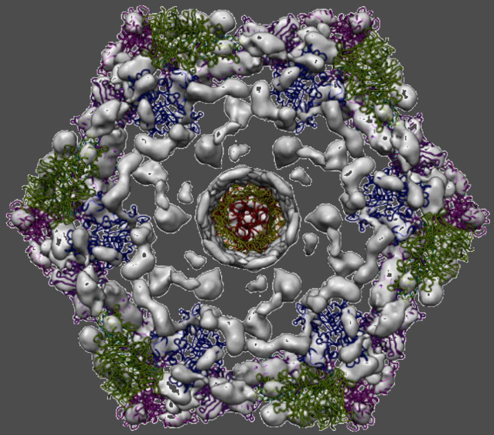
Extending UCSF Chimera With Python Code

Conrad Huang

June 2010

OVERVIEW

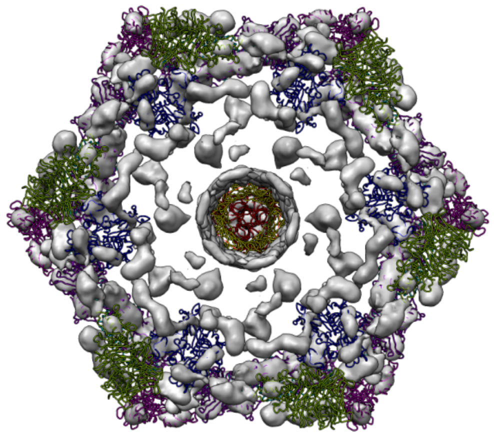
- Molecular visualization and UCSF Chimera
- Preparing Chimera input files with Python
  - Display atom property
- Programming Chimera with Python
  - Generate view of ligand for multiple PDB files
  - Display dipole moment
  - Focus in on active site
  - Display Ramachandran plot
- Wrap Up



MOLECULAR VISUALIZATION AND UCSF CHIMERA

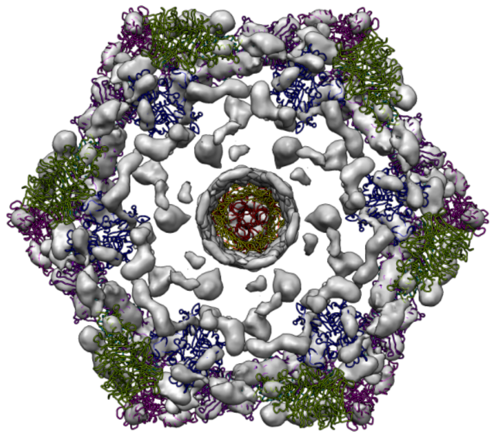
What can we visualize using Chimera?

Molecular Visualization



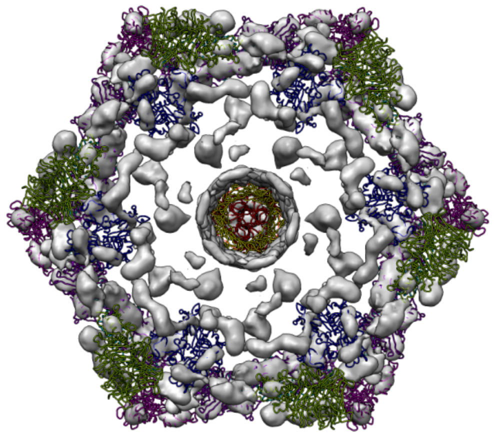
- Multiple data types in multiple file formats
  - Atomic coordinates (pdb, mol, mol2)
  - Molecular sequences (fasta, pir, msf)
  - Electron density maps (brix, ccp4, xplor)
  - Energy grids (phi, profec)
  - Microscopy data (hdf, spi)
  - Generic graphics (vrml, bild)
- Multiple representations
  - Wireframe, ball-and-stick, spheres
  - Surfaces, volumes
- Simple and complex operations
  - Menu and command line interfaces
  - Scripts and programs (cmd, py)

Visualization Uses



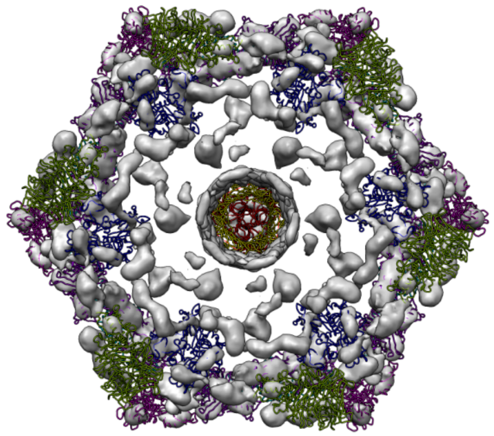
- Analysis
  - Structure properties, eg hydrogen bonding
  - Structure comparisons
  - Sequence-structure relationships
  - Effects of structure modification
- Interface to programs
  - Run bundled programs, eg compute SAXS profile
  - Prepare input for external programs, eg AMBER
- Presentation
  - Images for publications or talks
  - Animations

Customized Visualization

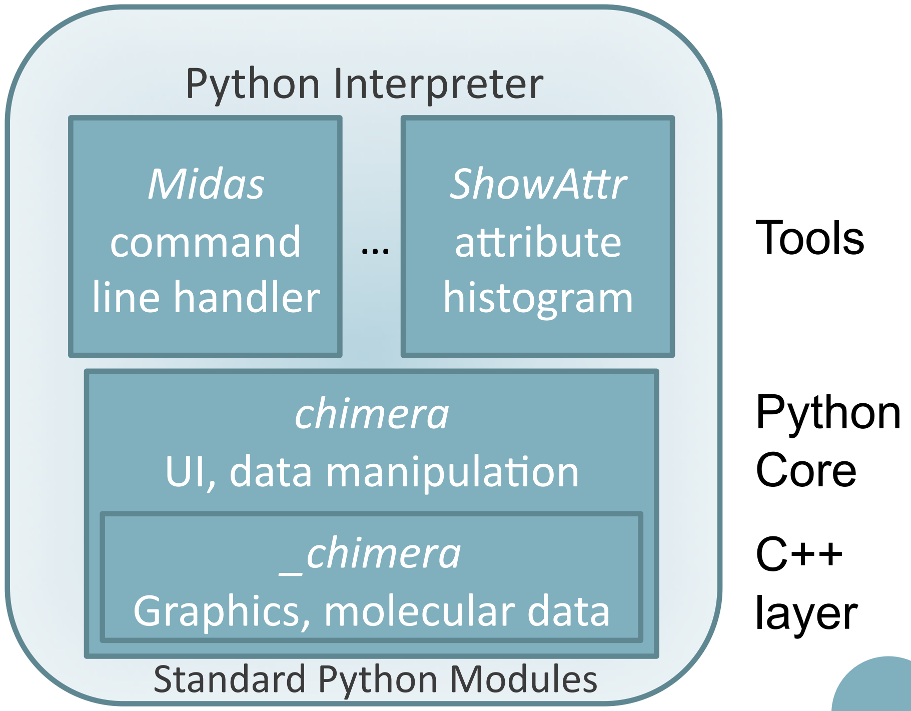


- What if your visualization program does not do what you need? For example, you use Chimera and you want a Ramachandran plot.
  - Switch program – use PyMOL
    - Another program to learn
    - Different functionality
  - Write your own
    - Limited features
    - Too much work
  - Extend existing program – write a Chimera script
    - Leverage existing functionality such as reading multiple input formats, display multiple representations, etc.
    - Hopefully, very easy to do

Chimera Software Architecture

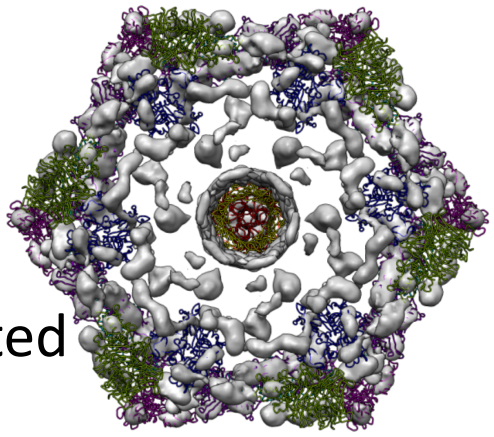


- Extensibility is a Chimera design goal
- Most of Chimera is implemented in Python, with performance-critical sections written in C++
- All data are accessible from Python
  - C++ portion designed to be accessible from Python and conforms to "Pythonic" conventions


Python Interpreter
Midas command line handler
…
ShowAttr attribute histogram
chimera UI, data manipulation
_chimera Graphics, molecular data
Standard Python Modules
Tools
Python Core
C++ layer

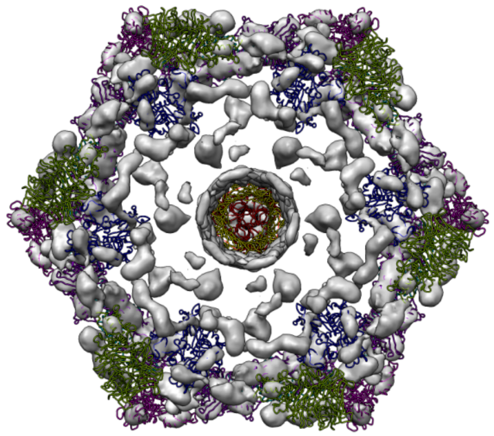
Chimera Architecture

PROGRAMMING PARADIGM



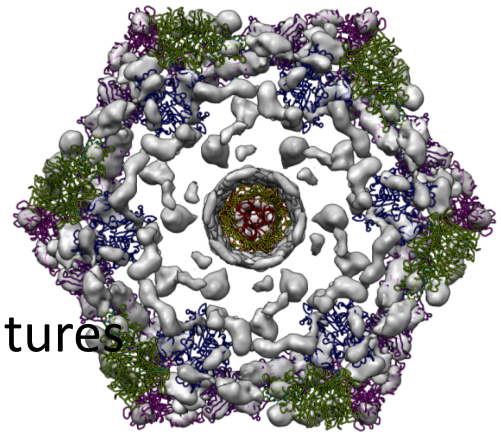
- Chimera is implemented using object-oriented programming methods
- Most data are accessible as instances of classes, eg Molecule, Atom, Vector
  - A class represents a "real world" concept, eg vector
  - Instance attributes store "raw" data or other objects, eg vector x, y and z coordinates
  - Instance methods provide functionality to manipulate one or more attributes in a consistent manner, eg vector normalization
- Python code manipulates data by either calling instance methods or, if the operation is really simple, accessing attributes directly

WRITING CODE



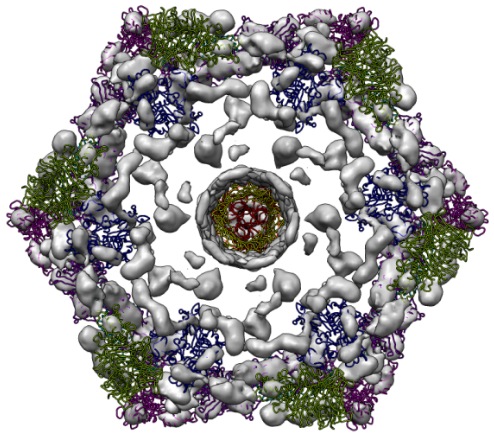
- Interface vs implementation
  - A class instance may be accessed by an external caller (a function or object unrelated to the class) or by an internal caller (an instance of the same class or subclass)
  - Interface is the set of attributes and methods that may be used by **external** caller (and therefore should not change incompatibly)
  - Implementation is the set of data and code that are not revealed to external callers (and therefore may be changed at will)
- Python programming conventions
  - Attributes and methods whose names start with underscore (_) are not part of the class interface
  - Python does not enforce this rule, so external callers are free to access "private" attributes and methods
  - If you do so, you deserve what you get

