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:sel–unite

Scheme Extension: Graph Theory, Booleans

Action: Unites two bodies.

Filename: sbool/sbool_scm/selective_scm.cxx

APIs: api_boolean, api_selective_unite

Syntax:
```
(bool:sel–unite blank–body tool–body
    option1 option2)
```

Arg Types:

blank–body	entity
tool–body	entity
option1	entity | #t
option2	entity | #t

Returns: entity

Errors: None

Description: This scheme extension performs a union operation between two bodies. The portions of bodies to keep is controlled by option1 and option2. The blank body is returned and the tool body is deleted.

blank–body is an input entity.

tool–body is an input entity.

option1 controls the portions of the body to be kept.

option2 controls the portions of the body to be kept.

Limitations: None

Example:

```
; bool:sel–unite
; Define a couple of solids and then do a
; selective boolean.
(define c1
    (solid:cylinder
    (position 0 0 –1)
    (position 0 0 1) 5))
;; c1
(define c2
    (solid:cylinder
    (position 0 0 –1)
    (position 0 0 1) 4.5))
;; c2
(define subtract (bool:subtract c1 c2))
;; subtract
(define b
    (solid:block
    (position –6 –6 –0.25)
    (position 6 1 0.25)))
;; b
(define pnt (position 0 0 0))
;; pnt
(define unite (bool:sel–unite c1 b #t pnt))
;; unite
```

bool:select1

Scheme Extension: Booleans

Action: Creates a graph for the first stage of selective Booleans from a tool body and a blank body.

Filename: sbool/sbool_scm/selective_scm.cxx

APIs: api_selective_boolean_stage1

Syntax:	`(`**`bool:select1`**` blank–body tool–body [acis–opts])`	
Arg Types:	blank–body	entity
	tool–body	entity
	acis–opts	acis–options
Returns:	graph	
Errors:	None	

Description: This Scheme extension creates a graph structure (e.g., graph theory) from the blank–body entity and the tool–body entity. Using Cellular Topology, distinctive cells of the blank–body and the tool–body become vertices of the graph. Overlapping portions of the cells become edges in the graph. Once the graph has been created, further graph theory operations can be performed to obtain a more desirable graph. This is then used as input to the second stage of selective Booleans, bool:select2.

blank–body is an input entity.

tool–body is an input entity.

The optional acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:select1
; Create a blank body
(define blank (solid:block (position 0 0 0)
    (position 25 10 10)))
;; blank
; blank => #[entity 2 1]
(define b2 (solid:block (position 5 0 0)
    (position 10 5 10)))
;; b2
(define b3 (solid:block (position 15 0 0)
    (position 20 5 10)))
;; b3
(define subtract (solid:subtract blank b2))
;; subtract
(define subtract2 (solid:subtract blank b3))
;; subtract2
; Create the tool body
(define tool (solid:cylinder
    (position -5 2.5 5) (position 30 2.5 5)1))
;; tool
; Change the color of the tool for viewing.
(entity:set-color tool 3)
;; ()
; OUTPUT Original

