Teaching Topology with Code

Brooks G. Mershon
Duke University, brooks.mershon@duke.edu

***Abstract* - The popular data visualization library D3.js was used to create illustrations of several key concepts covered in an undergraduate course in topology. Approaches for developing visualizations which augment the other learning resources found in undergraduate math courses are discussed. Three different visualizations focusing on the topics of height-functions, Rips-complexes, and dynamic sensor networks are offered as case studies in using code to illustrate abstract concepts. I give general suggestions for incorporating such learning tools into undergraduate math courses while analyzing the design of these interactive tools at a high level.**

Keywords - algorithms, data visualization, D3.js, information design, JavaScript, online education, pedagogy, topology,

MOTIVATION

Math 412: Topology with Applications is an undergraduate course offered at Duke University. The course seeks to introduce students to the subject of Topological Data Analysis (TDA). As the course is designed for an undergraduate audience of mixed mathematical background, the topic is explored by introducing applications of TDA early, proceeding throughout the semester by weaving in topological and algebraic tools [1]. A significant portion of lecture time is spent introducing concepts through chalkboard diagrams. Abstract concepts meet symbolic tools of manipulation, but not before being related to students in a spatial way.

I was interested in seeing how the static diagrams that can be drawn on a chalkboard and printed in a textbook could be expanded upon through the development of interactive "gadgets" that run in the web browser. How might students interact with software which allowed them to experiment with math concepts requiring a graphical representation? What types of design principles should guide the development of these interactive tools? I felt these questions could be best answered by writing some code and testing out several tools in order to gain a better understanding of the benefit such interactive tools might provide in an undergraduate math course. I began by illustrating some of the concepts I was learning in Math 412, with the goal of presenting the work for my final project. This project began with the intention of inspiring students who take the course in subsequent offerings to examine my code, the completed gadgets, and this paper. I hope others will consider continuing and improving upon the work I started for this project.

DIAGRAMS

At the heart of an introductory course in Topological Data Analysis is a rich set of concepts and intuitions that are likely to be novel for the intrepid student. Before the algebraic and topological tools supporting the many theories of the course can be introduced, it is necessary to find an entry-point for the student so that she can orient herself and prepare to relate a new idea to the existing mathematical intuitions she has already developed. A well-designed diagram can offer a useful pedagogical foothold.

Consider the zero-dimensional persistence diagram, which encodes the persistence of connected components. This concept is often introduced visually by presenting a real-valued function defined on the closed interval from zero to one. Then, for each real number $r \in \mathbb{R}$ we define the sublevel set, or threshold set, of our function with threshold parameter r to be:

$$I_r = f^{-1}\big((-\infty, r]\big) = \{x \in I \mid f(x) \leq r\}$$

The instructor might draw a squiggle on the chalkboard (which obeys the vertical line test) and ask students to consider the number of components at various "heights" on our curve. Natural spatial intuitions of connectedness and continuity are engaged. The student will notice that components are born at local minima, and two components merge at local maxima. Then, working out the pairs of local minima and maxima, dots corresponding to birth-death pairs may be plotted by hand in a persistence diagram. This visual approach to exploring mathematical concepts is ubiquitous, but its extension to "interactive diagrams," or lightweight pedagogical software tools, is not.

In an attempt to explore the use of interactive pedagogical tools as a supplementary resource for higher-level math courses, I developed several interactive "gadgets" which illustrate a few of the concepts covered in the course. The very first of these gadgets was designed to allow a user to easily create and modify a continuous function from which zero-dimensional persistence diagrams could be generated. Figure 1 shows a screenshot of this gadget running in a browser.

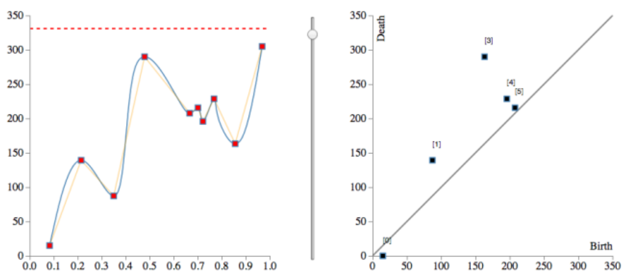


FIGURE 1
SCREENSHOT: A USER CAN CLICK ANYWHERE TO REVEAL THE CONTROL POINTS THAT DETERMINE THE CURVE'S SHAPE.

The goal of this first gadget was to allow users to create their own example curves, so that they can test out the various configurations of minima and maxima that arise, as well as study how altering the curve changes the corresponding zero-dimensional diagram. Reasoning spatially about the effect that shifting a local maximum in the curve will have on a particular dot of finite persistence requires a bit of mental juggling. However, armed with such an interactive tool, a student can iterate quickly through many examples and build up a new intuition. An interactive diagram or simulation augments the other mediums of learning, which have traditionally included lectures, printed textbooks, and pencil-and-paper exercises. There is a real potential for computer aided learning using graphical simulations of math concepts to help build new intuitions. As one mathematician who interacted with computer simulations of 4D geometric objects explained, "I tried turning the hypercube around, moving it away, bringing it up close, turning it around another way. Suddenly I could *feel* it! The hypercube had leaped into palpable reality, as I learned how to manipulate it, feeling in my fingertips the power to change what I saw and change it back again. The active control at the computer-console created a union of kinesthetic and visual thinking which brought the hypercube up to the level of intuitive understanding" [8]. To think such experiences might be possible using interactive simulations in the browser is not far-fetched.

