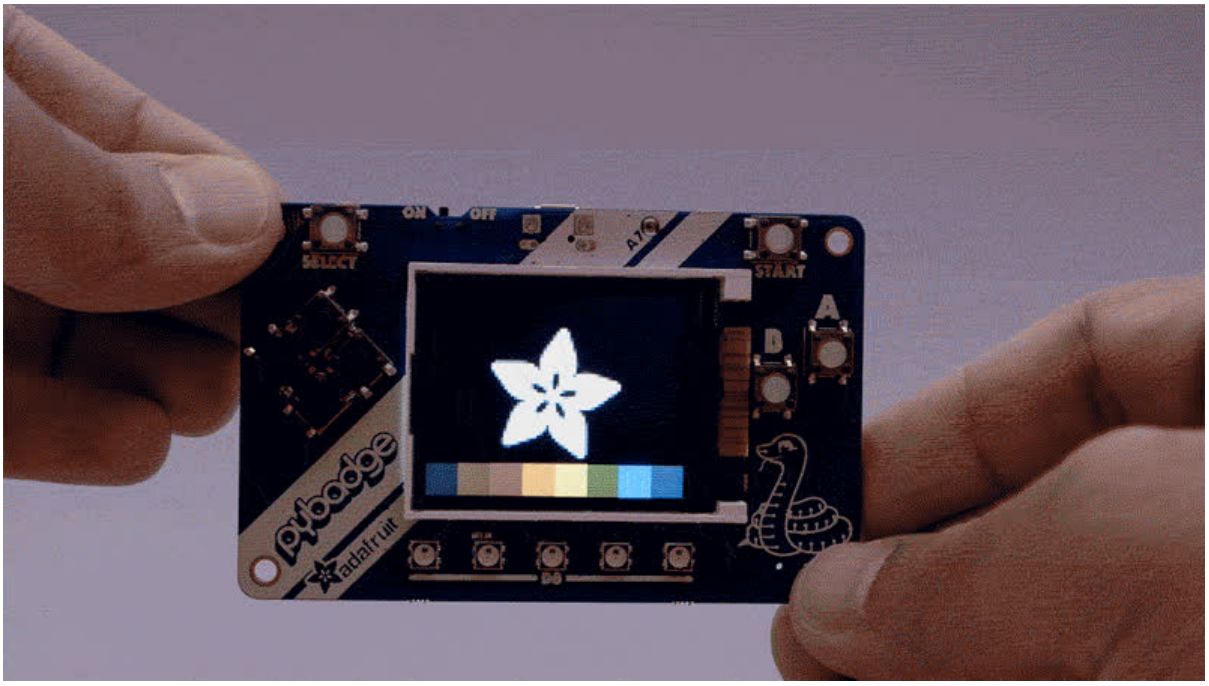
PixelDust Digital Sand Demos for Arcada

Created by lady ada



https://learn.adafruit.com/pixeldust-digital-sand-demos-for-arcada

Last updated on 2024-06-03 02:48:11 PM EDT

Table of Contents

Overview

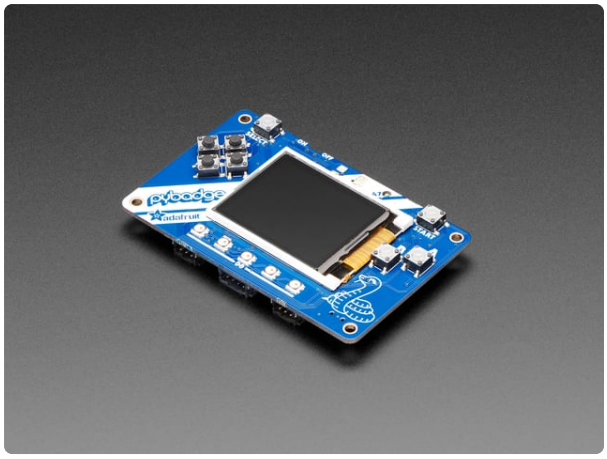
PyGamer and PyBadge have built in accelerometers - which you can use in your games or demos to make nifty motion-activated effects. In this mini guide we'll show you some examples of PaintYourDragon's PixelDust (https://adafru.it/E-p) library but for Arcada boards

Supported Hardware

You'll need a board with Adafruit Arcada (https://adafru.it/EF5) support + an accelerometer such as...

