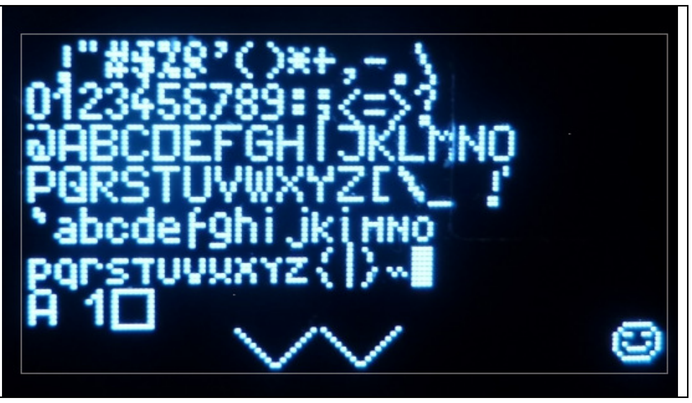
Oled I2C library - only 2k bytes

SSD 1306 OLED modules are cheap and easy to implement. Resolution is 126x64 pixels, small size but sharp. AdaFruit propose a nice GFX library, that need 10 to 20k for playing with graphic objects. A 1k buffer is required, and this is not available on small microcontrollers. The 2k OLED lib does not need a bitmap buffer, and is well suited as a debugging help. Variable and plots can be shown, also when the system is not connected to a PC.	 !"#$%&'()*+,-.\ 0123456789:;<=>? @ABCDEFGHIJKLMNO PQRSTUVWXYZ[_ ! `abcdefghijklmno pqrstuvwxyz{|}~ A 10

Oled I2C lib is a set of include files. Before listing them, let us explain some basic principles and options.

Hardware and communication

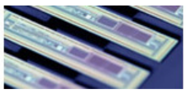
Cheap OLEDs, 128X64 are available SPI or I2C interface. SPI requires more lines and is not simpler on the software side.
I2C needs 2 lines with pull-ups. Only one OLED can be connected on the I2C bus, but many I2C devices can be connected on the same bus.
Many microcontrollers have internal hardware support for handling I2C (pin 18 and 19 on Arduino). Arduino IDE providse the Wire library, used by the OLED libs. No need for a complete lib. only the simple write is required by the SSD 1306.
This mean that a bitbang I2C function, easy to program, makes it possinble to use any pin.
Il place of the Wire lib, 2 include files can be used:
OledI2c.h uses 160 bytes
OledI2cbb.h (bit bang) uses 164 bytes
In comparison Wire needs about 2100 bytes

Arduino lib or include files

Arduino libraries are written in C++ and are portable through the family of Arduino cards. Their drawback is size and you have to be an experienced programmer to understand and modify.
The hardware is always well documented. When understood, C functions can do exactly what is required in an efficient way, as shown with I2C.
OLED hardware is a set of registers, A set of functions handles the register contents, another set of functions controls the graphic.

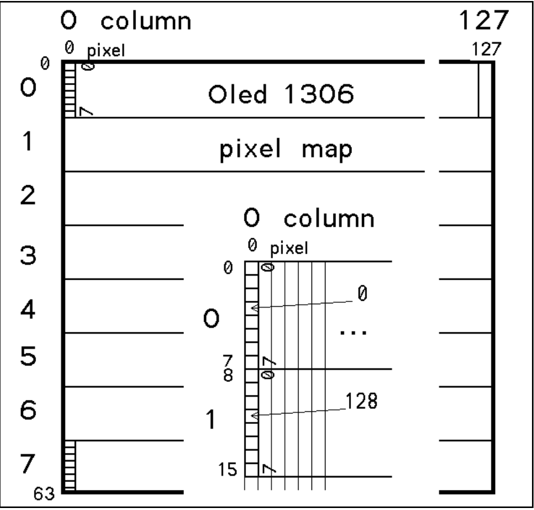
Oled 1306 pixels and buffers

The purpose of a screen is to show pixels to the user. The information is stored in a pixel buffer and a refresh process powers the visible pixels (shake your OLED to see the refresh rate).
The pixel buffer is a set of bytes. A programmable interface says how these bytes are transferred to the screen, and a serial interface, I2C in our case, receives the control and data bytes.
Solomon research, the manufacturer of the OLED controller hidden under the pixels, document how to interface the SSD 1306. Transfer functions hide this low level. Graphics can then be built. The first step is to understand how the pixels are arranged and how to act on them.


