# GenExLib v.1.0

## C/C++ knihovna

*Referenční příručka*

B. Kovář, J. Schier, M. Kuneš



TA01010931 – GenEx - Systém pro podporu vyhodnocování metody FISH

# Obsah

# 1

# Úvod

Cílem projektu GenEx[1] je vyvinout prototyp snímacího zařízení a obslužného software pro podporu vyhodnocování analýzy FISH (fluorescenční in-situ hybridizace), se zaměřením na vyhledávání nízkofrekvenčních mozaikových aberací. Snímací zařízení je vyvíjeno s tím, že ho bude možné použít i pro vyhodnocování jiných fluorescenčních analýz. Dá se tedy předpokládat, že některé části metodologie, popsané v kapitole 3, bude možné znovu použít v jiných aplikacích. Z tohoto důvodu je aplikace zpracování obrazu vyvíjena ve formě C/C++ knihovny. Při implementaci této knihovny jsme se snažili:

- vytvořit snadno použitelnou C/C++ knihovnu pro zpracování obrazu z fluorescenčního mikroskopu,
- knihovnu, která bude univerzálně použitelná i pro jiné úlohy zpracování obrazu v mikroskopii,
- která nebude omezena licenčními poplatky (a to včetně externích knihoven),
- bude optimalizovaná pro snímací zařízení vyvíjené na pracovišti Camea.

Základní informace o instalaci, použitých metodách a jejich implementacích čtenář nalezne v kapitolách 2 a 3. V kapitole 4 uvádíme příklad aplikace, která s použitím knihovny GenExLib v obraze detekuje buněčná jádra, spočítá základní charakteristiky detekovaných objektů a výsledky (výstupní obraz a tabulka příznaku) uloží na disk počítače. Kapitola 5 pak přináší referenční popis použitých C++ tříd, metod a atributů. Shrnutí základních vlastností této knihovny a její další vývoj uvádíme v závěru této příručky.

[1] GenEx - Systém pro podporu vyhodnocování metody FISH. Projekt TA01010931 programu Alfa, Technologická agentura ČR, http:\www.tacr.cz

# 2

# Instalace

Knihovna GenExLib využívá funkcí knihovny OpenCV[2]. Tato knihovna je vytvářena pod BSD licencí a je zdarma dostupná jak pro akademické, tak komerční projekty. V současné verzi (2.4.3) obsahuje více než 2500 optimalizovaných algoritmů pro zpracování obrazu v reálném čase. Je možné ji používat v programovacích jazycích C/C++, Python a brzo také v Java, pod operačními systémy Windows, Linux, Android a Mac. V tomto textu předpokládáme použití obou knihoven pod operačním systémem Windows. Všechny doporučení se tak týkají instalace a používání pod tímto operačním systémem.

Pro vývoj knihovny GenExLib byla použita verze OpenCV 2.2.0. Instalace této verze knihovny na počítač, kde bude používána knihovna GenExLib, je nezbytná. Dále předpokládáme, že je na počítači nainstalováno některé z vývojových prostředí (doporučené je Microsoft Visual Studio), případně alespoň kompilátor jazyka C/C++.

## 2.1 Instalace OpenCV v.2.2.0

Poslední stabilní verze, ale i verze předchozí, je možné získat na stránkách projektu OpenCV na webové adrese http://sourceforge.net/projects/opencvlibrary/files/. Jedná se o instalační balíčky, které byly předem zkompilovány a optimalizovány pro použití s Microsoft Visual Studio 2010 a novějším. Průběh instalace je znázorněn na Obrázku 1.

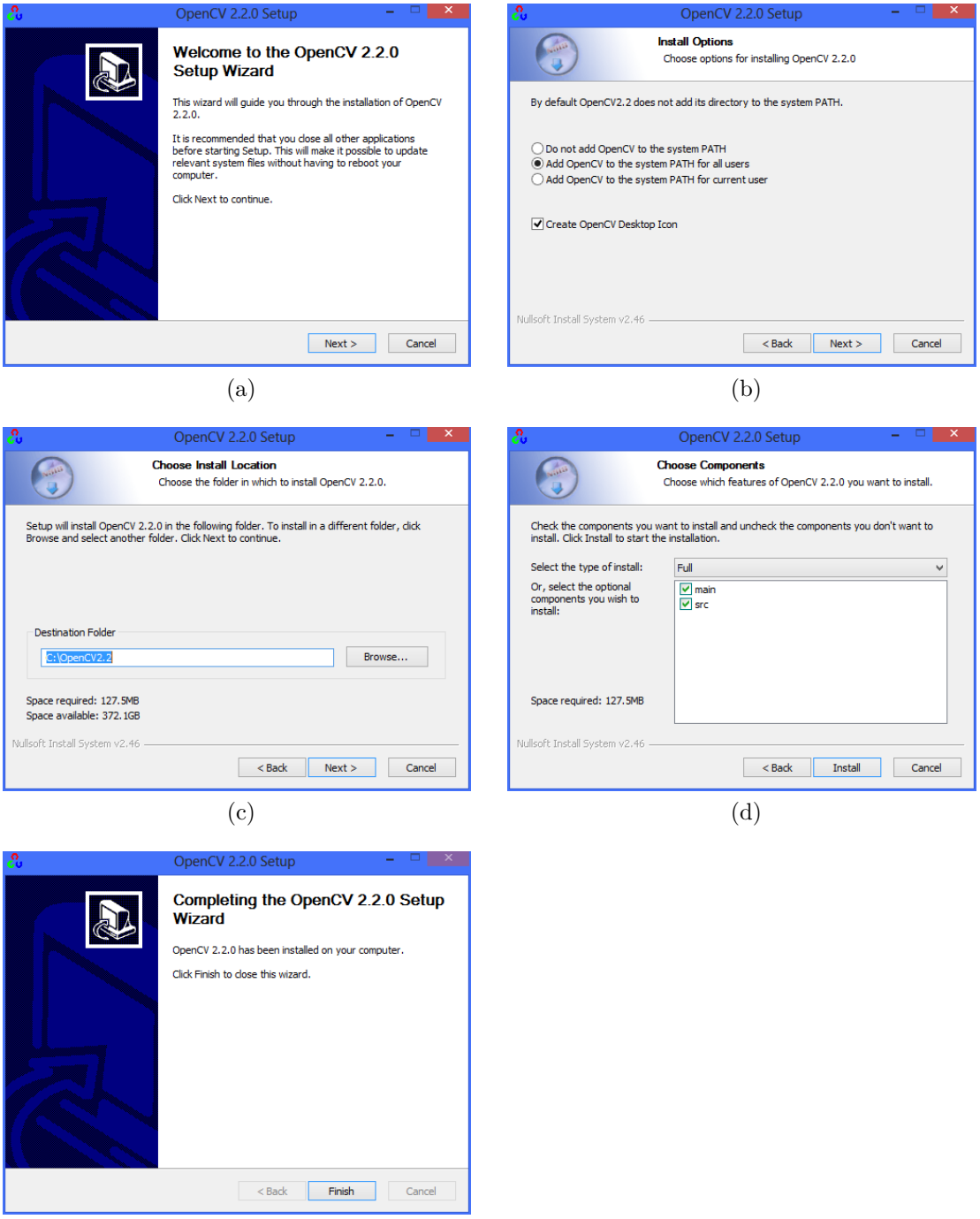
Při instalaci doporučujeme zvolit:

1. Add OpenCV to the system PATH for all users (Obrázek 1-b)
2. Destination folder: `C:\OpenCV2.2` (Obrázek 1-c), případně kořenový adresář jiného disku.
3. Zvolíme plnou instalaci (Obrázek 1-d)

Vzhledem k tomu, že se do systémové proměnné PATH přidává cesta k binárním souborům OpenCV, tak je doporučen restart počítače. Tím je OpenCV nainstalováno a připraveno k použití. Dodejme, že pokročilejší uživatel pravděpodobně zvolí možnost stažení nejnovějších verzí zdrojových kódů z SVN repozitáře[3] OpenCV a vlastní překlad knihovny. Potřebné informace, jak takovou instalaci provést, nalezne uživatel na webových stránkách OpenCV (http://opencv.willowgarage.com/wiki/).