Adafruit PyGamer Starter Kit
Please note: you may get a royal blue or purple case with your starter kit (they're both lovely colors)What fits in your pocket, is fully Open...
https://www.adafruit.com/product/4277

Adafruit PyGamer for MakeCode Arcade, CircuitPython or Arduino
What fits in your pocket, is fully Open Source, and can run CircuitPython, MakeCode Arcade or Arduino games you write yourself? That's right, it's the Adafruit...
https://www.adafruit.com/product/4242



Adafruit PyBadge for MakeCode Arcade, CircuitPython, or Arduino

What's the size of a credit card and can run CircuitPython, MakeCode Arcade or Arduino? That's right, its the Adafruit PyBadge! We wanted to see how much we...

https://www.adafruit.com/product/4200

USB cable - USB A to Micro-B

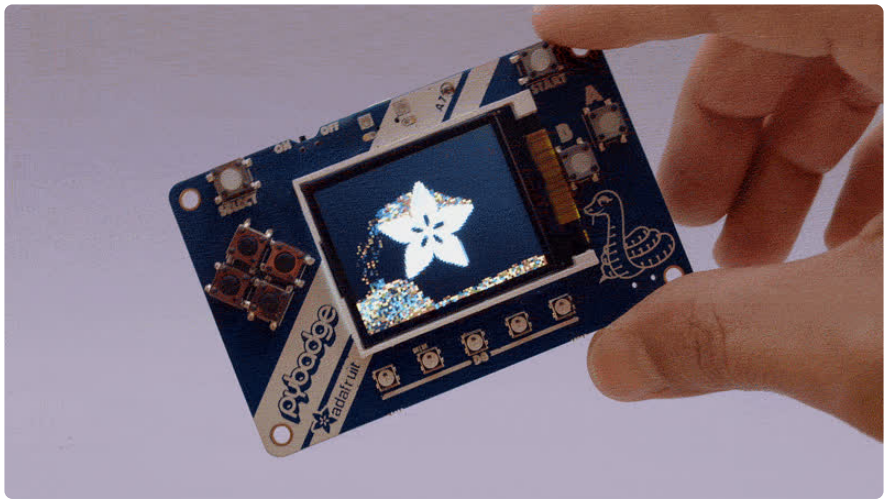
This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...

https://www.adafruit.com/product/592

Compile & Upload

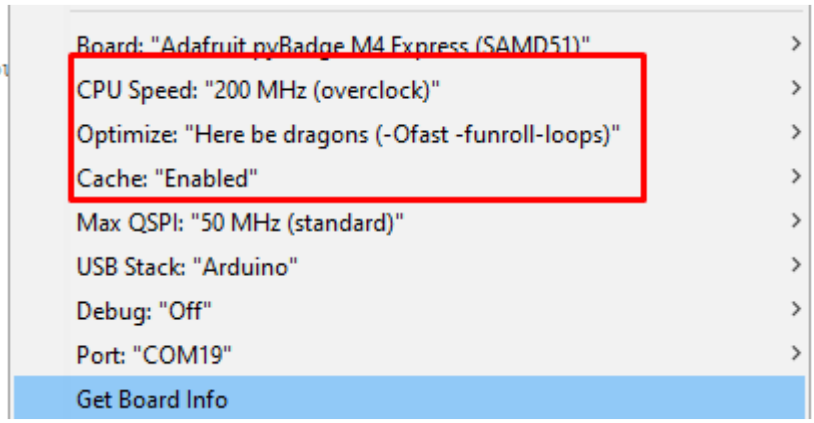
Start by following your board's guide on installing Arduino IDE, and support for the board you have. Then install the Adafruit Arcada libraries (https://adafru.it/EUk) (there's a lot of em!)

