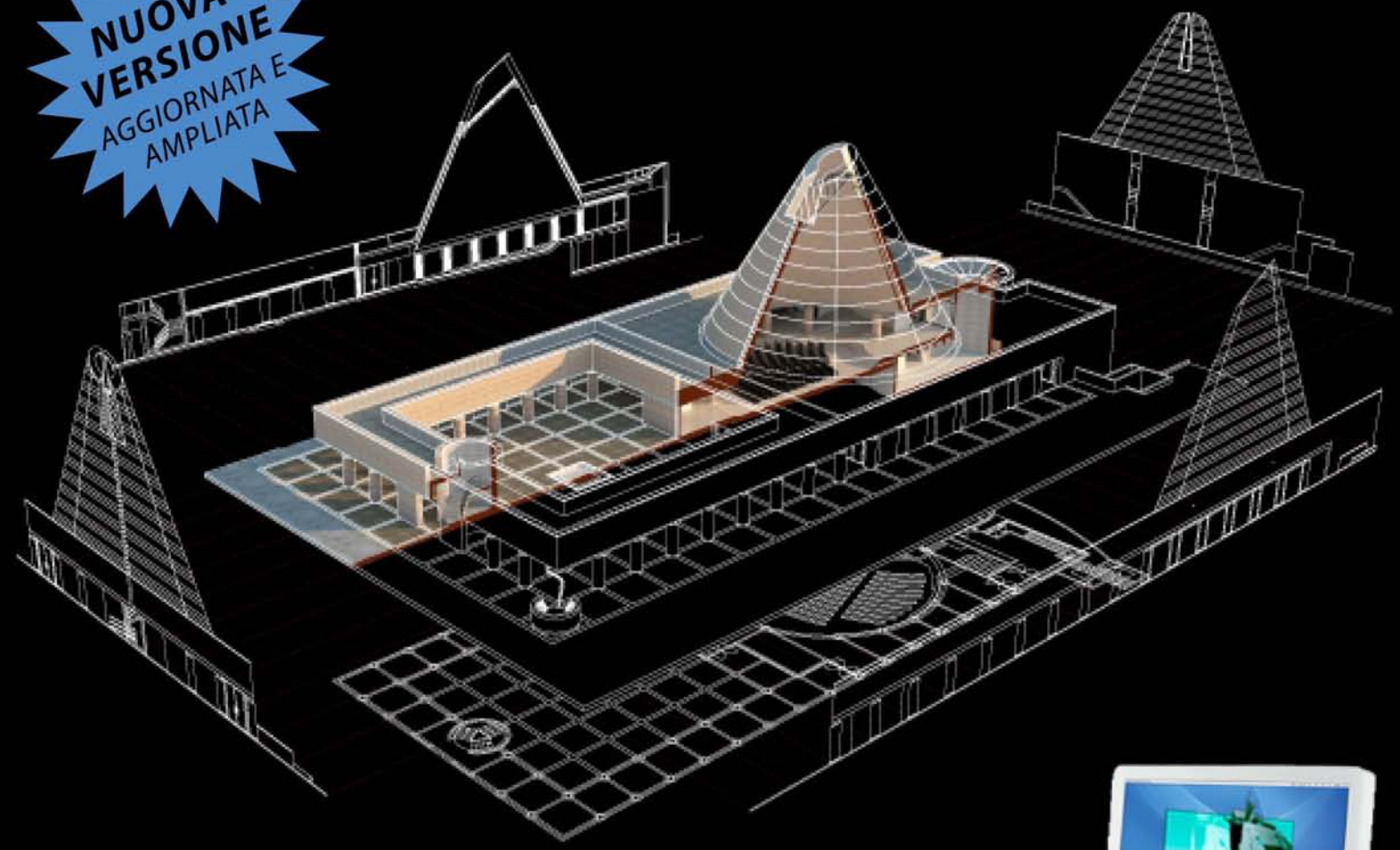
www.archiradar.it

introduzione al GDL

geometric description language

a cura di Roberto Corona

NUOVA VERSIONE AGGIORNATA E AMPLIATA



10 lezioni a difficoltà progressiva

semplici esercizi correlati

aggiornato con i parametri di ArchiCAD 9

appendice con tutte le parole chiave del GDL

ArchiCAD 9 - Guida all'uso

Autori:
Roberta Cecchi - Roberto Corona -
Daniele Raggi - Pietro Spampatti

Editore:
FAG Edizioni - Milano

Uscita:
Settembre 2005

Pagine:
432

Prezzo:
22,00 Euro

Aquistabile in tutte le librerie d'Italia e online presso il sito dell'editore:
www.fag.it

Roberto Corona
Geometra libero professionista, appassionato di informatica, membro del GDL Alliance, traduttore e scrittore di manuali informatici.

Introduzione al GDL geometric desciption language
Autore: Roberto Corona
Prima edizione

immagini in copertina a cura di Ismaele de Rosa

INTRO

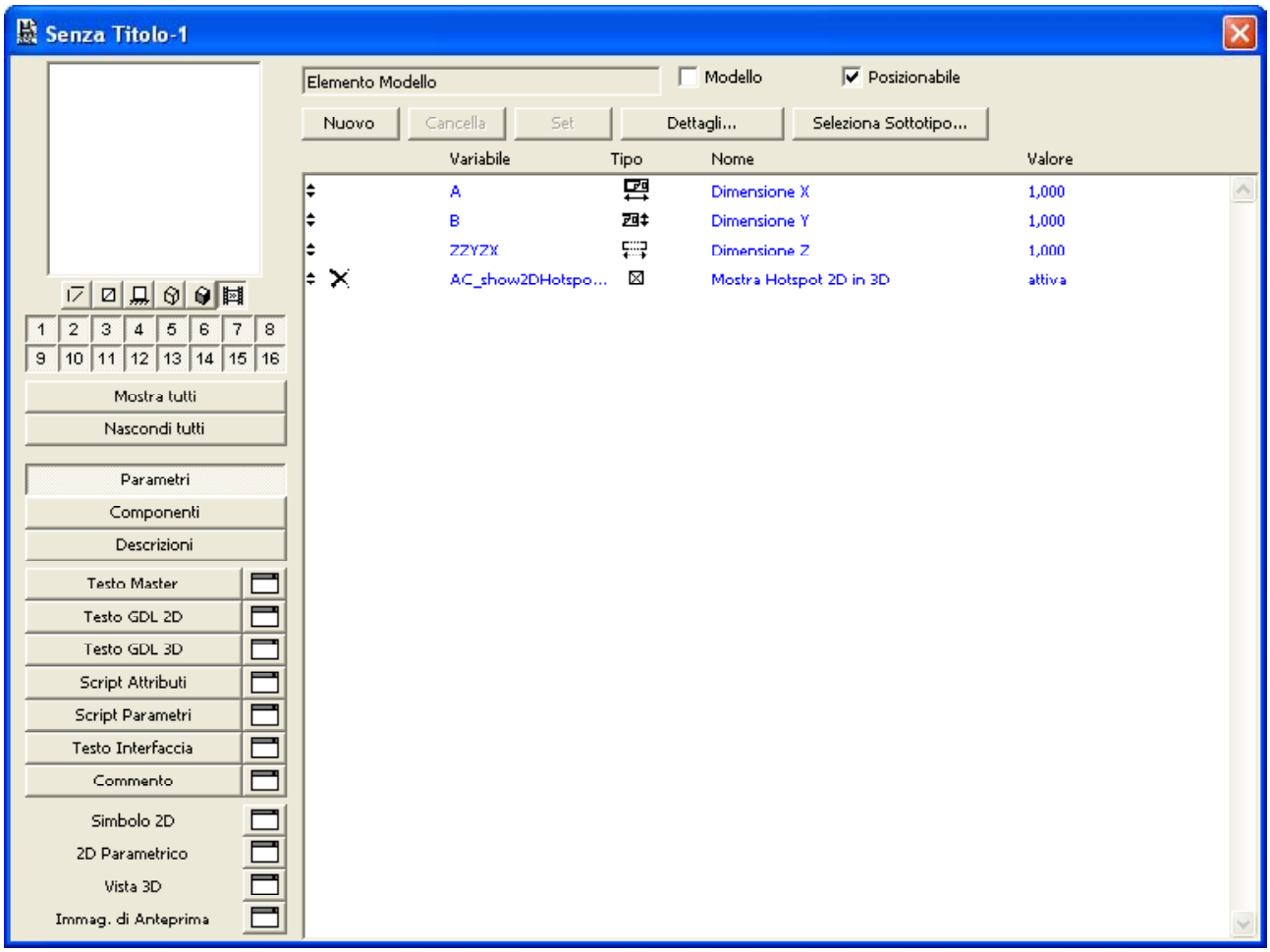
Queste Lezioni sono rivolte a chi desidera iniziare ad avvicinarsi al GDL (*Geometrical Description Language*), il linguaggio di descrizione geometrica sviluppato da Graphisoft e che sta alla base di ArchiCAD fin dalla sua nascita. Viene utilizzato per costruire elementi di libreria di vario tipo, dai più semplici simboli bidimensoinali alle forme spaziali più complesse.
Se ne consiglia la lettura solo in caso si sappia usare sufficientemente ArchiCAD, conoscendone cioè tecniche e terminologia, e si abbia dimestichezza nella gestione delle librerie del programma.
L'ambiente di sviluppo del GDL è parte integrante di ArchiCAD. Ogni elemento di libreria può comprendere uno o più testi (detti anche Script, composti da comandi GDL), che generano l'oggetto solido ed il suo simbolo planimetrico. Quest'ultimo può essere creato anche in modo grafico, disegnandolo con i normali strumenti 2D o, ancora, essere generato automaticamente dalla proiezione del solido 3D.
Per quanto complessi, la maggior parte degli oggetti possono essere scomposti in forme geometriche semplici. Analizzato e scomposto l'oggetto da realizzare, sarà possibile tradurre i suoi blocchi in linguaggio GDL. Alla base ovviamente deve esserci una buona percezione dello Spazio e una conoscenza di base della geometria descrittiva.
Il consiglio per chi si avvicina solo adesso a questo linguaggio di programmazione, è di non iniziare ad utilizzarlo avendo in mente oggetti troppo complessi. È importante procedere per gradi, sperimentando passo dopo passo tutte le tecniche e i comandi partendo da oggetti semplicissimi. Chi ha familiarità con qualsiasi linguaggio di programmazione come il BASIC o il Pascal non avrà nessuna difficoltà ad iniziare a programmare in GDL, ma chiunque abbia la curiosità ed il desiderio di esplorarlo imparerà presto ad utilizzarlo con profitto. Questo è appunto lo scopo delle dieci lezioni che vi presentiamo: dare veramente a TUTTI almeno una possibilità di PROVARE realmente a lavorare col GDL e far rientrare anche questo nell'arsenale dei propri strumenti di lavoro.
Sia chiaro che questo non può essere un corso completo di programmazione in GDL. Servirà però a prendere confidenza con il linguaggio, a capire i princìpi che ne regolano il funzionamento e permetterà di superare quei timori che caratterizzano i primi approcci del neofita.
Prima di iniziare la prima lezione di GDL facciamo le nostre premesse:

1. La versione di riferimento è ArchiCAD 9.0
2. Sapete già usare sufficientemente il programma... conoscete cioè tecniche e terminologia.
3. Avete dimestichezza nella gestione delle librerie.
4. Siete INTERESSATI ad imparare questo linguaggio di programmazione.
5. Avete il manuale GDL. Nella cartella Documentazione Graphisoft trovate la versione in PDF, mentre nell'Aiuto ArchiCAD c'è la versione consultabile con il Browswe Internet.

LEZIONE 01

01.01. Facciamo conoscenza con l'interfaccia

Cominciamo dall'inizio...
Quando **usiamo** un oggetto scegliamo l'icona corrispondente dalla Palette Strumenti. Per accedere alle impostazioni: doppio click (o freccia-sinistra).
Quella che si apre è la finestra dei settaggi. È una normale finestra di dialogo a pannelli, che ospita un gran numero di controlli. È piena di roba, ma è l'unica finestra con cui ha a che fare **l'utente** dell'oggetto.
Molto diversa è invece la vita del **creatore di oggetti**, che potrebbe avere a che fare anche con una dozzina di finestre diverse, per un solo elemento di libreria.
Spaventati? Be' non tutte devono essere aperte... Io per esempio non uso la parte relativa ai computi, e quelle finestre non le conosco nemmeno... figuriamoci insegnare a usarle...
Comunque procedete insieme a me.
Dal menu **Archivio** scegliete il comando **Oggetti GDL > Nuovo Oggetto**.
Compare la finestra principale di gestione dell'oggetto GDL.



Nella parte sinistra, sotto la sezione dell'anteprima, vediamo una serie di pulsanti.
Notiamo subito che per alcune voci è disponibile solo un pulsante col testo descrittivo, per altre solo un pulsante quadrato (a destra del testo) con l'icona di

una finestra; per altre ancora sono disponibili entrambe le opzioni. Il pulsante con il testo, se presente, permette di mostrare l'argomento relativo nell'area destra della finestra principale; il pulsante con la finestra aprirà invece una finestra indipendente.

Vediamo quali sono le voci disponibili (tenendo conto che "Mostra tutti" e "Nascondi tutti" si riferiscono solo ai pulsantini superiori):

Parametri
Conterrà l'elenco delle variabili accessibili all'utente. All'inizio sono presenti solo le variabili **A** e **B**, che rappresentano le dimensioni X e Y dell'oggetto.

Componenti
Usato per l'eventuale associazione dell'oggetto con elementi di un database esterno —*Si tratta di una parte relativa ai computi, che non tratteremo in queste lezioni*—

Descrizioni
Analogo al precedente, ma non legato ad un database esterno — *Si tratta di una parte relativa ai computi, che non tratteremo in queste lezioni* —

Testo Master
È un testo in comune tra Script 2D e Script 3D. Per ora non lo usiamo, ma potremo tornarci su in seguito.

Testo GDL 2D
E' una descrizione della geometria piana... Un altro modo offerto per rappresentare il simbolo in pianta dell'oggetto. Qui si deve usare il GDL. Ovviamente è meno semplice, ma permette di fare cose molto più articolate. Questa finestra si usa, normalmente, in alternativa alla precedente e ha la prevalenza su quella, se sono state utilizzate entrambe.

Testo GDL 3D
È una descrizione della geometria solida... Qui andremo a scrivere tante cose interessanti e misteriose usando le affascinanti istruzioni del GDL!

Script Attributi
Usato per l'eventuale associazione di descrizioni analitiche, per il computo dell'oggetto —*Si tratta di una parte relativa ai computi, che non tratteremo in queste lezioni*—

Script Parametri
Vedremo meglio in seguito, ma qui si possono definire delle "liste di scelta". Faccio solo un esempio: avete notato che quando tra i parametri c'è da scegliere un materiale appare un menu a comparsa con tutti i materiali, nel quale effettuare la selezione. Noi possiamo costruire dei piccoli menu personali con valori o testi. Inoltre qui potremo effettuare altre operazioni sui parametri: assegnargli un valore, o bloccarli per prevenirne la modifica.

Testo Interfaccia
Il GDL permette anche di realizzare una o più schermate personalizzate, per intervenire sui parametri e per esporre testi e valori, in alternativa alla normale lista di parametri disponibile nella finestra settaggi dell'elemento di libreria.

Commento
Usata pochissimo, purtroppo. Vi si può inserire un breve testo di commento o esplicativo. Comparirà nell'area di anteprima se l'utente seleziona la relativa iconcina nella finestra Settaggi Oggetto.

Simbolo 2D
È una finestra di grafica, simile a quella 2D del progetto. Possiamo usarla per

disegnare il simbolo dell'oggetto. Sono attivi solo gli strumenti Linea, Cerchio, Spline, Testo, Retino, Figura e Hotspot.

2D Parametrico

È la finestra che presenta il risultato del Testo GDL 2D. È una finestra di grafica nella quale non si può disegnare.

Vista 3D

È la finestra che presenta il risultato del Testo GDL 3D. È una finestra di grafica nella quale non si può disegnare ed è del tutto analoga alla finestra 3D del progetto.

Immagine Anteprima

In questa piccola finestra si può incollare una vista (in genere un render) dell'oggetto. L'immagine, se più grande, viene scalata per rientrare nelle dimensioni massime di 128x128 pixel. L'utente la può vedere se seleziona la relativa iconcina nella finestra Settaggi Oggetto.

01.02. Un po' di teoria

Il GDL è un vero e proprio linguaggio di programmazione. Quindi deve essere affrontato sapendo che...

- deve essere studiato (ha una sua sintassi, che non può essere ignorata).
- richiede un po' di disciplina e molta attenzione (nei primi tempi riceverete molti messaggi d'errore).
- dà grande soddisfazione, quando funziona (e alla fine funziona sempre).

Alla base di tutto c'è il sistema cartesiano. Presumo che tutti sappiate di cosa parlo: un'origine e tre assi ortogonali tra loro che definiscono tre direzioni nello spazio: X, Y e Z.

Nello spazio cartesiano possiamo definire dei solidi usando le istruzioni del GDL. Cominciamo a vedere la più semplice: **BLOCK**. Come avrete intuito realizza un 'blocco' cioè un parallelepipedo.

La sintassi di questo comando è

```
BLOCK p1, p2, p3
```

Analizziamola:

BLOCK - questa è una PAROLA CHIAVE. Tutte le istruzioni GDL cominciano con una parola chiave.

p1, p2, p3 - questi sono i parametri. Quasi tutte le istruzioni GDL comprendono dei parametri. Il significato specifico dei singoli parametri, per le istruzioni più semplici, si imparerà a memoria, per altre meglio non sforzarsi: si ricorrerà quasi sempre al manuale o all'help in linea (se l'avete installato). Nella fattispecie i tre parametri, l'avrete intuito, rappresentano lunghezza, larghezza e altezza. E sono in questo esatto ordine: **p1** rappresenta la dimensione lungo l'asse X, **p2** lungo l'asse Y e **p3** (indovinato?) lungo l'asse Z.

Vediamo più a fondo come si scrivono i comandi:

Chiariamo innanzi tutto che la parola chiave deve essere copiata tale e quale, mentre i parametri "rappresentano" dei valori. Nel testo GDL di un oggetto non scriveremo quindi le parole **p1**, **p2** e **p3**, ma dei valori numerici, o delle espressioni, o delle variabili (vedremo in seguito di cosa si tratta).

- Maiuscolo e minuscolo non sono rilevanti. Ma abituatevi ad utilizzare un metodo costante. Il metodo suggerito è: maiuscolo per i comandi (parole chiave), minuscolo tutto il resto.

- Dopo la parola chiave deve esserci uno spazio (o meglio: *almeno* uno spazio).
- I parametri sono separati tra loro da virgole. Gli spazi, dopo le virgole, aiutano a leggere (ed eventualmente correggere) le istruzioni, ma non sono indispensabili.
- Tra la parola chiave e il primo parametro NON deve esserci la virgola.
- Non è ammessa neanche una virgola dopo l'ultimo parametro.

N.B.: Si chiamano **parametri** sia i valori che seguono la Parola Chiave (più precisamente: Parametri del comando), sia l'elenco di elementi presente nella finestra principale dell'oggetto, e che permettono all'utente di interagire con esso, rendendolo, appunto, parametrico (questi sono i Parametri dell'Oggetto).

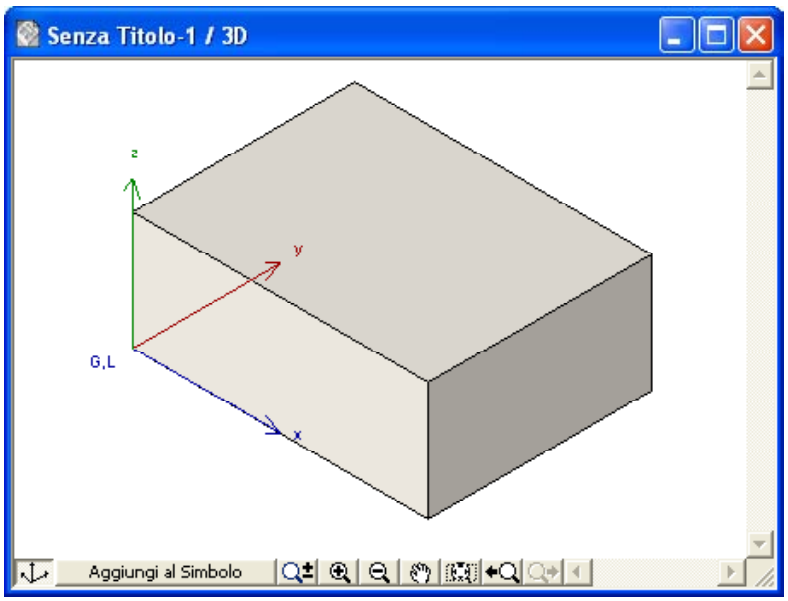
01.03. Un po' di pratica

Attivate la finestra **Testo GDL 3D**, cliccando sul relativo pulsante.
Scriviamo un'istruzione BLOCK per realizzare un blocco lungo due metri, largo un metro e mezzo e alto 80 cm.

```
BLOCK 2.00, 1.50, 0.80
```

Avete notato? Probabilmente il vostro computer è impostato (pannello di controllo ecc.) per riconoscere la virgola come separatore per i decimali, ma nel testo GDL dovete usare comunque il punto. La virgola serve esclusivamente per separare i vari elementi di un comando.
Attivate la finestra **Vista 3D** dell'oggetto (non quella del progetto).
Eccolo! C'è proprio un prisma. E' bruttino, ma è tutto nostro...
Nell'angolo inferiore sinistro della finestra 3D c'è il pulsante Mostra/Nascondi assi cartesiani.

Se non sono visibili, cliccate per mostrare gli assi. Nella finestra verrà mostrata l'origine dell'oggetto da cui si dipartono tre frecce colorate che indicano le tre direzioni X, Y e Z.
Noterete che il nostro oggetto ha un vertice sull'origine: questa è una delle caratteristiche del comando BLOCK. Altri comandi offrono maggiore libertà di posizionamento, ma ovviamente non sono altrettanto semplici da padroneggiare.



Alcuni comandi hanno tanti parametri. Non vi spaventate se dico che possono averne diverse decine (pensate a un solaio: è un semplice prisma, ma può avere molti vertici, e per ognuno devono essere definite le coordinate X e Y). Quindi è essenziale capire subito come scrivere un comando su più righe. È consentito andare a capo, senza che per questo il comando risulti interrotto, rispettando una semplice condizione:
Ogni riga, tranne l'ultima, deve terminare con la virgola, quella che fa da separatore tra i parametri.
Ovvero, detto in altri termini, dopo ogni parametro (e la sua virgola) può essere inserito un ritorno a capo. Anche gli spazi possono essere aggiunti a volontà, per rendere il codice più leggibile.

01.04. Ed ecco a voi... le variabili

Il nostro *Blocco* ha l'aspetto un po' statico...
Impostate come tipo di anteprima "Vista 3D" e scegliete quella a rimozione linee, o ombreggia. Ora provate a modificare i valori delle variabili **A** (Dimensione X), **B** (Dimensione Y) e **ZZYZX** (dimensione Z) sulla finestra principale dell'oggetto (usate il pulsante Parametri, se necessario).
Fatto?
Non succede niente!
Perché?
È evidente: nello script GDL non abbiamo mai fatto riferimento a queste variabili. Il nostro *Blocco* è indipendente (anarchico direi) e non va bene. Noi vogliamo avere il controllo assoluto... almeno su questo piccolo Blocco!
Editiamo quindi lo script.
Cancellate il numero 2.00 e scrivete **A** (proprio la lettera "A"), al posto di 1.50 scrivete **B** e al posto di 0.80 scrivete **ZZYZX**.

```
BLOCK a, b, zzyzx
```

Torniamo a modificare i valori delle variabili, nella finestra dei parametri, e vedremo che l'anteprima cambia.
Cosa è successo?
A, **B** e **ZZYZX** sono i nomi di due variabili, cioè, in un certo senso, contenitori di valori. Potrei dire *"Roberto è il mio nome, ma io non sono Roberto"* (io sono una persona fisica, il mio nome no). Roberto è solo una parola usata per rappresentarmi. Allo stesso modo **A** rappresenta la quantità numerica inserita nella casella "valore". Nello script, quando l'interprete GDL trova una variabile (la nostra lettera **A**) le sostituisce il valore corrispondente. Questo comportamento permette di dare un primo livello di flessibilità ai linguaggi di programmazione. Il valore di una variabile può essere modificato in qualunque momento (non a caso si chiama ... variabile) e il programma si comporta di conseguenza aggiornando in pratica il comando senza che si debba riscriverlo.

01.05. Ancora variabili

Giusto per provare, vediamo come fare se non volessimo usare le variabili predefinite. Ammettiamo che per una misura, ad esempio per l'altezza, voglio usare un'altra variabile. Niente di più facile: come al solito basta un click (ovviamente sul pulsante "Nuovo"). Apparirà, come per magia, una nuova riga. Nella colonna Variabile c'è un nome di default. Avventuriamoci subito in questo campo. Cancelliamo quello che ha messo ArchiCAD e mettiamoci la scritta **alt**. Cosa

abbiamo fatto? Abbiamo dato un nome alla variabile appena creata. E le abbiamo dato un nome mnemonico: a noi serve per l'**alt**ezza.
Nella casella "Variabile" abbiamo inserito il nome, ora nella casella "Nome" inseriamo una descrizione, e nella casella "Valore" l'altezza del Blocco.
"Tipo" è un menu pop-up che permette di scegliere che tipo di valori può accettare la nostra variabile (per ora non dobbiamo modificarlo). I più attenti avranno notato che la nostra variabile non è uguale alle altre. La "nostra" è nera, quelle predefinite sono blu (**A**, **B, ZZYZX e AC_show2DHotspotsIn3D** per gli Oggetti, più numerose per Porte, Finestre, Lampade, Zone ed altri tipi). Alle nostre variabili possiamo modificare il nome e altri attributi, a quelle predefinite no.
Apportiamo la necessaria modifica allo script, che ora sarà (non dovrei dirlo):

```
BLOCK a, b, alt
```

Proviamo a modificare i valori di tutti i parametri... Funziona (obbedisce).

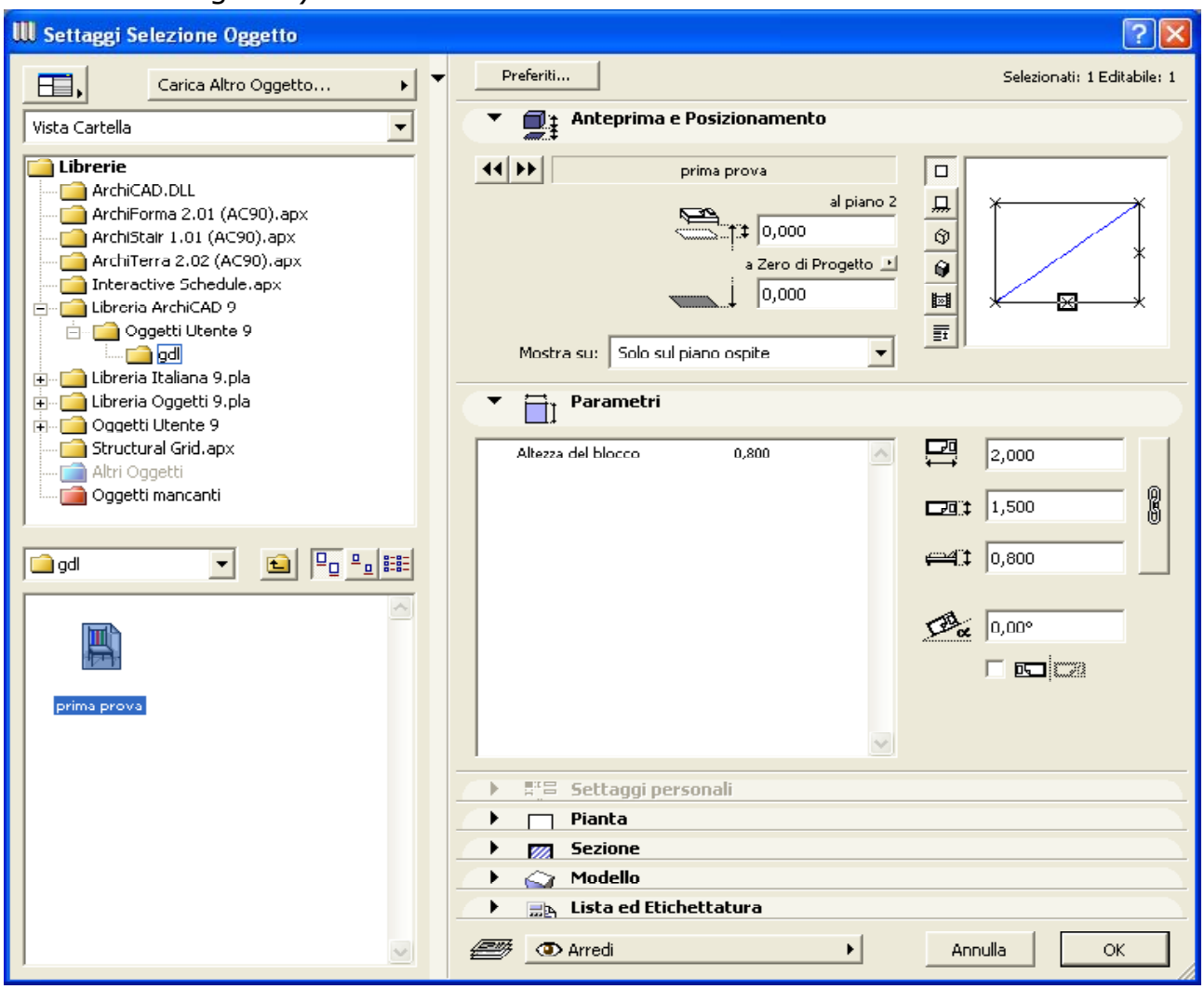
01.06. Toccata e fuga nel 2D

Come abbiamo visto, per un singolo oggetto ci sono varie finestre nelle quali scrivere istruzioni geometriche GDL: tra le altre una per il 3D e una per il 2D. Può apparire strano, ma sono due cose ben distinte (o almeno vengono trattate dal linguaggio in maniera distinta). Abbiamo quindi fatto solo metà del lavoro. Per questo primissimo oggetto NON utilizzeremo uno script 2D per generare il simbolo bidimensionale. Useremo un metodo "vecchio", ma in questo caso adeguato. Passate alla finestra 3D dell'oggetto. Potete usare anche il menu Finestre di ArchiCAD, dove è presente ora una nuova voce "Senza titolo" (il nostro oggetto) ed un sottomenu che riporta tutte le sue finestre.
Dal menu **Modello > Settaggi Proiezione 3D** impostate una vista assonometrica in Pianta dall'alto, con azimut telecamera a 270°. Ora guardate in basso, sulla cornice della finestra 3D dell'oggetto... dopo l'icona Mostra/Nascondi assi, si può leggere "Aggiungi al Simbolo". Fate click su questa scritta, e nella finestra **Simbolo 2D** verrà tracciato un bel rettangolo. Possiamo lavorare a piacere in questa finestra, con gli strumenti bidimensionali di ArchiCAD. Aggiungete una diagonale, per arricchire un po' il simbolo (una sola, che altrimenti si arricchisce troppo...!).
Adesso mettete un Hotspot in ogni angolo, uno al centro della linea di destra e un altro al centro della linea in alto.
Bene.

01.07. La montagna ha partorito il topolino

Andiamo nel menu **Archivio** e scegliamo il comando Registra. Funziona esattamente come per il progetto. Se è attiva la vista 3D dell'oggetto salveremo solo un'immagine, in tutti gli altri casi si salverà l'oggetto. Assegniamogli il nome "Prima prova" e salviamo in una cartella DESTINATA AI NOSTRI OGGETTI. Infatti usare le cartelle della libreria standard è una pessima idea... Reinstallando il programma (o facendo un aggiornamento) la libreria vecchia viene sostituita da quella nuova e i nostri sudati oggetti svanirebbero nel nulla (inutile cercarli nel cestino).
Fatto?
Ora, dalla finestra di pianta, attivate lo strumento **Oggetto**. ArchiCAD si è già posizionato sull'oggetto "Prima prova", dimostrando ancora una volta la sua intelligenza (ogni volta che modifichiamo o creiamo un oggetto lui capisce che ne

avevamo bisogno...).



Osserviamo la finestra dei settaggi e facciamo alcune considerazioni.

1. I nostri parametri vanno nella lista dei parametri; quelli predefiniti (A, B, ZZYZX) hanno una loro collocazione autonoma.
2. I valori di default che l'oggetto ci presenta sono quelli che erano assegnati ai parametri, al momento del salvataggio.
3. Ogni Hotspot posizionato sul simbolo è diventato una "maniglia". Possiamo cliccarci per utilizzarlo come punto di inserimento dell'oggetto. Più precisamente il PRIMO Hotspot piazzato sarà il punto di inserimento di default.
4. Gli stessi Hotspot saranno punti di selezione dell'oggetto, una volta posizionato.

Inseriamolo in pianta.
Funziona perfettamente. Notate che l'aver messo gli Hotspot sui punti estremi consente di *stretchare* l'oggetto: i due punti sui lati ci consentono di striarlo in lunghezza **o** in larghezza; quelli negli angoli consentono il ridimensionamento contemporaneo di lunghezza e larghezza.

LEZIONE 02

02.01. Nel giardino dell'Eden

Abbiamo creato *Blocco*. Ora gli daremo una compagna. Aggiungiamo quindi un'altra istruzione BLOCK al nostro Testo GDL 3D:

```
BLOCK a, b, zzyzx
BLOCK a, b, zzyzx
```

e verifichiamo il risultato.

Nella finestra 3D dell'oggetto, come era facile aspettarsi, c'è solo il nostro vecchio blocco, o almeno così sembra. In effetti abbiamo definito due entità identiche e sovrapposte. Introduciamo quindi un nuovo concetto fondamentale: **le espressioni**.

Così come ogni variabile viene sostituita "al volo" dal corrispondente valore, anche ogni espressione matematica viene immediatamente valutata e interpretata. Per qualsiasi parametro possiamo utilizzare, oltre ai valori numerici (es. **5.30**), le variabili (es. **alt**) o espressioni matematiche (es. **3 + 2.30**). A sua volta un'espressione può contenere, come elementi, sia numeri che variabili (es. **zzyzx + 2.30**)

02.02. "Mi faccia un'espressione intelligente"

Questi sono i principali operatori matematici:

^	elevamento a potenza	priorità 1	es.: 5 ^ 2 = 25
*****	moltiplicazione	priorità 2	es.: 5 * 2 = 10
/	divisione	priorità 2	es.: 5 / 2 = 2.5
MOD	resto della divisione	priorità 2	es.: 5 MOD 2 = 1
+	somma	priorità 3	es.: 5 + 2 = 7
-	sottrazione	priorità 3	es.: 5 - 2 = 3

Le operazioni vengono eseguite in ordine di priorità (da sinistra a destra, a parità di priorità).

Es.: 3*3+5*5 = 34

La priorità può essere modificata con l'uso delle parentesi.

Es.: 3*(3+5*5) = 84

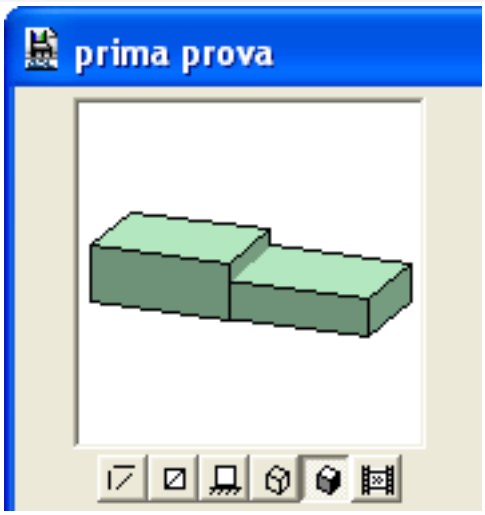
Ma torniamo alle nostre creature. Decidiamo che *Blocca* (!) deve essere un po' più piccola e grassa (!!). Precisamente alta i due terzi di *Blocco*, larga 10 cm più di lui e lunga il doppio.

Il primo parametro del comando BLOCK è la lunghezza. Perché il nuovo elemento sia il doppio di Blocco, dovremo scrivere **a*2**. Per la larghezza: **b+0.10**, e per l'altezza **zzyzx*2/3**.

Quindi ecco il nostro script 3D:

```
BLOCK a, b, zzyzx
BLOCK a*2, b+0.10, zzyzx*2/3
```

Verifichiamo il risultato.



Ora siamo in grado di distinguere Blocco e Blocca. Modifichiamo il valore dei parametri a, b e zzyzx e li vedremo muoversi... ma sono sempre uno sull'altra... ehm. Meglio separarli, prima che nascano i blocchettini!.
Farei subito una considerazione sulle buone norme di programmazione: il nostro oggetto, in questo momento, ha delle dimensioni che non corrispondono alle variabili **A** e **B**. Questo, in alcuni casi, sarà voluto, o inevitabile, ma dovrà essere evitato quando possibile. Per restare nelle dimensioni assegnate e avere le proporzioni volute usiamo più estesamente le espressioni:

```
BLOCK a/2, b-0.10, zzyzx
BLOCK a, b, zzyzx *2/3
```

Così *Blocca* è ancora lunga il doppio di *Blocco*, larga 10cm in più e alta i 2/3, ma il tutto resta confinato nei limiti delle dimensioni **A**, **B** e **ZZYZX**.

02.03. GDL odissea nello spazio

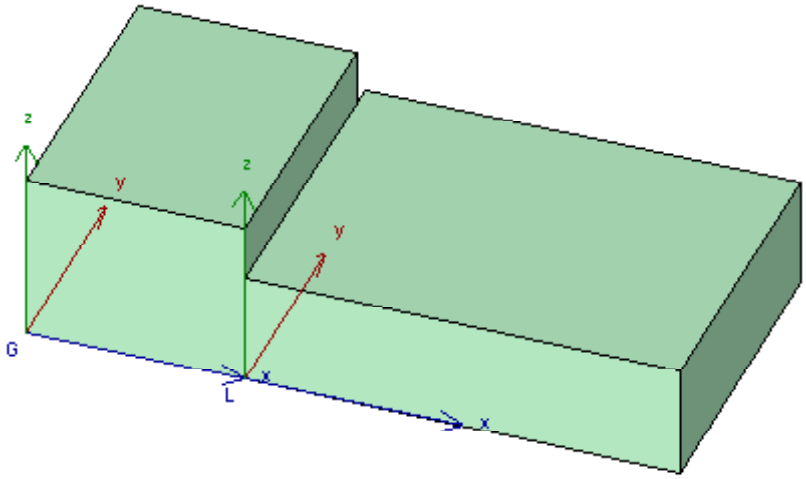
Ora prepariamoci ad un nuovo concetto di grande importanza: muoversi nello spazio. Ogni elemento viene posizionato in base alle sue coordinate spaziali: BLOCK, per esempio, è un parallelepipedo che ha un vertice in 0,0,0 e quello opposto in p1,p2,p3.
Il punto 0,0,0 è l'origine. Ogni oggetto ha un suo spazio tridimensionale con la propria origine, che può essere manipolata a volontà.
Può essere di aiuto considerare l'origine come un cursore 3D, che possiamo muovere con opportuni comandi di spostamento. Ogni istruzione impartita fa riferimento alla posizione corrente del cursore 3D. (Analogamente, in un word processor possiamo spostare il cursore su e giù tra le righe, e a destra o sinistra tra i caratteri. Quando scriviamo il testo appare nella posizione corrente del cursore).
Proviamo quindi a spostarci in modo da portare *Blocca* accanto a *Blocco*. La sintassi per uno spostamento lungo l'asse X è

```
ADDX n
```

Scriveremo quindi **ADDX a/2** prima della seconda istruzione BLOCK, per ottenere uno spostamento pari alla larghezza di Blocco:

```
BLOCK a/2, b-0.10, zzyzx
ADDX a/2
BLOCK a, b, zzyzx *2/3
```

Osserviamo il risultato nella finestra 3D tenendo attiva la visualizzazione degli assi.

