

Finding the Area of a Polygon... On a sphere!

Seth Hoffman, GEICO, Chevy Chase, MD

ABSTRACT

SAS/GIS provides a built-in function to calculate the area of a polygon, but Base SAS® 9.4 does not. This paper presents a simple algorithm for calculating the area of any non-self-intersecting polygon. In addition to generic polygons on a flat Cartesian plane, a modified method and additional math is included to make the same calculation for polygons on the surface of a sphere.

INTRODUCTION

While a picture is worth a thousand words, a few numbers are sometimes even better. This is especially the case when doing statistics. One basic way to describe regions like zip codes, counties, or states is their area. Such real world shapes require a general method beyond the simple “base times height” type calculations of grade-school geometry.

Beyond the complication of a city's arbitrary shape, there is also the fact that its borders are all drawn on the surface of an oblate spheroid, aka, the Earth. Calculations involving spheres require the use of trigonometric functions. Cities are relatively small sections of land, so there is not much error introduced if you stick with basic math. As the areas get larger though, spherical trigonometry is required to maintain accuracy.

SAS, SHAPES, AND DRAWING

When talking about shapes, it is helpful to know how to see them. SAS/GRAPH® includes PROC GMAP which can be used to visualize shape data. The procedure uses sets of ordered pairs to create an image. The coordinate variables must be named “x” and “y”. In order to know when to close off one shape and begin another, a variable for polygon IDs is also required.

Code Sample 1

```
DATA foo;
  poly_id = 1; x = 0; y = 0; OUTPUT;
  poly_id = 1; x = 1; y = 0; OUTPUT;
  poly_id = 1; x = 1; y = 1; OUTPUT;
  poly_id = 1; x = 0; y = 1; OUTPUT;
  poly_id = 2; x = 2; y = 0; OUTPUT;
  poly_id = 2; x = 4; y = 0; OUTPUT;
  poly_id = 2; x = 3; y = 2; OUTPUT;
  poly_id = 2; x = 2; y = 2; OUTPUT;
RUN;

PROC GMAP MAP=foo;
  ID poly_id;
  CHORO poly_id;
RUN;
QUIT;
```

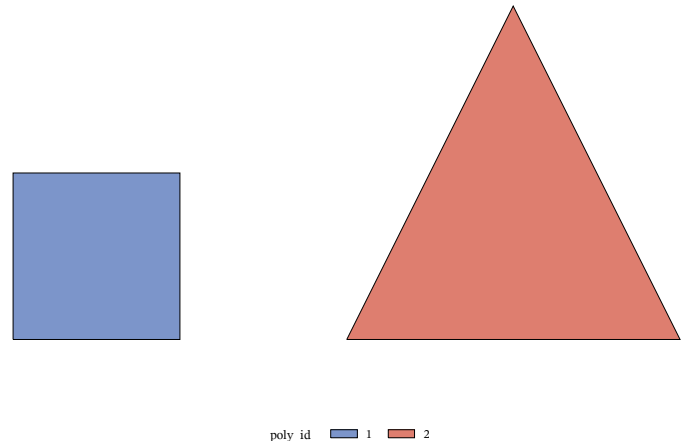


Figure 1. Example output of PROC GMAP

INTEGRATION BY PARTS

As we know from our every-day use of calculus, an integral is the measure of the area under a curve. One way to calculate this area is to slice up the curve into infinitely thin slices and add the area of each slice together. Each infinitesimal slice is essentially a rectangle topped by a right triangle.

Borders of real world areas however, are defined by straight lines between two points whose distance from each other is not infinitesimally small. Because of this, simple arithmetic is all that is needed to calculate area

Step 1:

Starting with the x, y point in the first row make a line to the x, y point in the second row. This line will be the hypotenuse of the triangle.

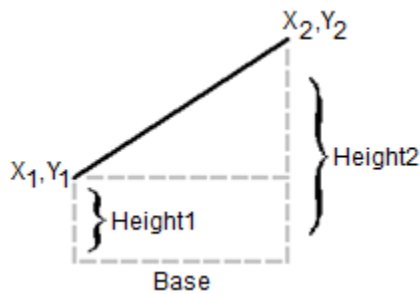


Figure 2. A single slice of a polygon

Step 2:

Repeat step one, adding the area as you go. When the last point is reached, calculate the final slice using the retained value of the first x, y point.

