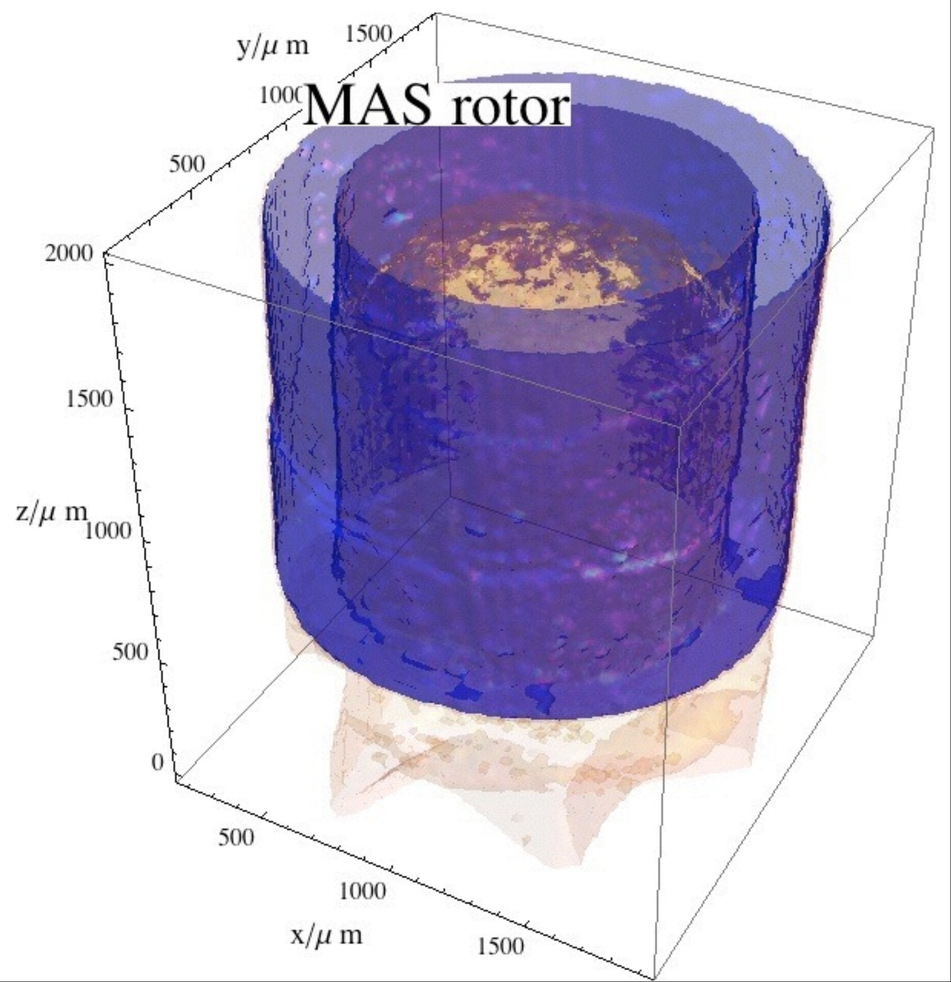
L13, 12 Mar, Segmentation: HW 7 Segmentation of MAS Rotor

1) Download Moodle/Assignments/HW7_Segmentation_MAS_Rotor.nb
2) Download Moodle/Assignments/MAS_rotor_cropped.h5 (177MB)

