

# Illustrazioni tridimensionali con Sketch/L<sup>A</sup>T<sub>E</sub>X/PSTricks/Tikz nella didattica della Dinamica del Volo

A. De Marco

## Sommario

In questo articolo viene mostrato come sia possibile utilizzare le potenzialità di L<sup>A</sup>T<sub>E</sub>X e dei pacchetti PSTricks e Tikz nella produzione di illustrazioni avanzate. La creazione di disegni che rappresentino delle scene tridimensionali con indicazioni di contenuto scientifico è di fatto possibile con L<sup>A</sup>T<sub>E</sub>X. L'autore mostra in che modo si riesca a manipolare e disporre oggetti tridimensionali in una scena con il programma Sketch di Eugene Ressler ed il suo intuitivo linguaggio di *scripting*, ottenendo un'*output* sotto forma di comandi PSTricks o Tikz. Il metodo di lavoro proposto permette di superare le limitazioni che gli utenti di un pacchetto come PSTricks, che pure possiede funzionalità di disegno tridimensionale relativamente avanzate, incontrano ogni qual volta si accingono a voler disegnare e manipolare scene tridimensionali contenenti oggetti non semplici e non *primitivi*.

La didattica di una materia ingegneristica come la Dinamica del Volo è un campo in cui l'autore opera ed al quale si riferiscono gli esempi concreti riportati nell'articolo.

## Abstract

This article shows how the combination of L<sup>A</sup>T<sub>E</sub>X with the package PSTricks or with Tikz can be used to produce advanced, nice-looking illustrations. As a matter of fact, the creation of drawings representing three-dimensional scenes with scientific or non-trivial annotations is possible with L<sup>A</sup>T<sub>E</sub>X. One of the goals of the article is introducing the program Sketch, by Eugene Ressler, and how one can manipulate and put in place objects in a three-dimensional scene by means of its intuitive scripting language. The output of Sketch is a set of PSTricks or Tikz commands that might be included by a master L<sup>A</sup>T<sub>E</sub>X document to produce the final picture. The technique proposed here enables to go over the limitations encountered by PSTricks or Tikz users when it comes to representing non-trivial three-dimensional scenes.

As a teacher of engineering subjects related to Flight Dynamics, I have reported some concrete examples that may help to better understand the potential of Sketch and of the workflow proposed in the article.

## 1 Introduzione

Il titolo dell'articolo non tragga in inganno il lettore: non si vuole esporre un argomento di nicchia. Piuttosto si intende trattare un problema che si presenta nei settori più svariati dell'ingegneria e delle scienze, quello della creazione di buone illustrazioni tridimensionali ad alto contenuto tecnico. L'approccio è quello di presentare dei casi studio basati su esperienze reali, fornendone i particolari implementativi, anziché impostare una discussione che replicherebbe quanto è già trattato nei riferimenti bibliografici.

Introduciamo, per cominciare, il contesto in cui è nato gran parte del materiale di questa comunicazione. Al lettore basti sapere che l'oggetto di studio fondamentale della *Dinamica del Volo* atmosferico è l'evoluzione dei velivoli in un dato sistema di riferimento collegato alla Terra. Insieme al moto dell'aeroplano vengono studiati tutti quei fenomeni fisici che lo determinano. Tra questi ricorrono un'importanza particolare i fenomeni di natura aerodinamica ed ancor più il modo in cui essi dipendono dalla configurazione architettonica del velivolo e dalla condizione di volo. Da qui si comincia a comprendere la necessità in questo campo, e in particolare nella didattica di tali argomenti, di trovare modi efficaci di rappresentare dei concetti fisico-matematici legati al volo. L'esperienza dell'autore è che con l'ausilio di rappresentazioni tridimensionali ben curate l'esposizione di molti concetti si arricchisce in immediatezza visiva e diventa enormemente più efficace. Ciò è vero non solo nella didattica ma anche in comunicazioni di carattere squisitamente scientifico.

La Figura 1 mostra un primo esempio di illustrazione utilizzata tipicamente nello studio della Dinamica del Volo per introdurre alcuni elementi di base. Qui si nota subito come la necessità di definire componenti di vettori – le componenti  $u$ ,  $v$  e  $w$  della velocità  $\vec{V}$  del baricentro  $G$ , quelle  $p$ ,  $q$  ed  $r$  della velocità angolare istantanea  $\vec{\Omega}$ , le componenti  $X$ ,  $Y$ ,  $Z$  ed  $\mathcal{L}$ ,  $\mathcal{M}$ ,  $\mathcal{N}$  delle risultanti delle forze e dei momenti esterni – lungo gli assi di un riferimento standard solidale al velivolo (riferimento degli *assi velivolo* o *body-fixed frame*) richiede naturalmente l'utilizzo di una rappresentazione tridimensionale.

Un secondo esempio di illustrazione è riportato

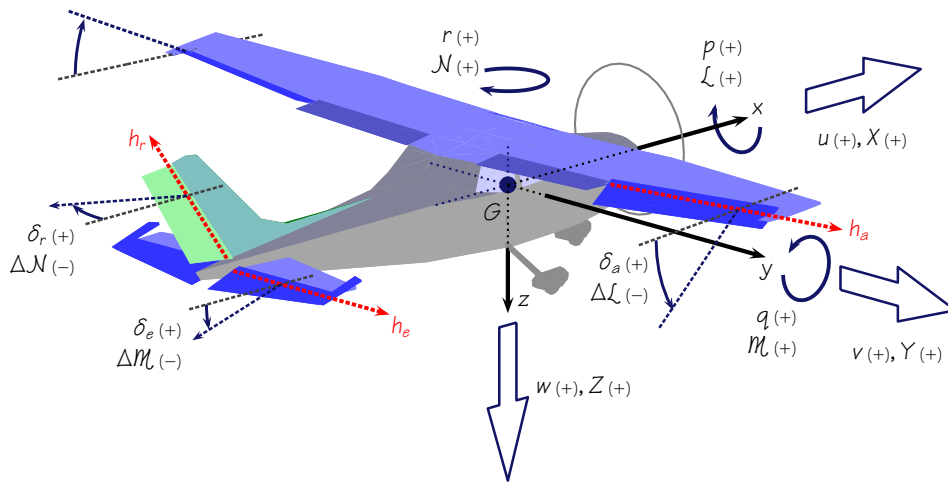


FIGURA 1: Esempio di definizione di alcune tra le principali grandezze fisiche collegate alla dinamica del volo di un velivolo. Per conferire ai disegni l'aspetto di un'illustrazione d'altri tempi l'autore spesso usa una versione modificata del font *Informal Math* della MicroPress Inc. (<http://www.micropress-inc.com/>).

in Figura 2. Da questa si può osservare come, in relazione al fatto che il volo è un fenomeno squisitamente tridimensionale, si rendono spesso necessari dei disegni più o meno articolati se si vogliono rappresentare in maniera efficace alcuni dettagli teoricamente importanti. Quando un velivolo possiede un assetto di volo non simmetrico, Figura 2(b), vi sono cioè un angolo d'attacco  $\alpha$  e di derapata  $\beta$  entrambi non nulli, i contributi all'azione aerodinamica totale di alcuni elementi architettonici del velivolo vanno evidenziati mediante l'uso di opportuni pedici (pedici "A"). Ad esempio: impennaggio verticale (*Vertical tail*, "V"), impennaggio orizzontale (*Horizontal tail*, "H"), alettone sinistro/destro (*left/right aileron*, "ail,r/l"), combinazione ala-fusoliera (*Wing-Body*, "WB"). La Dinamica del Volo è un esempio di quei campi in cui l'uso di pedici multipli è molto praticato, al punto che esistono precise convenzioni sull'uso delle notazioni dettate da organismi internazionali. I dettagli a cui l'uso di pedici multipli si riferisce hanno ovviamente una corrispondenza di natura fisica. La Figura 2(b) mostra una soluzione grafica che espone in un solo disegno un concetto che pure è espresso efficacemente da un appropriato modello matematico. Si può scrivere per esempio che:

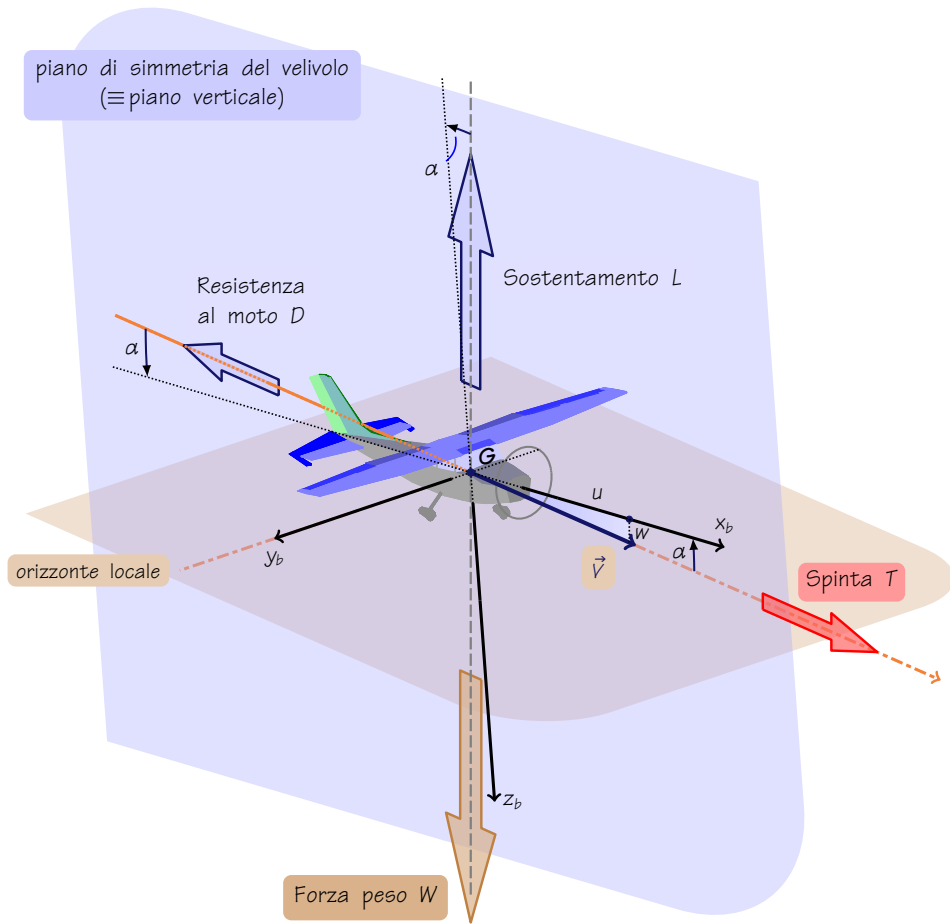
$$\mathcal{L}_A = \mathcal{L}_{A,WB} + \mathcal{L}_{A,V} + \mathcal{L}_{A,ail,r} + \mathcal{L}_{A,ail,l} \quad (1)$$

Una formula come quella appena scritta per uno studente di ingegneria aerospaziale dovrebbe essere abbastanza consistente da fargli subito interpretare la quantità  $\mathcal{L}_A$  come la *coppia di rollio* di natura aerodinamica (che tende a far ruotare l'aereo intorno all'asse della fusoliera). Gli addendi a secondo membro costituiscono dunque una semplice sovrapposizione di effetti dovuti, rispettivamente, alla combinazione aerodinamica ala-fusoliera ( $\mathcal{L}_{A,WB}$ ), all'impennaggio verticale di coda ( $\mathcal{L}_{A,V}$ )