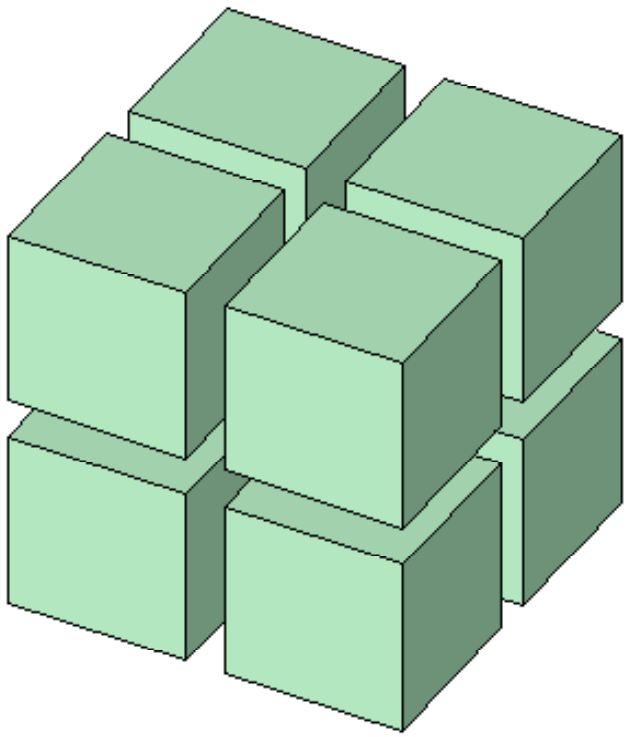


Vediamo che ora abbiamo DUE origini, ciascuna con i propri assi cartesiani. Quella contrassegnata dalla **G** (Global) è l'origine iniziale, mentre quella contrassegnata dalla **L** (Local) è l'origine "corrente", cioè quella utilizzata in questo momento.
Ricordando quanto detto poco fa, notiamo che ora l'intero oggetto è nuovamente più lungo della variabile **A**. Quando si programma non si può mai smettere di ragionare! Come riportare la dimensione X nei giusti limiti ed avere le proporzioni che ci siamo imposti? Credo sia evidente per tutti: *Blocco* è lungo un terzo di **A** e *Blocca* i restanti due terzi.

```
BLOCK a/3, b-0.10, zzyzx
ADDX a/3
BLOCK 2*a/3, b, zzyzx*2/3
```

Se volete salvare la versione modificata dell'oggetto ricordate di aggiornare il simbolo 2D, cancellando prima il contenuto precedente, e procedendo come già visto nella Lezione 01.

Bene. Facciamo un semplice esercizio per impratichirci nell'uso dello spostamento. Vogliamo realizzare un elemento di questo tipo:



una specie di dado tagliato. Definiamo sempre con esattezza le caratteristiche:

- le dimensioni X, Y, e Z devono essere variabili;
- i tagli devono stare a metà delle rispettive facce;
- i tagli devono essere larghi 1/10 della dimensione totale.

Stiamo cominciando a scrivere un oggetto composto da più di due o tre istruzioni, quindi trattiamolo come si deve. Cominciamo sempre con alcune righe di "documentazione". Nel GDL ogni riga che comincia con **!** viene ignorata. È un modo per inserire i cosiddetti *commenti*, utilissimi quando si riapre un oggetto per modificarlo. Iniziate sempre un oggetto con il suo nome o una brevissima descrizione, e mettete sempre la data e la firma.
Righe vuote (anch'esse ignorate) possono essere usate a volontà per migliorare la leggibilità del testo.
Come tutti gli utenti di computer, sapete perfettamente che ci sono molti modi per ottenere uno stesso risultato. Io in genere ne illustrerò uno, voi potete provarne altri che vi risultino più congeniali. Il metodo per prove ed errori è quello che permette di imparare (spesso scoprire) le varie funzioni di un linguaggio.

```
! Lezione 02
! blocco di blocchi

a2 = a * 0.45
b2 = b * 0.45
alt2 = zzyzx * 0.45
```

cominciamo con la definizione di tre variabili che ci tornano comode per realizzare i nostri blocchi. Abbiamo detto che il taglio occupa un decimo della dimensione di ciascuna faccia, quindi il "pieno" è pari ai restanti 9/10. Siccome il taglio è al centro, ogni parte è metà di 9/10, cioè il 45% di quel lato (spero di essere stato chiaro). Quindi le dimensioni dei blocchi sono pari a ciascun parametro dell'oggetto moltiplicato per 0.45 (ovvero moltiplicato per 45 e diviso per 100).
Le variabili "utente" si definiscono nella finestra dei parametri, ma lo script ne può contenere altre, interne, di cui l'utente non ha bisogno. Si definiscono semplicemente scrivendo il nome della variabile seguito dal segno di uguale e dal valore che vogliamo assegnargli.
(Il nome di una variabile può contenere lettere e numeri -senza spazi- e deve iniziare con una lettera).
IMPORTANTE: Le variabili non possono assumere lo stesso nome di una parola chiave! Se ricevete un messaggio di errore di cui non riuscite a trovare la causa verificate di non aver utilizzato, per le vostra variabili, una parola riservata. L'elenco è nell'appendice del manuale GDL, disponibile dall'Aiuto in linea di ArchiCAD.

Ora posizioniamo i primi due blocchi

```
BLOCK a2, b2, alt2
ADDX a2 + a / 10
BLOCK a2, b2, alt2
```

con ADDX ci siamo spostati a destra della larghezza del primo blocco (**a2**) più la larghezza del taglio (**a/10**). Potevamo usare anche l'espressione ADDX a * 0.55.
Ora dobbiamo spostarci lungo l'asse Y. Il comando (ovvio) è ADDY.

```
ADDY b2 + b / 10
BLOCK a2, b2, alt2
ADDX -a2 - a / 10
```

```
BLOCK a2, b2, alt2
```

Notate che il segno meno può essere messo davanti a una variabile rendendola negativa, proprio come si fa per i numeri.
Con l'ultimo ADDX ci siamo rispostati verso sinistra.
Questo metodo funziona perfettamente, ma non è efficiente. Nel realizzare un oggetto complesso si perde rapidamente il conto di dove si trovi il cursore. Peggio ancora, se dopo qualche tempo si vogliono fare delle modifiche all'oggetto bisognerà ristudiare tutto il percorso fatto. Anziché fare nuovi movimenti in direzione opposta per tornare indietro, si possono annullare quelli effettuati. A questo scopo ecco il comando DEL. Un comando **DEL 1** cancella l'ultimo spostamento. Per cancellare più spostamenti in una volta si usa la sintassi **DEL n**.
Inoltre prendiamo l'abitudine di differenziare i comandi di spostamento dagli altri, spostandoli a destra con un paio di spazi. Il nostro script completo potrebbe avere la seguente forma

```
! Lezione 02
! blocco di blocchi

a2 = a * 0.45
b2 = b * 0.45
alt2 = zzyzx * 0.45

! ---- piano inferiore
BLOCK a2, b2, alt2
   ADDX a2 + a / 10
BLOCK a2, b2, alt2
   DEL 1

   ADDY b2 + b / 10
BLOCK a2, b2, alt2
   ADDX a2 + a / 10
BLOCK a2, b2, alt2
   DEL 2

! ---- piano superiore
   ADDZ alt2 + zzyzx / 10

BLOCK a2, b2, alt2
   ADDX a2 + a / 10
BLOCK a2, b2, alt2
   DEL 1

   ADDY b2 + b / 10
BLOCK a2, b2, alt2
   ADDX a2 + a / 10
BLOCK a2, b2, alt2
   DEL 3

END
```

Cosa si può notare?

- In primo luogo le istruzioni di costruzione (BLOCK) ora si vedono meglio.

- Poi abbiamo documentato il nostro lavoro, "etichettando" due sezioni significative.
- Inoltre abbiamo messo la parola "FINE" al nostro script. L'istruzione **END** informa l'interprete GDL che il suo lavoro è terminato. In effetti l'avrebbe capito anche da solo, in quanto oltre questo punto non c'è più niente... ma in seguito vedremo che non sempre l'esecuzione termina all'ultima riga (anzi, questa dovrebbe essere piuttosto l'eccezione, se impareremo a mettere in pratica i princìpi della programmazione strutturata.)
- E i più perspicaci avranno notato che l'ultimo DEL riporta il cursore sull'origine globale, prima di terminare. Anche questo poteva essere evitato, ma non costa niente e in più ci permette, in futuro, di utilizzare questo oggetto chiamandolo dall'interno di un altro. Meglio essere lungimiranti.

Sorvolo su ADDZ, ma parliamo di un altro comando di spostamento: **ADD p1, p2, p3** che ha tre parametri e permette di definire, con una sola istruzione, uno spostamento lungo i tre assi (X, Y e Z rispettivamente). Per annullarlo (essendo un solo comando di spostamento) è sufficiente DEL 1.

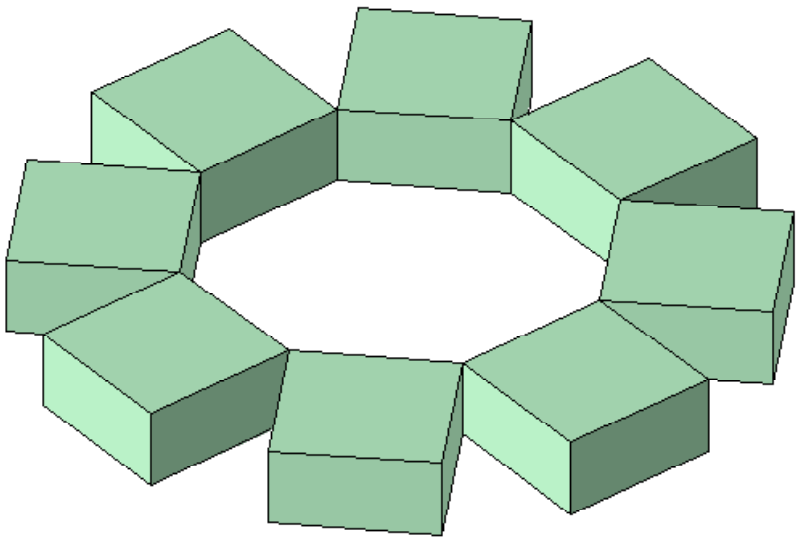
02.04. Tu mi fai girar ...

Ci si può accontentare della traslazioni? Ovviamente no! Deve esserci certo il modo di girarsi intorno. E il modo si chiama ROT.
Come per ADD abbiamo le tre forme semplici:

ROTX n
ROTY n
ROTZ n

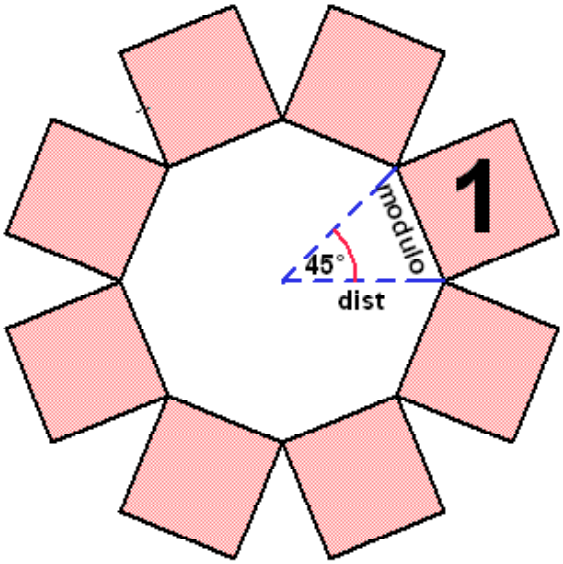
C'è anche una forma composta (ROT x, y, z, alpha) ma meglio trascurarla, per ora. I parametri di ADD sono lunghezze, questi invece sono valori angolari, espressi in gradi. Prepariamoci ad un altro esercizio. Ovviamente stiamo andando verso difficoltà crescenti.

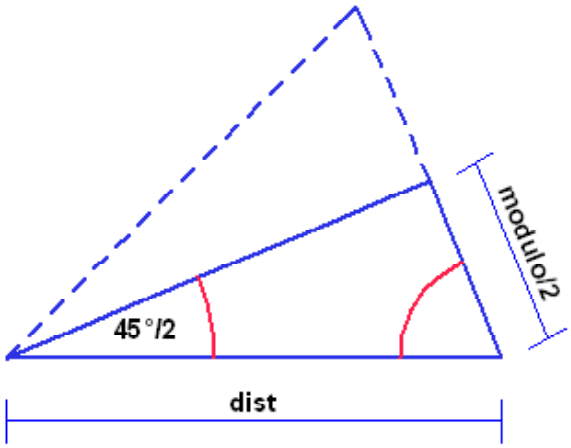
Vogliamo ottenere una disposizione ottagonale di blocchi a base quadrata.



In questo esercizio vogliamo che l'utente possa specificare il lato dei quadrati e l'altezza. Non abbiamo bisogno dell'ingombro e quindi le variabili A e B non verranno utilizzate. Si crei quindi la variabile utente **modulo** per il lato dei blocchi. Pianifichiamo le nostre operazioni. I blocchi hanno posizione e orientamento diverso, ma una simmetria che può tornare utile. Il punto più logico per

l'origine di questo oggetto è il suo centro. Quindi per piazzare il primo blocco (trovare le coordinate del suo vertice inferiore sinistro) dobbiamo trovare la distanza di questo punto dal centro. In pratica si deve risolvere un triangolo isoscele di cui è noto un lato (la variabile **modulo**) e l'angolo opposto (360° / 8 = 45°).





Possiamo risolverlo in più modi. Ad esempio dividendolo a metà, per avere un triangolo rettangolo.

- L'angolo al centro sarà 45/2=22.5°;
- un angolo è retto
- l'altro sarà 180-90-22.5=67.5.
- un cateto è mezzo-modulo
- l'ipotenusa (il dato che stiamo cercando) si può ottenere con: mezzo-modulo *diviso* coseno di 67.5.

proviamo a mettere giù questi dati in GDL.

```
ang1 = 45/2
ang2 = 180-90-ang1
dist = (modulo/2) / COS(ang2)
```

Avete visto? il GDL ci mette a disposizione (come dubitarne?) le necessarie funzioni trigonometriche. E anche le inverse. Ecco le sintassi:

SIN(x) seno dell'angolo x
COS(x) coseno dell'angolo x
TAN(x) seno dell'angolo x
ASN(x) arco seno del valore x
ACS(x) arco coseno del valore x
ATN(x) arco tangente del valore x

L'argomento delle funzioni deve stare sempre tra parentesi. Già che ci siamo metto qui anche altre due funzioni matematiche di uso frequente. Altre ... nel manuale!

INT(x) parte intera del numero x
SQR(x) radice quadrata di x

Torniamo a noi. Ora possiamo portare l'origine in posizione con **ADDX dist** e

orientarci opportunamente per generare il primo blocco.

```
 !----blocco n.1
    ADDX dist
    ROTZ 45/2
 BLOCK  modulo, modulo, zzyzx
    DEL 2
```

Anche per ROT come per ADD occorre usare l'istruzione DEL per tornare alla situazione precedente. Con DEL 2 siamo tornati al centro. E adesso? Tutto daccapo per ogni blocco?
Ovviamente no. Sfruttiamo la simmetria circolare dell'oggetto. Ruotiamo il nostro sistema cartesiano di 45° e andiamo avanti così per i restanti sette blocchi:

```
 !----blocco n.2
    ROTZ 45
    ADDX dist
    ROTZ 45/2
 BLOCK  modulo, modulo, zzyzx
    DEL 2
```

Con DEL 3 si sarebbe annullato anche il primo ROTZ, che vogliamo invece mantenere. Andiamo avanti...

```
 !----blocco n.3
    ROTZ 45
    ADDX dist
    ROTZ 45/2
 BLOCK  modulo, modulo, zzyzx
    DEL 2

 !----blocco n.4
    ROTZ 45
    ADDX dist
    ROTZ 45/2
 BLOCK  modulo, modulo, zzyzx
    DEL 2

 !----blocco n.5
    ROTZ 45
    ADDX dist
    ROTZ 45/2
 BLOCK  modulo, modulo, zzyzx
    DEL 2

 !----blocco n.6
    ROTZ 45
    ADDX dist
    ROTZ 45/2
 BLOCK  modulo, modulo, zzyzx
    DEL 2

 !----blocco n.7
    ROTZ 45
    ADDX dist
```

```
   ROTZ 45/2
BLOCK  modulo, modulo, zzyzx
   DEL 2

!----blocco n.8
   ROTZ 45
   ADDX dist
   ROTZ 45/2
BLOCK  modulo, modulo, zzyzx
   DEL 2

  END
```

02.05. Subroutines!

Non vi sarà sfuggita la scarsa efficienza del metodo illustrato. Ripetere tante volte le stesse istruzioni non sembra molto intelligente. E se poi, in seguito, vogliamo apportare delle modifiche... sarà ancora più frustrante.
Cercheremo quindi di isolare quelle istruzioni che rappresentano una singola "unità logica" e farne una specie di modulo interno, da richiamare ogni volta che serve. Queste parti vengono chiamate *subroutines*. Per essere utilizzate devono:

- iniziare con una cosiddetta Label (cioè un identificativo) che è un numero seguito da "**:**"
- terminare col comando RETURN.

```
! anello di blocchi
! Lezione 02

ang1 = 45/2
ang2 = 180-90-ang1
dist = (modulo/2) / COS(ang2)

!--- main
GOSUB 100
   ROTZ 45
GOSUB 100
   ROTZ 45
GOSUB 100
   ROTZ 45
GOSUB 100
   ROTZ 45
GOSUB 100
   ROTZ 45
GOSUB 100
   ROTZ 45
GOSUB 100
   ROTZ 45
GOSUB 100
   ROTZ 45
GOSUB 100
   DEL 7

END
```

```
!--- blocchi
100:
   ADDX dist
   ROTZ 45/2
BLOCK  modulo, modulo, zzyzx
   DEL 2
RETURN
```

Le subroutine vengono "chiamate", cioè eseguite, con il comando GOSUB seguito dall'indirizzo, cioè dalla "label" che individua la subroutine.
Commentiamo il nuovo listato.

- Appare molto più compatto
- Si direbbe senz'altro più "elegante", e nella programmazione le soluzioni più eleganti sono solitamente le più efficienti.
- Vediamo che è suddiviso in una serie di unità funzionali:
  1. intestazione
  2. dichiarazioni (definizione delle variabili interne)
  3. gestione generale
  4. subroutine.

Questa suddivisione, che consente di avere una gestione semplice anche per oggetti complessi, delegando i particolari a sezioni di programma indipendenti (le subroutine), si chiama *programmazione strutturata* ed è in assoluto la più efficiente. La parola chiave END si trova alla fine della parte centrale, prima delle subroutine.
Come abbiamo visto, GOSUB permette di "fare una deviazione" e spostare l'esecuzione del programma ad un'istruzione diversa da quella seguente. Da questa deviazione si torna indietro con RETURN. Se necessario, è possibile fare anche deviazioni di sola andata, senza ritorno, utilizzando il comando GOTO.

Il programma può essere migliorato ulteriormente, ma per ora facciamo una pausa.

02.06. Un'altra botta al 2D

Nel Capitolo 01 abbiamo imparato a generare un disegno 2D da utilizzare come simbolo dell'oggetto. Ora invece useremo la finestra **Testo GDL 2D** per scrivere qualche comando.
In effetti useremo un comando che genera automaticamente la vista in pianta dell'oggetto:

```
PROJECT2 cod, ang, met
```

cod è un codice che indica il tipo di vista. Noi useremo il **3** (pianta)
ang è l'angolo, l'azimut del punto di vista (telecamera). Per noi **270°**
met è il metodo: **1** per Filo di ferro, **2** per Rimozione linee.

```
PROJECT2 3, 270, 2
```

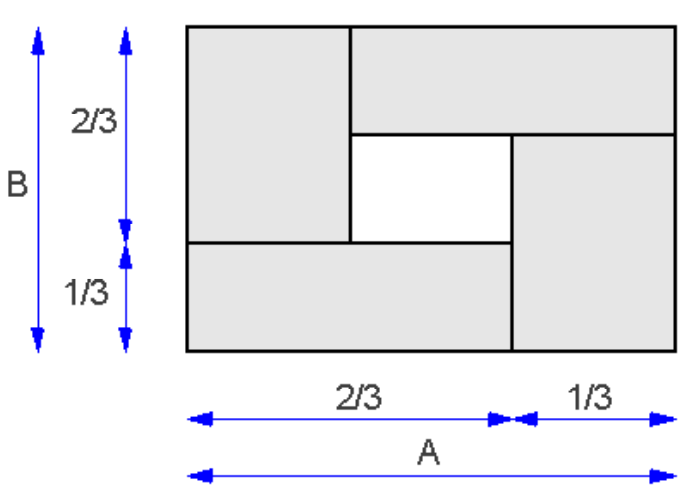
Questo è un metodo molto comodo per ottenere un simbolo 2D per i nostri oggetti. Fin troppo: per generarlo ArchiCAD deve leggere e interpretare tutto lo

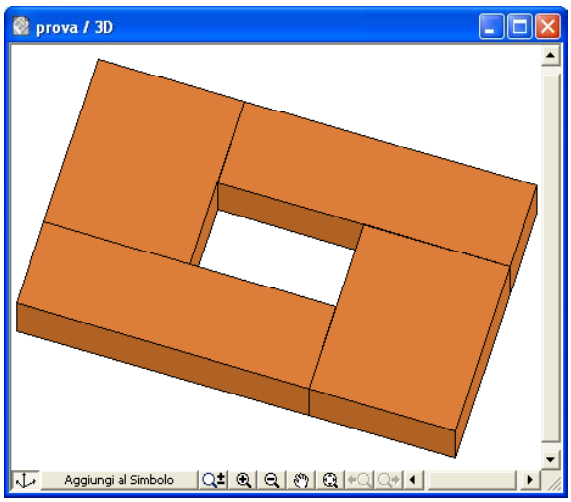
script 3D, costruirlo in memoria ed eseguire la proiezione. Per oggetti complessi questo può non essere efficiente, causando per la generazione delle piante un tempo che può equivalere a quello necessario per le viste tridimensionali. Comunque questo oggetto è molto semplice e rapido da calcolare, e per ora può andar bene così. Torneremo sul tema del 2D in seguito.

Ora possiamo salvare l'oggetto col nome "Blocchi_in_giro".

Esercizio A

È giunto il momento di vedere se siete in grado di mettere in pratica ciò che avete appreso. Provate a realizzare questo semplicissimo oggetto. Non occorre utilizzare le subroutine, dato l'esiguo numero di componenti uguali.





Le dimensioni generali sono **a**, **b**, **zzyzx**. Il lato lungo dei blocchi è pari ai due terzi del corrispondente lato dell'oggetto.

Ricordate che dovrete realizzare anche il simbolo 2D, altrimenti posizionando l'oggetto in pianta... non vedrete niente!

LEZIONE 03

03.01. Cominciamo a cambiare le carte in tavola

Abbiamo terminato la Lezione 02 con l'oggetto Blocchi_in_giro completo e funzionante. Ma non ne siamo del tutto soddisfatti. Ora vedremo come procedere per migliorare, o comunque aggiornare, un oggetto esistente.
La prima modifica che introdurremo dovrà permetterci di scegliere se avere tutti i blocchi della stessa altezza o averli di altezza decrescente (una specie di scala a chiocciola).

A questo scopo creiamo un nuovo parametro di tipo booleano.

Nome: **decre**
Tipo: (booleano)
Descr.: Altezza decrescente
Valore: attiva

E adesso vediamo come intervenire sullo script. Abbiamo bisogno di conoscere l'altezza di ogni blocco. Per il primo è **zzyzx**, per ciascuno dei successivi sette è 1/8 in meno. Calcoliamo quindi il passo verticale e assegnamolo ad una variabile. Inoltre prepariamo un'altra variabile d'*appoggio*, che chiameremo **alt2**, e che all'inizio avrà lo stesso valore di **zzyzx**, da usare per l'altezza dei blocchi. Le prime righe saranno quindi:

```
ang1  = 45/2
ang2  = 180-90-ang1
dist  = (modulo/2) / COS(ang2)
passo = zzyzx / 8
alt2  = zzyzx
```

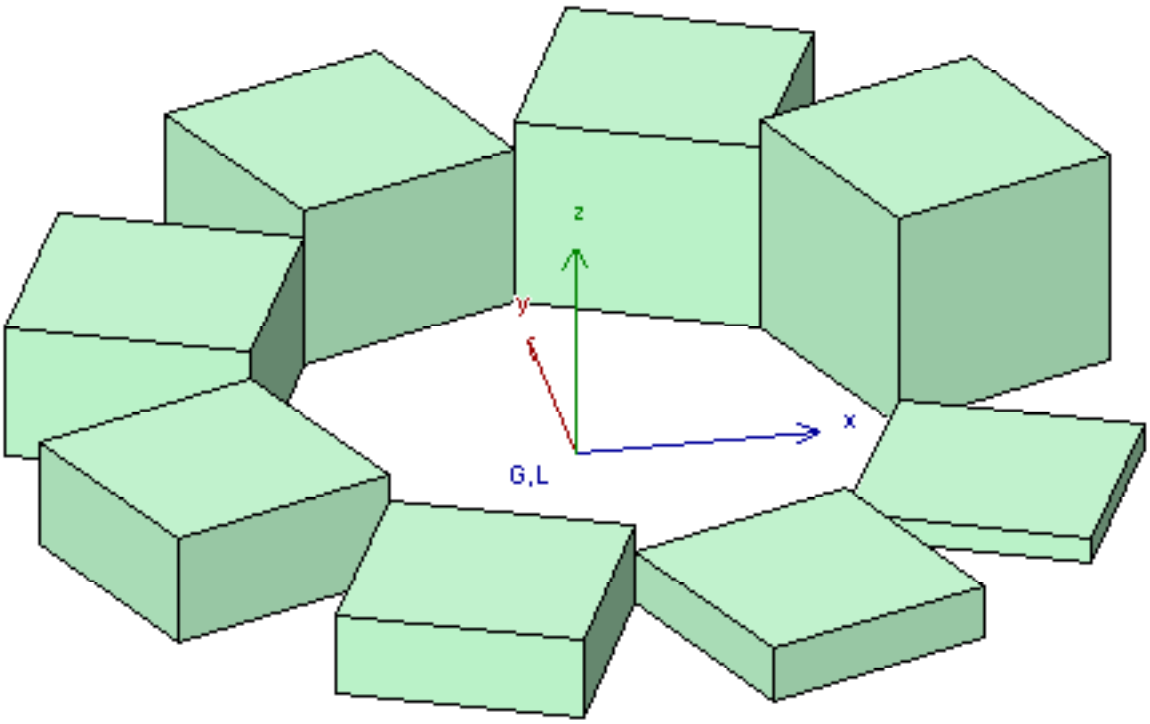
La parte centrale, con le chiamate alla subroutine e le rotazioni non ha bisogno di modifiche

```
!--- main

GOSUB 100
   ROTZ 45
GOSUB 100
   ROTZ 45
GOSUB 100
   ROTZ 45
GOSUB 100
   ROTZ 45
GOSUB 100

   ROTZ 45
GOSUB 100
   ROTZ 45
GOSUB 100
   ROTZ 45
```

```
GOSUB 100
   DEL 7

END
```

la subroutine stessa, invece, dovrà comportarsi con un po' di astuzia.

```
!--- blocchi
100:
   ADDX dist
   ROTZ 45/2
BLOCK  modulo, modulo, alt2
   alt2 = alt2 - passo
   DEL 2
RETURN
```



In primo luogo non usa più la variabile **zzyzx**, per l'altezza del blocco, ma la nuova **alt2**. Poi, subito dopo averla adoperata, ne riduce il valore, togliendogli una quantità pari a **passo**. Così ad ogni nuova esecuzione della subroutine l'altezza del blocco diminuisce. Per chi non ha dimestichezza con i linguaggi di programmazione la sintassi **alt2 = alt2 - passo** può sembrare priva di senso, se vista come un'equazione; tra l'altro con la stessa variabile da entrambe le parti dell'=. In realtà si tratta di un'istruzione di assegnazione che va letta "*assegna alla variabile **alt2** il valore corrente della variabile **alt2** meno il valore della variabile **passo***". Eseguite la modifica e provate a modificare i valori dei parametri. Fatto? Ecco, la scalettatura funziona, ma funziona anche troppo. Non abbiamo più la possibilità di avere tutti i blocchi della stessa altezza.
Infatti non abbiamo ancora utilizzato il parametro **decre**.

03.02. Ci vuole decisione!

Quello che differenzia un vero linguaggio di programmazione da un semplice linguaggio di descrizione, come può essere il DXF, è la possibilità di prendere delle decisioni. Possiamo inserire delle clausole in base alle quali una determinata azione può venire effettuata o meno.
L'istruzione che ci introduce alla vera programmazione si chiama **IF ... THEN ... [ELSE ...]**. Qui ci vuole attenzione, ragazzi. Allora, nella sua sintassi più semplice funziona così: tra IF e THEN si mette una condizione e, dopo il THEN, l'istruzione da eseguire se la condizione è vera. Nella pratica è abbastanza semplice, quindi meglio fare subito il nostro esempio:

```
!--- blocchi
100:
   ADDX dist
   ROTZ 45/2
BLOCK  modulo, modulo, alt2
IF decre = 1 THEN alt2 = alt2 - passo
   DEL 2
RETURN
```

Occorre premettere che le variabili booleane, come la nostra **decre**, possono assumere due soli valori: **1** (attribuito automaticamente se il check-box è barrato) e **0** (se il check-box è vuoto).
Dunque, il nostro interprete GDL arriva alla subroutine 100, si sposta a destra, si gira, costruisce un blocco, e trova un'istruzione condizionale (il nostro IF). Deve verificare se l'affermazione "**decre = 1**" è vera o falsa. Se è vera eseguirà l'istruzione di decremento (**alt2 = alt2 - passo**) altrimenti no. E questo accadrà ciascuna delle otto volte che la subroutine viene eseguita.

Prendetevi il tempo necessario per digerire questo primo assaggio di IF, perché bisogna mandarne giù dell'altro. Fate delle prove, e non andate avanti se, sulle parti spiegate fin qui, vi sono rimasti dei dubbi.

03.03. Avventuriamoci nell'acqua alta

La sintassi utilizzata, forse la più frequente, è quella più semplice. Nella forma completa si può avere non solo un'istruzione da eseguire se la condizione è vera, ma anche una alternativa, da eseguire se la condizione è falsa. Per esempio avremmo potuto avere

```
IF decre=1 THEN alt2=alt2-passo ELSE alt2=alt2+passo
```

che si interpreta così:
*SE **decre** ha valore **1** ALLORA sottrai da **alt2** il valore **passo** ALTRIMENTI aggiungi a **alt2** il valore **passo***.
Con questa istruzione l'utente può scegliere se avere dei blocchi che calano o che crescono, a partire dall'altezza data **zzyzx**.

Digerite anche questo, che ce n'è ancora.
La sintassi vista è quella mono-riga, da utilizzare per eseguire istruzioni singole. Se anziché decrementare o incrementare una variabile e basta, dovevamo com-

piere una serie di operazioni, avremmo dovuto adoperare l'istruzione IF nella forma multi-riga:

```
IF decre = 1 THEN
   ...istruzione 1
   ...istruzione 2
   ...
   ...istruzione n
ENDIF
```

si lascia la prima riga "aperta" dopo il THEN; si scrivono tutte le istruzioni necessarie; infine si conclude il costrutto con la parola chiave ENDIF. Le istruzioni 1,2,...,n vengono tutte eseguite solo se la condizione è vera.
Come per la forma mono-riga c'è anche la versione completa:

```
IF decre = 1 THEN
   ...istruzione 1
   ...istruzione 2
   ...
   ...istruzione n
ELSE
   ...istruzione 1
   ...istruzione 2
   ...
   ...istruzione n
ENDIF
```

in cui il secondo gruppo di istruzioni viene eseguito solo se la condizione è falsa. Tanto per vedere se vi viene l'indigestione, aggiungo che VERO e FALSO sono sinonimi booleani di 1 e 0. Quindi si poteva anche scrivere:

```
IF decre THEN
```

che vuol dire SE **decre** è VERO (cioè =1) ALLORA ...

03.04. Altro giro, altra modifica

Riguardate la sezione "main" del nostro programma. Come vi sembra?
Diciamo un po' ripetitiva... Non va bene! Non siamo noi a dover contare, a dirgli otto volte la stessa cosa. Il computer DEVE lavorare, non FAR lavorare.
Ed eccoci arrivati ad un'altra di quelle pietre miliari che delimitano i confini tra scripting e programmazione: i cicli.
Per dire ad un programma di eseguire n volte un'istruzione si usa il comando FOR ... NEXT. Questo comando definisce una variabile (chiamata contatore) assegnandogli un valore ... ma forse è meglio vedere subito un esempio:

```
!--- main

FOR k = 1 TO 8
   GOSUB 100
     ROTZ 45
```

```
NEXT k
     DEL 8

END
```

Ecco come si riduce la parte centrale del nostro programma usando il ciclo.
Le cose vanno in questo modo: l'interprete, quando arriva all'istruzione FOR assegna alla variabile **k** (il contatore) il valore 1, quindi esegue le istruzioni seguenti fino a quando incontra NEXT. A questo punto incrementa di uno il contatore e (se questo è ancora nel range definito dalla clausola TO) torna ad eseguire l'intero ciclo.
In pratica la prima volta che il ciclo viene attraversato **k** ha valore 1, la seconda 2, ecc. Quando **NEXT k** viene eseguito per l'ottava volta **k** assume il valore 9, e l'interprete non torna più dentro al ciclo, ma prosegue l'esecuzione dall'istruzione seguente.
Notate che, eseguendo otto volte l'intero ciclo, composto da **GOSUB** e **ROTZ**, viene eseguita una rotazione in più (non necessaria), rispetto al metodo precedente. Quindi non **DEL 7** per concludere, ma **DEL 8**.
Questa forma, decisamente compatta, sulle prime può sembrare più "difficile" da leggere, ma in realtà occorre solo prenderci confidenza. I vantaggi sono enormi (soprattutto in programmi più complessi).

03.05. Variazioni senza confini

Siamo sicuri che il nostro oggetto ci serve sempre con OTTO blocchi? e se ne vogliamo uno da SEI che si fa? Tutto daccapo?
La risposta, l'avete intuito, è **no!**

Vediamo, prima, come si dovrebbe modificare il programma per fare un ALTRO oggetto, composto da SEI blocchi, anziché otto:

```
! - ORIGINALE -
ang1  = 45/2
ang2  = 180-90-ang1
dist  = (modulo/2) / COS(ang2)
passo = zzyzx / 8
alt2  = zzyzx

!--- main
FOR k=1 TO 8
  GOSUB 100
    ROTZ 45
NEXT k
  DEL 8

END

!--- blocchi
100:
   ADDX dist
   ROTZ 45/2
BLOCK  modulo, modulo, alt2
```

```
IF decre = 1 THEN alt2 = alt2 - passo
   DEL 2

RETURN
```

```
! - MODIFICATO -
ang1 = 60/2
ang2 = 180-90-ang1
dist = (modulo/2) / COS(ang2)
passo = zzyzx / 6
alt2  = zzyzx

!--- main
FOR k=1 TO 6
  GOSUB 100
    ROTZ 60
NEXT k
  DEL 6

END

! --- blocchi
100:
   ADDX dist
   ROTZ 60/2
BLOCK  modulo, modulo, alt2
IF decre = 1 THEN alt2 = alt2 - passo
   DEL 2

RETURN
```

Questo è il primo passo per imparare a modificare decentemente un oggetto. Capire (ovviamente) *cosa* deve essere cambiato, ma anche *come* e *perché*.
Per inciso: avete notato che per ottenere otto blocchi e per ottenerne sei il numero di istruzioni è identico? Magia dei cicli! Pensate come sarebbe stato col "vecchio" metodo!
Naturalmente facendo semplicemente le modifiche e registrando con un altro nome otterremo un altro oggetto "stupido" come il primo. Ma se capiremo i princìpi, non solo i numeri, e li esprimeremo con delle variabili...

03.06. A domanda rispondi

— ***Perché 8 diventa 6?***
— *Oh beh, questo è il nostro dato di partenza... c'è poco da capire.*
— ***Perché 45 diventa 60?***
— *Quando i blocchi erano otto la rotazione di ciascuno era ovviamente un ottavo di 360. Ora che i blocchi sono sei la rotazione sarà un sesto dell'angolo giro, quindi 60 gradi.*

Insomma il principio è questo: se creiamo una variabile che memorizza il numero di blocchi (8 o 6) potremo usarla per definire tutte le variazioni da apportare

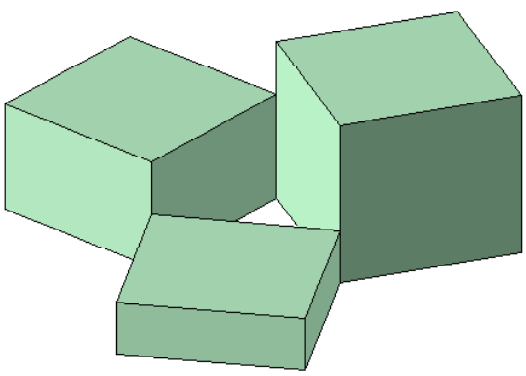
al programma:
detta **num** la variabile basterà mettere **num** al posto di **8** e **360/num** al posto di **45**. Per dare all'utente la possibilità di scegliere quanti blocchi vuole, quindi, aggiungiamo un nuovo parametro:

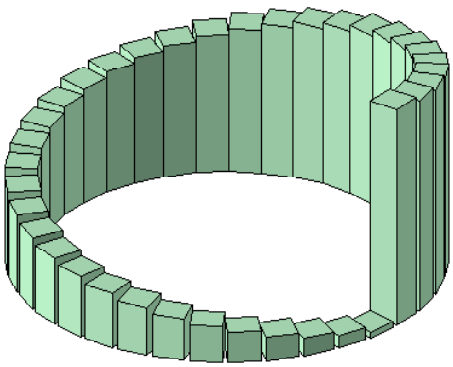
Nome: **num**
Tipo: (numero intero)
Descr.: Numero di blocchi
Valore: 6 (non dimenticate di immettere il valore)

Ora modifichiamo definitivamente il programma:

```
! Esercitazione: Giro di blocchi
! Lezione 03

ang1  = (360/num)/2
ang2  = 180-90-ang1
dist  = (modulo/2) / COS(ang2)
passo = zzyzx / num
alt2  = zzyzx

!——— main
FOR k=1 TO num
  GOSUB 100
    ROTZ 360/num
NEXT k
    DEL num
END ! ================

!——— blocchi
100:
   ADDX dist
   ROTZ (360/num)/2
BLOCK  modulo, modulo, alt2
IF decre = 1 THEN alt2 = alt2 - passo
   DEL 2
RETURN
```

Adesso il nostro bell'oggettino può fare un qualsiasi numero di blocchi: vediamo come si comporta con 3, poi con 36, aumentando anche l'altezza.





Bene, bene. Registratelo pure, direi che così può bastare.

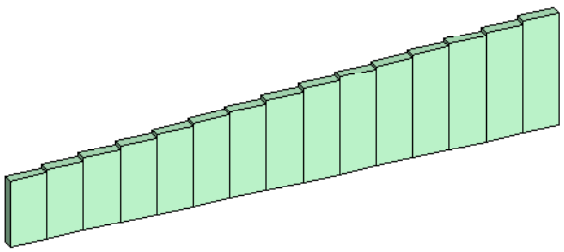
Adesso che ne avete visto la potenza concludiamo la trattazione del comando FOR ... NEXT.
La sintassi completa è:

```
FOR cont = val_ini TO val_fin [STEP incr]
   ...istruzione 1
   ...istruzione 2
   ...
   ...istruzione n
NEXT cont
```

FOR è la parola chiave che dà inizio al ciclo.
cont è una variabile. Può essere utilizzata come tutte le altre variabili, ma non bisogna modificarne il valore!
val_ini è il valore iniziale che verrà assegnato a **cont**
TO è un'altra parola chiave
val_fin è il valore finale a cui deve arrivare **cont**
STEP ancora una parola chiave, questa però è opzionale
incr è il valore di cui deve essere incrementato il contatore ad ogni ciclo.
Se la clausola **STEP incr** non viene usata, l'incremento è pari a uno.