DEVELOPING INTERACTIVE TOOLS

All of the interactive tools that I have developed for this course were written in JavaScript. I also make use of a popular data visualization library called D3.js, which is used by the New York Times and the Washington Post to produce the stunning interactive graphics that are routinely published on their websites. D3, which stands for Data-Driven Documents, allows designers to "selectively bind input data to arbitrary document elements, applying dynamic transforms to both generate and modify content" [3]. The JavaScript library is primarily designed for manipulating SVG elements in the DOM, making use of SVG, HTML and CSS standards to style and dynamically update the color, shape, size, and position of elements.

D3 is an ideal tool with which to build interactive gadgets for a college math course. The static two-dimensional diagrams encountered in math texts lend themselves to the customizability and dynamic nature of the visuals D3 can produce. Whenever a sequence of static images is used in a text to illustrate a process, a programmatically defined simulation may be used to give a continuous animation of the process. Figure 2 shows just such a diagram, which illustrates how a hole in the coverage of a communication graph can arise. Figure 3 shows a screenshot of an interactive tool I developed which illustrates this same concept.

However, the value in using a tool like D3 lies not in its ability to make diagrams dynamic, but rather in its ability to give control to the user. When a simulation or gadget is driven by user input, the student is more likely to be actively—not passively—taking in the information being presented. Interactive learning tools allow the user to learn through guided exploration, as well as create their own examples.

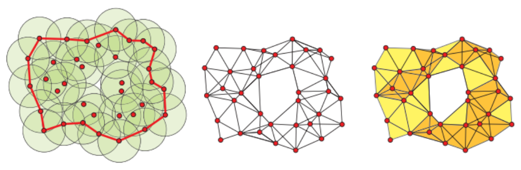


FIGURE 3. In a sensor network with a sufficiently large hole in coverage [left], the communication graph [center] has a cycle that cannot be 'filled in' by triangles. The filled in Rips complex [right] 'sees' this hole, even as an abstract complex devoid of sensor node location data.

FIGURE 2
THIS FIGURE ILLUSTRATES A DYNAMIC PROCESS [6].

Interactive pedagogical tools for a college math course should be intuitive to use and easily accessible. All of the gadgets that I developed for this project make use of a simple HTML page with one or more referenced JavaScript files. These scripts create a visual by latching on to a suitable element, such as a div or span in the document, and then creating one or more SVG elements which will contain the entirety of the visual. A typical gadget consists of an HTML page with a 960 x 500 pixel SVG containing all of the visual elements that will be produced and modified by the program. These pages can be easily hosted by a professor's course site, as they require only a single HTML document and the necessary JavaScript files to be accessed by students with a modern browser.

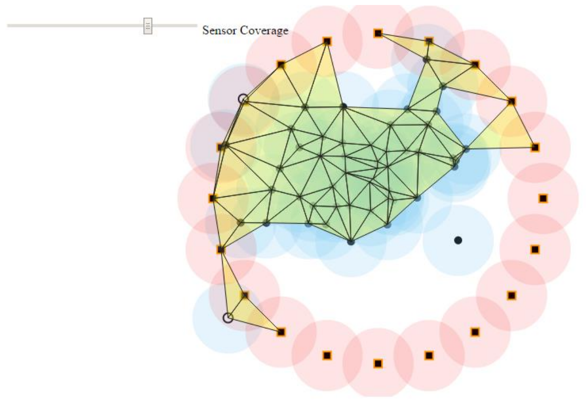


FIGURE 3
SCREENSHOT: SENSORS WITHIN THE CIRCULAR REGION MOVE ABOUT RANDOMLY. THE BLANKET OF COVERAGE UPDATES CONTINUALLY AND CAN BE MODIFIED BY CHANGING THE RADIUS OF THE COVERAGE DISK FOR EACH SENSOR BY DRAGGING A SLIDER.

ILLUSTRATING MATH CONCEPTS USING D3

An instructor who seeks to provide students with interactive pedagogical tools as a supplementary resource should aim to recruit students to build them. Nearly every topic in the lecture schedule provided at the beginning of Math 412 could benefit from the development of interactive examples or simulations. Students will find various concepts throughout the course that they either feel comfortable with, or would like to examine more deeply. The familiarity that many math students have with programming, as well as the extremely thorough and engaging documentation provided for D3.js, suggests that asking students to develop learning tools which could be used in subsequent offerings of the course is more than reasonable. If lab exercises or extra-credit opportunities were to be used to incentivize the development of such pedagogical tools, math courses would begin a process that benefits students in two ways:

1. The development of an illustration for a particular concept offers the opportunity for the student to gain greater insights into the concept itself, as well as the many challenges encountered in applying computer science concepts to a mathematical field (and vice versa).
2. Students who will later use these gadgets receive another learning resource—one that was designed by a student—and therefore can potentially avoid the presentation issues that might be found in the traditional texts and reading materials.

I tested out the application of the D3 library to three separate course topics. These gadgets illustrate the following concepts: zero-dimensional diagrams and height-functions (Figure 1); Rips-complexes and triangulation of point clouds; and the coverage of dynamic sensor networks. The following sections explain the motivations and design considerations of each of these gadgets.

HEIGHT-FUNCTION

The height-function gadget I previously introduced presents the user with control points for a B-spline (Figure 4). Clicking to the right of the existing curve creates a new control point. Clicking and dragging existing points changes the shape of the curve, and the vertical line rule is enforced (i.e. dragging a control point only allows for movement between adjacent control points). Control points can be deleted from the middle of a curve as well without starting over from scratch.

