Algoritmi di geometria descrittiva in AutoLISP su punti rette e piani

Giovanni Anzani – 2018
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 a rendere disponibili in AutoCAD alcune funzionalità utilizzate nell'ambito della Geometria Descrittiva. 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 Geometria Descrittiva, 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 altre pubblicazioni dello stesso autore[1]; si è ritenuto opportuno, inserirle comunque in appendice, per permettere la piena funzionalità del codice sorgente in appendice, senza la necessità di riferirsi al codice presente nelle precedenti pubblicazioni.

PROCEDURE E COMANDI REALIZZATI

La realizzazione in AutoLISP della dozzina di comandi descritti in seguito, 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) Settaggio di costanti. In diverse situazioni, ci si è dovuti riferire all'origine del sistema di riferimento, a seconda dei casi, globale (WCS) o utente (UCS); per questa ragione si è rivelato comodo definire un'opportuna costante rappresentante tale punto:

Pu0 → vale '(0.0 0.0 0.0).

2) 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 utile automatizzare alcuni settaggi preventivi, normalmente effettuati dall'operatore umano:

(PDS-6) → salva il valore corrente e setta la variabile "pdsize" pari a -6.

(PDM-34) → salva il valore corrente e setta la variabile "pdmode" pari a 34.

[1] Giovanni Anzani. 2015. *Algoritmi per l'elaborazione di estrazioni da nuvole di punti*. Lulu.
Giovanni Anzani. 2016. *Algoritmi per l'ottimizzazione di estrazioni da nuvole di punti*. Lulu

(UCS-0) → salva l'ucs corrente assegnandogli un nome nella lista degli ucs.

3) 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:

(PDS-P) → setta la variabile "pdsize" al valore iniziale

(PDM-P) → setta la variabile "pdmode" al valore iniziale

(UCS-P) → ripristina l'ucs precedentemente salvato recuperandolo in base al nome assegnatogli nella lista degli ucs

4) Settaggio dell'UCS di AutoCAD. Il sistema UCS (User Coordinate System) è un sistema di coordinate cartesiane utente XYZ mobile che definisce il piano di lavoro XY su cui giacciono la direzione orizzontale e quella verticale e che definisce implicitamente anche la direzione Z normale al piano di lavoro XY. Queste direzioni sono utili e indispensabili riferimenti geometrici nell'interoperabilità tra AutoLISP ed AutoCAD e vengono di volta in volta ridefinite dall'utente:

(UCS-P Pi1) → imposta l'origine e l'orientamento dell'UCS in maniera da far coincidere il piano di lavoro xy con il piano fornito nella variabile Pi1.

5) Funzioni di disegno. Le seguenti funzioni sono state predisposte per ottimizzare il disegno automatizzato in ambiente AutoCAD di alcune entità grafiche finalizzate alla rappresentazione degli enti geometrici fondamentali della Geometria descrittiva: punto, retta, piano. In tutte le procedure e in tutte la variabili definite nella presente trattazione, tali enti geometrici saranno indicati in maniera abbreviata con Pu per il punto, Re per la retta, Pi per il piano e numerati progressivamente in funzione della quantità richiesta nelle varie procedure.

(Draw_1Pu Pu1) → disegna il punto fornito nella variabile Pu1

(Draw_1Re Re1) → disegna la retta fornita nella variabile Re1

(Draw_1Pi Pi1) → disegna il piano fornito nella variabile Pi1

6) Funzioni sulle liste. Le seguenti funzioni svolgono specifiche operazioni sulle liste:

(1° lis) → restituisce il primo elemento della lista fornita nella variabile lis

(2° lis) → restituisce il secondo elemento della lista fornita nella variabile lis

(3° lis) → restituisce il terzo elemento della lista fornita nella variabile lis

(4° lis) → restituisce il quarto elemento della lista fornita nella variabile lis

7) Predicati. I predicati, sono funzioni particolari di verifica 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). In questa macro area vengono raggruppati i predicati relativi alla geometria descrittiva, ovvero quelli relativi: all'appartenenza, al parallelismo, alla perpendicolarità:

(P_coinc_1Pu_1Pu Pu1 Pu2) → verifica se i due punti forniti rispettivamente nelle due variabili Pu1 e Pu2 sono coincidenti

(P_coinc_1Re_1Re Re1 Re2) → verifica se le due rette fornite rispettivamente nelle due variabili Re1 e Re2 sono coincidenti

(P_coinc_1Pi_1Pi Pi1 Pi2) → verifica se i due piani forniti rispettivamente nelle due variabili Pi1 e Pi2 sono coincidenti

(P_allin_3Pu Pu1 Pu2 Pu3) → verifica se i tre punti forniti rispettivamente nelle tre variabili Pu1, Pu2 e Pu3 sono allineati

(P_compl_4Pu Pu1 Pu2 Pu3 Pu4) → verifica se i quattro punti forniti rispettivamente nelle quattro variabili Pu1, Pu2, Pu3 e Pu4 sono complanari

(P_compl_1Re_1Re Re1 Re2) → verifica se le due rette fornite rispettivamente nelle due variabili Re1 e Re2 sono complanari

(P_appar_1Pu_1Re Pu1 Re1) → verifica se il punto e la retta forniti rispettivamente nelle due variabili Pu1 e Re1 si appartengono

(P_appar_1Pu_1Pi Pu1 Pi1) → verifica se il punto e il piano forniti rispettivamente nelle due variabili Pu1 e Pi1 si appartengono

(P_appar_1Re_1Pi Re1 Pi1) → verifica se la retta e il piano forniti rispettivamente nelle due variabili Re1 e Pi1 si appartengono

(P_paral_2Ver Ver1 Ver2) → verifica se i due versori forniti rispettivamente nelle due variabili Ver1 e Ver2 sono paralleli

(P_paral_1Re_1Re Re1 Re2) → verifica se le due rette fornite rispettivamente nelle due variabili Re1 e Re2 sono parallele

(P_paral_1Re_1Pi Re1 Pi1) → verifica se la retta e il piano forniti rispettivamente nelle due variabili Re1 e Pi1 sono paralleli

(P_paral_1Pi_1Pi Pi1 Pi2) → verifica se i due piani forniti rispettivamente nelle due variabili Pi1 e Pi2 sono paralleli

(P_paral_1Re_1Re_1Re Re1 Re2 Re3) → verifica se le tre rette fornite rispettivamente nelle tre variabili Re1, Re2 e Re3 sono parallele. Questa funzione risulta utile in alcune casistiche particolari

(P_paral_1Pi_1Pi_1Pi Pi1 Pi2 Pi3) → verifica se i tre piani forniti rispettivamente nelle tre variabili Pi1, Pi2 e Pi3 sono paralleli. Questa funzione risulta utile in alcune casistiche particolari

(P_perpe_2Ver Ver1 Ver2) → verifica se i due versori forniti rispettivamente nelle due variabili Ver1 e Ver2 sono perpendicolari

(P_perpe_1Re_1Re Re1 Re2) → verifica se le due rette fornite rispettivamente nelle due variabili Re1 e Re2 sono perpendicolari

(P_perpe_1Re_1Pi Re1 Pi1) → verifica se la retta e il piano forniti rispettivamente nelle due variabili Re1 e Pi1 sono perpendicolari

(P_perpe_1Pi_1Pi Pi1 Pi2) → verifica se i due piani forniti rispettivamente nelle due variabili Pi1 e Pi2 sono perpendicolari

(P_perpe_1Re_1Re_1Re Re1 Re2 Re3) → verifica se le tre rette fornite rispettivamente nelle tre variabili Re1, Re2 e Re3 sono perpendicolari

(P_perpe_1Pi_1Pi_1Pi Pi1 Pi2 Pi3) → verifica se i tre piani forniti rispettivamente nelle tre variabili Pi1, Pi2 e Pi3 sono paralleli.

