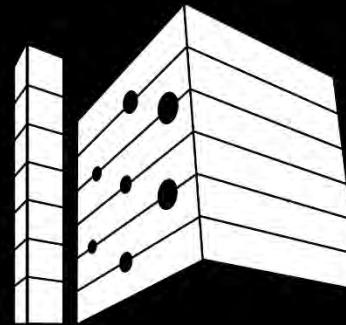


MOLECULAR
FOUNDRY



Making Figures and Movies with Code

Colin Ophus

NCEM, Molecular Foundry, Lawrence Berkeley National Laboratory

Tutorial 4 – Data Analysis and Modern Visualization

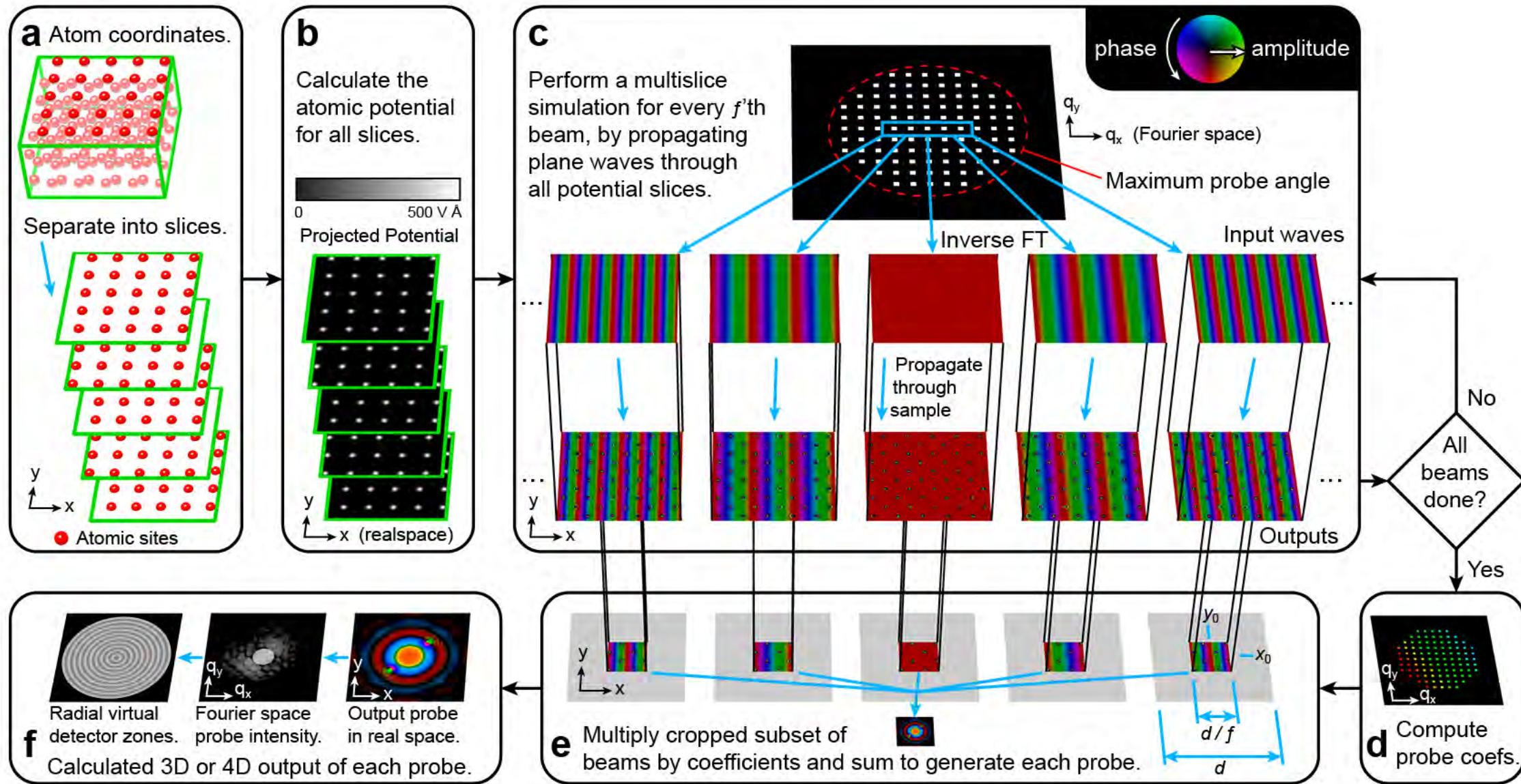
APS March Meeting – Pre-Meeting

Molecular Foundry – Proposals due Mar 31 2021 – foundry.lbl.gov

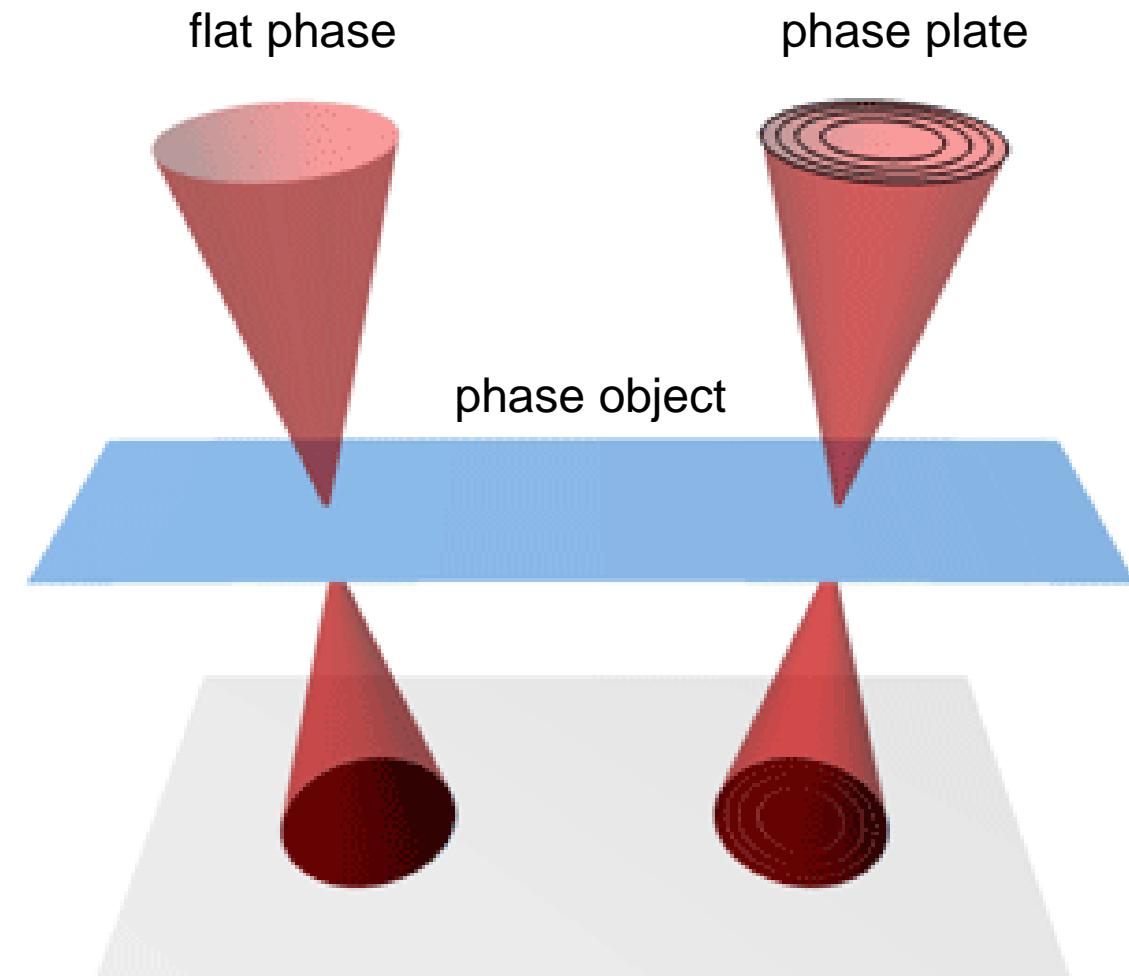
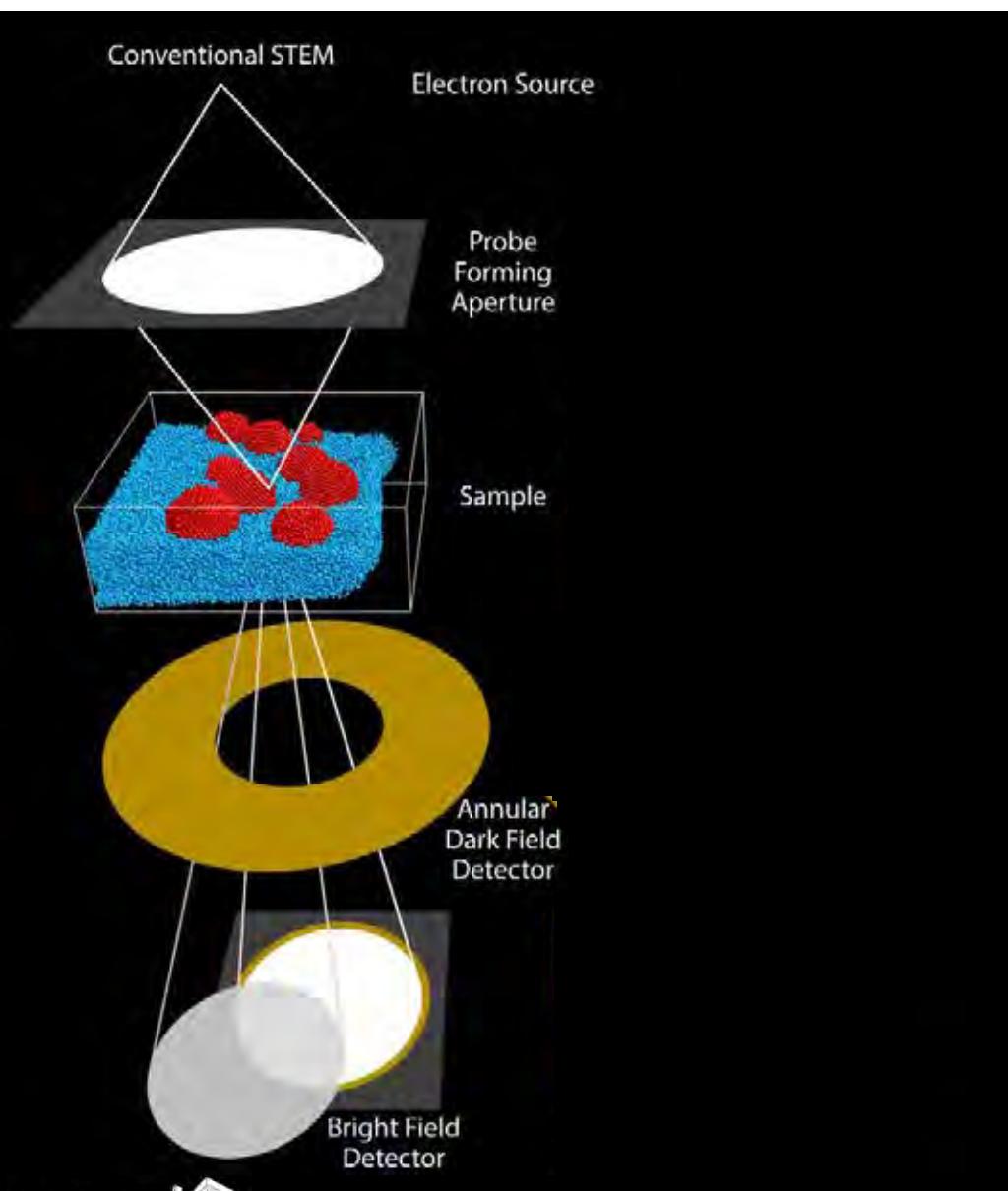


- We are a **user facility** at the Berkeley Lab, operated by the US Department of Energy.
- **Anyone** can submit a proposal (including for **computation**, **simulation** or **analysis!**).
- If accepted by independent review board, access to microscopes and staff is **free**.

Programmatic Visualization Examples



Programmatic Visualization Examples



Phase plate interference pattern can generate linear interference.

Programmatic Visualization Examples

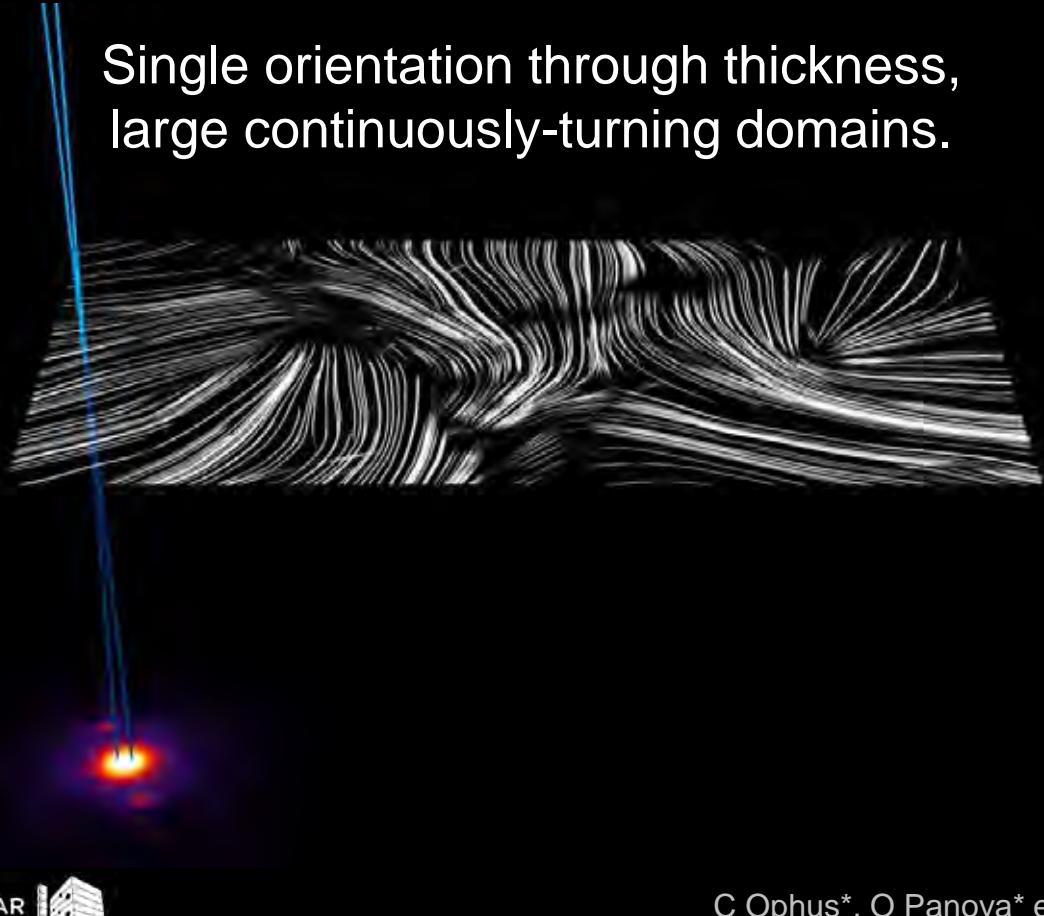


Programmatic Visualization Examples

Visualization of the two morphologies:

No additive to T1:

Single orientation through thickness,
large continuously-turning domains.

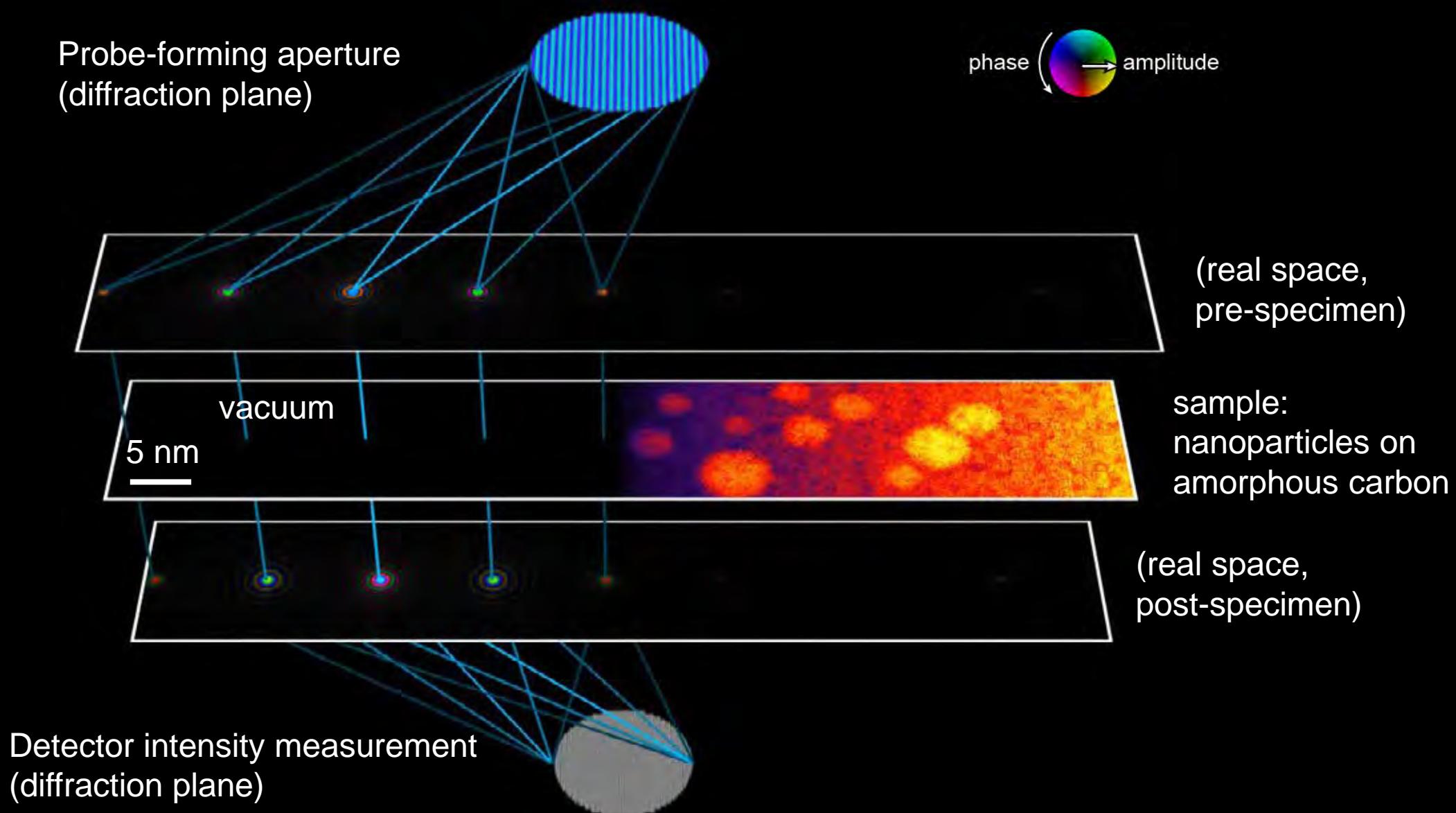


DIO additive to T1:

Multiple orientations through thickness,
small single-orientation domains.



Programmatic Visualization Examples

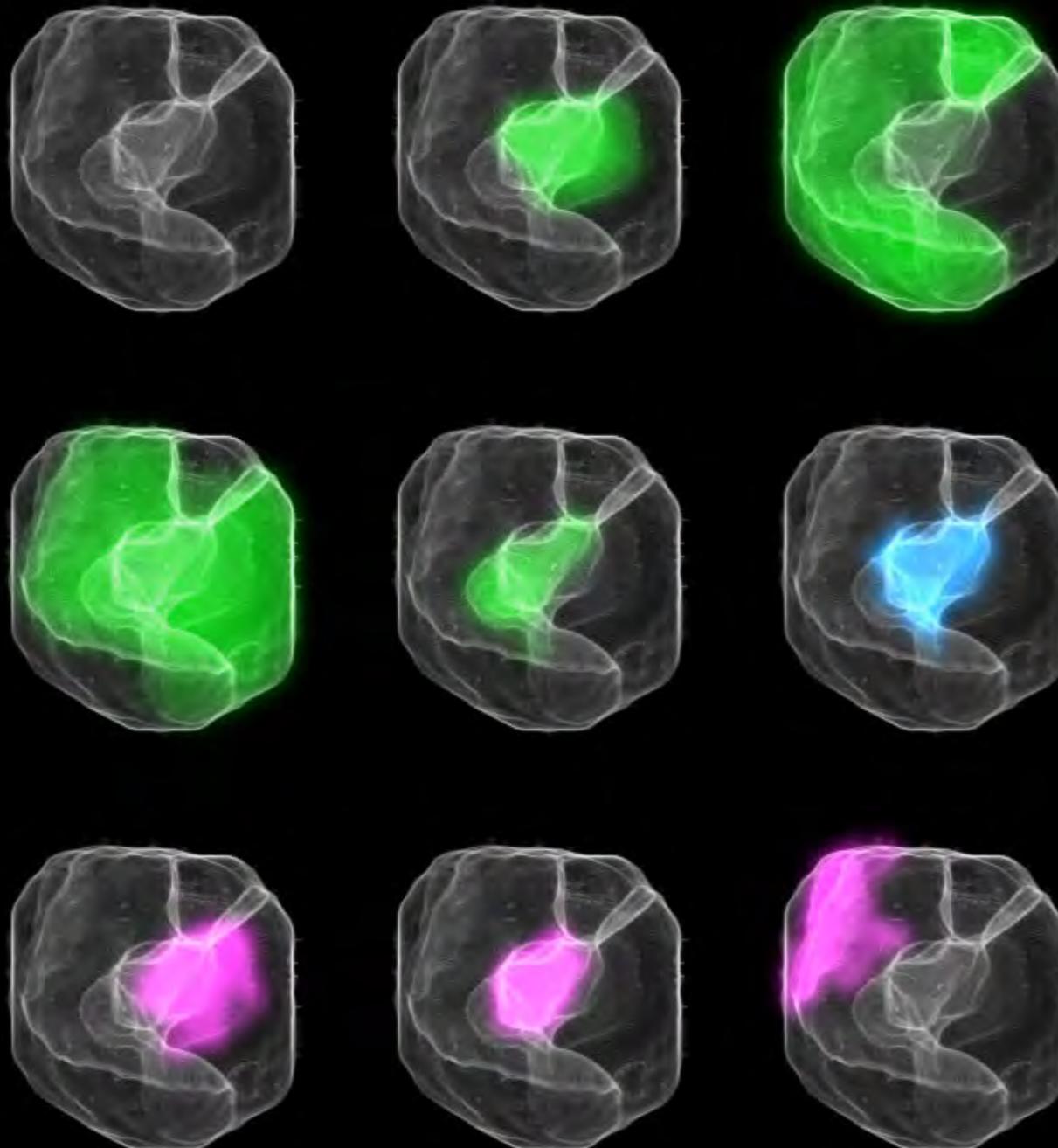


Programmatic Visualizations Example

ng a bit, since
elements and
ts were created
al render was
der.



Programmatic Visualization Examples

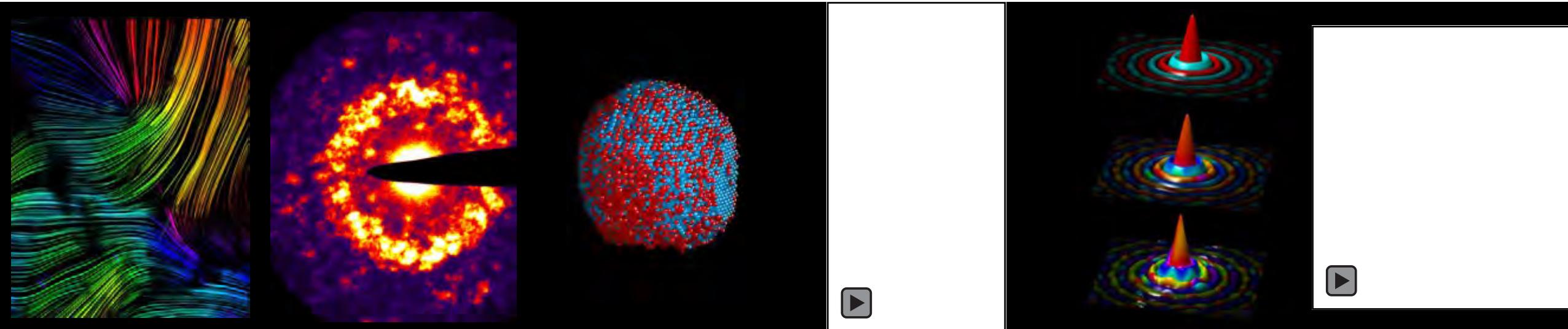


- Same data as previous slide, but visualized in a completely different way – grains become volumetric renders.

Programmatic Visualization Examples

Each of these visualizations was generated almost entirely from code –
and not very much code either!

Outline



1

Basic concepts
when making
figures and
movies – raster
vs vector.

2

Scaling,
coloring and
saving scalar
and vector
imaging data.

3

Plotting **points**
in 3D, highlight
internal features
– e.g. atomic
coordinates.

4

Animating 3D
points, making
movies
– e.g. atomic
coordinates.

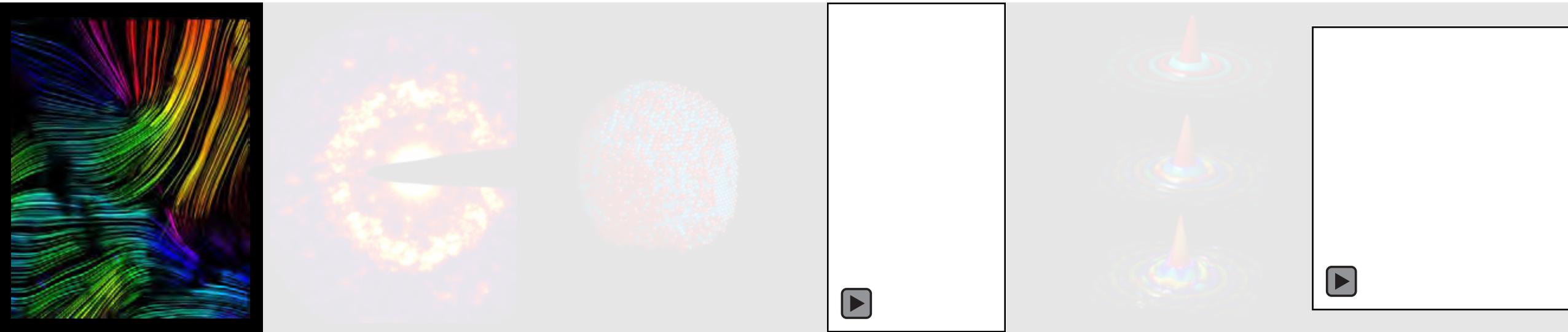
5

Building physics
visualization
and simulation
directly into
figures.

6

Making simple
animations to
explain
concepts in
physics.

Outline



1

Basic concepts
when making
figures and
movies – raster
vs vector.

2

Scaling,
coloring and
saving scalar
and vector
imaging data.

3

Plotting **points**
in 3D, highlight
internal features
– e.g. atomic
coordinates.

4

Animating 3D
points, making
movies
– e.g. atomic
coordinates.

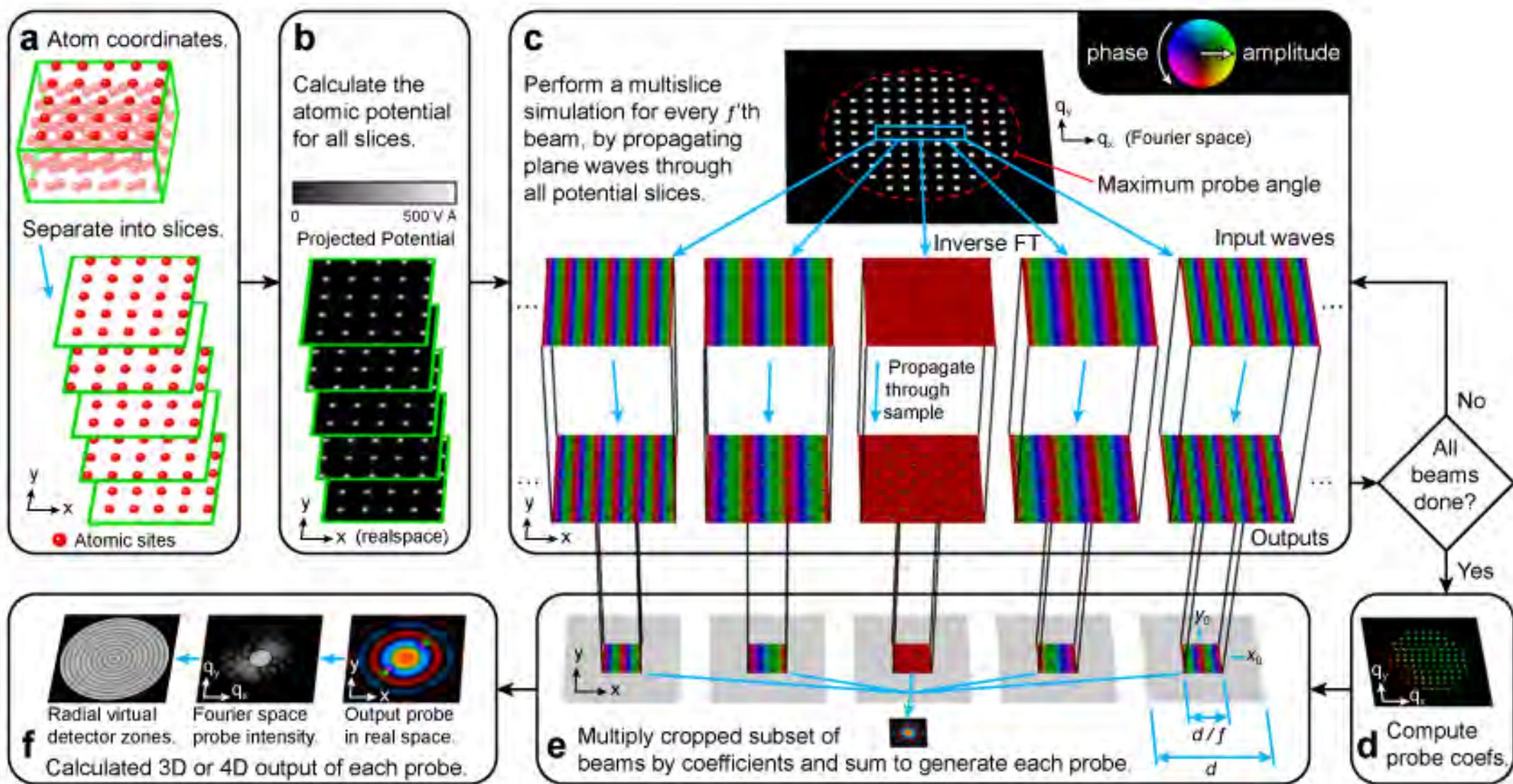
5

Building physics
visualization
and simulation
directly into
figures.

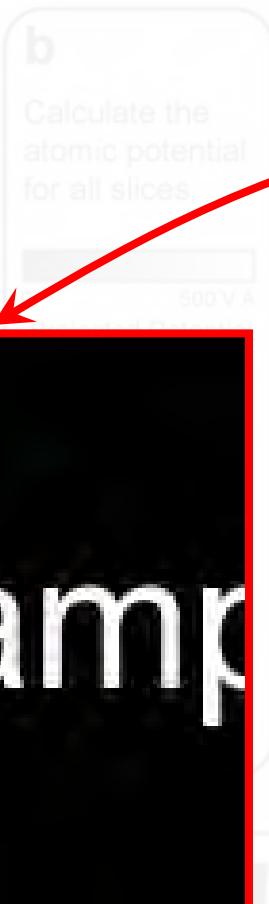
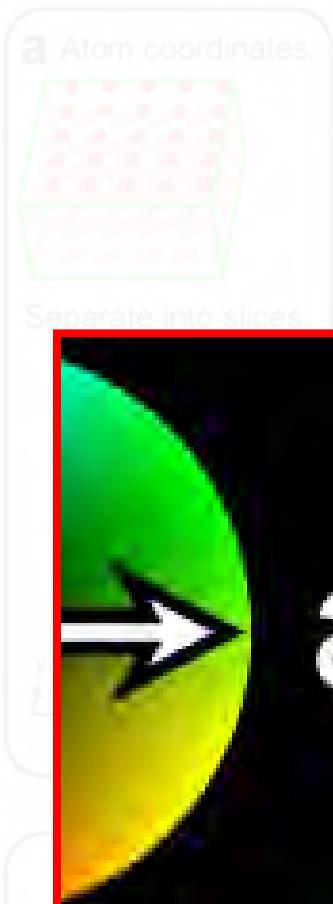
6

Making simple
animations to
explain
concepts in
physics.

Basics – Vector Versus Raster (Bitmap)



Basics – Vector Versus Raster (Bitmap)



Vector formats can easily be “rendered” into raster.

The reverse direction is not possible however!

Vector format – objects stored as vertices, curves, faces, patches ...

Raster format – RGB values stored in a grid.

Basics – Vector and Raster File Formats

Vector

ideal format

PDF (portable document format)

runner up formats

SVG (scalable vector graphics)
EPS (encapsulated post script)

don't use formats

AI (adobe illustrator)
CD (CorelDraw)
EMF / WMF (Windows vectors)
DOC, PPTX, Keynote
DXF (or any other drafting format)

Raster

PNG (for lossless images)
JPG (for lossy images)

TIFF (can contain metadata)
MPG, MP4, AVI, WEBM

BMP, PSD, TGA, ...
GIF (Maybe the worst format ever? Terrible for animations too!
Does support transparency ...)

