Chapter 15.
Scheme Extensions Ra thru Zz

Topic: Ignore

ray

Scheme Extension: Ray Testing

Action: Creates a ray given a base position and gvector.

Filename: kern/kern_scm/ray_scm.cxx

APIs: None

Syntax: `(ray position gvector)`

Arg Types:	gvector position	gvector position

Returns: ray

Errors: None

Description: A ray is not visible. An important usage for a ray is to pierce a body or object for selection purposes.

gvector specifies the direction of the ray.

position specifies the starting position.

Limitations: None

Example:

```
; ray
; Create ray1 from a position and a gvector.
(define ray1
    (ray (position 0 0 0) (gvector 0 1 0)))
;; ray1
; Create ray2 from a position and a gvector.
(ray (position 78 5 12) (gvector 0 0 1))
;; #[ray (78 5 12) (0 0 1)]
; Do something useful with ray.
; Create a block.
(define block1
    (solid:block (position –10 –5 –15)
    (position 25 25 25)))
;; block1
; Remember the face.
(define face1 (pick:face ray1))
;; face1
; Highlight the face crossed by the ray.
(entity:set–highlight face1 #t)
;; #[entity 3 1]
```

ray:gvector

Scheme Extension: Ray Testing

Action: Gets the gvector component of a ray.

Filename: kern/kern_scm/ray_scm.cxx

APIs: None

Syntax: (**ray:gvector** ray)

Arg Types: ray ray

Returns: gvector

Errors: None

Description: The argument ray specifies a position and a gvector. This extension returns the gvector direction from a ray.

ray specifies a ray.

Limitations: None

Example:

```
; ray:gvector
; Define a ray.
(define ray1 (ray (position –5 –5 –5)
    (gvector 1 0 0)))
;; ray1
; Extract the gvector component of a ray.
(ray:gvector ray1)
;; #[gvector 1 0 0]
; Get the gvector component of a read event.
(ray:gvector (pick:ray (read–event)))
; Using the mouse, select a screen position.
;; #[gvector 0.119924328873844 0.949842878199001
;; 0.288819428153296]
```

ray:position

Scheme Extension: Ray Testing

Action: Gets the position component of a ray.

Filename: kern/kern_scm/ray_scm.cxx

APIs: None

Syntax: **(ray:position** ray)

Arg Types: ray ray

Returns: position

Errors: None

Description: The argument ray specifies a position and a gvector. This extension returns the position of the ray.

ray specifies a ray.

Limitations: None

Example:

```
; ray:position
; Define a ray.
(define ray1 (ray (position –5 –5 –5)
    (gvector 1 0 0)))
;; ray1
; Extract the position component of a ray.
(ray:position ray1)
;; #[position –5 –5 –5]
; get position component of a read event
(ray:position (pick:ray (read–event)))
;; #[position –35.0111607852786 –48.3518844171113
;; –1.77635683940025e–15]
```

ray:queue

Scheme Extension: Ray Testing

Action: Adds a ray to the ray queue.

Filename: kern/kern_scm/ray_scm.cxx

APIs: None

Syntax: `(ray:queue posx posy posz vecx vecy vecz radius)`

Arg Types:

posx	real
posy	real
posz	real
vecx	real
vecy	real
vecz	real
radius	real

Returns: ray

Errors: None

Description: The parameters describe the position and vector for a ray as six reals. The given ray is constructed and added to the ray queue. It may later be read by read–ray.

posx, posy and posz specifies the x, y and z positions of the ray.

vecx, vecy and vecz specifies the x, y and z components of the ray vector.

radius is used for picking.

Limitations: None

Example:

```
; ray:queue
; Create a ray and add it to the ray queue
(ray:queue 1 2 3 4 5 6 7)
;; #[ray (1 2 3) (0.455842 0.569803 0.683763)]
```

ray:valid?

Scheme Extension: Ray Testing

Action: Tests whether a ray is valid.

Filename: kern/kern_scm/ray_scm.cxx

APIs: None

Syntax: `(ray:valid? ray)`

Arg Types:	ray	ray

Returns: boolean

Errors: None

Description: A ray is considered invalid if the length of the vector is zero.

ray specifies the ray to be tested for validity.

Limitations: None

Example:

```
; ray:valid?
; Test whether an interactively read ray is valid.
(ray:valid? (read-ray))
;; #t
```

ray?

Scheme Extension: Ray Testing

Action: Determines if a Scheme object is a ray.

Filename: kern/kern_scm/ray_scm.cxx

APIs: None

Syntax: `(ray? object)`

Arg Types:	object	scheme–object

Returns: boolean

Errors: None

Description: Refer to Action.

object specifies the scheme–object to be queried for a ray.

Limitations: None

Example:

```
; ray?
; Create a ray.
(define ray1 (ray (position -5 -5 -5)
    (gvector 1 0 0)))
;; ray1
; Determine if the ray is actually a ray.
(ray? ray1)
;; #t
; Determine if a position is a ray.
(ray? (position 0 0 0))
;; #f
```

read–ray

Scheme Extension: Ray Testing

Action: Reads a ray from the ray queue or from a pick–event.

Filename: kern/kern_scm/ray_scm.cxx

APIs: None

Syntax: (**read–ray**)

Arg Types: None

Returns: ray

Errors: None

Description: If the ray queue contains a ray, it is removed and returned and the ray size option is set from the queued information. Otherwise, a pick–event is read, using read–event, and converted to a ray. If journaling is on, a (ray: queue ....) command is written to the journal file. This results in a ray in the queue when the journal is replayed.

If the pick–event is from the right mouse button, the ray vector is set to an invalid zero length. This can be used as a stop condition in picking loops.

Limitations: None

Example:

```
; read–ray
; Create a ray using the mouse.
(read–ray)
; Use the left mouse button to create a pick–event.
;; #[ray (94.7806 –194.257 116.706)
;; (–0.408248 0.816497 –0.408248)]
```

shell?

Scheme Extension: Model Topology

Action: Determines if a Scheme object is a shell.

Filename: kern/kern_scm/ent_scm.cxx

APIs: None

Syntax: (**shell?** object)

Arg Types: object scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if the specified object is a shell; otherwise, it returns #f.

object specifies the scheme–object to be queried for a shell.

Limitations: None

Example:

```
; shell?
; Create a solid block.
(define block1
    (solid:block (position 0 0 0) (position 8 8 8)))
;; block1
; Get a list of the block's shells.
(define shells1
    (entity:shells block1))
;; shells1
; Determine if one of the shells
; is actually a shell.
(shell? (car shells1))
;; #t
; Determine if the block is a shell.
(shell? block1)
;; #f
```

solid?

Scheme Extension: Model Geometry

Action: Determines if a Scheme object is a solid.

Filename: kern/kern_scm/ent_scm.cxx

APIs: None

Syntax: (**solid?** object)

Arg Types: object scheme–object

Returns: boolean

Errors: None

Description: Refer to Action.

object specifies the scheme–object to be queried for a solid.

Limitations: None

Example:

```
; solid?
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 40 40 40)))
;; block1
; Determine if the solid block is a solid.
(solid? block1)
;; #t
; Create a solid sphere.
(define sphere1 (solid:sphere (position 0 0 0) 38))
;; sphere1
; Determine if the solid sphere is a solid.
(solid? sphere1)
;; #t
; Create a circular edge.
(define edge–1
    (edge:circular (position 0 0 0) 25 0 180))
;; edge–1
; Determine if the edge is a solid.
(solid? edge–1)
;; #f
```

surface:domain

Scheme Extension: Model Geometry

Action: Determines the domain of a face's surface.

Filename: kern/kern_scm/qfac_scm.cxx

APIs: None

Syntax: (**surface:domain** face)

Arg Types: face face

Returns: unspecified

Errors: None

Description: Given a face, returns the underlying surface's domain.

Limitations: None

Example:
```
; surface:domain
; Create a face to use.
(define face1 (face:law "vec(cos(x), y, x)"
    -20 (law:eval "10*pi") -10 10))
;; face1
(surface:domain face1)
;; ()
; Surface domain:
; -20 : 31.415926535898
; -10 : 10
```

surface:eval

Scheme Extension: Construction Geometry, Physical Properties

Action: Evaluates a position on a parametric surface.

Filename: kern/kern_scm/surf_scm.cxx

APIs: None

Syntax: (**surface:eval** srf su sv [level])

Arg Types:

srf	surface
su	real
sv	real
level	integer

Returns: real ...

Errors: None

Description: This extension finds the position on a parametric surface with the given parameter values and optionally calculates the first and second derivatives.

srf specifies the surface to evaluate.

su specifies the *u* parameter value.

sv specifies the *v* parameter value.

level specifies the depth of evaluation, and it has the following valid values:

0 = turns the position
1 = Returns the position and the first derivative (*xu*, *xv*)
2 = Returns the position, and the first and second derivatives (*xuu*, *xuv*, *xvv*)

Limitations: None

Example:

```
; surface:eval
; Create a planar face.
(define face1
    (face:plane (position 0 0 0) 30 40))
;; face1
; Evaluate the position of a point
; on the planar surface.
(surface:eval (surface:from–face face1) 15 15 0)
;; (#[position 15 15 0])
; Create a conical surface.
(define cone1
    (face:cone (position 0 0 0) (position 0 50 0)
    30 20 0 180))
;; cone1
; Evaluate the position and the first derivative
; of a point on the conical surface.
(surface:eval (surface:from–face cone1) 0.5 0.5 1)
;; (#[position 23.7458553521721 14.7087101353638
;; –12.9724199023621] #[gvector –5.16324300907818
;; 29.4174202707276 2.82069251152796]
;; #[gvector –12.9724199023621 0
;; –23.7458553521721])
```

surface:eval–curvatures

Scheme Extension: Construction Geometry, Physical Properties

Action: Evaluates a the principal curvatures and directions at the given uv on the surface

Filename: kern/kern_scm/surf_scm.cxx

APIs: None

Syntax: (**surface:eval–curvatures** srf su sv)

Arg Types:		
	srf	surface
	su	real
	sv	real

Returns: real ...

Errors: None

Description: This extension finds the two principal curvatures of the surface at the given parameter values. It also returns the directions of the principal curvatures.

srf specifies the surface to evaluate.

su specifies the *u* parameter value.

sv specifies the *v* parameter value.

