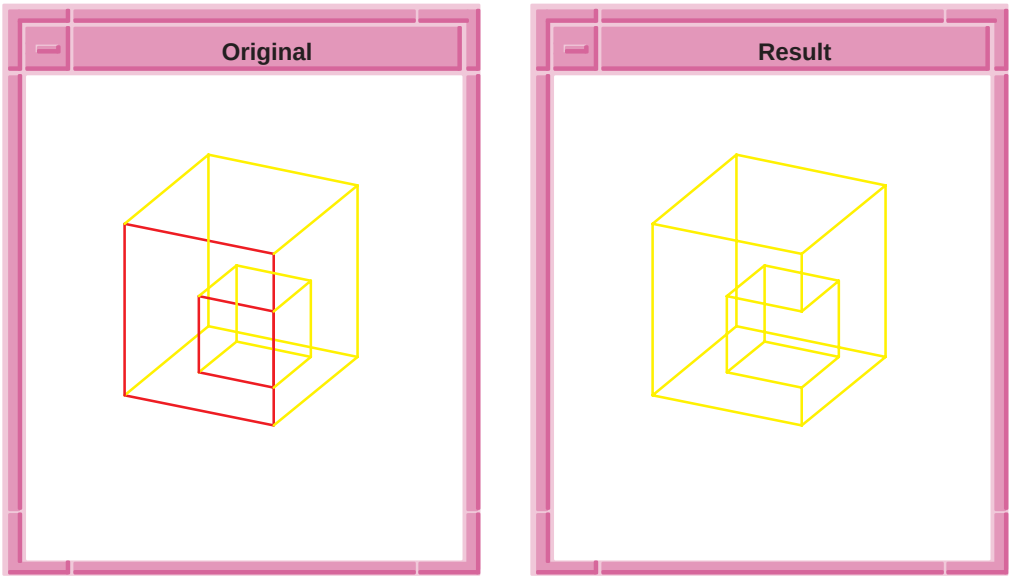

Figure 2-4. bool:glue–subtract – Example 2

bool:glue–subtract–inter–graph

Scheme Extension: Booleans

Action: Performs a subtraction on two bodies that have a set of overlapping, coincident faces and no penetrating face-face intersections.

Filename: bool/bool_scm/glue_scm.cxx

APIs: api_bool_make_intersection_graph

Syntax:
```
(bool:glue–subtract–inter–graph body1 body2 faces1
    faces2 [glue–opts | ”name–of–option” {value} ...]
    [acis–opts])
```

Arg Types:		
	body1	body
	body2	body
	faces1	face | (face ...)
	faces2	face | (face ...)
	glue–opts	glue–options
	”name–of–option”	string
	value	boolean
	acis–opts	acis–options

Returns: body

Errors: None

Description: This extension performs the first step of a Boolean subtract operation on two bodies in the special case where the intersection graph (see solid_slice) lies precisely on a set of overlapping, coincident faces, and neither body penetrates the faces of the other body.

This operation is designed to increase performance over the general computation of a Boolean operation intersection graph, solid:inter–graph.

Two faces are coincident if the intersection of their interior point sets is non-empty and bounded by the edges of either face, and on this overlap, their surface geometries are coincident.

The glue operation performs only those face-face intersections deemed necessary by the list of pairwise coincident faces, faces1 and faces2, of body1 and body2 respectively. There is no verification that each pair of faces is, indeed, coincident, so it is essential that these lists are accurate and complete.

Refer to glue:options for information on the options designed to enhance performance. The options may be specified directly or in a Scheme object.

body1 and body2 are two bodies, where the intersection graph lies precisely on a set of overlapping, coincident faces, and neither body penetrates the faces of the other body.

faces1 and faces2 are the faces of body1 and body2 respectively.

glue–opts options designed to enhance performance.

name–of–option is the alternative for glue option. It can take the values ”patch_and_face_cover”, blank_patches_strict_cover” and ”non–trivial”

value is set for name–of–option as true or false.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; bool:glue–subtract–inter–graph
; Create the entities
(define block1 (solid:block (position –20 –20 –20)
(position 20 25 25)))
;; block1
(define block2 (solid:block (position –10 –20 15)
(position 10 15 25)))
;; block2
(define faces1 (entity:faces block1))
;; faces1
(define faces2 (entity:faces block2))
;; faces2
(define f10 (list–ref faces1 0))
;; f10
(define f12 (list–ref faces1 2))
;; f12
(define f20 (list–ref faces2 0))
;; f20
(define f22 (list–ref faces2 2))
;; f22
(env:set–highlight–color 1)
;; ()
(entity:set–highlight (list f10 f20) #t)
;; (#[entity 3 1] #[entity 9 1])
(entity:set–highlight (list f12 f22) #t)
;; (#[entity 5 1] #[entity 11 1])
(define glue–opts (glue:options
"patch_and_face_cover" #t
"blank_patches_strict_cover" #t "non_trivial" #t))
;; glue–opts
(bool:start block1 block2)
;; #t
(bool:glue–subtract–inter–graph block1 block2 (list
f10 f12) (list f20 f22) glue–opts)
;;
(bool:complete "SUBTRACTION")
;; #[entity 1 1]
(entity:check block1 70)
;;
```

bool:glue–unite

Scheme Extension: Booleans

Action: Unites two bodies that have a set of overlapping, coincident faces and no penetrating face–face intersections.

Filename: bool/bool_scm/glue_scm.cxx

APIs: api_boolean_glue

Syntax:
```
(bool:glue–unite body1 body2 faces1 faces2
    [glue–opts | ”name–of–option” {value} ...]
    [keep–opt] [acis–opts])
```

Arg Types:

body1	body
body2	body
faces1	face | (face ...)
faces2	face | (face ...)
glue–opts	glue–options
”name–of–option”	string
value	boolean
keep–opt	string
acis–opts	acis–options

Returns: body

Errors: None

Description: This extension performs a Boolean unite operation on two bodies in the special case where the intersection graph (see solid:slice) lies precisely on a set of overlapping, coincident faces, and neither body penetrates the faces of the other body. This operation is designed to increase performance over the general Boolean operation, bool:unite.

Intuitively, this operation (when non–trivial) can be viewed as “gluing” two solid objects together by pasting their coincident faces.

Two faces are coincident if the intersection of their interior point sets is non–empty and bounded by the edges of either face, and on this overlap, their surface geometries are coincident.

The glue operation performs only those face–face intersections deemed necessary by the lists of pairwise coincident faces, faces1 and faces2, of body1 and body2 respectively. It is essential that the lists are accurate and complete because there is no verification that each pair of faces is coincident.

Refer to glue:options for information on the options designed to enhance performance. The options may be specified directly or in a Scheme object.

body1 and body2 are two bodies, where the intersection graph lies precisely on a set of overlapping, coincident faces, and neither body penetrates the faces of the other body.

faces1 and faces2 are the faces of body1 and body2 respectively.

glue–opts options designed to enhance performance.

name–of–option is the name of the glue option.

value is the value of the option.

keep–opt specifies whether some or all input bodies should be preserved.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; bool:glue–unite
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 25 25)))
;; block1
; Create a second solid block to glue to the first.
(define block2
    (solid:block (position –15 –30 –15)
    (position 15 –20 20)))
;; block2
; Retrieve the list of faces for each body
(define faces1 (entity:faces block1))
;; faces1
(define faces2 (entity:faces block2))
;; faces2
; Pick out the coincident faces on each body
(define f12 (list–ref faces1 2))
;; f12
(define f24 (list–ref faces2 4))
;; f24
(env:set–highlight–color 1)
;; ()
; Highlight the face pair to see they are coincident
(entity:set–highlight (list f12 f24) #t)
;; (#[entity 6 1] #[entity 14 1])
; OUTPUT Original
```

```
; Now create a glue options object
(define glue-opts (glue:options
    "face_pair_cover" #t "blank_patches_strict_cover"
    #t "non_trivial" #t))
;; glue-opts
; Perform a glue unite
(define glue-result (bool:glue-unite block1 block2
    (list f12) (list f24) glue-opts))
;; glue-result
```

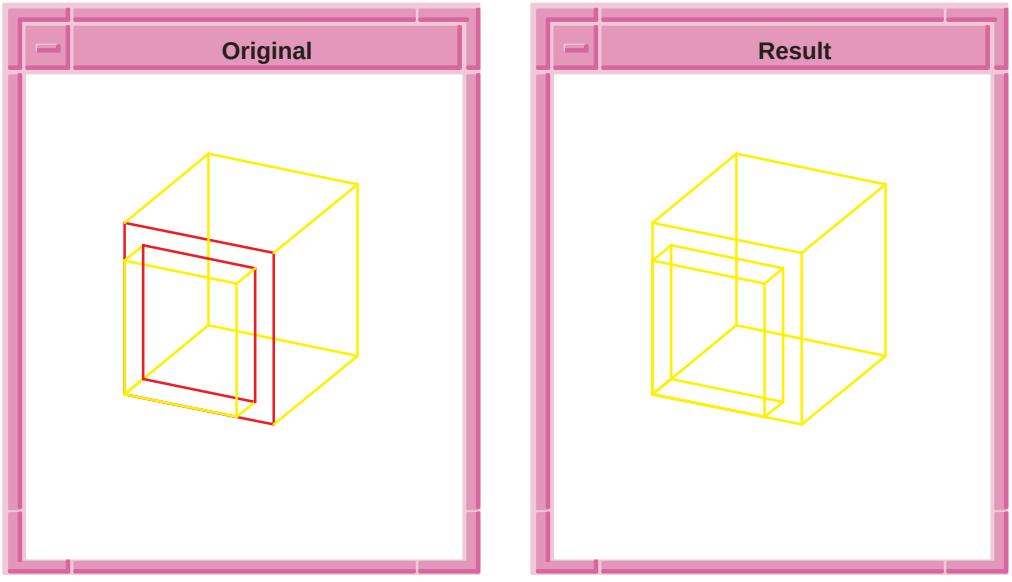

Figure 2-5. bool:glue–unite

```
; Example 2
(part:clear)
;; #t
(define block1 (solid:block
    (position –20 –20 –20) (position 20 25 25)))
;; block1
(define block2 (solid:block
    (position –10 –10 25) (position 30 10 35)))
;; block2
(define faces1 (entity:faces block1))
;; faces1
(define faces2 (entity:faces block2))
;; faces2
; Pick out the coincident faces
(define f10 (list–ref faces1 0))
;; f10
(define f21 (list–ref faces2 1))
;; f21
; Highlight the face pair to see they are coincident
(entity:set–highlight (list f10 f21) #t)
;; (#[entity 18 1] #[entity 25 1])
; OUTPUT Original
```

```
; Define a glue options object for a unite operation.
; face_pair_cover cannot be set to #t because f10
; does not cover f21 and f21 does not cover f10.
; blank_patches_strict_cover cannot be set to #t
; because face_pair_cover is not set to #t.
; non_trivial can be set to #t because the bodies
; lie outside one another.
(define g (glue:options "non_trivial" #t))
;; g
; g => #[Glue_Options  "face_pair_cover" -1
; "blank_patches_strict_cover" -1 "non_trivial" 1]
(define glue-result (bool:glue-unite
    block1 block2 (list f10) (list f21) g))
;; glue-result
(entity:check glue-result)
; checked:
;        1 lumps
;        1 shells
;        0 wires
;        12 faces
;        12 loops
;        56 coedges
;        28 edges
;        18 vertices
;; ()
; OUTPUT Result
```

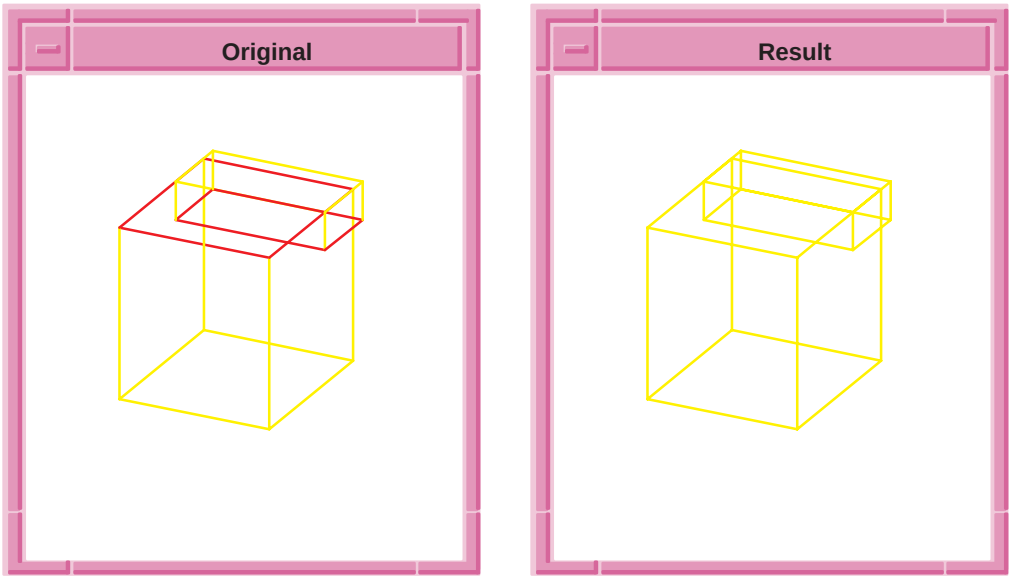

Figure 2-6. bool:glue–unite – Example 2

bool:glue–unite–inter–graph

Scheme Extension: Booleans

Action: Unites two bodies that have a set of overlapping, coincident faces and no penetrating face-face intersections.

Filename: bool/bool_scm/glue_scm.cxx

APIs: api_bool_make_intersection_graph

Syntax:
```
(bool:glue–unite–inter–graph body1 body2 faces1
    faces2 [glue–opts | ”name–of–option” {value} ...]
    [acis–opts])
```

Arg Types:		
	body1	body
	body2	body
	faces1	face | (face ...)
	faces2	face | (face ...)
	glue–opts	glue–options
	”name–of–option”	string
	value	boolean
	acis–opts	acis–options

Returns: body

Errors: None

Description: This extension performs the 1st step of the Boolean unite operation on two bodies in the special case where the intersection graph (see solid:slice) lies precisely on a set of overlapping, coincident faces, and neither body penetrates the faces of the other body.

This operation is designed to increase performance over the general computation of a Boolean operation intersection graph, solid:inter–graph.

Intuitively, this operation (when non-trivial) can be viewed as ”gluing” two solid objects together by pasting their coincident faces.

Two faces are coincident if the intersection of their interior point sets is non-empty and bounded by the edges of either face, and on this overlap, their surface geometries are coincident.

The glue operation performs only those face-face intersections deemed necessary by the lists of pairwise coincident faces, faces1 and faces2, of body1 and body2 respectively. It is essential that the lists are accurate and complete because there is no verification that each pair of faces is coincident.

Refer to glue:options for information on the options designed to enhance performance. The options may be specified directly or in a Scheme object.

body1 and body2 are two bodies, where the intersection graph lies precisely on a set of overlapping, coincident faces, and neither body penetrates the faces of the other body.

faces1 and faces2 are the faces of body1 and body2 respectively.

glue–opts options designed to enhance performance.

name–of–option is the name of the glue option.

value is the value of the option.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
;bool:glue–unite–inter–graph
; Create the entities
(define block1 (solid:block (position –10 –10 25)
(position 35 10 35)))
;; block1
(define block2 (solid:block (position 20 –10 15)
(position 35 10 25)))
;; block2
(define faces1 (entity:faces block1))
;; faces1
(define faces2 (entity:faces block2))
;; faces2
(define f11 (list–ref faces1 1))
;; f11
(define f20 (list–ref faces2 0))
;; f20
(env:set–highlight–color 1)
;; ()
(entity:set–highlight (list f11 f20) #t)
;; (#[entity 18 1] #[entity 23 1])
(define g (glue:options ”non_trivial” #t))
;; g
(bool:start block1 block2)
;; #t
(bool:glue–unite–inter–graph block1 block2 (list f11)
(list f20) g)
;;
(bool:complete ”UNION”)
;; #[entity 15 1]
(entity:check block1 70)
;;
```

bool:intersect

Scheme Extension: Booleans

Action: Intersects two or more bodies.

Filename: bool/bool_scm/bool_scm.cxx

APIs: None

Syntax: (**bool:intersect** body1 ... bodyn [keep–opt] [acis–opts])

Arg Types:	body1	body
	bodyn	body
	keep–opt	string
	acis–opts	acis–options

Returns: body

Errors: None

Description: This extension intersects two or more bodies. The resulting body is the space that is inside or on all of the input bodies. By default, the result of the intersection is returned as body1 and all other bodies are deleted.

The optional keep_opt specifies whether some or all input bodies should be preserved. Value "keep_blank" preserves body1 and returns the result of the intersection as a new body. "keep_tool" preserves body2 . . . bodyn. "keep_both" preserves all input bodies and returns the result of the intersection as a new body.

This extension performs a regularized intersection. Any faces, edges, and/or vertices not needed to support the topology of the intersection are removed before returning the resulting body.

body1 is the first body of intersection.

bodyn is the nth body of intersection.

keep–opt specifies whether some or all input bodies should be preserved.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:intersect
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 20 20 20)))
;; block1
; Create a solid sphere.
(define sphere1 (solid:sphere (position 0 0 0) 20))
;; sphere1
; OUTPUT Original

; Intersect the block with the sphere.
(define intersect (bool:intersect block1 sphere1))
;; intersect
; OUTPUT Result
```

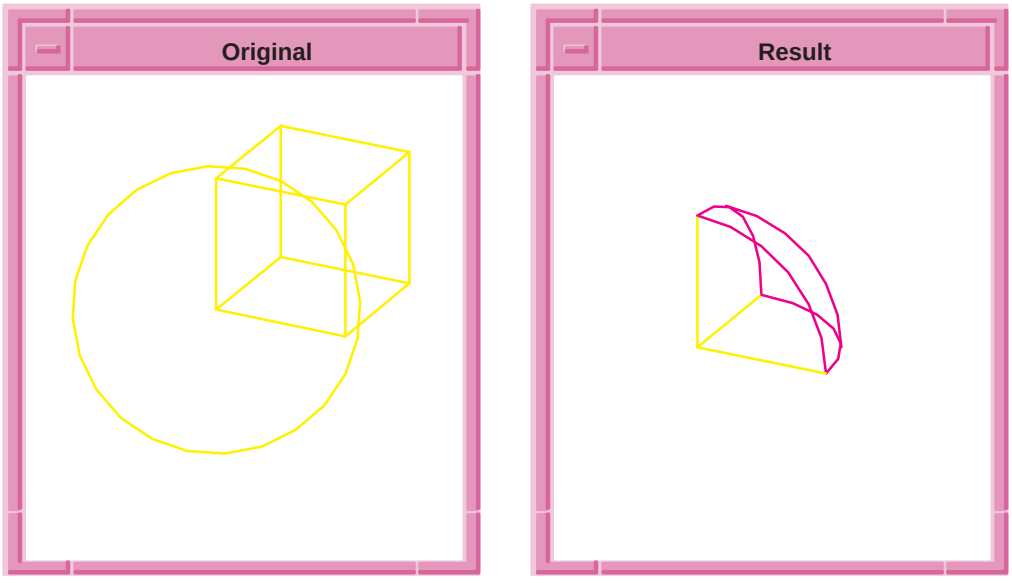

Figure 2-7. bool:intersect

bool:join–edges

Scheme Extension: Booleans

Action: Joins multiple edges into a single edge.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_get_owner, api_join_edges

Syntax: `(bool:join–edges edgelist [join] [acis–opts])`

Arg Types:		
	edgelist	edge | (edge...)
	join	boolean
	acis–opts	acis–options

Returns: edge

Errors: None

Description: This extension takes a list of edges in a body and attempts to join them into a single edge even if the underlying geometry of the edges is different. Edges need to be end to end (they can be individually flipped to make it so). Edges must be tangent continuous, only two edges should meet at each interior vertex, and edges can have any valid ACIS geometry.

edgelist is a list of edges in a body.

join is a boolean variable to join edges.

acis–opts contains parameters for versioning and journaling.

