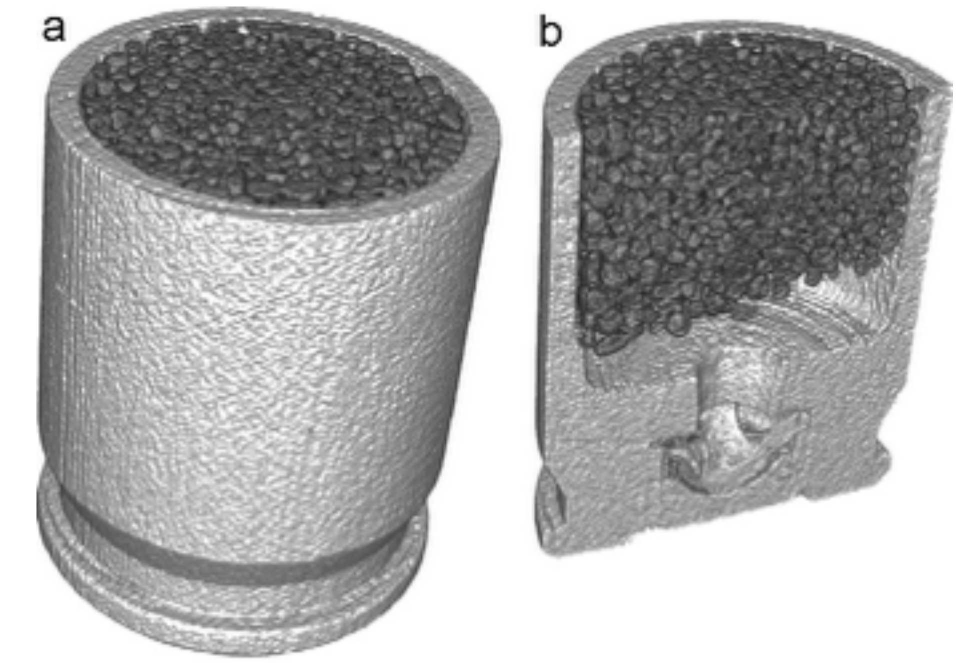
Mathematica: binary files and HDF5 files
exploration of the bullet volume



cartridge

To prepare:

1) Moodle: download Week 1/bullet dataset (binary, 26.8 MB)
2) Moodle: download Week 2/bullet dataset (HDF5, 107 MB)
3) Moodle: download Week 2/Pgm2_binary_HDF5.nb

Note “Step number” for the subsections. These are our navigation guides for the lecture.

■ Step 1: Filenames and files sizes

```
In[120]:= listOfFilenames = FileNames["volume*", NotebookDirectory[]]
          Map[ FileByteCount, listOfFilenames]

Out[120]= {/Users/lesbutler/Documents/h4581/volume_bullet_p134.h5,
           /Users/lesbutler/Documents/h4581/volume_bullet_p134_uint16.bin}

Out[121]= {107 235 032, 26 808 246}
```

FileNames

`FileNames[]`
lists all files in the current working directory.

`FileNames[`*form*`]`
lists all files in the current working directory whose names match the string pattern *form*.

`FileNames[{`$form_1$`, `$form_2$`, …}]`
lists all files whose names match any of the $form_i$.

`FileNames[`*forms*`, {`dir_1`, `dir_2`, …}]`
lists files with names matching *forms* in any of the directories dir_i.

`FileNames[`*forms*`, `*dirs*`, `*n*`]`
includes files that are in subdirectories up to *n* levels down.

Map (/@)

`Map[`*f*`, `*expr*`]` or *f* `/@` *expr*
applies *f* to each element on the first level in *expr*.

`Map[`*f*`, `*expr*`, `*levelspec*`]`
applies *f* to parts of *expr* specified by *levelspec*.

■ Step 2: Import the HDF5 file

In[122]:=
```
listOfFilenames = FileNames["volume*h5", NotebookDirectory[]]
volume = Import[listOfFilenames[[1]], {"Datasets", "/volume"} ];
{rows, columns, slices} = Dimensions[volume]
```

Out[122]= {/Users/lesbutler/Documents/h4581/volume_bullet_p134.h5}

Out[124]= {243, 243, 227}

In[125]:= 243 × 243 × 227 × 2

Out[125]= 26 808 246

In[126]:= 243 × 243 × 227 × 8

Out[126]= 107 232 984

In[127]:= 107 235 032 - %

Out[127]= 2048

HDF5 (.h5)

HDF data format Version 5.

General purpose format for representing multidimensional datasets and images.

Used for storage, management, and exchange of scientific data.

HDF is an acronym for Hierarchical Data Format.

Developed by the U.S. National Center for Supercomputing Applications (NCSA).

Binary file format.

Incompatible with HDF Version 4 and earlier.

- Import and Export support the HDF5 format.
- *Mathematica* reads and writes HDF5 images as data arrays.
- Compound data structures are ignored by Import.

▸ IMPORT AND EXPORT

▾ ELEMENTS

- General Import elements:

"Elements"	list of elements and options availabl
"Rules"	full list of rules for each element an
"Options"	list of rules for options, properties a

- Data representation elements:

"Data"	all datasets imported as a list of arr
"Datasets"	names of all datasets

▾ **Basic Examples** (4)

Show the datasets stored in a sample file:

In[1]:= `Import["ExampleData/image.h5"]`

Out[1]= {/image24bitpixel, /image8bit, /palette}

Import 8-bit RGB raster data and render it as an Image object:

In[1]:= `Image[Import["ExampleData/image.h5", {"Datasets", "/image24bitpixel"} ], "By`

Out[1]=

Export a matrix to HDF5:

In[1]:= `Export[ "matrix.h5", {{1, 2}, {2, 3}}]`

Out[1]= matrix.h5

Show the datasets contained in this file:

In[2]:= `Import["matrix.h5"]`

Out[2]= {/Dataset1}

Import "Dataset1":

In[3]:= `Import["matrix.h5", {"Datasets", "/Dataset1"}]`

Out[3]= {{1, 2}, {2, 3}}

Export a named dataset:

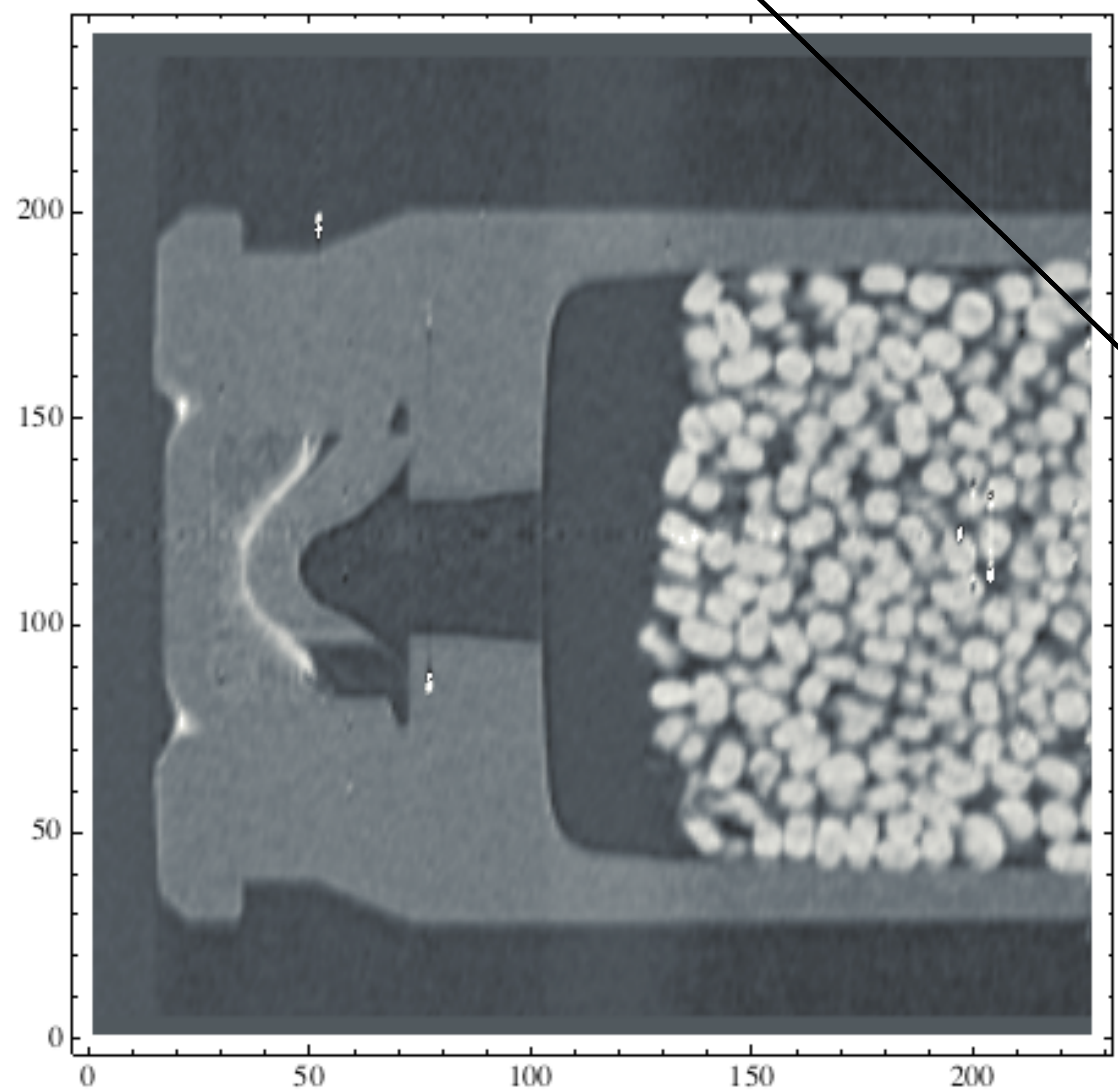
In[1]:= `Export["m1.h5", {{1, 2}, {2, 3}}, {"Datasets", "m1"}]`

Out[1]= m1.h5

■ **Step 4: Plot a slice**

```
28]:= aSlice = volume[[Round[rows / 2], All, All]];
      Dimensions[aSlice]
29]=  {243, 227}

30]:= {Min[aSlice], Max[aSlice]}
30]=  {-0.0587507, 0.0814862}

31]:= gSlice = ListDensityPlot[aSlice, ColorFunction → "GrayTones", PlotRange → {A
```

31]=



Table, Range,
Part,
First, Last, Take, Drop, Append,
Length, **Position**,
Flatten, Join, **Partition**, **Transpose**, Map

List Manipulation

Lists are central constructs in *Mathematica*, used to represent collections, arrays, sets, and se
structure and size, and can routinely involve even millions of elements. Well over a thousand I
operate directly on lists, making lists a powerful vehicle for interoperability.

Constructing Lists »

{*a*, *b*, …} (**List**) — specify a list explicitly

Table — make a table of values of an expression

Array — make an array of any dimension from a function

Range · **SparseArray** · **Tuples** · **NestList** · **Sow** · **Reap** · **...**

Elements of Lists »

list[[…]] (**Part**) — parts or sequences of parts (;;), resettable with =

First · **Last** · **Take** · **Drop** · **Extract** · **Append** · **ReplacePart** · **...**

Select — select according to a function

Cases — give cases matching a pattern

Length · **Position** · **MemberQ** · **DeleteDuplicates** · **...**

Rearranging & Restructuring Lists »

Flatten — flatten out nested lists

Join · **Partition** · **Transpose** · **Reverse** · **Sort** · **Split** · **Gather** · **Riffle** · **...**

Applying Functions to Lists »

Map (/@) — map a function over a list: *f* /@ {*a*, *b*, *c*} ⟶ {*f*[*a*], *f*[*b*], *f*[*c*]}

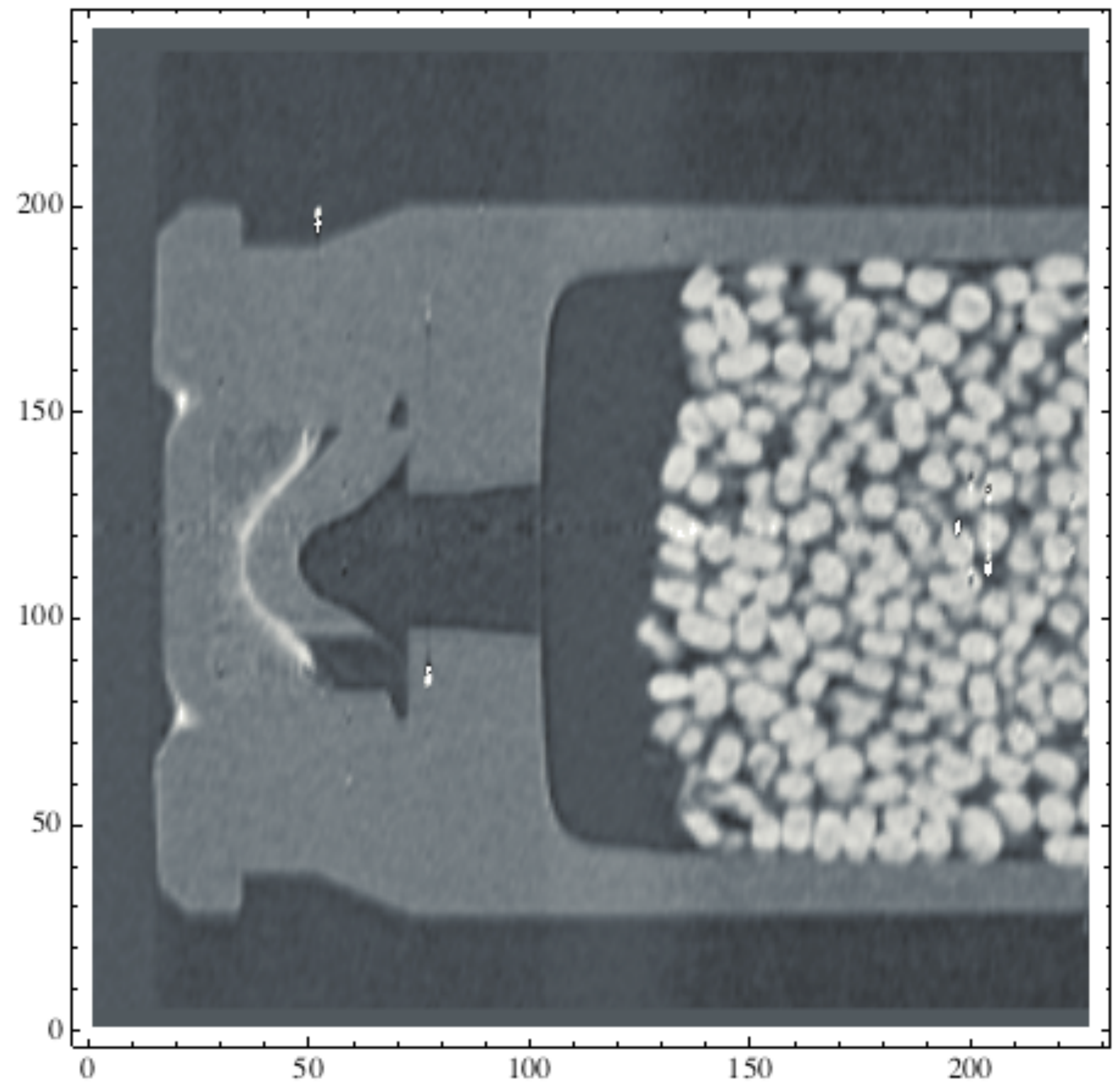
Apply (@@, @@@) — apply a function to a list: *f* @@ {*a*, *b*, *c*} ⟶ *f*[*a*, *b*, *c*]

