USER MANUAL

64 LED RGB MATRIX

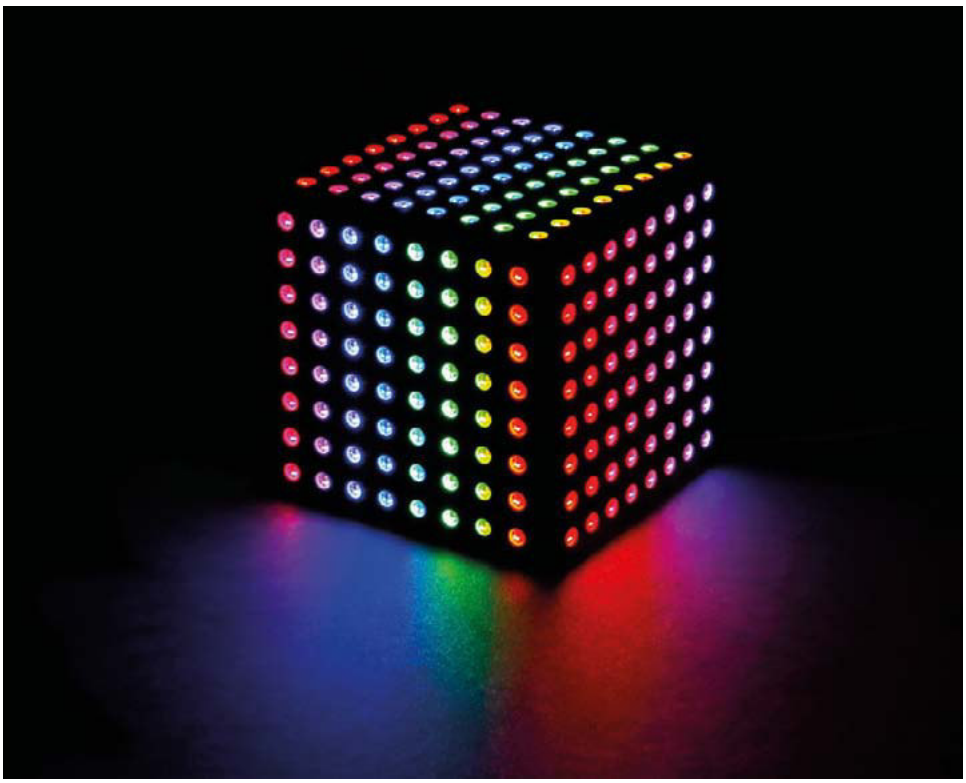
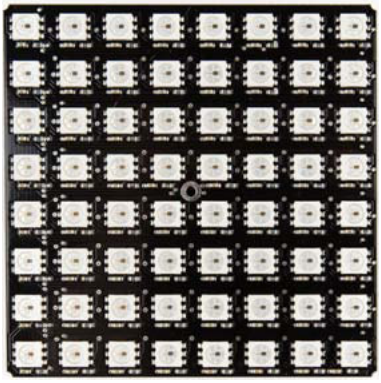


Table of contents



LED's get started!

Preparing the 64 LED RGB matrix

Before we go into detail about how to write software and display something on your panel(s) we'll explain how to prepare your panel(s) correctly.

Data lines

Before adding power, you need to understand how the WS2812 LEDs are connected to each other. Each LED acts as a shift register and shifts display data from its input to its output and over to the next LED. That is how the display data propagates through a panel, from one led to the next. In the VM207, the LEDs are connected in rows and each row has a returning line so the beginning of the next row can be connected to the end of the previous row. This feature has the benefit that rows are not "zigzaged" and it is possible to connect more panels together without having to jump through software hoops to control a bigger panel.

Study the drawing below carefully to understand how these LEDs are wired.

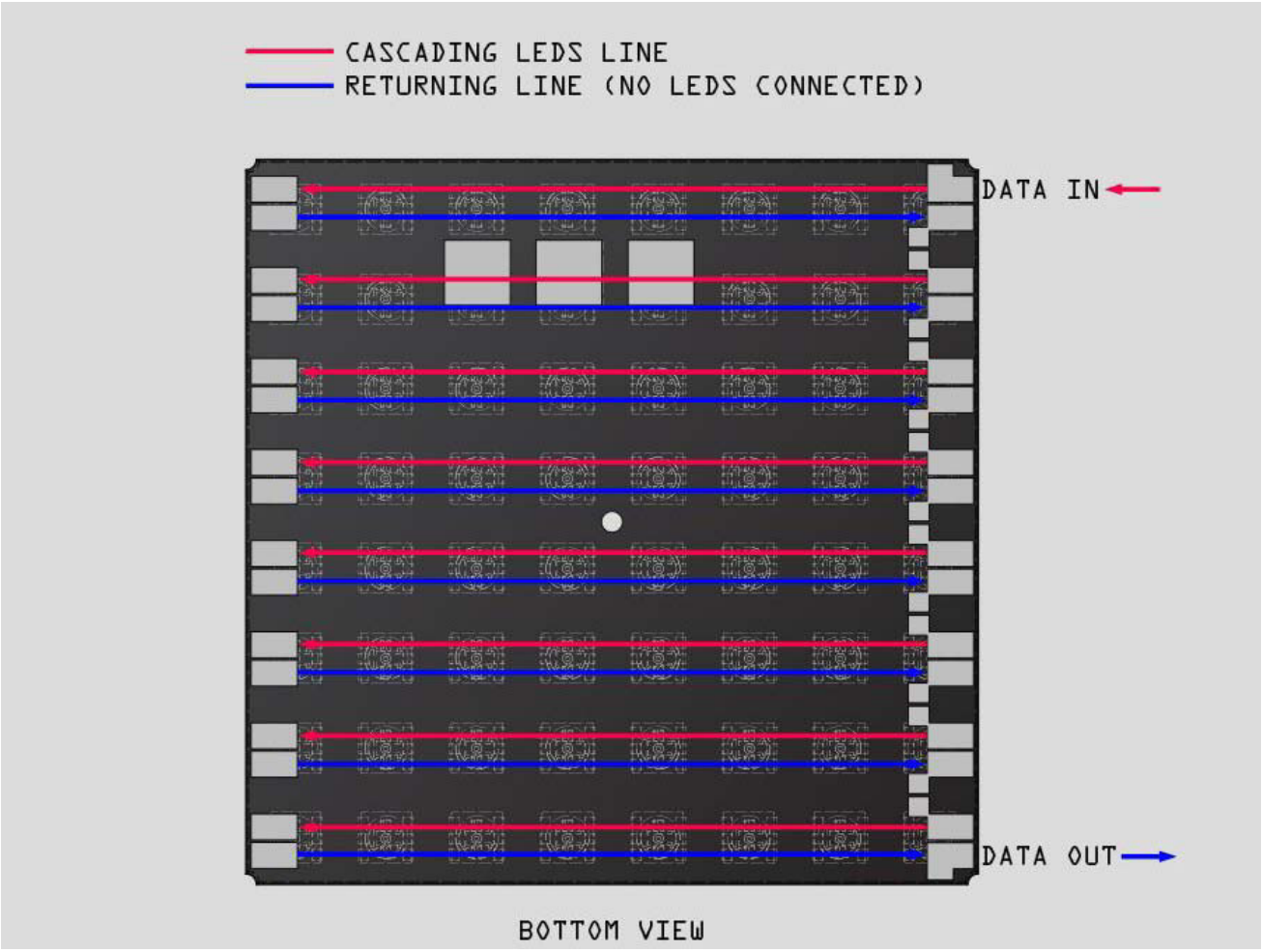


fig. 1

You can see that the panel is not going to work right away because the first row just ends and isn't connected to anything. The display data would just stop at the end of the first row of LEDs and not continue through the panel. To use the panel as an 8 x 8 display and let the display data travel to each LED you will have to connect the pads as shown in the drawing below (indicated by the yellow soldering connections). Just using a bit of solder to connect these pads is enough.

If you follow the display data pad now, you can see that all rows are interconnected and they effectively represent a long string of WS2812 LEDs.

