Prise en main caméra Basler

Projet :	Blue Point > CameraAcquisition
Auteur :	Roman CLAVIER
Date de création :	10/04/2025
Dernière modification :	10/04/2025

Liens utiles pour la documentation de Basler, pour utiliser le SDK Pylon :

- Pylon programmeurs guide : https://docs.baslerweb.com/pylonapi/cpp/pylon_programmingguide
- Format de pixels : https://docs.baslerweb.com/pixel-format
- Documentation sur les fonctionnalités de la caméra qui peuvent être modifié par code / Pylon Viewer : https://docs.baslerweb.com/features

Sujet : Spécificités et explications concernant l'acquisition de la caméra Basler, avec conversion optimisée du SDK Pylon vers OpenCV

Les explications suivantes portent sur le code des fichiers :

- BluePoint / CameraAcquisition / CameraProcessor.cpp
- BluePoint / CameraAcquisition / BaslerProcessor.cpp

CameraProcessor : Trouver les meilleures dimensions pour redimensionner une image, avec conservation du ratio, et qu'elle tienne dans une taille définie.

Admettons que nous ayons une image de dimensions et que nous souhaitons la faire rentrer dans un cadre dont les dimensions sont connues (et plus conventionnelles (ex : 640x480 | 4/3).

La question que nous nous posons est la suivante : Quelles sont les nouvelles dimensions qui permettent de faire rentrer cette image dans ce cadre, tout en conservant le ratio largeur/hauteur afin de ne pas déformer l'image.

La fonction suivante est faite pour répondre à ce problème :

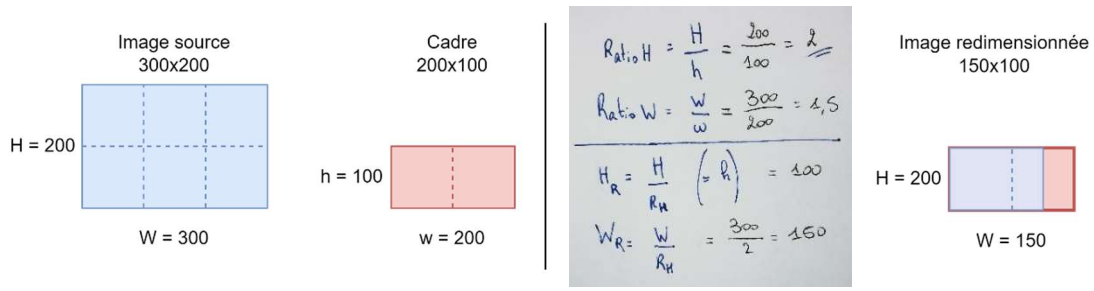
```
void findNewSizeToKeepAspectRatio(int originalWidth, int originalHeight, int& width, int& height) const;
```

⇨ ALGO :

Calculer les 2 ratios, et prendre le plus en tant que diviseur afin d'avoir les nouvelles dimensions. Il y a au minimum une dimension égale à une dimension du cadre. La seconde est soit plus petite, soit égale si les ratios w/h de l'image et du cadre sont égaux.

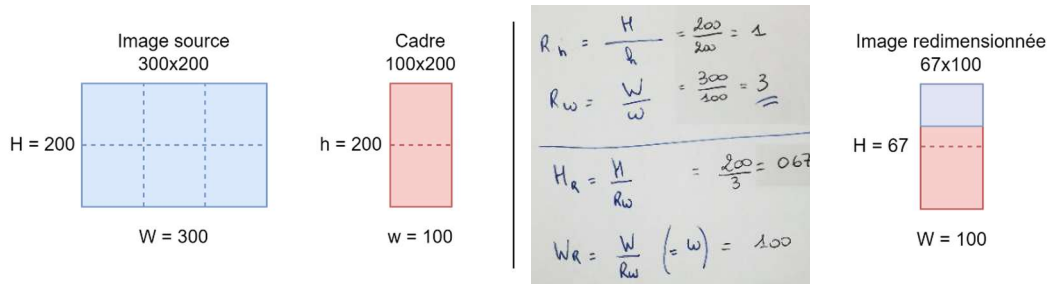
Les schémas suivants montrent par la pratique, les calculs à appliquer.

Exemple 1 :



→Ratio des hauteurs plus grand ➔ C'est lui le diviseur.

Exemple 2 :



→Ratio des largeurs plus grand ➔ C'est lui le diviseur.

