6. Vorlesung Mikroskopische Bildverarbeitung

Wahlpflichtmodul 9521: EI-M im 1. und 3. Fachsemester

Initialisierung

15. Januar 2016

Wiederholung aus der 5. Vorlesung

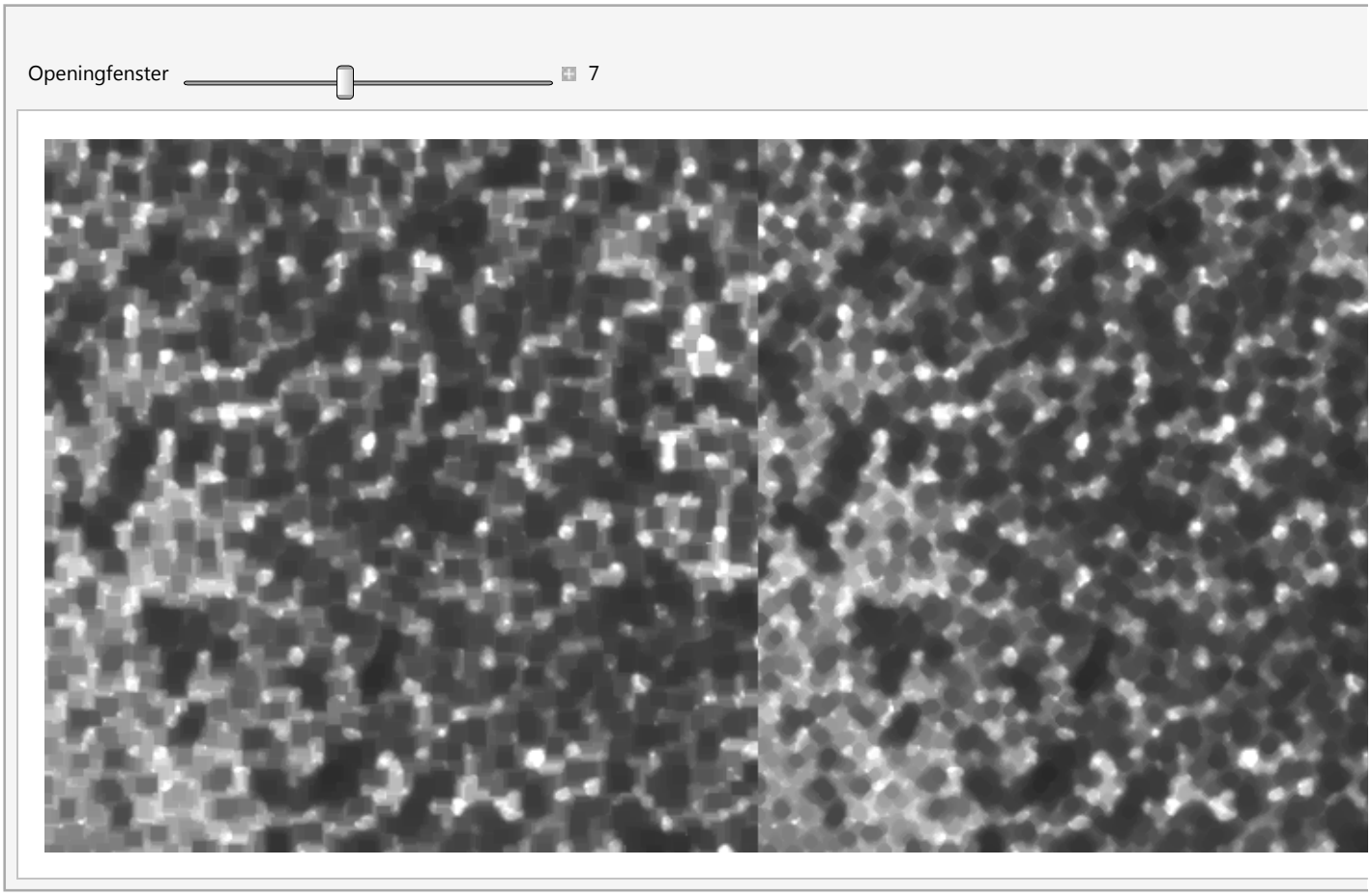
Aus 14. Nichtlineare Filter (Fortsetzung)

Gray-Scale-Closing: Minimumfilterung gefolgt auf Maximumfilterung

```
(*
 Clear[GrayClosing];
  GrayClosing[im_,el_]:=GrayErode[GrayDilate[im,el],el];
  *)
```

Gray-Scale-Closing mit boxförmigem und kreisförmigem Strukturelement

```
Manipulate[ControlActive[größe, Show[ImageAssemble[{Closing[#, boxmaske[(größe - 1) / 2]],
      Closing[#, kreismaske[(größe - 1) / 2]]}], ImageSize → {2, 1} * ImageDimensions[#]]],
  {{größe, 5, "Openingfenster"}, 1, 15, 2, Appearance → "Labeled"},
  ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[First@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```



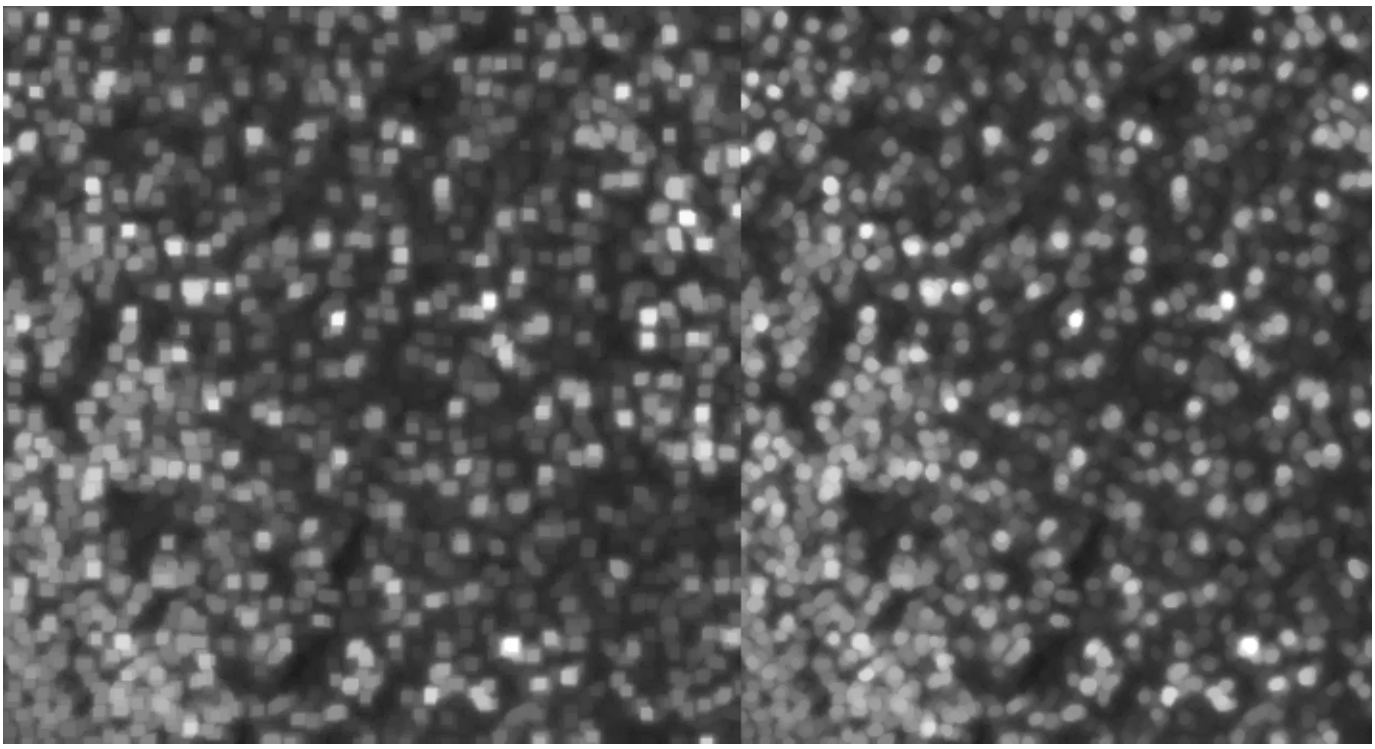
Gray-Scale-Opening: Maximumfilterung gefolgt auf Minimumfilterung

```
(*
 Clear[GrayOpening];
  GrayOpening[im_,el_]:=GrayDilate[GrayErode[im,el],el];
  *)
```

Gray-Scale-Opening mit boxförmigem und kreisförmigem Strukturelement

```
Manipulate[ControlActive[größe, Show[ImageAssemble[{Opening[#, boxmaske[(größe - 1) / 2]],
        Opening[#, kreismaske[(größe - 1) / 2]]}], ImageSize → {2, 1} * ImageDimensions[#]]],
   {{größe, 5, "Openingfenster"}, 1, 15, 2, Appearance → "Labeled"},
   ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[First@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```

Openingfenster 5



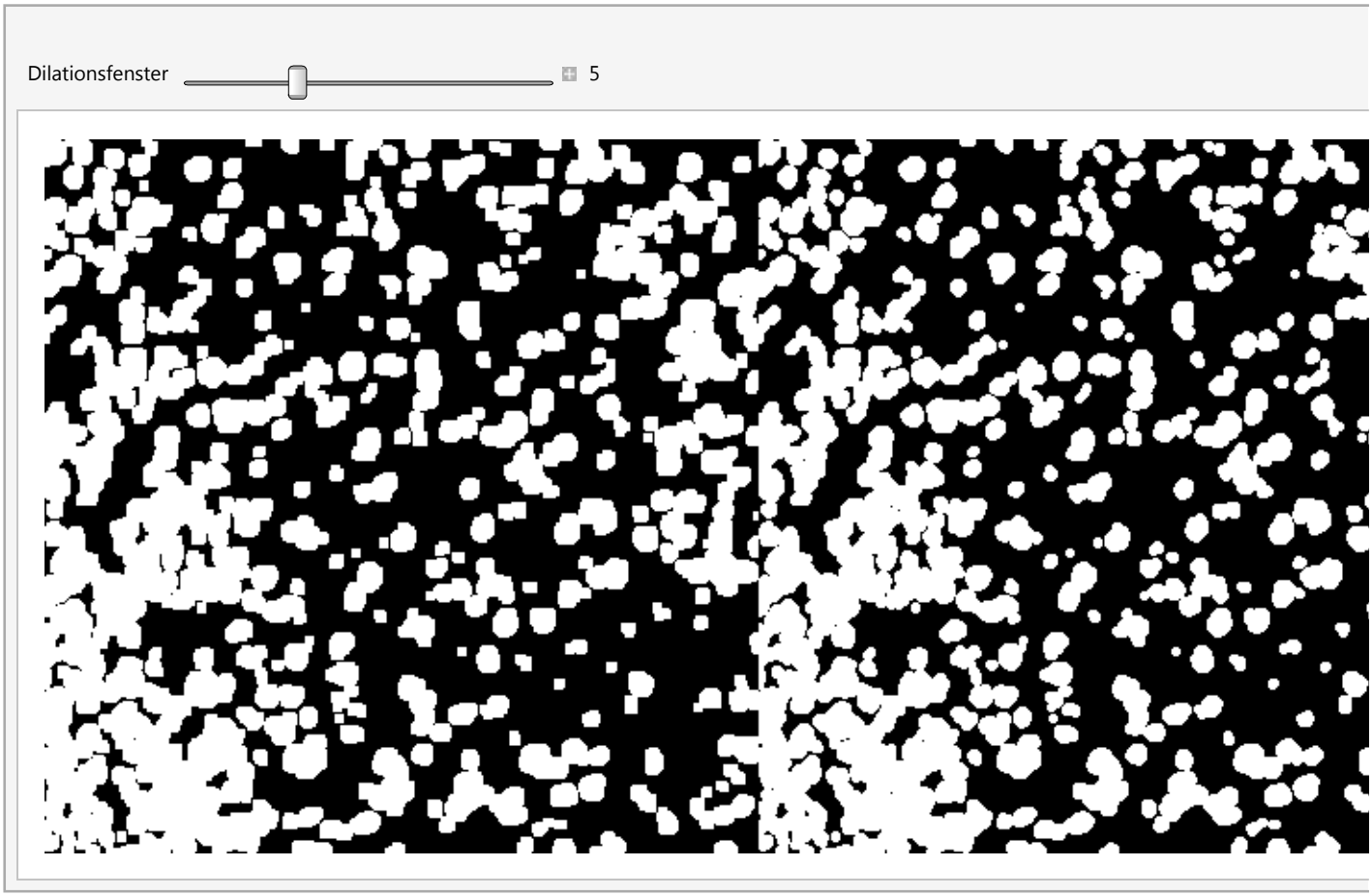
Dilationsfilter, eine vereinfachte Anwendung der Maximumfilter auf Binärbilder

- die Umgebung jedes Binärpixels wird entsprechend des verwendeten Strukturelements erweitert

```
(*
Clear[BinaryDilate];
BinaryDilate=Compile[{{im,_Integer,2},{el,_Integer,2}},
  ListConvolve[el,im,{Ceiling[Dimensions[el]/2]},im,BitAnd,BitOr],
  CompilationTarget→"WVM",RuntimeAttributes→{Listable},Parallelization→True];
*)
```

Binärdilation mit boxförmigem und kreisförmigem Strukturelement

```
Manipulate[ControlActive[größe, Show[ImageAssemble[{Dilation[#, boxmaske[(größe - 1) / 2]],
        Dilation[#, kreismaske[(größe - 1) / 2]]}], ImageSize → {2, 1} * ImageDimensions[#]]],
   {{größe, 5, "Dilationsfenster"}, 1, 15, 2, Appearance → "Labeled"},
   ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[Binarize[First@ColorSeparate@Ton2CD21dreikanalausgleichF1, 1 / 2], {384, 384}]]
```



Erosionsfilter, eine vereinfachte Anwendung der Minimumfilter auf Binärbilder

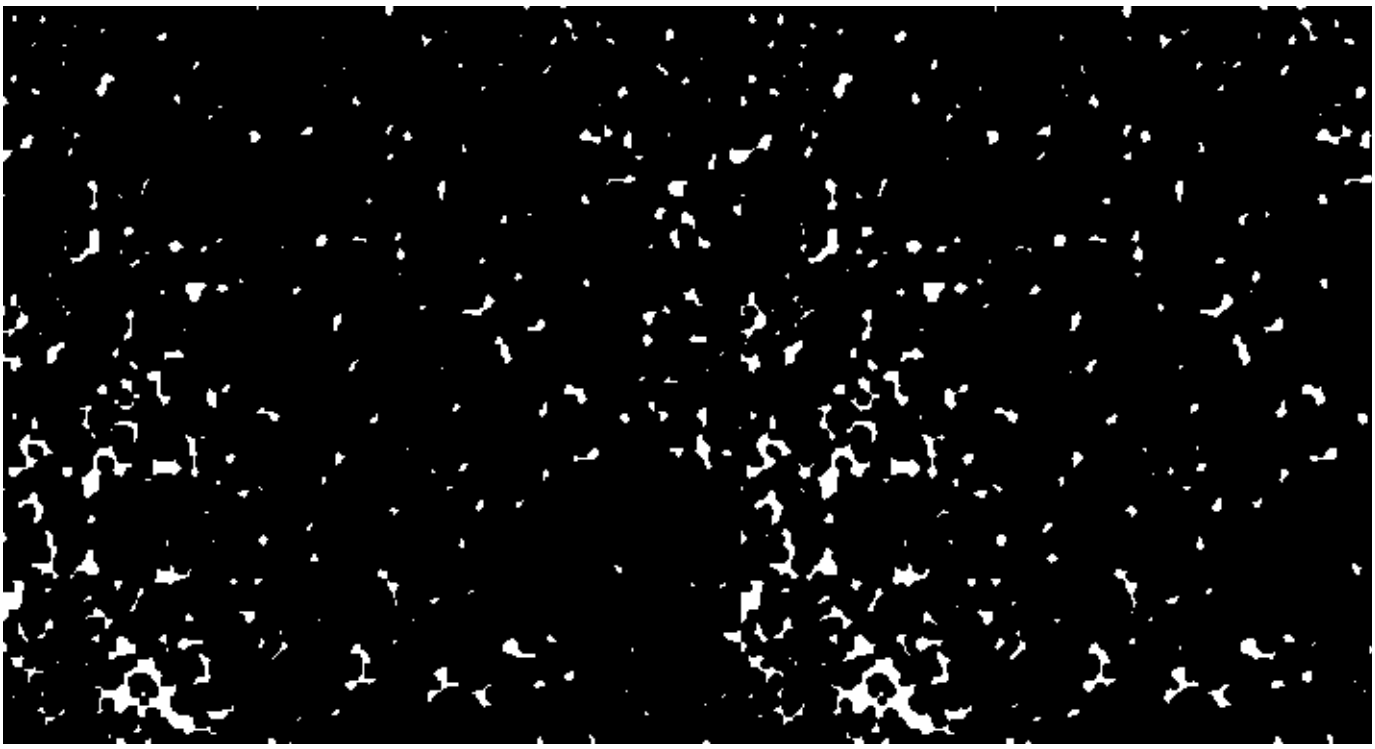
- Binärpixel werden nur belassen, wenn die Umgebung entsprechend des definierten Strukturelements ebenfalls belegt ist

```
(*
Clear[BinaryErode];
BinaryErode=Compile[{{im,_Integer,2},{el,_Integer,2}},
  ListConvolve[1-el,im,{Ceiling[Dimensions[el]/2]},im,BitOr,BitAnd],
  CompilationTarget→"WVM",RuntimeAttributes→{Listable},Parallelization→True];
*)
```

Binärerosion mit boxförmigem und kreisförmigem Strukturelement

```
Manipulate[ControlActive[größe, Show[ImageAssemble[{Erosion[#, boxmaske[(größe - 1) / 2]],
      Erosion[#, kreismaske[(größe - 1) / 2]]}], ImageSize → {2, 1} * ImageDimensions[#]]],
  {{größe, 5, "Erosionsfenster"}, 1, 15, 2, Appearance → "Labeled"},
  ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[Binarize[First@ColorSeparate@Ton2CD21dreikanalausgleichF1, 1 / 2], {384, 384}]]
```

Erosionsfenster 5

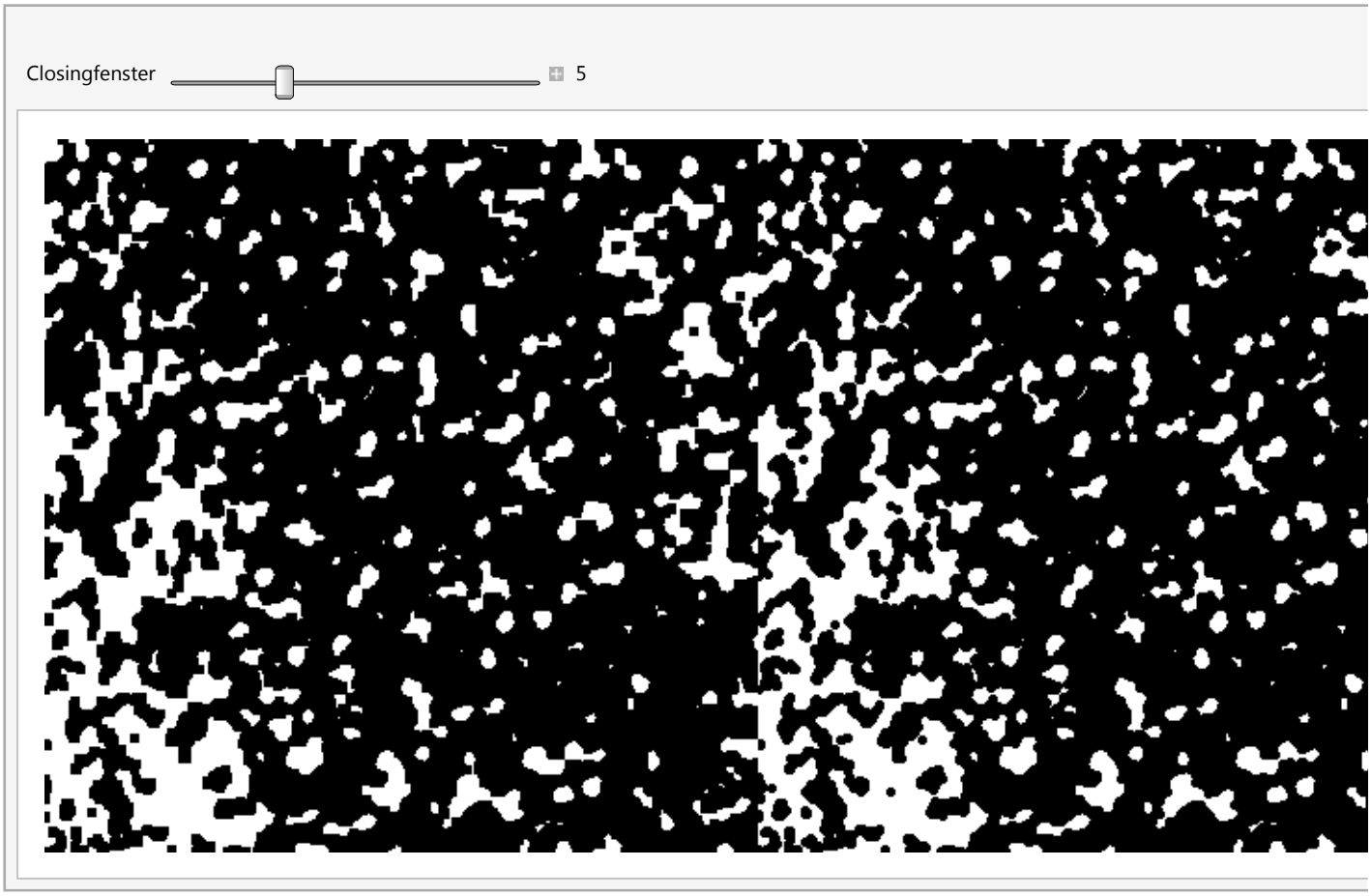


Binär-Closing: Erosion gefolgt auf Dilation

```
(*
Clear[BinaryClosing];
BinaryClosing[im_,el_]:=BinaryErode[BinaryDilate[im,el],el]
*)
```

Binär-Closing mit boxförmigem und kreisförmigem Strukturelement

```
Manipulate[ControlActive[größe, Show[ImageAssemble[{Closing[#, boxmaske[(größe - 1) / 2]],
      Closing[#, kreismaske[(größe - 1) / 2]]}], ImageSize → {2, 1} * ImageDimensions[#]]],
  {{größe, 5, "Closingfenster"}, 1, 15, 2, Appearance → "Labeled"},
  ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[Binarize[First@ColorSeparate@Ton2CD21dreikanalausgleichF1, 1 / 2], {384, 384}]]
```



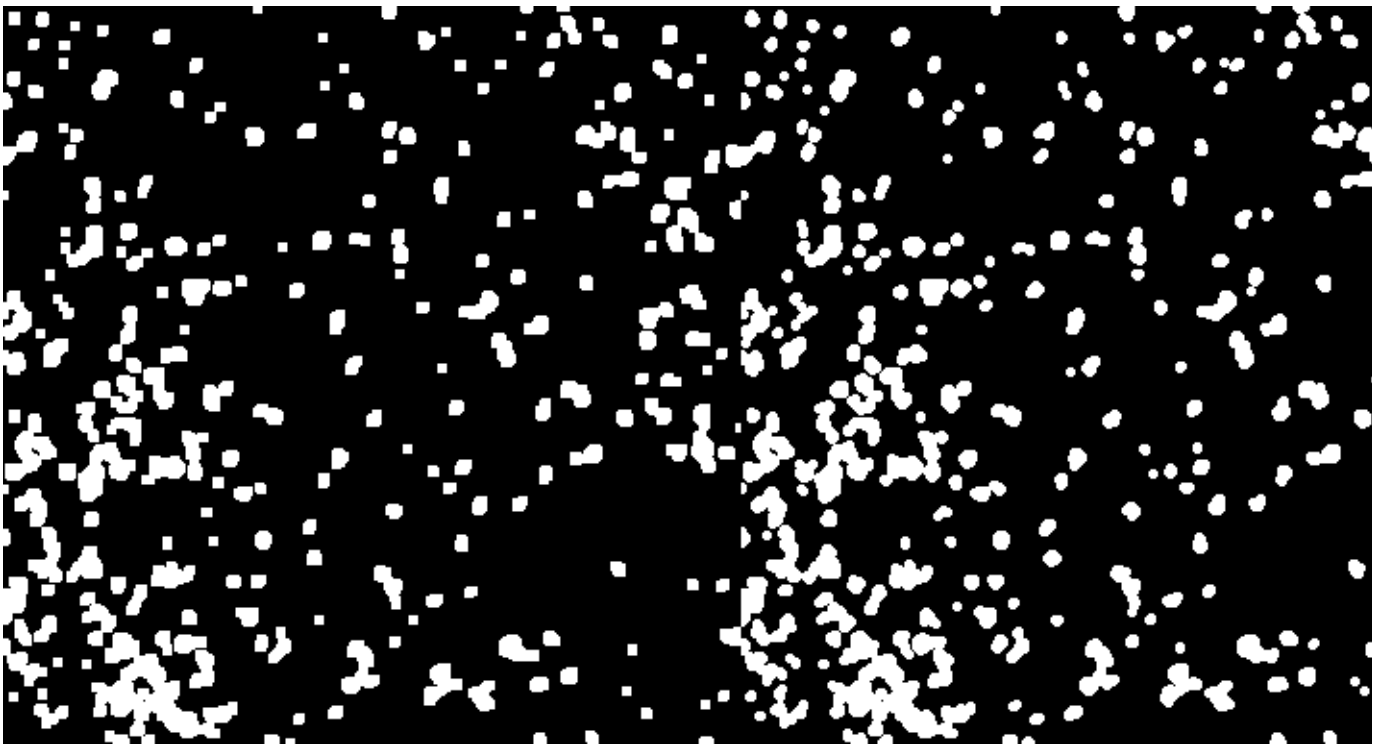
Binär-Opening: Dilation gefolgt auf Erosion

```
(*
Clear[BinaryOpening];
BinaryOpening[im_,el_]:=BinaryDilate[BinaryErode[im,el],el]
*)
```

Binär-Opening mit boxförmigem und kreisförmigem Strukturelement

```
Manipulate[ControlActive[größe, Show[ImageAssemble[{Opening[#, boxmaske[(größe - 1) / 2]],
      Opening[#, kreismaske[(größe - 1) / 2]]}], ImageSize → {2, 1} * ImageDimensions[#]]],
  {{größe, 5, "Openingfenster"}, 1, 15, 2, Appearance → "Labeled"},
  ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[Binarize[First@ColorSeparate@Ton2CD21dreikanalausgleichF1, 1 / 2], {384, 384}]]
```

Openingfenster 5



Top-Hat-Transformation als Differenz von Bild und Opening-Bild

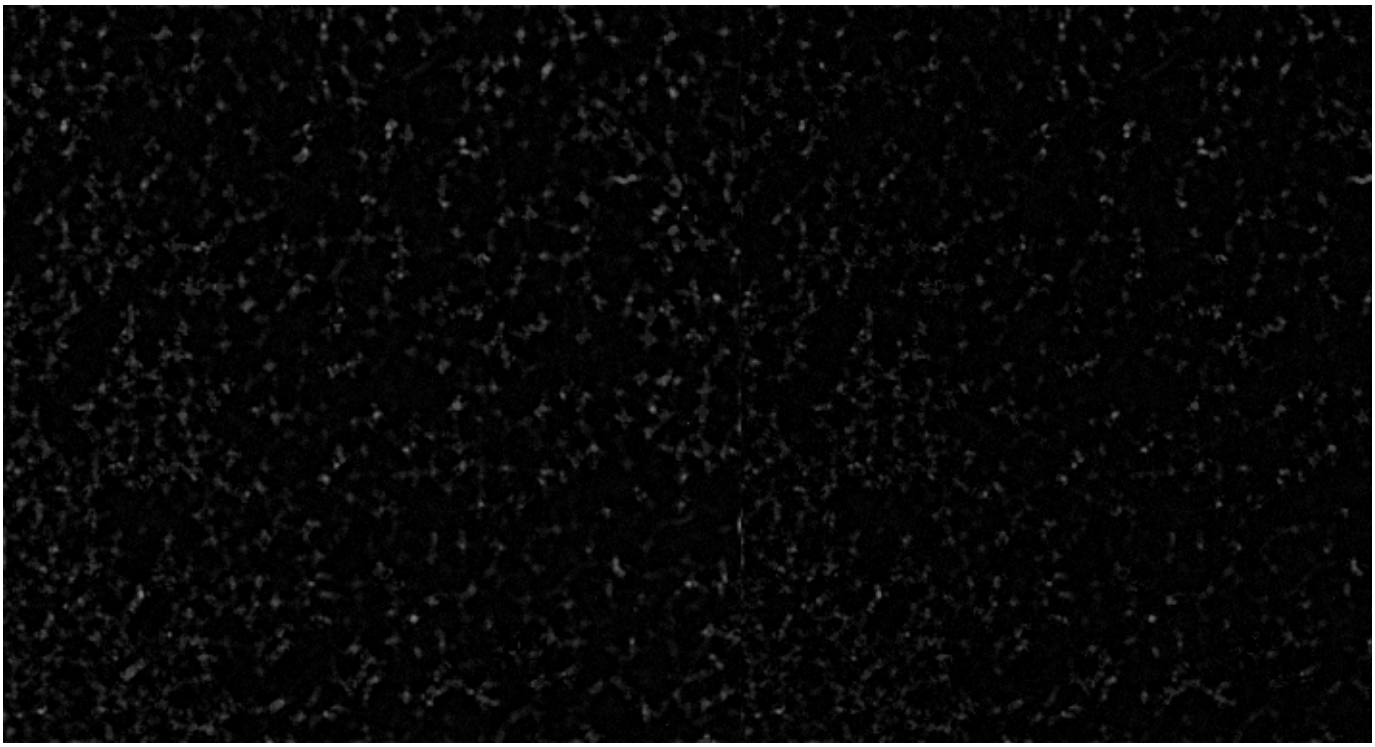
Betonung kleiner heller Bilddetails

```
(*
 Clear[MyTopHatTransform];
  MyTopHatTransform[im_,el_]:=im-GrayOpening[im,el];
  *)
```

Top-Hat-Transformation mit boxförmigem und kreisförmigem Strukturelement

```
Manipulate[ControlActive[größe,
    Show[ImageAssemble[{TopHatTransform[#, boxmaske[(größe - 1) / 2]], TopHatTransform[
        #, kreismaske[(größe - 1) / 2]]}], ImageSize → {2, 1} * ImageDimensions[#]]],
  {{größe, 11, "Top-Hat-Openingfenster"}, 1, 21, 2, Appearance → "Labeled"},
  ContinuousAction → False, SaveDefinitions → True] &[
ImageCrop[First@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```

Top–Hat–Openingfenster 5



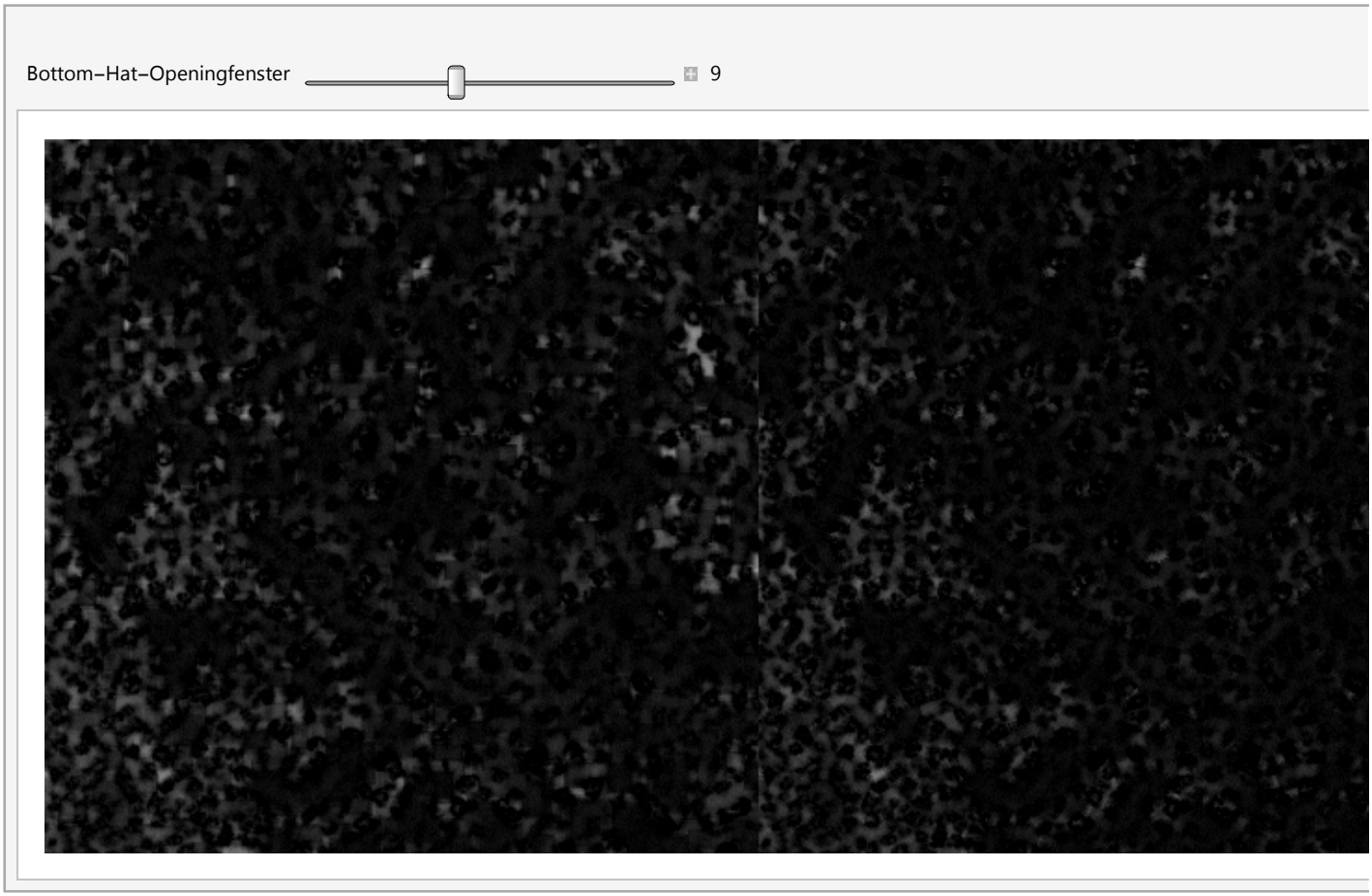
Bottom-Hat-Transformation als Differenz von Closing-Bild und Bild

Schwächung kleiner heller Bilddetails

```
(*
 Clear[MyBottomHatTransform];
  MyBottomHatTransform[im_,el_]:=GrayClosing[im,el]-im;
  *)
```

Bottom-Hat-Transformation mit boxförmigem und kreisförmigem Strukturelement

```
Manipulate[ControlActive[größe,
    Show[ImageAssemble[{BottomHatTransform[#, boxmaske[(größe - 1) / 2]], BottomHatTransform[
        #, kreismaske[(größe - 1) / 2]]}], ImageSize → {2, 1} * ImageDimensions[#]]],
  {{größe, 11, "Bottom-Hat-Openingfenster"}, 1, 21, 2, Appearance → "Labeled"},
  ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[First@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```



Aus 15. Lokale Kontrastanhebung

Lokale Kontrastanhebung in Abhängigkeit von lokaler Standardabweichung

(Narenda und Fitch 1981)

$$f(i, j) = m_x(i, j) + \frac{\kappa \sum_{i=0}^{M-1} \sum_{j-1}^{N-1} x(i,j)}{\sigma_x(i,j)\, MN} [x(i, j) - m_x(i, j)]$$

mit

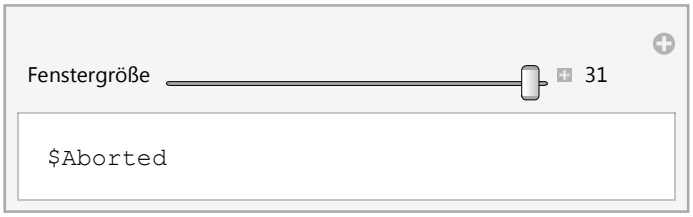
x(i,j): skalarer Bildwert
$\sigma_x(i, j)$: lokale Standardabweichung
$m_x(i, j)$: lokaler Mittelwert
κ: positiver skalarer Skalierungsparameter

f(i,j): kontrastmodfizierter Bildwert
M,N: Bildabmaße

```
Clear[LocalContrastKern];
LocalContrastKern = Compile[{{list, _Real, 1}, {kappatimesgm, _Real}},
    Module[{mean, stddev},
     mean = Mean[list];
     stddev = $MachineEpsilon;
     If[Length[list] ≠ 1, stddev = StandardDeviation[list] + $MachineEpsilon];
     kappatimesgm / stddev * (list[[Ceiling[Length[list] / 2]]] - mean) + mean
    ], CompilationTarget → "WVM", Parallelization → False];
(*Narenda and Fitch, 1981*)
Clear[MyLocalContrastFilter];
Options[MyLocalContrastFilter] =
  {"WindowHalfWidth" -> 7, "Scaling" -> Automatic, "Mask" → "Box"};
MyLocalContrastFilter[im_Image, OptionsPattern[]] :=
  Module[{flatelpos, padim, whw, kappa, kappatimesgm, el, imdata},
   whw = Round[OptionValue["WindowHalfWidth"]];
   kappa = OptionValue["Scaling"];
   If[kappa === Automatic, kappa = 1.0, kappa = N[kappa]];
   If[kappa < 0.0 || kappa > 1.0 || ! NumericQ[kappa], kappa = 1.0];
   el = Switch[OptionValue["Mask"],
     "Box", Table[1, {2 whw + 1}, {2 whw + 1}],
     "Circle", Ceiling[Rescale[
       Sign[Table[(x^2 + y^2), {x, -whw, whw}, {y, -whw, whw}] - whw * (whw + 1 / 2)], {1, -1}]],
     _, Table[1, {2 whw + 1}, {2 whw + 1}]
    ];
   imdata = ImageData[im, "Real"];
   kappatimesgm = kappa * Mean[Flatten[imdata]];
   flatelpos = Flatten[Position[Flatten[el], 1]];
   padim = ArrayPad[imdata, Floor[Dimensions[el] / 2], "Reversed"];
   padim =
    Developer`PartitionMap[(LocalContrastKern[Flatten[#, 1][[flatelpos]], kappatimesgm]) &,
     padim, Dimensions[el], {1, 1}, Ceiling[Dimensions[el] / 2]];
   Image[ArrayPad[padim, -Floor[Dimensions[el] / 2]], "Real"]
  ];
```

