Capitolo 3: Simulazione della proiezione di alcuni punti 3D su un piano immagine

In questo capitolo presento l'ambiente per la simulazione da me sviluppato in Python, denominato **CoordCode**, che consente di proiettare insiemi di punti tridimensionali (X, Y, Z) sul piano immagine (u, v) secondo il **modello pinhole**.
La simulazione implementa la relazione:

$$\boldsymbol{u} = f\frac{X}{Z} + c_x$$

$$\boldsymbol{v} = f\frac{Y}{Z} + c_y$$

ed offre una doppia visualizzazione: la nuvola 3D (in piano tridimensionale) e la corrispondente proiezione 2D (con rappresentazione nel piano immagine).
Si hanno inoltre, strumenti di ispezione (zoom e pan), funzioni di import/export dei dati e possibilità di collegare i punti immessi nel grafico, in modo da dare un'idea più concreta della figura analizzata. Queste funzionalità sono descritte nella testata del sorgente CoordCode.py e all'interno della struttura nella classe principale dell'applicazione.

3.1: Spiegazione della simulazione

La simulazione è organizzata come segue:

- Nella schermata principale, appena eseguito il software, si inseriscono i Parametri Intrinseci, (f, c_x, c_y).
- Dopo di che si passa nella schermata per l'inserimento delle coordinate.
- Si forniscono, uno alla volta, i punti $P = (X, Y, Z)$ con $Z > 0$ in quanto il punto si deve trovare davanti alla camera.
- A questo punto, ogni punto viene proiettato in tempo reale sulla vista 2D e registrato nella tabella insieme alle coordinate calcolate automaticamente (u, v). La dipendenza inversa dalla profondità Z fa sì che, a parità di (X, Y) i punti più lontani (Z maggiore) risultino più vicini al principal point (c_x, c_y) nel piano immagine.
- La GUI permette di osservare direttamente questo effetto, infatti, l'algoritmo di proiezione è implementato nella funzione dedicata della classe applicativa.

3.2: Programma simulativo (*CoordCode*)

Architettura generale

Il programma è una GUI in Python/Tkinter organizzata in una classe principale (ApplicazioneCoordCode) che gestisce:

- Stato dell'applicazione (intrinseci, liste dei punti 3D e delle proiezioni 2D. collegamenti manuali)
- Costruzione delle due pagine dell'interfaccia.
- Disegno delle viste 2D/3D con Matplotlib.
- Importazione ed esportazione su file di testo.

Queste responsabilità sono elencate all'inizio del sorgente e nei commenti di sezione.

3.2.1: Schermata iniziale

All'avvio viene presentata una pagina introduttiva con il titolo CoordCode, una breve descrizione d'uso e due campi sequenziali: la focale f $(in\ pixel)$ e il principal point (c_x, c_y), inseriti come valori separati da virgola.

L'inserimento di f abilita il secondo campo e, alla conferma di (c_x, c_y), l'applicazione passa alla schermata operativa.

La costruzione della pagina, il bind del tasto Invio e la validazione dei valori sono implementati nei metodi che creano la pagina 1 e che eseguono i controlli sugli input.

