POLYPACK, A PACKAGE OF ROUTINES TO MANIPULATE POLYGONS

David J. Kennison
NCAR, P.O. Box 3000, Boulder, Colorado 80307-3000
email: kennison@ncar.ucar.edu (128.117.64.4)

POLYPACK, A PACKAGE OF ROUTINES TO MANIPULATE POLYGONS

David J. Kennison
NCAR, P.O. Box 3000, Boulder, Colorado 80307-3000
email: kennison@ncar.ucar.edu (128.117.64.4)

1.0 INTRODUCTION

This document describes a small package of NCAR Graphics routines that allow a user to manipulate polygons in various ways. Each of the six principal routines is given a "clip" polygon, a "subject" polygon, and a workspace; it operates on the two polygons in some way to create either a derivative polygon or a set of trapezoids representing the interior of that derivative polygon and then passes these on to a user-specified processing routine.

The six basic routines provided by POLYPACK are as follows:

- **PPDIPO** generates and returns the boundary of the "difference" polygon, which consists of all points that are inside the subject polygon but not inside the clip polygon.
- **PPINPO** generates and returns the boundary of the "intersection" polygon, which consists of all points that are inside both the clip polygon and the subject polygon.
- **PPUNPO** generates and returns the boundary of the "union" polygon, which consists of all points that are inside either or both of the clip polygon and the subject polygon.
- **PPDITR** generates the difference polygon but returns a set of trapezoids representing its interior.
- **PPINTR** generates the intersection polygon but returns a set of trapezoids representing its interior.
- **PPUNTR** generates the union polygon but returns a set of trapezoids representing its interior.

Other POLYPACK routines are as follows:

- **PPPPAP** can be called to preprocess a polygon in such a way as to remove certain peculiarities that can cause minor cosmetic errors in the output from the routines that return trapezoids.
- **PPPLCL** will clip a polyline against a clipping rectangle. It is intended that, eventuallly, there should be a better routine that will clip a polyline against an arbitrary polygon, but that routine has not yet been written.

All of the routines mentioned above are discussed in detail below, in the section named "ROUTINES".

Currently, POLYPACK will accept as input only simple polygons. A simple polygon is defined by an ordered collection of points P_1, P_2, P_3, ... P_{m-1}, P_m, where P_m=P_1. The edge segments of the polygon are P_1P_2, P_2P_3, P_3P_4, ... $P_{m-2}P_{m-1}$, $P_{m-1}P_m$. A point in the plane is part of the interior of the polygon if and only if a half line emanating from the point passes through the edges of the polygon an odd number of times. Note that this definition requires that the boundary of the polygon be a single simple curve, traceable without lifting "pen" from "paper".

To represent a more complicated polygon (one that is in two or more separate pieces or that has one or more holes in it), one uses a simple polygon, having some extra edge segments, that has the same interior. For example, a polygon consisting of two separate pieces having simple boundaries $P_1P_2P_3$ ... P_m, where P_1=P_m, and $Q_1Q_2Q_3$ ... Q_n, where Q_1=Q_n, may be represented by $P_1P_2P_3$ ... $P_mQ_1Q_2Q_3$ ... Q_nP_1. In the same way, a "doughnut" with an outer edge $P_1P_2P_3$ ... P_m and a hole $Q_1Q_2Q_3$ ... Q_n may also be represented by $P_1P_2P_3$ ... $P_mQ_1Q_2Q_3$ ... Q_nP_1. The process can be generalized: for example, if the boundary of a complicated polygon consists of the three disconnected pieces $P_1P_2P_3$ ... P_m, $Q_1Q_2Q_3$ ... Q_n, and $R_1R_2R_3$ ... R_k, then the boundary of the simple equivalent polygon is $P_1P_2P_3$ ... $P_mQ_1Q_2Q_3$ ... $Q_nP_1R_1R_2R_3$ ... R_kP_1.

The Fortran representation of an m-point polygon requires two arrays of length "m" - one to hold the X coordinates of the points and one to hold the Y coordinates of the points. On input to a POLYPACK routine, the final pair of coordinates may be a duplicate of the first pair or not, whichever is more convenient for the user. (If one intends both to fill the polygon and to draw its boundary, it is convenient to repeat the first point at the end.) Output polygons from POLYPACK always have a final pair of coordinates that is a duplicate of the initial pair.

The POLYPACK routines that return output polygons - PPDIPO, PPINPO, and PPUNPO - do not automatically merge disconnected pieces of the boundary in the manner described in the preceding paragraph. The user-provided output routine will receive the separate, disconnected pieces of the boundary in separate calls; for some purposes, it will be necessary for the routine to merge these into a single boundary. See the example described in the section "EXAMPLES".

