Lecture 8

Image Processing

Prof. Steven Ludtke
N410.07, sludtke@bcm.edu

Team Learning Review

```
#!/usr/bin/env python
from pylab import *

x=arange(0,3*pi,0.05)
y=cos(x)
plot(x,y)
show()
```

```
#!/usr/bin/env python
from sys import argv

filein=open(argv[1],"r")
lines=filein.readlines()   # reads the entire file as a list of strings
for line in lines:
    print(line,end='')     # end='' suppresses the new line print normally adds
```

Team Learning Review

```
#!/usr/bin/env python
from sys import argv
from pylab import *

filein=open(argv[1],"r")
for line in filein:         # shortcut, iterate directly over the file
    print(line,end='')      # end='' suppresses the new line print normally adds

# We still need x & y

plot(x,y)
show()
```

List Comprehension

- A for loop inside a list definition !
- [x... for x in y]

  example:

  a=[i**2 for i in range(8)]

  print(a)

  [0,1,4,9,16,25,36,49]

  a=[i**2 for i in range(8) if i%3!=0]

  a=[[i*j for j in range(1,10)] for i in range(1,10)]

String -> x,y

- We have the string s="1 5". How to convert to numbers?

```
x=float(s.split()[0])
y=float(s.split()[1])
```

- We could make it more general:

```
a=[]
for f in s.split():
    a.append(float(f))
x=a[0]
y=a[1]
```

- We can make this more compact with List Comprehension

```
a=[float(f) for f in s.split()]
```

Team Learning Review

Approach #1

```
#!/usr/bin/env python
from sys import argv
from pylab import *

filein=open(argv[1],"r")
x=[]
y=[]
for line in filein:
    x.append(float(line.split()[0]))
    y.append(float(line.split()[1]))

plot(x,y)
show()
```

Team Learning Review

Approach #1 ... better if we only do split() once?

```
#!/usr/bin/env python
from sys import argv
from pylab import *

filein=open(argv[1],"r")
x=[]
y=[]
for line in filein:
    s=line.split()
    x.append(float(s[0]))
    y.append(float(s[1]))

plot(x,y)
show()
```

Team Learning Review

Approach #2
(works, but may be harder to read)

```
#!/usr/bin/env python
from sys import argv
from pylab import *

filein=open(argv[1],"r")
xy=[[float(i) for i in ln.split()] for ln in filein]
xy=array(xy).transpose()

plot(xy[0],xy[1])
show()
```

General Image Processing

- PIL/PILLOW (today)
- SciPy 'ndimage' module
  - Apply NumPy capabilities to image processing
- OpenCV
  - Computer vision library with Python bindings
- Mahotas
  - Another computer vision library
- EMAN2 (developed by my group)
  - Greyscale quantitative image processing
  - Links to structural biology (CyroEM)
- GIMP
  - Free Photoshop-like, cross-platform
  - Python based 'modules'

PIL/PILLOW

- Python Imaging Library
- Open-source/commercial
- PIL unofficially dead, PILLOW is the replacement
- Reads-writes many standard formats
- Generic image processing

