SUPPLEMENTAL INFORMATION

SUPPLEMENTAL TABLES

Supplemental Table S1. Overview of parameters recorded by the "endomembrane script". Parameters displayed in PERL graphics and used in genetic screening are highlighted in grey. Spots = membrane compartments.

	Output parameter	**Description**
1	Number of cells in stack	Number of recognized cells per image; cells that were too big (> 40 000 pixel) or small (< 800 pixel) were excluded (possibly false recognitions)
2	Number of intracellular spots in stack	Number of spots within recognized cells
3	Total number of spots in stack	
4	% of intracellular spots in stack	If >25 % of spots lie within recognized cells, average number of spots/image area is calculated.
5	Average area of cells in stack	Average area of cells in pixel (±SD)
6	% cell area in stack	
7	Average number of spots/cell/stack	Average number of spot/cell/stack (±SD)
8	Average number of spots/recognized area/stack	
9	Average intensity of spots/leaf	Average brightness of spots per picture
10	Average area of spots/leaf	Average area of spots per picture
11	Average length of spots/leaf	Average length of spots per picture
12	Average half width of spots/leaf	Average half width of spots per picture
13	Average width to length ratio of spots/leaf	Average width to length ratio of spots (±SD)
14	Average roundness of spots/leaf	Average roundness of spots (±SD)
15	Average contrast of spots/leaf	Average contrast of spots (±SD)
16	Average area of cells/leaf	Average area of cells in pixel (±SD)

17	Average number of spots/cell/leaf	Average number of spots per recognized cell (±SD) per leaf (stack 1-5)
18	Number of valid stacks in well	Number of stacks with good (valid) pictures in well
19[a]	Average number of spots/image area	Average number of intracellular spots calculated per 100 % image area per seedling
20[b]	Average number of total spots/image	Average number of recognized spots (total number of spots) per image per seedling
21[c]	Average number of spots/cell	Average number of spots per cell per seedling

[a] Parameter 19 was calculated as follows:

$$\text{Average number of spots/image area} = \frac{100 * \sum(\text{number of intracellular spots in stack})}{\sum(\text{\% cell area in stack})}$$

[b] Paramter 20 was calculated as follows:

$$\text{Average Number of total spots/image} = \frac{\sum(\text{total number of spots in stack})}{\text{number of valid stacks in well 1 and well2}}$$

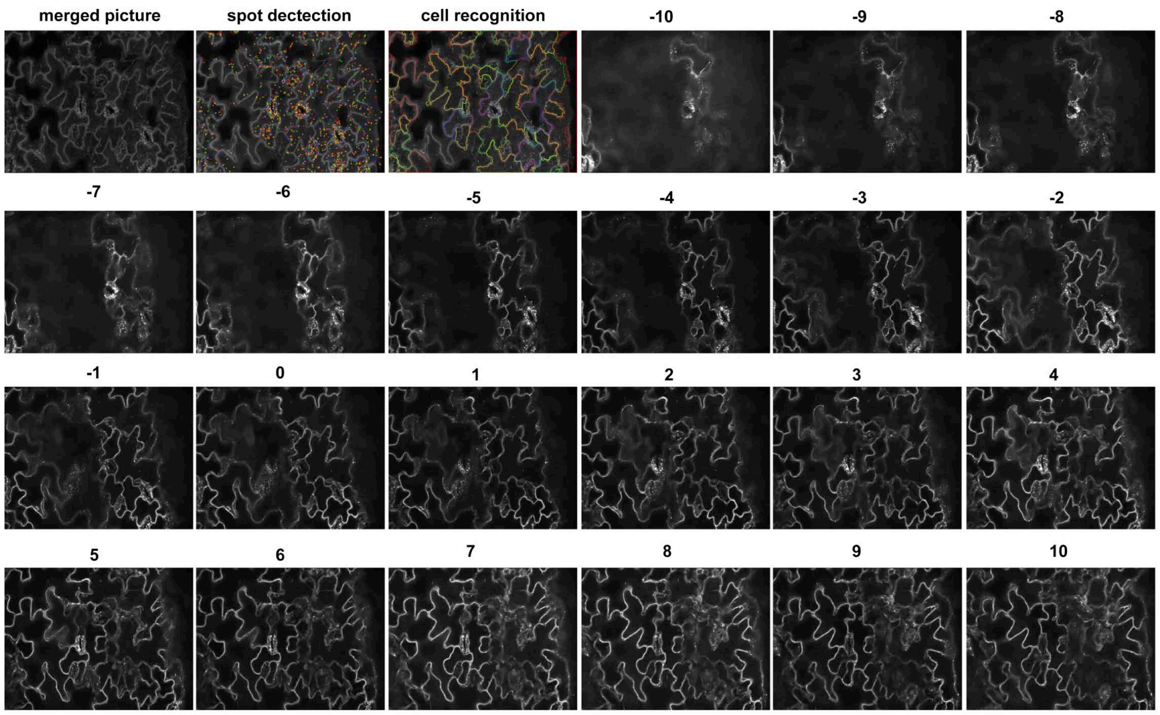
[c] Parameter 21 was calculated as follows:

$$\text{Average number of spots/cell} = \frac{\sum(\text{number of intracellular spots in stack})}{\sum(\text{number of cells in stack})}$$

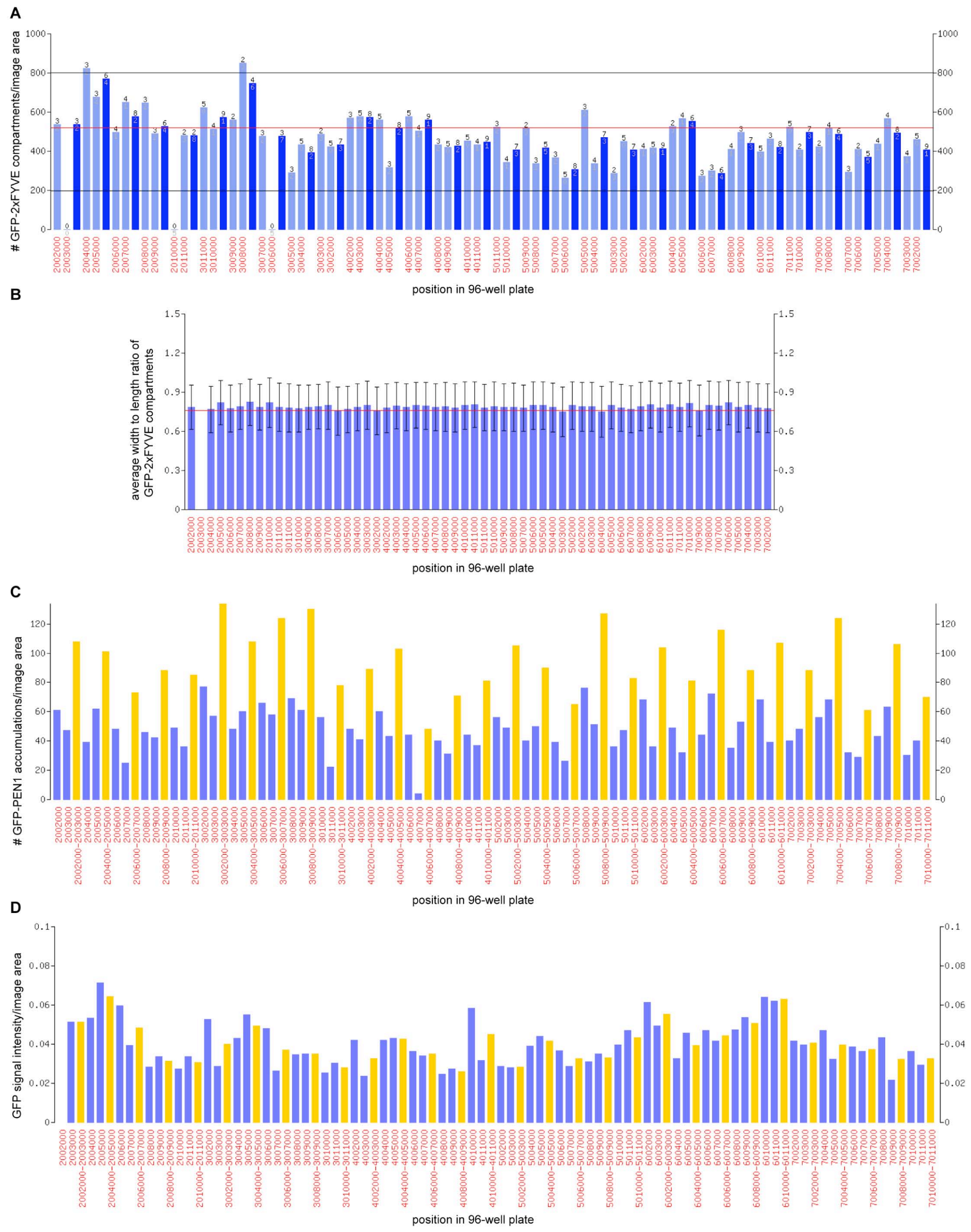
Supplemental Table S2. Overview of parameters recorded by the "plasma membrane microdomain script". Parameters displayed in PERL graphics and used in genetic screening are highlighted in grey.

	Output parameter	Description
1	Number of accumulation sites	Number of GFP-PEN1 focal accumulation sites
2	Number of GFP-PEN1 encased haustoria	
3	Number of accumulation sites per recognized cell area	Number of GFP-PEN1 focal accumulation sites per recognized epidermal cell area
4	Total integrated fluorescence signal of accumulation sites/ recognized cell area	Sum of GFP fluorescence signal of all accumulation sites identified in the recognized cell area
5	Total integrated fluorescence signal of accumulation sites/per recognized cell area, background subtracted	Sum of GFP fluorescence signal of all GFP-PEN1 accumulation sites identified in the recognized leaf area; background signal was subtracted from the GFP signal
6	Average fluorescence intensity of accumulation sites	Average GFP fluorescence intensity of all GFP-PEN1 accumulation sites
7	Average area of accumulation sites	Average area of all GFP-PEN1 accumulation sites
8	Total integrated accumulation site signal, over all accumulation sites	Sum of GFP fluorescence signal of all GFP-PEN1 accumulation sites
9	Total integrated fluorescence signal of accumulation sites, background subtracted, over all accumulation sites	Sum of GFP signal of all GFP-PEN1 focal accumulations; background signal was subtracted from the GFP signal
10	Average length of accumulation sites	Average length of all GFP-PEN1 focal accumulation sites
11	Average half width of accumulation sites	Average radius of all GFP-PEN1 focal accumulation sites
12	Average width to length ratio of accumulation sites	Average width to length ratio of all GFP-PEN1 focal accumulation sites
13	Average roundness of accumulation sites	Average roundness of all GFP-PEN1 focal accumulation sites
14	Average contrast of accumulation sites compared to the background signal	Average contrast between the GFP signal of the GFP-PEN1 focal accumulation site and the background signal
15	Average peak intensity of accumulation sites	Of all GFP-PEN1 focal accumulation sites the brightest pixel is taken into account, the average value of these is the average peak intensity
16	Integrated fluorescence signal of accumulation sites signal / recognized area, background subtracted, per accumulation site	Sum of GFP fluorescence signal of all GFP-PEN1 focal accumulation sites identified in the recognized epidermal cell area, divided by the number of GFP-PEN1 focal accumulation sites identified; background signal was subtracted from the GFP signal

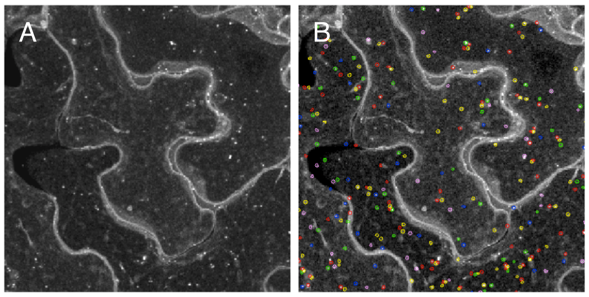
SUPPLEMENTAL FIGURES



Supplemental Figure S1. Z-sections are used to assemble pseudo-images. Confocal laser microscopy images were taken from cotyledon leaves of GFP-2xFYVE expressing Arabidopsis lines at 40x resolution. 21 images in 1µm distance were computationally merged to produce a pseudo-image. Image areas with low pixel contrast indicative of "out-of focus" were excluded. Recognition of leaf epidermal cells, and GFP-2xFYVE compartments were obtained by the "endomembrane script".



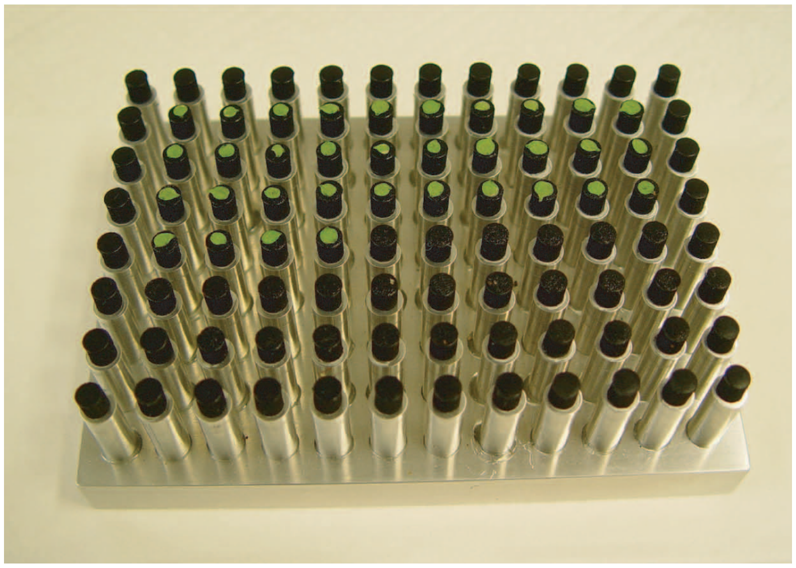
Supplemental Figure S2. Quantitative parameters of GFP-2xFYVE and GFP-PEN1 membrane compartments. (A) Average number of GFP-2xFYVE compartments per image area; **(B)** Average width to length ratio of compartments. Light blue bars: Average value per leaf. Dark blue bars: Average value per plant. Numbers above bars represent the number of valid images that were used to calculate the mean values. Numbers below bars refer to the number of images excluded from the analysis. Red lines indicate mean values of the reference line. **(C)** Average number of GFP-PEN1 accumulations per image area; **(D)** Average intensity of GFP signal per image area. Blue bars: Average value per leaf. Yellow bars: average value per plant.



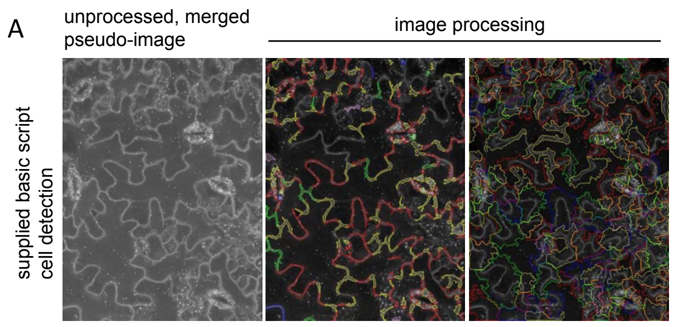
Supplemental Figure S3. Pattern recognition of GFP-2xFYVE compartments from images obtained by conventional confocal microscopy. GFP-2xFYVE cotyledon leaves were imaged at 63x magnification using a Leica SP5 confocal microscope (A) and images were analyzed for the detection of endosomal compartments by the "endomembrane script" (B). The flex format file of images obtained from the Opera microscope is a composite file (several images and associated meta-data) similar to a multi field tiff file. A maximum projection merges the images into a single image, selecting the brightest pixel at each position from the stacked images. The resulting image is a composite single layer version of the whole z-stack that combines the information from all images into one flattened image. A single tiff file of images from microscopes can be considered functionally equivalent in this sense to the projected image and both can serve as input for downstream analyses that use the flat image style of input. The main changes required to make use of tiff files from other microscopes files in a script are in the way that the script must handle the input files. For single images the projection stage can be omitted completely.

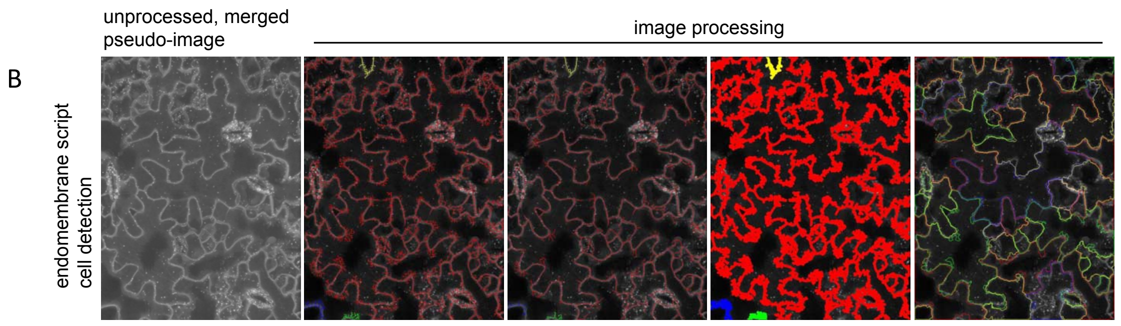
The following changes to the "endomembrane script" were made to process tiff file images:

```
// INPUT PARAMETERS – optional for each script; must be declared prior to image loading if image analysis needs input
input(StackNo, 0, "Stack No", "i", "Number of the stacks to analyze. If set to 0 all stacks are evaluated.")
input(BadStacksThreshold, 25, "Stack Rejection Threshold (%)", "i", "Percentage number of spots that must be within cells for Stack to be analysed")
input(ShowIllustrations, YES, "Show Illustrations", "y", "YES- Output illustrations are depicted. No- Output illustrations are not shown.")
input(WriteImages, YES, "Write Images", "y", "Write Images of the projected image and the detected spots to file")
input(debugImages,NO, "Debug Using Images", "y", "YES- all intermediary images are shown.")
// READS IN IMAGES
Singlewell(compact=no)
// COUNTS FIELDS AND CONTROLS IF ALL IMAGES ARE PRESENT
set(startStack = 1)
set(endStack = sourcedata.@rowcount)
set(imageFields = sourcedata.sourceImage) //actually image of each field
//CREATE GLOBAL WELL OBJECT PLACEHOLDERS
Create("objectlist" | all_LeafCells = objectlist)
Create("objectlist" | all_Spots = objectlist)
Create("objectlist" | all_ObjectsType2 = objectlist)
Create("objectlist" | all_Stomata = objectlist)
Create("objectlist" | all_Gaps = objectlist)
// LOOP OVER IMAGE FIELDS
Foreach(StartStack .. EndStack, "_StackCounter")
        set(IM_projected1 = imageFields[_StackCounter -1])
        set(IM_projected2 = IM_projected1)              // Image loaded and script proceeds.
```

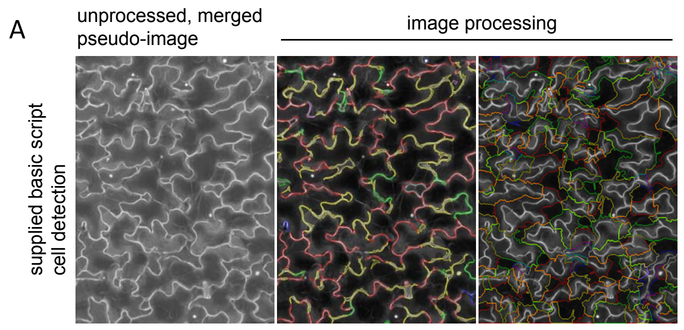


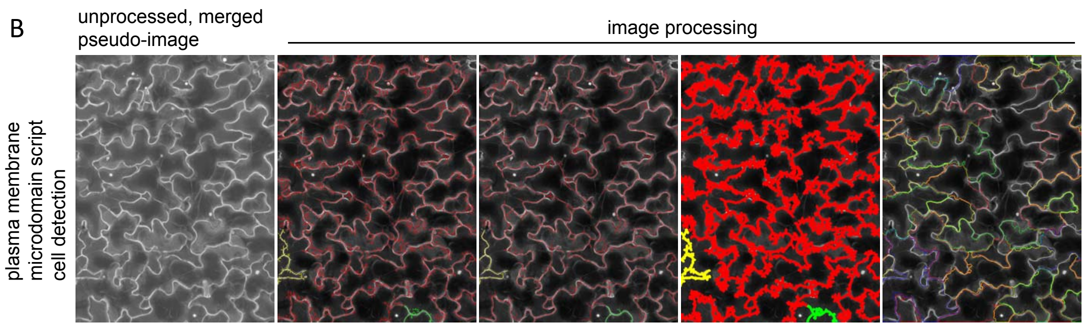
Supplemental Figure S4. Aluminium stamp with 96 metal pins and neoprene cushion tips. Some of the pin tips are loaded with green Arabidopsis cotyledon leaves. See Materials for detailed description.


A
unprocessed, merged pseudo-image
image processing
supplied basic script cell detection


B
unprocessed, merged pseudo-image
image processing
endomembrane script cell detection

Supplemental Figure S5. Comparative image processing of GFP-2xFYVE leaf samples. Automated cell detection of an unprocessed merged image was carried out using the ACAPELLA software (A) or the newly developed cell detection algorithm (B). GFP-2xFYVE plants were imaged at 40x magnification. Note that unambiguous cell identification is only possible in (B).


A
unprocessed, merged pseudo-image
image processing
supplied basic script cell detection


B
unprocessed, merged pseudo-image
image processing
plasma membrane microdomain script cell detection

Supplemental Figure S6. Comparative image processing of GFP-PEN1 leaf samples. Automated cell detection of an unprocessed merged image was carried out using the ACAPELLA software (A) or the newly developed cell detection algorithm (B). GFP-PEN1 plants were imaged at 20x magnification. Note that unambiguous cell identification is only possible in (B).

SUPPLEMENTAL DATA

Supplemental Data S1. Endomembrane script. Acapella system script for the detection of endomembrane spot-like features in collections of multi-plane confocal microscopy images.

```
# name of script: endomembrane script
# to execute this script rename .text to .script

//**************************************************************************

//  File Name:    MPI_leaves_h_seba.script

//  Purpose: Detects spot-like features on leaf images, images from MPIZ-Köln

//

//              EVOTEC  TECHNOLOGIES

//              Olavi Ollikainen

//  e-mail:   Olavi.Ollikainen@evotec-technologies.com

//phone :    00 372 52 75 262

//  With corrections from Kurt Stüber MPI Köln

//  phone:    +49 (0) 221 5062 120

//  e-mail     stueber@mpiz-koeln.mpg.de

//  With further modifications from Sebastian Schaaf MPIZ Köln

//  phone   +49 (0) 221 5062 323

// e-mail      schaaf@mpiz-koeln.mpg.de

//**************************************************************************

//

//          Output parameters:

//                          * Number of valid Cells in Stack
//                    * Number of valid Spots in Stack
//                    * Number of Spots in and out of Cells in Stack
//                    * Percents of inner Spots in Stack
//                    * Average Area of Cells in Stack
//                    * Average Area of Cells - Standard Deviation in Stack
//                    * Percents of found Cell Area in Stack
//                    * Average Number of Spots in Cells in Stack
//                     * Average Number of Spots in Cells - Standard Deviation in Stack
//                    * Average Number of Spots per recognized Area in Stack
//
//                    * Number of Leaf Cells in whole Well
//                    * Average Cell Area in whole Well
//                    * Average Cell Area in whole Well - Standard Deviation
//                    * Number of Spots in whole Well
//                    * Average Number of Spots per Cell in whole Well
//                    * Average Number of Spots per Cell in whole Well - Standard Deviation
//                    * Total Cell Area in Well
//                    * Percentage of total Cell Area in Well
//
//                    * Number Of Stomata
//
//                    * Average Intensity of Spots
//                    * Average Area of Spots
//                    * Average Length of Spots
//                    * Average Half Width of Spots
//                    * Average Width to Length Ratio of Spots
//                          * Average Width to Length Ratio of Spots - Standard Deviation
//                          * Average Roundness of Spots
//                          * Average Roundness of Spots - Standard Deviation
//                          * Average Contrast of Spots
//                          * Average Contrast of Spots - Standard Deviation
//                          * Average Peak Intensity of Spots
//
//                          * Total number of Stacks analyzed in Well
//                          * Number of valid Stacks in Well
//                          * Percentage of valid Stacks in Well
//
//***************************************************************************************************************************

/// Spot detection procedure. Finds spots on SpotImage. Spot candidates are detected as local intensity maximums. Thereupon the spots are selected by contrast and intensity parameters.
/// formerly called Spot_Detection_C
proc Spot_Detection_MPIZ(
  image SpotImage in " image with intensity information. Spots are detected by this image.",
  string SearchRegion="SearchRegion" in "Name of the attribute in input list WholeCells, which specifies the regions where spots are searched. In case of the empty string \"\" spots are searched over the whole SpotImage and the input object list is ignored.",
  bool ShowIllustrations=YES in "YES- Output illustrations are depicted. No- Output illustrations are not shown.",
  bool ShowSearchRegionBorder=NO in "YES- Output illustration with SearchRegion borders is depicted. No- Output illustration with SearchRegion borders is not shown.",
  bool ShowOutputParameters=YES in "YES- Output parameters are reported. No- Output Parameters are not reported.",

  objectlist WholeCells=none inout "Optional input object list, which defines the objects (e.g. ~cells) where the spots are searched. Output list contains numerical and geometrical spot attributes. Input list should involve the stencil-type attribute specified by the input SearchRegion. If the object list is not provided spots are searched over the whole image.",
  objectlist SpotCandidates out "Output object list of spot candidates with calculated numerical and geometrical attributes.",
  objectlist Spots out "Output object list of classified spots with calculated numerical and geometrical attributes.",
  double NumberOfSpotCandidates out "Number of spot candidates.",

  double NumberOfSpots out "Number of detected spots.",

  double SpotsPerObject out "Number of spots per object (i.e. number of spots per ~cell).",

  double SpotsPerArea out "Number of spots per SearchRegion area (i.e. number of spots per visible ~cell area).",

  double IntegratedSpotSignalPerCellularSignal out "Integrated spot signal over all spots normalized by integrated ~cellular signal (total signal over all SearchRegion area).",

  double IntegratedSpotSignalPerCellularSignal_BackgroundSubtracted out "Integrated spot signal over all spots background subtracted and normalized by integrated ~cellular signal (total signal over all SearchRegion area).",



  double IntegratedSpotSignalPerArea out "Integrated spot signal per SearchRegion area (per visible ~cell area).",

  double IntegratedSpotSignalPerArea_BackgroundSubtracted out "Integrated spot signal BackgroundSubtracted per SearchRegion area (per visible ~cell area)."



) spot detection, object recognition "Spot detection procedure. Finds spots on SpotImage. Spot candidates are detected as local intensity maximums. Thereupon the spots are selected by contrast and intensity parameters. Proper spot detection input parameters can be found with the template script spot_detection_parameter_scanner.script (spot_detection_parameter_scanner_Acapella10.script for Acaplella 1.0). Opera multiple field spot images can be evaluated based on the template script Opera_multifields_spots.script. See more in Opera spot detection manual."

{

  input(SpotMinimumDistance,3.0 , "SpotMinimumDistance","d", "Minimum allowed distance between two spot centers. Unit image pixel. Typical range 2.0 .. 5.0. Adjust by Illustration SpotSelection and Spots.")

  input(SpotPeakRadius, 1.0, "SpotPeakRadius","d", "Radius of the disk, where the spot peak intensity is calculated. Default value 0 means that peak intensity corresponds to the intensity of the maximum point, value 1 means that peak intensity is found as average intensity over the region with radius 1 pixel around the maximum point, i.e. over the region with area 5 pixels. Typically 0 or 1.")

  input(SpotReferenceRadius, 8.0,  "SpotReferenceRadius", "d", "Radius of Reference Region around the intensity maximum, i.e. around spot center. Typical range  2.0 .. 5.0.")

  input(SpotMinimumContrast,0.4, "SpotMinimumContrast", "d","Minimum allowed contrast between spot peak intensity and the reference intensity. The main spot selection parameter. Range from 0..1. Adjust parameter by illustrations  and Spots. If the parameter value is lowered the number of classified spots increases and vice versa.")

  input(SpotMinimumToCellIntensity, 1.0, "SpotMinimumToCellIntensity","d","Minimum allowed intensity ratio between Spot Peak Intensnity and the Average Intensity of the cell/object to which the spot is belongs. Range from 0..oo. Has smaller influence on the outputs than the main selection parameter SpotMinimumContrast. Adjust parameter by illustrations SpotSelection and Spots. If the parameter value is lowered the number of classified spots increases and vice versa.")



  if( SearchRegion!="" and defined("WholeCells"))

    set(Objects_in=WholeCells)

    if(!defined("WholeCells." & SearchRegion))
```

```
      error("Procedure Spots_Detection_C() input attribute SearchRegion " & SearchRegion & " does not exist. The input SearchRegion should correspond to the stencyl-type attribute in the input object list or to be an empty string \"\". The empty string means that spots are searched over the whole image and the input object list is ignored.")
    else()
      eval("set(SearchMask=WholeCells." & SearchRegion & ")")
      if (SearchMask.class!="intervalvector")
        error("Procedure Spots_Detection_C() input attribute SearchRegion " & SearchRegion & " is not a stencil. The input SearchRegion should correspond to the stencyl-type attribute in the input object list or to be an empty string \"\". The empty string means that spots are searched over the whole image and the input object list is ignored.")
      end()
    end()
    set(SearchObjects=WholeCells)
  end()


  if (ShowIllustrations==Yes)
    spot_illustrations_1()
  end()


  Spot_Detection_C_inner() // Spot detection


  if(ShowIllustrations)
    spot_illustrations_2()
    set(printf_text="Spot detection C. Number of Spot Candidates: " & SpotCandidates.count & "; Number Classified Spots: " & spots.count & "; Discarded by Contrast: " & NumberDiscradedByContrast & "; Discarded by SpotToCellIntensity: " & NumberDiscradedBySpotToCellIntensity & "\n")
    printf(printf_text)
  end()
  set(NumberOfSpotCandidates=SpotCandidates.count)
  set(NumberOfSpots=Spots.count)
  if(ShowOutputParameters)
```

```
    create_spot_outputs()

  end()

  set(printf_text="Spot detection C. Number of Spot Candidates: " & NumberOfSpotCandidates & "; Number Classified Spots: " & NumberOfSpots & "; Spots per cell/object: " & SpotsPerObject & "; Spot per area: "
& SpotsPerArea & "; Integrated Spot Signal Per Cellular Signal: " & IntegratedSpotSignalPerCellularSignal & "\n")

  printf(printf_text)

}




//////////////////////////////////////////////////////////////// PROCEDURES

proc CalcNodesLarge( string SkeletonName="skeleton" in "Stencil in input list,  Nodes are found for this stencil",

objectlist objects inout "Input-output object list") "Adds to object list a stencil-type attribute LargeNodes"

{

  if(!defined("objects." & SkeletonName))

    error("Error. Input stencil " & SkeletonName & " is not defined. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")

  else()

    eval("set(IfIntervalVector=Objects." & SkeletonName & ".class==\"intervalvector\")")

    if(!IfIntervalVector)

      error("Error. The attribute  " & SkeletonName & " given by the input SkeletonName is not a stencil. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")

    end()

  end()

  set(ob_in=objects)

  eval("setattr(stencil_temp,objects." & SkeletonName & ")")


  set(IM_skeleton=objects.stencil_temp.mask.image)


  blank(3,3,1)
```

```
set(kernel0=image)

set(kernel0[0,0]=0, kernel0[2,2]=0,  kernel0[0,2]=0, kernel0[2,0]=0)


set(kernel1=kernel0)

set(kernel1[1,0]=0)

convolution(image=IM_skeleton,mask=objects.stencil_temp.mask, faster=yes, convolutionkernel=kernel1.vector)

set(image.factor=1)

mask(4,image=image)

set(M_nodeslarge1=mask.image)


set(kernel2=kernel0)

set(kernel2[2,1]=0)

convolution(image=IM_skeleton,mask=objects.stencil_temp.mask, faster=yes, convolutionkernel=kernel2.vector)

set(image.factor=1)

mask(4,image=image)

set(M_nodeslarge2=mask.vector)


set(kernel3=kernel0)

set(kernel3[1,2]=0)

convolution(image=IM_skeleton,mask=objects.stencil_temp.mask, faster=yes, convolutionkernel=kernel3.vector)

set(image.factor=1)

mask(4,image=image)

set(M_nodeslarge3=mask.vector)


set(kernel4=kernel0)
```

```
  set(kernel4[0,1]=0)
  convolution(image=IM_skeleton,mask=objects.stencil_temp.mask, faster=yes, convolutionkernel=kernel4.vector)
  set(image.factor=1)
  mask(4,image=image)
  set(M_nodeslarge4=mask.vector)

  carrypixels(image=M_nodeslarge1,mask=M_nodeslarge2,data=1)
  carrypixels(image=image,mask=M_nodeslarge3,data=1)
  carrypixels(image=image,mask=M_nodeslarge4,data=1)

  calcnodes(stencil_temp)
  carrypixels(image=image,mask=objects.stencil_temp_nodes,data=1)
  and(image=objects.stencil_temp.image, mask=image)
  eval("setattr(" & SkeletonName & "_LargeNodes,image.vector, objects=ob_in)")
}


///////////////////////////////////////////////////////////////////////////////
proc CalcSkeletonBranches(
string SkeletonName="skeleton" in "Stencil-type attribute on which the  skeleton branches are found.",
string DeadEndStencil="deadend" in "Stencil-type attribute, which corresponds to DeadEnd tips",
objectlist objects inout "Input-output object list with the stencil-type attribute on which the branches are found. Input list must include also a stencil, which coresponds to dead end tips.",
objectlist Branches out "Output object list of Branch lines on the skeleton between nodes.  "
) "Finds skeleton branches between nodes. Outputs object list of skeleton branches"
{
  set(ob_in=objects)
```

```
if(!defined("objects." & SkeletonName))
  error("Error. Input stencil " & SkeletonName & " is not defined. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
else()
  eval("set(IfIntervalVector=Objects." & SkeletonName & ".class==\"intervalvector\")")
  if(!IfIntervalVector)
    error("Error. The attribute  " & SkeletonName & " given by the input SkeletonName is not a stencil. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
  end()
end()

if(!defined("objects." & DeadEndStencil))
  error("Error. Input stencil " & DeadEndStencil & " is not defined. Input DeadEndStencil should correspond to the existing stencil-type attribute in the input object list.")
else()
  eval("set(IfIntervalVector=Objects." & DeadEndStencil & ".class==\"intervalvector\")")
  if(!IfIntervalVector)
    error("Error. The attribute  " & DeadEndStencil & " given by the input DeadEndStencil is not a stencil. Input DeadEndStencil should correspond to the existing stencil-type attribute in the input object list.")
  end()
end()

eval("setattr(skeleton,ob_in." & SkeletonName & ")")
eval("setattr(DeadEnd,ob_in." & DeadEndStencil & ")")

calcnodeslarge(SkeletonName="skeleton")

convolutionmask("disk",1)
convolution(image=objects.skeleton_largenodes.mask.image, mask=objects.skeleton.mask,faster=yes)
```

```
set(image.factor=1)

mask(1)

set(nodes_surrounding=mask)

set(ob2=objects)

carrypixels(image=objects.skeleton.mask.image,mask=nodes_surrounding.vector,data=0)

set(M_connections=image)


mask2stencil(M_connections, Neighbourhood=8)

stencil2objects()


and(image=objects.body.image, mask=ob2.deadend.mask.image)

setattr(deadend, image.vector)

calcarea()

calcarea(deadend)

carrypixels(image=ob2.skeleton.image, mask=ob2.skeleton_largenodes, data=0)

calczone(1, stencil=image.vector)

zonemask(-1,oo)

renameattr(Branches=zonemask)

calcarea(Branches)

Set(Branches=objects)

and(image=ob2.skeleton.image,mask=Branches.body.mask.image)

Setattr(Branches,image.vector,objects=ob_in)

Setattr(Nodes,ob2.skeleton_LargeNodes)

Setattr(DeadEnd,ob2.skeleton_DeadEnd)

Setattr(DeadEnd_type1,ob2.skeleton_DeadEnd_type1)

Setattr(DeadEnd_type2,ob2.skeleton_DeadEnd_type2)
```

```
}



proc CalcDeadEndTypes(
string SkeletonName="skeleton" in "Stencil-type attribute, which corresponds to skeleton",
objectlist objects inout "Input-output object list"
) "Finds the deadend tips in the skeleton"
{
  set(ob_in=objects)
  if(!defined("objects." & SkeletonName))
    error("Error. Input stencil " & SkeletonName & " is not defined. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
  else()
    eval("set(IfIntervalVector=Objects." & SkeletonName & ".class==\"intervalvector\")")
    if(!IfIntervalVector)
      error("Error. The attribute  " & SkeletonName & " given by the input SkeletonName is not a stencil. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
    end()
  end()
  eval("setattr(skeleton,ob_in." & SkeletonName & ")")

  set(IM_skeleton=objects.skeleton.mask.image)
  not(image=IM_skeleton)
  set(IM_skeleton_not=image)
  //set(IM_skeleton_vec=objects.skeleton_skeleton.mask)
  //calcnodes(skeleton)
  CalcNodesLarge(SkeletonName="skeleton")
```

```
convolutionmask("ribbon",1, 1)

set(ConvolutionKernel_4n=ConvolutionKernel)

convolution(image=objects.skeleton_largenodes.mask.image,mask=objects.skeleton, faster=yes)

set(image.factor=1)

mask(1,image=image)

set(M_nodes_4n=mask.vector)


convolutionmask("ribbon",1.5, 1)

set(ConvolutionKernel_8n=ConvolutionKernel)


convolution(image=IM_skeleton,mask=objects.skeleton, faster=yes)

set(image.factor=1)

set(IM_convolution_8n=image)

mask(1,image=image)


set(M1=mask.image)

mask(2,image=image)

set(M2=mask.image)


carrypixels(image=m1,mask=M2.vector,data=0)

set(M_deadend0=image.vector)


convolution(image=objects.skeleton_largenodes.mask.image,mask=M_deadend0, faster=yes)

set(image.factor=1)

mask(1,image=image)
```

```
set(M_deadend_type1=image.vector)


//carrypixels(image=M_deadend0.image,mask=M_deadend_type1.vector,data=0)

//set(M_deadend0=image.vector)

////////////////////////////////////////////////////////////////////

blank(3,3,1)

set(IM1=image)


carrypixels(image=M_nodes_4n.image, mask=M_deadend0,data=0)

set(M_nodes_4n_mod=image.vector)


set(kernel1=IM1)

set(kernel1[2,0]=0,kernel1[2,1]=0, kernel1[2,2]=0, kernel1[1,1]=0)

convolution(image=IM_skeleton_not,mask=M_nodes_4n_mod, faster=yes, convolutionkernel=kernel1.vector)

set(image.factor=1)

mask(5,image=image)

set(M_deadend1=mask.image)


set(kernel2=IM1)

set(kernel2[0,0]=0,kernel2[0,1]=0, kernel2[0,2]=0, kernel2[1,1]=0)

convolution(image=IM_skeleton_not,mask=M_nodes_4n_mod, faster=yes, convolutionkernel=kernel2.vector)

set(image.factor=1)

mask(5,image=image)

set(M_deadend2=mask.image)
set(kernel3=IM1)

set(kernel3[0,0]=0,kernel3[1,0]=0, kernel3[2,0]=0, kernel3[1,1]=0)

convolution(image=IM_skeleton_not,mask=M_nodes_4n_mod, faster=yes, convolutionkernel=kernel3.vector)

set(image.factor=1)

mask(5,image=image)

set(M_deadend3=mask.image)


set(kernel4=IM1)

set(kernel4[0,2]=0,kernel4[1,2]=0, kernel4[2,2]=0, kernel4[1,1]=0)

convolution(image=IM_skeleton_not,mask=M_nodes_4n_mod, faster=yes, convolutionkernel=kernel4.vector)

set(image.factor=1)

mask(5,image=image)

set(M_deadend4=mask.image)


or(image=M_deadend1,mask=M_deadend2)

or(image=image,mask=M_deadend3)

or(image=image ,mask=M_deadend4)

or(image=image,mask=M_deadend_type1.image)

set(M_deadend_type1=image)

carrypixels(image=M_deadend0.image, mask=M_deadend1.vector, data=0)

set(M_deadend_type2=image)

or(image=M_deadend_type1.image, mask=M_deadend_type2.image)

set(M_deadend=image)

set(ob2=objects)


eval("setattr(" & SkeletonName & "_LargeNodes,ob2.skeleton_LargeNodes, objects=ob_in)")
```

```
  and(image=ob2.skeleton.image, mask=M_deadend.image)
  eval("setattr(" & SkeletonName & "_DeadEnd,image.vector)")

  and(image=ob2.skeleton.image, mask=M_deadend_type1.image)
  eval("setattr(" & SkeletonName & "_DeadEnd_type1,image.vector)")

  and(image=ob2.skeleton.image, mask=M_deadend_type2.image)
  eval("setattr(" & SkeletonName & "_DeadEnd_type2,image.vector)")
}



proc RemoveSkeletonLayer(
string SkeletonName="skeleton" in "Stencil-type attribute, which corresponds to skeleton",
int MinimumArea=10 in "DeadEnd skeleton branches with area less than the limit are discarded",
objectlist objects inout "Input-output object list"
) "Discards from skeleton the DeadEnd branches with area less than the limit MinimumArea"
{
  set(ob_in=objects)
  if(!defined("objects." & SkeletonName))
    error("Error. Input stencil " & SkeletonName & " is not defined. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
  else()
    eval("set(IfIntervalVector=Objects." & SkeletonName & ".class==\"intervalvector\")")
    if(!IfIntervalVector)
      error("Error. The attribute  " & SkeletonName & " given by the input SkeletonName is not a stencil. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
    end()
  end()
  eval("setattr(skeleton,ob_in." & SkeletonName & ")")

  CalcDeadEndTypes(SkeletonName="skeleton")
  carrypixels(image=objects.skeleton.image,mask=objects.skeleton_DeadEnd_type1.mask,data=0)
  setattr(skeleton_mod,image.vector)

  /////////////////////////////////////////////////////////////////////////////////////////
  set(ob2=objects)
  CalcSkeletonBranches(SkeletonName="skeleton_mod", deadendstencil="skeleton_DeadEnd_type2")
  ObjectFilter(MinimumArea>Branches_area and deadend_area>0, objects=Branches)
  carrypixels(image=ob2.skeleton_mod.image,mask=objects.Branches.mask,data=0)
  set(BranchesRemoved=objects)
  setattr(skeleton_LayerRemoved,image.vector, objects=ob2)
  and(image=ob_in.body.image, mask=BranchesRemoved.Branches.mask.image)
  setattr(skeleton_RemovedBranches,image.vector)
  set(ob3=objects)

  eval("setattr(" & SkeletonName & "_LargeNodes,ob3.skeleton_LargeNodes, objects=ob_in)")
  eval("setattr(" & SkeletonName & "_DeadEnd, ob3.skeleton_DeadEnd)")
  eval("setattr(" & SkeletonName & "_DeadEnd_type1, ob3.skeleton_DeadEnd_type1)")
  eval("setattr(" & SkeletonName & "_DeadEnd_type2, ob3.skeleton_DeadEnd_type2)")
  eval("setattr(" & SkeletonName & "_LayerRemoved,  ob3.skeleton_LayerRemoved)")
  eval("setattr(" & SkeletonName & "_RemovedBranches,  ob3.skeleton_RemovedBranches)")
}
```

```
proc SkeletonBranchesNodes(
string SkeletonName="skeleton" in "Stencil-type attribute, which corresponds to skeleton",
objectlist objects inout "Input-output object list",
objectlist Branches out "Output object list of skeleton branches",
objectlist Nodes out "Output object list of nodes"
//int MinimumArea=20 in "DeadEnd skeleton branches with area less than the limit are discarded"
) "Finds Brances and Nodes of skeleton. Outputs object lists of the found Branches and Nodes. In addition adds to the input list attributes branches and nodes."
{
  set(ob_in=objects)
  if(!defined("objects." & SkeletonName))
    error("Error. Input stencil " & SkeletonName & " is not defined. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
  else()
    eval("set(IfIntervalVector=Objects." & SkeletonName & ".class==\"intervalvector\")")
    if(!IfIntervalVector)
      error("Error. The attribute  " & SkeletonName & " given by the input SkeletonName is not a stencil. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
    end()
  end()
  eval("setattr(skeleton,ob_in." & SkeletonName & ")")

  CalcDeadEndTypes(SkeletonName="skeleton")
  carrypixels(image=objects.skeleton.image,mask=objects.skeleton_DeadEnd_type1.mask,data=0)
  setattr(skeleton_mod,image.vector)
  //////////////////////////////////////////////////////////////////////////////////////////////////////
  set(ob2=objects)
  CalcSkeletonBranches(SkeletonName="skeleton_mod", deadendstencil="skeleton_DeadEnd_type2")
  set(ob2=objects)
  mask2stencil(objects.nodes)
  stencil2objects()
  set(Nodes=objects)
  eval("setattr(" & SkeletonName & "_Nodes,ob2.Nodes, objects=ob_in)")
  eval("setattr(" & SkeletonName & "_Branches,ob2.Branches)")
  eval("setattr(" & SkeletonName & "_DeadEnd,ob2.DeadEnd)")
  eval("setattr(" & SkeletonName & "_DeadEnd_type1,ob2.DeadEnd_type1)")
  eval("setattr(" & SkeletonName & "_DeadEnd_type2,ob2.DeadEnd_type2)")
}


////////////////////// Fills small holes
proc FillSmallHoles(
int minimumholearea=20 in "Minimum area for holes. Holes with area less than the limit are filled",
objectlist objects inout "Input-output object list. Attributes in the input list are lost."
) "Fills small holes in objects. Holes with area less than the limit MinimumHoleArea are filled."
{
  set(ob_in=objects)
  fillobjects()
  set(ob01=objects)
  carrypixels(image=ob01.body.image,mask=ob_in.body.mask,data=0)
  stencil2mask(image.vector)
  mask2stencil()
  stencil2objects()
  calcarea()
  objectfilter(minimumholearea<=area)

  carrypixels(image=ob01.body.image,mask=objects.body.mask,data=0)
  stencil2objects(image.vector)
  xor(image=objects.body.image,mask=ob_in.body.mask.image)
  setattr(filled, image.vector)
}


proc ImageSubRegions(
int ImageWidth in,
int ImageHeight in,
int NumberOfLinearDivision in,
objectList objects out
)
{
  set(w0=int((1.0*ImageWidth)/(2.0*NumberOfLinearDivision)))
  set(h0=int((1.0*ImageHeight)/(2.0*NumberOfLinearDivision)))
  set(w1=2*w0, h1=2*h0)
  blank(ImageWidth, ImageHeight)
  set(image.type="mask")
  foreach(0..NumberOfLinearDivision-1  )
    foreach(0..NumberOfLinearDivision-1,"j")
      set(image[w0+i*w1, h0+j*h1]=1)
    end()
  end()
  mask2stencil(image)
  stencil2objects()
  set(eros_dist=w0+h0)
  calcerosion(-eros_dist)
}

proc DataCubeProjectionCorrection(
int FirstZPlane in,
int LastZPlane in,
datacube datacube inout,
image IM_planeNumber out,
image PlaneNUmberImage out
) [hidden]
{
  ////// Projects maximum intensity for each x-y position from datacube to image
  Maximums3D(0,image=datacube)
  StencilFrom3DTo2D(stencil=maximums,datacube=datacube) //Projects the found maximums from 3-dim to plane
  set(IM_plane=PlaneNumberImage)
  //delete(datacube)
  convolutionmask("ribbon",7.9,7)
```

```
convolution(image=valueimage)

minus(valueimage,image,neg_method="zero", result_type="unsigned,short")

set(r4=result)

mask(1, image=result,userealvalues=no)

set(M_bright2=mask)


mask(result.mean, image=r4)


and(image=M_bright2.image, mask=mask.image)

set(M_bright3=image)


mask2stencil(M_bright3)

if(stencil.itemcount>31999)

  ReduceStencilObjectsByAreaTo31999_MPIZ()

end()

stencil2objects()


CalcSkeletonByIntensity(image=r4)

CalcSkeletonByIntensity(skeleton, image=r4, IntensityEvalParam=-10)

CalcSkeletonByIntensity(skeleton_skeleton, image=r4, IntensityEvaluationMode=2, IntensityEvalParam=-4)

CalcSkeletonByIntensity(skeleton_skeleton_skeleton, image=r4, IntensityEvaluationMode=2, IntensityEvalParam=-20)

CalcSkeleton(skeleton_skeleton_skeleton_skeleton)

RenameAttr(Skeleton=skeleton_skeleton_skeleton_skeleton)

CalcArea(Skeleton)

set(Skeleton=objects)
```

```
ObjectFilter(Skeleton_area>10)

SkeletonBranchesNodes(skeletonName="skeleton")


Set(OL4=objects)

set(objects=branches)

ObjectFilter(area>10)

Set(OL7=objects)


ImageSubRegions(r4.width, r4.height,4)


And(Image=objects.eroded.image, mask=OL7.body.mask.image)

clearborders(image,9)

SetAttr(skeleton, stencil.vector)

CalcStat("quantile",0.7, stencil=skeleton, image=r4)

ThreshMask(stencil=skeleton,threshold=quantile,image=r4)

CalcStat("median",stencil=threshmask, image=PlaneNumberImage)

//CalcStat("Median",stencil=skeleton, image=PlaneNumberImage)

CarryObjects(image=objects.eroded.image, stencil=objects.eroded, data=objects.median)

//

//enlarge(1,image=image)   // AK 3.12.2007

redimension(image.width + 2, image.height + 2, 1, 1) // AK 3.12.2007

stencil2objects(image)

calcerosion(-2)


set(temp=objects.eroded.image)

crop(1,1,temp.width-1, temp.height-1,image=temp)
```

```
mean(31,image=image)

set(IM_PlaneNumber=Image)


delete(objects)

set(objects=OL7)

calcIntensity(body,image=r4)

calcstat("max",stencil=body,image=r4)

calcstat("Quantile",0.5,stencil=body,image=r4)


Threshmask(threshold=quantile,stencil=body,image=r4)

Calcintensity(Threshmask,image=r4)

//Calcstat("mean",stencil=threshmask,attrname="planeNumberMean", image=PlaneNumberImage)

Calcstat("median",stencil=body,attrname="planeNumberMean", image=PlaneNumberImage)

Calcerosion(-3)


CarryObjects(image=objects.eroded.image, stencil=objects.eroded, data=objects.max)

set(IM_intensity=Image)

maximums(20, mask=M_bright3.vector,image=r4)

set(max1=maximums)

maximums(70, mask=M_bright3.vector,image=r4)

set(max2=maximums)

or(image=max1.mask.image,mask=max2.mask.image)

and(image=objects.eroded.image,mask=image)

setattr(max1,image.vector)

calcarea(max1)
```

```
objectfilter(max1_area>0)

Calcerosion(-100,eroded,numberofsteps=20)

CarryObjects(image=objects.eroded.image, stencil=objects.eroded_eroded, data=objects.planeNumberMean)

set(IM_intensity2=Image)

//CarryObjects(image=objects.eroded.image, stencil=objects.eroded_eroded, data=objects.max)

//set(IM_max=Image)

//CarryObjects(image=objects.eroded_eroded.image, stencil=objects.eroded_eroded, data=objects.planeNumberMean)

//set(IM_planeNumberMean=Image)

set(OL5=objects)



and(image=objects.body.image,mask=max2.mask.image)

setattr(max2,image.vector)

calcarea(max2)

objectfilter(max2_area>0)

calcerosion(-10,max2,restrictivestencil=M_bright3,numberofsteps=2)

and(image=objects.max2_eroded.image,mask=skeleton.skeleton.mask.image)

setattr(skeleton1,image.vector)

Calcintensity(skeleton1,image=PlaneNumberImage)

CalcAttr(planenumber,round(skeleton1_intensity))

//CalcAttr(planenumber,iif(planenumber>10, planenumber-10, 1))

calcerosion(-100,max2_eroded,restrictivestencil=M_bright3,numberofsteps=20)

calcerosion(-200,max2_eroded_eroded,numberofsteps=20)

//CarryObjects(image=objects.max2_eroded_eroded_eroded.image, stencil=objects.max2_eroded_eroded_eroded, data=objects.planenumber)

//set(IM_planeNumberMean=Image)
```

```
  set(i=FirstZplane, i2=0)

  foreach(FirstZplane..LastZplane)

    set(threshold2=i2+1)

    minus(threshold2,IM_planeNumber,neg_method="abs",result_type="unsigned,short")

    mask(threshold=8,image=result)

    carrypixels(image=DataCube[i2],mask=mask.vector,data=0)

    set(DataCube[i2]=image)

    set(i2=i2+1)

  end()


}


//proc ReduceStencilObjectsTo31999_tech(

//stencil stencil inout "Input-output stencil, which object number will be reduced to fit the limit 31999. Only the first 31999 objects are taken into account and the others are discarded."

//) [hidden]

//{

// set(temp=stencil.vector.mask.image)

// blank(temp.width, temp.height)

// convelems(image, "integer", 2,"unsigned")

// append(avector,1..31999)

// create("vector","unsigned,int", stencil.itemcount-31999,0)

// append(avector,vector)

// carryobjects(image=result,stencil=stencil,data=avector)
// set(image.type="stencil")

// set(stencil=image.vector)

//}



proc ReduceStencilObjectsByAreaTo31999_MPIZ(

stencil stencil inout "Input-output stencil, which object number will be reduced to fit the limit 31999. Only the first 31999 objects are taken into account and the others are discarded."

) [hidden]


{

  set(alimit=32000)

  set(alimit2=alimit-1)

  eval("WarningFilter(disable=\"[Mask2Stencil]\")")

  Removes_OnePixel_Objects(stencil.vector.MASK)

  mask2stencil()

  if(stencil.itemcount<alimit)

    return()

  end()


  set(mask_in=mask)

  set(temp=stencil.vector.mask.image)

  blank(temp.width, temp.height)

  convelems(image, "integer", 2,"unsigned")


  append(bvector,1..alimit2)

  set(avector=bvector)
  set(stencil_in=stencil)
  while(stencil.itemcount>alimit2)
    set(stencil2=stencil)
    create("vector","unsigned,int", stencil.itemcount-alimit2,0)
    append(avector,vector)
    carryobjects(image=result,stencil=stencil,data=avector)
    set(image.type="stencil")
    set(stencil=image.vector)
    stencil2objects()
    calcarea()
    if(defined("v_area"))
      append(v_area,objects.area)
    else()
      set(v_area=objects.area)
    end()
    push(v_objects,objects)
    carrypixels(image=stencil2.image,mask=objects.body.mask, data=0)
    mask2stencil(image.vector.mask)

  end()
  stencil2objects()
  calcarea()

  append(v_area,objects.area)
  sort(v_area,yes)
```

```
  set(minarea=result[alimit2]+1)
  objectfilter(area<minarea)
  carrypixels(image=mask_in,mask=objects.body,data=0)
  foreach(v_objects)
    set(objects=i)
    objectfilter(area<minarea)
    carrypixels(image=image,mask=objects.body,data=0)
  end()
  mask2stencil(image)
  if(stencil.itemcount>31999)
    ReduceStencilObjectsTo31999_tech()
  end()
}

proc CreateDataCubeWithControl(
table sourcedata inout,
int FirstZPlane in,
int LastZPlane in,
int MinStdDev in,
DataCube DataCube out,
int newFirstZplane out,
int newLastZplane out
)
{
  //set( logfilename = "D:/dmeyer/test/filenames.txt" )
  set( Zplane = FirstZplane )
```

```
//write( "flex\t" & imagefilename1, logfilename, "ascii", append=true )

//write( "\tFirstZplane\t" & FirstZplane, logfilename, "ascii", append=true )

//write( FirstZplane, logfilename, "ascii", append=true )

//write( "\tLastZplane\t" & LastZplane, logfilename, "ascii", append=true )

//write( LastZplane, logfilename, "ascii", append=true )



////// Creates a datacube

set(CubeDepth=LastZplane-FirstZplane+1)

create("datacube",sourcedata.SourceImage[0].width,sourcedata.sourceImage[0].height,CubeDepth, "unsigned short")

set(FirstZplane_in=FirstZplane, LastZplane_in=LastZplane)

set(i=FirstZplane, i2=0)

// find new FirstZplane and LastZplane discarding featureless images:

set( is_empty="true")

set( newFirstZplane = FirstZplane)

foreach(FirstZplane..LastZplane)

  if( sourcedata.sourceImage[i-1].stddev < MinStdDev )

    if( is_empty == "true")

      set( newFirstZplane = i )

    end()

  else()

    set( is_empty= "false" )

  end()

end()

set( is_empty="true")
set( i=LastZplane)
set( newLastZplane = LastZplane)
foreach(LastZplane..FirstZplane)
  if( sourcedata.sourceImage[i-1].stddev < MinStdDev )
    if( is_empty == "true")
      set( newLastZplane = i )
    end()
  else()
    set( is_empty= "false" )
  end()
end()
set( FirstZplane = newFirstZplane)
set( LastZplane = newLastZplane)

if( LastZplane <FirstZplane )
  set(LastZPlane=FirstZPlane)
  return()
  //stop(1, "Stack out of focus, program execution stopped" )
end()

//write( "\tnew FirstZplane = " & FirstZplane, logfilename, "ascii", append=true)
//write( "\tnew LastZplane = " & LastZplane, logfilename, "ascii", append=true)

set(i=FirstZplane, i2=0)
foreach(FirstZplane..LastZplane)
  set(datacube[i2]=sourcedata.sourceImage[i-1])
```

```
    set(i2=i2+1)
  end()
  foreach(FirstZplane_in..LastZplane_in)
    delete(sourcedata.sourceImage[i-1])
  end()
}




proc LeafCellsDetection(
image image1 in "Input image with intensity information. Leaf cells are detected by this image",
bool showillustrations=yes in,
ObjectList LeafCells out "output list of leaf cells",
ObjectList BigLeafCells out "output list of big leaf cells",
objectlist Lines out "List of line structure",
)
{
  thresholdxx(4,image=Image1)
  mean(image=image1)
  mask(threshold,image=image)
  set(M_bg=mask)
  blank(image1.width, image1.height,1)

  set(image.type="mask")
  carrypixels(image=image,mask=M_bg.vector,data=0)
```

```
set(M_bg2=image)


blank(image1.width, image1.height,0)

set(image.type="mask")

set(IM_blank=image)

set(image[1,1]=1, image[image1.width-2,1]=1, image[image1.width-2,image1.height-2]=1, image[1,image1.height-2]=1)


mask2stencil(image)


stencil2objects()

calczone(50)

zonemask(-50,oo)

and(image=objects.zonemask.image,mask=M_bg2)

setattr(M_bg3,image.vector)

calcarea(M_bg3)

objectfilter(M_bg3_area>800)

set(ErosionDistance=sqrt(image.width*image.width+image.height*image.height))

set(NUmberOfSteps=round(ErosionDistance/3)-1)

calczone(ErosionDistance,stencil=M_bg2)

zonemask(-ErosionDistance)

CalcFillStencil_MPIZ(stencilname="zonemask")

renameattr(zonemask=zonemask_filled)

calcarea(zonemask)

objectfilter(zonemask_area>10000)


set(OL_erase=objects)
```

```
////// Creates initial mask of lines
Bright_mask(image1,27)
mask2stencil(M_bright)//, Neighbourhood=25)
if(stencil.itemcount>31999)
  ReduceStencilObjectsByAreaTo31999_MPIZ()
end()
stencil2objects()

calcarea()
set(ob1=objects)

set(ob00=objects)

FillSmallHoles()
//////////////////////////////////////////////////////////////////////////////////////////



///////////////////////////////////////////////////////////////////////////////// Cleanes the initial mask
///// Finds Skeleton

set(IM_start=objects.body.mask.image)
maskthinning_temp(objects,image1)
maskthinning_temp(objects,image1)
maskthinning_temp(objects,image1)
maskthinning_temp(objects,image1)


//if(Showillustrations)
  //imageview(image1, "Signal",image=image1,gamma=2.0)
  //imageview(objects.body, "Lines0",image=image1,gamma=2.0)
//end()
//CalcSkeletonByIntensity(image=image1,intensityEvalParam=0)
renameattr(skeleton=body)
CalcSkeletonByIntensity(stencil=skeleton, image=image1, IntensityEvaluationMode=2, IntensityEvalParam=-1)
renameattr(skeleton=skeleton_skeleton)
CalcSkeletonByIntensity(stencil=skeleton, image=image1, IntensityEvaluationMode=2, IntensityEvalParam=-2)
renameattr(skeleton_skeleton_byIntensity=skeleton_skeleton)
CalcSkeleton(stencil=skeleton_skeleton_byIntensity)
renameattr(skeleton_skeleton=skeleton_skeleton_byIntensity_skeleton)
set(obx=objects)
mask2stencil(objects.skeleton_skeleton.mask, neighbourhood=8)
stencil2objects()
setattr(skeleton_skeleton,body)
calczone(-40, stencil=obx.body)
zonemask(-40,oo)
calcarea()
objectfilter(area>20)


set(IM_skeleton=objects.skeleton_skeleton.mask.image)


renameattr(sk=skeleton_skeleton)
```

```
if(ShowIllustrations)
  //imageview(objects.sk, "Lines1",image=image1,gamma=2.0)
end()
div(result,image1, infinity=0, spreadfactor=1000, uncertainty=0, result_type="Unsigned,short")
mask(0.1,image=result)
set(M_10=mask)
mask(result.median, image=result)
and(image=M_10.image,mask=mask.image)
and(image=objects.sk.image,mask=image)
setattr(sk_thresholded,image.vector)
set(ob2=objects)

RemoveSkeletonLayer(skeletonName="sk")
set(ob3=objects)
RemoveSkeletonLayer(skeletonName="sk_LayerRemoved")
set(ob4=objects)
RemoveSkeletonLayer(skeletonName="sk_LayerRemoved_LayerRemoved")
renameattr(sk_mod=sk_LayerRemoved_LayerRemoved_LayerRemoved)
calcarea(sk_mod)
calcintensity(sk_mod,image=result)
set(ob5=objects)

/////////////////// The found skeleton is in the list ob5, stencil sk_mod
//////////////////////////////////////////////////////////////////////// End of  Finds Skeleton
```

```
////////////////////////////////////////////////// Cell detection, creates from skeleton cells borders

stencil2objects(objects.sk_mod)

FillSmallHoles()


setattr(filled,objects.filled,objects=ob5)

calcarea(filled)

calcattr(RelativeHoleArea,(1.0*filled_area)/(1.0*sk_mod_area))

objectfilter((sk_mod_area>100 or sk_mod_intensity>10) and sk_mod_area>30)

objectfilter(RelativeHoleArea<0.04 or sk_mod_area>100)


calcerosion(-3,stencil=sk)


Calcskeleton(sk_mod)

RemoveSkeletonLayer(skeletonName="sk_mod_skeleton")


set(Lines=objects)

if(ShowIllustrations)

  //imageview(lines.sk, "Lines2",image=image1,gamma=2.0)

  //imageview(lines.sk_LayerRemoved, "Lines3",image=image1,gamma=2.0)

  //imageview(lines.sk_mod, "Lines4",image=image1,gamma=2.0)

end()

blank(image1.width,image1.height,1)

set(IM_blank=image)

set(IM_blank.type="mask")

clearborders(IM_blank,4)
```

```
not(image=stencil)
clearborders(image,3)

or(image=stencil,mask=Lines.sk_eroded.mask.image)
set(M_border=image)
not(image=image)
clearborders(image,3)
set(M_body=stencil)

mask2stencil(M_body)
stencil2objects()

calcerosion(-5)
stencil2objects(objects.eroded)
calcarea()

RemoveSmallObjects(minarea=650)

FillSmallHoles(minimumholearea=400)
if(lines.count==0)
  calcarea()
  objectfilter(area>image1.width*image1.height+10)
else()
  if(lines.body.area>200)
```

```
    if(OL_erase.count>0)
      carrypixels(image=objects.body.image,mask=ol_erase.zonemask.mask,data=0)
      setattr(body,image.vector)
      setattr(index,image)
      calcborder()
      calcarea()
      objectfilter(area>1200)

      calcintensity(image=image1)
      and(image=objects.filled.image,mask=objects.body.mask.image)
      setattr(filled,image.vector)
    end()
  else()
    calcarea()

    objectfilter(area>image1.width*image1.height+10)
  end()
end()
set( keepobjects = objects)
objectfilter(body.area<40000)
objectfilter(body.area>800)
set(LeafCells=objects)
set( objects = keepobjects)
objectfilter( body.area >=40000)
set(BigLeafCells = objects)
```

```
}




proc DetectType2Objects(
  image image1 in "Input image with intensity information",
  objectlist Lines in "Input list of line structure",
  objectlist LeafCells inout "Input-output list of cells. In output list regions, which belong to type2 objects are erased.",
  bool showillustrations=yes in,
  objectlist ObjectsType2 out "List of detected type objects"
)
{
  ////////////////////////////////////////////////// Finds Type2 objects

  stencil2objects(Lines.sk_mod)

  FillSmallHoles(minimumholearea=1000)
  calcarea()
  calcarea(filled)
  mask2stencil(objects.filled.mask)
  stencil2objects()
  calcarea()
  objectfilter(area>150)
  calcerosion(-1)
```

```
and(image=objects.eroded.image, mask=lines.sk_mod.mask.image)

setattr(border_corrected,image.vector)

calcintensity(border_corrected,image=image1)


objectfilter(border_corrected_intensity>1000)




///New part added in Nov 9, 2006

Dark_Mask(image1, 7)

CalcErosion(-1)

Stencil2Objects(Objects.eroded)

Calcintensity(image=result)

renameattr(DarkSig=intensity)

Calcintensity(image=image1)

CalcAttr(RelDarkSig,(1.0*DarkSig)/(intensity))


CalcIntensity(border,image=image1)

Calczone(-3,zonetype="equidistant")

zonemask(-3,3)

renameattr(BorderRegion=zonemask)

Zonemask(3,oo)

Renameattr(InnerRegion=Zonemask)

Calcintensity(InnerRegion,image=image1)


Calcintensity(BorderRegion,image=image1)
```

```
CalcAttr(Inner2BorderRatio, InnerRegion_intensity/Border_intensity)

CalcAttr(Border2BorderRatio, BorderRegion_intensity/Border_intensity)

Zonemask(4, 5)

Renameattr(Region4=Zonemask)

CalcIntensity(Region4, image=image1)

Mean(5,image=image1)


CalcStat("Min",AttrName="MinIntensity", stencil=InnerRegion,image=image)

//Zonemask(7,oo)

//Renameattr(Region7=Zonemask)

//CalcIntensity(Region7, image=IM_projected1)

CalcAttr(MinToRegion4Ratio, (1.0*MinIntensity)/Region4_intensity)

CalcAttr(MinToBorderRatio, (1.0*MinIntensity)/Border_intensity)

CalcAttr(DarkToRegion4, (1.0*DarkSig)/Region4_intensity)

CalcArea()

//CalcWidthLength()

//CalcAttr(Width2LengthRatio,(2.0*Half_width)/(full_length))

deleteattr(zone,outerzone)

set(ObjectsType2=objects)



///// End of finds Type2 objects


///// Removes from list of cells Type2 objects

carrypixels(image=LeafCells.body.image,mask=ObjectsType2.body, data=0)
```

```
  //carryobjects(image=cells.body.image, stencil=ObjectsType2.body, data=ObjectsType2.ObjectNumber)

  stencil2objects(image.vector)

  set(LeafCells=objects)

}


proc CalcFillStencil_MPIZ(

  string StencilName="body" noquote in "Name of the stencil type attribute in the input object list, which will be filled. Only short attribute names are supported: StencilName=body is correct whereas centers=Objects.Body is erroneous. The attribute name can be given with and without the quotation marks, i.e. both are correct: StencilName=body and StencilName=\"body\".",

  objectlist objects inout "Input-output object list. The output list includes a new stencil type attribute, which corresponds to the created objects. A number of objects is equal to a number of the initial centers. In case of body stencil (StencilName=\"body\") the added attribute has the name \"Filled\" otherwise \"StencilName_Filled\". For example, if StencilName=\"BrightMask\" the attribute name is \"BrightMask_Filled.\"."

) object list attribute creation "Fills the gaps (holes) in the geometrical regions given by a stencil type attribute. The output object list includes a new stencil, which corresponds to the \"filled regions\". See also the module FillObjects(), which performs exactly the same operation. The difference is that CalcFillStencil_MPIZ() can be applied to all stencil type attributes while FillObjects() only to the body one."

{

  set(objects_in=objects)

  eval("set(Stencil=objects." & StencilName & ")")

  if(errorcode==0)

    if(Stencil.class!="intervalvector")

      error("Input StencilName does not correspond to the stencil-type attribute. The input StencilName must specify the stencil type attribute in the input object list, which defines the centers positions. Only short attribute names are supported: centers=body is correct whereras centers=Nuclei.Body is erroneous. The attribute name can be given with and without the quatation marks, both are correct: centers=body and centers=\"body\". It is assumed that the centers approximately correspond to the local intensity maximums (maximum regions) or to minimums (minimum regions).")

    end()

  else()

    FindShortAttrNameFromLongName(StencilName)

    eval("set(Stencil=objects." & ShortAttrName & ")")

    if(errorcode==0)

      if(Stencil.class=="intervalvector")

          error("Input StencilName does not correspond to the attribute in the input object list. Probably, a long attribute name was used instead of a short one. Only short attribute names are supported: StencilName=body is correct whereras StencilName=Objects.Body is erroneous. The attribute name can be given with and without the quatation marks, both are correct: StencilName=body and StencilName=\"body\". It is assumed that the centers approximately correspond to the local intensity maximums (maximum regions) or to minimums (minimum regions).")
      end()
    end()
    error("The input StencilName does not correspond to the existing stencil type attribute in the input object list. The input StencilName must specify the stencil type attribute in the input object list.. Only short attribute names are supported: StencilName=body is correct whereras StencilName=Objects.Body is erroneous. The attribute name can be given with and without the quatation marks, both are correct: StencilName=body and StencilName=\"body\".")
  end()


  stencil2objects(stencil)
  Fillobjects()
  if(objects.count!=objects_in.count)
    set(st_temp=objects.body)
    FillStencilfromCenter_tech(objects_in.body,objects.body)
    stencil2objects(stencil)
  end()
  set(ob2=objects)
  if(StencilName=="body")
    setattr(Filled,ob2.body,objects=objects_in)
    setattr(Filled_border,ob2.border)
  else()
    set(temp="" & StencilName & "_Filled,ob2.body,objects=objects_in)")
    eval("setattr(" & temp )
    set(temp="" & StencilName & "_Filled_border,ob2.border)")
    eval("setattr(" & temp )
  end()
}
```

```
proc Haustoria_detection(
//INPUT
image reference in "Nuclei stained image with intensity information. Nuclei are detected by this image. Supported image types are: 8-bit and 16-bit.",
objectlist Lines in "Input list of line structure. Used only for additional attribute creation",
bool ShowIllustrations=NO in "YES- Output illustrations are depicted. No- Output illustrations are not shown.",
//OUTPUT
objectlist Haustoria out "Output object list with initial objects. Object filter must be applied in the next steps for Haustoria classification.",
double line_mean out "Mean intensity of the line structure",
double nuclei_quantile7 out "Quantile 0.7 for intensity of nuclei"
) object recognition "Detects initial objects for haustoria detection. Object filter must be applied in the next steps for Haustoria classification. The procedure corresponds to nuclei_detection_A() routine."
{
 Input(MinimumHaustoriaArea, 120, "Minimum Haustorium Area","i", "Minimum allowed area for haustorium, objects with area less than the limit are removed.")


 // Inner-technical nuclei detection procedure
 //nuclei_detection_A_inner(reference, 0.9, MinimumHaustoriaArea, 0.4, 0.7)  // AK 3.12.2007
 nuclei_detection_A_inner(reference, 0.9, MinimumHaustoriaArea, 0.4, 1) // AK 3.12.2007


 ErrorReceiverForNucleiDetectionLibrary_v1()
 if(ShowIllustrations)
   imageview(item=InitialMask.border.mask, label="InitialMask", title="Mask after the initial thresholding. Adjust par. \"Threshold Adjustment\"", image=reference, gamma=2.6)
 end()
 // Removes breaking lines between the stuck nuclei,
 // Input parameter is fixed
 controlbreakinglines_v12(nuclei, 0.8)
 // Adds to object list a contrast attribute
```

```
object_contrast_general(cells_out, reference, InitialMask=InitialMask.body.mask)


// Creates the illustration "LowContrastObjects"

if(ShowIllustrations)

  set(CL_Temp=objects)

  objectfilter(contrast<=0.1)

  imageview(item=objects.border, label="LowContrastObjects", title="Discarded low contrast objects. Adjust par. \"Minimum Nuclear Contrast\"", image=reference, gamma=2.6)

  set(objects=CL_Temp)

end()

objectfilter(contrast>0.1)


if(ShowIllustrations)

  imageview(item=objects.border, label="InitialHaustorial", title="Initial candidates for Haustoria. Classification must be applied.", image=reference, gamma=2.6)

end()

calcarea()

calcintensity(image=reference)


and(image=objects.body.image, mask=Lines.sk_mod_skeleton_LayerRemoved.mask.image)

setattr(lines, image.vector)


if(Lines.sk_mod_skeleton_LayerRemoved.area>50)

  stat("mean",variable="line_mean", mask=Lines.sk_mod_skeleton_LayerRemoved,image=reference)

else()

  set(line_mean=reference.mean)

end()

if(objects.count>30)
```

```
      quantile(objects.intensity,0.7)
      set(nuclei_quantile7=quantile)
    else()
      if(objects.count>24)
        quantile(objects.intensity,0.6)
        set(nuclei_quantile7=quantile)
      else()
        if(objects.count>20)
          quantile(objects.intensity,0.5)
          set(nuclei_quantile7=quantile)
        else()
          set(nuclei_quantile7=line_mean)
        end()
      end()
    end()
    calcarea(lines)
    calcwidthlength()
    calcattr(Width2LengthRatio, (2.0*half_width)/(full_length))
    //objectfilter(intensity>line_mean and half_width>4 and lines_area<5 and  (Width2LengthRatio>0.45 or (intensity>1.4*line_mean and Width2LengthRatio>0.4)))

    deleteattr(zone, outerzone)
    set(Haustoria=objects)
  }

  proc SpotClassificationLeafCells(
  image IM_projected1 in "Input image with intensity information. Spots are detected by this image",
  objectlist spots in "Input list of the initial spots",
  bool showillustrations in "Yes- depicts the spot classification illustrations, No- the spot classification illustrations are not shown",
  objectlist SpotsFiltered out "List of the classified spots with the calculated attributes"
  ) "Detects the actual spot locations by the initial list. Creates a new list and calculates attributes like area, contrast, width, length etc and thereupon classifies/filters the spots."
  {
      input(spotMinimumArea, 1, "SpotMinimumArea","i","Minimum allowed area for spots. Objects with area less than the limit are discarded.")
      input(SpotMinimumContrast, 0.18, "SpotMinimumContrast", "d", "Minimum allowed contrast for spots. Objects with the parameter value less than the limit are discarded.")
      input(SpotMinimumRoundness, 0.1, "SpotMinimumRoundness", "d", "Minimum allowed roundness parameter for spots. Objects with the parameter value less than the limit are discarded.")
      input(MinimumWidth2LengthRatio, 0.5, "SpotMinimumWidth2LengthRatio","d", "Minimum allowed width to length ratio for spots. Objects with the ratio below the limit are discarded.")

      Bright_Mask(IM_projected1,3)
      set(r6=result)
      div(result, IM_projected1, result_type="unsigned,short", spreadfactor=1000, infinity=0)
      mask(threshold=0.05,image=result)
      set(M10=mask)
      mask(threshold=IM_projected1.median, image=r6)
      and(image=M10.image,mask=mask.image)
      set(M11=image)

      calcerosion(-3.3,spotcenters,objects=spots)
      renameattr(centerseroded3=spotcenters_eroded)

      calcerosion(-2,spotcenters)
      calcerosion(-12,spotcenters_eroded,restrictivestencil=m10, numberofsteps=3)
      CalcFillStencil_MPIZ(StencilName="spotcenters_eroded_eroded")
```

```
RenameAttr(body2=spotcenters_eroded_eroded_filled)

calcborder(body2)

CalcArea(body2)

set(OL_temp1=objects)

stencil2objects(objects.body2)

CalcWidthLength()

set(OL_temp2=objects)

setattr(FullLength,ol_temp2.full_length,objects=ol_temp1)

setattr(HalfWidth,ol_temp2.half_width)

CalcAttr(Width2LengthRatio, iif(FullLength>0,(2.0*HalfWidth/FullLength),0.0))

CalcRoundnessCorrected(Body2)


setattr(body4,body2)

calcerosion(1,body4)

calcerosion(-1,body4_eroded)

renameattr(body4=body4_eroded_eroded)

set(OL_temp1=objects)

stencil2objects(objects.body4)

CalcWidthLength()

set(OL_temp2=objects)

setattr(body4_FullLength,ol_temp2.full_length,objects=ol_temp1)

setattr(body4_HalfWidth,ol_temp2.half_width)

CalcAttr(body4_Width2LengthRatio, iif(FullLength>0,(2.0*HalfWidth/FullLength),0.0))

CalcArea(body4)
```

```
CalcRoundnessCorrected(Body4)

CalcAttr(AreaRatio42,iif(Body2_area>0, (1.0*Body4_area)/(1.0*Body2_area),0.0))


objectfilter(body2_area>0)

if(ShowIllustrations)

  imageview(objects.body2_border, "InitialSpots", image=IM_projected1, title="Spot candidates after the initial detection", gamma=2.0)

end()

ObjectFilter(HalfWidth>0.9)

//ObjectFilter(HalfWidth>2.01)

ObjectFilter(Width2LengthRatio>MinimumWidth2LengthRatio or (Width2LengthRatio>0.8*MinimumWidth2LengthRatio and body2_roundnesscorrected>1.1*SpotMinimumRoundness ) or (body4_Width2LengthRatio>0.6 and body4_RoundnessCorrected>0.8 and AreaRatio42>0.75) )

if(ShowIllustrations)

  //imageview(objects.body2_border, "WidthLength", image=IM_projected1, title="Spots after filtering by  Width and Width2LengthRatio", gamma=2.0)

end()

//set(ol_s1a=objects)


objectfilter(Body2_roundnesscorrected>=SpotMinimumRoundness )

if(ShowIllustrations)

  //imageview(objects.body2_border, "Roundness", image=IM_projected1, title="Spots after filtering by  Roundness", gamma=2.0)

end()



objectfilter(body2_area>=spotMinimumArea)

if(ShowIllustrations)

  //imageview(objects.body2_border, "Area", image=IM_projected1, title="Spots after filtering by  Area", gamma=2.0)

end()
```

```
CalcErosion(-2,body2)

calcborder(body2_eroded)

CalcIntensity(Body2_eroded_border, image=IM_projected1)

CalcStat("max",stencil=body2, image=IM_projected1)


CalcAttr(th, Body2_eroded_border_intensity+0.3*(max-Body2_eroded_border_intensity)+2.0*sqrt(Body2_eroded_border_intensity))

threshmask(stencil=body2_eroded,threshold=th,image=IM_projected1)

calcerosion(-13, spotcenters_eroded,restrictivestencil=threshmask)

renameattr(body3=spotcenters_eroded_eroded)

calcattr(body2_contrast, (spotpeakintensity-Body2_eroded_border_intensity)/(spotpeakintensity+Body2_eroded_border_intensity))

selectbrightspots(body2,body2,image=IM_projected1)

calcerosion(-1,brightspots)

carrypixels(image=objects.brightspots_eroded.image,mask=objects.brightspots.mask,data=0)

setattr(region8,image.vector)

calcintensity(region8,image=IM_projected1)

calcattr(PeakToReg8Intensity, (1.0*max-Region8_intensity)/(1.0*max-Body2_eroded_border_intensity))


objectfilter(body2_contrast>SpotMinimumContrast )

if(ShowIllustrations)

  imageview(objects.body2_border, "Contrast", image=IM_projected1, title="Spots after filtering by  Contrast", gamma=2.0)

end()


//objectfilter(PeakToReg8Intensity<0.5)

objectfilter(PeakToReg8Intensity<0.8)


if(ShowIllustrations)
```

```
      //imageview(objects.body2_border, "Peak2Neighborhood", image=IM_projected1, title="Spots after filtering by  Peak to neighborhood filter", gamma=2.0)

    end()


    calcerosion(-16,spotcenters_eroded,restrictivestencil=m10, numberofsteps=4)

    CalcFillStencil_MPIZ(StencilName="spotcenters_eroded_eroded")


    RenameAttr(body2=spotcenters_eroded_eroded_filled)

    calcborder(body2)

    CalcArea(body2)

    set(OL_temp1=objects)

    stencil2objects(objects.body2)

    CalcWidthLength()

    set(OL_temp2=objects)

    setattr(FullLength,ol_temp2.full_length,objects=ol_temp1)

    setattr(HalfWidth,ol_temp2.half_width)

    CalcAttr(Width2LengthRatio, iif(FullLength>0,(2.0*HalfWidth/FullLength),0.0))

    CalcRoundnessCorrected(Body2)



    //ObjectFilter((Width2LengthRatio>=MinimumWidth2LengthRatio and body2_roundnesscorrected>SpotMinimumRoundness) or (Width2LengthRatio>0.8*MinimumWidth2LengthRatio and SpotMinimumRoundness>1.1) )

    ObjectFilter((Width2LengthRatio>=0.8*MinimumWidth2LengthRatio and body2_roundnesscorrected>SpotMinimumRoundness))


    Set(SpotsFiltered=objects)

    // Cleanes the list, renames the attributes etc

    Stencil2Objects(objects.body2)
    CalcIntensity(image=IM_projected1)

    CalcStat("Sum",AttrName="IntegratedSpotSignal",Stencil=Body,image=IM_projected1)

    Setattr(Area,SpotsFiltered.body2_area)

    Setattr(RoundnessCorrected,SpotsFiltered.body2_RoundnessCorrected)

    Setattr(Width2lengthRatio,SpotsFiltered.Width2lengthRatio)

    Setattr(FullLength,SpotsFiltered.FullLength)

    Setattr(HalfWidth,SpotsFiltered.HalfWidth)

    Setattr(Contrast, SpotsFiltered.Body2_contrast)

    Setattr(PeakIntensity, SpotsFiltered.max)

    Setattr(SpotCenters, SpotsFiltered.SpotCenters)


    SetAttr(ReferenceRegions,SpotsFiltered.Body2_eroded_border)

    SetAttr(ReferenceRegions_intensity,SpotsFiltered.Body2_eroded_border_intensity)

    SetAttr(ReferenceIntensity, ReferenceRegions_intensity)

    CalcArea(ReferenceRegions)

    SetAttr(CellIntensity, SpotsFiltered.CellIntensity)

    CalcAttr(SpotToCellIntensity, (1.0*PeakIntensity)/(1.0*CellIntensity))

    CalcAttr(IntegratedSpotSignal_Backgroundsubtracted, 1.0*IntegratedSpotSignal-(area*ReferenceIntensity))

    CalcAttr(IntegratedSpotSignal_Backgroundsubtracted, iif(IntegratedSpotSignal_Backgroundsubtracted>0.0000001,IntegratedSpotSignal_BackgroundSubtracted,0.0))

    Set(SpotsFiltered=objects)

}


proc Spot_detection_outputs_modified(objectlist objects in  "Input object list of ~cells (SearchObjects) with spot data, must contain same attributes as spot detection output list WholeCells. Please note that list of spots can not be used here as it does not contain all required data, e.g. number of ~cells, ~cell area etc.",

  bool ShowOutputParameters=YES in "YES- Output parameters are reported to player and/or to database. No- Output Parameters are not reported.",

  string NamePreFix="" in "String, which is added to output names as prefix",
  double NumberOfSpots out "Total number of detected spots.",

  double SpotsPerObject out "Number of spots per object (i.e. number of spots per ~cell).",

  double SpotsPerArea out "Number of spots per SearchRegion area (i.e. number of spots per visible ~cell area).",

  double IntegratedSpotSignalPerCellularSignal out "Integrated spot signal over all spots normalized by integrated ~cellular signal (total signal over all SearchRegion area).",

  double IntegratedSpotSignalPerCellularSignal_BackgroundSubtracted out "Integrated spot signal over all spots background subtracted and normalized by integrated ~cellular signal (total signal over all SearchRegion area).",


  double IntegratedSpotSignalPerArea out "Integrated spot signal per SearchRegion area (per visible ~cell area).",

  double IntegratedSpotSignalPerArea_BackgroundSubtracted out "Integrated spot signal BackgroundSubtracted per SearchRegion area (per visible ~cell area)."


  )  spot detection "Supporting procedure for spot detection library. Reports spot detection outputs. Please note that output parameters can be reported by list of SearchObjects (e.g. list of ~cells), which contains spot data. List of spots does not contain information about number of ~cells, ~cell area etc and therefore all spot outputs can not be reported by this list. Input list Objects must contain same attributes (spot data) as spot detection output list WholeCells. Could be used in multiple-field scripts or after spot or ~cell classification etc. See more in Opera spot detection manual."

{


  if(objects.count>0)


    set(NumberOfSpots=1.0*objects.NumberOfspots.sum)

    set(SpotsPerObject=NumberOfSpots/(1.0*objects.count))

    if(objects.searchregionarea.sum>0)

      set(SpotsPerArea=NumberOfSpots/objects.searchregionarea.sum)

      set(IntegratedSpotSignalPerArea=(1.0*objects.integratedspotsignal.sum)/(1.0*objects.searchregionarea.sum))

      set(IntegratedSpotSignalPerArea_backgroundsubtracted=(1.0*objects.integratedspotsignal_backgroundsubtracted.sum)/(1.0*objects.searchregionarea.sum))


    else()

      set(SpotsPerArea=NAN)

      set(IntegratedSpotSignalPerArea=NAN)
      set(IntegratedSpotSignalPerArea_backgroundsubtracted=NAN)
    end()

    if(objects.searchregionintegratedSignal.sum>0)
      set(IntegratedSpotSignalPerCellularSignal=(1.0*objects.integratedspotsignal.sum)/(1.0*objects.searchregionintegratedSignal.sum))
      set(IntegratedSpotSignalPerCellularSignal_backgroundsubtracted=(1.0*objects.integratedspotsignal_backgroundsubtracted.sum)/(1.0*objects.searchregionintegratedSignal.sum))
    else()
      set(IntegratedSpotSignalPerCellularSignal=NAN)
      set(IntegratedSpotSignalPerCellularSignal_backgroundsubtracted=NAN)
    end()
  else()



    set(NumberOfSpots=0.0)
    set(SpotsPerObject=NAN)
    set(SpotsPerArea=NAN)
    set(IntegratedSpotSignalPerCellularSignal=NAN)
    set(IntegratedSpotSignalPerCellularSignal_backgroundsubtracted=NAN)
    set(IntegratedSpotSignalPerArea=NAN)
    set(IntegratedSpotSignalPerArea_backgroundsubtracted=NAN)
  end()

  if(ShowOutputParameters)
    create_spot_outputs()
  end()
```

```
}

//(proc Spot_detection_outputs_NoValidFields(

// bool ShowOutputParameters=YES in "YES- Output parameters are reported. No- Output Parameters are not reported.",

// string NamePreFix="" in "String, which will be added to output names as prefix",

// double NumberOfSpots out "Total number of detected spots.",

// double SpotsPerObject out "Number of spots per object (i.e. number of spots per ~cell).",

// double SpotsPerArea out "Number of spots per SearchRegion area (i.e. number of spots per visible ~cell area).",

// double IntegratedSpotSignalPerCellularSignal out "Integrated spot signal over all spots normalized by integrated ~cellular signal (total signal over all SearchRegion area).",

// double IntegratedSpotSignalPerCellularSignal_BackgroundSubtracted out "Integrated spot signal over all spots background subtracted and normalized by integrated ~cellular signal (total signal over all SearchRegion area).",

//

// double IntegratedSpotSignalPerArea out "Integrated spot signal per SearchRegion area (per visible ~cell area).",

// double IntegratedSpotSignalPerArea_BackgroundSubtracted out "Integrated spot signal BackgroundSubtracted per SearchRegion area (per visible ~cell area)."

//

// )   spot detection "Supporting procedure for spot detection library. Used in spot detection multiple field script. Reports spot detection outputs with NAN values, called if there are no valid fields. See paragraph about Opera_multifields_spots.script in Opera spot detection manual."

//{

// set(NumberOfSpots=NAN)

// set(SpotsPerObject=NAN)

// set(SpotsPerArea=NAN)

// set(IntegratedSpotSignalPerCellularSignal=NAN)

// set(IntegratedSpotSignalPerCellularSignal_backgroundsubtracted=NAN)

// set(IntegratedSpotSignalPerArea=NAN)

// set(IntegratedSpotSignalPerArea_backgroundsubtracted=NAN)
// if(ShowOutputParameters)

//   create_spot_outputs()

// end()

//})




proc CalcRoundnessCorrected(

  String StencilName="body" noquote in "Name of a stencil stencil type attribute in the input list. The input must correspond to the stencil type attribute in the input list. Only short syntax of attribute names is supported: StencilName=body is correct and  StencilName=Objects.Body is erroneous. The input can be given with and without the quatation marks, both are correct: StencilName=body and StencilName=\"body\".",

  objectlist objects inout "Input-output object list. The output list contains the calculated corrected roundness attribute RoundnessCorrected. In case of body stencil (StencilName=\"body\") the attribute name is \"RoundnessCorrected\" otherwise \"StencilName_RoundnessCorrected\". For example, if StencilName=\"MembraneRegion\" the attribute name is \"MembraneRegion_RoundnessCorrected\"."

) object list attribute creation "Finds the corrected roundness parameter for a stencil type attribute. The output list contains the found attribute RoundnessCorrected."

{

  set(objects_in=objects)

  if(!defined("objects." & StencilName))



    error("In the input object list there is no attribute with name " & StencilName & ". The input StencilName must correspond to the stencil type attribute in the input list. Only short syntax of attribute names is supported: StencilName=body is correct and  StencilName=Objects.Body is erroneous. The input can be given with and without the quatation marks, both are correct: StencilName=body and StencilName=\"body\".")

  end()

  eval("SetAttr(CurrentStencilX, objects." & StencilName & ")")

  if(errorcode!=0)

    error()

  end()

  if(objects.CurrentStencilX.class!="intervalvector")

      error(StencilName & " is not a stencil type attribute. The input StencilName must  correspond to a stencil type attribute in the input object list. Only short syntax of attribute names is supported: StencilName=body is correct and  StencilName=Objects.Body is erroneous. The input can be given with and without the quatation marks, both are correct: StencilName=body and StencilName=\"body\".")
  end()


  CalcArea(CurrentStencilX)

  CalcBorder(CurrentStencilX)

  CalcArea(CurrentStencilX_border)

  CalcAttr(RoundnessCorrected, 3.544*sqrt(1.0*CurrentStencilX_Area-CurrentStencilX_border_area/2.0)/CurrentStencilX_border_area-0.1, autorecalc=no)


  if(StencilName=="body")

    setattr(RoundnessCorrected, objects.RoundnessCorrected,objects=objects_in)

  else()

    set(temp="" & StencilName & "_RoundnessCorrected, objects.RoundnessCorrected,objects=objects_in)")

    eval("setattr(" & temp )

  end()

}
```

```
////////////////////////////////////////////////// PROCEDURE DEFINITIONS

// Procedure counts fields and controls if all images are present

proc ControlImageFieldsStackAC2(Table sourcedata in, int NumberOfChannels in, int ZplanesInStack in, int StackNo in, int StackCount out, int StartStack out, int EndStack out, int ImagesINOneStack out, int InvalidStacks out)

{

  set(ImagesINOneStack=NumberOfChannels*ZplanesInStack)    //calculates number of images per stack

  if(ImagesINOneStack<1)

    error("Number of channels or number of images in stack is not a positive number. Please select a number larger than 0.")
```

```
end()


set (ImageCount = sourcedata.rowcount)   // determine the number of images present



if ((ImageCount ~ ImagesINOneStack) != 0)
  error("Number of images ("&ImageCount&") is not a multiple of the number of images per stack ("&ImagesINOneStack&").")
end()

set (StackCount = int(ImageCount / ImagesINOneStack))


if (StackCount == 0)
  error("No sufficient number of images found in the data file. Only "&StackCount&" images found, but "&ImagesINOneStack&" needed for a stack.")
end()


if (StackNo == 0)   // all images present are analysed
  set (StartStack = 1)
  set (EndStack = StackCount)
else()
  if (StackNo > StackCount)       /// Error message if too high Image Field number was chosen
    error("Please select a smaller field number. There are only "&StackCount&" stacks in the file, but you selected to analyse stack "&StackNo&". ")
  end()
  if (StackNo < 0)   ////Error message if negative Image Field number was chosen
    error("Please select a positive stack number.")
  end()
  set (StartStack = StackNo)
```

```
    set (EndStack = StackNo)
    set (StackCount=1)
  end()
  set(InvalidStacks=0)   //variable for counting of invalid black image fields
}

//Procedure assigns image names for the current field
proc Assign_FirstLastZplane(Table sourcedata in, int _StackCounter in, int ImagesINOneStack in, int NumberOfChannels in, int FirstZplane out , int LastZplane out)
{
  set(FirstZplane=(_StackCounter-1)*ImagesINOneStack+1)
  set(LastZplane=(_StackCounter)*ImagesINOneStack-NumberOfChannels+1)
}



//////////////////////////////////////////////// MAIN SCRIPT STARTS
// INPUT PARAMETERS
input(StackNo, 0, "Stack No", "i", "Number of the stacks to analyze. If set to 0 all stacks are evaluated.")
input(NumberOfChannels, 1, "Number Of Channels", "i", "Number of channels per field")
input(ZplanesInStack,21,"Zplanes in stack","i","Number of z-planes in stack, a number of time moments in kinetic measurement")
input(ShowIllustrations,YES, "ShowIllustrations","y","YES- Output illustrations are depicted. No- Output illustrations are not shown.")

input(MinStdDev,4,"MinStdDev","i","Minimum standard deviation of pixel intensity in image.")
comment(
input(spotMinimumArea, 21, "SpotMinimumArea","i","Minimum allowed area for spots. Objects with area less than the limit are discarded.")
input(SpotMinimumContrast, 0.25, "SpotMinimumContrast", "d", "Minimum allowed contrast for spots. Objects with the parameter value less than the limit are discarded.")
input(SpotMinimumRoundness, 0.65, "SpotMinimumRoundness", "d", "Minimum allowed roundness parameter for spots. Objects with the parameter value less than the limit are discarded.")

input(MinimumWidth2LengthRatio, 0.5, "SpotMinimumWidth2LengthRatio","d", "Minimum allowed width to length ratio for spots. Objects with the ratio below the limit are discarded.")

)


Input(IN_WriteCellImages,yes,"Write Cell Images:Image Export","y","Write images with the cells marked to disk")

Input(IN_WriteSpotImages,yes,"Write Spot Images:Image Export","y","Write images with the spots marked to disk")

// READS IN IMAGES

Singlewell(compact=yes)


// COUNTS FIELDS AND CONTROLS IF ALL IMAGES ARE PRESENT

ControlImageFieldsStackAC2(SourceData, NumberOfChannels, ZplanesInStack, StackNo)


// LOOP OVER IMAGE FIELDS


set(InvalidStacks = 0)  //counts black/grey stacks without information

set(BadStacks = 0)   //counts stacks with Signal but no recognized cells

set(all_VarsOK = 0)

Foreach(StartStack .. EndStack, "_StackCounter")

  // Assigns image names for the current field, IM_CH1 - first channel image, IM_CH2 - second channel image etc

  Assign_FirstLastZplane()


  // User script STARTS

  set( imagefilename1 = SourceData.sourcefilename[0])

  //set( logfilename = "D:/dmeyer/test/filenames.txt" , imagefilename1=SourceData.sourcefilename[0])

  CreateDataCubeWithControl( sourcedata, FirstZPlane, LastZPlane, MinStdDev) // Creates the initial data cube

  DataCubeProjectionCorrection(FirstZplane, LastZPlane,DataCube) // Corrects the initial data cube (can be commented out)
```

```
////// Projects maximum intensity for each x-y position from datacube to image
rearrange(datacube,vec(vec(0,0,1),vec(1,0,0),vec(0,1,0)), vec(datacube.depth, datacube.width,datacube.height), reduce="max")

//Maximums3D(0,image=datacube)
//StencilFrom3DTo2D(stencil=maximums,datacube=datacube) //Projects the found maximums from 3-dim to plane
delete(datacube)
set(IM_projected1=result)
set(IM_projected2=result)
/////////////////////////////////////////////////////////////////////////////////////

LeafCellsDetection(IM_projected1) // Detection of Leaf cells

DetectType2Objects(IM_projected1, Lines,LeafCells) //Detects stomata and "holes between cells", Stomata classification is at the end of the script

/////////////////////////////////////////////////// Convolution before spot detection /////////////////////////
// added by kurt stueber 2006-09-08 //
//convolutionmask("Disk",3)
//convolution(image=IM_projected2, faster=yes)
//set(IM_projected2=image)

/////////////////////////////////////////////////// Spot detection
// Creates the SearchRegion, where the spots are searched - for too big Cells
CalcErosion(1,objects=BigLeafCells)
set(BigLeafCells=objects)
```

```
  carrypixels(image=objects.eroded.image, mask=Lines.sk_mod_skeleton_LayerRemoved, data=0)

  Setattr(SearchRegion,image.vector, objects=BigLeafCells)

  CalcArea(SearchRegion)

  Set(BigLeafCells=objects)


  spot_detection_c_inner(IM_projected2 , BigLeafCells.SearchRegion, BigLeafCells, SpotMinimumDistance=2, SpotPeakRadius=1, SpotReferenceRadius=10, SpotMinimumContrast=0.1, SpotminimumToCellIntensity=0.8)

  SpotClassificationLeafCells(IM_projected1,Spots)


  set( BigCellSpots = SpotsFiltered )

  //if(ShowIllustrations)

    //imageview(BigCellSpots.border, "Outer Spots",image=IM_projected2, title = "Spots out of recognized Cell Borders", gamma=2.0)

  //end()


  // Creates the SearchRegion, where the spots are searched - for normal Cells

  CalcErosion(1,objects=LeafCells)

  set(LeafCells=objects)

  carrypixels(image=objects.eroded.image, mask=Lines.sk_mod_skeleton_LayerRemoved, data=0)

  Setattr(SearchRegion,image.vector, objects=LeafCells)

  CalcArea(SearchRegion)

  Set(LeafCells=objects)


  // Detects the initial set of spots

  spot_detection_c_inner(IM_projected2, LeafCells.SearchRegion, LeafCells, SpotMinimumDistance=2, SpotPeakRadius=1, SpotReferenceRadius=10, SpotMinimumContrast=0.1, SpotminimumToCellIntensity=0.8)

  delete(spotcandidates,wholecells)
```

```
// Detects the actual spot locations and attributes like area, contrast, width, length etc and classifies the spots and filters the spots.

SpotClassificationLeafCells(IM_projected1,Spots)


//Detects the inital objects, outputs also statistics Line_mean and nuclei_quantile7

Haustoria_detection(IM_projected1,Lines, showillustrations=no)

// Haustoria classification

objectfilter(intensity>1.3*line_mean and intensity>nuclei_quantile7 and half_width>4 and lines_area<5 and  (Width2LengthRatio>0.45 or (intensity>1.4*line_mean and Width2LengthRatio>0.4)), objects=haustoria)

set(Haustoria=Objects)


// Discards spots, which overlap with haustoria

and(image=SpotsFiltered.body.image,mask=Haustoria.body.mask.image)

setattr(Haustoria,image.vector,objects=SpotsFiltered)

calcarea(Haustoria)

Objectfilter(Haustoria_area<1)

set(SpotsFiltered=objects)


Spot_detection_dataToSearchObjects(IM_projected1, "SearchRegion",SpotsFiltered,LeafCells)

set(LeafCells=Wholecells)


// Stomata and Gaps classification

objectfilter(area>300 and ((DarkToRegion4<0.32 and Border_intensity>Line_mean) or (DarkToRegion4<0.24 and Border_intensity>0.8*Line_mean)), objects=objectstype2)

set(Stomata=objects)

CalcStat("max",Stencil=body,image=Stomata.body.image,objects=objectstype2)

objectfilter(max==0)

set(Gaps=objects)
set( namelength = length( imagefilename1 ) )
set( corename = substr( imagefilename1, 1, namelength - 5 ) )
gamma( 2.0, image=IM_projected1)
set( Image_out = image)
writeimage( imagefile=corename & "_" & FirstZplane & "-" & LastZplane & ".png", image=Image_out, imageformat="png",  )
//write( "\tpng\t" & corename & "_" & FirstZplane & "-" & LastZplane & ".png" , logfilename, "ascii", append=true )

If(IN_WriteCellImages and SpotsFiltered.count>0)
  Gamma(2.0, image=IM_projected1)
  CarryObjects(Leafcells.border, image.max)
  CarryObjects(Leafcells.border, "Rainbow")
  WriteImage(imagefile=corename & "_" & FirstZplane & "-" & LastZplane & "_cells.png", image=Image, imageformat="png")
End()
If(IN_WriteSpotImages and SpotsFiltered.count>0)
  Gamma(2.0, image=IM_projected1)
  CarryObjects(SpotsFiltered.border, image.max)
  CarryObjects(SpotsFiltered.border, "Rainbow")
  WriteImage(imagefile=corename & "_" & FirstZplane & "-" & LastZplane & "_spots.png", image=Image, imageformat="png")
End()

if(showIllustrations)
  imageview(IM_projected1, "Signal",image=IM_projected1, gamma=2.0)
  if(SpotsFiltered.count>0)
    imageview(Leafcells.border, "Cells " & _StackCounter, image=IM_projected1, title="Detected Leaf Cells in Stack # " & _StackCounter, gamma=2.0)
    imageview(Leafcells.body, "Cells Area " & _StackCounter, image=IM_projected1,title="Detected Leaf Cells in Stack #" & _StackCounter, gamma=2.0)
    //imageview(gaps.border, "Gaps",image=IM_projected1,title="Detected gaps between leaf cells",gamma=2.0)
    //imageview(Stomata.border, "Stomata",image=IM_projected1,title="Detected Stomata",gamma=2.0)
    //imageview(Haustoria.border,"Haustoria", image=IM_projected1, title="Detected Haustoria", gamma=2)
    imageview(SpotsFiltered.border, "Spots " & _StackCounter, image=IM_projected1, title="Detected Spots in Stack # " & _StackCounter, gamma=2.0)
    imageview(BigCellSpots.border, "Outer Spots " & _StackCounter, image=IM_projected1, title = "Spots out of recognized Cell Borders in Stack # " & _StackCounter, gamma=2.0)
  else()
    set(InvalidStacks=InvalidStacks+1)
  end()
end()
delete(M10,M11,M8,m9, ol_temp1,ol_temp2,M_bright, m_bright4, IM_planenumber,IM_sk_mod_skeleton_LayerRemoved,mask,image,result,IM_projected2,spotsfilteredout,nuclei,r6,r7,spots,PlanenumberImage)
// User script ENDS

// Output sequence starts
set(EntireNumberOfSpotsInStack = (BigCellSpots.count + SpotsFiltered.count))
set(PercentageCellAreaInPicture = (100 * ((1.0*LeafCells.SearchRegionArea.sum)/(1.0*357760))))
set(SpotsPerValidArea = ((1.0 * SpotsFiltered.count) / (1.0 * LeafCells.SearchRegionArea.sum)))

if(StackNo == 0)  //if complete well
  if (_StackCounter == 1)    //steps for first stack
    push(NumberOfCellsInStackVector, leafcells.count)
    push(NumberOfSpotsInStackVector, SpotsFiltered.count)
    push(EntireNumberOfSpotsInStackVector, EntireNumberOfSpotsInStack)
    if(PercentageCellAreaInPicture > 0.3)
      push(SpotsPerValidAreaVector, SpotsPerValidArea)
    else()
      push(SpotsPerValidAreaVector, NAN)
    end()
    if(SpotsFiltered.count > 0)
      set(PercentageOfValidSpots = (100 * (1.0 * SpotsFiltered.count)/((1.0 * SpotsFiltered.count) + (1.0 * BigCellSpots.count))))
      push(PercentageOfValidSpotsVector, PercentageOfValidSpots)
    else()
      set(PercentageOfValidSpots = 0.0)
      push(PercentageOfValidSpotsVector, PercentageOfValidSpots)
    end()
    if(PercentageOfValidSpots > 25)
      push(AverageAreaOfCellsInStackVector, leafcells.SearchRegionArea.mean)
      push(AverageAreaOfCellsInStackStddevVector, leafcells.SearchRegionArea.stddev)
      push(AverageNumberOfSpotsPerCellVector, leafcells.NumberOfSpots.mean)
      push(AverageNumberOfSpotsPerCellStddevVector, leafcells.NumberOfSpots.stddev)
      push(PercentageCellAreaInPictureVector, PercentageCellAreaInPicture)
      if(all_VarsOK == 0)
        set(all_LeafCells=LeafCells)
        set(all_SpotsFiltered=SpotsFiltered)
        set(all_ObjectsType2=ObjectsType2)
        //set(all_Haustoria=Haustoria)
        set(all_Stomata=Stomata)
        set(all_Gaps=Gaps)
        set(all_VarsOK = all_VarsOK + 1)
      end()
    else()
      push(AverageAreaOfCellsInStackVector, NAN)
```

```
    push(AverageAreaOfCellsInStackStddevVector, NAN)
    push(AverageNumberOfSpotsPerCellVector, NAN)
    push(AverageNumberOfSpotsPerCellStddevVector, NAN)
    push(PercentageCellAreaInPictureVector, NAN)
    Set(BadStacks = BadStacks + 1)
  end()
  push( StacksUsed, _StackCounter )
else()  //steps for stacks # 2...oo
  push( NumberOfCellsInStackVector, leafcells.count)
  push(NumberOfSpotsInStackVector, SpotsFiltered.count)
  push(EntireNumberOfSpotsInStackVector, EntireNumberOfSpotsInStack)
  if(PercentageCellAreaInPicture > 0.3)
    push(SpotsPerValidAreaVector, SpotsPerValidArea)
  else()
    push(SpotsPerValidAreaVector, NAN)
  end()
  if(SpotsFiltered.count > 0)
    set(PercentageOfValidSpots = (100 * (1.0 * SpotsFiltered.count)/((1.0 * SpotsFiltered.count) + (1.0 * BigCellSpots.count))))
    push(PercentageOfValidSpotsVector, PercentageOfValidSpots)
  else()
    set(PercentageOfValidSpots = 0.0)
    push(PercentageOfValidSpotsVector, PercentageOfValidSpots)
  end()
  if(PercentageOfValidSpots > 25)
    push(AverageAreaOfCellsInStackVector, leafcells.SearchRegionArea.mean)
push(AverageAreaOfCellsInStackStddevVector, leafcells.SearchRegionArea.stddev)
push(AverageNumberOfSpotsPerCellVector, leafcells.NumberOfSpots.mean)
push(AverageNumberOfSpotsPerCellStddevVector, leafcells.NumberOfSpots.stddev)
push(PercentageCellAreaInPictureVector, PercentageCellAreaInPicture)
if(all_VarsOK == 0)
  set(all_LeafCells=LeafCells)
  set(all_SpotsFiltered=SpotsFiltered)
  set(all_ObjectsType2=ObjectsType2)
  //set(all_Haustoria=Haustoria)
  set(all_Stomata=Stomata)
  set(all_Gaps=Gaps)
  set(all_VarsOK = all_VarsOK + 1)
else()
  AddObjects(LeafCells, objects=all_LeafCells, CheckOverlap=no)
  set(all_LeafCells=objects)  // Renames output from AddObjects()
  AddObjects(SpotsFiltered, objects=all_SpotsFiltered, CheckOverlap=no)
  set(all_SpotsFiltered=objects)
  AddObjects(ObjectsType2, objects=all_ObjectsType2, CheckOverlap=no)
  set(all_ObjectsType2=objects)
  //AddObjects(Haustoria, objects=all_Haustoria, CheckOverlap=no)
  //set(all_Haustoria=objects)
  AddObjects(Stomata, objects=all_Stomata, CheckOverlap=no)
  set(all_Stomata=objects)
  AddObjects(Gaps, objects=all_Gaps, CheckOverlap=no)
  set(all_Gaps=objects)
end()
```

```
    else()
      push(AverageAreaOfCellsInStackVector, NAN)
      push(AverageAreaOfCellsInStackStddevVector, NAN)
      push(AverageNumberOfSpotsPerCellVector, NAN)
      push(AverageNumberOfSpotsPerCellStddevVector, NAN)
      push(PercentageCellAreaInPictureVector, NAN)
      set(BadStacks = BadStacks + 1)
    end()
    push( StacksUsed, _StackCounter )
  end()
else()  //if single stack
  push( NumberOfCellsInStackVector, leafcells.count)
  push(NumberOfSpotsInStackVector, SpotsFiltered.count)
  push(EntireNumberOfSpotsInStackVector, EntireNumberOfSpotsInStack)
  if(PercentageCellAreaInPicture > 0.3)
    push(SpotsPerValidAreaVector, SpotsPerValidArea)
  else()
    push(SpotsPerValidAreaVector, NAN)
  end()
  if(SpotsFiltered.count > 0)
    set(PercentageOfValidSpots = (100 * (1.0 * SpotsFiltered.count)/((1.0 * SpotsFiltered.count) + (1.0 * BigCellSpots.count))))
    push(PercentageOfValidSpotsVector, PercentageOfValidSpots)
  else()
    set(PercentageOfValidSpots = 0.0)
    push(PercentageOfValidSpotsVector, PercentageOfValidSpots)
```

```
end()
if(SpotsFiltered.count > 0)
  if(PercentageOfValidSpots > 25)
    push(AverageAreaOfCellsInStackVector, leafcells.SearchRegionArea.mean)
    push(AverageAreaOfCellsInStackStddevVector, leafcells.SearchRegionArea.stddev)
    push(AverageNumberOfSpotsPerCellVector, leafcells.NumberOfSpots.mean)
    push(AverageNumberOfSpotsPerCellStddevVector, leafcells.NumberOfSpots.stddev)
    push(PercentageCellAreaInPictureVector, PercentageCellAreaInPicture)
  else()
    Set(BadStacks = BadStacks + 1)
    push(AverageAreaOfCellsInStackVector, NAN)
    push(AverageAreaOfCellsInStackStddevVector, NAN)
    push(AverageNumberOfSpotsPerCellVector, NAN)
    push(AverageNumberOfSpotsPerCellStddevVector, NAN)
    push(PercentageCellAreaInPictureVector, NAN)
  end()
else()
  push(AverageAreaOfCellsInStackVector, NAN)
  push(AverageAreaOfCellsInStackStddevVector, NAN)
  push(AverageNumberOfSpotsPerCellVector, NAN)
  push(AverageNumberOfSpotsPerCellStddevVector, NAN)
end()
set(all_LeafCells=LeafCells)
set(all_SpotsFiltered=SpotsFiltered)
set(all_ObjectsType2=ObjectsType2)
//set(all_Haustoria=Haustoria)
```

```
    set(all_Stomata=Stomata)
    set(all_Gaps=Gaps)
    push( StacksUsed, _StackCounter )
  end()  // end of single stack / whole well contraint
  // set(all_LeafCells=LeafCells) // The first evaluated field
  // set(all_SpotsFiltered=SpotsFiltered)
  // set(all_ObjectsType2=ObjectsType2)
  // //set(all_Haustoria=Haustoria)
  // set(all_Stomata=Stomata)
  // set(all_Gaps=Gaps)
  //end()    //end of loop over stacks
  delete(wholecells)
end() // end of the foreach loop over stacks
if(all_VarsOK == 0)
  set(all_LeafCells=LeafCells)
  set(all_SpotsFiltered=SpotsFiltered)
  set(all_ObjectsType2=ObjectsType2)
  //set(all_Haustoria=Haustoria)
  set(all_Stomata=Stomata)
  set(all_Gaps=Gaps)
end()



// REPORTS SPOT DETECTION  OUTPUT PARAMETERS
```

```
if (InvalidStacks<=EndStack-StartStack)
  // At least one field was analysed and we have an object list
  set(OP_NumberOfLeafCells = all_LeafCells.count)
  set(OP_NumberOfSpots = all_SpotsFiltered.count)
  //set(OP_NumberOfHaustoria = all_Haustoria.count)
  set(OP_NumberOfStomata=all_Stomata.count)
  set(OP_NumberOfGaps = all_Gaps.count)
  set(OP_SpotAverageIntensity=all_SpotsFiltered.intensity.mean)
  set(OP_SpotAverageArea=all_SpotsFiltered.area.mean)
  set(OP_SpotTotalSignal=all_SpotsFiltered.IntegratedSpotSignal.sum)
  set(OP_SpotTotalSignal_backgroundsubtracted=all_SpotsFiltered.IntegratedSpotSignal_backgroundsubtracted.sum)
  set(OP_SpotAverageLength=all_SpotsFiltered.FullLength.mean)
  set(OP_SpotAverageHalfWidth=all_SpotsFiltered.HalfWidth.mean)
  set(OP_SpotAverageWidth2LengthRatio=all_SpotsFiltered.Width2LengthRatio.mean)
  set(OP_SpotAverageWidth2LengthRatioStddev=all_SpotsFiltered.Width2LengthRatio.stddev)
  set(OP_SpotAverageRoundness=all_SpotsFiltered.RoundnessCorrected.mean)
  set(OP_SpotAverageRoundnessStddev=all_SpotsFiltered.RoundnessCorrected.stddev)
  set(OP_SpotAverageContrast=all_SpotsFiltered.Contrast.mean)
  set(OP_SpotAverageContrastStddev=all_SpotsFiltered.Contrast.stddev)
  set(OP_SpotAveragePeakIntensity=all_SpotsFiltered.PeakIntensity.mean)
  set(OP_TotalCellArea=all_LeafCells.SearchRegion_area.sum)
  set(OP_TotalCellAreaStandardDeviation=all_LeafCells.SearchRegion_area.stddev)

  if(OP_NumberOfLeafCells>0)
    set(OP_NumberOfSpotsPerCell=(1.0*OP_NumberOfSpots)/(1.0*OP_NumberOfLeafCells))
  else()
```

```
      set(OP_NumberOfSpotsPerCell=NAN)
    end()
    if(OP_TotalCellArea>0)
      set(OP_NumberOfSpotsPerArea=(1.0*OP_NumberOfSpots)/(1.0*OP_TotalCellArea))
      set(OP_SpotTotalSignalPerArea=(1.0*OP_SpotTotalSignal)/(1.0*OP_TotalCellArea))
      set(OP_SpotTotalSignalPerArea_backgroundsubtracted=(1.0*OP_SpotTotalSignal_backgroundsubtracted)/(1.0*OP_TotalCellArea))
      set(OP_AverageCellArea=(1.0*OP_TotalCellArea)/(1.0*OP_NumberOfLeafCells))
    else()
      set(OP_NumberOfSpotsPerArea=NAN)
      set(OP_SpotTotalSignalPerArea=NAN)
      set(OP_SpotTotalSignalPerArea_backgroundsubtracted=NAN)
      set(OP_AverageCellArea=NAN)
    end()

  else()
    // NO valid stacks
    set(OP_NumberOfSpots = NAN)
    set(OP_NumberOfLeafCells = NAN)
    //set(OP_NumberOfHaustoria = NAN)
    set(OP_NumberOfStomata=NAN)
    set(OP_NumberOfGaps=NAN)
    set(OP_NumberOfSpotsPerCell=NAN)

    set(OP_SpotAverageIntensity= NAN)
    set(OP_SpotAverageArea= NAN)
  set(OP_SpotTotalSignal= NAN)
  set(OP_SpotTotalSignal_backgroundsubtracted= NAN)
  set(OP_SpotAverageLength= NAN)
  set(OP_SpotAverageHalfWidth= NAN)
  set(OP_SpotAverageWidth2LengthRatio= NAN)
  set(OP_SpotAverageWidth2LengthRatioStddev= NAN)
  set(OP_SpotAverageRoundness= NAN)
  set(OP_SpotAverageRoundnessStddev= NAN)
  set(OP_SpotAverageContrast= NAN)
  set(OP_SpotAverageContrastStddev= NAN)
  set(OP_SpotAveragePeakIntensity= NAN)
  set(OP_TotalCellArea= NAN)

  set(OP_NumberOfSpotsPerArea= NAN)
  set(OP_SpotTotalSignalPerArea= NAN)
  set(OP_SpotTotalSignalPerArea_backgroundsubtracted= NAN)
end()

// REPORTS ADDITIONAL OUTPUTS
set(OP_StackCount = StackCount) // Total number of image fields
set(OP_ValidStacks = StackCount - InvalidStacks) // Number of Valid Image fields is determined

set( vectorPos = 0 )
if(all_LeafCells.count > 0)
  ForEach(1..StackCount)
    output( NumberOfCellsInStackVector[ vectorPos ], "Number of valid Cells in Stack # " & StacksUsed[ vectorPos ] )
```

```
      output( NumberOfSpotsInStackVector[ vectorPos ], "Number of valid Spots in Stack # " & StacksUsed[ vectorPos ] )
      output( EntireNumberOfSpotsInStackVector[ vectorPos ], "Number of Spots in and out of Cells in Stack # " & StacksUsed[ vectorPos ] )
      output( PercentageOfValidSpotsVector[ vectorPos ], "Percents of inner Spots in Stack # " & StacksUsed[ vectorPos ] )
      output( AverageAreaOfCellsInStackVector[ vectorPos ], "Average Area of Cells in Stack #" & StacksUsed[ vectorPos ] )
      output( AverageAreaOfCellsInStackStddevVector[ vectorPos ], "Average Area of Cells - Standard Deviation in Stack # " & StacksUsed[ vectorPos ] )
      output( PercentageCellAreaInPictureVector[ vectorPos ], "Percents of found Cell Area in Stack # " & StacksUsed[ vectorPos ] )
      output( AverageNumberOfSpotsPerCellVector[ vectorPos ], "Average Number of Spots in Cells in Stack # " & StacksUsed[ vectorPos ] )
      output( AverageNumberOfSpotsPerCellStddevVector[ vectorPos ], "Average Number of Spots in Cells - Standard Deviation in Stack # " & StacksUsed[ vectorPos ] )
      output( SpotsPerValidAreaVector[ vectorPos ], "Average Number of Spots per recognized Area in Stack # " & StacksUsed[ vectorPos ] )
      set(vectorPos = vectorPos + 1 )
    end()
  else()
    ForEach(1..StackCount)
      output(NAN,"Number of valid Cells in Stack # " & StacksUsed[ vectorPos ])
      output(NAN, "Number of valid Spots in Stack # " & StacksUsed[ vectorPos ])
      output(NAN, "Number of Spots in and out of Cells in Stack # " & StacksUsed[ vectorPos ] )
      output(NAN, "Percents of inner Spots in Stack # " & StacksUsed[ vectorPos ])
      output(NAN, "Average Area of Cells in Stack #" & StacksUsed[ vectorPos ])
      output(NAN, "Average Area of Cells - Standard Deviation in Stack # " & StacksUsed[ vectorPos ])
      output(NAN, "Percents of found Cell Area in Stack # " & StacksUsed[ vectorPos ])
      output(NAN, "Average Number of Spots in Cells in Stack # " & StacksUsed[ vectorPos ])
      output(NAN, "Average Number of Spots in Cells - Standard Deviation in Stack # " & StacksUsed[ vectorPos ])
      output(NAN, "Average Number of Spots per recognized Area in Stack # " & StacksUsed[ vectorPos ])
      set(vectorPos = vectorPos + 1 )
    end()
```

```
end()



//ForEach(NumberOfCellsInStackVector)
// output( i, "Number of Cells in Stack # " & StacksUsed[ vectorPos ] )
// set( vectorPos = vectorPos + 1 )
//end()

//set( vectorPos = 0 )
//ForEach(NumberOfSpotsInStackVector)
// output( i, "Number of Spots in Stack # " & StacksUsed[ vectorPos ] )
// set( vectorPos = vectorPos + 1 )
//end()

//set( vectorPos = 0 )
//ForEach(AverageAreaOfCellsInStackVector)
// output( i, "Average Area of Cells in Stack # " & StacksUsed[ vectorPos ] )
// set( vectorPos = vectorPos + 1 )
//end()

//set( vectorPos = 0 )
//ForEach(AverageAreaOfCellsInStackStddevVector)
// output( i, "Average Area of Cells - Standard Deviation in Stack # " & StacksUsed[ vectorPos ] )
// set( vectorPos = vectorPos + 1 )
//end()
```

```
//set( vectorPos = 0 )
//ForEach(AverageNumberOfSpotsPerCellVector)
// output( i, "Average Number of Spots in Cells in Stack # " & StacksUsed[ vectorPos ] )
// set( vectorPos = vectorPos + 1 )
//end()

//set( vectorPos = 0 )
//ForEach(AverageNumberOfSpotsPerCellStddevVector)
// output( i, "Average Number of Spots in Cells - Standard Deviation in Stack # " & StacksUsed[ vectorPos ] )
// set( vectorPos = vectorPos + 1 )
//end()

//set( vectorPos = 0 )
//ForEach(NumberOfOuterSpotsInStackVector)
// output( i, "Number of Spots out of recognized Cells in Stack # " & StacksUsed[ vectorPos ] )
// set( vectorPos = vectorPos + 1 )
//end()

//set( vectorPos = 0 )
//ForEach(PercentageOfValidSpotsVector)
// output( i, "Percents of inner Spots in Stack # " & StacksUsed[ vectorPos ] )
// set( vectorPos = vectorPos + 1 )
//end()

//set( vectorPos = 0 )
```

```
//ForEach(PercentageCellAreaInPictureVector)
// output( i, "Percents of found Cell Area in Stack # " & StacksUsed[ vectorPos ] )
// set( vectorPos = vectorPos + 1 )
//end()



output(OP_NumberOfLeafCells, "Number of Leaf Cells in whole Well")
if(all_LeafCells.count > 0)
  output(all_LeafCells.SearchRegionArea.mean, "Average Cell Area in whole Well")
  output(all_LeafCells.SearchRegionArea.stddev, "Average Cell Area in whole Well - Standard Deviation")
else()
  output(0, "Average Cell Area in whole Well")
  output(0, "Average Cell Area in whole Well - Standard Deviation")
end()
output(OP_NumberOfSpots, "Number of Spots in whole Well")
if(all_SpotsFiltered.count > 0)
  output(all_LeafCells.NumberOfSpots.mean, "Average Number of Spots per Cell in whole Well")
  output(all_LeafCells.NumberOfSpots.stddev, "Average Number of Spots per Cell in whole Well - Standard Deviation")
else()
  output(0, "Average Number of Spots per Cell in whole Well")
  output(0, "Average Number of Spots per Cell in whole Well - Standard Deviation")
end()
output(OP_TotalCellArea, "Total Cell Area in Well")
set(OP_PercentageValidStacks = (100 * (1.0 * OP_ValidStacks) / (1.0 * OP_StackCount)))
if(OP_ValidStacks > 0)
  set(PercentageOfTotalCellAreaOverAllValidStacks = (100 * (1.0 * all_LeafCells.SearchRegionArea.sum) / (1.0 * OP_ValidStacks * 357760)))
```

```
else()
  set(PercentageOfTotalCellAreaOverAllValidStacks = 0)
end()
output(PercentageOfTotalCellAreaOverAllValidStacks, "Percentage of total Cell Area in Well")

//output(OP_NumberOfHaustoria, "Number of Haustoria")
output(OP_NumberOfStomata, "Number Of Stomata")
//Output(OP_NumberOfGaps, "Number Of gaps between leaf cells")

//output(OP_NumberOfSpotsPerArea, "Number of spots per area")
//output(OP_NumberOfSpotsPerCell, "Average Number of spots per cell")

//output(OP_SpotTotalSignalPerArea, "Total integrated spot signal per area")
//output(OP_SpotTotalSignalPerArea_backgroundsubtracted, "Total integrated spot signal per area background subtracted")

output(OP_SpotAverageIntensity, "Average Intensity of Spots")
output(OP_SpotAverageArea, "Average Area of Spots")
//output(OP_SpotTotalSignal, "Total integrated spot signal, over all spots")
//output(OP_SpotTotalSignal_backgroundsubtracted, "Total integrated spot signal background subtracted, over all spots")
output(OP_SpotAverageLength, "Average Length of Spots")
output(OP_SpotAverageHalfWidth, "Average Half Width of Spots")
output(OP_SpotAverageWidth2LengthRatio, "Average Width to Length Ratio of Spots")
output(OP_SpotAverageWidth2LengthRatioStddev, "Average Width to Length Ratio of Spots - Standard Deviation")
output(OP_SpotAverageRoundness, "Average Roundness of Spots")
output(OP_SpotAverageRoundnessStddev, "Average Roundness of Spots - Standard Deviation")
output(OP_SpotAverageContrast, "Average Contrast of Spots")

output(OP_SpotAverageContrastStddev, "Average Contrast of Spots - Standard Deviation")

output(OP_SpotAveragePeakIntensity, "Average Peak Intensity of Spots")

//if(StackNo<1)

  //output(OP_AverageCellArea, "Average Cell Area")

  //output(OP_TotalCellAreaStandard Deviation, "Total Cell Area Standard Deviation")

//end()

output(OP_StackCount, "Total number of Stacks analyzed in Well")

output(OP_ValidStacks, "Number of valid Stacks in Well")

set(OP_PercentageValidStacks = (100 * (1.0 * OP_ValidStacks) / (1.0 * OP_StackCount)))

output(OP_PercentageValidStacks, "Percentage of valid Stacks in Well")
```

