CS3110 Spring 2017 Lecture 15: Constructive Synthetic Geometry

Robert Constable

	Date for	Due Date
PS4	Out on March 20	March 30
PS5	Out on April 10	April 24
PS6	Out on April 24	May 8 (day of last lecture)

1 Synthetic Computational Geometry

In the current problem set we are investigating two problems in computational geometry using concepts from our study of the computable real numbers and calculus. The main problem we are examining is the notion of a set of points in the plane and how to compute their *convex hull.* In the last lecture we briefly discussed how it is possible to investigate computational geometry in a more abstract manner, in the spirit of Euclidean geometry. In such an approach we start with an abstract type of Points. We do not need to think that these will be represented as real numbers. We can imagine them as "points in the plane" and state the properties of line segments, angles, triangles, polygons, and so forth using abstract axiomatic mathematics in the spirit of Euclid, but with more precision. This is the way algebra is usually presented as we have seen in looking about the algebraic properties of the rational and reals. Both of these types of numbers satisfy the algebraic properties of a *field.* We can define this concept without reference to any specific instances. We are interested in the general properties of any field.

In the realm of geometry it is also possible to work abstractly and cover non-planar and non-Euclidean geometries as well. For example there is a very deep theory of three dimensional geometry and even n-dimensional geometry. The subject of topology is a generalization of geometric ideas, and the study of homotopy theory is the study of even more abstract geometric structures in high dimension. In several of these geometries there is the analogue problem of convex hulls. It is much harder for most people to visualize these high dimensional spaces. An axiomatic approach makes their study more tractable. The same principle holds for computational geometry. Some concepts and problems can be generalized to more abstract spaces and to higher dimensional geometries. Once we have isolated the key concepts we can express them using constructive logic to bring out the computational content in the most effective way.

This approach to geometry is independent of coordinate systems and numbers, and it is often called *synthetic geometry.* Euclidean geometry is one of the best known examples of synthetic geometry. We don't say what points are. They are treated abstractly. On the other hand, Euclid has in mind an explicit notion of *construction by ruler and compass.* So there is a sense in which some proofs tell us how to perform geometric constructions. The proof of *the very first theorem tells us how to construct an equilateral triangle.* Euclid's second theorem tells us how to simulate a non-collapsing compass with a collapsing compass, a technique widely used once the method is made rigorous by proof. Although Euclid's account of geometry is not thoroughly constructive in the sense that we can actually carry out all of the proofs in such a way that their computational meaning can be implemented, but achieving that is a guiding goal.

One area of active research in geometry has been to make Euclid's proofs more constructive [2, 3, 1]. There are also extensive results on this topic in Nuprl, accessible from the PRL web page, and in Coq as well where they also use non-constructive methods.

We will end this brief account of computational geometry by pointing out how it could be explored using a constructive *synthetic approach.* There are advantages to this in being able to formally prove that various geometric algorithms are correct without resorting to constructive real numbers. This has been done for the convex hull problem and other important examples. We may see more examples of such work because computational geometry is heavily used in a variety of applications, some of them safety critical. One side-effect of the practical importance of this topic and its suitability for functional programming is that there is more interest in revisiting Euclidean geometry to make it significantly more constructive as a basis for computational geometry.

This topic is not yet typically taught in functional programming courses, but we see that it fits well with functional programming. That is an interesting lesson in itself. This approach might become more widely used, but regardless, it is a concept that illustrates how computer science enriches many areas of mathematics and science, even some that are 2,500 years old.

1.1 Synthetic geometry primitives

Here are some of the basic concepts needed to make this approach to geometry both rigorous and constructive. We start with the abstract type of *Points*. This is not an OCaml primitive type, but it is a primitive type in our specification language, and we can easily axiomatize it in the logics of the modern proof assistants. One implementation is the type of pairs of real numbers, e.g. the x and y coordinates in the plane. However, we can also work with an abstract type of points, $Points$. There are an unbounded number of points. The primitive relation on them is *apartness*, $a \neq b$. We say that the points are *separated*. In the previous lecture we showed how to define a notion of equality on points as well. We define $a = b$ as $\neg(a \neq b)$. This approach depends on the following axiom.

1. **Axiom**: $\forall x, y : Points.\ (x \neq y) \ \Rightarrow\ \forall z : Points.\ (z \neq x) \vee (z \neq y)$.
2. $\forall a, b : Points.\ a \neq b$ we write ab for a line *segment*.
3. $\forall a, b, c : Points.\ a \neq b\ \&\ a \neq c\ \&\ b \neq c$ we write $a - b - c$ for *between*, where a, b, and c define a segment on which b is between a and c but never equal to either a or c.
4. $\forall a, b, c : Points.\ a \neq c$ we write $\underline{abc}$ for *non-strict between* where a, b, and c define a segment on which b is between a and c and possibly equal to either a or c.
5. $\forall a, b, c : Points.\ a \neq b\ \&\ a \neq c\ \&\ b \neq c$ we write $a \neq bc$ for a point a separated from a line bc.
6. $\forall a, b, c : Points.\ a \neq b\ \&\ a \neq c\ \&\ b \neq c$ we write $L(a, b, c)$ for a is left-of the segment bc.
7. **Axiom** $\forall a, b, c : Points.\ L(a, b, c) \vee L(a, c, b) \iff a \neq bc$. This captures the idea that a must be to the left of a segment pointing from b to c or to the left of a segment pointing from c to b. In either case, it is required that a is separated from the segment bc, thus the relationship is iff.