MapIndexed · **Scan** · **Thread** · **MapThread** · **Outer** · **FoldList** · **...**

▪ Step 4: Plot a slice

```
28]:= aSlice = volume[[Round[rows / 2], All, All]];
      Dimensions[aSlice]
29]= {243, 227}

30]:= {Min[aSlice], Max[aSlice]}
30]= {-0.0587507, 0.0814862}

31]:= gSlice = ListDensityPlot[aSlice, ColorFunction → "GrayTones", PlotRange → {All, All, {-0.01, 0.03}}]
```

31]=



ListDensityPlot

`ListDensityPlot[array]`
generates a smooth density plot from an array of values.

`ListDensityPlot[{{x1, y1, f1}, {x2, y2, f2}, ...}]`
generates a density plot with values defined at specified points.

MORE INFORMATION

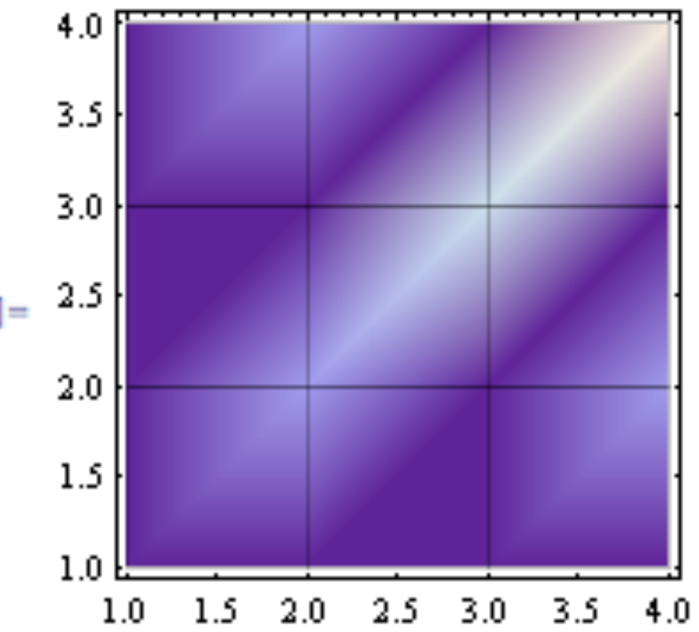
EXAMPLES

Basic Examples (3)

Use an array of values to define heights for a density:

```
In[1]:= ListDensityPlot[{{1, 1, 1, 1}, {1, 2, 1, 2}, {1, 1, 3, 1}, {1, 2, 1, 4}}, Mesh
```