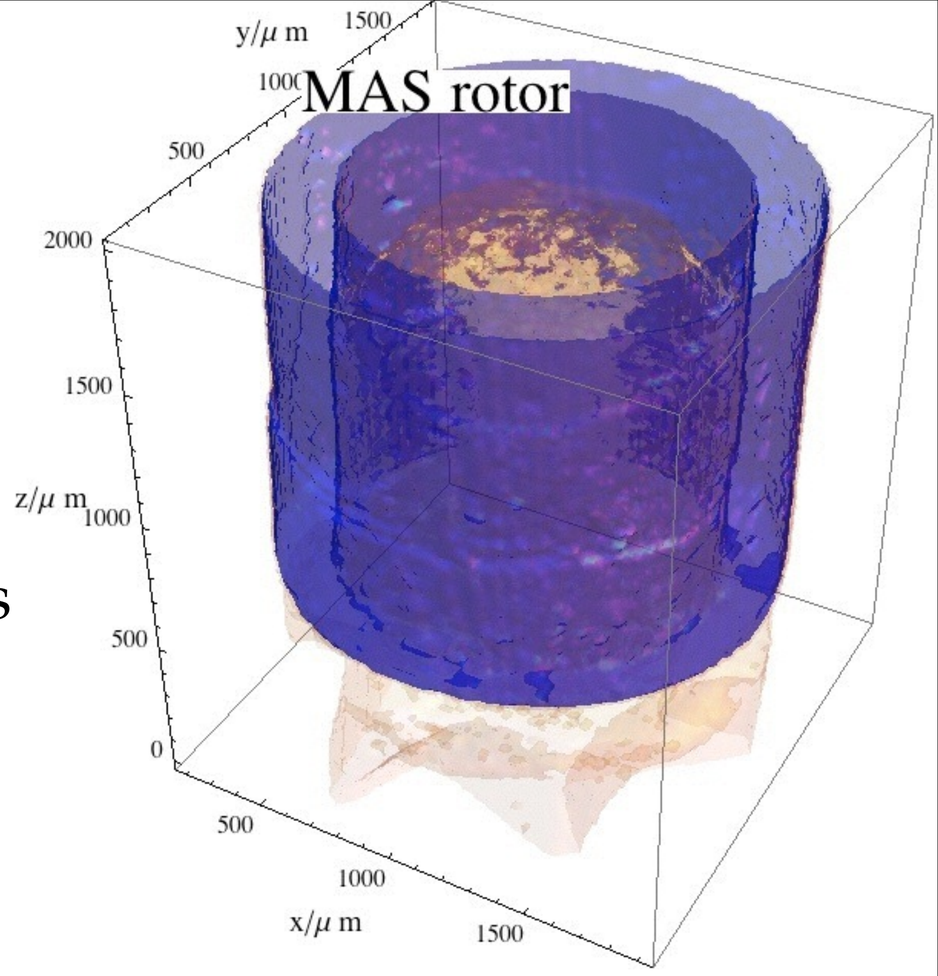
Part 4 Find all Zirconia and Kel-F in the *.h5 volume

- **Step 1: Import cropped volume from an *.h5 files. [[150;;500,150;;500,140;;500]]**
- **Step 2: Scan through some rows to get good values for zirconia and Kel-F (gaussian fit)**
- **Step 5: Function to find zirconia and Kel-F using interpolated starting values for the coefficients.**
  HW task: Improve the code highlighted in light blue.
- **Step 7: Build a segmented volume of Zirconia and Kel-F**
  HW task: Improve the code highlighted in light blue.
- **Exploratory workspace. Interesting rows are {1,50,100,106,110,120,300,351}**



