CHAPTER 12

Code/Art Approaches to Data Visualization

J. J. Sylvia IV

This chapter introduces a code/art approach to data visualization. Though coding has received increasingly greater amounts of attention within the field of Digital Humanities, it has primarily focused on more traditional types of visualizations such as charts and graphs. However, the **iteration** that is possible through **generative design** affords more artistic approaches. Casey Reas and Chandler McWilliams[1] claim that particular programming languages afford specific opportunities. In much the same way that a carpenter would select particular tools and a particular wood depending on the project, programming languages require similar careful selection. I use p5.js as the language of choice for this chapter because it is a free, open source language specifically designed for beginners, artists, and educators. It is a very accessible language that allows one to quickly begin creating dynamic content on the screen, even with very little previous programming experience.

[1] Casey Reas and Chandler McWilliams, *Form Code: In Design, Art, and Architecture* (New York: Princeton Architecturel Press, 2010).

J. J. Sylvia IV (✉)
Fitchburg State University, Fitchburg, MA, USA
e-mail: jsylvia3@fitchburgstate.edu

l. levenberg et al. (eds.), *Research Methods for the Digital Humanities*,
https://doi.org/10.1007/978-3-319-96713-4_12

My project, *Aperveillance: Watching with Open Data*, serves as the case study for this chapter. Through a web-based visualization, I take an artistic approach to data visualization that is afforded by iteration and generative design. Whereas a more traditional approach to data visualization would seek to answer questions or create clearer explanations through the process of visualization, my work instead aims to provoke further questions about the societal tensions between surveillance and privacy. My research question in light of this goal is: How can the artistic combination of disparate sources of publicly available data make clear the societal tensions between safety, surveillance, and privacy? A Digital Humanities approach to this research question can make these tensions more immediately present and real to a viewer by drawing them into the activity of surveillance as a participant.

Ethical Issues

Ethical issues related to code/art data visualizations mirror many of the larger ethical concerns of working with data in general. First, the data that one uses should be both reliable and verifiable to the furthest extent possible. In other words, any data used for a visualization should come from a trustworthy and accurate source, otherwise the visualization that is created from the data will be similarly misleading or inaccurate. This is especially important for artistic visualizations because it is often harder for viewers of the visualization to verify the accuracy of the data than it would be in a chart or graph. Because artistic visualizations offer different ways of seeing data, the underlying data can often be harder to quantify. While this other way of seeing data is an important benefit of artistic visualizations, it should also underscore the importance of beginning with accurate data.

Determining the reliability of data is often a difficult process, however, one indicator for quality is the source of the data. For example, data from open government sources, such as the City of Raleigh—Open Data initiative that I used for my project, is likely to be more reliable than data from unknown or unofficial sources on the web. Although you will be creating artistic projects, these visualizations can be persuasive, thus it remains important to use valid data for that persuasion.

The other major ethical issue concerns protecting the privacy of those who may be represented in the data. Although all reliable data sources

will have anonymized their data before making it accessible to the public, the vast amount of data that now exists means that it is increasingly likely that multiple data sources can be combined together in order to identify seemingly anonymized individuals. For example, in 2006, America Online released 20 million anonymized search queries from that year. Despite this anonymization, reporters from *The New York Times* were able to cross reference the queries and identify a particular user's search queries.[2] Additionally, if you collect information, even from public sources, there may be ethical problems with displaying that data in new ways. For example, a 2010 Harvard Facebook study and a 2016 release of data collected from public profiles on the OKCupid dating website both raised ethical concerns. Even though the information in these studies was publicly accessible, the arrangement of the data in a new way can draw extra attention to the data and put it a new context that was never intended by the user who originally posted it.[3] In other words, consent was not given for this particular use of the data. Collecting, formatting, and then using data in new ways changes its context and, if your data contains information about individual people, this is an issue for which extra care should be taken.

When academic researchers collect their own data, the context in which the data will be used must be approved by the participants before the data gathering process can begin. The protection and restriction of confidential data must be clearly elaborated and approved by review boards at universities. However, private corporations do not face the same ethical scrutiny and users often agree to allow carte-blanche use of their data by simply accepting the Terms of Service when they sign up for an account with a particular site or application. Users frequently approve these terms without reading them thoroughly, or at all. Because of these vastly different approaches to collecting and using data, academics must pay particular care to ethics when using data that they have not personally collected, especially if such data has been collected by a private company.

[2]Viktor Mayer-Schönberger and Kenneth Cukier, *Big Data: A Revolution That Will Transform How We Live, Work, and Think*, First Mariner Books edition (Boston: Mariner Books, Houghton Mifflin Harcourt, 2014).

[3]Michael Zimmer, "OkCupid Study Reveals the Perils of Big-Data Science," *Wired*, May 14, 2016. Accessed April 12, 2017, https://www.wired.com/2016/05/okcupid-study-reveals-perils-big-data-science/.