Limitations: Edges can not be branched.

Example:

```
; bool:join–edges
; Create/build geometry to demonstrate command.
; Create 3 edges.
(define edge1 (edge:linear (position 0 0 0)
    (position 50 0 0) ))
(define edge1 (edge:linear (position 0 0 0)
    (position 50 0 0) ))
;; edge1
(define edge2
    (edge:circular (position 50 25 0) 25 –90 90))
;; edge2
; Make the end points of edge1 and edge2 meet.
(define reverse (edge:reverse edge2))
;; reverse
; Create an edge that begins at the start of edge2.
(define edge3 (edge:linear (position 50 50 0)
    (position 0 50 0)))
;; edge3
; Create a wire body that includes all edges.
(define all (wire–body (list edge1 edge2 edge3)))
;; all
; Find the edges of the wire body.
(define getedges (entity:edges all))
;; getedges
; Join the edges into a single edge.
(define join (bool:join–edges getedges))
;; join
```

bool:merge

Scheme Extension: Booleans

Action: Combines faces and edges of equivalent geometry.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_clean_entity

Syntax: (**bool:merge** entity [acis–opts])

Arg Types: entity — entity
acis–opts — acis–options

Returns: entity

Errors: None

Description: Checks the geometric definitions of a body, face, or edge and merges adjacent faces when they have an equivalent definition. Faces are merged if they are adjacent or overlapping, have equivalent geometry, and have the same sense. Faces are merged by removing edges. Edges may be removed if they have the same surface on either side of the edge. A vertex may be removed if all of its associated edges are removed, if it separates edges which have the same geometrical support, and if it is not needed to define an end point of another edge.

entity is an input entity.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:merge
; Create a planar face.
(define plane1
    (face:plane (position –20 –40 0) 40 60))
;; plane1
; Set the color of plane1.
(entity:set–color plane1 2)
;; ()
; Create another planar face.
(define plane2
    (face:plane (position –20 –40 0) 60 40))
;; plane2
; Set the color of plane2.
(entity:set–color plane2 3)
;; ()
; OUTPUT Original

; Define a surface from the 2 planes.
(define one–surf
    (bool:unite (sheet:face plane1)
    (sheet:face plane2)))
;; one–surf
; Use merge to remove unnecessary faces/edges.
(define merge (bool:merge one–surf))
;; merge
; OUTPUT Result
```

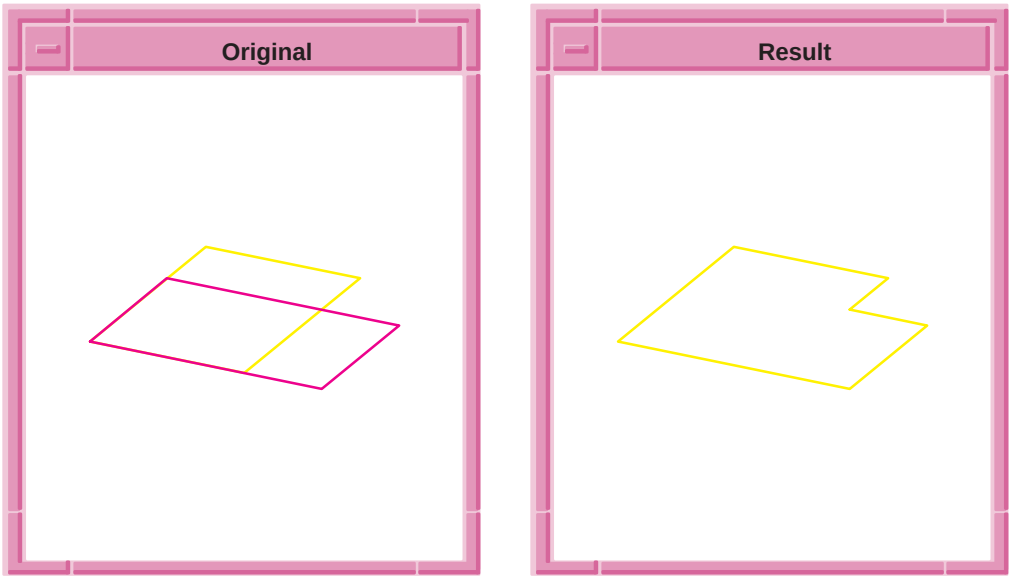

Figure 2-8. bool:merge

bool:merge–faces

Scheme Extension: Booleans

Action: Combines specific faces on a body.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_merge_faces

Syntax: `(bool:merge-faces body [type=plane] [acis-opts])`

Arg Types:		
	body	body
	type	string
	acis–opts	acis–options

Returns: entity

Errors: None

Description: Merge faces selectively by geometry type. The valid types are: plane, cone, cylinder, sphere, torus, mesh.

Checks the geometric definitions of a face and merges adjacent faces when they have an equivalent definition. Faces are merged if they are adjacent or overlapping, have equivalent geometry, and have the same sense. Faces are merged by removing edges. Edges may be removed if they have the same surface on either side of the edge. A vertex may be removed if all of its associated edges are removed, if it separates edges which have the same geometrical support, and if it is not needed to define an end point of another edge.

body is an input body.

type can be plane, cone, cylinder, sphere, torus, mesh.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:merge–faces
; Create a solid block.
(define block1
    (solid:block (position –30 –30 –30)
    (position 10 10 10)))
;; block1
; Create another solid block.
(define block2
    (solid:block (position –10 –10 –30)
    (position 30 30 10)))
;; block2
(define zoom (zoom–all))
;; zoom
; OUTPUT Two Blocks

; Imprint the two bodies.
(define imprint (solid:imprint block1 block2))
;; imprint
; Remove one entity to see the imprint lines.
(define delete (entity:delete block2))
;; delete
; OUTPUT Imprint

; Use merge to remove unnecessary edges.
(define merge (bool:merge–faces block1 ”plane”))
;; merge
; OUTPUT Merged Faces
```

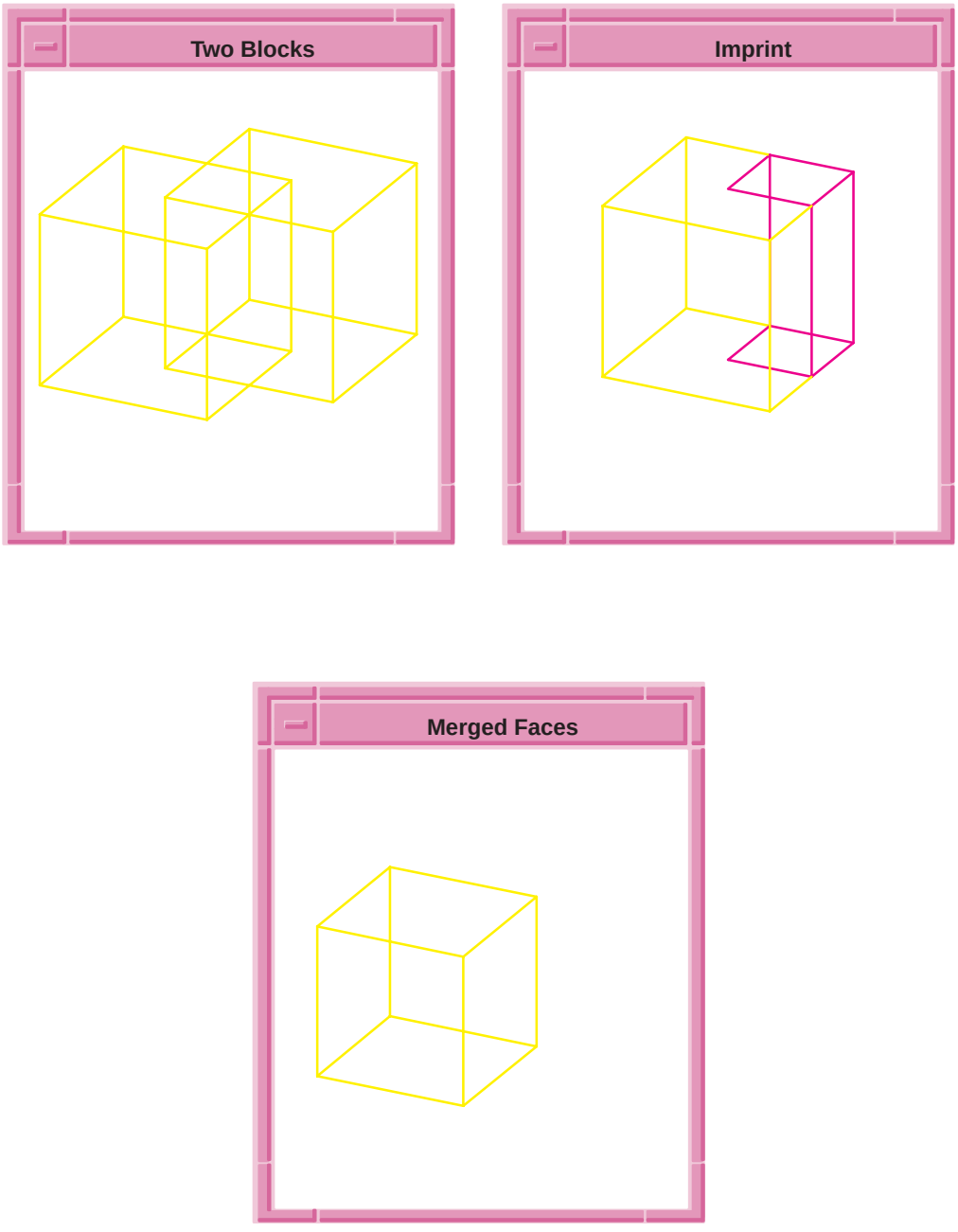

Figure 2-9. bool:merge–faces

bool:nonreg–chop

Scheme Extension: Booleans

Action: Simultaneously finds the nonregularized intersection and difference between two bodies.

Filename: bool/bool_scm/bool_scm.cxx

APIs: None

Syntax:
```
(bool:nonreg–chop body1 body2 [keep–leftovers=#t]
    [keep–opt] [acis–opts])
```

Arg Types:		
	body1	body
	body2	body
	keep–leftovers	boolean
	keep–opt	string
	acis–opts	acis–options

Returns: boolean

Errors: None

Description: This extension chops the blank body (body1) with the tool body (body2). The returned list contains the result of intersecting the two bodies, the result of subtracting body2 from body1, and (possibly) a body containing any “leftovers.” None of these resulting bodies are regularized. Either of the first two bodies returned may be empty. Leftovers can only exist if the tool body is incomplete. If there is a body containing leftovers, the keep–leftovers argument controls whether it is deleted or not. If keep–leftovers is #f, any leftovers are deleted and the returned list is always of length 2. If keep–leftovers is #t, the returned list is only of length 3 if there are some leftovers (i.e., an empty leftovers body is never returned).

The optional keep–opt specifies whether one or both input bodies should be preserved. Value “keep_blank” preserves body1 and returns the result of the intersection as a new body. “keep_tool” preserves body2. “keep_both” preserves both input bodies and returns the result of the chop as new bodies.

body1 is a blank body.

body2 is a tool body.

keep–leftovers argument controls whether body containing leftovers is deleted or not.

keep–opt specifies whether one or both input bodies should be preserved.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:nonreg–chop
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 25 25)))
;; block1
; Create another solid block.
(define block2
    (solid:block (position –5 –20 –30)
    (position 30 15 35)))
;; block2
; OUTPUT Original

; Imprint the faces of the two blocks.
(define chop (bool:nonreg–chop block1 block2))
;; chop
; (#[entity 2 1] #[entity 4 1])
(define erase (entity:erase block1))
;; erase
(view:refresh)
;; #[view 1076235512]
; OUTPUT Result
```

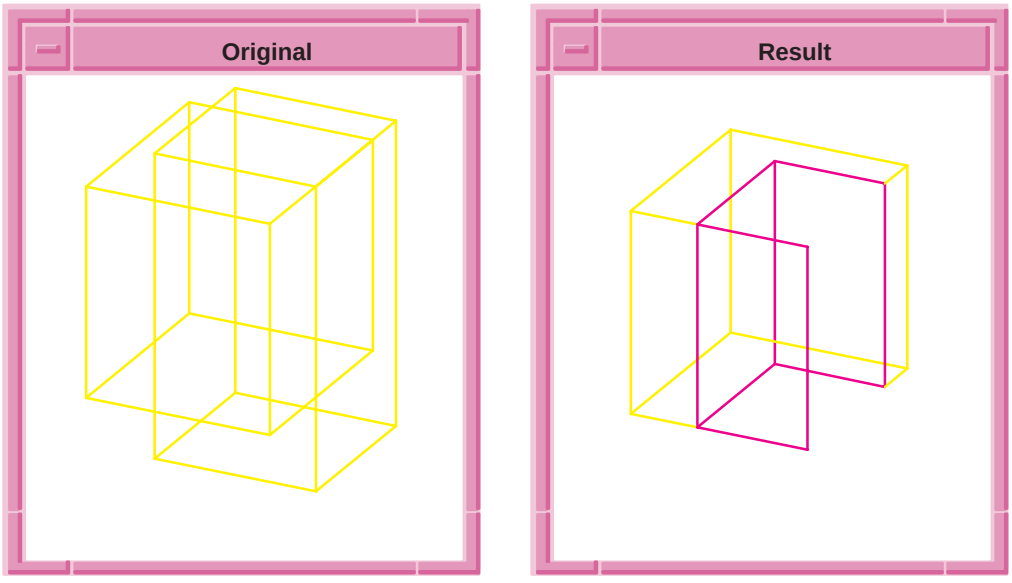

Figure 2-10. bool:nonreg–chop

bool:nonreg–intersect

Scheme Extension: Booleans

Action: Intersects two or more nonregularized bodies.

Filename: bool/bool_scm/bool_scm.cxx

APIs: None

Syntax: `(bool:nonreg–intersect body1 ... bodyn [keep–opt] [acis–opts])`

Arg Types:		
	body1	body
	bodyn	body
	keep–opt	string
	acis–opts	acis–options

Returns: body

Errors: None

Description: This extension intersects two or more bodies. By default, the result of the intersection is returned as body1 and all other bodies are deleted. The input body is trimmed to be only those parts inside or on all of the input bodies.

The optional keep–opt specifies whether some or all input bodies should be preserved. Value “keep_blank” preserves body1 and returns the result of the intersection as a new body. “keep_tool” preserves body2 . . . bodyn. “keep_both” preserves all input bodies and returns the result of the intersection as a new body.

After the nonregularized intersection, the returned body may still have faces, edges, and/or vertices that are not needed to support the topology. Single–sided faces that become double–sided–both–inside faces, internal faces, and coincident regions are retained as part of the body. This extension does not merge edges or vertices at the end of the Boolean.

body1 is the first body of intersection.

bodyn is the nth body of intersection.

keep–opt specifies whether some or all input bodies should be preserved.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:nonreg–intersect
; Create a solid block.
(define block1
    (solid:block (position –30 –20 0)
    (position 0 10 30)))
;; block1
; Set color for block1.
(entity:set–color block1 2)
;; ()
; Create another solid block.
(define block2
    (solid:block (position 0 –20 0)
    (position 30 10 30)))
;; block2
; Set color for block2.
(entity:set–color block2 3)
;; ()
; OUTPUT Original

; Nonregularize intersect blocks 1 with 2.
(define intersect
    (bool:nonreg–intersect block1 block2))
;; intersect
; #[entity 2 1]
; OUTPUT Result
```

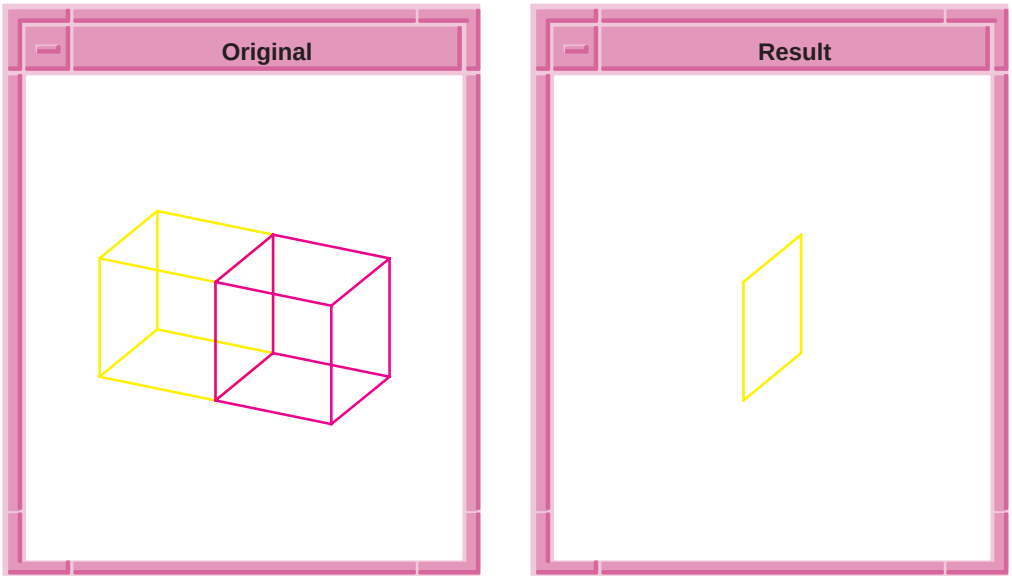

Figure 2-11. bool:nonreg–intersect

bool:nonreg–subtract

Scheme Extension: Booleans

Action: Subtracts one or more nonregularized bodies from a body.

