Costruzioni di geometria euclidea mediante l'ambiente **picture** esteso

Claudio Beccari

Sommario

Il pacchetto curve2e è stato arricchito con gli aggiornamenti di LaTeX 2_ε disponibili dal 2019. Con queste ultime estensioni è possibile creare le macro necessarie per disegnare i problemi di geometria euclidea all'interno dell'ambiente picture esteso.

Abstract

The curve2e package has been enriched with the upgrades of LaTeX that took place in 2019. With such extensions it is possible to define the necessary macros to draw many problems of Euclidean geometry with the extended picture environment.

1 Introduzione

È chiaro che un programma di tipocomposizione deve poter comporre anche disegni al tratto per illustrare quanto viene descritto, specialmente, ma non solo, nei testi di tipo didattico concernenti diversi aspetti della matematica.

Fin dall'inizio, il formato LaTeX prevedeva un ambiente di base per disegnare semplici diagrammi al tratto. A quei tempi, i font vettoriali erano agli inizi e LaTeX non era attrezzato per gestirli; inoltre, i file composti erano in formato DVI (*DeVice Independent*) e quindi era impensabile richiedere a LaTeX di produrre file vettoriali scalabili con continuità, senza perdere qualità né nello scritto né nelle parti grafiche.

Nel 2003 è apparsa la prima "modernizzazione" suggerita dallo stesso Leslie Lamport nella seconda edizione del suo manuale (Lamport, 1994); in quell'anno due esperti di programmazione LaTeX hanno materialmente creato il codice col pacchetto di estensione pict2e, ora ulteriormente aggiornato (Gäßlein *et al.*, 2016). Nel 2006 ho reso disponibile il pacchetto curve2e, il cui ultimo aggiornamento risale al 2020 (Beccari, 2020a); si veda anche il nuovo manuale (Beccari, 2020b).

Oltre all'ambiente picture e alle sue estensioni, nell'arco degli anni sono apparsi diversi pacchetti per il disegno programmato e/o programmi esterni capaci di esportare i loro disegni vettoriali in modo da essere compatibili con la qualità tipografica di LaTeX. Fra i secondi vorrei citare il tradizionale programma `gnuplot` (Miklavec, 2013) con interfaccia letterale, e il più moderno `asymptote` Hammerlind *et al.* (2018) con la sua interfaccia grafica.

Fra i primi, direi che non ci sia nulla di migliore della collezione che passa sotto il nome di Portable Graphic Format (PGF) (Tantau, 2019), di cui Ti*k*Z e pgfplots sono gli esponenti maggiori, diventati ormai quasi lo standard nella comunità degli utenti del sistema TeX sparsi per il mondo. Figlio del PGF, esiste anche il pacchetto tkz-euclide (Matthes, 2020).

Per gli amanti di PostScript esiste anche la collezione di pacchetti accessibili principalmente con pstricks e la sua collezione di librerie per categorie di disegni particolari (Van Zandt, 2003). Si veda anche l'articolo di De Marco (2009) su `asymptote` e De Marco (2019), che fa una bella panoramica dei vari strumenti a disposizione per il disegno di alta qualità adatto ai documenti composti con i programmi di composizione del sistema TeX.

Ma c'è anche METAPOST (Hobby, 2018), derivato da METAFONT (Knuth, 1986), quest'ultimo costruito dal padre di TeX, D.E. Knuth, per disegnare i font che lui stesso ha usato fin dagli albori di TeX nel 1978.

Perché allora dedicarsi ad ulteriori estensioni del vecchio ambiente picture?

I motivi possono essere diversi a seconda dei punti di vista; ne cito solo alcuni.

1. Il manuale di Ti*k*Z è lunghissimo; certamente le funzionalità di Ti*k*Z sono quasi illimitate, e quindi c'è molto da descrivere; ma prima di essere produttivi, bisogna studiarne a lungo il mastodontico manuale.
2. METAPOST deriva da METAFONT; il secondo disegna caratteri in un suo formato a matrici di punti; il primo esegue disegni vettoriali, volendo anche caratteri, ma, a parte la presa in carico della stessa sintassi e logica di METAFONT, non direi che sia nato solo per disegnare caratteri. Anzi, credo che sia usato più spesso per eseguire altri disegni.
3. Il manuale di pict2e, la prima estensione dell'ambiente picture, ammonta ad una ventina di pagine; la sua ulteriore estensione curve2e ha un manuale d'uso di poco più lungo. Entrambi sono abbastanza essenziali, ma usano le funzionalità del sistema TeX per definire altre macroistruzioni, per cui le funzionalità dei pacchetti possono essere ulteriormente estese dagli utenti, in modo da poter disporre di comandi personalizzati.

4. Il LaTeX 3 Team sta procedendo velocemente con la definizione del nuovo linguaggio LaTeX 3 (sigla L3). Molti moduli sono già in versione beta; quindi l'interfaccia verso l'utente è sostanzialmente stabile, anche se i dettagli delle definizioni interne possono ancora cambiare e/o le funzionalità possono aumentare.
   Fra questi moduli nuovi ce ne sono due, xparse e xfp, con caratteristiche particolarmente avanzate; il primo serve per estendere ulteriormente le definizioni di comandi e ambienti nuovi in modo da creare comandi con interfacce utente estremamente versatili. Il secondo mette a disposizione la libreria di calcoli decimali a virgola mobile (*floating point*, da cui deriva la parte 'fp' nel nome del pacchetto).
   La documentazione di xparse è completa; quella di xfp è molto (forse troppo) succinta, ma, se si legge la parte XXIII di `interface3.pdf`, si riesce a capire quali siano le funzionalità del pacchetto.
5. Il pacchetto xpicture (Fuster, 2012) è eccellente ed estende moltissimo l'ambiente picture; esso stesso usa i pacchetti pict2e e curve2e ma aggiunge molte funzionalità. In un certo senso potrebbe rimpiazzare tutto quello che viene esposto in questo articolo; ma, secondo me, la sua sintassi è un po' complessa anche se molto efficace. D'altra parte qui voglio mostrare come usare la grafica dell'ambiente picture, estesa con le funzionalità di curve2e, per costruire le procedure geometriche che consentono di risolvere graficamente alcuni problemi di geometria euclidea, con metodi non dissimili da quelli che una volta si sviluppavano con riga, squadra e compasso.

Stimolato da un articolo di Estevão Candia (Candia, 2019) su *TUGboat*, ho provato a vedere se quello che lui descrive nell'articolo, sinossi della sua tesi magistrale (Candia, 2018), sviluppato con METAPOST, si potesse fare con l'ambiente picture, quello esteso o estensibile come descritto sopra. Mi ci sono cimentato e ho ottenuto risultati simili, se non migliori, proprio con l'ambiente picture esteso. Inizialmente ho avuto qualche difficoltà perché nessuna delle estensioni di quell'ambiente consente di risolvere equazioni o sistemi di equazioni, sia pure solo lineari. METAPOST è nato per farlo, oltre che per disegnare; quindi, ovviamente, con METAPOST è marginalmente più facile ottenere i risultati desiderati, ma prima di essere capaci di usare correttamente METAPOST bisogna impararne il suo linguaggio che, però, non è semplice. Nel dir questo mi metto nei panni del normale utente di LaTeX che potrebbe trovare delle difficoltà.

Nel contempo ho imparato molte cose; per me quindi è stato utile; ma penso che sia utile anche agli insegnanti di scuola secondaria superiore che scrivono con LaTeX gli appunti o le dispense per i loro allievi. Forse possono essere utili anche per i docenti delle matematiche propedeutiche dei corsi universitari di laurea triennale.

Per questo motivo ho messo a disposizione il nuovo pacchetto euclideangeometry (Beccari, 2020c), che dispone di un manuale utente dettagliato (Beccari, 2020d) dove, tra l'altro, sono estesamente ripresi e commentati alcuni degli esempi mostrati in questo articolo. Il pacchetto dovrebbe già fare parte di ogni installazione completa e aggiornata di TeX Live e MiKTeX. L'utente interessato non ha altro che da caricare questo nuovo pacchetto, il quale provvede da solo a caricare curve2e, di cui è un'estensione; curve2e a sua volta carica pict2e cosicché la catena di estensioni dell'ambiente picture è completa.

Attenzione! Questo nuovo pacchetto euclideangeometry richiede l'uso dell'ultima versione di curve2e, con la data dell'ultima revisione non anteriore al 2020-01-18. In mancanza di questo requisito euclideangeometry emette un vistoso messaggio d'errore e termina il suo stesso caricamento oltre che il processo di compilazione. In sostanza questo pacchetto funziona solo con un'installazione completa e aggiornata del sistema TeX.

Questo articolo è diviso in due parti. La prima, di carattere generale, descrive i comandi basati sul pacchetto xfp che sono già incorporati nell'ultima versione di curve2e; sono già documentati nella descrizione d'uso (Beccari, 2020b) e in quella del codice di curve2e (Beccari, 2020a), ma merita descriverli succintamente anche qui come esempi di nuovi comandi per gestire i numeri in virgola mobile e per gestire i test da eseguire su tali numeri. Nella seconda parte descriverò come realizzare certi disegni geometrici che riguardano principalmente triangoli, poligoni regolari, circonferenze ed ellissi, in particolare quelle che si inscrivono nei triangoli.

2 Upgrade di curve2e

2.1 Retrocompatibilità

Da quando è divenuto disponibile il pacchetto xfp, curve2e è stato modificato per farne largo uso; però esiste un certo numero di utenti che non amano, o non sono interessati, o non possono caricarsi versioni più aggiornate del software del sistema TeX; i motivi possono essere svariati e giustificabili. Però aggiornare un pacchetto che si basa su funzionalità apparse verso la fine della validità di TeX Live 2018, avrebbe significato l'impossibilità di ricompilare vecchi documenti, che richiedono versioni precedenti del nucleo di LaTeX. Non è frequente, ma succede. Per questo curve2e ha un test iniziale che verifica la disponibilità di xfp e, se questo non è disponibile, carica la sua versione precedente a questo *upgrade*. Per ricompilare vecchi documenti questo è più che sufficiente. Ma se si vogliono sperimentare le macro descritte in questo articolo, è necessario servirsi di

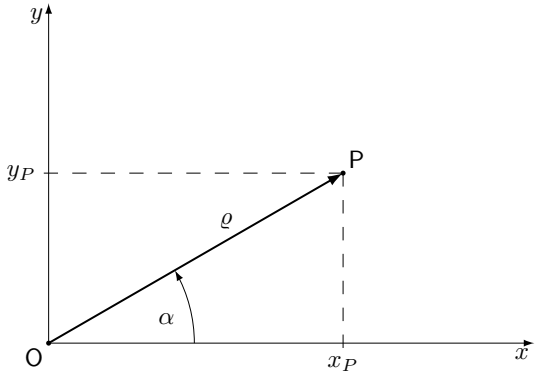

Figura 1: Coordinate cartesiane e polari

una versione aggiornata e completa di TEX Live o di MiKTEX.

2.2 Le estensioni attuali di curve2e

La particolarità di curve2e è che le coppie ordinate che vengono usate per indicare i punti del piano euclideo, o le direzioni in tale piano, sono trattate come numeri complessi. In realtà è solo una questione di terminologia; anche METAFONT, prima, e METAPOST, dopo, usano le coppie ordinate e le gestiscono come se fossero numeri complessi. Matematici e ingegneri hanno qualche differenza nella nomenclatura, e usano due terminologie diverse, ma per gli ingegneri i numeri complessi sono visti preferibilmente come vettori o operatori vettoriali; qui non mi riferirò alle loro terminologie, né alle proprietà di vettori, versori, operatori vettoriali, coppie ordinate, né alle classi di equipollenza e altre nomenclature relative a questi oggetti; tuttavia mi sarà comodo usare la qualifica di "complesso e coniugato" per certi numeri complessi e riferirmi all'"angolo" dei numeri complessi col nome di *argomento*.

La versione di curve2e disponibile in questo momento in cui sto scrivendo questo testo è successiva alla 2.0.8 del 2020-01-18. In quella versione, e nelle successive, tutte le macro disponibili per l'utente hanno la possibilità di scrivere le coordinate dei punti e le direzioni di segmenti e vettori indifferentemente mediante le coordinate cartesiane o polari (vedi la figura 1), racchiuse fra le normali parentesi tonde o senza delimitatori a seconda della sintassi dei vari comandi. La sintassi è la seguente:

$$\mathsf{P} = \begin{cases} x, y & \text{coordinate cartesiane} \\ \alpha{:}\,\varrho & \text{coordinate polari} \end{cases}$$

Inoltre, l'utente può definire macro per designare determinati punti del disegno con dei nomi che gli siano comodi, mnemonicamente legati a un particolare punto. Se se ne fa un uso corretto, le varie macro interne non interferiscono con i nomi usati dall'utente.[1] Di ogni macro che individua un punto si possono estrarre le componenti cartesiane o le componenti polari o trasformare le une nelle altre. Internamente, le macro di servizio usano quasi esclusivamente le coordinate cartesiane.

Per fare queste cose, le funzionalità che il pacchetto mette a disposizione, documentate nella descrizione del pacchetto xfp, sono quelle che ci si aspetta da una qualunque calcolatrice tascabile o dalle *app* del sistema operativo che abbiano la modalità *scientifica*; quindi, oltre alle quattro operazioni e alle radici, sono disponibili le funzioni circolari dirette e inverse, gli esponenziali e i logaritmi e altre utili funzioni; le funzioni circolari, con nomi diversi, accettano o i radianti o i gradi; in curve2e si accettano solo i gradi e si usano solo le funzioni trigonometriche per gestire i gradi.

Quello che non è citato nella documentazione di xfp consiste nel fatto che per il calcolo dell'arcotangente accetta sia un solo argomento, sia una coppia di valori separati da una virgola, in modo che può calcolare il valore (in radianti o) in gradi sia nell'intervallo $-90° < \alpha \leq 90°$ (se viene dato un solo argomento), o nell'intervallo $-180° < \alpha \leq 180°$, se vengono specificati due valori.[2] Lavorando con i numeri complessi, curve2e non ha più bisogno di fare lunghi test per calcolare l'argomento del numero a seconda del quadrante in cui si trova.

Dopo queste importanti modifiche, tutte le macro di curve2e per eseguire operazioni sui numeri complessi vengono eseguite correttamente con le nuove funzionalità; le macro assumono una forma facilmente decifrabile e perciò comode da curare nel caso in cui si manifestassero inconvenienti di qualche tipo. Ma anche le operazioni sulle lunghezze o in generale sui numeri reali sono molto più chiare e immediate. La divisione, in particolare, non solo si riduce a un'unica funzione del linguaggio L3, ma nemmeno richiede acrobazie per evitare gli overflow che con la versione precedente si manifestavano raramente, ma non erano esclusi. Anche la radice quadrata è formata da una sola funzione L3, e non è necessario ricorrere a nessun procedimento iterativo; prima, invece, era necessario ricorrere al procedimento di Newton.

Questo non vuol dire che ora il tempo di esecuzione sia diminuito; infatti, l'implementazione delle funzioni per il calcolo dei numeri decimali in virgola mobile e l'analisi delle espressioni da calcolare richiede non poche elaborazioni dietro le quinte, tutte codificate rigorosamente in linguaggio L3. Tuttavia, oggi gli elaboratori sono così veloci che il tempo di calcolo ammonta a pochi millisecondi, quindi generalmente si tratta di tempi del tutto trascurabili. Ciò che oggi "rallenta" l'elaborazione

1. In verità la macro `\Pi`, che mi sarebbe piaciuto usare per etichettare un "punto iniziale" non può essere usata dall'utente, perché L3 usa questa macro per indicare una certa versione del numero "pi greco".