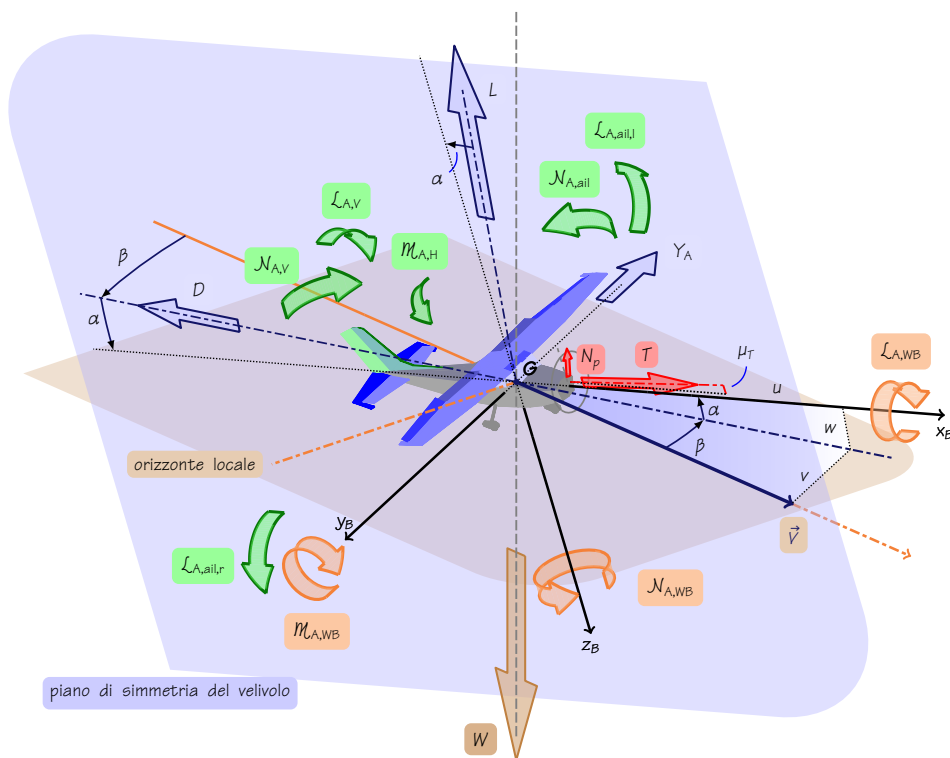
ed ai due alettoni, destro e sinistro ( $\mathcal{L}_{A,ail,r}$  e  $\mathcal{L}_{A,ail,l}$ ). Si potrebbe osservare a questo punto che la semplicità della formula (1) è superiore in immediatezza all'illustrazione di Figura 2(b). Ciò è ovviamente vero se ci si sofferma al fatto che una grandezza fisica – il primo membro della (1) – è data semplicemente dalla sovrapposizione di più contributi – cioè la somma a secondo membro. La potenza di una illustrazione completa come quella in questione si rivela quando si vuole spingere il fruitore, lo studente o il collega ad esempio, a immaginare da quali altre grandezze fisiche questi addendi dipendono e, per ciascuno di essi, in che misura. Dunque, non basta qui immaginare una semplice somma ma bisogna immaginare ad esempio come cambiano i singoli effetti al variare dell'angolo d'attacco o dell'angolo di derapata. Un tale sforzo cognitivo richiede di immaginare una variazione tridimensionale d'assetto del velivolo. Si vede allora come un'illustrazione tridimensionale ben concepita possa diventare un incredibile ausilio, specialmente per i soggetti non dotati di approccio "visuale" al ragionamento.

Per approfondimenti sulla Dinamica del Volo si rimanda a ETKIN (1982), SCHMIDT (1998), CALCARA (1988), CASAROSA (2004), MENGALI (2001).

L'elemento più rilevante delle figure richiamate sopra è costituito dalla vista assonometrica di un modello di velivolo. Le illustrazioni sono state realizzate con strumenti, per così dire "nativamente" 3D. In altre parole, non si tratta di tradizionali schizzi piani che nascono dalla mano (digitale) di un illustratore, ma di rappresentazioni (proiezioni) in un piano di oggetti posti in una scena tridimensionale. Simili risultati visivi vengono ottenuti tipicamente quando si effettua il *rendering* di una scena in applicazioni di computer grafica come *Blender* (ROOSEDAAL, 2007) o di simili prodotti com-



(a) Condizione di volo simmetrico, orizzontale, ad ali livellate



(b) Condizione di volo non simmetrico, orizzontale, ad ali non livellate

FIGURA 2: Definizione di azioni esterne su di un velivolo. Esse comprendono la forza peso  $\vec{W}$  e quelle di natura aerodinamica e propulsiva. In volo simmetrico (in alto) la situazione sembra molto più semplice di quella che si presenta per un velivolo in assetto di volo non simmetrico (in basso), quando vi sono un angolo d'attacco  $\alpha$  e di derapata  $\beta$  entrambi non nulli. Nella seconda figura si sono evidenziati con opportuni pedici i contributi all'azione aerodinamica (A) totale di alcuni elementi architettonici del velivolo.

merciali. Un fatto degno di nota è che le illustrazioni ottenute con il metodo proposto in questo articolo, grazie alla tecnologia Postscript e all'uscita di L<sup>A</sup>T<sub>E</sub>X in formato PDF, sono intrinsecamente vettoriali.

Nei paragrafi seguenti si cercherà di raccogliere gli elementi che permettano al lettore di creare le proprie illustrazioni 3D e di aggiungervi comodamente il testo ed i simboli matematici desiderati attraverso gli usuali comandi L<sup>A</sup>T<sub>E</sub>X.

## 2 Illustrazioni tridimensionali

Creare illustrazioni tridimensionali di buona qualità da utilizzare in pubblicazioni tecniche e scientifiche non è un compito banale. E ciò si intuisce ovviamente dall'esame delle figure citate in precedenza.

Dal punto di vista editoriale i disegni in questione possiedono delle caratteristiche importanti: (i) le annotazioni contengono testo, lettere greche e simboli matematici composti con L<sup>A</sup>T<sub>E</sub>X – risulteranno quindi perfettamente integrate con il resto di un eventuale documento composto in L<sup>A</sup>T<sub>E</sub>X che le contenga; (ii) i disegni sono prodotti in forma vettoriale (formati EPS e PDF); (iii) le distanze, le proporzioni e gli angoli sono accurati e visivamente corretti.

### 2.1 La via di PSTricks

Per la cronaca, nei suoi primi tentativi di creare disegni con simili caratteristiche l'autore ha tentato la via di PSTricks (TUG BOAT, 2007; VAN ZANDT, 2003). PSTricks rappresenta una collezione impressionante di macro per la creazione di disegni ed illustrazioni con T<sub>E</sub>X/L<sup>A</sup>T<sub>E</sub>X. È sufficiente visitare il sito di riferimento di questo pacchetto e consultare la galleria degli esempi per apprezzarne le potenzialità. PSTricks si dimostra uno strumento eccellente per illustrazioni bidimensionali. È possibile anche utilizzare alcune estensioni al pacchetto originale, come ad esempio i pacchetti `pst-3dplot` e `pst-3d`, per creare semplici disegni tridimensionali.

Quando si intende rappresentare una scena tridimensionale non eccessivamente semplice è fondamentale disporre di strumenti pratici che implementino degli operatori matematici di trasformazione geometrica. Tra questi compaiono immancabilmente le matrici di rotazione, le trasformazioni di traslazione, di scalatura e deformazione. Di fatto, questi operatori non sono disponibili in PSTricks oppure sono implementati con funzionalità limitate ad oggetti semplici in estensioni come `pst-3dplot`. PSTricks è un pacchetto basato su T<sub>E</sub>X e sul linguaggio Postscript e si può pensare di estenderne le possibilità di manipolazione nel mondo 3D. Sebbene sia certamente possibile, ciò è anche estremamente difficile (almeno per i non “T<sub>E</sub>X guru”).

### 2.2 Il programma Sketch

Esiste fortunatamente uno strumento che permette di ottenere dei risultati avanzati e che al tempo stesso si inserisce agilmente nel flusso di lavoro di produzione delle illustrazioni. Il suo nome è *Sketch*.

Sketch è un programma scritto in linguaggio C da Gene Ressler (RESSLER, 2007a), multipiattaforma e distribuito con licenza GPL. Si presenta come uno strumento leggero e semplice per la produzione di disegni 2D e 3D. L'autore di Sketch dichiara nel manuale (RESSLER, 2007b) di averlo concepito per creare delle illustrazioni raffinate, che contenessero in prevalenza annotazioni matematiche e che, quando incluse in documenti scientifici, non presentassero dettagli disomogenei all'impostazione tipografica adottata.

I disegni con Sketch sono composti a partire da oggetti primitivi, come punti e linee, da superfici definite per triangolazione ed eventualmente da oggetti testuali. Con opportuni comandi gli oggetti sono posti in una scena e quest'ultima viene proiettata in un piano secondo le classiche regole della prospettiva, in base ad un assegnato punto di vista ed ad una data direzione di osservazione della camera.

Sketch funziona come un comune compilatore ed è programmabile attraverso un linguaggio abbastanza intuitivo. L'output generato può essere del codice PSTricks o Tikz (TANTAU, 2006a,b; FAUSKE, 2007a) da includere opportunamente in un ambiente `figure`. Grazie ad un'apposita opzione della riga di comando è anche possibile ottenere in uscita un sorgente compilabile direttamente da L<sup>A</sup>T<sub>E</sub>X e produrre l'illustrazione desiderata in una semplice catena di compilazioni.

L'algoritmo di composizione della scena di Sketch possiede due caratteristiche particolarmente rilevanti: viene applicata per default la rimozione degli oggetti nascosti – meglio nota come algoritmo di *hidden surface removal* – ed esiste la possibilità di includere nei disegni delle annotazioni in forma di comandi L<sup>A</sup>T<sub>E</sub>X/PSTricks/Tikz, ancorate in una posizione qualsiasi della scena. Quest'ultima caratteristica offre agli utenti L<sup>A</sup>T<sub>E</sub>X degli ovvi vantaggi.

Quando si scopre Sketch ci si rende conto di avere accesso a tutti quegli strumenti matematici e geometrici di cui si ha bisogno per la produzione di disegni tridimensionali e che questi strumenti rendono molto più semplice e veloce il flusso di lavoro. Ad, esempio, in Sketch è così semplice cambiare il punto di vista ed ispezionare la scena che si può dire di avere a disposizione anche uno strumento di *debug visuale* dell'illustrazione.

È opinione dell'autore che i risultati ottenuti con Sketch ripaghino di gran lunga l'utente del tempo speso a studiarne uso e linguaggio.

### 3 Esempi introduttivi di programmazione in Sketch

Il materiale di questo paragrafo è ispirato in parte al manuale ufficiale di Sketch (RESSLER, 2007b) ed in parte al *tutorial* su Sketch presente sul sito web del norvegese Kjell Magne Fauske (FAUSKE, 2007b,c). Di quest'ultimo sarà istruttivo consultare, in particolare, la pagina dedicata ad una galleria di esempi realizzati con il pacchetto Tikz (FAUSKE, 2007a).

#### 3.1 Il ciclo di lavoro

Consideriamo per cominciare una semplice scena tridimensionale costituita dalla piramide e dal sistema di assi cartesiani di Figura 3.

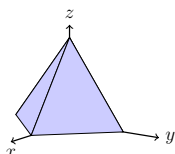


FIGURA 3: Una piramide con Sketch.