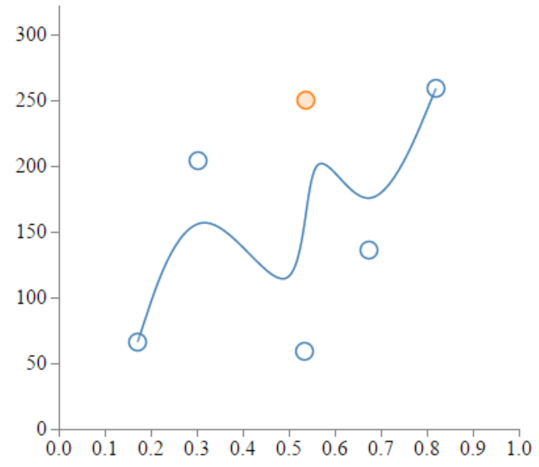


FIGURE 4
SCREENSHOT: THE CURRENTLY SELECTED CONTROL POINT IS HIGHLIGHTED. SMOOTH, CONTINUOUS CURVES CAN QUICKLY BE DRAWN AND THEN LATER MODIFIED.

A zero-dimensional persistence diagram is created from the drawn curve whenever the user clicks a button. This diagram (Figure 5) can be regenerated repeatedly as the curve is modified. Special markers indicate where dots in the persistence diagram have shifted from their previous position, as can be seen in Figure 6.

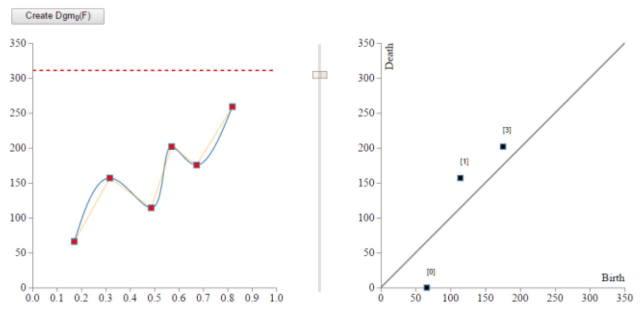


FIGURE 5
SCREENSHOT: THE ZERO-DIMENSIONAL DIAGRAM FOR A USER-DEFINED CURVE IS PRODUCED.

This gadget stands as a prototype for the typical interactive tool that could be used in a course like Math 412. It is self-contained, with few a basic keyboard (DELETE key) and mouse controls. The gadget has a familiar visual language: it uses vertical and horizontal axes, and simple circles and squares mark control points, critical points, and dots of both finite and infinite persistence. The diagram should be immediately familiar to a user who has encountered a text in Computational Topology [4].

The temptation to produce gadgets that are too complicated or strive to encompass too many concepts should be avoided. The height-function gadget does one thing and one thing only: it allows users to play with various curves and create zero-dimensional diagrams from them quickly and easily. The educational value of such a simple interactive tool is clear, as it allows a learner to ascertain what can be learned from two hours of pencil-and-paper exercises in ten minutes of concentrated interaction.

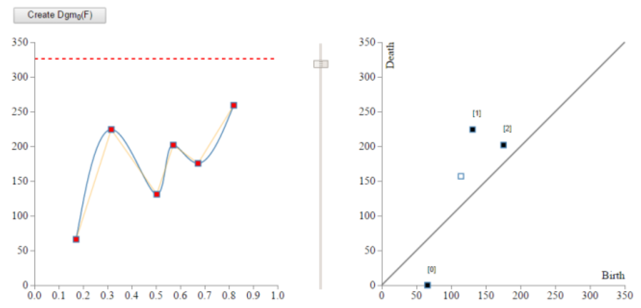


FIGURE 6
SCREENSHOT: ALTERING THE EXISTING CURVE AND CLICKING THE BUTTON AGAIN ALLOWS FOR A STUDENT TO STUDY THE EFFECT THAT CHANGING THE CURVE'S FEATURES HAS ON THE PERSISTENCE DIAGRAM.

RIPS-COMPLEX

The second interactive tool that I developed allows a user to create Rips-complexes from a point cloud. The point cloud is created by sampling points from a curve that is initially "hidden" from the user. By moving a slider controlling the radius r of the ball around each point, the user can interactively "grow" a simplicial complex, as Figure 7 shows.

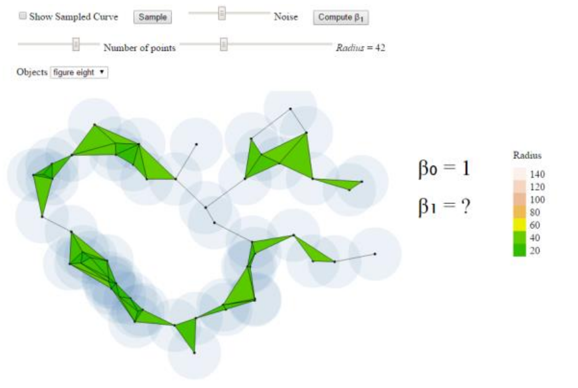


FIGURE 7
SCREENSHOT: VARIOUS SLIDERS ALLOW THE USER TO PLAY WITH DIFFERENT LEVELS OF SAMPLING NOISE, NUMBER OF POINTS, AS WELL AS CHANGE THE FIGURE SAMPLED.