[2] **Open** Source **C**omputer **V**ision

[3] http://code.opencv.org/svn/opencv/trunk/opencv




Obrázek 1: Instalace OpenCV

## 2.2 Instalace GenExLib v.1.0

Knihovna GenExLib je distribuována ve formátu ZIP archivu a přeložena pro 32bitovou verzi operačního systému Windows. Je dostupná partnerům projektu GenEx na interních webových stránkách a to včetně zdrojových kódů. Ostatním pak na vyžádání, bez zdrojových kódů. Archiv obsahuje adresáře uvedené v tabulce 1. Vlastní instalace pak spočívá

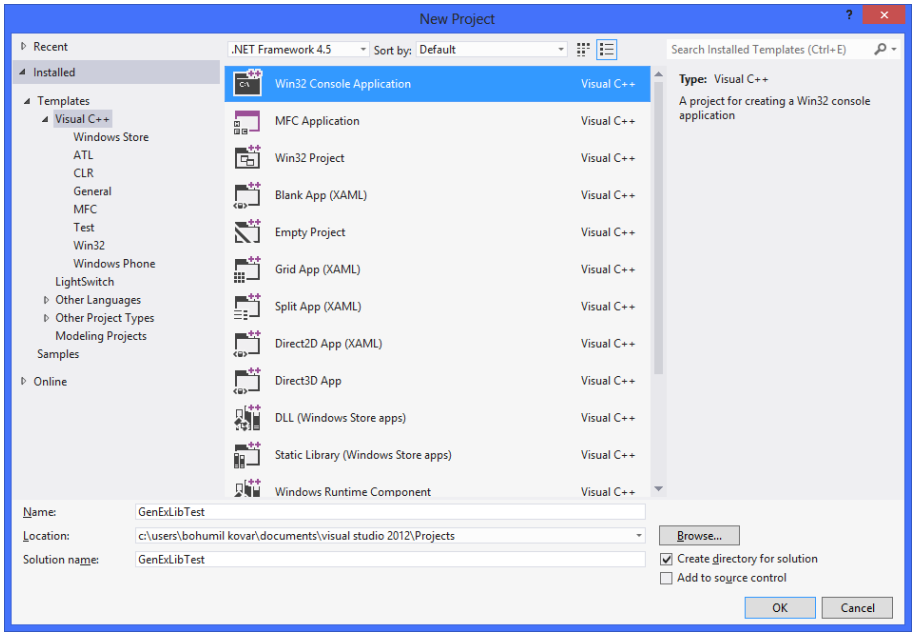
v rozbalení archivu na disk počítače.

| | |
|---|---|
| DOC | Dokumentace |
| EXAMPLES | Přeložené demonstrační programy. Pro jejich spuštění je třeba mít nainstalovaný MSVC run-time a OpenCV |
| INCLUDE | Include soubory knihovny GenExLib |
| LIB | Statická verze knihovny v debug a release verzi |
| SRC | Zdrojové kódy knihovny a demo aplikací |

Tabulka 1: Adresářová struktura distribuce GenExLib

## 2.3 Konfigurace Visual Studio 2012

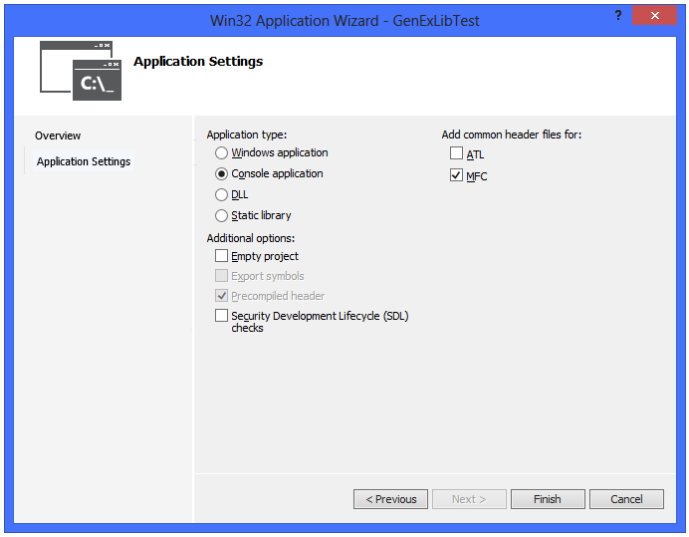
Knihovnu GenExLib je možné používat v libovolném C/C++ projektu Visual Studia. Konfiguraci Visual Studia 2012, pro použití s touto knihovnou, si ukážeme na demo aplikaci uvedené v kapitole 4. Předpokládejme, že na počítači máme správně nainstalováno vývojové prostředí Microsoft Visual Studio[4], knihovnu OpenCV ve verzi 2.2.0 a adresář s knihovnou GenExLib. Začněme tím, že ve Visual Studiu vytvoříme nový projekt (Obrázek 2 a 3):




Obrázek 2: Vytvoření nového projektu ve Visual Studiu 2012

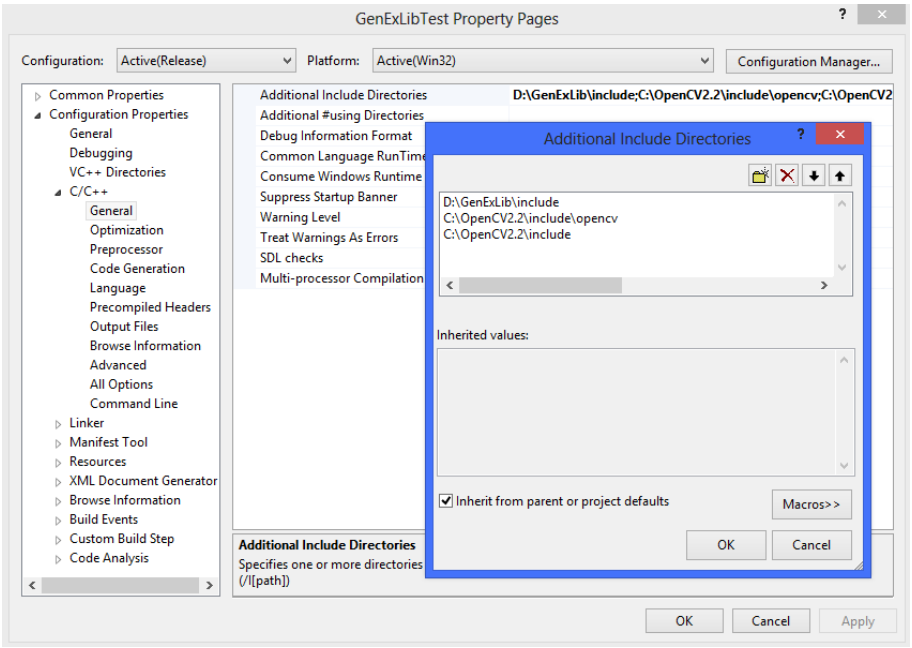
1. z menu zvolíme **File, New, Project**
2. v levém panelu, pod **Visual C++**, vybereme **Win32**
3. ve středu dialogu zvolíme **Win32 Console Application**

[4] http://www.microsoft.com/visualstudio/eng/downloads




Obrázek 3: Vytvoření nového projektu ve Visual Studiu 2012

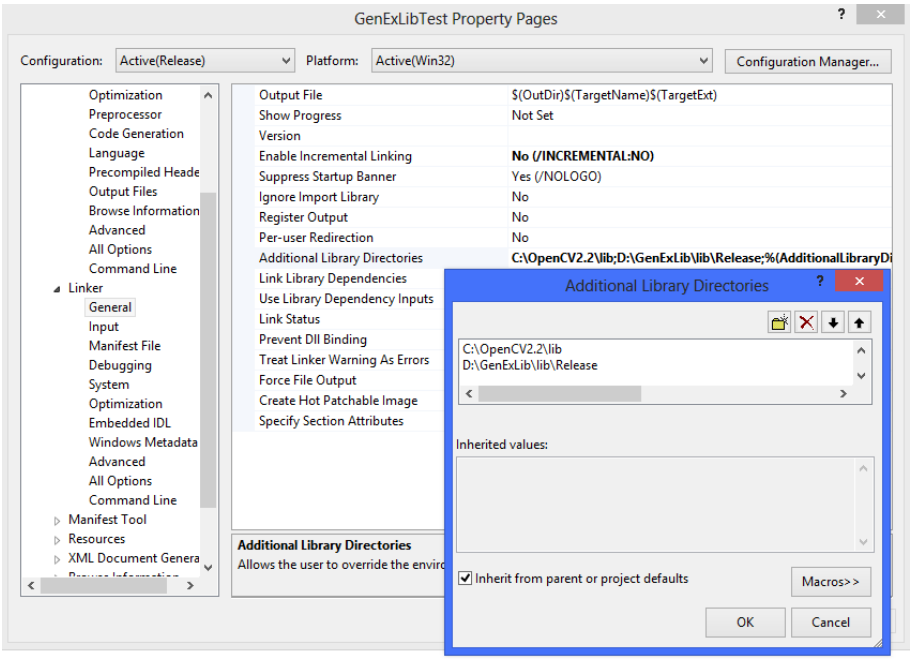
4. zvolíme jméno projektu, např. GenExLibTest
5. na straně **Overview** zvolíme **Next**
6. a v **Application settings** jako typ aplikace **Console Application**
7. dále můžeme zvolit podporu předkompilovaných hlavičkových souborů, případně knihovny MFC
8. volbou **Finish** vytvoříme projekt