Per comporre questa semplice scena con Sketch basta scrivere un codice sorgente come quello riportato nel listato seguente:

```

1 def p0 (0,0,0) % "punto", in origine
2 def vK [0,0,1] % "vettore", asse di rotazione
3 def dx 2.3 % "scalare"
4 def dy 2.5
5 def dz dx
6
7 % definiamo un oggetto "drawable"
8 def Axes {
9   % punti sugli assi
10  def pX (dx,0,0)
11  def pY (0,dy,0)
12  def pZ (0,0,dz)
13  % uniamo i punti
14  line[arrows=>,line width=.8pt](p0)(pX)
15  line[arrows=>,line width=.8pt](p0)(pY)
16  line[arrows=>,line width=.8pt](p0)(pZ)
17  % annotazioni: etichette degli assi
18  special |
19    \path #1 node[above] {$z$}
20          #2 node[below] {$x$}
21          #3 node[right] {$y$};|
22    (pZ)(pX)(pY)
23 }
24
25 % definiamo un secondo oggetto "drawable"
26 def Pyramid {
27   def p0 (0 ,0,2)
28   def p1 (1.5,0,0)
29   def n 4
30   % definiamo una "trasformazione"
31   def tMyRotation rotate(360 / n, (p0), [vK])
32   % piramide come rotazione di una linea
33   % intorno all'asse [vK] in n=4 passi
34   % in gergo: uno "sweep"
35   sweep[cull=false,fill=blue!20]
36     { n, [[tMyRotation]] } line(p0)(p1)
37 }
38
39 % infine definiamo una scena e
40 % rendiamo visibile il tutto
41
42 % collezione di oggetti "drawable"
43 def Scene { {Pyramid} {Axes} }

```

```

44
45 def pEye (7,5,2) % punto di vista
46 def pLookAt (0,0,0) % target camera
47
48 % disegno della scena
49 put { view( (pEye), (pLookAt), [0,0,1] ) }
50 {Scene}
51
52 global { language tikz }

```

Se questo codice viene conservato in un file, ad esempio `pyramid.sk`, e compilato con il comando:<sup>1</sup>

```
> sketch.exe pyramid.sk -o pyramid.tex
```

si otterrà in uscita il file `pyramid.tex`. Date le sue dimensioni contenute, il suo listato viene riportato qui di seguito integralmente:

```

1 \begin{tikzpicture}[join=round]
2 \filldraw[fill=blue!20]
3   (0,1.897)--(.849,.269)--
4   (1.189,-.192)--(0,1.897)--cycle;
5 \filldraw[fill=blue!20]
6   (0,1.897)--(-1.189,.192)--
7   (.849,.269)--(0,1.897)--cycle;
8 \filldraw[fill=blue!20]
9   (0,1.897)--(-.849,-.269)--
10  (-1.189,.192)--(0,1.897)--cycle;
11 \draw[arrows=-,line width=.8pt]
12   (0,0)--(0,1.897);
13 \draw[arrows=-,line width=.8pt]
14   (0,0)--(-.849,-.269);
15 \draw[arrows=-,line width=.8pt]
16   (0,0)--(1.189,-.192);
17 \filldraw[fill=blue!20]
18   (0,1.897)--(1.189,-.192)--
19   (-.849,-.269)--(0,1.897)--cycle;
20 \draw[arrows=>,line width=.8pt]
21   (0,1.897)--(0,2.182);
22 \draw[arrows=>,line width=.8pt]
23   (1.189,-.192)--(1.981,-.321);
24 \draw[arrows=>,line width=.8pt]
25   (-.849,-.269)--(-1.302,-.413);
26
27 \path (0,2.182) node[above] {$z$}
28        (-1.302,-.413) node[below] {$x$}
29        (1.981,-.321) node[right] {$y$};
30 \end{tikzpicture}

```

Esso contiene dei comandi Tikz, racchiusi in un ambiente `tikzpicture`, necessari a generare il disegno di Figura 3. In questo contesto non interessa entrare nei dettagli dei comandi Tikz, per i quali si rimanda alla documentazione, piuttosto importa chiarire come vada utilizzato Sketch.

Se si predisponesse il seguente prototipo di file principale di comandi L<sup>A</sup>T<sub>E</sub>X:

```

1 \documentclass[a4paper,12pt]{article}
2 \usepackage{amsmath}
3 \usepackage{tikz}
4 \usetikzlibrary{%
5   arrows,snakes,backgrounds,patterns}
6 \begin{document}
7 \pagestyle{empty}
8 \vspace*{\fill}
9 \begin{center}
10 %-----
11 %%SKETCH_OUTPUT%
12 %-----

```

1. Si supponga di lavorare su piattaforma Windows in una finestra di comandi DOS o, in alternativa, con una *bash shell* di Cygwin, o Linux, o Mac OS X.

```

13 \end{center}
14 \vspace*{\fill}
15 \end{document}

```

e lo si nomina, ad esempio, `main_template.tex`, con il comando

```
> sketch.exe -t main_template.tex \
    pyramid.sk -o main.tex
```

Sketch sostituirà la stringa `%%SKETCH_OUTPUT%%` con i comandi `Tikz` che generano il disegno e l'utente dovrà semplicemente compilare il file `main.tex` per ottenere l'uscita finale in formato Postscript o PDF:

```
> pdflatex main
> cp main.pdf pyramid.pdf
```

La Figura 4 presenta il flusso di lavoro di un utente che voglia creare un'illustrazione con Sketch, LATEX ed i pacchetti PSTricks e Tikz.

### 3.2 Scopriamo la sintassi di Sketch

Dall'esame del listato del file `pyramid.sk` si noteranno molte delle caratteristiche del linguaggio di Sketch. Per brevità si cercherà, ove è possibile, di spiegare il significato delle varie istruzioni con una notazione matematica standard.

Come si intuisce dal nome, il comando `def` viene usato per definire variabili, oggetti della scena, trasformazioni ed entità geometriche in generale. Tra le entità matematiche e geometriche fondamentali messe a disposizione da Sketch troviamo le seguenti. Le grandezze *scalari*, ad esempio:

```
def xA 2.0 def yA 0.0 def zA 0.0 % ← xA, yA, zA
def xB 4.5 def yB 0.0 def zB 2.0 % ← xB, yB, zB
```

I punti:

```
def pO (0,0,0) % ← O ≡ (0,0,0)
def pA (xA,yA,zA) % ← A ≡ (xA, yA, zA)
def pB (xB,yB,zB) % ← B ≡ (xB, yB, zB)
def pH ((pB)'x, (pB)'y, 0.0) % ← H ≡ (xB, yB, 0)
```

I vettori:

```
def vK [0,0,1] % ← 0· $\vec{i}$ +0· $\vec{j}$ +1· $\vec{k}$ 
def vAB [xB-xA,yB-yA,zB-zA]
def vOB (pB)-(pO) % ←  $\vec{v}_{OB} = B - O$ 
def pC (pA)+[0,2,0] % ← C = A + 2 $\vec{j}$ 
def vAC (pC)-(pA) % ←  $\vec{v}_{AC} = C - A$ 
% prodotto vettoriale
def vN_ABC [vAB]*[vAC] % ←  $\vec{v}_{AB} \times \vec{v}_{AC}$ 
def vUnitN_ABC unit([vN_ABC]) % ← versore  $\vec{n}_{ABC}$ 
% proiezione sul piano xy
def vNxy [ [vUnitN_ABC]'x,
           [vUnitN_ABC]'y,
           0 ] % ←  $\vec{n}_{xy} \cdot \vec{k} \equiv 0$ 
% centro del triangolo
def pM0 ( ((pA)'x+(pB)'x+(pC)'x)/3,
          ((pA)'y+(pB)'y+(pC)'y)/3,
          ((pA)'z+(pB)'z+(pC)'z)/3 )
% punto lungo la normale
def pM1 (pM0)
          +2.0*[vUnitN_ABC] % ← M1 ≡ M0 + 2 $\vec{n}_{ABC}$ 
```

Le trasformazioni:

```
% rotazione
def tRot rotate(180, (pH), [0,0,1])
% traslazione
def tTransl translate(2*[1,0,0])
% scaling
```

```
def sfz 0.8 def tScaleZ scale([1.0,1.0,sfz])
% roto-traslazione
def tMyT [[tRot]] then [[tTransl]]
```

Le istruzioni riportate sopra costituiranno un secondo esempio di composizione di una scena. Esse lasciano intuire la enorme potenzialità del simbolismo messo a disposizione dal linguaggio Sketch. Ciò che si vuole ottenere è il disegno di Figura 5.

Per quanto riguarda la designazione dei nomi delle variabili in Sketch, valgono le stesse regole del linguaggio C. Lo stesso dicasi per le usuali operazioni aritmetiche tra grandezze scalari. Esiste anche la possibilità di usare le più comuni funzioni matematiche. Ad esempio:

```
% distanze tra punti, moduli di vettori
def dAH |(pH)-(pA)| % ← dAH = |H - A|
def dBH |(pB)-(pH)|
% funzioni trigonometriche
def alpha atan2(dBH,dAH) % gradi
def dAB |dAH/cos(alpha)| % valore assoluto
                                % cos() accetta gradi
```

Per quanto riguarda i punti dello spazio di coordinate  $(x, y, z)$ , questi saranno definiti con la notazione usuale che fa uso di parentesi tonde (vedi definizione di `pO`, `pA`, ecc.). I nomi di variabili di tipo punto saranno utilizzabili in successive operazioni racchiudendoli tra parentesi tonde (vedi l'uso di `(pO)`, `(pA)`, `(pB)`, nelle definizioni di `pH`, `vOB`, `pC` ecc.).

Per quanto riguarda i vettori, intesi come *direzioni* nello spazio, la loro definizione in Sketch sarà simile a quella dei punti ma farà uso delle parentesi quadre (vedi definizioni di `vK` e `vAB`). Un vettore potrà anche essere definito come *differenza tra due punti* (vedi definizioni di `vOB`, `vAC`, ecc.) ed, analogamente, un punto potrà essere definito come somma di un altro punto e di un vettore, come se quest'ultimo fosse un vettore *applicato* (vedi definizioni di `pC` e `pM1`). I nomi di variabili di tipo vettore saranno utilizzabili in successive operazioni racchiudendoli tra parentesi quadre (vedi `[vAB]`, `[vAC]` nella definizione di `vN_ABC`). Se in una data espressione una variabile vettore viene manipolata utilizzando le parentesi tonde anziché quadre, ad esempio `(vAB)`, Sketch segnalerà un errore. Analoga situazione si avrà per uno scorretto uso delle parentesi con le variabili punto.

Sia per le variabili di tipo punto che per quelle di tipo vettore esiste la possibilità di estrarne le componenti con gli operatori `'x`, `'y`, `'z` (vedi le definizioni di `pH`, `vNxy` e `pM0`).

Per quanto riguarda le trasformazioni, esse sono in generale costituite da rotazioni, traslazioni, scalature, e da loro combinazioni. Nel linguaggio di Sketch queste trasformazioni vengono definite con una sintassi molto intuitiva. Ad esempio, la definizione di `tRot` lascia intendere che si tratta di una rotazione – in notazione matematica:  $\mathcal{T}_{\text{rot},H,z}$  – di 180° intorno ad un asse passante per il punto `(pH)` e parallelo al versore dell'asse  $z$ , `[0,0,1]`. Un

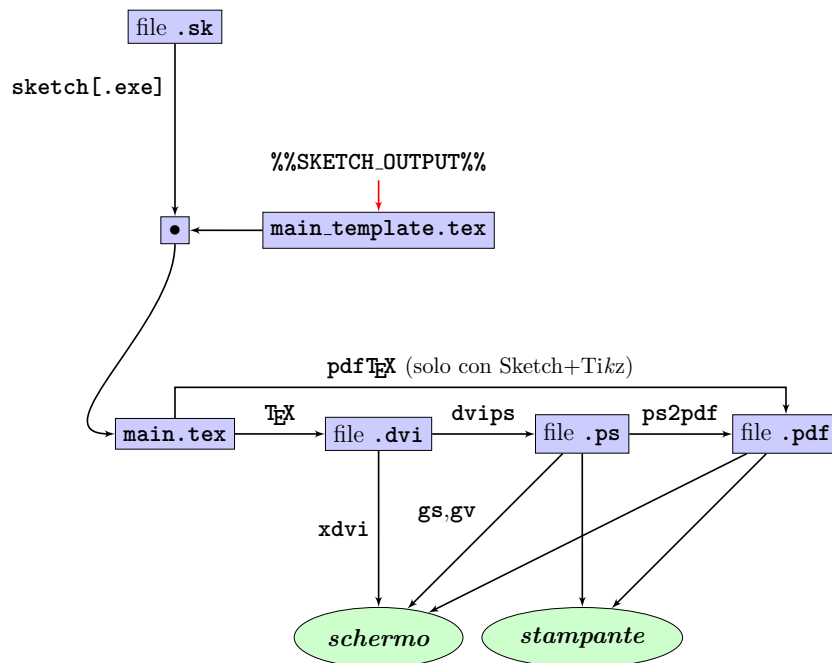


FIGURA 4: Il flusso di lavoro con Sketch e L<sup>A</sup>T<sub>E</sub>X.