Supplemental Data S2. Plasma membrane microdomain script. Acapella system script for the detection of membrane microdomain spot-like features in collections of mutli-plane confocal microscopy images.

```
# name of script: plasma membrane microdomain script
# to execute this script rename .txt to .script

//***************************************************************************

//  File Name:      MPI_leaves_e_4_AK2008-06-16.script

//  Purpose: Detects spot-like features on leaf cells images, images from MPI-Köln

//

//                  EVOTEC  TECHNOLOGIES

//                  Olavi Ollikainen

//  e-mail:     Olavi.Ollikainen@evotec-technologies.com

// phone :      00 372 52 75 262

//   With corrections from Kurt Stüber MPI Köln

//  phone:      +49 (0) 221 5062 120

//  e-mail       stueber@mpiz-koeln.mpg.de

//***************************************************************************

//

//          Output parameters:

//                          * Number of Spots

//                          * Number of Leaf Cells

//                          * Number of Haustoria

//                          * Number Of Stomata

//                          * Number Of gaps between leaf cells

//                          * Number of spots per area (per cell area)

//                          * Number of spots per cell

//                          * Total integrated spot signal per area
//                      * Total integrated spot signal per area background subtracted
//                      * Average intensity of spots
//                      * Average area of spots
//                      * Total integrated spot signal, over all spots
//                      * Total integrated spot signal background subtracted, over all spots
//                      * Average length of spots
//                      * Average Half width of spots
//                      * Average width to length ratio of spots
//                      * Average roundness of spots
//                      * Average contrast of spots
//                      * Average PeakIntensity of spots
//***************************************************************************


// ***************************************************************************
// Comment:
// changed dublicated procedure definitions ( calcfillstencil() and ReduceStencilObjectsByAreaTo31999() )
// to script specific names with suffix "_MPI"
// June 20, 2008
//     a. Procedure ReduceStencilObjectsByArea_MPI() was introduced (line 497) with an aim to avoid error in CalcSkeleton() module (line 507)
//     b. Script syntax was changed so that useless warning do not appear
// ***************************************************************************



/// Spot detection procedure. Finds spots on SpotImage. Spot candidates are detected as local intensity maximums. Thereupon the spots are selected by contrast and intensity parameters.
/// formerly called Spot_Detection_C
```

```
proc Spot_Detection_MPIZ(

  image SpotImage in "Input image with intensity information. Spots are detected by this image.",

  string SearchRegion="SearchRegion" in "Name of the attribute in input list WholeCells, which specifies the regions where spots are searched. In case of the empty string \"\" spots are searched over the whole SpotImage and the input object list is ignored.",

  bool ShowIllustrations=YES in "YES- Output illustrations are depicted. No- Output illustrations are not shown.",

  bool ShowSearchRegionBorder=NO in "YES- Output illustration with SearchRegion borders is depicted. No- Output illustration with SearchRegion borders is not shown.",

  bool ShowOutputParameters=YES in "YES- Output parameters are reported. No- Output Parameters are not reported.",



  objectlist WholeCells=none inout "Optional input object list, which defines the objects (e.g. ~cells) where the spots are searched. Output list contains numerical and geometrical spot attributes. Input list should involve the stencil-type attribute specified by the input SearchRegion. If the object list is not provided spots are searched over the whole image.",

  objectlist SpotCandidates out "Output object list of spot candidates with calculated numerical and geometrical attributes.",

  objectlist Spots out "Output object list of classified spots with calculated numerical and geometrical attributes.",



  double NumberOfSpotCandidates out "Number of spot candidates.",

  double NumberOfSpots out "Number of detected spots.",

  double SpotsPerObject out "Number of spots per object (i.e. number of spots per ~cell).",

  double SpotsPerArea out "Number of spots per SearchRegion area (i.e. number of spots per visible ~cell area).",

  double IntegratedSpotSignalPerCellularSignal out "Integrated spot signal over all spots normalized by integrated ~cellular signal (total signal over all SearchRegion area).",

  double IntegratedSpotSignalPerCellularSignal_BackgroundSubtracted out "Integrated spot signal over all spots background subtracted and normalized by integrated ~cellular signal (total signal over all SearchRegion area).",



  double IntegratedSpotSignalPerArea out "Integrated spot signal per SearchRegion area (per visible ~cell area).",

  double IntegratedSpotSignalPerArea_BackgroundSubtracted out "Integrated spot signal BackgroundSubtracted per SearchRegion area (per visible ~cell area)."



) spot detection, object recognition "Spot detection procedure. Finds spots on SpotImage. Spot candidates are detected as local intensity maximums. Thereupon the spots are selected by contrast and intensity parameters. Proper spot detection input parameters can be found with the template script spot_detection_parameter_scanner.script (spot_detection_parameter_scanner_Acapella10.script for Acaplella 1.0). Opera multiple field spot images can be evaluated based on the template script Opera_multifields_spots.script. See more in Opera spot detection manual."

{
  input(SpotMinimumDistance,3.0 , "SpotMinimumDistance","d", "Minimum allowed distance between two spot centers. Unit image pixel. Typical range 2.0 .. 5.0. Adjust by Illustration SpotSelection and Spots.")

  input(SpotPeakRadius, 1.0, "SpotPeakRadius","d", "Radius of the disk, where the spot peak intensity is calculated. Default value 0 means that peak intensity corresponds to the intensity of the maximum point, value 1 means that peak intensity is found as average intensity over the region with radius 1 pixel around the maximum point, i.e. over the region with area 5 pixels. Typically 0 or 1.")

  input(SpotReferenceRadius, 8.0,  "SpotReferenceRadius", "d", "Radius of Reference Region around the intensity maximum, i.e. around spot center. Typical range  2.0 .. 5.0.")

  input(SpotMinimumContrast,0.4, "SpotMinimumContrast", "d","Minimum allowed contrast between spot peak intensity and the reference intensity. The main spot selection parameter. Range from 0..1. Adjust parameter by illustrations SpotSelection and Spots. If the parameter value is lowered the number of classified spots increases and vice versa.")

  input(SpotMinimumToCellIntensity, 1.0, "SpotMinimumToCellIntensity","d","Minimum allowed intensity ratio between Spot Peak Intensnity and the Average Intensity of the cell/object to which the spot is belongs. Range from 0..oo. Has smaller influence on the outputs than the main selection parameter SpotMinimumContrast. Adjust parameter by illustrations SpotSelection and Spots. If the parameter value is lowered the number of classified spots increases and vice versa.")



  if( SearchRegion!="" and defined("WholeCells"))

    set(Objects_in=WholeCells)

    if(!defined("WholeCells." & SearchRegion))

      error("Procedure Spots_Detection_C() input attribute SearchRegion " & SearchRegion & " does not exist. The input SearchRegion should correspond to the stencyl-type attribute in the input object list or to be an empty string \"\". The empty string means that spots are searched over the whole image and the input object list is ignored.")

    else()

      eval("set(SearchMask=WholeCells." & SearchRegion & ")")

      if (SearchMask.class!="intervalvector")

        error("Procedure Spots_Detection_C() input attribute SearchRegion " & SearchRegion & " is not a stencil. The input SearchRegion should correspond to the stencyl-type attribute in the input object list or to be an empty string \"\". The empty string means that spots are searched over the whole image and the input object list is ignored.")

      end()

    end()

    set(SearchObjects=WholeCells)

  end()



  if (ShowIllustrations==Yes)

    spot_illustrations_1()

  end()
  Spot_Detection_C_inner() // Spot detection


  if(ShowIllustrations)

    spot_illustrations_2()

    set(printf_text="Spot detection C. Number of Spot Candidates: " & SpotCandidates.count & "; Number Classified Spots: " & spots.count & "; Discarded by Contrast: " & NumberDiscradedByContrast & "; Discarded by SpotToCellIntensity: " & NumberDiscradedBySpotToCellIntensity & "\n")

    printf(printf_text)

  end()

  set(NumberOfSpotCandidates=SpotCandidates.count)

  set(NumberOfSpots=Spots.count)

  if(ShowOutputParameters)

    create_spot_outputs()

  end()

  set(printf_text="Spot detection C. Number of Spot Candidates: " & NumberOfSpotCandidates & "; Number Classified Spots: " & NumberOfSpots & "; Spots per cell/object: " & SpotsPerObject & "; Spot per area: " & SpotsPerArea & "; Integrated Spot Signal Per Cellular Signal: " & IntegratedSpotSignalPerCellularSignal & "\n")

  printf(printf_text)

}




/////////////////////////////////////////////////////////// PROCEDURES

proc CalcNodesLarge( string SkeletonName="skeleton" in "Stencil in input list,  Nodes are found for this stencil",

objectlist objects inout "Input-output object list") "Adds to object list a stencil-type attribute LargeNodes"

{

  if(!defined("objects." & SkeletonName))

    error("Error. Input stencil " & SkeletonName & " is not defined. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")

  else()
    eval("set(IfIntervalVector=Objects." & SkeletonName & ".class==\"intervalvector\")")

    if(!IfIntervalVector)

      error("Error. The attribute  " & SkeletonName & " given by the input SkeletonName is not a stencil. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")

    end()

  end()

  set(ob_in=objects)

  eval("setattr(stencil_temp,objects." & SkeletonName & ")")


  set(IM_skeleton=objects.stencil_temp.mask.image)


  blank(3,3,1)

  set(kernel0=image)

  set(kernel0[0,0]=0, kernel0[2,2]=0,  kernel0[0,2]=0, kernel0[2,0]=0)

  set(kernel0.type="mask")

  set(kernel1=kernel0)

  set(kernel1[1,0]=0)

  convolution(image=IM_skeleton,mask=objects.stencil_temp.mask, faster=yes, convolutionkernel=kernel1.vector)

  set(image.factor=1)

  mask(4,image=image)

  set(M_nodeslarge1=mask.image)


  set(kernel2=kernel0)

  set(kernel2[2,1]=0)

  convolution(image=IM_skeleton,mask=objects.stencil_temp.mask, faster=yes, convolutionkernel=kernel2.vector)

  set(image.factor=1)

  mask(4,image=image)
```

```
  set(M_nodeslarge2=mask.vector)


  set(kernel3=kernel0)
  set(kernel3[1,2]=0)
  convolution(image=IM_skeleton,mask=objects.stencil_temp.mask, faster=yes, convolutionkernel=kernel3.vector)
  set(image.factor=1)
  mask(4,image=image)
  set(M_nodeslarge3=mask.vector)


  set(kernel4=kernel0)
  set(kernel4[0,1]=0)
  convolution(image=IM_skeleton,mask=objects.stencil_temp.mask, faster=yes, convolutionkernel=kernel4.vector)
  set(image.factor=1)
  mask(4,image=image)
  set(M_nodeslarge4=mask.vector)


  carrypixels(image=M_nodeslarge1,mask=M_nodeslarge2,data=1)
  carrypixels(image=image,mask=M_nodeslarge3,data=1)
  carrypixels(image=image,mask=M_nodeslarge4,data=1)


  calcnodes(stencil_temp)
  carrypixels(image=image,mask=objects.stencil_temp_nodes,data=1)
  and(image=objects.stencil_temp.image, mask=image)
  eval("setattr(" & SkeletonName & "_LargeNodes,image.vector, objects=ob_in)")
}
```

```
/////////////////////////////////////////////////////////////////////////////////////
proc CalcSkeletonBranches(
string SkeletonName="skeleton" in "Stencil-type attribute on which the  skeleton branches are found.",
string DeadEndStencil="deadend" in "Stencil-type attribute, which corresponds to DeadEnd tips",
objectlist objects inout "Input-output object list with the stencil-type attribute on which the branches are found. Input list must include also a stencil, which coresponds to dead end tips.",
objectlist Branches out "Output object list of Branch lines on the skeleton between nodes.  "
) "Finds skeleton branches between nodes. Outputs object list of skeleton branches"
{
  set(ob_in=objects)
  if(!defined("objects." & SkeletonName))
    error("Error. Input stencil " & SkeletonName & " is not defined. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
  else()
    eval("set(IfIntervalVector=Objects." & SkeletonName & ".class==\"intervalvector\")")
    if(!IfIntervalVector)
      error("Error. The attribute  " & SkeletonName & " given by the input SkeletonName is not a stencil. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
    end()
  end()

  if(!defined("objects." & DeadEndStencil))
    error("Error. Input stencil " & DeadEndStencil & " is not defined. Input DeadEndStencil should correspond to the existing stencil-type attribute in the input object list.")
  else()
    eval("set(IfIntervalVector=Objects." & DeadEndStencil & ".class==\"intervalvector\")")
    if(!IfIntervalVector)
      error("Error. The attribute  " & DeadEndStencil & " given by the input DeadEndStencil is not a stencil. Input DeadEndStencil should correspond to the existing stencil-type attribute in the input object list.")
  end()
end()

eval("setattr(skeleton,ob_in." & SkeletonName & ")")
eval("setattr(DeadEnd,ob_in." & DeadEndStencil & ")")

calcnodeslarge(SkeletonName="skeleton")

convolutionmask("disk",1)
convolution(image=objects.skeleton_largenodes.mask.image, mask=objects.skeleton.mask,faster=yes)
set(image.factor=1)
mask(1)
set(nodes_surrounding=mask)
set(ob2=objects)
carrypixels(image=objects.skeleton.mask.image,mask=nodes_surrounding.vector,data=0)
set(M_connections=image)

mask2stencil(M_connections, Neighbourhood=8)
stencil2objects()

and(image=objects.body.image, mask=ob2.deadend.mask.image)
setattr(deadend, image.vector)
calcarea()
calcarea(deadend)
carrypixels(image=ob2.skeleton.image, mask=ob2.skeleton_largenodes, data=0)
```

```
  calczone(1, stencil=image.vector)
  zonemask(-1,oo)
  renameattr(Branches=zonemask)
  calcarea(Branches)
  Set(Branches=objects)
  and(image=ob2.skeleton.image,mask=Branches.body.mask.image)
  Setattr(Branches,image.vector,objects=ob_in)
  Setattr(Nodes,ob2.skeleton_LargeNodes)
  Setattr(DeadEnd,ob2.skeleton_DeadEnd)
  Setattr(DeadEnd_type1,ob2.skeleton_DeadEnd_type1)
  Setattr(DeadEnd_type2,ob2.skeleton_DeadEnd_type2)
}


proc CalcDeadEndTypes(
string SkeletonName="skeleton" in "Stencil-type attribute, which corresponds to skeleton",
objectlist objects inout "Input-output object list"
) "Finds the deadend tips in the skeleton"
{
  set(ob_in=objects)
  if(!defined("objects." & SkeletonName))
    error("Error. Input stencil " & SkeletonName & " is not defined. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
  else()
    eval("set(IfIntervalVector=Objects." & SkeletonName & ".class==\"intervalvector\")")
    if(!IfIntervalVector)
      error("Error. The attribute  " & SkeletonName & " given by the input SkeletonName is not a stencil. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
  end()
end()
eval("setattr(skeleton,ob_in." & SkeletonName & ")")

set(IM_skeleton=objects.skeleton.mask.image)
not(image=IM_skeleton)
set(IM_skeleton_not=image)
//set(IM_skeleton_vec=objects.skeleton_skeleton.mask)
//calcnodes(skeleton)
CalcNodesLarge(SkeletonName="skeleton")

convolutionmask("ribbon",1, 1)
set(ConvolutionKernel_4n=ConvolutionKernel)
convolution(image=objects.skeleton_largenodes.mask.image,mask=objects.skeleton, faster=yes)
set(image.factor=1)
mask(1,image=image)
set(M_nodes_4n=mask.vector)

convolutionmask("ribbon",1.5, 1)
set(ConvolutionKernel_8n=ConvolutionKernel)

convolution(image=IM_skeleton,mask=objects.skeleton, faster=yes)
set(image.factor=1)
set(IM_convolution_8n=image)
mask(1,image=image)
```

```
set(M1=mask.image)

mask(2,image=image)

set(M2=mask.image)


carrypixels(image=m1,mask=M2.vector,data=0)

set(M_deadend0=image.vector)


convolution(image=objects.skeleton_largenodes.mask.image,mask=M_deadend0, faster=yes)

set(image.factor=1)

mask(1,image=image)

set(M_deadend_type1=mask.vector)


//carrypixels(image=M_deadend0.image,mask=M_deadend_type1.vector,data=0)

//set(M_deadend0=image.vector)

////////////////////////////////////////////////////////////////////

blank(3,3,1)

set(IM1=image)

set(IM1.type="mask")

carrypixels(image=M_nodes_4n.image, mask=M_deadend0,data=0)

set(M_nodes_4n_mod=image.vector)


set(kernel1=IM1)

set(kernel1[2,0]=0,kernel1[2,1]=0, kernel1[2,2]=0, kernel1[1,1]=0)

convolution(image=IM_skeleton_not,mask=M_nodes_4n_mod, faster=yes, convolutionkernel=kernel1.vector)

set(image.factor=1)
```

```
mask(5,image=image)

set(M_deadend1=mask.image)


set(kernel2=IM1)

set(kernel2[0,0]=0,kernel2[0,1]=0, kernel2[0,2]=0, kernel2[1,1]=0)

convolution(image=IM_skeleton_not,mask=M_nodes_4n_mod, faster=yes, convolutionkernel=kernel2.vector)

set(image.factor=1)

mask(5,image=image)

set(M_deadend2=mask.image)


set(kernel3=IM1)

set(kernel3[0,0]=0,kernel3[1,0]=0, kernel3[2,0]=0, kernel3[1,1]=0)

convolution(image=IM_skeleton_not,mask=M_nodes_4n_mod, faster=yes, convolutionkernel=kernel3.vector)

set(image.factor=1)

mask(5,image=image)

set(M_deadend3=mask.image)


set(kernel4=IM1)

set(kernel4[0,2]=0,kernel4[1,2]=0, kernel4[2,2]=0, kernel4[1,1]=0)

convolution(image=IM_skeleton_not,mask=M_nodes_4n_mod, faster=yes, convolutionkernel=kernel4.vector)

set(image.factor=1)

mask(5,image=image)

set(M_deadend4=mask.image)


or(image=M_deadend1,mask=M_deadend2)
```

```
  or(image=image,mask=M_deadend3)

  or(image=image ,mask=M_deadend4)

  or(image=image,mask=M_deadend_type1.image)

  set(M_deadend_type1=image)

  carrypixels(image=M_deadend0.image, mask=M_deadend1.vector, data=0)

  set(M_deadend_type2=image)

  or(image=M_deadend_type1.image, mask=M_deadend_type2.image)

  set(M_deadend=image)

  set(ob2=objects)


  eval("setattr(" & SkeletonName & "_LargeNodes,ob2.skeleton_LargeNodes, objects=ob_in)")


  and(image=ob2.skeleton.image, mask=M_deadend.image)

  eval("setattr(" & SkeletonName & "_DeadEnd,image.vector)")


  and(image=ob2.skeleton.image, mask=M_deadend_type1.image)

  eval("setattr(" & SkeletonName & "_DeadEnd_type1,image.vector)")


  and(image=ob2.skeleton.image, mask=M_deadend_type2.image)

  eval("setattr(" & SkeletonName & "_DeadEnd_type2,image.vector)")

}



proc RemoveSkeletonLayer(

string SkeletonName="skeleton" in "Stencil-type attribute, which corresponds to skeleton",

int MinimumArea=20 in "DeadEnd skeleton branches with area less than the limit are discarded",
objectlist objects inout "Input-output object list"
) "Discards from skeleton the DeadEnd branches with area less than the limit MinimumArea"
{
  set(ob_in=objects)
  if(!defined("objects." & SkeletonName))
    error("Error. Input stencil " & SkeletonName & " is not defined. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
  else()
    eval("set(IfIntervalVector=Objects." & SkeletonName & ".class==\"intervalvector\")")
    if(!IfIntervalVector)
      error("Error. The attribute  " & SkeletonName & " given by the input SkeletonName is not a stencil. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
    end()
  end()
  eval("setattr(skeleton,ob_in." & SkeletonName & ")")

  CalcDeadEndTypes(SkeletonName="skeleton")
  carrypixels(image=objects.skeleton.image,mask=objects.skeleton_DeadEnd_type1.mask,data=0)
  setattr(skeleton_mod,image.vector)

  ////////////////////////////////////////////////////////////////////////////////////////////////
  set(ob2=objects)
  CalcSkeletonBranches(SkeletonName="skeleton_mod", deadendstencil="skeleton_DeadEnd_type2")
  ObjectFilter(MinimumArea>Branches_area and deadend_area>0, objects=Branches)
  carrypixels(image=ob2.skeleton_mod.image,mask=objects.Branches.mask,data=0)
  set(BranchesRemoved=objects)
  setattr(skeleton_LayerRemoved,image.vector, objects=ob2)
  and(image=ob_in.body.image, mask=BranchesRemoved.Branches.mask.image)
  setattr(skeleton_RemovedBranches,image.vector)
  set(ob3=objects)

  eval("setattr(" & SkeletonName & "_LargeNodes,ob3.skeleton_LargeNodes, objects=ob_in)")
  eval("setattr(" & SkeletonName & "_DeadEnd, ob3.skeleton_DeadEnd)")
  eval("setattr(" & SkeletonName & "_DeadEnd_type1, ob3.skeleton_DeadEnd_type1)")
  eval("setattr(" & SkeletonName & "_DeadEnd_type2, ob3.skeleton_DeadEnd_type2)")
  eval("setattr(" & SkeletonName & "_LayerRemoved,  ob3.skeleton_LayerRemoved)")
  eval("setattr(" & SkeletonName & "_RemovedBranches,  ob3.skeleton_RemovedBranches)")
}



proc SkeletonBranchesNodes(
string SkeletonName="skeleton" in "Stencil-type attribute, which corresponds to skeleton",
objectlist objects inout "Input-output object list",
objectlist Branches out "Output object list of skeleton branches",
objectlist Nodes out "Output object list of nodes"
//int MinimumArea=20 in "DeadEnd skeleton branches with area less than the limit are discarded"
) "Finds Brances and Nodes of skeleton. Outputs object lists of the found Branches and Nodes. In addition adds to the input list attributes branches and nodes."
{
  set(ob_in=objects)
  if(!defined("objects." & SkeletonName))
    error("Error. Input stencil " & SkeletonName & " is not defined. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
  else()
    eval("set(IfIntervalVector=Objects." & SkeletonName & ".class==\"intervalvector\")")
    if(!IfIntervalVector)
      error("Error. The attribute  " & SkeletonName & " given by the input SkeletonName is not a stencil. Input SkeletonName should correspond to the existing stencil-type attribute in the input object list.")
    end()
  end()
  eval("setattr(skeleton,ob_in." & SkeletonName & ")")

  CalcDeadEndTypes(SkeletonName="skeleton")
  carrypixels(image=objects.skeleton.image,mask=objects.skeleton_DeadEnd_type1.mask,data=0)
  setattr(skeleton_mod,image.vector)

  ////////////////////////////////////////////////////////////////////////////////////////////////
  set(ob2=objects)
  CalcSkeletonBranches(SkeletonName="skeleton_mod", deadendstencil="skeleton_DeadEnd_type2")
  set(ob2=objects)
  mask2stencil(objects.nodes)
  stencil2objects()
  set(Nodes=objects)
  eval("setattr(" & SkeletonName & "_Nodes,ob2.Nodes, objects=ob_in)")
  eval("setattr(" & SkeletonName & "_Branches,ob2.Branches)")
  eval("setattr(" & SkeletonName & "_DeadEnd,ob2.DeadEnd)")
  eval("setattr(" & SkeletonName & "_DeadEnd_type1,ob2.DeadEnd_type1)")
  eval("setattr(" & SkeletonName & "_DeadEnd_type2,ob2.DeadEnd_type2)")
}
```

```
/////////////////////// Fills small holes
proc FillSmallHoles(
int minimumholearea=20 in "Minimum area for holes. Holes with area less than the limit are filled",
objectlist objects inout "Input-output object list. Attributes in the input list are lost."
) "Fills small holes in objects. Holes with area less than the limit MinimumHoleArea are filled."
{
  set(ob_in=objects)
  fillobjects()
  set(ob01=objects)
  carrypixels(image=ob01.body.image,mask=ob_in.body.mask,data=0)
  stencil2mask(image.vector)
  mask2stencil()
  stencil2objects()
  calcarea()
  objectfilter(minimumholearea<=area)

  carrypixels(image=ob01.body.image,mask=objects.body.mask,data=0)
  stencil2objects(image.vector)
  xor(image=objects.body.image,mask=ob_in.body.mask.image)
  setattr(filled, image.vector)
}


proc ImageSubRegions(
int ImageWidth in,
int ImageHeight in,
int NumberOfLinearDivision in,

objectList objects out

)

{

  set(w0=int((1.0*ImageWidth)/(2.0*NumberOfLinearDivision)))

  set(h0=int((1.0*ImageHeight)/(2.0*NumberOfLinearDivision)))

  set(w1=2*w0, h1=2*h0)

  blank(ImageWidth, ImageHeight)

  set(image.type="mask")


  foreach(0..NumberOfLinearDivision-1  )

    foreach(0..NumberOfLinearDivision-1,"j")

      set(image[w0+i*w1, h0+j*h1]=1)

    end()

  end()

  mask2stencil(image)

  stencil2objects()

  set(eros_dist=w0+h0)

  calcerosion(-eros_dist)

}


proc DataCubeProjectionCorrection(

int FirstZPlane in,

int LastZPlane in,

datacube datacube inout,
image IM_planeNumber out,
image PlaneNUmberImage out
) [hidden]
{
  ////// Projects maximum intensity for each x-y position from datacube to image
  Maximums3D(0,image=datacube)
  StencilFrom3DTo2D(stencil=maximums,datacube=datacube) //Projects the found maximums from 3-dim to plane
  set(IM_plane=PlaneNumberImage)
  //delete(datacube)
  convolutionmask("ribbon",7.9,7)

  convolution(image=valueimage)
  minus(valueimage,image,neg_method="zero", result_type="unsigned,short")
  set(r4=result)
  mask(1, image=result,userealvalues=no)
  set(M_bright2=mask)

  mask(result.mean, image=r4)

  and(image=M_bright2.image, mask=mask.image)
  set(M_bright3=image)
  eval("WarningFilter(disable=\"[Mask2Stencil]\")")
  mask2stencil(M_bright3)
  ReduceStencilObjectsByArea_MPI(stencil,10,ObjectNumberLimit=30000,faster=1)
  eval("WarningFilter(enable=\"[Mask2Stencil]\")")
  stencil2objects()
```

```
////////////////////////////////
calcarea()


//////////////////////////////
CalcSkeletonByIntensity(image=r4)
CalcSkeletonByIntensity(skeleton, image=r4, IntensityEvalParam=-10)
CalcSkeletonByIntensity(skeleton_skeleton, image=r4, IntensityEvaluationMode=2, IntensityEvalParam=-4)
CalcSkeletonByIntensity(skeleton_skeleton_skeleton, image=r4, IntensityEvaluationMode=2, IntensityEvalParam=-20)
CalcSkeleton(skeleton_skeleton_skeleton_skeleton)
RenameAttr(Skeleton=skeleton_skeleton_skeleton_skeleton)
CalcArea(Skeleton)
set(Skeleton=objects)
ObjectFilter(Skeleton_area>10)
SkeletonBranchesNodes(skeletonName="skeleton")


Set(OL4=objects)
set(objects=branches)
ObjectFilter(area>10)
Set(OL7=objects)


ImageSubRegions(r4.width, r4.height,4)


And(Image=objects.eroded.image, mask=OL7.body.mask.image)
clearborders(image,9)
SetAttr(skeleton, stencil.vector)
```

```
CalcStat("quantile",0.7, stencil=skeleton, image=r4)

ThreshMask(stencil=skeleton,threshold=quantile,image=r4)

CalcStat("median",stencil=threshmask, image=PlaneNumberImage)

//CalcStat("Median",stencil=skeleton, image=PlaneNumberImage)

CarryObjects(image=objects.eroded.image, stencil=objects.eroded, data=objects.median)

//

//enlarge(1,image=image) // AK 5.12.2007

redimension(image.width + 2, image.height + 2, 1, 1) // AK 5.12.2007

stencil2objects(image)

calcerosion(-2)


set(temp=objects.eroded.image)

crop(1,1,temp.width-1, temp.height-1,image=temp)

mean(31,image=image)

set(IM_PlaneNumber=Image)


delete(objects)

set(objects=OL7)

calcIntensity(body,image=r4)

calcstat("max",stencil=body,image=r4)

calcstat("Quantile",0.5,stencil=body,image=r4)


Threshmask(threshold=quantile,stencil=body,image=r4)

Calcintensity(Threshmask,image=r4)

//Calcstat("mean",stencil=threshmask,attrname="planeNumberMean", image=PlaneNumberImage)

Calcstat("median",stencil=body,attrname="planeNumberMean", image=PlaneNumberImage)
```

```
Calcerosion(-3)


CarryObjects(image=objects.eroded.image, stencil=objects.eroded, data=objects.max)

set(IM_intensity=Image)

maximums(20, mask=M_bright3.vector,image=r4)

set(max1=maximums)

maximums(70, mask=M_bright3.vector,image=r4)

set(max2=maximums)

or(image=max1.mask.image,mask=max2.mask.image)

and(image=objects.eroded.image,mask=image)

setattr(max1,image.vector)

calcarea(max1)


objectfilter(max1_area>0)

Calcerosion(-100,eroded,numberofsteps=20)

CarryObjects(image=objects.eroded.image, stencil=objects.eroded_eroded, data=objects.planeNumberMean)

set(IM_intensity2=Image)

//CarryObjects(image=objects.eroded.image, stencil=objects.eroded_eroded, data=objects.max)

//set(IM_max=Image)

//CarryObjects(image=objects.eroded_eroded.image, stencil=objects.eroded_eroded, data=objects.planeNumberMean)

//set(IM_planeNumberMean=Image)

set(OL5=objects)



and(image=objects.body.image,mask=max2.mask.image)
```

```
  setattr(max2,image.vector)
  calcarea(max2)
  objectfilter(max2_area>0)
  calcerosion(-10,max2,restrictivestencil=M_bright3,numberofsteps=2)
  and(image=objects.max2_eroded.image,mask=skeleton.skeleton.mask.image)
  setattr(skeleton1,image.vector)
  Calcintensity(skeleton1,image=PlaneNumberImage)
  CalcAttr(planenumber,round(skeleton1_intensity))
  //CalcAttr(planenumber,iif(planenumber>10, planenumber-10, 1))
  calcerosion(-100,max2_eroded,restrictivestencil=M_bright3,numberofsteps=20)
  calcerosion(-200,max2_eroded_eroded,numberofsteps=20)
  //CarryObjects(image=objects.max2_eroded_eroded_eroded.image, stencil=objects.max2_eroded_eroded_eroded, data=objects.planenumber)
  //set(IM_planeNumberMean=Image)



  set(i=FirstZplane, i2=0)
  foreach(FirstZplane..LastZplane)
    set(threshold2=i2+1)
    minus(threshold2,IM_planeNumber,neg_method="abs",result_type="unsigned,short")
    mask(threshold=8,image=result)
    carrypixels(image=DataCube[i2],mask=mask.vector,data=0)
    set(DataCube[i2]=image)
    set(i2=i2+1)
  end()

}
```

```
proc ReduceStencilObjectsByAreaTo31999_MPI(
stencil stencil inout "Input-output stencil, which object number will be reduced to fit the limit 31999. Only the first 31999 objects are taken into account and the others are discarded."
) [hidden]

{
  set(alimit=32000)
  set(alimit2=alimit-1)
  eval("WarningFilter(disable=\"[Mask2Stencil]\")")
  Removes_OnePixel_Objects(stencil.vector.MASK)
  mask2stencil()
  if(stencil.itemcount<alimit)
    return()
  end()

  set(mask_in=mask)
  set(temp=stencil.vector.mask.image)
  blank(temp.width, temp.height)
  convelems(image, "integer", 2,"unsigned")

  append(bvector,1..alimit2)
  set(avector=bvector)
  set(stencil_in=stencil)
```

```
while(stencil.itemcount>alimit2)
  set(stencil2=stencil)
  create("vector","unsigned,int", stencil.itemcount-alimit2,0)
  append(avector,vector)
  carryobjects(image=result,stencil=stencil,data=avector)
  set(image.type="stencil")
  set(stencil=image.vector)
  stencil2objects()
  calcarea()
  if(defined("v_area"))
    append(v_area,objects.area)
  else()
    set(v_area=objects.area)
  end()
  push(v_objects,objects)
  carrypixels(image=stencil2.image,mask=objects.body.mask, data=0)
  mask2stencil(image.vector.mask)

end()
stencil2objects()
calcarea()

append(v_area,objects.area)
sort(v_area,yes)
set(minarea=result[alimit2]+1)
objectfilter(area<minarea)
```

```
  carrypixels(image=mask_in,mask=objects.body,data=0)
  foreach(v_objects)
    set(objects=i)
    objectfilter(area<minarea)
    carrypixels(image=image,mask=objects.body,data=0)
  end()
  mask2stencil(image)
  if(stencil.itemcount>31999)
    ReduceStencilObjectsTo31999_tech()
  end()
}

proc CreateDataCubeWithControl(
table sourcedata inout,
int FirstZPlane in,
int LastZPlane in,
int MinStdDev in,
DataCube DataCube out,
int newFirstZplane out,
int newLastZplane out
)
{
  set( logfilename = "D:/dmeyer/test/filenames.txt" )
  set( Zplane = FirstZplane )
  //write( "flex\t" & imagefilename1, logfilename, "ascii", append=true )
```

```
//write( "\tFirstZplane\t" & FirstZplane, logfilename, "ascii", append=true )

//write( FirstZplane, logfilename, "ascii", append=true )

//write( "\tLastZplane\t" & LastZplane, logfilename, "ascii", append=true )

//write( LastZplane, logfilename, "ascii", append=true )




////// Creates a datacube

set(CubeDepth=LastZplane-FirstZplane+1)

create("datacube",sourcedata.SourceImage[0].width,sourcedata.sourceImage[0].height,CubeDepth, "unsigned short")

set(FirstZplane_in=FirstZplane, LastZplane_in=LastZplane)

set(i=FirstZplane, i2=0)

// find new FirstZplane and LastZplane discarding featureless images:

set( is_empty="true")

set( newFirstZplane = FirstZplane)

foreach(FirstZplane..LastZplane)

  if( sourcedata.sourceImage[i-1].stddev < MinStdDev )

    if( is_empty == "true")

      set( newFirstZplane = i )

    end()

  else()

    set( is_empty= "false" )

  end()

end()

set( is_empty="true")

set( i=LastZplane)

set( newLastZplane = LastZplane)
```

```
foreach(LastZplane..FirstZplane)
  if( sourcedata.sourceImage[i-1].stddev < MinStdDev )
    if( is_empty == "true")
      set( newLastZplane = i )
    end()
  else()
    set( is_empty= "false" )
  end()
end()
set( FirstZplane = newFirstZplane)
set( LastZplane = newLastZplane)

if( LastZplane <FirstZplane )
  set(LastZPlane=FirstZPlane)
  return()
  //stop(1, "Stack out of focus, program execution stopped" )
end()

//write( "\tnew FirstZplane = " & FirstZplane, logfilename, "ascii", append=true)
//write( "\tnew LastZplane = " & LastZplane, logfilename, "ascii", append=true)

set(i=FirstZplane, i2=0)
foreach(FirstZplane..LastZplane)
  set(datacube[i2]=sourcedata.sourceImage[i-1])
  set(i2=i2+1)
  end()
  foreach(FirstZplane_in..LastZplane_in)
    delete(sourcedata.sourceImage[i-1])
  end()
}



proc LeafCellsDetection(
image image1 in "Input image with intensity information. Leaf cells are detected by this image",
bool showillustrations=yes in,
ObjectList LeafCells out "output list of leaf cells",
objectlist Lines out "List of line structure",
)
{
  thresholdxx(4,image=Image1)
  mean(image=image1)
  mask(threshold,image=image)
  set(M_bg=mask)
  blank(image1.width, image1.height,1)

  set(image.type="mask")
  carrypixels(image=image,mask=M_bg.vector,data=0)
  set(M_bg2=image)

  blank(image1.width, image1.height,0)
```

```
set(image.type="mask")

set(IM_blank=image)

set(image[1,1]=1, image[image1.width-2,1]=1, image[image1.width-2,image1.height-2]=1, image[1,image1.height-2]=1)


mask2stencil(image)


stencil2objects()

calczone(50)

zonemask(-50,oo)

and(image=objects.zonemask.image,mask=M_bg2)

setattr(M_bg3,image.vector)

calcarea(M_bg3)

objectfilter(M_bg3_area>800)

set(ErosionDistance=sqrt(image.width*image.width+image.height*image.height))

set(NUmberOfSteps=round(ErosionDistance/3)-1)

calczone(ErosionDistance,stencil=M_bg2)

zonemask(-ErosionDistance)

calcfillstencil_MPI(stencilname="zonemask")

renameattr(zonemask=zonemask_filled)

calcarea(zonemask)

objectfilter(zonemask_area>10000)


set(OL_erase=objects)


////// Creates initial mask of lines
```

```
Bright_mask(image1,8)

mask2stencil(M_bright)//, Neighbourhood=8)

if(stencil.itemcount>31999)

  ReduceStencilObjectsByAreaTo31999_MPI()

end()

stencil2objects()


calcarea()

set(ob1=objects)


set(ob00=objects)


FillSmallHoles()

///////////////////////////////////////////////////////////////////////////////////




//////////////////////////////////////////////////////////////////////////// Cleanes the initial mask

///// Finds Skeleton


set(IM_start=objects.body.mask.image)

maskthinning_temp(objects,image1)

maskthinning_temp(objects,image1)

maskthinning_temp(objects,image1)

maskthinning_temp(objects,image1)


//if(Showillustrations)
```

```
  //imageview(image1, "Signal",image=image1,gamma=2.0)
  //imageview(objects.body, "Lines0",image=image1,gamma=2.0)
//end()
//CalcSkeletonByIntensity(image=image1,intensityEvalParam=0)
renameattr(skeleton=body)
CalcSkeletonByIntensity(stencil=skeleton, image=image1, IntensityEvaluationMode=2, IntensityEvalParam=-1)
renameattr(skeleton=skeleton_skeleton)
CalcSkeletonByIntensity(stencil=skeleton, image=image1, IntensityEvaluationMode=2, IntensityEvalParam=-2)
renameattr(skeleton_skeleton_byIntensity=skeleton_skeleton)
CalcSkeleton(stencil=skeleton_skeleton_byIntensity)
renameattr(skeleton_skeleton=skeleton_skeleton_byIntensity_skeleton)
set(obx=objects)
mask2stencil(objects.skeleton_skeleton.mask, neighbourhood=8)
stencil2objects()
setattr(skeleton_skeleton,body)
calczone(-40, stencil=obx.body)
zonemask(-40,oo)
calcarea()
objectfilter(area>20)

set(IM_skeleton=objects.skeleton_skeleton.mask.image)

renameattr(sk=skeleton_skeleton)
if(ShowIllustrations)
  //imageview(objects.sk, "Lines1",image=image1,gamma=2.0)
```

```
end()

div(result,image1, infinity=0, spreadfactor=1000, uncertainty=0, result_type="Unsigned,short")

mask(0.1,image=result)

set(M_10=mask)

mask(result.median, image=result)

and(image=M_10.image,mask=mask.image)

and(image=objects.sk.image,mask=image)

setattr(sk_thresholded,image.vector)

set(ob2=objects)


RemoveSkeletonLayer(skeletonName="sk")

set(ob3=objects)

RemoveSkeletonLayer(skeletonName="sk_LayerRemoved")

set(ob4=objects)

RemoveSkeletonLayer(skeletonName="sk_LayerRemoved_LayerRemoved")

renameattr(sk_mod=sk_LayerRemoved_LayerRemoved_LayerRemoved)

calcarea(sk_mod)

calcintensity(sk_mod,image=result)

set(ob5=objects)


/////////////////// The found skeleton is in the list ob5, stencil sk_mod

/////////////////////////////////////////////////////////////////////// End of  Finds Skeleton



/////////////////////////////////////////////// Cell detection, creates from skeleton cells borders

stencil2objects(objects.sk_mod)
```

```
FillSmallHoles()


setattr(filled,objects.filled,objects=ob5)

calcarea(filled)

calcattr(RelativeHoleArea,(1.0*filled_area)/(1.0*sk_mod_area))

objectfilter((sk_mod_area>100 or sk_mod_intensity>15) and sk_mod_area>30)

objectfilter(RelativeHoleArea<0.04 or sk_mod_area>600)


calcerosion(-3,stencil=sk)


Calcskeleton(sk_mod)

RemoveSkeletonLayer(skeletonName="sk_mod_skeleton")


set(Lines=objects)

if(ShowIllustrations)

  //imageview(lines.sk, "Lines2",image=image1,gamma=2.0)

  //imageview(lines.sk_LayerRemoved, "Lines3",image=image1,gamma=2.0)

  //imageview(lines.sk_mod, "Lines4",image=image1,gamma=2.0)

end()

blank(image1.width,image1.height,1)

set(IM_blank=image)

set(IM_blank.type="mask")

clearborders(IM_blank,4)


not(image=stencil)
```

```
clearborders(image,1)

or(image=stencil,mask=Lines.sk_eroded.mask.image)

set(M_border=image)
not(image=image)
clearborders(image,1)
set(M_body=stencil)

mask2stencil(M_body)
stencil2objects()

calcerosion(-5)
stencil2objects(objects.eroded)
calcarea()

RemoveSmallObjects(minarea=1200)

FillSmallHoles(minimumholearea=600)
if(lines.count==0)
  calcarea()
  objectfilter(area>image1.width*image1.height+10)
else()
  if(lines.body.area>200)

    if(OL_erase.count>0)
```

```
      carrypixels(image=objects.body.image,mask=ol_erase.zonemask.mask,data=0)
      setattr(body,image.vector)
      setattr(index,image)
      calcborder()

      calcarea()
      objectfilter(area>1200)
      calcintensity(image=image1)
      and(image=objects.filled.image,mask=objects.body.mask.image)
      setattr(filled,image.vector)
     end()
    else()
     calcarea()
     objectfilter(area>image1.width*image1.height+10)
    end()
   end()

   set(LeafCells=objects)

}




proc DetectType2Objects(
  image image1 in "Input image with intensity information",
  objectlist Lines in "Input list of line structure",
  objectlist LeafCells inout "Input-output list of cells. In output list regions, which belong to type2 objects are erased.",
  bool showillustrations=yes in,
  objectlist ObjectsType2 out "List of detected type objects"
)
{
  ////////////////////////////////////////////////// Finds Type2 objects

  stencil2objects(Lines.sk_mod)

  FillSmallHoles(minimumholearea=1000)
  calcarea()
  calcarea(filled)
  mask2stencil(objects.filled.mask)
  stencil2objects()
  calcarea()
  objectfilter(area>150)
  calcerosion(-1)

  and(image=objects.eroded.image, mask=lines.sk_mod.mask.image)
  setattr(border_corrected,image.vector)
  calcintensity(border_corrected,image=image1)

  objectfilter(border_corrected_intensity>1000)
```

```
///New part added in Nov 9, 2006

Dark_Mask(image1, 7)

CalcErosion(-1)

Stencil2Objects(Objects.eroded)

Calcintensity(image=result)

renameattr(DarkSig=intensity)

Calcintensity(image=image1)

CalcAttr(RelDarkSig,(1.0*DarkSig)/(intensity))


CalcIntensity(border,image=image1)

Calczone(-3,zonetype="equidistant")

zonemask(-3,3)

renameattr(BorderRegion=zonemask)

Zonemask(3,oo)

Renameattr(InnerRegion=Zonemask)

Calcintensity(InnerRegion,image=image1)


Calcintensity(BorderRegion,image=image1)

CalcAttr(Inner2BorderRatio, InnerRegion_intensity/Border_intensity)

CalcAttr(Border2BorderRatio, BorderRegion_intensity/Border_intensity)

Zonemask(4, 5)

Renameattr(Region4=Zonemask)

CalcIntensity(Region4, image=image1)

Mean(5,image=image1)
```

```
  CalcStat("Min",AttrName="MinIntensity", stencil=InnerRegion,image=image)

  //Zonemask(7,oo)

  //Renameattr(Region7=Zonemask)

  //CalcIntensity(Region7, image=IM_projected1)

  CalcAttr(MinToRegion4Ratio, (1.0*MinIntensity)/Region4_intensity)

  CalcAttr(MinToBorderRatio, (1.0*MinIntensity)/Border_intensity)

  CalcAttr(DarkToRegion4, (1.0*DarkSig)/Region4_intensity)

  CalcArea()

  //CalcWidthLength()

  //CalcAttr(Width2LengthRatio,(2.0*Half_width)/(full_length))

  deleteattr(zone,outerzone)

  set(ObjectsType2=objects)



  ///// End of finds Type2 objects


  ///// Removes from list of cells Type2 objects

  carrypixels(image=LeafCells.body.image,mask=ObjectsType2.body, data=0)

  //carryobjects(image=cells.body.image, stencil=ObjectsType2.body, data=ObjectsType2.ObjectNumber)

  stencil2objects(image.vector)

  set(LeafCells=objects)

}


proc CalcFillStencil_MPI(
  string StencilName="body" noquote in "Name of the stencil type attribute in the input object list, which will be filled. Only short attribute names are supported: StencilName=body is correct whereas centers=Objects.Body is erroneous. The attribute name can be given with and without the quotation marks, i.e. both are correct: StencilName=body and StencilName=\"body\".",

  objectlist objects inout "Input-output object list. The output list includes a new stencil type attribute, which corresponds to the created objects. A number of objects is equal to a number of the initial centers. In case of body stencil (StencilName=\"body\") the added attribute has the name \"Filled\" otherwise \"StencilName_Filled\". For example, if StencilName=\"BrightMask\" the attribute name is \"BrightMask_Filled.\"."

) object list attribute creation "Fills the gaps (holes) in the geometrical regions given by a stencil type attribute. The output object list includes a new stencil, which corresponds to the \"filled regions\". See also the module FillObjects(), which performs exactly the same operation. The difference is that CalcFillStencil() can be applied to all stencil type attributes while FillObjects() only to the body one."

{

  set(objects_in=objects)

  eval("set(Stencil=objects." & StencilName & ")")

  if(errorcode==0)

    if(Stencil.class!="intervalvector")

      error("Input StencilName does not correspond to the stencil-type attribute. The input StencilName must specify the stencil type attribute in the input object list, which defines the centers positions. Only short attribute names are supported: centers=body is correct whereras centers=Nuclei.Body is erroneous. The attribute name can be given with and without the quatation marks, both are correct: centers=body and centers=\"body\". It is assumed that the centers approximately correspond to the local intensity maximums (maximum regions) or to minimums (minimum regions).")

    end()

  else()

    FindShortAttrNameFromLongName(StencilName)

    eval("set(Stencil=objects." & ShortAttrName & ")")

    if(errorcode==0)

      if(Stencil.class=="intervalvector")

          error("Input StencilName does not correspond to the attribute in the input object list. Probably, a long attribute name was used instead of a short one. Only short attribute names are supported: StencilName=body is correct whereras StencilName=Objects.Body is erroneous. The attribute name can be given with and without the quatation marks, both are correct: StencilName=body and StencilName=\"body\". It is assumed that the centers approximately correspond to the local intensity maximums (maximum regions) or to minimums (minimum regions).")

      end()

    end()

    error("The input StencilName does not correspond to the existing stencil type attribute in the input object list. The input StencilName must specify the stencil type attribute in the input object list.. Only short attribute names are supported: StencilName=body is correct whereras StencilName=Objects.Body is erroneous. The attribute name can be given with and without the quatation marks, both are correct: StencilName=body and StencilName=\"body\".")

  end()
  stencil2objects(stencil)
  Fillobjects()
  if(objects.count!=objects_in.count)
    set(st_temp=objects.body)
    FillStencilfromCenter_tech(objects_in.body,objects.body)
    stencil2objects(stencil)
  end()
  set(ob2=objects)
  if(StencilName=="body")
    setattr(Filled,ob2.body,objects=objects_in)
    setattr(Filled_border,ob2.border)
  else()
    set(temp="" & StencilName & "_Filled,ob2.body,objects=objects_in)")
    eval("setattr(" & temp )
    set(temp="" & StencilName & "_Filled_border,ob2.border)")
    eval("setattr(" & temp )
  end()
}



proc Haustoria_detection(
//INPUT
image reference in "Nuclei stained image with intensity information. Nuclei are detected by this image. Supported image types are: 8-bit and 16-bit.",
objectlist Lines in "Input list of line structure. Used only for additional attribute creation",
bool ShowIllustrations=NO in "YES- Output illustrations are depicted. No- Output illustrations are not shown.",
//OUTPUT
```

```
objectlist Haustoria out "Output object list with initial objects. Object filter must be applied in the next steps for Haustoria classification.",
double line_mean out "Mean intensity of the line structure",
double nuclei_quantile7 out "Quantile 0.7 for intensity of nuclei"
) object recognition "Detects initial objects for haustoria detection. Object filter must be applied in the next steps for Haustoria classification. The procedure corresponds to nuclei_detection_A() routine."
{
  input(MinimumHaustoriaArea, 120, "Minimum Haustorium Area","i", "Minimum allowed area for haustorium, objects with area less than the limit are removed.")

  // Inner-technical nuclei detection procedure
    //nuclei_detection_A_inner(reference, 0.9, MinimumHaustoriaArea, 0.4, 0.7) // AK 5.12.2007
  nuclei_detection_A_inner(reference, 0.9, MinimumHaustoriaArea, 0.4, 1) // AK 5.12.2007

  ErrorReceiverForNucleiDetectionLibrary_v1()
  if(ShowIllustrations)
    imageview(item=InitialMask.border.mask, label="InitialMask", title="Mask after the initial thresholding. Adjust par. \"Threshold Adjustment\"", image=reference, gamma=2.6)
  end()
  // Removes breaking lines between the stuck nuclei,
  // Input parameter is fixed
  controlbreakinglines_v12(nuclei, 0.8)
  // Adds to object list a contrast attribute
  object_contrast_general(cells_out, reference, InitialMask=InitialMask.body.mask)

  // Creates the illustration "LowContrastObjects"
  if(ShowIllustrations)
    set(CL_Temp=objects)
    objectfilter(contrast<=0.1)
    imageview(item=objects.border, label="LowContrastObjects", title="Discarded low contrast objects. Adjust par. \"Minimum Nuclear Contrast\"", image=reference, gamma=2.6)
    set(objects=CL_Temp)
  end()
  objectfilter(contrast>0.1)

  if(ShowIllustrations)
    imageview(item=objects.border, label="InitialHaustorial", title="Initial candidates for Haustoria. Classification must be applied.", image=reference, gamma=2.6)
  end()
  calcarea()
  calcintensity(image=reference)

  and(image=objects.body.image, mask=Lines.sk_mod_skeleton_LayerRemoved.mask.image)
  setattr(lines, image.vector)

  if(Lines.sk_mod_skeleton_LayerRemoved.area>50)
    stat("mean",variable="line_mean", mask=Lines.sk_mod_skeleton_LayerRemoved,image=reference)
  else()
    set(line_mean=reference.mean)
  end()

  if(objects.count>30)
    quantile(objects.intensity,0.7)
    set(nuclei_quantile7=quantile)
  else()
    if(objects.count>24)
      quantile(objects.intensity,0.6)
```

```
        set(nuclei_quantile7=quantile)
      else()
        if(objects.count>20)
          quantile(objects.intensity,0.5)
          set(nuclei_quantile7=quantile)
        else()
          set(nuclei_quantile7=line_mean)
        end()
      end()
    end()
    calcarea(lines)
    calcwidthlength()
    calcattr(Width2LengthRatio, (2.0*half_width)/(full_length))
    //objectfilter(intensity>line_mean and half_width>4 and lines_area<5 and  (Width2LengthRatio>0.45 or (intensity>1.4*line_mean and Width2LengthRatio>0.4)))

    deleteattr(zone, outerzone)
    set(Haustoria=objects)
  }

  proc SpotClassificationLeafCells(
  image IM_projected1 in "Input image with intensity information. Spots are detected by this image",
  objectlist spots in "Input list of the initial spots",
  bool showillustrations in "Yes- depicts the spot classification illustrations, No- sthe pot classification illustrations are not shown",
  objectlist SpotsFiltered out "List of the classified spots with the calculated attributes"
  ) "Detects the actual spot locations by the initial list. Creates a new list and calculates attributes like area, contrast, width, length etc and thereupon classifies/filters the spots."
```

```
{
    input(spotMinimumArea, 25, "SpotMinimumArea","i","Minimum allowed area for spots. Objects with area less than the limit are discarded.")
    input(SpotMinimumContrast, 0.25, "SpotMinimumContrast", "d", "Minimum allowed contrast for spots. Objects with the parameter value less than the limit are discarded.")
    input(SpotMinimumRoundness, 0.65, "SpotMinimumRoundness", "d", "Minimum allowed roundness parameter for spots. Objects with the parameter value less than the limit are discarded.")
    input(MinimumWidth2LengthRatio, 0.5, "SpotMInimumWidth2LengthRatio","d", "Minimum allowed width to length ratio for spots. Objects with the ratio below the limit are discarded.")

    Bright_Mask(IM_projected1,3)
    set(r6=result)
    div(result, IM_projected1, result_type="unsigned,short", spreadfactor=1000, infinity=0)
    mask(threshold=0.05,image=result)
    set(M10=mask)
    mask(threshold=IM_projected1.median, image=r6)
    and(image=M10.image,mask=mask.image)
    set(M11=image)

    calcerosion(-3.3,spotcenters,objects=spots)
    renameattr(centerseroded3=spotcenters_eroded)

    calcerosion(-2,spotcenters)
    calcerosion(-12,spotcenters_eroded,restrictivestencil=m10, numberofsteps=3)
    CalcFillStencil_MPI(StencilName="spotcenters_eroded_eroded")

    RenameAttr(body2=spotcenters_eroded_eroded_filled)
    calcborder(body2)
    CalcArea(body2)
    set(OL_temp1=objects)
```

```
stencil2objects(objects.body2)
CalcWidthLength()
set(OL_temp2=objects)
setattr(FullLength,ol_temp2.full_length,objects=ol_temp1)
setattr(HalfWidth,ol_temp2.half_width)
CalcAttr(Width2LengthRatio, iif(FullLength>0,(2.0*HalfWidth/FullLength),0.0))
CalcRoundnessCorrected(Body2)


setattr(body4,body2)
calcerosion(1,body4)
calcerosion(-1,body4_eroded)
renameattr(body4=body4_eroded_eroded)
set(OL_temp1=objects)
stencil2objects(objects.body4)
CalcWidthLength()
set(OL_temp2=objects)
setattr(body4_FullLength,ol_temp2.full_length,objects=ol_temp1)
setattr(body4_HalfWidth,ol_temp2.half_width)
CalcAttr(body4_Width2LengthRatio, iif(FullLength>0,(2.0*HalfWidth/FullLength),0.0))
CalcArea(body4)
CalcRoundnessCorrected(Body4)
CalcAttr(AreaRatio42,iif(Body2_area>0, (1.0*Body4_area)/(1.0*Body2_area),0.0))


objectfilter(body2_area>0)
if(ShowIllustrations)
```

```
  imageview(objects.body2_border, "InitialSpots", image=IM_projected1, title="Spot candidates after the initial detection", gamma=2.0)

end()

ObjectFilter(HalfWidth>2.01)

ObjectFilter(Width2LengthRatio>MinimumWidth2LengthRatio or (Width2LengthRatio>0.8*MinimumWidth2LengthRatio and body2_roundnesscorrected>1.1*SpotMinimumRoundness ) or (body4_Width2LengthRatio>0.6 and body4_RoundnessCorrected>0.8 and AreaRatio42>0.75) )

if(ShowIllustrations)

  imageview(objects.body2_border, "WidthLength", image=IM_projected1, title="Spots after filtering by  Width and Width2LengthRatio", gamma=2.0)

end()

//set(ol_s1a=objects)


objectfilter(Body2_roundnesscorrected>=SpotMinimumRoundness )

if(ShowIllustrations)

  imageview(objects.body2_border, "Roundness", image=IM_projected1, title="Spots after filtering by  Roundness", gamma=2.0)

end()




objectfilter(body2_area>=spotMinimumArea)

if(ShowIllustrations)

  imageview(objects.body2_border, "Area", image=IM_projected1, title="Spots after filtering by  Area", gamma=2.0)

end()

CalcErosion(-2,body2)

calcborder(body2_eroded)

CalcIntensity(Body2_eroded_border, image=IM_projected1)

CalcStat("max",stencil=body2, image=IM_projected1)


CalcAttr(th, Body2_eroded_border_intensity+0.3*(max-Body2_eroded_border_intensity)+2.0*sqrt(Body2_eroded_border_intensity))
```

```
threshmask(stencil=body2_eroded,threshold=th,image=IM_projected1)

calcerosion(-13, spotcenters_eroded,restrictivestencil=threshmask)

renameattr(body3=spotcenters_eroded_eroded)

calcattr(body2_contrast, (spotpeakintensity-Body2_eroded_border_intensity)/(spotpeakintensity+Body2_eroded_border_intensity))

selectbrightspots(body2,body2,image=IM_projected1)

calcerosion(-1,brightspots)

carrypixels(image=objects.brightspots_eroded.image,mask=objects.brightspots.mask,data=0)

setattr(region8,image.vector)

calcintensity(region8,image=IM_projected1)

calcattr(PeakToReg8Intensity, (1.0*max-Region8_intensity)/(1.0*max-Body2_eroded_border_intensity))


objectfilter(body2_contrast>SpotMinimumContrast )

if(ShowIllustrations)

  imageview(objects.body2_border, "Contrast", image=IM_projected1, title="Spots after filtering by  Contrast", gamma=2.0)

end()


objectfilter(PeakToReg8Intensity<0.5)

if(ShowIllustrations)

  imageview(objects.body2_border, "Peak2Neighborhood", image=IM_projected1, title="Spots after filtering by  Peak to neighborhood filter", gamma=2.0)

end()


calcerosion(-16,spotcenters_eroded,restrictivestencil=m10, numberofsteps=4)

CalcFillStencil_MPI(StencilName="spotcenters_eroded_eroded")


RenameAttr(body2=spotcenters_eroded_eroded_filled)

calcborder(body2)
```

```
CalcArea(body2)

set(OL_temp1=objects)

stencil2objects(objects.body2)

CalcWidthLength()

set(OL_temp2=objects)

setattr(FullLength,ol_temp2.full_length,objects=ol_temp1)

setattr(HalfWidth,ol_temp2.half_width)

CalcAttr(Width2LengthRatio, iif(FullLength>0,(2.0*HalfWidth/FullLength),0.0))

CalcRoundnessCorrected(Body2)



//ObjectFilter((Width2LengthRatio>=MinimumWidth2LengthRatio and body2_roundnesscorrected>SpotMinimumRoundness) or (Width2LengthRatio>0.8*MinimumWidth2LengthRatio and SpotMinimumRoundness>1.1) )

ObjectFilter((Width2LengthRatio>=0.8*MinimumWidth2LengthRatio and body2_roundnesscorrected>SpotMinimumRoundness))


Set(SpotsFiltered=objects)

// Cleanes the list, renames the attributes etc

Stencil2Objects(objects.body2)

CalcIntensity(image=IM_projected1)

CalcStat("Sum",AttrName="IntegratedSpotSignal",Stencil=Body,image=IM_projected1)

Setattr(Area,SpotsFiltered.body2_area)

Setattr(RoundnessCorrected,SpotsFiltered.body2_RoundnessCorrected)

Setattr(Width2lengthRatio,SpotsFiltered.Width2lengthRatio)

Setattr(FullLength,SpotsFiltered.FullLength)

Setattr(HalfWidth,SpotsFiltered.HalfWidth)

Setattr(Contrast, SpotsFiltered.Body2_contrast)
```

```
    Setattr(PeakIntensity, SpotsFiltered.max)

    Setattr(SpotCenters, SpotsFiltered.SpotCenters)


    SetAttr(ReferenceRegions,SpotsFiltered.Body2_eroded_border)

    SetAttr(ReferenceRegions_intensity,SpotsFiltered.Body2_eroded_border_intensity)

    SetAttr(ReferenceIntensity, ReferenceRegions_intensity)

    CalcArea(ReferenceRegions)

    SetAttr(CellIntensity, SpotsFiltered.CellIntensity)

    CalcAttr(SpotToCellIntensity, (1.0*PeakIntensity)/(1.0*CellIntensity))

    CalcAttr(IntegratedSpotSignal_Backgroundsubtracted, 1.0*IntegratedSpotSignal-(area*ReferenceIntensity))

    CalcAttr(IntegratedSpotSignal_Backgroundsubtracted, iif(IntegratedSpotSignal_Backgroundsubtracted>0.0000001,IntegratedSpotSignal_BackgroundSubtracted,0.0))

    Set(SpotsFiltered=objects)

}


proc Spot_detection_outputs_modified(objectlist objects in  "Input object list of ~cells (SearchObjects) with spot data, must contain same attributes as spot detection output list WholeCells. Please note that list of spots can not be used here as it does not contain all required data, e.g. number of ~cells, ~cell area etc.",

  bool ShowOutputParameters=YES in "YES- Output parameters are reported to player and/or to database. No- Output Parameters are not reported.",

  string NamePreFix="" in "String, which is added to output names as prefix",

  double NumberOfSpots out "Total number of detected spots.",

  double SpotsPerObject out "Number of spots per object (i.e. number of spots per ~cell).",

  double SpotsPerArea out "Number of spots per SearchRegion area (i.e. number of spots per visible ~cell area).",

  double IntegratedSpotSignalPerCellularSignal out "Integrated spot signal over all spots normalized by integrated ~cellular signal (total signal over all SearchRegion area).",

  double IntegratedSpotSignalPerCellularSignal_BackgroundSubtracted out "Integrated spot signal over all spots background subtracted and normalized by integrated ~cellular signal (total signal over all SearchRegion area).",


  double IntegratedSpotSignalPerArea out "Integrated spot signal per SearchRegion area (per visible ~cell area).",
  double IntegratedSpotSignalPerArea_BackgroundSubtracted out "Integrated spot signal BackgroundSubtracted per SearchRegion area (per visible ~cell area)."


  )  spot detection "Supporting procedure for spot detection library. Reports spot detection outputs. Please note that output parameters can be reported by list of SearchObjects (e.g. list of ~cells), which contains spot data. List of spots does not contain information about number of ~cells, ~cell area etc and therefore all spot outputs can not be reported by this list. Input list Objects must contain same attributes (spot data) as spot detection output list WholeCells. Could be used in multiple-field scripts or after spot or ~cell classification etc. See more in Opera spot detection manual."

{


  if(objects.count>0)


    set(NumberOfSpots=1.0*objects.NumberOfspots.sum)

    set(SpotsPerObject=NumberOfSpots/(1.0*objects.count))

    if(objects.searchregionarea.sum>0)

      set(SpotsPerArea=NumberOfSpots/objects.searchregionarea.sum)

      set(IntegratedSpotSignalPerArea=(1.0*objects.integratedspotsignal.sum)/(1.0*objects.searchregionarea.sum))

      set(IntegratedSpotSignalPerArea_backgroundsubtracted=(1.0*objects.integratedspotsignal_backgroundsubtracted.sum)/(1.0*objects.searchregionarea.sum))


    else()

      set(SpotsPerArea=NAN)

      set(IntegratedSpotSignalPerArea=NAN)

      set(IntegratedSpotSignalPerArea_backgroundsubtracted=NAN)

    end()


    if(objects.searchregionintegratedSignal.sum>0)

      set(IntegratedSpotSignalPerCellularSignal=(1.0*objects.integratedspotsignal.sum)/(1.0*objects.searchregionintegratedSignal.sum))

      set(IntegratedSpotSignalPerCellularSignal_backgroundsubtracted=(1.0*objects.integratedspotsignal_backgroundsubtracted.sum)/(1.0*objects.searchregionintegratedSignal.sum))

    else()

      set(IntegratedSpotSignalPerCellularSignal=NAN)
      set(IntegratedSpotSignalPerCellularSignal_backgroundsubtracted=NAN)
    end()
  else()



    set(NumberOfSpots=0.0)
    set(SpotsPerObject=NAN)
    set(SpotsPerArea=NAN)
    set(IntegratedSpotSignalPerCellularSignal=NAN)
    set(IntegratedSpotSignalPerCellularSignal_backgroundsubtracted=NAN)
    set(IntegratedSpotSignalPerArea=NAN)
    set(IntegratedSpotSignalPerArea_backgroundsubtracted=NAN)
  end()

  if(ShowOutputParameters)
    create_spot_outputs()
  end()

}



proc ReduceStencilObjectsByArea_MPI(
  stencil stencil inout "Input-output stencil. On the output stencil object  number is limited to ObjectNumberLimit" ,
  int minarea=0 explicit in "Minimum allowed area for objects. Objects with area less than the limit are discarded",
  int ObjectNumberLimit=31999 explicit in "Maximum allowed number of objects on the stencil.",
  bool Faster=0 explicit in,
  ) [hidden] "Reduces a number of objects on the input stencil to the specified limit. The selection criterion is object area."
{
  if(stencil.itemcount<=ObjectNumberLimit)
    return()
  end()
  if(ObjectNumberLimit<1)
    blank(stencil.imagewidth,stencil.imageheight)
    convelems(image, size=2, allowfactorchange=no)
    set(result.type="stencil")
    set(stencil=stencil.vector)
    return()
  end()

  set(objectnumberLimit=objectnumberLimit+1)
  create("objectlist")
  SetAttr(body,stencil,objects=objectlist, autorecalc=no)
  CalcArea()

  if(minarea>0)
    objectfilter(area>=minarea)
    if(objects.count<objectnumberLimit)
      set(stencil=objects.body)
      return()
    end()
  end()
  if(Faster)
    set(table=tbl(area=objects.area))
    Sort_Prepare(table,0)
    Reorder(table,order)
    set(TH_area=result.area[objectnumberLimit-2])
    objectfilter(area>TH_area)
  else()
    tabulate("x",objects.count)
    setattr(objectno,result)
    set(table=tbl(area=objects.area, objectno=objects.objectno))
    Sort_Prepare(table,0)
    Reorder(table,order)

    set(Ol1=objects)
    set(TH_area=result.area[objectnumberLimit-2])
    set(TH_objectsno=result.objectno[objectnumberLimit-2])
    set(xm=OL1.count)
    calcattr(attr_final,iif(area<TH_area, 0,objectno+(area-TH_area)*xm))
    delete(objects.objectsno)
    objectfilter(attr_final<=TH_objectsno)
  end()
  set(stencil=objects.body)
}
```

```
proc CalcRoundnessCorrected(

  String StencilName="body" noquote in "Name of a stencil stencil type attribute in the input list. The input must correspond to the stencil type attribute in the input list. Only short syntax of attribute names is supported: StencilName=body is correct and  StencilName=Objects.Body is erroneous. The input can be given with and without the quatation marks, both are correct: StencilName=body and StencilName=\"body\".",

  objectlist objects inout "Input-output object list. The output list contains the calculated corrected roundness attribute RoundnessCorrected. In case of body stencil (StencilName=\"body\") the attribute name is \"RoundnessCorrected\" otherwise \"StencilName_RoundnessCorrected\". For example, if StencilName=\"MembraneRegion\" the attribute name is \"MembraneRegion_RoundnessCorrected\"."

) object list attribute creation "Finds the corrected roundness parameter for a stencil type attribute. The output list contains the found attribute RoundnessCorrected."

{

  set(objects_in=objects)

  if(!defined("objects." & StencilName))



    error("In the input object list there is no attribute with name " & StencilName & ". The input StencilName must correspond to the stencil type attribute in the input list. Only short syntax of attribute names is supported: StencilName=body is correct and  StencilName=Objects.Body is erroneous. The input can be given with and without the quatation marks, both are correct: StencilName=body and StencilName=\"body\".")

  end()

  eval("SetAttr(CurrentStencilX, objects." & StencilName & ")")

  if(errorcode!=0)

    error()

  end()

  if(objects.CurrentStencilX.class!="intervalvector")

      error(StencilName & " is not a stencil type attribute. The input StencilName must  correspond to a stencil type attribute in the input object list. Only short syntax of attribute names is supported: StencilName=body is correct and  StencilName=Objects.Body is erroneous. The input can be given with and without the quatation marks, both are correct: StencilName=body and StencilName=\"body\".")

  end()



  CalcArea(CurrentStencilX)

  CalcBorder(CurrentStencilX)

  CalcArea(CurrentStencilX_border)
  CalcAttr(RoundnessCorrected, 3.544*sqrt(1.0*CurrentStencilX_Area-CurrentStencilX_border_area/2.0)/CurrentStencilX_border_area-0.1, autorecalc=no)


  if(StencilName=="body")

    setattr(RoundnessCorrected, objects.RoundnessCorrected,objects=objects_in)

  else()

    set(temp="" & StencilName & "_RoundnessCorrected, objects.RoundnessCorrected,objects=objects_in)")

    eval("setattr(" & temp )

  end()

}





////////////////////////////////////////////////// PROCEDURE DEFINITIONS

// Procedure counts fields and controls if all images are present

proc ControlImageFieldsStackAC2(Table sourcedata in, int NumberOfChannels in, int ZplanesInStack in, int StackNo in, int StackCount out, int StartStack out, int EndStack out, int ImagesINOneStack out, int InvalidStacks out)

{

  set(ImagesINOneStack=NumberOfChannels*ZplanesInStack)

  if(ImagesINOneStack<1)

    error("Number of channels or number of images in stack is not a positive number. Please select a number larger than 0.")

  end()


  set (ImageCount = sourcedata.rowcount)   // determine the number of images present
```

```
if ((ImageCount ~ ImagesINOneStack) != 0)
  error("Number of images ("&ImageCount&") is not a multiple of the number of images per stack ("&ImagesINOneStack&").")
end()

set (StackCount = int(ImageCount / ImagesINOneStack))

if (StackCount == 0)
  error("No sufficient number of images found in the data file. Only "&StackCount&" images found, but "&ImagesINOneStack&" needed for a stack.")
end()

if (StackNo == 0)   // all images present are analysed
  set (StartStack = 1)
  set (EndStack = StackCount)
else()
  if (StackNo > StackCount)       /// Error message if too high Image Field number was chosen
    error("Please select a smaller field number. There are only "&StackCount&" stacks in the file, but you selected to analyse stack "&StackNo&". ")
  end()
  if (StackNo < 0)   ////Error message if negative Image Field number was chosen
    error("Please select a positive stack number number.")
  end()
  set (StartStack = StackNo)
  set (EndStack = StackNo)
  set (StackCount=1)
end()
set(InvalidStacks=0)   //variable for counting of invalid black image fields
```

```
}


//Procedure assigns image names for the current field
proc Assign_FirstLastZplane(Table sourcedata in, int _StackCounter in, int ImagesINOneStack in, int NumberOfChannels in, int FirstZplane out , int LastZplane out)
{
  set(FirstZplane=(_StackCounter-1)*ImagesINOneStack+1)
  set(LastZplane=(_StackCounter)*ImagesINOneStack-NumberOfChannels+1)
}



//////////////////////////////////////////////// MAIN SCRIPT STARTS
// INPUT PARAMETERS
input(StackNo, 0, "Stack No", "i", "Number of the stacks to analyze. If set to 0 all stacks are evaluated.")
input(NumberOfChannels, 1, "Number Of Channels", "i", "Number of channels per field")
input(ZplanesInStack,31,"Zplanes in stack","i","Number of z-planes in stack, a number of time moments in kinetic measurement")
input(ShowIllustrations,YES, "ShowIllustrations","y","YES- Output illustrations are depicted. No- Output illustrations are not shown.")

input(MinStdDev,150,"MinStdDev","i","Minimum standard deviation of pixel intensity in image.")
comment(
input(spotMinimumArea, 21, "SpotMinimumArea","i","Minimum allowed area for spots. Objects with area less than the limit are discarded.")
input(SpotMinimumContrast, 0.25, "SpotMinimumContrast", "d", "Minimum allowed contrast for spots. Objects with the parameter value less than the limit are discarded.")
input(SpotMinimumRoundness, 0.65, "SpotMinimumRoundness", "d", "Minimum allowed roundness parameter for spots. Objects with the parameter value less than the limit are discarded.")
input(MinimumWidth2LengthRatio, 0.5, "SpotMInimumWidth2LengthRatio","d", "Minimum allowed width to length ratio for spots. Objects with the ratio below the limit are discarded.")
)
// READS IN IMAGES
Singlewell(compact=yes)
```

```
// COUNTS FIELDS AND CONTROLS IF ALL IMAGES ARE PRESENT

ControlImageFieldsStackAC2(SourceData, NumberOfChannels, ZplanesInStack, StackNo)


// LOOP OVER IMAGE FIELDS

Foreach(StartStack .. EndStack, "_StackCounter")

  // Assigns image names for the current field, IM_CH1 - first channel image, IM_CH2 - second channel image etc

  Assign_FirstLastZplane()


  if(sourcedata.sourceImage[FirstZplane-1].max==0) // Invalid (black) image

    set(InvalidStacks=InvalidStacks+1)  // Counts invalid stacks

  else()

    // Evaluation of valid (not black) stacks

    // User script STARTS

    set( logfilename = "D:/dmeyer/test/filenames.txt" , imagefilename1=SourceData.sourcefilename[0])

    CreateDataCubeWithControl( sourcedata, FirstZPlane, LastZPlane, MinStdDev) // Creates the initial data cube

    DataCubeProjectionCorrection(FirstZplane, LastZPlane,DataCube) // Corrects the initial data cube (can be commented out)


    ////// Projects maximum intensity for each x-y position from datacube to image

    rearrange(datacube,vec(vec(0,0,1),vec(1,0,0),vec(0,1,0)), vec(datacube.depth, datacube.width,datacube.height), reduce="max")


    //Maximums3D(0,image=datacube)

    //StencilFrom3DTo2D(stencil=maximums,datacube=datacube) //Projects the found maximums from 3-dim to plane

    delete(datacube)

    set(IM_projected1=result)
    set(IM_projected2=result)

    //////////////////////////////////////////////////////////////////////////////////////////////////


    LeafCellsDetection(IM_projected1) // Detection of Leaf cells


    DetectType2Objects(IM_projected1, Lines,LeafCells) //Detects somata and "holes between cells", Somata classification is at the end of the script


    ////////////////////////////////////////////////////////////// Convolution before spot detection /////////////////////////////

    // added by kst 2006-09-08 //

    convolutionmask("Disk",3)

    convolution(image=IM_projected2, faster=yes)

    set(IM_projected2=image)


    ////////////////////////////////////////////////////////////// Spot detection

    // Creates the SerchRegion, where the spots are search

    CalcErosion(1,objects=LeafCells)

    set(leafCells=objects)

    carrypixels(image=objects.eroded.image, mask=Lines.sk_mod_skeleton_LayerRemoved, data=0)

    Setattr(SearchRegion,image.vector, objects=LeafCells)

    CalcArea(SearchRegion)

    Set(LeafCells=objects)


    // Detects the initial set of spots

    spot_detection_c_inner(IM_projected2, LeafCells.SearchRegion, LeafCells, SpotMinimumDistance=4, SpotPeakRadius=1, SpotReferenceRadius=10, SpotMinimumContrast=0.1,
SpotminimumToCellIntensity=0.8)

    delete(spotcandidates,wholecells)
```

```
// Detects the actual spot locations and attributes like area, contrast, width, length etc and classifies the spots

SpotClassificationLeafCells(IM_projected1,Spots)


//Detects the inital objects, outputs also statistics Line_mean and nuclei_quantile7

Haustoria_detection(IM_projected1,Lines, showillustrations=no)

// Haustoria classification

objectfilter(intensity>1.3*line_mean and intensity>nuclei_quantile7 and half_width>4 and lines_area<5 and (Width2LengthRatio>0.45 or (intensity>1.4*line_mean and Width2LengthRatio>0.4)),
objects=haustoria)

set(Haustoria=Objects)


// Discards spots, which overlap with haustoria

and(image=SpotsFiltered.body.image,mask=Haustoria.body.mask.image)

setattr(Haustoria,image.vector,objects=SpotsFiltered)

calcarea(Haustoria)

Objectfilter(Haustoria_area<1)

set(SpotsFiltered=objects)


Spot_detection_dataToSearchObjects(IM_projected1, "SearchRegion",SpotsFiltered,LeafCells)

set(LeafCells=Wholecells)


// Stomata and Gaps classification

objectfilter(area>300 and ((DarkToRegion4<0.32 and Border_intensity>Line_mean) or (DarkToRegion4<0.24 and Border_intensity>0.8*Line_mean)), objects=objectstype2)

set(Stomata=objects)

CalcStat("max",Stencil=body,image=Stomata.body.image,objects=objectstype2)

objectfilter(max==0)
```

```
set(Gaps=objects)


set( namelength = length( imagefilename1 ) )

set( corename = substr( imagefilename1, 1, namelength - 5 ) )

gamma( 2.0, image=IM_projected1)

set( Image_out = image)

writeimage( imagefile=corename & "_" & FirstZplane & "-" & LastZplane & ".png", image=Image_out, imageformat="png",  )

//write( "\tpng\t" & corename & "_" & FirstZplane & "-" & LastZplane & ".png" , logfilename, "ascii", append=true )




if(showIllustrations)

  imageview(IM_projected1, "Signal",image=IM_projected1, gamma=2.0)

  imageview(Leafcells.border, "Cells",image=IM_projected1, title="Detected Leaf Cells", gamma=2.0)

  imageview(Leafcells.body, "Cells2",image=IM_projected1,title="Detected Leaf Cells", gamma=2.0)

  imageview(Gaps.border, "Gaps",image=IM_projected1,title="Detected gaps between leaf cells",gamma=2.0)

  imageview(Stomata.border, "Stomata",image=IM_projected1,title="Detected Stomata",gamma=2.0)

  imageview(Haustoria.border,"Haustoria", image=IM_projected1, title="Detected Haustoria", gamma=2)

  imageview(spotsfiltered.border, "Spots", image=IM_projected1, title="Detected Spots", gamma=2.0)

end()

delete(M10,M11,M8,m9,                              ol_temp1,ol_temp2,M_bright,                              m_bright4,
IM_planenumber,IM_sk_mod_skeleton_LayerRemoved,mask,image,result,IM_projected2,spotsfilteredout,nuclei,r6,r7,spots,PlanenumberImage)

    // User script ENDS


// Collects results

if (StartStack+InvalidStacks<_StackCounter) // Not the first evaluated field

  AddObjects(LeafCells, objects=all_LeafCells, CheckOverlap=no)
```

```
      set(all_LeafCells=objects)  // Renames output from AddObjects()
      AddObjects(SpotsFiltered, objects=all_SpotsFiltered, CheckOverlap=no)
      set(all_SpotsFiltered=objects)
      AddObjects(ObjectsType2, objects=all_ObjectsType2, CheckOverlap=no)
      set(all_ObjectsType2=objects)
      AddObjects(Haustoria, objects=all_Haustoria, CheckOverlap=no)
      set(all_Haustoria=objects)
      AddObjects(Stomata, objects=all_Stomata, CheckOverlap=no)
      set(all_Stomata=objects)
      AddObjects(Gaps, objects=all_Gaps, CheckOverlap=no)
      set(all_Gaps=objects)
    else()
      set(all_LeafCells=LeafCells) // The first evaluated field
      set(all_SpotsFiltered=SpotsFiltered)
      set(all_ObjectsType2=ObjectsType2)
      set(all_Haustoria=Haustoria)
      set(all_Stomata=Stomata)
      set(all_Gaps=Gaps)
    end()
    delete(wholecells)
  end()   // end of the analysis of valid stacks
end() // end of the foreach loop over stacks


// REPORTS SPOT DETECTION  OUTPUT PARAMETERS
if (InvalidStacks<=EndStack-StartStack)
  // At least one field was analysed and we have an object list
```

```
set(OP_NumberOfLeafCells = all_LeafCells.count)  // 1
set(OP_NumberOfSpots = all_SpotsFiltered.count)  // 2
set(OP_NumberOfHaustoria = all_Haustoria.count)  // 3
set(OP_NumberOfStomata=all_Stomata.count)  // 4
set(OP_NumberOfGaps = all_Gaps.count)  // 5
set(OP_SpotAverageIntensity=all_SpotsFiltered.intensity.mean)
set(OP_SpotAverageArea=all_SpotsFiltered.area.mean)
set(OP_SpotTotalSignal=all_SpotsFiltered.IntegratedSpotSignal.sum)
set(OP_SpotTotalSignal_backgroundsubtracted=all_SpotsFiltered.IntegratedSpotSignal_backgroundsubtracted.sum)
set(OP_SpotAverageLength=all_SpotsFiltered.FullLength.mean)
set(OP_SpotAverageHalfWidth=all_SpotsFiltered.HalfWidth.mean)
set(OP_SpotAverageWidth2LengthRatio=all_SpotsFiltered.Width2LengthRatio.mean)
set(OP_SpotAverageRoundness=all_SpotsFiltered.RoundnessCorrected.mean)
set(OP_SpotAverageContrast=all_SpotsFiltered.Contrast.mean)
set(OP_SpotAveragePeakIntensity=all_SpotsFiltered.PeakIntensity.mean)
set(OP_TotalCellArea=all_LeafCells.SearchRegionArea.sum)

if(OP_NumberOfLeafCells>0)
  set(OP_NumberOfSpotsPerCell=(1.0*OP_NumberOfSpots)/(1.0*OP_NumberOfLeafCells))
else()
  set(OP_NumberOfSpotsPerCell=NAN)
end()
if(OP_TotalCellArea>0)
  set(OP_NumberOfSpotsPerArea=(1.0*OP_NumberOfSpots)/(1.0*OP_TotalCellArea))
  set(OP_SpotTotalSignalPerArea=(1.0*OP_SpotTotalSignal)/(1.0*OP_TotalCellArea))
```

```
    set(OP_SpotTotalSignalPerArea_backgroundsubtracted=(1.0*OP_SpotTotalSignal_backgroundsubtracted)/(1.0*OP_TotalCellArea))
  else()
    set(OP_NumberOfSpotsPerArea=NAN)
    set(OP_SpotTotalSignalPerArea=NAN)
    set(OP_SpotTotalSignalPerArea_backgroundsubtracted=NAN)
  end()
  if(OP_NumberOfSpots>0)
    set(OP_SpotTotalSignalPerArea_bg_subtr_per_spot=(1.0*OP_SpotTotalSignalPerArea_backgroundsubtracted)/(1.0*OP_NumberOfSpots))
  else()                                                    // error message from 2008-06-16  AK
    set(OP_SpotTotalSignalPerArea_bg_subtr_per_spot=NaN)  // error message from 2008-06-16  AK
  end()

else()
  // NO valid fields
  set(OP_NumberOfSpots = NAN)
  set(OP_NumberOfLeafCells = NAN)
  set(OP_NumberOfHaustoria = NAN)
  set(OP_NumberOfStomata=NAN)
  set(OP_NumberOfGaps=NAN)
  set(OP_NumberOfSpotsPerCell=NAN)

  set(OP_SpotAverageIntensity=nan)
  set(OP_SpotAverageArea=nan)
  set(OP_SpotTotalSignal=nan)
  set(OP_SpotTotalSignal_backgroundsubtracted=nan)
  set(OP_SpotAverageLength=nan)
  set(OP_SpotAverageHalfWidth=nan)
  set(OP_SpotAverageWidth2LengthRatio=nan)
  set(OP_SpotAverageRoundness=nan)
  set(OP_SpotAverageContrast=nan)
  set(OP_SpotAveragePeakIntensity=nan)
  set(OP_TotalCellArea=nan)

  set(OP_NumberOfSpotsPerArea=NAN)
  set(OP_SpotTotalSignalPerArea=NAN)
  set(OP_SpotTotalSignalPerArea_backgroundsubtracted=NAN)
  set(OP_SpotTotalSignalPerArea_bg_subtr_per_spot=NAN)
end()

// REPORTS ADDITIONAL OUTPUTS
set(OP_StackCount = StackCount) // Total number of image fields
set(OP_ValidStacks = StackCount - InvalidStacks) // Number of Valid Image fields is determined
output(OP_NumberOfSpots, "Number of Spots")
output(OP_NumberOfLeafCells, "Number of Leaf Cells")
output(OP_NumberOfHaustoria, "Number of Haustoria")
Output(OP_NumberOfStomata, "Number Of Stomata")
Output(OP_NumberOfGaps, "Number Of gaps between leaf cells")

output(OP_NumberOfSpotsPerArea, "Number of spots per area")
output(OP_NumberOfSpotsPerCell, "Number of spots per cell")
output(OP_SpotTotalSignalPerArea, "Total integrated spot signal per area")
output(OP_SpotTotalSignalPerArea_backgroundsubtracted, "Total integrated spot signal per area background subtracted")


output(OP_SpotAverageIntensity, "Average intensity of spots")

output(OP_SpotAverageArea, "Average area of spots")

output(OP_SpotTotalSignal, "Total integrated spot signal, over all spots")

output(OP_SpotTotalSignal_backgroundsubtracted, "Total integrated spot signal background subtracted, over all spots")

output(OP_SpotAverageLength, "Average length of spots")

output(OP_SpotAverageHalfWidth, "Average Half width of spots")

output(OP_SpotAverageWidth2LengthRatio, "Average width to length ratio of spots")

output(OP_SpotAverageRoundness, "Average roundness of spots")

output(OP_SpotAverageContrast, "Average contrast of spots")

output(OP_SpotAveragePeakIntensity, "Average PeakIntensity of spots")


output(OP_TotalCellArea, "Total Cell Area")


output(OP_StackCount, "Total number of Stacks analyzed")

output(OP_ValidStacks, "Number of valid Stacks")

output(OP_SpotTotalSignalPerArea_bg_subtr_per_spot, "Integrated spot signal per area bg subtracted per spot")
```

Supplemental Data S3. PERL scripts for “ Endomembrane script”. Scripts for reformatting and plotting data collected by the endomembrane script.

```
# name of script:PERL script for Endomembrane script A
# Program TransChart.pl
# for the transformation of .csv files from Opera multirun format to Acapella single run format.
# to execute this script rename .txt to .pl
print "\n";
print "Transform every chart in folder? (y/n)\n";
$all_files = <STDIN>;
chomp $all_files;
if( $all_files eq "n" )
{
print "Give name of .csv file in Opera multirun format: ";
$infile_name = <STDIN>;
chomp $infile_name;
push( @dateinamen, $infile_name)
}
if( $all_files eq "y" )
{
open( FILELIST, "dir *.csv /B |") || die print "could not open csv files.\n";
while( <FILELIST> )
{
$zeile = $_;
chomp $zeile;
print "$zeile\n";
if ( $zeile =~ /\.csv$/i )
{
push( @dateinamen, $zeile );
}
}
close FILELIST;
}
if( $all_files ne "y" | "n" )
{
print "Please select only y or n\n"
}
local $name = "xx";
foreach $name ( @dateinamen )
{
$file_format = "Normal";
@data = ();
$reading_data = "FALSE";
open( INFILE, "$name" ) || die print "Could not open $name\n";
print "reading file $name\n";
while ( <INFILE> )
{
$zeile = $_;
chomp $zeile;
@item = split( /;/, $zeile );
if( $item[0] =~ /Barcode/ )
{
$file_format = "Opera";
$outfile_name =$name;
$outfile_name =~ s/\.csv/_new\.csv/;
open( OUTFILE, "> $outfile_name" ) || die print "Could not open $outfile_name\n";
}
if( $reading_data eq "TRUE" && $file_format eq "Opera" )
{
printf OUTFILE "%01d%03d" . "000", $item[0], $item[1];
for ( $n = 3; $n < @item; $n = $n + 2 )
{
$item[ $n ] =~ s/\"//g;
print "--" . $item[ $n ];
if ( $item[ $n ] eq "" )
{
print OUTFILE ";nan";
}
else
{
print OUTFILE ";" . $item[ $n ];
}
}
print "\n";
print OUTFILE "\n";
}
if( $item[ 0 ] =~ /Row/ && $file_format eq "Opera" )
{
print OUTFILE "WellIndex";
for ( $n = 2; $n < @item; $n = $n + 2 )
{
$item[ $n ] =~ s/\"//g;
if( $item[ $n ] eq "" )
{
print OUTFILE ";nan";
}
else
{
print OUTFILE ";" . $item[ $n ];
}
}
$reading_data = "TRUE";
print OUTFILE "\n";
}
}
print "\n";
print "Chart(s) transformed\n";
close INFILE;
close OUTFILE;
}
# end of script: PERL script for Endomembrane script A
```

```
#name of script: PERL script for Endomembrane script B
# to execute this script rename .txt to .pl
#!/etc/bin/perl
# Program makeplot.pl to plot parameters in .csv files generated by
# Opera*/Acapella
# *using Opera-Charts requires additional program "TransChart2.pl"
# Authors: Kurt Stueber, Sebastian Schaaf, Susanne Salomon
# Date: 12.07.2007
# Structure of input data file:
#Output results;
#
#WellIndex;Number of valid Cells in Stack # 1;Number of valid Spots in Stack # 1;Number of Spots in and out of Cells in Stack # 1;Percents of inner Spots in Stack # 1;Average Area of Cells in Stack #1;Average
Area of Cells - Standarddeviation in Stack # 1;Percents of found Cell Area in Stack # 1;Average Number of Spots in Cells in Stack # 1;Average Number of Spots in Cells - Standarddeviation in Stack # 1;Average
Number of Spots per recognized Area in Stack # 1;Number of valid Cells in Stack # 2;Number of valid Spots in Stack # 2;Number of Spots in and out of Cells in Stack # 2;Percents of inner Spots in Stack #
2;Average Area of Cells in Stack #2;Average Area of Cells - Standarddeviation in Stack # 2;Percents of found Cell Area in Stack # 2;Average Number of Spots in Cells in Stack # 2;Average Number of Spots in Cells
- Standarddeviation in Stack # 2;Average Number of Spots per recognized Area in Stack # 2;Number of valid Cells in Stack # 3;Number of valid Spots in Stack # 3;Number of Spots in and out of Cells in Stack #
3;Percents of inner Spots in Stack # 3;Average Area of Cells in Stack #3;Average Area of Cells - Standarddeviation in Stack # 3;Percents of found Cell Area in Stack # 3;Average Number of Spots in Cells in Stack #
3;Average Number of Spots in Cells - Standarddeviation in Stack # 3;Average Number of Spots per recognized Area in Stack # 3;Number of valid Cells in Stack # 4;Number of valid Spots in Stack # 4;Number of
Spots in and out of Cells in Stack # 4;Percents of inner Spots in Stack # 4;Average Area of Cells in Stack #4;Average Area of Cells - Standarddeviation in Stack # 4;Percents of found Cell Area in Stack # 4;Average
Number of Spots in Cells in Stack # 4;Average Number of Spots in Cells - Standarddeviation in Stack # 4;Average Number of Spots per recognized Area in Stack # 4;Number of valid Cells in Stack # 5;Number of
valid Spots in Stack # 5;Number of Spots in and out of Cells in Stack # 5;Percents of inner Spots in Stack # 5;Average Area of Cells in Stack #5;Average Area of Cells - Standarddeviation in Stack # 5;Percents of
found Cell Area in Stack # 5;Average Number of Spots in Cells in Stack # 5;Average Number of Spots in Cells - Standarddeviation in Stack # 5;Average Number of Spots per recognized Area in Stack # 5;Number of
Leaf Cells in whole Well;Average Cell Area in whole Well;Average Cell Area in whole Well - Standard Deviation;Number of Spots in whole Well;Average Number of Spots per Cell in whole Well;Average Number of
Spots per Cell in whole Well - Standard Deviation;Total Cell Area in Well;Percentage of total Cell Area in Well;Number Of Stomata;Average Intensity of Spots;Average Area of Spots;Average Length of
Spots;Average Half Width of Spots;Average Width to Length Ratio of Spots;Average Width to Length Ratio of Spots - Standard Deviation;Average Roundness of Spots;Average Roundness of Spots - Standard
Deviation;Average Contrast of Spots;Average Contrast of Spots - Standard Deviation;Average Peak Intensity of Spots;Total number of Stacks analyzed in Well;Number of valid Stacks in Well;Percentage of valid
Stacks in Well
#2004000;21;47;211;22.2749;0;0;0;0;0;0.000751291;19;26;251;10.3586;0;0;0;0;0;0.000517022;0;0;0;0;0;0;0;0;0;nan;0;0;0;0;0;0;0;0;0;nan;0;0;0;0;0;0;0;0;0;nan;0;0;0;0;0;0;0;0;0;nan;nan;nan;nan;nan;nan;nan;nan;n
an;nan;nan;5;2;40
#3004000;55;323;353;91.5014;4683.04;4444.49;71.9944;5.87273;5.71889;0.00125404;6;4;64;6.25;0;0;0;0;0;0.000156195;0;0;0;0;0;0;0;0;0;nan;54;238;263;90.4943;4674.06;5520.61;70.5498;4.40741;7.24378;0.00
0942951;6;3;116;2.58621;0;0;0;0;0;0.000310623;109;4678.59;4983.29;561;5.14679;6.53031;462938;35.636;0;52.6384;19.3316;5.45615;1.96399;0.767773;0.183888;0.955193;0.158327;0.32055;0.118691;95.7825
;5;4;80
#4004000;0;0;0;0;0;0;0;0;0;nan;0;0;2;0;0;0;0;0;0;nan;6;24;39;61.5385;8806.17;10019.9;14.7688;4;5.32917;0.000454227;2;0;0;0;0;0;0;0;0;0;2;6;70;8.57143;0;0;0;0;0;0.00184445;6;8806.17;10019.9;24;4;5.32917;48
095;7.38442;0;64.0833;23.1667;6.01667;2.025;0.734602;0.208949;0.919499;0.165875;0.254007;0.0709465;98.1667;5;2;40
#5004000;0;0;11;0;0;0;0;0;0;nan;11;30;253;11.8577;0;0;0;0;0;0.0016355;1;0;2;0;0;0;0;0;0;0;19;25;267;9.3633;0;0;0;0;0;0.000284249;18;36;50;72;3743.94;3282.15;18.8369;2;2.72246;0.000534196;18;3743.94;328
2.15;36;2;2.72246;59857;6.27898;0;53.2085;26.1944;6.63056;2.01667;0.672295;0.213519;0.843285;0.202866;0.265367;0.0715352;83.9167;5;3;60
#6004000;0;0;0;0;0;0;0;0;0;nan;73;609;677;89.9557;3857.93;5349.51;78.7201;8.34247;11.5533;0.00216242;0;0;0;0;0;0;0;0;0;nan;53;571;843;67.7343;3275.45;3898.54;48.5239;10.7736;10.4212;0.00328919;0;0;4;
0;0;0;0;0;0;nan;126;3612.92;4784.3;1180;9.36508;11.1137;408506;63.622;0;67.4805;19.5941;5.43992;1.94932;0.770346;0.192196;0.964252;0.173165;0.353963;0.12519;129.799;5;2;40
#7004000;0;0;7;0;0;0;0;0;0;nan;20;101;203;49.7537;0;0;0;0;0;0.00255858;35;291;481;60.499;2972.97;3136.68;29.0849;8.31429;7.63803;0.00279662;0;0;0;0;0;0;0;0;0;nan;0;0;0;0;0;0;0;0;0;nan;35;2972.97;3136.68
;291;8.31429;7.63803;92790;14.5424;0;58.9333;18.7491;5.20584;1.93471;0.790554;0.184135;0.977386;0.166557;0.363436;0.129676;115.533;5;2;40
#C:/Evotec/Training_MPI_scripts/MPI_leaves_h_seba.script;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
# we collect values from columns 10, 20, 30, 40, 50 if there are 74 values in the row
# 10, 20, 30, 40 64
# we pool the values from each row.
# we pool first and second, third and fourth etc. rows.
use Time::Local;
use GD;
#use strict;
```

```
( $sec, $min, $hour, $mday, $mon, $year, $yday, $isdst) = localtime( time );
$jahr = 1900 + $year;
%monat = ( '0', 'Jan', '1', 'Feb', '2', 'Mar', '3', 'Apr',
'4', 'May', '5', 'Jun', '6', 'Jul', '7', 'Aug',
'8', 'Sep', '9', 'Oct', '10', 'Nov', '11', 'Dec' );
$month = $monat{ $mon };
sub get_wells
{
local @wells = ();
foreach $line ( @data )
{
@item = split( /;/, $line );
push( @wells, $item[ 0 ] );
}
return @wells
}
sub get_data
{
local $parameter_no = $_[ 0 ];
local @werte = ();
foreach $line ( @data )
{
@item = split( /;/, $line );
push( @werte, $item[ $parameter_no ] );
}
return @werte;
}
sub calculate_mean
{
# call: $mean = &calculate_mean( @meine_werte );
local @wert = @_;
local $no_of_values = @wert;
local $n = 0;
local $sum = 0;
foreach $n ( @wert )
{
$sum = $sum + $n;
}
if ($no_of_values < 1 )
{
return "nan";
}
else
{
return $sum / $no_of_values;
}
}
sub calculate_standard_deviation
{
# call: $stddev = &calculate_standard_deviation( @meine_werte );
local @wert = @_;
local $mean = 0;
$mean = &calculate_mean( @wert );
local $sum = 0;
local $no_of_values = @wert;
local $v = 0;
foreach $v ( @wert )
{
$sum = $sum + ( $v - $mean ) * ( $v - $mean );
}
if ( $no_of_values > 0 )
{
$sum = $sum / $no_of_values;
}
return sqrt( $sum );
}
sub calculate_sum
{
# call: $sum = &calculate_sum( @meine_werte );
local @wert = @_;
local $sum = 0;
local $no_of_values = @wert;
local $v = 0;
foreach $v ( @wert )
{
$sum = $sum + $v;
}
return $sum;
}
sub plot_secondary_values
{
# call: &plot_secondary_values( $parameter_name, $no_of_stacks_per_row, $file_name, red_line_height)
# $no_of_stacks_per_row can be 4 or 5
local $parameter_name = $_[ 0 ];
local $no_of_stacks = $_[ 1 ];
local $file_name = $_[ 2 ];
local $red_line_height = $_[ 3 ];
local $var_type = 0;
#&plot_secondary_values( "Average Number of Spots per Picture projected", 5, $name, 400 );
#&plot_secondary_values( "Average Number of Spots per Picture counted", 5, $name, 170 );
#&plot_secondary_values( "Average Number of Spots per Cell", 5, $name, 7.1 );
if ($parameter_name eq "Average Number of Spots per Picture projected")
{
$var_type = 2;
$subtitle = "counted up to 100% image area"
}
if ($parameter_name eq "Average Number of Spots per Picture counted")
{
$var_type = 3;
$subtitle = "counted in picture"
}
if ($parameter_name eq "Average Number of Spots per Cell")
```

```
{
$var_type = 8;
$subtitle = "per cell"
}
print CSVOUT "\nparameter: $parameter_name\n\n";
local @col1 = &get_data( $var_type );
local @percent1 = &get_data( 7 );
local @cells1 = &get_data( 1 );
local @col2 = &get_data( $var_type + 10 );
local @percent2 = &get_data( 17 );
local @cells2 = &get_data( 11 );
local @col3 = &get_data( $var_type + 20 );
local @percent3 = &get_data( 27 );
local @cells3 = &get_data( 21 );
local @col4 = &get_data( $var_type + 30 );
local @percent4 = &get_data( 37 );
local @cells4 = &get_data( 31 );
local @col5 = ();
local @percent5 = ();
local @cells5 = ();
if ( $no_of_stacks == 5 )
{
@col5 = &get_data( $var_type + 40 );
@percent5 = &get_data( 47 );
@cells5 = &get_data( 41 );
}
local $no_of_rows = @col1;
local @col55 = ();
if ( $var_type == 8 )
{
if ( $no_of_stacks == 5 )
{
@col55 = &get_data( 55 );
}
if ( $no_of_stacks == 4 )
{
@col55 = &get_data( 45 );
}
}
local @well = &get_wells;
#$dummy = <STDIN>;
# find mean values
local @mean = ();
local @stddev = ();
local @row_values = ();
local @percent_area = ();
local @clean_row_values = ();
local @last_clean_row_values = ();
local @collected_sum = ();
local @collected_mean = ();
local @collected_stddev = ();
local @collected_percent = ();
local $no_of_values_this_row = 0;
local $no_of_values_last_row = 0;
local @collected_no_of_values = ();
local @no_of_clean_values = ();
local @no_of_stacks_in_filter = ();
#local @no_of_clean_percents = ();
local @no_of_cells = ();
local @last_no_of_cells = ();
local @collected_cells = ();
local @spots_per_cell_weighted = ();
local @row_values = ();
local @percent_area = ();
local @no_of_cells = ();
local @weighted_row_values = ();
local @clean_cells = ();
local @cvsout_print_lines = ();
local @cvsout_collected_print_lines = ();
for ( $row_no = 0; $row_no < $no_of_rows; $row_no++ ) #for every row
{
if ( $no_of_stacks == 5 ) # 5 Stack Chart
{
@row_values = ( $col1[ $row_no ], $col2[ $row_no ], $col3[ $row_no ], $col4[ $row_no ], $col5[ $row_no ] );
@percent_area = ( $percent1[ $row_no ], $percent2[ $row_no ], $percent3[ $row_no ], $percent4[ $row_no ], $percent5[ $row_no ] );
@no_of_cells = ( $cells1[ $row_no ], $cells2[ $row_no ], $cells3[ $row_no ], $cells4[ $row_no ], $cells5[ $row_no ] );
}
else
{
if ($no_of_stacks == 4 ) # 4 Stack Chart
{
@row_values = ( $col1[ $row_no ], $col2[ $row_no ], $col3[ $row_no ], $col4[ $row_no ] );
@percent_area = ( $percent1[ $row_no ], $percent2[ $row_no ], $percent3[ $row_no ], $percent4[ $row_no ] );
@no_of_cells = ( $cells1[ $row_no ], $cells2[ $row_no ], $cells3[ $row_no ], $cells4[ $row_no ] );
}
else
{
die print "Unknown stack number. End of program.\n";
}
}
$row_sum = 0;
$value = 0;
$percent = 0;
@clean_row_values = ();
@clean_percent_area = ();
@in_filter = ();
#$no_of_filtered_stacks_this_row = 0;
$no_of_values_this_row = 0;
$no_of_values_this_row_uncleaned = @row_values;
#print "NoVu = $no_of_values_this_row_uncleaned\n";
@weighted_row_values = ();
@clean_no_of_cells = ();
```

```
for( $v = 0; $v < $no_of_values_this_row_uncleaned; $v++ )
{
$value = $row_values[ $v ];
$percent = $percent_area[ $v ];
$cells = $no_of_cells[ $v ];
#print "$v\n";
#$dummy = <STDIN>;
if ( $value ne "nan" && $value ne "" && $percent > 0 )
{
if ($var_type == 2)
{
push( @clean_row_values, $value * 100 );
}
else
{
push( @clean_row_values, $value );
if ($var_type == 8 ) #correcting weight of values for "spots per cell"
{
#print "Value = $value\n";
#print "Cells = $cells\n";
#$dummy = <STDIN>;
$value = $value * $cells;
push( @weighted_row_values, $value );
push( @clean_no_of_cells, $cells );
}
}
push( @clean_percent_area, $percent );
}
if ($percent eq "nan" && $value ne "nan")
{
push( @in_filter, 1);
}
}
$clean_cells = &calculate_sum( @clean_no_of_cells );
$clean_cells[ $row_no ] = $clean_cells;
$no_of_values_this_row = @clean_row_values;
$no_of_percents_this_row = @clean_percent_area;
$no_of_stacks_in_filter_this_row = @in_filter;
$no_of_clean_values[ $row_no ] = $no_of_values_this_row;
$no_of_weighted_values = @weighted_row_values;
#$no_of_clean_percents[ $row_no ] = $no_of_percents_this_row;
$no_of_stacks_in_filter[ $row_no ] = $no_of_stacks_in_filter_this_row;
#print "Number of values this row = $no_of_values_this_row\n";
#print "Number of percents this row = $no_of_percents_this_row\n";
#for( $n = 0; $n < $clean_row_values; $n++ )
# {
# print "clean row values: $clean_row_values[ $n ]\n";
# }
#$ = <STDIN>;
# rescaling to spots per image,
# one image has 688*520 = 357760 pixel
```

```
#local $nx = 0;
#for ( $nx = 0; $nx < $no_of_values_this_row; $nx++ )
# {
# print "crv = " . $clean_row_values[ $nx ] . "\n";
# #$dummy = <STDIN>;
# $clean_row_values[ $nx ] = $clean_row_values[ $nx ] / $clean_percent_area[ $nx ];
# }
if( $no_of_values_this_row > 0 )
{
$row_sum = &calculate_sum( @clean_row_values );
$row_percent = &calculate_sum( @clean_percent_area );
if ($var_type == 2)
{
$row_mean = $row_sum / $row_percent;
}
else
{
if ($var_type == 3)
{
$row_mean = $row_sum / $no_of_values_this_row;
}
}
}
else
{
$row_mean = "nan";
}
if ( $var_type == 8 )
{
$mean[ $row_no ] = $col55[ $row_no ];
}
else
{
$mean[ $row_no ] = $row_mean;
}
# find standard deviations
#$stddev[ $row_no ] = &calculate_standard_deviation( @clean_row_values );
####### calculating combined bars ######
if ( ( $row_no % 2 ) == 1 )
{
if ( $var_type != 8 )
{
$collected_sum[ $row_no ] = &calculate_sum( @clean_row_values, @last_clean_row_values );
#print "collected sum = $collected_sum[ $row_no ]\n";
}
$collected_percent[ $row_no ] = &calculate_sum( @clean_percent_area, @last_clean_percent_area );
#print "Size of collected sum over last two rows 1 = $collected_sum[ $row_no ]\n";
#print "Size of collected percent over last two rows 1 = $collected_percent[ $row_no ]\n";
$collected_filter = &calculate_sum( @no_of_stacks_in_filter, @last_no_of_stacks_in_filter );
$collected_no_of_values[ $row_no ] = $no_of_values_this_row + $no_of_values_last_row;
#print "Collected Number of values last two rows = $collected_no_of_values[ $row_no ]\n";
if ( $var_type == 8 )
{
#print "1\n";
$collected_cells[ $row_no ] = &calculate_sum( @clean_no_of_cells, @clean_last_no_of_cells );
#print "Number of Cells = $collected_cells[ $row_no ]\n";
$collected_sum[ $row_no ] = &calculate_sum( @weighted_row_values, @last_weighted_row_values);
print "Collected Sum = $collected_sum[ $row_no ]\n";
}
#print "Kombi\n";
if ( ( $collected_percent[ $row_no ] > 0 ) && $var_type == 2 )
{
$collected_mean[ $row_no ] = $collected_sum[ $row_no ] / $collected_percent[ $row_no ];
#print "2\n";
}
else
{
if ( ( $collected_no_of_values[ $row_no ] > 0 ) && $var_type == 3 )
{
#print "Size of collected sum over last two rows 2 = $collected_sum[ $row_no ]\n";
$collected_mean[ $row_no ] = $collected_sum[ $row_no ] / $collected_no_of_values[ $row_no ];
#print "Size of collected mean over last two rows 3 = $collected_mean[ $row_no ]\n";
#print "3\n";
}
else
{
if ( ( $collected_no_of_values[ $row_no ] > 0 ) && $var_type == 8 )
{
#print "4\n";
$collected_mean[ $row_no ] = $collected_sum[ $row_no ] / $collected_cells[ $row_no ];
#print "Collected mean = $collected_mean[ $row_no ]\n";
}
else
{
$collected_mean[ $row_no ] = 0;
#print "5\n";
}
}
}
#print "Number of collected mean over last two rows 4 = " . $collected_mean[ $row_no ] ."\n";
$collected_stddev[ $row_no ] = &calculate_standard_deviation( @clean_row_values, @last_clean_row_values );
#$dummy = <STDIN>;L
}
@last_clean_row_values = @clean_row_values;
@last_clean_percent_area = @clean_percent_area;
@last_no_of_stacks_in_filter = @no_of_stacks_in_filter;
$no_of_values_last_row = $no_of_values_this_row;
@clean_last_no_of_cells = @clean_no_of_cells;
@last_weighted_row_values = @weighted_row_values
#$dummy = <STDIN>;
}
#$dummy = <STDIN>;
```

```
# find max values
local $max_value_1 = 0;
for( $n = 0; $n < $no_of_rows; $n++ )
{
if ( $mean[ $n ] ne "nan" )
{
if ( $max_value_1 < ( $mean[ $n ] + $stddev[ $n ] ) )
{
$max_value_1 = $mean[ $n ] + $stddev[ $n ];
}
}
}
#print "max_value_1 wirklicher wert: $max_value_1\n";
# plot
#print "max_value_1 = $max_value_1\n"; # 101
local $factor = 1;
if ( $max_value_1 > 50 )
{
while ($max_value_1 > 50 )
{
$max_value_1 = $max_value_1 / 10;
$factor = $factor / 10;
}
}
if ( $max_value_1 <= 5 )
{
if ( $max_value_1 <= 0 )
{
$max_value_1 = 1;
}
while ($max_value_1 <= 5 )
{
$max_value_1 = $max_value_1 * 10;
$factor = $factor * 10;
}
}
##### PLOT #####
#print "Corrected max_value_1 = $max_value_1\n";
$file_name =~ s/\.csv//;
local $outfile_name = "$file_name" . "_" . $parameter_name . ".png";
open( PLOT, "> $outfile_name" ) || print "Could not open $outfile_name\n";
local $balkenbreite = 15;
local $max_balken_hoehe = 300;
local $max_name_length = 7; # "1002000" = well name
local $xoffset = 20 + 8 * length ( $max_value_1 / $factor );
if ($var_type == 2)
{
$xoffset = $xoffset - 100;
}
local $yoffset = 70;
local $breite = 2 * $xoffset + 1.5 * $no_of_rows * $balkenbreite;
local $hoehe = 2 * $yoffset + $max_balken_hoehe + $max_name_length * 8;
print "Breite = $breite, Hoehe = $hoehe\n";
local $im = new GD::Image($breite,$hoehe);
# allocate some colors
local $white = $im->colorAllocate(255,255,255); # BZX
local $black = $im->colorAllocate(0,0,0);
local $red = $im->colorAllocate(255,0,0); # DEP
local $blue = $im->colorAllocate(0,0,255);
local $light_blue = $im->colorAllocate(120,145,242);# RK
local $orange = $im->colorAllocate(255,203,0); # FWY
local $yellow = $im->colorAllocate(255,255,0); # C
local $cyan = $im->colorAllocate(255,0,255); # GP
local $green = $im->colorAllocate(0,255,0); # NQST
local $pink = $im->colorAllocate(255,174,173); # AILMV
local $grey = $im->colorAllocate(170,170,170);
# Put a black frame around the picture
$im->rectangle(0,0,$breite-1,$hoehe-1,$black);
#$im->rectangle($xoffset, $yoffset, $xoffset+$laenge, $yoffset+$laenge,$black);
$im->string( gdLargeFont, $xoffset, $yoffset - 56, $project_name, $black );
$im->string( gdSmallFont, $xoffset, $yoffset - 26, "Average number of spots $subtitle, $no_of_stacks stacks per well", $black );
$im->string( gdSmallFont, $xoffset, $yoffset + 370, "Values above bars: number of stacks entering statistics", $black );
if ( $var_type == 8)
{
$im->string( gdSmallFont, $xoffset, $yoffset + 382, "Values in bars: number of recognized cells in whole well", $black );
}
else
{
$im->string( gdSmallFont, $xoffset, $yoffset + 382, "Values in bars: number of stacks filtered by showing less than 50% of all found spots inside of recognized cells", $black );
}
$im->string( gdSmallFont, $xoffset, $yoffset + 394, "Difference of both values to 10 (5 stack wells) or 8 (4 stack wells) means number of stacks without any recognized spots", $black );
local $x_position = 0;
local $y_position = 0;
local $tick_size = ( int ( $max_value_1 - ( $max_value_1 % 5 ) ) ) / 5;
print "tick size = $tick_size\n";
local $step_size = 0;
if ( $max_value_1 > 0 )
{
$step_size = ( $tick_size * 300 ) / $max_value_1;
}
#print "step size = $step_size\n";
#print "red line height = $red_line_height\n";
$im->line( $xoffset - 5, $yoffset + 300, $xoffset - 5, $yoffset, $black );
#$im->line( $xoffset + 1.5 * $no_of_rows * $balkenbreite + 5, $yoffset + 300, $xoffset + 1.5 * $no_of_rows * $balkenbreite + 5, $yoffset, $black );
$im->line( $xoffset + 1.5 * $no_of_rows * $balkenbreite, $yoffset + 300, $xoffset + 1.5 * $no_of_rows * $balkenbreite, $yoffset, $black );
#$im->line( $xoffset - 5, $yoffset + 300, $xoffset + $no_of_rows * $balkenbreite, $yoffset + 300, $black );
#$xx = $red_line_height * $factor * 300 / $max_value_1;
#print "Value of red line height to plot: $xx\n";
local $y_pos = 0;
while( $y_pos <= 300 && $step_size > 0 )
{
$label = $y_position / $factor;
```

```
$im->string( gdLargeFont, $xoffset - 12 - 8 * length( $label ) , $yoffset + 292 - $y_pos, $label, $black );
$im->line( $xoffset - 10, $yoffset + 300 - $y_pos, $xoffset - 5, $yoffset + 300 - $y_pos, $black );
$im->string( gdLargeFont, $xoffset + 1.5 * $no_of_rows * $balkenbreite + 7 , $yoffset + 292 - $y_pos, $label, $black );
$im->line( $xoffset + 1.5 * $no_of_rows * $balkenbreite , $yoffset + 300 - $y_pos, $xoffset + 1.5 * $no_of_rows * $balkenbreite + 5, $yoffset + 300 - $y_pos, $black );
#$a = $xoffset - 10;
#$b = $yoffset + 300 - $y_pos;
#$c = $xoffset - 5;
#$d = $yoffset + 300 - $y_pos;
#print "Drawing line at: " . $a . "-" . $b . "-" . $c . "-" . $d . "\n";
$y_pos = $y_pos + $step_size;
$y_position = $y_position + $tick_size;
}
for( $row_no = 0; $row_no < $no_of_rows; $row_no++ )
{
# find indices of $row
#print "Working on " . $well[ $well_no ] . "\n";
if ( $csvout_print_lines[ $row_no ] == "" )
{
$csvout_print_lines[ $row_no ] = $well[ $row_no ] . "\t";
}
# first index = horizontal = value, second index = vertical = well
if ( $mean[ $row_no ] ne "" ) # why???
{
if ( $mean[ $row_no ] ne "" )
{
print CSVOUT $well[ $row_no ] . "\t" . $mean[ $row_no ] . "\n";
$csvout_print_lines[ $row_no ] .= $mean[ $row_no ] . "\t";
$hoehe = ( $mean[ $row_no ] * $factor * 300 ) / $max_value_1;
$bar_upper_end = ( ( $mean[ $row_no ] + $stddev[ $row_no ] ) * $factor * 300 ) / $max_value_1;
$bar_lower_end = ( ( $mean[ $row_no ] - $stddev[ $row_no ] ) * $factor * 300 ) / $max_value_1;
print "Hoehe = $hoehe\n";
print "mean[ $row_no ] = " . $mean[ $row_no ] . "\n";
print "stddev[ $row_no ] = " . $stddev[ $row_no ] . "\n";
#print "factor = $factor\n";
print "max_value_1 = $max_value_1\n";
print "Bar upper end = $bar_upper_end\n";
print "Bar lower end = $bar_lower_end\n";
$x1 = $xoffset + $x_position;
$y1 = $yoffset + 300;
$y1a = $yoffset + 300 - $bar_upper_end;
$x2 = $xoffset + $x_position + 10;
$y2 = $yoffset + 300 - $hoehe;
$y2a = $yoffset + 300 - $bar_lower_end;
$im->filledRectangle( $x1, $y2, $x2, $y1, $light_blue );
$im->string( gdSmallFont, $x2 - 7, $y2 - 12, $no_of_clean_values[ $row_no ], $black );
if ( $var_type == 8 )
{
$im->stringUp( gdSmallFont, $x1 - 2, $y2 + 18, $clean_cells[ $row_no ], $grey );
}
else
{
```

```
$im->string( gdSmallFont, $x2 - 7, $y2 - 1, $no_of_stacks_in_filter[ $row_no ], $grey );
}
#$im->line( $x1 + 2, $y1a, $x1 + 8, $y1a, $black );
#$im->line( $x1 + 2, $y2a, $x1 + 8, $y2a, $black );
#$im->line( $x1 + 5, $y1a, $x1 + 5, $y2a, $black );
}
#print "Koords: $x1 + 5, $y1a, $x1 + 5, $y2a\n";
}
$x1 = $xoffset + $x_position;
$y1 = $yoffset + 300;
$im->stringUp( gdLargeFont, $x1 - 3, $y1 + 10 + length( $well[ $row_no ] ) * 8, $well[ $row_no ], $red );
if ( ( $row_no % 2 ) == 1 )
{
if ( $csvout_collected_print_lines[ $row_no ] == "" )
{
$csvout_collected_print_lines[ $row_no ] = $well[ $row_no - 1 ] . "-" . $well[ $row_no ] . "\t";
}
if ( $collected_mean[ $row_no ] ne "" )
{
print CSVOUT $well[ $row_no - 1 ] . "-" . $well[ $row_no ] . "\t" . $collected_mean[ $row_no ] . "\n";
$csvout_collected_print_lines[ $row_no ] .= $collected_mean[ $row_no ] . "\t";
$x_position = $x_position + $balkenbreite;
$hoehe = ( $collected_mean[ $row_no ] * $factor * 300 ) / $max_value_1;
$bar_upper_end = ( ( $collected_mean[ $row_no ] + $collected_stddev[ $row_no ] ) * $factor * 300 ) / $max_value_1;
$bar_lower_end = ( ( $collected_mean[ $row_no ] - $collected_stddev[ $row_no ] ) * $factor * 300 ) / $max_value_1;
print "Hoehe = $hoehe\n";
print "mean[ $row_no ] = " . $collected_mean[ $row_no ] . "\n";
print "stddev[ $row_no ] = " . $collected_stddev[ $row_no ] . "\n";
print "factor = $factor\n";
print "max_value_1 = $max_value_1\n";
print "Bar upper end = $bar_upper_end\n";
print "Bar lower end = $bar_lower_end\n";
#print "Number of collected mean over last two rows = $collected_mean[ $row_no ]\n";
$x1 = $xoffset + $x_position;
$y1 = $yoffset + 300;
$y1a = $yoffset + 300 - $bar_upper_end;
$x2 = $xoffset + $x_position + 10;
$y2 = $yoffset + 300 - $hoehe;
$y2a = $yoffset + 300 - $bar_lower_end;
$im->filledRectangle( $x1, $y2, $x2, $y1, $blue );
$im->string( gdSmallFont, $x2 - 7, $y2 - 12, $no_of_clean_values[ $row_no ] + $no_of_clean_values[ $row_no - 1 ], $black );
if( $var_type == 8 )
{
$im->stringUp( gdSmallFont, $x1 - 2, $y2 + 18, $clean_cells[ $row_no ] + $clean_cells[ $row_no - 1 ], $grey );
}
else
{
if( $var_type == 2 || $var_type == 3 )
{
$im->string( gdSmallFont, $x2 - 7, $y2 - 1, $no_of_stacks_in_filter[ $row_no ] + $no_of_stacks_in_filter[ $row_no - 1 ], $grey );
}
```

```
}
#$im->line( $x1 + 2, $y1a, $x1 + 8, $y1a, $black );
#$im->line( $x1 + 2, $y2a, $x1 + 8, $y2a, $black );
#$im->line( $x1 + 5, $y1a, $x1 + 5, $y2a, $black );
}
}
$x_position = $x_position + $balkenbreite;
}
print "red line height = $red_line_height\n";
$xx = $red_line_height * $factor * 300 / $max_value_1;
#print "Value of red line height to plot: $xx\n";
$im->line( $xoffset - 4, $yoffset + 300 - ( $red_line_height * $factor * 300 ) / $max_value_1 , $xoffset - 1 + 1.5 * $no_of_rows * $balkenbreite, $yoffset + 300 - ( $red_line_height * $factor * 300 ) / $max_value_1,
$red );
# make sure we are writing to a binary stream
binmode PLOT;
print PLOT $im->png;
close PLOT;
#$dummy = <STDIN>;
$var_type = 0;
}
sub plot_values
{
# call: &plot_values( $n, $parameter_name[ $n ], $next_parameter_name[ $n ], $project_name; $name, $given_max_value, $red_line_height );
# plots the data of a single parameter in a PNG Graphics.
# add Standard Deviation bars (if present in next column...)
# next column is marked with " - Standard Deviation".
# if no $given_max_value is provided a 0 (zero) is given.
local $parameter_no = $_[ 0 ];
local $parameter_name = $_[ 1 ];
local $next_parameter_name = $_[ 2 ];
local $project_name = $_[ 3 ];
local $file_name = $_[ 4 ];
local $max_value_1 = $_[ 5 ];
local $red_line_height = $_[ 6 ];
print "at proc start: red line height = $red_line_height\n";
$short = $next_parameter_name;
$short =~ s/ - Standard Deviation//;
$short =~ s/ - Standarddeviation//;
if ( $parameter_name eq $short )
{
$add_stddev_bars = "TRUE";
}
else
{
$add_stddev_bars = "FALSE";
}
local $p = sprintf( "_p%02d", $parameter_no );
$file_name =~ s/\.csv//;
local $outfile_name = "$file_name" . "_" . $parameter_name . ".png";
open( PLOT, "> $outfile_name" ) || print "Could not open $outfile_name\n";
local $balkenbreite = 15;
local $max_balken_hoehe = 300;
local $max_name_length = 7; # "1002000" = well name
local @next_wert = ();
local @wert = &get_data( $parameter_no );
local $no_of_collected_wells = @wert;
if ( $add_stddev_bars eq "TRUE" )
{
@next_wert = &get_data( $parameter_no + 1 );
}
else
{
for ( $xx = 0; $xx < $no_of_collected_wells; $xx++ )
{
$next_wert[ $xx ] = 0;
}
}
local @well = &get_wells;
#print "No of collected wells = $no_of_collected_wells\n";
#print "Value of wert = " . $wert[ 2 ] . "\n";
#print "Value of next_wert = " . $next_wert[ 2 ] . "\n";
#local $max_value_1 = 0;
local $n = 0;
if ( $max_value_1 == 0 )
{
for ( $n = 0; $n < $no_of_collected_wells; $n++ )
{
if ( $max_value_1 < ($wert[ $n ] + $next_wert[ $n ] ) )
{
($max_value_1 = $wert[ $n ] + $next_wert[ $n ] );
}
}
}
#print "max_value_1 = $max_value_1\n"; # 101
local $factor = 1;
if ( $max_value_1 > 50 )
{
while ($max_value_1 > 50 )
{
$max_value_1 = $max_value_1 / 10;
$factor = $factor / 10;
}
}
if ( $max_value_1 <= 5 )
{
if ( $max_value_1 <= 0 )
{
$max_value_1 = 1;
}
while ($max_value_1 <= 5 )
{
$max_value_1 = $max_value_1 * 10;
$factor = $factor * 10;
}
}
#print "Corrected max_value_1 = $max_value_1\n";
local $xoffset = 50 + 8 * length ( $max_value_1 / $factor );
local $yoffset = 70;
local $breite = 2 * $xoffset + $no_of_collected_wells * $balkenbreite;
local $hoehe = 2 * $yoffset + $max_balken_hoehe + $max_name_length * 8;
print "Breite = $breite, Hoehe = $hoehe\n";
local $im = new GD::Image($breite,$hoehe);
# allocate some colors
local $white = $im->colorAllocate(255,255,255); # BZX
local $black = $im->colorAllocate(0,0,0);
local $red = $im->colorAllocate(255,0,0); # DEP
local $blue = $im->colorAllocate(0,0,255);
local $light_blue = $im->colorAllocate(99,101,255); # RK
local $orange = $im->colorAllocate(255,203,0); # FWY
local $yellow = $im->colorAllocate(255,255,0); # C
local $cyan = $im->colorAllocate(255,0,255); # GP
local $green = $im->colorAllocate(0,255,0); # NQST
local $pink = $im->colorAllocate(255,174,173); # AILMV
# make the background transparent and interlaced
#$im->transparent($white);
#$im->interlaced('true');
# Put a black frame around the picture
$im->rectangle(0,0,$breite-1,$hoehe-1,$black);
#$im->rectangle($xoffset, $yoffset, $xoffset+$laenge, $yoffset+$laenge,$black);
# find max value of parameter $parameter_no
$im->string( gdLargeFont, $xoffset, $yoffset - 56, $project_name, $black );
$text_to_plot = $parameter_name . " --- " . "$file_name";
$part1 = $text_to_plot;
$part2 = "";
$max_text_length = int( $breite / 5 );
if ( length( $text_to_plot ) > $max_text_length )
{
$split_at = $max_text_length - 5;
while( substr( $text_to_plot, $split_at, 1 ) ne " " && $split_at > 0 )
{
$split_at--;
}
$split_at++;
$part1 = substr( $text_to_plot, 0, $split_at );
$part2 = substr( $text_to_plot, $split_at );
}
$im->string( gdTinyFont, 10, $yoffset - 40, $part1, $black );
$im->string( gdTinyFont, 10, $yoffset - 30, $part2, $black );
local $x_position = 0;
local $y_position = 0;
local $tick_size = ( int ( $max_value_1 - ( $max_value_1 % 5 ) ) ) / 5;
print "tick size = $tick_size\n";
local $step_size = 0;
```

```
if ( $max_value_1 > 0 )
{
$step_size = ( $tick_size * 300 ) / $max_value_1;
}
#print "step size = $step_size\n";
print "red line height = $red_line_height\n";
$im->line( $xoffset - 5, $yoffset + 300, $xoffset - 5, $yoffset, $black );
#$im->line( $xoffset + 1.5 * $no_of_collected_wells * $balkenbreite + 5, $yoffset + 300, $xoffset + 1.5 * $no_of_collected_wells * $balkenbreite + 5, $yoffset, $black );
$im->line( $xoffset + $no_of_collected_wells * $balkenbreite, $yoffset + 300, $xoffset + $no_of_collected_wells * $balkenbreite, $yoffset, $black );
#$im->line( $xoffset - 5, $yoffset + 300, $xoffset + $no_of_collected_wells * $balkenbreite, $yoffset + 300, $black );
$xx = $red_line_height * $factor * 300 / $max_value_1;
#print "Value of red line height to plot: $xx\n";
local $y_pos = 0;
while( $y_pos <= 300 && $step_size > 0 )
{
$label = $y_position / $factor;
$im->string( gdLargeFont, $xoffset - 12 - 8 * length( $label ) , $yoffset + 292 - $y_pos, $label, $black );
$im->line( $xoffset - 10, $yoffset + 300 - $y_pos, $xoffset - 5, $yoffset + 300 - $y_pos, $black );
$im->string( gdLargeFont, $xoffset + $no_of_collected_wells * $balkenbreite + 7 , $yoffset + 292 - $y_pos, $label, $black );
$im->line( $xoffset + $no_of_collected_wells * $balkenbreite , $yoffset + 300 - $y_pos, $xoffset + $no_of_collected_wells * $balkenbreite + 5, $yoffset + 300 - $y_pos, $black );
#$a = $xoffset - 10;
#$b = $yoffset + 300 - $y_pos;
#$c = $xoffset - 5;
#$d = $yoffset + 300 - $y_pos;
#print "Drawing line at: " . $a . "-" . $b . "-" . $c . "-" . $d . "\n";
$y_pos = $y_pos + $step_size;
$y_position = $y_position + $tick_size;
}
local $well_no = 0;
for( $well_no = 0; $well_no < $no_of_collected_wells; $well_no++ )
{
# find indices of $well
#print "Working on " . $well[ $well_no ] . "\n";
# first index = horizontal = value, second index = vertical = well
if ( $wert[ $well_no ] ne "" ) # why???
{
if ( $wert[ $well_no ] ne "" )
{
$hoehe = ( $wert[ $well_no ] * $factor * 300 ) / $max_value_1;
$bar_upper_end = ( ( $wert[ $well_no ] + $next_wert[ $well_no ] ) * $factor * 300 ) / $max_value_1;
$bar_lower_end = ( ( $wert[ $well_no ] - $next_wert[ $well_no ] ) * $factor * 300 ) / $max_value_1;
print "Hoehe = $hoehe\n";
print "wert[ well_no ] = " . $wert[ $well_no ] . "\n";
print "next_wert[ well_no ] = " . $next_wert[ $well_no ] . "\n";
print "factor = $factor\n";
print "max_value_1 = $max_value_1\n";
print "Bar upper end = $bar_upper_end\n";
print "Bar lower end = $bar_lower_end\n";
$x1 = $xoffset + $x_position;
$y1 = $yoffset + 300;
$y1a = $yoffset + 300 - $bar_upper_end;
$x2 = $xoffset + $x_position + 10;
$y2 = $yoffset + 300 - $hoehe;
$y2a = $yoffset + 300 - $bar_lower_end;
$im->filledRectangle( $x1, $y2, $x2, $y1, $light_blue );
if ( $add_stddev_bars eq "TRUE" )
{
$im->line( $x1 + 2, $y1a, $x1 + 8, $y1a, $black );
$im->line( $x1 + 2, $y2a, $x1 + 8, $y2a, $black );
$im->line( $x1 + 5, $y1a, $x1 + 5, $y2a, $black );
}
}
#print "Koords: $x1 + 5, $y1a, $x1 + 5, $y2a\n";
#if ( $parameter_no == 66 ) { $dummy = <STDIN>; }
}
$x1 = $xoffset + $x_position;
$y1 = $yoffset + 300;
$im->stringUp( gdLargeFont, $x1 - 3, $y1 + 10 + length( $well[ $well_no ] ) * 8, $well[ $well_no ], $red );
$x_position = $x_position + $balkenbreite;
}
$im->line( $xoffset - 4, $yoffset + 300 - ( $red_line_height * $factor * 300 ) / $max_value_1 , $xoffset - 1 + $no_of_collected_wells * $balkenbreite, $yoffset + 300 - ( $red_line_height * $factor * 300 ) /
$max_value_1, $red );
$im->line( $xoffset - 5, $yoffset + 300, $xoffset + $no_of_collected_wells * $balkenbreite, $yoffset + 300, $black );
# make sure we are writing to a binary stream
binmode PLOT;
print PLOT $im->png;
close PLOT;
}
print "Name of the project: ";
$project_name = <STDIN>;
chomp $project_name;
# open for Windows:
open( FILELIST, "dir *.csv /B |") || die print "could not open csv files.\n";
# open for UNIX:
#open( FILELIST, "ls |" ) || die print "could not open csv files.\n";
while( <FILELIST> )
{
$zeile = $_;
chomp $zeile;
print "$zeile\n";
if ( $zeile =~ /\.csv$/i )
{
push( @dateinamen, $zeile );
}
}
close FILELIST;
foreach $name ( @dateinamen )
{
@data = ();
$reading_data = "FALSE";
open( CSVOUT, ">$name" . ".means.txt" ) || die print "could not open $name" . ".means.txt\n";
open( CSV, "$name" ) || die print "Could not open $name\n";
```

```
print "reading file $name\n";
while ( <CSV> )
{
$zeile = $_;
chomp $zeile;
@item = split( /;/, $zeile );
if ( $item[ 0 ] eq "WellIndex" )
{
@parameter_name = @item;
$reading_data = "TRUE";
}
else
{
if ( $reading_data eq "TRUE" )
{
if ( $item[ 0 ] =~ /^\d\d\d\d\d\d\d$/ )
{
push( @data, $zeile );
push( @well, $item[ 0 ] );
print "$zeile\n";
}
else
{
$reading_data eq "FALSE";
}
}
}
}
# messwerte sortieren
#@value = &get_values( 1 );
# plots erstellen
#$no_of_values = @parameter_name;
#for ( $n = 46; $n < $no_of_values; $n++ )
# {
# &plot_values( $n, $parameter_name[ $n ], $parameter_name[ $n + 1 ], $project_name, $name );
# }
$groesse_tabelle = @parameter_name;
print CSVOUT "\nfile: $name\n\n";
if ( $groesse_tabelle == 74 )
{
&plot_values( 64, $parameter_name[ 64 ], $parameter_name[ 65 ], $project_name, $name, 1.5, 0.76 );
&plot_values( 66, $parameter_name[ 66 ], $parameter_name[ 67 ], $project_name, $name, 1.5, 0.96 );
&plot_values( 68, $parameter_name[ 68 ], $parameter_name[ 69 ], $project_name, $name, 0.6, 0.33 );
&plot_secondary_values( "Average Number of Spots per Picture projected", 5, $name, 520 );
&plot_secondary_values( "Average Number of Spots per Picture counted", 5, $name, 360 );
&plot_secondary_values( "Average Number of Spots per Cell", 5, $name, 7.7 );
print CSVOUT "\n";
}
if ( $groesse_tabelle == 64 )
{
&plot_values( 54, $parameter_name[ 54 ], $parameter_name[ 55 ], $project_name, $name, 1.5, 0.76 );
&plot_values( 56, $parameter_name[ 56 ], $parameter_name[ 57 ], $project_name, $name, 1.5, 0.96 );
&plot_values( 58, $parameter_name[ 58 ], $parameter_name[ 59 ], $project_name, $name, 0.6, 0.33 );
&plot_secondary_values( "Average Number of Spots per Picture projected", 4, $name, 520 );
&plot_secondary_values( "Average Number of Spots per Picture counted", 4, $name, 360 );
&plot_secondary_values( "Average Number of Spots per Cell", 4, $name, 7.7 );
print CSVOUT "\n";
}
close CSV;
}
print CSVOUT "\n";
print CSVOUT "Well No\tAverage Number of Spots per Picture projected\tAverage Number of Spots per Picture counted\tAverage Number of Spots per Cell\n";
foreach $print_line ( @csvout_print_lines )
{
if ($print_line ne "" )
{
print CSVOUT $print_line . "\n";
}
}
print CSVOUT "\n";
print CSVOUT "Collected Wells No\tAverage Number of Spots per Picture projected\tAverage Number of Spots per Picture counted\tAverage Number of Spots per Cell\n";
foreach $print_line ( @csvout_collected_print_lines )
{
if ($print_line ne "" )
{
print CSVOUT $print_line . "\n";
}
}
close CSVOUT;
#end of script: PERL script for Endomembrane script B
```

