

IMAGER XLISP INTERFACE

Marc W. Cygnus

October 1992

Copyright (c) 1992 by the Washington Technology Center.

*** "Conceptual Issues" documentation, mentioned at times below, doesn't quite exist yet. Soon... ***

(The form of this documentation is preliminary; I'll try to have a more friendly arrangement in another month)

Now an IMPORTANT NOTE about long and short term software stability:

The Imager is an evolving software package, as is most all of our work-in-progress. However, constant evolution does not necessarily imply backwards incompatibility, and in this case it must not, without agreement among the groups who will have to deal with the changes.

The functions described in this document, except as noted, are stable as of 10-02-92. You may build code which depends upon them. They may be expanded at some point in the future with optional parameters, but the function name, its required parameters, and their ordering is stable. If there is a consensus that something needs to be changed, then it will be, but otherwise I guarantee the stability of the functions herein.

The two exceptions to this rule are new functions under development (marked `fragile') and temporary functions used only during debugging (marked `temporary'). Fragile functions may be used at your own risk until they become stable. Temporary functions may be used for debugging (if for some strange reason they're useful to you), but are not intended for general use and will likely go away sooner or later.

Several functions below are marked fragile only because small changes need to be made before they're stable. If this is the case, the intended change will be documented so that you may plan for it.

`im-imager-init` &optional <videosplitter_flag>

Initializes the imager internals and **must** be called before any other functions may be used. If the optional arg is t, the videosplitter hardware will be used (if present) when `im-stereo-view` is called.

Ex: `(im-imager-init)`
`(im-imager-init nil)` ; same as above
`(im-imager-init t)` ; use videosplitter

`im-stereo-view` <iod> <offset_x> <offset_y> <offset_z> [Fragile]

Sets up a stereo viewport pair in the lower (default) or upper two screen quadrants. The perspective projections are calculated to match the physical characteristics of the VPL EyePhones(tm).

<iod> is the interocular distance of the user given in meters. The last three args comprise a 3D offset vector which relates the center of the headtracking sensor to a point midway between the two eyes of the EyePhone (see external diagram). The offset is given in meters.

The function's operation is stable. The only thing which is to change is the offset specification. Currently, `stereo-view` expects three reals for the offset. The offset should really be a true vector, however.

Ex: `(im-stereo-view 0.06 0.0 0.18 0.06)` ; 6 cm interocular,
; sensor is 18 cm up and 6 cm back from EyePhone reference pt.

; this will change to: `(im-stereo-view 0.06 '#(0.0 0.18 0.06))`

`im-set-ocular` <iod>

Changes the interocular distance for the current stereo view. <iod> is the new interocular distance to use, given in meters.

This function only works with stereo views; it is ignored if used with a mono view.

Ex: `(im-set-ocular 0.065)` ; set interocular to 6.5 cm

`im-window-view` &optional <width> [<x-loc> <y-loc>]

Sets up a nonresizable mono viewport in an IRIX window. With no arguments, creates a 600 x 400 window which approximates the view one would get looking through EyePhones(tm). If <width> is given, the window will be <width> wide and 0.66*<width> high (3:2 aspect ratio). If <x-loc> and <y-loc> are specified after <width>, the window will be automatically placed with its upper left corner at (<x-loc>, <y-loc>).

Ex: `(im-window-view)` ; open a user-placed 600 x 400 mono window
`(im-window-view 1000)` ; open a user-placed 1000 x 667 mono window
`(im-window-view 300 200 100)`
; auto open a 300 x 200 mono window at (200,100)

`im-new-object` <groupID> <def-or-path>

Creates one or more new graphical objects. <groupID> specifies the initial group membership for all objects defined during one call to the function. Presently, objects may be members of only one group at a time. <groupID> may be up to 31 bits long (unsigned).

Objects are collections of graphical primitives specified in DOG format (see the revised DOG documentation, and also the Conceptual Issues documentation). <def-or-path> is a string which may be either the actual DOG text or a pathname to a file. If a pathname is specified, it must be either an absolute pathname or it must be literally referenced from the directory current at runtime (ie: it must begin with "/" or "./").

Returns a list of dotted pairs, one for each object defined by the given DOG text, or nil if an error occurred. If one or more objects were successfully defined, each dotted pair in the returned list will contain the name of the object as given in the DOG text (or nil if no name was given) and the object identifier which must be used to refer to the object in all future imager operations on that object. The object ID is generated by the imager internals and is a 31 bit unsigned quantity unique to each object.