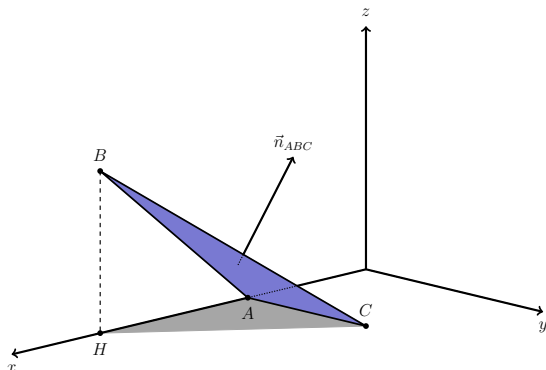


FIGURA 5: Una scena composta con Sketch.

punto  $P_2$  ottenuto dal punto  $P_1$  attraverso questa trasformazione sarà dato dall'istruzione:

```
def pP2 [[tRot]]*(pP1) % ←  $P_2 = T_{rot,H,z}(P_1)$ 
```

Come si vede da questo esempio, una volta definite, le trasformazioni esse vengono riutilizzate racchiudendone il nome tra doppie parentesi quadre: `[[tRot]]`. L'esempio mostra anche che la trasformazione di un punto si ottiene attraverso un'opportuna operazione di moltiplicazione. Ciò corrisponde al fatto che Sketch definisce una trasformazione come  $T_{rot,H,z}$  attraverso un operatore matriciale di uno spazio di *coordinate omogenee*.

Le coordinate omogenee di un punto  $P \equiv (x, y, z)$  dello spazio euclideo tridimensionale sono date da una qualsiasi quaterna  $(\xi, \eta, \zeta, w)$  di numeri reali tali che:  $w \neq 0$ ,  $x = \xi/w$ ,  $y = \eta/w$  e  $z = \zeta/w$ . Dunque, il punto espresso in coordinate  $(\xi, \eta, \zeta, w)$  nello spazio ampliato è equivalente al punto reale  $(\xi/w, \eta/w, \zeta/w)$ . I punti in coordinate omogenee con coordinata  $w$  nulla sono detti im-

propri, e non hanno nessun significato geometrico nello spazio cartesiano, ma possono rappresentare un punto all'infinito, nella direzione del vettore tridimensionale  $(x, y, z)$ . Le coordinate omogenee permettono dunque di rappresentare punti all'infinito, e consentono di esprimere *tutte* le trasformazioni di coordinate in forma matriciale. L'insieme costituito da tutte le quaterne non nulle forma uno spazio proiettivo tridimensionale.

Le coordinate omogenee sono particolarmente vantaggiose in computer grafica per il fatto non banale che qualunque trasformazione affine è rappresentabile con un prodotto tra matrici, ma lo è anche la stessa proiezione prospettica. L'uso di coordinate omogenee è particolarmente importante, perché implica che Sketch è in grado di operare trasformazioni per proiezioni in prospettiva.

Le rimanenti trasformazioni sono anch'esse definite intuitivamente, vedi ad esempio la definizione della traslazione `tTransl` di entità 2, parallelamente all'asse  $x$  – in notazione matematica, una funzione  $T_{trasl,x}$ ; o anche la scalatura, una riduzione dell'80%, della coordinata  $z$  definita come `tScaleZ`. Esiste inoltre la possibilità di definire delle trasformazioni composte, cioè di definire delle trasformazioni come sequenza di altre, come nel caso della trasformazione di roto-traslazione definita dalla variabile `tMyT`. La sequenza di trasformazioni viene ottenuta attraverso l'uso della parola chiave `then`.

Accanto agli oggetti definiti in una scena, Sketch offre la possibilità di inserire annotazioni costituite da vere e proprie finestre di comandi L<sup>A</sup>T<sub>E</sub>X. Ciò è ottenuto mediante l'uso del comando `special`. Per esempio, un utente di PStricks po-

trebbe pensare di utilizzare il comando `\pscircle` per ottenere un punto pieno di diametro `2pt` in corrispondenza delle posizioni dei punti  $A$ ,  $B$  e  $C$  nella scena e contemporaneamente posizionare le etichette  $A$ ,  $B$  e  $C$  con il comando `\uput`:

```
special |
\pscircle*[fillcolor=black]#1{2pt}
\uput{2pt}[-90]{0}#1{A}
\pscircle*[fillcolor=black]#2{2pt}
\uput{2pt}[ 0]{0}#2{B}
\pscircle*[fillcolor=black]#3{2pt}
\uput{2pt}[ 90]{0}#3{C}
|(pA)(pB)(pC)
```

Il comando `special` si comporta come una funzione che accetta un numero variabile di parametri di tipo punto. I parametri in questo caso sono i punti `(pA)`, `(pB)` e `(pC)`. Le loro coordinate tridimensionali riferite alla scena verranno trasformate da Sketch in coordinate piane secondo regole prospettiche e sostituite nel testo del comando `special` racchiuso dai delimitatori `|...|` al posto delle stringhe `#1`, `#2` e `#3`. Ciò consente di ottenere delle annotazioni e in generale degli effetti grafici di basso livello in posizioni precise di un disegno, corrispondenti a punti dello spazio a tre dimensioni. L'utente non è tenuto a conoscere la posizione finale sul piano del foglio delle variabili punto che intende passare al comando. Questa cambierà infatti se verrà variata la posizione del punto di osservazione o la direzione di osservazione della camera (vedi comando `view` nel listato del file `pyramid.sk` o più avanti).

Il comando `special` è un comando molto flessibile e permette di ottenere effetti grafici non banali, combinando la potenza degli ambienti e dei pacchetti LATEX e la possibilità di manipolare accuratamente punti della scena tridimensionale. Un esempio è costituito dal disegno della curva tridimensionale di Figura 13. Per via del fatto che la versione attuale di Sketch non permette di disegnare delle curve spaziali con un comando apposito, che accetti dei punti come argomenti, la traiettoria del velivolo è stata ottenuta in maniera artificiosa. Si è definito un insieme di punti lungo la traiettoria, nella scena tridimensionale. Tali punti sono stati poi passati come argomenti ad un comando `special` all'interno del quale si è provveduto, con comandi PSTricks, a disegnare una curva piana che unisce le proiezioni prospettiche nel piano del foglio dei punti della curva tridimensionale. Il risultato grafico, seppure approssimato, appare soddisfacente.

### 3.3 Gerarchie di oggetti e trasformazioni

Vediamo ora come si applica una trasformazione come `tMyT` agli oggetti della scena di Figura 5. Il risultato che vogliamo ottenere è un disegno come quello di Figura 6.

Per arrivare ad un uso elegante delle trasformazioni, vediamo come si può organizzare una gerarchia di oggetti grafici e di annotazioni nel disegno

di Figura 5. Introduciamo il concetto di *polygon*. Nel disegno di partenza compaiono due superfici piane triangolari, una di vertici  $(A, B, C)$ , l'altra di vertici  $(A, H, C)$ . In Sketch, con uscita Tikz, questi due triangoli sono definiti come segue:

```
def Triangle1 {
% solo contorni, no riempimento
def stylecontour
[lay=over,line width=1.4pt,fill=None]
polygon[stylecontour](pA)(pB)(pC)
% riempimento colorato
def stylefill
[style=solid,
color=blue!50!gray!70]
polygon[stylefill](pA)(pB)(pC)
% cerchi pieni per ogni punto
special |
\fill [style=solid] #1 circle (2pt);
\fill [style=solid] #2 circle (2pt);
\fill [style=solid] #3 circle (2pt);
|(pA)(pB)(pC)
}
def Triangle2 {
% riempimento, no contorni
def stylefill
[line width=0pt,style=solid,
color=gray!70]
polygon[stylefill](pA)(pH)(pC)
special |
\fill [style=solid] #1 circle (2pt);
|(pH)
% tratteggio BH
special |
\draw[-,dashed,line width=0.8pt]
#1 -- #2;
|(pB)(pH)
}
```

Le istruzioni precedenti definiscono i due oggetti grafici `Triangle1` e `Triangle2` contenenti il comando `polygon`. Quest'ultimo accetta delle opzioni tra parentesi quadre per il controllo del riempimento e dei contorni. Nel caso particolare le opzioni seguono la sintassi di Tikz, sono definite come stringhe di caratteri (vedi definizioni di `stylecontour` e `stylefill`) e passate infine al comando racchiuse da parentesi quadre (vedi ad esempio `polygon[stylefill](pA)(pB)(pC)`). Il comando `polygon` definisce in questo caso una superficie triangolare piana che unisce i tre punti passati come argomenti. Si rimanda al manuale di riferimento per approfondimenti sull'uso di questo comando.

I due triangoli rappresentano un esempio di oggetti "drawable", nel gergo di Sketch, cioè di oggetti *disegnabili*. Al contrario, gli oggetti punto come `(pA)`, `(pB)`, e così via, gli oggetti vettore e le trasformazioni definite sopra *non* sono entità disegnabili. Per il meccanismo di composizione della scena e di produzione del disegno finale sarà necessario che i due triangoli, dopo la loro definizione, siano *posti nella scena* con un'apposita sintassi. Il disegno di oggetti *drawable* sarà ottenuto racchiudendone il nome tra parentesi graffe. Ad esempio, l'istruzione:

```
{Triangle1} {Triangle2}
```

permette di ottenere il disegno dei triangoli per

valori di default della posizione del punto di vista e dell'orientamento della camera. L'utente potrà intervenire tanto sulla posizione dell'osservatore della scena, tanto sulla direzione di osservazione con una trasformazione apposita, `view`, in combinazione con il comando `put`. Si potrà avere ad esempio (vedi anche il listato di `pyramid.sk`):

```
def pEye (2,2,0.7) % punto di vista
def pLookAt (0,0,0) % target camera
% disegno
put { view( (pEye), (pLookAt), [0,0,1] ) } {
  {Triangle1} {Triangle2}
}
```

Il comando `put` serve a produrre il disegno di oggetti *drawable* dopo avervi applicato una o più trasformazioni. Nell'esempio la trasformazione utilizzata è `view`, che accetta tre argomenti: un punto, (`pEye`), che identifica la posizione dell'obiettivo della camera, un secondo punto, (`pLookAt`), che identifica la posizione nella scena *puntata* dalla camera, ed un vettore, nell'esempio `[0,0,1]`, che indica la direzione verticale (si consulti il manuale di Sketch per i dettagli). Due possibili alternative di uso di `put` in questo contesto sono date dagli esempi seguenti:

```
put { view( (pEye), (pLookAt), [0,0,1] )
  then [[ScaleZ]] } {
  {Triangle1} {Triangle2}
}
```

oppure

```
put { view( (pEye), (pLookAt), [0,0,1] ) {
  put { translate([0,0,-3]) } {
    {Triangle1}
  }
  {Triangle2}
}
```

in cui si vede che `put` accetta anche sequenze di trasformazioni e che comandi `put` possono essere innestati l'uno nell'altro per un maggiore controllo del posizionamento degli oggetti nella scena.

Sketch permette di costruire degli oggetti *drawable* come delle gerarchie, per composizione di altri oggetti *drawable*. Se si intende disegnare sempre insieme i due triangoli `Triangle1` e `Triangle2` basta definire un nuovo oggetto come segue:

```
def Triangles {
  {Triangle1} {Triangle2}
  {VecNormal2Triangle1} % vettore normale
}
```

È interessante vedere a questo punto quali sono gli effetti grafici di una trasformazione come `tMyT` definita in precedenza. Il risultato della roto-traslazione applicata ai triangoli ed al vettore definiti sopra come unico oggetto *drawable* `Triangles` è mostrato in Figura 6.

