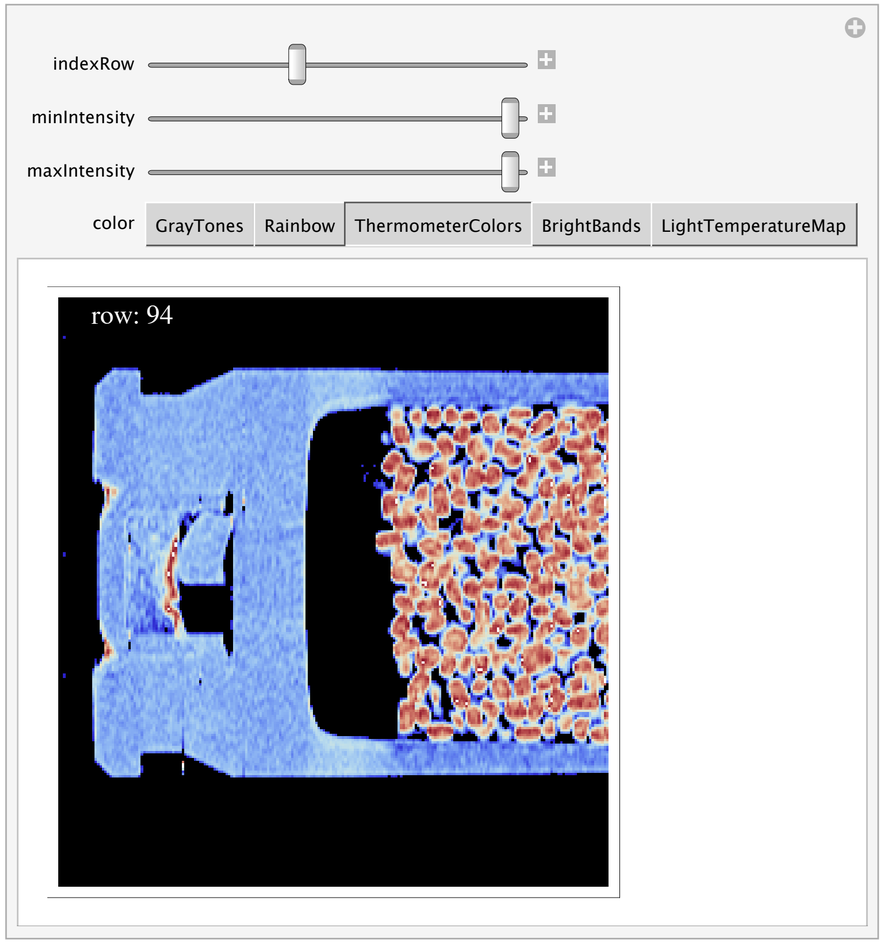
L6 Manipulate, Orthoslice viewer

1) Download Moodle/Week 2/bullet dataset (HDF5)
2) Download Moodle/Week 4/Pgm6_Manipulate_Orthoslice.nb

This lecture starts HW4 Manipulate, due Wednesday, 15 Feb.

Out[415]=



Philosophy:

The lecture of 3 Feb showed grain segmentation using the software Avizo Fire. There are commands (3D watershed) in Avizo Fire that are not present in Avizo Standard, ImageJ, or Mathematica. What to do?

1) Buy more tokens from Avizo so more simultaneous Avizo Fire sessions? We've asked Le Yan to get a quote, then we write proposal, then buy it. Might be purchased in about a year.
2) Search for ImageJ plugins for 3D watershed?
3) Write Mathematica code for 3D watershed?

ListDensityPlot

- **Step 2: Plot a slice with ListDensityPlot. Note the time required to make a plot.**

```
Options[ListDensityPlot]
```

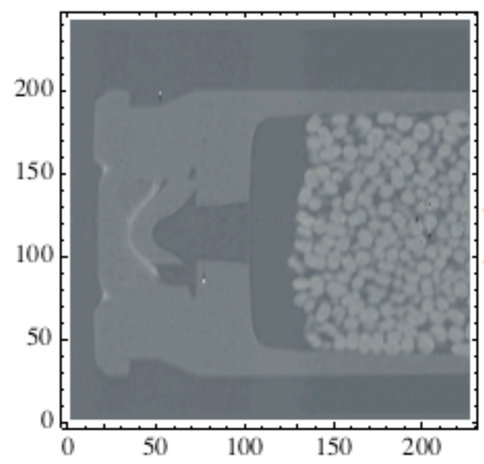
```
{AlignmentPoint → Center, AspectRatio → 1, Axes → False, AxesLabel → None, AxesOrigin → Automatic,
 AxesStyle → {}, Background → None, BaselinePosition → Automatic, BaseStyle → {},
 BoundaryStyle → None, BoxRatios → Automatic, ClippingStyle → None, ColorFunction → Automatic,
 ColorFunctionScaling → True, ColorOutput → Automatic, ContentSelectable → Automatic,
 CoordinatesToolOptions → Automatic, DataRange → Automatic, DisplayFunction :→ $DisplayFunction,
 Epilog → {}, FormatType :→ TraditionalForm, Frame → True, FrameLabel → None, FrameStyle → {},
 FrameTicks → Automatic, FrameTicksStyle → {}, GridLines → None, GridLinesStyle → {},
 ImageMargins → 0., ImagePadding → All, ImageSize → Automatic, ImageSizeRaw → Automatic,
 InterpolationOrder → None, LabelStyle → {}, LightingAngle → None, MaxPlotPoints → Automatic,
 Mesh → None, MeshFunctions → {#1 &, #2 &}, MeshStyle → Automatic, Method → Automatic,
 PerformanceGoal :→ $PerformanceGoal, PlotLabel → None, PlotRange → {Full, Full, Automatic},
 PlotRangeClipping → True, PlotRangePadding → Automatic, PlotRegion → Automatic,
 PreserveImageOptions → Automatic, Prolog → {}, RegionFunction → (True &),
 RotateLabel → True, Ticks → Automatic, TicksStyle → {}, VertexColors → Automatic}
```

```
02]:= aSlice = volume[[Round[rows / 2], All, All]];
      Dimensions[aSlice]
      {Min[aSlice], Max[aSlice]}
      Timing[ gSlice = ListDensityPlot[aSlice, ColorFunction → "GrayTones", PlotRange → {All, All, All}] ]
```

```
103]= {243, 227}
```

```
104]= {-0.0587507, 0.0814862}
```

```
105]= {1.16626,
```



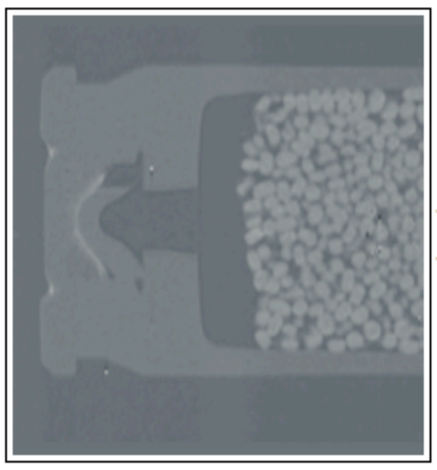
Plot quality is good. 0.6 s to plot a slice. Ok, but let's look for a faster plot command.

Useful plot options: BoxRatios, ClippingStyle, DataRange, Gridlines, InterpolationOrder, MaxPointPlots

ArrayPlot

■ **Step 3: Plot a slice with ArrayPlot. Note the time required to make a plot.**

```
In[6]:= Options[ArrayPlot]

Out[6]= {AlignmentPoint → Center, AspectRatio → Automatic, Axes → False, AxesLabel → None,
  AxesOrigin → Automatic, AxesStyle → {}, Background → None, BaselinePosition → Automatic,
  BaseStyle → {}, ClippingStyle → None, ColorFunction → Automatic, ColorFunctionScaling → True,
  ColorOutput → Automatic, ColorRules → Automatic, ContentSelectable → Automatic,
  CoordinatesToolOptions → Automatic, DataRange → All, DataReversed → False,
  DisplayFunction :→ $DisplayFunction, Epilog → {}, FormatType :→ TraditionalForm,
  Frame → Automatic, FrameLabel → None, FrameStyle → {}, FrameTicks → None,
  FrameTicksStyle → {}, GridLines → None, GridLinesStyle → {}, ImageMargins → 0.,
  ImagePadding → All, ImageSize → Automatic, ImageSizeRaw → Automatic, LabelStyle → {},
  MaxPlotPoints → ∞, Mesh → False, MeshStyle → GrayLevel[-1 + GoldenRatio], Method → Automatic,
  PixelConstrained → False, PlotLabel → None, PlotRange → All, PlotRangeClipping → False,
  PlotRangePadding → Automatic, PlotRegion → Automatic, PreserveImageOptions → Automatic,
  Prolog → {}, RotateLabel → True, Ticks → Automatic, TicksStyle → {}}

In[7]:= aSlice = volume[[Round[rows / 2], All, All]];
  Dimensions[aSlice]
  {Min[aSlice], Max[aSlice]}
  Timing[ gSlice = ArrayPlot[aSlice, ColorFunction → "GrayTones", PlotRange → {All, All, All}] ]

Out[8]= {243, 227}

Out[9]= {-0.0587507, 0.0814862}

Out[10]= {0.165494,
```



}

Plot quality is good. 0.14 s to plot a slice. Better.

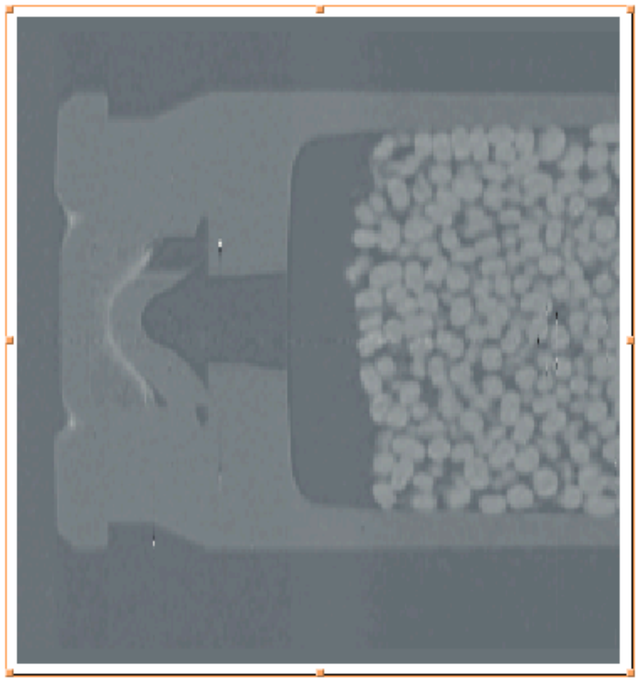
Useful plot options: Mesh, MeshStyle, and Axes->True can bring back axes labelling.

Manipulate

■ **Step 4: Add Manipulate around ArrayPlot**

2]:=
```
Manipulate[
 ArrayPlot[volume[[indexRow, All, All]],
  ColorFunction → "GrayTones",
  PlotRange → {All, All, All}, ImageSize → 300] , {{indexRow, Round[rows / 2]}, 1, rows, 1}]
```

indexRow



12]=

Comments:

1) Want to kill Manipulate? Delete the last ",1" from the command to give ....{{indexRow,Round[rows/2]},1,rows}]. To recover, may need to deselect Evaluation/Dynamic Updating Enabled.

2) Display "flashes" on scrolling through the rows. Got to fix this with a change to PlotRange for the intensity limits.

3) The row value is not displayed. Need to add some text with Epilog into the upper left corner.

4) How to save a good image to disk? Need to assign the graph to some variable. Need to record the graph parameters for inclusion in the filename.

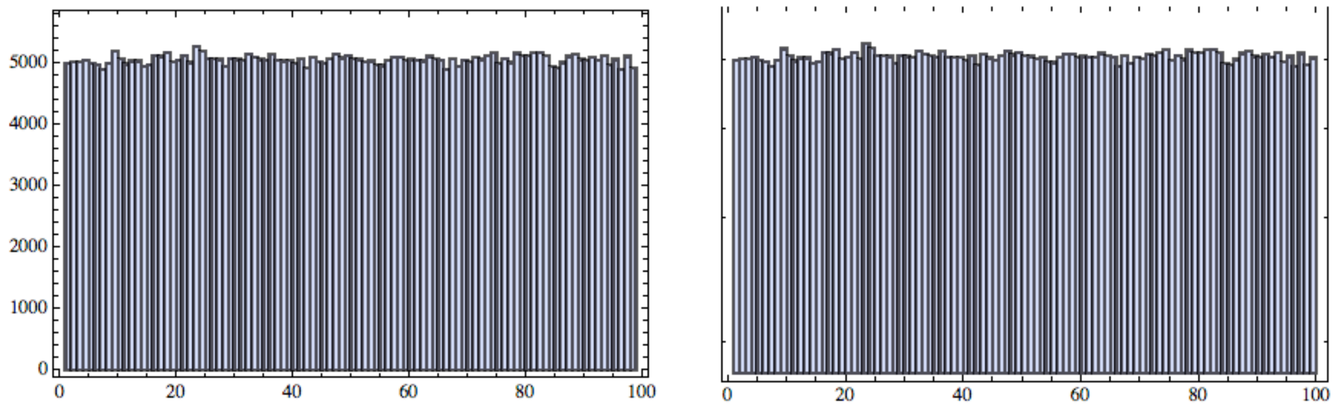
5) Any other colors besides gray?

6) Other orientations?

Step 5: Introduction to Histograms

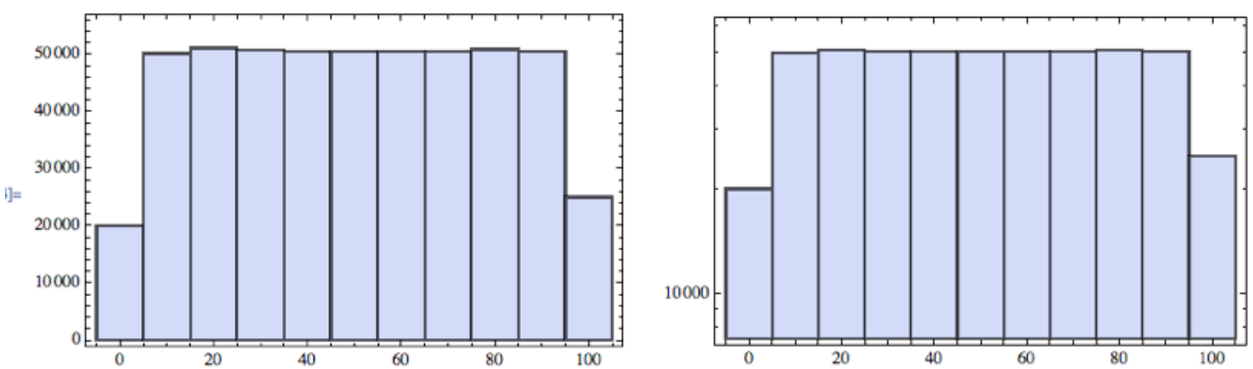
```
numberOfSamples = 500000;
numberOfBins = 100;
SeedRandom[1];
listOfNumbers = RandomInteger[{1, 99}, numberOfSamples];
{Min[listOfNumbers], Mean[listOfNumbers] // N, Max[listOfNumbers]}
Timing[gHistogramLinear = Histogram[listOfNumbers, Automatic, Frame → True];]
Timing[gHistogramLog = Histogram[listOfNumbers, numberOfBins, "LogCount", Frame → True];]
GraphicsRow[{gHistogramLinear, gHistogramLog}, ImageSize → 700]
```

```
{1, 50.0075, 99}

{1.96687, Null}

{1.94781, Null}
```




```
listOfBins = {Table[i + 5, {i, -10, 100, 10}]}
Timing[gHistogramLinear = Histogram[listOfNumbers, listOfBins, Frame → True];]
Timing[gHistogramLog = Histogram[listOfNumbers, listOfBins, "LogCount", Frame → True];]
GraphicsRow[{gHistogramLinear, gHistogramLog}, ImageSize → 700]
```

```
{{-5, 5, 15, 25, 35, 45, 55, 65, 75, 85, 95, 105}}

{1.67238, Null}

{1.6644, Null}
```



Histograms

a diagram consisting of rectangles whose area is proportional to the frequency of a variable and whose width is equal to the class interval.

Concerns:

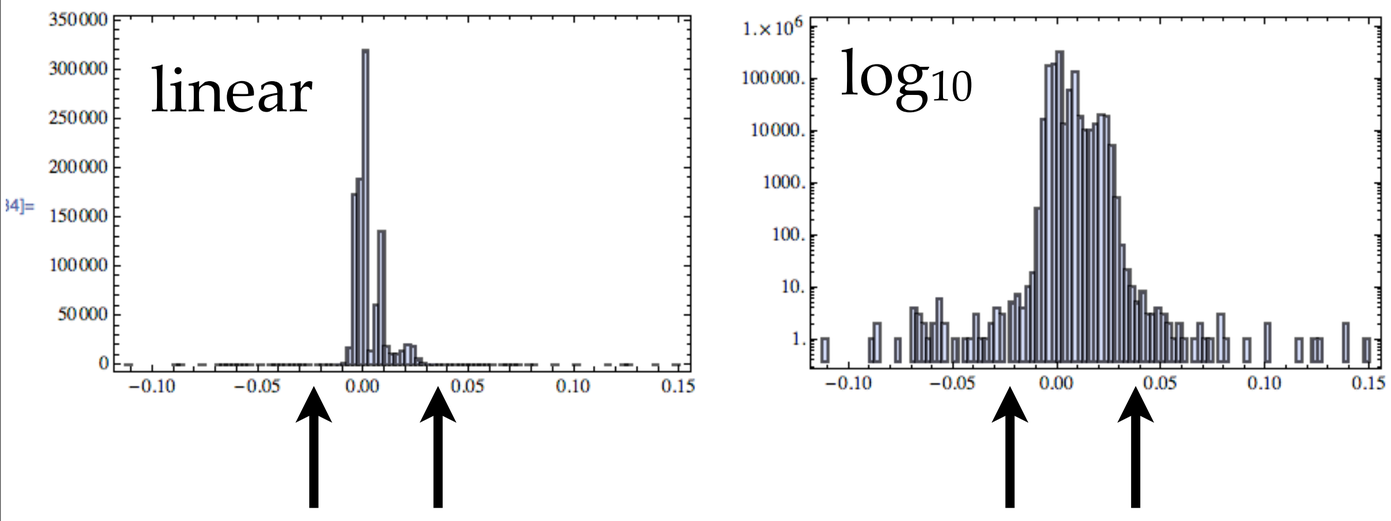
1) must be a 1D list of numbers
2) slow for >1 million numbers
3) precise control over bins is difficult.

Here, listOfNumbers is 500,000 integers randomly chosen from [1,99]. It's pretty random; the mean is 50.0075.

At 500,000 numbers, 100 bins, it takes 1.6 seconds to generate a histogram. The bullet volume has 13 million numbers.

■ Step 6: Finding intensity limits from a histogram

```
5]:= numberOfSamples = 10^6;
     maxNumberOfSamples = rows × columns × slices
     numberOfBins = 100;

6]= 13 404 123

8]:= SeedRandom[1];
     RandomChoice[Flatten[volume], 10]
     {Min[volume], Mean[Flatten[volume]], Max[volume]}

9]= {0., 0.00691298, 0.00879451, 0.0189237, -0.00218527, 0.0168477, 0., 0., -0.00259126, 0.}

0]= {-0.12809, 0.00240564, 0.17781}

1]:= listOfIntensities = RandomChoice[Flatten[volume], Min[{numberOfSamples, maxNumberOfSamples}]];
     Timing[gHistogramLinear = Histogram[listOfIntensities, numberOfBins, Frame → True];]
     Timing[gHistogramLog = Histogram[listOfIntensities, numberOfBins, "LogCount", Frame → True];]
     GraphicsRow[{gHistogramLinear, gHistogramLog}, ImageSize → 700]

2]= {4.13398, Null}

3]= {4.11197, Null}
```



4]=

intensityLimits = {-0.025, 0.028}

Histograms

1) Flatten[volume] yields a 1D list.
2) RandomChoice extracts some numbers from the list.
3) Both linear and $\log_{10}$ plots are useful.

Next, use intensityLimits to control the plot range for all orthoslices.

Histograms

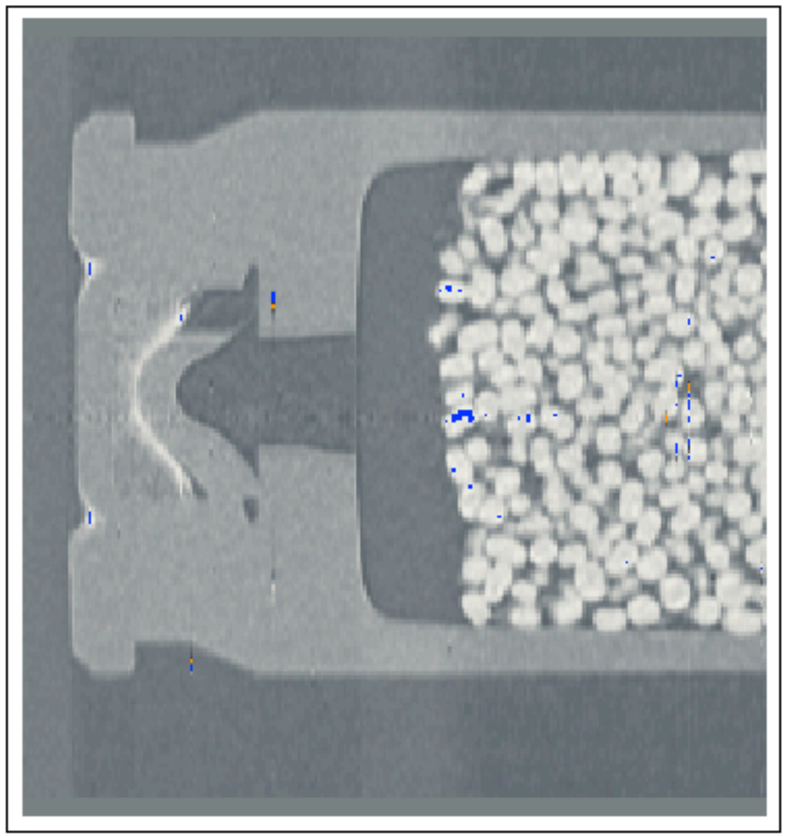
Use intensityLimits to control the plot range for an orthoslice.

Use ClippingStyle to define colors for the intensities below and above the limits.

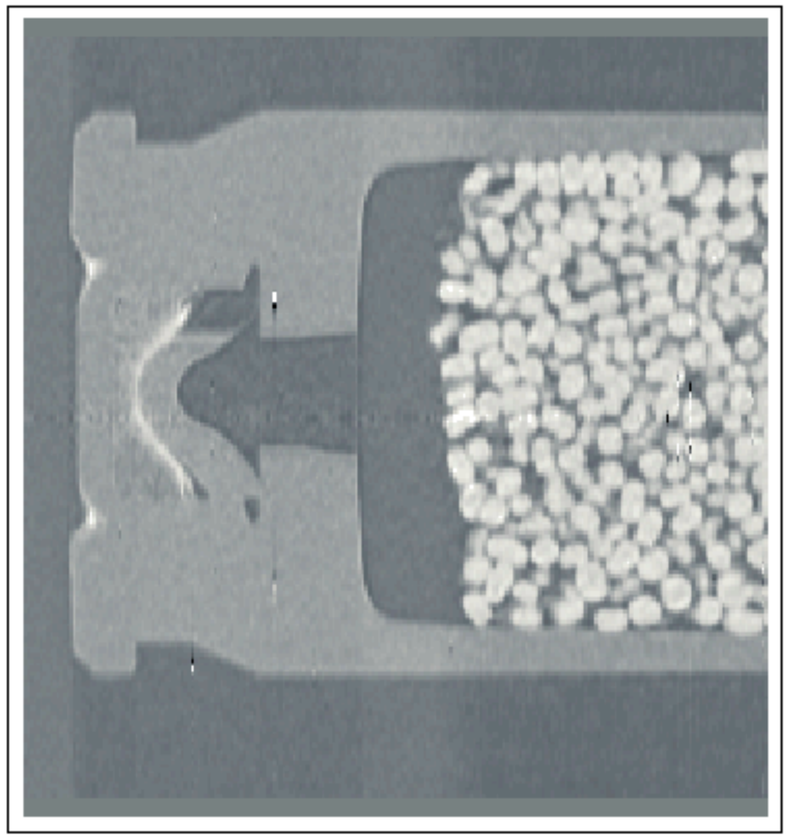
In Manipulate, we probably want control for minimum and maximum plot limits.

```
intensityLimits = {-0.025, 0.028}
```

```
ArrayPlot[volume[[Round[rows / 2], All, All]],
 ColorFunction → "GrayTones", ClippingStyle → {Orange, Blue},
 PlotRange → {All, All, intensityLimits}, ImageSize → 300]
```



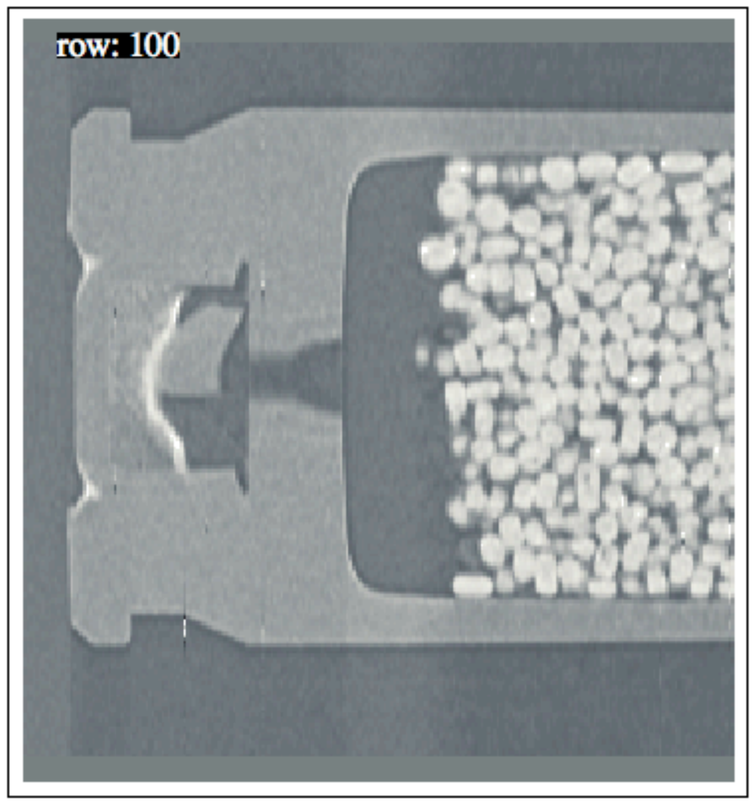
```
ArrayPlot[volume[[Round[rows / 2], All, All]],
 ColorFunction → "GrayTones", ClippingStyle → {Black, White},
 PlotRange → {All, All, intensityLimits}, ImageSize → 300]
```



■ **Step 7: Put the row index on the plot with Epilog, Inset**

```
In[48]:= indexRow = 100;
         textStringForPlot = "row: " <> ToString[indexRow];
         textStringForPlot

Out[50]= row: 100

In[51]:= gText = Text[Style[textStringForPlot, 14, White, Background → Black]];
         gText

Out[52]= row: 100

In[53]:= intensityLimits = {-0.025, 0.028};
         ArrayPlot[volume[[indexRow, All, All]],
          ColorFunction → "GrayTones",
          PlotRange → {All, All, intensityLimits}, ImageSize → 300,
          Epilog → Inset[gText, Scaled[{0.15, 0.95}]]]
```

Out[54]=



Epilog - add notes to a plot.

Useful commands:

StringQ - is the result really a string?

Text - makes graphics object from string

Style - font, color, background, etc.

Inset[*object, position*] - insets graphic object

position - can be {row, column} or scaled [0,1] coordinates.

9

FileNames and more

■ **Step 8: Save the graph to a variable, generate a filename, and save image as jpg**

```
7]:= listOfFilenames = FileNames["volume*.h5", NotebookDirectory[]]
     listOfFilenames[[1]]
     FileNameSplit[listOfFilenames[[1]] ]
     Last[FileNameSplit[listOfFilenames[[1]] ]]
     StringSplit[Last[FileNameSplit[listOfFilenames[[1]] ]], {"."}]
     First[StringSplit[Last[FileNameSplit[listOfFilenames[[1]] ]], {"."}]]

7]= {/Volumes/Sab-Data-1/t4581/wk4/volume_bullet_p134.h5}

8]= /Volumes/Sab-Data-1/t4581/wk4/volume_bullet_p134.h5

9]= {, Volumes, Sab-Data-1, t4581, wk4, volume_bullet_p134.h5}

0]= volume_bullet_p134.h5

1]= {volume_bullet_p134, h5}

2]= volume_bullet_p134

3]:= textStringForPlot
     StringReplace[textStringForPlot, {": " → ""}]

3]= row: 100

4]= row100

5]:= newFileName = First[StringSplit[Last[FileNameSplit[listOfFilenames[[1]] ]], {"."}]] <>
       "_" <> StringReplace[textStringForPlot, { ": " → ""}] <> ".jpg"

5]= volume_bullet_p134_row100.jpg

6]:= gPlot = ArrayPlot[volume[[indexRow, All, All]],
       ColorFunction → "GrayTones",
       PlotRange → {All, All, All}, ImageSize → 800];
     Export[NotebookDirectory[] <> newFileName, gPlot, "JPG"];
     Show[gPlot, ImageSize → 300]
```

Export - writes data to a file.

Export[*file, expression, format*]

My strategy for *file* is to re-use as much as possible, without re
typing. If the *.h5 is "volume_bullet_p135.h5", then orthoslice
of row 100 will be called "volume_bullet_p135_row100.jpg"

Useful commands:

Filenames[*"volume*.h5"*,* **NotebookDirectory[]]** - yields a list
filenames. In visualization work, this list can be 1 to several
thousand files.

NotebookDirectory - quick way to get most of the *file* information.

Last[FileNameSplit - yields the *.h5 filename

First[StringSplit[*"string",{"."}***]**] - yields the string before .h5

StringReplace["row: 100", {": "->""}] deletes ": " from string
give "row100"

So, the complete command is this long thing:

```
newFileName = First[StringSplit[Last[FileNameSplit[
        listOfFilenames[[1]] ]], {"."}]] <>   "_" <>
        StringReplace[textStringForPlot, { ": " -> ""}] <> ".jpg"
```

Tested various min,max values in PlotRange

- **Step 9: Colors besides gray?**

From Documentation / Color Schemes

Color Schemes

Mathematica includes a wide selection of carefully chosen color schemes that can immediately be used throughc graphics and visualization system.

ColorData — named color gradients and collections
"Image" ▪ **"Panel"** ▪ **"ColorFunction"** ▪ **...**
Palettes ▸ Color Schemes — a palette for selecting a color scheme
ColorDataFunction — color scheme object

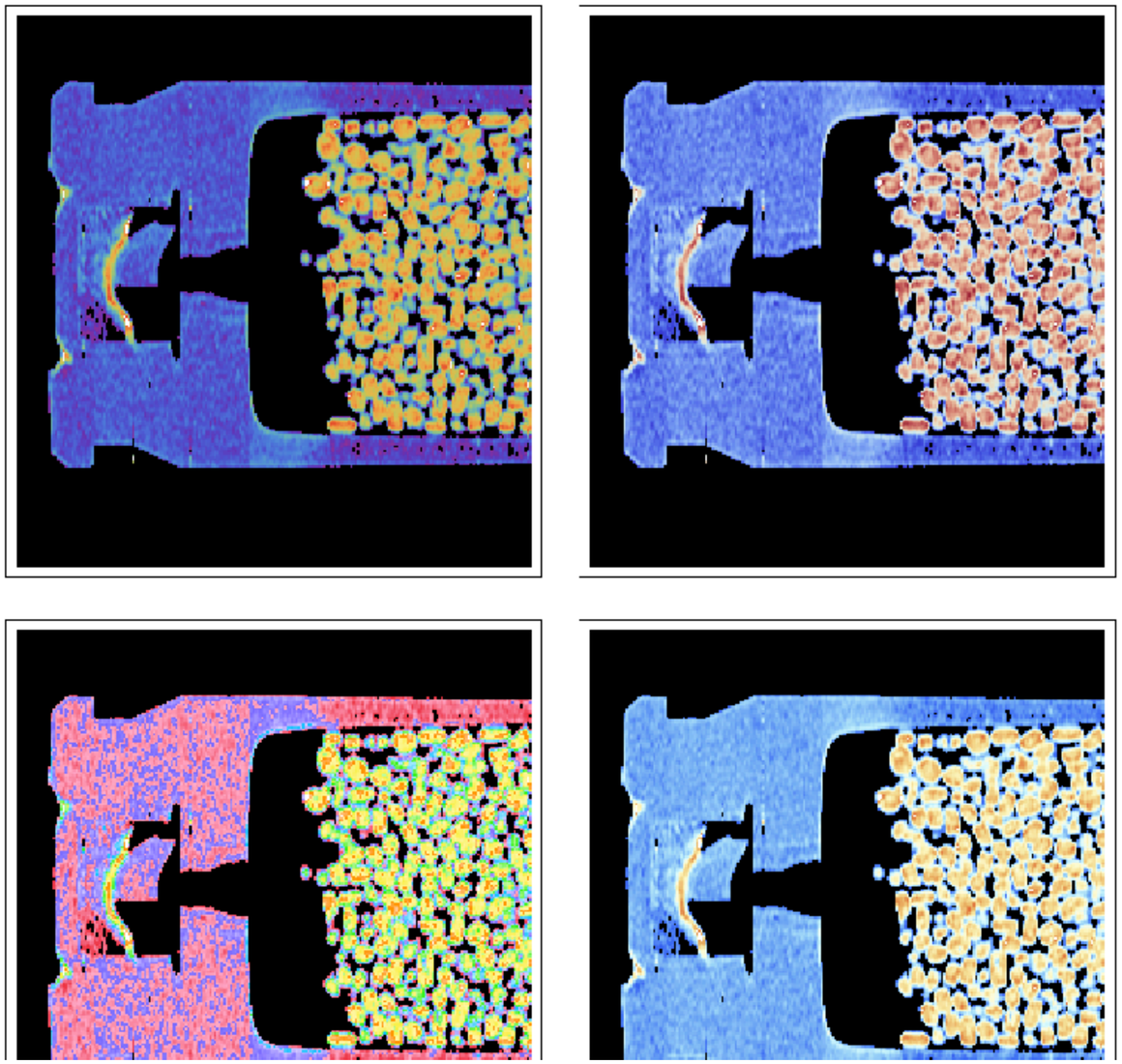
Color Gradients

"Gradients" — list of named color gradients

"AlpineColors"	—	"LakeColors"	—
"ArmyColors"	—	"MintColors"	—
"AtlanticColors"	—	"NeonColors"	—
"AuroraColors"	—	"PearlColors"	—
"AvocadoColors"	—	"PlumColors"	—
"BeachColors"	—	"RoseColors"	—
"CandyColors"	—	"SolarColors"	—
"CMYKColors"	—	"SouthwestColors"	—
"DeepSeaColors"	—	"StarryNightColors"	—
"FallColors"	—	"SunsetColors"	—
"FruitPunchColors"	—	"ThermometerColors"	—
"IslandColors"	—	"WatermelonColors"	—
"BrassTones"	—	"GreenPinkTones"	—
"BrownCyanTones"	—	"PigeonTones"	—
"CherryTones"	—	"RedBlueTones"	—
"CoffeeTones"	—	"RustTones"	—
"FuchsiaTones"	—	"SiennaTones"	—
"GrayTones"	—	"ValentineTones"	—
"GrayYellowTones"	—		
"DarkTerrain"	—	"LightTerrain"	—
"GreenBrownTerrain"	—	"SandyTerrain"	—
"BrightBands"	—	"DarkBands"	—
"Aquamarine"	—	"Pastel"	—
"BlueGreenYellow"	—	"Rainbow"	—
"DarkRainbow"	—	"TemperatureMap"	—
"LightTemperatureMap"	—		

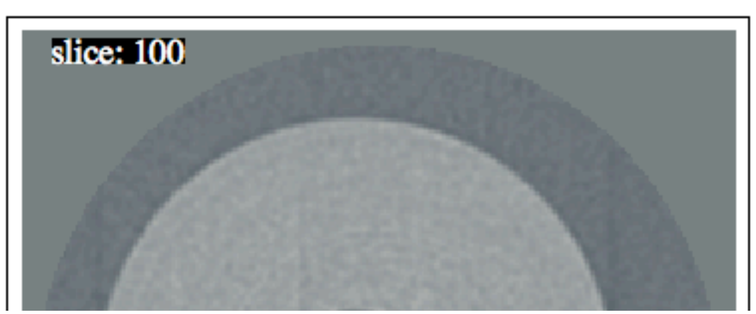
```
pR = {All, All, {0.005, 0.028}};
cS = {Black, White};
colorList = {"Rainbow", "ThermometerColors", "BrightBands", "LightTemperatureMap"};
g1 = ArrayPlot[volume[[indexRow, All, All]], ColorFunction → colorList[[1]], PlotRange → pR,
   ClippingStyle → cS];
g2 = ArrayPlot[volume[[indexRow, All, All]], ColorFunction → colorList[[2]], PlotRange → pR,
   ClippingStyle → cS];
g3 = ArrayPlot[volume[[indexRow, All, All]], ColorFunction → colorList[[3]], PlotRange → pR,
   ClippingStyle → cS];
g4 = ArrayPlot[volume[[indexRow, All, All]], ColorFunction → colorList[[4]], PlotRange → pR,
   ClippingStyle → cS];
GraphicsGrid[{{g1, g2}, {g3, g4}}, ImageSize → 600]
```

=



■ Step 10: Which orientation to plot?

```
]:= orientationList = {"columns slices", "rows slices", "rows columns"};
   index = 100;
   orientation = orientationList[[3]]

2]= rows columns

3]:= Which[orientation == orientationList[[1]],
     Module[{}, index = Min[{index, rows}];
      aSlice = volume[[index, All, All]];
      textStringForPlot = "row: " <> ToString[index]; ],

     orientation == orientationList[[2]],
     Module[{}, index = Min[{index, columns}];
      aSlice = volume[[ All, index, All]];
      textStringForPlot = "column: " <> ToString[index]; ],

     True,
     Module[{}, index = Min[{index, slices}];
      aSlice = volume[[ All, All, index]];
      textStringForPlot = "slice: " <> ToString[index]; ] ];

gText = Text[Style[textStringForPlot, 14, White, Background → Black]];
intensityLimits = {-0.025, 0.028};
ArrayPlot[aSlice,
 ColorFunction → "GrayTones",
 PlotRange → {All, All, intensityLimits}, ImageSize → 300,
 Epilog → Inset[gText, Scaled[{0.15, 0.95}]]]
```



Which

Conditionals - If, Which, Switch

Testing Expressions - == != < <=
> >= && ||

Concerns:

1) Avoid index too large for that dimension

2) Update textStringForPlot with row, column, or slice text as needed.

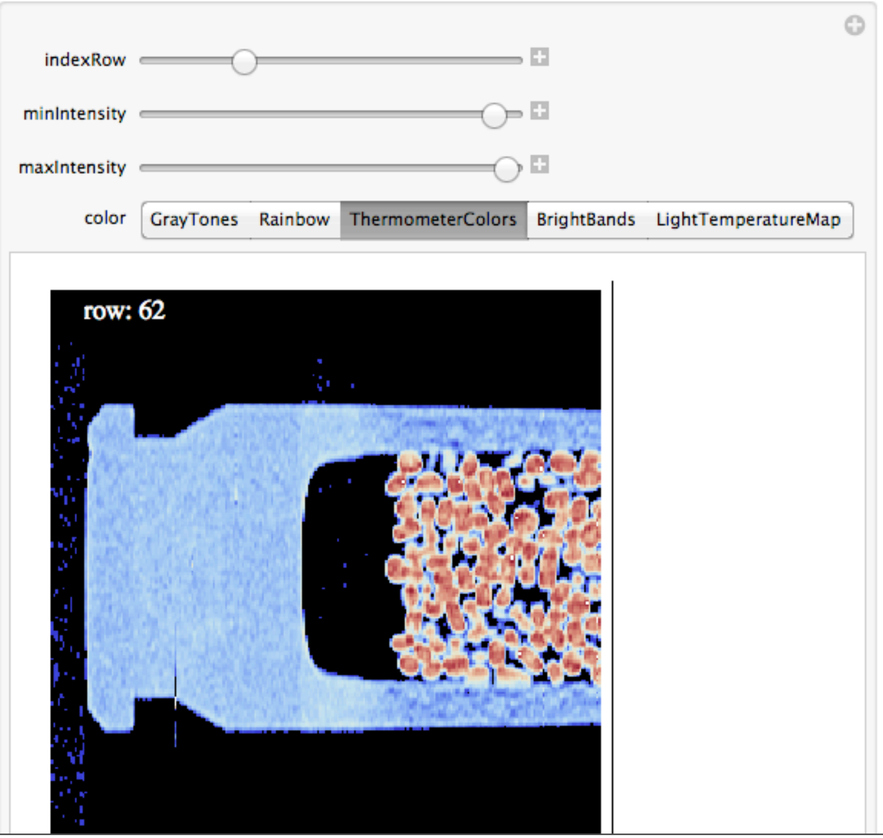
3) If there is a typo in “orientation”, the last “True” in Which[] will plot “rows columns”.

12

- **Step 11: Some of the fancy stuff (intensity limits, epilog, colors)**
  (a) Debug the code in one cell
  (b) Debug the code inside a Module[{},]
  (c) With operational Module, wrap Manipulate around Module and add controls

```
intensityLimits = {-0.025, 0.028};
colorList = {"GrayTones", "Rainbow", "ThermometerColors", "BrightBands", "LightTemperatureMap"}
Manipulate[
 Module[{},
  textStringForPlot = "row: " <> ToString[indexRow];
  gText = Text[Style[textStringForPlot, 14, White, Background → Black]];
  gPlot = ArrayPlot[volume[[indexRow, All, All]],
    ColorFunction → color, ClippingStyle → {Black, White},
    PlotRange → {All, All, {minIntensity, maxIntensity}}, ImageSize → 300,
    Epilog → Inset[gText, Scaled[{0.15, 0.95}]]];
  Show[gPlot]
 ],
 {{indexRow, Round[rows / 2]}, 1, rows, 1},
 {{minIntensity, Min[intensityLimits]}, Min[intensityLimits], Mean[intensityLimits]},
 {{maxIntensity, Max[intensityLimits]}, Mean[intensityLimits], Max[intensityLimits]},
 {{color, colorList[[1]]}, colorList}
]
```

```
{GrayTones, Rainbow, ThermometerColors, BrightBands, LightTemperatureMap}
```



My strategy:

a) I copy good code from above into a cell and run it. Sometimes use a second notebook as a scratch file.

b) Copy into a second cell and wrap a Module[{}, ...] around the main code and run it.

c) Copy into a third cell and wrap a Manipulate[*expr*, {{u, $u_{initial}$}, u_{min}, u_{max}, du}] around the main code and hope it works.

There is a lot of back and forth with a,b,c as the programming flow develops.

13

Task 1: Write a Manipulate for showing any row, setting min,max intensity values, and setting several different colors.
Concerns: Use legal values for initial row, min intensity, max intensity.
BTW: an answer (no requirement to use my programming style) is hidden under the Manipulator output. Try not to peak.

Task 2: Upgrade with ability to save a jpg file.

Task 3: Upgrade with ability to switch to the other orientations.

14