(define g (bool:select1 tool blank))
;; g
; OUTPUT Result
```

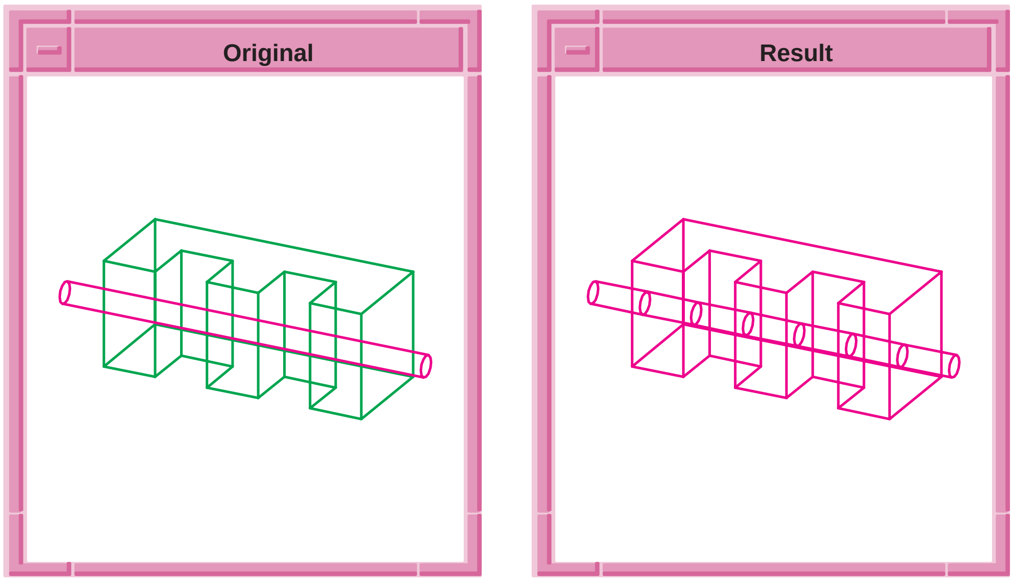


Figure 2-1. bool:select1

bool:select2

Scheme Extension: Booleans

Action:	Completes the selective Boolean process for the cells selected.
Filename:	sbool/sbool_scm/selective_scm.cxx
APIs:	api_selective_boolean_stage2
Syntax:	`(`**`bool:select2`**` in-body in-cells [acis-opts])`

Arg Types:

in–body	entity
in–cells	graph | (entity ...)
acis–opts	acis–options

Returns: entity

Errors: None

Description: This extension modifies the in–body body based on either the results of graph theory or a Cellular Topology cell list in in–cells. When in–cells contains cell entities, these are the only entities that should be kept. When in–cells is a graph, the graph could be the result of extensive graph theory operations. However, the resulting graph still maps to cells in the body and represents the cells to keep.

The mapping of cells in a graph to entities will not be the same from execution to execution.

in–body is an input entity.

in–cells is an input entity that contains either cell entities or a graph.

The optional acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:select2
; Create a selective boolean example.
(define blank (solid:block (position 0 0 0)
    (position 25 10 10)))
;; blank
; blank => #[entity 2 1]
(define b2 (solid:block (position 5 0 0)
    (position 10 5 10)))
;; b2
(define b3 (solid:block (position 15 0 0)
    (position 20 5 10)))
;; b3
(define subtract (solid:subtract blank b2))
;; subtract
(define subtract2 (solid:subtract blank b3))
;; subtract2
(define tool (solid:cylinder
    (position –5 2.5 5) (position 30 2.5 5)1))
;; tool
; Change the color of the tool for viewing.
(entity:set–color tool 3)
;; ()
(define g (bool:select1 blank tool))
;; g
; CAREFUL: The mapping of cell names to entities
; may not be the same each time this is run.
; Find the cells in the blank created by graph.
(define cells (entity:cells blank))
;; cells
; Put together list of things to keep.
; Leave out 2 and 6.
(define in–cells (list
    (list–ref cells 0)
    (list–ref cells 1)
    (list–ref cells 3)
    (list–ref cells 4)
    (list–ref cells 5)
    (list–ref cells 7)))
;; in–cells
; OUTPUT Original

(define select (bool:select2 blank in–cells))
;; select
; OUTPUT Result
```