Installing PIL

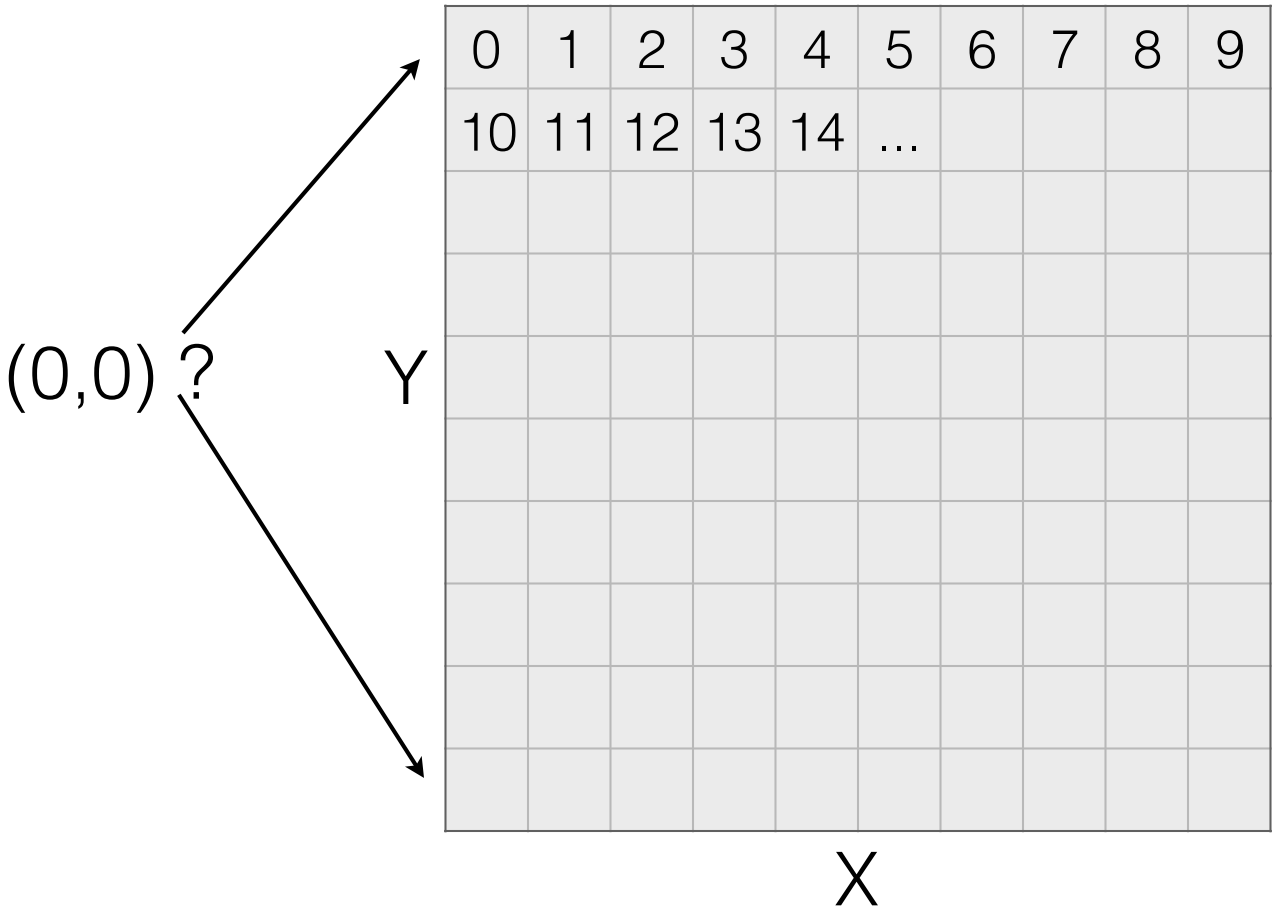
- (SHOULD ALREADY BE INSTALLED WITH ANACONDA)

- https://pypi.python.org/pypi/Pillow
- Windows:
  - Standard installer packages should work
- Linux
  - Should be in package installer (may need to look for python imaging)
- Mac
  - Make sure you have Xcode & command-line tools installed
  - Install libjpeg (http://www.ijg.org/)
    - http://snippets.dzone.com/posts/show/38
  - sudo easy_install pillow
  - If this doesn't work, you may want to try the instructions on either the pillow or the pil website.

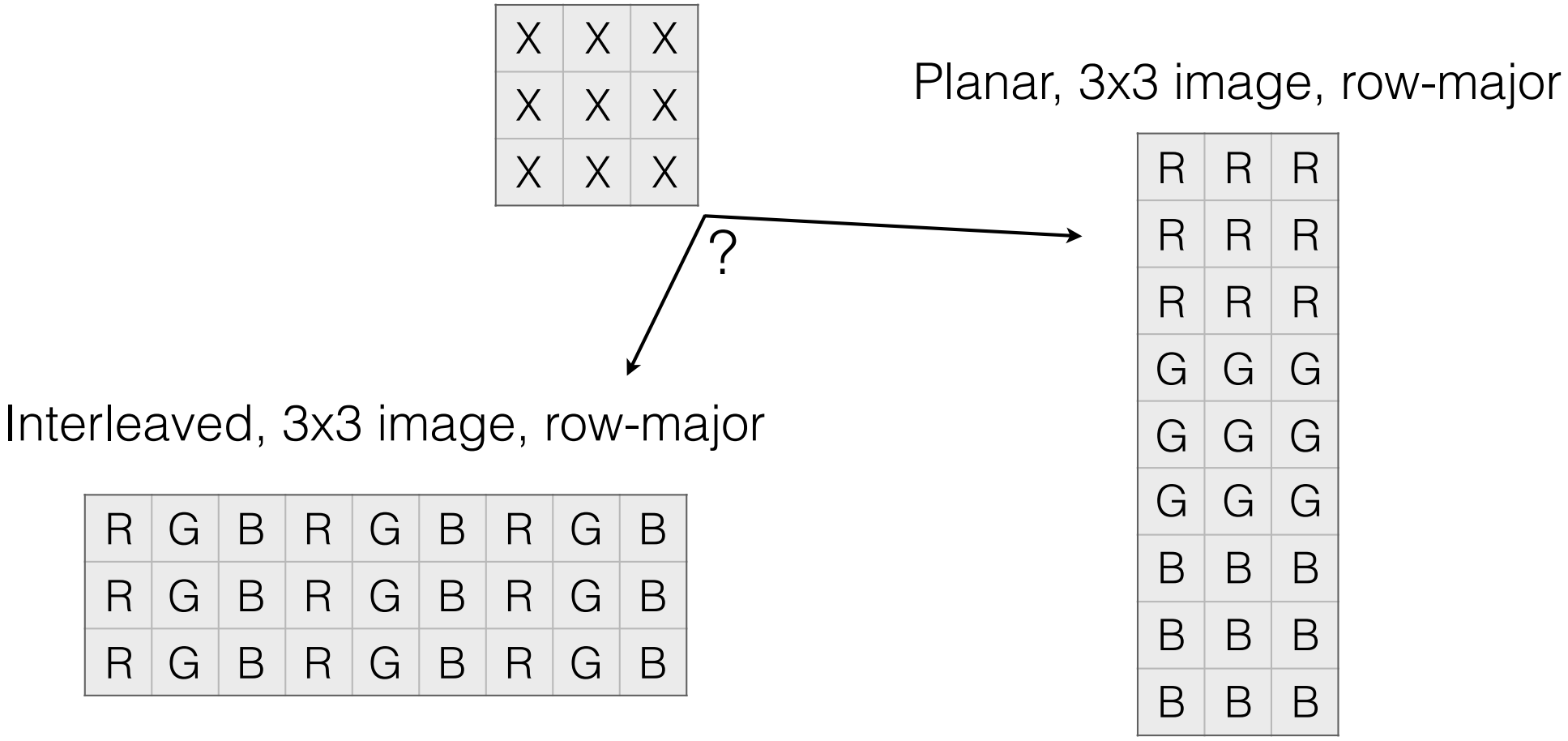
- to test: 'import PIL'