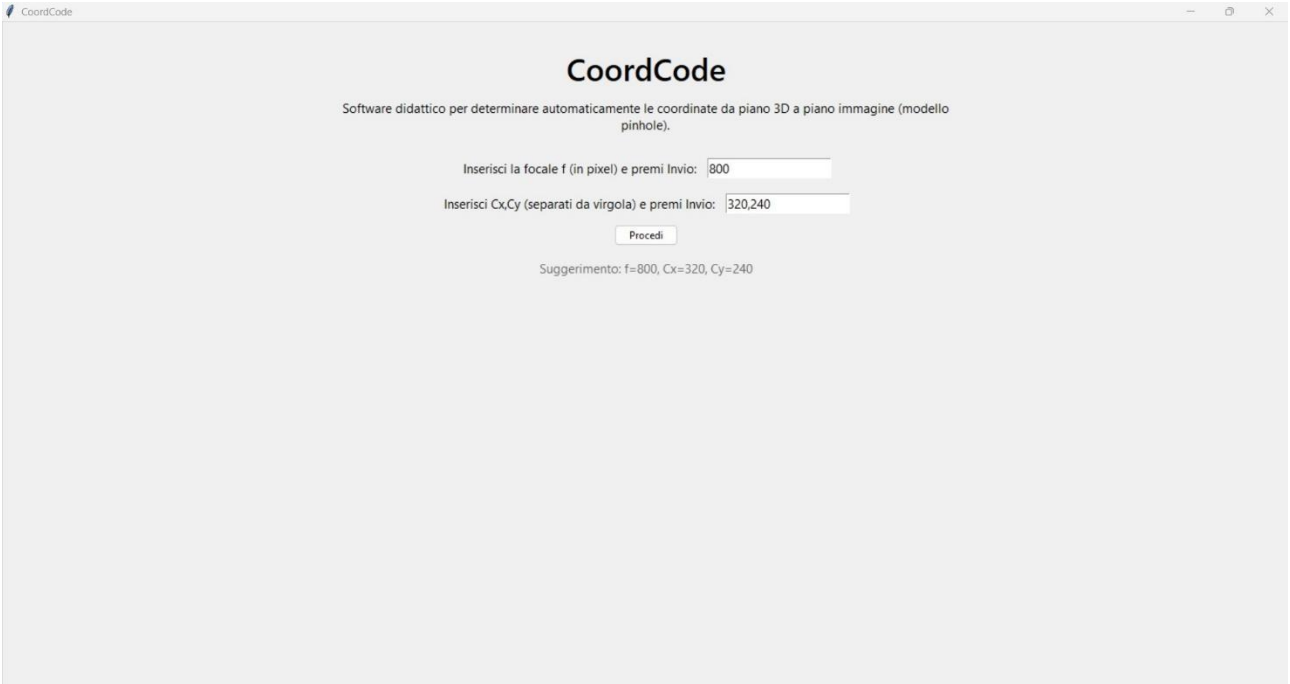

Figura: Schermata iniziale, è richiesto l'inserimento della focale e le coordinate di camera.

3.2.2: Schermata operativa (inserimento e visualizzazione)

La seconda pagina divide la finestra in due colonne:

- **Colonna sinistra:** campo di inserimento del punto (X, Y, Z), tabella con le colonne $\#, X, Y, Z, u, v$, pulsanti Esporta txt…, Importa txt… e Reset. L'aggiornamento della tabella avviene all'aggiunta di ciascun punto; l'intestazione e la creazione del widget Treeview con le colonne sopra indicate sono configurate a codice.

- **Colonna destra:** pannello Visualizzazione con scelta della Vista 2D (piano immagine) o Vista 3D (piano 3D) tramite radio button; opzioni Collega punti in ordine, Chiudi poligono e Mostra spigoli manuali tramite checkbox; box per inserire e gestire collegamenti manuali tra punti mediante una coppia di indici (i, j). La vista 2D mostra anche il principal point come segno "+"; entrambe le viste includono la toolbar di Matplotlib per zoom/pan.

La commutazione fra viste e il relativo ridisegno sono gestiti da metodi dedicati, con autoscale automatico dell'inquadratura; la toolbar viene istanziata per ciascuna canvas.

Qui riportato vi sono alcuni estratti di maggiore importanza del codice implementativo, e un esempio di come risulta il file di testo generato dall'esportazione:

- **Funzione di proiezione (modello pinhole)**

  È il cuore matematico: calcola le coordinate immagine a partire da un punto 3D.

```
def proietta_punto(self, x, y, z):
    if z <= 0:
        raise ValueError("Z deve essere > 0 "
                         "(il punto deve trovarsi davanti alla camera).")
    u = self.focale * (x / z) + self.cx
    v = self.focale * (y / z) + self.cy
    return u, v
```

- **Inserimento punto 3D e aggiornamento tabella**

  Gestisce input utente, calcola la proiezione e aggiorna GUI e grafico.

```
def aggiungi_punto(self, event=None):
    valori = self.entry_punto.get().split(",")
    x, y, z = map(float, valori)
    u, v = self.proietta_punto(x, y, z)
    self.punti3d.append((x, y, z))
    self.punti2d.append((u, v))
    self.tree.insert("", "end",
        values=(len(self.punti3d), f"{x:.2f}", f"{y:.2f}",
                f"{z:.2f}", f"{u:.2f}", f"{v:.2f}"))
    self.ridisegna_corrente()
```