Also install the Adafruit PixelDust (https://adafru.it/E-p) library



Compilation Settings

As you get to a few thousand particles, you'll want to speed up your board as much as possible. Compile with ultra-speed settings such as `200MHz overclock`, `-Ofast optimizations` and `Cache enabled`.



Runtime Settings

There's not a lot of things you can adjust but here's a few common ones:

```
#define CHUNKY_SAND
```

If this is at the top of the code, it will make each particle a 2x2 pixel rather than a single pixel. this makes it look a little better, but you can't fit as many particles on the screen.

```
#define N_FLAKES 2000
```

How many particles to simulate. More look cooler but too many and it slows down! 1000-2000 seems to be a good number, especially with `CHUNKY_SAND` turned on.

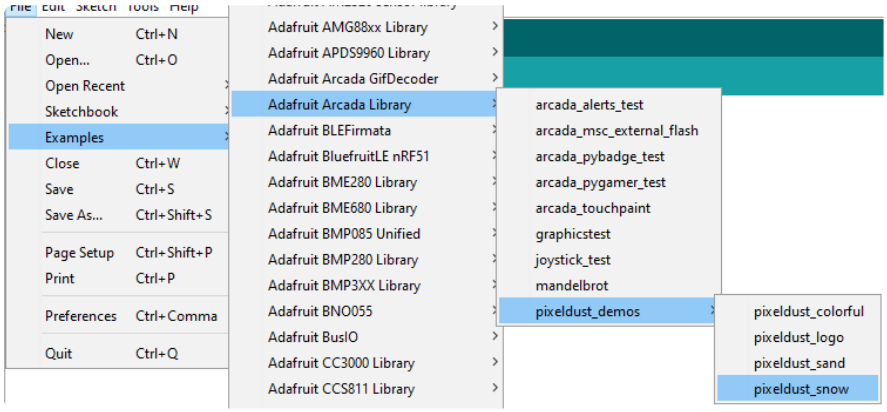
On this line in the loop:

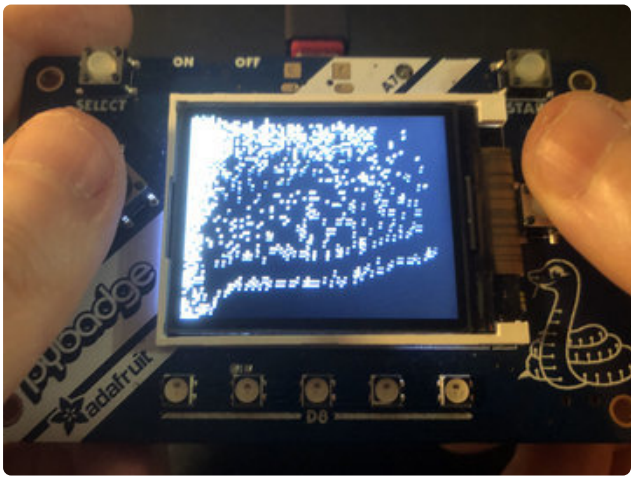
```
pixeldust->iterate(xx * 3000.0, yy * 3000.0, zz * 3000.0);
```

The multiplier affects the 'gravity' of the pixels. Larger numbers will drag the pixels down faster, smaller numbers will make the pixels float a little more.

Snow Demo

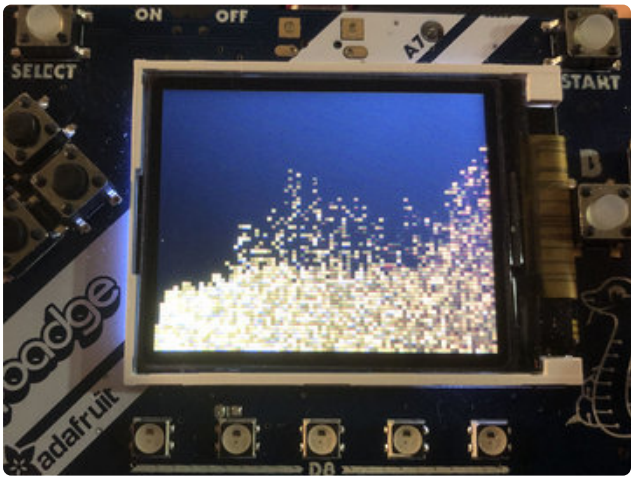
Start with the **pixeldust_demos->pixeldust_snow** example, its the simplest demo - each pixel is the same white color.





Upload and enjoy!

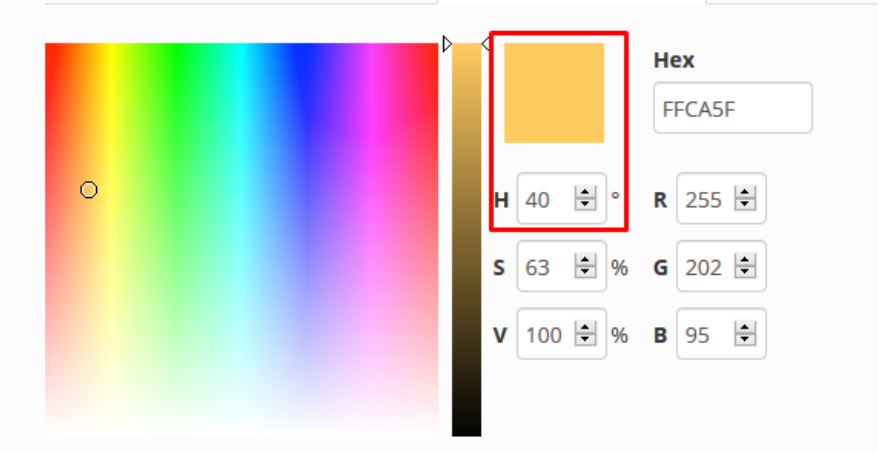
Sand Demo



This demo builds on the snow version to add speckled yellow colors to each particle, to create a sand-effect!

You can add color to each pixel by creating a new array of 16-bit colors as we do in this demo with the creation of `uint16_t *flake_colors;` and then later `flake_colors = (uint16_t *)malloc(N_FLAKES *2)`;

Then you can assign the colors, we'll use an HSV picker to find a hue we think is sandy...



And randomly assign brightness/saturations so we get a range of sandy colors!

```
  // randomize colors
  for (int i=0; i&lt; N_FLAKES; i++) {
    flake_colors[i] =
      __builtin_bswap16(arcada.ColorHSV565(40, // Hue (sandy)
                                           random(50, 100),  // saturation
                                           random(50, 100))); // brightness
  }
```

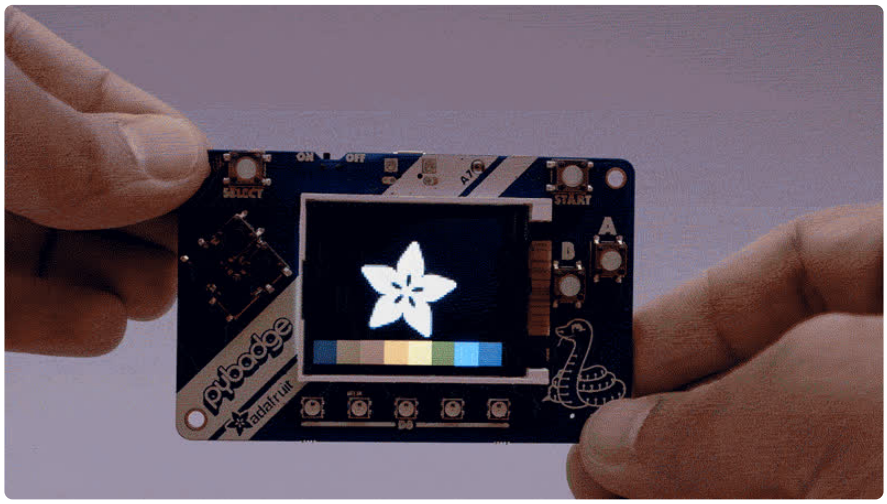
Note we use `__builtin_bswap16` on each color word. That's because we later use DMA to write out all the pixels and we need to have the high/low bytes of color swapped in order for it to run as fast as possible (its a weird effect of TFT DMA on Arduino)

