Pixel Art Library

Tomi Neste tneste@common-lisp.net

October 18, 2007

Contents

1 Introduction and installation

1.1 What is Pixel Art Library

PAL is a Common Lisp library for developing applications with fast 2d graphics and sound. Internally it uses SDL for sound, event handling and window initialisation and OpenGL for fast hardware accelerated graphics but its API has little to do with the aforementioned libraries.

PAL's design goals are ease of use, portability and reliability. It tries to provide all the *common* functionality that is needed when creating 2d games and similar applications. As such it neither provides higher level specialised facilities like sprites or collision detection, or lower level OpenGL specific functionality. If the user is familiar with Common Lisp and OpenGL this kind of functionality should be easy to implement on top of PAL.

1.2 Requirements

- Pixel Art Library requires the SDL, SDL_image and SDL_mixer libraries. For Windows users it's easiest to use the ones included in the PAL releases, Linux users should be able to easily install these through their distros package management. *Note: These come with their own license.*

- Like most modern CL libraries PAL uses ASDF to handle compilation and loading. If you are using SBCL this is included with the default installation and can be loaded with (REQUIRE :ASDF), with other systems you may need to download it separately.

- For interfacing with the foreign libraries PAL uses the excellent CFFI library. It's available from http://common-lisp.net/project/cffi

- For creating the bitmap fonts that PAL uses you need the font creator that is included in Haaf's Game Engine. This will be fixed in the future releases.

- To get anywhere near reasonable performance you need a graphics card and driver that is capable of hardware accelerated OpenGL graphics.

1.3 Installation

After installing CFFI (and possibly ASDF) and downloading and unpacking PAL you should

- Under Windows copy the .dlls to somewhere where they can be found, for example in your Lisp implementations home folder.

- Under Linux, check that the SDL, SDL_mixer and SDL_image packages are installed.

- Copy the PAL folder to where you usually keep your ASDF systems. If you are unsure you can check and modify this through ASDF:*CENTRAL-REGISTRY* variable
- In your Lisp prompt do (ASDF:OOS 'ASDF:LOAD-OP :PAL) and after awhile everything should be compiled and loaded in your Lisp session. In case of errors first check that everything, including the foreign libraries can be found by the system. If nothing works feel free to bug the Pal-dev mailing list.
- If everything went fine you can now try your first PAL program, enter in the following:

```
   (with-pal (:title “PAL test”)
     (clear-screen 255 255 0)
     (with-transformation (:pos (v 400 300) :angle 45f0 :scale
4f0)
       (draw-text “Hello World!”  (v 0 0))
       (wait-keypress)))
```

2 Opening and closing PAL and handling resources

2.1 Introduction

2.2 Functions

OPEN-PAL (&key *width height fps title fullscreenp paths*)

Opens and initialises PAL window.

width, width of the screen.

height, height of the screen. If width and height are 0 then the default desktop dimensions are used.

fps, maximum number of times per second that the screen is updated.

title, title of the screen.

fullscreenp, open in windowed or fullscreen mode.

paths, pathname or list of pathnames that the load-* functions use to find resources. Initially holds *default-pathname-defauls* and PAL installation directory.

CLOSE-PAL ()

Closes PAL screen and frees all loaded resources.

WITH-PAL (&key *width height fps title fullscreenp paths* &body *body*)

Opens PAL, executes *body* and finally closes PAL. Arguments are same as with OPEN-PAL.

FREE-RESOURCE (*resource*)

Frees the *resource* (image, font, sample or music).

FREE-ALL-RESOURCES ()

Frees all allocated resources.

WITH-RESOURCE (*var init-form*) &body *body*

Binds *var* to the result of *init-form* and executes *body.* Finally calls FREE-RESOURCE on *var.*

GET-SCREEN-WIDTH () => *number*

GET-SCREEN-HEIGHT () => *number*

Returns the dimensions of PAL screen.

3 Event handling

3.1 Introduction

There are two ways to handle events in PAL; the callback based HANDLE-EVENTS or EVENT-LOOP that call given functions when an event happens, or directly polling for key and mouse state with TEST-KEYS, KEY-PRESSED-P and GET-MOUSE-POS.

NOTE: Even if you don't need to use the callback approach it is still necessary to call HANDLE-EVENTS on regular intervals, especially on Windows. Running an EVENT-LOOP does this automatically for you and is the preferred way to handle events.

3.2 Functions

HANDLE-EVENTS (&key *key-up-fn key-down-fn mouse-motion-fn quit-fn*)

Get next event, if any, and call appropriate handler function.

