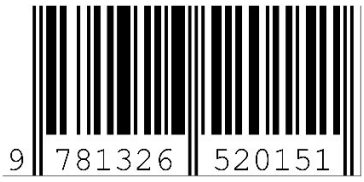
Algoritmi per l'elaborazione di estrazioni da nuvole di punti

Giovanni Anzani – 2015
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'elaborazione 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.

PROCEDURE E COMANDI REALIZZATI

La realizzazione in AutoLISP dei comandi descritti sopra, ha richiesto la realizzazione di un centinaio 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) Caricamento delle funzioni VL e delle funzioni VLAX. Per poter utilizzare nel codice realizzato delle funzioni VisualLISP è necessario provvedere al caricamento di un modulo aggiuntivo richiamabile dalla seguente funzione:

(vl-load-com) → carica le funzioni VisualLISP

2) Settaggio di costanti. In determinate condizioni avvicinandosi a valori nulli o a valori infiniti le capacità di calcolo di AutoLISP possono andare in crisi dando luogo a risposte erronee; per questa ragione è opportuno definire una coppia di costanti che consentano, con una ragionevole precisione, di approssimare il valore nullo e il valore infinito:

0.0 → vale 1e-8

inf → vale 1e+8

3) Inizializzazione di variabili d'ambiente di AutoCAD. Nella gestione dell'interoperabilità tra AutoLISP ed AutocCAD, come ad esempio nella fase di disegno automatizzato di entità grafiche nei disegni in AutoCAD comandati tramite procedure AutoLISP, è necessario effetuare alcuni settaggi preventivi, normalmente effettuati dall'operatore umano:

(BLI-1) → salva il valore corrente e setta la variabile "blipmode" pari a 1

(BLI-0) → salva il valore corrente e setta la variabile "blipmode" pari a 0

(OSM-0) → salva il valore corrente esetta la variabile "osmode" pari a 0

(CMD-0) → salva il valore corrente e setta la variabile "cmdecho" pari a 0

(SPL-0) → salva i valori correnti e setta le variabile " splinetype " pari a 5 e "splinesegs" pari a 256

(UCS-0) → salva il valore corrente dell'UCS assegnando un nome a tale UCS

4) Ripristino di variabili d'ambiente di AutoCAD. Prima di terminare la fase di interoperabilità tra AutoLISP ed AutocCAD, è necessario ripristinare le variabili d'ambiente modificate ai valori originali presenti in AutoCAD prima dell'avvio della procedura automatizzata di disegno:

(BLI-P) → setta la variabile "blipmode" al valore iniziale

(OSM-P) → setta la variabile "osmode" al valore iniziale

(CMD-P) → setta la variabile "cmdecho" al valore iniziale

(SPL-P) → setta le variabili " splinetype " e "splinesegs" ai valori iniziali

(UCS-P) → ripristina l'UCS iniziale in base al nome assegnatogli

5) Funzioni di modifica dell'UCS. Le funzioni che seguono permettono di modificare, quando necessario e in maniera automatica, il settaggio dell'UCS corrente, durante l'inrteroperabilità tra AutoLISP e AutoCAD:

(UCSW) → setta l'UCS al valore globale

(UCSP) → setta l'UCS al valore precedente

(UCS_3pu P1 P2 P3) → setta l'UCS in base ai tre punti forniti, quali origine del sistema di rifarimento, direzione dell'asse x e direzione dell'asse y; i tre punti forniti non dovranno risultare allineati

6) Settaggi di inizializzazione e ripristino raggruppati. Le funzioni che seguono permettono di effettuare settaggi multipli contemporaneamente relativamente alle variabili d'ambiente descritte ai punti 3 e 4:

(VAR-0) → setta una serie di variabili d'ambiente ai valori richiesti in fase iniziale tipicamente dalle procedure di disegno durante l'interoperabilità tra AutoLISP e AutoCAD

(VAR-P) → ripristina una serie di variabili d'ambiente ai valori iniziali a lori assegnati in ambiente AutoCAD, prima dell'uso delle procedure di disegno, durante l'interoperabilità tra AutoLISP e AutoCAD

(VAR-P+enla) → come la precedente, ma fornendo in uscita, il nome dell'ultima entità grafica creata in AutoCAD, durante l'interoperabilità tra AutoLISP e AutoCAD