Lokale Kontrastanhebung mit kreisförmigem Strukturelement (prolif. Zellen)

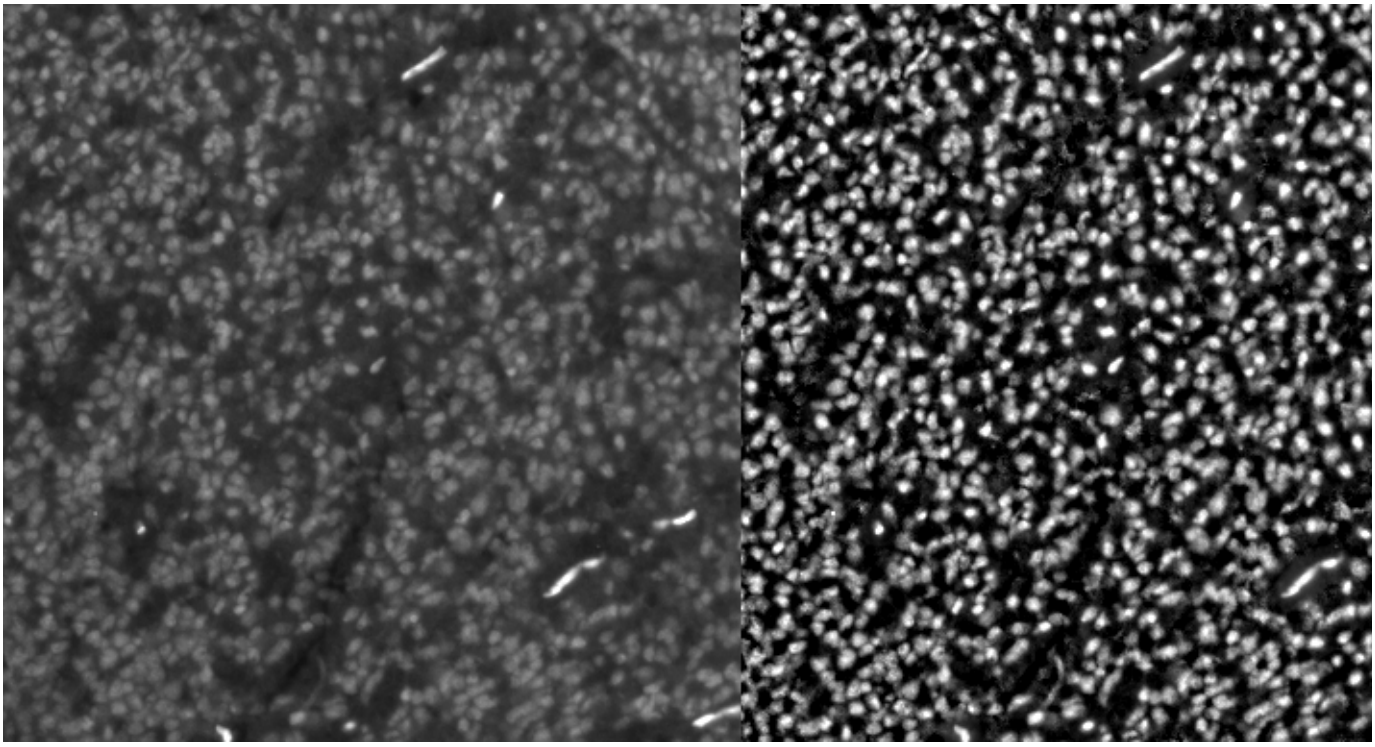
```
Manipulate[ControlActive[größe, Show[ImageAssemble[
      {#, MyLocalContrastFilter[#, "WindowHalfWidth" → (größe - 1) / 2, "Mask" → "Circle"]}],
     ImageSize → {2, 1} * ImageDimensions[#]]], {{größe, 15, "Fenstergröße"}, 1, 31,
    2, Appearance → "Labeled"}, ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[First@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```



Lokale Kontrastanhebung mit kreisförmigem Strukturelement (B-Zellkerne)

```
Manipulate[ControlActive[größe, Show[ImageAssemble[
      {#, MyLocalContrastFilter[#, "WindowHalfWidth" → (größe - 1) / 2, "Mask" → "Circle"]}],
    ImageSize → {2, 1} * ImageDimensions[#]]], {{größe, 15, "Fenstergröße"}, 1, 31,
   2, Appearance → "Labeled"}, ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[First@Rest@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```

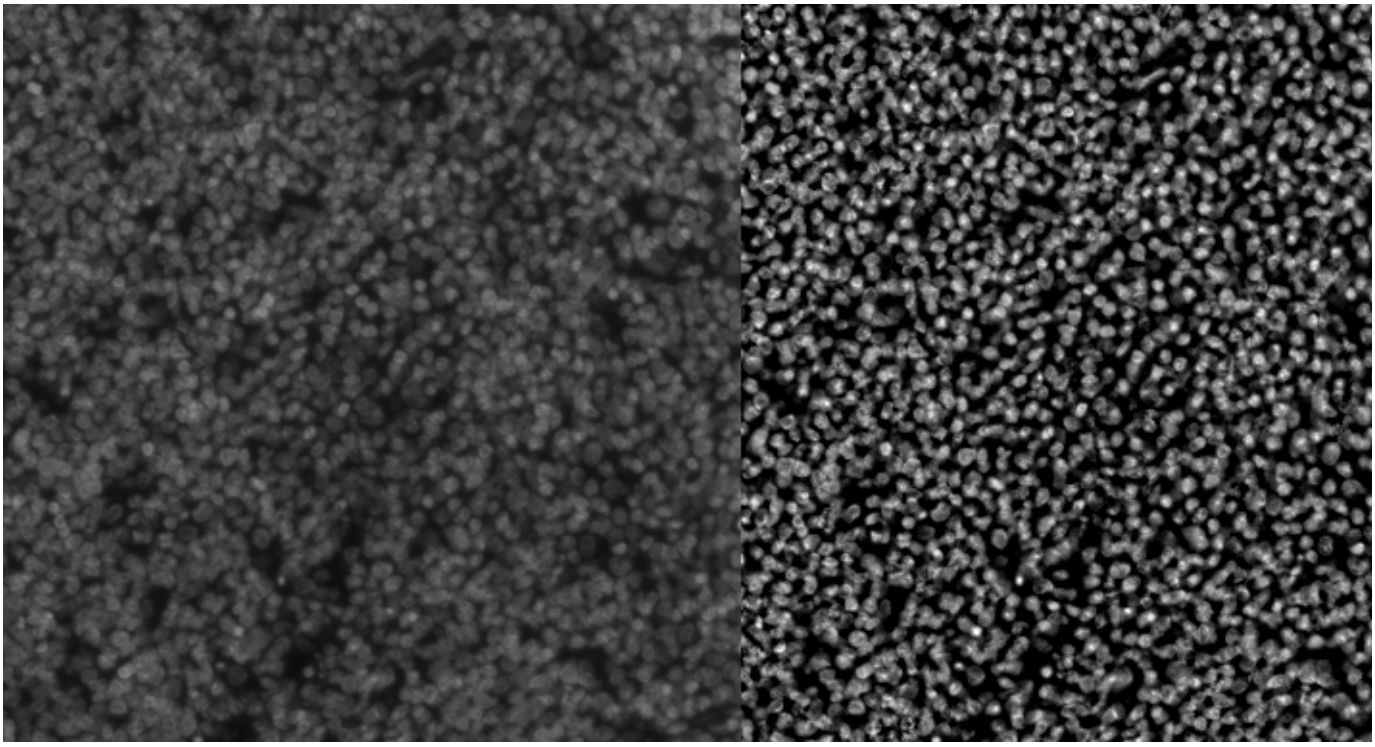
Fenstergröße 19



Lokale Kontrastanhebung mit kreisförmigem Strukturelement (alle Zellkerne)

```
Manipulate[ControlActive[größe, Show[ImageAssemble[
      {#, MyLocalContrastFilter[#, "WindowHalfWidth" → (größe - 1) / 2, "Mask" → "Circle"]}],
    ImageSize → {2, 1} * ImageDimensions[#]]], {{größe, 15, "Fenstergröße"}, 1, 31,
   2, Appearance → "Labeled"}, ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[Last@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```

Fenstergröße 15



Aus 16. Binarisierung durch lokale Schwellwertbestimmung

Lokale Schwellwertbildung in Abhängigkeit von Mittel aus lokalem Maximum und Minimum

Bernsen 1986

$$t(i, j) = \frac{max_x + min_x}{2}$$

außer falls $(max_x - min_x) < c$, dann Binarisierung mit Ausnahmeregel; c (min. Kontrastschwelle) wählbar, bei 8bit typ. c=15, Umgebungsgröße typ. 15x15

Ausnahmeregel (verbal): falls t(i,j) näher an min_x als an max_x, dann Nullsetzen, sonst Einssetzen.

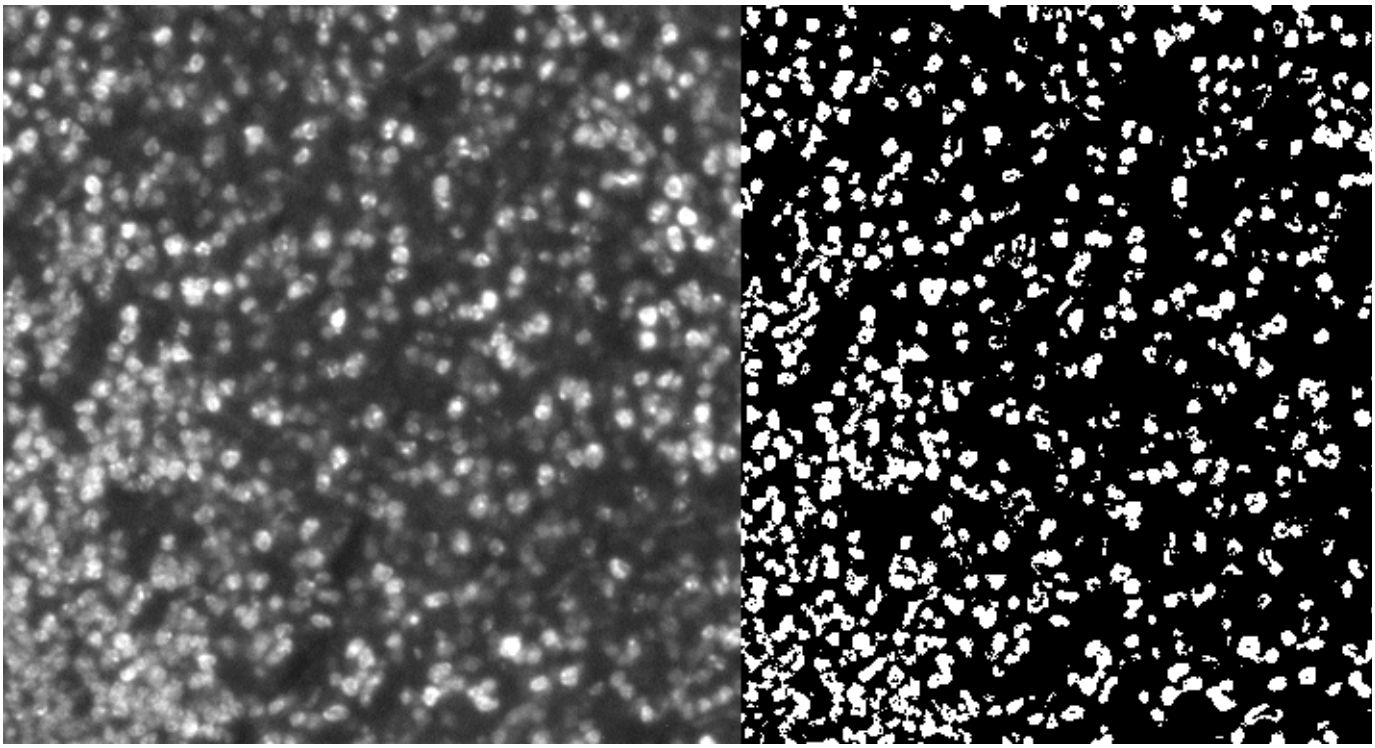
```
Clear[BernsenKern];
BernsenKern = Compile[{{list, _Real, 1}, {cm, _Real}},
   Module[{min, max, diff, sum, thr},
    max = Max[list];
    min = Min[list];
    diff = (max - min);
    sum = max + min;
    thr = 0.5 * sum;
    If[diff < cm,
     If[EuclideanDistance[thr, max] < EuclideanDistance[thr, min], thr = min, thr = max]];
    UnitStep[list[[Ceiling[Length[list] / 2]]] - thr]
   ], CompilationTarget → "WVM", Parallelization → False];
(*Bernsen, 1986*)
Clear[MyBernsenFilter];
Options[MyBernsenFilter] = {"ContrastMeasure" → 15, "WindowHalfWidth" → 15, "Mask" → "Box"};
MyBernsenFilter[im_Image, OptionsPattern[]] := Module[{flatelpos, padim, whw, cm, el},
   whw = Round[OptionValue["WindowHalfWidth"]];
   cm = N[OptionValue["ContrastMeasure"] / 255];
   el = Switch[OptionValue["Mask"],
     "Box", Table[1, {2 whw + 1}, {2 whw + 1}],
     "Circle", Ceiling[Rescale[
       Sign[Table[(x^2 + y^2), {x, -whw, whw}, {y, -whw, whw}] - whw * (whw + 1 / 2)], {1, -1}]],
     _, Table[1, {2 whw + 1}, {2 whw + 1}]
    ];
   flatelpos = Flatten[Position[Flatten[el], 1]];
   padim = ArrayPad[ImageData[im, "Real"], Floor[Dimensions[el] / 2], "Reversed"];
   padim = Developer`PartitionMap[(BernsenKern[Flatten[#, 1][[flatelpos]], cm]) &,
     padim, Dimensions[el], {1, 1}, Ceiling[Dimensions[el] / 2]];
   Image[ArrayPad[padim, -Floor[Dimensions[el] / 2]], "Bit"]
  ];
```

Lokale Schwellwertbildung (Bernsen) mit kreisförmigem Strukturelement (prolif. Zellen)

```
Manipulate[ControlActive[größe, Show[ImageAssemble[
      {#, MyBernsenFilter[#, "ContrastMeasure" → minkontrastschwelle, "WindowHalfWidth" →
          (größe - 1) / 2, "Mask" → "Circle"]}], ImageSize → {2, 1} * ImageDimensions[#]]],
   {{größe, 11, "Bernsen-Fenstergröße"}, 1, 61, 2, Appearance → "Labeled"},
   {{minkontrastschwelle, 50, "min. Kontrastschwelle"}, 0, 255, 1, Appearance → "Labeled"},
   ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[First@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```

Bernsen–Fenstergröße 9

min. Kontrastschwelle 61

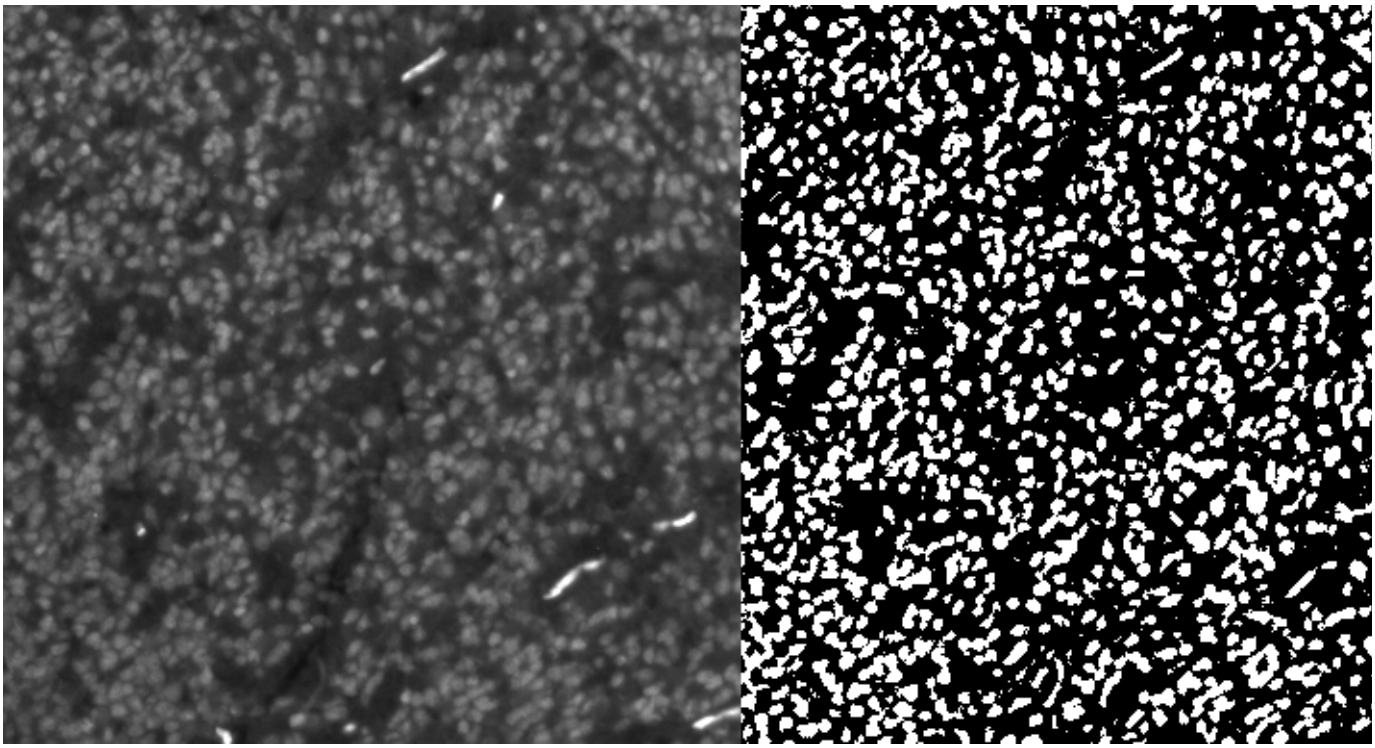


Lokale Schwellwertbildung (Bernsen) mit kreisförmigem Strukturelement (B-Zellkernen)

```
Manipulate[ControlActive[größe, Show[ImageAssemble[
      {#, MyBernsenFilter[#, "ContrastMeasure" → minkontrastschwelle, "WindowHalfWidth" →
          (größe - 1) / 2, "Mask" → "Circle"]}], ImageSize → {2, 1} * ImageDimensions[#]]],
   {{größe, 11, "Bernsen-Fenstergröße"}, 1, 61, 2, Appearance → "Labeled"},
   {{minkontrastschwelle, 25, "min. Kontrastschwelle"}, 0, 255, 1, Appearance → "Labeled"},
   ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[First@Rest@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```

Bernsen–Fenstergröße 11

min. Kontrastschwelle 25

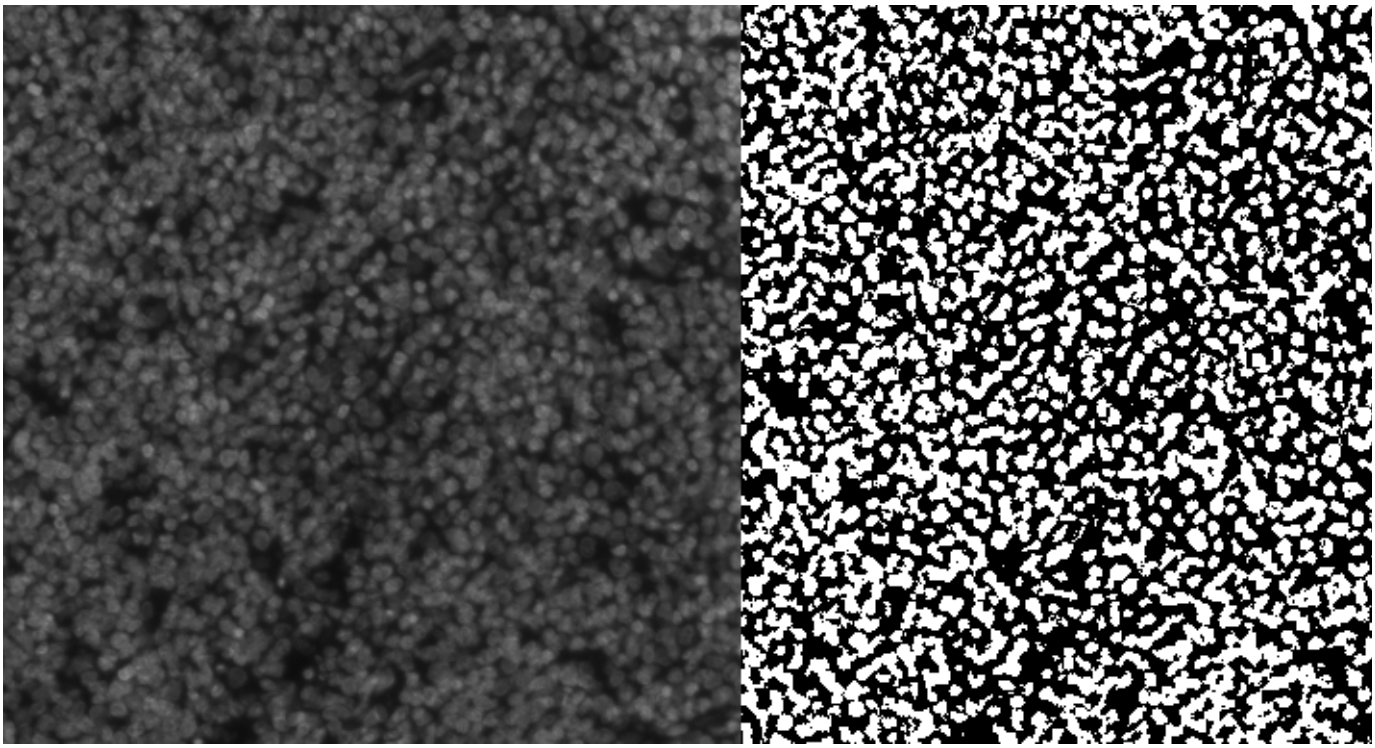


Lokale Schwellwertbildung (Bernsen) mit kreisförmigem Strukturelement (alle Zellkerne)

```
Manipulate[ControlActive[größe, Show[ImageAssemble[
      {#, MyBernsenFilter[#, "ContrastMeasure" → minkontrastschwelle, "WindowHalfWidth" →
          (größe - 1) / 2, "Mask" → "Circle"]}], ImageSize → {2, 1} * ImageDimensions[#]]],
   {{größe, 11, "Bernsen-Fenstergröße"}, 1, 61, 2, Appearance → "Labeled"},
   {{minkontrastschwelle, 25, "min. Kontrastschwelle"}, 0, 255, 1, Appearance → "Labeled"},
   ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[Last@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```

Bernsen–Fenstergröße 11

min. Kontrastschwelle 25



Lokale Schwellwertbildung in Abhängigkeit von lokalem Mittelwert m_x und Standardabweichung σ_x

Niblack 1986

$$t(i, j) = m_x(i, j) + k * \sigma_x(i, j)$$

Umgebungsgröße typ. 15x15, k typ. 0.2

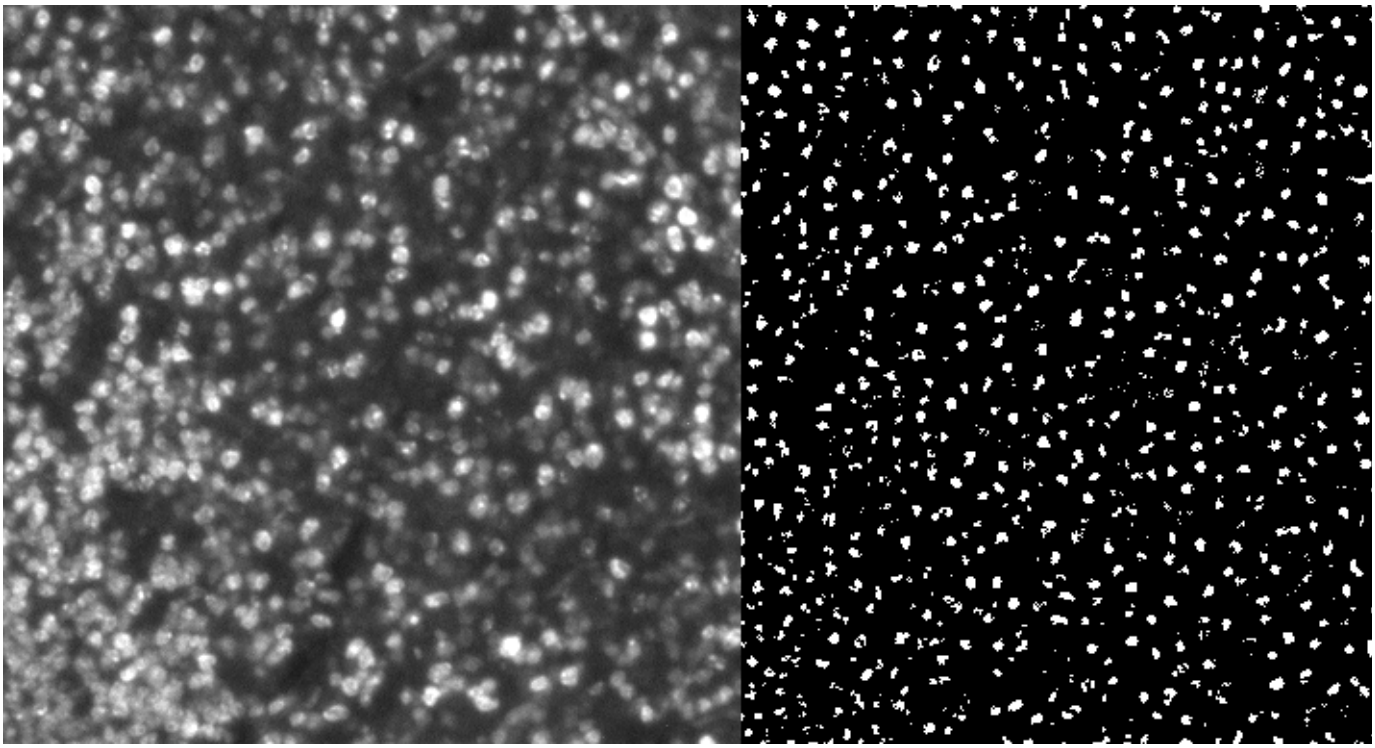
```
Clear[NiblackKern];
NiblackKern = Compile[{{list, _Real, 1}, {sdc, _Real}},
    Module[{mean, stddev, thr},
     mean = Mean[list];
     stddev = StandardDeviation[list];
     thr = mean + sdc * stddev;
     UnitStep[list[[Ceiling[Length[list] / 2]]] - thr]
    ], CompilationTarget → "WVM", Parallelization → False];
(*Niblack, 1986*)
Clear[MyNiblackFilter];
Options[MyNiblackFilter] =
  {"StdDevCoefficient" → 0.2, "WindowHalfWidth" → 15, "Mask" → "Box"};
MyNiblackFilter[im_Image, OptionsPattern[]] := Module[{flatelpos, padim, whw, sdc, el},
    whw = Round[OptionValue["WindowHalfWidth"]];
    sdc = OptionValue["StdDevCoefficient"];
    el = Switch[OptionValue["Mask"],
      "Box", Table[1, {2 whw + 1}, {2 whw + 1}],
      "Circle", Ceiling[Rescale[
        Sign[Table[(x^2 + y^2), {x, -whw, whw}, {y, -whw, whw}] - whw * (whw + 1 / 2)], {1, -1}]],
      _, Table[1, {2 whw + 1}, {2 whw + 1}]
     ];
    flatelpos = Flatten[Position[Flatten[el], 1]];
    padim = ArrayPad[ImageData[im, "Real"], Floor[Dimensions[el] / 2], "Reversed"];
    padim = Developer`PartitionMap[(NiblackKern[Flatten[#, 1][[flatelpos]], sdc]) &,
      padim, Dimensions[el], {1, 1}, Ceiling[Dimensions[el] / 2]];
    Image[ArrayPad[padim, -Floor[Dimensions[el] / 2]], "Bit"]
  ];
```

Lokale Schwellwertbildung (Niblack) mit kreisförmigem Strukturelement (prolif. Zellen)

```
Manipulate[ControlActive[größe, Show[ImageAssemble[
      {#, MyNiblackFilter[#, "StdDevCoefficient" → stdabwkoeff, "WindowHalfWidth" →
          (größe - 1) / 2, "Mask" → "Circle"]}], ImageSize → {2, 1} * ImageDimensions[#]]],
   {{größe, 15, "Niblack-Fenstergröße"}, 1, 61, 2, Appearance → "Labeled"},
   {{stdabwkoeff, 1., "Std.-Abw.-Koeff."}, -2., 2., 0.05, Appearance → "Labeled"},
   ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[First@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```

Niblack–Fenstergröße 15

Std.–Abw.–Koeff. 1.2

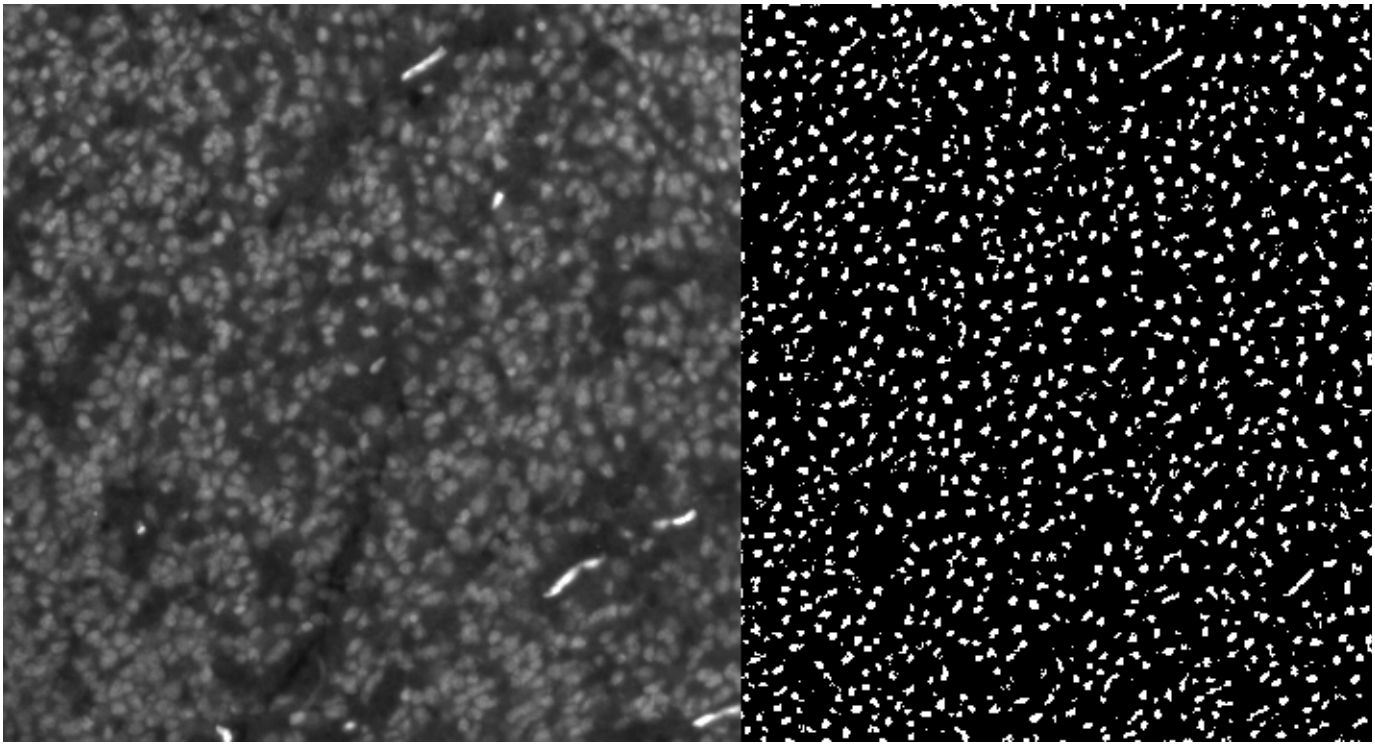


Lokale Schwellwertbildung (Niblack) mit kreisförmigem Strukturelement (B-Zellkerne)

```
Manipulate[ControlActive[größe, Show[ImageAssemble[
      {#, MyNiblackFilter[#, "StdDevCoefficient" → stdabwkoeff, "WindowHalfWidth" →
          (größe - 1) / 2, "Mask" → "Circle"]}], ImageSize → {2, 1} * ImageDimensions[#]]],
   {{größe, 11, "Niblack-Fenstergröße"}, 1, 61, 2, Appearance → "Labeled"},
   {{stdabwkoeff, 1., "Std.-Abw.-Koeff."}, -2., 2., 0.05, Appearance → "Labeled"},
   ContinuousAction → False, SaveDefinitions → True] &[
ImageCrop[First@Rest@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```

Niblack–Fenstergröße 11

Std.–Abw.–Koeff. 1.