Facciamo un esempio:

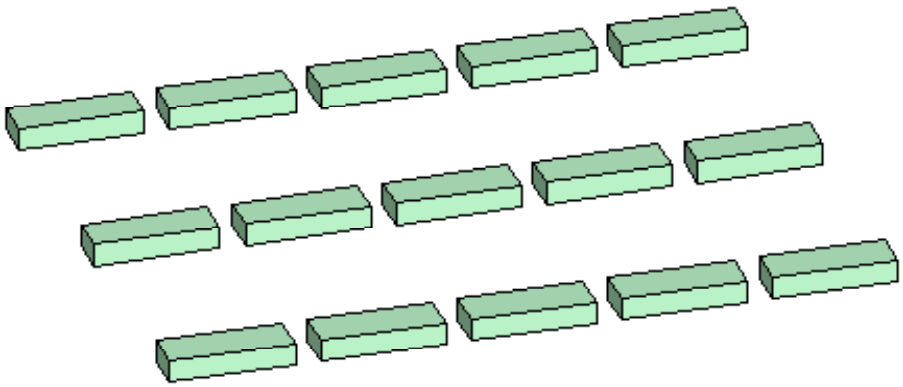
```
FOR k = 2 TO 3.50 STEP 0.10
  BLOCK 1.00, 0.30, k
    ADDX 1.00
NEXT k
```



Qui l'istruzione BLOCK utilizza direttamente la variabile **k** (il contatore). Abbiamo ottenuto una serie di blocchi (per ora non sappiamo fare altro!) di altezza crescente, da 2 metri a 3 metri e mezzo, con incrementi di 10 cm.

Giusto per rendere un po' più ricco il piatto, aggiungiamo che i cicli possono essere, come si dice, nidificati. Cioè se ne può mettere uno dentro un altro.

```
FOR kk = 1 TO 3               !--------\ ciclo esterno
                                        !
  FOR k = 1 TO 5           !-\          !
    BLOCK 1.00, 0.35, 0.20  ! ciclo     !
      ADDX 1.20             ! interno   !
  NEXT k                   !-/          !
                                        !
  DEL 5                                 !
    ADDY 2.00                           !
NEXT kk                     !--------/
```

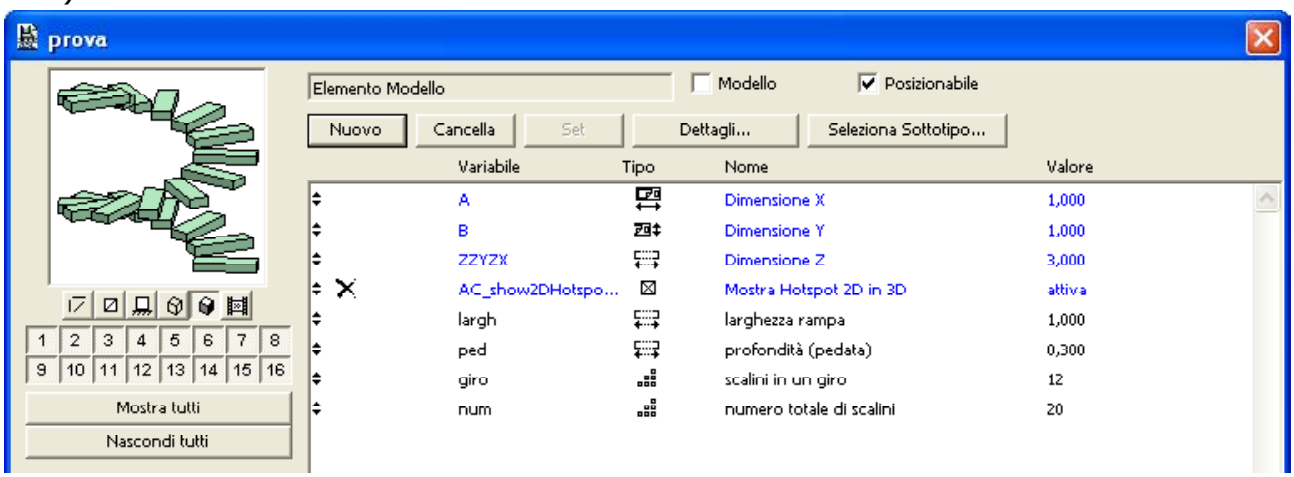


In questo esempio il ciclo esterno viene eseguito tre volte, e ogni volta il ciclo interno viene eseguito 5 volte.
Il ciclo interno realizza una fila di blocchi, spostandosi in X di un metro e venti; al termine del ciclo DEL 5 riporta il cursore 3D alla posizione X iniziale; Poi ADDY lo sposta ortogonalmente di due metri e riparte un altro giro.
Per usare i cicli nidificati occorre fare attenzione a due regole:

1. i cicli non devono mai incrociarsi. Possono esserci anche più di due livelli, ma ognuno deve essere interamente contenuto all'interno del precedente.
2. i nomi dei contatori devono essere diversi.

Esercizio B

Con le conoscenze acquisite non si potrebbe far molto, eppure dovreste essere già in grado di realizzare un oggettino non proprio banale. Una specie di scala a chiocciola! Credete che sia al di sopra delle vostre capacità?
No, basta solo accontentarsi, per ora, di farla con gli scalini rettangolari, e senza poter regolare a piacere il diametro dell'anima (che peraltro è vuota). I parametri da utilizzare sono quelli dell'illustrazione seguente (**A** e **B** non vengono utilizzati):

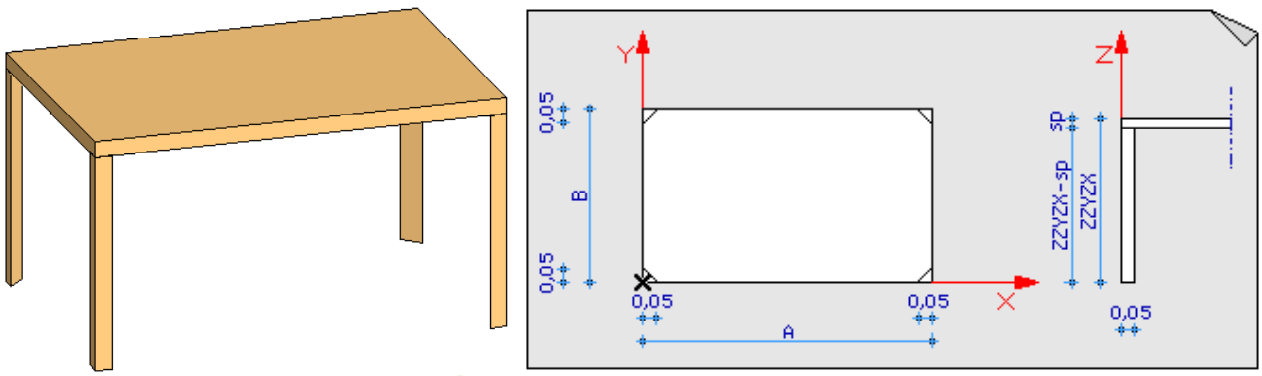


Se non ve la sentite di scriverlo "da zero", provate a partire dall'oggetto precedentemente salvato (Blocchi_in_giro), le modifiche da apportare non sono molte.

LEZIONE 04

04.01. Per fare un tavolo ci vuole un fiore...

... o almeno ci voleva al tempo in cui cantava Sergio Endrigo. Noi proveremo a farlo senza. Solo col GDL.
Prima di tutto dobbiamo **definire** l'oggetto che ci accingiamo a realizzare. Di solito questa è una fase "implicita", ci si limita a pensarci, magari a fare qualche schizzo a mano. In questo caso, essendo un esercizio collettivo, procederemo con una descrizione meticolosa.
Vogliamo ottenere un tavolo col piano rettangolare, quattro gambe di sezione triangolare, metà di un quadrato di 5x5 cm, allineate al filo esterno del piano. Le dimensioni X, Y e Z devono essere modificabili, e anche lo spessore del piano deve essere regolabile.
Da buoni "caddisti" non abbiamo problemi a immaginarci la struttura tridimensionale di questo oggetto.
Ora cominciamo a preparare uno schema delle variabili di cui abbiamo bisogno. Sappiamo dal Capitolo 01 che per la lunghezza e la larghezza dobbiamo usare le variabili predefinite **A** e **B**. Per l'altezza procederemo col solito criterio creando una variabile che chiameremo **alt**, e poi ne useremo un'altra per lo spessore del piano, che chiameremo **sp**. A questo punto la geometria del nostro tavolo può già essere definita completamente. Facciamoci un bel disegnino con pianta e sezione, dove andremo a mettere tutti i dati, quotando ogni elemento lungo i tre assi.



Da questo disegno deve essere possibile ottenere tutte le informazioni dimensionali. Per esempio essendo **zzyzx** l'altezza totale e **sp** lo spessore del piano si ricava che l'altezza delle gambe è data dalla formula (**zzyzx**-**sp**).
Tenere sott'occhio uno schema di questo tipo mentre si crea un oggetto è solitamente di grande aiuto, e carta e penna dovrebbero essere sempre a portata di mano.
Aprite ArchiCAD. Menu **Archivio Oggetti GDL / Nuovo Oggetto**.
Create la variabile **sp**. Assegnate dei valori a tutte le variabili (es.: A=1.60 B=0.80 zzyzx=0.72 sp=0.03) e aprite la finestra **Testo GDL 3D**.

04.02. Fa il suo ingresso PRISM, il factotum

Come abbiamo imparato a fare, inseriamo le prime righe:

```
! Tavolo
```

```
! Lezione 04

altg = zzyzx - sp ! altezza delle gambe
dimg = 0.05     ! dimensione laterale gambe
```

altg non è indispensabile, ma fa comodo tenere una variabile per l'altezza delle gambe, anziché una formula.
dimg sarebbe ancor meno necessaria, in quanto abbiamo stabilito che la dimensione delle gambe è fissa, non modificabile dall'utente. Ma vedremo che anche questo ci aiuta; in oggetti più complessi avere dei nomi, anziché numeri, all'interno dei comandi, rende tutto il listato più comprensibile.
Ora realizziamo il piano, portandoci alla giusta quota

```
  ADDZ altg
BLOCK a, b, sp ! – piano
  DEL 1
```

E a questo punto dobbiamo fermarci. Non ci sono più blocchi da costruire (infatti le gambe sono prismi a sezione triangolare) e noi non sappiamo fare altro.
È giunto il momento di imparare il secondo elemento tridimensionale (due in quattro capitoli... non sembra molto incoraggiante...). Il comando **PRISM** (con le sue varianti) è forse il più utilizzato e, nella sua forma base, è anche piuttosto semplice.
Il manuale GDL lo presenta così:

PRISM n, h, x_1 , y_1 ,..., x_n , y_n
Un prisma verticale con il suo poligono di base sul piano x-y (vedi POLY). L'altezza lungo l'asse z è data dal valore di h. Anche valori negativi di h possono essere usati: in tal caso il secondo poligono di base sarà posizionato al di sotto del piano x-y.
Restrizione dei parametri:
$n \geq 3$

C'è poco da aggiungere, forse. Comunque provo a descriverlo con parole mie.
Si tratta essenzialmente di un solaio, quindi avete presente di cosa si tratta: pianta poligonale, con le facce superiore e inferiore orizzontali, mentre quelle laterali sono verticali. La base inferiore, rispetto agli assi di riferimento del GDL, giace sul piano X-Y, ovvero si trova sempre a quota Z=0.00.
Il primo parametro (**n**) dichiara il numero di vertici del poligono di base, il secondo (**h**) è l'altezza (questo parametro accetta anche valori negativi, il prisma risultante avrà la base superiore *sotto* quella inferiore... insomma sarà semplicemente capovolto). Poi ci sono **n** coppie di coordinate x,y per gli n vertici. In pratica si devono indicare le coordinate di ogni angolo del perimetro, iniziando da un vertice qualunque e proseguendo in senso orario o antiorario.
Per non perdere contatto col nostro beneamato blocco, vediamo che l'istruzione

```
BLOCK a, b, c
```

può essere sostituita da

```
PRISM 4, c, 0.00, 0.00, a, 0.00, a, b, 0.00, b
```

La sintassi è più lunga ma, una volta fattaci l'abitudine, vedrete che la userete alla grande. Tra i vantaggi, da non trascurare che per PRISM non è necessario che l'origine sia preventivamente portata a coincidere con uno dei suoi vertici.
Avrete capito che questo è un comando che ha un numero VARIABILE di parame-

tri, che dipende dal numero di vertici. Possono essere anche diverse decine, e già con quattro la leggibilità (cioè la capacità, per chi osserva, di COMPRENDERE il contenuto del comando) è scarsa. In effetti nell'uso del "buon programmatore" la grafia utilizzata prevede di portare le coordinate di ogni punto su una riga distinta, con rientro di quattro o cinque spazi. Se poi vogliamo essere ancora più bravini cercheremo di allineare le colonne. Talvolta ci saranno dentro anche delle lunghe formule, e non sarà possibile allineare tutto correttamente, ma se ci riusciamo è meglio.

```
PRISM 4, c,
     0.00,  0.00,
     a   ,  0.00,
     a   ,  b,
     0.00,  b
```

Riuscite, mentalmente, a seguire il perimetro? Visualizzate un prisma con 4 angoli, di altezza **c**. Il perimetro inizia dal punto 0,0 poi si sposta in X della quantità **a** ...

04.03. Mettiamolo al lavoro

Insomma è venuto il momento di mettere al lavoro questo nuovo arrivato. Cominciamo a farlo brutalmente:

```
PRISM 3, altg,
     0.00,  0.00,
     dimg,  0.00,
     0.00,  dimg
```

Questa era la prima gamba. Per ora andiamo avanti

```
PRISM 3, altg,
     a     ,  0.00,
     a-dimg,  0.00,
     a     ,  dimg

PRISM 3, altg,
     0.00,  b,
     dimg,  b,
     0.00,  b-dimg

PRISM 3, altg,
     a     ,  b,
     a-dimg,  b,
     a     ,  b-dimg
```

Proviamolo subito, che abbiamo bisogno di quella soddisfazione che solo vedere la nostra creatura che prende vita, ci può dare...
Fatto? funziona? bene! Adesso torniamo un po' indietro. Se il nostro oggetto è "usa-e-getta", cioè ci serve una volta e mai più, possiamo anche lasciarlo così, brutto e sporco. Se invece pensiamo che possa avere vita lunga... è decisamente meglio scriverlo come si deve.

```
! Tavolo
! Lezione 04

altg = zzyzx - sp ! altezza delle gambe
dimg = 0.05     ! dimensione laterale gambe
```

```
  ADDZ altg
BLOCK a, b, sp ! - piano
  DEL 1

GOSUB 150  ! - 1 -

  ADDX a
  ROTZ 90
GOSUB 150  ! - 2 -
  DEL  2

  ADDY b
  ROTZ -90
GOSUB 150  ! - 3 -
  DEL  2

  ADD  a, b, 0.00
  ROTZ 180
GOSUB 150  ! - 4 -
  DEL  2

END ! ================

150: !-------gamba
PRISM 3, altg,
     0.00, 0.00,
     dimg, 0.00,
     0.00, dimg
RETURN
```

Fate il 2D e salvate l'oggetto col nome "TavoloMio".

04.04. Diventiamo materialisti

Per un po' cambiamo argomento, così non vi annoiate. Abbiamo realizzato il nostro tavolo, è quasi come lo volevamo. In effetti a noi serviva un tavolo col piano di marmo e le gambe di legno. Piazzandolo al suo posto nel progetto, però, ci rendiamo conto che possiamo sì assegnargli un materiale, ma solo uno. O tutto marmo o tutto legno.
Selezioniamo l'oggetto in pianta, poi Menu **Archivio > Oggetti GDL > Nuovo Oggetto.** Ormai ci siamo abituati a mettere le mani "dentro". Con *nonchalance* aggiungiamo un nuovo parametro:

Nome: **matp**
Tipo: (materiale)
Descr.: Materiale del piano
Valore: Marmo, levigato

Ora prima dell'istruzione BLOCK inseriamo

```
MATERIAL matp
```

e subito dopo

```
MATERIAL 0 !(zero)
```

La prima istruzione imposta il materiale "corrente", cioè quello da utilizzare, fino a nuovo ordine. La seconda "azzera" il materiale, ovvero imposta il materiale "GENERALE". Per fare le cose fatte bene, anziché GENERALE, noi vorremmo riassegnare il materiale impostato per l'oggetto nella finestra settaggi oggetto. Questo si può ottenere usando una variabile interna messa a disposizione dal GDL (più precisamente vengono chiamate *variabili globali*, e sono tantissime. Potete trovarle nell'appendice del Manuale GDL, disponibile dall'Aiuto in linea di ArchiCAD). Il suo nome attuale è **SYMB_MAT**; quello "vecchio", ancora utilizzabile è **M_**.

```
   ADDZ alt-sp
 MATERIAL matp
 BLOCK a, b, sp ! - piano
 MATERIAL Symb_Mat
   DEL 1
```

A questo punto gli elementi successivi vengono realizzati col materiale indicato nella finestra "Settaggi Oggetto".
Notate che, nella finestra dei parametri, avete scelto il materiale "Marmo, levigato" dalla vostra lista dei materiali. Ora nella casella valore c'è un numero (per me 39) che è il numero d'ordine di questo materiale all'interno della lista. Nelle versioni precedenti del programma (e ancora in qualche vecchio oggetto) non c'erano i menu a tendina per questi parametri, e occorreva scrivere direttamente il valore numerico.
Se avessimo voluto usare un materiale fisso, non modificabile dall'utente, (ad esempio vetro), avremmo potuto scrivere **MATERIAL 24** oppure **MATERIAL "Vetro"** (con le virgolette).

Questo paragrafo è stato fin troppo facile... Aggiungiamo allora un altro argomento: bastano poche parole per descrivere il comando **PEN**.
Funziona in maniera del tutto analoga a MATERIAL, ma accetta solo numeri, o variabili numeriche (le penne non hanno un nome), da 0 a 255.
Ad esempio, inserendo il comando PEN 7, gli elementi seguenti saranno disegnati con la penna numero 7.

```
 FOR k=1 TO 10
   PEN k
   BLOCK 1,1,1
   ADDX 1
```

NEXT k
Questo listato crea 10 blocchi affiancati, usando le penne da 1 a 10. Ovviamente, come accade per il materiale, anche per le penne esiste un apposito tipo di parametro, che mostrerà all'utente la palette di penne correnti.
Già fatto! Semplice, no? Fin troppo...! Allora aggiungiamo anche, per analogia, che esiste il tipo di parametro "Tipo linea", a cui corrisponde il comando [SET] LINE_TYPE che imposta il tipo di linea per il disegno. L'elemento SET è tra parentesi quadre perché è opzionale.
Attenzione! Questo comando è utilizzabile solo nello script 2D, mentre MATERIAL ha effetto solo nello script 3D. PEN, invece, può essere usato in entrambi.

04.05. Non siamo ancora contenti

E adesso il tavolo è finalmente finito? Certo che no! adesso anzi viene il bello. Vogliamo poter scegliere se avere le gambe triangolari, come nei nostri propositi

iniziali, o cilindriche, più tradizionali.
Quindi ecco che inseriamo l'ennesimo parametro:

Nome: **forma**
Tipo: (booleano)
Descr.: Gambe triangolari
Valore: attiva

e apportiamo le modifiche necessarie al programma. L'uso della programmazione strutturata ci permette di farlo anche senza preoccuparci, per ora, di come realizzare le subroutine.

```
! Tavolo
! Lezione 04

altg = zzyzx - sp ! altezza delle gambe
dimg = 0.05       ! dimensione laterale gambe

  ADDZ altg
BLOCK a, b, sp ! - piano
  DEL 1

IF forma = 1 THEN
     GOSUB 100 ! -gambe triangolari
  ELSE
     GOSUB 200 ! -gambe cilindriche
  ENDIF

END ! ================

100: ! -- triangolari --
GOSUB 150  ! - 1 -

  ADDX a
  ROTZ 90
GOSUB 150  ! - 2 -
  DEL  2

  ADDY b
  ROTZ -90
GOSUB 150  ! - 3 -
  DEL  2

  ADD  a, b, 0.00
  ROTZ 180
GOSUB 150  ! - 4 -
  DEL  2

RETURN

150: !-------gamba tri
PRISM 3, altg,
```

```
      0.00, 0.00,
      dimg, 0.00,
      0.00, dimg
RETURN

200: ! -- cilindriche --
RETURN
```

Provate a studiare un po' il listato. Cosa ho fatto?
Ho trasformato l'intera parte che creava le gambe (chiamando la subroutine 150) in una subroutine a sua volta (100), che viene chiamata solo se la variabile **forma** è uguale a 1 (quindi check-box selezionato, nella finestra dei parametri). Altrimenti viene chiamata la routine 200. Che per ora non fa niente, ma impedisce che il GDL segnali errore (l'esistenza di un'istruzione GOSUB 200 dà errore se non c'è una label **200:**, anche se nella fattispecie non viene utilizzata).
Testate già il programma, attivando la vista 3D e provando ad attivare e disattivare il check-box. Sappiamo che non può funzionare, almeno in parte, ma i controlli frequenti sono una buona abitudine. Gli eventuali errori devono essere individuati il più presto possibile. E poi queste verifiche sono effettivamente semplici e veloci.

04.06. Cosa uscirà dal cilindro?

Già, come si fa una cilindro? È facilissimo: **CYLIND h, r**
Può sembrare addirittura più semplice di BLOCK: due soli parametri. In realtà io, per esempio, non riesco mai a ricordare se i parametri sono nell'ordine **h**, **r** oppure **r**, **h**. Ogni volta devo fare una prova e guardare il risultato...
Questo comando genera un cilindro intorno all'asse Z, di altezza **h** e con raggio **r**. La base giace sempre sul piano X-Y, a quota 0.00, quindi anche per il cilindro, come per BLOCK, occorre prima portare l'origine sul punto desiderato, poi eseguire il comando.
Vediamo dunque che per la prima gamba del nostro tavolo dobbiamo effettuare uno spostamento pari al raggio, sia in X che in Y. Dato che questo valore (la misura del raggio) ci servirà in diverse istruzioni, ritengo più pratico predisporre un'apposita variabile.

```
200: ! -- cilindriche --
raggio = dimg / 2

  ADD raggio, raggio, 0.00
CYLIND altg, raggio
  DEL 1

RETURN
```

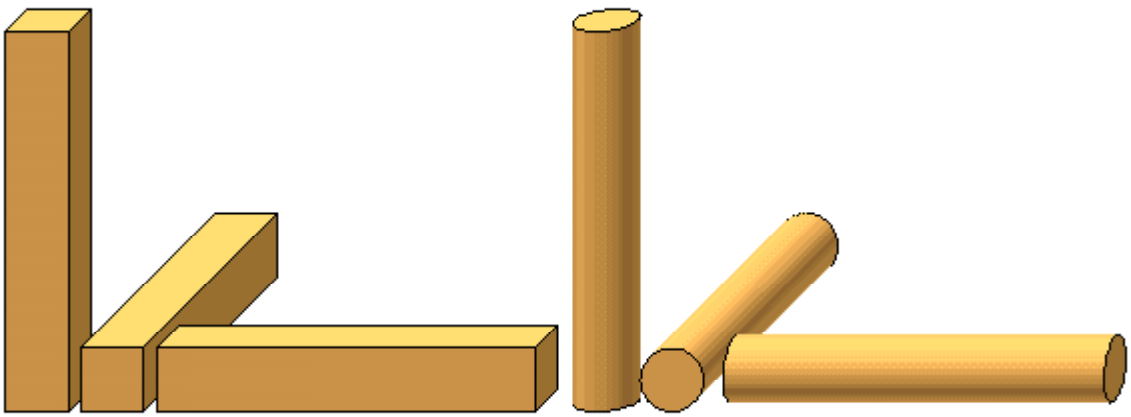
Come sempre, prima di scrivere troppe istruzioni, proviamo a vedere subito il risultato in 3D. Avuta la conferma che stiamo andando bene, proseguiamo sulla falsariga della subroutine 100.

```
200: ! -- cilindriche --
raggio = dimg / 2
```

```
  ADD raggio, raggio, 0.00
GOSUB 250  ! - 1 -
  DEL 1

  ADD a - raggio, raggio, 0.00
GOSUB 250  ! - 2 -
  DEL 1

  ADD raggio, b - raggio, 0.00
GOSUB 250  ! - 3 -
  DEL 1

  ADD a - raggio, b - raggio, 0.00
GOSUB 250  ! - 4 -
  DEL 1

RETURN

250: !-------gamba cil
CYLIND altg, raggio

RETURN
```

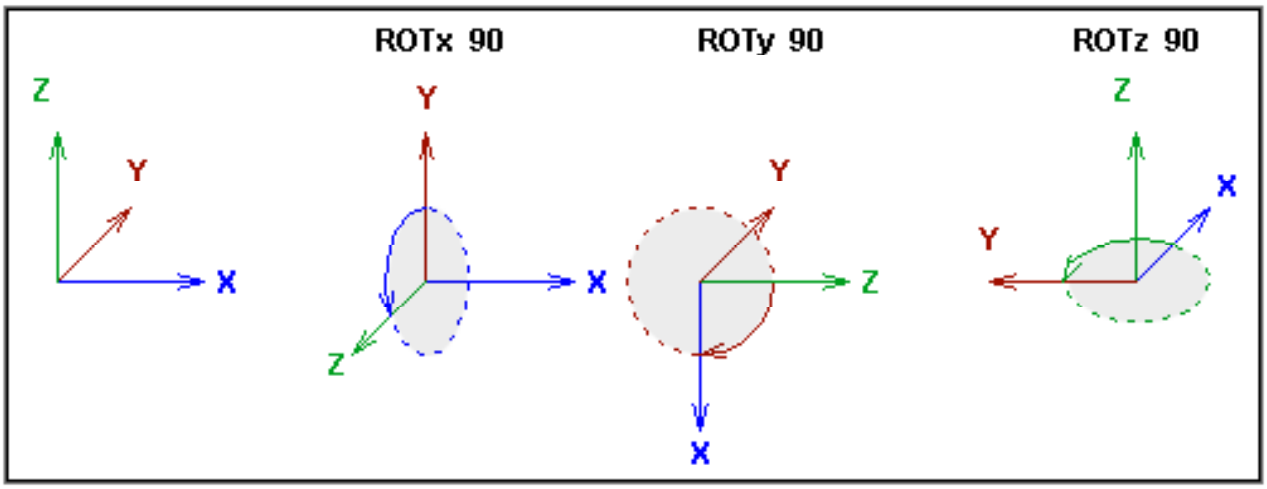
In effetti, in questo caso, la routine 250 può apparire superflua (esegue una sola istruzione: si poteva scriverla direttamente al posto dei GOSUB. Mi fa pensare alle directory create per contenere un solo file...), ma comunque teniamola così. Se in seguito volessimo aggiungere, che so, dei piedini, o le ruote, saremmo già pronti.
Un'ultima verifica al 3D per essere sicuri che tutto funzioni (se non vedete bene le gambe cilindriche, che sono troppo sottili, basta cambiare temporaneamente il valore 0.05 assegnato a **dimg**, ad esempio in 0.25. Ecco che l'aver usato una variabile, al posto di un valore numerico, torna comodo una volta di più).
Salvate l'oggetto e finiamo la trattazione del cilindro.

Il nostro vecchio amico BLOCK potremmo definirlo isomorfo, nel senso che ha lo stesso aspetto lungo tutti e tre gli assi. Il cilindro invece in una direzione ha forma circolare.



Per realizzare i tre oggetti a sinistra, basta regolare i parametri del comando BLOCK. Per i tre a destra, no. Occorre ruotare il cursore 3D in modo da portare

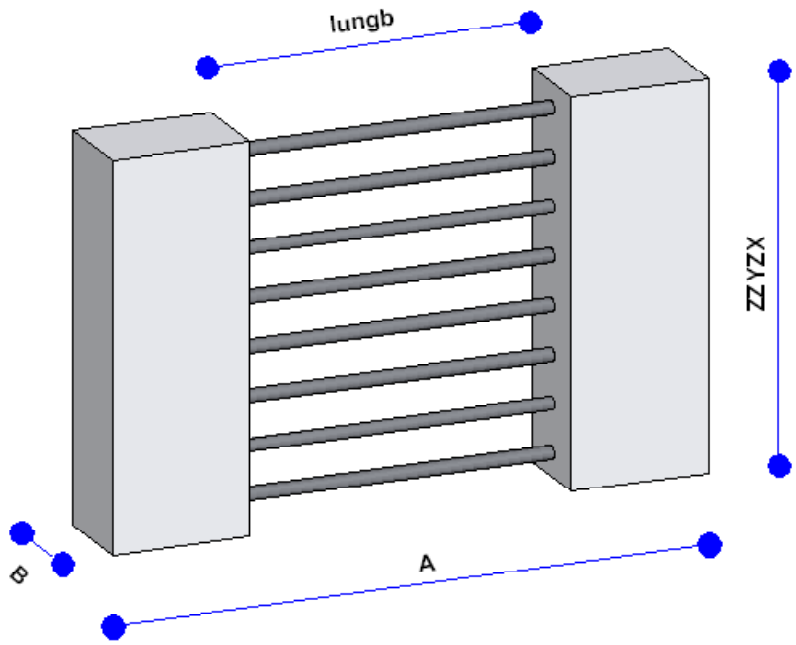
l'asse Z a puntare nella direzione voluta. Il cilindro infatti si sviluppa solo lungo questo asse.



È necessario cominciare a familiarizzare con questo tipo di rotazioni, che tendono a cambiare la posizione degli assi nello spazio, anche se, è giusto dirlo, spesso si procede per tentativi, provando una rotazione e guardando se era quella desiderata. Comunque precisiamo che la rotazione positiva è in senso antiorario. Occorre immaginare di trovarsi su quell'asse e guardare in direzione dell'origine. È possibile anche usare angoli negativi: ROTX -90 è equivalente a ROTX 270.

Esercizio C

A questo punto siete già dei programmatori in erba. Credo di potervi affidare un compito più complesso, che stimolerà il vostro senso dell'avventura nel mondo tridimensionale.
Si tratta di una balaustra formata da due parti laterali piene e una centrale a barre orizzontali. I parametri sono i seguenti:



a	Lunghezza
b	Larghezza (spessore)
zzyzx	Altezza
lungb	Lunghezza barre

numb Numero di barre
diam Diametro barre
matb Materiale barre

Tutti i parametri devono essere liberamente modificabili.
Per le barre dovete usare, ovviamente, un ciclo FOR...NEXT.
Direi anche che, per questo oggetto, potete utilizzare una programmazione lineare, senza strutturarlo in subroutine.

Può sembrare facile, a vederlo così, ma non lo sottovalutate. Cercate di realizzarlo, studiando la spaziatura tra le barre e la divisione dello spazio residuo sopra e sotto.

LEZIONE 05

05.01. Scansati Einstein, arriva MUL

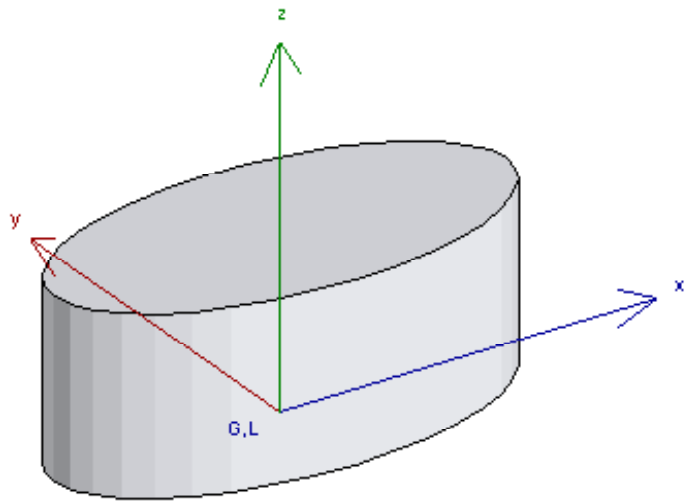
Abbiamo chiuso il la Lezione 04 con la capacità di fare cilindri in piedi e per traverso, ma sempre cilindri... cilindrici! Non c'è un'istruzione per ottenere in modo semplice e diretto un prisma a base ellittica, anziché circolare. Possiamo quindi usare CYLIND per realizzare un tavolo tondo (il piano è un cilindro largo e di altezza molto ridotta), ma per fare un tavolo ovale? Possiamo arrivarci per via indiretta... deformando lo spazio. Abbiamo imparato a usare ADD e ROT; l'ultimo comando della stessa serie si chiama MUL. È anch'esso disponibile nei formati "singolo asse" e "completo":

MULX dx
MULY dy
MULZ dz
MUL dx, dy, dz

Con questi comandi tutte le misure lungo un determinato asse vengono moltiplicate per il relativo parametro. Per esempio:

```
MULX 2
CYLIND 0.50, 0.30
DEL 1
```

produce un "cilindro" alto mezzo metro, largo (in direzione Y) 60cm (due volte il raggio, che è 0.30) e lungo (in direzione X) un metro e venti (due volte il raggio, *moltiplicato* per 2).
Come per gli altri comandi che agiscono sullo spazio 3D, per riportare la situazione alla normalità si usa il comando DEL.



05.02. Un po' d'ordine

Prendiamoci una pausa per fare una semplice considerazione. Creando i nuovi parametri vi sarete resi conto che spesso si vengono a trovare in un ordine poco logico. Quando sono due o tre, questo non è un grosso problema, ma potrebbero essere decine. Avete notato la piccola icona formata da due triangoli, all'estrema

sinistra, prima del nome delle variabili? Serve a trascinarle, per cambiarne l'ordine nella lista. A questo punto vi chiederete a cosa servono sono le altre icone.

La prima, quella con la X, serve a nascondere il parametro. Potete usarlo se non volete cancellare un parametro, ma non volete che compaia nella lista dei parametri, nella finestra settaggi dell'Oggetto. La seconda dichiara che questo parametro è subordinato al precedente. Il parametro della riga superiore avrà il triangolo che permette di "aprirlo" e mostrare i parametri subordinati, che possono essere anche più di uno. L'icona con la B mostrerà il parametro in grassetto. La U, invece, è usata per definire parametri unici, il cui valore non deve essere passato con la combinazione Ctrl-Alt (cursore a siringa).
L'icona a destra del Tipo serve a creare delle matrici, cioè una specie di variabili composte, una cosa un po' complicata, utile quando serve una gran quantità di valori coordinati. Per ora non pensateci.
Già che ci siamo, diciamo anche cosa sono gli ultimi due Tipi di variabili, disponibili nella palette a comparsa. Non sono veri parametri, infatti sono entrambi privi di valore. Separatore è una semplice linea di divisione, come quelle che si vedono spesso anche nei menu del programma. Serve a separare gruppi di parametri. Titolo, come si intuisce dal nome, è un semplice testo, che compare sempre in grassetto nella lista parametri.

05.03. Faccia a faccia con i coni

Torniamo al tavolo che abbiamo amorevolmente realizzato. Siamo così orgogliosi delle sue gambe intercambiabili che decidiamo di farne un'altra serie. A tronco di cono.
Questo introduce due nuovi problemi:

1. non sappiamo fare i coni interi, figuriamoci i coni tronchi.
2. per scegliere il tipo di gambe abbiamo usato un parametro (**forma**) di tipo booleano, che offre due stati: **on** e **off** (più esattamente **vero** e **falso**, corrispondenti ai numeri **1** e **0**). Come fare se le opzioni sono più di due?

Cominciamo dal cono. Si potrebbe pensare a una sintassi praticamente uguale a quella del cilindro (altezza e raggio), e invece no. Col cono hanno voluto strafare, offrendo, in un solo comando, diverse opzioni. La sintassi è:

```
CONE h, r1, r2, ang1, ang2
```

h altezza
r1 raggio della base inferiore
r2 raggio della base superiore
ang1, ang2 inclinazione delle basi

Tanto per cominciare è già un tronco di cono. Per ottenere un cono vero e proprio occorre assegnare zero al parametro **r2**. Poi le due basi non sono necessariamente orizzontali. Perché lo siano occorre assegnare 90 ai parametri **ang1** e **ang2**.
Si potrebbe addirittura usare per fare un cilindro:

```
CONE 3.00, 0.15, 0.15, 90, 90
```

questa istruzione realizza una colonna alta 3 metri, col diametro di 30 cm. Nulla vieta, poi, che la base superiore sia più grande di quella inferiore.

05.04. Liste!

Il secondo problema che avevamo riguardava: dare all'utente la possibilità di scegliere tra più opzioni. ArchiCAD, dalla versione 6.0, ci offre una possibilità grandiosa: creare i nostri menu di scelta personalizzati.
Il manuale ne parla in maniera un po' criptica, ma non è difficile da utilizzare.
Scriverò le istruzioni passo-passo, mentre le eseguo sull'oggetto TavoloMio:

1. Cambiamo il tipo del parametro **forma** da Booleano a Testo.
2. Apriamo la finestra Script Parametri.
3. Scriviamo:

```
VALUES "forma" "Triangolari", "Cilindriche", "Coniche"
```

Notate che il primo parametro del comando VALUES è il nome della variabile a cui vogliamo associare la lista, e i successivi sono gli elementi della lista stessa. Tutti sono racchiusi tra virgolette e separati da virgole, ma la virgola NON c'è tra il nome e i valori della lista.

A questo punto il campo valore viene grigiato e compare un menu a tendina con la nostra lista.
Nella finestra Script Parametri possono essere presenti più comandi VALUES, per creare più liste da associare ad altrettante variabili.

05.05. Ma le gambe...

Adesso abbiamo un parametro con la lista delle gambe disponibili. Il tipo di variabile prima era Booleano, adesso l'abbiamo trasformato in Testo, e c'è qualche piccola differenza. Se provate ad ottenere una vista 3D otterrete un messaggio di errore, perché abbiamo dichiarato che **forma** deve avere un valore testuale (è una "stringa" come si chiama in gergo), mentre nel nostro script gli assegniamo un valore numerico. Apportiamo, per ora, soltanto le modifiche che ci permettono di far funzionare il programma come prima, solo con gambe triangolari e cilindriche. Questa era la struttura precedente:

```
IF forma = 1 THEN
     GOSUB 100 ! —gambe triangolari
ELSE
     GOSUB 200 ! —gambe cilindriche
ENDIF
```

la prima modifica essenziale è:

```
IF forma = "Triangolari" THEN
     GOSUB 100 ! —gambe triangolari
ELSE
     GOSUB 200 ! —gambe cilindriche
ENDIF
```

ricordiamo che gli argomenti di tipo testo devono stare tra virgolette (altrimenti l'interprete GDL non potrebbe distinguere tra i *nomi* delle variabili ed i loro *valori*).
N.B. Per essere certi che la condizione venga correttamente interpretata, state attenti all'uso di maiuscole e minuscole, che sono sempre significative, nei testi tra virgolette.

Adesso preoccupiamoci di inserire tutte le condizioni. Come accade quasi sempre, sono possibili diverse soluzioni. Io ne propongo solo una: anziché una strut-

tura IF...THEN...ELSE...ENDIF userò tre strutture IF...THEN... monoriga.

```
IF forma = "Triangolari" THEN GOSUB 100
IF forma = "Cilindriche" THEN GOSUB 200
IF forma = "Coniche"     THEN GOSUB 300
```

e aggiungiamo subito, in fondo al listato:

```
300: ! -- coniche --
RETURN
```

Questo ci permette di provare subito la funzionalità del programma. Provate a impostare i tre valori possibili di **forma**. Ovviamente scegliendo "Coniche" otterremo il solo piano...
Tutto regolare. Andiamo adesso tranquillamente a completare la subroutine 300, copiando semplicemente le subroutine 200 e 250, e apportando le poche modifiche necessarie:

```
300: ! -- coniche --
raggio  = dimg / 2
raggiobase = dimg / 4

  ADD raggio, raggio, 0.00
GOSUB 350  ! - 1 -
  DEL 1

  ADD a - raggio, raggio, 0.00
GOSUB 350  ! - 2 -
  DEL 1

  ADD raggio, b - raggio, 0.00
GOSUB 350  ! - 3 -
  DEL 1

  ADD a - raggio, b - raggio, 0.00
GOSUB 350  ! - 4 -
  DEL 1

RETURN

350: !-------gamba cono
CONE altg, raggiobase, raggio, 90, 90
RETURN
```

Ed ecco che, in quattro e quattr'otto, abbiamo dato al nostro tavolo un'altra serie di gambe.

