Method for flagging shaded solar data using a semi-graphical representation

Daniel Perez-Astudillo[1], Dunia Bachour[1] and Antonio Sanfilippo[1]

[1] Qatar Environment and Energy Research Institute, HBKU, Qatar Foundation, Doha (Qatar)

Abstract

Solar radiation monitoring instruments should have unobstructed views to the sun at all times of the day and the year as it moves across the local sky, and measured data should be quality-flagged automatically, as large amounts of data are to be processed. If shading is identified, the data at the shaded areas should be properly flagged and excluded from the 'usable' data; for this, the shaded areas have to be expressed in the data processing code as equations or conditions that define the areas and allow to determine whether any measurement was taken inside the shaded area. There is no established method for identifying and flagging those shaded data when analysing solar radiation measurements. This work presents a method that uses CERN's ROOT Data Analysis Framework to define the shaded area graphically in an efficient and precise way, translating this into a region against which each measurement is then easily checked in the code, and discarded or flagged accordingly.

Keywords: solar resource assessment, data quality, sensor shading

1. Introduction

When measuring solar radiation, the radiometers should have unobstructed views of the sun through the day from sunrise to sunset, throughout the full year. Sometimes, nevertheless, the site conditions are not ideal, and surrounding structures (buildings, trees, etc.) can obscure the sun from the line of view of the radiometers for varying periods of the day along the year. Even if at the time of installation there were no such obstructions, they can appear at later times and be either temporary or permanent. In any case, these obstructions result in incorrect measurements affecting all solar radiation components (Gb, G and Gd, i.e. the direct, global horizontal and diffuse horizontal irradiances, respectively).

To analyse data sets of solar radiation measurements, quality tests following international standards can be applied in order to identify erroneous values in the data and to flag them for their quality; however, the shaded data are not identified by these tests, since a shade cast by a cloud may have the same effect on the measurements as any other shade caused by an unwanted obstruction: Gb is reduced (as much as down to zero, depending on the size and opacity of the cloud or object) while Gd increases correspondingly, and G decreases to a lesser degree. While the data under clouds are realistic measurements, the shaded data resulting from an obstruction on the sensor do not represent the real site conditions and should be removed from the analysis or application.

One useful tool to identify shading issues is the azimuth-elevation (A-E) plot, which presents the irradiance values in a hemispheric sky view (Figure 1). In these graphs, similar to sun charts (Duffie and Beckman, 2013), the measured irradiances are plotted at all daytime moments through the year as function of the solar position in the local sky, covering up to 360° of azimuth in the x-axis, and 0 to 90° of elevation in y. For the northern hemisphere, each day's measurements trace an arc starting on the eastern side at the horizon, rising as the sun moves to the south direction at solar noon and again falling to the horizon towards sunset, but now on the western side; these daily paths are lower and narrower towards winter, and higher and wider towards summer. Since the extremes of the sun path are the summer and winter solstices, the year can be divided at those points so that the data points do not overlap on the plot; this results in two charts, one for each half of the year. Having these 2 charts per year also helps in the observation of persistent patterns of lower irradiance.

Direct normal irradiance, Gb, is much more sensitive to obstructions than the global or diffuse radiations, so in an A-E plot of Gb the obstructions can be more easily identified by eye as large dips in irradiance resembling the

© 2021. The Authors. Published by International Solar Energy Society
Selection and/or peer review under responsibility of Scientific Committee
doi:10.18086/swc.2021.39.01 Available at http://proceedings.ises.org

shape of the obstructing object. Irradiance can also decrease due to atmospheric conditions (mainly clouds or heavy aerosol/dust events), but a human can recognise obstructing patterns, especially with some training and experience. For instance, a cloudy day appears as an arc of lower irradiances, and random clouds appear as also randomly distributed dips, whereas a static obstruction will appear for localised times of the day but spanning through a number of days, potentially weeks or months.