2. Attenzione: il primo argomento riguarda l'asse verticale e il secondo l'asse orizzontale. Tuttavia, esiste anche la funzione arcocotangente che scambia di posizione i due valori, quindi basta solo essere attenti a usare la funzione giusta. Insomma, se è $P = (x, y)$, l'argomento di P si può calcolare sia con $\arctan(y/x)$, sia con $\operatorname{arccot}(x/y)$.

di un documento sono principalmente le operazioni di lettura e scrittura su disco, anche se la macchina è dotata di un disco allo stato solido. Proprio per questo, l'esecuzione di alcuni disegni eseguiti con le nuove macro talvolta può richiedere alcuni secondi, ma è quasi tutto tempo impegnato nelle operazioni di *paging*, cioè nel caricare e scaricare ripetutamente la memoria virtuale del calcolatore. Di solito questi improvvisi rallentamenti derivano da registri di memoria troppo pieni legati a oggetti flottanti di grandissime dimensioni; questo non succede solo con questo pacchetto, ma accomuna anche T*i*kZ, tanto per citare un esempio.

Quello che il pacchetto xfp oggi non fornisce ancora, sono i comandi per gestire i test e per eseguire cicli; per questo nella nuova versione di curve2e compaiono anche le righe seguenti scritte con l'interfaccia per il linguaggio L3:

```
\ExplSyntaxOn
\AtBeginDocument{%
\ProvideExpandableDocumentCommand\fptest%
    {m m m}{\fp_compare:nTF{#1}{#2}{#3}}
\ProvideExpandableDocumentCommand\fpdowhile%
    {m m}\fp_do_while:nn{#1}{#2}
}
\ExplSyntaxOff
```

Le loro sintassi sono le seguenti:

```
\fptest{⟨test⟩}⟨vero⟩⟨falso⟩
\fpdowhile{⟨test⟩}⟨operazioni da ripetere⟩
```

Nel primo comando si esegue un ⟨*test*⟩ su espressioni relazionali (a loro volta con operandi costituiti da numeri interi o fratti e operatori di relazione come `>, <, =`) legate con gli operatori logici `||` (OR), `&&` (AND), `!` (NOT) e altri operatori meno comuni. Sono previste entrambe le uscite per eseguire il contenuto del primo o del secondo argomento che segue il test a seconda che il test risulti vero o falso.

Nel secondo comando indicato sopra si esegue un ciclo `do while`; nel primo argomento va il ⟨*test*⟩ da eseguire per controllare le iterazioni, mentre nel secondo, fra le ⟨*operazioni da ripetere*⟩ ciclicamente, deve esserci anche la modifica di qualcosa che influisca sul ⟨*test*⟩, affinché esso diventi prima o poi falso ed eviti un ciclo infinito. È evidente che prima di attivare questa funzione L3, bisogna impostare i parametri da cui dipende il ⟨*test*⟩ in modo che questo sia inizialmente vero.

I comandi che più si sono giovati delle nuove funzionalità sono quelli per le operazioni ripetitive come `\multiput` e il nuovo `\xmultiput`. Il primo è perfettamente retrocompatibile con la versione originale, ma la sua definizione consente di fare cose impossibili con quella versione. La sintassi è

```
\multiput[⟨shift⟩](⟨inizio⟩)(⟨incremento⟩)
    {⟨numero⟩}{⟨oggetto⟩}[⟨operazioni⟩]
```

Infatti tolti il primo e l'ultimo argomento facoltativi, la sintassi è identica a quella del comando originale; lo ⟨*shift*⟩ è una coppia ordinata di valori che serve per spostare il disegno prodotto da `\multiput` di quanto specificato con quel numero complesso. Le ⟨*operazioni*⟩ permettono di modificare l'incremento del valore specificato alle funzioni L3 che si possono usare in questo argomento. Il comando `\xmultiput` ha una sintassi del tutto simile, ma il controllo delle iterazioni è eseguito in modo diverso e più adatto per certi scopi; si veda per esempio la figura 5 nella pagina 9.

Le preesistenti macro dell'ambiente curve2e `\Dashline` e `\Dotline` sono ridefinite usando queste nuove modalità di calcolo, e certi inconvenienti che prima si manifestavano non si manifestano più.

Nelle figure 2 e 3 si vedono dei fasci di linee tratteggiate e punteggiate descritte con le direzioni espresse in forma sia cartesiana sia polare.

Il comando `\multiput` si avvale di un contatore interno definito con i nomi di TEX, mentre i nomi dei contatori di LATEX sono preceduti dalla stringa `c@` prima di poter accedere ai contatori TEX. La macro `\multiput` contiene al proprio una ridefinizione del contatore in modo da renderlo accessibile alle ⟨*operazioni*⟩ attraverso il nome `multicnt` mediante i comuni comandi LATEX come, per esempio, `\value`.[3]

Nella figura 4 appaiono diversi esempi di uso del comando `\multiput`.

Si noti che le macro `\R` e `\D` sono macro interne a `\multiput` e indicano rispettivamente la posizione incrementale e l'incremento ciclico del processo iterativo. Nei primi due esempi si è usato il comando `\multiput` nel modo tradizionale; nel terzo esempio si è usato anche il primo argomento facoltativo del nuovo comando `\multiput` per dislocare lo stesso codice del primo esempio di 5 unità in orizzontale e altrettanto in verticale; si è usato un quadratino invece di un dischetto in modo da mostrare la differenza di posizione.

Nel quarto esempio si è usato un incremento in coordinate polari; esso contiene un angolo di rotazione di 30° ma ha un modulo unitario; quindi può essere usato come operatore di rotazione; le stelline sono quindi collocate lungo un cerchio a distanza di 30° l'una dall'altra, ma per ottenere questo effetto si è usato il secondo argomento facoltativo dove la posizione `\R` viene ruotata moltiplicandola per il numero complesso di rotazione `\D`, in modo che il successivo ⟨*oggetto*⟩, la stellina, sia collocata lungo il cerchio. Se l'incremento non avesse avuto il modulo unitario, le stelline sarebbero state collocate lungo un arco di spirale.

3. Bisogna ricordarsi però di *non* usare i comandi LATEX `\setcounter`, `\addtocounter` e simili, perché sono comandi globali, cioè, sono comandi il cui effetto non è soggetto al limite del gruppo, o dell'ambiente, o della scatola e influisce sull'intero documento.

```
\begin{picture}(40,30)
\AutoGrid
\Dashline(0,0)(40,10){2}\Dashline(0,0)(40,20){2}
\Dashline(0,0)(40,30){2}\Dashline(0,0)(30,30){2}
\Dashline(0,0)(20,30){2}\Dashline(0,0)(10,30){2}
\Dashline(40,0)(108:30){2}
\Dashline(40,0)(126:30){2}
\Dashline(40,0)(144:30){2}
\Dashline(40,0)(162:30){2}
\end{picture}
```

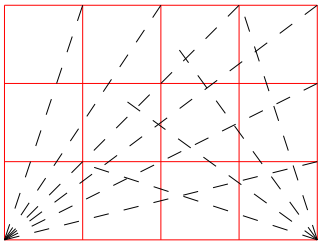

FIGURA 2: Alcune linee tratteggiate con l'⟨*incremento*⟩ espresso in forma cartesiana o polare

```
\begin{picture}(40,30)
\AutoGrid
\Dotline(0,0)(40,10){1.5}[2]\Dotline(0,0)(40,20){1.5}[2]
\Dotline(0,0)(40,30){1.5}[2]\Dotline(0,0)(30,30){1.5}[2]
\Dotline(0,0)(20,30){1.5}[2]\Dotline(0,0)(10,30){1.5}[2]
\Dotline(40,0)(108:30){1.5}[2]
\Dotline(40,0)(126:30){1.5}[2]
\Dotline(40,0)(144:30){1.5}[2]
\Dotline(40,0)(162:30){1.5}[2]
\end{picture}
```

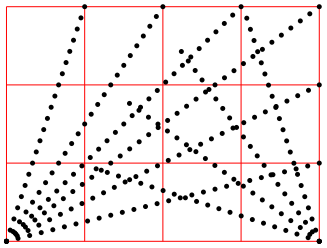

FIGURA 3: Alcune linee punteggiate con l'incremento espresso in forma cartesiana o polare

```
\unitlength=0.01\textwidth
\begin{picture}(100,100)
\AutoGrid
\multiput(0,0)(10,10){10}{\circle*{1.5}}
\multiput(0,10)(10,10){10}{\circle*{1.5}}
\multiput[4,4](0,0)(10,10){10}
   {\polygon*(0,0)(2,0)(2,2)(0,2)}
\multiput[50,50](0,30)(30:1){12}%
   {\makebox(0,0){\color{blue}\Huge$\star$}}
   [\MultVect\R by\D to\R]%
\multiput[50,50](90:36)(30:1){12}%
   {\makebox(0,0){\Roman{multicnt}}}%
   [\Multvect{\R}{\D}{\R}]
\end{picture}
```

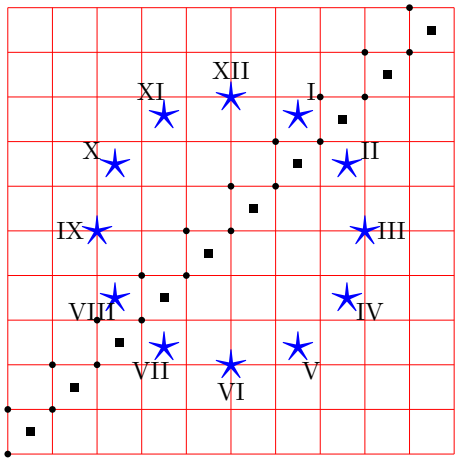

FIGURA 4: Alcuni esempi d'uso del comando `\multiput`

Il quinto esempio è del tutto simile al quarto per quel che riguarda la posizione dell'oggetto, che però questa volta è il contatore di iterazione contenuto nel contatore TEX `\multicnt` camuffato da contatore LATEX, così da potergli applicare la trasformazione in numeri romani maiuscoli attraverso il comando `\Roman`.

Sono esempi banali, tuttavia mostrano le nuove funzionalità del comando `\multiput`.

Il comando `\xmultiput` differisce dal comando standard nel fatto che il ciclo iterativo è eseguito e controllato dalla funzione L3 `\fpdowhile` definita precedentemente. L'utente è più libero nel controllare la posizione dell'oggetto e, volendo, può anche usare i comandi di disegno di basso livello come `\moveto`, `\lineto` e soci, per disegnare poligoni regolari e/o curve continue a tratti; lavorando con punti sufficientemente vicini, si possono disegnare anche curve mediante spezzate che sembrano delle curve "dolci"; certo, i punti devono essere piuttosto vicini per ottenere un effetto gradevole. Nella figura 5 è mostrato il codice e il risultato di questa operazione per disegnare un arco di iperbole.

Il trucco di definire per nome un registro numerico, qui identificato con il numero 2560 [4], rende le varie operazioni molto più semplici da leggere; il contatore identificato con `\I`, usciti dall'ambiente picture, non è più accessibile con quel nome e, se precedentemente conteneva un valore, esso viene ripristinato col valore precedente.

Questo modo di usare i nuovi comandi in linguaggio L3, rende anche agevole disegnare curve

4. Con le versioni moderne del sistema TEX non è vietato usare numeri superiori a 255.

```
\unitlength =0.008\linewidth
\begin{picture}(100,100)
\AutoGrid
\VECTOR(0,0)(100,0)\Pbox(100,0)[tr]{x}[0]
\VECTOR(0,0)(0,100)\Pbox(0,100)[tr]{y}[0]
\Pbox(0,0)[r]{O}[3pt]
\thicklines
\moveto(10,100)\countdef\I=2560 \I=11
\xmultiput(0,0)(1,0){101}%
  {\lineto(\I,\fpeval{1000/\I})}%
  [\advance\I by1 \value{multicnt}\I]
\strokepath
\end{picture}
```

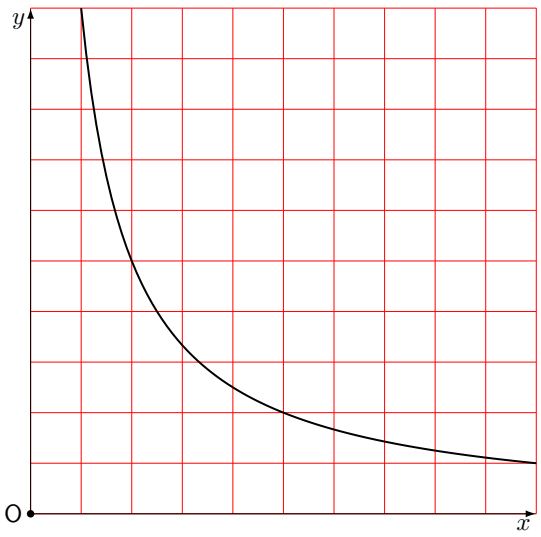

Figura 5: Disegno di un'iperbole mediante una spezzata

definite con equazioni parametriche. A titolo di esempio, si traccia una curva a forma di cuore[5], le cui equazioni parametriche sono le seguenti[6]:

$$x(t) = \sin^3(t)$$
$$y(t) = \frac{13\cos(t) - 5\cos(2t) - 2\cos(3t) - \cos(4t)}{16}$$

Conviene definire una macro tale che, ricevuto in ingresso il valore del parametro, produca in uscita il numero complesso $P = (x, y)$; bisogna anche che tenga conto delle scale del disegno, mediante un coefficiente di scala `\scala`; conviene che l'intero disegno sia centrato anche verticalmente.

```
\newcommand\cuore[3]{%
\edef\X{\fpeval{#1*16*(sind(#2)^3)}}
\edef\Y{\fpeval{#1*(13*cosd(#2) - 5*cosd(2*#2)
  - 2*cosd(3*#2) -cosd(4*#2)+2.4)}}
\CopyVect\X,\Y to#3}
```

In questa macro il primo argomento riceve il fattore di scala, il secondo il valore del parametro e il terzo produce in uscita una macro che contiene il numero complesso calcolato: la costante 2.4 che compare nel codice serve per spostare il disegno in alto in modo che sia centrato.

Ciò fatto, il disegno del diagramma e il suo codice sono mostrati nella figura 6.

5. Quanto qui esposto è una possibile soluzione ad un quesito posto sul forum del GuIt. In quel quesito si indicava il sito https://tex.stackexchange.com/questions/139733/can-we-make-a-love-heart-with-latex, dove sono state date molte soluzioni, una più bella dell'altra, ma nessuna eseguita con curve2e. Nella risposta che ho dato nel forum, ho mostrato una soluzione ottenuta con la macro `\Curve` (con o senza asterisco) che però è un disegno eseguito con spline di Bézier, non il tracciamento della curva parametrica come fatto qui.

6. Vedi http://mathworld.wolfram.com/HeartCurve.html, che contiene le equazioni di diversi "cuori" di forme diverse

Si noti che in questo caso il parametro indica il valore dell'angolo in gradi; infatti, lavorando con i numeri interi si è sicuri di far variare il parametro esattamente da 0° a 360°; inoltre il passo di incremento del parametro viene fissato a 3°; l'esecuzione del disegno avviene in modo decisamente più rapido rispetto al tempo che si avrebbe scegliendo un valore più piccolo, per esempio 1°; nello stesso tempo la figura non mostra le giunzioni fra i brevi segmenti che formano la spezzata. Infine, se al posto di `\strokepath`, si usa `\fillpath`, quel contorno viene riempito con il colore corrente; l'impostazione del colore va fatta prima di attivare il comando `\fpdowhile`.

