Chapter 2.
Scheme Extensions Aa thru Lz

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.

background

Scheme Extension: Backgrounds and Foregrounds

Action: Creates a new background.

Filename: rbase/rnd_scm/bkgd_scm.cxx

APIs: api_rh_create_background

Syntax: `(background type)`

Arg Types:	type	string

Returns: background

Errors: None

Description: Creates a background shader entity for the given part. A background is an entity that specifies a shader and its defining characteristics. The extension render:set–background assigns the shader to the part.

Note *Only plain uniform color backgrounds are displayed, including* "None", "plain", "graduated", *and* "clouds".

type specifies a string defining the background.

Limitations: None

Example:

```
; background
; Create a “clouds” background.
(define back1 (background ”clouds”))
;; back1
; Set the current background to be rendered.
(render:set–background back1)
;; ()
; Display the background.
(render)
;; ()
; OUTPUT Example
```

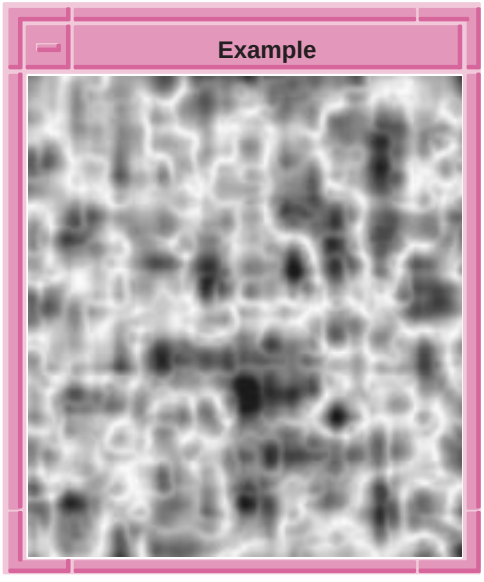

Figure 2-1. background

background:props

Scheme Extension: Backgrounds and Foregrounds

Action: Gets the properties of a background.

Filename: rbase/rnd_scm/bkgd_scm.cxx

APIs: api_rh_get_background_args

Syntax: `(background:props background)`

Arg Types: background background

Returns: (pair . . .)

Errors: None

Description: This extension returns a list containing pairs of argument types and their present values. The car element of the pair list returned is the property and the cdr of the pair list returned is its value.

background specifies a background entity.

Limitations: None

Example:

```
; background:props
; Create a “graduated” background.
(define back1 (background ”graduated”))
;; back1
; Get the “graduated” background properties.
(background:props back1)
;; ((”top color” . #[color 0 0 0])
;; (”bottom color” . #[color 1 1 1]))
; Create a “clouds” background.
(define back2 (background ”clouds”))
;; back2
; Get the “clouds” background properties.
(background:props back2)
;; ((”scale” . 1)
;; (”background color” . #[color 0 0 1])
;; (”clouds color” . #[color 1 1 1]) (”detail” . 2))
; Create a “plain” background.
(define back3 (background ”plain”))
;; back3
; Get the “plain” background properties.
(background:props back3)
;; ((”color” . #[color 0 0 0]))
```

background:set–prop

Scheme Extension: Backgrounds and Foregrounds

Action: Sets the properties of a background.

Filename: rbase/rnd_scm/bkgd_scm.cxx

APIs: api_rh_get_background_args, api_rh_set_background_arg

Syntax: (**background:set-prop** background name value)

Arg Types:

background	background
name	string
value	real | integer | color | string

Returns: unspecified

Errors: None

Description: Refer to Action.

background specifies a background entity.

name specifies the name of the property. The properties include "None", "plain", "graduated", and "clouds".

"none" contains no background (black).

"plain" is a plain, uniform background with these default values:

color . . . . . #[color 0 0 0]

"graduated" is a graduation between two colors. The image is in one color at the top and graduates to the other color at the bottom. The default values are:

top color . . . . . #[color 0 0 0]
bottom color . . . . . #[color 1 1 1]

"clouds" provides a cloudy appearance. The complexity of the cloud's outline is specified by the detail property; the larger the value, the more fine the detail. The scale property controls the size of the clouds; the larger the value, the larger the cloud size. The default values are:

scale . . . . . 1.0 (real)
background color . . . . . #[color 0 0 1]
clouds color . . . . . #[color 1 1 1]
detail . . . . . 2 (int)

value specifies the value of the property.

Limitations: None

Example:

```
; background:set–prop
; Create a “graduated” background.
(define back1 (background ”graduated”))
;; back1
; Get the “graduated” background properties.
(background:props back1)
;; ((”top color” . #[color 0 0 0])
;; (”bottom color” . #[color 1 1 1]))
; Set a “graduated” background property.
(background:set–prop back1 ”top color”
    (color:rgb 0 1 0))
;; ()
(render:set–background back1)
;; ()
; Display the background.
(render)
;; ()
; OUTPUT Graduated

; Create a “clouds” background.
(define back2 (background ”clouds”))
;; back2
; Get the “clouds” background properties.
(background:props back2)
;; ((”scale” . 1)
;; (”background color” . #[color 0 0 1])
;; (”clouds color” . #[color 1 1 1]) (”detail” . 2))
; Set a “clouds” background property.
(background:set–prop back2 ”clouds color”
    (color:rgb 0 1 0))
;; ()
(render:set–background back2)
;; ()
; Display the background.
(render)
;; ()
; OUTPUT clouds
```

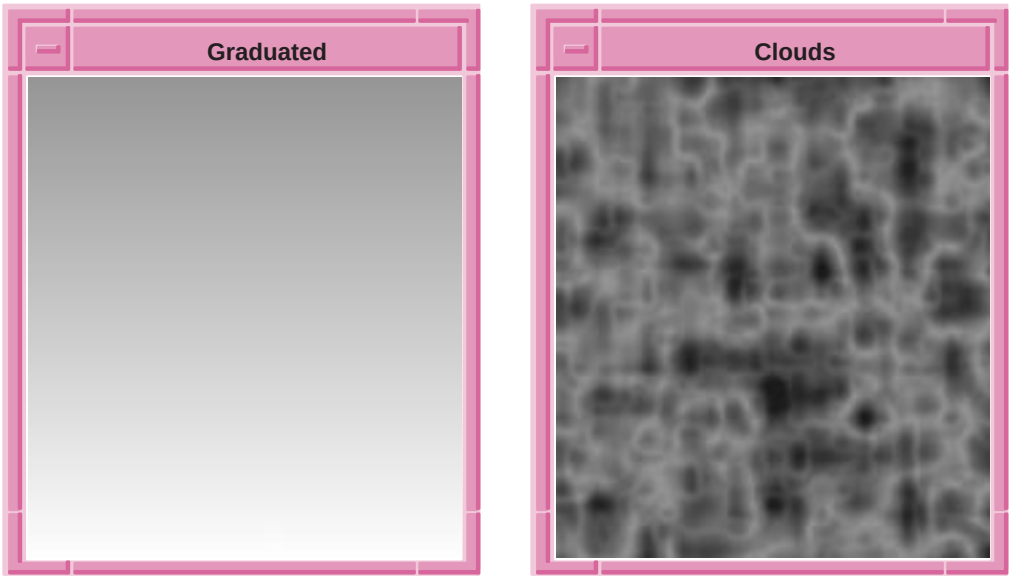

Figure 2-2. background:set–prop

background:type

Scheme Extension: Backgrounds and Foregrounds

Action: Gets the type of background.

Filename: rbase/rnd_scm/bkgd_scm.cxx

APIs: api_rh_get_background_args

Syntax: `(background:type background)`

Arg Types: background — background

Returns: string

Errors: None

Description: This extension returns the type of the background in a string. Refer to background:set–prop to change settings.

background specifies a background entity.