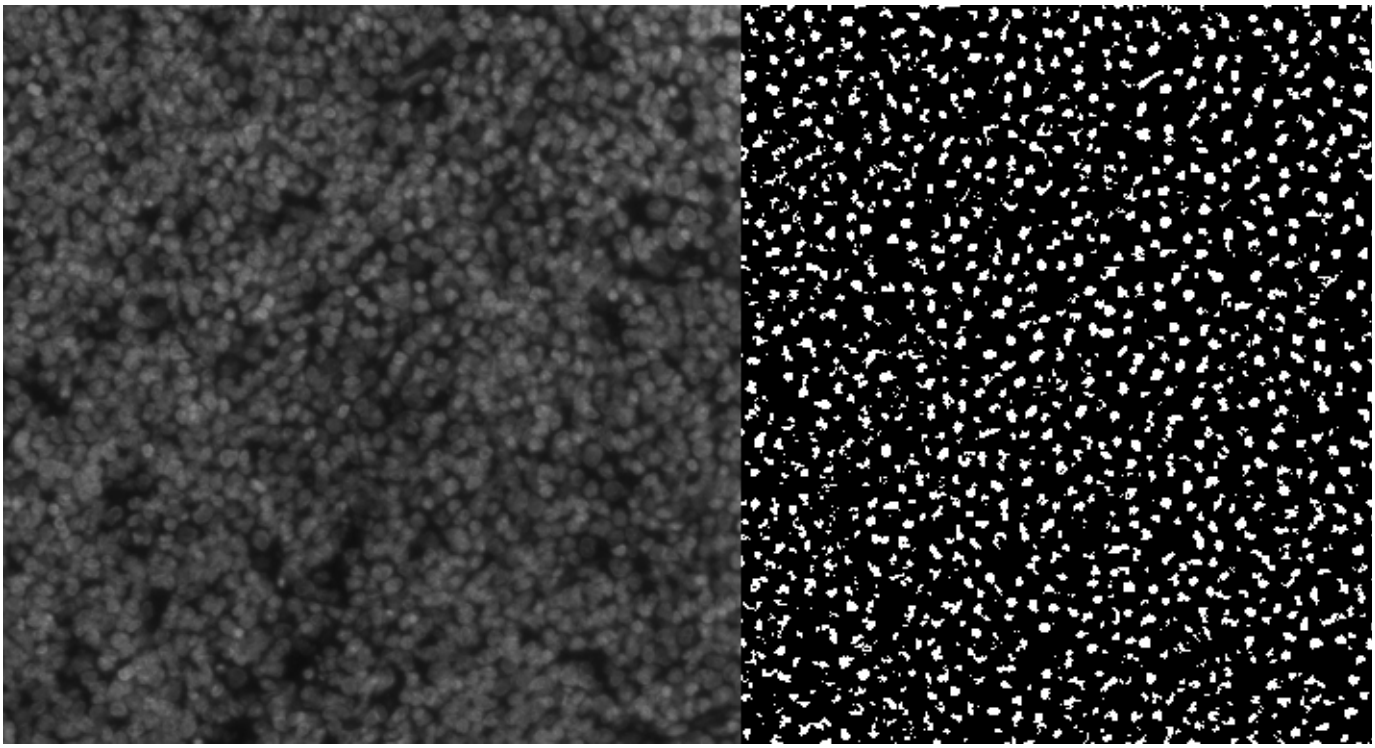


Lokale Schwellwertbildung (Niblack) mit kreisförmigem Strukturelement (alle Zellkerne)

```
Manipulate[ControlActive[größe, Show[ImageAssemble[
      {#, MyNiblackFilter[#, "StdDevCoefficient" → stdabwkoeff, "WindowHalfWidth" →
          (größe - 1) / 2, "Mask" → "Circle"]}], ImageSize → {2, 1} * ImageDimensions[#]]],
   {{größe, 15, "Niblack-Fenstergröße"}, 1, 61, 2, Appearance → "Labeled"},
   {{stdabwkoeff, 1., "Std.-Abw.-Koeff."}, -2., 2., 0.05, Appearance → "Labeled"},
   ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[Last@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```

Niblack–Fenstergröße 15

Std.–Abw.–Koeff. 1.



Modifizierte lokale Schwellwertbildung in Abhängigkeit von lokalem Mittelwert m_x und Standardabweichung σ_x

Sauvola und Pietikäinen 2000

$$t(i,j) = m_x(i,j)\left[1 + k * \left(1 - \frac{\sigma_x(i,j)}{R}\right)\right]$$

Umgebungsgröße typ. 15x15, k typ. 0.2, Dynamikbereichsparameter R typ. 128 (bei 8bit)

```
Clear[SauvolaKern];
SauvolaKern = Compile[{{list, _Real, 1}, {sdc, _Real}, {dr, _Real}},
    Module[{mean, stddev, thr},
     mean = Mean[list];
     stddev = StandardDeviation[list];
     thr = mean * (1 + sdc * (1 - stddev / dr));
     UnitStep[list[[Ceiling[Length[list] / 2]]] - thr]
    ], CompilationTarget → "WVM", Parallelization → False];
(*Sauvola and Pietikäinen, 2000*)
Clear[MySauvolaFilter];
Options[MySauvolaFilter] =
  {"StdDevCoefficient" → 0.5, "DynamicRange" → 128, "WindowHalfWidth" → 15, "Mask" → "Box"};
MySauvolaFilter[im_Image, OptionsPattern[]] := Module[{flatelpos, padim, whw, sdc, dr, el},
    whw = Round[OptionValue["WindowHalfWidth"]];
    sdc = OptionValue["StdDevCoefficient"];
    dr = N[OptionValue["DynamicRange"] / 255];
    el = Switch[OptionValue["Mask"],
      "Box", Table[1, {2 whw + 1}, {2 whw + 1}],
      "Circle", Ceiling[Rescale[
        Sign[Table[(x^2 + y^2), {x, -whw, whw}, {y, -whw, whw}] - whw * (whw + 1 / 2)], {1, -1}]],
      _, Table[1, {2 whw + 1}, {2 whw + 1}]
     ];
    flatelpos = Flatten[Position[Flatten[el], 1]];
    padim = ArrayPad[ImageData[im, "Real"], Floor[Dimensions[el] / 2], "Reversed"];
    padim = Developer`PartitionMap[(SauvolaKern[Flatten[#, 1][[flatelpos]], sdc, dr]) &,
      padim, Dimensions[el], {1, 1}, Ceiling[Dimensions[el] / 2]];
    Image[ArrayPad[padim, -Floor[Dimensions[el] / 2]], "Bit"]
   ];
```

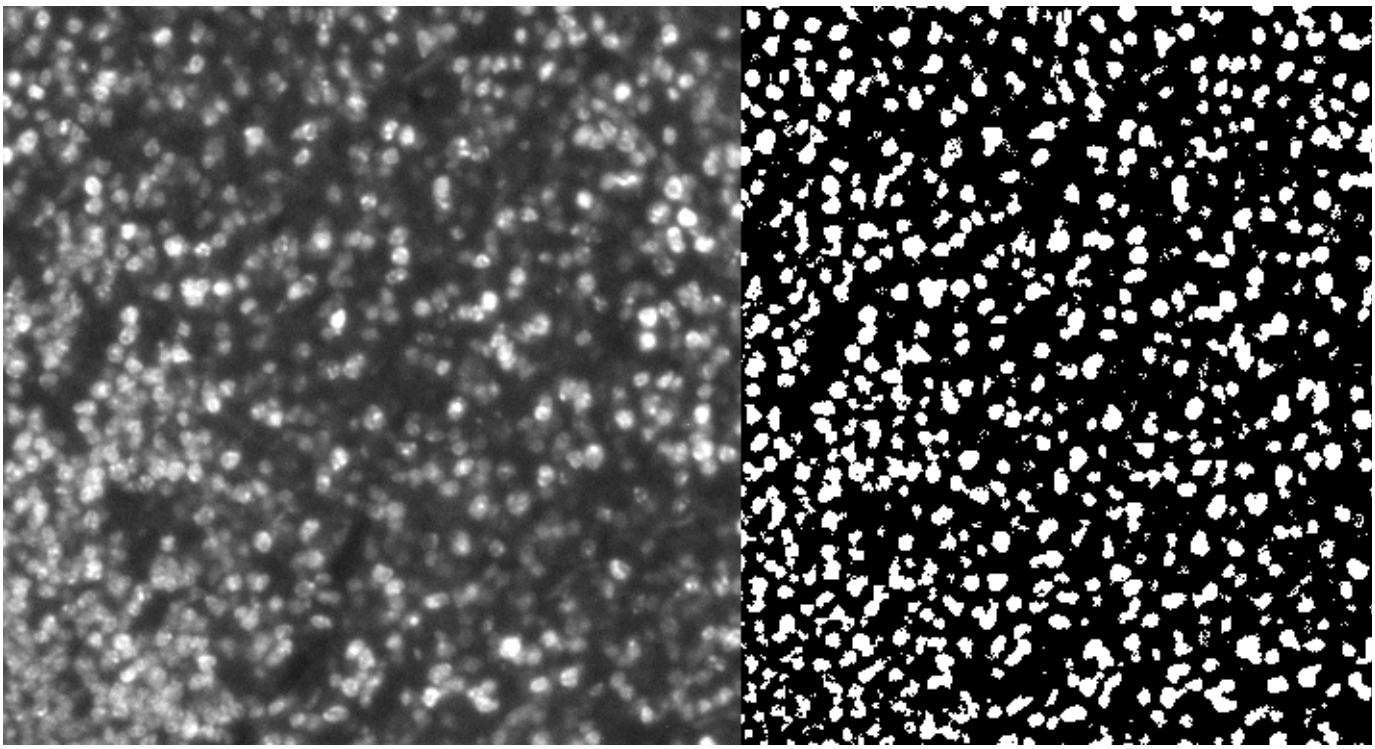
Lokale Schwellwertbildung (Sauvola) mit kreisförmigem Strukturelement (prolif. Zellen)

```
Manipulate[ControlActive[größe,
    Show[ImageAssemble[{#, MySauvolaFilter[#, "StdDevCoefficient" → stdabwkoeff,
        "DynamicRange" → dynamikbereich, "WindowHalfWidth" → (größe - 1) / 2,
        "Mask" → "Circle"]}], ImageSize → {2, 1} * ImageDimensions[#]]],
   {{größe, 15, "Sauvola-Fenstergröße"}, 1, 61, 2, Appearance → "Labeled"},
   {{stdabwkoeff, 0.2, "Std.-Abw.-Koeff."}, -2., 2., 0.05, Appearance → "Labeled"},
   {{dynamikbereich, 64, "Dynamikbereich"}, 1, 255, 1, Appearance → "Labeled"},
   ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[First@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```

Sauvola–Fenstergröße 15

Std.–Abw.–Koeff. 0.2

Dynamikbereich 64



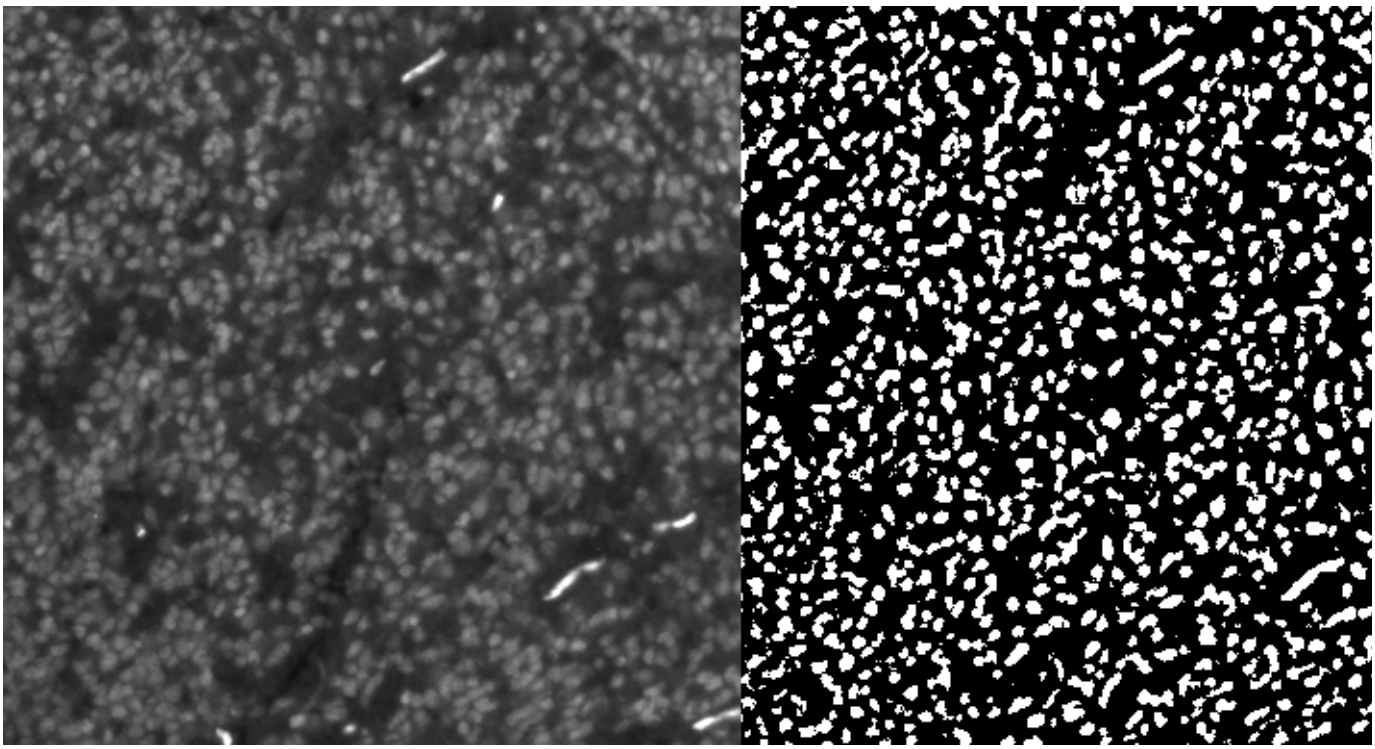
Lokale Schwellwertbildung (Sauvola) mit kreisförmigem Strukturelement (B-Zellkerne)

```
Manipulate[ControlActive[größe,
    Show[ImageAssemble[{#, MySauvolaFilter[#, "StdDevCoefficient" → stdabwkoeff,
        "DynamicRange" → dynamikbereich, "WindowHalfWidth" → (größe - 1) / 2,
        "Mask" → "Circle"]}], ImageSize → {2, 1} * ImageDimensions[#]]],
   {{größe, 15, "Sauvola-Fenstergröße"}, 1, 61, 2, Appearance → "Labeled"},
   {{stdabwkoeff, 0.2, "Std.-Abw.-Koeff."}, -2., 2., 0.05, Appearance → "Labeled"},
   {{dynamikbereich, 42, "Dynamikbereich"}, 0, 255, 1, Appearance → "Labeled"},
   ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[First@Rest@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```

Sauvola–Fenstergröße 15

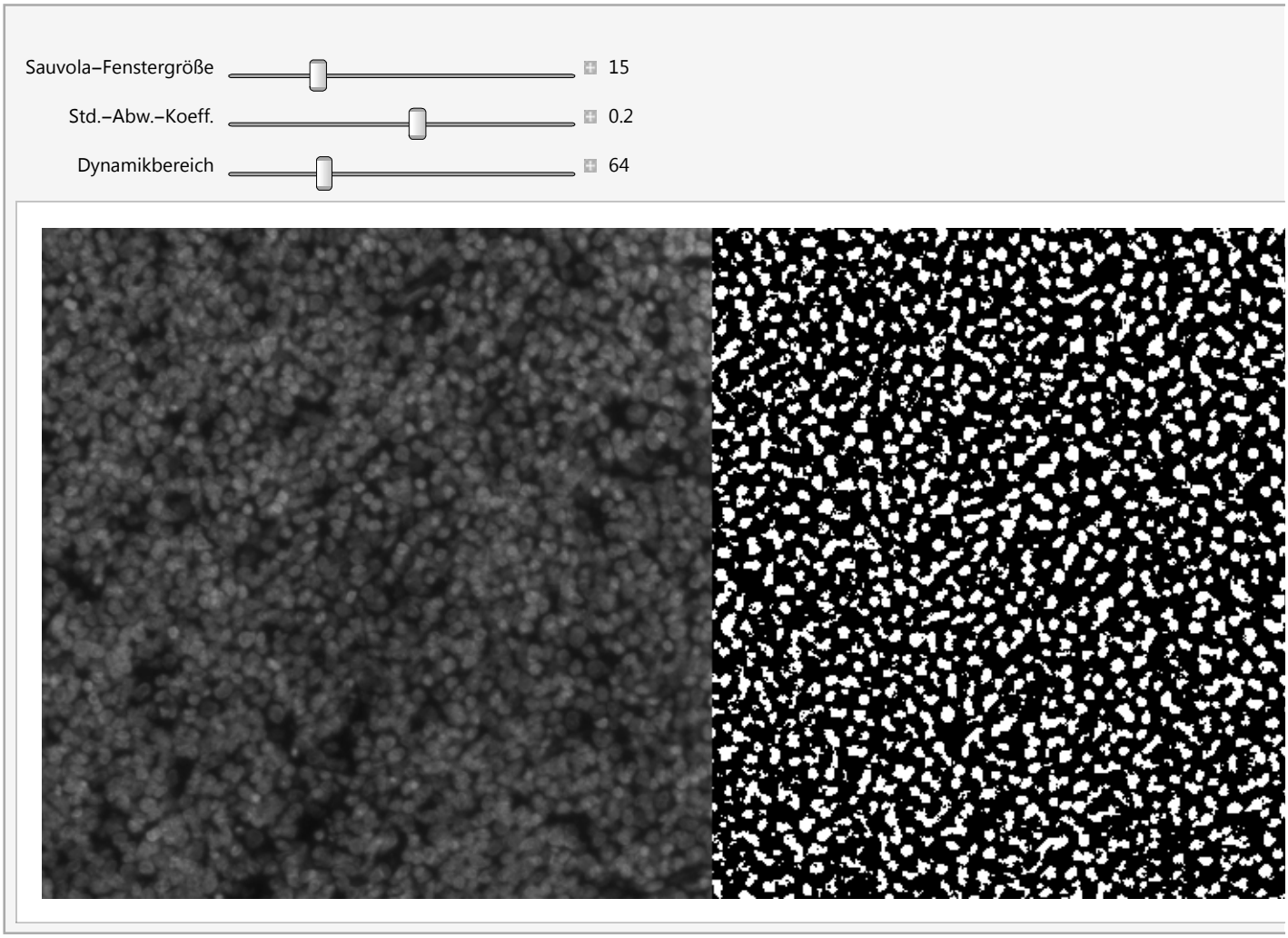
Std.–Abw.–Koeff. 0.2

Dynamikbereich 42



Lokale Schwellwertbildung (Sauvola) mit kreisförmigem Strukturelement (alle Zellkerne)

```
Manipulate[ControlActive[größe,
    Show[ImageAssemble[{#, MySauvolaFilter[#, "StdDevCoefficient" → stdabwkoeff,
        "DynamicRange" → dynamikbereich, "WindowHalfWidth" → (größe - 1) / 2,
        "Mask" → "Circle"]}], ImageSize → {2, 1} * ImageDimensions[#]]],
   {{größe, 15, "Sauvola-Fenstergröße"}, 1, 61, 2, Appearance → "Labeled"},
   {{stdabwkoeff, 0.2, "Std.-Abw.-Koeff."}, -2., 2., 0.05, Appearance → "Labeled"},
   {{dynamikbereich, 64, "Dynamikbereich"}, 0, 255, 1, Appearance → "Labeled"},
   ContinuousAction → False, SaveDefinitions → True] &[
 ImageCrop[Last@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```



Aus 17. Labeling in binären Bildern

um Binärbilder in indizierte Bilder zu überführen

→ jedes einzelne Segment verbundener Pixel bekommt eine “Hausnummer” und ist damit adressierbar

```
Clear[LabelForeground2D];
LabelForeground2D[input_?ArrayQ, threshold_Integer, debug_ : False] :=
  Module[{src = input, n, stack = {}, label = 0,
```

```
 nx, ny, mu, labelimage, i, j, is, js, i1, i2, j1, j2, ii, jj},
n = ArrayDepth[input];
If[n == 2,
 {nx, ny} = Dimensions[src];
 mu = nx * ny;
 src = UnitStep[src - $MachineEpsilon - threshold];
 (*Schwellwert muß überschritten sein*)
 (*src=UnitStep[src-threshold];(*Schwellwert muß erreicht sein*)*)
 labelimage = Developer`ToPackedArray[Table[0, {nx}, {ny}]];
 For[i = 1, i ≤ nx, i++, (*globale Schleife*)
  For[j = 1, j ≤ ny, j++,
    If[src[[i, j]] > 0, (*ist ein Vordergrund-Pixel?*)
      If[labelimage[[i, j]] == 0,
        (*Vordergrund-Pixel wurde noch nicht mit Label versehen!*)
        stack = Prepend[stack, {i, j}];
        (*Lege die Koordinaten
         dieses bisher unmarkierten Pixels auf den Stapelspeicher*)
        labelimage[[i, j]] = ++label; (*inkrementiere den Label-Zähler*)
        If[debug == True, Print[Image[Rescale[labelimage]]]];
        While[Length[stack] > 0,
         {is, js} = First[stack];
         stack = Rest[stack];
         (*entnimm den nächsten Eintrag vom Stapelspeicher*)
         i1 = Max[is - 1, 1];
         (*Definition einer Achter-Nachbarschaft um {is,js},
         wobei ein mögliches Verlassen des Bildbereichs abgefangen wird*)
         i2 = Min[is + 1, nx];
         j1 = Max[js - 1, 1];
         j2 = Min[js + 1, ny];
         For[ii = i1, ii ≤ i2, ii++, (*lokale Schleife innerhalb der Nachbarschaft*)
          For[jj = j1, jj ≤ j2, jj++,
            If[ii != is || jj != js, (*betrachte nur Pixel
              um ein Zentrum herum, nicht das Zentralpixel selbst*)
             If[src[[ii, jj]] > 0, (*falls es sich um ein Vordergrund-Pixel handelt*)
               If[labelimage[[ii, jj]] == 0,
                 (*wurde das Pixel noch nicht mit Label versehen?*)
                 labelimage[[ii, jj]] = label;
                 If[debug == True, Print[Image[Rescale[labelimage]]]];
                 stack = Prepend[stack, {ii, jj}];(*Lege die Koordinaten des
                  eben mit einem Label versehenen Pixels auf den Stack,
                 um später noch dessen Nachbarschaft zu untersuchen*)
                ];
               ];
              ];
            ];
          ];
         ];
        ];
      ];
    ];
 ];
 ,
 Return[$Failed];(*falls es sich nicht um ein Bild (2D-Liste) handelt*)
];
```

```
    Return[labelimage];
   ];
```

Weiterer Test: Labeling nur der Segmente des Testbild, man erkennt, daß erst das zuerst gefundene Segment komplettiert wird, bevor in Scanline-Order ab der zweiten Hälfte der zweiten Zeile nach einem zweiten Segment weitergesucht wird:

```
LabelForeground2D[ImageData@ , 0, True];
```









Automatische Farbzuweisung für die Labels

```
Colorize[LabelForeground2D[ImageData@ , 0]]
```

Ergebnis des Labelings sind willkürliche Labelzuordnungen, die sich aus der jeweiligen Scanline-Order ergeben. Demgegenüber kann es interessanter sein, die Labels in einer bestimmten Sortierreihenfolge zu vergeben, zum Beispiel anhand der Ordnung der Größe der Segmente:

```
Clear[GrößenrichtigesLabeling];
GrößenrichtigesLabeling[bm_] := Module[{lut},
  lut =
   Ordering[Join[{0}, Sort[Rest[Sort[Tally[Flatten[bm]]]], #1[[2]] < #2[[2]] &][[All, 1]]]] - 1;
  Map[lut[[# + 1]] &, bm, {2}]
 ]
```

Je größer die Segmente, desto heller:

```
Image[Rescale[GrößenrichtigesLabeling[LabelForeground2D[ImageData@ , 0]]]]
```

Umkehrung: die kleinsten Segmente sind am hellsten:

```
Image[1 - Rescale[GrößenrichtigesLabeling[LabelForeground2D[ImageData@ , 0]]] /. (1 → 0)]
```

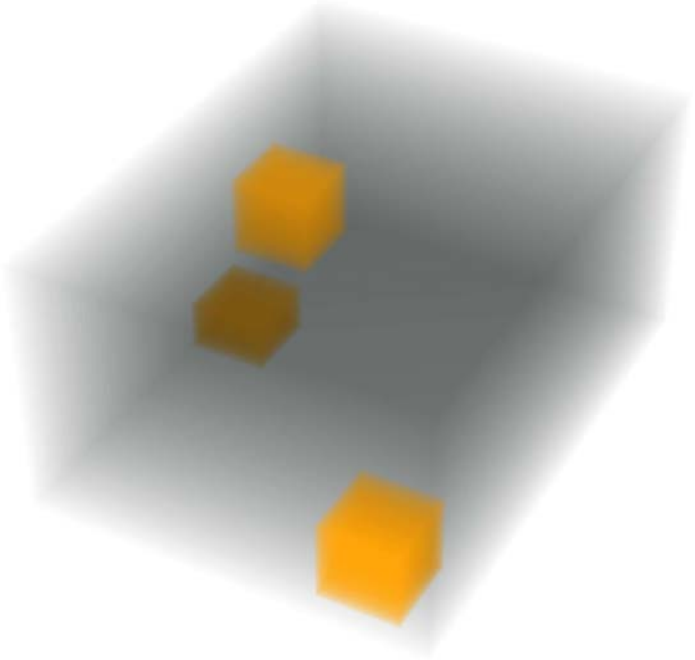
In *Mathematica* ist bereits ein Labeling eingebaut:

```
 // MorphologicalComponents // Colorize
```

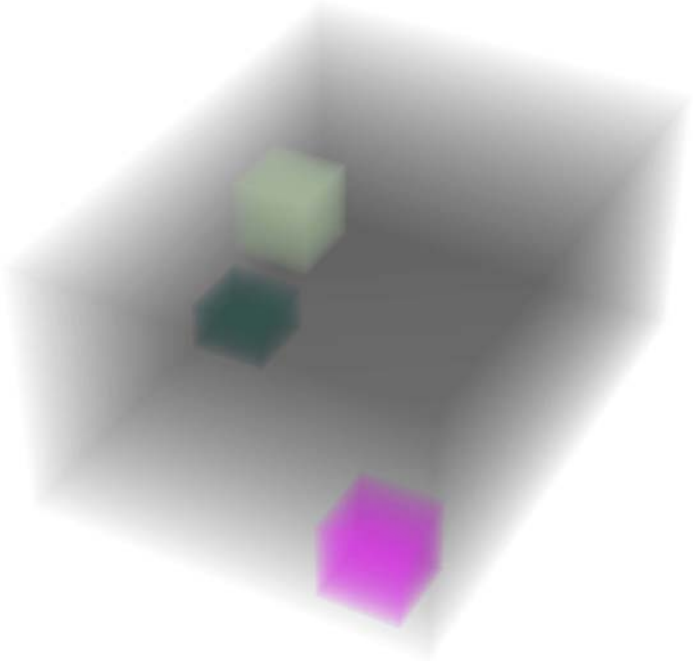
Das Labeling geht auch in 3D:

```
Clear[LabelForeground3D];
LabelForeground3D[input_?ArrayQ, threshold_Integer] :=
  Module[{src = input, n, stack = {}, label = 0, nx, ny, nz,
    mu, labelimage, i, j, k, is, js, ks, i1, i2, j1, j2, k1, k2, ii, jj, kk},
   n = ArrayDepth[input];
   If[n == 3,
    {nx, ny, nz} = Dimensions[src];
    mu = nx * ny * nz;
    src = UnitStep[src - $MachineEpsilon - threshold];
    (*threshold must be exceeded*)
    (*src=UnitStep[src-threshold];(*threshold must be reached*)*)
    labelimage = Developer`ToPackedArray[Table[0, {nx}, {ny}, {nz}]];
    For[i = 1, i ≤ nx, i++,
     For[j = 1, j ≤ ny, j++,
       For[k = 1, k ≤ nz, k++,
         If[src[[i, j, k]] > 0, (*is a foreground voxel*)
           If[labelimage[[i, j, k]] == 0, (*the voxel was not labeled yet*)
             stack = Prepend[stack, {i, j, k}]; (*Push*)
             labelimage[[i, j, k]] = ++label;
             While[Length[stack] > 0,
              {is, js, ks} = First[stack]; stack = Rest[stack]; (*Pop*)
              i1 = Max[is - 1, 1];
              i2 = Min[is + 1, nx];
              j1 = Max[js - 1, 1];
              j2 = Min[js + 1, ny];
              k1 = Max[ks - 1, 1];
              k2 = Min[ks + 1, nz];
              For[ii = i1, ii ≤ i2, ii++,
               For[jj = j1, jj ≤ j2, jj++,
                  For[kk = k1, kk ≤ k2, kk++,
                    If[ii != is || jj != js || kk != ks, (*only consider non-center voxels*)
                      If[src[[ii, jj, kk]] > 0, (*is a foreground voxel*)
                        If[labelimage[[ii, jj, kk]] == 0, (*the voxel was not labeled yet*)
                          labelimage[[ii, jj, kk]] = label;
                          stack = Prepend[stack, {ii, jj, kk}];(*Push*)
                         ];
                       ];
                     ];
                   ];
                 ];
               ];
             ];
           ];
         ];
       ];
     ];
    ,
    Return[$Failed];(*input dimensions do not equal 3*)
   ];
   Return[labelimage];
  ];
```

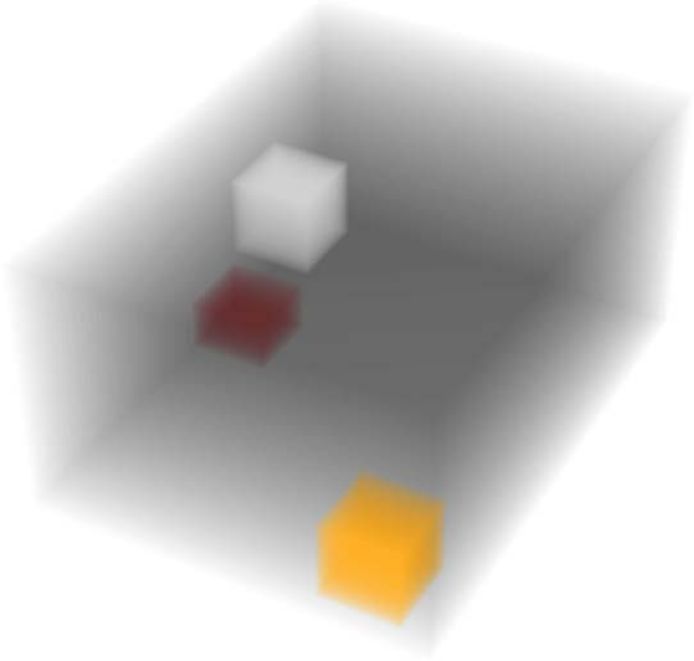
```
SeedRandom[2405];
img3d =
 Dilation[ColorNegate[Image3D[Table[If[RandomReal[] > N[1 / 1000], 1, 0], {10}, {20}, {15}]]], 1]
```



```
LabelForeground3D[ImageData[img3d, "Bit"], 0] // Colorize
```



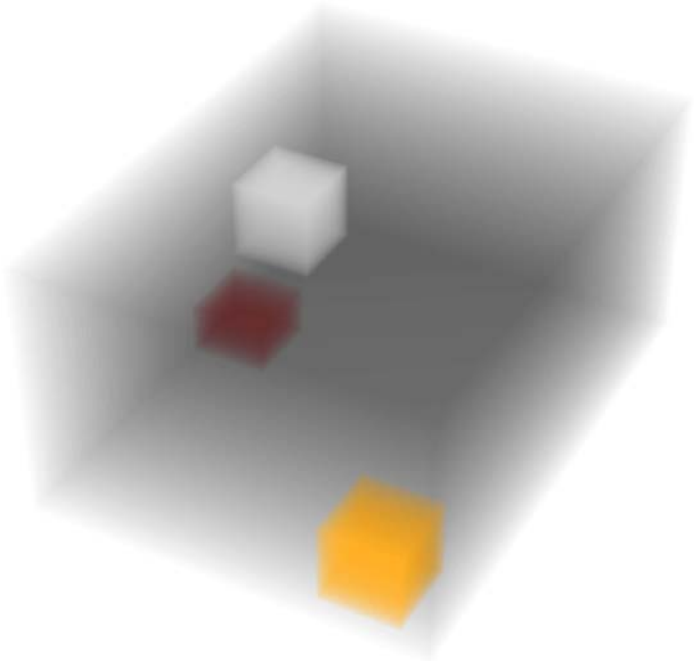
```
farbfunktion[wert_] := ColorData["SunsetColors"][wert];
Colorize[#1, ColorFunction → Function[{v}, farbfunktion[v / #2]],
      ColorFunctionScaling → False] &[GrößenrichtigesLabeling[#], Max[#]] &[
 LabelForeground3D[ImageData[img3d, "Bit"], 0]]
```



Mathematica unterstützt auch 3D-Labeling direkt:

```
MorphologicalComponents[img3d] // Colorize
```

```
farbfunktion[wert_] := ColorData["SunsetColors"][wert];
Colorize[#1,
      ColorFunction → Function[{v}, farbfunktion[v / #2]], ColorFunctionScaling → False] &[
   GrößenrichtigesLabeling[#], Max[#]] &[MorphologicalComponents[img3d]]
```



Aus 18. Binarisierung mit dem Otsu-Verfahren

nach Nobuyuki Otsu (1979)

1. Ziel des Verfahrens ist es, die Varianz der skalaren Bildwerte innerhalb der zu findenden zwei Klassen zu minimieren
2. Ziel ist es ebenso, die Varianz zwischen den zu findenden zwei Klassen zu maximieren
3. Das Verfahren setzt auf diskretisierten Daten auf bzw. erfordert ein Binning mit einer bestimmten Intervallvorgabe

Formelmäßiger Hintergrund

$H(i)$: absolute Häufigkeit von skalaren Werten im Intervall i, $0 \le i < L$

$p_i = P(i) = H(i) / (N \cdot M)$: Auftrittswahrscheinlichkeit für Werte im Intervall i

$\sum_{i=0}^{L-1} p_i = 1$

$P_b(k) = \sum_{i=0}^{k} p_i$: Zugehörigkeitswahrscheinlichkeit
für die Hintergrundklasse C_b bei gegebenem Schwellwert k

$P_f(k) = \sum_{i=k+1}^{L-1} p_i = 1 - P_b(k)$: Zugehörigkeitswahrscheinlichkeit
für die Vordergrundklasse C_f bei gegebenem Schwellwert k

$\mu_T = E(x) = \frac{1}{N \cdot M} \sum_{n=1}^{N} \sum_{m=1}^{M} x_{nm} = \sum_{i=0}^{L-1} i \cdot p_i$: Gesamtmittelwert

$\mu_b(k) = E(x \mid x \le k) =$
$\sum_{i=0}^{k} i \cdot P(i \mid C_b) = \sum_{i=0}^{k} i \cdot P(C_b \mid i) \cdot P(i) / P(C_b) = \frac{1}{P_b(k)} \sum_{i=0}^{k} i \cdot p_i$: Mittelwert für
die Hintergrundklasse C_b bei gegebenem Schwellwert k
mit $P(C_b \mid i) = 1$, da nur i von C_b betrachtet, sowie $P(C_b) = P_b(k)$

$\mu(k) = \sum_{i=0}^{k} i \cdot p_i = \mu_b(k) \cdot P_b(k)$: kumulatives Mittel bis zum Level k

$\mu_f(k) = E(x \mid x > k) =$
$\sum_{i=0}^{k} i \cdot P(i \mid C_f) = \sum_{i=k+1}^{L-1} i \cdot P(C_f \mid i) \cdot P(i) / P(C_f) = \frac{1}{P_f(k)} \sum_{i=k+1}^{L-1} i \cdot p_i$: Mittelwert für
die Vordergrundklasse C_f bei gegebenem Schwellwert k
mit $P(C_f \mid i) = 1$, da nur i von C_f betrachtet, sowie $P(C_f) = P_f(k)$

$\sigma^2{}_T =$
$V(x) = E\left((x - \mu_T)^2\right) = \frac{1}{N \cdot M} \sum_{n=1}^{N} \sum_{m=1}^{M} (x_{nm} - \mu_T)^2 = \sum_{i=0}^{L-1} (i - \mu_T)^2 \cdot p_i$: Gesamtvarianz

$\sigma^2{}_b(k) = V(x \mid x \le k) = E\left((x - \mu_T)^2 \mid x \le k\right) = \frac{1}{P_b(k)} \sum_{i=0}^{k} (i - \mu_b)^2 \cdot$
p_i : Varianz für die Hintergrundklasse C_b bei gegebenem Schwellwert k

$\sigma^2_f(k) = V(x \mid x > k) = E\left((x - \mu_T)^2 \mid x > k\right) = \frac{1}{P_f(k)} \sum_{i=k+1}^{L-1} (i - \mu_f)^2 \cdot$

p_i : Varianz für die Vordergrundklasse C_f bei gegebenem Schwellwert k

$\sigma^2_W(k) = P_b(k) \cdot \sigma^2_b(k) + P_f(k) \cdot \sigma^2_f(k)$:

Varianz innerhalb beider Klassen bei gegebenem Schwellwert k

$\sigma^2_B(k) = \sigma^2_T - \sigma^2_W(k) = P_b(k) \cdot (\mu_b(k) - \mu_T)^2 + P_f(k) \cdot (\mu_f(k) - \mu_T)^2$:

Varianz zwischen beiden Klassen bei gegebenem Schwellwert k

$\frac{\sigma^2_B(k)}{\sigma^2_W(k)} \rightarrow$ Max

Zusammenfassung von Version 4 des Otsu-Binarisierungsverfahrens:

1. Berechne das normierte Histogramm des skalaren Eingangbildes, also die p_i
2. Berechne die kumulativen Summen $P_b(k)$ für alle $k = 0, \ldots, L - 1$
3. Berechne die kumulativen Mittel $\mu(k)$ für alle $k = 0, \ldots, L - 1$
4. Berechne das globale Mittel μ_T
5. Berechne daraus die Zwischen-Klassen-Varianz $\sigma^2_B(k)$ für alle $k = 0, \ldots, L - 1$
6. Bestimme das k^*, für das $\sigma^2_B(k^*) = \text{Max}\left(\sigma^2_B(k)\right)$ für $0 \le k \le L - 1$

Version 4: Maximierung nur der Varianz zwischen beiden Klassen, Variante b

Ansatz über weitere Vereinfachung von $\sigma^2_B(k) = (\mu_b(k) - \mu_f(k))^2 \cdot P_b(k) \cdot P_f(k)$, so daß auf die wiederkehrende Berechnung von Varianzen für Vorder- und Hintergrundklasse ganz verzichtet werden kann!

$$\sigma^2_B(k) = (\mu_b(k) - \mu_f(k))^2 \cdot P_b(k) \cdot P_f(k)$$

$$\text{mit } \mu_b(k) = \mu(k) / P_b(k)$$

$$\text{und mit } \mu_f(k) = \frac{\mu_T - P_b(k) \cdot \mu_b(k)}{P_f(k)} = \frac{\mu_T - \mu(k)}{P_f(k)} \quad \text{wegen} \quad \mu_T = P_b(k) \cdot \mu_b(k) + P_f(k) \cdot \mu_f(k)$$

$$\sigma^2_B(k) = \left(\frac{\mu(k)}{P_b(k)} - \frac{\mu_T - \mu(k)}{P_f(k)} \right)^2 \cdot P_b(k) \cdot P_f(k)$$

$$= \left(\frac{\mu(k) \cdot P_f(k)}{P_b(k) \cdot P_f(k)} - \frac{(\mu_T - \mu(k)) \cdot P_b(k)}{P_f(k) \cdot P_b(k)} \right)^2 \cdot P_b(k) \cdot P_f(k)$$

$$= \left(\frac{\mu(k) \cdot P_f(k) - (\mu_T - \mu(k)) \cdot P_b(k)}{P_f(k) \cdot P_b(k)} \right)^2 \cdot P_b(k) \cdot P_f(k)$$

$$= (\mu(k) \cdot P_f(k) - (\mu_T - \mu(k)) \cdot P_b(k))^2 \cdot \frac{1}{P_b(k) \cdot P_f(k)}$$

$$= (\mu(k) \cdot P_f(k) - \mu_T \cdot P_b(k) + \mu(k) \cdot P_b(k))^2 \cdot \frac{1}{P_b(k) \cdot P_f(k)}$$

