Chapter 3.
Scheme Extensions Va thru Zz

Topic: Ignore

view:axes

Scheme Extension: Viewing, Backgrounds and Foregrounds,

Action: Enables and disables display of coordinate axes within an OpenGL view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: `(view:axes [view=active] [length=10])`

Arg Types:		
	view	view
	length	real

Returns: view

Errors: None

Description: Define length argument with a positive number to enable the viewing of coordinate axes. Use a nonpositive length to disable display of axes.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:axes
; work in OpenGL
(define view1 (view:gl))
;;view1
;create an axes (default length is 10)
(view:axes)
;; #[view 1180742]
; create an axes (axes length is 20)
(view:axes 20)
;; #[1180742]
; turn off coordinate axes display.
(view:axes 0)
;; #[view 1180742]
; specify a specific viewpoint
(view:axes 20 view1)
;; #[view 1180742]
; obtain the current axes length (or 0 when OFF)
(view:axes?)
;; 20
```

view:axes?

Scheme Extension: Viewing, Backgrounds and Foregrounds,

Action: Returns the length of coordinate axes (0 = do not draw).

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:axes?** [view=active])

Arg Types:	view	view

Returns: integer

Errors: None

Description: Refer to Action.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:axes?
; work in OpenGL
(define view1 (view:gl))
;;view1
;create an axes (default length is 10)
(view:axes)
;; #[view 1180742]
; create an axes with length of 20)
(view:axes 20)
;; #[1180742]
; turn off coordinate axes display.
(view:axes 0)
;; #[view 1180742]
; specify a specific viewpoint
(view:axes 20 view1)
;; #[view 1180742]
; obtain the current axes length (or 0 when OFF)
(view:axes?)
;; 20
```

view:coedge–direction

Scheme Extension: Viewing

Action: Displays directional arrows indicating the direction of a coedge or group of coedges.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax:

```
(view:coedge–direction coedge | coedge–list | loop |
    face coedges–on–off=#t [view=active])
```

Arg Types:		
	coedge	entity
	coedge–list	entity | (entity...)
	loop	entity
	face	entity
	coedges–on–off	boolean
	view	view

Returns: view

Errors: None

Description: The first argument can be a coedge, a coedge list, a loop, or a face. coedges–on–off set to #t causes directional arrows showing the coedge's direction to be displayed. coedges–on–off set to #f hides the coedge's directional arrow. view specifies in which view the coedge arrow is to be displayed.

Passing in a loop or face displays arrows on all coedges corresponding to the specified loop (or face).

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in DL views only.

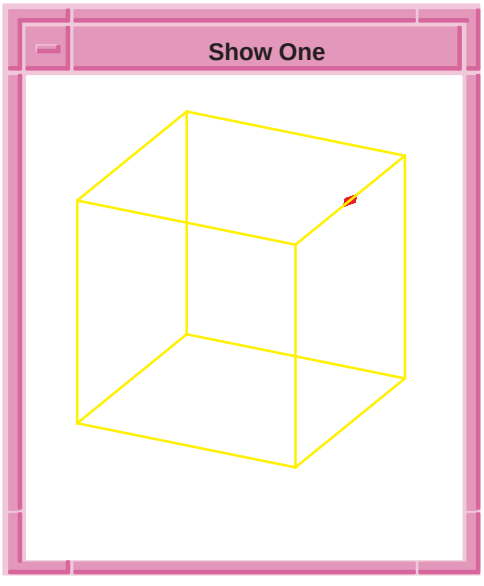
Example:

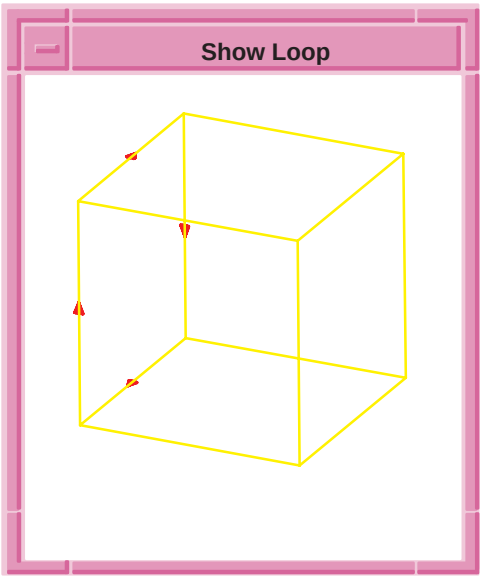
```
; view:coedge–direction
; Define geometry to demonstrate command.
(define block (solid:block –10 –10 –10 10 10 10))
;; block
; Display one of the coedges' direction.
(define block_coedges (entity:coedges block))
;; block_coedges
(define direction (view:coedge–direction
    (list–ref block_coedges 0) #t))
;; direction
; OUTPUT Show One

; View all coedges.
(define show–all (view:coedge–direction
    block_coedges #t))
;; show–all
; OUTPUT Show All
```

```
; Turn off coedge arrow on first edge.
(define off-one (view:coedge-direction
    (list-ref block_coedges 0) #f))
;; off-one
; Turn off coedges on all edges.
(define off-all (view:coedge-direction
    block_coedges #f))
;; off-all
; Show coedges on one face.
(define face (entity:faces block) )
;; face
(define show-face (view:coedge-direction
    (list-ref face 1) #t))
;; show-face
; Show coedges on one loop.
(define loop (entity:loops block) )
;; loop
(define show-loop (view:coedge-direction
    (list-ref loop 3) #t))
;; show-loop
; OUTPUT Loop

; Turn off all coedge arrows
(define view-off (view:coedge-direction-clear))
;; view-off
```





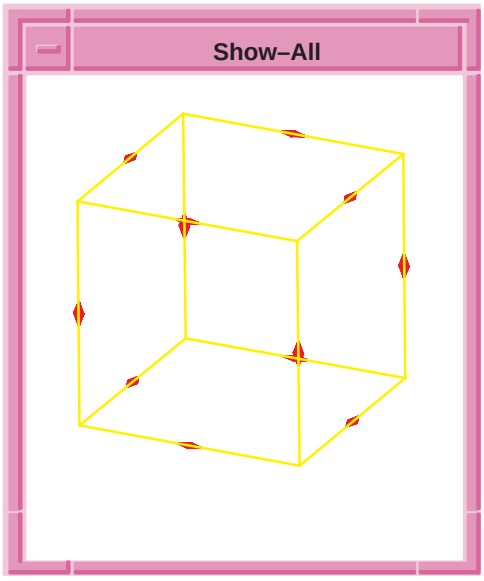


Figure 3-1. view:coedge–direction

view:coedge–direction–clear

Scheme Extension: Viewing

Action: Clears all directional arrows indicating the direction of a coedge or group of coedges.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: `(view:coedge-direction-clear [view=active])`

Arg Types: view view

Returns: view

Errors: None

Description: Clears all coedge directional arrows. view specifies what view will be cleared.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in DL views only.

