TEACHING MATHEMATICS AND ITS APPLICATIONS, Volume 26, No. 2, 2007 103
Advance Access Published on 7 January 2007 doi:10.1093/teamat/hrl017

Computing areas using Green's Theorem and a Software Planimeter

Paul Davis and Şerban Raianu

Submitted July, 2006; *accepted* September, 2006

Abstract

According to the Merriam-Webster dictionary, a planimeter is 'an instrument for measuring the area of a plane figure by tracing its boundary line'. Even without knowing how a planimeter works, it is clear from the definition that the idea behind it is that one can compute the area of a figure just by 'walking' on the boundary. For someone who has taken calculus, this immediately suggests Green's Theorem. The aim of this note is to clarify for others why this principle works. We do this by using points of view from linear algebra to elementary plane geometry in order to obtain an intuitive justification for Green's Theorem. As an application, we show how the reader can easily construct a 'software planimeter'. The idea behind this is certainly not original; the formula for the area of a polygon used in this process (see Theorem 1.2 and Remark 1.5 (3)) is surely folklore for experienced programmers (1). What we would like to emphasize is, on the one hand, the beautiful interplay between various branches of mathematics and elementary computer graphics in this case. On the other hand, since our application is (in our view) particularly fun to work with, we hope that this approach can be successfully used in class as an aid in teaching Green's Theorem, or in a calculus lab, or even to show younger and more inexperienced students that sometimes deep mathematical results can be surprisingly accessible and entertaining.

1. Computing areas with Green's Theorem

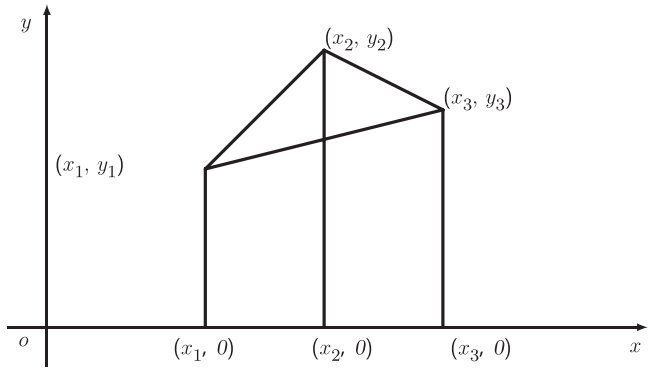
The starting point of our journey is the following result (2, p. 146):

Theorem 1.1

Let (x_1, y_1), (x_2, y_2) and (x_3, y_3) be the vertices of a triangle, oriented clockwise. Then the area of the triangle is given by the formula

$$Area = -\frac{1}{2}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

Proof: We are going to work on the following drawing (note that the case when the projection of (x_3, y_3) on the x-axis lies between the ones of (x_1, y_1) and (x_2, y_2) can be treated similarly):



Denote by T_1 the trapezoid with vertices (x_1, y_1), (x_2, y_2), $(x_2, 0)$ and $(x_1, 0)$, by T_2 the trapezoid with vertices (x_2, y_2), (x_3, y_3), $(x_3, 0)$ and $(x_2, 0)$, and by T_3 the trapezoid with vertices (x_1, y_1), (x_3, y_3), $(x_3, 0)$ and $(x_1, 0)$. It is then clear that the area of the triangle is the area of T_1 plus the area of T_2 minus the area of T_3. This can be written as

$$\begin{aligned} Area &= \frac{1}{2}(y_1 + y_2)(x_2 - x_1) + \frac{1}{2}(y_2 + y_3)(x_3 - x_2) + \frac{1}{2}(y_3 + y_1)(x_1 - x_3) \\ &= -\frac{1}{2}(x_2y_3 - x_3y_2 - x_1y_3 + x_3y_1 + x_1y_2 - x_2y_1) = -\frac{1}{2}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}. \end{aligned}$$

■

It is easy to see that this way of computing areas extends from triangles to arbitrary polygons. We do this in the next result.

Theorem 1.2

Let $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) = (x_1, y_1)$ be the vertices of a polygon, oriented clockwise. Then the area of the polygon is given by the formula

$$Area = \frac{1}{2}\sum_{i=1}^{n-1}(y_i + y_{i+1})(x_{i+1} - x_i).$$

Proof: We have that $\frac{1}{2}(y_i + y_{i+1})(x_{i+1} - x_i)$ is the area of the trapezoid with vertices (x_i, y_i), (x_{i+1}, y_{i+1}), $(x_{i+1}, 0)$ and $(x_i, 0)$, if $x_i < x_{i+1}$, and the negative of that area otherwise. Therefore, when computing the sum in the statement, the areas of the portions outside the polygon will cancel out. ■

Remark 1.3

The sign of the coordinates of the vertices is irrelevant. Nevertheless, in order to justify the use of the drawing from the proof of Theorem 1.1, one can always translate the polygon into the first quadrant.