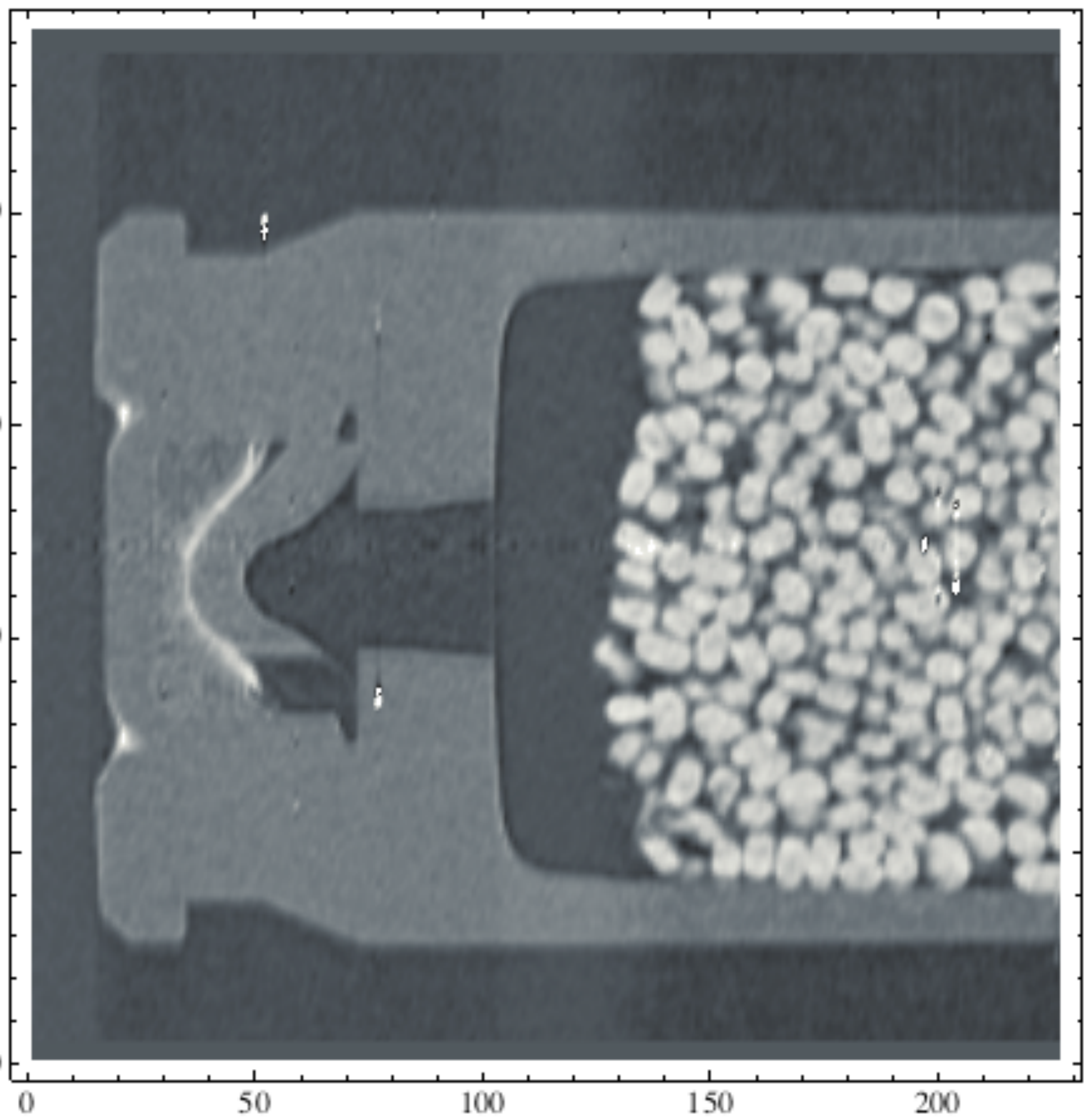
Out[1]=



■ **Step 4: Plot a slice**

```
28]:= aSlice = volume[[Round[rows / 2], All, All]];
      Dimensions[aSlice]

29]=  {243, 227}

30]:= {Min[aSlice], Max[aSlice]}

30]=  {-0.0587507, 0.0814862}

31]:= gSlice = ListDensityPlot[aSlice, ColorFunction → "GrayTones", PlotRange → {All, All, {-0.01, 0.03}}]
```

31]=



ColorFunction

`ColorFunction`
is an option for graphics functions that specifies a function to apply to determine colors of elements.

MORE INFORMATION

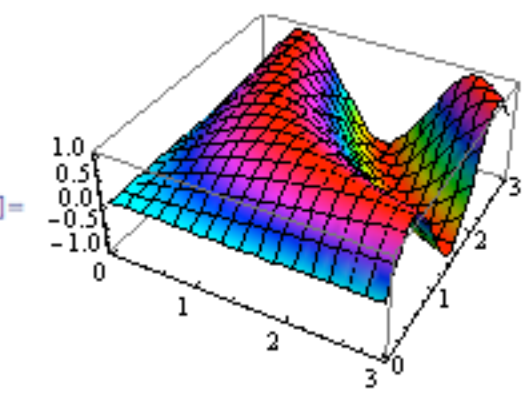
EXAMPLES

Basic Examples (4)

Color the surface by height:

```
In[1]:= Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}, ColorFunction → Function[{x, y, z}, Hue[z]]]
```

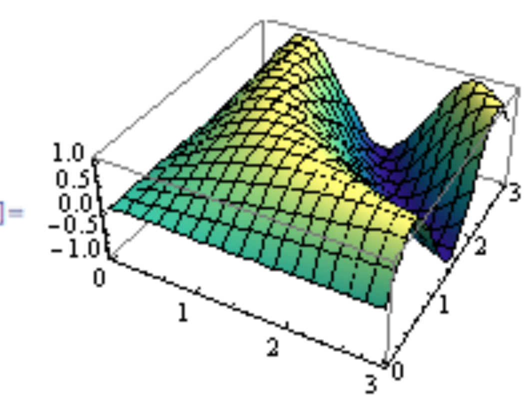
Out[1]=



Use predefined gradients:

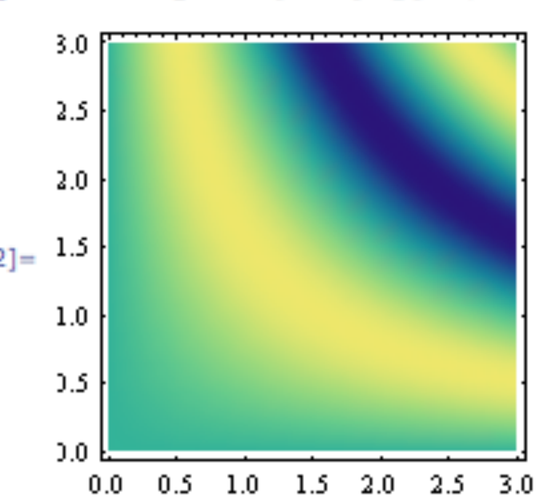
```
In[1]:= Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}, ColorFunction → "BlueGreenYellow"]
```

Out[1]=



```
In[2]:= DensityPlot[Sin[x y], {x, 0, 3}, {y, 0, 3}, ColorFunction → "BlueGreenYellow"]
```

