Compte Rendu TP4
Détection de contours d'une image avec utilisation du gradient (1er ordre)

Ivan Lejeune

14 février 2024

Table des matières

1 Création de la carte de gradient d'une image

1.1 Programme

On commence par créer le programme `norme_gradient.cpp` qui en chaque point d'une image calcule les gradients horizontaux et verticaux, puis retourne la norme du gradient. On crée ensuite une nouvelle image avec les valeurs de la norme du gradient. Enfin, on comparera les profils d'une ligne de l'image originale et de l'image de la norme du gradient.

L'essentiel du code est le suivant :

```
// calcul du gradient pour chaque pixel
for (int i = 0; i < nH; i++){
    for (int j = 0; j < nW; j++) {
        if (i == 0 || i == nH - 1 || j == 0 || j == nW - 1) {
            // ignore edges
            ImgOut[i * nW + j] = 0;
        } else {
            int dx = ImgIn[(i + 1) * nW + j] - ImgIn[(i - 1) * nW + j];
            int dy = ImgIn[i * nW + j + 1] - ImgIn[i * nW + j - 1];
            ImgOut[i * nW + j] = sqrt(dx * dx + dy * dy);
        }
    }
}
```

FIGURE 1 – Code de norme_gradient.cpp

La transformation de l'image donne :

FIGURE 2 – Image originale

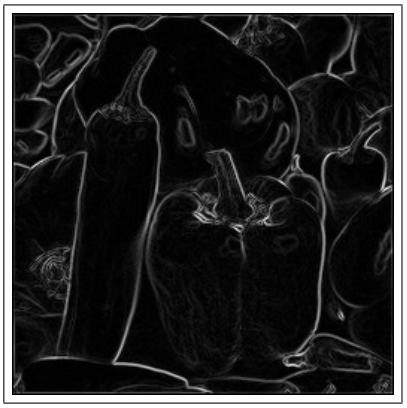

FIGURE 3 – Image de la norme

On peut voir que les contours de l'image sont bien mis en évidence.

1.2 Comparaison des profils

On compare les profils d'une ligne de l'image originale et de l'image de la norme du gradient. On peut voir que les contours sont bien mis en évidence sur l'image de la norme du gradient.

La différence entre les deux profils est bien visible sur la colonne 80 :

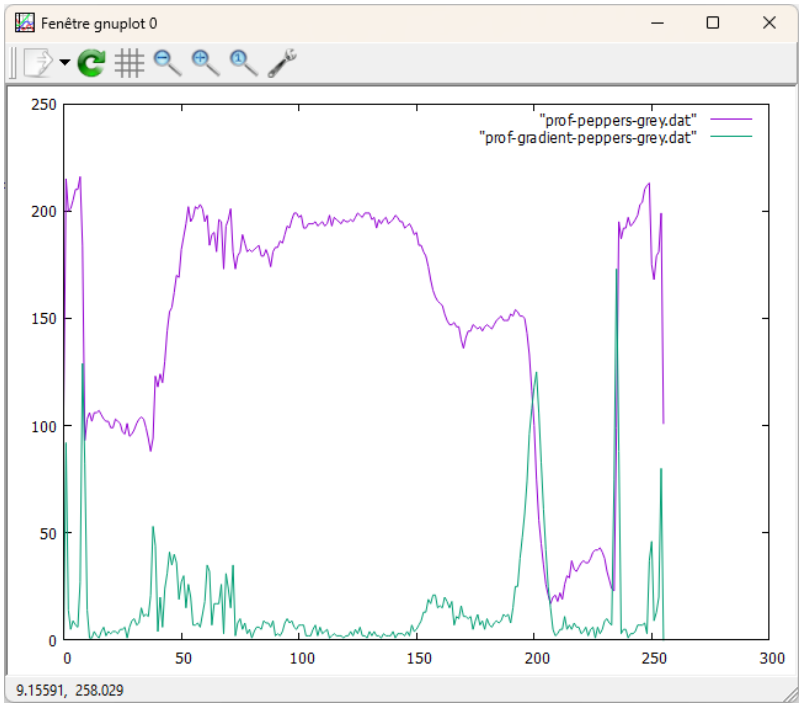

FIGURE 4 – Comparaison des profils

2 Extraction des maxima locaux par seuillage

On va extraire des maxima locaux de l'image de la norme du gradient par seuillage. On crée un programme `maxima_locaux.cpp` qui prend en entrée une image de la norme du gradient et un seuil. Le programme retourne une image binaire avec les maxima locaux.