Basics – Vector and Raster Editing Software

Vector

Free programs

Inkscape

powerful, some quirks, SVG files

Good programs

Illustrator

high learning curve

CorelDraw

easy to learn

Bad programs (for editing figures)

Microsoft Office Suite
(Powerpoint)

low quality output, inconsistent support, missing most advanced tools, cannot control resolution

OfficeLibre (Open Office)

low quality output, inconsistent support, missing most advanced tools, cannot control resolution

Raster

Gimp

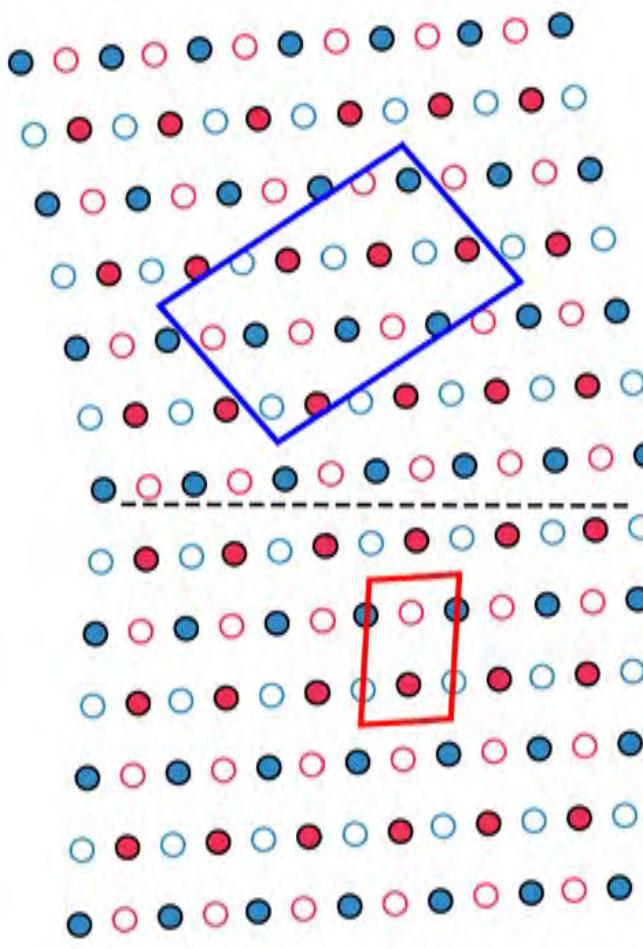
powerful, duplicates most photoshop features

www.photopea.com

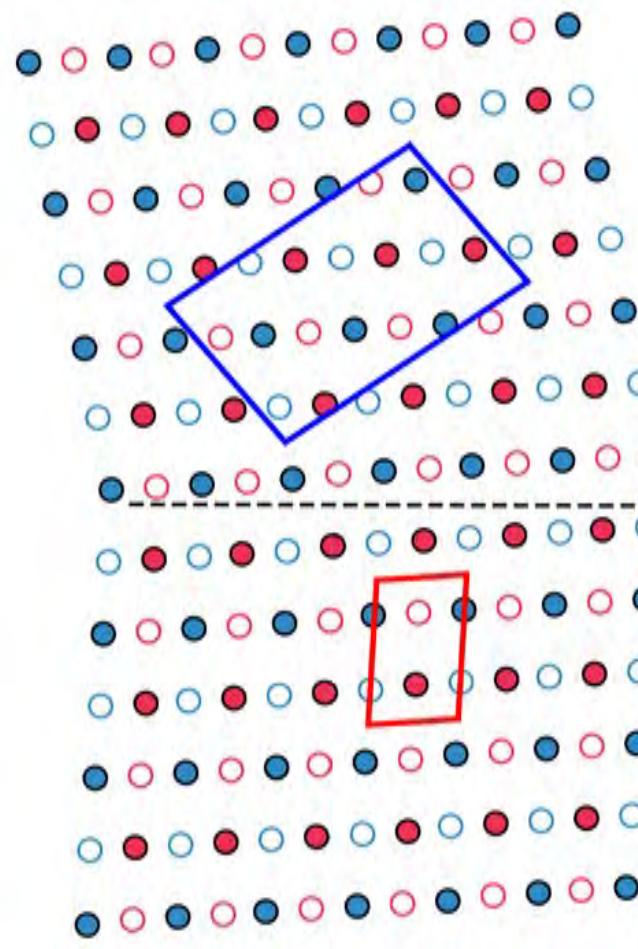
Photoshop

powerful, easy to use

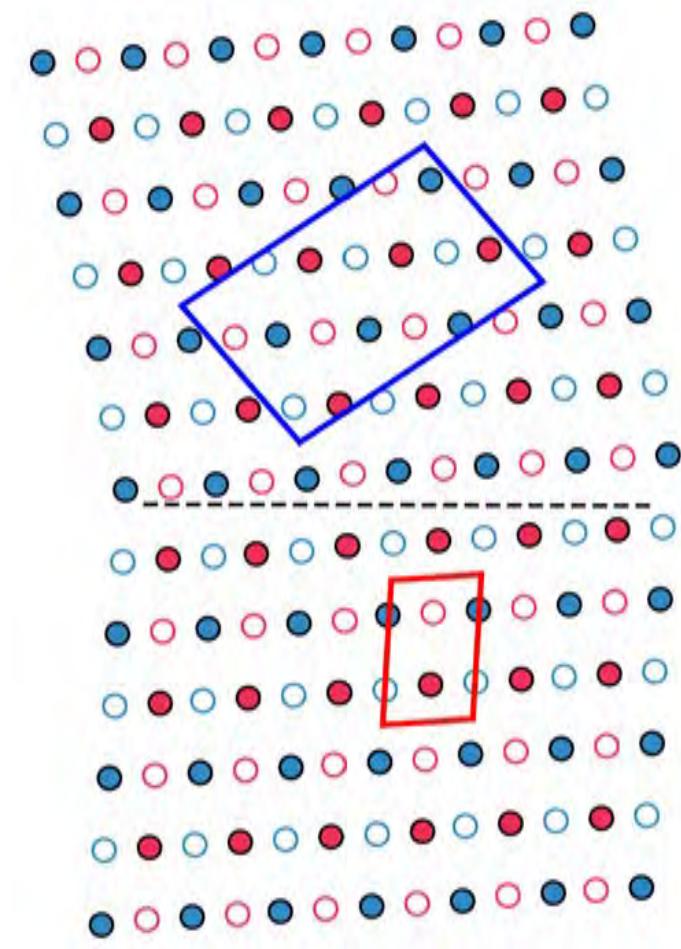
Basics – Vector and Raster File Sizes



PDF – 21 kilobytes



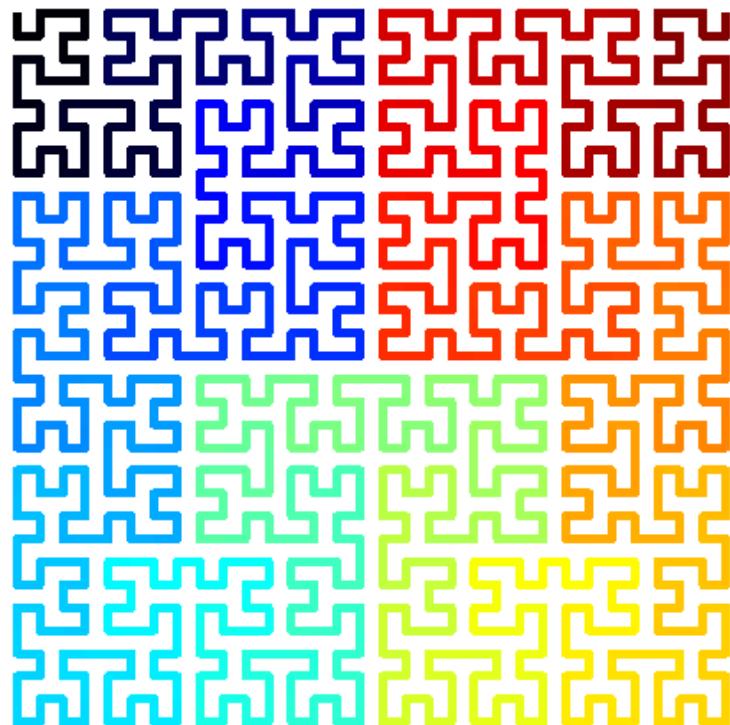
PNG – 187 kilobytes



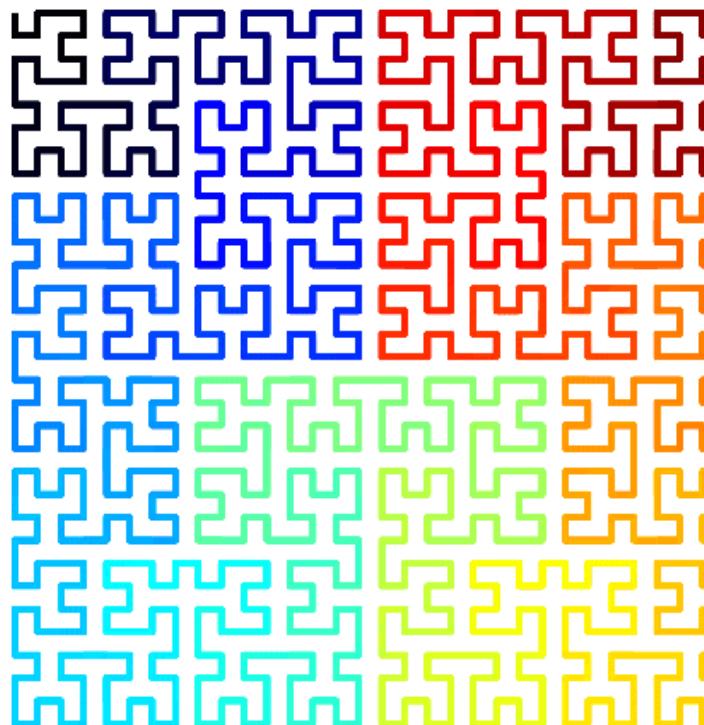
JPG – 352 kilobytes

For relatively simple vector objects, PDF is the best (in general vector is also better than raster!)

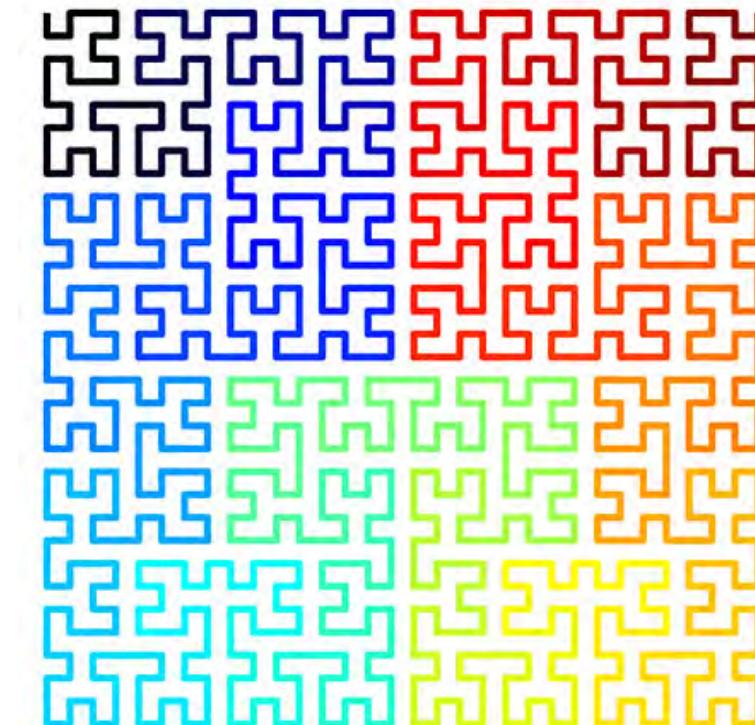
Basics – Vector and Raster File Sizes



PDF – 117 kilobytes



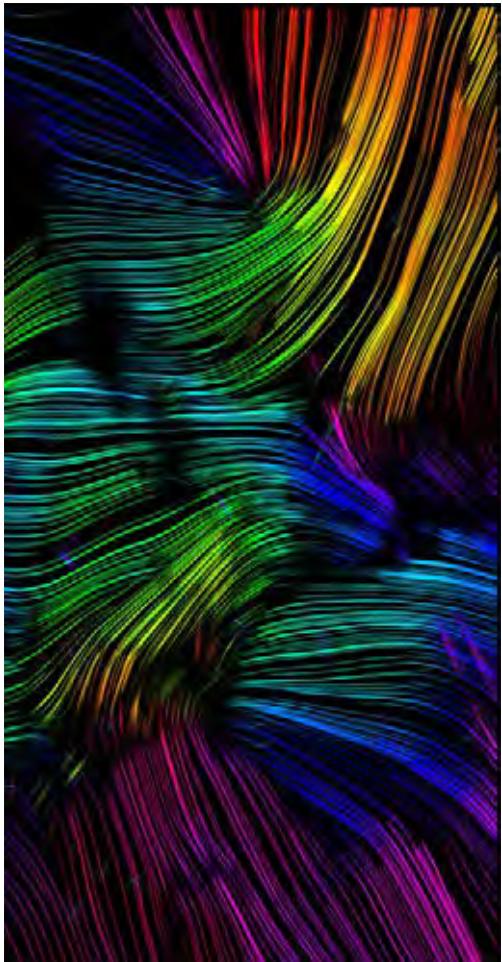
PNG – 15 kilobytes



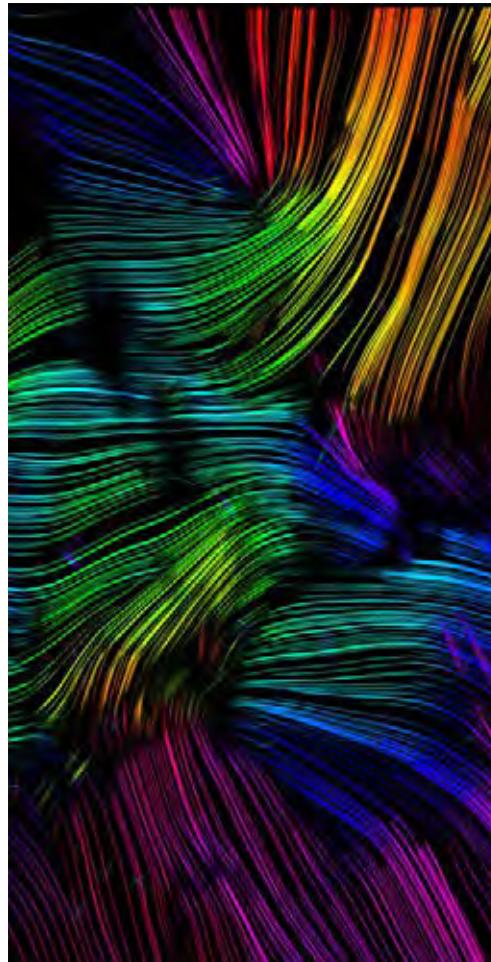
JPG – 123 kilobytes

For images with few colors (<256), flat square regions, PNG compression is very good.

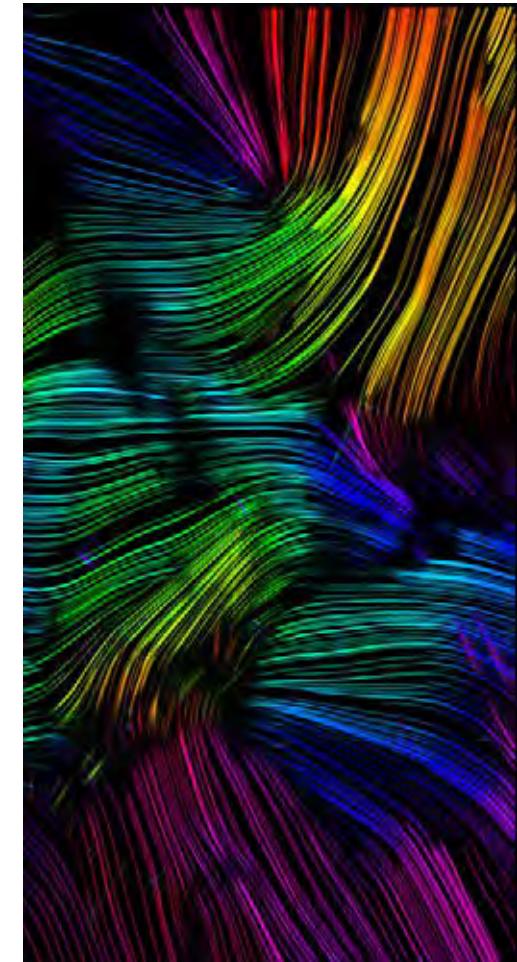
Basics – Vector and Raster File Sizes



PDF – 10 851 kilobytes



PNG – 1 710 kilobytes



JPG – 576 kilobytes

If the image contains many “high energy” regions, many colors, lossy compression such as JPG is best.

Basics – Zooming into Vector and Raster



bitmap,
nearest
neighbor



bitmap,
bilinear



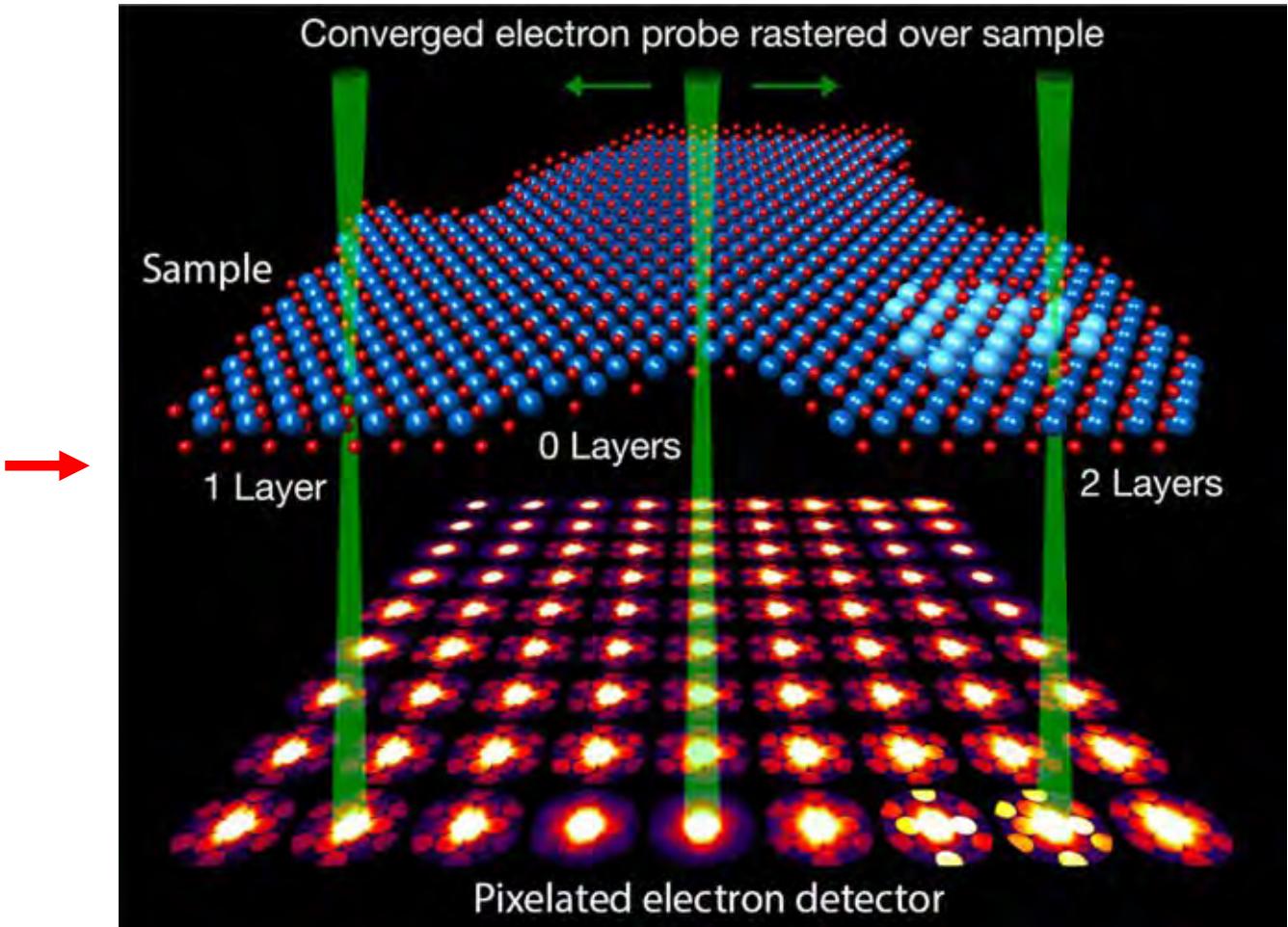
vector

Basics – Vector and Raster Figures

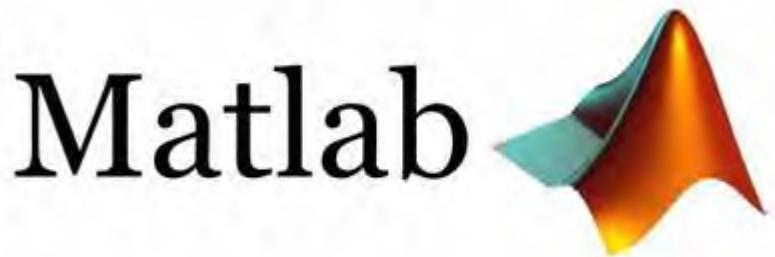
Take home messages:

- Virtually every figure should be saved in **vector format** (PDF).
- **Vector** files can mix & match **images**, **text**, and **curves** such as lines & markers.
- **DO NOT** burn labels such as (a)/(b)/..., scale markers, overlays into images.
- Much easier to “remix” figures later if you cleanly **separate all objects**.
- Sometimes a PDF file size is just too large, in this case “**render**” a bitmap output.
(e.g. arXiv uploads must be <10 megs – 50 megs total)

Creating Visualization / Figures With Programming



What Language Should You Use For Figures?



Advantages

- Very large number of functions, libraries, toolboxes built in, even more on file exchange.
- Easy to use, quite fast (though not C speed)
- Very powerful 2D and 3D plotting tools.
- No compiler needed (interpreted).
- Widely used in scientific research.

Disadvantages

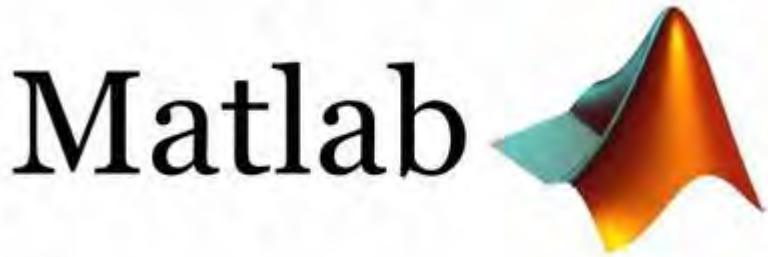
- Expensive license (cheap for academics).
- Poor support for GUIs, Java front end crashes.
- Slow development relative to open source.
- Slow (offset by Cython, Numba, PyTorch).
- Poor memory efficiency.
- Unreliable versioning ... e.g. Python 2 vs 3

What Language will this Tutorial Use?



Today's tutorial will use Matlab, but concepts apply just as easily to python or any language!

What Language will this Tutorial Use?



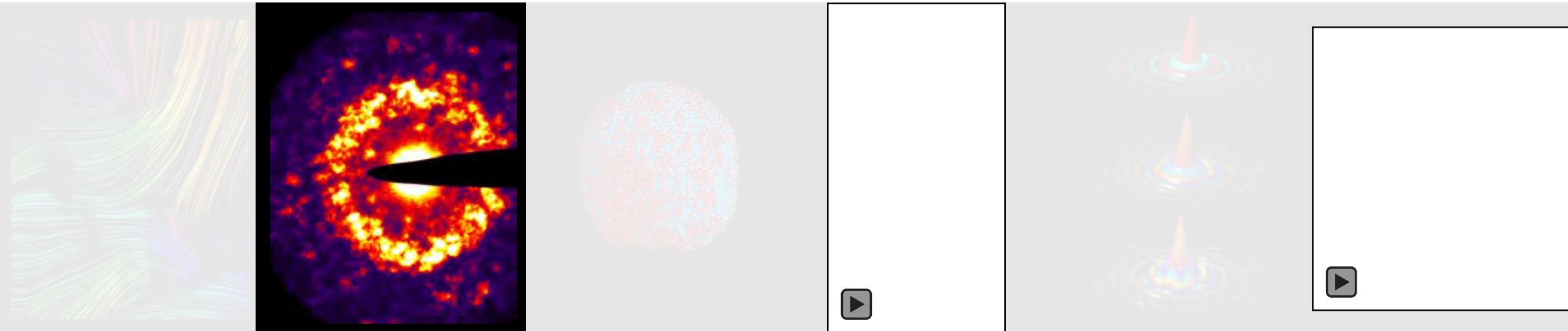
File Exchange

- export_fig – often better raster image export than built in Matlab “save.”
www.mathworks.com/matlabcentral/fileexchange

Other notes

- Matlab used to have terrible PDF support – better results saving as EPS, or using export_fig. Much improved recently though! (python/matplotlib has excellent PDF support)
- Matlab has terrible movie support – don’t even bother. Instead, export images and then use external tools to create / re-encode movies:
 - FFMPEG www.ffmpeg.org – very widespread, command line interface.
 - Handbrake handbrake.fr – Movie won’t play in PPT? Easy to use, GUI.
- Always save **both the data and code** to generate plots – your future self will be grateful!

Outline



1

Basic concepts
when making
figures and
movies – raster
vs vector.

2

Scaling,
coloring and
saving scalar
and vector
imaging data.

3

Plotting **points**
in 3D, highlight
internal features
– e.g. atomic
coordinates.

4

Animating 3D
points, making
movies
– e.g. atomic
coordinates.

5

Building physics
visualization
and simulation
directly into
figures.

6

Making simple
animations to
explain
concepts in
physics.

Making Image Figures from Scalar Data

Very common application – scale contrast, output imaging data in a raster file format.

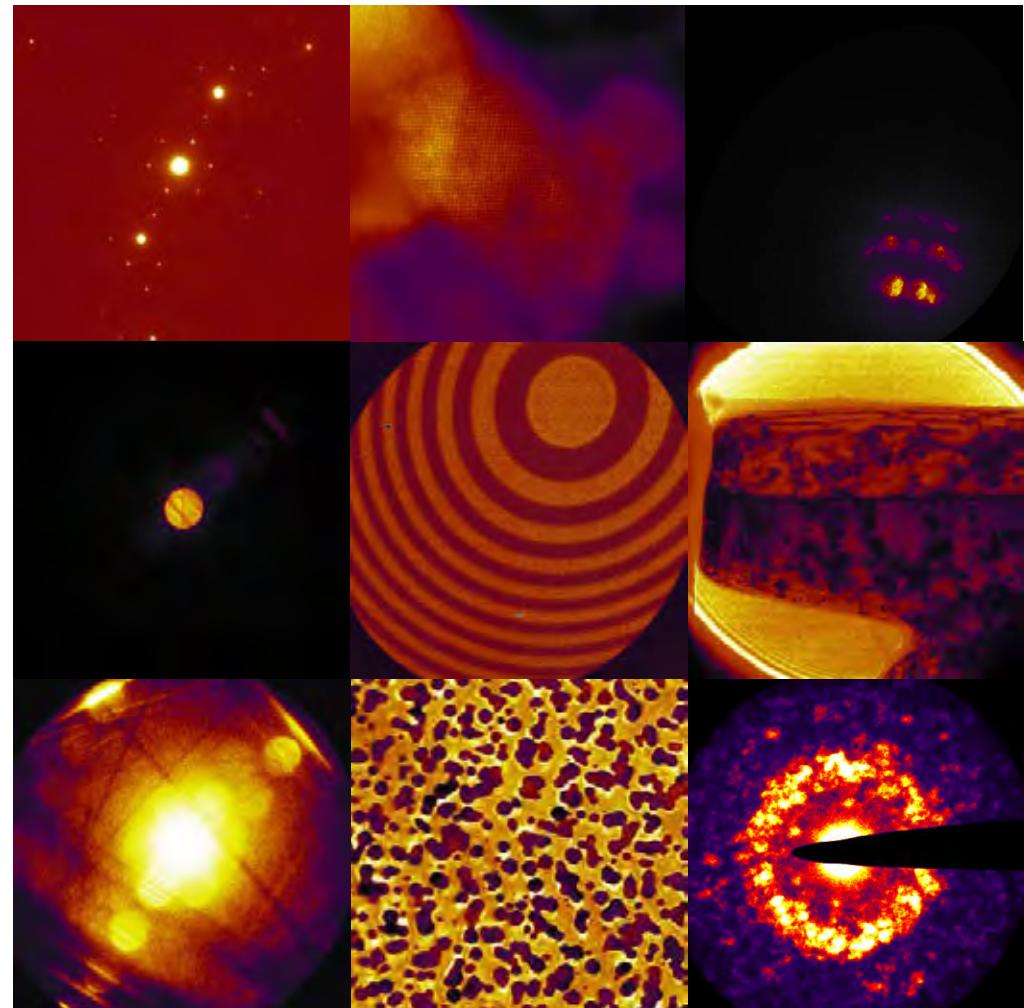
Loading the data in Matlab:

```
>> load('example_02_various_images.mat')
```

Matlab has a function for saving image data:

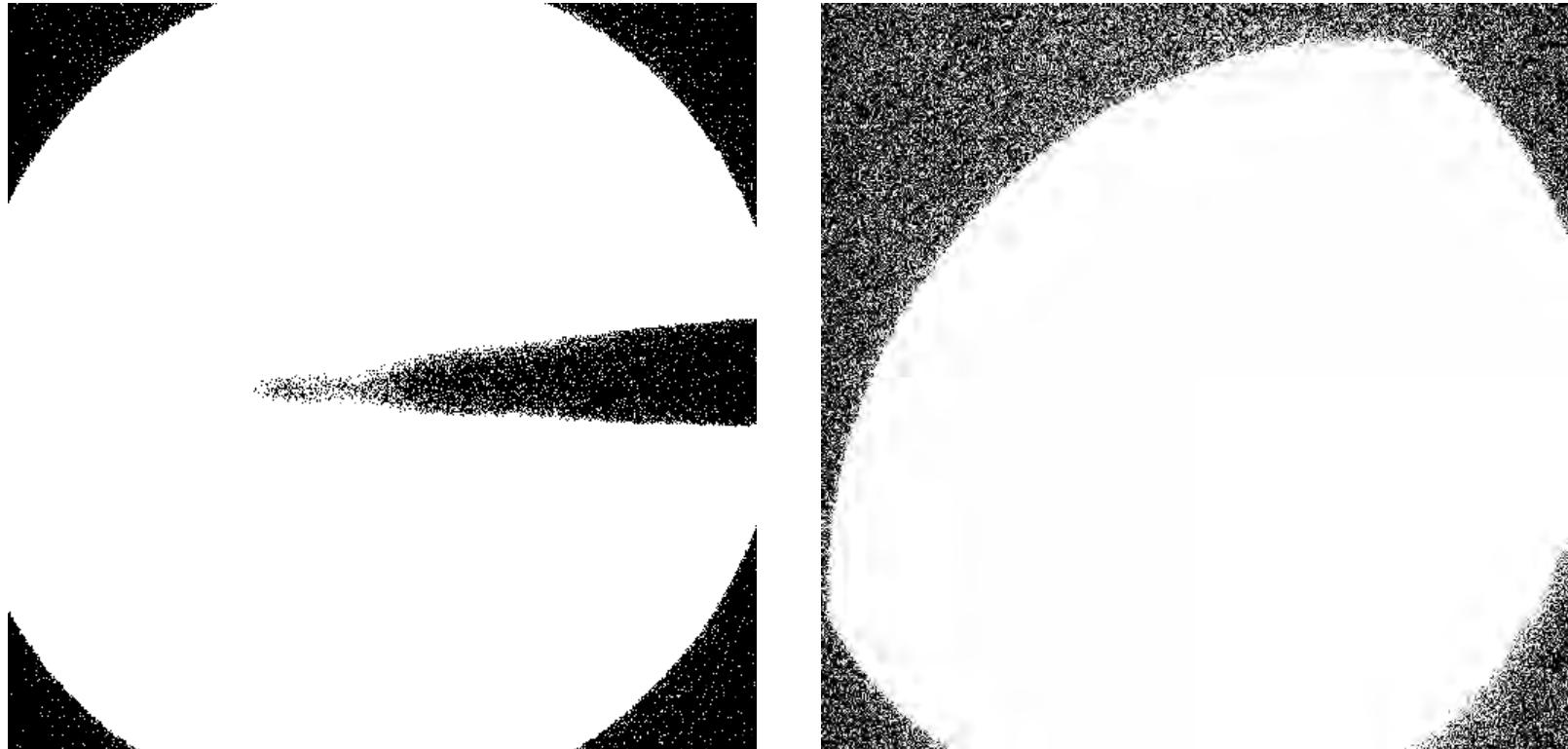
```
>> imwrite( imageArray , fileName );
```

Let's try it out on the sample images.



Making Image Figures from Scalar Data

Very common application – scale contrast, output imaging data in a raster file format:



```
imwrite(image01_UO2_01,'image01.png');
```

```
imwrite(image02_amor_diff,'image02.png');
```

```
imwrite(image03_diff,'image03.png');
```

Making Image Figures from Scalar Data

Very common application – scale contrast, output imaging data in a raster file format:



```
imwrite(image04_pillar,'image04.png');
```



```
imwrite(image05_rings01,'image05.png');
```

```
>> imwrite(image06_gold,'image06.png');
Warning: Data loss and unexpected result
> In imwrite (line 447)
Error using writepng>parseInputs (line 1)
Expected input to be one of these types

double, single, logical, uint8, uint16

Instead its type was int32.
Error in writepng (line 20)
[results, unmatched] = parseInputs(data);
Error in imwrite (line 472)
    feval(fmt_s.write, data, map, f.
```

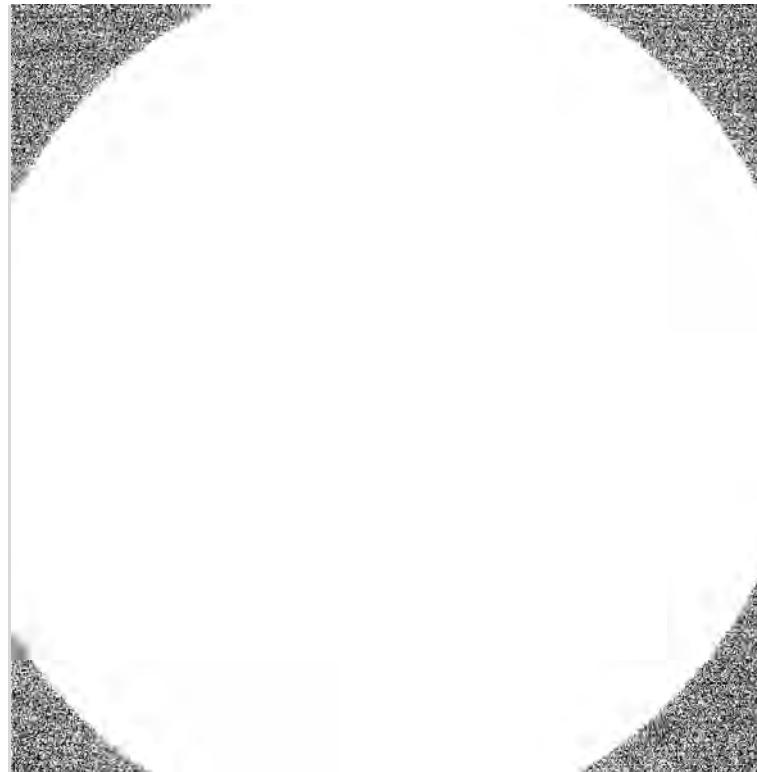
```
imwrite(image06_gold,'image06.png');
```

Making Image Figures from Scalar Data

Very common application – scale contrast, output imaging data in a raster file format:



```
imwrite(image07_diff01,'image07.png');
```



```
imwrite(image08_CBED01,'image08.png');
```

```
imwrite(image09_Pt,'image09.png');
```

Making Image Figures from Scalar Data

Convert to floating point, scale intensity from minimum to maximum value:

```
function [ ] = plotImageFullRange( imageInput, fileNameBase )  
  
% convert to double floating point  
imageInput = double( imageInput );  
  
% scale image from 0 to 1  
imageInput(:) = imageInput - min( imageInput(:) );  
imageInput(:) = imageInput / max ( imageInput(:) );  
  
% output file  
fileName = [ fileNameBase '.png' ];  
imwrite( round(imageInput*255)+1 , inferno , filename );
```

scale image intensity
to be from 1 to 256

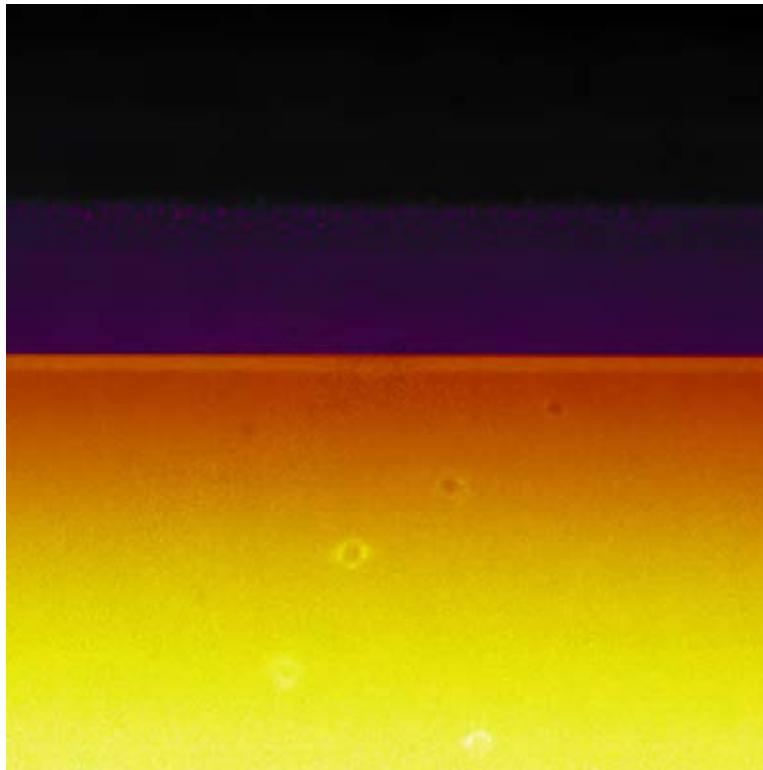


colormap: min value (1) max value (256)

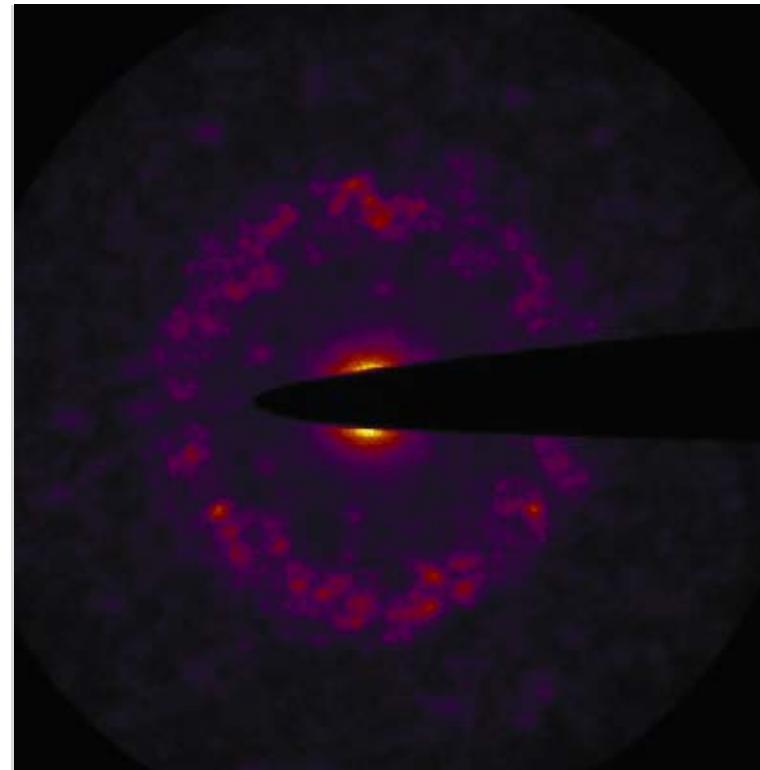


Making Image Figures from Scalar Data

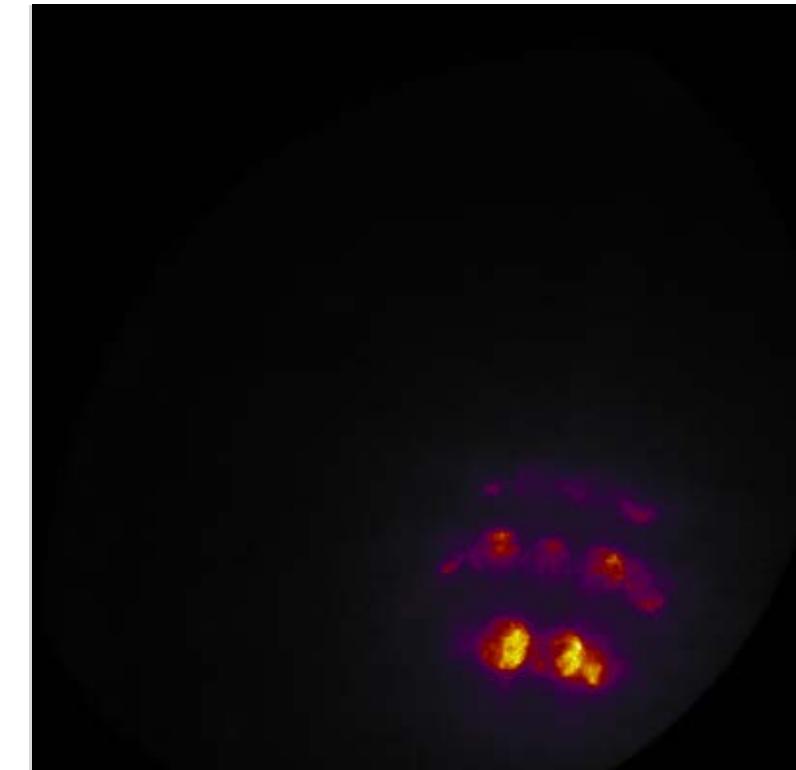
Convert to floating point, scale intensity from minimum to maximum value:



```
plotImageFullRange(image01_UO2_01,'image01');
```