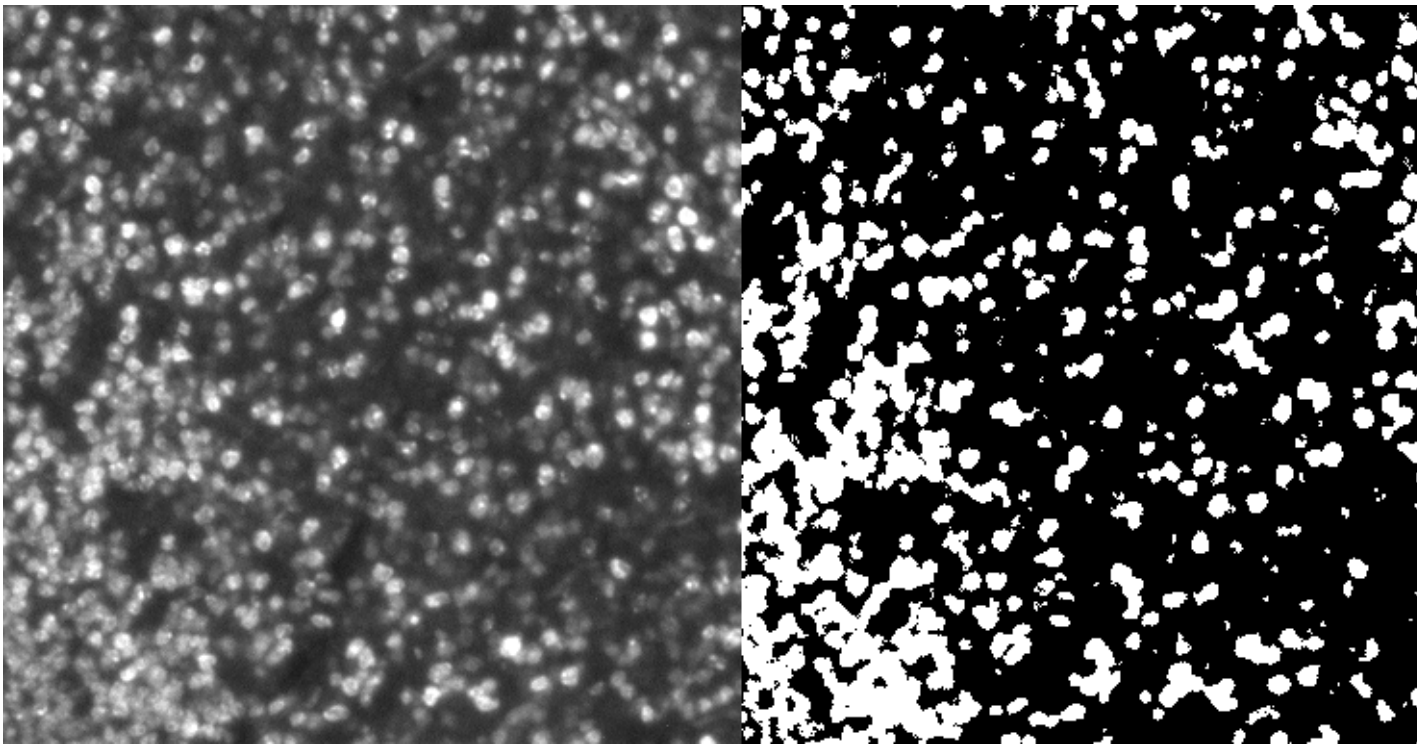
$$= (\mu(k) \cdot (1 - P_b(k)) - \mu_T \cdot P_b(k) + \mu(k) \cdot P_b(k))^2 \cdot \frac{1}{P_b(k) \cdot P_f(k)}$$

$$= (\mu(k) - \mu(k) \cdot P_b(k) - \mu_T \cdot P_b(k) + \mu(k) \cdot P_b(k))^2 \cdot \frac{1}{P_b(k) \cdot P_f(k)}$$

$$= \frac{(\mu(k) - \mu_T \cdot P_b(k))^2}{P_b(k) \cdot (1 - P_b(k))}$$

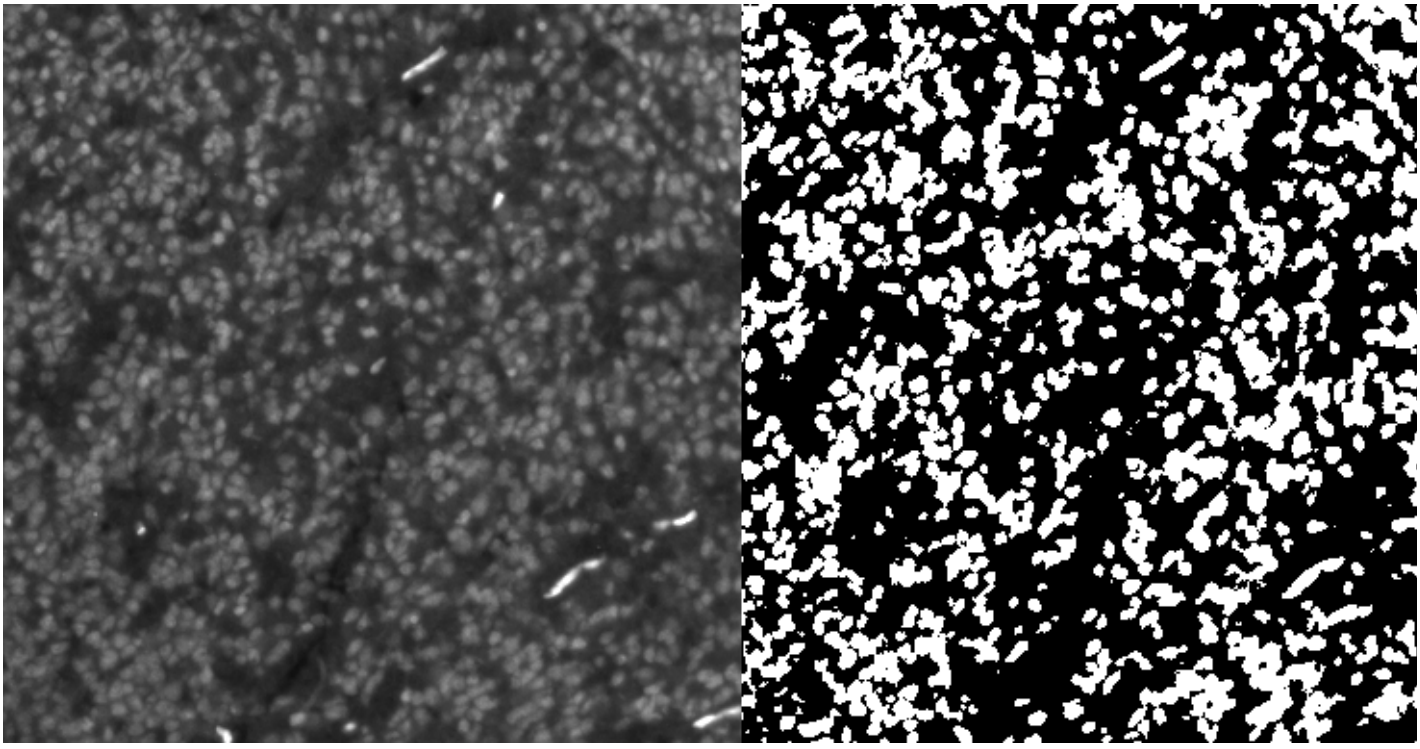
```
Clear[OtsuVersion4];
(*Maximize Between-class variance, Variante b*)
OtsuVersion4[bm_] := Module[
  {minbm = Min[bm], maxbm = Max[bm], histogramm, k, kopt, μT, ωf, ωb, μbtωb, μb, μf, σqB, σqBmax},
  histogramm = N[BinCounts[Flatten[bm], {0, maxbm + 1, 1}]] / (Times @@ Dimensions[bm]);
  (*normalized overall histogramm, contains all p_i*)
  (*μT=N[Mean[Flatten[bm]]];*)
  μT = Total[Range[0, maxbm] * histogramm[[1 ;; maxbm + 1]]];
  (*Total, overall mean*)
  σqBmax = 0.;
  (*σqBmax is the maximum Between-class variance*)
  kopt = 0;
  (*kopt is the k-1 for best threshold*)
  For[k = minbm + 1, k ≤ maxbm, k++,
   ωb = Total[histogramm[[1 ;; k]]];
   (*ωb is the k-dependent weight for background, i.e. P_b(k)*)
   ωf = (1 - ωb);
   (*ωf is the k-dependent weight for foreground, i.e. P_f(k)*)
   μbtωb = Total[Range[0, k - 1] * histogramm[[1 ;; k]]];
   (*μbtωb is the k-dependent mean for background times weight for background,
   i.e. kumulative mean up to level k*)
   If[ωb ≠ 1. && ωb ≠ 0., σqB = (μT * ωb - μbtωb) * (μT * ωb - μbtωb) / (ωb * ωf), σqB = 0.];
   (*σqB is the k-dependent Between-class variance*)
   If[σqB > σqBmax, σqBmax = σqB; kopt = k - 1];
   (*update kopt according to σqBmax*)
   (*Print[{{k,ωb,ωf,σqB}}];*)
  ];
  Print[kopt];
  kopt
 ]

Show[ImageAssemble[{Image[#, "Byte"], Image[MyBinarize[#, OtsuVersion4[#]]]}],
   ImageSize → {2, 1} * Reverse@Dimensions[#]] &[
 ImageData[ImageCrop[First@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}], "Byte"]]
```

```
113
```



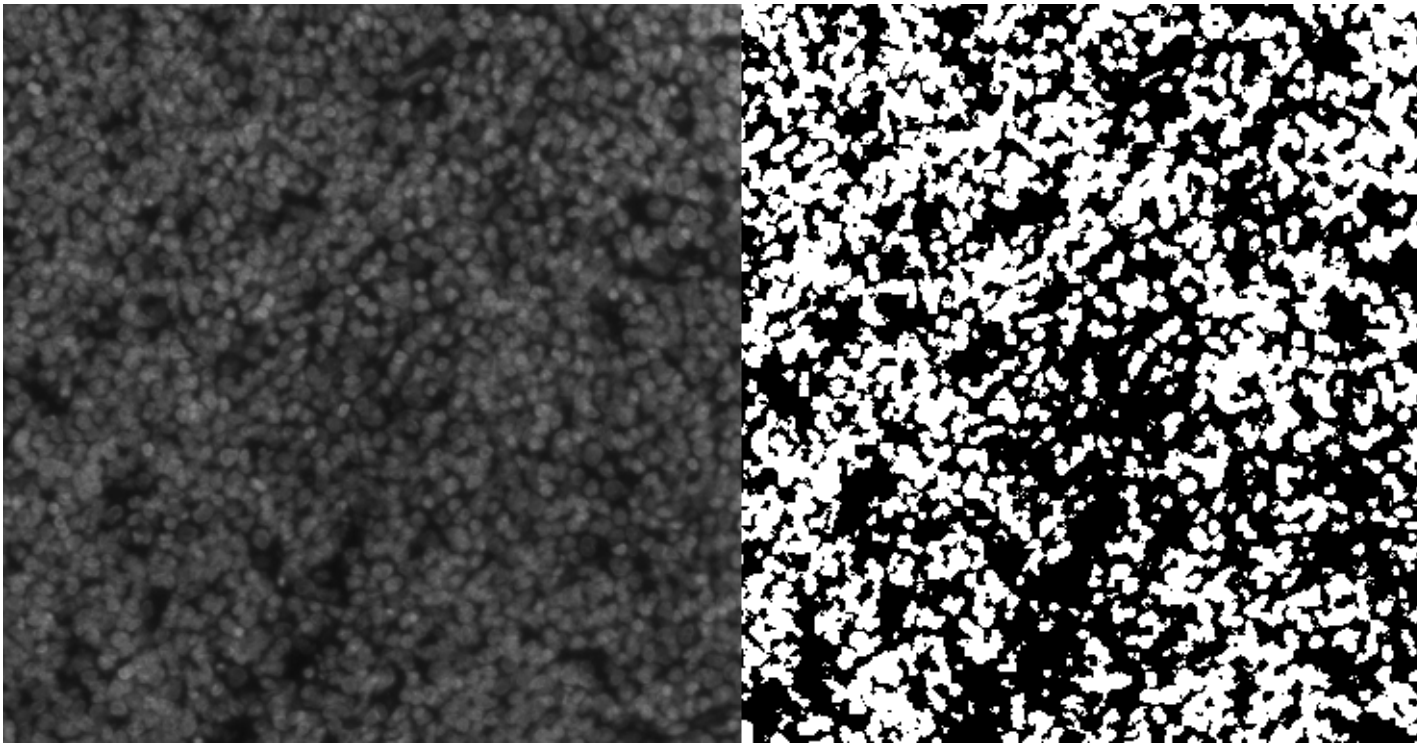
```
Show[ImageAssemble[{Image[#, "Byte"], Image[MyBinarize[#, OtsuVersion4[#]]]}],
    ImageSize → {2, 1} * Reverse@Dimensions[#]] &[ImageData[
  ImageCrop[First@Rest@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}], "Byte"]]
```

78



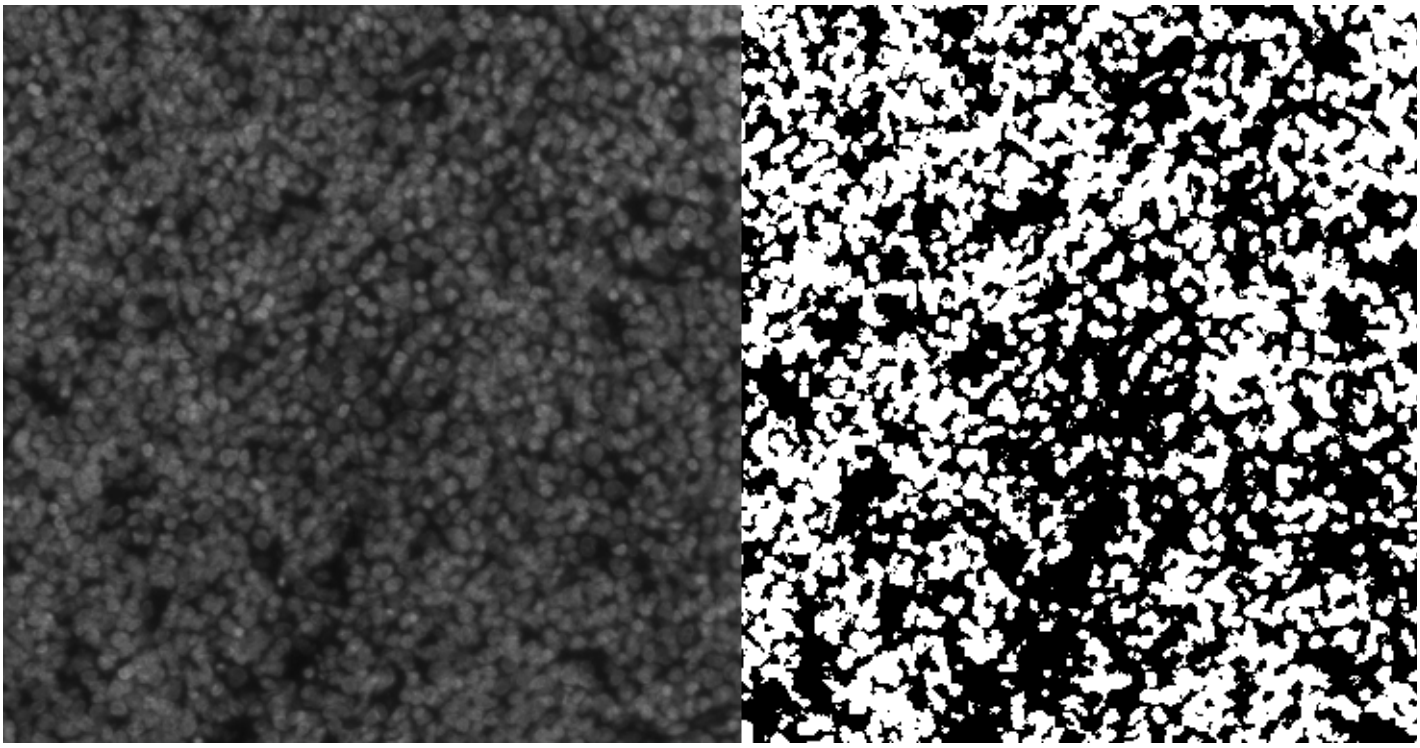
```
Show[ImageAssemble[{Image[#, "Byte"], Image[MyBinarize[#, OtsuVersion4[#]]]}],
    ImageSize → {2, 1} * Reverse@Dimensions[#]] &[
 ImageData[ImageCrop[Last@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}], "Byte"]]
```

61



In *Mathematica* ist bereits eine Funktion eingebaut, die das Otsu-Verfahren realisiert:

```
Show[ImageAssemble[{#, Binarize[#, Method → "Cluster"]}],
   ImageSize → {2, 1} * ImageDimensions[#]] &[
 ImageCrop[Last@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]
```



```
FindThreshold[#, Method → "Cluster"] &[
  ImageCrop[Last@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]] * 255
```

```
61.
```

Aus 19. Binarisierung mit dem ISODATA-Verfahren

Iterative Self-Organising Data Analysis Technique (ISODATA, Ridler und Calvard 1978)

→ “Picture thresholding using an iterative selection method”

→ kann als Spezialfall des (nachfolgend dargestellten) k-Means-Verfahren angesehen werden (2-Means, skalare Daten)

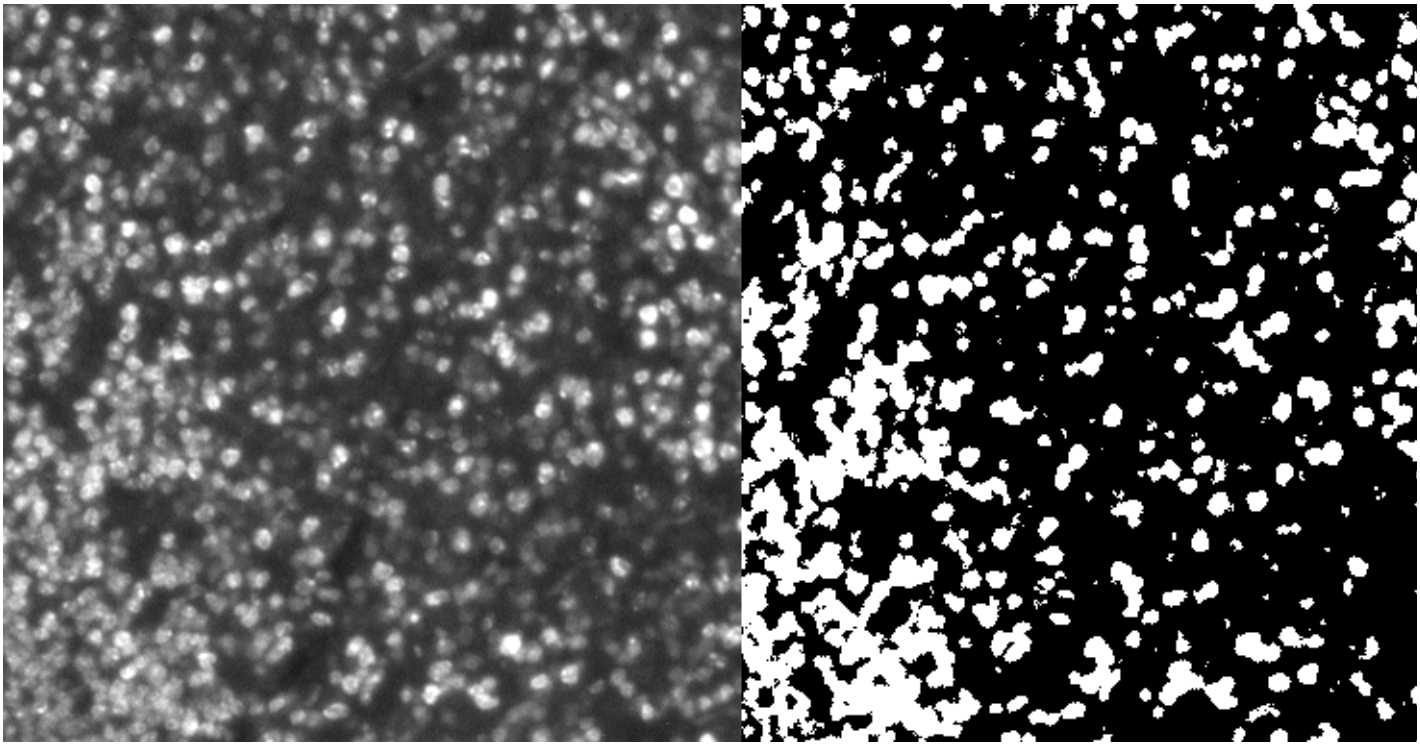
1. Bestimmung eines Startschwellwertes
2. Einteilung aller Bildwerte und unter- und überschwellige Mengen
3. Bestimmung jeweils eines Mittelwertes für beide Mengen
4. Bestimmung eines neues Schwellwertes als Mittelwert beider Mittelwerte
5. Wiederholung der Schritte 2 bis 4, bis Mittelwerte konvergieren und die Mengeneinteilung unverändert bleibt

```
Clear[MyISODATA];
MyISODATA[bm_] := Module[{bin, pos, neueschwelle, mean0,
    mean1, lenpos0, lenpos1, startschwellwert = N[(Max[bm] - Min[bm]) / 2]},
   Print[{{"N.A.", "N.A."}, N[Mean[Map[bm[[Sequence @@ #]] &, Position[
           MyBinarize[bm, startschwellwert], #], {1}]] & /@ {0, 1}], startschwellwert}];
   Clear[schritte];
   schritte[alteschwelle_] := Module[{},
     bin = Map[(If[# < alteschwelle, 0, 1]) &, bm, {2}];
     pos = Position[bin, 0];
     lenpos0 = Length[pos];
     If[pos ≠ {}, mean0 = N[Mean[Map[bm[[Sequence @@ #]] &, pos, {1}]]], mean0 = 0.];
     pos = Position[bin, 1];
     lenpos1 = Length[pos];
     If[pos ≠ {}, mean1 = N[Mean[Map[bm[[Sequence @@ #]] &, pos, {1}]]], mean1 = 0.];
     neueschwelle = N[(mean0 + mean1) / 2];
     Print[{{lenpos0, lenpos1}, {mean0, mean1}, neueschwelle}];
     neueschwelle
    ];
   FixedPoint[schritte, N[startschwellwert]]
  ];

daten = {{1, 2, 3}, {2, 3, 4}, {0, 2, 4}, {2, 1, 0}};
MyISODATA[daten]
```

```
{{N.A., N.A.}, {1.25, 3.5}, 2.}
{{4, 8}, {0.5, 2.75}, 1.625}
{{4, 8}, {0.5, 2.75}, 1.625}
1.625
```

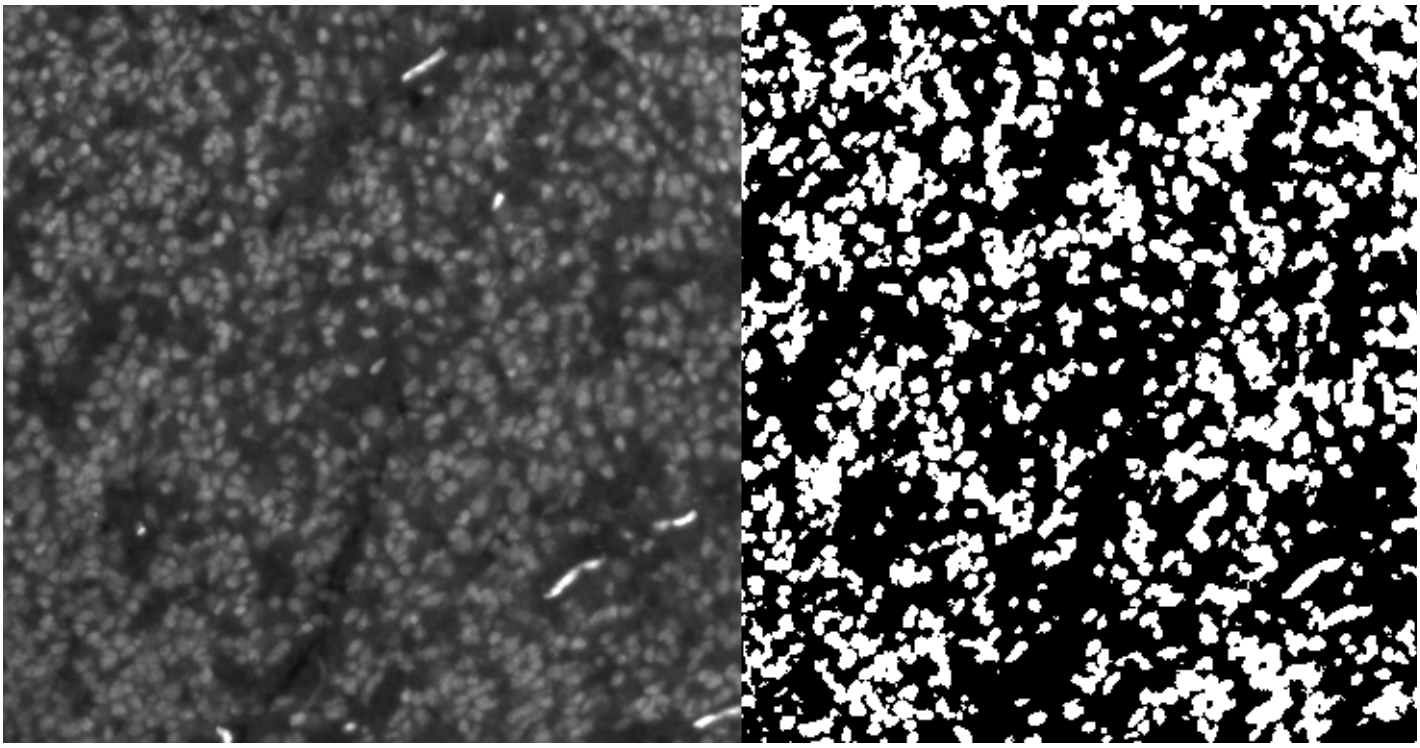
```
Show[ImageAssemble[{Image[#1, "Byte"], Image[MyBinarize[ImageData[#1, "Byte"], #2]]}],
   ImageSize → {2, 1} * ImageDimensions[#1]] &[
 Sequence @@ ({#, MyISODATA[ImageData[#, "Real"] * 255]}) &[
  ImageCrop[First@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]]
```

```
{{N.A., N.A.}, {72.0914, 157.622}, 115.5}
{{111 995, 35 461}, {72.0914, 157.622}, 114.857}
{{111 205, 36 251}, {71.7866, 156.693}, 114.24}
{{111 205, 36 251}, {71.7866, 156.693}, 114.24}
```



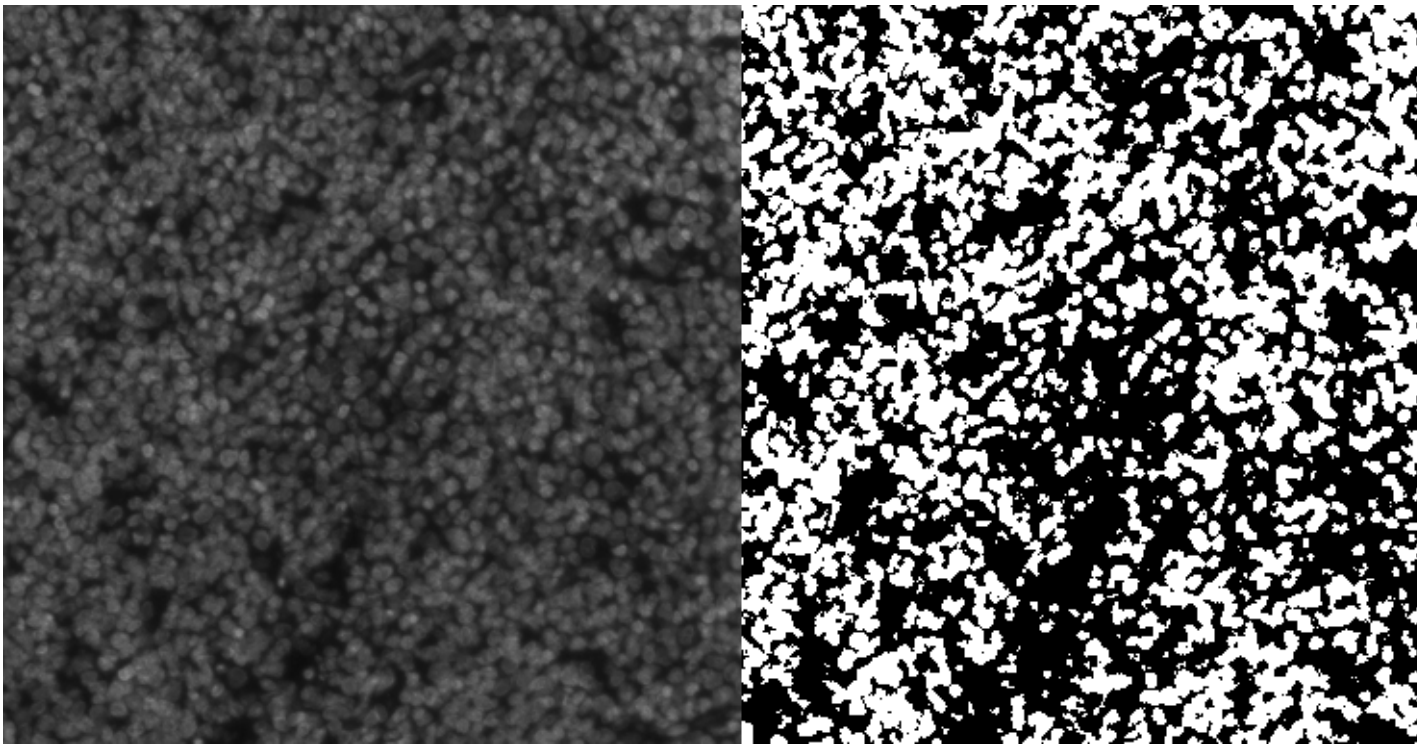
```
Show[ImageAssemble[{Image[#1, "Byte"], Image[MyBinarize[ImageData[#1, "Byte"], #2]]}],
    ImageSize → {2, 1} * ImageDimensions[#1]] &[
 Sequence @@ ({#, MyISODATA[ImageData[#, "Real"] * 255]}) &[
  ImageCrop[First@Rest@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]]
```

```
{{N.A., N.A.}, {69.675, 137.178}, 120.}
{{141 207, 6249}, {69.516, 135.952}, 102.734}
{{129 353, 18 103}, {65.8167, 118.882}, 92.3495}
{{117 853, 29 603}, {62.7537, 110.462}, 86.6077}
{{109 420, 38 036}, {60.703, 105.784}, 83.2434}
{{104 767, 42 689}, {59.6234, 103.52}, 81.5715}
{{101 271, 46 185}, {58.8336, 101.929}, 80.3811}
{{99 434, 48 022}, {58.4241, 101.128}, 79.7761}
{{97 798, 49 658}, {58.0632, 100.432}, 79.2476}
{{97 798, 49 658}, {58.0632, 100.432}, 79.2476}
```



```
Show[ImageAssemble[{Image[#1, "Byte"], Image[MyBinarize[ImageData[#1, "Byte"], #2]]}],
    ImageSize → {2, 1} * ImageDimensions[#1]] &[
 Sequence @@ ({#, MyISODATA[ImageData[#, "Real"] * 255]}) &[
  ImageCrop[Last@ColorSeparate@Ton2CD21dreikanalausgleichF1, {384, 384}]]]
```

```
{{N.A., N.A.}, {51.4083, 87.6163}, 74.5}
{{110 663, 36 793}, {51.4083, 87.6163}, 69.5123}
{{99 150, 48 306}, {49.0194, 83.8901}, 66.4547}
{{91 361, 56 095}, {47.4018, 81.6827}, 64.5422}
{{86 221, 61 235}, {46.3225, 80.3248}, 63.3237}
{{83 573, 63 883}, {45.7624, 79.6482}, 62.7053}
{{80 932, 66 524}, {45.1999, 78.9872}, 62.0936}
{{80 932, 66 524}, {45.1999, 78.9872}, 62.0936}
```



Verallgemeinerung des ISODATA-Verfahrens für vektoriellen Daten (2-Means)

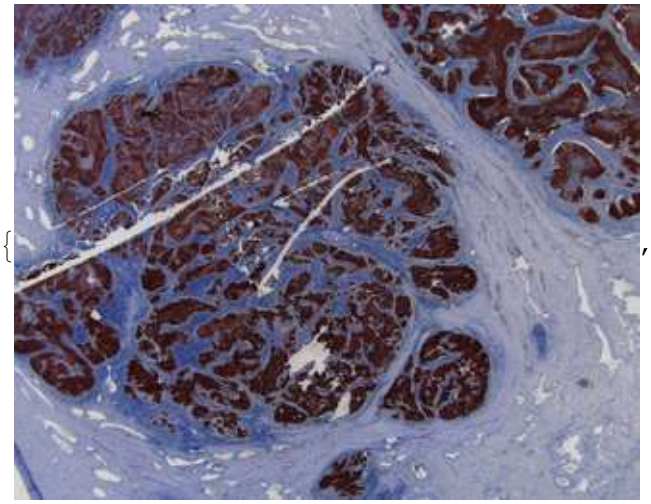
1. Bestimmung zweier Startmittelwertvektoren
2. Einteilung aller Bildwerte in zwei Mengen entsprechend der je Pixel kürzesten Euklidschen Distanz zu einem der Mittelwertvektoren
3. Bestimmung jeweils eines neuen Mittelwertvektors für beide Mengen
4. Wiederholung der Schritte 2 bis 3, bis Mittelwertvektoren konvergieren und die Mengeneinteilung unverändert bleibt

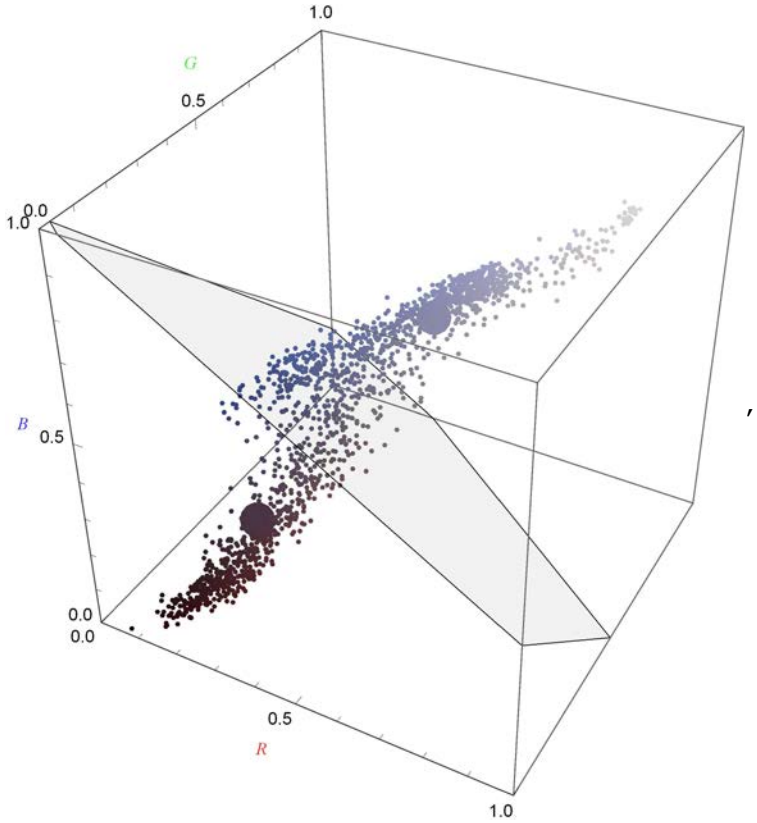
```
Clear[MyISODATARGB];
MyISODATARGB[bm_] :=
  Module[{bin, pos, neuesmean0, neuesmean1, lenpos0, lenpos1,
    altesmean0 = N[Mean[Flatten[bm, 1]] - .1], altesmean1 = N[Mean[Flatten[bm, 1]] + .1]},
   Print[{{"N.A.", "N.A."}, {altesmean0, altesmean1}}];
   Clear[schrittergb];
   schrittergb[{altesmean0_, altesmean1_}] := Module[{},
     bin = Map[(If[EuclideanDistance[altesmean0, #] < EuclideanDistance[altesmean1, #], 0, 1]) &,
       bm, {2}];
     pos = Position[bin, 0];
     lenpos0 = Length[pos];
     If[pos ≠ {},
      neuesmean0 = N[Mean[Map[bm[[Sequence @@ #]] &, pos, {1}]]], neuesmean0 = altesmean1];
     pos = Position[bin, 1];
     lenpos1 = Length[pos];
     If[pos ≠ {},
      neuesmean1 = N[Mean[Map[bm[[Sequence @@ #]] &, pos, {1}]]], neuesmean1 = altesmean0];
     Print[{{lenpos0, lenpos1}, {neuesmean0, neuesmean1}}];
     {neuesmean0, neuesmean1}
    ];
   FixedPoint[schrittergb, {altesmean0, altesmean1}]
  ];

datenrgb = {{{1, 2, 3}, {2, 3, 4}, {4, 5, 6}}, {{3, 2, 2}, {4, 3, 2}, {6, 5, 4}}};
MyISODATARGB[datenrgb]
```