Selecting one of several different curves from a drop-down menu and experimenting with varying numbers of points and sampling noise allows a user to gain a visual understanding of a phenomenon that is rather hard to draw by hand. Indeed, one of the more entertaining aspects of this gadget is the randomness that it introduces. When a student notices that the same Betti values are achieved at roughly the same ball radius r over many samples of the same curve, her understanding of the stability of the 1-dimensional persistence algorithm will have been *experienced*, rather than memorized. Figures 8 through 10 show the evolution of a Rips-complex for a particular curve.

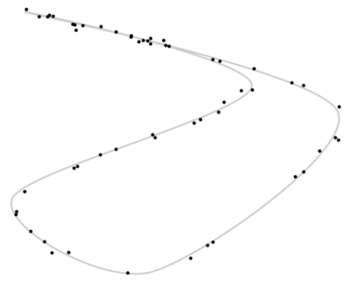


FIGURE 8
SCREENSHOT: THE USER CHECKS THE BOX "SHOW SAMPLED CURVE." GAUSSIAN NOISE IS APPLIED TO POINTS WHICH HAVE BEEN UNIFORMLY SAMPLED ALONG THE CLOSED CURVE'S LENGTH.

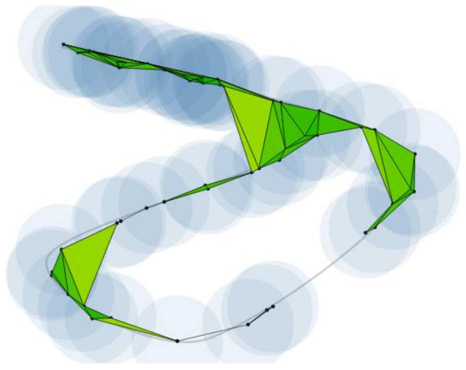


FIGURE 9
SCREENSHOT: CLOSE INSPECTION OF THE FORMATION OF EDGES AND TRIANGLES IS ACCOMPLISHED BY SLOWLY MOVING THE RADIUS SLIDER.

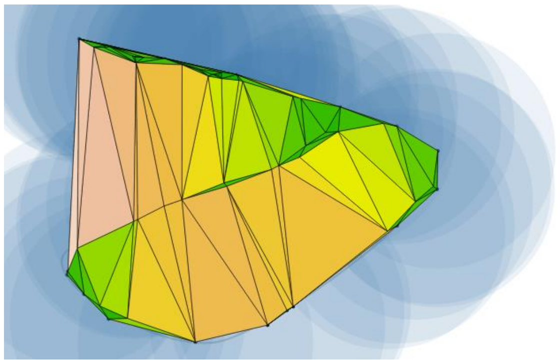


FIGURE 10
SCREENSHOT: THE COLORING OF EACH TRIANGLE CORRESPONDS TO THE LEGEND INCLUDED IN FIGURE 7. THE RELATIVE PERSISTENCE OF "HOLES" WHICH HAVE BEEN FILLED IN IS ENCODED LOOSELY IN THE COLORS CORRESPONDING TO THE RADIUS AT WHICH TRIANGLES APPEAR.

The user can see how edges form when balls intersect and triangles are added whenever the three faces of a 2-simplex (triangle) are present. The implementation of this gadget entailed creating a Delaunay triangulation of the point cloud and using this triangulation as a limiter when adding edges and triangles. Edges are only added if they are present in the Delaunay triangulation. Triangles are added if their three edges are present. The Delaunay triangulation needed to create this interactive tool is one example of a small problem which a student developer would have to overcome. In creating this particular example, I came to appreciate the unfortunate algorithmic complexities of reducing a boundary-matrix to find 1-cycles.

The process of transforming ideas and diagrams from a course like Math 412 into interactive learning tools encourages the very sort of interdisciplinary thinking this course was designed around. How can mathematical ideas be applied to real world applications requiring fast computation and algorithmic approaches to solving problems? Bite-sized interactive examples provide students with a manageable goal: to illustrate a particular math concept using data structures and algorithms that are increasingly becoming a form of literacy in STEM education. Developing the Rips-complex gadget certainly provided me with a challenge encompassing skills from the fields of computer science, mathematics, and visual design. I would imagine other students would find the development of gadgets addressing different topics just as engaging—and rewarding.

DYNAMIC SENSOR NETWORKS

The third gadget that I designed for this project was inspired by Vin de Silva and Robert Ghrist's paper, *Coordinate-free coverage in sensor networks with controlled boundaries via homology*, which discusses applications of homology to sensor networks capable of making only local measurements in a specified region [6]. I was interested in the included diagrams, particularly those which described the holes that form in the coverage of a dynamic sensor network (Figure 11).

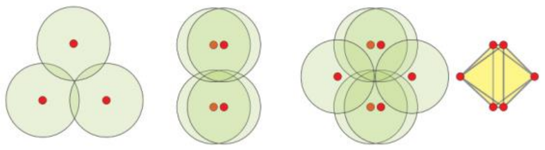


FIGURE 11
THE FORMATION OF A RIPS-COMPLEX AROUND SENSORS OF A GIVEN RADIUS. THE RESULTING COMPLEX IS NOT NECESSARILY IN THE DESIRED DIMENSION [6].

The paper poses the following question to the reader: can an evader avoid being caught in the time-dependent union of coverage discs? Mentally pushing around holes in a network blanket and trying to imagine what the formation and collapse of such holes might look like is difficult. The Dynamic Sensor Network gadget was built to help illustrate this dynamic process, as is seen in the screenshots of Figures 3 and 12.

