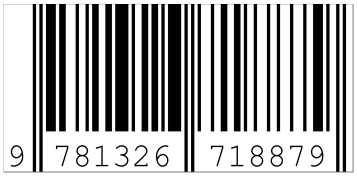
Algoritmi per l'ottimizzazione di estrazioni da nuvole di punti

Giovanni Anzani – 2016
Università degli Studi di Firenze - Dipartimento di Architettura DIDA
Via Della Mattonaia 14 - 50121 Firenze (FI)
Mob.: +39/339/7056663 Skype: giovanni.anzani.unifi
giovanni.anzani@unifi.it



ABSTRACT

Questa pubblicazione tratta la realizzazione di una serie di procedure in AutoLISP finalizzate all'ottimizzazione di estrazioni di punti da delle nuvole di punti. Nella realizzazione dei comandi associati, viene fatto largo uso delle possibilità di interoperabilità tra AutoCAD ed AutoLISP sfruttando le funzionalità avanzate introdotte con VisualLISP.

Parole chiave

Algoritmi, Disegno, AutoLisp, AutoCAD.

INTRODUZIONE

La trattazione che segue descrive dapprima le procedure ed i comandi realizzati organizzandoli in macro aree ed è seguita da un'appendice contenente il codice sorgente completo che rende disponibili le funzionalità descritte in ambiente AutoCAD. Si fa presente che una parte delle procedure riportate in appendice, non trova in questo saggio una descrizione nella parte dedicata alle macro aree, in quanto tali procedure sono state precedentemente descritte in altra pubblicazione dello stesso autore[1]; si è ritenuto opportuno inserirle per permettere la funzionalità del codice sorgente in appendice.

PROCEDURE E COMANDI REALIZZATI

La realizzazione in AutoLISP dei comandi descritti sopra, ha richiesto la realizzazione di una ventina di funzioni; per una trattazione più agevole ed ordinata, l'insieme degli algoritmi realizzati è stato suddiviso in macro aree contenenti la descrizione sintetica degli algoritmi afferenti a ciascuna macro area.

1) Funzioni VLAX. La funzione che segue permette di inserire valori in formato lista nelle proprietà di alcuni oggetti presenti nel database di autocad:

(put_vvertex_2D ent index Pu)) → inserisce il punto specificato nella variabile Pu in forma (x y) o (x y z), in una proprietà compatibile, come ad esempio potrebbe essere la posizione del centro di una circonferenza; in questo modo si cambierebbe, nel disegno, la posizione della circonferenza. Qualora la proprietà compatibile contenesse una lista di punti, specificando la posizione desiderata nella variabile index è possibile inserire il punto fornito nella variabile Pu in una specifica posizione.

2) Funzioni e comandi relativi all'algoritmo di Douglas Pecker. L'algoritmo di Douglas-Peucker, permette di rimappare una spezzata in maniera variabile in funzione del suo andamento: nei tratti ad andamento pseudo lineare, la quantità di punti risulterà

[1] Giovanni Anzani. 2015. *Algoritmi per l'elaborazione di estrazioni da nuvole di punti*. Lulu.

ridotta; nei tratti ad andamento più variabile la quantità di punti verrà mantenuta più alta. L'algoritmo riesce a ridurre il numero dei vertici necessari alla rappresentazione della spezzata originale, mantenendo una rappresentazione abbastanza fedele, a livello percettivo, della spezzata originale L'algoritmo non determina nuovi punti, ma filtra i punti esistenti secondo quanto appena descritto, pertanto la spezzata risultante passerà da una selezione dei punti originali. I due comandi realizzati basandosi sull'algoritmo sopra descritto, permetteranno di interrompere la selezione dei punti da utilizzarsi nella nuova entità, o al raggiungimento di una distanza minima limite dei punti estratti dalla spezzata originale o al raggiungimento di una quantità massima limite di punti estratti dalla spezzata originale:

(PT_farest_cur lP1 cur)) → data una lista di punti fornita nella variabile lP1 ed il nome di un'entità grafica di AutoCAD, la procedura restituisce una lista contenente le coordinate del punto più lontano dall'entità grafica indicata e la sua distanza da tale entità.

(lP_set_nil_all lP) → data una lista di punti 2D o 3D nella forma rispettivamente (x y) o (x y z) fornita nella variabile lP, la procedura restituisce una lista nella quale ad ogni punto è stata aggiunta una componente assegnandogli il valore nil. I punti nella lista risulteranno nella forma (x y nil) o (x y z nil).

(lP_Tnil_set_P_T P lP_Tnil) → data un punto 2D o 3D nella forma rispettivamente (x y) o (x y z) fornito nella variabile P e data una lista di punti 2D o 3D nella forma (x y w) o (x y z w) fornita nella variabile lP_Tnil (con w pari a T o nil), la procedura assegnerà al punto P eventualmente trovato nella lista fornita il valore T alla sua componente w restituendo la lista aggiornata.

(lP_Tnil_set_start_P_T lP_Tnil) → data una lista di punti 2D o 3D nella forma (x y w) o (x y z w) fornita nella variabile lP_Tnil (con w pari a T o nil), la procedura assegnerà al primo punto della lista fornita il valore T alla sua componente w restituendo la lista aggiornata.

(lP_Tnil_set_end_P_T (lP_Tnil) → data una lista di punti 2D o 3D nella forma (x y w) o (x y z w) fornita nella variabile lP_Tnil (con w pari a T o nil), la procedura assegnerà all'ultimo punto della lista fornita il valore T alla sua componente w restituendo la lista aggiornata.

(lP_Tnil_sel_T lP_Tnil) → data una lista di punti 2D o 3D nella forma (x y w) o (x y z w) fornita nella variabile lP_Tnil (con w pari a T o nil), la procedura restituirà una sotto lista con i soli punti aventi la componente w pari a T.

(lP_Tnil_sel_nil lP_Tnil) → data una lista di punti 2D o 3D nella forma (x y w) o (x y z w) fornita nella variabile lP_Tnil (con w pari a T o nil), la procedura restituirà una sotto lista con i soli punti aventi la componente w pari a nil.