Writing Code

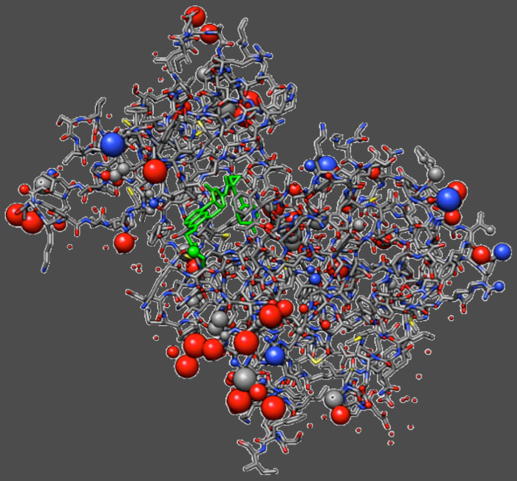


- Python object-oriented programming (OOP) features
  - Inheritance
    - Let a subclass (derived class) automatically reuse attributes and methods defined in a superclass (base class)
    - Simplifies writing classes with similar implementation by reusing code
  - Polymorphism
    - Allows uniform management of instances of different classes that share the same interface
- Chimera uses all these OOP features
  - Very few attributes or methods are private, but that is more a shortcoming of the implementation than by design
  - Inheritance is used, eg **Selection** is the base class for **ItemizedSelection**, **OSLSelection** and **CodeSelection**
  - Polymorphism is also used, eg **openModels** manages all open models, including molecules, surfaces, and graphics objects

EXTENDING CHIMERA



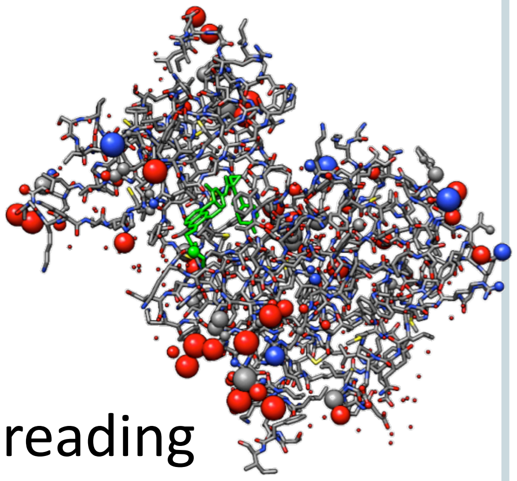
- Chimera supports both a command language for end users and Python for developers
- Chimera may be scripted using just the command language
  - Documentation is extensive, both user's guide and tutorials
- Extending Chimera with Python code requires understanding (and exploring) available modules
  - Python code can execute Chimera commands to simplify accessing functionality
  - Documentation is sparse, but we answer questions on chimera-dev@cgl.ucsf.edu very diligently
  - NOTE: Chimera is an application, not a package
    - Chimera is the main program that runs your Python scripts (rather than your Python programs calling Chimera functions)
    - Chimera is distributed with its own Python to avoid incompatibility between Chimera modules and installed Python



Preparing Chimera Input Files

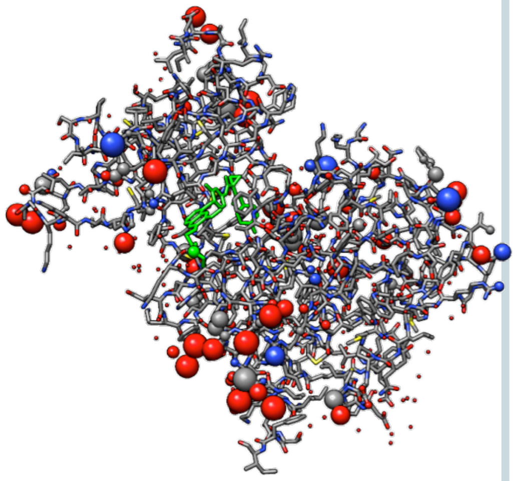
Using Python to feed Chimera

Using Python To Prepare Chimera Input Data



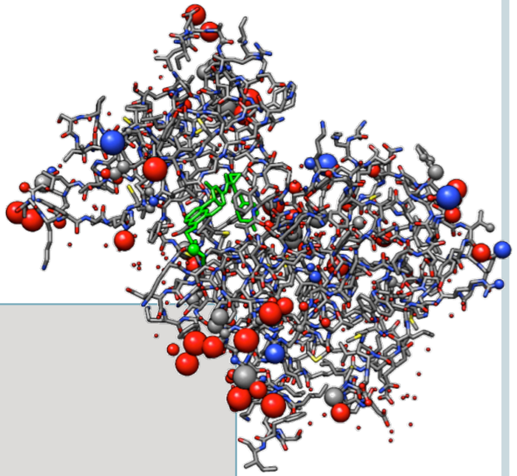
- Chimera has facilities for adding attributes to instances (atoms, residues, molecules) by reading values from a file
  - Menu: **Tools → Structure Analysis → Define Attribute**
  - Command line: **defattr**
- This may be the path of least resistance if the input data can be easily transformed

Display Molecular Annotations



- How can we display molecular annotations that are computed and stored separately from atomic coordinates files such as PDB?
- Example: How can we display crystal contact information available from OCA for PDB files?
  - Data is available as a fixed-format text file
  - Columns include residue type, residue sequence, chain identifier, atom name, crystal and PDB surface area
  - We want to show surface area differential
- Plan of attack
  - Parse input OCA crystal contact file
  - Construct atom specifier for each value
  - Generate Chimera input file for Define Attribute tool
  - Display property using **Render by Attribute** tool

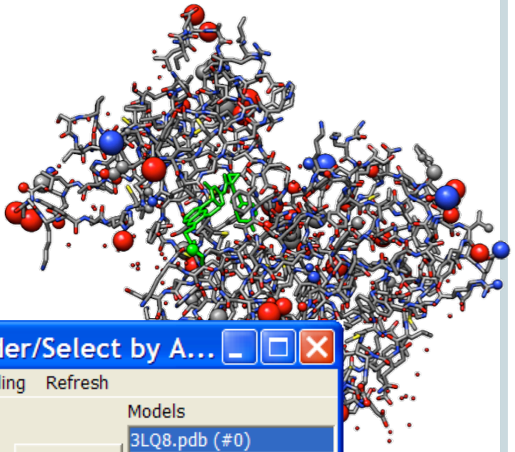
Converting Data Formats

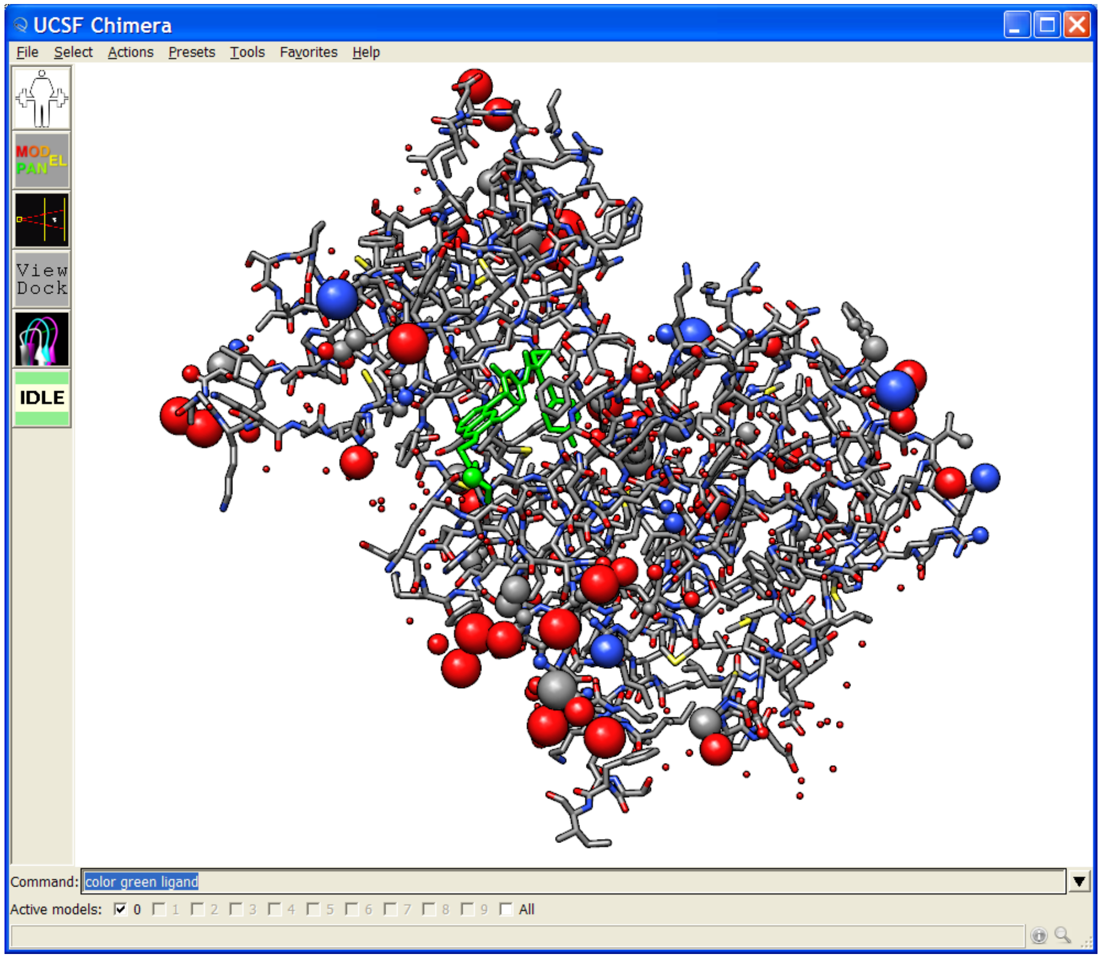


```
List of all atomic contacts for PDB entry 3LQ8
                                             […]
 SER 1056   A CA   10.3  11.4 SER 1056   A N    1.5 44.9 7 3
 SER 1056   A CA   10.3  11.4 SER 1056   A C    1.5 43.5 7 6
 SER 1056   A CA   10.3  11.4 SER 1056   A CB   1.5 31.0 7 6
 SER 1056   A CA   10.3  11.4 SER 1056   A O    2.4  1.6 7 2
 SER 1056   A CA   10.3  11.4 SER 1056   A OG   2.4  0.7 7 1
 SER 1056   A CA   10.3  11.4 LEU 1055   A C    2.5  3.4 7 6
 SER 1056   A CA   10.3  11.4 LEU 1055   A O    2.9  0.2 7 2
 SER 1056   A CA   10.3  11.4 ASN  712   X ND2  4.3  1.1 7 3    A1081   8
 SER 1056   A CA   10.3  11.4 LEU 1058   A O    5.0  0.2 7 2
 SER 1056   A C     0.7   0.9 SER 1056   A O    1.2 35.0 6 2
 SER 1056   A C     0.7   0.9 ALA 1057   A N    1.3 43.7 6 3
 SER 1056   A C     0.7   0.9 SER 1056   A CA   1.5 41.1 6 7
                                             […]
```

```
attribute: csu
match mode: non-zero
recipient: atoms
      :1056.A@CA      1.1
      :1056.A@C       0.2
                                             […]
```