Il disegno della nuova coppia di triangoli si otterrà semplicemente con l'istruzione seguente:

```
% nuovo drawable
def NewTriangles [[tMyT]]*{Triangles}
% disegno
put { view( (pEye), (pLookAt), [0,0,1] ) {
  {NewTriangles}
}
```

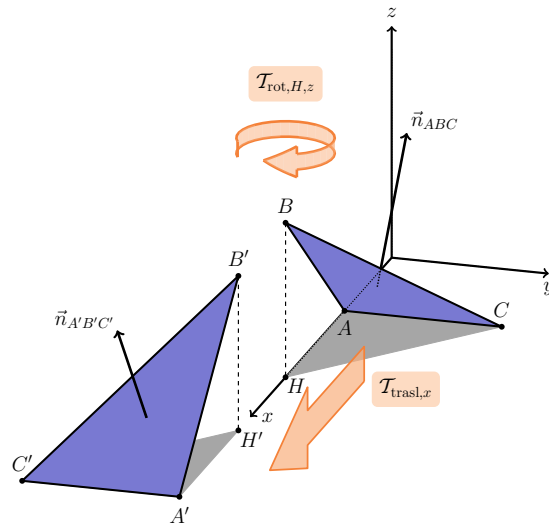


FIGURA 6: Una trasformazione che ruota e trasla i triangoli di Figura 5. Il punto di vista è stato cambiato (vedi sopra).

Un disegno completo come quello di Figura 6 sarà prodotto dalle istruzioni:

```
% Scena: collezione di oggetti drawable
def Scene {
  {Triangles} {Labels}
  {NewTriangles} {LabelsNew}
}
% disegno della scena
put { view( (pEye), (pLookAt), [0,0,1] ) }
  {Scene}
% produci output per Tikz
global { language tikz }
```

### 3.4 Operazioni di disegno cicliche

In Sketch esiste la possibilità di effettuare delle operazioni di disegno ripetute. Le funzionalità di disegno cicliche di Sketch, sebbene attualmente limitate, si presentano comunque come degli strumenti utili e snelli per particolari esigenze di disegno. Il concetto che ne sta alla base è quello di poter disegnare un certo numero di istanze di un oggetto *drawable*, ciascuna delle quali è ottenuta dalla precedente attraverso una trasformazione prefissata. Il comando corrispondente è dato dalla parola chiave `repeat` e si rimanda il lettore al manuale di riferimento per un approfondimento dei dettagli.

Un comando simile a `repeat`, ma che permette di creare curve e superfici a partire da oggetti trasformati ripetutamente nello spazio, è dato dal comando `sweep`. La Figura 7 presenta il disegno di una curva dello spazio detta *elica*. Essa si può ottenere componendo un moto di rotazione a velocità angolare costante del punto  $H_0$  intorno all'asse  $z$  con un moto di traslazione a velocità costante parallelo all'asse  $z$ . Il risultato è una curva la cui tangente ha una pendenza  $\theta$  costante rispetto al piano  $xy$ . La tangente della pendenza è detta *passo* e viene tipicamente indicata con  $p = \tan \theta$ .

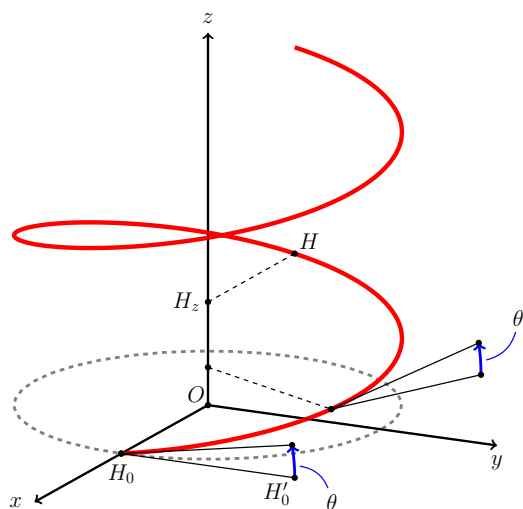


FIGURA 7: Un'elica di asse  $z$  e passo  $p = \tan \theta$ .

La curva di Figura 7 si otterrà con i comandi seguenti:

```
def p0 (0,0,0) def p0 (3,0,0)
def vK [0,0,1]
def Helix {
  % parametri
  def nsteps 180
  def angle_step (360+180)/nsteps
  def pitch_angle 10
  def radius |(p0)-(p0)|
  def pitch radius * (angle_step/57.3)
  * sin(pitch_angle)/cos(pitch_angle)
  def tHelix rotate(angle_step,(0,0,0),[vK])
  then translate(pitch*[vK])
  % elica
  sweep[line width=2.8pt, color=red] {
    nsteps, [[tHelix]]
  } (p0)
  % punto pieno in H0
  special |
  \fill [style=solid] #1 circle (2pt);
  |(p0)
  % cerchio di base, z=0
  def BaseCircle {
    sweep[line width=1.8pt, color=gray,
      style=dashed] {
      36,
      rotate(360/36, (0,0,0), [vK])
    } (p0)
  }
  % commenta per non disegnare il cerchio
  {BaseCircle}
}
put { view((pEye), (pLookAt), [0,0,1]) } {
  {Helix}
}
```

L'istruzione `sweep` viene utilizzata sia per disegnare l'elica che per disegnare la circonferenza tratteggiata sul piano  $z = 0$ . Per usare questo comando bisogna specificare: (i) il numero di cicli di disegno (vedi `nsteps`), (ii) la trasformazione da applicare all'istanza precedente per ottenerne una corrente (vedi `[[tHelix]]`) e (iii) un oggetto *generatore*, nel nostro esempio un punto (vedi `(p0)`) nella definizione dell'elica e di `BaseCircle`).

Si osservi come il comando `sweep` sia stato utilizzato nel listato di `pyramid.sk` specificando una linea (vedi `line(p0) (p1)`) come oggetto generatore

ed ottenendo come risultato la superficie laterale della piramide di Figura 3.

Un'ulteriore esempio di effetto grafico ottenuto per mezzo di un ciclo è costituito dalle annotazioni di Figura 7 che mostrano l'entità dell'angolo  $\theta$ . Un'istruzione di `sweep` viene qui utilizzata per disegnare gli archi di circonferenza che terminano con una freccia. Se si considera, ad esempio, il punto  $H_0$ , per applicare correttamente il comando `sweep` va definito un punto  $H'_0$  lungo la retta orizzontale uscente da  $H_0$ . Il punto  $H'_0$  diventa il punto generatore dello *sweep* e viene trasformato nel punto  $H''_0$ , appartenente alla tangente all'elica passante per  $H_0$ . In linguaggio Sketch si scriverà:

```
% vettore lungo il raggio
def vR0 unit( (pH0)-(p0) )
% vettore orizzontale
def vH0 unit( [vR0]*[vK] )
% punto H'_0
def pH01 (pH0)+3*[vH0]
def tH02 rotate(pitch_angle,(pH01),[vR0])
% punto H''_0 lungo la tangente all'elica
def pH02 [[tH02]]*(pH01)
% arco
sweep[line width=1.8pt, color=blue,
  style=solid,arrows=->]{
  4, rotate(pitch_angle/4,(pH0),[vR0])
}(pH01)
% linee e punti con Tikz
special|
\draw[-,solid,line width=0.8pt]#1 -- #2;
\fill [style=solid] #1 circle (2pt);
\fill [style=solid] #2 circle (2pt);
\fill [style=solid] #3 circle (2pt);
\draw[-,solid,line width=0.8pt]#1 -- #3;
|(pH0)(pH01)(pH02)
% simbolo theta con Tikz
special|
\path[line width=0pt]#1
  to node [node distance=0pt,pos=0.4,
    pin={[pin distance=24pt,
      pin edge={bend left,blue,thick}]
      -45:\theta$}]#2;
|(pH01)(pH02)
```

Per un punto  $H$  qualsiasi lungo l'elica, l'effetto voluto viene realizzato potendo comodamente ottenere in Sketch un vettore dal prodotto vettoriale di altri due. Si costruiranno: un punto  $H'$  staccandolo lungo una retta orizzontale uscente da  $H$  e normale al raggio locale, ed un punto  $H_z$  proiettando  $H$  sull'asse. Osservando che i due vettori  $H-H_z$  ed  $(H'-H) \times \vec{k}$  sono paralleli, si potrà definire un vettore da utilizzare nell'istruzione `sweep` come asse di rotazione. Tale operazione avrà come centro il punto  $H$  e come punto generatore il punto  $H'$ . L'effetto grafico voluto è mostrato in Figura 7.

## 4 Uso avanzato di Sketch

Le caratteristiche del linguaggio di composizione di una scena messe a disposizione da Sketch sono tali da poter dire che le illustrazioni tridimensionali con esso ottenibili sono di fatto delle illustrazioni avanzate.

La possibilità di costruire delle superfici dello spazio come reticolati, cioè come insiemi di tasselli

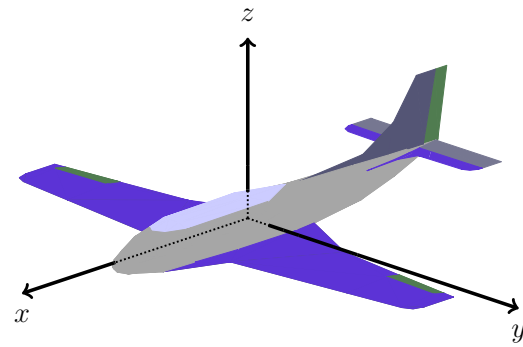
triangolari, attraverso il comando `polygon`, l'algoritmo di *hidden surface removal* (anche noto come algoritmo "del pittore") e la possibilità di definire trasformazioni di coordinate aprono la strada ad un uso ancora più spinto di Sketch.

Un utente potrebbe pensare, ad esempio, di costruire una collezione di modelli 3D e di utilizzarli a proprio uso e consumo. Sarà possibile così comporre delle scene avanzate inserendovi uno o più oggetti opportunamente disposti e corredati di specifiche annotazioni, come avviene in molte illustrazioni tecniche. Alla luce di ciò, si pensi ai popolari formati grafici STL o OBJ, che descrivono superfici 3D come tassellazioni, e a tutti quei modelli tridimensionali liberamente scaricabili da internet o creabili con programmi *open source* come Blender. Con queste idee in mente l'autore ha creato col tempo, a partire da modelli in formato STL, una libreria personale di modelli di velivolo in formato Sketch. Per fare ciò è bastato convertire le tassellazioni STL in istruzioni `polygon` mediante l'uso di un'apposito programma (DE MARCO, 2007) basato sul software *open source* IVCon-TL di John Burkardt (BURKARDT, 2007).

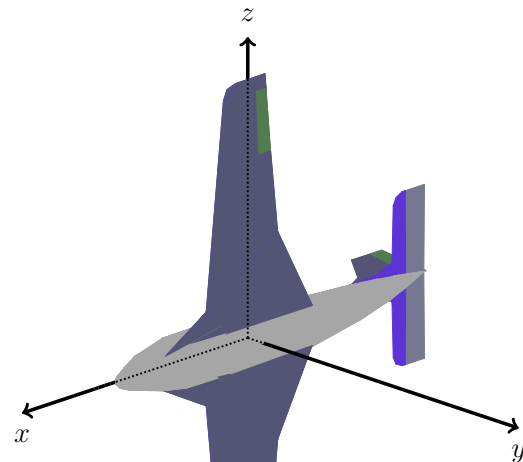
A seconda dell'esigenza specifica, un dato modello di velivolo può essere opportunamente composto e posizionato nella scena, una o più volte. Si veda a tal proposito la Figura 8.