Limitations: This extension does not take account of surface discontinuities, so that evaluating the curvatures at such points may be unreliable, as no attempt is made to compute sided curvatures

Example:

```
;surface:eval-curvatures
;; Create the surface
(solid:wiggle 50 50 50 "anti")
;; #[entity 1 1]
;; (ray:queue 258.188 -312.023 293.684 -0.537024
0.628851 -0.562273 1)
(define f (pick-face))
;; f
(surface:eval-curvatures (surface:from-face f) .5 .5)
;; (0.067385584485619 #[gvector -0.576407926659592
;; -0.281960099592504 -0.766976143254649]
;; 0.120983661175455 #[gvector -0.595361733728196
;; 0.787806079907338 0.157816306106784])
```

surface:eval–normal

Scheme Extension: Construction Geometry, Physical Properties

Action: Gets the normal of a surface at the specified parameter values.

Filename: kern/kern_scm/surf_scm.cxx

APIs: None

Syntax: (**surface:eval-normal** surf u v)

Arg Types:

surf	surface
u	real
v	real

Returns: gvector

Errors: None

Description: Refer to Action.

surf specifies a surface.

u specifies a *u* parameter value.

v specifies a *v* parameter value.

Limitations: None

Example:

```
; surface:eval–normal
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 40 40 40)))
;; block1
; Get a list of the faces of the solid block.
(define entities1
    (entity:faces block1))
;; entities1
; Convert one face to a surface.
(define surface1 (surface:from–face
    (car entities1)))
;; surface1
; Evaluate the normal of the surface with the
; specified parameter values.
(surface:eval–normal surface1 1 1)
;; #[gvector 0 0 1]
```

surface:eval–pos

Scheme Extension: Construction Geometry, Physical Properties

Action: Gets the position on a surface at the specified parameter values.

Filename: kern/kern_scm/surf_scm.cxx

APIs: None

Syntax: (**surface:eval–pos** surf u v)

Arg Types:

surf	surface
u	real
v	real

Returns: position

Errors: None

Description: Refer to Action.

surf specifies a surface.

u specifies a *u* parameter value.

v specifies a *v* parameter value.

Limitations: None

Example:

```
; surface:eval–pos
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 40 40 40)))
;; block1
; Get a list of the faces of the solid block.
(define entities1
    (entity:faces block1))
;; entities1
; Convert one face to a surface.
(define surface1 (surface:from–face
    (car entities1)))
;; surface1
; Evaluate the position on the surface at the
; specified parameter values.
(surface:eval–pos surface1 1 1)
;; #[position 21 21 40]
```

surface:from–face

Scheme Extension: Model Geometry

Action: Creates a surface from a face.

Filename: kern/kern_scm/surf_scm.cxx

APIs: None

Syntax: (**surface:from–face** face)

Arg Types:	face	face

Returns: surface

Errors: None

Description: Refer to Action.

face specifies a face.

Limitations: None

Example:
```
; surface:from–face
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 20 30 40)))
;; block1
; Get a list of the block’s faces.
(define entities1
    (entity:faces block1))
;; entities1
; Convert one of the block’s faces to a surface.
(surface:from–face (car (cdr entities1)))
;; #[plane surface 401c2310]
```

surface:point–perp

Scheme Extension: Construction Geometry, Physical Properties

Action: Reverse evaluates a position on a parametric surface.

Filename: kern/kern_scm/surf_scm.cxx

APIs: None

Syntax: (**surface:point–perp** srf pos)

Arg Types:

srf	surface
pos	position pair (position point_on_surface gvector

Returns: boolean

Errors: None

Description: Given a surface and a position on the surface, this extension returns the surface position and gvector.

srf specifies a parametric surface.

pos specifies a position on the surface.

Limitations: None

Example:

```
; surface:point–perp
; Create a conical surface.
(define cone1(face:cone (position 20 20 4)
    (position 20 20 8)2.5 0 0 360 0.6))
;; cone1
; Demonstrate the C++ function surface::point_perp()
; through the Scheme interface.
(surface:point–perp (surface:from–face cone1)
    (position 22 20 2.5))
;; (#[position 23.0337078651685 20 3.14606741573034]
;; #[gvector 0.847998304005088 0 0.52999894000318]
;; #[par–pos –0.402799194402416 0])
```

surface:range

Scheme Extension: Construction Geometry, Physical Properties

Action: Returns the parameter range of a surface.

Filename: kern/kern_scm/surf_scm.cxx

APIs: None

Syntax: (**surface:range** srf [box–pos–1 box–pos–2])

Arg Types:		
	srf	surface
	box–pos–1	position
	box–pos–2	position

Returns: pair

Errors: None

Description: Given a surface and (optionally) a 3–space box, returns the u and v parameter ranges of the surface. Interfaces to the C++ surface::param_range member function.

The returned value is a pair with the first being the u range and the second one being the v range. Note #f is used when any end of an interval is unbounded, so for example, (0 . #f) represents an interval starting at 0 but which is unbounded above. An empty interval can be recognized by the first value of the pair being greater than the second.

srf specifies a surface.

box–pos–1 defines the first of diagonal corners of an option box.

box–pos–2 defines the other diagonal corner.

Limitations: None

Example:
```
; surface:range
; Create a conical surface.
(define cone1 (face:cone (position 20 20 4)
    (position 20 20 8) 2.5 0 0 360 0.6))
;; cone1
(surface:range (surface:from–face cone1))
;; ((#f . 1.88679622641132)
;;  (–3.14159265358979 . 3.14159265358979))
```

surface?

Scheme Extension: Physical Properties

Action: Determines if a Scheme entity is a surface.

Filename: kern/kern_scm/surf_scm.cxx

APIs: None

Syntax: (**surface?** object)

Arg Types: object scheme–object

Returns: boolean

Errors: None

Description: The extension returns #t if the object is a surface, and #f if it is not a surface.

object specifies the scheme–object that has to be queried for a surface.

Limitations: None

Example:

```
; surface?
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Create a cylindrical face.
(define cyl1
    (face:cylinder (position 0 0 0)
    (position 0 20 0) 10))
;; cyl1
; Determine if the solid block is a surface.
(surface? block1)
;; #f
; Get a list of the faces of the block.
(define entities1
    (entity:faces block1))
;; entities1
; Convert one face to a surface.
(define surface1 (surface:from–face
    (car entities1)))
;; surface1
; Determine if one of the surfaces of the block
; is actually a surface.
(surface? surface1)
;; #t
```

test:deep–copy

Scheme Extension: Debugging

Action: Tests the deep copy functionality for improper sharing.

Filename: kern/kern_scm/ent_scm.cxx

APIs: api_test_deep_copy

Syntax:

```
(test:deep–copy entity–list [tolerance] [report–all]
    [file1 [file2]])
```

Arg Types:

entity–list	entity | (entity ...)
tolerance	real
report–all	boolean
file1	string
file2	string

Returns: boolean

Errors: None

Description: The entities are saved and restored to test for sharing after api_deep_copy_entity has been called. If the command is used in a debug build, pattern filling the original entities allows a greater level of checking. For debug or release builds, the original and the deep copy SAT files can be compared for differences.

This function is primarily for internal testing. However, if derived entities are used outside of ACIS, this command can verify if the entities are deep copyable.

entity–list specifies a list of any entities.

If tolerance is specified, the geometry is compared before and after the deep copy.

If report–all is set to true (#t), then all the attributes that could not be deep copied are reported.

file1 specifies the name of the file for storing the entities before deep–copy.

file2 specifies the name of the file for storing the entities after deep–copy.

Limitations: None

Example:
```
; test:deep–copy
; Create a wiggle
(define wiggle1 (solid:wiggle 25 25 25 ”anti”))
;; wiggle1
(test:deep–copy wiggle1)
;; #t
```

test:equal

Scheme Extension: Debugging

Action: Compares two arguments for equality.

Filename: kern/kern_scm/law_scm.cxx

APIs: None

Syntax: `(`**`test:equal`** `arg1 arg2 [tolerance] [msg])`

Arg Types:	arg1	boolean | real | position | gvector | par_pos | number_array | graph | string
	arg2	boolean | real | position | gvector | par_pos | number_array | graph | string
	tolerance	real
	msg	string

Returns: boolean

Errors: None

Description: Compares arg1 and arg2 and returns true if they are equal. “Equality” for reals, positions, gvectors, and the elements of number arrays, is defined as being within the tolerance value (default SPAresabs).

arg1 and arg2 specify the arguments to be tested for equality.

tolerance specifies the tolerance value to be used if the input arguments belong to one of the following types: reals, positions, gvectors, elements of number arrays.

If a msg string is specified, this string prints out to the debug file if the result is false.

Limitations: None

Example:

```
; test:equal
(test:equal 2 2)
;; #t
(test:equal (position 0 0 0) (position 0 0 1))
;; #f
(test:equal 5 5 “different number”)
;; #t
(test:equal 5 2 “different number”)
;; *** Error test:equal: different number
;; #f
```

test:greater–than

Scheme Extension: Debugging

Action: Compares two real numbers.

Filename: kern/kern_scm/law_scm.cxx

APIs: None

Syntax: `(test:greater–than variable fixed [tolerance] [msg])`

Arg Types:

variable	real
fixed	real
tolerance	real
msg	string

Returns: boolean

Errors: None

Description: Compares variable to fixed, returning true if the variable is greater than the fixed. If the variable is less than the fixed by an amount less than the tolerance, the result is also true.

variable and fixed specify the arguments to be tested.

tolerance specifies the tolerance value to be used.

If a msg string is specified, this string prints out to the debug file if the result is false.

Limitations: None

Example:

```
; test:greater–than
(test:greater–than 2 1)
;; #t
(test:greater–than 2.0 2.1 0.2
    “greater within tolerance”)
;; #t
```

test:less–than

Scheme Extension: Debugging

Action: Compares two real numbers.

Filename: kern/kern_scm/law_scm.cxx

APIs: None

Syntax: `(test:less–than variable fixed [tolerance] [msg])`

Arg Types:

variable	real
fixed	real
tolerance	real
msg	string

Returns: boolean

Errors: None

Description: Compares variable to fixed, returning true if the variable is less than the fixed. If the variable is greater than the fixed by an amount less than the tolerance, the result is also true.

variable and fixed specify the arguments to be tested.

tolerance specifies the tolerance value to be used.

If a msg string is specified, this string prints out to the debug file if the result is false.

Limitations: None

Example:
```
; test:less-than
(test:less-than 1 2)
;; #t
(test:less-than 2.1 2.0 0.2
    "less-than within tolerance")
;; #t
```

test:performance

Scheme Extension: Debugging

Action: Runs the performance test(s).

Filename: kern/kern_scm/perf_scm.cxx

APIs: None

Syntax: (**test:performance** string)

Arg Types: string string

Returns: string

Errors: None

Description: Runs performance test(s) specified by the string argument. The format of string varies based on the test(s) requested. Currently, only the ENTITY_LIST performance test is available.