- **Scelta della vista e opzioni di collegamento**

  Definisce radio button e check box che controllano la visualizzazione.

```
ttk.Radiobutton(opzioni_vista, text="Vista 2D (piano immagine)",
    variable=self.modalita_vista, value="2D",
    command=self.cambia_vista).pack(anchor="w")

ttk.Radiobutton(opzioni_vista, text="Vista 3D (piano 3D)",
    variable=self.modalita_vista, value="3D",
    command=self.cambia_vista).pack(anchor="w")

ttk.Checkbutton(opzioni_linee, text="Collega punti in ordine",
    variable=self.collega_in_ordine_var,
    command=lambda: self.ridisegna_corrente(True)).pack(anchor="w")
```

- **Esportazione su file di testo**

  Scrive i parametri e i punti su file “.txt”, permettendo di salvare la sessione.

```
def esporta_txt(self):
    file = filedialog.asksaveasfilename(defaultextension=".txt")
    if not file: return
    with open(file, "w") as f:
        f.write("[Camera]\n")
        f.write(f"f={self.focale}, cx={self.cx}, cy={self.cy}\n\n")
        f.write("[Punti]\n")
        for i, ((x, y, z), (u, v)) in \
            enumerate(zip(self.punti3d, self.punti2d), start=1):
            f.write(f"{i}: {x},{y},{z} -> {u},{v}\n")
```

E questo è un esempio di come appare la paginazione nel documento “.txt”:

```
==================== COORDCODE — ESPORTAZIONE DATI ====================
Descrizione: punti 3D, proiezioni (u,v) sul piano immagine e collegamenti definiti dall'utente.
Nota: i punti sono elencati nell'ordine di inserimento; le coordinate u,v sono in pixel.

[Camera]
  f  = 800        # focale in pixel
  cx = 320        # coordinata u del punto principale
  cy = 240        # coordinata v del punto principale

[Punti]
  # indice | X | Y | Z || u | v
   1)  X=200       Y=200       Z=2000       ==>  u=400.0000   v=320.0000
   2)  X=200       Y=-200      Z=2000       ==>  u=400.0000   v=160.0000
   3)  X=-200      Y=200       Z=2000       ==>  u=240.0000   v=320.0000
   4)  X=-200      Y=-200      Z=2000       ==>  u=240.0000   v=160.0000
   5)  X=200       Y=200       Z=2500       ==>  u=384.0000   v=304.0000
   6)  X=200       Y=-200      Z=2500       ==>  u=384.0000   v=176.0000
   7)  X=-200      Y=200       Z=2500       ==>  u=256.0000   v=304.0000
   8)  X=-200      Y=-200      Z=2500       ==>  u=256.0000   v=176.0000
   9)  X=100       Y=100       Z=2150       ==>  u=357.2093   v=277.2093
  10)  X=-100      Y=-100      Z=2150       ==>  u=282.7907   v=202.7907
  11)  X=-100      Y=100       Z=2150       ==>  u=282.7907   v=277.2093
  12)  X=100       Y=-100      Z=2150       ==>  u=357.2093   v=202.7907
  13)  X=-100      Y=-100      Z=2350       ==>  u=285.2174   v=205.2174
  14)  X=100       Y=100       Z=2350       ==>  u=354.7826   v=274.7826
  15)  X=-100      Y=100       Z=2350       ==>  u=285.2174   v=274.7826
  16)  X=100       Y=-100      Z=2350       ==>  u=354.7826   v=205.2174

[SpigoliManuali]
  # elenco di coppie (i, j) che collegano i punti con indici i e j
  (5, 7)
  (7, 8)
  (3, 7)
  (5, 6)
  (6, 8)
  (2, 6)
  (1, 3)
  (1, 2)
  (1, 5)
  (2, 4)
  (3, 4)
  (4, 8)
  (13, 15)
  (13, 16)
  (14, 15)
  (14, 16)
  (10, 11)
  (10, 12)
  (9, 12)
  (9, 11)
  (11, 15)
  (9, 14)
  (12, 16)
  (10, 13)
  (8, 13)
  (7, 15)
  (5, 14)
  (1, 9)
  (2, 12)
  (4, 10)
========================== FINE ESPORTAZIONE ==========================
```