Images

Pixel stored at location x+nx*y (row major)
or y+ny*x (column major, less common)



Color Images



PIL - File Formats*

Fmt	Bits	Loss	Cmpr	Notes
BMP	8 or less			
GIF	8 total, cmap	X	X	
IM	all modes !			LabEye & IFUNC
JPEG	8	X	X	
PCX	8 or less			
PNG	8		X	
PPM	8			PBM,PGM,PPM
TIFF	8	R	R	
EPS				Needs GS to read most
PDF				Write only

* - Only the most common ones

Image Modes

- 1 (1-bit pixels, black and white, stored with one pixel per byte)
- L (8-bit pixels, black and white)
- P (8-bit pixels, mapped to any other mode using a colour palette)
- RGB (3x8-bit pixels, true colour)
- RGBA (4x8-bit pixels, true colour with transparency mask)
- RGBX (3x8-bit pixels, true colour with padding byte)
- CMYK (4x8-bit pixels, colour separation)
- YCbCr (3x8-bit pixels, colour video format)
- I (32-bit signed integer pixels
- F (32-bit floating point pixels)

PIL

```
from PIL import Image
im=Image.open("4012.jpg")
data=b"\0"*(128*128*4)       # 'bytes' of zero pixels
from array import array      # note this is NOT numpy
data=array("b",data)         # convert to an array object
im=Image.frombuffer("RGBX",(128,128),data,"raw","RGBX",0,1)
im.show()                    # machine specific display
pix=im.load()                # for pixel access
pix[x,y]                     # access pixel at x,y
im.save(filename,[format],[options])
```

Using Numpy

```
from numpy import *

from PIL import Image

a=fromfunction(lambda x,y:sin(x/10.0)*cos(y/10.0),(128,128))

im=Image.fromarray(a)

im.show()                # Image is black !?!?

a+=1

a*=127

im2=Image.fromarray(a)

im2.show()
```

PIL

```
im=Image.open("test.jpg")

a=array(im)

a[0,0]

im2=Image.fromarray(a)
```

PIL

```
a=fromfunction(lambda x,y:(127+sin(x/100.)*127., 127+cos(y/
100.)*127.,127+sin(x/500.)*127.),(256,256))

b=dstack(a)

c=b.astype(uint8)

im=Image.fromarray(c)

im.show()
```

PIL

- ImageChops - invert(a), lighter(a,b), darker(a,b), add(a,b), subtract(a,b), difference(a,b), screen(a,b)

- ImageDraw

```
from PIL import Image,ImageDraw

a=Image.new("RGBA",(128,128))

draw=ImageDraw.Draw(a)

draw.line((x0,y0,x1,y1),fill="red") # point, rectangle, arc, chord, ellipse, text
```

- ImageFilter - BLUR, CONTOUR, DETAIL, EDGE_ENHANCE, EDGE_ENHANCE_MORE, EMBOSS, FIND_EDGES, SMOOTH, SMOOTH_MORE, SHARPEN

- etc.