(dppl_fA) → questa funzione raggruppa alcuni passaggi da utilizzarsi nel primo dei due comandi realizzati in questa macro area.

(dppl_fAc) → questa funzione raggruppa alcuni passaggi da utilizzarsi nel secondo dei due comandi realizzati in questa macro area.

(dppl_fB) → questa funzione raggruppa alcuni passaggi da utilizzarsi nei due comandi realizzati in questa macro area nel caso in cui si scelga di fermare l'algoritmo di calcolo al raggiungimento di un numero di punti stabilito.

(dppl_fC) → questa funzione raggruppa alcuni passaggi da utilizzarsi nei due comandi realizzati in questa macro area nel caso in cui si scelga di fermare l'algoritmo di calcolo al raggiungimento di una distanza minima stabilita.

(c:dppl) → Sfruttando l'interoperabilità tra AutoLISP e AutoCAD tale comando ottimizza l'entità selezionata in base all'impostazione, da parte dell'utente, dei parametri disponibili.

(c:dpplc) → Sfruttando l'interoperabilità tra AutoLISP e AutoCAD tale comando ottimizza l'entità selezionata in base all'impostazione, da parte dell'utente, dei parametri disponibili. A differenza del precedente comando permette di riprendere una precedente ottimizzazione variandone i parametri operativi senza dover riprendere il lavoro dal punto iniziale.

3) Polilinea passante da n punti noti con possibilità di filtraggio dei punti. Tale comando permette di disegnare un entità grafica passante per una serie di punti selezionati dall'utente; grazie ad alcuni accorgimenti ed utilizzando un eventuale filtraggio dei punti da utilizzare consente di ottimizzare il risultato ottenuto. Alla base di tale comando c'è l'idea di ordinare tali punti in maniera tale da determinare una polilinea non intersecantesi trascurando eventuali punti la cui distanza dalla traiettoria di riferimento indicata risulti essere eccessiva:

(c:ent_from_PT) → Sfruttando l'interoperabilità tra AutoLISP e AutoCAD tale comando, dopo aver chiesto di fornire dei punti selezionandoli nell'ambiente AutocAD, permette di indicare una traiettoria spezzata di orientamento per la creazione dell'entità desiderata che è possibile scegliere tra la polilinea 2D la polilinea 3D e la spline; opzionalmente è possibile indicare una distanza limite di tolleranza per escludere i punti che si trovino ad una distanza maggiore da quella indicata, rispetto alla polilinea traiettoria indicata.

4) Funzioni sulle medie semplici o ponderate. Tali funzioni predispongono una serie di algoritmi per il calcolo della media semplice o ponderata che saranno utilizzati nella successiva macro area dedicata all'interpolazione di Lagrange e Bézier dasata sul calcolo di medie mobili ti tipo semplice o ponderato:

(PT_med_lPT_semp lPT) → calcola la media semplice, delle coordinate dei punti forniti nella lista lPT e restituisce le coordinate del punto che rappresenta la media semplice. I punti forniti possono essere sia 2D che 3D espressi rispettivamente nella forma (x y) o (x y z).

(gen_pesi_mm ord_mm tipo_mm) → genera una lista di valori rappresentanti il peso da assegnare ad una quantità di punti specificata nella variabile ord_mm che sia preferibilmente dispari (per ragioni di simmetria dei pesi rispetto alla posizione centrale); tale peso, a seconda di come specificato nella variabile tipo:mm, può essere costante per l'uso nella media mobile semplice, a crescita lineare per l'uso nella media mobile ponderata linearmente, a crescita geometrica per l'uso nella media mobile ponderata geometricamente.

(prodotto_peso_PT peso PT) → moltiplica le coordinate del punto 2D o 3D fornito nella variabile PT nella forma (x y) o (x y z), per il valore fornito nella variabile peso; in questo modo si predispone il punto per l'utilizzo in una media mobile ponderata.

(PT_med_lPT_pond lis_PT tipo_mm) → calcola la media ponderata o semplice, dei punti forniti nella variabile lis_PT e restituisce le coordinate del punto che rappresenta tale media. La media, a seconda di come indicato nella variabile tipo_mm, può essere semplice, ponderata linearmente o ponderata geometricamente. I punti forniti possono essere sia 2D che 3D espressi rispettivamente nella forma (x y) o (x y z).

5) Funzioni e comandi per l'interpolazione di Lagrange e Bézier.
La funzione ed il comando che seguono sono rispettivamente dedicati alla determinazione dei nuovi punti di rimappaggio e al disegno della nuova entità rimappata:

(PT_curve_interpole ent_ori ord_med tipo_med) → questa funzione, dopo aver estratto le coordinate dei vertici dell'entità specificata nella variabile ent_ori, ne calcola e restituisce delle nuove coordinate interpolate utilizzando una media mobile di ordine (3 5 7 9 …) come indicato nella variabile ord_med e di tipo (Semplice Lineare Geometrica) come indicato nella variabile tipo_med.

(c:ent_interpole)) → questo comando rende disponibile in AutoCAD la funzionalità di interpolazione di un entità di tipo polilinea 2D o 3D o di tipo splinea secondo quanto predisposto nella funzione sopra descritta. Nella fase di interoperabilità tra AutoCAD e Autolisp il comando richiede di selezionare l'entità da interpolare, di specificare il tipo di entità che si vuole creare e di specificare l'ordine ed il tipo di media mobile da utilizzare.

BIBLIOGRAFIA

[1] Abelson Harold, Disessa Andrea. 1986. *La geometria della tartaruga: esplorare la matematica con il computer*. Franco Muzzio & c. editore. Padova

[2] Agosto M. 1993. *AutoLISP. Corso base per utenti non programmatori.* Tecniche Nuove, Milano.

[3] Anzani Giovanni. 2015. *Algoritmi per l'elaborazione di estrazioni da nuvole di punti*. Lulu.

