

Initiation au traitement des images

Détection automatique Mesure de la surface des zones urbaines des images satellitaires

Cet atelier propose de mesurer l'emprise urbaine sur l'espace rural à partir d'une détection automatique des zones urbaines dans des images satellitaires. Les images sont extraites directement de Google Maps (<http://maps.google.fr/>) et concernent la région de Creully dans le Calvados. Le travail pratique consiste à créer un programme informatique qui va permettre de détecter automatiquement les zones urbaines dans les images.



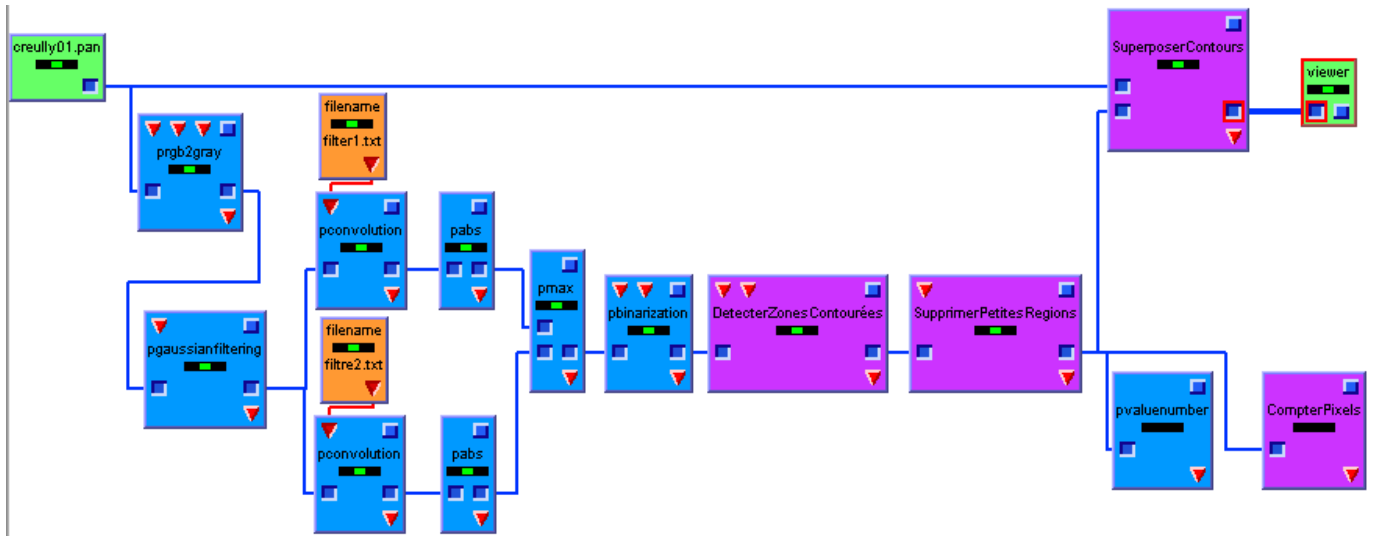
Les images que l'on va traiter présentent les caractéristiques suivantes :

Type	image couleur, 1 pixel est donc codé sur 3 octets (1 octet pour coder l'intensité en rouge, 1 octet pour coder l'intensité en vert et 1 octet pour coder l'intensité en bleu)
Taille	940 colonnes x 665 lignes

Échelle de prise de vue	pixel → 5,932 m sur le terrain
--------------------------------	--------------------------------

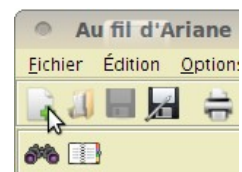
Exercice : Détection de zones urbaines

Le programme à créer doit permettre de localiser automatiquement les régions correspondant à des zones urbaines. En suivant les étapes ci-après, vous devez réaliser un programme informatique ressemblant au graphe ci-dessous :



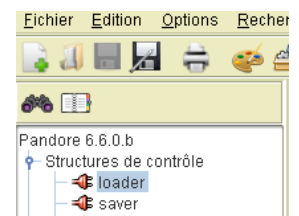
1/ Création d'un espace de travail

- ❑ Cliquer sur le menu **Fichier** puis **Nouveau**, ou cliquer directement sur l'icône **Nouveau** de la barre d'outils. Cela doit créer un espace de travail gris.