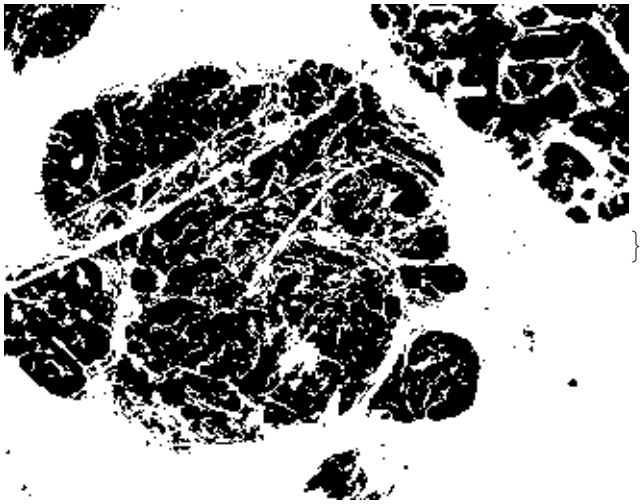
```
{{N.A., N.A.}, {{3.23333, 3.23333, 3.4}, {3.43333, 3.43333, 3.6}}}
{{4, 2}, {{2.5, 2.5, 2.75}, {5., 5., 5.}}}
{{4, 2}, {{2.5, 2.5, 2.75}, {5., 5., 5.}}}
{{2.5, 2.5, 2.75}, {5., 5., 5.}}
```

```
{Show[#1],
   Show[{Graphics3D[
         ({RGBColor[#], PointSize[.0075], Point[#]}) & /@ Flatten[ImageData[#1], 1][[ ;; ;; 50]],
         PlotRange → {{0, 1}, {0, 1}, {0, 1}}, AxesLabel → {Text[Style["R", Red, Italic]],
           Text[Style["G", Green, Italic]], Text[Style["B", Blue, Italic]]},
         Axes → True, RotationAction → "Clip", ImageSize → 384], Graphics3D[
         ({RGBColor[#], PointSize[.05], Point[#]}) & /@ (#2 / 255)], MyPlane[#2 / 255]}] &[#1, #2],
   Show[Image[MyBinarizeRGB[ImageData[#1, "Byte"], #2]]]} &[
 Sequence @@ ({#, MyISODATARGB[ImageData[#, "Real"] * 255]}) &[
  ImageResize[p16bild, Scaled[1 / 4]]]]
```

```
{{N.A., N.A.}, {{109.698, 102.993, 124.499}, {109.898, 103.193, 124.699}}}
{{40 577, 43 273}, {{76.0578, 59.3715, 75.2426}, {141.436, 144.09, 170.881}}}
{{38 811, 45 039}, {{75.3641, 57.5283, 72.2534}, {139.47, 142.357, 169.707}}}
{{37 636, 46 214}, {{74.8271, 56.2599, 70.3016}, {138.278, 141.233, 168.818}}}
{{36 793, 47 057}, {{74.4579, 55.3435, 68.8555}, {137.43, 140.427, 168.184}}}
{{36 226, 47 624}, {{74.2086, 54.7166, 67.8719}, {136.87, 139.891, 167.75}}}
{{35 832, 48 018}, {{74.0407, 54.2815, 67.1751}, {136.481, 139.517, 167.45}}}
{{35 557, 48 293}, {{73.92, 53.972, 66.6907}, {136.214, 139.259, 167.236}}}
{{35 371, 48 479}, {{73.8376, 53.7625, 66.3614}, {136.035, 139.085, 167.09}}}
{{35 263, 48 587}, {{73.7958, 53.6416, 66.1651}, {135.927, 138.983, 167.009}}}
{{35 197, 48 653}, {{73.7729, 53.5688, 66.0424}, {135.86, 138.92, 166.961}}}
{{35 142, 48 708}, {{73.7519, 53.5071, 65.9419}, {135.805, 138.868, 166.92}}}
{{35 111, 48 739}, {{73.7398, 53.4732, 65.8847}, {135.774, 138.838, 166.897}}}
{{35 094, 48 756}, {{73.7329, 53.4536, 65.8543}, {135.757, 138.823, 166.883}}}
{{35 079, 48 771}, {{73.7266, 53.4365, 65.8275}, {135.743, 138.809, 166.871}}}
{{35 069, 48 781}, {{73.7217, 53.4252, 65.8099}, {135.734, 138.799, 166.863}}}
{{35 057, 48 793}, {{73.7172, 53.4116, 65.788}, {135.721, 138.788, 166.854}}}
{{35 049, 48 801}, {{73.7135, 53.4021, 65.7742}, {135.714, 138.781, 166.848}}}
{{35 039, 48 811}, {{73.7123, 53.3914, 65.7538}, {135.702, 138.771, 166.842}}}
{{35 036, 48 814}, {{73.7115, 53.388, 65.7481}, {135.699, 138.768, 166.839}}}
{{35 034, 48 816}, {{73.7103, 53.3857, 65.7448}, {135.697, 138.766, 166.838}}}
{{35 034, 48 816}, {{73.7103, 53.3857, 65.7448}, {135.697, 138.766, 166.838}}}
```

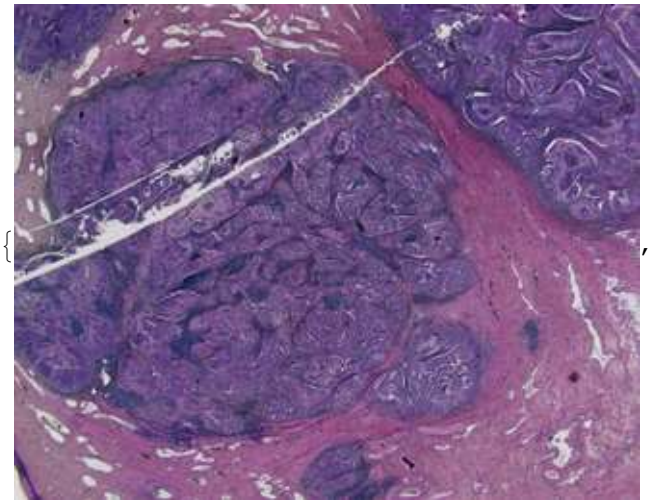
{  ,

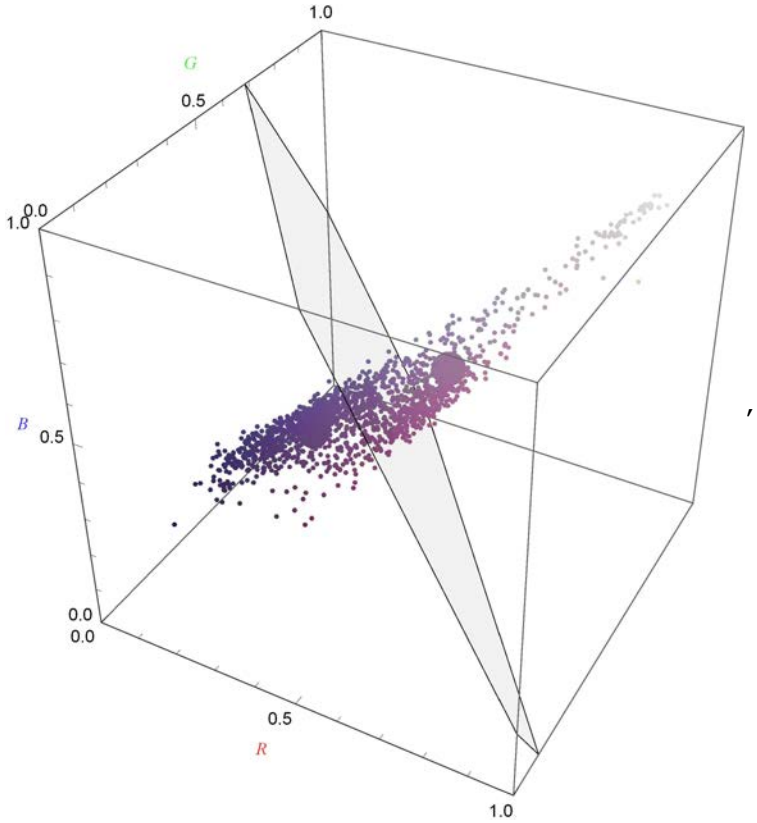

1.0
G
0.5
0.0
1.0
B
0.5
0.0
0.0
0.5
R
1.0
,


}

```
{Show[#1],
   Show[{Graphics3D[
        ({RGBColor[#], PointSize[.0075], Point[#]}) & /@ Flatten[ImageData[#1], 1][[ ;; ;; 50]],
        PlotRange → {{0, 1}, {0, 1}, {0, 1}}, AxesLabel → {Text[Style["R", Red, Italic]],
          Text[Style["G", Green, Italic]], Text[Style["B", Blue, Italic]]},
        Axes → True, RotationAction → "Clip", ImageSize → 384], Graphics3D[
        ({RGBColor[#], PointSize[.05], Point[#]}) & /@ (#2 / 255)], MyPlane[#2 / 255]}] &[#1, #2],
   Show[Image[MyBinarizeRGB[ImageData[#1, "Byte"], #2]]]} &[
 Sequence @@ ({#, MyISODATARGB[ImageData[#, "Real"] * 255]}) &[ImageResize[hebild, Scaled[1 / 4]]]]
```

```
{{N.A., N.A.}, {{119.681, 89.0555, 133.941}, {119.881, 89.2555, 134.141}}}
{{46 423, 37 427}, {{95.7714, 70.063, 120.887}, {149.561, 112.837, 150.357}}}
{{49 157, 34 693}, {{96.9991, 71.1482, 121.938}, {152.061, 114.67, 151.191}}}
{{50 778, 33 072}, {{97.9088, 71.732, 122.405}, {153.363, 115.907, 151.907}}}
{{51 790, 32 060}, {{98.499, 72.0893, 122.674}, {154.16, 116.724, 152.403}}}
{{52 444, 31 406}, {{98.8869, 72.3137, 122.848}, {154.671, 117.279, 152.732}}}
{{52 836, 31 014}, {{99.1204, 72.4475, 122.952}, {154.978, 117.62, 152.932}}}
{{53 073, 30 777}, {{99.2629, 72.5285, 123.014}, {155.163, 117.828, 153.057}}}
{{53 224, 30 626}, {{99.3537, 72.5798, 123.054}, {155.281, 117.962, 153.136}}}
{{53 293, 30 557}, {{99.3943, 72.6043, 123.072}, {155.336, 118.022, 153.172}}}
{{53 337, 30 513}, {{99.4212, 72.62, 123.082}, {155.37, 118.06, 153.198}}}
{{53 371, 30 479}, {{99.441, 72.6321, 123.091}, {155.398, 118.089, 153.215}}}
{{53 393, 30 457}, {{99.4532, 72.6405, 123.098}, {155.417, 118.107, 153.225}}}
{{53 404, 30 446}, {{99.4601, 72.6441, 123.1}, {155.425, 118.117, 153.232}}}
{{53 409, 30 441}, {{99.4626, 72.6461, 123.102}, {155.429, 118.121, 153.234}}}
{{53 414, 30 436}, {{99.4653, 72.648, 123.104}, {155.434, 118.125, 153.236}}}
{{53 414, 30 436}, {{99.4653, 72.648, 123.104}, {155.434, 118.125, 153.236}}}
```

{  ,

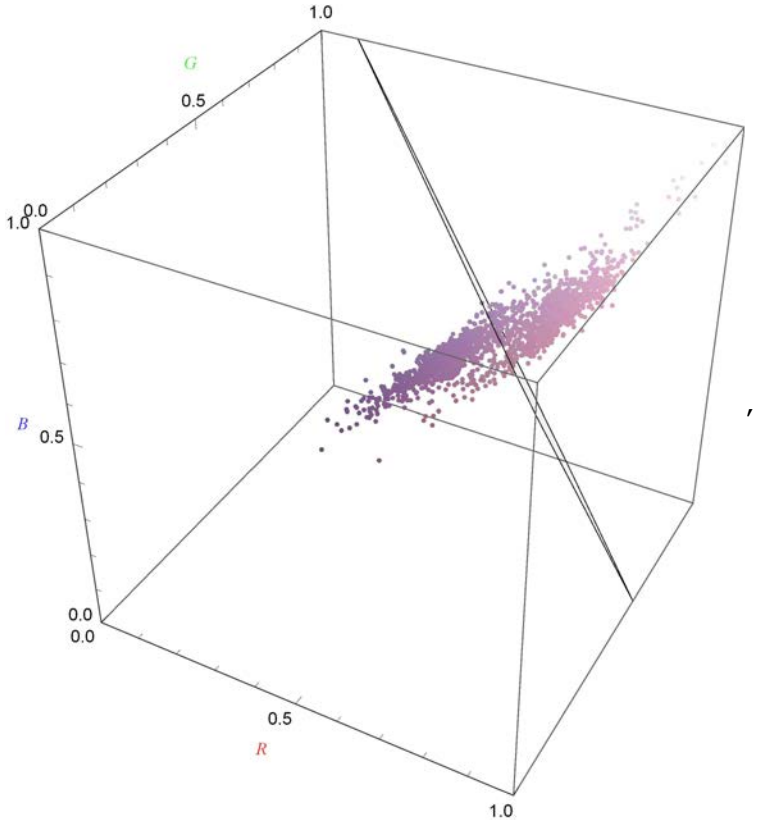

1.0
G
0.5
0.0
1.0
B
0.5
0.0
0.0
0.5
R
1.0
,

 }

```
{Show[#1],
   Show[{Graphics3D[
         ({RGBColor[#], PointSize[.0075], Point[#]}) & /@ Flatten[ImageData[#1], 1][[ ;; ;; 50]],
         PlotRange → {{0, 1}, {0, 1}, {0, 1}}, AxesLabel → {Text[Style["R", Red, Italic]],
           Text[Style["G", Green, Italic]], Text[Style["B", Blue, Italic]]},
         Axes → True, RotationAction → "Clip", ImageSize → 384], Graphics3D[
         ({RGBColor[#], PointSize[.05], Point[#]}) & /@ (#2 / 255)], MyPlane[#2 / 255]}] &[#1, #2],
   Show[Image[MyBinarizeRGB[ImageData[#1, "Byte"], #2]]]} &[
 Sequence @@ ({#, MyISODATARGB[ImageData[#, "Real"] * 255]}) &[
  ImageResize[zervixhe1, Scaled[1 / 4]]]]
```

```
{{N.A., N.A.}, {{176.889, 136.507, 170.256}, {177.089, 136.707, 170.456}}}
{{41 314, 42 536}, {{154.38, 116.837, 157.072}, {198.949, 155.808, 183.258}}}
{{41 031, 42 819}, {{154.067, 116.698, 157.192}, {198.954, 155.684, 182.97}}}
{{40 892, 42 958}, {{153.982, 116.631, 157.162}, {198.889, 155.621, 182.914}}}
{{40 826, 43 024}, {{153.945, 116.599, 157.144}, {198.856, 155.592, 182.892}}}
{{40 800, 43 050}, {{153.93, 116.587, 157.137}, {198.843, 155.58, 182.883}}}
{{40 791, 43 059}, {{153.926, 116.582, 157.134}, {198.838, 155.577, 182.881}}}
{{40 784, 43 066}, {{153.922, 116.578, 157.131}, {198.834, 155.574, 182.879}}}
{{40 780, 43 070}, {{153.92, 116.576, 157.13}, {198.831, 155.572, 182.878}}}
{{40 778, 43 072}, {{153.919, 116.575, 157.13}, {198.83, 155.571, 182.877}}}
{{40 775, 43 075}, {{153.918, 116.573, 157.128}, {198.829, 155.57, 182.876}}}
{{40 773, 43 077}, {{153.917, 116.572, 157.128}, {198.827, 155.569, 182.876}}}
{{40 773, 43 077}, {{153.917, 116.572, 157.128}, {198.827, 155.569, 182.876}}}
```

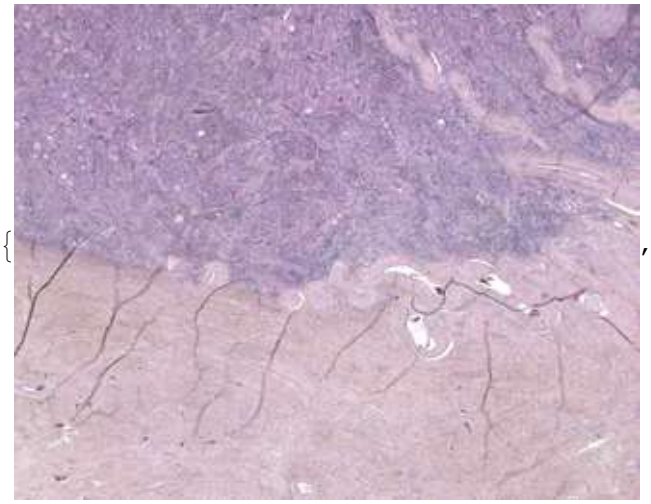
{  ,

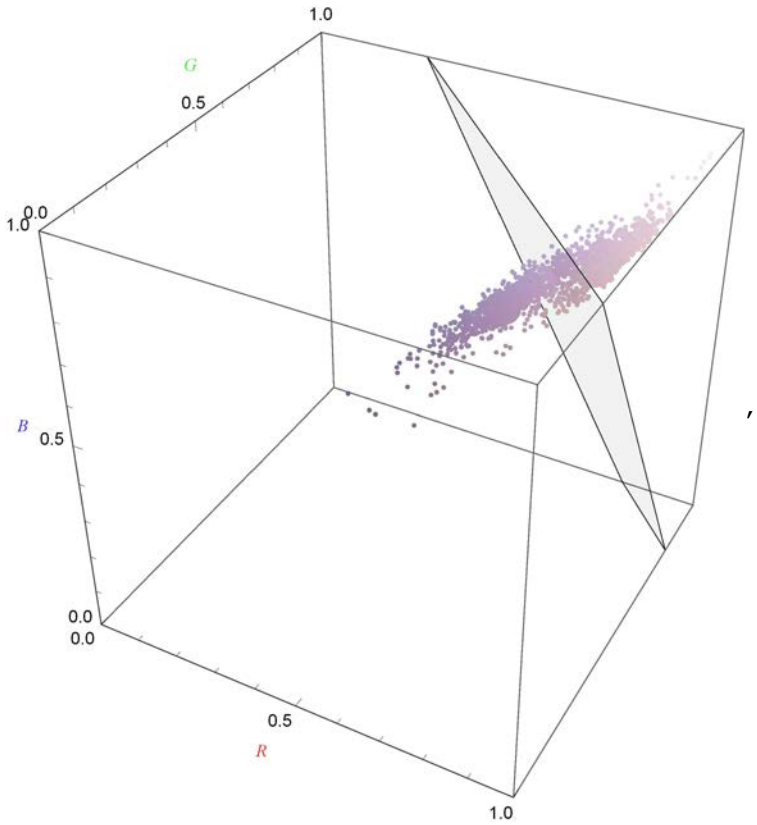

1.0
G
0.5
0.0
1.0
B
0.5
0.0
0.0
0.5
R
1.0
,

 }

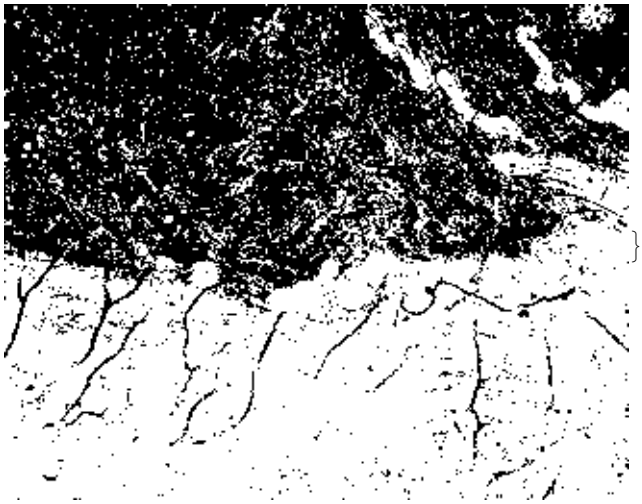
```
{Show[#1],
   Show[{Graphics3D[
         ({RGBColor[#], PointSize[.0075], Point[#]}) & /@ Flatten[ImageData[#1], 1][[ ;; ;; 50]],
         PlotRange → {{0, 1}, {0, 1}, {0, 1}}, AxesLabel → {Text[Style["R", Red, Italic]],
           Text[Style["G", Green, Italic]], Text[Style["B", Blue, Italic]]},
         Axes → True, RotationAction → "Clip", ImageSize → 384], Graphics3D[
         ({RGBColor[#], PointSize[.05], Point[#]}) & /@ (#2 / 255)], MyPlane[#2 / 255]}] &[#1, #2],
   Show[Image[MyBinarizeRGB[ImageData[#1, "Byte"], #2]]]} &[
 Sequence @@ ({#, MyISODATARGB[ImageData[#, "Real"] * 255]}) &[
  ImageResize[zervixhe2, Scaled[1 / 4]]]]
```

```
{{N.A., N.A.}, {{193.489, 169.819, 192.143}, {193.689, 170.019, 192.343}}}
{{38 938, 44 912}, {{172.819, 149.225, 180.58}, {211.597, 187.86, 202.354}}}
{{37 834, 46 016}, {{172.103, 148.559, 180.526}, {211.254, 187.481, 201.877}}}
{{37 272, 46 578}, {{171.801, 148.271, 180.352}, {211.025, 187.242, 201.758}}}
{{37 051, 46 799}, {{171.679, 148.16, 180.284}, {210.935, 187.146, 201.711}}}
{{36 968, 46 882}, {{171.637, 148.117, 180.255}, {210.899, 187.111, 201.696}}}
{{36 924, 46 926}, {{171.613, 148.095, 180.24}, {210.881, 187.091, 201.688}}}
{{36 889, 46 961}, {{171.596, 148.079, 180.223}, {210.866, 187.075, 201.685}}}
{{36 878, 46 972}, {{171.589, 148.073, 180.221}, {210.861, 187.07, 201.681}}}
{{36 873, 46 977}, {{171.586, 148.07, 180.221}, {210.86, 187.068, 201.679}}}
{{36 873, 46 977}, {{171.586, 148.07, 180.221}, {210.86, 187.068, 201.679}}}
```

{  ,



,

 }

20. Segmentierung mit dem k-Means-Verfahren

k-Means Segmentation Algorithm (Steinhaus 1957, MacQueen 1967)

→ wenn mehr als nur Binarisierung interessiert

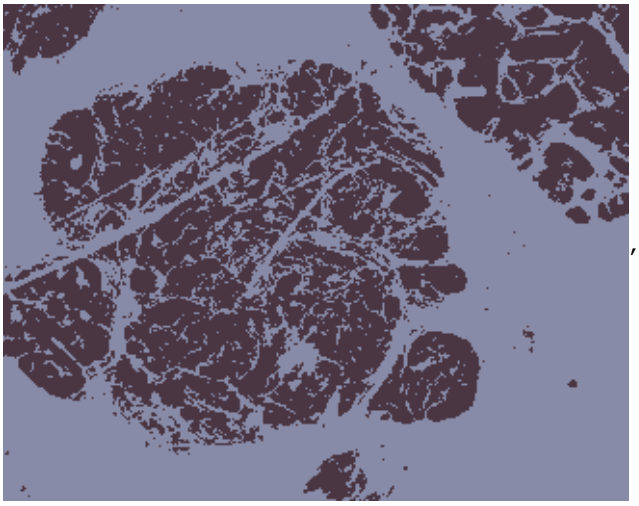
1. k Clusterzentren werden zufällig verteilt.
2. Jeder lokale Merkmalsvektor wird dem nächstliegenden Cluster zugeordnet
3. Für jedes Cluster wird der Schwerpunkt neu berechnet
4. Basierend auf den neu berechneten Zentren werden die Merkmalsvektoren wieder wie in Schritt 2 den Clustern zugeordnet
5. Iteration von 2 bis 4 wird wiederholt, bis eine festgelegte maximale Iterationszahl erreicht wurde oder sich die Schwerpunkte nicht mehr bewegen, d. h. kein Merkmalsvektor mehr einem anderen Cluster zugeordnet wird

```
SeedRandom[2405];
(*Generierung von multimodaler Verteilung 2D*)
data = Join[RandomReal[NormalDistribution[0, .4], {500, 2}],
    RandomReal[NormalDistribution[1, .4], {500, 2}],
    Transpose[{RandomReal[NormalDistribution[1.2, .3], 500],
      RandomReal[NormalDistribution[0, .3], 500]}]];
(*Vorgabe: 3 Klassen, und Einsatz auf Level 1 (vektorielle Daten)*)
cs = ClusteringComponents[data, 3, 1, Method → "KMeans"];
km = Mean[Map[data[[Sequence @@ #]] &, #]] & /@ (Position[cs, #] & /@ Union[Flatten[cs]]);
Show[{ListPlot[data, AspectRatio → 1, PlotRange → {{-1, 2}, {-1, 2}}],
  ListPlot[km, PlotStyle -> Directive[Red, PointSize[Large]]], DiagramPlot[km],
  MyLine[#] & /@ (km[[#]] & /@ Flatten[Table[{i, j}, {i, Length[km] - 1}, {j, i + 1, Length[km]}], 1])
 }]
```

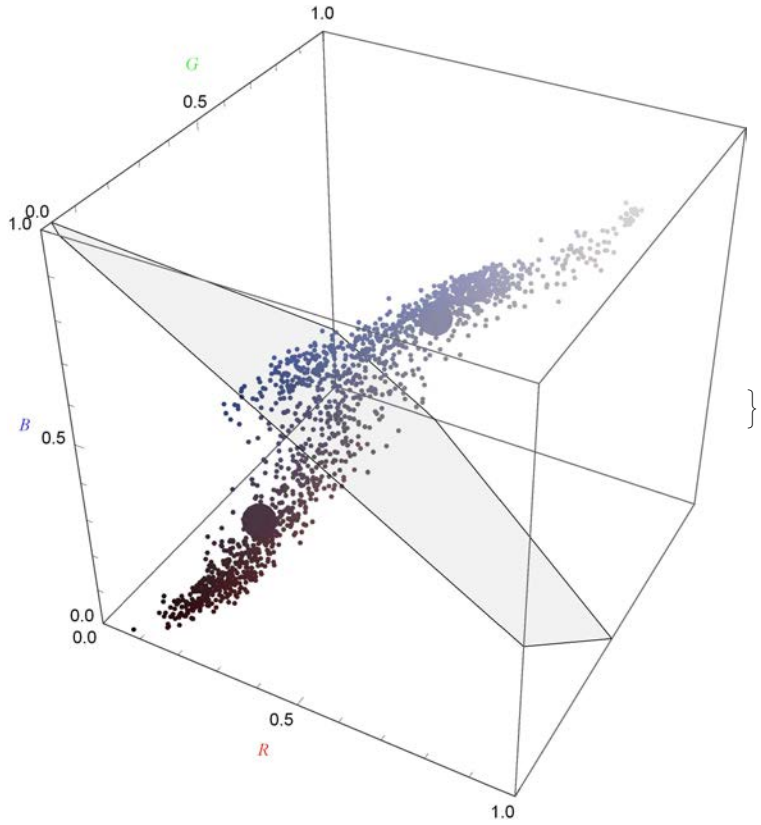
```
Clear[kmeans];
kmeans[bild_Image, klassen_Integer, ebenen_: True] :=
 Module[{c, i, m},
  c = ClusteringComponents[bild, klassen, Method → "KMeans"];
  i = ImageData[bild];
  m = Mean[Map[i[[Sequence @@ #]] &, #]] & /@ (Position[c, #] & /@ Union[Flatten[c]]);
  If[ebenen == True,
   Print[{MatrixForm[m], Show[Colorize[c,
         ColorFunction → (RGBColor[Sequence @@ m[[#]]] &), ColorFunctionScaling → False]],
       Show[{Graphics3D[({RGBColor[#], PointSize[.0075], Point[#]}) & /@
            Flatten[ImageData[bild], 1][[ ;; ;; 50]], PlotRange → {{0, 1}, {0, 1}, {0, 1}},
          AxesLabel → {Text[Style["R", Red, Italic]], Text[Style["G", Green, Italic]], Text[
             Style["B", Blue, Italic]]}, Axes → True, RotationAction → "Clip", ImageSize → 384],
         Graphics3D[({RGBColor[#], PointSize[.05], Point[#]}) & /@m], MyPlane[#] & /@
          (m[[#]] & /@ Flatten[Table[{i, j}, {i, Length[m] - 1}, {j, i + 1, Length[m]}], 1])}]}];
   ,
   Print[{MatrixForm[m], Show[Colorize[c,
         ColorFunction → (RGBColor[Sequence @@ m[[#]]] &), ColorFunctionScaling → False]],
       Show[{Graphics3D[({RGBColor[#], PointSize[.0075], Point[#]}) & /@
            Flatten[ImageData[bild], 1][[ ;; ;; 50]], PlotRange → {{0, 1}, {0, 1}, {0, 1}},
          AxesLabel → {Text[Style["R", Red, Italic]], Text[Style["G", Green, Italic]],
            Text[Style["B", Blue, Italic]]}, Axes → True, RotationAction → "Clip",
          ImageSize → 384], Graphics3D[({RGBColor[#], PointSize[.05], Point[#]}) & /@m]}]}];
  ];
  m
 ]
```

```
(*hier mit k=2 zum Vergleich mit ISODATA (s.o.)*)
kmeans[ImageResize[p16bild, Scaled[1 / 4]], 2]
```

$$\left\{\begin{pmatrix} 0.28906 & 0.209356 & 0.257823 \\ 0.532146 & 0.544182 & 0.654265 \end{pmatrix},\right.$$



,



$\left.\right\}$

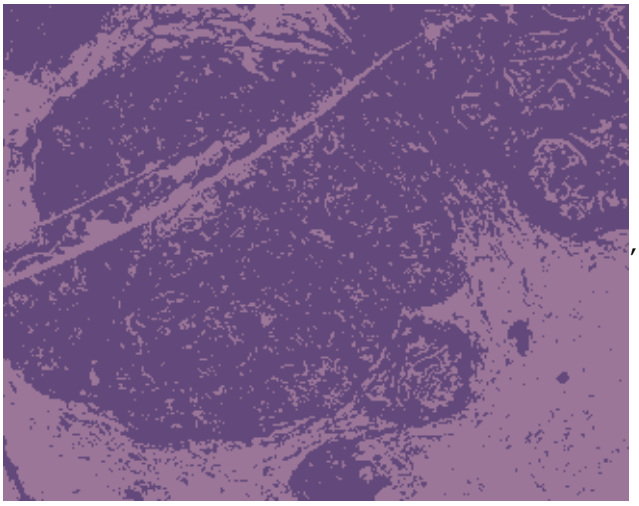
```
{{0.28906, 0.209356, 0.257823}, {0.532146, 0.544182, 0.654265}}
```

```
(*Hier zur Gegenüberstellung die eigene Implementierung zur Binarisierung*)
MyISODATARGB[ImageData[ImageResize[p16bild, Scaled[1 / 4]], "Real"]]
```

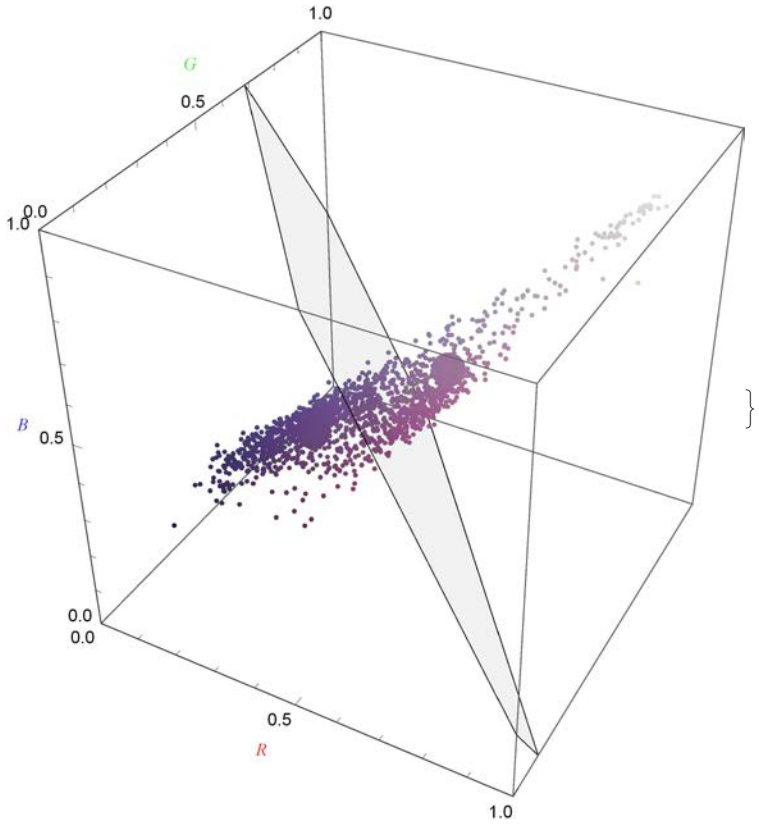
```
{{N.A., N.A.}, {{0.33058, 0.304286, 0.388625}, {0.53058, 0.504286, 0.588625}}}
{{40 577, 43 273}, {{0.298266, 0.23283, 0.295069}, {0.554652, 0.56506, 0.670121}}}
{{38 811, 45 039}, {{0.295545, 0.225601, 0.283347}, {0.546943, 0.558262, 0.665517}}}
{{37 636, 46 214}, {{0.29344, 0.220627, 0.275693}, {0.542266, 0.553854, 0.662033}}}
{{36 793, 47 057}, {{0.291992, 0.217033, 0.270021}, {0.538941, 0.550695, 0.659546}}}
{{36 226, 47 624}, {{0.291014, 0.214575, 0.266164}, {0.536744, 0.548592, 0.657843}}}
{{35 832, 48 018}, {{0.290356, 0.212868, 0.263432}, {0.535219, 0.547125, 0.656668}}}
{{35 557, 48 293}, {{0.289882, 0.211655, 0.261532}, {0.534173, 0.546115, 0.655827}}}
{{35 371, 48 479}, {{0.289559, 0.210833, 0.260241}, {0.533472, 0.545431, 0.655257}}}
{{35 263, 48 587}, {{0.289395, 0.210359, 0.259471}, {0.533049, 0.545032, 0.654937}}}
{{35 197, 48 653}, {{0.289305, 0.210074, 0.25899}, {0.532783, 0.544784, 0.654749}}}
{{35 142, 48 708}, {{0.289223, 0.209832, 0.258596}, {0.532567, 0.544581, 0.654587}}}
{{35 111, 48 739}, {{0.289176, 0.209699, 0.258372}, {0.532447, 0.544464, 0.654496}}}
{{35 094, 48 756}, {{0.289148, 0.209622, 0.258252}, {0.532382, 0.544402, 0.654444}}}
{{35 079, 48 771}, {{0.289124, 0.209555, 0.258147}, {0.532325, 0.544348, 0.654398}}}
{{35 069, 48 781}, {{0.289105, 0.209511, 0.258078}, {0.532288, 0.544311, 0.654366}}}
{{35 057, 48 793}, {{0.289087, 0.209457, 0.257992}, {0.532241, 0.544267, 0.65433}}}
{{35 049, 48 801}, {{0.289072, 0.20942, 0.257938}, {0.532212, 0.544239, 0.654304}}}
{{35 039, 48 811}, {{0.289068, 0.209378, 0.257858}, {0.532165, 0.5442, 0.654281}}}
{{35 036, 48 814}, {{0.289065, 0.209365, 0.257836}, {0.532153, 0.544189, 0.654272}}}
{{35 034, 48 816}, {{0.28906, 0.209356, 0.257823}, {0.532146, 0.544182, 0.654265}}}
{{35 034, 48 816}, {{0.28906, 0.209356, 0.257823}, {0.532146, 0.544182, 0.654265}}}
{{0.28906, 0.209356, 0.257823}, {0.532146, 0.544182, 0.654265}}
```

```
(*hier mit k=2 zum Vergleich mit ISODATA (s.o.)*)
kmeans[ImageResize[hebild, Scaled[1 / 4]], 2];
```

$$\left\{\begin{pmatrix} 0.39006 & 0.284894 & 0.48276 \\ 0.609545 & 0.463237 & 0.600924 \end{pmatrix},\right.$$