Out[2]=

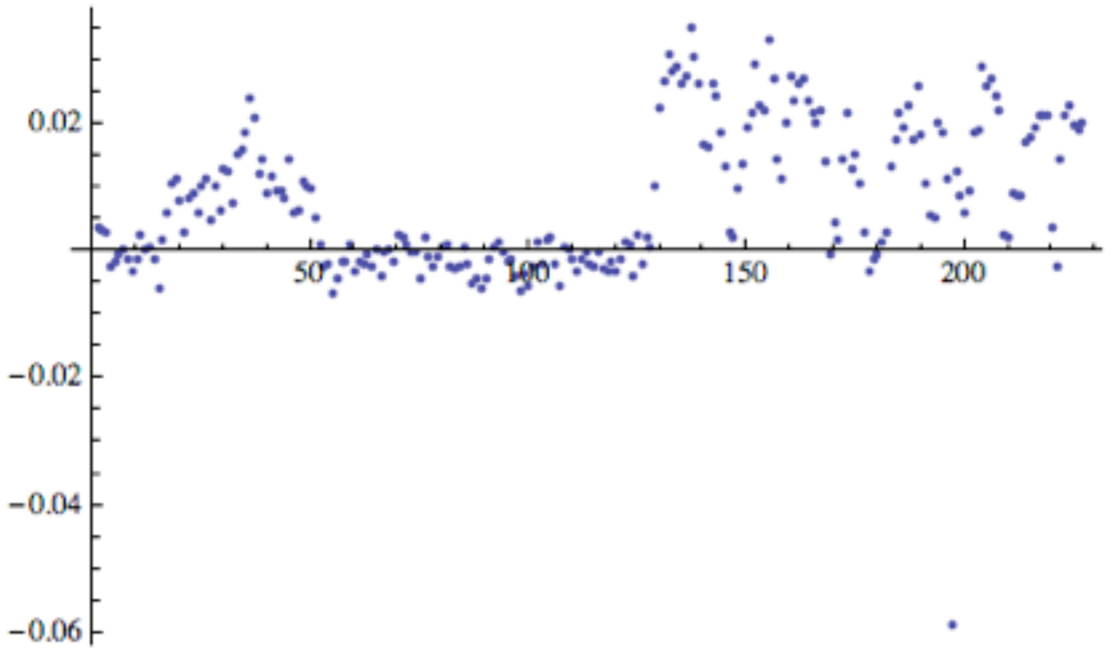


■ **Step 5: Plot a line**

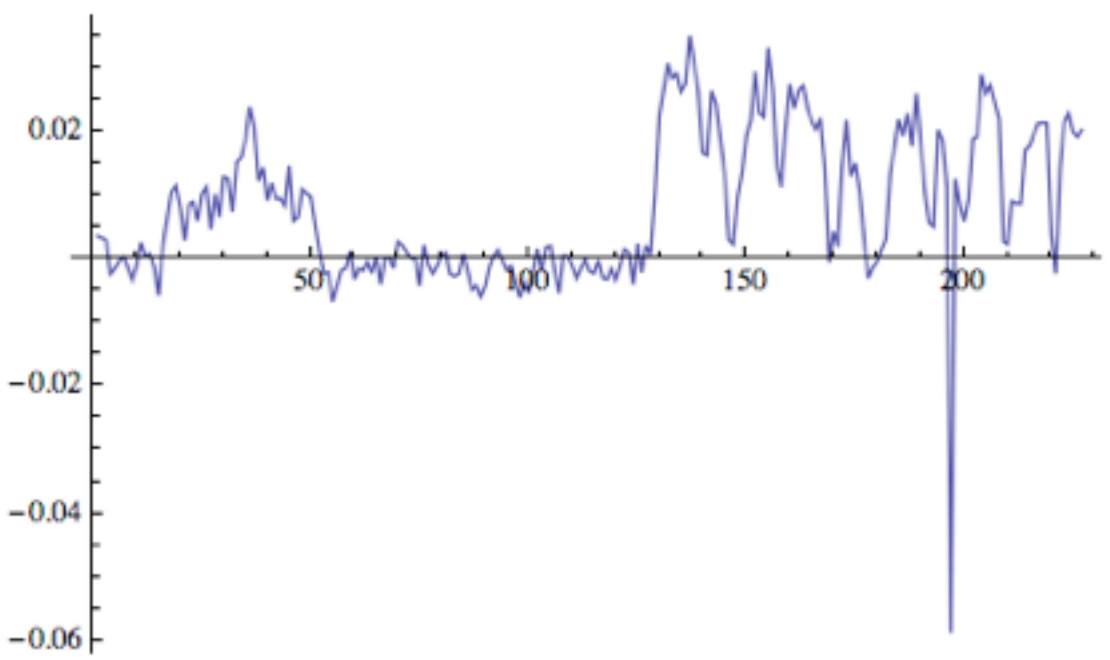
In[132]:=
```
aLine = volume[[Round[rows / 2], Round[columns / 2], All]];
Dimensions[aLine]
```
Out[133]= {227}

In[134]:=
```
{Min[aLine], Max[aLine]}
```
Out[134]= {-0.0587507, 0.0349275}

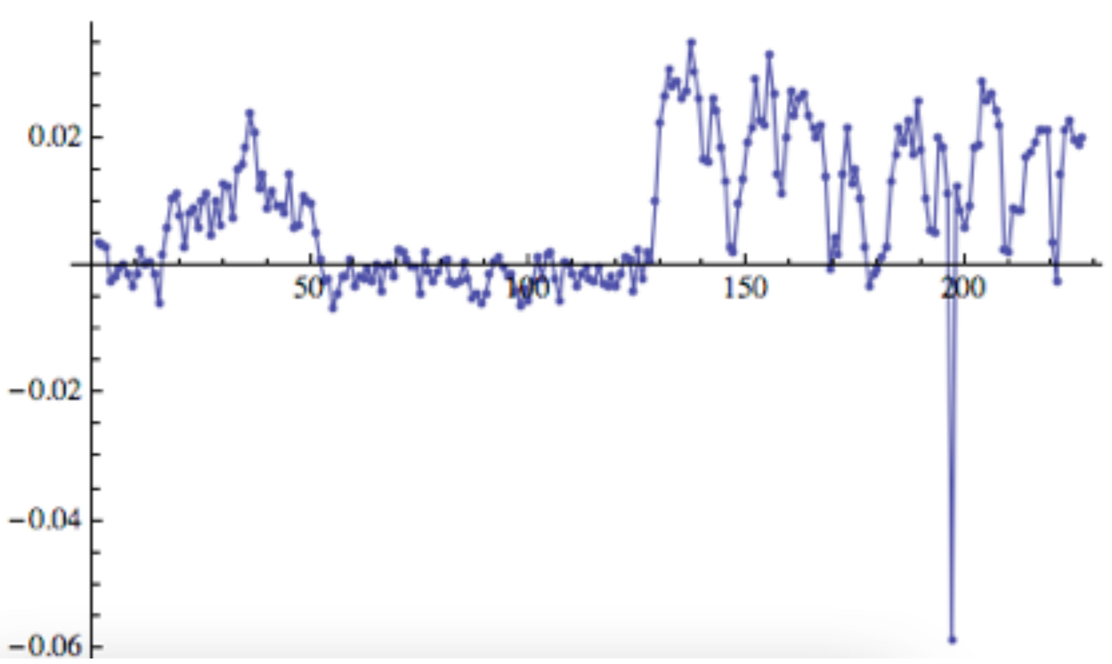
In[135]:=
```
gLine = ListPlot[aLine, PlotRange → {All, All}]
```
Out[135]=



In[136]:=
```
gLineV2 = ListPlot[aLine, PlotRange → {All, All}, Joined → True
```
Out[136]=



In[137]:=
```
Show[{gLine, gLineV2}]
```
Out[137]=



ListPlot

`ListPlot[{y1, y2, ...}]`
plots points corresponding to a list of values, assumed to correspond

`ListPlot[{{x1, y1}, {x2, y2}, ...}]`
plots a list of points with specified x and y coordinates.

`ListPlot[{list1, list2, ...}]`
plots several lists of points.

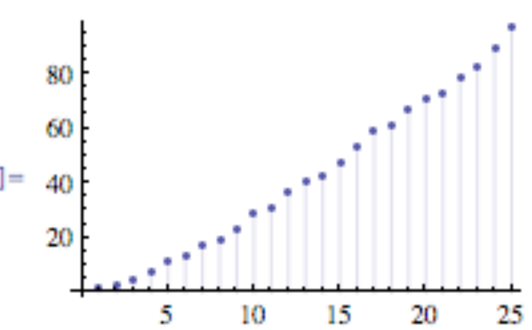
▸ MORE INFORMATION

▾ EXAMPLES

▾ **Basic Examples** (3)

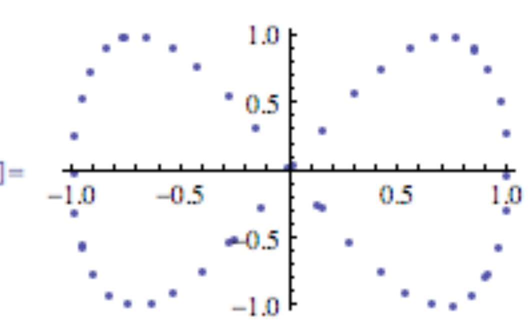
Plot a list of y values:

In[1]:=
```
ListPlot[Prime[Range[25]], Filling → Axis]
```
Out[1]=



Plot a list of x, y pairs:

In[1]:=
```
ListPlot[Table[{Sin[n], Sin[2 n]}, {n, 50}]]
```
Out[1]=



Show

`Show[graphics, options]`
shows graphics with the specified options added.

`Show[g1, g2, ...]`
shows several graphics combined.

Plot

`Plot[f, {x, xmin, xmax}]`
generates a plot of f as a function of x from x_{min} to x_{max}.

`Plot[{f1, f2, ...}, {x, xmin, xmax}]`
plots several functions f_i.