BaslerProcessor : Acquisition optimisée d'une caméra Basler : conversion d'une image Pylon vers une image OpenCV, avec redimensionnement de l'image et centrage dans une image de dimensions conventionnelles.

Nous nous intéressons ici à l'acquisition d'image avec une caméra Basler. Pour cela, nous utilisons le SDK de Basler : **Pylon**.

Il est nécessaire de l'installer :

https://www.baslerweb.com/fr-fr/downloads/software/?downloadCategory.values.label.data=pylon

À ce jour, la dernière version du SKD est : Pylon 8.1.0 (mars 2025). Je recommande d'installer la suite logicielle pour Windows qui inclue les API nécessaires (C, C++, C#) et également le logiciel **Pylon Viewer**, qui permet, entre autres, de configurer une caméra Basler, vérifier qu'elle fonctionne, acquérir des images, des vidéos et les enregistrer. Il faut redémarrer l'ordinateur pour que le PATH Windows soit bien mis à jour et connu dans Visual Studio.

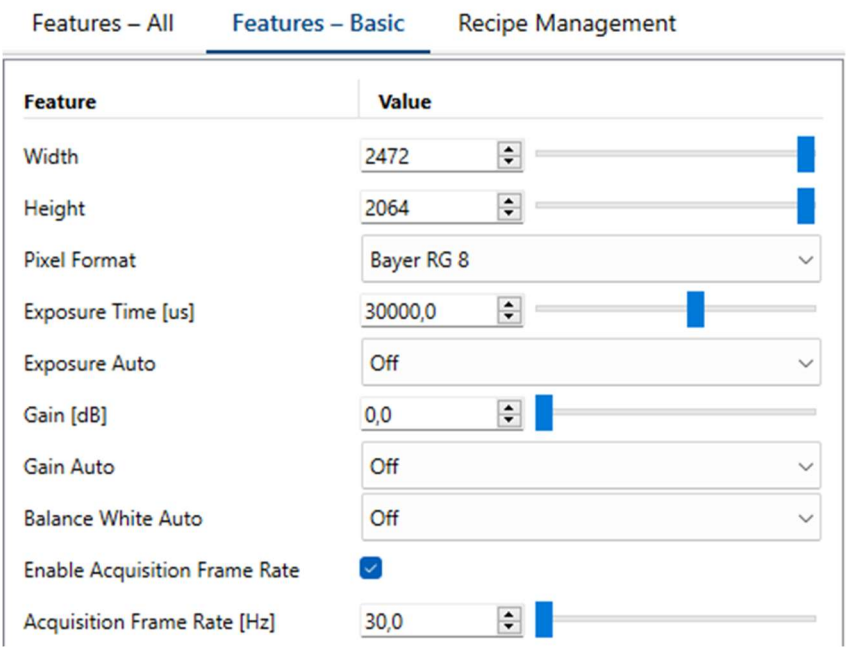
Voici les principaux éléments fournis par la documentation Basler afin de se connecter à la caméra et à démarrer l'acquisition :

```
#include "pylon/PylonIncludes.h"
```

```
Pylon::PylonInitialize(); // à faire une seule fois dans le constructeur

m_camera = new Pylon::CInstantCamera(Pylon::CTlFactory::GetInstance().CreateFirstDevice());
m_camera->Open();
m_camera->StartGrabbing();

Pylon::PylonTerminate();    // à la faire une seule fois dans le destructeur
```

Avec Pylon Viewer, il est possible de savoir les dimensions des frames envoyées ainsi que le format des pixels. Ces valeurs peuvent être vérifiées et éditées dans Pylon Viewer, ou par code. Par défaut, nous avons :

- Dimensions : 2472x2064
- Format : BayerRG8



L'image acquise n'est donc pas en RGB : les composantes sont reconstruites avec un dématriçage (demosaicing). Cela permet de passer d'un format **BayerRG8** (ou tout autre format Bayer) à **RGB8** (ou **BGR8** car OpenCV fonctionne en BGR). Il est évidemment possible de régler le format de pixel directement sur du RGB8, mais un effet dégradant sur les fps a été observé.

De plus, l'image est assez grande (donc lourde pour des traitements). Il faut donc la redimensionner, tout en gardant le ratio. Néanmoins, le ratio de l'image source n'est pas conventionnelle : 2472/2064 ≈ 1.198. Si nous souhaitons l'avoir dans un ratio plus conventionnel comme du 4/3 (640x480), alors nous devons insérer l'image redimensionnée centrée dans une image 640x480 (il y aura une marge sur les côtés). De plus, si nous souhaitons afficher l'image dans une fenêtre, il faut créer une QImage.

Le code suivant permet de faire ça avec des fonctions natives de Pylon et d'OpenCV :

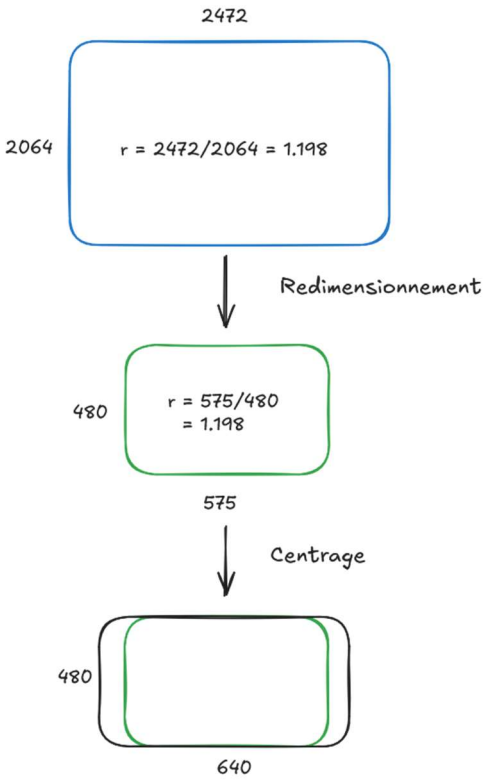
```
Pylon::CGrabResultPtr ptrGrabResult;
Pylon::CImageFormatConverter converter;
converter.OutputPixelFormat = Pylon::PixelType_BGR8packed;

while (m_camera->IsGrabbing())
{
    m_camera->RetrieveResult(5000, ptrGrabResult, Pylon::TimeoutHandling_ThrowException);
    if (ptrGrabResult->GrabSucceeded())
    {
        // Convertir la frame de Bayer RG8 en BGR8
        Pylon::CPylonImage image;
        converter.Convert(image, ptrGrabResult);

        // Conversion PylonImage (BGR8) vers OpenCV (BGR8)
        cv::Mat frame = cv::Mat(ptrGrabResult->GetHeight(), ptrGrabResult->GetWidth(), CV_8UC3, (uint8_t*)image.GetBuffer());

        // Redimensionnement
        int newwidth, newHeight;
        findNewSizeToKeepAspectRatio(ptrGrabResult->GetWidth(), ptrGrabResult->GetHeight(), newwidth, newHeight);
        cv::Mat resizedFrame = cv::Mat::zeros(cv::Size(newwidth, newHeight), CV_8UC3); // 575x480 (issue de 2472x2064)
        cv::resize(frame, resizedFrame, cv::Size(newwidth, newHeight)); // redimensionnement avec les meilleurs proportions

        // Centrage dans une frame 640x480
        int offsetX = (640 - resizedFrame.cols) / 2;
        int offsetY = (480 - resizedFrame.rows) / 2;
        cv::Rect roi(offsetX, offsetY, resizedFrame.cols, resizedFrame.rows);
        cv::Mat workFrame = cv::Mat::zeros(cv::Size(640, 480), CV_8UC3); // 640x480
        resizedFrame.copyTo(workFrame(roi));
        // workFrame contient maintenant la frame de la camera, centrée, de taille 640x480

        // Conversion OpenCV (BGR8) vers QImage (RGB8)
        QImage outImage(workFrame.data, workFrame.cols, workFrame.rows, workFrame.step, QImage::Format_BGR888);
    }
}
```

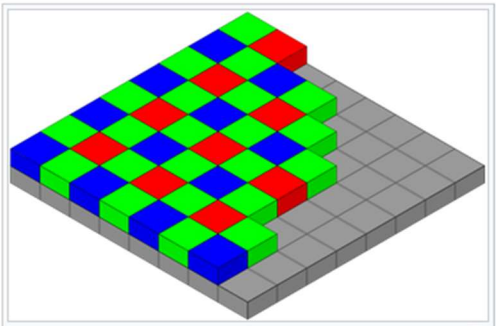
Vous trouverez une version similaire dans la méthode mise en commentaire (donc inactive) BaslerProcessor::CaptureLoop.