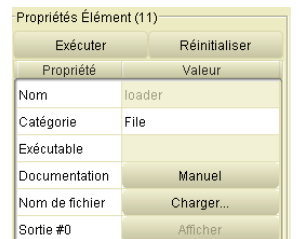


2/ Chargement d'une image dans l'espace de travail

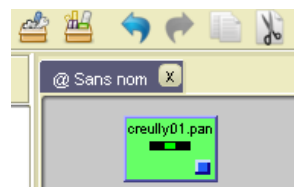
- ❑ Déplacer la structure de contrôle **loader** dans l'espace de travail par un cliquer-glisser.



- ❑ Dans la fenêtre des propriétés à droite de l'interface Ariane, cliquer sur le bouton '**Charger...**'.
- ❑ Ouvrir le dossier **images-aeriennes** sur la clé ENSICAEN.
- ❑ Sélectionner l'image **creully01.pan**



- ❑ Appuyer sur le **bouton vert d'exécution** de la barre d'outils. La signalétique au centre du **loader** doit passer au vert. *Si elle passe au rouge, c'est que le fichier n'a pas été correctement choisi. Il faut recommencer l'opération de sélection.*

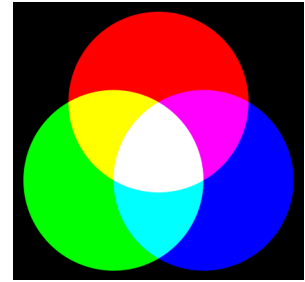


3/ Transformer l'image couleur en image de niveaux de gris

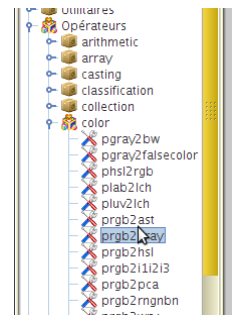
Observation : La couleur n'est pas une information discriminante pour repérer les zones urbaines. Les toits des maisons peuvent être de n'importe quelle couleur. Nous allons donc n'utiliser que l'intensité lumineuse reçue en chaque point de l'image (de clair à sombre) : la luminance.

Pour créer une image de luminance à partir d'une image couleur, on utilise simplement la moyenne des trois composantes rouge, vert et bleu. On se base ici sur le fait que la lumière blanche contient toutes les couleurs avec la même proportion. Donc, pour chaque pixel de l'image couleur, la valeur de luminance correspondante est égale à :

$$\text{luminance} = (1 \times \text{rouge} + 1 \times \text{vert} + 1 \times \text{bleu}) / 3.$$



- Dans l'explorateur situé à gauche de l'interface Ariane, déplacer l'opérateur **color::prgb2gray** sur l'espace de travail.
- Relier **prgb2gray** au **loader** en tirant une ligne fictive entre le plot de sortie du **loader** et le plot d'entrée de **prgb2gray** par *cliquer-glisser*.



- Cliquer sur **prgb2gray**.
- Dans la fenêtre de propriétés située à droite de l'interface, mettre les valeurs : paramètre \$0=1, paramètre \$1=1 et paramètre \$2=1.
- Exécuter l'opérateur à partir du bouton vert dans la barre d'outil.
- Afficher le résultat en appuyant sur le bouton Afficher de la sortie #0 dans la fenêtre de propriétés.

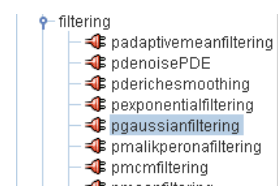
Propriétés Element (3)	
Exécuter	Réinitialiser
Propriété	Valeur
Nom	
Catégorie	color
Exécutable	prgb2gray
Documentation	Manuel
Paramètre \$0	1
Paramètre \$1	1
Paramètre \$2	1
Entrée #0	Afficher
Sortie #0	Afficher
Valeur de retour	-


4/ Lissage de l'image initiale

Observation : Les images issues de Google Maps présentent quelques irrégularités dans les valeurs de couleur. Cela est dû aux défauts inhérents des systèmes d'acquisition d'images et de l'algorithme de stockage des images (ici JPEG). Si l'on fait un zoom assez fort, on verra que dans une zone d'image qui semble uniforme, en fait les valeurs de pixels proches ont des valeurs différentes.

