# Implementazione in Python del Poisson Image Editing

Cristian Casali

October 2024

## 1 Descrizione

Lo scopo di questo progetto, sviluppato per il corso di Computer Graphics, era quello di implementare il metodo chiamato *Poisson Image Editing* che consente di effettuare il blending di immagini senza vedere i bordi. Per l'implementazione, è stata seguita la pubblicazione [2].

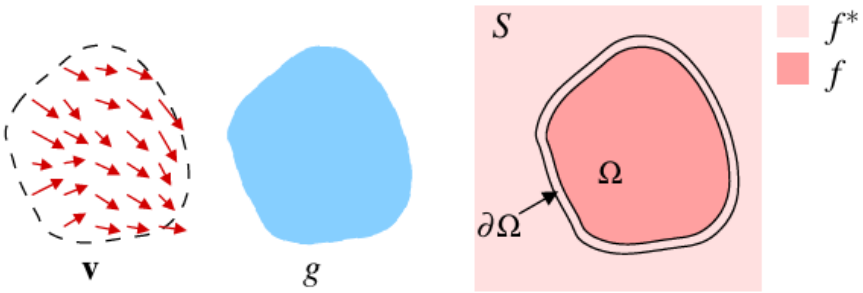



Figura 1: **Notazioni:** La funzione incognita $f$ interpola nel dominio $\Omega$ la funzione di destinazione $f^*$, sotto la guida del campo vettoriale $\mathbf{v}$, che può essere il campo del gradiente di una funzione sorgente $g$

La funzione che consente di interpolare $f$ a partire da $f^*$ su $\Omega$ è la seguente:

$$min_f \int\int |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\delta\Omega} = f^*|_{\delta\Omega} \tag{1}$$

nella quale è stato aggiunto il campo vettoriale $\mathbf{v}$ come ulteriore vincolo. La sua soluzione deve quindi soddisfare l'equazione

$$\Delta f = div\mathbf{v} \text{ over } \Omega, \text{ with } f|_{\delta\Omega} = f^*|_{\delta\Omega} \tag{2}$$

dove $div\mathbf{v} = \frac{\delta u}{\delta x} + \frac{\delta v}{\delta y}$ è la *divergence* di $\mathbf{v}$ = (u, v).

Naturalmente, lavorando con immagini digitali, queste equazioni vanno discretizzate. Senza perdere di generalizzazione, terremo le stesse notazioni del caso continuo: $S$ e $\Omega$ diventano quindi insiemi finiti di punti (con $S$ che può includere tutti i punti di un'immagine o un loro sottoinsieme). Per ogni punto $p$ in $S$, sia $N_p$ l'insieme dei suoi vicini 4-connected (in $S$), mentre $\langle p, q\rangle$ indica la coppia di pixel $p$ e $q$ tale che $q \in N_p$. Infine, il bordo di $\Omega$ è ora definito come l'insieme $\delta\Omega = \{p \in S \setminus \Omega : N_p \cap \Omega \neq \emptyset\}$.
Portando quindi l'equazione (1) nel dominio discreto si ottiene:

$$min_{f|_\Omega} \sum_{\langle p,q\rangle \cap \Omega \neq \emptyset} (f_p - f_q - v_{pq})^2, \text{ with } f_p = f_p^*, \text{ for all } p \in \delta\Omega \tag{3}$$

Le cui soluzioni soddisfano l'equazione:

$$\text{for all } p \in \Omega, |N_p| f_p - \sum_{q \in N_p \cap \Omega} f_q = \sum_{q \in N_p \cap \delta\Omega} f_q^* + \sum_{q \in N_p} v_{pq} \tag{4}$$

In questo progetto, si andrà ad implementare l'equazione (4), scegliendo come $\mathbf{v}$ il gradiente preso direttamente dall'immagine sorgente. Perciò, chiamata $g$ l'immagine sorgente, l'interpolazione è effettuata sotto la guida di

$$\mathbf{v} = \nabla g \tag{5}$$

e (2) si legge ora come

$$\Delta f = \Delta g \text{ over } \Omega, \text{ with } f|_{\delta\Omega} = f^*|_{\delta\Omega} \tag{6}$$

In campo discreto, (5) diventa:

$$\text{for all } \langle p, q \rangle, v_{pq} = g_p - g_q \tag{7}$$

che va quindi inserita in (4).

