DOI: 10.14714/CP97.1675

Frequently Updated Maps and their Public Display

Michael P. Peterson (he/him)
University of Nebraska at Omaha
mpeterson@unomaha.edu

Paul Hunt (he/him)
University of Nebraska at Omaha
phunt@unomaha.edu

The display of maps on computer monitors in a public setting can be used to emphasize their value in conveying spatial patterns. For thematic maps, by removing the possibility for interaction, more attention can be focused on the mapped distributions. Maps that lend themselves best for public display are those that are frequently updated, such as weather maps. Other types of frequently updated maps (FUMs) include those of earthquakes, air pollution, and health conditions, such as the spread of a virus. These types of maps are increasingly provided through the internet in an interactive format, making the resultant maps less suited for public display. Described here are available maps that could be displayed in a public setting, and a method to make maps for quick display based on available data. A series of these maps can then be assembled and shown in a continuous loop. The display of maps for the public can be implemented using the low-cost, Raspberry Pi computer. Maps that are suitable for public display, instructions for implementation, and the required code are available at: ***maps.unomaha.community/FUMPD/About.html****.*

INTRODUCTION

A fundamental change in cartography since the beginning of the 1990s has been the incorporation of interaction (Peterson 1995) into many maps. However, this trend has not always been for the best. For example, the most common type of interaction that is implemented for thematic maps is the ability for users to view individual data values. As a result, the map is reduced to little more than a spatial table, often to the detriment of its main purpose of communicating spatial patterns.

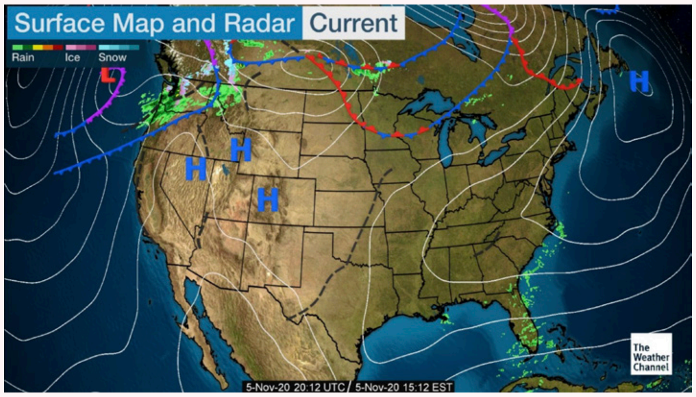
One way around this problem is to offer non-interactive maps more often, to promote better pattern recognition. These may be shown as part of automated map displays in public settings, such as on monitors in lobbies, airports, offices, etc. Frequently updated maps (FUMs; Peterson and Wendel 2003) work well here, as they generate the most interest in a public setting. However, the trend toward interactive map design means that there are few ready-made examples to display in such settings. In this article, we introduce (1) an inventory of FUMs that could be part of such an automated display; (2) methods for converting existing interactive maps to non-interactive, but frequently updated maps; (3) how to make maps directly from the underlying data; and (4) a low-cost solution for setting up a public display.

The FUMforPD website (**maps.unomaha.community/FUMPD/About.html**) accompanies this article. It assembles many of the currently available FUMs and shows how maps for public display can be made from the underlying data. It also includes code for the display of a series of maps in a continuous loop and shows how a low-cost computer can be used for such displays.

EXAMPLES OF FREQUENTLY UPDATED MAPS

Weather maps are a prime example of FUMs, with some being refreshed every 30 minutes. Initially, these maps were once only available through the internet as static images. Today, most weather websites have incorporated interaction into their display. **Weather.com**, for example, offers multi-scale pannable maps (MSP), implemented through Mapbox, to display radar imagery and storm paths. A limited number of static maps, more suitable for a public display, are also available under a "Classic Weather Maps" link (Figure 1). Likewise, the Sat24.com website offers similar products for Europe (Figure 2).

While weather maps have been available through the internet for many years, a new kind of FUM became necessary in early 2020: maps that depicted the spread of COVID-19. Figure 3 shows two maps from ourworldindata.org/coronavirus that depict cases and deaths over a two-week period. These maps are updated daily and made available in SVG, a vector format suitable for display through a web browser.