Mathematica commands - image processing

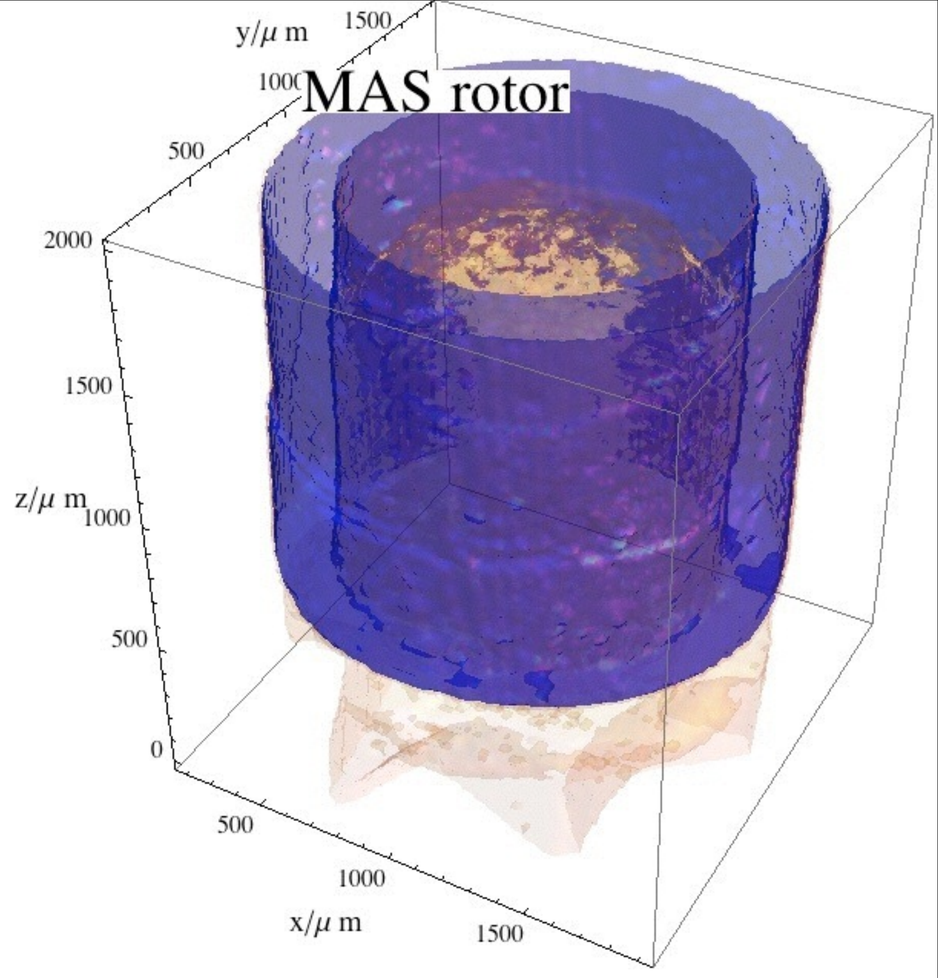
MorphologicalBinarization - a binarization
GeometricMeanFilter - a median filter
TotalVariationFilter - noise reducing filter
ImageMultiply -
ImageSubtract -
ImageData - convert image format to array of numbers
MorphologicalComponents - makes a label field
ComponentMeasurements - inspects a label field
Erosion
Closing
ImageAdjust



NOTE: Know these commands. See Mathematica help files.

Mathematica commands - list commands

- Position
- Ordering
- Reverse
- First, Rest
- ConstantArray
- Length
- Dimensions
- Flatten
- Append



Mathematica commands - set commands

- Intersection

NOTE: Know these commands. See Mathematica help files.

HW7_Segmentation_MAS_Rotor.nb

Run Steps 1, 2, and 5 (taken without number change from previous MAS Rotor program)

- **Step 1: Import cropped volume from an *.h5 files. [[150;;500,150;;500,140;;500]]**
- **Step 2: Scan through some rows to get good values for zirconia and Kel-F (gaussian fit)**
- **Step 5: Function to find zirconia and Kel-F using interpolated starting values for the coefficients.**