7) Funzioni VLAX. Le funzioni che seguono permettono estrarre valori dalle proprietà degli oggetti presenti nel database di autocad o di inserire in tali proprietà dei nuovi valori:

(get_vval) → estrae un valore numerico da una proprietà come potrebbe essere il valore del raggio di una circonferenza

(get_vlis) → estrae una lista da una proprietà come potrebbe essere il valore delle coordinate (x y z) del centro di una circonferenza

(put_vval) → inserisce un valore numerico in una proprietà come potrebbe essere il valore del raggio di una circonferenza, in questo modo si cambierebbe, nel disegno, il raggio della circonferenza

8) Funzioni di disegno. Le seguenti funzioni sono state predisposte per ottimizzare il disegno automatizzato in ambiente AutoCAD di alcune entità grafiche; tali procedure si avvalgono, in entrata e in uscita, delle funzioni definite al punto 6.

(Draw_lPu lPu) → disegna una serie di punti forniti tramite una lista

(Draw_pline2D lPu) → disegna una polilinea 2D passante per i punti forniti tramite una lista

(Draw_pline3D lPu) → disegna una polilinea 3D passante per i punti forniti tramite una lista

(Draw_spline lPu) → disegna una spline passante per i punti forniti tramite una lista

(Draw_xli_2Pu Pu1 Pu2) → disegna una xlinea passante per i due punti forniti

(Draw_lin_2Pu Pu1 Pu2) → disegna una linea passante per i due punti forniti

(Draw_Cer_Cr Cen rag) → disegna un cerchio avente per centro il punto fornito e come raggio il valore fornito

9) Predicati. I predicati, sono funzioni particolari che restituiscono semplicemente il risultato di una verifica logica che abbia come risultato il valore vero o il valore falso, questo in AutoLISP si traduce nella restituzione o del valore T (vero) o del valore nil (falso):

(P_ucsname ucsname) → verifica se il nome fornito è presente nel database di autocad quale nome di un UCS precedentemente salvato

(P_Pu_2D Pu1) → verifica se il dato fornito è una lista con una coppia di valori, tale da poter corrispondere alle coordinate x e y di un punto

(P_Pu_3D Pu1) → verifica se il dato fornito è una lista con una terna di valori, tale da poter corrispondere alle coordinate x, y e z di un punto

(P_Pu Pu1) → verifica se il dato fornito è una lista con una coppia o con una terna di valori, tale da poter corrispondere alle coordinate x, y o alle coordinate x, y e z di un punto (P_lPu_2D lPu) → verifica se il dato fornito è

(P_lPu_2D lPu) → verifica se il dato fornito è una lista di punti 2D

(P_lPu_3D lPu) → verifica se il dato fornito è una lista di punti 3D

(P_lPu lPu) → verifica se il dato fornito è una lista di punti 2D o 3D

(P_WCS ucs_vers) → verifica se il dato fornito è una terna di valori corrispondenti al versore globale (0.0 0.0 1.0)

10) Funzioni sulle liste. Le seguenti funzioni svolgono specifiche operazioni sulle liste, frequentemente tali liste rappresenteranno liste di punti, per questa ragione si riporta, per alcune funzioni, oltre la descrizione teorica della loro operatività anche la descrizione di un esempio d'uso nel caso delle liste di punti:

(lis_car lis) → restituisce il primo elemento di ogni sotto lista contenuta nella lista fornita, ovvero restituisce una lista contenente la coordinata x di ciascun punto contenuto nella lista

(lis_cadr lis) → restituisce il secondo elemento di ogni sotto lista contenuta nella lista fornita, ovvero restituisce una lista contenente la coordinata y di ciascun punto contenuto nella lista

(lis_caddr lis) → restituisce il terzo elemento di ogni sotto lista contenuta nella lista fornita, ovvero restituisce una lista contenente la coordinata z di ciascun punto contenuto nella lista

(l_del_dup lis) → elimina eventuali elementi duplicati dalla lista, ovvero elimina punti duplicati

(rem_last lis) → elimina l'ultimo elemento dalla lista, ovvero elimina l'ultimo punto presente in lista

(lis_from_ss ss) → dato un set di selezione contenente punti disegnati in AutoCAD, restituisce una lista contenente l'estrazione dal disegno dei valori delle cordinate dei punti;tale funzione è applicabile solo a set di selezione contenenti punti