Obrázek 4: Přidání adresářů s include soubory

Tím máme vytvořený prázdný projekt, který budeme dál upravovat. Začněme tím, že vytvořený zdrojový kód (soubor GenExLibTest.cpp) upravíme tak, aby odpovídal výpisu z kapitoly 4. Pokud se nyní pokusíme projekt přeložit, tak skončí chybovou hláškou, že není možné nalézt některé hlavičkové soubory, definované v prvních řádcích zdrojového kódu. Upravme tedy konfiguraci projektu. Zvolme:

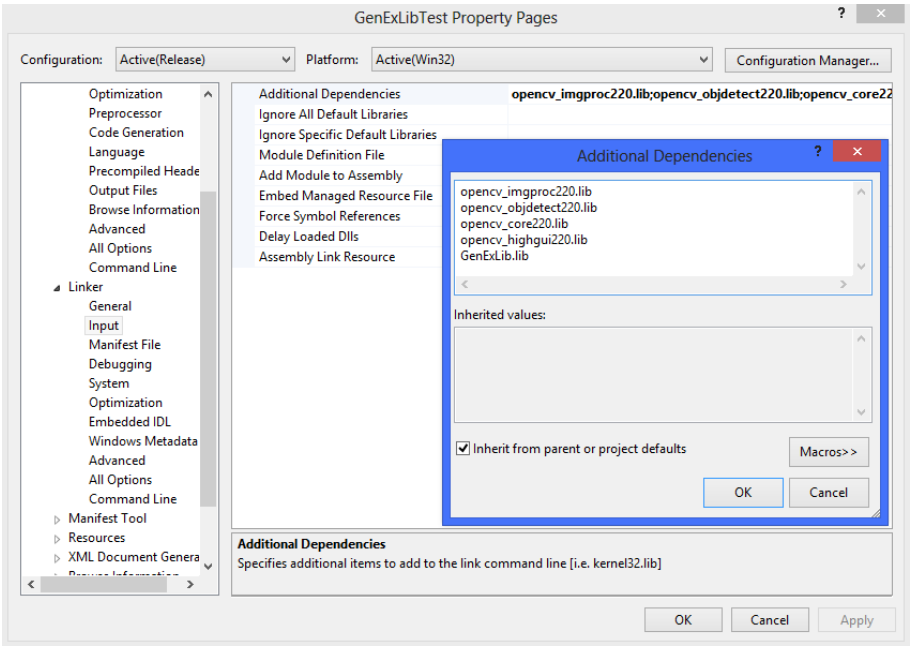
1. z menu **Project, Properties**
2. v levém panelu zvolíme **Configuration Properties, C/C++**
3. a pak ve střední části dialogu **Additional Include Directories**
4. a přidáme adresáře, ve kterých jsou hlavičkové soubory OpenCV a GenExLib




Obrázek 5: Konfigurace linkeru – přidání dalších adresářů s knihovnami

Nyní, pokud projekt přeložíme, tak zjistíme, že není možné přilinkovat knihovny OpenCV a GenExLib. Zvolme tedy:

1. z menu **Project, Properties**
2. v levém panelu zvolíme **Configuration Properties, Linker, General**
3. a pak ve střední části dialogu **Additional Library Directories**
4. a přidáme adresáře, ve kterých jsou potřebné knihovny
5. ty pak specifikujeme v **Configuration Properties, Linker, Input** v kolonce **Additional Dependencies**




Obrázek 6: Konfigurace linkeru – specifikace dalších knihoven

# 3

# Metodologie a implementace

V této kapitole bude popsána knihovna GenExLib z pohledu použitých metod, algoritmů a jejich implementací v jazyce C/C++ s použitím knihovny OpenCV. U čtenáře předpokládáme alespoň základní znalosti v oblasti zpracování obrazu a programování. Podrobný popis a vysvětlení používaných pojmů a metod lze nalézt v odborné literatuře, například [1, 2, 3]. Všechny popisované algoritmy jsou implementovány jako metody v třídách CPreprocessing, CBinarize, CCell a CStat. Reference jednotlivých metod jsou uvedeny v kapitole 5. Obecný postup konstrukce objektu dané třídy a volání metod pak je tato:

```
1  #include "GenExLib.h"
2  ...
3  int main(int argc, TCHAR* argv[])
4  {
5      C{class_name} *object_name = new C{class_name}(<constructor_parameters>);
6      <output> = object_name->object_method(<method_parameters>);
7      delete object_name;

9  return;
10 }
```

## 3.1 Předzpracování obrazu

Kvalita snímků z fluorescenčního mikroskopu se může měnit a to i v rámci jednoho vzorku. Může docházet ke změnám v rozložení jasu, různé hladině obrazového šumu vzniklého při snímání obrazu atp. Důvodem mohou být residua fluoroforu v pozadí snímku, nevyhnutelná i při přesném dodržení předepsaného laboratorního postupu přípravy vzorků. Další vliv zcela jistě bude mít konstrukce snímacího zařízení a drobné změny expozičního nastavení kamery v průběhu snímání vzorku. Vzhledem k tomu, že se tyto odchylky projeví ve všech barevných kanálech, tak je vhodné tyto změny potlačit a minimalizovat tím jejich vliv na další algoritmy.

### 3.1.1 Korekce obrazového šumu

Šum v obraze můžeme potlačit tím, že obraz „rozmažeme“ použitím jednoduchého filtru. Nejpoužívanější filtr je lineární, ve kterém se výstupní hodnota pixelu na pozici

$[i, j]$, označme ji $g(i, j)$, spočte jako vážený součet vstupních pixelů $f(i, j)$ v určitém, obdélníkovém okolí $\delta$ bodu $[i, j]$

$$g(i, j) = \sum_{k,l \in \delta} f(i + l, j + l)h(k, l) \tag{1}$$

Tato matematická operace (1) je označována jako konvoluce funkce $f$ s konvolučním jádrem $h$, kde konvoluční jádro obsahuje koeficienty (váhy) filtru. V knihovně GenExLib jsou implementovány tyto metody odstranění šumu:

1. BLUR - lineární konvoluce s $h(k, l) = \frac{1}{h_w \cdot h_h}, \forall k, l$, kde $h_w$ a $h_h$ jsou dimenze obdélníkového konvolučního jádra $h$.
2. GAUSSIAN - lineární konvoluce s gaussovským konvolučním jádrem
3. MEDIAN - hodnota $g(i, j)$ je stanovena jako *medián* z čtvercového okolí hodnot funkce $f(i, j)$.
4. BILATERAL - jednoduchá metoda odstranění šumu, která neovlivní hrany v obraze. Bližší popis je možné najít například zde [4].

Pro ilustraci použití těchto metod pro potlačení šumu uvedeme příklad kódu:

```
1  #include "GenExLib.h"
2  ...

4  CString filename = _T("test.tif"); // 8bit gray scale image
5  IplImage* pSrcImage = cvLoadImage(CT2CA(filename),-1);
6  IplImage* pOutBlur = cvCloneImage(pSrcImage);
7  IplImage* pOutGaussian = cvCloneImage(pSrcImage);
8  IplImage* pOutMedian = cvCloneImage(pSrcImage);
9  IplImage* pOutBilateral = cvCloneImage(pSrcImage);

11 CPreprocessing* imgproc = new CPreprocessing(pSrcImage);
12 pOutBlur = imgproc -> Blur(3,3);
13 pOutGaussian = imgproc -> Gaussian(7,7,2.0);
14 pOutMedian = imgproc -> Median(5,5);
15 pOutBilateral = imgproc -> Bilateral(5,5,30.0,30.0);
16 delete imgproc;
17 ...
```

### 3.1.2 Jasové korekce

Rozložení jasu v jednotlivých snímcích, i u stejného vzorku, se může lišit. Základní (a často jedinou) globální informaci o rozložení jasu v obraze nám dává *histogram*. Histogram můžeme chápat jako četnosti výskytu jednotlivých úrovní jasu v obraze. Lze jej tedy použít jako odhad hustoty pravděpodobnosti rozložení jasu v obraze. Na základě této informace můžeme:

1. měnit dynamický rozsah jasu (histogram stretching) a

2. vyrovnat rozložení jasu (histogram equalization).

