File created: 14-May-2023 15:47:43 **{DSK}<home>larry>il>medley>sources>XXGEOM.;5**

edit by: lmm

changes to: (MACROS GETLINEORIG)

previous date: 13-Jun-2021 14:39:29 {DSK}<home>larry>il>medley>sources>XXGEOM.;4

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

```
(RPAQQ XXGEOMCOMS
       (
```

;;; Integer Geometry Library

;;; Scalar methods

```
        (DECLARE%: EVAL@COMPILE DONTCOPY (MACROS \SGN))
        (FNS \IRND)
```

;;; XYpt object and methods

```
        (RECORDS XYPT)
        (DECLARE%: EVAL@COMPILE DONTCOPY (MACROS XYPT.X XYPT.Y))
        (DECLARE%: EVAL@COMPILE DONTCOPY (MACROS HEADPT NEXTPT HEADPTY NEXTPTY))
        (FNS MAKEXYPT IRNDLIST NORMLOOP SLITLOOP PREPLOOP YMAPLIST IMAPLIST UNIQLIST MERGLIST MMLTLIST IMLTLIST
             XYPT.LESSP PATH.LESSP CONVEXP)
```

;;; Line object and methods

```
        (RECORDS XXLINE)
        (DECLARE%: EVAL@COMPILE DONTCOPY (MACROS GETLINEDIFF GETLINEORIG \GETLINEDIFF \GETLINEORIGY
                                                 \GETLINEDIFFY))
        (FNS MAKELINE MSECT XSECT YSECT XYSECT KNOTLINE KNOTLOOP LINE.LESSP)
        (FNS LINEY MIDDX INITX TERMX SCANX XPROD)
```

;;; line segment methods

```
        (FNS XYSECTLSEG)
```

;;; Bresenham line object and methods

```
        (RECORDS BRES)
        (FNS MAKEBRES)
        (DECLARE%: EVAL@COMPILE DONTCOPY (MACROS BRESSTEP))
```

;;; Debugging control panel

```
        (DECLARE%: EVAL@COMPILE DONTCOPY (VARS \GEOM.PANEL))
```

;;; Trapezoidal decomposition

```
        (FNS TRAPLOOP TRAPMAKE)
        (VARS TRAP.DEBUG)
        (DECLARE%: EVAL@COMPILE DONTCOPY (VARS \TRAP.PANEL))))
```

;;; Integer Geometry Library

;;; Scalar methods

```
(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: EVAL@COMPILE

(PUTPROPS \SGN DMACRO ((VAL)
                       (if (IGREATERP VAL 0)
                           then 1
                         elseif (ILESSP VAL 0)
                           then -1
                         else 0)))
)
)

(DEFINEQ
```

(\IRND

```
  [LAMBDA (N D)                                          ; Edited 24-Aug-87 22:14 by FS
    ;; integer round operation
    (if (MINUSP D)
        then (SETQ N (IMINUS N))
             (SETQ D (IMINUS D)))
    (IQUOTIENT (if (MINUSP N)
                   then (IDIFFERENCE N (LRSH D 1))
                 else (IPLUS N (LRSH D 1)))
          D])

)


;;; XYpt object and methods

(DECLARE%: EVAL@COMPILE

(RECORD XYPT (X . Y))
)

(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: EVAL@COMPILE

(PUTPROPS XYPT.X DMACRO ((PT)
                         (CAR PT)))

(PUTPROPS XYPT.Y DMACRO ((PT)
                         (CDR PT)))
)
)

(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: EVAL@COMPILE

(PUTPROPS HEADPT DMACRO ((PATH)
                         (CAR PATH)))

(PUTPROPS NEXTPT DMACRO ((PATH)
                         (CADR PATH)))

(PUTPROPS HEADPTY DMACRO ((PATH)
                          (CDAR PATH)))

(PUTPROPS NEXTPTY DMACRO ((PATH)
                          (CDADR PATH)))
)
)

(DEFINEQ
```

(MAKEXYPT

```
  [LAMBDA (IX IY)                                        (* FS "10-Feb-86 12:11")

          (* * Create and return an XYPT object)

    (create XYPT
            X _ IX
            Y _ IY])
```

(IRNDLIST

```
  [LAMBDA (PATH)                                         (* FS "10-Feb-86 12:14")

          (* * Return integer version of list (should round) * *)

    (LET (X Y)
         (for I in PATH collect (SETQ X (XYPT.X I))
                                (SETQ Y (XYPT.Y I))
                                [if (NOT (FIXP X))
                                    then (SETQ X (FIX (PLUS X 0.5]
                                [if (NOT (FIXP Y))
                                    then (SETQ Y (FIX (PLUS Y 0.5]
                                (CONS X Y])
```

(NORMLOOP

```
  [LAMBDA (LIST)                                         (* FS "10-Feb-86 16:56")

          (* * make a true loop out of list, then snip at a local maxima or minima.
          This is defined as normal form, where a loop begins at a local max/min)

    (PROG (LOOP HEAD LAST Y1 Y2 PREVSGN ANTISGN)
          (SETQ LOOP (COPY LIST))
```

```
          (SETQ LAST (LAST LOOP))
          (RPLACD LAST LOOP)

          (* * run until nonhorizontal section, loop points to lead cons cell)

          (SETQ Y1 (HEADPTY LOOP))
          (SETQ Y2 (NEXTPTY LOOP))
          (while (AND (NEQ LOOP LAST)
                      (EQ Y1 Y2))
             do (SETQ LOOP (CDR LOOP))
                (SETQ Y1 Y2)
                (SETQ Y2 (NEXTPTY LOOP)))
          (SETQ PREVSGN (\SGN (IDIFFERENCE Y2 Y1)))

          (* * handle degenerate flat outlines)

          (if (EQ PREVSGN 0)
              then (SETQ HEAD (CDR LOOP))
                   (RPLACD LOOP NIL)
                   (RETURN HEAD))

          (* * run until strictly opposite section)

          (SETQ Y1 Y2)
          (SETQ Y2 (CDADDR LOOP))
          (SETQ ANTISGN (\SGN (IDIFFERENCE Y1 Y2)))
          [while (NEQ ANTISGN PREVSGN) do (SETQ LOOP (CDR LOOP))
                                          (SETQ Y1 Y2)
                                          (SETQ Y2 (CDADDR LOOP))
                                          (SETQ ANTISGN (\SGN (IDIFFERENCE Y1 Y2]
          (SETQ HEAD (CDR LOOP))
          (RPLACD LOOP NIL)
          (RETURN HEAD])
```