Why Use a Code/Art Approach to Visualization?

In teaching p5.js as part of both classes and Digital Humanities related workshops, I have developed a better understanding of the strengths and weaknesses of a creative coding approach. First, the artistic nature of p5.js affords the opportunity to create data visualizations that offer a high level of affective impact, which is the often emotional or visceral transformation that one experiences in reaction to a new experience. Viewers are more often engaged with an artistic visualization than they would be with a chart or graph. All data visualizations are engaging, but creative code approaches afford the opportunity to open spaces for questions by allowing the viewer to be an active participant in the process. In other words, the meaning of a particular visualization may be somewhat ambiguous in a way that asks more questions than it answers. Viewers may be able to ask their own questions about ethics or meaning based on their reaction to a visualization.

Said another way, traditional visualizations aim to render complicated data in a way that it is easily understandable at a glance. A bar chart, for example, can make fluctuations in income across a span of years quickly recognizable in a way that looking through a larger spreadsheet of numbers would not. A bar chart, though commonly visualized using computer software, is still something that could easily be created by hand. I could fairly quickly recreate a bar chart using a piece of paper and a pen. Coding approaches to visualization unlock techniques that would be either impossible or highly infeasible by hand. Both the generative nature of coding—the ability of the computer to run through thousands of iterations of code per second—and its ability to use multimedia inputs highlight the strengths of this approach.

The generative nature of coding offers the possibility to create elaborate artistic pieces with comparatively little code. For example, the following code, adapted from Casey Reas, Lauren McCarthy, and Ben Fry's introductory text on p5.js, will create a series of kinked lines (Fig. 12.1):

Fig. 12.1 Example of kinked lines

```
for (var i = 3; i < 400; i += 3) {
          line(i, 0, i + i/2, 200);
               line (i + i/2, 200, i*1.2, 400);
          }
```

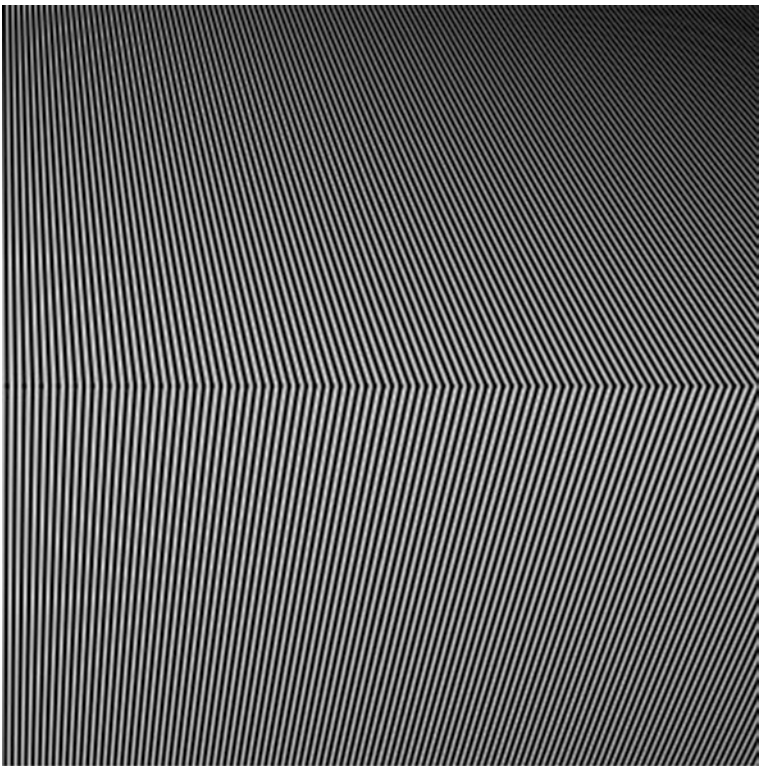

Fig. 12.2 Generative design

These few lines of code can generate over one hundred lines on the screen in a fraction of second (see Fig. 12.2). This generative approach can also be used with various types of data, as a way to influence the iterations that occur, or as data that is displayed to viewers. In addition to data, p5.js facilitates access to images, video, and sound, allowing multimedia to be integrated into and manipulated by the generative process.

This generative design approach can be used in a variety of ways. On the one hand, it might be used to answer difficult design questions. For example, in 2004, NASA engineers used generative **evolutionary algorithms** to design an effective X-band antenna: "Whereas the current practice of designing antennas by hand is severely limited because it is both time and labor intensive and requires a significant amount of domain knowledge, evolutionary algorithms can be used to search the

design space and automatically find novel antenna designs that are more effective than would otherwise be developed".[4] In other words, generative design was used to create random antennae that were then tested for effectiveness. The most effective ones were allowed to evolve further until a final most efficient design was derived.