On va chercher à diminuer ces irrégularités en appliquant un opérateur de **lissage** qui a le même effet sur l'image que lorsque l'on fronce les yeux en regardant l'image (cela a pour effet d'atténuer les irrégularités).

- Réafficher l'image initiale et zoomer par 4 sur une zone qui semble homogène (Menu View::Zoom 400% du logiciel de visualisation d'images).
- Constaté les irrégularités des valeurs de pixel.



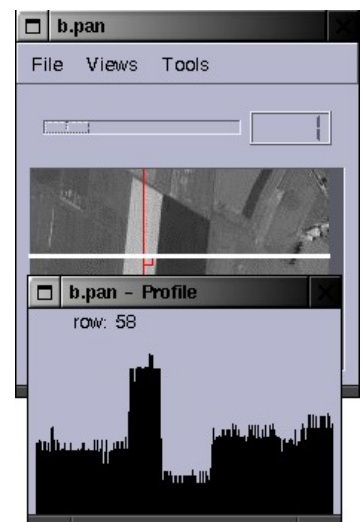
- ❑ Déplacer l'opérateur **filtering::p gaussianfiltering** de l'explorateur situé à gauche de l'interface Ariane sur l'espace de travail.
- ❑ Dans la fenêtre des propriétés située à droite de l'interface Ariane mettre la valeur du paramètre \$0=0.5, qui correspond à une intensité de lissage faible.
- ❑ Relier **p gaussianfiltering** au **loader** en tirant une ligne fictive entre le plot de sortie du loader et le plot d'entrée de **p gaussianfiltering**.
- ❑ Exécuter l'opérateur en appuyant sur le **bouton vert d'exécution** .
- ❑ Afficher l'image de sortie de l'élément en cliquant sur le bouton Afficher de la sortie #0 dans la fenêtre des propriétés.
- ❑ Zoomer par 4 sur une zone qui semble homogène. Constater que maintenant les valeurs sont plus homogènes.

Propriétés Élément (17)	
Exécuter	Réinitialiser
Propriété	Valeur
Nom	
Catégorie	filtering
Exécutable	p gaussianfiltering
Documentation	Manuel
Paramètre \$0	0.5
Entrée #0	Afficher
Sortie #0	Afficher
Valeur de retour	-

5/ Détecter les bords des maisons

Observation : On cherche maintenant à détecter les zones urbaines. Pour cela, on peut remarquer que les zones urbaines sont les régions de l'image qui contiennent le plus de contours, entre autre les bords des maisons, les bords des rues, alors que les autres objets de l'image sont moins perturbés.

On perçoit les bords d'un objet parce qu'il y a un brusque changement de la valeur d'intensité de part et d'autre du bord de l'objet. Par exemple dans l'image ci-contre, un champ clair se distingue d'un champ voisin plus sombre par un changement dans la fonction d'intensité lumineuse.



Calcul la dérivée d'une image

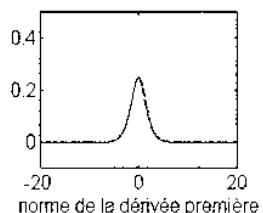
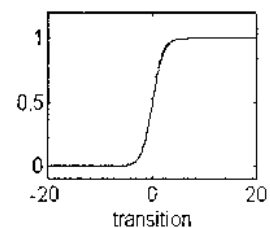
Pour rechercher les bords des objets, on va utiliser la **dérivation**, en considérant le contenu d'une image comme une fonction mathématique à 2 dimensions. La dérivée d'une fonction permet de détecter l'endroit où il y a un changement dans la progression des valeurs le long de la fonction. Chaque valeur non nulle de la dérivée indique un changement dans la suite des valeurs qui est de plus proportionnelle à l'amplitude du changement. Tant que la suite des valeurs de l'image est constante, la valeur de la dérivée est nulle. L'exemple à droite montre que la dérivée permet de localiser l'endroit où une courbe possède une inflexion.