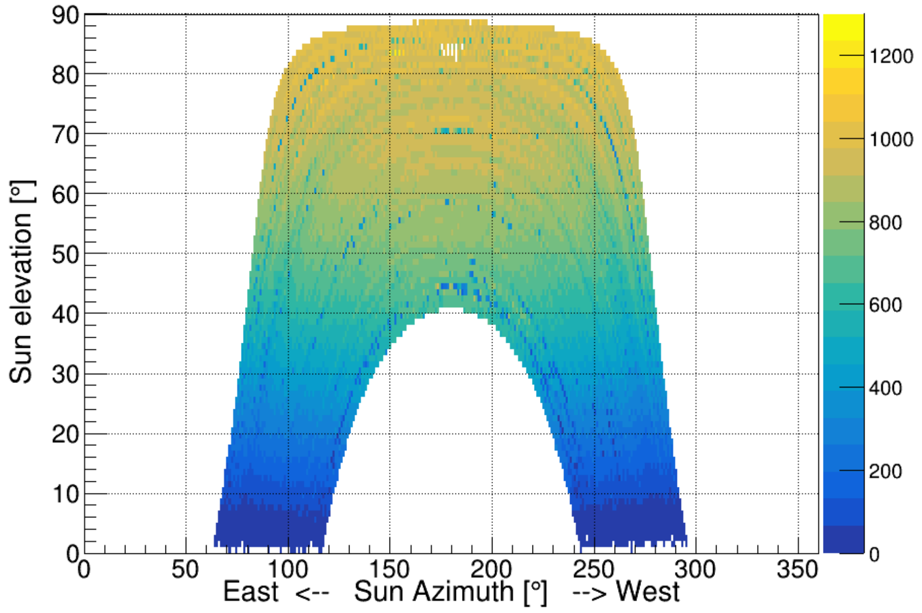


Figure 1: Example of azimuth-elevation graph; the colour scale represents irradiances, in this case G in W/m^2. North is 0 (and 360) degrees in azimuth, while east, south and west are 90, 180 and 270 degrees, respectively.

Once undesired shaded areas have been visually identified, the problem remains of dealing automatically and efficiently with the measurements within those areas, that is, the processing code must determine whether a measurement was taken under the shaded conditions, to quality-flag and filter the data as needed. These shaded areas can be expressed as conditions and/or equations that identify data within them, and these conditions can be expressed as their corresponding solar zenith and azimuth region/s, or as timestamps, in the programming language used for the analysis (C++, MATLAB, Python,…). For the simplest geometries, such as squares or rectangles having their sides parallel to the horizontal and vertical, these conditions can be introduced easily in the code, for instance as a number of “if-then” expressions; however, more complex shapes (often the case of real obstructions) quickly become quite cumbersome to describe; even a rotated square is not simple to describe anymore due to the changing positions in both altitude and azimuth.

In this work, the ROOT data analysis framework (Brun and Rademakers, 1997), developed at CERN and based on C++ (but also integrated with Python and R), is used to facilitate the process of introducing the shaded areas into the data processing and quality-flagging code. Taking advantage of ROOT’s general capabilities, the area is identified through a graphical definition, which is then easily integrated into the programming code to flag data inside this area.

2. Methodology

The ROOT framework contains many libraries and methods to analyse and manipulate data, to produce plots, do simulations, and many more tasks. It should be mentioned that ROOT has not been developed for solar resource applications; in fact, it was initially used for high-energy particle physics, but has enough flexibility for many other uses. Similarly to Matlab, for example, code for ROOT can be typed directly on an interactive prompt or saved in so-called ‘macros’ (e.g. C++ or Python files that use ROOT libraries) that can be executed from the interactive prompt, or in batch mode, or even compiled to executable files.

The key ROOT functionalities for the application in this work are:

- Plots can be displayed in an interactive window called a 'canvas', on which the user can add different elements such as more plots, text, shapes, etc. Then, the elements present in a canvas can be saved as code, so that it can be reproduced by running this code (note that this is independent of any code that may have been used to create that canvas and its elements); this code is automatically generated by ROOT when the user chooses to save the canvas as a macro file.
- On a canvas, the user can draw areas in the form of open or closed polygons, that can be used as 'graphical cuts' and saved as code (basically, as x-y pairs that define the polygon corners) with the canvas.
- The function IsInside(), part of the "TMath" library (TMath, 2021), checks whether a given point (x,y) is inside or outside of a defined polygon.

With these tools, the strategy to introduce the shaded areas to the processing code is straightforward:

- Produce an interactive canvas with an A-E plot for a given irradiance (G, Gb or Gd).
- Visually identify the obstruction area/s and use the "Graphical Cut" tool in ROOT to define an area enclosing the shaded data that should be flagged.
- Save the canvas as .C file and find in it the graphical cut code, through which the cut is created.
- Add the graphical cut code into the macro that does the quality checks on the data. The azimuth and elevation angles of each irradiance data point are passed to the IsInside method of the graphical cut; if the point is inside, flag the entry as 'bad'.