(lis_*_from_2lis lis1 lis2) → date due liste di valori restituisce una lista contenente il prodotto tra gli elementi corrispondenti delle due liste, tale funzione non ha una specifica applicazione nel caso dei punti

(Sum_ln lis) → data una lista di valori restituisce il valore corrispondente alla sommatoria di tali valori, tale funzione non ha una specifica applicazione nel caso dei punti

(Sum_ln*lm lis1 lis2) → date due liste di valori restituisce il valore corrispondente alla sommatoria dei prodotti tra gli elementi corrispondenti delle due liste; se le due liste sono uguali il risultato coincide con la sommatoria dei quadrati degli elementi della lista, tale funzione non ha una specifica applicazione nel caso dei punti

(Sum_ln*lm*lo lis1 lis2 lis3) → date tre liste di valori restituisce il valore corrispondente alla sommatoria dei prodotti tra gli elementi corrispondenti delle tre liste; se le tre liste sono uguali il risultato coincide con la sommatoria dei cubi degli elementi della lista, tale funzione non ha una specifica applicazione nel caso dei punti

11) Funzione di estrazione nomi di entità seguenti. La seguente funzioni svolge la specifica operazione sotto descritta, necessaria per l'interoperabilità tra AutoLISP ed AutoCAD:

(entnext_from_ent_to_end ent0) → partendo dal nome dell'entità grafica presente nel disegno di AutoCAD, la funzione restituisce una lista contenente i nomi di tutte le entità grafiche presenti nel database di AutoCAd e successive a quella fornita, ovvero che sono state disegnate in un tempo successivo

12) Funzioni di estrazione dati dai solidi. Le seguenti funzioni suno state specificatamente relaizzate per estrarre i vertici e gli spigoli dei solidi in autocad.

(estrai_spigoli_da_solido ent0 erase) → estrae gli spigoli dall'entità solida fornita e disegna per ciascuno di essi una specifica entità di autocad che lo rappresenti; a seconda del valore del parametro erase, cancella l'entità fornita o la lascia nel disegno; la funzione restituisce una lista con i nomi di tutte le entità spigolo realizzate dalla procedura

(estrai_vertici_da_solido ent0 erase) → estrae dall'entità fornita le coordinate dei suoi vertici e restituisce una lista di tali punti; a seconda del valore del parametro erase, cancella l'entità fornita o la lascia nel disegno;

13) Funzioni sulle matrici. Le seguenti funzioni svolgono specifiche operazioni sulle matrici: è stato necessario definire tali funzioni perchè sfortunatamente l'ambiente AutoLISP non ha specifici comandi dedicati alle matrici:

(MA-TRA M) → data una matrice M restituisce la sua trasposta

(MA-TRA-O M) → data una matrice M restituisce una matrice nella quale le righe sono invertite di posizione senza modificare il loro contenuto, ovvero la prima riga inverte la sua posizione con l'ultima, la seconda con la penultima e via dicendo. Se il numero delle righe è dispari la riga centrale rimane invariata

(MA-TRA-V M) → data una matrice M restituisce una matrice nella quale le colonne sono invertite di posizione senza modificare il loro contenuto, ovvero la prima colonna inverte la sua posizione con l'ultima, la seconda con la penultima e via dicendo. Se il numero delle colonne è dispari la colonna centrale rimane invariata

(MA-DIV-R va ri) → data una lista di valori ri appresentante una riga di una matrice divide ogni valore della lista per il valore va fornito; la funzione restituisce la lista di valori rappresentante la riga modificata

(MA-GET-R M r) → data una matrice M restituisce la riga indicata dall'indice r; la prima riga ha indice 0

(MA-GET-C M c) → data una matrice M restituisce la colonna indicata dall'indice c; la prima colonna ha indice 0

(MA-DEL-R M r) → data una matrice M ne elimina la riga indicata dall'indice r; la prima riga ha indice 0; la funzione restituisce la matrice risultante

(MA-DEL-C M c) → data una matrice M ne elimina la colonna indicata dall'indice r; la prima colonna ha indice 0; la funzione restituisce la matrice risultante

(MA-APP-R M Mr) → data una matrice M aggiunge ad essa una riga fornita come matrice Mr contenente una sola riga

(MA-APP-C M Mc) → data una matrice M aggiunge ad essa una colonna fornita come matrice Mc contenente una sola colonna