Limitations: None

Example:

```
; background:type
; Create background 1.
(define back1 (background ”graduated”))
;; back1
; Create background 2.
(define back2 (background ”clouds”))
;; back2
; Get the type of background 1.
(background:type back1)
;; ”graduated”
; Get the type of background 2.
(background:type back2)
;; ”clouds”
```

background:types

Scheme Extension: Backgrounds and Foregrounds

Action: Gets the list of available background types.

Filename: rbase/rnd_scm/bkgd_scm.cxx

APIs: api_rh_get_background_types

Syntax: (**background:types**)

Arg Types: None

Returns: (string ...)

Errors: None

Description: This extension returns a list of all valid background types.

Limitations: None

Example:

```
; background:types
; Get a list of available background types.
(background:types)
;; (”image” ”plain” ”none” ”graduated” ”clouds”)
```

background?

Scheme Extension: Backgrounds and Foregrounds

Action: Determines if a Scheme object is a background.

Filename: rbase/rnd_scm/bkgd_scm.cxx

APIs: None

Syntax: `(background? object)`

Arg Types:

object	scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if a Scheme object is a background; otherwise, it returns #f.

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

Limitations: None

Example:

```
; background?
; Create a background.
(define back1 (background ”graduated”))
;; back1
; Determine if the entity is actually a background.
(background? back1)
;; #t
```

color:rgb

Scheme Extension: Colors

Action: Creates a red–green–blue color object.

Filename: rbase/rnd_scm/rgb_scm.cxx

APIs: None

Syntax: `(color:rgb red green blue)`

Arg Types:

red	real
green	real
blue	real

Returns: color

Errors: None

Description: Specify red, green, and blue color components with normalized real numbers, ranging from 0 to 1.

Limitations: None

Example:
```
; color:rgb
; Create a blue color object by specifying
; the red, green, and blue values.
(define color (color:rgb 0 0 1))
;; color
```

color:rgb?

Scheme Extension: Colors

Action: Determines if a Scheme object is a color.

Filename: rbase/rnd_scm/rgb_scm.cxx

APIs: None

Syntax: (**color:rgb?** object)

Arg Types:	object	scheme–object

Returns: boolean

Errors: None

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

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

Limitations: None

Example:
```
; color:rgb?
; Create a blue color object by specifying
; the red, green, and blue values.
(define color (color:rgb 0 0 1))
;; color
(color:rgb? color)
;; #t
; Determine if the object is a color.
(color:rgb? (env:default–color))
;; #t
```

entity:material

Scheme Extension: Materials

Action: Gets the material associated with a geometric entity.

Filename: rbase/rnd_scm/mat_scm.cxx

APIs: api_rh_get_material

Syntax: (**entity:material** entity)

Arg Types: entity entity

Returns: material

Errors: None

Description: This extension returns the material (if any) associated with the entity; otherwise it returns #f.

entity specifies the entity to be queried.

Limitations: None

Example:

```
; entity:material
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 5 30 0) 8))
;; cyl1
; Determine if the cylinder has a material.
(entity:material cyl1)
;; #f
; Create a material.
(define matter1 (material))
;; matter1
; Set the material.
(entity:set–material cyl1 matter1)
;; ()
; Get the material assigned to the cylinder.
(define material (entity:material cyl1))
;; material
```

entity:material–color

Scheme Extension: Materials, Colors

Action: Gets the color of the material associated with a geometric entity.

Filename: rbase/rnd_scm/mat_scm.cxx

APIs: api_rh_get_material_color

Syntax: `(entity:material–color entity)`

Arg Types: entity — entity

Returns: color

Errors: Returns boolean #f if the material or material–color cannot be found.

Description: This extension returns the material color (if any) associated with the entity.

entity is the entity being queried.

Limitations: None

Example:

```
; entity:material–color
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 30 30 0) 20))
;; cyl1
; Determine if the cylinder has a material.
(entity:material cyl1)
;; #f
; Create a material.
(define matter1 (material))
;; matter1
; Set the material.
(entity:set–material cyl1 matter1)
;; ()
; Set the color of the material.
(entity:set–material–color cyl1
    (color:rgb 1 1 1))
;; ()
; Get the color of the material.
(entity:material–color cyl1)
;; #[color 1 1 1]
```

entity:material–reflection

Scheme Extension: Materials, Reflectance

Action: Gets the reflection properties of the material associated with a geometric entity.

Filename: rbase/rnd_scm/mat_scm.cxx

APIs: api_rh_get_material_reflection

Syntax: `(entity:material–reflection entity)`

Arg Types: entity — entity

Returns: (real, real, real, real)

Errors: Returns boolean #f if the material or material–reflection cannot be found.

Description: This extension returns the material reflection properties (if any) associated with the entity. The first three real values returned are the ambient reflection, diffuse reflection, and specular reflection, respectively, which range from 0 to 1. The last real value returned is the exponent.

entity specifies an entity to be queried.

Limitations: None

Example:

```
; entity:material–reflection
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 30 30 0) 20))
;; cyl1
; Determine if the cylinder has a material.
(entity:material cyl1)
;; #f
; Create a material.
(define matter1 (material))
;; matter1
; Set the material.
(entity:set–material cyl1 matter1)
;; ()
; Get the default material reflection properties.
(entity:material–reflection cyl1)
;; (1 0.75 0.5 10)
```

entity:material–texture

Scheme Extension: Materials, Textures

Action: Gets the texture file name in a material associated with a geometric entity.

Filename: rbase/rnd_scm/mat_scm.cxx

APIs: api_rh_get_material_texture

Syntax: `(entity:material–texture entity)`

Arg Types: entity entity

Returns: string

Errors: Returns boolean #f if the material or material–texture cannot be found.

Description: This extension sets the texture file name in a material associated with the entity. The argument string specifies the file name of the texture attached to the material.

entity specifies an entity to which the material is attached.

Limitations: None

Example:

```
; entity:material–texture
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 10 0 30) 20))
;; cyl1
; Determine if the cylinder has a material.
(entity:material cyl1)
;; #f
; Create a material.
(define matter1 (material))
;; matter1
; Set the material.
(entity:set–material cyl1 matter1)
;; ()
; Set the material's texture.
(entity:set–material–texture cyl1 "brick.bmp")
; Get the material's new texture.
;; ()
(entity:material–texture cyl1)
;; "brick.bmp"
```

entity:material–transparency

Scheme Extension: Materials, Transparency

Action: Gets the transparency property in the material associated with a geometric entity.

Filename: rbase/rnd_scm/mat_scm.cxx

APIs: api_rh_get_material_transp

Syntax: `(entity:material–transparency entity)`

Arg Types: entity entity

Returns: real

Errors: Returns the string entity:material–transparency: operation unsuccessful.

Description: This extension gets the transparency property (if any) in the material associated with a geometric entity. The return value real specifies the transparency factor, which ranges from 0 (transparent) to 1 (opaque).

entity specifies an entity to query.

Limitations: None

Example:

```
; entity:material–transparency
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 30 5 15) 20))
;; cyl1
; Determine if the cylinder has a material.
(entity:material cyl1)
;; #f
; Create a material.
(define matter1 (material))
;; matter1
; Set the material.
(entity:set–material cyl1 matter1)
;; ()
; Set the transparency property of the material.
(entity:set–material–transparency cyl1 0.8)
;; ()
; Get the material's transparency.
(entity:material–transparency cyl1)
;; 0.8
```

entity:render–sides

Scheme Extension: Physical Properties, Rendering Control

Action: Gets the sidedness of an entity for rendering purposes.

Filename: rbase/rnd_scm/avh_scm.cxx

APIs: api_rh_get_sidedness

Syntax: `(entity:render–sides entity)`

Arg Types: entity — entity

Returns: integer

Errors: None

Description: This extension returns an integer representing entity sidedness: 0 for undefined sidedness, 1 for single–sided, and 2 for double–sided. By default, a newly created entity has undefined render sidedness.

entity specifies an entity to query.