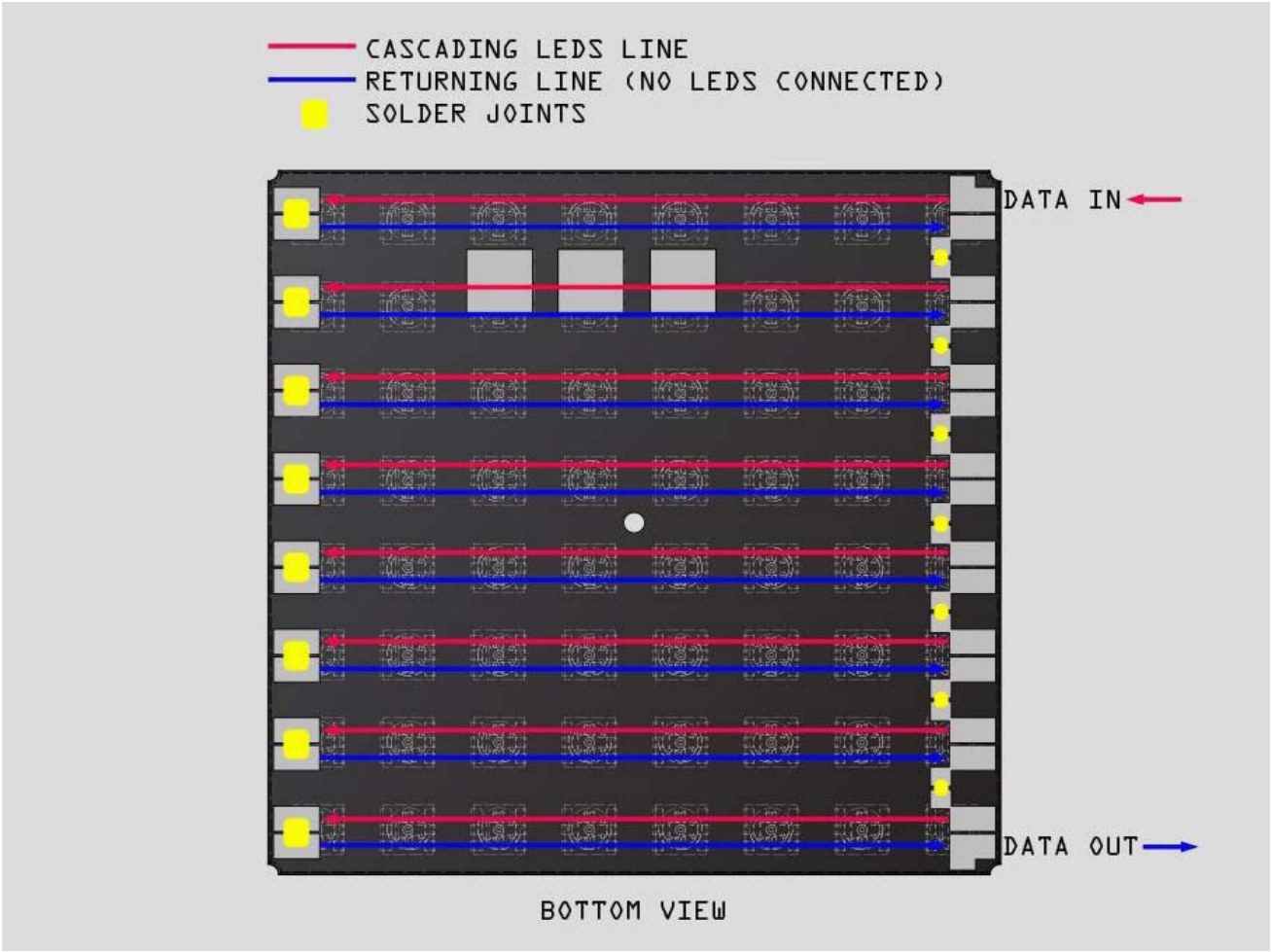


fig. 2

The drawing below shows how you can tile two (or more) panel(s) next to each other by connecting the solder pads at the sides with some bare wire.

You can see that a row now exists out of 16 LEDs instead of 8.

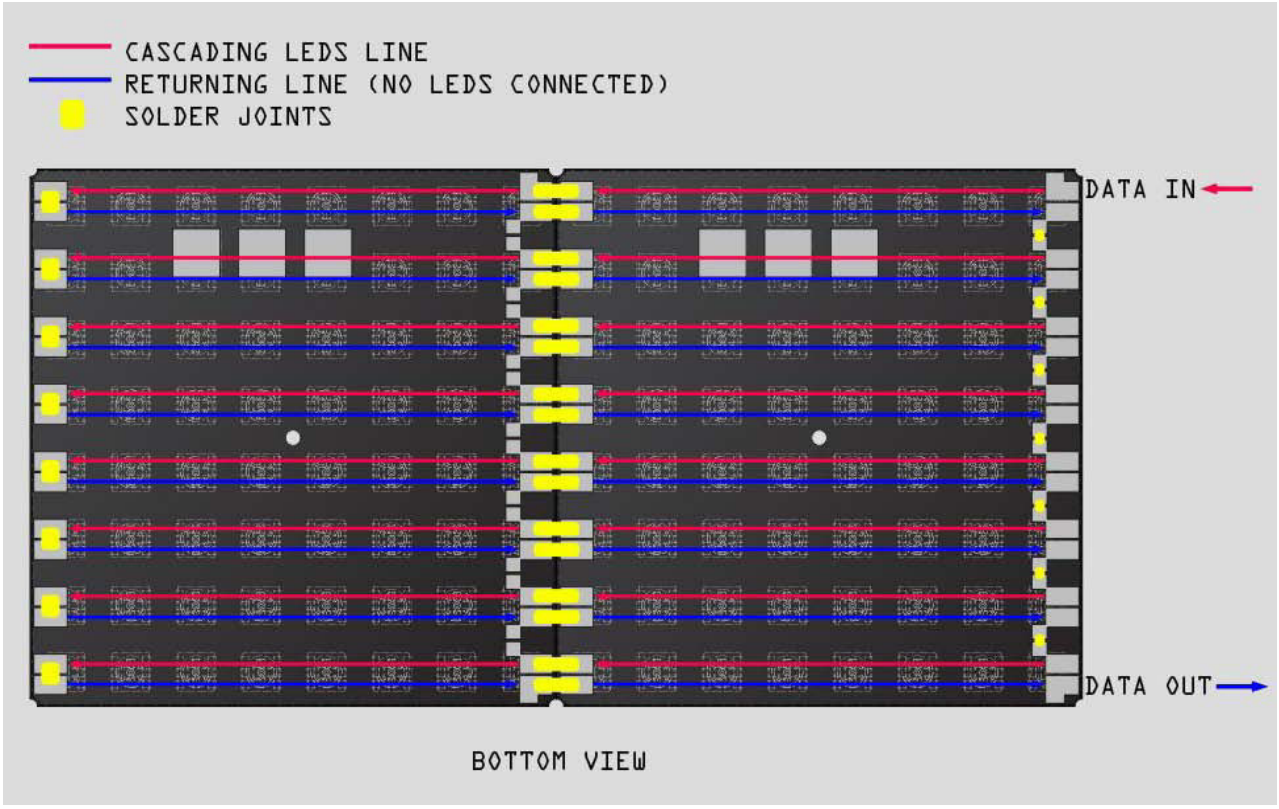


fig. 3

You can also tile the panels vertically, this is done by just connecting the DATA OUT of the previous row of panels to the DATA IN of a new row of panels. Be careful as this connection can be fragile when you have larger assemblies.

