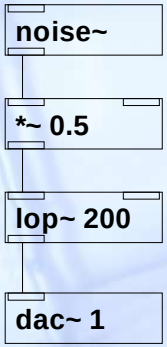
Gem's rendering engine: a mystery unrevealed

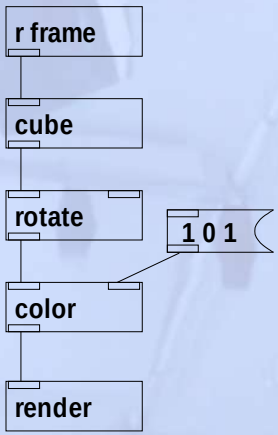
presentation at the pd~convention04

IOhannes m zmölnig

zmoelnig@iem.at

Graphics Environment for Max (historic)





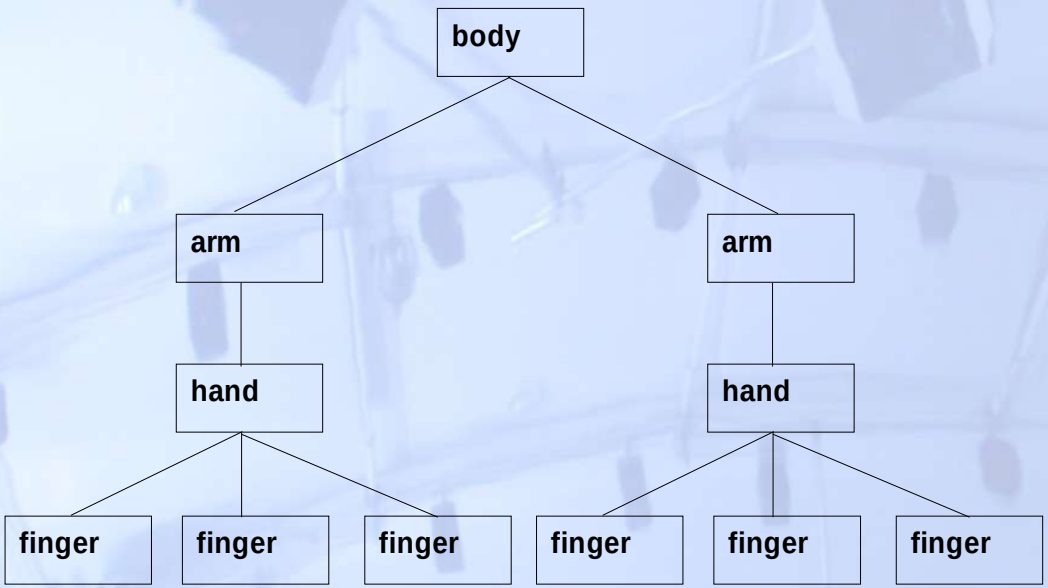
concept of signal~flow

contradictory to the underlying *openGL* state machine

objects add operations to the **gem_list** which is executed by the **[render]**-object