***key-up-fn*,** called with the released key-sym. For key-syms see chapter 3.3

***key-down-fn*,** called with the pressed key-sym. When *key-down-fn* is not defined pressing Esc-key causes a quit event.

***mouse-motion-fn*,** called with x and y mouse coordinates.

***quit-fn*,** called without any arguments when user presses the windows close button. Also called when Esc key is pressed, unless *key-down-fn* is defined.

UPDATE-SCREEN ()

Updates the PAL screen. No output is visible until UPDATE-SCREEN is called.

EVENT-LOOP ((&key *key-up-fn key-down-fn mouse-motion-fn quit-fn*) &body *body*)

Repeatedly calls *body* between HANDLE-EVENT and UPDATE-SCREEN. Arguments are the same as with HANDLE-EVENTS. Returns when (return-from event-loop) is called, or, if quit-fn is not given when quit event is generated.

GET-MOUSE-POS () => *vector*

GET-MOUSE-X () => *number*

GET-MOUSE-Y () => *number*

Returns the current position of mouse pointer.

SET-MOUSE-POS (*vector*)

Sets the position of mouse pointer.

KEY-PRESSED-P (*keysym*) => *bool*

Test if the key *keysym* is currently pressed down. For keysyms see chapter 3.3

TEST-KEYS ((*key* | (*keys*) *form*))

Tests if any of the given keys are currently pressed. Evaluates *all* matching forms.

Example:

```
(test-keys
  (:key-left (move-left sprite))
  (:key-right (move-right sprite))
  ((:key-ctrl :key-mouse-1) (shoot sprite))
```

KEYSYM-CHAR (*keysym*) => *char*

Returns the corresponding Common Lisp character for *keysym*.

WAIT-KEYPRESS () => *key*

Waits until a key is pressed and released

3.3 Keysyms

These are the symbols used to identify keyboard events. Note that mouse button and scroll wheel events are also represented as keysyms.

```
:key-mouse-1
:key-mouse-2
:key-mouse-3
:key-mouse-4
:key-mouse-5
:key-unknown
:key-first
:key-backspace
:key-tab
:key-clear
:key-return
:key-pause
:key-escape
:key-space
:key-exclaim
:key-quotedbl
:key-hash
:key-dollar
```

:key-ampersand
:key-quote
:key-leftparen
:key-rightparen
:key-asterisk
:key-plus
:key-comma
:key-minus
:key-period
:key-slash
:key-0
:key-1
:key-2
:key-3
:key-4
:key-5
:key-6
:key-7
:key-8
:key-9
:key-colon
:key-semicolon
:key-less
:key-equals
:key-greater
:key-question
:key-at
:key-leftbracket
:key-backslash
:key-rightbracket
:key-caret
:key-underscore
:key-backquote
:key-a
:key-b
:key-c
:key-d
:key-e
:key-f
:key-g
:key-h
:key-i
:key-j
:key-k
:key-l
:key-m

:key-n
:key-o
:key-p
:key-q
:key-r
:key-s
:key-t
:key-u
:key-v
:key-w
:key-x
:key-y
:key-z
:key-delete
:key-world_0
:key-world_1
:key-world_2
:key-world_3
:key-world_4
:key-world_5
:key-world_6
:key-world_7
:key-world_8
:key-world_9
:key-world_10
:key-world_11
:key-world_12
:key-world_13
:key-world_14
:key-world_15
:key-world_16
:key-world_17
:key-world_18
:key-world_19
:key-world_20
:key-world_21
:key-world_22
:key-world_23
:key-world_24
:key-world_25
:key-world_26
:key-world_27
:key-world_28
:key-world_29
:key-world_30
:key-world_31

:key-world_32
:key-world_33
:key-world_34
:key-world_35
:key-world_36
:key-world_37
:key-world_38
:key-world_39
:key-world_40
:key-world_41
:key-world_42
:key-world_43
:key-world_44
:key-world_45
:key-world_46
:key-world_47
:key-world_48
:key-world_49
:key-world_50
:key-world_51
:key-world_52
:key-world_53
:key-world_54
:key-world_55
:key-world_56
:key-world_57
:key-world_58
:key-world_59
:key-world_60
:key-world_61
:key-world_62
:key-world_63
:key-world_64
:key-world_65
:key-world_66
:key-world_67
:key-world_68
:key-world_69
:key-world_70
:key-world_71
:key-world_72
:key-world_73
:key-world_74
:key-world_75
:key-world_76
:key-world_77