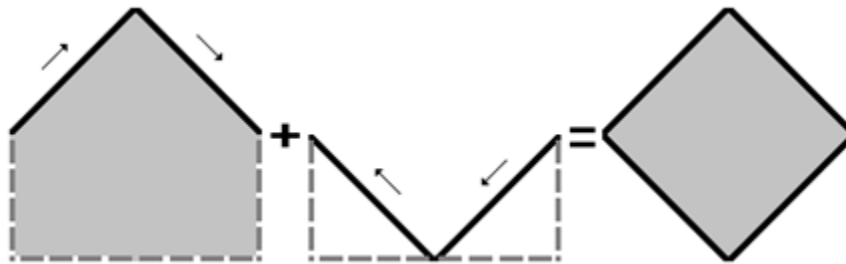


Figure 3. Summing slices of a polygon to give the final area.

The middle area in Figure 3 is added to the area found to the left. Since the direction of travel is in the $-x$ direction, it is adding a negative value. The diamond shape above is a very simple example. Figure 4 shows a slightly more complex shape.

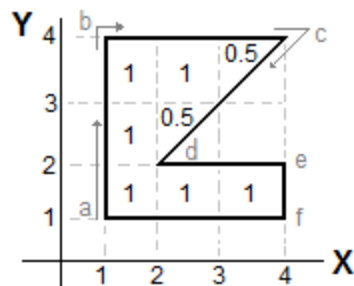


Figure 4. Another example, on graph paper for easy counting.

Using the above method, you can follow the points: $a \rightarrow b$ (0) + $b \rightarrow c$ (12) + $c \rightarrow d$ (-6) + $d \rightarrow e$ (4) + $e \rightarrow f$ (0) + $f \rightarrow a$ (-3) = 7. The same answer is found when counting up the squares contained within the polygon: $1 + 1 + 1 + 1 + 1 + 1 + 1 + 0.5 + 0.5 = 7$. A detail important to correct calculations is that every slice has the same base. One easy method is to find

the lowest 'y' point and make sure it is above the origin line. If any of the points are below the $y = 0$ line, then it is a simple matter to shift the entire polygon up so that the lowest 'y' point will be 0.

Confident that this is a valid algorithm, we can move on to look at the code for a single polygon stored in one data set.

Code Sample 2

```
PROC SQL;
  SELECT CASE WHEN (a.min_y < 0) THEN -1 * a.min_y ELSE 0 END INTO: y_up
  FROM (SELECT DISTINCT MIN(y) AS min_y
        FROM test_polygon) AS a;
QUIT;

DATA area(KEEP=area);
  SET test_polygon END=last_point;
  FORMAT area 6.2;
  RETAIN prev_x prev_y start_x start_y area;
  y = y + &y_up;
  IF _N_ EQ 1 THEN DO;
    prev_x = x; prev_y = y;
    start_x = x; start_y = y;
    area = 0.00;
  END;
  ELSE DO;
    base = x - prev_x;
    height1 = MIN(y, prev_y);
    height2 = MAX(y, prev_y);
    area = ABS(area + (base * height1) + (base * (height2 - height1) / 2));
    prev_x = x; prev_y = y;
  END;
  IF last_point EQ 1 THEN DO;
    base = start_x - prev_x;
    height1 = MIN(start_y, prev_y);
    height2 = MAX(start_y, prev_y);
    area = ABS(area + (base * height1) + (base * (height2 - height1) / 2));
    OUTPUT;
  END;
  ELSE DELETE;
RUN;
```

A keen observer might note that the sign of the area will be different depending on whether the polygon is being traversed clockwise or counter-clockwise. For the sake of efficiency, because SAS data sets only process in one direction, and the rules of space-time dictate that area is a positive value, it is enough to put use the absolute value function on the final area assignment.

FLAT-EARTH CONSPIRACY THEORISTS, STOP READING NOW!

Even on a curved surface, small areas can be treated as flat without a noticeable lack of accuracy. As the size of the area being measured increases, this abstraction will cause increasing errors. Before even talking about how to estimate this error, there is one modification to the above code that needs to be made.