### Histogram stretching

Předpokládejme, že šedotónový obraz v jednom z barevných kanálů obsahuje úrovně šedé od $In_L$ do $In_H$. My požadujeme, aby výsledkem této transformace byl obraz, kde tyto úrovně šedé budou mezi $Out_L$ a $Out_H$. Výsledkem tedy bude obraz s vyšším nebo nižším rozsahem úrovní jasu a tedy i vyšším nebo nižším kontrastem. Tuto transformaci lze popsat touto rovnicí:

$$g(i,j) = \frac{Out_H - Out_L}{In_H - In_L}(f(i,j) - In_L) + Out_L \tag{2}$$

kde $f(i,j)$ je intenzita jasu vstupního obrazu na pozici $i,j$ a $g(i,j)$ intenzita jasu po transformaci.

### Histogram equalisation

Ekvalizace histogramu je algoritmus, který změní rozložení jasu v obraze tak, aby se v něm všechny úrovně vyskytovaly přibližně se stejnou četností. Ekvalizace umožňuje u obrazu s vysokým kontrastem zvýraznit špatně rozeznatelné detaily s nízkým kontrastem. Tato transformace je definována rovnicí:

$$H_e(j) = \frac{q_k - q_0}{M \cdot N} \sum_{i=0}^{j} H_{in}(i) + q_0 \tag{3}$$

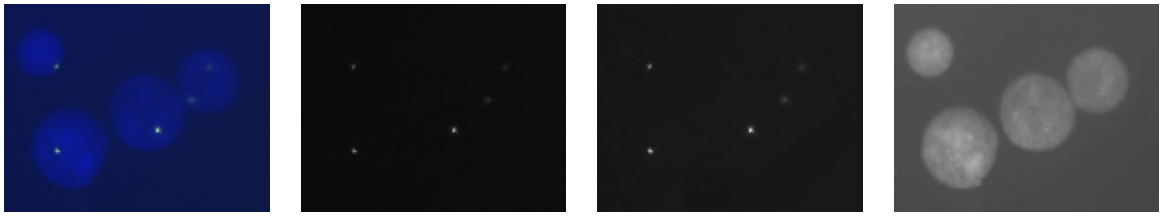
kde $H_{in}$ je původní histogram, $H_e$ ekvalizovaný histogram, $q_0$ až $q_k$ je rozsah požadovaných úrovní šedé v transformovaném obraze se šířkou $M$ a výškou $N$ pixelů. Příklad zdrojového kódu pro ekvalizaci histogramu a korekci kontrastu:

```
1  #include "GenExLib.h"
2  ...
3  CString filename = _T("test.tif"); // 8bit gray scale image
4  IplImage* pSrcImage = cvLoadImage(CT2CA(filename),-1);
5  IplImage* pEqvImage = cvCreateImage( cvGetSize(pSrcImage), pSrcImage->depth,1 );
6  IplImage* pStrImage = cvCreateImage( cvGetSize(pSrcImage), pSrcImage->depth,1 );

8  CPreprocessing* imgproc = new CPreprocessing(pSrcImage);
9  pOutImage = imgproc -> EqualizeHist(pSrcImage);
10 pStrImage = imgproc -> StretchHist(pSrcImage,30,180);
11 delete imgproc;
12 ...
```

## 3.2 Detekce buněčných jader a signálů

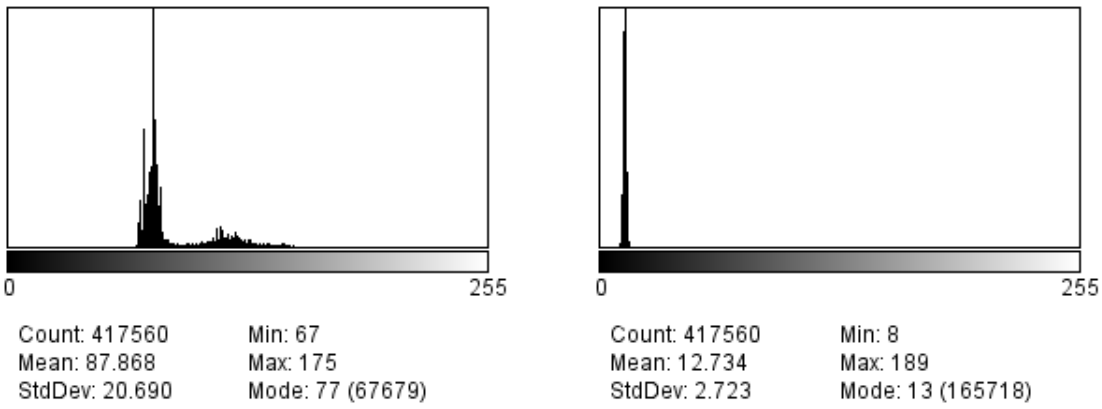
Při detekci buněčných jader a fluorescenčních signálů využijeme skutečnosti, že buněčná jádra a metafázní shluky jsou patrné zejména v modrém (DAPI) barevném kanálu, signály



(a) Barevný obraz (b) Červený kanál (c) Zelený kanál (d) Modrý kanál

Obrázek 7: Separace barevného obrazu do 8bitových RGB kanálů.

pak, dle zvoleného barviva v červeném nebo zeleném kanálu (Obrázek 7). Z uvedeného obrázku je zřejmé, že jsou buněčná jádra i signály vzhledem k téměř homogennímu pozadí dostatečně kontrastní. Tento předpoklad potvrzuje i histogram rozložení jasu v modrém kanále, kde je pozadí definováno výrazným maximem (Obrázek 8). To je dáno tím, že v




(a) Modrý kanál (b) Červený kanál

Obrázek 8: Histogram rozložení jasu v modrém a červeném kanále.

obou případech je plocha pozadí větší než plocha objektů v popředí. Za povšimnutí stojí, že signály, vzhledem k zanedbatelné ploše, nejsou v histogramu červeného kanálu identifikovatelné jako lokální maximum. S tím budeme muset při volbě vhodné segmentační metody počítat. Na základě výše uvedených skutečností jsme jako segmentační metodu zvolili **prahování**. Pokročilé segmentační algoritmy založené na modelech a optimalizačních metodách (level-sets, aktivní kontury, atp.), případně statistických metodách nejsou v knihovně GenExLib ve verzi 1.0 implementované.

Segmentace prahováním transformuje (binarizuje) vstupní obraz na objekty v popředí $b(i,j) = 1$, a na pozadí $b(i,j) = 0$.

$$b(i,j) = \begin{cases} 1, & \text{když } f(i,j) \geq T \quad (4) \\ 0, & \text{když } f(i,j) < T \quad (5) \end{cases}$$

Vzhledem k tomu, že počet segmentovaných objektů (a tedy i plocha popředí) není konstantní, tak není možné použít metody založené na pevném prahu. Z tohoto důvodu jsme se zaměřily na metody, u kterých je práh stanoven adaptivně buď analýzou his-

togramu, nebo statistickou optimalizací. Při experimentech se nám osvědčily tyto dvě metody výpočtu prahu:

1. OTSU [5] a
2. Triangle [6].

**OTSU**
Šedotónový obraz obsahuje celkem $M \times N$ pixelů s úrovněmi šedi od $[1, \cdots, L]$. Počet pixelů s úrovní šedi $i$ označme $f_i$ . Potom pravděpodobnost výskytu úrovně šedi $i$ v obraze bude definována

$$p_i = \frac{f_i}{M \cdot N} \tag{6}$$

V případě prahování s jedním prahem budou úrovně šedi rozděleny do dvou tříd – třída $C_1$ s úrovněmi šedi $[1, \cdots, t]$ a třída $C_2$ s úrovněmi šedi $[t+1, \cdots, L]$. Potom distribuce pravděpodobností pro jednotlivé intenzity bude pro tyto třídy

$$C_1 : = \{p_1/\omega_1(t), \cdots, p_t/\omega_1(t)\} \tag{7}$$
$$C_2 : = \{p_{t+1}/\omega_2(t), \cdots, p_L/\omega_2(t)\} \tag{8}$$

kde

$$\omega_1(t) = \sum_{i=1}^{t} p_i \quad \text{a} \quad \omega_2(t) = \sum_{i=t+1}^{L} p_i. \tag{9}$$

Dále střední hodnoty intenzity pro tyto dvě třídy jsou

$$\mu_1(t) = \sum_{i=1}^{t} \frac{i \cdot p_i}{\omega_1(t)} \quad \mu_2(t) = \sum_{i=t+1}^{L} \frac{i \cdot p_i}{\omega_2(t)}. \tag{10}$$