Ex:

```
(im-new-object 0 "object testObject { ... }")  
; DOG text given literally.  
; returns ((testObject . ##)), where ## is a unique ID
```

```
(im-new-object 8118 "/home/ernie/good.dog")
; good.dog might contain:
;   object { ... }
;   object second-object { ... }
;   object LastObject { ... }

; returns:
;   ( (nil . #1)
;     (second-object . #2)
;     (LastObject . #3)
;   )
; ...where #1, #2, and #3 are unique IDs for each object.
; All objects will have initial group membership of 8118.
```

```
(im-new-object 456 "./cow.dog")
; cow.dog is in the directory which is current at runtime.
```

`im-instance-object <groupID> <template-objID>` [Fragile]

Creates a new object directly from an existing "template" object, given by <template-objID>, which must exist. <groupID> specifies the object's initial group membership. Returns the new object's ID (a single fixnum).

For the moment, this call is effectively a copy-object-geometry call, since the template object and the newly formed object have independent attributes (color, transform, polymode, etc). In fact, the new object will have all its attributes set to default values; for example, if the template object has had a scaling and transformation applied to it, the new object will be created with the same untransformed geometry, but with unity scaling and identity transform (ie: don't let this surprise you).

This function is marked 'fragile' because it's new and some things don't work quite correctly yet, namely texturing. Undefined behavior will result if either the template object is textured or you attempt to texture the new object.

```
Ex: (im-instance-object 33 12)
; returns a unique ID for the new object, which will be
; an instance of object 12, in group 33, with its attributes
; set to default values.
```

`im-delete-object <objID>`

Deletes graphical object <objID>.

`im-xform-object <objID> <absmatrix>`

Transforms object `<objID>` by absolute matrix `<absmatrix>`; `<absmatrix>` replaces the object's position/orientation matrix.

Ex: `(im-xform-object 2 '#(...))`

`im-xform-object-delta <objID> <deltamatrix>`

Transforms object `<objID>` by delta matrix `<deltamatrix>`; the object's position/orientation matrix is preconcatenated with `<deltamatrix>`.

Ex: `(im-xform-object-delta 2 '#(...))`

`im-xform-group <groupID> <absmatrix>`

Transforms all objects belonging to group `<groupID>` by absolute matrix `<absmatrix>`. Works just like `im-xform-object`, except that all objects are transformed before a new frame is generated (ie: if there are ten objects in group `<groupID>`, `im-xform-group` guarantees that all ten objects will be in their new positions before a new graphical view is drawn).

If `<groupID>` does not exist, the function is essentially a no-op.

`im-xform-group-delta <groupID> <deltamatrix>`

Transforms all objects belonging to group `<groupID>` by delta matrix `<deltamatrix>`. Works just like `im-xform-object-delta`, except that all objects are transformed before a new frame is generated, as with `im-xform-group`.

`im-scale-object <objID> <scale-vector>`

[Fragile]

Scales object `<objID>` in object coordinates by the 3D scale vector `<scale-vector>`, which is `(sx sy sz)` where scales are reals. The scaling effects occur before any world-coordinate transforms specified by the `im-xform-*` commands.

Ex: `(im-scale-object 3 '#(2.0 1.0 1.0)) ; scales object x2 in X dir.`

im-set-obj-group <objID> <groupID> [Fragile]

Specifies a new group membership <groupID> for object <objID>, which must already exist.

This function is marked fragile pending a decision on whether or not to support general, arbitrary grouping. Presently, an object may be a member of only one group at a time.

im-drawframe

Allows the rendering process to begin a new frame. This function should be called repeatedly, preferably once through each "modelling" loop and at a known location in time. Object operations occurring after a given im-drawframe are guaranteed not to be displayed until the next im-drawframe.

Ex: (loop ...
 ...
 (im-drawframe))

im-set-void-color <r> <g>

Sets the absolute color of the void (ie: the `background' of the drawing area). <r>, <g>, and are normalized to [0.0,1.0], and specify the color of the void in terms of proportions of the primary colors.

im-set-obj-color <objID> <r> <g>

Sets the ambient and diffuse reflectivity of object <objID>, which must already exist. <r>, <g>, and are normalized to [0.0,1.0], and represent the percentage of a given primary color the object will reflect diffusely.

This function will be expanded in the near future to accept either three reals or a single three-element vector as the color specification. The function does not currently accept a vector, only three separate reals.