Il semplice modello di aereo di Figura 8 è stato concepito per scopi didattici. Si distingue la diversa colorazione di alcune parti architettoniche: la fusoliera, il tettuccio, le ali (dorso e ventre), la deriva verticale, gli alettoni, la parte fissa e la parte mobile (equilibratore) del piano di coda orizzontale, il timone. Ciascuno di questi sotto-componenti risiede in un file separato, contenente gli appositi comandi `polygon` e viene richiamato all'occorrenza con il comando `input`. Ad esempio:

```
% polygon files
def Aircraft_Fuselage {
  def bodystyle [line width=Opt,
    style=solid,color=gray!70]
  input{small_aircraft_fus.sk}
}
def Aircraft_Canopy {
  def bodystyle [line width=Opt,
    style=solid,color=mylightblue!100!gray]
  input{small_aircraft_canopy.sk}
}
def Aircraft_Fin {
  def bodystyle [line width=Opt,
    style=solid,color=mydarkblue!40!gray]
  input{small_aircraft_fin.sk}
}
% ...
% The Aircraft
def Aircraft {
  % ...
  {Aircraft_Fuselage} {Aircraft_Canopy}
  {Aircraft_Fin} {Aircraft_HTail}
  {Aircraft_WingsTop} {Aircraft_WingsBottom}
  %-----
  % Scommentare ed editare come si conviene
  % se si vogliono ruotare questi oggetti
  %-----
  % put{
```



(a)



(b)

FIGURA 8: Un semplice modello tridimensionale di velivolo con Sketch.

```
% rotate( 0, (pElevatorHingeB),
% [vElevatorAxis] ) }
{Aircraft_Elevator}
% put{
% rotate( -30, (pRudderHingeA),
% [vRudderAxis] ) }{Rudder}
{Aircraft_Rudder}
% put{ rotate( -30, (pAileronHingeRA),
% [vAileronAxisR] ) }{AileronR}
{Aircraft_AileronR}
% put{ rotate( -30, (pAileronHingeLA),
% [vAileronAxisL] ) }{AileronL}
{Aircraft_AileronL}
}
```

Le definizioni riportate sopra si fondano sull'istruzione `input` e sulla convenzione che ciascun file contenente la descrizione di una superficie presenti i comandi `polygon` con un campo delle opzioni fittizio del tipo `[bodystyle]`. Un esempio di file `small_aircraft_elevator.sk` è il seguente:

```
%-----
% SKETCH file: small_aircraft_fus.sk
% Original data: small_aircraft_fus.stl
polygon[bodystyle]
( 9.050886, -624.006042, 41.119484 )
( -0.105902, -693.260681, 43.389565 )
( 240.739075, -695.190979, 43.000031 )
```

```

polygon[bodystyle]
( 9.050886, -624.006042, 41.119484 )
( 240.739075, -695.190979, 43.000031 )
( 240.739075, -624.028137, 43.000027 )
% ...
%-----EOF

```

Il comando `input`, come in altri linguaggi, permette di includere comandi contenuti in un file separato. Nell'esempio si è usata la possibilità di definire in Sketch delle variabili stringa di caratteri. La stringa `bodystyle` viene definita localmente a ciascun oggetto (vedi ad esempio `Aircraft_Fuselage`) prima di richiamare i comandi del tipo

```

polygon[bodystyle]( <p1> ), ( <p2> ), ( <p3> )

```

contenuti nel file esterno (vedi ad esempio il frammento di file `small_aircraft_fus.sk` riportato sopra). La variabile `bodystyle` ha *campo di visibilità locale*, cioè non risultra definita al di fuori della definizione di un oggetto come `Aircraft_Fuselage`. Ciò permette di ridefinire tale variabile più volte, una per ciascun oggetto che andrà a comporre l'oggetto *drawable* gerarchicamente più importante: `Aircraft`. Sarà quest'ultimo ad essere disegnato nella scena con un opportuno comando `put`. Ad esempio, il disegno del velivolo di Figura 8(b) si otterrà con comandi del tipo:

```

put { view((pEye), (pLookAt), [0,0,1]) } {
  put { rotate(90,(0,0,0),[1,0,0]) }{
    {Aircraft}
  }
}

```

## 5 Piccola galleria di illustrazioni

Per terminare con degli esempi, si presenta una piccola galleria di illustrazioni realizzate con Sketch e sfruttando le caratteristiche sia di Tikz che di PSTricks.

L'illustrazione di Figura 10 è stata creata per mostrare le azioni esterne su un velivolo durante una virata. Si noti l'uso di trasparenze per mezzo di comandi Tikz e l'uso di frecce tridimensionali sia dritte che curve. Queste ultime sono state definite analiticamente in Sketch e vengono utilizzate con opportune impostazioni di colore e di fattore di scala. Nell'esempio specifico le frecce aiutano a capire che la portanza  $L$  e la resistenza  $D$  aerodinamiche del velivolo sono disposte in un piano inclinato di  $\phi_W$  sulla verticale locale.

L'illustrazione di Figura 11 è stata creata per mostrare un caso particolare di evoluzione, l'evoluzione di *pull-up* sostenuta fino a compiere un cosiddetto *loop*, durante la quale le storie temporali degli angoli di Eulero del velivolo presentano delle discontinuità. Le annotazioni e le colorazioni sono state ottenute con PSTricks.

I disegni di Figura 12 sono stati anch'essi ottenuti come quello precedente e rappresentano classiche illustrazioni in cui l'orientamento del velivolo

nello spazio, definito dagli angoli di Eulero, è visualizzato per mezzo di una sospensione cardanica. Gli oggetti grafici sono stati modellati con un software CAD, esportati in formato STL e convertiti in poligoni in formato Sketch.

L'illustrazione di Figura 13 mostra un secondo esempio di evoluzione di volo. Come per la Figura 11, il modello 3D del velivolo è quello presentato in Figura 8. In questo esempio uno solo degli angoli di Eulero, l'angolo detto di rollio  $\phi$ , presenta una discontinuità nel tempo.

## 6 Vantaggi e svantaggi nell'uso di Sketch

### 6.1 Osservazioni

L'uso di Sketch presenta vantaggi e svantaggi. Questi saranno più o meno sentiti a seconda del tipo di utente, del suo *background* culturale, della sua esperienza. Ad esempio alcuni potrebbero muovere la seguente osservazione: *Quale vantaggio comporta l'uso di Sketch rispetto all'ovvia alternativa di poter realizzare con uno strumento di modellazione tridimensionale delle viste prospettiche in formato bitmap o vettoriale? Queste immagini potrebbero essere importate ed annotate con codice LATEX extra che consenta di collocare le scritte nei punti giusti* (vedi Figura 9). *Oppure, se le immagini sono vettoriali, si potrebbero inglobare in esse dei frammenti di codice ad hoc da poter manipolare con LATEX in fase di produzione del documento finale.*

Una simile osservazione vale per chi propone l'uso di PSTricks e Tikz: *Perché ci si dovrebbe ostinare ad usare simili pacchetti quando esistono applicazioni commerciali e non<sup>2</sup> con le quali poter produrre illustrazioni anche molto complesse?* È una questione di "look and feel", direbbero gli americani. Eppure, il sito web di PSTricks è uno dei più visitati e dei più attivi; lo si vede dall'evoluzione che hanno avuto negli ultimi due anni le sezioni dedicate agli esempi ed ai pacchetti collegati.

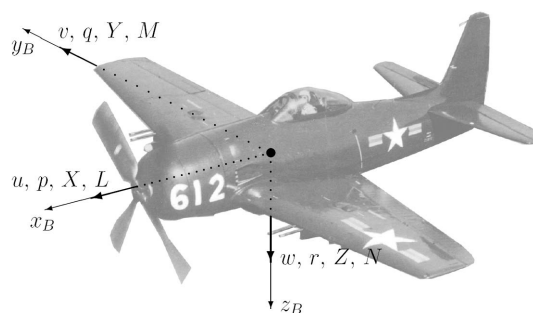


FIGURA 9: Esempio di immagine bitmap con annotazioni apposte nei punti giusti.

2. Ad esempio, Macromedia FreeHand, Adobe Illustrator, Corel Draw, Inkscape, XFig, WinFig.

Una risposta ragionevole alle osservazioni precedenti è che ci sono diverse tipologie di utenti – a volte completamente opposte – e di volta in volta il metodo di lavoro più appropriato dipende anche dal carattere e dalla complessità dell’illustrazione che si intende realizzare. Non trascurabile altresì è lo scopo al quale l’illustrazione è destinata: una pubblicazione, una monografia, una dispensa didattica informale, una presentazione, un rapporto tecnico.

Per valutare le possibili situazioni di vantaggio derivanti dall’uso di Sketch va tenuto presente che tale approccio è destinato ad utilizzatori di L<sup>A</sup>T<sub>E</sub>X/PSTricks/Tikz, cioè ad utenti che hanno sposato ed in alcuni casi si sono piegati ad un metodo di lavoro non-WYSIWYG. In ogni caso la produzione di un’illustrazione di buona qualità si concluderà al termine di un processo iterativo durante il quale saranno probabilmente necessarie diverse ricompilazioni prima di raggiungere un risultato esteticamente soddisfacente.

La via delle annotazioni ad un’immagine generata con strumenti esterni richiederà l’importazione dell’immagine bitmap, ad esempio nell’ambiente `pspicture` di PSTricks. Questo metodo è di pratico impiego e può avvalersi dell’uso di una griglia di riferimento temporanea opportunamente sovrapposta all’immagine originale al fine di posizionare le annotazioni nei punti appropriati. Tale metodo è da preferirsi certamente se il numero delle annotazioni è contenuto e se esse si limitano a semplici simboli o paragrafi. Esso diventa laborioso se l’illustrazione che si intende realizzare dovesse contenere molti simboli e l’indicazione di angoli, assi e rette di riferimento o di punti particolari in relazione agli oggetti presenti nella scena (ad esempio il baricentro di un velivolo oppure gli assi di cerniera delle superfici aerodinamiche di governo). In tal caso l’esperienza insegna che la realizzazione di annotazioni grafiche ben proporzionate e visivamente corrette non è immediata e richiede quasi certamente più iterazioni di *editing*-stampa-ispezione.

Un discorso analogo vale per la manipolazione di viste esportate in formato vettoriale, con annotazioni inglobate e manipolabili *a posteriori* secondo un dato protocollo. Anche in questo caso sono inevitabili alcune iterazioni prima di arrivare ad un risultato finale soddisfacente: almeno le dimensioni e le posizioni effettive delle annotazioni vanno controllate a valle della produzione finale del documento.

Si tenga presente inoltre che spesso si dà per scontato l’uso di strumenti di modellazione tridimensionale. Tuttavia, se si vogliono valutare correttamente i vantaggi di un dato metodo di lavoro per la produzione di illustrazioni, vanno certamente considerati i costi ed i tempi di apprendimento del software CAD che si intende usare. I costi nel caso di CAD commerciali sono alti. I

software di modellazione 3D non commerciali più usati sono Blender e K-3D<sup>3</sup>: sebbene ancora nella fase *pre-release*, essi sono dotati di caratteristiche funzionali paragonabili a quelle di costose applicazioni concorrenti. Tutti i CAD, commerciali e non, hanno comunque delle interfacce utente ricche di funzionalità ed a volte estremamente complesse. Quelle di Blender e K-3D sono poi note per essere non del tutto convenzionali ed orientate ad utenti programmatori o esperti. Dunque, in ogni caso, i tempi di apprendimento di strumenti di modellazione e manipolazione di scene tridimensionali sono senz’altro non brevi.

## 6.2 Aspetti del metodo di lavoro proposto

L’uso di Sketch al contrario può essere appreso in tempi sorprendentemente brevi. Un utente mediamente esperto, in possesso di nozioni di Geometria Analitica, grazie soprattutto all’intuitività della sintassi ed agli ottimi esempi guidati forniti nel manuale d’uso, arriva ad apprendere il linguaggio di Sketch ed a realizzare una prima illustrazione di media complessità nel giro di 2 ore di lavoro. Uno studente universitario, già in grado di padroneggiare l’uso di L<sup>A</sup>T<sub>E</sub>X e dei programmi correlati, dovrebbe poter raggiungere lo stesso risultato in mezza giornata di lavoro.

Le figure 5, 6 e 7, concepite appositamente per questo articolo, hanno richiesto all’autore un’ora di lavoro complessivo. Le due illustrazioni di Figura 8, grazie alla piccola libreria messa a disposizione da DE MARCO (2007), hanno richiesto 10 minuti di lavoro. A giudizio dell’autore il tempo massimo impiegato per la preparazione di un’illustrazione o di un grafico di buona qualità da inserire in un lavoro scientifico è accettabile se va dai 30 ai 60 minuti. Se l’illustrazione deve essere inserita in un libro i tempi accettabili sono di un ordine di grandezza superiore. Pertanto i tempi di lavoro richiesti da Sketch sono molto bassi per illustrazioni semplici, e mediamente bassi per illustrazioni di complessità superiore. Tuttavia se l’illustrazione deve essere particolarmente ricca di oggetti e di annotazioni o di effetti di *shading*, è preferibile scegliere un metodo alternativo o affidarsi a professionisti.