A simple rectangle is defined as a quadrilateral with four right angles. When trying to draw one on the surface of a sphere by using 2 sets of parallel lines, the sum of the 4 angles will always be greater than 360 degrees. Fortunately, this "parallel latitudes and longitudes" rectangle can be cut, diagonally, into two triangles. For the code above, this means each "slice" will now consist of three triangles instead of a triangle and rectangle.

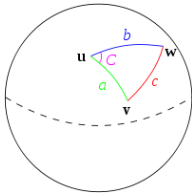


Figure 5. Drawing a triangle on the surface of a sphere

Traditional “flat” triangles have three angles which add up to 180 degrees. For spherical triangles, the three angles will have a sum larger than 180, and the amount by which it is over is called the spherical excess, “E”. If the angles are unknown, but the length of the sides are given, L’Huillier’s Theorem⁽¹⁾ can be used.

$$\tan\left(\frac{1}{4}E\right) = \sqrt{\tan\left(\frac{1}{2}s\right)\tan\left[\frac{1}{2}(s-a)\right]\tan\left[\frac{1}{2}(s-b)\right]\tan\left[\frac{1}{2}(s-c)\right]}$$

Where a, b, c are the sides of the triangle and s is the semi-perimeter $(a + b + c) / 2$.

Another change when working on a sphere is that the Pythagorean theorem is no longer useful for finding distances. Instead, the Haversine formula is the proper choice. In your leisure-time mathematical reading, you will come across many methods of calculating arc segment distances. For the surface of the Earth, the Haversine⁽²⁾ is used. This choice is due to the Earth really being a slightly squashed sphere (aka, an oblate spheroid). Using a formula that does not take this “squashing” into account will introduce error into your data.

If you were to walk around the Earth along the equator, it would take a very long time and some very buoyant shoes. However, walking in a circle around the Earth just below the North Pole may be a matter of feet or inches, depending on how close you are. As such, the trick of translating an entire polygon to make sure it is above the origin line will not work with latitudes and longitudes.

SAS/GRAPH® again comes to the rescue. While the topic of map projections is beyond the scope of this paper, PROC GPROJECT has options that will clip a polygon. Using the LATMIN= and LATMAX= options, any region that extends both above and below the equator can be divided into northern and southern hemisphere parts.