[4] Asperl Andreas, Hofer Michael, Kilian Axel, Pottman Helmut, 2007, *Architectural Geometry*, Bentley Institute Press, Exton Pennsylvania USA

[5] Blumenthal Leonard M., Menger Karl. 1970. *Studies in geometry*. W. H. Freeman and company. San Francisco USA

[6] Bousfield Trevor 1999. *AutoCAD AutoLISP.Guida pratica.* Tecniche Nuove, Milano. (edizione originale: Bousfield T. 1998. *A practical Guide to AutoCAD Autolisp.* Addison Wesley Longman Limited, England.)

[7] Gesner Rusty, Smith Joseph. 1991. *Maximizing AutoCAD: inside AutoLISP volume II.* New Riders Publishing. Gresham USA

[8] Gesner Rursty, Smith Joseph, 1990, *AutoLISP. Tecniche di programmazione,* Jackson libri, Milano

[9] Kramer Bill, 1997, *AutoLISP treasure chest: Programming gems, cool routines, and useful utilities*, Miller Freeman Books, San Francisco USA

[10] Grippo luigi, Sciandrone Marco. 2011. *Metodi di ottimizzazione non vincolata*. Springer. Milano

[11] Krawczyk Robert J. 2009. *The Codewriting Workbook. Creating computational Architecture in AutoLISP.* Princeton architectural press. New York

[12] Paoluzzi Alberto, 2003, *Informatica grafica e CAD*, Hoepli, Milano

[13] Piccini Claudio 2007, *LISP Trek: Guida all'uso del linguaggio LISP in ambiente CAD*, Lampi di stampa, Milano

[14] Piccini Claudio 2007, *Opus incerta: Istantanee di un viaggio attorno alla computer grafica*, Lampi di stampa, Milano

[15] Styandiford Kevin, 2001, *AutoLISP to VisuaLISP: Design solutions for AutoCAD*, autodesk press, Canada

[16] Togores Reinaldo N., 2012, *Autocad expert's Visual LISP*, Createspace Independent Pub, USA

[17] Togores Fernandez R. and Otero Gonzales C. 2003. *Programacion en AutoCAD con Visual LISP.* Mc Graw Hill, Madrid.

[18] Vuldy Jean Louis. 1985. *Grafica tridimensionale per il personal computer*. Tecniche nuove. Milano (edizione originale: Vuldy Jean Louis. 1983. *Graphisme 3D sur votre micro-ordinateur*. Eyrolles. Parigi)