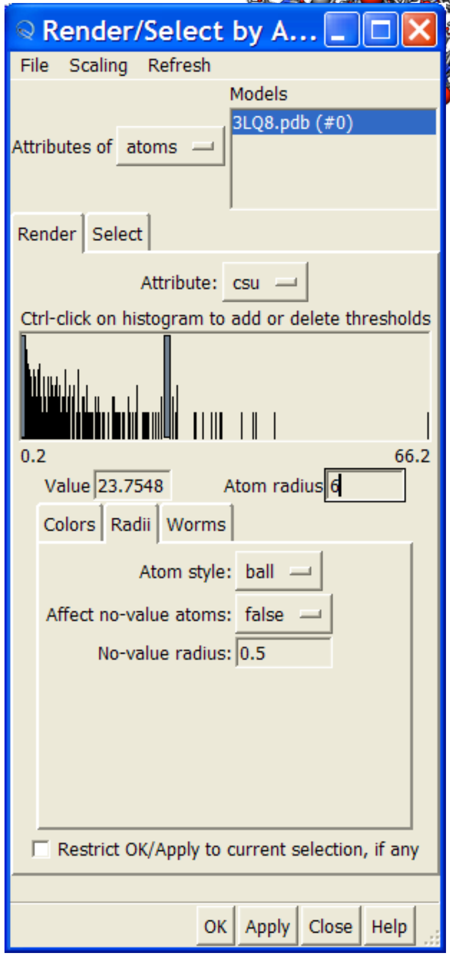
Importing Attributes





Command: color green ligand

Active models: ☑ 0



Models: 3LQ8.pdb (#0)

Attributes of: atoms

Attribute: csu

Ctrl-click on histogram to add or delete thresholds

0.2 66.2

Value 23.7548 Atom radius 6

Atom style: ball

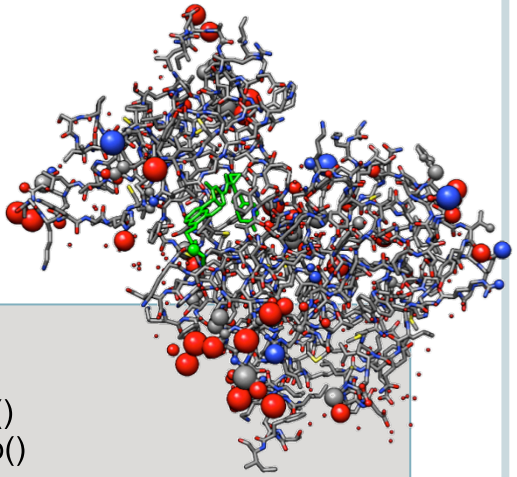
Affect no-value atoms: false

No-value radius: 0.5

☐ Restrict OK/Apply to current selection, if any

Crystal contact data for **PDB structure 3LQ8**

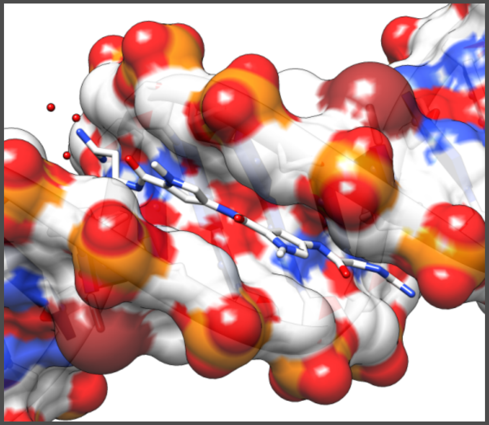
CONVERSION CODE



```
def convert(inFile, outFile, verbose):
        # Generate output file and header
        out = open(outFile, "w")
        print >> out, "attribute: csu"
        print >> out, "match mode: non-zero"
        print >> out, "recipient: atoms"
        f = open(inFile)
        f.readline()                    # Skip header
        seen = {}
```

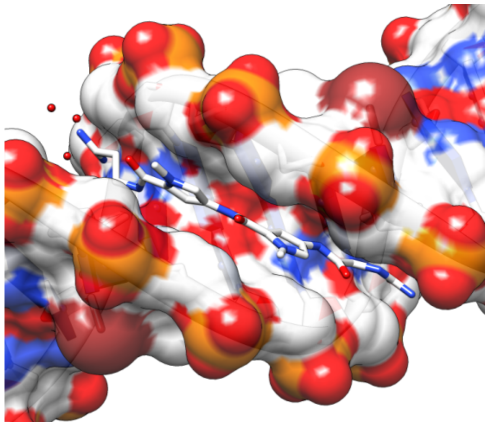
```
        for line in f:
                rSeq = line[5:9].strip()
                chainId = line[12].strip()
                atom = line[14:18].strip()
                if not chainId:
                        sel = ":%s@%s" % (rSeq, atom)
                else:
                        sel = ":%s.%s@%s" % (rSeq, chainId, atom)
                if sel in seen:
                        continue
                crystalSAS = float(line[18:22])
                pdbSAS = float(line[24:29])
                delta = pdbSAS - crystalSAS
                seen[sel] = delta
                if delta != 0:
                        print >> out, "\t%s\t%.3g" % (sel, delta)
        f.close()
        out.close()
```

csu2attr.py – convert OCA crystal contacts to Chimera attributes



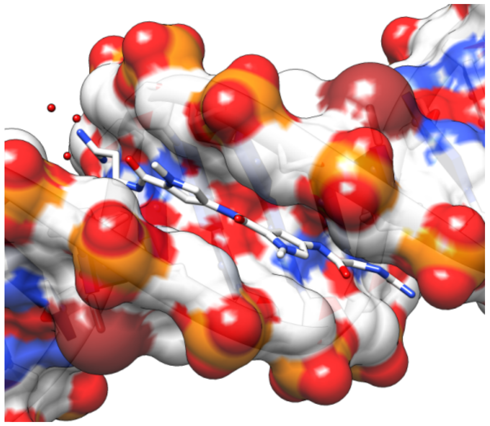
Programming Chimera

PROGRAMMING CHIMERA



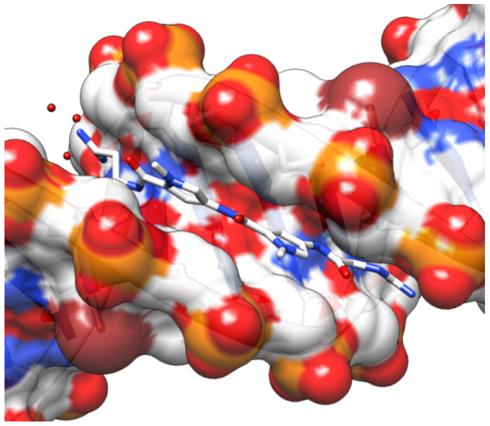
- Write a Chimera script
  - Only use Chimera command language
  - Good for simple, linear tasks (no control flow)
- Write a Python script
  - Use Python for most flexibility
  - Can execute Chimera commands
  - Good for tasks not requiring extensive user interactions
- Write a Chimera extension
  - Implemented in Python
  - Provides user interface (whether command line or graphical)
  - Good for tasks requiring user interactions or maintaining state across interactions

PROCESSING MULTIPLE DATA FILES



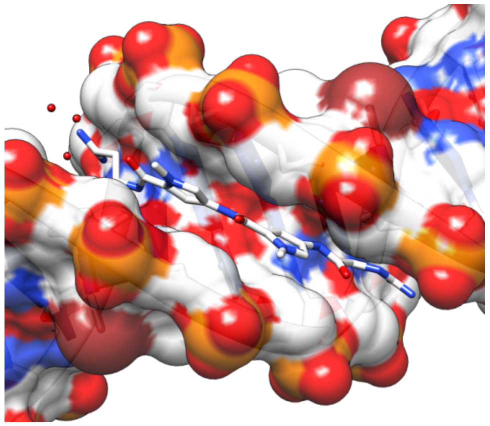
- How can we generate a view of bound ligands for all the PDB files in a folder?
- Plan of attack
  - Loop over all files with .pdb extension in the target folder
  - For each file, execute Chimera commands to:
    - open the file in Chimera using **open** command
    - orient the ligand using **align** command
    - update the view to place ligand in the center with **focus** command
    - select publication quality display settings with **preset** command
    - display surface for receptor with **surface** command
    - make surface translucent with **surftransp** command
    - generate image with **copy** command
    - close session in preparation for next model with **close** command

PROCESSING MULTIPLE DATA FILES



- Hints
  - To select a directory from Chimera, use:
    - **from OpenSave import OpenModal**
      from chimera.tkgui import app
      dirList = OpenModal(dirsOnly=True).run(app)
    - **dirList** is a list of 2-tuples (directory path, type); or **None** if user hit *Cancel*
  - To run a Chimera command from Python, use:
    - **chimera.runCommand(***command_string***)**
  - To display a status messages, use:
    - **chimera.replyobj.status(***message***)**
- To execute a Python script in Chimera, just open the .py file using **File** → **Open**

SOLUTION SCAFFOLD

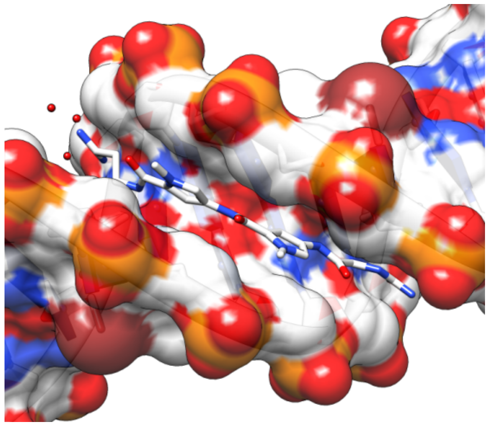


```
import os
curdir = os.getcwd()      # save current folder

# Import Chimera modules and functions
# Ask user to select folder
# Loop over folder list, ignoring type parameter
  # Assume current folder path is “dirPath”
  os.chdir(dirPath)         # set current folder
  # Get list of file names ending in .pdb extensions
  # (os.listdir lists all files in a folder)
```

```
open 6BNA.pdb
align ligand ~ligand
focus ligand
surf
preset apply publication 1
surftransp 15
copy file 6BNA.png supersample 3
close all
```

```
  # Loop through file names
    # Assume file name is “fn”
    status("Processing " + fn)
    # Open file in Chimera
    chimera.runCommand("align ligand ~ligand")
    # Line above positions ligand in front of
    # other parts of the molecule
    # Run more Chimera commands to set
    # up the view
    # Save image using “copy” command
    # (note that you need to construct an output
    # file name, preferably from the input name)
    chimera.runCommand("close all")
  status("Finished " + dirPath)
os.chdir(curdir)            # return to original folder
```

PROCESSING MULTIPLE DATA FILES



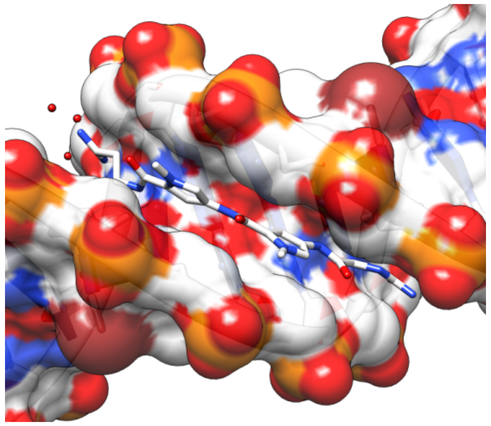
```
import os
curdir = os.getcwd()

from chimera import runCommand as rc
from chimera.replyobj import status
from OpenSave import OpenModal
from chimera.tkgui import app
dirList = OpenModal(dirsOnly=True).run(app)
for dirPath, _ in dirList:
  # loop through the files, opening,
  # processing, and closing each in turn
  os.chdir(dirPath)
  file_names = [fn for fn in os.listdir(".")
                                   if fn.endswith(".pdb")]
```