2.3 Il comando \Pbox

I vari esempi mostrati fino a questo punto mostrano l'uso di alcune delle funzionalità presenti nell'ultima versione del pacchetto curve2e. Il lettore può vedere qui come sia definito quel comando `\Pbox` che serve per mettere l'etichetta agli assi e agli elementi geometrici che compaiono nelle figure:

```
\providecommand\Pbox{}
\RenewDocumentCommand\Pbox
  {D(){0,0} O{cc} m O{0.5ex}}{%
\put(#1){%
\dimendef\Dim=2566\relax
\settowidth\Dim{#2}%
\edef\Rapp{\fpeval{\Dim/{1ex}}}%
\fptest{\Rapp > 1.5}%
{\fboxsep=0.5ex}{\fboxsep=0.75ex}%
\fboxrule=0pt
\fptest{#4 = 0sp}%
  {\makebox(0,0)[#2]{\fbox{$\relax#3\relax$}}}%
  {\edef\Diam{\fpeval{#4/\unitlength}}%
   \makebox(0,0){\circle*{\Diam}}%
   \makebox(0,0)[#2]{%
     \fbox{$\relax\mathsf#3\relax$}}}%
}\ignorespaces}
```

La sua versatile sintassi è la seguente:

```
\unitlength=0,004\linewidth
\begin{picture}(200,200)(-100,-100)
\AutoGrid
\VECTOR(-100,0)(100,0)\Pbox(100,0)[br]{x}[0]
\VECTOR(0,-100)(0,100)\Pbox(0,100)[tl]{y}[0]
\Pbox(0,0)[tr]{O}\linethickness{1pt}\bgroup
\edef\scala{\fpeval{100/16}}
\countdef\I=2560 \I=0\roundjoin
\fpdowhile{\I !>360}{\cuore\scala\I\Punto
\ifnum\I=0 \moveto(\Punto)\else \lineto(\Punto)\fi
\advance\I by 3}\strokepath\egroup
\end{picture}
```

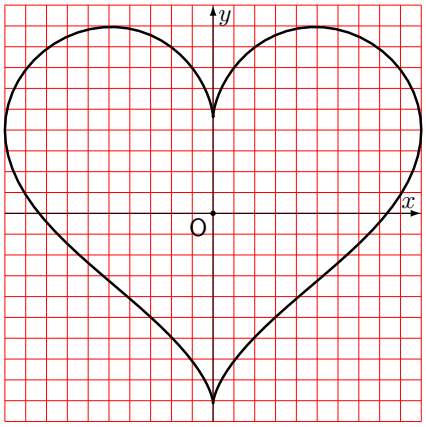

Figura 6: Disegno di una curva a forma di cuore

`\Pbox(`⟨*posizione*⟩`)[`⟨*allineamento*⟩`]%`
`{`⟨*etichetta*⟩`}[`⟨*diametro*⟩`]`

dove la ⟨*posizione*⟩ indica dove verrà collocato il pallino che identifica l'oggetto da etichettare; se il pallino ha un ⟨*diametro*⟩ nullo, è invisibile; in ogni caso, il ⟨*diametro*⟩ viene specificato con unità di misura esplicite[7] e ci pensa la macro a trasformarle in multipli di `\unitlength`; l'⟨*etichetta*⟩ viene sempre composta in modo matematico, ma se il ⟨*diametro*⟩ è non nullo, essa identifica un oggetto non misurabile e, secondo le norme ISO, viene usato il carattere senza grazie; se la si vuole in modo testo, bisogna racchiuderla esplicitamente fra due segni di dollaro, per passare dal modo matematico predefinito al modo testo; l'⟨*allineamento*⟩, formato dai soliti codici `c` (*center*), `t` (*top*), `b` (*bottom*), `r` (*right*), `l` (*left*), specifica (di fatto) la *posizione* del pallino rispetto all'etichetta; il doppio allineamento centrato (verticale e orizzontale) predefinito, serve per produrre un risultato inaccettabile, che segnala all'utente l'obbligo di specificare i propri codici di posizionamento.

`tl`	`t`	`tr`	`r`	`l`	`bl`	`b`	`br`
NW	N	NE	E	W	SW	S	SE

I punti cardinali indicati sotto i codici forse sono più espliciti per indicare la ⟨*posizione*⟩ del pallino rispetto all'etichetta.

3 Poligoni regolari, triangoli ed ellissi

Le macro e le estensioni descritte nel paragrafo precedente tornano utili per disegnare curve di vario genere e per eseguire le costruzioni con riga, squadra e compasso tipiche della geometria euclidea.

7. Volendo si può omettere l'unità di misura, nel qual caso il valore è assunto come misurato in punti tipografici.

3.1 Poligoni regolari

Per disegnare poligoni regolari col solo contorno o ripieni di colore, variamente orientati, con i lati di spessore diverso da quello predefinito, si può definire una macro che faccia tutto in un colpo solo. Eccone il codice.

```
\NewDocumentCommand\RegPolygon%
  {s D(){0,0}  m m O{0} D<>{\relax} }{{%
\countdef\I=258 \I=0
\CopyVect#5:#3to\P
\CopyVect\fpeval{360/#4}:1to\R
\put(#2){#6\relax
  \moveto(\P)\fpdowhile{\I < #4}%
  {\MultVect\P by\R to\P
  \lineto(\P)\advance\I by 1}%
  \IfBooleanTF{#1}%
  {\fillpath}{#6\strokepath}}}\ignorespaces}
```

La sintassi della macro è la seguente:

`\RegPolygon`⟨*⋆*⟩`(`⟨*centro*⟩`){`⟨*raggio*⟩`}%`
`{`⟨*numero*⟩`}[`⟨*angolo*⟩`]<`⟨*impostazioni*⟩`>`

Il primo asterisco è facoltativo; se c'è, l'area racchiusa dal poligono viene colorata con il colore corrente, oppure vengono usati il colore e/o gli spessori specificati con le ⟨*impostazioni*⟩; si possono specificare le coordinate del ⟨*centro*⟩, ma se non lo si fa, il centro viene messo nell'origine degli assi. Bisogna evidentemente specificare il ⟨*raggio*⟩ del cerchio circoscritto e il ⟨*numero*⟩ di lati; facoltativamente si può specificare (in gradi) l'angolo di rotazione del primo vertice; infine, racchiuse fra i segni `< >`, si possono indicare le ⟨*impostazioni*⟩ che si desiderano. Se si vuole il perimetro del poligono disegnato con un colore diverso da quello del suo interno, bisogna sovrapporre il poligono col solo contorno al poligono colorato internamente. La figura 7 contiene diversi esempi; essa è ottenuta con il codice seguente:

```
\unitlength=0.006\linewidth
\begin{picture}(120,100)
```

```
%
\RegPolygon(9,20){20}{6}%
  <\linethickness{3pt}\color{red}>
\RegPolygon(55,20){20}{7}[90]
\RegPolygon(100,20){20}{8}[22.5]%
  <\linethickness{0.5ex}\color{blue}>
\put(0,50){%
\RegPolygon(20,20){20}{3}
\RegPolygon(20,20){20}{3}[30]
\RegPolygon(20,20){20}{3}[60]
\RegPolygon(20,20){20}{3}[90]
%
\RegPolygon*(62,20){20}{4}<\color{green}>
\RegPolygon(62,20){20}{4}%
  <\linethickness{1ex}>
%
\RegPolygon*(100,20){20}{4}[45]%
  <\color{orange}>
\RegPolygon(100,20){20}{4}[45]%
  <\linethickness{1ex}\color{blue}>
}
\end{picture}
```

3.2 Ellissi

Il pacchetto curve2e non contiene macro per disegnare ellissi. In realtà, il loro disegno di base è semplicissimo; il codice è il seguente:

```
\def\ellisse#1#2{%
\bgroup\edef\ELa{#1}\edef\ELb{#2}%
\edef\ELx{\fpeval{4*(sqrt(2)-1)/3}}%
\edef\ELax{\fpeval{\ELx*\ELa}}%
\edef\ELbx{\fpeval{\ELx*\ELb}}%
\moveto(\ELa,0)
\curveto(\ELa,\ELbx)(\ELax,\ELb)(0,\ELb)
\curveto(-\ELax,\ELb)(-\ELa,\ELbx)(-\ELa,0)
\curveto(-\ELa,-\ELbx)(-\ELax,-\ELb)(0,-\ELb)
\curveto(\ELax,-\ELb)(\ELa,-\ELbx)(\ELa,0)
\fillstroke\egroup\ignorespaces}
```

Questo codice funziona benissimo, ma non è sufficientemente versatile; in particolare, non lo è per gli scopi di questo articolo. È solo usato internamente da `\Xellisse`, che ne imposta tutte le caratteristiche e, in particolare, specifica il significato della macro `\fillstroke`; questa viene in effetti posta equivalente a `\strokepath` quando si vuole disegnare solo il contorno, oppure a `\fillpath` quando si vuole riempire il contorno con un colore. La macro per l'utente è la seguente:

```
\NewDocumentCommand{\Xellisse}%
  { s D(){0,0} O{0} m m O{} o}%
    {\IfBooleanTF{#1}%
      {\let\fillstroke\fillpath}%
      {\let\fillstroke\strokepath}%
    \put(#2){\rotatebox{#3}{{#6\ellisse{#4}{#5}}%
    \IfValueTF{#7}{\let\fillstroke\strokepath
    #7\ellisse{#4}{#5}}{}}}%
  }
```

I suoi sette argomenti permettono di fare qualunque cosa con l'ellisse; sia per usare i colori sia per disegnarne il contorno; a differenza dal comando `\RegPolygon`, non richiede all'utente di disegnare due ellissi per disegnarne il contorno sovrapposto al solo interno colorato, perché fa tutto da solo; anzi, questo codice potrebbe essere preso a modello per ridefinire `\RegPolygon` perché si comporti nello stesso modo. La sua sintassi è la seguente:

```
\Xellisse⟨⋆⟩(⟨centro⟩)[⟨angolo⟩]%
    {⟨semiassse-a⟩}{⟨semiasse-b⟩}%
    [⟨impostazioni-1⟩][⟨impostazioni-2⟩]
```

L'asterisco facoltativo controlla il riempimento di colore; se manca, viene disegnato solo il contorno e le ⟨*impostazioni-2*⟩, che normalmente ricevono le impostazioni per il contorno, possono essere omesse in quanto collocabili anche nel campo ⟨*impostazioni-1*⟩; viceversa, se l'asterisco è presente, le ⟨*impostazioni-1*⟩ riguardano il riempimento, mentre le ⟨*impostazioni-2*⟩, facoltative, riguardano il contorno. Per l'ellisse si deve esplicitare il ⟨*centro*⟩, si può specificare l'⟨*angolo*⟩ di rotazione dell'intera ellisse; ovviamente si devono specificare il semiasse principale (maggiore) ⟨*semiasse-a*⟩ e quello secondario (minore) ⟨*semiasse-b*⟩, anche se, in realtà, la macro non fa differenze fra i due semiassi: il primo è quello che, senza rotazione, è orizzontale e il secondo quello verticale. Le impostazioni sono già state descritte prima. Si fa notare che queste impostazioni di colore per il riempimento e il bordo sono utili in altri contesti, non solo nelle costruzioni geometriche; l'uso è già stato descritto in un altro articolo (Beccari, 2018a). Qui quello che interessa è l'oggetto geometrico "ellisse", più che le sue decorazioni; tuttavia, anche nelle applicazioni di questo articolo, non è fuori luogo impostare lo spessore del contorno.

Per tornare a questo articolo è meglio usare una macro un po' più complessa, ma che permetta di operare trasformazioni affini sull'ellisse; fra queste compaiono gli spostamenti, le rotazioni e gli scorrimenti, *shear* in inglese[8]. Lo scorrimento di una griglia di rette incrociate ad angolo retto, diventa una griglia di rette che si incrociano con un angolo diverso; per esemplificare con una figura geometrica comune, un rettangolo diventa un parallelogramma con angoli interni diversi da 90°. Non ci sono problemi con gli spostamenti, perché basta specificare le coordinate del ⟨*centro*⟩; non ci sono problemi con le rotazioni, perché si può sempre specificare l'⟨*angolo*⟩. Invece, per le trasformazioni affini, il nucleo di LaTeX e il pacchetto curve2e non dispongono di nessun comando. Il pacchetto xpicture (Fuster, 2012), invece, le gestisce; infatti è pensato proprio per gli scopi di un matematico, ma forse è un po' troppo complesso da usare quando il

8. Talvolta si usa la parola *skew*, ma questo ha un generico significato di inclinazione, mentre *shear* si riferisce ad un preciso scorrimento laterale "a strati".

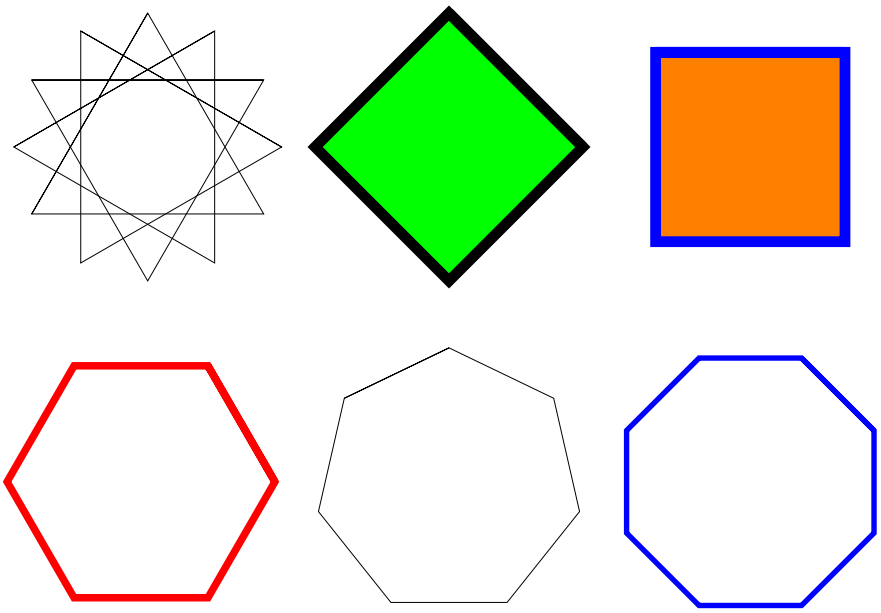

Figura 7: Alcuni poligoni

problema si può risolvere con una macro; vedremo più avanti come.

La figura 8 contiene alcuni esempi ottenuti con il codice seguente:

```
\unitlength=0.007\linewidth
\begin{picture}(100,100)
\AutoGrid
\Xellisse(30,20){30}{20}
  \Pbox(30,20)[l]{C_1}[2pt]
\Xellisse(80,50)[90]{30}{20}
  \Pbox(80,50)[l]{C_2}[2pt]
\Xellisse(30,70)[45]{30}{20}%
  [\linethickness{1pt}]
  \Pbox(30,70)[l]{C_3}[2pt]
\end{picture}
```

Si vedrà più avanti come sostituire la macro `\ellisse` con un'altra che consenta di eseguire anche la trasformazione affine di scorrimento.

