Projet Cvlib

Implémentation OpenCV pour Pure Data

(environnement Flext)

Jean-Yves Gratius, Mars 2010

jy.gratius.free.fr

Introduction

OpenCV est une bibliothèque de fonctions C++ fournissant nombre d'outils pour la vision et la reconnissance d'images par ordinateur.

Le projet Cvlib vise à implémenter un certain nombre de ces fonctions dans l'environnement de programmation graphique Pure Data.

Le hangar : http://hangar.org/wikis/lab/doku.php?id=start:puredata_opencv propose un certain nombre d'externals compatibles avec les deux principales bibliothèques graphiques de Pure Data, pdp et GEM .
Ces externals, dérivés des exemples de la documentation openCV, permettent de réaliser des opérations très utiles (« *face detection* », etc...), mais ne sont pas disponibles sous Windows.

D'un autre côté, cv.jit.(http://www.iamas.ac.jp/~jovan02/cv/), disponible pour Max MSP semble permettre des opérations d'un niveau plus élémentaire sur opencv.

Actuellement, pour Pure Data, il n'existe pas de solution multiplateforme (windows / mac / linux) d'implémentation d'OpenCV à la fois haut et bas niveau.

La bibliothèque d'externals cvlib vise à remédier à cela.

La bibliothèque d'externals cvlib

Les caractéristiques de la bibliothèque d'externals Cvlib sont les suivantes.

- Multiplateformes. Pour l'instant, seule une version Windows est proposée.
- Portage pour la plateforme Windows des objets existants pix_opencv_xxxxxxx proposés par hangar.org, jusqu'à présent seulement accessible aux utilisateurs de linux / OSX.
- Cvlib se veut un outil pour le prototypage rapide dans Pure Data d'algorithmes basés sur OpenCV. Par conséquent, le coeur de cette bibliothèque est constitué d'objets bas niveaux reprenant les noms des fonctions de base d'OpenCV qu'ils implémentent.

Compatibilité avec GEM

Les objets de la bibliothèque cvlib peuvent fonctionner dans deux modes différents :

- Un mode compatible avec GEM, la bibliothèque graphique de référence pour PureData sous Windows.
- Un mode natif OpenCV, dans lequel les objets communiquent entre eux en s'échangeant des index pointant vers des structurs de données standard d'OpenCV (IplImage, cvRect, etc....)

  Les objets acceptent indifféremment les deux modes et permettent facilement le passage d'un mode à l'autre. Il est possible de mixer les deux modes dans un même patch.

Le framework associé à cvlib

Conjointement à cvlib, nous proposons un framework de développement multiplateformes pour faciliter l'implémentation dans Pure Data des fonctionnalités de OpenCV. Les classes de base C++ de cvlib ont pour but de faciliter la création de nouveaux objets, avec gestion intégrée des messages et données (inlet, outlet, import/export GEM)

Nous avons choisi de développer ce framework dans l'environnement FLEXT développé par Thomas Grill (http://puredata.info/Members/thomas/flext/) qui présente deux avantages principaux :

- Cet environnement permet d'écrire en C++ (comme OpenCV), contrairement à Pure Data qui est écrit en C.
- Cet environnement est multiplateforme (tout comme OpenCV), ce qui facilitera le port vers les différentes plateformes. Un port vers MAX est théoriquement envisageable.

Structure des données

En mode natif openCV, Cvlib utilise les structures de données d'OpenCV, « empaquetées » dans un format conteneur spécifique qui permet de partager des données entre différents objets dans pure-data. Le principe est de transmettre un identificateur de type (exemple : cvImage, cvContours) suivi d'un index qui fait référence à un pointeur interne indiquant où se trouve la structure de données OpenCV correspondante.

Ce principe permet de partager différents types de données (et pas seulement des images), dans un contexte sécurisé puisque le type est non seulement explicitement spécifié dans les messages qui transitent entre objets, mais est aussi revérifié en interne dans la structure de stockage elle-même.