8) Funzioni sui vettori. Le seguenti funzioni svolgono specifiche operazioni sui vettori:

(modulo_2Pu Pu1 Pu2) → calcola la lunghezza del segmento avente per estremi i due punti forniti nelle due variabili Pu1 e Pu2

(vettore_2pu Pu1 Pu2) → calcola le componenti xyz del vettore avente per estremi i due punti forniti nelle due variabili Pu1 e Pu2. Ovvero calcola il vettore (Pu2 – Pu1)

(versore_2Pu Pu1 Pu2) → calcola le componenti xyz del versore orientato secondo la direzione e il verso espressi dai due punti forniti nelle due variabili Pu1 e Pu2.

(vettore_3Pu Pu1 Pu2 Pu3) → calcola le componenti xyz di un vettore normale al piano definito dai tre punti forniti nelle tre variabili Pu1, Pu2 e Pu3 e passante dall'origine del sistema di riferimento globale indicato come *Pu0*.

(versore_3Pu Pu1 Pu2 Pu3) → calcola le componenti xyz del versore normale al piano definito dai tre punti forniti nelle tre variabili Pu1, Pu2 e Pu3 e passante dall'origine del sistema di riferimento globale indicato come *Pu0*.

(prodotto_1Sc_1Ve Sc1 Ve1) → calcola il prodotto tra uno scalare e un vettore forniti rispettivamente nelle variabili Sc1 e Ve1.

(prodotto_scalare_2Ve Ve1 Ve2) → calcola il prodotto scalare tra i due vettori forniti nelle variabili Ve1 e Ve2.

(prodotto_vettoriale_2Ve Ve1 Ve2) → calcola il prodotto vettoriale tra i due vettori forniti nelle variabili Ve1 e Ve2.

(somma_vettoriale_2Ve Ve1 Ve2) → calcola la somma vettoriale tra i due vettori forniti nelle variabili Ve1 e Ve2.

(differenza_vettoriale_2Ve Ve1 Ve2) → calcola la differenza vettoriale tra i due vettori forniti nelle variabili Ve1 e Ve2.

9) Funzioni sui punti. Le seguenti funzioni svolgono specifiche operazioni sui punti:

(flat_z_1Pu Pu1) → restituisce un punto le cui componenti x e y coincidono con quelle x e y del punto fornito nella variabile Pu1 e la cui componente z corrisponde a 0.

(flat_xy_1Pu Pu1) → restituisce un punto le cui componenti x e y valgono 0 e la cui componente z coincide con quella z del punto fornito nella variabile Pu1.

(combine_Pu1_xy_Pu2_z Pu1 Pu2) → restituisce un punto le cui componenti x e y coincidono con quelle x e y del punto fornito nella variabile Pu1 e la cui componente z coincide con quella z del punto fornito nella variabile Pu2.

10) Funzioni sulle proiezioni. Le seguenti funzioni svolgono specifiche operazioni di proiezione intesa sempre quale proiezione ortogonale:

(proiez_1Pu_2Pu Pu1 Pu2 Pu3) → restituisce la proiezione del punto fornito nella variabile Pu1 sulla retta passante per i due punti forniti nelle variabili Pu2 e Pu3.

(proiez_1Pu_1Re Pu1 Re1) → restituisce la proiezione del punto fornito nella variabile Pu1 sulla retta fornita nella variabile Re1.

(proiez_1Pu_3Pu Pu1 Pu2 Pu3 Pu4) → restituisce la proiezione del punto fornito nella variabile Pu1 sul piano passante per i tre punti forniti nelle variabili Pu2, Pu3 e Pu4.

(proiez_2Pu_3Pu Pu1 Pu2 Pu3 Pu4 Pu5) → restituisce la proiezione dei punti forniti nelle variabili Pu1 e Pu2 sul piano passante per i tre punti forniti nelle variabili Pu3, Pu4 e Pu5.

(proiez_1Pu_1Pi Pu1 Pi1) → restituisce la proiezione del punto fornito nella variabile Pu1 sul piano fornito nella variabile Pi1.

(proiez_lPu_1Re lPu Re1) → restituisce la lista contenente la proiezione dei punti forniti nella variabile lPu sulla retta fornita nella variabile Re1.

(proiez_lPu_1Pi lPu Pi1) → restituisce la lista contenente la proiezione dei punti forniti nella variabile lPu sul piano fornito nella variabile Pi1.

(proiez_1Re_1Pi Re1 Pi1) → restituisce la proiezione della retta fornita nella variabile Re1 sul piano fornito nella variabile Pi1.

11) Funzioni sulle distanze. Le seguenti funzioni svolgono specifiche operazioni sulle distanze:

(distan_1Pu_1Pu Pu1 Pu2) → restituisce la distanza tra due punti forniti nelle variabili Pu1 e Pu2.

(distan_1Pu_1Re Pu1 Re1) → restituisce la distanza tra un punto ed una retta forniti rispettivamente nelle variabili Pu1 e Re1.

(distan_1Pu_1Pi Pu1 Pi1) → restituisce la distanza tra un punto ed un piano forniti rispettivamente nelle variabili Pu1 e Pi1.

(distan_1Re_1Re Re1 Re2) → restituisce la distanza tra due rette fornite nelle variabili Re1 e Re2. La procedura considera pari a 0 la distanza tra due rette coincidenti o incidenti in un punto proprio e calcola la distanza tra due rette parallele o la minima distanza tra due rette sghembe.

(distan_1Pi_1Pi Pi1 Pi2) → restituisce la distanza tra due piani forniti nelle variabili Pi1 e Pi2. La procedura considera pari a 0 la distanza tra due piani coincidenti o non paralleli e calcola la distanza tra due piani paralleli.

11) Funzioni sulle medie. Le seguenti funzioni svolgono specifiche operazioni di determinazione di entità poste in posizione intermedia rispetto alle entità fornite:

(Pu_medio_1Pu_1Pu Pu1 Pu2) → restituisce il punto medio tra due punti forniti nelle variabili Pu1 e Pu2.

