Analyse avec le logiciel ImageJ d'un lot d'images en microscopie par immunofluorecence de cellules de fibroblastes irradiées en X à l'E.S.R.F.

F. Chandez, Gerard Montarou

PCCF RI 1004
24 Juillet 2010

Analyse avec le logiciel ImageJ d'un lot d'images en microscopie par immunofluorecence de cellules de fibroblastes irradiées en X à l'E.S.R.F.

F.Chandez, G.Montarou

Laboratoire de Physique Corpusculaire de Clermont-Ferrand
CNRS/IN2P3, Clermont Université
F-63177 Aubière Cedex France

Sommaire

Le problème de radiosensibilité individuelle est plus d'actualité que jamais face à l'apparition d'une diversification des types de faisceaux et notamment l'apparition des faisceaux de particules lourdes (hadronthérapie par ions légers tels les ions carbone) pour le traitement de tumeurs cancéreuses radio-résistantes aux traitements classiques par photons. C'est dans ce cadre que le programme de recherche expérimentale ROSIRIS (RadiobiOlogie des Systèmes Intégrés pour l'optimisation des traitements utilisant des rayonnements ionisants et évaluation du RISque associé) a été initié par l'IRSN et l'INSERM.
Un des objectifs de ce programme scientifique est de corréler les observables caractérisant les événements biologiques précoces aux événements physiques résultant des dépôts d'énergie des rayonnements ionisants dans les cellules. L'originalité de l'approche réside dans l'utilisation des techniques les plus récentes d'identification des cassures double-brin de l'ADN. La méthode choisie pour l'étude des cassures doubles brins est l'observation des foci H2AX en microscopie par immunofluorescence. L'immunofluorescence avec des anticorps spécifiques contre les formes phosphorylées de l'histone H2AX (γ-H2AX) permet en effet de mettre en évidence les cassures double-brin (DSB) sous forme de spots fluorescents appelés foci. Suivant la nature et l'énergie des particules, le nombre, la taille, l'intensité lumineuse et la répartition spatiale des foci γH2AX est susceptible de changer. Le projet ROSIRIS a pour objectif de constituer des plateformes d'irradiation, ainsi qu'une plateforme de traitement et des analyses des images cellulaires, basée sur l'utilisation d'un microscope automatisée. Cette plateforme devrait permettre de traiter une grande quantité d'images et donc de constituer une base de données significatives permettant de tester les observables biologiques en fonctions des conditions d'irradiation. Parallèlement à ce dispositif expérimental d'acquisition d'image, il sera nécessaire de mettre au point un logiciel de traitement et d'analyse le plus indépendant possible de l'opérateur permettant une analyse automatique des données. Cette note décrit les résultats d'une première approche de ce domaine, destiné à en préciser les difficultés et les défis pour obtenir et réaliser une telle plateforme d'analyse d'image.

ajouter les keyword en francais et en anglais

Le projet ROSIRIS

Les individus présentent de façon innée une sensibilité variable aux rayonnements ionisants. Il a été démontré que certains patients qui sont soumis à des irradiations thérapeutiques montrent une réponse à l'irradiation plus forte qu'attendu au niveau des tissus sains, aussi bien en termes d'effets précoces que tardifs.
Les complications inhérentes aux surdosages au niveau des organes à risque justifient une évaluation des risques liés aux radiothérapies, mais aussi une prise en compte de la radiosensibilité individuelle du patient et de son statut génétique.
Le problème de radiosensibilité individuelle est plus d'actualité que jamais face à l'apparition d'une diversification des types de faisceaux et notamment l'apparition des faisceaux de particules lourdes (hadronthérapie par ions légers tels les ions carbone) dont les indications relèvent précisément des situations de radiorésistance aux photons.
C'est dans ce cadre que le programme de recherche expérimentale ROSIRIS (RadiobiOlogie des Systèmes Intégrés pour l'optimisation des traitements utilisant des rayonnements ionisants et évaluation du RISque associé) a été initié par l'IRSN et l'INSERM.
Un des objectifs général de ce projet est le développement de la modélisation biophysique des événements précoces induits par les rayonnements ionisants, notamment la modélisation des événements stochastiques primaires. L'enjeu principal est de proposer une nouvelle définition de la dose d'irradiation qui rende mieux compte de l'hétérogénéité des événements physiques qui surviennent quelques 10^{-6}s après l'irradiation à l'échelle du noyau cellulaire et/ou de la cellule afin de mieux prédire les effets biochimiques et biologiques correspondants.
Il s'agit également de corréler les observables caractérisant les événements biologiques précoces aux événements physiques. L'originalité de l'approche réside dans l'utilisation des techniques les plus récentes d'identification des cassures double-brin de l'ADN. Les actions à mener sont de:

- définir un modèle biologique « simple » et des observables significatives à l'échelle subcellulaire qui soient adaptées à la modélisation de la topologie des évènements biologiques radioinduits,
- choisir et développer des outils de modélisation de la topologie des dépôts d'énergie (traces) qui permettent de prédire ces observables en fonction des conditions expérimentales d'irradiation.

La méthode choisie pour l'étude des cassures doubles brins est l'observation des *foci* H2AX en microscopie par immunofluorescence. L'immunofluorescence avec des anticorps spécifiques contre les formes phosphorylées de l'histone H2AX (γ-H2AX) permet en effet de mettre en évidence les cassures double-brin (DSB) sous forme de spots fluorescents appelés *foci*
Suivant la nature et l'énergie des particules, le nombre, la taille, l'intensité lumineuse et la répartition spatiale des *foci* γ H2AX est susceptible de changer. En effet, des observations très récentes (N Foray *et al.*) ont montré que la décondensation de la chromatine, notamment due à des cassures simple-brin de l'ADN (SSB) pouvait disperser le signal lumineux du focus initial et aboutir à la formation quasi-homogène de *foci* de plus petite taille.
De plus, en plus du dénombrement automatique des *foci* et de leur taille, des tests peuvent être effectués, à dose macroscopique égale, avec des agents chimiques condenseurs/ou décondenseurs de l'ADN comme le DMSO, le butyrate de sodium ou certains agents intercalants.
Enfin, des données préliminaires obtenues avec des irradiations cellulaires par des neutrons indiquent que le nombre, l'intensité lumineuse, la taille et la dispersion des *foci* varient drastiquement avec l'énergie incidente des neutrons
Des techniques d'analyse d'image permettent de quantifier le nombre, la taille et l'intensité de chaque *foci*. Il est alors possible de corréler ces trois premières observables (nombre, taille et intensité des *foci*) aux événements physiques produits dans des conditions d'irradiation variées, à partir d'un grand nombre d'images pour chaque configuration différente d'irradiation. Une telle acquisition de données est aujourd'hui rendue possible par l'automatisation de la collecte des images microscopiques sur des plateformes automatisées.
L'analyse de la répartition spatiale des *foci* et des alignements possibles des événements biologiques dans le noyau cellulaire irradié est plus complexe et nécessite un développement méthodologique plus important.
A partir des données topologiques des dépôts d'énergie primaires obtenues par simulation, il est possible d'agréger (clusterisation ou voxelisation) ces dépôts afin de pouvoir les corréler aux dommages de l'ADN matérialisés par les *foci* γ H2AX.

Pour ce faire, différentes tailles d'agrégats correspondant aux dimensions des cassures de l'ADN (10-100 nm) peuvent être exploré. Des corrélations seront recherchées entre tailles d'agrégats, intensité lumineuse et taille des *foci* d'une part, et nombre des évènements physiques et des *foci* d'autre part.
La sensibilité de la méthode pourra être améliorée dans un second temps, en prenant en compte la topologie de la distribution des agrégats dans les évènements simulés.

Mise en évidence des dommages ADN par marqueurs moléculaires

La molécule d'ADN se trouve sous différentes formes de condensation au cours du cycle cellulaire. L'unité de base de la chromatine, appelée nucléosome, regroupe 146 paires de bases d'ADN enroulées autour d'un groupement de petites protéines basiques. Il s'agit d'un octamère de protéines appelées histones. Cette structure est constituée de divers types de protéines histones. À ce jour, cinq histones sont décrites:

- les histones H2A, H2B, H3 et H4 forment un octamère globulaire (de structure 2x(H2A,H2B,H3,H4)) qui permet l'enroulement de 146 paires de bases d'ADN en un tour trois quart afin de former un nucléosome ou "collier de perles" : 1er degré de condensation de l'ADN (11 nm)
- l'histone H1 permet quant à elle la compaction des nucléosomes et rigidifie la structure hélicoïdale ainsi obtenue (obtention d'un solénoïde : 2ème degré de condensation de l'ADN - 30 nm -).

Puisque les processus tels que la réparation et la transcription de l'ADN doivent accéder à l'ADN, le nucléosome est une structure dynamique qui est régulée par de nombreuses modifications covalentes affectant principalement leur extrémité N et C-terminale (acétylation, méthylation, phosphorylation,…..)

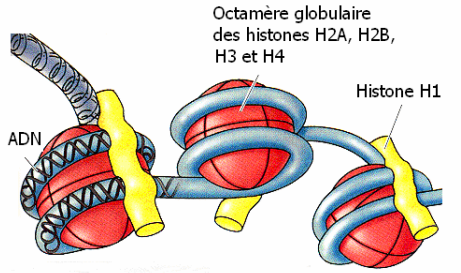


Figure (1) : Vue de nucléosomes et de l'enroulement de l'ADN autour de celle-ci

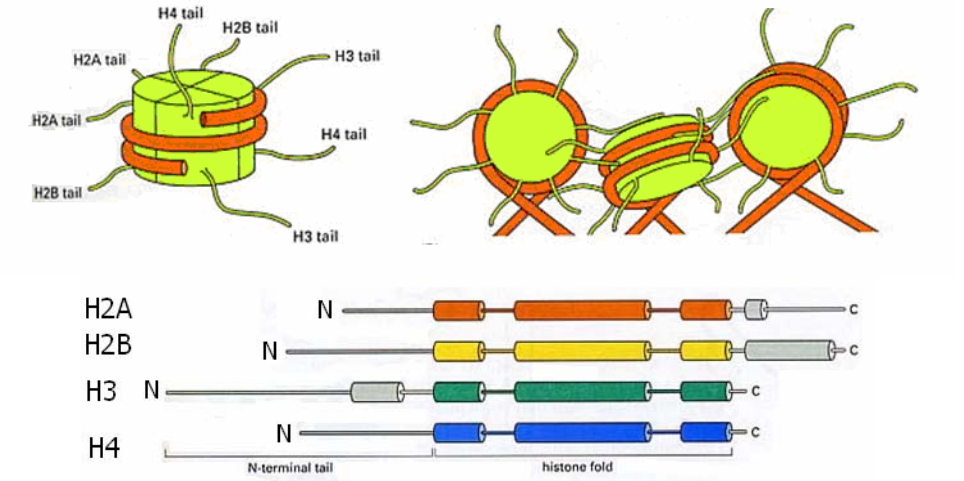


Figure (2) : Organisation structurelle des histones de l'octamère globulaire : chacune des protéines histones subissent en permanence des modifications covalentes affectant principalement leur extrémité N-terminale (acétylation, méthylation, polyribosylation, ubiquitinylisation, phopshorylation).

L'histone H2AX, une isoforme de l'histone H2A, représente 2 à 25 % du taux d'expression relatif de la protéine H2A, en fonction du type cellulaire ou tissulaire étudié et est distribuée de façon homogène sur la chromatine. Elle possède une extrémité C-terminale plus longue, avec un motif SQE particulier, phosphorylable par l'intermédiaire d'une sérine associée au motif SQE par les kinases de la famille PIKK: ATM (Ataxia telangiectasia mutated), ATR et DNA-PKcs.

D'un point de vue fonctionnel, H2AX semble être principalement associé au maintien de l'intégrité du génome, au moyen de sa participation à la réparation les bris double-brin à l'ADN, introduits de façon exogène par un dommage environnemental (radiation ionisante, produits chimiques).
Immédiatement après l'apparition d'une cassure double brin de l'ADN, la double hélice d'ADN se décondense sur une région de 2 MB encadrant la cassure La protéine histone H2AX devient alors accessible et la sérine 139 de cette protéine est phosphorylée de manière dose dépendante par les kinases de la famille des PI3K reconnaissant spécifiquement un motif SQE.
Chez les mammifères, cette famille compte plusieurs protéines impliquées dans la réparation des cassures double brin, dont ATM (Ataxia Telangiectasia Mutated), ATR (AT-Related) et DNA-PK (protéine kinase dépendante de l'ADN). Pour chaque cassure double brin de l'ADN, 2000 molécules d'histone H2AX sont phosphorylées
Cette phosphosrylation peut être mise en évidence à l'aide d'anticorps spécifiques, faisant apparaître la cassure double-brin sous forme de spots fluorescents appelés *foci* γ-H2AX comme le montre la figure (3). On suppose qu'il existe une corrélation un-pour-un a été obtenue entre le nombre de cassures double-brin et le nombre de *foci*. Mais d'aucuns considèrent que la relation γ-H2AX/DSB n'est plus fondée si l'on tient compte du phénomène de condensation de la chromatine. D'autres marqueurs moléculaires précoces comme pDNA-PK, MRE11, pATM, ATR peuvent être éventuellement utilisés pour compléter l'analyse des dommages radioinduits de l'ADN (Joubert *et al.*, 2008).

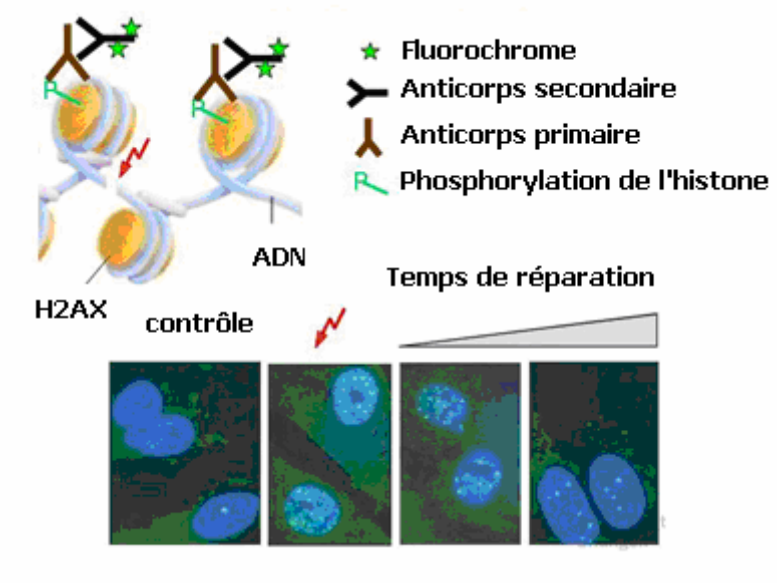


Figure (3) : Visualisation des dommages induits par radiation ionisante par l'analyse des *foci* γ-H2AX.

Description du Protocole des expériences

La lignée cellulaire choisie dans le projet ROSIRIS pour les irradiations est une lignée des fibroblastes humains non-transformés provenant d'un témoin radiorésistant (lignée 1BR3), provenant d'un témoin radio-résistant. Cette lignée a été choisie pour son adhérence, son homogénéité, sa stabilité génomique, la facilité d'obtenir une population cellulaire en phase de plateau (99% de cellules en G0/G1), ainsi que pour l'ensemble de données existantes sur ces cellules irradiées aux rayons X et la large gamme disponible de fibroblastes humains issus de pathologies

différentes. Ces cellules, bien caractérisées radiobiologiquement, ont un ADN homogène, bien modélisable.
Afin de faire varier les conditions du dépôt d'énergie lors des irradiations, il est nécessaire d'utiliser des rayonnements de *TEL* différent. Dans un premier temps, deux types de rayonnement sont utilisés :

- un rayonnement de bas *TEL* (rayons *X*),
- un rayonnement de haut *TEL* (faisceaux monoénergétiques de neutrons de 2 keV à 19 MeV).

Le mode d'irradiation est également important :

- en mode macro-faisceau, le nombre de particules incidentes est inconnue ; de même que les points d'impact et donc l'identification claire des cellules touchées. Le contrôle de la dose est difficile. Le dépôt d'énergie dans le milieu n'est pas homogène.
- En mode microfaisceau, le nombre de particules délivrées est contrôlé avec précision. Il est possible d'étudier les effets induits par de très faibles doses y compris par une seule particule. Un microfaisceau permet d'irradier sélectivement une zone précise au niveau de la cellule individuelle avec un nombre contrôlé de particule incidente.

Les cellules sont irradiées et les dommages de l'ADN sont analysées juste après l'irradiation, au moment où tous les dommages sont reconnus et bien avant que les phénomènes de réparation ne fassent varier leur nombre.

Expérience d'irradiation à l'ESRF

Les données de base consistent en plusieurs séries d'images de cellules de fibroblaste (1BR3) irradiées à 2 Gy en X avec 4 heures de réparation, acquises lors d'une manipulation, le 6 février 2009 à l'ERSF
Les cellules sont disposées sur des lamelles (1,5x1,5 cm). Après irradiation, le protocole de traitement des cellules irradiées a été le suivant :

- Rinçage au PBS (tampon phosphate salin)
- Fixation au paraformaldéhyde
- Lyse (détergent) pour rendre la membrane poreuse (1 à 5 minutes)
- Application anticorps primaire (40 minutes à 37°)
- Application anticorps secondaire (20 minutes)
- Dépôt d'une goutte de VectaShield, incluant la solution DAPI sur la lame de verre de support
- Application de la lamelle avec collage par le VectaShield
- Dépôt de vernis sur les cotés de la lamelle pour rendre hermétique la couche entre la lame et la lamelle

La durée totale du processus de traitement est de deux heures environ.

L'acquisition des d'images est effectuée avec un microscope BX51 de marque Olympus équipé de plusieurs objectifs dont les plus couramment utilisés sont le 10X et le 100X, les autres sont : 2X, 40X et 60X. Une brève description de cet appareil, ainsi que ses principes de fonctionnement, sont donnés dans l'annexe A. Dans le cas de l'utilisation en 100X une goutte d'huile assure la liaison entre l'objectif et la lame. Le microscope est équipé d'une caméra numérique CCD Olympus DP70 dont la description et les principes de fonctionnement sont décrits dans l'annexe B.
La présence de *foci* γ-H2AX dans le plan de focalisation du microscope apparait clairement sur l'image comme le montre la figure (4). Bien que la microscopie à focale fixe ne restitue pas la 3D comme la microscopie confocale, les cellules fibroblastes, du fait de leur forme aplaties (type « œuf au plat ») restituent bien les focis sur le plan de focalisation du microscope.
Sur une lamelle il y a environ 10^5 cellules et quelques centaines sont visibles dans le champ du microscope avec un objectif 10X. 75% des cellules sont en phases G1. Lors de l'examen des champs les cellules sont « triées » selon leur phase dans le cycle cellulaire
Globalement le nombre de focis visibles décroit avec le temps de réparation laissé aux cellules, par contre leur intensité augmente. Très schématiquement, pour des irradiations en *X* , pour 10 minutes on a beaucoup de petit focis, pour 24 heures on a quelques gros focis très lumineux
Les lames peuvent être également examinées en changeant la longueur d'onde d'excitation (voir Annexe B). Ceci permet de visualiser l'ADN sous sa forme condensée (chromatine) par l'intermédiaire de la fluorescence de la molécule DAPI (couleur bleue) dans le noyau comme le montre la figure (5).

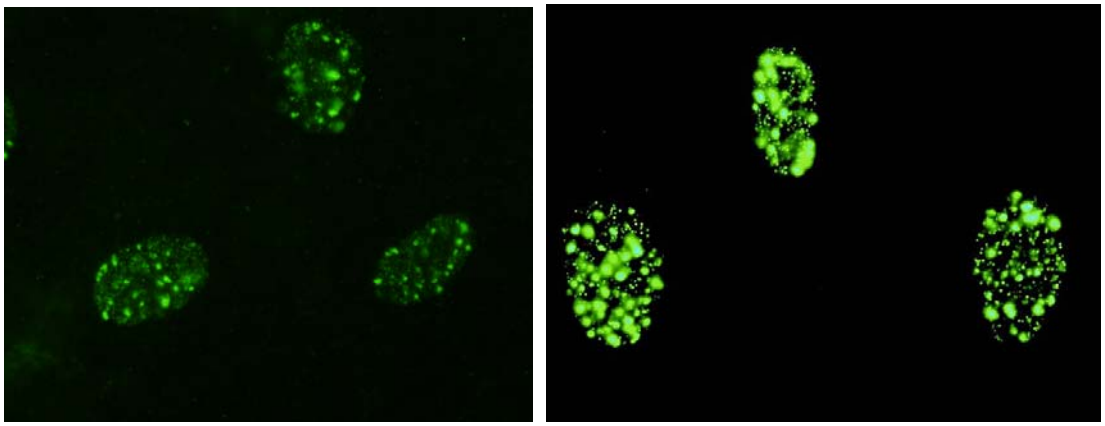


Figure (4) : cellules fibroblastes irradiées montrant la présence de cassures double-brin sous forme de *foci* γ -H2AX

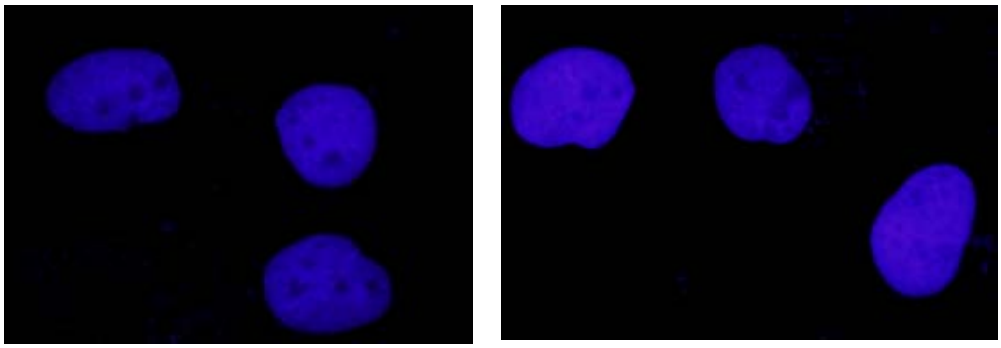


Figure (5) : fluorescence bleue au niveau de la chromatine dans les noyaux marqués au DAPI

Plusieurs résolutions sont disponibles. Les résolutions utilisées durant l'acquisition ont été faites en :

- Résolution standard 1360x1024 pixels
- Haute résolution 4080x3072 pixels

Avec un grossissement 10X, on peut observer plusieurs centaines de noyaux. En grossissement 100X, on peut avoir 3 ou 4 noyaux en moyenne par champ
A la longue, l'illumination des cellules engendre un phénomène de « fading », c'est à dire une perte de fluorescence ; il faut alors augmenter le temps de pose pour avoir une intensité équivalente dans l'image.

L'analyse des lames irradiées a permis de constituer un lot de 123 images différentes dont la notation originelle est « image_x » avec x=1 à 123. Une pré-analyse de ce lot d'image a permis d'éliminer les images « ratées » à la suite d'une mauvaise manipulation ou configuration. Après ce premier tri les images ont été classées en 5 catégories et renumérotées

Images Dapi
Ces images ont été acquises avec un grossissement 10X et un temps de pose de vue de 100 ms. Les deux modes de résolutions ont été utilisées:

- Haute résolution (4080x3072 pixels) pour les images « *fibro_dapi_10x_HR_1 à 5* »
- Résolution standard (1360x1024 pixels) pour les images « *fibro_dapi_10x_RS_1 à 5* »

Images 100X
Ces images ont été acquises avec un grossissement 100X . Pour chaque champ sélectionné, deux images différentes ont été acquise :

- une en mode Dapi avec un temps de pose de 50 ms ;
- une en mode H2AX avec un temps de pose de 1 s.

Les images sont en haute résolution (4080x3072 pixels). La numérotation des images permet d'associer les images par paires et donc d'avoir les deux modes par champ acquis

- Dapi pour les images « *fibro_dapi_100x_HR_1 à 31* »
- H2AX pour les images « *fibro_h2ax_100x_HR_1 à 31* »

Images de calibration «*Dark*»
Le microscope permet d'enregistrer des images obtenues uniquement par le CCD. Ces images enregistrent le courant d'obscurité de la CCD (images « *black* »), avec deux temps de pause différents : 1 s et 23 ms

- temps de pause de 1 s pour les images « *fibro_black_HR_1s_1 à 11* »
- temps de pause de 23 ms pour les images « *fibro_black_HR_23ms_1 à 11* »

Images de calibration « *Offset* »
On peut également enregistrer des images avec un « *shutter* » interrompant le chemin de la lumière avant l'arrivée dans le CCD. Ces images ont été acquise avec un temps de pause 1s, identique au temps de pose avec cellule (images « *shutter* »)

- Images « *fibro_shutter_HR_1 à 11* »

Images de calibration « PLU »
Afin de tester l'uniformité de la réponse du CCD, des images de lame vierge ont été acquises avec différents filtres (images de Plage de Luminosité Uniforme) en grossissement 100X et haute résolution: 4080x3072 pixels

- mode Dapi sur 10 s pour les images « *fibro_PLU_DAPI_10s_1 à 3* »
- mode H2AX sur 10 s pour les images « *fibro_PLU_h2ax_10s_1* »
- mode H2AX sur 60 s pour les images « *fibro_PLU_h2ax_60s_1 à 2* »

Les mesures effectuées, visant à corriger les défauts inhérents au système d'imagerie (« Black », « PLU », « shutter »), pourront servir à donner une première connaissance des bruits à prendre en compte pour obtenir les « meilleures » images que possible. L'origine de ces bruits est présentée dans l'annexe C.

Pour la suite de cette étude nous avons sélectionné les images acquises avec un grossissement 100X .Pour chaque champ sélectionné, deux images différentes sont disponibles:

- en mode Dapi pour les images fibro_dapi_100x_HR_1 à 30
- en mode H2AX pour les images fibro_h2ax_100x_HR_1 à 30

Description du logiciel ImageJ

Deux notions d'échantillonnage interviennent pour produire une image numérique :

- échantillonnage en coordonnées spatiales,
- échantillonnage en intensité.

Une image est constituée d'un ensemble de pixels. Le pixel représente ainsi le plus petit élément constitutif d'une image numérique. Le nombre de pixels dans les deux dimensions détermine la définition de l'image. Nos images ont une définition de 4080x3072 pixels en haute résolution.
Les images sont au format TIFF (Tagged Image File Format**).** Ce format permet de garder le maximum d'informations au contraire du format JPEG. Tous les fichiers TIFF peuvent être lus sur PC, Mac et Unix. L'entête contient d'autres informations, nommées étiquettes (tags), permettant l'insertion de documentation supplémentaire standard. Par contre le format JPEG fonctionne par décomposition des images en parties de 8x8 pixels. Il supprime des détails dans chaque partie (niveaux de compression de 1% à 99%).
Une image couleur est décomposée en 3 composantes, R(Red), G(Green) et B(Blue). Pour chaque pixel, le codage de la couleur est représenté par trois valeurs, chacune codée sur un octet (de 0 à 255). Ceci correspond à $256^{\wedge 3}$ combinaisons, c'est-à-dire au total 24 bits soit plus de 16 millions de couleurs. Par conséquent la taille d'une image est de 3x4080x3072 octets, soit 36 Megaoctets
Une image couleur peut être condensée en faisant une somme pondérée des trois composantes, R(Red), G(Green) et B(Blue). Pour chaque pixel, l'intensité est égal à la somme des trois valeurs, pondérées chacune d'un certain poids. Par exemple, on peut attribuer un poids égal à chacune des composantes couleurs, le résultat étant un niveau de gris sur un octet (de 0 à 255). Cette opération est utilisable à des fins de présentation mais pas forcément utile pour l'analyse des images

ImageJ est un des meilleurs logiciels Open Source pour le traitement d'image en biologie et en médecine pour le traitement et l'analyse d'images. Il peut :

- afficher, éditer, traiter et analyser la plupart des formats de fichier existants
- proposer un grand nombre d'algorithmes pour l'analyse de l'image.

A ce jour, ImageJ est un des logiciels les plus utilisés pour le traitement d'images acquise par microscopie. Le programme s'exécute soit sur une machine virtuelle Java ou sur un JRE (Java Runtime Environment). De fait, le langage Java rend ImageJ multiplateforme (Windows, Mac, Linux, Unix, …).
Le principe de base d'ImageJ est d'être conçu pour être extensible sous forme de modules d'extensions. ImageJ dispose de fonctions pour le chargement de modules Java : les « plugins ».
Les formats d'images supportés en natif sont classiques, mais la librairie « OME plugins for ImageJ » proposé par le projet *www.openmicroscopy.org* ajoutent une collection de plugin pour manipuler les formats de fichiers les plus populaires en microscopie.
En parallèle à la version standard d'ImageJ, l'utilisation d'une version adaptée à biologie permet de bénéficier des très nombreux et puissants plugins dédiés à la microscopie, associés à un manuel en ligne très bien documenté.
ImageJ bénéficie d'une grande communauté de développeurs qui contribue à étendre les fonctionnalités du noyau. L'utilisateur dispose maintenant d'une bibliothèque de plus de 300 fonctions disponibles gratuitement. De nombreuses opérations de traitement d'images sont réalisables avec ImageJ. Parmi lesquelles :

- visualisation, analyse, et traitement d'images 8/16/32 bits
- formats d'image utilisables : TIFF, GIF, JPEG, BMP, DICOM, …
- calculs d'aires, affichage de l'échelle, mesures de distances et d'angles,
- modification de contraste, smoothing, détection des contours, filtre médian,
- ajustement de l'histogramme des niveaux de gris, débruitage, correction d'éclairage,
- opérations logiques et arithmétiques entre images, traitements morphologies : érosion/dilatation, ligne de partage des eaux, squelettisation.

L'utilisateur peut reproduire des séquences de commandes en enregistrant une macro-commande.
Plus de renseignements concernant ImageJ sont disponibles sur:

- La page d'accueil du site officiel sur ImageJ: *http://rsbweb.nih.gov/ij/*
- Le portail pour la documentation sur ImageJ: *http://imagejdocu.tudor.lu*

Principe du prétraitement des images avec ImageJ

Les images issues de l'imagerie de fluorescence, acquise par le détecteur (CCD) présentent un certain nombre de defaults qui peuvent affecter la qualité de ces images et par conséquent les performances de l'analyse. Cette section a pour but d'introduire les éléments permettant de mettre au point un protocole de prétraitement des images pour améliorer la qualité des images finales. Pour cela il faut se rappeler que le capteur utilisé est un CCD et donc comme tout instrument possède des defaults intrinsèques (bruits intrinsèques). En fait le CCD n'est pas le seul élément susceptible d'introduire des biais dans le signal (bruits externes). L'annexe (C) fait un bilan exhaustif de ces différents bruits.
Dans le cas de l'analyse des images acquises à l'ESRF, nous ne considérerons uniquement que le bruit thermique du CCD. D'autre part, nous supposerons que la pose est suffisamment longue pour que le bruit de lecture finisse par devenir négligeable devant le bruit thermique.

Le prétraitement, ou calibration, consiste à retrouver l'image originale formée sur la matrice du capteur par le flux de photons incidents. Le prétraitement idéale nécessiterait d'acquérir au début de chaque prise d'image, des acquisitions spécifiques, appelées : « *offset* », « *dark* », « *flat-field* »:

Signal d'offset (« *offset* ») **:** On fait une pose la plus brève possible, dans l'obscurité totale (obturateur fermé).

Signal thermique (« *dark* ») **:** On va acquérir une image avec un temps de pose du même ordre de grandeur que l'image, en bloquant toute entrée de lumière qui pourrait arriver sur le CCD (obturateur fermé) et en prenant le même temps d'exposition que celle de l'image. L'image «*dark*» comprend uniquement le signal de bruit de fond électronique de l'appareil. Donc lorsque le temps de pose augmente, les charges thermiques deviennent très importantes.

Signal de Plage de Luminosité Uniforme (« *flat-field* ») : Une dernière catégorie de bruits provient de la variation de sensibilité [1] des pixels du CCD et des défauts du système optique. Par exemple les

[1] *Les pixels des capteurs n'ont pas une sensibilité homogène à la lumière. Un capteur observant une scène uniforme ne donnera pas forcement une image uniforme. L'effet peut être lié à des impuretés présentent dans le silicium du CCD. La probabilité d'un pixel à créer un photoélectron varie fortement avec la longueur d'onde. Certains pixels sont souvent aveugles à la plupart des photons en dehors de leur plage normale. Ils ne peuvent donc détecter qu'une fraction des photons qui pourrait être détectée. Chaque pixel dans le détecteur a sa propre efficacité (bien que les variations de pixel à pixel soient extrêmement faibles) et cette efficacité est propre à chaque longueur d'onde. Heureusement, ce comportement est stable et peut être corrigé en divisant les images brutes par des images de PLU contenant ces effets.*

poussières sur les surfaces optiques peuvent produire des ombres. De même les variations de luminosité conduisent à un vignettage (non-uniformité de l'intensité de la lumière à l'intérieur du champ du microscope). L'image « *flat-field* » est utilisée pour isoler et corriger des imperfections de la chaîne optique. On fait une pose longue sur une image de fluorescence supposée uniforme en luminosité, avec la même mise au point que les images. Cette image de référence doit être obtenue à partir d'une lame contenant un film de colorant fluorescent, réputé uniforme, à défaut un objet uniformément fluorescent. La même configuration de filtre pour l'acquisition des images doit être utilisée

Si on appelle :

- $S(X,Y)$ l'image brute en sortie du CCD,
- $I(X,Y)$ l'image après prétraitement
- $Dark(X,Y)$ la matrice du signal thermique (« Dark »)
- $PLU(X,Y)$ la matrice en illumination uniforme (« PLU »)
- $Offset(X,Y)$ la matrice du signal d'offset (« Offset »)

On peut écrire

$$I(X,Y) = \frac{\left[S(X,Y) - Dark(X,Y) - Offset(X,Y)\right]}{\left[PLU(X,Y) - Dark(X,Y) - Offset(X,Y)\right]} \approx \frac{\left[S(X,Y) - Dark(X,Y)\right]}{\left[PLU(X,Y) - Dark(X,Y)\right]}$$

On obtient finalement une image dite « calibrée » du bruit de fond parasite et des variations d'intensité des pixels. Ces traitements de base sont effectués via le traitement des opérations sur des images. Il faut rappeler néanmoins que toute opération mathématique (soustraction, multiplication) introduit également une dégradation du rapport signal sur bruit.

Les figures suivantes montrent la distribution de probabilité (exprimées en %) de la valeur d'offset calculées à partir

- des images de type « *Black* » (bruit thermique moyen sur 10 images acquises pendant une seconde chacune) pour la figure (6)
- des images de type « *shutter*» (image moyenne sur 10 images acquises pendant une seconde chacune) pour la figure (7)

Dans les deux cas la somme des probabilités sur l'ensemble des valeurs d'offset (entre 0 et 257) est égale à 100. Les résultats des deux plots montrent clairement que les images ne sont pas affectées d'un quelconque piédestal puisque à 99%, les valeurs des pixels est égal à 0. Par conséquent dans la suite le courant d'obscurité sera considéré comme négligeable. Cependant l'évaluation de ces distributions est un préalable à toute analyse.

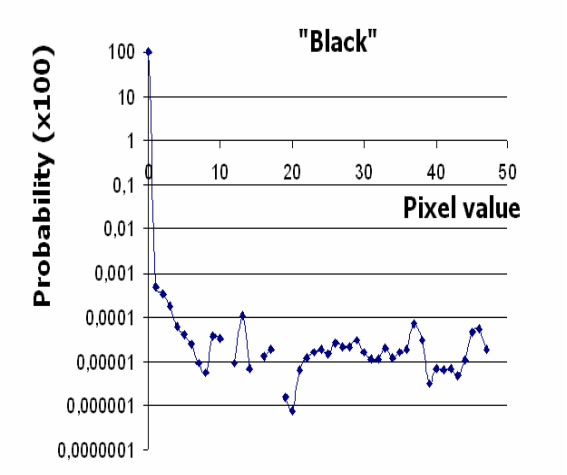


Figure (6) : Distribution des probabilités des valeurs d'offset d'après les images de type « black » (bruit thermique 10 images acquises pendant une seconde)

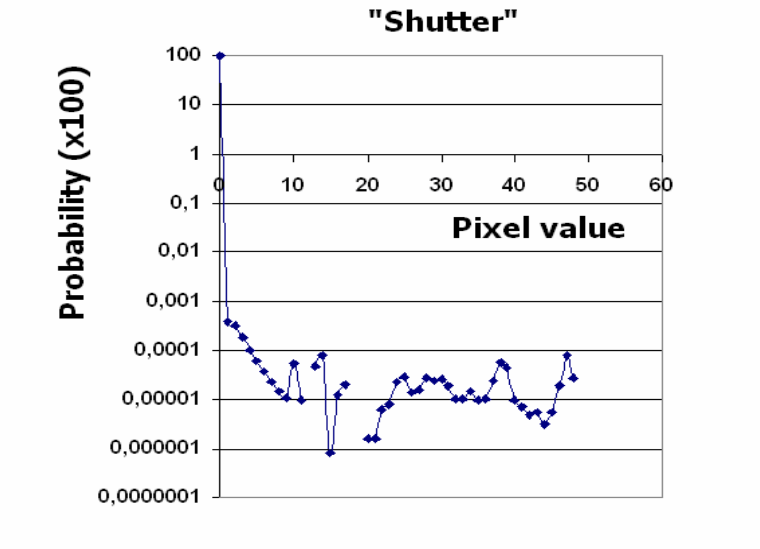


Figure (7) : Distribution des probabilités des valeurs d'offset d'après les images de type « shutter» (10 images acquises pendant une seconde)

Les images de type « PLU » ; c'est-à-dire « *fibro_PLU_h2ax_10s*» ou « *fibro_PLU_h2ax_60s*» on été utilisées pour estimer la distribution des valeurs des pixels de ces images. En cas d'illumination uniforme, les spectres des trois composantes devraient être une fonction delta. La figure (8) montre ces distributions pour une illumination de 10 secondes. L'illumination n'est pas suffisante pour que les valeurs de chacune des composantes soient significatives. La figure (9) par contre montre les distributions pour une illumination de 60 secondes. A partir de ces distributions, les valeurs moyennes et les largeurs totales à mi-hauteur des distributions sont résumées dans le tableau (1).

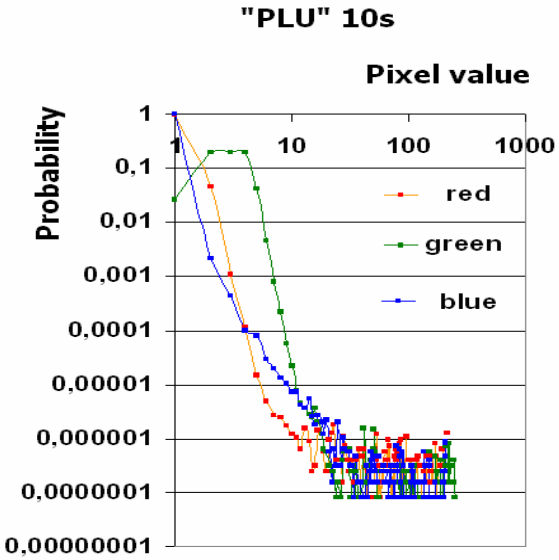


Figure (8) : Distribution des probabilités des valeurs des pixels dans les trois composantes Red, Green et Blue, d'après une image de type «PLU» (10 secondes)

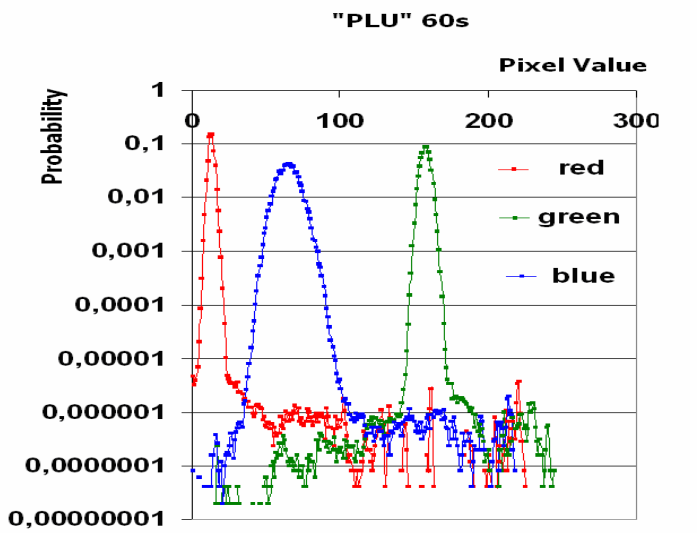


Figure (9) : Distribution des probabilités des valeurs des pixels dans les trois composantes Red Green et Blue, d'après une image de type «PLU» (60 secondes)

	Red	Blue	Green
Valeur moyenne	12,3	64,2	157
LTMH	5,05	16,5	4,44
R	0,41	0,26	0,03

Un plugin de prétraitement sous ImageJ a été mis au point pour améliorer le rapport signal sur bruit des images de focis en H2AX (« *fibro_h2ax_100x_HR* ») ou de DAPI (« *fibro_DAPI_100x_HR* »)à l'aide des images de type PLU (« *fibro_PLU_h2ax_60s_1 à 2* »).

- On part donc d'image PLU; dans le cas présent on utilise les images « *fibro_PLU_h2ax_60s_1* » et « *fibro_PLU_H2ax_60s_2* » que l'on combine en une image moyenne pour réduire le bruit statistique

- On décompose cette image moyenne de champ uniforme, codées en RGB 24-bit, en 3 images de niveaux de gris 8-bit, chacune de ces images correspondant à un des canaux de couleur ; R, G et B.

- Pour chaque composante de couleur de l'image moyenne, on calcule la valeur moyenne sur l'ensemble des pixels :

$$\left\langle PLU_{8bits}(i,j)\right\rangle_{Canal=R,G,B} = S_{Canal=R,G,B}$$

- Chaque composante de couleur de l'image moyenne est convertie du mode 8-bits au mode 32-bits

$$PLU_{8bits}(i,j)_{Canal=R,G,B} \rightarrow PLU_{32bits}(i,j)_{Canal=R,G,B}$$

- A partir des trois moyennes $S_{Canal=R,G,B}$ et des trois images moyennes (R,G,B) en mode 32-bits $PLU_{32bits}(i,j)_{Canal=R,G,B}$, on calcule trois « masques »; c'est-à-dire on constitue une image $I_2(i,j)$ où pour chaque pixel (i,j) $Canal = R,G,B$ est associé le scalaire

$$\lambda(i,j)_{Canal=R,G,B} = \left\langle PLU_{32bits}(i,j)\right\rangle_{Canal=R,G,B} \Big/ PLU_{32bits}(i,j)_{Canal=R,G,B}$$
$$= S_{Canal=R,G,B} \Big/ PLU_{32bits}(i,j)_{Canal=R,G,B}$$

- Le masque est une image qui est utilisée pour créer une nouvel image I' obtenue par la fonction « Image calculator » d'ImageJ ; c'est-à-dire en multipliant, pixel à pixel une image originale $I_1(i,j)$ par le masque $I_2(i,j)$

$$I'(i,j) \rightarrow I_1(i,j) \times I_2(i,j)$$

$I'(i,j)$ est l'image finale. L'utilisation du mode 32-bit évite ainsi les débordements provoqués par la multiplication de 2 pixels d'une image 8-bit. Par exemple la valeur 150 multipliée par 4 deviendra 255 en mode 8-bit au lieu de 600 en mode 32-bit. Une image 16-bit non signée peut suffire.

Si on utilise le masque d'un canal (R,G ou B) avec la composante de la PLU moyenne correspondant au même de canal de couleur, on obtient une image uniforme dont la valeur pour chaque pixel est égal à la valeur moyenne utilisée pour calculer le masque. En effet

$$I'(i,j)_{Canal=R,G,B} = \lambda(i,j)_{Canal=R,G,B} \times PLU_{32bits}(i,j)_{Canal=R,G,B}$$

$$= \left(\left\langle PLU_{32bits}(i,j) \right\rangle_{Canal=R,G,B} \Big/ PLU_{32bits}(i,j)_{Canal=R,G,B} \right) \times PLU_{32bits}(i,j)_{Canal=R,G,B}$$

$$= \left\langle PLU_{32bits}(i,j) \right\rangle_{Canal=R,G,B}$$

Dans le cas ou l'on utilise les images de type PLU (« *fibro_PLU_h2ax_60s_1 à 2* »), et selon la figure (7), ces valeurs sont égales à 12,3 pour le canal R, 157 pour le canal G et 64,2 pour le canal B.

<u>Les masques servent donc a corriger les images de la non uniformité de la réponse de la camera dans chacune des trois composantes de couleur</u>

- On décompose l'image à traiter (fibroblaste en H2AX ou Dapi) selon les composantes R, G et B : $ImageSource(i,j)_{Cannal=R,G,B}$

- On applique sur chaque composantes R, G et B, le masque correspondant

$$ImageCorrigée(i,j)_{Canal=R,G,B} = ImageSource(i,j)_{Canal=R,G,B} \times \lambda(i,j)_{Canal=R,G,B}$$

- Les 3 composantes corrigées sont fusionnées en une image RGB qui est donc l'image finale corrigée

Nous avons réalisé un plugin d'ImageJ, « *PLU_RGB_Avg* », qui a pour objectif d'effectuer le prétraitement d'images Dapi ou H2ax à partir de PLU(s) selon le protocole décrit précédemment.

Au démarrage de ce plugin, il faut choisir le répertoire de travail ; c'est à dire l'emplacement du fichier de log et lui donner un nom (par défaut le nom du plugin). Le plugin crée alors le fichier log récapitulatif des opérations effectuées tout au long de l'exécution du plugin. Le plugin agit sur une pile d'images de PLU sélectionnées par l'utilisateur dans un répertoire.

Pour faciliter la vérification du plugin, nous avons crée des images réduites composées de matrice de 85x64 pixels obtenues à partir des images originales haute résolution. Cela n'affecte en rien le processus de prétraitement, si ce n'est le temps de calcul et l'espace mémoire nécessaire pour traiter les images. Le plugin a également été vérifié sur des images complètes. Les fichiers d'essai sont donc :

- <u>Image de PLU</u> : « fibro_PLU_1.tif » et «fibro_PLU_2.tif »
- <u>Image de fibroblastes</u> H2AX : «fibro_9.tif »

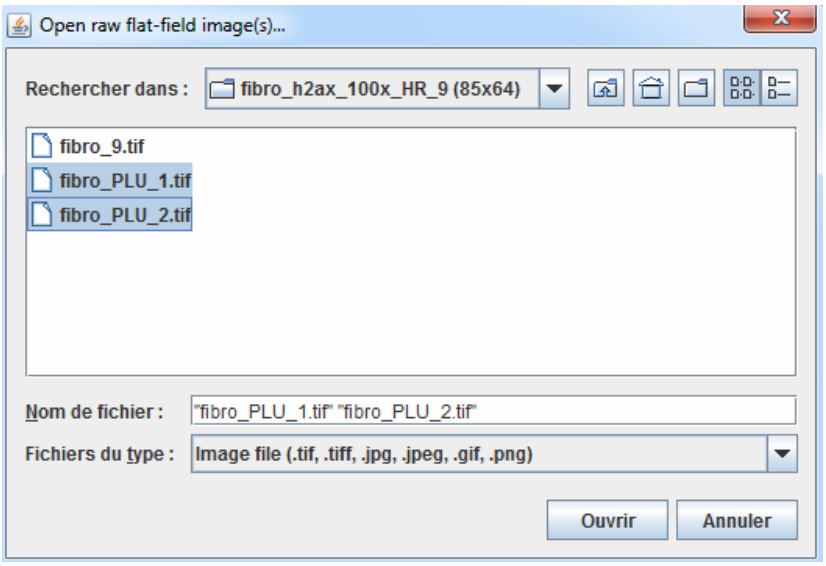


Figure (10) : Chargement des images TIFF de type RGB, utilisées par le pluggin « *PLU_RGB_Avg* »: images de type PLU et l'image à corriger

Le plugin applique une projection sur les images de PLU pour obtenir une image de même type (toutes les images utilisées doivent avoir la même résolution). La méthode de projection sur la pile d'images de PLU est par défaut une moyenne.
Tous les paramètres nécessaires pour le prétraitement s'initialisent dans le plugin au moyen d'un panel comme le montre la figure suivante:

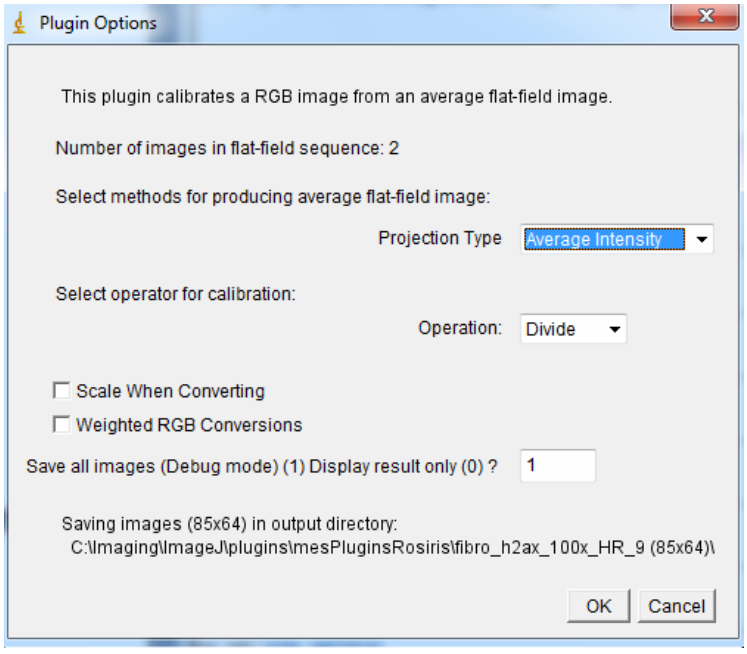


Figure (11) : Choix des options de prétraitement

Le plugin affiche alors l'image de la pile d'images de PLU projetée avec la méthode de projection choisie, dans notre cas « Average Intensity » :

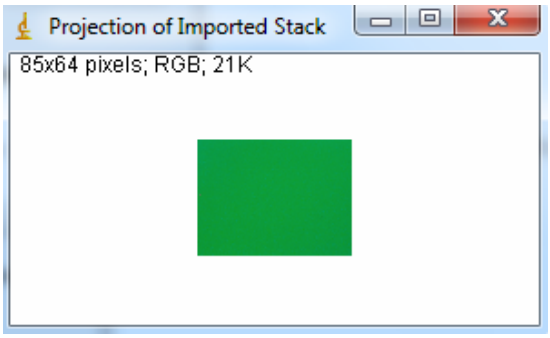


Figure (12) : PLU moyenne de la projection des images fibro_PLU_1.tif et fibro_PLU_2.tif

Le plugin sépare ensuite l'image PLU moyenne en trois images 8-bit représentant les canaux R, G et B et récupère la valeur moyenne pour chaque canal. Puis divise indépendamment les 3 canaux par leur valeur moyenne (le plugin ne permet pas d'autre possibilité que la moyenne).

On choisit ensuite à partir d'un panel, l'image à prétraiter :

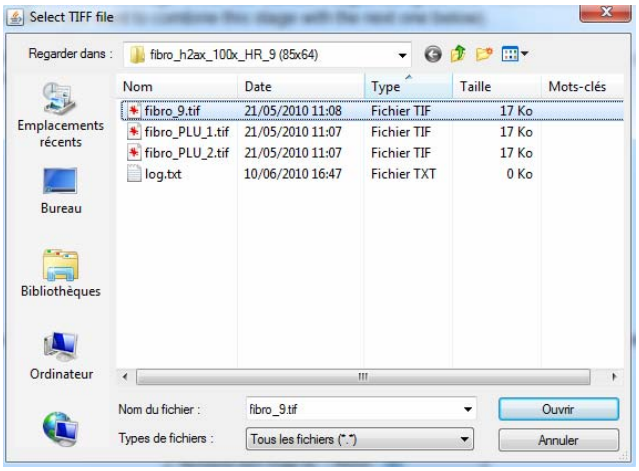


Figure (13) : Sélection de l'image DAPI ou H2AX à prétraiter (à condition que celle-ci ait été acquise dans les mêmes conditions que les images PLU).

On divise (méthode par défaut) les images, pixel-à-pixel, pour obtenir une image résultante. La formule utilisée pour l'opérateur 'Division' est la suivante :

$$ImageCorrigée(i,j)_{Canal=R,G,B} = ImageSource(i,j)_{Canal=R,G,B} \times \lambda(i,j)_{Canal=R,G,B}$$

Les 3 canaux corrigés séparément sont fusionnés en une image RGB finale, qui est affichée par le plugin, ainsi que l'histogramme de celle-ci :

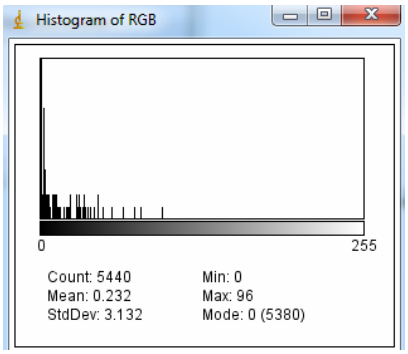


Figure (14) : Histogramme de l'image RGB finale

Résultat et évaluation des corrections du prétraitement

Pour évaluer l'effet des corrections induites par le prétraitement, on utilise comme image originale l'image « *fibro_ h2ax_100x_HR_21* » ainsi que l'image Dapi correspondante.

L'évaluation des corrections du prétraitement peut être rapidement effectuée par comparaison de l'aspect général de l'image corrigée par rapport à l'image originale. Les figures (17) et (18) montrent ces comparaisons pour le couple d'images #21 dans les deux modes d'acquisition: DAPI et H2AX.

Cette évaluation est très subjective. Cependant, on remarque quelques modifications dans les images corrigées :

- Le contraste est beaucoup plus important, aussi bien dans les images DAPI et H2AX,
- de par leur intensité, un plus grand nombre de *foci* apparaissent plus facilement dans l'image H2AX,
- les pixels chauds ont disparus dans les images corrigées, mais les traces parasites ont rehaussées notamment dans l'image DAPI. Ces amas peuvent être un problème dans l'analyse ultérieure des images

Cette augmentation importante du contraste a en fait pour origine une correction apportée par ImageJ durant les calculs de conversion *(contrast stretching)*. Le rehaussement du contraste se produit au moment de la conversion des images 32-bit en 8-bit de chaque composante RGB par redéfinition de la dynamique.
Si on a une composante de couleur d'une image 32-bit dont la dynamique est restreinte à l'intervalle [0,210]. Comme le montre la figure (15), l'expansion de la dynamique par défaut lors de la conversion 32-bit en 8-bit, va repartir la dynamique de l'image brute sur les 256 niveaux de gris disponibles (correspondant à une image 8-bit)

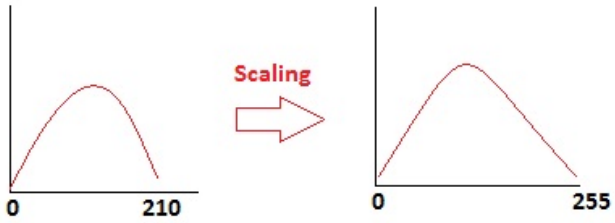


Figure (15) : Résultat de l'expansion de la dynamique par défaut lors de la fusion des composantes de couleur : l'histogramme de la composante est répartie sur toute la palette des niveaux de gris ; à gauche l'histogramme original, à droite l'histogramme redistribué

L'expansion de la dynamique est introduite par défaut lors de la fusion RGB pour former les images finales. Par contre les valeurs d'intensité restent inchangées lors de la conversion inverse; c'est le cas de la conversion de 8-bit en 32-bits.

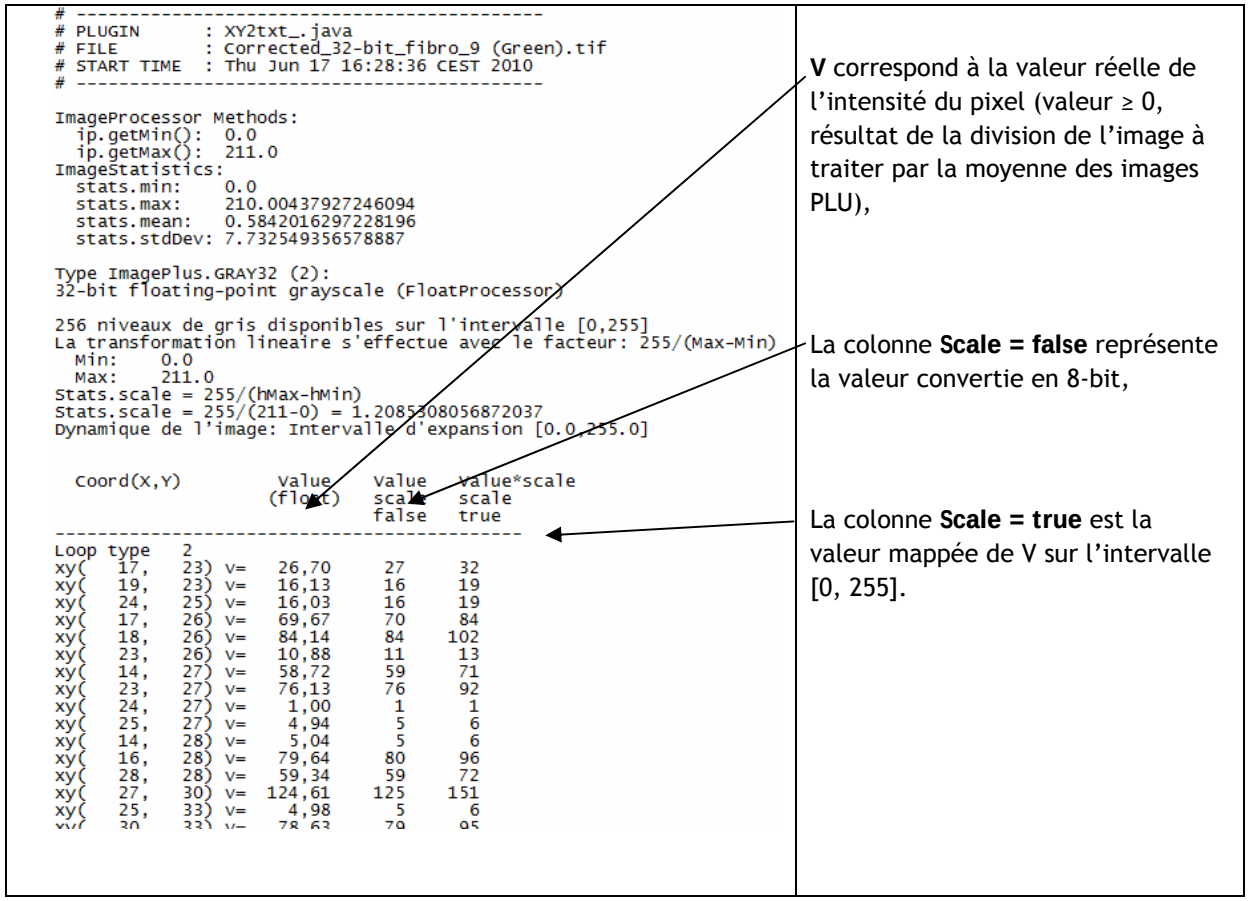
Lorsque le prétraitement de calibration est réalisé avec ImageJ, les résultats obtenus dépendent fortement des mécanismes de conversion d'images réalisés au cours du traitement et notamment lors de la fusion des composantes (en 32 bits) à l'image RGB (en 8 bits). Lors de cette étape, on a deux possibilités :

- Soit on fusionne les 3 images correspondantes à chaque composante en une image RGB en autorisant l'expansion de la dynamique, c'est-à-dire l'adaptation de la dynamique à la gamme. Cette méthode est l'option par défaut dans ImageJ,
- Soit on fusionne les 3 images correspondantes à chaque composante en une image RGB en conservant la dynamique d'origine.

L'expansion de la dynamique (« rescaling ») appliquée par défaut par ImageJ est une transformation linéaire simple:

$$M(V) = math.round\left(2^{profondeur\ de\ codage} - 1\right) \times \left[(V - min)/(max - min)\right]$$

Un plugin « *xy2txt* » a été écrit pour afficher les intensités (en mode grayscale ou RGB). Les sorties de ce plugin sont représentés sur la figure (16).



```
# -------------------------------------------
# PLUGIN      : XY2txt_.java
# FILE        : Corrected_32-bit_fibro_9 (Green).tif
# START TIME  : Thu Jun 17 16:28:36 CEST 2010
# -------------------------------------------

ImageProcessor Methods:
  ip.getMin():  0.0
  ip.getMax():  211.0
ImageStatistics:
  stats.min:    0.0
  stats.max:    210.00437927246094
  stats.mean:   0.5842016297228196
  stats.stdDev: 7.732549356578887

Type ImagePlus.GRAY32 (2):
32-bit floating-point grayscale (FloatProcessor)

256 niveaux de gris disponibles sur l'intervalle [0,255]
La transformation lineaire s'effectue avec le facteur: 255/(Max-Min)
  Min:    0.0
  Max:    211.0
Stats.scale = 255/(hMax-hMin)
Stats.scale = 255/(211-0) = 1.2085308056872037
Dynamique de l'image: Intervalle d'expansion [0.0,255.0]


  Coord(X,Y)          Value     Value   Value*scale
                     (float)    scale     scale
                                false     true
-----------------------------------------------
Loop type   2
xy(   17,   23) v=   26,70      27      32
xy(   19,   23) v=   16,13      16      19
xy(   24,   25) v=   16,03      16      19
xy(   17,   26) v=   69,67      70      84
xy(   18,   26) v=   84,14      84     102
xy(   23,   26) v=   10,88      11      13
xy(   14,   27) v=   58,72      59      71
xy(   23,   27) v=   76,13      76      92
xy(   24,   27) v=    1,00       1       1
xy(   25,   27) v=    4,94       5       6
xy(   14,   28) v=    5,04       5       6
xy(   16,   28) v=   79,64      80      96
xy(   28,   28) v=   59,34      59      72
xy(   27,   30) v=  124,61     125     151
xy(   25,   33) v=    4,98       5       6
```

V correspond à la valeur réelle de l'intensité du pixel (valeur ≥ 0, résultat de la division de l'image à traiter par la moyenne des images PLU),

La colonne **Scale = false** représente la valeur convertie en 8-bit,

La colonne **Scale = true** est la valeur mappée de V sur l'intervalle [0, 255].

```
# -------------------------------------------
# PLUGIN      : XY2txt_.java
# FILE        : RGB result (scale=false).tif
# START TIME  : Tue Jun 08 16:50:11 CEST 2010
# -------------------------------------------

 width   :  0
 height  :  0
ImageStatistics:
  stats.min:    0.0
  stats.max:    96.0
  stats.mean:   0.2318014705882353
  stats.stdDev: 3.132447319277818

Type ImagePlus.COLOR_RGB (4):
        32-bit RGB color (ColorProcessor)

 Coord(X,Y)   Channels(R,G,B)    Weight-   No Weighting
                                   ed     Factors(R+G+B)/3
-----------------------------------------------------------
xy(  17,  23) RGB(  4,  27,   0)    10      17
xy(  19,  23) RGB(  2,  16,   0)     6      10
xy(  24,  25) RGB(  3,  16,   0)     6      10
xy(  17,  26) RGB( 12,  70,   2)    28      45
xy(  18,  26) RGB( 15,  84,   4)    34      54
```

```
# -------------------------------------------
# PLUGIN      : XY2txt_.java
# FILE        : RGB result (scale=true).tif
# START TIME  : Tue Jun 08 16:50:49 CEST 2010
# -------------------------------------------

 width   :  0
 height  :  0
ImageStatistics:
  stats.min:    0.0
  stats.max:    249.0
  stats.mean:   0.4248161764705882
  stats.stdDev: 6.185602226659023

Type ImagePlus.COLOR_RGB (4):
        32-bit RGB color (ColorProcessor)

 Coord(X,Y)   Channels(R,G,B)    Weight-   No Weighting
                                   ed     Factors(R+G+B)/3
-----------------------------------------------------------
xy(  17,  23) RGB( 18,  32,   0)    17      24
xy(  19,  23) RGB(  7,  19,   0)     9      13
xy(  24,  25) RGB( 13,  19,   0)    11      15
xy(  17,  26) RGB( 48,  84,  31)    54      67
xy(  18,  26) RGB( 60, 102,  71)    78      86
```

Figure (16) : Outputs du plugin « *xy2txt* ». Les tableaux du bas représentent le contenu des pixels après fusion des composantes RGB sans expansion de la dynamique (à gauche) et avec (à droite).

Pour le premier pixel d'intensité > 0 et de coordonnées (17, 23), la valeur de l'intensité corrigée vaut 26.7. La profondeur de codage d'une image en 256 niveaux de gris nécessite 8 bits. La valeur mappée pendant l'étape de conversion réalisée par ImageJ avec l'option « *scaling=true* », est :

$$M(27,7) = 255 \times \left[(26,7 - 0) / (210 - 0) \right] = 32,4$$

,arrondi à 32

Dans le plugin « *PLU_RGB_Avg* », on choisit d'agir sur la dynamique dès la phase de conversion des images 32-bit en 8-bit, en autorisant ou non l'expansion de la dynamique par l'intermédiaire d'une commande sur le panel des paramètres du plugin (comme on peut le voir sur la figure (11)

Dans le cas où on fusionne les 3 composantes RGB en autorisant le rescaling de l'image, on obtient un effet artificiel d'augmentation du contraste. De plus cette augmentation de contraste va dépendre de la dynamique de l'image. Par exemple, dans le cas d'un histogramme couvrant l'intervalle [0, 250], il n'y aura quasiment pas d'augmentation puisque la composante est déjà à 250. Par contre pour les composantes de couleurs secondaires dont les moyennes sont en général faibles, le rescaling va augmenter artificiellement le poids de ces composantes
Il faut aussi prendre garde à cette manipulation qui aide à la compréhension des images mais qui ne convient pas si on doit quantifier les images (modification des images). En outre, l'utilisation a posteriori du rehaussement de contraste via le menu d'ImageJ (Image →Adjust → Brightness/Contraste) aura le même effet.
L'option de « *rescaling* », appliquée par défaut lors de la conversion des images 16 ou 32-bit en 8-bit, optimisera le contraste visuelle des images mais ne modifiera pas le rapport signal/bruit qui est comme nous le soulignions est le facteur principal pour l'analyse ultérieure des images.
En toute rigueur, il serait donc nécessaire d'interdire le rescaling au moment de fusionner les composantes pour créer l'image RGB finale.

Les figures (17) et (18) représentent les images DAPI et H2AX #21 avec l'expansion de la dynamique. Les figures (19) et (20) représentent les images DAPI et H2AX #21 sans l'expansion de la dynamique. La comparaison semble décevante à première vue. Il est vrai que dans ce cas, le microscope comme le CCD sont de bonne qualité et bien réglés. Cependant il y a toujours une inhomogénéité de la réponse du système (optique et CCD) qui doit être au moins évaluée pour montrer son importance. Par exemple les figures (21) montrent les images, converties en gris (sur 8 bits) des trois composantes de couleur (RGB) du masque obtenu à partir des images de PLU acquise sur 10 secondes. Dans la colonne de droite, on a représenté l'histogramme associé à chacune des composantes.
Le contraste et la luminosité ont été ajustés pour percevoir l'inhomogénéité des distributions d'éclairement. Outre l'assombrissement des coins haut-gauche et bas-droit et des bords de chacune des images, on voit par ailleurs que chaque image présente un aspect « granuleux » dont l'origine réside certainement dans les fluctuations statistiques résultant du temps de pose (10 secondes) avec lequel les PLU ont été acquises.
On remarque également sur les trois images, une « tache » sombre dans la partie basse-droite. Cette tache est présente dans toutes les images de PLU acquises lors de cette sessions de prise d'image.
Ce default est mis en évidence dans la figure (23) qui représente l'image moyenne des PLU acquises avec un temps de pose de 60 secondes ; la PLU moyenne a été convertie en gris sur 8 bits, puis l'intensité et le contraste on été ajusté pour faire ressortir le défaut optique
La mauvaise uniformité de la lumière (assombrissement sur les bords de l'image) apparait encore plus clairement sur cette image. La partie (b) de la figure 23 présente un grossissement de l'image autour du défaut ; on y voit clairement que ce défaut résultant d'un objet situé sur le chemin optique est relativement important en termes de nombre de pixels concernés. Ce défaut peut être une poussière située sur une des lentilles ou sur le filtre utilisé lors de la prise d'image. Elle apparait sous forme d'anneaux concentriques de diffraction comme on le voit sur ces images.
De plus ce grossissement fait apparaitre un pixel « mort ».La partie inférieure de l'image (23), représente le profil de l'intensité dans les pixels situés sur le chemin indiqué sur l'image présentée sur la partie (d). Ce chemin passant par le pixel « mort », on voit sur la partie (c) que l'intensité présente un creux brutal au passage de ce pixel

En conclusion le prétraitement a une portée réduite sur les images acquises à l'ESRF, du fait de la bonne qualité du capteur de la caméra associé au microscope. Mais elle peut avoir toute son importance dans le cas d'une mauvaise homogénéité du capteur associé au microscope. En tout état de cause, cette uniformité doit être vérifiée de manière systématique et doit faire partie des vérifications à réaliser avant toute prise d'images (calibration). En effet, ce prétraitement corrige systématiquement certains défauts de l'optique et permet d'uniformiser les images avant de les quantifier. De même, nous n'avons pas utilisé la correction de bruit thermique (offset des images) du fait de la bonne qualité de la CCD. De manière identique, il faudra effectuer la vérification du bruit thermique systématiquement avant l'acquisition massive d'image.

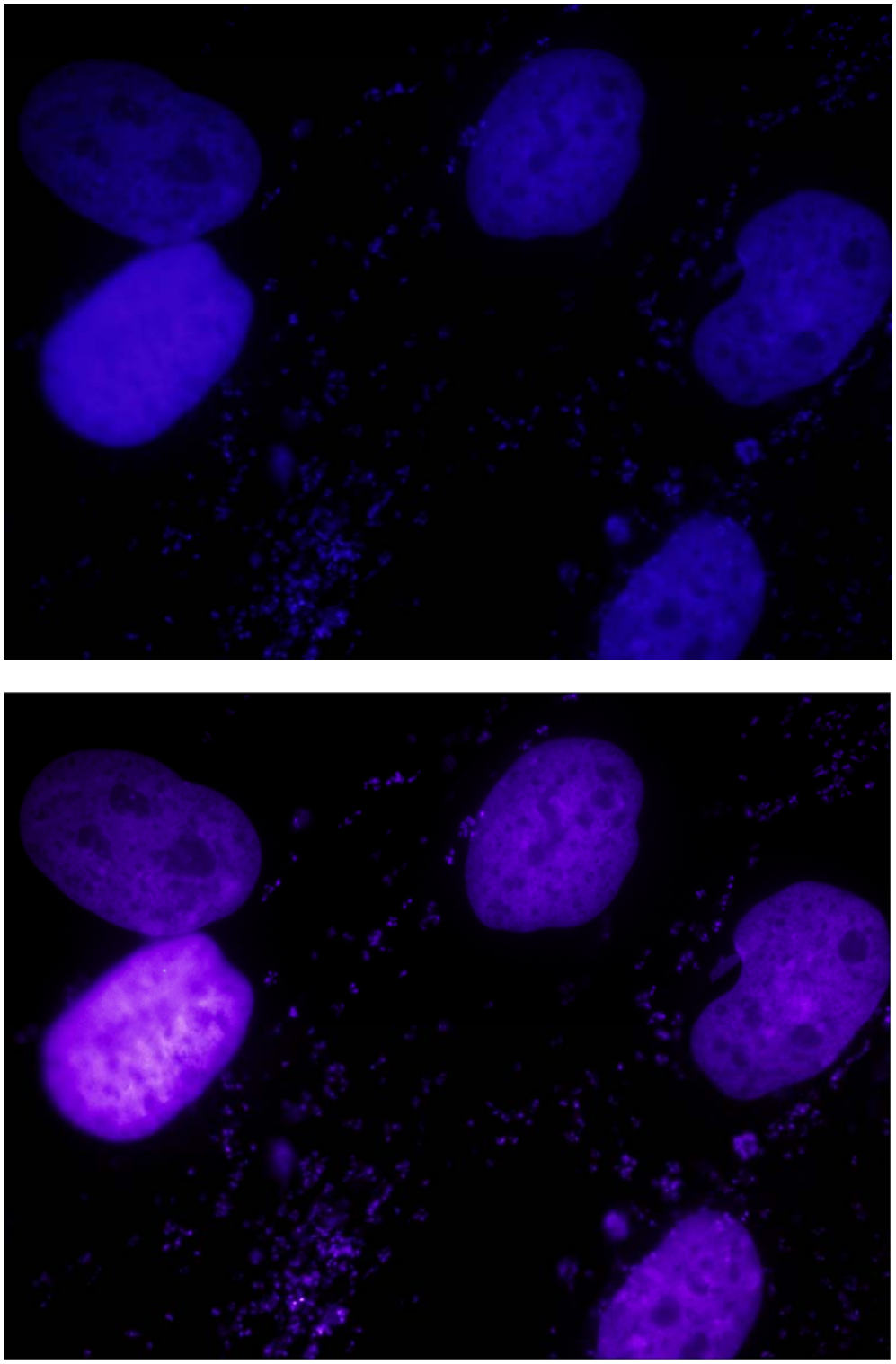


Figure (17) : Correction de l'image Dapi #21 en haute résolution: image originale (en haut) et image corrigée (en bas). L.es images sont corrigées en utilisant trois images PLU (temps d'exposition de 10s). L'option de rescaling, appliquée par défaut lors de la conversion des images 32-bit en 8-bit, optimise le contraste visuel des images mais ne modifiera pas le rapport signal/bruit.

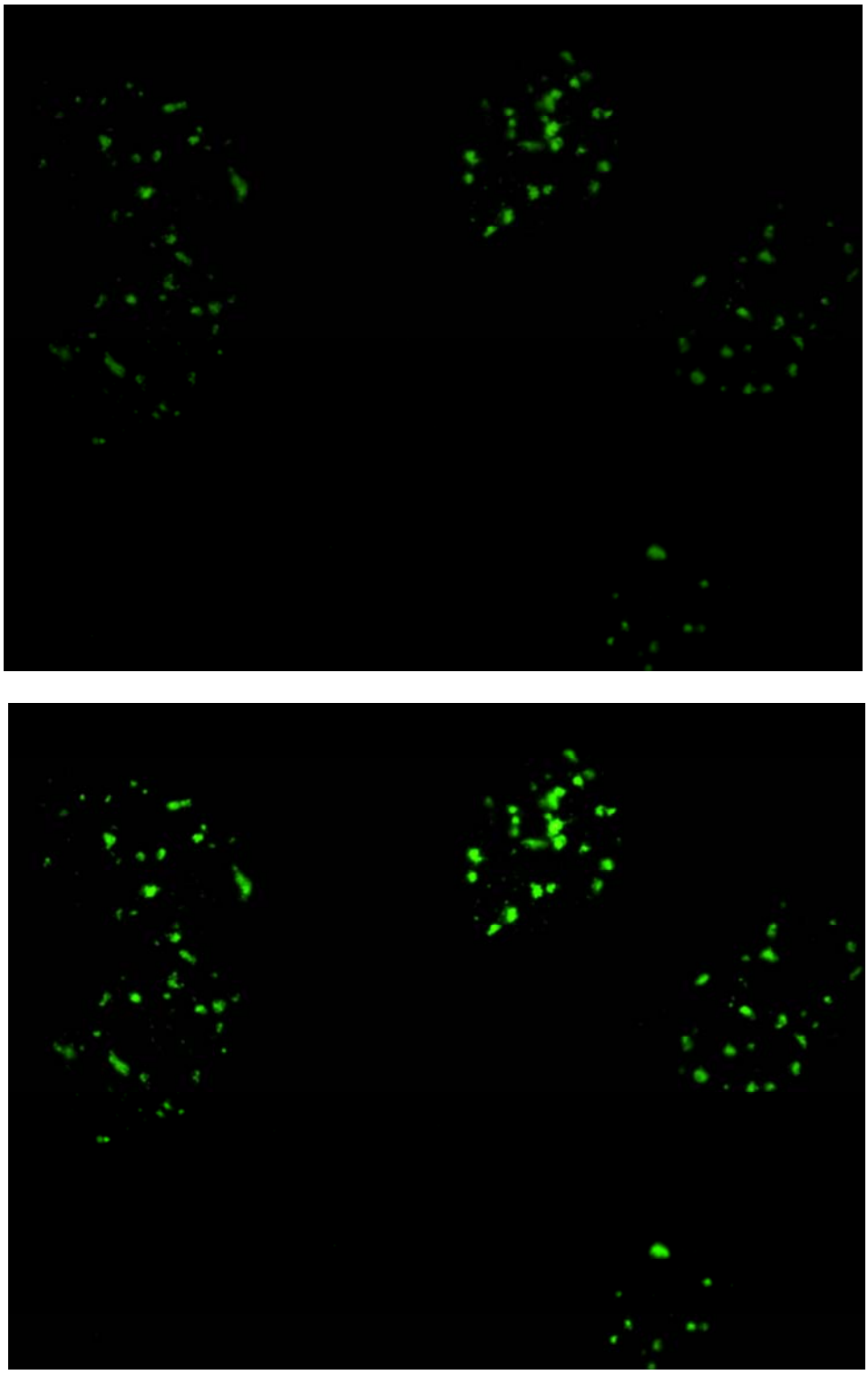


Figure (18) : Correction de l'image H2AX #21 en haute résolution: image originale (en haut) et image corrigée (en bas). Les images sont corrigées en utilisant 2 images de PLU (temps d'exposition de 60s). L'option de rescaling, appliquée par défaut lors de la conversion des images 16 ou 32-bit en 8-bit, optimise le contraste visuel des images mais ne modifiera pas le rapport signal/bruit.

.

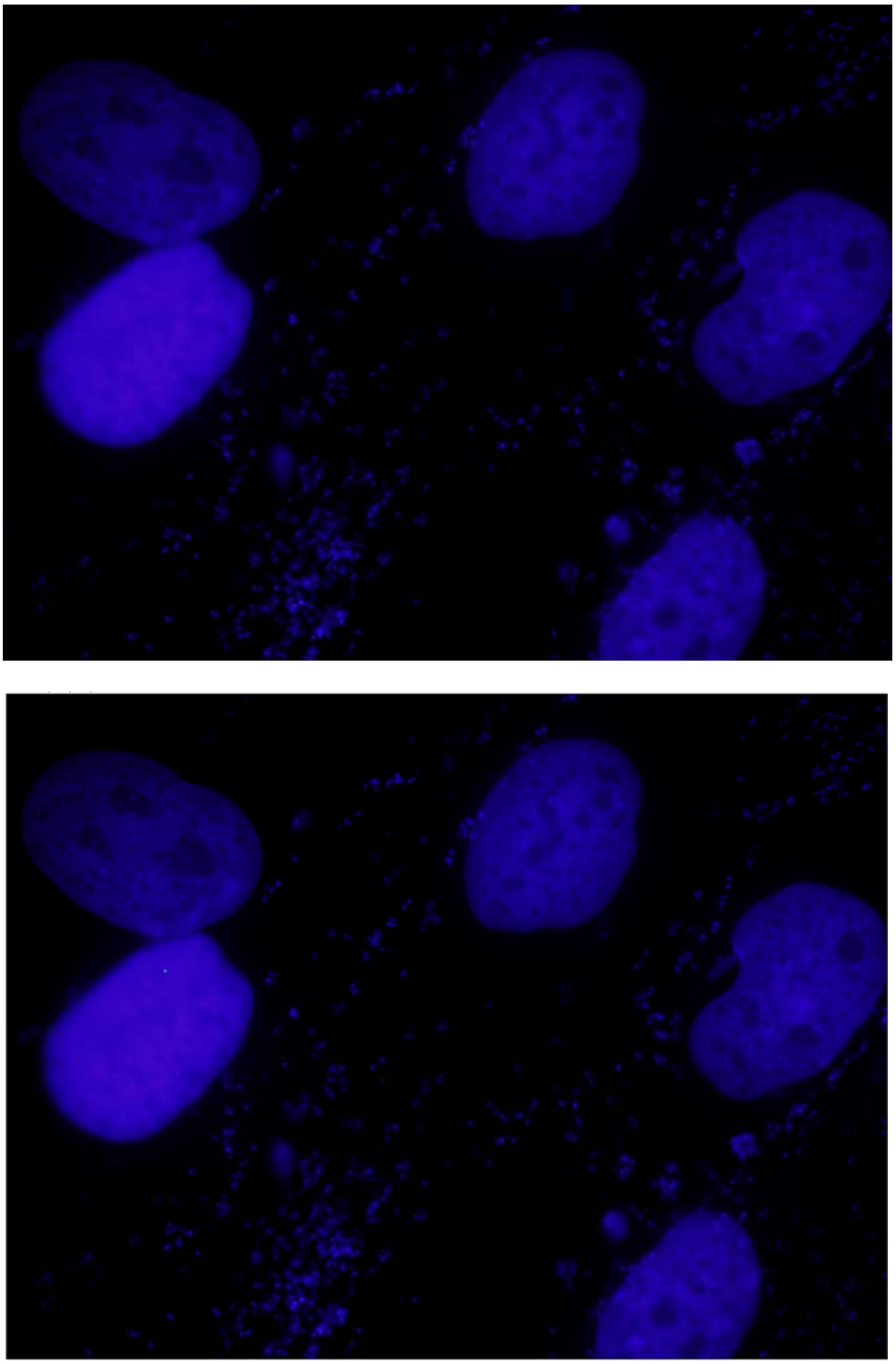


Figure (19) : Image Dapi #21 en haute résolution: l'image originale (en haut) et l'image corrigée (en bas). L'option de rescaling n'a pas été appliquée pour corriger l'image. Le prétraitement a simplement eu pour effet (non perceptible à cette échelle) de supprimer les « pixels chauds ».

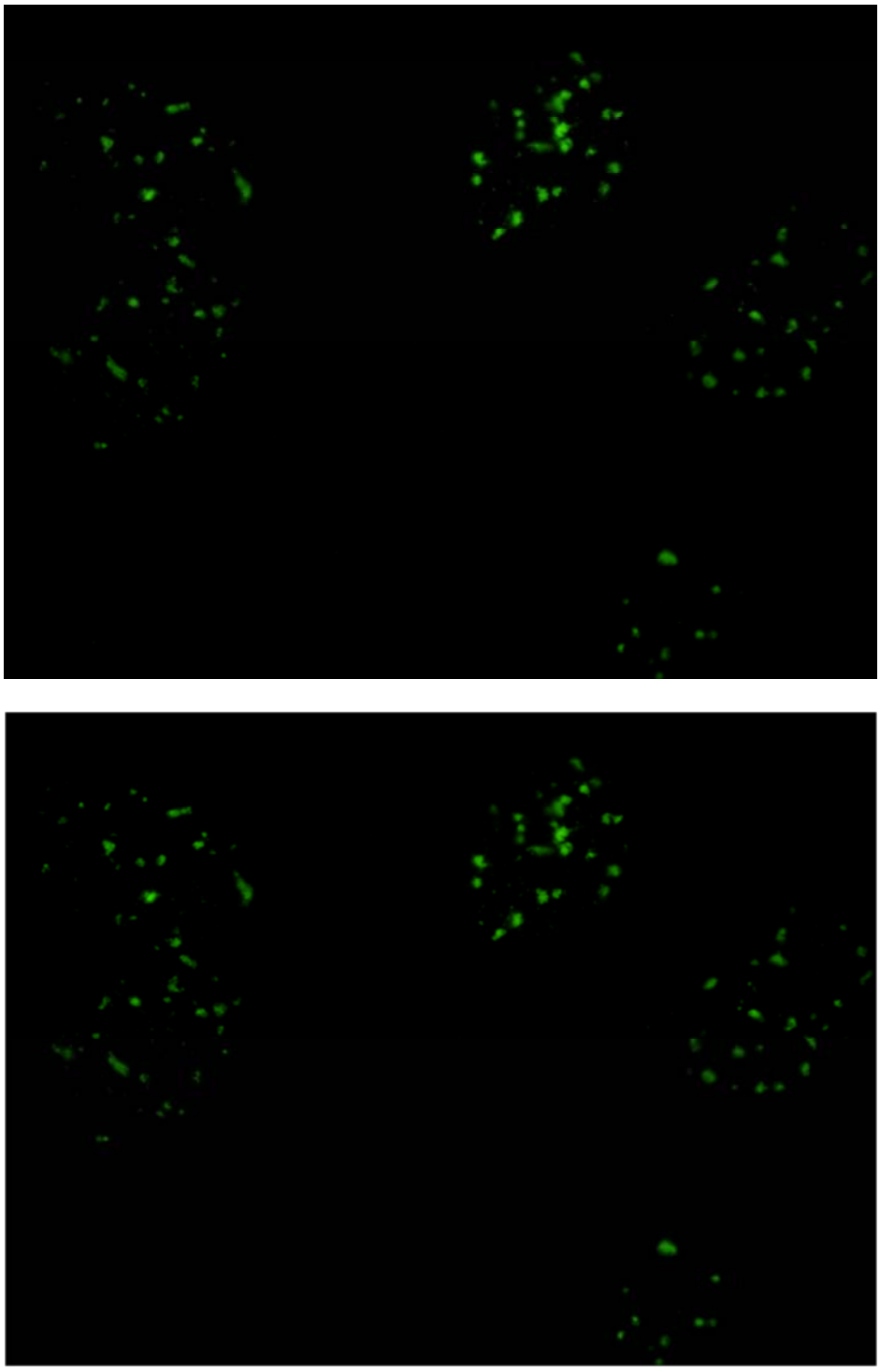
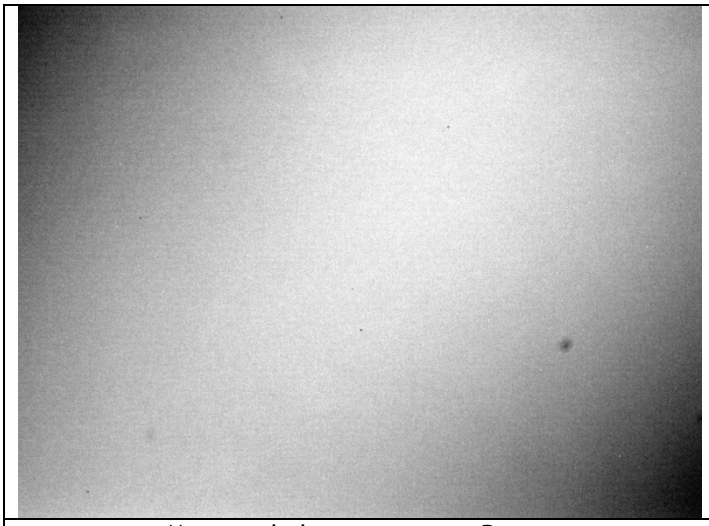
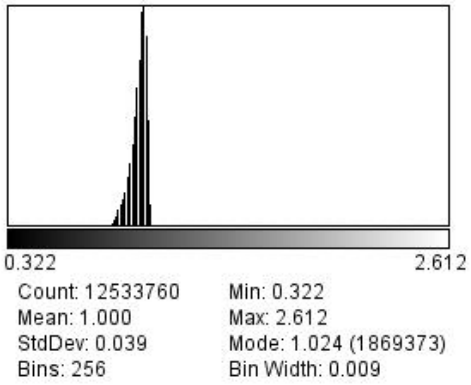
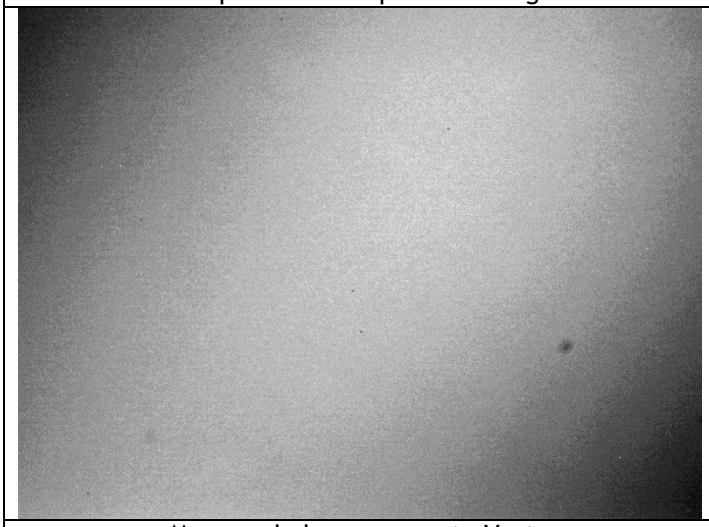


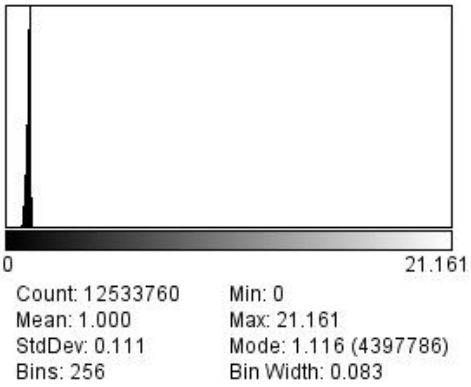
Figure (20) : Image H2AX #21 en haute résolution: l'image originale (en haut) et l'image corrigée (en bas). L'option de rescaling n'a pas été appliquée pour corriger l'image. Le prétraitement a simplement eu pour effet (non perceptible à cette échelle) de supprimer les « pixels chauds ».



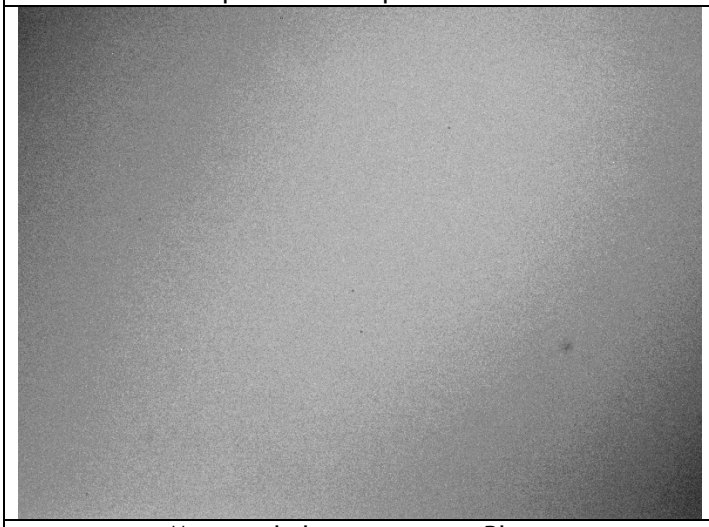


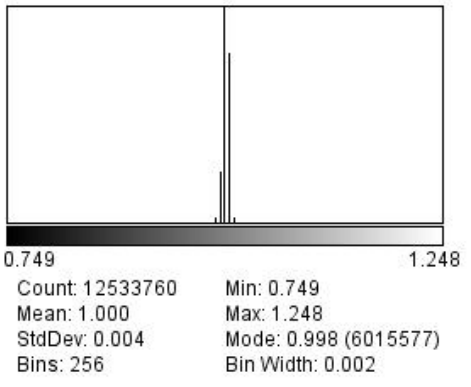
Masque de la composante Rouge	Histogramme





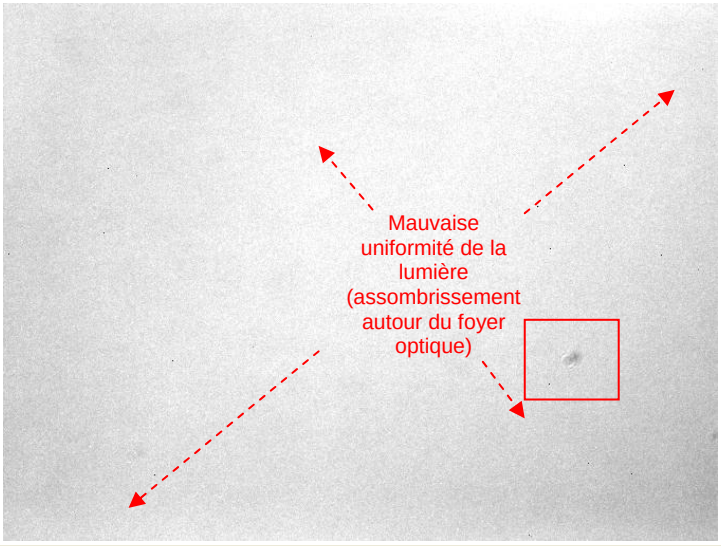
Masque de la composante Verte	Histogramme



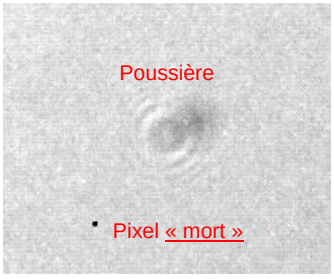


Masque de la composante Bleue	Histogramme

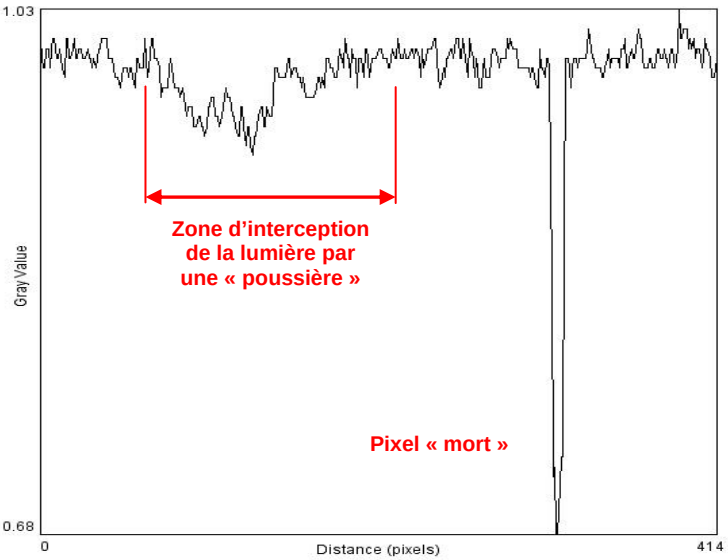
Figure (22) : A gauche, les masques des composantes RGB obtenus à partir d'une pile de 3 images de PLU projetées par la méthode de la moyenne (tps d'exposition de 10s). Le contraste et la luminosité ont été ajustés pour percevoir l'inhomogénéité des distributions d'éclairement. A droite, l'histogramme de chaque composante est présenté.



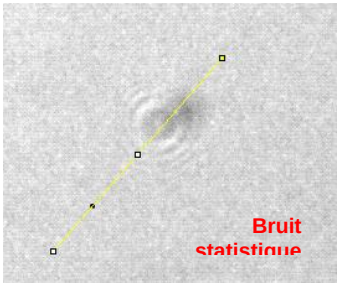
a) Aspect de la PLU moyennée.



b) Zoom sur deux types d'obstacle à la lumière.



c) Zoom sur l'interception de la lumière par une poussière.



d) Coupe sur les défauts de la fenêtre de zoom.

Figure (23) - Défauts optiques du capteur :pixel « mort » et « poussière » présente sur le chemin optique du microscope.

Description de la macro de comptage des *foci* par noyau

L'objectif étant de faire des quantifications sur les images des cellules irradiées, l'analyse des images doit consister à :

- reconnaitre les noyaux dans les images,
- reconnaitre les *foci* dans les noyaux pour caractériser leurs caractéristiques : intensité, surface, périmètre, variables de forme, etc).

Cet outil d'analyse automatique de reconnaissance et de mesure des *foci* permettra d'obtenir une liste des *foci* par noyaux qui sera utilisée ensuite pour extraire les informations statistiques sur la topologie des *foci* .Les algorithmes de traitement d'images sont inclus dans une macro ImageJ.

Cette macro utilise des sélections de partie des images au moyen de zone d'intérêt (« Region Of Interest » ou ROI dans le formalisme d'ImageJ).

Une ROI est donc une partie de l'image sur laquelle sera effectuée des opérations ou des mesures; le reste de l'image étant ignoré.
On peut définir une ROI par la création d'un masque binaire, qui est une image binaire de la même taille que l'image originale, mais pour laquelle les pixels contenus dans la ROI sont à « 1 » et les pixels hors de la ROI sont à « 0 ». On définit plus couramment les ROI par l'intermédiaire d'un seuillage sur l'intensité associée à chaque pixel : si l'intensité est au delà du seuil, les pixels sont considérés comme appartenant à la ROI (pixels à « 1 »), les pixels dont l'intensité est inférieure au seuil sont considérés comme n'appartenant pas à la ROI (pixels à « 0 »).
Les pixels d'une ROI peuvent définir une zone continue, c'est le cas lorsque la ROI est définit géométriquement par une forme fermée (polygone, cercle, rectangle..). Dans le cas d'un seuillage, la ROI peut être formé par des groupes de pixels non voisins.
On peut définir plus d'un ROI dans une image; toute les ROIs définies sont gérées par le « ROI Manager » d'ImageJ

La méthode de sélection des *focis* repose sur l'utilisation de couples d'images associées ; l'image DAPI permet de définir la ROI correspondant aux noyaux cellulaires ; l'image H2AX associée permet ensuite de définir les *foci* dans la ROI définissant ces noyaux cellulaires. Il est donc nécessaire, au préalable, de disposer d'un dossier contenant deux sous-dossiers *obligatoirement* nommés « DAPI » et « H2AX ». Dans les deux dossiers 'DAPI' et 'H2AX, les images à analyser DAPI et H2AX doivent être appariées correctement dans le même ordre comme le montre la figure (24).

Dossier images DAPI	Dossier images H2AX
fibro_dapi_100x_HR_1.tif	fibro_h2ax_100x_HR_1.tif
fibro_dapi_100x_HR_2.tif	fibro_h2ax_100x_HR_2.tif
fibro_dapi_100x_HR_3.tif	fibro_h2ax_100x_HR_3.tif
fibro_dapi_100x_HR_4.tif	fibro_h2ax_100x_HR_4.tif
fibro_dapi_100x_HR_5.tif	fibro_h2ax_100x_HR_5.tif
fibro_dapi_100x_HR_6.tif	fibro_h2ax_100x_HR_6.tif
fibro_dapi_100x_HR_7.tif	fibro_h2ax_100x_HR_7.tif
fibro_dapi_100x_HR_8.tif	fibro_h2ax_100x_HR_8.tif
fibro_dapi_100x_HR_9.tif	fibro_h2ax_100x_HR_9.tif
fibro_dapi_100x_HR_10.tif	fibro_h2ax_100x_HR_10.tif
fibro_dapi_100x_HR_11.tif	fibro_h2ax_100x_HR_11.tif
fibro_dapi_100x_HR_12.tif	fibro_h2ax_100x_HR_12.tif

Figure (24) : Répertoire 'DAPI' et 'H2AX contenant les images appariées, qui vont être utilisées par la macro pour l'analyse

Donc dans un premier temps, l'image DAPI est convertie sur 8 bits, en niveau de gris. Dans le cas présent, on s'est contenté de la conversion standard dans ImageJ ; c'est-à-dire *gray=0.299red+0.587green+0.114blue*

Puis l'image est utilisée pour définir, par seuillage sur l'intensité des pixels de l'image DAPI, des masques binaires définissant potentiellement les noyaux cellulaires. Le réglage du niveau de seuillage est ajusté automatiquement d'après l'histogramme de l'image sélectionnée (méthode « SetAutoThreshold() »). Les ROI potentiels sont sélectionnés selon des caractéristiques de configuration qui sont mesurés sur les masques. Le choix de la mesure de ces caractéristiques est prédéfini dans la macro par « run("Set Measurements...", "area mean centroid redirect="…" decimal=3 circularity perimeter fit") ». Parmi la liste des caractéristiques, la macro mesure :

- «*area*»: Surface du masque (noyau ou *foci*) dans l'unité définie par la calibration spatiale des images. Dans notre cas, les unités sont des « inches » avec la conversion de 200 pixels par « inch ».
- «*mean*»: Intensité moyenne (en valeurs de gris) sur la surface du masque sélectionné ; c'est-à-dire la somme des intensités de gris associés à chaque pixel, divisée par le nombre de pixels.
- «*centroid*»': calcul des coordonnées du centre de gravité du « centroid ». Le calcul du centre de masse peut être pondéré par les intensités des pixels. Les unités utilisées pour exprimer ces coordonnées sont des pixels
- « *circularity* »: critère géométrique dont le calcul utilise la formule suivante,

$$Circularity = 4\pi\left(\frac{aera}{perimeter^2}\right)$$

Une valeur de circularité de 1 correspond à une forme circulaire; alors qu'une circularité de 0 indique plutôt un polygone allongé (un trait)

Les critères utilisés pour la sélection des ROI des noyaux peuvent être ajusté par l'utilisateur au cours de l'exécution de la macro par l'intermédiaire d'un panneau comme le montre la figure (25) Pour ce qui concerne les masques associés au noyau, les critères de sélection sont les suivants

- « *aera* » supérieure à 10000 pixels
- « *circularity* » comprise entre 0 et 1

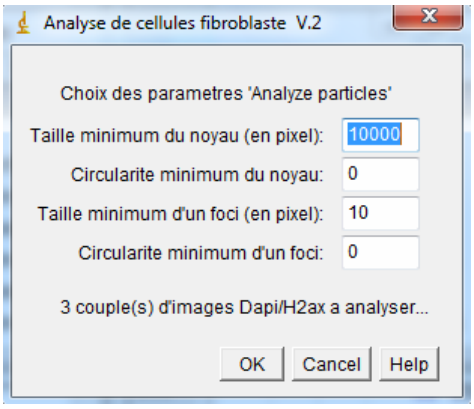
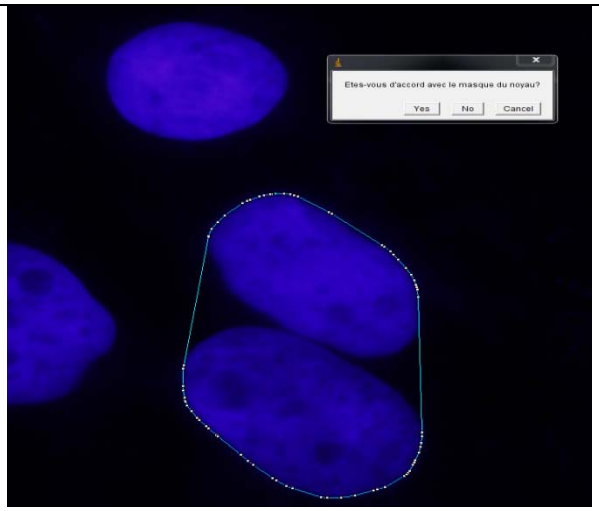


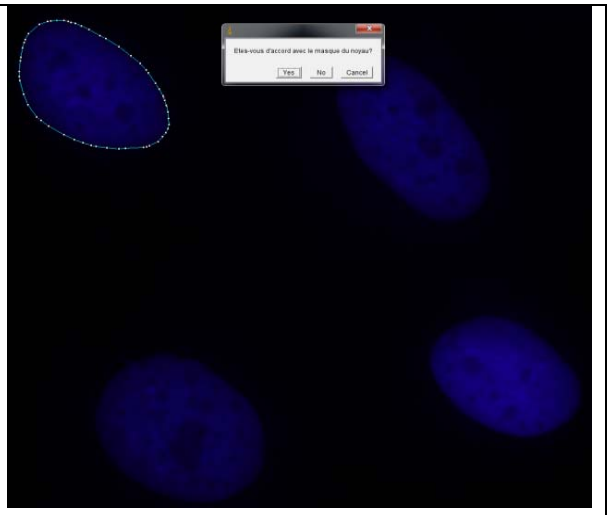
Figure (25) : Détermination des paramètres de l'analyse pour les masques des noyaux et des *foci* : seuil sur l'aire du masque, paramètre de circularité

Après cette première sélection, on obtient une liste de masques qui définissent les ROI correspondant aux noyaux. L'utilisateur est alors invité à valider manuellement chacune de ces sélections, et éventuellement refuser une sélection inappropriée.

La figure (26) représente sur la partie gauche une mauvaise sélection du fait de deux noyaux adjacents, et sur la partie droite une bonne sélection.



Rejet de la sélection sur l'image DAPI



Un masque de noyau acceptable

Figure (26) : deux exemples de masque définissant les noyaux cellulaires : la sélection de gauche est annulée par l'utilisateur, alors que la sélection de la partie droite est validée

Les sélections des ROI des noyaux sont renommées (préfixe "Cell_**")**. L'ensemble des ROI des noyaux validés sont sauvegardé dans un fichier «*fibro_h2ax_100x_HR_Y.tif_ROIs.zip* » ou *Y* est le numéro de l'image. Chacun des masques validés est également sauvegardé dans un fichier séparé «*fibro_h2ax_100x_HR_Y.tif_ROIs_Cell_*X *.zip* » ou *X* est le numéro du noyau dans l'image

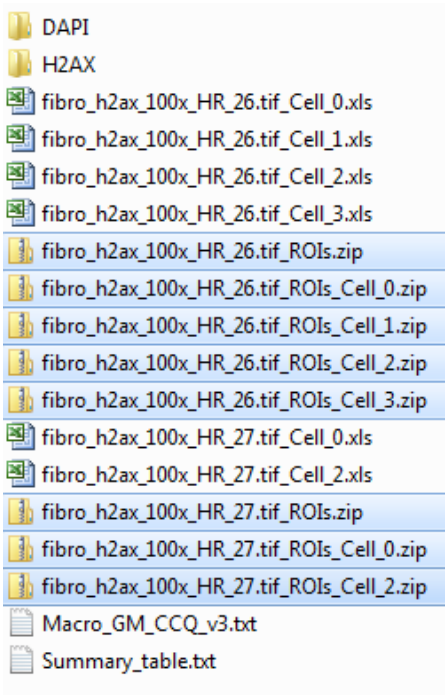


Figure (27) : Résultats sauvegardés dans le répertoire de travail

Les « ROI » contiennent toutes les informations correspondantes aux masques. Cette sauvegarde permet de vider le ROI Manager de tout masque avant de commencer la deuxième phase du traitement.

La deuxième phase consiste à définir des *focis* par la création de masques correspondant à l'ensemble des pixels représentant un *foci* dans l'image H2AX. Les masques de *foci* sont obtenus par un seuillage automatique de l'image avec la méthode « *SetAutoThreshold* ».
Les *foci* potentiels sont également mesurés pour effectuer un filtrage selon la valeur de l'aire et de la circularité du masque correspondant. Pour ce qui concerne les masques associés au *foci*, les critères de sélection sont les suivants

- « aera » supérieure à 10 pixels
- « circularité » comprise entre 0 et 1

Les figures (28) et (29) présentent les masques définis pour un noyau cellulaire à partir de l'image DAPI, et les masques correspondants aux focis dans ce noyau cellulaire, déterminés dans l'image H2AX correspondante en utilisant le masque du noyau pour limiter la recherche des focis.

En fin d'analyse, les résultats des analyses (mesures de surface, périmètre, position et ratio divers) des *foci* par noyau sont placés dans des fichiers « .xls » dans le répertoire parent.

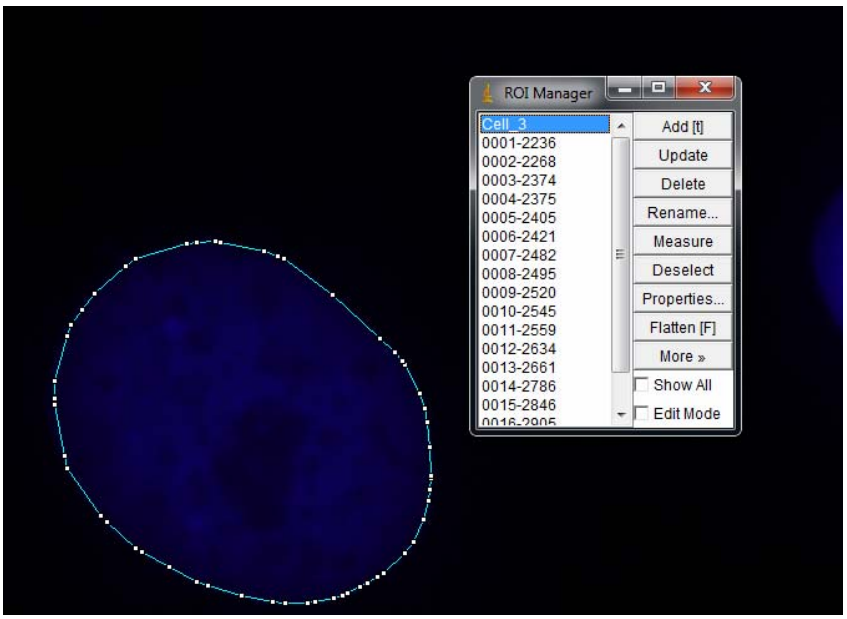


Figure (28) : Présentation de l'enveloppe convexe du noyau

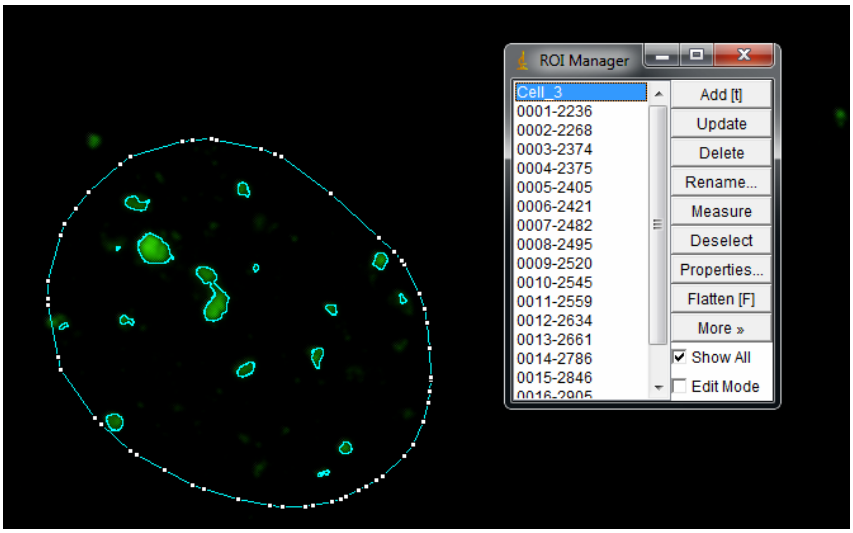


Figure (29) : Présentation de tous les *foci* détectés avec les préréglages choisis

Slice	Count	Total Area	Average Size	Area Fraction	Mean	Perim.	Major	Minor	Angle	Circ.	Solidity
Mask_foci.tif	24	0.389	0.016	3.5	29.757	0.604	0.157	0.111	98.810	0.474	0.811
Mask_foci.tif	25	0.346	0.014	2.6	29.968	0.543	0.137	0.101	88.915	0.455	0.776
Mask_foci.tif	14	0.199	0.014	1.9	28.383	0.593	0.153	0.103	77.609	0.474	0.802
Mask_foci.tif	17	0.403	0.024	2.7	29.414	0.757	0.186	0.115	98.358	0.384	0.747
Mask_foci.tif	37	0.718	0.019	8.2	36.257	0.598	0.163	0.112	99.195	0.532	0.823
Mask_foci.tif	17	0.329	0.019	2.6	42.142	0.663	0.176	0.117	66.327	0.468	0.832

Figure (30) : Synthèse des résultats (Nombre de *foci* par noyau)

La macro se termine par l'affichage de la distribution du nombre de *foci* par noyau:

- Sur l'axe X, le nombre de *foci* par noyau,
- Sur l'axe Y, le nombre de noyaux candidats.

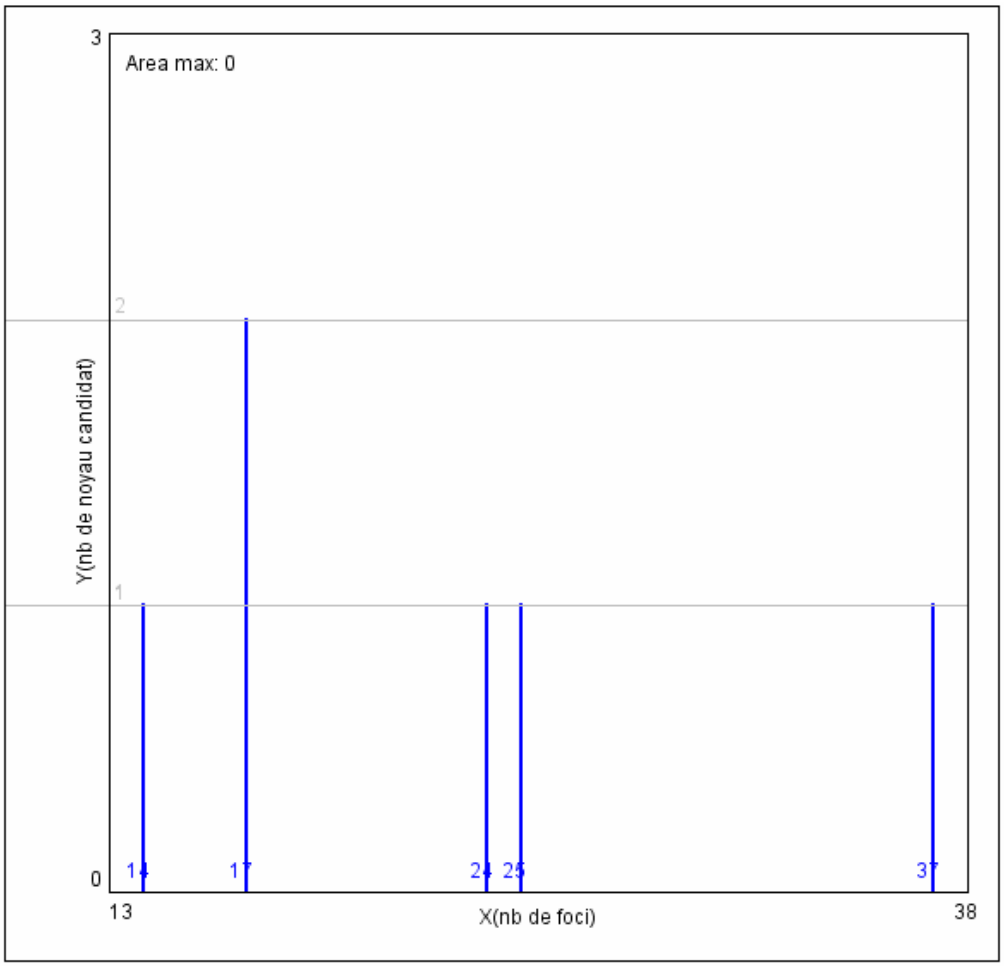


Figure (31) : Plot du graphe de fréquence

Pour l'image #26, le premier noyau retenu (cell_0) contient l'ensemble de mesures des 24 *foci* du noyau:

	Area	Mean	X	Y	Perim.	Major	Minor	Angle	Circ.	AR	Round	Solidity
1	0.002	23.355	2.508	0.845	0.231	0.051	0.039	122.412	0.365	1.296	0.771	0.614
2	0.038	38.827	2.886	0.994	1.048	0.279	0.171	48.087	0.430	1.628	0.614	0.871
3	0.012	27.382	3.358	1.613	0.601	0.155	0.097	4.980	0.412	1.593	0.628	0.806
4	0.019	37.639	2.386	1.784	0.712	0.198	0.122	92.794	0.472	1.618	0.618	0.861
5	0.018	24.526	4.699	2.007	0.776	0.243	0.097	114.647	0.386	2.519	0.397	0.784
6	0.004	24.226	2.146	2.001	0.284	0.076	0.071	140.980	0.655	1.068	0.937	0.851
7	0.025	32.334	2.364	2.548	0.787	0.188	0.166	111.105	0.498	1.133	0.882	0.875
8	0.038	42.799	2.767	2.667	1.097	0.285	0.168	37.466	0.392	1.698	0.589	0.856
9	0.052	41.440	4.751	2.709	1.086	0.269	0.244	158.153	0.549	1.105	0.905	0.902
10	0.001	22.578	5.167	2.626	0.150	0.042	0.034	126.785	0.627	1.221	0.819	0.789
11	0.014	30.172	5.345	3.067	0.611	0.170	0.109	73.221	0.488	1.557	0.642	0.869
12	0.010	26.540	5.002	3.112	0.485	0.145	0.085	161.664	0.517	1.711	0.585	0.830
13	0.010	25.413	2.711	3.265	0.491	0.139	0.093	118.062	0.531	1.496	0.669	0.817
14	0.008	24.660	2.340	3.279	0.544	0.142	0.076	175.705	0.358	1.868	0.535	0.766
15	0.011	30.085	3.214	3.410	0.510	0.125	0.110	38.663	0.524	1.136	0.880	0.812
16	0.014	29.765	2.304	3.455	0.615	0.157	0.117	8.372	0.479	1.333	0.750	0.823
17	0.012	25.939	4.719	3.563	0.529	0.135	0.112	162.038	0.536	1.201	0.832	0.824
18	0.006	25.155	2.664	3.643	0.505	0.093	0.086	156.081	0.311	1.087	0.920	0.747
19	0.008	22.917	2.928	3.827	0.390	0.114	0.091	102.325	0.674	1.263	0.792	0.853
20	0.009	27.140	5.435	3.893	0.581	0.145	0.077	60.677	0.325	1.880	0.532	0.761
21	0.001	21.589	3.869	3.938	0.269	0.050	0.035	20.700	0.243	1.426	0.701	0.544
22	0.037	39.146	4.651	4.136	0.989	0.230	0.205	115.102	0.476	1.122	0.891	0.875
23	0.034	44.258	5.105	4.196	0.875	0.243	0.180	46.189	0.565	1.349	0.741	0.899
24	0.005	26.272	3.705	4.199	0.330	0.085	0.073	175.243	0.561	1.167	0.857	0.839

Figure (32) : fibro_h2ax_100x_HR_26.tif_Cell_0.xls

Analyse des images H2AX acquises à l'ESRF

La macro de reconnaissance des *foci* a été utilisée pour analyser les 31 images de cellule fibroblastes, irradiées à 1 Gray sur le faisceau de rayon X de l'ESRF. Compte tenu que le prétraitement des images n'apporte pas une amélioration importante dans le cas présent, à la reconnaissance des *foci* dans les cellules, nous avons utilisé les images originales acquises en haute résolution.

Après une validation visuelle des noyaux, nous obtenons un lot de 83 noyaux, contenant 1785 *foci*. Les données obtenues pour chaque noyau dans chaque cellule sont stockées dans un fichier séparé de type « excel ». Ce fichier est converti en fichiers « texte », stockés dans un sous répertoire du répertoire de travail. Un programme en C examine le contenu de ce sous répertoire pour combiner l'ensemble des données et les utiliser pour calculer et représenter par l'intermédiaire du logiciel ROOT les caractéristiques de ce lot de cellules.

Les figures (33) représentent la distribution du nombre de *foci* pour l'ensemble des 83 noyaux. On obtient une valeur moyenne du nombre de *foci* par noyau de 20.75 avec un sigma de 6.96. La partie gauche de la figure est la distribution de l'intensité totale des focis dans les noyaux ; intensité calculée comme la somme des produits des surfaces de chaque *foci* (en pixel carré) multiplié par l'intensité moyenne de ce même *foci*. Cette distribution présente un pic marqué pour une valeur de 6-7 . Ce pic est dominé par des *foci* de petite surface.

En effet comme le montre la figure (34), 350 des 1785 focis ont une surface inférieure ou égale à 100 pixel-carré (soit ~20 % de l'ensemble des pixels). L'intensité moyenne est relativement bien définie : 31 avec un sigma de 10. Par contre la forme de ces *foci* est très variable car comme le montre la distribution de la circularité de ceux-ci, cette forme varie depuis un cercle presque parfait (circularité proche de 1), à une forme relativement allongée (circularité proche de 0).

$$Circularity = 4\pi\left(\frac{aera}{perimeter^2}\right)$$

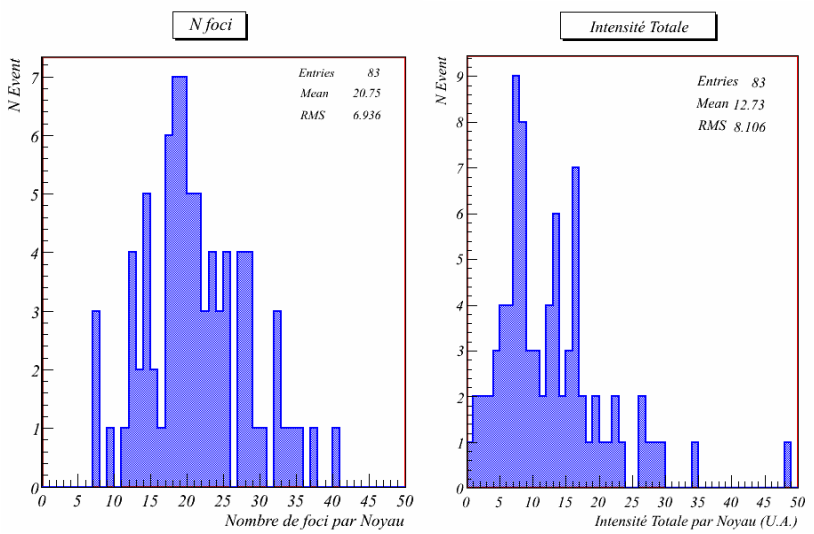


Figure (33) : Distribution du nombre de *foci* par noyau (à gauche), et de l'intensité totale de ces focis dans les noyaux (à droite)

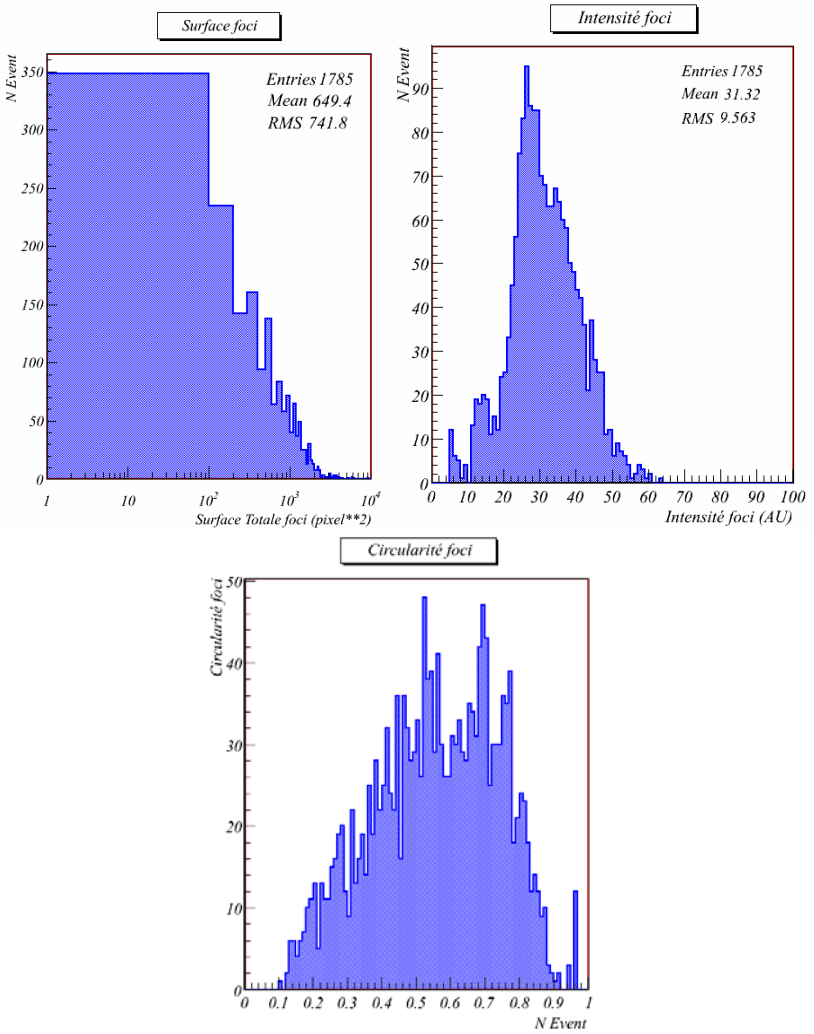


Figure (34) : Distribution de la surface des 1785 *foci* en pixel-carré (à gauche), et de leur intensité moyenne (à droite). L'histogramme du bas représente la distribution de la circularité des *foci*.

L'analyse de la topologie des focis dans les noyaux peut être faite à partir des coordonnées en x et y de ces *foci*. Dans un premier temps, nous avons simplement calculé la valeur moyenne des

distances entre le centroid des positions des focis à l'intérieur du noyau. Une fois les coordonnées du centroid calculées, on se sert de celle-ci pour déterminer la distance carré moyenne entre ce centroid et tous les focis. La distribution de la racine carrée de cette variable est représentée sur la figure (35). La sensibilité de cette variable de dispersion n'est pas suffisante pour caractériser la distribution des focis dans les noyaux, il faut donc faire appel à d'autres types de variables.

La figure (35) représente le diagramme de corrélation entre l'intensité totale des focis mesurée dans les noyaux et le nombre de *foci* détectés. Une corrélation existe car il est possible de définir à partir du nuage de point une ligne de corrélation. Cependant cette corrélation n'est pas aussi marquée que l'on pourrait attendre.

La figure (36) représente le diagramme de corrélation entre l'intensité totale des *foci* mesurés dans les noyaux et la dispersion des *foci* détectés. Il n'y a apparemment aucune corrélation. Cependant cette variable de dispersion ne semble pas assez sensible et significative pour mesurer une quelconque variation.

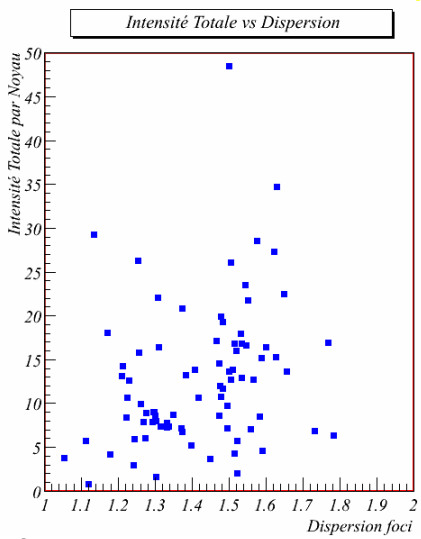


Figure (35) : Diagramme de corrélation entre la dispersion des *foci* (sur l'axe des abscisses), et l'intensité totale (sur l'axe des ordonnées) dans les 83 noyaux mesurés

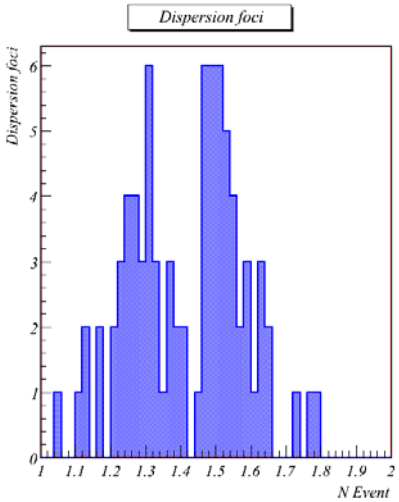


Figure (36) : Distribution de la dispersion des *foci* dans les noyaux. La définition de cette dispersion est expliquée dans le texte

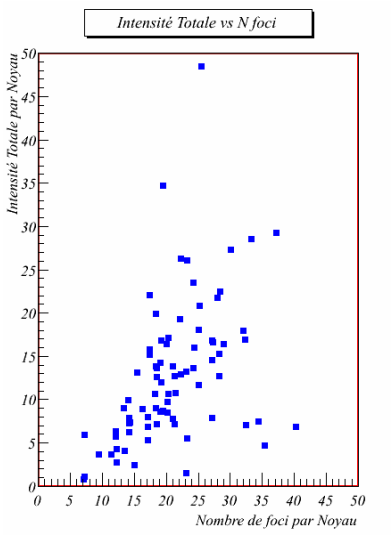


Figure (37) : Diagramme de corrélation entre le nombre de *foci* (sur l'axe des abscisses), et l'intensité totale (sur l'axe des ordonnées) dans les 83 noyaux mesurés

Conclusion et perspectives

Cette étude avait pour but d'examiner sur lot d'images obtenues par microscopie à épifluorecence, significatives des images susceptibles d'être traitées dans le cadre du projet ROSIRIS. Puis de montrer quelles pouvaient être les difficultés rencontrées pour effectuer l'analyse de ces images dans le but d'obtenir des informations pertinentes sur les processus biologiques découlant de l'irradiation de cellule.

Dans un premier temps, nous avons décrit succinctement les principes de fonctionnement des éléments matériels utilisés :

- Microscope Olympus BX51 (Annexe A),
- CCD Olympus DP70 (Annexe B),

en vue de comprendre quelle était les divers facteurs susceptible d'altérer la qualité des images.

Cette analyse été abordée selon le principe de calibration nécessaire à tout appareillage de mesure. Nous avons donc envisagé une phase préalable de prétraitement (calibration) qui devrait être effectuée simultanément à toute phase d'acquisition d'image. Bien sur cette phase de prétraitement est fortement dépendante du matériel employé et peut demander un travail de mesure très important (quantimétrie). Le travail présenté dans ce document n'est qu'une première ébauche destiné à cerner les points les plus importants.

A partir d'un certain nombre de prise d'images réalisées lors de l'acquisition des images cellulaires avec le microscope à épifluorecence dans des conditions particulières : illumination uniforme, image « noire », image d'offset… Nous avons réalisé quelques prétraitements.

Le logiciel utilisé pour ces travaux est le logiciel ImageJ dont les capacités et les performances en font l'outil indispensable pour l'analyse des images. Ce travail nous a donc permis de réaliser l'apprentissage de ce logiciel et de la programmation des macros utilisables dans ce contexte.

Nous avons montré que le matériel utilisé dans le cadre de cette expérience était de bonne qualité et relativement bien réglé : peu de pixels mort, non uniformité relativement faible. De sorte que les images originales étaient déjà de bonne qualité. Une amélioration significative des images nécessiterait un travail beaucoup plus important de quantimétrie : mesure de la non-linéarité, mesure de la PSF.

A partir du batch de 31 couples d'images (DPAI et H2AX) ; ce qui représente environ une centaine de noyaux cellulaires, nous avons mis au point une première macro d'analyse, écrite en langage ImageJ.

L'efficacité de reconnaissance de cette macro est remarquable lorsque l'on valide visuellement les résultats de la recherche de focis sur les images originales. Cette macro nous a permis de découvrir les difficultés inhérentes à une analyse automatique d'une grande quantité d'images.

En conclusion, ce travail précurseur à une plateforme d'exploitation et de traitement d'image dans le contexte du projet ROSIRIS nous a permis de comprendre les difficultés de ce type de projet Un modèle générique du processus complet du traitement des images, depuis l'acquisition des images jusqu'a l'interprétation finale et l'évaluation des observables est représentée sur la figure (38).

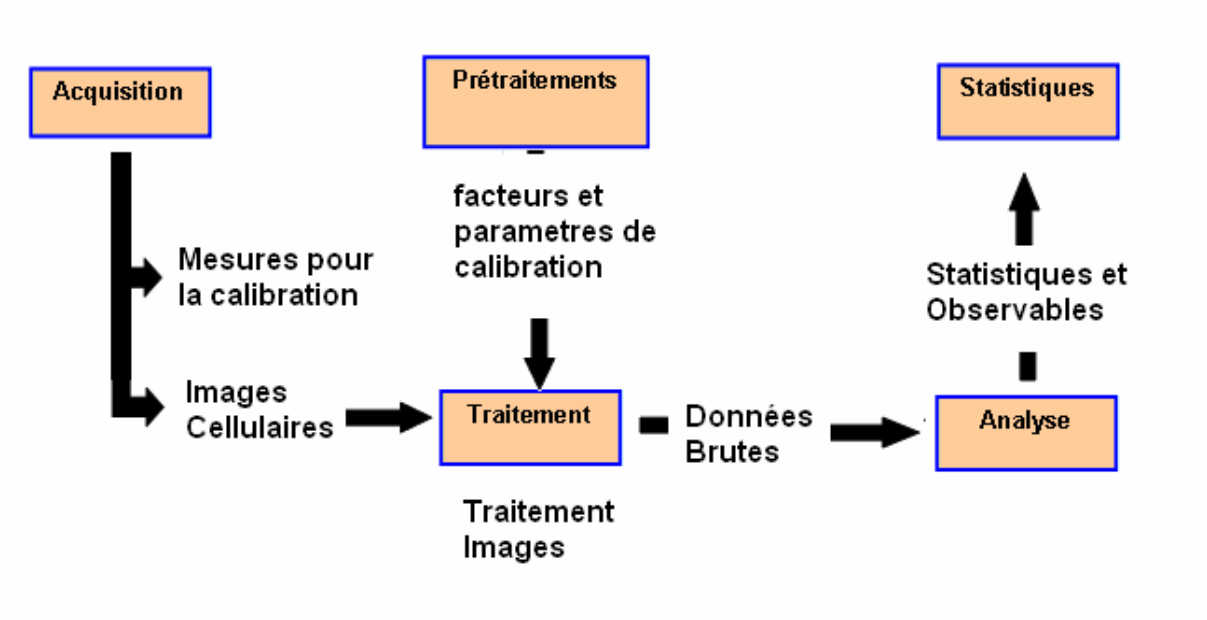


Figure (38) : Enchainement des étapes pour la reconstruction des *foci*

Annexe A
Microscope Olympus BX51

Les schémas de microscope présentés dans cette annexe sont tirés de la documentation du constructeur, et illustrent le fonctionnement du microscope Olympus BX51 équipé d'un bloc d'éclairage et des sources lumineuses pour le fonctionnement en transmission (tungstène-halogène) et en épifluorecence (arc à mercure). Pour ce dernier mode de fonctionnement, le microscope est équipé d'une tourelle de filtres de fluorescence, logée à l'avant du bloc d'éclairage.
Le principe fondamental de la microscopie de fluorescence, est de visualiser des substances fluorescentes. Dans les cellules, les molécules non fluorescentes sont marquées par des substances appelées fluorochromes. Les molécules maintenant fluorescentes sont excitées par une lumière dont la longueur d'onde excite directement le fluorophore. Ce fluorophore se désexcite en émettant alors de la lumière à une énergie plus basse. C'est le phénomène de fluorescence.

Les fluorochromes les plus classiques sont la rhodamine et ses dérivés, la fluorescéine et ses dérivés et le DAPI. Ils émettent respectivement dans le rouge, le vert et le bleu.

La fluorescence observée peut avoir plusieurs origines :

- Fluorescence naturelle d'une substance située dans la cellule, exemple : la chlorophylle fluoresce naturellement en rouge.
- Utilisation d'une substance fluorescente se fixant spécifiquement sur une structure. Par exemple, le DAPI se fixe spécifiquement sur l'ADN et fluoresce en bleu.
- Utilisation d'une substance non spécifique fluorescente naturellement, exemple : rhodamine et fluorescéine. Cette substance est fixée sur un anticorps spécifique d'un antigène. La spécificité est due à l'anticorps. La fluorescence observée permet de localiser l'antigène.

Un microscope à fluorescence est un microscope photonique équipé de deux lampes, une lampe ordinaire pour une observation classique par transmission et une lampe à arc pour la fluorescence. Un microscope équipé en épifluorescenceépifluorecence est pourvu de plusieurs jeux de filtres d'excitation permettant de choisir la longueur d'onde incidente et des filtres d'émission (ou d'arrêt) permettant de sélectionner les radiations émises par l'objet excité.

L'utilisateur peut sélectionner une combinaison de la longueur d'onde de la lumière d'excitation et du spectre de la lumière d'émission qui est renvoyée vers le capteur (CCD) :

- Excitation dans l'ultraviolet (350 nm) - Emission dans le bleue (450 nm)
- Excitation dans le bleue (450 nm) - Emission dans le vert (550 nm)
- Excitation dans le vert (550 nm) - Emission dans le jaune (580 nm)
- Excitation dans l'orange (600 nm) - Emission dans le rouge (620 nm)
- Excitation dans le rouge (650 nm) - Emission dans l'infra rouge (690 nm)

Les deux coupes suivantes montrent les chemins de la lumière, depuis la source lumineuse jusqu'au capteur final de la lumière émise par l'échantillon, dans deux cas :

- cas A : excitation dans le bleu (450 nm) et émission dans le vert (550 nm) ;
- cas B : excitation dans le vert (550 nm) et émission dans le jaune (580 nm)

Nous présentons ci après trois exemples d'utilisation de la fluorescence : choix du fluorochrome, caractéristiques optiques du fluorochrome, spectres des filtres et résultats d'une expérience.

- utilisation du TRITC (Tetra methyl Rhodamine Iso Thio Cyanate, dérivé de la Rhodamine) pour la fluorescence rouge
- utilisation du FITC (Fluoresceine Iso Thio Cyanate, dérivé de la fluorescéine) pour la fluorescence verte
- utilisation du DAPI (Di Aminido Phenyl lndo, réactif spécifique de l'ADN) pour la fluorescence bleue

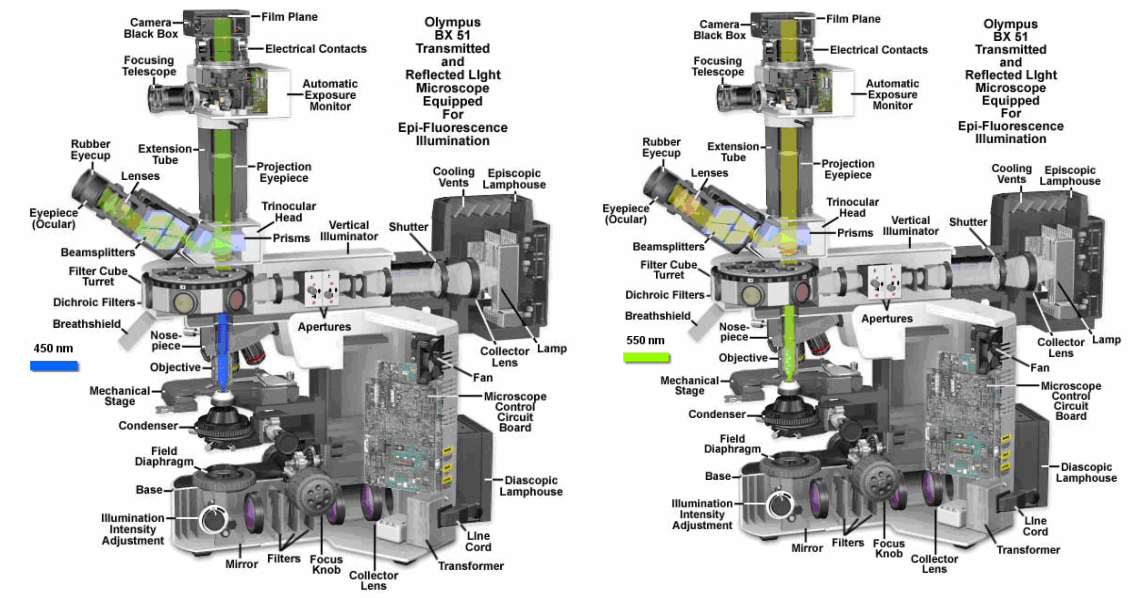
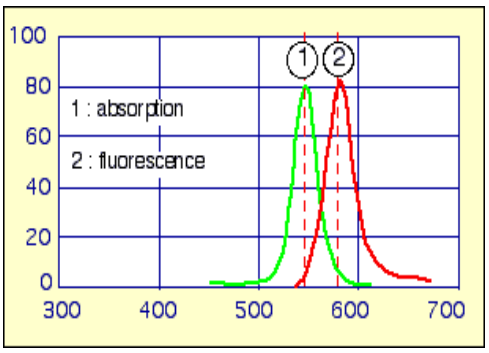


Figure (A-1) : Coupe du microscope Olympus BX51 en mode de fonctionnement épifluorecence. Les chemins de la lumière est présentée depuis la source lumineuse jusqu'au capteur final de la lumière émise par l'échantillon. A gauche excitation dans le bleu (450 nm) et émission dans le vert (550 nm) ; à droite excitation dans le vert (550 nm) et émission dans le jaune (580 nm)

Utilisation d'un fluorochrome à fluorescence rouge

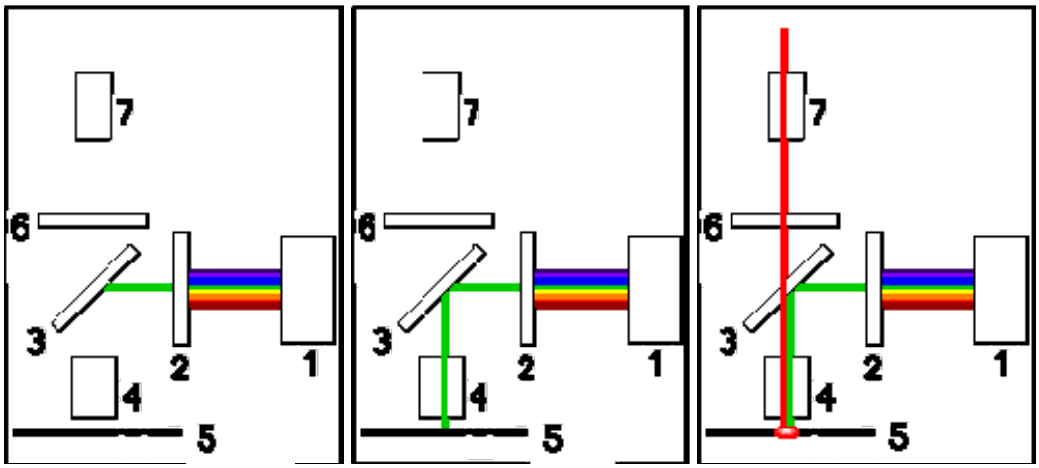
Les colorants fluorescents en rouge les plus connus sont des dérivés de la rhodamine comme le TRITC (Tetra methyl Rhodamine Iso Thio Cyanate). Comme le montre la figure suivante, la rhodamine absorbe les radiations vertes (max 541nm) et restitue une fluorescence rouge (max 572nm).



Le microscope doit être équipé d'un jeu de filtres correspondant aux caractéristiques du fluorochrome :

- 1-un filtre d'excitation permettant la sélection des radiations absorbées de la radiation absorbée par le fluorochrome (ici autour de 541nm),
- 2-un miroir dichroïque réfléchissant les radiations absorbables vers l'échantillon et ne laissant passer par transmission que les radiations rouges et au dessus (ici >580nm)
- 3-un filtre d'émission ne laissant passer par transmission que les radiations rouges et au dessus (>560nm).

Le trajet de la radiation lumineuse est le suivant.



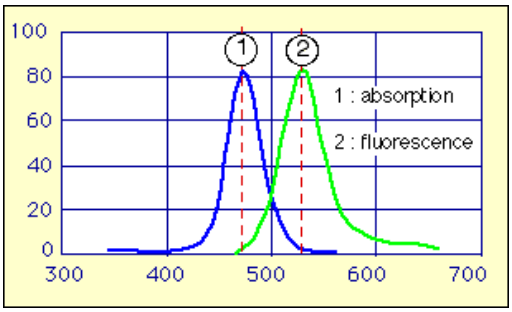
Le filtre d'excitation sélectionne les radiations spécifiques du fluorochrome...

qui sont réfléchies par le miroir et éclairent l'échantillon...

celui-ci émet les radiations de fluorescence qui seules atteignent l'oculaire.

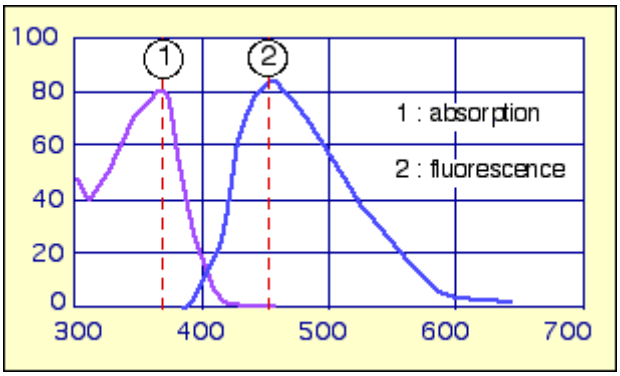
Utilisation d'un fluorochrome à fluorescence verte

Les colorants fluorescents en vert les plus connus sont des dérivés de la fluorescéine comme le FITC (Fluorescéine Iso Thio Cyanate). Comme on le voit, la fluorescéine absorbe les radiations /bleues (max 490nm) et restitue une fluorescence verte (max 520nm).



Utilisation d'un fluorochrome à fluorescence bleue

Les colorants fluorescents en bleu sont nombreux. L'exemple choisi est celui du DAPI (Di Aminido Phenyl lndol) colorant utilisé en cytochimie et spécifique de l'ADN. Le DAPI (Di Aminido Phenyl lndol) se fixe spécifiquement sur l'ADN. Eclairé en lumière violette (max 372nm), il émet une fluorescence bleue (max 456nm). Comme on le voit, le DAPI absorbe les radiations violettes (max 372nm) et restitue une fluorescence bleue (max 456nm).



Annexe B
Olympus DP70 Digital Camera System

Le microscope Olympus BX51 est équipé d'une caméra numérique couleur Olympus DP70 refroidie, de 12.5 millions de pixel, qui incorpore les dernières innovations technologiques. Le capteur DP70, représenté sur la figure (A-2), utilise un CCD (Charge-Coupled Device, ou dispositif à transfert de charge) avec filtration des couleurs primaires (RVB).

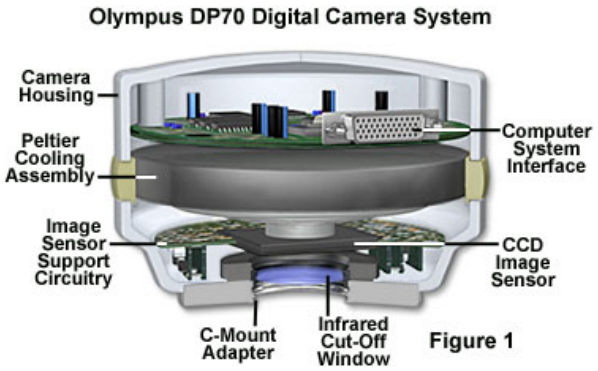


Figure (B-1) : Configuration du CCD Olympus DP70

Naturellement, ces capteurs sont sensibles à l'ensemble du spectre de la lumière visible. Grâce à un filtre de Bayer, constitué de cellules colorées des couleurs primaires, chaque photosite du capteur ne voit qu'une seule couleur : rouge, vert ou bleu. Du fait de la précision requise, les pastilles colorées du filtre sont déposées directement sur le capteur avec une technologie proche de la photolithographie des circuits intégrés. Pour chaque canal, les pixels absents sont obtenus dans le processus de restitution de l'image complète par interpolation.

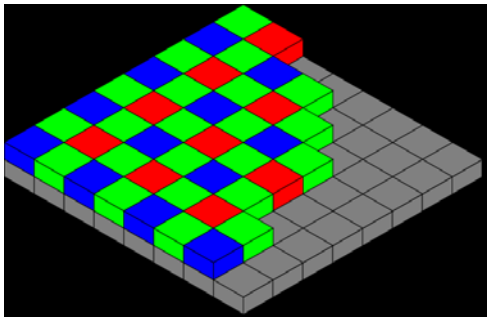


Figure (B-2) : Arrangement des filtres de couleur dans la configuration de Bayer

Ce capteur CCD incorpore 1.45 million de pixels efficaces. Des images peuvent être acquises à la résolution maximum de 4080 x 3072 pixels. La résolution du DP70 est de 12 bits, produisant des images en couleur de 36 bits RVB

Anatomie d'un CCD

Les dispositifs à couplage de charge (CCDs) sont des circuits intégrés silicium se composant d'une matrice de photosite (photodiode) qui fonctionne par conversion de l'énergie incidente sous forme de photons en une charge électronique. Quand un photon dans la gamme du visible, de l'ultra-violet ou de l'infrarouge heurte un atome de silicium, il produira habituellement un électron libre et un "trou" créé par l'absence provisoire de l'électron dans le réseau cristallin de silicium.

Les électrons produits par l'interaction des photons avec des atomes de silicium sont stockés dans un puits de potentiel maintenu à chaque photosite. Le nombre d'électrons collectés est proportionnel à la quantité de lumière reçue.

Le schéma de principe illustré sur la figure (B-3) montre les divers composants qui composent l'anatomie d'un CCD typique.

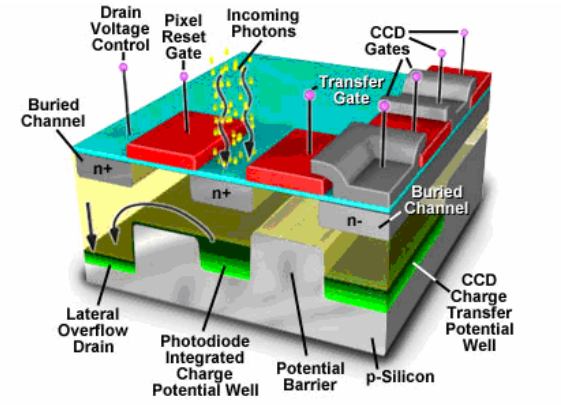


Figure (B-3) : Anatomie d'un élément de la matrice d'un CCD typique

Les différents photosites sont isolés électriquement de leurs voisines. Le dispositif architectural principal d'un CCD est donc une large matrice d'électrodes (« portes ») formée d'une couche conductrice de polysilicium dopé, séparé du substrat de silicium par une couche mince isolante de dioxyde de silicium.

Système de transfert de charge

À la fin de l'exposition, après que les électrons aient été collectés dans chaque photodiode de la matrice, les charges sont transférées de photosite en photosite jusqu'à un système d'amplification et de sortie unique par le jeu de variations de potentiel cycliques appliquées aux grilles.
Ce décalage est accompli en changeant le potentiel du négatif du puits de potentiel, tout en simultanément augmentant la polarisation de la prochaine électrode (porte) à une valeur positive.
Le substrat de silicium placé directement sous la porte devient alors un puits potentiel capable de rassembler les électrons produits localement par la lumière incidente. Les portes voisines aident à confiner des électrons dans le puits de potentiel en formant des zones des potentiels plus élevés entourant le puits.
Les charges sont transférées à travers chaque photodiode dans un processus utilisant des cycles plusieurs étapes comme le montre la figure (B-4).
La vitesse de transfert est suffisamment rapide pour être accomplie au cours de la période d'intégration de charge pour la prochaine image.
Le processus de transfert des charges est commandé par une série d'horloges qui opèrent toutes les portes simultanément dans la matrice, y compris les portes de transfert entre les registres séries et parallèles et les portes de remise à zéro des photodiodes. Pour les matrices de capteurs CCD, le transfert de charge peut s'effectuer de trois manières :

- **Système de transfert parallèle-série (capture pleine trame)** : Les cellules de la matrice sont couplées verticalement et toutes les lignes verticales sont transférées en parallèle dans un registre à décalage de lecture très rapide. Ce type de capteur nécessite un obturateur électromécanique pour maintenir la zone photosensible dans l'obscurité pendant les opérations de transfert.
- **Système de transfert interligne** : Le capteur comprend une zone image entrelacée avec des colonnes de stockage. A la fin de l'acquisition, les charges stockées dans les colonnes d'acquisition sont transférées dans les colonnes de stockage. Le contenu des colonnes de stockage est ensuite transféré à la sortie vidéo selon un mode parallèle-série. Dans cette technique une faible proportion de la surface du capteur est l'élément photosensible proprement dit (10 à 20 %). Il est néanmoins possible d'améliorer la sensibilité en utilisant des micro-lentilles (--> amélioration jusqu'à x4)
- **Système de transfert de trame** : Dans la technique de transfert de trame les colonnes d'acquisition et de stockage sont alignées verticalement. Le principe reste le même que dans le cas précédent. Cette solution permet d'avoir des dynamiques élevées, car la totalité de la zone image est photosensible. En revanche, ce système provoque des effets de traînage vertical important.

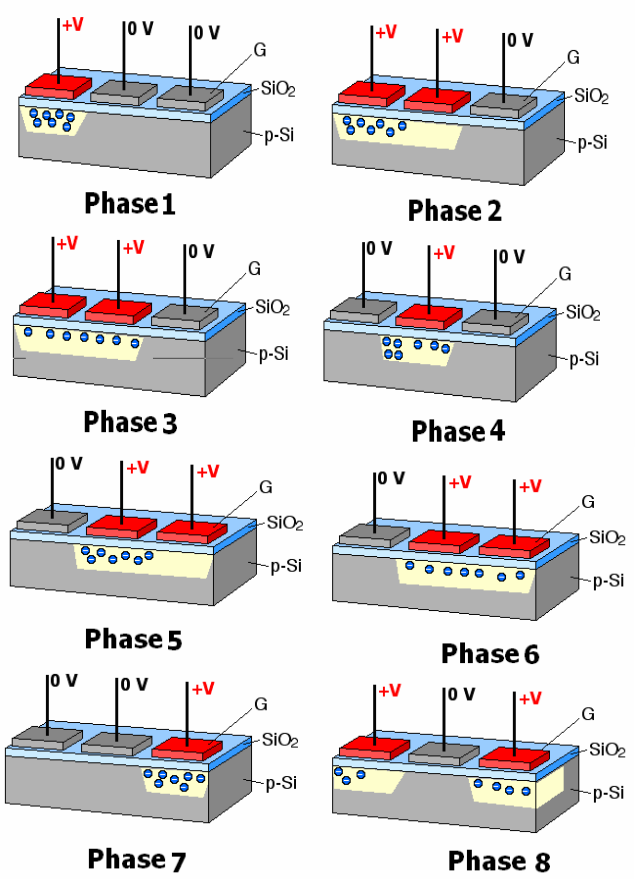


Figure (B-4) : Transfert des charges de photosite en photosite par le jeu de variations de potentiel cycliques appliquées aux grilles (bandes conductrices horizontales, isolées entre elles par une couche de SiO_2) jusqu'à un système d'amplification et de sortie unique.

Annexe C
Notions de Bruit dans les CCD

Le bruit, inhérent à tout capteur d'images numériques, a de multiples origines. Le rapport signal/bruit (*SNR*) est déterminant pour la qualité des mesures comme le montre la figure suivante.

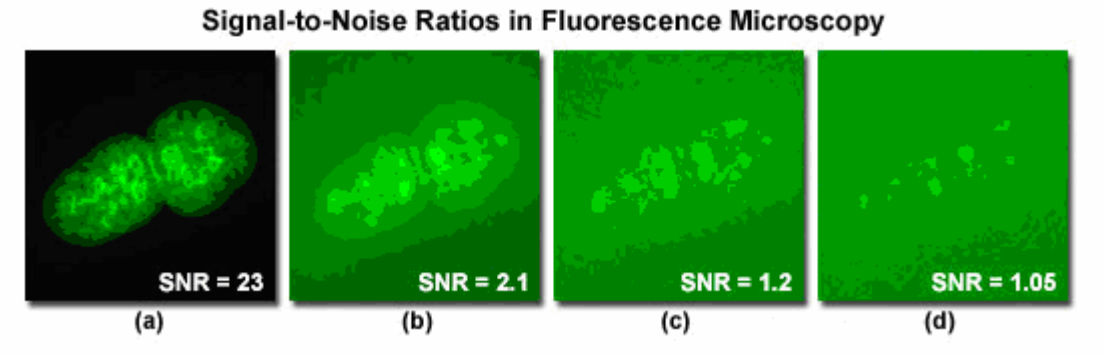


Figure (C-1) : Qualité d'une image de microscope à fluorescence en fonction du rapport signal/bruit (*SNR* **).**

Le signal enregistré par la caméra CCD est parasité par différents bruits. Les bruits inhérents au CCD sont fortement dépendant de sa structure interne et implique la compréhension du fonctionnement de celui-ci. Dans le cas du dispositif utilisé pour ces acquisitions, le CCD utilisé est l'Olympus DP70 dont une brève description est donnée en annexe B. Les principaux bruits peuvent être subdivisés en 3 catégories :

- Les bruits intrinsèques, produits par la chaîne de détection : Le CCD, les divers étages d'amplification, les circuits électroniques, le numériseur, etc. Nous considérons uniquement au bruit thermique
- Les bruits externes, que nous limiterons ici au bruit de signal.
- Les bruits de traitement numérique des images.

Ces trois bruits interviennent dans le calcul du ratio signal/bruit.

Bruits intrinsèques-Bruit thermique (σ_d)

Dans un CCD, il existe un signal thermique, encore appeler signal d'obscurité, qui a pour origine la génération thermique de porteurs qui vont s'accumuler dans les puits de potentiel au même titre que ceux produits par effet photoélectrique (signal). Il est possible de le constater en plaçant le CCD dans le noir complet pendant quelques secondes par exemple. Ce bruit dépend fortement de la température, et varie proportionnellement au temps de pose pour une température donnée.

Bruits intrinsèques-Bruit de lecture (σ_g)

Il existe également un bruit de lecture qui vient du passage des charges des pixels vers la sortie du CCD. En effet une fois l'intégration achevée, les charges accumulées sont transférées de proche en proche dans les registres du CCD. Malheureusement, à chaque transfert, une petite fraction des charges d'un pixel est perdue et récupérée par le pixel suivant. En moyenne la fraction perdue dépend du nombre de charges N_S contenues dans le pixel et de l'inefficacité de transfert ε .

Le taux de charges laissées en arrière varie autour d'une valeur moyenne d'un transfert à l'autre. Le bruit associé à ce phénomène, en électrons, est :

$$\sigma_g = \sqrt{2\varepsilon n N_S}$$

Ou n est le nombre de transferts nécessaires pour amener les charges d'un pixel donné dans l'étage de sortie. Le bruit de lecture est dû d'autre part à la précision de l'amplification analogique. Ces deux quantités diminuent quand la vitesse de lecture du CCD augmente. Les CCD présentent typiquement un bruit de lecture compris entre 10 et 100 électrons par pixel.

Le bruit de lecture intervient une fois lors de chaque mesure de façon indépendante ; pour cette raison, la somme de plusieurs images n'est pas équivalente à une seule pose de la durée totale des poses élémentaires.

Bruits intrinsèques-Bruit de reset(σ_r)

Le principe de conversion des charges en tension dans l'étage de sortie d'un CCD est très spécifique.

- Dans un premier temps, une petite capacité est chargée à un niveau de référence, puis dans un second temps le paquet de charges correspondant à un pixel décharge cette capacité d'une quantité proportionnelle au nombre d'électrons contenu dans le paquet.
- Vu de l'extérieur, le signal se présente donc sous la forme d'un palier dit de *référence* correspondant au niveau de *précharge*, suivit d'un palier dit *signal* de valeur négative par rapport au palier de référence.
- Le circuit électronique intégré dans le CCD qui réalise la *précharge* n'assure malheureusement pas une amplitude du niveau de référence rigoureusement égale d'un pixel à l'autre.
- L'origine du problème est à rechercher dans le circuit résistif qui produit le courant de charge durant un bref instant, ce courant étant plus ou moins bruité en fonction de la température de fonctionnement (bruit dit Johnson), et dans les couplages capacitifs.
- Après la *précharge*, le bruit est gelé, de même amplitude et de même signe sur le palier de référence et le palier signal.

La solution optimale pour mesurer le signal d'un pixel consiste donc à déterminer la différence entre ces deux paliers. Le fait de calculer (analogiquement ou numériquement) la différence entre le palier de référence et le palier vidéo élimine complètement le bruit de reset compte tenu de la forte corrélation de ce bruit sur les deux paliers. Cette tache incombe le plus souvent à un circuit électronique spécial (circuit *CDS* pour *Correlated Double Sampling*) ou éventuellement se limite à une opération arithmétique réalisée dans l'ordinateur.

Bruits intrinsèques-Bruit effectif de l'amplificateur (σ_a)

Les circuits électroniques servant à conditionner, amplifier et numériser le signal vidéo apportent leur lot de bruit. Pour l'essentiel, le bruit de la chaîne électronique de traitement vidéo provient du premier étage d'amplification et du convertisseur analogique/numérique.

Dans une électronique bien conçue, le signal en sortie du CCD est rapidement amplifié. Ceci permet aux circuits qui suivent de travailler avec de forts signaux, ce qui prévient dans une large mesure un apport supplémentaire de bruit dans le signal.

Le bruit ramené en entrée d'amplificateur, c'est en dire en sortie de CCD, s'obtient en divisant par le gain de l'amplificateur. Ce bruit est généralement faible par rapport à la contribution de nombreux autres facteurs de bruits (il est facile de trouver des amplificateurs ayant un bruit de quelques nV/Hz seulement).

Bruits intrinsèques-Bruit de quantification (σ_q)

Le bruit σ_q associé au convertisseur analogique/numérique s'appelle le bruit de quantification. Il prend vie dès que l'on discrétise un signal continûment variable. Il traduit l'approximation réalisée lors de l'opération de numérisation, une troncature en quelque sorte.

Le bruit de quantification est l'erreur moyenne commise en échantillonnant le signal analogique sur un nombre fini de pas d'ADC. L'écart-type de cette erreur vaut $1/\sqrt{12}$ en pas d'ADC.

On a donc intérêt à coder le signal analogique sur un nombre élevé de pas-codeurs, c'est à dire à coder le signal sur un grand nombre de bits (le nombre de pas-codeurs est 2^N , où N est le nombre de bits de l'ADC). Si g est le gain de la caméra en nombre d'électrons par pas codeur, on montre que :

$$\sigma_q = \frac{g}{\sqrt{12}}$$

Le gain g est égal à :

$$g = \frac{E}{G \cdot S \cdot 2^N}$$

Avec, *E* le signal de pleine échelle à l'entrée du convertisseur en volt, *G* le gain d'amplificateur de la chaîne électronique du signal vidéo, *S* la sensibilité de l'étage de sortie du CCD en volt/e- et *N* le nombre de bits utilisés pour la numérisation.
Le bruit de quantification sera d'autant moins important que le gain *g* sera de faible valeur. Normalement, le concepteur de la caméra CCD optimise le nombre de bits de manière à ce que le bruit soit quantifié sur quelques pas de quantification, entre 2 et 5 pour fixer les idées. De la sorte, on est sûr que la numérisation ne va masquer absolument aucune information utile dans le signal.

Bruit intrinsèque total du CCD
Le bruit total est la moyenne quadratique des différents bruits entachant le signal, car ceux-ci sont indépendants. Il s'écrit :

$$\sigma_t^2 = \sigma_d^2 + \sigma_g^2 + \sigma_r^2 + \sigma_a^2 + \sigma_q^2$$

Si nous faisons le récapitulatif des bruits intrinsèques de la caméra CCD, il apparaît que les sources principales de bruits sont le bruit thermique (σ_d), le bruit de lecture (σ_g) et éventuellement du bruit de transfert pour peu que les pixels reçoivent du signal. Dans tous les cas, le bruit de numérisation est négligeable devant le bruit de lecture.

Bruits externes-bruit de signal (σ_S)
On peut limiter les bruits externes au bruit de signal. En effet on pourrait ajouter dans cette catégorie les bruits électromagnétiques, dont les sources peuvent être un émetteur radio voisin du site d'observation, la propre alimentation de la caméra, l'ordinateur lui-même, etc.
Tous ces parasites sont captés par les liaisons électriques qui constituent l'électronique de la caméra. On laisse de coté les perturbations électromagnétiques car leurs effets sont terriblement dépendants de l'environnement de la caméra et de la manière dont celle-ci est conçue. On suppose ici que cette conception est suffisamment seine pour ne pas savoir ce type de problème (boîtier faisant office de bonne cage de Faraday, plan de masse du schéma électronique correctement dessiné, alimentations correctement filtrées, etc.).
Du fait de la nature corpusculaire de la lumière, si dans un laps de temps donné nous comptons les photons en provenance d'une source réputée stable en moyenne, nous n'obtiendrons pas le même résultat d'une expérience à l'autre.
La variation du résultat par rapport à la moyenne, est un bruit de Poisson. Si *Np* est le nombre de photons enregistrés, le bruit de photons (ou « *shot noise* » en anglais) est :

$$\sigma_S = \sqrt{Np}$$

Le bruit dépend de l'intensité de la source observée et du temps de pose. Le bruit de lecture est dominant aux courts temps de pose, et finit par devenir négligeable devant le bruit thermique quand la pose s'allonge.

Annexe D
Listing du plugin de prétraitement
« PLU_RGB_Avg_v3_java »

Ce Plugin ImageJ a été écrit pour améliorer la qualité des images de microscopie par épifluorecence a partir d'images de calibration, acquises en même temps que la prise des images des cellules. Ces images correspondent à des acquisitions spécifiques, appelées : « *offset* », « *dark* », « *flat-field* »:

« *offset* **» :** On fait une pose la plus brève possible, dans l'obscurité totale (obturateur fermé).

« *dark* **» :** On va acquérir une image avec un temps de pose du même ordre de grandeur que l'image, en bloquant toute entrée de lumière qui pourrait arriver sur le CCD (obturateur fermé) et en prenant le même temps d'exposition que celle de l'image. L'image **«***dark***»** comprend uniquement le signal de bruit de fond électronique de l'appareil. Donc lorsque le temps de pose augmente, les charges thermiques deviennent très importantes.

« *flat-field* **»** ou **«** *PLU* **»** : Une dernière catégorie de bruits provient de la variation de sensibilité des pixels du CCD et des défauts du système optique. Par exemple les poussières sur les surfaces optiques peuvent produire des ombres. De même les variations de luminosité conduisent à un vignettage (non-uniformité de l'intensité de la lumière à l'intérieur du champ du microscope). L'image « *flat-field* » est utilisée pour isoler et corriger des imperfections de la chaîne optique. On fait une pose longue sur une image de fluorescence supposée uniforme en luminosité, avec la même mise au point que les images. Cette image de référence doit être obtenue à partir d'une lame contenant un film de colorant fluorescent, réputé uniforme, à défaut un objet uniformément fluorescent. La même configuration de filtre pour l'acquisition des images doit être utilisée

.
Utilisation de ce plugin:

- Ouvrir une ou plusieurs images **«** *flat-field* **»** de format TIFF codées en RGB couleur.
- Donner un nom pour créer le fichier log récapitulatif.
- Projection des **«** *flat-field* **»** pour obtenir une image moyenne et création des masques dans les trois composantes de couleur.
- Choisir une image à corriger (condition d'acquisition conforme).
- Les composantes RGB sont corrigées indépendamment par les trois masques RGB
- La stack des composantes RGB de l'image est fusionnée en une image RGB finale.
- Contrôle de l'aspect de l'image corrigée.

Pour des images en résolution 4080x3072, augmenter la mémoire (Edit/Options/Memory=1500)

```
/*
	PLU_RGB_Avg_v3_.java

	Last update : 10/06/2010

	Plugin ImageJ ecrit pour ameliorer la qualite des images de
	microscopie par immunofluorescence a partir d'images de calibration.

	Usage:
	 - Ouvrir une ou plusieurs images TIFF de PLU de type RGB couleur.
	 - Donner un nom pour creer le fichier log recapitulatif.
	 - Projection des PLU
	 - Choisir une image a corriger (condition d'acquisition conforme).
	 - Les composantes sont corrigees independamment.
	 - La stack de l'image est fusionnee en une image RGB finale.
	 - Controle de l'aspect de l'image corrigee.

	Pour des images en resolution 4080x3072, augmenter la memoire (Edit/Options/Memory=1500)
*/

import ij.*;
import ij.process.*;
import ij.gui.*;
import java.awt.*;
import ij.plugin.filter.*;
import ij.plugin.PlugIn;
import ij.io.*;
import javax.swing.*;
import java.io.PrintWriter;  // PrintWriter
import ij.io.FileSaver;
import java.io.*;
import ij.plugin.RGBStackMerge;  // RGB Splitter
import ij.measure.*;
import ij.process.ImageStatistics;

// FileChooser
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;
import ij.io.Opener;

// Utilities
import java.util.ArrayList;
import java.util.List;
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;

// Array of constantes (operator)
import ij.plugin.ZProjector;

// Use Logger in any java application
import java.util.logging.*;

// Use DecimalFormat
import java.text.DecimalFormat;
```

```
// Convert Stack to RGB
import ij.process.StackConverter;

// classe des modèles de couleurs
import java.awt.image.ColorModel;
//import java.awt.image.ComponentColorModel;
import java.awt.image.IndexColorModel;
import java.awt.image.DirectColorModel;

public class PLU_RGB_Avg_v3_ implements PlugIn {

    //Static variables
    private     int[] wList;
    static      File dir = null;
    static      PrintWriter pw = null;

    private static int width, height;
    private int stackPLUSize;

    //Runtime variables
    ImageStack[]    stacks = new ImageStack[3]; // R,G, B stacks
    //String        directory = null;
    String          wd = null; // working directory
    String          strDirectory =
"C:\\Imaging\\ImageJ\\plugins\\mesPluginsRosiris\\fibro_h2ax_100x_HR_9 (85x64)";

    String          outputDir = null; // saving files
    int             debug; // debuging flag
    // declare and initialize array of channels
    String RGB[] = {"red", "green", "blue"};
    // Déclaration d'une énumération
    enum enumRGB {R, G, B};

    // Get all the images from the selected channel
    ImagePlus[] rgbPLUmask = new ImagePlus[3];
    //ImagePlus[] rgbPLUmask = null;

    private int startSlice = 1; // Projection starts from this slice
    private int stopSlice = 1;  // Projection ends at this slice
    private int method = ZProjector.AVG_METHOD; // Average Stack

    private static String[] operators = {"Add","Subtract","Multiply","Divide"};
    private static String[] lcOperators = {"add","sub","mul","div"};
    private static int operator = 3; // default operator: 'div'
    private static boolean doScaling = false;

    private String logname; // txt log file name

    public static Logger logger; // create a Logger object
    public static FileHandler fh; // the human readable handler

    JTextField status = new JTextField("!"); // new text field

    static long start; // Chrono

    /*
        This method runs the plugin
```

```
*/
public void run(String arg) {

    // Initialiaze plugin (open log)
    if (!init_plugin()) {
        //IJ.showMessage("Dialog LOG aborted", "STOP_PLUGIN");
        stop_plugin();
        return;
    }

    // Add all open images to a stack
    ImagePlus PLUImgStack = createStackPLU();
    if (PLUImgStack == null) { stop_plugin(); return; }

    // Perform the projection on PLU image constructs from a stack
    ImagePlus PLUImg = computePLUProjection(PLUImgStack);
    PLUImgStack.close();  // les canaux ont ete projete.

    // int width/height for check all images size
    initParameters(PLUImg);

    // Open dialog for PLU master (a 8-Bit Image)
    if (!showDialog(PLUImg)) { stop_plugin(); return; }

    PLUImg.show();  // display average flat-field after dialog

    // Creates red, green abd blue mask
    // Tous les pixels de chaque canal de la PLU moyenne projetee
    // vont etre divisee par la valeur moyenne du canal
    createMask(PLUImg);  // les 3 canaux de PLUImg sont divises par la moyenne canal */

    // Open the specified file as a tiff image and returns an ImagePlus object if successful.
    ImagePlus scienceImg = null;

    if ((scienceImg=openFile()) == null) { stop_plugin(); return; }

    // Pour forcer une image cible sans ouvrir de file chooser
    /*
        Opener opener = new Opener();
        scienceImg = opener.openImage(strDirectory, "fibro_9.tif");
        if (scienceImg == null ) {
            IJ.showMessage("Chargement de l'image", "Impossible");
            return;
        }
    */

    scienceImg.show();
    // ImagePlus.getStatistics() method class to get the histogram.
    int[] histogram = scienceImg.getStatistics().histogram;

    // Apply correction with masks (PLUredmask, PLUgreenmask, PLUbluemask):
    //     For each channel, multiply science image by PLU/mean
    // Creates a ImageStack("red"+"green"+"blue")
    // Converts this Stack to RGB
    applyCorrection(scienceImg);

    stop_plugin();
```

```
}

/*
     Creates and uses a log file for saving calculs
*/
boolean init_plugin() {

     start = System.currentTimeMillis(); // Start chrono

     // Get the current class
     String classNameMethodName = this.getClass().getName(); // Full class name

     wd = System.getProperty("user.dir");
     File file = new File(classNameMethodName+".txt");
     String logname = null;
     JFileChooser fc = new JFileChooser();

     fc.setSelectedFile(file);

     // We want 'txt' extension filter
     FileFilter filter =
         new FileNameExtensionFilter("Text file (.log, .txt, .dat)",
         new String[] {"log", "txt", "dat"});
     fc.addChoosableFileFilter(filter);
     fc.setDialogTitle("Select a log filename");

     if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
         file = fc.getSelectedFile();
         logname = file.getAbsolutePath();
     }
     else {
         System.out.println("File chooser cancel button clicked");
         return false;
     }

     // Create log
     try {
         FileOutputStream fos = new FileOutputStream(logname/*strDirectory+logname*/);
         BufferedOutputStream bos = new BufferedOutputStream(fos);
         pw = new PrintWriter(bos);

         // Include the current class nameit in a log.
         pw.println("Summary for plugin "+classNameMethodName+".java");
         pw.println(    "");
         DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
         Date date = new java.util.Date();
         String datetime = dateFormat.format(date);
         pw.println("Date/heure: " +  datetime);
         pw.println("");

         // Display some useful information about this file.
         pw.println("Some useful information about log file "+file.getName());
         pw.println("getAbsolutePath : " + file.getAbsolutePath());
         pw.println("getName         : " + file.getName());
         pw.println("getParent       : " + file.getParent());
         pw.println("getPath         : " + file.getPath());
         pw.println("isDirectory     : " + file.isDirectory());
```

```
            pw.println("isFile          : " + file.isFile());
            pw.println("isHidden        : " + file.isHidden());
            pw.println("lastModified    : " + new Date(file.lastModified()).toString());
            pw.println("length          : " + file.length());
            pw.println("");
        }
        catch (IOException e) {
            IJ.log("" + e);
            System.exit(0);
        }

        //init_logging();

        // We would like to have a default directory name
        wd = file.getParent();

        return true;
    }

    void stop_plugin() {
        if (pw==null) return; // case pw not initialized
        pw.println("elapsed time = " + IJ.d2s((System.currentTimeMillis()-start)/1000.0, 2)+
" seconds");
        pw.flush();
        pw.close();
        //stop_logging();
    }

    public void init_logging() {
        openLogger(strDirectory); // Open log
    }

    public void stop_logging() {
        // use the 'least important' type of message
        logger.info("Logging done!");
        // Clear the existing handler
        fh.flush();
        fh.close();
        logger.removeHandler(fh);
    }


    /*
        Opens all the images and creates PLU stack of 8-bit images
    */
    public ImagePlus createStackPLU() {

        // for "open file" dialog
        JTextField filename = new JTextField();
        JTextField dir = new JTextField();

        //JFileChooser fileChooser = new JFileChooser("C:\\Imaging\\ImageJ\\plugins\\mesPluginsRosiris\\fibro_h2ax_100x_HR_9 (85x64)");
        JFileChooser fileChooser = new JFileChooser(wd);

        // get a list of all files in a directory

        File[] files = null;
```

```
/*
// Lignes a activer pour forcer une liste precise de fichiers PLU
	File directory = new File(strDirectory);
	fileChooser.setCurrentDirectory(directory);
	// We want image extension filters
	FileFilter filter =
		new FileNameExtensionFilter("Image file (.tif, .tiff, .jpg, .jpeg, .gif, .png)",
		new String[] {"tif", "tiff", "jpg", "jpeg", "gif", "png"});
	fileChooser.addChoosableFileFilter(filter);
	fileChooser.setMultiSelectionEnabled(true);

	ArrayList<File> listSelectedFiles = new ArrayList<File>();
	listSelectedFiles.add(new File(strDirectory, "fibro_PLU_1.tif"));
	listSelectedFiles.add(new File(strDirectory, "fibro_PLU_2.tif"));

	// Open selected files
	fileChooser.setSelectedFiles((File[])listSelectedFiles.toArray(new File[listSelectedFiles.size()]));
	files = fileChooser.getSelectedFiles();
*/

pw.println("Current directory: " + System.getProperty("user.dir"));
//System.setProperty("user.dir", strDirectoy);  // change CD
//System.out.println("CD=" + System.getProperty("user.dir"));

// We want image extension filters
FileFilter filter =
    new FileNameExtensionFilter("Image file (.tif, .tiff, .jpg, .jpeg, .gif, .png)",
    new String[] {"tif", "tiff", "jpg", "jpeg", "gif", "png"});
fileChooser.addChoosableFileFilter(filter);
fileChooser.setMultiSelectionEnabled(true);

fileChooser.setDialogTitle("Open raw flat-field image(s)...");
int retVal = fileChooser.showOpenDialog(null);
if (retVal == JFileChooser.CANCEL_OPTION) {
    pw.println("Open file chooser command cancelled by user.");
    //logger.fine("Open command cancelled by user.");
    return null;
}
// Open selected files
files = fileChooser.getSelectedFiles();

pw.println("You selected:");
String[] fileNames = new String[files.length];
int i=0;
while (i < files.length) {
    fileNames[i] = files[i].getPath();
    pw.println("\t" + fileNames[i]);
    // check correct filename
    if (fileNames[i] == null) return null;
    i++;
}
pw.println(i+" file(s) selected.");

// check that there all images have the same width, height, and color model
// Make sure the input image is a stack.
if(fileNames==null || fileNames.length<1) {
    IJ.showMessage("Plugin error", "This plugin must be called on at least one PLU
```

```
image");
            return null;
        }

        if (fileNames==null) return null;

        ImageStack stackPLU = null; // temporary stack (arg for projImagePLU contructor)
        ImagePlus projImagePLU = null; // Image to hold z-projection
        double min = Double.MAX_VALUE;
        double max = -Double.MAX_VALUE;
        for (i=0; i<fileNames.length; i++) {
            if (fileNames[i].startsWith(".")) continue;
            Opener opener = new Opener();
            projImagePLU = opener.openImage(fileNames[i]);
            if (projImagePLU==null)
                IJ.log(fileNames[i] + ": unable to open");
            else {
                if (stackPLU==null)
                    stackPLU = projImagePLU.createEmptyStack();
                    try {
                        ImageProcessor ip = projImagePLU.getProcessor();
                        if (ip.getMin()<min) min = ip.getMin();
                        if (ip.getMax()>max) max = ip.getMax();
                        stackPLU.addSlice(fileNames[i], ip);
                    }
                    catch(OutOfMemoryError e) {
                        IJ.outOfMemory("createPluStack");
                        stackPLU.trim();
                        break;
                    }
                    IJ.showStatus((stackPLU.getSize()+1) + ": " + fileNames[i]);
                }
        }
        if (stackPLU==null) {
            IJ.showMessage("Plugin error", "Can't creates PLU stack");
            return null;
        }
        // Check dim compatibility
        ImageProcessor ip = stackPLU.getProcessor(1);
        width = ip.getWidth();
        height = ip.getHeight();
        for (int n = 2; n <= stackPLU.getSize(); n++) { // loop through all stack images
            ip = stackPLU.getProcessor(n);
            if (ip.getWidth()!=width && ip.getHeight()!=height) {
                IJ.showMessage("Dimensions incompatibles", "Hauteurs et/ou largeurs
image/PLU differentes!");
                return null;
            }
        }

        projImagePLU = new ImagePlus("PLU_Stack", stackPLU);
        // display the resulting image stack
        projImagePLU.show();

        outputDir = files[0].getParentFile()+File.separator;
        if (debug==1) {
            String fileName = outputDir+projImagePLU.getTitle()+"_(b._projection).tif";
```

```
            IJ.showMessage("outputDir: files[0].getParentFile()+File.separator", outputDir+
"\n"+filename);
            new FileSaver(projImagePLU).saveAsTiff("filename");
         }
        return projImagePLU;
    }

    /*
        Performs the projection using the current projection method.
        Calculates the values of pixels located at the same position
        in each slice of the stack

        param imp: the ImagePlus object on which to perform the projection

        Return the resulting ImagePlus object (projection of stackPLU)
 */
    ImagePlus computePLUProjection(ImagePlus PLUImgStack) {

        // Validate the arguments.
        if(PLUImgStack==null) { IJ.noImage(); return null; }
        stackPLUSize = PLUImgStack.getStackSize();
        if (stackPLUSize==0)
            throw new IllegalArgumentException("Empty stack.");

        // Create an empty stack to hold the projections.
        // It will have the same width, height, and color model as stackPLU.
        ImageStack tmp_out_stack = PLUImgStack.createEmptyStack();

        // Initialize the ZProjector object.
        ZProjector zproj = new ZProjector(PLUImgStack);

        // Slices are numbered 1,2,...,n where n is the number of slices in the stack.
        startSlice = 1;
        stopSlice = stackPLUSize;
        zproj.setStartSlice(startSlice);
        zproj.setStopSlice(stopSlice);
        zproj.setMethod(method);
        // Check for RGB image (RGB stacks are supported)
        if(PLUImgStack.getBitDepth()==24 )
             zproj.doRGBProjection();
        else
             zproj.doProjection();
        // Image to hold z-projection
        ImagePlus projImage = null;
        // ImageJ retourne une l'image de la + recente operation de projection
        projImage =  zproj.getProjection();

        // PROJECTION DONE

        ImageProcessor improc = projImage.getProcessor();
        tmp_out_stack.addSlice("Projection_of_"+PLUImgStack.getTitle(), improc);

        // Constructs an ImagePlus from a stack.
        ImagePlus out_image = new ImagePlus("Projection_of_"+PLUImgStack.getTitle(),
tmp_out_stack);
        out_image.show();
```

```
        // projection result Backup
        // PLU stack saved into ImagePlus object
        String filename = outputDir+out_image.getTitle()+"_("+ZProjector.METHODS[method]+
"_projection).tif";
        new FileSaver(out_image).saveAsTiff(filename);

        return out_image;
    }

    /*
        Multiply each image channel  by a mean of channel values
    */
    public void createMask(ImagePlus PLUImg /* Stack des PLU moyennees */) {

        // Checking
        if (PLUImg==null) {
            IJ.log("Unfortunately, image was null.");
            IJ.noImage(); return;
        }
        if ( (PLUImg.getType()!=ImagePlus.COLOR_RGB) || (PLUImg.getBitDepth()!=24) ) {
            IJ.showMessage("the plugin requires an RGB 24-bit image");
            return;
        }
        // Obtain an RGB image stack
        ImageStack is = PLUImg.getStack();
        pw.println("Entry createMask: "+is.getWidth()+"x"+is.getHeight()+"x"+is.getSize());

        // Splits an RGB image (or stack) into 3 8-bit grayscale images or stacks
        IJ.showStatus("Splits an RGB image into three 8-bit grayscale images");
        ImageStack rgbStack = splitStack(PLUImg);

        // rgbStack with 3 channels required
        if (rgbStack.getSize()!=3)
            { IJ.error("Stack Required"); return; }

        String title = PLUImg.getTitle();
        pw.println("createMask PLUImg title = " + title);

        int slices = rgbStack.getSize();  // Returns the number of channels.
        pw.println("createMask Num. of slices "+ slices);

        DecimalFormat dec = new DecimalFormat("##0.0000");

        // Recuperer les 3 slices correspondants aux canaux RGB
        // Iterate all slices (slices are 1<=i<=3)
        for(int rgbcycle=0; rgbcycle<slices; rgbcycle++) {

            //Returns an ImageProcessor for the specified slice, were 1<=n<=nslices.
            ImageProcessor rgbip = rgbStack.getProcessor(rgbcycle+1);

            // Recuperer la mesure de la moyenne d'une image RGB
            IJ.showStatus("Getting statistics");
            //ImageStatistics stats = imp_temp.getStatistics(Measurements.MEAN, null);
            ImageStatistics stats = ImageStatistics.getStatistics(rgbip, Measurements.MEAN,
null);

            // Recuperation des valeurs de l'histo
```

```
            //stats = rgbPLUmask[rgbcycle].getStatistics();
            double mean, stdDev, hMin, hMax, binWidth;
            mean = stats.mean;
            stdDev = stats.stdDev;
            hMax = stats.max;
            hMin = stats.min;
            int nBins = stats.histogram.length;
            // Means stats by Channel
            // [0] -> red Pixels, [1] -> green Pixels, [2] -> blue Pixels
            pw.println("\tMean of channel "+RGB[rgbcycle]+ "\t= "+ dec.format(mean)+
                       "\tnBins="+nBins+"\tstdDev="+dec.format(stdDev));
            /*****************************************************************************/
            /* Point operation: All pixels divided by a mean value (scalar multiplication) */
            /*****************************************************************************/
            rgbip = rgbip.convertToFloat(); // 32-bit conversion
            rgbip.multiply(1.0D/mean);
            rgbPLUmask[rgbcycle] = new ImagePlus("Projection_of_Mask_PLU_("+RGB[rgbcycle]+")"
, rgbip);
            if (debug==1) {
                //rgbPLUmask[rgbcycle].show();
                String fileName = outputDir+rgbPLUmask[rgbcycle].getTitle()+".tif";
                new FileSaver(rgbPLUmask[rgbcycle]).saveAsTiff(fileName);
            }
        } // end rgbcycle
    }

    /*
        Splits the specified RGB image or stack into a stack of three 8-bit images or stacks
        representing the red, green and blue components of the original image.
    */
    public ImageStack splitStack(ImagePlus imp) {

        ImageProcessor ip = imp.getProcessor();
        if (!(ip instanceof ColorProcessor)) {
            IJ.showMessage("Only RGB images are currently supported.");
            return null;
        }

        // Split the RGB image or stack Channels into three 8-bit images
        RGBStackSplitter splitter = new RGBStackSplitter();
        // Method RGBStackSplitter.split(ImageStack rgb, boolean keepSource)
        splitter.split(imp.getStack(), true);

        // Fetch the three stacks created by the split method
        ImagePlus red = new ImagePlus("Red", splitter.red); // stack red channel
        ImagePlus green = new ImagePlus("Green", splitter.green); // green
        ImagePlus blue = new ImagePlus("Blue", splitter.blue); // blue

        // image stacks consist of images (slices)
        // access the stack by retrieving it from the ImagePlus using ImageStack getStack()
        ImageStack is = imp.getStack();
        ImageStack rgbStack = new ImageStack(is.getWidth(), is.getHeight());

        // color images can be decomposed into the separate 8-bit R,G and B grayscale images
        // using the Image > Color > RGB split function
        byte[] r,g,b;
        ColorProcessor cp; // ColorProcessor returns the separated RGB components
```

```
        r = new byte[width*height];
        g = new byte[width*height];
        b = new byte[width*height];
        cp = (ColorProcessor)is.getProcessor(1);
        cp.getRGB(r,g,b);
        //rgbStack.deleteSlice(1); // delete 'imp' stack
        rgbStack.addSlice(null,r);
        rgbStack.addSlice(null,g);
        rgbStack.addSlice(null,b);

        pw.println("Separating channels and creating RGB mask for " + imp.getTitle());
        pw.println("rgb stack size= "+rgbStack.getSize());

        // Return an 'ImageStack' rgbStack with separating channels
        return rgbStack;
    }

    /*
        Apply correction on scienceImg with PLUredmask, PLUgreenmask and PLUbluemask)
        For each channel, multiply science image by PLU/mean
        Creates a ImageStack("red"+"green"+"blue")
        Converts this Stack to RGB

        Image/Color/Stack to RGB
            Cree une stack avec les canaux R, G, B corriges
            Converts a 2 or 3-slice stack to an RGB image assuming that the slices are in R, G, B order.

            Equivalence Menu:
                "Image/Color/Channels Tool.../More>>/Convert to RGB"
            Equivalence Commande:
                IJ.doCommand("Stack to RGB"); // The stack must be 8-bit or 16-bit grayscale.
    */
    void applyCorrection(ImagePlus scienceImg) {

        if (rgbPLUmask[0] == null || rgbPLUmask[1]== null|| rgbPLUmask[2]== null) {
            IJ.showMessage("Chargement de l'image", "Impossible");
            return;
        }
        // Splits the specified RGB image or stack into three 8-bit grayscale images or stacks.
        RGBStackSplitter s = new RGBStackSplitter();
        s.split(scienceImg.getStack(), true);
        // Get filename without Extension
        int index = scienceImg.getTitle().lastIndexOf('.');
        String title = scienceImg.getTitle().substring(0, index);

        ImagePlus scienceRedChannel = new ImagePlus(title + " (Red)", s.red);
        ImagePlus scienceGreenChannel = new ImagePlus(title + " (Green)", s.green);
        ImagePlus scienceBlueChannel = new ImagePlus(title + " (Blue)", s.blue);

        if (debug==1) {
            String fileName = null;
            //scienceRedChannel.show();
            fileName = outputDir+"splitter "+scienceRedChannel.getTitle()+".tif";
            new FileSaver(scienceRedChannel).saveAsTiff(fileName);
            //scienceGreenChannel.show();
            fileName = outputDir+"splitter "+scienceGreenChannel.getTitle()+".tif";
            new FileSaver(scienceGreenChannel).saveAsTiff(fileName);
```

```
        //scienceBlueChannel.show();
        fileName = outputDir+"splitter "+scienceBlueChannel.getTitle()+".tif";
        new FileSaver(scienceBlueChannel).saveAsTiff(fileName);
    }

    // For each channel, multiply science image by PLU/mean
    // Creates 3 "Corrected_" RGB images
    ImagePlus correctedRedChannel = doCorrection(scienceRedChannel, rgbPLUmask[0]);
    ImagePlus correctedGreenChannel = doCorrection(scienceGreenChannel, rgbPLUmask[1]);
    ImagePlus correctedBlueChannel = doCorrection(scienceBlueChannel, rgbPLUmask[2]);

    if (debug==1) {
        String fileName = null;
        //correctedRedChannel.show();
        fileName = outputDir+correctedRedChannel.getTitle()+".tif";
        new FileSaver(correctedRedChannel).saveAsTiff(fileName);
        //correctedGreenChannel.show();
        fileName = outputDir+correctedGreenChannel.getTitle()+".tif";
        new FileSaver(correctedGreenChannel).saveAsTiff(fileName);
        //correctedBlueChannel.show();
        fileName = outputDir+correctedBlueChannel.getTitle()+".tif";
        new FileSaver(correctedBlueChannel).saveAsTiff(fileName);
    }

    /*
    Lors de la conversion d'images 16-bit (entier) ou 32-bit (réel) en image couleur,
    il y a toujours une perte d'information, puisque chaque canal d'images en couleurs
    ne contient que 8 bits d'information (valeurs entre 0 et 255) .
    Cette information est perdue et ne peut pas être récupérés en utilisant la couleur,
    le contraste, ou la luminosité.
    Malheureusement, c'est exactement la façon standard de produire des images en couleur
    à partir d'images en niveaux de gris dans ImageJ, donc si des images avec plus
    de dynamique sont utilisées, elles doivent toujours être dégradé en 8-bits avant
    d'être convertie en couleurs RVB.
    */

    // Do Stack conversions
    //--------------------
    // ImageProcessor convertion to 8-bit grayscale

    // Construction d'une stack d'images avec des valeurs réelles
    ImageStack stack = new ImageStack(width, height); // former la stack a convertir en RGB
    stack.addSlice("red", correctedRedChannel.getProcessor());
    stack.addSlice("green", correctedGreenChannel.getProcessor());
    stack.addSlice("blue", correctedBlueChannel.getProcessor());
    // Constructs an ImagePlus from a stack.
    title = "ImageStack 32-bit (avant conversion)";
    ImagePlus imp_corrected = new ImagePlus(title, stack);

    if (debug==1) {
        String fileName = outputDir+imp_corrected.getTitle()+".tif";
        new FileSaver(imp_corrected).saveAsTiffStack(fileName);
    }

    /* 1. Quand la variable doScaling == false:
        Tous les pixels en dessous de la valeur de certains niveaux de gris sont mises à 0
        et tous ceux qui figurent au dessus de la valeur maximale (pour les images 8-bit)
```

```
        sont mis a cette valeur maximale qui est 255.
        Ca revient a effectuer un seuillage!

      2. Quand la variable doScaling == true:
       Toute la plage des pixels est utilisee meme si min/max ne l'utilise pas

    /*
        Conversion en niveaux de gris 8-bit les 3 image 32-bit grayscale.
        ImageJ convertit les images 16-bit et 32-bit en 8-bit
        avec une echelle linéaire de min-max à 0-255,
         où min et max sont les deux valeurs affichées dans
           "Image>Adjust>Brightness/Contrast".
        La commande "Image>Show Info" affiche ces deux valeurs comme plage d'affichage.

        REMARQUE :
        ----------
        Cette extension ne se fait pas si "Scale When Converting" n'est pas coché dans
           "Edit>Options>Conversions".
        Avec les stacks, toutes les slices sont scalees a 8-bit en utilisant les min et
        max de la slice en cours d'affichage.
    */

    // Converts 32-bit grayscale to 8-bit grayscale.
    ImageProcessor ipr = correctedRedChannel.getProcessor().convertToByte(doScaling);
    correctedRedChannel.setProcessor(ipr);
    ImageProcessor ipg = correctedGreenChannel.getProcessor().convertToByte(doScaling);
    correctedGreenChannel.setProcessor(ipg);
    ImageProcessor ipb = correctedBlueChannel.getProcessor().convertToByte(doScaling);
    correctedBlueChannel.setProcessor(ipb);

    if (debug==1) {
        String fileName = null;
        String fileNameExtension = "_convertToByte(scale="+(doScaling?"true":"false")+
").tif";
        fileName = outputDir+correctedRedChannel.getTitle()+fileNameExtension;
        new FileSaver(correctedRedChannel).saveAsTiff(fileName);
        fileName = outputDir+correctedGreenChannel.getTitle()+fileNameExtension;
        new FileSaver(correctedGreenChannel).saveAsTiff(fileName);
        fileName = outputDir+correctedBlueChannel.getTitle()+fileNameExtension;
        new FileSaver(correctedBlueChannel).saveAsTiff(fileName);
    }
    /*
    if (debug==1) {
        //correctedRedChannel.show();
        //correctedGreenChannel.show();
        //correctedBlueChannel.show();
        IJ.showMessage("Les 3 composantes RGB de la stack 32-bit ont ete converties en byte.");
    }
    */

    // Converts a 3 slices 8-bit stack to RGB.

    ImageStack redStack = new ImageStack(width,height);
    redStack.addSlice("redConvertToByte", correctedRedChannel.getProcessor());
    ImageStack greenStack = new ImageStack(width,height);
    greenStack.addSlice("greenConvertToByte", correctedGreenChannel.getProcessor());
    ImageStack blueStack = new ImageStack(width,height);
```

```
        blueStack.addSlice("blueConvertToByte", correctedBlueChannel.getProcessor());

        // do stack conversions
        RGBStackMerge stackMerge = new RGBStackMerge();
        int d = redStack.getSize();
        boolean keep  = true;
        ImageStack stackRGB = stackMerge.mergeStacks(width, height, d, redStack, greenStack,
blueStack, keep);
        ImagePlus correctRGBVersion = new ImagePlus("RGB result of "+scienceImg.getShortTitle
(), stackRGB);
        new FileSaver(correctRGBVersion).saveAsTiff(outputDir+"RGB result of "+scienceImg.
getShortTitle()+" (scale="+(doScaling?"true":"false")+").tif");
        correctRGBVersion.show();

        IJ.run("Histogram");
    }

    /*
        Do multiplication/division/addition/sousytaction of two images
        Call on each RVB channel
        Creates a "Corrected_" image
    */
    public static ImagePlus doCorrection(ImagePlus imp1, ImagePlus imp2) {

        ImageProcessor ip1 = imp1.getProcessor(); // target processor ip1
        ImageProcessor ip2 = imp2.getProcessor(); // source processor ip2

        // 8-bit image contains pixels which values range from 0 to 255
        // If you have a pixel with a value of 200 and you multiply it by 3 it will become 255 not 600.

        // So, 32 bits conversion!
        // Take original image transform it into 32-bit (Image > Type > 32-bit)
        // Mathematic operation should not have under/overflow problems
        ip1 = ip1.convertToFloat();
        ip2 = ip2.convertToFloat();

        ip1 = ip1.duplicate(); // Returns a duplicate of this image.

        try {
            if (ip1 instanceof ColorProcessor)
                IJ.showMessage("calculate", "Error : ip must be converted to float");
            switch (operator) {
                case 0: operator = Blitter.ADD; break;
                case 1: operator = Blitter.SUBTRACT; break;
                case 2: operator = Blitter.MULTIPLY; break;
                case 3: operator = Blitter.DIVIDE; break;
            }
            // Performs correction using specified method
            ip1.copyBits(ip2, 0, 0, operator);
        }
        catch (IllegalArgumentException e) {
            IJ.error("My error " + imp1.getTitle() + ": " + e.getMessage());
            return null;
        }

        ImagePlus result = new ImagePlus("Corrected_32-bit_"+imp1.getTitle(), ip1);
```

```
        return result;
    }

    /*
        Open GenericDialog for select image files
    */
    private boolean showDialog(ImagePlus imp) {

        width = imp.getWidth();
        height = imp.getHeight();
        // Conversion Options (/src/ij/plugin/Options.java)
        double[] weights = ColorProcessor.getWeightingFactors();
        boolean weighted = !(weights[0]==1d/3d && weights[1]==1d/3d && weights[2]==1d/3d);

        GenericDialog gd = new GenericDialog("Plugin Options");
        String msg = "This plugin calibrates a RGB image from an average flat-field
image.\n\n";
        gd.addMessage(msg);
        msg = "Number of images in flat-field sequence: "+stackPLUSize;
        gd.addMessage(msg);
        // Different kinds of projections.
        msg = "Select methods for producing average flat-field image:";
        gd.addMessage(msg);
        gd.addChoice("Projection Type", ZProjector.METHODS, ZProjector.METHODS[method]);
        msg = "Select operator for calibration:";
        gd.addMessage(msg);
        gd.addChoice("Operation:", operators, operators[operator]);

        gd.addCheckbox("Scale When Converting", ImageConverter.getDoScaling());
        String prompt = "Weighted RGB Conversions";
        if (weighted)
            prompt+=" ("+IJ.d2s(weights[0])+","+IJ.d2s(weights[1])+","+IJ.d2s(weights[2])+")";
        gd.addCheckbox(prompt, weighted);
        gd.addNumericField("Save all images (Debug mode) (1) Display result only (0) ? ", 1,
0);

        msg = "Saving images ("+width+"x"+height+") in output directory:\n  "+outputDir;
        gd.addMessage(msg);

        gd.showDialog();
        if (gd.wasCanceled())
            return false;

        // Gets Projection methode
        method = gd.getNextChoiceIndex();
        // Gets Processing operator
        operator = gd.getNextChoiceIndex();
        // Gets do scaling option
        doScaling = gd.getNextBoolean();
        ImageConverter.setDoScaling(doScaling);
        // Gets conversion option
        Prefs.weightedColor = gd.getNextBoolean();
        if (!Prefs.weightedColor)
                ColorProcessor.setWeightingFactors(1d/3d, 1d/3d, 1d/3d);
        else if (Prefs.weightedColor && !weighted)
                ColorProcessor.setWeightingFactors(0.299, 0.587, 0.114);
        // Asking for debug mode
```

```
        String sPrompt = "Debug mode ? ";
        // debug = 1: Running in mode debug
        debug = (int)(gd.getNextNumber());

        pw.println("Method selected: " + ZProjector.METHODS[method]);
        pw.println("operator selected: " + operators[operator]);
        pw.println("Scaling: " + (ImageConverter.getDoScaling()?"true":"false"));
        pw.println("weighted color: " + (Prefs.weightedColor?"true":"false"));

        int size = width * height;
        int nSlices = imp.getStackSize();
        pw.println("Science Image Size ("+width+"x"+height+") = "+size);
        pw.println("num Slices = " + nSlices);
        pw.println("number of channels = " + imp.getNChannels()); // the number of channels.
        pw.println("number of slices = " + imp.getNSlices());
        pw.println("number of frames = " + imp.getNFrames());

        pw.println("");

        IJ.showStatus("Saving log...");

        return true;
    }

    /*
        The method showAbout displays an about dialog.
    */
    void showAbout() {
        IJ.showMessage("PLU_RGB_avg version 3");
    }

    /*
        Display a file open dialog, gets the directory and file name the user has selected.
        If the user has cancelled, we return. Otherwise a non-nullimage is returned.
     */
    public static ImagePlus openFile() {
        // Open the specified file as a tiff image and returns an ImagePlus object if successful.
        OpenDialog od = new OpenDialog("Select TIFF file", null);
        // check that any TIFF file was selected
        String dirname = od.getDirectory(); // Returns the selected directory
        if (null == dirname) return null; // dialog was canceled
        dirname = dirname.replace('\\', '/'); // Windows safe
        if (!dirname.endsWith("/")) dirname += "/";

        String filename = od.getFileName(); // Returns the selected file name
        if (null == filename) {
            return null;
        } else if (!filename.endsWith(".tif")) {
            IJ.showMessage("Not a TIFF file!");
            return null;
        }
        // Show  path
        IJ.showStatus("Loading TIFF File: " + dirname  + filename);
        pw.println("directory = " + dirname);
        pw.println("filename  = " + filename);

        File file = new File(dirname, filename);
```

```
        if (!file.exists()) {
            pw.println("File " + file + " does not exist.");
            return null;
        }
        Opener opener = new Opener();
        ImagePlus img = opener.openImage(file.getParent(), file.getName());

        if (img.getWidth()!=width && img.getHeight()!=height) {
            IJ.showMessage("Dimensions incompatibles", "Hauteurs et/ou largeurs PLU/Image
differentes!");
            return null;
        }

        FileInfo fi = img.getFileInfo();
        if (fi==null)
            return null;
        pw.println();
        pw.println("-------IJTiffTest results-------");
        pw.println("Details for result image "+dirname+filename);
        pw.println(fi);
        pw.println("info=" + fi.info);
        pw.println("description=" + fi.description);

        IJ.showStatus("");
        return img;
    }

    /*
        to use Logger in any java application
    */
    // Set the human readable handler
    public void openLogger(String directory) {
        // typically use one logger per class
        logger = Logger.getLogger(PLU_RGB_Avg_v3_.class.getName());
        fh = null;
        try {
            // Get the current class
            String classNameMethodName = this.getClass().getName(); // Full class name

            // This block configure the logger with handler and formatter
            //fh = new FileHandler("C:\\Imaging\\ImageJ\\plugins\\Innotep_\\"+classNameMethodName+".log", true);
            fh = new FileHandler(directory+classNameMethodName+".log", true);
            logger.addHandler(fh);
            logger.setLevel(Level.ALL);
            SimpleFormatter formatter = new SimpleFormatter();
            //fh.setFormatter(new XMLFormatter());
            fh.setFormatter(formatter);

            // the following statement is used to log any messages
            logger.log(Level.WARNING,"My log");

        } catch (SecurityException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        logger.severe("my severe message");
```

```
        logger.warning("my warning message");
        logger.info("my info message");
    }

    public void initParameters(ImagePlus imp) {
        width  = imp.getWidth();
        height = imp.getHeight();
    }
}
```