La formulation continue de la dérivée en un point en fonction de ses voisins est donnée par :

$$f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a-h)}{2 \times h}$$

L'image étant une représentation 2D, on utilise les dérivées partielles en X et en Y. On calcule donc la dérivée en X pour chaque point, puis la dérivée en Y pour les mêmes points et on ne garde que le maximum de la valeur absolue de ces deux valeurs pour calculer l'amplitude de la dérivée.

La formulation discrète applicable aux images permettant de calculer



l'amplitude de la dérivée s'écrit :

$$image'(x) = \frac{image(x+1) - image(x-1)}{2}$$
 , où $image(x+1)$ est la valeur de l'image au point $x+1$, et $image'(x)$ la valeur de la dérivée au point x .

Exercice de calcul

Dans le cas d'une image, nous ne disposons pas de la forme algébrique de la fonction, mais par contre nous disposons de ses valeurs en chaque point. Le calcul de la dérivée de l'image se fait donc en chaque point x , à partir de la valeur de ses voisins immédiats :

$$image'(x) = \frac{image(x+1) - image(x-1)}{2}$$

- Aller en dernière page et calculer à la main les valeurs de la dérivée en x .
- Dessiner la courbe résultante sous la courbe initiale.

Application au programme en cours

Avec Ariane, le calcul de la dérivée d'une image se fait par deux **convolutions** de l'image avec un **filtre** pour le calcul de la dérivée en X, et un **filtre** pour la dérivée en Y. La **convolution** consiste à appliquer le filtre sur chaque pixel successivement et à remplacer la valeur du pixel par la somme pondérée, avec les coefficients du filtre, des voisins du pixel (de la même manière que ce que vous venez de faire à la main sur l'exemple précédent).

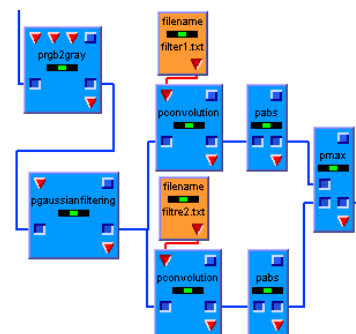
Le premier filtre permet de détecter les variations verticales :

-1	0	1
----	---	---

Il correspond à l'expression de la dérivée selon l'axe X

$$image'(x, y) = \frac{-image(x-1, y) + image(x+1, y)}{2}$$

- Créer un fichier texte avec « Bloc Note » qui contient les 2 lignes suivantes :
1*3
-1 0 +1
- Ajouter l'opérateur **arithmetics::pconvolution** sur l'espace de travail.
- Ajouter la structure de contrôle **filename** sur l'espace de travail.
- Charger le fichier construit avec « Bloc Note » comme paramètre du **filename**.
- Relier la valeur de sortie de filename au paramètre de **pconvolution** comme sur le schéma page 2.
- Relier l'entrée de cet élément à la sortie de l'élément **pgaussianfiltering**.



Le second filtre permet de détecter les variations horizontales :

-1
0
1

et correspond à l'expression de la dérivée selon l'axe Y

$$image'(x, y) = \frac{-image(x, y-1) - image(x, y+1)}{2}$$

Finalement, la valeur d'amplitude finale est prise comme le **maximum** de la **valeur absolue** des valeurs correspondantes dans les deux images.

- Créer un fichier texte avec « Bloc Note » qui contient les 2 lignes suivantes :

3*1

-1 0 +1

- Ajouter l'opérateur **arithmetic::pconvolution** sur l'espace de travail.
- Ajouter la structure de contrôle **filename** sur l'espace de travail.
- Charger le fichier construit avec « Bloc Note » comme paramètre du **filename**.
- Relier la valeur de sortie de **filename** au paramètre de **pconvolution** comme sur le schéma page 2.
- Relier l'entrée de cet élément à la sortie de l'élément **pgaussianfiltering**.
- Ajouter un opérateur **arithmetic::pabs** après chaque **pconvolution**.
- Ajouter l'opérateur **pmax** et le relier aux deux opérateurs **pabs**.
- Exécuter et afficher l'image de sortie.

6/ Sélection des vrais bords d'objet

Observation : Quand on utilise la dérivée d'une image réelle qui présente beaucoup de défaut d'homogénéité, il y a beaucoup de variations détectées, il faut donc sélectionner celles qui sont dues à de vrais bords d'objet. Pour les images utilisées ici, on considère qu'une valeur d'amplitude inférieure à 30 ne présente pas une différence assez marquée pour représenter un vrai bord d'objet. Nous allons donc les éliminer.