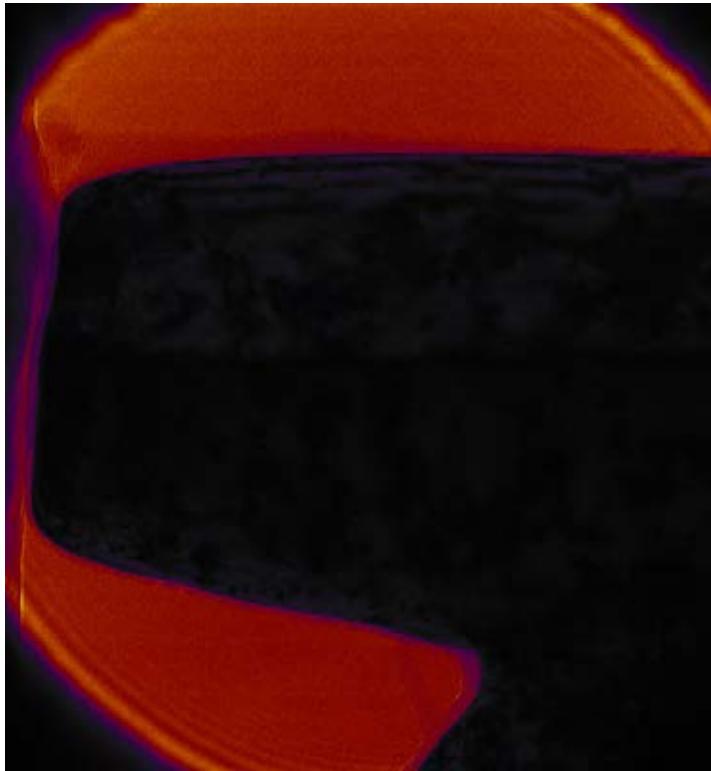
```
plotImageFullRange(image02_amor_diff,'image02');
```



```
plotImageFullRange(image03_diff,'image03');
```

Making Image Figures from Scalar Data

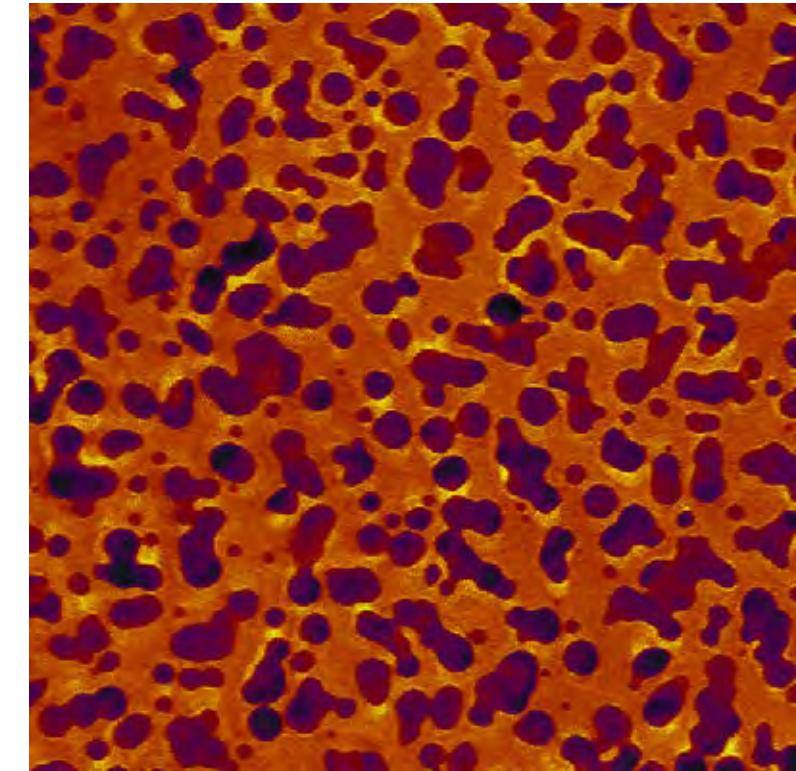
Convert to floating point, scale intensity from minimum to maximum value:



```
plotImageFullRange(image04_pillar,'image04');
```



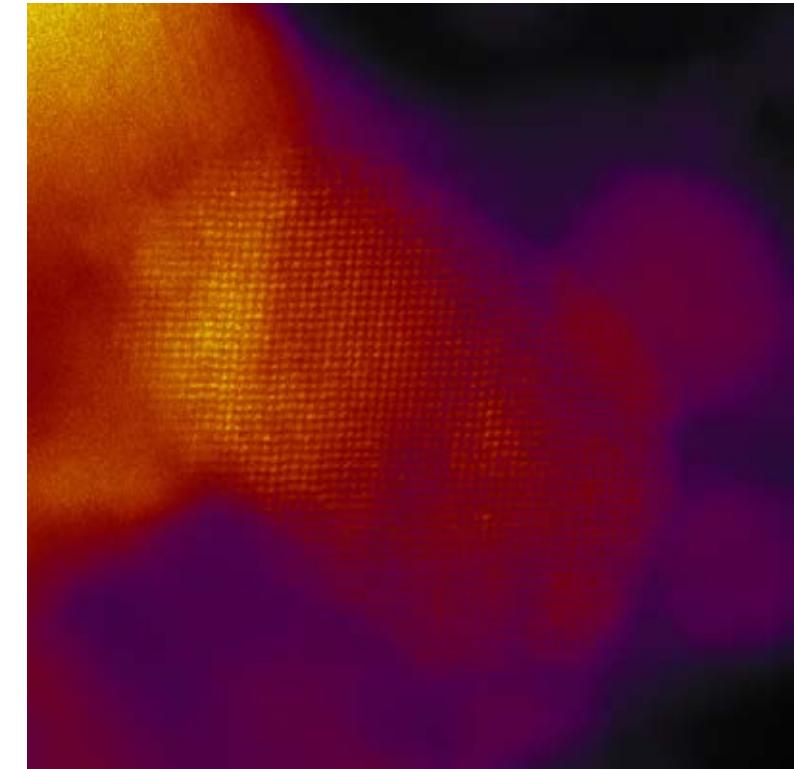
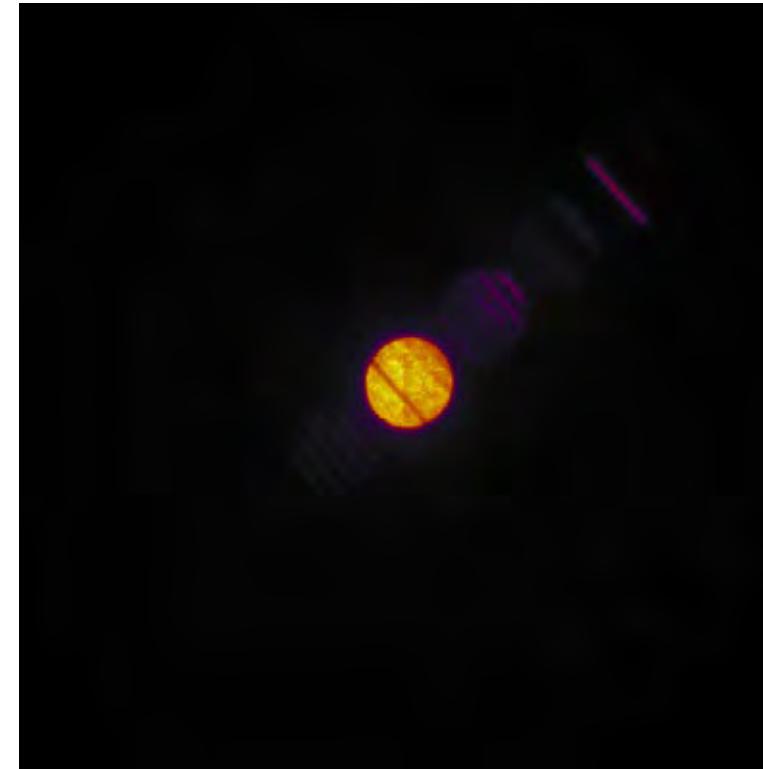
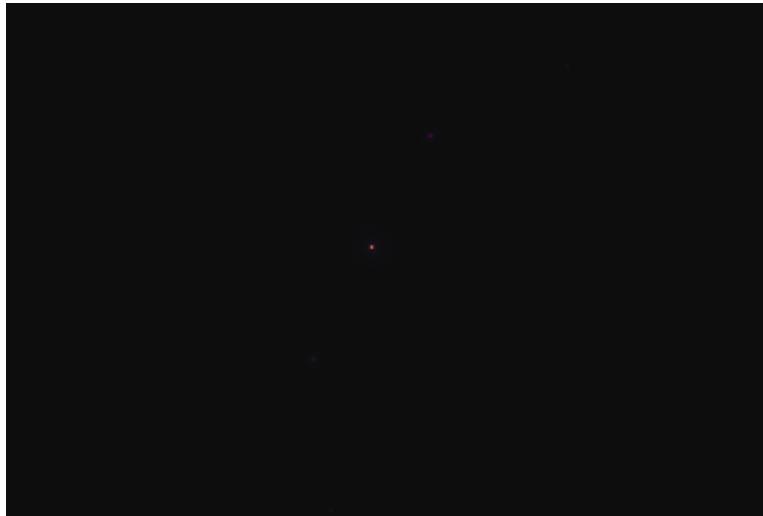
```
plotImageFullRange(image05_rings01,'image05');
```



```
plotImageFullRange(image06_gold,'image06');
```

Making Image Figures from Scalar Data

Convert to floating point, scale intensity from minimum to maximum value:



```
plotImageFullRange(image07_diff01,'image07');
```

```
plotImageFullRange(image08_CBED01,'image08');
```

```
plotImageFullRange(image09_Pt,'image09');
```

Making Image Figures from Scalar Data

Scale intensity automatically, using statistics:

Mean and std. dev. work for most images

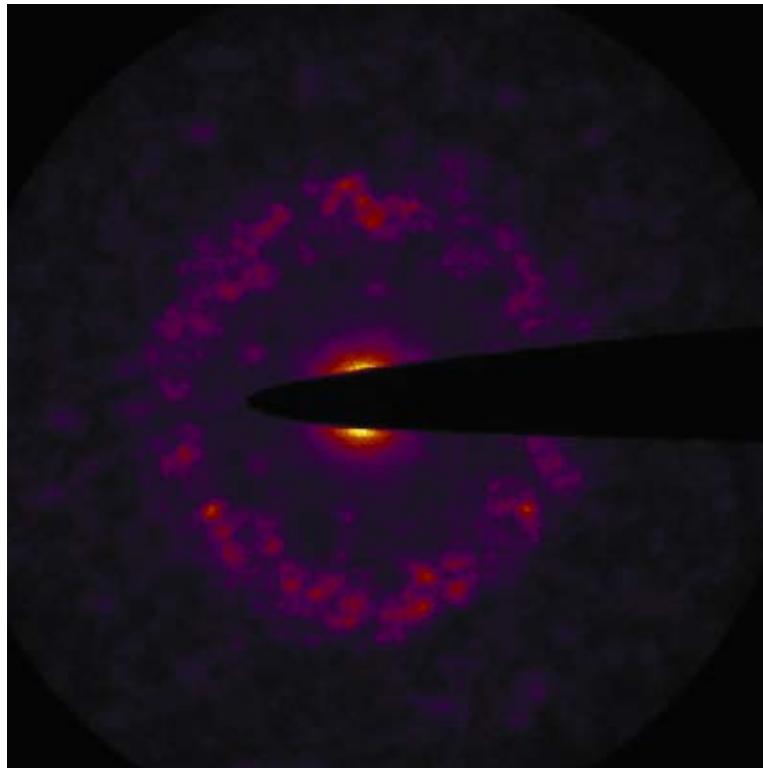
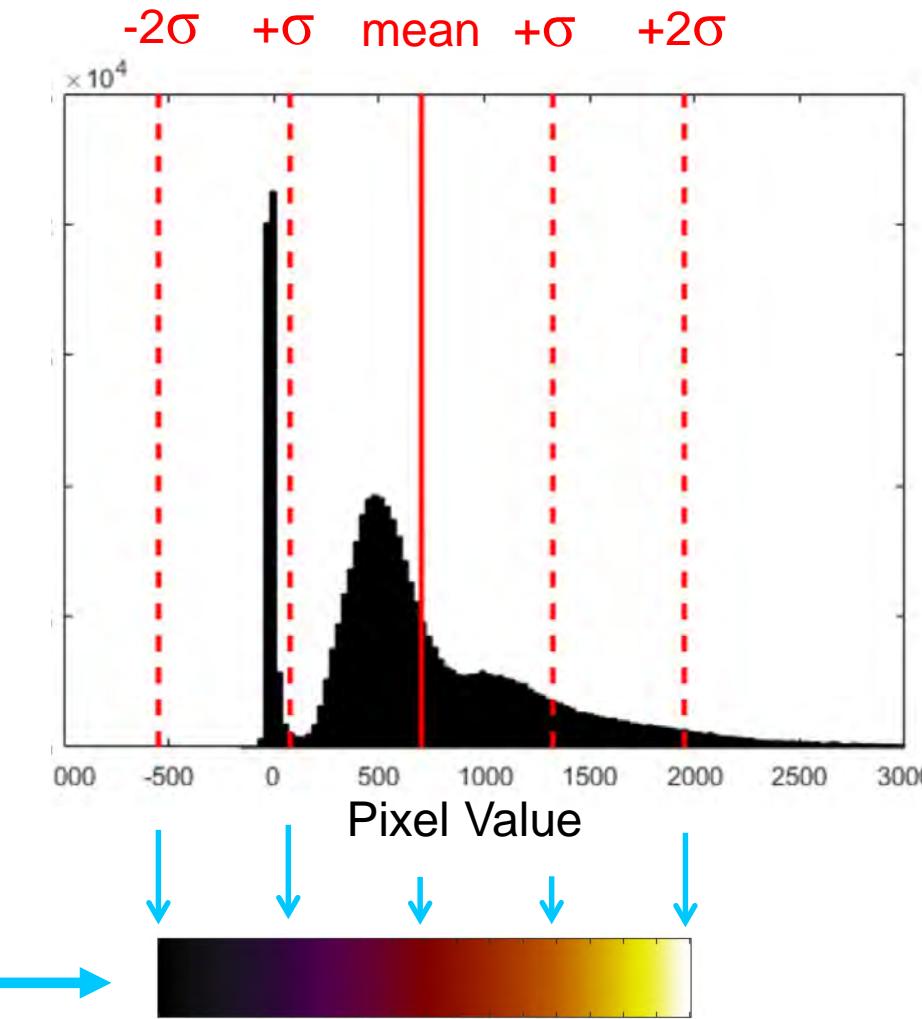
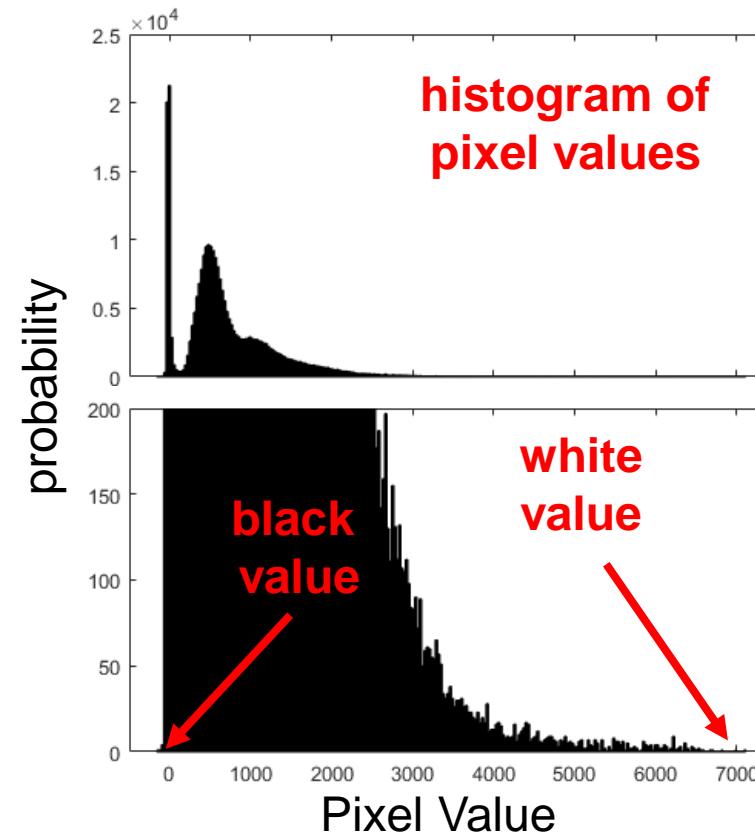


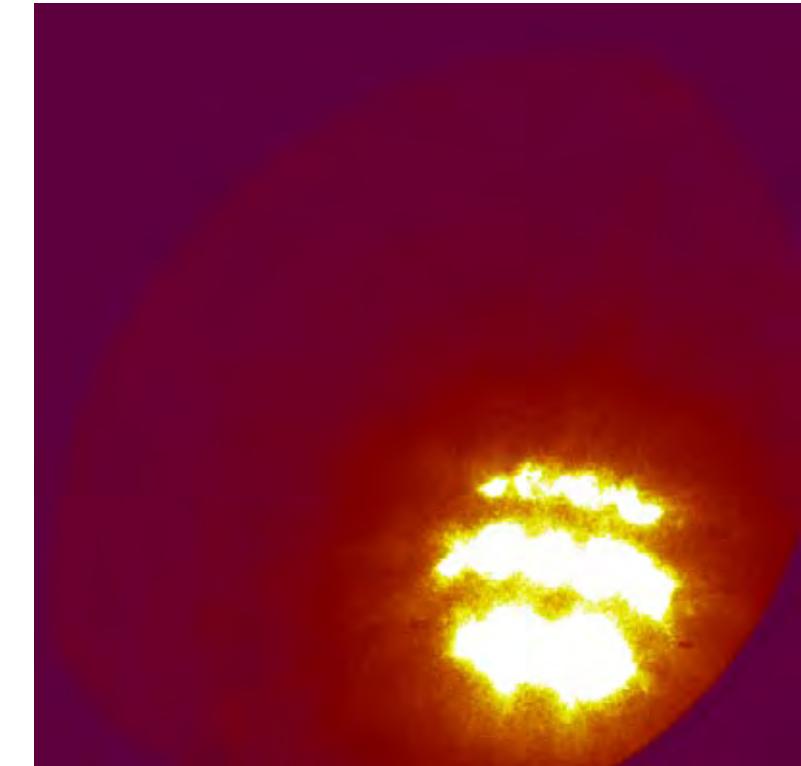
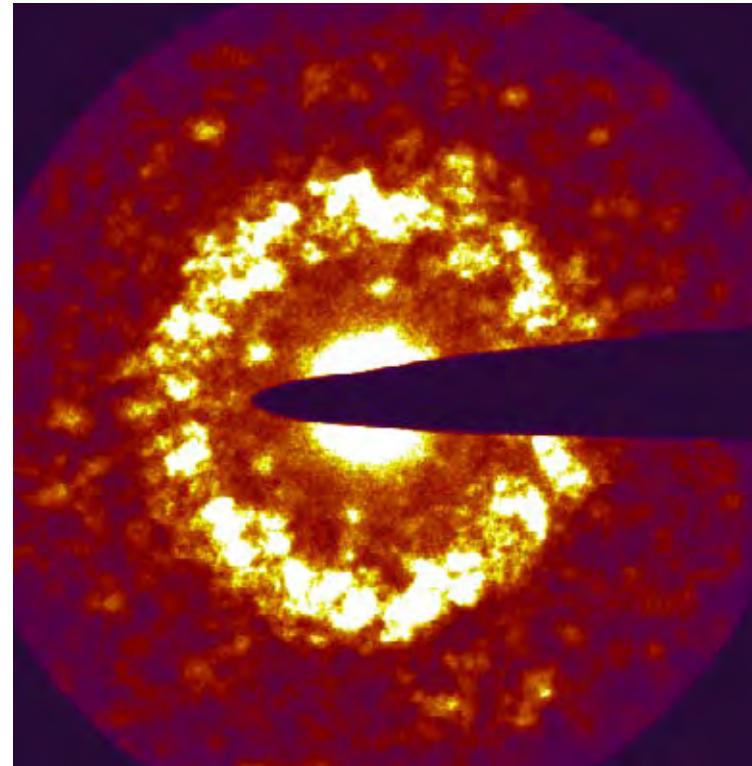
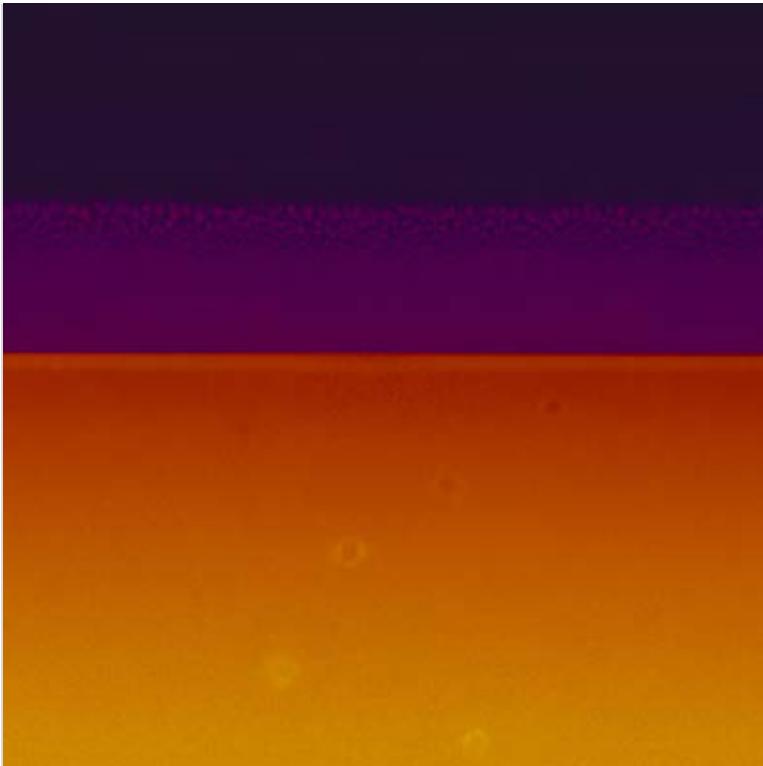
image02_amor_diff



Colormap – this is a mapping from pixel values to RGB triplet

Making Image Figures from Scalar Data

Scale intensity automatically from -2 to +2 standard deviations of intensity:



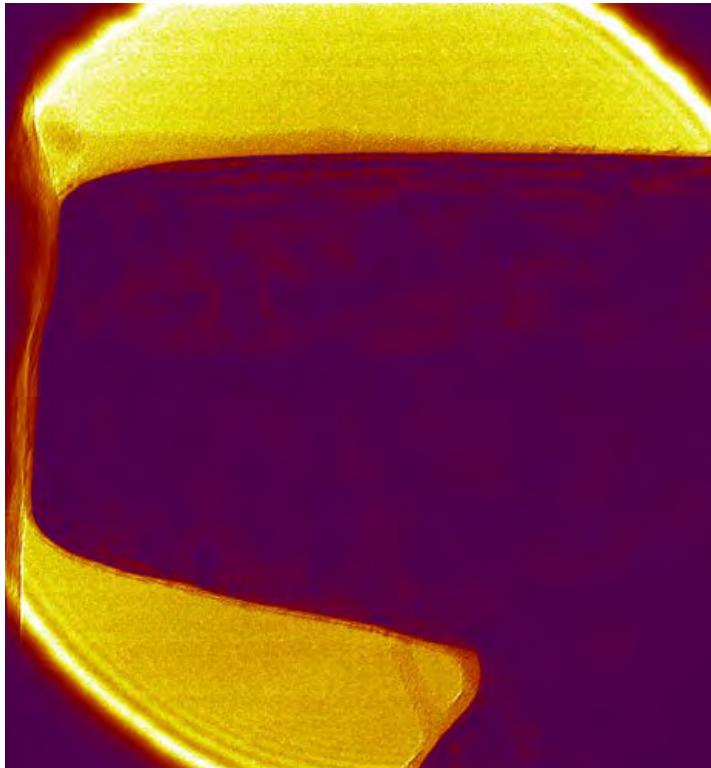
```
imwrite(image01_UO2_01,'image01.png');
```

```
imwrite(image02_amor_diff,'image02.png');
```

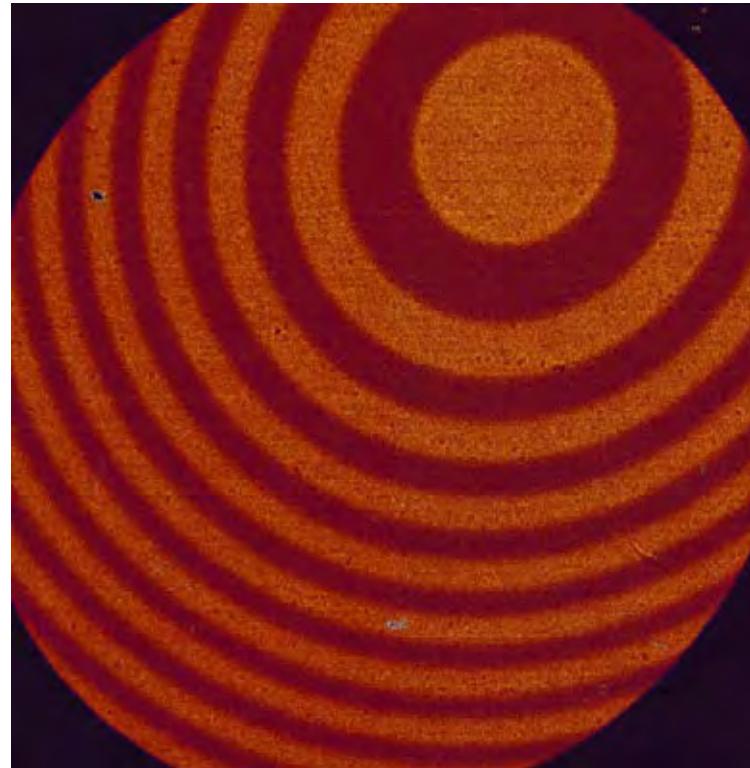
```
imwrite(image03_diff,'image03.png');
```

Making Image Figures from Scalar Data

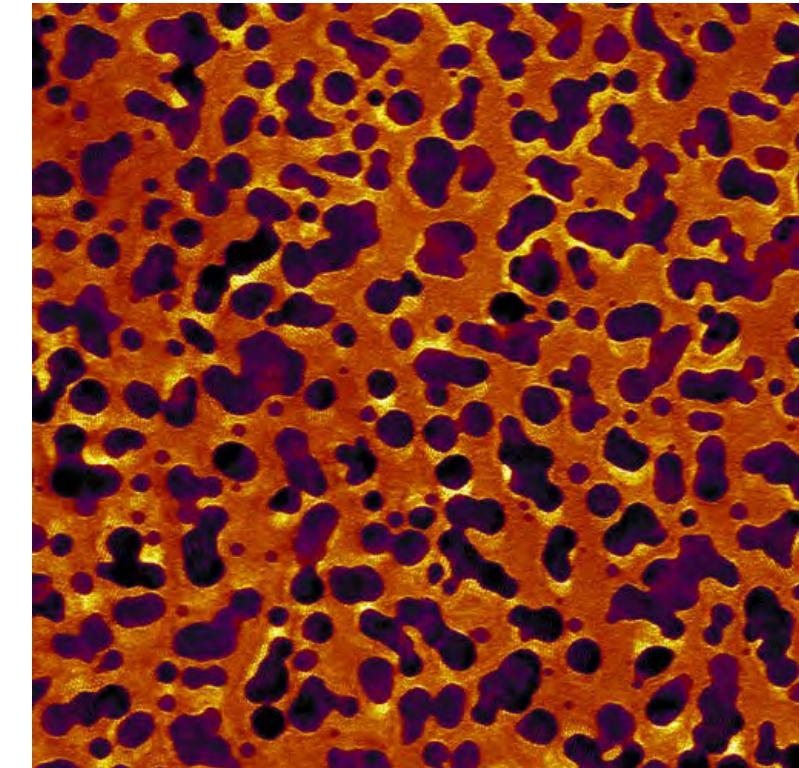
Scale intensity automatically from -2 to +2 standard deviations of intensity:



```
imwrite(image01_UO2_01,'image01.png');
```



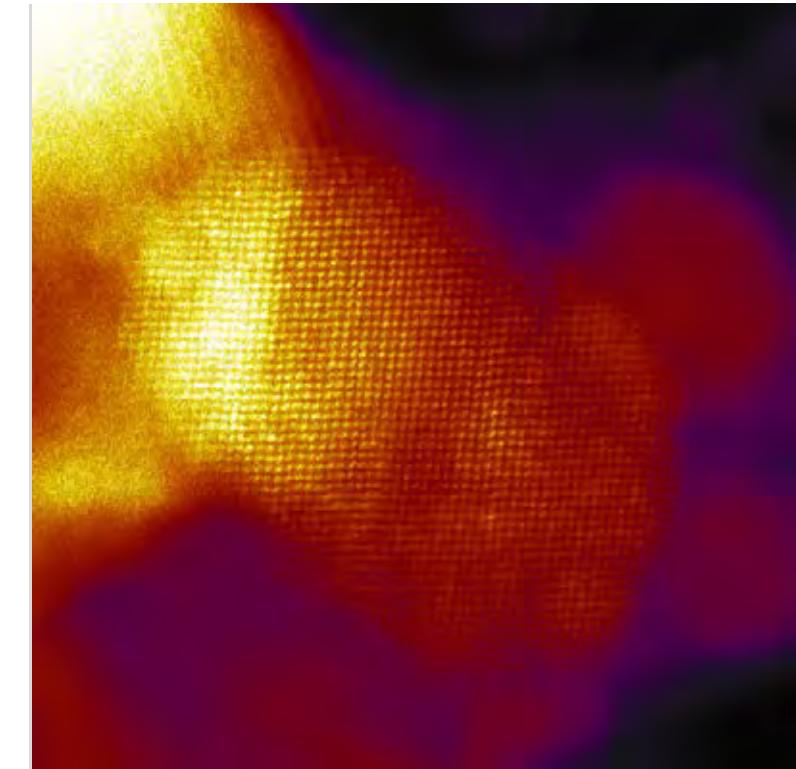
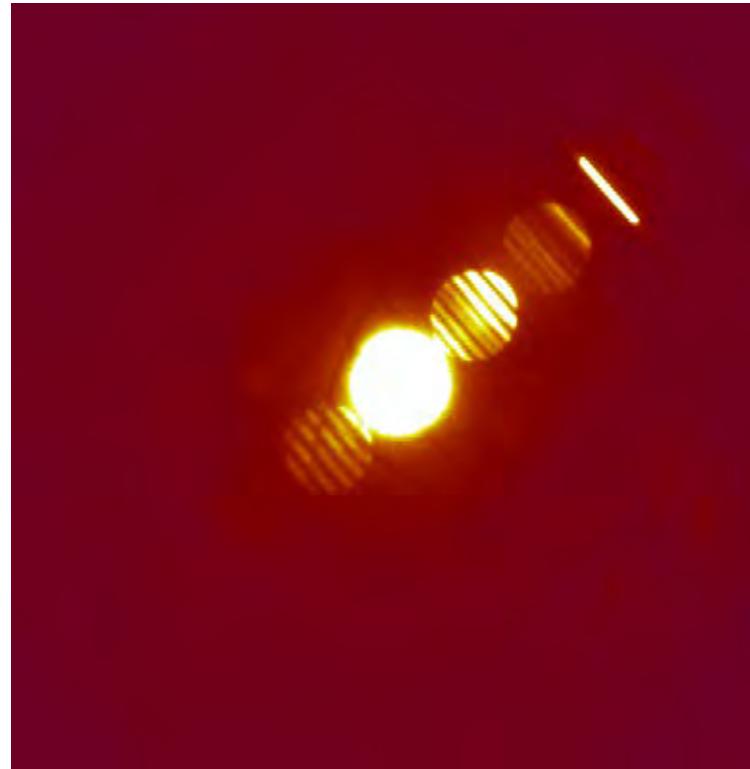
```
imwrite(image02_amor_diff,'image02.png');
```



```
imwrite(image03_diff,'image03.png');
```

Making Image Figures from Scalar Data

Scale intensity automatically from -2 to +2 standard deviations of intensity:



```
imwrite(image01_UO2_01,'image01.png');
```

```
imwrite(image02_amor_diff,'image02.png');
```

```
imwrite(image03_diff,'image03.png');
```

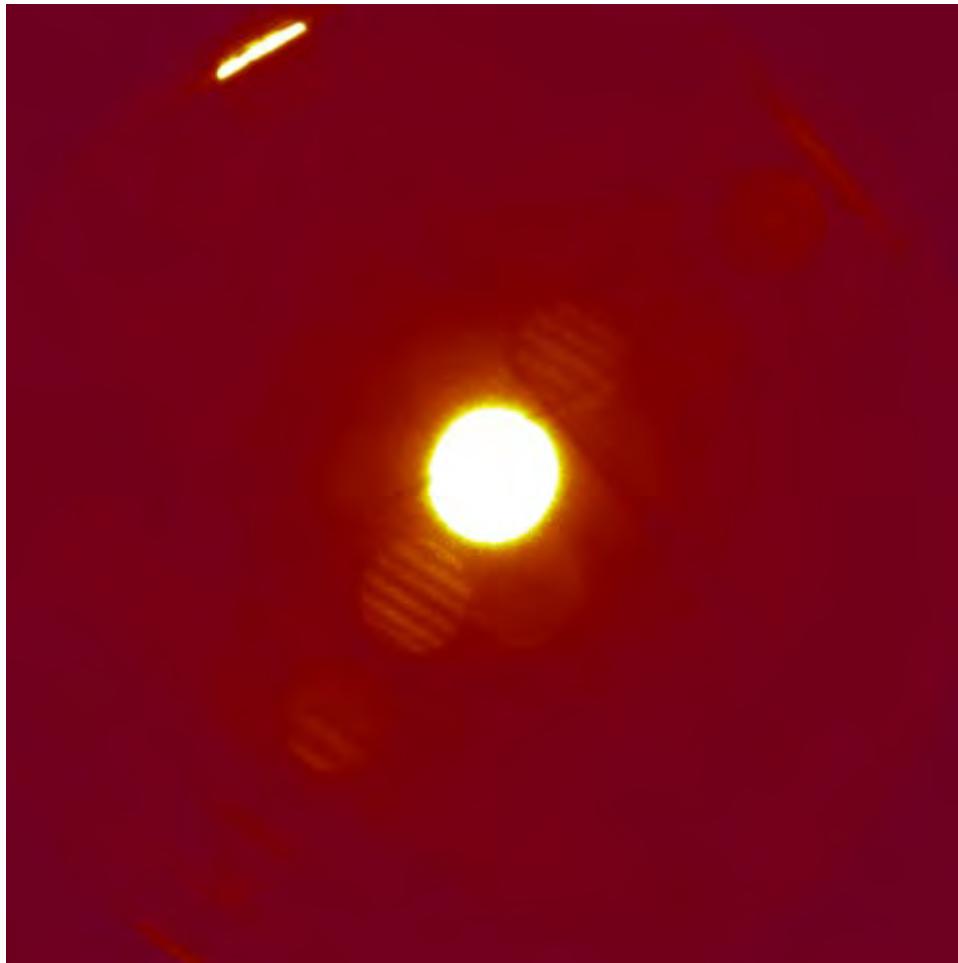
Making Image Figures from Scalar Data

The rest of the examples have been put into a single general purpose script:

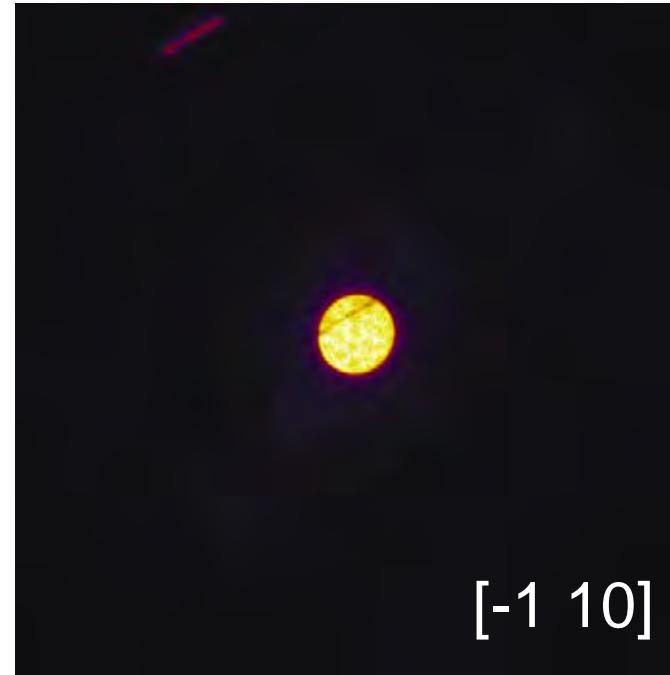
```
plotImage( ... )
```

Making Image Figures from Scalar Data

Many other automatic scaling schemes are possible, for example:

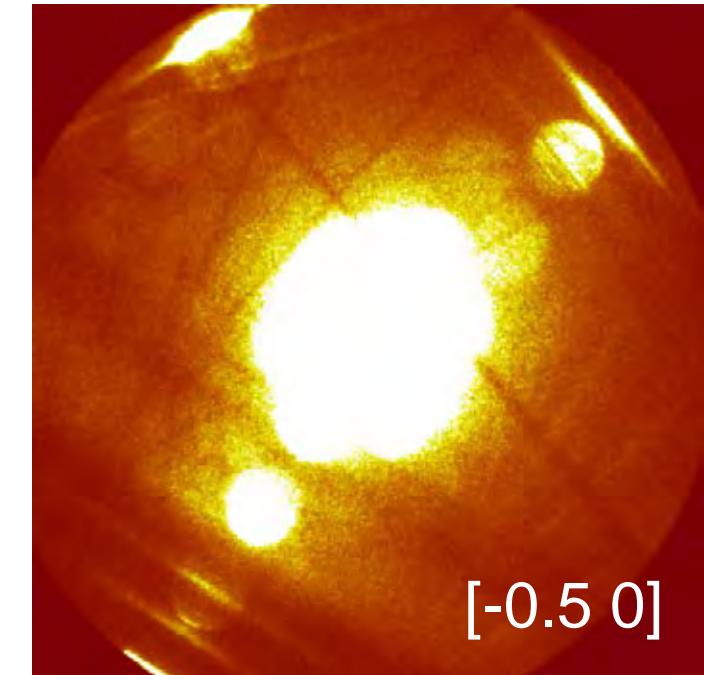


```
plotImage(image08_CBED03,'image08');
```



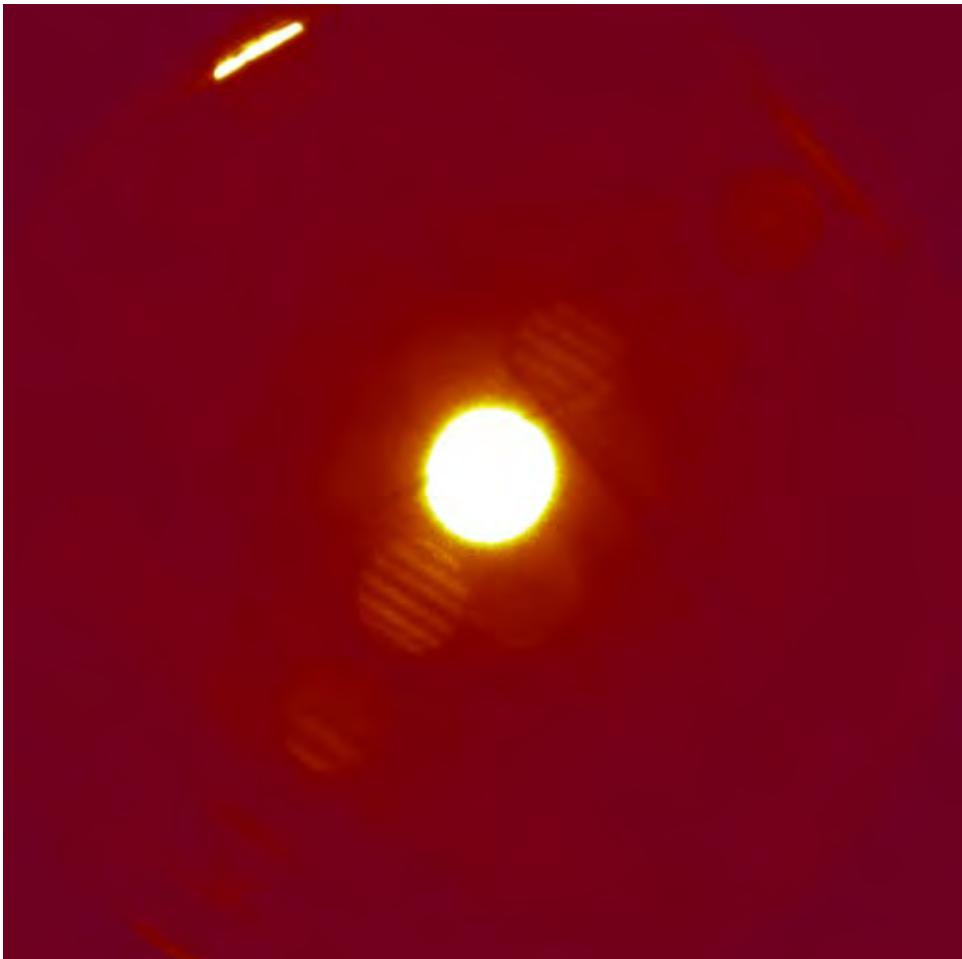
```
plotImage(image08_CBED03,'image08', [minVal maxVal]);
```

Is it impossible to linearly
visualize all image regions?