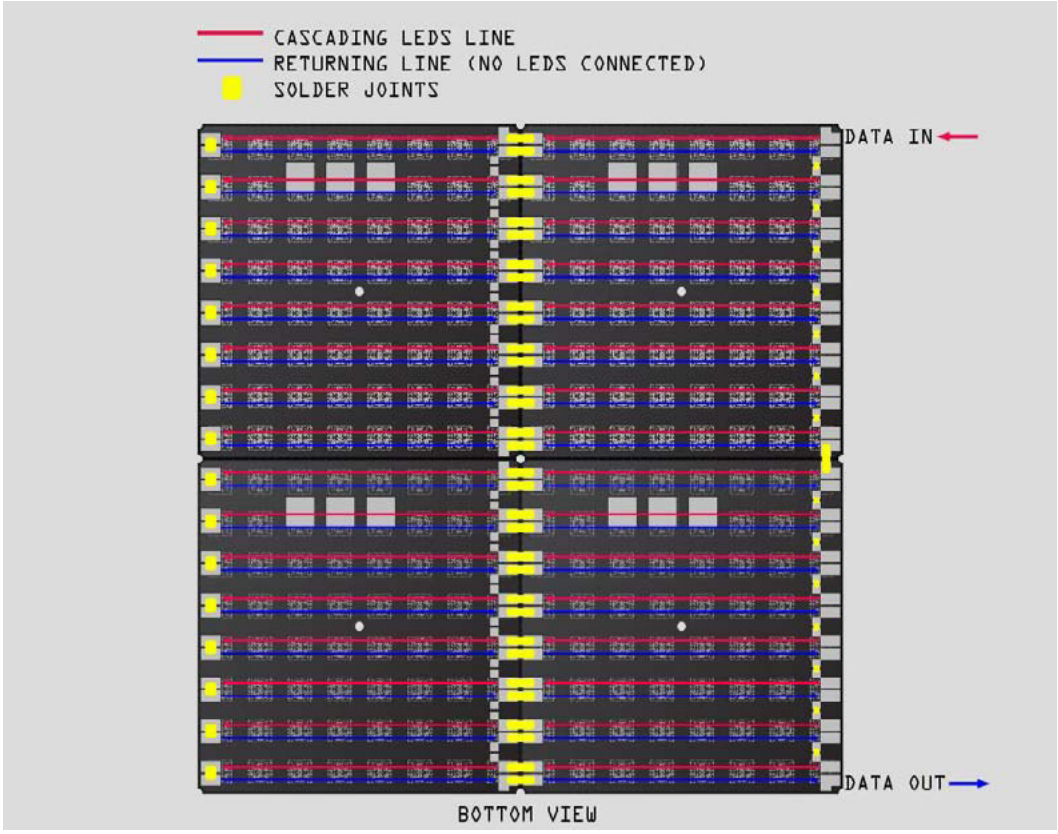


fig. 4

SOME THINGS TO REMEMBER

- Keep the data connections as short as possible.
- Do not split the data line as this will not work. (One LED cannot send data to two or more following LEDs)
- The LEDs expect a TTL level (5 V) data signal but you can get an assembly working with an 3.3 V micor-controller output (but 5V is better...)
- Place a 470 Ohm resistor between the controller data output pin and the input of the first panel. (This is not always necessary but it can help when you are having trouble with noisy data lines)

Power supply

Powering the VM207 is a bit simpler. There are 3 contacts labeled **DV+**, **LV+** and **GND**, they respectively stand for **Data Voltage +**, **LED Voltage +**, and **Ground**. The WS2812 LEDs that are mounted on the panels are the 6 pin variant and these have separate pins for the 5 V for the LED die and the 5 V for the IC die inside the package. So the DV+ pad is connected to all the IC die pins and the LD+ pin is connected to all the LED die pins. In most occasions you can just connect these two and all will be OK. If you are worried by brown-outs due to line-loss when you have a lot of panels that are operating at high brightness, you can have the IC dies on a seperate 5 V supply (common ground!)

So the easiest way to power the panel is as shown in the drawing below:

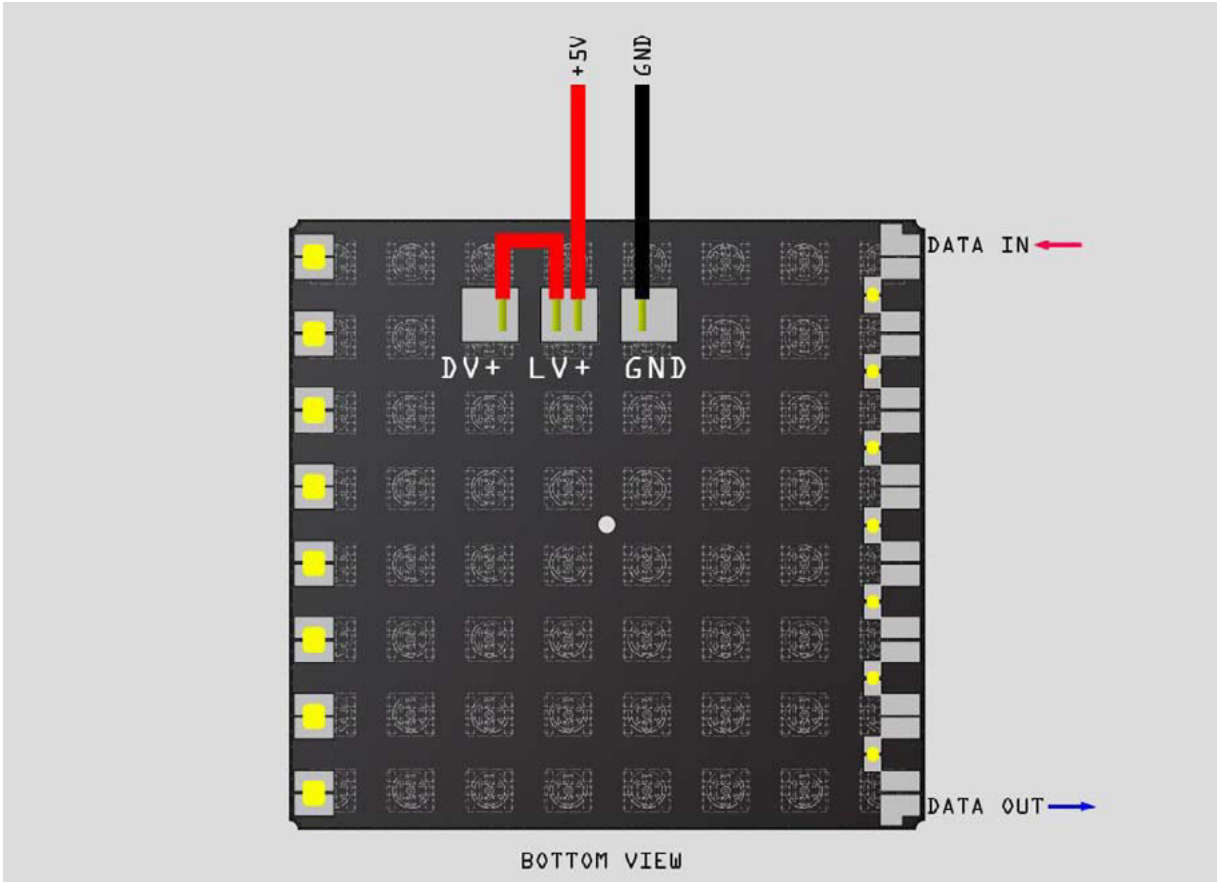


fig. 5

When you have more panels you will need to place them in parallel. **When you are having a lot of panels, make sure you are using wires of the correct gauge or use multiple wires from the supply to the panels!**