Cela doit éviter les erreurs de type « pointeur fou » ou de mélange entre types de données en cas d'erreurs dans le patch.

Pourquoi ne pas utiliser la structure de données de GEM ou pdp ?

Le choix de ce format conteneur spécifique est justifié par le fait que OpenCV propose de nombreux types de données et qu'il est souhaitable d'accéder à toute la puissance de traitement de cet outil en adoptant ses formats.

Par exemple, comparée à la structure ImageStruct de GEM, la structure IplImage de OpenCV permet de définir un COI (channel of interest) ou une ROI (region of interest), données très utiles pour le travail sur les images.

Cependant, afin d'assurer la compatibilité avec GEM, les objets créés avec le framework convertissent les messages de type *gemstate <pointer><pointer>* issus d'objets pix. Dans ce cas, ils effectuent également la conversion vers Gem en sortie.

Il est possible de mixer les deux types de formats sur le même objet.

La compatibilité avec pdp n'est pas envisagée pour l'instant.

Ecriture d'une fonction simple de filtre d'image

Voici un exemple d'une fonction cvErode

Au niveau de l'implémentation d'un module openCV, cela se résumera à écrire une fonction processImage(), les conversions éventuelles vers GEM étant intégrées dans la classe de base.

Exemple simple de module : cvlib_erode

Le fichier header : cvlib_erode.h

On définit une classe cvlib_erode qui est dérivée de la classe cvlib_ImageFilter_1_1

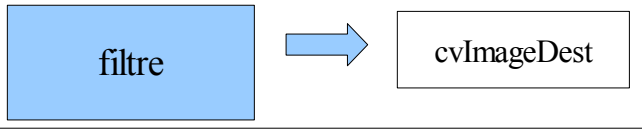
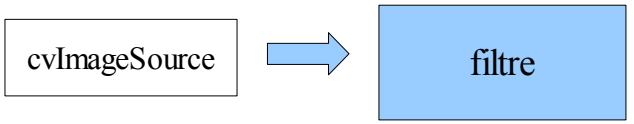
```
1       #include "cvlib_ImageFilter_1_1.h"
2       #include <stdio.h>
3       class cvlib_erode:
4               public cvlib_ImageFilter_1_1  // classe de base cvlib_ImageFilter_1_1
5       {
6               FLEXT_HEADER(cvlib_erode,cvlib_ImageFilter_1_1)
7               public:
8                       cvlib_erode(int argc,t_atom *argv);
9                        ~cvlib_erode();
10                      int iterations;
11              protected:
12                      virtual int process_default_image();
13                      void set_iterations(int i);
14              private:
15                      FLEXT_CALLBACK_I(set_iterations)
16      };
```

Le fichier C++ : cvlib_erode.cpp

```
1       #include "cvlib_erode.h"
2       // CONSTRUCTEUR
3       cvlib_erode::cvlib_erode(int argc,t_atom *argv)
4       {
5               AddInInt();  // inlet pour spécifier le nombre d'itérations de la méthode erode
6               FLEXT_ADDMETHOD(1,set_iterations);
```

```
7               // initialisation
8               set_iterations(1);
9               if(argc >= 1)
10                      if  (IsFloat(argv[0]))
11                              set_iterations(GetInt(argv[0]))
12      }
13
14      // DESTRUCTEUR
15      cvlib_erode::~cvlib_erode()
16      {}
17
18      // GESTION INLET DE DROITE
19      void cvlib_erode::set_iterations(int i)
20      {
21              iterations=min(64,max(0, i));
22      }
23
24      // MODULE DE TRAITEMENT PRINCIPAL : process_default_image()
25      int cvlib_erode::process_default_image()
26      {
27              // prepare dest image de  même taille que l'original
28              int result=cvlib_prepareIplImage(&cvImageDest, cvImageSource->width, cvImageSource->height,
                                                                                        cvImageSource->nChannels);
29              if (result)
30                      cvErode( cvImageSource, cvImageDest, 0,iterations );
31              return result;
32      }
33      // DECLARATION DE L'EXTERNAL
34      FLEXT_LIB_V("cvErode",cvlib_erode);
```

Les classes de base

cvlib_ImageFilter_0_1	Source d'image	
cvlib_ImageFilter_1_0	Destination d'image	
cvlib_ImageFilter_1_1	Filtre	
cvlib_ImageFilter_2_1	opérateur sur deux images sources	
cvlib_image_transformation	modification des dimensions (crop, resize, channel extract)	
Etc.....		