Pokud $\mu_T$ bude střední hodnota intenzity jasu v celém obrázku, pak je zřejmé, že $\omega_1\mu_1 + \omega_2\mu_2 = \mu_T$ a $\omega_1 + \omega_2 = 1$. Vážený průměr rozptylů intenzit mezi třídou $C_1$ a $C_2$ pak je definován rovnicí

$$\sigma_\omega^2 = \omega_1(\mu_1 - \mu_T)^2 + \omega_2(\mu_2 - \mu_T)^2. \tag{11}$$

Otsu ověřil, že optimální práh $t^*$ získáme tehdy, když $\sigma_\omega^2$ bude minimální:

$$t^* = \text{ArgMax}\{\sigma_\omega^2(t)\}, \quad 1 \le t \le L, \tag{12}$$

Pokračujme tím, jak je tato metoda výpočtu segmentačního prahu implementována.

```
1  IplImage* CBinarize::OTSU(void)
2  {
3      if(m_pImage == NULL)
4          return NULL;
5      else {
6          IplImage* imgBin = cvCreateImage(cvSize(m_pImage->width,m_pImage->height),
7                                           IPL_DEPTH_8U,1);
8          const int GRAYLEVEL = 256;
9          #define MAX_BRIGHTNESS 255;
10         int  hist[GRAYLEVEL];
```

```
11        double prob[GRAYLEVEL];
12        double omega[GRAYLEVEL]; // prob of graylevels
13        double myu[GRAYLEVEL]; // mean value for separation
14        double max_sigma;
15        double sigma[GRAYLEVEL]; // inter-class variance
16        int threshold; // threshold for binarization
17        // Histogram generation
18        memset((int*) hist , 0, GRAYLEVEL * sizeof(int) );
19        CvSize size = cvGetSize(m_pImage);
20        for (int i = 0; i < size.height; ++i) {
21            unsigned char* pData = (unsigned char*) (m_pImage->imageData +
22                                        i * m_pImage->widthStep);
23            for (int j = 0; j < size.width; ++j) {
24                int k = (int)((unsigned char) *(pData+j));
25                hist[k]++;
26            }
27        }
28        int area = size.width * size.height;
29        // calculation of probability density
30        for (int i = 0; i < GRAYLEVEL; ++i )
31            prob[i] = (double) ((double)hist[i] / (double)area);
32        // omega & myu generation
33        omega[0] = prob[0];
34        myu[0] = 0.0;
35        for (int i = 1; i < GRAYLEVEL; i++) {
36            omega[i] = omega[i-1] + prob[i];
37            myu[i] = myu[i-1] + (i*prob[i]);
38        }
39        // sigma maximization
40          // -- sigma stands for inter-class variance and determines optimal threshold value
41        threshold = 0;
42        max_sigma = 0.0;
43        for (int i = 0; i < GRAYLEVEL-1; i++) {
44            if (omega[i] != 0.0 && omega[i] != 1.0)  {
45                sigma[i] = ((myu[(GRAYLEVEL-1)]*omega[i] - myu[i]) *
46                            (myu[GRAYLEVEL-1]*omega[i] - myu[i])) / (omega[i]*(1.0 - omega[i]));
47            }
48            else
49                sigma[i] = 0.0;
50            if (sigma[i] > max_sigma) {
51                max_sigma = sigma[i];
52                threshold = i;
53            }
54        }
55        // binarization output into imgBin
56        for (int y = 0; y < size.height; ++y) {
57            unsigned char* pData = (unsigned char*) (m_pImage->imageData +
58                                        (y * m_pImage->widthStep));
```

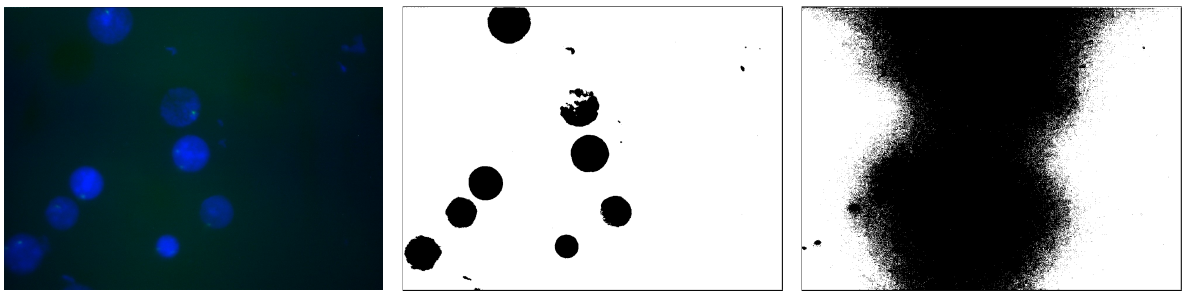
```
59            unsigned char* pDataBin = (unsigned char*) (imgBin->imageData +
60                                        (y * imgBin->widthStep));
61            for (int x = 0; x < size.width; ++x) {
62                if ( *(pData+x) > threshold){
63                    *(pDataBin+x) = MAX_BRIGHTNESS;
64                }
65                else
66                    *(pDataBin+x) = 0;
67            }
68        }
69    return imgBin;
70    }
71 }
```

Vlastní použití výpočtu prahu a binárního obrazu s použitím knihovny GenExLib je pak velmi snadné:

```
1  #include "GenExLib.h"
2   ...
3  CString filename = _T("test.tif"); // 8bit gray scale image
4  IplImage* pSrcImage = cvLoadImage(CT2CA(filename),-1);
5  IplImage* pBinImage = cvCreateImage( cvGetSize(pSrcImage), pSrcImage->depth,1 );

7  CBinarize* binarize = new CBinarize(pSrcImage);
8  pBinImage = binarize->OTSU();
9  delete(binarize);
10  ...
```

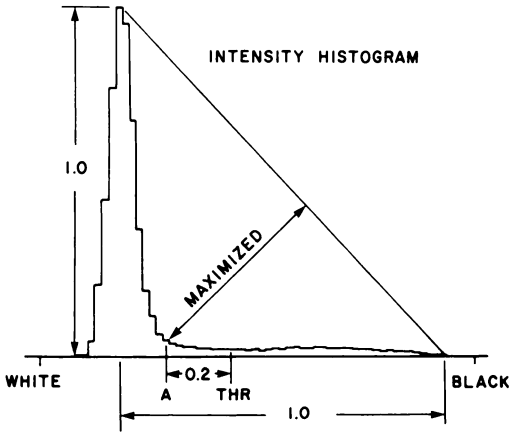
Z rovnic 7,8 a obrázku 8(a) je zřejmé, že tato segmentační metoda je vhodná pro obraz, ve kterém jsou objekty v popředí dostatečně velké, tedy v histogramu identifikovatelné jako lokální maximum. To není splněno v případě segmentace signálů, jejichž velikost je v porovnání s pozadím zanedbatelná. Z tohoto důvodu tato metoda při segmentaci signálů selhává, zvláště pokud pozadí není dostatečně homogenní – například vlivem nestejnoměrného nasvícení vzorku. Výsledky segmentace buněčných jader a signálů touto metodou uvádíme v obrázku 9. Pro srovnání, výsledky segmentace metodou Triangle, která bude popsána v další části textu, jsou zobrazeny na obrázku 11.



Obrázek 9: Segmentace buněčných jader a fluorescenčních signálů metodou OTSU

**TRIANGLE**

Tuto metodu publikoval Zack a kol. [6] už v roce 1977. Tedy zhruba ve stejné době jako Otsu [5]. Triangle algoritmus pro stanovení segmentačního prahu je založen na analýze tvaru histogramu. Je vhodný zejména pro obraz, ve kterém jas objektů v popředí vytváří v histogramu nevýrazná lokální maxima. Případně jas těchto objektů není možné v histogramu identifikovat. Princip algoritmu je zřejmý z obrázku 10. Nechť $a$ je nejmenší a




Obrázek 10: Triangle algoritmus stanovení segmentačního prahu. Zdroj [6]

$b$ největší nenulová hodnota jasu na x-ové ose histogramu. Nalezneme maximální hodnotu histogramu $H_{max}(i)$ a jí odpovídající úroveň jasu $i$; $i \in \langle a, b\rangle$. Dále zjistíme, který z intervalů $\langle a, i\rangle, \langle i, b\rangle$ je větší. V něm budeme hledat segmentační práh. V dalším textu předpokládejme, že větší interval je stejně jako na obrázku 10 ten na pravé straně od maxima. Pro $\forall j \in \langle i, b\rangle$ veďme úsečku spojující $H_{max}(i)$ s $j$ a změřme její kolmou vzdálenost $L(j)$ od histogramu. Práh $A$ zvolíme tam, kde je $L(j)$ maximální. Takto určený práh bývá někdy upraven přičtením konstantní hodnoty jako kompenzace nehomogenního pozadí.