Filename: bool/bool_scm/bool_scm.cxx

APIs: None

Syntax:
```
(bool:nonreg–subtract body1 ... bodyn [keep–opt]
    [acis–opts])
```

Arg Types:		
	body1	body
	bodyn	body
	keep–opt	string
	acis–opts	acis–options

Returns: body

Errors: None

Description: This extension subtracts one or more bodies from the first body, and it returns the result as body1. The other bodies are deleted.

The optional keep–opt specifies whether some or all input bodies should be preserved. Value ”keep_blank” preserves body1 and returns the result of the subtraction as a new body. ”keep_tool” preserves body2 ... bodyn. ”keep_both” preserves all input bodies and returns the result of the subtraction as a new body.

This extension performs a nonregularized subtract. The resulting body may have faces, edges, and/or vertices that are not needed to support the topology. These are not removed before returning the resulting body.

Single–sided faces that become double–sided–both–inside faces, internal faces, and coincident regions are retained as part of the body. This extension does not merge edges or vertices at the end of the Boolean.

The following example creates two solid blocks that share three common face regions. The boolean nonregularized subtract removes the smaller cube from the larger cube, leaving a 3D void; however, nonregularized Boolean operations retain coincident regions. The three coincident faces are retained as well. (A regularized subtract would remove these three faces.)

body1 is the first body of subtraction.

bodyn is the nth body of subtraction.

keep–opt specifies whether some or all input bodies should be preserved.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:nonreg–subtract
; Create a solid block.
(define block1 (solid:block
    (position –20 –20 –20) (position 0 0 0)))
;; block1
; Set color for block1.
(entity:set–color block1 2)
;; ()
; Create another solid block.
(define block2 (solid:block
    (position –20 –20 –20) (position 20 20 20)))
;; block2
; Set color for block2.
(entity:set–color block2 3)
;; ()
; OUTPUT Original
```

```
; Nonregularize subtract blocks 1 from 2.
(define subtract
    (bool:nonreg-subtract block2 block1))
;; subtract
; #[entity 3 1]
; OUTPUT Result
```

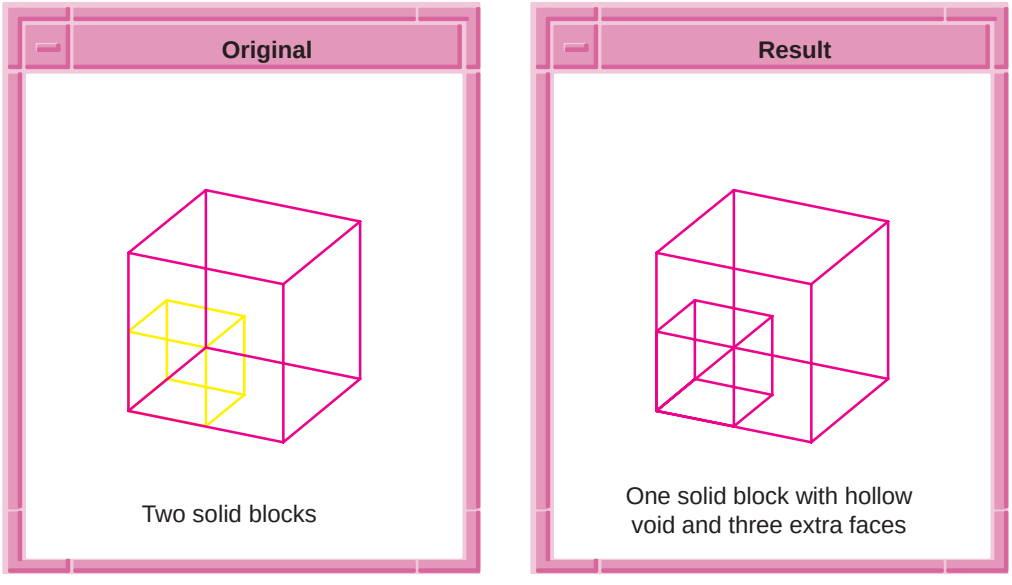

Figure 2-12. bool:nonreg–subtract

bool:nonreg–unite

Scheme Extension: Booleans

Action: Unites two or more nonregularized bodies.

Filename: bool/bool_scm/bool_scm.cxx

APIs: None

Syntax:
```
(bool:nonreg-unite body1 ... bodyn [keep-opt]
    [acis-opts])
```

Arg Types:		
	body1	body
	bodyn	body
	keep–opt	string
	acis–opts	acis–options

Returns: body

Errors: None

Description: This extension unites two or more bodies. By default, the result of the unite is returned as body1 and the remaining bodies are deleted.

The optional keep–opt specifies whether some or all input bodies should be preserved. Value ”keep_blank” preserves body1 and returns the result of the unite as a new body. ”keep_tool” preserves body2 ... bodyn. ”keep_both” preserves all input bodies and returns the result of the unite as a new body.

This extension performs a nonregularized unite. The resulting body may have faces, edges, and/or vertices not needed to support the topology. These are not removed before returning the resulting body.

Single–sided faces that become double–sided, both–inside faces, internal faces, and coincident regions are retained as part of the body. This extension does not merge edges or vertices at the end of the Boolean.

body1 is the first body of unite.

bodyn is the nth body of unite.

keep–opt specifies whether some or all input bodies should be preserved.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:nonreg–unite
; Create a solid block.
(define block1
    (solid:block (position –30 –20 0)
    (position 0 10 30)))
;; block1
; Set color for block1.
(entity:set–color block1 2)
;; ()
; Create another solid block.
(define block2
    (solid:block (position 0 –20 0)
    (position 30 10 30)))
;; block2
; Set color for block2.
(entity:set–color block2 3)
;; ()
; OUTPUT Original
```

```
; Nonregularize unite blocks 1 and 2
(define unite (bool:nonreg–unite block1 block2))
;; unite
; #[entity 2 1]
; OUTPUT Result
```

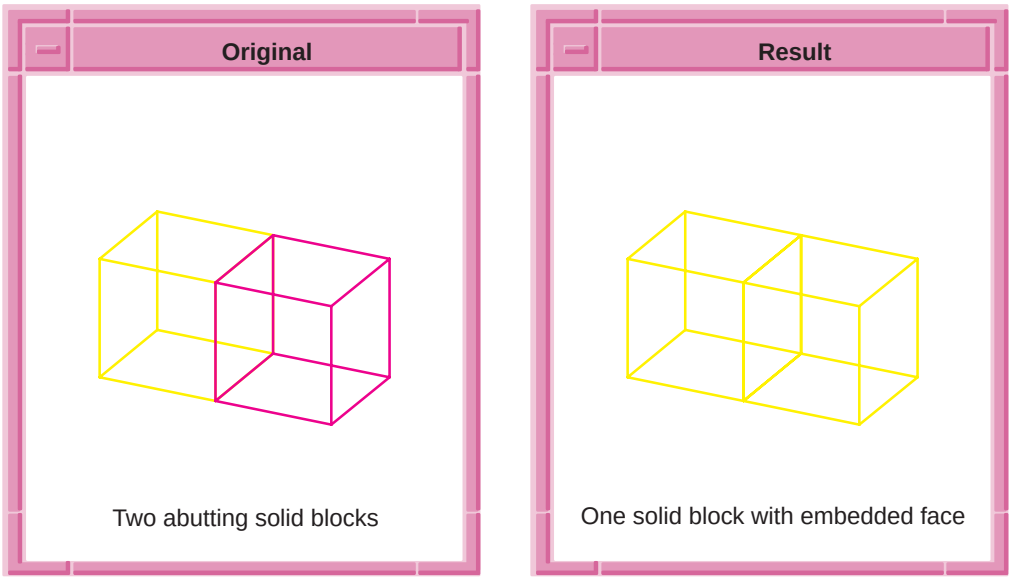

Figure 2-13. bool:nonreg–unite

bool:regularise

Scheme Extension: Booleans

Action: Regularizes an entity.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_regularise_entity

Syntax: (**bool:regularise** ent [acis–opts])

Arg Types:	entity	entity
	acis–opts	acis–options

Returns: entity

Errors: None

Description: This extension removes single–sided faces that become double–sided–both–inside faces, internal faces and coincident regions. This extension also merges edges and vertices.

entity is an input entity.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:regularise
; Create a solid block.
(define block1
    (solid:block (position –30 –20 0)
    (position 0 10 30)))
;; block1
; Create another solid block.
(define block2
    (solid:block (position 0 –20 0)
    (position 30 10 30)))
;; block2
; Nonregularize unite blocks 1 and 2.
(define unite (bool:nonreg–unite block1 block2))
;; unite
; OUTPUT Original

; Regularize the result.
(define reg (bool:regularise block1))
;; reg
; OUTPUT Result
```

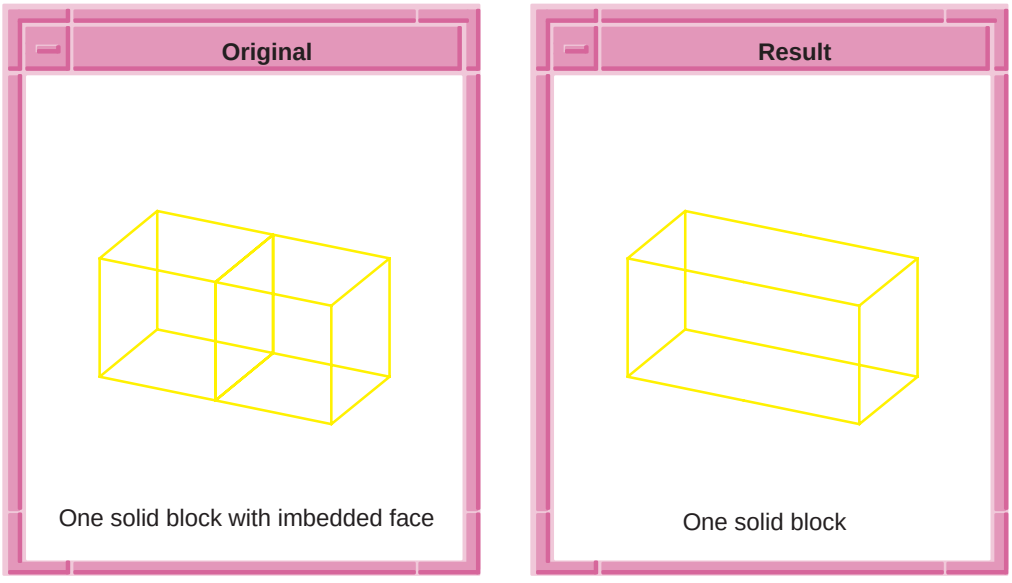

Figure 2-14. bool:regularise

bool:sel–imprint

Scheme Extension: Booleans

Action: Imprints the intersection graph of a set of selected faces of the tool body and a set of selected faces of blank body.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_selectively_imprint

Syntax:
```
(bool:sel–imprint tool–body–or–faces
blank–body–or–faces [split–check=#t] [acis–opts])
```

Arg Types:		
	tool–body–or–faces	body | (face ...)
	blank–body–or–faces	body | (face ...)
	split–check	boolean
	acis–opts	acis–options

Returns: entity

Errors: no_intsct, improper_split

Description: This extension imprints the intersection graph of the tool body or faces and the blank body or faces.

The first argument requires the tool or a set of faces on the tool. The second argument requires the blank or a set of faces on the blank.

tool–body–or–faces is the tool or a set of faces on the tool.

blank–body–or–faces is the blank body or faces.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:sel–imprint
; Create a blank body
(define block1 (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Create a tool body
(define block2 (solid:block (position –30 10 –20)
    (position 20 0 30)))
;; block2
; OUTPUT Original

; Pick a blank face to imprint.
(define faces (entity:faces block1))
;; faces
(define face (list–ref faces 3))
;; face
(define imprint (bool:sel–imprint block2 face))
;; imprint
; OUTPUT Result
```

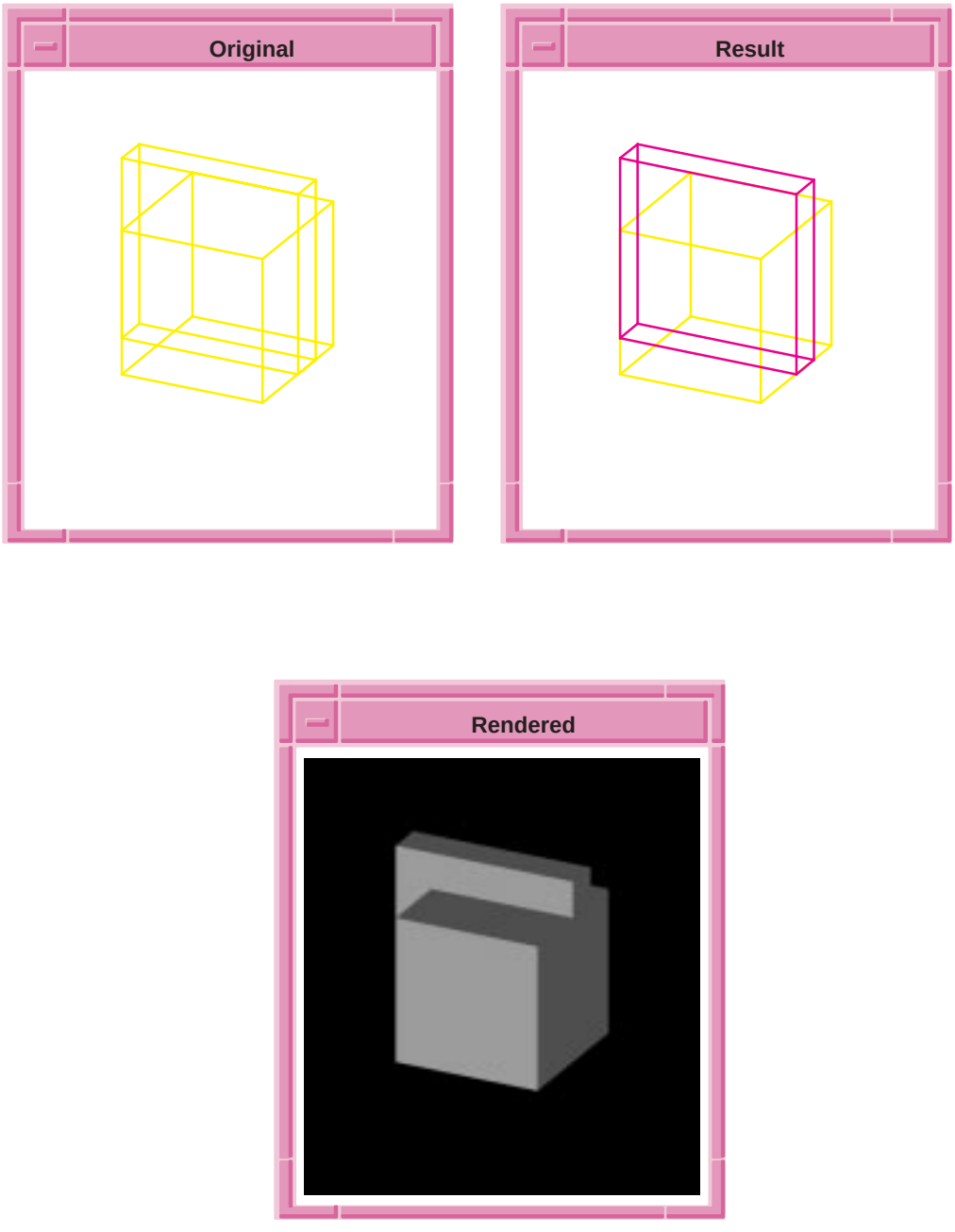

Figure 2-15. bool:sel–imprint

bool:subtract

Scheme Extension: Booleans

Action: Subtracts one or more bodies from a body.