:key-world_78
:key-world_79
:key-world_80
:key-world_81
:key-world_82
:key-world_83
:key-world_84
:key-world_85
:key-world_86
:key-world_87
:key-world_88
:key-world_89
:key-world_90
:key-world_91
:key-world_92
:key-world_93
:key-world_94
:key-world_95
:key-kp0
:key-kp1
:key-kp2
:key-kp3
:key-kp4
:key-kp5
:key-kp6
:key-kp7
:key-kp8
:key-kp9
:key-kp_period
:key-kp_divide
:key-kp_multiply
:key-kp_minus
:key-kp_plus
:key-kp_enter
:key-kp_equals
:key-up
:key-down
:key-right
:key-left
:key-insert
:key-home
:key-end
:key-pageup
:key-pagedown
:key-f1
:key-f2

:key-f3
:key-f4
:key-f5
:key-f6
:key-f7
:key-f8
:key-f9
:key-f10
:key-f11
:key-f12
:key-f13
:key-f14
:key-f15
:key-numlock
:key-capslock
:key-scrollock
:key-rshift
:key-lshift
:key-rctrl
:key-lctrl
:key-ralt
:key-lalt
:key-rmeta
:key-lmeta
:key-lsuper
:key-rsuper
:key-mode
:key-compose
:key-help
:key-print
:key-sysreq
:key-break
:key-menu
:key-power
:key-euro
:key-undo
:key-last

4 Images and drawing

4.1 CLEAR-SCREEN

4.2 DRAW-POINT

4.3 DRAW-LINE

4.4 DRAW-ARROW

4.5 LOAD-IMAGE

4.6 IMAGE-WIDTH, IMAGE-HEIGHT

4.7 DRAW-IMAGE

4.8 DRAW-IMAGE*

4.9 DRAW-RECTANGLE

4.10 DRAW-CIRCLE

4.11 DRAW-POLYGON

4.12 DRAW-POLYGON*

4.13 SET-CURSOR

4.14 IMAGE-FROM-ARRAY

4.15 IMAGE-FROM-FN

4.16 LOAD-IMAGE-TO-ARRAY

4.17 SCREEN-TO-ARRAY

5 Handling graphics state

5.1 ROTATE

5.2 TRANSLATE

5.3 SCALE

5.4 WITH-TRANSFORMATION

5.5 SET-BLEND-MODE

5.6 RESET-BLEND-MODE

5.7 SET-BLEND-COLOR

5.8 WITH-BLEND

5.9 WITH-CLIPPING

6 Music and samples

6.1 LOAD-SAMPLE, SAMPLE-P

6.2 PLAY-SAMPLE

6.3 SET-SAMPLE-VOLUME

6.4 LOAD-MUSIC

6.5 SET-MUSIC-VOLUME

6.6 PLAY-MUSIC

6.7 HALT-MUSIC

7 Fonts

7.1 LOAD-FONT

7.2 GET-TEXT-SIZE

7.3 GET-FONT-HEIGHT

7.4 DRAW-TEXT

7.5 DRAW-FPS

7.6 MESSAGE

8 Tags

8.1 DEFINE-TAGS

8.2 ADD-TAG

8.3 TAG

9 Vector and math operations

9.1 V, COPY-VEC, V=

9.2 ANGLE-V, V-ANGLE, V-ROTATE

9.3 VX, VY

9.4 V+, V-, V*, V/, V+!, V-!, V*!, V/!

9.5 V-MAX, V-MIN

9.6 V-TRUNCATE

9.7 V-DIRECTION

9.8 CLOSEST-POINT-TO-LINE

9.9 DISTANCE-FROM-LINE

9.10 POINT-IN-LINE-P

9.11 LINES-INTERSECTION

9.12 CIRCLE-LINE-INTERSECTION

9.13 POINT-INSIDE-RECTANGLE-P, POINT-INSIDE-CIRCLE-P

9.14 CIRCLES-OVERLAP-P

10 Miscellaneous functions and macros

10.1 GET-GL-INFO

10.2 DATA-PATH

10.3 GET-FPS

10.4 LOAD-FOREIGN-LIBRARIES

10.5 GET-APPLICATION-FOLDER, GET-APPLICATION-FILE

10.6 DATA-PATH

10.7 RANDOMLY

10.8 RANDOM-ELEMENT

10.9 CLAMP

10.10 DO-N

10.11 CURRY