L'essentiel du code est le suivant :

```
// find local maxima in regard to S
// if the norm of gradient is greater than S, then it's a local maxima
for (int i = 0; i < nH; i++){
    for (int j = 0; j < nW; j++) {
        if (i == 0 || i == nH - 1 || j == 0 || j == nW - 1) {
            // ignore edges
            ImgOut[i * nW + j] = 0;
        } else {
            int dx = ImgIn[(i + 1) * nW + j] - ImgIn[(i - 1) * nW + j];
            int dy = ImgIn[i * nW + j + 1] - ImgIn[i * nW + j - 1];
            int norm = sqrt( x: dx * dx + dy * dy);
            if (norm > S) {
                ImgOut[i * nW + j] = 255;
            } else {
                ImgOut[i * nW + j] = 0;
            }
        }
    }
}
```

FIGURE 5 – Code de maxima_locaux.cpp

On applique le programme sur l'image de la norme du gradient avec 3 seuils différents, ici 80, 60 et 30 :

FIGURE 6 – Image avec un seuil de 80

FIGURE 7 – Image avec un seuil de 60

FIGURE 8 – Image avec un seuil de 30

On peut voir que plus le seuil est bas, plus les maxima locaux sont nombreux, et donc plus de contours sont détectés.

3 Seuillage par hystérésis des maxima locaux

On va maintenant appliquer un seuillage par hystérésis sur l'image de la norme du gradient. On crée un programme `hysteresis.cpp` qui prend en entrée une image de la norme du gradient et deux seuils. Le programme retourne une image binaire avec les contours détectés.

L'essentiel du code est le suivant :

```cpp
// first time
for (int i = 0; i < nH; i++){
    for (int j = 0; j < nW; j++) {
        if (i == 0 || i == nH - 1 || j == 0 || j == nW - 1) {
            // ignore edges
            ImgOut[i * nW + j] = 0;
        } else {
            int dx = ImgIn[(i + 1) * nW + j] - ImgIn[(i - 1) * nW + j];
            int dy = ImgIn[i * nW + j + 1] - ImgIn[i * nW + j - 1];
            int norm = sqrt( X: dx * dx + dy * dy);
            if (norm > S) {
                ImgTemp[i * nW + j] = 255;
            } else if (norm < T) {
                ImgTemp[i * nW + j] = 0;
            } else {
                ImgTemp[i * nW + j] = 128;
            }
        }
    }
}
```

FIGURE 9 – Code de hysteresis.cpp (1/2)

```cpp
// second time
for (int i = 0; i < nH; i++){
    for (int j = 0; j < nW; j++) {
        if (i == 0 || i == nH - 1 || j == 0 || j == nW - 1) {
            // ignore edges
            ImgOut[i * nW + j] = 0;
        } else {
            if (ImgTemp[i * nW + j] == 128) {
                int found = 0;
                for (int k = -1; k <= 1; k++) {
                    for (int l = -1; l <= 1; l++) {
                        if (ImgTemp[(i + k) * nW + j + l] == 255) {
                            found = 1;
                        }
                    }
                }
                if (found) {
                    ImgTemp2[i * nW + j] = 255;
                } else {
                    ImgTemp2[i * nW + j] = 0;
                }
            } else {
                ImgTemp2[i * nW + j] = ImgTemp[i * nW + j];
            }
        }
    }
}
```

FIGURE 10 – Code de hysteresis.cpp (2/2)

On applique le programme sur l'image de la norme du gradient avec trois paires de seuils différents, ici (80, 60), (60, 30) et (30, 10) :

FIGURE 11 – Image avec des seuils de 80 et 60

FIGURE 12 – Image avec des seuils de 60 et 30

FIGURE 13 – Image avec des seuils de 30 et 10

On peut voir que plus les seuils sont bas, plus de contours sont détectés.

4 Prétraitement par filtrage

On recommence les étapes précédentes avec une image prétraitée par filtrage.