Filename: bool/bool_scm/bool_scm.cxx

APIs: None

Syntax:
```
(bool:subtract body1 ... bodyn [keep–opt]
    [acis–opts])
```

Arg Types:		
	body1	body
	bodyn	body
	keep–opt	string
	acis–opts	acis–options

Returns: body

Errors: None

Description: This extension subtracts one or more bodies from the first body. By default, the result of the subtract is returned as body1 and the remaining bodies are deleted.

The optional keep–opt specifies whether some or all input bodies should be preserved. Value ”keep_blank” preserves body1 and returns the result of the subtraction as a new body. ”keep_tool” preserves body2 ... bodyn. ”keep_both” preserves all input bodies and returns the result of the subtraction as a new body.

This extension performs a regularized subtraction. Any faces, edges, and/or vertices not needed to support the topology are removed before returning the resulting body.

body1 is the first body of subtraction.

bodyn is the nth body of subtraction.

keep–opt specifies whether some or all input bodies should be preserved.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:subtract
; Create a solid block.
(define block1
    (solid:block
    (position –20 –20 –20) (position 0 0 0)))
;; block1
; Set color for block1.
(entity:set–color block1 2)
;; ()
; Create another solid block.
(define block2
    (solid:block
    (position –20 –20 –20) (position 20 20 20)))
;; block2
; Set color for block2.
(entity:set–color block2 3)
;; ()
; OUTPUT Original

; Subtract solid blocks 1 from 2.
(define subtract (bool:subtract block2 block1))
;; subtract
; #[entity 3 1]
; OUTPUT Result
```

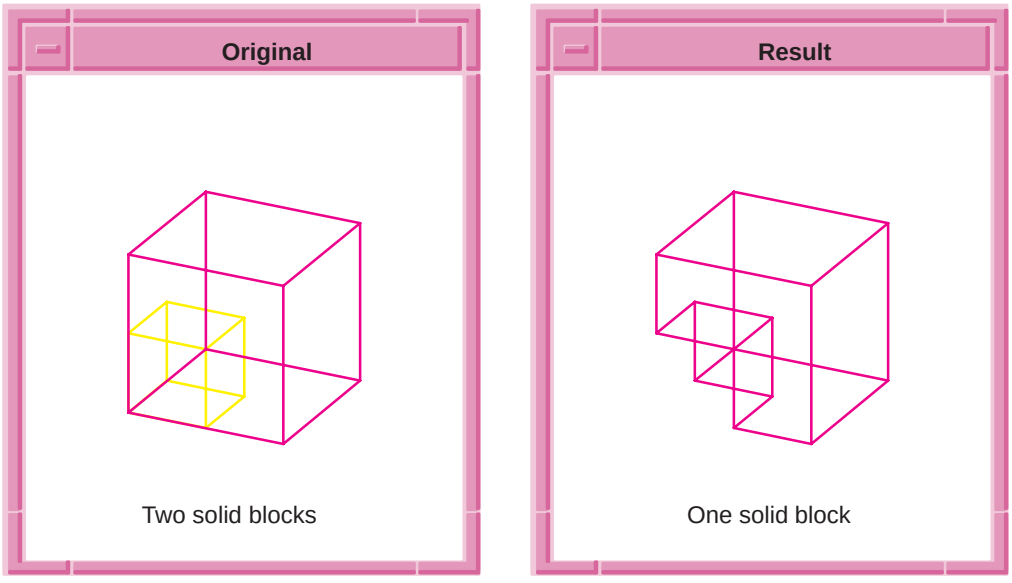

Figure 2-16. bool:subtract

bool:trim–faces

Scheme Extension: Booleans

Action: Trims the surfaces of the given faces.

Filename: bool/bool_scm/bool_scm.cxx

APIs: None

Syntax: `(bool:trim–faces faces [trim] [acis–opts])`

Arg Types:		
	faces	face | (face...)
	trim	string
	acis–opts	acis–options

Returns: (entity...)

Errors: None

Description: Trim each input face in the following way:

If the surface on which this face lies is parametric, trim it back to the face boundaries. Whether the trimming is done by subsetting or by splitting the surface is controlled by the ”trim” parameter, which specifies which surface types should be subset rather than split. This scheme extension only affects spline or torus surfaces, and only splines can be split, so the following trim options are available:

”all”	subset both spline and toruses, ignore all others
”spline”	subset splines only, ignore all other types
”torus”	subset tori only, splie splines, ignore all others
”none”	do not subset any surface type, split splines

If any surface is not subsetted, it will be split explicitly. Any surface other than a spline ot torus will be left alone.

face is an input face.

trim is a different trim options available.

acis–opts contains parameters for versioning and journaling.

Limitations: Only operates on splines and tori. Surface is trimmed to parametric box of face’s pcurves.

Example:

```
; bool:trim–faces
; create solid block.
(define b (solid:block (position 0 0 0)
    (position –10 –10 –10)))
;; b
; Trim the surfaces of the defined faces.
(define trim (bool:trim–faces
    (entity:faces b) ”none”))
;; trim
```

bool:unite

Scheme Extension: Booleans

Action: Unites two or more bodies.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_boolean

Syntax:

```
(bool:unite body1 ... bodyn [keep–opt]
    [acis–opts])
```

Arg Types:		
	body1	body
	bodyn	body
	keep–opt	string
	acis–opts	acis–options

Returns: body

Errors: None

Description: This extension unites two or more bodies,. By default, the result of the unite is returned as body1 and the remaining bodies are deleted.

The optional keep–opt specifies whether some or all input bodies should be preserved. Value ”keep_blank” preserves body1 and returns the result of the unite as a new body. ”keep_tool” preserves body2 ... bodyn. ”keep_both” preserves all input bodies and returns the result of the unite as a new body.

This extension checks for overlapping bodies. If it is known that the bodies do not overlap, use body:combine, because it does not perform intersection checks. If it is known that the bodies overlap, or if there is some uncertainty as to whether the bodies overlap, use body:unite.

This extension performs a regularized unite. Any faces, edges, and/or vertices not needed to support the topology are removed before returning the resulting body.

body1 is the first body of unite.

bodyn is the nth body of unite.

keep–opt specifies whether some or all input bodies should be preserved.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:unite
; Create a solid block.
(define block1
    (solid:block
    (position 30 10 20) (position –30 –10 –20)))
;; block1
; Set color for block1.
(entity:set–color block1 2)
;; ()
; Create a solid cylinder.
(define cyl2
    (solid:cylinder
    (position 20 0 –20) (position 20 0 20) 20))
;; cyl2
; Set color for cyl2.
(entity:set–color cyl2 3)
;; ()
; OUTPUT Original
```

```
; Unite the block and cylinder
(define unite (bool:unite block1 cyl2))
;; unite
; #[entity 2 1]
; OUTPUT Result
```

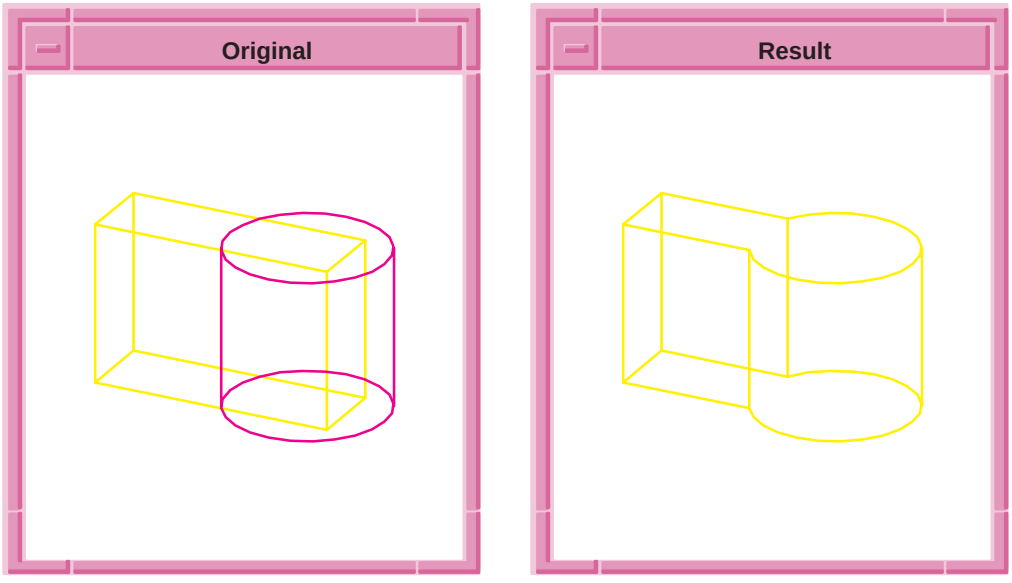

Figure 2-17. bool:unite

bool:unite–wires

Scheme Extension: Booleans

Action: Unites the wires of the tool body with the wires of the blank body.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_unite_wires

Syntax: (**bool:unite–wires** tool blank [acis–opts])

Arg Types:		
	tool	body
	blank	body
	acis–opts	acis–options

Returns: body

Errors: None

Description: This scheme command unites the wires of the tool body with the wires of the blank body where they meet at common vertices. The result is the blank body. The tool body is deleted.

tool is an input tool body.

blank is a blank body.

acis–opts parameter can be used for setting the version and journal information.

Limitations: None

Example:
```
; bool:unite–wires
; Create a tool body
(define t (wire–body:points
    (list (position 0 0 0)
    (position 1 0 0)
    (position 1 1 0)
    (position 0 1 0))))
;; t
; Create a blank body
(define b (wire–body:points
    (list (position 1 1 0)
    (position 1 1 1)
    (position 1 2 1)
    (position 1 2 0))))
;; b
; Unite the wires of both the bodies
(bool:unite–wires b t)
; OUTPUT Result
```

bool:wifa–imp

Scheme Extension: Booleans

Action: Imprints a wire on one or more faces of another body.

Filename: bool/bool_scm/wifaimp.cxx

APIs: None

Syntax: (**bool:wifa–imp** edge face*)

Arg Types:		
	wire	[edge | wire]
	face	face

Returns: boolean

Errors: Typical errors resulting from a boolean operation.

Description: Imprints the given wire or wire edge onto the given face (or faces) of another. Note the caller is promising that the wire lies exactly in the face (though the wire might be longer or shorter than the face). If the wire is not coincident with the face everywhere, the results are undefined.

Note that we actually imprint the intersection graph on ”both” entities, so the given wire might, for example, acquire extra vertices where it crosses face boundaries.

wire is the input wire or wire edge.

face is an input face.

Limitations: If an edge is supplied, it must belong to a wire body and be the wire’s only edge. The input wire must lie on the given faces. The faces must all be of the same body. Otherwise, undefined results.

Example:

```
; bool:wifa–imp
; Create a solid block
(define b (solid:block (position –30 –30 –30)
(position 30 30 30)))
;; b
; Create a wire body from an edge.
(define lin (edge:linear (position –30 –30 –10)
(position 30 –30 10)))
;; lin
(define w (wire–body lin))
;; w
; Select the face along which the wire body lies.
(define fs (entity:faces b))
;; fs
(define f (list–ref fs 2))
;; f
; Select the wire body’s (only) edge.
(define es (entity:edges w))
;; es
(define e (car es))
;; e
; Imprint the edge onto the face.
(bool:wifa–imp e f)
;; #t
```

edge:set–no–merge–attrib

Scheme Extension: Booleans

Action: Sets a NO_MERGE_ATTRIB to each edge in the input list of edges.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_set_no_merge_attrib

Syntax: `(edge:set–no–merge–attrib [edge–list] [acis–opts])`

Arg Types:

edge–list	(edge ...)
acis–opts	acis–options

Returns: unspecified

Errors: None

Description: This extension sets the NO_MERGE_ATTRIB for each edge in the input edge–list.

edge–list is an input edge list.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; edge:set–no–merge–attrib
; Create two blocks
(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 new faces and edges
(solid:imprint b1 b2)
;; #t
; Remove one of the bodies for visual purposes
(define erase (entity:erase b2))
;; erase
; #[entity 3 1]
; Pick one of the new imprinted faces
(define f1
    (pick:face (ray (position 0 1 1)
    (gvector 1 0 0)) 1))
;; f1
; Get the edges of the face
(define ed–list (entity:edges f1))
;; ed–list
; Apply the NO_MERGE_ATTRIB to the face edges.
(edge:set–no–merge–attrib ed–list)
;; ()
; Merge out the common face and edges. Note that the
; face with the marked edges is not removed by the
; merge command.
(define merge1 (bool:merge–faces b1 ”planes”))
;; merge1
; To test this, remove the NO_MERGE_ATTRIB from the
; edges.
(edge:remove–no–merge–attrib ed–list)
;; ()
; Repeat the merge command. Note that the edges
; now merge.
(define merge2 (bool:merge–faces b1 ”PLANES”))
;; merge2
```

entity:refresh–pattern

Scheme Extension: Patterns

Action: Refreshes the elements of a pattern to incorporate changes made to one of them.

Filename: bool/bool_scm/ref_pat_scm.cxx

APIs: api_refresh_entity_pattern

Syntax:
```
(entity:refresh–pattern entity refresh–list pattern
    [copy–pattern] [seed–index] [no–cross–faces]
[check])
```

Arg Types:

entity	entity
refresh–list	(entity ...)
pattern	pattern
copy–pattern	boolean
seed–index	integer
no–cross–faces	(entity ...)
check	integer | boolean

Returns: entity

Errors: The argument check does not lie in the range from –1 to 1.

Description: This Scheme extension refreshes the elements of the pattern in_pat to incorporate changes made to one of them. The entity in_ent should be taken from the modified element, while refresh–list should contain all pattern entities to be refreshed. It should not include anything from the modified element. By default, a copy of the pattern is made, and it is the copy that is actually applied to the entity. This behavior can be overridden by setting copy–pattern to FALSE. However, when copying is overridden and in_pat is shared by multiple bodies, a transform placed upon the bodies is transferred to the pattern multiple times, which is clearly undesirable. Also by default, entity is associated with the first pattern element (index 0), but may be associated with another element by furnishing the associated zero–based seed–index.

For cases in which the pattern is applied to a “bump” on a substrate rather than to an autonomous entity, the limits of the bump are automatically computed, but the user may choose to override the default limits by furnishing a list of no–cross–faces.

For performance reasons, the extension does not check the generated pattern of entities for intersection, containment, or compatibility unless the user sets the flag check. When set to 1 (or #t), the operation will be undone if the resulting body is invalid; when set to 2, the invalid pattern elements are automatically dropped (if necessary), and a valid body made from the valid elements. A value of 0 (or #f) yields the default behavior.

entity is an input entity.

refresh–list contains all pattern entities to be refreshed.

pattern contains number of elements.

copy–pattern is a boolean variable which can be set to have copy of pattern or not.

seed–index is used to associate element with pattern.

no–cross–faces is used to override the default limits.

check is a flag which can be set to check the generated pattern of entities for intersection, containment, or compatibility.

Limitations: None

Example:

```
; entity:refresh-pattern
; Create a body
(define body (solid:block (position 0 0 0)
    (position 10 10 10)))
;; body
; Create a pattern.
(define pat (pattern:linear (gvector 20 0 0) 4))
;; pat
; Apply the pattern to the body.
(define apply (entity:pattern body pat))
;; apply
; Blend an edge.
(define edge (list-ref (entity:edges body) 15))
;; edge
(define chamfer (blend:chamfer-on-edge
    (list edge) 5))
;; chamfer
(define fix (blend:fix edge))
;; fix
; Refresh the pattern.
(define lump_list (entity:lumps body))
;; lump_list
(define seed_lump (list-ref lump_list 1))
;; seed_lump
(define refresh_list (list (list-ref lump_list 0)
    (list-ref lump_list 2) (list-ref lump_list 3)))
;; refresh_list
(define refresh (entity:refresh-pattern seed_lump
    refresh_list pat #t 1))
;; refresh
```

entity:remove–pcurves

Scheme Extension: Model Topology

Action: Removes pcurves from all coedges on analytic faces of the body or the selected entity.

Filename: bool/bool_scm/bool_scm.cxx

APIs: None

Syntax: (**entity:remove-pcurves** entity)

Arg Types:	entity	entity

Returns: entity

Errors: None

Description: This extension removes all pcurves underlying the coedges on analytic faces of the body or selected entity. The selected entity can be either an edge or a face.

entity is an input face or edge.

Limitations: None

Example:

```
; entity:remove–pcurves
; Create an entity
(define block1 (solid:block (position –10 –10 –10)
    (position 10 10 10)))
;; block1
; Add pcurves to all coedges
(define add–p (entity:reset–pcurves block1 #t))
;; add–p
(entity:debug block1 3)
;   1 body record,     36 bytes
;   2 attrib records,  100 bytes
;   1 lump record,     36 bytes
;   1 transform record,136 bytes
;   1 shell record,    44 bytes
;   6 face records,    288 bytes
;   6 loop records,    216 bytes
;   6 surface records, 1056 bytes
;   24 coedge records, 1248 bytes
;   12 edge records,   960 bytes
;   24 pcurve records, 2496 bytes
;   8 vertex records,  256 bytes
;   12 curve records,  1536 bytes
;   8 point records,   512 bytes
; Total storage 8920 bytes
;; ”solid body”
; Remove pcurves from coedges on analytic faces
; – all coedges in this case
(define rem–p (entity:remove–pcurves block1))
;; rem–p
(entity:debug block1 3)
;   1 body record,     36 bytes
;   2 attrib records,  100 bytes
;   1 lump record,     36 bytes
;   1 transform record,136 bytes
```