3.3: Esempi pratici

In questa sezione presento alcune configurazioni di punti 3D, generate per verificare visivamente la coerenza tra vista tridimensionale e proiezione sul piano immagine. In tutti i casi, i parametri intrinseci sono:

$$\boldsymbol{f} = 800, \boldsymbol{c_x} = 320, \boldsymbol{c_y} = 240 \ (\textit{unità in pixel})$$

Le figure che seguono sono ottenute con CoordCode, impiegando le opzioni di collegamento automatico e/o gli spigoli manuali quando necessario (si veda il pannello dedicato nella GUI).

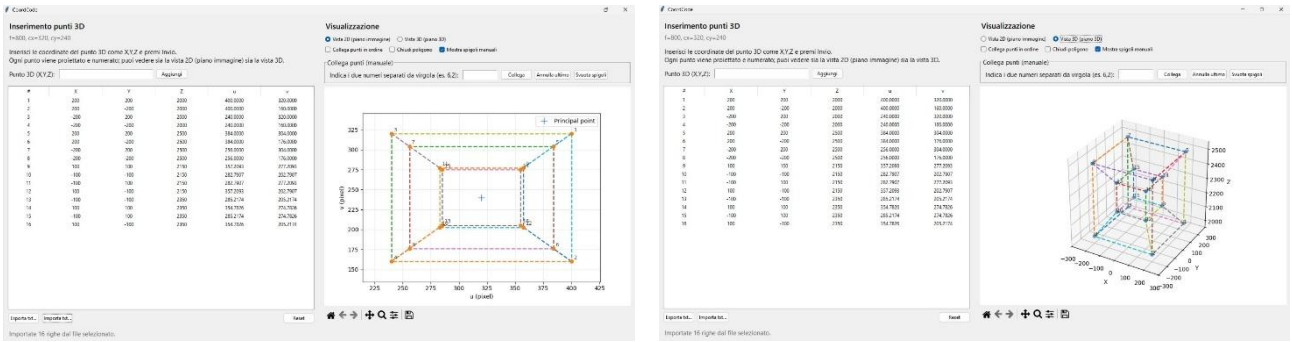

Figure: *"Doppio cubo": a sinistra visione 2D in piano Immagine, due cubi a profondità distinte proiettano due "cornici" concentriche: il cubo più lontano ha ingombro minore nell'immagine. A destra visione 3D in spazio Reale, la vista tridimensionale conferma la diversa quota Z dei due solidi e la coerenza con la proiezione osservata.*

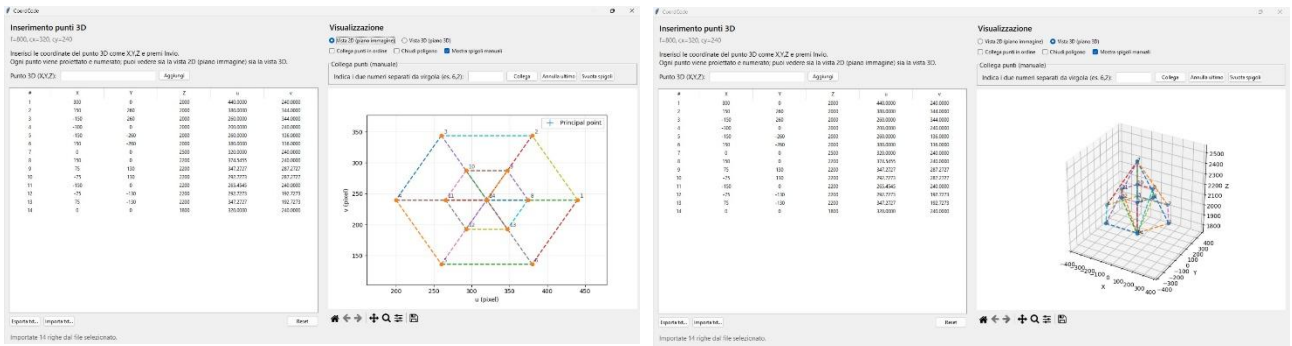

Figure: *"Doppia piramide": a sinistra visione 2D in piano Immagine, la figura mostra le proiezioni* (u, v) *dei vertici di due piramidi con quote Z differenti: la piramide più distante risulta contratta verso* (c_x, c_y)*, coerentemente con il fattore 1/Z nella proiezione pinhole. A destra visione 3D in spazio Reale, evidenzia le altezze e la disposizione dei vertici; i collegamenti riproducono quelli usati nella 2D.*