,

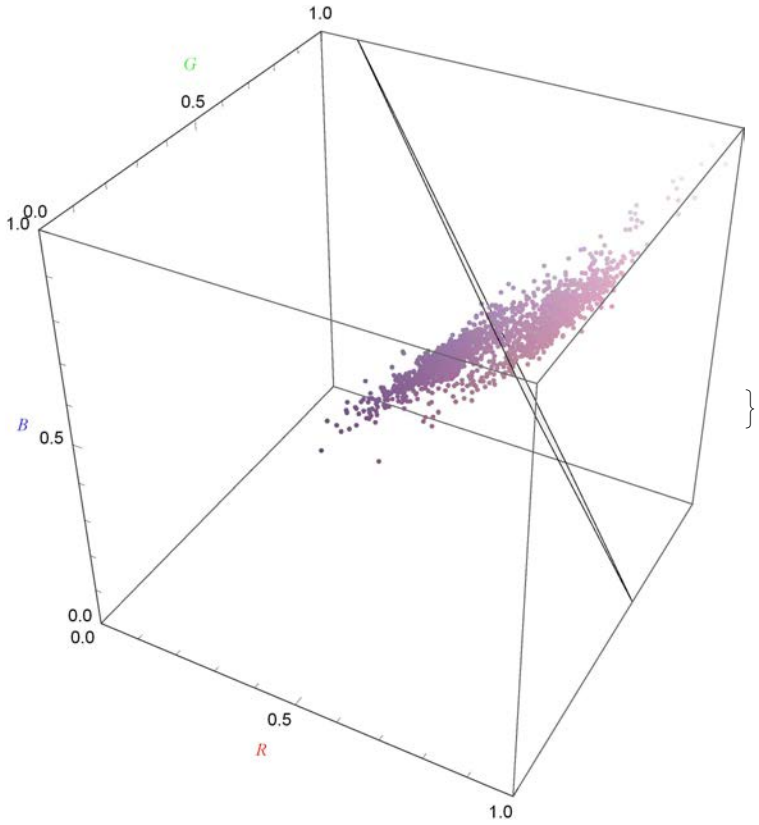


$\left.\right\}$

```
(*hier mit k=2 zum Vergleich mit ISODATA (s.o.)*)
kmeans[ImageResize[zervixhe1, Scaled[1/4]], 2];
```

$$\left\{\begin{pmatrix} 0.603578 & 0.457136 & 0.616185 \\ 0.779707 & 0.610064 & 0.717149 \end{pmatrix},\right.$$

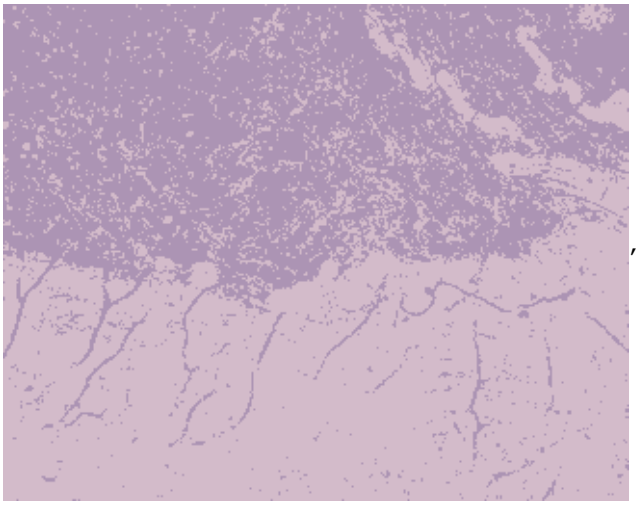
,



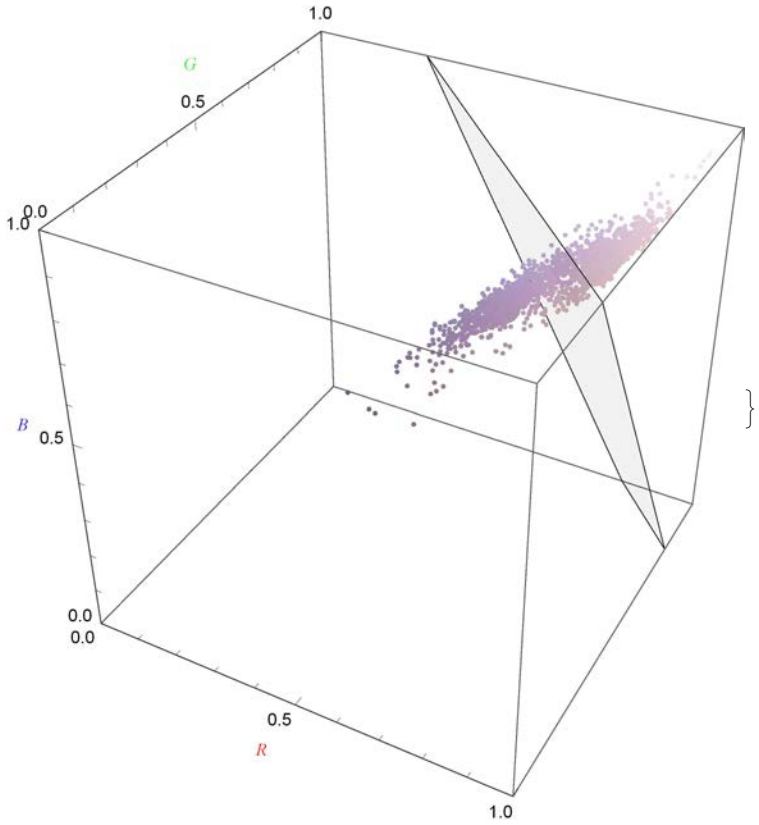
$\left.\right\}$

```
(*hier mit k=2 zum Vergleich mit ISODATA (s.o.)*)
kmeans[ImageResize[zervixhe2, Scaled[1/4]], 2];
```

$$\left\{\begin{pmatrix} 0.672878 & 0.580662 & 0.706742 \\ 0.826895 & 0.733594 & 0.790898 \end{pmatrix},\right.$$



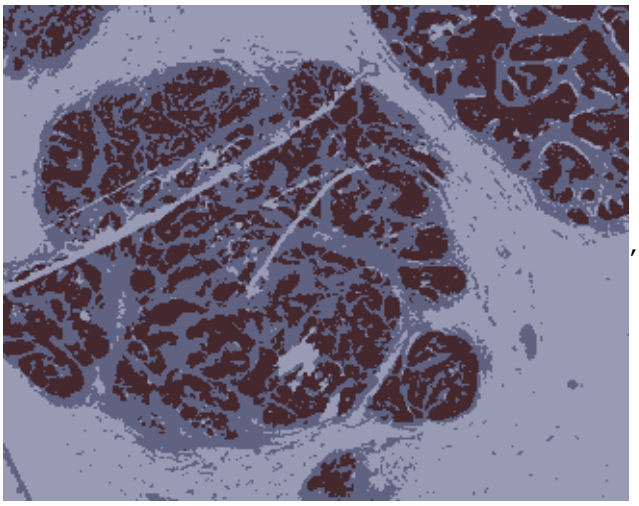
,



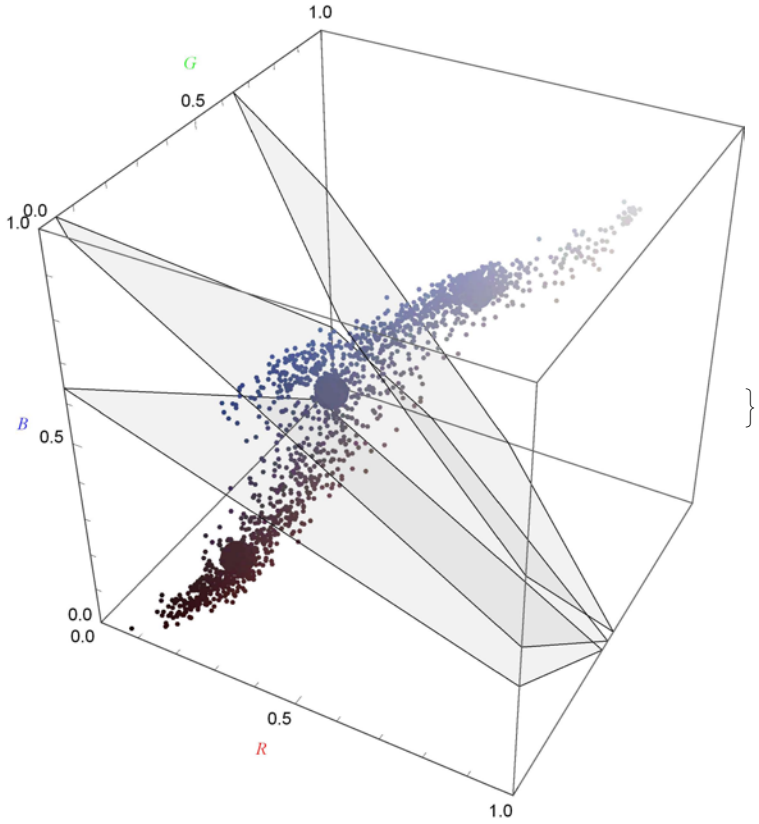
$\left.\right\}$

```
kmeans[ImageResize[p16bild, Scaled[1/4]], 3];
```

$$\left\{\begin{pmatrix} 0.376713 & 0.384722 & 0.509935 \\ 0.267305 & 0.157901 & 0.179791 \\ 0.598325 & 0.605496 & 0.702282 \end{pmatrix},\right.$$



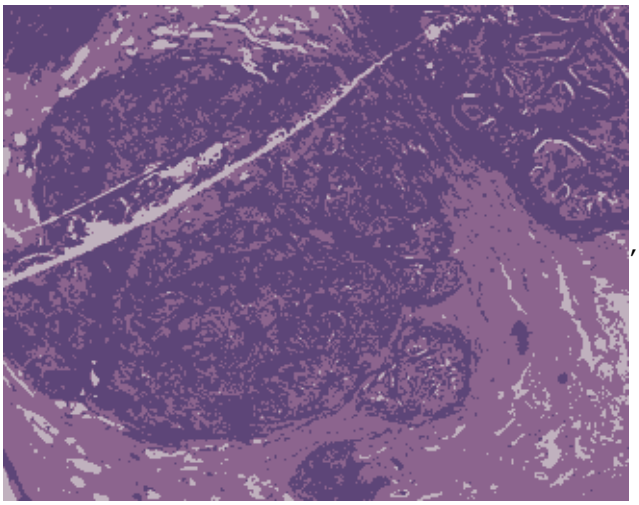
,



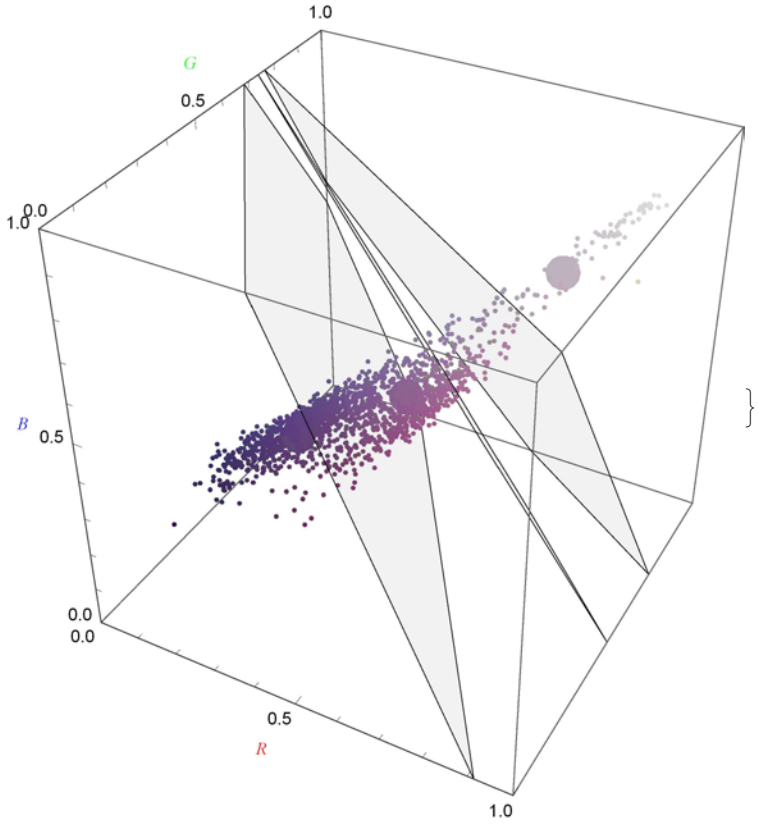
$\left.\right\}$

```
kmeans[ImageResize[hebild, Scaled[1 / 4]], 3];
```

$$\left\{\begin{pmatrix} 0.364858 & 0.269907 & 0.470856 \\ 0.550922 & 0.392179 & 0.557477 \\ 0.751354 & 0.694892 & 0.745313 \end{pmatrix},\right.$$

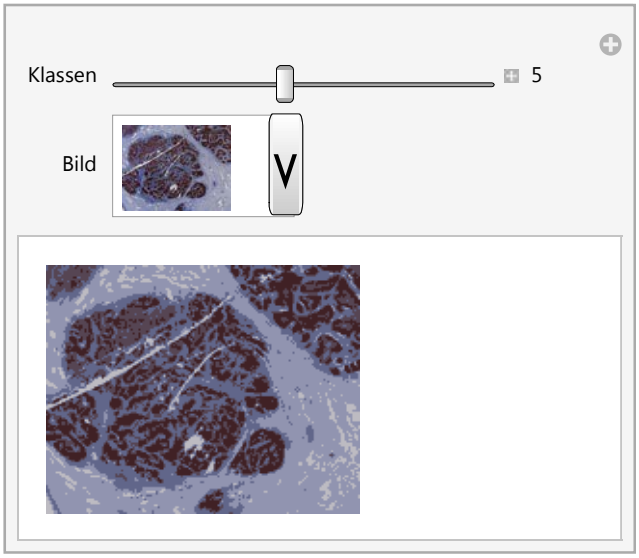


,



}

```
Manipulate[Block[{c, i, m}, ControlActive[klassen, ControlActive[bild,
    c = ClusteringComponents[bild, klassen, Method → "KMeans"];
    i = ImageData[bild];
    m = Mean[Map[i[[Sequence @@ #]] &, #]] & /@ (Position[c, #] & /@ Union[Flatten[c]]);
    Show[Colorize[c, ColorFunction → (RGBColor[Sequence @@ m[[#]]] &),
      ColorFunctionScaling → False], ImageSize → ImageDimensions[bild]]]],
 {{klassen, 4, "Klassen"}, 1, 10, 1, Appearance → "Labeled"},
 {{bild, Image@ImageData@tablettenrgb, "Bild"},
  {ImageResize[p16bild, Scaled[1 / 8]], ImageResize[hebild, Scaled[1 / 8]],
   ImageCrop[Lym3CD21dreikanalausgleichF2, {384, 384}], lenargb, Image@ImageData@tablettenrgb,
   ImageResize[ExampleData[{"TestImage", "Apples"}], Scaled[1 / 8]]}, ImageSize → Tiny,
  ControlType → PopupMenu}, ContinuousAction → False, SaveDefinitions → True]
```



21. Segmentierung durch hierarchisch aufteilende Partitionierung

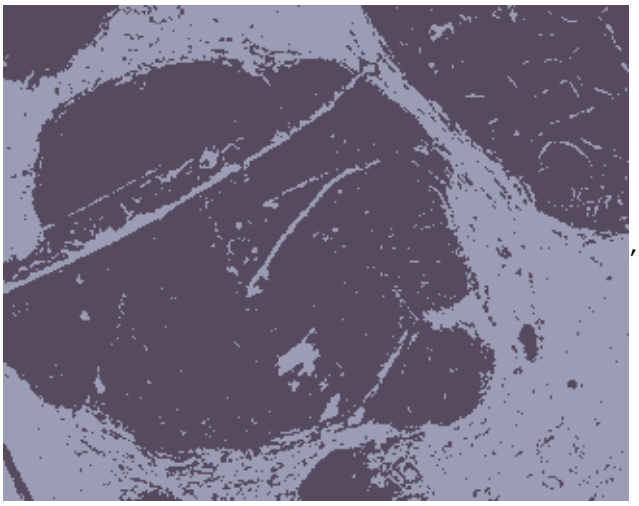
Hierarchisch aufteilende Partitionierung

1. es wird mit einem Clusterzentrum (Mittelwert) begonnen
2. es erfolgt die Wahl, welcher Cluster in zwei neue Cluster aufgeteilt wird, z.B. anhand der Varianz
3. es werden die Merkmalsvektoren nur dieser beiden Cluster neu zugeordnet (z.B. mit k-Means) und damit die neuen Cluster spezifiziert
4. Schritte 2 und 3 solange wiederholt, bis die festzulegende Höchstzahl an Clustern angelegt ist

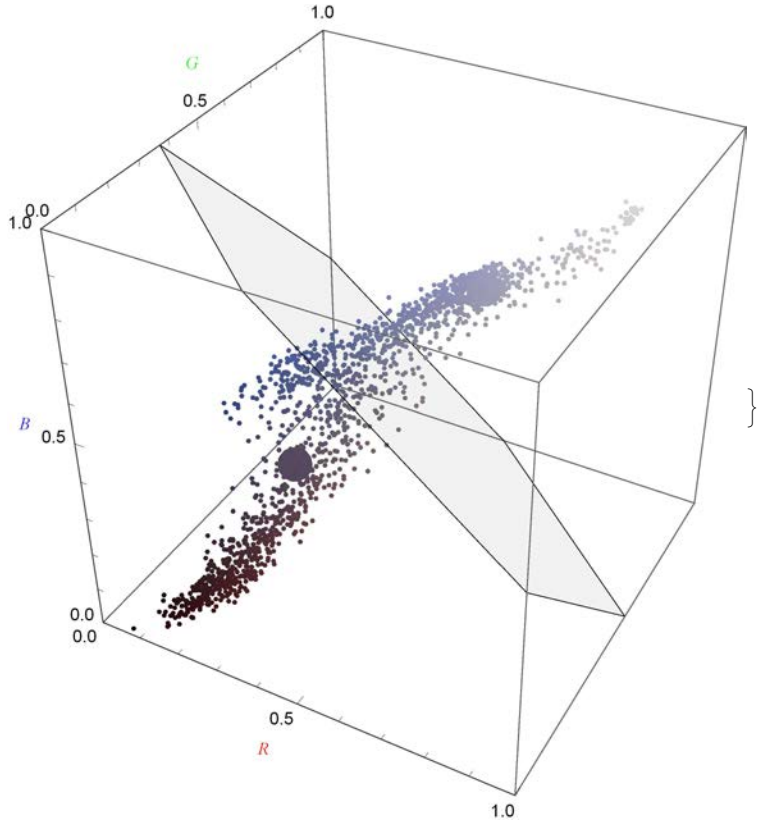
```
Clear[colquant];
colquant[bild_Image, klassen_Integer, ebenen_: True] :=
 Module[{cq, m},
  cq = ColorQuantize[bild, klassen, Dithering → False];
  m = Union[Flatten[ImageData[cq], 1]];
  If[ebenen == True,
   Print[{MatrixForm[m], Show[cq],
     Show[{Graphics3D[({RGBColor[#], PointSize[.0075], Point[#]}) & /@
         Flatten[ImageData[bild], 1][[ ;; ;; 50]], PlotRange → {{0, 1}, {0, 1}, {0, 1}},
       AxesLabel → {Text[Style["R", Red, Italic]], Text[Style["G", Green, Italic]], Text[
          Style["B", Blue, Italic]]}, Axes → True, RotationAction → "Clip", ImageSize → 384],
      Graphics3D[({RGBColor[#], PointSize[.05], Point[#]}) & /@m], MyPlane[#] & /@
       (m[[#]] & /@ Flatten[Table[{i, j}, {i, Length[m] - 1}, {j, i + 1, Length[m]}], 1])}]}];
   ,
   Print[{MatrixForm[m], Show[cq],
     Show[{Graphics3D[({RGBColor[#], PointSize[.0075], Point[#]}) & /@
         Flatten[ImageData[bild], 1][[ ;; ;; 50]], PlotRange → {{0, 1}, {0, 1}, {0, 1}},
       AxesLabel → {Text[Style["R", Red, Italic]], Text[Style["G", Green, Italic]],
         Text[Style["B", Blue, Italic]]}, Axes → True, RotationAction → "Clip",
       ImageSize → 384], Graphics3D[({RGBColor[#], PointSize[.05], Point[#]}) & /@m]}]}];
  ];
  m
 ]
```

```
colquant[ImageResize[p16bild, Scaled[1 / 4]], 2];
```

$$\left\{\begin{pmatrix} 0.333333 & 0.290196 & 0.368627 \\ 0.607843 & 0.615686 & 0.705882 \end{pmatrix},\right.$$



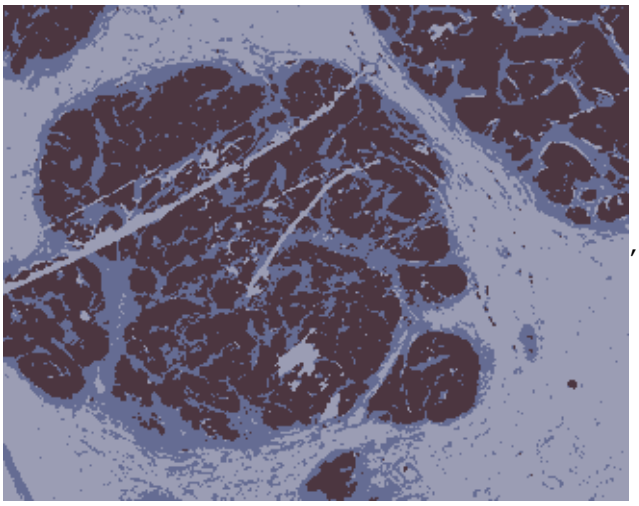
,



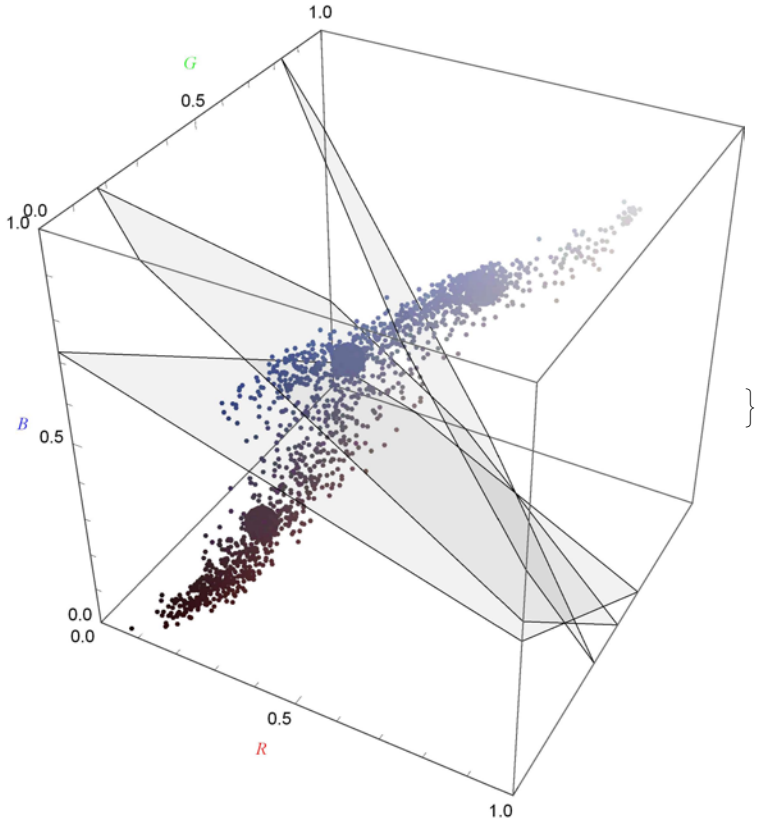
$\left.\right\}$

```
colquant[ImageResize[p16bild, Scaled[1 / 4]], 3];
```

$$\left\{\begin{pmatrix} 0.298039 & 0.211765 & 0.25098 \\ 0.396078 & 0.423529 & 0.576471 \\ 0.607843 & 0.615686 & 0.705882 \end{pmatrix},\right.$$



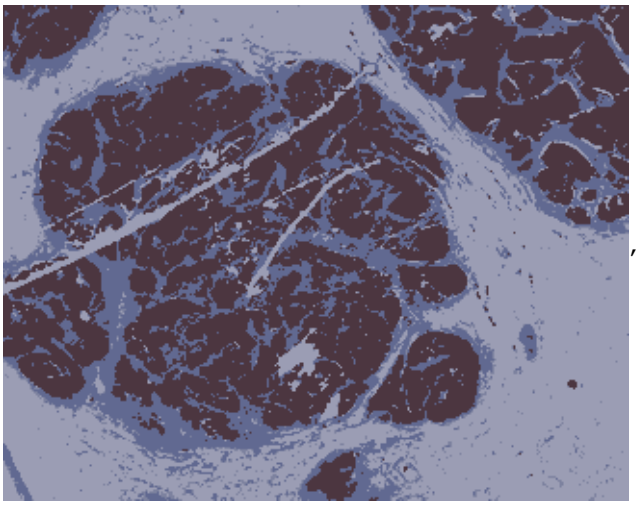
,



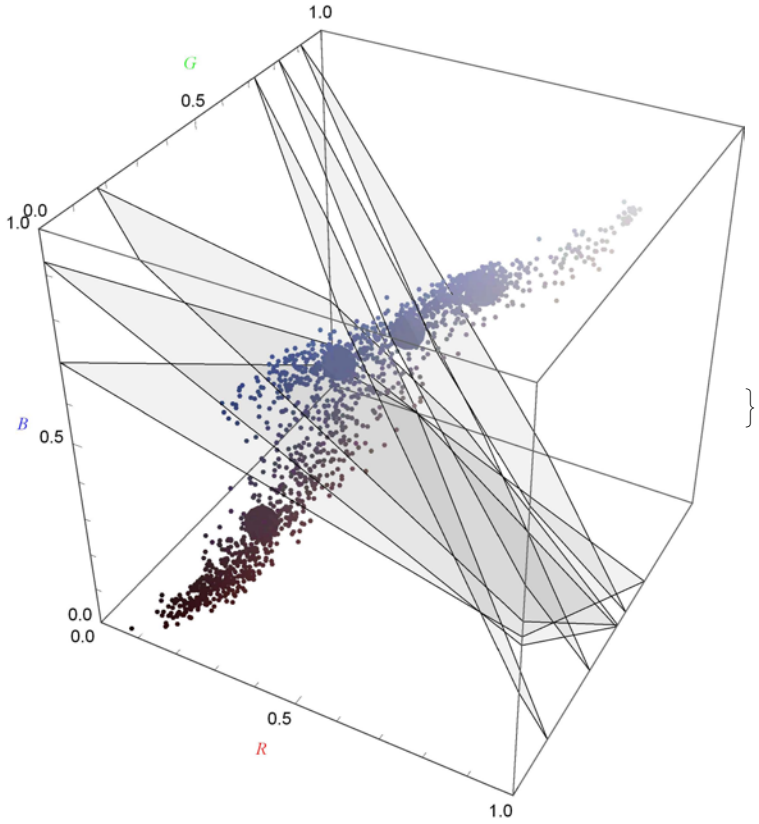
$\left.\right\}$

```
colquant[ImageResize[p16bild, Scaled[1 / 4]], 4];
```

$$\left\{\begin{pmatrix} 0.298039 & 0.211765 & 0.25098 \\ 0.380392 & 0.411765 & 0.568627 \\ 0.490196 & 0.509804 & 0.623529 \\ 0.607843 & 0.615686 & 0.705882 \end{pmatrix},\right.$$

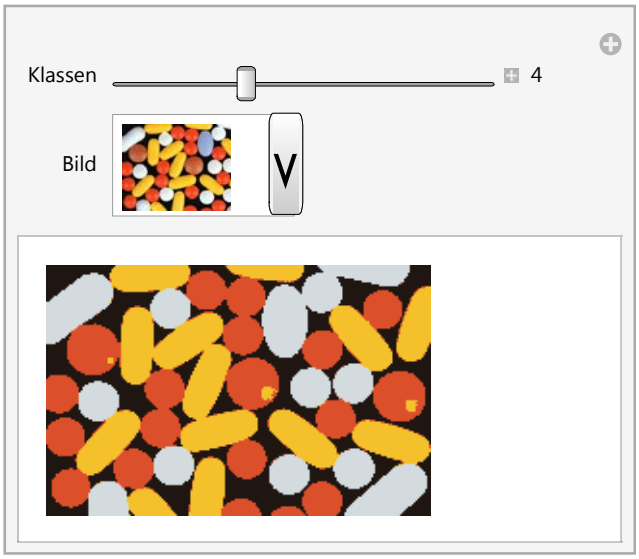


,



}

```
Manipulate[Block[{cq}, ControlActive[klassen, ControlActive[bild,
    cq = ColorQuantize[bild, klassen, Dithering → False];
    Show[cq, ImageSize → ImageDimensions[bild]]]]],
 {{klassen, 4, "Klassen"}, 1, 10, 1, Appearance → "Labeled"},
 {{bild, Image@ImageData@tablettenrgb, "Bild"},
  {ImageResize[p16bild, Scaled[1 / 8]], ImageResize[hebild, Scaled[1 / 8]],
   ImageCrop[Lym3CD21dreikanalausgleichF2, {384, 384}], lenargb, Image@ImageData@tablettenrgb,
   ImageResize[ExampleData[{"TestImage", "Apples"}], Scaled[1 / 8]]}, ImageSize → Tiny,
  ControlType → PopupMenu}, ContinuousAction → False, SaveDefinitions → True]
```



22. Segmentierung durch “Mittelwertverschiebung”

Mean Shift Segmentation Algorithm (Fukunaga und Hostetler 1975)

1. iterative Bestimmung lokaler Dichtemaxima (Suche von Moden)
2. initiale Schätzung solcher Moden ist das Bild selbst
3. Kerndichteschätzer zur Bestimmung lokaler Mittelwerte in Nachbarschaft (ausgenommen starke lokale Abweichungen)
4. Ersetzen der lokalen Werte durch die lokalen Mittelwerte
5. Wiederholung bis Mittelwerte konvergieren
6. Clusterbestimmung, z.B. indem alle Werte, die näher als bestimmten Radius zusammenliegen, zusammengefaßt werden

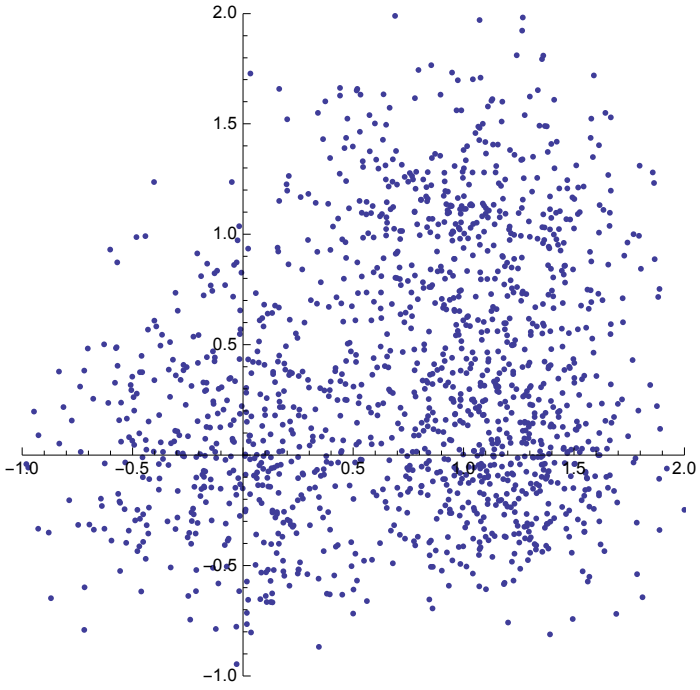
```
MeanShift[ {0, 1, 10, 11}, 1]
```

$$\left\{\frac{1}{2}, \frac{1}{2}, \frac{21}{2}, \frac{21}{2}\right\}$$

```
MeanShiftFilter[ {0, 1, 10, 11}, 1, 1.0, MaxIterations → 1]
```

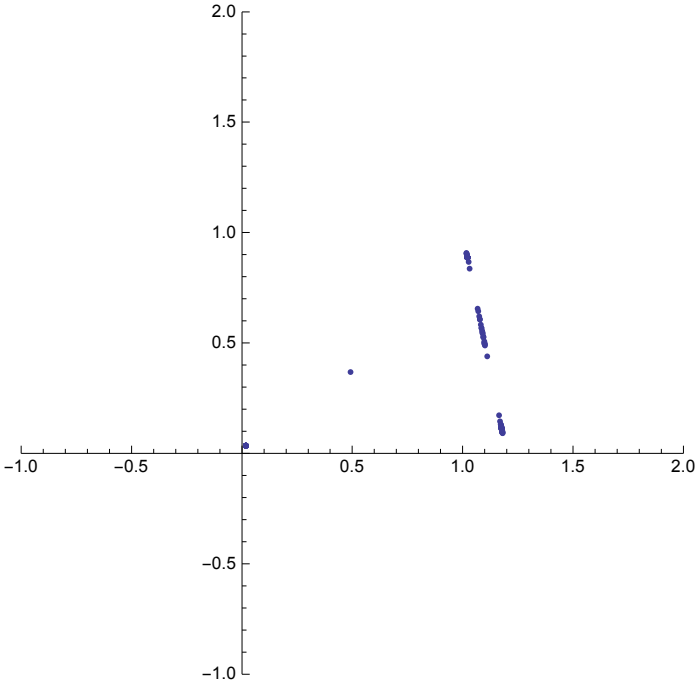
```
{0.5, 0.5, 10.5, 10.5}
```

```
(*Auswahlfunktion zur Selection nur bis zu einem bestimmten Betrag abweichender Elemente*)
Clear[MySelect];
MySelect[in_, ref_, d_] := Select[in, EuclideanDistance[#, ref] ≤ d &];

(*Eigentliche Implementierung,
es wird eine zyklisch geschlossene Bildfläche (Torus) verarbeitet*)
Clear[MyMeanShiftFilter];
MyMeanShiftFilter[im_, el_, dist_] := Module[{flatelpos, zentralpos},
    (*lineare Positionsindizes für alle Elemente
     in lokaler Nachbarschaft entsprechend des Strukturelements*)
    flatelpos = Flatten[Position[Flatten[el], 1]];
    (*Index des zentralen Elements innerhalb der Nachbarschaft*)
    zentralpos = Ceiling[Dimensions[Flatten[el]] / 2];
    (*MySelect hat drei Argumente: alle Elemente entspr. Strukturelement,
    Wert des zentralen Pixels als Bezug, maximal zulässiger Werteradius*)
    Developer`PartitionMap[
     Mean[MySelect[Flatten[#, 1][[flatelpos]], First[Flatten[#, 1][[zentralpos]]], dist]] &,
     N[im], Dimensions[el], 1, Ceiling[Dimensions[el] / 2]]
   ];

(*Test mit selben Daten wie oben*)
MyMeanShiftFilter[{0, 1, 10, 11}, {1, 1, 1}, 1]
{0.5, 0.5, 10.5, 10.5}

SeedRandom[2405];
data = Join[RandomReal[NormalDistribution[0, .4], {500, 2}],
    RandomReal[NormalDistribution[1, .4], {500, 2}],
    Transpose[{RandomReal[NormalDistribution[1.2, .3], 500],
      RandomReal[NormalDistribution[0, .3], 500]}]];

data // Dimensions
{1500, 2}

ListPlot[data, AspectRatio → 1, PlotRange → {{-1, 2}, {-1, 2}}]
```

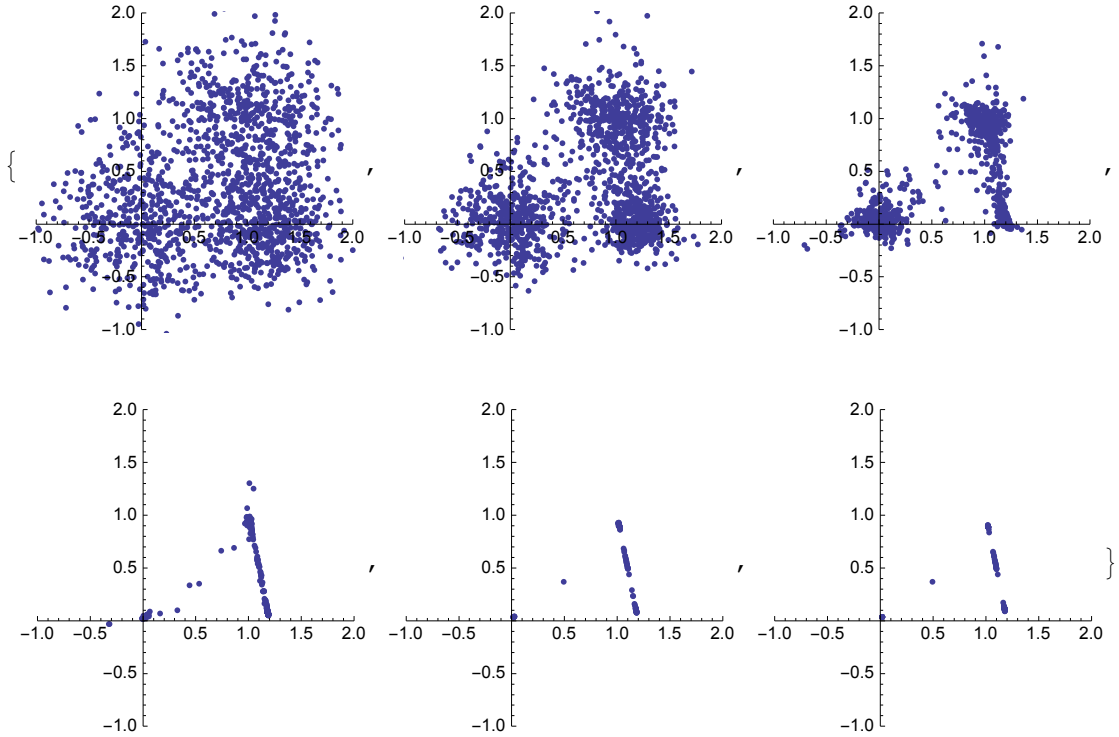



```
ms = NestList[MeanShiftFilter[#, 500, {0.55, 0.6}] &, data, 5];
```