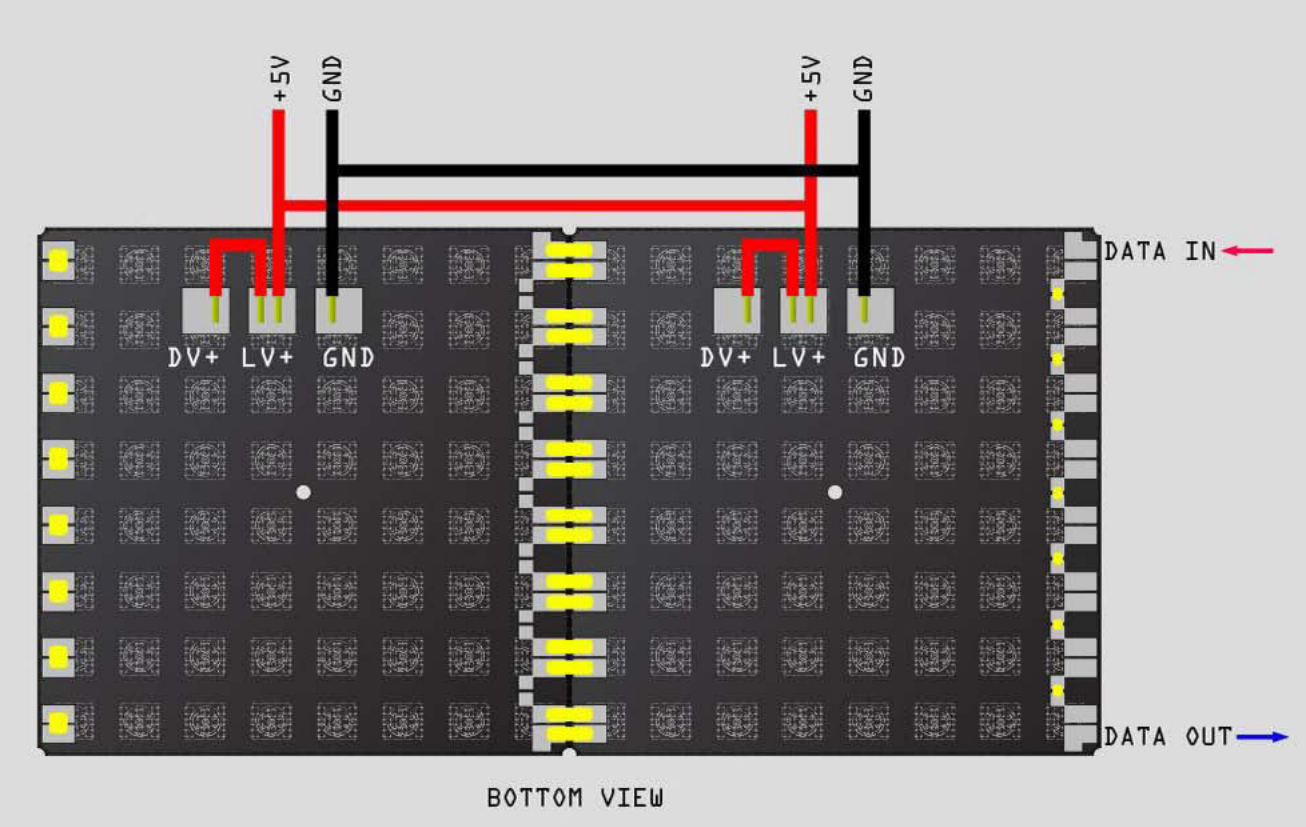


fig. 6

SOME THINGS TO REMEMBER

- Always use a 1000 µf capacitor (included) in parallel with your power supply.
- Do not turn your power supply on and off with the panels connected as power-up/down spikes can harm the LEDs on your panel.
- Always connect ground first.
- Use an appropriate cable to power your panels as these can draw a lot of amps (3.5 A per panel at full brightness). Line-loss voltage can be a factor when working with these panels.
- Do not stare at the panels at full brightness from a close distance, this can be disorientating.
- Remember that using a lot of panels at full brightness can generate a lot of heat, in some applications you will have to address this heat build-up (fan, heatsink, ...) to ensure the life of the panels.

Controlling the 64 LED RGB matrix

What microcontroller should I use?

You can use a lot of microcontrollers or microcontroller platforms to control your panel. But to make things easy and fast, we are going to explain how you use your panel with a microcontroller platform that is compatible with the Arduino IDE, like the Uno, Mega, Teensy and many others.

If you do not want to use any of these platforms, you will have to dive a bit deeper and learn how to create the data stream that is needed to control a bunch of WS2812 LEDs. You can find the needed information here: WS2812 DATASHEET.

Connecting a panel to an Arduino Uno

Follow the drawing below to connect a panel to an Arduino Uno. At the moment we use pin 6 as the data output but this can be changed in the code later. Supply the Arduino with power as well as the VM207 and connect the grounds together.

ATTENTION

Do not use the 5 V from the Arduino to power one or more panels. These panels draw to much current for the regulator on the Arduino. If you do this you could break your Arduino board.

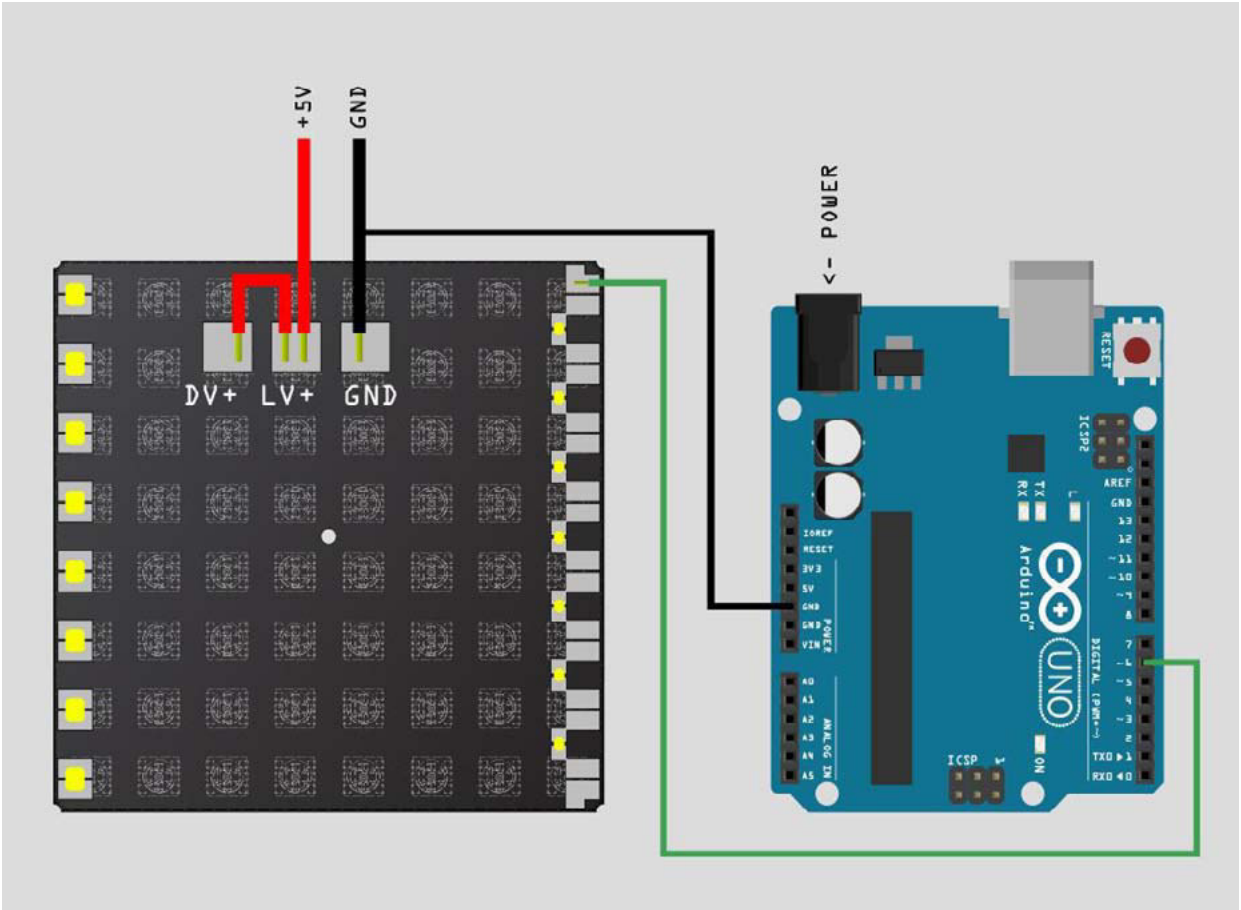


fig. 7

Adafruit neopixel library

First we will explain the ADAFRUIT NEOPIXEL library. This library written by Adafruit can control a whole bunch of WS2812 LEDs seperatly. So this isn't exactly useful when you want to draw text or shapes on your panel but it is when you want to control each LED separately and do your own thing with it.

You can download the ADAFRUIT NEOPIXEL library here: https://github.com/adafruit/Adafruit_NeoPixel (press the **Download ZIP** button)

Then you can install this library in your Arduino installation (place the downloaded and unpacked folder in the libraries folder of the Arduino installation) and then start the Arduino software.

If you now go to: **File > Examples > Adafruit Neopixel > simple**, Arduino will open that sketch.