The format for string is: "ENTITY_LIST [all | add | clear | assign | lookup | next | index] list_size iteration". list_size is the size of the entity_list and iteration specifies how many times the test is to be run. If list_size and iteration are not specified, the default values 1000 and 1 (respectively) are assigned.

Limitations: Only the ENTITY_LIST performance test is currently available.

Example:

```
; test:performance
; Run entity_list performance test. Request list_size
; and iteration different from defaults.
(test:performance "entity_list all 100 10000")
;; ()
; Size of the list = 100
; Number of iteration = 10000
; 0.810000 seconds to add 100 items 10000 times
; 0.390000 seconds to clear 100 items 10000 times
; 1.151000 seconds to assign 100 items 10000 times
; 0.190000 seconds to lookup 100 items 10000 times
; 0.170000 seconds to next 100 items 10000 times
; 0.541000 seconds to index 100 items 10000 times
```

text:font

Scheme Extension: Text

Action: Gets a text entity's font.

Filename: kern/kern_scm/text_scm.cxx

APIs: None

Syntax: (**text:font** text–entity)

Arg Types:	text–entity	text

Returns: string

Errors: None

Description: This extension returns the font name assigned to the text–entity, which is enclosed in quotes. The font name is composed of foundry, the family, the weight, the slant, and the set–width. In X Windows, use the xlsfonts tool to display a list of available fonts.

text–entity specifies a text entity.

Limitations: None

Example:

```
; text:font
; Create a text entity.
(define words (text (position 5 10 15)
    "Hello World"
    "new century schoolbook-bold-i-normal" 20))
;; words
; Get the font value of the text entity.
(text:font words)
;; "new century schoolbook-bold-i-normal"
```

text:location

Scheme Extension: Text

Action: Gets a text entity's location.

Filename: kern/kern_scm/text_scm.cxx

APIs: None

Syntax: (**text:location** text-entity)

Arg Types: text–entity text

Returns: position

Errors: None

Description: This extension returns the position of the point of origin of the text–entity. The origin is located at the left edge of the left–most (leading) character in the string, at the baseline on which the character sits.

text–entity specifies a text entity.

Limitations: None

Example:

```
; text:location
; Create a text entity.
(define words (text (position 5 10 15)
    "Hello World"
    "new century schoolbook-bold-i-normal" 20))
;; words
; Get the location of the text entity.
(text:location words)
;; #[position 5 10 15]
```

text:set–font

Scheme Extension: Text

Action: Sets a text entity's font.

Filename: kern/kern_scm/text_scm.cxx

APIs: None

Syntax: (**text:set–font** text–entity font)

Arg Types:

text–entity	text
font	string

Returns: text

Errors: None

Description: Refer to Action.

text–entity specifies a text entity.

font specifies a text font to be used, which is enclosed in quotes. font is interpreted by the current windowing driver; for example, if the system is running X Windows, available fonts are those listed by running the xlsfonts tool. If the specified font name is not a perfect match, this extension inserts wildcards into the font at various places to search for an approximate match.

X Windows specifies fonts as the foundry, the family, the weight, the slant, the set–width, any additional style, the pixel size, the point size, the *x*–resolution, the *y*–resolution, the spacing, the average width, the registry, and the encoding. For example:

```
–adobe–courier–medium–r–normal––8–80–75–75–m–50–iso8859–1
```

The only fields used in this extension are family, weight, slant, and set–width. In the preceding example, these fields are the substring:

```
courier–medium–r–normal
```

Limitations: None

Example:

```
; text:set–font
; Create the first text entity.
(define words (text (position –20 0 0)
    ”abcdefghijklm” ”symbol–medium–r–normal” 30))
;; words
; Create the second text entity.
(define words2 (text (position –20 –20 –20)
    ”nopqrstuvwxyz” ”symbol–medium–r–normal” 30))
;; words2
; OUTPUT Original
```

```
; Set the font of the first text entity.
(text:set-font words "courier-bold-r-normal")
;; #[entity 2 1]
; Set the font of the second text entity.
(text:set-font words2 "helvetica-bold-r-normal")
;; #[entity 3 1]
; OUTPUT Result
```

Figure 15-1. text:set–font

text:set–location

Scheme Extension: Text

Action: Sets a text entity's location.

Filename: kern/kern_scm/text_scm.cxx

APIs: None

Syntax: (**text:set–location** text–entity location)

Arg Types:		
	text–entity	text
	location	position

Returns: text

Errors: None

Description: This extension sets the point of origin (location) for a text–entity. The origin is located at the left edge of the left–most (leading) character in the string, at the baseline on which the character sits.

text–entity specifies a text entity.

location specifies the origin for the text entity.

Limitations: None

Example:

```
; text:set-location
; Create a text entity.
(define words (text
    (position 0 0 0) "Hello World."
    "times-medium-r-normal" 30))
;; words
; Set the location of the text entity.
(define location (text:set-location words
    (position 50 -20 10)))
;; location
```

text:set–size

Scheme Extension: Text

Action: Sets a text entity's font size in points.

Filename: kern/kern_scm/text_scm.cxx

APIs: None

Syntax: (**text:set-size** text-entity size)

Arg Types:		
	text–entity	text
	size	integer

Returns: text

Errors: None

Description: This extension sets the size of a text–entity.

text–entity specifies a text entity.

size is an integer that specifies the size of the font in points. (Usually, business correspondence uses 10 or 12 point fonts.) If the exact size font cannot be found, the font nearest in size is used. When searching for a font, the size specified as part of font is discarded and replaced with the size specified in size.

Limitations: None

Example:
```
; text:set–size
; Create a text entity.
(define words (text
    (position 0 0 0) ”Hello World.”
    ”times–medium–r–normal” 30))
;; words
; Set the size of the text entity.
(define size (text:set–size words 10))
;; size
```

text:set–string

Scheme Extension: Text

Action: Sets a text entity’s string.

Filename: kern/kern_scm/text_scm.cxx

APIs: None

Syntax: (**text:set–string** text–entity string)

Arg Types:		
	text–entity	text
	string	string

Returns: text

Errors: None

Description: Refer to Action.

text–entity specifies a text entity.

string, which is enclosed in quotes, specifies the text string to be displayed by a text–entity.

Limitations: None

Example:
```
; text:set–string
; Create a text entity.
(define words (text
    (position 0 0 0) ”Hello World.”
    ”times–medium–r–normal” 30))
;; words
; Set the string of the text entity.
(define string
    (text:set–string words ”Hello Universe!”))
;; string
```

text:size

Scheme Extension: Text

Action: Gets a text entity's font size in points.

Filename: kern/kern_scm/text_scm.cxx

APIs: None

Syntax: `(text:size text–entity)`

Arg Types:

text–entity	text

Returns: integer

Errors: None

Description: This extension returns the font size, in points, of a text–entity.

text–entity specifies a text entity.

Limitations: None

Example:

```
; text:size
; Create a text entity.
(define words (text
    (position 0 0 0) "Hello World."
    "times–medium–r–normal" 20))
;; words
; Get the size of the text entity.
(text:size words)
;; 20
```

text:string

Scheme Extension: Text

Action: Gets a text entity's string.

Filename: kern/kern_scm/text_scm.cxx

APIs: None

Syntax: `(text:string text–entity)`

Arg Types:

text–entity	text

Returns: string

Errors: None

Description: This extension returns the text string in a text–entity, which is enclosed in quotes.

Limitations: None

Example:
```
; text:string
; Create a text entity.
(define words (text (position 0 0 0) "Hello World."
    "times-medium-r-normal" 30))
;; words
; Get the text entity string.
(text:string words)
;; "Hello World."
```

text?

Scheme Extension: Text

Action: Determines if a Scheme object is a text entity.

Filename: kern/kern_scm/text_scm.cxx

APIs: None

Syntax: (**text?** object)

Arg Types:	object	scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if the object is a text entity; otherwise, it returns #f.

object specifies a scheme–object that has to be queried for a text entity.

Limitations: None

Example:
```
; text?
; Create a text string.
(define words (text (position 0 0 0) "Hello World."
    "times-medium-r-normal" 30))
;; words
; Determine if the string is a text string.
(text? words)
;; #t
```

timer:end

Scheme Extension: Debugging

Action: Stops the timer.

Filename: kern/kern_scm/law_scm.cxx

APIs: None

Syntax: (**timer:end**)

Arg Types: None

Returns: string

Errors: None

Description: The commands timer:start, timer:end, timer:show–time, and timer:get–time are used to measure performance of some command or series of commands. They measure only the CPU time required to execute the command and not any delays incurred from entering the commands into Scheme.

Limitations: None

Example:

```
; timer:end
; Start the timer
(timer:start)
;; "timer on"
; Create a solid block.
(define blockA (solid:block
    (position 0 0 0) (position 20 20 20)))
;; blockA
; Create another solid block
(define blockB (solid:block (position 0 0 0)
    (position –20 –20 –20)))
;; blockB
;Stop timer.
(timer:end)
;; "timer off, use timer:get–time"
(timer:get–time)
;; 21.271
```

timer:get–time

Scheme Extension: Debugging

Action: Calculates and returns the amount of time elapsed since timer:start was executed.

Filename: kern/kern_scm/law_scm.cxx

APIs: None

Syntax: (**timer:get–time**)

Arg Types: None

Returns: real

Errors: None

Description: The commands timer:start, timer:end, timer:show–time, and timer:get–time are used to measure performance of some command or series of commands. They measure only the CPU time required to execute the command and not any delays that might incur from entering the commands into Scheme. They are used most often to determine the time required to execute one or more commands.

timer:get–time can be executed/interspersed any number of times throughout a command or series of commands.

You can also use timer:show–time to display the amount of time elapsed since timer:start was executed.

Limitations: None

Example:

```
; timer:get–time
; Start the timer
(timer:start)
;; ”timer on”
; Create a solid block.
(define blockA (solid:block
    (position 0 0 0) (position 20 20 20)))
;; blockA
; Create another solid block
(define blockB (solid:block (position 0 0 0)
    (position –20 –20 –20)))
;; blockB
;Stop timer.
(timer:end)
;; ”timer off, use timer:get–time”
(timer:get–time)
;; 21.271
```

timer:show–time

Scheme Extension: Debugging

Action: Calculates and returns the amount of time elapsed since the timer:start command was executed. This is most often used to measure performance.

Filename: kern/kern_scm/law_scm.cxx

APIs: None

Syntax: (**timer:show–time**)

Arg Types: None

Returns: real

Errors: None

Description: The commands timer:start, timer:end, timer:get–time and timer:show–time are used to measure performance of some command or series of commands. This feature measures only the CPU time required to execute the command and not any delays incurred from entering the commands into Scheme or any other interference or condition.

timer:show–time can be executed/interspersed any number of times throughout a command or series of commands.

timer:show–time returns a real number only after timer:start is executed. If timer:start has not been executed, timer:show–time returns zero (0).