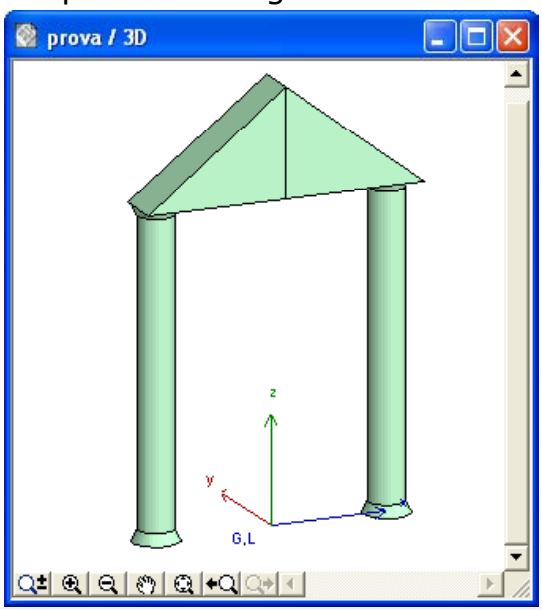
05.06. Una riflessione su MUL

Il comando **MUL** accetta parametri interi, frazionari e... negativi. Quest'ultimo caso può apparire più una bizzarria che una cosa utile, e invece... provate questo

```
raggio1 = 0.15
raggio2 = 0.20
h1      = 0.10
h2      = 2.80
half    = 1.00
```

```
GOSUB 100
  MULX -1
GOSUB 100
  DEL 1

END ! ==============

100:   !— mezza struttura
  ADDX half
CONE h1, raggio2, raggio1, 90, 90

  ADDZ h1
CYLIND h2, raggio1

  ADDZ h2
CONE h1, raggio1, raggio2, 90, 90

  ADDZ h1
  ADDY raggio2 / 2
  ROTX 90
PRISM 3, raggio2 * 2,
    raggio2, 0,
    -half,   0,
    -half,   half
  DEL 6

RETURN
```

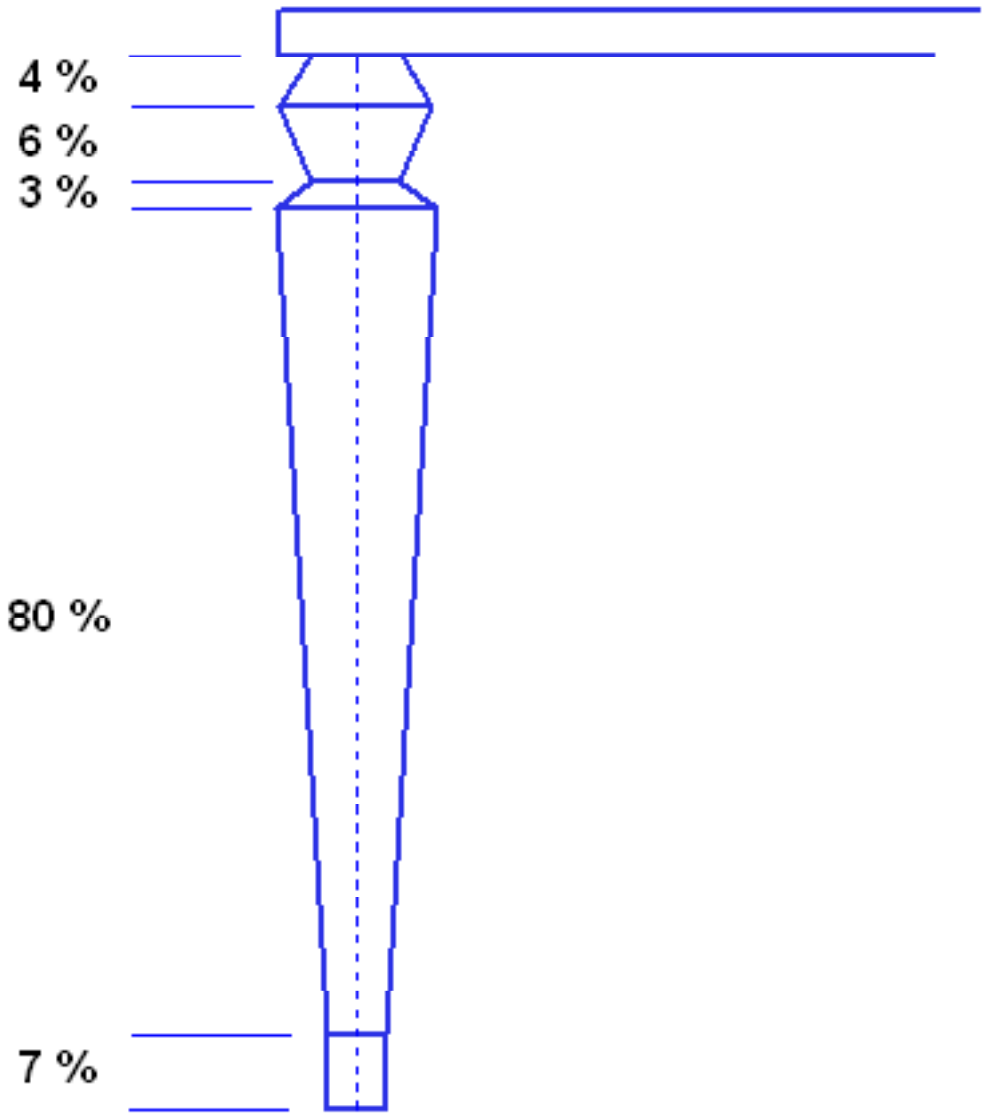
Devo spiegarlo?
Il primo GOSUB crea un elemento (in questo caso a destra dell'origine), poi l'istruzione MULX -1 "inverte" (specchia) lo spazio 3D, rendendo negativi tutti i valori che prima erano positivi (e viceversa), moltiplicandoli per meno uno. Quindi il secondo GOSUB (alla stessa subroutine) costruisce lo stesso elemento dall'altra parte dell'origine.



Con questa tecnica si può semplificare notevolmente l'esecuzione di oggetti complessi che mostrino una simmetria, infatti abbiamo praticamente riprodotto il funzionamento del comando "specchia una copia".

Esercizio D

A parer mio è più un esercizio di "disciplina", che di abilità, non ho dubbi, infatti, che tutti siate perfettamente in grado di realizzarlo. Si tratta di aggiungere un'ulteriore serie di gambe (chiamiamole "Sagomate") al nostro quadrupede. Vedrete che una volta preso il via... è difficile fermarsi!



Vogliamo che le nostre gambucce rimangano proporzionate nei vari elementi, anche se si dovesse cambiare l'altezza, per farne un tavolino da fumo, quindi l'altezza dei vari pezzi (una serie di CONE sovrapposti) deve essere calcolata in percentuale rispetto a **altg**. Per i diametri, regolatevi a piacere.

LEZIONE 06

06.01. Parole, parole

Come era nelle premesse di questo piccolo corso, non darò spiegazioni dettagliate di ogni comando del GDL, ma cercherò di fornire a tutti la capacità di affrontarli con le proprie forze. È giunto quindi per voi il momento di aprire il manuale e... studiare. In questa lezione dovrete prendere confidenza con i comandi del 2D. C'è da dire che il GDL è uno solo, ma oltre ai vari comandi "generici" ve ne sono altri specifici, che possono essere usati solo nelle apposite finestre. Abbiamo visto, ad esempio, il comando VALUES, che deve essere sempre messo nella finestra "Script Parametri". Vi sono quindi i comandi per il disegno del simbolo che devono stare nella finestra "Testo GDL 2D", la quale non può contenere comandi relativi allo spazio 3D. Nella parte superiore delle finestre degli script ci sono dei pulsanti. Cliccando su Controlla Testo il contenuto della finestra viene verificato, ma solo dal punto di vista della correttezza sintattica. Se la logica è sbagliata, ovviamente, non verrà segnalato alcun errore, ma non verrete avvertiti nemmeno se i comandi si trovano in una finestra che non è di loro competenza...

I comandi, peraltro abbastanza semplici, che dovete studiare da soli sono i seguenti (più o meno in ordine di difficoltà):

HOTSPOT2
LINE2
CIRCLE2
RECT2
ARC2

Inoltre date un'occhiata anche alle versioni "piane" delle trasformazioni:

ADD2
ROT2
MUL2

infine provate (fatelo!) a capire anche quest'altro:

POLY2

Con questi comandi, e pochi altri, si può creare il simbolo bidimensionale dell'oggetto, facendo nel 2D più o meno quello che facciamo nel 3D.

— *Ma perché farlo? Abbiamo imparato (Lezione 01) a creare un disegno dalla proiezione del 3D.*

"Non è parametrico" rispondo io.

— *Sì ma sappiamo usare PROJECT2 che crea un disegno sempre aggiornato in automatico.*

"Ma su questo non abbiamo controllo. Per gli oggetti complessi, in particolare con elementi curvi, i tempi di ridisegno sono troppo lunghi. E poi se l'oggetto è ricco di dettagli il simbolo 2D può risultare visivamente troppo carico, e non c'è modo di rimuovere le parti indesiderate".

Insomma bisogna farlo. Anche perché capiterà di creare oggetti puramente bidimensionali, ad esempio simboli impiantistici.

Studiate **davvero** le istruzioni indicate e poi potrete continuare con que-

sta lezione

06.02. Fantasia in cucina

Non bisogna trascurare che la programmazione è fatta di due ingredienti: conoscenza e fantasia. Il primo ingrediente è indispensabile, ma il secondo è il più importante... Non avremmo sui nostri computers tanti bei programmi, se non ci fossero stati tanti programmatori decisi a non rimanere negli schemi. Per dare un esempio di cosa si può fare con un pizzico di inventiva (e nient'altro, praticamente) copiate questo listato nella finestra Testo GDL 2D di un nuovo oggetto:

```
! Lezione 06 - GDL 2D test

FOR k= 1 TO 4
  GOSUB 100
    ROT2 90
NEXT k
    DEL 4

HOTSPOT2    0,    0
HOTSPOT2 -a/2,    0
HOTSPOT2  a/2,    0
HOTSPOT2  b/2,    b/2
HOTSPOT2  b/2,   -b/2

END ! ======================

100:
  LINE2    0,    0,    a/2, 0
  LINE2 b/2, b/2,    a/2, 0
  LINE2 b/2,-b/2,    a/2, 0
  LINE2    0,    0,    b/2, b/2

RETURN
```

È un simbolo che usa in maniera originale i parametri A e B. A definisce le dimensioni X e Y dell'oggetto, mentre B ne definisce l'ampiezza. Il simbolo è *stretchabile* in pianta, e tirandolo può comportarsi in maniera affascinante. Questo resta comunque solo un esercizio... l'uso delle variabili A e B per funzioni diverse da quelle a cui sono normalmente destinate può confondere l'utente finale.

06.03. Rimaniamo nel 2D

Se avete fatto i bravi e avete davvero studiato –o almeno letto– il manuale, avrete notato alcune cose. Intanto alcuni comandi sono spiegati poco o pochissimo. Per i primi comandi, abbastanza intuitivi, la cosa non preoccupa, ma arrivando a quelli più complessi le cose si complicano. La spiegazione c'è, ma spesso è incomprensibile.

Se poi avete provato a usare i comandi (dovreste averlo fatto, se avete un minimo di curiosità e spirito di avventura) avrete visto che POLY2, che dovrebbe fare un poligono campito con un retino, non ne vuole sapere assolutamente di lasciarsi riempire.

Siamo arrivati al nocciolo della questione: la colpa (la difficoltà) non sta nel GDL, ma nel manuale, che sembra rivolto a chi conosce il GDL, non a chi voglia impararlo.
Proverò a darvi qualche spiegazione aggiuntiva io, ma attenzione: queste lezioni non andranno avanti a lungo... non mi addentrerò a fondo nei vari comandi (sono tanti, davvero) quindi cominciate a fare la spola tra quello che trovate qui e quello che dice il manuale. Imparate (se possibile) a interpretarlo e capire quello che *non* dice... perché in seguito dovrete affrontarlo da soli.
Allora, riporto intanto la sintassi:

POLY2 n, framefill, x_1, y_1, ..., x_n, y_n

n - è chiaramente il numero di vertici. Il poligono può essere aperto, quindi il numero minimo di vertici è 2 (praticamente una linea); in questo caso il numero di lati è 1.

framefill - è un codice. Per capirlo meglio bisognerebbe fare una lezioncina di numerazione binaria (è carinissima, vi assicuro) comunque ne parlerò tra breve.

x e **y** - sono le varie coppie di coordinate dei vertici.

Dunque: **framefill** è un numero (tra 0 e 7 compresi) che porta dentro di sé una serie di informazioni. Possiamo immaginarlo come un quadro elettrico con tre interruttori, ciascuno dei quali ha un significato e, se in posizione ON, lascia passare un determinato valore.

- Il primo (valore 1) è da attivare se si vuole il contorno.
- Il secondo (valore 2) è da attivare se si vuole la campitura.
- Il terzo (valore 4) è da attivare se si vuole la chiusura automatica del poligono.

Così il valore 3 (cioè 1+2) indica che vogliamo il retino e il contorno; il valore 6 (2+4) invece ordina di chiudere il poligono e campirlo, senza perimetrarlo, ecc.
Notate il fascino della numerazione binaria:

- Ogni interruttore vale il DOPPIO del precedente.
- Ogni valore (di framefill, in questo caso) può essere ottenuto con UNA SOLA combinazione (univoca) di interruttori accesi e spenti.

Divagazione: Perché una scheda video a 8 bit consente 256 colori? Ha 8 interruttori, che valgono 1, 2, 4, 8, 16, 32, 64, 128. Possono essere, come minimo, tutti spenti (valore 0) e come massimo tutti accesi (valore 1+2+...+128=255). Quindi i colori possibili della palette a 8 bit sono 256 (da 0 a 255). Con 16 bit i colori diventano 65.536. La stessa logica basata sulla numerazione binaria spiega perché la RAM dei computer è espressa da numeri come 32, 128, ecc. Sono tutti multipli di 2. La sequenza 1, 2, 4, 8, 16.... è esprimibile anche come 2^0, 2^1, 2^2, 2^3, 2^4, ecc.

È bene precisare che:

- il terzo interruttore può essere trascurato (io lo consiglio), aggiungendo un ultimo punto coincidente col primo (in pratica chiudiamo noi il perimetro);
- la campitura si può avere solo per poligoni chiusi;
- in alcuni casi non otterremo alcun disegno... ad esempio con framefill=0 richiediamo un poligono senza lati e senza riempimento. Stesso risultato con framefill=2 e perimetro aperto.

È il caso che vi dica anche un'altra cosa, visto che sul manuale c'è, ma è ben nascosta da tutt'altra parte. Anche se provate adesso a utilizzare il comando

POLY2... non otterrete comunque nessuna campitura! Occorre PRIMA utilizzare un'istruzione SET FILL (cioè imposta retino).

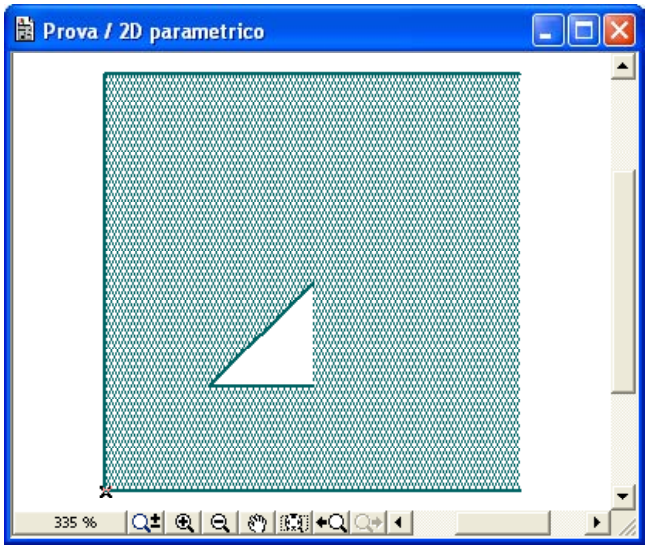
Tutto questo dovevate capirlo (!) dalle poche righe (**mezze** righe, in realtà) che il manuale gli dedica.
Per completezza (e permettervi di provare POLY2) diciamo come utilizzare SET FILL:
Tanto per cominciare la parola SET è opzionale (può essere omessa) e poi funziona come il comando MATERIAL. Ogni retino, come ogni materiale, ha un nome e un numero, ma mentre il numero del materiale è visibile nelle varie liste, quello del retino no.
Possiamo scrivere ad esempio: FILL 18 oppure FILL "Isolamento Acustico". In genere si usa un parametro (tipo: Retino) per far scegliere all'utente la campitura desiderata. Se il parametro lo chiamiamo **riempi**, quindi, scriveremo **FILL riempi**. Con FILL 0 (zero) si imposta un retino vuoto.
Il manuale, a proposito di [SET] FILL dice: "Tutti i poligoni bidimensionali generati dopo questa dichiarazione verranno campiti con il retino dato." In realtà si intende i poligoni tipo POLY2. Ad esempio RECT2 **è** un poligono (secondo me, almeno...), ma non viene riempito.

06.04. Altri interruttori in arrivo

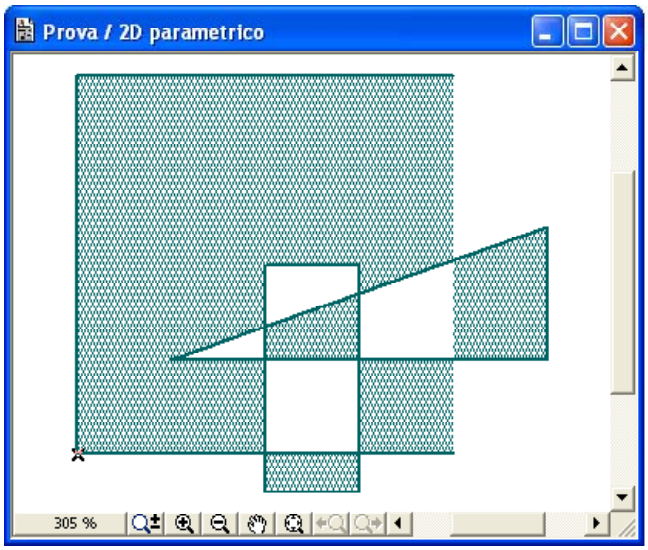
Dopo avervi sconvolti con una semplice istruzione POLY2 come vi posso consolare? Nel dubbio vi darò qualche altra bastonata. Proviamo ad affrontare POLY2_ (*poli-due-underscore*).
È un'estensione del precedente, a cui viene aggiunto un ulteriore parametro per ogni vertice, subito dopo ciascuna coppia di coordinate x-y. Questo parametro viene chiamato "valore di status" e permette di fare due cose: scegliere quali lati visualizzare; fare buchi.
Il valore di status, per questo comando, può essere 0, 1 o -1:
0 = il lato che inizia da questo punto (e va al successivo) non verrà visualizzato.
1 = il lato che inizia da questo punto (e va al successivo) verrà visualizzato.
-1 = definisce la fine di un perimetro (esterno o foro).

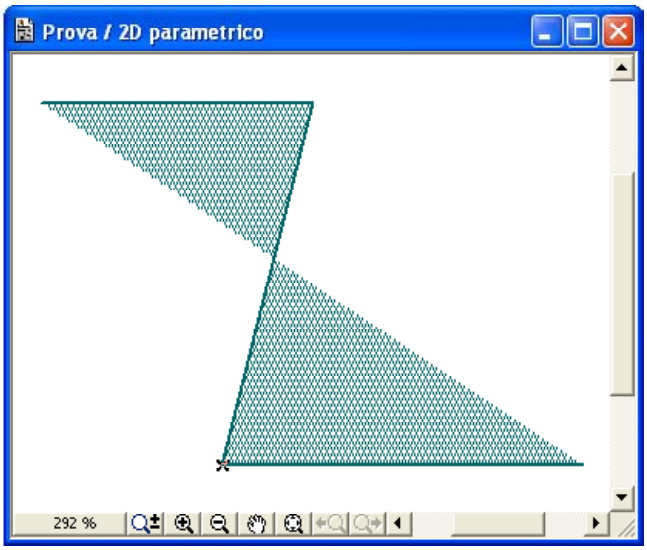
Faccio un esempio:

```
FILL 18
POLY2_ 9, 3,
    0.00, 0.00, 1,   ! \
    2.00, 0.00, 0,     ! perim.
    2.00, 2.00, 1,     ! esterno
    0.00, 2.00, 1,     !
    0.00, 0.00, -1,  ! /
    0.50, 0.50, 1,   ! \
    1.00, 0.50, 0,     ! perim.
    1.00, 1.00, 1,     ! foro
    0.50, 0.50, -1   ! /
```



La prima cosa da dire è che, se si vogliono fare dei fori (usando quindi lo status -1) il perimetro deve essere chiuso, altrimenti si ottiene un messaggio di errore. In altre parole le coordinate del primo punto e quelle dell'ultimo (status -1) devono essere coincidenti. Questo comporta che un rettangolo si definisce con 5 punti, un triangolo con 4 ecc. Quindi la nostra forma, che ha sette lati (4 esterni e 3 interni) deve essere definita come un poligono con (n=) 9 vertici.
Ho scelto di iniziare entrambi i perimetri (esterno e interno) dall'angolo inferiore sinistro e procedere in senso antiorario. Per far sì che i lati a destra non vengano tracciati, quindi, ho messo status 0 alla seconda coordinata di ciascuno.
Si possono definire anche più fori, inoltre non si ha errore nel caso di intersezioni, e nemmeno di... fori completamente esterni al poligono! (ma è meglio evitare queste situazioni anomale: versioni successive del GDL potrebbero essere meno tolleranti, e i vostri oggetti smetterebbero di funzionare).





Ora che vi ho intontiti a sufficienza guardiamo rapidamente le ultime due mutazioni di POLY:

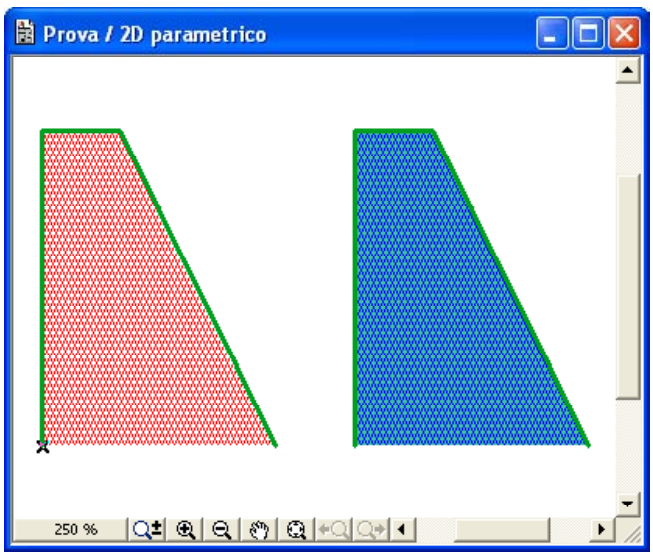
POLY2_A n, framefill, fillpen, x_1, y_1, s_1, ..., x_n, y_n, s_n

POLY2_B n, framefill, fillpen, bgpen, x_1, y_1, s_1, ..., x_n, y_n, s_n

Nel primo comando è prevista la possibilità (parametro **fillpen**) di definire il colore (penna) del retino. Nel secondo è possibile (parametro **bgpen**) definire

anche il colore di sfondo.

```
FILL 18
PEN 4

POLY2_A 5, 3,
    10,       ! penna 10
    0.00, 0.00, 0,
    1.50, 0.00, 1,
    0.50, 2.00, 1,
    0.00, 2.00, 1,
    0.00, 0.00, -1

  ADD2 2,0

POLY2_B 5, 3,
    7, 8,     ! penne 7 e 8
    0.00, 0.00, 0,
    1.50, 0.00, 1,
    0.50, 2.00, 1,
    0.00, 2.00, 1,
    0.00, 0.00, -1
```



Per vedere effettivamente il colore di sfondo, comunque deve essere attiva la relativa opzione in **Opzioni Video**.

06.05. Ne avete abbastanza?

Come già detto, i comandi più "sofisticati", non verranno trattati. Li affronterete da soli, quando vi sentirete pronti e se ne avrete bisogno. Per completare con i comandi "basilari" del 2D manca solo il testo.
Per campire il POLY2 era necessario impostare in anticipo il retino con SET FILL, analogamente per usare il testo occorre prima impostare lo stile, solo che lo stile deve essere prima creato...
Quindi dobbiamo utilizzare ben tre comandi:

1. DEFINE STYLE
2. [SET] STYLE
3. TEXT2

Vediamoli tutti e leviamoci il pensiero.

```
DEFINE STYLE name font, hmm, anchor, cod
```

name - è un nome da assegnare allo style, per poterlo richiamare poi con l'istruzione [SET] STYLE. Deve stare tra virgolette, e ha gli stessi limiti dei nomi delle variabili. Dopo questo parametro NON c'è la virgola.

font - il nome di un carattere disponibile nel computer. Anche questo, essendo un valore di tipo testo, deve stare tra virgolette (a meno che non si utilizzi una variabile).

hmm - l'altezza del carattere, sempre espressa in millimetri.

anchor - può essere un numero da 1 a 9 e definisce il punto di ancoraggio (inserimento) del testo. I punti sono disposti come i numeri sulle tastiere dei telefoni.

cod - Un codice, secondo la tabella seguente:

0	normale
1	grassetto
2	corsivo
4	sottolineato

Notato niente? I codici procedono "al raddoppio", quindi col sistema binario. Possono essere sommati per usare più caratteristiche insieme (es.: 5 per un grassetto sottolineato).

```
[SET] STYLE name
```

Questo non ha bisogno di tante spiegazioni. La parola SET è opzionale, quindi basta scrivere STYLE seguito dal nome dello stile. Di solito non si utilizzano variabili per gli stili (a differenza di quanto accade per retini, materiali ecc., perché lo stile deve essere sempre creato nello script stesso), quindi dobbiamo metterlo tra virgolette.

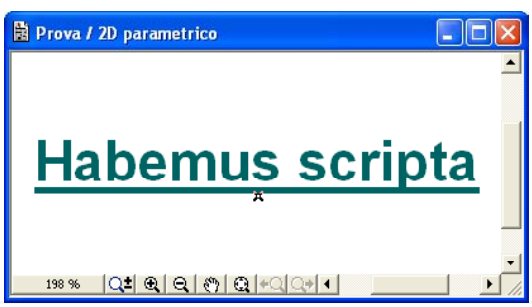
```
TEXT2 x, y, testo
```

x e **y** - sono le coordinate del punto di inserimento (anchor) del blocco di testo.

testo - il testo da scrivere. Può essere un numero, una variabile o una stringa di testo (in quest'ultimo caso il testo deve stare tra virgolette). Può essere anche una formula (verrà scritto il risultato).

Finiamo col classico esempio:

```
DEFINE STYLE "prova" "arial",
       4,    ! mm
       8,    ! in basso al centro
       5     ! grassetto sottolineato

SET STYLE "prova"

TEXT2 0.00, 0.00, "Habemus scripta"

END
```



06.06. Il 2D disegnato

Siamo tutti d'accordo che il GDL tridimensionale è decisamente più interessante. Comunque siamo quasi giunti alla fine, e vale la pena di fare ancora un piccolo sforzo e completare la trattazione con le ultime notiziole sul 2D.

- Prima di tutto diciamo che un oggetto non può vivere senza un punto di inserimento, quindi è necessario mettere sempre almeno un hotspot. Se non ne mettiamo neanche uno, ArchiCAD ne aggiungerà cinque! (ai vertici e al centro). Cliccando sul pulsante Dettagli, sopra l'area dei parametri, si accede ad una finestra in cui c'è un check-box (Hotspot di ingombro) per attivare/disattivare l'aggiunta di questi punti; vengono effettivamente omessi solo se è stato piazzato almeno un hotspot (nella finestra Simbolo 2D, o col comando HOTSPOT2 nella finestra Testo GDL 2D).
- Se anziché utilizzare comandi GDL optiamo per il disegno diretto nella finestra Simbolo 2D, abbiamo a disposizione 16 lucidi (chiamati *frammenti*). La visibilità o meno dei frammenti (solo in editazione) si gestisce tramite i pulsantini numerati da 1 a 16 sotto il riquadro di anteprima dell'oggetto.
- Come simbolo, per l'oggetto, verrà utilizzato il disegno (finestra Simbolo 2D) solo se manca lo script GDL 2D. Se sono presenti entrambi, il GDL ha la prevalenza, e il disegno non verrà visualizzato.

C'è però una "terza via": il GDL può chiamare esplicitamente (visualizzare) il contenuto di uno o più frammenti. In genere questo tipo di oggetti non ha comandi di disegno 2D (anzi, più spesso neanche comandi 3D, si tratta spesso di simboli piani), ma solo uno o più comandi FRAGMENT2 e poco altro.
Per esempio potremmo disegnare un simbolo per una fossa biologica:
Disegnate solo l'ingombro con quattro linee sul frammento 1 (l'utilizzo è quasi identico a quello dei lucidi). Aggiungete anche gli hotspot.
Disegnate gli ulteriori particolari (chiusini, spessori, ecc. a piacere) sul frammento 2.
Create un parametro tipo booleano, chiamato **completo**, per scegliere se visualizzare i particolari.
Nella finestra Testo GDL 2D inserite:

```
FRAGMENT2 1, 0
IF completo THEN FRAGMENT2 2, 0
END
```

Il primo comando ordina di visualizzare il frammento 1. La seconda istruzione dice di visualizzare il frammento 2 solo se **completo** è VERO (cioè la relativa casella è attiva).
Il primo parametro del comando FRAGMENT2 è il numero del lucido da attivare, il secondo può essere 1 o 0 e serve a definire se il frammento deve essere visualizzato con i colori, tipo linea, tipo retino ecc. originali, o quelli correnti dello script 2D.
Al posto del numero di frammento si può scrivere ALL per visualizzarli tutti.

Per risollevarvi un po' vediamo come automatizzare ulteriormente questo oggetto.
Vogliamo che la fossa biologica sia disegnata in maniera dettagliata se stiamo facendo un disegno in scala fino al 100. Se siamo a scala maggiore (es. 200 o 500) vogliamo solo l'ingombro. Bene, ArchiCAD sa a che scala stiamo lavorando, e il GDL può farselo dire, utilizzando una delle apposite Variabili Globali (ne abbiamo già parlato, ricordate?). Questa si chiama **Glob_Scale**.

```
FRAGMENT2 1, 0
IF Glob_Scale <= 100 THEN FRAGMENT2 2, 0
END
```

Se la variabile **Glob_Scale** ha un valore non superiore a 100 viene visualizzato il frammento numero 2. La visualizzazione di questo oggetto viene automaticamente modificata, se cambiamo la scala del progetto in ArchiCAD.

06.07. Relazioni particolari

Forse è la prima volta che vediamo questo tipo di confronto, quindi ne parlo un po'. Finora avevamo usato IF...THEN solo con un confronto tipo **x = y**. "Glob_Scale <= 100" è una *affermazione* e l'interprete GDL deve verificare se è vera. Gli *operatori relazionali*, solitamente utilizzati per i confronti, sono i seguenti:

=	uguale a
<>	diverso da
<	minore di
<=	minore di o uguale a
>	maggiore di
>=	maggiore di o uguale a

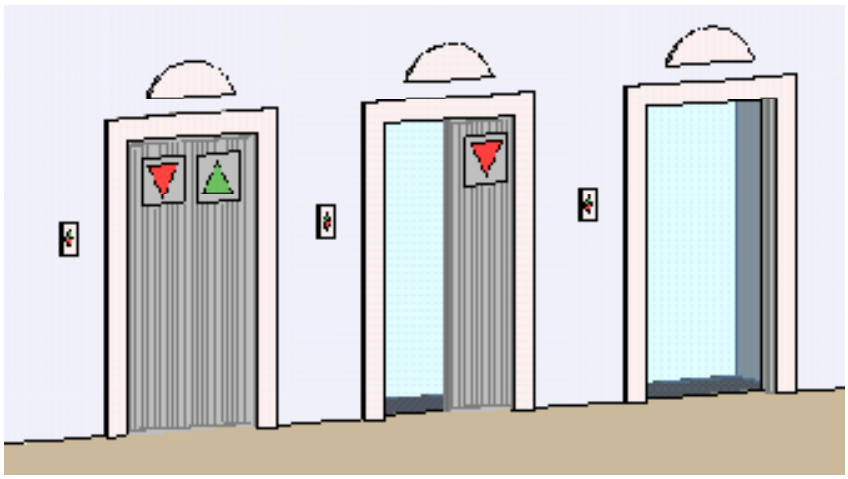
Ma possiamo anche controllare la veridicità di asserzioni più complesse, usando anche gli *operatori booleani*:

AND	"e" logico (questo **e** quello)
OR	"o" logico (questo **o** quello)

Ad esempio potremmo dire

```
IF a=1.00 AND (b<0.05 OR b>10.00) THEN ...
```

Tutta l'asserzione risulta VERA se: la variabile **a** vale 1 metro e la variabile **b** NON è compresa tra 5 cm e 10 metri.
Forse è troppo astratto... allora immaginiamo una situazione realistica un po' complessa (poco, davvero):



Diciamo che stiamo facendo un ascensore con la porta scorrevole che può essere aperta, chiusa o semiaperta. Sulla porta ci sono due stemmi affiancati (in modo tale che con la porta semiaperta se ne vede solo uno), ma l'utente ha a disposizione un parametro per omettere del tutto gli stemmi. Come fare a stabilire se e quali stemmi devono essere realizzati?
Le variabili usate sono: **ste** (1=visualizza; 0=ometti) e **porta** ("Chiusa", "Semiaperta", "Aperta").
Immaginando che la porta scorra verso destra è chiaro che lo stemma destro dovremo farlo solo se **ste** ha valore 1 e **porta** è "Chiusa", cioè le due condizioni

devono essere entrambe vere:

```
IF ste = 1 AND porta = "Chiusa" THEN GOSUB 310
```

Lo stemma sinistro si vede con la porta chiusa o semiaperta:

```
IF ste=1 AND (porta="Chiusa" OR porta="Semiaperta") THEN GOSUB 320
```

In quest'ultimo caso l'interprete GDL lavora nel seguente modo: deve valutare se tutta l'asserzione è vera o falsa. Essendo formata da più elementi, comincia dalla parte chiusa in parentesi: se almeno una delle affermazioni legate da OR è vera, allora il risultato riportato è "VERO". Passa poi a valutare le due espressioni legate da AND, ed il risultato finale sarà VERO se sono vere entrambe. E solo in questo caso si avrà l'esecuzione dell'istruzione posta dopo THEN.
ATTENZIONE: Il principiante è spesso portato a scrivere

```
IF porta = "Chiusa" OR = "Semiaperta"
```

come si usa nella lingua parlata. L'interprete GDL, però, valuta ogni asserzione singolarmente e la prima

```
porta = "Chiusa"
```

la capisce, mentre l'altra

```
= "Semiaperta"
```

la trova incompleta, quindi segnala un errore.

Ai fini del programma avremmo potuto anche scrivere

```
IF ste = 1 AND porta <> "Aperta" THEN GOSUB 320
```

ma ai fini della lezione non sarebbe stato lo stesso!

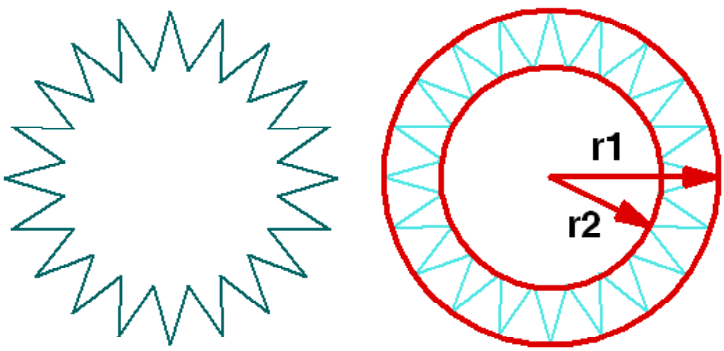
Per i curiosi ecco le due "tabelle di verità" (specie di tavole pitagoriche) per AND e OR, dove 1 rappresenta il valore VERO e 0 il valore FALSO:

1 AND 1 = 1	1 OR 1 = 1
1 AND 0 = 0	1 OR 0 = 1
0 AND 1 = 0	0 OR 1 = 1
0 AND 0 = 0	0 OR 0 = 0

da cui risulta che AND restituisce il risultato VERO solo se i due operandi sono veri, mentre OR restituisce FALSO solo se nessuno dei due è vero.

Esercizio E

Come esercizio, per questo capitolo, voglio proporvi un elemento poligonale stellato.



È abbastanza semplice: oltre alla dimensione l'utente deve poter scegliere il numero di punte e un valore (percentuale) di "stellamento". Le punte della stella si trovano su una circonferenza esterna di raggio **r1** e i punti intermedi sono su una circonferenza più interna di raggio **r2** pari a **r1** moltiplicato per la percentuale data (con 100 si ottiene un poligono convesso regolare, con 0 una raggiera).

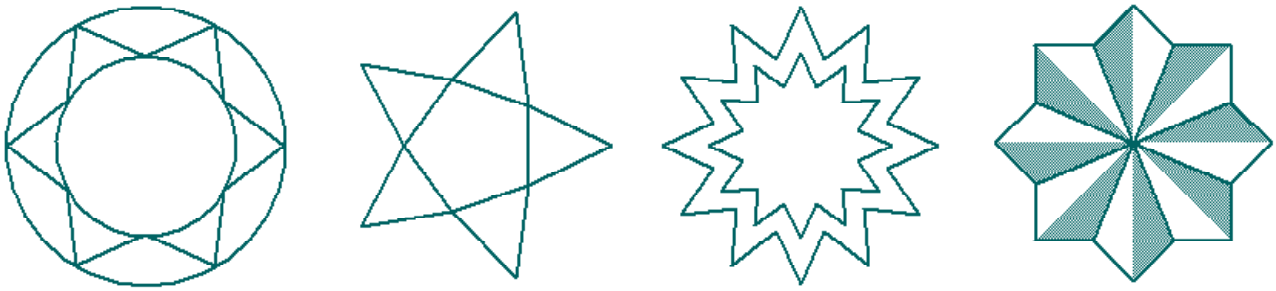
Piazzate subito un hotspot al centro, poi usate un ciclo FOR...NEXT per disegnare i lati e mettere un hotspot su ogni punta.
Usate la variabile **a** nello script, per determinare **r1**, createne una chiamata **ste**, per derminare **r2**, e una chaimata **num**, per il numero di punte, mentre **b** non verrà presa in considerazione. Quando avrete verificato che tutto funziona, mettete all'inizio la seguente istruzione:

```
MUL2 1, b/a
```

con cui i valori lungo l'asse X restano invariati (moltiplicati per 1), mentre quelli in Y vengono ridimensionati in proporzione al rapporto tra le variabili **a** e **b**.
Se volete che l'oggetto sia stretchy (stirabile in pianta) potete mettere due hotspot agli angoli estremi:

```
HOTSPOT2 -a/2, -a/2
HOTSPOT2  a/2,  a/2
```

(Ho detto di usare solo **a**, ricordate?)



Se ci prendete gusto provate a fare delle varianti: campite con POLY2 la metà di ogni punta, o aggiungete cerchi, linee e altre amenità.

VOLETE MIGLIORARE?
Per dare un tocco professionale aggiungiamo un piccolo segmento di codice per la gestione degli errori.
Chiamiamo **ste** il parametro percentuale. Il tipo deve essere numero intero. All'inizio del nostro script mettiamo (attenti, vi sto insegnando di straforo due nuove funzioni):

```
ste = MIN(ste, 100)
```

se l'utente inserisce un numero minore di zero, non verrà accettato "a monte", perché abbiamo impostato il tipo intero, che sottintende "intero positivo". Se inserisce un numero maggiore di 100, questa istruzione interviene, assegnando alla variabile **ste** il valore più basso (MINore) tra quelli elencati all'interno delle parentesi (possono esserci più di due valori).
Nel caso voleste avere la possibilità di usare valori decimali (e quindi il tipo intero non andasse bene) dovreste usare due istruzioni:

```
ste = MAX(ste, 0)
ste = MIN(ste, 100)
```

che riportano il valore di **ste** all'interno del range desiderato (0-100).