(Re_media_1Re_1Re Re1 Re2) → restituisce la retta media tra due rette parallele fornite nelle variabili Re1 e Re2.

(Pi_medio_1Pi_1Pi Pi1 Pi2) → restituisce la piano medio tra due piani paralleli forniti nelle variabili Pi1 e Pi2.

11) Funzioni sulle intersezioni. Le seguenti funzioni svolgono specifiche operazioni di intersezione tra rette e piani forniti restituendo l'entità determinata, di volta in volta, quale intersezione:

(Pu_inters_1Re_1Pi Re1 Pi1) → restituisce il punto di intersezione tra una retta ed un piano forniti nelle variabili Re1 e Pi1. La procedura individua i casi particolari restituendo T (vero) in caso di appartenenza tra la retta e il piano e nil (falso) in caso di parallelismo tra la retta e il piano; nei casi restanti restituisce il punto di intersezione tra la retta e il piano.

(Re_inters_1Pi_1Pi Pi1 Pi2) → restituisce la retta di intersezione tra due piani forniti nelle variabili Pi1 e Pi2. La procedura individua i casi particolari restituendo T (vero) in caso di coincidenza tra i due piani e nil (falso) in caso di parallelismo tra i due piani; nei casi restanti restituisce la retta di intersezione tra i due piani.

(Pu_inters_1Pi_1Pi_1Pi Pi1 Pi2 Pi3) → restituisce il punto di intersezione tra tre piani forniti nelle variabili Pi1, Pi2 e Pi3.

(Pi_inters_1Re_1Re Re1 Re2) → restituisce il piano formato da due rette complanari fornite nelle variabili Re1 e Re2.

(Pu_inters_1Re_1Re Re1 Re2 OnPia) → restituisce il punto di intersezione tra due rette fornite nelle variabili Re1 e Re2. Se il valore fornito nella variabile OnPia è T l'intersezione è calcolata solo tra due rette incidenti non parallele, se il valore fornito è nil l'intersezione è calcolata anche tra due rette sghembe quale punto di minima distanza tra le due rette; se le due rette sono parallele restituisce il valore nil.

(Pu_inters_nRe lRe) → restituisce la migliore soluzione possibile per l'intersezione, calcolata tramite scarto quadratico medio, di un fascio proprio di rette complanari non convergenti esattamente in un unico punto.

11) Funzioni di trasformazione varie. Le seguenti funzioni svolgono specifiche operazioni di trasformazione, si segnalano in particolare quelle che di fatto fungono da definizione di retta e piano per tutte le procedure che ne fanno uso e che, a partire da due o tre punti forniti, definiscono i parametri atti a identificare in maniera univoca una retta ed un piano nel disegno:

(trasfo_2Pu_1Re Pu1 Pu2) → restituisce la retta passante dai due punti forniti nelle variabili Pu1 e Pu2. La retta è definita da una lista contenente: il versore parallelo alla retta data e passante per l'origine globale *Pu0* del sistema di riferimento; il punto di intersezione della retta con il piano ortogonale alla retta stessa passante per il punto *Pu0* in coordinate assolute e la sua conversione in coordinate orientate secondo la retta.

(trasfo_3Pu_1Pi Pu1 Pu2 Pu3) → restituisce il piano passante dai tre punti forniti nelle variabili Pu1, Pu2 e Pu3. Il piano è definito da una lista contenente: il versore normale al piano passante per l'origine globale *Pu0* del sistema di riferimento; il punto di intersezione del piano con la retta ortogonale al piano stesso passante per il punto *Pu0* in coordinate assolute e la sua conversione in coordinate orientate secondo il piano.

(trasfo_1Re_2Pu Re1) → restituisce una lista contenente due punti appartenenti alla retta fornita nella variabile Re1: il punto di intersezione della retta con il piano ortogonale alla retta stessa passante per il punto *Pu0* in coordinate assolute e un punto posto a distanza *inf* (pari a 1e+8) dal precedente e posto sulla retta.

(trasfo_1Pi_3Pu Pi1) → restituisce una lista contenente tre punti appartenenti al piano fornito nella variabile Pi1: il punto di intersezione del piano con la retta ortogonale al piano stesso passante per il punto *Pu0* in coordinate assolute, e una coppia di punti, rispettivamente in direzione orizzontale e verticale rispetto al piano, posti a distanza *inf* dal precedente punto e posti sul piano.

(trasfo_vett_vers Vet1) → restituisce il versore del vettore fornito nella variabile Vet1.

11) Funzioni get. 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 occupandosi al contempo di fornire indicazioni dettagliate all'operatore che deve fornire tali valori e verificando se necessario la correttezza dei valori forniti:

(get_opz §ini §txt §opz §def) → una procedura generale predisposta per essere riutilizzata da alcune delle procedure che seguono; le quattro stringhe di testo richieste dalla funzione come variabili in ingresso sono: l'inizializzazione della funzione get (§ini), un testo di supporto alla scelta che verrà richiesta all'operatore (§txt), un testo contenente le possibili opzioni disponibili e l'opzione di default (§opz), il testo indicante il valore di default (§def).

(get_yes_no §txt) → una procedura predisposta per ricevere una scelta tra si e no da parte dell'operatore che accetta come valore di default si; il testo fornito nella variabile §txt è di supporto alla scelta.

(get_no_yes §txt) → una procedura predisposta per ricevere una scelta tra si e no da parte dell'operatore che accetta come valore di default no; il testo fornito nella variabile §txt è di supporto alla scelta.

(get_point_princ Pu §txt) → una procedura generale predisposta per essere riutilizzata in alcune procedure che seguono e atta a ricevere la scelta di un punto in Autocad da parte dell'operatore e stamparne le coordinate; il punto eventualmente fornito nella variabile Pu funger da riferimento per la scelta del punto da parte dell'operatore, il testo fornito nella variabile §txt è di supporto alla scelta.

(GetPu) → una procedura atta a ricevere in Autocad un punto.

(GetRe) → una procedura atta a ricevere in Autocad due punti per identificare una retta. La procedura verifica che i due punti forniti non siano coincidenti.

(GetPi) → una procedura atta a ricevere in Autocad tre punti per identificare un piano. La procedura verifica che i tre punti forniti non siano allineati.

(GetEnt T1 T2 T3 T4) → una procedura atta a ricevere in Autocad fino a quatro entità del tipo punto retta o piano. I valori forniti nella quattro variabili T1 T2 T3 T4 specificano il tipo di entità che si vuole ricevere: se viene indicato il valore 1 verrà richiesto un punto, se viene indicato 2 verrà richiesta uan retta, se viene indicato 3 verrà richiesto un piano, se viene indicato 0 l'entità non verrà richiesta.