# 2 Implementazione[1]

Il metodo *poisson_editing* accetta come parametri tre immagini sottoforma di array numpy:

- **source** dalla quale viene presa la patch da copiare;
- **target** sulla quale viene inserita la patch;
- **mask** che seleziona i pixel di interesse della source.

Accetta inoltre due parametri facoltativi:

- **offset** che indica di quanti pixel va spostata verticalmente e orizzontalmente la patch (di default è (0,0));
- **mixing** un flag che consente di mescolare i gradienti di source e target.

Come prima cosa calcola le regioni di interesse delle immagini source e target e la loro dimensione. A causa dell'offset infatti, potrebbe verificarsi la situazione in cui le dimensioni della regione non sono quelle della mask.

Dopo aver fatto ciò, calcola le matrici **A** e **b** che corrispondono ai due membri dell'equazione (4). **A** è il membro a sinistra che viene moltiplicato per l'incognita *f*, mentre **b** è il termine noto a destra dell'equazione. Chiamata *size = region_size[0] * region_size[1]*, **A** è una matrice quadrata di dimensioni *size*size*. Ogni riga e colonna identificano un pixel dell'immagine: se un determinato pixel si trova all'interno della mask, allora il valore della diagonale principale di **A** corrispondente a quel pixel è 4 (o inferiore in base alla cardinalità di $N_p$), altrimenti è 1. Inoltre, sulla riga e sulla colonna di quel pixel, vengono posti a -1 tutti i valori dei pixel appartenenti a $N_p \cap \Omega$. In questo modo è stato ricostruito il primo membro dell'equazione (4). Ad esempio, data la mask $\begin{pmatrix} 0 & 255 & 0 \\ 255 & 255 & 255 \\ 0 & 255 & 0 \end{pmatrix}$ la matrice **A** diventa:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{8}$$

L'unico elemento posto a 4 è quello centrale perché è l'unico a non essere ai limiti della mask.

Nel vettore **b** di lunghezza *size*, ogni elemento rappresenta un pixel dell'immagine. Se il pixel non appartiene alla mask allora viene semplicemente copiato il valore che quel pixel ha nell'immagine target. Se invece il pixel in questione appartiene alla mask, per ognuno dei quattro vicini 4-connected (se esistono), viene calcolata la differenza tra il valore del pixel e quello del vicino nell'immagine sorgente. Nel caso in cui il flag *mixing* fosse posto a *True*, viene invece applicato il seguente codice (supponendo di stare valutando il pixel (i,j) e il vicino (i-1,j)):

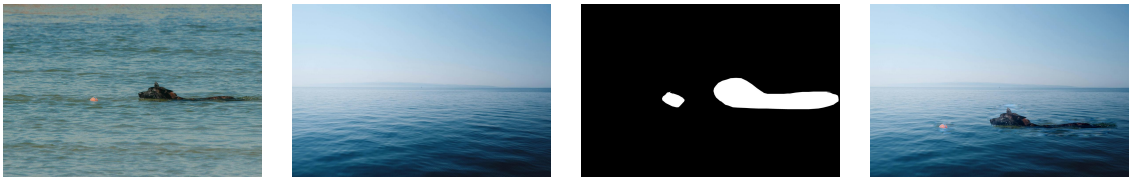
```
diff_g = source[i,j] - source[i-1,j]
diff_f = target[i,j] - target[i-1,j]
if abs(diff_f) > abs(diff_g):
    b[index] += diff_f
else:
    b[index] += diff_g
```

[1] Per la stesura del codice di questo progetto è stato consultato e in parte utilizzato il codice presente in [1]

Infine, se il vicino appartiene al bordo della regione, viene aggiunto anche il valore che quel pixel vicino ha nell'immagine target.