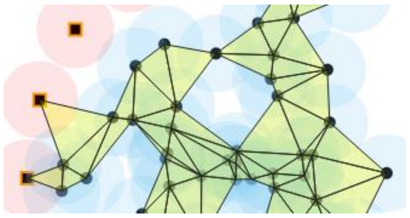


FIGURE 12
SCREENSHOT: HOLES FORM, MOVE AROUND, AND COLLAPSE IN REAL TIME.

This third gadget demonstrates a concept which benefits from a spatial representation, yet does not necessarily require user input. Unlike the height-function gadget which requires the user to interact with the tool, and the Rips-complex which gives the user various parameters to adjust, the dynamic sensor network gadget allows for close inspection of a process that is perhaps best presented in a dynamic way. However, this gadget is not video; this illustration is programmatically driven. It takes advantage of temporality to illustrate a process, and it introduces randomness in its design to achieve what is difficult to illustrate by hand.

TIPS AND TRICKS

A student attempting to master new mathematical concepts should use all of the tools in their toolbox to build useful intuitions. The disciplines of math, engineering, computer science, and physics tend to use spatial reasoning to illustrate relations between concepts, processes, and ideas which otherwise become too cumbersome to leave to plain text [7]. As we have seen here, two-dimensional graphics need not be the only visual aids students encounter in college math courses like Math 412. Interactive pedagogical tools that model and illustrate math concepts would certainly benefit the students of any undergraduate course. Furthermore, the path to developing these gadgets can be made short and fun by encouraging students to choose concepts that they would like to illustrate—and then have them build those gadgets. What follows is an examination of some of the design considerations I had throughout the development of my first suite of gadgets.

The Rips-complex gadget received some design tweaks late in its development. After I solved the problem of over-triangulating the point cloud (Figure 13) by using the Delaunay tessellation as a limiter, it was suggested to me to expand the gadget by including a 1-dimensional persistence diagram. This diagram would be updated whenever the user clicked a button to compute β_1. I was reluctant to complicate the visual by tying multiple SVG elements to the same script, as I had done with the height-function gadget. I felt the addition of the diagram would detract from the simplicity of the diagram and its small set of controls. Not wanting to significantly alter the user's experience of moving sliders and watching the simplicial complex change, I settled on an easy tweak to the code which produces an interesting encoding of information.

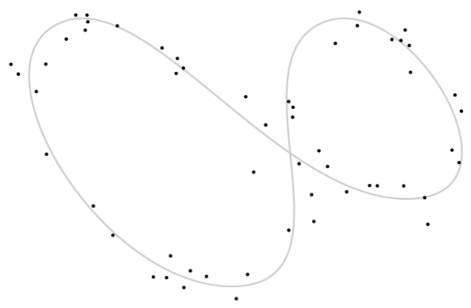


FIGURE 13
SCREENSHOT: "POINT CLOUD" SAMPLED FROM SVG PATH ELEMENT

Figures 14 through 17 show the progression of a user increasing the radius parameter r for a point cloud sampled from a figure-eight. We see in Figure 13 and 14 that there are clearly two significant "holes" in the point cloud. The legend shows the values of r that each color corresponds to. The first triangles produced will be colored a dark green, which corresponds to a small radius for the balls around the points. As the radius expands, more triangles appear, and their colors reflect the larger radius at which they arrive.

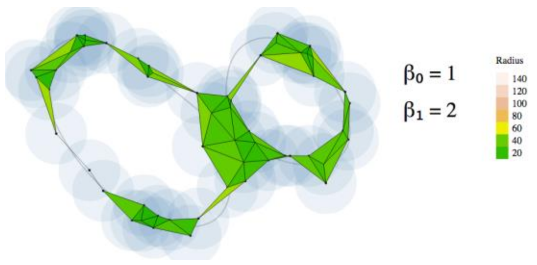


FIGURE 14
SCREENSHOT: THE LEGEND TO THE RIGHT OF THE DIAGRAM SHOWS THE COLORS CORRESPONDING TO THE RADIUS AT WHICH TRIANGLES APPEAR

This color treatment presents an alternative interpretation of encoding persistence. While the coloring of the triangles is only loosely coupled to the actual values of r at which various 1-cycles are born and die, upon inspection of Figure 17, it is clear that the coloring gives a general idea of the relative significance of each hole that has been filled in. The 1-cycles of greatest persistence are revealed to us by the lightly colored triangles which tessellate their interiors. For some, this interpretation may be useful, as it encodes and overlays information pertaining to persistence on the object being studied. This information makes spatial sense in a way that persistence diagrams may not for some students, and is therefore a worthwhile addition to this gadget.

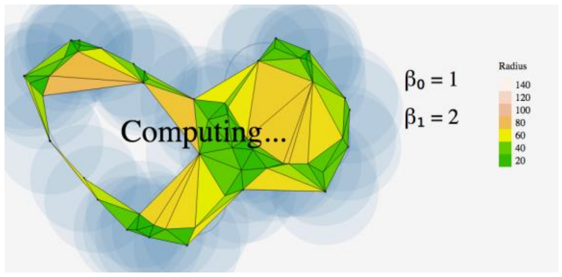


FIGURE 15
SCREENSHOT: CLICKING A BUTTON UPDATES THE BETTI NUMBERS. A BOUNDARY-MATRIX IS REDUCED USING THE "LOWEST-ONE" ALGORITHM.

