

Les 8 Voisins et le Périmètre

Calculs avec NumPy

Guide visuel pour Game of Life et traitement d'images

1. Les 8 voisins d'une cellule

NW (-1,-1)	N (0,-1)	NE (+1,-1)
W (-1,0)	CELLULE (x,y)	E (+1,0)
SW (-1,+1)	S (0,+1)	SE (+1,+1)

Légende : N=Nord, S=Sud, E=Est, W=Ouest

2. Game of Life - Calcul avec slicing

Pour compter les 8 voisins d'une cellule à (x,y) :

```
neighbours = sum(world[x-1][y-1:y+2]) \
              + sum(world[x][y-1:y+2:2]) \
              + sum(world[x+1][y-1:y+2])

# Détail:
# world[x-1][y-1:y+2] → 3 voisins (ligne du haut)
# world[x][y-1:y+2:2] → 2 voisins (gauche et droite)
# world[x+1][y-1:y+2] → 3 voisins (ligne du bas)
# TOTAL: 8 voisins
```

Règles de Game of Life

État	Voisins	Résultat
Morte (0)	3	Naissance → 1
Morte (0)	Autre	Reste morte → 0
Vivante (1)	2 ou 3	Survie → 1
Vivante (1)	< 2	Meurt (solitude) → 0
Vivante (1)	> 3	Meurt (surpopulation) → 0

3. Slicing NumPy pour accéder aux voisins

Opération	Description	Effet
<code>image[:-1, :]</code>	Retire dernière ligne	Décalage HAUT
<code>image[1:, :]</code>	Retire première ligne	Décalage BAS
<code>image[:, :-1]</code>	Retire dernière colonne	Décalage GAUCHE
<code>image[:, 1:]</code>	Retire première colonne	Décalage DROITE

Important : Les dimensions des slices sont réduites ! Une matrice (10,10) devient (9,10) ou (10,9).

Exemple visuel du décalage

```
# Image 5x5:
image = [[0,1,2,3,4],
         [5,6,7,8,9],
         [A,B,C,D,E],
         [F,G,H,I,J],
         [K,L,M,N,O]]

image[:-1,:] → [[0,1,2,3,4], # Lignes 0-3
               [5,6,7,8,9],
               [A,B,C,D,E],
               [F,G,H,I,J]]

image[1:,:] → [[5,6,7,8,9], # Lignes 1-4
               [A,B,C,D,E],
               [F,G,H,I,J],
               [K,L,M,N,O]]
```

4. Calcul du périmètre

Le périmètre compte les pixels de bord de la forme (qui touchent le fond).

Algorithme complet :

```
def perimeter(image):
    # 1. Masque booléen
    form_mask = image != 0

    # 2. Voisins (4 directions cardinales)
    top = form_mask[:-1, :]
    bottom = form_mask[1:, :]
    left = form_mask[:, :-1]
    right = form_mask[:, 1:]

    # 3. Détecter les bords
    # Un pixel est un bord si lui=1 ET son voisin=0
    top_edge = form_mask[1:, :] & ~top
    bottom_edge = form_mask[:-1, :] & ~bottom
    left_edge = form_mask[:, 1:] & ~left
    right_edge = form_mask[:, :-1] & ~right

    # 4. Détecter les coins (double comptage)
    top_left = form_mask[1:, 1:] & ~top[:, 1:] & ~left[1:, :]
    top_right = form_mask[1:, :-1] & ~top[:, :-1] & ~right[1:, :]
    bottom_left = form_mask[:-1, 1:] & ~bottom[:, 1:] & ~left[:-1, :]
    bottom_right = form_mask[:-1, :-1] & ~bottom[:, :-1] & ~right[:-1, :]

    # 5. Périmètre = Bords - Coins
    return (np.sum(top_edge) + np.sum(bottom_edge) +
            np.sum(left_edge) + np.sum(right_edge) -
            (np.sum(top_left) + np.sum(top_right) +
             np.sum(bottom_left) + np.sum(bottom_right)))
```

Pourquoi soustraire les coins ?

Les coins sont comptés **deux fois** : une fois dans le bord horizontal ET une fois dans le bord vertical. En les détectant et en les soustrayant, on corrige ce double comptage.

Exemple d'un coin supérieur-gauche:

```
0  0  0
0  ■  ■  ← Le pixel (1,1) est détecté comme:
0  ■  ■      • Bord supérieur (voisin du haut = 0)
              • Bord gauche (voisin de gauche = 0)
              → Compté 2 fois sans correction !
```

5. Game of Life vs Périmètre

Aspect	Game of Life	Périmètre
Voisins considérés	8 (avec diagonales)	4 (cardinaux uniquement)
But	Compter tous les voisins vivants	Détecter les bords de forme
Opération	Somme des valeurs	Opérations booléennes
Résultat	Nombre (0-8)	Longueur du contour

6. Opérateurs logiques NumPy

Opérateur	Symbole	Description	Exemple
AND	&	Vrai si les deux sont vrais	<code>mask & neighbor</code>
OR		Vrai si au moins un est vrai	<code>mask1 mask2</code>
NOT	~	Inverse (True→False, False→True)	<code>~mask</code>
XOR	^	Vrai si exactement un est vrai	<code>mask1 ^ mask2</code>

Note importante : Utilisez & | ~ pour NumPy, pas 'and' 'or' 'not' (qui sont pour les scalaires).

7. Exemple pratique complet

```
import numpy as np

# Image en forme de losange
image = np.array([[0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,1,1,0,0,0,0],
                  [0,0,0,1,1,1,1,0,0,0],
                  [0,0,0,1,1,1,1,0,0,0],
                  [0,0,0,1,1,1,1,0,0,0],
                  [0,0,0,0,1,1,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0]])

# Calcul étape par étape
form_mask = image != 0
top = form_mask[:-1, :]
bottom = form_mask[1:, :]
left = form_mask[:, :-1]
right = form_mask[:, 1:]

top_edge = form_mask[1:, :] & ~top          # 4 pixels
bottom_edge = form_mask[:-1, :] & ~bottom  # 4 pixels
```

```

left_edge = form_mask[:, 1:] & ~left      # 4 pixels
right_edge = form_mask[:, :-1] & ~right   # 4 pixels

total_edges = np.sum(top_edge) + np.sum(bottom_edge) + \
              np.sum(left_edge) + np.sum(right_edge)
# total_edges = 16

# Détecter les coins
top_left = form_mask[1:,1:] & ~top[:,1:] & ~left[1:,:] # 2
top_right = form_mask[1:,-1] & ~top[:,:-1] & ~right[1:,:] # 2
bottom_left = form_mask[:-1,1:] & ~bottom[:,1:] & ~left[:-1,:] # 2
bottom_right = form_mask[:-1,-1] & ~bottom[:,:-1] & ~right[:-1,:] # 2

total_corners = np.sum(top_left) + np.sum(top_right) + \
                np.sum(bottom_left) + np.sum(bottom_right)
# total_corners = 8

perimeter = total_edges - total_corners
# perimeter = 16 - 8 = 8

```

Résultat : Le périmètre du losange est 8 pixels.

8. Résumé des concepts clés

- ✓ **Slicing pour décalage** : Accéder aux voisins sans boucles
- ✓ **Opérations logiques** : Détecter les conditions avec &, |, ~
- ✓ **Vectorisation** : Opérations sur toute la matrice simultanément
- ✓ **Double comptage** : Soustraire les coins pour corriger
- ✓ **8 vs 4 voisins** : Selon le contexte (connectivité vs contour)

Applications pratiques

- **Game of Life** : Automate cellulaire
- **Détection de contours** : Vision par ordinateur
- **Segmentation d'images** : Traitement d'images médicales
- **Morphologie mathématique** : Érosion, dilatation
- **Analyse de formes** : Calcul d'aire, périmètre, centroïde

Avantages de NumPy

Aspect	Sans NumPy	Avec NumPy
Performance	Lent (boucles Python)	Rapide (code C optimisé)
Code	Verbeux, complexe	Concis, élégant
Mémoire	Inefficace	Optimisée
Lisibilité	Difficile	Clair et mathématique