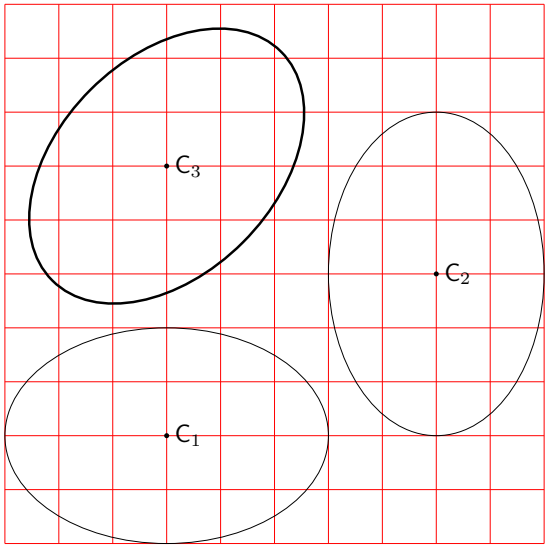

Figura 8: Alcune ellissi

3.3 Triangoli

È chiaro che un triangolo è il più semplice dei poligoni, ma questa figura ha diverse proprietà, in particolare dispone di diversi centri: il baricentro, l'ortocentro, l'incentro, il circocentro, il centro del cerchio dei nove punti, eccetera. Qui mostreremo come determinare questi centri ed eventualmente come disegnare i cerchi associati ad alcuni di questi centri.

Intanto, ecco un disegno con un triangolo e alcune sue linee importanti: mediana, bisettrice e altezza, figura 9. Le macro che servono per tracciare queste linee o i loro estremi sono esposte nei paragrafi successivi.

3.3.1 Il baricentro

Il *baricentro* è il punto di intersezione delle mediane, cioè i segmenti che uniscono ogni vertice con il punto mediano del lato opposto; vedi la figura 10.

Il codice per disegnare la figura è il seguente:

```
\unitlength=0.01\linewidth
\begin{picture}(100,100)
\AutoGrid
\CopyVect10,20to\Pu \CopyVect60,10to\Pd
\CopyVect80,90to\Pt
\polygon(\Pu)(\Pd)(\Pt)
\Pbox(\Pu)[tr]{P_1}[2pt] \Pbox(\Pd)[l]{P_2}[2pt]
\Pbox(\Pt)[tl]{P_3}[2pt]
\TriangleMedianBase\Pu on\Pd and\Pt to\Pmu
```

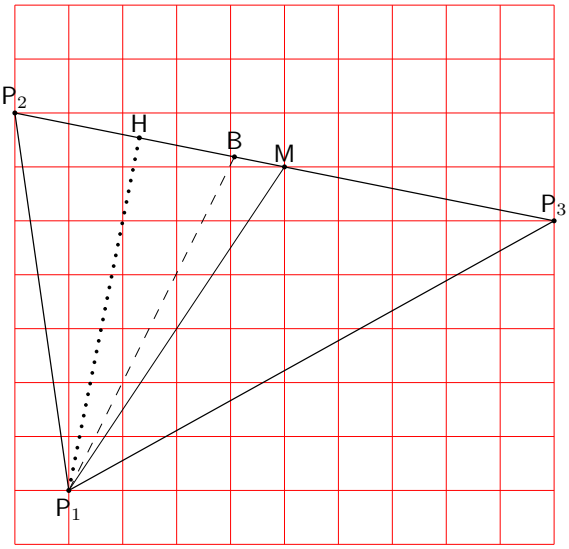

FIGURA 9: Un triangolo con una mediana, un'altezza e una bisettrice

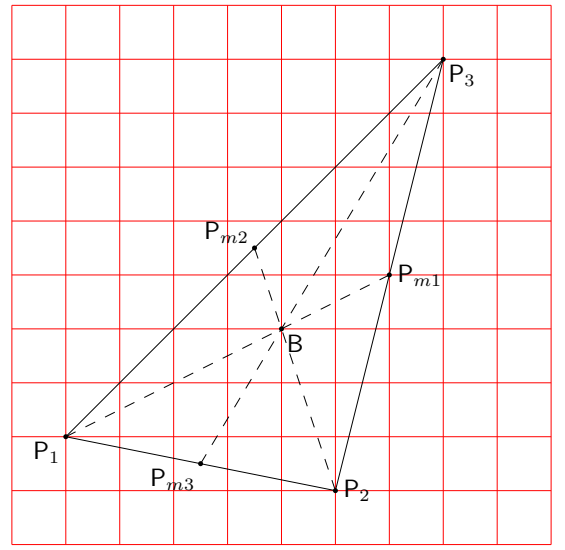

FIGURA 10: Determinazione del baricentro

```
\TriangleMedianBase\Pd on \Pt and\Pu to\Pmd
\Pbox(\Pmu)[l]{P_{m1}}[2pt]
\Pbox(\Pmd)[br]{P_{m2}}[2pt]
\Dashline(\Pu)(\Pmu){2}\Dashline(\Pd)(\Pmd){2}
\IntersectionOfSegments(\Pu)(\Pmu)%
  and(\Pd)(\Pmd)to\B
\Pbox(\B)[tl]{B}[2pt]
\end{picture}
```

Come si vede, si sono usati alcuni nuovi comandi: `\IntersectionOfSegments` e `\TriangleMedianBase`; quest'ultimo in realtà si basa a propria volta su `\IntersectionOfLines`. In realtà si sarebbe potuto determinare direttamente il baricentro con l'unica macro `\TriangleBarycenter`, ma non sarebbe stata disegnata l'intera costruzione che porta al risultato. Questi comandi sono definiti cosi:

```
\def\TriangleMedianBase#1on#2and#3to#4{%
\SubVect#1from#2to\TMBu
\SubVect#1from#3to\TMBd
\SubVect\TMBu from\TMBd to\Base
\ScaleVect\Base by0.5to\TMBm
\AddVect#2and\TMBm to#4\ignorespaces}

\def\IntersectionOfSegments(#1)(#2)and(#3)(#4)to#5{%
\SubVect#1from#2to\IoSvectu
\DirOfVect\IoSvectu to\DirIoSVecu
\SubVect#3from#4to\IoSvectd
\DirOfVect\IoSvectd to\DirIoSVecd
\IntersectionOfLines(#1)(\DirIoSVecu)%
  and(#3)(\DirIoSVecd)to#5 \ignorespaces}

\def\IntersectionOfLines(#1)(#2)and(#3)(#4)to#5{%
\bgroup
\def\IntPu{#1}\def\Uu{#2}
\def\IntPd{#3}\def\Ud{#4}%
 \DirOfVect\Uu to\Du
 \DirOfVect\Ud to\Dd
 \XpartOfVect\Du to \a
 \YpartOfVect\Du to \b
 \XpartOfVect\Dd to \c
 \YpartOfVect\Dd to \d
 \XpartOfVect\IntPu to \xu
 \YpartOfVect\IntPu to \yu
 \XpartOfVect\IntPd to \xd
 \YpartOfVect\IntPd to \yd
 \edef\Den{\fpeval{-(\a*\d-\b*\c)}}%
 \fptest{abs(\Den)<1e-5}%
 {% determinante quasi nullo
   \def#5{0,0}%
 }{% Determinante non quasi nullo
   \edef\Numx{\fpeval{(\c*(\b*\xu-\a*\yu)
     -\a*(\d*\xd-\c*\yd))/\Den}}%
   \edef\Numy{\fpeval{(\d*(\b*\xu-\a*\yu)
     -\b*(\d*\xd-\c*\yd))/\Den}}%
   \CopyVect\Numx,\Numy to\Paux
   \edef\x{\egroup\noexpand
     \edef\noexpand#5{\Paux}}\x\ignorespaces
  }}

\def\TriangleBarycenter(#1)(#2)(#3)to#4{%
\TriangleMedianBase#1on#2and#3to\Pa
\TriangleMedianBase#2on#3and#1to\Pb
\DistanceAndDirOfVect#1minus\Pa to\ModPa and\AngPa
\DistanceAndDirOfVect#2minus\Pb to\ModPb and\AngPb
\IntersectionOfLines(#1)(\AngPa)and(#2)(\AngPb)to#4}
```

Come si vede, il cuore di tutta l'elaborazione è costituito dalla macro `\IntersectionOfLines` basata sulla matematica seguente. Si hanno due linee ciascuna definita da un punto P_i e una direzione $u_i = \exp(\mathrm{i}\beta_i)$; queste, se non sono parallele o anti-parallele, si intersecano in un punto Q:

$$\begin{cases} Q = P_1 + t_1 u_1 \\ Q = P_2 + t_2 u_2 \end{cases}$$

Eliminando Q fra le due equazioni, e ponendo

$$P_2 - P_1 = P_D = D\,\mathrm{e}^{\mathrm{i}\delta}$$

si ottiene

$$t_1 u_1 - t_2 u_2 = P_D$$

Infine, separando le componenti orizzontali da quelle verticali, si ha:

$$\begin{cases} t_1 \cos\beta_1 - t2 \cos\beta_2 = D\cos\delta \\ t_1 \sin\beta_1 - t2 \sin\beta_2 = D\sin\delta \end{cases}$$

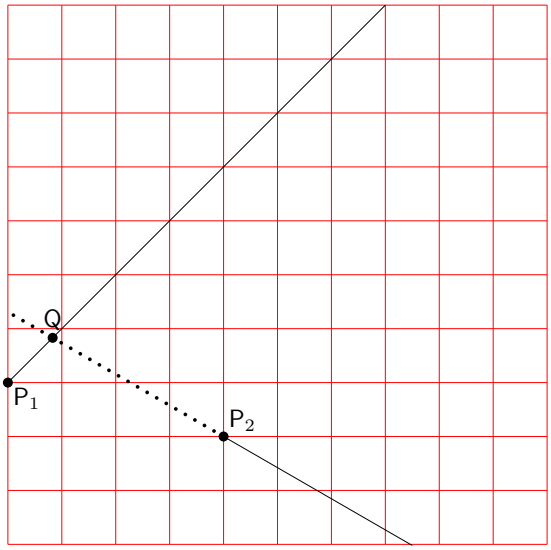

FIGURA 11: Intersezione di due rette

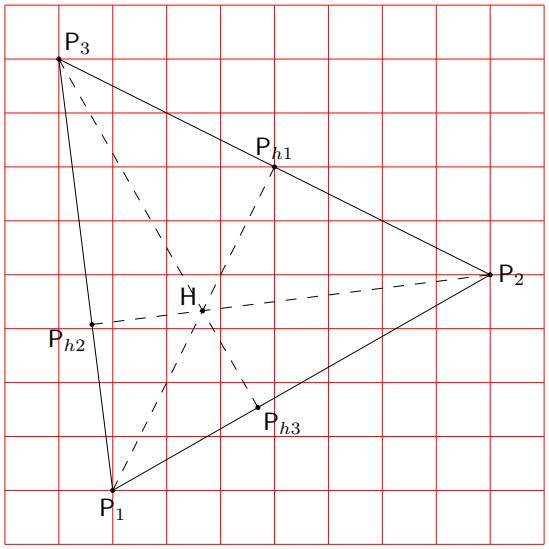

FIGURA 12: Determinazione dell'ortocentro

Risolvendo rispetto a uno dei due parametri t_i si ha, per esempio:

$$t_1 = D\frac{\sin(\delta - \beta_2)}{\sin(\beta_1 - \beta_2)}$$

da cui il punto di intersezione è dato da:

$$Q = P_1 + t_1 u_1 = P_1 + D\frac{\sin(\delta - \beta_2)}{\sin(\beta_1 - \beta_2)} u_1$$

La figura 11 mostra la semplice costruzione geometrica. Vi si vede, fra l'altro, che l'intersezione Q avviene sul lato "negativo" della seconda retta, il cui versore punta verso destra in basso: infatti l'angolo della sua direzione `\Udue` vale $-30°$.

3.3.2 L'ortocentro

L'*ortocentro* di un triangolo è il punto dove si intersecano le tre altezze rispetto ai tre lati; in un triangolo acutangolo esso cade all'interno del triangolo; in un triangolo rettangolo coincide con il vertice dell'angolo retto; in un triangolo scaleno l'ortocentro si trova all'esterno del triangolo. La figura 12 mostra il caso di un triangolo acutangolo.

```
\unitlength=0.01\linewidth
\begin{picture}(100,100)
\AutoGrid
\CopyVect20,10to\Pu\Pbox(\Pu)[t]{P_1}[2pt]
\CopyVect90,50 to\Pd\Pbox(\Pd)[l]{P_2}[2pt]
\CopyVect10,90 to\Pt\Pbox(\Pt)[bl]{P_3}[2pt]
\polygon(\Pu)(\Pd)(\Pt)
\TriangleHeightBase\Pu on\Pd and\Pt to\Phu
\TriangleHeightBase\Pd on\Pt and\Pu to\Phd
\TriangleHeightBase\Pt on\Pu and\Pd to\Pht
\Dashline(\Pu)(\Phu){2}
\Pbox(\Phu)[b]{P_{h1}}[2pt]
\Dashline(\Pd)(\Phd){2}
\Pbox(\Phd)[tr]{P_{h2}}[2pt]
\Dashline(\Pt)(\Pht){2}
\Pbox(\Pht)[tl]{P_{h3}}[2pt]
\IntersectionOfSegments(\Pu)(\Phu)
  and(\Pd)(\Phd)to\H
\Pbox(\H)[br]{H}[2pt]
\end{picture}
```

Si noti che, come nel caso del baricentro, non si è usato il comando `\IntersectionOfLines` ma `\IntersectionOfSegments`. Non sono proprio equivalenti, perché nel primo caso bisogna specificare un punto e una direzione, mentre nel secondo caso bisogna specificare due punti (distinti). La differenza sostanziale è che, appunto, il comando per i segmenti calcola lui stesso la direzione in base agli estremi del segmento e poi invoca il comando per le linee. La direzione non è sempre intuitiva, quindi è sempre meglio lasciare fare i calcoli alle macro correttamente definite. In questo caso, invece, si è invocato il comando `\TriangleHeightBase`, che permette di tracciare anche le linee tratteggiate per eseguire la costruzione geometrica. Esiste anche il comando `\TriangleOrthoCenter`, che però determina direttamente le coordinate dell'ortocentro `\H`, ma non i punti che permettono di disegnare le linee tratteggiate. In ogni caso, quei comandi sono definiti come segue:

```
\def\TriangleHeightBase#1on#2and#3to#4{%
% #1 := Vertice del triangolo
% #2 e #3 :=   estremi del lato opposto
% #4 := piede dell'altezza
\SubVect#2from#3to\Base
\ArgOfVect\Base to\Ang
\CopyVect\fpeval{\Ang+90}:1 to\Perp
\IntersectionOfLines(#1)(\Perp)
  and(#2)(\Base)to#4\ignorespaces}

\def\TriangleOrthocenter(#1)(#2)(#3)to#4{%
\TriangleHeightBase#1on#2and#3to\Pa
\TriangleHeightBase#2on#3and#1to\Pb
\DistanceAndDirOfVect#1minus\Pa
  to\ModPa and\AngPa
```

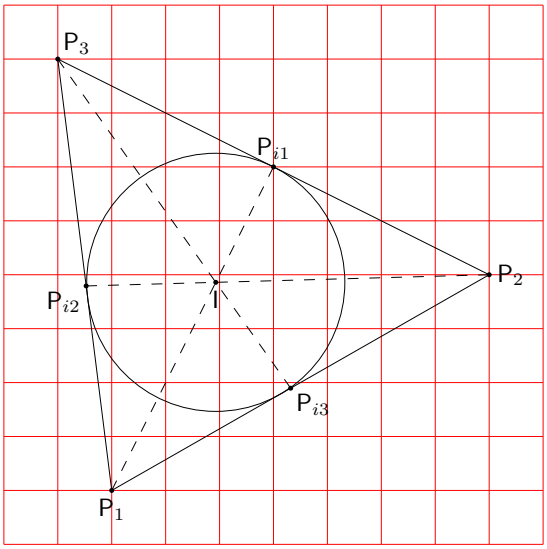

FIGURA 13: Determinazione dell'incentro e del cerchio inscritto

```
\DistanceAndDirOfVect#2minus\Pb
  to\ModPb and\AngPb
\IntersectionOfLines(#1)(\AngPa)
  and(#2)(\AngPb)to#4}
```

3.3.3 L'incentro

L'*incentro* è l'intersezione delle bisettrici degli angoli interni di un triangolo; ogni punto di ciascuna bisettrice è equidistante dai due lati del triangolo che racchiudono l'angolo considerato per definire la bisettrice. Il punto di incrocio delle tre bisettrici è perciò equidistante da ciascuno dei tre lati, quindi è il centro dell'incerchio[9], cioè del cerchio inscritto nel triangolo e tangente a tutti e tre i lati. È possibile disegnare l'intera costruzione dell'incentro e si può tracciare direttamente anche l'incerchio, come si vede nella figura 13.