We are now ready to apply Green's Theorem to computing areas (3, p. 1102). Let us point out that we are not going to give a rigorous proof for this result. Instead, we are going to use the idea from Theorem 1.2 (with rectangles replacing trapezoids) in order to make the result believable.

Theorem 1.4

Let γ be a clockwise-oriented, piecewise-smooth, simple closed curve in the plane, and let D be the region bounded by γ. Then

$$Area(D) = \int_{\gamma} y\,\mathrm{d}x.$$

Proof: Let us first give a heuristic definition of the line integral $\int_{\gamma}$ (vaguely, in the old-fashioned style of the XIXth century, but without mentioning infinitesimals) as follows: let the point of coordinates (x, y) go along the boundary γ clockwise. For each position, consider a small interval of length $\mathrm{d}x$ and compute the area of the thin rectangle of height y and width $\mathrm{d}x$. The right-hand side of the formula in the statement is the sum of all these areas. The fact that this sum is in fact the area of D follows exactly as in the proof of Theorem 1.2: the areas of the portions outside D will cancel out. ■

Remarks 1.5

(1) What happens if we use trapezoids instead of rectangles? Approximating γ by a polygon, considering the formula from Theorem 1.2, and letting the x's be very close, we set $\mathrm{d}x = x_{i+1} - x_i$ and we get that the area of D is

$$\begin{aligned}\frac{1}{2}\sum(y_i + y_{i+1})(x_{i+1} - x_i) &= \frac{1}{2}\Big(\sum y_i(x_{i+1} - x_i) + \sum y_{i+1}(x_{i+1} - x_i)\Big)\\ &= \frac{1}{2}\Big(\int_{\gamma} y\,\mathrm{d}x + \int_{\gamma} y\,\mathrm{d}x\Big),\end{aligned}$$

which is exactly Theorem 1.4.

(2) The formula in Theorem 1.2 is a variation on the well known 'Surveyor's Formula' (see (3, Problem 21 b, p. 1125), or (4, Problem 1, p. 85)):

$$Area(D) = \frac{1}{2}\sum_{i=1}^{n-1}\begin{vmatrix} x_i & y_i \\ x_{i+1} & y_{i+1}\end{vmatrix}.$$

To see this is the same formula, just add and subtract $\sum_{i=1}^{n} x_i y_i$ and regroup terms.

(3) It is perhaps a surprising fact that students do not have to wait until they meet Green's Theorem to see this machinery in action. As early as (3, 11.2), when they study calculus with parametric curves, they could use the same idea to compute the area under a simple curve, given parametrically, starting on the line $x = a$, ending on the line $x = b$ and contained in the region $a \leq x \leq b$, without having to check that the curve is the graph of some function. The fact that the areas of the appropriate regions cancel out enables the students to integrate from 0 to 2π rather than from 0 to π when they compute the area of the ellipse in (3, Problem 31, p. 783), for example.

2. A software planimeter

It should be clear now how to write a program for computing areas of polygons. The only things needed to be used as input are the coordinates of the vertices of the polygon, with the last point equal to the first one. Then we use the formula from Theorem 1.2. The interested reader may download a sample of such a program from the site **http://www.csudh.edu/math/sraianu** by clicking on the 'Download the planimeter' link. The program is written in JAVA, works in any environment (Windows, Mac, Linux) and could be used as follows: show the students how it works (it computes the area of a closed contour after tracing its boundary); ask students how they think it was done, in particular how hard or how long do they think this is; explain how it works using the previous section.

Another possiblity is to encourage the students with some programming experience to write their own version. In this case, the students could be given the following assignment description.

2.1. Statement of scope

A simple software planimeter program should allow a user to load a scaled image, set the correct scale, trace regions of the image with the mouse, compute the resulting areas and display the results.

2.2. Program requirements

The primary objective of our software planimeter is to demonstrate an application of Green's Theorem. To keep things simple, we will only derive requirements that are necessary and sufficient to accomplish our objective and, at the same time, make it entertaining.

The program requirements are as follows:

- The user shall be able to load a JPEG or GIF image.
- The user shall be able to set the scale.
- The user shall be able to use the mouse to trace a closed region of the image.
- The program shall automatically adjust the last data point to ensure closure of the region.
- The program shall apply the formula from Theorem 1.2 to the points collected from the traced region.
- The program shall display the results of the area computation to the user.

The first three requirements pose the greatest challenge. Depending on the selected programming language, a programmer may need detailed knowledge of the JPEG and GIF file formats in order to load the images. Moreover, one would also require a detailed understanding of the mouse's hardware interface in order to develop a reasonable driver. To overcome such obstacles, a high level language is ideal (e.g. JAVA).