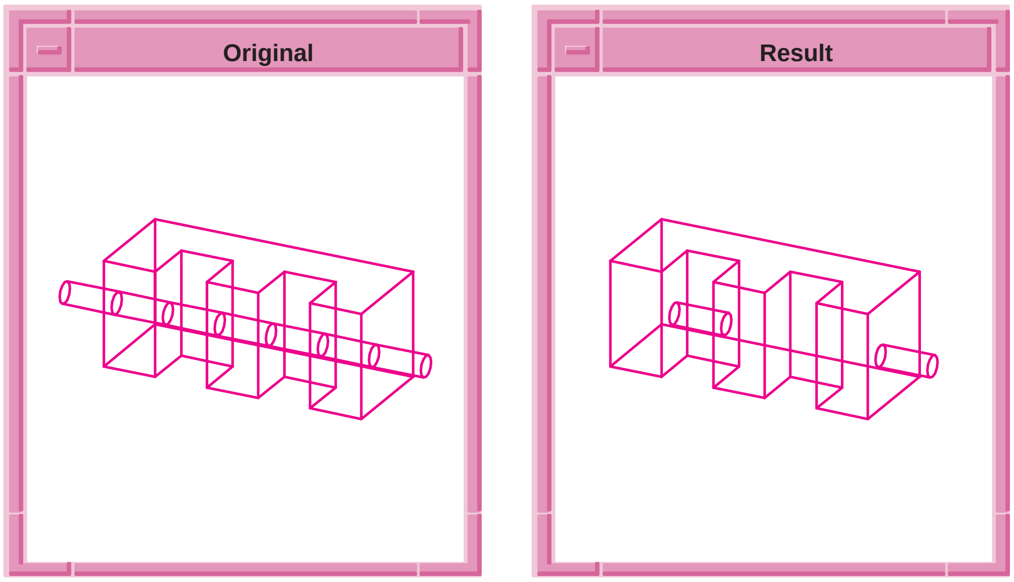


Figure 2-2. bool:select2

bool:tube

Scheme Extension: Booleans

Action: Performs a selective Boolean operation, as specified in the given option.

Filename: sbool/sbool_scm/selective_scm.cxx

APIs: api_boolean_tube_body

Syntax: `(`**`bool:tube`**` blank tool start end opt [acis-opts])`

Arg Types:

blank	body
tool	body
start	position | (position)
end	position | (position)
opt	string
acis–opts	acis–options

Returns: entity

Errors: None

Description: The bool:tube performs a Boolean operation on the given blank and tool bodies. The type of operation performed is specified by the opt of type Tube–Options. The start and end regions for the operation can be defined using positions or arrays of positions.

The purpose of this is to make selective Booleans more general so they can be used by sweeping and other modeling operations.

The API api_boolean_tube_body, which bool:tube uses, takes two bodies (blank and tube) as input and calls the selective Booleans stage 1 once and stage 2 the number_of_options times. Each time these selective Booleans are called, this function automatically sets up the subgraph from a set of options and start and end conditions.

blank is an input body.

tool is an input body.

start gives the start region for the operation in terms of position or array of positions.

end gives the end region for the operation in terms of position or array of positions.

opt specifies the type of operation performed.

The optional acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; bool:tube – Example 1
; Create a block
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
; b => #[entity 2 1]
; set new color for entity b
(entity:set–color b 3)
;; ()
; Create solid cylinder 1
(define c1 (solid:cylinder (position –5 5 2)
    (position 15 5 2)1))
;; c1
;set new color for entity c1
(entity:set–color c1 4)
;; ()
; Create solid cylinder 2
(define c2 (solid:cylinder (position –5 5 8)
    (position 15 5 8)1))
;; c2
; set new color for entity c2
(entity:set–color c2 5)
;; ()
; Combine solid cylinders 1 and 2
(define c (solid:unite c1 c2))
;; c
; set new color for entity c
(entity:set–color c 6)
;; ()
; OUTPUT Original

; Define the start position
(define start (list (list–ref (entity:faces c) 1)
    (list–ref (entity:faces c) 4)))
;; start
; Define the end position
(define end (list (list–ref (entity:faces c) 2)
    (list–ref (entity:faces c) 5)))
;; end
```