4.1 Filtre moyenneur

On crée un programme `filtre_moyenneur.cpp` qui prend en entrée une image et un masque de filtrage. Le programme retourne une image filtrée.

L'essentiel du code est le suivant :

```
// apply the filter
for (int i = 0; i < nH; i++){
    for (int j = 0; j < nW; j++) {
        int sum = 0;
        int count = 0;
        for (int k = -n; k <= n; k++) {
            for (int l = -n; l <= n; l++) {
                if (i + k >= 0 && i + k < nH && j + l >= 0 && j + l < nW) {
                    sum += ImgIn[(i + k) * nW + j + l];
                    count++;
                }
            }
        }
        ImgOut[i * nW + j] = sum / count;
    }
}
```

FIGURE 14 – Code de filtre_moyenneur.cpp

On applique le programme sur l'image originale avec un masque de 3x3 :

FIGURE 15 – Image originale

FIGURE 16 – Image filtrée

4.2 Filtre gaussien

On crée un programme `filtre_gaussien.cpp` qui prend en entrée une image et un masque de filtrage. Le programme retourne une image filtrée.

L'essentiel du code est le suivant :

```
// apply the filter
for (int i = 0; i < nH; i++){
    for (int j = 0; j < nW; j++) {
        float sum = 0;
        float count = 0;
        for (int k = -n; k <= n; k++) {
            for (int l = -n; l <= n; l++) {
                if (i + k >= 0 && i + k < nH && j + l >= 0 && j + l < nW) {
                    float g = exp( x: -(k * k + l * l) / (2 * n * n));
                    sum += g * ImgIn[(i + k) * nW + j + l];
                    count += g;
                }
            }
        }
        ImgOut[i * nW + j] = sum / count;
    }
}
```

FIGURE 17 – Code de filtre_gaussien.cpp

On applique le programme sur l'image originale avec un masque de 3x3 :

FIGURE 18 – Image originale

FIGURE 19 – Image filtrée

4.3 Résultats

On recommence les étapes précédentes avec les images filtrées par les deux filtres.
On applique le programme `norme_gradient.cpp` sur les images filtrées par les deux filtres :

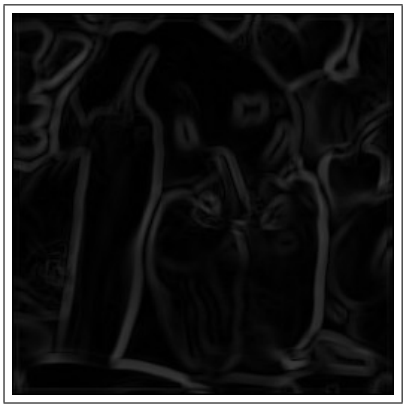

FIGURE 20 – Image de la norme avec filtre moyenneur

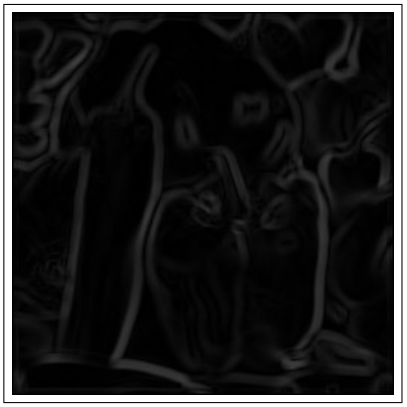

FIGURE 21 – Image de la norme avec filtre gaussien

On applique le programme `maxima_locaux.cpp` sur les images de la norme avec les deux filtres :

FIGURE 22 – Image avec filtre moyenneur et seuil de 20

FIGURE 23 – Image avec filtre gaussien et seuil de 20

On applique le programme `hysteresis.cpp` sur les images de la norme avec les deux filtres :

FIGURE 24 – Image avec filtre moyenneur et seuils de 20 et 10

FIGURE 25 – Image avec filtre gaussien et seuils de 20 et 10

5 Conclusion

On a vu que les contours d'une image peuvent être détectés avec le gradient de l'image. On a aussi vu que le filtrage de l'image peut améliorer la détection des contours. On a donc pu comparer les résultats de la détection de contours avec et sans filtrage.