(SLITLOOP

```
  [LAMBDA (KLST)                                                    (* FS "10-Feb-86 16:57")

          (* * Makes a copy of a normalized knot list representing a loop, and slits it into monotonic sections on y)

    (PROG (SECTLIST CURRLIST CURRSGN PREVSGN PREV CURR LAST)

          (* * add first pt to tail to represent all edges)

          (SETQ KLST (COPY KLST))
          (SETQ LAST (COPY (HEADPT KLST)))
          (SETQ KLST (NCONC KLST (LIST LAST)))

          (* * slice into monotonic knot lists)

          (SETQ PREV (HEADPT KLST))
          (SETQ KLST (CDR KLST))
          (SETQ CURR (HEADPT KLST))
          [SETQ PREVSGN (\SGN (IDIFFERENCE (XYPT.Y CURR)
                                           (XYPT.Y PREV]
          (SETQ CURRLIST (LIST PREV))
          (SETQ SECTLIST (LIST CURRLIST))
          (for CURR in KLST do [SETQ CURRSGN (\SGN (IDIFFERENCE (XYPT.Y CURR)
                                                                (XYPT.Y PREV]
                               (if (EQ CURRSGN 0)
                                   then (SETQ CURRSGN PREVSGN))
                               (if (EQ CURRSGN PREVSGN)
                                   then (ATTACH CURR CURRLIST)
                                 else (SETQ CURRLIST (LIST CURR (COPY PREV)))
                                      (SETQ SECTLIST (CONS CURRLIST SECTLIST))
                                      (SETQ PREVSGN CURRSGN))
                               (SETQ PREV CURR))

          (* * currently, each monotonic section is reversed, no effect on algorithms?)

          (RETURN (REVERSE SECTLIST])
```

(PREPLOOP

```
  [LAMBDA (PathOrPathList)                                          (* FS "29-Aug-85 17:47")

          (* * Normalizes and slits a single path or list of paths, returns list of monotones ascending or descending * *)

    (LET (NLIST ILIST LLIST TLIST)
         (if (NUMBERP (CAAR PathOrPathList))
             then (SETQ PathOrPathList (LIST PathOrPathList)))
         (for I in PathOrPathList do (SETQ NLIST (NORMLOOP (IRNDLIST I)))
                                     (SETQ TLIST (SLITLOOP NLIST))
                                     (SETQ ILIST (NCONC TLIST ILIST)))
         ILIST])
```

(YMAPLIST

```
  [LAMBDA (PATH1 PATH2)                                    (* FS "10-Feb-86 12:30")

          (* * project y values from path 1 onto path2 resulting in pts which are on path 2, assumes paths are sorted in y ascending)

    (PROG (YMAP X Y CURRPT NEXTPT X0 Y0 DX DY)
          (SETQ Y (HEADPTY PATH2))
          (while (AND PATH1 (IGEQ Y (HEADPTY PATH1))) do (SETQ PATH1 (CDR PATH1)))
          [for I in PATH1 do (SETQ Y (XYPT.Y I))

          (* * advance path2 until first pt is below current)

                          (while (AND (CDR PATH2)
                                      (IGEQ Y (NEXTPTY PATH2)))
                             do (SETQ PATH2 (CDR PATH2)))
                          (if (CDR PATH2)
                              then (SETQ CURRPT (HEADPT PATH2))
                                   (SETQ NEXTPT (NEXTPT PATH2))
                                   (SETQ X0 (XYPT.X CURRPT))
                                   (SETQ Y0 (XYPT.Y CURRPT))
                                   (SETQ DX (IDIFFERENCE (XYPT.X NEXTPT)
                                                  X0))
                                   (SETQ DY (IDIFFERENCE (XYPT.Y NEXTPT)
                                                  Y0))
                                   (SETQ X (MIDDX X0 Y0 DX DY Y))
                                   (SETQ YMAP (CONS (CONS X Y)
                                                    YMAP]
          (RETURN (REVERSE YMAP])
```

(IMAPLIST

```
  [LAMBDA (PATH1 PATH2)                                    (* FS "10-Feb-86 12:19")

          (* * project y values from path 1 onto path2 resulting in pts which are on path 2, assumes paths are sorted in y ascending)

    (PROG (ADVANCE IMAP PT CURR1 NEXT1 CURR2 NEXT2 X0 Y0 DX DY U0 V0 DU DV X1 Y1 U1 V1)

          (* * should advance both tapes first * *)

          (SETQ CURR1 (HEADPT PATH1))
          (SETQ CURR2 (HEADPT PATH2))
          (SETQ NEXT1 (NEXTPT PATH1))
          (SETQ NEXT2 (NEXTPT PATH2))
          (SETQ Y1 (XYPT.Y NEXT1))
          (SETQ V1 (XYPT.Y NEXT2))

          (* * force path2 update *)

          (if (ILEQ V1 Y1)
              then (SETQ ADVANCE 2)                        (* will init if fix u1 v1)
                   (SETQ U1 (XYPT.X CURR2))
                   (SETQ V1 (XYPT.Y CURR2))
                   (SETQ X0 (XYPT.X CURR1))                (* need to init path1)
                   (SETQ Y0 (XYPT.Y CURR1))
                   (SETQ X1 (XYPT.X NEXT1))
                   (SETQ DX (IDIFFERENCE X1 X0))
                   (SETQ DY (IDIFFERENCE Y1 Y0))
            else (SETQ ADVANCE 1)                          (* will init if fix x1 y1)
                 (SETQ X1 (XYPT.X CURR1))
                 (SETQ Y1 (XYPT.Y CURR1))
                 (SETQ U0 (XYPT.X CURR2))                  (* need to init path2)
                 (SETQ V0 (XYPT.Y CURR2))
                 (SETQ U1 (XYPT.X NEXT2))
                 (SETQ DU (IDIFFERENCE U1 U0))
                 (SETQ DV (IDIFFERENCE V1 V0)))

          (* *)

          (while (AND (CDR PATH1)
                      (CDR PATH2))
             do

          (* * find intersection * *)

                 (if (EQ ADVANCE 1)
                     then (SETQ X0 X1)
                          (SETQ Y0 Y1)
                          (SETQ NEXT1 (NEXTPT PATH1))
                          (SETQ X1 (XYPT.X NEXT1))
                          (SETQ Y1 (XYPT.Y NEXT1))
                          (SETQ DX (IDIFFERENCE X1 X0))
                          (SETQ DY (IDIFFERENCE Y1 Y0))
                   else (SETQ U0 U1)
                        (SETQ V0 V1)
                        (SETQ NEXT2 (NEXTPT PATH2))
                        (SETQ U1 (XYPT.X NEXT2))
                        (SETQ V1 (XYPT.Y NEXT2))
```

```
                    (SETQ DU (IDIFFERENCE U1 U0))
                    (SETQ DV (IDIFFERENCE V1 V0)))

          (* * find intersection * *)

                (SETQ PT (XYSECTLSEG X0 Y0 DX DY U0 V0 DU DV))
                (if (NEQ PT NIL)
                    then (SETQ IMAP (CONS PT IMAP)))

          (* * advance appropriate path)

                (if (ILEQ V1 Y1)
                    then (SETQ PATH2 (CDR PATH2))
                         (SETQ ADVANCE 2)
                  else (SETQ PATH1 (CDR PATH1))
                         (SETQ ADVANCE 1)))
          (RETURN (REVERSE IMAP])
```