- Ajouter l'élément **thresholding::pbinarization**.
- Mettre les valeurs de paramètre \$0=30 et \$1=255.
- Relier à l'opérateur **pabs** puis l'exécuter.

7/ Localisation des zones à forte densité de contours

Observation : Pour délimiter les zones à forte densité de contours, une méthode consiste à dilater les contours de façon à regrouper ceux qui sont proches dans des régions compactes. On estime que deux

maisons proches ont des bords à une distance de moins de 5 pixels dans l'image. Il suffit donc de dilater les contours d'au moins 3 pixels pour fusionner deux contours ayant une distance ≤ 5 pixels.

Après cette étape, les zones de forte densité de contours sont donc regroupées dans des régions, qu'il suffit alors de sélectionner à partir de leur taille.

Il faut ensuite supprimer les trous à l'intérieur de ces régions pour avoir des zones compactes. Toutefois, on ne supprime que les trous qui ont une surface inférieure à 10x10 pixels. Les trous plus grands correspondent en général à des espaces verts à l'intérieur de la zone urbaine qu'il ne faut pas agglomérer à la zone urbaine correspondante.

2 contours proches



Dilatation des contours de taille 5



Dilatation des contours de taille 10. Ils ne forment alors plus qu'un contour unique.

- Ajouter la routine **selection::DetecterZonesContrastées**.
- Mettre les valeurs \$0=5 et \$1=10 correspondant respectivement à la distance minimum entre deux bords de maisons (5 pixels) et la taille minimum de ce qui peut être considéré comme un espace vert à l'intérieur d'une zone urbaine (10x10 pixels).
- Exécuter et afficher l'image de sortie.

8/ Suppression des petites zones

Observation : Les trop petites régions ne correspondent pas à des zones urbaines. On va donc les éliminer sur un critère de taille. On considère qu'il ne peut pas y avoir de zone urbaine inférieure à 90 m x 90 m sur le terrain.

- **Calcul** : Pour pouvoir continuer le traitement, il est nécessaire de calculer la valeur que l'on va utiliser comme paramètre pour l'opérateur de sélection des régions sur la valeur de surface. On cherche ici la valeur de la **longueur minimale** des zones urbaines en nombre de pixels.
- La longueur minimale sur le terrain est de 90 m.
 1. L'échelle de prise de vue est de 1 pixel pour 5,932 m sur le terrain. *Combien font 90 m sur le terrain en pixels dans l'image ?*
 2. Reporter le calcul à effectuer et la valeur de surface trouvée en dernière page.

Remarque : vous disposez d'une calculatrice sur l'ordinateur dans le menu "Programmes::accessoires" de Windows.

- Ajouter la routine **selection::SupprimerPetitesRegions**.
- Affecter la valeur trouvée précédemment au paramètre \$0 de la routine **SupprimerPetitesRegions**.
- Exécuter (cela peut prendre du temps en fonction de la valeur que vous avez trouvée).
- Afficher l'image de sortie et vérifier.

9/ Visualisation des résultats

Pour bien visualiser les résultats, on va superposer les frontières des zones urbaines trouvées à l'image initiale.

- Déplacer la routine **visualisation::SuperposerContours** sur l'espace de travail.
- Relier la sortie de **loader** à l'entrée n° 1 de **SuperposerContours** et la sortie de **SupprimerPetitesRegions** à l'entrée n°2. Mettez la valeur de *mask* à 3.
- Ajouter un élément **Structures de contrôle::viewer** en fin de chaîne.
- Exécuter et vérifier la cohérence des résultats.

10/ Calculer le rapport de surface

Une fois les zones urbaines repérées dans des régions, il suffit de calculer la taille totale de ces régions et de rapporter ce nombre au nombre de pixels total de l'image pour avoir le taux d'emprise des zones urbaines sur le paysage.