```
;    1 shell record,   44 bytes
;   6 face records,    288 bytes
;   6 loop records,        216 bytes
;   6 surface records,     1056 bytes
;   24 coedge records,     1248 bytes
;   12 edge records,       960 bytes
;   8 vertex records,      256 bytes
;   12 curve records,      1536 bytes
;   8 point records,       512 bytes
;   Total storage 6424 bytes
;; ”solid body”
```

entity:reset–pcurves

Scheme Extension: Model Topology

Action: Removes and then adds back pcurves from all coedges of the body or the selected entity.

Filename: bool/bool_scm/bool_scm.cxx

APIs: None

Syntax: (**entity:reset–pcurves** entity [coedges])

Arg Types:		
	entity	entity
	coedges	boolean

Returns: entity

Errors: None

Description: The pcurve underlying each coedge of the body or the selected entity is first removed and then regenerated. The selected entity can be either an edge or a face. In the case of an edge, the pcurve of each coedge of the edge is removed, and then regenerated. In the case of a face, each pcurve of each coedge of each loop of the face is removed and then regenerated.

When coedges are supplied and have a value of TRUE, all coedges (even those originally without pcurves) will have pcurves after this command is executed. Otherwise, only coedges on non–analytic faces will be given pcurves.

entity is an input face or edge.

coedges is a boolean argument which can be set to either have all the pcurves or not.

Limitations: None

Example:

```
; entity:reset–pcurves
; Create an entity
(define wiggle (solid:wiggle 10 10 10 –2 –1 2 1))
;; wiggle
; Regenerate pcurves on coedges on spline faces
(define reset (entity:reset–pcurves wiggle))
;; reset
(entity:debug wiggle 3)
;   1 body record,          36 bytes
;   2 attrib records,       100 bytes
;   1 lump record,          36 bytes
;   1 shell record,         44 bytes
;   6 face records,         288 bytes
;   6 loop records,         216 bytes
;   6 surface records,      1064 bytes
;   24 coedge records,      1248 bytes
;   12 edge records,        960 bytes
;   4 pcurve records,       416 bytes
;   8 vertex records,       256 bytes
;   12 curve records,       1504 bytes
;   8 point records,        512 bytes
;   Total storage 6680 bytes
;; ”solid body”

; Regenerate pcurves already present and
; add pcurves to any coedges lacking pcurves
(define reset2 (entity:reset–pcurves wiggle #t))
;; reset2
(entity:debug wiggle 3)
;   1 body record,          36 bytes
;   2 attrib records,       100 bytes
;   1 lump record,          36 bytes
;   1 shell record,         44 bytes
;   6 face records,         288 bytes
;   6 loop records,         216 bytes
;   6 surface records,      1064 bytes
;   24 coedge records,      1248 bytes
;   12 edge records,        960 bytes
;   24 pcurve records,      2496 bytes
;   8 vertex records,       256 bytes
;   12 curve records,       1504 bytes
;   8 point records,        512 bytes
;   Total storage 8760 bytes
;; ”solid body”
```

entity:spline–convert

Scheme Extension: Model Topology

Action: Creates a new entity which is a spline equivalent of the original entity.

Filename: bool/bool_scm/cspl_scm.cxx

APIs: api_convert_to_spline, api_edge_to_spline

Syntax: (**entity:spline–convert** entity [acis–opts])

Arg Types:

entity	entity
acis–opts	acis–options

Returns: entity

Errors: None

Description: Creates a new entity which is a spline equivalent of the original entity with analytical faces or a spline edge equivalent of the original edge. The original entity must be an edge, body, face, or shell.

entity is an edge, body, face, or shell.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; entity:spline–convert
; Create solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 20 20 20)))
;; block1
; Define the faces of that block.
(define faces1 (entity:faces block1))
;; faces1
(face:spline? (list–ref faces1 0))
;; #f
(define blend (solid:blend–edges
    (list–ref (entity:edges block1) 3) 3))
;; blend
; (#[entity 2 1])
(define blocks2 (entity:spline–convert block1))
;; blocks2
(define faces2 (entity:faces blocks2))
;; faces2
(face:spline? (list–ref faces2 3))
;; #t
```

```
; Example 2
; Create sphere 1.
(define sphere1
    (solid:sphere (position 0 0 0) 40))
;; sphere1
; Create sphere 2
(define sphere2
    (solid:sphere (position 20 –20 0) 30))
;; sphere2
; Subtract one sphere from another to generate edges.
(define combo
    (solid:subtract sphere1 sphere2))
;; combo

; Spline convert the result and use debug to examine
; the unconverted and converted bodies.
(define cnv1 (entity:spline–convert sphere1))
;; cnv1
; Should not show spline curves in the output.
(entity:debug sphere1 3)
;   1 body record,     36 bytes
;   2 attrib records,  100 bytes
;   1 lump record,     36 bytes
;   1 transform record,136 bytes
;   1 shell record,    44 bytes
;   2 face records,    96 bytes
;   2 loop records,    72 bytes
;   2 surface records, 368 bytes
;   2 coedge records,  104 bytes
;   1 edge records,    80 bytes
;   1 vertex records,  32 bytes
;   1 curve records,   216 bytes
;   1 point records,   64 bytes
;   Total storage 1384 bytes
;; ”solid body”
```

```
; Should now show newly generated spline curves
; in the output.
(entity:debug cnv1 3)
;   1 body record,          36 bytes
;   2 attribute records,    100 bytes
;   1 lump record,          36 bytes
;   1 transform record,     136 bytes
;   1 shell record,         44 bytes
;   4 face records,         192 bytes
;   4 loop records,         144 bytes
;   4 surface records,      736 bytes
;   18 coedge records,      936 bytes
;   9 edge records,         720 bytes
;   18 pcurve records,      1872 bytes
;   7 vertex records,       224 bytes
;   9 curve records,        1944 bytes
;   7 point records,        448 bytes
;   Total storage 7568 bytes
;; ”solid body”
```

face:intersect

Scheme Extension: Booleans

Action: Gets the intersection curve between two faces.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_clean_wire, api_fafa_int

Syntax: (**face:intersect** face1 face2 [acis–opts])

Arg Types:		
	face1	face
	face2	face
	acis–opts	acis–options

Returns: wire–body

Errors: None

Description: This extension finds the intersection curve between two faces. It returns the curve in the form of a wire–body with one or more circuits.

face1 is an input face.

face2 is the other input face.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; face:intersect
; Create a planar face.
(define face1 (face:plane
    (position –30 –20 –10) 60 40))
;; face1
; Set color for face1.
(entity:set–color face1 2)
;; ()
; Create another planar face.
(define face2
    (face:plane (position 0 –20 20)
    60 40 (gvector 1 0 0)))
;; face2
; Set color for face2.
(entity:set–color face2 3)
;; ()
; OUTPUT Two Faces

; Find the intersection curve between the two faces.
(define intersect1
    (face:intersect face1 face2))
;; intersect1
; OUTPUT Intersection
```

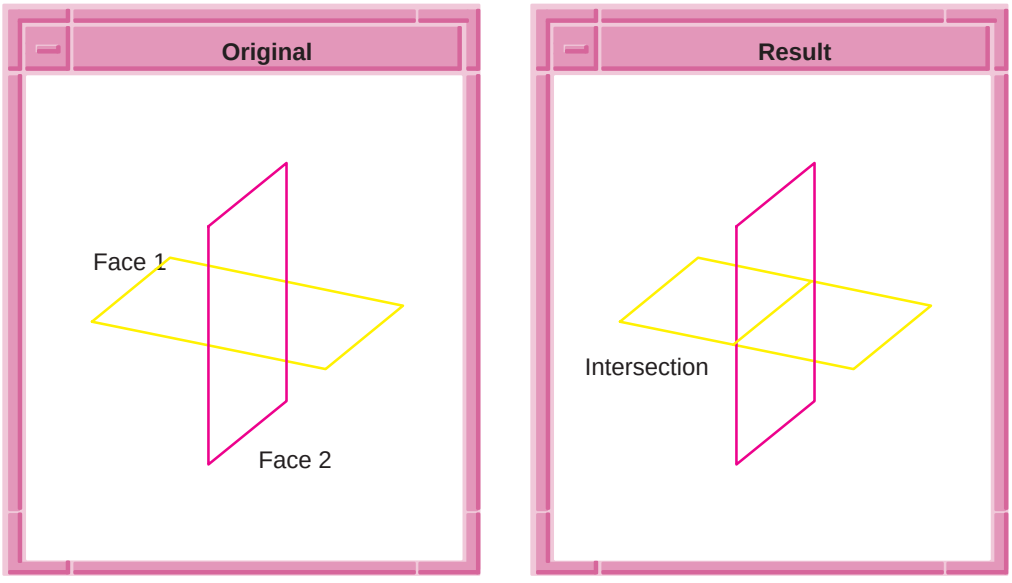

Figure 2-18. face:intersect

face:remove

Scheme Extension: Model Topology

Action: Removes a face from a body.

Filename: bool/bool_scm/rfac_scm.cxx

APIs: api_remove_face

Syntax: `(face:remove face)`

Arg Types: face face

Returns: body

Errors: None

Description: This removes the face from its owning body. When edges and vertices are no longer needed to support faces, they are also removed. This differs from face:uncover.

face is an input face.

Limitations: None

Example:

```
; face:remove
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Get a list of the block’s faces.
(define faces (entity:faces block1))
;; faces
; OUTPUT Original

; Remove three of the faces to observe the result.
(define remove1 (face:remove (car faces)))
;; remove1
(define remove2 (face:remove (car (cdr faces))))
;; remove2
(define remove3
    (face:remove (car (cdr (cdr faces)))))
;; remove3
; OUTPUT Result
```

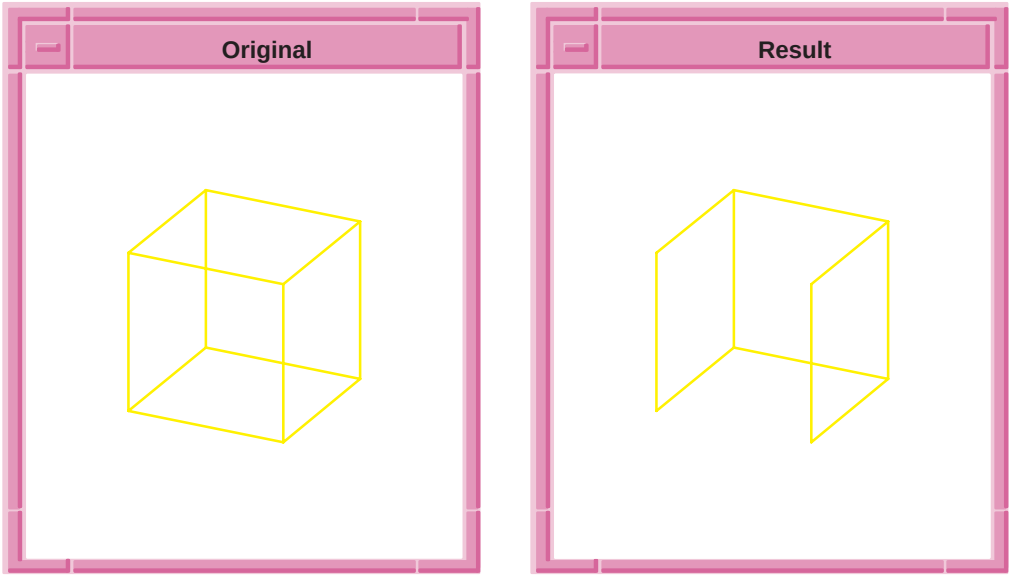

Figure 2-19. face:remove

face:split

Scheme Extension: Model Topology

Action: Splits a face along “u” or “v” at a given value.

Filename: bool/bool_scm/rfac_scm.cxx

APIs: api_split_face

Syntax:
```
(face:split face split–along–u use–percent
    param–value [acis–opts])
```

Arg Types:

face	face
split–along–u	boolean
use–percent	boolean
param–value	real
acis–opts	acis–options

Returns: boolean

Errors: None

Description: This extension splits a face along the ”u” or ”v” axis at a given parameter supplied. Set use–percent to #t for the split to occur at a relative portion of the face between 0 and 1. This value is passed in by the real param–value.

face is an input face.

split–along–u is a boolean variable which can be set to split along “u” axis.

use–percent set to #t for the split to occur at a relative portion of the face between 0 and 1.

param–value is the value of split.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; face:split
; Create a block
(define b (solid:block (position –10 –10 –10)
    (position 10 10 10)))
;; b
; Get the faces of the block
(entity:faces b)
;; (#[entity 3 1] #[entity 4 1] #[entity 5 1]
;; #[entity 6 1] #[entity 7 1] #[entity 8 1])
; Define a face
(define face1 (entity 5))
;; face1
(define facesplit (face:split face1 #t #f 7))
;; facesplit
```

face:split–at–disc

Scheme Extension: Model Topology

Action: Splits a face along G1 or G2 discontinuities.

Filename: bool/bool_scm/rfac_scm.cxx

APIs: api_split_face_at_g_disc

Syntax: (**face:split–at–disc** face [disc–order = 1] [acis–opts])

Arg Types:		
	face	face
	disc–order	integer
	acis–opts	acis–options

Returns: (face1 face2 ....)

Errors: face does not contain discontinuity information

Description: This extension splits a face along the ”u” or ”v” isoparametric lines at G1 or G2 discontinuity parameter. The result is the list of new faces. The optional acis–opts contains parameters for versioning and journaling.

face is an input face.

disc–order indicates the type of discontinuity .

acis–opts contains parameters for versioning and journaling.

Limitations: If the supplied face is an independent face (i.e. no body, lump or shell) it will return a list of faces which share edges and do not belong to a shell lump or body either. It will also split at discontinuities in the range of the face.

Example:

```
(part:clear)
; create a discontinuos in G1 curve by creating a
; spline with duplicate knots in the interior
(define ctrlpts_pos (list (position –2 0 0)(position
–1.5 0 0)
                          (position 0 0 0)(position
.75 0.5 0)
                          (position 1 0.5
0)(position 1.1 1 0)
                          (position 1.2 1
0)(position 1.5 1 0)
                          (position 1.9 .75
0)(position 2 0.5 0)
                          (position 2.1 .25
0)(position 2.5 0 0)
                          (position 3 0 0)(position
4 0 0)) )
;; ctrlpts_pos
(define knot_v (list  0 0 0 1 2 3 4 4 4 5 6 7 7 7 8 9
9 9) )
;; knot_v
(define spline (edge:spline–from–ctrlpts ctrlpts_pos
knot_v) )
;; spline
(define spline (sweep:law spline (gvector 0 0 2)  ))
;; spline
(define f (list–ref (entity:faces spline)0))
;; f
(entity:check f)
;; checked:
;;        0 lumps
;;        0 shells
;;        0 wires
;;        1 faces
;;        1 loops
;;        4 coedges
;;        4 edges
;;        4 vertices
;;()
(face:split–g f )
;;
(entity:check spline)
;;
```

face:uncover

Scheme Extension: Model Topology

Action: Removes the surface of a face, leaving its edges.

Filename: bool/bool_scm/rfac_scm.cxx

APIs: api_mk_by_faces, api_uncover_face

Syntax: `(face:uncover face [acis–opts])`

Arg Types:		
	face	face
	acis–opts	acis–options

Returns: body

Errors: None

Description: This extension removes a surface of a face leaving its supporting edges and vertices. This results in a partly covered body. The return body is the owner of the input face.

The input face is removed by deleting it and its surface, any associated loops, and any associated coedges. It leaves the face's edges. If removing a coedge leaves the edge with no coedges, it is kept. In addition, its immediate owner is changed either to the shell or to one of the wires in the newly created wireframe. This differs from the face:unhook, which removes the face's edges.

face is an input face.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; face:uncover
; Create a solid block.
(define block1
    (solid:block (position –30 –15 –45)
    (position 20 5 40)))
;; block1
; List the current faces of block1.
(entity:faces block1)
;; (#[entity 3 1] #[entity 4 1] #[entity 5 1]
;; #[entity 6 1] #[entity 7 1] #[entity 8 1])
; Create a list of faces from the block.
(define faces (entity:faces block1))
;; faces
```

```
; Uncover one of the faces.
(face:uncover (list-ref faces 5))
;; #[entity 2 1]
; Verify a face has been removed by listing current
; faces.
(entity:faces block1)
;; (#[entity 3 1] #[entity 4 1] #[entity 5 1]
;; #[entity 6 1] #[entity 7 1])
```

face:unhook

Scheme Extension: Model Topology

Action: Removes a face from a body.

Filename: bool/bool_scm/rfac_scm.cxx

APIs: api_unhook_face

Syntax: (**face:unhook** face [acis-opts])

Arg Types:	face	face
	acis–opts	acis–options

Returns: body

Errors: None

Description: This Scheme extension removes a face from its owning body and puts a copy into a new body. When edges and vertices are no longer needed to support faces, they are also removed. The return body is the owner of the input face.

This differs from the face:uncover. face:unhook makes a copy of the face. It removes the face, its surface, its associated loops, its associated coedges, and the associated edges and vertices no longer needed to support the remaining faces. face:uncover leaves the face's edges.

face is an input face.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; face:unhook
; Create a solid block.
(define block1
    (solid:block
    (position –10 –15 0) (position 5 10 10)))
;; block1
; Create a list of faces from the block.
(define faces (entity:faces block1))
;; faces
; OUTPUT Original

; Unhook one of the faces.
(define face1 (face:unhook (list–ref faces 2)))
;; face1
; Verify one face has been unhooked.
faces
;; (#[entity 3 1] #[entity 4 1] #[(deleted) entity 5]
;; #[entity 6 1] #[entity 7 1] #[entity 8 1])
(define move1
    (transform:translation (gvector 0 –2 0)))
;; move1
(define transform (entity:transform face1 move1))
;; transform
(view:refresh)
;; #[view 10759869]
; OUTPUT Result
```

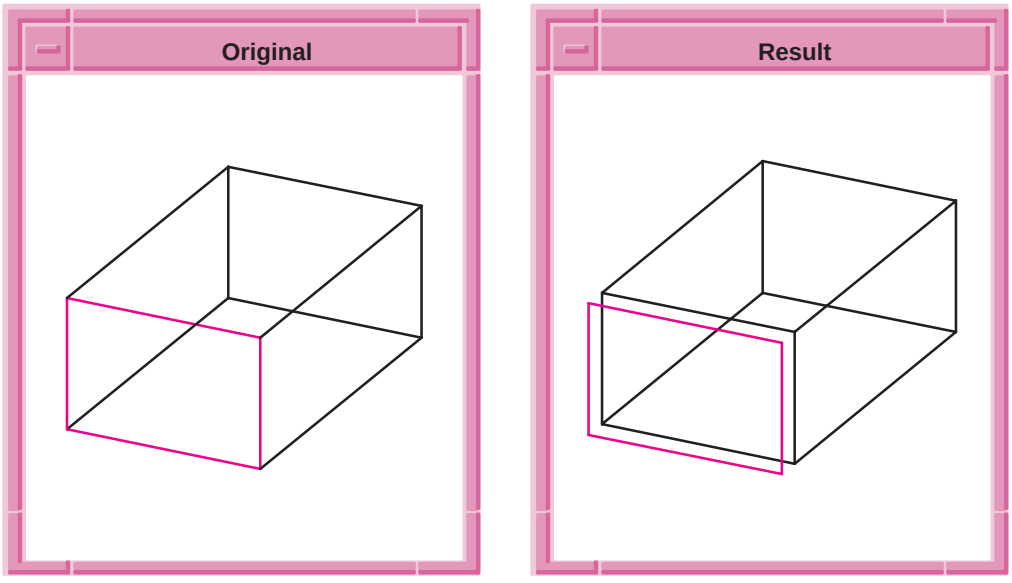

Figure 2-20. face:unhook

glue:options

Scheme Extension: Booleans

Action: Sets the options in the data structure to be used by glue operations.