Per disegnare la figura 13 si è usato il codice seguente dove compaiono alcune nuove macro.

```
\unitlength=0.01\linewidth
\begin{picture}(100,100)
\AutoGrid
\CopyVect20,10to\Pu\Pbox(\Pu)[t]{P_1}[2pt]
\CopyVect90,50 to\Pd\Pbox(\Pd)[l]{P_2}[2pt]
\CopyVect10,90 to\Pt\Pbox(\Pt)[bl]{P_3}[2pt]
\polygon(\Pu)(\Pd)(\Pt)
\TriangleBisectorBase\Pu on\Pd and\Pt to\Piu
\TriangleBisectorBase\Pd on\Pt and\Pu to\Pid
\TriangleBisectorBase\Pt on\Pu and\Pd to\Pit
\Dashline(\Pu)(\Piu){2}
  \Pbox(\Piu)[b]{P_{i1}}[2pt]
\Dashline(\Pd)(\Pid){2}
  \Pbox(\Pid)[tr]{P_{i2}}[2pt]
\Dashline(\Pt)(\Pit){2}
  \Pbox(\Pit)[tl]{P_{i3}}[2pt]
\IntersectionOfSegments(\Pu)(\Piu)
  and(\Pd)(\Pid)to\I
\Pbox(\I)[t]{I}[2pt]
\SegmentLength(\I)(\Piu)to\R
\Circlewithcenter\I radius\R
\end{picture}
```

Le nuove macro sono le seguenti. `\TriangleBisectorBase` che consente di disporre dei dati intermedi e di disegnare le linee tratteggiate necessarie per descrivere la costruzione. `\SegmentLength` consente di misurare la distanza di due punti che potrebbero essere gli estremi di un segmento. `\Circlewithcenter` serve per disegnare un cerchio noti il centro e il raggio (non il diametro, contrariamente alla definizione originale dell'ambiente picture) e non richiede di essere messo in posizione con `\put`. Infine `\TriangleIncenter` permette di trovare l'incentro ma non consente di usare i dati interni del codice.

```
\def\TriangleBisectorBase#1on#2and#3to#4{%
% #1 := Vertice del triangolo
% #2 e #3 :=  estremi del lato opposto
% #4 := piede della bisettrice
\SubVect#2from#1to\Luno
\SubVect#3from#1to\Ldue
\SubVect#2from#3to\Base
\ArgOfVect\Luno to\Arguno
\ArgOfVect\Ldue to\Argdue
\edef\ArgBis{\fpeval{(\Arguno+\Argdue)/2}}%
\CopyVect \ArgBis:1to \Bisect
\IntersectionOfLines(#2)(\Base)
  and(#1)(\Bisect)to#4\ignorespaces}

\def\SegmentLength(#1)(#2)to#3{%
  \SubVect#1from#2to\Segm
  \ModOfVect\Segm to#3}

\def\Circlewithcenter#1radius#2{%
\put(#1){\circle{\fpeval{2*#2}}}
\ignorespaces}

\def\TriangleIncenter(#1)(#2)(#3)to#4{%
\TriangleBisectorBase#1on#2and#3to\Pa
\TriangleBisectorBase#2on#3and#1to\Pb
\DistanceAndDirOfVect#1minus\Pa
  to\ModPa and\AngPa
\DistanceAndDirOfVect#2minus\Pb
  to\ModPb and\AngPb
\IntersectionOfLines(#1)(\AngPa)
  and(#2)(\AngPb)to#4}
```

3.3.4 Il cerchio dei tre punti

Un cerchio e una circonferenza sono definiti dal raggio e dalla circonferenza. Ma questi dati possono essere ricavati dalla lista di tre punti per i quali passa la circonferenza. Geometricamente è semplicissimo individuare centro e raggio: basta disegnare il triangolo che ha i tre punti come vertici,

9. La geometria distingue la *circonferenza* dal *cerchio*, essendo la prima solo il contorno del secondo. L'*incerchio* potrebbe essere chiamato *incirconferenza*, ma non si è trovato questo termine in nessun testo di geometria. In questo contesto, sembra superfluo distinguere le due cose, ma formalmente usare indifferentemente i due nomi non è corretto.

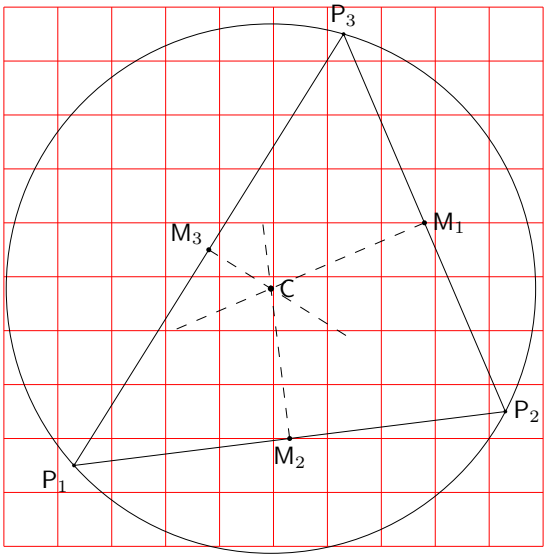

FIGURA 14: Circonferenza per tre punti dati

per poi incrociare gli assi di due lati. Il punto di intersezione è equidistante da tutti e tre i vertici, proprio perché si trova sugli assi di due distinti lati del triangolo. La figura 14 mostra la costruzione.

Il disegno della figura 14 è stato eseguito con il codice seguente:

```
\centering
\unitlength=0.01\linewidth
\begin{picture}(100,100)
\AutoGrid
\CopyVect13,15 to\Pu\Pbox(\Pu)[tr]{P_1}[1.5]
\CopyVect93,25 to\Pd\Pbox(\Pd)[l]{P_2}[1.5]
\CopyVect63,95 to\Pt\Pbox(\Pt)[b]{P_3}[1.5]
\polygon(\Pu)(\Pd)(\Pt)
\AxisOf\Pu and\Pd to\Md\Dd
\AxisOf\Pu and\Pt to\Mt\Dt
\AxisOf\Pd and\Pt to\Mu\Du
\IntersectionOfLines(\Md)(\Dd)and(\Mt)(\Dt)
  to\Centro
\SubVect\Md from\Centro to\Rd
\ArgOfVect\Rd to\Angd
\SubVect\Mt from\Centro to\Rt
\ArgOfVect\Rt to\Angt
\SubVect\Mu from\Centro to\Ru
\ArgOfVect\Ru to\Angu
\Dashline(\Md)(\Angd:40){2}
\Pbox(\Md)[t]{M_2}[2]
\Dashline(\Mt)(\Angt:30){2}
\Pbox(\Mt)[br]{M_3}[2]
\Dashline(\Mu)(\Angu:50){2}
\Pbox(\Mu)[l]{M_1}[2]
\ThreePointCircle*(\Pu)(\Pd)(\Pt)
\Pbox(\C)[l]{C}[2.5]
\end{picture}
```

Vi compaiono alcune nuove macro che sono riportate qui di seguito:

```
\def\AxisOf#1and#2to#3#4{%
\SubVect#1from#2to\Base
\ScaleVect\Base by0.5to\Base
\AddVect\Base and#1to#3
\MultVect\Base by0,1to#4}

\NewDocumentCommand\ThreePointCircle%
   {s d() d() d()}{%
\AxisOf#2and#3to\Mu\Du \AxisOf#2and#4to\Md\Dd
\IntersectionOfLines(\Mu)(\Du)and(\Md)(\Dd)
  to\C
\SubVect#2from\C to\R \ScaleVect\R by2to\D
\ModOfVect\D to\D
\IfBooleanTF{#1}%
  {\CircleWithCenter\C Radius\R}{}%
\ignorespaces}
```

Si noti che `\ArgOfVect` è già presente in curve2e. Nella definizione di `\ThreePointCircle` è possibile usare un asterisco facoltativo; la sua presenza permette di disegnare effettivamente il cerchio per tre punti, ma se lo si omette, rende solo accessibili le coordinate del centro dalla sua macro interna `\C`.

Senza ricorrere ad asterischi si può usare la macro `\ThreePointCircleCenter` che ha la seguente sintassi:

`\ThreePointCircleCenter%`
`(`⟨*P1*⟩`)(`⟨*P2*⟩`)(`⟨*P3*⟩`) to`⟨*centro*⟩

dove ⟨*centro*⟩ è la macro che riceve le coordinate del centro del cerchio dei tre punti.

3.3.5 La circonferenza dei nove punti

Dato un triangolo, i nove punti per cui passa circonferenza sono i seguenti:

- i tre punti medi dei lati
- i piedi delle tre altezze
- i punti medi dei segmenti che uniscono ogni vertice con l'ortocentro

Questa è una proprietà geometrica dei triangoli e di questo particolare cerchio; per tracciare la sua circonferenza basta usare quanto descritto nel paragrafo precedente scegliendo tre punti a piacere; nel disegno della figura 15 si sono scelti i punti medi dei lati.

Il codice usato per la figura 15 è il seguente, dove l'unica macro nuova è `\MiddlePointOf`.

```
\unitlength=0.01\linewidth
\begin{picture}(100,100)
\AutoGrid
\CopyVect20,0to\Pu \Pbox(\Pu)[t]{P_1}[2pt]
\CopyVect10,80to\Pd \Pbox(\Pd)[br]{P_2}[2pt]
\CopyVect100,60to\Pt \Pbox(\Pt)[l]{P_3}[2pt]
{\polygon(\Pu)(\Pd)(\Pt)\ignorespaces}
\TriangleMedianBase\Pu on\Pd and\Pt to\Pmu
\Pbox(\Pmu)[bl]{M_1}[2pt]
\TriangleMedianBase\Pd on\Pt and\Pu to\Pmd
\Pbox(\Pmd)[tl]{M_2}[2pt]
\TriangleMedianBase\Pt on\Pu and\Pd to\Pmt
\Pbox(\Pmt)[tr]{M_3}[2pt]
\ThreePointCircle(\Pmu)(\Pmd)(\Pmt)
\TriangleHeightBase\Pu on\Pd and\Pt to\Phu
\Pbox(\Phu)[b]{H_1}[2pt]
\TriangleHeightBase\Pd on\Pt and\Pu to\Phd
\Pbox(\Phd)[tl]{H_2}[2pt]
```

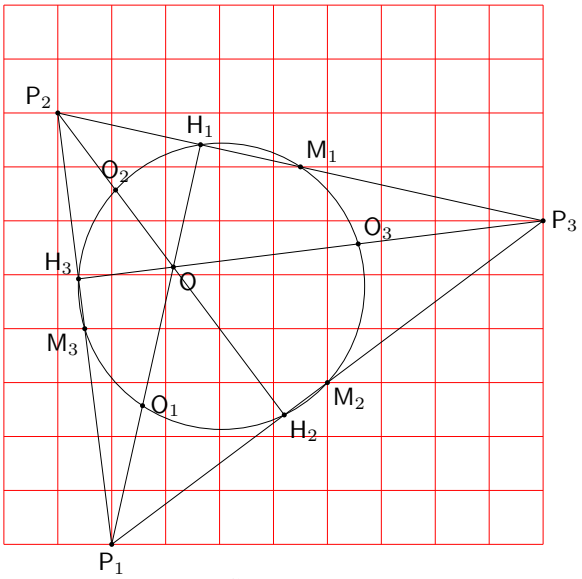

FIGURA 15: Cerchio dei nove punti

```
\TriangleHeightBase\Pt on\Pu and\Pd to\Pht
\Pbox(\Pht)[br]{H_3}[2pt]
\segment(\Pu)(\Phu)
\segment(\Pd)(\Phd)
\segment(\Pt)(\Pht)
\SubVect\Pu from\Phu to\DirHu
\SubVect\Pd from\Phd to\DirHd
\SubVect\Pt from\Pht to\DirHt
\IntersectionOfLines(\Pu)(\DirHu)
  and(\Pt)(\DirHt)to\Po
\Pbox(\Po)[tl]{O}[2pt]
\MiddlePointOf(\Po)(\Pu)to\Pou
\Pbox(\Pou)[l]{O_1}[2pt]
\MiddlePointOf(\Po)(\Pd)to\Pod
\Pbox(\Pod)[b]{O_2}[2pt]
\MiddlePointOf(\Po)(\Pt)to\Pot
\Pbox(\Pot)[bl]{O_3}[2pt]
\ThreePointCircle*(\Pou)(\Pod)(\Pot)
\end{picture}
```

E la macro nuova è definita così:

```
\def\MiddlePointOf(#1)(#2)to#3{%
\SubVect#1from#2to\Base
\ScaleVect\Base by0.5to\Base
\AddVect\Base and#1to#3\ignorespaces}
```

3.3.6 L'inellisse di Steiner

In un triangolo possono essere iscritte infinite ellissi tangenti internamente a tutti e tre i suoi lati. L'inellisse di Steiner, però, è unica; essa è definita come l'ellisse tangente internamente ai lati di un triangolo nei punti medi dei lati.

Esistono diverse dimostrazioni e costruzioni geometriche per costruirla, ma qui ho seguito una via percorribile con alcune macro che si possono aggiungere a quelle descritte in questo articolo.

L'enumerazione che segue mostra i punti su cui si basa il ragionamento.