Example:

```
; view:coedge–direction–clear
; Create geometry to demonstrate command.
(define block (solid:block
    (position –10 –10 –10) (position 10 10 10)))
;; block
; Show coedges on one loop.
(define loop (entity:loops block))
;; loop
(define show–loop (view:coedge–direction
    (list–ref loop 3) #t))
;; show–loop
; Show coedges on one face.
(define face (entity:faces block))
;; face
(define show–face (view:coedge–direction
    (list–ref face 1) #t))
;; show–face
; Turn off all coedge arrows.
(define show–none (view:coedge–direction–clear))
;; show–none
```

view:coedges

Scheme Extension: Viewing

Action: Enables and disables display of tolerant coedges within an OpenGL view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:coedges** [view=active] [edges])

Arg Types:	view	view
	edges	boolean

Returns: view

Errors: None

Description: This extension enables (#t) and disables (#f) a model's olerant tolerant coedges within a GL view. When enabled and when shading (view:shaded) is disabled, the result is similar to a wireframe representation only without the silhouette lines.

Limitations: Works only in HOOPS and Visman.

Example:

```
; view:coedges
; Define a view.
(define view1 (view:gl))
;; #[view 1075519376]
; Define a block.
(define block1
    (solid:block (position 0 0 0)
                 (position 25 20 30)))
;; block1
(define cylinder1
    (solid:cylinder (position –10 –10 –10)
                    (position –10 –10 30) 5))
;; cylinder1
; Verify that edges are turned on
(view:coedges #t view1)
;; #[view 1075519376]
; If coedges are not turned on, you won’t see
; anything when shading is turned off.
; Turn shading off.
(view:shaded #f view1)
;; #[view 1075519376]
; The coedges of the model should be visible.
; Note that silhouettes are not visible on the
; cylinder. Verify that shading is turned on.
(view:shaded #t view1)
;; #[view 1075519376]
; Turn on polygon offset to make coedges
; easier to see
(view:polygonoffset #t view1)
;; #[view 1075519376]
(view:coedges? view1)
;; #t
```

view:coedges?

Scheme Extension: Viewing

Action: Determines if edges are displayed in a view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:coedges?** [view=active])

Arg Types: view view

Returns: view

Errors: None

Description: Refer to action.

Limitations: Works only in HOOPS and Visman.

Example:

```
; view:coedges?
; Define a view.
(define view1 (view:gl))
;; #[view 1075519376]
; Define a block.
(define block1
    (solid:block (position 0 0 0)
                    (position 25 20 30)))
;; block1
(define cyl1 (solid:cylinder
                (position –10 –10 –10)
                (position –10 –10 30) 5))
;; cyl1
; Verify that edges are turned on
(view:edges #t view1)
;; #[view 1075519376]
; If edges are not turned on, you won’t see
; anything when shading is turned off.
; Turn shading off.
(view:shaded #f view1)
;; #[view 1075519376]
; The edges of the model should be visible.
; Note that silhouettes are not visible on the
; cylinder.
; Verify that shading is turned on
(view:shaded #t view1)
;; #[view 1075519376]
; Turn on polygon offset to make edges easier to see
(view:polygonoffset #t view1)
;; #[view 1075519376]
(view:coedges? view1)
;; #t
```

view:compute–extrema

Scheme Extension: Viewing

Action: Sets the view window size to the extrema of the part.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:compute-extrema** [view=active])

Arg Types: view view

Returns: unspecified

Errors: None

Description: Pans the specified view or the active view. This is performed by translating the view's target and eye position by the same gvector. Then the view's width and height are reset so the entities in the display fill as much of the view window as possible. Use the view:refresh extension to display the computed extrema.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:compute-extrema
; Determine the extrema of the display list,
; then reset the view window to the extrema.
(define block1
    (solid:block (position 5 5 5)
    (position 15 15 15)))
;; block1
(define block2
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block2
; OUTPUT Original

(view:compute-extrema)
;; ()
(view:refresh)
;; #[view 1075533160]
; OUTPUT Result
```

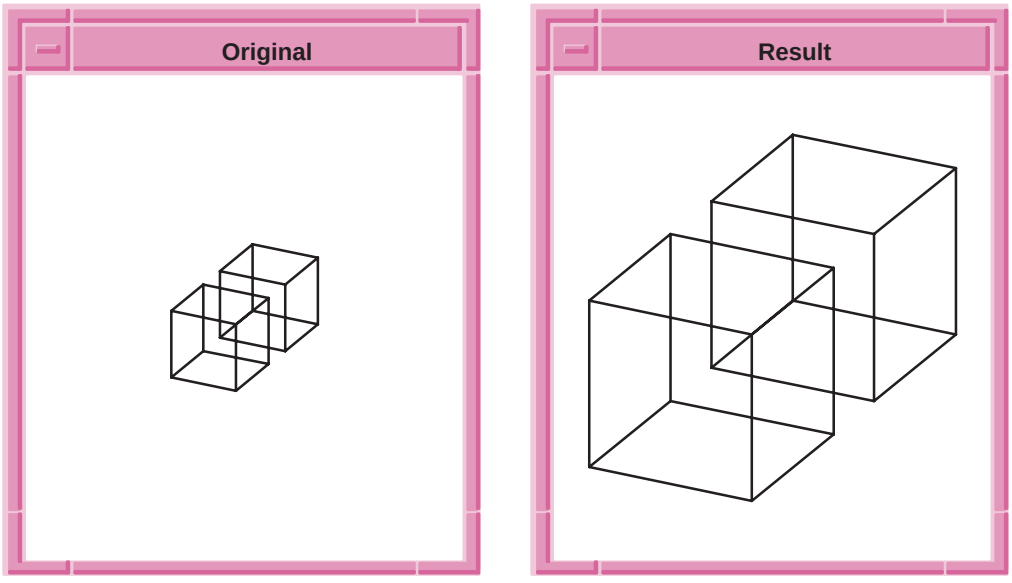


Figure 3-2. view:compute–extrema

view:context

Scheme Extension: Viewing

Action: Gets the name of the rendering context that is displayed in a view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: `(view:context [view=active])`

Arg Types: view view

Returns: string

Errors: None

Description: This extension returns a string that describes the type of rendering_context used to display a part in the view.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:context
; Define a new view.
(define view1 (view:dl))
;; view1
; Get the rendering context for a view.
(view:context view1)
;; "dl_context"
```

view:copy–to–clipboard

Scheme Extension: Viewing

Action: Copies a region of a window to the clipboard.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:copy–to–clipboard** event1 event2)