(UNIQLIST

```
  [LAMBDA (LIST)                                                    (* FS "10-Feb-86 12:33")
```

(* * removes duplicate items from a listy dups defined if same scan line)

```
    (LET (Y V NEWLIST)
         (SETQ Y (HEADPTY LIST))
         (SETQ NEWLIST (CONS (HEADPT LIST)
                             NIL))
         [for PT in (CDR LIST) do (SETQ V (XYPT.Y PT))
                                  (if (NEQ Y V)
                                      then (SETQ Y V)
                                           (SETQ NEWLIST (CONS PT NEWLIST]
         (REVERSE NEWLIST])
```

(MERGLIST

```
  [LAMBDA (PATH1 PATH2)                                             (* FS "10-Feb-86 12:48")
```

(* * Merge two nondescending knot lists, NOTE%: cannot use XYPT.LESSP since we don't want to lose order of x values, also note we are projecting path1 onto path2, so merge is not commutative, order of pts in path2 is preserved, order in path1 is not * *)

```
    (PROG (IMAP CURR1 CURR2 Y1 Y2)
```

(* * should advance both tapes first * *)

```
          (if (EQ PATH1 NIL)
              then (RETURN PATH2))
          (if (EQ PATH2 NIL)
              then (RETURN PATH1))
```

(* *)

```
          (SETQ CURR1 (HEADPT PATH1))
          (SETQ CURR2 (HEADPT PATH2))
          (SETQ Y1 (XYPT.Y CURR1))
          (SETQ Y2 (XYPT.Y CURR2))
          [while (OR PATH1 PATH2) do (if (ILEQ Y2 Y1)
                                         then (SETQ IMAP (CONS CURR2 IMAP))
                                                                    (* insert pt2)
```

(* * if eq, place all such path1 pts)

```
                                              (while (EQ Y1 Y2) do (SETQ IMAP (CONS CURR1 IMAP))
                                                                   (SETQ PATH1 (CDR PATH1))
                                                                   (if PATH1
                                                                       then (SETQ CURR1 (HEADPT PATH1))
                                                                            (SETQ Y1 (XYPT.Y CURR1))
                                                                     else (SETQ Y1 MAX.INTEGER)))
```

(* * update path2)

```
                                              (SETQ PATH2 (CDR PATH2))
                                              (if PATH2
                                                  then (SETQ CURR2 (HEADPT PATH2))
                                                       (SETQ Y2 (XYPT.Y CURR2))
                                                else (SETQ Y2 MAX.INTEGER))
                                         else (SETQ IMAP (CONS CURR1 IMAP))
                                              (SETQ PATH1 (CDR PATH1))
                                              (if PATH1
                                                  then (SETQ CURR1 (HEADPT PATH1))
                                                       (SETQ Y1 (XYPT.Y CURR1))
                                                else (SETQ Y1 MAX.INTEGER]
```

(* *)

```
          (RETURN (REVERSE IMAP])
```

(MMLTLIST

```
[LAMBDA (KLIST M11 M12 M13 M21 M22 M23)                    (* FS "10-Feb-86 12:51")

        (* * matrix multiply vector of points)

   (LET (NLIST X Y U V)
        (SETQ NLIST (for PT in KLIST collect (SETQ X (XYPT.X PT))
                                             (SETQ Y (XYPT.Y PT))
                                             (SETQ U (PLUS (TIMES X M11)
                                                           (TIMES Y M12)
                                                           M13))
                                             (SETQ V (PLUS (TIMES X M21)
                                                           (TIMES Y M22)
                                                           M23))
                                             (CONS U V])
```

(IMLTLIST

```
[LAMBDA (KLIST M11 M12 M13 M21 M22 M23)                    (* FS "10-Feb-86 12:53")

        (* * matrix multiply vector of points, make integers)

   (LET (NLIST X Y U V)
        (SETQ NLIST (for PT in KLIST collect (SETQ X (CAR PT))
                                             (SETQ Y (CDR PT))
                                             (SETQ U (PLUS (TIMES X M11)
                                                           (TIMES Y M12)
                                                           M13))
                                             (SETQ V (PLUS (TIMES X M21)
                                                           (TIMES Y M22)
                                                           M23))
                                             (CONS (FIX U)
                                                   (FIX V])
```

(XYPT.LESSP

```
[LAMBDA (PT1 PT2)                                          (* FS "10-Feb-86 12:57")

        (* * comment)

   (LET (V1 V2)
        (SETQ V1 (XYPT.Y PT1))
        (SETQ V2 (XYPT.Y PT2))
        (if (NEQ V1 V2)
            then (ILEQ V1 V2)
          else (ILEQ (XYPT.X PT1)
                     (XYPT.X PT2])
```