(MA-SOLVE Ma Mb) → dato un sistema di equazioni lineari definito tramite le due matrici Ma ed Mb fornisce le soluzioni del sistema

14) Funzioni di gestione di liste di punti. Le seguenti funzioni svolgono specifiche operazioni sulle liste di punti occupandosi della conversione tra i diversi formati di gestione che, in diversi contesti di AutocAD, risultano essere differenti oppure occupandosi di compiere specifiche operazioni sulle liste:

(lPu3D_->_lPu2D lPu3D) → data una lista di punti 3D ciascuno di essi espresso nella forma (x y z) la converte in una lista di punti 2D espressi nella forma (x y)

(lPu3D_->_lvPu3D lPu3D) → data una lista di punti 3D espressa nella forma ((x1 y1 z1) … (xn yn zn)) la converte in una lista di punti 3D espressa nella forma (x1 y1 z1 … xn yn zn)

(lPu2D_->_lvPu2D lPu2D) → data una lista di punti 2D espressa nella forma ((x1 y1) … (xn yn)) la converte in una lista di punti 3D espressa nella forma (x1 y1 … xn yn)

(lPu3D_->_lvPu2D lPu3D) → data una lista di punti 3D espressa nella forma ((x1 y1 z1) … (xn yn zn)) la converte in una lista di punti 2D espressa nella forma (x1 y1 … xn yn)

(lvPu2D_->_lPu2D lvPu2D) → data una lista di punti 2D espressa nella forma (x1 y1 … xn yn) la converte in una lista di punti 2D espressa nella forma ((x1 y1) … (xn yn))

(lvPu3D_->_lPu3D lvPu3D) → data una lista di punti 3D espressa nella forma (x1 y1 z1 … xn yn zn) la converte in una lista di punti 3D espressa nella forma ((x1 y1 z1) … (xn yn zn))

(lvPu2D_->_lPu3D lvPu2D Puz) → data una lista di punti 2D espressa nella forma (x1 y1 … xn yn) la converte in una lista di punti 3D espressa nella forma ((x1 y1 Puz) … (xn yn Puz)) inserendo come coordinata z il valore fornito Puz

(lvPu2DT_->_lPu3D lvPu2D Puz vers) → funzione analoga alla precedente, ma che si occupa anche di trasformare le coordinate di ogni punto, da un sistema di riferimento utente specificato nella variabile vers, al sistema di riferimento globale

(ord_cre_pos lista pos)) → data una lista di punti 2D o 3D ciascuno di essi espresso nella forma (x y) o (x y z) la ordina in senso crescente secondo la componente x y o z a seconda del valore specificato nella variabile pos

(ord_dec_pos lista pos)) → data una lista di punti 2D o 3D ciascuno di essi espresso nella forma (x y) o (x y z) la ordina in senso crescente secondo la componente x y o z a seconda del valore specificato nella variabile pos

15) Funzioni e comandi sulle entità grafiche. Le seguenti funzioni si occupano di consentire l'interoperabilità tra Autolisp ed AutoCAD relativamente alle entità grafiche:

(extract_PT ent) → data un'entità grafica appartenente ad una lista di tipologie accettate, restituisce una serie di informazioni numeriche rilevate nel database di AutoCAD relativamente a tale entità; tali informazioni dipendono dal tipo di entità selezionata

(c:ent->PT) → come la precedente funzione ma in forma di comando reso di sponibile all'interno di AutoCAD

(extract_length ent) → data un'entità grafica appartenente ad una lista di tipologie accettate, restituisce la lunghezza del suo perimetro, rilevata nel database di AutoCAD relativamente a tale entità

(c:ent->length) → come la precedente funzione ma in forma di comando reso di sponibile all'interno di AutoCAD

(Erase_ent ent) → elimina dal database di AutoCAD l'entità grafica di cui viene fornito il nome nella variabile ent

16) Funzioni GET preassemblate. Le seguenti funzioni si occupano di consentire l'interoperabilità tra Autolisp ed AutoCAD relativamente al recupero di valori specifici dall'ambiente AutoCAD per l'uso in AutoLISP:

(get_opz §ini §txt §opz §def) → una funzione generalista che tramite le quattro stringhe di testo fornite permette di assemblare agevolmente diverse funzioni per la scelta di valori opzionali all'interno di liste convenientemente predisposte occupandosi al contempo di indicare un valore di default da scegliere semplicemente con l'invio