Arg Types:		
	event1	pick–event
	event2	pick–event

Returns: unspecified

Errors: None

Description: Copies a rectangular region to the clipboard of the given system. The rectangle is defined by diagonally opposite corners of the rectangular region. The corners are obtained by the event arguments, event1 and event2. They must be events in the same view window. They identify the type of event used, which is typically, the mouse (read event). If a clipboard is not supported on the platform, a message is sent to the output window.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:copy-to-clipboard
; Create a solid sphere.
(define sphere1 (solid:sphere (position 0 0 0) 25))
;; sphere1
; Render the solid sphere.
; Set render mode to full.
(render:set-mode "full")
;; ()
; Render the sphere
(render)
;; ()
; Copy the view to the clipboard.
(view:copy-to-clipboard (read-event) (read-event))
; With the mouse, select two positions on the screen
; diagonally so that the entity is in the box formed
; by the picked positions.
;; ()
```

view:debug

Scheme Extension: Viewing, Debugging

Action: Gets debug information for the display list.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:debug** [level] [view=active])

Arg Types:		
	level	integer
	view	view

Returns: unspecified

Errors: None

Description: This extension returns debug information for the display list in the specified or active view. If debug output is directed to a filename, the debug information is sent to that file. The optional argument level specifies the amount of information to be displayed. The values and their meaning are:

0, displays the rendering context and display list parent header.
1, displays parent info and solids header.
2, displays solids info and lumps header.
3, displays lumps info and shells header.
4, displays shells info and faces header.
5, displays faces info and loops header.
6, displays loops info and edges header.
7, displays edges info and display list polyline header.
8, displays display list polyline fill and points info.

The argument view identifies which view is used to determine the display list.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:debug
; Print debug information about the display list.
(view:debug)
; Rendering Context Type = dl_context
; DL_parent: color = 0, 1, 0,
; highlight = 0, inherit = 1
;; ()
```

view:dl

Scheme Extension: Viewing

Action: Creates a new view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax:

```
(view:dl [x–window y–window width height]
    [backup] [part])
```

Arg Types:		
	x–window	integer
	y–window	integer
	width	integer
	height	integer
	backup	boolean
	part	part

Returns: view

Errors: None

Description: The optional arguments x–window and y–window specify the *xy* location on the screen of the upper left–hand corner of the view window. The optional argument width specifies the horizontal size of the window in pixels. The optional argument height specifies the vertical size of the window in pixels. If no arguments are specified, a window is created based on the size of the display used. The default for this extension displays the view with the *y*–axis up, the *x*–axis to the right, and the *z*–axis coming out of the screen.

The default values for the location and size arguments on Windows systems are (values in parentheses are for non–Windows):

x–window . . . . . = 1 (0)
y–window . . . . . = 1 (0)
width . . . . . = 256 (600)
height . . . . . = 256 (600)

Any arguments specified are assumed to belong to parameters at the beginning of the argument list; therefore, any unspecified arguments receive their default values from the remainder of the default list. For example,

```
(view:dl 10 30 #t)
```

on a Windows system becomes:

```
(view:dl 10 30 256 256 #t)
```

The optional argument backup, when #t, buffers display information in memory, so that refresh operations to rendered images, for example, are not converted back to wireframe representations; the rendered image is refreshed. The default is #f. The optional argument part specifies which part is to be displayed in the view window.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:dl
; Create a new view.
(define view1 (view:dl 0 0 200 200))
;; view1
; Create a block.
(define block1
    (solid:block (position 5 5 5)
    (position 15 15 15)))
;; block1
; Get the eye position of view.
(view:eye view1)
;; #[position 0 0 500]
; Change the eye position for better viewing.
(view:set–eye (position 50 –100 50) view1)
;; #[position 0 0 500]
; Refresh view to see the results of
; the new eye position.
(view:refresh view1)
;; #[view 1076012760]
```

view:edge–hedgehog

Scheme Extension: Viewing

Action: Displays a hedgehog out of the derivatives of a curve.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax:

```
(view:edge–hedgehog edge [”u” u–min u–max ]
 [”der” der–type = 1] [”nu” nu = 20 ]
 [”scale” factor = 1])
```

Arg Types:		
	edge	edge
	u–min	double
	u–max	double
	nu	integer
	factor	double

Returns: view

Errors: none

Description: Displays hedgehogs for a given edge. The resulting arrows will represent the derivatives at evaluating in the u parameter provided.

der–type can be set to: first...= 1, second..= 2

nu can be set to the number of arrows.

The scale factor is applied to the resulting arrows.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in DL views only.

Example:

```
; view:edge–hedgehog
; Create a sample edge
(part:clear)
(view:dl)
(define e (edge:spline (list
       (position 0 0 0)(position 1 1 0)
       (position 2 0 0)(position 3 1 0))))
(iso)
(zoom–all)
(curve:domain e)
(view:edge–hedgehog e 'u 1 4 'nu 15 'scale 1 'der 1)
(view:edge–hedgehog e 'u 1 4 'nu 15 'scale 1 'der 2)
(view:hedgehog–clear)
```

view:edges

Scheme Extension: Viewing

Action: Enables and disables display of edges within an OpenGL view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: `(view:edges [edges] [view=active])`

Arg Types:		
	view	view
	edges	boolean

Returns: view

Errors: None

Description: The edges argument enables (#t) and disables (#f) a model's edges within a view. When enabled and when shading (view:shaded) is disabled, the result is similar to a wireframe representation only without the silhouette lines.

Shading (view:shaded) and edge (view:edges) display must be enabled for doing “fake” hidden line (view:hiddenline).

Displaying edges while shading of faces is turned off is not the same thing as a wire frame representation. An ACIS wire frame representation shows silhouette lines, whereas displaying edges does not. The following figure shows the silhouette lines of a wire frame representation of a cylinder (on the left) and the edges of a cylinder with face shading turned off (on the right).

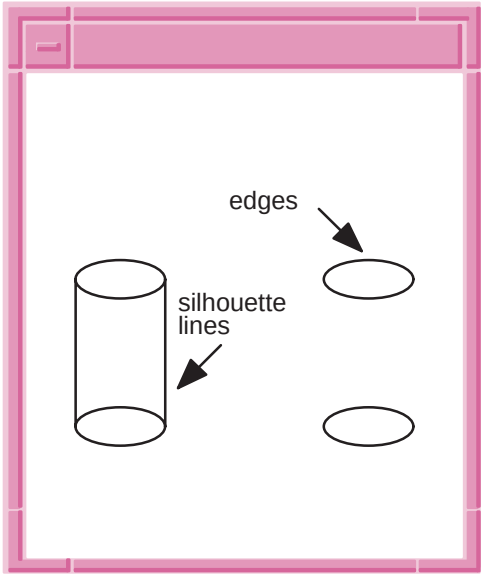


Figure 3-3. Silhouette lines versus Edges

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in OpenGL views only.

Example:

```
; view:edges
; Define a view.
(define view1 (view:gl))
;; #[view 1075519376]
; Define a block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 20 30)))
;; block1
(define cylinder1
    (solid:cylinder (position –10 –10 –10)
    (position –10 –10 30) 5))
;; cylinder1
; Verify that edges are turned on
(view:edges #t view1)
;; #[view 1075519376]
; If edges are not turned on, you won’t see
; anything when shading is turned off.
; Turn shading off.
(view:shaded #f view1)
;; #[view 1075519376]