▾ MORE INFORMATION

- `Plot` treats the variable x as local, effectively using `Block`.
- `Plot` has attribute `HoldAll`, and evaluates f only after assigning specific numerical values to x.
- In some cases it may be more efficient to use `Evaluate` to evaluate f symbolically before specific numerical values a
- No curve is drawn in any regions where f evaluates to `None`.
- `Plot` has the same options as `Graphics`, with the following additions and changes:

AspectRatio	1 / GoldenRatio	ratio of width to height
Axes	True	whether to draw axes
ClippingStyle	None	what to draw where curves are clipped »
ColorFunction	Automatic	how to determine the coloring of curves
ColorFunctionScaling	True	whether to scale arguments to ColorFunction
EvaluationMonitor	None	expression to evaluate at every function evaluat
Exclusions	Automatic	points in x to exclude
ExclusionsStyle	None	what to draw at excluded points
Filling	None	filling to insert under each curve
FillingStyle	Automatic	style to use for filling
MaxRecursion	Automatic	the maximum number of recursive subdivisions
Mesh	None	how many mesh points to draw on each curve
MeshFunctions	{#1 &}	how to determine the placement of mesh points
MeshShading	None	how to shade regions between mesh points
MeshStyle	Automatic	the style for mesh points
Method	Automatic	the method to use for refining curves
PerformanceGoal	$PerformanceGoal	aspects of performance to try to optimize
PlotPoints	Automatic	initial number of sample points
PlotRange	{Full, Automatic}	the range of y or other values to include
PlotRangeClipping	True	whether to clip at the plot range
PlotStyle	Automatic	graphics directives to specify the style for each c
RegionFunction	(True &)	how to determine whether a point should be inc
WorkingPrecision	MachinePrecision	the precision used in internal computations

- Interactive labeling can be specified for curves using `Tooltip`, `StatusArea`, or `Annotation`.
- `Plot[Tooltip[{f1, f2, ...}], {x, xmin, xmax}]` specifies that the f_i should be displayed as tooltip labels for the cor curves.
- `Tooltip[f, label]` specifies an explicit tooltip label for a curve.
- `Plot` initially evaluates f at a number of equally spaced sample points specified by `PlotPoints`. Then it uses an ada choose additional sample points, subdividing a given interval at most `MaxRecursion` times.
- You should realize that with the finite number of sample points used, it is possible for `Plot` to miss features in your f your results, you should try increasing the settings for `PlotPoints` and `MaxRecursion`.

- **Step 6: Student Requests**

In[138]:=

Time to take student requests for the ListPlot command.

Change plot range?

Change axes origin?

Change plot marker?

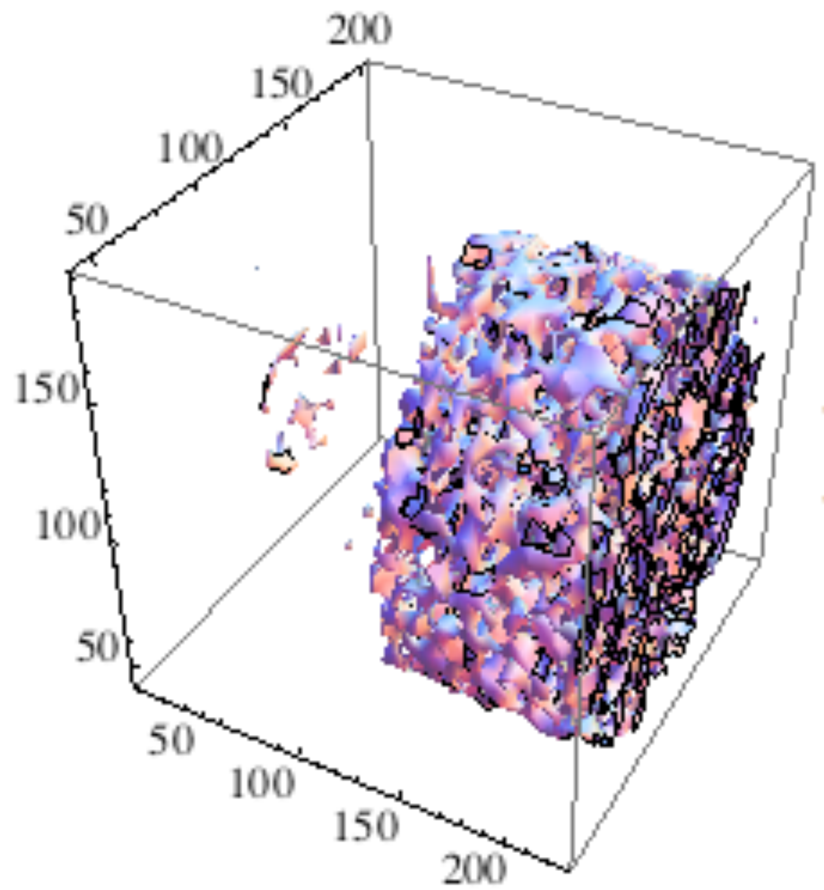
Change line width?

Add axes labels?

Add box around graph?

- **Step 7: Plot one isosurface (speed, quality)**

```
In[119]:= Timing[ ListContourPlot3D[volume, Contours → {0.02},
   MaxPlotPoints → 50,
   PerformanceGoal → "Speed"] ]
```

Out[119]= {0.387013,  }

Interrupting Calculations

There will probably be times when you want to stop *Mathematica* in the middle of a calculation. Perhaps you realize that you asked *Mathematica* to do the wrong thing. Or perhaps the calculation is just taking a long time, and you want to find out what is going on.

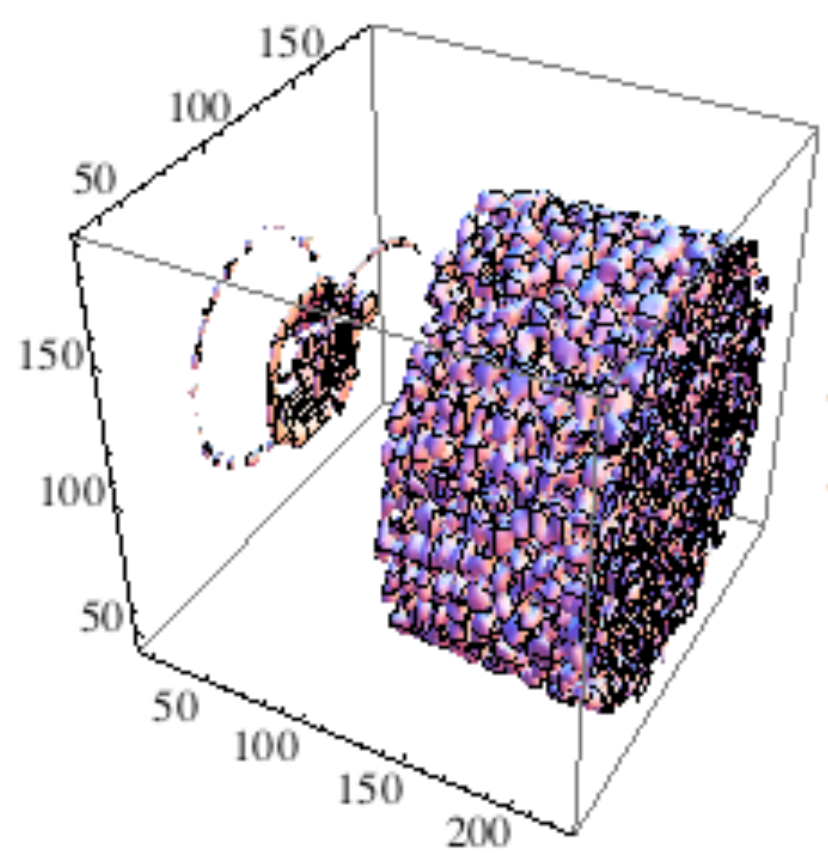
The way that you interrupt a *Mathematica* calculation depends on what kind of interface you are using.