Analogo discorso per il numero di punte, che non può essere minore di 3:

```
pun = MAX(pun, 3)
```

oppure (c'è sempre un'alternativa) potete usare:

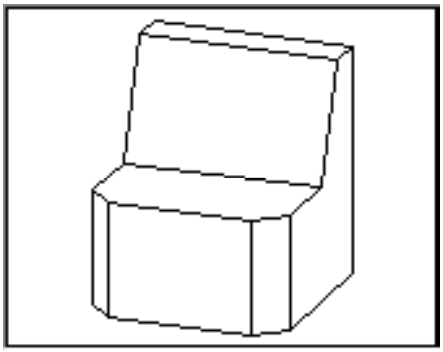
```
IF pun < 3 THEN pun = 3
```

quest'ultimo è il sistema più classico, usato dai programmatori della vecchia scuola. Io lo uso ancora abbastanza spesso (le cattive abitudini sono dure a morire) ma lo trovo meno elegante e anche meno efficiente.

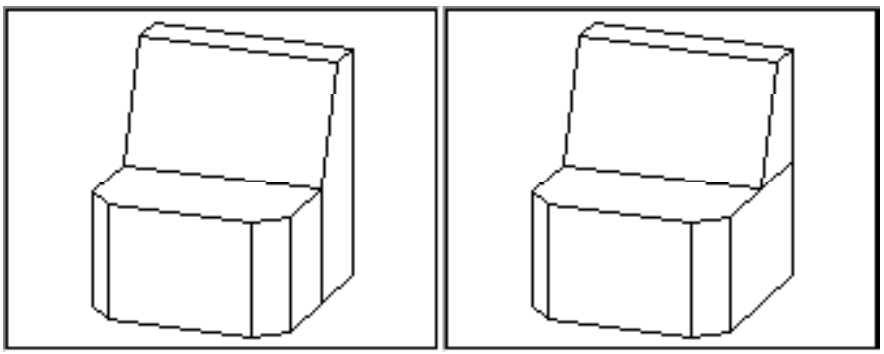
LEZIONE 07

07.01. Ritorno al 3D

Blocchi, coni, cilindri e prismi vari. Ma come si fa a generare forme complesse? Per esempio una nemmeno tanto complessa come questa?



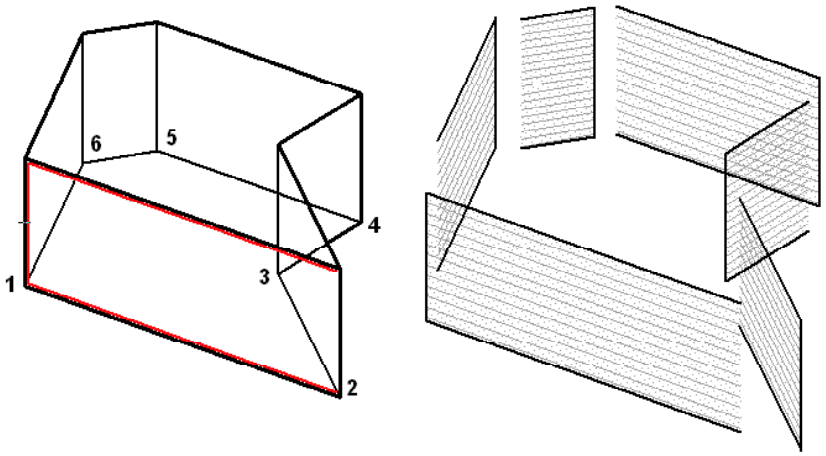
Evidentemente dovremo cercare di ottenerla mettendo insieme più pezzi. Se la studiamo e cerchiamo di scomporla in qualcosa di riconoscibile vediamo che il sedile è un semplice prisma (tipo solaio, per intenderci), mentre lo schienale è rastremato verso l'alto. Ma i più accorti scopriranno presto che anche questo si può fare con un PRISM. Basta ruotare l'origine. In effetti siamo in grado di vederlo anche come un solaio coricato, quindi è un ottimo cliente per il comando PRISM. Abbiamo anche la scelta tra due soluzioni:



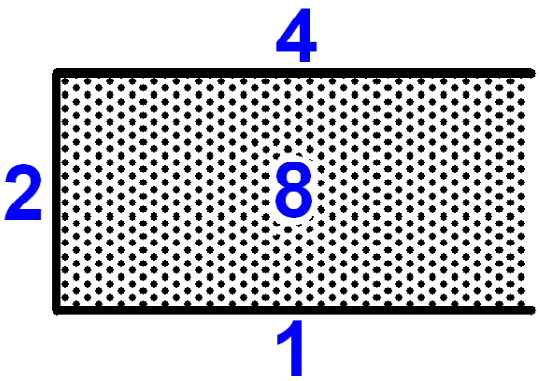
L'unico problema è che noi non vorremmo vedere la linea di giunzione tra i due solidi... Detto che questo problema è relativo solo alle viste con linee (non influenza i rendering) andiamo ad esplorare la soluzione offertaci dal GDL. Si tratta della versione "potenziata" di PRISM, e si chiama PRISM_. La sintassi è:

PRISM_ n, h, x_1, y_1, s_1, ..., x_n, y_n, s_n

e come vedete ha un parametro aggiuntivo (status) per ogni vertice.
Il valore di status va –solitamente– da 0 a 15 e determina la visibilità di tutti gli spigoli, nonché delle facce verticali (le due orizzontali sono sempre visibili). Non è molto difficile comprendere i valori di status, ma non è nemmeno tanto semplice da spiegare. La prima cosa da capire è, visto che nel comando ci sono **n** valori status, e nel prisma abbiamo **n * 3** (ricordate che asterisco vuol dire moltiplicato. Ormai siamo programmatori e questo è il nostro gergo!) spigoli (linee), quali sono i tre spigoli di competenza di ogni vertice? Il primo è la linea verticale che ha proprio sulla sua testa, e gli altri due sono le linee orizzontali che partono da questo e vanno verso il punto successivo.



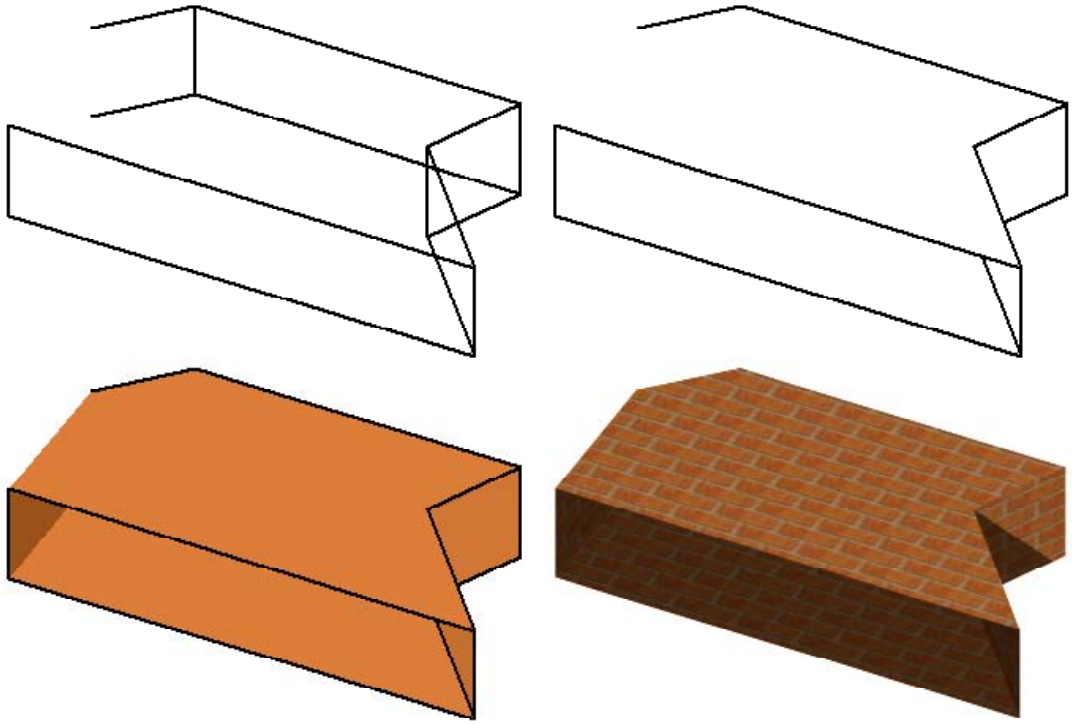
Prendiamo il punto 1 di questa figura, il suo valore di status controlla le tre linee che ho segnato in rosso. A destra vediamo come dobbiamo immaginare scomposto il nostro prisma, e per ogni vertice quali sono le linee (e le superfici) controllate dal relativo status. In questo caso tutto procede in senso antiorario, ma è stata una mia scelta arbitraria.
Fate riferimento al seguente schema:



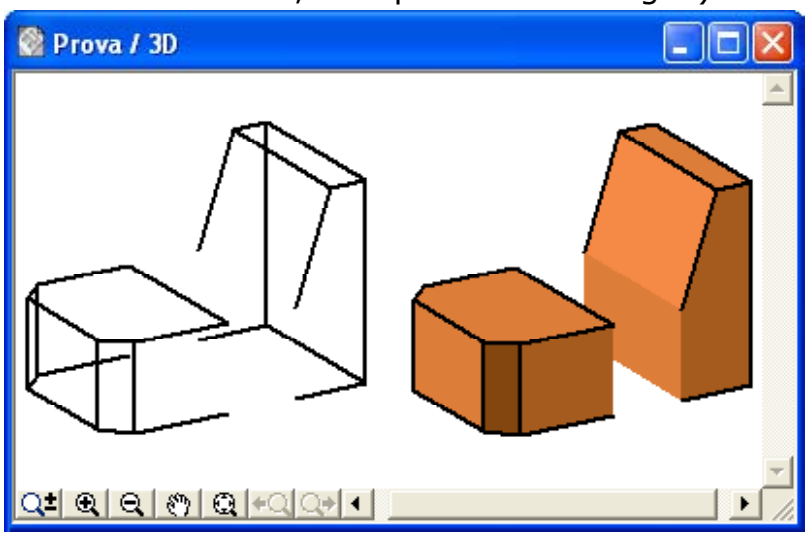
I valori 1, 2 e 4 rendono visibili le rispettive linee. Il valore 8 rende visibile la faccia. Come sempre i valori possono essere sommati, per esempio: 13 rende visibili le linee orizzontali e la superficie (1+4+8).
Al valore 2 si può aggiungere 64, se si desidera che lo spigolo verticale sia visibile solo nel caso in cui sia sul "bordo" dell'oggetto, dal punto di vista corrente (ad esempio, se facciamo una colonna ottagonale, e vogliamo che siano visibili solo le due linee estreme, non quelle centrali).

```
PRISM_ 6, 0.50,
     0.00,  0.00,   7,  ! . . . 1
     2.00,  0.00,  15,  ! . . . 2
     1.50,  0.60,  15,  ! . . . 3
     1.70,  1.00,  15,  ! . . . 4
     0.30,  1.00,  15,  ! . . . 5
     0.00,  0.70,   8   ! . . . 6
```

Questo listato (ho numerato i vertici) produce il prisma che vediamo qui sotto nelle versioni fil di ferro, rimozione linee, ombreggia e render (dove l'interno del prisma resta sempre in ombra, anche se i lati vengono omessi). Il valore di status più frequente è il 15 (tutti gli interruttori accesi), seguito da 13 (omette la linea verticale) o 79 (15+64, che omette la linea verticale se non è di margine).



Ecco quindi come si presentano le due parti della nostra poltroncina (opportunamente distanziate, solo per vederle meglio):



Ed ecco il listato. Ho usato le variabili **a**=0.70, **b**=0.80 e **zzyzx**=0.80. L'origine dell'oggetto è al centro della parte posteriore (trovo che questo sia il punto di inserimento più pratico per piazzare una poltrona).

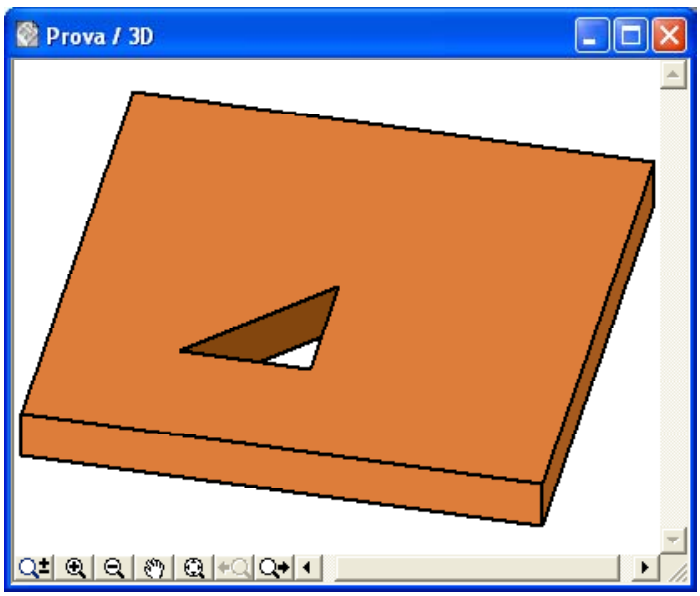
```
PRISM_ 6, .4,    ! -Base
    -a/2,         -0.3, 13,
    -a/2,       -b+0.1, 15,
    -a/2+0.1,   -b,     15,
     a/2-0.1,   -b,     15,
     a/2,       -b+0.1, 15,
     a/2,         -0.3, 12

  ADDX -a/2
  ROTY 90

PRISM_ 5, a,    ! -Schienale
```

```
       0.0,    0.0,   15,
    -zzyzx,    0.0,   15,
    -zzyzx,  -0.15,   15,
      -0.4,   -0.3,    8,
       0.0,   -0.3,   13

  DEL 2
END
```

07.02. E i buchi?

Magari non serve in una poltrona, ma qualche volta avremo bisogno di fare un prisma bucato. Come per POLY2_ c'è un valore speciale di status (-1) che permette di definire dei fori all'interno del prisma. In effetti il comando è molto simile a quello visto nella lezione 05... ho cambiato solo la parola chiave, il secondo parametro (per l'altezza) e i valori di status.

```
PRISM_ 9, 0.20,
   0.00, 0.00, 15,    ! \
   2.00, 0.00, 15,      ! perim.
   2.00, 2.00, 15,      ! esterno
   0.00, 2.00, 15,      !
   0.00, 0.00, -1,    ! /
   0.50, 0.50, 15,    ! \
   1.00, 0.50, 15,      ! perim.
   1.00, 1.00, 15,      ! foro
   0.50, 0.50, -1     ! /
```



07.03. Altre mutazioni

Se è vero che questi prismi sono una specie di solai, perché non possiamo assegnare un materiale alle superfici superiore, inferiore e laterale?
Certo che possiamo! Basta usare il giusto tipo di prisma. Questo, il più completo, si chiama CPRISM_, ed ha appunto tre parametri in più di PRISM_:

`CPRISM_ matsup, matinf, matlat, n, h, x1, y1, s1, ..., xn, yn, sn`

e indicano, nell'ordine, il materiale della faccia superiore, di quella inferiore e

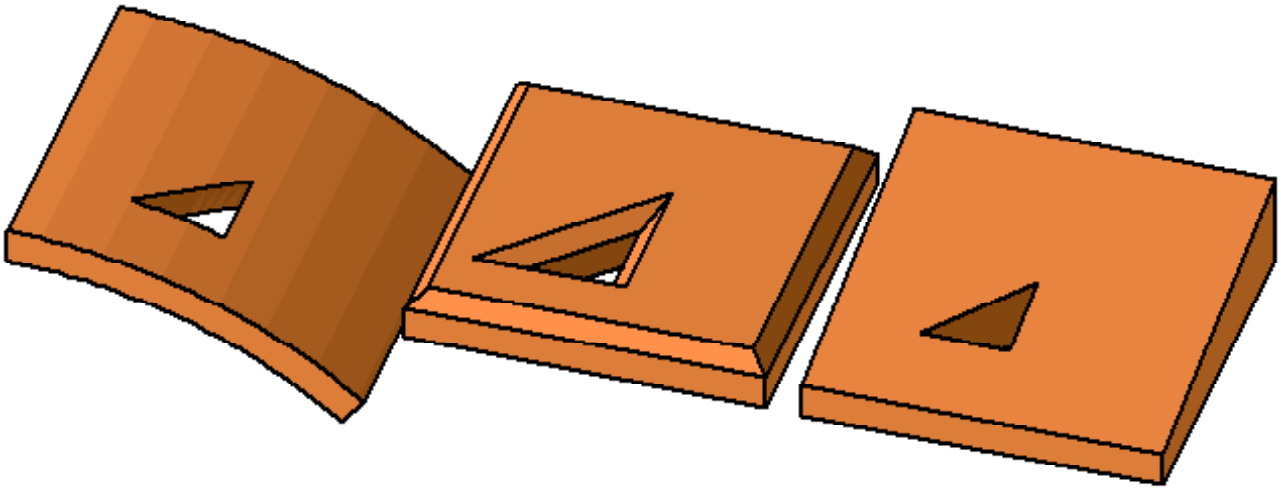
delle facce laterali.

CPRISM_ viene effettivamente usato da ArchiCAD quando trova un solaio nel corso dell'operazione "registra come oggetto".
Esistono altri tipi di prismi, meno semplici (questi erano semplici?), ma di uso decisamente meno frequente. L'importante è sapere che esistono, e se serve basta andare a vedere sul manuale, per avere la giusta sintassi. Se avete capito quanto abbiamo visto finora, non dovreste avere problemi con gli altri.
Vi accenno solo di cosa si tratta:

BPRISM_ crea un prisma simile a CPRISM_, ma curvato, come se fosse poggiato non su un piano ma su una volta a botte.

FPRISM_ crea un prisma in cui è realizzata una smussatura tra il piano superiore e i lati.

SPRISM_ crea un prisma con la superficie superiore inclinata.

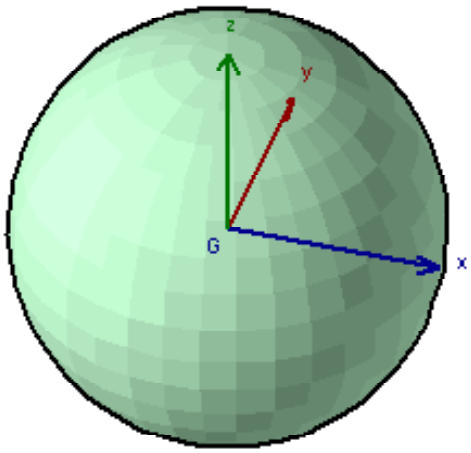


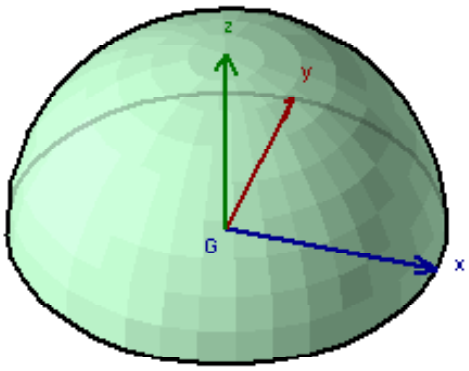
07.04. Ma la Terra non è piatta

Dopo tanti prismi, proviamo a fare qualcosa di meno spigoloso. La sfera, per esempio. Credo che sia il comando 3D più semplice in assoluto:

```
SPHERE r
```

Ha un solo parametro, per il raggio, e costruisce una palla intorno all'origine. In genere, quindi, sarà preceduto da ADDZ r, per non avere mezza palla sotto terra.





Un altro comando, quasi altrettanto semplice è

```
ELLIPS h, r
```

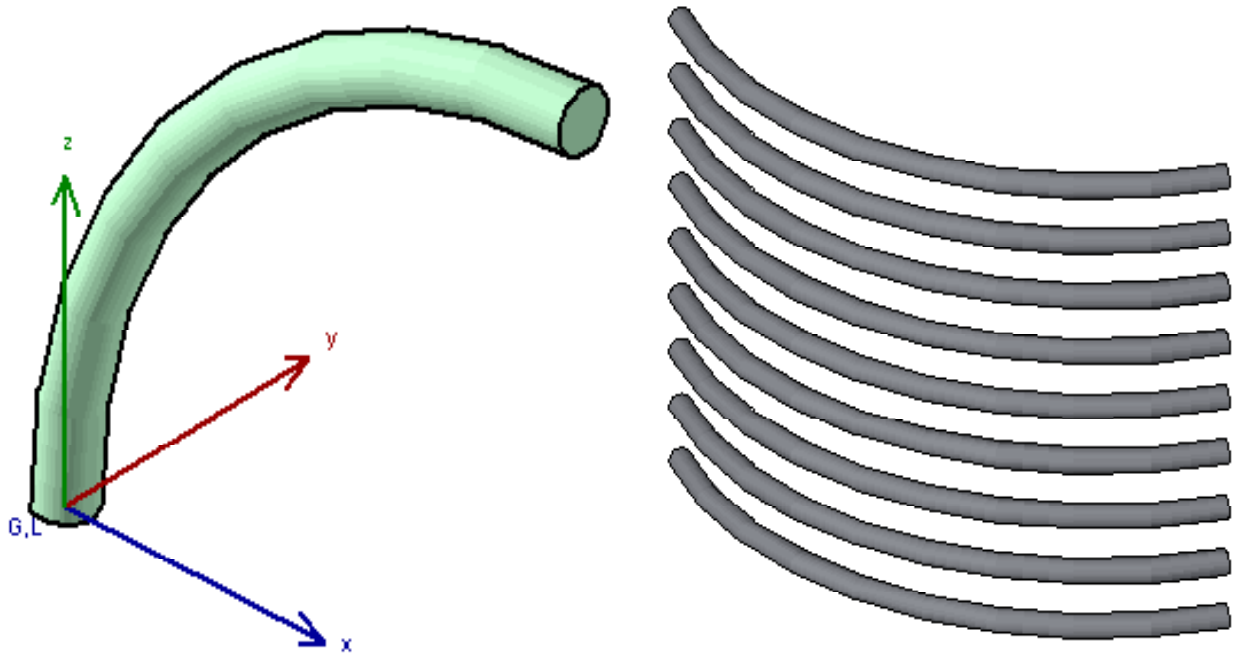
che crea un mezzo ellissoide (una specie di cupola, però piena) di altezza pari al parametro **h**, e con la base circolare, di raggio **r**, il cui centro coincide con l'origine.

07.05. Curve pericolose

Tanto per restare in tema, vediamo quest'altro comando:

```
ELBOW r1, alfa, r2
```

con cui si possono realizzare elementi tubolari curvi. Il tubo ha raggio **r2**, mentre **r1** è il raggio di curvatura e **alfa** l'angolo di sviluppo. Il tutto parte con il cerchio generatore orizzontale, poggiato sull'origine, e si sviluppa verso l'alto, curvando in direzione dell'asse X. Insomma guardate l'illustrazione, che è meglio!



Spesso servirà un tubo con andamento orizzontale, e in questi casi lo faremo precedere da **ROTX 90**.

Questo comando ci introduce su un sentiero alquanto spinoso, meglio affrontato su The GDL Cookbook, di David Nicholson-Cole (la cui lettura è consigliata a chiunque sia in grado di capire un po' di inglese scritto): le curve non sono mai veramente curve. Sono sempre approssimate tramite segmenti lineari, come si vede bene nell'immagine della sfera (parlo di elementi solidi, quelli bidimensionali SONO curvi). Se avete avuto modo di utilizzare il comando "Settaggi Bacchetta Magica" del menu Strumenti di ArchiCAD siete già sulla buona strada per capire l'argomento.
Anche nel GDL abbiamo più opzioni per impostare la risoluzione degli elementi curvi:

RESOL n - è il più utilizzato, divide l'angolo giro in n parti. L'impostazione di default nel GDL è RESOL 36, per cui, ad esempio, una curva di 90° è rappresentata con una spezzata di 9 segmenti.

TOLER delta - con questo si indica l'errore massimo, cioè la distanza (freccia) tra l'arco effettivo e il segmento che lo approssima. TOLER 0.02 fa sì che

l'errore massimo sia pari a 2 centimetri.

RADIUS rmin, rmax - definisce due misure limite. I cerchi con raggio inferiore a **rmin** vengono rappresentati come esagoni, quelli con raggio superiore a **rmax** come poligoni con 36 lati. Per gli elementi di misura intermedia il numero di lati è calcolato proporzionalmente (la formula usata è sul manuale).

Tenete presente che ciascuno di questi tre comandi, quando viene utilizzato, esclude gli altri due. Solo uno di essi può essere attivo in un dato momento.
La difficoltà di cui si parlava a proposito di ELBOW, quindi, è che, se abbiamo un arco grande, ad esempio una ringhiera attorno ad una aiuola circolare di 2 o 3 metri di raggio, per non vedere la segmentazione vorremmo usare, ad esempio, RESOL 180, per avere un segmento ogni due gradi, che con tre metri di raggio equivale a circa 10cm. Il problema è che questo non produce 180 segmenti e basta. Anche il cerchio che genera il tubo risente della **stessa** risoluzione, quindi ognuno dei 180 segmenti di tubo è di fatto un prisma con 180 facce... quindi abbiamo raggiunto in un momento la bella cifra di 32.400 poligoni da calcolare, per ogni rendering, rimozione linee ecc.
Una sfera normale (RESOL 36) è composta da 1.296 facce. Occorre trovare, di volta in volta, il giusto compromesso (come sempre!).

07.06. Una vecchia pendenza

Dopo i prismi e le palle, con cosa volete giocare? Beh, i più svegli si saranno accorti che alla nostra scatola di costruzioni manca qualcosa. Abbiamo i pezzi per fare muri, colonne, solai, ma ci mancano i tetti. Niente paura, eccoli qua. Si chiamano SLAB e sono fatti così:

SLAB n, h, x_1, y_1, z_1, ..., x_n, y_n, z_n

questo è il tipo semplice. Per ogni vertice devono essere definite le tre coordinate X, Y e Z. La superficie deve essere piana, ma l'interprete GDL non effettua alcun controllo. Quindi l'uso è semplice solo se la forma è semplice... La coordinata Z indica la quota dei punti sulla superficie inferiore (a meno che non si indichi un'altezza **h** negativa...).

SLAB_ n, h, x_1, y_1, z_1, s_1, ..., x_n, y_n, z_n, s_n

e questo è il tipo che permette, con le stesse modalità viste per PRISM_, di utilizzare i valori di status per omettere linee e superfici.

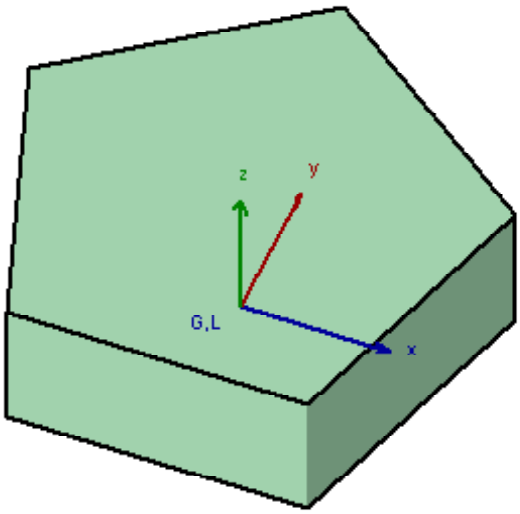
CSLAB_ matsup, matinf, matlat, n, h, x_1, y_1, z_1, s_1, ..., x_n, y_n, z_n, s_n

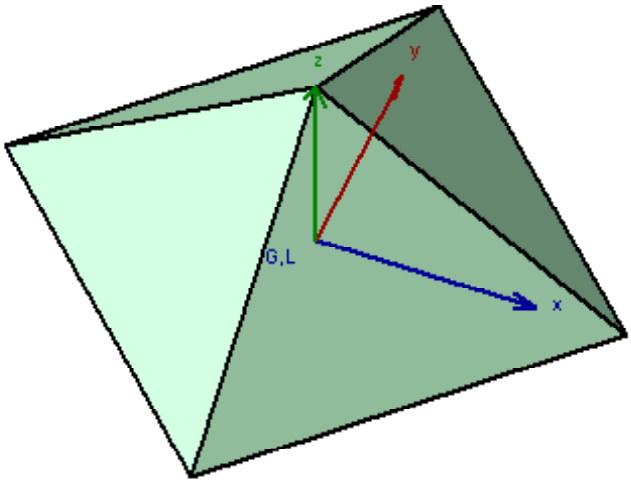
quest'ultimo infine ci consente di scegliere anche il materiale da associare alle facce superiore, inferiore e laterali. È il tipo che usa ArchiCAD quando crea gli oggetti dalla pianta.

Avete notato che ultimamente le mie spiegazioni sono molto più sintetiche? Se riuscite a seguire ugualmente il discorso, anche se con un po' di sforzo, vuol dire che state imparando...

Esercizio F

Studiate un po' RESOL. Può essere utilizzato per ottenere prismi regolari: RESOL 5 seguito da CYLIND 1,1 fa un prisma pentagonale. RESOL 4 seguito da CONE 1,1,0,90,90 ci dà una piramide a base quadrata!





Come si può notare il primo vertice è sempre posto in direzione dell'asse X. Si può intervenire, se serve, con ROTZ.

Siete in grado di fare un oggetto tale che l'utente possa scegliere sia il numero di lati che la misura degli stessi? (ovvero: sapete ricavare la misura del raggio da utilizzare?)

LEZIONE 08

08.01. La rivoluzione informatica

Abbiamo già affrontato la maggior parte degli argomenti, ma certo programmare vuol dire molto di più che conoscere i comandi di un determinato linguaggio. Ci sono cose che si imparano da soli, con la pratica e la curiosità. Ognuno sviluppa un proprio stile di programmazione, ed un buon consiglio è quello di non smettere mai di provare e sperimentare. Ed aprite tutti quegli oggetti che vi incuriosiscono, sbirciate dentro e cercate di capire come funzionano.
In questa disordinata presentazione dei comandi che io giudico più importanti, siamo giunti a quelli più complessi e di uso meno frequente.
Il più necessario, e forse lo stavate aspettando, è quello che consente di realizzare solidi di rotazione: si chiama REVOLVE e questa è la sua sintassi:

REVOLVE n, alpha, mask, x_1, y_1, s_1, ..., x_n, y_n, s_n

La rotazione avviene attorno all'asse X, e i parametri sono i seguenti:
n - numero di nodi della sagoma da far ruotare;
alpha - angolo di rotazione. Con 360 si ottiene una rivoluzione completa;
mask - è un codice binario, che controlla vari aspetti secondari della forma generata;
x, **y** - coordinate dei nodi che definiscono il profilo;
s - un altro codice (status), che stabilisce la modalità di rappresentazione degli spigoli trasversali.

valori di mask:
1 crea la superficie di chiusura alla base (cioè in corrispondenza del primo nodo);
2 crea la superficie di chiusura alla sommità (cioè in corrispondenza dell'ultimo nodo);
4 se alpha è minore di 360, crea una superficie tra l'asse X e la sagoma, in corrispondenza dell'inizio della rotazione;
8 se alpha è minore di 360, crea una superficie tra l'asse X e la sagoma, in corrispondenza del termine della rotazione;
16 la sagoma è visibile, in corrispondenza dell'inizio della rotazione;
32 la sagoma è visibile, in corrispondenza del termine della rotazione;
64 tutti gli spigoli longitudinali, creati dalla rotazione, sono visibili.
La risoluzione dell'elemento è modificabile, anteponendo un'istruzione RESOL, TOLER o RADIUS.

Vediamo come realizzare una semplice lesena semicircolare. Potremmo tracciare prima la sagoma sulla pianta, per determinare le coordinate dei vertici.

La semplice istruzione

```
REVOLVE 5, 180, 1+2+4+8+16+32,
   0.00, 0.30, 0,
   0.10, 0.20, 0,
   1.05, 0.30, 1,
   2.90, 0.20, 0,
   3.00, 0.30, 0
```

posta nel Testo GDL 3D è sufficiente a generare l'oggetto, che però giace sul piano orizzontale. Per orientarlo correttamente occorre ruotare l'asse Y in senso orario (ricordiamo che il verso si determina guardando dall'asse verso l'origine) e quindi si dovrà anteporre il comando ROTY -90. Modificando l'angolo alpha da 180 a 360 si otterrebbe una colonna completa. Utilizzando un parametro, si può lasciare all'utente la possibilità di immettere il valore desiderato per l'angolo.
Come avrete notato, il valore mask può essere scritto anche come espressione, cioè come somma dei singoli valori che lo compongono.
Per il terzo nodo ho utilizzato lo status 1, per non mostrare lo spigolo il corrispondenza dell'entasi (la "spanciatura" della colonna).

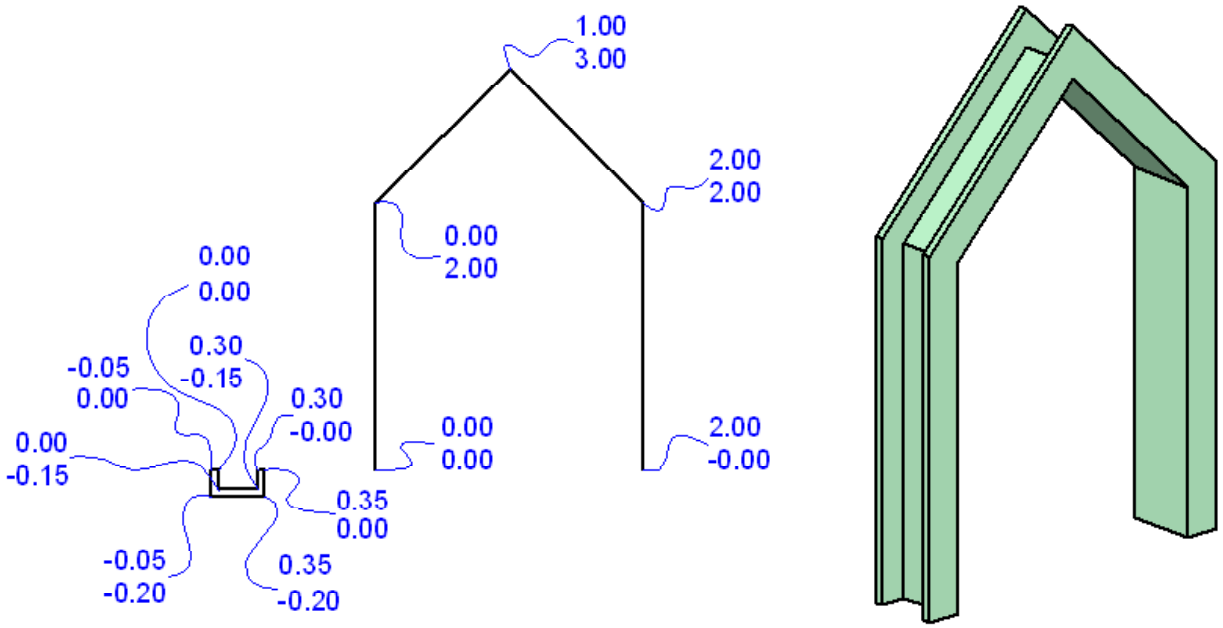
08.02. Un riposino all'ombra

Prima del prossimo comando difficile, mettiamoci in forze con qualcosa di semplice. In qualunque punto di uno script 3D possiamo inserire l'istruzione SHADOW OFF. Tutto quello che verrà realizzato da quel momento in poi, fino ad un eventuale SHADOW ON, non farà ombra. Escludere dal calcolo delle ombre dettagli non significativi (o anche interi oggetti) è un ottimo metodo per velocizzare i rendering. Pensiamo a degli alberi che stanno DIETRO al nostro edificio, ma sono più alti di questo. Vogliamo vederli, ma la loro ombra non ci interessa proprio. Può essere un buona idea, per certi elementi, offrire all'utente la possibilità di scelta per mezzo di un check-box (parametro booleano).
Se lo chiamiamo Ombre, nello script potremo inserire semplicemente:

```
IF Ombre = 0 THEN SHADOW OFF
```

08.03. Quel comando del tubo

Eccoci giunti al piatto forte del giorno. Questo comando richiede molta concentrazione da parte vostra: non è semplice, ma usarlo proficuamente … è ancora più complicato. Ciononostante credo fermamente che debba stare qui, tra i comandi fondamentali. Sto parlando di TUBE che, guarda caso, serve a fare i tubaggi, cioè quelle forme generate da una sagoma che si muove lungo un percorso. Come al solito, per capirci meglio facciamo un esempio: la cornice intorno ad un'apertura è ottenibile con un tubaggio. La sagoma è la forma a "C" della sua sezione, e il percorso è il perimetro dell'apertura.



La sintassi, necessariamente piuttosto articolata, è la seguente:

```
TUBE n, m, mask,
 u1, w1, s1, ..., un, wn, sn,
 x1, y1, z1, angle1, ..., xm, ym, zm, anglem
```

n - numero di nodi del poligono di base;
m - numero di nodi del percorso, più due,
mask - codice binario che controlla vari aspetti secondari del tubaggio:

- 1 = superficie iniziale di chiusura
- 2 = superficie finale di chiusura
- 16 = spigoli iniziali visibili
- 32 = spigoli terminali visibili
- 64 = spigoli di giunzione visibili, ai nodi del percorso;

u, **w** - coordinate x,y (sul piano orizzontale) dei vertici del poligono di base;
s - codice di status dello spigolo (0=sempre visibile, 1=visibile se di bordo);
x, **y**, **z** - coordinate spaziali del percorso di tubaggio;
angle - angolo di rotazione del poligono di sezione.

La sezione del tubaggio viene fatta scorrere ortogonalmente lungo i segmenti del percorso. Le giunzioni sono fatte su un piano posto sulla bisettrice dei segmenti. Gli estremi del tubaggio non sono necessariamente ortogonali. Per questo motivo il percorso contiene due punti extra, detti "punti fantasma", uno

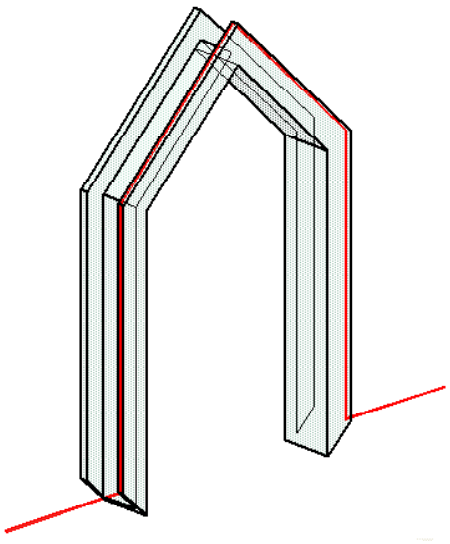
prima del primo punto effettivo del percorso, e uno dopo l'ultimo. Questi sono utilizzati solo per calcolare la bisettrice e stabilire di conseguenza l'orientamento degli estremi. Quello che segue è il comando che esegue la cornice del portale visto nell'esempio precedente.

```
TUBE 8, 5+2, 1+2+16+32,
    0.00, 0.00, 0, ! punto 1 della sagoma
   -0.05, 0.00, 0,
   -0.05,-0.20, 0,
    0.35,-0.20, 0,
    0.35, 0.00, 0,
    0.30, 0.00, 0,
    0.30,-0.15, 0,
    0.00,-0.15, 0, ! punto 8 della sagoma

    0.00, 0.00,-1.00, 0, ! punto "fantasma" iniziale
    0.00, 0.00, 0.00, 0, ! primo punto del percorso
    0.00, 0.00, 2.00, 0,
    1.00, 0.00, 3.00, 0,
    2.00, 0.00, 2.00, 0,
    2.00, 0.00, 0.00, 0, ! ultimo punto del percorso
    2.00, 0.00,-1.00, 0  ! punto "fantasma" finale
```

Modificando i punti fantasma in questo modo:

```
    -1.00, 0.00, 0.00, 0, ! punto "fantasma" iniziale
     3.00, 0.00, 0.00, 0  ! punto "fantasma" finale
```

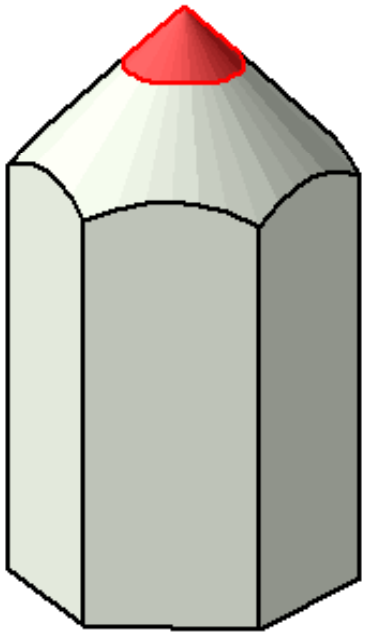
si ottiene un taglio a 45° degli estremi (nella figura è evidenziato lo sviluppo del percorso, compresi i punti fantasma):



08.04. Diamoci un taglio

Abbiamo visto che con il GDL riusciamo a fare un po' tutto. In particolare ci interessa fare cose che ArchiCAD, da solo, non ci consente.
Se volessimo realizzare questo... monumento al lapis ignoto, come dovremmo procedere? Ovviamente il problema è realizzare la parte di raccordo tra la parte prismatica e quella conica. Dobbiamo allora arrenderci e cercare soluzioni di

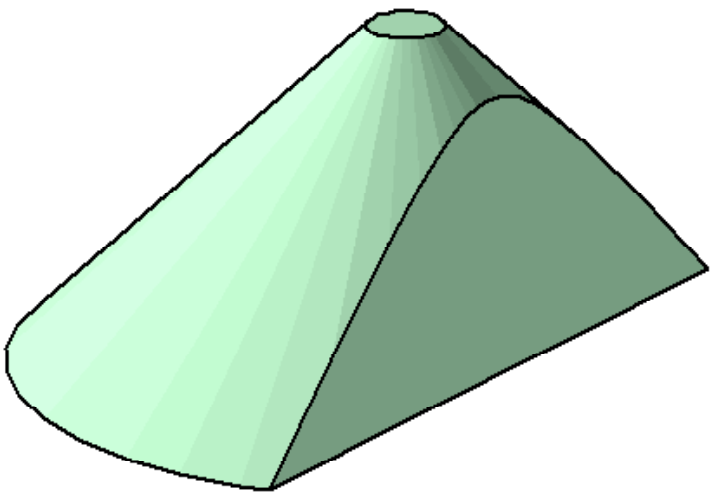
ripiego? Giammai! Affilate i vostri coltelli e seguitemi...