; The edges of the model should be visible.
; Note that silhouettes are not visible on the
; cylinder. Verify that shading is turned on.
(view:shaded #t view1)
;; #[view 1075519376]
; Turn on polygon offset to make edges easier to see
(view:polygonoffset #t view1)
;; #[view 1075519376]
(view:edges? view1)
;; #t
```

view:edges?

Scheme Extension: Viewing

Action: Determines if edges are displayed in a view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:edges?** [view=active])

Arg Types: view view

Returns: view

Errors: None

Description: Refer to action.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in OpenGL views only.

Example:

```
; view:edges?
; Define a view.
(define view1 (view:gl))
;; #[view 1075519376]
; Define a block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 20 30)))
;; block1
(define cyl1
    (solid:cylinder
    (position –10 –10 –10)
    (position –10 –10 30) 5))
;; cyl1
; Verify that edges are turned on
(view:edges #t view1)
;; #[view 10748640]
; If edges are not turned on, you won’t see
; anything when shading is turned off.
; Turn shading off.
(view:shaded #f view1)
;; #[view 1075519376]
; The edges of the model should be visible.
; Note that silhouettes are not visible on the
; cylinder.
; Verify that shading is turned on
(view:shaded #t view1)
;; #[view 1075519376]
; Turn on polygon offset to make edges easier to see
(view:polygonoffset #t view1)
;; #[view 1075519376]
(view:edges? view1)
;; #t
```

view:face–hedgehog

Scheme Extension: Viewing

Action: Displays a hedge hog of surface normals on face or surface.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax:

```
(view:face–hedgehog face
["uv" u–min u–max v–min v–max]
["vec" vec–type = 0] ["nuv" nu = 20 nu = 20 ]
["scale" factor = 1]
    )
```

Arg Types:

u–min	double
u–max	double
v–min	double
v–max	double
nu	integer
nv	integer
factor	double

Returns: view

Errors: none

Description: Displays hedgehogs for a given surface. The resulting arrows will represent the vec–type evaluating in the u,v range provided.

vec–type can be set to: display normal...= 0, tangent u...= 1, tangent v...= 2

nuv can be set to the number of rows and columns in the u and v direction. uv can be set to the UV maximum to minimum box size. The scale factor is applied to the resulting arrows.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in DL views only.

Example:

```
;view:face–hedgehog
;Create a sample edge
(part:clear)
(view:dl)
(define e (edge:spline (list
       (position 0 0 0)(position 1 1 0)
       (position 2 0 0)(position 3 1 0))))
(define s (sweep:law e (gvector 0 0 1)))
(iso)
(zoom–all)
(define f (list–ref (entity:faces s)0))
(surface:domain f)
(view:face–hedgehog f 'uv –2 –1 0.12 .5 'nuv 15 15
'scale 0.1)
(view:face–hedgehog f 'vec 1 'uv –3 –2 0 .12 'nuv 15
15 'scale 0.1)
(view:face–hedgehog f 'vec 2 'uv –1 0  0.1 .5 'nuv 15
15 'scale 0.1)
(view:face–hedgehog f 'vec 3 'uv  –4 –3  0 1 'nuv 15
15 'scale 0.1)
(view:hedgehog–clear)
```

view:flush

Scheme Extension: Viewing

Action: Flushes the display buffer for a view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:flush** [view=active])

Arg Types: view view

Returns: view

Errors: None

Description: This extension flushes the display buffer of the specified or active view. The optional argument view specifies the view to flush. Use this extension only with X Windows. If the view is associated with a file, this command writes any buffered information to the file.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:flush
; Define a new view.
(define view1 (view:dl))
;; view1
; Create a block.
(define block1
    (solid:block (position 5 5 5)
    (position 15 15 15)))
;; block1
; Flush view's display buffer.
(view:flush view1)
;; #[view 1075519376]
```

view:hedgehog–clear

Scheme Extension: Viewing

Action: Clears all directional arrows

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:hedgehog–clear** [view=active])

Arg Types:	view	view

Returns: view

Errors: None

Description: Clears all directional arrows. view specifies what view will be cleared.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in DL views only.

Example:

```
;view:hedgehog–clear
;Create a sample edge
(part:clear)
(view:dl)
(define e (edge:spline (list
       (position 0 0 0)(position 1 1 0)
       (position 2 0 0)(position 3 1 0))))
(define s (sweep:law e (gvector 0 0 1)))
(iso)
(zoom–all)
(define f (list–ref (entity:faces s)0))
(surface:domain f)
(view:face–hedgehog f 'uv –2 –1 0.12 .5 'nuv 15 15
'scale 0.1)
(view:face–hedgehog f 'vec 1 'uv –3 –2 0 .12 'nuv 15
15 'scale 0.1)
(view:face–hedgehog f 'vec 2 'uv –1 0  0.1 .5 'nuv 15
15 'scale 0.1)
(view:face–hedgehog f 'vec 3 'uv  –4 –3  0 1 'nuv 15
15 'scale 0.1)
(view:hedgehog–clear)
```

view:hiddenline

Scheme Extension: Viewing

Action: Enables and disables "fake" hidden line within an OpenGL view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:hiddenline** [on–off] [view=active])

Arg Types:		
	on–off	boolean
	view	view

Returns: view

Errors: None

Description: The on–off argument enables (#t) and disables (#f) "fake" hidden line display on a per model basis. Multiple models within a scene may not produce a correct hidden line display. Shading (view:shaded) and edge (view:edges) display must be enabled. Faces are being rendered, but with the background. To help improve visibility, turning view:polygonoffset pushes the faceted polygon edges used in rendering back so the model's edges become more visible.

OpenGL rendering uses two buffers to control pixels: a z–buffer and a color buffer. When performing fake hidden–line, the color buffer is temporarily turned off. While off, a model's objects are drawn into the z–buffer. When a model's objects clash as a given *xy* pixel location, only the closest point (smallest number) is stored. Thus, a near model face wins out over a far model face. After the z–buffer has been filled, the color buffer is turned back on. The color buffer *xy* pixel is assigned the color of the associated z–buffer element. In this manner, the hidden–line calculations are automatically performed.

This OpenGL operation is called "fake" hidden–line because background faces and edges have not been removed or otherwise disabled from a display list. Moreover, it is dependent on the viewport (pixel) size and the faceting refinements. "Real" hidden–line, such as the Precise Hidden Line Component**,** performs the hidden–line calculation for the whole model and scene. It passes accurate model vector data to downstream devices, such as plotters, instead of pixel values. Vector data is scalable for plotters, while pixel data is not.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in OpenGL views only. Shading and edge display must be enabled. Works only on single models and not on multiple models within a single scene.