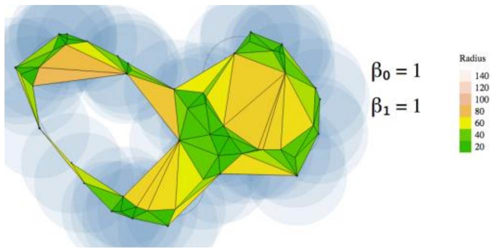


FIGURE 16
SCREENSHOT: ONE OF THE MAJOR HOLES HAS BEEN FILLED IN. NOTE THE LARGE RADIUS OF THE BALLS AROUND EACH POINT.

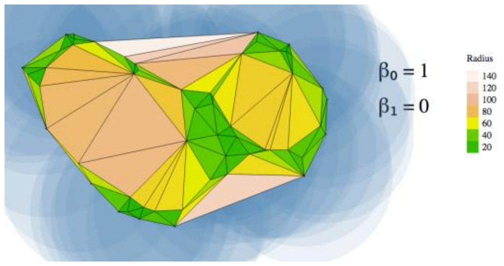
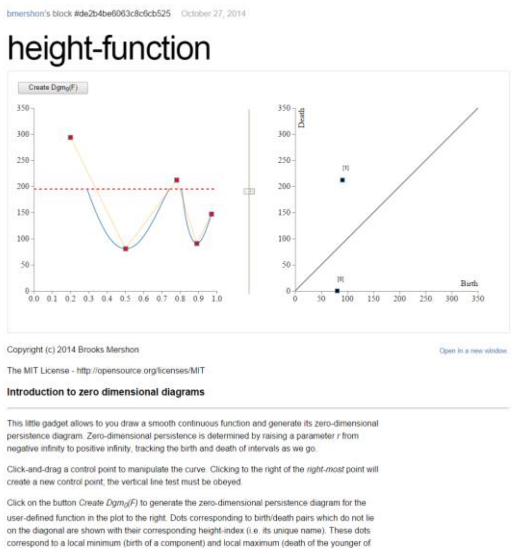


FIGURE 17
SCREENSHOT: ALTHOUGH NO 1-CYCLES EXIST, OUR COLOR CODE PROVIDES A CLEAR VISUAL CUE TO THE PERSISTENCE OF THE 1-CYCLES THAT HAVE BEEN FORMED.

ANATOMY OF A "GADGET"

Making a gadget like the ones I have shown here requires a few simple steps. I include here the specific method I use to deploy these gadgets for public access. While there are alternative ways of making interactive gadgets easily accessible for students, this example serves a good starting point. The outline that follows assumes familiarity with GitHub and the D3 library. Mike Bostock, creator of the D3 library and an interactive graphics designer for The New York Times, has made a short write-up which explains the process of hosting simple visualizations [2].

1. Prepare a bare-bones HTML document which references a current version of D3.js, as well as the JavaScript files containing the code for your project.
2. Push an index.html file, your scripts, and any other referenced files containing the data needed by your gadget to a GitHub Gist repository, including a README file for the gadget description. The Rips-complex gadget, for example, uses a separate .txt file which defines the names and shapes of the curves a user can sample.
3. Mike Bostock's website, bl.ocks.org, scrapes users' gists from GitHub and renders the output in a simple iframe. Figure 18 shows the standard appearance of a gadget found on bl.ocks.org. A page containing thumbnails and links to all of the gists a GitHub user has created is found at bl.ocks.org/USERNAME.



bmershon's block #de2b4be6063c8c6cb525 October 27, 2014

height-function

Copyright (c) 2014 Brooks Mershon

The MIT License - http://opensource.org/licenses/MIT

Introduction to zero dimensional diagrams

This little gadget allows to you draw a smooth continuous function and generate its zero-dimensional persistence diagram. Zero-dimensional persistence is determined by raising a parameter *r* from negative infinity to positive infinity, tracking the birth and death of intervals as we go.

Click-and-drag a control point to manipulate the curve. Clicking to the right of the *right-most* point will create a new control point; the vertical line test must be obeyed.

Click on the button *Create Dgm$_0$(F)* to generate the zero-dimensional persistence diagram for the user-defined function in the plot to the right. Dots corresponding to birth/death pairs which do not lie on the diagonal are shown with their corresponding height-index (i.e. its unique name). These dots correspond to a local minimum (birth of a component) and local maximum (death of the younger of

FIGURE 18
SCREENSHOT: A "BL.OCK": THE DESCRIPTION OF THE GADGET AND AN IFRAME CONTAINING THE OUTPUT OF THE GIST CONTAINING THE DEVELOPER'S CODE IS INCLUDED.

Having a convenient third-party host like bl.ocks.org helps student developers get their gadgets off the ground with little trouble. A class web page could also be used to host the interactive tools for a course, but it would be difficult to provide the free service of cached, continually updated versions of the code on GitHub that bl.ocks.org provides. Figure 19 shows an example gallery of the gadgets I created for this project. This page can be accessed by visiting bl.ocks.org/bmershon. All of the gadgets have been tested in Chrome.