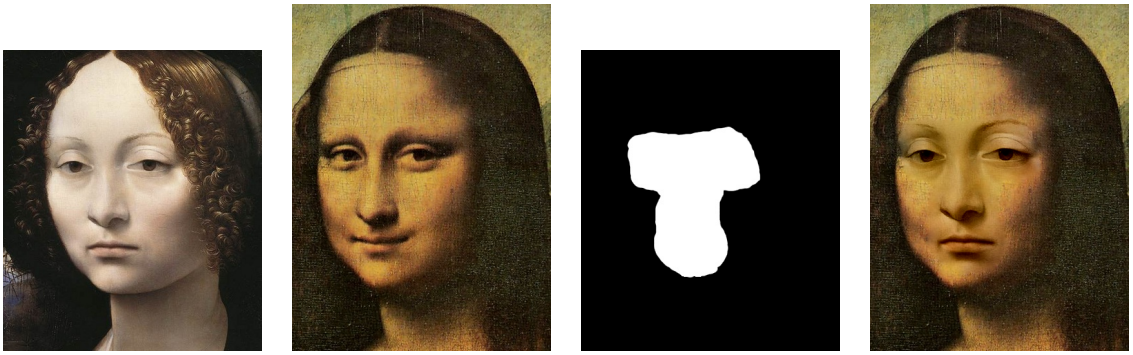
Dati $\mathbf{A}$ e $\mathbf{b}$, vengono quindi calcolati i valori incogniti risolvendo l'equazione $\mathbf{A}x = \mathbf{b}$. $\mathbf{A}$ è unica per ogni canale dell'immagine, mentre $\mathbf{b}$ e l'incognita $x$ vengono calcolati per ogni canale.

# 3 Risultati[2]



(a) Source (b) Target (c) Mask (d) Result

Figura 2: **Inserimento:** Le potenzialità del Poisson Image Editing vengono pienamente apprezzate quando si inseriscono oggetti con bordi irregolari in nuovi background



(a) Source (b) Target (c) Mask (d) Result

Figura 3: **Feature exchange:** Un'altra delle possibilità che offre è quella di scambiare alcune feature tra un oggetto e un altro

(a) Source (b) Target (c) Mask

Figura 4.1: **Trasferimento monocromatico:** A volte si possono ottenere risultati migliori rendendo l'immagine sorgente in scala di grigi prima di effettuare l'operazione, come visto in fig. 4.2

[2]Tutte le immagini presenti sono state prese da [3] o scattate direttamente dall'autore

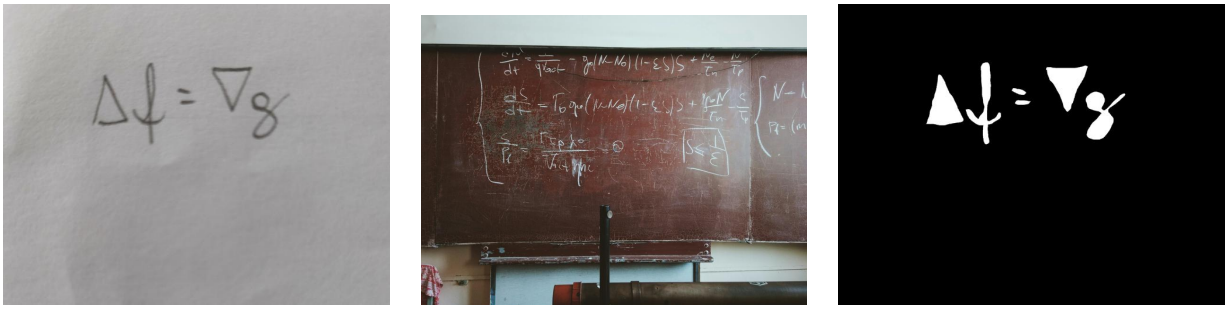
(a) Risultato ottenuto con il metodo classico

(b) Risultato ottenuto rendendo grayscale l'immagine sorgente

(c) Risultato ottenuto mescolando i gradienti

(d) Risultato ottenuto mescolando i gradienti, con l'immagine sorgente grayscale

Figura 4.2: **Trasferimento monocromatico risultati:** Osservando i risultati ottenuti si può notare che, rendendo monocromatica l'immagine sorgente, la patch copiata diventa del colore dell'immagine target perdendo il colore originale e rendendo il risultato più verosimile (come in fig. 4.2b). Utilizzando poi la tecnica del *mixing gradient* possiamo mantenere anche la texture dell'immagine target (fig. 4.2c e 4.2d).