1mmx8mm - 120 pads

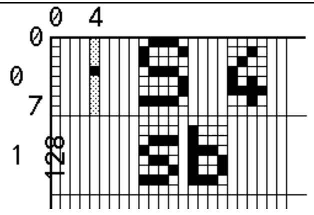
Pixel adressing

Screen shows 128x64 pixels taken from a pixel map of 128x8 bytes, least significant bit on top.
To fill the pixel map, one clears the buffer pointer and writes 1024 bytes. Bytes are shown on the screen immediately (within the 20 ms refresh rate).
A single byte can also be addressed and written. In this case, the buffer must be seen as 8 lines of 128 columns. The pointer increases after every write and zig-zags down the buffer.
Scrolling an object or positioning a character at any pixel position is time consuming.



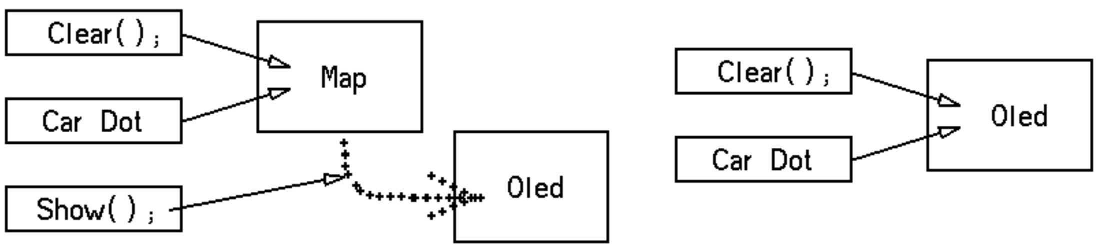
It is not possible to read a byte. This is a problem if one needs to modify a single pixel. It is part of a byte and all the byte will be modified. This a reason to prefer interacting with a software bitmap, the other reason being the software bitmap can have an organization optimized for pixel drawing, not related to OLED pixel map arrangement.

Due to the 1306 pixel addressing, it is much simpler to align the letters and numbers on a line (0..7) and column address (0..127). Letters are 5x7, numbers and lower case are 4x5.. For dots, address is as usually 0 to 63 vertically, 0 to 127 horizontally.



Local bitmap or direct transfers?

Known libraries use an additional bit map that gives the freedom to read, modify, write bytes. But to show every modification, the complete buffer must be transferred (one could imagine to transfer only the modified part, but it is not worth for such a small bitmap).
The bitmap is 1kbyte, this is a problem with cheap microcontrollers that could benefit to interact with a small OLED.
Writing directly to the pixel map indeed does not change the user software. The only drawback is writing a pixel above an object will clear some of its pixels. The advantage is to suppress refresh transfers, that take 50 ms.



OledMap and OledPix libraries

Libraries documented as a set of .h files have the advantage to be modular. With a minimum of understanding of C, you can modify or complete a module, suppress what is not required for the application, check code size and execution time, use a cheaper microcontroller for an industrial product.
OledMap saves space and download time on Arduino.
OledPix is the only solution for AVR ATtiny, Microchip 16F and TI low end MSP430 microcontrollers.