Limitations: None

Example:

```
; entity:render–sides
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 8 8 0) 20))
;; cyl1
; Set the sidedness for the cylinder.
(entity:set–render–sides cyl1 2)
;; ()
; Get the sidedness of the cylinder.
(entity:render–sides cyl1)
;; 2
```

entity:set–material

Scheme Extension: Entity, Materials

Action: Sets the material attribute of an entity.

Filename: rbase/rnd_scm/mat_scm.cxx

APIs: api_rh_set_material

Syntax: `(entity:set–material entity–list material)`

Arg Types:

entity–list	entity | (entity ...)
material	material | #f

Returns: unspecified

Errors: None

Description: Refer to Action.

entity–list specifies an entity or entity list to which the material is to be attached.

material specifies the type of material entity to attach to the entity. If the argument material is #f, then the material attribute is removed from the entity or entity list.

Limitations: None

Example:

```
; entity:set–material
; Create a material.
(define matter1 (material))
;; matter1
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 15 10 5)))
;; block1
; Assign the material to the block.
(entity:set–material block1 matter1)
;; ()
; Remove the material from the block.
(entity:set–material block1 #f)
;; ()
```

entity:set–material–color

Scheme Extension: Materials, Colors

Action: Sets the color of the material associated with a geometric entity.

Filename: rbase/rnd_scm/mat_scm.cxx

APIs: api_rh_set_material_color

Syntax: (**entity:set–material–color** entity rgb–color)

Arg Types:		
	entity	entity
	rgb–color	color

Returns: unspecified

Errors: None

Description: This extension sets the material color (if any) to be associated with the entity. This changes the material's color type to "plain". It does not affect its displacement, reflection, or transparency types or properties.

When a new “plain” color type material is created, the default reflection type is “phong”. It has the following properties:

“ambient factor” 1
“diffuse factor” 0.75
“specular factor” 0.5
“exponent” 10
“specular color” . . . . . #[color 1 1 1]

A high “ambient factor” causes the image to appear bright white in views regardless of the color that has been set. To reveal the color, reset the reflection properties to less extreme values. See material:set–color–prop for more details.

entity is the entity or entity list for which the color is set.

rgb–color is the color to set.

Limitations: None

Example:

```
; entity:set–material–color
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 30 10 3) 20))
;; cyl1
; Determine if the cylinder has a material.
(entity:material cyl1)
;; #f
; Create a material.
(define matter1 (material))
;; matter1
; Get the default type values.
(material:color–type matter1)
;; “plain”
; Get the default color values.
(material:color–props matter1)
;; ((”color” . #[color 1 1 1]))
; Set the material.
(entity:set–material cyl1 matter1)
;; ()
; Set the color of the material.
(entity:set–material–color cyl1
    (color:rgb 1 1 0))
;; ()
```

entity:set–material–reflection

Scheme Extension: Materials, Reflectance

Action: Sets the reflection properties in the material associated with a geometric entity.

Filename: rbase/rnd_scm/mat_scm.cxx

APIs: api_rh_set_material_reflection

Syntax:
```
(entity:set–material–reflection entity
    ambient–ref diffuse–ref specular–ref exponent)
```

Arg Types:		
	entity	entity
	ambient–ref	real
	diffuse–ref	real
	specular–ref	real
	exponent	real

Returns: unspecified

Errors: None

Description: Sets the reflection properties in the material associated with a geometric entity.

This extension sets the material color (if any) to be associated with the entity. This changes the material's color type to "plain". It does not affect its displacement, reflection, or transparency types or properties. The argument rgb_color is the color to set.

When a new "plain" color type material is created, the default reflection type is "phong". It has the following properties:

"ambient factor" 1
"diffuse factor" 0.75
"specular factor" 0.5
"exponent" 10
"specular color" . . . . . #[color 1 1 1]

A high "ambient factor" causes the image to appear bright white in views regardless of the color that has been set. To reveal the color, reset the reflection properties to less extreme values. For information about reflection properties is given in material:set–reflection–prop.

entity specifies an entity to which the material is to be attached.

ambient–ref specifies the ambient reflection. The value ranges from 0 to 1.

diffuse–ref specifies the diffuse reflection. The value ranges from 0 to 1.

specular–ref specifies the specular reflection. The value ranges from 0 to 1.

exponent specifies the exponent.

Limitations: None

Example:

```
; entity:set–material–reflection
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 30 30 0) 20))
;; cyl1
; Determine if the cylinder has a material.
(entity:material cyl1)
;; #f
; Create a material.
(define matter1 (material))
;; matter1
; Get the default material properties.
(material:reflection–props matter1)
;; ((”ambient factor” . 1) (”diffuse factor” . 0.75)
;; (”specular factor” . 0.5) (”exponent” . 10)
;; (”specular color” . #[color 1 1 1]))
; Set the material.
(entity:set–material cyl1 matter1)
;; ()
; Set the reflection of the material.
(entity:set–material–reflection cyl1
    0.4 0.6 0.2 10)
;; ()
```

entity:set–material–texture

Scheme Extension: Entity, Materials, Textures

Action: Sets the texture file name in a material associated with a geometric entity.

Filename: rbase/rnd_scm/mat_scm.cxx

APIs: api_rh_set_material_texture

Syntax: (**entity:set–material–texture** entity filename)

Arg Types:	entity	entity
	filename	string

Returns: material

Errors: None

Description: This extension sets the texture file name in a material accosted with the entity. If no named file is found, it returns #f. The color type is set to “wrapped image” and sets its “file name” property to the filename string. See material:set–color–prop for more details.

entity specifies an entity to which the material is attached.

filename specifies the file name of the texture to be attached.

Limitations: None

Example:
```
; entity:set–material–texture
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 30 30 0) 20))
;; cyl1
; Determine if the cylinder has a material.
(entity:material cyl1)
;; #f
; Create a material.
(define matter1 (material))
;; matter1
; Set the material.
(entity:set–material cyl1 matter1)
;; ()
; Set the texture of the material.
(entity:set–material–texture cyl1 ”brick.bmp”)
;; ()
```

entity:set–material–transparency

Scheme Extension: Entity, Materials, Transparency

Action: Sets the transparency property in the material associated with a geometric entity.

Filename: rbase/rnd_scm/mat_scm.cxx

APIs: api_rh_set_material_transp

Syntax: (**entity:set–material–transparency** entity transparency–factor)

Arg Types:	entity	entity
	transparency–factor	real

Returns: unspecified

Errors: None

Description: This extension sets the transparency property in the material associated with a geometric entity. See material:set–transparency–prop for more details.

entity specifies an entity to which the material is to be attached.

transparency–factor specifies the transparency factor, which ranges from 0 (transparent) to 1 (opaque).

Limitations: None

Example:

```
; entity:set–material–transparency
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 30 5 20) 20))
;; cyl1
; Determine if the cylinder has a material.
(entity:material cyl1)
;; #f
; Create a material.
(define matter1 (material))
;; matter1
; Set the material.
(entity:set–material cyl1 matter1)
;; ()
; Set the transparency of the material.
(entity:set–material–transparency cyl1 0.8)
;; ()
```

entity:set–render–sides

Scheme Extension: Rendering Control

Action: Sets the sidedness of an entity or list of entities.

Filename: rbase/rnd_scm/avh_scm.cxx

APIs: api_rh_set_sidedness

Syntax: `(entity:set–render–sides [entity–list] sides)`

Arg Types:

entity–list	entity | (entity ...)
sides	integer

Returns: unspecified

Errors: None

Description: Refer to Action.

entity–list specifies any entity or a list of entities.

sides specifies the number of sides to render, which include 0 (undefined sidedness), 1 (single–sided), and 2 (double–sided).

Limitations: None

Example:

```
; entity:set–render–sides
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 8 8 0) 20))
;; cyl1
; Set the number of sides for rendering.
(entity:set–render–sides cyl1 2)
;; ()
; Get the number of sides for rendering.
(entity:render–sides cyl1)
;; 2
```

entity:set–texture–space

Scheme Extension: Texture Spaces

Action: Sets the texture space associated with an entity.