You can also use timer:get–time to display the amount of time elapsed since timer:start was executed.

Limitations: None

Example:

```
; timer:show–time
; Start the timer
(timer:start)
;; ”timer on”
; Create a solid block
(define blockA (solid:block (position 0 0 0)
    (position 20 20 20)))
;; blockA
; Create another solid block
(define blockB (solid:block (position 0 0 0)
    (position –20 –20 –20)))
;; blockB
; Set color for blockB
(entity:set–color blockB 7)
;; ()
;show how much time has passed.
(timer:show–time)
;; Elapsed time = 33.989
;; 356.722
```

```
; Create a third solid block
(define blockC (solid:block (position 0 0 0)
    (position 40 40 40)))
;; blockC
; Set color for blockB
(entity:set-color blockC 4)
;; ()
;turn timer off.
(timer:end)
;; "timer off, use timer:get-time"
(timer:get-time)
;; 4.556
```

timer:start

Scheme Extension: Debugging

Action: Starts an internal clock/timer. The timer: commands are most often used to measure performance of some command or series of commands.

Filename: kern/kern_scm/law_scm.cxx

APIs: None

Syntax: (**timer:start**)

Arg Types: None

Returns: string

Errors: None

Description: The commands timer:start, timer:end, timer:show–time, and timer:get–time are used to measure performance of some command or series of commands. It measures only the CPU time required to execute the command and not any delays that might incur from entering the commands into Scheme.

Limitations: None

Example:

```
; timer:start
; Start the timer
(timer:start)
;; ”timer on”
; Create a solid block.
(define blockA (solid:block
    (position 0 0 0) (position 20 20 20)))
;; blockA
; Create another solid block
(define blockB (solid:block (position 0 0 0)
    (position –20 –20 –20)))
;; blockB
;Stop timer.
(timer:end)
;; ”timer off, use timer:get–time”
(timer:get–time)
;; 2.274
```

transform:axes

Scheme Extension: Transforms, Modifying Models

Action: Creates a transform that takes an object from model space to the space defined by the new origin and axes.

Filename: kern/kern_scm/tran_scm.cxx

APIs: None

Syntax: (**transform:axes** origin–position x–axis y–axis)

Arg Types:

origin–position	position
x–axis	gvector
y–axis	gvector

Returns: transform

Errors: None

Description: After applying this transform to an entity, the entity has the same relationship to the working coordinate system that it had to the model coordinate system.

origin–position specifies the origin of the new coordinate system.

x–axis specifies the *x*–axis gvector.

y–axis specifies the *y*–axis gvector. The *y*–axis is perpendicular to the *x*–axis.

Limitations: None

Example:

```
; transform:axes
; Create a WCS.
(define wcs1
    (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 1 0)))
;; wcs1
; Set the color of the wcs.
(entity:set–color wcs1 6)
;; ()
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 15 20 15)))
;; block1
; OUTPUT Original

; Transform the solid block from model space to
; the created working space as defined by the WCS.
(define transform (entity:transform block1
    (transform:axes (position 0 0 0)
    (gvector –8 0 0) (gvector 0 6 0))))
;; transform
; OUTPUT Result
```

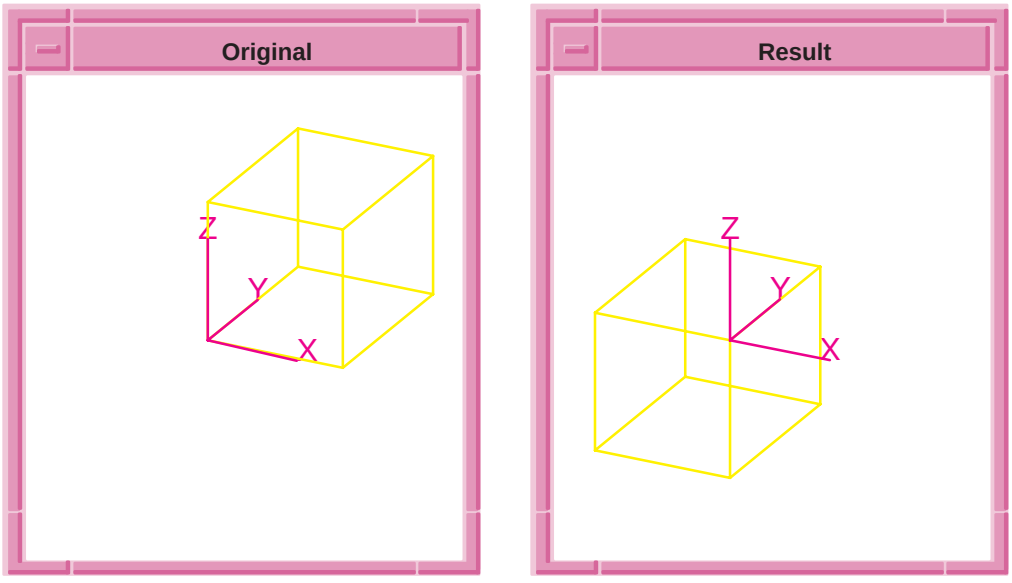

Figure 15-2. transform:axes

transform:compose

Scheme Extension: Transforms, Modifying Models

Action: Concatenates two transforms.

Filename: kern/kern_scm/tran_scm.cxx

APIs: None

Syntax: `(transform:compose transform1 transform2)`

Arg Types:		
	transform1	transform
	transform2	transform

Returns: transform

Errors: None

Description: Concatenates two transforms. Permits the creation of more complex transforms from simpler transforms such as reflection, rotation, scaling, and translation.

transform1 and transform2 arguments specify the transformations to be concatenated.

Limitations: None

Example:

```
; transform:compose
; Create a WCS.
(define wcs1 (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 1 0)))
;; wcs1
; Set a color for the wcs.
(entity:set–color wcs1 6)
;; ()
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 5 10 15)))
;; block1
; OUTPUT Original

; Transform the solid block from model space to
; the created working space as defined by the WCS.
; Concatenate two transforms.
(define comp1 (transform:compose
    (transform:rotation (position 0 0 0)
    (gvector –3 0 0) 45)
    (transform:axes (position 0 0 0)
    (gvector –8 0 0) (gvector 0 6 0))))
;; comp1
; comp1
(define transform (entity:transform block1 comp1))
;; transform
; OUTPUT Result
```

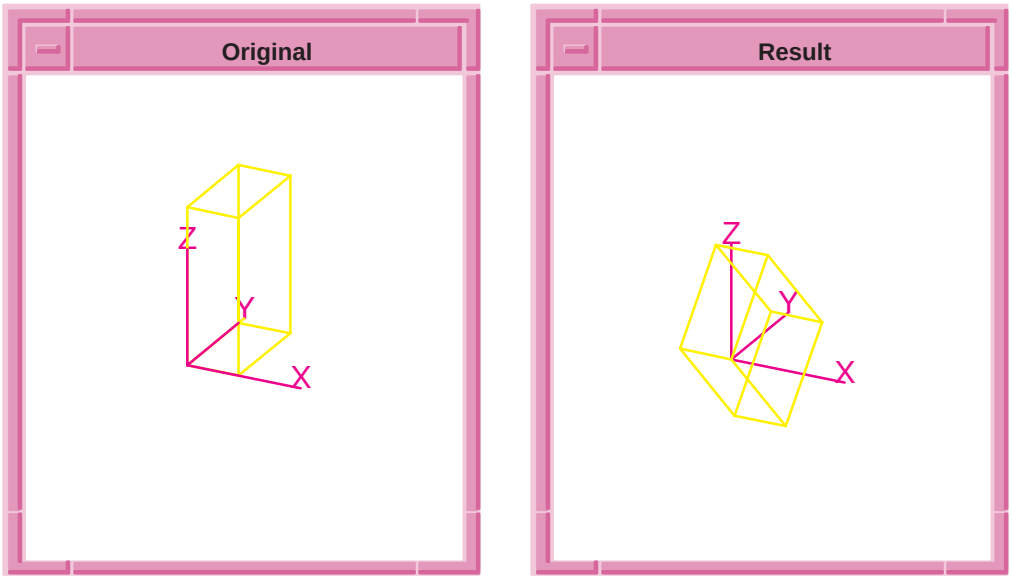

Figure 15-3. transform:compose

transform:copy

Scheme Extension: Transforms, Modifying Models

Action: Copies a transform.

Filename: kern/kern_scm/tran_scm.cxx

APIs: None

Syntax: `(transform:copy transform)`

Arg Types: transform — transform

Returns: transform

Errors: None

Description: Copies a transform into a duplicate but distinct transform.

transform specifies the transformation to be copied.

Limitations: None

Example:

```
; transform:copy
; Create a WCS.
(define wcs1
    (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 1 0)))
;; wcs1
; Set a color for the wcs.
(entity:set–color wcs1 6)
;; ()
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 5 10 15)))
;; block1
; OUTPUT Original

; Apply a new transform by copying the existing
; transform and adding a translation.
(define transform1
    (transform:translation (gvector 4 0 0)))
;; transform1
(define copy1 (transform:copy transform1))
;; copy1
(define entity1
    (entity:transform block1 transform1))
;; entity1
(roll –1)
;; –1
(define entity1
    (entity:transform block1 copy1))
;; entity1
; OUTPUT Result
```

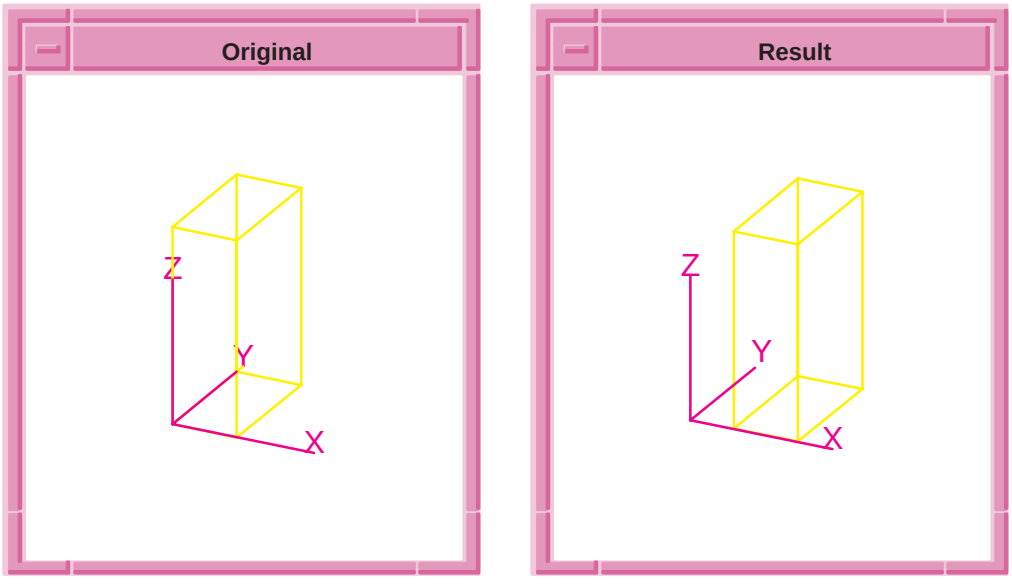

Figure 15-4. transform:copy

transform:identity

Scheme Extension: Transforms, Modifying Models

Action: Creates an identity transform.