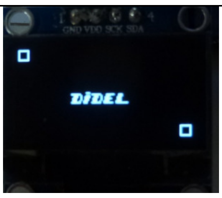
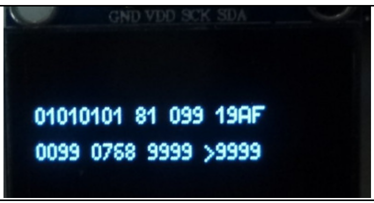
Oled Lib Include files

An **OledMap.ino** or **.c** program includes several files, not all mandatory.
OledMap.h (~300 bytes) includes the setup and functions to control the bitmap pointers and transfer bytes and stored bitmaps.
OledI2C.h (160 bytes) has only 3 primitives: Start(); Write(dd); Stop(): It is assumed the Oled is connected, and works.
OledI2Cbb.h (162 bytes) has the same functions and include the definitions of the Sck and Sda pins.
OledGenc.h (440 bytes) list the character generators: numbers (128 bytes), upper case (160 bytes) amd lower case (128 bytes). Three sprites (40 bytes) are given as example.
OledCar.h (~550 bytes). transfers characters and text. Double size characters can be removed to save 300 bytes.
OledNum.h () (~450 bytes) displays binary, hexa and decimal numbers, showing the non significative zero. Essential for debugging.
OledGra.h () (~120 bytes) defines Dot(x,y) and BigDot (if 2 curves mut be shown). Plus Hline and Vline, but no circle, etc.

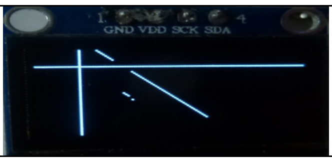
An **OledPix.ino** or **.c** program needs only **45 bytes of Ram**. It uses the same files, except the two that have to write characters or byte directly on the oled and not on the 1k bitmap. The files are **OledFix.h , OledI2C.h or OledI2Cbb.h, OledGenC.h, OledCarFix.h, OledGraFix.h and OledNum.h.**

Examples used for the evaluation of the code size

Arduino, no calls
 	
472 bytes
	#include "OledI2C.h"
	#include "OledGenc.h"
	#include "OledMap.h"
	#include "OledCar.h"
	#include "OledNum.h"
	#include "OledGra.h"
	void setup(){
	
	}
	void loop(){
	
	for(;;); }
		OledMap.h min
		
		796 bytes
		
		I2C only
		OledI2c 630 bytes
		OledI2Cbb 634 bytes
			…
			void setup(){
			SetupI2c();
			SetupOledMap(); //with clear
			}
			void loop(){
			
			Show();
			for(;;);
			}

OledMap.h
 	
820 bytes
994 with sprites
	…
	void loop(){
	LiCol(0,0); Sprite (carre);
	LiCol(7,120); Sprite (carre);
	LiCol(4,40); Sprite (didel);
	Show(); >>
	for(;;); }
		 DIDEL	 01010101 81 099 19AF
			0099 0768 9999 >9999

OledCar.h
 	
2126 bytes
	…
	void loop(){
	LiCol(0,8);
	Car('A');Car('a');Car('2');
	LiCol(2,8);
	Big('A');Big('b');Big('2');
	Show();
	for(;;); }
	
	vvv
		OledNum.h
		
		2224 bytes
			. . . ^^^
			void loop(){
			LiCol(3,0); Bin8(0x55); Hex8 (129);
			Dec8 (0x63); Hex16 (0x19AF);
			LiCol(5,0); Dec9999(0x63);
			Dec9999(0x300); Dec9999(0x270F);
			Dec9999(0x3000);
			Show();
			for(;;); }

	Ab2 Ab2 Ab2 Ab2		

OledCar.h
 	
2170 bytes
	… ^^^
	void loop(){
	LiCol(0,8); Car('A');Car('a');Car('2');
	LiCol(2,8); Big('A');Big('b');Big('2');
	Cmd (0xda);Cmd (0x02);
	Show();
	for(;;); }
		OledGra.h
		
		1136 bytes
			void loop(){ ^^^
			Dot (40,40); Dot (41,41); Dot (42,42);
			DDot (30,50);
			hLine(20); vLine(20);
			for (byte i=0; i<50; i++) Dot(20+i,i);
			Show();
			for(;;); }

jdn 170430

Code coming (may-june 2017) on github nicoud.