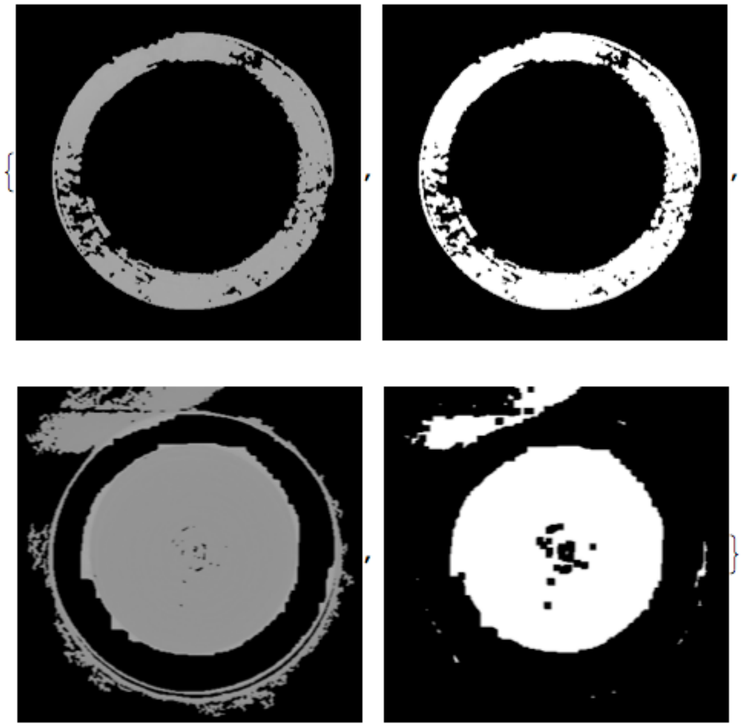
HW task: Improve the code highlighted in light blue.

```
funcMakeZirconiaAndKelFImages[indexRow_, numberColumns_, numberSlices_] := Module[{},
  segmentedSlice = ConstantArray[0, {numberColumns, numberSlices}];
  thresholdCenter = Interpolation[allParametersZirconia[[All, {1, 2}]]][indexRow];

  slice = volMAS[[indexRow, All, All]];
  image = Image[slice, "Bit16"];
  imageTV = TotalVariationFilter[GeometricMeanFilter[image, 1], 0.35, Method → "Laplacian"];
  imageZirconiaBinary = MorphologicalBinarize[imageTV, {0.0, 0.02} + thresholdCenter];
  imageZirconia = ImageMultiply[imageTV, imageZirconiaBinary];

  thresholdCenter = Interpolation[allParametersKelF[[All, {1, 2}]]][indexRow];
  slice = volMAS[[indexRow, All, All]];
  image = Image[slice, "Bit16"];
  imageTV = TotalVariationFilter[GeometricMeanFilter[image, 1], 0.35, Method → "Laplacian"];
  If[indexRow ≥ 106,
   imageTVSubtract = ImageSubtract[imageTV, Closing[imageZirconiaBinary, 10]],
   imageTVSubtract = imageTV];
  imageKelFBinary = MorphologicalBinarize[imageTVSubtract, {0.0, 0.02} + thresholdCenter];
  imageKelF = ImageMultiply[imageTVSubtract, imageKelFBinary];
  (*Print[{image,imageTVSubtract,imageLowBinary,imageLow}]; *)

  imageKelFBinary = Erosion[imageKelFBinary, 3];
  imageKelFBinary = Closing[imageKelFBinary, 1];
  (* Print[{imageKelF,imageKelFBinary}]; *)
 ]
```

```
funcMakeZirconiaAndKelFImages[120, columns, slices];
{imageZirconia, imageZirconiaBinary, imageKelF, imageKelFBinary}
```

imageZirconia

imageZirconiaBinary



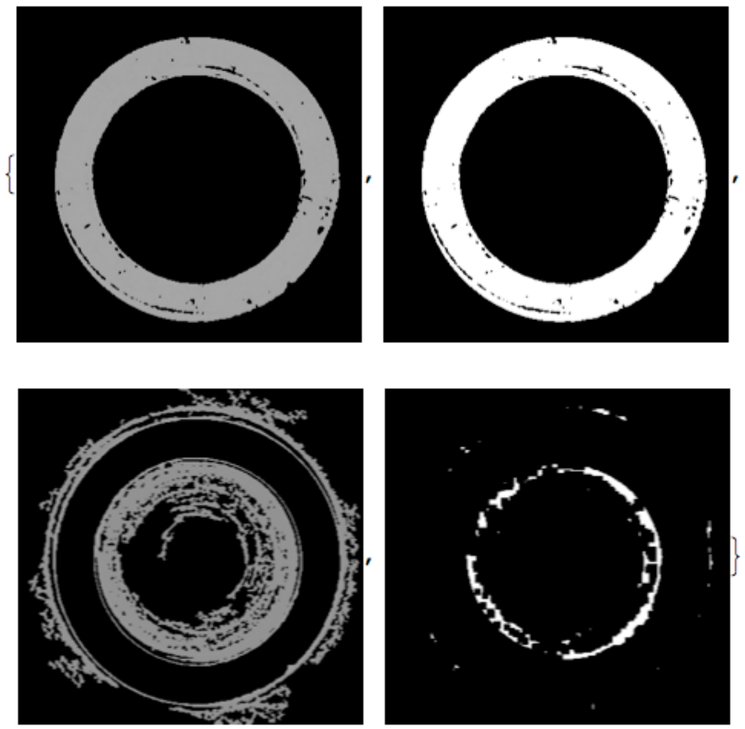
imageKelF

imageKelFBinary

Recommendation: Do code testing in section “Exploratory workspace”.
Same code as Step 5, but taken out of the function/module structure.