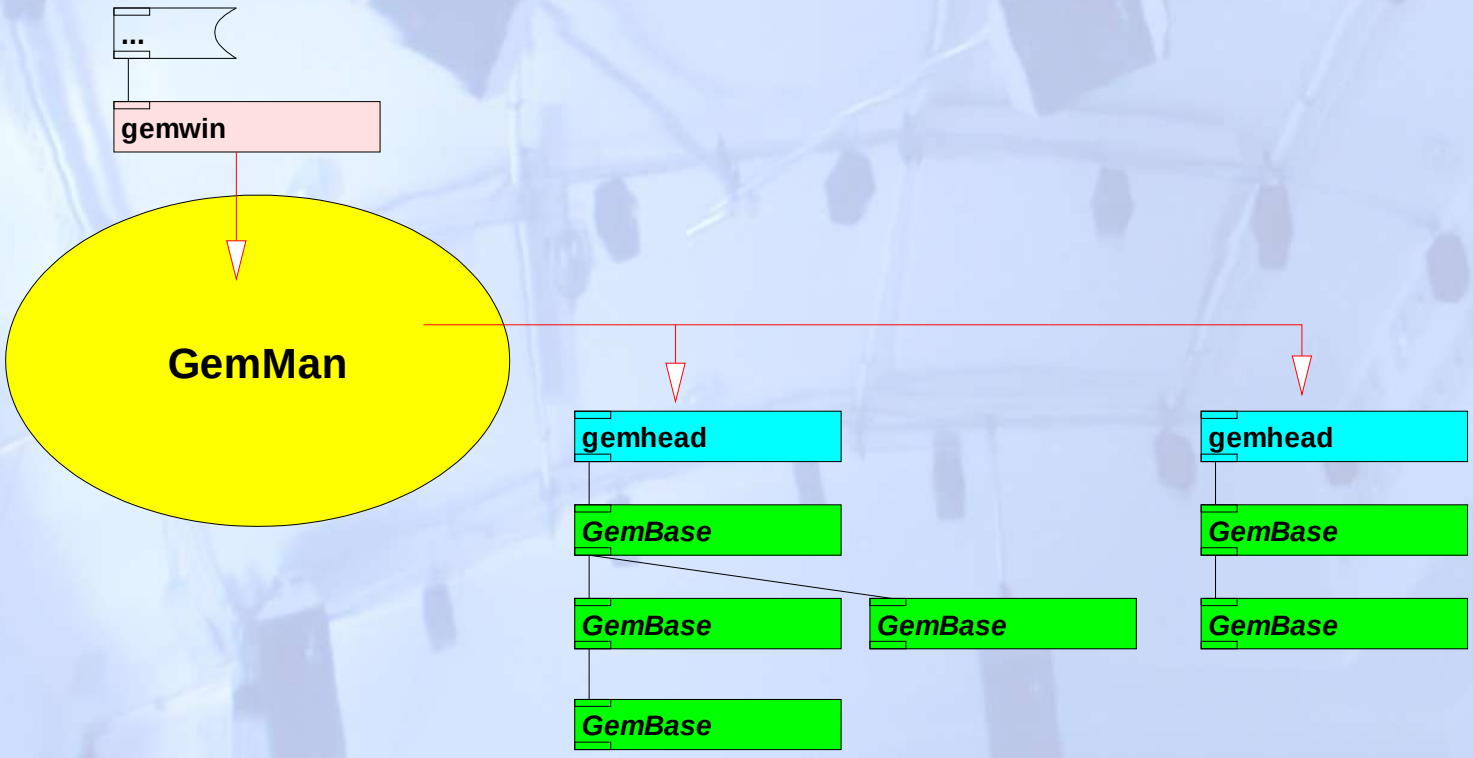
Hierarchic model

Level of Detail: “top-down”



mirrored by openGL's state machine

Structure of Gem



most important files

- Base/GemMan : render management
- Control/gemwin : controlling the GemMan
- Control/gemhead : start of render-chain
- Base/GemBase : base class for gem-objects
- Base/GemState : data passed between gem-objects
- Base/GemCache : shared memory for talkback

GemMan: managing Gem

- static context
- created on loading Gem

- interface to window-manager
  - creates a single window and binds the openGL-context to it
- scheduler (“render-tick”)
- initialization of openGL-state
- execution of render-chains (**[gemhead]**)

[gemwin]

- no *private* data
- interface to control the global *GemMan*

[gemhead]

- start of a render-chain
- registers itself to the *GemMan*
- gets called by *GemMan* each render-cycle
- executes the connected *GemBase*-objects

GemBase

- base-class for each “gem-object”
- provides methods that are executed when rendering
  - startRendering()
  - stopRendering()
  - render(...)
  - postrender(...)

passing data between objects

some gem-objects need no data at all

(only the *openGL*-state is modified)

other objects need complex data
(images=pixel data+texture information)

GemState

General information about rendering:

- framerate
- lighting model
- ...

Complex data:

- pointer to image-data
- texture-information (texcoords,...)
- ...

GemCache

memory shared between all objects of a render-chain

“talkback”: *GemBase* -> [gemhead]

eg: resend images

rendering a frame

GemMan::render()

- the openGL state is reset: *resetValues()*
  - lighting, fog,...
  - viewpoint,...
- the *GemState* is initialized (general values)
- all positive [gemhead]s are rendered
- the viewpoint is reset
- all negative [gemhead]s are rendered
- buffers are swapped
- the next call of *render()* is scheduled.

calling the render-chains

- each [gemhead] registers itself with *GemMan::addObj()*
- 2 ordered lists
  - *s_linkHead*
  - *s_linkHead_2*
- on rendering, *GemMan* calls the *renderGL()*-function of each node of the linked lists

rendering a chain

gemhead::renderGL(GemState)*

- the current *openGL*-state is pushed to the stack
- the *GemState* is “localized”
  - *dirty*-flag taken from *GemCache*
  - image-data is cleared
- a pd-message is emitted:
  - `gem_state <*GemCache> <*GemState>`
- the *openGL*-state is popped off the stack

render an object

- the `gem_state` message is received
- the *GemCache* is saved to *m_cache*
- the *render(GemState*)*-function is called
  - this might change the *GemState*
- the *continueRender(GemState*)-function is called*
  - this emits a `gem_state` message
  - this calls all subsequent gem-objects
- call the *postrender(GemState*)-function*

window-handling

- Window-management is done is *GemMan*
- only one window can be handled
- only one output-device can be handled
- separate rendering passes into different buffers hardly possible (e.g. selection buffer)
  - 1 object != 1 context