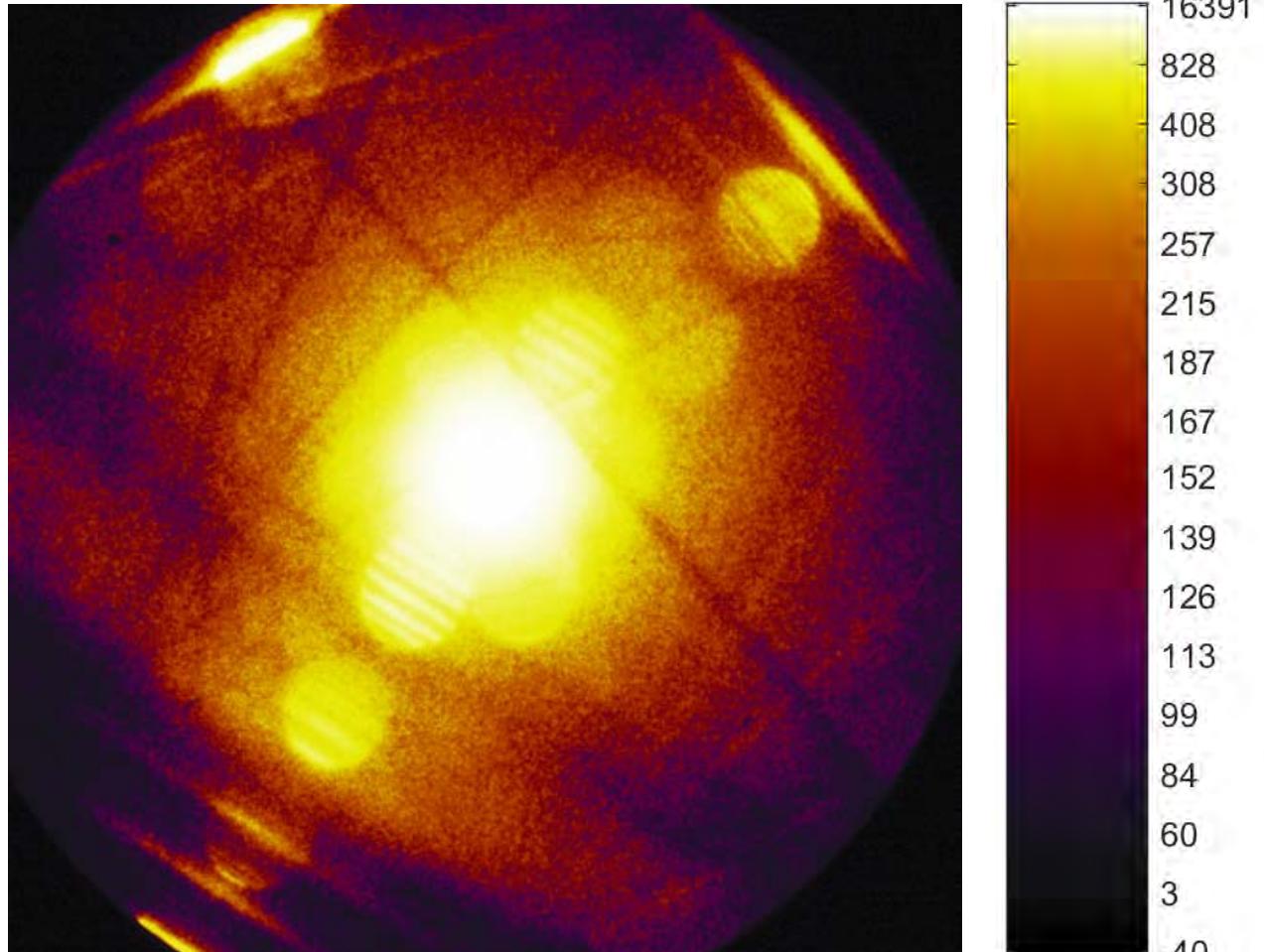


Making Image Figures from Scalar Data

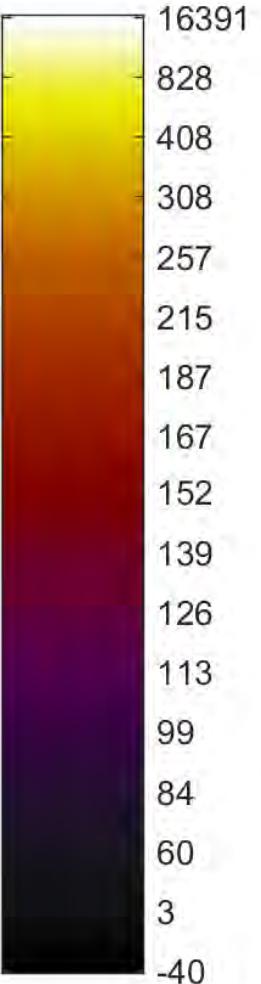
Many other automatic scaling schemes are possible, for example:



```
plotImage(image08_CBED03,'image08');
```

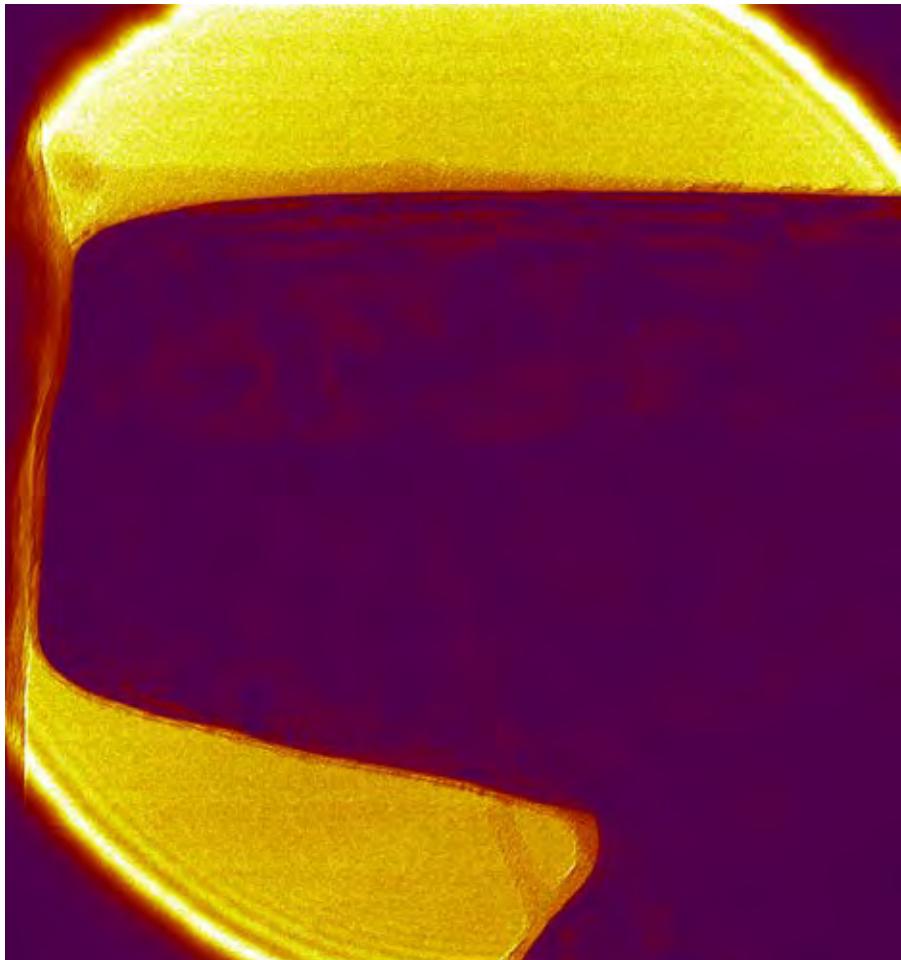


```
plotImage(image08_CBED03,'image08','ordered');
```

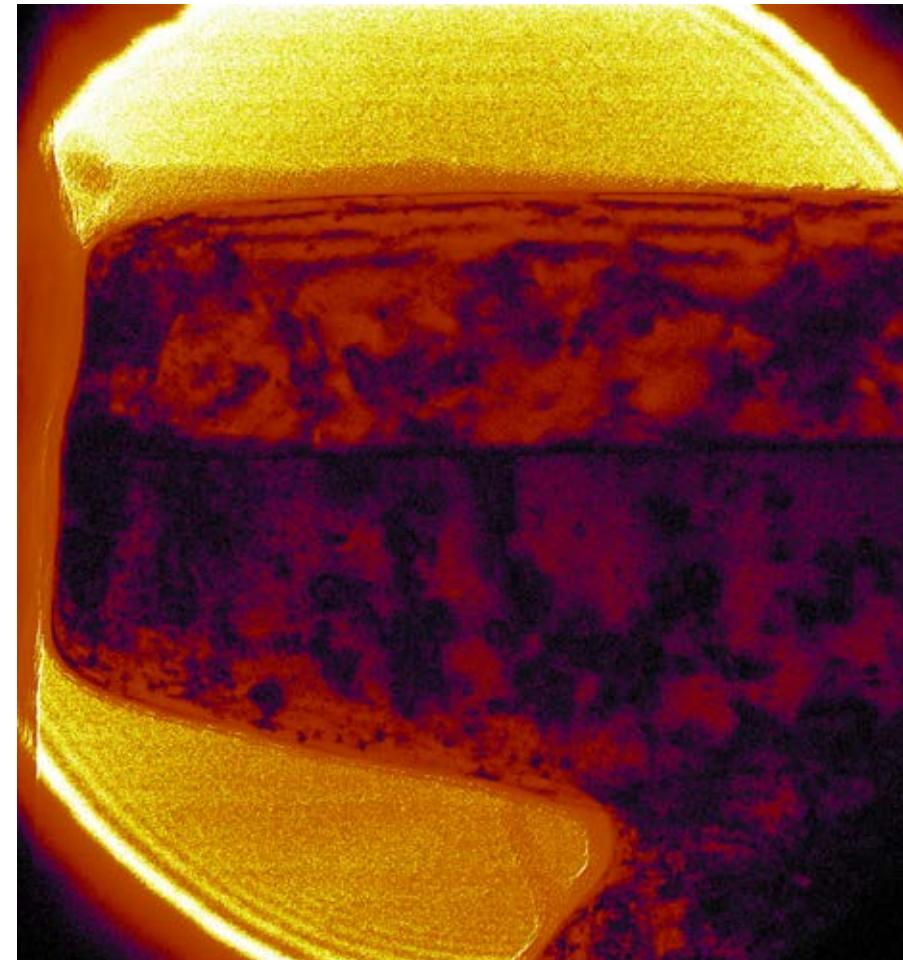


Making Image Figures from Scalar Data

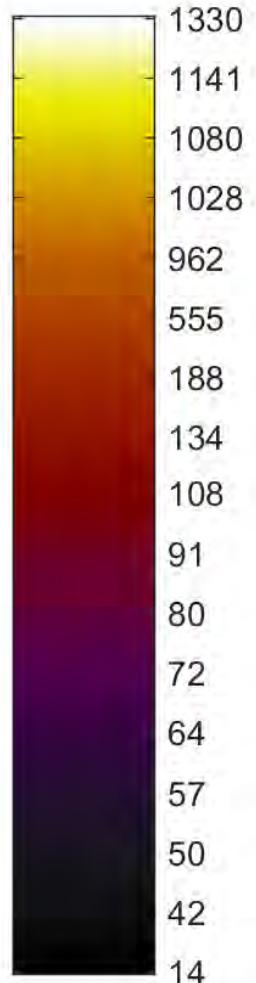
Many other automatic scaling schemes are possible, for example:



```
plotImage(image04_pillar, 'image04');
```



```
plotImage(image04_pillar, 'image04','ordered');
```

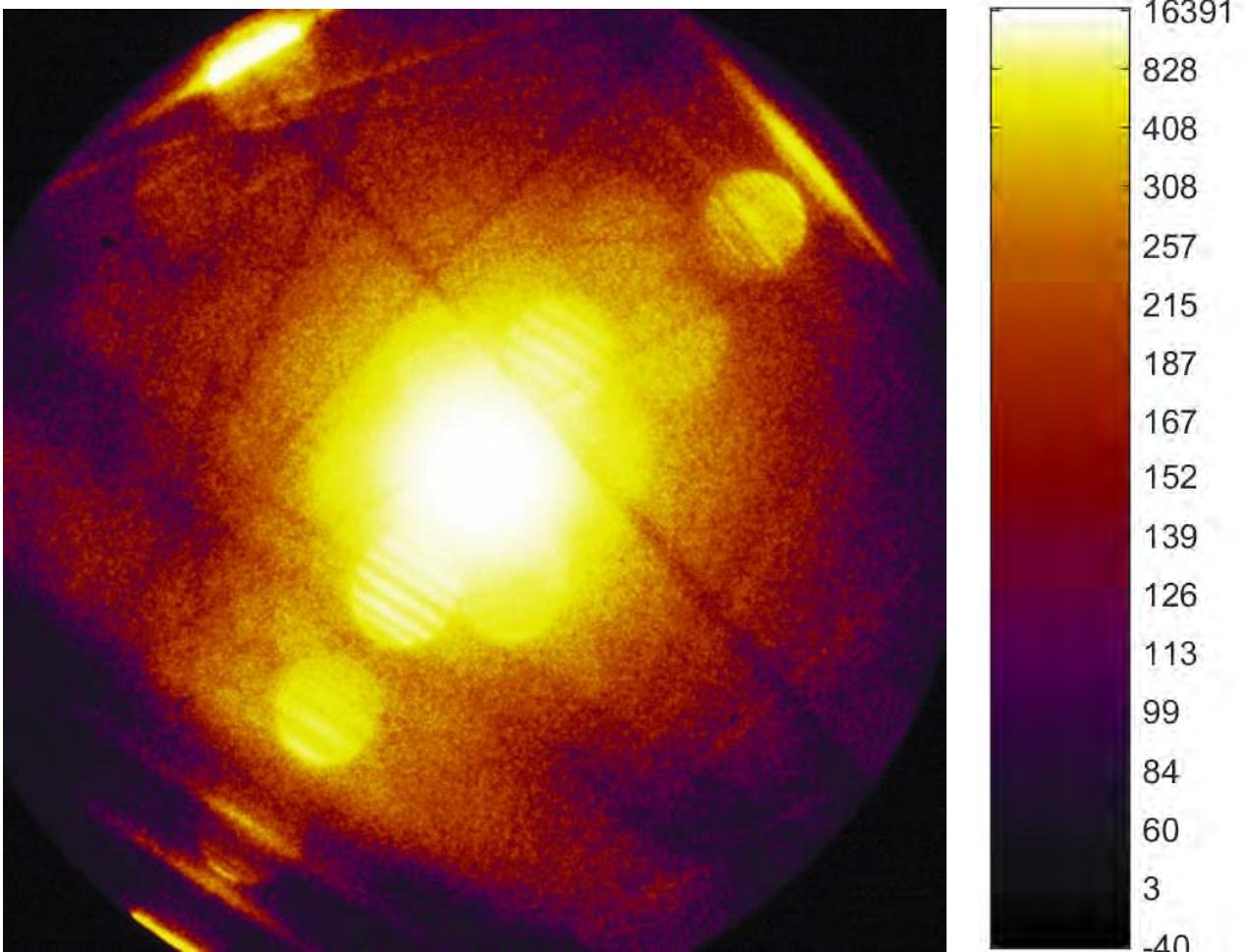


Making Image Figures from Scalar Data

Many other automatic scaling schemes are possible, for example:

Ordered Method

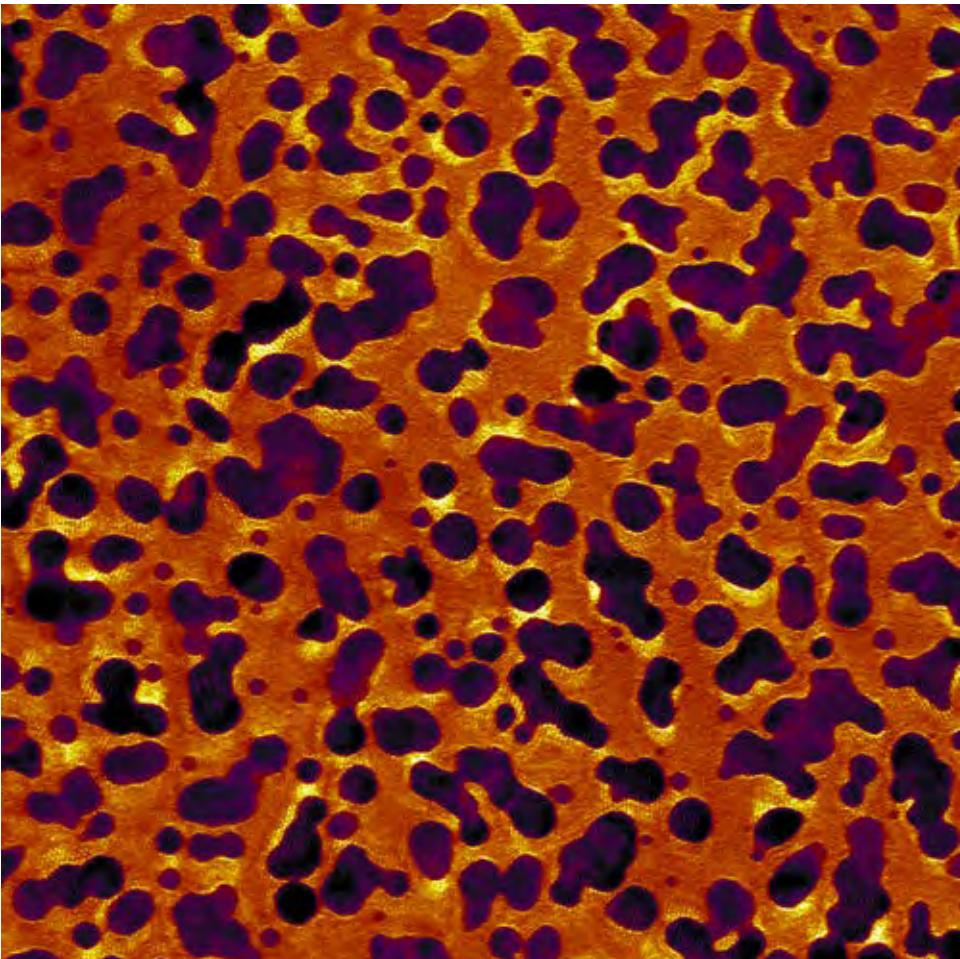
- All pixels of image are sorted from lowest to highest values.
- For a colormap with 100 colors specified, we assign falling in the range 0.00-0.01 to the first color, from 0.01-0.02 to the second color and so on.
- Note that this means the color legend will in general NOT be linear, logarithmic, or any simple relationship.



```
plotImage(image08_CBED03,'image08','ordered');
```

Making Image Figures from Scalar Data

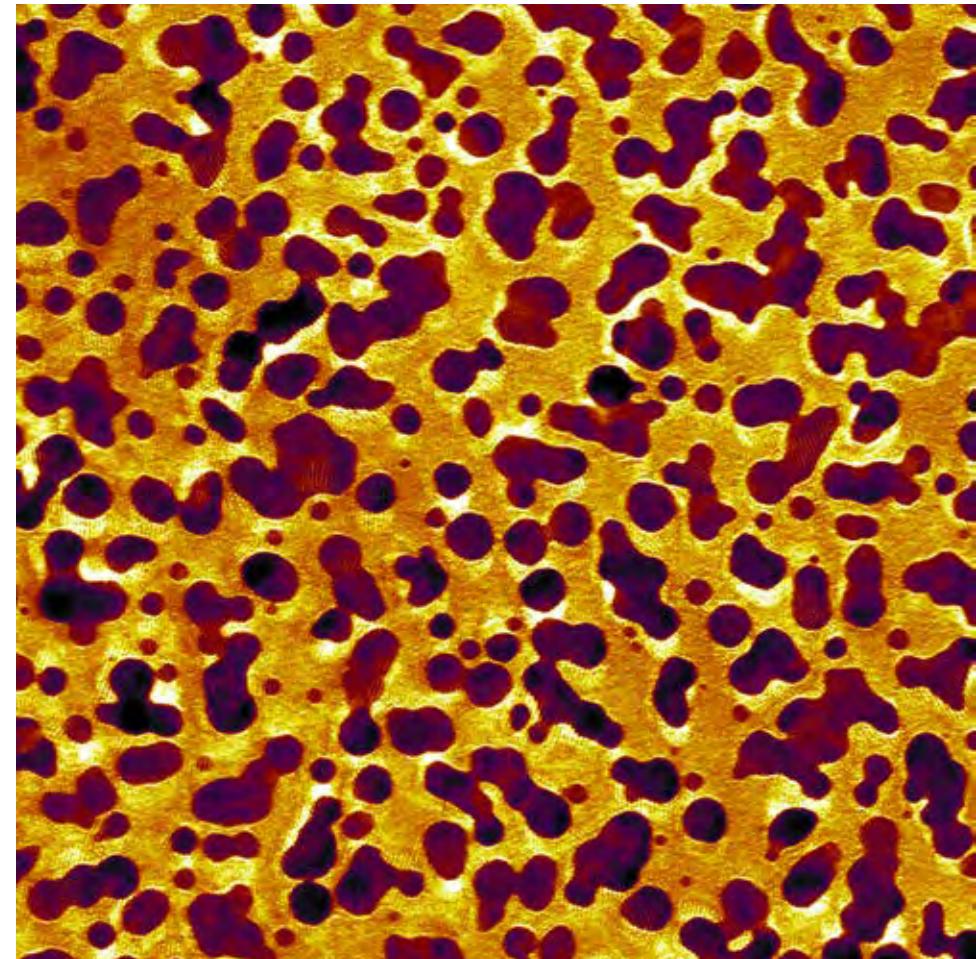
Many other automatic scaling schemes are possible, for example:



```
plotImage(image06_gold, 'image06');
```

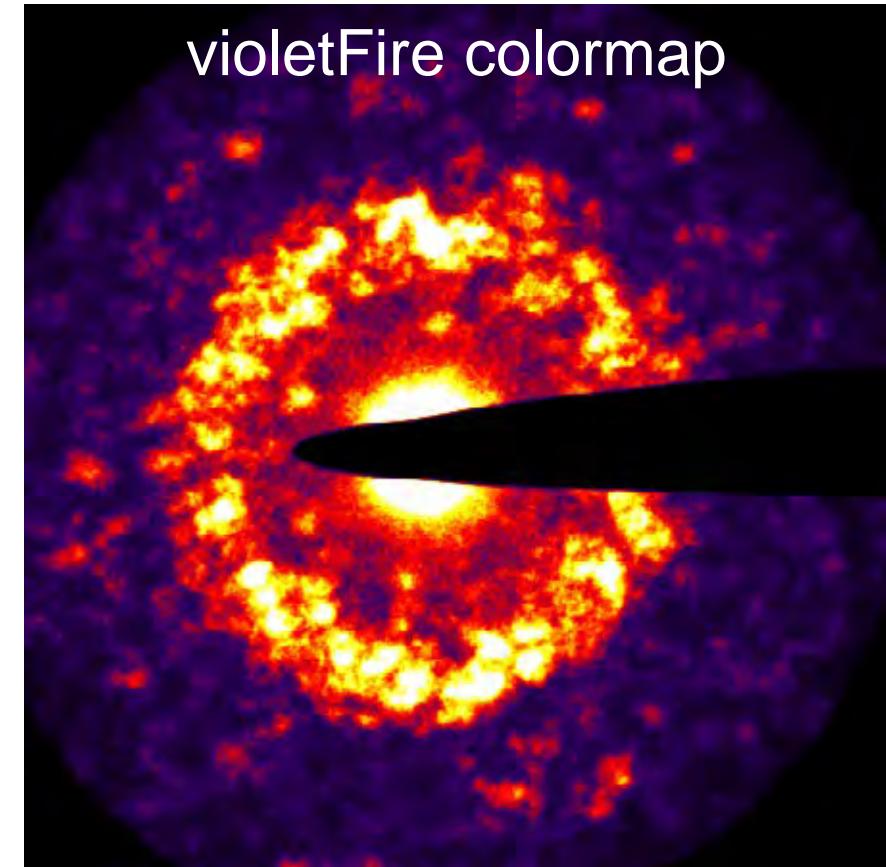
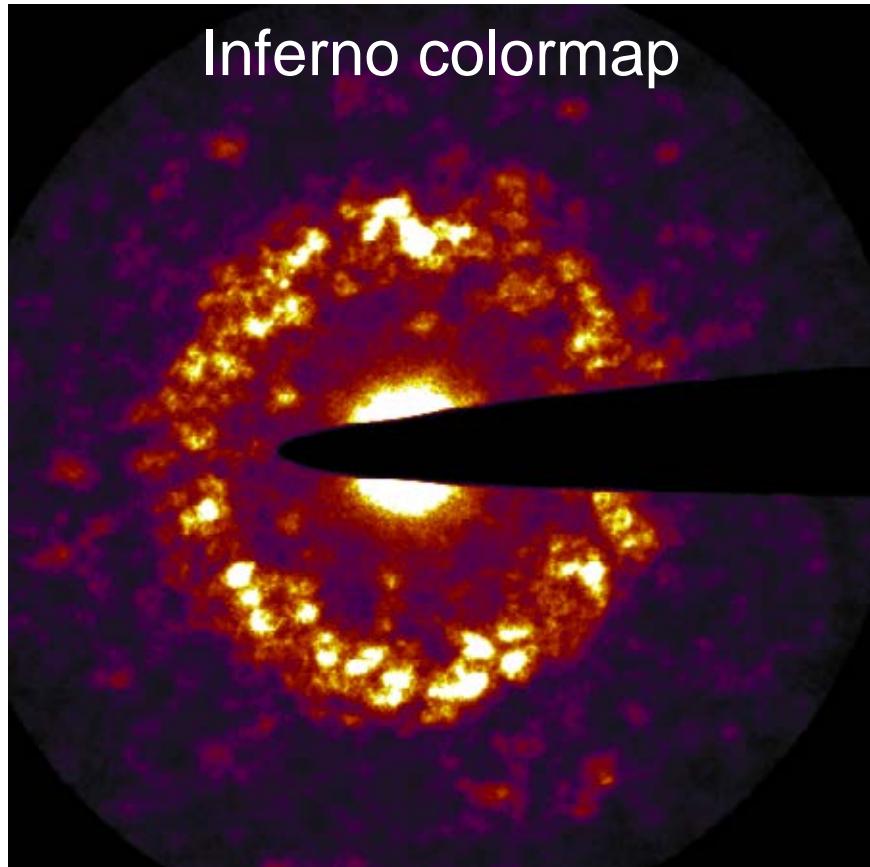


Asymmetric uses a different std. dev. in the positive and negative direction.



```
plotImage(image06_gold, 'image06', 'asymmetric');
```

A Note on Colormaps

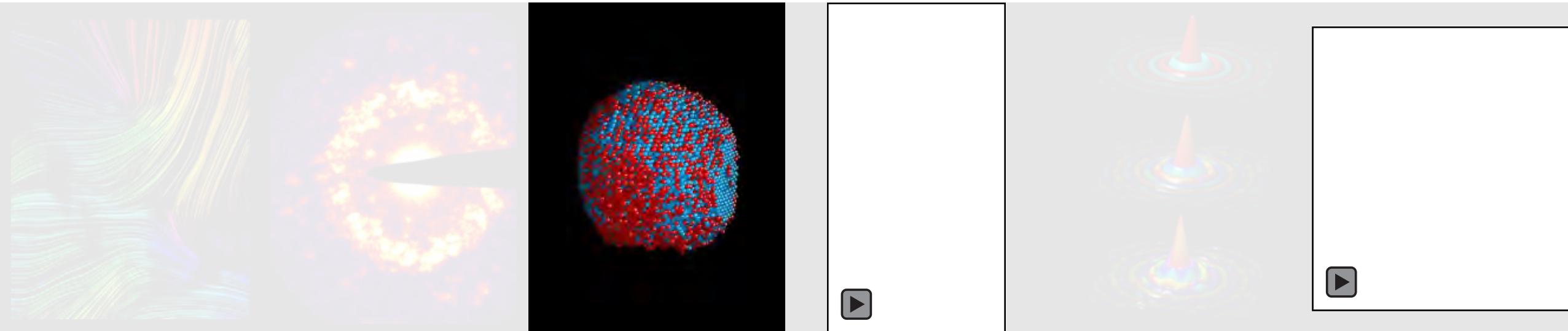


```
plotImage(image02_amor_diff,[-1 3],'image02_inferno');
```

```
plotImage(image02_amor_diff,violetFire,[-1 3],'image02_violetFire');
```

“Eye-balled” the inferno colormap - my own similar map “violetFire” from several years back is brighter, but doesn’t have the same perceptual uniformity.

Outline



1

Basic concepts
when making
figures and
movies – raster
vs vector.

2

Scaling,
coloring and
saving scalar
and vector
imaging data.

3

Plotting **points**
in 3D, highlight
internal features
– e.g. atomic
coordinates.

4

Animating 3D
points, making
movies
– e.g. atomic
coordinates.

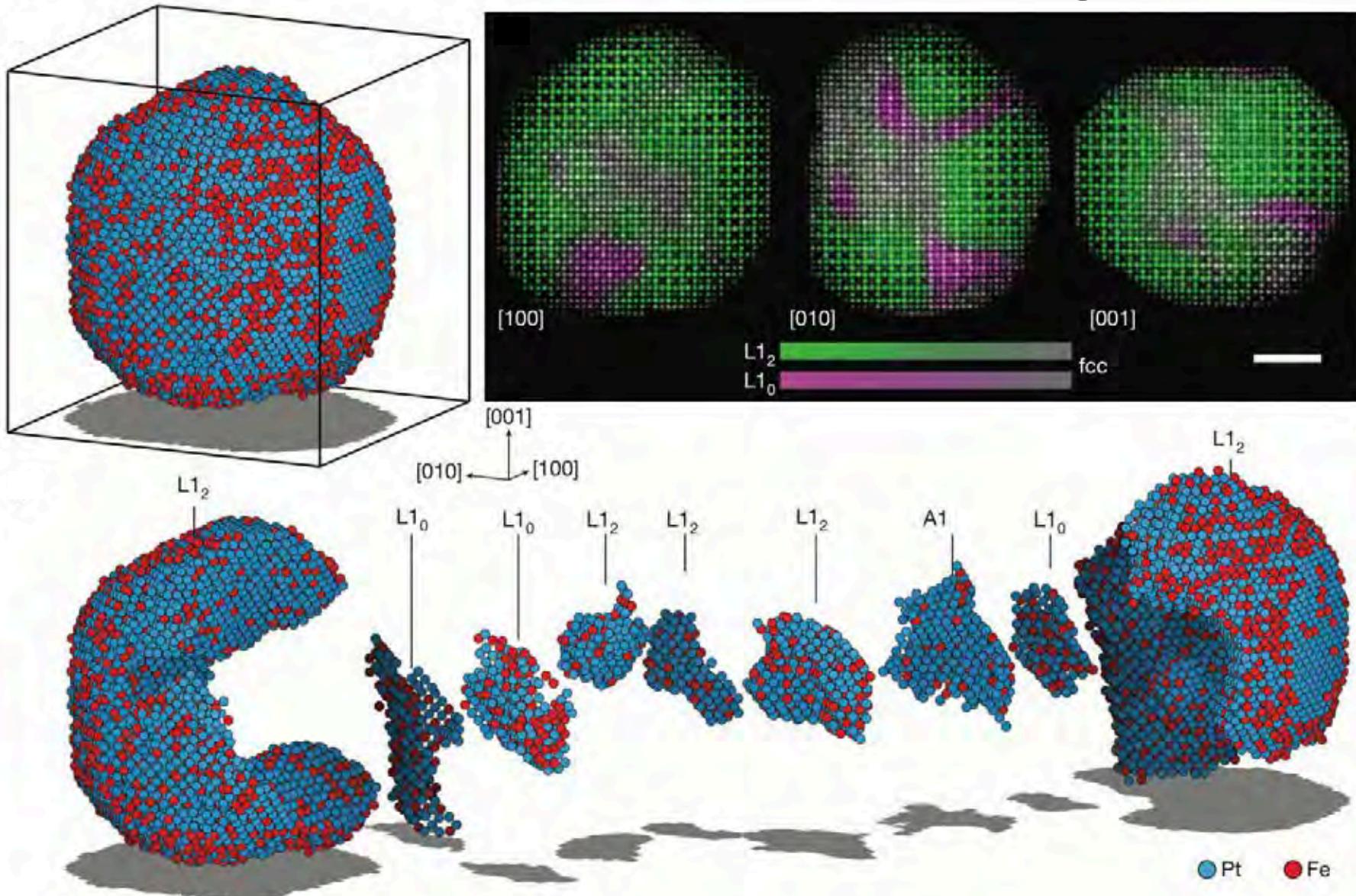
5

Building physics
visualization
and simulation
directly into
figures.

6

Making simple
animations to
explain
concepts in
physics.