1. Sia dato un triangolo T comunque orientato nel piano; si lavori su un triangolo equivalente T_0 ottenuto da T mediante rotazione, in modo che un lato sia orizzontale e il disegno del resto del triangolo si svolga nel semipiano superiore; è banale eseguire una traslazione e una rotazione di T per portarlo nella posizione specificata per T_0. Bisogna memorizzare l'ammontare della traslazione e della rotazione per riportare il tutto nella posizione iniziale alla fine della costruzione. In realtà, la traslazione e la rotazione non sarebbero necessarie per la costruzione; qui si usano per rendere più chiara la procedura per determinare questa ellisse.
2. Si costruisce il triangolo equilatero T_1 avente la stessa base di T_0; è evidente che l'inellisse di Steiner di questo triangolo equilatero è un cerchio e che questo è anche l'incerchio di centro C.
3. Si deformi T_1 fino a farlo diventare il triangolo isoscele T_2 di altezza pari all'altezza di T_0; si tratta di una affinità di deformazione verticale eseguita con un fattore di scala delle sole ordinate pari al rapporto fra l'altezza di T_0 e quella di T_1. Ne consegue che l'incerchio di T_1 si deforma nell'inellisse di cui l'asse verticale viene scalato con lo stesso fattore di scala delle ordinate, mentre l'asse orizzontale rimane pari al diametro dell'incerchio di $T1$. Per effetto di questa trasformazione il centro C si sposta nel punto C_e. Gli elementi per disegnare questa ellisse sono quindi tutti noti e la sua costruzione è immediata.
4. Bisogna ora deformare il triangolo isoscele T_2 con un movimento di *shear* o di scorrimento orizzontale proporzionale alle ordinate, in modo che si trasformi nel triangolo T_0; questa è una affinità di taglio che permette di muovere ogni punto del triangolo T_2 nel triangolo T_0; nel far questo, l'inellisse di T_2, di centro C_e, si trasforma nell'inellisse di T_0, di centro C_i, solo che gli assi della prima ellisse, che erano perpendicolari, si trasformano in due diametri obliqui dell'ellisse trasformata. Questo è quanto succede con l'affinità di taglio.
5. Si noti che l'affinità di taglio permette di calcolare direttamente le coordinate del centro della nuova ellisse, la cui ordinata rimane costante, ma l'ascissa si sposta di $b \tan \alpha$. Quindi si possono esprimere le equazioni delle ellissi rispetto a un sistema di coordinate con l'origine nel loro centro. La macro `\Xellisse` accetta sia le coordinate del centro, sia l'angolo di taglio, oltre ai semiassi, quindi non è un problema disegnare l'ellisse trasformata con i mezzi a disposizione nell'ambiente picture. Tuttavia, le equazioni da risolvere per determinare l'angolo di rotazione e i semiassi dell'inellisse sono piuttosto complesse; invece non è un problema quello di creare una macro come `\XSellisse`,

che vedremo fra poco, che attiva a propria volta una nuova macro di base `\Sellisse` per disegnare ellissi tenendo conto anche dello scorrimento. `\Sellisse` e `\XSellisse` rimangono retrocompatibili rispettivamente con `\ellisse` e `\Xellisse` e la `S` nel loro nome ricorda che gestiscono anche lo scorrimento.

6. L'unica informazione per gestire lo scorrimento consiste appunto nel determinare l'angolo α di cui si muovono le rette verticali per prendere la stessa inclinazione della mediana di T_0 rispetto alla sua base. La tangente di questo angolo rappresenta il parametro che gestisce l'affinità di scorrimento governata dalle due formule seguenti:
$$\begin{cases} x' = x + y\tan\alpha \\ y' = y \end{cases} \tag{1}$$
7. Conviene definire una macro, `\EllisseSteiner`, che riceva solo i vertici del triangolo e poi faccia da sola tutti i calcoli e le necessarie costruzioni. Conviene che disponga di una variante (selezionabile con un asterisco facoltativo) che permetta di costruire tutti passaggi intermedi, oppure che esponga solo il risultato finale.
8. L'ultimo passo è quello di ripristinare la posizione della costruzione, riportando il triangolo T_0 con la sua inellisse nella posizione originale.

La macro per eseguire i calcoli e disegnare il risultato finale, eventualmente con i passi intermedi della costruzione, si chiama `\EllisseSteiner` che verrà descritta fra poco; ma grazie a essa si può disegnare la costruzione dell'ellisse di Steiner come nella figura 16 con il codice

```
\unitlength=0.009\linewidth
\begin{picture}(100,100)
\AutoGrid
\EllisseSteiner*(10,10)(90,20)(70,80)[3]
\end{picture}
```

Senza ingombrare il disegno con la costruzione, si può disegnare il risultato finale nella figura 17 con il codice

```
\unitlength=0.009\linewidth
\begin{picture}(100,100)
\AutoGrid
\EllisseSteiner(10,10)(90,20)(70,80)[2]
\end{picture}
```

Come si vede, la differenza fra i due disegni è ottenuta solamente con un asterisco facoltativo per la macro `\EllisseSteiner`.

Il codice di `\EllisseSteiner` può sembrare complicato; ma altro non è se non quanto descritto sopra; si può seguire passo passo semplicemente rileggendo i punti dell'enumerazione precedente. Certo, è stato necessario aggiungere diverse istruzioni per fare o non fare certe cose a seconda della presenza dell'asterisco; tuttavia non confondono le idee circa il processo vero e proprio per determinare l'inellisse.

La macro ha la sintassi seguente:

`\EllisseSteiner`⟨⋆⟩`(`⟨*primo vertice*⟩`)`
`(`⟨*secondo vertice*⟩`)`
`(`⟨*terzo vertice*⟩`)[`⟨*diametro*⟩`]`

dove l'asterisco facoltativo serve per disegnare o non disegnare i passi della costruzione; seguono i tre vertici del triangolo presi nell'ordine tale che la traslazione e la rotazione iniziali portino la costruzione sull'asse delle ascisse e il resto del triangolo si svolga nel semipiano superiore. Infine, il ⟨*diametro*⟩ indica il diametro dei pallini neri con cui si marcano i punti salienti; se si omettono le dimensioni, si assume che siano indicati in punti tipografici. Questa impostazione facoltativa è interessante perché il pallino deve essere visibile indipendentemente dalla scala del disegno.

```
\NewDocumentCommand\EllisseSteiner%
  {s d() d() d() O{1}}{\bgroup
%
\IfBooleanTF{#1}{}{\put(#2)}{%
  \CopyVect0,0to\Pu
  \SubVect#2from#3to\Pd
  \SubVect#2from#4to\Pt
  \IfBooleanTF{#1}{}{%
    \Pbox(\Pu)[r]{P_1}[#5]\Pbox(\Pd)[t]{P_2}[#5]
    \Pbox(\Pt)[b]{P_3}[#5]}
  \ModAndAngleOfVect\Pd to\M and\Rot
  \MultVect\Pd by-\Rot:1 to\Pd
  \MultVect\Pt by-\Rot:1 to\Pt
  \IfBooleanTF{#1}{}{\rotatebox{\Rot}}%
  {\makebox(0,0)[bl]{%
    \IfBooleanTF{#1}{%
    \Pbox(\Pu)[r]{P_1}[#5]\Pbox(\Pd)[t]{P_2}[#5]
    \Pbox(\Pt)[b]{P_3}[#5]}{}%
  \polygon(\Pu)(\Pd)(\Pt)%
  \edef\B{\fpeval{\M/2}}%
  \edef\H{\fpeval{\B*tand(60)}}
  \IfBooleanTF{#1}{\Pbox(\B,\H)[b]{H}[#5]
  \polygon(\Pu)(\B,\H)(\Pd)}{}%
  \edef\R{\fpeval{\B*tand(30)}}
  \IfBooleanTF{#1}{\Pbox(\B,\R)[bl]{C}[#5]
    \Circlewithcenter\B,\R radius{\R}}{}%
  \GetCoord(\Pt)\Xt\Yt
  \edef\VScale{\fpeval{\Yt/\H}}
  \IfBooleanTF{#1}{%
    \polyline(\Pu)(\B,\Yt)(\Pd)
    \Pbox(\B,\Yt)[b]{V}[#5]}{}%
  \edef\Ce{\fpeval{\R*\VScale}}
  \IfBooleanTF{#1}{%
    \Xellisse(\B,\Ce){\R}{\Ce}
    \Pbox(\B,\Ce)[r]{C_e}[#5]
    \Pbox(\B,0)[t]{B}[#5]}{}%
  \SubVect\B,0 from\Pt to\SlMedian
  \IfBooleanTF{#1}%
    {\Dotline(\B,0)(\Pt){2}[1.5]}{}%
  \ModAndAngleOfVect\SlMedian to\Med and\Alfa
  \edef\Alfa{\fpeval{90-\Alfa}}
    \IfBooleanTF{#1}{%
      \Dotline(\B,\Yt)(\B,0){2}[1.5]
      \Pbox(\fpeval{\B+\Ce*tand{\Alfa}},%
      \Ce)[l]{C_i}[#5]
```

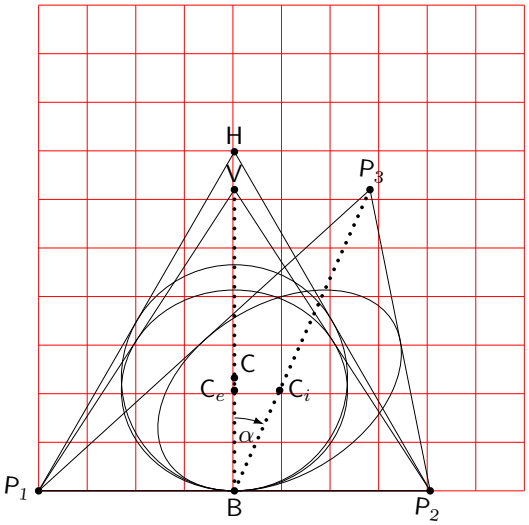

FIGURA 16: Sequenza delle operazioni per la determinazione dell'inellisse

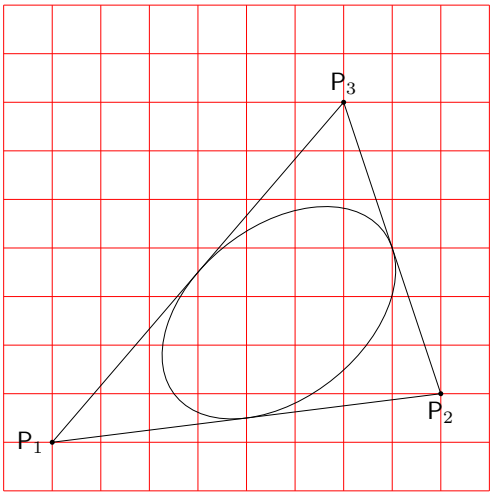

FIGURA 17: Inellisse finale

```
      \VectorArc(\B,0)(\B,15){-\Alfa}
      \Pbox(\fpeval{\B+2.5},14)[t]{\alpha}[0]}{}%
  \edef\a{\R}\edef\b{\Ce}%
\CopyVect\fpeval{\B+\Ce*tand{\Alfa}},\Ce to\CI
\XSellisse(\CI)<\Alfa>{\R}{\Ce}
}}}\egroup\ignorespaces}
```

Tuttavia bisogna ancora disporre dei codici di `\Sellisse` e di `\XSellisse`.

Il codice di `\Sellisse` sembra terribilmente complicato rispetto al semplice codice di `\ellisse`; in realtà i comandi usati hanno nomi tali che si capisce subito a che cosa servono. Infatti, `\ScaleVector` è molto simile al comando `\ScaleVect` facente parte di curve2e; in effetti, il comando originale scala un numero complesso, quindi un vettore che va dall'origine a un punto dato; questo nuovo comando, invece, scala un vettore fra due punti dati. Il comando `\ShearVect`, invece, esegue l'affinità di scorrimento per un vettore fra due punti dati. I punti su cui operare sono gli stessi sui quali opera la macro `\ellisse` e sono dodici in totale, tre per ogni lato del rettangolo circoscritto all'ellisse prima di applicare l'affinità. Quindi, fra scalamenti e scorrimenti ci sono molti comandi da eseguire.

```
\def\ShearVect(#1)(#2)by#3to#4{%
\SubVect#1from#2to\AUX
\GetCoord(\AUX)\Aux\Auy
\edef\Aux{\fpeval{\Aux + #3*\Auy}}%
\edef\Auy{\fpeval{\Auy}}%
\AddVect\Aux,\Auy and#1to#4\ignorespaces}

\def\ScaleVector(#1)(#2)by#3to#4{%
\SubVect#1from#2to\AUX
\ScaleVect\AUX by#3to\AUX
\AddVect\AUX and#1to#4\ignorespaces}

\NewDocumentCommand\Sellisse{s m m O{0}}{\bgroup
\CopyVect#2,#3to\Ptr \ScaleVect\Ptr by-1to\Pbl
\CopyVect#2,-#3to\Pbr \ScaleVect\Pbr by-1to\Ptl
\edef\Ys{\fpeval{tand{#4}}}%
\edef\K{\fpeval{4*(sqrt(2)-1)/3}}%
%
\ShearVect(0,0)(0,#3)by\Ys to\Pmt
\ShearVect(0,0)(0,-#3)by\Ys to\Pmb
\ShearVect(0,0)(#2,0)by\Ys to\Pmr
\ShearVect(0,0)(-#2,0)by\Ys to\Pml
%
\ShearVect(\Pmr)(\Ptr)by\Ys to\Ptr
\ShearVect(\Pml)(\Ptl)by\Ys to\Ptl
\ShearVect(\Pmr)(\Pbr)by\Ys to\Pbr
\ShearVect(\Pml)(\Pbl)by\Ys to\Pbl
%
\IfBooleanTF{#1}{%
  \Pbox(\Ptr)[bl]{P_{tr}}
  \Pbox(\Pbl)[tr]{P_{bl}}
  \Pbox(\Pbr)[tl]{P_{br}}
  \Pbox(\Ptl)[br]{P_{tl}}
  \polygon(\Pbr)(\Ptr)(\Ptl)(\Pbl)}{}%
%
\ScaleVector(\Pmr)(\Ptr)by\K to\Crt
\ScaleVector(\Pmr)(\Pbr)by\K to\Crb
\ScaleVector(\Pml)(\Ptl)by\K to\Clt
\ScaleVector(\Pml)(\Pbl)by\K to\Clb
\ScaleVector(\Pmt)(\Ptr)by\K to\Ctr
\ScaleVector(\Pmt)(\Ptl)by\K to\Ctl
\ScaleVector(\Pmb)(\Pbr)by\K to\Cbr
\ScaleVector(\Pmb)(\Pbl)by\K to\Cbl
%
\IfBooleanTF{#1}{%
 \Pbox(\Crt)[l]{C_{rt}}\Pbox(\Crb)[l]{C_{rb}}
 \Pbox(\Clt)[r]{C_{lt}}\Pbox(\Clb)[r]{C_{lb}}
 \Pbox(\Ctr)[b]{C_{tr}}\Pbox(\Ctl)[b]{C_{tl}}
 \Pbox(\Cbr)[t]{C_{br}}\Pbox(\Cbl)[t]{C_{bl}}
%
\Pbox(\Pmr)[l]{P_{mr}}\Pbox(\Pmt)[b]{P_{mt}}%
\Pbox(\Pml)[r]{P_{ml}}\Zbox(\Pmb)[t]{P_{mb}}%
%
\polygon(\Pbr)(\Ptr)(\Ptl)(\Pbl)\thicklines}{}%
%
\moveto(\Pmr)
\curveto(\Crt)(\Ctr)(\Pmt)
\curveto(\Ctl)(\Clt)(\Pml)
\curveto(\Clb)(\Cbl)(\Pmb)
\curveto(\Cbr)(\Crb)(\Pmr)
\fillstroke
\egroup}
```

La sintassi di `\Sellisse` è la seguente:

```
\Sellisse⟨⋆⟩{⟨semiasse-a⟩}{⟨semiasse-b⟩}%
    <⟨shear⟩>
```

L'asterisco facoltativo permette di disegnare la costruzione, mentre senza di esso si ha solo il risultato finale; come la corrispondente macro `\ellisse`, questa nuova macro non dovrebbe essere usata direttamente, ma solo attraverso `\XSellisse`, che le trasferisce i dati essenziali. In alternativa, se l'utente prima si usarla specifica l'equivalenza di `\fillstroke` con `\strokepath`, l'ellisse che si ottiene ha il centro nell'origine degli assi. Nelle figure 18 e 19 si vedono appunto i due risultati.