11) Comandi. I seguenti comandi rendono disponibili in Autocad, grazie all'interoperabilità con AutoLISP, funzionalità aggiuntive relative all'applicazione di alcuni criteri geometrici applicati a punti, rette e piani:

(c:pumed2pu) → Disegna il punto medio tra due punti forniti dall'operatore.

(c:remed2re) → Disegna la retta media tra due rette parallele fornite dall'operatore.

(c:pimed2pi) → Disegna il piano medio tra due piani paralleli forniti dall'operatore.

(c:puint2re) → Disegna il punto di intersezione tra due rette, fornite dall'operatore, secondo quanto determinato dalla specifica procedura di intersezione associata.

(c:piint2re) → Disegna il piano comune a due rette complanari, fornite dall'operatore, secondo quanto determinato dalla specifica procedura di intersezione associata.

(c:puintrepi) → Disegna il punto comune tra una retta e un piano, forniti dall'operatore, secondo quanto determinato dalla specifica procedura di intersezione associata.

(c:reint2pi) → Disegna la retta comune tra due piani, forniti dall'operatore, secondo quanto determinato dalla specifica procedura di intersezione associata.

(c:puint3pi) → Disegna il punto comune tra tre piani, forniti dall'operatore, secondo quanto determinato dalla specifica procedura di intersezione associata.

(c:pupropi) → Disegna il punto risultante dalla proiezione di un punto su di un piano, entrambi forniti dall'operatore, secondo quanto determinato dalla specifica procedura di proiezione associata.

(c:puprore) → Disegna il punto risultante dalla proiezione di un punto su di una retta, entrambi forniti dall'operatore, secondo quanto determinato dalla specifica procedura di proiezione associata.

(c:repropi) → Disegna la retta risultante dalla proiezione di una retta su di un piano, entrambi forniti dall'operatore, secondo quanto determinato dalla specifica procedura di proiezione associata.

(c:puintnre) → Disegna, aggiornandolo di volta in volta, il punto di intersezione, determinato tramite lo scarto quadratico medio, di un fascio proprio di segmenti complanari non convergenti esattamente in un unico punto. La procedura restituisce, per ogni nuovo segmento fornito, la miglior soluzione per il punto di intersezione.

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] Anzani Giovanni. 2016. *Algoritmi per l'ottimizzazione di estrazioni da nuvole di punti*. Lulu

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

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

[7] 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.)