PIL Attributes

- im.format
- im.mode
- im.size
- im.info

Images in MatPlotLib

```
from pylab import *

im=fromfunction(lambda x,y:sin(x/10.0)*cos(y/10.0),(64,64))

imshow(im)

imshow(im,cmap=cm.gray)

savefig("a.png")
```

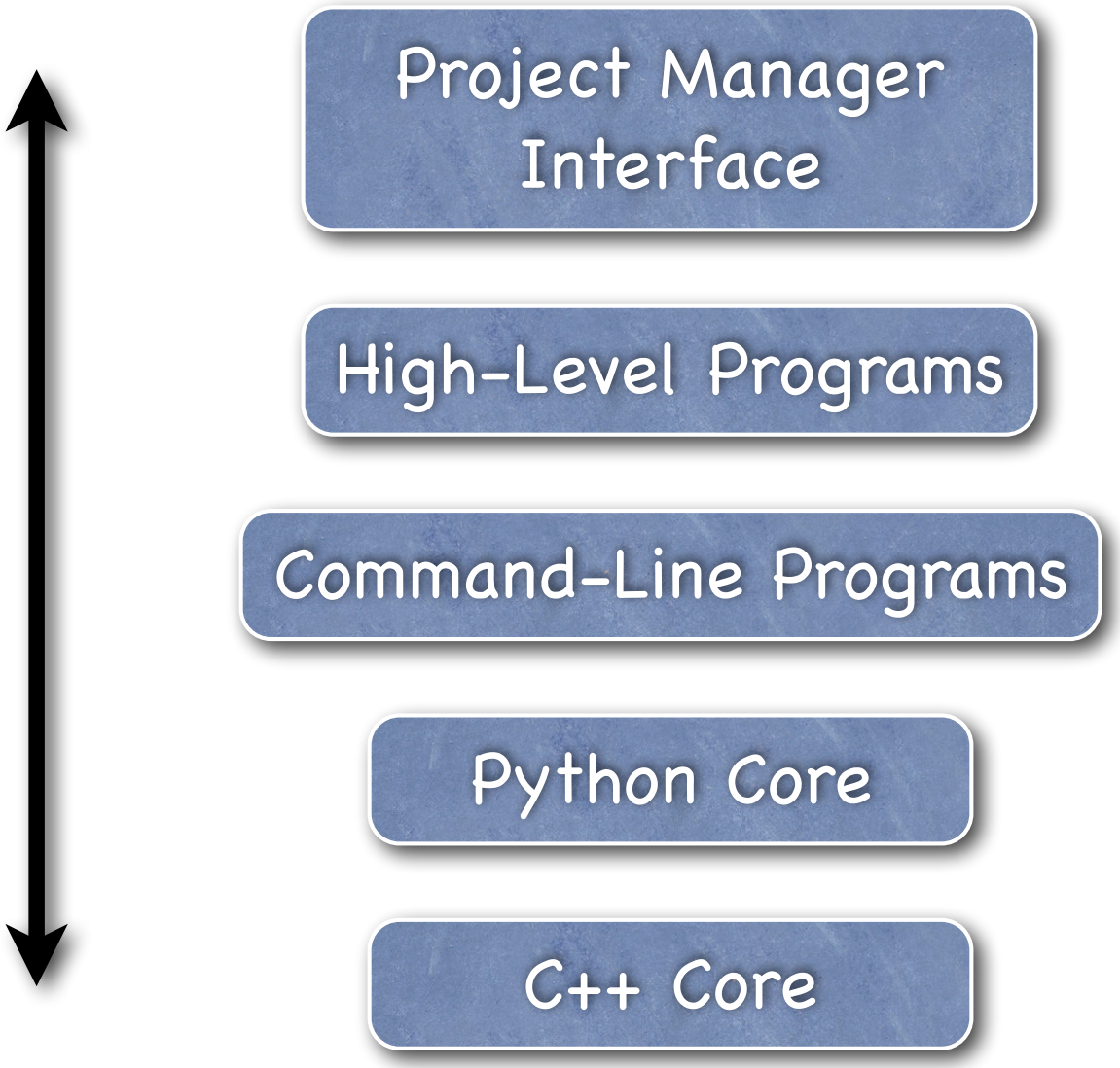
EMAN2

- EMAN2 Wiki:
  - http://eman2.org
- Discussion Mailing List/Google Group:
  - https://groups.google.com/forum/?fromgroups#!forum/eman2

EMAN2 Features

- Complete graphical workflow, all steps of single particle refinement
- Project system which organizes data and records all reconstruction info.
- Qt/OpenGL for 2d & 3d display.
- Support for all documented cryoEM file formats
- Over 200 general purpose image processing algorithms
- Interoperability features with Frealign, Relion, ResMap…
- Tilt Validation, Random Conical Tilt, Single Particle Tomography
- Parallel processing using MPI and/or Threads