Common Vis Task – Plotting 3D Atoms

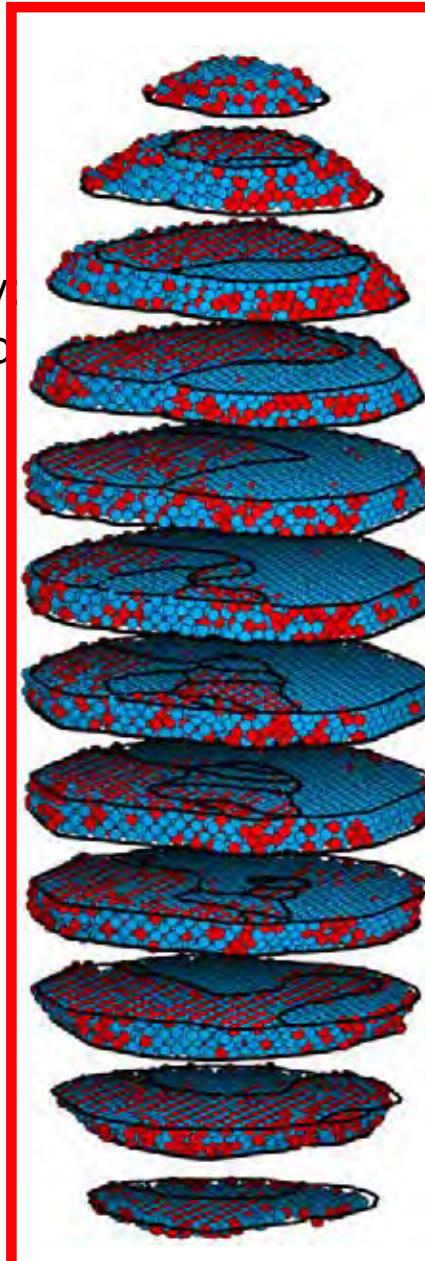


atomic
electron
tomography
position and
species

● Fe
● Pt

GBs:
L₁₂
and
L₁₀
grains

Tutorial on
how I drew
this panel:



Common Visualization Task – 3D Atoms

For generating plots from code, you should “plot as you code.”

By monitoring your vis progress, you can catch mistakes early!

As your visualization codes become more advanced, you will perform less checks.

Load the atomic coordinate data:

```
>> load( 'atomData02.mat' )
```

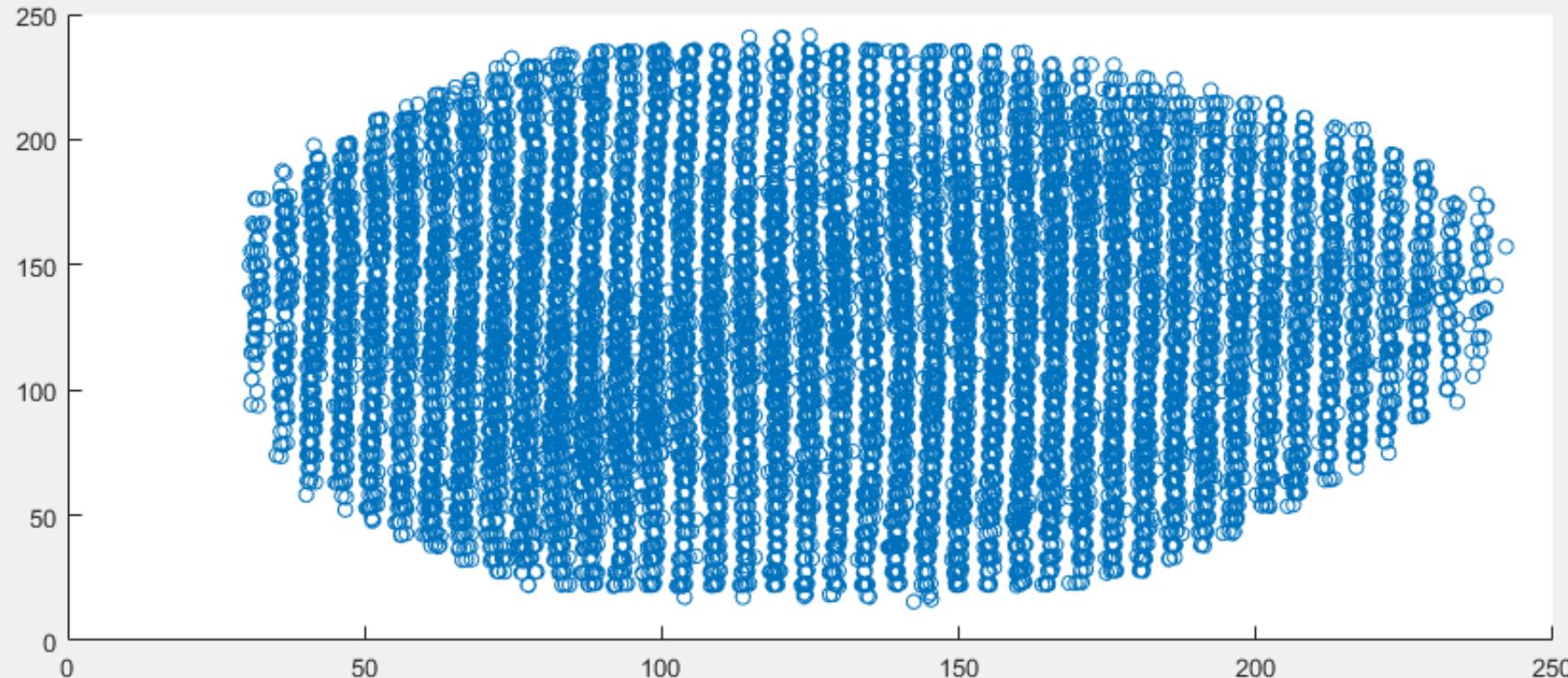
Running the plotting scripts:

```
>> plotAtoms00( atomData, atomLattice );  
>> plotAtoms01( atomData, atomLattice );
```

Our goal is to understand all of the plotting elements in plotAtoms01(...)

Various plotting commands are commented out, we will take a visual tour through these commands. To try it yourself, just uncomment / comment the appropriate lines of code!

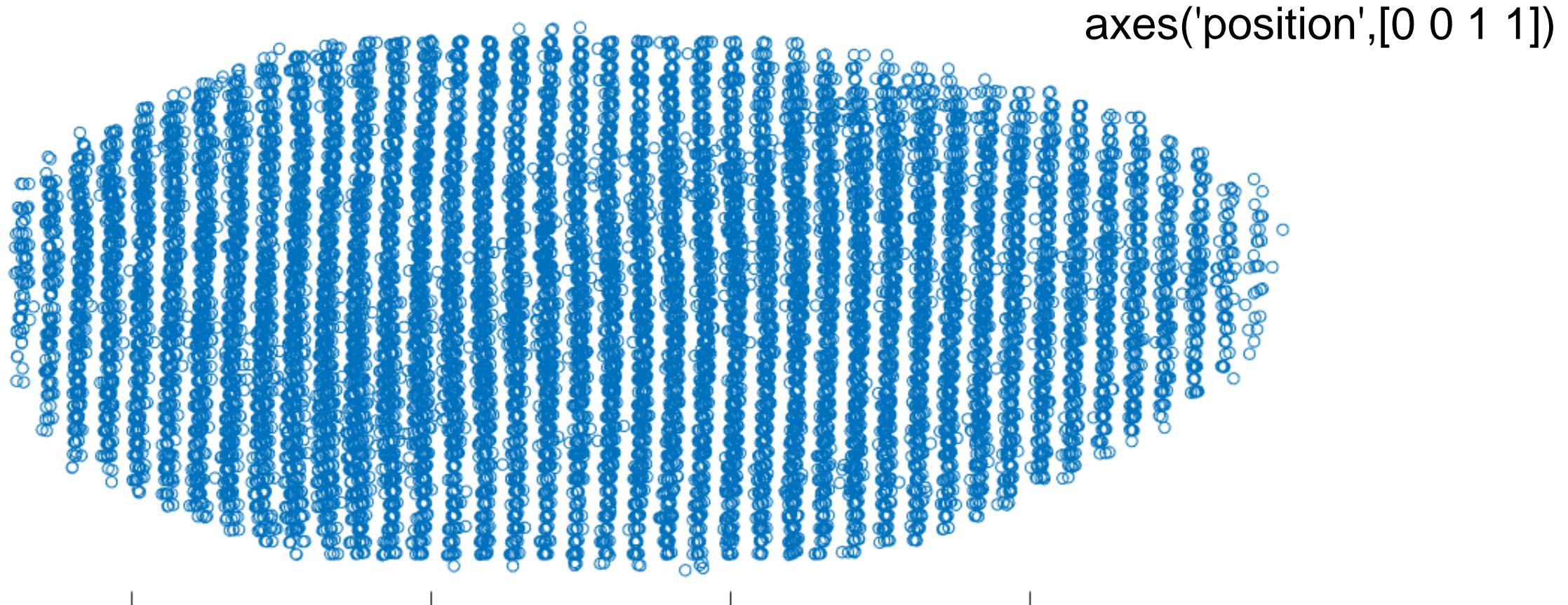
Common Visualization Task – 3D Atoms



```
scatter3( ...  
atomPos(:,1),...  
atomPos(:,2),...  
atomPos(:,3));
```

A 3D scatter plot of the coordinates, default appearance – pretty ugly!

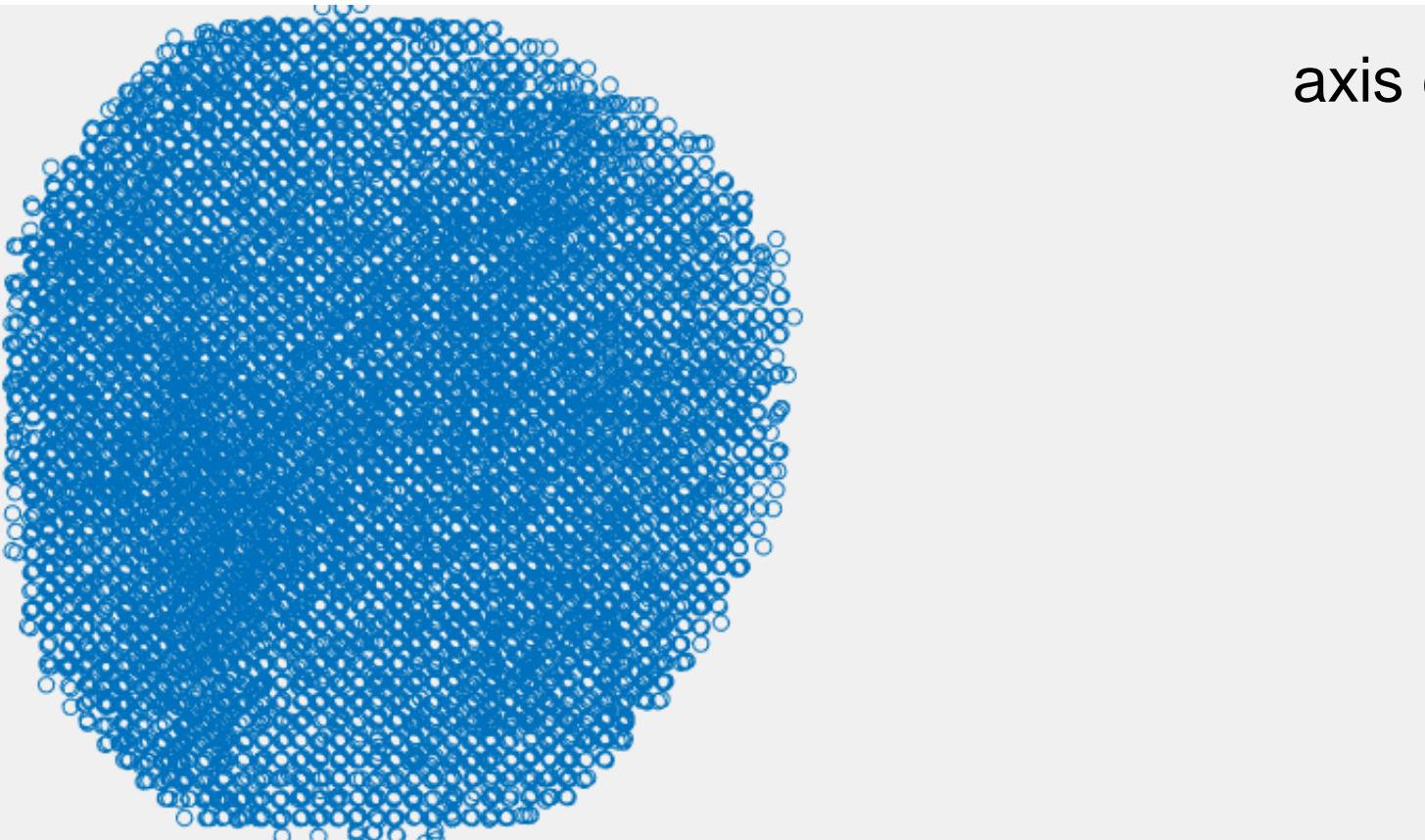
Common Visualization Task – 3D Atoms



This “axes” command will stretch the axes to fill the field of view.

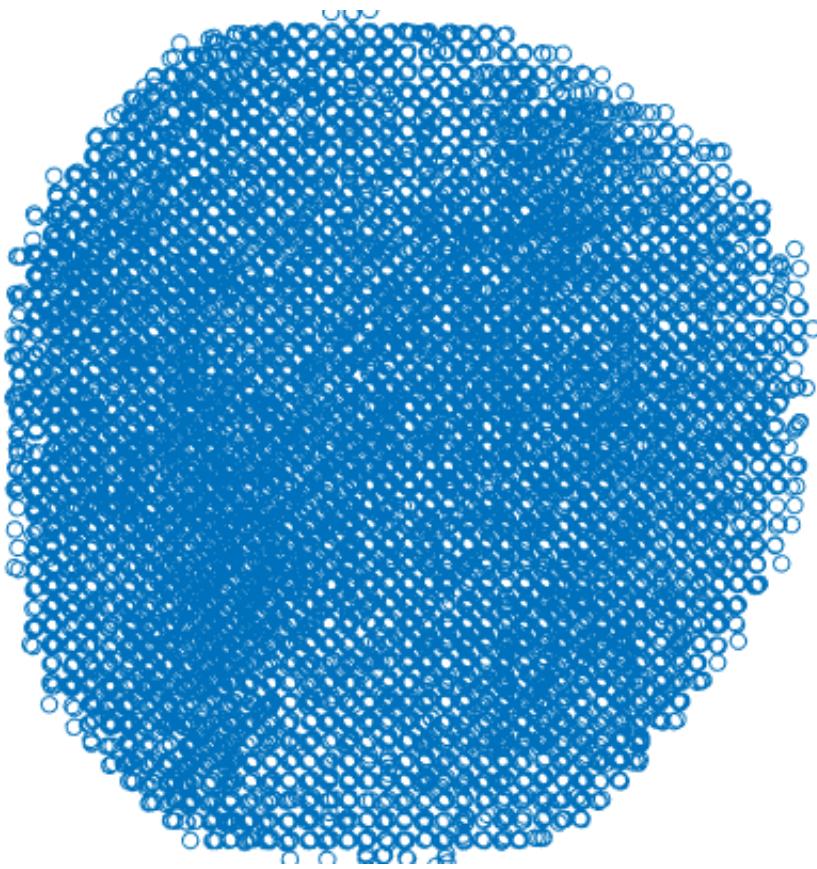
[x_origin y_origin x_width y_width] – all values normalized from 0 to 1.

Common Visualization Task – 3D Atoms



Set the aspect ratio of the 3 directions to be equivalent, hide the axes objects.

Common Visualization Task – 3D Atoms



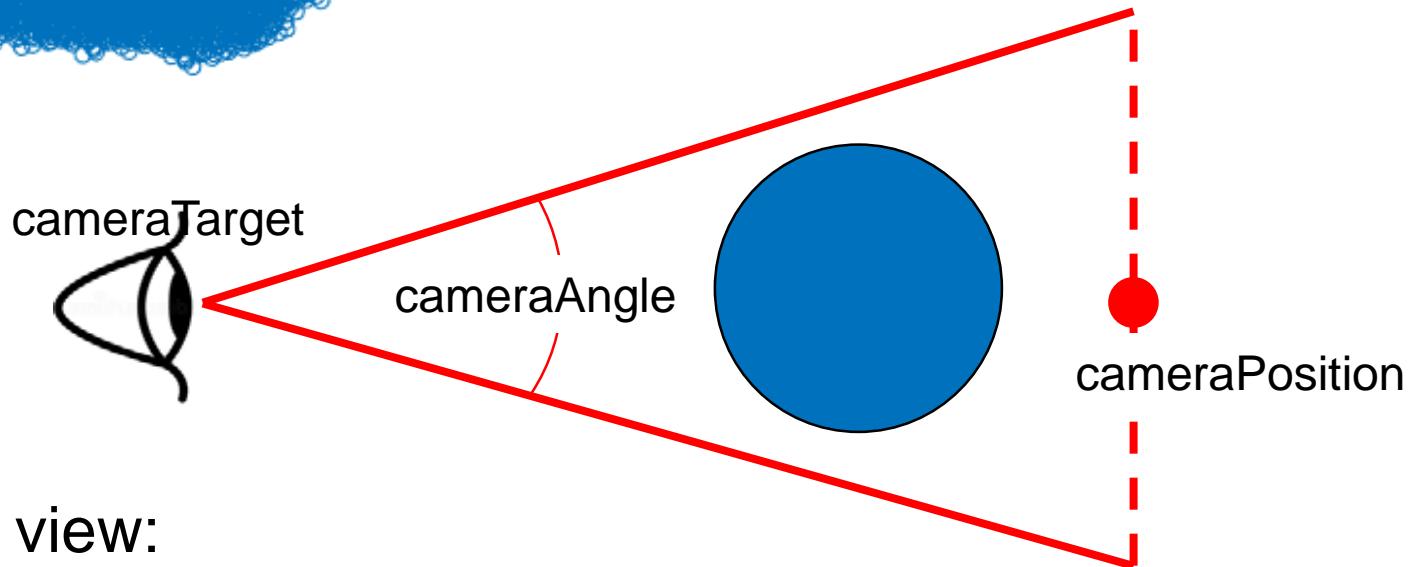
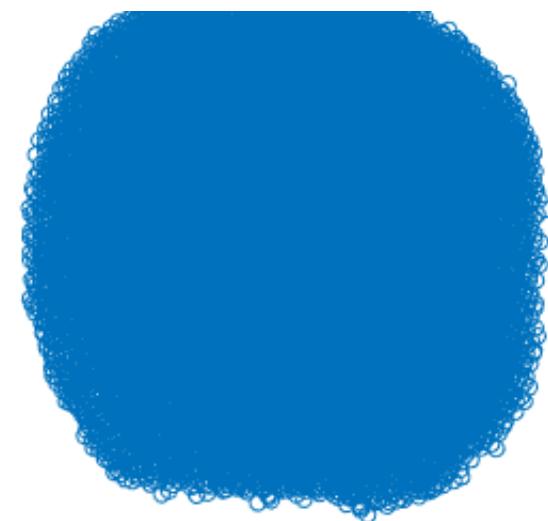
`set(gcf,'color','w');`

or equivalently

`set(gcf,'color',[1 1 1]);`

“gcf” = get current figure handle, set its color to white (red = 1, green = 1, blue = 1)

Common Visualization Task – 3D Atoms



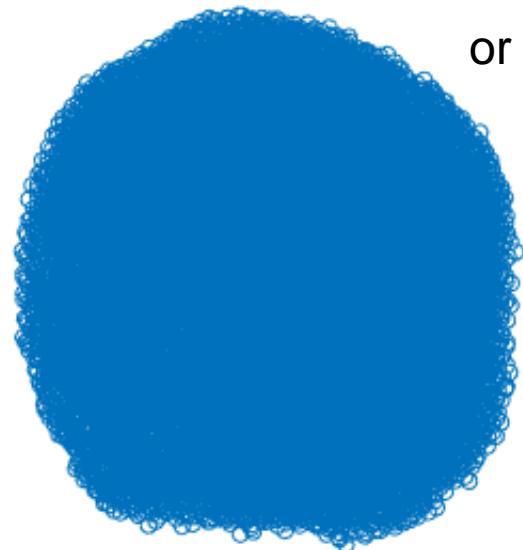
```
camtarget(cameraTarget)  
campos(cameraTarget ...  
+ cameraPosition);  
camva(cameraAngle)  
camproj('perspective')
```

Set up a rendering camera for a 3D view:

Common Visualization Task – 3D Atoms

```
for a0 = 1:3  
    atomPos(:,a0) = atomPos(:,a0) - mean(atomPos(:,a0));  
end
```

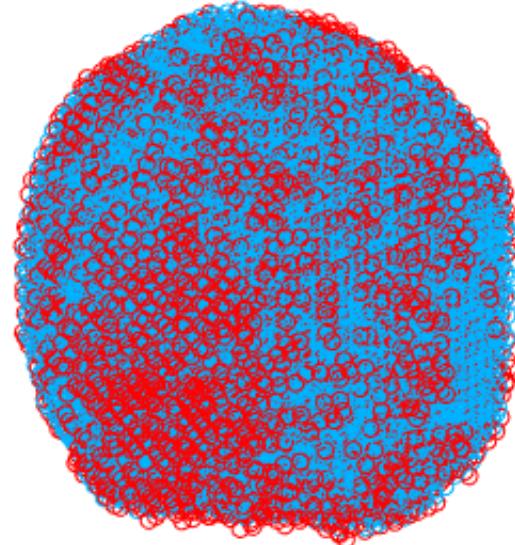
or `atomPos = atomPos - mean(atomPos, 1);`



Center the group of atoms on $(x,y,z) = (0,0,0)$ by subtracting the mean position.

Common Visualization Task – 3D Atoms

- Pt
- Fe

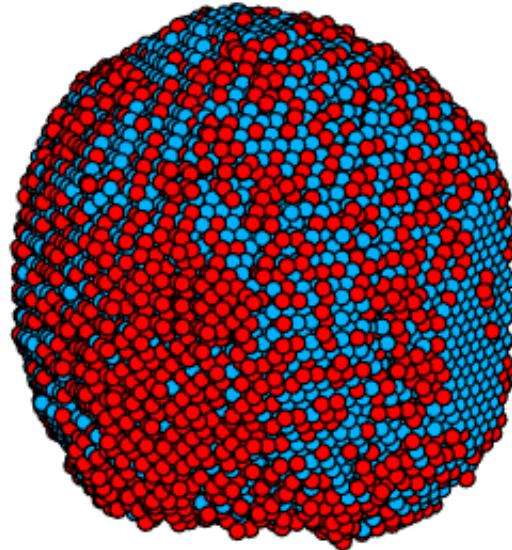


```
scatter3( ...
    atomPos(:,1),...
    atomPos(:,2),...
    atomPos(:,3),...
    ones(numberAtoms,1)*atomSize,...  
    atomRGB);
```

Give a unique color (and marker size) to each atom, in order to identify Fe and Pt sites.

Common Visualization Task – 3D Atoms

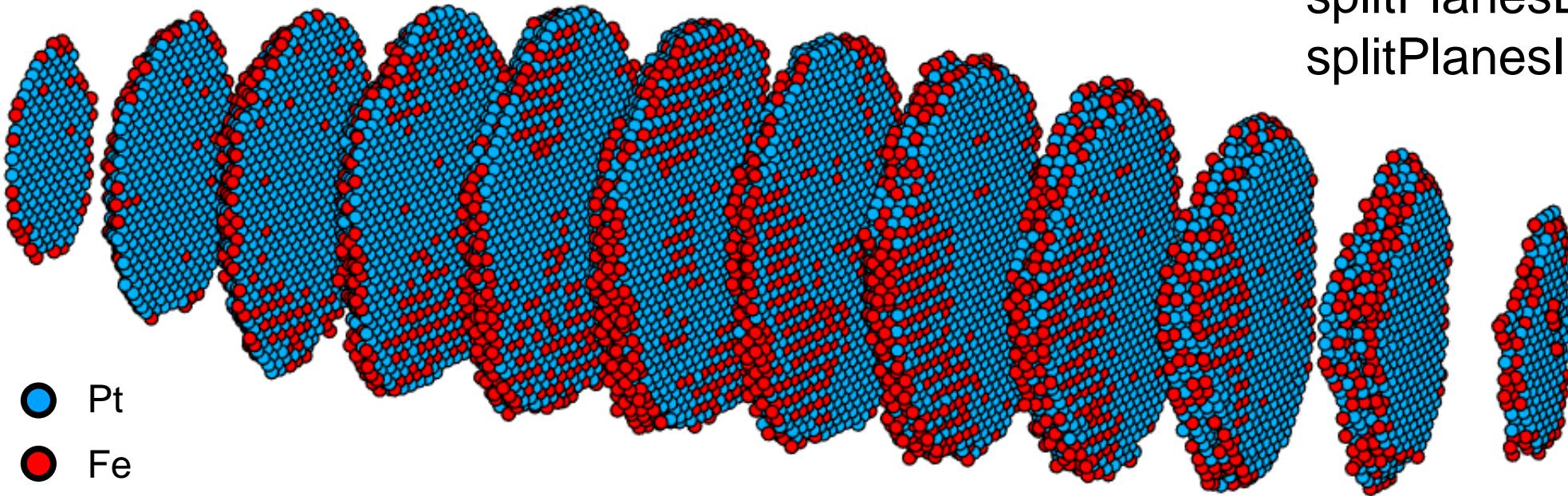
- Pt
- Fe



```
scatter3( ...
    atomPos(:,1),...
    atomPos(:,2),...
    atomPos(:,3),...
    ones(numberAtoms,1)*atomSize,...
    atomRGB,...
    'filled','marker','o',...
    'linewidth',atomLineWidth,...
    'markeredgecolor',atomColorEdges);
```

Update the marker appearance, filled circles, black edges, etc.

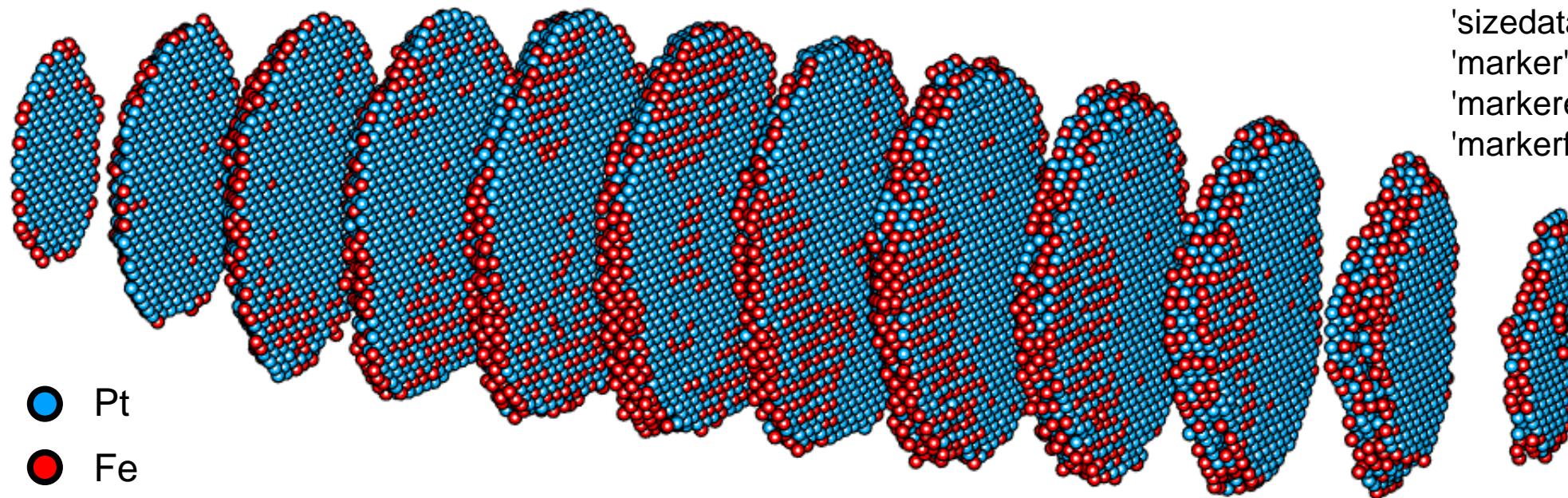
Common Visualization Task – 3D Atoms



Now we're moving beyond simple scatter plots – atoms are “bunched” up into sets of 4 atomic planes, and then each plane is moved 70 plotting units apart from the neighboring planes. This operation performed along dimension 2 (y axis).

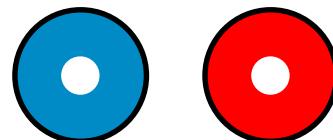
This plotting allows us to see the rich internal structure of this experiment – note the alternating Fe-Pt atomic planes.

Common Visualization Task – 3D Atoms

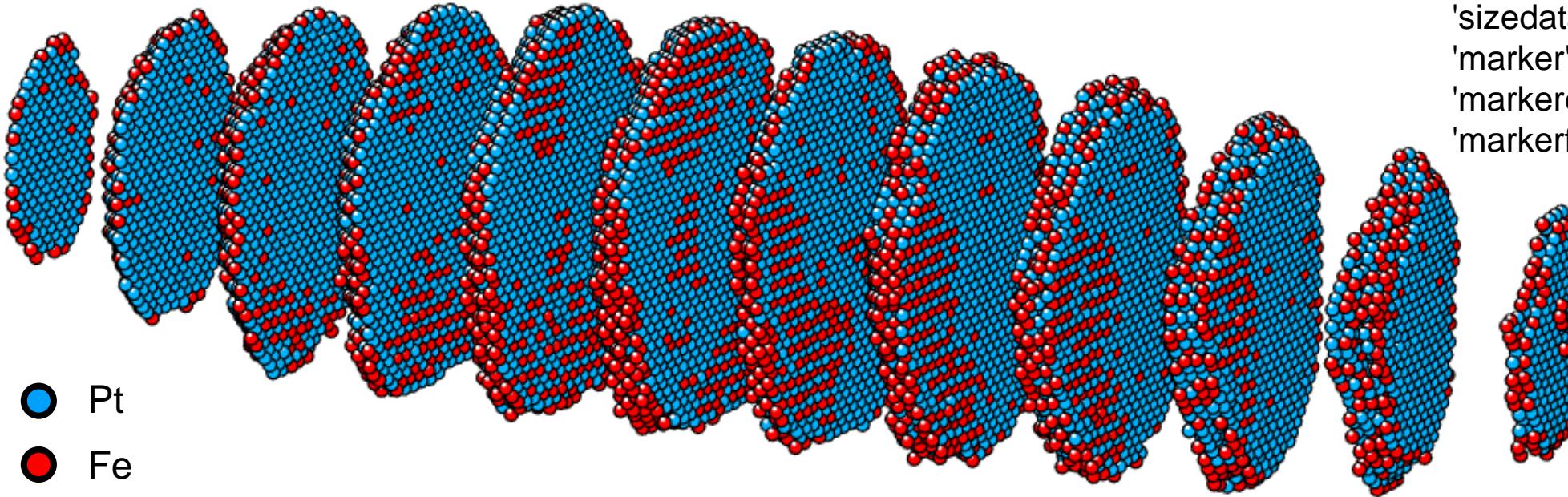


```
scatter3( ...  
atomPos(:,1),...  
atomPos(:,2),...  
atomPos(:,3),...  
'sizedata',atomSizeTint,...  
'marker','o',...  
'markeredgecolor','none',...  
'markerfacecolor',atomColorTint);
```

Let's make a fake specular reflection spot:



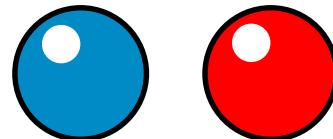
Common Visualization Task – 3D Atoms



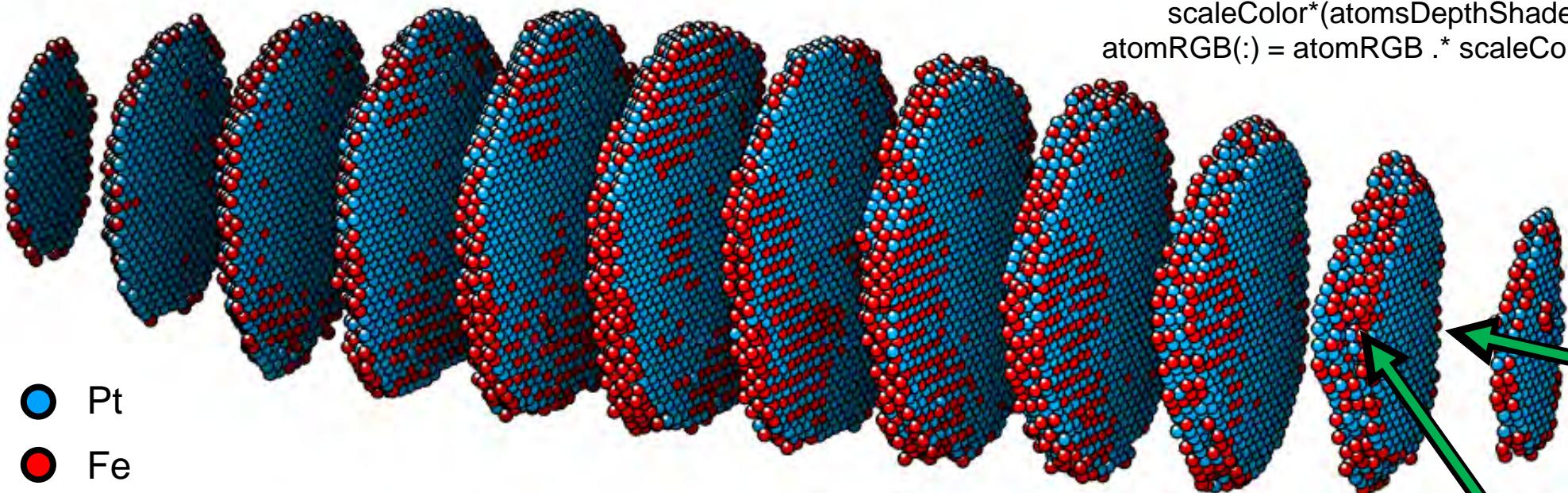
scatter3(...

```
atomPos(:,1) + shiftTint(1),...  
atomPos(:,2) + shiftTint(2),...  
atomPos(:,3) + shiftTint(3),...  
'sizedata',atomSizeTint,...  
'marker','o',...  
'markeredgecolor','none',...  
'markerfacecolor',atomColorTint);
```

Let's make a fake specular reflection spot:



Common Visualization Task – 3D Atoms



● Pt

● Fe

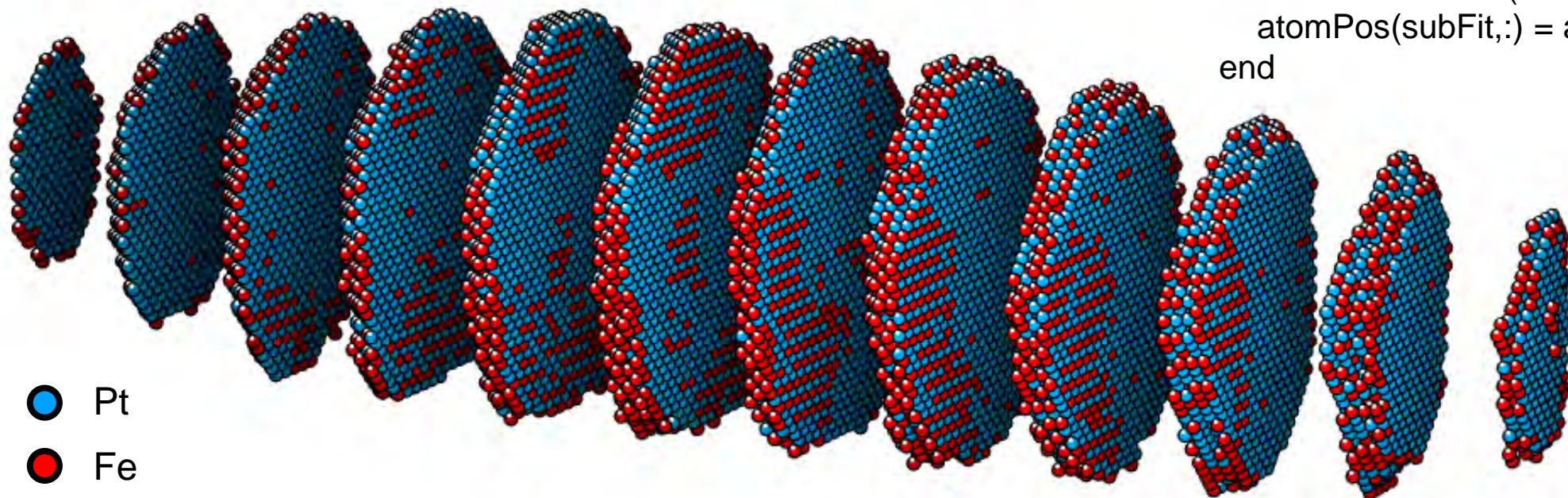
```
scaleColor = distCamera - min(distCamera);
scaleColor(:) = scaleColor / max(scaleColor);
scaleColor(:) = 3*scaleColor(:).^2 - 2*scaleColor(:).^3;
scaleColor(:) = scaleColor.^atomsDepthPower;
scaleColor = atomsDepthShade(1) + ...
scaleColor*(atomsDepthShade(2) - atomsDepthShade(1));
atomRGB(:) = atomRGB .* scaleColor;
```

Sites far from the camera are shaded to a darker color.

Sites closest to the camera are left alone.

This is a subtle effect to create a 3D view – “depth cueing”

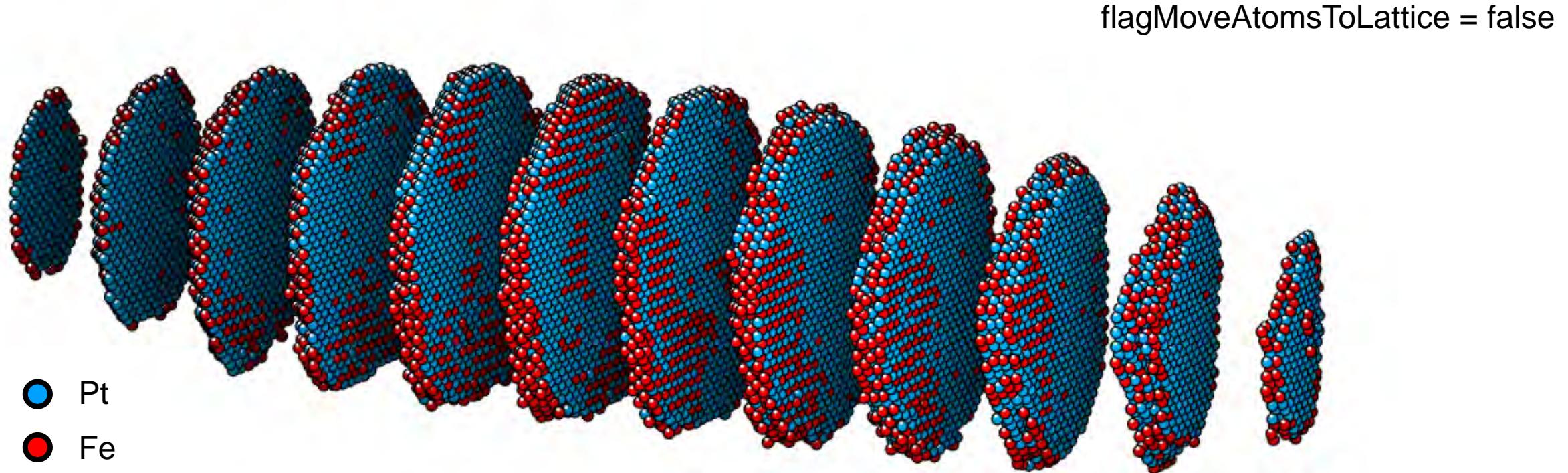
Common Visualization Task – 3D Atoms



```
if flagMoveAtomsToLattice == true
    subFit = ~isinf(sum(atomLattice,2)) ...
        & ~isnan(sum(atomLattice,2));
    lat = atomLattice(subFit,:)\atomPos(subFit,:);
    atomPos(subFit,:) = atomLattice(subFit,:)*lat;
end
```

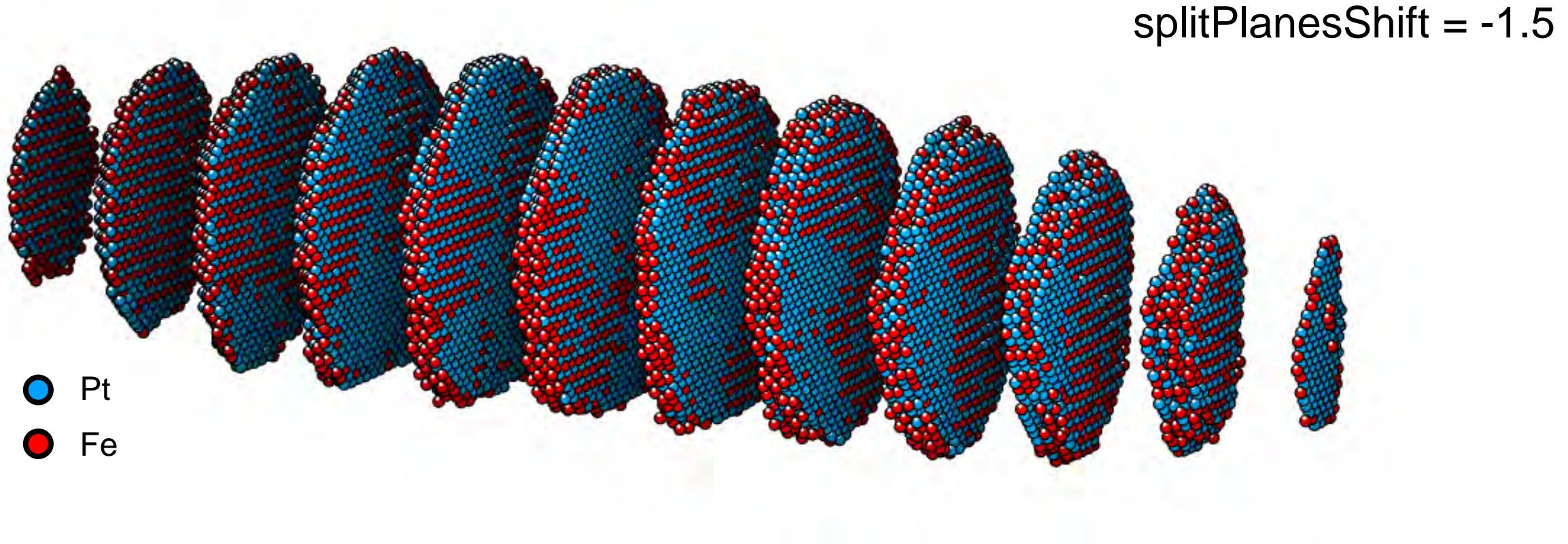
This line is complex – moving the atoms from their measured sites to best fit lattice.

Common Visualization Task – 3D Atoms



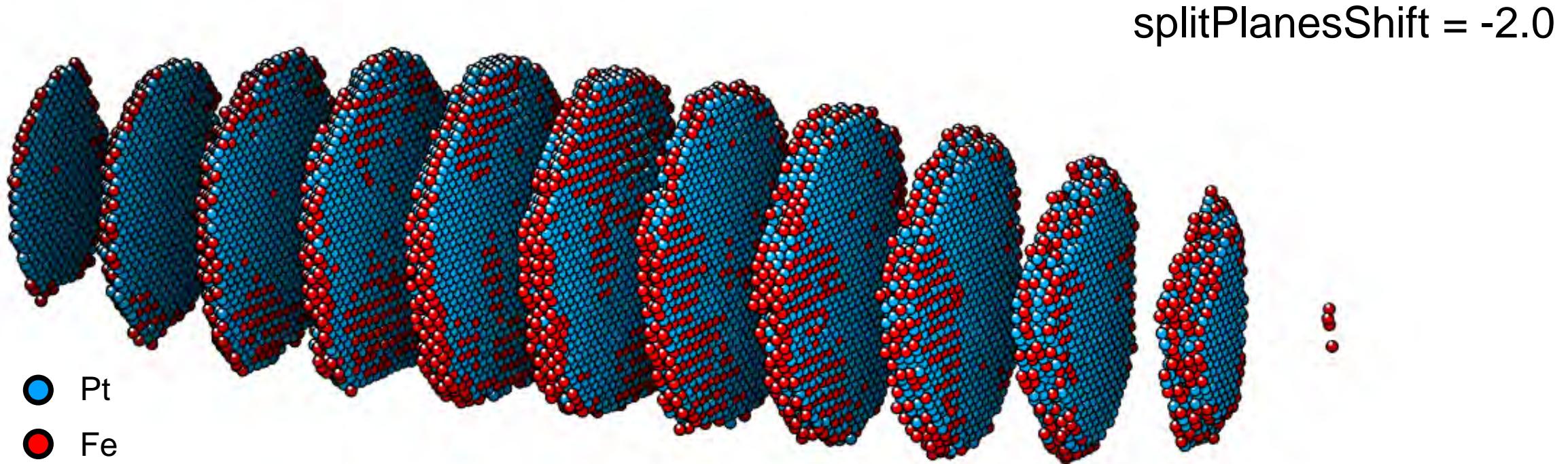
Turning that flag back off – flags are useful to plot multiple conditions / parameters.

Common Visualization Task – 3D Atoms



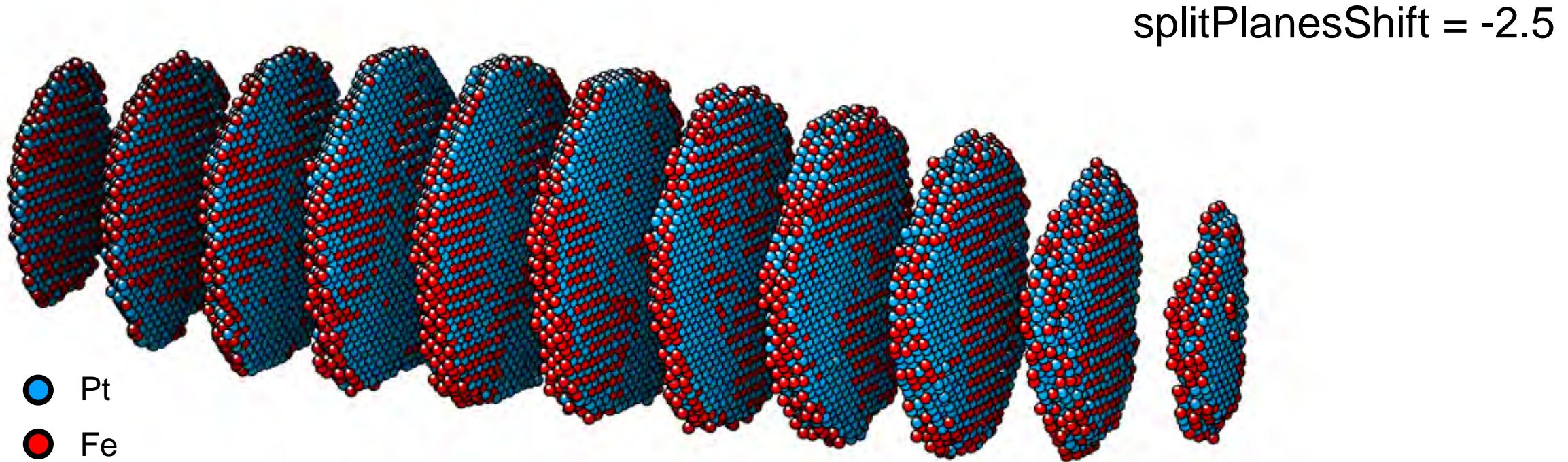
Shifting the planes we are cutting at.

Common Visualization Task – 3D Atoms



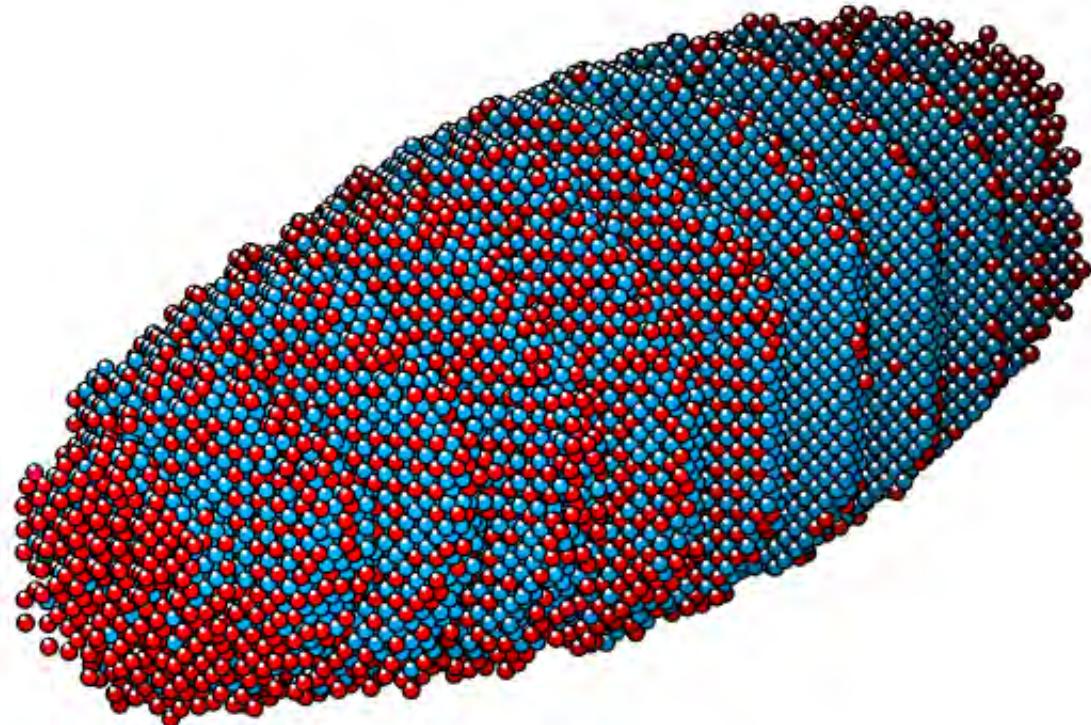
Shifting the planes we are cutting at.

Common Visualization Task – 3D Atoms



Shifting the planes we are cutting at.

Common Visualization Task – 3D Atoms

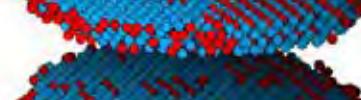
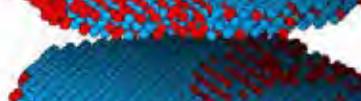
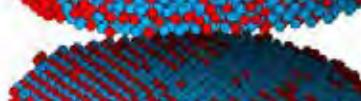
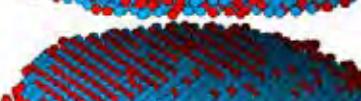
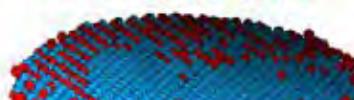
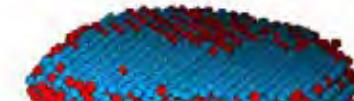


splitPlanesIndex = 1

- Pt
- Fe

Splitting apart the particle along the 1st dimension.

Common Visualization Task – 3D Atoms



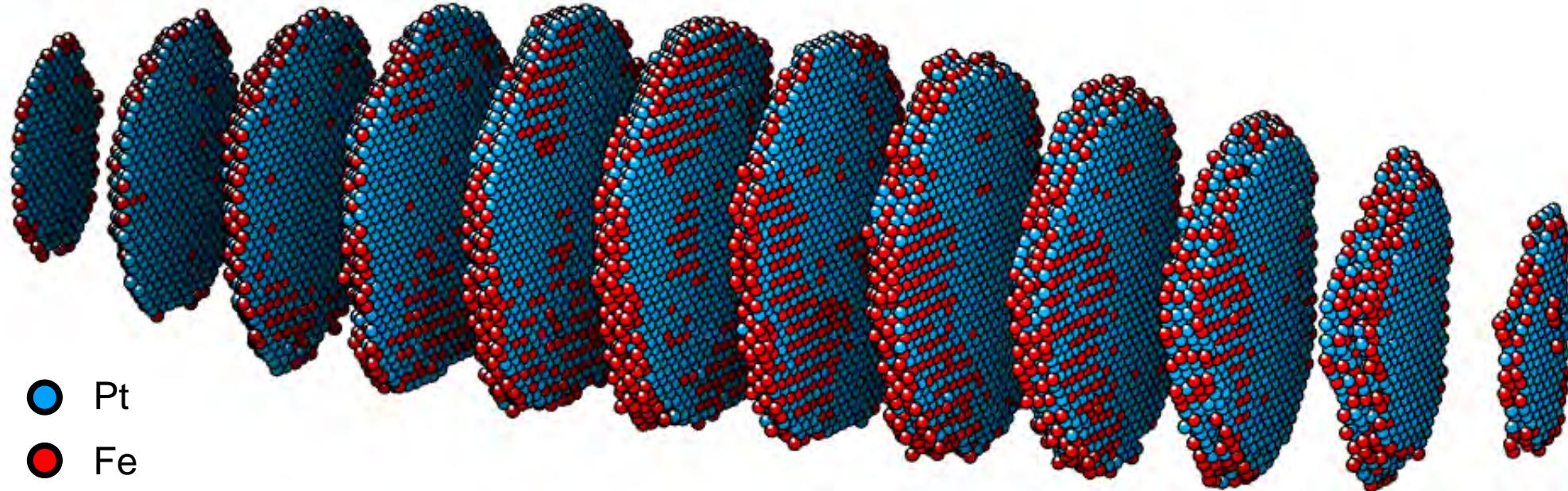
splitPlanesIndex = 3

● Pt

● Fe

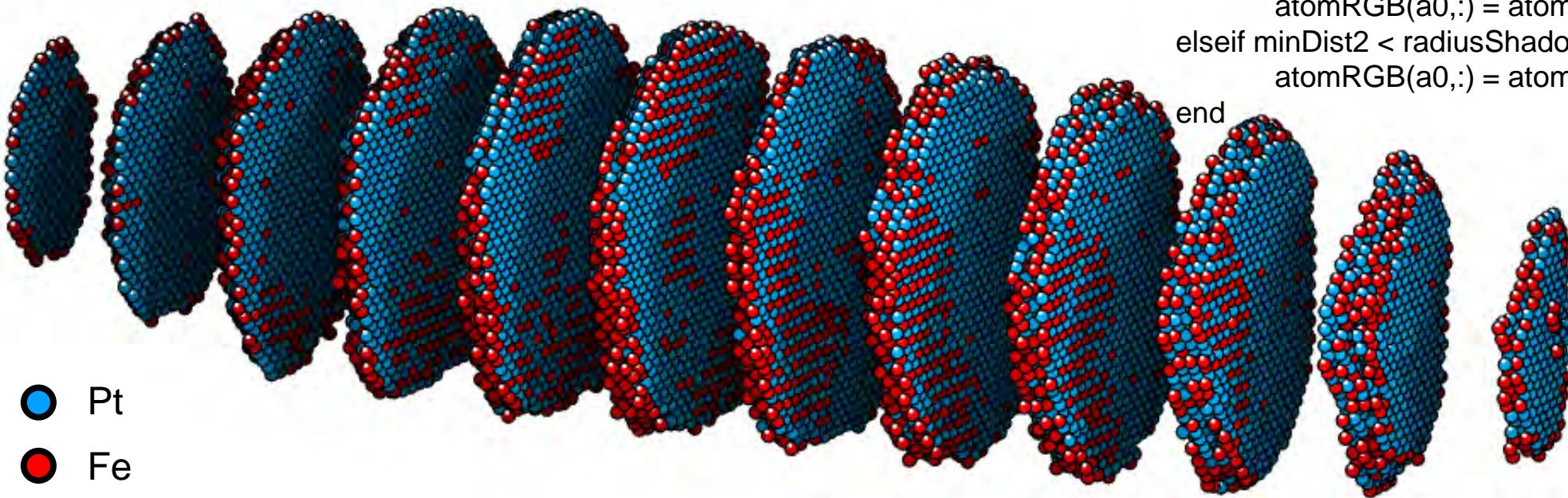
Splitting apart the particle along
the 3rd dimension.

Common Visualization Task – 3D Atoms



So far we are just manipulating parameters and positions – can we do more?

Common Visualization Task – 3D Atoms

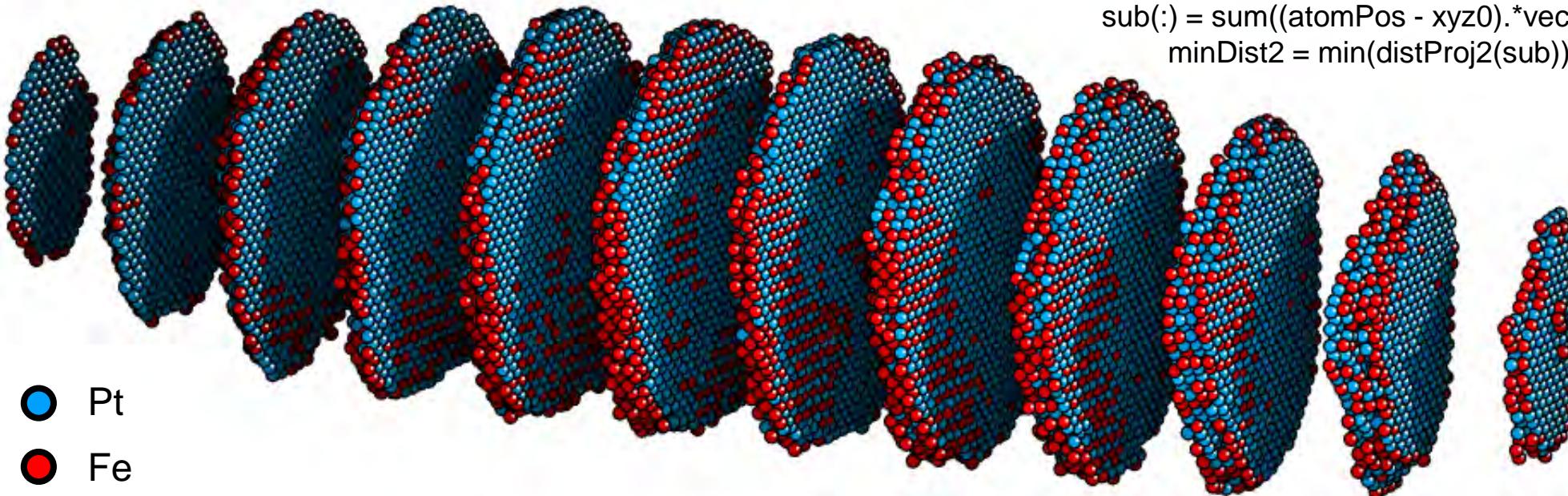


Ray traced shadows!

```
distProj2() = sum(cross(atomPos - xyz0, vecRep).^2,2);
sub() = sum((atomPos - xyz0).*vecShadow,2) > 0;
minDist2 = min(distProj2(sub));
```

```
if minDist2 < radiusShadow2(1)
    atomRGB(a0,:) = atomRGB(a0,:)*shadingShadow(1);
elseif minDist2 < radiusShadow2(2)
    atomRGB(a0,:) = atomRGB(a0,:)*shadingShadow(2);
end
```

Common Visualization Task – 3D Atoms

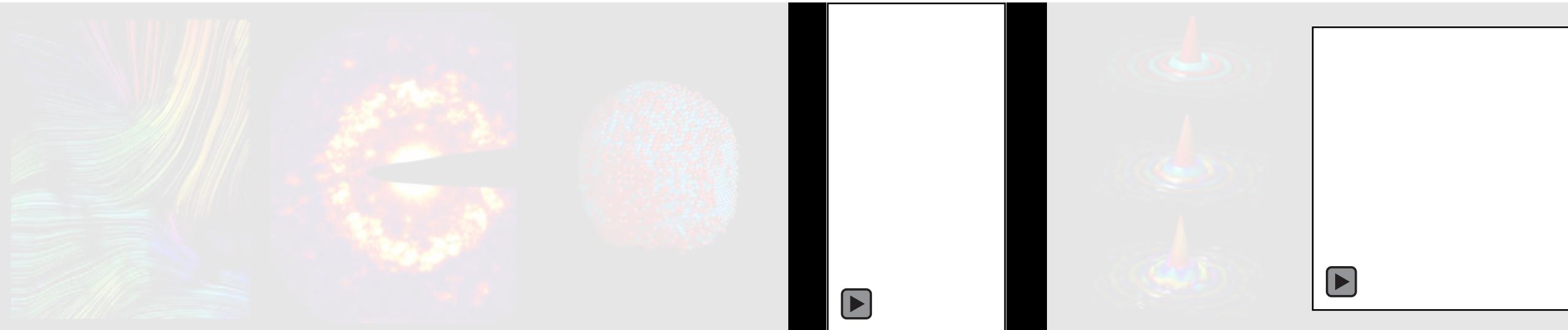


description

```
distProj2(:) = sum(cross(atomPos - xyz0, vecRep).^2,2);
sub(:) = sum((atomPos - xyz0).*vecShadow,2) > 0;
minDist2 = min(distProj2(sub));
```

```
if minDist2 < radiusShadow2(1)
    atomRGB(a0,:) = atomRGB(a0,:)*shadingShadow(1);
    subShadowed(a0) = true;
elseif minDist2 < radiusShadow2(2)
    atomRGB(a0,:) = atomRGB(a0,:)*shadingShadow(2);
    subShadowed(a0) = true;
end
```

Outline



1

Basic concepts
when making
figures and
movies – raster
vs vector.

2

Scaling,
coloring and
saving scalar
and vector
imaging data.

3

Plotting **points**
in 3D, highlight
internal features
– e.g. atomic
coordinates.

4

Animating 3D
points, making
movies
– e.g. atomic
coordinates.

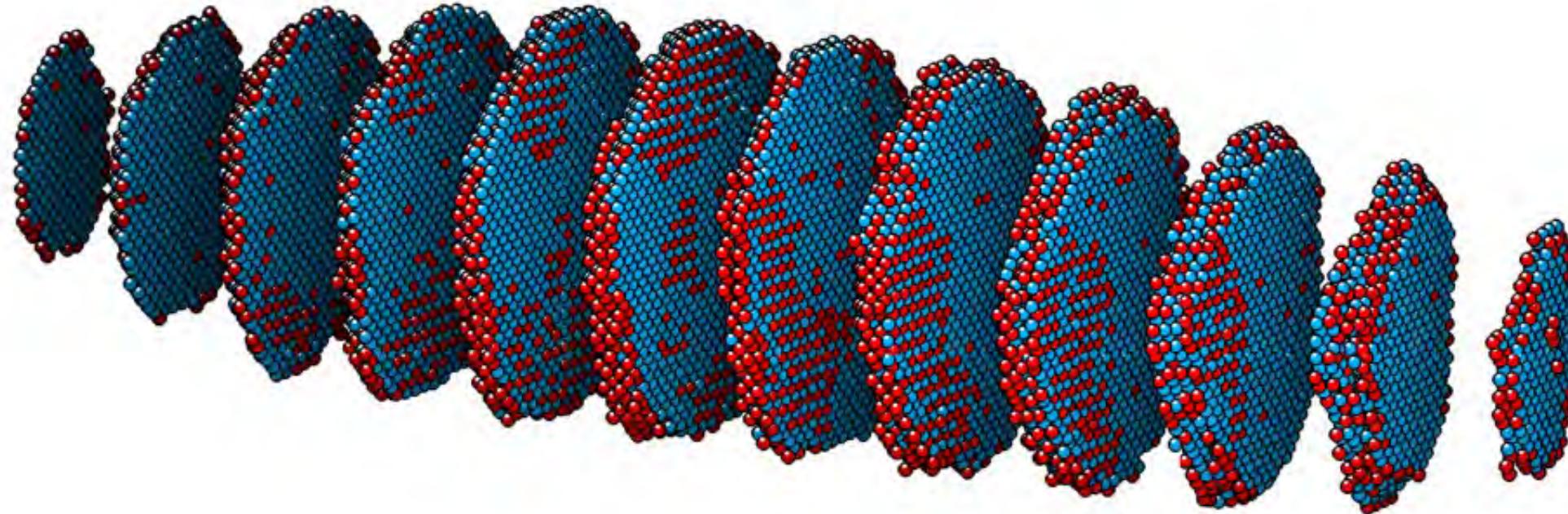
5

Building physics
visualization
and simulation
directly into
figures.

6

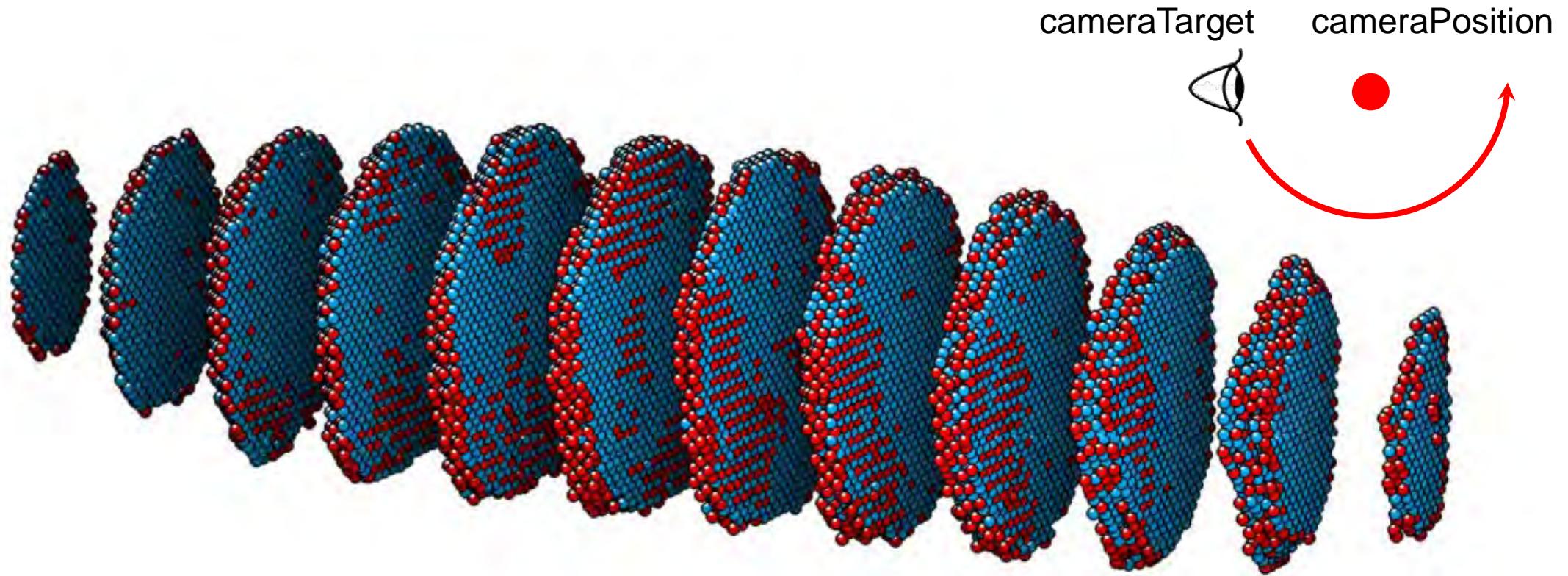
Making simple
animations to
explain
concepts in
physics.

Making Movies from Images Using FFmpeg



Each time plotting script is called, move camera, save image.

Making Movies from Images Using FFmpeg



Each time plotting script is called, move camera, save image.

Making Movies from Images Using Code

Step 1 – Modify drawing script to have input variables.

Step 2 – Write a second script to repeatedly call the first script, changing the inputs.

Step 3 – Each time the drawing script is called, render an image, output to file.

Step 4 – Use FFmpeg to stitch / encode images into a movie. Handbrake edit if needed.

Tutorial example, animated 3D atoms plot with `animAtoms01a()`, `animAtoms01b()`

Making Movies from Images Using FFmpeg

FFmpeg command line example for low frame rate movies:

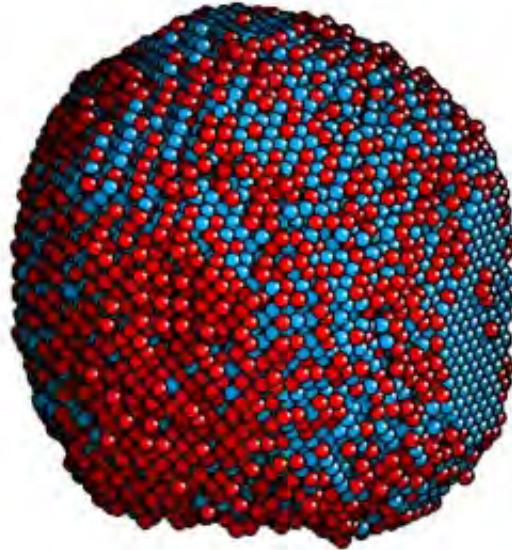
```
ffmpeg  
-f image2  
-i image%04d.png  
-vcodec mpeg4  
-qmax 4  
-vf setpts=4.0*PTS  
movieFileName.mp4
```

FFmpeg command line example for high frame rate movies:

```
ffmpeg  
-f image2 -r 60 -i image%04d.png  
-c:v libx264 -preset slow  
-profile:v high -level:v 4.0  
-pix_fmt yuv420p -filter:v fps=fps=60  
movieFileName.mp4
```

These are just examples – lots of settings to tweak in FFmpeg!

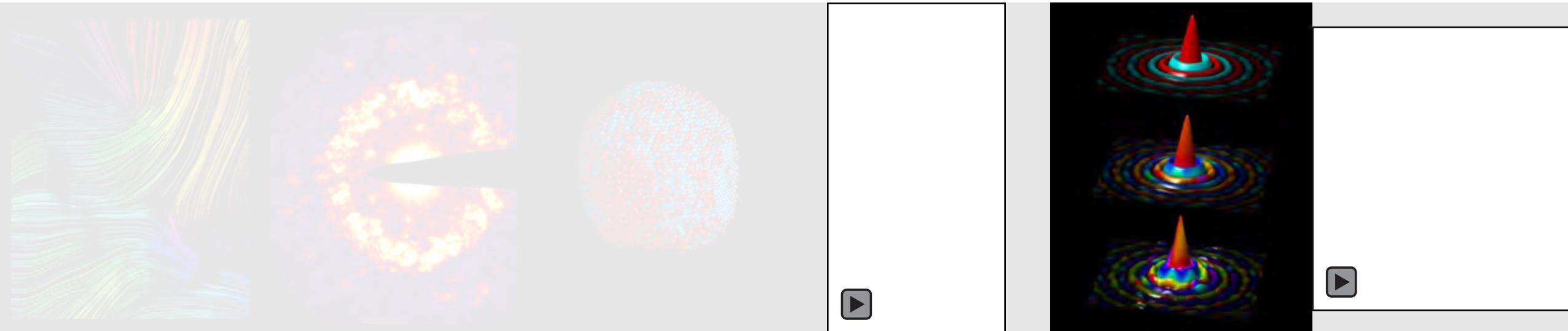
Making Movies from Images Using FFMPEG



- Pt
- Fe

Each time plotting script is called, change `splitPlanes` from 0 to 70, save image.

Outline



1

Basic concepts
when making
figures and
movies – raster
vs vector.

2

Scaling,
coloring and
saving scalar
and vector
imaging data.

3

Plotting **points**
in 3D, highlight
internal features
– e.g. atomic
coordinates.

4

Animating 3D
points, making
movies
– e.g. atomic
coordinates.

5

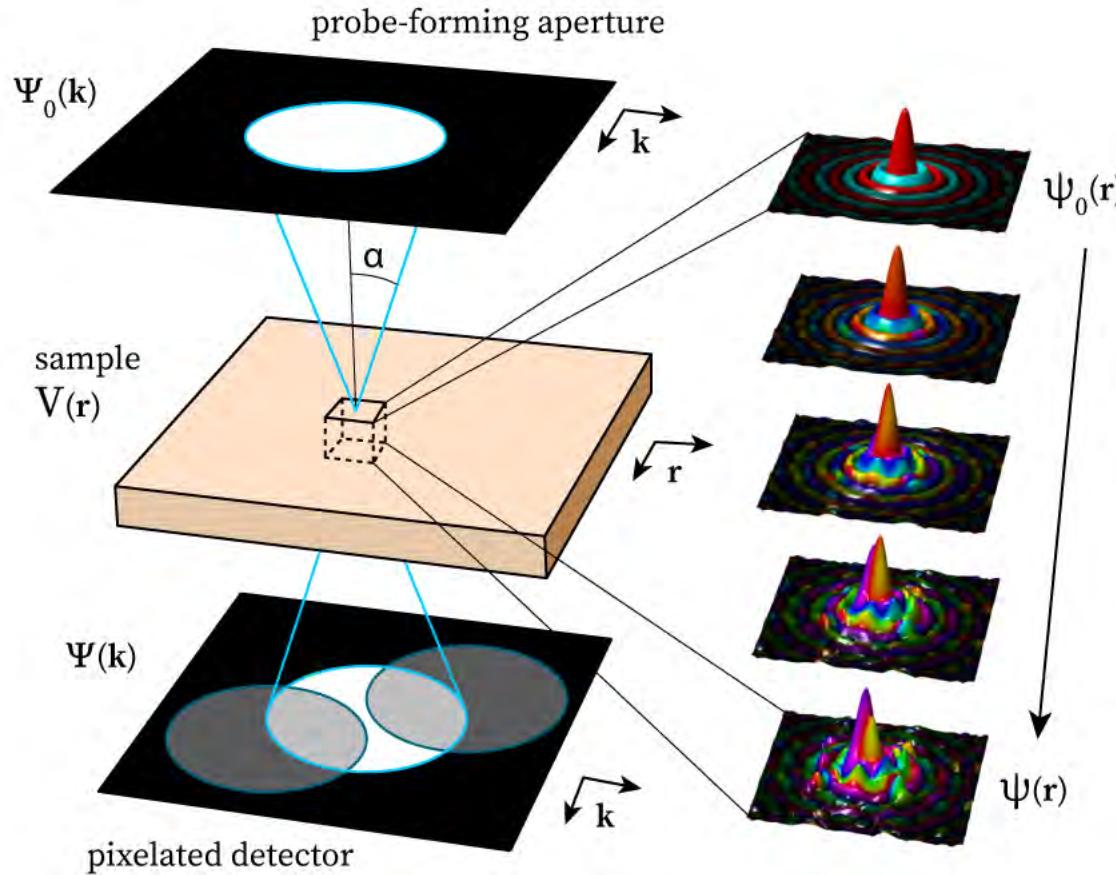
Building physics
visualization
and simulation
directly into
figures.

6

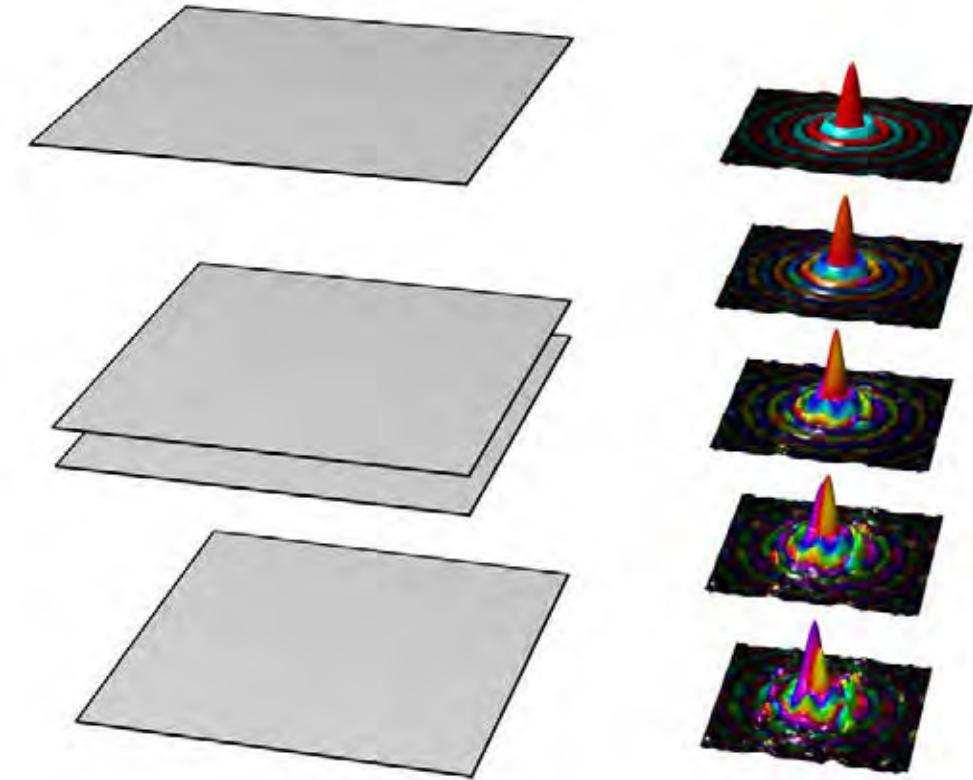
Making simple
animations to
explain
concepts in
physics.