```
(define opt1 (tube:options ”keep_law” ”x=0”
    ”bool_type” ”UNITE”))
;; opt1
; View details of opt1
opt1
;; #[Tube_Options ”keep_law” X=0 ”keep_branches”
    #f”unite”]
; Define an entity to be solid block ”b” with
; entities ”b” and ”c” removed from the interior
; and top
(define d (bool:tube b c start end opt1))
;; d
; OUTPUT Result
```

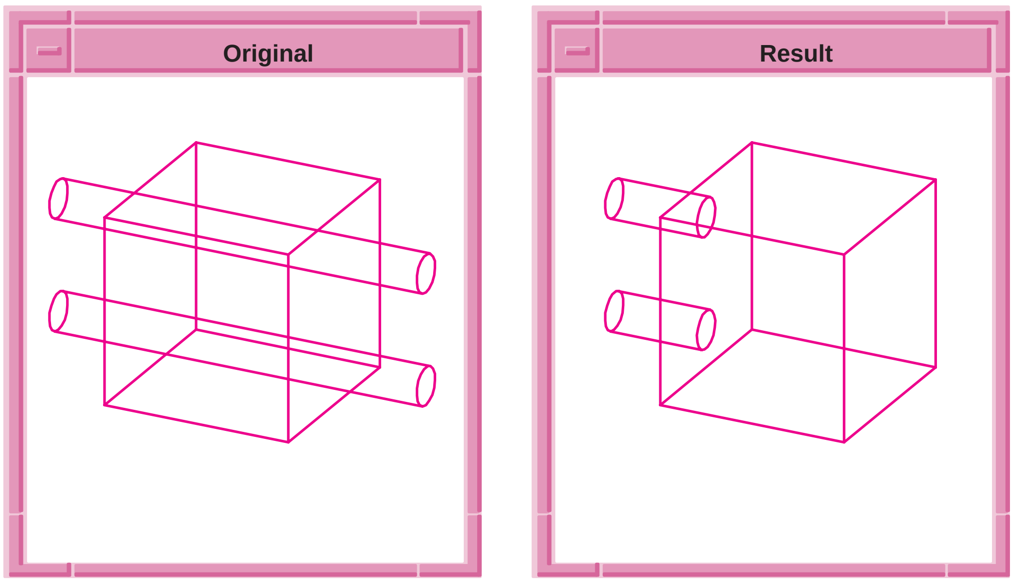


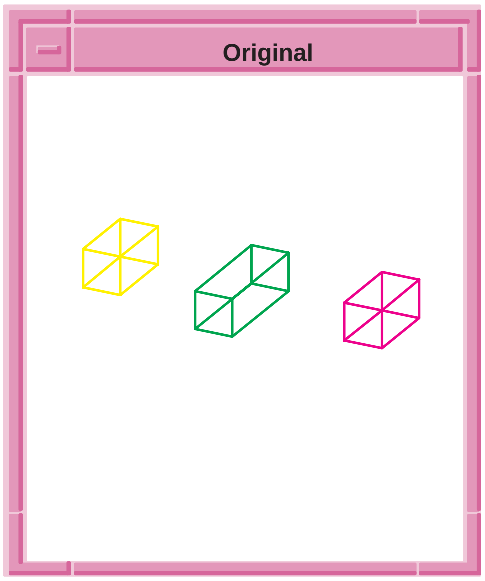
Figure 2-3. bool:tube (Example 1)

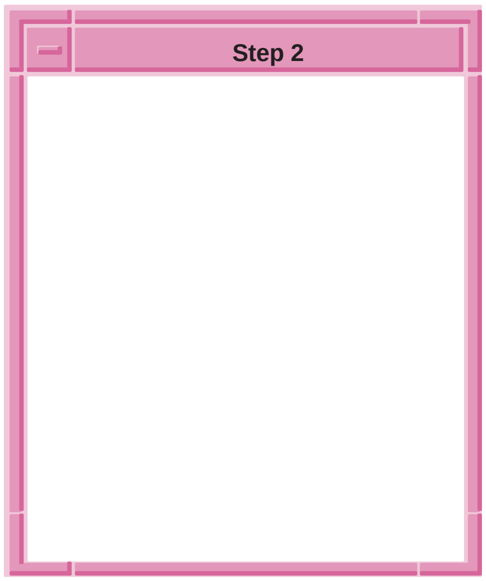
```
; bool:tube – Example 2
; sel_bool unite operations with
; descriptions:
; Delete all entities from the active area.
(part:clear)
;; #t
; create a solid block (vertical rectangle).
(define b1 (solid:block (position –40 0 0)
    (position 40 10 10)))
;; b1
; set new color for entity b1
(entity:set–color b1 4)
;; ()
; create solid block 2.
(define b2 (solid:block (position –40 10 0)
    (position –30 30 10)))
;; b2
; set new color for entity b1
(entity:set–color b2 5)
;; ()
; create solid block 3.
(define b3 (solid:block (position 30 10 0)
    (position 40 30 10)))
;; b3
; set new color for entity b1
(entity:set–color b3 6)
;; ()
; create solid block 4 (center block).
(define b4 (solid:block (position –5 0 0)
    (position 5 30 10)))
;; b4
; set new color for entity b1
(entity:set–color b4 2)
;; ()
; OUTPUT Original