Filename: kern/kern_scm/tran_scm.cxx

APIs: None

Syntax: (**transform:identity**)

Arg Types: None

Returns: transform

Errors: None

Description: This extension creates an identity transform with the following rotation matrix and translation elements:

```
    |   1   0   0   |
R   =   |   0   1   0   |   T   =   (0 0 0)
        |   0   0   1   |
```

transform:identity is useful as a starting point and for operations that require a transform as an input, but nothing is required to be accomplished.

Limitations: None

Example:
```
; transform:identity
; Create an identity transform.
(transform:identity)
;; #[transform 1074574992]
```

transform:inverse

Scheme Extension: Transforms, Modifying Models

Action: Creates an inverse transform.

Filename: kern/kern_scm/tran_scm.cxx

APIs: None

Syntax: (**transform:inverse** transform)

Arg Types:	transform	transform

Returns: transform

Errors: None

Description: Refer to Action.

transform specifies the transformation to be inversed.

Limitations: None

Example:

```
; transform:inverse
; Create a transform, and then create its inverse.
(define transform1 (transform:axes
    (position 0 0 0) (gvector 1 0 0)
    (gvector 0 0 1)))
;; transform1
(define inverse1 (transform:inverse transform1))
;; inverse1
; Create a WCS.
(define wcs1
    (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 1 0)))
;; wcs1
; Set a color for the wcs.
(entity:set–color wcs1 6)
;; ()
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 5 10 15)))
;; block1
(define transform
    (entity:transform block1 transform1))
;; transform
; OUTPUT Original

(define transform2
    (entity:transform block1 inverse1))
;; transform2
; OUTPUT Result
```

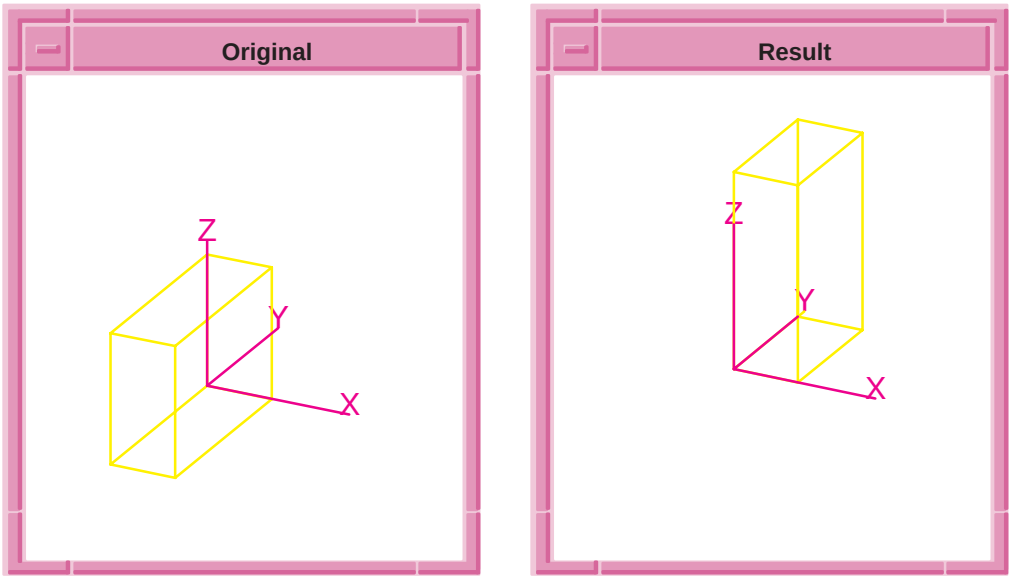

Figure 15-5. transform:inverse

transform:print

Scheme Extension: Transforms, Modifying Models

Action: Prints a transform.

Filename: kern/kern_scm/tran_scm.cxx

APIs: None

Syntax: `(transform:print transform)`

Arg Types: transform transform

Returns: transform

Errors: None

Description: Prints the details of a transform.

transform specifies the transform to use.

Limitations: None

Example:

```
; transform:print
; create a scaling transform
(define s (transform:scaling 1 2 3))
;; s
; create a rotation transform
(define r (transform:rotation
    (position 0 0 0) (gvector 0 0 1) 30))
;; r
; create a translation transform
(define t (transform:translation (gvector 3 4 5)))
;; t
; compose the three
(define sr (transform:compose s r))
;; sr
(define srt (transform:compose
    (transform:compose s r) t))
;; srt
; print the result
(transform:print srt)
;; #[transform 72496304]
; rotation no reflection shear not identity
; translation part:
; 3.000000  4.000000  5.000000
; affine part:
; 0.231455  0.133631  0.000000
; –0.267261 0.462910  0.000000
; 0.000000  0.000000  0.801784
; scaling part:
; 3.741657
```

transform:reflection

Scheme Extension: Transforms, Modifying Models

Action: Creates a transform to mirror an object through an axis.

Filename: kern/kern_scm/tran_scm.cxx

APIs: None

Syntax: (**transform:reflection** plane–position plane–direction)

Arg Types:		
	plane–position	position
	plane–direction	gvector

Returns: transform

Errors: None

Description: Refer to Action.

plane–position specifies the location to mirror an object.

plane–direction specifies the normal of the mirror in the plane.

Limitations: None

Example:

```
; transform:reflection
; Define a working coordinate system.
(define wcs1
    (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 1 0)))
;; wcs1
; Set a color for the wcs.
(entity:set–color wcs1 6)
;; ()
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 5 10 15)))
;; block1
; OUTPUT Original

; Create a transform and then create its reflection.
(define transform1
    (transform:reflection
    (position 0 0 0) (gvector –1 0 0)))
;; transform1
; Apply the transform to reflect the block.
(define transform
    (entity:transform block1 transform1))
;; transform
; OUTPUT Result
```

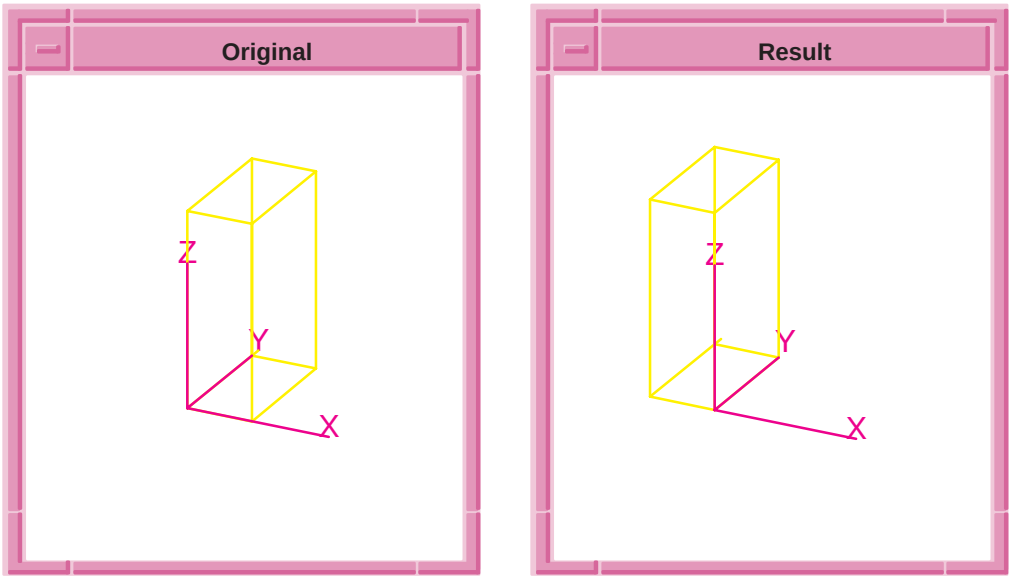

Figure 15-6. transform:reflection

transform:rotation

Scheme Extension: Transforms, Modifying Models

Action: Creates a transform to rotate an object about an axis.

Filename: kern/kern_scm/tran_scm.cxx

APIs: None

Syntax:
```
(transform:rotation origin–position
    axis–direction angle)
```

Arg Types:		
	origin–position	position
	axis–direction	gvector
	angle	real

Returns: transform

Errors: None

Description: Refer to Action.

origin–position specifies the start location of the rotate.

axis–direction specifies the axis direction of rotation. The right–hand rule determines the direction of rotation.

angle specifies the angle in degrees to rotate the object.

Limitations: None

Example:

```
; transform:rotation
; Define a working coordinate system.
(define wcs1
    (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 1 0)))
;; wcs1
; Set a color for the wcs.
(entity:set–color wcs1 6)
;; ()
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 5 10 15)))
;; block1
; OUTPUT Original

; Create a transform and then create its reflection.
(define transform1
    (transform:rotation
    (position 0 0 0) (gvector –3 0 0) 45))
;; transform1
; Apply the transform to reflect the block.
(define transform
    (entity:transform block1 transform1))
;; transform
; OUTPUT Result
```

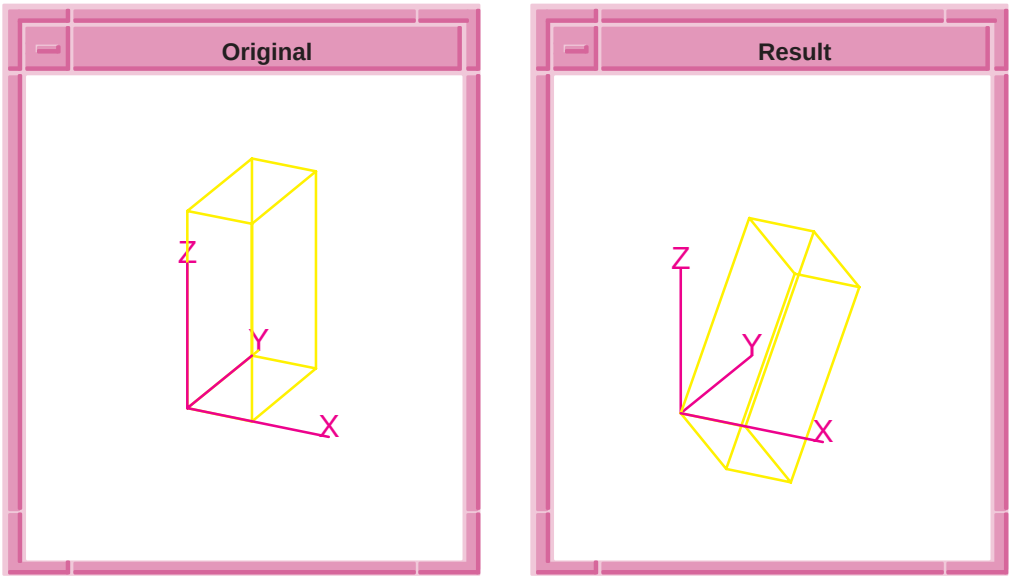

Figure 15-7. transform:rotation

transform:scaling

Scheme Extension: Transforms, Modifying Models

Action: Creates a scaling transform.