Filename: bool/bool_scm/glue_scm.cxx

APIs: None

Syntax: `(glue:options "name-of-option" {value} [glue-opts])`

Arg Types:		
	"name–of–option"	string
	value	boolean
	glue–opts	glue–options

Returns: glue–options

Errors: None

Description: Every option default value is set to #f (unset). These options are designed to provide additional information to the glue operation for improved performance. The information provided must be accurate because the glue operation relies heavily on this information.

See bool:glue–unite for the definition of coincident faces.

Given bodies b1 and b2, a coincident patch P1 in b1 is a maximal set of connected faces of b1 such that there exists a corresponding maximal set P2 of connected faces in b2 and a (well–defined onto) coincidence mapping from P1 to P2.

Face f1 covers face f2 if the point set of f2 is a subset of the point set of f1. Patch P1 covers patch P2 if the point set of P2 is a subset of the point set of P1. Given a pair of coincident patches P1 and P2, P1 is a strict cover of P2 if the point set of P2 is a subset of the interior point set of P1.

The name–of–option argument could take one of the following values:

patch_and_face_cover
blank_patches_strict_cover
non_trivial

Setting patch_and_face_cover to #t will induce a performance enhancement. patch_and_face_cover may be set to #t if the following conditions are satisfied:

for every pair of coincident faces (to be specified in the glue operation), one face covers the other face;
for every pair of coincident patches, one patch covers the other patch.

In addition to setting patch_and_face_cover to #t, setting blank_patches_strict_cover to #t will induce another performance enhancement. blank_patches_strict_cover may be set to #t if the following conditions are satisfied:

patch_and_face_cover is set to #t;
every patch in the blank (first) body is a strict cover of its corresponding patch in the tool (second) body.

non_trivial may be set to #t if it is guaranteed that the Boolean operation will be non–trivial. In the case of glue–unite, this is when the tool body lies outside the blank body. In the case of glue–subtract, this is when the tool body is completely contained in the blank body. This will induce another performance enhancement. It is not dependent on the previous flags.

name–of–option argument could take one of the following values:
patch_and_face_cover blank_patches_strict_cover non_trivial

value is the value of the option.

glue–opts is the glue options.

Limitations: None

Example:

```
; glue:options
; Create a block
(define block1 (solid:block
    (position –10 –10 25) (position 35 10 35)))
;; block1
(define block2 (solid:block
    (position 20 –10 15) (position 35 10 25)))
;; block2
(define faces1 (entity:faces block1))
;; faces1
(define faces2 (entity:faces block2))
;; faces2
; Pick out the coincident faces on each body
(define f11 (list–ref faces1 1))
;; f11
(define f20 (list–ref faces2 0))
;; f20
; change highlight color for images
(env:set–highlight–color 1)
;; ()
; Highlight the face pair to see they are coincident
(entity:set–highlight (list f11 f20) #t)
;; (#[entity 33 1] #[entity 38 1])
; Construct a glue options object.
; face_pair_cover can be set to #t because
; f11 covers f20.
; blank_patches_strict_cover cannot be set
; to #t because f11 (a patch with one face) does not
; strictly cover f20 (the corresponding patch with
; one face) –– the patches have coincident boundary
; edges. non_trivial can be set to #t because the
; bodies lie outside one another.
(define g (glue:options ”face_pair_cover” #t
    ”non_trivial” #t))g
;; g
;; #[Glue_Options  ”face_pair_cover” 1
;; ”blank_patches_strict_cover” –1 ”non_trivial” 1]
(define glue–result (bool:glue–unite block1 block2
    (list f11) (list f20) g))
;; glue–result
(entity:check glue–result)
```

```
; checked:
;        1 lumps
;        1 shells
;        0 wires
;        8 faces
;        8 loops
;        36 coedges
;        18 edges
;        12 vertices
;; ()
; OUTPUT Original


; Now create another block
(define block3 (solid:block
    (position –20 –20 –20) (position 20 25 25)))
;; block3
(define faces3 (entity:faces block3))
;; faces3
(define faces4 (entity:faces glue–result))
;; faces4
; Pick out the coincident faces
(define f30 (list–ref faces3 0))
;; f30
(define f35 (list–ref faces3 5))
;; f35
(define f42 (list–ref faces4 2))
;; f42
(define f41 (list–ref faces4 1))
;; f41
; Highlight each face pair to see they are coincident
(entity:set–highlight (list f30 f42) #t)
;; (#[entity 17 1] #[entity 23 1])
(entity:set–highlight (list f35 f41) #t)
;; (#[entity 22 1] #[entity 13 1])
; Construct a glue options object.
; face_pair_cover can be set to #t because
; f30 covers f43 and f35 covers f41.
; blank_patches_strict_cover can be set to #t because
; the patch consisting of f30 and f35 strictly covers
; the patch consisting of f43 and f41.  The boundary
; edges of the smaller patch (f43 and f41)
; do not touch the boundary edges of the larger
; patch (f30 and f35). non_trivial can be set to #t
; because the bodies lie outside one another.
; In this example, we re–use the previous glue
```

```
; options object.
(define g (glue:options
    "blank_patches_strict_cover" #t g))
;; g
; g => #[Glue_Options  "face_pair_cover" 1
; "blank_patches_strict_cover" 1 "non_trivial" 1]
(bool:glue-unite block3 glue-result
    (list f30 f35) (list f42 f41) g)
;;  #[entity 16 1]
(entity:check block3)
; checked:
;         1 lumps
;         1 shells
;         0 wires
;         12 faces
;         12 loops
;         60 coedges
;         30 edges
;         20 vertices
;; ()
; OUTPUT Result
```

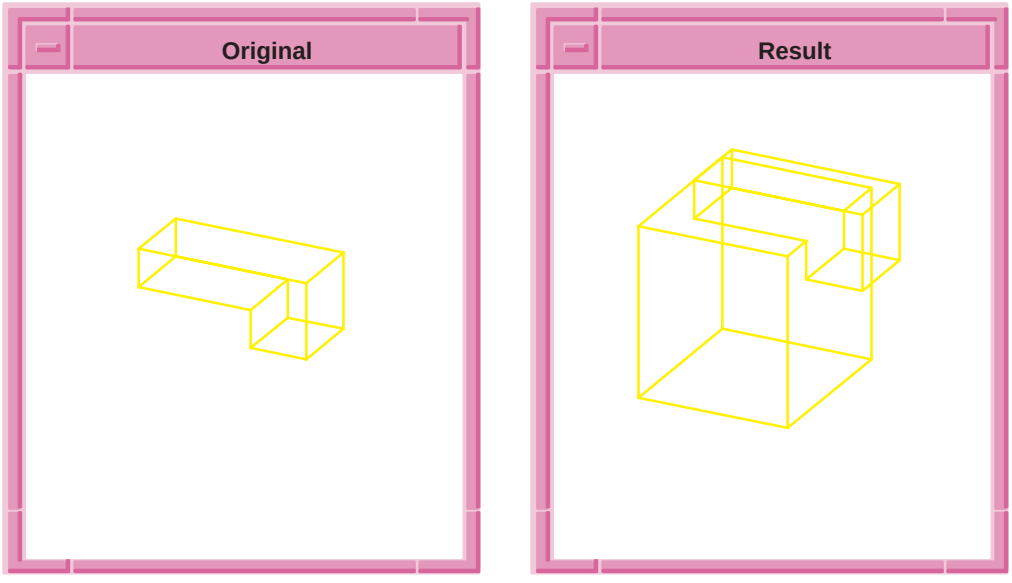

Figure 2-21. glue:options

solid:check–ff–intersections

Scheme Extension: Booleans

Action: Checks all faces for improper intersections.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_check_entity_ff_ints

Syntax:
```
(solid:check–ff–intersections entity [file–name]
    [acis–opts])
```

Arg Types:

entity	entity
file–name	string
acis–opts	acis–options

Returns: entity | entity ...

Errors: None

Description: This extension checks all faces for improper intersections.

entity is an input entity.

file–name is the name of the file.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:
```
; solid:check–ff–intersections
; Define a solid block
(define b (solid:block (position 0 0 0)
    (position 20 20 20)))
;; b
(solid:check–ff–intersections b)
;; ()
```

solid:imprint

Scheme Extension: Booleans

Action: Imprints curves of intersection of two bodies onto the faces of bodies.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_imprint

Syntax:
```
(solid:imprint body1 body2 [acis–opts])
```

Arg Types:	body1	body
	body2	body
	acis–opts	acis–options

Returns: boolean

Errors: None

Description: Calculates the curves of intersection of the two bodies and imprints these curves onto the faces of the two bodies. This results in the original faces being split either by breaking the face into several pieces or by adding slits to the face.

If no intersections exist between the bodies, a "no intersection" message occurs.

body1 is one of the input body.

body2 is the other input body.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; solid:imprint
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 25 25)))
;; block1
; Set color for block1.
(entity:set–color block1 2)
;; ()
; Create another solid block.
(define block2
    (solid:block (position –5 –20 –30)
    (position 30 15 35)))
;; block2
; Set color for block2.
(entity:set–color block2 3)
;; ()
; OUTPUT Original
```

```
; Imprint the faces of the two blocks.
(solid:imprint block1 block2)
;; #t
(define erase (entity:erase block2))
;; erase
; #[entity 3 1]
(view:refresh)
;; #[view 1076235512]
; OUTPUT Result
```

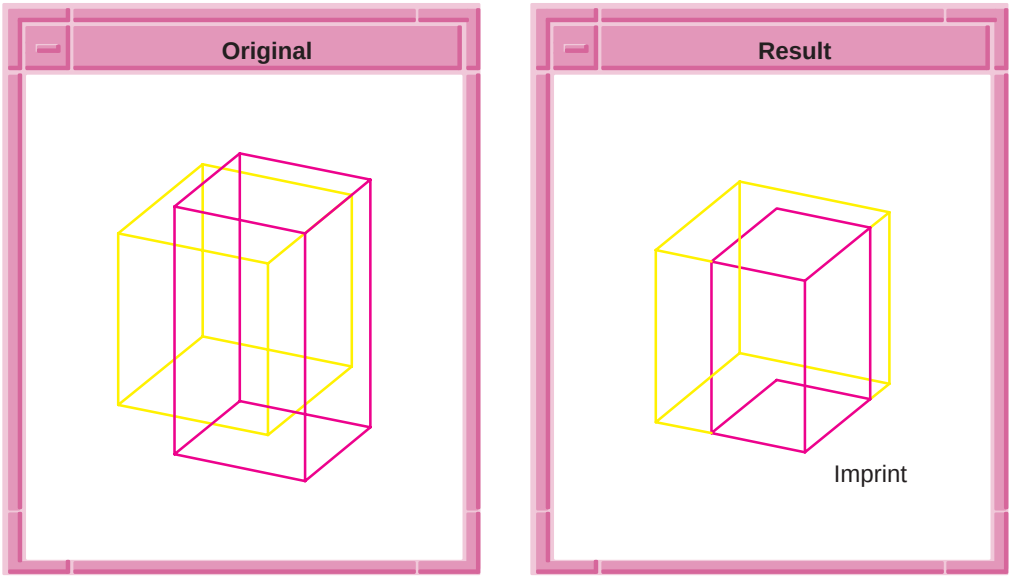

Figure 2-22. solid:imprint

solid:imprint–stitch

Scheme Extension: Booleans

Action: Joins body1 and body2 along the intersection graph.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_imprint_stitch

Syntax: `(solid:imprint-stitch body1 body2 [acis-opts])`

Arg Types:		
	body1	body
	body2	body
	acis–opts	acis–options

Returns: body

Errors: None

Description: Joins body1 and body2 along edges or vertices of the intersection curve between the body's faces. Like solid:stitch, solid:imprint–stitch performs a local union operation when creating a nonmanifold edge and may change face sidedness and containment. Bodies with no face/face intersections will be grouped in the output body (body1) regardless. body2 is deleted.

body1 is one of the input body.

body2 is the other input body.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; solid:imprint–stitch
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Set color for block1.
(entity:set–color block1 2)
;; ()
; Create another solid block.
(define block2
    (solid:block (position –30 –30 –30)
    (position 10 10 10)))
;; block2
; Change the color of block2.
(entity:set–color block2 3)
;; ()
; OUTPUT Original

; Imprint stitch the 2 blocks.
(define imprint (solid:imprint–stitch block1 block2))
;; imprint
; #[entity 2 1]
; OUTPUT Result
```

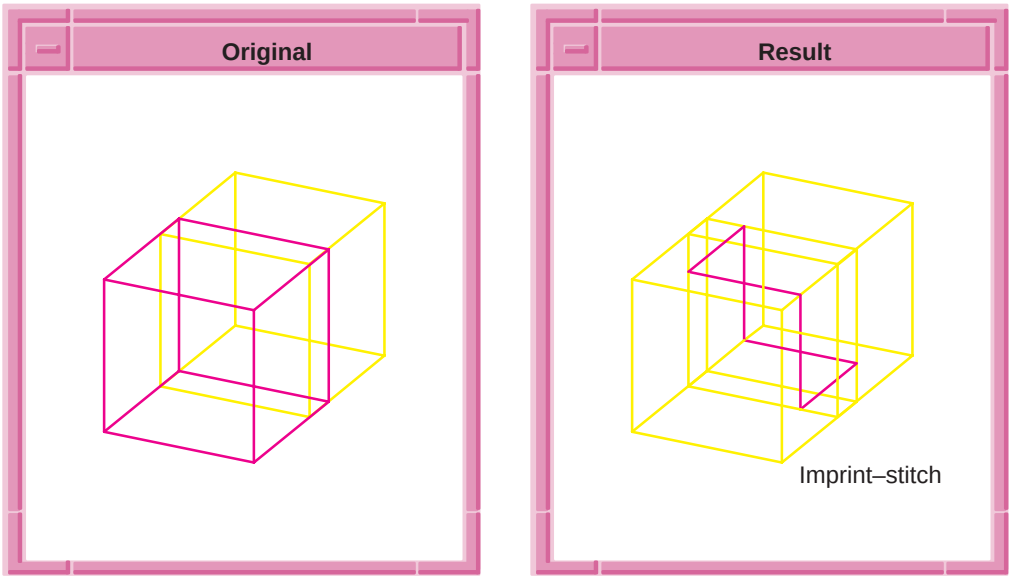

Figure 2-23. solid:imprint–stitch

solid:inter–graph

Scheme Extension: Booleans

Action: Gets the intersection graph between two bodies and returns it as a wire body.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_bool_make_intersection_graph

Syntax: `(solid:inter–graph blank tool [acis–opts])`

Arg Types:		
	blank	body
	tool	body
	acis–opts	acis–options

Returns: wire-body

Errors: None

Description: Gets the intersection graph between two bodies and returns it as a wire body. A wire body representing the intersection of the two bodies is returned. If no intersections exist between the bodies, a ”no intersection” message occurs.

blank is the body to be sliced.