COI, ROI, mono/multichannel : Stratégies de prise en charge

Certaines fonctions de OpenCV ne prenant pas en charge COI (Channel of Interset), ROI (Region of Interest) ou le multichannel, il convient de définir une stratégie à suivre (conversion) dans le cas où l'image source présente des caractéristiques différentes de celles normalement prises en charge par la fonction openCV à implémenter.

Plusieurs scénarios sont par exemple possibles :

Fonction openCV	**Scénario 1**	**Scénario 2**	**Scénario 3**
Multichannel non pris en charge (exemple : cvCanny)	ne pas effectuer le traitement et afficher un message d'erreur en console	Convertir l'image source en gray, effectuer le traitement sur l'image convertie (monochannel)	Extraire chaque channel, effectuer le traitement sur chacun d'entre eux, puis reconstituer l'image multi-channel en sortie
Monochannel non pris en charge	ne pas effectuer le traitement et afficher un message d'erreur en console	Convertir l'image source en multichannel, effectuer le traitement sur l'image convertie (multichannel)	Conversion interne en multichannel
COI/ROI non pris en charge	ne pas effectuer le traitement et afficher un message d'erreur en console	Ignorer le paramètre COI/ROI, effectuer les conversions nécessaires de format	Faire une copie en prenant en compte le COI/ROI, opérer des conversions internes si nécessaire.

Le scénario 3 a l'avantage d'accroitre la polyvalence des fonctions openCV dans pure-data mais a pour inconvénients de rendre les erreurs moins faciles à déceler, et de s'éloigner des conventions OpenCV.
Il peut néanmoins être utile dans le cas où l'on souhaite écrire des modules de traitement de haut niveau.

Le scénario 2 est moins logique et peut porter à confusion.

Le scénario 1 permet de rester le plus près possible des fonctions natives d'OpenCV. Les conversions nécessaires devront être effectuées par l'utilisateur dans pure data à l'aide de modules appropriés (test, copie, extraction, combinaison, fusion, etc...)

Encodage de la stratégie choisie :

En décidant quelles fonctions virtuelles de la classe de base seront surchargées, on définit la stratégie de prise en charge des formats / du ROI/COI.

Fonction virtuelle	Rôle	Comportement par défaut :
virtual int process_roi_IplImage();	appelée lorsque l'image source contient un COI/ROI.	Génère un message d'erreur
virtual int process_default_image();	appelée lorsque l'image source ne correspond à aucune fonction surchargée	Génère un message d'erreur
virtual int process_4channels_image();	appelée lorsque l'image source contient 4 canaux	Appelle process_default_image()
virtual int process_RGBA_image();	appelée lorsque l'image source est RGBA	Appelle process_4channels_image()
virtual int process_BGRA_image();	appelée lorsque l'image source est BGRA	Appelle process_4channels_image()
virtual int process_3channels_image();	appelée lorsque l'image source contient 3 canaux	Appelle process_default_image()
virtual int process_RGB_image();	appelée lorsque l'image source est RGB	Appelle process_3channels_image();
virtual int process_BGR_image();	appelée lorsque l'image source est BGR	Appelle process_3channels_image()
virtual int process_2channels_image();	appelée lorsque l'image source contient 2 canaux	Appelle process_default_image()
virtual int process_1channel_image();	appelée lorsque l'image source est monocanal	Appelle process_default_image()