Filename: kern/kern_scm/tran_scm.cxx

APIs: None

Syntax: `(transform:scaling x–scale [y–scale z–scale])`

Arg Types:

x–scale	real
y–scale	real
z–scale	real

Returns: transform

Errors: None

Description: Although the extension accepts a scale of 0 or less, only a positive scale factor is recommended.

Uniform scaling is recommended, whereby only the x–scale term is supplied. All three components of the position, of the gvector, and dimensions along the axes are multiplied by the same x–scale factor.

x–scale specifies the scaling factor for the *x*–axis direction.

y–scale is an optional argument that specifies the scaling factor for the *y*–axis direction.

z–scale is an optional argument that specifies the scaling factor for the *z*–axis direction.

Limitations: None

Example:

```
; transform:scaling
; Define a working coordinate system.
(define wcs1
    (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 1 0)))
;; wcs1
; Set a color for the wcs.
(entity:set–color wcs1 6)
;; ()
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 15 20 15)))
;; block1
; OUTPUT Original

; Create a transform and then create its reflection.
(define transform1
    (transform:scaling 0.75 0.75 0.25))
;; transform1
; Apply the transform to reflect the block.
(define transform
    (entity:transform block1 transform1))
;; transform
; OUTPUT Result
```

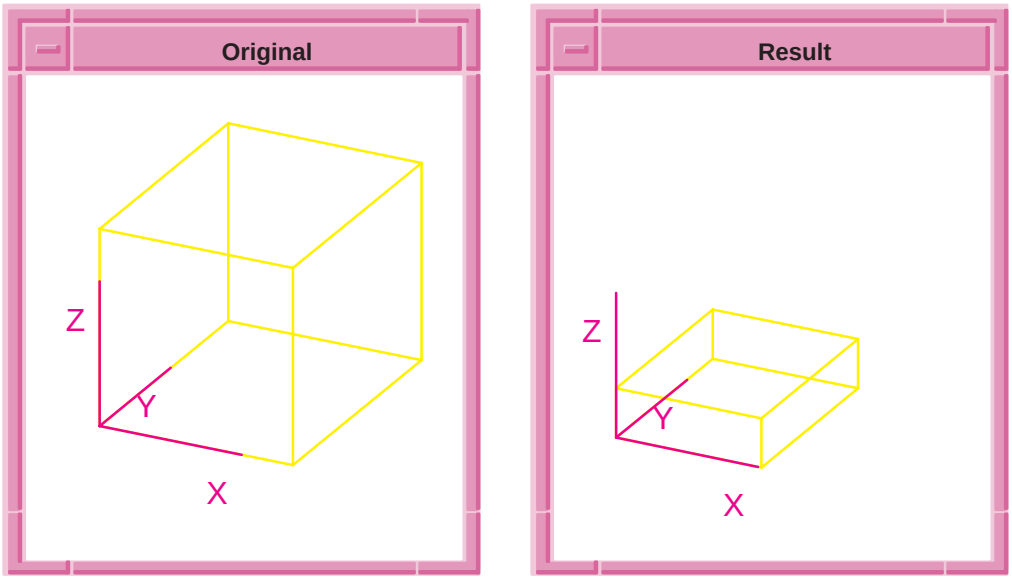

Figure 15-8. transform:scaling

transform:translation

Scheme Extension: Transforms, Modifying Models

Action: Creates a translation transform.

Filename: kern/kern_scm/tran_scm.cxx

APIs: None

Syntax: `(transform:translation gvector)`

Arg Types: gvector gvector

Returns: transform

Errors: None

Description: Refer to Action.

gvector translates an object based on the supplied direction vector.

Limitations: None

Example:

```
; transform:translation
; Define a working coordinate system.
(define wcs1
    (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 1 0)))
;; wcs1
; Set a color for the wcs.
(entity:set–color wcs1 6)
;; ()
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 15 20 15)))
;; block1
; OUTPUT Original

; Create a transform and then create its reflection.
(define transform1
    (transform:translation (gvector 10 12.5 0)))
;; transform1
; Apply the transform to reflect the block.
(define transform
    (entity:transform block1 transform1))
;; transform
; OUTPUT Result
```

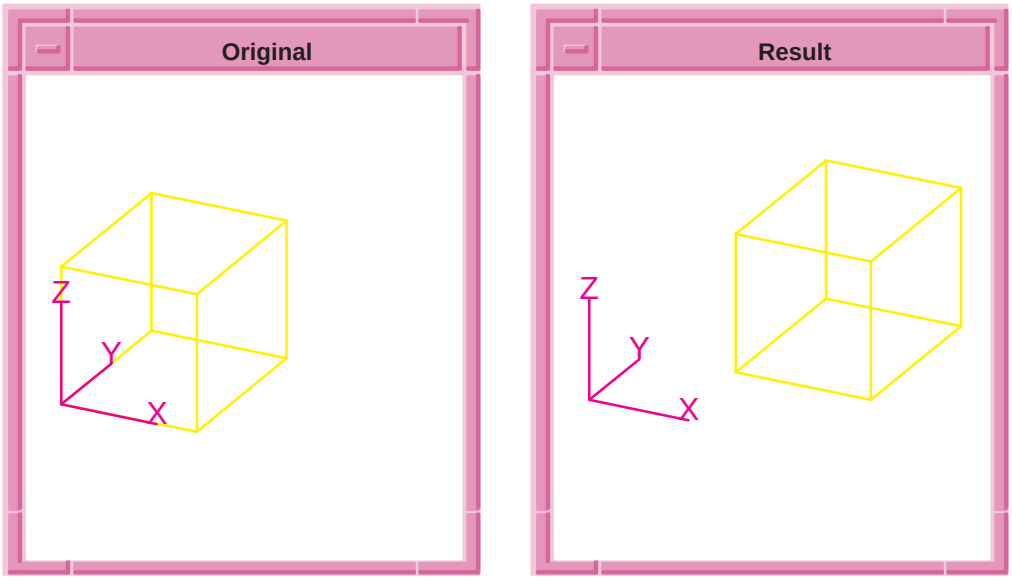

Figure 15-9. transform:translation

transform?

Scheme Extension: Transforms, Modifying Models

Action: Determines if a Scheme object is a transform.

Filename: kern/kern_scm/tran_scm.cxx

APIs: None

Syntax: `(transform? object)`

Arg Types: object scheme–object

Returns: boolean

Errors: None

Description: Refer to Action.

object specifies the scheme–object that has to be queried for a transform.

Limitations: None

Example:

```
; transform?
; Create a transform.
(define transform1
    (transform:translation
    (gvector 1 0 0)))
;; transform1
; Determine if the transform and its identity
; are actually transforms.
(transform? transform1)
;; #t
(transform? (transform:identity))
;; #t
```

versiontag

Scheme Extension: Persistent ID

Action: Gets the current version tag or creates a new version tag scheme object. Default is the current version.

Filename: kern/kern_scm/version_tag_scm.cxx

APIs: None

Syntax: (**versiontag** [major minor release])

Arg Types:

major	integer
minor	integer
release	integer

Returns: scheme–object

Errors: None

Description: Returns a version tag scheme object.

major specifies the major version number.

minor specifies the minor version number.

release specifies the release number.

Limitations: None

Example:

```
; versiontag
; Query for the current versiontag.
(versiontag)
;; #[Major=7 Minor=0 Point=0 Tag=70000]
; Create a new version tag scheme object.
(versiontag 7 1 0)
;; #[Major=7 Minor=1 Point=0 Tag=70100]
```

vertex:position

Scheme Extension: Physical Properties

Action: Gets the position of a vertex.

Filename: kern/kern_scm/vrtx_scm.cxx

APIs: None

Syntax: (**vertex:position** vertex)

Arg Types: vertex vertex

Returns: position

Errors: None

Description: Returns the position of a vertex.

vertex specifies a vertex.

Limitations: None

Example:

```
; vertex:position
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 35 35 35)))
;; block1
; Get a list of the vertices.
(entity:vertices block1)
;; (#[entity 3 1] #[entity 4 1] #[entity 5 1]
;; #[entity 6 1] #[entity 7 1] #[entity 8 1]
;; #[entity 9 1] #[entity 10 1])
; Determine the position of one vertex.
(vertex:position (entity 7))
;; #[position 35 35 0]
```

vertex?

Scheme Extension: Physical Properties

Action: Determines if a Scheme object is a vertex.

Filename: kern/kern_scm/vrtx_scm.cxx

APIs: None

Syntax: `(vertex? object)`

Arg Types:

object	scheme–object

Returns: boolean

Errors: None

Description: Refer to Action.

object specifies the scheme–object that has to be queried for a vertex.

Limitations: None

Example:

```
; vertex?
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 35 35 35)))
;; block1
; Get a list of the block's vertices.
(entity:vertices block1)
;; (#[entity 3 1] #[entity 4 1] #[entity 5 1]
;; #[entity 6 1] #[entity 7 1] #[entity 8 1]
;; #[entity 9 1] #[entity 10 1])
; Determine if a vertex is actually a vertex.
(vertex? (entity 6))
;; #t
```

wcs

Scheme Extension: Work Coordinate Systems

Action: Creates a work coordinate system.