Code Sample 3

```
PROC GPROJECT DATA=test_polygon
    OUT=test_northern_hemisphere
    LONGMIN=-180
    LONGMAX=180
    LATMIN=0
    LATMAX=90;
    ID polygon_id;
RUN;

PROC GPROJECT DATA=test_polygon
    OUT=test_southern_hemisphere
    LONGMIN=-180
    LONGMAX=180
    LATMIN=-90
    LATMAX=0;
    ID polygon_id;
RUN;

DATA spherical_area(KEEP=area);
    SET test_northern_hemisphere;
    FORMAT area 6.2;
    BY segment;
    earth_r = 6378.1; /*a common value of Earth's radius, in kilometers*/
    lon1 = long * 3.1415926535/180; /*Convert degrees to radians*/
    lat1 = lat * 3.1415926535/180;
```

```

IF first.segment THEN DO;
  lat2 = lat1;
  lon2 = lon1;
  area = 0.00;
END;

ELSE DO; /* Lots of math*/
  /* Triangle 1*/
  a1 = SIN((lat1 - lat2) / 2)**2 +
    COS(lat1) * COS(lat2) * SIN((lon1 - lon2) / 2)**2; /*hypotenuse*/
  len_a1 = 2 * ARSIN(MIN(1, SQRT(a1)));

  b1 = SIN(0)**2 + COS(MIN(lat1, lat2)) *
    COS(MIN(lat1, lat2)) * SIN((lon1 - lon2) / 2)**2; /*horizontal*/
  len_b1 = 2 * ARSIN(MIN(1, SQRT(b1)));

  c1 = SIN((lat2 - lat1) / 2)**2 +
    COS(lat1) * COS(lat2) * SIN(0)**2; /*vertical*/
  len_c1 = 2 * ARSIN(MIN(1, SQRT(c1)));

  s1 = (len_a1 + len_b1 + len_c1) / 2;
  e1 = TAN(s1 / 2) * TAN((s1 - len_a1) / 2) *
    TAN((s1 - len_b1) / 2) * TAN((s1 - len_c1) / 2);
  e1 = ATAN(4 * SQRT(e1));
  area1 = e1 * earth_r**2;

  /* Triangle 2*/
  a2 = SIN(MIN(lat1, lat2) / 2)**2 +
    COS(MIN(lat1, lat2)) * COS(0) * SIN((lon2 - lon1) / 2)**2;
  len_a2 = 2 * ARSIN(MIN(1, SQRT(a2)));

  b2 = SIN(0)**2 + COS(0) * COS(0) * SIN((lon2 - lon1) / 2)**2;
  len_b2 = 2 * ARSIN(MIN(1, SQRT(b2)));

  c2 = SIN(MIN(lat1, lat2) / 2)**2 +
    COS(MIN(lat1, lat2)) * COS(0) * SIN(0)**2;
  len_c2 = 2 * ARSIN(MIN(1, SQRT(c2)));

  s2 = (len_a2 + len_b2 + len_c2) / 2;
  e2 = TAN(s2 / 2) * TAN((s2 - len_a2) / 2) *
    TAN((s2 - len_b2) / 2) * TAN((s2 - len_c2) / 2);
  e2 = ATAN(4 * SQRT(e2));
  area2 = e2 * earth_r**2;

  /* Triangle 3*/
  len_a3 = len_a2; /*same hypotenuse*/

  b3 = SIN(0)**2 + COS(MIN(lat1, lat2)) *
    COS(MIN(lat1, lat2)) * SIN((lon2 - lon1) / 2)**2;
  len_b3 = 2 * ARSIN(MIN(1, SQRT(b3)));

  len_c3 = len_c2; /*same vertical*/

  s3 = (len_a3 + len_b3 + len_c3) / 2;
  e3 = TAN(s3 / 2) * TAN((s3 - len_a3) / 2) *
    TAN((s3 - len_b3) / 2) * TAN((s3 - len_c3) / 2);
  e3 = ATAN(4 * SQRT(e3));
  area3 = e3 * earth_r**2;

  IF (lon1 GT lon2) THEN area = area + area1 + area2 + area3;
  ELSE area = area - (area1 + area2 + area3);

```

```
/*retain lat and lon for the next arc segment*/
lat2 = lat1; lon2 = lon1;
END;

/*The last row has the final answer*/
IF last.segment THEN DO;
    area = ABS(area);
    OUTPUT;
END;
END;
RUN;
```

One additional difference in code sample 3 is the final if-then-else statement for each slice. With the simple area calculations in code sample 2, going left would give a negative value and going right would give positive value. In the code above, positive values are returned for each. In order to retain the same addition and subtraction of areas, one direction is arbitrarily chosen to be the positive direction. If the arbitrary decision happens to be incorrect for a particular polygon, that problem is taken care of, again, by the final ABS().

REFERENCES

1. Wolfram MathWorld. "Spherical Excess". (9/9/2016)
<http://mathworld.wolfram.com/SphericalExcess.html>
2. Frank Ivis. "Calculating Geographic Distance: Concepts and Methods" NESUG 2006, Canadian Institute for Health Information, Toronto, Ontario, Canada.
<http://www.lexjansen.com/nesug/nesug06/dm/da15.pdf>

CONTACT INFORMATION

All questions, comments, and suggestions are welcome. Contact the author at:

Name:	Seth W. Hoffman
Work Phone:	301-986-3072
E-mail:	Seth.W.Hoffman@gmail.com

ACKNOWLEDGEMENTS

I would like to thank Meenal Sinha and Brandon Welch, Statistics/Data Analysis Section Chairs for accepting my abstract and paper; Brian Varney SESUG 2016 Academic Chair, Kristen Harrington, SESUG 2016 Operations Chair, members of the South East SAS Users Group (SESUG) Executive board, and SAS Institute for organizing and supporting this conference. I would also like to thank Professor Ranga Myneni of Boston University, without whose generosity and support, I would still think geography was mostly about knowing state capitols.

TRADEMARK CITATIONS

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies.