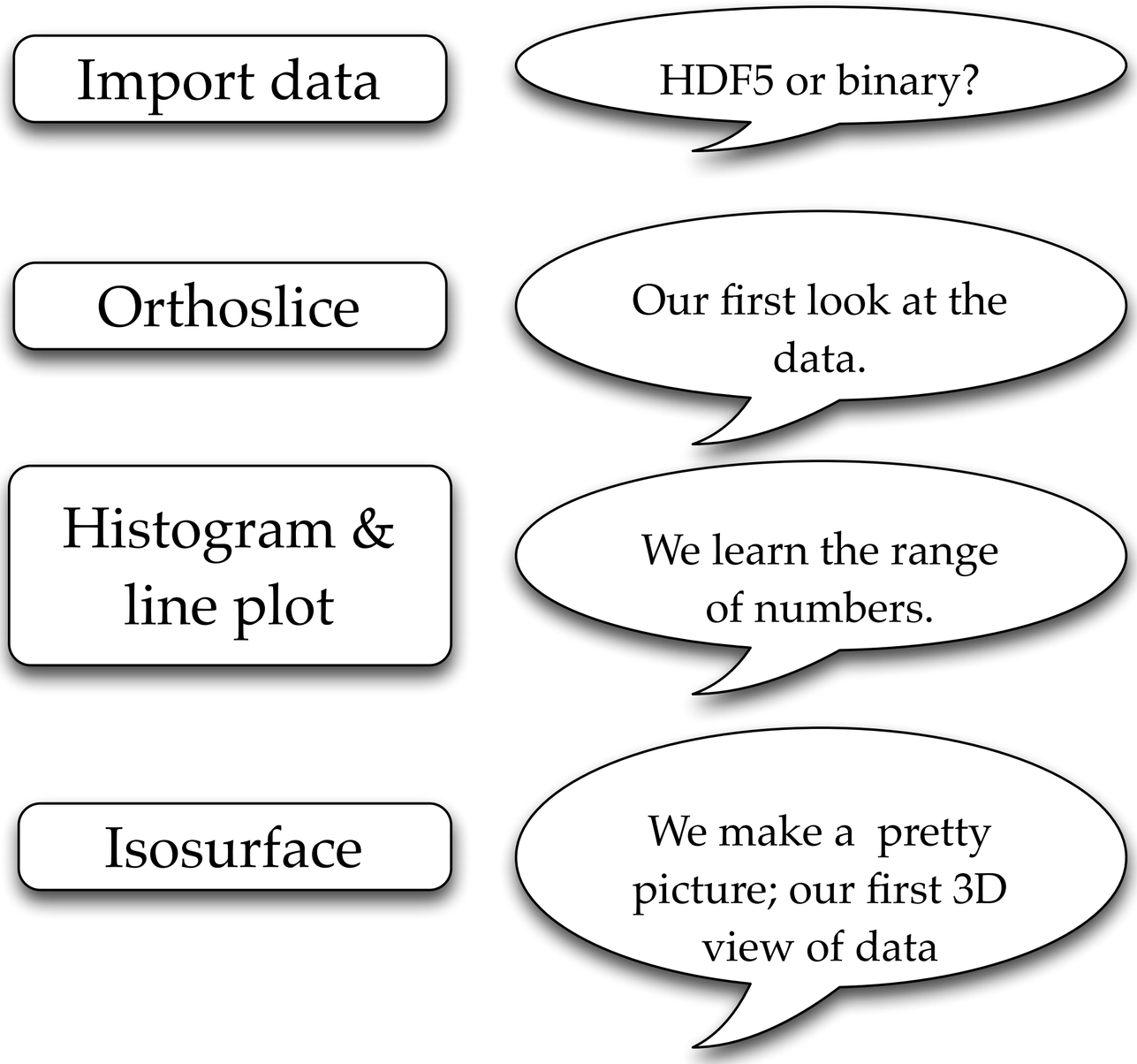
L5 Workflows, Manipulate, and Transforms (distance, gradient, watershed)
1 February, 2012

Use data set MAS_rotor_slice.h5  and bullet

1) Download Moodle  Week 3/MAS_rotor_slice.h5
2) Download Moodle  Week 3/Pgm5_manipulate_distance_watershed.nb
3) Download Moodle  Week 1/bullet dataset  (either bin or HDF5)

4) Launch ImageJ and Mathematica

# A standard workflow




We have done once through:

- ✓ ImageJ
- ✓ Mathematica
- ✓ Avizo

How much more practice is needed? Some HWs?

- ➡ ImageJ
- ➡ Mathematica
- ➡ Avizo

# Another workflow

Import data

Binarize

Erode

Close holes

Dilate

Connected Component Analysis

Morphological Analysis

This makes a label field.

From label field, we can count objects, sort by size, etc.

We have done once through:

- ✓ ImageJ
- ✓ Mathematica
- ✓ Avizo

How much more practice is needed? Some HWs?

- ➡ ImageJ
- ➡ Mathematica
- ➡ Avizo

# Another workflow

Import data

Binarize

Erode

Close holes

Dilate

Distance Transform

Gradient Transform

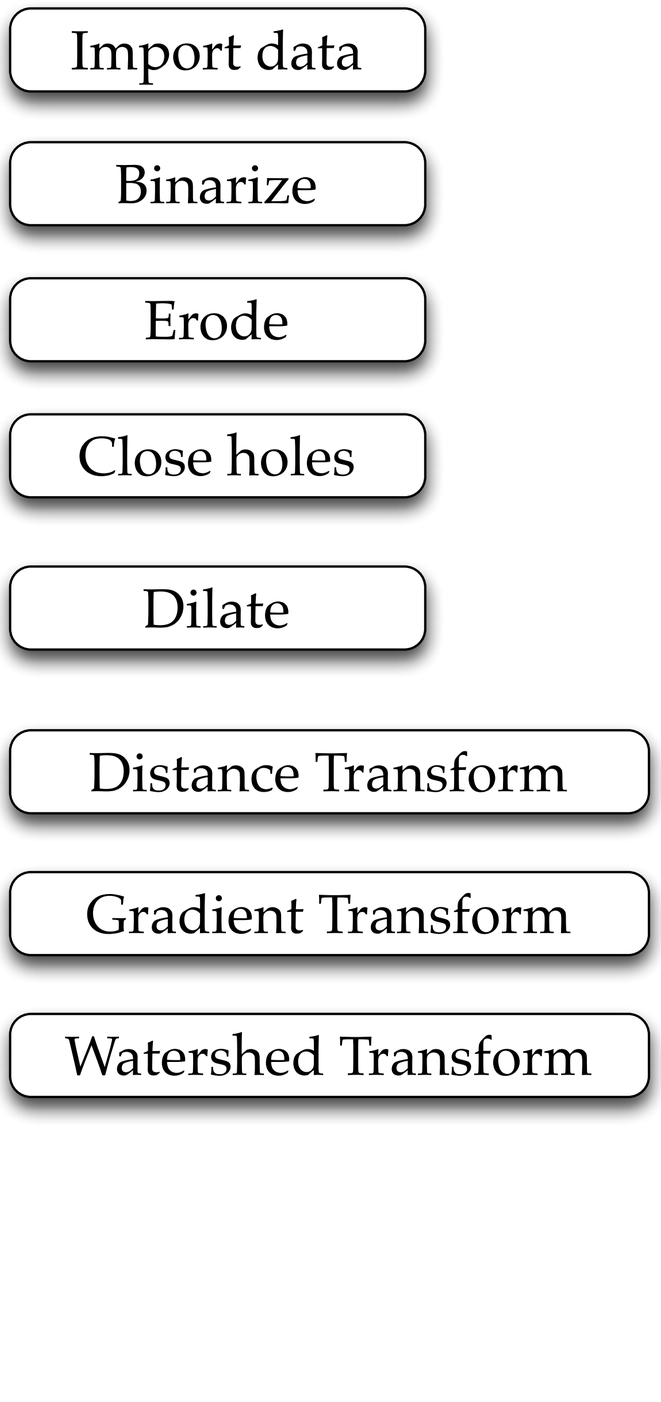
Watershed Transform

Connected Component Analysis

Morphological Analysis

Today (L5), a Mathematica discussion of distance, gradient, and watershed transforms. These are good for grain separations.

# Another workflow

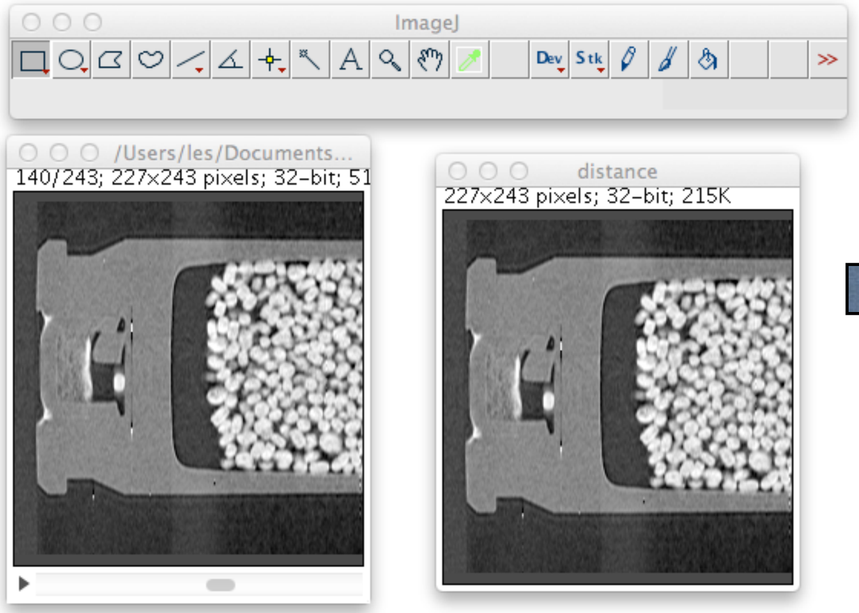


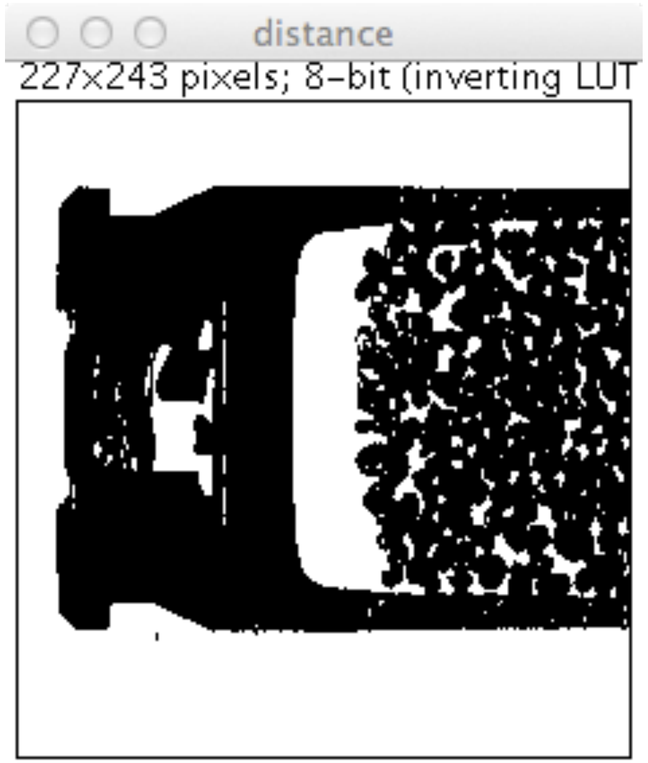

Today (L5), a Mathematica discussion of the Manipulate command.
Allows better user interaction for commands like binarize, erode, etc.

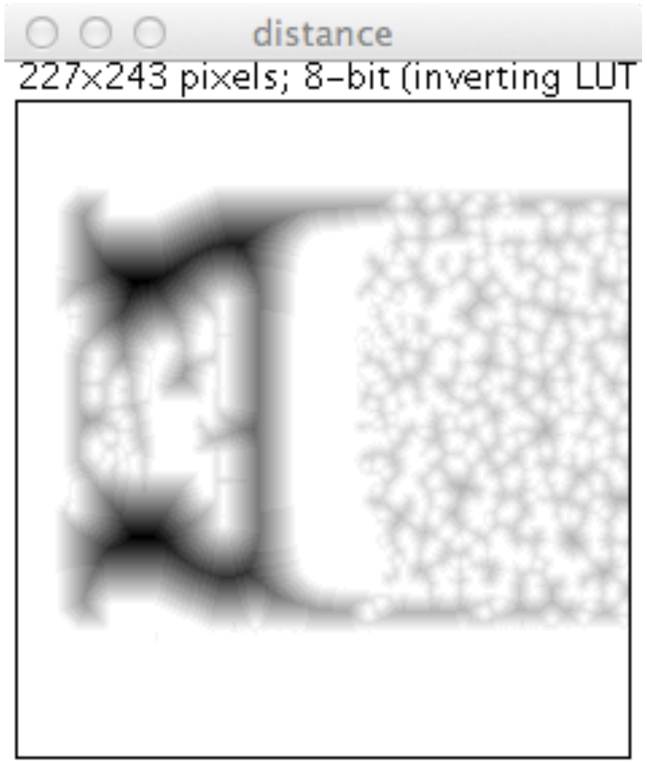
Note: Before next Monday, 6 Feb, please watch the Mathematica video “Manipulate” listed in Moodle, Week 4.

In ImageJ:  the **distance** transform

1) Duplicated one slice and name it “distance”
2) Process / Binary / Make Binary
3) Process / Binary / Distance Map

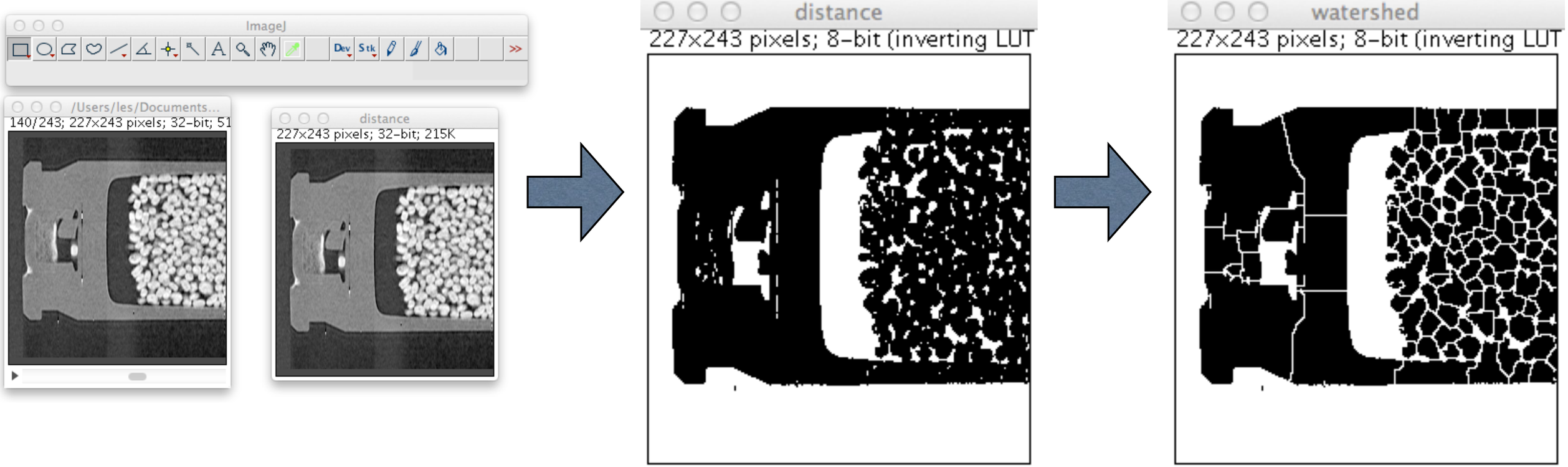


In ImageJ:  the **watershed** transform

1) Duplicated one slice and name it “watershed”
2) Process / Binary / Make Binary
3) Process / Binary / Watershed



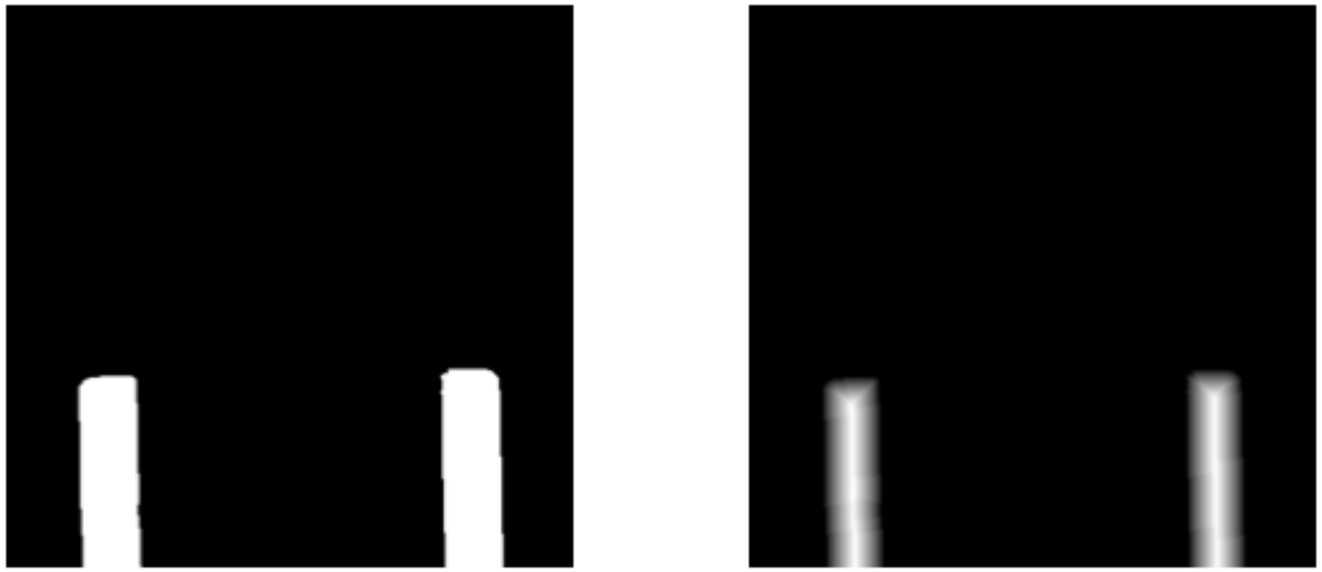
# Pgm5_distance_watershed.nb
# 1 Feb 2012
# Les Butler

MAS_rotor.h5 {400x400x400 x Real32}
MAS_rotor_slice.h5 {400x400x Real32}

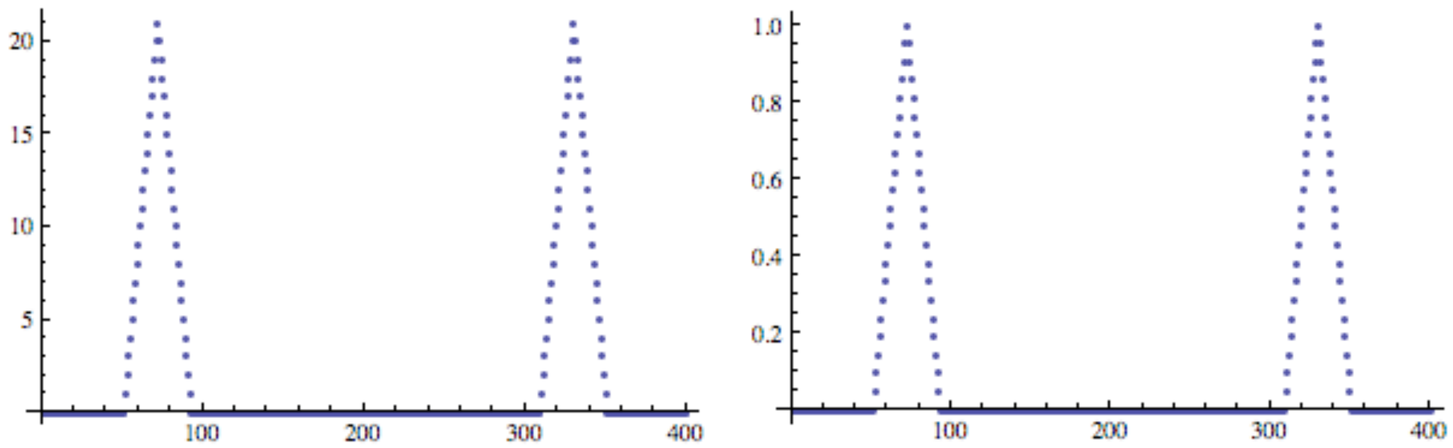
- **Step 1: Import MAS_Rotor_slice.h5 and plot**
- **Step 2: Convert to image format, binarize, erode, close, dilate**
- **Step 3: Distance Transform (without ImageAdjust)**
- **Step 4: Distance Transform (with ImageAdjust)**
- **Step 5: Distance Transform (compare with and without ImageAdjust)**
- **Step 6: Negate the result of the distance transform**
- **Step 7: Let's apply gradient to the imageDilate**
- **Step 8: Let's apply watershed to the gradient of the imageDilate**
- **Step 9: Our first experience with Manipulate**

## ■ Step 5: Distance Transform (compare with and without ImageAdjust)

```
In[736]:= gWithout = DistanceTransform[imageDilate] ;
          gWith = ImageAdjust[DistanceTransform[imageDilate] ];
          dataWithout = ImageData[DistanceTransform[imageDilate] , "Real"];
          gDataWithout = ListPlot[dataWithout[[300, All]], PlotRange → {All, All} ];
          dataWith = ImageData[ImageAdjust[DistanceTransform[imageDilate]] , "Real"];
          gDataWith = ListPlot[dataWith[[300, All]], PlotRange → {All, All} ];
          GraphicsGrid[{{gWithout, gWith}, {gDataWithout, gDataWith}}, ImageSize → 600]
```



Out[742]=




## DistanceTransform

*Updated in* 8

Show changes ⊞

`DistanceTransform[`*image*`]`
gives the distance transform of *image*, in which the value of each pixel is replaced by its distance to the nearest background pixel.

`DistanceTransform[`*image*`, `*t*`]`
treats values above *t* as foreground.

Distance transform of a binary image:

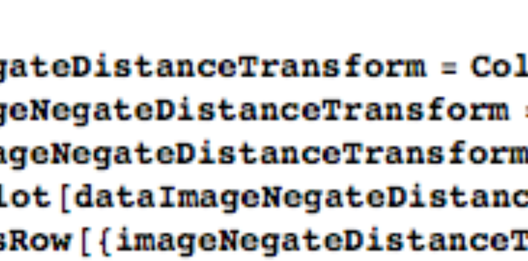
```
In[1]:= DistanceTransform[ ] // ImageAdjust
```

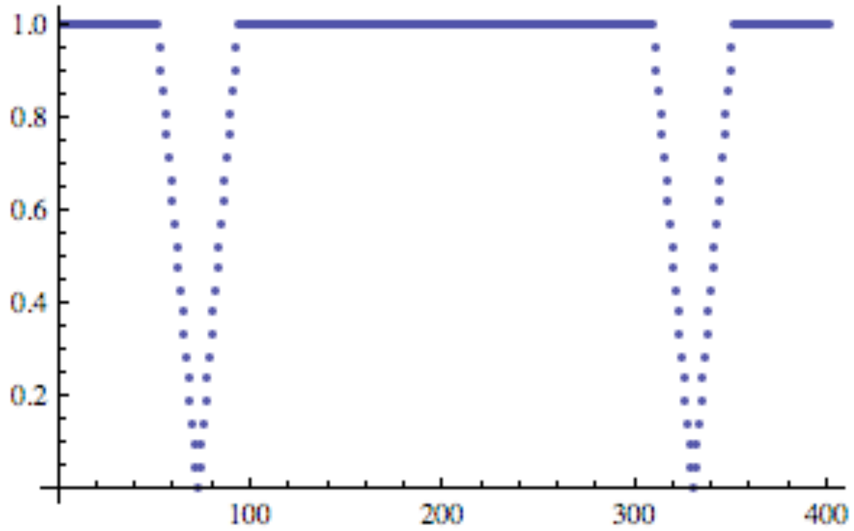
Out[1]=

## ■ Step 6: Negate the result of the distance transform

In[746]:=
```
imageNegateDistanceTransform = ColorNegate[ImageAdjust[DistanceTransform[imageDilate] ]];
dataImageNegateDistanceTransform = ImageData[imageNegateDistanceTransform, "Real"];
gDataImageNegateDistanceTransform =
  ListPlot[dataImageNegateDistanceTransform[[300, All]], PlotRange → {All, All} ];
GraphicsRow[{imageNegateDistanceTransform, gDataImageNegateDistanceTransform}, ImageSize → 600]
```

Out[749]=



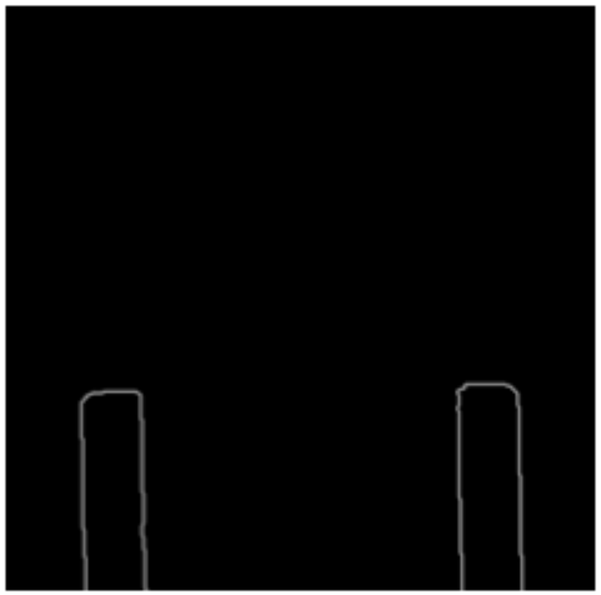


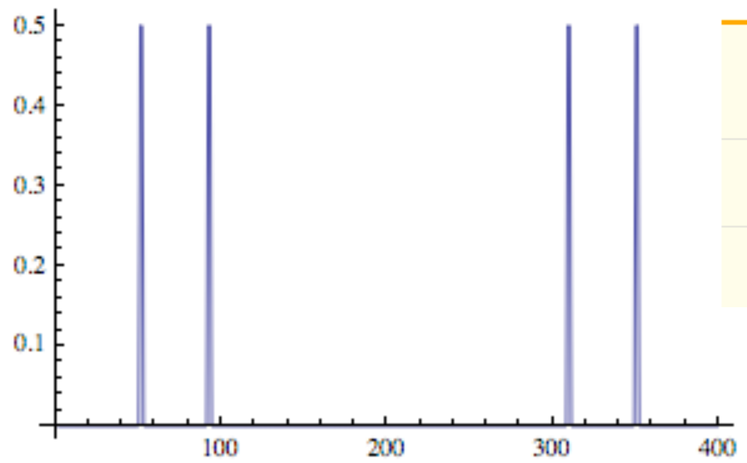

Conceptionally, the combination of the distance and negation transforms leads into the watershed transform.

## ■ Step 7: Let's apply gradient to the imageDilate

```
9]:= imageGradient = GradientFilter[imageDilate, 1] ;
     dataImageGradient = ImageData[imageGradient, "Real"];
     gDataImageGradient = ListPlot[dataImageGradient[[300, All]], Joined → True, PlotRange → {All, All} ];
     GraphicsRow[{imageGradient, gDataImageGradient}, ImageSize → 600]
```

12]=






# GradientFilter

*Updated* in 8

Show changes

`GradientFilter[`*image*`, r]`
gives an image corresponding to the magnitude of the gradient of *image*, computed using discrete derivatives of a Gaussian of pixel radius *r*.

`GradientFilter[`*image*`, {r, σ}]`
uses a Gaussian with standard deviation *σ*.

`GradientFilter[`*image*`, {{r1, r2}, ...}]`
uses a Gaussian with radii $r_i$ etc. in vertical and horizontal directions.

## ▼ Basic Examples (3)

Gradient filtering of a multichannel image:

```
In[1]:= GradientFilter[ <image> , 2] // ImageAdjust
```

