SDL$_{pcf}$

Release 0.0.1

Dec 31, 2021

Getting Started:

With **SDL_pcf** you can know easily use pixel-perfect bitmap fonts with SDL2:

- Use thousands of existing X11 bitmap fonts, including terminus
- Direct-to-surface writing (no temp surface needed)
- Hardware-accelerated rendering supported with SDL2 Renderer API or SDL_gpu

Using PCF fonts is as simple as:

```
PCF_Font *font;
font = PCF_OpenFont("ter-x24n.pcf.gz");
PCF_FontWrite(font, "Hello world !", white, screenSurface, &location);
```

CHAPTER 1

Building SDL_pcf

SDL_pcf depends on SDL2 and zlib. It has optional support for SDL_gpu

1.1 Unix (Autotools)

```
$ ./configure --prefix=/usr
$ make
$ make check                # build demos in ./test (optional)
$ [sudo] make install       # install to system (optional)
```

make will build cglm to **src/.libs** sub folder in project folder. If you don't want to install **cglm** to your system's folder you can get static and dynamic libs in this folder. Headers (*.h) will be found in the src folder.

1.2 Documentation (Sphinx):

SDL_pcf documentation is based on sphinx.

To build html documentation do the following:

```
$ cd docs
$ make html
```

CHAPTER 2

Using SDL_pcf

SDL_pcf works with bitmap fonts distributed as PCF files. Those fonts originates from Unix X11 and are distributed as a collection of files each file being a representation of characters in the font in a specific size/variant, variant being italic and bold.

For example, the file that comes with SDL_pcf tests, **ter-x24n.pcf.gz** is the Terminus font, "normal" variant, 24 points. The "bold" variant would be **ter-x24b.pcf.gz**. Chances are high that you already have a couple of fonts installed on your system.

Find them out with:

```
$ find /usr/share/fonts/ -name "*.pcf" -or -name "*.pcf.gz"
```

SDL_pcf can be used in two ways:

- Direct writing of any character of the font to an SDL_Surface or using a SDL_Renderer.
- Pre-render a set of characters to a surface or a texture and then blit those pre-rendererd characters to their destination. This can be hardware accelerated (SDL_Renderer or SDL_gpu).

2.1 Direct writing

This mode is the simplest and the more flexible, all characters from the font can be drawn. However it requires pixel-level access as SDL_pcf will use the data from the PCF font to light the appropriate pixels.

It can be used either with SDL_Surfaces or go through the SDL_Renderer API.

A typical use case would be like:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <SDL.h>
4
5  #include "SDL_pcf.h"
6
```

(continues on next page)

(continued from previous page)

```c
#define SCREEN_WIDTH 640
#define SCREEN_HEIGHT 480

int main(int argc, char *argv[])
{
    SDL_Window* window = NULL;
    SDL_Surface* screenSurface = NULL;
    Uint32 black, white;
    PCF_Font *font;

    if (SDL_Init(SDL_INIT_VIDEO) < 0) {
        fprintf(stderr, "could not initialize sdl2: %s\n", SDL_GetError());
        return 1;
    }

    window = SDL_CreateWindow(
                "SDL_pcf test drive",
                SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED,
                SCREEN_WIDTH, SCREEN_HEIGHT,
                SDL_WINDOW_SHOWN
                );
    if (window == NULL) {
        fprintf(stderr, "could not create window: %s\n", SDL_GetError());
        return 1;
    }

    screenSurface = SDL_GetWindowSurface(window);
    if(!screenSurface){
        printf("Error: %s\n",SDL_GetError());
        exit(-1);
    }

    white = SDL_MapRGB(screenSurface->format, 0xFF, 0xFF, 0xFF);
    black  = SDL_MapRGB(screenSurface->format, 0x00, 0x00, 0x00);
    SDL_FillRect(screenSurface, NULL, black);

    font = PCF_OpenFont("ter-x24n.pcf.gz");
    if(!font){
        printf("%s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }

    PCF_FontWrite(font, "Hello, World!", white, screenSurface, NULL);
    SDL_UpdateWindowSurface(window);

    SDL_Delay(4000);

    PCF_CloseFont(font);

    SDL_DestroyWindow(window);
    SDL_Quit();

    exit(EXIT_SUCCESS);
}
```

Build it with:

```
$ gcc simple-test.c -o simple-test `pkg-config sdl2 SDL2_pcf --libs --cflags`
```

The relevant part is highlighted. The call to *PCF_FontWrite()* will write the **"Hello, World!"** string at 0,0 on **screenSurface**.

You can see a more complex example of this in **test/ayba.c**.

2.2 Static fonts

Static fonts are a set of pre-rendered characters built from a font. This allows to avoid the need of direct pixel access to the destination and allows faster blitting and hardware acceleration.

Static fonts are built from a loaded font and a set of characters. They are thus limited to that set.

Note: Static fonts can be compiled to use either SDL_Renderer(default) **or** SDL_gpu. This is defined at compile time and can be check using *PCF_TEXTURE_TYPE* which will be either equal to *PCF_TEXTURE_SDL2* or *PCF_TEXTURE_GPU*.

SDL_pcf provide support functions to locate the character to copy but the actual blitting must be done by the client code using the StaticFont struct **texture** member which will be of **SDL_Texture**(default) or ***GPU_Image** type, depending of build-time configuration (see note below).

You can see a good example of how all of this works together in: - **test/ayba-sf.c** for the SDL_Renderer API - **test/ayba-sf-gpu.c** for usage with SDL_gpu

CHAPTER 3

API documentation

3.1 Direct Writing

3.1.1 Functions

3.1.2 Functions documentation

PCF_Font ***PCF_OpenFont** (const char **filename*)

Opens a PCF font file. Supports both .pcf and .pcf.gz.

Parameters: **filename** The file to open

Returns: a PCF_Font opaque struct representing the font. The caller must call PCF_CloseFont when done using the font.

void **PCF_CloseFont** (PCF_Font **self*)

Free resources taken up by a loaded font. Caller code must always call PCF_CloseFont on all fonts it allocates. Each PCF_OpenFont must be paired with a matching PCF_CloseFont.

Parameters: **self** The font to free.

bool **PCF_FontWriteChar**(PCF_Font **font*, int *c*, Uint32 *color*, SDL_Surface **destination*, SDL_Rect **location*)

Writes a character on screen, and advance the location by one char width. If the surface is too small to fit the char or if the glyph is partly out of the surface (start writing a 18 pixel wide char 2 pixels before the edge) only the pixels that can be written will be drawn, resulting in a partly drawn glyph and the function will return false.

Parameters:

c The ASCII code of the char to write. You can of course use 'a' instead of 97.

font The font to use to write the char. Opened by PCF_OpenFont.

color The color of text. Must be in @param destination format (use SDL_MapRGB/SDL_MapRGBA to build a suitable value).

destination The surface to write to.

location Where to write on the surface. Can be NULL to write at 0,0. If not NULL, location will be advanced by the width.

Returns: True on success(the whole char has been written), false on error/partial draw. Details of the failure can be retreived with SDL_GetError().

bool **PCF_FontWrite**(PCF_Font **font*, const char **str*, Uint32 *color*, SDL_Surface **destination*, SDL_Rect **location*)

Writes a string on screen. This function will try it's best to write as many chars as possible: If the surface is not wide enough to accomodate the whole string, it will stop at the very last pixel (and return false). This function doesn't wrap lines. Use PCF_FontGetSizeRequest to get needed space for a given string/font.

Parameters:

str The string to write.

font The font to use. Opened by PCF_OpenFont.

color The color of text. Must be in @param destination format (use SDL_MapRGB/SDL_MapRGBA to build a suitable value).

destination The surface to write to.

location Where to write on the surface. Can be NULL to write at 0,0. If not NULL, location will be advanced by the width of the string.

Returns: True on success(the whole string has been written), false on error/partial draw. Details of the failure can be retreived with SDL_GetError().

bool **PCF_FontRenderChar**(PCF_Font **font*, int *c*, SDL_Renderer **renderer*, SDL_Rect **location*)

Writes a character on a SDL_Renderer, and advance the given location by one char width. If the renderer is too small to fit the char or if the glyph is partly out of the surface (start writing a 18 pixel wide char 2 pixels before the edge) only the pixels that can be written will be drawn, resulting in a partly drawn glyph and the function will return false.

Note that there is no color parameter: This is controlled at the SDL_Renderer level with SDL_SetRenderDrawColor.

Parameters: c The ASCII code of the char to write. You can of course use 'a' instead of 97. font The font to use to write the char. Opened by PCF_OpenFont. renderer The renderer that will be used to draw. location Location within the renderer. Can be NULL to write at 0,0. If not NULL, location will be advanced by the width.

Returns: True on success(the whole char has been written), false on error/partial draw. Details of the failure can be retreived with SDL_GetError().

bool **PCF_FontRender**(PCF_Font **font*, const char **str*, SDL_Color **color*, SDL_Renderer **renderer*, SDL_Rect **location*)

Writes a string on renderer. This function will try it's best to write as many chars as possible: If the renderer

is not wide enough to accomodate the whole string, it will stop at the very last pixel (and return false). This function doesn't wrap lines. Use PCF_FontGetSizeRequest to get needed space for a given string/font.

Parameters: str The string to write. font The font to use. Opened by PCF_OpenFont. color The color of text. If not NULL, it will overrede the current renderer's color. If NULL, the current renderer's color will be used. renderer The rendering context to use. location Where to write on the renderer. Can be NULL to write at 0,0. If not NULL, location will be advanced by the width of the string.

Returns: True on success(the whole string has been written), false on error/partial draw. Details of the failure can be retreived with SDL_GetError().

void **PCF_FontGetSizeRequest**(PCF_Font **font*, const char **str*, Uint32 **w*, Uint32 **h*)

Computes space (pixels width*height) needed to draw a string using a given font. Both @param w and @param h can be NULL depending on which metric you are interested in. The function won't fail if both are NULL, it'll just be useless.

Parameters: str The string whose size you need to know. font The font you want to use to write that string w Pointer to somewhere to place the resulting width. Can be NULL. h Pointer to somewhere to place the resulting height. Can be NULL.

void **PCF_FontGetSizeRequestRect**(PCF_Font **font*, const char **str*, SDL_Rect **rect*)

Same PCF_FontGetSizeRequest as but fills an SDL_Rect. Rect x and y get initialized to 0.

Parameters: str The string whose size you need to know. font The font you want to use to write that string rect Pointer to an existing SDL_Rect (cannot be NULL) to fill with the size request.

3.2 Static fonts

3.2.1 Macros

1. *PCF_TEXTURE_TYPE*
2. *PCF_TEXTURE_SDL2*
3. *PCF_TEXTURE_GPU*
4. *PCF_LOWER_CASE*
5. *PCF_UPPER_CASE*
6. *PCF_ALPHA*
7. *PCF_DIGITS*

3.2.2 Functions

1. *PCF_FontCreateStaticFont()*
2. *PCF_FontCreateStaticFontVA()*
3. *PCF_FreeStaticFont()*
4. *PCF_StaticFontGetCharRect()*
5. *PCF_StaticFontGetSizeRequest()*
6. *PCF_StaticFontGetSizeRequestRect()*
7. *PCF_StaticFontCanWrite()*
8. *PCF_StaticFontCreateTexture()*

3.2.3 Structure documentation

PCF_StaticFont
The structure has the following public members:

```
typedef struct{
    SDL_Surface *raster;
    xCharInfo   metrics;
    SDL_Texture|GPU_Image  *texture;
}PCF_StaticFont;
```

PCF_StaticFont **raster**
The pre-rendered characters for that font, in a raster. Usable for any software blitting operation.

PCF_StaticFont **texture**
The pre-rendered characters for that font in a GPU-friendly texture. Be sure to call *PCF_StaticFontCreateTexture()* before using it.

PCF_StaticFont **metrics**
See somewhere else

3.2.4 Macros documentation

PCF_TEXTURE_TYPE
Defined at build time to either *PCF_TEXTURE_SDL2* (default) or *PCF_TEXTURE_GPU*

PCF_TEXTURE_SDL2
SDL_pcf is built to support SDL2 textures. *PCF_StaticFont* **texture** member is of SDL_Texture* type

PCF_TEXTURE_GPU
SDL_pcf is built to support SDL_gpu textures. *PCF_StaticFont* **texture** member is of GPU_Image* type

PCF_LOWER_CASE
Pre-defined character set for use with *PCF_FontCreateStaticFont()*

```
#define PCF_LOWER_CASE "abcdefghijklmnopqrstuvwxyz"
```

PCF_UPPER_CASE
Pre-defined character set for use with *PCF_FontCreateStaticFont()*

```
#define PCF_UPPER_CASE "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

PCF_ALPHA
Pre-defined character set for use with *PCF_FontCreateStaticFont()*

```
#define PCF_ALPHA "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

PCF_DIGITS
Pre-defined character set for use with *PCF_FontCreateStaticFont()*

```
#define PCF_DIGITS "0123456789"
```

3.2.5 Functions documentation

PCF_StaticFont ***PCF_FontCreateStaticFont** (PCF_Font **font*, SDL_Color **color*, int *nsets*, ...)
Creates and return a pre-drawn set of characters. The font can be closed afterwards. The return value must be

freed by the caller using *PCF_FreeStaticFont()*.

Once drawn, static fonts are immutable: You can't add characters on the fly, or change colors. You'll need to create a new static font to do that. The purpose of PCF_StaticFont is to integrate with rendering systems based on fixed bitmap data + coordinates, like SDL_Renderer or OpenGL.

Parameters:

font The font to draw with

color The color of the pre-rendered glyphs

nsets The number of glyph sets that follows

... Sets of glyphs to include in the cache, as const char*. You can use pre-defined sets such as *PCF_ALPHA*, *PCF_DIGITS*, etc. The function will filter out duplicated characters.

Returns: a newly allocated PCF_StaticFont or NULL on error. The error will be available with SDL_GetError()

PCF_StaticFont ***PCF_FontCreateStaticFontVA**(PCF_Font **font*, SDL_Color **color*, int *nsets*, size_t *tlen*, va_list *ap*)

va_list version of PCF_FontCreateStaticFont. The only difference is that this function needs to be provided with the total(cumulative) length of all the strings that it gets through ap. This is due to the fact that va_list can't be rewinded when passed as an argument to a non-variadic function

Parameters:

font See *PCF_FontCreateStaticFont()* font

color See *PCF_FontCreateStaticFont()* color

nsets See *PCF_FontCreateStaticFont()* nsets

tlen Total (cumulative) len of the strings passed in.

ap List of nsets char*

Returns: See *PCF_FontCreateStaticFont()*.

void **PCF_FreeStaticFont**(*PCF_StaticFont *self*)

Frees memory used by a static font. Each static font created using PCF_FontCreateStaticFont should be released using this function.

Parameters: self The PCF_StaticFont to free.

int **PCF_StaticFontGetCharRect**(*PCF_StaticFont *font*, int *c*, SDL_Rect **glyph*)

Find the area in self->raster holding a glyph for c. The area is suitable for a SDL_BlitSurface or a SDL_RenderCopy operation using self->raster as a source

Parameters:

font The static font to search in.

c The char to search for.

glyph Location where to put the coordinates, when found.

Returns: 0 for whitespace (**glpyh** untouched), non-zero if **font** has something printable for **c**: 1 if the char as been found, -1 otherwise. When returning -1, **glpyh** has been set to the default glyph.

void **PCF_StaticFontGetSizeRequest**(*PCF_StaticFont *font*, const char **str*, Uint32 **w*, Uint32 **h*)

Computes space (pixels width*height) needed to draw a string using a given font. Both **w** and **h** can be NULL depending on which metric you are interested in. The function won't fail if both are NULL, it'll just be useless.

Parameters:

str The string whose size you need to know.

font The font you want to use to write that string

w Pointer to somewhere to place the resulting width. Can be NULL.

h Pointer to somewhere to place the resulting height. Can be NULL.

void **PCF_StaticFontGetSizeRequestRect**(*PCF_StaticFont *font*, const char **str*, SDL_Rect **rect*)
Same PCF_StaticFontGetSizeRequest as but fills an SDL_Rect. Rect x and y get initialized to 0.

Parameters:

str The string whose size you need to know.

font The font you want to use to write that string

rect Pointer to an existing SDL_Rect (cannot be NULL) to fill with the size request.

bool **PCF_StaticFontCanWrite**(*PCF_StaticFont *font*, SDL_Color **color*, const char **sequence*)
Check whether **font** can be used to write all chars given in **sequence** in color **color**.

Parameters:

color The color you want to write in

sequence All the chars you may want to use

Returns: true if all chars of **sequence** can be written in **color**, false otherwise.

void **PCF_StaticFontCreateTexture**()
Creates a hardware-friendly texture into **font**. Parameters depends on which support (SDL2_Renderer or SDL_gpu) was compiled in.

Parameters:

font The font to act on

renderer When using SDL2_Renderer, the renderer which the texture will belong to

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index