- **Exploratory workspace. Interesting rows are {1,50,100,106,110,120,300,351}**

```
indexRow = 351;

thresholdCenter = Interpolation[allParametersZirconia[[All, {1, 2}]]][indexRow];

slice = volMAS[[indexRow, All, All]];
image = Image[slice, "Bit16"];
imageTV = TotalVariationFilter[GeometricMeanFilter[image, 1], 0.35, Method → "Laplacian"];
imageZirconiaBinary = MorphologicalBinarize[imageTV, {0.0, 0.02} + thresholdCenter];
imageZirconia = ImageMultiply[imageTV, imageZirconiaBinary];

thresholdCenter = Interpolation[allParametersKelF[[All, {1, 2}]]][indexRow];

slice = volMAS[[indexRow, All, All]];
image = Image[slice, "Bit16"];
imageTV = TotalVariationFilter[GeometricMeanFilter[image, 1], 0.35, Method → "Laplacian"];
If[indexRow ≥ 106,
  imageTVSubtract = ImageSubtract[imageTV, Closing[imageZirconiaBinary, 10]],
  imageTVSubtract = imageTV];
imageKelFBinary = MorphologicalBinarize[imageTVSubtract, {0.0, 0.02} + thresholdCenter];
imageKelF = ImageMultiply[imageTVSubtract, imageKelFBinary];

imageKelFBinary = Erosion[imageKelFBinary, 3];
imageKelFBinary = Closing[imageKelFBinary, 1];

{imageZirconia, imageZirconiaBinary, imageKelF, imageKelFBinary}
```



Recommendation: Do code testing in section “Exploratory workspace”.
Same code as Step 5, but taken out of the function/module structure.

- **Exploratory workspace. Interesting rows are {1,50,100,106,110,120,300,351}**

```
indexRow = 351;

thresholdCenter = Interpolation[allParametersZirconia[[All, {1, 2}]]][indexRow];

slice = volMAS[[indexRow, All, All]];
image = Image[slice, "Bit16"];
imageTV = TotalVariationFilter[GeometricMeanFilter[image, 1], 0.35, Method → "Laplacian"];
imageZirconiaBinary = MorphologicalBinarize[imageTV, {0.0, 0.02} + thresholdCenter];
imageZirconia = ImageMultiply[imageTV, imageZirconiaBinary];

thresholdCenter = Interpolation[allParametersKelF[[All, {1, 2}]]][indexRow];

slice = volMAS[[indexRow, All, All]];
image = Image[slice, "Bit16"];
imageTV = TotalVariationFilter[GeometricMeanFilter[image, 1], 0.35, Method → "Laplacian"];
If[indexRow ≥ 106,
  imageTVSubtract = ImageSubtract[imageTV, Closing[imageZirconiaBinary, 10]],
  imageTVSubtract = imageTV];
imageKelFBinary = MorphologicalBinarize[imageTVSubtract, {0.0, 0.02} + thresholdCenter];
imageKelF = ImageMultiply[imageTVSubtract, imageKelFBinary];

imageKelFBinary = Erosion[imageKelFBinary, 3];
imageKelFBinary = Closing[imageKelFBinary, 1];
```

```
mageKelFBinary}
```

1) read a slice based on index=row
2) median filter, 1 pixel
3) total variation filter, 0.35
4) binarize (double threshold)
   imageZirconiaBinary
5) image x binary image (mask)
   imageZirconia
6) read a slice based on index=row
7) median filter, 1 pixel
8) total variation filter, 0.35
9) If both zirconia and Kel-F
   10) closing(imageZirconiaBinary)
   11) subtraction
12) binarize (double threshold)
13) image x binary image (mask)
   imageKelF
14) erode, close imageKelFBinary

Old code

1) read a slice based on index=row
2) median filter, 1 pixel
3) total variation filter, 0.35
4) binarize (double threshold)
   imageZirconiaBinary
5) image x binary image (mask)
   imageZirconia

6) read a slice based on index=row
7) median filter, 1 pixel
8) total variation filter, 0.35
9) If both zirconia and Kel-F
   10) closing(imageZirconiaBinary)
   11) subtraction
12) binarize (double threshold)
13) image x binary image (mask)
   imageKelF
14) erode, close imageKelFBinary

Some Ideas

a) Why no closing between #3 and #4?
b) Replace #11 subtraction with multiply?
c) Replace/modify/enhance median, total variation filter (#2,3 and #7,8)

d) use the binary images as marker images for watershed segmentation

e) perform a distance transform on binary images and use that as marker image for watershed segmentation

f) use edge detection

g) instead of pulling a slice out of index=row, try a slice from index=column.

