Squirrel n2SDLib Library

Release 3.1 stable

Alberto Demichelis, Florian "Eiyeron" Dormont, Matrefeytontias

April 08, 2016

CONTENTS

CHAPTER

ONE

INTRODUCTION

The n2DLib library consist in a set of functions inmplemented in C. They allow a TI-Nspire developper access the screen and the keybaord and offers functions to allow interacting with them.

The library uses buffering to display smoothly ; that means that all drawing is done to a separate buffer in memory which is then copied to the screen.

Thus, n2DLib offers various functions from drawing a pixel, a sprite or circles to open a bitmap file to parse it and detect which key has been pressed.

1.1 Data structures

All colors, unless specified, must be in R5G6B5 format, meaning a 16-bits number organized this way

```
Bit   | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0
Color | R  | R  | R  | R  | R  | G  | G | G | G | G | G | B | B | B | B | B
```

2DLib uses a variation on the NTI image format to store images. All images must be arrays of unsigned short organized this way

```
Entry |        0       |        1       |         2       |   3-...
Use   | Width in pixels|Height in pixels|Transparent color|Image data
```

n2DLib also offers the Rect structure that has four members : *x*, *y*, *w* and *h*. You're free to use it for various purposes.

1.2 About fixed-point numbers

In addition to graphical commands, n2DLib offers a fast alternative to floating-point numbers that is 24.8 fixed-point numbers. Those numbers are integers whose bits 0-7 are treated like a decimal part, and whose bits 8-24 are treated like an integer part. That means 256 is actually 1.0 when used with the appropriate commands. FIxed-point angles are written in binary angles, meaning a full period is [0, 256].

Since fixed-point numbers are special numbers, they can't interact normally with integers. The following array describes what you can and can't do normally :

	integer	fixed-point
integer	+-*/	•
fixed-point	*/	+-

You need a special routine to : - add an integer to a fixed-point number - divide an integer by a fixed-point number - multiply a fixed-point number by another fixed-point number - divide a fixed-point number by another fixed-point number

See the "Math routines" part of the doc to find how to do all of this.

1.3 Using the key detection functions

n2DLib provides additional key detection functions to those already provided by Ndless, and aims for full compatibility with those latters. That's why n2DLib uses Ndless's t_key struct :

```
typedef struct {
    int row, col, tpad_row, tpad_col;
    tpad_arrow_t tpad_arrow;
} t_key;
```

Here's the structure seend from squirrel's side. Note that the key listing is usable with the prefix *n2dk*. Example : *n2dk.ENTER* to acces *ENTER*'s index.

```
local key =  {
    row : 0, /*number*/
    col : 0, /*number*/
    tpad_row : 0, /*number*/
    tpad_col : 0; /*number*/
    tpad_arrow : 0; /*number*/
};
```

Each key is stored depending on its position on the Nspire's keypad, row and column. You can see two different versions for each variable ; that's because the clickpad keypad and the touchpad keypad have different keymappings. This is taken care of by n2DLib internally, so you don't have to worry about it.

For more info about how the keypad(s) work(s), see http://hackspire.unsads.com/wiki/index.php/Keypads .

See the "Key detection functions" part of the doc to see what you can do. You don't necessarily need these routines unless you want to do specific things like custom keybinding, since what Ndless provides is enough for most cases .

CHAPTER

TWO

N2DLIB FUNCTIONS

2.1 Graphics

clearBufferB()
Fills the buffer with black color.

clearBufferW()
Fills the buffer with White color.

clearBuffer(*color*)
Fills the buffer with White color.

getPixel(*image*, *x*, *y*)
Returns the color of a given pixel in an image, or the transparent color of this image if the coordinates given are out-of-bounds.

deinitBuffering()
Frees memory used by the buffering functionalities. Call this as the very last instruction of your program (excepting return).

drawLine(*x1*, *y1*, *x2*, *y2*, *color*)
Draws a line between two points of the given color.

drawLine(*sprite*, *x*, *y*, *flash*, *flashColor*)
Draws an image to the given coordinates.

drawSpritePart(*sprite*, *x*, *y*, *xp*, *yp*, *wp*, *hp*, *flash*, *flashColor*)
Draws part of an image to the given coordinates. The routine draws what of the sprite is in the rectangle given by *xp*, *yp*, *wp* and *hp*.

drawSpriteRotated(*sprite*, *xs*, *ys*, *ws*, *hs*, *xr*, *yr*, *angle*, *flash*, *flashColor*)
Draws a sprite rotated by a given angle at the coordinates given by *xs*, *ys*, *ws* and *hs* so that the rotation of the sprite is performed around the point (*xr*, *yr*), which is relative to the sprite itself. The center of the rotation will always be displayed at the coordinates (*xs*, *ys*). For example, if *xr* and *yr* are half the sprite's width and height, the sprite will be rotated around its center.

drawSpriteScaled(*sprite*, *xs*, *ys*, *ws*, *hs*, *flash*, *flashColor*)
Draws a sprite by scaling it so that it fits perfectly in the rectangle given by *xs*, *ys*, *ws* and *hs*.

fillCircle(*x*, *y*, *radius*, *color*)
Fills a circle of the given color.

fillEllipse(*x*, *y*, *w*, *h*, *color*)
Fills an ellipse of the given size with the given color.

fillRect(*x*, *y*, *w*, *h*, *color*)
Fills a rectangle of the given dimensions with the given color ; does clipping.

initBuffering()
Initializes the buffering functionalities. Call this as the very first instruction of your program if you want to use n2DLib's buffering.

setPixel(*x*, *y*, *color*)
Sets a pixel to the given color after verifying the pixel is actually in the screen's dimensions.

setPixelRGB(*x*, *y*, *r*, *g*, *b*)
Sets a pixel to the given color after verifying the pixel is actually in the screen's dimensions and using three color components.

setPixelUnsafe(*x*, *y*, *color*)
Sets a pixel to the given color, but does not make sure the pixel is in the screen's dimensions. Faster than setPixel, but use only if you know you can't draw out of the screen.

updateScreen()
Copies the content of the buffer to the screen. This does not clear the buffer.

2.2 Text routines

drawChar(*x*, *y*, *margin*, *char*, *fontColor*, *outlineColor*)
Draws a single character at the given position with the given front and outline color using n2DLib's built-in font. Does clipping and supports newlines with n. When n is passed as ch, the function resets the X value to the passed margin value and Y goes to newline. X and Y are modified like a cursor position would and are returned in an array.

drawDecimal(*x*, *y*, *number*, *fontColor*, *outlineColor*)
Draws a signed integer at the given position with the given front and outline color using n2DLib's built-in font. Does clipping. X and Y are modified like a cursor position would. Use this as a fast way to display integers only.

drawString(*x*, *y*, *margin*, *str*, *fontColor*, *outlineColor*)
Draws a string of characters at the given position with the given front and outline color using n2DLib's built-in font. Does clipping and supports newlines with n. When n is encountered in the string, the function resets the X value to the passed margin value and Y goes to newline. X and Y are modified like a cursor position would. Use this as a fast way to display strings only.

stringWidth(*str*)
Returns the width of the string in pixels when using n2DLib's built-in font.

numberWitdh(*number*)
Returns the width of the decimal in pixels when drawn to the screen using n2DLib's built-in font.

2.3 Math routines

fixcos(*angle*)
Returns the cosinus of a binary angle in fixed-point format.

fixdiv(*a*, *b*)
Performs a division between two fixed-point numbers.

fixmul(*a*, *b*)
Performs a multiplication between two fixed-point numbers.

fixsin(*angle*)
Returns the sinus of a binary angle in fixed-point format.

fixtoi (*f*)
Turns a fixed-point number into an integer.

itofix (*i*)
Turns an integer into a fixed-point number.

2.4 Key detection routines

getKeyPressed (*key*)
Detects if any key is being pressed ; if so, returns the corresponding data. If no key is being pressed, returns the structure filled with _KEY_DUMMY and returns it.

NOTE This doesn't detect the touchpad's arrow keys.

_n2d_isKey (*key*, *key_index*)
Returns *true* if given key match the enumeration index, or *false* if not.

USAGE

```
value = n2d.isKey(key, n2dk.ENTER)
```

NOTE This is very useful for example when comparing a key that has been filled with get_key_pressed() with an Ndless *n2dk.* constant.

2.5 Image loading routines

_n2d_loadBMP (*path*, *colorMask*)
Returns a blob containing the loaded sprite with given alpha color mask if the file correctly loaded or null if it coudln't load it.

NOTE This function load only 24-bit (R8G8B8) bitmap files. Make sure that you chose the correct format.

WARNING! Current version of this function doesn't correctly load pictures with width not multiple of 16 (or 8). This has been issued to the library developper.

Symbols

C

D

F

G

I

N

S

U