The top section of the file that you just opened will look like this:

```
// NeoPixel Ring simple sketch (c) 2013 Shae Erisson
// released under the GPLv3 license to match the rest of the AdaFruit NeoPixel library

#include <Adafruit_NeoPixel.h>
#ifdef __AVR__
  #include <avr/power.h>
#endif

// Which pin on the Arduino is connected to the NeoPixels?
// On a Trinket or Gemma we suggest changing this to 1
#define PIN            6

// How many NeoPixels are attached to the Arduino?
#define NUMPIXELS      16

// When we setup the NeoPixel library, we tell it how many pixels, and which pin to use to send sig-
nals.
// Note that for older NeoPixel strips you might need to change the third parameter--see the strand-
test
// example for more information on possible values.
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
```

What is going on here?

Well the first part is just about including the Adafruit library.

Then **#define PIN 6** tells the program that PIN equals 6 and that the datastream will come out pin 6 of the Arduino, so if we want to change that to 13 we would have to change this line of code into: **#define PIN 13**

Next, we have the **#define NUMPIXELS 16** line which tells the program how many LEDs are going to be controlled. So to control 1 panel, this line should look like this: #define NUMPIXELS 64. If you have 2 panels this should be 128, 3 panels = 192, and so on.

Next we see this line: **Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);**

This line gives a name to our bunch of LEDs, here this is "**pixels**". You could easily change this to "panel" or something else but you would have to change all the occurrences of "pixels" in the rest of the program.

Then we tell the program how "pixels" (our panel in this case) is build-up: **Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);**

- NUMPIXELS = We have defined this value earlier in the program. This is the amount of pixels that need to be controlled.
- PIN = We have defined this value earlier in the program. This is the output pin where the display data-stream will be located.
- NEO_GRB = Leave this for standard WS2812 LEDs.
- NEO_KHZ800 = Leave this for standard WS2812 LEDs.

Then we have this code which is just a variable that stores a value which will be used as a delay in the main function. Change this value and the speed at which the for-loop in the loop function iterates will change.

```
int delayval = 500; // delay for half a second

Below that piece of code there is the setup function that looks like this:

void setup() {
  // This is for Trinket 5V 16MHz, you can remove these three lines if you are not using a Trinket
  #if defined (__AVR_ATtiny85__)
    if (F_CPU == 16000000) clock_prescale_set(clock_div_1);
  #endif
  // End of trinket special code

  strip.begin();
}
```

The most important part of this piece of code is the **strip.begin();** line. This initialises the LEDs. Do not forget this step in your own code.

Next, we have the main loop function:

```
void loop() {

  // For a set of NeoPixels the first NeoPixel is 0, second is 1, all the way up to the count of pix-
els minus one.

  for(int i=0;i<NUMPIXELS;i++){

    // pixels.Color takes RGB values, from 0,0,0 up to 255,255,255
    pixels.setPixelColor(i, pixels.Color(0,150,0)); // Moderately bright green color.

    pixels.show(); // This sends the updated pixel color to the hardware.

    delay(delayval); // Delay for a period of time (in milliseconds).

  }
}
```

This is where the magic happens. The loop function repeats itself every time, and inside that loop function there is a for-loop that will execute these lines every half a second:

```
    pixels.setPixelColor(i, pixels.Color(0,0,255)); // Bright blue color.

    pixels.show(); // This sends the updated pixel color to the hardware.

    delay(delayval); // Delay for a period of time (in milliseconds).
```

The first line will give the pixel at the place that is stored in value "i" a green color, but the panel will only be updated with this information when the second line "pixels.show" has been called. Since these two lines get called every 500 ms and i always increases with 1 (wonder why this is so? Read here: https://www.arduino.cc/en/reference/for about how for-loops work) you can see that the panel should slowly fill up with green LEDs when you run this code on an Arduino.

Let's try that, compile and upload the code to your Arduino and hook everything up. You should see the panel slowly (every 500 ms) light up with one more green LED. You will also notice that once the panel has filled up with green pixels it stays green. To change that, you could try changing the loop function to this and also try changing the delay value mentioned before (double click to select and copy):

```
void loop() {

  // For a set of NeoPixels the first NeoPixel is 0, second is 1, all the way up to the count of pix-
els minus one.

  for(int i=0;i<NUMPIXELS;i++){
    // pixels.Color takes RGB values, from 0,0,0 up to 255,255,255
    pixels.setPixelColor(i, pixels.Color(0,0,255)); // Brigh blue color.
```

```
    pixels.show(); // This sends the updated pixel color to the hardware.

    delay(delayval); // Delay for a period of time (in milliseconds).
  }
  for(int i=0;i<NUMPIXELS;i++){
    pixels.setPixelColor(i, pixels.Color(0,0,0)); // No color (dark).
  }
  pixels.show(); //Updating the panel to show nothing.
}
```

This code will erase all values after the initial for-loop using a second for-loop. Also, we have changed the colour to fully saturated blue. You will see that this is very bright. To change the brightness of the complete panel you can use: **pixels.setBrightness(20);** this function will accept a value between 0 and 255 where 0 is dark and 255 is full brightness. You can see how this affects the display if you change the loop function to this (double click to select and copy):

```
void loop() {
  pixels.setBrightness(20); // Setting the brightness really low.
  // For a set of NeoPixels the first NeoPixel is 0, second is 1, all the way up to the count of pix-
els minus one.
  for(int i=0;i<NUMPIXELS;i++){
    // pixels.Color takes RGB values, from 0,0,0 up to 255,255,255
    pixels.setPixelColor(i, pixels.Color(0,0,255)); // Bright blue color.

    pixels.show(); // This sends the updated pixel color to the hardware.

    delay(delayval); // Delay for a period of time (in milliseconds).
  }
  for(int i=0;i<NUMPIXELS;i++){
    pixels.setPixelColor(i, pixels.Color(0,0,0)); // No color (dark).
  }
  pixels.show(); // Updating the panel to show nothing.
}
```

Below you can find how the complete code is looking at the moment. I encourage you to experiment with the things you learned in this chapter before continuing. Double click to select and copy.

```
// NeoPixel Ring simple sketch (c) 2013 Shae Erisson
// released under the GPLv3 license to match the rest of the AdaFruit NeoPixel library

#include <Adafruit_NeoPixel.h>
#ifdef __AVR__
  #include <avr/power.h>
#endif

// Which pin on the Arduino is connected to the NeoPixels?
```

```
// On a Trinket or Gemma we suggest changing this to 1
#define PIN            6

// How many NeoPixels are attached to the Arduino?
#define NUMPIXELS      64

// When we setup the NeoPixel library, we tell it how many pixels, and which pin to use to send signals.
// Note that for older NeoPixel strips you might need to change the third parameter--see the strandtest
// example for more information on possible values.
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);

int delayval = 50; // delay for half a second

void setup() {
  // This is for Trinket 5V 16MHz, you can remove these three lines if you are not using a Trinket
#if defined (__AVR_ATtiny85__)
  if (F_CPU == 16000000) clock_prescale_set(clock_div_1);
#endif
  // End of trinket special code

  pixels.begin(); // This initializes the NeoPixel library.
}

void loop() {
  pixels.setBrightness(20); // Setting the brightness really low.
  // For a set of NeoPixels the first NeoPixel is 0, second is 1, all the way up to the count of pixels minus one.
  for(int i=0;i<NUMPIXELS;i++){
    // pixels.Color takes RGB values, from 0,0,0 up to 255,255,255
    pixels.setPixelColor(i, pixels.Color(0,0,255)); // Bright blue color.

    pixels.show(); // This sends the updated pixel colour to the hardware.

    delay(delayval); // Delay for a period of time (in milliseconds).
  }
  for(int i=0;i<NUMPIXELS;i++){
    pixels.setPixelColor(i, pixels.Color(0,0,0)); // Moderately bright green color.
  }
  pixels.show(); // Updating the panel to show nothing.
}
```

Adafruit neomatrix library

If you want to display text or draw shapes on your panel(s) you can use the ADAFRUIT NEOMATRIX library. To make this library work you actually also need to use the ADAFRUIT GFX libraries. This one takes care of the shapes, letters and colours while the neomatrix takes care of sending all that data to the panels.