Les tests effectués en configuration x64/Debug ont donné un temps moyen de **21 ms** pour réaliser ces 5 étapes. Sachant que la caméra a été configuré pour du 30 fps, nous disposons donc de **33 ms** entre chaque image. 64% du temps disponible est donc déjà pris uniquement pour obtenir une image.

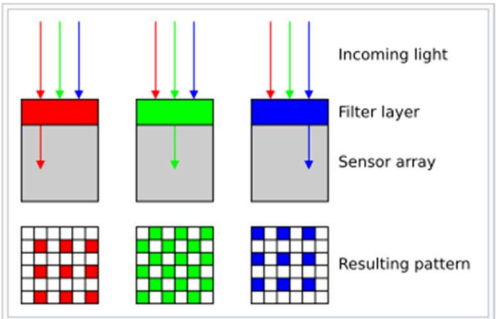
Afin d'optimiser tout cela, une fonction a été créée afin de réaliser la conversion BayerRG8 à BGR8, le redimensionnement, le centrage et la conversion dans une trame OpenCV, ainsi que la conversion en QImage en BGR8. Ainsi, tout est fait en une seule lecture du buffer reçu, ce qui améliore les drastiquement les performances.

Le schéma ci-dessous rappelle ce que nous devons faire :





The Bayer arrangement of color filters on the pixel array of an image sensor



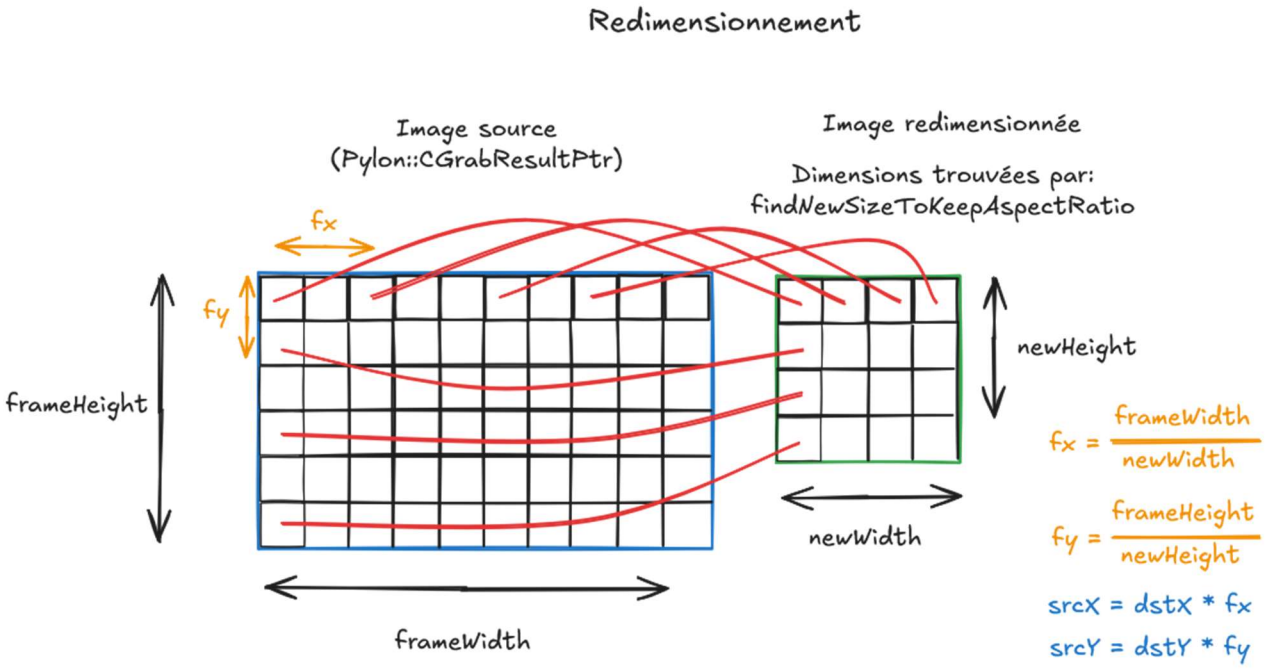
Profile/cross-section of sensor

Les prochaines étapes / explications / schémas permettent de comprendre la fonction :

```
void convertAndResize1(const Pylon::CGrabResultPtr& ptrGrabResult, cv::Mat& outFrame, QImage& outImage, int newWidth, int newHeight)
```

Les schémas feront apparaître le nom des variables utilisées pour comprendre leur utilité.

Redimensionnement



En itérant sur les dimensions de l'image redimensionnée, on peut trouver les coordonnées correspondantes dans l'image source.