EMAN2 Architecture

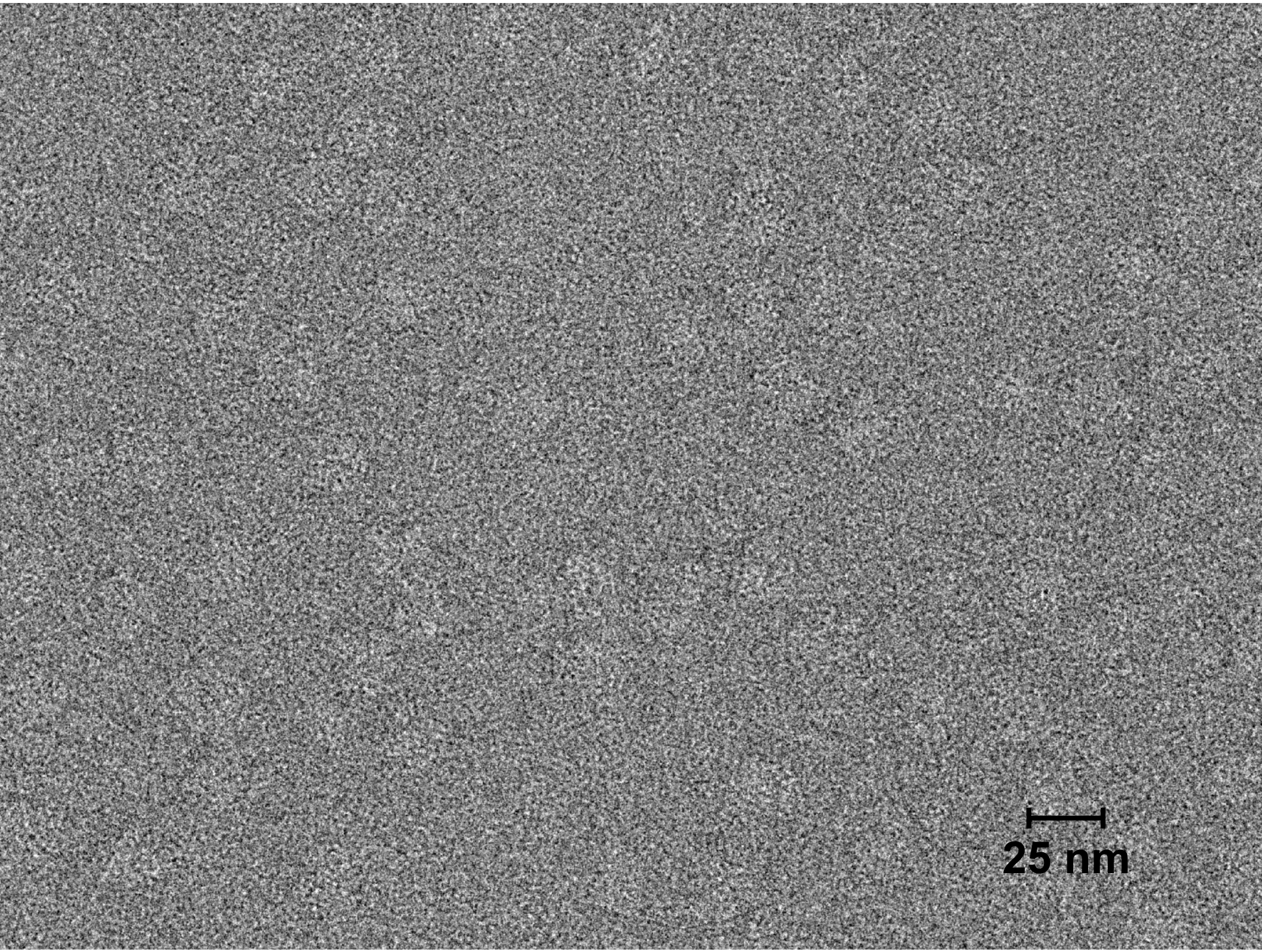


Extensible Core

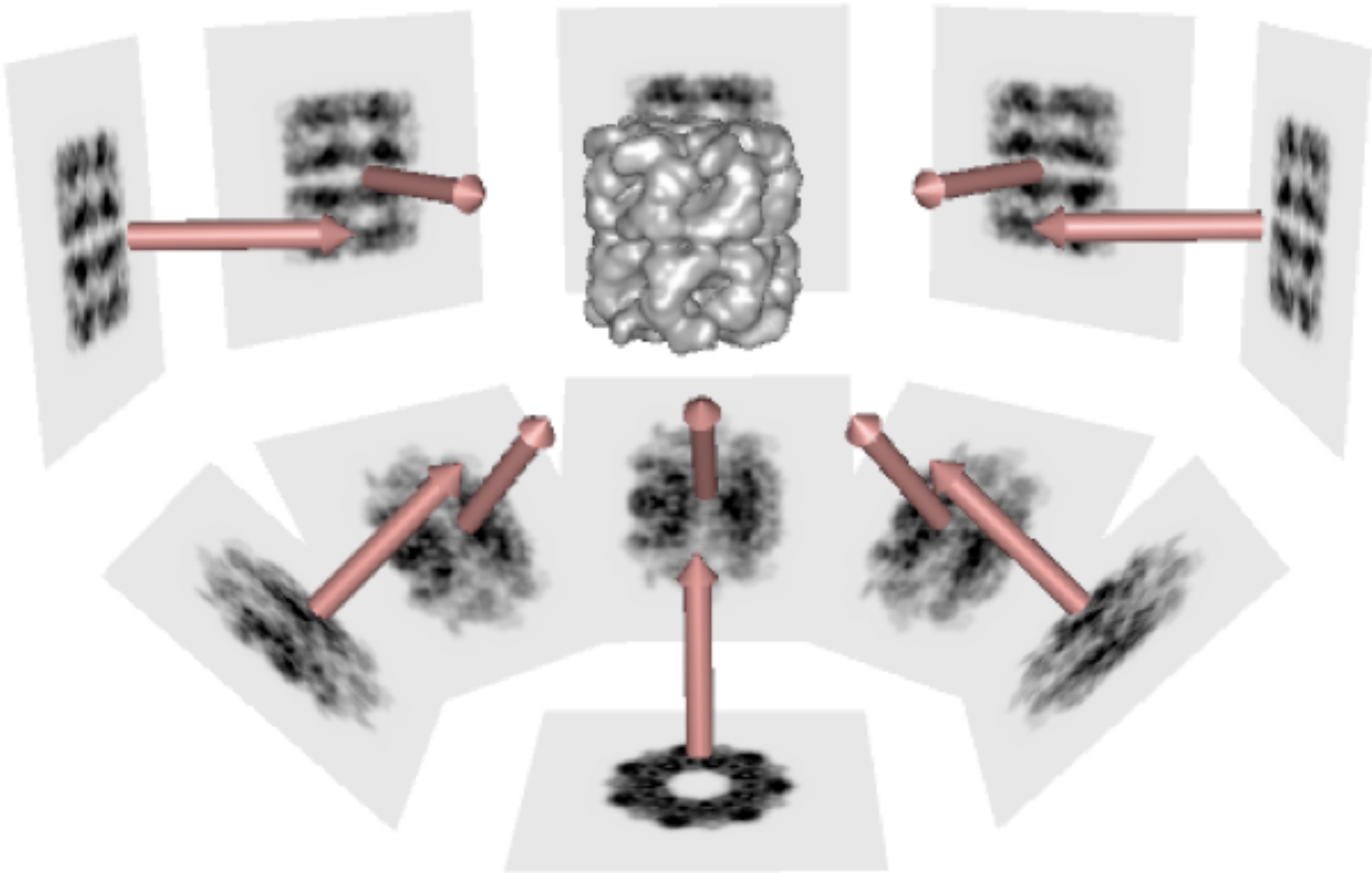
Type	Description	#
Processor	Generic image processing algorithms, filters, masks, thresholds, etc.	220
Aligner	Algorithms used to align 2 images or volumes to each other	32
Projector	Routines to generate 2-D projections of 3-D objects	7
Reconstructor	Routines to reconstruct 3-D objects from 2-D projections	13
Cmp	Similarity metrics used to compare two images or volumes	15
Averager	Average together stacks of images in various ways	12
Analyzer	Perform various operations on sets of images, such as classification or PCA	9
Orientgen	Routines describing how projections cover the asymmetric triangle	7