Ex: (im-set-obj-color 2 0.0 1.0 0.0) ; saturated, high intens. green

; this function will be expanded so that you may alternately do:
; (im-set-obj-color 2 '#(0.0 1.0 0.0))

`im-set-group-color <groupID> <r> <g> `

Sets the ambient and diffuse reflectivity of all objects belonging to group `<groupID>`. Works just like `im-set-obj-color`, except that all objects are colored before a new frame is generated.

As with `im-set-obj-color`, this function will be expanded in the near future to accept either three reals or a single three-element vector as the color specification.

`im-set-obj-polymode <objID> <mode>` [Fragile]

Sets the polygon drawing mode of object `<objID>`, which must already exist. `<mode>` must be one of:

- 2 - draw solid filled polygons
- 1 - draw wireframe polygons
- 0 - only draw points at polygon vertices [Fragile option]

Ex: `(im-set-obj-polymode 12 1)` ; set obj 12 to wireframe

`im-xform-view <absmatrix>`

Transforms the viewpoint by absolute matrix `<absmatrix>`; `<absmatrix>` replaces the current view orientation matrix.

Viewpoint transformation is identical to object transformation. That is, if a matrix `M` transforms an object forward two meters, the same matrix `M` applied to the viewpoint will move the view forward two meters, not the world.

Ex: `(im-xform-view '#(...))`

`im-xform-view-delta <deltamatrix>`

Transforms the viewpoint by delta matrix `<deltamatrix>`; the view orientation matrix is preconcatenated with `<deltamatrix>`.

Ex: `(im-xform-view-delta '#(...))`

`im-crystaleyes-view <iod> <scrw> <scrh> <scrd> <vout>` [Fragile]

Sets up a stereo viewport pair for the CrystalEyes LCD shutter system. See external documentation for information on specifying view volume parameters.

This function is experimental. It has been tested and appears to work with no problems, however. If you have an opportunity to use it, please do and report any bugs to cygnus@hitl.washington.edu, but it is not extremely high on the immediate to-do list.

`im-new-texture <tex-path> &optional <alpha-path>` [Fragile]

Defines a new texture map. `<tex-path>` and `<alpha-path>` if given are pathnames to PPM, PGM, or PBM files.

This function currently supports four different texture types: one-component, one-component plus alpha, three-component, and three-component plus alpha.

The texture ID of the newly defined texture is returned on success, nil otherwise.

This function by default will cause the texture to be wrapped in the s and t directions; in the future, wrapping or clamping behaviour may be specified by additional optional parameters.

`im-set-obj-texture <objID> <texID>`

Associates a texturemap `<texID>` with an object `<objID>`. This function may be called more than once to change object/texturemap associations.

`im-set-obj-tex-mapping <objID> <tg-point> <tg-s-vector> <tg-t-vector>`

Enables automatic texture vertex generation for object `<objID>` and specifies the origin and orientation of the texture plane in 3-space, which determines how a texture is applied to an object (for more information on texturemapping and automatic texture application, see the Conceptual Issues documentation).

`im-enable-obj-texture <objID> &optional <flag>`

Enables or disables texturing for object `<objID>`, which must exist. If called with only `<objID>`, it will enable texturing; otherwise, the optional argument `<flag>` specifies whether to enable or disable texture (t or nil, respectively).

In order for an object to actually appear with texture, however, a texture must have been defined with `im-new-texture` and applied with `im-set-obj-texture`.

Ex: `(im-enable-obj-texture 12)` ; enables texturing for obj 12
`(im-enable-obj-texture 12 t)` ; same as above
`(im-enable-obj-texture 15 nil)` ; disables texturing for obj 15)

`im-xform-obj-texture <objID> <absmatrix>`

Transforms the origin and orientation of the texture mapping plane associated with object `<objID>`, which must already exist. Automatic texture vertex generation must already have been enabled for the object (ie: `im-set-obj-tex-mapping` must have been called). The texture plane is transformed by absolute matrix `<absmatrix>`, which specifies a new position and orientation relative to the plane's original position as given in the call to `im-set-obj-tex-mapping`. (for more information on texturemapping and automatic texture application, see the Conceptual Issues documentation)

`im-xform-obj-texture-delta <objID> <deltamatrix>`

Transforms the origin and orientation of the texture mapping plane associated with object `<objID>`; works like `im-xform-obj-texture` (see above), except `<deltamatrix>` specifies a new texture plane position and orientation relative to the plane's current position and orientation.