; create a blank.
; Unite the 4 block entities.
(define blank (bool:unite b1 b2 b3 b4))
;; blank
; OUTPUT Step 2
; create another solid block.
(define tool (solid:block (position –45 15 0)
    (position 45 25 10)))
;; tool
; OUTPUT Step 3

; define the faces
(define faces (entity:faces tool))
;; faces
; Define new color for ”faces”
(entity:set–color faces 6)
;;()
; define the start position
(define start (list–ref faces 3))
;; start
; define the end position
(define end (list–ref faces 5))
;; end
; x1 (x) = order of this vertex
; x2 (y) = largest order in component
; x3 (z) = TRUE if from tool
; x4 = TRUE if from blank
; x5 = TRUE if start cell
; x6 = TRUE if end cell
; In this example, there are 4 possible cells that
; could be included in the unite operation.
; The first cell is 0, the last is 3.
; x1 takes on all possible values, 0, 1, 2, and 3.
; x2 is 3.
; x3 is always TRUE if we are doing a unite.
; x4 is always FALSE if we are doing a unite.
; x5 is TRUE if x1 is 0.
; x6 is TRUE if x1 is 3.
```

```
(define opts1 (tube:options ”keep_law” ”x!=x2”
    ”bool_type” ”u”))
;; opts1
; keep all but last cell
(define opts2
    (tube:options ”keep_law” ”x5” ”bool_type” ”u”))
;; opts2
; keep all start cells
(define opts3 (tube:options ”keep_law” ”x=2 or
    x6””bool_type” ”u”))
;; opts3
; keep the second and last cells
(bool:tube blank tool start end opts3)
;; Try opts1, opts2, and opts3.
; OUTPUT Result
```





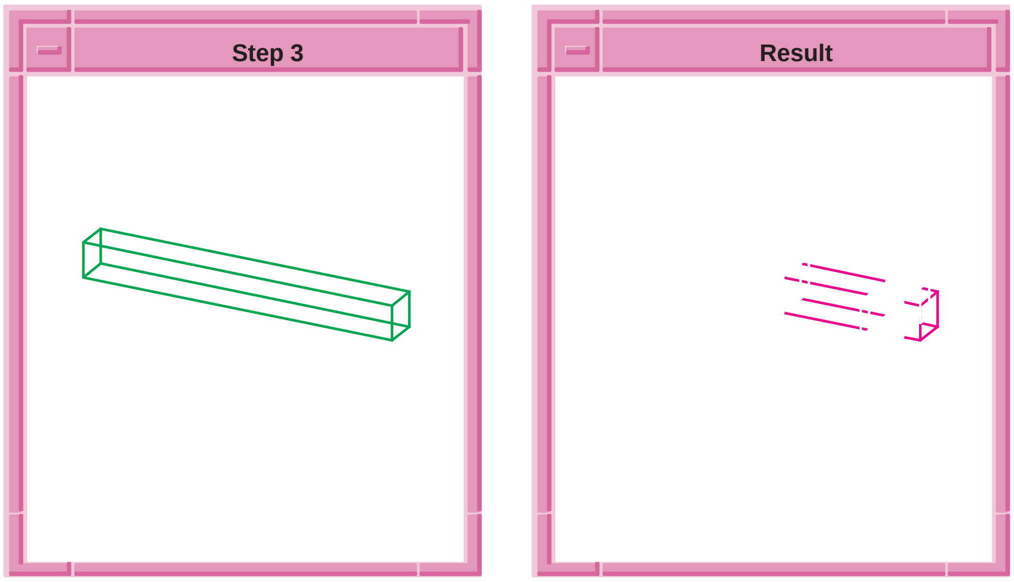


Figure 2-4. bool:tube (Example 2)

graph

Scheme Extension: Graph Theory

Action:	Creates a graph used in graph theory.
Filename:	sbool/sbool_scm/selective_scm.cxx
APIs:	api_create_graph_from_cells, api_create_graph_from_edges, api_create_graph_from_faces, api_pm_lookup_entity

Syntax:

```
(graph {graph-string | face-ent-list | cell-list}
    [acis-opts])
```

Arg Types:

graph–string	string
face–ent–list	entity | (entity ...)
cell–list	cell | (cell ...)
acis–opts	acis–options

Returns: graph

Errors: None

Description: This extension creates a graph that can be used for graph theory operations and selective Booleans. When creating a graph using a graph–string, the entire string must be within double quotation marks. The vertex names within the string can be any number of characters long without spaces or dashes (–). Edges between vertices are indicated by a dash. The items in the list of graph items within the string are separated by spaces.

When creating a graph using face–ent–list, the entities within the entity list must be face entities. The face entities become vertices of the graph. The ACIS topology is analyzed to determine which faces are connected. The connections between faces become edges (or dashes) of the graph.

When creating a graph using cell–list, the list must contain cells from Cellular Topology. The ACIS topology is analyzed to determine which cells are connected. The connections between cells become edges (or dashes) of the graph.

graph–string is an input string that can be used for creating a graph.

face–ent–list is an input entity that can be used for creating a graph.

cell–list is an input cell that can be used for creating a graph.

The optional acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; graph
; Create a simple example
(define g1 (graph ”me–you us–them”))
;; g1
; Create an example using entities.
(define b1 (solid:block (position –5 –10 –20)
    (position 5 10 15)))
;; b1
(define faces1 (entity:faces b1))
;; faces1
; Turn the block faces into vertices of the graph.
(define g3 (graph faces1))
;; g3
```

graph:subgraph–2dcell

Scheme Extension: Graph Theory

Action: Returns a subgraph containing only the vertices that are 2D cells.

Filename:	sbool/sbool_scm/selective_scm.cxx
APIs:	api_subgraph_2dcell
Syntax:	(**graph:subgraph–2dcell** whole–graph [acis–opts])

Arg Types:

whole–graph	graph
acis–opts	acis–options

Returns: graph

Errors: None

Description: Refer to Action.

whole–graph is an input argument.

The optional acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:

```
; graph:subgraph–2dcell
; Create entities
(define b1 (solid:block (position –40 0 0)
    (position 40 10 10)))
;; b1
(define b2 (solid:block (position 30 –10 –10)
    (position 45 20 20)))
;; b2
(define face1 (face:planar–disk (position –10 0 0)
    (gvector 1 0 0) 20))
;; face1
(define sheet1 (sheet:face face1))
;; sheet1
(define sh2d–1 (sheet:2d sheet1))
;; sh2d–1
(define face2 (face:planar–disk (position –30 0 0)
    (gvector 1 0 0) 20))
;; face2
(define sheet2 (sheet:face face2))
;; sheet2
(define sh2d–2 (sheet:2d sheet2))
;; sh2d–2
(define face3 (face:planar–disk (position –20 15 0)
    (gvector 0 1 0) 30))
;; face3
(define sheet3 (sheet:face face3))
```

```
;; sheet3
(define sh2d–3 (sheet:2d sheet3))
;; sh2d–3
(define e (bool:nonreg–unite b1 b2 sh2d–1 sh2d–2
    sh2d–3))
;; e
(define plane–pos (position –20 0 0))
;; plane–pos
(define plane–normal (gvector 1 0 0))
;; plane–normal
(define imprint–face (face:planar–disk
    plane–pos plane–normal 50))
;; imprint–face
(define imprint–sheet (sheet:face imprint–face))
;; imprint–sheet
(define sh2d–4 (sheet:2d imprint–sheet))
;; sh2d–4
(define sol–im1 (solid:imprint e imprint–sheet))
;; sol–im1
(cell:attach e)
;; (#[entity 12 1] #[entity 13 1] #[entity 14 1]
;; #[entity 15 1] #[entity 16 1] #[entity 17 1]
;; #[entity 18 1] #[entity 19 1] #[entity 20 1]
;; #[entity 21 1])
(define g (graph (entity:cells e)))
;; g
(define g–2d (graph:subgraph–2dcell g))
;; g–2d
(define g–3d (graph:subgraph–3dcell g))
;; g–3d
```