Segmentation Analysis

Mathematica includes a variety of image segmentation techniques such as clustering, watershed, region growing, and level set as well as a rich set of functions for post-processing and analyzing the result of the segmentation.

Image Preparation

ColorQuantize — reduce the number of distinct colors in an image

FillingTransform — reduce noise to create smooth regions in an image

GradientFilter, **RangeFilter** — create an edge map from an image

FindThreshold ▪ **Threshold** ▪ **ImageClip**

Binary Segmentation

Binarize — segmentation by thresholding pixel intensities

MorphologicalBinarize ▪ **RegionBinarize** ▪ **ChanVeseBinarize**

Segmentation

ArrayComponents — find identical components

MorphologicalComponents — find morphologically connected components

ImageForestingComponents — image segmentation using various methods

ClusteringComponents — segmentation based on cluster analysis

WatershedComponents — segmentation based on watershed methods

Component Analysis

ComponentMeasurements — shape and color analysis

SelectComponents ▪ **DeleteSmallComponents** ▪ **DeleteBorderComponents**

Colorize — color every segment differently

MORE ABOUT

- Image Processing & Analysis
- Image Filtering & Neighborhood Processing
- Morphological Image Processing

Mathematical Morphology

Combining methods from set theory, topology, and discrete mathematics, mathematical morphology provides a powerful approach to processing images and other discrete data. *Mathematica* includes an extensive and efficient implementation of mathematical morphology, fully integrated with *Mathematica*'s general image and data processing.

Image Preparation

Binarize — convert any image to black and white

ColorNegate — flip black and white

Basic Operations

Dilation ▪ **Erosion** ▪ **Opening** ▪ **Closing**

Morphological Transforms

DistanceTransform ▪ **InverseDistanceTransform** ▪ **HitMissTransform** ▪ **TopHatTransform** ▪ **BottomHatTransform**

MinDetect ▪ **MaxDetect** ▪ **FillingTransform**

MorphologicalTransform — general block-based binary morphological operation

MorphologicalGraph — generate a graph from an image skeleton

Morphological Analysis

GeodesicDilation ▪ **GeodesicErosion** ▪ **SkeletonTransform** ▪ **Thinning** ▪ **Pruning** ▪ **MorphologicalBranchPoints**

MorphologicalEulerNumber ▪ **MorphologicalPerimeter**

MorphologicalComponents — identify connected components

CornerNeighbors — option to specify neighborhood configuration

Component Analysis

ComponentMeasurements — component shape and color analysis

SelectComponents ▪ **DeleteSmallComponents** ▪ **DeleteBorderComponents**

Colorize — color every component differently

MORE ABOUT

- General Image Processing
- Segmentation Analysis
- Neighborhood Processing
- List Manipulation

Image Filtering & Neighborhood Processing

Mathematica not only includes highly optimized implementations of standard image processing filters, but also uses its general symbolic architecture to allow arbitrarily sophisticated filtering and neighborhood processing strategies to be set up using the full mathematical and algorithmic power of *Mathematica*.

Linear Filters

Blur, **Sharpen** — blur, sharpen over a range

GaussianFilter — Gaussian and Gaussian derivatives filter

GradientFilter ▪ **LaplacianGaussianFilter** ▪ **LaplacianFilter** ▪ **MeanFilter** ▪ **WienerFilter**

ImageConvolve, **ImageCorrelate** — general linear convolution, correlation

DerivativeFilter — general-order derivative filter

Nonlinear Filters

MedianFilter ▪ **MinFilter** ▪ **MaxFilter** ▪ **CommonestFilter** ▪ **RangeFilter**

EntropyFilter ▪ **StandardDeviationFilter** ▪ **HarmonicMeanFilter** ▪ **GeometricMeanFilter** ▪ **KuwaharaFilter**

BilateralFilter ▪ **MeanShiftFilter**

PeronaMalikFilter ▪ **CurvatureFlowFilter**

Nonlocal Filters

ImageDeconvolve ▪ **TotalVariationFilter**

Region-of-Interest Processing

Masking — specify the image or graphics description of the region to which filters will be applied

General Neighborhood Processing

ImageFilter — apply an arbitrary function to blocks of pixel values

Convolution Kernels »

DiskMatrix ▪ **BoxMatrix** ▪ **DiamondMatrix** ▪ **CrossMatrix** ▪ **GaussianMatrix** ▪ **...**