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

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

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

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

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

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

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

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

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

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

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

[19] 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 alcuni precedenti saggi, 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.

```
;-------------------------------------------------------------------------------------------------------------------------------------------
;SETTAGGIO DI COSTANTI
;-------------------------------------------------------------------------------------------------------------------------------------------
(vl-load-com)                       ; per caricare le funzioni VL e VLAX
(setq   *0.0*       1e-8)           ; tolleranza da usare in vari casi di equal
(setq   *inf*       1e+8)           ; per punti di conversione di rette e piani
(setq   *Pu0*       '(0.0 0.0 0.0)) ; origine globale
;-------------------------------------------------------------------------------------------------------------------------------------------
;SETTAGGIO DI VARIABILI
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun BLI-0    ()  (setq bl_old    (getvar "blipmode"    ))  (setvar "blipmode"    0    ))
(defun PDS-6    ()  (setq ps_old    (getvar "pdsize"      ))  (setvar "pdsize"      -6   ))
(defun PDM-34   ()  (setq pm_old    (getvar "pdmode"      ))  (setvar "pdmode"      34   ))
(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 BLI-P    ()  (setvar "blipmode"    bl_old    ))
(defun PDS-P    ()  (setvar "pdsize"      ps_old    ))
(defun PDM-P    ()  (setvar "pdmode"      pm_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-0  ()  (if (P_ucsname "ucs_old")  (vl-cmdf "._ucs" "_na" "_s" "ucs_old"  "_y")    (vl-cmdf "._ucs" "_na" "_s" "ucs_old" )))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun UCS-P  ()  (if (P_ucsname "ucs_old")  (progn  (vl-cmdf "._ucs" "_na" "_r" "ucs_old" ) (vl-cmdf "._ucs" "_na" "_d" "ucs_old" ))))
;-------------------------------------------------------------------------------------------------------------------------------------------
(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)  (command "_redraw")  (princ)                           )
(defun VAR-P+enla   ()  (CMD-P)  (OSM-P)  (UCS-P)  (SPL-P)  (BLI-P)  (command "_redraw")  (princ)  (entlast)  )
;-------------------------------------------------------------------------------------------------------------------------------------------
;SETTAGGIO UCS
;-------------------------------------------------------------------------------------------------------------------------------------------
 (defun UCS_Pi   (Pi1)       (vl-cmdf "._ucs" "_za" (cadr Pi1)  (somma_vettoriale_2Ve (cadr Pi1) (car Pi1)))  )
;-------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI DI DISEGNO
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun Draw_1Pu  (Pu1)  (VAR-0)                                                                 (vl-cmdf "._point"   Pu1)          (VAR-P+enla))
(defun Draw_1Re  (Re1)  (VAR-0)  (mapcar 'set '(Pu1 Pu2) (trasfo_1Re_2Pu Re1))  (vl-cmdf "._xline"  Pu1 Pu2  "")    (VAR-P+enla))
(defun Draw_1Pi  (Pi1)  (VAR-0)  (UCS_Pi Pi1)                                  (vl-cmdf "._circle" (trans (cadr Pi1) 0 1) 1000)  (VAR-P+enla))
;-------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI SULLE LISTE
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun 1°   (lis)   (nth 0 lis))
(defun 2°   (lis)   (nth 1 lis))
(defun 3°   (lis)   (nth 2 lis))
(defun 4°   (lis)   (nth 3 lis))
;-------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;PREDICATI GENERICI
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(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      (lPu)   (apply  'and    (mapcar     'P_Pu   lPu)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;PREDICATI SULL'APPARTENENZA
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_coinc_1Pu_1Pu   (Pu1 Pu2)          (equal 0.0 (distance Pu1 Pu2) *0.0*))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_coinc_1Re_1Re  (Re1 Re2)       (and    (P_paral_1Re_1Re Re1 Re2)       (P_coinc_1Pu_1Pu    (cadr Re1)      (cadr Re2))))
(defun P_coinc_1Pi_1Pi  (Pi1 Pi2)       (and    (P_paral_1Pi_1Pi Pi1 Pi2)       (P_coinc_1Pu_1Pu    (cadr Pi1)      (cadr Pi2))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_allin_3Pu       (Pu1 Pu2 Pu3)      (P_coinc_1Pu_1Pu    Pu1    (proiez_1Pu_2Pu   Pu1   Pu2    Pu3)))
(defun P_compl_4Pu       (Pu1 Pu2 Pu3 Pu4)  (P_coinc_1Pu_1Pu    Pu1    (proiez_1Pu_3Pu   Pu1   Pu2    Pu3    Pu4)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_compl_1Re_1Re      (Re1    Re2    /    Pu1a    Pu1b    Pu2a    Pu2b)
    (mapcar     'set    '(Pu1a  Pu1b)       (trasfo_1Re_2Pu    Re1))
    (mapcar     'set    '(Pu2a  Pu2b)       (trasfo_1Re_2Pu    Re2))
    (P_compl_4Pu       Pu1a    Pu1b    Pu2a    Pu2b))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_appar_1Pu_1Re      (Pu1    Re1)       (P_coinc_1Pu_1Pu    Pu1    (proiez_1Pu_1Re   Pu1    Re1)))
(defun P_appar_1Pu_1Pi      (Pu1    Pi1)       (P_coinc_1Pu_1Pu    Pu1    (proiez_1Pu_1Pi   Pu1    Pi1)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_appar_1Re_1Pi      (Re1 Pi1)
    (and    (equal  0.0 (prodotto_scalare_2Ve     (car Re1)   (car Pi1))   *0.0*)
            (P_appar_1Pu_1Pi      (cadr Re1)      Pi1)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;PREDICATI SUL PARALLELISMO
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_paral_2Ver   (Ver1   Ver2)       (or     (equal  Ver1    Ver2                                *0.0*)
                                                  (equal  Ver1    (prodotto_1Sc_1Ve     -1   Ver2)   *0.0*)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_paral_1Re_1Re      (Re1 Re2)      (P_paral_2Ver      (car Re1)      (car Re2)))
(defun P_paral_1Re_1Pi      (Re1 Pi1)      (P_perpe_2Ver      (car Re1)      (car Pi1)))
(defun P_paral_1Pi_1Pi      (Pi1 Pi2)      (P_paral_2Ver      (car Pi1)      (car Pi2)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_paral_1Re_1Re_1Re      (Re1    Re2     Re3)
    (and    (P_paral_2Ver      (car Re1)       (car Re2))
            (P_paral_2Ver      (car Re1)       (car Re3))
            (P_paral_2Ver      (car Re2)       (car Re3))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_paral_1Pi_1Pi_1Pi      (Pi1    Pi2     Pi3)
    (and    (P_paral_2Ver      (car Pi1)       (car Pi2))
            (P_paral_2Ver      (car Pi1)       (car Pi3))
            (P_paral_2Ver      (car Pi2)       (car Pi3))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------
;PREDICATI SULLA PERPENDICOLARITA'
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun P_perpe_2Ver (Ver1 Ver2)
    (equal 0.0 (apply '+ (mapcar '* Ver1 Ver2)) *0.0*))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun P_perpe_1Re_1Re (Re1 Re2) (P_perpe_2Ver (car Re1) (car Re2)))
(defun P_perpe_1Re_1Pi (Re1 Pi1) (P_paral_2Ver (car Re1) (car Pi1)))