Per quanto riguarda la sostenibilità del ciclo di lavoro con Sketch e L<sup>A</sup>T<sub>E</sub>X, sia gli utenti che fanno uso di IDE (*Integrated Development Environment*) dedicati, come WinEdt<sup>4</sup> o TexnicCenter<sup>5</sup>, sia quelli che preferiscono usare *editor* generici e gestiscono le compilazioni con un Makefile non dovrebbero apportare sostanziali sconvolgimenti al loro paradigma di lavoro se non aggiungere un passo di compilazione in più (vedi Figura 4).

La fase fondamentale del lavoro con Sketch è costituita dall’*editing* di codice sorgente nel suo

3. <http://www.k-3d.org>

4. <http://www.winedt.com>

5. <http://www.texniccenter.org>

TABELLA 1: Alcuni dati sui tempi di compilazione. Si considerano due tipologie di illustrazione, una semplice ed una alquanto complessa. Risultati ottenuti con un PC dotato di processore Pentium 4 a 3.2 GHz, 2 Gb di memoria RAM, Windows XP, MikTeX 2.6.

|                        | Figura 7  |         |        |
|------------------------|-----------|---------|--------|
|                        | sk→tex    | tex→dvi | dvi→ps |
| tempi di calcolo (s)   | 0.015     | 2.23    | 0.31   |
| dimensioni output (Mb) | 0.007     | 0.025   | 0.050  |
|                        | Figura 13 |         |        |
|                        | sk→tex    | tex→dvi | dvi→ps |
| tempi di calcolo (s)   | 0.38      | 8.31    | 4.56   |
| dimensioni output (Mb) | 0.670     | 7.88    | 8.189  |

linguaggio di modellazione. In questa fase avere accesso ad un *editor* con evidenziazione della sintassi personalizzabile diventa di fondamentale importanza. L'autore ha creato e messo a disposizione il file `sketch.vim` nell'archivio `skthings.zip` (DE MARCO, 2007). Questo *add-on* permette agli utilizzatori di *Vim*<sup>6</sup> di scrivere sorgenti in linguaggio Sketch usufruendo del *syntax highlighting* sia per i costrutti specifici di Sketch che per quelli in linguaggio L<sup>A</sup>T<sub>E</sub>X inclusi nei frammenti `special`.

Per quanto riguarda i tempi di compilazione, questi dipendono dalla complessità della scena e dal numero di elementi che compongono ciascun oggetto. La compilazione con Sketch è tipicamente molto veloce, anche nei casi in cui si richiede di rimuovere le facce nascoste nella rappresentazione di oggetti molto complessi. Meno veloce è la compilazione da parte di L<sup>A</sup>T<sub>E</sub>X dell'output di Sketch. Si riportano in Tabella 1 i tempi macchina richiesti per produrre due illustrazioni di diversa complessità.

In alcuni casi, quando il carico computazionale è appesantito dalla presenza nella scena di uno o più oggetti di grandi dimensioni è necessario accrescere la dimensione del *buffer* di memoria principale di L<sup>A</sup>T<sub>E</sub>X rispetto al valore di default. Per esempio, l'illustrazione di Figura 13 è stata ottenuta passando alla riga di comando di L<sup>A</sup>T<sub>E</sub>X l'opzione `-mem-max=9000000`.

Nel manuale utente (RESSLER, 2007b) si riporta il risultato di un test estremo a cui Sketch è stato sottoposto al fine di verificarne la robustezza e la velocità di esecuzione. Ne risulta che la compilazione del cosiddetto *Stanford Bunny*<sup>7</sup>, un modello tipicamente utilizzato per la verifica di algoritmi di computer grafica, costituito da circa 70000 triangoli, ha richiesto circa 6 secondi<sup>8</sup>. Una larga parte del tempo di esecuzione è stata richiesta

6. <http://www.vim.org>

7. <http://www.cc.gatech.edu/~turk/bunny/bunny.html>

8. Non è specificato il tipo di computer utilizzato per la prova né il sistema operativo.

per la scrittura in output del codice L<sup>A</sup>T<sub>E</sub>X mentre la rimanente è stata consumata dall'algoritmo di rimozione degli oggetti in secondo piano.

## 7 Conclusioni

In questo articolo si è presentato un possibile approccio alla produzione di illustrazioni di elevata qualità a scopo didattico, contenenti delle rappresentazioni prospettiche di scene tridimensionali. Uno degli scopi dell'articolo è quello di introdurre il programma Sketch, uno strumento relativamente recente ed ancora poco conosciuto nella comunità di utilizzatori L<sup>A</sup>T<sub>E</sub>X.

Mediante alcuni esempi guidati sono state presentate le caratteristiche principali del linguaggio di modellazione di Sketch nonché le sue potenzialità per la rappresentazione di scene complesse.

Infine sono stati discussi alcuni aspetti salienti del metodo di lavoro proposto: valutazione approssimativa dei tempi di apprendimento, dimensioni gestibili, tempi di calcolo.

## Riferimenti bibliografici

- BURKARDT, J. (2007). IVCon 3D Graphics File Converter. URL <http://ivcon-tl.sourceforge.net>.
- CALCARA, M. (1988). *Elementi di Dinamica del Velivolo, Vol. I*. Edizioni CUEN, Napoli.
- CASAROSA, C. (2004). *Meccanica del Volo*. Edizioni Plus, Pisa.
- DE MARCO, A. (2007). Modelli di velivolo in Sketch ed esempi di script. URL <http://www.dpa.unina.it/demarco/skthings.zip>.
- ETKIN, B. (1982). *Dynamics of Flight, Stability and Control*. John Wiley & Sons.

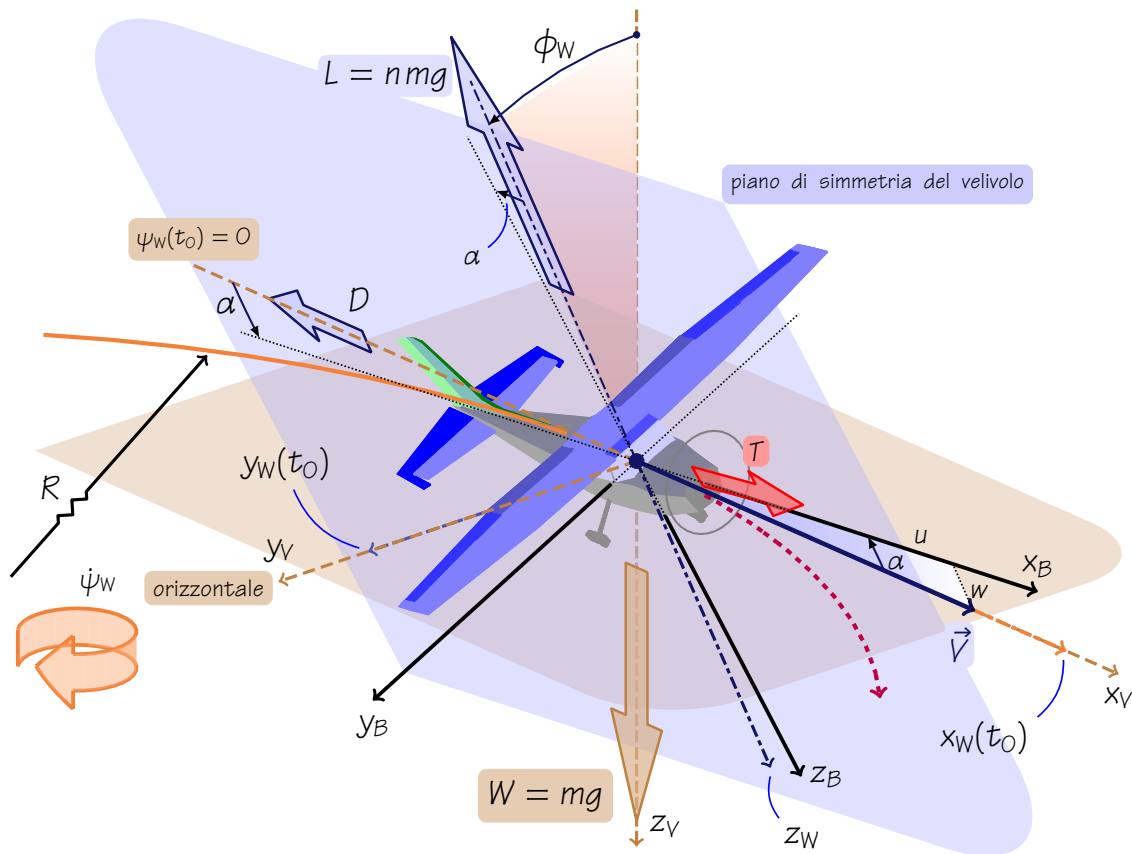


FIGURA 10: Definizione delle azioni esterne su di un velivolo in un'evoluzione di *virata corretta* a quota costante. Si evidenziano alcuni dei diversi sistemi di riferimento standard: il riferimento *verticale locale* (V, *local vertical*), quello degli *assi velivolo* (B, *body*) e quello degli *assi vento* (W, *wind*).

FAUSKE, K. M. (2007a). A PGF and TikZ examples gallery. URL <http://www.fauskes.net/nb/pgftikzexamples/>.

— (2007b). An introduction to Sketch 3D for PGF and TikZ users. URL <http://www.fauskes.net/nb/introduction-to-sketch/>.

— (2007c). Three dimensional graphics, illustrations and animations. URL <http://www.fauskes.net/nb/threedill/>.

MENGALI, G. (2001). *Elementi di Dinamica del Volo con Matlab*. Edizioni ETS, Pisa.

RESSLER, G. (2007a). Sketch website: Free Graphics Software for the TeX, LaTeX, and PSTricks Community. URL <http://www.frontiernet.net/~eugene.ressler>.

— (2007b). *Sketch. Simple 3D sketching*. URL <http://www.frontiernet.net/~eugene.ressler/manual.pdf>.

ROSENDAAL, T. (2007). Blender website. URL <http://www.blender.org>.

SCHMIDT, L. V. (1998). *Introduction to Aircraft Flight Dynamics*. AIAA Education Series.

TANTAU, T. (2006a). PGF/Tikz website – Graphic systems for TeX. URL <http://sourceforge.net/projects/pgf>.

— (2006b). *The TikZ and PGF Packages*. URL <http://sourceforge.net/projects/pgf>.

TUG BOAT (2007). Pstricks website. URL <http://tug.org/PSTricks/>.

VAN ZANDT, T. (2003). *PSTricks. PostScript macros for Generic TeX*. URL <http://www.ctan.org/tex-archive/graphics/pstricks/base/doc/pstricks-doc.pdf>. Version 97.