Image Tiling & Compositing

ImagePartition — partition an image into an array of subimages

ImageAssemble — assemble an image from an array of subimages

ImageCompose — overlay, alpha-blend or compose images

List-Based Operations »

ImageData — extract an array of data from an image

Partition — generalized partitioning

Image Processing & Analysis

Mathematica provides broad and deep built-in support for both programmatic and interactive modern industrial-strength image processing—fully integrated with *Mathematica*'s powerful mathematical and algorithmic capabilities. *Mathematica*'s unique symbolic architecture and notebook paradigm allow images in visual form to be included and manipulated directly both interactively and in programs.

Creating & Importing Images

Copy, Drag/Drop — copy and paste an image directly into a notebook

Import — programmatically import any standard format (TIFF, PNG, JPEG, DICOM, ...)

Image — create an image from an array of data, and represent any multichannel image

Rasterize — convert expressions, notebooks, or any *Mathematica* object to raster form

CurrentImage — capture an image or video in real time from a camera or other device

ImageCapture — open a graphical user interface for capturing images

RandomImage — create an image from a symbolic distribution

Image Representation »

ImageData — extract the array of raster data from an image

ImageDimensions ▪ **ImageChannels** ▪ **ImageType** ▪ **ImageHistogram** ▪ **...**

Thumbnail — display an image in thumbnail form

Basic Image Manipulation »

ImageCrop ▪ **ImagePad** ▪ **ImageTake** **BorderDimensions** ▪ **...**

ImageResize ▪ **ImageRotate** ▪ **ImageReflect**

ImageAdjust — adjust levels, brightness, contrast, gamma, etc.

Sharpen ▪ **Blur** ▪ **Lighter** ▪ **Darker**

ImageEffect — special image and photographic effects

Inpaint — retouch parts of an image

Image Geometry »

ImageTransformation ▪ **ImagePerspectiveTransformation** ▪ **...**

Color Manipulation »

Colorize ▪ **ColorConvert** ▪ **ColorSeparate** ▪ **ColorQuantize** ▪ **...**

Filtering & Neighborhood Processing »

ImageFilter ▪ **ImageConvolve** ▪ **ImageCorrelate**

GaussianFilter ▪ **LaplacianFilter** ▪ **DerivativeFilter**

MeanFilter ▪ **MedianFilter** ▪ **BilateralFilter** ▪ **PeronaMalikFilter** ▪ **...**

MinFilter ▪ **MedianFilter** ▪ **GradientFilter** ▪ **EntropyFilter** ▪ **WienerFilter** ▪ **BilateralFilter** ▪ **...**

Feature Detection

Using a variety of state-of-the-art methods, *Mathematica* provides immediate functions for detecting and extracting features in images and other arrays of data. *Mathematica* supports specific geometrical features such as edges and corners, as well as general keypoints that can be used to register and compare images.

Edge and Line Detection

EdgeDetect — detect edges in an image using Canny and other methods

CrossingDetect, **ContourDetect** — detect zeros and zero crossings

ImageLines — find Hough lines in an image

GradientFilter ▪ **LaplacianGaussianFilter** ▪ **ImageFilter** ▪ **ImageConvolve**