Annexe E
Listing de la Macro d'analyse
« Macro_GM_CCQ_v9 »

Cette macro simplifie la tache de comptage des *foci* par noyau et de mesure des caractéristiques des *foci* dans des cellules marquées au DAPI et au H2AX par l'analyse des images de cellules de fibroblaste irradiées à l'ESRF

Les images des masques, les ROI et des *foci* repérés sont stockés dans une pile. Une boite de dialogue demande un dossier contenant deux sous-dossiers obligatoirement nommés DAPI et H2AX pour automatiser les traitements et répéter les mesures.

- Les images Tiff, (DAPI et H2AX), à analyser doivent être triées dans le même ordre pour être appariées correctement lors de l'analyse,
- On exécute la macro par la touche [g] ou dans "Plugins>Macros>Run",
- Elle boucle sur l'ensemble des images des deux répertoires,
- La macro propose automatiquement une sélection pour chaque noyau analysé: l'opérateur doit accepter ou rejeter la sélection. ==> 'Yes' ajoute la sélection au « ROI manager », 'No' la supprime.
- Les sélections sont renommées (préfixe "Cell_"),
- En fin d'analyse, les résultats sont sauvés dans le répertoire parent:
    - Les sélections (ROI) sont sauvegardées dans les fichiers de type « zip » (Qu'elles soient multiples ou unique).
    - Les résultats des analyses des *foci* par noyau (mesures de surface, périmètre, position…) sont placés dans les fichiers de type « xls »
    - Les piles d'images mémorisent les masques ayant servis à la quantification.

```
// Macro_GM_CCQ_v7.txt

// Macro_convex_hull_distribution.txt (video)

// Last uUpdate : 21/07/2010

/**
 *Cette macro simplifie la tache de comptage des focis par noyau et de mesure
 *des caracteristiques des focis dans des cellules marques au DAPI et au H2AX.
 *Les images de masque, les ROI et des focis reperes sont stockes dans une stack.
 *Une boite de dialogue demande un dossier contenant deux sous-dossiers *obligatoirement*
 *nommes DAPI et H2AX pour automatiser les traitements et repeter les mesures.
 *
 * @auteurs Frederic Chandez &G Montarou
 */

macro "Macro_GM_CCQ_v9" {
// Realise l'analyse (surface, position, en pixels) des images de cellules de fibroblaste irradiées à l'ESRF
// Instructions:
//   - Dans un repertoire quelconque, creer 2 dossiers appeles 'DAPI' et 'H2AX' contenant les images à analyser.
//   - Les images Tiff, (DAPI et H2AX), à analyser doivent etre triees dans le meme ordre pour etre appariees correctement lors de l'analyse
//   - Lancer la macro par la touche [g] ou dans "Plugins>Macros>Run".
//     Elle boucle sur l'ensemble des images des deux repertoires.
//   - La macro propose automatiquement une selection pour chaque noyau analysé:
//     . Accepter ou rejeter la selection. ==> 'Yes' ajoute la selection au ROI manager, 'No' la supprime.
//     . Les selections sont renommees (prefixe "Cell_")
//   - En fin d'analyse, les resultats sont sauvés dans le repertoire parent:
//     . Les selections (ROI) sont sauvees dans les fichiers termines en '.zip'. (qu'elles soient multiple ou unique).
//     . Les resultats des analyses (mesures de surface, perimetre, position et ratio divers) des focis par noyau sont places dans les fichiers .xls
//     . Les stack d'images memorisent les masques ayant servis à la quantification.

requires("1.44c");
version = "9";
FS = File.separator();
print("\\Clear");

// Définition des repertoires de travail (images DAPI et H2AX)
dir = getDirectory("Choisir le repertoire racine de ss-repertoires DAPI et H2AX :");
working_path = dir;
print("\nRepertoire de travail:\n   " + working_path);

// Definition du chemin du sous repertoire des images DAPI
DAPI_path = working_path + "DAPI" + FS; print("Dossier image DAPI:\n   " + DAPI_path);

// Extraction des fichiers présents dans le repertoire DAPI
fileListDAPI = getFileList(DAPI_path);

// Definition du chemin du directory H2AX
H2AX_path = working_path + "H2AX" + FS; print("Dossier image H2AX:\n   " + H2AX_path);

// Extraction des fichiers présents dans le repertoire H2AX
fileListH2AX = getFileList(H2AX_path);

// Test de la longueur des piles d'images DAPI et H2AX
// Si nombre de fichiers DAPI et H2AX differents ==> erreur
if (fileListDAPI.length != fileListH2AX.length) {
        showMessage("Cette macro necessite des couples d'images",
```

```
            "Nombre de fichiers DAPI et H2AX differents!");
    exit();
}
// Definition des parametres par default pour la recherche des noyaux et des foci
MIN_DAPI_THRESH = 10000;      // taille minimum noyau ==> en pixel
COUNT_DAPI_THRESH  = 0.00;    // circularite minimum de la forme des noyaux
MIN_FOCI_THRESH = 10;         // taille minimum foci ==> en pixel
COUNT_FOCI_THRESH  = 0.00;    // circularite minimum de la forme des foci

msg = ""+fileListDAPI.length+" couple(s) d'images Dapi/H2ax a analyser...";

// Ouverture d'une boite de dialogue ImageJ permettant d'afficher les parametres de selection
// des zones de noyau en DAPI et des focis en H2AX
Dialog.create("Analyse de cellules fibroblaste  V."+version);
Dialog.addMessage("Choix des parametres 'Analyze particles'");
Dialog.addNumber("Taille minimum du noyau (en pixel):", MIN_DAPI_THRESH);
Dialog.addNumber("Circularite minimum du noyau:", COUNT_DAPI_THRESH);
Dialog.addNumber("Taille minimum d'un foci (en pixel):", MIN_FOCI_THRESH);
Dialog.addNumber("Circularite minimum d'un foci:", COUNT_FOCI_THRESH);

Dialog.addMessage(msg);
Dialog.addHelp("http://imagejdocu.tudor.lu/doku.php?id=macro:roi_color_coder");
Dialog.show();

// Saisie des différents paramètres de selection
MIN_DAPI_THRESH = Dialog.getNumber();
COUNT_DAPI_THRESH  = Dialog.getNumber();
MIN_FOCI_THRESH = Dialog.getNumber();
COUNT_FOCI_THRESH  = Dialog.getNumber();

// Initialisation des parametres pour l'histogramme des frequences de foci par noyau
maxFociByNuclei=500;
cumFreq = newArray(maxFociByNuclei);
nbTotalNuclei=0; // Stocke le nombre total de noyaux retenu

print("");
print("Taille minimum noyau  : "+MIN_DAPI_THRESH+"-infini");
print("Circularite minimum    : "+COUNT_DAPI_THRESH+"-1.00");
print("Taille minimum foci    : "+MIN_FOCI_THRESH+"-infini");
print("Circularite minimum    : "+COUNT_FOCI_THRESH+"-1.00");

// Nombre de couple d'images DAPI et H2AX associées
num_of_associations = fileListDAPI.length;

// Boucle principale sur les couples d'images
for (i=0; i<num_of_associations; i++) {

    // Barre de progression du traitement
    showProgress(i+1, num_of_associations);

    // Affichage du numero du couple d'image en cours de traitement
    print("\nAnalyse du couple d'images Dapi/H2ax (" +(i+1)+"/"+num_of_associations+")");

    //definition des noms des images à traiter
    fileDAPI=fileListDAPI[i];
    fileH2AX=fileListH2AX[i];
```

```
// Ouverture Image DAPI
open(DAPI_path+fileDAPI);

file = getTitle();
print("Ouverture du fichier DAPI: "+file);
run("Duplicate...", "title=DAPI");

// Convertion en 8 bits ==> passage en mode gris =0.299red+0.587green+0.114blue
selectWindow("DAPI");
run("8-bit");

// Reglage du seuil de definition des masques des noyaux
// Reglage automatique des niveaux haut et bas par une methode de la classe "ImageProcessor" d'ImageJ
// La definition des seuils est basé sur l'analyse de l'histogramme de l'image sélectionnée (image DAPI)
setAutoThreshold();

// Le seuillage pour obtenir les masques est appliqué par la commande "Convert to Mask" sur l'image DAPI sélectionnée
run("Convert to Mask");

// Definition des mesures effectuées sur les masques definissant les noyaux dans l'image DAPI selectionnée
// Ces mesures sont:
//  - l'aire sélectionnée en pixels carré,
//      - intensité moyenne, du gris dans l'aire sélectionnée
//  - position du centre de la selection,
//  - la circularité et le perimetre
// L'image sélectionnée par l'argument redirect="+fileDAPI+" est utilisée comme cible de l'analyse
// l'ensemble des données sont stockées dans un fichier excel

run("Set Measurements...", "area mean centroid redirect="+fileDAPI+" decimal=3
circularity perimeter fit");

// Selection des masques ==> La commande "Analyse Particle" compte et mesure les objets dans l'image seuillée et ajoute les ROI au ROI
Manager
// On ne retient que les masques qui remplissent un certain nombre de criteres:
//      - size ==> les objets hors de la gamme spécifiée sont ignorés; c'est à dire non compris entre MIN_DAPI_THRESH  et "infinity"
//  - circularity ==> les objets hors de la gamme spécifiée sont ignorés; c'est à dire non compris entre COUNT_DAPI_THRESH  et 1
//              0 correspond à un polygone infiniment allongé et 1 a une cercle parfait
// l'option "Show" spécifie quelle image ImageJ doit afficher aprés l'analyse :
//  Masks   ==> image binaire 8-bits contenant les zones remplies des objets mesurés
//  exclude ==> les objets mesurés touchant le bord sont annulés
//  include ==> les trous internes des objets  mesurés sont inclus
//  add     ==> les  objets sélectionnés sont ajoutés au ROI Manager

run("Analyze Particles...", "size="+MIN_DAPI_THRESH+"-Infinity pixel circularity="+
COUNT_DAPI_THRESH+"-1.00 show=Masks exclude include add");

// Sauvegarde de l'image des objets sélectionnés (masques)
run("Duplicate...", "title=Mask_kernels.tif");

//  Extraction du nombre de masques sélectionnés dans le ROI Manager
roi_size = roiManager("count");

// Initialisation des piles des noms des masques retenus (full path filename)
Cell_selection_names = newArray(roi_size);

// Initialisation des piles des flags des masques retenus
sel_names = newArray(roi_size);
```

```
// A ce stade, le  ROI Manager contient les masques potentiels que l'utilisateur peut accepter ou rejeter
// les ROI sont rangées dans une pile, on va extraire en tete de pile chaque ROI,
// l'afficher sur l'image DAPI, la valider ou pas.
// Si elle est validée, elle est rangée en queue de la pile du ROI Manager en le renomant, alors que la ROI
// originelle, en debut de pile est effacée

// On boucle sur les  masques potentiels des noyaux (ROI)

for (roi = 0; roi<roi_size; roi++) {
     // Activation de l'image avec les masques
     selectImage("Mask_kernels.tif");
     // Selectionne un item dans la liste du ROI Manager
     // l'index est egale à zero  ==> on selectionne le premier de la liste
     roiManager("Select", 0);
     // on efface ce qui est a l'exterieur de la ROI
     run("Clear Outside");
     // On remplace le polygone de la selection courante par son envelloppe convexe
     run("Convex Hull");
     // on ajoute cette ROI en fond de liste du ROI MAnager
     roiManager("Add");
     // On update les modifs du ROI Manager
     roiManager("Update");
     // on selectionne la derniere ROI en fin de ROI Manager
     roiManager("Select", roiManager("count")-1);
     // on la renome
     roiManager("Rename", "Cell_"+roi);
      // on reprend l'image DAPI dans le windows manager
     selectImage(fileDAPI);
     // on selectionne la derniere ROI en fin de ROI Manager (Cell_x )
     roiManager("Select", roiManager("count")-1);
     roiManager("Update");

     // on a l'image dapi avec la ROI autour du noyau correspondant
     // et l'on va demander à l 'utilisateur de valider cette ROI
     msg = "Etes-vous d'accord avec le masque du noyau?";
     if (getBoolean(msg) == true) {
          // mise a jour la table des etats de validation des ROI
          sel_names[roi]=true;
          // On crée une nouvelle image 8-bit  appelé "Mask"
          //  les pixels prennent la valeur  [255]  à l'intérieur et [0] à l'extérieur
          run("Create Mask");
          // On selectionne le masque dans le windows manager
          selectImage("Mask");
          // On incremente le nombre total de noyaux retenu
          nbTotalNuclei++;
     }
     else {
          // mise a jour la table des etats de validation des ROI
          sel_names[roi]=false;
          // On selectionne la derniere ROI en fin de ROI Manager (Cell_x )
          roiManager("Select", roiManager("count")-1);
          // On delete cette ROI portant le nom Cell_x qui n'est pas acceptée
          roiManager("Delete");
     }
     // On selectionne la premiere ROI en debut du ROI Manager (xxxx-yyyy)
     // Cette ROI est la ROI courante de la boucle, c'est celle qui a servi de base
```

```
        // pour la validation
        roiManager("Select", 0);
        // On delete  la ROI courante de la boucle du ROI Manager
        roiManager("Delete");
    }

    // A la sortie de la boucle, l'ensemble des ROI on ete tour a tour selectionnées
    // dans la pile du ROI Manager, en partant de la premiere, remplacée ou pas
    // en queue de la pile, puis effacée ==> il ne reste plus dans la pile du ROI Manager
    // que les ROI validées et renommées sous la forme Cell_X

    // Sauvegarde de l'ensemble des masques des noyaux retenus pour une image
    // Fichiers  de type zip avec pour extension tif_ROIs
    roiCellNames = working_path+fileH2AX+"_ROIs.zip";

    // Extraction du nombre de masque dans la liste du ROI manager
    ROI_Manager_Cell_size = roiManager("count");

    // Test si le ROI MAnager courant contient au moins un masque
    if (ROI_Manager_Cell_size>0) {
        selectWindow("ROI Manager");
        roiManager("Save", roiCellNames);
    }

    // Sauvegarde de chaque masque retenu par image
    // Fichiers  de type zip avec pour extension tif_ROIs_Cell_x (x numero noyau dans image)
    for (roi = 0; roi<ROI_Manager_Cell_size; roi++) {
        roiManager("Reset");
        open(roiCellNames);
        roiName = call("ij.plugin.frame.RoiManager.getName", roi);
        selectImage("Mask");
        roiManager("Select", 0);

        // on discarde tous les ROI au dela de l'index corant (roi)
        roi_tmp_size = roiManager("count");
        while (roi != roi_tmp_size-1) {
            roiManager("Select", roi+1);
            roiManager("Delete");
            roi_tmp_size = roiManager("count");
        }
        // on discarde tous les ROI en deca de l'index courant (roi)
        for (roi_tmp = 0; roi_tmp<roi; roi_tmp++) {
            roiManager("Select", 0);
            roiManager("Delete");
            // aprés l'annulation de la ROI courante en tete de la pile du ROI Manager
            // l'ensemble des ROI remonte==> au tour suivant, la prochaine ROI à annuler
            // sera en position 0 ( tete de pile)
        }

        selectImage("Mask");
        // Sauvegarde du ROI courant qui, maintenant, se trouve  seul  dans le manager
        roiManager("Select", 0);
        roiCellName = working_path+fileH2AX+"_ROIs_"+roiName+".zip";
        roiManager("Save", roiCellName);  // exemple: 'fibro_h2ax_100x_HR_18.tif_ROIs_Cell_2.zip'
        Cell_selection_names[roi] = roiCellName;
    }
```

```
    //cleanup
    selectImage("Mask_kernels.tif"); close();
    selectWindow(fileDAPI); close();
    selectWindow("DAPI"); close();

    // A ce stade, le ROI Manager ne contient plus rien!
    // Les masques correspondants à chaque noyau retenus "Cell_x"
    // sont sauvegardés individuellement dans des fichiers

    // Passage au traitement des Images H2AX

    if (ROI_Manager_Cell_size>0) {  // Seulement s'il y a au moins un masque retenu

        // Ouverture du fichier H2AX
        print("Ouverture du fichier H2AX:"+H2AX_path+fileListH2AX[i]);
        open(H2AX_path+fileListH2AX[i]);

        // Selection fenetre
        selectWindow(fileH2AX);

        // Convertion en 8 bits ==> passage en mode gris =0.299red+0.587green+0.114blue
        run("8-bit");
        // On duplique l'image H2AX sur laquelle on va appliquer un seuillage automatique
        // Cette nouvelle image va devenir l'image de masque des foci (on anticipe donc
        // en la nommant "Mask_foci_<Nom_du_fichier_H2AX>)
        run("Duplicate...", "title=Mask_foci_"+fileH2AX);

        // Fixation var. Mask_foci sur Summary table
        //Mask_foci = "Mask_foci.tif"; // fileListH2AX[i]

        // Reglage du seuil de definition des masques des foci
        // Reglage automatique des niveaux haut et bas par une methode de la classe "ImageProcessor" d'ImageJ
        // La definition des seuils est basé sur l'analyse de l'histogramme de l'image sélectionnée (image H2AX)
        setAutoThreshold();

        // Le seuillage pour obtenir les masques est appliqué par la
        // commande "Convert to Mask" sur l'image H2AX sélectionnée
        run("Convert to Mask");

        // Definition des mesures effectuées sur les masques definissant les noyaux dans l'image DAPI selectionnée
        // Ces mesures sont:
        //  - l'aire sélectionnée en pixels carré,
        //     - intensité moyenne, du gris dans l'aire sélectionnée
        //  - position du centre de la selection,
        //  - la circularité et le perimetre
        // L'image sélectionnée par l'argument redirect="+fileH2AX+" est utilisée comme cible de l'analyse
        // l'ensemble des données sont stockées dans un fichier excel

        run("Set Measurements...", "area mean centroid redirect="+fileH2AX+" decimal=3
circularity perimeter fit");

        // On vide le tableau des mesures par "Analyse" des masques
        run("Clear Results");

        // Boucle sur les masques des noyaux retenus  lors du traitement de l'image DAPI
        roiManager("reset");
        for(j=0; j<ROI_Manager_Cell_size; j++)
```

```
        {
            // ouverture du fichier contenant la ROI d'un des noyaux dans un fichier zip
            open(Cell_selection_names[j]);  // Cell selection
            // On recharge le masque du noyau courant
            roiCurrentName = call("ij.plugin.frame.RoiManager.getName", 0);
            // On test si le fichier existe
            if (!startsWith(roiCurrentName, "Cell_")) {
                showMessage("Summary loop", "Unexpected cell name '"+roiCurrentName+"'");
                exit("No Cell selection");
            }
            // On reactive l'image depuis le Windows manager
            //selectImage("Mask_foci.tif");
            selectImage("Mask_foci_"+fileH2AX);

            // On active la ROI sur l'image courante
            roiManager("Select", 0);

            // Recherche des focis a l'interieur de la ROI  par la commande run("Analyze Particles..)
            // Le tableau "result" est rempli a partir de l'analyse de l'imahge H2AX dans la ROI du noyau

            run("Analyze Particles...",
                "display results size="+MIN_FOCI_THRESH+"-Infinity pixel circularity="+
COUNT_FOCI_THRESH+"-1.00 exclude include add summarize");

            // On deselect la ROI courante correspondant au noyau
            // pour sauvegarder l'ensemble du ROI manager (le ROI Cell_x et le(s) ROI foci)
            roiManager("Deselect");
            // Sauvegarde des ROIs du manager (masque d'un noyau) augmenté des ROIs de foci
            roiManager("Save", Cell_selection_names[j]);
            roiManager("reset");

            // Sauvegarde Measurements des foci du noyau courant dans un fichier résultat excel
            selectWindow("Results");
            path = working_path+fileH2AX+"_"+roiCurrentName+".xls";
            saveAs("Measurements", path);

            // On print le resultat de l'analyse
            print("le noyau n°"+(j)+" compte "+nResults+" foci");
            // On upgrade l'histogramme du nombre de foci par noyau
            cumFreq[nResults]++;  // incremente compteur
            // Efface la table des resultats
            run("Clear Results");
        }
        // O ferme le maximum de fenetres
        //if (isOpen("Mask_foci.tif")) { selectImage("Mask_foci.tif"); close();
        if (isOpen("Mask_foci_"+fileH2AX)) { selectImage("Mask_foci_"+fileH2AX); close();
"title=
        }
    }

    // Fermeture des masques
    if (isOpen("Mask of DAPI")) { selectWindow("Mask of DAPI"); close(); }
    if (isOpen("Mask")) {selectImage("Mask"); close(); }

    if (ROI_Manager_Cell_size > 0) {

        //Espace réservé pour des Sauvegarde utilisateurs
```

```
    }
    else {
        showMessage("Analyse avortee", "Pas de selection dans l'image DAPI, donc pas de
foci...");
        print("Aucune selection de noyau dans l'image");
    }

    // Fermeture image H2AX
    if (isOpen(fileH2AX)) { selectWindow(fileH2AX); close(); }

} // Fin de boucle principale sur les couples d'images DAPI/H2AX

//select Log-window
selectWindow("Log");
// Save log file
path = working_path+fileH2AX+"_log"+".txt";
saveAs("Text", path);

// Summary (Recapitulatif de toutes les mesures sur foci par noyau)
//
// 'Analyze Particles' recapitule dans une table 'Summary':
//  - Slice    : Nom de l'image H2AX cible
//  - Count    : Nb de foci enumeres dans l'image
//  - Total Area : Cumul de l'aire de tous les foci
//  - ...
// Sortie des resultats pour chaque  noyau dans un fichier summary de type .txt
selectWindow("Summary");
path = working_path+"Summary.txt";
saveAs("Text", path);

//------------------------------------------------------------------
// Plot de verification ==> histogramme du nombre de foci par noyau
//------------------------------------------------------------------

// Tri des noyaux par nb de focis
print("\nRecapitulatif (max:" +  maxFociByNuclei+")");
xLimit=yLimit=0;
xMinLimit=yMinLimit=0;
nBins=0;
for (n=0; n<maxFociByNuclei; n++) {
    if (cumFreq[n]>0) {
        print("cumFreq["+n+"] = "+cumFreq[n]);
        nBins++;
        print(cumFreq[n]+" noyau de "+n+" foci");
        if (cumFreq[n]>yLimit) {yLimit = cumFreq[n];}
        if (n>xLimit) {xLimit = n;}
        if (xMinLimit==0) {xMinLimit = n;}
    }
}
print("");
print("Noyaux retenus = ", nbTotalNuclei);
print("yLimit         = ", yLimit);
print("xLimit         = ", xLimit);
print("xMinLimit      = ", xMinLimit);

print("\nPlot graph");
```

```
// Plot graph
run("Profile Plot Options...", "width=500 height=500 "); // remove default gridlines
Plot.create("Distribution du nombre de foci par noyau", "X(nb de foci)", "Y(nb de noyau
candidat)");
Plot.setLimits(xMinLimit-1, xLimit+1, yMinLimit, yLimit+1);
Plot.addText("Nb total de noyaux retenus: "+nbTotalNuclei, 0.02, 0.05);

Plot.setColor("blue");
for (n=0; n<maxFociByNuclei; n++) {
    if (cumFreq[n]>0) {
        Plot.setColor("blue");
        Plot.setLineWidth(2);
        Plot.drawLine(n, 0, n, cumFreq[n]);

        label=""+n;

        Plot.addText(label, ((n-(xMinLimit-1))/(xLimit-xMinLimit+2))-0.02, 0.99);
        // // Grid lines
        Plot.setColor("lightGray");
        Plot.setLineWidth(1);
        // Left Y axis
        Plot.setJustification("lef");
        Plot.addText(" "+d2s(cumFreq[n],0), 0, 1-(cumFreq[n]/(yLimit+1)));
        Plot.drawLine(0, cumFreq[n], xLimit+1, cumFreq[n]);
    }
}
print("\nFin plot graph");
Plot.show;

showMessage("Macro_GM_CCQ_v4.txt", "Fin d'analyse de la pile");
print("\nFin d'analyse de la pile");
// clear memory usage
call("java.lang.System.gc");
```