graph:subgraph–3dcell

Scheme Extension: Graph Theory

Action: Returns a subgraph containing only the vertices that are 3D cells.

Filename: sbool/sbool_scm/selective_scm.cxx

APIs: api_subgraph_3dcell

Syntax: (**graph:subgraph–3dcell** whole–graph [acis–opts])

Arg Types:

whole–graph	graph
acis–opts	acis–options

Returns: graph

Errors: None

Description: Refer to Action.

whole–graph is an input argument.

The optional acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:
```
; graph:subgraph–3dcell
; Create entities
(define b1 (solid:block (position –40 0 0)
    (position 40 10 10)))
;; b1
(define b2 (solid:block (position 30 –10 –10)
    (position 45 20 20)))
;; b2
(define face1 (face:planar–disk (position –10 0 0)
    (gvector 1 0 0) 20))
;; face1
(define sheet1 (sheet:face face1))
;; sheet1
(define sh2d–1 (sheet:2d sheet1))
;; sh2d–1
(define face2 (face:planar–disk (position –30 0 0)
    (gvector 1 0 0) 20))
;; face2
(define sheet2 (sheet:face face2))
;; sheet2
(define sh2d–2 (sheet:2d sheet2))
;; sh2d–2
(define face3 (face:planar–disk (position –20 15 0)
    (gvector 0 1 0) 30))
;; face3
(define sheet3 (sheet:face face3))
;; sheet3
(define sh2d–3 (sheet:2d sheet3))
;; sh2d–3
(define e (bool:nonreg–unite b1 b2 sh2d–1 sh2d–2
    sh2d–3))
;; e
(define plane–pos (position –20 0 0))
;; plane–pos
```

```
(define plane-normal (gvector 1 0 0))
;; plane-normal
(define imprint-face (face:planar-disk
    plane-pos plane-normal 50))
;; imprint-face
(define imprint-sheet (sheet:face imprint-face))
;; imprint-sheet
(define sh2d-4 (sheet:2d imprint-sheet))
;; sh2d-4
(define sol-im1 (solid:imprint e imprint-sheet))
;; sol-im1
(cell:attach e)
;; (#[entity 12 1] #[entity 13 1] #[entity 14 1]
;; #[entity 15 1] #[entity 16 1] #[entity 17 1]
;; #[entity 18 1] #[entity 19 1] #[entity 20 1]
;; #[entity 21 1])
(define g (graph (entity:cells e)))
;; g
(define g-2d (graph:subgraph-2dcell g))
;; g-2d
(define g-3d (graph:subgraph-3dcell g))
;; g-3d
```

graph:subset–with–plane

Scheme Extension: Graph Theory

Action: Finds the subset of a graph on one side of a plane.

Filename: sbool/sbool_scm/selective_scm.cxx

APIs: api_subset_graph_with_plane

Syntax:

```
(graph:subset-with-plane whole-graph plane-origin
    plane-normal [acis-opts])
```

Arg Types:

whole–graph	graph
plane–origin	position
plane–normal	vector
acis–opts	acis–options

Returns: graph

Errors: None

Description: Finds the subset of a graph on one side of a plane. The graph must first be split at that plane. It is assumed that the graph is made of entities which are either faces or cells.

whole–graph is an input graph that is split to get a subset of the graph.

plane–origin is the position of the plane.

plane–normal is a vector to the plane.

The optional acis–opts contains parameters for versioning and journaling.

