



Shake Away 2020 with MatrixPortal

Created by Phillip Burgess



Last updated on 2020-12-30 05:49:03 PM EST

Guide Contents

Guide Contents	2
Overview	3

Overview

Shake away those 2020 blues with a MatrixPortal project that will help you celebrate the end of one year, and the bright-and-shiny promise of a new year!

Here are a couple of pre-compiled .UF2 files. To install, connect MatrixPortal to computer with USB cable. Double-tap the reset button, wait for **MATRIXBOOT** drive to appear and drag one of the .UF2 files to **MATRIXBOOT**.

Both start with “2020” on the matrix. With a quick shake, it’s reduced to pixel sand.

The first of these UF2s then switches to the glimmering “2021” after a few seconds of sand (**six seconds** from the shake, if you’re counting down to midnight). Tap the reset button to start over.

<https://adafru.it/PCd>

<https://adafru.it/PCd>

The other UF2 instead returns to 2020 and watches for another shake:

<https://adafru.it/PCe>

<https://adafru.it/PCe>



If you'd like to check out the code, [it's based off of the Protomatter PixelDust demo \(https://adafru.it/PCg\)](https://adafru.it/PCg), where we use a bitmap of the 2020/2021 text for the pixels.

Click **Download Project Zip** in the embed below to get both the Arduino sketch and the extra files that store the bitmap

```
#include <Adafruit_LIS3DH.h> // For accelerometer
#include <Adafruit_PixelDust.h> // For simulation
#include <Adafruit_Protomatter.h> // For LED matrix
#include "2020.h" // 2020 bitmap data
#include "2021.h" // 2021 bitmap data

bool show_2021 = true;

#define SHAKE_ACCEL_G 2.9 // Force (in Gs) to trigger shake
#define SHAKE_ACCEL_MS2 (SHAKE_ACCEL_G * 9.8) // Convert to m/s^2
#define SHAKE_ACCEL_SQ (SHAKE_ACCEL_MS2 * SHAKE_ACCEL_MS2) // Avoid sqrt() in accel check
#define SHAKE_EVENTS 30 // Number of accel readings to trigger sand
```

```

#define SHAKE_PERIOD    2000                // Period (in ms) when SHAKE_EVENTS must happen
#define SAND_TIME      6000                // Time (in ms) to run simulation before restarting

uint8_t rgbPins[] = {7, 8, 9, 10, 11, 12};
uint8_t addrPins[] = {17, 18, 19, 20};
uint8_t clockPin = 14;
uint8_t latchPin = 15;
uint8_t oePin = 16;

// 64x32 pixel matrix, 6-bit depth
Adafruit_Protomatter matrix(
  64, 6, 1, rgbPins, 4, addrPins, clockPin, latchPin, oePin, true);

Adafruit_LIS3DH accel = Adafruit_LIS3DH(); // Accelerometer

#define MAX_FPS 60 // Maximum redraw rate, frames/second
uint32_t prevTime = 0; // For frames-per-second throttle
uint16_t n_grains = 0; // Number of sand grains (counted on startup)
Adafruit_PixelDust *sand; // Sand object (allocated in setup())

// Error handler used by setup()
void err(int x) {
  uint8_t i;
  pinMode(LED_BUILTIN, OUTPUT); // Using onboard LED
  for(i=1;i++) { // Loop forever...
    digitalWrite(LED_BUILTIN, i & 1); // LED on/off blink to alert user
    delay(x);
  }
}

// SETUP - RUNS ONCE AT PROGRAM START -----

void setup(void) {
  uint8_t i, j, bytes;
  Serial.begin(115200);
  //while (!Serial);

  ProtomatterStatus status = matrix.begin();
  Serial.printf("Protomatter begin() status: %d\n", status);

  // Count number of 'on' pixels (sand grains) in bitmap_2020
  for (int i=0; i<sizeof(bitmap_2020); i++) {
    for (int b=0; b<8; b++) {
      if (bitmap_2020[i] & (1 << b)) {
        n_grains++;
      }
    }
  }
  Serial.printf("Bitmap has %d grains\n", n_grains);

  // Allocate sand object based on matrix size and bitmap 'on' pixels
  sand = new Adafruit_PixelDust(matrix.width(), matrix.height(), n_grains, 1);
  if (!sand->begin()) {
    Serial.println("Couldn't start sand");
    err(1000); // Slow blink = malloc error
  }

  if (!accel.begin(0x19)) {
    Serial.println("Couldn't find accelerometer");
    err(250); // Fast blink = I2C error
  }
  Serial.println("Accelerometer OK");
  accel.setRange(LIS3DH_RANGE_8_G);