Stejně jako v případě OTSU algoritmu uveďme i pro metodu Triangle její implementaci v knihovně GenExLib.

```
1  #include <math.h>

3  int CBinarize::Triangle(void)
4  {
5      if(m_pImage == NULL)
6          return NULL;
7      else {
8          IplImage* image = m_pImage;
9          int width = image->width;
10         int height = image->height;
11         IplImage* imageBin = cvCreateImage(cvSize(image->width, image->height), 8, 1);
12         // create histogram
13         int data[256] = {0};
14         for(int i=0; i<h;i++) {
15             for(int j=0;j<w;j++) {
16                 CvScalar m, n;
17                 m = cvGet2D(image, i, j);
```

```
18                  data[(int)m.val[0]]++;
19              }
20          }
21          // find min and max
22          int min = 0, dmax=0, max = 0, min2=0;
23          for (int i = 0; i < data.length; i++) {
24           if (data[i]>0){
25              min=i;
26              break;
27           }
28        }
29          if (min>0) min--; // line to the (p==0) point, not to data[min]
30          //  the other extreme.
31        for (int i = 255; i >0; i-- ) {
32           if (data[i]>0){
33              min2=i;
34              break;
35           }
36        }
37          if (min2<255) min2++; // line to the (p==0) point, not to data[min]
38          for (int i =0; i < 256; i++) {
39           if (data[i] >dmax) {
40              max=i;
41              dmax=data[i];
42           }
43        }
44          // find which is the furthest side
45        boolean inverted = false;
46        if ((max-min)<(min2-max)){
47           // reverse the histogram
48           inverted = true;
49           int left  = 0;            // index of leftmost element
50           int right = 255; // index of rightmost element
51           while (left < right) {
52              // exchange the left and right elements
53              int temp = data[left];
54              data[left]  = data[right];
55              data[right] = temp;
56              // move the bounds toward the center
57               left++;
58              right--;
59           }
60           min=255-min2;
61           max=255-max;
62        }
63          if (min == max){
64           return min;
65        }
```

```
66          // describe line by nx * x + ny * y - d = 0
67          double nx, ny, d;
68          // nx is just the max frequency as the other point has freq=0
69          nx = data[max];  //-min; // data[min]; // lowest value bmin = (p=0)% in the image
70          ny = min - max;
71          d = sqrt(nx * nx + ny * ny);
72          nx /= d;
73          ny /= d;
74          d = nx * min + ny * data[min];
75          // find split point
76          int split = min;
77          double splitDistance = 0;
78          for (int i = min + 1; i <= max; i++) {
79              double newDistance = nx * i + ny * data[i] - d;
80              if (newDistance > splitDistance) {
81                  split = i;
82                  splitDistance = newDistance;
83              }
84          }
85          split--;
86          if (inverted) {
87              // The histogram might be used for something else, so let's reverse it back
88              int left  = 0;
89              int right = 255;
90              while (left < right) {
91                  int temp = data[left];
92                  data[left]  = data[right];
93                  data[right] = temp;
94                  left++;
95                  right--;
96              }
97              return (255-split);
98          }
99          else
100             return split;
101     }
102 }
```

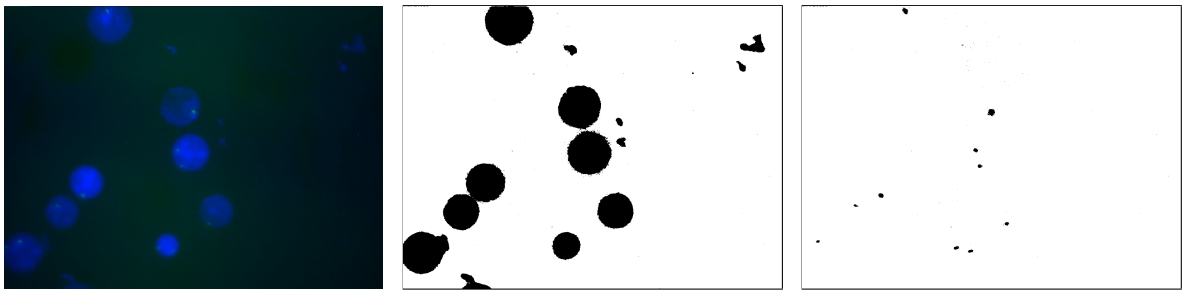
Vlastní použití výpočtu prahu a binárního obrazu s použitím knihovny GenExLib je pak velmi snadné:

```
1 #include "GenExLib.h"
2 ...
3 CString filename = _T("test.tif"); // 8bit gray scale image
4 IplImage* pSrcImage = cvLoadImage(CT2CA(filename),-1);
5 IplImage* pBinImage = cvCreateImage( cvGetSize(pSrcImage), pSrcImage->depth,1 );

7 CBinarize* binarize = new CBinarize(pSrcImage);
```

```
8  int threshold = binarize->Triangle();
9  delete(binarize);

11 cvThreshold( pSrcImage, pBinImage, threshold, 255, CV_THRESH_BINARY );
12 ...
```



Obrázek 11: Segmentace buněčných jader a fluorescenčních signálů metodou Triangle

## 3.3 Popis segmentovaných objektů

Základním prvkem pro popis segmentovaných objektů v knihovně GenExLib je *kontura*. Kontura je sekvence bodů, která v obraze definuje křivku. V OpenCV je kontura reprezentována jako dynamický spojový seznam, ve kterém každý prvek nese zároveň informaci (odkaz) na prvek následující. V naší implementaci konturu určíme přímo z binarizovaného obrazu algoritmem, který v roce 1985 publikoval Suzuki [8]. Uveďme ukázku zdrojového kódu pro popis segmentovaných objektů konturou:

```
1  #include "GenExLib.h"
2  ...
3  CString filename = _T("test.tif"); // 8bit gray scale image
4  // pSrcImage -- vstupni barevny obraz
5  // pBinImage -- binarni segmentovany obraz
6  CvSeq* contours = 0;
7  CvMemStorage* storage = cvCreateMemStorage(0);

9  int level = cvFindContours(pBinImage, storage, &contours, sizeof(CvContour),
10                    CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0) );
11 ...
```

Pro visualizaci segmentovaných objektů používáme funkce, které jsou dostupné v OpenCV. V současné verzi je možné pro každý segmentovaný objekt vykreslit jeho konturu (implementace)

```
12 CvScalar contour_color = CV_RGB(0,255,0); // contour color
13 // draw all contours
14 for(CvSeq* c = contours; c != NULL; c = c->h_next)
15         cvDrawContours(pSrcImage,c,contour_color,contour_color,0,2,8,cvPoint(0,0));
```

případně objekt aproximovat opsaným obdelníkem

```
16 CvRect rect;
17 for(CvSeq* c = contours; c != NULL; c = c->h_next) {
18         rect = cvBoundingRect(c);
19         cvRectangle(pSrcImage, cvPoint(rect.x, rect.y),
20                                cvPoint(rect.x + rect.width, rect.y + rect.height),
21                                CV_RGB(255, 0, 0));
22 }
```

nebo elipsou.

```
16 CvBox2D32f* box;
17 for(CvSeq* c = contours; c != NULL; c = c->h_next) {
18     CvPoint center; CvSize size; int i;
19     int count = c>total;
20     // Number point must be more than or equal to 6
21     if(count < 6)
22         continue;
23     // Alloc memory for contour point set.
24     PointArray = (CvPoint*)malloc( count*sizeof(CvPoint) );
25     PointArray2D32f= (CvPoint2D32f*)malloc( count*sizeof(CvPoint2D32f) );
26     // Alloc memory for ellipse data.
27     box = (CvBox2D32f*)malloc(sizeof(CvBox2D32f));
28     // Get contour point set.
29     cvCvtSeqToArray(c, PointArray, CV_WHOLE_SEQ);
30     // Convert CvPoint set to CvBox2D32f set.
31     for(i=0; i<count; i++) {
32         PointArray2D32f[i].x = (float)PointArray[i].x;
33         PointArray2D32f[i].y = (float)PointArray[i].y;
34     }
35     // Fits ellipse to current contour.
36     cvFitEllipse(PointArray2D32f, count, box);
37     // Convert ellipse data from float to integer representation.
38     center.x = cvRound(box->center.x);
39     center.y = cvRound(box->center.y);
40     size.width = cvRound(box->size.width*0.5);
41     size.height = cvRound(box->size.height*0.5);
42     box->angle = -box->angle;
43     // Draw ellipse.
44     cvEllipse(image04, center, size, box->angle, 0, 360,
45                     CV_RGB(0,0,255), 1, CV_AA, 0);
46     // Free memory.
47     free(PointArray);
48     free(PointArray2D32f);
49     free(box);
50 }
51 ...
```