L'opération suivante a pour but de calculer la surface des régions en nombre de pixels :

- Ajouter l'opérateur **imagefeatureextraction::pvaluenumber** pour calculer le nombre de pixels dans les zones urbaines (pixels ayant une valeur non nulle).
- Le relier à **SupprimerPetitesRegions** (voir figure page 2).
- Sélectionner l'opérateur puis exécuter à partir du bouton vert.
- Pour connaître la surface en pixels, il suffit de regarder la " valeur de retour " affichée dans la fenêtre des propriétés de l'opérateur.
- Reporter la valeur dans le tableau des résultats en dernière page, et calculer le rapport entre la surface occupée par les zones urbaines et la surface totale de l'image.

11/ Changer d'image et relancer l'exécution

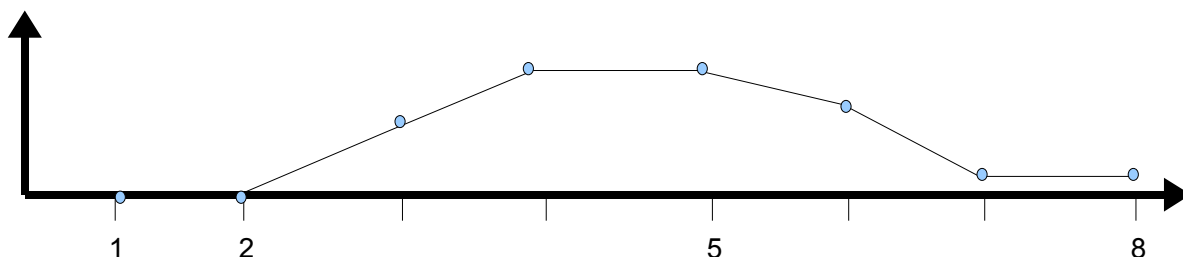
- Relancer ce même graphe sur les trois autres images du dossier image. Pour cela, il suffit de cliquer sur le **loader** et de changer le nom du fichier image.
- À chaque fois reporter les résultats dans le tableau en dernière page.

Feuille résultat

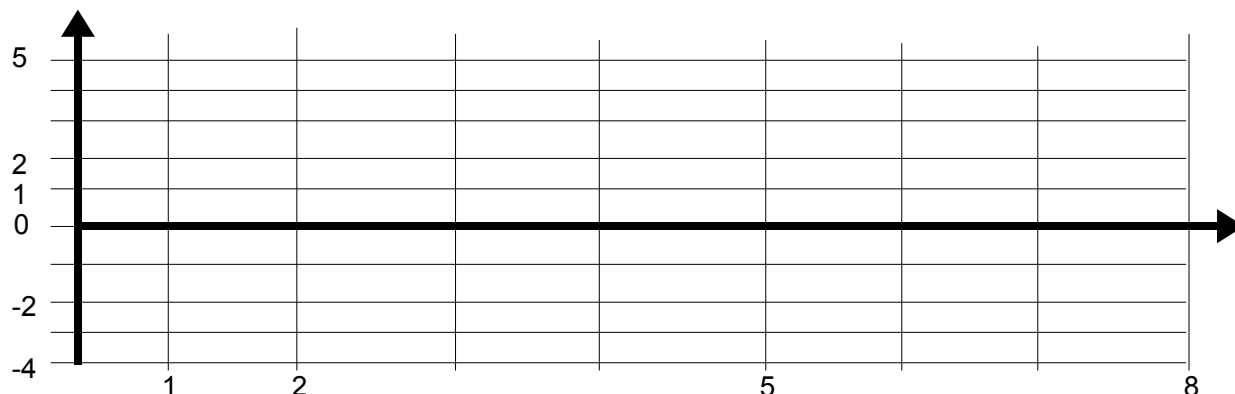
1/ Calcul de la dérivée d'une ligne d'image

x	1	2	3	4	5	6	7	8
image(x)	0	0	5	10	10	7	1	1
image'(x)	0							0

Courbe correspondant à image(x) :



Dessiner la courbe correspondant à image'(x) :



2/ Valeur du paramètre de suppression des petites zones

90m sur le terrain = pixels sur l'image

3/ Résultats du programme sur les images

1 pixel sur l'écran = km² sur le terrain

Images	Surface des zones urbaines (pixels)	Surface des zones urbaines (km²)	Pourcentage d'occupation
creully01.p an			
creully02.p an			
creully03.p an			