Il codice di `\XSellisse` è leggermente più complesso di quello di `\Xellisse`, ma solo perché deve passare più argomenti alla macro subalterna `\Sellisse`.

```
\NewDocumentCommand\XSellisse%
  { s D(){0,0} O{0} D<>{0} m m s O{} o}%
  {\IfBooleanTF{#1}%
     {\let\fillstroke\fillpath}%
     {\let\fillstroke\strokepath}%
  \put(#2){\rotatebox{#3}{#8\relax
  \IfBooleanTF{#7}%
     {\Sellisse*{#5}{#6}[#4]}%
     {\Sellisse{#5}{#6}[#4]}%
  \IfValueTF{#9}%
    {\let\fillstroke\strokepath
    #9\Sellisse{#5}{#7}[#4]}{}}}%
  \ignorespaces}
```

La sua sintassi è la seguente:

```
\XSellisse⟨⋆⟩(⟨centro⟩)[⟨angolo⟩]%
    <⟨shear⟩>{⟨semiasse-a⟩}{⟨semiasse-b⟩}%
    ⟨⋆⟩[⟨impostazioni-1⟩]%
    [⟨impostazioni-2⟩]
```

Gli argomenti hanno gli stessi significati che hanno per la macro `\Xellisse`, tranne ⟨*shear*⟩ e il secondo asterisco facoltativo. Si noti che il parametro ⟨*shear*⟩, facoltativo e preimpostato a zero, indica l'angolo in gradi sessagesimali di cui ruotano le righe verticali soggette all'affinità di scorrimento; nella figura 18 si vede che è stato specificato il valore di 20°, e l'inclinazione dei lati obliqui del parallelogramma rispetto alla verticale è appunto di 20° positivi in senso orario. Il secondo asterisco, come si vede osservando la figura 18, che è stata composta con questo secondo asterisco, contiene tutta la costruzione alle spalle dell'affinità. Invece la figura 19, costruita senza il secondo asterisco, mostra solo il risultato finale.

3.3.7 Ellisse tangente internamente a un triangolo e di cui sia specificato un fuoco

Questo problema di geometria piana è ripreso dall'articolo di Estevão Candia Candia (2019), o meglio dalla sua tesi (Candia, 2018). Egli cita lo scritto di Sergio Alves (Alves, 2018) fornendo il riferimento che rimanda alla Rivista dei Professori di Matematica Brasiliani. Non sono riuscito ad accedervi, perché bisogna essere soci dell'associazione che la pubblica, ma ho trovato in rete un altro documento dello stesso autore e con lo stesso titolo *Elipses inscritas num triângulo* (vedi la nota nel riferimento Alves (2018)). In questo testo è presentato per gli allievi delle scuole secondarie superiori un certo numero di esercizi da svolgere con riga, squadra e compasso, fra i quali non c'è quello che riguarda questo paragrafo, ma ve ne sono le premesse. Nell'articolo di Candia (2019) c'è la costruzione completa della soluzione ottenuta con METAPOST. Qui mi interessa mostrare come la soluzione si possa trovare anche con i mezzi dell'ambiente picture esteso.

Il problema è il seguente: costruire l'ellisse tangente internamente ai lati di un triangolo dato, di cui sono noti i tre vertici e, per l'ellisse, un fuoco interno al triangolo. Ora è evidente che, se si riescono a determinare l'altro fuoco e i punti di tangenza, il problema è quasi risolto. Infatti, uno qualunque dei punti di tangenza permette di determinare la lunghezza dell'asse maggiore grazie alla definizione stessa dell'ellisse: essa è il luogo dei punti la cui somma delle distanze dai due fuochi è pari all'asse maggiore; se $2c$ è la distanza fra i due fuochi e $2a$ è la lunghezza dell'asse maggiore, allora è noto che l'asse minore $2b$ è legato agli altri due parametri dalla relazione:

$$a^2 = b^2 + c^2 \tag{2}$$

A questo proposito, quell'equazione si ricava a vista dal disegno della figura 20, dove il triangolo costruito intersecando la semicirconferenza superiore di raggio a con il semiasse maggiore interseca due punti che permettono di determinare il triangolo rettangolo POF_2 i cui cateti valgono b e c e l'ipotenusa vale a; Pitagora ci fornisce dunque la relazione (2) fra i semiassi e la semidistanza focale.

Determinare il centro dell'ellisse e l'inclinazione dell'asse maggiore conoscendo la posizione dei due fuochi è banale, per cui la macro `\Xellisse` può svolgere il proprio compito e disegnare l'ellisse cercata.

Merita osservare che, essendo noti i vertici del triangolo P_1, P_2 e P_3, su cui dovranno giacere i punti di tangenza, il primo fuoco F può essere collocato in una posizione qualsiasi all'interno di quel triangolo. Per questo motivo, ogni triangolo può contenere infinite ellissi tangenti internamente ai suoi lati. Ma, specificato un fuoco, l'ellisse cercata è unica.

Secondo le indicazioni indirettamente contenute in Alves (2018), ma esplicitamente indicate in Candia (2018), la determinazione del secondo fuoco e dei punti di tangenza avviene seguendo i seguenti passi.

```
\unitlength=0.009\linewidth
\begin{picture}(100,100)(-50,-50)
\AutoGrid
\Pbox(0,0)[rt]{O}[3]
\VECTOR(0,-50)(0,50)\Pbox(0,50){r}{y}[0]
\VECTOR(-50,0)(50,0)\Pbox(50,0){r}{x}[0]
\XSellisse<20>{40}{30}*
\end{picture}
\end{figure}
```

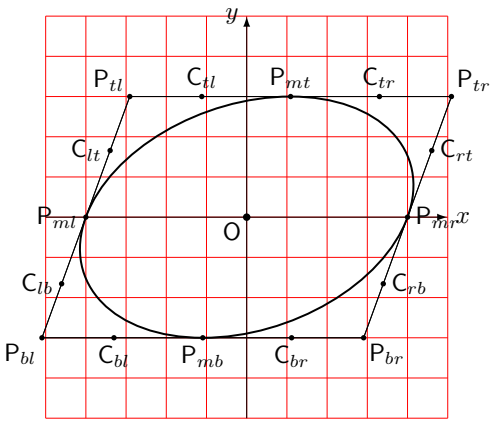

Figura 18: Costruzione di un'ellisse ottenuta con una affinità di scorrimento

```
\unitlength=0.009\linewidth
\begin{picture}(100,100)(-50,-50)
\AutoGrid
\Pbox(0,0)[rt]{O}[3]
\VECTOR(0,-50)(0,50)\Pbox(0,50){r}{y}[0]
\VECTOR(-50,0)(50,0)\Pbox(50,0){r}{x}[0]
\XSellisse<20>{40}{30}
\end{picture}
\end{figure}
```

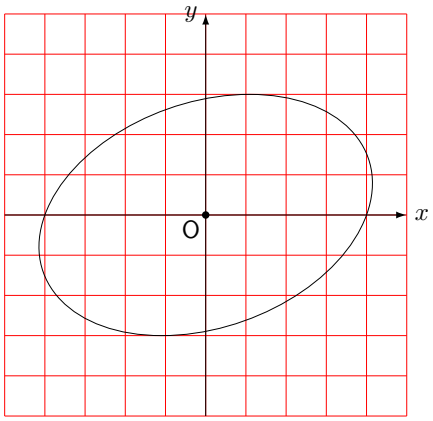

Figura 19: Una ellisse ottenuta con una affinità di scorrimento

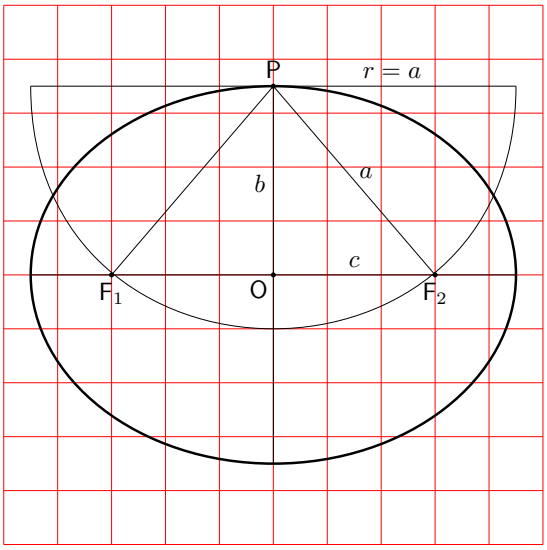

Figura 20: Legame fra gli assi e i fuochi

1. Si determinano i tre punti G_1, G_2 e G_3 esterni al triangolo, in modo che ciascuno sia il simmetrico di F rispetto a ciascun lato. Bisogna quindi determinare le perpendicolari ai lati che si dipartono da F e determinarne le intersezioni M_i con i lati stessi; questi rappresentano i punti medi fra F e ciascun punto G_i:

$$M_i = \frac{F + G_i}{2} \qquad \forall\, i = 1, 2, 3 \qquad (3)$$

Detto in altre parole, ciascun punto M_i è il punto di mezzo fra il fuoco e il corrispondente punto esterno al triangolo. Fra le macro già descritte, abbiamo già `\SegmentCenter`, ma non è questa quella che ci serve, perché qui l'incognita è un estremo del segmento; abbiamo bisogno della soluzione dell'equazione (3)

$$G_i = 2M_i - F \qquad \forall\, i = 1, 2, 3 \qquad (4)$$

Per ciascun valore di i si può usare invece una nuova macro `\SymmetricalPointOf` definita come

```
\def\SymmetricalPointOf#1respect#2to#3{%
  \ScaleVect#2by2to\Segm
  \SubVect#1from\Segm to#3\ignorespaces}
```

dove, nel nostro caso, il primo argomento è costituito dal fuoco, il secondo argomento dal punto medio, il terzo è la macro che riceve le coordinate del punto simmetrico.

2. Disponendo dei tre punti G_i simmetrici rispetto al fuoco, con la macro `\ThreePointCircle`

si può di disegnare un cerchio il cui centro costituisce il secondo fuoco F' dell'ellisse che si sta cercando. Si potrebbe evitare di disegnare questo cerchio esterno che la macro asterisca `\ThreePointCircle` restituisce in `\C`. Quindi bisogna conservare questo valore in una macro il cui nome, possibilmente, ricordi che si tratta di F' (per esempio `\Fp`). Sarebbe ancora più conveniente usare la macro `\ThreePointCircleCenter` assegnando direttamente il centro calcolato alla macro `\Fp`. Ma qui vorremo anche disegnare la circonferenza, quindi è meglio usare la prima macro, che ci dà entrambi i risultati. Questa circonferenza appena descritta viene chiamata in alcune lingue *circonferenza direttrice*; si può constare che questa circonferenza ha molte proprietà interessanti; ma qui non è il caso di scendere nei particolari; basti sapere che essa ha il raggio pari all'asse maggiore $2a$ dell'ellisse. Volendo, quindi, si potrebbero saltare un paio dei punti seguenti in questa enumerazione.

3. I punti di tangenza T_1, T_2 e T_3, invece, sono le intersezioni con i tre lati dei tre raggi che vanno dal centro F' della circonferenza ai tre vertici G_i con i quali la si è costruita. E questa è un'altra proprietà della circonferenze direttrice.
4. Ora si dispone di tutti gli elementi per disegnare l'ellisse; il comando

   ```
   \SegmentCenter(\F)(\Fp)to\C
   ```

   permette di determinare il centro dell'ellisse; il comando

   ```
   \SegmentArg(\F)(\Fp)to\AngFocalAxis
   ```

   permette di determinare l'inclinazione dell'asse focale. Infine il comando

   ```
   \SegmentLength(\F)(\Fp)to\D
   ```

   dà la distanza focale, pari al doppio di c.
5. La macro `\SegmentLength` usata due volte permette di determinare le due distanze di uno qualsiasi dei punti di tangenza dai due fuochi; la loro semisomma dà a.
6. Noti a e c, la relazione (2) diventa la base per definire `\AxisFromAxisAndFocus` con la sintassi:

   `\AxisFromAxisAndFocus`⟨*argomento1*⟩ `and`⟨*argomento2*⟩ `to{`⟨*argomento3*⟩`}`

   e la sua definizione è la seguente:

   ```
   \def\AxisFromAxisAndFocus#1and#2to#3%
      {\fptest{abs(#1)>abs(#2)}%
      {\edef#3{\fpeval{sqrt(#1**2-#2**2)}}}%
      {\edef#3{\fpeval{sqrt(#2**2+#1**2)}}}}
   ```

   La macro è congegnata in modo tale che se il primo argomento è maggiore del secondo fornisce la differenza pitagorica, altrimenti fornisce la somma pitagorica; nel nostro caso a è il primo argomento ed è maggiore del secondo argomento c, quindi il risultato è la differenza pitagorica b.
7. Ora si hanno il centro `\C`, l'inclinazione dell'asse focale `\AngFocalAxis`, e i due semiassi `\a` e `\b`; quindi la macro `\Xellisse` può disegnare l'ellisse cercata.
8. Come nel caso dell'inellisse di Steiner, conviene creare un'unica macro che accetti un asterisco per disegnare tutta la costruzione geometrica o il solo risultato finale.

La figura 21 mostra appunto l'ellisse finale avendo usato il codice:

```
\unitlength0.0095\linewidth
\begin{picture}(150,150)(-30,-20)
\AutoGrid
\EllisseConFuoco(10,20)(120,-10)(0,100)(20,40)
\end{picture}
```

mentre la figura 22 mostra la costruzione per ottenere il risultato; il suo codice è il seguente:

```
\unitlength0.0095\linewidth
\begin{picture}(150,150)(-30,-20)
\AutoGrid
\EllisseConFuoco*(10,20)(120,-10)(0,100)(20,40)
\end{picture}
```

Come si vede, la macro `\EllisseConFuoco` con l'asterisco produce tutta la costruzione; togliendo l'asterisco, il triangolo e l'ellisse vengono disegnati senza le linee e i cerchi usati nella costruzione.

Non resta che esporre la definizione della macro `\EllisseConFuoco`, la cui sintassi è la seguente:

`\EllisseConFuoco`⟨⋆⟩`(`⟨*primo vertice*⟩`)%`
⟨*secondo vertice*⟩`)(`⟨*terzo vertice*⟩`)(`⟨*fuoco*⟩`)`

dove l'asterisco facoltativo specifica se si vuole disegnare l'intera costruzione o se basta disegnare il risultato finale. Seguono le coordinate dei vertici del triangolo e le coordinate del fuoco, tutte racchiuse fra parentesi tonde. La macro che ora viene descritta contiene anche numerosi comandi `\Pbox` per etichettare i vari punti del disegno; i loro parametri facoltativi sono impostati per i disegni della figura 21 e, specialmente, della figura 22; potrebbero non essere ottimali per disegnare una costruzione diversa, cioè con un triangolo diverso e/o una diversa posizione del fuoco. Con un triangolo diverso e/o con il fuoco in un'altra posizione, la non ottimalità si potrebbe manifestare con le etichette dei punti sovrapposte ad alcune linee della costruzione; i pallini neri sono collocati correttamente, ma l'etichetta potrebbe richiedere allineamenti diversi. Spesso per vedere bene tutti gli elementi della costruzione basta ingrandirla a piena giustezza, invece che alla giustezza di una colonna. Tuttavia esistono anche altre vie.

Infatti, l'utente che voglia servirsi di questa macro potrebbe eliminare quasi tutti questi comandi `\Pbox` seguendo l'una o l'altra delle seguenti alternative.

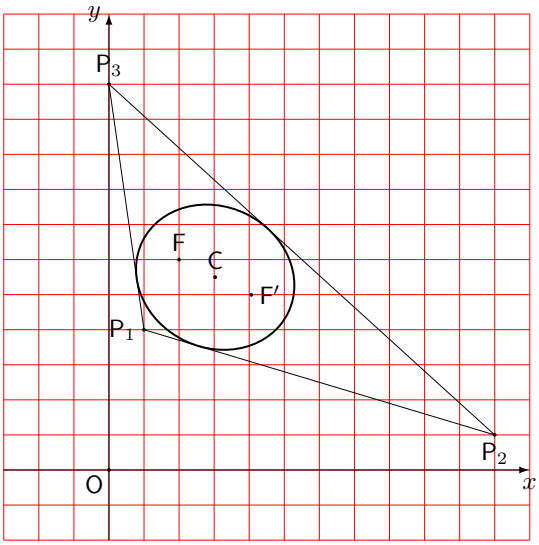

FIGURA 21: Un'ellisse tangente internamente a un triangolo, noto un fuoco dell'ellisse

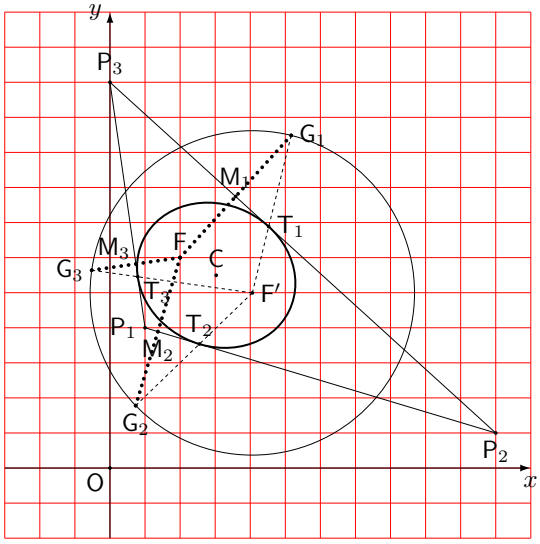

FIGURA 22: Costruzione dell'ellisse tangente internamente a un triangolo, noto un fuoco dell'ellisse

1. Etichetta a mano i punti che ritiene siano da marcare nel suo disegno specifico; la presenza della griglia agevola questo compito perché consente di stimare la posizione dei punti, sia pure in modo approssimato, per cui procedere a mano richiede diverse compilazioni in modo da aggiustare con precisione le coordinate dei punti da marcare.
2. Lascia i comandi `\Pbox` nella macro senza mettere nulla nel campo con l'etichetta, in modo che i pallini siano posti con precisione, poi mette a mano le etichette con altrettanti comandi `\Pbox` impostando a zero il diametro del pallino (ma specificando a mano che vuole comporre i simbolo del punto con il font senza grazie, per esempio `\Pbox(0,110)[b]{\mathsf P_3}[0]`), poiché, quando il diametro del pallino è nullo, l'etichetta viene composta in corsivo matematico. Aggiustare a mano la posizione dell'etichetta è più facile perché non richiede altrettanta precisione di quella richiesta per collocare il pallino al punto giusto.

```
\NewDocumentCommand\EllisseConFuoco%
  {s d() d() d() d()}{\bgroup%
\CopyVect#2to\Puini \CopyVect#3to\Pd
\CopyVect#2to\Pu
\CopyVect#3to\Pd
\CopyVect#4to\Pt
\CopyVect#5to\F
\polygon(\Pu)(\Pd)(\Pt)
\Pbox(\Pu)[r]{P_1}[1.5]
\Pbox(\Pd)[t]{P_2}[1.5]
\Pbox(\Pt)[b]{P_3}[1.5]
\Pbox(\F)[b]{F}[1.5]
\SegmentArg(\Pu)(\Pt)to\At
\SegmentArg(\Pu)(\Pd)to\Ad
\SegmentArg(\Pd)(\Pt)to\Au
\IntersectionOfLines(\Pu)(\At:1)
  and(\F)(\fpeval{\At+90}:1)to\Mt
\IntersectionOfLines(\Pd)(\Ad:1)
  and(\F)(\fpeval{\Ad+90}:1)to\Md
\IntersectionOfLines(\Pd)(\Au:1)
  and(\F)(\fpeval{\Au+90}:1)to\Mu
\IfBooleanTF{#1}%
 {\Pbox(\Mt)[br]{M_3}[1.5]
  \Pbox(\Md)[t]{M_2}[1.5]
  \Pbox(\Mu)[b]{M_1}[1.5]}{}
\SymmetricalPointOf\F respect\Mu to\Gu
\IfBooleanTF{#1}{\Pbox(\Gu)[l]{G_1}[1.5]}{}
\SymmetricalPointOf\F respect \Md to\Gd
\IfBooleanTF{#1}{\Pbox(\Gd)[t]{G_2}[1.5]}{}
\SymmetricalPointOf\F respect \Mt to\Gt
\IfBooleanTF{#1}{\Pbox(\Gt)[r]{G_3}[1.5]}{}
\IfBooleanTF{#1}%
  {\ThreePointCircle*(\Gu)(\Gd)(\Gt)}%
  {\ThreePointCircle(\Gu)(\Gd)(\Gt)}
\CopyVect\C to\Fp \Pbox(\Fp)[l]{F'}[1.5]
\IfBooleanTF{#1}%
  {\Dotline(\F)(\Gt){2}[1.5]
   \Dotline(\F)(\Gd){2}[1.5]
   \Dotline(\F)(\Gu){2}}[1.5]{}
\IntersectionOfSegments(\Pu)(\Pt)
  and(\Fp)(\Gt)to\Tt
\IntersectionOfSegments(\Pu)(\Pd)
  and(\Fp)(\Gd)to\Td
\IntersectionOfSegments(\Pd)(\Pt)
  and(\Fp)(\Gu)to\Tu
\IfBooleanTF{#1}%
  {\Pbox(\Tu)[l]{T_1}[1.5]
   \Pbox(\Td)[b]{T_2}[1.5]
   \Pbox(\Tt)[tl]{T_3}[1.5]
   \Dashline(\Fp)(\Gu){1}
   \Dashline(\Fp)(\Gd){1}
   \Dashline(\Fp)(\Gt){1}}{}
\DistanceAndDirOfVect\Fp minus\Tt
  to\DFp and\AFu
\DistanceAndDirOfVect\F  minus\Tt
  to\DF and\AF
\SegmentCenter(\F)(\Fp)to\CE
\Pbox(\CE)[b]{C}[1.5]
```

```
\edef\a{\fpeval{(\DFp+\DF)/2}}
\SegmentArg(\F)(\Fp)to\AngFocalAxis
\SegmentLength(\F)(\CE)to\c
\AxisFromAxisAndFocus\a and\c to\b
\Xellisse(\CE)[\AngFocalAxis]{\a}{\b}%
  [\thicklines]
\VECTOR(-30,0)(120,0)
\Pbox(120,0)[t]{x}[0]
\VECTOR(0,-20)(0,130)
\Pbox(0,130)[r]{y}[0]
\Pbox(0,0)[tr]{O}[1.5]
\egroup\ignorespaces}
```

4 Commenti

Questo articolo propone alcune tecniche che si possono usare per disegnare costruzioni geometriche che richiedano solamente l'uso di riga, squadra e compasso. Certamente con METAPOST alcune di queste costruzioni potrebbero essere più semplici, perché METAPOST è nato per lavorare con numeri complessi, ed è in grado di risolvere equazioni. In realtà è in grado di determinare le intersezioni non solo di segmenti, ma anche di curve di Bézier cubiche; siccome è nato da METAFONT, la precisione dei calcoli era sufficiente per disegnare i caratteri, ma non era, forse, sufficientemente precisa in altri casi.

Knuth, nel suo testo (Knuth, 1986) dice che in realtà lui usava impropriamente METAFONT dalla finestra comandi come se fosse una calcolatrice, tanto era soddisfatto dei calcoli che eseguiva; ma erano calcoli basati sull'aritmetica intera su cui sono fondati sia METAFONT, sia TEX. Con questo tipo di aritmetica, i numeri fratti sono trattati come numeri a virgola fissa, con 16 cifre binarie nella parte fratta; poi sono scalati mediante il fattore 2^{16} in modo che siano espressi come numeri interi; questa notazione a virgola fissa è usata in entrambi i programmi per esprimere le lunghezze al loro interno mediante il loro valore intero espresso in *scaled points*.

Inoltre la sintassi del linguaggio di METAPOST è decisamente diversa da quella di TEX e LATEX, per cui l'utente deve imparare un altro linguaggio di programmazione. Se questo valga la pena rispetto a quanto ho descritto in questo articolo è più una questione di gusti personali, che di efficienza e praticità d'uso. Tuttavia, personalmente, preferisco definire e usare macro in linguaggio TEX, tanto più che ora, con il linguaggio LATEX 3, le prestazioni sono quasi illimitate. Questo "quasi" dipende dal fatto che, per quanto io sappia, questo linguaggio ancora non consente di risolvere direttamente le equazioni. Anche i semplici sistemi lineari richiedono di determinare analiticamente le formule risolutive che poi sono facili da implementare con le funzionalità del pacchetto xfp. La precisione dei calcoli è certamente maggiore di quanto serva per la grafica.

Tuttavia, sono convinto che sia i docenti delle scuole secondarie di secondo livello, sia i docenti universitari dei corsi propedeutici di matematica e geometria possono trovare molto giovamento se, invece di dettare appunti (o lasciare che gli allievi prendano appunti), distribuissero in rete dei fascicoli, o dispense, delle loro lezioni; invece dei terribili formati EPUB, buoni per un romanzo, ma decisamente scadenti per un testo scientifico, è meglio che usino il formato standard PDF, in modo che gli allievi di oggi, informatizzati fin dai primi mesi di vita, possano sfruttare i loro potenti mezzi per disporre di scritti, composti, e disegnati come si deve.[10]

5 Ringraziamenti

Ringrazio di cuore Enrico Gregorio che mi ha aiutato molto a capire quello che si può fare con la libreria per il calcolo decimale in virgola mobile del linguaggio LATEX 3. Mi ha anche scritto un documento (incompiuto, ma per me sufficiente) che forse in futuro sostituirà la scarna documentazione di xfp oggi distribuita con il sistema TEX.

Ringrazio anche Francesco Biccari che, avendo desiderio di comporre carte millimetrate lineari e logaritmiche, mi ha stimolato ad approfondire la mia conoscenza della libreria di calcolo del linguaggio LATEX 3.

Riferimenti bibliografici

Alves, Sergio (2018). «Elipses inscritas num triângulo». *Revista do Professor de Matemática*, **96**. Questo è il sito da dove è possibile scaricare l'articolo solo da parte dei soci. L'articolo in formato PDF è però scaricabile dalla rete eseguendo una ricerca con la stringa "Sergio Alves Elipses inscritas num triangulo".

Beccari, Claudio (2018a). «Introduzione a xparse». *ArsTEXnica*, (26).

— (2018b). *PM-ISOmath – The poor-man ISO math bundle*. GUIT. Version 1.0.04. Leggibile con `texdoc pm-isomath`.

— (2020a). *The extension package curve2e*. GUIT. Version 2.0.8. Leggibile con `texdoc curve2e`.

— (2020b). «The curve2e manual». GUIT. Leggibile col comando `texdoc curve2e-manual`.

— (2020c). «The euclideangeometry package». PDF document. Leggibile col comando `texdoc euclideangeometry`.

— (2020d). «The euclideangeometry package manual». PDF document. Leggibile col comando `texdoc euclideangeometry-man`.

10. È bene ricordare che il formato PDF è quello prescritto dalle norme ISO per l'archiviabilità dei documenti in formato elettronico, (C.V. Radhakrishnan *et al.*, 2018); per comporre la matematica con le notazioni per la fisica e della tecnologia conformi con le norme ISO, vedano (Beccari, 2018b).

Candia, Estevão Vinicius (2018). *Matemática e o METAPOST*. Tesi di Dottorato, Universidade Federal de Mato Grosso do Sul.

— (2019). «A Brazilian Portuguese work on MetaPost, and how mathematics is embedded in it». *TUGboat*, **40** (3), pp. 247–250.

C.V. Radhakrishnan, Hàn Thế Thành, Ross Moore e Peter Selinger (2018). *Generation of PDF/X- and PDF/A- compliant PDFs with* `PdfTeX`. River Valley Technologies, Trivandrum, India. Versione 1.6. Leggibile con `texdoc pdfx`.

De Marco, Agostino (2009). «Produrre grafica vettoriale di alta qualità programmando `asymptote`». *ArsTEXnica*, (8), pp. 25–39.

— (2019). «Graphics for LaTeX users». *ArsTEXnica*, (28).

Fuster, Robert (2012). «The xpicture package – Several extensions of the picture standard environment – User Manual». PDF document. Leggibile col comando `texdoc xpicture`.

Gässlein, Hubert, Rolf NiePraschk e Josef Tkadlec (2016). *The pict2e package*. TUG. Version 0.3b. Readable with `texdoc pict2e`.

Hammerlind, Andy, John Bowman e Tom Prince (2018). *Asymptote: the vector graphics language*. TUG. Version 244. Readable with `texdoc asymptote`.

Hobby, John D. (2018). *METAPOST– A users' manual*. TUG. Version 2.00 (2.0rc2). Readable with `texdoc metapost`.

Knuth, Donald Ervin (1986). *The METAFONT book*. Addison Wesley, Reading, Mass.

Lamport, Leslie (1994). *A document preparation system — LaTeX — User's guide and reference manual*. Addison Wesley, Reading, Mass., 2ª edizione.

Matthes, Alain (2020). «AlterMundus — `tkz-euclide`». PDF document. Leggibile col comando `texdoc tkzdoc-euclide`. Il pacchetto è interessantissimo e ricopre in parte quanto descritto in questo articolo. Ma la documentazione, pur estesa, lascia a desiderare, con diversi esempi non documentati affatto; oltretutto con cambi continui di lingua dall'inglese al francese e viceversa.

Miklavec, Mojca (2013). *Using* `context` *and tikz terminals for gnuplot in ConTEXt*. Readable with `texdoc gnuplot`. This document gives instructions to install the external program `gnuplot`, version 4.6.0 or later, and to configure it to be used by the typesetting programs of the TeX system.

Tantau, Till (2019). *TikZ & PGF*. TUG. Version 3.1.1 – Readable with `texdoc tikz` or with `texdoc pgf` or with `texdoc pgfmanual`.

Van Zandt, Timothy (2003). *PSTricks – User's guide*. TUG. Version 97. Readable with `texdoc pstricks`.

▷ Claudio Beccari
`claudio dot beccari at gmail dot com`