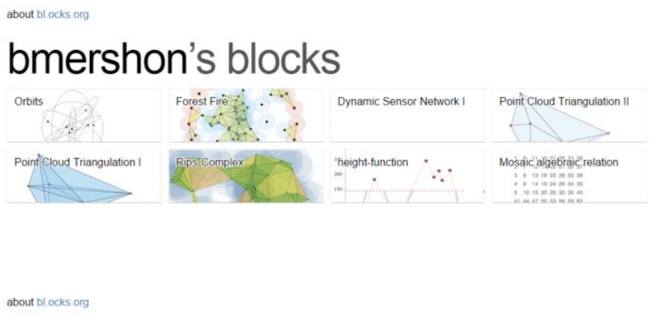
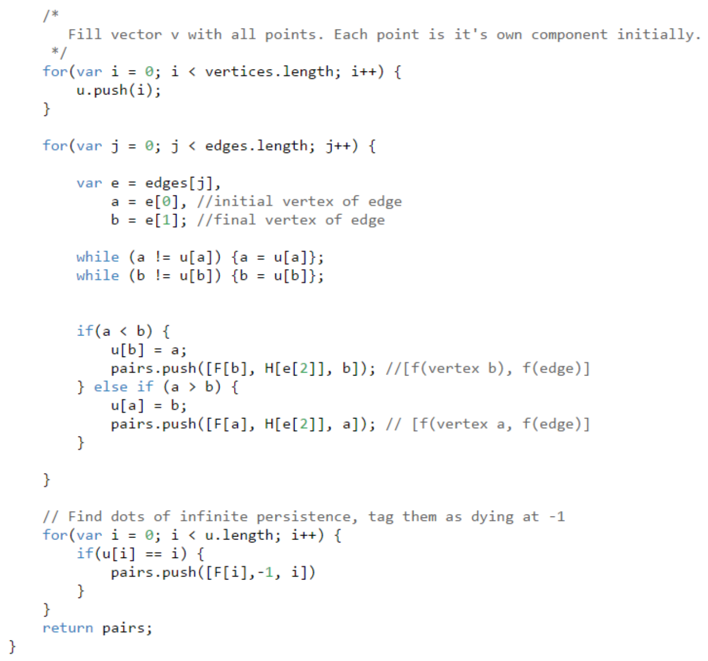


FIGURE 19
SCREENSHOT: A GALLERY SHOWS THUMBNAILS OF EACH USER'S GIST. NOTE: THUMBNAILS ARE MANUALLY UPLOADED.

In addition to the iframe used in each bl.ock, there is a short description of the gadget as specified in the README and a syntax-highlighted presentation of the code used by the gist. This presentation is useful or letting students peek "under the hood" of the illustration to see some of the algorithms (pseudocode) studied in class implemented in JavaScript (Figure 20).



```
    /*
      Fill vector v with all points. Each point is it's own component initially.
     */
    for(var i = 0; i < vertices.length; i++) {
        u.push(i);
    }

    for(var j = 0; j < edges.length; j++) {

        var e = edges[j],
            a = e[0], //initial vertex of edge
            b = e[1]; //final vertex of edge

        while (a != u[a]) {a = u[a]};
        while (b != u[b]) {b = u[b]};


        if(a < b) {
            u[b] = a;
            pairs.push([F[b], H[e[2]], b]); //[f(vertex b), f(edge)]
        } else if (a > b) {
            u[a] = b;
            pairs.push([F[a], H[e[2]], a]); // [f(vertex a, f(edge)]
        }

    }

    // Find dots of infinite persistence, tag them as dying at -1
    for(var i = 0; i < u.length; i++) {
        if(u[i] == i) {
            pairs.push([F[i],-1, i])
        }
    }
    return pairs;
}
```

FIGURE 20
SCREENSHOT: A SECTION OF SYNTAX-HIGHLIGHTED CODE IS SHOWN BELOW EACH GADGET ON BL.OCKS.ORG. A UNION-FIND ALGORITHM FINDS THE BIRTH-DEATH PAIRS FOR THE HEIGHT-FUNCTION GADGET.

HIDDEN CONNECTIONS

One of the joys of working on this project was finding small problems or micro-examples of a concept which could be extracted from a particular gadget and presented on its own. Sometimes these micro-examples fall outside of the original area of study.

In order to build the Dynamic Sensor Network gadget, I needed to apply the edge and triangle drawing portions of the Rips-complex code to a dynamic point cloud. The problem I now faced was not one of topology, but rather one of random motion and wandering behavior. If I wanted my sensors to move in a random way, I would need to way of emulating what I was picturing in my head: animals pick up cheap forest fire sensors in a specified region and provide some smooth locomotion for the vertices of my simplicial complex. However, I wanted to keep my code simple, and not wanting to dive into Craig Reynolds' steering behavior algorithms [5], I decided on a compromise solution which uses very little code. Each point in the Dynamic Sensor Network gadget is given a random orbit that it follows slowly enough so that the periodicity of the illustration unnoticeable. My Orbits gadget, which can be accessed from the gallery seen in Figure 19, demonstrates my technique for moving sensors in the larger gadget from which the example is borrowed. Figure 21 shows the ortbits drawn for each dot as pre-calculated SVG path elements.