Alt+, or Cmd+Option+.	notebook interfaces
Ctrl+C	text-based interfaces

Typical keys to interrupt calculations in *Mathematica*.

When doing some operations, it may take *Mathematica* some time to respond to your interrupt. When *Mathematica* does respond, it will give you a menu of possible things to do. This will typically include not only aborting your computation, but also entering a subsession or "dialog" to inspect its state.

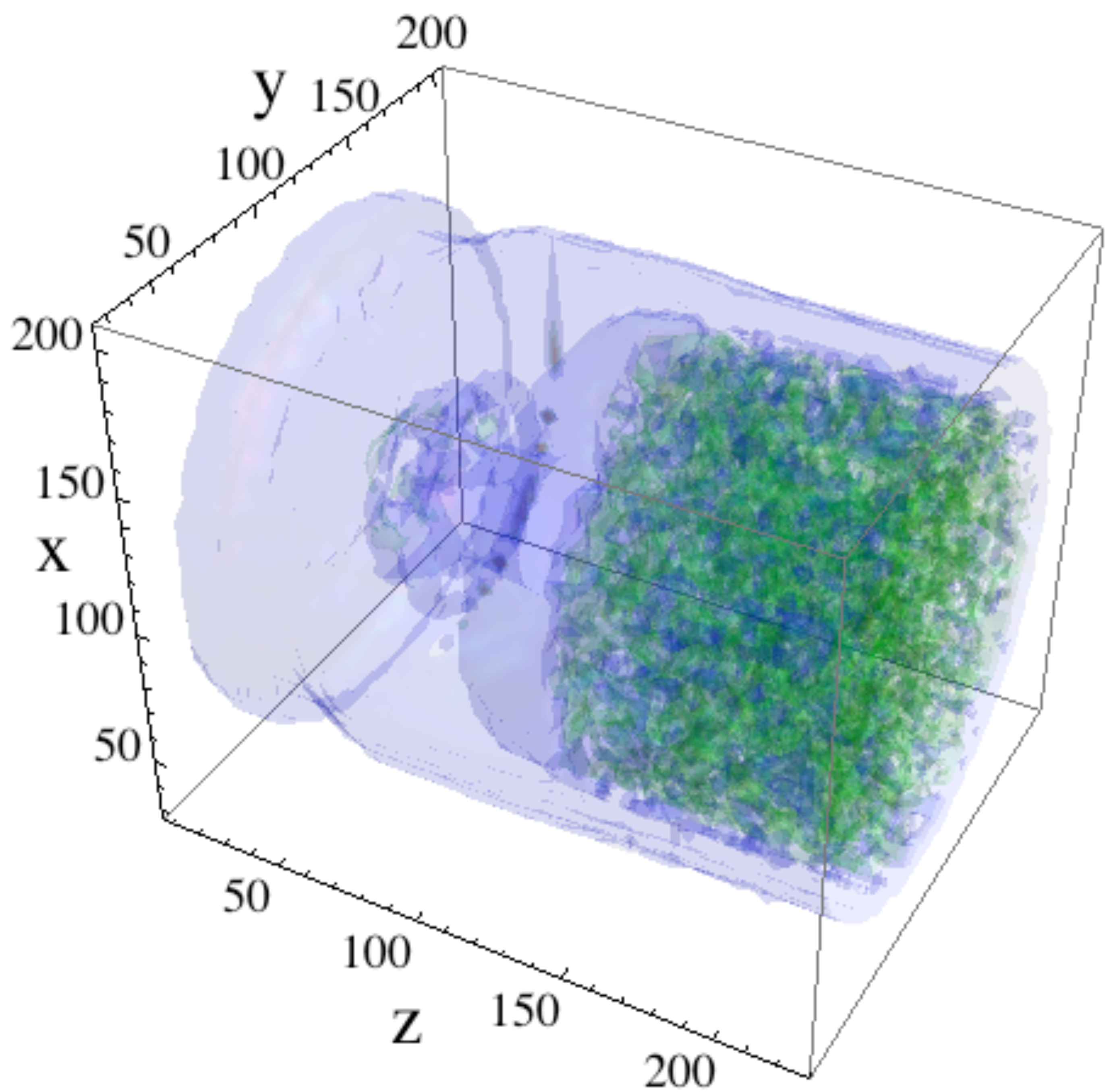
```
In[118]:= Timing[ ListContourPlot3D[volume, Contours → {0.02},
   MaxPlotPoints → 100,
   PerformanceGoal → "Quality"] ]
```

Out[118]= {3.34009,  }

- **Step 8: Plot several isosurfaces**

In[141]:=
```
contourLevels3D = {0.005, 0.02, 0.03, 0.04, 0.05};
gPrettyPlot = ListContourPlot3D[volume, Contours -> contourLevels3D, Mesh → None,
  AspectRatio → Automatic, BoxRatios → Automatic, Axes → True,
  AxesLabel → {Style["z", FontSize → 24], Style["y", FontSize → 24], Style["x", FontSize → 24]},
  LabelStyle → Directive[Black, FontSize → 16],
  ContourStyle → {Directive[Blue, Opacity[0.1], Specularity[White, 30]],
    Directive[Green, Opacity[0.1], Specularity[White, 30]],
    Directive[Yellow, Opacity[0.1], Specularity[White, 30]],
    Directive[Orange, Opacity[0.1], Specularity[White, 30]],
    Directive[Red, Opacity[0.1], Specularity[White, 30]]},
  BoundaryStyle → None,
  MaxPlotPoints → 50,
  PerformanceGoal → "Quality"]
```

Out[142]=



In[148]:=
```
Export[NotebookDirectory[] <> "temp.jpg", gPrettyPlot, "JPG", ImageSize → 800]
```

Out[148]=
```
/Users/lesbutler/Documents/h4581/temp.jpg
```

- **Step 3: Import the binary file**

```
In[149]:= 243 × 243 × 227
Out[149]= 13 404 123

In[162]:= listOfFilenames = FileNames["volume*bin", NotebookDirectory[]]
Out[162]= {/Users/lesbutler/Documents/h4581/volume_bullet_p134_uint16.bin}
```

- **Step 3a: explore partition step by step**

```
In[165]:= volume = BinaryReadList[listOfFilenames[[1]], "UnsignedInteger16"]
          Dimensions[volume]
Out[166]= {13 404 123}

In[167]:= temp = Partition[volume, 227];
          Dimensions[temp]
Out[168]= {59 049, 227}

In[169]:= temp = Partition[temp, 243];
          Dimensions[temp]
Out[170]= {243, 243, 227}

In[171]:= volume = temp;
          Clear[temp]
          {rows, columns, slices} = Dimensions[volume]
Out[173]= {243, 243, 227}
```

BinaryReadList

`BinaryReadList["file"]`
reads all remaining bytes from a file, and returns them as a list of integers from 0 to 255.

`BinaryReadList["file", type]`
reads objects of the specified type from a file, until the end of the file is reached. The list of objects read is returned.

`BinaryReadList["file", {type1, type2, ...}]`
reads objects with a sequence of types, until the end of the file is reached.

`BinaryReadList["file", types, n]`
reads only the first n objects of the specified types.

BinaryRead

`BinaryRead[stream]`
reads one byte of raw binary data from an input stream, and returns an integer from 0 to 2

`BinaryRead[stream, type]`
reads an object of the specified type.

`BinaryRead[stream, {type1, type2, ...}]`
reads a sequence of objects of the specified types.

MORE INFORMATION

- Possible types to read are:

"Byte"	8-bit unsigned integer
"Character8"	8-bit character
"Character16"	16-bit character
"Complex64"	IEEE single-precision complex number
"Complex128"	IEEE double-precision complex number
"Complex256"	IEEE quad-precision complex number
"Integer8"	8-bit signed integer
"Integer16"	16-bit signed integer
"Integer24"	24-bit signed integer
"Integer32"	32-bit signed integer
"Integer64"	64-bit signed integer
"Integer128"	128-bit signed integer
"Real32"	IEEE single-precision real number
"Real64"	IEEE double-precision real number
"Real128"	IEEE quad-precision real number
"TerminatedString"	null-terminated string of 8-bit characters
"UnsignedInteger8"	8-bit unsigned integer
"UnsignedInteger16"	16-bit unsigned integer
"UnsignedInteger24"	24-bit unsigned integer
"UnsignedInteger32"	32-bit unsigned integer
"UnsignedInteger64"	64-bit unsigned integer
"UnsignedInteger128"	128-bit unsigned integer

- **Step 3b: do the both partitions in one big step**

In[174]:=
```
volume = BinaryReadList[listOfFilenames[[1]], "UnsignedInteger16"];
volume = Partition[Partition[volume, 227], 243];
Dimensions[volume]
```

Out[176]=
```
{243, 243, 227}
```

- **Step 3c: make slice and line plots**

In[177]:=
```
aSlice = volume[[Round[rows / 2], All, All]];
Dimensions[aSlice]
```

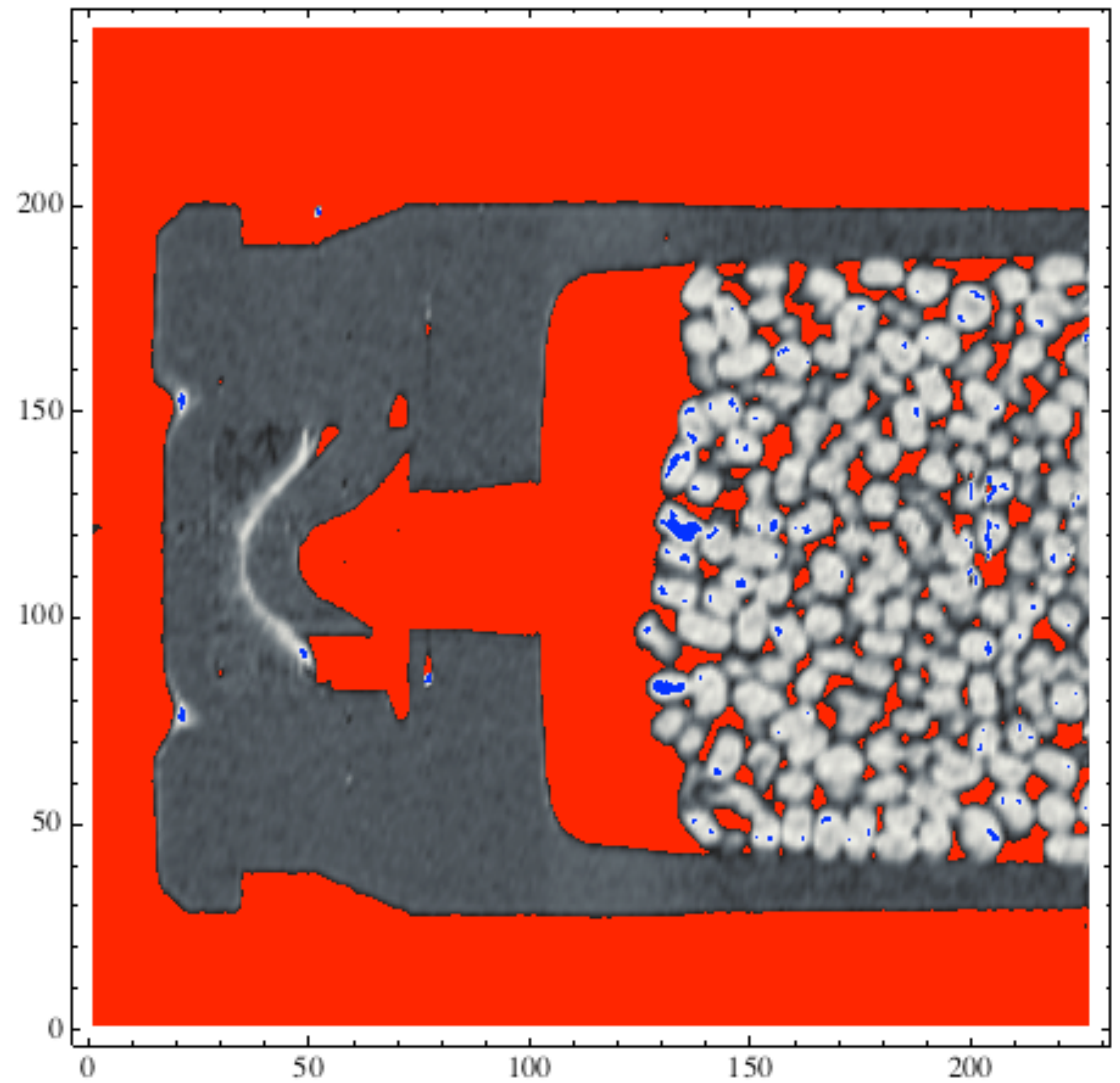
Out[178]= {243, 227}

In[179]:=
```
{Min[aSlice], Max[aSlice]}
```

Out[179]= {14 855, 44 899}

In[180]:=
```
gSlice = ListDensityPlot[aSlice, ColorFunction → "GrayTones", PlotRange → {All, All, {28 000, 33 000}},
  ClippingStyle → {Red, Blue}]
```

Out[180]=



In[181]:=
```
aLine = volume[[Round[rows / 2], Round[columns / 2], All]];
Dimensions[aLine]
{Min[aLine], Max[aLine]}
gLine = ListPlot[aLine, PlotRange → {All, All}];
gLineV2 = ListPlot[aLine, PlotRange → {All, All}, Joined → True];
Show[{gLine, gLineV2}]
```

Out[182]= {227}

Out[183]= {14 855, 34 924}

Out[186]=