(defun P_perpe_1Pi_1P1 (Pi1 Pi2) (P_perpe_2Ver (car Pi1) (car Pi2)))
;-------------------------------------------------------------------------------------------------------------------------------------------
 (defun P_perpe_1Re_1Re_1Re (Re1 Re2 Re3)
    (and (P_perpe_2Ver (car Re1) (car Re2))
         (P_perpe_2Ver (car Re1) (car Re3))
         (P_perpe_2Ver (car Re2) (car Re3))))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun P_perpe_1Pi_1Pi_1Pi (Pi1 Pi2 Pi3)
    (and (P_perpe_2Ver (car Pi1) (car Pi2))
         (P_perpe_2Ver (car Pi1) (car Pi3))
         (P_perpe_2Ver (car Pi2) (car Pi3))))
;-------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI SUI VETTORI
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun modulo_2Pu (Pu1 Pu2) (distance Pu1 Pu2))
(defun vettore_2pu (Pu1 Pu2) (mapcar '- Pu2 Pu1))
(defun versore_2Pu (Pu1 Pu2) (mapcar '(lambda (x) ( / x (modulo_2Pu Pu1 Pu2)))
                                     (vettore_2pu Pu1 Pu2)))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun vettore_3Pu (Pu1 Pu2 Pu3 / x1 x2 x3 y1 y2 y3 z1 z2 z3)
    (mapcar 'set '(x1 y1 z1) Pu1)
    (mapcar 'set '(x2 y2 z2) Pu2)
    (mapcar 'set '(x3 y3 z3) Pu3)
    (list (- (* (- y2 y1) (- z3 z1)) (* (- y3 y1) (- z2 z1)))
          (- (* (- x3 x1) (- z2 z1)) (* (- x2 x1) (- z3 z1)))
          (- (* (- x2 x1) (- y3 y1)) (* (- x3 x1) (- y2 y1)))))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun versore_3Pu (Pu1 Pu2 Pu3 / vett_Pi1)
    (setq vett_Pi1 (vettore_3Pu Pu1 Pu2 Pu3))
    (mapcar '(lambda (x) ( / x (modulo_2Pu *Pu0* vett_Pi1))) vett_Pi1))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun prodotto_1Sc_1Ve (Sc1 Ve1) (mapcar '(lambda (x) (* x Sc1)) Ve1))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun prodotto_scalare_2Ve (Ve1 Ve2) (apply '+ (mapcar '* Ve1 Ve2)))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun prodotto_vettoriale_2Ve (Ve1 Ve2)
    (mapcar 'set '(i1 j1 k1) Ve1)
    (mapcar 'set '(i2 j2 k2) Ve2)
    (list (- (* j1 k2) (* k1 j2))
          (- (* k1 i2) (* i1 k2))
          (- (* i1 j2) (* j1 i2))))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun somma_vettoriale_2Ve (Ve1 Ve2) (mapcar '+ Ve1 Ve2))
(defun differenza_vettoriale_2Ve (Ve1 Ve2) (mapcar '- Ve1 Ve2))
;-------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI SUI PUNTI
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun flat_z_1Pu                (Pu1)          (mapcar   '*  Pu1   '(1 1 0)))
(defun flat_xy_1Pu               (Pu1)          (mapcar   '*  Pu1   '(0 0 1)))
(defun combine_Pu1_xy_Pu2_z      (Pu1   Pu2)    (list     (car    Pu1)   (cadr   Pu1)   (caddr  Pu2)))
;-------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI SULLE PROIEZIONI
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun proiez_1Pu_2Pu             (Pu1   Pu2    Pu3    /    vett_Pu2_Pu3)
    (setq    vett_Pu2_Pu3       (vettore_2pu   Pu2    Pu3))
    (trans   (combine_Pu1_xy_Pu2_z  (trans   Pu2    0   vett_Pu2_Pu3)
                                    (trans   Pu1    0   vett_Pu2_Pu3))    vett_Pu2_Pu3   0))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun proiez_1Pu_1Re     (Pu1   Re1    /    vers_Re1   orig_Re1   orig_re1T)
    (mapcar   'set    '(vers_Re1    orig_Re1   orig_re1T)    Re1)
    (trans   (combine_Pu1_xy_Pu2_z      orig_re1T      (trans   Pu1    0   vers_Re1))     vers_Re1   0))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun proiez_1Pu_3Pu     (Pu1   Pu2    Pu3    Pu4    /    vett_Pu2_Pu3_pu4)
    (setq    vett_Pu2_Pu3_Pu4     (vettore_3pu   Pu2    Pu3    Pu4))
    (trans   (combine_Pu1_xy_Pu2_z      (trans   Pu1    0   vett_Pu2_Pu3_Pu4)
                                        (trans   Pu2    0   vett_Pu2_Pu3_Pu4))   vett_Pu2_Pu3_Pu4      0))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun proiez_2Pu_3Pu     (Pu1   Pu2    Pu3    Pu4    Pu5)
    (list    (proiez_1Pu_3Pu   Pun1   Pun3   Pun4   Pun5)
             (proiez_1Pu_3Pu   Pun2   Pun3   Pun4   Pun5)))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun proiez_1Pu_1Pi     (Pu1   Pi1    /    vers_Pi1   orig_Pi1   orig_Pi1T)
    (mapcar   'set    '(vers_Pi1   orig_Pi1   orig_Pi1T)    Pi1)
    (trans   (combine_Pu1_xy_Pu2_z      (trans   Pu1    0   vers_Pi1)      orig_Pi1T)     vers_Pi1   0))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun proiez_lPu_1Re     (lPu   Re1)    (mapcar   '(lambda   (x)    (proiez_1Pu_1Re   x   Re1))   lPu))
(defun proiez_lPu_1Pi     (lPu   Pi1)    (mapcar   '(lambda   (x)    (proiez_1Pu_1Pi   x   Pi1))   lPu))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun proiez_1Re_1Pi     (Re1   Pi1    /    lPu    lPuP)
    (apply   'trasfo_2Pu_1Re    (proiez_lPu_1Pi    (trasfo_1Re_2Pu    Re1)    Pi1)))
;-------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI SULLE DISTANZE
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun distan_1Pu_1Pu     (Pu1 Pu2)      (distance   Pu1    Pu2))
(defun distan_1Pu_1Re     (Pu1 Re1)      (distan_1Pu_1Pu   Pu1    (proiez_1Pu_1Re   Pu1    Re1)))
(defun distan_1Pu_1Pi     (Pu1 Pi1)      (distan_1Pu_1Pu   Pu1    (proiez_1Pu_1Pi   Pu1    Pi1)))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun distan_1Re_1Re     (Re1   Re2    /    Pui)
    (cond        ((P_coinc_1Re_1Re    Re1    Re2)   0.0)
                 ((P_paral_1Re_1Re    Re1    Re2)   (distan_1Pu_1Pu   (cadr Re1)      (cadr Re2)))
                 ((P_compl_1Re_1Re    Re1    Re2)   0.0)
                 (T                                  (setq Pui            (Pu_inters_1Re_1Re    Re1    Re2    nil))
                                                     (distan_1Pu_1Pu      (proiez_1Pu_1Re   Pui      Re1)
                                                                          (proiez_1Pu_1Re   Pui      Re2)))))
;-------------------------------------------------------------------------------------------------------------------------------------------
(defun distan_1Pi_1Pi      (Pi1    Pi2)
    (cond   ((P_coinc_1Pi_1Pi     Pi1     Pi2)    0.0)
            ((P_paral_1Pi_1Pi     Pi1     Pi2)    (distan_1Pu_1Pu   (cadr Pi1)
                                                                    (cadr Pi2)))
            (T                                    0.0)))
;-------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI SULLE MEDIE
;-------------------------------------------------------------------------------------------------------------------------------------------------
(defun Pu_medio_1Pu_1Pu (Pu1 Pu2)
   (mapcar '(lambda (x) ( / x 2.0)) (mapcar '+ Pu1 Pu2)))
;-------------------------------------------------------------------------------------------------------------------------------------------------
(defun Re_media_1Re_1Re (Re1 Re2)
   (if (P_paral_1Re_1Re Re1 Re2) (list (car Re1)
                                       (Pu_medio_1Pu_1Pu (cadr Re1) (cadr Re2))
                                       (Pu_medio_1Pu_1Pu (caddr Re1) (caddr Re2)))))
;-------------------------------------------------------------------------------------------------------------------------------------------------
(defun Pi_medio_1Pi_1Pi (Pi1 Pi2)
   (if (P_paral_1Pi_1Pi Pi1 Pi2) (list (car Pi1)
                                       (Pu_medio_1Pu_1Pu (cadr Pi1) (cadr Pi2))
                                       (Pu_medio_1Pu_1Pu (caddr Pi1) (caddr Pi2)))))
;-------------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI SULLE INTERSEZIONI