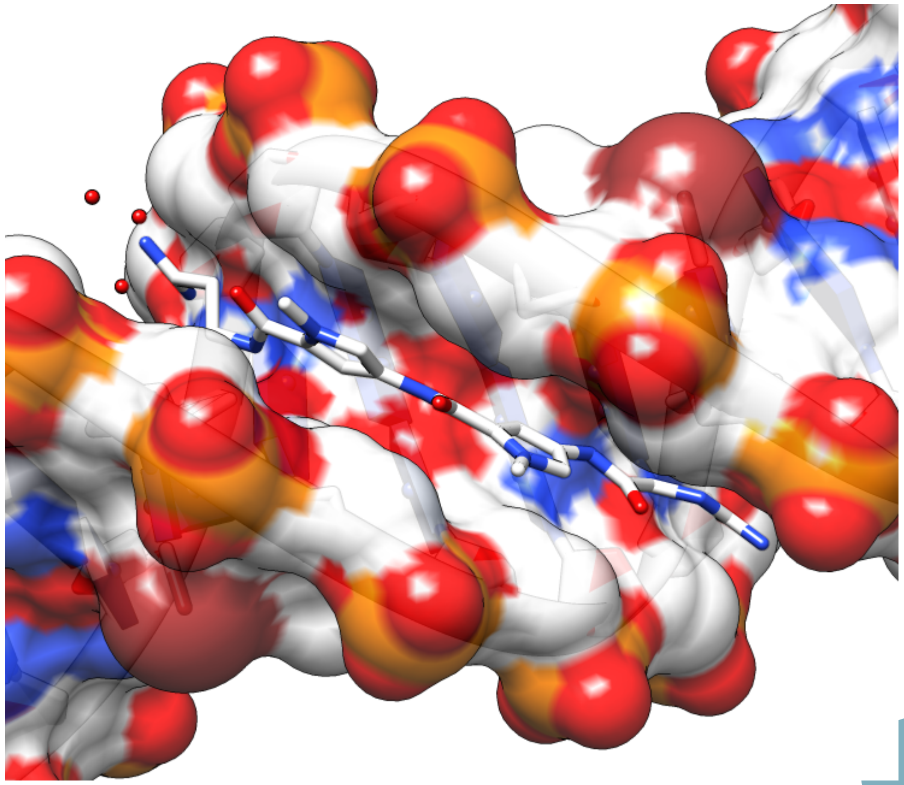
```
  for fn in file_names:
    status("Processing " + fn)
    rc("open " + fn)
    rc("align ligand ~ligand")
    rc("focus ligand")
    rc("surf")
    rc("preset apply publication 1")
    rc("surftransp 15“)
    png_name = fn[:-3] + "png"
    rc("copy file " + png_name + " supersample 3")
    rc("close all")
  status("Finished " + dirPath)
os.chdir(curdir)
status("Finished")
```

multiple_files.py – process multiple PDB files to generate images of bound ligands

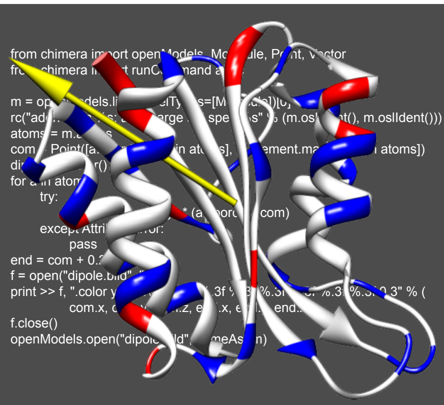
Processing Multiple Data Files

- More complete description of this script is on the web
  - http://www.cgl.ucsf.edu/chimera/docs/ProgrammersGuide/basicPrimer.html





PDB entry 6BNA
3BX2

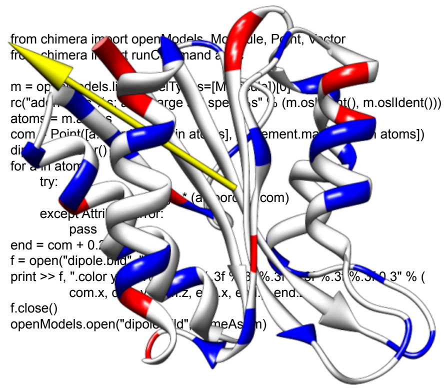


```
from chimera im ort open odels, Mo ule, Po nt, V ctor
fr  chimera i  t runC  mand a

m = op   dels.li    elTy  s=[M    ule])[0]
rc("ad         s: a    arge      spe   s" % (m.os    nt(), m.oslIdent()))
atoms = m.a   s
com   Point([a.           in at   s],      ement.ma         atoms])
di         r()
for a in atom
        try:
                         * (a  oor       com)
        except Attri       ror:
                pass
end = com + 0.
f = open("dipole.bild"
print >> f, ".color y          .3f %   %.3f    %.3  %.3 0.3" % (
                com.x, c     .z, e  .x, e    end.
f.close()
openModels.open("dipol  ld",   meAs   n)
```

Task:

Compute And Display The Dipole Moment Of A Molecule

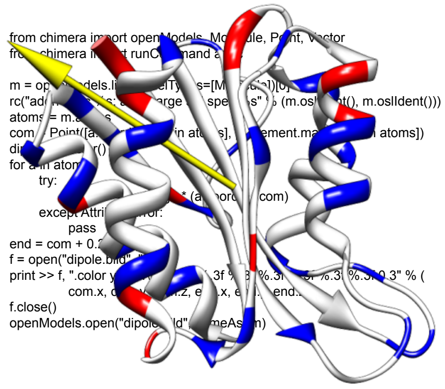
Display Dipole Moment



```
from chimera im[illegible]ort open[illegible]odels, Mo[illegible]ule, Po[illegible]nt, V[illegible]ctor
fr[illegible] chimera i[illegible]t runC[illegible]mand a[illegible]

m = op[illegible]dels.li[illegible]elTy[illegible]s=[M[illegible]])[0]
rc("ad[illegible]s: a[illegible]arge [illegible] spe[illegible]s" % (m.osl[illegible]t(), m.oslIdent()))
atoms = m.a[illegible]s
com [illegible] Point([a[illegible]in at[illegible]s], [illegible]ement.ma[illegible] atoms])
di[illegible]r()
for a in atom[illegible]
        try:
                [illegible] * (a[illegible]ord[illegible]com)
        except Attri[illegible]rror:
                pass
end = com + 0.[illegible]
f = open("dipole.bild"[illegible]
print >> f, ".color y[illegible].3f %[illegible]%.3f[illegible]%.3[illegible]%.3[illegible]0.3" % (
                com.x, c[illegible]z, e[illegible].x, e[illegible], end.[illegible]
f.close()
openModels.open("dipol[illegible]ld", [illegible]meAs[illegible]m)
```

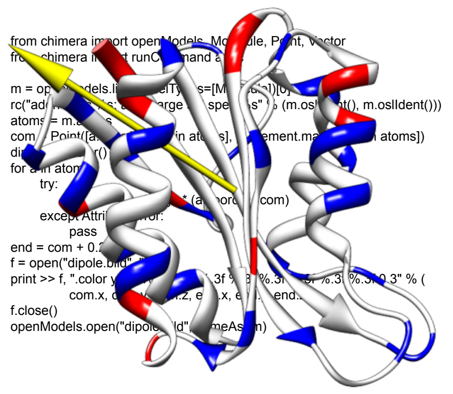
- How do we compute and display the dipole moment of a molecule?
- Plan of attack
  - Select a molecule
  - Add hydrogens with **addh** command and assign atomic partial charges with **addcharge** command
  - Compute center of mass
  - Compute dipole moment relative to center of mass
  - Compute the end points of an arrow representing the dipole moment
  - Construct and open graphics (bild) file of a single arrow

Where To Find Documentation



- Hardest part of writing Chimera scripts is finding the objects, methods and functions to use
- Places to look:
  - Chimera Programmer's Guide
    - http://www.cgl.ucsf.edu/chimera/docs/ProgrammersGuide/Examples/index.html
  - Chimera example scripts
    - http://plato.cgl.ucsf.edu/trac/chimera/wiki/Scripts
  - Chimera mailing list archives
    - http://www.cgl.ucsf.edu/chimera/docs/feedback.html
- People to ask:
  - Chimera developers' mailing list
    - chimera-dev@cgl.ucsf.edu
    - Questions are answered promptly by Chimera staff members
    - All questions are welcome!

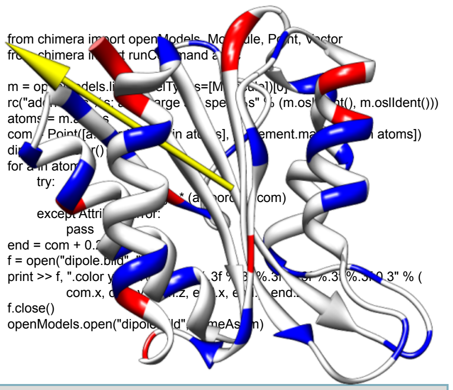
DIY Exploration



```
from chimera im[illegible]ort openM[illegible]odels, Mo[illegible]ule, P[illegible]nt, V[illegible]ctor
fr[illegible] chimera i[illegible]t runC[illegible]mand a[illegible]

m = op[illegible]dels.li[illegible]elTy[illegible]s=[M[illegible]])[0]
rc("ad[illegible]s: a[illegible]arge [illegible] spe[illegible]s" % (m.osl[illegible]t(), m.oslIdent()))
atoms = m.a[illegible]s
com [illegible] Point([a[illegible]in at[illegible]s], [illegible]ement.ma[illegible] atoms])
di[illegible]r()
for a in atom[illegible]
        try:
                [illegible]* (a[illegible]ord[illegible]com)
        except Attri[illegible]rror:
                pass
end = com + 0.[illegible]
f = open("dipole.bild"[illegible]
print >> f, ".color y[illegible].3f %[illegible]%.3f[illegible]%.3[illegible]%.3[illegible]0.3" % (
                com.x, c[illegible]z, e[illegible].x, e[illegible] end.[illegible]
f.close()
openModels.open("dipol[illegible]ld", [illegible]meAs[illegible]m)
```

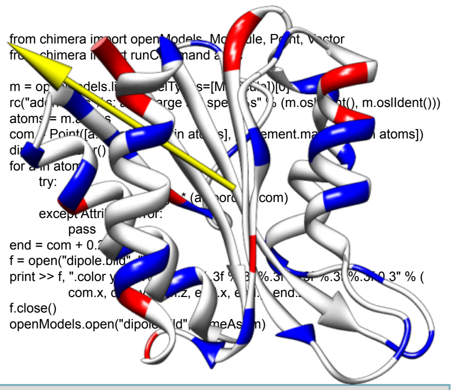
- If there are no answers to your questions on the web, and you don't want to wait for the Chimera staff to answer them, you can still Do It Yourself
- Chimera comes with the Python Integrated Development Environment (**Tools → General Controls → IDLE**)
  - Use **help(*object*)** to get code documentation (if supplied)
  - Use **dir(*object*)** to see attributes and methods
  - Use auto-completion to see available alternatives
  - Type open parenthesis to get call tip describing list of arguments to functions and methods
- Chimera comes with source code
  - All Chimera Python sources are in the share folder under the Chimera installation location