Filename: kern/kern_scm/wcs_scm.cxx

APIs: api_wcs_create

Syntax: `(wcs origin–pos {x–pos | x–vec} {y–pos | y–vec})`

Arg Types:

origin–pos	position
x–pos	position
x–vec	gvector
y–pos	position
y–vec	gvector

Returns: wcs

Errors: None

Description: The origin–pos, x–pos and y–pos are used to define the *xy*–plane. The *z*–axis is defined by the right hand rule.

origin–pos specifies the center of model space.

x–pos specifies the *x*–axis location.

y–pos specifies the *y*–axis location. The *y*–axis is defined perpendicularly to the *x*–axis in the *xy*–plane.

x–vec specifies the axis in the *x*–direction.

y–vec specifies the axis in the *y*–direction.

Limitations: None

Example:

```
; wcs
; Create a new WCS.
(define wcs1
    (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 1 0)))
;; wcs1
; OUTPUT WCS 1

; Create a second WCS.
(define wcs2
    (wcs (position 30 0 0)
    (position 30 30 0) (position 30 0 10)))
;; wcs2
; Set a color for the wcs.
(entity:set–color wcs1 6)
;; ()
; OUTPUT WCS 2
```

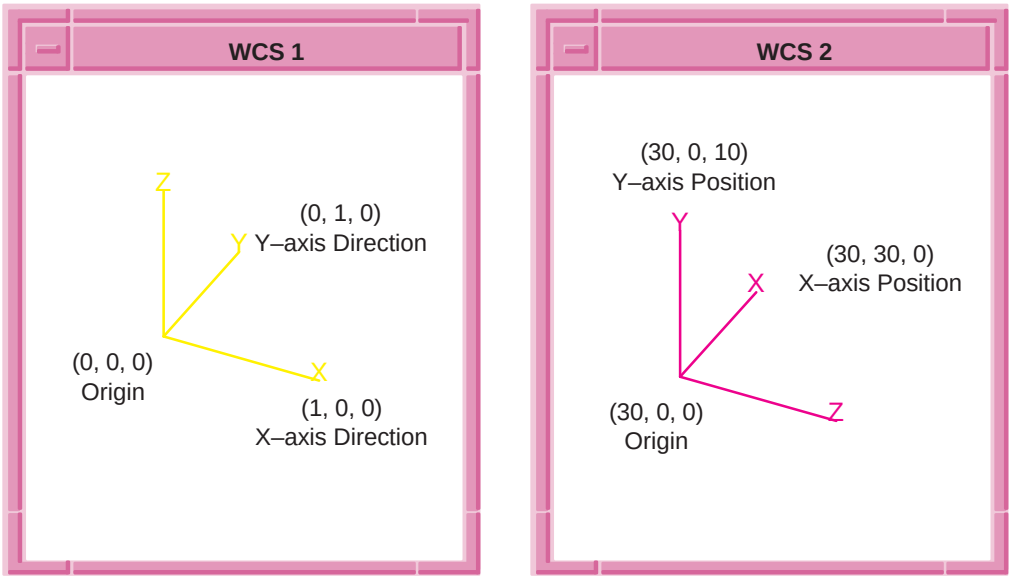

Figure 15-10. wcs

wcs:active

Scheme Extension: Work Coordinate Systems

Action: Gets the active work coordinate system.

Filename: kern/kern_scm/wcs_scm.cxx

APIs: api_wcs_get_active

Syntax: (**wcs:active**)

Arg Types: None

Returns: wcs | boolean

Errors: None

Description: This extension returns the active WCS; otherwise, it returns #f if no WCS is active.

Limitations: None

Example:

```
; wcs:active
; Create a WCS.
(define wcs1 (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 1 0)))
;; wcs1
; Determine if the WCS is active.
(wcs:active)
;; #f
; Set the new WCS to be active.
(wcs:set–active wcs1)
;; ()
; Get the active WCS.
(wcs:active)
;; #[entity 2 1]
```

wcs:from–transform

Scheme Extension: Work Coordinate Systems

Action: Creates a work coordinate system given a transform.

Filename: kern/kern_scm/wcs_scm.cxx

APIs: None

Syntax: (**wcs:from–transform** transform)

Arg Types:

transform	transform

Returns: wcs

Errors: None

Description: Refer to Action.

transform maps from the model space to the new working coordinate system.

Limitations: None

Example:

```
; wcs:from–transform
; Create a new WCS.
(define wcs1
    (wcs (position 0 0 0) (position 5 0 0)
    (position 0 5 0)))
;; wcs1
(define transform1 (transform:rotation
    (position 0 0 0) (gvector –10 0 0) 60))
;; transform1
; Create a new WCS from a transform rotation.
(define wcs2 (wcs:from–transform transform1))
;; wcs2
(define transform2
    (transform:translation (gvector 10 10 20)))
;; transform2
; Create a new WCS from a transform translation.
(define wcs3 (wcs:from–transform transform2))
;; wcs3
; OUTPUT Example
```

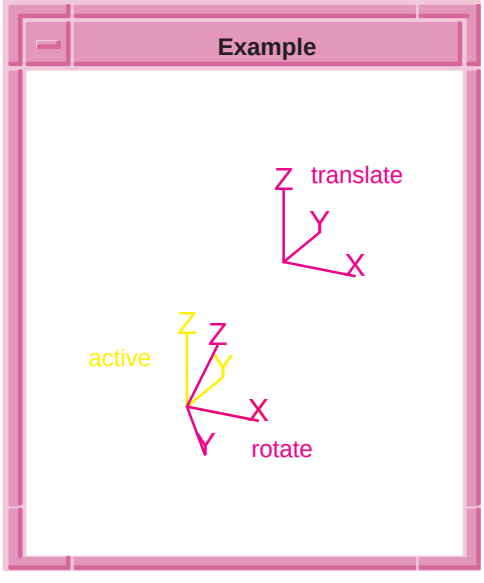

Figure 15-11. wcs:from–transform

wcs:origin

Scheme Extension: Work Coordinate Systems

Action: Gets the origin of the active WCS.

Filename: kern/kern_scm/wcs_scm.cxx

APIs: None

Syntax: (**wcs:origin** [wcs])

Arg Types: wcs wcs

Returns: position

Errors: None

Description: Refer to Action.

wcs is an optional argument that specifies the working coordinate system to search. If wcs is not specified, the active WCS is used.

Limitations: None

Example:

```
; wcs:origin
; Get the origin position of the current WCS.
(wcs:origin)
;; #[position 0 0 0]
; Define a new WCS.
(define wcs1 (wcs (position –10 –10 –10)
    (position 0 10 0) (position 10 0 0)))
;; wcs1
; Get the origin position of the new WCS.
(wcs:origin wcs1)
;; #[position –10 –10 –10]
```

wcs:set–active

Scheme Extension: Work Coordinate Systems

Action: Sets the active work coordinate system to the specified WCS.

Filename: kern/kern_scm/wcs_scm.cxx

APIs: api_wcs_set_active

Syntax: (**wcs:set–active** wcs)

Arg Types: wcs wcs | #f

Returns: unspecified

Errors: None

Description: Refer to Action.

If a wcs is specified, it becomes the active wcs. If boolean #f is specified, the system is returned to its initial state of no active wcs.

Limitations: None

Example:

```
; wcs:set–active
; Define working coordinate system.
(define wcs1 (wcs (position 0 0 0)
    (position 5 0 0) (position 0 5 0)))
;; wcs1
; Set a color for wcs1.
(entity:set–color wcs1 6)
;; ()
; Define a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Set a color for block 1.
(entity:set–color block1 3)
;; ()
; Define a second working coordinate system.
(define wcs2 (wcs (position 15 0 0)
    (position 20 0 0) (position 0 20 0)))
;; wcs2
; OUTPUT WCS1 and Block1

; Set the specified WCS to be the active WCS.
(wcs:set–active wcs2)
;; ()
; Define another solid block in the specified WCS.
(define block2
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block2
; Set a color for the second block.
(entity:set–color block2 1)
;; ()
; OUTPUT WCS2 and Block2
```

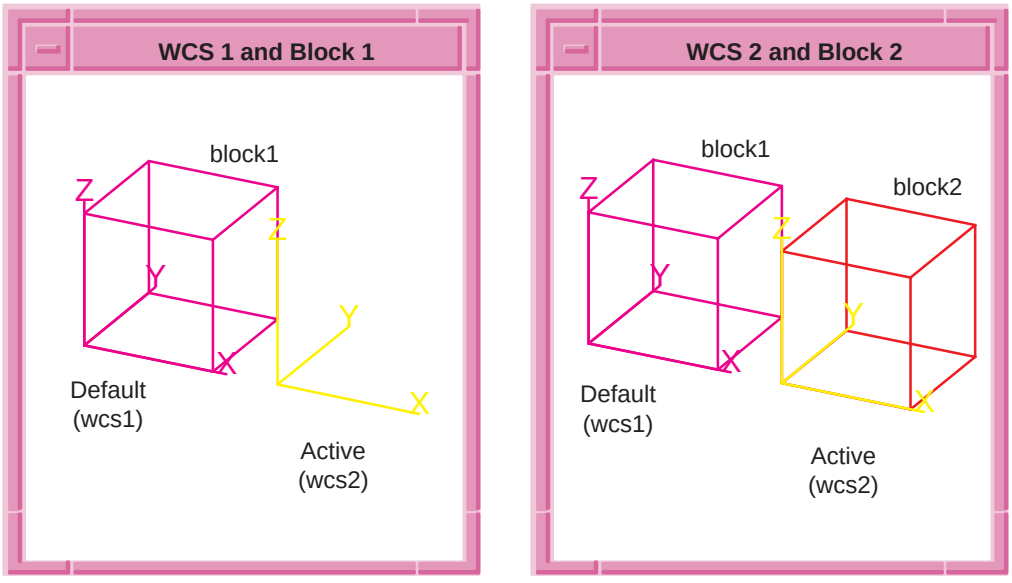

Figure 15-12. wcs:set–active

wcs:to–model–transform

Scheme Extension: Work Coordinate Systems, Transforms

Action: Gets the transform of the active WCS to model space.

Filename: kern/kern_scm/wcs_scm.cxx

APIs: None

Syntax: `(wcs:to-model-transform [wcs])`

Arg Types: wcs wcs

Returns: transform

Errors: None

Description: Refer to Action.

wcs is an optional argument that specifies the WCS to search. If wcs is not specified, the active WCS is used.

Limitations: None

Example:

```
; wcs:to–model–transform
; Find the transform of the current WCS to
; model space.
(define transform1 (wcs:to–model–transform))
;; transform1
; Define a new WCS.
(define wcs1 (wcs (position –10 –10 –10)
    (position 0 10 0) (position 10 0 0)))
;; wcs1
; Find the transform of the new WCS to model space.
(wcs:to–model–transform wcs1)
;; #[transform 1074571672]
```

wcs:to–wcs–transform

Scheme Extension: Work Coordinate Systems, Transforms

Action: Gets the transform from the active WCS to the specified WCS.