Note that, naturally, in addition to the measured data (G, Gb, Gd), the azimuth and elevation (or the zenith, its complementary) angles of the sun at the corresponding site and time of each measurement must also be available in order to produce the A-E plot, independently of the tool used to make this plot. There are a number of online calculators, with different precisions, from which the solar position can be obtained, and source code is also freely available so users can implement it in their own system; for instance, NOAA compiled a set of simple equations (NOAA, 2017), and NREL has the SOLPOS 2.0 and SPA algorithms (NREL, 2021).

The source code of the IsInside function in ROOT, as all ROOT code, is available online by following the link on the IsInside method description in the TMath library documentation (c.f. above). In essence, the function consists of just one 'for' loop and two nested 'if' conditions; in a simplified way, the strategy is to count how many times a horizontal line passing through the tested point intersects the sides of the polygon to either the right or the left (both give the same result) of the test point. If the final number of crossings on either side is odd, the point is inside the polygon, and outside otherwise. The implementation in ROOT does this by going through each pair of corners, first checking whether the y coordinate of the test point is within the y coordinates of the two corners and if so, and to determine on which side the test point is, the slope between the test point and one corner is compared to the slope of the line of the two corners. Details and graphic examples of this method and other strategies to address the same problem can be found in the literature and online, e.g. in Finley (2007), Bourke (1997), Shimrat (1962), Hormann and Agathos (2001), Wiler (1994), Galetzka and Glauner (2017).

3. Results

Figures 2(a) and 2(b) show A-E plots covering five months of 1-minute measurements of Gb and G, respectively, from a station that has a high tower nearby (photo insert), at an azimuth of about 120° (30° south of east) and up to an elevation of around 35°. Note that for this location, at latitude 25° N, the sun reaches a maximum elevation of around 88° near the summer solstice, but the available data were only from December to April, which is, however, enough to include the full area of the sun paths obstructed by the tower.