(get_yes_no §txt) → permette di scegliere tra Si e No fornendo il messaggio indicato nella variabile §txt; il valore di default è il primo della lista

(get_no_yes §txt) → permette di scegliere tra No e Si fornendo il messaggio indicato nella variabile §txt; il valore di default è il primo della lista

(get_tol_num §txt) → permette di scegliere tra Tolleranza e Numero fornendo il messaggio indicato nella variabile §txt; il valore di default è il primo della lista

(get_dis_num §txt) → permette di scegliere tra Distanza e Numero fornendo il messaggio indicato nella variabile §txt; il valore di default è il primo della lista

(get_tipo_ent §txt) → permette di scegliere tra Pline, Spline e 3Dpoly fornendo il messaggio indicato nella variabile §txt; il valore di default è il primo della lista

(get_sem_pon §txt) → permette di scegliere tra Semplice Lineare e Geometrica fornendo il messaggio indicato nella variabile §txt; il valore di default è il primo della lista

(get_dist §txt dist0) → permette di scegliere una distanza fornendo il messaggio indicato nella variabile §txt; il valore di default è fornito nella variabile dist0

(get_int §txt int0) → permette di scegliere un numero intero fornendo il messaggio indicato nella variabile §txt; il valore di default è fornito nella variabile int0

(get_ss_ent §txt tipo_ent) → una funzione generalista realizzata per permettere la realizzazione speditiva di funzioni dedicate alle selezioni di entità grafiche nel database di AutoCAD specificando come filtro della selezione il tipo di entita e fornendo il messaggio indicato nella variabile §txt

(get_ss_PT §txt) → permettere la selezione di entità di tipo punto dal database di AutoCAD fornendo il messaggio indicato nella variabile §txt

17) Cerchio di miglior approssimazione tramite scarto quadratico medio noti n punti. La funzione ed il comando che seguono sono rispettivamente dedicati alla determinazione dei prarametri e al disegno della circonferenza:

(sqm_cer lis_Pu) → tale funzione permette di determinare, utilizzando lo scarto quadratico medio, i coefficienti dell'equazione della circonferenza che risulti essere la miglior approssimazione, noti i punti forniti nella lista lis_Pu

(c:cer_from_lPT) → tale comando permette di rendere disponibile in AutoCAD la funzionalità di determinazione della circonferenza sopra descritta, occupandosi di acquisire i punti noti, determinare i coefficienti della circonferenza e infine disegnare la circonferenza

18) Retta e Segmento di miglior approssimazione tramite scarto quadratico medio noti n punti. La funzione ed i comandi che seguono sono rispettivamente dedicati alla determinazione dei prarametri e al disegno della retta (xline) e del segmento (linea):

(sqm_ret lis_Pu) → tale funzione permette di determinare, utilizzando lo scarto quadratico medio, i coefficienti dell'equazione della retta che risulti essere la miglior approssimazione, noti i punti forniti nella lista lis_Pu

(c:xlin_from_lPT) → tale comando permette di rendere disponibile in AutoCAD la funzionalità di determinazione della xline sopra descritta, occupandosi di acquisire i punti noti, determinare i coefficienti della retta e infine disegnare la xline

(c:lin_from_lPT) → tale comando permette di rendere disponibile in AutoCAD la funzionalità di determinazione della linea sopra descritta, occupandosi di acquisire i punti noti, determinare i coefficienti della retta e infine disegnare la linea

19) Polilinea passante da n punti noti. Il comando che segue permette di ottimizzare la determinazione di una polilinea passante da una serie di punti forniti. Alla base di tale comando c'è l'idea di ordinare tali punti in maniera tale da determinare una polilinea non intersecantesi:

(c:upg) → tale comando permette di definire una direzione di orientamento ed un piano di disegno, finalizzati all'ottimizzazione della polilinea da determinarsi in base ai punti forniti durante l'interoperativita tra AutoLISP ed AutoCAD; la procedura permette altresì di scegliere il tipo di entità che verrà realizzata segliendo tra polilinea 2D polilinea 3D e spline

20) Rimappaggio di una polilinea. 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_remap ent delta_length) → tale funzione permette di determinare i punti che consentono di rimappare l'entità indicata nella variabile ent in base alla distanza tra due successivi punti di rimappaggio il cui valore viene fornito nella variabile delta_length

(c:ent_remap) → tale comando permette di definire i punti da utlilizzare per il rimappaggio dell'entità selezionata; è possibile sceglire se indicare la distanza tra due punti successivi o il numero di punti da utilizare e in funzione di tale numero determinare la distanza tra tali punti. Il comando permette di scegliere anche il tipo di entità da realizzare segliendo tra polilinea 2D, polilinea 3D e spline

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

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

[5] 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.)