2.0 ROUTINES

2.1 SUBROUTINES PPDIPO, PPINPO, AND PPUNPO

2.1.1 Usage

Each of the FORTRAN statements

```
CALL PPDIPO (XCCP,YCCP,NCCP,XCSP,YCSP,NCSP,RWRK,IWRK,NWRK,URPP,IERR)
CALL PPINPO (XCCP,YCCP,NCCP,XCSP,YCSP,NCSP,RWRK,IWRK,NWRK,URPP,IERR)
CALL PPUNPO (XCCP,YCCP,NCCP,XCSP,YCSP,NCSP,RWRK,IWRK,NWRK,URPP,IERR)
```

causes the formation of a polygon (the difference polygon, the intersection polygon, or the union polygon, respectively) and the delivery of that polygon's boundary, piece by piece, to the user-specified polygon-processing routine URPP.

The difference polygon is the subject polygon minus the clip polygon, rather than the other way around.

2.1.2 Arguments

XCCP and **YCCP** (input arrays of type REAL) are the X and Y coordinate arrays defining the clip polygon.

NCCP (an input expression of type INTEGER) is the number of points defining the clip polygon.

XCSP and **YCSP** (input arrays of type REAL) are the X and Y coordinate arrays defining the subject polygon.

NCSP (an input expression of type INTEGER) is the number of points defining the subject polygon.

RWRK and **IWRK** (scratch arrays, dimensioned NWRK, of types REAL and INTEGER, respectively) are workspace arrays. Because of the way in which these are used, RWRK and IWRK may be EQUIVALENCEd (and, to save space, they should be).

NWRK (an input expression of type INTEGER) is the length of the workspace array(s). It is a bit difficult to describe how much space might be required. At the moment, I would recommend using NWRK equal to about ten times the total of the number of points in the input polygons and the number of intersection points. This situation will change with time; at the very least, I would like to put in an internal parameter that will tell one how much space was actually used on a given call, but I have not yet done so.

URPP is the name of a user-provided routine to process the polygon-boundary pieces. This name must appear in an EXTERNAL statement in the routine that calls PPDIPO/PPINPO/PPUNPO and the routine itself must have the following form:

```
SUBROUTINE URPP (XCRA,YCRA,NCRA)
  DIMENSION XCRA(NCRA),YCRA(NCRA)
  ...(code to process polygon boundary piece defined by arguments)...
  RETURN
END
```

Each of the arguments XCRA and YCRA is a real array, dimensioned NCRA; the former holds the X coordinates, and the latter the Y coordinates, of a piece of the polygon boundary. It will be the case that XCRA(NCRA)=XCRA(1) and YCRA(NCRA)=YCRA(1).

IERR (an output variable of type INTEGER) is returned with the value zero if no errors occurred in the execution of PPDIPO/PPINPO/PPUNPO or with a small positive value if an error did occur. The value 1 indicates that a degenerate clip polygon was detected, the value 2 that a degenerate subject polygon was detected, and the value 3 that the workspace provided was too small; values greater than 3 should be reported to the author, as they indicate some problem with the algorithm. Currently, if IERR is returned non-zero, one can be sure that no calls to URPP were executed; in the future, this could change, but, in that case, there will be an internal parameter allowing one to request the current behavior.

2.2 SUBROUTINES PPDITR, PPINTR, AND PPUNTR

2.2.1 Usage

Each of the FORTRAN statements

```
CALL PPDITR (XCCP,YCCP,NCCP,XCSP,YCSP,NCSP,RWRK,IWRK,NWRK,URPT,IERR)
CALL PPINTR (XCCP,YCCP,NCCP,XCSP,YCSP,NCSP,RWRK,IWRK,NWRK,URPT,IERR)
CALL PPUNTR (XCCP,YCCP,NCCP,XCSP,YCSP,NCSP,RWRK,IWRK,NWRK,URPT,IERR)
```

causes the formation of a polygon (the difference polygon, the intersection polygon, or the union polygon, respectively) and the delivery of a set of trapezoids comprising the interior of that polygon, one at a time, to the user-specified trapezoid-processing routine URPT.

The difference polygon is the subject polygon minus the clip polygon, rather than the other way around.

2.2.2 Arguments

XCCP and **YCCP** (input arrays of type REAL) are the X and Y coordinate arrays defining the clip polygon.

NCCP (an input expression of type INTEGER) is the number of points defining the clip polygon.