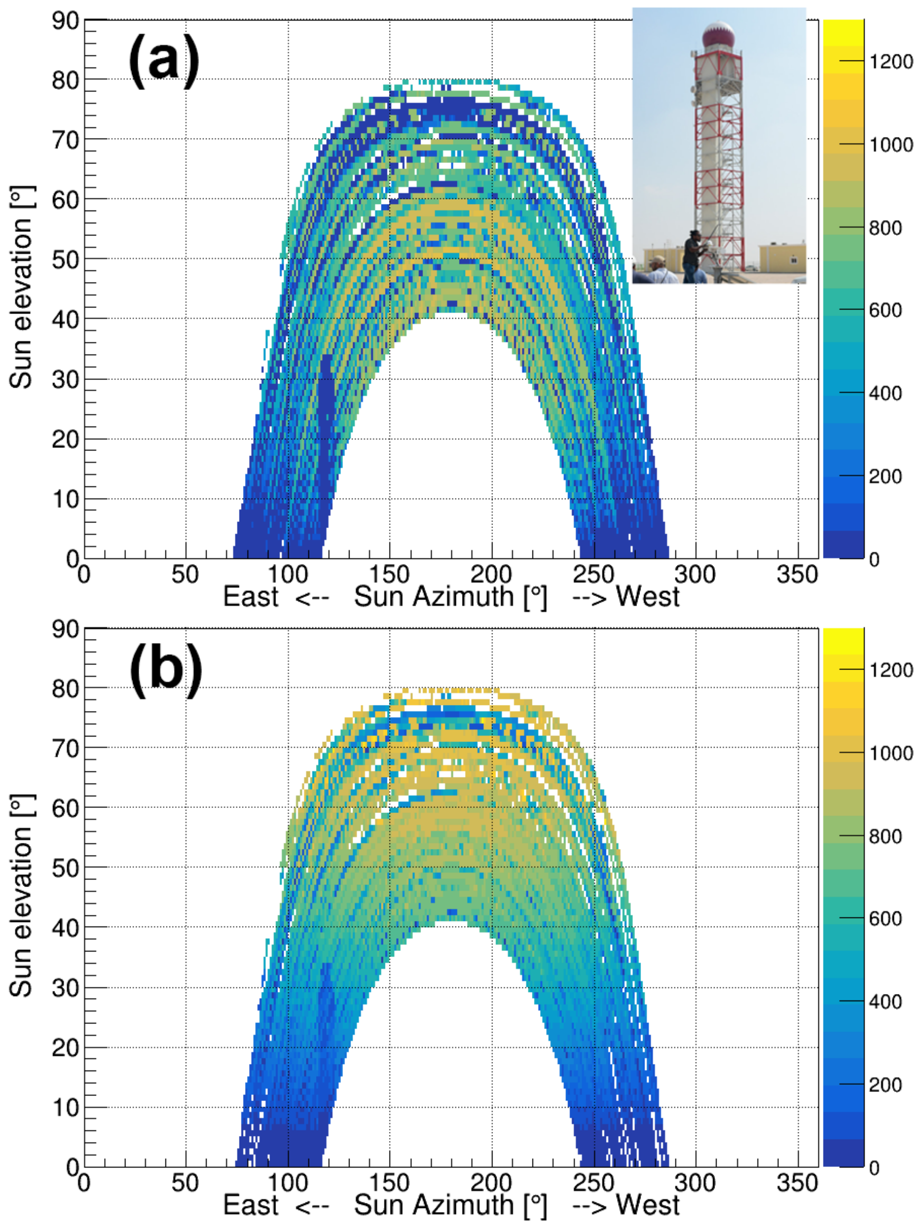
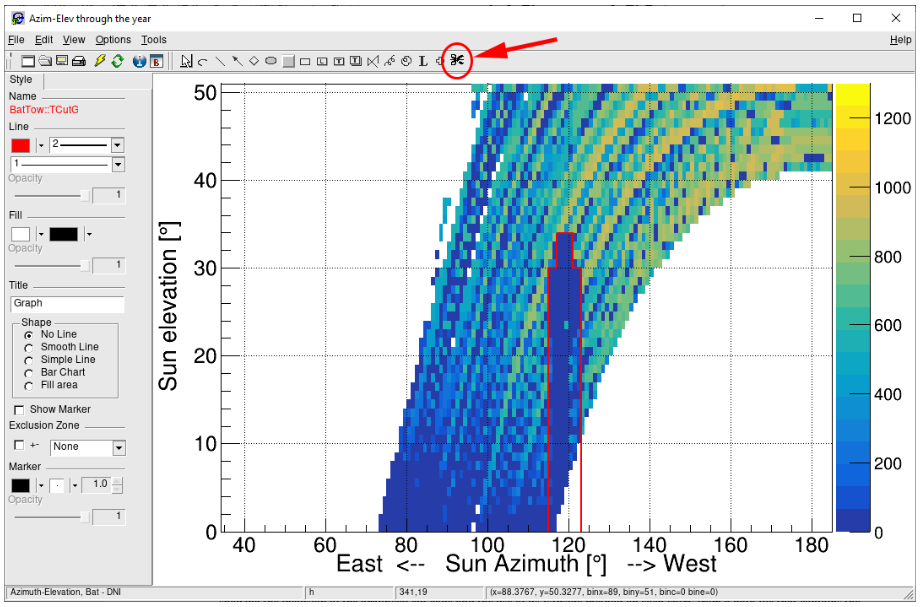


Figure 2: A-E plots of Gb and G, (a) and (b) respectively, from December to April, for a site in which a nearby tower (photo insert) blocks the sensors' line of view to the sun for part of the morning during some months of the year.

The shading from this tower can be seen on the A-E plot of G; however, it is better identified on Gb, so this graph is used to define the graphical cut, first by zooming in on both x and y axes, then selecting the graphical cut tool and clicking on the all points (nine in this case) that define the corners of the resulting red polygon. Figure 3 shows the location of the Graphical Tool on a ROOT canvas Toolbar. Each point is added with a single mouse click (the last one requires a double-click), from which ROOT creates lines to define the polygon sides.