You can download the ADAFRUIT NEOMATRIX library here: https://github.com/adafruit/Adafruit_NeoMatrix (press the **Download ZIP** button)

You can download the ADAFRUIT GFX library here:https://github.com/adafruit/Adafruit-GFX-Library (press the **Download ZIP** button)

Then you can again install these libraries in your Arduino installation (place the downloaded and unpacked folders in the libraries folder of the Arduino installation) and then start the Arduino software (make sure you also have the ADAFRUIT NEOPIXEL library installed).

If you now go to: **File > Examples > Adafruit Neopixel > simple** Arduino will open that sketch.

The top section of the file that you just opened will look like this:

```
#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>
#include <Adafruit_NeoPixel.h>
#ifndef PSTR
 #define PSTR // Make Arduino Due happy
#endif

#define PIN 6

// MATRIX DECLARATION:
// Parameter 1 = width of NeoPixel matrix
// Parameter 2 = height of matrix
// Parameter 3 = pin number (most are valid)
// Parameter 4 = matrix layout flags, add together as needed:
//   NEO_MATRIX_TOP, NEO_MATRIX_BOTTOM, NEO_MATRIX_LEFT, NEO_MATRIX_RIGHT:
//     Position of the FIRST LED in the matrix; pick two, e.g.
//     NEO_MATRIX_TOP + NEO_MATRIX_LEFT for the top-left corner.
//   NEO_MATRIX_ROWS, NEO_MATRIX_COLUMNS: LEDs are arranged in horizontal
//     rows or in vertical columns, respectively; pick one or the other.
//   NEO_MATRIX_PROGRESSIVE, NEO_MATRIX_ZIGZAG: all rows/columns proceed
//     in the same order, or alternate lines reverse direction; pick one.
//   See example below for these values in action.
// Parameter 5 = pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400  400 KHz (classic •vĬ (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
```

```
// Example for NeoPixel Shield.  In this application wĕd like to use it
// as a 5x8 tall matrix, with the USB port positioned at the top of the
// Arduino.  When held that way, the first pixel is at the top right, and
// lines are arranged in columns, progressive order.  The shield uses
// 800 KHz (v2) pixels that expect GRB colour data.
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(5, 8, PIN,
  NEO_MATRIX_TOP     + NEO_MATRIX_RIGHT +
  NEO_MATRIX_COLUMNS + NEO_MATRIX_PROGRESSIVE,
  NEO_GRB            + NEO_KHZ800);
```

The first part includes all 3 libraries (NEOPIXEL, NEOMATRIX and GFX).

Next we will define an output pin again. This is the same principle as in the previous chapter.

And now we have a comment section where will be explained how to configure the library, so it knows how your panel assembly looks like. First we will give a name to our panel, in this case:"matrix". Then there are 5 parameters that you need to take care of:

- width of your matrix (number of LEDs that span the width of your matrix (8, 16, 32, ...)
- height of your matrix (number of LEDs that span the height of your matrix (8, 16, 32, ...)
- PIN number of the data output pin.
- This parameter describes the layout of the panel assembly and is made out of flags that you need to add together. First you need to specify where the first LED in the matrix sits. Mostly this is in the top left corner so this should be: **NEO_MATRIX_TOP + NEO_MATRIX_LEFT**. Then you need to choose wether the LEDs are chained in rows or columns. If you use the panels horizontally, you need to choose: **NEO_MATRIX_ROWS**. Then you need to choose wether the rows (or columns) are progressive or zigzagged, with the VM207 this is ALWAYS progressive so this becomes: **NEO_MATRIX_PROGRESSIVE**.
- The last parameter is about what LEDs you use with the VM207, this is ALWAYS: **NEO_GRB + NEO_KHZ800**.

So all these parameters need to be put together as follows to drive 1 panel, for example (you will need to alter the code from the example to the following code as this is made to drive an 5 x 8 panel with different properties):

```
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(8, 8, PIN,
  NEO_MATRIX_TOP     + NEO_MATRIX_LEFT +
  NEO_MATRIX_ROWS + NEO_MATRIX_PROGRESSIVE,
  NEO_GRB            + NEO_KHZ800);
```

If you ever build a bigger display out of VM207 panels you just need to think of it as one big panel and adjust the parameters accordingly.

The following code in the program is just an array with 3 colours so we can cycle through some colours to make the example a bit more special:

```
const uint16_t colors[] = {
  matrix.Color(255, 0, 0), matrix.Color(0, 255, 0), matrix.Color(0, 0, 255) };
```

In the setup function there are a few things we must take care of, but the most important part is the **matrix.begin();** function as this will initialise the panel. All the code that follows takes care of the animating text that will be displayed on the panel. So lets compile the code and upload it to your Arduino Uno and see if it works! If you need the full code you can copy it below:

```
// Adafruit_NeoMatrix example for single NeoPixel Shield.
// Scrolls •Howdў across the matrix in a portrait (vertical) orientation.

#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>
#include <Adafruit_NeoPixel.h>
#ifndef PSTR
 #define PSTR // Make Arduino Due happy
#endif

#define PIN 6

// MATRIX DECLARATION:
// Parameter 1 = width of NeoPixel matrix
// Parameter 2 = height of matrix
// Parameter 3 = pin number (most are valid)
// Parameter 4 = matrix layout flags, add together as needed:
//   NEO_MATRIX_TOP, NEO_MATRIX_BOTTOM, NEO_MATRIX_LEFT, NEO_MATRIX_RIGHT:
//     Position of the FIRST LED in the matrix; pick two, e.g.
//     NEO_MATRIX_TOP + NEO_MATRIX_LEFT for the top-left corner.
//   NEO_MATRIX_ROWS, NEO_MATRIX_COLUMNS: LEDs are arranged in horizontal
//     rows or in vertical columns, respectively; pick one or the other.
//   NEO_MATRIX_PROGRESSIVE, NEO_MATRIX_ZIGZAG: all rows/columns proceed
//     in the same order, or alternate lines reverse direction; pick one.
//   See example below for these values in action.
// Parameter 5 = pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400  400 KHz (classic •vĭ (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)


// Example for NeoPixel Shield.  In this application wĕd like to use it
// as a 5x8 tall matrix, with the USB port positioned at the top of the
// Arduino.  When held that way, the first pixel is at the top right, and
// lines are arranged in columns, progressive order.  The shield uses
// 800 KHz (v2) pixels that expect GRB color data.
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(8, 8, PIN,
  NEO_MATRIX_TOP     + NEO_MATRIX_LEFT +
```

```
  NEO_MATRIX_ROWS + NEO_MATRIX_PROGRESSIVE,
  NEO_GRB            + NEO_KHZ800);

const uint16_t colors[] = {
  matrix.Color(255, 0, 0), matrix.Color(0, 255, 0), matrix.Color(0, 0, 255) };

void setup() {
  matrix.begin();
  matrix.setTextWrap(false);
  matrix.setBrightness(40);
  matrix.setTextColor(colors[0]);
}

int x    = matrix.width();
int pass = 0;

void loop() {
  matrix.fillScreen(0);
  matrix.setCursor(x, 0);
  matrix.print(F("Howdy"));
  if(--x < -36) {
    x = matrix.width();
    if(++pass >= 3) pass = 0;
    matrix.setTextColor(colors[pass]);
  }
  matrix.show();
  delay(100);
}
```

You should see the word “Howdy” cycle on the panel in 3 different colours if everything went correct.
This is a quick example of what you can do with these libraries but ADAFRUIT GFX library can do a lot more and has a lot of functions you can use. These are all explained in the following manual: https://learn.adafruit.com/downloads/pdf/adafruit-gfx-graphics-library.pdf .

