Interactive Geometric Algorithm Visualization in a Browser*

Lynn Asselin[1], Kirk P. Gardner[2], and Donald R. Sheehy[3]

1 **University of Connecticut**
`lynn.asselin@uconn.edu`
2 **University of Connecticut**
`kirk.gardner@uconn.edu`
3 **University of Connecticut**
`don.r.sheehy@gmail.com`

Abstract

We present an online, interactive tool for writing and presenting interactive geometry demos suitable for classroom demonstrations. Code for the demonstrations is written in JavaScript using `p5.js`, a JavaScript library based on Processing.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems—Geometrical problems and computations.

Keywords and phrases Computational Geometry, Processing, JavaScript, Visualisation, Incremental Algorithms

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Motivation

At CG Week 2013 in Rio de Janeiro, Suresh Venkatasubramanian presented at the Workshop on Geometric Computing Challenges with a strong plea for more and better tools for producing interactive pedagogical geometry demos. In his abstract he describes the situation quite clearly:

> The early days of the Web were a goldmine for geometric algorithm demos. The visual and interactive nature of geometric algorithms, coupled with the proliferation of lightweight Java applets, made the HTML canvas a fantastic portable framework for teaching and demonstrations.
> Fast forward to 2013. We have a blizzard of Javascript frameworks, sophisticated HTML5 based in-browser animations, and a host of new platforms on which to demo algorithms. We also have a mature geometric computing platform in the form of CGAL. But we no longer have geometric demos that work well (or at all in many cases).

Here in 2016, the situation is not much different. In this paper, we describe a tool that fills this gap. It is informed by experience collected from teaching an introductory course on Computational Geometry at the University of Connecticut in which students produced such demos as course projects. In most cases, the challenges of **dealing with interaction and visualization dominated the geometric ideas**.

* This work was partially supported by the National Science Foundation under grant CCF-1525978.

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–4

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our interactive geometric algorithm visualizer is designed with the following goals.

1. It should be possible to visualize a geometric algorithm without writing any explicit visualization code.
2. It should be possible to produce an interactive (incremental) demonstration without writing any explicit code to deal with interaction.

Satisfying these two goals implies that all of the visualization and interaction code is abstracted away from the user, who is presented with a simple editor and a canvas. When more complex interaction or visualization is desired, access to these elements is available.

This project in active development, and the source code is available at `https://github.com/compugeom/compugeom.github.io`. A live, interactive sandbox is available at `http://compugeom.github.io`.

2 Geometric Primitives

Drawable classes take the form of primitives as well as data structures, and can be instantiated as geometric objects complete with visualization specifications, as well as any support functions needed to concisely represent the underlying mathematics. These provide a foundation for other data structures and algorithms to be built upon, as well as a natural way to abstract away visualization. The main objects currently provided are:

- Vertices, Edges and Faces - Objects which encapsulate `p5.js` primitives such as *ellipse* and *line* that can be instantiated, visualized, and modified.
- Point sets - In particular, we allow for dynamic point sets which are generated by user input.
- Polygons - Our polygon implementation provides a cyclic Edge list data structure, and allows for easy indexing into the vertices, automatically wrapping indices around using modular arithmetic.
- Planar Straight Line Graphs - A simple half-edge data structure is used to store and traverse the Edges of an embedded planar graph.

As our objective is to construct a means for geometers, students and developers to develop and visualize algorithms in a way that mimics the elegance of the underlying mathematics we adopt a hierarchical paradigm in order to minimize redundancy within the framework. These objects therefore make extensive, but not unnecessary use of lower dimensional primitives and data structures, with visualizations and functionality building naturally off of their component parts. In addition to these, we provide standard linear predicates such as triangle orientation tests (i.e. CCW tests), currently implemented as utility functions within a number of classes.

3 Technical Details

Our goal in this project is not only to build a web-based pedagogical tool, but also to provide a comprehensive and intuitive computational geometry library that allows students, educators and developers to quickly and easily implement well known geometric procedures. The `p5.js` library provides a pure JavaScript implementation of the Processing software sketchbook: an API for rendering graphics such as shapes, lines and ellipses in 2 and 3D. These objects are rendered by placing them in an event loop, however there are no geometric objects provided by the `p5.js` library. In order to implement geometric algorithms in `p5.js/Processing` the provided visualization primitives must therefore be wrapped by classes that encapsulate

geometric primitives. These primitives can then be packaged alongside linear predicates and geometric data structures in order to provide data abstraction and visualization

The first step in constructing such a framework is therefore to handle the instantiation and visualization of drawable objects. This can be accomplished by wrapping the event loop in a data structure that maintains the order of execution and animation of our algorithm visualizer. Thus the `CG_Environment` class serves to abstract away as much of the embedded event loop as possible without restricting the user's access to the `p5.js` library. We have also created geometric primitives as objects, each containing relevant (and natural) properties and functions, as well as the ability to draw themselves when instantiated by a `CG_Environment` object. This is achieved by keeping matrix of objects `CG_Environment.things` sorted by dimension that are to be drawn in the event loop. Objects can also be used strictly as data structures without visualization by instantiating them directly from the library, as in Figure 1

4 Example: Incremental Convex Hull

As a proof of concept, we give an example of a incremental algorithm for computing the convex hull of a set of points. Usually, the term *incremental algorithm* means that the input is processed one element at a time. The primary invariant maintained after processing each element is that the state of the algorithm is a correct output if the current element were the last one. That is, the correctness condition stands in as a loop invariant. An alternative view is that the input arrives online and as each new input arrives, we update the output. This is the more interactive version and it's the perspective we take.