APPENDICE

Nel seguente paragrafo è riportato il codice sorgente delle procedure sopra descritte, unitamente al codice sorgente già commentato in un precedentre saggio, ma necessario al corretto funzionamento delle procedure descritte; è possibile usare il seguente codice menzionandone l'autore. L'autore non si assume alcuna responsabilità circa il corretto funzionamento di tali procedure nelle versioni passate, attuale e future di AutoCAD.

```
;-------------------------------------------------------------------------------------------------------------------------------------------
;CARICAMENTO DELLE FUNZIONI VL E VLAX
;-------------------------------------------------------------------------------------------------------------------------------------------
(vl-load-com)
;-------------------------------------------------------------------------------------------------------------------------------------------
;SETTAGGIO DI COSTANTI
;-------------------------------------------------------------------------------------------------------------------------------------------
(setq *0.0* 1e-8)  ;tolleranza sullo 0 da usare in vari casi in particolare in alcune situazioni di equal
(setq *inf* 1e+8)  ;tolleranza sull'infinito da usare in vari casi
;-------------------------------------------------------------------------------------------------------------------------------------------
;INIZIALIZZAZIONE DI VARIABILI
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun BLI-1        () (setq bl_old      (getvar "blipmode"   )) (setvar "blipmode"    1 ))
(defun BLI-0        () (setq bl_old      (getvar "blipmode"   )) (setvar "blipmode"    0 ))
(defun OSM-0        () (setq os_old      (getvar "osmode"     )) (setvar "osmode"      0 ))
(defun CMD-0        () (setq cm_old      (getvar "cmdecho"    )) (setvar "cmdecho"     0 ))
(defun SPL-0        () (setq st_old      (getvar "splinetype" )) (setvar "splinetype"  5 )
                       (setq ss_old      (getvar "splinesegs" )) (setvar "splinesegs"  256 ))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun UCS-0        () (if   (P_ucsname "ucs_old")          (vl-cmdf "._ucs" "_na" "_s" "ucs_old"  "_y")
                                                            (vl-cmdf "._ucs" "_na" "_s" "ucs_old" )))
;-------------------------------------------------------------------------------------------------------------------------------------------
;RIPRISTINO DI VARIABILI
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun BLI-P        ()          (setvar "blipmode"    bl_old  ))
(defun OSM-P        ()          (setvar "osmode"      os_old  ))
(defun CMD-P        ()          (setvar "cmdecho"     cm_old  ))
(defun SPL-P        ()          (setvar "splinetype"  st_old  )
                                (setvar "splinesegs"  ss_old  ))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun UCS-P        () (if (P_ucsname "ucs_old")            (progn    (vl-cmdf "._ucs" "_na" "_r" "ucs_old" )
                                                                      (vl-cmdf "._ucs" "_na" "_d" "ucs_old" ))))
;-------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI DI MODIFICA DELL'UCS
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun UCSW         ()                  (vl-cmdf "._ucs" "_w")                          )
(defun UCSP         ()                  (vl-cmdf "._ucs" "_p")                          )
(defun UCS_3Pu      (Pu1 Pu2 Pu3)       (vl-cmdf "._ucs" "_3p" Pu1 Pu2 Pu3)             )
;-------------------------------------------------------------------------------------------------------------------------------------------
; SETTAGGI DI INIZIALIZZAZIONE E RIPRISTINO RAGGRUPPATI
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun VAR-0        () (CMD-0)   (OSM-0)   (UCS-0)   (SPL-0)   (BLI-0))
(defun VAR-P        () (CMD-P)   (OSM-P)   (UCS-P)   (SPL-P)   (BLI-P)   (princ))
(defun VAR-P+enla   () (CMD-P)   (OSM-P)   (UCS-P)   (SPL-P)   (BLI-P)   (princ) (entlast))
;-------------------------------------------------------------------------------------------------------------------------------------------
;TOOLS VLAX
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun get_vval (ent §prop / vent prop)
        (setq   vent (vlax-ename->vla-object ent)
                prop (read §prop))
        (if     (vlax-property-available-p vent prop)
                (vlax-get-property vent prop)))
;-------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------
 (defun get_vlis (ent §prop / vent prop)
          (setq      vent (vlax-ename->vla-object ent)
                     prop (read §prop))
          (if        (vlax-property-available-p vent prop)
                     (vlax-safearray->list          (vlax-variant-value          (vlax-get-property vent prop)))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun put_vval (ent §prop val / vent prop)
          (setq      vent (vlax-ename->vla-object ent)
                     prop (read §prop))
          (if        (vlax-property-available-p vent prop)
                     (vlax-put-property vent prop val)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun put_vvertex_2D (ent index Pu)
  (if (= (length Pu) 3) (setq Pu (rem_last Pu)))
  (setq vent (vlax-ename->vla-object ent))
  (setq vPu  (vlax-make-safearray vlax-vbDouble '(0 . 1)))
  (vlax-safearray-fill vPu Pu)
  (vla-AddVertex vent index vPu)
  (vla-Update    vent))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;COMANDI DI DISEGNO
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Draw_lPu          (lPu)          (VAR-0)     (foreach Pu lPu  (vl-cmdf "._point" Pu))                                    (VAR-P))
(defun Draw_pline2D      (lPu)          (VAR-0)     (vl-cmdf "._pline")    (mapcar 'vl-cmdf lPu) (vl-cmdf "")          (VAR-P+enla))
(defun Draw_pline3D      (lPu)          (VAR-0)     (vl-cmdf "._3dpoly") (mapcar 'vl-cmdf lPu) (vl-cmdf "")          (VAR-P+enla))
(defun Draw_spline       (lPu)          (VAR-0)     (vl-cmdf "._spline")  (mapcar 'vl-cmdf lPu) (vl-cmdf "" "" "")  (VAR-P+enla))
(defun Draw_xli_2Pu      (Pu1 Pu2)      (VAR-0)     (vl-cmdf "._xline"       Pu1 Pu2        "")                         (VAR-P+enla))
(defun Draw_lin_2Pu      (Pu1 Pu2)      (VAR-0)     (vl-cmdf "._line"        Pu1 Pu2        "")                         (VAR-P+enla))
(defun Draw_Cer_Cr       (Cen rag)      (VAR-0)     (vl-cmdf "._circle"      Cen rag)                                    (VAR-P+enla))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;PREDICATI
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_ucsname (ucsname / v1 v2)
  (vlax-for v1 (vla-get-UserCoordinateSystems (vla-get-Activedocument (vlax-get-acad-object)))
               (setq v2 (cons (vla-get-Name v1) v2)))
  (if (member ucsname v2) T nil))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_Pu_2D    (Pu1)               (and Pu1 (listp Pu1) (= 2 (length Pu1))))
(defun P_Pu_3D    (Pu1)               (and Pu1 (listp Pu1) (= 3 (length Pu1))))
(defun P_Pu       (Pu1)               (or (P_Pu_2D Pu1)(P_Pu_3D Pu1)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------

;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_lPu_2D   (lPu)               (apply 'and (mapcar 'P_Pu_2D lPu)))
(defun P_lPu_3D   (lPu)               (apply 'and (mapcar 'P_Pu_3D lPu)))
(defun P_lPu      (lPu)               (apply 'and (mapcar 'P_Pu    lPu)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_WCS      (ucs_vers)          (equal ucs_vers '(0.0 0.0 1.0)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI SULLE LISTE
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun lis_car     (lis)     (mapcar 'car     lis))
(defun lis_cadr    (lis)     (mapcar 'cadr    lis))
(defun lis_caddr   (lis)     (mapcar 'caddr   lis))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun l_del_dup   (l_raw / element l_clean)     (while l_raw        (setq     element   (car l_raw)
                                                                               l_clean   (cons element l_clean)
                                                                               l_raw     (vl-remove element l_raw)))
                                                 (reverse l_clean))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun rem_last    (lis)                         (reverse (cdr (reverse lis))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun lis_from_ss (ss / cont lis)
   (setq cont 0)
   (repeat         (sslength ss)
                   (setq     lis     (append lis (list (cdr (assoc 10 (entget (ssname ss cont))))))
                             cont    (1+ cont)))
 lis)
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun lis_*_from_2lis     (lis1 lis2) (mapcar '* lis1 lis2))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Sum_ln              (lis)               (apply '+ lis))
(defun Sum_ln*lm           (lis1 lis2)         (apply '+ (mapcar '* lis1 lis2)))
(defun Sum_ln*lm*lo        (lis1 lis2 lis3)    (apply '+ (mapcar '* lis1 lis2 lis3)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONE DI ESTRAZIONE NOMI DI ENTITA'
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun entnext_from_ent_to_end (ent0 / ent1 lent)
   (while (setq ent1 (entnext ent0))
          (setq lent (cons ent1 lent) ent0 ent1)
          lent))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI DI ESTRAZIONE DATI DAI SOLIDI
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun estrai_spigoli_da_solido (ent0 erase / Pu0 ent1 lis_ent)
 (setq Pu0 '(0.0 0.0 0.0))
 (command-s "._copy" ent0 "" Pu0 Pu0)
 (setq ent1 (entlast))
 (command-s "._copy" ent1 "" Pu0 Pu0)
 (repeat 2 (foreach ent  (entnext_from_ent_to_end ent1)
                   (command-s "._explode" ent "")))
 (setq lis_ent (entnext_from_ent_to_end ent1))
 (if erase (command-s      "._erase"          ent0      ent1 "")
           (command-s      "._erase"                    ent1 ""))
 lis_ent)
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun estrai_vertici_da_solido (ent0 erase / eent Pu1 Pu2 lis_Pu)
   (foreach        ent (estrai_spigoli_da_solido ent0 erase)
                   (if ent (progn       (setq     eent                                 (vlax-ename->vla-object   ent)
                                                  Pu1                                  (vlax-curve-getStartPoint eent)
                                                  eent_open                 (not       (vlax-curve-isClosed      eent))
                                                  Pu2               (if eent_open      (vlax-curve-getEndPoint   eent)))
                                (command-s "._erase" ent "")))
                   (if Pu1 (setq lis_Pu (if (member Pu1 lis_Pu) lis_Pu (cons Pu1 lis_Pu))))
                   (if Pu2 (setq lis_Pu (if (member Pu2 lis_Pu) lis_Pu (cons Pu2 lis_Pu)))))
 (setq lis_Pu (l_del_dup lis_Pu)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI SULLE MATRICI
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun MA-TRA              (M)                 (apply 'mapcar (cons 'list M)))
(defun MA-TRA-O            (M)                 (reverse M))
(defun MA-TRA-V            (M)                 (mapcar 'reverse M))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun MA-DIV-R          (va ri)          (mapcar '(lambda (x) (/ x (float va))) ri))
(defun MA-GET-R          (M r)            (list (nth r M)))
(defun MA-GET-C          (M c)            (MA-TRA (MA-GET-R (MA-TRA M) c)))
(defun MA-DEL-R          (M r)            (vl-remove (nth r M) M))
(defun MA-DEL-C          (M c)            (MA-TRA (MA-DEL-R (MA-TRA M) c)))
(defun MA-APP-R          (M Mr)           (append M Mr))
(defun MA-APP-C          (M Mc)           (mapcar 'append M Mc))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun MA-SOLVE (Ma Mb / #va va Mab Mab2 Mab3)
 (setq Mab (MA-APP-C Ma Mb))
 (setq #va 0)
 (repeat (length Ma)
  (setq Mab2 nil)
  (foreach ri Mab
    (setq Mab2   (if   (= (nth #va ri) 0)     (append Mab2 (list ri))     (cons (MA-DIV-R (float (nth #va ri)) ri) Mab2))))
  (setq Mab nil)
  (foreach ri (cdr Mab2)    (setq Mab (append Mab (list  (if   (= (nth #va ri) 0)       ri    (mapcar '- ri (nth 0 Mab2)))))))
  (setq #va (1+ #va))
  (setq Mab3 (append Mab3 (list (nth 0 Mab2)))))
 (setq Mab (MA-TRA-O Mab3))
 (setq Mab (MA-APP-C (MA-TRA-V (MA-DEL-C Mab (length Mab)))
                (MA-GET-C Mab (length Mab))))
 (setq #va 0 Mab3 nil)
 (repeat (length Ma)
   (setq Mab2 nil)
   (foreach ri Mab
     (setq Mab2 (append Mab2 (list  (if   (= (nth #va ri) 0)      ri       (MA-DIV-R (float (nth #va ri)) ri))))))
   (setq Mab nil)
   (foreach ri (cdr Mab2)
     (setq Mab  (append Mab  (list (if (= (nth #va ri) 0)       ri       (mapcar '- ri (nth 0 Mab2)))))))
   (setq #va (1+ #va))
   (setq Mab3 (append Mab3 (list (nth 0 Mab2)))))
 (setq Mab (MA-TRA-O Mab3))
 (setq Mab (MA-APP-C (MA-TRA-V (MA-DEL-C Mab (length Mab)))
                (MA-GET-C Mab (length Mab))))
 (MA-GET-C Mab (length Mab)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI DI GESTIONE DI LISTE DI PUNTI
;-------------------------------------------------------------------------------------------------------------------------------------------------------
 (defun lPu3D_->_lPu2D   (lPu3D)     (mapcar 'rem_last lPu3D))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun lPu3D_->_lvPu3D  (lPu3D)     (apply 'append lPu3D))
(defun lPu2D_->_lvPu2D  (lPu2D)     (apply 'append lPu2D))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun lPu3D_->_lvPu2D  (lPu3D)     (apply 'append (lPu3D_->_lPu2D lPu3D)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun lvPu2D_->_lPu2D  (lvPu2D / Pu lPu2D)         (while lvPu2D  (setq Pu      (list (car lvPu2D) (cadr lvPu2D))
                                                                          lvPu2D  (cddr lvPu2D)
                                                                          lPu2D   (append lPu2D (list Pu)))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun lvPu3D_->_lPu3D  (lvPu3D / Pu lPu3D)          (while   lvPu3D  (setq    Pu  (list (car lvPu3D) (cadr lvPu3D) (caddr lvPu3D))
                                                                               lvPu3D  (cdddr lvPu3D)
                                                                               lPu3D   (append lPu3D (list Pu)))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun lvPu2D_->_lPu3D  (lvPu2D Puz / Pu lPu2D)     (while lvPu2D  (setq     Pu      (list (car lvPu2D) (cadr lvPu2D) Puz)
                                                                               lvPu2D  (cddr lvPu2D)
                                                                               lPu2D   (append lPu2D (list Pu)))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun lvPu2DT_->_lPu3D (lvPu2D Puz vers / Pu lPu2D)
      (while lvPu2D                                                    (setq   Pu       (trans (list (car lvPu2D) (cadr lvPu2D) Puz) vers 0)
                                                                               lvPu2D   (cddr lvPu2D)
                                                                               lPu2D    (append lPu2D (list Pu)))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun ord_cre_pos   (lista pos)          (vl-sort lista (function (lambda (e1 e2) (< (nth pos e1)(nth pos e2))))))
(defun ord_dec_pos   (lista pos)          (vl-sort lista (function (lambda (e1 e2) (> (nth pos e1)(nth pos e2))))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI E COMANDI SULLE ENTITA' GRAFICHE
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun extract_PT (ent / ObjectName Elevation Radius SplineMethod Coordinates Normal FitPoints Center StartPoint EndPoint MajorAxis
MinorAxis)
 (setq      ObjectName        (get_vval ent "ObjectName"   )          Elevation        (get_vval ent "Elevation"    )
            Radius            (get_vval ent "Radius"       )          SplineMethod     (get_vval ent "SplineMethod" ))
 (put_vval ent "SplineMethod" 0)
 (setq      Coordinates       (get_vlis ent "Coordinates"  )          FitPoints        (get_vlis ent "FitPoints"    )
            Normal            (get_vlis ent "Normal"       )          Center           (get_vlis ent "Center"       )
            StartPoint        (get_vlis ent "StartPoint"   )          EndPoint         (get_vlis ent "EndPoint"     )
            MajorAxis         (get_vlis ent "MajorAxis"    )          MinorAxis        (get_vlis ent "MinorAxis"    ))
 (put_vval ent "SplineMethod" SplineMethod)
 (setq lis_PT  (cond     ((equal ObjectName "AcDbPolyline"   ) (if (P_WCS Normal)
                                                                   (lvPu2D_->_lPu3D    Coordinates  Elevation)
                                                                   (lvPu2DT_->_lPu3D   Coordinates  Elevation Normal)))
                         ((equal ObjectName "AcDb3dPolyline" )    (lvPu3D_->_lPu3D    Coordinates))
                         ((equal ObjectName "AcDbSpline"     )    (lvPu3D_->_lPu3D    FitPoints))
                         ((equal ObjectName "AcDbLine"       )    (list  StartPoint  EndPoint))
                         ((equal ObjectName "AcDbCircle"     )
                              (list  Center  (trans (polar (trans Center 0 Normal) 0.0 Radius) Normal 0) Radius  Normal))
                         ((equal ObjectName "AcDbEllipse"    )
                             (list  Center  (mapcar '+ Center MajorAxis) (mapcar '+ Center MinorAxis) Normal))
                         ((equal ObjectName "AcDb3dSolid"    )    (estrai_vertici_da_solido ent nil)))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
 (defun extract_length (ent / ent_Length)
  (setq ObjectName  (get_vval ent "ObjectName"))
  (setq ent_Length  (cond      ((equal ObjectName "AcDbPolyline"     ) (get_vval ent "Length"       ))
                               ((equal ObjectName "AcDb3dPolyline" ) (get_vval ent "Length"       ))
                               ((equal ObjectName "AcDbSpline"       ) (vlax-curve-getDistAtParam ent (vlax-curve-getEndParam ent)))
                               ((equal ObjectName "AcDbLine"         ) (get_vval ent "Length"       ))
                               ((equal ObjectName "AcDbCircle"       ) (get_vval ent "Circumference" ))
                               ((equal ObjectName "AcDbEllipse"      ) (vlax-curve-getDistAtParam ent (vlax-curve-getEndParam ent)))
                               ((equal ObjectName "AcDb3dSolid"    ) nil))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Erase_ent       (ent)      (VAR-0)    (vl-cmdf "._erase"       ent "")                          (VAR-P))
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI GET PRECONFEZIONATE
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun get_opz  (§ini §txt §opz §def / opz)  (initget §ini) (setq opz (getkword (strcat "\n" §txt §opz))) (if (not opz) (setq opz §def)) opz)
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun get_yes_no       (§txt)   (get_opz  "Si No"                  §txt  " [Si No] <Si>: "                    "Si"))
(defun get_no_yes       (§txt)   (get_opz  "Si No"                  §txt  " [Si No] <No>: "                    "No"))
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun get_tol_num   (§txt) (get_opz  "Tolleranza Numero"               §txt " [Tolleranza Numero] <Tolleranza>: "          "Tolleranza"))
(defun get_dis_num   (§txt) (get_opz  "Distanza Numero"                 §txt " [Distanza Numero] <Distanza>: "              "Distanza"))
(defun get_tipo_ent  (§txt) (strcat "._" (get_opz  "Pline Spline 3Dpoly"  §txt " [Pline Spline 3Dpoly] <Pline>: "          "Pline")))
(defun get_sem_pon   (§txt) (get_opz  "Semplice Lineare Geometrica"     §txt " [Semplice Lineare Geometrica] <Semplice>: "  "Semplice"))
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun get_dist   (§txt dist0 / dist)     (if dist0 (progn (setq dist (getdist (strcat "\n" §txt " <"(rtos dist0 2 4) ">: ")))
                                                           (setq dist (if dist dist dist0)))
                                                  (progn (initget 7)
                                                           (setq dist (getdist (strcat "\n" §txt ": "))))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun get_int   (§txt int0 / int)        (if int0  (progn (setq int (getint (strcat "\n" §txt " <"(itoa int0) ">: ")))
                                                           (setq int (if int int int0)))
                                                  (progn (initget 7)
                                                           (setq int (getint (strcat "\n" §txt ": "))))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun get_ss_ent   (§txt tipo_ent) (print (strcat "\n" §txt ": ")) (ssget (list (cons 0 tipo_ent))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun get_ss_PT    (§txt)  (get_ss_ent §txt "POINT"))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
; Funzioni e comandi relativi all'algoritmo di Douglas Pecker
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun PT_farest_cur (lP1 cur) ; funziona su WCS - 8 minuti
(setq df 0)
(foreach PT lP1
  (setq PTd (distance PT (vlax-curve-getClosestPointTo cur PT T)))
  (if (> PTd df) (setq PTf PT df PTd))
  (append PTf (list df))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun lP_set_nil_all   (lP)      (mapcar '(lambda (x) (append x '(nil))) lP))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun lP_Tnil_set_P_T        (P lP_Tnil)   (subst (append P '(T)) (append P '(nil)) lP_Tnil))
(defun lP_Tnil_set_start_P_T  (lP_Tnil)     (lP_Tnil_set_P_T (rem_last (car  lP_Tnil)) lP_Tnil))
(defun lP_Tnil_set_end_P_T    (lP_Tnil)     (lP_Tnil_set_P_T (rem_last (last lP_Tnil)) lP_Tnil))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun lP_Tnil_sel_T    (lP_Tnil / lP_sel) (foreach P_Tnil lP_Tnil (setq lP_sel (if (last P_Tnil)        (append lP_sel (list (rem_last P_Tnil)))
lP_sel  ))))
(defun lP_Tnil_sel_nil (lP_Tnil / lP_sel) (foreach P_Tnil lP_Tnil (setq lP_sel (if (last P_Tnil) lP_sel (append lP_sel (list (rem_last P_Tnil)))
))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun dppl_fA ( / c_ori lPc_ori nPc_ori lPC01 lPc00 opz)
 (setq c_ori     (car (entsel "\nSelezionare la polilinea da ottimizzare: ")))
 (setq opz       (get_tol_num "Scegliere se fornire una distanza di tolleranza o un numero di punti"))

 (setq lPc_ori   (l_del_dup (extract_PT c_ori))
nPc_ori   (length lPc_ori)
lPC01     (lP_Tnil_set_end_P_T  (lP_Tnil_set_start_P_T  (lP_set_nil_all  lPc_ori)))
       lPc00     (vl-remove (car  lPc_ori) (vl-remove (last lPc_ori) lPc_ori))
       c_ini     (Draw_pline2D (lP_Tnil_sel_T  lPc01)))
 (list lPc_ori nPc_ori lPC01 lPc00 c_ini opz))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun dppl_fAc ( / c_ori c_par opz lPc_ori nPc_ori lPc_par nPc_par lPC01 lPc00)
 (setq c_ori   (car (entsel "\nSelezionare la polilinea originale non ottimizzata: ")))
 (setq c_par   (car (entsel "\nSelezionare la polilinea di parziale ottimizzazione associata: ")))
 (setq opz     (get_tol_num "Scegliere se fornire una distanza di tolleranza o un numero di punti"))

 (setq lPc_ori   (l_del_dup (extract_PT c_ori))  nPc_ori   (length lPc_ori)
lPc_par   (l_del_dup (extract_PT c_par))  nPc_par   (length lPc_par)
       lPc00     lPc_ori
lPc01     (lP_set_nil_all  lPc_ori))
 (foreach PT lPc_par  (setq lPc01  (lP_Tnil_set_P_T  PT  lPc01)
                        lPc00  (vl-remove PT  lPc00)))
 (list lPc_ori nPc_ori lPc_par nPc_par lPC01 lPc00 c_ori c_par opz))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun dppl_fB (num_PT c_ini lPc00 lPc01 / PF lPC11)
  (repeat num_PT (setq PF     (rem_last (PT_farest_cur    lPc00    c_ini))
                       lPc01  (lP_Tnil_set_P_T  PF lPc01)
                       lPc00  (vl-remove        PF lPc00)
                            lPC11  (lP_Tnil_sel_T lPC01))
              (put_vvertex_2D c_ini (vl-position PF lPC11) PF)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun dppl_fC (dis_min c_ini lPc00 lPc01 / run Pf+dis PF disF lPC11)
  (setq run T)
  (while run  (setq Pf+dis  (PT_farest_cur  lPc00    c_ini)
                    PF      (rem_last Pf+dis)
                    disF    (last Pf+dis))
            (if (> disF dis_min)
               (progn  (setq lPc01  (lP_Tnil_set_P_T  PF lPc01)
                             lPc00  (vl-remove        PF lPc00)
                                    lPC11  (lP_Tnil_sel_T lPC01))
                       (put_vvertex_2D c_ini (vl-position PF lPC11) PF))
               (setq run nil))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
; DPPL - Douglas Pecker Poli Linea - ottimizzazione polilinea esistente con controllo della distanza o  con controllo della quantità di punti
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun c:dppl ( / lPc_ori nPc_ori lPc01 lPc00 c_ini opz dis_min run Pf+dis PF) ;funziona solo su WCS
  (VAR-0)
  (mapcar 'set '(lPc_ori nPc_ori lPC01 lPc00 c_ini opz) (dppl_fA))
  (cond ((= opz "Tolleranza")  (setq dis_min (get_dist "Distanza minima di tolleranza per i punti da ignorare"   dis_min))
                       (dppl_fC dis_min c_ini lPc00 lPc01))
 ((= opz "Numero")     (setq num_PT  (get_int  (strcat "Numero di punti (max " (itoa nPc_ori) " punti) da estrarre" ) num_PT))
                       (dppl_fB num_PT c_ini lPc00 lPc01)))
  (VAR-P))
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
; DPPLDC - come la precedente ma ripartendo dai dati precedenti cambiando la distanza o il numero di punti per un'approssimazione diversa
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun c:dpplc (/ lPc_ori nPc_ori lPc_par nPc_par lPc00 lPc01 c_ori c_par opz Pf+dis PT_far dis_far)
  (VAR-0)
  (mapcar 'set '(lPc_ori nPc_ori lPc_par nPc_par lPC01 lPc00 c_ori c_par opz) (dppl_fAc))

  (cond ((= opz "Tolleranza")  (setq dis_min (get_dist "Nuova distanza minima di tolleranza per i punti da ignorare"   dis_min))
                       (dppl_fC dis_min c_par lPc00 lPc01))
 ((= opz "Numero")     (setq num_PT    (get_int  (strcat "Numero di punti già estratti " (itoa nPc_par)  " su " (itoa nPc_ori) " della polilinea
originale"
                                                     "\nNumero di punti (max " (itoa (- nPc_ori nPc_par)) " punti) da estrarre ulteriormente")
num_PT))
                       (dppl_fB num_PT c_par lPc00 lPc01)))
  (VAR-P))
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------
; polilinea passante da n punti noti con possibilità di filtraggio dei punti
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun c:ent_from_PT ( / cont PTgen PTpro d_PTgen_EPOL lPT+dis lPT_ord) ;ok wcs per ora no ucs

 (setq scelta (if (and ssPT EPOL dis_lim) (get_no_yes "Vuoi usare i dati precedenti") "No"))
 (if (equal scelta "No") (setq ssPT (get_ss_ent "Selezionare i punti da unire in una polilinea" "POINT")
                               EPOL (car (entsel "\nSelezionare la polilinea traiettoria: "))))

 (setq clo_ent_tra  (vlax-curve-isClosed EPOL))
 (setq tipo_ent     (get_tipo_ent "Scegliere il tipo di oggetto che si vuole disegnare"))
 (setq scelta (get_no_yes "Vuoi impostare una distanza limite per ignorare i punti più lontani"))
  (if (equal scelta "Si") (setq dis_lim (get_dist "Distanza limite per i punti da ignorare" dis_lim))
                          (setq dis_lim nil))
 (VAR-0) (setq cont 0)
 (repeat (sslength ssPT)
       (setq PTgen        (cdr (assoc 10 (entget (ssname ssPT cont))))
      PTpro        (vlax-curve-getClosestPointTo EPOL PTgen T)    ;T estende la curva
                   cont         (1+ cont))
       (if (or (not dis_lim)
               (< (distance PTgen PTpro) dis_lim))
          (setq lPT+dis   (append lPT+dis (list (append PTgen (list (vlax-curve-getDistAtPoint EPOL PTpro))))))))

 (setq lPT_ord (mapcar 'rem_last (ord_cre_pos lPT+dis 3)))

 (cond ((= tipo_ent "._Pline"  ) (Draw_pline2D  lPT_ord) (if clo_ent_tra (put_vval (entlast) "Closed"  -1)))
((= tipo_ent "._3Dpoly" ) (Draw_pline3D  lPT_ord) (if clo_ent_tra (put_vval (entlast) "Closed"  -1)))
((= tipo_ent "._Spline" ) (Draw_spline   lPT_ord) (if clo_ent_tra (put_vval (entlast) "Closed2" -1))))
 (Draw_pline2D lPT_ord)  (VAR-P))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;Funzioni sulle medie semplici o ponderate
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun PT_med_lPT_semp  (lPT / SPT PTm) (setq SPT '(0 0 0)) (foreach PT lPT (setq SPT (mapcar '+ SPT PT))) (setq PTm (mapcar '(lambda
(x) (/ x (length lPT))) SPT)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun gen_pesi_mm (ord_mm tipo_mm / cont peso lis_pesi)
 (setq cont 1 peso 1 lis_pesi nil)
 (cond ((= tipo_mm "Semplice"  ) (repeat ord_mm  (setq lis_pesi (append lis_pesi (list peso)))))
((= tipo_mm "Lineare"   ) (repeat ord_mm  (setq lis_pesi (append lis_pesi (list peso)))
                                  (if (< cont (/ (1+ ord_mm) 2))  (setq cont (1+ cont) peso (+ peso 1))
                                                                  (setq cont (1+ cont) peso (- peso 1)))))
((= tipo_mm "Geometrica") (repeat ord_mm  (setq lis_pesi (append lis_pesi (list peso)))
                                  (if (< cont (/ (1+ ord_mm) 2))  (setq cont (1+ cont) peso (* peso 2))
                                                                  (setq cont (1+ cont) peso (/ peso 2))))))
 lis_pesi)
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun prodotto_peso_PT (peso PT) (mapcar '(lambda (x) (* x peso)) PT))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun PT_med_lPT_pond  (lis_PT tipo_mm / ord_mm lis_pesi_mm tot_pesi_mm lis_PT_pes somma_PT_pes PT_med_sem PT_med_pes)
 (if (= tipo_mm "Semplice") (progn (setq PT_med_sem     (PT_med_lPT_semp lis_PT)))
                            (progn (setq ord_mm         (length lis_PT)
                                         lis_pesi_mm    (gen_pesi_mm ord_mm tipo_mm)
                                         tot_pesi_mm    (apply '+ lis_pesi_mm)
                                         lis_PT_pes     (mapcar 'prodotto_peso_PT  lis_pesi_mm  lis_PT)
                                         somma_PT_pes  '(0 0 0))
                                   (foreach PT_pes  lis_PT_pes
                                            (setq somma_PT_pes (mapcar '+ somma_PT_pes PT_pes)))
                                   (setq PT_med_pes     (mapcar '(lambda (x) (/ x tot_pesi_mm)) somma_PT_pes)))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
;Funzioni e comandi per l'interpolazione di Lagrange e Bézier
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun PT_curve_interpole (ent_ori  ord_med  tipo_med / PT_ent_ori num_PT_ent_ori start_index end_index lPT_temp lPT_med cont)
  (setq PT_ent_ori      (l_del_dup (extract_PT ent_ori))
 num_PT_ent_ori  (length PT_ent_ori)
 start_index     (/ (- ord_med 1) 2)
 end_index       (- num_PT_ent_ori start_index))
  (setq lPT_med nil lPT_temp nil cont 0)
  (repeat end_index (repeat ord_med (setq lPT_temp (append lPT_temp (list (nth cont PT_ent_ori)))
                                          cont     (1+ cont)))
                (setq lPT_med  (append lPT_med (list (PT_med_lPT_pond lPT_temp tipo_med)))
                      lPT_temp nil
                      cont     (- cont (- ord_med 1))))
  lPT_med)
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun c:ent_interpole ( / ent_ori length_ent_ori num_PT_ent_ori clo_ent_ori opz1 opz2)

 (setq ent_ori         (car (entsel "\nSelezionare l'oggetto da rimappare: "))
clo_ent_ori      (vlax-curve-isClosed ent_ori)
      opz1            (get_tipo_ent "Scegliere il tipo di oggetto che si vuole disegnare")
tipo_mm         (get_sem_pon  "Scegliere il tipo di media mobile Semplice, ponderata Lineare o ponderata Geometrica" )
      ord_mm                (get_int      "Ordine della media mobile [3/5/7/9/...]:"  ord_mm))

 (VAR-0)
 (cond ((= opz1 "._Pline"  ) (Draw_pline2D  (PT_curve_interpole  ent_ori  ord_mm  tipo_mm)) (if clo_ent_ori (put_vval (entlast) "Closed"  -
1)))
((= opz1 "._3Dpoly" ) (Draw_pline3D  (PT_curve_interpole  ent_ori  ord_mm  tipo_mm)) (if clo_ent_ori (put_vval (entlast) "Closed"  -1)))
((= opz1 "._Spline" ) (Draw_spline   (PT_curve_interpole  ent_ori  ord_mm  tipo_mm)) (if clo_ent_ori (put_vval (entlast) "Closed2" -1))))
 (VAR-P))
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
;End Of File
;-------------------------------------------------------------------------------------------------------------------------------------------------------------
```