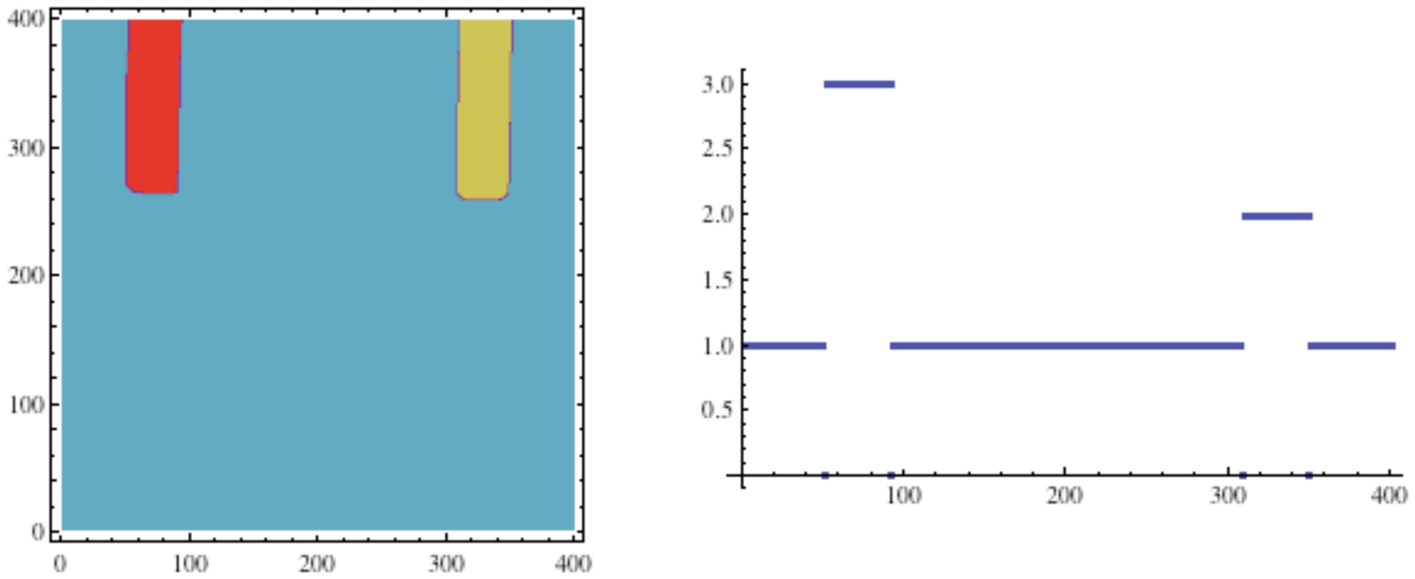
Out[1]=

## ■ Step 8: Let's apply watershed to the gradient of the imageDilate

```
In[821]:= dataWatershed = WatershedComponents[GradientFilter[imageDilate, 1] ];
          {Min[dataWatershed], Max[dataWatershed]}
          gListDensityPlot = ListDensityPlot[dataWatershed, ColorFunction → "Rainbow",
             PlotRange → {All, All, All}];
          gListPlot = ListPlot[dataWatershed[[300, All]], PlotRange → {All, All} ];
          GraphicsRow[{gListDensityPlot, gListPlot}, ImageSize → 600]
```

Out[822]= {0, 3}

Out[825]=




# WatershedComponents

*New in* 8

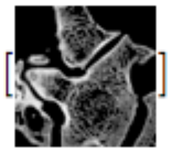
`WatershedComponents[`*image*`]`
computes the watershed transform of *image*, returning the result as a matrix in which positive integers label the catchment basins.

`WatershedComponents[`*image*`, `*marker*`]`
uses a binary image *marker* to indicate regions where basins may be created.

### ▼ Basic Examples (3)

Use watershed contours to illustrate the trabecular bone structure:

```
In[1]:= WatershedComponents[ ] // Image
```



Out[1]=

■ **Step 9: Our first experience with Manipulate**

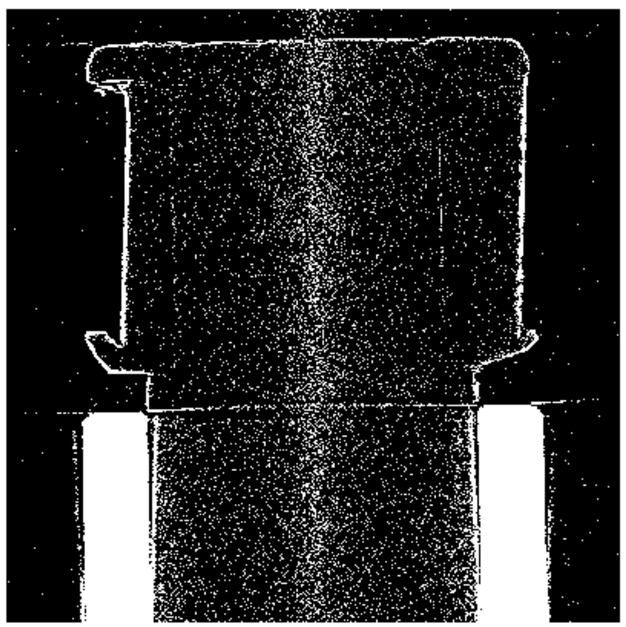
```
In[826]:= imageBinary = Binarize[imageSlice, 29000/65535]
```

Out[826]=

```
In[827]:= Manipulate[Binarize[imageSlice, numerator/65535], {numerator, 0, 65535}]
```

numerator

Out[827]=



# Manipulate

**Manipulate**[*expr*, {*u*, $u_{min}$, $u_{max}$}]
generates a version of *expr* with controls added to allow interactive manipulation of the value of *u*.

**Manipulate**[*expr*, {*u*, $u_{min}$, $u_{max}$, *du*}]
allows the value of *u* to vary between $u_{min}$ and $u_{max}$ in steps *du*.

**Manipulate**[*expr*, {{*u*, $u_{init}$}, $u_{min}$, $u_{max}$, ...}]
takes the initial value of *u* to be $u_{init}$.

**Manipulate**[*expr*, {{*u*, $u_{init}$, $u_{lbl}$}, ...}]
labels the controls for *u* with $u_{lbl}$.

**Manipulate**[*expr*, {*u*, {$u_1$, $u_2$, ...}}]
allows *u* to take on discrete values $u_1$, $u_2$, ....

**Manipulate**[*expr*, {*u*, ...}, {*v*, ...}, ...]
provides controls to manipulate each of the *u*, *v*, ....

**Manipulate**[*expr*, "$c_u$" -> {*u*, ...}, "$c_v$" -> {*v*, ...}, ...]
links the controls to the specified controllers on an external device.

Manipulate a continuous parameter:

```
In[1]:= Manipulate[Plot[Sin[x (1 + a x)], {x, 0, 6}], {a, 0, 2}]
```

Out[1]=