Filename: kern/kern_scm/wcs_scm.cxx

APIs: None

Syntax: (**wcs:to–wcs–transform** [wcs])

Arg Types:

wcs	wcs

Returns: transform

Errors: None

Description: Refer to Action.

wcs is an optional argument that specifies the WCS to search. If wcs is not specified, the active WCS is used.

Limitations: None

Example:

```
; wcs:to–wcs–transform
; Get the transform from the active coordinate
; system to the current WCS.
(define transform1 (wcs:to–wcs–transform))
;; transform1
; Define a new WCS.
(define wcs1 (wcs (position 0 0 0)
    (position 0 10 0) (gvector 1 0 0)))
;; wcs1
; Get the transform from the active coordinate
; system to the new WCS.
(wcs:to–wcs–transform wcs1)
;; #[transform 1074571664]
```

wcs:x–axis

Scheme Extension: Work Coordinate Systems

Action: Gets the *x*–direction of the active coordinate system.

Filename: kern/kern_scm/wcs_scm.cxx

APIs: None

Syntax: (**wcs:x–axis** [wcs])

Arg Types: wcs wcs

Returns: gvector

Errors: None

Description: Refer to Action.

wcs is an optional argument that specifies the WCS to search. If wcs is not specified, the active WCS is used.

Limitations: None

Example:

```
; wcs:x–axis
; Find the x–direction of the current WCS.
(wcs:x–axis)
;; #[gvector 1 0 0]
; Define a new WCS.
(define wcs1 (wcs (position 10 10 10)
    (position 10 0 0) (position 0 10 0)))
;; wcs1
; Get the x–direction of the new WCS.
(wcs:x–axis wcs1)
;; #[gvector 0 –0.707106781186548
;; –0.707106781186548]
; Define WCS wcs3.
(define wcs13 (wcs (position 3 2 1) (gvector 0 1 0)
    (position 1 0 0)))
;; wcs13
; Find the x–direction of WCS wcs3.
(wcs:x–axis wcs13)
;; #[gvector 0 1 0]
```

wcs:y–axis

Scheme Extension: Work Coordinate Systems

Action: Gets the *y*–direction of the active coordinate system.

Filename: kern/kern_scm/wcs_scm.cxx

APIs: None

Syntax: (**wcs:y–axis** [wcs])

Arg Types: wcs wcs

Returns: gvector

Errors: None

Description: Refer to Action.

wcs is an optional argument that specifies the WCS to search. If wcs is not specified, the active WCS is used.

Limitations: None

Example:

```
; wcs:y–axis
; Get the y–direction of the current WCS.
(wcs:y–axis)
;; #[gvector 0 1 0]
; Define a new WCS.
(define wcs1 (wcs (position 10 10 10)
    (position 10 0 0) (position 0 10 0)))
;; wcs1
; Get the y–direction of the new WCS.
(wcs:y–axis wcs1)
;; #[gvector –0.816496580927726 0.408248290463863
;; –0.408248290463863]
; Define WCS wcs3.
(define wcs3 (wcs (position 3 2 1) (gvector 0 1 0)
    (position 1 0 0)))
;; wcs3
; Get the y–direction of wcs3.
(wcs:y–axis wcs3)
;; #[gvector –0.894427190999916 0
;; –0.447213595499958]
```

wcs:z–axis

Scheme Extension: Work Coordinate Systems

Action: Gets the *z*–direction of the active coordinate system.

Filename: kern/kern_scm/wcs_scm.cxx

APIs: None

Syntax: **(wcs:z–axis** [wcs])

Arg Types: wcs wcs

Returns: gvector

Errors: None

Description: Refer to Action.

wcs is an optional argument that specifies the WCS to search. If wcs is not specified, the active WCS is used.

Limitations: None

Example:

```
; wcs:z–axis
; Get the z–axis of the current WCS.
(wcs:z–axis)
;; #[gvector 0 0 1]
; Define a new WCS.
(define wcs1 (wcs (position 10 10 10)
    (position 10 0 0) (position 0 10 0)))
;; wcs1
; Get the z–axis of the new WCS.
(wcs:z–axis wcs1)
;; #[gvector 0.577350269189626 0.577350269189626
;; –0.577350269189626]
; Define wcs3.
(define wcs3 (wcs (position 3 2 1) (gvector 0 1 0)
    (position 1 0 0)))
;; wcs3
; Get the z–axis of wcs3.
(wcs:z–axis wcs3)
;; #[gvector –0.447213595499958 0 0.894427190999916]
```

wcs?

Scheme Extension: Work Coordinate Systems

Action: Determines if a Scheme object is a WCS.

Filename: kern/kern_scm/wcs_scm.cxx

APIs: None

Syntax: `(wcs? object)`

Arg Types: object scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if the specified object is a WCS entity; otherwise, it returns #f.

object specifies the scheme–object that has to be queried for a wcs.

Limitations: None

Example:

```
; wcs?
; Create a wcs.
(define wcs1 (wcs (position 0 0 0) (gvector 0 –1 0)
    (gvector 1 0 0)))
;; wcs1
; Determine if the WCS is actually a WCS.
(wcs? wcs1)
;; #t
```

wire–body:planar?

Scheme Extension: Physical Properties

Action: Determines if a Scheme object is a planar wire body.

Filename: kern/kern_scm/ent_scm.cxx

APIs: None

Syntax: `(wire–body:planar? object)`

Arg Types: object scheme–object

Returns: boolean

Errors: None

Description: Refer to Action.

object specifies the scheme–object that has to be queried for a planar wire body.

Limitations: None

Example:

```
; wire–body:planar?
; Create edge 1.
(define edge1 (edge:linear (position –20 –10 0)
    (position 10 –10 0)))
;; edge1
; Create edge:circular 2.
(define edge2 (edge:circular
    (position 10 0 0) 10 270 360))
;; edge2
; Create edge 3.
(define edge3 (edge:linear
    (position 20 0 0) (position 20 20 0)))
;; edge3
; Create edge 4.
(define edge4 (edge:linear
    (position 20 20 0) (position –10 20 0)))
;; edge4
; Create edge:circular 5.
(define edge5 (edge:circular
    (position –10 10 0) 10 90 180))
;; edge5
; Create edge 6.
(define edge6 (edge:linear
    (position –20 10 0) (position –20 –10 0)))
;; edge6
; Create a wire–body from the edges.
(define wirebody1 (wire–body (list
    edge1 edge2 edge3
    edge4 edge5 edge6)))
;; wirebody1
; Determine if the wire–body is actually a wire–body.
(wire–body:planar? wirebody1)
;; #t
```

wire–body?

Scheme Extension: Physical Properties

Action: Determines if a Scheme object is a wire body.

Filename: kern/kern_scm/ent_scm.cxx

APIs: None

Syntax: (**wire–body?** object)

Arg Types: object scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if the specified object is a wire body; otherwise, it returns #f.

object specifies the scheme–object that has to be queried for a wire body.

Limitations: None

Example:

```
; wire–body?
; Create an edge.
(define edge1 (edge:circular
    (position 0 0 0) 25 180 270))
;; edge1
; Check to see if the edge is a wire–body.
(wire–body? edge1)
;; #f
; Create a wire–body from the edge.
(define wirebody1 (wire–body edge1))
;; wirebody1
; Check to see if the wire–body is a wire–body.
(wire–body? wirebody1)
;; #t
```

wire:closed?

Scheme Extension: Physical Properties

Action: Determines if a Scheme object is a closed wire.

Filename: kern/kern_scm/ent_scm.cxx

APIs: None

Syntax: (**wire:closed?** object)

Arg Types: object scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if the specified object is a wire body; otherwise, it returns #f.

object specifies the scheme–object that has to be queried for a closed wire.

Limitations: None

Example:

```
; wire:closed?
; Create an edge.
(define edge1 (edge:circular
    (position 0 0 0) 25 180 270))
;; edge1
; Check to see if the edge is a wire–body.
(wire–body? edge1)
;; #f
; Create a wire–body from the edge.
(define wirebody1 (wire–body edge1))
;; wirebody1
; Check to see if the wire–body is a wire–body.
(wire:closed? wirebody1)
; Entity is not a wire.
;; #f
```

wire:planar?

Scheme Extension: Physical Properties

Action: Determines if a Scheme object is a planar wire.

Filename: kern/kern_scm/ent_scm.cxx

APIs: None

Syntax: (**wire:planar?** object)

Arg Types:	object	scheme–object

Returns: boolean

Errors: None

Description: Refer to Action.

object specifies the scheme–object that has to be queried for a planar wire.

Limitations: None

Example:

```
; wire:planar?
; Create edge 1.
(define edge1 (edge:linear (position –20 –10 0)
    (position 10 –10 0)))
;; edge1
; Create edge:circular 2.
(define edge2 (edge:circular
    (position 10 0 0) 10 270 360))
;; edge2
; Create edge 3.
(define edge3 (edge:linear
    (position 20 0 0) (position 20 20 0)))
;; edge3
; Create edge 4.
(define edge4 (edge:linear
    (position 20 20 0) (position –10 20 0)))
;; edge4
; Create edge:circular 5.
(define edge5 (edge:circular
    (position –10 10 0) 10 90 180))
;; edge5
; Create edge 6.
(define edge6 (edge:linear
    (position –20 10 0) (position –20 –10 0)))
;; edge6
; Create a wire–body from the edges.
(define wirebody1 (wire–body (list
    edge1 edge2 edge3
    edge4 edge5 edge6)))
;; wirebody1
; Determine if the wire–body is actually a wire–body.
(wire:planar? wirebody1)
;; #f
(wire:planar? (car (entity:wires wirebody1)))
;; #t
```

wire?

Scheme Extension: Physical Properties

Action: Determines if a Scheme object is a wire.

Filename: kern/kern_scm/ent_scm.cxx

APIs: None

Syntax: `(wire? object)`

Arg Types: object scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if the object is a wire; otherwise, it returns #f. (Use entity:wires to extract a list of wires from a wire body.)

object specifies the scheme–object that has to be queried for a wire.

Limitations: None

Example:

```
; wire?
; Create an edge.
(define edge1 (edge:circular
    (position 0 0 0) 25 180 270))
;; edge1
; Check to see if the edge is a wire.
(wire? edge1)
;; #f
; Create a wire–body from the edge.
(define body1 (wire–body edge1))
;; body1
; Check to see if the wire–body is a wire.
(wire? body1)
;; #f
; Create a wire from the wire–body.
(define wires1 (entity:wires body1))
;; wires1
; Check to see if the wire is a wire.
(wire? (car wires1))
;; #t
```