File Formats

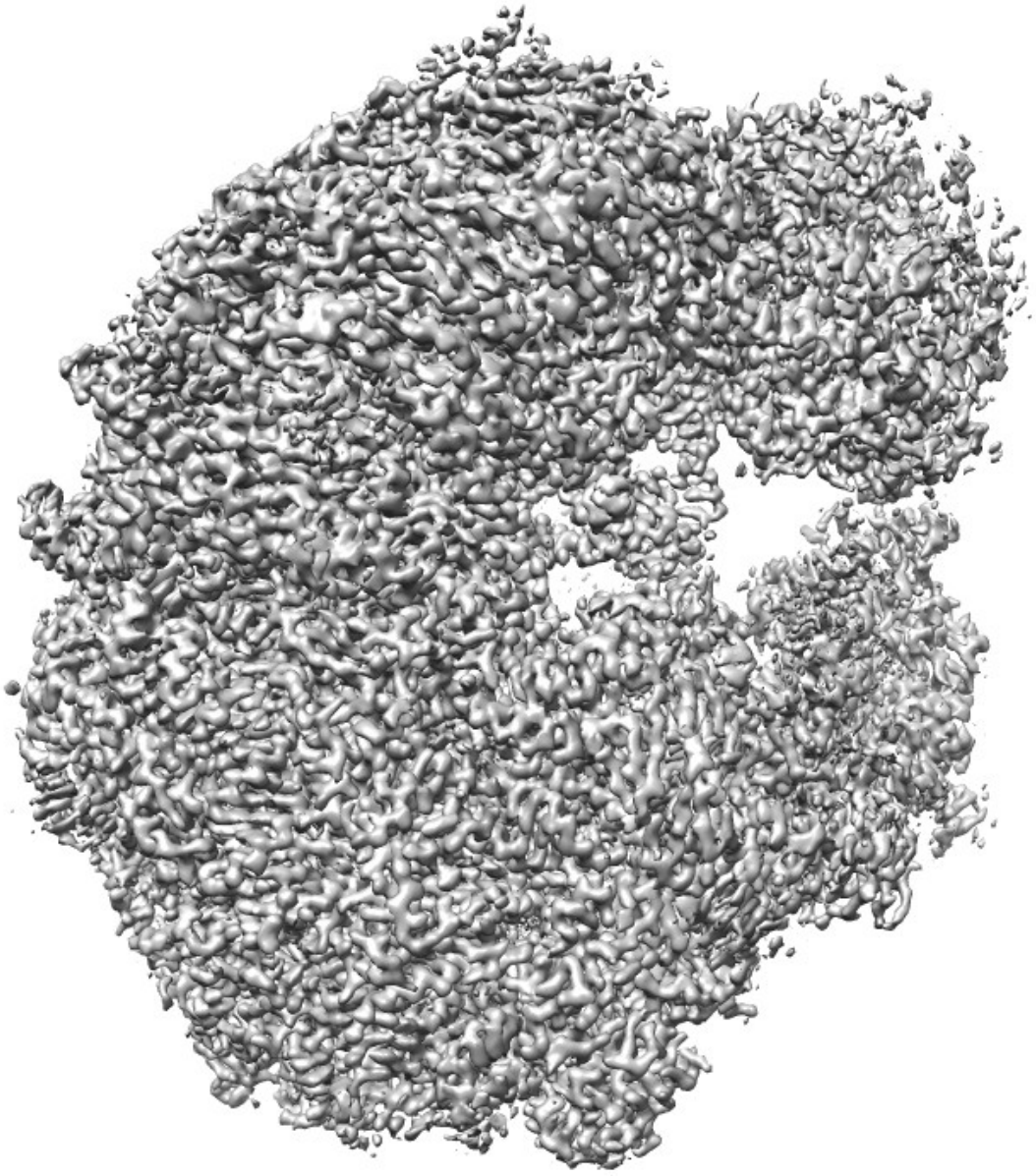
MRC	R/W	IMAGIC	R/W
SPIDER	R/W	HDF5	R/W
PIF	R/W	ICOS	R/W
VTK	R/W	PGM	R/W
Amira	R/W	Xplor	W
Gatan DM2	R	Gatan DM3	R
Gatan DM4	R	FEI SER	R
TIFF	R/W	Scans-a-lot	R
LST	R/W	PNG	R/W
Video-4-Linux	R	JPEG	W