## 3.4 Příznaky

Příznaky (základní popisné charakteristiky) detekovaných objektů jsou vypočítány pouze pro ty, které s velkou pravděpodobností odpovídají buněčným jádrům nebo fluorescenčním signálům. Experimentálně jsme ověřily, že uspokojivé výsledky získáme tehdy, pokud pro filtraci detekovaných objektů použijeme jejich plochu a excentricitu. Vlastní filtrování objektů pak probíhá takto:

```
1  // CvSeq* pCur sekvence kontur
2  while(pCur != 0L)
3  {
4      //i++;
5      CStat* stat = new CStat(pCur);
6      stat->CalculateStatistics();
7      if(stat->stats.eccentricity < 0.5) {
8          CvSeq* pTmp;
9          pTmp = pCur;
10         if(pCur->h_prev == 0L) {  // first element
11             pCur = pCur->h_next;
12             if(pCur != 0L)
13                 pCur->h_prev = 0L;
14             pTmp->h_next = 0L;
15             *cells  = pCur;
16         }
17         else if(pCur->h_next == 0L) { // last element
18             pCur->h_prev->h_next = 0L;
19             pCur = 0L;
20             pTmp->h_prev = 0L;
21         }
22         else {
23             pCur->h_next->h_prev = pCur->h_prev;
24             pCur->h_prev->h_next = pCur->h_next;
25             pCur = pCur->h_next;
26             pTmp->h_prev = 0L;
27             pTmp->h_next = 0L;
28         }
29     }
30     else
31         pCur = pCur->h_next;
32     delete stat;
33 }
```

Po této operaci jsou ze spojového seznamu detekovaných kontur odstraněny ty, které nesplňují naše požadavky na excentricitu, případně plochu. Na zbytku jsou pak napočítány příznaky. Knihovna GenExLib má v aktuální verzi implementovány tyto popisné statistiky:

1. plocha objektu,

2. délka opsané kontury,

3. excentricita,

4. souřadnice těžiště.

Vlastní výpočet příznaků je pak snadný

```
for(CvSeq* c = m_pContours; c != NULL; c = c->h_next) {
     CStat* statistics  = new CStat(c);
     statistics ->CalculateStatistics();

    if( statistics ->stats.area == 0.0)
    {
        idx++;
        continue;
    }
     // napocitane  statistiky
    //  statistics ->stats.area,
    //  statistics ->stats.length,
    //  statistics ->stats.eccentricity,
    //  statistics ->stats.c_of_gr.x,
    //  statistics ->stats.c_of_gr.y,
    idx++;
}
```

Výsledkem jsou například data uvedená v tabulce 2.

# 4

# Příklady použití

Vytvořme s použitím knihoven OpenCV a GenExLib aplikaci, která v obraze z fluorescenčního mikroskopu detekuje buněčná jádra, označí je konturou, každému detekovanému jádru přiřadí identifikátor a napočítá jeho základní parametry. Pokud v Microsoft Visual Studiu 2012 vytvoříme projekt pro novou konzolovou Win32 aplikaci, postupem popsaným v kapitole 2.3, bude automaticky vygenerovaný zdrojový kód vypadat takto:

```
1  #include "stdafx.h"
2  #include "cv.h"
3  #include "cxcore.h"
4  #include "highgui.h"
5  #include "GenExLibTest.h"
6  #include "GenExLib.h"

8  #ifdef _DEBUG
9  #define new DEBUG_NEW
10 #endif

12 CWinApp theApp;

14 using namespace std;
15 using namespace cv;

17 int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
18 {
19     int nRetCode = 0;

21     HMODULE hModule = ::GetModuleHandle(NULL);

23     if (hModule != NULL)
24     {
25         // initialize MFC and print and error on failure
26       if (!AfxWinInit(hModule, NULL, ::GetCommandLine(), 0))
27       {
28           _tprintf(_T("Fatal Error: MFC initialization failed\n"));
29           nRetCode = 1;
30       }
31       else
32       {
```

```
33            /*- misto pro vlastni kod algoritmu */
34        }
35    }
36    else
37    {
38        // TODO: change error code to suit your needs
39        _tprintf(_T("Fatal Error: GetModuleHandle failed\n"));
40        nRetCode = 1;
41    }
42    return nRetCode;
43 }
```

Do kódu jsme pouze vložili načtení hlavičkových souborů knihovny OpenCV na řádcích 2 - 4, hlavičkový soubor knihovny GenExLib.h na řádku 6 a pak jmenný prostor OpenCV na řádku 15. Vlastní kód algoritmu budeme vkládat na řádek 33. Nejprve načteme vstupní obraz. Předpokládejme, že soubor bude ve stejném adresáři jako výsledný program a bude se jmenovat "test.tif".

```
1 CString m_strImageFileName = _T("test.tif");
2 IplImage* m_pImage = cvLoadImage(CT2CA(m_strImageFileName),-1);
```

Obraz je tvořen třemi barevnými kanály RGB. Víme, že buněčná jádra jsou výrazná zejména v modrém kanálu. Proto si vytvoříme tři pomocné 8bitové monochromatické obrázky, pro každý barevný kanál jeden a vstupní obraz do nich rozložíme. Dále pak budeme pracovat pouze s modrým barevným kanálem.

```
3 IplImage* r_image = cvCreateImage( cvGetSize(m_pImage), m_pImage->depth,1 );
4 IplImage* g_image = cvCreateImage( cvGetSize(m_pImage), m_pImage->depth,1 );
5 IplImage* b_image = cvCreateImage( cvGetSize(m_pImage), m_pImage->depth,1 );
6 cvSplit(m_pImage,b_image,g_image,r_image,NULL);
```

V dalším kroku z obrazu odstraníme šum (řádek 9) a provedeme jeho binarizaci.

```
7  IplImage* thr_image = cvCreateImage(cvGetSize(m_pImage), m_pImage->depth,1 );
8  IplImage* temp = cvCreateImage( cvGetSize(m_pImage), m_pImage->depth,1 );
9  cvSmooth( b_image, temp, CV_GAUSSIAN, 7, 7 );
10 b_image = temp;
11 cvReleaseImage(&temp);

13 CBinarize* binarize = new CBinarize(b_image);
14 thr_image = binarize->OTSU();
15 delete(binarize);
```

Na řádku 13 dynamicky vytvoříme nový objekt *binarize*. Pro prahování použijeme algoritmus OTSU, který je volán na řádku 14 jako metoda objektu *binarize*. Objekt je pak na řádku 15 ukončen a dealokován destruktorem. Tím získáme binarizovaný obraz, který obsahuje bílé objekty v popředí na černém pozadí. V dalším kroku hranice těchto objektů

popíšeme konturou.

```
16 CvSeq* contours = 0;
17 CvMemStorage* storage = cvCreateMemStorage(0);
18 IplImage* segm_image = cvCloneImage(thr_image);

20 int level = cvFindContours(segm_image, storage, &contours, sizeof(CvContour),
21                            CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0) );
```

*Contours* je spojový seznam, který obsahuje struktury, popisující jednotlivé spočtené kontury. Jejich celkový počet je uložen v proměnné *level*. V závěrečné fázi vytvoříme CSV soubor s popisnými příznaky jednotlivých jader, která budou, pro snadnou identifikaci, ve výstupním obraze označena.

```
22 CString logname = _T("test.csv");
23 CString imagename = _T("test_labels.jpg");

25 CLogFile* log = new CLogFile(logname,m_pImage,contours);
26 log->LabelImage();
27 log->SaveImage(imagename);
28 log->CSVStatistics();
29 delete(log);
```