2.3. User interface design

Designing a robust graphical user interface (GUI) is a tough challenge. For our purposes, we only need enough capability to meet the requirements put forth in the previous section. In order to achieve this, an appropriate application programming interface (API) must be selected. An API is an interface that permits external software applications to programmatically call functionality

within another software application (e.g. to make windows or use the mouse). This depends on the language and platform(s) selected and is beyond our scope.

2.4. Program flow

2.4.1. Component descriptions

The design of the simple software planimeter should be derived from the requirements laid out in section 2. We will assume the desired API has been selected and skip any discussion of GUI initialization and image loading. Setting the scale and tracing the region(s) also depend on the API selected. We briefly discuss methods for accomplishing these tasks without reference to specific details that will depend on the API selected.

2.4.2. Setting the scale

To set the scale, allow the user to double click any two points on the loaded image, then compute the distance between them. This value will be used later in the algorithm for computing the area. Prompt the user for the units of the scale. For example, after the distance between the two points is computed, the user should tell the program that this distance is equal to, say, 300 miles.

2.4.3. Tracing the region

With the left mouse button depressed, the user should trace a region on the loaded image. The program should capture the coordinates until the left mouse button is released. The user will most likely be unable to end the tracing exactly on the starting point, thus we must make the adjustment automatically. Check the first and last points. If they match, then do nothing, otherwise add the first point to the end of the array of points. With the array of points and the scale we can now implement the formula from Theorem 1.2 to compute the area. The absolute value is used to make sure the result is independent of the orietation used when tracing the boundary.

2.4.4. Computing the area

The algorithm which implements Theorem 1.2 is simple. However, there is one important detail that must not be overlooked. In Theorem 1.2 the indices begin at 1. In many programming languages, indexing of arrays begins with 0. We must make the appropriate adjustment in the algorithm to guarantee the correct result. Observe that re-indexing the sigma formula in the theorem does not do the job.

Example: Consider the storage array of $x[4] = \{1, 2, 3, 4\}$, where $x[0] = 1, \ldots, x[3] = 4$ and $n = 4$. If we want to calculate sum(x[i], 1, n), we cannot just do $x[1] + x[2] + x[3] + x[4]$ because $x[4]$ is undefined. Re-indexing gives us $\text{sum}(x[i+1],\ 0,\ n-1)$. This is still $x[1] + \ldots + x[4]$. We must subtract 1 from the indices when accessing the storage array. That is, $x[1-1] = x[0] = 1$, $x[2-1] = x[1] = 2, \ldots, x[n-1] = x[4-1] = x[3] = 4$. Either summation formula now works.

With the above example in mind, we have the following algorithm: Let x[i] and y[i] be our storage arrays of x and y coordinates with each having length n. Note $i = 0, 1, \ldots, n-1$. Then applying Theorem 1.2, we have

```
for(i = 1; i <= n−1; i++);
total_area += 0.5*(y[i−1] + y[i])*(x[i]−x[i−1]);
total_area = abs(total_area)*scale^2
```

2.5. Restrictions, limitations and constraints

In order to keep the program simple, the program does not take advantage of interpolating algorithms that could be used to improve accuracy. Moreover, the accuracy will vary depending on the accuracy of the images, hand-eye coordination and hardware limits (the mouse).

Acknowledgements

We thank Frédéric Brulois and Frank Miles for helpful comments.

References

1. Bontaş, G. Dăncescu, S. Lustig, A. and Vemuri, M. Private communications.
2. Larson, R. and Edwards, B.H. (2000) *Elementary Linear Algebra*, 4th edn. Boston, New York: Houghton Mifflin Company.
3. Matthews, K. (1991) Elementary Linear Algebra (see http://www.numbertheory.org/book/).
4. Stewart, J. (1999) *Calculus*, 4th edn. Pacific Grove: Brooks/Cole Publishing Company.

Şerban Raianu is a Professor of Mathematics at California State University Dominguez Hills. He is an algebraist mainly interested in studying Hopf algebras acting on algebras and coalgebras and in involving undergraduate students in research projects.

Paul Davis is a Software Engineer at Northrop Grumman Mission Systems. He is mainly interested in General Topology, Geospatial Information Systems and Digital Image Processing. The present paper was written when he was an undergraduate student at California State University Dominguez Hills.

Address for correspondence: Şerban Raianu, California State University Dominguez Hills, Department of Mathematics, 1000 E Victoria St, Carson, CA 90747. Tel. No: 310-243-3139; Fax No: 310-516-3987. E-mail: sraianu@csudh.edu

Paul Davis, E-mail: pdavis2@gmu.edu