USEFUL CHIMERA INTERFACES



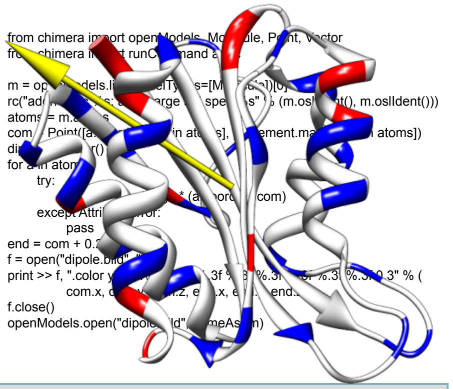
```
IDLE 2.6.4      ==== No Subprocess ====
>>> 
>>> import chimera
>>> from chimera import openModels, Molecule, Residue, Atom
>>> help(openModels.list)
Help on method list in module chimera:

list(self, id=-99, subid=-99, modelTypes=[], hidden=False, all=False) method of
_chimera.OpenModels instance
    List models from the list of open models.

    list(id = None, hidden=False, all=False, modelTypes=[]) => [models]

    id is a model identifier (an integer).  If hidden is true,
    then the hidden models are returned.  If all is true, then
    both hidden and non-hidden models are returned.  modelTypes
    is a list of model types (see the global list modelTypes)
    that restricts the types of the models returned.
```

USEFUL CHIMERA INTERFACES



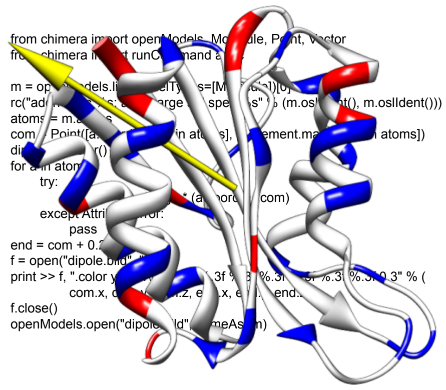
```
>>> help(openModels.open)
Help on method open in module chimera:

open(self, filename, type=None, baseId=-99, subid=-99, sameAs=None,
shareXform=True, hidden=False, defaultType=None, prefixableType=False,
checkForChanges=True, noprefs=False, *args, **kw) method of
_chimera.OpenModels instance
    Read in a file and add the models within.

    open(filename, type=None, [add arguments,] *args, **kw) -> [model(s)]

    If the type is given, then then open handler for that type is
    used.  Otherwise the filename suffix is examined to determine
    which open function to call.  See the add documentation above.

    The filename is a list of paths if the file type is specified and
    registered with a batch file reader, i.e. fileInfo.batch(type) = True.
    If the file type is not given then the filename must be a string.
```

USEFUL CHIMERA INTERFACES



```
>>> help(Atom)
Help on class Atom in module _chimera:

class Atom(Selectable)
[…]
 |  Methods defined here:
[…]
 |  Data descriptors defined here:
[…]
 |
 |  element
 |      Element
[…]
>>> from chimera import Element
>>> help(Element)
Help on class Element in module _chimera:
[…]
```

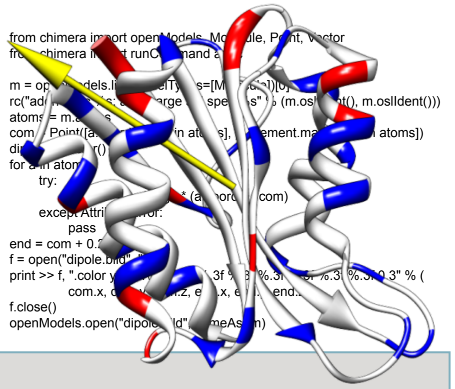
Display Dipole Moment



```
from chimera i[illegible]ort open[illegible]odels, Mo[illegible]ule, P[illegible]nt, V[illegible]ctor
fr[illegible] chimera i[illegible]t runC[illegible]mand a[illegible]

m = op[illegible]dels.li[illegible]elTy[illegible]s=[M[illegible]]) [0]
rc("ad[illegible]s; a[illegible]arge [illegible] spe[illegible]s" % (m.osl[illegible]nt(), m.oslIdent()))
atoms = m.a[illegible]s
com = Point([a.[illegible]in at[illegible]s], [illegible]ement.ma[illegible] atoms])
di[illegible]r()
for a in atom[illegible]
        try:
                [illegible]* (a[illegible]ord[illegible]com)
        except Attri[illegible]ror:
                pass
end = com + 0.2[illegible]
f = open("dipole.bild", "[illegible]
print >> f, ".color y[illegible].3f %[illegible]%.3f[illegible]%.3f %.3f 0.3" % (
                com.x, c[illegible]m.z, e[illegible].x, e[illegible] end.[illegible]
f.close()
openModels.open("dipole[illegible]ld", [illegible]meAs[illegible]m)
```

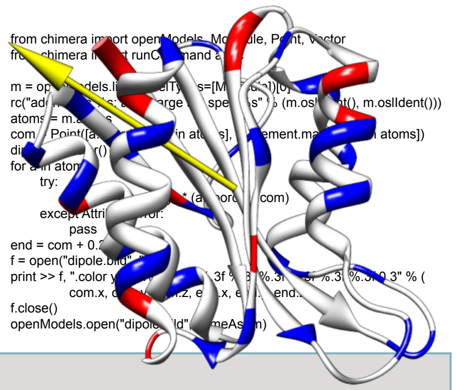
- Hints
  - To get the first molecule opened, use:
    - **m = openModels.list(modelTypes=[Molecule])[0]**
  - To assign partial charges to atoms in molecule **m**, use:
    - **chimera.runCommand("addh spec %s; addcharge std spec %s" % (m.oslIdent(), m.oslIdent()))**
    - This adds a **charge** attribute to each atom in standard residues
  - To get a list of atoms from molecule **m**, use:
    - **atoms = m.atoms**
  - To get the coordinates for atom a, use **a.coord()** which returns a Point instance
  - To get the atomic mass for atom **a**, use **a.element.mass**
  - To display arrows and other graphical objects, search for "bild" using **Help → Search Documentation…** in Chimera

Solution Scaffold



```
from chimera import openModels, Molecule, Point, Vector
from chimera import runCommand as rc

# Select model “m”
# Add hydrogens and assign charges
# Extract list of “atoms” from “m”
# Compute center of mass using special Point constructor
com = Point([a.coord() for a in atoms], [a.element.mass for a in atoms])
# Compute dipole moment from atoms with assigned charges
# Compute “end” of arrow for displaying dipole moment
# Write out a “bild” file (looks something like)
#              .color yellow
#              .arrow start_x start_y start_z end_x end_y end_z
# Open bild file using either “openModels.open” or “runCommand”
```

Display Dipole Moment

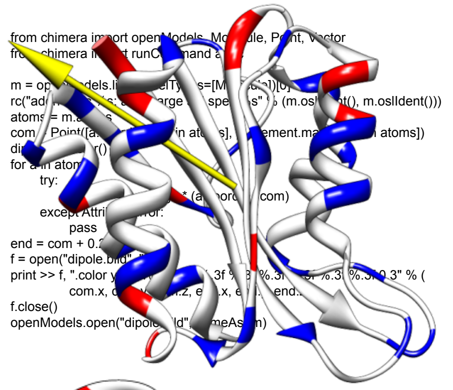


```
from chimera import openModels, Molecule, Point, Vector
from chimera import runCommand as rc

m = openModels.list(modelTypes=[Molecule])[0]
rc("addh spec %s; addcharge std spec %s" % (m.oslIdent(), m.oslIdent()))
atoms = m.atoms
com = Point([a.coord() for a in atoms], [a.element.mass for a in atoms])
dipole = Vector()
for a in atoms:
        try:
                dipole += a.charge * (a.coord() - com)
        except AttributeError:
                pass
end = com + 0.25 * dipole
f = open("dipole.bild", "w")
print >> f, ".color yellow\n.arrow %.3f %.3f %.3f %.3f %.3f %.3f" % (
                                        com.x, com.y, com.z, end.x, end.y, end.z)
f.close()
openModels.open("dipole.bild", sameAs=m)
```

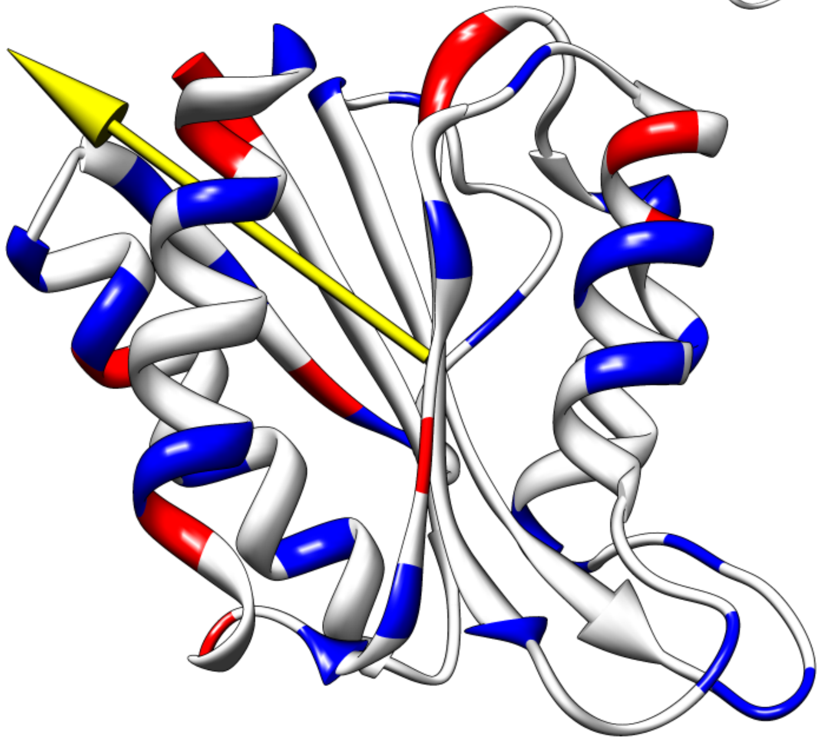
dipole.py – display a yellow arrow in the direction of the dipole moment; assumes that charges for all atoms have been assigned

Running The Script

- Opening a script in Chimera executes it
  - PDB entry 3FX2 displayed using ribbon preset in rainbow colors from N- to C-terminus
  - If we assign charges, we can assign residue ribbon color by total atomic charge
  - Execute dipole.py to display dipole moment

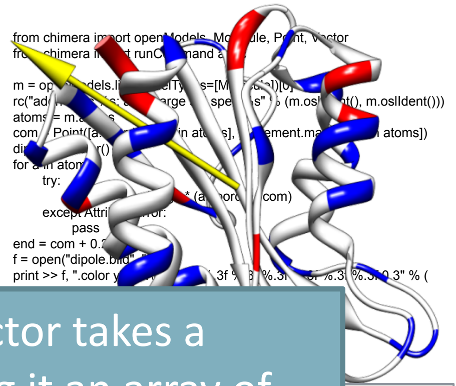


```
from chimera im[illegible]ort openModels, Mo[illegible]ule, Po[illegible]nt, Vector
fro[illegible] chimera i[illegible]t runCo[illegible]mand a[illegible]

m = op[illegible]odels.li[illegible]elTy[illegible]s=[M[illegible]])[0]
rc("add[illegible]arge [illegible] spe[illegible]s" % (m.osl[illegible]t(), m.oslIdent()))
atoms = m.a[illegible]s
com = Point([a.[illegible]in ato[illegible]s], [illegible]ement.ma[illegible] atoms])
di[illegible]r()
for a in atom[illegible]
        try:
                [illegible]* (a[illegible]ord[illegible]com)
        except Attri[illegible]ror:
                pass
end = com + 0.2[illegible]
f = open("dipole.bild", "[illegible]
print >> f, ".color y[illegible].3f %[illegible]%.3f [illegible]%.3[illegible]%.3f 0.3" % (
                com.x, c[illegible]m.z, e[illegible].x, e[illegible]end.[illegible]
f.close()
openModels.open("dipole.bild", [illegible]meAs[illegible]m)
```



PDB entry 3FX2

ANNOTATED CODE



Compute the center of mass. The **Point** constructor takes a variety of arguments. In this case, we are passing it an array of

Compute the dipole moment. The **Vector** constructor does not

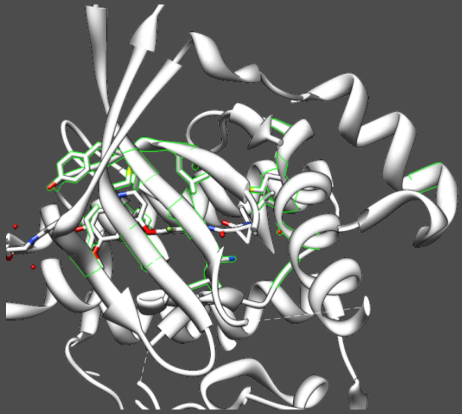
The **chimera** module provides many of the objects and classes that you'll need. **openModels** is a singleton object that manages open models. **Molecule**, **Point** and **Vector** are classes that we

To display the dipole moment, we need the coordinates of the two ends of the vector. The head of the vector is at the center of mass of atoms and points along the dipole moment vector. We scale the dipole moment vector length so that the arrow is not

Finally, we open the graphics file using the **open** method in **openModels**. Note that we associate the model with the molecule from which the dipole moment was computed. This makes our arrow rotate and translate synchronously with the molecule.