;-------------------------------------------------------------------------------------------------------------------------------------------------
(defun Pu_inters_1Re_1Pi (Re1 Pi1 / numer denom)
   (setq numer (- (prodotto_scalare_2Ve (car Pi1) (cadr Pi1))
                  (prodotto_scalare_2Ve (car Pi1) (cadr Re1)))
         denom (prodotto_scalare_2Ve (car Pi1) (car Re1)))
   (cond ((P_appar_1Re_1Pi Re1 Pi1) T)
         ((P_paral_1Re_1Pi Re1 Pi1) nil)
         ((/= denom 0.0) (mapcar '+ (cadr Re1) (prodotto_1Sc_1Ve ( / numer denom) (carRe1))))))
;-------------------------------------------------------------------------------------------------------------------------------------------------
(defun Re_inters_1Pi_1Pi (Pi1 Pi2 / vers_Pi1 orig_Pi1 vers_Pi2 orig_Pi2 orig_Pi1_pro_Pi2 orig_Pi2_pro_Pi1
                                  orig_Re1 vers_Re1)
   (mapcar 'set '(vers_Pi1 orig_Pi1) Pi1)
   (mapcar 'set '(vers_Pi2 orig_Pi2) Pi2)
   (cond ((P_coinc_1Pi_1Pi Pi1 Pi2) T)
         ((P_paral_1Pi_1Pi Pi1 Pi2) nil)
         (T (setq orig_Pi1_pro_Pi2 (proiez_1Pu_1Pi orig_Pi1 Pi2)
                  orig_Pi2_pro_Pi1 (proiez_1Pu_1Pi orig_Pi2 Pi1))
            (setq orig_Re1 (cond ((equal orig_Pi1 orig_Pi2 *0.0*) orig_Pi1)
                                 ((equal orig_Pi1 *Pu0* *0.0*)
                                    (if (P_perpe_1Pi_1P1 Pi1 Pi2)
                                        orig_Pi2
                                        (inters orig_Pi1
                                                orig_Pi2_pro_Pi1
                                                orig_Pi2
                                                (proiez_1Pu_1Pi orig_Pi2_pro_Pi1 Pi2)
                                                nil)))
                                 ((equal orig_Pi2 *Pu0* *0.0*)
                                    (if (P_perpe_1Pi_1P1 Pi1 Pi2)
                                        orig_Pi1
                                        (inters orig_Pi1
                                                (proiez_1Pu_1Pi orig_Pi1_pro_Pi2 Pi1)
                                                orig_Pi2
                                                orig_Pi1_pro_Pi2
                                                nil)))
                                 (T
                                    (inters orig_Pi1 orig_Pi2_pro_Pi1
                                            orig_Pi2 orig_Pi1_pro_Pi2 nil))))
            (setq vers_Re1 (trasfo_vett_vers (prodotto_vettoriale_2Ve vers_Pi1 vers_Pi2)))
            (list vers_Re1 orig_Re1 (flat_z_1Pu (trans orig_Re1 0 vers_Re1))))))
;-------------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Pu_inters_1Pi_1Pi_1Pi      (Pi1    Pi2     Pi3     /    ReA    ReB)
    (setq    ReA    (Re_inters_1Pi_1Pi     Pi1     Pi2)
             ReB    (Re_inters_1Pi_1Pi     Pi1     Pi3))
    (if      (and    ReA    ReB)       (Pu_inters_1Re_1Re    ReA    ReB    T)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Pi_inters_1Re_1Re     (Re1    Re2     /    Pu1A   Pu1B   Pu2A   Pu2B)
    (mapcar    'set     '(Pu1A     Pu1B)     (trasfo_1Re_2Pu    Re1))
    (mapcar    'set     '(Pu2A     Pu2B)     (trasfo_1Re_2Pu    Re2))
    (cond      ((P_paral_1Re_1Re    Re1     Re2)      (trasfo_3Pu_1Pi    Pu1A   Pu1B   Pu2A))
               ((P_compl_1Re_1Re    Re1     Re2)      (trasfo_3Pu_1Pi    Pu1A   Pu1B   Pu2A))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------

;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Pu_inters_1Re_1Re    (Re1    Re2    OnPia  /    Pu1A   Pu1B   Pu2A   Pu2B   Pu1AT     Pu1BT     Pu2AT     Pu2BT
                                                    PuI_2D     PuI_3D     vers_Pi1     Pui_3D_flat)
  (mapcar   'set     '(Pu1A Pu1B)      (trasfo_1Re_2Pu Re1))
  (mapcar   'set     '(Pu2A Pu2B)      (trasfo_1Re_2Pu Re2))
  (cond     ((P_paral_1Re_1Re    Re1     Re2)    (setq    PuI_2D     nil
                                                    PuI_3D     nil))
            ((P_compl_1Re_1Re    Re1     Re2)    (setq    PuI_2D     (inters  Pu1A Pu1B  Pu2A Pu2B  nil)
                                                    PuI_3D     PuI_2D))
            (T                                   (setq    vers_Pi1        (trasfo_vett_vers (prodotto_vettoriale_2Ve (car Re1) (car Re2))))
                                                 (setq    Pu1AT           (trans   Pu1A   0    vers_Pi1)
                                                          Pu1BT           (trans   Pu1B   0    vers_Pi1)
                                                          Pu2AT           (trans   Pu2A   0    vers_Pi1)
                                                          Pu2BT           (trans   Pu2B   0    vers_Pi1))
                                                 (setq    PuI_3D_flat     (inters  (flat_z_1PuPu1AT)        (flat_z_1PuPu1BT)
                                                                                   (flat_z_1PuPu2AT)        (flat_z_1PuPu2BT)      nil))
                                                 (setq    PuI_2D          nil
                                                          PuI_3D          (trans   (combine_Pu1_xy_Pu2_z
                                                                                       PuI_3D_flat
                                                                                       (Pu_medio_1Pu_1Pu   Pu1AT      Pu2AT))
                                                                                   vers_Pi1    0))))
  (if    OnPia       PuI_2D     PuI_3D))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Pu_inters_nRe       (lRe    /    x1       x2       y1       y2       Ca       Cb       Cc       Saa      Sbb      Sab      Sac      Sbc)
    (setq    Saa     0       Sbb     0       Sab     0       Sac     0       Sbc     0)
    (foreach    Re      lRe
                (mapcar     'set     '(x1     y1)     (car     Re))
                (mapcar     'set     '(x2     y2)     (cadr    Re))
                (setq    Ca      (-   y1       y2)
                         Cb      (-   x2       x1)
                         Cc      (-   (*   x1   y2)     (*   x2   y1)))
                (setq    Saa     (+   Saa      (*   Ca       Ca))
                         Sab     (+   Sab      (*   Ca       Cb))
                         Sac     (+   Sac      (*   Ca       Cc))
                         Sbb     (+   Sbb      (*   Cb       Cb))
                         Sbc     (+   Sbc      (*   Cb       Cc))))
    (list    (/   (-   (*   Sab     Sbc)    (*   Sac      Sbb))   (-   (*   Saa      Sbb)    (*   Sab      Sab)))
             (/   (-   (*   Sab     Sac)    (*   Sbc      Saa))   (-   (*   Saa      Sbb)    (*   Sab      Sab)))
    0.0))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI DI TRASFORMAZIONE VARIE
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun trasfo_vett_vers      (Vet1)      (versore_2Pu    *Pu0*   Vet1))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun trasfo_2Pu_1Re        (Pu1    Pu2    /    vers_Re1   orig_Re1T      orig_Re1)
    (setq    vers_Re1       (versore_2Pu   Pu1    Pu2)
             orig_Re1T      (flat_z_1Pu    (trans  Pu1            0          vers_Re1))
             orig_Re1                      (trans  orig_Re1T      vers_Re1   0))
 (list   vers_Re1   orig_Re1   orig_Re1T))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun trasfo_3Pu_1Pi      (Pu1    Pu2    Pu3    /    vers_Pi1   orig_Pi1)