tool is the slicing body.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; solid:slice
; Create a solid block.
(define block1 (solid:block (position –20 –20 –20)
                            (position 20 20 20)))
;; block1
; Set color for block1
(entity:set–color block1 2)
;; ()
; Create another solid block
(define block2 (solid:block (position –30 –30 –30)
                            (position 10 10 10)))
;; block2
; Set color for block2
(entity:set–color block2 3)
;; ()
; Slice the two blocks
(define slice (solid:inter–graph block1 block2))
;; slice
; OUTPUT Original Sliced Blocks
; Remove the blocks to see the slice.
(entity:delete (list block1 block2))
;; ()
; OUTPUT Resulting Slice
```

solid:intersect

Scheme Extension: Booleans

Action: Intersects a list of solids.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_intersect

Syntax: (**solid:intersect** bodies [acis–opts])

Arg Types:

bodies	entity | (entity ...)
acis–opts	acis–options

Returns: entity

Errors: None

Description: This extension requires that a minimum of two bodies be specified. body2 is intersected with body1. The name of the body1 becomes the result. For example, when intersecting body1 and body2, the result is a new body1 and body2 is deleted.

bodies is the input bodies.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; solid:intersect
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 20 20 40)))
;; block1
; Set color for block1.
(entity:set–color block1 2)
;; ()
; Create a solid sphere.
(define sphere1
    (solid:sphere (position 10 10 0) 20))
;; sphere1
; Set color for sphere1.
(entity:set–color sphere1 3)
;; ()
; OUTPUT Original

; Intersect the two solids.
(define intersect1 (solid:intersect block1 sphere1))
;; intersect1
; OUTPUT Result
```

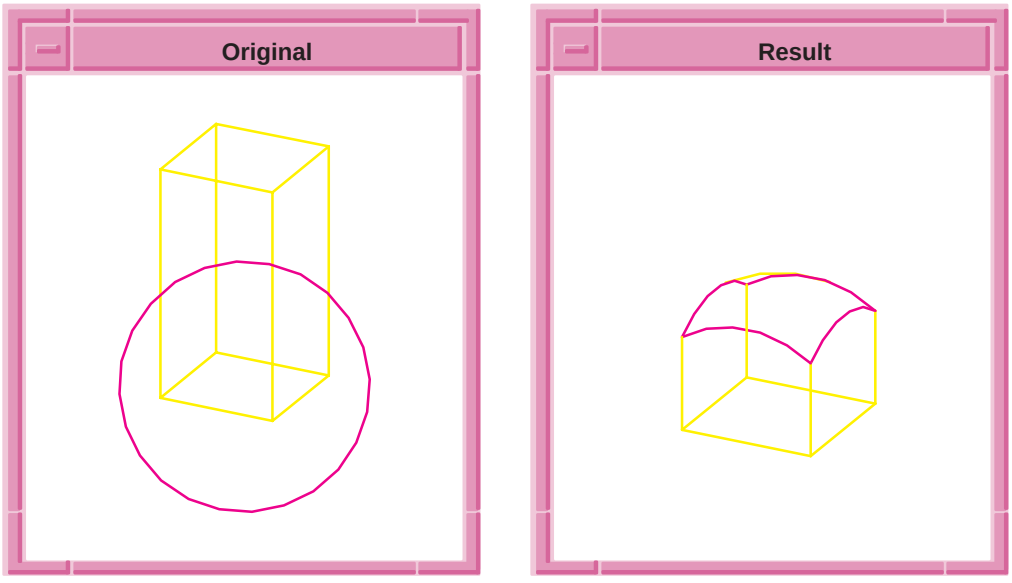

Figure 2-24. solid:intersect Example 1

```
; Example 2 solid:intersect
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 20 20 40)))
;; block1
; Set color for block1.
(entity:set-color block1 2)
;; ()
; Create another solid block.
(define block2
    (solid:block (position 10 10 10)
    (position 30 20 40)))
;; block2
; Set color for block2.
(entity:set-color block2 3)
;; ()
; OUTPUT Example 2 Original

; Intersect solid blocks 1 and 2.
(define intersect2 (solid:intersect block1 block2))
;; intersect2
; OUTPUT Example 2 Result
```

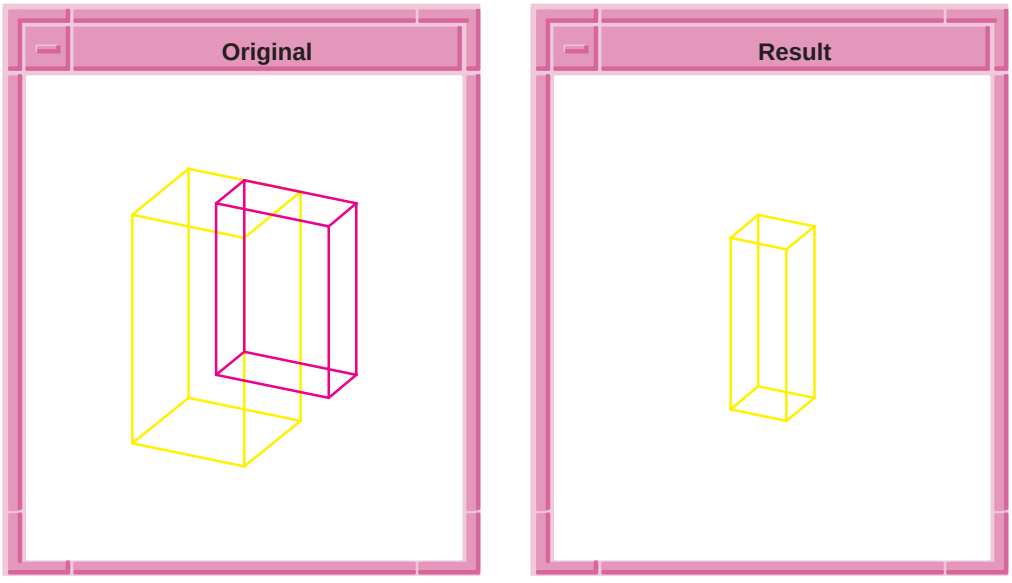

Figure 2-25. solid:intersect Example 2

solid:planar–slice

Scheme Extension: Booleans

Action: Slices a solid body with a plane to produce a wire body.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_planar_slice

Syntax:
```
(solid:planar-slice body plane-position
    plane-normal [acis-opts])
```

Arg Types:		
	body	body
	plane–position	position
	plane–normal	gvector
	acis–opts	acis–options

Returns: wire–body

Errors: None

Description: The argument body requires a solid body entity.

body is a solid body entity.

plane–position specifies the location to slice.

plane–normal specifies the direction to slice.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; solid:planar–slice
; Create a solid block.
(define block1
    (solid:block (position –10 –10 –10)
    (position 20 20 20)))
;; block1
; OUTPUT Original

; Slice the solid block with a plane.
(define slice (solid:planar–slice block1
    (position 5 5 5) (gvector 0.5 1 0.25)))
;; slice
; OUTPUT Result
```

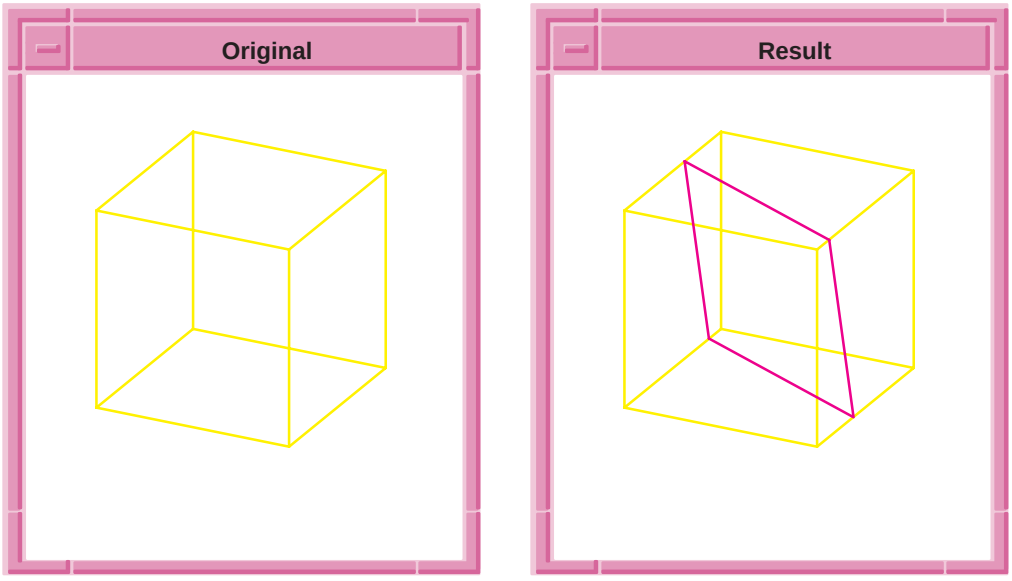

Figure 2-26. solid:planar–slice

solid:slice

Scheme Extension: Booleans

Action: Gets the intersection graph between two bodies and returns it as a wire body.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_clean_wire, api_slice

Syntax: `(solid:slice blank tool [acis–opts])`

Arg Types:

blank	body
tool	body
acis–opts	acis–options

Returns: wire–body

Errors: None

Description: Gets the intersection graph between two bodies and returns it as a wire body. A wire body representing the intersection of the two bodies is returned. If no intersections exist between the bodies, a “no intersection” message occurs.

blank is a blank body.

tool is a tool body.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; solid:slice
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Set color for block1.
(entity:set–color block1 2)
;; ()
; Create another solid block.
(define block2
    (solid:block (position –30 –30 –30)
    (position 10 10 10)))
;; block2
; Set color for block2.
(entity:set–color block2 3)
;; ()
; Slice the two blocks.
(define slice (solid:slice block1 block2))
;; slice
; OUTPUT Original Sliced Blocks

; Remove the blocks to see the slice.
(entity:delete (list block1 block2))
;; ()
; OUTPUT Resulting Slice
```

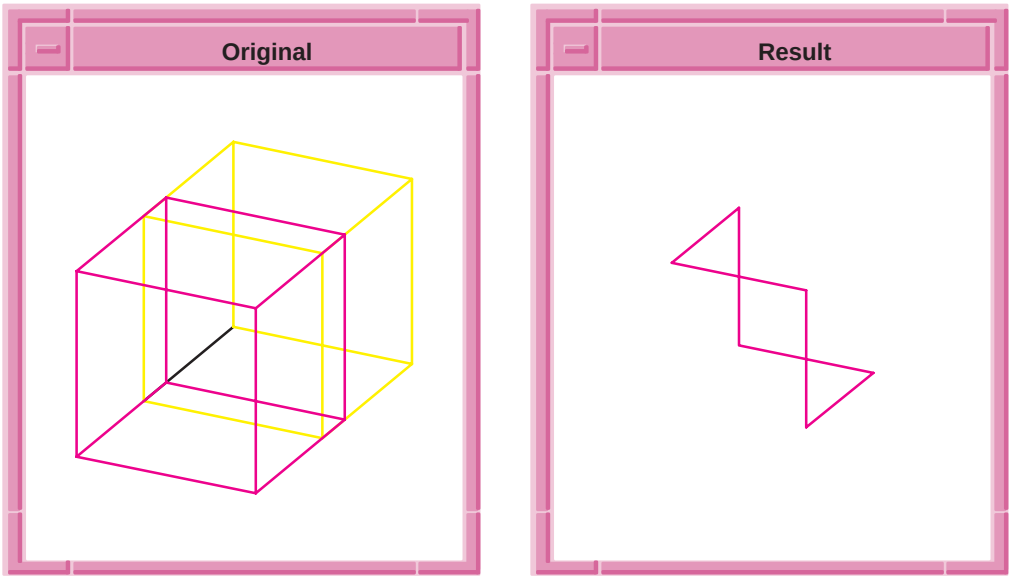

Figure 2-27. solid:slice

solid:split

Scheme Extension: Booleans

Action: Splits all periodic faces of a body along the seams.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_split_periodic_faces

Syntax: `(solid:split body [acis–opts])`

Arg Types:	body	body
	acis–opts	acis–options

Returns: body

Errors: None

Description: This command ensures that a body is well formed by splitting all periodic faces of the body along *u*, *v*, or both. The periodic faces of the body are split at the lower parameter limit, and halfway between the lower and upper parameter limit. Analytic surfaces are handled, but periodic spline surfaces are not.

body is an input body.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; solid:split
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 25 25 15)
    (position 15 –20 15) 20))
;; cyl1
; OUTPUT Original

; Split the cylinder.
(define split (solid:split cyl1))
;; split
; OUTPUT Result
```

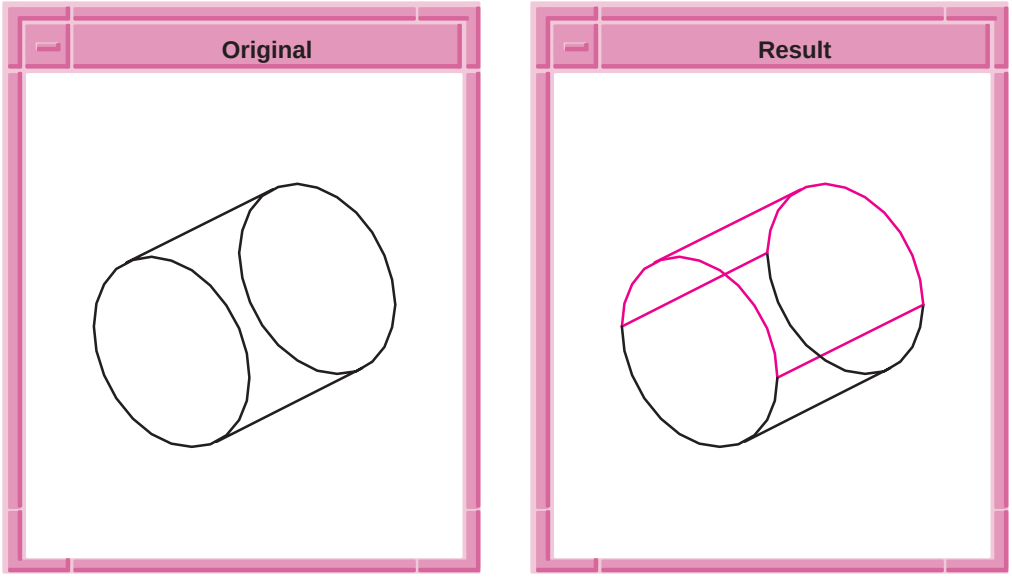

Figure 2-28. solid:split

solid:stitch

Scheme Extension: Booleans

Action: Joins two bodies (faces) along edges or vertices of identical geometry.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_stitch

Syntax: `(solid:stitch body1 body2 [split] [acis–opts])`

Arg Types:

body1	body
body2	body
split	boolean
acis–opts	acis–options

Returns: body

Errors: None

Description: Joins two face bodies, body1 and body2, along edges or vertices that are identical. Stitching only operates on faces, not on wires, and only stitches faces to faces. No faces are deleted in order to complete the stitch function. If wires exist in one of the bodies being stitched, but do not participate in the stitch (i.e., they do not coincide with edges in the other body), they will transfer to the resulting body.

The argument body1 is returned as the resulting body. body2, is deleted unless it is the same as body1. For example, a body might need internal stitching. One can stitch body1 to body1, and body1 is not deleted.

If the split option is not specified, the edges must be identical along their entire length, otherwise stitch splits edges along identical edge subregions. Single–sided faces that are stitched together to form a two–manifold edge must have compatible orientations. This is not required for double–sided faces.

Unlike extensions such as solid:unite, solid:stitch is *not* a Boolean operation. Stitching is simpler than a Boolean because it avoids face–face intersections and the evaluation of lump and shell containments.

body1 and body2 are input bodies which needs to be united along edges or vertices.

split is an option to specify whether, the edges are identical along their entire length.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; solid:stitch
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Create a list of faces from the block.
(define faces (entity:faces block1))
;; faces
; Uncover a face.
(define uncover (face:uncover (list–ref faces 5)))
;; uncover
; Copy the open block.
(define block2
    (entity:copy block1))
;; block2
; Flip and position the block to match open faces.
(define transform1 (entity:transform block2
    (transform:rotation
    (position 0 0 0) (gvector 0 0 1) 180)))
;; transform1
(define transform2 (entity:transform block2
    (transform:translation
    (gvector 40 0 0))))
;; transform2
; Join block1 and block2 to form one lump. (Note:
; you can run (entity:check stitch) to check results.
(define stitch (solid:stitch block1 block2))
;; stitch
```

solid:subtract

Scheme Extension: Booleans

Action: Subtracts a list of solids from a solid.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_subtract

Syntax: (**solid:subtract** body–1 body–2 [...body–n] [acis–opts])

Arg Types:		
	body–1	body
	body–2	body
	body–n	body
	acis–opts	acis–options

Returns: body

Errors: None

Description: Subtracts a list of solids from a solid. All bodies must refer to solid bodies. body–2 through body–n are subtracted from body–1. This extension returns the name of body–1 and deletes body–2 through body–n.

body–1, body–2, body–n are input bodies.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; solid:subtract
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 10 10 10)))
;; block1
; Set color for block1.
(entity:set–color block1 2)
;; ()
; Create another solid block.
(define block2
    (solid:block (position –12 –12 –12)
    (position 30 30 30)))
;; block2
; Set color for block2.
(entity:set–color block2 3)
;; ()
; OUTPUT Original