Il comando CUTPLANE permette di definire un piano di taglio infinito. Come dice David Nicholson-Cole nel suo GDL Cookbook, che descrive con grande efficacia questo argomento, dobbiamo immaginarlo come un comando che piazza, in una determinata posizione, la lama circolare di una gigantesca affettatrice. Tutto quello che la attraversa viene tagliato via. Esistono diverse versioni del comando CUTPLANE, ma noi ne useremo una sola, la più semplice e la più efficace, che posiziona la lama sul piano X-Y, in modo che tagli tutto ciò che ha una coordinata Z positiva. Per utilizzarlo dovremo quindi piazzare e orientare convenientemente il cursore 3D.
Facciamo subito una prova:

```
! prova CUTPLANE
ADDX 0.50       ! spostiamo l'origine temporanea
ROTY 90         ! portiamo in verticale il piano di taglio
CUTPLANE        ! lama rotante in azione !
DEL 2           ! ripristiniamo la posizione dell'origine
CONE 1.80, 2.00, 0.20, 90, 90
CUTEND          ! rimuovere la lama dopo l'uso
```

Ed ecco quello che si ottiene:



Proviamo quindi a realizzare un ciclo, per effettuare tutti i tagli necessari.

```
! prova CUTPLANE
FOR f = 1 TO 6     ! ripete per  6 volte...
   ADDX 0.50       !
   ROTY 90         !
   CUTPLANE        ! ...il posizionamento di una lama
   DEL 2           !
   ROTZ 60         !
NEXT f             !
DEL 6
CONE 1.80, 2.00, 0.20, 90, 90
CUTEND : CUTEND : CUTEND
CUTEND : CUTEND : CUTEND
MATERIAL 0
PEN 10
ADDZ 1.80
CONE 0.20, 0.20, 0.00, 90, 90
DEL 1
END
```

Avete notato i sei comandi CUTEND. Ogni lama, quando non serve più, deve essere eliminata. Ciascun comando CUTEND rimuove la lama piazzata più recentemente. Sarebbe stato comodo un comando CUTEND n (analogo a DEL n) per eliminare più lame in una volta sola, ma non c'è. Avete anche notato che ho messo sei comandi in due righe. E' infatti possibile, nel GDL, usare una riga per più comandi. Basta usare il simbolo ":" (due-punti) per separarli. Di solito questo nuoce alla leggibilità dello script, quindi è meglio non abusarne.
Nel nostro caso sappiamo di avere 6 lame in azione (la matita è esagonale) e quindi sappiamo quanti CUTEND ci servono, ma se volessimo dare all'utente la possibilità di scegliere la forma con un parametro? Ancora una volta la soluzione più funzionale la suggerisce Nicholson-Cole. Facciamo l'esempio di una variabile "lati" che definisca il numero di facce della nostra matita (create un parametro di tipo Intero).

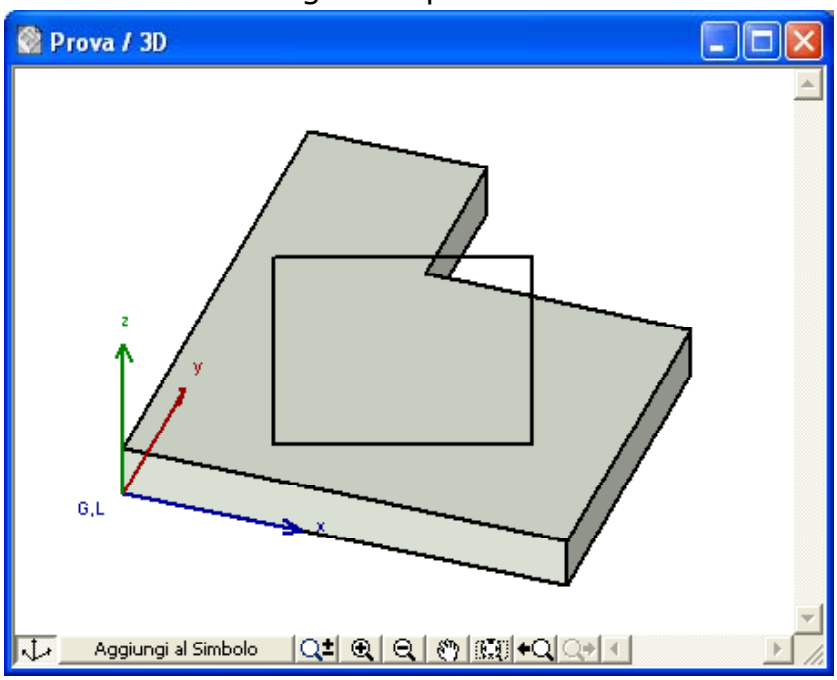
```
! Prova CUTPLANE
kut = 0                    ! prepariamo un contatore di tagli
PEN 2
FOR f = 1 TO lati
   ADDX 0.50
   ROTY 90
   CUTPLANE : kut = kut+1  !contiamo le lame, mentre le
 posizioniamo
   DEL 2
   ROTZ 360 / lati
NEXT f
DEL lati
CONE 1.80, 2.00, 0.20, 90, 90
FOR k = 1 TO kut        ! usiamo il contatore di tagli
   CUTEND                ! per eseguire altrettanti CUTEND
NEXT k
MATERIAL 0
```

```
PEN 10
ADDZ 1.80
CONE 0.20, 0.20, 0.00, 90, 90
DEL 1
END
```

In questo modo, utilizzando una variabile di servizio per contare i comandi CUTPLANE, possiamo impostare un ciclo che esegue l'esatto numero di CUTEND necessari a eliminare tutte le lame.
CUTPLANE ha una caratteristica, che può risultare fastidiosa: la superficie che viene esposta a seguito del taglio utilizza il materiale corrente e, in mancanza, viene sempre colorata con la penna corrente. Occorrerà quindi, se necessario, impostare il parametro PEN prima di posizionare la lama. Per ottenere due tagli di colore diverso, anche se complanari, occorrono due lame distinte. Il colore della penna, nelle viste ombreggiate e nei rendering, sarà visibile solo se il materiale corrente è "GENERALE" (lo si può imporre, nel GDL, scrivendo MATERIAL 0), altrimenti il comando CUTPLANE utilizzerà per la superficie il materiale corrente.
Per vedere l'effetto, nel listato precedente, provate a mettere PEN f, tra ROTY 90 e CUTPLANE, oppure usate MATERIAL f*3.

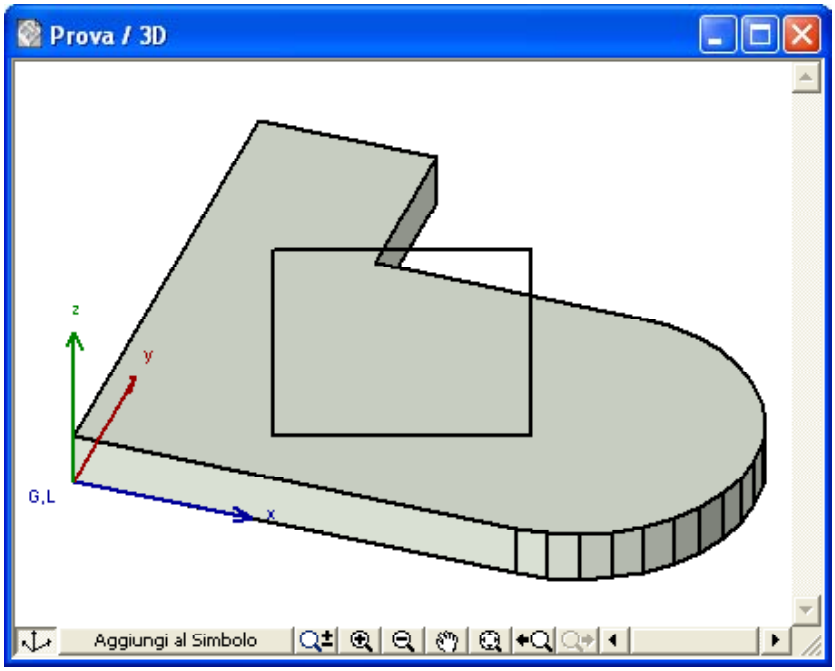
08.05 Nuovi codici in arrivo

Presentando i comandi PRISM_ e CPRISM_ abbiamo detto che si tratta praticamente di un solaio, del tutto analogo a quello ottenibile con il corrispondente strumento di ArchiCAD, compresa la definizione di fori e di materiali diversi per le superfici. Ma c'è qualcos'altro che possiamo fare con un solaio… definire lati curvi.
Come è possibile ottenere un lato curvo? A questo scopo sono stati creati dei valori "speciali" di status, che possono essere aggiunti ai valori normali.
Ne esistono diversi, descritti un po' cripticamente nel manuale (sono chiamati "codici di status addizionali"). Noi vedremo solo quelli più utili.
Consideriamo il seguente prisma:



```
PRISM_ 7, 0.30,
   0.00, 0.00, 15,   ! nodo 1
   2.50, 0.00, 15,
   2.50, 2.00, 15,   ! nodo 3
   1.00, 2.00, 15,
   1.00, 3.00, 15,
   0.00, 3.00, 15,   ! nodo 6
   0.00, 0.00, -1    ! chiusura
```

possiamo trasformarlo nel seguente, semplicemente aggiungendo il codice 1000 al valore status del terzo nodo.




```
   2.50, 2.00, 15+1000   ! nodo 3
```

Questo codice dice, in pratica, "*arriva fino a questo punto, realizzando un raccordo circolare*".
(La le linee verticali di segmentazione possono essere nascoste aggiungendo 64 al valore di status del nodo 2, come abbiamo visto nella lezione 07.01)

Il **1000** è il più semplice e il più utile; altri codici che creano segmenti curvi sono 2000 e 4000.

2000 - in questo caso non si devono indicare le coordinate x e y del nodo di arrivo, ma il raggio e l'ampiezza angolare dell'arco.

```
   1.00, 180, 15+2000   ! nodo 3
```

4000 - questo realizza una curva dato l'angolo di sviluppo e il centro. L'angolo, come nel caso del codice 2000, deve essere messo al posto della coordinata y. Per il centro occorrono ancora due coordinate (x e y)... Ecco che si ha la necessità di indicare un nuovo punto. Questo avrà codice **900**; deve essere posto prima del codice 4000, e dovrete tenerne conto nel numero di punti del comando PRISM, aumentando di uno il valore di **n**. Il primo valore della riga con codice 4000 non viene utilizzato, e solitamente si mette zero.

```
   2.50, 1.00, 900       ! nodo 3 - centro
   0.00, 180,  15+4000   ! nodo 4
```

Oltre a questi, utilizzati per realizzare curve, si possono trovare altri codici utili:
100 - fa sì che le coordinate x-y siano considerate non come assolute, ma rela-

tive al nodo precedente.
200 - anche questo codice permette di utilizzare coordinate relative, ma polari anziché cartesiane. Si usano lunghezza e direzione al posto dei valori x e y.
700 - chiude il perimetro, tornando al punto iniziale.

I codici addizionali possono essere utilizzati aggiungendoli al valore di status di numerosi comandi.

08.06. Qualche prisma perde la retta via

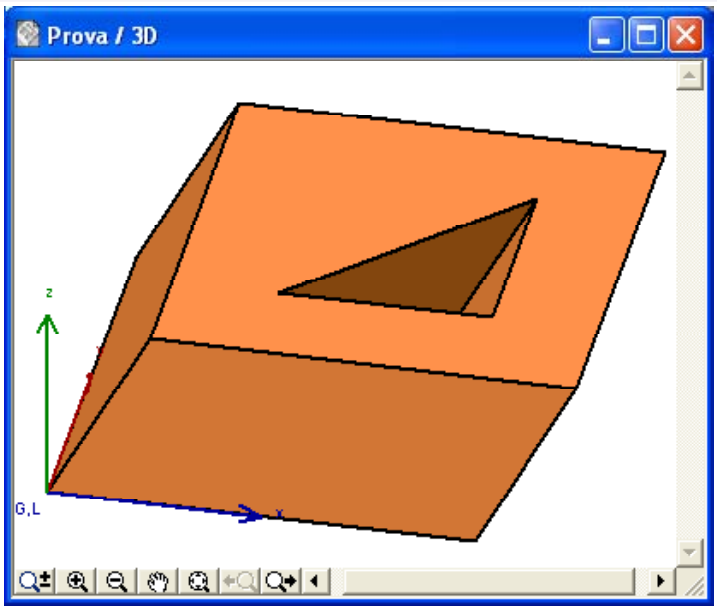
Con questo comando si può definire un un prisma in modo analogo a quanto si fa con il comando PRISM_, ma si ha la possibilità di indicare uno spostamento della faccia superiore rispetto a quella inferiore.
La sintassi è

```
EXTRUDE n, dx, dy, dz, mask, x1, y1, s1, ..., xn, yn, sn
```

n - numero di nodi del perimetro.
dx, dy, dz - scostamento lungo i tre assi della superficie superiore rispetto a quella inferiore; dz rappresenta l'altezza.
mask - un codice relativo alle superfici orizzontali e ai loro spigoli.
1 = superficie inferiore esistente
2 = superficie superiore esistente
4 = superficie di chiusura esistente (se il perimetro di base non è già chiuso)
16 = linee perimetrali inferiori visibili
32 = linee perimetrali superiori visibili
x, y - coordinate dei vertici.
s - un altro codice:
0 = spigolo laterale visibile
1 = spigolo laterale non visibile (a meno che sia di "bordo", dal punto di vista corrente)
-1 = fine perimetro. Permette di definire fori nella forma da estrudere.

Vediamo un semplice esempio:

```
EXTRUDE 9, 0.50, -0.10, 1.00, 63,
    0.00, 0.00, 0,    ! \
    2.00, 0.00, 0,      ! perim.
    2.00, 2.00, 0,      ! esterno
    0.00, 2.00, 0,      !
    0.00, 0.00, -1,   ! /

    0.50, 0.50, 0,    ! \
    1.50, 0.50, 0,      ! perim.
    1.50, 1.50, 0,      ! foro
    0.50, 0.50, -1    ! /
```



In qualche modo analogo ad EXTRUDE, il comando PYRAMID permette di realizzare una sorta di estrusione verso un punto.

PYRAMID n, h, mask, x_1, y_1, s_1, ..., x_n, y_n, s_n

mask - un codice relativo alla visualizzazione di alcuni elementi:

1 = superficie inferiore esistente

4 = superficie laterale di chiusura esistente (se il perimetro di base non è già chiuso)

16 = linee perimetrali inferiori visibili

x, y - coordinate dei vertici.

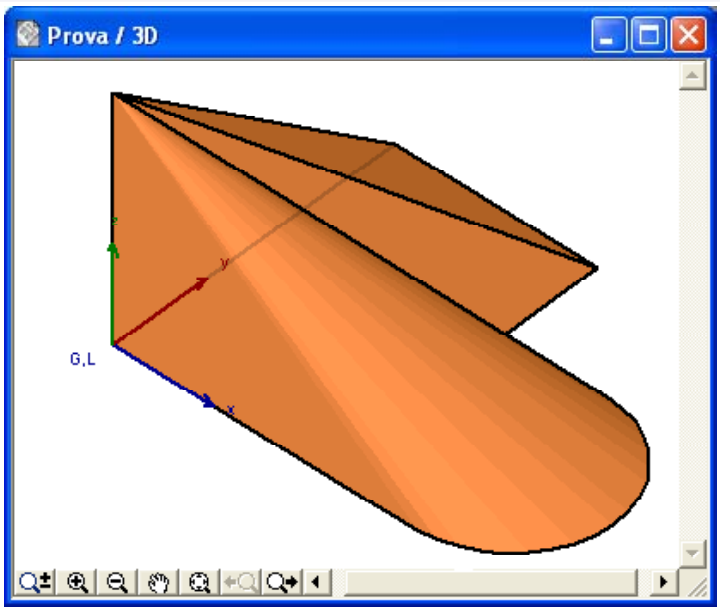
s - un altro codice:

0 = spigolo laterale visibile

1 = spigolo laterale non visibile (a meno che sia di "bordo", dal punto di vista corrente)

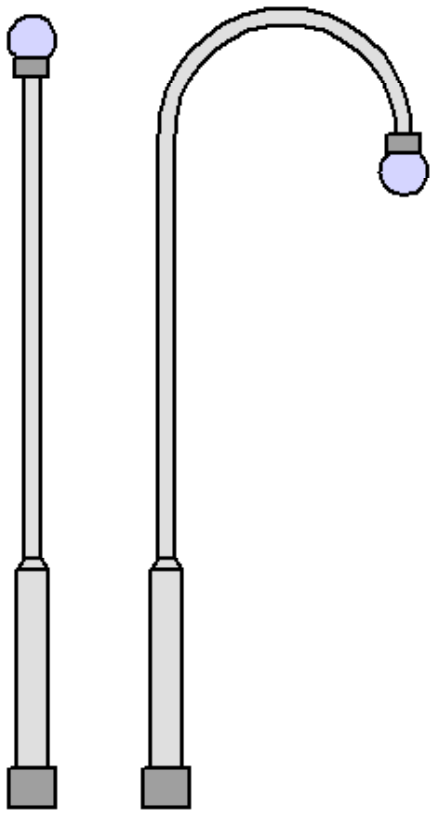
A differenza di EXTRUDE, il comando PYRAMID non ammette fori.
La particolarità è che il vertice è sempre posto alle coordinate x=0.00; y=0.00, e può trovarsi anche esternamente, rispetto alla base.

```
PYRAMID 6, 2.50, 21,
   0.00, 3.00, 0,
   0.00, 0.00, 0,
   3.00, 0.00, 1,
   3.00, 2.00, 1+1000,
   2.00, 2.00, 0,
     2.00, 3.00, 0
```



Esercizio G

Dimostrate di saperci fare, e mettete insieme un oggetto che, a scelta dell'utente, realizzi un lampione a stelo dritto, o curvo.
Usate la variabile **B** per il diametro della sfera e anche del basamento. Nel caso di lampione arcuato adoperate la variabile **A** per definire la sporgenza.

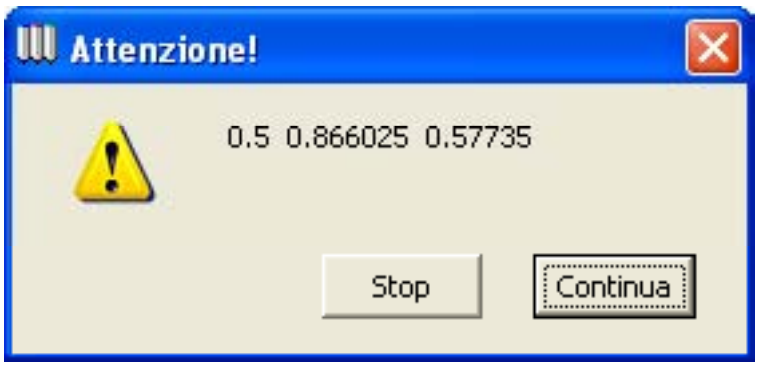


LEZIONE 09

09.01. Avvisi ai naviganti

Cominciamo con un comando che non serve a FARE elementi 2D o 3D, ma aiuta in molte circostanze: PRINT p1 [,p2 ...]
Durante l'esecuzione dello script in cui si trova (2D o 3D), genera una *finestra di avviso* nella quale compare il parametro (o i parametri) del comando.
Può avere vari utilizzi:

- Scrivete PRINT SIN(30), COS(30), TAN(30) nello script 3D, e attivate la finestra Vista 3D. Apparirà subito:



  Ecco che, mentre programmiamo in GDL, abbiamo a disposizione una semplice calcolatrice scientifica a portata di mano!
- Ricerca di errori: durante la realizzazione di oggetti complessi capita spesso di non ottenere subito i risultati attesi. Un buon metodo per capire cosa non va può essere quello di monitorare –in uno o più punti del programma– il valore di alcune variabili. Es: PRINT "spessore:", spe, "raggio:", r1. Questo produce la scritta delle parole "spessore" e "raggio" seguite ciascuna dal valore della variabile indicata (spe e r1).
- Messaggi per l'utente: possiamo segnalare che i valori immessi per qualche parametro non sono corretti. Es: IF lati < 3 THEN PRINT "Attenzione: il poligono non può avere meno di 3 lati".

Il comando PRINT gestisce numeri, variabili, espressioni e stringhe. Diamo una rapida occhiata alla gestione delle stringhe, utile anche con TEXT e TEXT2.

AZIONE	RISULTATO	OSSERVAZIONI
nom="Pinco" cog="Pallino" PRINT nom, cog	PincoPallino	se vogliamo uno spazio, occorre dirlo
PRINT nom, " ", cog	Pinco Pallino	
PRINT nom, "\n", cog	Pinco Pallino	**\n** è un codice speciale, che inserisce un "accapo". Ne esistono altri, come **\t** per la tabulazione
tizio = nom + " " + cog PRINT tizio	Pinco Pallino	il segno + si può usare con le stringhe. Questa operazione si chiama concatenazione.

xx=1 yy=2 PRINT "*", xx, yy, "*" ww=-1 zz=-2 PRINT "*", ww, zz, "*"	* 1 2*	davanti ai numeri c'è sempre lo spazio per il segno, usato solo nel caso di numeri negativi
PRINT 1/3	0,333333	per i decimali vengono visualizzate fino a sei cifre
alt=3.15 num=17 aa=alt/num pp=0.28 PRINT "alzata:", alt/num, "\n", "rapporto:", 2*aa+pp	alzata: 0,185294 rapporto: 0,650588	Per un avviso, può andare bene, ma se usiamo ad esempio TEXT2 per scrivere questi valori, forse vorremmo meno decimali...

Per gestire il testo... ci sono ovviamente delle funzioni apposite.

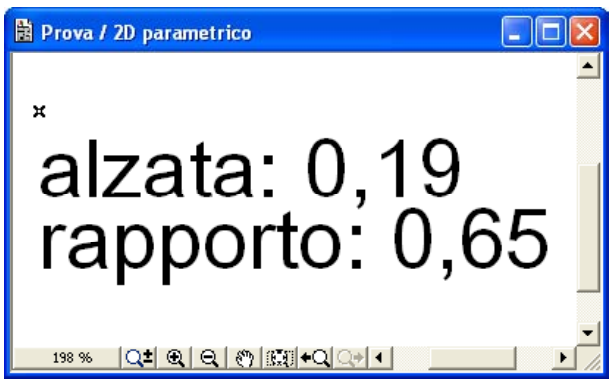
09.02. Torniamo all'ABC

Per risolvere il problema precedente è necessario imparare ad utilizzare le stringhe di testo. La funzione più importante è forse STR(). Ne esistono due versioni:

```
STR (num, lun, dec)
STR (formatstring, num)
```

Entrambe generano una stringa di testo a partire dal numero **num**. La prima permette di definire la lunghezza totale minima (**lun**) e il numero di decimali (**dec**); la seconda utilizza una stringa di formato, in realtà un po' complessa, che permette di definire l'allineamento, l'unità di misura, la precisione e la presenza o meno dello zero intero e del segno '+' per i numeri positivi. Se pensate di averne bisogno, utilizzate il manuale GDL per trovare i codici di formato necessari. Qui vediamo invece un esempio di utilizzo della prima forma:

```
alt = 3.15
num = 17
aa  = alt/num
pp  = 0.28
testo1 = "alzata: "   + STR(3.15/17, 4, 2)
testo2 = "rapporto: " + STR(2*aa+pp, 4, 2)
TEXT2 0.00, 0.00, testo1
TEXT2 0.00,-0.60, testo2
```



Il parametro **lun** definisce il numero di caratteri minimo della stringa restituita. Questo valore comprende i decimali ed il separatore (punto o virgola). Se necessario verranno aggiunti spazi a sinistra del numero.

Ci sono altre *funzioni stringa* da conoscere. Possono essere complesse da utilizzare, e non è il caso di studiarle finché non se ne senta veramente la necessità. L'importante è sapere che esistono, e che il manuale è pronto a fornire la sintassi e la descrizione formale.

SPLIT - permette di estrarre da una stringa di testo sia numeri sia sottostringhe.
STRLEN - fornisce il numero di caratteri di cui è formata una stringa.
STRSTR - fornisce la posizione di una sottostringa, all'interno di una stringa.
STRSUB - estrae una sottostringa da una stringa
STW - fornisce la lunghezza in millimetri di una stringa, o meglio la lunghezza reale di stampa. Il risultato dipende dallo STYLE corrente, che definisce la dimensione del carattere in mm.

Esempi:

```
nn=SPLIT("marmo 1.25x0.30", "%s %nx%n", mat, xx, yy )
```

a seguito di questo comando, le variabili **mat**, **xx** e **yy** conterranno i valori "marmo", 1.25 e 0.30.

```
nn = STRLEN("conta caratteri")
```

nn conterrà il valore 15 (numero di caratteri, compreso lo spazio)

```
testo1 = "stringa di partenza"
nn = STRSTR(testo1, "parte")
```

nn conterrà il valore 12 (la stringa "parte" inizia al 12° carattere della stringa "stringa di partenza")

```
testo2 = "estrazione di sottostringa"
tt = STRSUB(testo2, 4, 7)
```

tt conterrà la stringa "razione" (7 caratteri della stringa "estrazione di sottostringa", a partire dal 4°)

```
DEFINE STYLE "stile1" "Arial", 5, 7, 0
SET STYLE "stile1"
testo = "quanto è lungo?"
TEXT2 0.00, 0.00, testo
xx = STW(testo)
LINE2 0.00, 0.00, xx/1000*a_, 0.00
```

La variabile **xx** contiene la lunghezza della stringa **testo**, permettendomi, in questo caso, di realizzare una linea della stessa lunghezza. L'operazione **/1000*a_** esegue un adattamento alla scala corrente del disegno. (a_, oppure Glob_Scale, è una variabile globale di ArchiCAD che memorizza la scala corrente).

09.03. Non tutto il 3D è solido

È giunta l'ora che prendiate di nuovo il Manuale GDL e studiate un po' di comandi da soli. Sono comandi 3D che generano forme piane, come ad esempio un cerchio o un rettangolo. Quando ne ricorrono le circostanze, il loro uso è da preferire rispetto ai comandi 3D classici, perché il modello risulta formato da un numero molto minore di poligoni, quindi le richieste di memoria ed i tempi di calcolo ne vengono influenzati in maniera significativa.
I comandi sono i seguenti:

HOTSPOT - inserisce un hotspot. Nel 3D, di solito, sono visibili tutti gli hotspot del simbolo 2D, posti alla quota di posizionamento dell'oggetto. Se si usa il parametro zzyzx, si avrà anche una seconda serie di hotspot, all'altezza

definita da questa variabile. Con il comando HOTSPOT si possono aggiungere nodi, che diverranno punti di snap riconosciuti dal cursore nella finestra 3D e nelle Sezioni/Alzati.

LIN_ - realizza una linea 3D. Le linee sono visibili solo nelle viste "a linee nascoste", oppure in "Ombreggia con contorno", purché, in Settaggi Finestra 3D, sia impostata la qualità migliore e il calcolo analitico. Nei rendering non sono mai visibili.

RECT - un semplice rettangolo; è analogo ad un BLOCK privo di spessore

POLY - un poligono piano di n lati; è analogo ad un PRISM privo di spessore

POLY_ - come il precedente, ma con codici di status per omettere lati e realizzare fori.

CIRCLE - un semplice cerchio; analogo ad un CYLIND privo di spessore.

ARC- una linea curva, con le stesse limitazioni di LIN_

PLANE - un poligono inclinato (non verticale); è analogo ad uno SLAB privo di spessore.

PLANE_ - come il precedente, ma con codici di status per omettere lati e realizzare fori.

09.04. Strane forme all'orizzonte

Esiste, oltre alle forme 3D viste finora, anche una serie di solidi e superfici più complesse. Le vedremo rapidamente, senza soffermarci troppo. Il loro uso non è frequente, e se ne avrete bisogno dovrete ricorrere al manuale e... a molta pazienza.

RULED crea una superficie di raccordo tra un perimetro orizzontale, posto a quota 0.00, ed un altro avente lo stesso numero di punti, i quali possono trovarsi a quote differenti.

RULED n, mask, u_1, v_1, s_1, ..., u_n, v_n, s_n,
x_1, y_1, z_1, ..., x_n, y_n, z_n

n - numero di nodi delle basi

mask - codice per la generazione di superfici e spigoli:

- 1 = superficie di base presente
- 2 = superficie superiore presente (solo se il perimetro superiore è piano)
- 4 = superficie laterale di chiusura presente (due triangoli, se non è piana)
- 16 = spigoli inferiori visibili
- 32 = spigoli superiori visibili
- 64 = spigoli delle triangolazioni laterali visibili

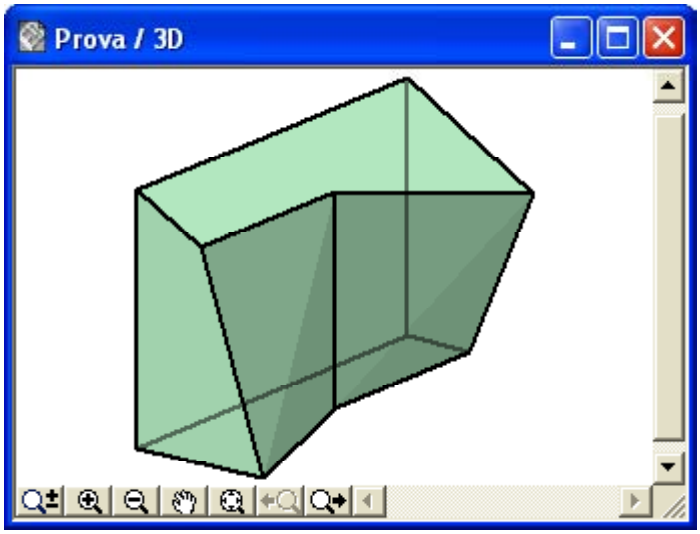
u, v - coordinate x,y del perimetro di base

s - status degli spigoli laterali (0=visibili comunque, 1=visibili solo se formano il contorno)

x, y, z - coordinate spaziali del secondo perimetro

```
RULED 5, 55,
   0.00, 0.00, 0,
   2.00, 0.00, 0,
   1.00, 2.00, 0,
   1.00, 4.00, 0,
   0.00, 4.00, 0,

   0.00, 0.00, 3.00,
```

```
    1.00, 0.00, 2.50,
    1.00, 2.00, 2.50,
    2.00, 4.00, 2.00,
    0.00, 4.00, 3.00
```



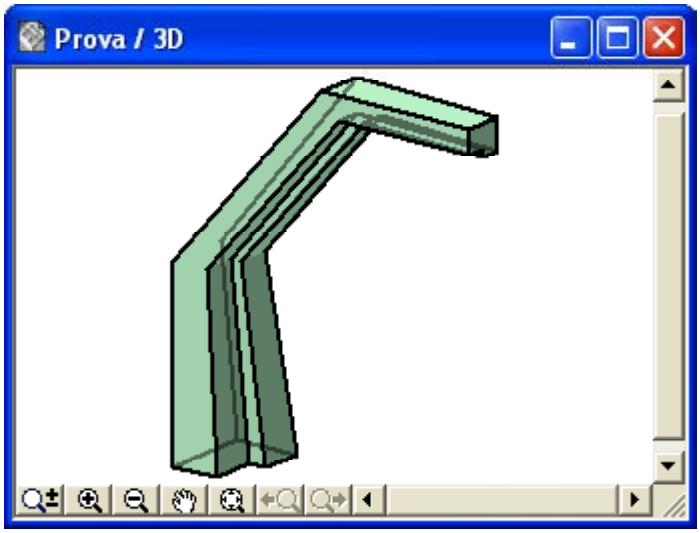
SWEEP crea una superficie generata da un perimetro di base che scorre lungo un percorso spaziale. È una variante del comando TUBE, che permette di scalare e ruotare la sagoma, nel corso dello sviluppo.

```
SWEEP n, m, alfa, scala, mask,
   u1, v1, s1, ..., un, vn, sn,
   x1, y1, z1, ..., xm, ym, zm
```

n - numero di nodi della sagoma di base
m - numero di nodi del percorso
alfa - angolo di rotazione della sagoma, per ogni segmento del percorso
scala - fattore di scala apportato alla sagoma, per ogni segmento del percorso
mask - codice per la generazione di superfici e spigoli:
    1 = superficie di base presente
    2 = superficie superiore presente
    4 = superficie laterale di chiusura presente
    16 = spigoli inferiori visibili
    32 = spigoli superiori visibili
    64 = spigoli delle triangolazioni laterali visibili
u, v - coordinate x,y della sagoma di base
s - status degli spigoli laterali (0=visibili comunque, 1=visibili solo se formano il contorno)
x, y, z - coordinate spaziali del percorso

```
SWEEP 6, 4, 0, 0.75, 1+2+16+32,
    0.00, 0.00, 0,
    0.30, 0.00, 0,
    0.30, 0.20, 0,
    0.40, 0.20, 0,
    0.40, 0.40, 0,
    0.00, 0.40, 0,

    0.00, 0.00, 0.00,
```

```
    0.00, 0.00, 1.00,
    1.00, 0.00, 2.00,
    2.00, 0.00, 2.00
```

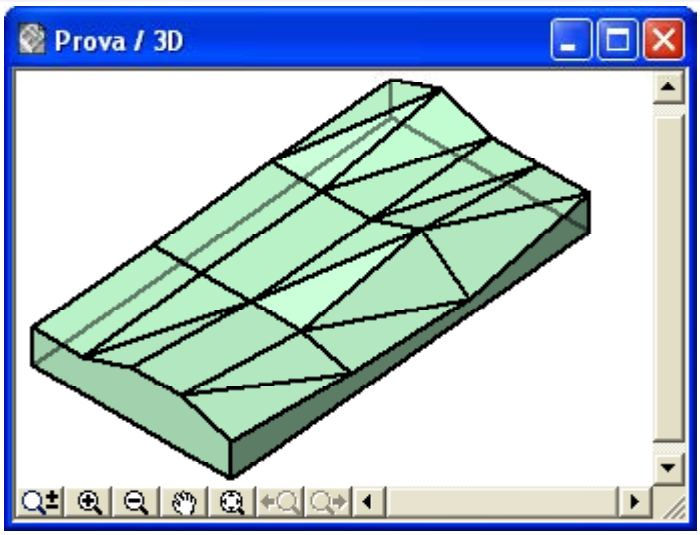


MESH permette di creare una superficie a pianta rettangolare, suddivisa da un numero arbitrario di linee parallele ai lati, disposte a distanze regolari. I parametri del comando consentono di attribuire la quota ad ogni nodo di tale maglia.

```
MESH x, y, m, n, mask,
   z1-1, z1-2, ..., z1-m,
   z2-1, z2-2, ..., z2-m,
   ...
   zn-1, zn-2, ..., zn-m
```

x,y - dimensioni del rettangolo di base. Un angolo della maglia è posto nella punto x=0.00, y=0.00.
m - numero di nodi lungo l'asse x
n - numero di nodi lungo l'asse y
mask - codice per la generazione di superfici e spigoli:
1 = superficie di base presente (un piano a quota z=0.00)
4 = superfici laterali presenti
16 = spigoli di base e verticali visibili
32 = spigoli della maglia visibili
64 = spigoli della maglia visibili, e non levigati nei rendering
z - quota di ciascun nodo.

```
MESH 6, 10, 5, 4, 63,
  1.00, 1.00, 1.50, 1.50, 1.00,
  1.00, 1.00, 1.00, 1.00, 0.60,
  1.00, 1.00, 1.00, 1.50, 0.40,
  1.00, 1.50, 1.00, 1.00, 1.00
```



COONS è una superficie interpolata su 4 polilinee spaziali di perimetro. I lati opposti del "quadrilatero" devono avere un ugual numero di nodi.