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

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

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

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

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

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

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

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

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

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

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

[17] 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 l'intero codice sorgente delle procedure sopra 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)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;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 (ci sono duplicati da eliminare in entrambe le funzioni)
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(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 c:ent->PT      ( / lPu)            (VAR-0)  (setq lPu (extract_PT (car (entsel)))) (Draw_lPu lPu)   (VAR-P) lPu)
(defun c:ent->length  (/ ent_Length)      (VAR-0)  (setq ent_Length (extract_length (car (entsel))))       (VAR-P) ent_Length)
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(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"))
;-------------------------------------------------------------------------------------------------------------------------------------------
;cerchio da n punti con SQM
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun sqm_cer (lis_Pu / S1 Sxy ly Sy Syy Syyy Sxyy lx Sx Sxx Sxxx Sxxy H K X)
  (setq     lx    (lis_car  lis_Pu)           ly    (lis_cadr  lis_Pu)
            S1    (length lis_Pu)             Sx    (Sum_ln lx)             Sy    (Sum_ln ly)
            Sxx   (Sum_ln*lm lx lx)       Syy   (Sum_ln*lm ly ly)       Sxy   (Sum_ln*lm lx ly)
            Sxxx  (Sum_ln*lm*lo lx lx lx)         Syyy  (Sum_ln*lm*lo ly ly ly)
            Sxyy  (Sum_ln*lm*lo lx ly ly)         Sxxy  (Sum_ln*lm*lo lx lx ly))
  (setq     H  (list    (list Sxx Sxy Sx)             (list Sxy Syy Sy)             (list Sx  Sy  S1))
            K  (list    (list (- (+ Sxxx Sxyy)))      (list (- (+ Sxxy Syyy)))      (list (- (+ Sxx  Syy )))))
  (setq X (MA-SOLVE H K))
  (list   1   1          (car (nth 0 X))       (car (nth 1 X))       (car (nth 2 X))))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun c:cer_from_lPT ()
  (setq     lPT   (lis_from_ss (get_ss_PT "Selezionare i punti noti"))
            lcc   (sqm_cer lPT)
            ca    (nth 2 lcc)
            cb    (nth 3 lcc)
            cc    (nth 4 lcc)
            CE    (list (/ ca -2.0) (/ cb -2.0))
            ra    (/ (sqrt (+ (* ca ca)(* cb cb)(* -4 cc))) 2.0))
  (Draw_Cer_Cr      CE ra))
;-------------------------------------------------------------------------------------------------------------------------------------------
;Retta e segmento da n punti con SQM
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun sqm_ret   (lis_Pu / lx ly S1 Sx Sy Sxx Syy Sxy A B P1r P2r P1t P2t)
  (setq     lx   (lis_car  lis_Pu)            ly   (lis_cadr  lis_Pu)           S1  (length lis_Pu)
            Sx   (Sum_ln lx)                  Sy   (Sum_ln ly)
            Sxx  (Sum_ln*lm lx lx)        Syy  (Sum_ln*lm ly ly)        Sxy  (Sum_ln*lm lx ly))
  (setq     A (/        (- (* S1 Sxy)(* Sx Sy))
                        (- (* S1 Sxx)(* Sx Sx)))
            B (/        (- (* Sy Sxx)(* Sx Sxy))
                        (- (* S1 Sxx)(* Sx Sx))))
  (setq     P1r         (car  lis_Pu)         P1t         (list (car P1r) (+ (* A (car P1r)) B) 0.0)
            P2r         (last lis_Pu)         P2t         (list (car P2r) (+ (* A (car P2r)) B) 0.0))
  (list P1t P2t))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun c:xlin_from_lPT ()
  (setq lPT (lis_from_ss (get_ss_PT "Selezionare i punti noti")))