Building Physics into Figure Generation

Final figure – STEM probe wavefunction

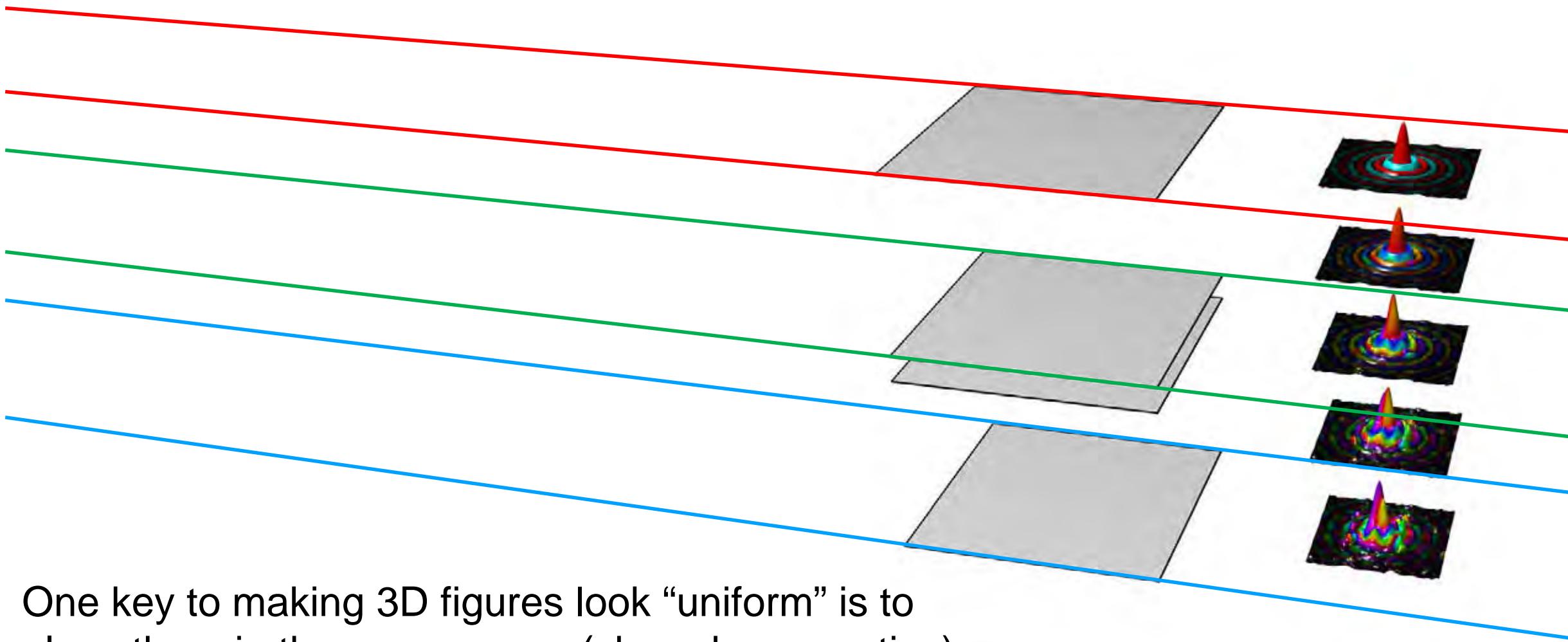


Original figure generated in Matlab



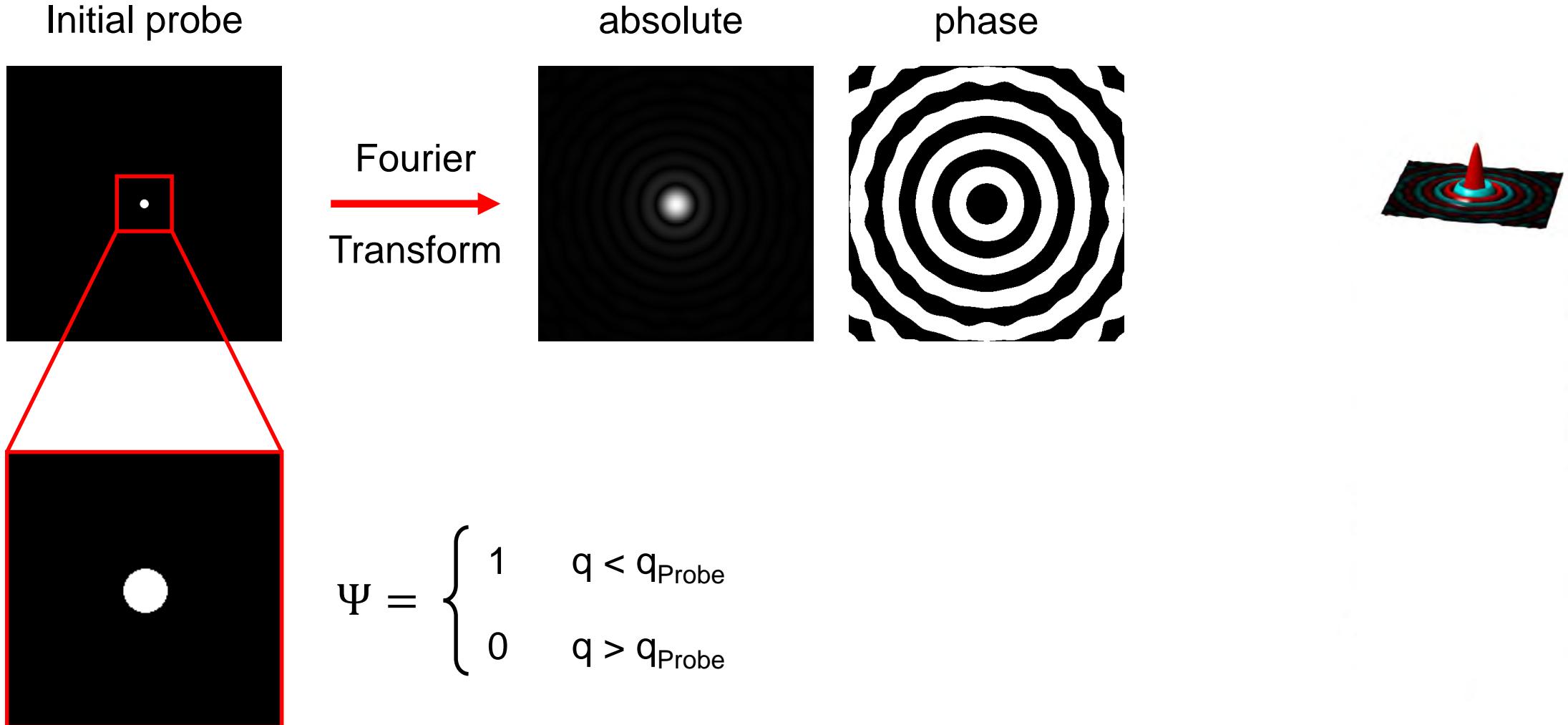
To modify: play with settings, run **schematic01()**;

Building Physics into Figure Generation



One key to making 3D figures look “uniform” is to place them in the same space (shared perspective).

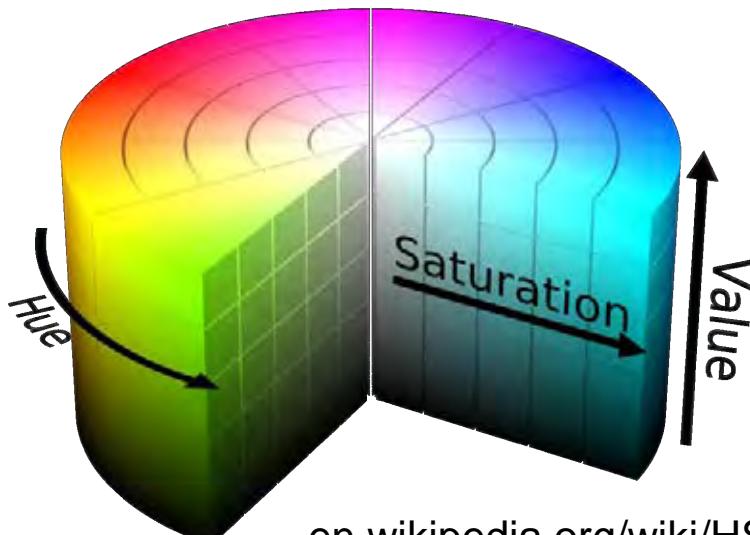
Building Physics into Figure Generation



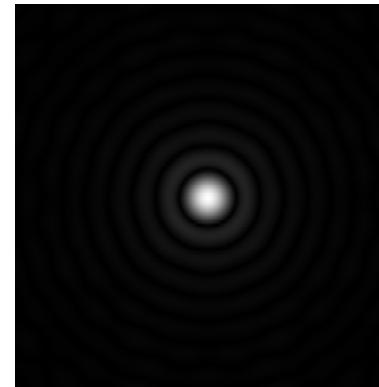
Building Physics into Figure Generation

How can we represent complex numbers?

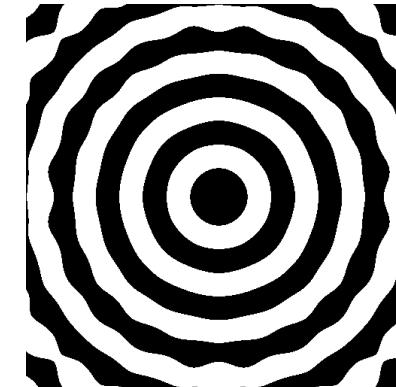
Two values per pixel requires a vector representation with at least 2 coordinates:



absolute

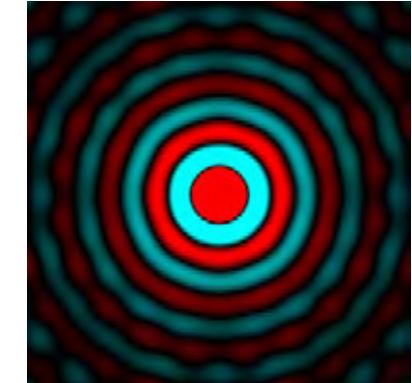


phase



We can map the amplitude to value (or saturation), and phase to the hue.

Hue wraps at $360^\circ / 2\pi$ rads, making it a good choice for phase.

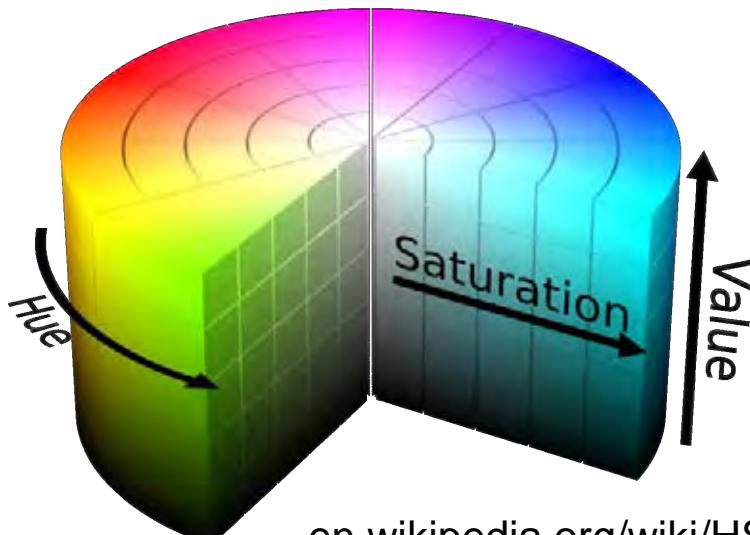


Zero phase color is arbitrary

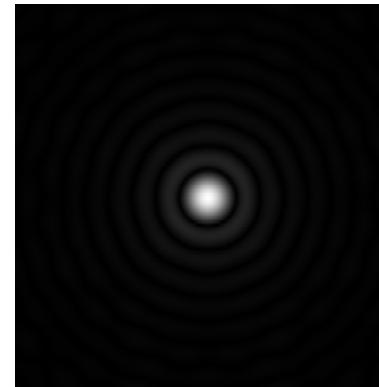
Building Physics into Figure Generation

How can we represent complex numbers?

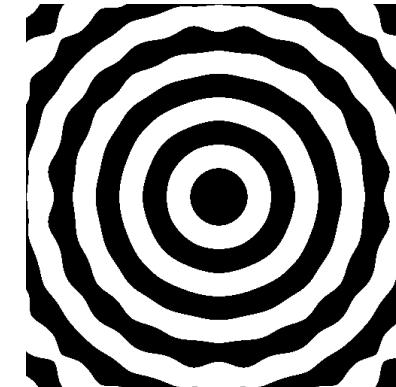
Two values per pixel requires a vector representation with at least 2 coordinates:



absolute

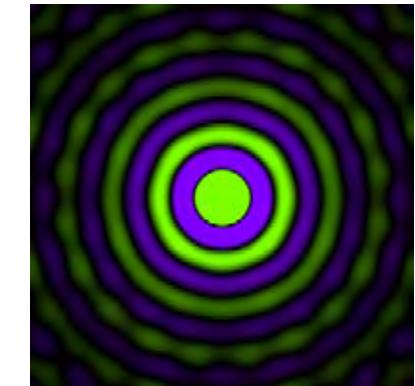


phase



We can map the amplitude to value (or saturation), and phase to the hue.

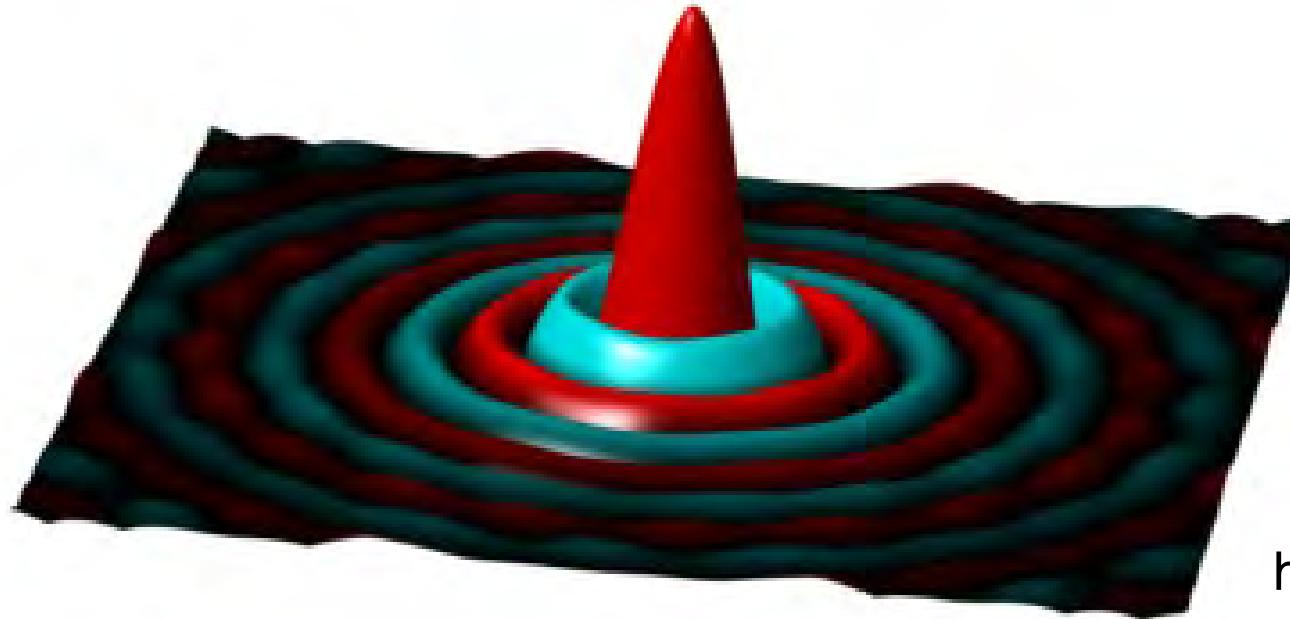
Hue wraps at $360^\circ / 2\pi$ rads, making it a good choice for phase.



Zero phase color is arbitrary

Building Physics into Figure Generation

To make it easier to visualize the wave amplitude, we can add height:

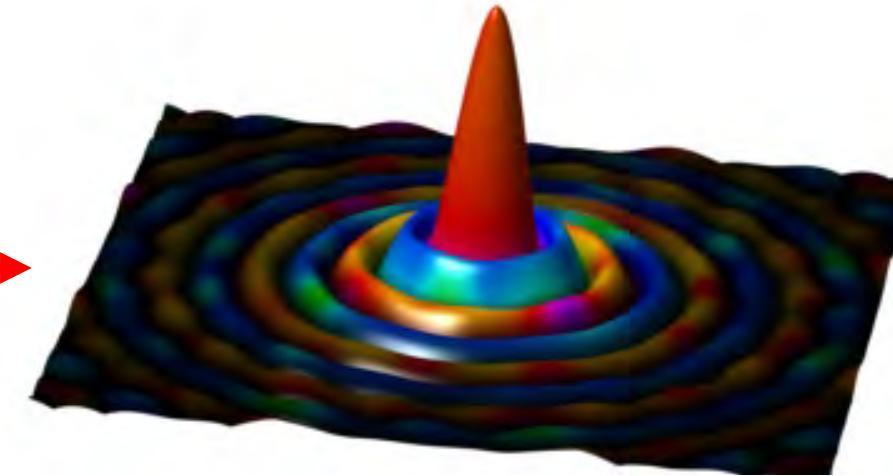


$h = \text{abs}(\psi).^0.8 * 100 * \text{scale}(a0);$

Scaling is tweaked until it looks good – goal is to visualize the complex wavefunction in one image.

Building Physics into Figure Generation

Next up is evolution of the wavefunction during scattering / propagation:



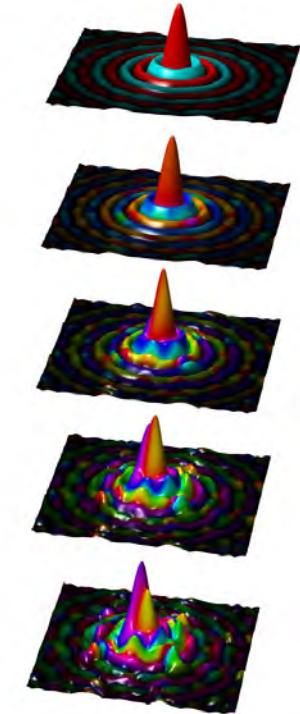
```
Psi = Psi.*prop;  
Psi = fft2(ifft2(Psi) .* trans);
```

```
prop = exp(-1i*pi*0.4*(q_x^2 + q_y^2));
```

```
trans = exp(1i*pot*0.5*2);
```

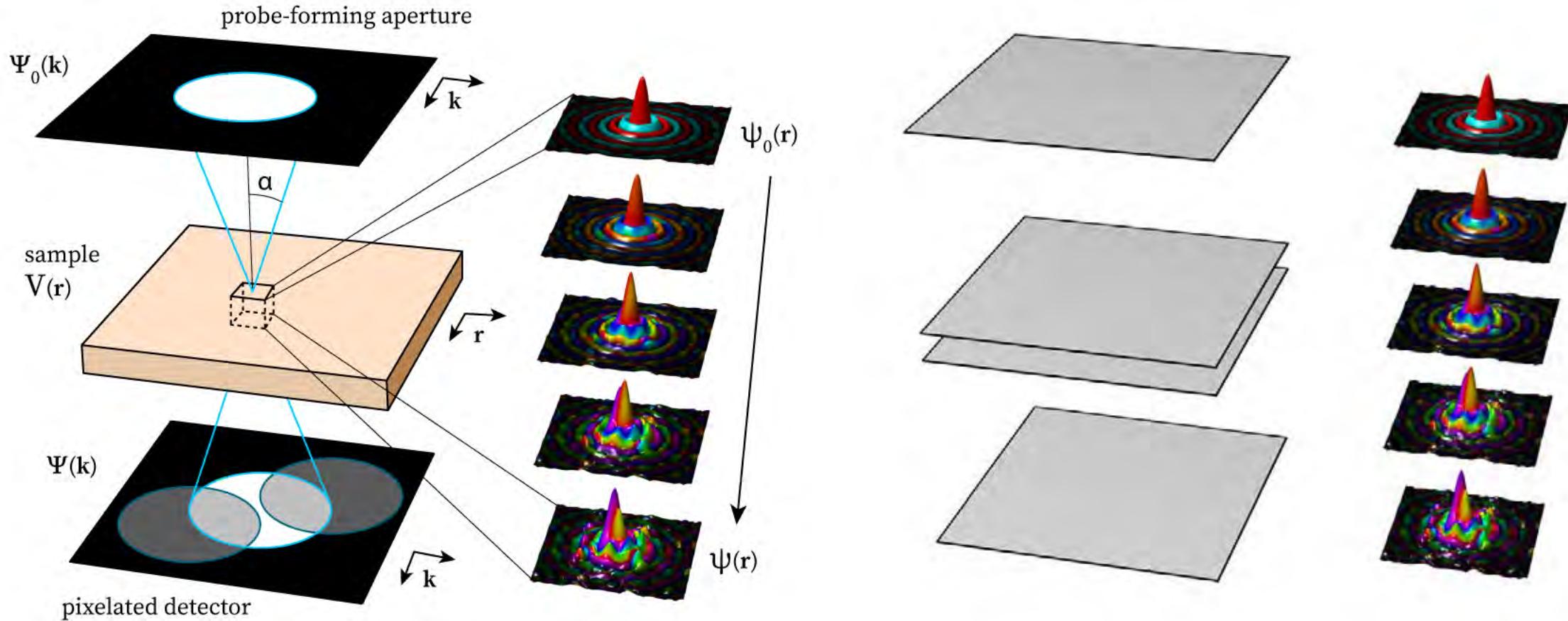
Evolution via Schrödinger eq:

$$i\hbar \frac{\partial}{\partial t} \psi(\vec{r}) = \hat{H}(\vec{r}) \psi(\vec{r}) \rightarrow \frac{\partial}{\partial z} \psi(\vec{r}) = \frac{i\lambda}{4\pi} \nabla_{xy}^2 \psi(\vec{r}) + i\sigma V(\vec{r}) \psi(\vec{r})$$



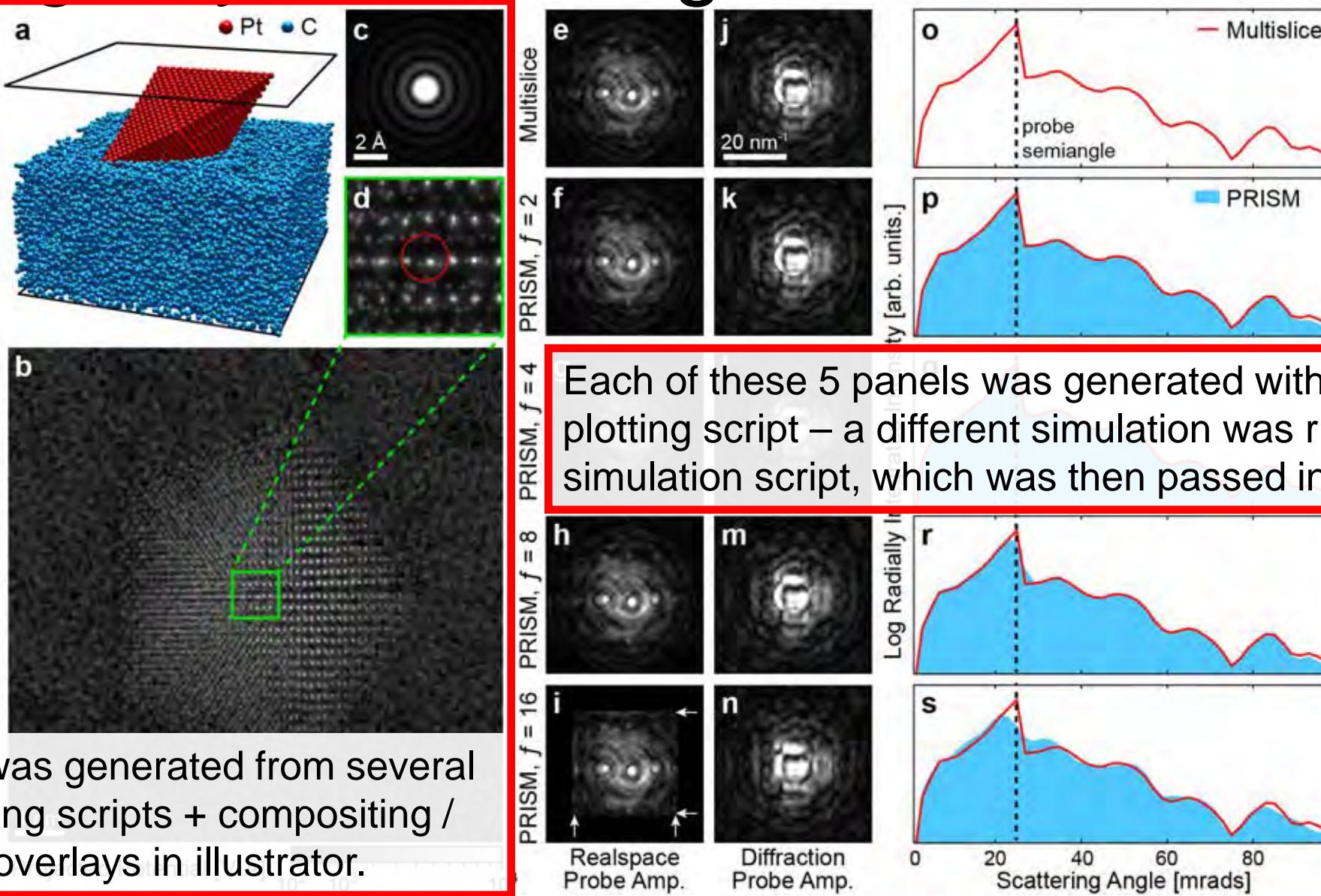
Building Physics into Figure Generation

Add 5 time points, composite in illustrator:

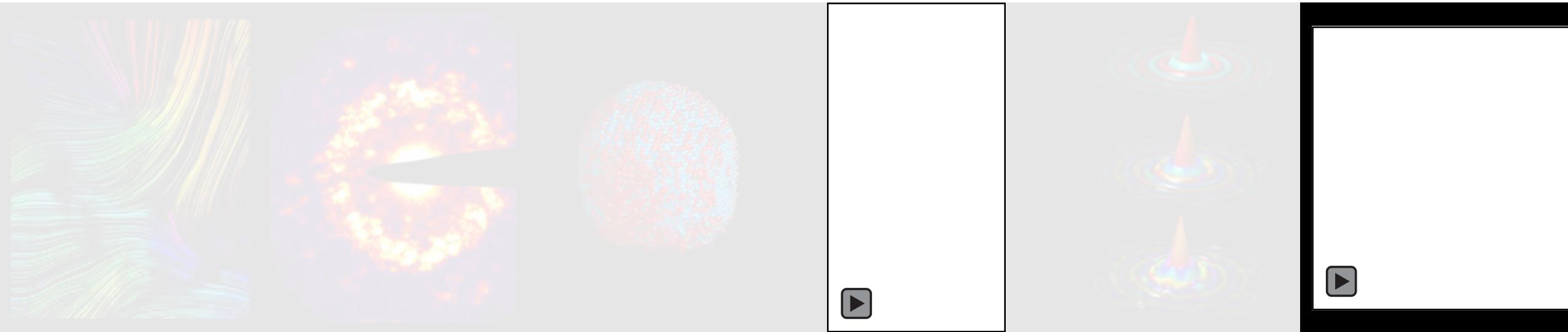


Labels and text added, saved in PDF format

Building Physics into Figure Generation



Outline



1

Basic concepts
when making
figures and
movies – raster
vs vector.

2

Scaling,
coloring and
saving scalar
and vector
imaging data.

3

Plotting **points**
in 3D, highlight
internal features
– e.g. atomic
coordinates.

4

Animating 3D
points, making
movies
– e.g. atomic
coordinates.

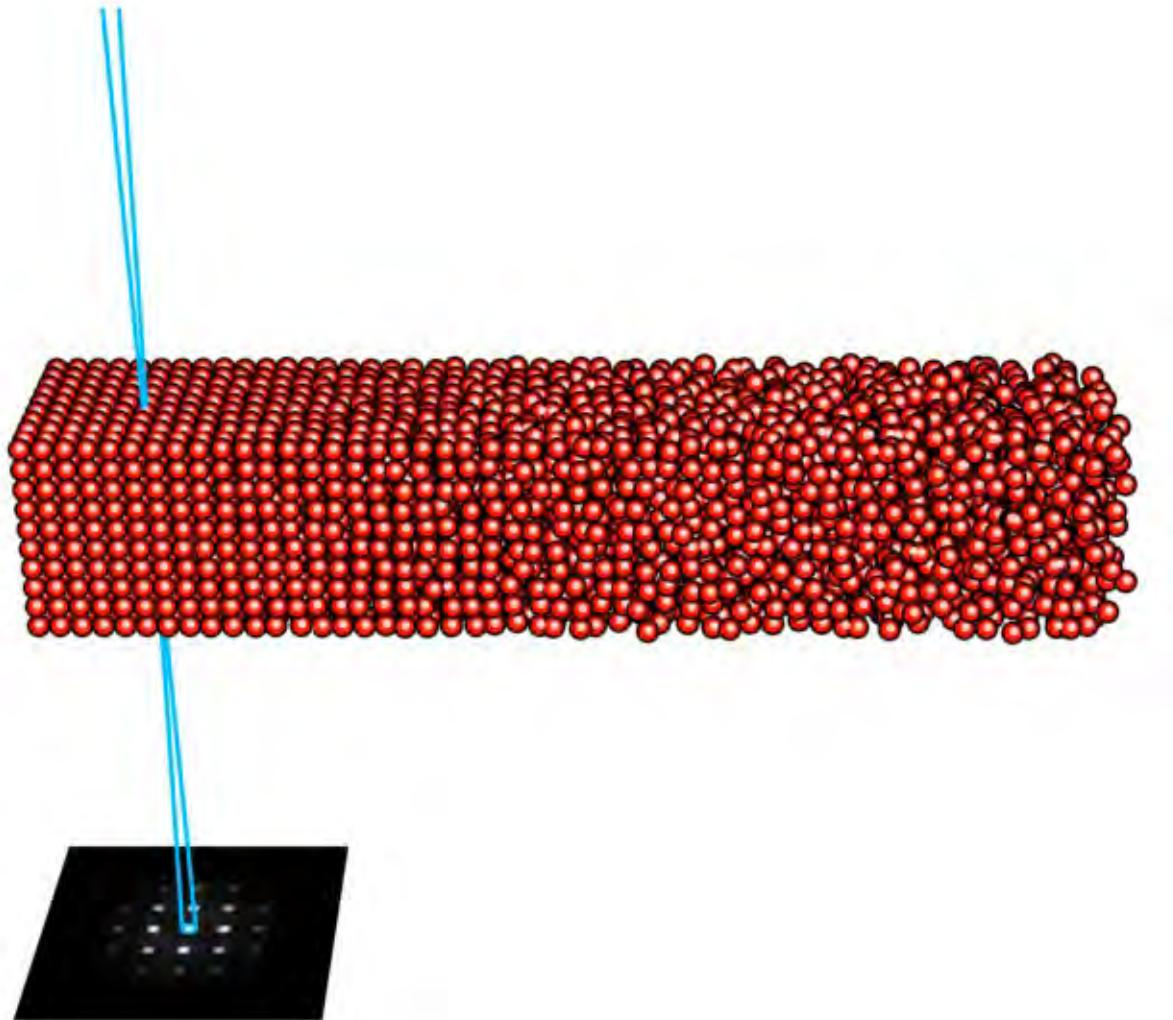
5

Building physics
visualization
and simulation
directly into
figures.

6

Making simple
animations to
explain
concepts in
physics.

Examples of Using Movies to Explain Physics

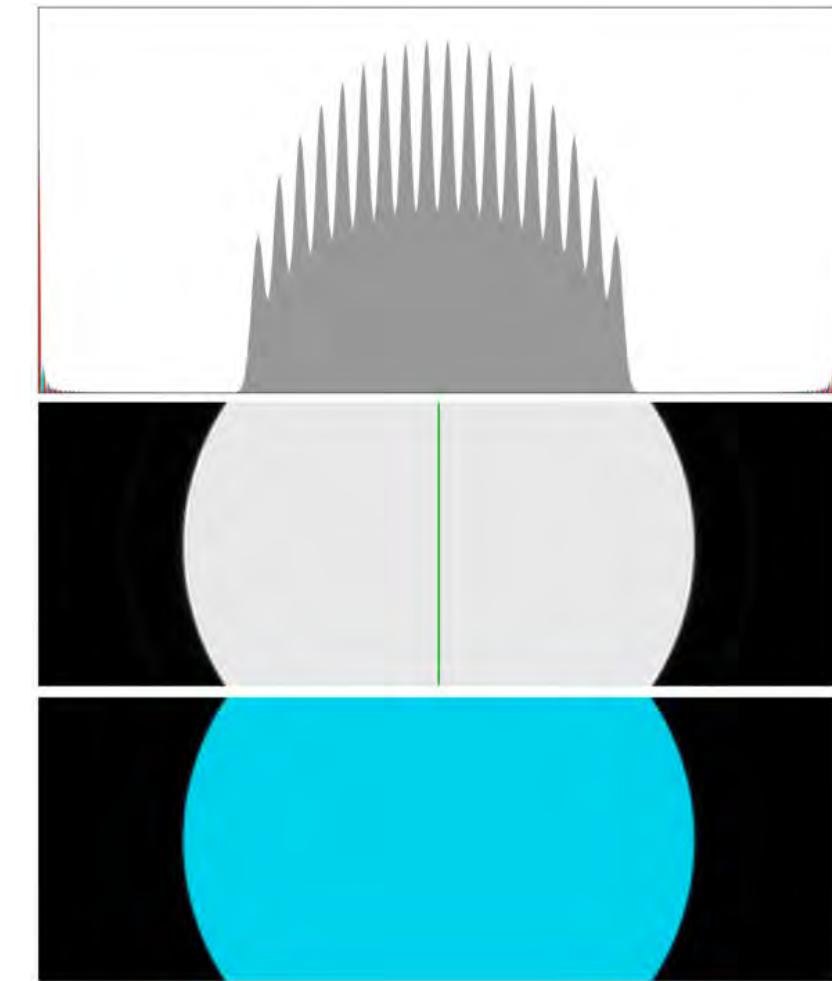
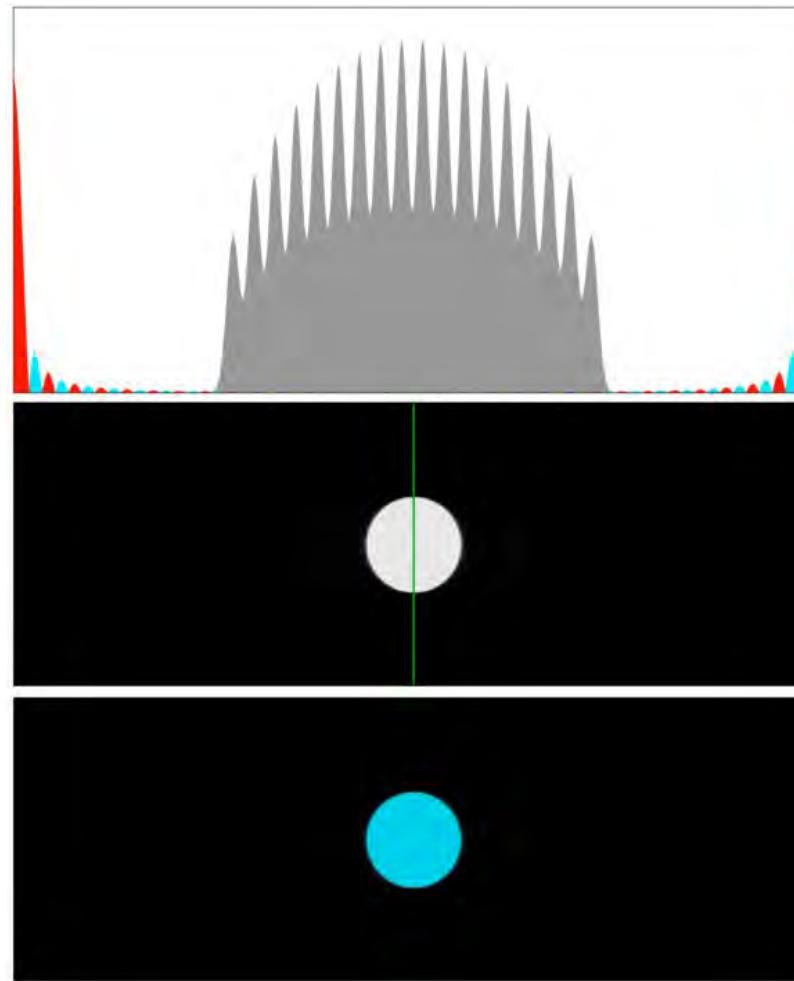


`animFEM01(probeSites, rScale)`
draws the scene,

`animFEM02()`
animates the atoms transforming
from ordered → disordered,

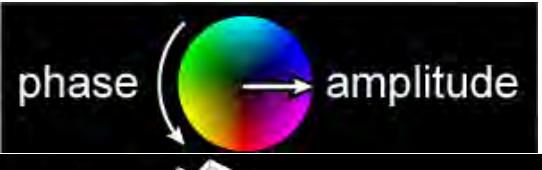
`animFEM03()`
animates STEM probe scanning.