Proposal: multiple windows

- remove all window-handling code from *GemMan*
- *GemMan* should handle render-chains
- scheduler could stay within *GemMan*
- stay compatible with older version of Gem

[gemcontrol]

- new object for controlling the *GemMan*
- controlling the scheduler
  - turn rendering on/off
  - change framerate
- distribute “render”-command to various output modules.
- (get information from *GemMan*)

GemOutput

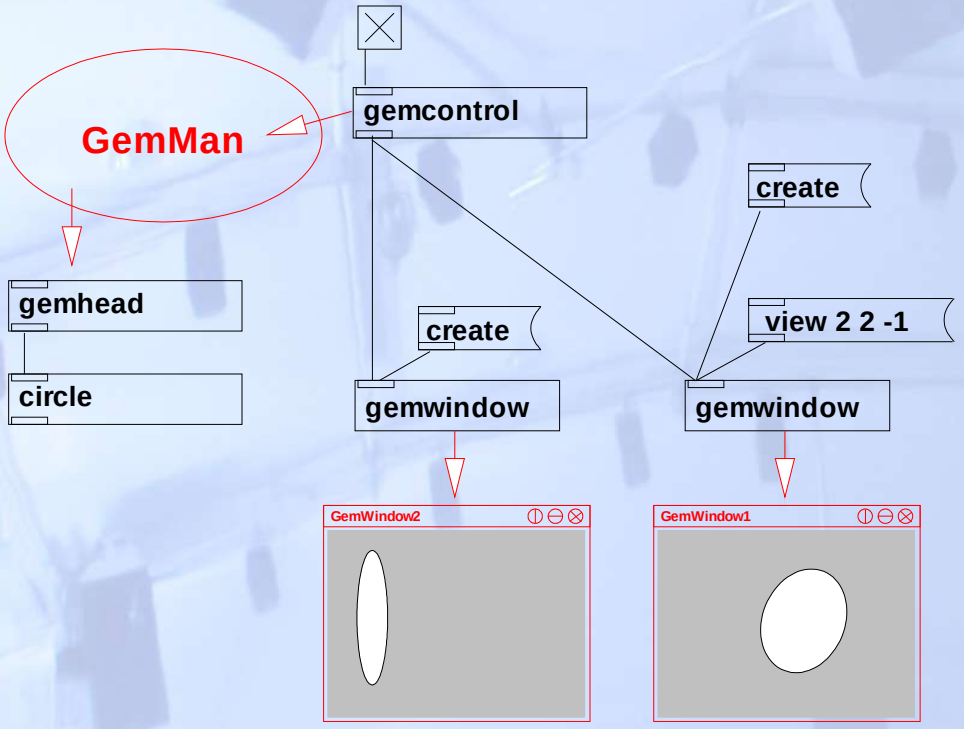
- base class for “render targets”
- camera, background,...-settings are kept locally
- provides virtual methods for
  - makeCurrent() : make the local openGL-context the current one
  - doRender() : call *GemMan::render()*
  - postRender() : swap buffers

GemOutput (cont.)

- output-module might render to
  - window, terminal, video-out
  - file (movie), ieee1394
  - buffers (pbuffer, selection buffer, ...)
- display-list sharing
  - the render-tree ([gemhead]s) are executed only once in a render-cycle
  - the render-tree is compiled into a display-list
  - the output-modules execute the display-list

[gemwindow]

- implements *GemOutput*
- renders to a “normal” window



Step by step

- each [gemcontrol] registers itself with *GemMan::addCtrl()*
- Each render-cycle, the *gemcontrol::render()* methods are called by *GemMan*
- the called [gemcontrol] emits a pd-message `gem_render`
- *GemOutput::makeCurrent()*-method
- *GemOutput::resetState() resets* the openGL-state
- *GemOutput::doRender(GemState)* calls *GemMan::render1(GemState)* (do the rendering)
- *GemOutput::postrender()* swaps the buffers

Compatibility

[gemcontrol] does not reject unknown commands but passes them on!

Conclusio

Thanks for your patience!