(PATH.LESSP

```
[LAMBDA (PATH1 PATH2)                                      (* FS "10-Feb-86 12:21")

        (* * y coordinate dominates, otherwise mean x value of edge)

   (LET (PT1 PT2 Y1 Y2 X1 X2 U1 U2)
        (if (EQ NIL PATH1)
            then T
          elseif (EQ NIL PATH2)
            then NIL
          else (SETQ PT1 (CAR PATH1))
               (SETQ PT2 (CAR PATH2))
               (SETQ Y1 (XYPT.Y PT1))
               (SETQ Y2 (XYPT.Y PT2))
               (if (NEQ Y1 Y2)
                   then (ILEQ Y1 Y2)
                 else (SETQ X1 (XYPT.X PT1))
                      (SETQ X2 (XYPT.X PT2))
                      (if (EQ X1 X2)
                          then (PATH.LESSP (CDR PATH1)
                                           (CDR PATH2))
                        elseif (AND (CDR PATH1)
                                    (CDR PATH2))
                          then (SETQ PT1 (CADR PATH1))
                               (SETQ PT2 (CADR PATH2))
                               (SETQ U1 (XYPT.X PT1))
                               (SETQ U2 (XYPT.X PT2))
                               (ILEQ (IPLUS X1 U1)
                                     (IPLUS X2 U2))
                          else (ILEQ X1 X2])
```

(CONVEXP

```
[LAMBDA (PATH)                                             (* FS "10-Feb-86 16:58")
```

(* * tests whether polygon represented by knot list is convex, by checking whether next vertex is on left/right of origin current and tangental vectors)

```
    (PROG (RESULT NEGSGN TSGN CSGN OSGN PT X Y LASTX LASTY ORIGX0 ORIGY0 ORIGDX ORIGDY CURRX0 CURRY0 CURRDX
                  CURRDY TANGX0 TANGY0 TANGDX TANGDY)

          (* * degenerates assumed convex * *)

          (if (ILEQ (LENGTH PATH)
                    3)
              then (RETURN T))
          (SETQ PT (CAR PATH))
          (SETQ ORIGX0 (XYPT.X PT))
          (SETQ ORIGY0 (XYPT.Y PT))
          (SETQ PT (CADR PATH))
          (SETQ TANGX0 (XYPT.X PT))
          (SETQ TANGY0 (XYPT.Y PT))
          (SETQ ORIGDX (IDIFFERENCE TANGX0 ORIGX0))
          (SETQ ORIGDY (IDIFFERENCE TANGY0 ORIGY0))
          (SETQ PT (CADDR PATH))
          (SETQ LASTX (XYPT.X PT))
          (SETQ LASTY (XYPT.Y PT))
          (SETQ TANGDX (IDIFFERENCE LASTX TANGX0))
          (SETQ TANGDY (IDIFFERENCE LASTY TANGY0))
          (SETQ CURRX0 ORIGX0)
          (SETQ CURRY0 ORIGY0)
          (SETQ CURRDX (IDIFFERENCE LASTX ORIGX0))
          (SETQ CURRDY (IDIFFERENCE LASTY ORIGY0))
          (SETQ NEGSGN (IMINUS (XPROD ORIGX0 ORIGY0 ORIGDX ORIGDY LASTX LASTY)))
          (SETQ NEGSGN (\SGN NEGSGN))                          (* hope first three pts not collinear, else this wont work)
          (SETQ PATH (CDDDR PATH))
          (SETQ RESULT T)                                      (* 4th pt and beyond *)
          (while PATH do (SETQ PT (CAR PATH))
                         (SETQ X (XYPT.X PT))
                         (SETQ Y (XYPT.Y PT))
                         (SETQ TSGN (XPROD TANGX0 TANGY0 TANGDX TANGDY X Y))
                         (SETQ CSGN (XPROD CURRX0 CURRY0 CURRDX CURRDY X Y))
                         (SETQ OSGN (XPROD ORIGX0 ORIGY0 ORIGDX ORIGDY X Y))
                         (SETQ TSGN (\SGN TSGN))
                         (SETQ CSGN (\SGN CSGN))
                         (SETQ OSGN (\SGN OSGN))
                         (if (OR (EQ TSGN NEGSGN)
                                 (EQ CSGN NEGSGN)
                                 (EQ OSGN NEGSGN))
                             then (SETQ RESULT NIL)
                                  (RETURN)                     (* Exit loop not function *))
                         (SETQ PATH (CDR PATH))
                         (SETQ TANGX0 LASTX)
                         (SETQ TANGY0 LASTY)
                         (SETQ TANGDX (IDIFFERENCE X LASTX))
                         (SETQ TANGDY (IDIFFERENCE Y LASTY))
                         (SETQ CURRDX (IDIFFERENCE X CURRX0))
                         (SETQ CURRDY (IDIFFERENCE Y CURRY0))
                         (SETQ LASTX X)
                         (SETQ LASTY Y))
          (RETURN RESULT])

)


;;; Line object and methods

(DECLARE%: EVAL@COMPILE

(RECORD XXLINE (ORIG DIFF WIND))
)

(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: EVAL@COMPILE

(PUTPROPS GETLINEDIFF DMACRO [(SELF DX DY)
                              ;; External method, get dx, dy from line
                              (WITH XXLINE SELF (SETQ DX (ffetch (XYPT X) of DIFF))
                                    (SETQ DY (ffetch (XYPT Y) OF DIFF])

(PUTPROPS GETLINEORIG DMACRO [(SELF OX OY)
                              ;; External method, get dx, dy from line
                              (WITH XXLINE SELF (SETQ OX (ffetch (XYPT X) of ORIG))
                                    (SETQ OY (ffetch (XYPT Y) OF ORIG])

(PUTPROPS \GETLINEDIFF DMACRO [(SELF DX DY)                    (* edited " 1-Jan-00 00:00")
                              ;; Degenerate private method, get dx, dy from line
                              (WITH XXLINE SELF (SETQ DX (FFETCH (XYPT X) OF DIFF))
                                    (SETQ DY (FFETCH (XYPT T) OF DIFF])
```

```
(PUTPROPS \GETLINEORIGY DMACRO [(SELF YPTR)
                                (SETQ YPTR (FFETCH (XYPT Y) of (FFETCH (XXLINE ORIG) of SELF])

(PUTPROPS \GETLINEDIFFY DMACRO [(SELF YPTR)
                                (SETQ YPTR (FFETCH (XYPT Y) of (FFETCH (XXLINE DIFF) of SELF])
)
)

(DEFINEQ

(MAKELINE
  [LAMBDA (X0 Y0 DX DY)                                          ; Edited 13-Jun-2021 14:38 by rmk:

          (* * To simplify bresenham arithmetic, switch all lines to have positive dy.
          True direction is held in WIND.)

    (LET (DIR)
         (if (ILESSP DY 0)
             then (SETQ X0 (IPLUS X0 DX))
                  (SETQ Y0 (IPLUS Y0 DY))
                  (SETQ DX (IMINUS DX))
                  (SETQ DY (IMINUS DY))
                  (SETQ DIR -1)
           else (SETQ DIR 1))
         (create XXLINE
                 ORIG _ (MAKEXYPT X0 Y0)
                 DIFF _ (MAKEXYPT DX DY)
                 WIND _ DIR])

(MSECT
  [LAMBDA (X0 Y0 DX DUDY U0 V0 DU DXDV)                          (* FS "15-Aug-85 23:38")

          (* * Major axis intersection of line by edge, assuming x is shallow)

          (* * "(DUDY*X0 - DXDV*U0 + DX*DU*(V0-Y0)) / (DUDY - DXDV)" * *)

    (IQUOTIENT (IPLUS (ITIMES DUDY X0)
                      (IMINUS (ITIMES DXDV U0))
                      (ITIMES DX DU (IDIFFERENCE V0 Y0)))
          (IDIFFERENCE DUDY DXDV])

(XSECT
  [LAMBDA (X0 Y0 DX DY U0 V0 DU DV)                              (* FS "15-Aug-85 23:51")

          (* * X intersection of two bresenham line and edge, on line)

    (LET (X Y DUDY DXDV)
         (SETQ DUDY (ITIMES DU DY))
         (SETQ DXDV (ITIMES DX DV))
         (if (IGEQ (IABS DX)
                   (IABS DY))
             then (SETQ X (MSECT X0 Y0 DX DUDY U0 V0 DU DXDV))
           else (SETQ Y (MSECT Y0 X0 DY DXDV V0 U0 DV DUDY))
                (SETQ X (LINEY Y0 X0 DY DX Y)))
         X])

(YSECT
  [LAMBDA (X0 Y0 DX DY U0 V0 DU DV)                              (* FS "15-Aug-85 23:50")

          (* * Y intersection of two bresenham line and edge, on line)

    (LET (X Y DUDY DXDV)
         (SETQ DUDY (ITIMES DU DY))
         (SETQ DXDV (ITIMES DX DV))
         (if (IGEQ (IABS DX)
                   (IABS DY))
             then (SETQ X (MSECT X0 Y0 DX DUDY U0 V0 DU DXDV))
                  (SETQ Y (LINEY X0 Y0 DX DY X))
           else (SETQ Y (MSECT Y0 X0 DY DXDV V0 U0 DV DUDY)))
         Y])

(XYSECT
  [LAMBDA (X0 Y0 DX DY U0 V0 DU DV)                              (* FS "16-Aug-85 00:42")

          (* * Intersection of two bresenham line and edge, on line)

    (LET (X Y DUDY DXDV)
         (SETQ DUDY (ITIMES DU DY))
         (SETQ DXDV (ITIMES DX DV))
         (if (NEQ DUDY DXDV)
             then (if (IGEQ (IABS DX)
                            (IABS DY))
                      then (SETQ X (MSECT X0 Y0 DX DUDY U0 V0 DU DXDV))
```

```
                        (SETQ Y (LINEY X0 Y0 DX DY X))
                    else (SETQ Y (MSECT Y0 X0 DY DXDV V0 U0 DV DUDY))
                         (SETQ X (LINEY Y0 X0 DY DX Y)))
                (CONS X Y)
          else NIL])
```