Dématriçage BayerRG8 en RGB

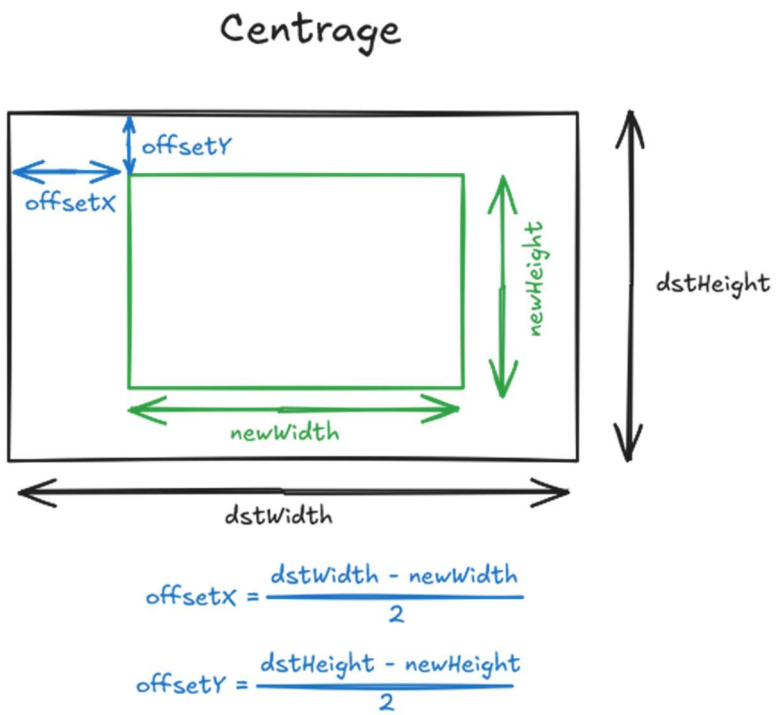
À présent que nous connaissons les coordonnées dans l'image source, nous faisons appel à la fonction suivante afin de recomposer les composantes RGB :

```
void demosaicPixel1(const uint8_t* bayer, int x, int y, int width, int height, int stride, uint8_t& B, uint8_t& G, uint8_t& R)
```

Cette fonction s'occupe du dématriçage du format Bayer RG8 en RGB8. Ainsi, nous pouvons écrire l'image OpenCV en BGR et l'image Qt en RGB.

Centrage

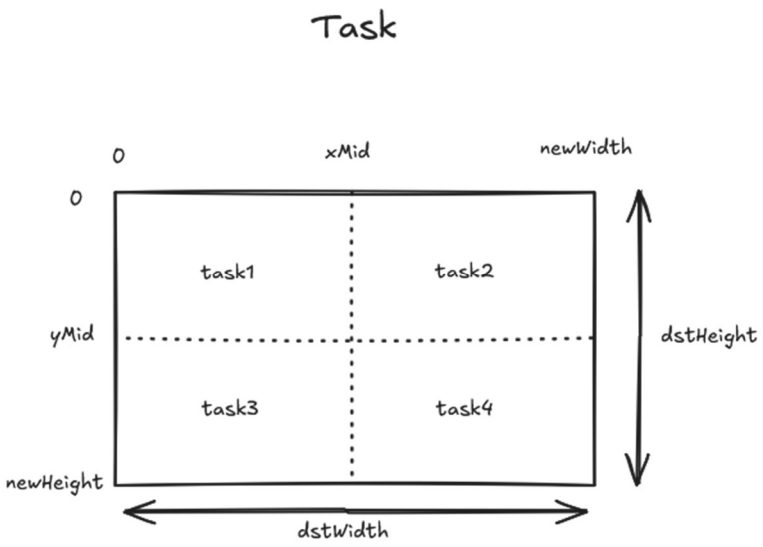
Afin de construire nos deux images, nous n'utilisons pas directement les dstX et dstY (coordonnées dans l'image redimensionnée), nous appliquons également un offset afin d'appliquer le centrage dans l'image finale.



Optimisation

À présent, vous pouvez comprendre l'algorithme dans la fonction **convertAndResize1**. Cette fonction a ensuite été optimisé afin d'éviter un maximum d'allocation mémoire durant le processus : c'est la fonction **convertAndResize2**. Pour cela, toutes les variables sont créées en amont, et les arguments de la fonction **demosaicPixel2** sont désormais passer par référence, pour éviter les copies inutiles.