25 nm

TEM Produces Projections



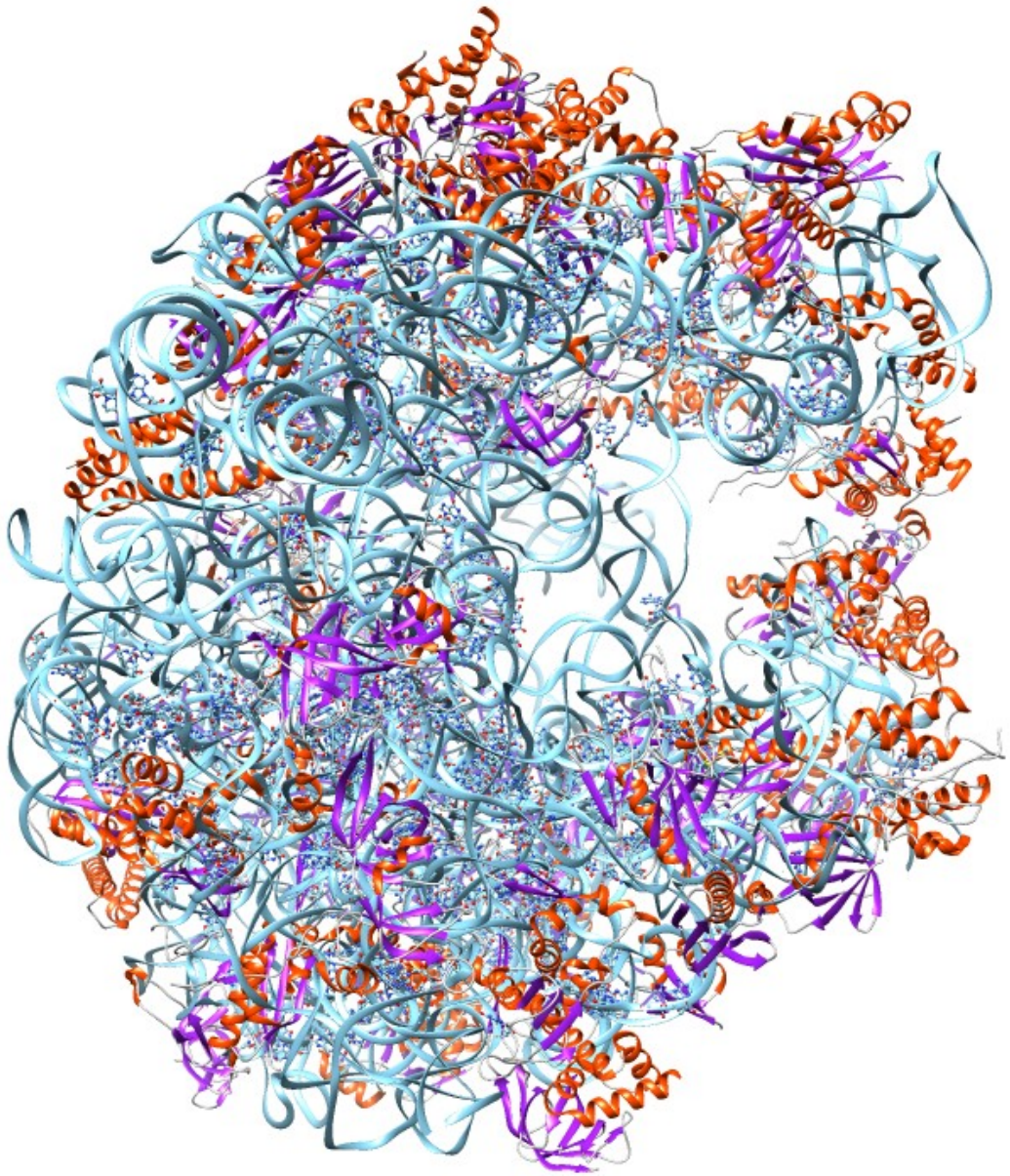
70S Ribosome



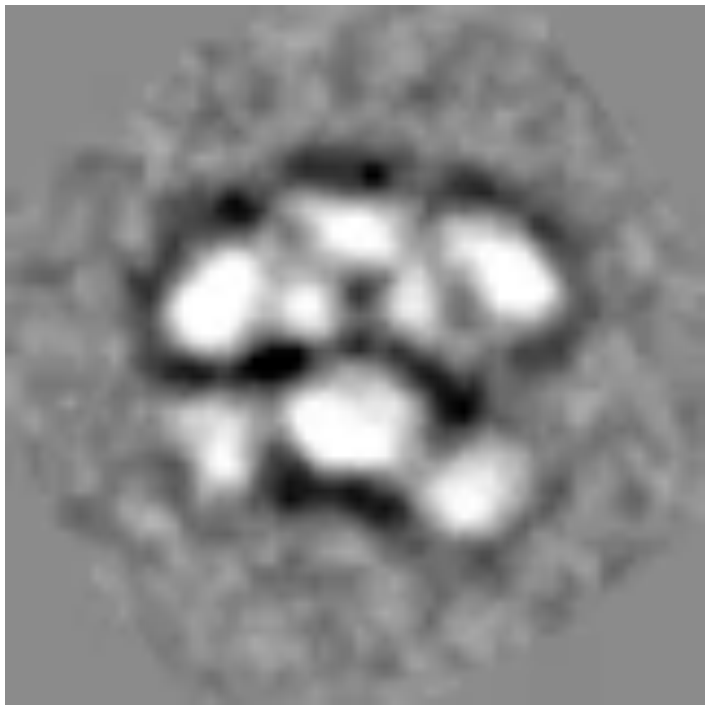
5 nm

N.R. James, et.al. (2016) *Science.* **354,** 1437-1440

70S Ribosome



Fatty Acid Synthase, ~30 Å motion



15 nm

Homework 8

- Write a program that finds all of the .jpg files in the current directory, reads each one, performs some sort of image processing with PIL on each, and writes each back to disk with "_proc" in its name, ie - image.jpg -> image_proc.jpg

- tip - you may find the os library useful