Surface Map and Radar Current
Rain Ice Snow
The Weather Channel
5-Nov-20 20:12 UTC / 5-Nov-20 15:12 EST

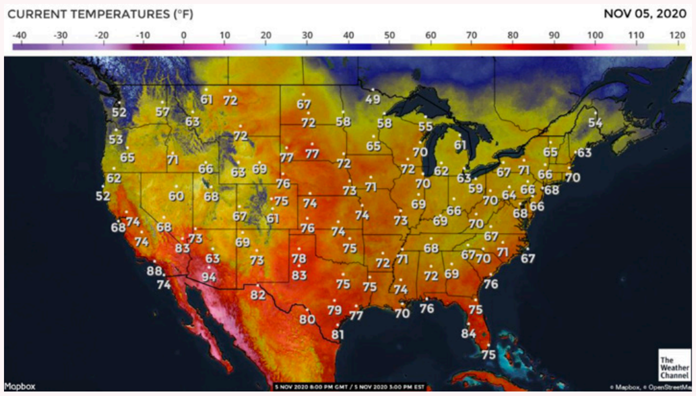
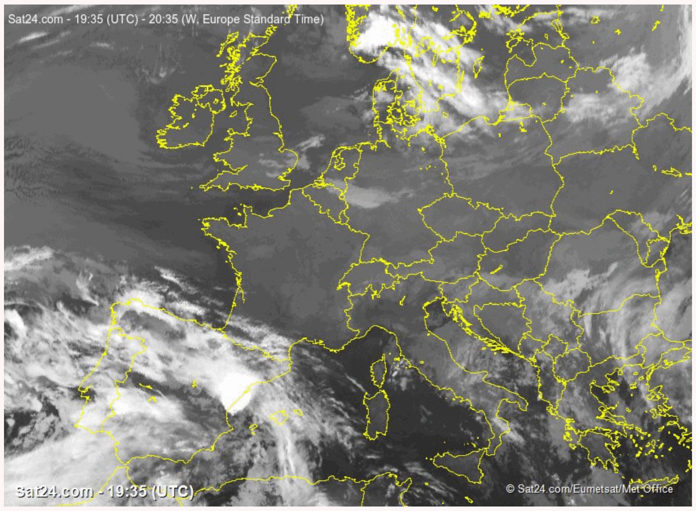


Figure 1. Two examples of "classic" maps from Weather.com.


Sat24.com - 19:35 (UTC) - 20:35 (W. Europe Standard Time)
Sat24.com - 19:35 (UTC)
© Sat24.com/Eumetsat/Met Office

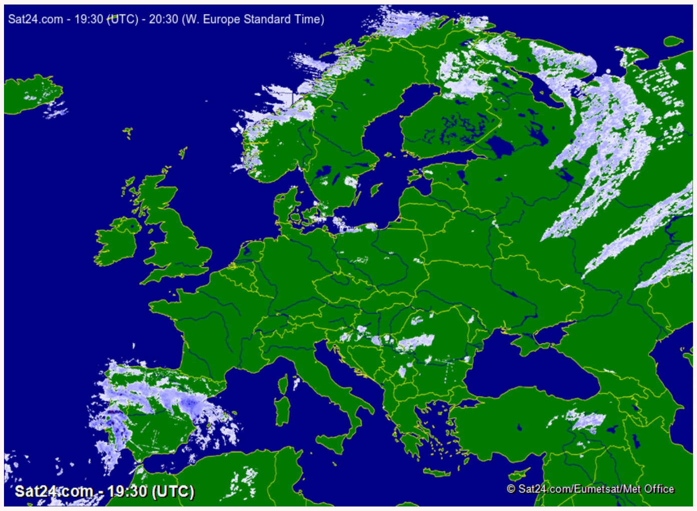
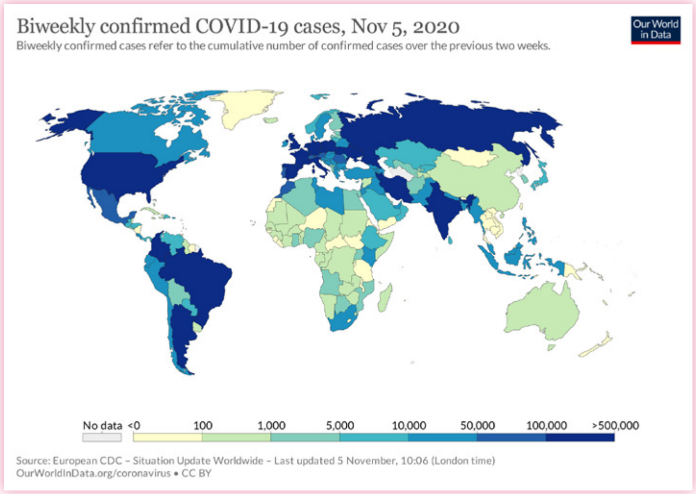