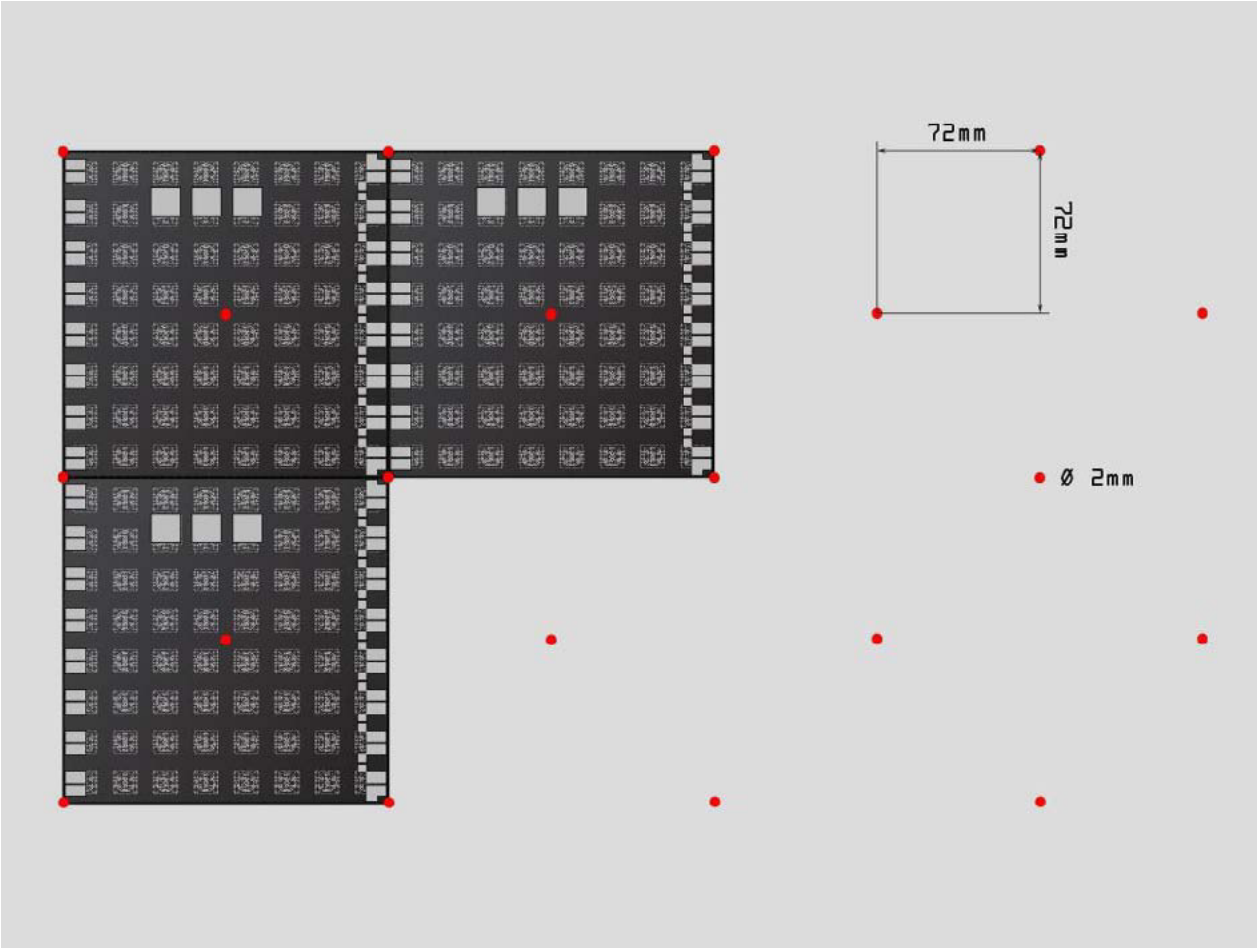
This manual explaines everything with a real display in mind but everything stays the same for the panels if you use the NEOMATRIX library in conjunction with the GFX library.

So read up on all the cool things you can do with these libraries and create some cool displays!

Mounting the 64 LED RGB matrix

Mounting holes

The panels have mounting holes in a pattern that tiles when you use multiple panels. These holes are 2 mm in diameter so use a small M2 screw to mount the panels. They are spaced 72 mm apart and staggered. See the drawing below:

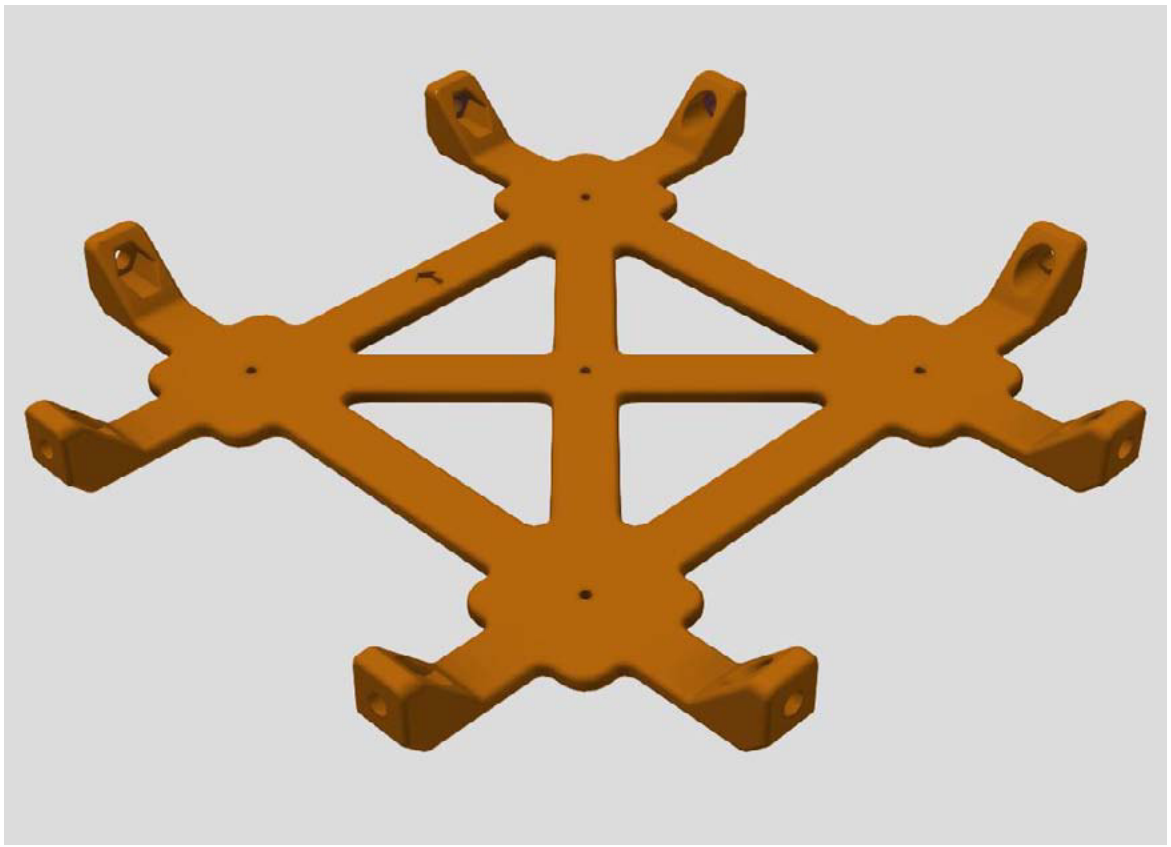


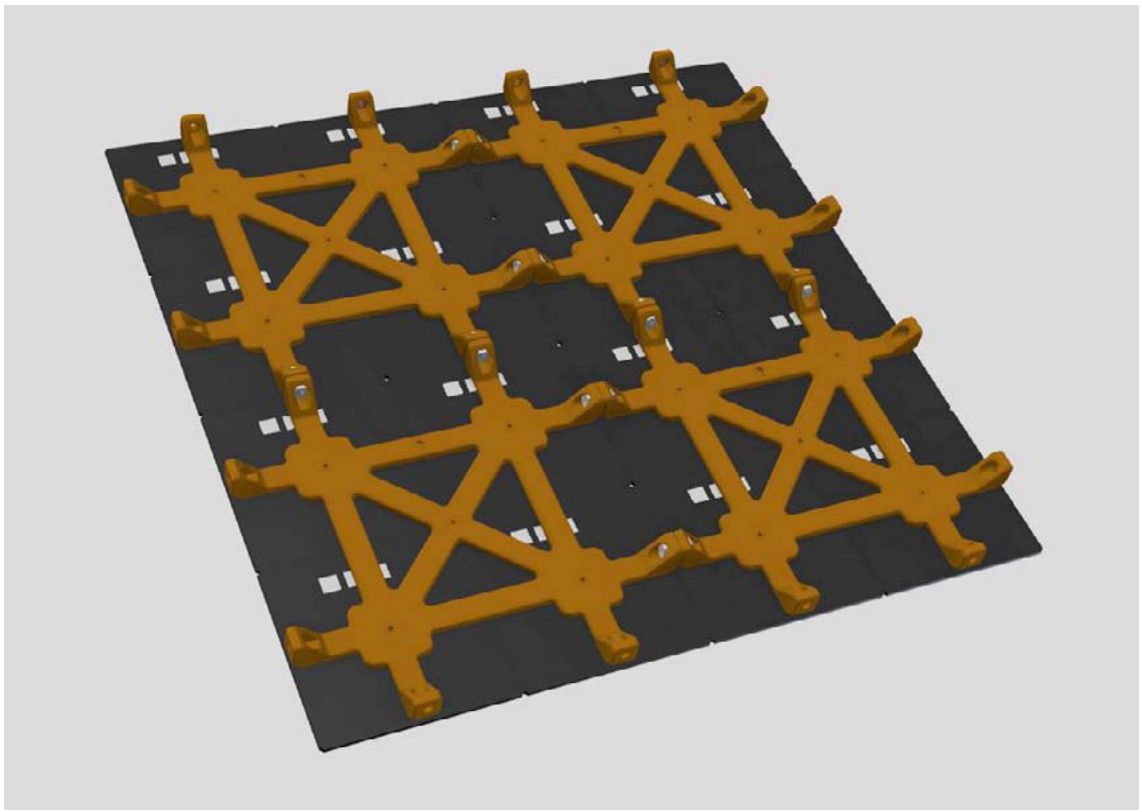
3D print mounting brackets

You can download the following mounting brackets if you have a 3D printer:

vm207-tile_mount.stl

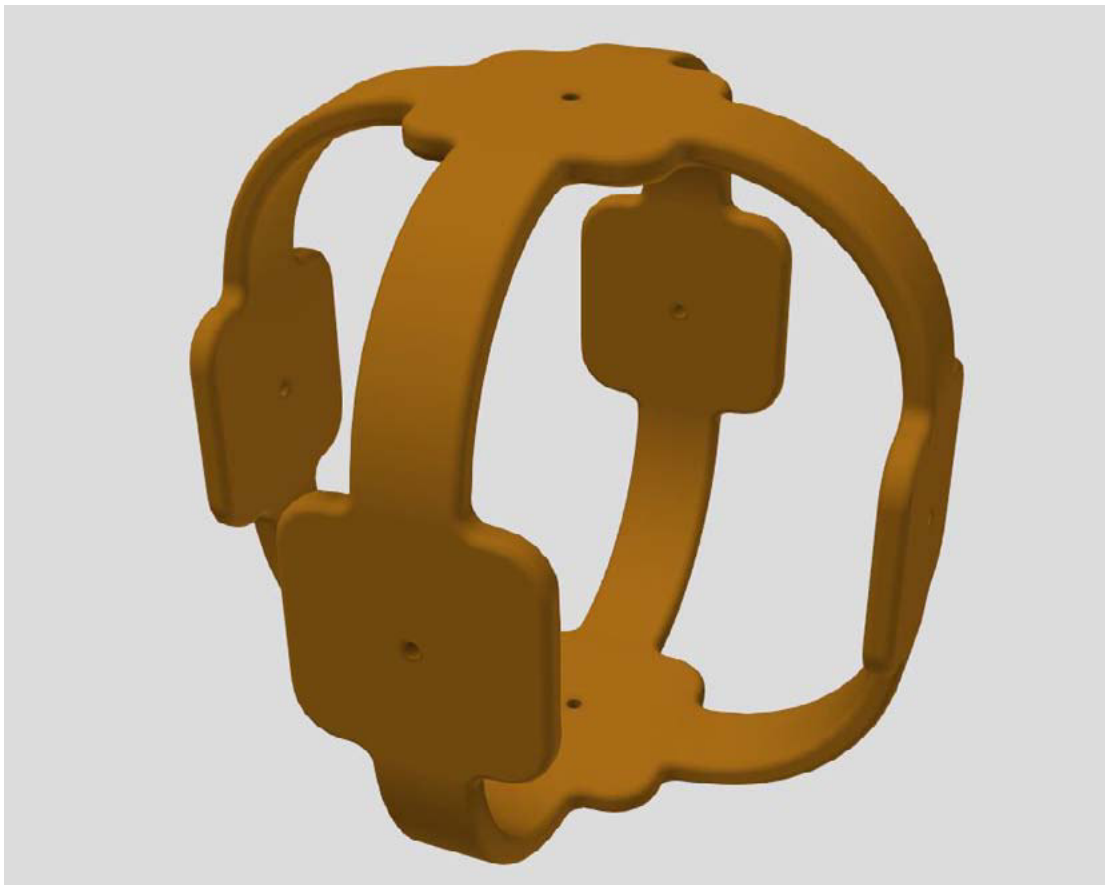
Several mounts connect together with M3 x 10 bolts.

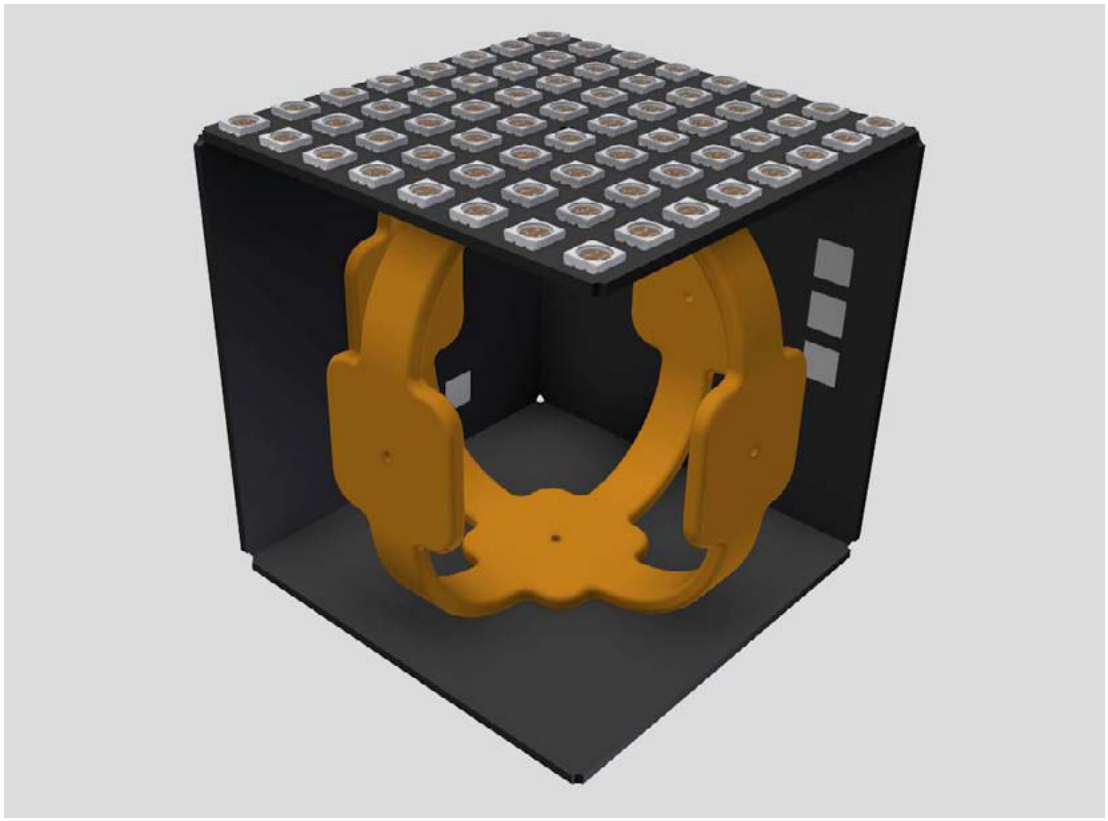




vm207-cube_mount.stl

Requires 6 x VM207 panels. Pay extra attention to how the panels need to be connected to each other!





ORDERCODE: VM207

REVISION: HVM207'1

VellemanProjects @Velleman_RnD
VELLEMAN nv - Legen Heirweg 33, Gavere (Belgium)
vellemanprojects.com