Jafar : Introduction au module Image

Le module Image

Thèmes abordés

- Problématique
- OpenCV et la structure *IPLImage*
- La classe *JfrImage*
- Le lien avec OpenCV
- Petits exemples ...

Problématique

- Pourquoi un module image ?
- Besoins :
  - Unification des structures images pour tous les modules de vision a venir
  - Structure de donnée moderne/extensible
  - Fonctions I/O, utilitaires et de traitements d'image bas niveau
- Solution ?

Problématique

- L'existant :
  - Calife !
    - Structures de données obsolètes
    - Fonctions utilitaires
    - Fonctions de traitement d'image orientée vision
    - Fonctions orientée robotiques

Problématique

- Le challenger retenu :
  - OpenCV
    - Structure de données plus complète (en C)
    - Fonctions utilitaires
    - Fonctions de traitement d'image orientée vision
    - Fonctions orientée robotique pratiquement inexistantes

Thèmes abordés

- Problématique
- **OpenCV et la structure *IPLImage***
- La classe *JfrImage*
- Le lien avec OpenCV
- Petits exemples ...

OpenCV et la structure IplImage

- OpenCV
  - Librairie de fonctions relatives a la vision développée initialement chez Intel
  - OpenSource
  - Très grande communauté (active)
  - Codes optimisés
  - Ne fonctionnent que sur des machines x86 (Windows/Linux) et PowerPC (MacOS X)

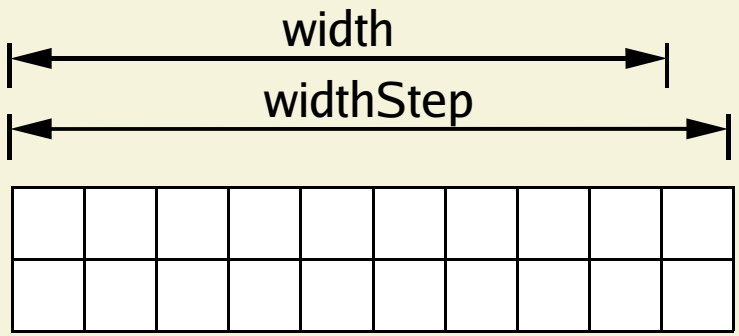
OpenCV et la structure IplImage

- *IplImage*
  - Structure employée par toutes les fonctions de traitement d'image d'OpenCV
  - Paramètres importants :
    - Taille de l'image : *width*, *height*
    - Nombre de plans chromatiques (*nChannels*)
    - Profondeur des pixels (*depth*) :
    - Le tableau de pixels : *imageData*
    - Rembourrage sur la largeur : *widthStep*

OpenCV et la structure IplImage

Précisions sur les paramètres de l'*IplImage*

- *nChannels* : 1,2,3 ou 4. Mais les fonctions d'OpenCV ne travaillent en général qu'avec 1 ou 3 plans.
- Profondeur possibles (*depth*) :
  - unsigned 1-bits/8-bits/16-bits
  - signed 8-bits/16-bits/32bits
  - floating point 32-bits/64-bits
- *widthStep* :