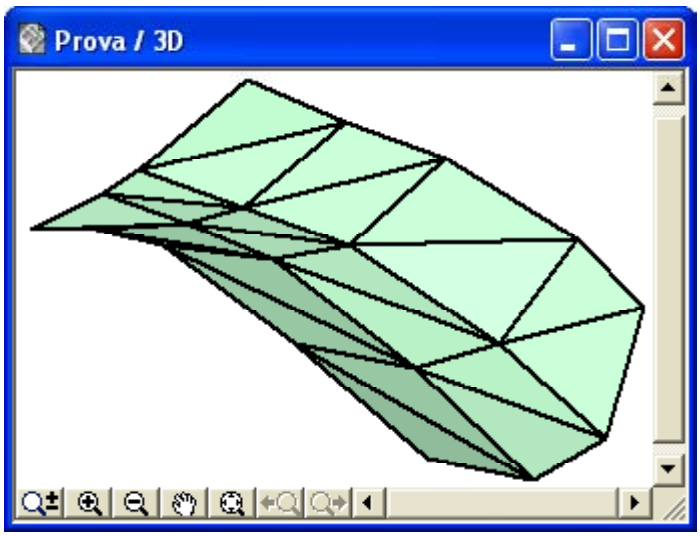
```
COONS n, m, mask,
 x1₁, y1₁, z1₁, ..., x1ₙ, y1ₙ, z1ₙ,
 x2₁, y2₁, z2₁, ..., x2ₙ, y2ₙ, z2ₙ,
 x3₁, y3₁, z3₁, ..., x3ₘ, y3ₘ, z3ₘ,
 x4₁, y4₁, z4₁, ..., x4ₘ, y4ₘ, z4ₘ
```

n - numero di nodi delle prime due polilinee
m - numero di nodi delle altre due polilinee
mask - codice per la generazione degli spigoli:
- 4 = spigoli della prima polilinea visibili
- 8 = spigoli della seconda polilinea visibili (è il lato opposto al precedente)
- 16 = spigoli della terza polilinea visibili
- 32 = spigoli della quarta polilinea visibili (è il lato opposto al precedente)
- 64 = spigoli di triangolazione della superficie visibili

x, y, z - coordinate spaziali delle polilinee.
Le quattro polilinee devono formare un poligono spaziale chiuso. Ne consegue che gli estremi di ogni polilinea coincidono con quelli delle due polilinee contigue, e questi punti coincidenti devono essere inseriti due volte nella sintassi, come quelli che ho evidenziato con l'asterisco (estremo finale della prima polilinea e estremo iniziale della quarta).

```
COONS 4, 5, 63+64,
  0.00, 0.00, 1.00,
  0.00, 1.00, 0.80,
  0.00, 1.50, 0.80,
  0.00, 3.00, 1.00, !*

  6.00, 0.00, 1.00,
  6.50, 1.00, 0.30,
  6.50, 2.00, 0.20,
  6.00, 3.00, 1.00,

  0.00, 0.00, 1.00,
  1.00, 0.00, 1.50,
```

```
  2.00, 0.00, 1.80,
  4.00, 0.00, 1.50,
  6.00, 0.00, 1.00,

  0.00, 3.00, 1.00, !*
  1.50, 3.00, 1.20,
  3.00, 3.00, 1.50,
  5.00, 3.00, 1.40,
  6.00, 3.00, 1.00
```



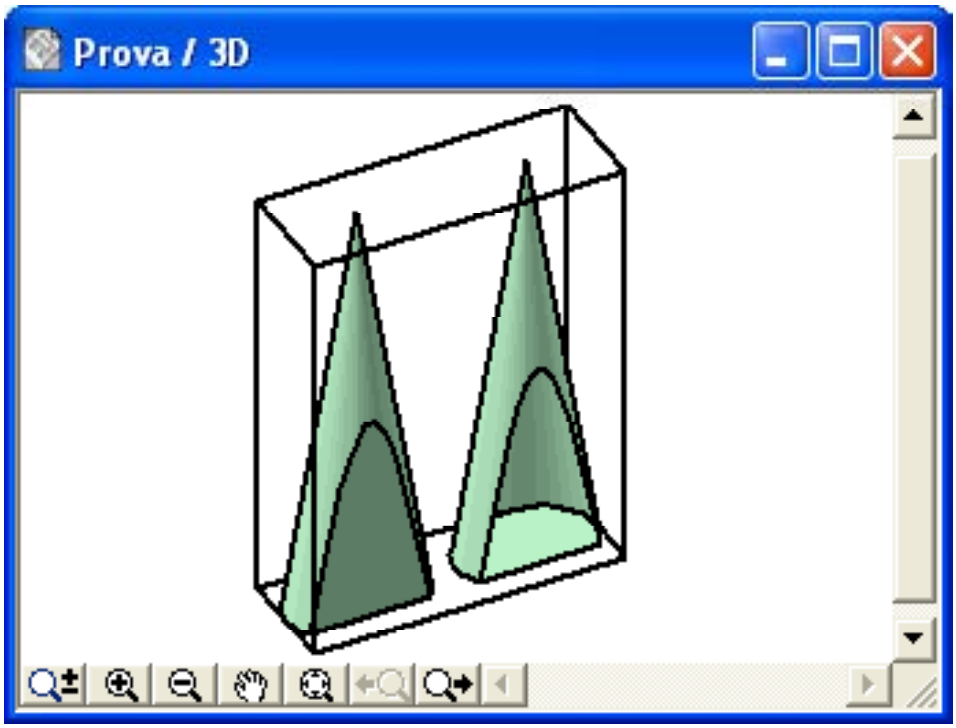
09.05. Pieno o vuoto?

Quando si realizzano dei corpi solidi in GDL, ad esempio BLOCK, CYLIND, SPHERE, ecc. questi sono considerati "pieni", nel senso che, se vengono sezionati, in corrispondenza del taglio vedremo una superficie, e non l'interno del solido.
Abbiamo comunque un'istruzione apposta che permette di definire la modalità di costruzione del modello.
Il comando, o meglio la direttiva, si chiama MODEL, e può essere seguita da una delle seguenti opzioni: SOLID, SURFACE, WIRE.
MODEL SOLID - è la situazione di default, e realizza corpi solidi.
MODEL SURFACE - impone la modellazione per superfici.
MODEL WIRE - impone una modellazione a filo di ferro. Saranno visibili solo gli spigoli dei solidi.

La modellazione prosegue con la modalità impostata fino alla successiva direttiva MODEL, o fino alla fine dello script. La modellazione per superfici è più onerosa, in termini di tempo di calcolo, perché il programma deve prendere in considerazione anche la parte interna dei solidi.

```
MODEL WIRE
BLOCK 1.00, 2.20, 3.00
BODY -1

  ROTy 90
  ADDz 0.75
CUTPLANE
  DEL 2
```

```
MODEL SOLID
  ADD 0.50, 0.50, 0.00
CONE 3.00, 0.50, 0.00, 90, 90
BODY -1
  DEL 1

MODEL SURFACE
  ADDy 1.20
  ADD 0.50, 0.50, 0.00
CONE 3.00, 0.50, 0.00, 90, 90
BODY -1
  DEL 2

CUTEND
```



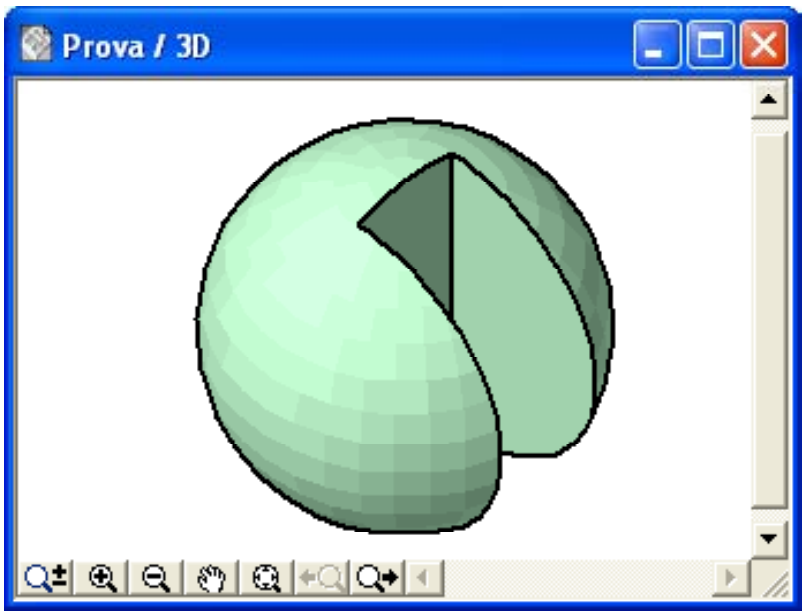
Avete notato le istruzioni BODY -1 che seguono la generazione di ogni solido? Si tratta di un'istruzione che costringe l'interprete GDL a considerare conclusa la definizione di un solido. È nata per altri scopi, ma torna utile spesso anche quando si definiscono solidi non corretti, aperti, sezionati o con fori, ed il risultato ottenuto non corrisponde a quello atteso.

Parlando di tagli, può essere il caso di accennare anche ad un altro comando che, a differenza di CUTPLANE, effettua un taglio di forma poligonale.

```
CUTPOLY n, x1, y1, ..., xn, yn
```

n - numero di nodi del poligono di taglio
x, y - coordinate dei nodi.
Il poligono non può contenere fori né intersecarsi, ed effettua un taglio di lunghezza infinita lungo l'asse z.
Verranno assoggettati al taglio tutti i solidi definiti successivamente, fino al comando CUTEND, che è obbligatorio come nel caso di CUTPLANE.

```
CUTPOLY 4,
  0.00, -0.10,
  1.00, -0.10,
  1.00,  0.10,
  0.00,  0.10

SPHERE .3

CUTEND
```



Una variante, chiamata CUTPOLYA, permette di utilizzare dei codici di status e mask per definire la visibilità degli spigoli e delle superfici generati dal taglio. Del tutto analogo a CUTPOLYA è il comando WALLHOLE, che può essere usato quando si realizzano porte e finestre. La presenza di uno o più comandi WALLHOLE nello script permette di definire tagli personalizzati nel muro che ospiterà l'infisso. Se non vi sono comandi WALLHOLE, ArchiCAD realizza sempre un'apertura pari alle dimensioni A e B dell'oggetto Porta/Finestra.

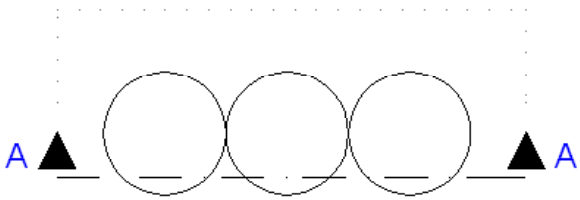
E giacché stiamo parlando di tagli, diciamo che esiste anche l'istruzione SECT_FILL, che permette di dichiarare il retino da utilizzare nella finestra Sezione/Alzato, se l'oggetto viene sezionato. La direttiva rimane valida per tutti gli elementi 3D creati successivamente, fino al prossimo comando SECT_FILL, o fino al termine dello script.

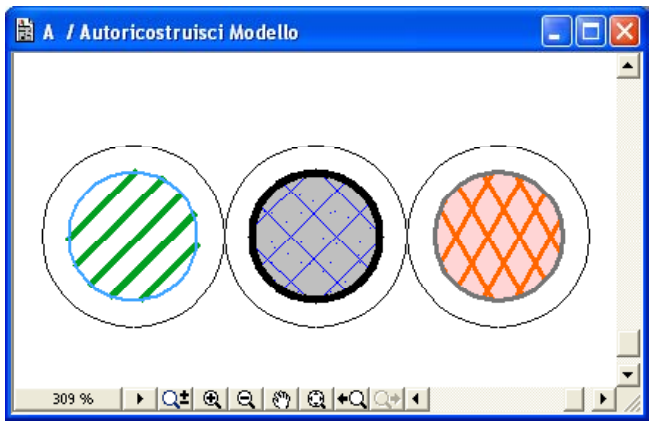
```
SECT_FILL fill, spen, rpen, cpen
```

fill - retino da utilizzare. Può essere un numero (indice) un testo tra virgolette (nome) o, più spesso, un parametro utente di tipo retino.
spen - penna di sfondo del retino
rpen - penna della campitura del retino
cpen - penna di contorno del retino.
Il retino e le penne qui definiti verranno utilizzati solo se nella finestra settaggi dell'oggetto, nel pannello Sezione, è attiva la casella "Usa Attributi Sezione del Simbolo".

```
SECT_FILL "mattoni", 91, 4, 6
```

```
SPHERE 1.00

  ADDX 2.00
SECT_FILL 10, 93, 7, 1
SPHERE 1.00

  ADDX 2.00
SECT_FILL retino, 20, 3, 2 ! "retino" è un parametro utente
SPHERE 1.00
```





09.06 Scripta manent

Il comando TEXT2 permette di aggiungere scritte al simbolo 2D degli oggetti. Non è possibile usare del normale testo nelle finestre 3D, ma il GDL permette di creare dei solidi tratti dai caratteri presenti nel sistema, grazie al comando TEXT. Anche in questo caso occorre utilizzare preventivamente i comandi DEFINE STYLE e SET STYLE, per definire le caratteristiche del testo.

```
TEXT spess, 0, testo
```

spess - spessore di estrusione delle lettere.
0 - zero. È un parametro che non viene utilizzato.
testo - la stringa o il valore numerico da riprodurre. Può essere anche un parametro utente.

A differenza di quanto accade nel 2D, l'altezza del testo 3D non è riferita alla dimensione di stampa ma, per questioni di compatibilità, è comunque espressa in millimetri, nel comando DEFINE STYLE.
Per conoscere la lunghezza di una stringa 3D, è quindi necessario utilizzare la formula STW(testo)/1000.

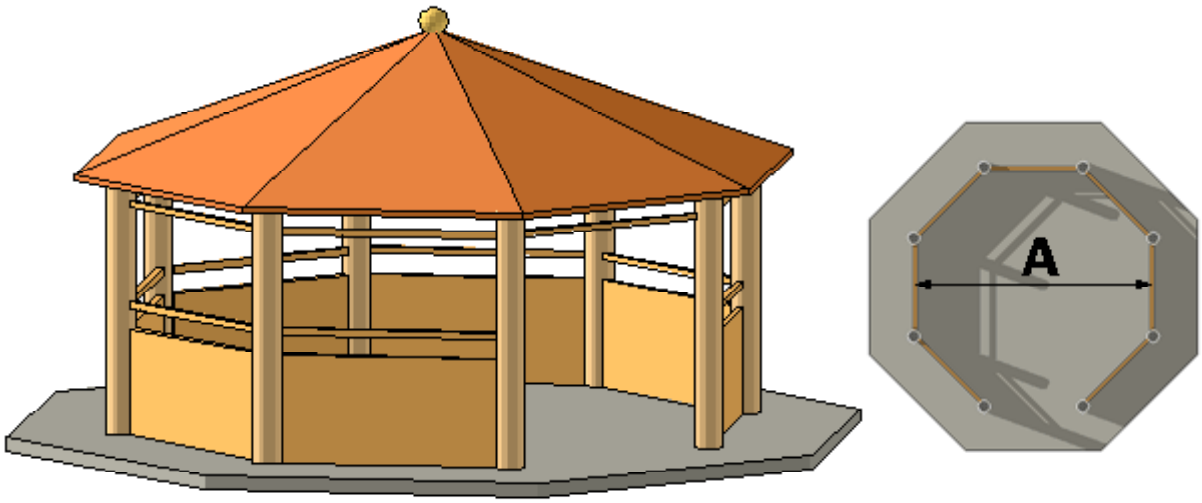
```
DEFINE STYLE "st1" "Garamond", 500, 7, 0
SET STYLE "st1"
TEXT 0.20, 0, "SCRITTA"

  ADD -0.10, -0.10, -0.05
MATERIAL "Alluminio"
BLOCK STW("SCRITTA")/1000+0.20, 0.70, 0.05
  DEL 1
```

09.07. A grande richiesta

Per concludere questa lezione accenniamo ad una serie di funzioni "speciali", che permettono di porre delle "domande" al programma. Si chiamano REQ e REQUEST; ne esistono diverse, e le trovate elencate nell'appendice del Manuale GDL. Possono fornire informazioni come il nome del progetto, i valori RGB (componenti rosso, verde, blu) di una penna, o viceversa (la penna che più si adatta ad una terna RGB), il numero e il nome del piano corrente, l'interlinea del carattere nello STYLE corrente, ecc.

Esercizio H

Realizzate un bel gazebo.
Pianta ottagonale, pilastrini, tetto a piramide con tanto di palla in cima. Parapetti su sette lati (uno aperto, altrimenti da dove si entra?) e piattaforma di base. Siete ormai grandi abbastanza per assumervi le vostre responsabilità... quindi sta a voi scegliere quali elementi (forme, colori, dimensioni ecc.) debbano essere modificabili tramite parametri. Io vi suggerisco solo di usare la variabile **A** per determinare l'ingombro, da lato a lato (**B** non viene utilizzata).



Questo tipo di misura anziché quella da angolo a angolo, è più laboriosa, da utilizzare, ma è più conforme all'uso corrente, quindi più pratica per l'utente. E visto che gli utenti siamo noi... possiamo bene usarci questa accortezza! Vorrei inoltre che scriveste le istruzioni GDL 2D per il simbolo.

LEZIONE 10

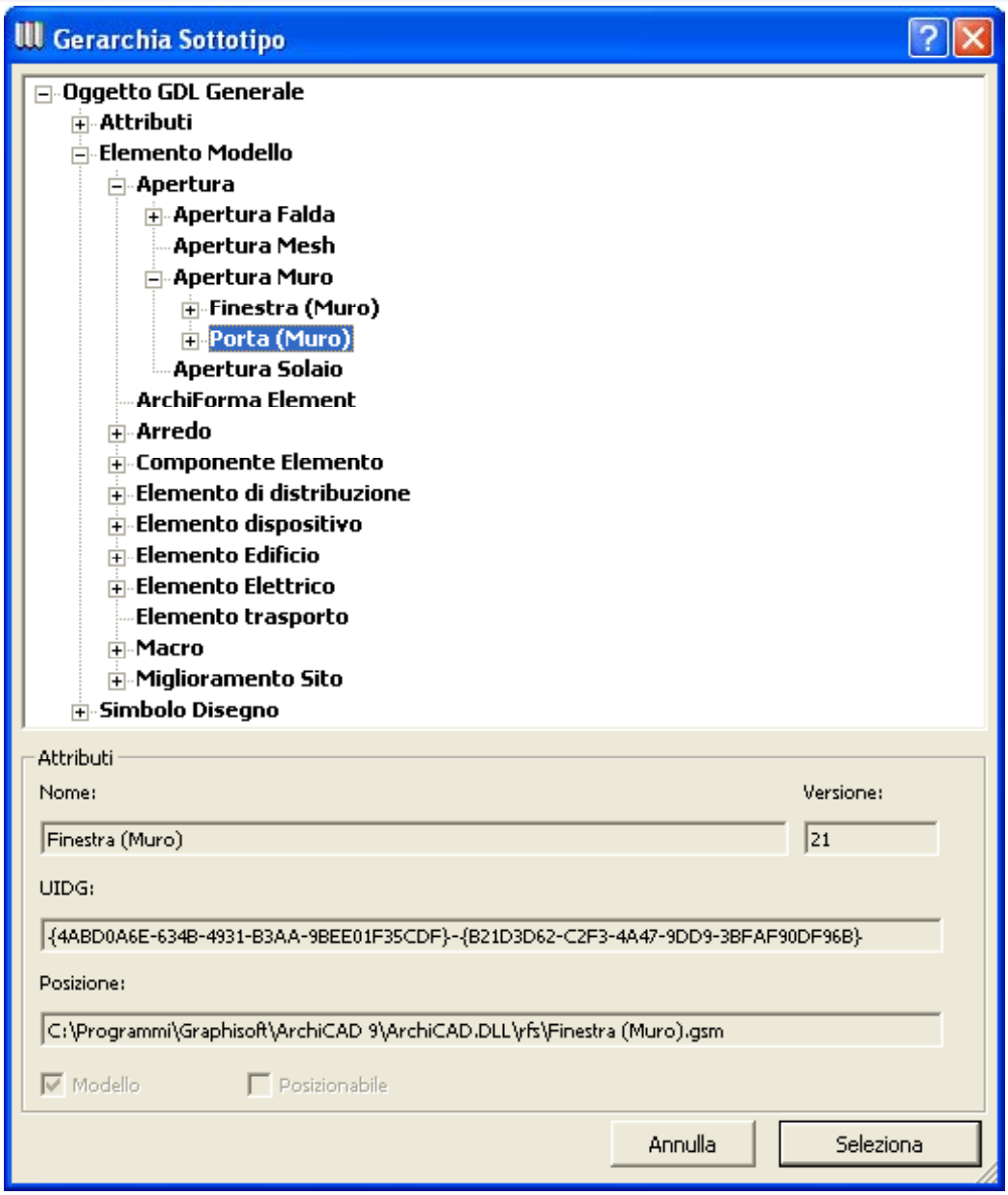
10.01. Altri personaggi in libreria

Abbiamo detto già molto, ma non tutto. Questa era in effetti una delle premesse: l'obiettivo di questo testo non è quello di sostituirsi al manuale, né quello di insegnare tutti i segreti della programmazione.
Il primo punto non vogliamo conseguirlo, anzi speriamo proprio che il manuale e questa guida possano costituire due parti che si integrano a vicenda, illustrando gli argomenti in maniera complementare. Il secondo punto non siamo in grado di raggiungerlo. Programmare è un'attività complessa e "personale". Noi speriamo di aver dato spunti e indicazioni sufficienti, ma ciascuno avrà esigenze diverse, e modi propri per affrontare e risolvere le situazioni che incontrerà.

In quest'ultima lezione parleremo di ciò che ancora non è stato detto, e di alcuni comandi ancora non trattati. Tenete presente, inoltre, che per alcuni comandi non abbiamo detto tutto... in qualche caso abbiamo volutamente tralasciato alcune opzioni secondarie, che avrebbero solo complicato l'esposizione.

Finora abbiamo parlato di "oggetti" senza specificare ulteriormente, ma oltre a quelli *generici* sapete che ne esistono di vario tipo: Porte, Finestre , Lampade, Timbri, Etichette, Macro, e altri.
Per creare un elemento specifico, anziché uno generico, occorre selezionarlo dall'elenco di quelli disponibili, usando il pulsante Seleziona Sottotipo, in alto, nella Finestra Principale dell'Oggetto.

L'Oggetto generico corrisponde al sottotipo "Elemento Modello". Scegliendo un sottotipo diverso noterete che ArchiCAD vi proporrà una diversa serie di parametri predefiniti, riconoscibili dal colore blu. Per esempio, per gli oggetti generici abbiamo visto che propone A, B e ZZYZX. Per le etichette, che non hanno 3D, non si avrà ZZYZX. Per alcuni sottotipi le variabili predefinite possono essere molto numerose, ma non è obbligatorio usarle tutte. Possono essere anche nascoste usando l'icona con la X.



Porte - la particolarità di questi oggetti risiede nel fatto che devono essere costruiti "orizzontalmente". Il piano x-y del GDL diventerà la facciata del muro su cui verrà posizionata la porta. La quota 0.00 corrisponde alla faccia esterna del muro e la posizione dell'origine è in basso, al centro dell'apertura. Elementi sporgenti (es. una soglia aggettante) devono essere costruiti a quote z negative (sotto il livello zero). Se nello script 3D (o Master) non ci sono istruzioni WALLHOLE, ArchiCAD userà le variabili A e B per creare l'apertura rettangolare nel muro ospite. Il pulsante **Dettagli** (sopra la sezione dei parametri) dà accesso ad una schermata in cui è possibile definire alcune opzioni, tra cui dei valori di scostamento per i 4 lati dell'apertura, ad esempio per ospitare una riquadratura, o un architrave particolare.

Finestre - sono del tutto analoghe alle Porte.

Lampade - Per emettere luce, devono contenere uno o più comandi LIGHT (per la sintassi, piuttosto complessa, consultate il manuale). Oppure si può richiamare una luce di libreria, con il comando CALL. Tra i numerosi parametri ve ne sono alcuni di tipo specifico, come le componenti cromatiche e l'intensità. Il colore della luce può essere impostato anche cliccando sul

campione in alto a destra, accanto al pulsante Seleziona Sottotipo.

Timbri - I timbri di zona sono elementi puramente bidimensionali (manca lo script 3D) ed hanno un gran numero di parametri predefiniti. Per citarne qualcuno, abbiamo ad esempio il volume della zona, il perimetro, il numero di angoli, la larghezza totale di tutte le porte, il livello del piano, ecc. Questi parametri vengono "valorizzati" da ArchiCAD durante l'utilizzo, desumendo i dati dalla zona cui fanno riferimento.

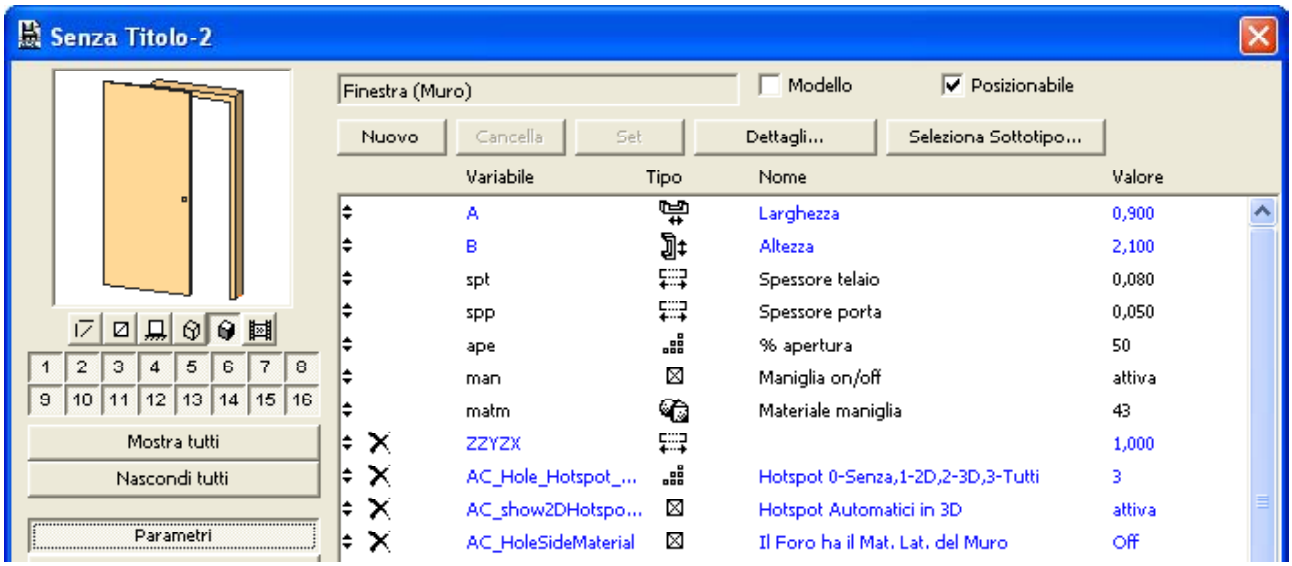
Etichette - Anche le etichette sono elementi esclusivamente bidimensionali. Per mezzo delle Variabili Globali possono essere usate per visualizzare informazioni sugli elementi cui vengano associate.

Macro - Con il termine Macro si indicano elementi GDL che vengono richiamati da altri elementi. In genere si tratta di componenti, che si comportano un po' come delle subroutine esterne. Qualsiasi oggetto può essere "chiamato" per mezzo dell'istruzione CALL. Per gli elementi creati esclusivamente a questo scopo, che non vengono usati direttamente dall'utente, si può disattivare la casella "Posizionabile", in alto a destra, per evitare che questi elementi risultino visibili quando si scorre la libreria dall'interno del programma.

10.02. Qualche esempio

Una volta il modo migliore per imparare ad usare praticamente il GDL era quello di aprire gli oggetti di libreria e guardare come erano fatti... ora questo non è più vero. Attualmente gli elementi sono estremamente sofisticati e hanno una tale complessità che molto difficilmente si riesce a capire come possano funzionare... Per ovviare parzialmente a questa difficoltà, inserisco qui pochi esempi di elementi semplici. Potrete copiarli e provare a modificarli, per vedere come funzionano.

ESEMPIO PORTA




```
! 2D - Porta

mez=a/2 ! -un mezzo della larghezza

! ---Telaio
```

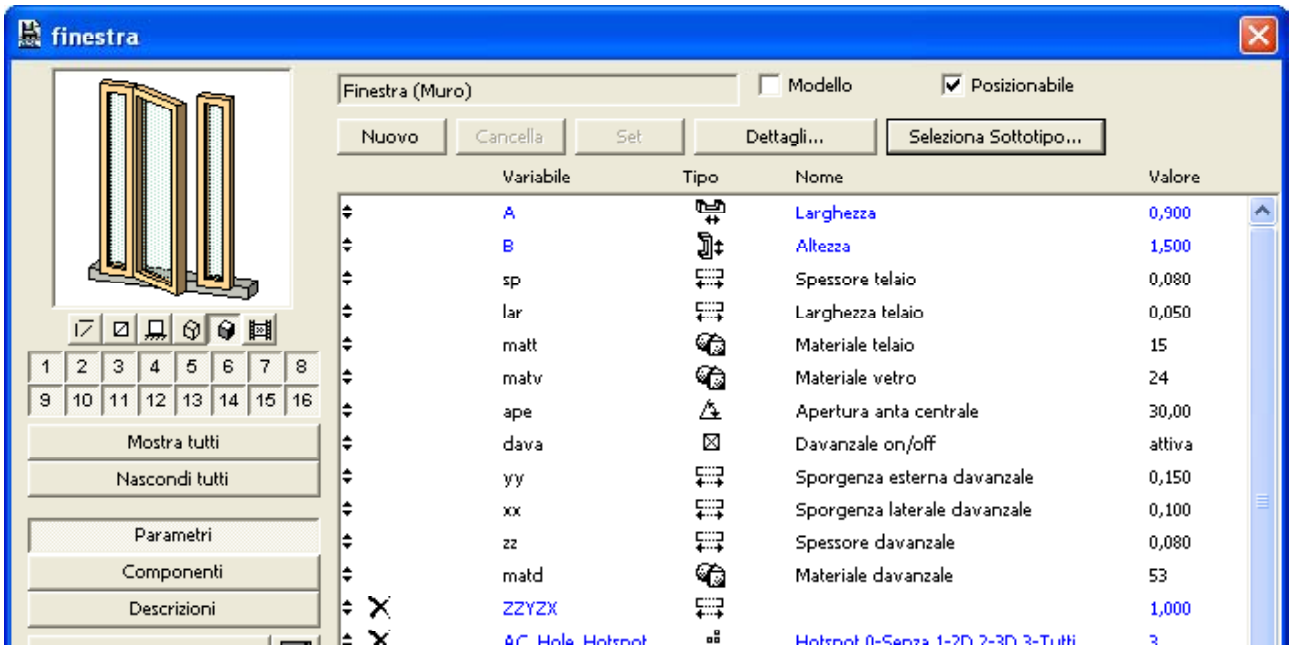
```
RECT2 -mez, -Wall_Thickness, -mez+spt, 0.00
RECT2  mez, -Wall_Thickness,  mez-spt, 0.00

! ---Anta
  ADD2 -mez+a*ape/100, 0.00
RECT2 0.00, 0.00, a, spp

END
```

```
! 3D - Porta

mez=a/2 ! —un mezzo della larghezza

! ---Telaio
PRISM 8, Wall_Thickness,
  -mez    , 0.00,
  -mez+spp, 0.00,
  -mez+spp, b-spp,
   mez-spp, b-spp,
   mez-spp, 0.00,
   mez    , 0.00,
   mez    , b,
  -mez    , b

! --- Anta
  ADDx a*ape/100
  ADDz -spp
PRISM 4, spp,
  -mez, 0.00,
   mez, 0.00,
   mez, b,
  -mez, b

IF man = 0 THEN END

! --- Maniglia
  ADD -mez+.05, 1.00, spp
MATERIAL matm
CYLIND .03, .02
  ADDz -spp-.02
CYLIND .03, .02
```

Commenti:
Vediamo prima di tutto che ho definito una stessa variabile sia nel testo 2D che in quello 3D (**mez**). In questi casi è possibile utilizzare lo Script Master. Ciò che si trova nella finestra Script Master è come se fosse presente anche all'inizio degli script 2D e 3D.
La variabile globale **Wall_Thickness** fornisce lo spessore del muro in cui si trova la porta o la finestra, e mi permette quindi di realizzare la riquadratura della dimensione corretta. Notate che l'intero oggetto viene traslato, se si utiliz-

za il settaggio apposito in ArchiCAD, per l'arretramento dell'infisso. Anche questo dato può comunque essere rilevato e, se necessario, corretto utilizzando un'altra variabile globale (Wido_Sill), come vedremo nell'esempio seguente.

ESEMPIO FINESTRA




```
! 2D - Finestra

RECT2 -a/2, 0.00, -a/4, -sp
RECT2  a/4, 0.00,  a/2, -sp

  ADD2 -a/4, -sp
HOTSPOT2 0,0
  ROT2 -ape
RECT2 0.00, 0.00, a/2, sp
  DEL 2

IF dava THEN
  RECT2 -a/2-xx, Wido_Sill, a/2+xx, Wido_Sill+yy
ENDIF

! 3D - Finestra

anta   = a/2
antina = a/4
spv    = 0.005 ! spessore vetro

  ADDX -a/2
w = antina
GOSUB 100
  DEL 1
  ADDx a/2-w
GOSUB 100
```

```
  DEL 1
  ADD -anta/2, 0.00, sp
  ROTy -ape
  ADDz -sp
w = anta
GOSUB 100
  DEL 3

IF dava THEN
  ADDz -Wido_Sill
  MATERIAL matd
  PRISM 4, -yy,
    -a/2-xx, 0.00,
     a/2+xx, 0.00,
     a/2+xx, -zz,
    -a/2-xx, -zz
ENDIF

END

100:
!-Specchiature
MATERIAL matt
PRISM_ 10, sp,
   0.00, 0.00, 15,
   w,    0.00, 15,
   w,    b,    15,
   0.00, b,    15,
   0.00, 0.00, -1,
   lar,   lar,   15,
   lar,   b-lar, 15,
   w-lar, b-lar, 15,
   w-lar, lar,   15,
   lar,   lar,   -1

MATERIAL matv
 ADDz sp/2-spv/2
PRISM 4, spv,
   lar ,  lar,
   lar,   b-lar,
   w-lar, b-lar,
   w-lar, lar

 DEL 1
RETURN
```

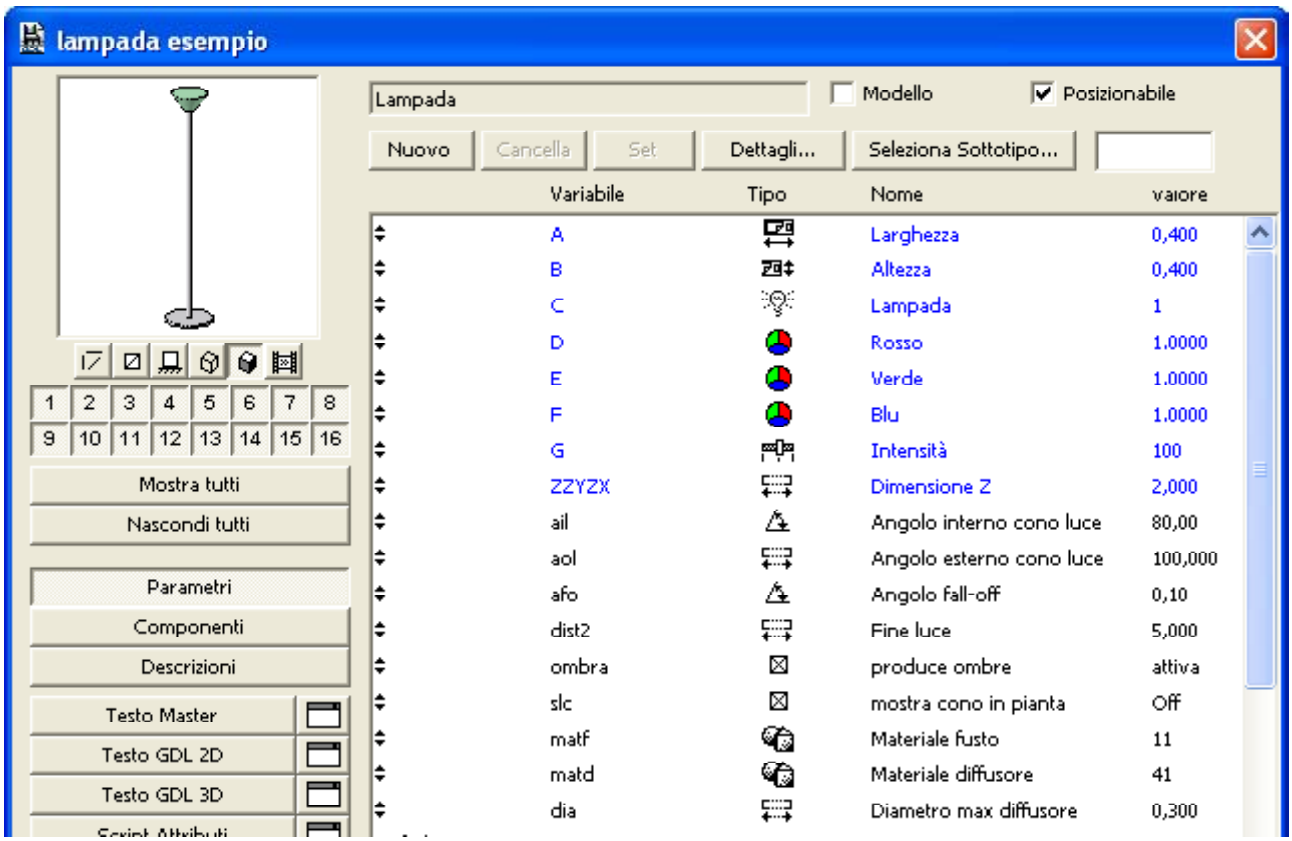
Commenti:
Il 2D è volutamente molto semplice. Provate ad aggiungere alcune linee, per mostrare il telaio ed il vetro.
Un'unica subroutine realizza le tre ante. Una variabile di servizio (**w**) viene uti-

lizzata per determinare la larghezza dell'anta da realizzare.

Se volete utilizzare il comando PROJECT2 con porte e finestre dovreste utilizzare le seguenti istruzioni:

```
ROT2 180
PROJECT2 4, 90, 1
```

ESEMPIO LAMPADA

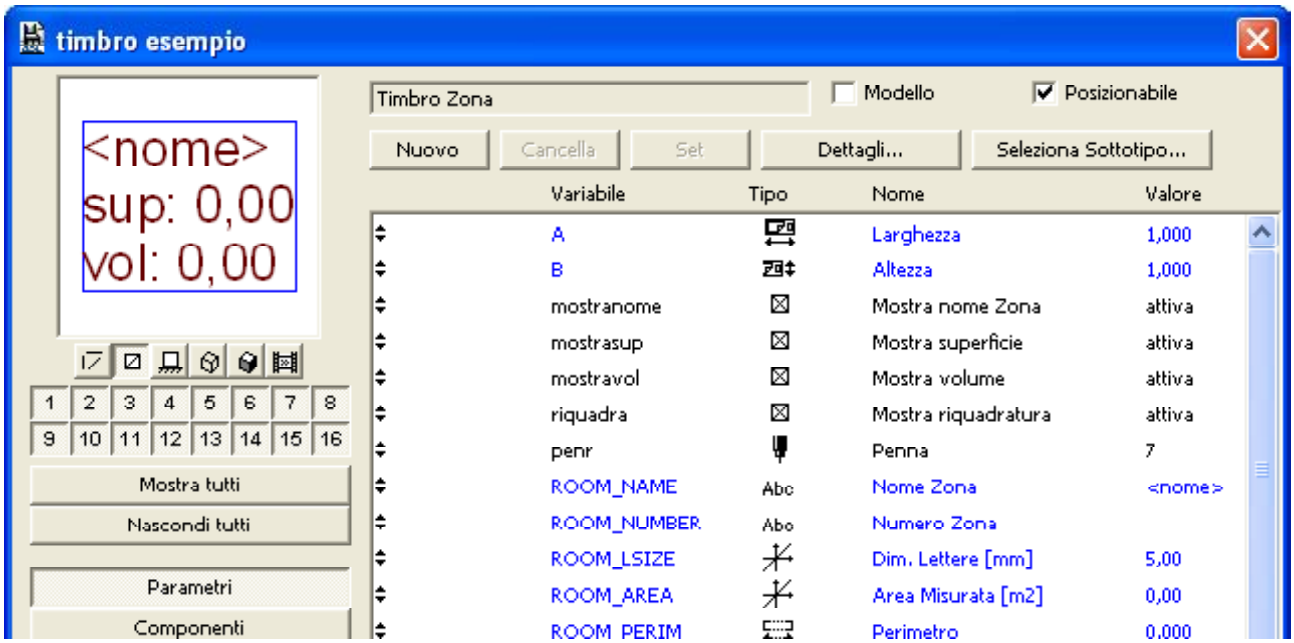



```
! 2D - Lampada
CIRCLE2 0.00, 0.00, dia/2
IF a>dia THEN CIRCLE2 0.00, 0.00, a/2

! 3D - Lampada

fusto= zzyzx-0.02-0.15

MATERIAL matf
CYLIND 0.02, a/2
  ADDz 0.02
CYLIND  fusto, .02
  ADDz fusto
MATERIAL matd
CONE 0.15, 0.04, dia/2, 90, 90
  DEL 2

ADDz zzyzx
CALL "Cono di luce verso alto" PARAMETERS a=a, b=b,
```

```
    c=c, d=d, e=e, f=f, g=g, gs_detlevel_3D="dettagliato",
    ail=ail, aol=aol, afo=afo, dist2=dist2, light_shadow=sc, slc=0
```

Commenti:
Nel 2D la seconda istruzione traccia il cerchio corrispondente alla base solo se il diametro è maggiore di quello del diffusore.
Nel 3D si utilizza una chiamata (CALL) ad un oggetto di libreria. Abbiamo utilizzato una luce presente nella cartella Sorgenti luminose della Libreria di Archi-CAD 7.0. Per dare la possibilità di utilizzarne i parametri, questi sono stati semplicemente copiati da tale oggetto. Nell'istruzione CALL la parola chiave PARAMETERS permette di assegnare un valore ai parametri (tutti o una parte) dell'oggetto chiamato; di solito, per semplicità, si utilizzano gli stessi nomi nei due oggetti, ma non è necessario.