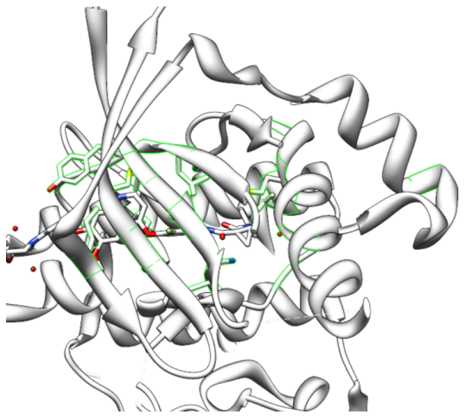
```
from chimera
from chimera

m = openMod
rc("addh spec
atoms = m.at
com = Point([
dipole = Vect
for a in atoms
	try:
		dip
	except A
		pas
end = com +
f = open("dipo
print >> f, ".co

f.close()
openModels.open("dipole.bild", sameAs=m)
```

dipole.py – display a yellow arrow in the direction of the dipole moment; assumes that charges for all atoms have been assigned



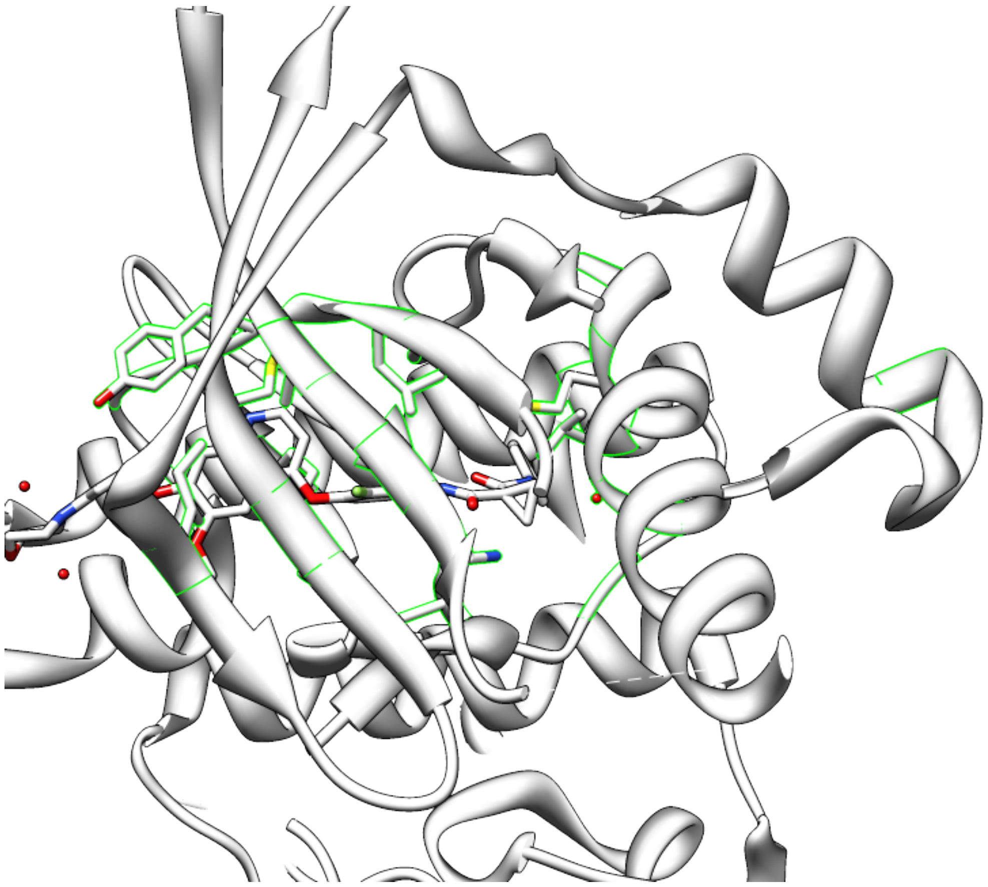
Task:
Autofocus for Active Sites

Focus On Active Site



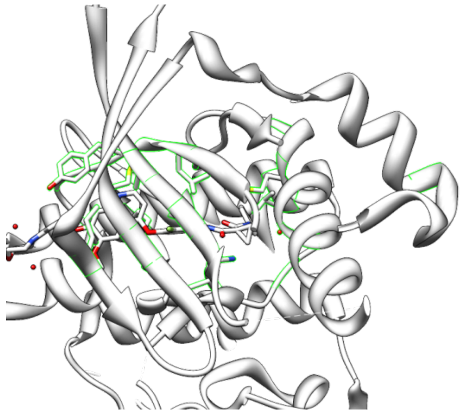
- How do we automatically focus in on an active site?
  - We cheat - by using PDB SITE records that list the residues in the active site
- Plan of attack
  - Get a molecule
    - Use code from example above
  - Extract SITE records
    - Need to know where these are kept
  - Parse SITE records and find active site residues
    - Need to know format of SITE records
  - Select active site residues
    - Need to know how to manipulate Chimera selections
  - Use Chimera’s focus command to get a reasonable view
    - Use **chimera.runCommand** again

Focus On Active Site



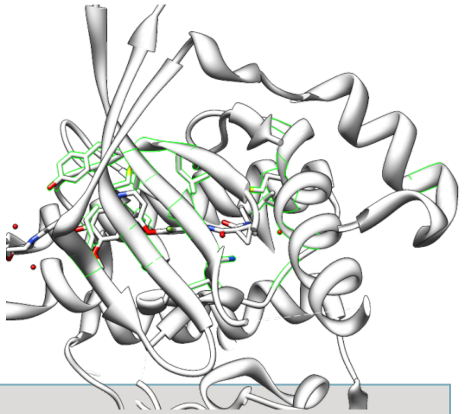
PDB entry 3LQ8 with active site selected

Focus On Active Site



- Hints
  - Some PDB records for molecule **m** are available as **m.pdbHeaders**, which is a dictionary whose keys are PDB record types and whose values are lists of PDB records
  - The PDB SITE record format is defined by the wwPDB
    - http://www.wwpdb.org/documentation/format23/sect7.html#SITE
  - To find a residue in molecule **m** when given its type, sequence number, insert code and chain identifier, use:
    - **mrid = chimera.MolResId(***chainIdentifier***,** *sequence***, insert=***insertCode***)**
      r = m.findResidue(mrid, *residueType***)**
    - **r** is either a residue or **None**
  - To create a Chimera selection from a list of atoms or residues and make it the current Chimera selection, use:
    - **sel = chimera.selection.ItemizedSelect(***list***)**
      chimera.selection.makeCurrent(sel)
  - The **focus sel** command recomputes the view to display the currently selected atoms nicely

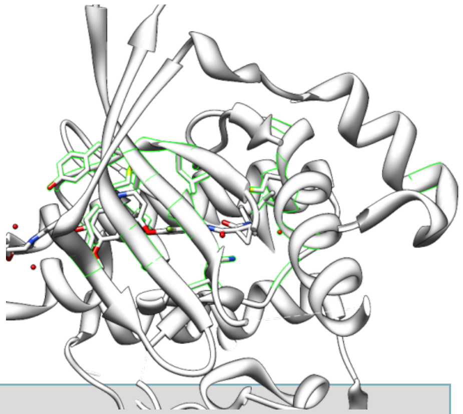
SOLUTION SCAFFOLD



```
def focusSite():
        # Select model “m”
        # Extract SITE records
        # Loop over SITE records to get residues
                # Loop over each of four residue fields and find corresponding Chimera residue
        # Select residues in Chimera using either “selection” module or “select” command
        # Update view using “focus sel” command
if __name__ == "chimeraOpenSandbox":
    focusSite()
```

```
          1         2         3         4         5
012345678901234567890123456789012345678901234567890123456789

SITE     1 AC1 23 HOH A  37  GLU A1061  ILE A1084  VAL A1092
SITE     2 AC1 23 ALA A1108  LYS A1110  GLU A1127  MET A1131
SITE     3 AC1 23 PHE A1134  LEU A1140  LEU A1157  PRO A1158
SITE     4 AC1 23 TYR A1159  MET A1160  LYS A1161  GLY A1163
SITE     5 AC1 23 LEU A1195  HIS A1202  MET A1211  VAL A1220
SITE     6 AC1 23 ALA A1221  ASP A1222  PHE A1223
```

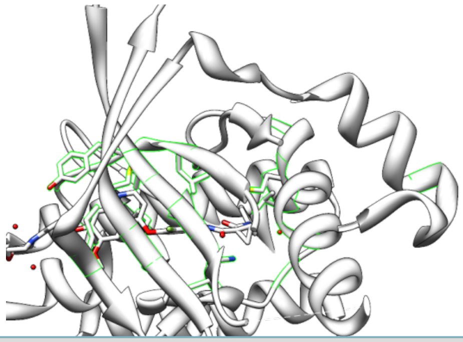
Focus On Active Site



```
def focusSite():
        from chimera import UserError
        from chimera import openModels, Molecule
        mList = openModels.list(modelTypes=[Molecule])                              # get molecule
        if not mList:
                raise UserError("There are no open molecules")
        m = mList[0]
        try:                                                                         # extract SITE records
                siteRecords = m.pdbHeaders["SITE"]
        except (KeyError, AttributeError), e:
                raise UserError("There are no SITE records for %s" % m.name)
        rList = []                                                                   # find residues
        for sr in siteRecords:
                rList.extend(parseSiteRecord(m, sr))
        from chimera import selection                                               # select residues
        selection.setCurrent(selection.ItemizedSelection(rList))
        from chimera import runCommand                                              # run ‘focus’ command
        runCommand("focus sel")
```

focus_site.py – focus on residues listed in SITE records in PDB file (continued on next slide)