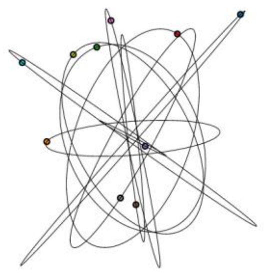
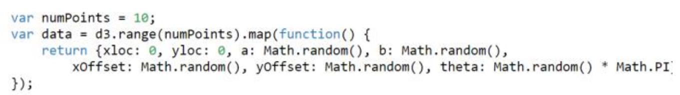


FIGURE 21
SCREENSHOT: SVG PATHS ARE USED TO SHOW THE "ORBITS" EACH SENSOR FOLLOWS IN THE DYNAMIC SENSOR NETWORK GADGET

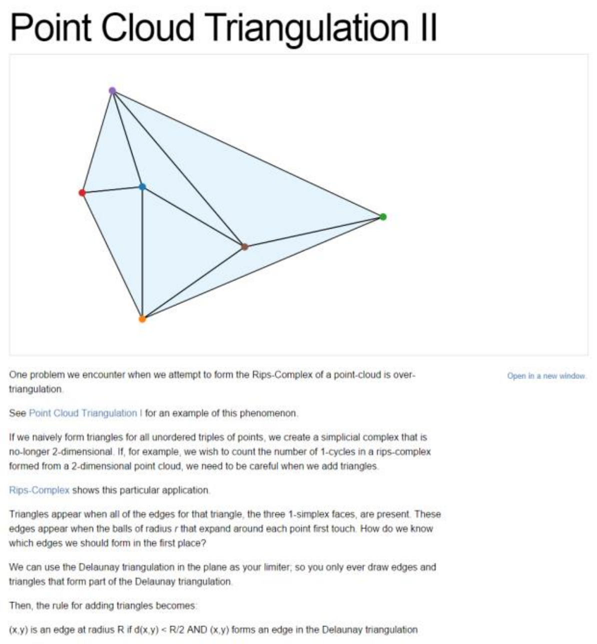
Figure 22 shows the code pertaining to the movements used in the Dynamic Sensor Network and Orbits gadgets. Many gadgets will create jumping off points to expand upon many concepts they address. Smaller-scale visualizations of point cloud triangulation using Delaunay Triangulation as a limiter serves as another "example within an example" approach to illustrating the concepts of a math course (Figure 23).



```
var numPoints = 10;
var data = d3.range(numPoints).map(function() {
    return {xloc: 0, yloc: 0, a: Math.random(), b: Math.random(),
        xOffset: Math.random(), yOffset: Math.random(), theta: Math.random() * Math.PI
});
```

FIGURE 22
SCREENSHOT: A PART OF THE CODE FROM THE BL.OCK CONTAINING THE ORBITS GADGET. THE DATA STRUCTURE FOR THE DYNAMIC POINTS IS DEFINED.

I hope that my first attempt to illustrate a few of the topics covered in Math 412: Topology with Applications demonstrates the need for interactive pedagogical tools in undergraduate math courses. Visual approaches to math and an emphasis on spatial reasoning deserve more attention than they currently receive. The development of teaching tools using tools like JavaScript and D3 can and should be carried out by the current students of undergraduate as well as graduate math courses.



Point Cloud Triangulation II

One problem we encounter when we attempt to form the Rips-Complex of a point-cloud is over-triangulation.

Open in a new window.

See Point Cloud Triangulation I for an example of this phenomenon.

If we naively form triangles for all unordered triples of points, we create a simplicial complex that is no-longer 2-dimensional. If, for example, we wish to count the number of 1-cycles in a rips-complex formed from a 2-dimensional point cloud, we need to be careful when we add triangles.

Rips-Complex shows this particular application.

Triangles appear when all of the edges for that triangle, the three 1-simplex faces, are present. These edges appear when the balls of radius *r* that expand around each point first touch. How do we know which edges we should form in the first place?

We can use the Delaunay triangulation in the plane as your limiter; so you only ever draw edges and triangles that form part of the Delaunay triangulation.

Then, the rule for adding triangles becomes:

(x,y) is an edge at radius R if d(x,y) < R/2 AND (x,y) forms an edge in the Delaunay triangulation

FIGURE 23
SCREENSHOT: A SIMPLER EXAMPLE OF THE CONCEPTS ILLUSTRATED BY THE RIPS-COMPLEX GADGET.

As the ability to program with proficiency in languages like JavaScript becomes more common among students, it should be expected that many will welcome the opportunity to bolster both their own education and the departments in which they study. Students invest a significant amount of time during the school year working on projects which only serve to teach one individual. The development of tools like the ones I have shown here would provide students with projects which benefit others long after they have finished the course.

ACKNOWLEDGMENT

I would like to thank Paul Bendich for supporting this project. His suggestions for topics for me to illustrate were greatly appreciated, as was his willingness to work with me to understand the key concepts required for this work. His feedback throughout the preparation of my final presentation was invaluable.

REFERENCES

[1] Paul Bendich. Math 412 Syllabus. 2014.

[2] Mike Bostock. Let's Make a Block. *Bost.ocks.org/mike/block*. 2014.

[3] Michael Bostock, Vadim Ogievetsky, Jeffrey Heer. D3: Data-Driven Documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis), 2011*.

[4] Herbert Edelsbruner and John Harer. *Computational Topology: An Introduction*.

[5] Daniel Shiffman. The Nature of Code. Chapter 6.

[6] V. de Silva and R. Ghrist. Coordinate-free coverage in sensor networks with controlled boundaries via homology. *The International Journal of Robotics Research*, 25(12):1205-1222, 2006.

[7] Diana S. Sinton. Spatial Learning in Higher Education. In Daniel R. Montello, Karl Grossner, and Donald G. Janelle. *Space in Mind: Concepts for Spatial Learning in Education*. Page 220.

[8] Ranxiao Frances Wang. Can Humans Form Four-Dimensional Spatial Representations? In Daniel R. Montello, Karl Grossner, and Donald G. Janelle. *Space in Mind: Concepts for Spatial Learning in Education*. Page 131.