```

```

    accel.setRange(L15501_RANGE_G_0);
}

void loop() {
  Serial.print("Tick");
  uint16_t sandColor = show_2021 ? 0xF800 : 0xFFFF; // Red or white

  // Set initial sand pixel positions and draw initial matrix state
  sand->clear();
  matrix.fillScreen(0);
  int grain = 0, pixel = 0; // Sand grain and pixel indices
  for (int i=0; i<sizeof(bitmap_2020); i++) {
    for (int b=0; b<8; b++, pixel++) {
      if (bitmap_2020[i] & (1 << (7-b))) {
        int x = pixel % BITMAP_WIDTH;
        int y = pixel / BITMAP_WIDTH;
        //Serial.printf("Set pixel %d @ (%d, %d)\n", grain, x, y);
        sand->setPosition(grain++, x, y);
        matrix.drawPixel(x, y, sandColor);
      }
    }
  }
  matrix.show();

  // Wait for shake
  uint32_t first_event_time = millis() - SHAKE_PERIOD * 2, last_event_time = first_event_time;
  uint8_t num_events = 0;
  sensors_event_t event;
  for (;;) {
    uint32_t t = millis(); // Current time
    accel.getEvent(&event);
    float mag2 = event.acceleration.x * event.acceleration.x +
                event.acceleration.y * event.acceleration.y +
                event.acceleration.z * event.acceleration.z;
    if (mag2 >= SHAKE_ACCEL_SQ) { // Accel exceeds shake threshold
      if ((t - last_event_time) > SHAKE_PERIOD) { // Long time since last event?
        first_event_time = t; // Start of new count
        num_events = 1;
      } else if ((t - first_event_time) < SHAKE_PERIOD) { // Still in shake interval?
        if (++num_events >= SHAKE_EVENTS) { // Enough events?
          break;
        }
      }
      last_event_time = t;
    }
  }

  // Run sand simulation for a few seconds
  uint32_t elapsed, sandStartTime = millis();

  while((elapsed = (millis() - sandStartTime)) < SAND_TIME) {

    // Limit the animation frame rate to MAX_FPS.
    uint32_t t;
    while(((t = micros()) - prevTime) < (1000000L / MAX_FPS));
    prevTime = t;

    // Read accelerometer...
    sensors_event_t event;
    accel.getEvent(&event);

    // Run one frame of the simulation
    sand->iterate(event.acceleration.x * 1024, event.acceleration.y * 1024, event.acceleration.z * 1024);
  }
}

```

```

if (elapsed > SAND_TIME * 3 / 4) {
  float scale = 1.0 - (float)(elapsed - (SAND_TIME * 3 / 4)) / (float)(SAND_TIME / 4);
  if (scale < 0.0) scale = 0.0;
  else if (scale > 1.0) scale = 1.0;
  scale = pow(scale, 2.6);
  uint16_t rb = (int)(31.0 * scale + 0.5);
  uint16_t g = (int)(63.0 * scale + 0.5);
  if (show_2021)
    sandColor = (rb * 0b100000000000); // Just show red
  else
    sandColor = (rb * 0b100000000001) + (g << 5);
}

// Update pixel data in LED driver
matrix.fillScreen(0);
dimension_t x, y;
for(int i=0; i<n_grains ; i++) {
  sand->getPosition(i, &x, &y);
  matrix.drawPixel(x, y, sandColor);
}
matrix.show();
}

// If the show_2021 flag is set, don't return to 2020 shake detect,
// instead switch to sparkly '2021' display forever (reset to start over)
if (show_2021) {
  uint16_t frame = 0;
  matrix.fillScreen(0);
  for(;;) {
    int pixel = 0;
    for (int i=0; i<sizeof(bitmap_2021); i++) {
      for (int b=0; b<8; b++, pixel++) {
        if (bitmap_2021[i] & (1 << (7-b))) {
          int x = pixel % BITMAP_WIDTH;
          int y = pixel / BITMAP_WIDTH;
          matrix.drawPixel(x, y, (random() & 1) ? (((x - y + frame) / 8) & 1) ? 0xFFFF : 0x001F : 0);
        }
      }
    }
    matrix.show();
    delay(18);
    frame++;
  }
}
}
}

```