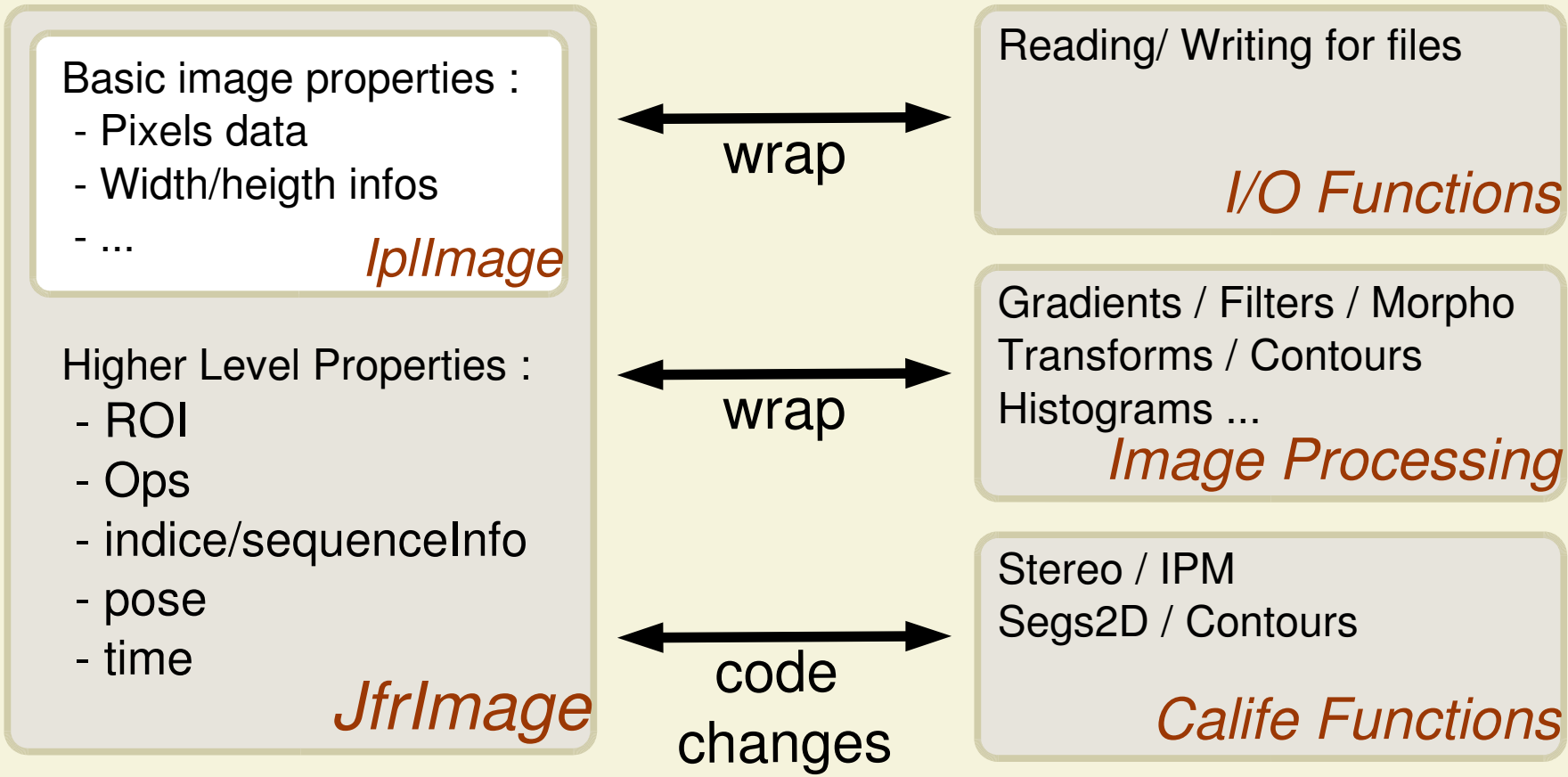


Thèmes abordés

- Problématique
- OpenCV et la structure *IPLImage*
- **La classe *JfrImage***
- Le lien avec OpenCV
- Petits exemples ...

La classe JfrImage

« Un gros schéma ...



La classe JfrImage

- ... vaut mieux qu'un petit discours »
  - Création d'une classe *JfrImage* (C++)
  - Encapsulation de la structure *IplImage* utilisée par OpenCV (C)
  - Réemploi au plus simple des fonctions OpenCV « bas-niveau »
  - Réinjection du code Calife « haut-niveau » en le modifiant

La classe JfrImage

- Les paramètres de *JfrImage* :
  - *opencvImage*, pointeur vers une *IplImage (IplImage*)*
  - *listROI*, liste de régions d'intérêts (*list<IplROI*>*)
  - *listOperations*, liste d'opérations (*list<string>*)
  - *sequenceInfo*, des infos sur la séquence (*string*)
  - *indice*, un indice (*int*)
  - *colorSpace*, type d'espace de couleur (*TypeColorSpace*)
  - *pose*, informations de position (*t3d**)
  - *timeStamp*, un temps (*timespec*)

La classe JfrImage

- Les paramètres de *JfrImage* :
  - *Le type TypeIplDepth :*
    - *enum TypeIplDepth {u1 = IPL_DEPTH_1U, u8 = IPL_DEPTH_8U, s8 = IPL_DEPTH_8S, u16 = IPL_DEPTH_16U, s16 = IPL_DEPTH_16S, s32 = IPL_DEPTH_32S, f32 = IPL_DEPTH_32F, f64 = IPL_DEPTH_64F}*
  - Chaque fonction de la classe *JfrImage* qui utilise le paramètre *depth* utilise ce type pour la définir

La classe JfrImage

- I/O fonctions :
  - Support du *depth* u8 uniquement
  - Support de 1 ou 3 plans chromatiques
  - Formats de fichiers :
    - Windows bitmaps : BMP, DIB
    - JPEG files : JPEG, JPG, JPE
    - Portable Network Graphics : PNG
    - Portable image format : PBM, PGM, PPM
    - Sun rasters : SR, RAS
    - TIFF files - TIFF, TIF

La classe JfrImage

- Fonctions utilitaires :
  - Cloner une image sans recopie
  - Copie du header *JfrImage*
  - Conversion de *depth*
  - Changement d'échelle
  - Conversion d'espace de couleur

La classe JfrImage

Accéder aux pixels :

- Les pixels sont stockés dans le *char* imageData*.
- C'est donc toujours un *char** quelque soit le *depth* choisi : *u8* (unsigned 8-bits) ou *f32* (floating-point 32-bits)
- Lors d'un accès aux pixels un *cast* de *imageData* est donc nécessaire.
- Deux méthodes sont possibles ...