Sat24.com - 19:30 (UTC) - 20:30 (W. Europe Standard Time)
Sat24.com - 19:30 (UTC)
© Sat24.com/Eumetsat/Met Office

Figure 2. Single frames from two animated GIFs from Sat24.com, showing cloud cover (left) and rainfall (right).


Biweekly confirmed COVID-19 cases, Nov 5, 2020
Biweekly confirmed cases refer to the cumulative number of confirmed cases over the previous two weeks.
Our World in Data
No data <0 100 1,000 5,000 10,000 50,000 100,000 >500,000
Source: European CDC – Situation Update Worldwide – Last updated 5 November, 10:06 (London time)
OurWorldInData.org/coronavirus • CC BY

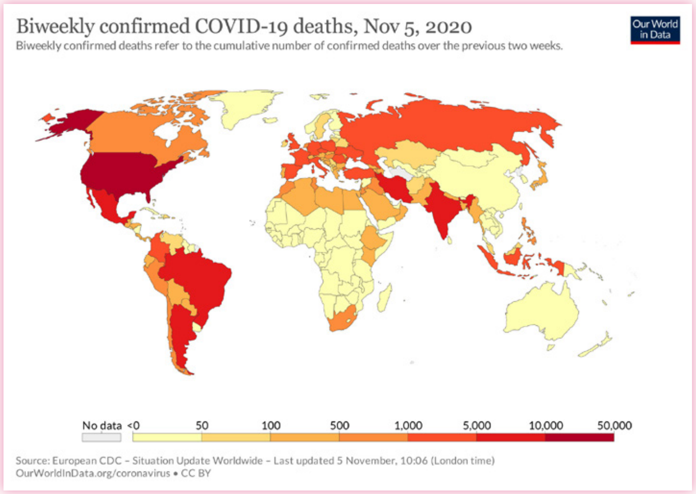
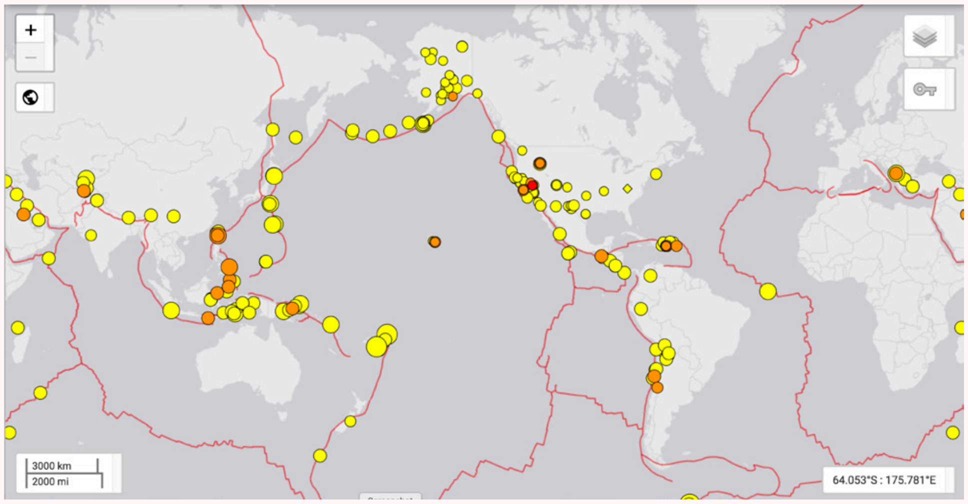

Biweekly confirmed COVID-19 deaths, Nov 5, 2020
Biweekly confirmed deaths refer to the cumulative number of confirmed deaths over the previous two weeks.
Our World in Data
No data <0 50 100 500 1,000 5,000 10,000 50,000
Source: European CDC – Situation Update Worldwide – Last updated 5 November, 10:06 (London time)
OurWorldInData.org/coronavirus • CC BY

Figure 3. Two maps of COVID-19 from ourworldindata.org/coronavirus, showing biweekly cases and deaths.