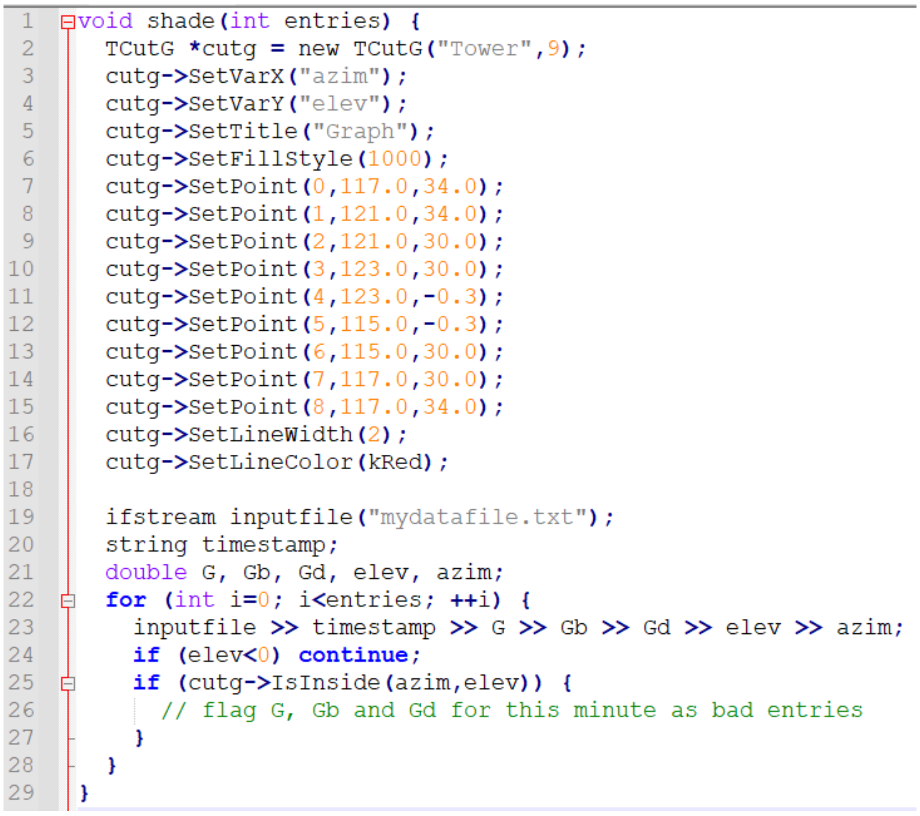

Azim-Elev through the year
File Edit View Options Tools Help
Sun elevation [°]
East <-- Sun Azimuth [°] --> West
0
10
20
30
40
50
40
60
80
100
120
140
160
180
0
200
400
600
800
1000
1200

Style
Name
BatTow::TCutG
Line
2
1
Opacity 1
Fill
Opacity 1
Title
Graph
Shape
[✓ No Line]
Smooth Line
Simple Line
Bar Chart
Fill area
Show Marker
Exclusion Zone
None
Marker
1.0
Opacity 1

Azimuth-Elevation, Bat - DNI | h | 341,19 | (x=88.3767, y=50.3277, binx=89, biny=51, binc=0 bine=0)

Figure 3: The graphical cut tool (indicated with the red arrow) on a canvas. Select the tool and click on each point of the canvas defining a corner of the polygon. The line colour, thickness and style can be changed on the Editor sidebar.

With the polygon created, the canvas can be saved as a ROOT Macro (*.C) file, from which the lines describing the polygon, which start with the definition of a TCutG object, can be retrieved. Figure 4 shows a sample ROOT macro that includes, in lines 2-17, the definition of the graphical cut representing the polygon of Figure 3. Excluding the lines within the 'if IsInside' condition (line 26), this could be a very simple but complete macro with all the user code needed to separate data inside a shaded area; moreover, note that lines 6, 16 and 17, for example, are not needed for this use, but were kept for completeness.



```
 1  void shade(int entries) {
 2    TCutG *cutg = new TCutG("Tower",9);
 3    cutg->SetVarX("azim");
 4    cutg->SetVarY("elev");
 5    cutg->SetTitle("Graph");
 6    cutg->SetFillStyle(1000);
 7    cutg->SetPoint(0,117.0,34.0);
 8    cutg->SetPoint(1,121.0,34.0);
 9    cutg->SetPoint(2,121.0,30.0);
10    cutg->SetPoint(3,123.0,30.0);
11    cutg->SetPoint(4,123.0,-0.3);
12    cutg->SetPoint(5,115.0,-0.3);
13    cutg->SetPoint(6,115.0,30.0);
14    cutg->SetPoint(7,117.0,30.0);
15    cutg->SetPoint(8,117.0,34.0);
16    cutg->SetLineWidth(2);
17    cutg->SetLineColor(kRed);
18
19    ifstream inputfile("mydatafile.txt");
20    string timestamp;
21    double G, Gb, Gd, elev, azim;
22    for (int i=0; i<entries; ++i) {
23      inputfile >> timestamp >> G >> Gb >> Gd >> elev >> azim;
24      if (elev<0) continue;
25      if (cutg->IsInside(azim,elev)) {
26        // flag G, Gb and Gd for this minute as bad entries
27      }
28    }
29  }
```