Example:

```
; view:hiddenline
; Define a block.
(define gl–view (view:gl))
;; gl–view
(define iso1 (view:set (position 100 –200 100)
    (position 0 0 0) (gvector 0 0 1) gl–view))
;; iso1
(define block1
    (solid:block (position 0 0 0)
    (position 25 20 30)))
;; block1
(define cyl1
    (solid:cylinder (position –10 –10 –10)
    (position –10 –10 30) 5))
;; cyl1
; Verify that shading is turned on
(view:shaded #t)
;; #[view 1075519376]
; Verify that edges are turned on
(view:edges #t)
;; #[view 1075519376]
; Turn on polygon offset to make edges easier to see
(view:polygonoffset #t)
;; #[view 1075519376]
; Turn hidden line on.
(view:hiddenline #t gl–view)
;; #[view 1075519376]
; The edges of the model should be visible.
; Note that silhouettes are not visible on the
; cylinder. Only the edges are visible.
```

view:hiddenline?

Scheme Extension: Viewing

Action: Determines if hiddenline is enabled in a view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:hiddenline?** [view=active])

Arg Types: view view

Returns: boolean

Errors: None

Description: Refer to action.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:hiddenline?
; Define a block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 20 30)))
;; block1
(define cylinder1 (solid:cylinder
    (position –10 –10 –10) (position –10 –10 30) 5))
;; cylinder1
; Verify that shading is turned on
(view:shaded #t)
;; #[view 1075519376]
; Verify that edges are turned on
(view:edges #t)
;; #[view 1075519376]
; Turn on polygon offset to make edges easier to see
(view:polygonoffset #t)
;; #[view 1075519376]
; Turn hidden line on.
(view:hiddenline #t)
;; #[view 1075519376]
; The edges of the model should be visible.
; Note that silhouettes are not visible on the
; cylinder. Only the edges are visible.
(view:hiddenline?)
;; #t
```

view:interleaf

Scheme Extension: Viewing, Image Output

Action: Creates an Interleaf image file of a view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax:

```
(view:interleaf [filename=plotfile.il] [color]
    [x–size=190 y–size=254] [view=active])
```

Arg Types:	filename	string
	color	boolean
	x–size	real
	y–size	real
	view	view

Returns: view

Errors: None

Description: The optional argument filename specifies a string for the filename where the displayed objects are saved; the default filename is plotfile.il. The argument color specifies whether the view should be drawn in color (#t) or black and white (#f). The optional arguments x–size and y–size specify the dimensions of the rectangle within which the image will be placed. These arguments need to be positive and represent millimeters. The defaults are 190. mm by 254 mm, which gives a half inch border when printed on 8.5” by 11” paper.

The optional argument view specifies the view object to send to the filename. If view is not specified, the default is the active view. This always uses a white background for wireframe images. Backgrounds for rendered images are treated as graphics.

The scale of a representation is determined by the relationship between the size of the viewport and the view width and height. For example, a default viewport size of 500 by 500 pixels, the default view can be doubled in viewed size by changing the view width and height from 200 to 100 units.

The resolution of the printed image is determined by the relationship between the size of the viewport and the maximum image size, specified when the file view is created.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:interleaf
; Define a new view.
(define view1 (view:dl))
;; view1
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 35 35 35)))
;; block1
; Create an Interleaf view 1" by 1"
(define ileaf–view
    (view:interleaf "block1.doc" 25.8 25.8 view1))
;; ileaf–view
; 1" at 300 dpi is 1200

; Another example where the size of the view
; and viewport are changed before an output image
; is made.
; Define a new view.
(define view2 (view:dl 100 200))
;; view2
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 35 35 35)))
;; block1
; This queries the current view size and then
; resizes the viewport. Then it makes the
; object appear smaller by changing the width
; and height parameters of the view.
(let ((w (view:width view2)))
    (view:set–viewport 0 0 300 300 view2)
    (view:set–size (* w 2) (* w 2) view2))
;; #[view 109820984]
; Refresh the view to see the changes
(view:refresh view2)
;; #[view 109829897]
; Create an Interleaf view 1" by 1"
(define ileaf–view2
    (view:interleaf "block2.il" 25.8 25.8 view2))
;; ileaf–view2
```

view:link–to–window

Scheme Extension: Viewing

Action: Maps a view to an externally–created window.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: `(view:link–to–window window [parent] [part])`

Arg Types:

window	integer
parent	integer
part	part

Returns: view

Errors: None

Description: Maps a view to an externally–created window. To associate a window with an external application, use this extension to assign the external (parent) application to the window. The argument window specifies the window identification number assigned to that window. To determine the current viewing parameters, use the env:views extension. The optional argument parent specifies the external window ID to associate with the visible window. It is used to resolve ambiguities in cases where the external application generates 16–bit window handles. If the optional argument part is specified, then the view is associated with that part; otherwise, it is associated with the active part.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:link–to–window
; Get a list of all current views.
(env:views)
;; (#[view 1075533160])
; Link a view to an externally created window.
(view:link–to–window 1075488368)
;; #[view 1075533160]
```

view:part

Scheme Extension: Viewing, Part Management

Action: Gets the part that is displayed in a view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: `(view:part [view=active])`

Arg Types:		
	view	view

Returns: part

Errors: None

Description: Returns the part object for the specified or active window. This is useful in determining the part displayed in one window, so that the same part can be displayed in another window with the view:set–part command.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:part
; Define a new view.
(define view1 (view:dl))
;; view1
; Get the part displayed in a view.
(view:part view1)
;; #[part 1]
(define next–view (view:dl))
;; next–view
(view:set–part (view:part view1) next–view)
;; ()
```

view:permanent–point

Scheme Extension: Viewing

Action: Displays a point on the specified view. The point remains in the view until view:permanent-point-clear is called.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax:

```
(view:permanent–point position [size=36]
    [view=active])
```

Arg Types:		
	position	position
	size	integer
	view	view

Returns: view

Errors: None

Description: Displays a point at the given position. size specifies the point size (default = 36). view specifies in which view the point is displayed.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in DL views only.

Example:
```
; view:permanent–point
; Create point with command.
(define apoint (view:permanent–point
    (position 0 0 0)))
;; apoint
```

view:permanent–point–clear

Scheme Extension: Viewing

Action: Clears all permanent points on the specified view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:permanent–point–clear** [view=active])

Arg Types: view — view

Returns: view

Errors: None

Description: Clears all permanent points. view specifies from which view the point(s) are cleared.

Warning: Calling this also clears any text that is present in the view.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in DL views only.

Example:
```
; view:permanent–point–clear
; Create point with command.
(define apoint (view:permanent–point
    (position 0 0 0)))
;; apoint
; Remove point from view.
(view:permanent–point–clear)
;; #[view 28639672]
```

view:polygonoffset

Scheme Extension: Viewing

Action: Enables and disables polygonoffset within an OpenGL view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: `(view:polygonoffset [on–off] [view=active])`

Arg Types:

on–off	boolean
view	view

Returns: view

Errors: None

Description: The on–off argument enables (#t) and disables (#f) polygon offsetting. The polygons refer to those used in faceting a model, which is a requirement for rendering. When enabled and when model edges (view:edges) enabled, this command pushes the faceted polygon edges back so the model edges become more visible. Prevents z–buffer problems when displaying edges on top of the shaded object.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in OpenGL views only. view:edges should be enabled to see any effect.