Supplemental Data S4. PERL script for "Plasma membrane microdomain script". Script for reformatting and plotting data collected by the plasma membrane microdomain script.

```
# name of script:PERL script for Plasma membrane microdomain script
# Program merge.pl
# written by Kurt Stueber, November 2006
# to execute this script rename .txt to .pl
# add two csv-files together, original output from the program MPIZ_leaves_e.script
# acapella script.
use Time::Local;
use GD;
( $sec, $min, $hour, $mday, $mon, $year, $yday, $isdst) = localtime( time );
# This is the Millenium bug:
$jahr = 1900 + $year;
%monat = ( '0', 'Jan', '1', 'Feb', '2', 'Mar', '3', 'Apr',
'4', 'May', '5', 'Jun', '6', 'Jul', '7', 'Aug',
'8', 'Sep', '9', 'Oct', '10', 'Nov', '11', 'Dec' );
$month = $monat{ $mon };
sub add_values_and_print
{
local $sum = 0;
local $value1 = $_[0];
local $value2 = $_[1];
if ( ( $value1 eq "nan" || $value1 eq "" ) &&
( $value2 eq "nan" || $value2 eq "" ) )
{
$sum = "";
}
else
{
if ( $value1 ne "nan" && $value1 ne "" )
{
$sum = $value1;
}
else
{
$sum = 0;
}
if ( $value2 ne "nan" && $value2 ne "" )
{
$sum = $sum + $value2;
}
}
print OUTFILE "$sum;";
return $sum;
}
sub add_mean_values_and_print
{
local $sum = 0;
local $value1 = $_[0];
local $value2 = $_[1];
local $factor1 = $_[2];
local $factor2 = $_[3];
if ( ( $value1 eq "nan" || $value1 eq "" ) &&
( $value2 eq "nan" || $value2 eq "" ) )
{
$sum = "";
}
else
{
if ( $value1 ne "nan" && $value1 ne "" )
{
$sum = $factor1 * $value1;
}
else
{
$sum = 0;
}
if ( $value2 ne "nan" && $value2 ne "" )
{
$sum = $sum + $factor2 * $value2;
}
if( $factor1 + $factor2 > 0 )
{
$sum = $sum / ( $factor1 + $factor2 );
}
}
print OUTFILE "$sum;";
return $sum;
}
sub add_values
{
local $sum = 0;
local $value1 = $_[0];
local $value2 = $_[1];
local $position1 = $_[2];
local $position2 = $_[3];
if ( ( $value1 eq "nan" || $value1 eq "" ) &&
( $value2 eq "nan" || $value2 eq "" ) )
{
$sum = "";
}
else
{
if ( $value1 ne "nan" && $value1 ne "" && $position1 > -1 )
{
$sum = $value1;
}
else
{
$sum = 0;
```

```
}
if ( $value2 ne "nan" && $value2 ne "" && $position2 > -1 )
{
$sum = $sum + $value2;
}
}
return $sum;
}
sub add_mean_values
{
local $sum = 0;
local $sum1 = 0;
local $sum2 = 0;
local $value1 = $_[0];
local $value2 = $_[1];
local $factor1 = $_[2];
local $factor2 = $_[3];
local $position1 = $_[4];
local $position2 = $_[5];
if ( ( $value1 eq "nan" || $value1 eq "" ) &&
( $value2 eq "nan" || $value2 eq "" ) )
{
$sum = "";
}
else
{
if ( $value1 ne "nan" && $value1 ne "" && $position1 > -1 )
{
$sum1 = $factor1 * $value1;
}
else
{
$sum1 = "";
}
if ( $value2 ne "nan" && $value2 ne "" && $position2 > -1 )
{
$sum2 = $factor2 * $value2;
}
else
{
$sum2 = "";
}
if( $factor1 > 0 && $sum1 ne "" )
{
if ( $factor2 > 0 && $sum2 ne "" )
{
$sum = ( $sum1 + $sum2 ) / ( $factor1 + $factor2 );
}
else
{
$sum = $value1;
}
}
else
{
if ( $factor2 > 0 && $sum2 ne "" )
{
$sum = $value2;
}
else
{
$sum = "";
}
}
}
return $sum;
}
sub swap_values_1
{
local $n = 0;
local $temp = "";
local $pos = $_[ 0 ];
# first index = horizontal = value, second index = vertical = well
print "Swapping at $pos\n";
for( $n = 0; $n < $no_of_values; $n++ )
{
$temp = $values_1[ $n ][ $pos ];
$values_1[ $n ][ $pos ] = $values_1[ $n ][ $pos + 1 ];
$values_1[ $n ][ $pos + 1 ] = $temp;
}
print "Swapped: " . $values_1[ 0 ][ $pos ] . " and " . $values_1[ 0 ][ $pos + 1 ] . "\n";
}
sub swap_values_2
{
local $n = 0;
local $temp = "";
local $pos = $_[ 0 ];
# first index = horizontal = value, second index = vertical = well
for( $n = 0; $n < $no_of_values; $n++ )
{
$temp = $values_2[ $n ][ $pos ];
$values_2[ $n ][ $pos ] = $values_2[ $n ][ $pos + 1 ];
$values_2[ $n ][ $pos + 1 ] = $temp;
}
}
sub get_col_row
{
# call: ( $col, $row ) = &get_col_row( "2011000" );
# read col and row from well name: 2011000
# ^ ^^
# row col
local $well = $_[ 0 ];
local $row = 0;
local $col = 0;
$row = substr( $well, 0, 1 );
$col = substr( $well, 2, 2 );
return ($row, $col );
}
sub get_well
{
# call: $well = &get_well( $row, $col );
local $row = $_[ 0 ];
local $col = $_[ 1 ];
local $result = "0000000";
$result = sprintf( "%01d", $row ) . "0" . sprintf( "%02d", $col ) . "000";
return $result;
}
sub get_second_well_info
{
# returns true if well is second well of a pair of wells
# pairs of well are side be side, have the same $col but consecutive $row
# smallest $row is $rowmin[ $col ];
# call: $is_second_well = &get_second_well_info( $this_col, $this_row );
local $col = $_[0];
local $row = $_[1];
local $result = "FALSE";
local $modulus = $rowmin[ $col ] % 2;
if ( $modulus ne $row % 2 )
{
$result = "TRUE";
}
else
{
$result = "FALSE";
}
return $result;
}
sub get_position_of_well
{
# get no of well in array @collected_wells
local $well = $_[ 0 ];
local $n = 0;
local $no_of_wells = @collected_wells;
local $result = -1;
for ( $n = 0; $n < $no_of_wells; $n++ )
{
if ( $well eq $collected_wells[ $n ] )
{
$result = $n;
}
}
return $result;
}
```

```
sub plot_parameter
{
# call: &plot_parameter( $parameter_no );
# plots the data of a single parameter in a PNG Graphics.
local $parameter_no = $_[ 0 ];
local $p = sprintf( "_p%02d", $parameter_no );
local $outfile_name = "$file1_core" . "_" . "$file2_core" . $p . ".png";
open( PLOT, "> $outfile_name" ) || print "Could not open $outfile_name\n";
local $balkenbreite = 15;
local $max_balken_hoehe = 300;
local $max_name_length = 15; # "1001000-1002000" = well name
print "No of collected wells = $no_of_collected_wells\n"; # 20
local $max_value_1 = 0;
local $n = 0;
for ( $n = 0; $n < $no_of_collected_wells; $n++ )
{
if ( $max_value_1 < $values_3[ $parameter_no ][ $n ] )
{
$max_value_1 = $values_3[ $parameter_no ][ $n ];
}
if ( $max_value_1 < $values_4[ $parameter_no ][ $n ] )
{
$max_value_1 = $values_4[ $parameter_no ][ $n ];
}
}
print "max_value_1 = $max_value_1\n"; # 101
local $factor = 1;
if ( $max_value_1 > 50 )
{
while ($max_value_1 > 50 )
{
$max_value_1 = $max_value_1 / 10;
$factor = $factor / 10;
}
}
if ( $max_value_1 <= 5 )
{
if ( $max_value_1 <= 0 )
{
$max_value_1 = 1;
}
while ($max_value_1 <= 5 )
{
$max_value_1 = $max_value_1 * 10;
$factor = $factor * 10;
}
}
print "Corrected max_value_1 = $max_value_1\n";
local $xoffset = 50 + 8 * length ( $max_value_1 / $factor );
local $yoffset = 50;
local $breite = 2 * $xoffset + 1.5 * $no_of_collected_wells * $balkenbreite;
local $hoehe = 2 * $yoffset + $max_balken_hoehe + $max_name_length * 8;
print "Breite = $breite, Hoehe = $hoehe\n";
local $im = new GD::Image($breite,$hoehe);
# allocate some colors
local $white = $im->colorAllocate(255,255,255); # BZX
local $black = $im->colorAllocate(0,0,0);
local $red = $im->colorAllocate(255,0,0); # DEP
local $blue = $im->colorAllocate(0,0,255);
local $light_blue = $im->colorAllocate(99,101,255); # RK
local $orange = $im->colorAllocate(255,203,0); # FWY
local $yellow = $im->colorAllocate(255,255,0); # C
local $cyan = $im->colorAllocate(255,0,255); # GP
local $green = $im->colorAllocate(0,255,0); # NQST
local $pink = $im->colorAllocate(255,174,173); # AILMV
# make the background transparent and interlaced
#$im->transparent($white);
#$im->interlaced('true');
# Put a black frame around the picture
$im->rectangle(0,0,$breite-1,$hoehe-1,$black);
#$im->rectangle($xoffset, $yoffset, $xoffset+$laenge, $yoffset+$laenge,$black);
# find max value of parameter $parameter_no
$im->string( gdLargeFont, $xoffset, $yoffset - 20, $parameter_name[ $parameter_no ] . " --- " . "$file1_core" . "_" . "$file2_core" , $black );
local $x_position = 0;
local $y_position = 0;
local $tick_size = ( int ( $max_value_1 - ( $max_value_1 % 5 ) ) ) / 5;
print "tick size = $tick_size\n";
local $step_size = 0;
if ( $max_value_1 > 0 )
{
$step_size = ( $tick_size * 300 ) / $max_value_1;
}
print "step size = $step_size\n";
$im->line( $xoffset - 5, $yoffset + 300, $xoffset - 5, $yoffset, $black );
$im->line( $xoffset + 1.5 * $no_of_collected_wells * $balkenbreite + 5, $yoffset + 300, $xoffset + 1.5 * $no_of_collected_wells * $balkenbreite + 5, $yoffset, $black );
local $y_pos = 0;
while( $y_pos <= 300 && $step_size > 0 )
{
$label = $y_position / $factor;
$im->string( gdLargeFont, $xoffset - 12 - 8 * length( $label ) , $yoffset + 292 - $y_pos, $label, $black );
$im->line( $xoffset - 10, $yoffset + 300 - $y_pos, $xoffset - 5, $yoffset + 300 - $y_pos, $black );
$im->string( gdLargeFont, $xoffset + 1.5 * $no_of_collected_wells * $balkenbreite + 12 , $yoffset + 292 - $y_pos, $label, $black );
$im->line( $xoffset + 1.5 * $no_of_collected_wells * $balkenbreite + 5, $yoffset + 300 - $y_pos, $xoffset + 1.5 * $no_of_collected_wells * $balkenbreite + 10, $yoffset + 300 - $y_pos, $black );
#$a = $xoffset - 10;
#$b = $yoffset + 300 - $y_pos;
#$c = $xoffset - 5;
#$d = $yoffset + 300 - $y_pos;
#print "Drawing line at: " . $a . "-" . $b . "-" . $c . "-" . $d . "\n";
$y_pos = $y_pos + $step_size;
$y_position = $y_position + $tick_size;
}
local $well_no = 0;
```

```
for( $well_no = 0; $well_no < $no_of_collected_wells; $well_no++ )
{
# find indices of $well
print "Working on " . $collected_wells[ $well_no ] . "\n";
# first index = horizontal = value, second index = vertical = well
if ( $values_3[ 0 ][ $well_no ] ne "" ) # why???
{
if ( $values_3[ $parameter_no ][ $well_no ] ne "" )
{
$hoehe = ( $values_3[ $parameter_no ][ $well_no ] * $factor * 300 ) / $max_value_1;
print "Hoehe = $hoehe\n";
$x1 = $xoffset + $x_position;
$y1 = $yoffset + 300;
$x2 = $xoffset + $x_position + 10;
$y2 = $yoffset + 300 - $hoehe;
$im->filledRectangle( $x1, $y2, $x2, $y1, $light_blue );
}
#print "Koords: $x1, $y1, $x2, $y2\n";
}
$x1 = $xoffset + $x_position;
$y1 = $yoffset + 300;
$im->stringUp( gdLargeFont, $x1, $y1 + 10 + length( $values_3[ 0 ][ $well_no ] ) * 8, $values_3[ 0 ][ $well_no ], $red );
$x_position = $x_position + $balkenbreite;
( $this_col, $this_row ) = &get_col_row( $collected_wells[ $well_no ] );
$is_second_well = "FALSE";
$is_second_well = &get_second_well_info( $this_col, $this_row );
# $position1 = no of previous well in &collected_wells
# $position2 = no of this well in &collected_wells
if ( $is_second_well eq "TRUE" )
{
if ( $values_4[ $parameter_no ][ $well_no ] ne "" )
{
$hoehe = ( $values_4[ $parameter_no ][ $well_no ] * $factor * 300 ) / $max_value_1;
print "Hoehe = $hoehe\n";
$x1 = $xoffset + $x_position;
$y1 = $yoffset + 300;
$x2 = $xoffset + $x_position + 10;
$y2 = $yoffset + 300 - $hoehe;
$im->filledRectangle( $x1, $y2, $x2, $y1, $orange );
}
#print "Koords: $x1, $y1, $x2, $y2\n";
$x1 = $xoffset + $x_position;
$y1 = $yoffset + 300;
$im->stringUp( gdLargeFont, $x1, $y1 + 10 + length( $values_4[ 0 ][ $well_no ] ) * 8, $values_4[ 0 ][ $well_no ], $red );
$x_position = $x_position + $balkenbreite;
}
}
# make sure we are writing to a binary stream
binmode PLOT;
print PLOT $im->png;
close PLOT;
```

```
}
###################################################################
############## MAIN start ############################################
###################################################################
print " Give name of first .CSV file: ";
$file1_name = <STDIN>;
chomp $file1_name;
open( FILE1, "$file1_name" ) || die print "Could not open first file.\n";
print "Give name of second .CSV file: ";
$file2_name = <STDIN>;
chomp $file2_name;
open( FILE2, "$file2_name" ) || die print "Could not open second file.\n";
$file1_core = $file1_name;
$file1_core =~ s/\.csv//;
if ( $file1_core =~ /\// )
{
$file1_core =~ s/.*\///;
}
if ( $file1_core =~ /\\/ )
{
$file1_core =~ s/.*\\//;
}
$file2_core = $file2_name;
$file2_core =~ s/\.csv//;
if ( $file2_core =~ /\// )
{
$file2_core =~ s/.*\///;
}
if ( $file2_core =~ /\\/ )
{
$file2_core =~ s/.*\\//;
}
$output_file_name = "$file1_core" . "_" . "$file2_core" . ".csv";
open( OUTFILE, "> $output_file_name" ) || die print "Could open output file.\n";
# Structure of .csv file:
#Input parameters;
#label;value
#Stack No;2
#Number Of Channels;1
#Zplanes in stack;31
#ShowIllustrations;1
#MinStdDev;150
#spotMinimumArea;25
#Threshold Adjustment;1.5
#Minimum Nuclei Distance;7
#Nuclear Splitting Adjustment;7
#Individual Threshold Adjustment;0.4
#Minimum Nuclear Area;120
#Minimum Nuclear Contrast;0.1
#Output results;;;;;;;;;;;;;;;;;
#WellIndex;Number of Spots;Number of Leaf Cells;Number of Type 2 objects;Number of Haustoria;Average intensity of spots;Average area of spots;Total integrated spot signal, over all spots;Total integrated spot
```

```
signal background subtracted, over all spots;Average length of spots;Average Half width of spots;Average width to length ratio of spots;Average roundness of spots;Average contrast of spots;Average PeakIntensity
of spots;Number of spots per cell;Total number of Stacks analyzed;Number of valid Stacks
#2002000;8;25;5;0;1986.75;48.5;767880;501030;7.75;2.8;0.728003;0.933349;0.478567;3099.63;0.32;1;1
#2003000;13;29;14;1;1840.1;53.2308;1329515;952547;8.87692;2.90769;0.658923;0.914065;0.465579;2452.08;0.448276;1;1
# etc....
# Alternate structure of .csv file (as created from OPERA directly):
#Barcode ="0104a"
#Area = "1"
# 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
#Row;Column;"Number of Spots";" ";"Number of Leaf Cells";" ";"Number of Haustoria";" ";"Number Of Stomata";" ";"Number Of gaps between leaf cells";" ";"Number of spots per area";" ";"Number of spots per cell";"
";"Total integrated spot signal per area";" ";"Total integrated spot signal per area background subtracted";" ";"Average intensity of spots";" ";"Average area of spots";" ";"Total integrated spot signal, over all spots";"
";"Total integrated spot signal background subtracted, over all spots";" ";"Average length of spots";" ";"Average Half width of spots";" ";"Average width to length ratio of spots";" ";"Average roundness of spots";"
";"Average contrast of spots";" ";"Average PeakIntensity of spots";" ";"Total Cell Area";" ";"Total number of Stacks analyzed";" ";"Number of valid Stacks";" ";
#2;2;"Number of Spots";"32";"Number of Leaf Cells";"104";"Number of Haustoria";"5";"Number Of Stomata";"4";"Number Of gaps between leaf cells";"20";"Number of spots per area";"0,000024";"Number of spots per
cell";"0,307692";"Total integrated spot signal per area";"3,534095";"Total integrated spot signal per area background subtracted";"2,465794";"Average intensity of spots";"2713,725411";"Average area of
spots";"51,343750";"Total integrated spot signal, over all spots";"4716363";"Total integrated spot signal background subtracted, over all spots";"3290680,339755";"Average length of spots";"8,734375";"Average Half
width of spots";"2,725000";"Average width to length ratio of spots";"0,635560";"Average roundness of spots";"0,914391";"Average contrast of spots";"0,482621";"Average PeakIntensity of
spots";"3815,812500";"Total Cell Area";"1334532";"Total number of Stacks analyzed";"4";"Number of valid Stacks";"4";
#2;3;"Number of Spots";"15";"Number of Leaf Cells";"78";"Number of Haustoria";"2";"Number Of Stomata";"5";"Number Of gaps between leaf cells";"13";"Number of spots per area";"0,000012";"Number of spots per
cell";"0,192308";"Total integrated spot signal per area";"1,473442";"Total integrated spot signal per area background subtracted";"0,957851";"Average intensity of spots";"2358,921365";"Average area of
spots";"53,666667";"Total integrated spot signal, over all spots";"1898205";"Total integrated spot signal background subtracted, over all spots";"1233978,779159";"Average length of spots";"9,426667";"Average Half
width of spots";"2,866667";"Average width to length ratio of spots";"0,626293";"Average roundness of spots";"0,889244";"Average contrast of spots";"0,475333";"Average PeakIntensity of
spots";"3295,066667";"Total Cell Area";"1288279";"Total number of Stacks analyzed";"4";"Number of valid Stacks";"4";
#2;4;"Number of Spots";"23";"Number of Leaf Cells";"105";"Number of Haustoria";"0";"Number Of Stomata";"9";"Number Of gaps between leaf cells";"26";"Number of spots per area";"0,000017";"Number of spots per
cell";"0,219048";"Total integrated spot signal per area";"1,796929";"Total integrated spot signal per area background subtracted";"1,255225";"Average intensity of spots";"1998,696354";"Average area of
spots";"46,739130";"Total integrated spot signal, over all spots";"2381495";"Total integrated spot signal background subtracted, over all spots";"1663567,653401";"Average length of spots";"8,460870";"Average Half
width of spots";"2,652174";"Average width to length ratio of spots";"0,642901";"Average roundness of spots";"0,904976";"Average contrast of spots";"0,440524";"Average PeakIntensity of
spots";"2762,608696";"Total Cell Area";"1325314";"Total number of Stacks analyzed";"4";"Number of valid Stacks";"4";
$headline = "";
$no_of_values = 0;
$no_of_items = 0;
@values_1 = ();
@wellindex_1 = (); # 0
@rowindex_1 = ();
@colindex_1 = ();
@no_of_spots_1 = (); # 1
@no_of_leaf_cells_1 = (); #2
@no_of_haustoria_1 =(); #3
@no_of_stomata_1 = (); #4
@no_of_gaps_between_leaf_cells_1 = (); #5
@no_of_spots_per_area_1 = (); #6
@no_of_spots_per_cell_1 = (); #7
@total_int_spot_signal_per_area_1 = (); #8
@total_int_spot_signal_per_area_bgd_subtracted_1 = (); #9
@av_intensity_of_spots_1 = (); # 10
@av_area_of_spots_1 = (); #11
@total_int_spot_signal_over_all_spots_1 = (); #12
@total_int_spot_signal_over_all_spots_bgd_subtracted_1 = (); #13
@av_length_of_spots_1 = (); #14
@av_half_width_of_spots_1 = (); #15
@av_width_to_length_ratio_of_spots_1 = (); #16
```

```
@av_roundness_of_spots_1 = (); #17
@av_contrast_of_spots_1 = (); #18
@av_peakintensity_of_spots_1 = (); #19
@total_cell_area_1 =(); #20
@total_number_of_stacks_analyzed_1 = (); #21
@no_of_valid_stacks_1 = (); #22
@total_int_spot_signal_per_spot = (); #23
@values_2 = ();
@wellindex_2 = (); # 0
@rowindex_2 = ();
@colindex_2 = ();
@values_3 = (); # "sum" of @values_1 and @values_2
@keep_values_1 = ();
@keep_values_2 = ();
$reading_Wells = "FALSE";
$no = 0;
while (<FILE1>)
{
$zeile = $_;
chomp( $zeile );
print OUTFILE "$zeile\n";
#print "--$zeile--\n";
@item = split( /\;/, $zeile );
$no_of_elements = @item;
$no_of_items = 24; ### formerly 22 !!! ###
for( $nn = 0; $nn < $no_of_elements; $nn++ )
{
$item[ $nn ] =~ s/\"//g;
$item[ $nn ] =~ s/,/\./g;
}
#if( $item[0] eq "WellIndex" )
if( $item[0] eq "Row" )
{
$headline = $zeile;
$reading_Wells = "TRUE";
$no_of_values = $no_of_items;
}
else
{
if( $reading_Wells eq "TRUE" && $item[ 0 ] ne "" )
{
$values_1[ 0 ][ $no ] = sprintf( "%01d", $item[0] ) . "0" . sprintf( "%02d", $item[1] ) . "000";
$values_1[ 1 ][ $no ] = $item[ 3 ];
$values_1[ 2 ][ $no ] = $item[ 5 ];
$values_1[ 3 ][ $no ] = $item[ 7 ];
$values_1[ 4 ][ $no ] = $item[ 9 ];
$values_1[ 5 ][ $no ] = $item[ 11 ];
$values_1[ 6 ][ $no ] = $item[ 13 ];
$values_1[ 7 ][ $no ] = $item[ 15 ];
$values_1[ 8 ][ $no ] = $item[ 17 ];
$values_1[ 9 ][ $no ] = $item[ 19 ];
$values_1[ 10 ][ $no ] = $item[ 21 ];
$values_1[ 11 ][ $no ] = $item[ 23 ];
$values_1[ 12 ][ $no ] = $item[ 25 ];
$values_1[ 13 ][ $no ] = $item[ 27 ];
$values_1[ 14 ][ $no ] = $item[ 29 ];
$values_1[ 15 ][ $no ] = $item[ 31 ];
$values_1[ 16 ][ $no ] = $item[ 33 ];
$values_1[ 17 ][ $no ] = $item[ 35 ];
$values_1[ 18 ][ $no ] = $item[ 37 ];
$values_1[ 19 ][ $no ] = $item[ 39 ];
$values_1[ 20 ][ $no ] = $item[ 41 ];
$values_1[ 21 ][ $no ] = $item[ 43 ];
$values_1[ 22 ][ $no ] = $item[ 45 ];
$values_1[ 23 ][ $no ] = $item[ 47 ];
#for ( $item_no = 0; $item_no < $no_of_items; $item_no++ )
# {
# $values_1[ $item_no ][ $no ] = $item[ $item_no ];
# }
$no++;
}
else
{
$reading_Wells = "FALSE";
}
}
}
close FILE1;
#sort first list of values:
$no_of_wells_1 = $no; # Z.B. 20 wells, enumerated 0 to 19
print "no of wells 1 = $no_of_wells_1\n";
#for ( $n = 0; $n < $no_of_wells_1; $n++ )
# {
# for ( $m = 0; $m < $no_of_wells_1 - 1; $m++ )
# {
#
# if (defined $values_1[ 0 ][ $m ] && defined $values_1[ 0 ][ $m + 1 ] )
# {
# if ( $values_1[ 0 ][ $m ] > $values_1[ 0 ][ $m + 1 ] )
# {
# &swap_values_1( $m );
# }
# }
# }
# }
for ( $n = 0; $n < $no_of_wells_1; $n++ )
{
for ( $m = 0; $m < $no_of_values; $m++ )
{
print $values_1[ $m ][ $n ] . ";";
}
print "\n";
```

```
}
$no = 0;
$reading_Wells = "FALSE";
while (<FILE2>)
{
$zeile = $_;
chomp( $zeile );
print OUTFILE "$zeile\n";
#print "--$zeile--\n";
@item = split( /\;/, $zeile );
$no_of_elements = @item;
$no_of_items = 22;
for( $nn = 0; $nn < $no_of_elements; $nn++ )
{
$item[ $nn ] =~ s/\"//g;
$item[ $nn ] =~ s/,/\./g;
}
#print "no of items = $no_of_items\n";
#if( $item[0] eq "WellIndex" )
if( $item[0] eq "Row" )
{
$reading_Wells = "TRUE";
#@parameter_name = @item;
#Row;Column;"Number of Spots";" ";"Number of Leaf Cells";" ";"Number of Haustoria";" ";"Number Of Stomata";" ";"Number Of gaps between leaf cells";" ";"Number of spots per area";" ";"Number of spots per cell";"
";"Total integrated spot signal per area";" ";"Total integrated spot signal per area background subtracted";" ";"Average intensity of spots";" ";"Average area of spots";" ";"Total integrated spot signal, over all spots";"
";"Total integrated spot signal background subtracted, over all spots";" ";"Average length of spots";" ";"Average Half width of spots";" ";"Average width to length ratio of spots";" ";"Average roundness of spots";"
";"Average contrast of spots";" ";"Average PeakIntensity of spots";" ";"Total Cell Area";" ";"Total number of Stacks analyzed";" ";"Number of valid Stacks";" ";
$parameter_name[ 0 ] = "WellIndex";
$parameter_name[ 1 ] = $item[ 2 ];
$parameter_name[ 2 ] = $item[ 4 ];
$parameter_name[ 3 ] = $item[ 6 ];
$parameter_name[ 4 ] = $item[ 8 ];
$parameter_name[ 5 ] = $item[ 10 ];
$parameter_name[ 6 ] = $item[ 12 ];
$parameter_name[ 7 ] = $item[ 14 ];
$parameter_name[ 8 ] = $item[ 16 ];
$parameter_name[ 9 ] = $item[ 18 ];
$parameter_name[ 10 ] = $item[ 20 ];
$parameter_name[ 11 ] = $item[ 22 ];
$parameter_name[ 12 ] = $item[ 24 ];
$parameter_name[ 13 ] = $item[ 26 ];
$parameter_name[ 14 ] = $item[ 28 ];
$parameter_name[ 15 ] = $item[ 30 ];
$parameter_name[ 16 ] = $item[ 32 ];
$parameter_name[ 17 ] = $item[ 34 ];
$parameter_name[ 18 ] = $item[ 36 ];
$parameter_name[ 19 ] = $item[ 38 ];
$parameter_name[ 20 ] = $item[ 40 ];
$parameter_name[ 21 ] = $item[ 42 ];
$parameter_name[ 22 ] = $item[ 44 ];
$parameter_name[ 23 ] = $item[ 46 ];
```

```
}
else
{
if( $reading_Wells eq "TRUE" && $item[ 0 ] ne "" )
{
print "Storing values\n";
$values_2[ 0 ][ $no ] = sprintf( "%01d", $item[0] ) . "0" . sprintf( "%02d", $item[1] ) . "000";
$values_2[ 1 ][ $no ] = $item[ 3 ];
$values_2[ 2 ][ $no ] = $item[ 5 ];
$values_2[ 3 ][ $no ] = $item[ 7 ];
$values_2[ 4 ][ $no ] = $item[ 9 ];
$values_2[ 5 ][ $no ] = $item[ 11 ];
$values_2[ 6 ][ $no ] = $item[ 13 ];
$values_2[ 7 ][ $no ] = $item[ 15 ];
$values_2[ 8 ][ $no ] = $item[ 17 ];
$values_2[ 9 ][ $no ] = $item[ 19 ];
$values_2[ 10 ][ $no ] = $item[ 21 ];
$values_2[ 11 ][ $no ] = $item[ 23 ];
$values_2[ 12 ][ $no ] = $item[ 25 ];
$values_2[ 13 ][ $no ] = $item[ 27 ];
$values_2[ 14 ][ $no ] = $item[ 29 ];
$values_2[ 15 ][ $no ] = $item[ 31 ];
$values_2[ 16 ][ $no ] = $item[ 33 ];
$values_2[ 17 ][ $no ] = $item[ 35 ];
$values_2[ 18 ][ $no ] = $item[ 37 ];
$values_2[ 19 ][ $no ] = $item[ 39 ];
$values_2[ 20 ][ $no ] = $item[ 41 ];
$values_2[ 21 ][ $no ] = $item[ 43 ];
$values_2[ 22 ][ $no ] = $item[ 45 ];
$values_2[ 23 ][ $no ] = $item[ 47 ];
#for ( $item_no = 0; $item_no < $no_of_items; $item_no++ )
# {
# $values_2[ $item_no ][ $no ] = $item[ $item_no ];
# }
$no++;
print "No = $no\n";
}
else
{
$reading_Wells = "FALSE";
}
}
}
close FILE2;
# sorting of values
$no_of_wells_2 = $no;
#for ( $n = 0; $n < $no_of_wells_2; $n++ )
# {
# for ( $m = 0; $m < $no_of_wells_2 - 1; $m++ )
# {
# if (defined $values_2[ 0 ][ $m ] && defined $values_2[ 0 ][ $m + 1 ] )
# {
# if ( $values_1[ 0 ][ $m ] > $values_1[ 0 ][ $m + 1 ] )
# {
# &swap_values_2( $m );
# }
# }
# }
# }
for ( $n = 0; $n < $no_of_wells_2 - 1; $n++ )
{
for ( $m = 0; $m < $no_of_values; $m++ )
{
print $values_2[ $m ][ $n ] . ";";
}
print "\n";
}
print OUTFILE "\n\nSummary:\n\n";
# Reformat headline to new format:
$headline =~ s/\"//g;
$headline =~ s/Row\;Column/WellIndex/;
$headline =~ s/; ;/;/g;
print OUTFILE "$headline\n";
print "No of wells = $no_of_wells_1\n";
# collect well names from file1 and file2, unite them to a list
# and sort this list.
$well_list = "";
for( $well_no = 0; $well_no < $no_of_wells_1; $well_no++ )
{
$current_well_name = $values_1[ 0 ][ $well_no ];
if ( $well_list !~ /$current_well_name/ )
{
$well_list = $well_list . ";" . $current_well_name;
}
}
for( $well_no = 0; $well_no < $no_of_wells_2; $well_no++ )
{
$current_well_name = $values_2[ 0 ][ $well_no ];
if ( $well_list !~ /$current_well_name/ )
{
if ( $well_list eq "" )
{
$well_list = $current_well_name;
}
else
{
$well_list = $well_list . ";" . $current_well_name;
}
}
}
@collected_wells = split( /;/, $well_list );
@collected_wells = sort @collected_wells;
```

```
# remove empty well names
# add missing wells
@colrow = ();
$maxcols = 100;
$maxrows = 100;
for ($n = 1; $n < $maxcols; $n++ )
{
for ($m = 1; $m < $maxrows; $m++ )
{
$colrow[ $n ][ $m ] = 0;
}
}
foreach $this_well ( @collected_wells )
{
( $col, $row ) = &get_col_row( $this_well );
$colrow[ $col ][ $row ] = 1;
print "col=$col, row=$row, $this_well\n";
}
$colmax = 0;
$colmin = 100;
@rowmax = ();
$smallest_rowmin = 100;
@rowmin = ();
for ($this_col = 1; $this_col < $maxcols; $this_col++ )
{
$rowmax[ $this_col ] = 0;
$rowmin[ $this_col ] = 100;
for ($this_row = 1; $this_row < $maxrows; $this_row++ )
{
if ( $colrow[ $this_col ][ $this_row ] == 1 )
{
if ( $this_col < $colmin )
{
$colmin = $this_col;
}
if ( $this_col > $colmax )
{
$colmax = $this_col;
}
if ( $this_row < $rowmin[ $this_col ] )
{
$rowmin[ $this_col ] = $this_row;
}
if ( $this_row > $rowmax[ $this_col ] )
{
$rowmax[ $this_col ] = $this_row;
}
}
if ( $smallest_rowmin > $rowmin[ $this_col ] )
{
$smallest_rowmin = $rowmin[ $this_col ];
```

```
}
}
if ( $rowmax[ $this_col ] > 0 || $rowmin[ $this_col ] < 100 )
{
print "rowmax[ $this_col ] = " . $rowmax[ $this_col ] . ", rowmin[ $this_col ] = " . $rowmin[ $this_col ] . "\n";
}
}
# put the $rowmin[ $this_col ] to $smallest_rowmin, all rows the same.
for ($this_col = 1; $this_col < $maxcols; $this_col++ )
{
$rowmin[ $this_col ] = $smallest_rowmin;
}
print "colmax = $colmax, colmin = $colmin\n";
$filled_up_wells = "";
for ( $n = $colmin; $n <= $colmax; $n++ )
{
for ( $m = $rowmin[ $n ]; $m <= $rowmax[ $n ]; $m++ )
{
if ( $filled_up_wells eq "" )
{
$filled_up_wells = &get_well( $n, $m );
}
else
{
$filled_up_wells = $filled_up_wells . ";" . &get_well( $n, $m );
}
$testwell = &get_well( $n, $m );
print "$testwell from $n, $m\n";
}
}
@collected_wells = split( /;/, $filled_up_wells );
$no_of_collected_wells = @collected_wells;
print "no_of_collected_wells = $no_of_collected_wells\n";
for( $well_no = 0; $well_no < $no_of_collected_wells; $well_no++ )
{
# find indices of $well, if it exists...
print "Working on " . $collected_wells[ $well_no ] . "\n";
$position1 = -1;
for( $n = 0; $n < $no_of_wells_1; $n++ )
{
if ( $values_1[ 0 ][ $n ] eq $collected_wells[ $well_no ] )
{
$position1 = $n;
}
}
$position2 = -1;
for( $n = 0; $n < $no_of_wells_2; $n++ )
{
if ( $values_2[ 0 ][ $n ] eq $collected_wells[ $well_no ] )
{
$position2 = $n;
```

```
}
}
print "position1 = $position1, position2 = $position2\n";
# first index = horizontal = value, second index = vertical = well
#print OUTFILE $values_1[ 0 ][ $well_no ] . ";";
# all values here should be defined!
$values_3[ 0 ][ $well_no ] = $collected_wells[ $well_no ];
$values_3[ 1 ][ $well_no ] = &add_values( $values_1[ 1 ][ $position1 ], $values_2[ 1 ][ $position2 ], $position1, $position2 );
$values_3[ 2 ][ $well_no ] = &add_values( $values_1[ 2 ][ $position1 ], $values_2[ 2 ][ $position2 ], $position1, $position2 );
$values_3[ 3 ][ $well_no ] = &add_values( $values_1[ 3 ][ $position1 ], $values_2[ 3 ][ $position2 ], $position1, $position2 );
$values_3[ 4 ][ $well_no ] = &add_values( $values_1[ 4 ][ $position1 ], $values_2[ 4 ][ $position2 ], $position1, $position2 );
$values_3[ 5 ][ $well_no ] = &add_values( $values_1[ 5 ][ $position1 ], $values_2[ 5 ][ $position2 ], $position1, $position2 );
$values_3[ 6 ][ $well_no ] = &add_mean_values( $values_1[ 6 ][ $position1 ], $values_2[ 6 ][ $position2 ] , $values_1[ 20 ][ $position1 ], $values_2[ 20 ][ $position2 ], $position1, $position2 );
$values_3[ 7 ][ $well_no ] = &add_mean_values( $values_1[ 7 ][ $position1 ], $values_2[ 7 ][ $position2 ] , $values_1[ 2 ][ $position1 ], $values_2[ 2 ][ $position2 ], $position1, $position2 );
$values_3[ 8 ][ $well_no ] = &add_mean_values( $values_1[ 8 ][ $position1 ], $values_2[ 8 ][ $position2 ] , $values_1[ 20 ][ $position1 ], $values_2[ 20 ][ $position2 ], $position1, $position2 );
$values_3[ 9 ][ $well_no ] = &add_mean_values( $values_1[ 9 ][ $position1 ], $values_2[ 9 ][ $position2 ] , $values_1[ 20 ][ $position1 ], $values_2[ 20 ][ $position2 ], $position1, $position2 );
$values_3[ 10 ][ $well_no ] = &add_mean_values( $values_1[ 10 ][ $position1 ], $values_2[ 10 ][ $position2 ] , $values_1[ 1 ][ $position1 ], $values_2[ 1 ][ $position2 ], $position1, $position2 );
$values_3[ 11 ][ $well_no ] = &add_mean_values( $values_1[ 11 ][ $position1 ], $values_2[ 11 ][ $position2 ] , $values_1[ 1 ][ $position1 ], $values_2[ 1 ][ $position2 ], $position1, $position2 );
$values_3[ 12 ][ $well_no ] = &add_values( $values_1[ 12 ][ $position1 ], $values_2[ 12 ][ $position2 ], $position1, $position2 );
$values_3[ 13 ][ $well_no ] = &add_values( $values_1[ 13 ][ $position1 ], $values_2[ 13 ][ $position2 ], $position1, $position2 );
$values_3[ 14 ][ $well_no ] = &add_mean_values( $values_1[ 14 ][ $position1 ], $values_2[ 14 ][ $position2 ] , $values_1[ 1 ][ $position1 ], $values_2[ 1 ][ $position2 ], $position1, $position2 );
$values_3[ 15 ][ $well_no ] = &add_mean_values( $values_1[ 15 ][ $position1 ], $values_2[ 15 ][ $position2 ] , $values_1[ 1 ][ $position1 ], $values_2[ 1 ][ $position2 ], $position1, $position2 );
$values_3[ 16 ][ $well_no ] = &add_mean_values( $values_1[ 16 ][ $position1 ], $values_2[ 16 ][ $position2 ] , $values_1[ 1 ][ $position1 ], $values_2[ 1 ][ $position2 ], $position1, $position2 );
$values_3[ 17 ][ $well_no ] = &add_mean_values( $values_1[ 17 ][ $position1 ], $values_2[ 17 ][ $position2 ] , $values_1[ 1 ][ $position1 ], $values_2[ 1 ][ $position2 ], $position1, $position2 );
$values_3[ 18 ][ $well_no ] = &add_mean_values( $values_1[ 18 ][ $position1 ], $values_2[ 18 ][ $position2 ] , $values_1[ 1 ][ $position1 ], $values_2[ 1 ][ $position2 ], $position1, $position2 );
$values_3[ 19 ][ $well_no ] = &add_mean_values( $values_1[ 19 ][ $position1 ], $values_2[ 19 ][ $position2 ] , $values_1[ 1 ][ $position1 ], $values_2[ 1 ][ $position2 ], $position1, $position2 );
$values_3[ 20 ][ $well_no ] = &add_values( $values_1[ 20 ][ $position1 ], $values_2[ 20 ][ $position2 ], $position1, $position2 );
$values_3[ 21 ][ $well_no ] = &add_values( $values_1[ 21 ][ $position1 ], $values_2[ 21 ][ $position2 ], $position1, $position2 );
$values_3[ 22 ][ $well_no ] = &add_values( $values_1[ 22 ][ $position1 ], $values_2[ 22 ][ $position2 ], $position1, $position2 );
#$values_3[ 23 ][ $well_no ] = &add_mean_values( $values_1[ 23 ][ $position1 ], $values_2[ 23 ][ $position2 ], $position1, $position2 );
$values_3[ 23 ][ $well_no ] = &add_mean_values( $values_1[ 23 ][ $position1 ], $values_2[ 23 ][ $position2 ], $values_1[ 1 ][ $position1 ], $values_2[ 1 ][ $position2 ], $position1, $position2 ); # korrigiert kst 2009-
12-21
}
for( $well_no = 0; $well_no < $no_of_collected_wells; $well_no++ )
{
# find indices of $well
print "Working on " . $collected_wells[ $well_no ] . "\n";
# first index = horizontal = value, second index = vertical = well
if ( $values_3[ 0 ][ $well_no ] ne "" ) # why???
{
for ( $m = 0; $m < $no_of_values; $m++ )
{
print OUTFILE $values_3[ $m ][ $well_no ] . ";";
}
print OUTFILE "\n";
}
( $this_col, $this_row ) = &get_col_row( $collected_wells[ $well_no ] );
$is_second_well = "FALSE";
$is_second_well = &get_second_well_info( $this_col, $this_row );
# $position1 = no of previous well in &collected_wells
# $position2 = no of this well in &collected_wells
```

```
if ( $is_second_well eq "TRUE" )
{
$position1 = &get_position_of_well( &get_well( $this_col, $this_row - 1 ) );
$position2 = $well_no;
$values_4[ 0 ][ $well_no ] = $collected_wells[ $position1 ] . "-" . $collected_wells[ $position2 ];
$values_4[ 1 ][ $well_no ] = &add_values( $values_3[ 1 ][ $position1 ], $values_3[ 1 ][ $position2 ], $position1, $position2 );
$values_4[ 2 ][ $well_no ] = &add_values( $values_3[ 2 ][ $position1 ], $values_3[ 2 ][ $position2 ], $position1, $position2 );
$values_4[ 3 ][ $well_no ] = &add_values( $values_3[ 3 ][ $position1 ], $values_3[ 3 ][ $position2 ], $position1, $position2 );
$values_4[ 4 ][ $well_no ] = &add_values( $values_3[ 4 ][ $position1 ], $values_3[ 4 ][ $position2 ], $position1, $position2 );
$values_4[ 5 ][ $well_no ] = &add_values( $values_3[ 5 ][ $position1 ], $values_3[ 5 ][ $position2 ], $position1, $position2 );
$values_4[ 6 ][ $well_no ] = &add_mean_values( $values_3[ 6 ][ $position1 ], $values_3[ 6 ][ $position2 ] , $values_3[ 20 ][ $position1 ], $values_3[ 20 ][ $position2 ], $position1, $position2 );
$values_4[ 7 ][ $well_no ] = &add_mean_values( $values_3[ 7 ][ $position1 ], $values_3[ 7 ][ $position2 ] , $values_3[ 2 ][ $position1 ], $values_3[ 2 ][ $position2 ], $position1, $position2 );
$values_4[ 8 ][ $well_no ] = &add_mean_values( $values_3[ 8 ][ $position1 ], $values_3[ 8 ][ $position2 ] , $values_3[ 20 ][ $position1 ], $values_3[ 20 ][ $position2 ], $position1, $position2 );
$values_4[ 9 ][ $well_no ] = &add_mean_values( $values_3[ 9 ][ $position1 ], $values_3[ 9 ][ $position2 ] , $values_3[ 20 ][ $position1 ], $values_3[ 20 ][ $position2 ], $position1, $position2 );
$values_4[ 10 ][ $well_no ] = &add_mean_values( $values_3[ 10 ][ $position1 ], $values_3[ 10 ][ $position2 ] , $values_3[ 1 ][ $position1 ], $values_3[ 1 ][ $position2 ], $position1, $position2 );
$values_4[ 11 ][ $well_no ] = &add_mean_values( $values_3[ 11 ][ $position1 ], $values_3[ 11 ][ $position2 ] , $values_3[ 1 ][ $position1 ], $values_3[ 1 ][ $position2 ], $position1, $position2 );
$values_4[ 12 ][ $well_no ] = &add_values( $values_3[ 12 ][ $position1 ], $values_3[ 12 ][ $position2 ], $position1, $position2 );
$values_4[ 13 ][ $well_no ] = &add_values( $values_3[ 13 ][ $position1 ], $values_3[ 13 ][ $position2 ], $position1, $position2 );
$values_4[ 14 ][ $well_no ] = &add_mean_values( $values_3[ 14 ][ $position1 ], $values_3[ 14 ][ $position2 ] , $values_3[ 1 ][ $position1 ], $values_3[ 1 ][ $position2 ], $position1, $position2 );
$values_4[ 15 ][ $well_no ] = &add_mean_values( $values_3[ 15 ][ $position1 ], $values_3[ 15 ][ $position2 ] , $values_3[ 1 ][ $position1 ], $values_3[ 1 ][ $position2 ], $position1, $position2 );
$values_4[ 16 ][ $well_no ] = &add_mean_values( $values_3[ 16 ][ $position1 ], $values_3[ 16 ][ $position2 ] , $values_3[ 1 ][ $position1 ], $values_3[ 1 ][ $position2 ], $position1, $position2 );
$values_4[ 17 ][ $well_no ] = &add_mean_values( $values_3[ 17 ][ $position1 ], $values_3[ 17 ][ $position2 ] , $values_3[ 1 ][ $position1 ], $values_3[ 1 ][ $position2 ], $position1, $position2 );
$values_4[ 18 ][ $well_no ] = &add_mean_values( $values_3[ 18 ][ $position1 ], $values_3[ 18 ][ $position2 ] , $values_3[ 1 ][ $position1 ], $values_3[ 1 ][ $position2 ], $position1, $position2 );
$values_4[ 19 ][ $well_no ] = &add_mean_values( $values_3[ 19 ][ $position1 ], $values_3[ 19 ][ $position2 ] , $values_3[ 1 ][ $position1 ], $values_3[ 1 ][ $position2 ], $position1, $position2 );
$values_4[ 20 ][ $well_no ] = &add_values( $values_3[ 20 ][ $position1 ], $values_3[ 20 ][ $position2 ], $position1, $position2 );
$values_4[ 21 ][ $well_no ] = &add_values( $values_3[ 21 ][ $position1 ], $values_3[ 21 ][ $position2 ], $position1, $position2 );
$values_4[ 22 ][ $well_no ] = &add_values( $values_3[ 22 ][ $position1 ], $values_3[ 22 ][ $position2 ], $position1, $position2 );
#$values_4[ 23 ][ $well_no ] = &add_mean_values( $values_3[ 23 ][ $position1 ], $values_3[ 23 ][ $position2 ], $position1, $position2 );
$values_4[ 23 ][ $well_no ] = &add_mean_values( $values_3[ 23 ][ $position1 ], $values_3[ 23 ][ $position2 ], $values_3[ 1 ][ $position1 ], $values_3[ 1 ][ $position2 ], $position1, $position2 ); #corrigiert kst 2009-12-
21
for ( $m = 0; $m < $no_of_values; $m++ )
{
print OUTFILE $values_4[ $m ][ $well_no ] . ";";
}
print OUTFILE "\n";
}
}
close OUTFILE;
# do graphics
&plot_parameter( 1 );
&plot_parameter( 2 );
&plot_parameter( 3 );
&plot_parameter( 4 );
&plot_parameter( 5 );
&plot_parameter( 6 );
&plot_parameter( 7 );
&plot_parameter( 8 );
&plot_parameter( 9 );
&plot_parameter( 10 );
&plot_parameter( 11 );
&plot_parameter( 12 );
```

```
&plot_parameter( 13 );
&plot_parameter( 14 );
&plot_parameter( 15 );
&plot_parameter( 16 );
&plot_parameter( 17 );
&plot_parameter( 18 );
&plot_parameter( 19 );
&plot_parameter( 20 );
&plot_parameter( 21 );
&plot_parameter( 22 );
&plot_parameter( 23 );
print "Done.\n";
# end of script:PERL script for Plasma membrane microdomain script
```