    (setq    vers_Pi1       (versore_3Pu       Pu1    Pu2    Pu3)
             orig_Pi1T      (flat_xy_1Pu       (trans  Pu1    0   vers_Pi1))
             orig_Pi1                          (trans  (flat_xy_1Pu   (trans   Pu10   vers_Pi1))      vers_Pi1    0))
    (list    vers_Pi1   orig_Pi1   orig_Pi1T))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun trasfo_1Re_2Pu        (Re1    /    vers_Re1   orig_Re1)
    (mapcar          'set    '(vers_Re1      orig_Re1   orig_re1T)      Re1)
    (list    orig_Re1
             (trans  (list    (car     orig_re1T)      (cadr    orig_re1T)          *inf*)       vers_Re1   0)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun trasfo_1Pi_3Pu      (Pi1    /    vers_Pi1   orig_Pi1   orig_Pi1T)
    (mapcar          'set    '(vers_Pi1  orig_Pi1   orig_Pi1T)      Pi1)
    (list    orig_Pi1
             (trans  (list    *inf*   0.0     (caddr      orig_Pi1T))    vers_Pi1   0)
             (trans  (list    0.0     *inf*   (caddr      orig_Pi1T))    vers_Pi1   0)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;FUNZIONI GET
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(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_point_princ     (Pu     §txt)        (initget 9)  (princ  (if  Pu  (getpoint   Pu      §txt)
                                                                                (getpoint            §txt))))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun GETPU               ()       (get_point_princ    nil       "\nPunto: "))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun GETRE     (   /    Pu1     Pu2)
    (setq    Pu1     *Pu0*
             Pu2     *Pu0*)
    (while       (P_coinc_1Pu_1Pu      Pu1     Pu2)
                 (setq    Pu1     (get_point_princ    nil       "\n1° punto della Retta: ")
                          Pu2     (get_point_princ    Pu1       "\n2° punto della Retta: "))
                 (princ   (if      (P_coinc_1Pu_1Pu      Pu1     Pu2)    "\nSono stati forniti due punti coincidenti riprovare" "\n")))
    (trasfo_2Pu_1Re    Pu1     Pu2))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun GETPI     (   /    Pu1     Pu2     Pu3     vers_Pi1   orig_Pi1)
    (setq    Pu1     *Pu0*           Pu2     *Pu0*           Pu3     *Pu0*)
    (while       (P_allin_3Pu    Pu1     Pu2     Pu3)
                 (setq    Pu1     (get_point_princ    nil       "\n1° punto del Piano: ")
                          Pu2     (get_point_princ    Pu1       "\n2° punto del Piano: ")
                          Pu3     (get_point_princ    Pu2       "\n3° punto del Piano: "))
                 (princ   (if      (P_allin_3Pu    Pu1     Pu2     Pu3)    "\nSono stati forniti tre punti allineati riprovare" "\n")))
    (trasfo_3Pu_1Pi    Pu1     Pu2     Pu3))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun GETENT (T1 T2 T3 T4 / ?1 opz en1 en2 en3 en4 eqd)
  (setq ?0 "\n"
        ?1 "\n-----------------------------------------------------\n"
        ?2 "non richiesta"
        ?3 " entità\n" )
  (if listent_old (progn (initget 0 "Si No")
                         (setq opz (getkword "Riutilizzare i dati già forniti? (Si o No) <No>: "))))
  (if (/= opz "Si")
      (progn (princ (strcat ?1 "Prima" ?3)) (setq en1 (cond ((= T1 0) (princ ?2))
                                                            ((= T1 1) (GETPU))
                                                            ((= T1 2) (GETRE))
                                                            ((= T1 3) (GETPI))))
             (princ (strcat ?1 "Seconda" ?3)) (setq en2 (cond ((= T2 0) (princ ?2))
                                                              ((= T2 1) (GETPU))
                                                              ((= T2 2) (GETRE))
                                                              ((= T2 3) (GETPI))))
             (princ (strcat ?1 "Terza" ?3)) (setq en3 (cond ((= T3 0) (princ ?2))
                                                            ((= T3 1) (GETPU))
                                                            ((= T3 2) (GETRE))
                                                            ((= T3 3) (GETPI))))
             (princ (strcat ?1 "Quarta" ?3)) (setq en4 (cond ((= T4 0) (princ ?2))
                                                             ((= T4 1) (GETPU))
                                                             ((= T4 2) (GETRE))
                                                             ((= T4 3) (GETPI))))
             (princ ?1)
             (setq listent_old (list en1 en2 en3 en4)))
      (progn listent_old)))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
;COMANDI
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun c:pumed2pu ( / le) (setq le (GetEnt 1 1 0 0))
                          (Draw_1Pu (Pu_medio_1Pu_1Pu (1° le) (2° le) )) (princ))
(defun c:remed2re ( / le) (setq le (GetEnt 2 2 0 0))
                          (Draw_1Re (Re_media_1Re_1Re (1° le) (2° le) )) (princ))
(defun c:pimed2pi ( / le) (setq le (GetEnt 3 3 0 0))
                          (Draw_1Pi (Pi_medio_1Pi_1Pi (1° le) (2° le) )) (princ))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun c:puint2re ( / le) (setq le (GetEnt 2 2 0 0))
                          (Draw_1Pu (Pu_inters_1Re_1Re (1° le) (2° le) nil )) (princ))
(defun c:piint2re ( / le) (setq le (GetEnt 2 2 0 0))
                          (Draw_1Pi (Pi_inters_1Re_1Re (1° le) (2° le) )) (princ))
(defun c:puintrepi ( / le) (setq le (GetEnt 2 3 0 0))
                           (Draw_1Pu (Pu_inters_1Re_1Pi (1° le) (2° le) )) (princ))
(defun c:reint2pi ( / le) (setq le (GetEnt 3 3 0 0))
                          (Draw_1Re (Re_inters_1Pi_1Pi (1° le) (2° le) )) (princ))
(defun c:puint3pi ( / le) (setq le (GetEnt 3 3 3 0))
                          (Draw_1Pu (Pu_inters_1Pi_1Pi_1Pi (1° le) (2° le) (3° le) )) (princ))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun c:pupropi ( / le) (setq le (GetEnt 1 3 0 0))
                         (Draw_1Pu (proiez_1Pu_1Pi (1° le) (2° le) )) (princ))
(defun c:puprore ( / le) (setq le (GetEnt 1 2 0 0))
                         (Draw_1Pu (proiez_1Pu_1Re (1° le) (2° le) )) (princ))
(defun c:repropi ( / le) (setq le (GetEnt 2 3 0 0))
                         (Draw_1Re (proiez_1Re_1Pi (1° le) (2° le) )) (princ))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(defun c:puintnre ( / v1 v2 v3 v4 BM0 PM0 PS0 lre PT1 PT2 PTi more)
  (CMD-0) (BLI-0) (PDS-6) (PDM-34)
  (setq more "Si")
  (while (equal "Si" more)
         (setq Ren (GETRE))
         (setq lre (append lre (list (trasfo_1Re_2Pu Ren))))
         (if (>= (length lre) 3) (VL-CMDF "_Erase" "_l" ""))
         (if (>= (length lre) 1) (Draw_1Re Ren))
         (if (>= (length lre) 2) (progn (setq PTi (Pu_inters_nRe lre))
                                        (Draw_1Pu PTi)
                                        (princ "\nPunto di intersezione: ") (princ PTi)
                                        (setq more (get_yes_no "continuare?")))))
  (PDM-P) (PDS-P) (BLI-P) (CMD-P)
  (princ))
;-------------------------------------------------------------------------------------------------------------------------------------------------------
(alert "Caricato fino a EOF")
;-------------------------------------------------------------------------------------------------------------------------------------------------------
```