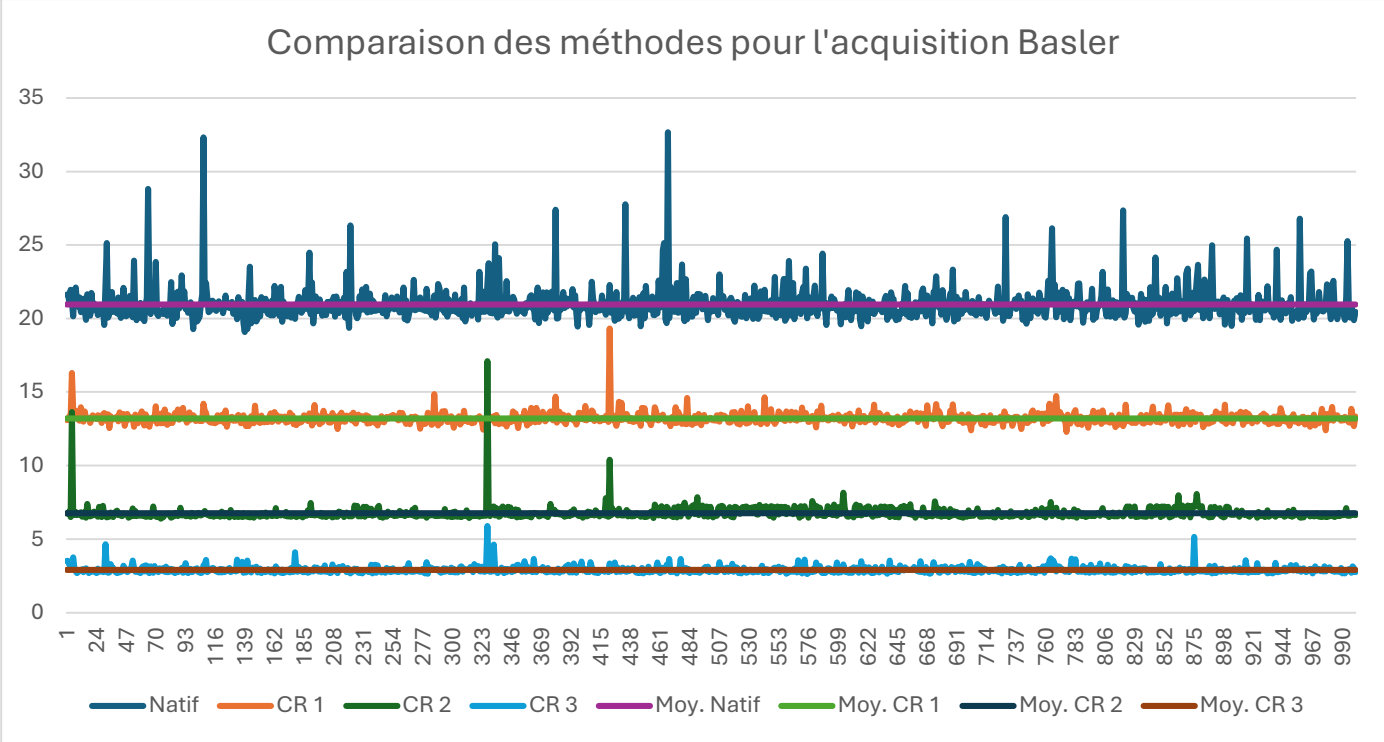
Enfin, afin d'être encore plus performant, la fonction **convertAndResize2** a été encore optimiser : fonction **convertAndResize3**, qui utilise toujours **demosaicPixel2**. Dans cette fonction, la tâche a été divisé en 4 tâches parallèles qui écrivent chacune une partie de l'image.



La fonction **convertAndResize3** créer 4 tâches **convertAndResize3Task** qui prennent en entrée les limites d'écriture.

Les performances de ces 4 algorithmes (natif, convertAndResize1, convertAndResize2 et convertAndResize3) ont été mesurées et comparées.

En mode Debug, nous obtenons ces données :



Avec des temps moyens (en millisecondes) à :

Moy. Natif	Moy. CR1	Moy. CR2	Moy. CR3
20,96	13,21	6,76	2,90

Soit une amélioration moyenne de **7,22**.

En Release, nous obtenons :

Moy. Natif	Moy. CR1	Moy. CR2	Moy. CR3
5,97	4,16	2,42	1,09

Soit une amélioration moyenne de **5,48**.