Current maps of earthquakes are provided by the United States Geological Survey (USGS) as MSP maps, made with the Leaflet API, atop a basemap from OpenStreetMap (earthquake.usgs.gov). The underlying data is also available, provided in a JSON format. Maps specifically designed for non-interactive display can be created from this data using a variety of online tools. Figure 4 shows both the map provided by USGS and a map made from USGS-supplied, real-time data using the Google Maps API. The latter was saved in the PNG format for display.

Frequently updated maps of air pollution are also available. PurpleAir operates a citizen network of over 16,000 sensors that measure particulate pollution, both PM2.5 and PM10 (purpleair.com). They offer a web map that uses the Mapbox API to display data from these sensors (purpleair.com/map), as well as the underlying data (purpleair.com/data.json). Both the PurpleAir map, and a map that we prepared based upon their data, can be seen in Figure 5. Our map was built using the Google Maps



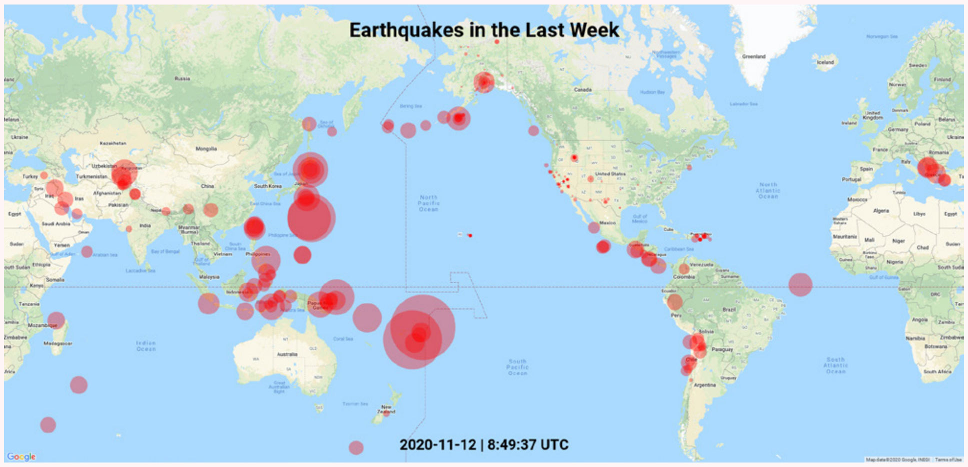
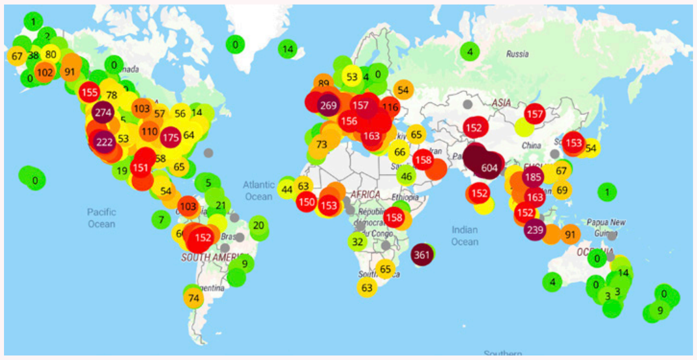


Figure 4. Two maps of earthquakes for a 7-day period. The top, interactive, map, including plate boundaries, is from the USGS (orange and red circles indicate more recent earthquakes). The bottom map is based on the same USGS data feed. It was made with the Google Maps API and saved in the PNG format.