The `CG_DynamicPointSet` class defines a point set for which new points are generated with a mouse click on the canvas. Upon user input the update function is called, passing the new point as a parameter. Thus, to write a program, one need only instantiate a new `DynamicPointSet`, a new `Polygon` (for the output), and write the update function. Some example code is given below.

```
demo.P = demo.DynamicPointSet();
var H = demo.Polygon();
// dynamically compute the convex hull of the point set
demo.P.updateFunction = function(q) {
    if (H.VertexList.length < 2) {
      H.newVertex(q);
      return;
    }
    var left = -1, right = -1;
    for (var i = 0; i < H.VertexList.length; i++) {
      if (H.orient(i, i+1, q) && H.orient(i, i-1, q)){ right = i; }
      if (H.orient(i+1, i, q) && H.orient(i-1, i, q)){ left = i; }
    }
    // if q is outside the hull remove vertices and insert q
    if (left != -1 && right != -1) H.replace(q, left, right);
};
```

The demo object is an instance of the `CG_Environment` class, and is used here to instantiate a `DynamicPointSet` to handle user input and a `Polygon`, the convex hull. The rest of the code is intended to be representative of the algorithm specification by making use of predicates and functions that are natural to the data types in question.

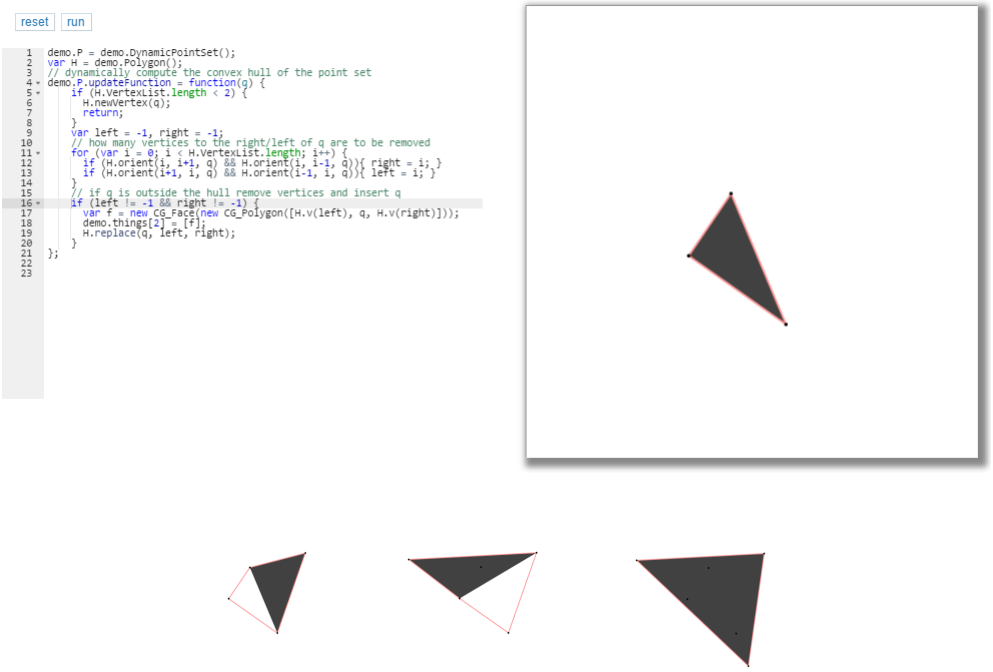

Figure 1 An augmented convex hull algorithm that visualizes the edited region of the polygon by constructing a face, `var f = new CG_Face(new CG_Polygon([H.v(left), q, H.v(right)]))`, and setting it as the only 2D object that is to be draw, `demo.things[2] = [f]`.

Figure 1 illustrates a simple augmentation of the convex hull algorithm that makes use of the `CG_Face` class, taken directly from the web-page. In contrast to the factory function provided by the `CG_Environment` class, here we directly access the matrix, `CG_Environment.things`, of drawn objects in order to ensure we only have one 2D object: the face that represents the most recent change in the hull.

5 Future Work

There are many new features in active development. Among the most immediately useful are methods for visualizing geometric predicates. The main appeal of this project is the ability to quickly and easily implement and visualize geometric algorithms. Predicate visualizations would allow users to not only implement and view the algorithm's output, but also step through them visually. Our goal is that this project will be beneficial to educators and students alike, allowing a clear interactive visual aid to procedures that would otherwise be presented in a textbook or lecture slide.

In addition to predicate visualization we are in the process of integrating support for 3D algorithms and visualizations. By using the perspective and camera controls of `p5.js` we hope to provide an intuitive and painless interface for 3D input processing and visualization. We are also experimenting with other methods for stepping through code for non-incremental algorithms.