Points-of-Interest Detection

ImageKeypoints — find keypoints and associated feature vectors in an image

CornerFilter — compute a measure for the presence of a corner

ImageCorrespondingPoints — find corresponding keypoints in pairs of images

---

Region Detection

MinDetect, **MaxDetect** — detect regional and extended minima and maxima

---

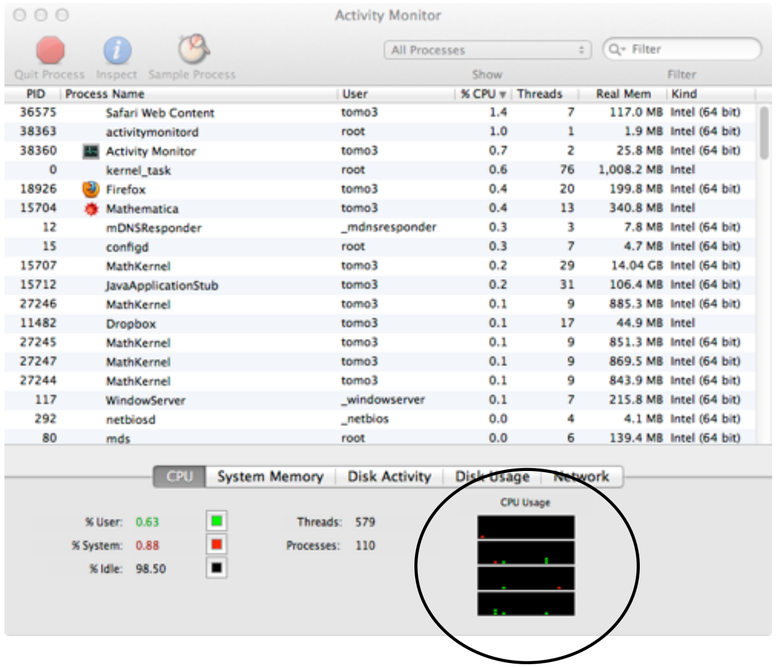
Radon, **InverseRadon** — Radon and inverse Radon transforms

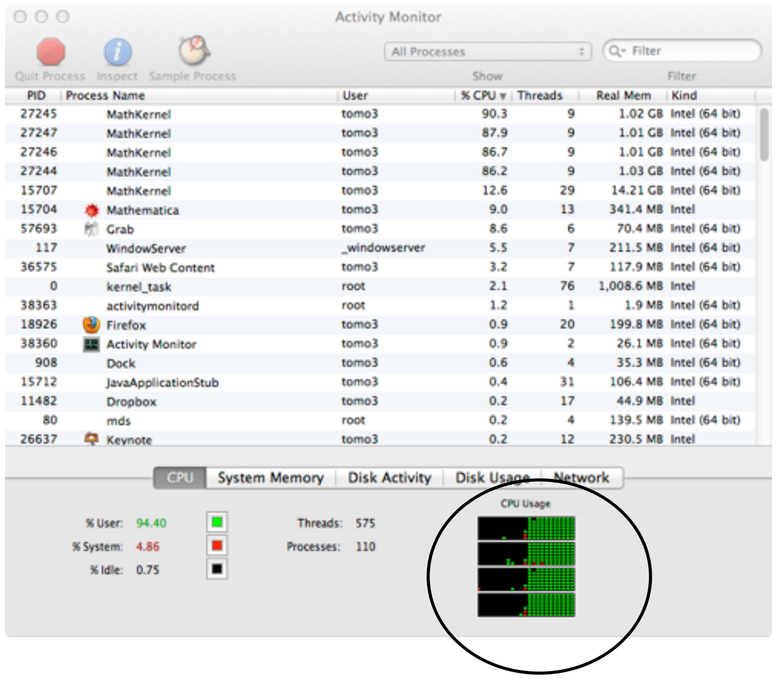
TextRecognize — extract characters from an image

MORE ABOUT

- Image Processing & Analysis
- Image Filtering & Neighborhood Processing
- Morphological Image Processing
- Segmentation Analysis

Parallelization





- **Step 7: Build a segmented volume of Zirconia and Kel-F**

HW task: Improve the code highlighted in light blue.

```
= {rows, columns, slices} = Dimensions[volMAS]
  segmentedVolume = ConstantArray[0, {rows, columns, slices}];
  segmentedSlice = ConstantArray[0, {columns, slices}];

= {351, 351, 361}

= SetSharedVariable[segmentedVolume]
  Timing[
   ParallelDo[Module[{},
     funcMakeZirconiaAndKelFImages[indexRow, columns, slices];

     If[indexRow ≥ 106, Module[{},
       componentsZirconia = MorphologicalComponents[Closing[imageZirconiaBinary, 5]];
       zirconiaAreas = ComponentMeasurements[componentsZirconia, "Area"];
       indexBestZirconia = First[Reverse[Ordering[zirconiaAreas[[All, 2]]]]];
       labelBestZirconiaComponent = zirconiaAreas[[indexBestZirconia, 1]];
       coordinatesZirconiaPixels = Position[componentsZirconia,
```

```
(kernel 4) 1
(kernel 3) 19
(kernel 2) 37
(kernel 1) 55
(kernel 4) 2
(kernel 3) 20
(kernel 2) 38
(kernel 1) 56
(kernel 4) 3
(kernel 3) 21
(kernel 2) 39
(kernel 1) 57
```