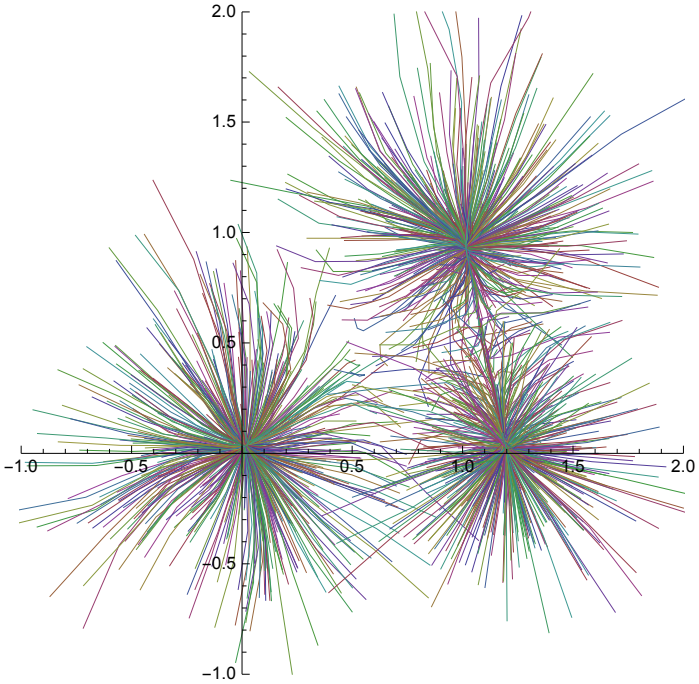
```
ListPlot[Last[ms], AspectRatio → 1, PlotRange → {{-1, 2}, {-1, 2}}]
```



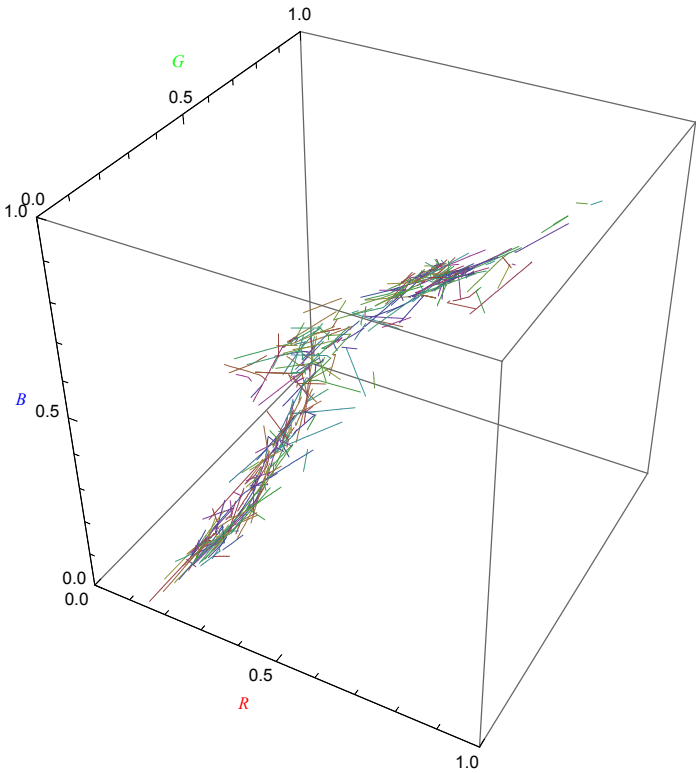

```
ListPlot[#, AspectRatio → 1, PlotRange → {{-1, 2}, {-1, 2}}] & /@ ms
```



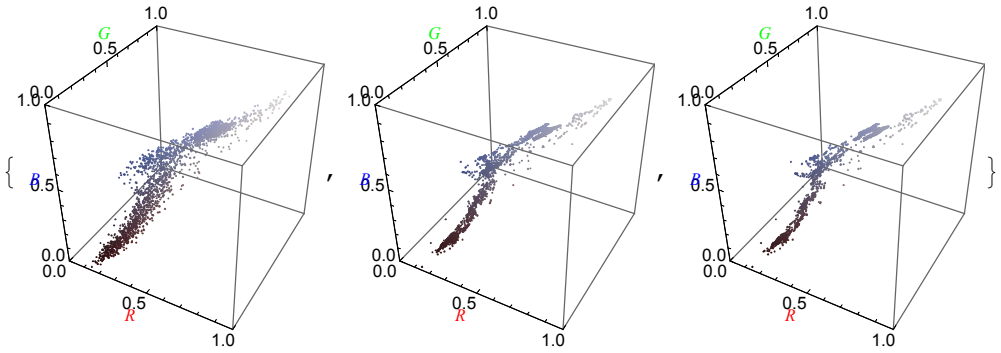
```
ListLinePlot[Transpose[ms], AspectRatio -> 1, PlotRange → {{-1, 2}, {-1, 2}}]
```



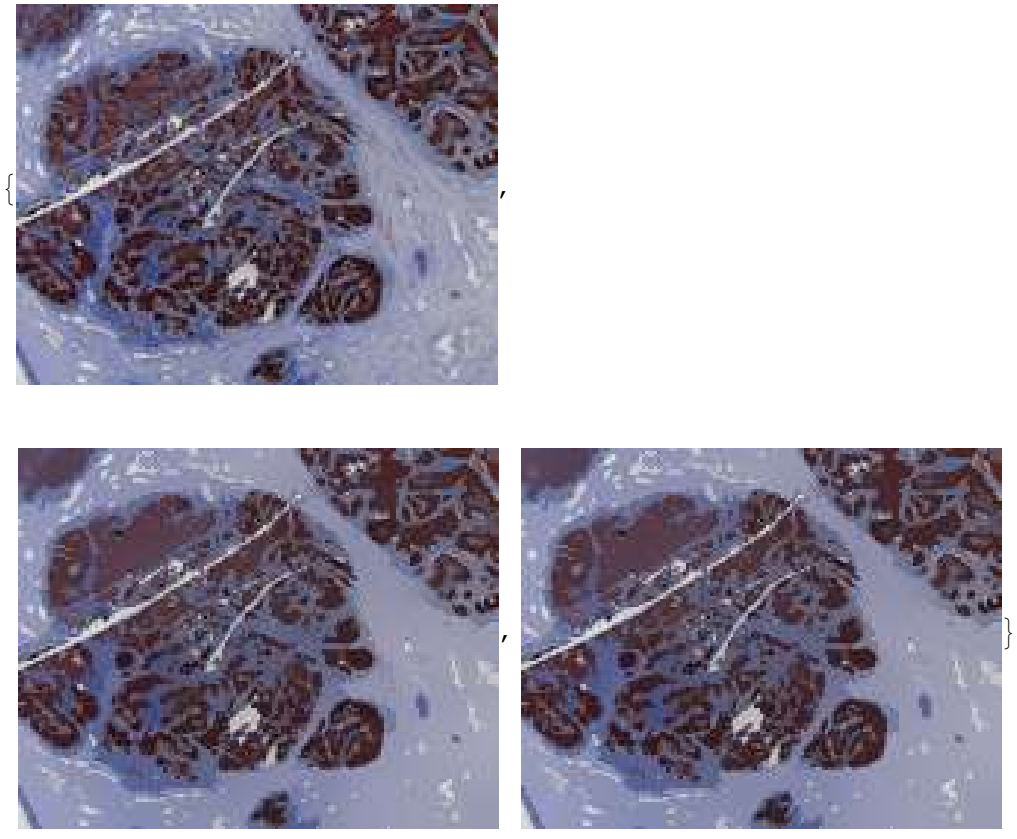

```
mstestbild = NestList[MeanShiftFilter[#, 3, 0.05, MaxIterations → 10] &,
    ImageResize[p16bild, Scaled[1 / 8]], 2];

Graphics3D[({ColorData[1, RandomInteger[100]], Line[#]}) & /@
   Flatten[Transpose[ImageData[#][[ ;; ;; 8, ;; ;; 8]] & /@mstestbild, {3, 1, 2, 4}], 1],
 PlotRange → {{0, 1}, {0, 1}, {0, 1}},
 AxesLabel → {Text[Style["R", Red, Italic]], Text[Style["G", Green, Italic]],
    Text[Style["B", Blue, Italic]]}, Axes → True, RotationAction → "Clip"]
```



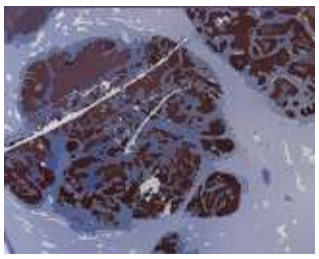
```
Graphics3D[
    ({RGBColor[#], PointSize[.0075], Point[#]}) & /@ Flatten[ImageData[#], 1][[ ;; ;; 10]],
    PlotRange → {{0, 1}, {0, 1}, {0, 1}}, AxesLabel → {Text[Style["R", Red, Italic]],
      Text[Style["G", Green, Italic]], Text[Style["B", Blue, Italic]]},
    Axes → True, RotationAction → "Clip", ImageSize → 160] & /@ mstestbild
```




```
(*Schrittvariation*)
segbildliste =
  Show[MeanShiftFilter[ ImageResize[p16bild, Scaled[1 / 8]], 3, .05, MaxIterations → #],
     ImageSize → 250] & /@ (Range[2] * 10);
segbildliste = Prepend[segbildliste, Show[ImageResize[p16bild, Scaled[1 / 8]], ImageSize → 250]]
```



```
segbild = MeanShiftFilter[ImageResize[p16bild, Scaled[1 / 8]], 3, .05, MaxIterations → 10]
```



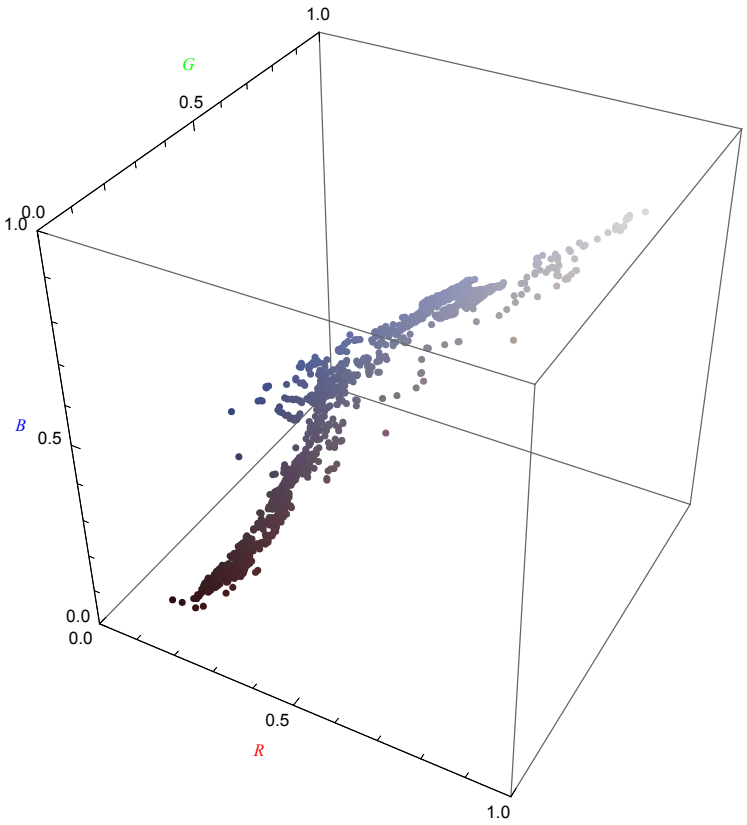
```
Union[Flatten[Round[ImageData[ImageResize[p16bild, Scaled[1 / 8]]], 0.1], 1]] // Length
```

90

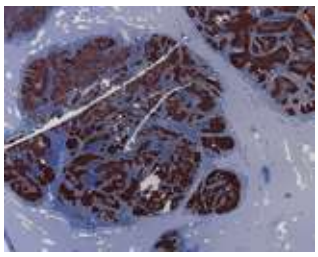
```
Union[Flatten[Round[ImageData[segbild], 0.1], 1]] // Length
```

```
62
```

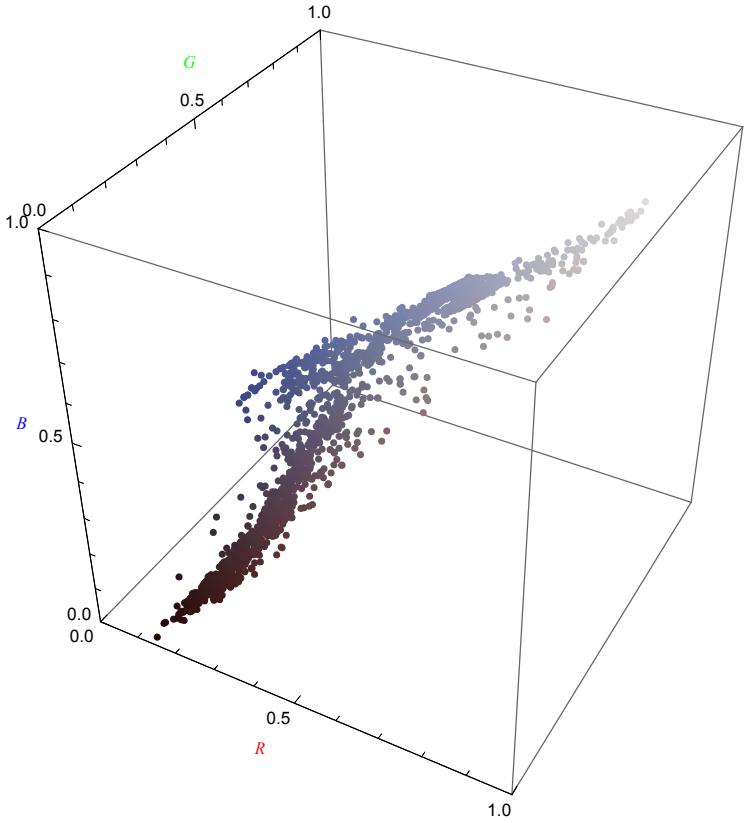
```
Graphics3D[({RGBColor[#], PointSize[.01], Point[#]}) & /@
  Flatten[ImageData[segbild][[ ;; ;; 10]], 1], PlotRange → {{0, 1}, {0, 1}, {0, 1}},
 AxesLabel → {Text[Style["R", Red, Italic]], Text[Style["G", Green, Italic]],
   Text[Style["B", Blue, Italic]]}, Axes → True, RotationAction → "Clip", ImageSize → 384]
```



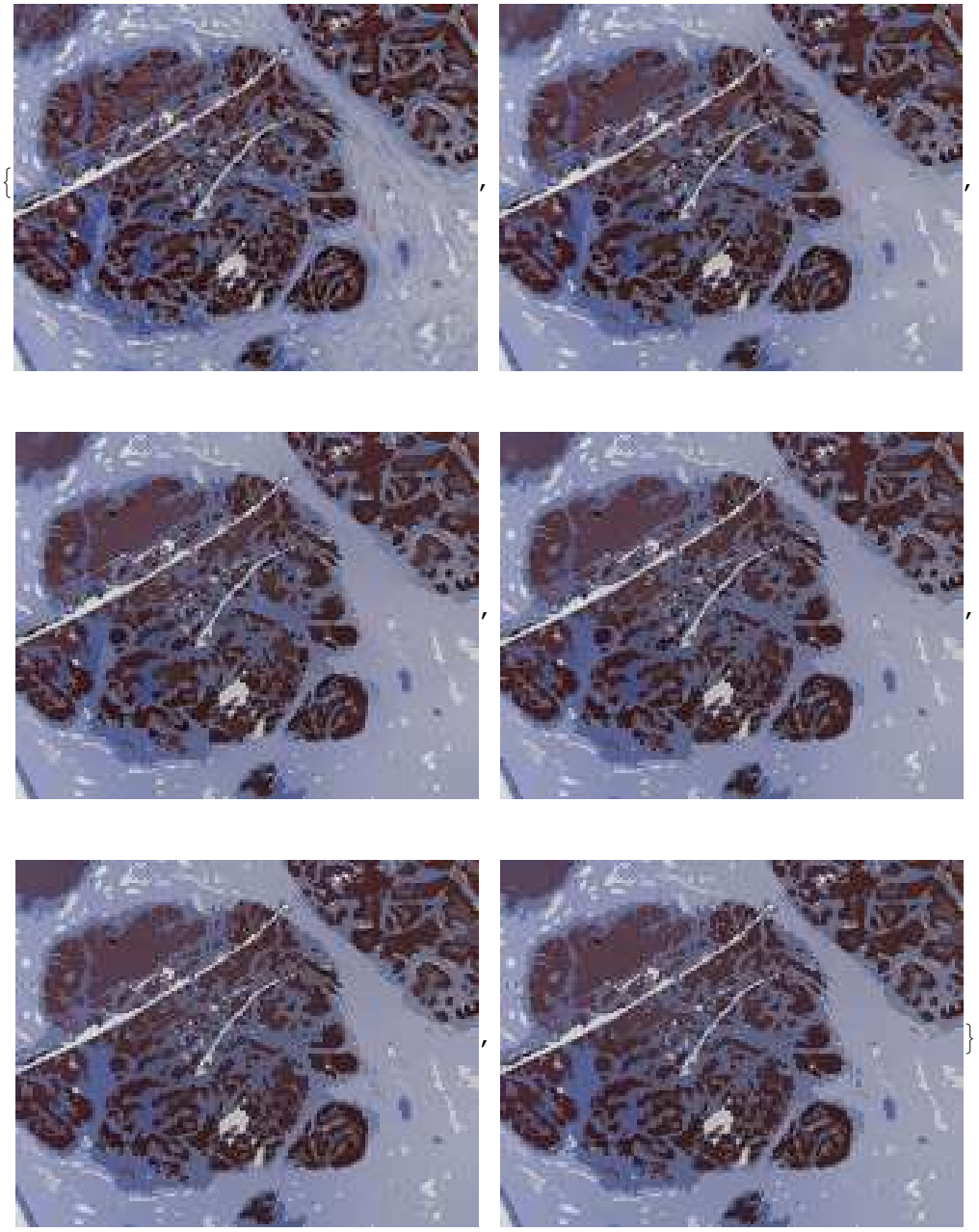

```
(*nach 10 Schritten, aber mit Kreismaske*)
eigenes = Image[Nest[MyMeanShiftFilter[#, kreismaske[3], 0.05] &,
   ImageData[ImageResize[p16bild, Scaled[1 / 8]]], 10]]
```



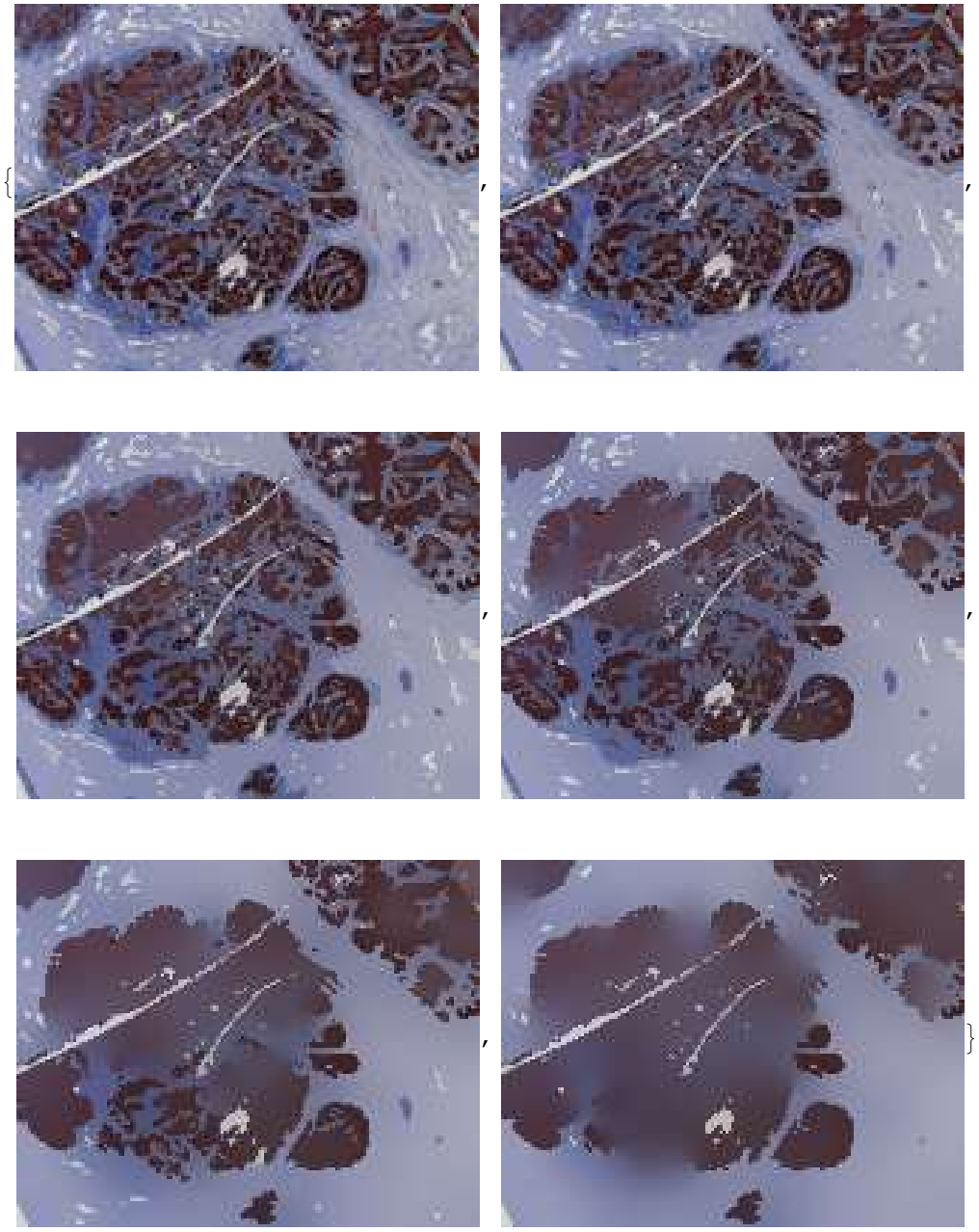
```
Graphics3D[({RGBColor[#], PointSize[.01], Point[#]}) & /@
  Flatten[ImageData[eigenes], 1][[ ;;  ;; 10]], PlotRange → {{0, 1}, {0, 1}, {0, 1}},
 AxesLabel → {Text[Style["R", Red, Italic]], Text[Style["G", Green, Italic]],
   Text[Style["B", Blue, Italic]]}, Axes → True, RotationAction → "Clip", ImageSize → 384]
```



```
(*Radiusvariation*)
segbildliste =
 Show[MeanShiftFilter[ImageResize[p16bild, Scaled[1 / 8]], #, .05, MaxIterations → 20],
    ImageSize → 250] & /@ {0, 1, 2, 3, 4, 5}
```



```
(*Distanzvariation*)
segbildliste =
 Show[MeanShiftFilter[ImageResize[p16bild, Scaled[1 / 8]], 3, #, MaxIterations → 20],
    ImageSize → 250] & /@ {0, .025, 0.05, 0.075, 0.1, 0.125}
```

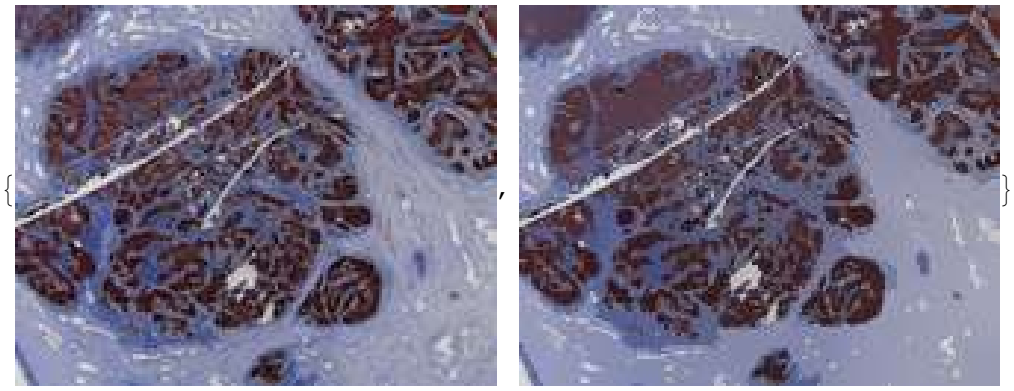


Eigenschaftes des Mean-Shift-Verfahrens:

1. die lokalen Dichtemaxima im Farbraum werden unter Einbeziehung einer lokalen Nachbarschaft bestimmt
2. es erfolgt keine Bestimmung eines Indexbildes mit einer Anzahl vorgegebener Klassen
3. das Verfahren homogenisiert lokal, so daß nachfolgend eingesetzte Verfahren wie k-Means weniger störanfällig für Schattierungen etc. werden
4. das Verfahren ist außerordentlich berechnungsaufwendig

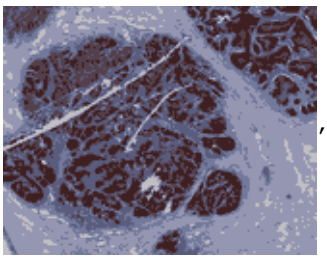
Erweiterung: Kaskadierung von Mean-Shift und k-Means bzw. hierarch. Partitionierung

```
Show[#, ImageSize → 250] & /@ {First[mstestbild], Last[mstestbild]}
```

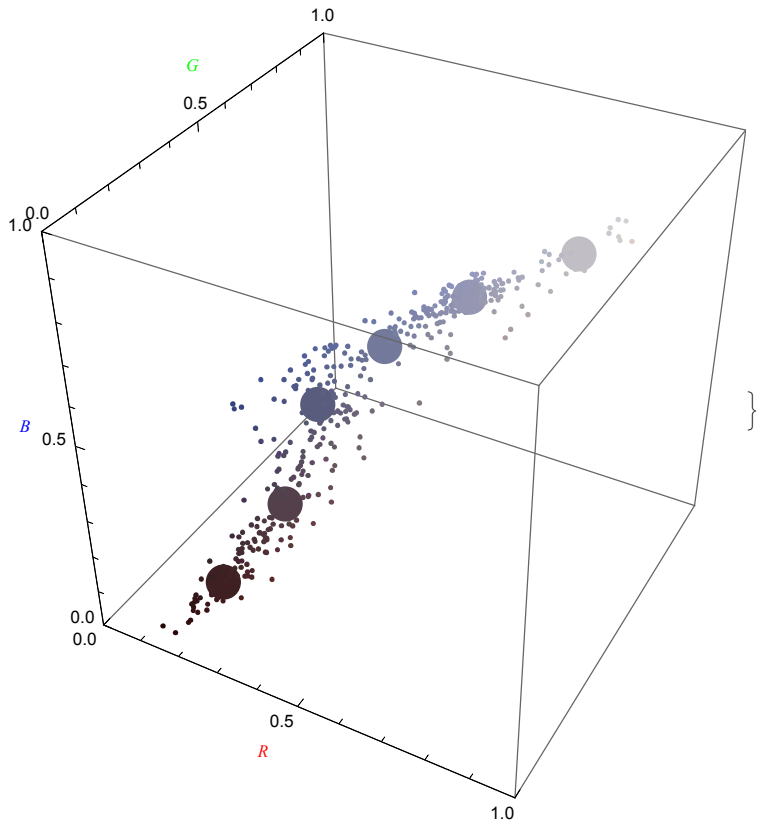


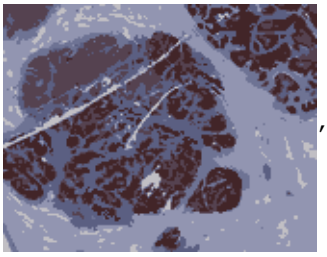
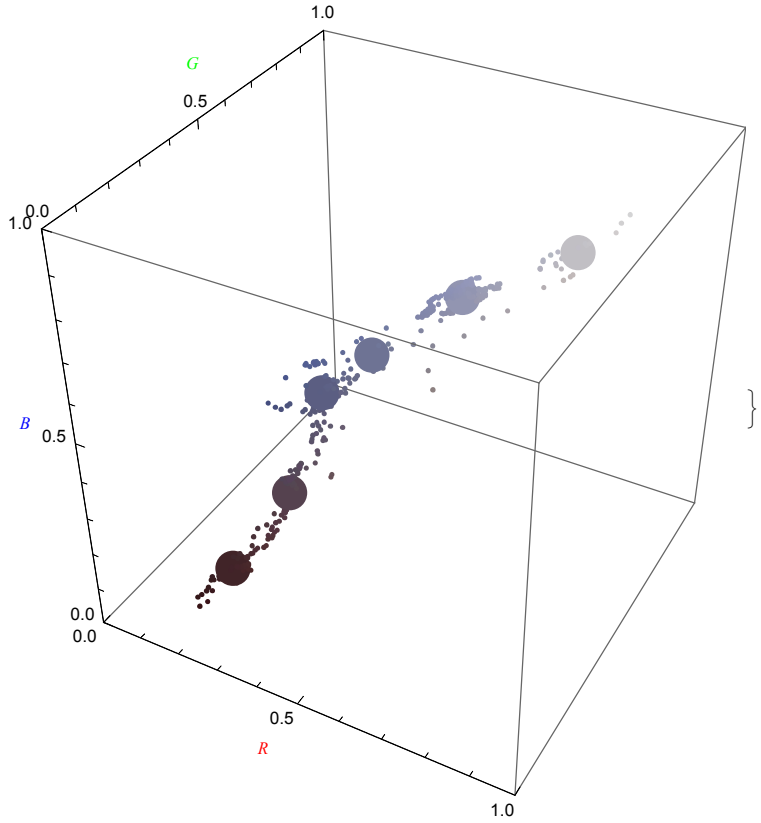
```
kmeans[#, 6, False] & /@ {First[mstestbild], Last[mstestbild]};
```

$$\left\{\begin{pmatrix} 0.351587 & 0.360779 & 0.493017 \\ 0.329471 & 0.250522 & 0.293788 \\ 0.24414 & 0.123086 & 0.134965 \\ 0.450297 & 0.473013 & 0.604415 \\ 0.584479 & 0.593299 & 0.698999 \\ 0.757516 & 0.748911 & 0.76965 \end{pmatrix},\ \right.$$

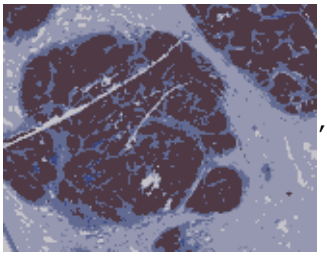
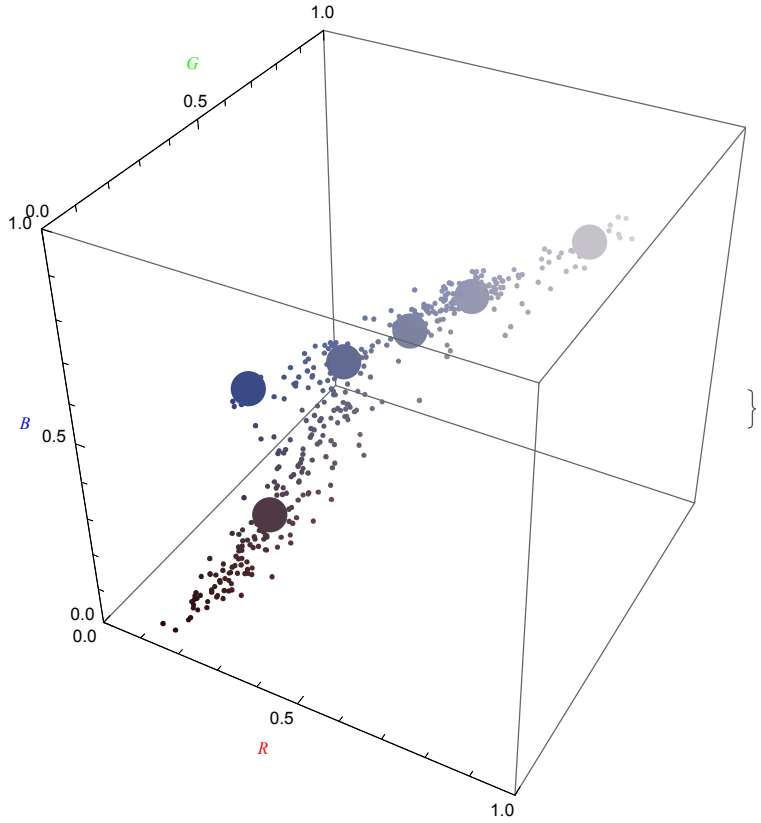


,

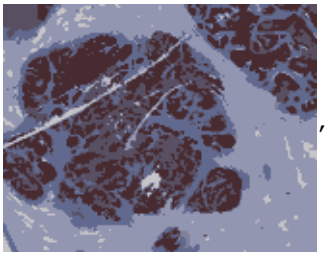


$$\left\{\begin{pmatrix} 0.335003 & 0.260154 & 0.312298 \\ 0.431052 & 0.452554 & 0.582013 \\ 0.573051 & 0.583589 & 0.693741 \\ 0.756055 & 0.747175 & 0.767057 \\ 0.356018 & 0.370956 & 0.510928 \\ 0.257064 & 0.143239 & 0.157177 \end{pmatrix},\ \text{  },\ \text{  }\right\}$$

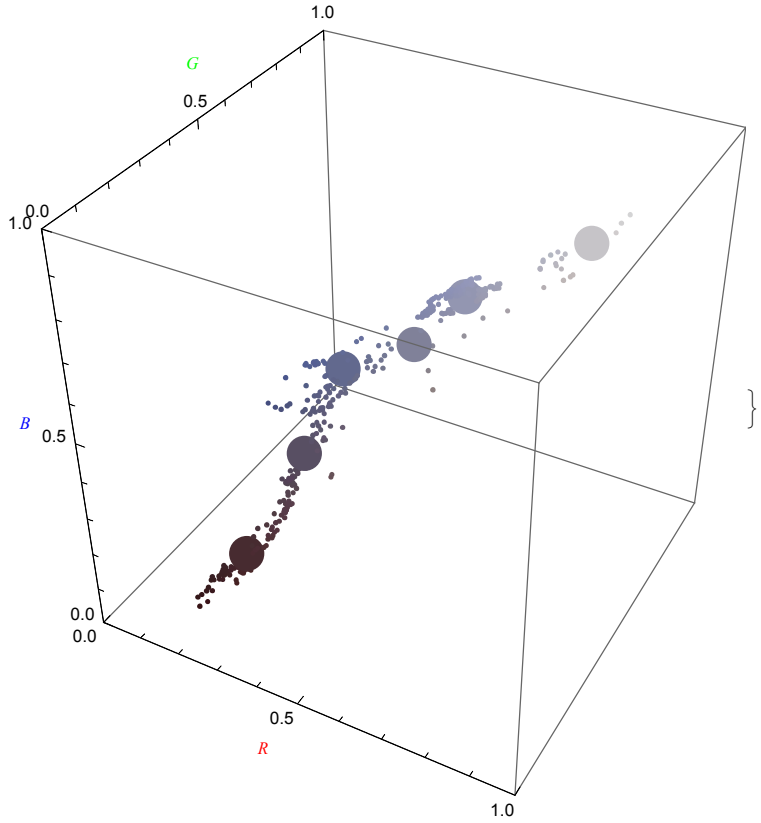

```
colquant[#, 6, False]; & /@ {First[mstestbild], Last[mstestbild]};
```

$$\left\{\begin{pmatrix} 0.227451 & 0.290196 & 0.529412 \\ 0.305882 & 0.223529 & 0.266667 \\ 0.384314 & 0.415686 & 0.572549 \\ 0.490196 & 0.509804 & 0.627451 \\ 0.588235 & 0.596078 & 0.694118 \\ 0.772549 & 0.764706 & 0.788235 \end{pmatrix},\ \text{  },\ \text{  }\right\}$$

$$\left\{\begin{pmatrix} 0.278431 & 0.168627 & 0.188235 \\ 0.345098 & 0.309804 & 0.388235 \\ 0.384314 & 0.411765 & 0.556863 \\ 0.501961 & 0.505882 & 0.6 \\ 0.576471 & 0.588235 & 0.694118 \\ 0.776471 & 0.768627 & 0.784314 \end{pmatrix},\right.$$

