TP 4. Opérateurs morphologiques

1. Opérateurs morphologiques

$I := \text{READ\_IMAGE}(\text{"croixb.bmp"})$ Nombre de colonnes $Nx := \text{cols}(I)$ $Nx = 140$ Nombre de lignes $Ny := \text{rows}(I)$ $Ny = 152$

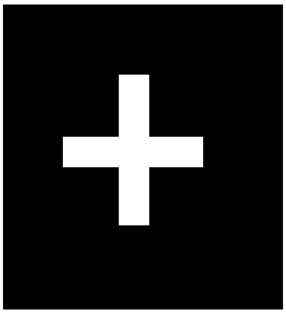
Q4.1. Transposer sous Python les algorithmes de dilatation (respectivement d'erosion). Y-a-t-il une tendance vers une stabilité du motif après un grand nombre de dilatations (respectivement d'erosions) ? Que se passe-t-il si on applique une erosion suivie d'une dilatation ? Une dilatation suivie d'une erosion ?
Comment utiliser ces algorithmes pour extraire les contours d'une image binaire ? Test sur autres images binaires

```
Dilatation(I) ≡ | J ← I
                | for y ∈ 1 .. rows(I) − 2
                |   for x ∈ 1 .. cols(I) − 2
                |     | for i ∈ 0 .. 2
                |     |   for j ∈ 0 .. 2
                |     |     v_{3·i+j} ← I_{y+i−1, x+j−1} / 255
                |     | Test ← v_0
                |     | for i ∈ 1 .. 8
                |     |   Test ← v_i ∨ Test  if i ≠ 4
                |     | J_{y,x} ← 255  if Test = 1
                |     | J_{y,x} ← I_{y,x}  otherwise
                | J
```

```
Erosion(I) ≡ | J ← I
             | for y ∈ 1 .. rows(I) − 2
             |   for x ∈ 1 .. cols(I) − 2
             |     if I_{y,x} = 255
             |       | for i ∈ 0 .. 2
             |       |   for j ∈ 0 .. 2
             |       |     v_{3·i+j} ← I_{y+i−1, x+j−1} / 255
             |       | Test ← v_0
             |       | for i ∈ 1 .. 8
             |       |   Test ← v_i ∧ Test  if i ≠ 4
             |       | J_{y,x} ← I_{y,x}  if Test = 1
             |       | J_{y,x} ← 0  otherwise
             | J
```

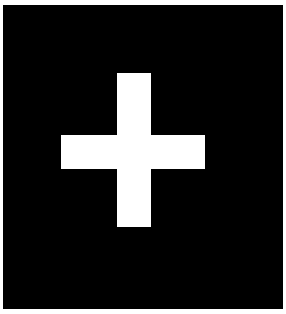
Dilatation : $J := \text{Dilatation}(I)$ $K := \text{Dilatation}(\text{Dilatation}(I))$

Image originale



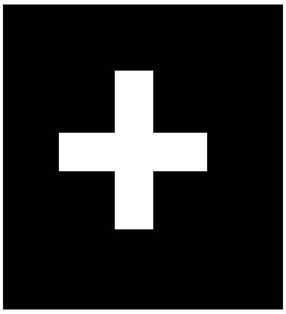
I

Image dilatée



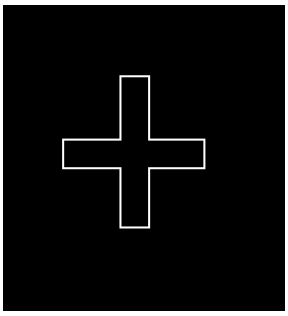
J

Image dilatée 2 fois



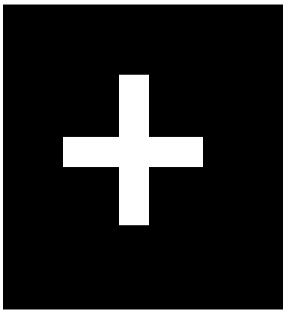
K

Contours par érosion



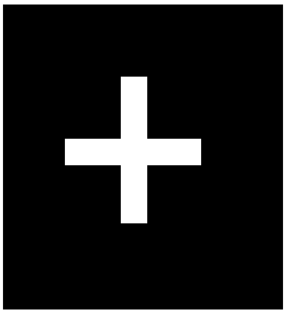
Erosion : $J := \text{Erosion}(I)$ $K := \text{Erosion}(\text{Erosion}(I))$

Image originale



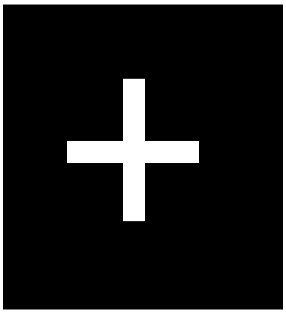
I

Image érodée



J

Image érodée 2 fois



K

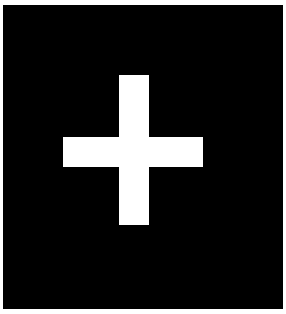
Axe Médian : ***Q4.2. Transposer sous Python l'algorithme d'axe median. Test sur autres images binaires***

Bords noirs

$$
\begin{array}{l}
\mathrm{AxeMedian}(I) \equiv \\
\quad \text{for } y \in 0..\mathrm{rows}(I)-1 \\
\quad\quad I_{y,0} \leftarrow 0 \\
\quad\quad I_{y,\mathrm{cols}(I)-1} \leftarrow 0 \\
\quad \text{for } x \in 0..\mathrm{cols}(I)-1 \\
\quad\quad I_{0,x} \leftarrow 0 \\
\quad\quad I_{\mathrm{rows}(I)-1,x} \leftarrow 0 \\
\quad \mathrm{stable} \leftarrow 0 \\
\quad \text{while } \mathrm{stable} = 0 \\
\quad\quad \mathrm{stable} \leftarrow 1 \\
\quad\quad \text{for } y \in 0..\mathrm{rows}(I)-1 \\
\quad\quad\quad \text{for } x \in 0..\mathrm{cols}(I)-1 \\
\quad\quad\quad\quad \text{if } I_{y,x} = 255 \\
\quad\quad\quad\quad\quad F1 \leftarrow \frac{I_{y-1,x+1}}{255} \wedge \frac{I_{y,x+1}}{255} \wedge \frac{I_{y+1,x+1}}{255} \wedge \neg\left(\frac{I_{y-1,x-1}}{255} \vee \frac{I_{y,x-1}}{255} \vee \frac{I_{y+1,x-1}}{255}\right) \\
\quad\quad\quad\quad\quad F2 \leftarrow \frac{I_{y,x+1}}{255} \wedge \frac{I_{y+1,x+1}}{255} \wedge \frac{I_{y+1,x}}{255} \wedge \neg\left(\frac{I_{y-1,x}}{255} \vee \frac{I_{y-1,x-1}}{255} \vee \frac{I_{y,x-1}}{255}\right) \\
\quad\quad\quad\quad\quad F3 \leftarrow \frac{I_{y+1,x+1}}{255} \wedge \frac{I_{y+1,x}}{255} \wedge \frac{I_{y+1,x-1}}{255} \wedge \neg\left(\frac{I_{y-1,x+1}}{255} \vee \frac{I_{y-1,x}}{255} \vee \frac{I_{y-1,x-1}}{255}\right) \\
\quad\quad\quad\quad\quad F4 \leftarrow \frac{I_{y+1,x}}{255} \wedge \frac{I_{y+1,x-1}}{255} \wedge \frac{I_{y,x-1}}{255} \wedge \neg\left(\frac{I_{y,x+1}}{255} \vee \frac{I_{y-1,x+1}}{255} \vee \frac{I_{y-1,x}}{255}\right) \\
\quad\quad\quad\quad\quad F5 \leftarrow \frac{I_{y+1,x-1}}{255} \wedge \frac{I_{y,x-1}}{255} \wedge \frac{I_{y-1,x-1}}{255} \wedge \neg\left(\frac{I_{y+1,x+1}}{255} \vee \frac{I_{y,x+1}}{255} \vee \frac{I_{y-1,x+1}}{255}\right) \\
\quad\quad\quad\quad\quad F6 \leftarrow \frac{I_{y,x-1}}{255} \wedge \frac{I_{y-1,x-1}}{255} \wedge \frac{I_{y-1,x}}{255} \wedge \neg\left(\frac{I_{y+1,x}}{255} \vee \frac{I_{y+1,x+1}}{255} \vee \frac{I_{y,x+1}}{255}\right) \\
\quad\quad\quad\quad\quad F7 \leftarrow \frac{I_{y-1,x-1}}{255} \wedge \frac{I_{y-1,x}}{255} \wedge \frac{I_{y-1,x+1}}{255} \wedge \neg\left(\frac{I_{y+1,x-1}}{255} \vee \frac{I_{y+1,x}}{255} \vee \frac{I_{y+1,x+1}}{255}\right) \\
\quad\quad\quad\quad\quad F8 \leftarrow \frac{I_{y-1,x}}{255} \wedge \frac{I_{y-1,x+1}}{255} \wedge \frac{I_{y,x+1}}{255} \wedge \neg\left(\frac{I_{y,x-1}}{255} \vee \frac{I_{y+1,x-1}}{255} \vee \frac{I_{y+1,x}}{255}\right) \\
\quad\quad\quad\quad\quad \text{if } F1 \vee F2 \vee F3 \vee F4 \vee F5 \vee F6 \vee F7 \vee F8 \\
\quad\quad\quad\quad\quad\quad J_{y,x} \leftarrow 0 \\
\quad\quad\quad\quad\quad\quad \mathrm{stable} \leftarrow 0 \\
\quad\quad\quad\quad\quad J_{y,x} \leftarrow I_{y,x} \quad \text{otherwise} \\
\quad\quad\quad\quad J_{y,x} \leftarrow I_{y,x} \quad \text{otherwise} \\
\quad\quad I \leftarrow J \\
\quad J
\end{array}
$$

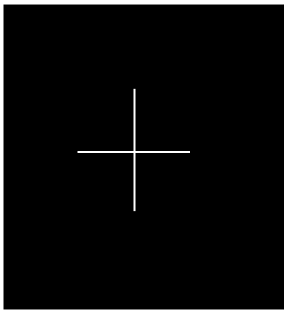
$J := \mathrm{AxeMedian}(I)$

Image originale



I

Image Axe Médian



J

Squelettisation : ***Q4.3. Coder (Python) la squelettisation. L'effet de fourche est-il normal ? Test sur images binaires***

$$
\mathrm{Correctif}(index, indexes, Dx, Dy, I, x, y) \equiv
\left|
\begin{array}{l}
epluch \leftarrow 1 \\
\text{if } index = indexes_1 \\
\quad \left|
\begin{array}{l}
index2 \leftarrow 0 \\
\text{for } i \in 0..7 \\
\quad \left|
\begin{array}{l}
y2 \leftarrow y + 1 + Dy_i \\
x2 \leftarrow x + Dx_i \\
index2 \leftarrow index2 + 2^{i+1} \cdot \dfrac{I_{y2,x2}}{255}
\end{array}
\right. \\
epluch \leftarrow 0 \quad \text{if } index2 = indexes_3
\end{array}
\right. \\
\text{if } index = indexes_2 \\
\quad \left|
\begin{array}{l}
index2 \leftarrow 0 \\
\text{for } i \in 0..7 \\
\quad \left|
\begin{array}{l}
y2 \leftarrow y + Dy_i \\
x2 \leftarrow x + 1 + Dx_i \\
index2 \leftarrow index2 + 2^{i+1} \cdot \dfrac{I_{y2,x2}}{255}
\end{array}
\right. \\
epluch \leftarrow 0 \quad \text{if } index2 = indexes_0
\end{array}
\right. \\
epluch
\end{array}
\right.
$$

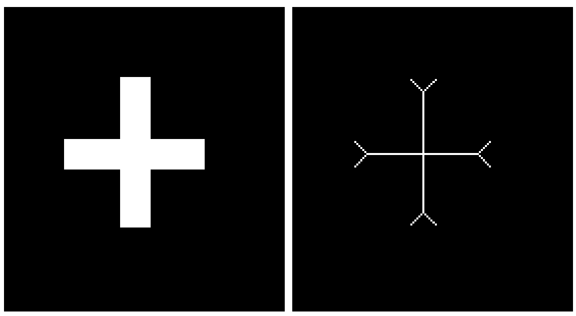
```
Epluchage(I) ≡ | Dx ← (−1  0  1  1  1  0  −1  −1)^T
               | Dy ← (−1  −1  −1  0  1  1  1  0)^T
               | indexes ← (454  496  124  286  248  398  482  62  240  270  60  450  480  30  390  120)^T
               | stable ← 0
               | while stable = 0
               |   | stable ← 1
               |   | for y ∈ 0..rows(I) − 1
               |   |   for x ∈ 0..cols(I) − 1
               |   |     | if I_{y,x} = 255
               |   |     |   | index ← 0
               |   |     |   | for i ∈ 0..7
               |   |     |   |   | y1 ← y + Dy_i
               |   |     |   |   | x1 ← x + Dx_i
               |   |     |   |   | index ← index + 2^(i+1) · I_{y1,x1} / 255
               |   |     |   | epluch ← 0
               |   |     |   | for i ∈ 0..15
               |   |     |   |   if index = indexes_i
               |   |     |   |     | epluch ← 1
               |   |     |   |     | epluch ← Correctif(index, indexes, Dx, Dy, I, x, y)
               |   |     |   |     | break
               |   |     |   | stable ← 0  if epluch = 1
               |   |     |   | J_{y,x} ← 0  if epluch = 1
               |   |     |   | J_{y,x} ← I_{y,x}  otherwise
               |   |     | J_{y,x} ← I_{y,x}  otherwise
               |   | I ← J
               |   | J ← 0
               | I
```

```
Nettoyage(I) ≡ | Dx ← (−1  0  1  1  1  0  −1  −1)^T
               | Dy ← (−1  −1  −1  0  1  1  1  0)^T
               | indexes ← (2  8  32  128  4  16  64  256)^T
               | for y ∈ 0..rows(I) − 1
               |   for x ∈ 0..cols(I) − 1
               |     | if I_{y,x} = 255
               |     |   | index ← 0
               |     |   | for i ∈ 0..7
               |     |   |   | y1 ← y + Dy_i
               |     |   |   | x1 ← x + Dx_i
               |     |   |   | index ← index + (2^(i+1) · I_{y1,x1}) / 255
               |     |   | nettoie ← 0
               |     |   | for i ∈ 0..7
               |     |   |   if index = indexes_i
               |     |   |     | nettoie ← 1
               |     |   |     | break
               |     |   | J_{y,x} ← 0  if nettoie = 1
               |     |   | J_{y,x} ← I_{y,x}  otherwise
               |     | J_{y,x} ← I_{y,x}  otherwise
               | J
```

```
Squelettisation(I) ≡ | for y ∈ 0..rows(I) − 1      Bords noirs
                     |   | I_{y,0} ← 0
                     |   | I_{y,cols(I)−1} ← 0
                     | for x ∈ 0..cols(I) − 1
                     |   | I_{0,x} ← 0
                     |   | I_{rows(I)−1,x} ← 0
                     | J ← Epluchage(I)
                     | K ← Nettoyage(J)
                     | K
```

J := Squelettisation(I)

Image originale Image squelettisée



I J

2. Opérateurs morphologiques (érosion, dilatation) sur images en niveaux de gris

I := READ_IMAGE("coco.bmp")

Nombre de colonnes : Nx := cols(I) Nx = 147

Nombre de lignes : Ny := rows(I) Ny = 195

Dilatation / Erosion en niveaux de gris

Q4.4. : Transposer sous Python dilatation et erosion sur images de luminance (non binaires). Quelle application ?

```
DilatationGris(I) ≡ | for y ∈ 1 .. rows(I) − 2
                    |   for x ∈ 1 .. cols(I) − 2
                    |     | for i ∈ 0 .. 2
                    |     |   for j ∈ 0 .. 2
                    |     |     v[3·i+j] ← I[y+i−1, x+j−1]
                    |     | J[y,x] ← max(v)
                    | J

ErosionGris(I) ≡    | for y ∈ 1 .. rows(I) − 2
                    |   for x ∈ 1 .. cols(I) − 2
                    |     | for i ∈ 0 .. 2
                    |     |   for j ∈ 0 .. 2
                    |     |     v[3·i+j] ← I[y+i−1, x+j−1]
                    |     | J[y,x] ← min(v)
                    | J
```

J := DilatationGris(I) K := ErosionGris(I)

Image originale Image dilatée Image érodée

I J K

Fonctions Bibliothèque :

```
Histo(I) ≡ | for n ∈ 0..255
           |   H_n ← 0
           | for y ∈ 0..rows(I) − 1
           |   for x ∈ 0..cols(I) − 1
           |     | n ← I_{y,x}
           |     | H_n ← H_n + 1
           | H
```

```
RecadrageDyn(I) ≡ | H ← Histo(I)
                  | for i ∈ 0..255
                  |   maxi ← i  if H_i ≠ 0
                  | for i ∈ 255,254..0
                  |   mini ← i  if H_i ≠ 0
                  | Δ ← 255 / (maxi − mini)
                  | for y ∈ 0..rows(I) − 1
                  |   for x ∈ 0..cols(I) − 1
                  |     J_{y,x} ← round[(I_{y,x} − mini)·Δ]
                  | J
```

```
Convol3x3(I, Mask3x3) ≡ | for y ∈ 1..rows(I) − 2
                        |   for x ∈ 1..cols(I) − 2
                        |     | som ← 0
                        |     | for i ∈ 0..2
                        |     |   for j ∈ 0..2
                        |     |     som ← som + round(I_{y+i−1,x+j−1}·Mask3x3_{i,j})
                        |     | som ← 255  if som > 255
                        |     | som ← 0  if som < 0
                        |     | J_{y,x} ← som
                        | J
```

```
Derivation(I, Mask3x3) ≡ | for y ∈ 1..rows(I) − 2
                         |   for x ∈ 1..cols(I) − 2
                         |     | som ← 0
                         |     | for i ∈ 0..2
                         |     |   for j ∈ 0..2
                         |     |     som ← som + I_{y+i−1,x+j−1}·Mask3x3_{i,j}
                         |     | som ← |som|
                         |     | som ← 255  if som > 255
                         |     | som ← 0  if som < 0
                         |     | J_{y,x} ← som
                         | J
```

```
Binarisation(I, seuil) ≡ | for y ∈ 0 .. rows(I) − 1
                         |   for x ∈ 0 .. cols(I) − 1
                         |     | J[y,x] ← 0   if I[y,x] < seuil
                         |     | J[y,x] ← 255   otherwise
                         | J
```

```
T2D(I) ≡ | for y ∈ 0 .. rows(I) − 1
         |   for x ∈ 0 .. cols(I) − 1
         |     J[y,x] ← I[y,x]
         | J
```

```
T1DColonne(I, f) ≡ | for n ∈ 0 .. rows(I) − 1
                   |   g[n] ← f[n]
                   | g
```

```
T1D(I) ≡ | for x ∈ 0 .. cols(I) − 1
         |   | f ← I⟨x⟩
         |   | J⟨x⟩ ← T1DColonne(I, f)
         | return J
```

```
Median(I) ≡ | for y ∈ 1 .. rows(I) − 2
            |   for x ∈ 1 .. cols(I) − 2
            |     | for i ∈ 0 .. 2
            |     |   for j ∈ 0 .. 2
            |     |     v[3·i+j] ← I[y+i−1, x+j−1]
            |     | vtri ← sort(v)
            |     | J[y,x] ← vtri[4]
            | J
```

```
Nagao(I) ≡ | for y ∈ 2 .. rows(I) − 3
           |   for x ∈ 2 .. cols(I) − 3
           |     | for j ∈ 0 .. 2
           |     |   for i ∈ 0 .. 2
           |     |     | for m ∈ 0 .. 2
           |     |     |   for k ∈ 0 .. 2
           |     |     |     Tmp[m·3+k] ← I[y+j+m−2, x+i+k−2]
           |     |     | Moy[3·j+i] ← mean(Tmp)   if i·j ≠ 1
           |     |     | Moy[3·j+i] ← 0   otherwise
           |     |     | Var[3·j+i] ← max(Tmp) − min(Tmp)   if i·j ≠ 1
           |     |     | Var[3·j+i] ← 255   otherwise
           |     | Mini ← min(Var)
           |     | for i ∈ 0 .. 8
           |     |   W[y−2, x−2] ← Moy[i]   if Var[i] = Mini
           | W
```

$$D_1 \equiv \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad D_2 \equiv \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad D_3 \equiv \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad D_4 \equiv \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$D_5 \equiv \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad D_6 \equiv \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix} \quad D_7 \equiv \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \quad D_8 \equiv \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

```
Chanda(I) ≡ | for y ∈ 2 .. rows(I) − 3
            |   for x ∈ 2 .. cols(I) − 3
            |     | SDG ← 0
            |     | SG ← 0
            |     | for k ∈ 1 .. 8
            |     |   | S ← 0
            |     |   | for j ∈ 0 .. 4
            |     |   |   for i ∈ 0 .. 4
            |     |   |     S ← S + [ I[y+j−2, x+i−2] · (D[k])[j, i] ]
            |     |   | S ← S / 6
            |     |   | S2 ← 0
            |     |   | for j ∈ 0 .. 4
            |     |   |   for i ∈ 0 .. 4
            |     |   |     S2 ← S2 + [ I[y+j−2, x+i−2] · (D[k])[j, i] ]^2
            |     |   | S2 ← S2 / 6
            |     |   | D ← S − I[y, x]
            |     |   | G ← ( |S2 − S^2| )^(1/2)
            |     |   | SDG ← SDG + D·G
            |     |   | SG ← SG + G
            |     | J[y, x] ← I[y, x] − SDG / SG
            |     | J[y, x] ← 0 if J[y, x] < 0
            |     | J[y, x] ← 255 if J[y, x] > 255
            | J
```

$$D_1 \equiv \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad D_2 \equiv \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad D_3 \equiv \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad D_4 \equiv \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$D_5 \equiv \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad D_6 \equiv \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix} \quad D_7 \equiv \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \quad D_8 \equiv \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

```
ChandaV(I) ≡ | for y ∈ 2 .. rows(I) − 3
             |   for x ∈ 2 .. cols(I) − 3
             |     | SDG ← 0
             |     | SG ← 0
             |     | for k ∈ 1 .. 8
             |     |   | S ← 0
             |     |   | for j ∈ 0 .. 4
             |     |   |   for i ∈ 0 .. 4
             |     |   |     S ← S + [I_(y+j−2, x+i−2) · (D_k)_(j,i)]
             |     |   | S ← S / 6
             |     |   | S2 ← S^2 / 6
             |     |   | D ← S − I_(y,x)
             |     |   | G ← (|S2 − S^2|)^(1/2)
             |     |   | SDG ← SDG + D·G
             |     |   | SG ← SG + G
             |     | J_(y,x) ← I_(y,x) − SDG / SG
             |     | J_(y,x) ← 0 if J_(y,x) < 0
             |     | J_(y,x) ← 255 if J_(y,x) > 255
             | J
```