ESEMPIO TIMBRO ZONA




```
! 2D - Timbro sup-vol

hh    = Room_Lsize *1.5 *a_/1000 ! - altezza riga
ww    = 0
righe = 0

DEFINE STYLE "temp" "arial", Room_Lsize, 1, 0
STYLE temp

IF mostranome THEN
  TEXT2 0.00, 0.00, Room_Name
  righe = righe + 1
  ww = MAX(ww, STW(Room_Name)*a_/1000)
  ADD2 0.00, -hh
ENDIF

IF mostrasup THEN
  ra = "sup: " + STR(Room_Area, 4, 2)
```

```
  TEXT2 0.00, 0.00, ra
  righe = righe + 1
  ww = MAX(ww, STW(ra)*a_/1000)
  ADD2 0.00, -hh
ENDIF

IF mostravol THEN
  rv = "vol: " + STR(Room_Volume, 4, 2)
  TEXT2 0.00, 0.00, rv
  righe = righe + 1
  ww = MAX(ww, STW(rv)*a_/1000)
  ADD2 0.00, -hh
ENDIF

DEL righe

IF riquadra THEN
  PEN penr
  RECT2 0.00, 0.00, ww, -hh*righe
  ADD2 0.00, -hh
ENDIF
```

Commenti:
L'operazione ***a_/1000**, come abbiamo già visto, permette di trasformare in metri "di progetto" le dimensioni espresse in millimetri "di stampa", adattando la misura alla scala corrente. I timbri di zona hanno molti parametri predefiniti. Questo esempio utilizza Room_Lsize (dimensione del testo), Room_Name (nome della zona), Room_Area (area della zona) e Room_Volume (volume della zona... l'avreste mai detto ?!).
All'inizio ho creato la variabile **hh** per l'altezza di ogni riga di testo, calcolata come una volta e mezza l'altezza del carattere.
La variabile **righe** è usata per contare il numero di informazioni mostrate, incrementandola di uno ogni volta che si scrive una nuova riga di testo. Successivamente è usata per annullare (con DEL) gli spostamenti verso il basso, effettuati per scrivere le righe una sotto l'altra. Infine è utilizzata anche per calcolare l'altezza della riquadratura.
La variabile **ww** è inizialmente definita come zero, poi viene confrontata con la larghezza di ciascun testo. La funzione MAX fa in modo che le venga assegnato il valore più alto, fra quello corrente e la larghezza della stringa attuale. Al termine dei confronti **ww** conterrà la larghezza della scritta più lunga, fra quelle inserite. Tale larghezza viene infine utilizzata per la riquadratura.

10.03. Ultimi giri

Vi indico alcune istruzioni ancora, ma non vi dirò di preciso come si usano:
- DO ... WHILE
- WHILE ... DO ... ENDWHILE
- REPEAT ... UNTIL

Consultate il manuale se volete provare ad utilizzare queste istruzioni, che permettono di eseguire porzioni di codice verificando una condizione, come accade per IF... THEN, ma tornando ad eseguirle e verificando nuovamente la condizio-

ne, un po' come nel caso di FOR...NEXT, che controlla la variabile utilizzata come contatore, per stabilire se il ciclo deve essere eseguito ancora.
Quando utilizzate i cicli fate attenzione a non impostare un ciclo infinito, altrimenti potrebbe risultare impossibile bloccarne l'esecuzione (ad esempio FOR x = 1 TO 10 STEP 0; con incremento nullo non si arriverà mai a 10, e quindi a completare il ciclo).

10.04. Un po' di make-up all'interfaccia

I comandi sotto elencati sono utilizzabili unicamente nello script Testo Interfaccia. Permettono di creare delle schermate personalizzate visibili nella finestra Settaggi Oggetto. Questa schermata (chiamata *User Interface*, Interfaccia Utente) può essere usata per l'inserimento dei parametri, in alternativa o in aggiunta alla normale lista presentata da ArchiCAD.
Per la sintassi esatta di ciascun comando consultate il manuale.

UI_DIALOG - è una dichiarazione obbligatoria, che deve esser sempre messa all'inizio dello script.

UI_PAGE - Permette di definire l'inizio di una nuova schermata. L'interfaccia utente può essere composta di più schermate, e si usa l'istruzione UI_BUTTON per passare dall'una all'altra.

UI_BUTTON - posiziona un pulsante nella schermata. L'opzione UI_NEXT porta alla schermata (UI_PAGE) successiva; l'opzione UI_PREV a quella precedente.

UI_STYLE - definisce lo stile di testo da utilizzare. Il carattere è sempre quello di sistema, ma sono previste 3 dimensioni (medio, piccolo, grande) e vari stili (normale, grassetto, corsivo, ecc.).

UI_OUTFIELD - permette di definire un "campo di output" ovvero, più semplicemente, del testo da scrivere nella schermata.

UI_INFIELD - permette di definire un "campo di input" ovvero, più semplicemente, un campo di immissione per un parametro. Il parametro deve comunque essere creato preventivamente nel modo consueto. Questo comando ha due sintassi: una "normale" e una "estesa". Quest'ultima permette di creare una sorta di menu di scelta grafico, associato ad una lista valori (deve essere presente il corrispondente comando VALUES nello Script Parametri). Per un esempio di tale utilizzo provate a vedere la scelta delle pannellature in molte delle porte delle libreria standard.

UI_PICT - dà la possibilità di aggiungere un'immagine. Il file corrispondente deve trovarsi nella libreria.

UI_SEPARATOR - Una semplice linea di divisione, orizzontale o verticale, per separare gruppi di parametri o altre informazioni.

UI_GROUPBOX - Un riquadro rettangolare, con un titolo opzionale, per racchiudere gruppi di parametri o altre informazioni.

Nella parte superiore della finestra è presente il pulsante **Setta come default**. Attivandolo, la schermata Interfaccia Utente verrà presentata per prima, quando si apre la finestra settaggi dell'oggetto. Accanto a questo il pulsante **Anteprima** dà la possibilità di prendere visione della schermata, mentre si sta editando lo script.

10.05. Qualcosa in più sui parametri

Nello script dei Parametri, la parola chiave LOCK permette di bloccare il valore di un parametro, in modo che l'utente non possa modificarlo.
Facciamo un semplice esempio:
Abbiamo un mobiletto, ed un parametro booleano (**spo**) che ci permette di scegliere se avere lo sportello oppure no. Un altro parametro (**mats**) consente la scelta del materiale per lo sportello. Possiamo fare in modo che quest'ultimo sia disponibile solo se l'opzione per lo sportello è attiva.
Nello script dei Parametri possiamo inserire:

```
IF spo=0 THEN LOCK "mats"
```

in questo modo il parametro **mats** risulta bloccato se **spo** è zero (checkbox non selezionato). Il comando HIDEPARAMETER funziona in modo analogo, ma invece di mostrare il parametro bloccato, lo fa sparire dalla lista parametri (solo durante l'uso, nella finestra settaggi, non durante l'editazione dell'oggetto). Notate che questi due comandi (come VALUES) richiedono che il nome del parametro sia messo tra virgolette, mentre il successivo no.

Nei testi GDL possiamo modificare il valore assegnato alle variabili, compresi i parametri dell'oggetto, ma i valori modificati non vengono visualizzati nella lista dei parametri. Questo si può ottenere utilizzando il comando PARAMETERS nello script dei Parametri.
Anche in questo caso un esempio:
Abbiamo il solito mobiletto. Il parametro **zzyzx** definisce l'altezza, e il parametro **rip** permette di immettere il numero di ripiani. Stabiliamo che il passo tra i ripiani non può essere inferiore a 10 centimetri, quindi con l'istruzione PARAMETERS correggiamo il valore di **rip**, se questa condizione non è verificata; ciò può accadere sia a seguito della modifica dell'altezza (riducendo **zzyzx**) che della modifica del numero di ripiani (aumentando **rip**).
Inserendo le seguenti istruzioni nello Script Parametri

```
passo = zzyzx / (rip+.001) ! evita divisione per zero!
num   = zzyzx / 0.20       ! passo ideale a 20 cm

IF passo < 0.10 THEN PARAMETERS rip = num
```

il valore di **rip** verrà modificato ogni volta che lo spazio diventa insufficiente, e verrà proposto all'utente un valore che permetta di inserire un numero "giusto" di ripiani.

10.06. Un nuovo movimento

Di sicuro conoscete la comodità degli hotspot editabili. Sono quelli a forma di diamante, riconoscibili anche per il colore differente, che permettono di manipolare graficamente i parametri lineari o angolari di alcuni oggetti. Come si creano? Con una sintassi particolare dei comandi HOTSPOT e HOTSPOT2.
Per definire i nodi editabili, ogni comando HOTSPOT avrà tre parametri in più: un numero identificativo UNICO, il nome del parametro associato, ed un codice, che descrive il tipo di punto.
Per associare un Hotspot editabile ad un parametro di tipo "Lunghezza", che chiameremo lun, si dovrà inserire la seguente serie di comandi:

```
HOTSPOT2   0, 0, 996, lun, 1
HOTSPOT2 lun, 0, 997, lun, 2
```

```
HOTSPOT2  -1, 0, 998, lun, 3
```

Questa sequenza definisce:

- un punto di origine del movimento, alle coordinate 0, 0 (codice 1). Questo hotspot, nell'oggetto selezionato, può essere nascosto aggiungendo 128 al codice. Il punto può essere reso a sua volta editabile, aggiungendo 256 al codice.
- il punto mobile, alle coordinate lun, 0 (codice 2). Questo hotspot si sposterà lungo la retta che unisce i punti con codice 1 e 3.
- un punto di riferimento, che definisce la direzione negativa dell'asse di scorrimento, alle coordinate -1,0 (codice 3). Questo hotspot è sempre invisibile.

Ho usato i numeri 996, 997 e 998, per assegnare un codice unico ai tre comandi HOTSPOT2, ma c'è un sistema più pratico. Usare una variabile, incrementandola ogni volta che si aggiunge un comando HOTSPOT.

```
hid = 1
HOTSPOT2   0, 0, hid, lun, 1 : hid = hid + 1
HOTSPOT2 lun, 0, hid, lun, 2 : hid = hid + 1
HOTSPOT2  -1, 0, hid, lun, 3 : hid = hid + 1
```

Se nello stesso oggetto si definiscono due hotspot di questo tipo, su rette incidenti, e si posiziona il nodo mobile nel punto di intersezione, il movimento non sarà più vincolato alla retta. Non è semplice da capire, a parole, quindi faccio un esempio. Provatelo e... ve ne innamorerete!

Variabile	Tipo	Nome	Valore
A		Dimensione X	1,400
B		Dimensione Y	0,800
ZZYZX		Dimensione Z	0,720
spe		spessore piano	0,030
diamg		diametro gambe	0,060
xg		posizione X gambe	0,100
yg		posizione Y gambe	0,100

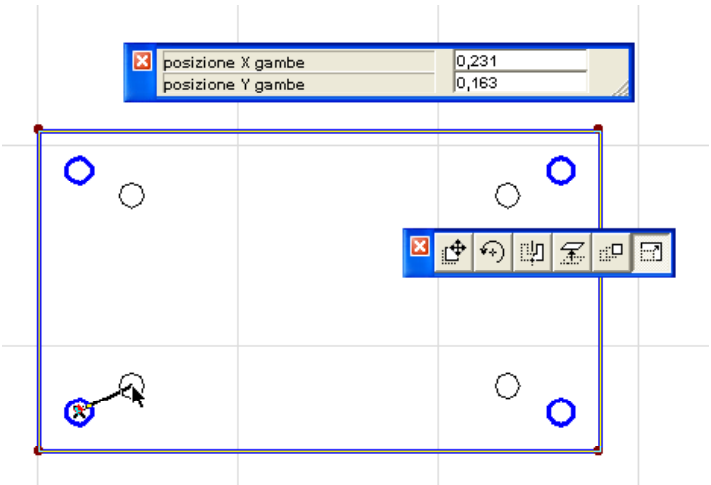
```
!! Script 3D !!
  ADDz zzyzx-spe
BLOCK a, b, spe
  DEL 1

  ADD xg, yg, 0
CYLIND zzyzx-spe, diamg/2
  DEL 1

  ADD a-xg, yg, 0
CYLIND zzyzx-spe, diamg/2
  DEL 1

  ADD xg, b-yg, 0
CYLIND zzyzx-spe, diamg/2
  DEL 1
```

```
  ADD a-xg, b-yg, 0
CYLIND zzyzx-spe, diamg/2
  DEL 1

!! Script 2D !!
RECT2 0, 0, a, b
CIRCLE2 xg,     yg, diamg/2
CIRCLE2 a-xg,   yg, diamg/2
CIRCLE2 xg,   b-yg, diamg/2
CIRCLE2 a-xg, b-yg, diamg/2

! - Hotspot normali
HOTSPOT2 0, 0
HOTSPOT2 a, 0
HOTSPOT2 0, b
HOTSPOT2 a, b

! - Hotspot editabili
hid = hid+1 : HOTSPOT2  0, yg, hid, xg, 1+128
hid = hid+1 : HOTSPOT2 xg, yg, hid, xg, 2
hid = hid+1 : HOTSPOT2 -1, yg, hid, xg, 3

hid = hid+1 : HOTSPOT2 xg,  0, hid, yg, 1+128
hid = hid+1 : HOTSPOT2 xg, yg, hid, yg, 2
hid = hid+1 : HOTSPOT2 xg, -1, hid, yg, 3
```

Il 3D è molto semplice. Nel 2D:

- Le prime istruzioni tracciano un rettangolo, per il piano, e quattro cerchi che indicano la posizione delle gambe.
- Le istruzioni seguenti posizionano quattro Hotspot normali, agli angoli del piano.
- Gli ultimi due gruppi di istruzioni posizionano l'hotspot editabile in corrispondenza del punto xg,yg.

La posizione della gamba in basso a sinistra può essere trascinata direttamente con il mouse. Questa operazione modifica il valore delle variabili xg e yg. Le stesse variabili sono usate anche per posizionare le altre tre gambe, quindi durante l'editazione in Pianta vedrete muoversi i quattro cerchi contemporaneamente.



È sempre possibile anche l'input numerico, per mezzo della palette di immissione. I campi possono essere attivati anche da tastiera, usando il tasto 'n' in modo analogo ai campi x, y, z, r, a (Barra coordinate) e b, t (Palette Informazioni).
Nel 3D gli Hotspot editabili funzionano in modo analogo. Se volete provarlo, con l'oggetto precedente, inserite le seguenti righe nello script 3D.

```
hid = hid+1 : HOTSPOT 0, 0, zzyzx,      hid, spe, 1+128
hid = hid+1 : HOTSPOT 0, 0, zzyzx-spe, hid, spe, 2
hid = hid+1 : HOTSPOT 0, 0, zzyzx+1,    hid, spe, 3
```

Potrete così controllare graficamente lo spessore del piano. I numeri unici degli hotspot possono essere duplicati, se si trovano in script diversi. Ricordate che questo è un semplice esempio. Occorrerebbe aggiungere dei controlli, per evitare uno spessore inferiore a zero (che con BLOCK dà errore) o per tenere le gambe… all'interno del tavolo! Per esempio, nello Script Parametri, potreste inserire:

```
IF spe<0.02 THEN PARAMETERS spe=0.02
```

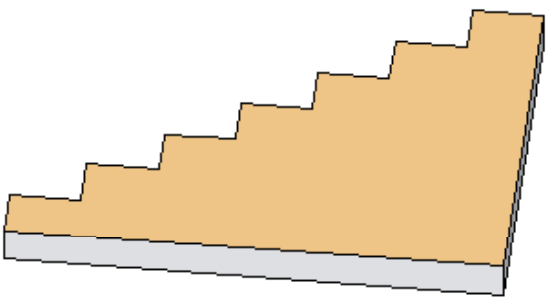
per limitare lo spessore minimo a 2 centimetri Queste funzioni operano anche in tempo reale, durante l'editazione grafica dei parametri..

Per i parametri di tipo Angolo, i codici sono 4 (origine dello spostamento), 5 (punto mobile) e 6 (centro di rotazione). Nel 2D è possibile aggiungere 512 al codice 6 per usare angoli orari, anziché antiorari. Nel 3D, invece, è necessaria un'istruzione HOTSPOT in più, con codice 7, per definire l'asse Z locale, passante per il centro di rotazione.
Nell'esempio seguente sono definiti i parametri rag (Lunghezza) e ang (Angolo). La posizione del punto mobile deve essere calcolata usando le funzioni trigonometriche.

```
hid = hid+1 : HOTSPOT2 rag,           0,             hid, ang,
 4+128
hid = hid+1 : HOTSPOT2 COS(ang)*rag, SIN(ang)*rag, hid, ang, 5
hid = hid+1 : HOTSPOT2 0,             0,             hid, ang, 6

FOR f = 1 TO 4
ARC2 0, 0, rag/4*f, 0, ang
NEXT f
```

10.07. Impara l'arte e mettila da parte

Mi sembra giusto, prima di chiudere questa trattazione, accennare all'esistenza e all'uso del cosiddetto *buffer*. È un'area di memoria in cui possono essere temporaneamente immagazzinati dei valori, per poterli successivamente richiamare. L'utilizzo più adeguato è quello che permette di realizzare, ad esempio, un prisma con un numero variabile di punti, cosa altrimenti impossibile.



Ammettiamo di avere il parametro **n** (intero) e di voler realizzare un prisma scalettato, con **n** "scalini".
Non è possibile utilizzare cicli o altri artifici, all'interno di un comando PRISM, e anzi dobbiamo dichiarare subito il numero di nodi della base.
Possiamo realizzarlo utilizzando il buffer. Con l'istruzione PUT si inseriscono i valori, che poi si estrarranno con GET (o con USE, se devono essere utilizzati ulteriormente). La funzione NSP permette di conoscere quanti valori sono attualmente presenti.
Vediamo dunque come si utilizza:

```
! Testo 3D - PUT/GET

passox = a/n
passoy = b/n

FOR f = 0 TO n-1
  PUT f * passox,  f * passoy
  PUT f * passox,  f * passoy+passoy
NEXT f

PRISM n*2+2, zzyzx,
  GET(NSP),
  a, b,
  a, 0.00
```

Uso prima un ciclo per caricare nel buffer le coordinate dei vertici degli scalini. Il numero di punti che compongono la base del prisma è 2 per ogni scalino, più 2 per completare la figura, quindi n*2+2.
Come potete notare, l'istruzione GET può essere usata all'interno di un comando, perché restituisce i valori come se fossero separati da virgole. Nel nostro caso, per vuotare l'intero buffer, ho utilizzato GET(NSP). Avrei potuto usare anche GET (n*4), perché per ogni scalino (**n**) ho inserito (PUT) 4 coordinate (x e y di dei due punti che lo definiscono). Completo il comando con i due nodi più a destra, di cui non ho inserito le coordinate nel buffer (avrei potuto farlo, ad esempio, aggiungendo PUT a, b, a, 0.00 dopo la fine del ciclo FOR/NEXT).

Se avessi voluto usare il comando PRISM_, ad esempio, che richiede un valore status per ogni coordinata, avrei usato la forma

```
PUT  f*passox,  f*passoy, 15
```

aggiungendo semplicemente anche il valore status nel buffer.

Ultimissimo. Il GDL può fare tutto quello che fa ArchiCAD. Anzi, per essere più precisi, è ArchiCAD che può fare tutto quello che fa il GDL. Non c'è da sorprendersi, quindi, se diciamo che la versione corrente del GDL è in grado di eseguire le operazioni Booleane sui solidi. Non intendiamo addentrarci in questa trattazione, ma illustreremo rapidamente il comando più utilizzato, la sottrazione booleana.
Per prima cosa è necessario definire due "gruppi", formati da uno o più solidi. Questo si realizza semplicemente racchiudendo i comandi tra le dichiarazioni GROUP e ENDGROUP.

```
GROUP "uno"
   MATERIAL 1
   BLOCK 1.00, 0.80, 0.80
ENDGROUP

GROUP "due"
   MATERIAL 2
   ADD 0.50, 0.40, 0.00
   CYLIND 1.00, 0.30
   DEL 1
   ADD 0.50, 0.00 ,0.40
   ROTx -90
   CYLIND 1.00, 0.30
   DEL 2
ENDGROUP

tre = SUBGROUP ("uno", "due")
PLACEGROUP tre
```

Fatto questo si genera un nuovo gruppo, usando il comando SUBGROUP, che effettua la sottrazione. Esistono anche i comandi ADDGROUP (somma), ISECTGROUP (intersezione) e altri. Il gruppo risultante, come pure quelli definiti direttamente, possono essere posizionati solo in modo esplicito, portandosi nel punto desiderato e usando il comando PLACEGROUP.
I nomi dei gruppi originali, normalmente, sono costanti stringa, quindi racchiusi tra virgolette ("uno" e "due"). I risultati (tre) sono variabili stringa, e come tali devono essere scritti senza virgolette.
Quando una gruppo non serve più, è possibile scaricare dalla memoria la sua definizione, con il comando KILLGROUP. Nel nostro esempio, dopo aver ottenuto il gruppo tre, potevamo inserire

```
KILLGROUP "uno", "due"
```

In ogni caso, al termine dello script, i gruppi vengono comunque scaricati.

Questo è tutto.

La scuola è finita, quindi oggi niente compiti a casa.
Vi lascio con il mio personale augurio: *happy coding* !

APPENDICE 01

Soluzioni

Di seguito troverete una possibile soluzione per gli esercizi proposti nel testo. Ovviamente sono possibili soluzioni molto diverse... l'importante è che funzionino. Ho cercato di mantenere lo svolgimento semplice, per rendere più abbordabile l'analisi del listato. In molti casi sarebbero opportune delle integrazioni, ma spero che troviate il tempo e la voglia per provarci da soli.

Esercizio A

```
! 4Blocchi
! Script 3D

   BLOCK a*2/3, b/3, zzyzx
     ADDX a*2/3
   BLOCK a/3, b*2/3, zzyzx
     DEL 1
     ADDY b/3
   BLOCK a/3, b*2/3, zzyzx
     DEL 1
     ADD a/3, b*2/3, 0
   BLOCK a*2/3, b/3, zzyzx

! Script 2D

   PROJECT2 3, 270, 2
```

Esercizio B

parametri:

largh, **ped** (lunghezza);
giro, **num** (intero)

```
! Spirale di blocchi
! Script 3D

   ang1  = (360/giro)/2
   ang2  = 180-90-ang1
   dist  = (ped/2) / COS(ang2)
   spes  = zzyzx / num

   !- main
   FOR k=1 TO num
     GOSUB 100
```

```
      ROTZ 360/giro
   NEXT k
     DEL num
   END ! ================

   !- blocchi
   100:
      ADDX dist
      ROTZ (360/giro)/2
   BLOCK  largh, ped, spes
      DEL 2
      ADDz spes
   RETURN

! Script 2D

   PROJECT2 3, 270, 2
```

Esercizio C

parametri:

lungb, **diam** (lunghezza);
numb (intero);
matb (materiale)

```
! Balaustra
! Script 3D

   parte = (a-lungb)/2   ! lung. parte piena
   passo = zzyzx/numb    ! distanza fra le barre

   BLOCK parte, b, alt   ! primo pezzo pieno
     ADDx parte+lungb    ! spostamento
   BLOCK parte, b, alt   ! secondo pezzo pieno
     DEL 1               ! annulla lo spostamento

     ADD parte, b/2, passo/2 ! origine nel punto di partenza
                             ! della prima barra
     ROTY 90                 ! posizionamento dell'asse Z in
                             ! orizzontale

   MATERIAL matb             ! materiale delle barre
   FOR f=1 TO numb           ! impostazione del ciclo
     CYLIND lungb, diam/2    ! singola barra
       ADDX -passo           ! elevazione: a seguito del ROTY 90,
                             ! l'asse verticale è -X
   NEXT f

   DEL TOP
```

```
    END

! Script 2D

    PROJECT2 3, 270, 2
```

Esercizio D

parametri:

sp,**vdimg** (lunghezza);
matp (materiale);
forma (testo)

```
! Tavolo
! -Script 3D

    altg = zzyzx - sp ! altezza delle gambe

      ADDZ zzyzx-sp
    MATERIAL matp
    BLOCK a, b, sp ! - piano
    MATERIAL SYMB_MAT
      DEL 1


    IF forma = "Triangolari" THEN GOSUB 100
    IF forma = "Cilindriche" THEN GOSUB 200
    IF forma = "Coniche"     THEN GOSUB 300
    IF forma = "Sagomate"    THEN GOSUB 400

    END ! ================

    100: ! -- triangolari --
    GOSUB 150  ! - 1 -

      ADDX a
      ROTZ 90
    GOSUB 150  ! - 2 -
      DEL  2

      ADDY b
      ROTZ -90
    GOSUB 150  ! - 3 -
      DEL  2

      ADD  a, b, 0.00
      ROTZ 180
    GOSUB 150  ! - 4 -
      DEL  2
```

```
RETURN

150: !---gamba tri
PRISM 3, altg,
   0.00, 0.00,
   dimg, 0.00,
   0.00, dimg
RETURN

200: ! - cilindriche -
raggio = dimg / 2

  ADD raggio, raggio, 0.00
GOSUB 250  ! - 1 -
  DEL 1

  ADD a - raggio, raggio, 0.00
GOSUB 250  ! - 2 -
  DEL 1

  ADD raggio, b - raggio, 0.00
GOSUB 250  ! - 3 -
  DEL 1

  ADD a - raggio, b - raggio, 0.00
GOSUB 250  ! - 4 -
  DEL 1

RETURN

250: !-------gamba cil
CYLIND altg, raggio
RETURN

300: ! -- coniche --
raggio  = dimg / 2
raggiobase = dimg / 4

  ADD raggio, raggio, 0.00
GOSUB 350  ! - 1 -
  DEL 1

  ADD a - raggio, raggio, 0.00
GOSUB 350  ! - 2 -
  DEL 1

  ADD raggio, b - raggio, 0.00
GOSUB 350  ! - 3 -
  DEL 1
```

```
    ADD a - raggio, b - raggio, 0.00
  GOSUB 350  ! - 4 -
    DEL 1

  RETURN

  350: !-------gamba cono
  CONE altg, raggiobase, raggio, 90, 90
  RETURN

  400: ! - Sagomate -
  raggio  = dimg / 2
  raggiobase = dimg / 4
  raggio2 = dimg/3

    ADD raggio, raggio, 0.00
  GOSUB 450  ! - 1 -
    DEL 1

    ADD a - raggio, raggio, 0.00
  GOSUB 450  ! - 2 -
    DEL 1

    ADD raggio, b - raggio, 0.00
  GOSUB 450  ! - 3 -
    DEL 1

    ADD a - raggio, b - raggio, 0.00
  GOSUB 450  ! - 4 -
    DEL 1

  RETURN

  450: !-------gamba sagomata
  CYLIND altg*0.07, raggiobase
    ADDZ altg*0.07
  CONE altg*0.80, raggiobase, raggio, 90, 90
    ADDZ altg*0.80
  CONE altg*0.03, raggio, raggio2, 90, 90
    ADDZ altg*0.03
  CONE altg*0.06, raggio2, raggio, 90, 90
    ADDZ altg*0.06
  CONE altg*0.04, raggio, raggio2, 90, 90
    DEL 4
  RETURN

! Script Parametri
```

```
    VALUES "forma" "Triangolari", "Cilindriche", "Coniche",
     "Sagomate"

! Script 2D

    PROJECT2 3, 270, 2
```

Esercizio E

parametri:

num (intero);
ste (reale)

```
! Stella
! Script 2D

    r1 = a/2
    r2 = r1*ste/100
    ang = 360 / num ! distanza angolare tra 2 punte

    MUL2 1, b/a
    HOTSPOT2 0, 0
    HOTSPOT2 -a/2, -a/2
    HOTSPOT2  a/2,  a/2

    FOR k = 1 TO num
       HOTSPOT2 r1, 0
       LINE2 r1, 0,   r2*COS(ang/2), r2*SIN(ang/2)
       LINE2 r2*COS(ang/2), r2*SIN(ang/2),   r1*COS(ang),
     r1*SIN(ang)
       ROT2 ang
    NEXT k

    DEL TOP
    END
```

Esercizio F

parametri:

lato (lunghezza);
num (intero);
piramide (booleano)

```
! Prisma Piramide
! Script 3D

    ang1  = (360/num)/2
    ang2  = 180-90-ang1
    rag   = (lato/2) / COS(ang2)
```

```
    RESOL num
    ROTZ 90     ! vertice verso l'alto
    IF piramide THEN
       CONE zzyzx, rag, 0.00, 90, 90
    ELSE
       CYLIND zzyzx, rag
    ENDIF

! Script 2D

    PROJECT2 3, 270, 2
```

Esercizio G

parametri:

curvo (booleano);
matb, **matp**, **matc**, **mats** (materiale)

```
! Lampione
! Script 3D

    h1 = 0.25     ! h zoccolo
    h2 = zzyzx/4  ! h parte inferiore
    h3 = 0.08     ! h collare (userò h3*1.5)
    h4 = b/4      ! h parte conica di raccordo
    r1 = b/3      ! raggio parte inferiore
    r2 = b/6      ! raggio parte superiore
    r3 = b*0.35   ! raggio collare

    MATERIAL matb
    CYLIND h1, b/2
      ADDZ h1
    MATERIAL matp
    CYLIND h2, r1
      ADDZ h2
    CONE h4, r1, r2, 90, 90
      ADDZ h4
    IF curvo THEN
       GOSUB 100
    ELSE
       GOSUB 200
    ENDIF

    END

    100: !  - - - curvo
       alt= zzyzx - h1-h2-h4 - a/2
       CYLIND alt, r2
```

```
      ADDZ alt
    ELBOW a/2, 180, r2
      ADDX a
      MULZ -1   ! inversione asse Z. Ora è positivo verso il
 basso
    MATERIAL matc
    CYLIND h3*1.5, r3 ! altezza maggiorata, per
                      ! compenetrare la sfera
      ADDZ h3 + b/2
    MATERIAL mats
    SPHERE b/2
 RETURN

 200: !  - - - dritto
    alt= zzyzx - h1-h2-h4-b-h3
    CYLIND alt, r2
      ADDZ alt
    MATERIAL matc
    CYLIND h3*1.5, r3 ! altezza maggiorata, per
                      ! compenetrare la sfera
      ADDZ h3 + b/2
    MATERIAL mats
    SPHERE b/2
 RETURN

! Script 2D

 IF curvo THEN
    PROJECT2 3, 270, 2
 ELSE
    CIRCLE2 0, 0, b/2
 ENDIF
```

Esercizio H

parametri:

lati (intero);
mar, **alt** (lunghezza);
matm, **mats**, **matc** (materiale)

```
! Gazebo
! Script Master

 ang   = 360/lati              ! ampiezza di ogni "spicchio"
 lato  = 2 * (a/2*TAN (ang/2)) ! dimensione di un lato
 latom = 2 * ((a/2+mar)*TAN (ang/2))  ! lato marciapiede
 latoc = 2 * ((a/2+0.30)*TAN (ang/2)) ! lato copertura
 xx    = a/2 + 0.30            ! posizione bordo copertura
 hm    = 0.15                  ! altezza marciapiede
```

```
   hp    = zzyzx - hm - alt      ! altezza pilastri
   rag   = 0.10                  ! raggio pilastri
   s     = 0.08                  ! spessore parapetto e copertura

! Script 3D - Gazebo

   FOR f = 1 TO lati
      MATERIAL matm
      PRISM_ 4, hm,
         0.00,     0.00 ,     0,
         a/2+mar, -latom/2 , 15,
         a/2+mar,  latom/2 ,  0,
         0.00,     0.00 ,    -1

      MATERIAL matc
      SLAB 3, s,
         0.00, 0.00, zzyzx-s,
         a/2+0.30,  -latoc/2, hp,
         a/2+0.30, latoc/2, hp

      MATERIAL mats
        ADDz hm
      IF f>1 THEN PRISM 4, 0.90,
         a/2,   -lato/2+rag,
         a/2,    lato/2-rag,
         a/2-s,  lato/2-rag,
         a/2-s, -lato/2+rag

        ADD a/2, lato/2, 0.00
      CYLIND hp, rag
        DEL 2

      ROTz ang
   NEXT f

   ADDZ zzyzx + 0.10
   MATERIAL "oro"
   SPHERE 0.10
   DEL 1

! Script 2D - Gazebo

   HOTSPOT2 0.00, 0.00
   FOR f = 1 TO lati
      LINE2 0.00, 0.00,        xx, -latoc/2
      LINE2 xx, -latoc/2,      xx, latoc/2
      LINE2 a/2+mar, -latom/2, a/2+mar, latom/2
      HOTSPOT2 xx,      latoc/2
      HOTSPOT2 a/2+mar, latom/2
      ROT2 ang
```

```
   NEXT f

   DEL TOP

! Script Parametri

   PARAMETERS b=a
   LOCK "B"
```

APPENDICE 02

Le parole del GDL

Quello che segue è un elenco di tutte le parole chiave del GDL. Questi termini non possono essere usati per identificare variabili o parametri. La maggior parte corrisponde a comandi o funzioni del linguaggio, altre (come ALL o RANGE) fanno parte della sintassi di altri comandi. Quelle segnate con un asterisco sono solo parole riservate, previste per l'utilizzo in versioni precedenti o future, ma non utilizzate nella versione corrente del GDL.

ABS	BRICK	DO	GOTO
ACS	BWALL_	DOORS	GROUP
ADD	CALL	DRAWINDEX	HATCHES
ADD2	CEIL	DRAWING	HIDEPARAMETER
ADDGROUP	CEILS	DRAWING2	HIP_ROOFS
ADDITIONAL_DATA	CIRCLE	DRAWING3	HOTARC2
ADDX	CIRCLE2	EDGE	HOTLINE2
ADDY	CLOSE	ELBOW	HOTSPOT
ADDZ	COLUMNS	ELLIPS	HOTSPOT2
ALL	COMPONENT	ELSE	HPRISM_
AND	CONE	EMPTY_FILL	IF
ARC	COONS	END	IND
ARC2	COOR	ENDGROUP	INPUT
ARMC	COS	ENDIF	INT
ARME	CPRISM_	ENDWHILE	ISECTGROUP
ASN	CROOF_	EXIT	ISECTLINES
ATN	CSLAB_	EXOR	KILLGROUP
BAS *	CUSTOM	EXP	LET
BASE	CUTEND	EXTRUDE	LGT
BASED_ON	CUTFORM	FILE_DEPENDENCE	LIGHT
BEAM	CUTPLANE	FILL	LIGHTS
BEAMS	CUTPOLY	FILLA	LIN *
BINARY	CUTPOLYA	FILLTYPES_MASK	LIN_
BINARYPROP	CUTSHAPE	FILTER *	LINE *
BITSET	CWALL_	FOR	LINE_PROPERTY
BITTEST	CYLIND	FPRISM_	LINE_TYPE
BLOCK	DATABASE_SET	FRA	LINE2
BODY	DEFINE	FRAGMENT2	LOCK
BOX *	DEL	GDLBIN *	LOG
BPRISM_	DESCRIPTOR	GET	MASS
BREAKPOINT	DIM	GOSUB	MATERIAL

MAX
MESH
MESHES
MIN
MOD
MODEL
MUL
MUL2
MULX
MULY
MULZ
NEXT
NOD *
NODE *
NOT
NSP
NTR
OBJECTS
OPEN
OR
ORIGO *
OUTPUT
PARAGRAPH
PARAMETERS
PARS *
PEN
PGON
PI
PICTURE
PICTURE2
PIPG
PITCHED_ROOFS
PLACEGROUP
PLANE
PLANE_
PLOTMAKER *
PLOTTER *
POLY
POLY_
POLY2
POLY2_
POLY2_A
POLY2_B
POSITION
PRINT
PRISM
PRISM_
PROJECT2
PUT
PYRAMID
RADIUS
RANGE
RECT
RECT_ *
RECT2
REF
REPEAT
REQ
REQUEST
RESOL
RETURN
REVOLVE
RICHTEXT
RICHTEXT2
RND
ROOMS
ROT
ROT2
ROTX
ROTY
ROTZ
ROUND_INT
RULED
SECT_FILL
SET
SFLINE *
SGN
SHADOW
SIN
SLAB
SLAB_
SOLID
SOLID_FILL
SPHERE
SPLINE2
SPLINE2A
SPLIT
SPRISM_
SQR
STEP
STR
STRLEN
STRSTR
STRSUB
STW
STYLE
SUBGROUP
SURFACE
SURFACE3D
SWEEP
SWEEPGROUP
SYMBOL_FILL
SYMBOL_LINE
TAN
TET *
TETRA *
TEVE
TEXT
TEXT2
TEXTBLOCK
THEN
TO
TOLER
TOP
TRI *
TUBE
TUBEA
UI_BUTTON
UI_CANCEL *
UI_DIALOG
UI_GROUPBOX
UI_INFIELD
UI_NEXT
UI_OK *
UI_OUTFIELD
UI_PAGE
UI_PICT
UI_PREV
UI_SEPARATOR
UI_STYLE
UNTIL
USE
VALUES
VARDIM1
VARDIM2
VARTYPE
VECT
VERT
VOCA *
VOLUME3D
WALL_ *
WALLHOLE
WALLNICHE
WALLS
WHILE
WINDOWS
WIRE
XFORM
XWALL_

SOMMARIO

LEZIONE 04

Analisi preliminare dell'oggetto / PRISM / MATERIAL/ Una variabile globale: SYMB_MAT/ PEN / Fare una scelta coi parametri booleani / CYLIND / Ancora rotazioni

LEZIONE 05

Ridimensionare gli assi: MUL, MULx, MULy, MULz / Riordinare e nascondere i parametri / CONE / Liste di valori: VALUES / Parametri di tipo testo / Uso di MULX -1

LEZIONE 06

Il 2D parametrico / POLY2 / Numerazione binaria applicata ai parametri / SET FILL / status/ Definire fori nelle campiture / POLY2_, POLY2_A, POLY2_B / DEFINE STYLE, SET STYLE, TEXT2 / Fragment: i lucidi del simbolo / FRAGMENT2 / Simboli che variano con la scala / Glob_Scale / Operatori relazionali / Operatori booleani / MIN, MAX

LEZIONE 07

Scomporre forme complesse / PRISM_ / Valori di status per i prismi / Definire fori nei prismi / CPRISM_, BPRISM_, FPRISM_, SPRISM_ / SPHERE, ELLIPS / ELBOW / RESOL, TOLER, RADIUS / SLAB, SLAB_, CSLAB

07.01. Ritorno al 3D
07.02. E i buchi?
07.03. Altre mutazioni
07.04. Ma la Terra non è piatta
07.05. Curve pericolose
07.06. Una vecchia pendenza
Esercizio F

LEZIONE 08

REVOLVE / Disattivare le ombre: SHADOW / TUBE / Piani di taglio: CUTPLANE / Codici di status addizionali / EXTRUDE / PYRAMID

08.01. La rivoluzione informatica
08.02. Un riposino all'ombra
08.03. Quel comando del tubo
08.04. Diamoci un taglio
08.05. Nuovi codici in arrivo
08.06. Qualche prisma perde la retta via
Esercizio G

LEZIONE 09

Gli usi del comando PRINT / Funzioni stringa: STR, SPLIT, STRLEN, STRSTR, STRSUB, STW / Cenni sulle forme piane del 3D: LIN_, RECT, POLY, POLY_, CIRCLE, ARC, PLANE, PLANE_ / Solidi complessi: RULED, SWEEP, MESH, COONS / MODEL SOLID, MODEL SURFACE, MODEL WIRE / BODY -1 / CUTPOLY, cenni su CUTPOLYA e WALLHOLE / La campitura nelle sezioni: SECT_FILL / TEXT / Cenni su REQ e REQUEST

09.01. Avvisi ai naviganti
09.02. Torniamo all'ABC
09.03. Non tutto il 3D è solido
09.04. Strane forme all'orizzonte
09.05. Pieno o vuoto?
09.06. Scripta manent
09.07. A grande richiesta
Esercizio H

LEZIONE 10

I sottotipi / Una Porta, una Finestra, una Lampada e un Timbro Zona / Cenni su altri cicli: WHILE, DO, REPEAT / Cenni sull'Interfaccia Utente / Gestione dei

parametri utente: LOCK, HIDEPARAMETER, PARAMETERS / Editazione grafica con gli Hotspot intelligenti / Uso del Buffer: PUT, GET, USE, NSP / Le operazioni booleane nel GDL: i gruppi

introduzione al

GDL

geometric description language

www.archiradar.it