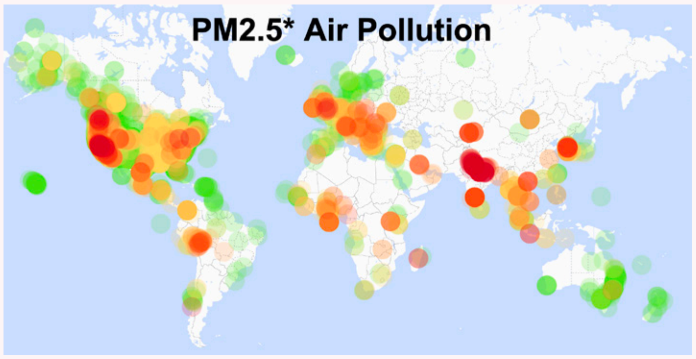


Figure 5. Two maps of PM2.5 particulate air pollution. The map on the left is from PurpleAir, while the map on the right was produced by us based on PurpleAir data. The map on the right is not interactive and does not include numbers within each circle. The green forest cover shading has also been removed from the basemap. While the green/yellow/red color scheme may create an accessibility issue for those with atypical color vision, the scheme was used to mimic the original PurpleAir map.

API, and has been simplified, as compared to PurpleAir's, by removing the green shading for forest cover to improve the visibility of the green symbols. The symbols have also been made partly transparent.

CONVERSION OF MAPS TO IMAGES

Now that we have some idea of the available maps and datasets, we'll walk through how we showed them on a public display. One option would have been to simply point a web browser at one of the interactive maps described above and let it refresh at regular intervals. However, many interactive maps (such as those for earthquakes and air pollution) require considerable time to display, due to the quantity of data being loaded. Even with a fast internet connection, the loading time can sometimes exceed 30 seconds. The recognition of spatial patterns on maps is generally thought to occur more quickly. As a result, the slow display interferes with spatial pattern recognition, particularly in a public setting where people may have limited time to examine the map. The fastest image display times are achieved with pre-made, suitably-sized maps in either the PNG, JPG, or GIF formats. Vector SVG files can also be displayed quickly depending on their complexity.

To make interactive maps more suitable for display, we made use of Puppeteer, a Node JavaScript API, that can take static screenshots of the interactive maps. The API can extract data from websites, a process called web scraping (Leitner 2019). Puppeteer installs the Chromium browser that works with the API. Example 1 shows a segment of Puppeteer code that will capture a screenshot. All of our code is available at the FUMforPD website (**maps.unomaha.community/FUMPD/About.html**). Puppeteer sets an initial page size to 800×600px, and the size of the image can be customized with `Page.setViewport()`. The size of both the earthquake and air pollution maps seen in this article was 2100×1000 pixels.

To simplify the maintenance and infrastructure needs of Puppeteer, an Amazon Web Services (AWS) serverless architecture was implemented. AWS Lambda (**aws.amazon.com/lambda**) is a serverless compute service that runs code in response to events and manages the underlying compute resources. AWS Lambda extends other AWS services by, for example, creating back-end services such as an HTTP request. The server-side JavaScript screenshot code is executed here with AWS Lambda.

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');
  await page.screenshot({path: 'example.png'});

  await browser.close();
})();
```

Example 1. Puppeteer code that works with the Chromium browser to capture a screenshot of a web page.

We used AWS Simple Storage Solutions (S3), an object storage service, to store the static screenshot images generated by Puppeteer. AWS S3 is essentially storage for the internet. It can be used to store and retrieve differing amounts of data from anywhere on the web. Within the S3 service, users create "Buckets," which can be thought of as folders that are used to store the object-based files. In our case, the PNG screenshots were stored in a bucket called "peterson-screenshots," producing the following URL for one of the png files: **peterson-screenshots.s3.amazonaws.com/2_Day.png**. This particular PNG file depicts earthquakes for the past day using the Google Maps API, and is updated every hour. The USGS supplies JSON files for earthquakes in four time intervals: past hour, past 24 hours, past week, and past month. We added some code that incorporates a time-stamp into the bottom of each map (see Example 2).

For showing air pollution, we downloaded data from PurpleAir and produced multiple maps at different scales for multiple regions of the world. To limit server load, PurpleAir allows their data to be accessed once every 30 seconds. Because multiple maps are being made, the full dataset is temporarily downloaded to an S3 bucket, and our code makes maps based on this local file. The data is then overwritten an hour later when the next series of maps are made.

```
// Put timestamp on the map
        var myTitle = document.createElement('h1');
        myTitle.style.font = 'bold 28px arial';
        myTitle.style.color = 'Black';

        var today = new Date();
        var date = today.getFullYear()+'-'+(today.getMonth()+1)+'-'+today.getDate();
        var time = today.getHours() + ":" + AddZero(today.getMinutes()) +
                ":" + AddZero(today.getSeconds());
        dateTime=date+' | '+time+' UTC';

        myTitle.innerHTML = dateTime;
        var myTextDiv = document.createElement('div');
        myTextDiv.appendChild(myTitle);
        map.controls[google.maps.ControlPosition.BOTTOM_CENTER].push(myTextDiv);
```

Example 2. JavaScript code that adds the time stamp to each map. The time for AWS Lambda servers corresponds to UTC.

The PNG maps of earthquakes and air pollution are updated hourly. This is achieved by establishing an AWS CloudWatch EventBridge (**docs.aws.amazon.com/eventbridge**). The EventBridge framework allows for scheduling the execution of the cloud-based screenshot function in AWS Lambda.

AUTOMATED DISPLAY OF IMAGES

After generating images, the next step is to automatically display them. Again, all the code is available through the **FUMforPD website**. There, you can find HTML/JavaScript that can be run on Google Chrome on Windows or Mac OS. Example 3 presents a part of both the HTML and JavaScript code where the images are referenced. In our example, the size of the images has been adjusted for a 1920×1200 pixel monitor, with some made larger to zoom-in on an area of interest.

To display other images, download the code from any of the examples on the FUMforPD website and change the addresses of the images shown in both the HTML and JavaScript parts of the code as shown in Example 3. The width of the images can be matched to the monitor using a value of 100% or zoomed to a particular area of interest by using a larger number for the width or height of the image.

```
<div class="fadein">
    <img src="https://s.w-
    x.co/staticmaps/uksat_1280x720.jpg"
    width=2200 id = "myImage1"/>
    <img
    src="https://api.sat24.com/animated/EU
    /infraPolair/3/" width=2200 id =
    "myImage2"/>
    <img
    src="https://api.sat24.com/animated/GB
    /rainTMC/3/" width=1920 id =
    "myImage3"/>
</div>
```

```
myImageElement1.src = 'https://s.w-
x.co/staticmaps/uksat_1280x720.jpg?
rand=' + Math.random()
myImageElement2.src =
'https://api.sat24.com/animated/EU/
infraPolair/3/?rand=' +
Math.random()
myImageElement3.src =
'https://api.sat24.com/animated/GB/
rainTMC/3/?rand=' + Math.random()
```

Example 3. Code segment for the automated display of images. The addresses of the images are entered in both the HTML (top) and JavaScript (bottom) parts of the code.

Once we have the code to show the maps, the next step is to display them seamlessly. We recommend the Google Chrome browser, which can be automatically started in full-screen mode. Full-screen mode is set in the view menu (Figure 6). In the browser's Preferences, you can also set it to "Continue where you left off," on starting the browser, meaning that Chrome will open in full-screen mode, displaying the last webpage viewed in the browser.

The last step to automating this process is to ensure that Chrome is launched on startup. In this way, if the computer is reset, Chrome will automatically begin again, go to full screen, and return to the map display. On a Mac, this is done using the **Login Items** tab under the **Users and Groups System Preference**. In Windows, in the **Start** button, select **Settings** > **Apps** > **Startup** and select Google Chrome to run at **startup.**

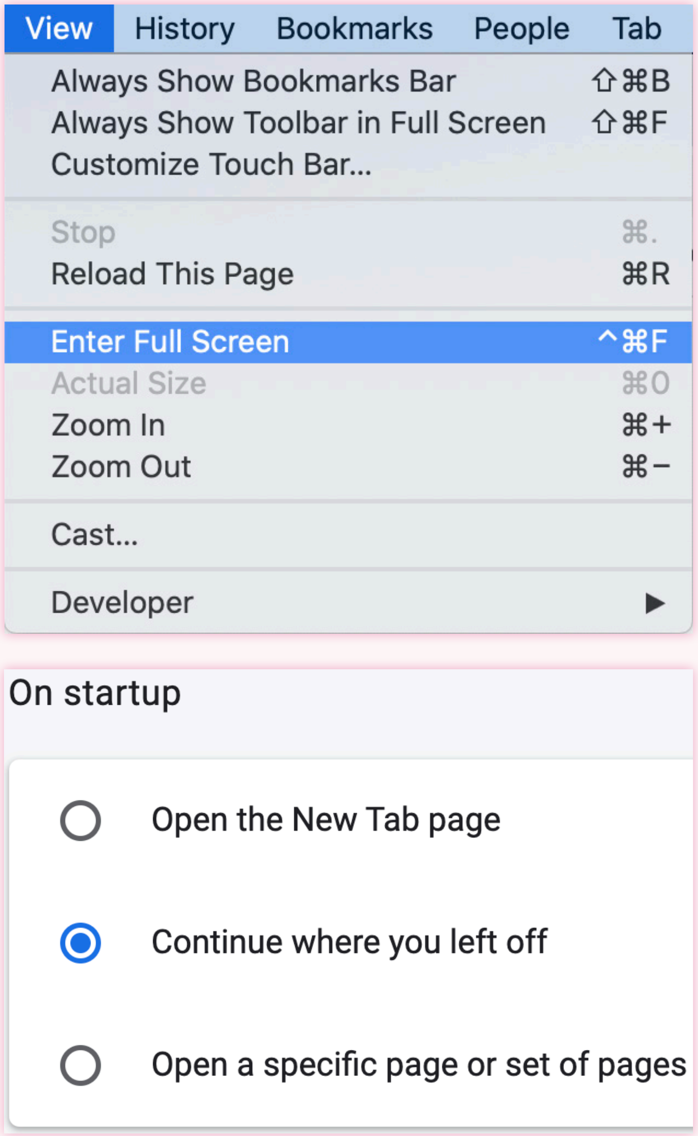
Depending on your settings, an electronic timer may be necessary to control when the display is active. While some computers can be set to turn on and off at specified times, this control may not extend to the monitor. Whether or not an electronic timer is used, the computer should be set to start automatically after a power outage.

Figure 7 shows a series of six automated weather and map displays at the University of Nebraska at Omaha. Initially based on older computers, primarily Macs, most displays have been converted to use a Raspberry Pi.

DISPLAY WITH A RASPBERRY PI

To reliably display maps in a public setting, we found the Raspberry Pi (RPi) computer to be a low-cost and low-power solution. An RPi consists of the computer (see Figure 8), a power supply, and an SD memory card. While these are all sold separately, they can be acquired as a kit for about US $65. A plastic case for the computer can also be purchased separately.

When purchased, the RPi does not include any software—even an operating system. All required software can be freely downloaded from raspberrypi.org. In a process called "flashing," the RPi OS is loaded onto the micro-SD card, which acts like a hard drive for the computer. The flashing process requires a Windows, Mac, or Ubuntu computer, running a flashing program such as Etcher (balena.io/etcher). Once the SD card has been flashed with the operating system, it can be physically installed



On startup

- ○ Open the New Tab page
- ● Continue where you left off
- ○ Open a specific page or set of pages

Figure 6. Google Chrome options for full-screen display and continuing with the current settings after re-start, including the full-screen display.

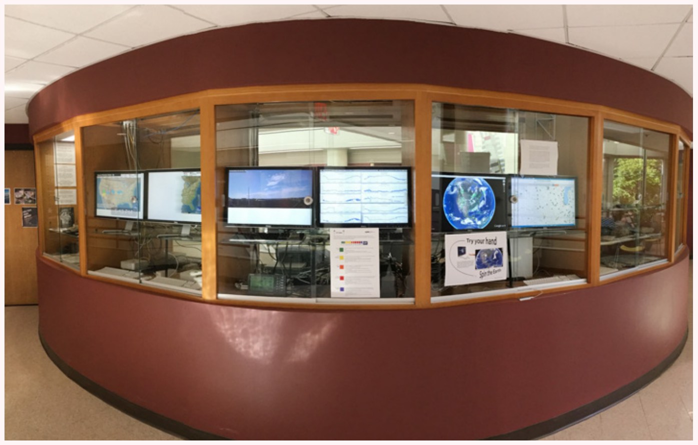


Figure 7. An automated map display consisting of six computers at the University of Nebraska at Omaha.

in the RPi. The computer then boots to LXDE (Lightweight X11 Desktop Environment), a Windows-like desktop environment that includes standard menus and dialogs.

Some setup is necessary to configure the RPi to open a webpage at startup that displays the maps. This webpage should be configured to continuously cycle through a series of maps, and it's best if it webpage resides on a separate server so that it can be more easily modified.

Figure 8. The $65 Raspberry Pi Model 4 computer. The device can be easily programmed to automatically display maps in a public setting.

The RPi uses the Chromium browser, and much like with the Chrome browser above, this needs to be set to enter full-screen mode on startup. Although the RPi can be set to start and shut down at specified times, it does not have this control over the monitor. A simple external timer must be used to remove power to both the computer and monitor during hours when it is not being viewed. The RPi and monitor will power up automatically when power is restored. The Chromium browser is set to start automatically, retrieving the webpage from a server that displays the maps.

Let's start setting things up by entering this command in LXTerminal:

```
raspi-config
```

and select "boot to desktop," as well as your local time zone. Next, to configure the RPi's wifi access, enter:

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.
 conf

network={

ssid="YOUR_NETWORK_NAME"

psk="YOUR_WIFI_PASSWORD"

}
```

Save changes and quit (ctrl-o, ctrl-x). The Wifi address must be straightforward. For example, it is not a simple matter to enter the necessary parameters for an eduroam connection.

The unclutter app hides the mouse pointer, for a cleaner display. It is installed using this command:

```
sudo apt-get install unclutter
```

The screen is forced to stay on (not sleep) and the Chromium browser is automatically started by editing this file:

```
sudo nano /home/pi/.config/lxsession/LXDE-pi/
 autostart
```

If this file does not open, try the alternate location for the autostart file:

```
sudo nano /etc/xdg/lxsession/LXDE-pi/autostart
```

Then add these lines at the end of the file:

```
@xset s off

@xset -dpms

@xset s 0 0

@xset s noblank

@xset s noexpose

@xset dpms 0 0 0

@chromium-browser --noerrdialogs --incognito
 --autoplay-policy=no-user-gesture-required
 --check-for-update-interval=1 --simulate-
 critical-update --kiosk https://URL of the
 web page that cycles through the maps
```

The computer can then be rebooted by entering:

```
sudo reboot
```

or by temporarily removing power.

When the RPi restarts, the images specified in the HTML link will be displayed in full-screen mode. They will automatically update and continue displaying until the computer is shut down or power is removed.

A good exercise in working with the RPi is to create a DAKboard (**dakboard.com**), a customizable web-based display. The instructions found at **blog.dakboard.com/diy-wall-display** explain the process of setting up a Raspberry Pi to create a personalized wall display via DAKboard.

CONCLUSION

The display of thematic maps in a public setting encourages spatial pattern recognition. However, many interactive maps available online load slowly enough that they can interfere with this pattern recognition. While watching a map being drawn on the screen might attract attention, interactive maps are rarely updated in a way that encourages the recognition of broad patterns. If pattern recognition occurs quickly—as is generally thought to be the case—any delay in creating the pattern can only be detrimental to pattern recognition. This would be especially be true in a public setting where the map viewer may not take the time to wait for the map to be completed.

Converting interactive maps to images for quick display may be the best solution to further spatial pattern recognition. The specific method we demonstrate here involves the hourly updating of earthquake and air pollution data, while incorporating design elements to promote pattern recognition. The maps available through the FUMforPD website will continue to be updated as long as the necessary infrastructure remains in place. We hope that others will create similar displays of frequently updated data, and expand on this method to show new datasets, and alternative visualizations of existing datasets. For example, PurpleAir data could be processed to display day-to-day *changes* in air pollution.

While we encourage the public display of maps and the use of the low-cost Raspberry Pi computer, we recognize that these displays will only be only viewed by a limited number of people. We believe, however, that displays of maps in a public setting will encourage better map design, towards more thoughtful visualizations that promote spatial pattern recognition. The public display of maps that we advocate should encourage better thematic map design for *all* applications.

REFERENCES

Leitner, Christopher. 2019. “How to Scrape Amazon Product Information with Nodejs & Puppeteer.” *Zenscrape*. Accessed June 24, 2021. **https://zenscrape.com/how-to-scrape-amazon-product-information-with-nodejs-and-puppeteer/**

Peterson, Michael P., and Jochen Wendel. 2003. “The Animation of Frequently Updated Maps.” *Annual Meeting of the Association of American Geographers*, New Orleans, LA, Febuary 28–March 3, 2003.

Peterson, Michael P. 1995. *Interactive and Animated Cartography*. Upper Saddle River, NJ: Prentice-Hall.