Example:

```
; view:polygonoffset
; Define a view.
(define view1 (view:gl))
;; #[view 1075519376]
; Define a block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 20 30)))
;; block1
(define cylinder1
    (solid:cylinder (position –10 –10 –10)
    (position –10 –10 30) 5))
;; cylinder1
; Verify that edges are turned on
(view:edges #t view1)
;; #[view 1075519376]
; Turn on polygon offset to make edges easier to see
(view:polygonoffset #t view1)
;; #[view 1075519376]
; If edges are not turned on, you won't see
; anything when shading is turned off.
```

view:polygonoffset?

Scheme Extension: Viewing

Action: Determines if polygonoffest is enabled in a view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:polygonoffset?** [view=active])

Arg Types:	view	view

Returns: boolean

Errors: None

Description: Refer to action.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in OpenGL views only.

Example:

```
; view:polygonoffset?
; Define a view.
(define view1 (view:gl))
;; #[view 1075519376]
; Define a block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 20 30)))
;; block1
(define cylinder1
    (solid:cylinder (position –10 –10 –10)
    (position –10 –10 30) 5))
;; cylinder1
; Verify that edges are turned on
(view:edges #t view1)
;; #[view 1075519376]
; Turn on polygon offset to make edges easier to see
(view:polygonoffset #t view1)
;; #[view 1075519376]
; If edges are not turned on, you won't see
; anything when shading is turned off.
(view:polygonoffset?)
;; #t
```

view:postscript

Scheme Extension: Viewing, Image Output

Action: Creates a PostScript file of a view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax:

```
(view:postscript [filename=plotfile.ps] [color=#f]
    [x–size=190 y–size=254] [view=active])
```

Arg Types:		
	filename	string
	color	boolean
	x–size	real
	y–size	real
	view	view

Returns: view

Errors: None

Description: The optional argument filename specifies a string filename to send the displayed objects. The default filename is plotfile.ps. The optional x–size and y–size specify the dimensions in millimeters of a rectangle into which you want the printed image of the viewport to fit as tightly as possible without distortion. The optional view specifies the view object to send to the filename. If view is not specified, the default is the active view.

The optional argument filename specifies a string for the filename where the displayed objects are saved; the default filename is plotfile.ps. Wireframe images are written in vector graphics format. Temporary text uses PostScript fonts rather than as graphical images. Rendered images are written in raster form.

The argument color specifies whether the view should be drawn in color (#t) or black and white (#f). This always uses a white background for wireframe images. Backgrounds for rendered images are treated as graphics.

The optional arguments x–size and y–size specify the dimensions of the rectangle within which the image will be placed. These arguments need to be positive and represent millimeters. The defaults are 190. mm by 254 mm, which gives a half inch border when printed on 8.5” by 11” paper.

The optional argument view specifies the view object to send to the filename. If view is not specified, the default is the active view.

The scale of a representation is determined by the relationship between the size of the viewport and the view width and height. For example, a default viewport size of 500 by 500 pixels, the default view can be doubled in viewed size by changing the view width and height from 200 to 100 units.

The resolution of the printed image is determined by the relationship between the size of the viewport and the maximum image size, specified when the file view is created.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:postscript
; Define a new view.
(define view1 (view:dl))
;; view1
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 35 35 35)))
;; block1
; Create a PostScript view.
(view:postscript ”geom.ps” 100 100 view1)
;; #[view 1073761432]
```

view:refresh

Scheme Extension: Viewing

Action: Refreshes the display list for a view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:refresh** [view=active] [clear])

Arg Types:		
	view	view
	clear	boolean

Returns: view

Errors: None

Description: Call this extension after changing viewing parameters. This displays changes made to the part in the specified or active window. The optional argument view specifies the view to refresh. If view is not specified, the active view is used.

If the optional argument clear is #t or not present, the window is cleared and the wireframe is redisplayed from the display list contents. If the argument clear is #f, then the view is not cleared before redisplaying the display list.

If the view is associated with a file, a “new page” instruction is written to the file when the view:clear is called with #t or no argument clear. If the argument clear is #f, no “new page” is written to the file. view:refresh places the image on the following page of the file.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:refresh
; Define a top view.
(define top (view:dl))
;; top
; Alter the perspective for better viewing
(view:set
    (position 50 –100 50)
    (position 0 0 0) (gvector 0 0 1) top)
;; #[view 1076063416]
; Define a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Define rendering parameters so it does it in color.
(entity:set–material–color block1 6)
;; ()
; Perform rendering in the top view.
(render top)
;; ()
; Refresh the display list for the top view.
(view:refresh #f)
;; #[view 1075519376]
```

view:resize

Scheme Extension: Viewing

Action: Sets correct aspect ratio and centers model after the window has been resized.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:resize** [view=active])

Arg Types: view view

Returns: view

Errors: None

Description: Computes the width and height of a view after the window has been resized so the aspect ratio remains correct and the model is recentered. The viewport is reset so that it coincides with the window's client area. Resets the view height and width so the scale remains unchanged. Centers the model within the viewport. However, view:refresh makes these changes visible. The optional argument view specifies the view to resize. If view is not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:resize
; Define a new view.
(define view1 (view:dl))
; Resize the current view.
;; view1
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 35 35 35)))
;; block1
; Resize the active view.
(view:resize)
;; #[view 1075533160]
; Recompute a view's width and height after
; window resizing, and recenter the model.
(view:resize view1)
;; #[view 1075519376]
(view:refresh view1)
;; #[view 1075519376]
```

view:set–context

Scheme Extension: Viewing

Action: Sets the rendering context that is displayed in the view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax:

```
(view:set–context [view=active]
    [context=dl_context])
```

Arg Types:		
	view	view
	context	string

Returns: boolean

Errors: None

Description: Use this extension to change the rendering_context used to render the PART in the specified view. context is specified as a string. It first removes view from its current rendering context and adds it to the rendering context corresponding to the input string. If the new rendering context does not exist, it is created. If no argument for context is provided, the default is “dl_context”.

This extension returns #t if the rendering context change is successful; otherwise, it returns #f.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in OpenGL views only.

Example:

```
; view:set–context
; Define a new view.
; You must have OpenGL.
(define view1 (view:gl))
;; view1
(define view2 (view:dl))
;; view2
(view:context view1)
;; “gl_context”
(view:context view2)
;; “dl_context”
; Set the rendering_context for the new view.
(view:set–context view2 ”dl_context”)
;; #[view 1075541080]
(view:context view2)
;; “dl_context”
```

view:set–part

Scheme Extension: Viewing, Part Management

Action: Sets the view in which a part is displayed.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:set–part** [view=active] [part])

Arg Types:		
	view	view
	part	part

Returns: unspecified

Errors: None

Description: Sets the view in which a part is displayed. Uses the specified view or the active view. Also uses the specified part or the active part. If necessary, creates a new rendering context of the same type to which the view initially belonged. Refreshes the view in a window with all the elements in its display list. If the view is a file, it writes a “new page” instruction to the file; then it writes information to the file according to the displayed objects of the new part.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–part
; Define a new view.
(define v1 (view:dl))
;; v1
; Define a new part.
(define p1 (part:new))
;; p1
; Get the part displayed in the view.
(view:part v1)
;; #[part 1]
; Set the new part to be displayed in the view.
(view:set–part v1 p1)
;; ()
; Get the part displayed in a view.
(view:part v1)
;; #[part 2]
```

view:shaded

Scheme Extension: Viewing

Action: Enables and disables shading of faces within an OpenGL view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:shaded** [on–off] [view=active])

Arg Types:		
	view	view
	on–off	boolean

Returns: view

Errors: None

Description: The on–off argument enables (#t) and disables (#f) shading within a view. When disabled and model edges (view:edges) enabled, the result is similar to a wireframe representation only without the silhouette lines.

Shading (view:shaded) and edge (view:edges) display must be enabled for doing “fake” hidden line (view:hiddenline).

Displaying edges while shading of faces is turned off is not the same thing as a wire frame representation. An ACIS wire frame representation shows silhouette lines, whereas displaying edges does not. The following figure shows the silhouette lines of a wire frame representation of a cylinder (on the left) and the edges of a cylinder with face shading turned off (on the right).

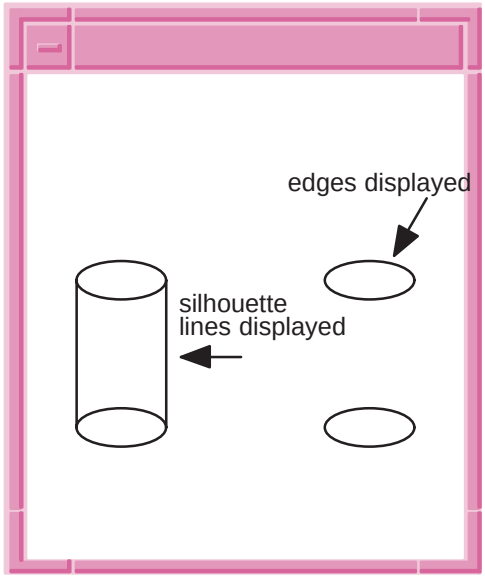


Figure 3-4. Silhouette lines versus Edges

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in OpenGL views only. When shading is turned off, view:edges should be enabled, otherwise nothing will be visible.

Example:

```
; view:shaded
; Define a view.
(define view1 (view:gl))
;; #[view 1075519376]
; Define a block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 20 30)))
;; block1
(define cyl1
    (solid:cylinder (position –10 –10 –10)
    (position –10 –10 30) 5))
;; cyl1
; Verify that edges are turned on
(view:edges #t view1)
;; #[view 1075519376]
; If edges are not turned on, you won’t see
; anything when shading is turned off.
; Turn shading off.
(view:shaded #f view1)
;; #[view 1075519376]
; The edges of the model should be visible.
; Note that silhouettes are not visible on the
; cylinder.
; Verify that shading is turned on
(view:shaded #t view1)
;; #[view 1075519376]
; Turn on polygon offset to make edges easier to see
(view:polygonoffset #t view1)
;; #[view 1075519376]
```

view:shaded?

Scheme Extension: Viewing

Action: Determines if shading is on for a view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:shaded?** [view=active])

Arg Types:	view	view

Returns: view

Errors: None

Description: Refer to action.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in OpenGL views only.

Example:

```
; view:shaded?
; Define a view.
(define view1 (view:gl))
;; #[view 1075519376]
; Define a block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 20 30)))
;; block1
(define cylinder1
    (solid:cylinder (position –10 –10 –10)
    (position –10 –10 30) 5))
;; cylinder1
; Verify that edges are turned on
(view:edges #t view1)
;; #[view 1075519376]
; If edges are not turned on, you won’t see
; anything when shading is turned off.
; Turn shading off.
(view:shaded #f view1)
;; #[view 1075519376]
; The edges of the model should be visible.
; Note that silhouettes are not visible on the
; cylinder.
; Verify that shading is turned on
(view:shaded #t view1)
;; #[view 1075519376]
; Turn on polygon offset to make edges easier to see
(view:polygonoffset #t view1)
;; #[view 1075519376]
(view:shaded?)
;; #t
```

view:sil

Scheme Extension: Viewing

Action: Enables and disables display of silhouettes.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:sil** [view=active] [sil])

Arg Types:

view	view
sil	boolean

Returns: view

Errors: None

Description: The sil argument enables (#t) and disables (#f) a model's silhouettes within a view. The result is similar to a wireframe representation.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in OpenGL views only.

Example:

```
; view:sil
; Define a view.
(define view1 (view:gl))
;; #[view 1075519376]
; Define a block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 20 30)))
;; block1
(define cylinder1
    (solid:cylinder (position –10 –10 –10)
    (position –10 –10 30) 5))
;; cylinder1
; Verify that edges are turned on
(view:edges #t view1)
;; #[view 1075519376]
```

```
; If edges are not turned on, you won't see
; anything when shading is turned off.
; Turn shading off.
(view:shaded #f view1)
;; #[view 1075519376]
; The edges of the model should be visible.
; Note that silhouettes are not visible on the
; cylinder.
; Verify that shading is turned on
(view:shaded #t view1)
;; #[view 1075519376]
; Turn on polygon offset to make edges easier to see
(view:polygonoffset #t view1)
;; #[view 1075519376]
(view:sil #t view1)
;; #[view 1075519376]
(view:sil? view1)
;; #t
```

view:sil?

Scheme Extension: Viewing

Action: Determines if silhouettes are displayed in a view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:sil?** [view=active])

Arg Types: view view

Returns: boolean

Errors: None

Description: Refer to action.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in OpenGL views only.

Example:

```
; view:sil?
; Define a view.
(define view1 (view:gl))
;; #[view 1075519376]
; Define a block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 20 30)))
;; block1
(define cylinder1
    (solid:cylinder (position –10 –10 –10)
    (position –10 –10 30) 5))
;; cylinder1
; Verify that edges are turned on
(view:edges #t view1)
;; #[view 1075519376]
; If edges are not turned on, you won’t see
; anything when shading is turned off.
; Turn shading off.
(view:shaded #f view1)
;; #[view 1075519376]
; The edges of the model should be visible.
; Note that silhouettes are not visible on the
; cylinder.
; Verify that shading is turned on
(view:shaded #t view1)
;; #[view 1075519376]
; Turn on polygon offset to make edges easier to see
(view:polygonoffset #t view1)
;; #[view 1075519376]
(view:sil #t view1)
;; #[view 1075519376]
(view:sil? view1)
;; #t
```

view:tcoedges

Scheme Extension: Viewing

Action: Enables and disables display of tolerant coedges within an OpenGL view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:tcoedges** [view=active] [edge–on–off])

Arg Types:	view edge–on–off	view boolean
Returns:	view	
Errors:	None	

Description: This extension enables (#t) and disables (#f) a model's tolerant coedges within a GL view. When enabled and when shading (view:shaded) is disabled, the result is similar to a wireframe representation only without the silhouette lines.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in OpenGL views only.

Example:

```
; view:tcoedges
; Define a view.
(define view1 (view:gl))
;; #[view 1075519376]
; Define a block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 20 30)))
;; block1
(define cylinder1
    (solid:cylinder (position –10 –10 –10)
    (position –10 –10 30) 5))
;; cylinder1
; Verify that edges are turned on
(view:tcoedges #t view1)
;; #[view 1075519376]
; If tcoedges are not turned on, you won't see
; anything when shading is turned off.
; Turn shading off.
(view:shaded #f view1)
;; #[view 1075519376]
; The tcoedges of the model should be visible.
; Note that silhouettes are not visible on the
; cylinder. Verify that shading is turned on.
(view:shaded #t view1)
;; #[view 1075519376]
; Turn on polygon offset to make tcoedges
; easier to see
(view:polygonoffset #t view1)
;; #[view 1075519376]
(view:tcoedges? view1)
;; #t
```

view:tcoedges?

Scheme Extension: Viewing

Action: Determines if tolerant coedges are displayed in a view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: `(view:tcoedges? [view=active])`

Arg Types:	view	view

Returns: view

Errors: None

Description: Refer to Action.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in OpenGL views only.

Example:

```
; view:tcoedges?
; Define a view.
(define view1 (view:gl))
;; #[view 1075519376]
; Define a block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 20 30)))
;; block1
(define cyl1
    (solid:cylinder
    (position –10 –10 –10)
    (position –10 –10 30) 5))
;; cyl1
; Verify that tcoedges are turned on
(view:tcoedges #t view1)
;; #[view 1075519376]
; If tcoedges are not turned on, you won’t see
; anything when shading is turned off.
; Turn shading off.
(view:shaded #f view1)
;; #[view 1075519376]
; The tcoedges of the model should be visible.
; Note that silhouettes are not visible on the
; cylinder.
; Verify that shading is turned on
(view:shaded #t view1)
;; #[view 1075519376]
; Turn on polygon offset to make tcoedges easier
; to see
(view:polygonoffset #t view1)
;; #[view 1075519376]
(view:tcoedges? view1)
;; #t
```

view:vertices

Scheme Extension: Viewing

Action: Enables and disables display of vertices.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:vertices** [view=active] [vertices])

Arg Types: view — view
vertices — boolean

Returns: view

Errors: None

Description: The vertices argument enables (#t) and disables (#f) vertex display on a per model basis.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in OpenGL views only.

Example:

```
; view:vertices
; Display vertices in the active view.
(define view1 (view:gl))
;; #[view 1075519376]
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
(view:vertices #t)
;; #[view 1075519376]
```

view:vertices?

Scheme Extension: Viewing

Action: Determines if vertices are displayed in the view.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:vertices?** [view=active])

Arg Types: view — view

Returns: boolean

Errors: None

Description: Refer to action.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL. Works in OpenGL views only.

Example:

```
; view:vertices?
; Display vertices in the active view.
(define view1 (view:gl))
;; #[view 1075519376]
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
(view:vertices?)
;; #f
```

view:zoom–window

Scheme Extension: Viewing

Action: Zooms a view to a size defined by two pick events.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**view:zoom–window** event1 event2 [view=active])

Arg Types:		
	event1	pick–event
	event2	pick–event
	view	view

Returns: view

Errors: None

Description: Zooms the specified view or the active view to a size defined by two pick events. The pick events define a rectangle. The area within the rectangle is modified to fit as tightly as possible within the associated viewport while maintaining the aspect ratio. The arguments event1 and event2 specify the type of pick event used, typically, the mouse (read event). The optional argument view specifies the view to zoom, and it defaults to the active view. This allows the user to specify a region in one view, yet zoom a different view. The two events must be within the same window. The zoom affect is achieved by panning the view and offsetting both the eye position and the target by the same gvector. It then enlarges the display image and reduces the view width and height by the same factor. A view:refresh is needed to visualize the changes.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:zoom-window
; Define a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 15 15 15)))
;; block1
; OUTPUT Original

(view:zoom-window (read-event) (read-event))
; Use the mouse to select diagonal
; corners of the block.
;; #[view 1075533160]
(view:refresh)
;; #[view 1075533160]
; OUTPUT Result
```

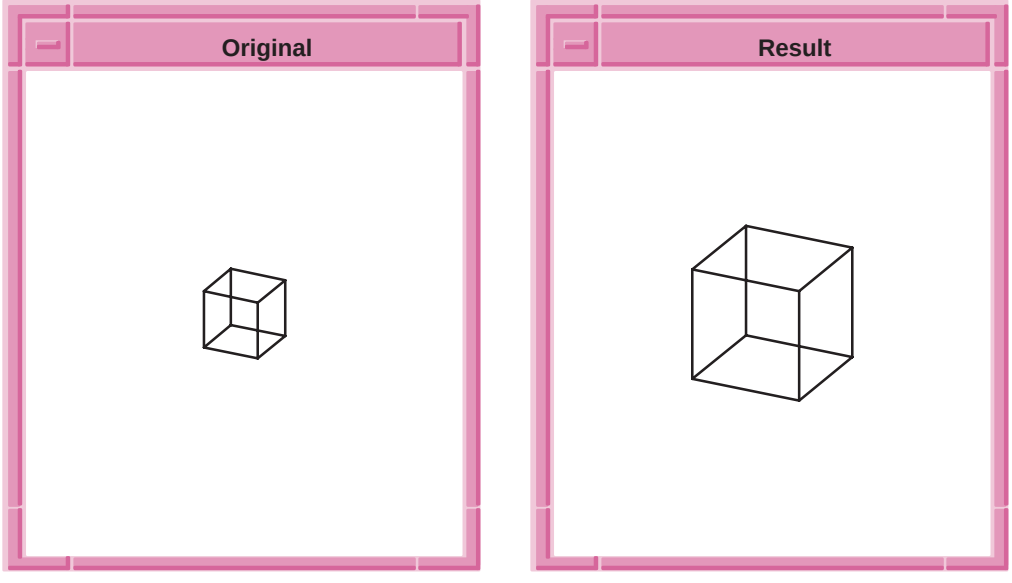


Figure 3-5. view:zoom–window