`im-scale-obj-texture <objID> <s-scale> <t-scale>`

Specifies a texturemap scale factor, in the texture's `s` and `t` directions, for object `<objID>`, which must already exist. Texture scaling only works when automatic texture vertex generation has been enabled for an object. `<s-scale>` and `<t-scale>` are reals. (for more information on texturemapping and automatic texture application, see the Conceptual Issues documentation)

`im-set-obj-visible <objID> <visibility>`

Changes the visibility of object `<objID>`, which must already exist. `<visibility>` is `t` or `nil`, and makes the object visible or invisible, respectively.

Ex: `(im-set-obj-visible 11 nil)` ; set obj 11 invisible

`im-set-group-visible <groupID> <visibility>`

Changes the visibility of all objects belonging to group `<groupID>`. Works just like `im-set-obj-visible`, except that all objects are updated before a new frame is generated.

If `<groupID>` does not exist, the function is essentially a no-op.

`im-set-obj-visible-default <def-visibility>`

Sets the default visibility of objects created with `im-new-object`. `<def-visibility>` is `t` or `nil`. If `t`, all new objects appear automatically after they are defined with `im-new-object`. If `nil`, new objects are defined and may be transformed, but they are invisible and must be made visible via a call to `im-set-obj-visible` (or `im-set-group-visible`) before they will be rendered.

`im-statistics <stats-flag>` [Fragile]

This function currently prints an estimate of rendered frames-per-second in the center of the output window. The characters are too small currently to be very useful in the EyePhones(tm), however.

In the future, it will control miscellaneous statistics such as current rendering frame rate, object polygon count reporting, etc.

`im-set-global-light <dir-vector> <color> <ambient-color>` [Fragile]

Changes the direction and color of the single (global) light source. For now, there is only one light source in all scenes. It is an infinitely-distant point light source (think of the sun). You may control the direction from which light appears to be coming, the color of the light, and the color of the ambient (nondirectional) light in the environment.

`<dir-vector>` is a vector pointing in the direction of the light source; it does not need to be unit-length. For example, if `<dir-vector>` is `#(0.0 13.0 0.0)`, the light source will be directly 'overhead', ie: light will be 'travelling' in the direction `<0.0, -1.0, 0.0>`.

`<color>` is a vector specifying in `rgb` the color of the light. The `rgb` components are normalized to `[0.0,1.0]`, and specify proportions of primary colors emitted by the light source.

`<ambient-color>` is a vector specifying in `rgb` the color of the ambient light present in the environment.

Ex: (im-set-global-light '#(10.0 80.0 10.0)
 '#(0.9 0.0 0.74)
 '#(0.3 0.3 0.35))
; Makes the light appear to be coming from above and a little to
; the right and behind, assuming one is looking down the negative-z
; axis. The light is a purplish color (90% red, 74% blue, no green).
; The ambient light in the environment is bluish, but not very bright
; (30% red and green, 35% blue).

(im-set-global-light '#(-123.0 0.0 0.0)
 '#(1.0 1.0 1.0)
 '#(0.0 0.0 0.0))
; Makes the light appear to be coming from the left, again assuming
; one is looking down the negative-z axis. The light is fully white.
; There is no ambient light in the environment. This setting will
; give a kind-of `outer-spacey' look to objects; since there's no
; ambient light whatsoever, the `dark side' of an object will reflect
; no light and appear completely black.

im-set-viewer-scale <scale> [Fragile]

Experimental. Effects a viewer scaling of <scale> by scaling the environment by 1/<scale>.

im-set-fog <density> <color> [Fragile]

Experimental. Enables SGI-specific fog effects. <density> is fog density and is in the range [0.0,1.0]. A density of 0.0 disables fog. <color> is a vector specifying in rgb the color of the fog (ie: the color to which rendered objects fade as they move off into the distance).

For fog to work correctly, the void (background) color must be set to <color>, too, because fog only affects objects. Use im-set-void-color to do this.

im-open-struct <...>	[Temporary]
im-close-struct <...>	[Temporary]
im-delete-struct <...>	[Temporary]
im-show-struct <...>	[Temporary]
im-show-structlist <...>	[Temporary]
im-protect-frame <...>	[Temporary]
im-testy <...>	[Temporary]
im-fix-matrix <...>	[Temporary]
im-hack <...>	[Temporary]

All these functions are for debugging purposes only and should not be used.