Later on, when we draw the pixels, we'll look up the corresponding color before we draw the color to our framebuffer:

```
  for(int i=0; i&lt;N_FLAKES; i++) {
    pixeldust-&gt;getPosition(i, &amp;x, &amp;y);
    //Serial.printf("(%d, %d) -&gt; %d\n", x, y, x * width + y);
    uint16_t flakeColor = flake_colors[i];
#ifdef CHUNKY_SAND
    framebuffer[2*y * width + 2*x] = flakeColor;
    framebuffer[2*y * width + 2*x+1] = flakeColor;
    framebuffer[(2*y+1) * width + 2*x] = flakeColor;
    framebuffer[(2*y+1) * width + 2*x + 1] = flakeColor;
#else
    framebuffer[y * width + x] = flakeColor;
#endif
  }
```

Logo Demo

Finally, the most advanced of the demos adds a logo 'obstacle' both as an image and a 'mask' that tells PixelDust where not to let pixels go. This makes for lovely effects as particles slide around.



For the logo, which is 8-bit grayscale and stored in the header, you can use a tool like this that will take an image and convert it into a header file (https://adafru.it/E-q).

Like the sand demo we will store a color for each particle. Except this time instead of randomly placing them on the display, they are put into boxes along the bottom of the screen:

```
// Set up initial sand coordinates, in 8x8 blocks
int n = 0;
for(int i=0; i&lt;N_COLORS; i++) {
  int xx = i * play_width / N_COLORS;
  int yy =  play_height - BOX_HEIGHT;
  for(int y=0; y&lt;BOX_HEIGHT; y++) {
    for(int x=0; x&lt;play_width / N_COLORS; x++) {
      //Serial.printf("#%d -&gt; (%d, %d)\n", n,  xx + x, yy + y);
      pixeldust-&gt;setPosition(n++, xx + x, yy + y);
    }
  }
}
```

Since the chunks of particle divide up into 8 colors, we dont have to store the color of each one, we know that the index of the particle, divided by 8, gives the color index. Notes we have to `bswap16` the color here like we did before.

```
colors[0] = arcada.color565(40 , 40, 40);   // Dark Gray
colors[1] = arcada.color565(120, 79, 23);   // Brown
colors[2] = arcada.color565(228,  3,  3);   // Red
colors[3] = arcada.color565(255,140,  0);   // Orange
colors[4] = arcada.color565(255,237,  0);   // Yellow
colors[5] = arcada.color565(  0,128, 38);   // Green
```

```
  colors[6] = arcada.color565(  0, 77,255);   // Blue
  colors[7] = arcada.color565(117,  7,135); // Purple
  for (int i=0; i&lt;N_COLORS; i++) {
    colors[i] = __builtin_bswap16(colors[i]);  // we swap the colors here to speed
up DMA
  }
```

Then before we draw all the particles, we also have to draw the logo:

```
  int logo_origin_x = (width  - 2*LOGO_WIDTH ) / 2;
  int logo_origin_y = (height - 2*LOGO_HEIGHT ) / 2;
  // Draw the logo atop the background...
  for(int yl=0; yl&lt;LOGO_HEIGHT; yl++) {
    for(int xl=0; xl&lt;LOGO_WIDTH; xl++) {
      uint16_t c =
          __builtin_bswap16(arcada.color565(logo_gray[yl][xl], logo_gray[yl][xl],
logo_gray[yl][xl]));
      x = logo_origin_x + 2*xl;
      y = logo_origin_y + 2*yl;

      framebuffer[y * width + x] = c;
      framebuffer[y * width + x+1] = c;
      framebuffer[(y+1) * width + x] = c;
      framebuffer[(y+1) * width + x+1] = c;
    }
  }
```