FOCUS ON ACTIVE SITE

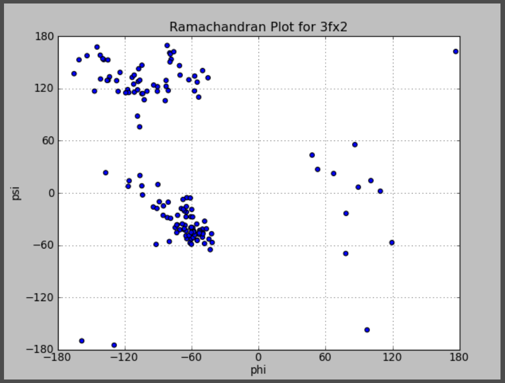


```
def parseSiteRecord(m, sr):
    print sr
    from chimera import MolResId
    rList = []
    for srPos in SiteResiduePositions:
        s, e = srPos[2]
        seq = sr[s:e].strip()
        if not seq:
            # No sequence -> no residue
            continue
        seq = int(seq)
        s, e = srPos[0]
        rType = sr[s:e].strip()
        s, e = srPos[1]
        chainId = sr[s:e].strip()
        s, e = srPos[3]
        iCode = str(sr[s:e])
        mrid = MolResId(chainId, seq, insert=iCode)
        r = m.findResidue(mrid, rType)
        if r:
            rList.append(r)
    return rList
```

```
SiteResiduePositions = (
    # Each SITE record lists up to four residues
    # Each residue consists of name, chain id,
    # sequence number and insert code
    ( (18,21), (22,23), (23,27), (27,28) ),
    ( (29,32), (33,34), (34,38), (38,39) ),
    ( (40,43), (44,45), (45,49), (49,50) ),
    ( (51,54), (55,56), (56,60), (60,61) ),
)
```

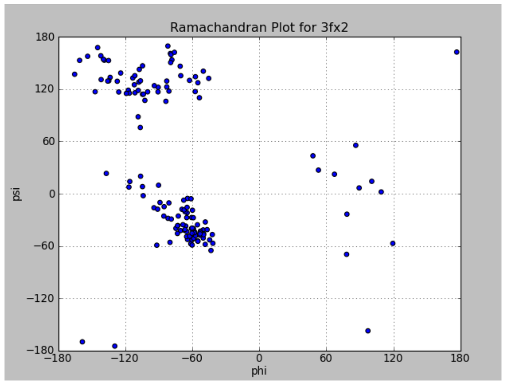
```
if __name__ == "chimeraOpenSandbox":
    focusSite()
```

focus_site.py – focus on residues listed in SITE records in PDB file



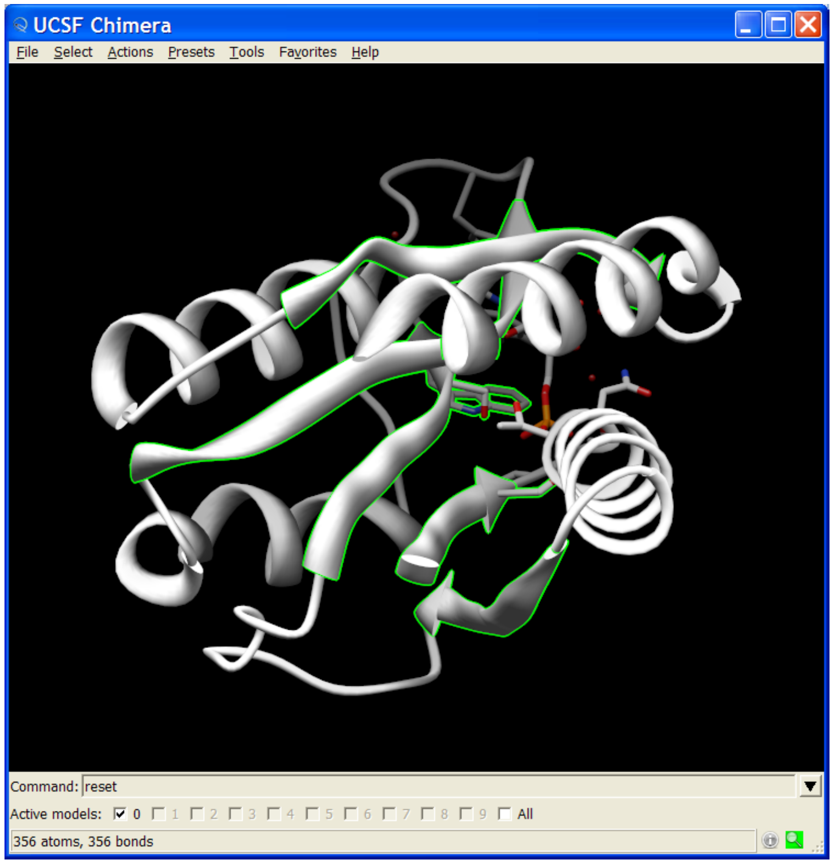
Task:
Create Ramachandran Plot

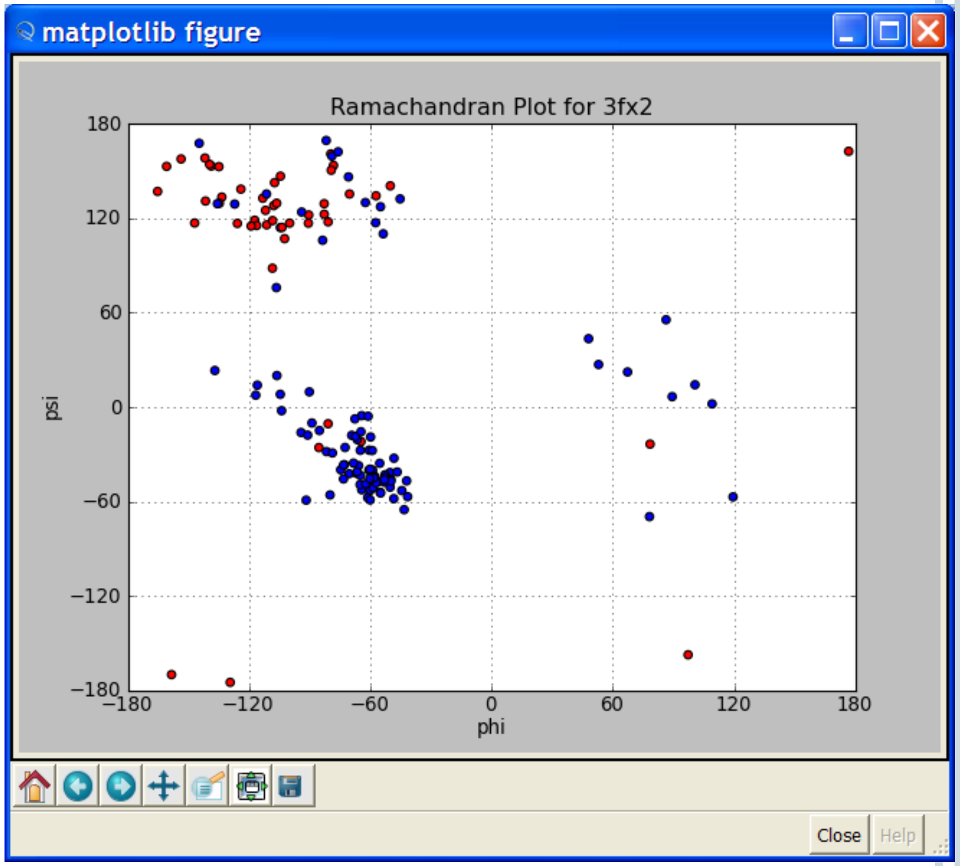
CREATE RAMACHANDRAN PLOT



- How can we display a Ramachandran (ϕ-ψ) plot for a protein?
- Plan of attack
  - Select molecule
  - Extract ϕ and ψ angles
    - hint: they're precomputed for amino acid residues
  - Display scatter plot using matplotlib
    - hint: matplotlib documentation has very nice examples
      - http://matplotlib.sourceforge.net/users/screenshots.html#scatter-demo
    - use glue code on slide below to create matplotlib figures in Chimera
  - Optional: select residues when plot points are selected
  - Extra Credit: highlight plot points when residues are selected

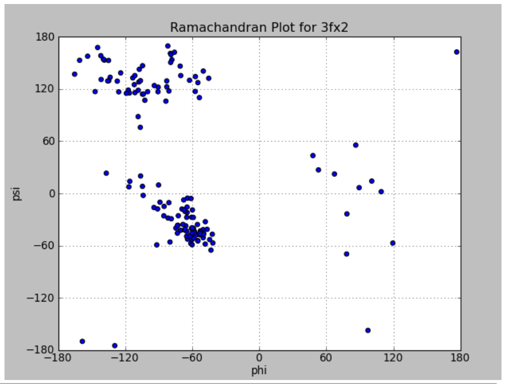
Create Ramachandran Plot


UCSF Chimera
File Select Actions Presets Tools Favorites Help
Command: reset
Active models: 0 1 2 3 4 5 6 7 8 9 All
356 atoms, 356 bonds



PDB entry 3FX2

Create Ramachandran Plot

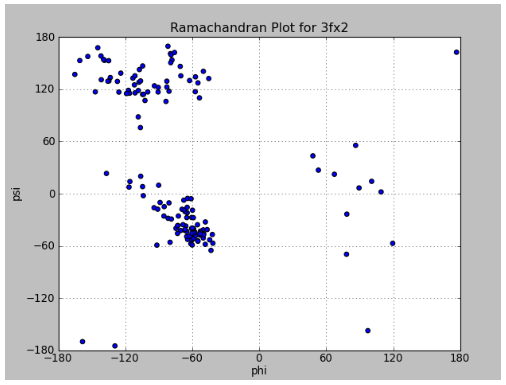


```
from chimera.baseDialog import ModelessDialog
class PlotDialog(ModelessDialog):
  "PlotDialog is a Chimera dialog whose content is a matplotlib figure"
  buttons = ('Close',)
  title = "matplotlib figure"
  def __init__(self, showToolbar=True, **kw):
    self.showToolbar = showToolbar
    ModelessDialog.__init__(self, **kw)
  def fillInUI(self, parent):
    from matplotlib.figure import Figure
    self.figure = Figure()
    from matplotlib.backends.backend_tkagg \
      import FigureCanvasTkAgg, NavigationToolbar2TkAgg
    fc = FigureCanvasTkAgg(self.figure, master=parent)
    fc.get_tk_widget().pack(side="top", fill="both", expand=True)
    self.figureCanvas = fc
    if self.showToolbar:
      nt = NavigationToolbar2TkAgg(fc, parent)
      nt.update()
      self.navToolbar = nt
    else:
      self.navToolbar = None
```

```
  def add_subplot(self, *args):
    return self.figure.add_subplot(*args)

  def delaxes(self, ax):
    self.figure.delaxes(ax)

  def draw(self):
    self.figureCanvas.draw()

  def registerPickHandler(self, func):
    self.figureCanvas.mpl_connect("pick_event", func)
```

plotdialog.py – glue code for creating a **matplotlib** figure inside a Chimera dialog

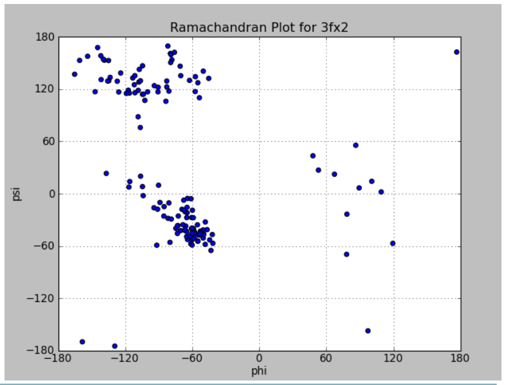
Solution Scaffold



```
import sys; sys.path.append('.')
from plotdialog import PlotDialog
del sys.path[-1]

class RamachandranPlot(PlotDialog):
    def __init__(self, m):
        PlotDialog.__init__(self)                       # Initialize PlotDialog base class
        # Construct list of phi, psi lists from molecule "m"
        self.subplot = self.add_subplot(1,1,1)
        self._displayData()

    def _displayData(self):
        ax = self.subplot
        # Display scatter plot data
        # Set plot labels, axis, etc.
        self.draw()

from chimera import openModels, Molecule                # Select first open model as "m"
m = openModels.list(modelTypes=[Molecule])[0]
RamachandranPlot(m)                                     # Construct Ramachandran plot from "m"
```