(KNOTLINE

```
  [LAMBDA (KNOTS)                                                    (* FS "10-Feb-86 14:06")
```

(* * turns a single knot list, which represents an open path, into a list of line objects, assume integer values)

```
    (PROG (I ALIST ALINE CURRX CURRY PREVX PREVY)
          (SETQ I (CAR KNOTS))
          (SETQ PREVX (XYPT.X I))
          (SETQ PREVY (XYPT.Y I))
          (SETQ KNOTS (CDR KNOTS))
          (for I in KNOTS do (SETQ CURRX (XYPT.X I))
                             (SETQ CURRY (XYPT.Y I))
                             (SETQ ALINE (MAKELINE PREVX PREVY (DIFFERENCE CURRX PREVX)
                                                   (DIFFERENCE CURRY PREVY)))
                             (SETQ ALIST (CONS ALINE ALIST))
                             (SETQ PREVX CURRX)
                             (SETQ PREVY CURRY))
          (RETURN (REVERSE ALIST])
```

(KNOTLOOP

```
  [LAMBDA (KNOTS)                                                    (* FS "10-Feb-86 14:06")
```

(* * turns a single knot list, which represents a closed path, into a list of line objects)

```
    (PROG (I ALIST ALINE CURRX CURRY PREVX PREVY)
          (SETQ I (CAR (LAST KNOTS)))
          (SETQ PREVX (XYPT.X I))
          (SETQ PREVY (XYPT.Y I))
          (for I in KNOTS do (SETQ CURRX (XYPT.X I))
                             (SETQ CURRY (XYPT.Y I))
                             (SETQ ALINE (MAKELINE PREVX PREVY (IDIFFERENCE CURRX PREVX)
                                                   (IDIFFERENCE CURRY PREVY)))
                             (SETQ ALIST (CONS ALINE ALIST))
                             (SETQ PREVX CURRX)
                             (SETQ PREVY CURRY))
          (RETURN (REVERSE ALIST])
```

(LINE.LESSP

```
  [LAMBDA (ARG1 ARG2)                                                (* FS " 6-Aug-85 17:58")
```

(* * comment)

```
    (LET (Y1 Y2 DY1 DY2)
         (\GETLINEORIGY ARG1 Y1)
         (\GETLINEORIGY ARG2 Y2)
         (if (NEQ Y1 Y2)
             then (ILEQ Y1 Y2)
           else (\GETLINEDIFFY ARG1 DY1)
                (\GETLINEDIFFY ARG2 DY2)
                (ILEQ DY1 DY2])

)

(DEFINEQ
```

(LINEY

```
  [LAMBDA (X0 Y0 DX DY X)                                            (* FS "10-Feb-86 17:35")
```

(* * returns y on shallow line given x.)