On the other hand, a humanities-based project might raise more questions than it answers. For example, the 2006 exhibit, *Being not truthful*, by Ralph Ammer and Stefan Sagmeister, provokes questions about truth and vulnerability:

> "Being not truthful works against me" is part of a list in Stefan Sagmeister's diary entitled "Things I have learned in my life so far." In the installation *Being not truthful*, this maxim is woven into a virtual spiderweb, which rips if a viewer's shadow touches it; then, bit by bit, the web reconstructs itself. This fragile construction serves as a metaphor for the vulnerability of Sagmeister's aphorism and the effort required to follow it, raising questions about the nature of truth and the value of sincerity.[5]

The movement of those who are viewing the exhibit is incorporated into the project itself, highlighting the possibility of drawing on information from the surrounding environment as part of the project. This type of project does not give us any particular answers—no best design emerges and we do not straightforwardly learn anything about the truth. Instead, we are provoked to ask even more questions about the nature of truth.

One weakness of this approach is that it can actually be more difficult to use than traditional data visualization tools when one is trying to get the same types of results as traditional data visualizations. For instance, when I first introduce these creative visualization projects in classes and workshops, many students immediately think about a variety of charts or graphs that they might create about their text. It is quite possible to do this kind of visualization through a programming language like p5.js, but other tools such as Tableau are able to do this much easier with a wider variety of analytic tools quickly available. Many people who have done past data visualization work tend to default to the same types of questions they would attempt to answer with those tools. For example, I have had

[4] Gregory Hornby et al., "Automated Antenna Design with Evolutionary Algorithms," *American Institute of Aeronautics and Astronautics* (2006), https://doi.org/10.2514/6.2006-7242.

[5] Hartmut Bohnacker et al., *Generative Design: Visualize, Program, and Create with Processing* (New York: Princeton Architectural Press, 2012).

multiple students who wanted to analyze the frequency of occurrence for a particular word or words within a text. While this is possible to do with a creative coding tool such as p5.js, there are easier ways of doing this.

Therefore, the major weakness and strength of this approach are related. A creative coding approach is strongest when it is used to create outside-the-box visualizations based on repetition and iteration. However, to use the tools in this way, one must begin by thinking about data sets in a way that differs from traditional data visualization approaches. Creative approaches are less about analysis and more about exploration. They are less about answering questions and more about generating new questions.

Step-by-Step Guide

To provide a guide to using a code art approach I will draw examples from my research project, *Aperveillance: Watching with Open Data* (aperveillance.com). This project contributes to the significant body of scholarship on surveillance in the field of Communication and Media Studies, by raising questions about the types of watching that are now becoming increasingly possible with open data. Thus, I coined the term *aperveillance*[6] for my project as a way to distinguish it from more traditional methods of surveillance, which are typically only available to powerful entities like governments or large corporations. My project instead focuses on the type of surveillance that is available to everyone.

Michel Foucault is widely considered a founding figure in the area of surveillance studies, though the area only coalesced as a definite area of study after the 9/11 attack in the United States and the passing of the Patriot Act in its wake.[7] Gary Marx[8] offers a thorough overview of the field, showing how surveillance has shifted as technology has changed. For example, he cites the way that technology has impacted the scale and ease of

[6] [ap-er-**vay**-lans] *n.* derives from the Latin "*aper*" meaning "open", and "*veiler*" meaning "to watch." In the context of this project, aperveillance means "open watching," or a form of watching with open data.

[7] Michel Foucault, *Discipline and Punish: The Birth of the Prison*, 2nd ed. (New York: Vintage Books, [1975] 1995); Peter Monaghan, "Watching the Watchers," *The Chronicle of Higher Education*, 2006; Kirstie Ball, Kevin Haggerty, and David Lyon, eds., *Routledge Handbook of Surveillance Studies* 1. Paperback ed. Routledge International Handbooks (London: Routledge, 2012).

[8] Gary T. Marx, "Surveillance Studies," in *International Encyclopedia of the Social & Behavioral Sciences* (2015): 733–741, https://doi.org/10.1016/b978-0-08-097086-8.64025-4

routine strategic surveillance. An example of strategic surveillance would be Lantern Laws that required slaves to carry a lantern with them while they were out at night. Technologies such as big data, GPS, ubiquitous cameras, and DNA analysis, on the other hand, have enabled strategic surveillance to become pervasive and less visible, facilitating involuntary participation. For example, a citizen may not even be aware of the extensive system of cameras set up in their city. A primary concern of surveillance is its conflict with expectations of privacy.[9] An emerging practice of sousveillance, or the use of personal devices for recording an activity by a participant who is involved in the activity, considers how one might use devices to monitor those who are doing the surveillance. Aperveillance aims to open a field of questions somewhere between surveillance and sousveillance. What kind of watching is possible with data and images that are publically available?

My first step consisted of using this context to develop a clear research question: How can the artistic combination of disparate sources of publicly available data make clear the societal tensions between safety, surveillance, and privacy? Based on this research question, I began to brainstorm ways to address this question by visualizing data and drawing viewers into the process as an active participant.