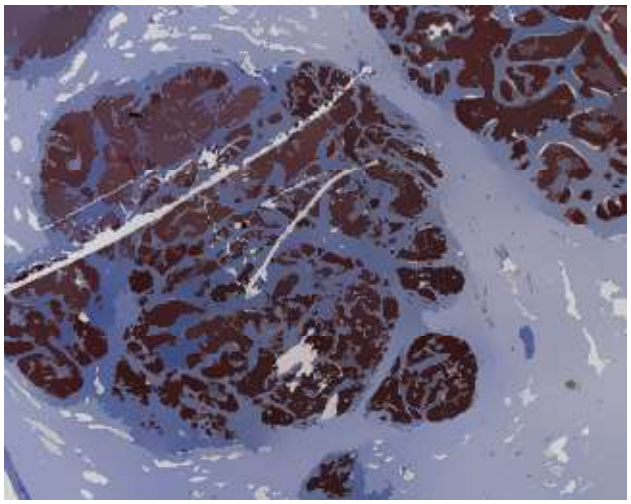


,

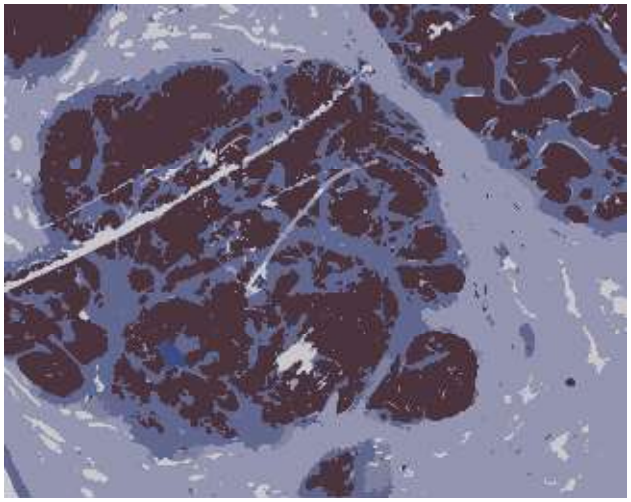


}

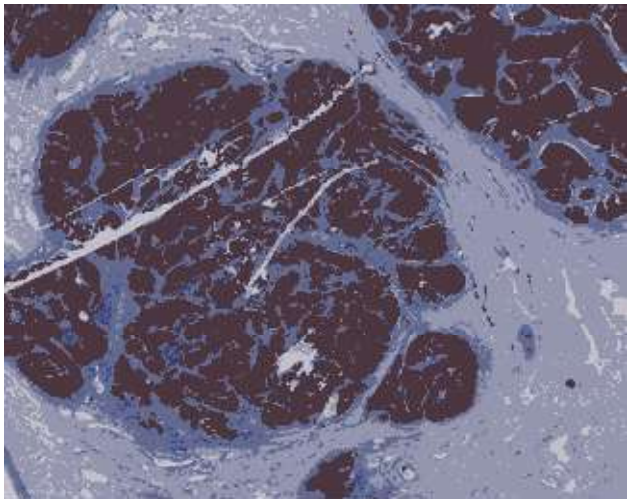
```
t = MeanShiftFilter[ImageResize[p16bild, Scaled[1 / 4]], 3, .05, MaxIterations → 30]
```



```
(*hierarch. Partitionierung das Mean-Shift-Ergebnisses*)
cq = ColorQuantize[t, 6, Dithering → False]
```



```
(*hierarch. Partitionierung direkt angewendet*)
ColorQuantize[ImageResize[p16bild, Scaled[1 / 4]], 6, Dithering → False]
```

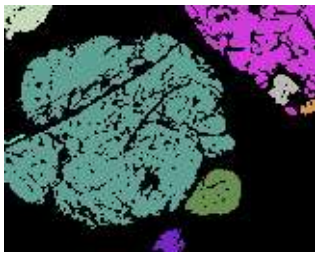


23. Einige Aspekte der Formbeschreibung

```
testbild1 = DeleteSmallComponents[ColorNegate@
      Image@MyBinarizeRGB[ImageData[#, "Byte"], MyISODATARGB[ImageData[#, "Real"] * 255]],
    10] &[ImageResize[p16bild, Scaled[1 / 8]]]
```

```
{{N.A., N.A.}, {{109.699, 102.995, 124.499}, {109.899, 103.195, 124.699}}}
{{10 429, 10 598}, {{78.1173, 61.5992, 77.3211}, {140.975, 143.928, 171.124}}}
{{10 159, 10 868}, {{77.836, 60.5955, 75.5269}, {139.677, 142.821, 170.47}}}
{{9985, 11 042}, {{77.5661, 59.9236, 74.46}, {138.946, 142.133, 169.939}}}
{{9886, 11 141}, {{77.428, 59.545, 73.8362}, {138.523, 141.738, 169.644}}}
{{9829, 11 198}, {{77.3468, 59.3247, 73.4788}, {138.284, 141.513, 169.47}}}
{{9788, 11 239}, {{77.2914, 59.1657, 73.2195}, {138.11, 141.352, 169.346}}}
{{9759, 11 268}, {{77.2343, 59.0493, 73.0499}, {138.003, 141.241, 169.245}}}
{{9745, 11 282}, {{77.2109, 58.9927, 72.9657}, {137.947, 141.188, 169.199}}}
{{9735, 11 292}, {{77.1944, 58.9538, 72.9042}, {137.908, 141.149, 169.166}}}
{{9727, 11 300}, {{77.1843, 58.9233, 72.8523}, {137.874, 141.117, 169.143}}}
{{9723, 11 304}, {{77.1777, 58.9072, 72.828}, {137.858, 141.102, 169.13}}}
{{9718, 11 309}, {{77.167, 58.8868, 72.7995}, {137.84, 141.083, 169.112}}}
{{9718, 11 309}, {{77.167, 58.8868, 72.7995}, {137.84, 141.083, 169.112}}}
```

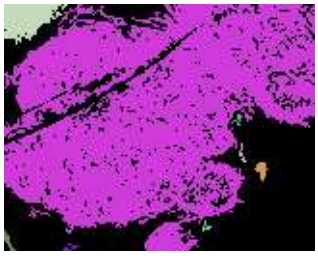
Colorize@MorphologicalComponents@testbild1



```
testbild2 = DeleteSmallComponents[ColorNegate@
      Image@MyBinarizeRGB[ImageData[#, "Byte"], MyISODATARGB[ImageData[#, "Real"] * 255]],
    10] &[ImageResize[hebild, Scaled[1 / 8]]]
```

```
{{N.A., N.A.}, {{119.685, 89.0628, 133.946}, {119.885, 89.2628, 134.146}}}
{{11 844, 9183}, {{97.4374, 71.7808, 122.352}, {148.609, 111.582, 149.129}}}
{{12 622, 8405}, {{98.7044, 72.8971, 123.419}, {151.442, 113.589, 150.006}}}
{{13 052, 7975}, {{99.595, 73.4553, 123.863}, {152.828, 114.87, 150.712}}}
{{13 305, 7722}, {{100.15, 73.7815, 124.088}, {153.617, 115.665, 151.205}}}
{{13 444, 7583}, {{100.457, 73.9531, 124.223}, {154.052, 116.128, 151.462}}}
{{13 515, 7512}, {{100.611, 74.0462, 124.292}, {154.282, 116.359, 151.595}}}
{{13 551, 7476}, {{100.687, 74.096, 124.328}, {154.402, 116.473, 151.662}}}
{{13 561, 7466}, {{100.707, 74.1111, 124.339}, {154.438, 116.502, 151.679}}}
{{13 566, 7461}, {{100.719, 74.1169, 124.344}, {154.453, 116.52, 151.689}}}
{{13 566, 7461}, {{100.719, 74.1169, 124.344}, {154.453, 116.52, 151.689}}}
```

Colorize@MorphologicalComponents@testbild2



Einfachste Größen: Pixelzahl (Fläche) und Kantenpixelzahl (Umfang)

ComponentMeasurements[testbild1, {"Count"}]

```
{1 → {294}, 2 → {1645}, 3 → {6881}, 4 → {142}, 5 → {41}, 6 → {18}, 7 → {18}, 8 → {480}, 9 → {130}}
```

flächen1 = ComponentMeasurements[testbild1, {"Count"}][[All, 2]] // Flatten

```
{294, 1645, 6881, 142, 41, 18, 18, 480, 130}
```

ComponentMeasurements[testbild2, {"Count"}]

```
{1 → {430}, 2 → {12 776}, 3 → {14}, 4 → {14}, 5 → {38}, 6 → {17}, 7 → {12}, 8 → {32}, 9 → {18}}
```

flächen2 = ComponentMeasurements[testbild2, {"Count"}][[All, 2]] // Flatten

```
{430, 12 776, 14, 14, 38, 17, 12, 32, 18}
```

ComponentMeasurements[testbild1, {"PolygonalLength"}]

```
{1 → {115.876}, 2 → {735.118}, 3 → {3035.09}, 4 → {69.5477},
 5 → {30.6924}, 6 → {13.6569}, 7 → {18.4142}, 8 → {163.01}, 9 → {71.4056}}
```

ComponentMeasurements[testbild1, {"PerimeterLength"}]

```
{1 → {134}, 2 → {924}, 3 → {3746}, 4 → {88}, 5 → {40}, 6 → {20}, 7 → {34}, 8 → {182}, 9 → {98}}
```

umfänge1 = ComponentMeasurements[testbild1, {"PerimeterLength"}][[All, 2]] // Flatten

```
{134, 924, 3746, 88, 40, 20, 34, 182, 98}
```

```
ComponentMeasurements[testbild2, {"PolygonalLength"}]
```

```
{1 → {122.027}, 2 → {4106.52}, 3 → {14.364}, 4 → {14.5355},
 5 → {24.8995}, 6 → {17.9853}, 7 → {12.3284}, 8 → {29.2782}, 9 → {17.6213}}
```

```
ComponentMeasurements[testbild2, {"PerimeterLength"}]
```

```
{1 → {156}, 2 → {4790}, 3 → {32}, 4 → {32}, 5 → {34}, 6 → {48}, 7 → {26}, 8 → {42}, 9 → {36}}
```

```
umfänge2 = ComponentMeasurements[testbild2, {"PerimeterLength"}][[All, 2]] // Flatten
```

```
{156, 4790, 32, 32, 34, 48, 26, 42, 36}
```

Kompaktheit: Segmentfläche bezogen auf Kreisfläche gleichen Umfangs

äquiperimetr. Kreisfläche bestimmt über $A_c = \pi d^2$ und $P_c = \pi d$ als $A_c = \frac{P_c^2}{4\pi}$

```
äquiflächen1 = (umfänge1^2) / (4 π) // N
```

```
{1428.89, 67941.3, 1.11667×10^6, 616.248, 127.324, 31.831, 91.9916, 2635.92, 764.262}
```

```
äquiflächen2 = (umfänge2^2) / (4 π) // N
```

```
{1936.6, 1.82583×10^6, 81.4873, 81.4873, 91.9916, 183.346, 53.7944, 140.375, 103.132}
```

```
kompaktheiten1 = flächen1 / äquiflächen1
```

```
{0.205754, 0.0242121, 0.00616206, 0.230427, 0.322013, 0.565487, 0.19567, 0.182099, 0.170099}
```

```
Flatten@ComponentMeasurements[testbild1, {"Circularity"}][[All, 2]]^2
```

```
{0.205754, 0.0242121, 0.00616206, 0.230427, 0.322013, 0.565487, 0.19567, 0.182099, 0.170099}
```

```
kompaktheiten2 = flächen2 / äquiflächen2
```

```
{0.222039, 0.00699735, 0.171806, 0.171806, 0.413081, 0.0927206, 0.223072, 0.227961, 0.174533}
```

```
Flatten@ComponentMeasurements[testbild2, {"Circularity"}][[All, 2]]^2
```

```
{0.222039, 0.00699735, 0.171806, 0.171806, 0.413081, 0.0927206, 0.223072, 0.227961, 0.174533}
```

```
gewichtetegesamtkompaktheit1 = Total[kompaktheiten1 * flächen1] / Total[flächen1]
```

```
0.0323211
```

```
gewichtetegesamtkompaktheit2 = Total[kompaktheiten2 * flächen2] / Total[flächen2]
```

```
0.0164836
```

Solidität: Segmentfläche bezogen auf Fläche der konvexen Hülle

```
ComponentMeasurements[testbild1, {"ConvexCount"}]
```

```
{1 → {337}, 2 → {2327}, 3 → {9496}, 4 → {167}, 5 → {47}, 6 → {18}, 7 → {26}, 8 → {531}, 9 → {166}}
```

```
hüllflächen1 = ComponentMeasurements[testbild1, {"ConvexCount"}][[All, 2]] // Flatten // N
```

```
{337., 2327., 9496., 167., 47., 18., 26., 531., 166.}
```

```
ComponentMeasurements[testbild2, {"ConvexCount"}]
```

```
{1 → {493}, 2 → {17228}, 3 → {24}, 4 → {25}, 5 → {45}, 6 → {42}, 7 → {18}, 8 → {40}, 9 → {28}}
```

```
hüllflächen2 = ComponentMeasurements[testbild2, {"ConvexCount"}][[All, 2]] // Flatten // N
```

```
{493., 17228., 24., 25., 45., 42., 18., 40., 28.}
```

```
soliditäten1 = flächen1 / hüllflächen1
```

```
{0.872404, 0.706919, 0.724621, 0.850299, 0.87234, 1., 0.692308, 0.903955, 0.783133}
```

```
soliditäten2 = flächen2 / hüllflächen2
```

```
{0.872211, 0.741583, 0.583333, 0.56, 0.844444, 0.404762, 0.666667, 0.8, 0.642857}
```

```
gewichtetegesamtsolidität1 = Total[soliditäten1 * flächen1] / Total[flächen1]
```

```
0.738746
```

```
gewichtetegesamtsolidität2 = Total[soliditäten2 * flächen2] / Total[flächen2]
```

```
0.745238
```

Zirkularität: Kreisumfang gleicher Fläche bezogen auf Segmentumfang

```
gleichflächigekreisumfänge1 = Sqrt[4 flächen1 * Pi] // N
```

```
{60.7825, 143.776, 294.056, 42.2425, 22.6985, 15.0398, 15.0398, 77.665, 40.4182}
```

```
gleichflächigekreisumfänge2 = Sqrt[4 flächen2 * Pi] // N
```

```
{73.5088, 400.684, 13.2638, 13.2638, 21.8523, 14.616, 12.2799, 20.053, 15.0398}
```

```
zirkularitäten1 = gleichflächigekreisumfänge1 / umfänge1
```

```
{0.453601, 0.155602, 0.0784988, 0.480028, 0.567462, 0.751988, 0.442346, 0.426731, 0.41243}
```

```
ComponentMeasurements[testbild1, {"Circularity"}]
```

```
{1 → {0.453601}, 2 → {0.155602}, 3 → {0.0784988}, 4 → {0.480028},
 5 → {0.567462}, 6 → {0.751988}, 7 → {0.442346}, 8 → {0.426731}, 9 → {0.41243}}
```

```
zirkularitäten2 = gleichflächigekreisumfänge2 / umfänge2
```

```
{0.47121, 0.0836502, 0.414495, 0.414495, 0.642714, 0.304501, 0.472305, 0.477453, 0.417771}
```

```
ComponentMeasurements[testbild2, {"Circularity"}]
```

```
{1 → {0.47121}, 2 → {0.0836502}, 3 → {0.414495}, 4 → {0.414495},
 5 → {0.642714}, 6 → {0.304501}, 7 → {0.472305}, 8 → {0.477453}, 9 → {0.417771}}
```

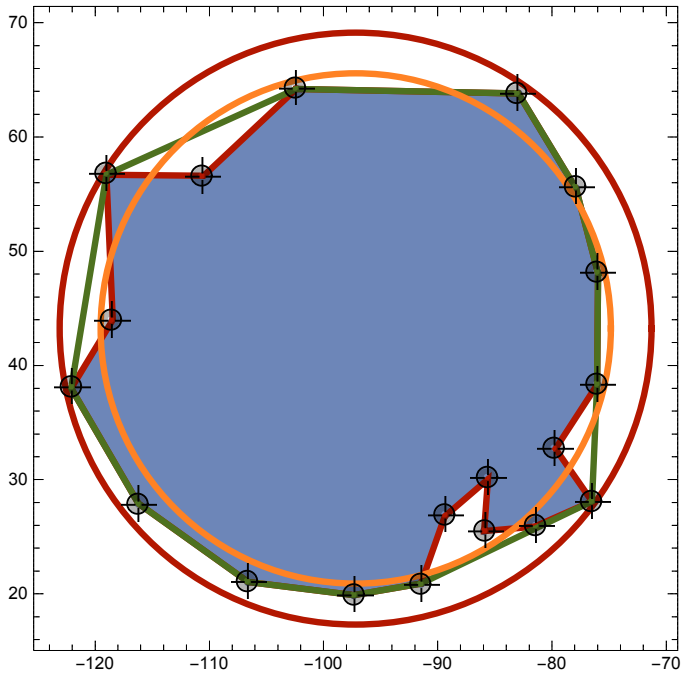
```
gewichtetegesamtzirkularität1 = Total[zirkularitäten1 * flächen1] / Total[flächen1]
```

```
0.134817
```

```
gewichtetegesamtzirkularität2 = Total[zirkularitäten2 * flächen2] / Total[flächen2]
```

```
0.100442
```

BeispielZirkularitätKompaktheitSolidität



polygonale Fläche A_o	1567.43
konvexe Hüllfläche A_h	1672.6
zu A_o gehöriger polygonaler Umfang L_o	162.788
zu A_o flächengleicher Kreisumfang L_{oc}	140.346
zu A_o umfangsgleiche (L_o) Kreisfläche A_c	2108.79
Zirkularität $Z=L_{oc}/L_o$	**0.862139**
Kompaktheit $Z^2=K=A_o/A_c$	**0.743284**
Solidität $S=A_o/A_h$	**0.937125**

Diskrete Kompaktheit: alternative Kompaktheitsbestimmung über Pixelkontaktkantenanzahl

Ernesto Bribiesca 1997

Ansatz:

$$2 P_k + P = 4 n$$

mit P_k als Pixelkontaktkantenanzahl
und P als Anzahl der äußeren Pixelkanten des Segments

$$P_{min} = 4 \sqrt{n}$$

daraus folgt durch Einsetzen die im Minimalfall P_{min} vorhandene maximale Pixelkontaktkantenzahl:

$$P_{k_{max}} = \frac{4n - 4\sqrt{n}}{2} = 2\left(n - \sqrt{n}\right)$$

Die minimale Pixelkontaktkantenzahl $P_{k_{min}}$ ist entweder 0 (8er-Nachbarschaft) oder $n - 1$ (4er-Nachbarschaft).

Damit wird die diskrete Kompaktheit definiert als

$$C_d = \frac{P_k - P_{k_{min}}}{P_{k_{max}} - P_{k_{min}}}$$

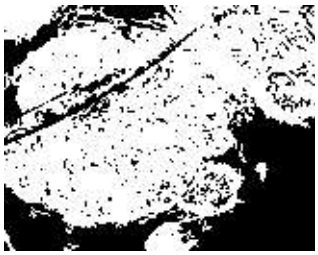
im Falle der 4er-Nachbarschaft:

$$C_d = \frac{P_k}{P_{k_{max}}} = \frac{P_k}{2\left(n - \sqrt{n}\right)} = \frac{n - P/4}{n - \sqrt{n}}$$

```
CountEdges[img_Image] := Module[{bm = ImageData[img, "Bit"]},
  Plus @@ Flatten[ListConvolve[{{0, 1, 0}, {0, 0, 1}, {0, 0, 0}}, ArrayPad[bm, 1], {-1, 1}] * bm]]

DiskreteKompaktheit2D[img_Image] := Module[
   {labels, uc, allekomponenten, ngesamt, ucmaxgesamt, kompaktheiten, kompaktheitsummiert},
   labels = MorphologicalComponents[ImagePad[img, 1], 0.5];
   uc = Function[{label}, CountEdges[Binarize[Image[labels], # == label &]]] /@
     Rest[Sort[Union[Flatten[labels]]]];
   allekomponenten = Length[uc];
   Print["Anzahl der Segmente: ", allekomponenten];
   ngesamt = Plus @@ Rest[Sort[Tally[Flatten[labels]]]][[All, 2]];
   ucmaxgesamt = (ngesamt - ngesamt^(1/2)) * 2 // N;
   kompaktheitsummiert = (Plus @@ uc) / ucmaxgesamt;
   Print["Gesamtkompaktheit (min=0): ", kompaktheitsummiert];
   kompaktheitsummiert
  ];

testbild1
DiskreteKompaktheit2D[testbild1];
```

```
Anzahl der Segmente: 9

Gesamtkompaktheit (min=0): 0.872443
```

```
testbild2
DiskreteKompaktheit2D[testbild2];
```



```
Anzahl der Segmente: 9
Gesamtkompaktheit (min=0): 0.910585
```

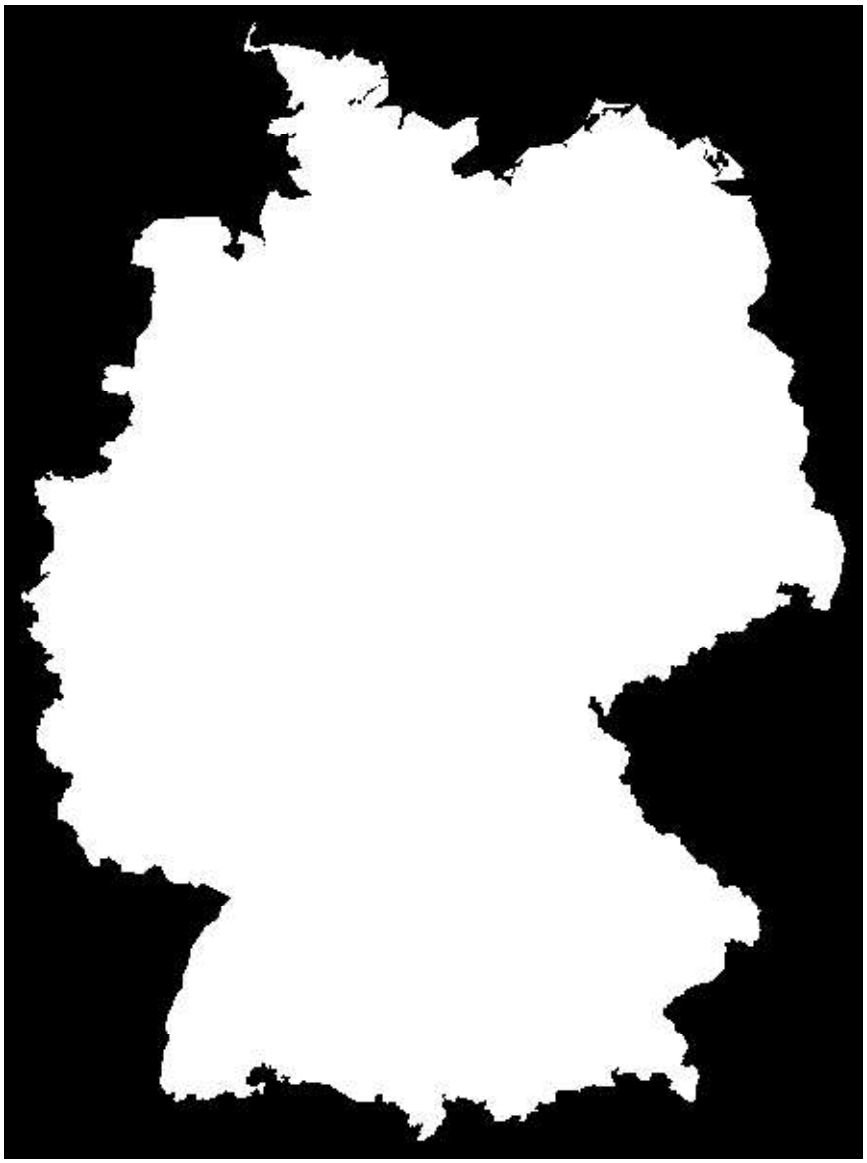
Zum Vergleich die Kompaktheit aus Segmentfläche bezogen auf Kreisfläche gleichen Umfangs

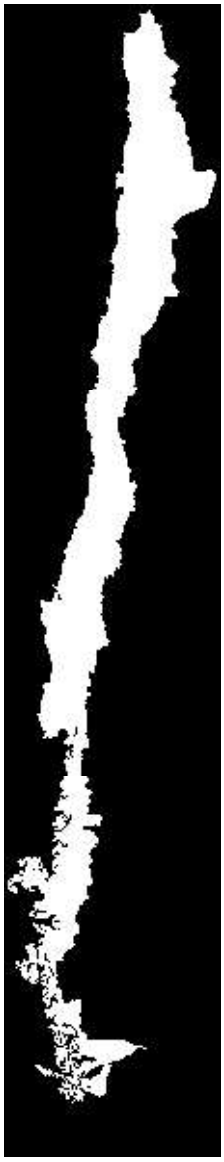
```
{Total[#[[All, 1]] / (#[[All, 2]] ^2. / (4 π)) * #[[All, 1]]] / Total[#[[All, 1]]]} &[
 Transpose@{Flatten[ComponentMeasurements[#, {"Count"}][[All, 2]]],
     Flatten[ComponentMeasurements[#, {"PerimeterLength"}][[All, 2]]]} &[testbild1]]

{Total[#[[All, 1]] / (#[[All, 2]] ^2. / (4 π)) * #[[All, 1]]] / Total[#[[All, 1]]]} &[
 Transpose@{Flatten[ComponentMeasurements[#, {"Count"}][[All, 2]]],
     Flatten[ComponentMeasurements[#, {"PerimeterLength"}][[All, 2]]]} &[testbild2]]
```

```
{0.0323211}
{0.0164836}
```

Die diskrete Kompaktheit ist sehr unempfindlich gegenüber “bloßen” Berandungsänderungen:

```
mexicoshapeimage = DeleteSmallComponents@ColorNegate@
    Binarize[Image[Graphics[CountryData["Mexico", "Shape"][[1, 3]]], ImageResolution → 100], .5]
germanyshapeimage = DeleteSmallComponents@ColorNegate@Binarize[
     Image[Graphics[CountryData["Germany", "Shape"][[1, 3]]], ImageResolution → 100], .5]
chileshapeimage = DeleteSmallComponents@ColorNegate@
    Binarize[Image[Graphics[CountryData["Chile", "Shape"][[1, 3]]], ImageResolution → 100], .5]
```





```
DiskreteKompaktheit2D[mexicoshapeimage];
DiskreteKompaktheit2D[germanyshapeimage];
DiskreteKompaktheit2D[chileshapeimage];
```

```
Anzahl der Segmente: 1
Gesamtkompaktheit (min=0): 0.988646
Anzahl der Segmente: 1
Gesamtkompaktheit (min=0): 0.996413
Anzahl der Segmente: 1
Gesamtkompaktheit (min=0): 0.951641
```

Zum Vergleich die Kompaktheit (Segmentfläche bezogen auf Kreisfläche gleichen Umfangs)

```
(Flatten[ComponentMeasurements[#, {"Count"}][[All, 2]]]) /
   ((Flatten[ComponentMeasurements[#, {"PerimeterLength"}][[All, 2]]]^2.) / (4 π)) &[
 mexicoshapeimage]
(Flatten[ComponentMeasurements[#, {"Count"}][[All, 2]]]) /
   ((Flatten[ComponentMeasurements[#, {"PerimeterLength"}][[All, 2]]]^2.) / (4 π)) &[
 germanyshapeimage]
(Flatten[ComponentMeasurements[#, {"Count"}][[All, 2]]]) /
   ((Flatten[ComponentMeasurements[#, {"PerimeterLength"}][[All, 2]]]^2.) /
     (4 π)) &[chileshapeimage]
```

```
{0.0630345}
```

```
{0.132681}
```

```
{0.0183908}
```

Zum Vergleich die Solidität (Segmentfläche bezogen auf Fläche der konvexen Hülle)

```
(Flatten[ComponentMeasurements[#, {"Count"}][[All, 2]]]) /
   ((Flatten[ComponentMeasurements[#, {"ConvexCount"}][[All, 2]]]) // N) &[mexicoshapeimage]
(Flatten[ComponentMeasurements[#, {"Count"}][[All, 2]]]) /
   ((Flatten[ComponentMeasurements[#, {"ConvexCount"}][[All, 2]]]) // N) &[germanyshapeimage]
(Flatten[ComponentMeasurements[#, {"Count"}][[All, 2]]]) /
   ((Flatten[ComponentMeasurements[#, {"ConvexCount"}][[All, 2]]]) // N) &[chileshapeimage]
```

```
{0.6233}
```

```
{0.821815}
```

```
{0.371299}
```

Zum Vergleich die Zirkularität (Kreisumfang gleicher Fläche bezogen auf Segmentumfang)

```
(Sqrt[4 Flatten[ComponentMeasurements[#, {"Count"}][[All, 2]]] * Pi] / Flatten[
      ComponentMeasurements[#, {"PerimeterLength"}][[All, 2]]] // N) &[mexicoshapeimage]
(Sqrt[4 Flatten[ComponentMeasurements[#, {"Count"}][[All, 2]]] * Pi] / Flatten[
      ComponentMeasurements[#, {"PerimeterLength"}][[All, 2]]] // N) &[germanyshapeimage]
(Sqrt[4 Flatten[ComponentMeasurements[#, {"Count"}][[All, 2]]] * Pi] /
     Flatten[ComponentMeasurements[#, {"PerimeterLength"}][[All, 2]]] // N) &[chileshapeimage]
```

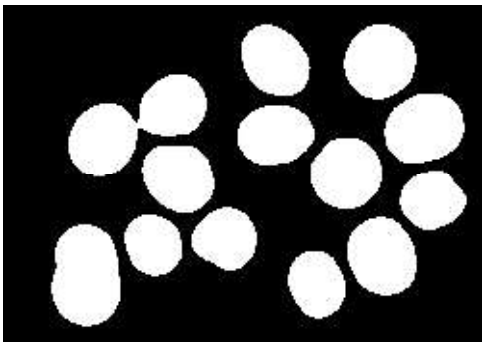
```
{0.251067}
```

```
{0.364254}
```

```
{0.135613}
```

weitere Erläuterungen zu Zirkularität, Kompaktheit und Solidität anhand eines willkürlichen Testbildes aus der Lymphkrebsforschung:

```
testbild =
```



```
;
```

In die Zirkularität gehen die Segmentfläche *A* und Segmentumfang *P* ein:

```
ComponentMeasurements[testbild, {"Count", "PerimeterLength"}]
```

```
{1 → {1008, 117.983}, 2 → {1135, 123.64}, 3 → {1973, 228.309}, 4 → {1187, 127.397},
 5 → {984, 115.74}, 6 → {1086, 119.397}, 7 → {1048, 118.569}, 8 → {798, 103.397}, 9 → {831, 105.983},
 10 → {749, 100.083}, 11 → {1148, 125.397}, 12 → {1535, 149.64}, 13 → {804, 105.497}}
```

Die Zirkularität wird bestimmt als das Verhältnis des Umfangs $P_c = \sqrt{4\,\pi\,A}$ eines flächengleichen Kreises zum gemessenen Umfang *P* als $Z = \frac{\sqrt{4\,\pi\,A}}{P}$:

```
Z = Sqrt[#[[2, 1]] * 4 π] / #[[2, 2]] & /@
  ComponentMeasurements[testbild, {"Count", "PerimeterLength"}]
```

```
{0.95393, 0.965929, 0.689678, 0.958675, 0.960768, 0.978423,
 0.967868, 0.968498, 0.964207, 0.969358, 0.957831, 0.928138, 0.952776}
```

Hier wird das arithmetische Mittel aus der Liste aller Zirkularitätswerte gebildet:

```
mtfZ = N@Mean[Sqrt[#[[2, 1]] * 4 π] / #[[2, 2]] & /@
    ComponentMeasurements[testbild, {"Count", "PerimeterLength"}]]
```

```
0.939698
```

Hier wird der Medianwert aus der Liste aller Zirkularitätswerte bestimmt:

```
mdfZ = N@Median[Sqrt[#[[2, 1]] * 4 π] / #[[2, 2]] & /@
    ComponentMeasurements[testbild, {"Count", "PerimeterLength"}]]
```

```
0.960768
```

Hier werden die einzelnen Zirkularitätswerte mit ihrer anteiligen Größe bezogen auf die gesamte Fläche von Segmenten gewichtet aufaddiert (größengewichtete Mittelung):

```
ggfZ = N[(Divide @@ Plus @@ ({Times @@ {N@Sqrt[(Last@#) * 4 π] / (First@#), Last@#}, Last@#} & /@
       ComponentMeasurements[testbild, {"PerimeterLength", "Count"}][[All, 2]]))]
```

```
0.921835
```

In die Kompaktheit gehen ebenfalls die Segmentfläche *A* und Segmentumfang *P* ein:

```
ComponentMeasurements[testbild, {"Count", "PerimeterLength"}]
```

```
{1 → {1008, 117.983}, 2 → {1135, 123.64}, 3 → {1973, 228.309}, 4 → {1187, 127.397},
 5 → {984, 115.74}, 6 → {1086, 119.397}, 7 → {1048, 118.569}, 8 → {798, 103.397}, 9 → {831, 105.983},
 10 → {749, 100.083}, 11 → {1148, 125.397}, 12 → {1535, 149.64}, 13 → {804, 105.497}}
```

Die Kompaktheit wird bestimmt als das Verhältnis der Segmentfläche *A* zur umfangsgleichen Kreisfläche $A_c = \frac{P^2}{4\pi}$ als $K = \frac{4\pi A}{P^2}$, womit zudem gilt $Z^2 = K$:

```
K = #[[2]] / (#[[1]]^2 / (4 π)) & /@
  ComponentMeasurements[testbild, {"PerimeterLength", "Count"}][[All, 2]]
```

```
{0.909983, 0.933019, 0.475655, 0.919057, 0.923075, 0.957311,
 0.936768, 0.937988, 0.929695, 0.939656, 0.91744, 0.86144, 0.907782}
```

```
Z^2
```

```
{0.909983, 0.933019, 0.475655, 0.919057, 0.923075, 0.957311,
 0.936768, 0.937988, 0.929695, 0.939656, 0.91744, 0.86144, 0.907782}
```

Hier wird das arithmetische Mittel aus der Liste aller Kompaktheitswerte gebildet:

```
mtfK = N@Mean[#[[2]] / (#[[1]]^2 / (4 π)) & /@
    ComponentMeasurements[testbild, {"PerimeterLength", "Count"}][[All, 2]]]
```

```
0.888375
```

Man beachte aber, daß das Quadrat der mittleren Zirkularität i.a. nicht der mittleren Kompaktheit entspricht:

```
mtfZ^2
```

```
0.883033
```

Hier wird der Medianwert aus der Liste aller Kompaktheitswerte bestimmt:

```
mdfK = N@Median[#[[2]] / (#[[1]]^2 / (4 π)) & /@
    ComponentMeasurements[testbild, {"PerimeterLength", "Count"}][[All, 2]]]
```

```
0.923075
```

Hingegen stimmt das Quadrat der medianen Zirkularität mit der medianen Kompaktheit überein:

```
mdfZ^2
```

```
0.923075
```

Hier werden die einzelnen Kompaktheitswerte mit ihrer anteiligen Größe bezogen auf die gesamte Fläche von Segmenten gewichtet aufaddiert (größengewichtete Mittelung):

```
ggfK = N[(Divide @@ Plus @@ ({Times @@ {Last@#/(First@#)^2 * 4 π, Last@#}, Last@#} & /@
       ComponentMeasurements[testbild, {"PerimeterLength", "Count"}][[All, 2]]))]
```

```
0.858574
```

Nur zum prinzipiellen Vergleich die diskrete Kompaktheit:

```
DiskreteKompaktheit2D[testbild];
```

```
Anzahl der Segmente: 13

Gesamtkompaktheit (min=0): 0.972719
```

Auch hinsichtlich der größengewichtet gemittelten Kompaktheit gilt i.a. nicht, daß diese dem Quadrat der größengewichtet gemittelten Zirkularität entspricht:

```
ggfZ^2
```

```
0.849781
```

In die Solidität geht neben der Segmentfläche *A* noch die konvexe Hüllfläche A_h ein:

```
ComponentMeasurements[testbild, {"ConvexCount", "Count"}]
```

```
{1 → {1028, 1008}, 2 → {1154, 1135}, 3 → {2406, 1973}, 4 → {1208, 1187},
 5 → {1001, 984}, 6 → {1102, 1086}, 7 → {1064, 1048}, 8 → {810, 798}, 9 → {850, 831},
 10 → {767, 749}, 11 → {1168, 1148}, 12 → {1574, 1535}, 13 → {820, 804}}
```

Die Solidität wird bestimmt als das Verhältnis der Segmentfläche *A* zur konvexen Hüllfläche A_h als $S = \frac{A}{A_h}$:

```
N@#[[2]] / #[[1]] & /@ ComponentMeasurements[testbild, {"ConvexCount", "Count"}][[All, 2]]
```

```
{0.980545, 0.983536, 0.820033, 0.982616, 0.983017, 0.985481,
 0.984962, 0.985185, 0.977647, 0.976532, 0.982877, 0.975222, 0.980488}
```

Hier wird das arithmetische Mittel aus der Liste aller Solidätswerte gebildet:

```
mtfS =
 N@Mean[#[[2]] / #[[1]] & /@ ComponentMeasurements[testbild, {"ConvexCount", "Count"}][[All, 2]]]
```

```
0.969088
```

Hier wird der Medianwert aus der Liste aller Kompaktheitswerte bestimmt:

```
mdfS = N@Median[
   #[[2]] / #[[1]] & /@ ComponentMeasurements[testbild, {"ConvexCount", "Count"}][[All, 2]]]
```

```
0.982616
```

Hier werden die einzelnen Soliditätswerte mit ihrer anteiligen Größe bezogen auf die gesamte Fläche von Segmenten gewichtet aufaddiert (größengewichtete Mittelung):

```
ggfS = N[(Divide @@ Plus @@ ({Times @@ {Last@# / First@#, Last@#} , Last@#} & /@
       ComponentMeasurements[testbild, {"ConvexCount", "Count"}][[All, 2]]))]
```

```
0.959149
```