(* * "Y0 + Round (DY*(X-X0) / DY)" * *)

```
    (if (OR (EQ DX 0)
            (EQ DY 0))
        then (IPLUS Y0 (IQUOTIENT DY 2))
      else (SETQ X (IDIFFERENCE X X0))
           (IPLUS Y0 (\IRND (ITIMES X DY)
                            DX])
```

(MIDDX

```
  [LAMBDA (X0 Y0 DX DY Y)                                            (* FS "10-Feb-86 17:35")
```

(* * returns middle x on shallow line given y.)

```
    (if (OR (EQ DX 0)
            (EQ DY 0))
        then (IPLUS X0 (IQUOTIENT DX 2))
```

```
      else (SETQ Y (IDIFFERENCE Y Y0))
           (if (NEQ (\SGN Y)
                    (\SGN DY))
               then X0
             else (IPLUS X0 (\IRND (ITIMES Y DX)
                                   DY])
```

(INITX

```
  [LAMBDA (X0 Y0 DX DY Y)                                          (* FS "10-Feb-86 16:59")

          (* * returns minimum x on shallow line.)

    (PROG (X)
          (if (OR (EQ DX 0)
                  (EQ DY 0))
              then (RETURN X0))
          (SETQ Y (IDIFFERENCE Y Y0))
          (if (NEQ (\SGN Y)
                   (\SGN DY))
              then (RETURN X0))

          (* * "X0 + 1 + (2*Y*DX - DX) / 2*DY" * *)

          [SETQ X (IPLUS X0 1 (IQUOTIENT (IDIFFERENCE (ITIMES 2 Y DX)
                                                      DX)
                                         (ITIMES 2 DY]
          (RETURN X])
```

(TERMX

```
  [LAMBDA (X0 Y0 DX DY Y)                                          (* FS "10-Feb-86 17:00")

          (* * returns maximum x on shallow line.)

    (PROG (X D)
          (if (OR (EQ DX 0)
                  (EQ DY 0))
              then (RETURN (IPLUS X0 DX)))
          (SETQ Y (IDIFFERENCE Y Y0))
          (if (NEQ (\SGN Y)
                   (\SGN DY))
              then (RETURN X0))

          (* * "X0 + (2*Y*DX + DX) / 2*DY" *)

          [SETQ X (IPLUS X0 (IQUOTIENT (IPLUS DX (ITIMES 2 Y DX))
                                       (ITIMES 2 DY]
          (RETURN X])
```

(SCANX

```
  [LAMBDA (X0 Y0 DX DY Y)                                          (* FS "10-Feb-86 17:00")

          (* * returns scan x values on shallow line.)

    (PROG (YDX2 DY2 XL XR)
          [if (OR (EQ DX 0)
                  (EQ DY 0))
              then (RETURN (CONS X0 (IPLUS X0 DX]
          (SETQ Y (IDIFFERENCE Y Y0))
          [if (NEQ (\SGN Y)
                   (\SGN DY))
              then (RETURN (CONS X0 (IPLUS X0 DX]
          (SETQ YDX2 (ITIMES 2 Y DX))
          (SETQ DY2 (ITIMES 2 DY))
          (SETQ XL (IPLUS X0 1 (IQUOTIENT (IDIFFERENCE YDX2 DX)
                                          DY2)))
          (SETQ XR (IPLUS X0 (IQUOTIENT (IPLUS YDX2 DX)
                                        DY2)))
          (RETURN (CONS XL XR])
```

(XPROD

```
  [LAMBDA (X0 Y0 DX DY X Y)                                        (* edited%: " 9-Aug-85 21:27")

          (* * returns cross product of a vector and the vector from vector origin to a point.
          If the sgn of the x-product is positive, the point lies on the relative left of the
          (vector, assuming right handed system) * *)

    (SETQ X (IDIFFERENCE X X0))
    (SETQ Y (IDIFFERENCE Y Y0))
    (IDIFFERENCE (ITIMES DX Y)
           (ITIMES DY X])

)
```

;;; line segment methods

```
(DEFINEQ
```

(XYSECTLSEG

```
  [LAMBDA (X0 Y0 DX DY U0 V0 DU DV)                                    (* FS "10-Feb-86 14:10")
```

(* * Intersection analogue of lines, but for lsegs, assume dy positive * *)

(* * if line segments intersect, return y else y0)

```
    (PROG (PT Y XT YT UT VT XMAX XMIN UMAX UMIN)
          (SETQ XT (IPLUS X0 DX))
          (SETQ YT (IPLUS Y0 DY))
          (SETQ UT (IPLUS U0 DU))
          (SETQ VT (IPLUS V0 DV))
```

(* * Check X extents first * *)

```
          (if (MINUSP DX)
              then (SETQ XMIN XT)
                   (SETQ XMAX X0)
            else (SETQ XMIN X0)
                 (SETQ XMAX XT))
          (if (MINUSP DU)
              then (SETQ UMIN UT)
                   (SETQ UMAX U0)
            else (SETQ UMIN U0)
                 (SETQ UMAX UT))
          (if (OR (ILEQ XMAX UMIN)
                  (ILEQ UMAX XMIN))
              then (RETURN NIL))
```

(* * find y intersection)

```
          (SETQ PT (XYSECT X0 Y0 DX DY U0 V0 DU DV))
          (if (EQ PT NIL)
              then (RETURN NIL))
```

(* * Make sure point lies in both line y extents)

```
          (SETQ Y (XYPT.Y PT))
          (if (OR (ILEQ Y Y0)
                  (ILEQ YT Y)
                  (ILEQ Y V0)
                  (ILEQ VT Y))
              then (RETURN NIL))
          (RETURN PT])

)
```

;;; Bresenham line object and methods

```
(DECLARE%: EVAL@COMPILE

(DATATYPE BRES (MAJOR X Y P IX IY IPX IPY))
)

(/DECLAREDATATYPE 'BRES '(POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER)
```

;; ---field descriptor list elided by lister---

```
       '16)

(DEFINEQ
```

(MAKEBRES

```
  [LAMBDA (X0 Y0 DX DY)                                                ; Edited 24-Aug-87 22:29 by FS
```

;; Assume canonical form, dy is positive

```
    (LET (SELF PX PY PXY)
         (SETQ SELF (create BRES))
         [with BRES SELF                                               (*)
               (SETQ X X0)
               (SETQ Y Y0)
               (SETQ IX (\SGN DX))
               (SETQ IY (\SGN DY))
               (SETQ PX (IABS DX))
               (SETQ PY (IABS DY))
               (SETQ PXY (IDIFFERENCE PX PY))
               (if (EQ PY 0)
                   then (SETQ MAJOR 'NIL)
                        (SETQ X (IPLUS X0 DX))
                        (SETQ IPX 1)
                        (SETQ IPY 1)
                        (SETQ P -1)
```

```
                elseif (IGEQ PX PY)
                   then (SETQ MAJOR 'X)
                        (SETQ IPX (IMAX 1 (ITIMES 2 PY)))
                        (SETQ P (IDIFFERENCE IPX PX))
                        (SETQ IPY (ITIMES 2 (IDIFFERENCE PY PX)))
                else (SETQ MAJOR 'Y)
                     (SETQ IPY (ITIMES 2 PX))
                     (SETQ P (IDIFFERENCE IPY PY))
                     (SETQ IPX (ITIMES 2 (IDIFFERENCE PX PY]

          (* * (replace (BRES X) of SELF with X0) (replace (BRES Y) of SELF with Y0)
          (replace (BRES IX) of SELF with (\SGN DX)) (replace (BRES IY) of SELF with
          (\SGN DY)) (SETQ PX (IABS DX)) (SETQ PY (IABS DY)) (SETQ PXY
          (IDIFFERENCE PX PY)) (if (EQ PY 0) then (replace (BRES MAJOR) of SELF with
          (QUOTE NIL)) (replace (BRES X) of SELF with (IPLUS X0 DX))
          (replace (BRES IPX) of SELF with 1) (replace (BRES IPY) of SELF with 1)
          (replace (BRES P) of SELF with -1) elseif (IGEQ PX PY) then (replace
          (BRES MAJOR) of SELF with (QUOTE X)) (replace (BRES IPX) of SELF with
          (IMAX 1 (ITIMES 2 PY))) (replace (BRES P) of SELF with (IDIFFERENCE
          (fetch (BRES IPX) of SELF) PX)) (replace (BRES IPY) of SELF with
          (ITIMES 2 (IDIFFERENCE PY PX))) else (replace (BRES MAJOR) of SELF with
          (QUOTE Y)) (replace (BRES IPY) of SELF with (ITIMES 2 PX))
          (replace (BRES P) of SELF with (IDIFFERENCE (fetch (BRES IPY) of SELF) PY))
          (replace (BRES IPX) of SELF with (ITIMES 2 (IDIFFERENCE PX PY)))))

        SELF])

)

(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: EVAL@COMPILE

(PUTPROPS BRESSTEP DMACRO ((SELF SCANY INITX TERMX)

          (* * Assume that this Y is exactly incremented by one from the last call, so DDA can be used.
          Then passed parameter scany is not used.)

          (* * (WITH BRES SELF (*) (SETQ INITX X) (IF (EQ MAJOR (QUOTE X)) THEN
          (while (ILESSP P 0) do (SETQ X (IPLUS X IX)) (SETQ P (IPLUS P IPX)))
          (IF (ILEQ INITX X) THEN (SETQ TERMX X) ELSE (SETQ TERMX INITX)
          (SETQ INITX X)) (SETQ X (IPLUS X IX)) (SETQ P (IPLUS P IPY)) ELSE
          (SETQ TERMX X) (if (ILESSP P 0) then (SETQ P (IPLUS P IPY)) else
          (SETQ P (IPLUS P IPX)) (SETQ X (IPLUS X IX))))))

                                  (LET (X0 DX D DDX DDY)

          (* * (WITH BRES SELF (*) (SETQ X0 X) (SETQ DX IX) (SETQ D P)
          (SETQ DDX IPX) (SETQ DDY IPY)))

                                       (SETQ D (FFETCH (BRES P) OF SELF))
                                       (SETQ X0 (FFETCH (BRES X) OF SELF))
                                       (SETQ DX (FFETCH (BRES IX) OF SELF))
                                       (SETQ DDX (FFETCH (BRES IPX) OF SELF))
                                       (SETQ DDY (FFETCH (BRES IPY) OF SELF))

          (* * Above faster than WITH form * *)

                                       (SETQ INITX X0)
                                       [IF (EQ (FFETCH (BRES MAJOR) OF SELF)
                                               'X)
                                           THEN (while (ILESSP D 0) do (SETQ X0 (IPLUS X0 DX))
                                                                       (SETQ D (IPLUS D DDX)))
                                                (IF (ILEQ INITX X0)
                                                    THEN (SETQ TERMX X0)
                                                  ELSE (SETQ TERMX INITX)
                                                       (SETQ INITX X0))
                                                (SETQ X0 (IPLUS X0 DX))
                                                (SETQ D (IPLUS D DDY))
                                         ELSE (SETQ TERMX X0)
                                              (if (ILESSP D 0)
                                                  then (SETQ D (IPLUS D DDY))
                                                else (SETQ D (IPLUS D DDX))
                                                     (SETQ X0 (IPLUS X0 DX]

          (* * (WITH BRES SELF (*) (SETQ X X0) (SETQ P D)))

                                       (FREPLACE (BRES X) OF SELF with X0)
                                       (FREPLACE (BRES P) OF SELF with D))))
)
)


;;; Debugging control panel

(DECLARE%: EVAL@COMPILE DONTCOPY
```

```
(RPAQQ \GEOM.PANEL
       ((* * Debugging control panel * *)
        (PROG NIL (SETQQ KLIST ((10 . 10)
                                (20 . 20)
                                (30 . 20)
                                (40 . 30)
                                (50 . 10)
                                (30 . 0)))
              (SETQ NLIST (NORMLOOP KLIST))
              (SETQ SLIST (SLITLOOP NLIST))
              (MYPGON MYWIN KLIST)
              (MYNUMB MYWIN KLIST))))
)


;;; Trapezoidal decomposition

(DEFINEQ

(TRAPLOOP
  [LAMBDA (PATH)                                                    (* FS "10-Feb-86 14:12")

          (* * decomposes single path or pathlist into trapezoids, odd winding rule)

    (PROG (KNOTS PLIST CLIST LEN IPATH JPATH KPATH XPATH TEMP PT.LESSP)
          (SETQ PT.LESSP 'XYPT.LESSP)

          (* * Handle path or list)

          (SETQ PLIST (PREPLOOP PATH))
          (SETQ PATH (COPYALL PATH))
          [if (NUMBERP (CAAR PATH))
              then (SETQ KNOTS PATH)
            else (for LOOP in PATH do (SETQ KNOTS (NCONC KNOTS LOOP]

          (* * Force monotonic lists to ascend)

          (SETQ PLIST (for I in PLIST collect (if [IGREATERP (HEADPTY I)
                                                             (XYPT.Y (CAR (LAST I]
                                                  then (REVERSE I)
                                                else I)))
          (if TRAP.DEBUG
              then (DV PLIST))
          (SETQ LEN (LENGTH PLIST))

          (* * find all intersections, seed into critical lists * *)

          (* * (SETQ CLIST NIL) (FOR I IN CLIST DO (SETQ CLIST (CONS NIL CLIST))))

          (SETQ CLIST (COPYALL PLIST))
          [for I from 1 to (SUB1 LEN)
             do (SETQ IPATH (CAR (NTH PLIST I)))
                (for J from (ADD1 I) to LEN do (SETQ JPATH (CAR (NTH PLIST J)))
                                               (SETQ XPATH (IMAPLIST IPATH JPATH))
                                               (SETQ CLIST (for K from 1 to LEN
                                                              collect (SETQ KPATH (CAR (NTH CLIST K)))
                                                                      (if (OR (EQ K I)
                                                                              (EQ K J))
                                                                          then (MERGLIST (COPY XPATH)
                                                                                         KPATH)
                                                                        else (SETQ TEMP (YMAPLIST XPATH KPATH))
                                                                             (MERGLIST TEMP KPATH]
          (if TRAP.DEBUG
              then (DV CLIST))

          (* * cull out duplicates and combine with joints * *)

          (SORT KNOTS PT.LESSP)
          (SETQ TEMP (for I in PLIST collect (YMAPLIST KNOTS I)))
          [SETQ CLIST (for I from 1 to LEN collect (MERGLIST (CAR (NTH TEMP I))
                                                             (CAR (NTH CLIST I]

          (* * pull out trapezoids from critical pt list * *)

          (RETURN (TRAPMAKE CLIST])

(TRAPMAKE
  [LAMBDA (EDGELIST)                                                (* FS "10-Feb-86 14:13")

          (* * Given decomposed edges, traverse making trapezoids)

    (PROG (TRAPLIST UNFINISHED PTR LIST1 LIST2 PT1 PT2 PT3 PT4 TRAP CURRY ALIST)

          (* * theoretically, each pair of paths must represent a trapezoid for ith and i+1th pts * *)

          (SETQ UNFINISHED T)
```

```
          (while UNFINISHED do (SORT EDGELIST 'PATH.LESSP)
                               (SETQ CURRY (CDAAR EDGELIST))
                               (SETQ PTR EDGELIST)
                               (SETQ ALIST NIL)
                               (while (EQ CURRY (CDAAR PTR))
                                  do (SETQ LIST1 (CAR PTR))
                                     (SETQ LIST2 (CADR PTR))
                                     (SETQ PT1 (HEADPT LIST1))
                                     (SETQ PT2 (NEXTPT LIST1))
                                     (SETQ PT4 (HEADPT LIST2))
                                     (SETQ PT3 (NEXTPT LIST2))

          (* * check for duplicate y vals)

                                     (while (AND (EQ (XYPT.Y PT1)
                                                     (XYPT.Y PT2))
                                                 (CDR LIST1))
                                        do (SETQ PT1 PT2)
                                           (SETQ LIST1 (CDR LIST1))
                                           (SETQ PT2 (NEXTPT LIST1)))

          (* * check for duplicate y vals)

                                     (while (AND (EQ (XYPT.Y PT4)
                                                     (XYPT.Y PT3))
                                                 (CDR LIST2))
                                        do (SETQ PT4 PT3)
                                           (SETQ LIST2 (CDR LIST2))
                                           (SETQ PT3 (NEXTPT LIST2)))

          (* * make trap, advance)

                                     (SETQ TRAP (LIST PT1 PT2 PT3 PT4))
                                     (if [ILESSP (XYPT.Y PT2)
                                                 (XYPT.Y (CAR (LAST LIST1]
                                         then (SETQ ALIST (CONS (CDR LIST1)
                                                                ALIST)))
                                     (if [ILESSP (XYPT.Y PT3)
                                                 (XYPT.Y (CAR (LAST LIST2]
                                         then (SETQ ALIST (CONS (CDR LIST2)
                                                                ALIST)))

          (* * this had better be a trapezoid, put a debugging hook here * *)

                                     (if (OR (NEQ (CDR PT1)
                                                  (CDR PT4))
                                             (NEQ (CDR PT2)
                                                  (CDR PT3)))
                                         then (printout T "NON-TRAPEZOID FOUND!" T)
                                              (DV TRAP))
                                     (SETQ TRAPLIST (CONS TRAP TRAPLIST))
                                     (SETQ PTR (CDDR PTR)))
                               (SETQ EDGELIST (MERGE ALIST PTR 'PATH.LESSP))
                               (if (IGREATERP 2 (LENGTH EDGELIST))
                                   then (SETQ UNFINISHED NIL)))
          (RETURN TRAPLIST])

)

(RPAQQ TRAP.DEBUG NIL)

(DECLARE%: EVAL@COMPILE DONTCOPY

(RPAQQ \TRAP.PANEL
       [(WHILE (EQ MYWIN (WHICHW (GETPOSITION)))
               DO
               (CLEARW MYWIN2)
               (CLEARW MYWIN)
               (NILL (SETQ RLIST (MYRAND MYWIN 7)))
               (MYPGON MYWIN2 RLIST)
               (POLYSHADE.DISPLAY MYWIN2 RLIST GRAYSHADE)
               (NILL (SETQ RZOID (TRAPLOOP RLIST)))
               (PRINTOUT MYWIN "NUMBER OF TRAPEZOIDS: " (LENGTH RZOID)
                      T)
               (FOR I IN RZOID DO (POLYSHADE.DISPLAY MYWIN I (RAND)))
               (MYPGON MYWIN RLIST))
        (NILL (NILL (SETQ RLIST (MYRAND MYWIN 5)))
              (MYPGON MYWIN RLIST)
              (FOR I IN RZOID DO (MYPGON MYWIN I))
              (MYPGON MYWIN (CAR (NTH RZOID 27)))
              (FOR I FROM 23 TO 25 DO (POLYSHADE.DISPLAY MYWIN (CAR (NTH RZOID I))
                                      GRAYSHADE)))
        (NILL (WHILE (EQ MYWIN (WHICHW (GETPOSITION)))
                     DO
                     (CLEARW MYWIN)
                     (SETQ RLIST (MYDRAW MYWIN NIL))
                     (MYPGON MYWIN RLIST)
```

```
                    (PRINTOUT MYWIN "CONVEX: " (CONVEXP RLIST)
                           T])
)
```

FUNCTION INDEX

MACRO INDEX

VARIABLE INDEX

RECORD INDEX