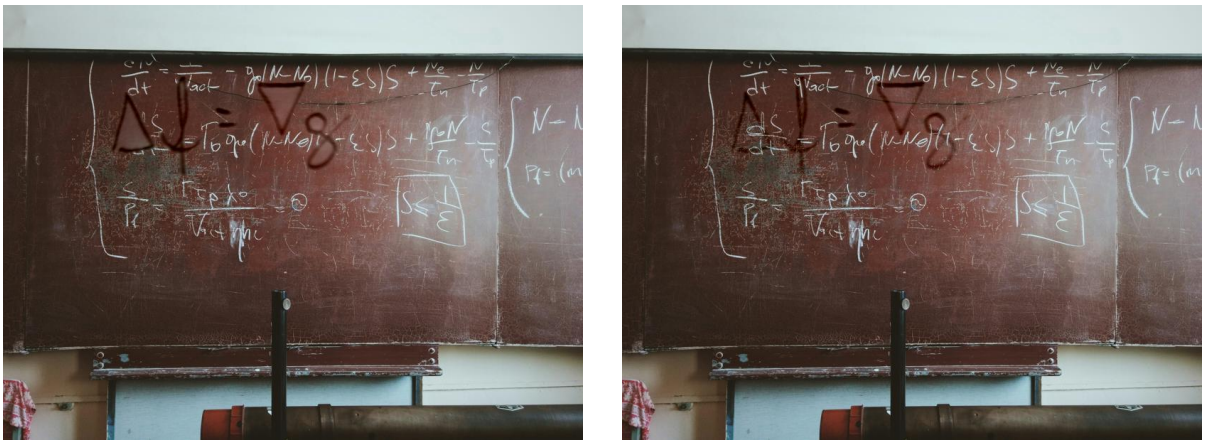


(a) Source

(b) Target

(c) Mask

Figura 5.1: **Inserire oggetti con buchi:** Questo metodo consente anche di inserire oggetti con fori al loro interno senza dover utilizzare maschere troppo complesse.



(a) Risultato ottenuto con il metodo classico

(b) Risultato ottenuto con il mescolando i gradienti

Figura 5.2: **Risultati:** Il metodo classico (fig. 5.2a) lascerebbe aloni indesiderati, mentre utilizzando la tecnica del mixing gradient, è possibile ottenere buoni risultati anche con una mask più semplice come quella in 5.1c (fig. 5.2b).

(a) Source (b) Target (c) Mask

Figura 6.1: **Inserire oggetti trasparenti:** Mescolando i gradienti è possibile anche trasferire oggetti parzialmente trasparenti come l'arcobaleno di questo esempio. La selezione non lineare del gradiente prende la struttura più importante in ogni punto.

(a) Risultato ottenuto con il metodo classico

(b) Risultato ottenuto mescolando i gradienti

(c) Risultato senza offset ottenuto con il metodo classico

(d) Risultato senza offset con gradiente mescolato

Figura 6.2: **Risultati:** Confrontando le due immagini senza offset (6.2c e 6.2d) si può notare bene la differenza tra il metodo classico e la tecnica del mixing gradient.

(a) Source (b) Target (c) Mask (d) Result

Figura 7: **Local color changes:** Il Poisson Editing è anche un potente mezzo per manipolare i colori. Ad esempio, il processo di rendere un'immagine completamente monocromatica, con l'eccezione di una zona di interesse può essere effettuato utilizzando come target la componente della luminosità dell'immagine sorgente (fig. 7b).

(a) Source (b) Mask (c) Result

Figura 8: **Texture flattening:** Grazie al Poisson Editing è anche possibile rimuovere le texture di certe regioni dell'immagine. È sufficiente fare uso di un algoritmo di edge detector (nel caso in questione è stato utilizzato quello disponibile nella libreria Pillow) e sostituire a $v_{pq}$ la seguente espressione per ogni $\langle p, q \rangle$:

$$v_{pq} = \begin{cases} f_p - f_q & \text{se è presente un bordo tra p e q} \\ 0 & \text{altrimenti} \end{cases}$$

# Riferimenti bibliografici


[1] parosky. *poissonblending*. https://github.com/parosky/poissonblending. 2013.

[2] Patrick Pérez, Michel Gangnet e Andrew Blake. "Poisson image editing". In: *ACM SIGGRAPH 2003 Papers*. SIGGRAPH '03. San Diego, California: Association for Computing Machinery, 2003, pp. 313–318. ISBN: 1581137095. DOI: 10.1145/1201775.882269. URL: https://doi.org/10.1145/1201775.882269.

[3] *Unsplash*. URL: https://unsplash.com/.