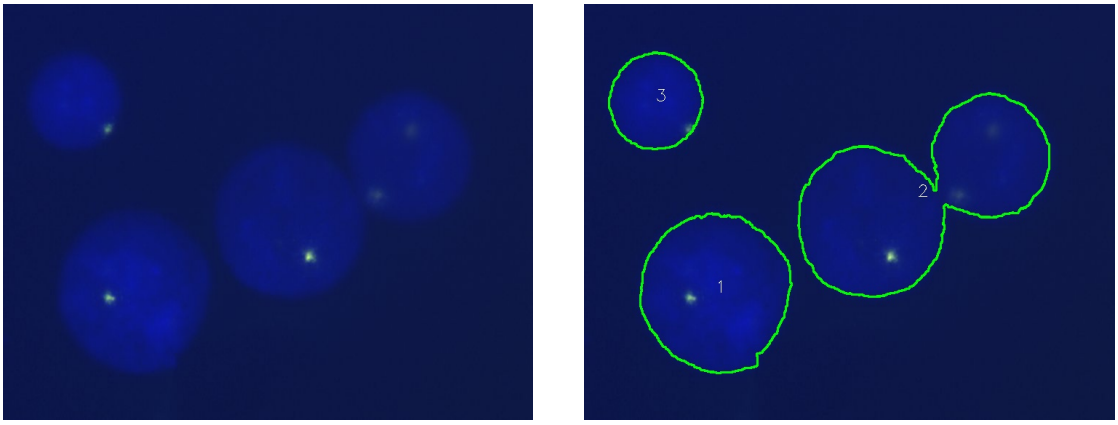
Proměnné na řádcích 22 a 23 definují jména CSV souboru a souboru s výstupním obrazem. Oba soubory budou vytvořeny ve stejném adresáři, ve kterém bude spuštěný program GenExLibTest. Na řádku 25 je dynamicky vytvořen objekt *log* a pomocí metod LabelImage() a CSVStatistics() jsou vytvořena výstupní data. Objekt je poté ukončen destruktorem. Na závěr dealokujeme i ostatní proměnné.

```
22 cvReleaseMemStorage(&storage);
23 cvReleaseImage(&m_pImage);
24 cvReleaseImage(&r_image);
25 cvReleaseImage(&g_image);
26 cvReleaseImage(&b_image);
27 cvReleaseImage(&segm_image);
28 cvReleaseImage(&thr_image);
```

Program spustíme z příkazové řádky. Výstupem je tabulka 2 a obrázek 12 s označenými jádry

| ID | Area | Length | Eccentricity | CoG (x) | CoG (y) |
|---|---|---|---|---|---|
| 1 | 34579.00000 | 753.06810 | 0.34054 | 180.49921 | 396.28864 |
| 2 | 53940.00000 | 1264.31283 | 0.88500 | 458.85843 | 264.30641 |
| 3 | 13171.50000 | 457.54624 | 0.29372 | 98.41728 | 133.32914 |

Tabulka 2: Základní popisné charakteristiky detekovaných objektů




Obrázek 12: Vstupní a výstupní obraz programu GenExLibTest

# 5

# Reference funkcí

## 5.1 Class Preprocessing

CPreprocessing::CPreprocessing

### Konstruktor

```
CPreprocessing();
CPreprocessing(IplImage* image);
```

### Metody

```
IplImage* Blur(int width, int height);
IplImage* Gaussian(int width, int height, flout sigma);
IplImage* Median(int width, int height);
IplImage* Bilateral(int width, int height, float a, float b);
IplImage* EqualizeHist(IplImage* pSrcImage);
IplImage* StretchHist(IplImage* pSrcImage, int lower, int upper);
```

### Atributy

```
IplImage* m_pImage;
```

### Destruktor

```
// CPreprocessing* preproc = new CPreprocessing(IplImage* image)
delete preproc;
```

## 5.2 Class Binarize

CBinarize::CBinarize

### Konstruktor

```
CBinarize();
CBinarize(IplImage* image);
```

### Metody

```
IplImage* OTSU();
int Triangle();
```

### Atributy

```
IplImage* m_pImage;
```

### Destruktor

```
// CBinarize* binarize = new CBinarize(IplImage* image)
delete binarize;
```

## 5.3 Class Cell

CCell::CCell

### Konstruktor

```
CCell();
CCell(IplImage* image);
CCell(IplImage* image, IplImage* c_img, IplImage* i_img);
CCell(CvSeq* p_contours, IplImage* image);
```

### Metody

```
CvSeq* Find(int type);
void Cell(CvSeq** cells);
void Interphase(CvSeq** cells);
```

### Atributy

```
CvSeq* m_pContours;
IplImage* m_pImage;
IplImage* m_pcImage;
IplImage* m_piImage;
```

### Destruktor

```
// CCell* cell = new CCell(IplImage* image)
delete cell;
```

## 5.4 Class Stat

```
CStat::CStat
```

### Struktury

```
typedef struct {
    CvMoments moments;
    double eccentricity;
    double area;
    double length;
    CvPoint2D64f c_of_gr;
}  Statistics ;
```

### Konstruktor

```
CStat(void);
CStat(CvSeq* contour);
```

### Metody

```
Statistics  CalculateStatistics ();
```

### Atributy

```
CvSeq* m_pContour;
Statistics  stats;
```

### Destruktor

```
// CStat* stats = new CStat(CvSeq* contour)
delete stats;
```

## 5.5 Class LogFile

```
CLogFile::CLogFile
```

### Konstruktor

```
CLogFile(void);
CLogFile(CString logname, IplImage *image, CvSeq* contours);
```

### Metody

```
void LabelImage();
void CSVStatistics();
void SaveImage(CString filename);
```

### Atributy

```
CString m_strLog;
IplImage* m_pImage;
CvSeq* m_pContours;
```

### Destruktor

```
// CLogFile* logfile = new CLogFile(CString logname, IplImage *image, CvSeq* contours);
delete logfile;
```

## 5.6 Class Visualization

```
CVisualization::CVisualization
```

### Konstruktor

```
CVisualization();
CVisualization(IplImage* image);
CVisualization(IplImage* image, CvSeq* contours);
```

### Metody

```
void DrawContours();
void DrawBoundingBox();
void DrawEllipse();
void SaveImage(CString filename);
```

### Atributy

```
IplImage* m_pImage;
CvSeq* m_pContours;
```

### Destruktor

```
// CVisualization* visualization = new CVisualization(IplImage* image, CvSeq* contours);
delete visualization;
```

## 5.7 Class Config

Třída Config není v současné verzi knihovny implementována. Předpokládáme ale, že bude nutné před vlastním vyhodnocením vzorku systém konfigurovat například dle použitého hardware (objektiv mikroskopu, kamera), fluorescenčních barviv, typu experimentu atp. Tato třída nám do budoucna umožní systém snadno nastavit pro optimální zpracování konkrétní úlohy.

# 6

# Závěr

V tomto dokumentu jsme představili C/C++ knihovnu GenExLib ve verzi 1.0, která vznikla jako jeden z výsledků projektu GenEx. K závěru roku 2012 obsahuje algoritmy, které byly vyvinuté a experimentálně ověřené v průběhu dosavadního řešení projektu. Knihovna umožňuje v obraze z fluorescenčního mikroskopu prahováním segmentovat objekty, dle jejich velikosti a kruhovitosti je separovat na buněčná jádra nebo fluorescenční signály a přiřadit k nim jejich základní popisné charakteristiky. Ty jsou pak používány k dalšímu – statistickému vyhodnocení vzorku. Je pravděpodobné, že knihovna bude průběžně rozšiřována o další metody, které vzniknou dalším vývojem systému pro podporu vyhodnocení metody FISH. Knihovna je volně přístupná řešitelskému týmu projektu GenEx včetně zdrojových kódů. Ostatním je dostupná na vyžádání v binární formě.

Tento dokument vznikl díky podpoře projektu **TA01010931** - GenEx – Systém pro podporu vyhodnocování metody FISH, programem **Alfa** Technologické agentury České Republiky. http:\www.tacr.cz


# Literatura


[1] Linda G. Shapiro and George C. Stockman, *Computer Vision*, Prentice Hall, Massachusetts, 2nd Edition, 2001, ISBN 0-13-030796-3.

[2] Milan Sonka, Vaclav Hlavac and Roger Boyle, *Image Processing, Analysis, and Machine Vision*, Thomson, 2008, ISBN 0-495-08252-X.

[3] Bruce Eckel, *Thinking in C++: Introduction to Standard C++*, Prentice Hall, 2 edition, 2000, ISBN: 0-13-979809-9.

[4] C. Tomasi and R. Manduchi, *Bilateral Filtering for Gray and Color Images*, Proceedings of the 1998 IEEE International Conference on Computer Vision, Bombay, India, 1998.

[5] Otsu N., *A Threshold Selection Method from Gray-Level Histograms*, IEEE Transactions on Systems, man, and Cybernetics, 1979, pp 62-66.

[6] Zack GW, Rogers WE, Latt SA, *Automatic measurement of sister chromatid exchange frequency*, J. Histochem. Cytochem, 1977, 25 (7): 741–53, PMID 70454

[7] Canny, J. *A Computational Approach To Edge Detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.

[8] Suzuki, S. and Abe, K., *Topological Structural Analysis of Digitized Binary Images by Border Following*, CVGIP 30 1, pp 32-46, 1985