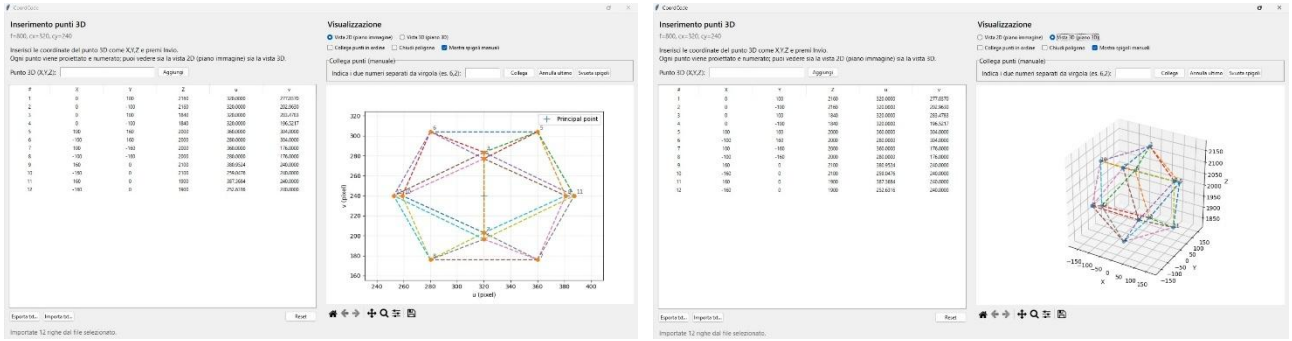

Figure: *"Icosaedro": a sinistra visione 2D in piano Immagine, la proiezione di un poliedro regolare mette in evidenza la prospettiva: lati non paralleli e variazioni di scala legate alla profondità. A destra visione 3D in spazio Reale, la geometria dei vertici e la connettività risultano pienamente leggibili nella vista 3D e rispecchiano i legami tracciati sul piano immagine.*

3.4: Spiegazione del risultato

Gli esperimenti mostrano che la pipeline implementata da CoordCode restituisce, per ogni punto 3D, una proiezione 2D coerente con il modello pinhole.

La trasformazione:

$$(X, Y, Z) \rightarrow (u, v)$$

dipende linearmente da X/Z e Y/Z e incorpora la traslazione del principal point (c_x, c_y).

L'effetto più evidente è la **dipendenza inversa dalla profondità**: a parità di (X, Y), la proiezione si avvicina a (c_x, c_y) all'aumentare di Z; questo comportamento è verificabile direttamente nella GUI passando dalla vista 3D alla corrispondente 2D.

La realizzazione pratica sfrutta la funzione di proiezione dell'applicazione e i metodi di ridisegno delle viste con autoscale e strumenti integrati di ispezione.

3.4.1: Perdita di informazione e ambiguità della proiezione.

I test condotti con *CoordCode* mettono in evidenza una perdita di informazione intrinseca della proiezione prospettica: la mappa $(X, Y, Z) \rightarrow (u, v)$ non è iniettiva.

In particolare, per ogni $\lambda > 0$ i punti (X, Y, Z) e $(\lambda X, \lambda Y, \lambda Z)$ producono la stessa proiezione, poiché:

$$\boldsymbol{u} = f\frac{X}{Z} + c_x = f\frac{\lambda X}{\lambda Y} + c_x$$

$$\boldsymbol{v} = f\frac{Y}{Z} + c_y = f\frac{\lambda Y}{\lambda Z} + c_y$$

Ne consegue che tutti i punti allineati sulla medesima semiretta uscente dal centro ottico risultano sovrapposti in 2D e dunque indistinguibili sul piano immagine. Inoltre, la riduzione di dimensionalità da $\mathbb{R}^3$ a $\mathbb{R}^2$ comporta la perdita della coordinata di profondità: la tridimensionalità della figura non è più direttamente accessibile, rendendo più complessa la comprensione della struttura a partire da una singola vista.