In this setting we can describe an algorithm for finding the convex hull that does not use coordinates given by numbers. We are confident that we can form the convex hull in this abstract setting, and the work of Knuth on *Axioms and Hulls* [5] seems to confirm this. It is not clear that Knuth believes this approach can lead to efficient convex hull algorithms, but it is a topic that may be worth further research. We are exploring this question and might have results before the end of the semester, but it is not appropriate to spend time on new research topics in this course. It is appropriate to mention how close we are to an interesting research question. In the lecture I will mention how we came upon this research theme.

One interesting fact about this synthetic approach is that we can use it to describe the slow convex hull algorithm given in *Computational Geometry* [4]. Knuth uses it to describe another convex hull algorithm in his book. Neither of these algorithms is efficient, but they show that it is possible to solve computational geometry problems without directly using calculus. This allows us to think about these algorithms more abstractly in the style of Euclidean Geometry. It is possible that exploring this approach will lead to more intuitive specifications and to synthetic descriptions of algorithms that can be automatically compiled into efficient numerical code.

It is interesting that in Knuth's convex hull algorithm presented abstractly he uses binary search trees. This is a fortuitous coincidence since we will start the study of binary search trees later in this lecture and develop them in the next two or three lectures.

1.2 Slow convex hull algorithm synthetically

The method of the slow convex hull algorithm in the book *Computational Geometry* can be derived using the axiomatic method of *Axioms and Hulls*, and this will open a new approach to computational geometry. We will look at this idea briefly and then move on to talking about binary search trees. In the first lecture after spring break we will revisit this topic and look at a provably correct implementation of a synthetic convex hull algorithm.

We see from the slow convex hull that the specification takes as input a list of points in the region of interest. This can be presented as a numbering of points $p_1, p_2, p_3, ..., p_n$. From these points, segments are formed such as p_1p_2, p_1p_3, ... p_1p_n. The output is a list of segments that determine the convex hull. In the simple algorithm, we actually take the segments in two directions, e.g. p_1p_2, then p_2p_1 and so forth. So we will have both the upper and lower hull among these points. Then for each segment we ask whether there are points of the list defining the hull to the left. If there are none, then we know that the segment is part of the hull. If there are, then we know it is not part of the hull, and we remove it. By this simple method which can be defined with the simple left vs right primitives, we can find all the elements of the convex hull one by one. On the other hand, this is the "slow convex hull" algorithm disparaged in the book on computational geometry [4] we are studying.

Once we have listed all of these segments, we can use our primitive decidable relations to ask for each segment whether there is a point of the region defined by the list of points to the left of this segment. If there is, we know that the segment is not part of the convex hull. If there are no such points, then the segment might be part of the convex hull. We mark those points and then link them together by the relation that p_ip_j links to p_jp_k. However, this link might be tentative. We will also find a link from p_ip_k, in which case we replace the previous link by the direct one.

2 On Cantor's Theorem

In this section we will discuss issues related to the extra credit problem presented in the last lecture.

The question is "how to enumerate the rationals?"

$$\mathbb{N}\times\mathbb{N}\to\mathbb{N} \qquad\qquad \pi(x,y)=\tfrac{1}{2}(x+y)(x+y+1)+y$$

$$\text{inverse} \qquad w=\tfrac{\lfloor\sqrt{8z+1}-1\rfloor}{2}$$

$$t=\tfrac{w^2+w}{2}$$

$$y=z-t,\ x=w-y\ .$$

Another approach,

$$\begin{tikzcd}
\frac{1}{1} \arrow[d] & \frac{1}{2} \arrow[r] & \frac{1}{3} \arrow[dl] & \frac{1}{4} \arrow[r] & \frac{1}{5} \arrow[dl] \\
\frac{2}{1} \arrow[ur] & \frac{2}{2} \arrow[dl] & \frac{2}{3} \arrow[ur] & \frac{2}{4} \arrow[dl] & \\
\frac{3}{1} \arrow[d] & \frac{3}{2} \arrow[ur] & \frac{3}{3} \arrow[dl] & \frac{3}{4} & \\
\frac{4}{1} \arrow[ur] & \frac{4}{2} \arrow[dl] & & \vdots & \\
\frac{5}{1} & & & & \\
\vdots & & & &
\end{tikzcd}$$

Theorem 2.19 Let a_n be a sequence of real numbers and x_0, y_0 reals such that $x_0 < y_0$. Then we can construct a real number x such that $x_0 \le x \le y_0$ and for *all* n, $x \neq a_n$.

References

[1] Jeremy Avigad, Edward Dean, and John Mumma. A formal system for Euclid's Elements. *Review of Symbolic Logic*, 2:700–768, 2009.

[2] Michael J. Beeson. Constructive Geometry. *Proceedings of the tenth Asian logic colloquium*, pages 19–84, 2009.

[3] Michael J. Beeson. Logic of ruler and compass constructions. In S. Barry Cooper, Anuj Dawar, and Benedict Loewe, editors, *Computability in Europe 2012*, Lecture Notes in Computer Science, pages 46–55. Springer, 2012.

[4] M deBerg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry.* Springer, 2008.

[5] Donald E. Knuth. *Axioms and Hulls.* Lecture Notes in Computer Science 606, 1992.