XCSP and **YCSP** (input arrays of type REAL) are the X and Y coordinate arrays defining the subject polygon.

NCSP (an input expression of type INTEGER) is the number of points defining the subject polygon.

RWRK and **IWRK** (input arrays, dimensioned NWRK, of types REAL and INTEGER, respectively) are workspace arrays. Because of the way in which these are used, RWRK and IWRK may be EQUIVALENCEd (and, to save space, they should be).

NWRK (an input expression of type INTEGER) is the length of the workspace array(s). The routines that produce trapezoids use a lot less workspace than the ones that produce polygons. The amount of space used at a particular time during the execution of one of these routines is roughly ten or eleven times the number of intersections of a horizontal "scan line" with the edge segments making up the input polygons, plus about three times the number of points above the current scan line that are "local minima". (A "local minimum" is a point on one of the polygons that is connected to two other points on the polygon having larger Y coordinates than its own). I have not yet developed a rule of thumb for setting the value of NWRK. Ultimately, I would like to at least put in an internal parameter that will tell one how much space was actually used on a given call, but I have not yet done so.

URPT is the name of a user-provided routine to process the trapezoids. This name must appear in an EXTERNAL statement in the routine that calls PPDITR/PPINTR/PPUNTR and the routine itself must have the following form:

```
SUBROUTINE URPP (XCBL,XCBR,YCOB,DXLE,DXRE,YCOT)
  ...(code to process the trapezoid defined by the arguments)...
  RETURN
END
```

The bottom and top of the trapezoid are horizontal (parallel to the X axis); the arguments XCBL and XCBR define the X coordinates of its bottom left and bottom right corners, YCOB is the Y coordinate of its bottom edge, DXLE and DXRE are the inverses (dx/dy) of the slopes of its left and right edges, and YCOT is the Y coordinate of its top edge. The corners of the trapezoid are therefore as follows: (XCBL,YCOB), (XCBR,YCOB), (XCBL+DXLE*(YCOT-YCOB),YCOT), and (XCBR+DXRE*(YCOT-YCOB),YCOT).

IERR (an output variable of type INTEGER) is returned with the value zero if no errors occurred in the execution of PPDITR/PPINTR/PPUNTR or with a small positive value if an error did occur. The value 1 indicates that a degenerate clip polygon was detected, the value 2 that a degenerate subject polygon was detected, and the value 3 that the workspace provided was too small; values greater than 3 should be reported to the author, as they probably indicate some problem with the algorithm. If IERR is returned non-zero, one can be sure that there have been no calls to URPT only if IERR = 1 or 2; otherwise, some trapezoids probably have been generated and delivered to URPT.

2.3 SUBROUTINE PPPPAP

2.3.1 Usage

The FORTRAN statement

```
CALL PPPPAP (XCIP,YCIP,NCIP,NBTS)
```

causes preprocessing of the X and Y coordinates of the points defining a polygon which is to be used as input to one of the principal POLYPACK routines that produce trapezoids. The object is to cure an annoying (but basically cosmetic) problem that sometimes occurs.

The nature of the problem is as follows: Sometimes, when adjacent points have Y coordinates that differ only very slightly, there will be, among the output trapezoids, degenerates, of essentially zero height, that stick out to the left or right from the body of the polygon of which each trapezoid is a part. This happens because, in the calls to the user-defined trapezoid-processing routine URPT, the values of DXLE and DXRE become very large and the difference between the values of YCOT and YCOB becomes very small; in URPT, then, to get the X coordinates at the ends of the top of the trapezoid, the very large values and the very small values are multiplied together and the result is highly inaccurate.

What PPPPAP does is this: Each of the input coordinates is first modified by zeroing out all but the first NBTS bits of its fractional part; then, any point with the same coordinates as the preceding point is culled. This ensures that there are no adjacent points with X or Y coordinates that are so nearly identical as to cause the observed problem, but has little real effect on the values of the coordinates.

There are several reasons why this was not done automatically:

- Because it's a little time-consuming (zeroing the low-order bits in a way that doesn't violate the Fortran-77-standard is a bit difficult), it shouldn't be done if it doesn't have to be done; the user may have reason to know that the problem doesn't arise.
- The problem may occur in one of the polygons, but not the other (most likely, in the subject polygon, but not in the clip polygon).
- The user knows best what precision needs to be maintained in the data.

2.3.2 Arguments

XCIP and **YCIP** (input/output arrays of type REAL) are the X and Y coordinate arrays defining a polygon to be used as input to one of the routines PPDITR, PPINTR, or PPUNTR. PPPPAP will alter the contents of these arrays.