  (setq lPT (vl-sort lPT (function (lambda (e1 e2) (< (car e1) (car e2))))))
  (setq ret (sqm_ret lPT))
  (Draw_xli_2Pu  (car ret)(cadr ret)))
;-------------------------------------------------------------------------------------------------------------------------------------------
 (defun c:lin_from_lPT ()
  (setq lPT (lis_from_ss (get_ss_PT "Selezionare i punti noti")))
  (setq lPT (vl-sort lPT (function (lambda (e1 e2) (< (car e1) (car e2))))))
  (setq ret (sqm_ret lPT))
  (Draw_lin_2Pu  (car ret)(cadr ret)))
;-------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;polilinea da n punti
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun c:upg ()
 (UCS_3Pu          (getpoint "\n Primo punto asse x: ")
                   (getpoint "\n Secondo punto asse x: ")
                   (getpoint "\n Punto del piano da definire: "))
 (setq     sPT (ssget '((0 . "POINT")))
           #PT (sslength sPT))
 (setq     opz (get_tipo_ent "Scegliere il tipo di oggetto che si vuole disegnare"))
 (setq     cont 0 lPTs nil)
 (repeat #PT
   (setq   lPTs  (append lPTs (list (trans (cdr (assoc 10 (entget (ssname sPT cont)))) 0 1)))
           cont  (1+ cont)))
 (setq lPTs (vl-sort lPTs (function (lambda (e1 e2)  (< (car e1) (car e2))))))
 (cond     ((= opz "._Pline"  )            (Draw_pline2D  lPTs))
           ((= opz "._3Dpoly" )            (Draw_pline3D  lPTs))
           ((= opz "._Spline" )            (Draw_spline   lPTs)))
 (repeat 3 (UCSP)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;rimappaggio di una polilinea
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun PT_curve_remap  (ent delta_length / length_Curve delta_length next_length nextPT lPTr)  ; oggetto da rimappare, distanza tra i punti
 (setq length_Curve   (extract_length ent)
      next_length    0.0)
(while (< next_length length_Curve)
      (setq nextPT       (trans (vlax-curve-getPointAtParam  ent (vlax-curve-getParamAtDist  ent next_length))  0 1)
     next_length  (+ next_length delta_length))
      (setq lPTr (append lPTr (list nextPT))))
(setq lPTr (append lPTr (list (trans (vlax-curve-getPointAtParam ent (vlax-curve-getEndParam    ent)) 0 1)))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun c:ent_remap ( / ent_ori length_ent_ori num_PT_ent_ori clo_ent_ori opz1 opz2)
 (setq     ent_ori                (car (entsel "\nSelezionare l'oggetto da rimappare: "))
           length_ent_ori         (extract_length ent_ori)
           num_PT_ent_ori         (length (l_del_dup (extract_PT ent_ori)))
           clo_ent_ori            (vlax-curve-isClosed ent_ori)
           opz1                    (get_tipo_ent "Scegliere il tipo di oggetto che si vuole disegnare")
           opz2                   (get_dis_num  "Scegliere se fornire la distanza tra i punti o il numero di punti"))
 (cond     ((= opz2 "Distanza")
                   (setq dis_PT   (get_dist (strcat "Distanza tra i punti (max " (rtos length_ent_ori 2 4) ") di rimappaggio" )  dis_PT)
                         delta_length  dis_PT))
           ((= opz2 "Numero"  )
                   (setq num_PT (get_int  (strcat "Numero di punti (attuali "  (itoa num_PT_ent_ori)     ") di rimappaggio" )  num_PT)
                         delta_length  (/ length_ent_ori (- num_PT 1)))))
 (VAR-0)
 (cond     ((= opz1 "._Pline"  )
                   (Draw_pline2D  (PT_curve_remap  ent_ori  delta_length)) (if clo_ent_ori (put_vval (entlast) "Closed"  -1)))
           ((= opz1 "._3Dpoly" )
                   (Draw_pline3D  (PT_curve_remap  ent_ori  delta_length)) (if clo_ent_ori (put_vval (entlast) "Closed"  -1)))
           ((= opz1 "._Spline" )
                   (Draw_spline   (PT_curve_remap  ent_ori  delta_length)) (if clo_ent_ori (put_vval (entlast) "Closed2" -1))))
 (VAR-P))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;End Of File
;-------------------------------------------------------------------------------------------------------------------------------------------------------
```