This web-based visualization project uses webcam images from Raleigh and the larger North Carolina area that are publicly available. When a viewer loads the project on a personal computer, the code will check to see if there is an accessible camera associated with the computer, such as the camera that comes built-in on most Apple computers. If there is a camera, the code will use that to capture a still image of the viewer that will be randomly included among the other images that have been pulled from the webcams in the larger community. It also uses Raleigh's open crime data to randomly display information about the previous day's crimes, juxtaposed on top of the webcam images. This is intended to provoke questions about the type of watching that we as citizens can do with open data on the web. The inclusion of the image from the viewer's own camera is often disorienting, because it is not immediately obvious that the viewers themselves have been included in the visualization. It also creates confusion about whether their image is viewable to everyone who is accessing the website.

[9] Helen Nissenbaum, "Protecting Privacy in an Information Age: The Problem of Privacy in Public," *Law and Philosophy* 17, no. 5/6 (November 1998): 559, https://doi.org/10.2307/3505189.

Fig. 12.3 Aperveillance on display in Hunt Library's code+art gallery at North Carolina State University

Using examples from the programming language p5.js, I demonstrate the potential for using code and algorithms to generate data-based visualizations through **repetition**, **modulation**, **transformation**, and **parameterization**. This work exists at the intersections of **data visualization**, **Media Studies**, **big data**, and **information aesthetics** (Fig. 12.3).

This project uses p5.js, a language based on Processing and created to be accessible to beginners, educators, and artists. The processing language was originally developed for visual artists, with the goal of making coding accessible to a wider audience and promoting software and visual literacy. p5.js builds upon this foundation and extends these goals. However, because it is a JavaScript based language, it is native to the web, and enables the creation of interactive elements directly within a web browser. One important caveat to note is that p5.js is a living and

evolving language, which means that some of the functionality may change over time, or most likely, new features and functions will continue to be added to the core functions. These changes should not be cause for alarm, however, as the core functionality has remained the same, and all the updates have been aimed at making development easier and more accessible.

Getting Ready to Code

The first step to creating a code/art visualization is downloading the files needed to begin coding. One example of the changes associated with p5.js is the method used to write the code itself. Early in the development of the language, there was a self-contained editor available from the website. This development environment has been depreciated and instead, users are encouraged to use other software developed to help with coding. There are a wide variety of free and open source options for coding editors, but one widely used example is Brackets (http://www.brackets.io).

Next, you will need to download the p5.js core files, which enable its functionality. To do this, visit http://www.p5js.org and click on the Download tab. You should download the Complete Library, as this contains not only the files that offer the most robust functionality of p5.js, but also a helpful example project. It is worth noting that there is a very early alpha version of an online editor available for p5.js, accessible at https://alpha.editor.p5js.org. This may be the best option for those who are most interested in a quick overview of the language and smaller, simpler projects. At this point in time, larger and more complex projects are most suited for development using the downloaded complete library. After downloading and unzipping the complete library, the following files should be accessible (Fig. 12.4):

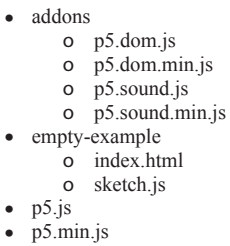

Fig. 12.4 p5.js core files

The p5.js file is the main file that contains the core of the language. p5.sound.js and p5.dom.js add extended functionality. The versions that contain "min" in the file name are compressed to speed up loading times.

In order to view the output of the code, use a web browser to access the index.html file in the empty-example folder. It is set up by default to load the p5 related JavaScript files, but will appear blank if opened in a browser. It is helpful to create a master copy of this file so that you can use it as a template for future projects.

Opening the index.html file in a code editor such as Brackets offers a glimpse of how the p5.js files are loaded and accessed:

The file below (Fig. 12.5) loads the JavaScript files that contain the core functionality of p5.js as well as the sketch.js file, which will contain the code created for new projects. One detail worth noting here is that the first three JavaScript files are being loaded directly from the website cloudflare.com. However, it is also possible to load the files locally, in the same way that the sketch.js file is being loaded. You would simply need to move the other JavaScript files into the same folder as your index.html file.

The last thing to think about is how the public will access your project. Currently, the index.html and sketch.js files are only stored locally on your own machine. This is preferable while you are creating and coding your project, but you will want others to be able to access them more widely when you are done. The simplest option, if you already have web hosting, is to upload these files to your existing server and then access them through the associated URL for your hosting.