La classe JfrImage

- La mauvaise :

```
#include "image/JfrImage.hpp"
using namespace jafar::image;

JfrImage img(800, 600, s16, 1);
int* pxPtr;

pxPtr = static_cast<int*>(img.getImageData());
```

  - Sur des machines *x86* sous *Linux* le type *int* est associé à *signed 16-bits* => ça marche
  - Mais quand est-il sur MacOS X ou de futurs plateformes ou avec d'autres compilateurs ?

La classe JfrImage

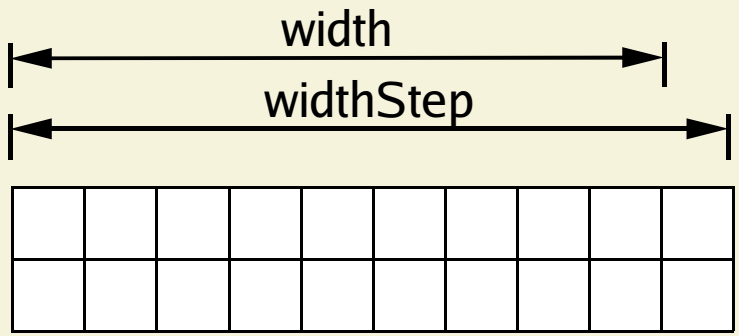
- La bonne :

```
#include "image/JfrImage.hpp"
using namespace jafar::image;

JfrImage img(800, 600, s16, 1);
JfrImage::s16_cell::cell_type *pxPtr;

 pxPtr = img.imageData<JfrImage::s16>();
```

  - Cette méthode garantit l'utilisation du bon type pour parcourir les tableaux de pixels et fonctionne sur tout types de plateformes
  - Merci a fréderic

La classe JfrImage

- WidthStep :
  - Alignement des lignes sur 8 octets
  - WidthStep est la vrai longueur des lignes exprimee en nombre de sizeof(char)
  - Conversion penible



La classe JfrImage

- Un vrai bout de code :

```
#include "image/JfrImage.hpp"
using namespace jafar::image;

JfrImage img(800, 600, JfrImage::s16, 1);
JfrImage::s16_cell::cell_type *pxPtr = src_.imageData<JfrImage::s16>();

int nBytes =
      depth_utility::bit_size<JfrImage::s16_cell::cell_type>::size_in_bytes;

for(int i=0;i<height;i++) {
   for(int j=0;j<width;j++)
      *(cell++) = 1;
   cell += widthStep/nBytes - width;
}
```

Thèmes abordés

- Problématique
- OpenCV et la structure *IPLImage*
- La classe *JfrImage*
- **Le lien avec OpenCV**
- Petits exemples ...

Utilisation des fonctions d'OpenCV

- Principe :
  - Encapsulation d'une *IplImage* dans la classe *JfrImage* => permet l'utilisation des fonctions d'*OpenCV*
  - Comment appeler les fonctions ?
    - Directement :

      cvLaplace(source.getIplImage(),dst.getIplImage());
    - A travers un *wrap* de la fonction *OpenCV* :

      laplace(const JfrImage& source, JfrImage& dst);
      laplace(source,dst);

Utilisation des fonctions d'OpenCV

- Pourquoi wrapper ?
  - Pour être indépendant d'*OpenCV*. Un changement de librairie n'occasionne que des changements dans les fonctions de *wrap*
  - Parce qu'*OpenCV* n'a pas de gestion d'erreur mais qu'on peut en mettre une dans le *wrap*
- Toutes les fonctions ne sont pas encore wrappées

Utilisation des fonctions d'OpenCV

Un exemple, la fonction *Gradient::canny()*

```
class Gradient {
private :
public :
  static inline void canny(const JfrImage& src_, JfrImage& dst_,
    double th1_, double th2_, int aperture_size_=3) {
  cvCanny(src_.getIplImage(), dst_.getIplImage(), th1_, th2_,
    aperture_size_);
  std::ostringstream stream;
  stream << "canny : th1=" << th1_ << " th2=" << th2_ << "
    aperture_size=" << aperture_size_ ;
  dst_.addOperation(stream.str());
  }
}; // class Gradient
```

Thèmes abordés

- Problématique
- OpenCV et la structure *IPLImage*
- La classe *JfrImage*
- Le lien avec OpenCV
- **Petits exemples ...**