Diff. Phase Contrast – Effect of Probe Semiangle

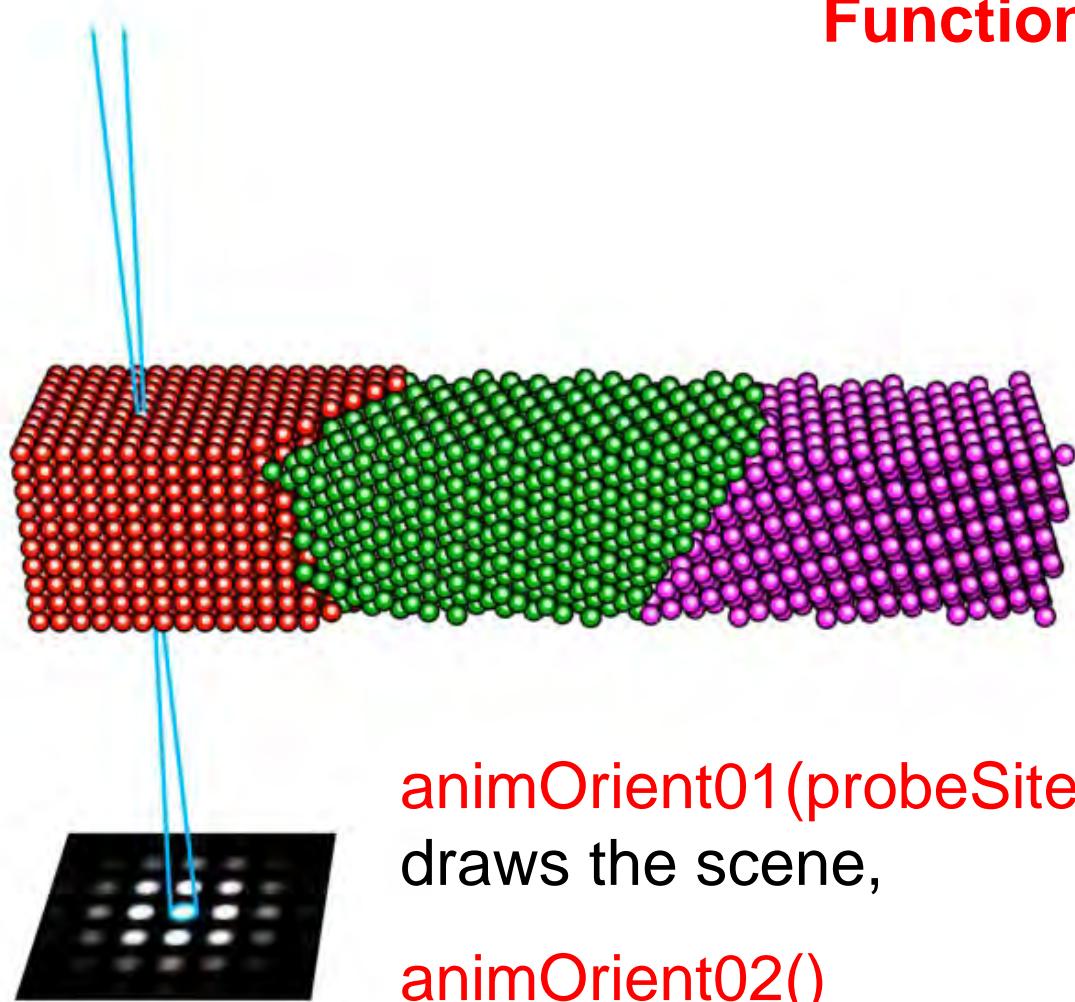


Same sim., plotting and output script for all movies – just change one number!

Try it:
`>> sDPC = DPC01();
>> DPC02(sDPC);`



Using a Movie to Explain Diffraction Physics



`animOrient01(probeSites)`
draws the scene,

`animOrient02()`
animates it.

Functions

`line(x,y,z,...)`

Probe is just 2 lines and a circle

`scatter3(x,y,z,...)`

Three grains tiled & rotated, each with 2 scatter plots for the atoms and tints.

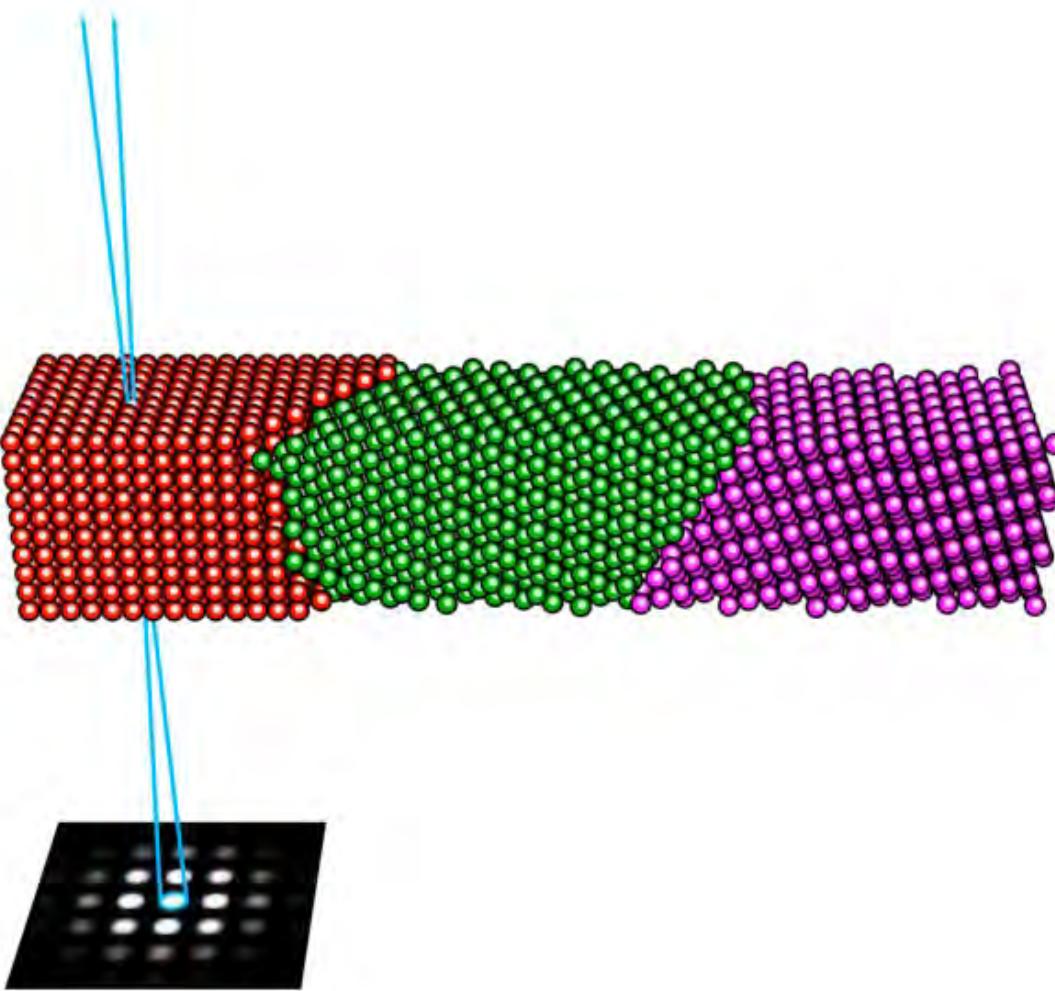
`sum(), conv2(), fft2(), ...`

Images are generated by summing atoms along dim 3, taking the FFT.

`warp(), conv2(), fft2(), ...`

Easy methods for placing a texture-mapped surface in 3D, modifying images.

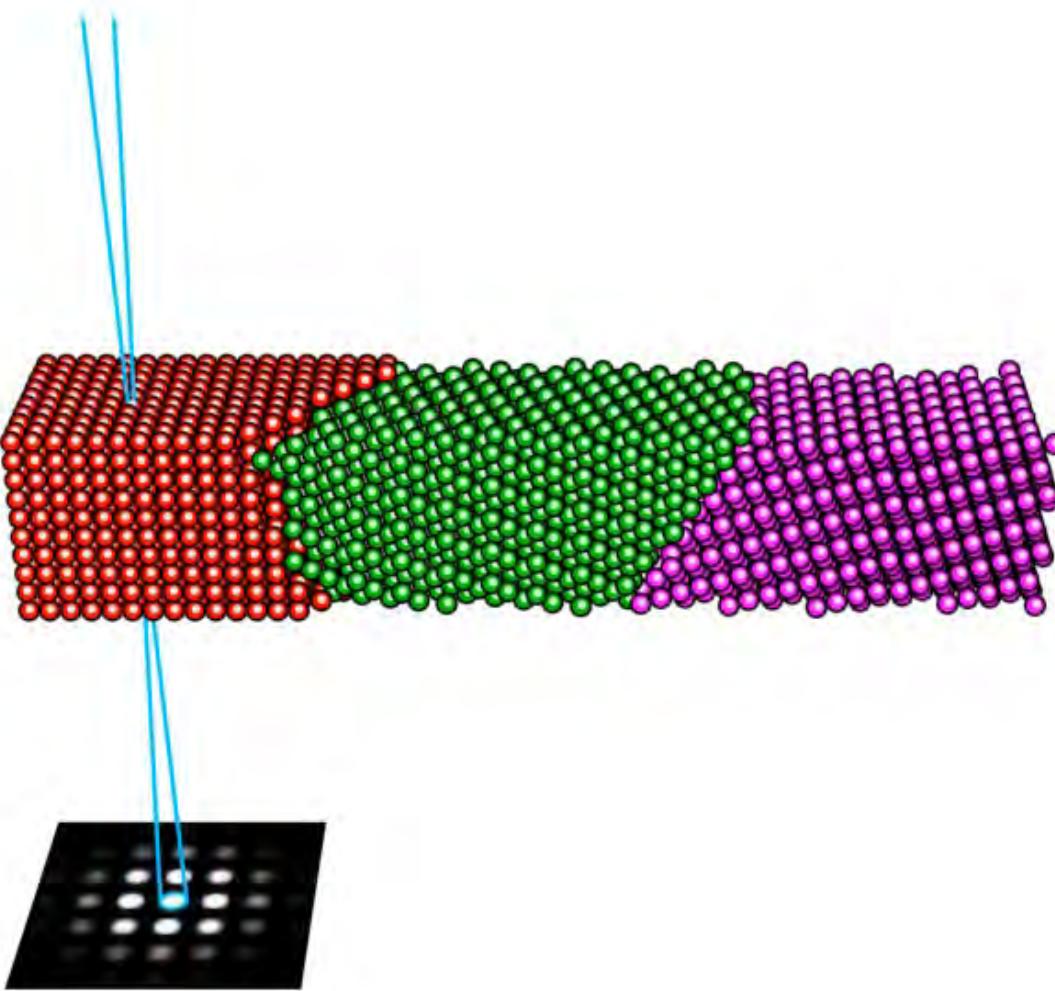
Using a Movie to Explain Diffraction Physics



Building a block of atoms

```
numAtomsMax = max(numAtoms);  
v = -round(numAtomsMax/2):round(numAtomsMax/2);  
[yy, xx, zz] = meshgrid(v, v, v);  
p0 = [xx(:) yy(:) zz(:)];
```

Using a Movie to Explain Diffraction Physics



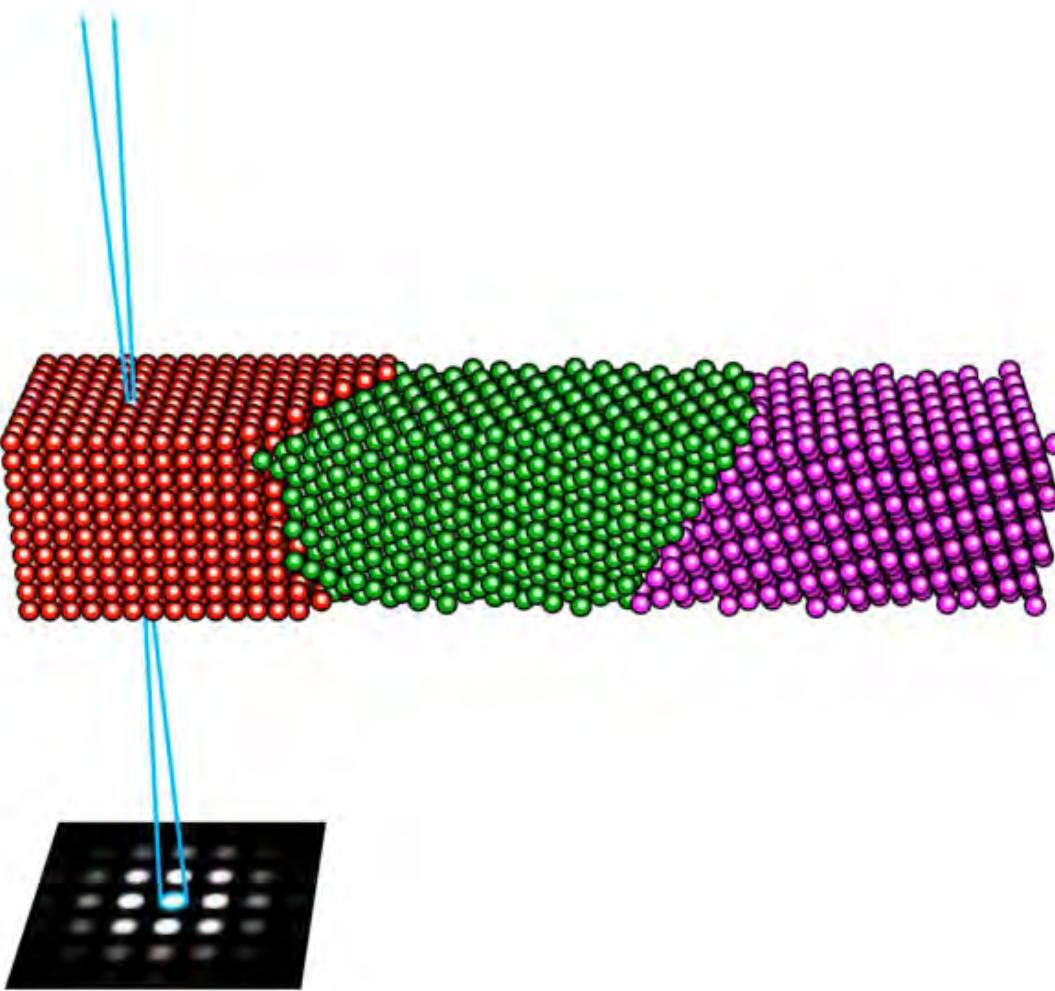
Rotating a block of atoms

```
t = [45 54 -18.43]*pi/180;  
m1 = [cos(t(1)) -sin(t(1)) 0;  
      sin(t(1)) cos(t(1)) 0;  
      0 0 1];  
m2 = [1 0 0;  
      0 cos(t(2)) -sin(t(2));  
      0 sin(t(2)) cos(t(2))];  
m3 = [cos(t(3)) -sin(t(3)) 0;  
      sin(t(3)) cos(t(3)) 0;  
      0 0 1];  
p2 = p2 * m1 * m2 * m3;
```

Euler angle rotations:

Axis	Matrix
z	m1
x	m2
z	m3

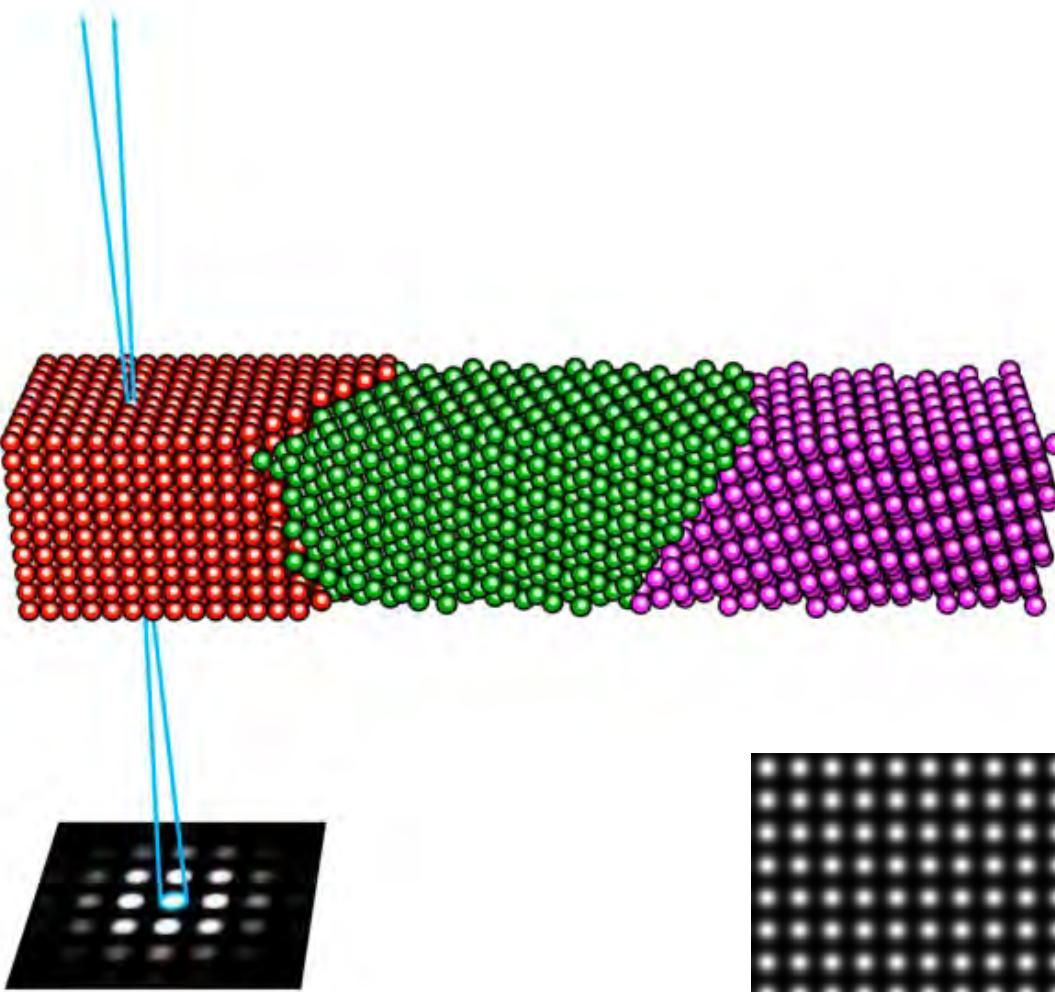
Using a Movie to Explain Diffraction Physics



Deleting atoms on one side of a plane:

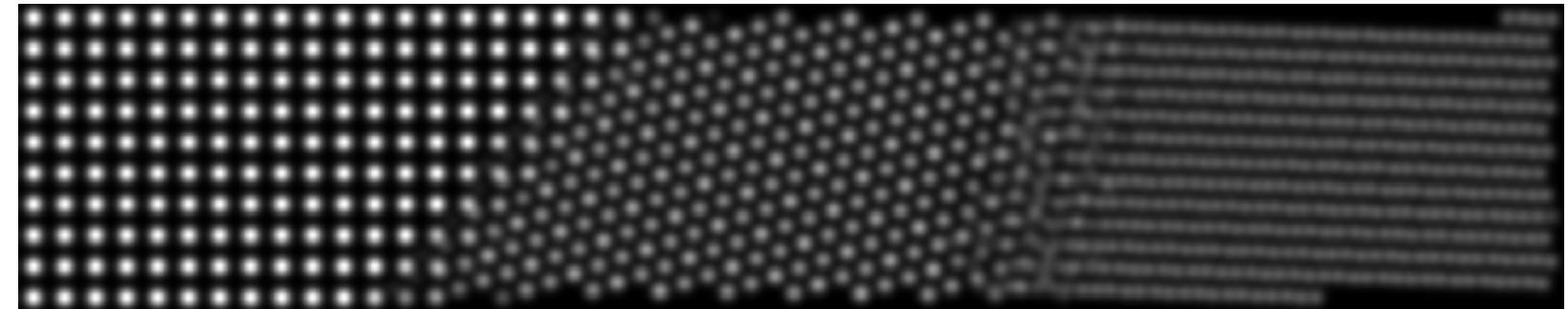
```
abcd12 = [0.2 1 1.2/2 22];
del = p1(:,1) < 1 | p1(:,1) > cellDim(1) ...
| p1(:,2) < 1 | p1(:,2) > cellDim(2) ...
| p1(:,3) < 1 | p1(:,3) > cellDim(3) ...
| p1(:,1)*abcd12(1) + p1(:,2)*abcd12(2) + ...
p1(:,3)*abcd12(3) > abcd12(4) - 0.1;
p1(del,:) = [];
```

Using a Movie to Explain Diffraction Physics



Create atom image by summing along dim 3, blurring:

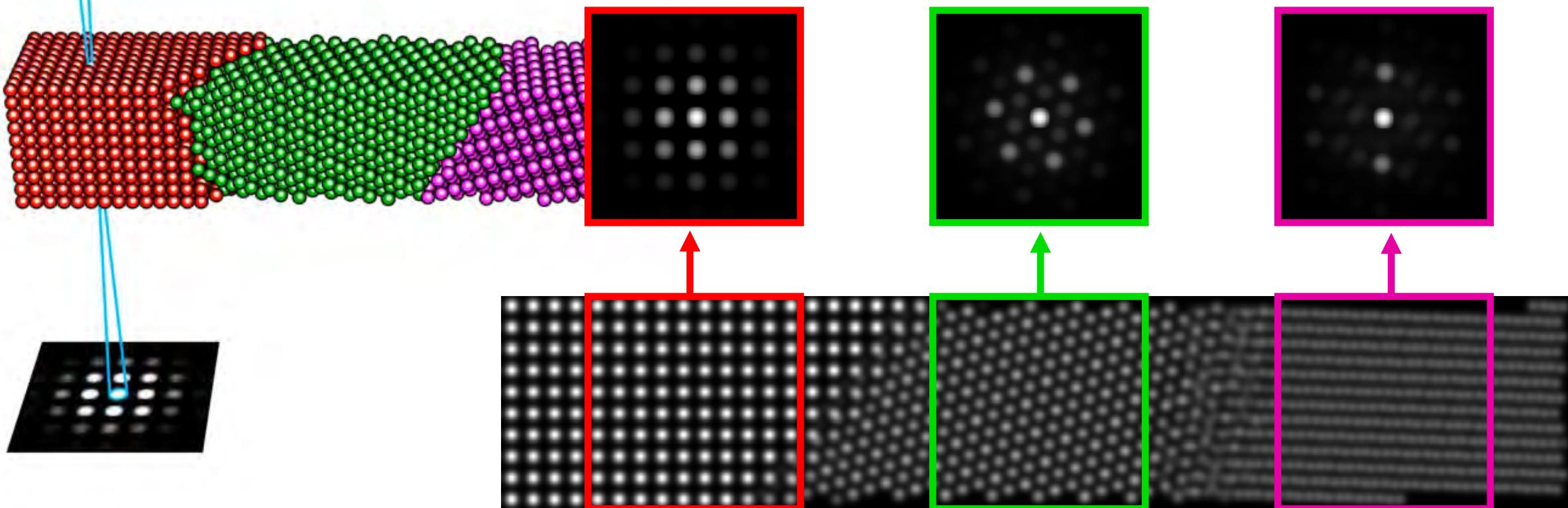
```
k = fspecial('gaussian',2*ceil(4*sigmaAtoms)+1,sigmaAtoms);  
imageSize = round(cellDim(1:2) ./ pixelSize);  
xInd = (p(:,1) - 0.5) / pixelSize;  
yInd = (p(:,2) - 0.5) / pixelSize;  
xInd = min(max(round(xInd),1),imageSize(1));  
yInd = min(max(round(yInd),1),imageSize(2));  
imagePot = accumarray([xInd yInd],...  
    ones(length(xInd),1),imageSize);  
imagePot(:) = conv2(imagePot,k,'same');
```



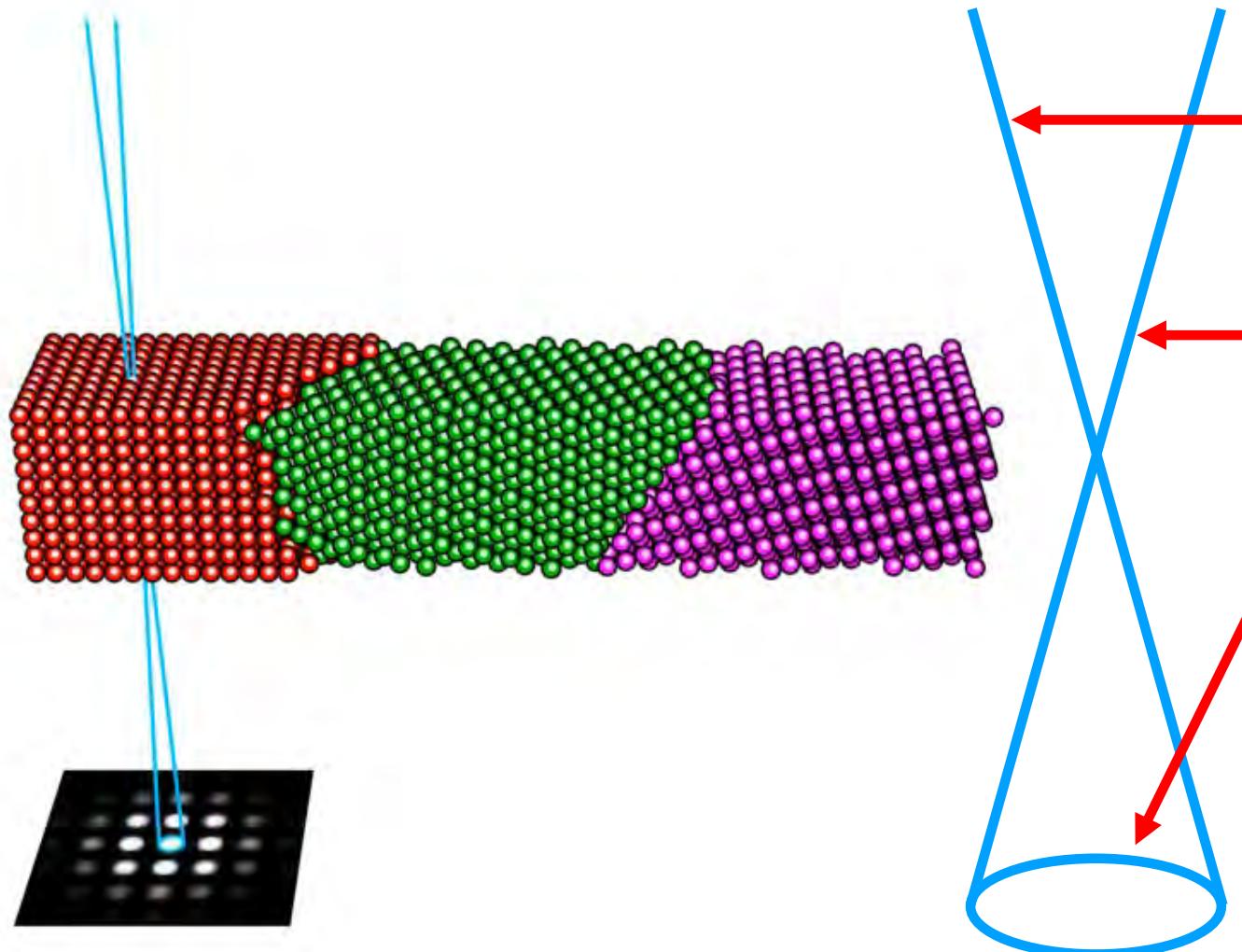
Using a Movie to Explain Diffraction Physics

Generate diffraction images:

```
xInds = mod((1:imageSizeOutput(1)) - imageSizeOutput(1)/2 + probeSites(a0,1)/pixelSize-1, imageSize(1))+1;  
yInds = mod((1:imageSizeOutput(2)) - imageSizeOutput(2)/2 + probeSites(a0,2)/pixelSize-1, imageSize(2))+1;  
imageCut = imagePot(round(xInds),round(yInds));  
imageCut = fft2(repmat(imageCut .* w2,[1 1]*probeRep));
```



Using a Movie to Explain Diffraction Physics



Draw the STEM probe:

```
line([-1 1]*probeRadius+probeSites(a0,2),...  
[0 0]+probeSites(a0,1),...  
probePos([1 3]),...  
'linewidth', linewidthProbes, 'color',[0 0.8 1])  
line([1 -1]*probeRadius+probeSites(a0,2),...  
[0 0]+probeSites(a0,1),...  
probePos([1 3]),...  
'linewidth', linewidthProbes, 'color',[0 0.8 1])  
line(ct*probeRadius+probeSites(a0,2),...  
st*probeRadius+probeSites(a0,1),...  
probePos(3)+ct*0,...  
'linewidth', linewidthProbes, 'color',[0 0.8 1])
```

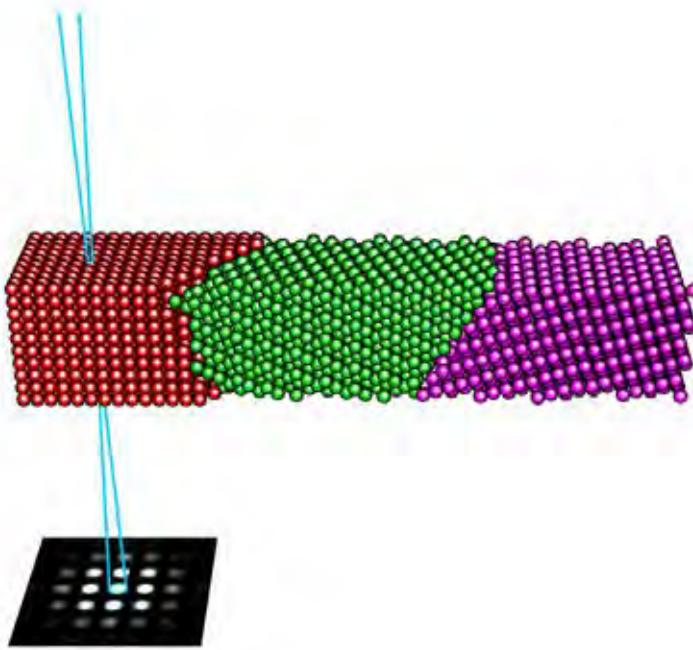
```
t = linspace(0,2*pi,180+1);
```

```
ct = cos(t);
```

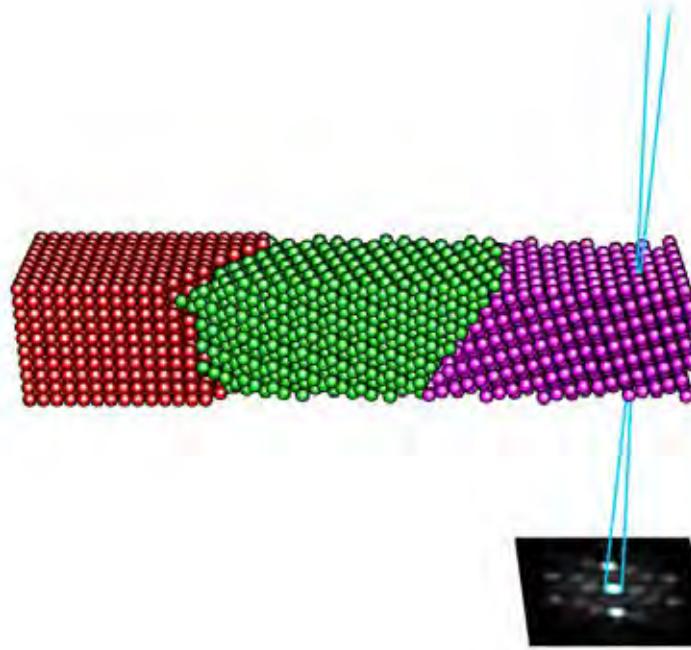
```
st = sin(t);'linewidth', linewidthProbes, 'color',[0 0.8 1])
```

Using a Movie to Explain Diffraction Physics

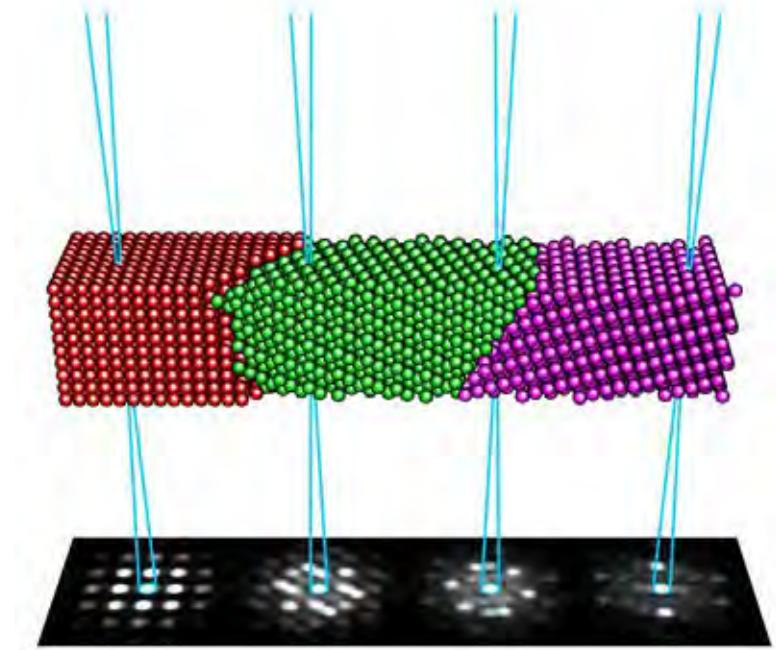
```
>> animOrient01( [5 45] );
```



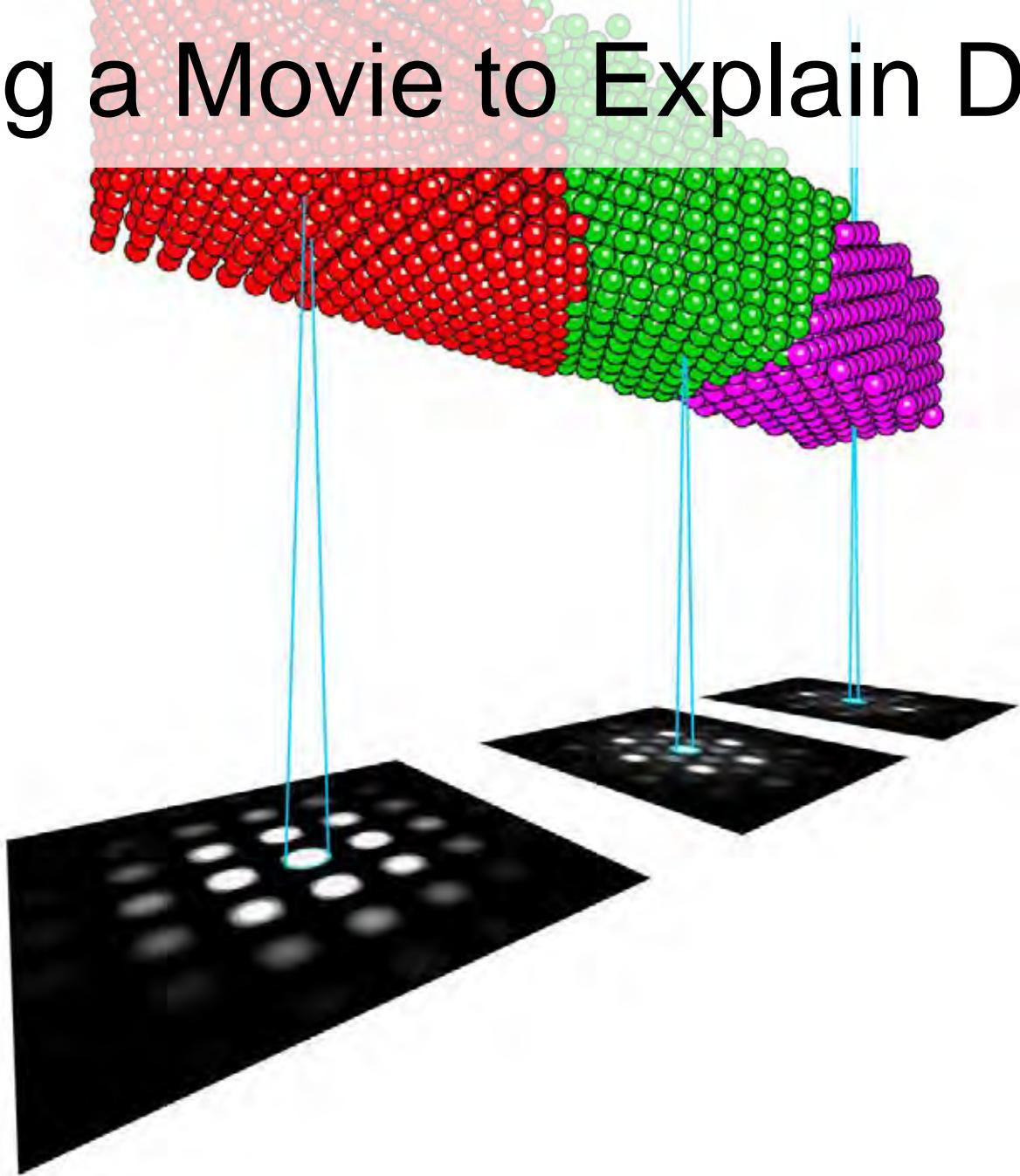
```
>> animOrient01( [5 5] );
```



```
>> animOrient01( [5 4; 5 18;5 32;5 46] );
```



Using a Movie to Explain Diffraction Physics



Just play with the camera, plot, image settings until you are happy with the result!

For example:

```
cTar = [0 0 0] + cellDim * 0.5 + [10 5 -15];  
cPos = [-2.25 1.75 0]*13;  
cAngle = 75;
```

Colin Research Stuff – clophus@lbl.gov