Finally, if you open sketch.js using your editor of preference, there is a bare-bones p5.js project ready for creation. The limitations imposed by the format of a book chapter make it infeasible to offer a full introduction to either programming in general or p5.js in particular, but there are many great resources available, including the tutorials on the p5.js website at: https://p5js.org/tutorials/ and the book authored by the creators of the language: *Getting Started with p5.js: Making Interactive Graphics in JavaScript and Processing (Make)* by McCarthy, Reas, and Fry (2015). The remainder of this guide covers how I accessed the data I used and a general overview of how it was visualized using p5.js.

```
<!DOCTYPE html>
<html>
  <head>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.5.11/p5.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.5.11/addons/p5.dom.min.js"></sc
ript>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.5.11/addons/p5.sound.min.js"></
script>
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>
  <body>
    <script src="sketch.js"></script>
  </body>
</html>
```

Fig. 12.5 Example of index.html file

Accessing Data

This project arose from an assignment created for students in my course *Big Data and the Rhetoric of Information*, in which I challenged students to create a variety of visualizations based on open source data. I was also creating my own project along with them as an example, and I knew I wanted to use local data available from the *City of Raleigh—Open Data* project at https://data.raleighnc.gov.

One of the intriguing visualizations on the site is a map of the previous day's crime incidents. In thinking about visualizations, I also reflected on the large number of webcams in our area, many set up to monitor traffic and/or weather patterns. Although I originally considered trying to match webcams to the locations of crimes, I quickly realized that despite the prevalence of webcams, this would be an infrequent pairing. In addition, simply juxtaposing crime data with the variety of webcams publicly available created its own set of intriguing questions about the type of surveillance the average citizen can perform (Fig. 12.6).

The next step in the project was to access the data associated with the visualization, so I could use it within p5.js. The export function of the website offered several different formats in which I could access the raw data. These included Comma Separated Value (CSV), CSV for Excel, CSV for Excel (Europe), JSON, RDF, RSS, TSV for Excel, and XML. I selected the CSV option and downloaded it directly to my computer, but I also realized that if I right-clicked the link, I could copy the URL

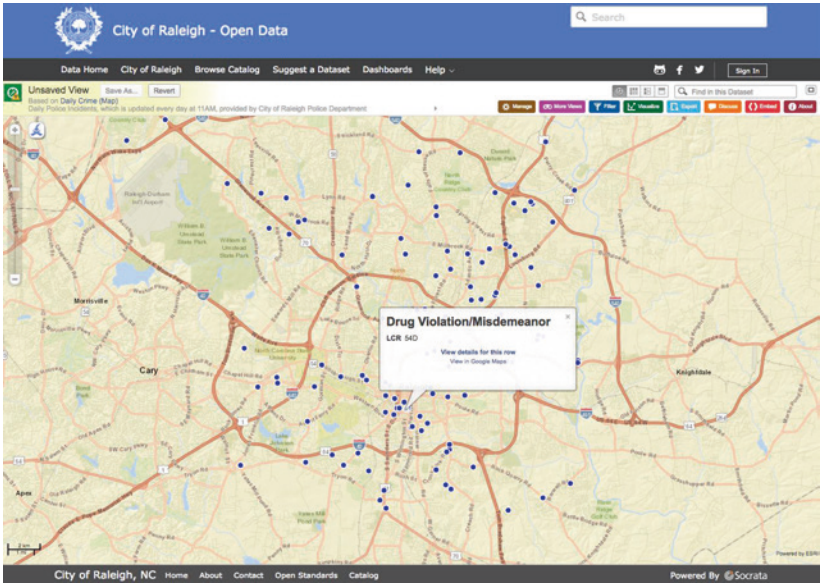

Fig. 12.6 The previous day's crime incidents—city of Raleigh

where the CSV file was accessible on the web. This file is updated once per day at 11 am.

I tried to access and load this CSV file directly within p5.js, but this created an error, likely because server settings prevented this type of access. This can sometimes be a problem with images and other files as well. To work around this, I created an Automater script through macOS that would automatically download the file locally.[10] This was done with the following Automater actions:

1. **Get Specified Finder Items**: here I selected the location of the current version of the file, which resided in my Dropbox folder.
2. **Move Finder Items to Trash**: This deleted the current version identified in the previous step.

[10]For more information on Automator for MacOS, see https://support.apple.com/guide/automator/welcome/mac.

3. **Get Specified URLs**: Here I used the URL that I copied above: https://data.raleighnc.gov/ap/views/guyh-emm5/rows.csv?accessType=Download.
4. **Download URLs**: Using this action, I specified that the file should be saved into the same folder as the previous version of the file I had deleted in step 2.

This Automater task deletes the previous day's file and then downloads the most recent version of the crime data, titled Daily_Police_Incidents.csv. I then used the Calendar application on my Mac to automatically run this script once per day. I then created similar scripts for the webcam images I wanted to access. I set this script to run every 15 minutes rather than once a day.

There are a few important caveats about this method. First, downloading these files and saving them to my computer ensures that my application will still run even if the webcam images are temporarily unavailable on the internet. Second, you may have noticed above that I was downloading these images directly to my Dropbox (http://www.dropbox.com). I have long used my Dropbox as a web server, and hosted my Aperveillance project directly from there. However, this ability to use Dropbox as web server was restricted to paid subscribers only in 2016 and is slated to be eliminated entirely at the end of 2017. Therefore, in the future, it will be necessary to take the additional step of uploading these files to a web server. With the CSV and webcam JPG images now saved to the same folder, I can create the code for my visualization.