▷ A. De Marco  
 Università degli Studi di Napoli  
 “Federico II”  
 Dipartimento di Ing. Aerospaziale  
 agostino.demarco@unina.it

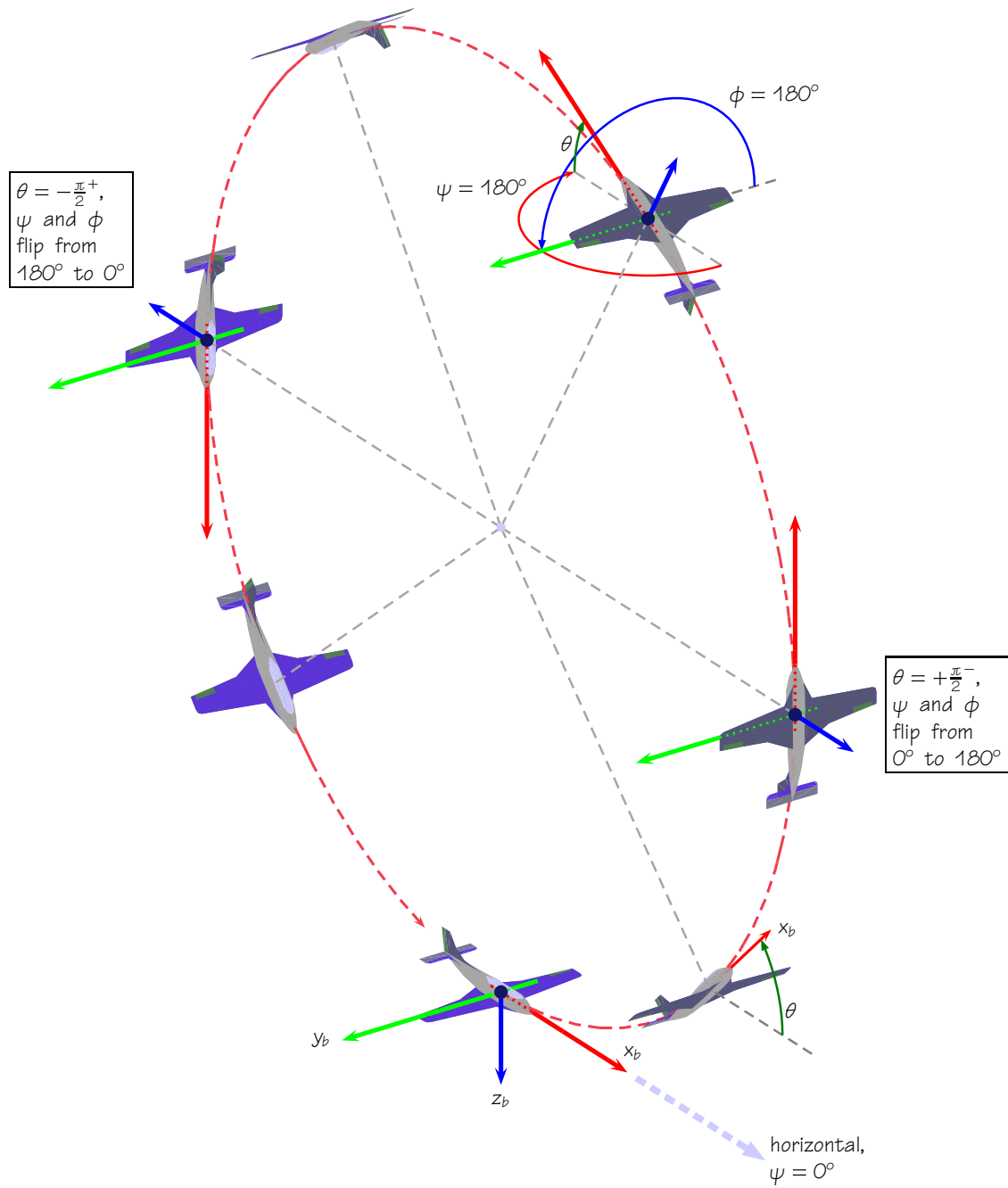
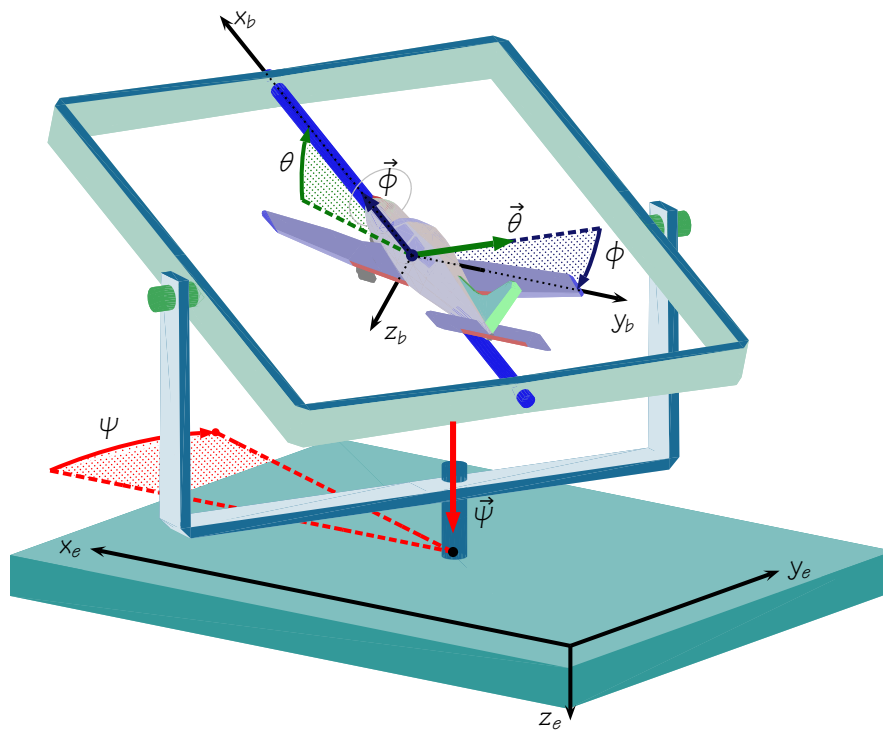
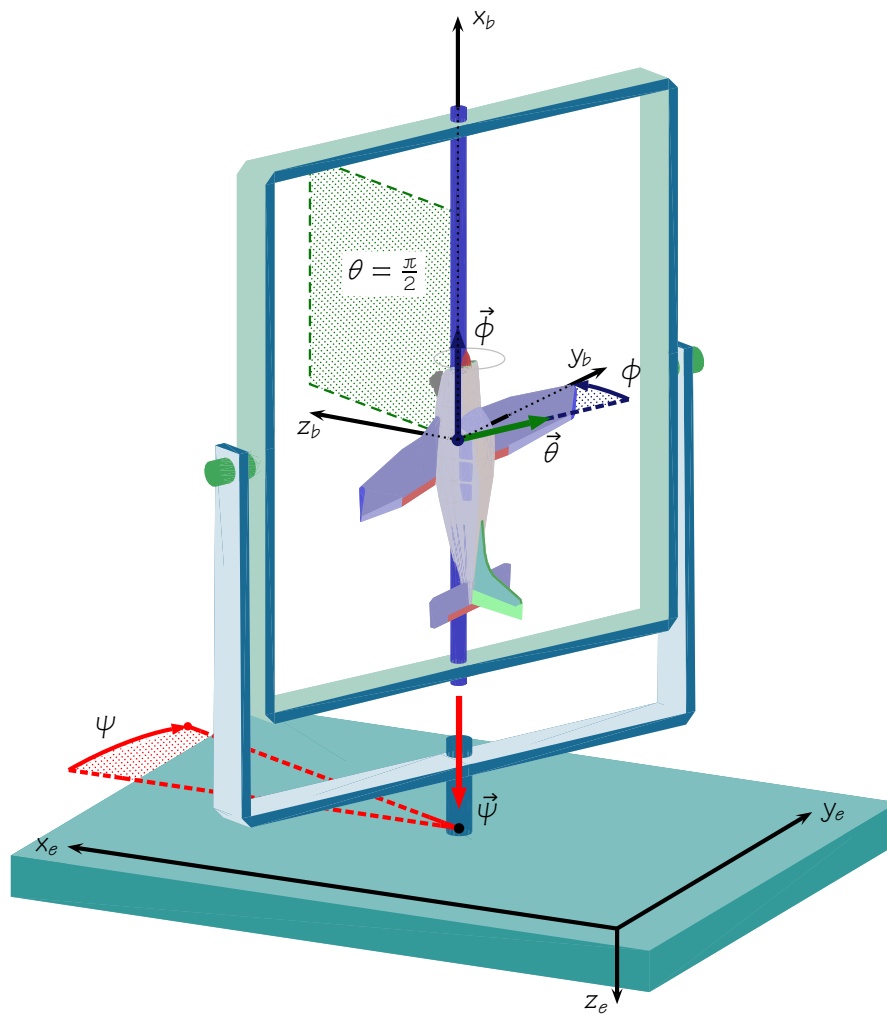


FIGURA 11: Un'illustrazione che mostra l'evoluzione di un velivolo detta *pull-up*. In un *pull-up* le storie temporali degli angoli di Eulero non sono continue.



(a) un orientamento generico



(b) l'assetto noto come *gimbal lock* in cui gli angoli di Eulero non sono definiti

FIGURA 12: Una classica illustrazione in cui l'orientamento del velivolo nello spazio, definito dagli angoli di Eulero, è visualizzato per mezzo di una sospensione cardanica.

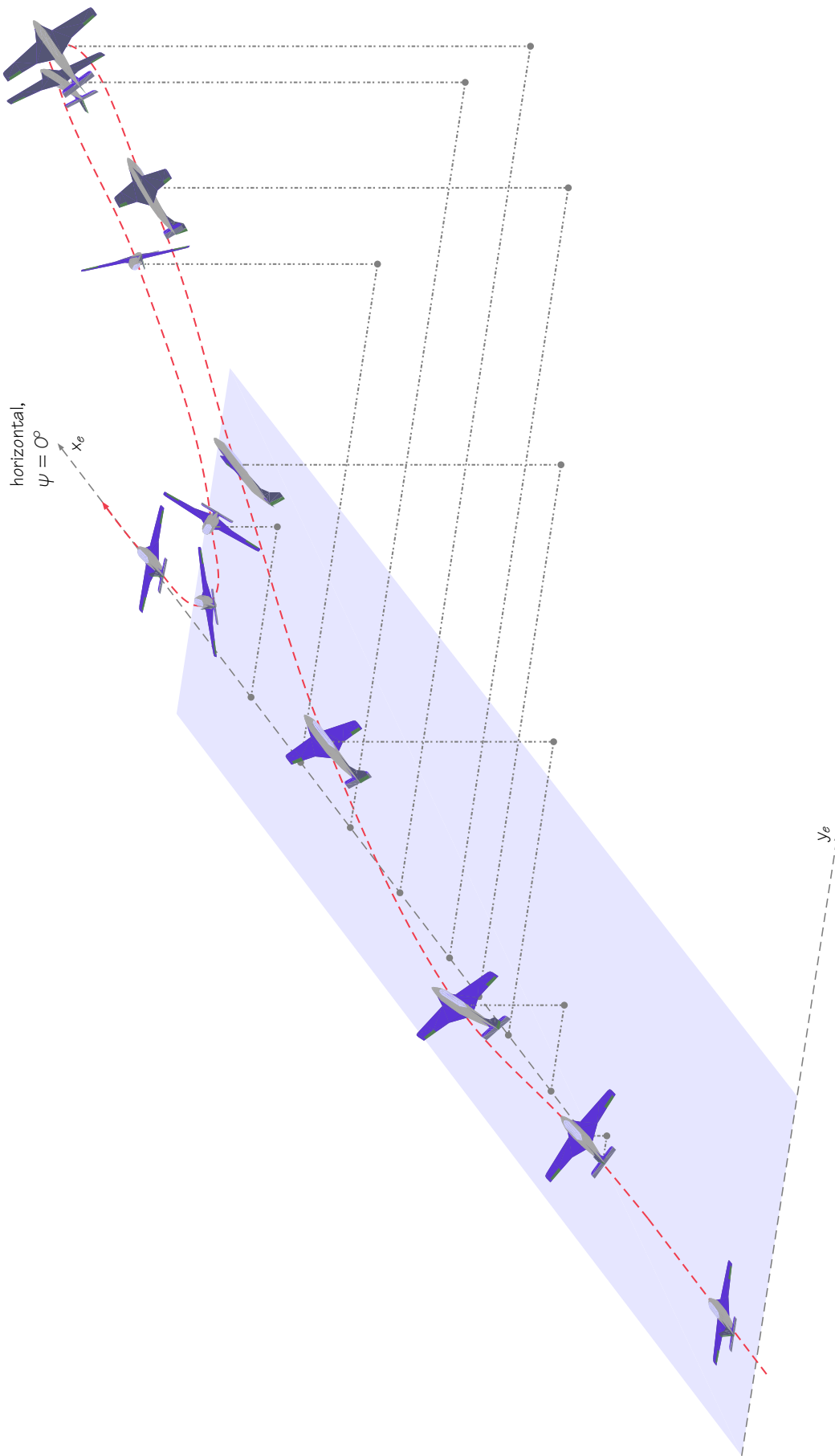


FIGURA 13: Esempio di evoluzione lungo una traiettoria non piana e con storie temporali non banali degli angoli di Eulero.