Limitations: None

Example:
```
; graph:subset-with-plane
; Create entities
(define b1 (solid:block (position -40 0 0)
    (position 40 10 10)))
;; b1
(define b2 (solid:block (position -40 10 0)
    (position -30 30 10)))
;; b2
(define b3 (solid:block (position 30 10 0)
    (position 40 30 10)))
;; b3
(define b4 (solid:block (position -5 0 0)
    (position 5 30 10)))
;; b4
(define b5 (solid:block (position -35 15 0)
    (position 35 25 10)))
;; b5
(define u1 (bool:unite b1 b2 b3))
;; u1
(define whole-entity (bool:nonreg-unite u1 b4 b5))
;; whole-entity
(cell:attach whole-entity)
;; (#[entity 7 1] #[entity 8 1] #[entity 9 1]
;; #[entity 10 1] #[entity 11 1] #[entity 12 1]
;; #[entity 13 1] #[entity 14 1] #[entity 15 1]
;; #[entity 16 1])
(define whole-graph (graph
    (entity:cells whole-entity)))
;; whole-graph
(define plane-pos (position 5 0 0))
;; plane-pos
(define plane-normal1 (gvector 1 0 0))
;; plane-normal1
(define plane-normal2 (gvector:transform
    plane-normal1 (transform:scaling -1)))
;; plane-normal2
```

```
(define partial-graph1 (graph:subset-with-plane
    whole-graph plane-pos plane-normal1))
;; partial-graph1
(define partial-graph2 (graph:subset-with-plane
    whole-graph plane-pos plane-normal2))
;; partial-graph2
; Make sure two halves have nothing in common.
(law:equal-test (graph:intersect
    partial-graph1 partial-graph2) (graph ""))
;; #t
; Make sure two halves are both non-empty
(law:equal-test
    (equal? partial-graph1 (graph "")) #f)
;; #t
(law:equal-test
    (equal? partial-graph2 (graph "")) #f)
;; #t
```

tube:options

Scheme Extension: Booleans

Action: Creates a data structure used by selective Booleans.

Filename: sbool/sbool_scm/selective_scm.cxx

APIs: None

Syntax:

```
(tube:options [keep-law-string keep-law]
    [keep-branches-string branch-bool]
    [bool-type-string bool-type]
    ["connected" logical]
    ["merge" logical]
    ["only_from" integer])
```

Arg Types:

keep–law–string	string
keep–law	law
keep–branches–string	string
branch–bool	boolean
bool–type–string	string
bool–type	string
“connected”	string
logical	boolean
“merge”	string
logical	boolean
“only_from”	integer

Returns: entity

Errors: None

Description: The purpose of the Tube–Options data structure is to hold information between selective Boolean operations stages 1 and 2 and to make selective Booleans accessible by other modeling operations. Sweeping, among other modeling operations, creates and uses this data structure. The data structure is used as input to bool:tube.

The options are explained in detail in the discussion topic *Options for a Law Sweep* and include:

– bool_type
– keep_branches
– keep_law

Limitations: None

Example:

```
; tube:options
; Define some entities
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define c1 (solid:cylinder (position –5 5 2)
    (position 15 5 2)1))
;; c1
(define c2 (solid:cylinder (position –5 5 8)
    (position 15 5 8)1))
;; c2
(define c (solid:unite c1 c2))
;; c
(define start (list (list–ref (entity:faces c) 1)
    (list–ref (entity:faces c) 4)))
;; start
(define end (list (list–ref (entity:faces c) 2)
    (list–ref (entity:faces c) 5)))
;; end
(define opt1 (tube:options ”keep_law” ”x=0”
    ”bool_type” ”UNITE”))
;; opt1
(define tube (bool:tube b c start end opt1)))
;; tube
```

```
; Another Example.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define c (solid:cylinder (position –5 5 5)
    (position 15 5 5)2))
;; c
(define start (list–ref (entity:faces c) 1))
;; start
(define end (list–ref (entity:faces c) 2))
;; end
(define opt1 (tube:options ”keep_law”
    ”x=0” ”bool_type” ”unite”))
;; opt1
opt1
; #[Tube_Options ”keep_law” X=0 ”keep_branches”
    #f”unite”]
(define d (bool:tube b c start end opt1))
;; d
```