Coding

The image in Fig. 12.7 represents the final version of the project as seen in the browser. There are several images from webcams displayed in a grid on the screen, including an image from the viewer's own local camera on the bottom right. Each of these images is placed randomly and updated and moved approximately every 15 seconds. In addition, the randomly placed white boxes contain information about crimes that were committed in Raleigh, including the date and time of the crime, the GPS coordinates where the incident occurred, and the type of incident. The grid is automatically scaled to fit any screen and if the device used to access the page does not have or does not allow access to the local

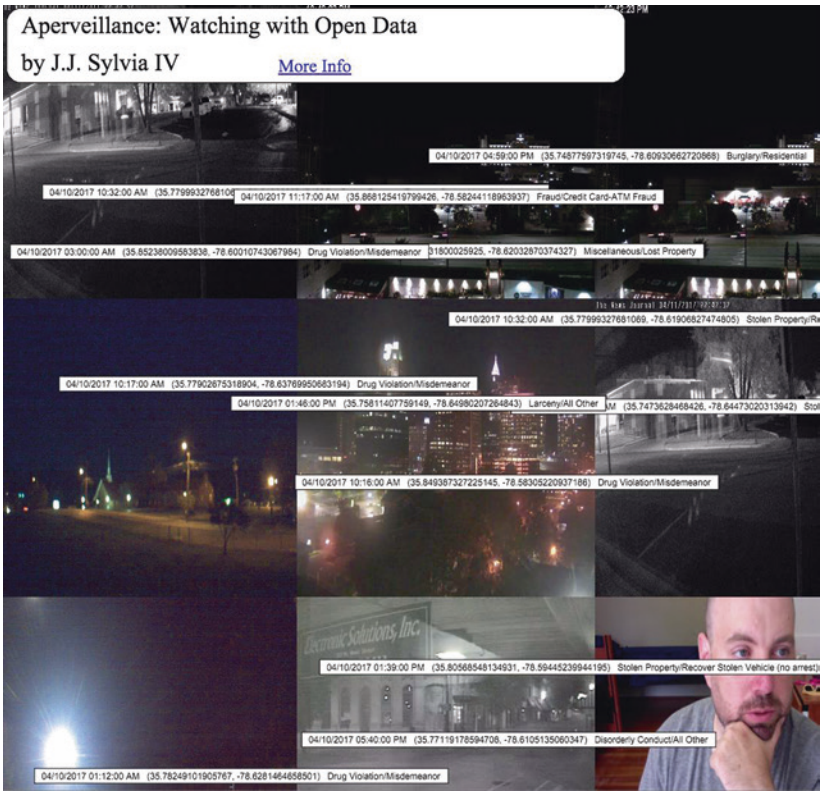

Fig. 12.7 Aperveillance final

webcam, that image will be replaced by another random webcam image from Raleigh.

The full p5.js code needed to create this project is listed below. Though it may look complex for those with little or no coding background, the amount of code required to create this project is quite small. Following the code, I will briefly walk you through how each section works.