; Subtract block 1 from 2.
(define subtract (solid:subtract block1 block2))
;; subtract
; OUTPUT Result
```

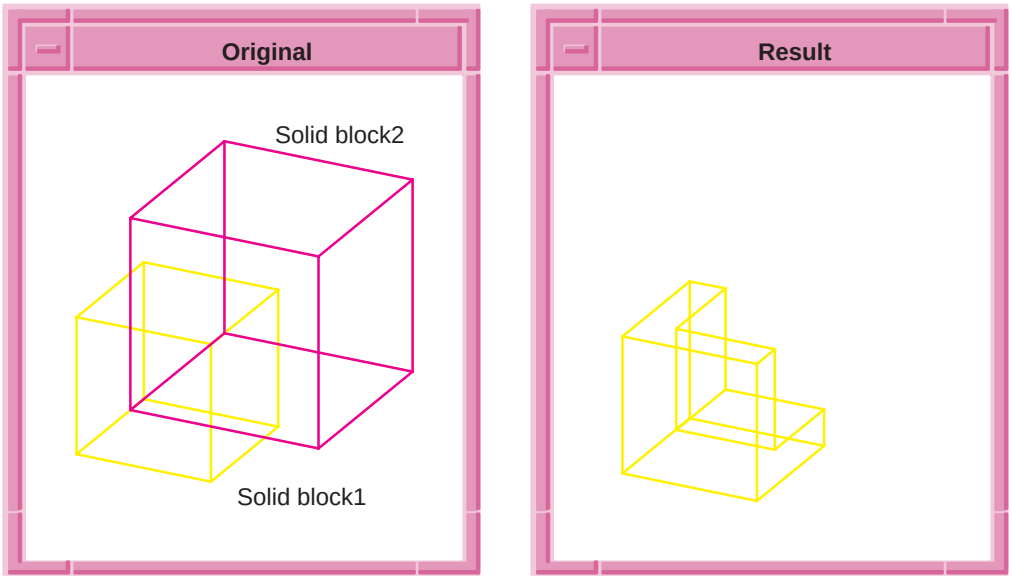

Figure 2-29. solid:subtract

solid:unhook–wire–edge

Scheme Extension: Booleans

Action: Unhooks an edge belonging to a wire from a body and returns a new wire-body.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_unhook_wire_edge

Syntax: `(solid:unhook–wire–edge edge [acis–opts])`

Arg Types:

edge	edge
acis–opts	acis–options

Returns: body

Errors: None

Description: This extension unhooks an edge belonging to a wire from a body and returns a new wire-body.

edge is an input edge belonging to a wire from a body.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; solid:unhook–wire–edge
; Define a wire
(define wire (wire–body:points (list (position 0 0 0)
    (position 40 0 0) (position 40 40 0)
    (position 40 40 40))))
;; wire
; wire has 3 edges
(entity:check wire)
;; checked:
;;        1 lumps
;;        1 shells
;;        1 wires
;;        0 faces
;;        0 loops
;;        3 coedges
;;        3 edges
;;        4 vertices
;; ()
(define edge0 (car (entity:edges wire)))
;; edge0
; unhook the edge
(define new_wire (solid:unhook–wire–edge edge0))
;; new_wire
; new_wire has 1 edge
(entity:check new_wire)
;; checked:
;;        1 lumps
;;        1 shells
;;        1 wires
;;        0 faces
;;        0 loops
;;        1 coedges
;;        1 edges
;;        2 vertices
;; ()
; original wire has 2 edges
(entity:check wire)
;; checked:
;;        1 lumps
;;        1 shells
;;        1 wires
```

```
;;        0 faces
;;        0 loops
;;        2 coedges
;;        2 edges
;;        3 vertices
;; ()
```

solid:unite

Scheme Extension: Booleans

Action: Unites two or more solids.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_unite

Syntax: (**solid:unite** body–1 body–2 [...body–n] [acis–opts])

Arg Types:		
	body–1	body
	body–2	body
	body–n	body
	acis–opts	acis–options

Returns: body

Errors: None

Description: Combines two solids. All bodies must refer to solid bodies. body–2 through body–n are united with body–1. This extension returns the name of body–1 and deletes body–2 through body–n.

body–1, body–2, body–n are input bodies.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; solid:unite
; Create a solid block.
(define block1
    (solid:block (position –10 –10 –10)
    (position 40 40 40)))
;; block1
; Set color for block1.
(entity:set–color block1 2)
;; ()
; Create another solid block.
(define block2
    (solid:block (position –30 –30 –30)
    (position 0 0 0)))
;; block2
; Set color for block2.
(entity:set–color block2 3)
;; ()
; OUTPUT Blocks 1 and 2

; Unite solid block 1 and 2.
(define unite1 (solid:unite block1 block2))
;; unite1
; OUTPUT United blocks 1 and 2

; Create another solid block.
(define block3
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block3
; Set color for block3.
(entity:set–color block3 4)
;; ()
; OUTPUT Additional block 3

; Unite the previous entity with solid block 3.
(define unite2 (solid:unite block1 block3))
;; unite2
; OUTPUT All blocks united
```

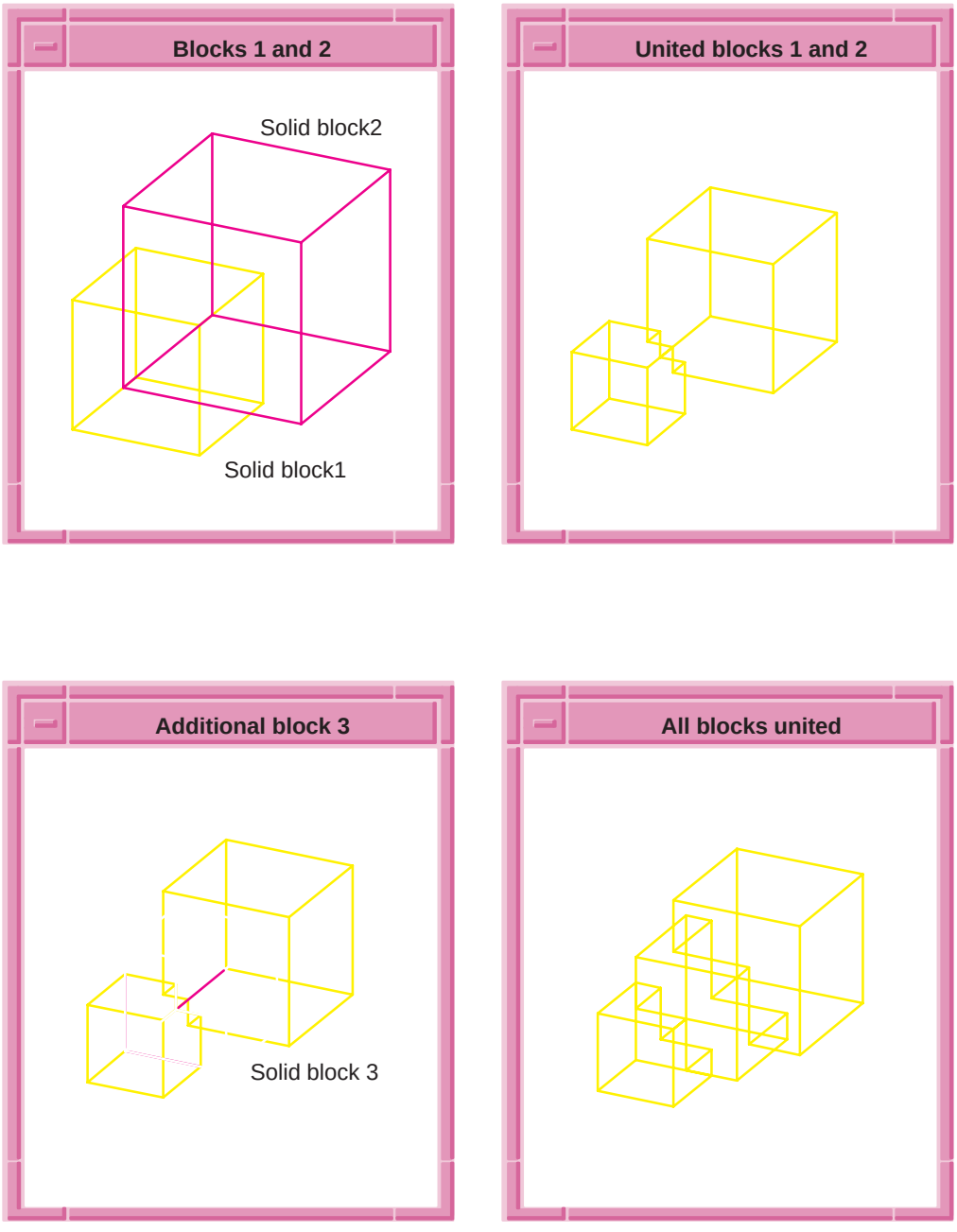

Figure 2-30. solid:unite

solid:unstitch–nm

Scheme Extension: Booleans

Action: Decomposes the input body along nonmanifold edges and vertices.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_unstitch_nonmani

Syntax: `(solid:unstitch–nm body1 [acis–opts])`

Arg Types:

body1	body
acis–opts	acis–options

Returns: body | body ...

Errors: None

Description: Decomposes the input body along nonmanifold edges and vertices, duplicating edges and vertices and making each decomposed region a separate lump in the output body. All lumps in the body become manifold unless they had self–nonmanifold edges. The command also inserts all groups of wire edges into the body's wire list if they weren't already there. Returned list of bodies contains original body separated into 3d bodies, sheet bodies, laminar bodies, and wire bodies, as present. Refer to api_unstitch_nonmani for additional information.

body1 is an input body.

acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; solid:unstitch–nm
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Create a list of faces from the block.
(define faces (entity:faces block1))
;; faces
; Uncover a face.
(define uncover (face:uncover (list–ref faces 5)))
;; uncover
; Copy the open block.
(define block2
    (entity:copy block1))
;; block2
; Flip and position the block to match open faces.
(define transform1 (entity:transform block2
    (transform:rotation
    (position 0 0 0) (gvector 0 0 1) 180)))
;; transform1
(define transform2 (entity:transform block2
    (transform:translation
    (gvector 40 0 0))))
;; transform2
; Join block1 and block2 to form one lump. (Note:
; you can run (entity:check stitch) to check results.
(define stitch (solid:stitch block1 block2))
;; stitch
; Unstitch the newly stitched block.
(define unstitch (solid:unstitch–nm stitch))
;; unstitch
```

tolerant:detect–short–edges

Scheme Extension: Model Topology, Tolerant Modeling

Action: Returns all edges from either a body or a wire that are shorter in length than the specified tolerance.

Filename: bool/bool_scm/sliver_scm.cxx

APIs: api_detect_short_edges

Syntax: (**tolerance:detect–short–edges** entity [tolerance] [replace])

Arg Types:	entity tolerance replace	entity real boolean

Returns: entity ...

Errors: None

Description: Returns all edges from the entity passed in that are shorter then the specified tolerance. If no tolerance is specified, SPAresfit is used by default. The third optional argument specifies if detected short edges should be automatically replaced. By default, short edges are not replaced.

entity is an input entity.

tolerance is the specified tolerance for an input entity.

replace is an optional argument specifies short edges should be automatically replaced, if detected.

Limitations: None

Example:

```
; tolerant:detect-short-edges
; Define a simple skin that results in a
; short edge being created
(define edge1 (wire-body (list
    (edge:linear (position 0 0 0)
    (position 1 0 0)))))
;; edge1
(define edge2 (wire-body (list (edge:linear
    (position 0.5 1 0) (position 0.500001 1 0)))))
;; edge2
    (define skin_ops (skin:options "arc_length"
    #f "no_twist" #t "align" #t "solid" #f))
;; skin_ops
(define skin (sheet:skin-wires
    (list edge1 edge2) skin_ops))
;; skin
; Check the body before the short edge replacement
(entity:check skin)
; checked:
;       1 lumps
;       1 shells
;       0 wires
;       1 faces
;       1 loops
```

```
;       4 coedges
;       4 edges
;       4 vertices
;; ()
; Detect and replace the short edge
(define short_edge
    (tolerant:detect-short-edges skin #t))
;; short_edge
; Validate the remaining body
(entity:check skin)
;       1 lumps
;       1 shells
;       0 wires
;       1 faces
;       1 loops
;       3 coedges
;       3 edges
;       3 vertices
;; ()
(define v (entity:tvertices skin))
;; v
(define tol_edge (tolerant:report skin))
; Vertex tolerance = 4.472138e-007
; No tolerant edges
;; tol_edge
```

tolerant:detect–sliver–faces

Scheme Extension: Model Topology, Tolerant Modeling

Action: Returns all sliver faces from a body whose maximum distance among the edges is smaller than the given tolerance.

Filename: bool/bool_scm/sliver_scm.cxx

APIs: api_detect_sliver_faces

Syntax:
```
(tolerant:detect-sliver-faces entity
    [tolerance] [replace=false])
```

Arg Types:		
	entity	entity
	tolerance	real
	replace	boolean

Returns: entity ...

Errors: None

Description: Returns all sliver faces from the entity passed in whose maximum distance among the edges is smaller than the specified tolerance. If no tolerance is specified, the lesser of (10*SPAresfit) and (minimum side of bounding box 1250) is used by default. If the third argument replace is set TRUE, the detected removable sliver faces will be automatically replaced with tolerant edges.

When the option replace is set TRUE(#t), this command first detects all the short edges and replaces them with tvertices. Next, it attempts to detect all the 2– or 3-edge sliver faces. However, the extension tolerant:replace-face-with-tedge does not attempt to replace the short edges with tvertices. Therefore, the sliver faces detected by this command may not be replaced with tedges with the extension tolerant:replace-face-with-tedge.

entity is an input entity.

tolerance is the specified tolerance for an input entity.

replace is an optional argument specifies short edges should be automatically replaced, if detected.

Limitations: None

Example:
```
; tolerant:detect-sliver-faces
; No example available at this time.
```

tolerant:replace–edge–with–tvertex

Scheme Extension: Model Topology, Tolerant Modeling

Action: Replaces a list of edges with tolerant vertices.

Filename: bool/bool_scm/sliver_scm.cxx

APIs: api_replace_edge_with_tvertex

Syntax: `(tolerant:replace-edge-with-tvertex edge-list)`

Arg Types: edge–list entity ...

Returns: entity ...

Errors: None

Description: The argument edge–list specifies a list of edges that will be converted to tolerant vertices.

The extension returns a list of tolerant vertices that were created during the replacement. This extension does not take edge lengths into consideration. It simply replaces all edges specified unless the edges are in a closed loop. If the edges are closed, replacement is not performed. If replacing short edges is desired, the extension tolerant:detect-short-edges with its replace option set to TRUE is recommended.

edge–list specifies a list of edges that will be converted to tolerant vertices.

Limitations: None

Example:

```
; tolerant:replace–edge–with–tvertex
; Define a wire body with an edge to be replaced
(define wire (wire–body
    (list (edge:linear
        (position 0 0 0)
        (position 3 0 0)))))
;; wire
(define edge1 (edge:linear
               (position 3 0 0)
               (position 3.1 0 0)))
;; edge1
(define edge2 (edge:linear
               (position 3.1 0 0)
               (position 5 0 0)))
;; edge2
; Replace the middle edge with a tvertex.
; First use tolerant–detect–short–edges with
; a specified tolerance to detect it.
(define short_edge
    (tolerant:detect–short–edges wire 0.2))
;; short_edge
(define replace
    (tolerant:replace–edge–with–tvertex  short_edge))
;;
; Check the resulting body and tvertex
(entity:check wire)
;;
(entity:tvertices wire)
;;
(tolerant:report wire)
;;
```

tolerant:replace–face–with–tedge

Scheme Extension: Model Topology, Tolerant Modeling

Action: Replaces a 2– or 3-edge face with a tolerant edge.

Filename: bool/bool_scm/sliver_scm.cxx

APIs: api_replace_face_with_tedge

Syntax: `(tolerant:replace–face–with–tedge face)`

Arg Types: face face

Returns: entity ...

Errors: None

Description: This replaces a face having a single loop of two or three edges with a tolerant edge. The face is removed, an edge of the face is converted into a tolerant edge, and other subsequent topological manipulations are performed.

Since the Scheme command tolerant:detect-sliver-faces detects all the sliver faces regardless of whether or not they can be replaced with tedges, the sliver faces detected cannot not be replaced with tedges with this command. To replace sliver faces in an entity appropriately, please use tolerant:detect-sliver-faces with the replace option set to TRUE.

face is an input face.

Limitations: None

Example:

```
; tolerant:replace–face–with–tedge
; No example available at this time.
```

wire:clean

Scheme Extension: Booleans

Action: Removes the attributes and extra coedges present on a wire body generated by the section or slice operation.

Filename: bool/bool_scm/bool_scm.cxx

APIs: api_clean_wire

Syntax: `(wire:clean wire [acis–opts])`

Arg Types:	wire acis–opts	wire–body acis–options
Returns:	wire-body	
Errors:	None	
Description:	This extension removes the attributes and extra coedges present on a wire body generated by the section or slice operation. The optional acis–opts contains parameters for versioning and journaling. wire is an input wire body. acis–opts contains parameters for versioning and journaling.	
Limitations:	None	

Example:

```
; wire:clean
; Define a solid sphere
(define b1 (solid:sphere (position 0 0 0) 1))
;; b1
; Define a solid block
(define b2 (solid:block (position 0 0 0) (position 1
1 1)))
;; b2
; compute the slice keeping all attributes
(define sli1 (solid:inter–graph b1 b2))
;; sli1
; check that we have the attributes
(entity:debug sli1 3)
;;   1 body record,    40 bytes
;;  21 attrib records,    1460 bytes
;;   1 wire record,    48 bytes
;;   1 transform record,    144 bytes
;;  12 coedge records,    672 bytes
;;   3 edge records,    240 bytes
;;   3 vertex records,    108 bytes
;;   3 curve records,    672 bytes
;;   3 point records,    216 bytes
;; Total storage 3600 bytes
;; ”wire body”
(wire:clean sli1)
;; #[entity 3 1]
; check that only display_attribute and id_attribute
remain
(entity:debug sli1 3)
;;   1 body record,    40 bytes
;;   2 attrib records,    108 bytes
;;   1 wire record,    48 bytes
;;   1 transform record,    144 bytes
;;   3 coedge records,    168 bytes
;;   3 edge records,    240 bytes
;;   3 vertex records,    108 bytes
;;   3 curve records,    672 bytes
;;   3 point records,    216 bytes
;; Total storage 1744 bytes
;; ”wire body”
```