CREATE RAMACHANDRAN PLOT

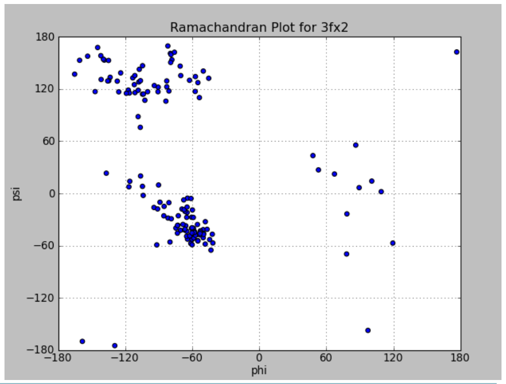
```
import sys; sys.path.append('.')
from plotdialog import PlotDialog
del sys.path[-1]

class RamachandranPlot(PlotDialog):
    def __init__(self, m):
        PlotDialog.__init__(self)
        self.mol = m
        self.phi = []
        self.psi = []
        self.residues = []
        for r in m.residues:
            if r.phi is None or r.psi is None:
                continue
            self.phi.append(r.phi)
            self.psi.append(r.psi)
            self.residues.append(r)
        self.selectedIndices = []
        self.subplot = self.add_subplot(1,1,1)
        self._displayData()

        self.registerPickHandler(self.onPick)
        from chimera import triggers
        self.selHandler = triggers.addHandler(
                    "selection changed",
                    self._selectionChangedCB, None)
```



```
    def _displayData(self):
        colors = ['b'] * len(self.phi)
        for n in self.selectedIndices:
            colors[n] = 'r'
        ax = self.subplot
        ax.clear()
        ax.scatter(self.phi, self.psi, c=colors,
                            picker=True)
        ax.set_xlabel("phi")
        ax.set_xlim(-180, 180)
        ax.set_xticks(range(-180, 181, 60))
        ax.set_ylabel("psi")
        ax.set_ylim(-180, 180)
        ax.set_yticks(range(-180, 181, 60))
        ax.set_title("Ramachandran Plot for %s" %
                            self.mol.name)
        ax.grid(True)
        self.draw()
```

ram3.py – display Ramachandran plot of first open molecule (continued on next slide)

Create Ramachandran Plot



```
def onPick(self, event):
    residues = [ self.residues[i] for i in event.ind ]
    from chimera import selection
    sel = selection.ItemizedSelection(residues)
    selection.setCurrent(sel)

def _selectionChangedCB(self, trigger, userData, ignore):
    from chimera import selection
    residues = selection.currentResidues()
    self.selectedIndices = []
    for r in residues:
        try:
            n = self.residues.index(r)
        except ValueError:
            pass
        else:
            self.selectedIndices.append(n)
    self._displayData()
```

```
def destroy(self):
    if self.selHandler:
        from chimera import triggers
        triggers.deleteHandler(
                "selection changed",
                self.selHandler)
        self.selHandler = None
    PlotDialog.destroy()

from chimera import openModels, Molecule
m = openModels.list(modelTypes=[Molecule])[0]
RamachandranPlot(m)
```

ram3.py – display Ramachandran plot of first open molecule

Wrap Up

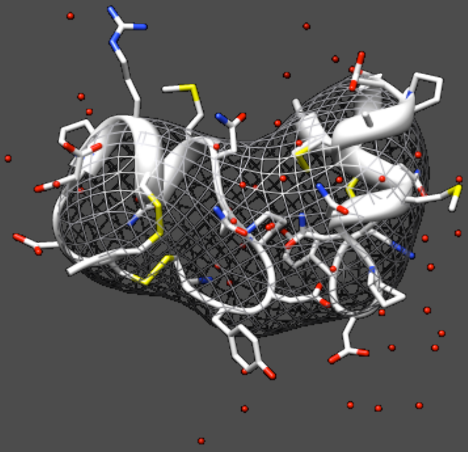
What didn't we cover?

More Elaborate Projects

- Write Chimera scripts that do not require graphical display
  - Use the –nogui command line flag
- Write Chimera scripts that generate images without using a display window
  - Use the headless version of Chimera
- Write Chimera extensions with graphical user interfaces
  - See TemplateExtension folder
- Use web services to fetch information in addition to data available in Chimera
  - Use WebServices package from Chimera
- Write Chimera scripts that use grid data
  - Use VolumeViewer package from Chimera

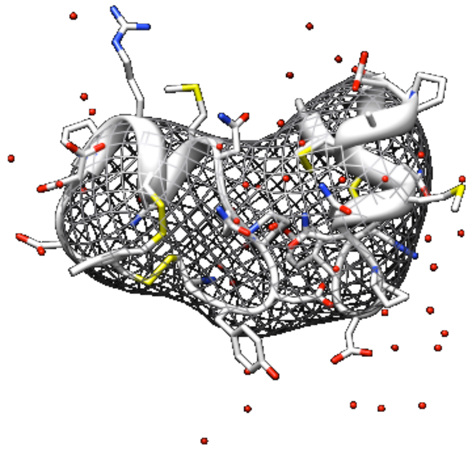
The End

- Questions?



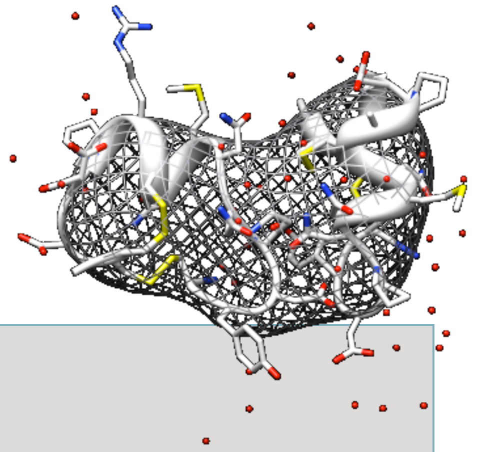
Bonus:
Display Protein Core

DISPLAY PROTEIN CORE



- How can we display the volume occupied by the "core" of a protein?
  - Example is contrived to show some non-molecular data display in Chimera
- Plan of attack
  - Select molecule
  - Construct empty data grid based on molecular dimensions
  - Add weighted sphere for each atom
    - sphere weight scaled by atom temperature factor
    - sphere radius proportional to atom van der Waals radius
    - Chimera uses lots of **NumPy** arrays for volume calculations
  - Display grid in Volume Viewer

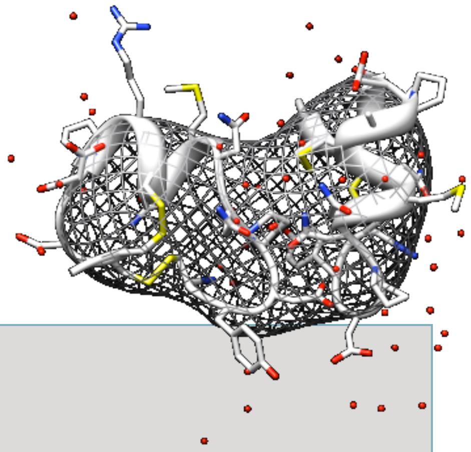
DISPLAY PROTEIN CORE



```
def bfactor_map(molecule, grid_spacing = 1, grid_padding = 5,
                cutoff_range = 5, bspread = 25):

   # Figure our atom coordinates and bounding box.
   atoms = [a for a in molecule.atoms
          if hasattr(a, 'bfactor') and a.bfactor > 0]
   from _multiscale import get_atom_coordinates, bounding_box
   xyz = get_atom_coordinates(atoms)
   xyz_min, xyz_max = bounding_box(xyz)
   from math import ceil, pi, pow
   shape = [int(ceil((xyz_max[a]-xyz_min[a]+2*grid_padding) / grid_spacing))
          for a in (2,1,0)]

   # Determine Gaussian standard deviations (index coordinates) and amplitudes
   origin = [x-grid_padding for x in xyz_min]
   ijk = (xyz - origin) / grid_spacing
   f = bspread/(8*pi*pi*grid_spacing)
   sdevs = [a.bfactor*f for a in atoms]
   sdev3d = [(sd,sd,sd) for sd in sdevs]
   scale = [a.element.number / pow(a.bfactor,3) for a in atoms]
```

corevolume.py – s how a surface for the rigid core of a molecule, determined from B-factors (continued on next slide)

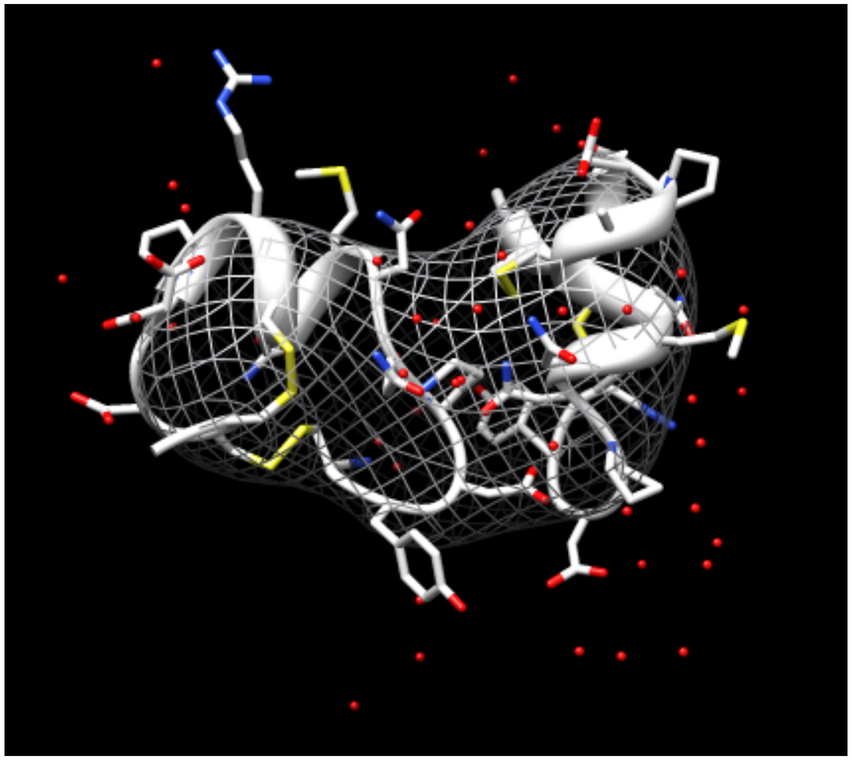
DISPLAY PROTEIN CORE



```
    # Make 3-d array by adding Gaussians.
    from numpy import zeros, float32
    matrix = zeros(shape, float32)
    from _gaussian import sum_of_gaussians
    sum_of_gaussians(ijk, scale, sdev3d, cutoff_range, matrix)

    # Create and display volume model.
    from VolumeData import Array_Grid_Data
    grid = Array_Grid_Data(matrix, origin,
                           (grid_spacing,grid_spacing,grid_spacing),
                           name = 'B-factor map')
    from VolumeViewer import volume_from_grid_data
    v = volume_from_grid_data(grid, 'mesh')

    return v

# ---------------------------------------------------------------------------
#
from chimera import openModels, Molecule
for m in openModels.list(modelTypes = [Molecule]):
    bfactor_map(m)
```

corevolume.py – s how a surface for the rigid core of a molecule, determined from B-factors

Display Protein Core



PDB entry 1A0M displayed with isosurface of volume computed using atom temperature factors