NCIP (an input/output variable of type INTEGER) is the number of points defining the input polygon. PPPPAP may reduce the value of NCIP.

NBTS (an input expression of type INTEGER) is the number of significant bits to be left unzeroed in the fractional part of each coordinate. Generally, one would probably not want to use a value less than 10 or 12. On a 32-bit machine on which reals have 24-bit fractions, 18 may be a good choice; on a 64-bit machine with 48-bit fractions, larger values may be desirable.

2.4 SUBROUTINE PPPLCL

2.4.1 Usage

The FORTRAN statement

```
CALL PPPLCL (XMIN,XMAX,YMIN,YMAX,XCPL,YCPL,NCPL,RWRK,LRWK,URPF,IERR)
```

causes a user-defined polyline to be clipped against a user-defined clipping rectangle. The pieces of the polyline resulting from the clipping process are delivered, one at a time, to a user-specified routine for processing.

2.4.2 Arguments

XMIN, **XMAX**, **YMIN**, and **YMAX** (input expressions of type REAL) are the X and Y coordinate values defining the clipping rectangle.

XCPL and **YCPL** (input arrays of type REAL) are the X and Y coordinate arrays defining the polyline that is to be clipped.

NCPL (an input expression of type INTEGER) is the number of points defining the input polyline (the number of meaningful elements in each of the arrays XCPL and YCPL).

RWRK (a scratch array of type REAL) is a workspace array for PPPLCL to use. It is divided into halves; one half is used for X coordinates and the other half for Y coordinates. This array doesn't have to be too big unless it is important not to break the polyline except at the edges of the clipping rectangle.

LRWK (an input expression of type INTEGER) is the length of the array RWRK.

URPF is the name of a user-provided routine to process the pieces of the polyline resulting from the clipping process. This name must appear in an EXTERNAL statement in the routine that calls PPPLCL and the routine itself must have the following form:

```
SUBROUTINE URPF (XCRA,YCRA,NCRA)
  DIMENSION XCRA(NCRA),YCRA(NCRA)
  ...(code to process the polyline defined by the arguments)...
  RETURN
END
```

Each of the arguments XCRA and YCRA is a real array, dimensioned NCRA; the former holds the X coordinates, and the latter the Y coordinates, of a piece of the polyline.

IERR (an output variable of type INTEGER) is returned with the value zero if no errors occurred in the execution of PPPLCL or with a small positive value if an error did occur. The value 1 indicates that NCPL is less than or equal to zero and the value 2 indicates that LRWK is less than 4.

3.0 INTERNAL PARAMETERS

As of now, POLYPACK has no internal parameters. There are some planned extensions that may involve adding internal parameters and access routines for them.

4.0 ERROR HANDLING

POLYPACK routines in which detectable errors can occur return an error code that, when non-zero, indicates that an error has occurred.

5.0 EXAMPLES

On a Unix system on which NCAR Graphics has been installed, the command “ncargex” may be used to acquire the code for and run examples called “ppex01” and “tppack” that demonstrate the use of various POLYPACK routines. Each of these examples includes a routine FILLPO that fills polygons, a routine FILLTR that fills trapezoids, and a routine MERGPO that illustrates how to merge output polygons in such a way as to avoid problems on devices with deficient fill algorithms.

6.0 MISCELLANY

POLYPACK is based on algorithms described by Bala R. Vatti in the article “A Generic Solution to Polygon Clipping”, in “COMMUNICATIONS OF THE ACM”, July, 1992. I do not handle horizontal segments quite as Vatti suggests (actually, the description of this, like most of the article, was a bit sketchy), but the technique I devised seems to work well.

The POLYPACK code was written in the summer of 1993. The master version is written in IFTRAN, which has many advantages for me; as a result of this, though, the checked-in FORTRAN source code is of the less-than-attractive variety typical of IFTRAN output. If the reader wishes to extend the code in some manner, he/she would be better off getting a copy of the IFTRAN source and working from that, rather than trying to work from the FORTRAN code.

In the early summer of 1994, a co-worker decided to use POLYPACK routines to clip polygons in the NCAR Graphics version of GKS. This led to my putting the code through a severe testing regimen that left me very confident of its correctness and robustness; however, it is certainly possible that errors remain and I would be interested in receiving programs (preferably simple ones) that generate errors.

The POLYPACK routines are quite fast and could probably be made even faster by taking out a collection of error checks for “algorithm failures” that **probably** can’t occur, but I am reluctant to take these out until the code has gone through a longer shakedown period.