The code begins by defining the **variables** (especially **arrays**) that will be used as part of the program. Next, the preload function loads all of the media and other files that will be used within the code.

```
var img = [];
var r3 = 0;
var stats;
var dt = [];
var loc = [];
var desc = [];
var howMany = 0;
var mic;


function preload() {
  img[0] = loadImage("raleigh-640x480.jpg");
  img[1] = loadImage("apex_skycam-640x480.jpg");
  img[2] = loadImage("airport_skycam-640x480.jpg");
  img[3] = loadImage("auburn_tower_ptz-640x480.jpg");
  img[3] = loadImage("wral_gardens-640x480.jpg");
  img[4] = loadImage("durham_skycam-640x480.jpg");
  img[5] = loadImage("truelook_chapel_hill-640x480.jpg");
  img[6] = loadImage("fayetteville_skycam-640x480.jpg");
  img[6] = loadImage("roxboro_skycam-640x480.jpg");
  img[7] = loadImage("wilson_skycam-640x480.jpg");
  img[8] = loadImage("wilimington_ptz-640x480.jpg");
  img[9] = loadImage("hive5-640x480.jpg");
  img[10] = loadImage("HGHPP_l.jpg");
  img[11] = loadImage("current.jpg");
  img[14] = loadImage("ELZBL_l.jpg");
  img[12] = loadImage("carolinabeach-640x480.jpg");
  img[16] = loadImage("201NST.jpg");
  img[17] = loadImage("auburntower-640x480.jpg");
  img[13] = loadImage("RLGSH_l.jpg");
  img[14] = loadImage("current-1.jpg");
  img[15] = loadImage("current-2.jpg");
  img[16] = loadImage("THMSD_l.jpg");
  img[17] = loadImage("current-3.jpg");
  stats = loadTable("Daily_Police_Incidents.csv")
}

function setup() {
  createCanvas(windowWidth, windowHeight);
  var options = {
    audio: false,
    video: true
  };

  capture = createCapture(options);
  capture.hide();
```

```
  var aper1 = createDiv("<font size='6'>Aperveillance: Watching
with Open Data</font>");
  aper1.position(30, 15);
  var aper2 = createDiv("<font size='6'>by J.J. Sylvia
IV</font>");
  aper2.position(30, 65);

  var rowCount = stats.getRowCount();
  for (var i = 0; i < rowCount; i++) {
    desc[i] = stats.getString(i, 1);
    loc[i] = stats.getString(i, 4);
    dt[i] = stats.getString(i, 2);
    howMany++;
    frameRate(0.1);
  }
}

function draw() {

 var aperveillance = createDiv("<font size='5'><a
href='http://www.jjsylvia.com/aperveillance/'>More
Info</a></font>");
aperveillance.position(375, 75);

 for (var y = 0; y <= height; y += 400) {
for (var x = 0; x <= width; x += 400) {
     var index = floor(random(img.length));
image(img[index], x, y, 400, 400);
}
  }
  var theWidth = width / 400;
  theWidth = round(theWidth);
  var r5 = random(1, theWidth);
  r5 = round(r5);
  r5 = r5*400;
  var theHeight = height / 400;
  theHeight = round(theHeight);
  var r6 = random(1, theHeight);
  r6 = round(r6);
  r6 = r6*400;
  image(capture, r5, r6, 400, 400);
  var rowCount = stats.getRowCount();
  for (var i = 1; i < 26; i++) {
    var r4 = floor(random(desc.length));
    var r = random(0, width-400);
    var r2 = random(175, 1900);
```

```
  var bWidth = textWidth(desc[r4]);
  var cWidth = textWidth(dt[r4]);
  var eWidth = textWidth(loc[r4]);
  rect(r-10, r2-15, bWidth+cWidth+eWidth+40, 20);
  textSize(12);
  text(dt[r4], r, r2);
  text(loc[r4], r+cWidth+10, r2);
  text(desc[r4], r+cWidth+eWidth+20, r2);
  }

rect(10,10,830,100,20);
  textSize(45);
}
```

This ensures that the media loads and the code does not encounter an error in middle of executing.

In the set up function, the canvas—where the code will be executed—is created to match the height and width of the window in which it is being loaded. The createCapture code accesses the webcam using options that set it to only access the video feed and not the audio feed. The createDiv functions then create a place to insert HTML, which creates the header that contains the title and byline. Finally, a **for loop** accesses all of the data from the CSV file and stores it in variables so that it can be manipulated with code.

The draw function is used to put the majority of the content onto the screen. The first loop featured in this section selects a random webcam image and places it in the next spot in the grid, sized to 400 pixels wide by 400 pixels high. It does this until the entire screen, or canvas, is filled. The next section of code determines a random grid spot on which to overlay the webcam image from the viewer's own camera, if available. The image is then displayed using the image(capture, r5, r6, 400, 400); code. The majority of the remaining code determines the length of the crime incident data and places it randomly across the canvas in a white rectangle.

The main generative nature of this project involves the random placement of random webcam images in juxtaposition with crime data and the image from the viewer's camera. The positioning of these are changed approximately every 15 seconds to keep the display fresh and to prevent the viewer from being able to dwell too long on the relationship between the placement of the text and the images. However, it would also be quite easy to change the project and, for example, display the most common types of crimes, each sized larger depending on their frequency of occurrence.

The key to an artistic approach such as this is to try to think outside of the normal data visualization approach. In the case of this project, I was less interested in the frequency of any particular crimes—a more traditional data analysis question—and more interested in bigger picture Humanities-esque questions about the nature of surveillance. For example, should we have an expectation of privacy when we are in public places?[11] Aperveillance extends that question to ask who should have access to the data generated by surveillance and what opportunities such data affords when combined in novel or unexpected ways. p5.js allowed me to create a provocative artistic interpretation of that question in a way that a more traditional data visualization tool like Tableau would not. Aperveillance, for example, was displayed in a manner that mimics the aesthetic of rows of footage from surveillance cameras. Such an aesthetic harkens back to Jeremy Bentham's Panopticon.[12] The Panopticon is a circular building designed to serve as a prison. A circular guard tower sits in the middle, with individual cells located in the outer wall. A bright light from the guard tower would enable the guards to see all of the prisoners, but prevent the prisoners from being able to see the guards or even confirm whether or not they were in the tower. Camera-based security systems mirror this aesthetic with rows of televisions that display camera images. The guard sees out through the cameras, but those being observed cannot see the guards or even know if the footage from the cameras is being monitored. This aesthetic is important because the power disparity between the guard and the prisoners emerges through the aesthetic design.

Finally, my index.html and sketch.js files, along with the images and CSV file were made publicly available through my Dropbox. This could also be done by uploading them to a web server. I linked the URL aperveillance.com to the URL for the index.html file in my Dropbox and then anyone in the world could access the program through that URL.

[11] Helen Nissenbaum, "Protecting Privacy in an Information Age: The Problem of Privacy in Public," *Law and Philosophy* 17, no. 5/6 (November 1998): 559. https://doi.org/10.2307/3505189.

[12] Jeremy Benthamn, *Works of Jeremy Bentham* (S.l. London: Forgotten Books, [1789] 2015); Michel Foucault, *Discipline and Punish: The Birth of the Prison*, 2nd ed. (New York: Vintage Books, [1975] 1995).

Conclusions

Code/art projects make use of the iterative nature of generative design. It is this iteration, in part, that affords the possibility of creating artistic data visualizations. This artistic approach allows one to ask different questions than those that could be answered through traditional data analysis. Based on my analysis of qualitative verbal feedback provided by viewers of the Aperveillance project, several conclusions can be drawn. First, many people are unaware of the amount of data that is openly accessible on the internet, and have infrequently, if ever, thought about the consequences of combining multiple data sources. While this accessibility bothered viewers in a general way, they were often unable to connect it to concerns about their own individual privacy. Only upon realizing that a camera was incorporating their image into the project was this connection made. Many viewers were bothered about their unwilling inclusion in surveillance, though this process mirrors the form of much of contemporary technological surveillance. By experiencing their own surveillance as both the surveilled and the one surveilling, viewers were able to articulate questions about the tensions between privacy and surveillance of which they were previously unaware or unconcerned.

While there are a wide variety of languages available to use, p5.js offers an option that is accessible to beginners. My students and workshop participants with little prior programming experience have been able to master the basics of the language within as little as one week. Nonetheless, it is important to consider the goal of your data visualization when selecting the tool, as p5.js can be a more difficult option when trying to complete traditional data visualization analyses.

References

Ball, Kirstie, Kevin Haggerty, and David Lyon, eds. *Routledge Handbook of Surveillance Studies* 1. Paperback ed. Routledge International Handbooks. London: Routledge, 2012.

Benthamn, Jeremy. *Works of Jeremy Bentham*. S.l. London: Forgotten Books, 2015.

Berman, Jules J. *Principles of Big Data: Preparing, Sharing, and Analyzing Complex Information*. Amsterdam: Elsevier, Morgan Kaufmann, 2013.

Bohnacker, Hartmut, Benedikt Gross, Julia Laub, and Claudius Lazzeroni. *Generative Design: Visualize, Program, and Create with Processing*. New York: Princeton Architectural Press, 2012.

Eiben, Agoston E., and James E. Smith. *Introduction to Evolutionary Computing*, 1st ed., 2 printing, Softcover version of original hardcover ed. 2003. Natural Computing Series. Berlin: Springer, 2010.

Foucault, Michel. *Discipline and Punish: The Birth of the Prison*. 2nd Vintage Books ed. New York: Vintage Books, 1975.

Hornby, Gregory, Al Globus, Derek Linden, and Jason Lohn. "Automated Antenna Design with Evolutionary Algorithms." American Institute of Aeronautics and Astronautics, 2006. https://doi.org/10.2514/6.2006-7242.

Manovich, Lev. "Info Aesthetics," 2001. http://www.manovich.net.

Marx, Gary T. "Surveillance Studies." In *International Encyclopedia of the Social & Behavioral Sciences*, 733–741, 2015. https://doi.org/10.1016/b978-0-08-097086-8.64025-4.

Mayer-Schönberger, Viktor, and Kenneth Cukier. *Big Data: A Revolution that Will Transform How We Live, Work, and Think*. First Mariner Books edition. Boston: Mariner Books, Houghton Mifflin Harcourt, 2014.

McCarthy, Lauren, Casey Reas, and Ben Fry. *Getting Started with p5.js: Making Interactive Graphics in JavaScript and Processing*. San Francisco, CA: Maker Media, Inc., 2015.

Monaghan, Peter. "Watching the Watchers." *The Chronicle of Higher Education*, 2006.

Nissenbaum, Helen. "Protecting Privacy in an Information Age: The Problem of Privacy in Public." *Law and Philosophy* 17, no. 5/6 (November 1998): 559. https://doi.org/10.2307/3505189.

Reas, Casey, and Chandler McWilliams. *Form Code: In Design, Art, and Architecture*. New York: Princeton Architecturel Press, 2010.

Zimmer, Michael. "OkCupid Study Reveals the Perils of Big-Data Science." *Wired*, May 14, 2016. Accessed April 12, 2017. https://www.wired.com/2016/05/okcupid-study-reveals-perils-big-data-science/.