Figure 4: Code corresponding to the polygon in Figure 3 (lines 1 to 16), and testing each measurement to see whether they were taken inside the azimuth-elevation area defined by that polygon.

Note that the coordinates of the TCutG points can easily be edited by the user, for example to ensure perfectly vertical or horizontal lines if desired (as was done in the example shown here). Lines 18 to 24 in the same figure 4 show how the IsInside method is used to check and flag the azimuth-elevation coordinate pairs: as each irradiance and corresponding solar angles are being read, the solar angles are passed to the IsInside method of the TCutG object, which returns a boolean value of True if the pair (azimuth-elevation) is inside the polygon, and False otherwise. If the point is inside, the corresponding measured irradiances should be directly discarded (if they were going to be used at this point), or flagged as erroneous (if additional processing is to be done).

4. Conclusions

Measuring solar resources requires that, ideally, the radiometers have unobstructed views of the full sky hemisphere. This is not always possible, and some elements in the surroundings may directly block the sun from the line of view of the instruments for parts of the day for varying numbers of days, weeks or months. Given that automatic quality checks are not able to identify these conditions, users must first identify the problem areas and incorporate these in the code before processing the measured solar irradiances, in order to appropriately quality-flag and exclude the affected data. This work presents a method based on the existing capabilities of the ROOT data analysis framework, to easily define through a graphical interface the shaded areas and incorporate them in the processing code, to easily flag measurements taken when the sun was inside that area.

Although the procedure shown here is specific to ROOT, a user of other data analysis tools can translate the methodology used here to their own data analysis framework. For example, the points of the polygon could be defined ‘by hand’ and stored in vectors or arrays, and then a similar IsInside method can be implemented in any language, as the code is openly accessible. However, and as mentioned previously, ROOT already provides the IsInside method, and has the additional advantage of allowing to define the polygon in an interactive, graphical way, by simply clicking on the plot, an advantage that quickly becomes more convenient when the shaded area is not a simple geometrical shape.

5. References

Bourke, 1997. https://www.eecs.umich.edu/courses/eecs380/HANDOUTS/PROJ2/InsidePoly.html (last access: 11 October, 2021).

Brun, R., Rademakers, F., 1997. ROOT - An Object Oriented Data Analysis Framework. Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86.

Duffie, J. A., Beckman, W. A., 2013. Solar engineering of thermal processes, 4th ed. Wiley & Sons, New Jersey. Examples in pages 30-34.

Finley, 2007. http://alienryderflex.com/polygon (last access: 11 October, 2021).

Galetzka, M., Glauner, P., 2017. A Simple and Correct Even-Odd Algorithm for the Point-in-Polygon Problem for Complex Polygons. Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2017), Volume 1: GRAPP, Pages 175-178. arXiv:1207.3502v2 ([cs.CG] - Computational Geometry).

Hormann, K., Agathos, A., 2001. The point in polygon problem for arbitrary polygons. Computational Geometry. 20 (3): 131. doi:10.1016/S0925-7721(01)00012-8.

NOAA, 2017. http://www.esrl.noaa.gov/gmd/grad/solcalc/solareqns.PDF (last access: 11 October, 2021).

NREL, 2021. https://midcdmz.nrel.gov/solpos (last access: 11 October, 2021).

Shimrat, M., 1962. Algorithm 112: Position of point relative to polygon. Communications of the ACM Volume 5 Issue 8, Aug. 1962.

TMath, 2021. https://root.cern/doc/master/namespaceTMath.html (last access: 11 October, 2021).

Weiler, K., 1994. An Incremental Angle Point in Polygon Test, in: Heckbert, P. S. (Ed.), Graphics Gems IV. San Diego, CA, USA: Academic Press Professional, Inc., pp. 16–23, ISBN 0-12-336155-9.