Filename: rbase/rnd_scm/tmap_scm.cxx

APIs: api_rh_set_texture_space

Syntax: `(entity:set–texture–space entity–list texture–space)`

Arg Types:

entity–list	entity | (entity ...)
texture–space	entity

Returns: unspecified

Errors: None

Description: The texture space is used during rendering to control how a wrapped material entity is applied to any face or solid body. Refer to texture–space:set–prop for more details.

entity–list specifies the entity to set the texture space.

texture–space specifies the texture space entity.

Limitations: None

Example:
```
; entity:set–texture–space
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position –25 –25 –25)))
;; block1
; Create a texture space.
(define texture1 (texture–space ”auto axis”))
;; texture1
; Assign the texture space to the block.
(entity:set–texture–space block1 texture1)
;; ()
(define texture (entity:texture–space block1))
;; texture
```

entity:texture–space

Scheme Extension: Texture Spaces

Action: Gets the texture space associated with a geometric entity.

Filename: rbase/rnd_scm/tmap_scm.cxx

APIs: api_rh_get_texture_space

Syntax: (**entity:texture–space** entity)

Arg Types: entity entity

Returns: texture–space

Errors: None

Description: Refer to Action.

entity specifies the entity to query.

Limitations: None

Example:
```
; entity:texture–space
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 15 25 5)))
;; block1
; Create a texture space.
(define tspace1 (texture–space ”auto axis”))
;; tspace1
; Assign the texture space to the block.
(entity:set–texture–space block1 tspace1)
;; ()
; Inquire the texture space assigned to the block.
(entity:texture–space block1)
;; #[entity 3 1]
```

environment–map:data

Scheme Extension: Environment Maps

Action: Creates an environment map from data.

Filename: rbase/rnd_scm/emap_scm.cxx

APIs: api_rh_create_cube_environment

Syntax: (**environment–map:data** colors widths heights filename)

Arg Types:		
	colors	integer
	widths	integer | (integer ...)
	heights	integer | (integer ...)
	filename	string | (integer ...)

Returns: environment–map

Errors: None

Description: This extension creates an environment map from data supplied in a file or the command line.

Environment maps simulate reflections that cannot be viewed in the Basic Rendering Component “flat” or “gouraud” render modes; use the Advanced Rendering Component to view reflections.

An environmental map is an entity associated with rendering reflections. It may be thought of as the six faces of a cube surrounding a model. Each of the inward facing faces contains an image that is wrapped onto reflective objects during rendering. Therefore, environmental maps only provide direction reflections. Reflections of reflections requires ray tracing rendering methods. Environmental maps are saved when the part is saved. It is represented in the form #[entity m n], where *m* refers to the entity and *n* to the part.

colors specifies the type of colors supplied; valid options are 1 (greyscale data) or 3 (RGB color data).

widths and heights arguments specify the sizes in pixels of the six images. If less than six values are given, the extension uses the last specified value for unspecified values.

If filename is a string, it provides the name of a file containing binary image data; otherwise, it is a list containing integer image data. For greyscale data, each pixel is represented by a single value from 0 (black) to 255 (white). Each pixel of RGB data is represented by three values (one each for red, green, and blue intensity) from 0 (none) to 255 (full). Values for each image are ordered from left–to–right in each row and top–to–bottom for each image. Images are provided in the following order: +X, –X, +Y, –Y, +Z, –Z.

Limitations: None

Example:

```
; environment–map:data
; Create an environment map from the supplied data.
(define emap1
    (environment–map:data 3 20 20 ”emap.data”))
;; emap1
; Assign the environment map to an entity.
(render:set–environment–map emap1)
;; ()
; Define a sphere.
(define sphere1 (solid:sphere (position 0 0 0) 30))
;; sphere1
; Define a material.
(define matter1 (material))
;; matter1
; Set the reflection type to be ”environment”.
(material:set–reflection–type emap1 ”environment”)
;; ()
; Set the entity and the material.
(entity:set–material sphere1 matter1)
;; ()
; Set the render mode to full.
(render:set–mode ”full”)
;; ()
; Render the image.
(render)
;; ()
```

environment–map:lwi

Scheme Extension: Environment Maps

Action: Creates an environment map from .lwi files.

Filename: rbase/rnd_scm/emap_scm.cxx

APIs: api_rh_create_cube_environment

Syntax:

```
(environment–map:lwi img+x img–x
    img+y img–y img+z img–z)
```

Arg Types:		
	img+x	string
	img–x	string
	img+y	string
	img–y	string
	img+z	string
	img–z	string

Returns: environment–map

Errors: None

Description: This extension creates a six sided cubical environment map used to simulate reflections.

Environment maps simulate reflections that cannot be viewed in the Basic Rendering Component “flat” or “gouraud” render modes. Use the Advanced Rendering Component to view reflections.

An environmental map is an entity associated with rendering reflections. It may be thought of as the six faces of a cube surrounding a model. Each of the inward facing faces contains an image that is wrapped onto reflective objects during rendering. Therefore, environmental maps only provide direction reflections. Reflections of reflections require ray tracing rendering methods. Environmental maps are saved when the part is saved. It is represented in the form #[entity m n], where *m* refers to the entity and *n* to the part.

img+x, img–x, img+y, img–y, img+z, and img–z are strings specifying the names of the files that holds an image in .lwi (Lightworks Image) format to be mapped to a side of the cube.

Limitations: None

Example:

```
; environment–map:lwi
; Create an environment–map from LightWorks
; Image format data files.
(define emap1 (environment–map:lwi
    ”xp.lwi” ”xm.lwi” ”yp.lwi” ”ym.lwi”
    ”zp.lwi” ”zm.lwi”))
;; emap1
; Render the set environment map.
(render:set–environment–map emap1)
;; ()
; Define a sphere.
(define sphere1 (solid:sphere (position 0 0 0) 30))
;; sphere1
; Define a material.
(define matter1 (material))
;; matter1
; Set the reflection type of the material.
(material:set–reflection–type matter1 ”environment”)
;; ()
; Set the material onto the sphere.
(entity:set–material sphere1 matter1)
;; ()
; Set the render mode to full.
(render:set–mode ”full”)
;; ()
; Render the image.
(render)
;; ()
```

environment–map:render

Scheme Extension: Environment Maps, Rendering Control

Action: Creates an environment map by rendering.

Filename: rbase/rnd_scm/emap_scm.cxx

APIs: api_rh_render_cube_environment

Syntax:

```
(environment–map:render entities=all
    resolution position)
```

Arg Types:		
	entities	entity | (entity ...)
	resolution	integer
	position	position

Returns: environment–map

Errors: None

Description: This extension creates an environment map by rendering six images of the specified entities as viewed from the specified position.

Rendering an environment map takes as long as rendering the image normally, and the image must be re–rendered using the displayed environment map. Therefore, environment–map:render is efficient only when the image is redisplayed many times from the same viewpoint.

Environment maps simulate reflections that cannot be viewed in the Basic Rendering Component flat or gouraud render modes. Use the Advanced Rendering Component to view reflections.

An environmental map is an entity associated with rendering reflections. It may be thought of as the six faces of a cube surrounding a model. Each of the inward facing faces contains an image that is wrapped onto reflective objects during rendering. Therefore, environmental maps only provide direction reflections. Reflections of reflections require ray tracing rendering methods. Environmental maps are saved when the part is saved. It is represented in the form #[entity m n], where *m* refers to the entity and *n* to the part.

entities specifies the entity–list to render; the default is all entities.

resolution specifies the width and height in pixels for the generated images.

position specifies the position to render the images.

Limitations: None

Example:

```
; environment–map:render
; Create an environment map.
(define emap1
    (environment–map:render (part:entities)
    256 (position 0 0 0)))
;; emap1
; Render the environment map.
(render:set–environment–map emap1)
;; ()
; Define a sphere.
(define sphere1 (solid:sphere (position 0 0 0) 30))
;; sphere1
; Define a material.
(define matter1 (material))
;; matter1
; Set the reflection of the material.
(material:set–reflection–type matter1 ”environment”)
;; ()
; Set the material onto the sphere.
(entity:set–material sphere1 matter1)
;; ()
; Set the render mode to full.
(render:set–mode ”full”)
;; ()
; Render the image.
(render)
;; ()
```

environment–map:stripe

Scheme Extension: Environment Maps

Action: Creates a procedurally–defined environment map.

Filename: rbase/rnd_scm/emap_scm.cxx

APIs: api_rh_create_cube_environment

Syntax: (**environment–map:stripe** stripe–width widths heights)

Arg Types:

stripe–width	integer
widths	integer | (integer ...)
heights	integer | (integer ...)

Returns: environment–map

Errors: None

Description: This extension creates a simple stripe pattern to illustrate the use of procedurally–defined environment maps. Each image consists of alternating stripes of black and red, green, or blue. One to six widths and heights can be specified.

Environment maps simulate reflections that cannot be viewed in the Basic Rendering Component "flat" or "gouraud" render modes. Use the Advanced Rendering Component to view reflections.

An environmental map is an entity associated with rendering reflections. It may be thought of as the six faces of a cube surrounding a model. Each of the inward facing faces contains an image that is wrapped onto reflective objects during rendering. Therefore, environmental maps only provide direction reflections. Reflections of reflections require ray tracing rendering methods. Environmental maps are saved when the part is saved. It is represented in the form #[entity m n], where *m* refers to the entity and *n* to the part.

stripe–width specifies the width in pixels of each stripe.

widths and heights argument specify the sizes in pixels of the six images (+X, –X, +Y, –Y, +Z, –Z). The last specified value for each dimension is used for any unspecified values.

Limitations: None

Example:

```
; environment–map:stripe
; Create a procedurally defined environment
; map, representing 20x20 pixel images with
; 2 pixel wide stripes
(define emap1 (environment–map:stripe 10 20 20))
;; emap1
; Render the set environment map.
(render:set–environment–map  emap1)
;; ()
; Define a sphere.
(define sphere1 (solid:sphere (position 0 0 0) 30))
;; sphere1
; Define a material.
(define matter1 (material))
;; matter1
; Set the reflection type of the material.
(material:set–reflection–type  matter1
    ”environment”)
;; ()
; Set the material onto the sphere.
(entity:set–material sphere1 matter1)
;; ()
; Set the render mode to full.
(render:set–mode ”full”)
;; ()
; Render the image.
(render)
;; ()
; OUTPUT Example
```

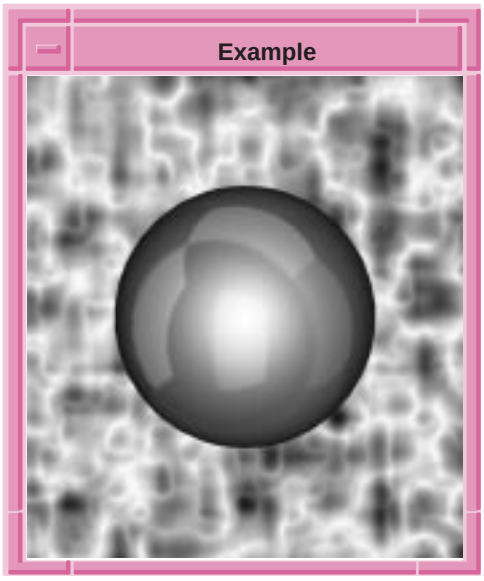

Figure 2-3. environment–map:stripe

environment–map?

Scheme Extension: Environment Maps

Action: Determines if a Scheme object is an environment map.

Filename: rbase/rnd_scm/emap_scm.cxx

APIs: None

Syntax: `(environment-map? object)`

Arg Types:	object	scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if the specified object is an environment map; otherwise, it returns #f.

An environmental map is an entity associated with rendering reflections. It may be thought of as the six faces of a cube surrounding a model. Each of the inward facing faces contains an image that is wrapped onto reflective objects during rendering. Therefore, environmental maps only provide direction reflections. Reflections of reflections require ray tracing rendering methods. Environmental maps are saved when the part is saved. It is represented in the form #[entity m n], where *m* refers to the entity and *n* to the part.

object specifies the scheme–object that has to be queried for an environment map.

Limitations: None

Example:

```
; environment–map?
; Create a procedurally defined environment map.
(define emap1 (environment–map:stripe 5 20 30))
;; emap1
; Determine if it is actually an environment map.
(environment–map? emap1)
;; #t
```

foreground

Scheme Extension: Backgrounds and Foregrounds

Action: Creates a new foreground.

Filename: rbase/rnd_scm/frgd_scm.cxx

APIs: api_rh_create_foreground

Syntax: (**foreground** type)

Arg Types:	type	string

Returns: foreground

Errors: None

Description: A foreground is an entity. render:set–foreground assigns a foreground shader for rendering. A foreground shader filters and transforms the color of a model's pixels.

type specifies a string defining the foreground.

Limitations: Only plain uniform color foregrounds are displayed. Only one foreground can be active at a time. Only the Advanced Rendering Component supports foregrounds. The only available foreground for the other renderers is "None".

Example:

```
; foreground
; Create a fog foreground.
(define fore1 (foreground ”fog”))
;; fore1
; Set the current foreground to be rendered.
(render:set–foreground fore1)
;; ()
```

foreground:props

Scheme Extension: Backgrounds and Foregrounds

Action: Gets the properties of a foreground.

Filename: rbase/rnd_scm/frgd_scm.cxx

APIs: api_rh_get_foreground_args

Syntax: (**foreground:props** foreground)

Arg Types: foreground — foreground

Returns: (pair. . .)

Errors: None

Description: This extension returns a list containing pairs of argument types and their present values.

foreground specifies a foreground entity.

Limitations: None

Example:

```
; foreground:props
; Create a ”depth cue” foreground.
(define fore1 (foreground ”depth cue”))
;; fore1
; Get the properties of a ”depth cue” foreground.
(foreground:props fore1)
;; ((”near” . 0) (”far” . 1000)
;; (”background color” . #[color 0 0 0]))
; Create a ”none” foreground.
(define another–fore (foreground ”none”))
;; another–fore
; Determine the properties of a ”none” foreground.
(foreground:props another–fore)
;; ()
```

foreground:set–prop

Scheme Extension: Backgrounds and Foregrounds

Action: Sets the properties of a foreground.

Filename: rbase/rnd_scm/frgd_scm.cxx

APIs: api_rh_get_foreground_args, api_rh_set_foreground_arg

Syntax: `(foreground:set–prop foreground name value)`

Arg Types:

foreground	foreground
name	string
value	real | integer | color | string

Returns: unspecified

Errors: None

Description: Refer to Action.

foreground specifies a foreground entity.

name specifies the name of the property. The properties include “None”, “depth cue”, and “fog”.

“None” is for no foreground (black). Each pixel is shown in its true color.

“depth cue” is for a linear attenuation within a specified depth range. The effect depends upon the distance from the view’s eye plane to the corresponding points of the model. Pixels that are closer to the eye plane than the near property but are beyond the hither plane are shown in their true colors. Pixels that are further away from the eye plane than the far property but are closer than the yon plane are shown in the background color. (background color is not the same as background.) Pixel locations in between these two are shown in a blend color.

near . . . . . 0.0 (real)
far . . . . . 1000.0 (real)
background color . . . . . #[color 0 0 0]

“fog” is for an exponential attenuation for atmospheric simulation. The pixel color is a weighted average of the fog color property and the pixel’s true color. The average is given by:

```
(f*fog_color)+[(1–f)*true_color], where
f = exp(– point_to_plane / distance).
```

point_to_plane is the distance from the model point to the respective eye or hither plane. distance–prop in the equation is the distance property. The fog_color is generally white (#[color 1 1 1]) or black (#[color 0 0 0]).

distance ..... 1000.0 (real)
fog color ..... #[color 1 1 1]

value specifies the value of the property.

Limitations: Only the Advanced Rendering Component supports foregrounds. The only available foreground for the other renderers is “None”.

Example:

```
; foreground:set-prop
; Create a ”depth cue” foreground.
(define depth1 (foreground ”depth cue”))
;; depth1
; Get the properties of a ”depth cue” foreground.
(foreground:props depth1)
;; ((”near” . 0) (”far” . 1000)
;; (”background color” . #[color 0 0 0]))
; Set a property for the ”depth cue” foreground.
(foreground:set-prop depth1 ”background color”
    (color:rgb 0 1 0))
;; ()
; Create a ”fog” foreground.
(define fog1 (foreground ”fog”))
;; fog1
; Get the properties of the ”fog” foreground.
(foreground:props fog1)
;; ((”distance” . 1000)
;; (”fog color” . #[color 1 1 1]))
; Set a property for the ”fog” foreground.
(foreground:set-prop fog1 ”fog color”
    (color:rgb 0 1 0))
;; ()
```

foreground:type

Scheme Extension: Backgrounds and Foregrounds

Action: Gets the type of foreground.

Filename: rbase/rnd_scm/frgd_scm.cxx

APIs: api_rh_get_foreground_args

Syntax: (**foreground:type** foreground)

Arg Types:	foreground	foreground

Returns: string

Errors: None

Description: This extension returns the type of the foreground in a string.

foreground specifies a foreground entity.

Limitations: Only the Advanced Rendering Component supports foregrounds. The only available foreground for the other renderers is “None”.

Example:

```
; foreground:type
; Create foreground 1.
(define depth1 (foreground ”depth cue”))
;; depth1
; Create foreground 2.
(define fog1 (foreground ”fog”))
;; fog1
; Get the type of foreground 1.
(foreground:type depth1)
;; ”depth cue”
; Get the type of foreground 2.
(foreground:type fog1)
;; ”fog”
```

foreground:types

Scheme Extension: Backgrounds and Foregrounds

Action: Gets the list of available foreground types.

Filename: rbase/rnd_scm/frgd_scm.cxx

APIs: api_rh_get_foreground_types

Syntax: (**foreground:types**)

Arg Types: None

Returns: (string ...)

Errors: None

Description: This extension returns a list of all valid foreground types.

Note *Only the* Advanced Rendering Component *supports foregrounds. The only available foreground for the other renderers is* “None”.

Limitations: None

Example:

```
; foreground:types
; Get a list of available foreground types.
(foreground:types)
;; (”fog” ”depth cue” ”none”)
```

foreground?

Scheme Extension: Backgrounds and Foregrounds

Action: Determines if a Scheme object is a foreground.

Filename: rbase/rnd_scm/frgd_scm.cxx

APIs: None

Syntax: (**foreground?** object)

Arg Types: object scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if a Scheme object is a foreground; otherwise, it returns #f.

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

Limitations: None

Example:

```
; foreground?
; Create a foreground.
(define fore1 (foreground ”none”))
;; fore1
; Determine if the entity is actually a foreground.
(foreground? fore1)
;; #t
```

light

Scheme Extension: Lights and Shadows

Action: Creates a light entity.

Filename: rbase/rnd_scm/lite_scm.cxx

APIs: api_rh_create_light

Syntax: `(light light–type)`

Arg Types:

light–type	string

Returns: light

Errors: None

Description: A light represents a source of illumination in a rendered scene. Light properties can be set using light:set–prop and queried with light:props.

Basic rendering permits the creation and deletion of shadow maps, but does not display shadows. The Advanced Rendering Component supports the display of shadows from certain light types.

light–type is a string specifying the light type. Valid values for light–type are “ambient”, “distant”, “spot”, “eye”, and “point”. The extension light:types also displays the available light types.

Limitations: None

Example:

```
; light
; What type of lights are available.
(light:types)
;; (”ambient” ”distant” ”eye” ”point” ”spot”)
; Create a spot light.
(define light1 (light ”spot”))
;; light1
```

light:create–shadows

Scheme Extension: Lights and Shadows

Action: Creates a shadow map for a light and a list of entities.

Filename: rbase/rnd_scm/lite_scm.cxx

APIs: api_rh_create_light_shadow

Syntax: `(light:create–shadows light entity–list)`

Arg Types:

light	light
entity–list	entity | (entity ...)

Returns: unspecified

Errors: None

Description: Creates a shadow map based on the input entities and associates the map to the input light entity.

Basic rendering permits the creation and deletion of shadow maps, but does not display shadows. The Advanced Rendering Component supports the display of shadows from certain light types.

light specifies a light entity. The only light entities that support shadows are “distant”, “spot”, and “point”.

entity–list specifies an entity list for creating a shadow map.

Limitations: None

Example:

```
; light:create–shadows
; Create solid block.
(define b1 (solid:block (position –40 –30 –5)
    (position 40 30 0)))
;; b1
; Create six solid spheres.
(define s1 (solid:sphere (position 0 0 20) 10))
;; s1
(define s2 (solid:sphere (position 0 0 40) 5))
;; s2
(define s3 (solid:sphere (position 0 15 15) 2.5))
;; s3
; Note that sphere s3 is behind s1.
(define s4 (solid:sphere (position 0 –15 15) 2.5))
;; s4
(define s5 (solid:sphere (position 20 0 20) 5))
;; s5
(define s6 (solid:sphere (position –20 0 20) 5))
;; s6
; Create a spot light.
(define spot–x (light ”spot”))
;; spot–x
; Set the list of currently–active lights
(light:set #t spot–x)
;; ()
; Set light properties.
(light:set–prop spot–x ”intensity” 3)
;; ()
(light:set–prop spot–x ”color”
    (color:rgb 0.94 0.50 0.50))
```

```
;; ()
(light:set-prop spot-x "location"
    (gvector 30 20 100))
;; ()
(light:set-prop spot-x "to" (gvector 0 0 0))
;; ()
(light:set-prop spot-x "fall off" 0)
;; ()
(light:set-prop spot-x "cone angle" 45)
;; ()
(light:set-prop spot-x "cone delta angle" 5)
;; ()
(light:set-prop spot-x "beam distribution" 4)
;; ()
; Set the render mode to full.
(render:set-mode "full")
;; ()
; Render the image.
(render)
;; ()
; OUTPUT Original

; Create a shadow map.
(light:create-shadows spot-x
    (list s1 s2 s3 s4 s5 s6))
;; ()
; Set shadow properties.
(light:set-prop spot-x "shadows" #t)
;; ()
(light:set-prop spot-x "shadow resolution" 512)
;; ()
(light:set-prop spot-x "shadow quality" 8)
;; ()
(light:set-prop spot-x "shadow softness" 10)
;; ()
; Render the image to see the shadows.
(render)
;; ()
; OUTPUT Result
```

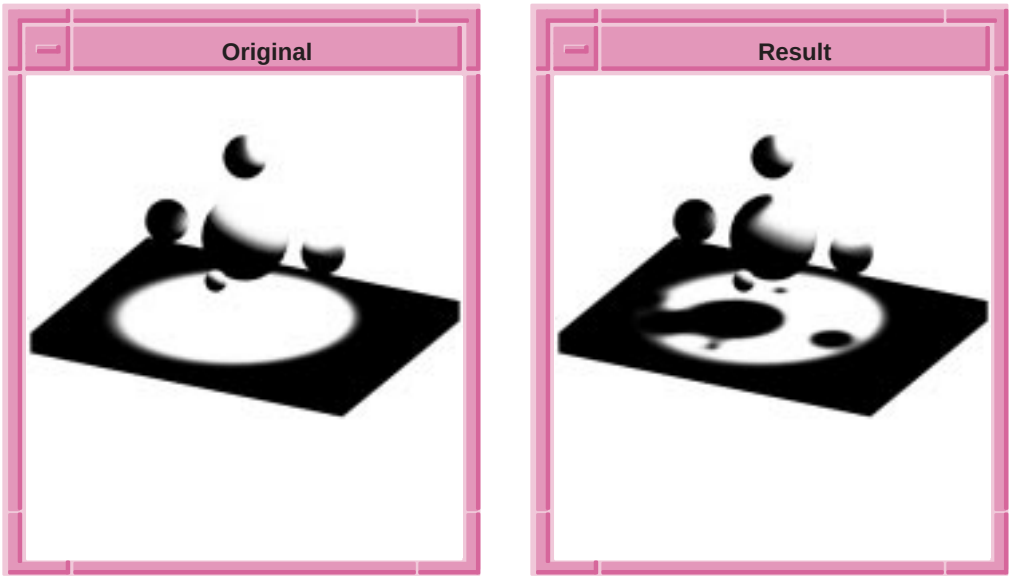

Figure 2-4. light:create–shadows

light:delete–shadows

Scheme Extension: Lights and Shadows

Action: Deletes the shadow map for a light.

Filename: rbase/rnd_scm/lite_scm.cxx

APIs: api_rh_delete_light_shadow

Syntax: `(light:delete–shadows light)`

Arg Types: light light

Returns: unspecified

Errors: None

Description: Deletes the shadow map for a light.

Basic rendering permits the creation and deletion of shadow maps, but does not display shadows. The Advanced Rendering Component supports the display of shadows from certain light types.

light specifies a light entity. The only light entities that support shadows are “distant”, “spot”, and “point”.

Limitations: None

Example:

```
; light:delete–shadows
; Create solid block.
(define b1 (solid:block (position –40 –30 –5)
    (position 40 30 0)))
;; b1
; Create six solid spheres.
(define s1 (solid:sphere (position 0 0 20) 10))
;; s1
(define s2 (solid:sphere (position 0 0 40) 5))
;; s2
(define s3 (solid:sphere (position 0 15 15) 2.5))
;; s3
; Note that sphere s3 is behind s1.
(define s4 (solid:sphere (position 0 –15 15) 2.5))
;; s4
(define s5 (solid:sphere (position 20 0 20) 5))
;; s5
(define s6 (solid:sphere (position –20 0 20) 5))
;; s6
; Create a spot light.
(define spot–x (light ”spot”))
;; spot–x
; Set the list of currently–active lights
(light:set #t spot–x)
;; ()
; Set light properties.
(light:set–prop spot–x ”intensity” 5)
;; ()
(light:set–prop spot–x ”color”
    (color:rgb 0.7 0.93 0.53))
;; ()
(light:set–prop spot–x ”location”
    (gvector 30 20 100))
;; ()
(light:set–prop spot–x ”to” (gvector 0 0 0))
;; ()
(light:set–prop spot–x ”fall off” 0)
;; ()
(light:set–prop spot–x ”cone angle” 30)
;; ()
(light:set–prop spot–x ”cone delta angle” 5)
;; ()
(light:set–prop spot–x ”beam distribution” 4)
```

```
;; ()
; Create a shadow map.
(light:create-shadows spot-x
    (list s1 s2 s3 s4 s5 s6))
;; ()
; Set shadow properties.
(light:set-prop spot-x "shadows" #t)
;; ()
(light:set-prop spot-x "shadow resolution" 512)
;; ()
(light:set-prop spot-x "shadow quality" 8)
;; ()
(light:set-prop spot-x "shadow softness" 10)
;; ()
; Set the render mode to full.
(render:set-mode "full")
;; ()
; Render the image to see the shadows.
(render)
;; ()
; OUTPUT Original

; Delete the shadows.
(light:delete-shadows spot-x)
;; ()
; Render the image without the shadows.
(render)
;; ()
; OUTPUT Result
```

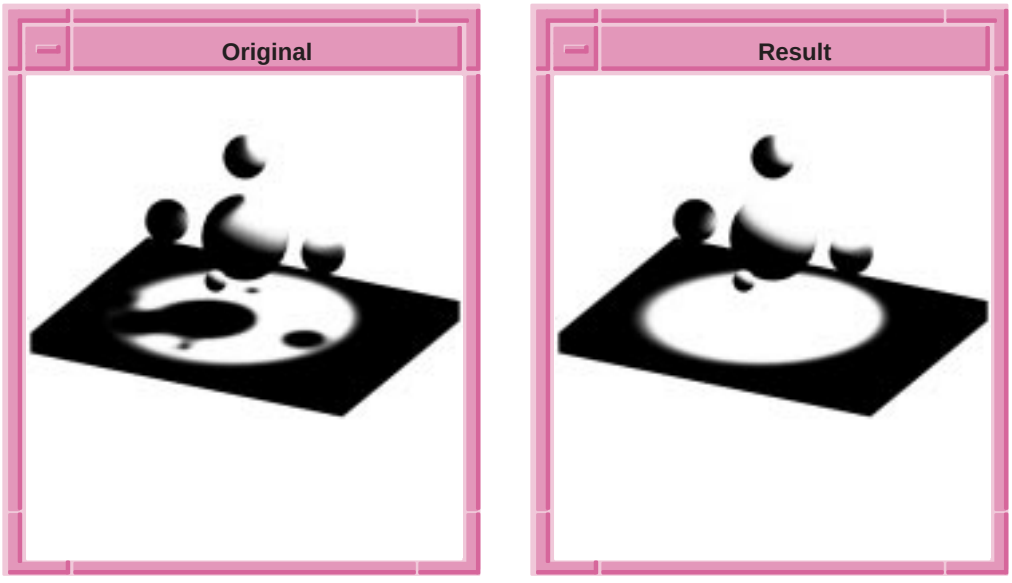

Figure 2-5. light:delete–shadows

light:list

Scheme Extension: Lights and Shadows

Action: Gets the list of active lights.

Filename: rbase/rnd_scm/lite_scm.cxx

APIs: api_rh_get_light_list

Syntax: **(light:list)**

Arg Types: None

Returns: (light ...)

Errors: None

Description: This extension returns the list of all active lights.

Limitations: None

Example:

```
; light:list
; Set active light 1.
(define distant1 (light ”distant”))
;; distant1
; Set active light 2.
(define spot1 (light ”spot”))
;; spot1
; Set active light 3.
(define ambient1 (light ”ambient”))
;; ambient1
; Put the three active lights into the light list.
(light:set #t (list
    distant1 spot1 ambient1))
;; ()
; Get the list of all active lights.
(light:list)
;; (#[entity 2 1] #[entity 3 1] #[entity 4 1])
```

light:on?

Scheme Extension: Lights and Shadows

Action: Determines if a light entity is active.

Filename: rbase/rnd_scm/lite_scm.cxx

APIs: None

Syntax: (**light:on?** entity)

Arg Types: entity — entity

Returns: boolean

Errors: None

Description: Refer to Action.

entity specifies a light entity to be queried.

Limitations: None

Example:

```
; light:on?
; Set active light 1.
(define distant1 (light ”distant”))
;; distant1
; Determine if a light is active.
(light:on? distant1)
;; #f
(light:set #t distant1)
;; ()
; Determine if a light is active.
(light:on? distant1)
;; #t
```

light:props

Scheme Extension: Lights and Shadows

Action: Gets the properties of a light.

Filename: rbase/rnd_scm/lite_scm.cxx

APIs: api_rh_get_light_args

Syntax: `(light:props light–type)`

Arg Types:

light–type	light

Returns: (string (string . string | integer | real | color | gvector) ...)

Errors: None

Description: This extension returns a list of the light–type, followed by pairs containing the property name and the present value.

light–type specifies a light entity.

Limitations: None

Example:

```
; light:props
; Create a distant light.
(define distant1 (light ”distant”))
;; distant1
; Get the properties of the distant light.
(light:props distant1)
;; ((”intensity” . 1) (”color” . #[color 1 1 1])
;; (”location” . #[gvector 0 0 1])
;; (”to” . #[gvector 0 0 0]) (”shadows” . #f)
;; (”shadow resolution” . 256) (”shadow quality” . 4)
;; (”shadow softness” . 1))
; Create a spot light.
(define spot1 (light ”spot”))
;; spot1
; Get the properties of the spot light.
(light:props spot1)
;; ((”intensity” . 1) (”color” . #[color 1 1 1])
;; (”location” . #[gvector 0 0 1])
;; (”to” . #[gvector 0 0 0]) (”fall off” . 0)
;; (”cone angle” . 60) (”cone delta angle” . 5)
;; (”beam distribution” . 2) (”shadows” . #f)
;; (”shadow resolution” . 256) (”shadow quality” . 4)
;; (”shadow softness” . 1))
```

light:set

Scheme Extension: Lights and Shadows

Action: Sets the list of currently–active lights.

Filename: rbase/rnd_scm/lite_scm.cxx

APIs: api_rh_get_light_list, api_rh_set_light_list, api_rh_set_light_state

Syntax: `(`**`light:set`** `on–off light–list)`

Arg Types:		
	on–off	boolean
	light–list	light | (light ...)

Returns: unspecified

Errors: None

Description: Before executing this extension, create lights and set their parameters.

Set on–off to true (#t) to activate the lights specified in the list or false (#f) to deactivate the lights. Setting on–off to true (#t) adds the lights to the global light list and setting it to false removes the lights from the global list.

light–list specifies a list of lights to be activated.

Limitations: None

Example:

```
; light:set
; Set active light 1.
(define distant1 (light ”distant”))
;; distant1
; Set active light 2.
(define spot1 (light ”spot”))
;; spot1
; Set active light 3.
(define ambient1 (light ”ambient”))
;; ambient1
; Put the three active lights into the light list.
(light:set #t (list
    distant1 spot1 ambient1))
;; ()
; Get the list of all active lights.
(light:list)
;; (#[entity 2 1] #[entity 3 1] #[entity 4 1])
```

light:set–prop

Scheme Extension: Lights and Shadows

Action: Sets a property of a light.

Filename: rbase/rnd_scm/lite_scm.cxx

APIs: api_rh_get_light_args, api_rh_set_light_arg

Syntax: (**light:set–prop** light name value)

Arg Types:

light	light
name	string
value	real | integer | string | color | gvector

Returns: unspecified

Errors: None

Description: The following lists the property names and associated default values.

Note *Only property names and values associated with a specific light entity can be set.*

"ambient" illuminates all surfaces equally regardless of orientation. The intensity of the source (as a scalar quantity) is determined by intensity and color. The default values are:

intensity . . . . . 1.0 (real)
color . . . . . #[color 1 1 1]

"distant" projects parallel light from a distant source. An orthographic projection is set up with a camera positioned at the location point of the light and directed towards the to point of the light. Clipping planes are set to ensure that entities are not excluded. The default values are:

intensity . . . . . 1.0 (real)
color . . . . . #[color 1 1 1]
location . . . . . #[gvector 0 0 1]
to . . . . . #[gvector 0 0 0]
shadows . . . . . #f (boolean)
shadow resolution . . . . . 256 (int)
shadow quality . . . . . 4 (int)
shadow softness . . . . . 1.0 (real)

"spot" directs light from a spot source constrained to a cone. A perspective projection is set up with a camera positioned at the light's location point, and directed towards the light's to point. The field of view is specified by cone angle. Clipping planes are set to exclude entities behind the light. The default values are:

intensity . . . . . 1.0 (real)
color . . . . . #[color 1 1 1]
location . . . . . #[gvector 0 0 1]
to . . . . . #[gvector 0 0 0]
fall off . . . . . 0 (integer)
cone angle . . . . . 0.0 (real)
cone delta angle . . . . . 5.0 (real)
beam distribution . . . . . 2.0 (real)
shadows . . . . . #f (boolean)
shadow resolution . . . . . 256 (int)
shadow quality . . . . . 4 (int)
shadow softness . . . . . 1.0 (real)

"eye" emits light from the view position. No shadows are ever cast from an eye light. The default values are:

intensity . . . . . 1.0 (real)
color . . . . . #[color 1 1 1]

"point" emits light from a point equally in all directions. The position of the light is specified by location. The default values are:

intensity	1.0 (real)
color	#[color 1 1 1]
location	#[gvector 0 0 1]
fall off	0 (integer)
shadows	#f (boolean)
shadow resolution	256 (int)
shadow quality	4 (int)
shadow softness	1.0 (real)

light specifies the light entity.

name specifies the property name that attaches to the light entity.

value specifies the value associated with the property name.

Limitations: None

Example:

```
; light:set–prop
; Create a spot light.
(define spot1 (light ”spot”))
;; spot1
; Get the properties of the spot light.
(light:props spot1)
;; ((”intensity” . 1) (”color” . #[color 1 1 1])
;; (”location” . #[gvector 0 0 1])
;; (”to” . #[gvector 0 0 0]) (”fall off” . 0)
;; (”cone angle” . 60) (”cone delta angle” . 5)
;; (”beam distribution” . 2) (”shadows” . #f)
;; (”shadow resolution” . 256) (”shadow quality” . 4)
;; (”shadow softness” . 1))
; Set properties for the spot light.
(light:set–prop spot1 ”color”
    (color:rgb 0.5 0.5 0.5))
;; ()
; Get the properties of the spot light.
(light:props spot1)
;; ((”intensity” . 1)
;; (”color” . #[color 0.5 0.5 0.5])
;; (”location” . #[gvector 0 0 1])
;; (”to” . #[gvector 0 0 0]) (”fall off” . 0)
;; (”cone angle” . 60) (”cone delta angle” . 5)
;; (”beam distribution” . 2) (”shadows” . #f)
;; (”shadow resolution” . 256) (”shadow quality” . 4)
;; (”shadow softness” . 1))
; Create a point light.
(define point1 (light ”point”))
;; point1
; Set properties for the point light.
(light:set–prop point1 ”shadows” 1)
;; ()
```

light:type

Scheme Extension: Lights and Shadows

Action: Gets the type of a light.

Filename: rbase/rnd_scm/lite_scm.cxx

APIs: api_rh_get_light_args

Syntax: (**light:type** light)

Arg Types:	light	light
Returns:	string	
Errors:	None	
Description:	Refer to Action.	
	light specifies a light entity.	
Limitations:	None	

Example:

```
; light:type
; Create a spot light.
(define spot1 (light ″spot″))
;; spot1
; Determine the type of light.
(light:type spot1)
;; ″spot″
```

light:types

Scheme Extension: Lights and Shadows

Action:	Gets a list of available light types.
Filename:	rbase/rnd_scm/lite_scm.cxx
APIs:	api_rh_get_light_types
Syntax:	(**light:types**)
Arg Types:	None
Returns:	(string ...)
Errors:	None
Description:	This extension returns all valid light types as a list of strings.
Limitations:	None

Example:

```
; light:types
; Get a list of the available light types.
(light:types)
;; (″ambient″ ″distant″ ″eye″ ″point″ ″spot″)
```

light?

Scheme Extension: Lights and Shadows

Action:	Determines if a Scheme object is a light.

Filename: rbase/rnd_scm/lite_scm.cxx

APIs: None

Syntax: `(light? 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 light.

Limitations: None

Example:

```
; light?
; Create a spot light.
(define light1 (light ”spot”))
;; light1
; Determine if the object is a light.
(light? light1)
;; #t
```