Programming Scalable Scientific Workflows

Instructors:
- Michel Sanner, Ph.D. (TSRI)
- Guillaume Vareille, (TSRI)
- Luca Clementi, (UCSD)
- Jane Ren, (UCSD)
- Wilfred Li, Ph.D. (UCSD)

Summer Institute 2008, San Diego CA, August 3-7
Week Overview

• Day 1: Introduction to Vision (M. Sanner, G. Vareille)
  • Basic interactions, basic networks, ImageViewer

• Day 2: Building an application (M. Sanner, G. Vareille)
  • Writing nodes, User Library, User panels, moving widgets, noGUI execution

• Day 3: Extending Pmv (M. Sanner, G. Vareille)
  • Extending Pmv using Vision, Icosahedral capsid

• Day 4: Web Services (L. Clementi, J. Ren, W. Li)
  • Building a Virtual Screening application

• Day 5: (M. Sanner, G. Vareille)
  • MatPlotlib, Volume, Student applications, wrap-up
Day 1 Overview

• Background and Motivation
• Installing and starting Vision
• Interacting with the Vision GUI
  – GUI elements
  – Mouse bindings
• Building Networks
  – Image Viewer
  – . . .
Visual Programming

- Abstraction of programming syntax and data structure
- Setting up flexible workflows

Libraries of nodes performing simple tasks
Programming areas where computational nodes are assembled into networks

Show Image
Vision: what is different

• Vision is built from software components
Vision: what is different

- Vision is a software component itself
Vision: what is different

- No constraining data types or data model
- Fully scriptable
- Highly interactive
Installing Vision

http://mgltools.scripps.edu/download
Installing Vision

Welcome to the InstallJammer Wizard for MGLTools

This will install MGLTools version 1.4.4 on your computer. MGLTools is developed at the Molecular Graphics Lab (MGL) of the Scripps Research Institute for 3D visualization and analysis of molecular structures. The following stand-alone applications are included in MGLTools:

- PMV - Python Molecular Viewer
- Vision - Visual Programming Environment
- ADT - AutoDockTools

Click Next to continue or Cancel to exit Setup.

Thank you for downloading MGLTools version 1.4.4

This distribution contains a pre-compiled Python interpreter (version 2.4), a collection of pure Python packages (including Pmv, AutoDockTools, DojoVu, MolKit, VizualFramework, Vision, NetworkEditor, etc.) various platform-independent packages (including mml, openGL, gnome, gle, stride, elf, cAutoDock, LIPSackages, etc.), and the PMV, ADT and VISION applications build on top of these software packages.
Commercial versus Academic

The following software components have restrictions for commercial use:

- Mslib: Computing molecular surfaces
- Stride: Secondary structure assignment
- UTPackages: Volume rendering and iso-contouring

The installer will display a GUI to pick which packages to install.
Commercial versus Academic

Check to install all 3 packages

Check to not install any of the 3 packages

Select individual packages for installation

Deselect individual packages for installation
Starting Vision

The installer places 3 icons on the desktop. To start Vision double click on the Vision Icon.
Vision: GUI elements

Menus
Node Library
Node Category
Node

Current network name Network 0
Programming Canvas for Network 0
Node Documentation

NOTE Node documentation displays as tooltips

Turn an incoming object into some other type. The cast method of the newtype will be called with the incoming data. If successful, the result will be output.

Input Ports
- data: any python object or list of python objects
- newtype: (bound to a combo box widget) name of a registered type as obtained from the editor's typeManager

Output Ports
- result: the new object
Vision: GUI elements

- Create new network
- Merge network from file into current network
- Save network
- Load network from file
- Load libraries
- Print network
- Minimal run
- Pause execution
- Detached run
- Immediate mode toggle
- Run network
- Stop execution
- Stop detached execution
Vision: GUI elements

- Show execution
- Time GANTT chart
- Step to next node
- Display advanced Search interface
- Run new network
- In debug mode
- Cut, Copy, Paste
- Selected nodes
- Search nodes by name in loaded libraries
Some Vision node libraries:
- Standard: default Vision node library
- Pmv: Vision interface to Pmv
- Imaging: Interface to the Python Imaging Library (PIL)
- Matplotlib: 2D graphing library
- MolKit: working with biological molecules
- Volume: working with 3D regular grids of scalar data
- Ipython: Vision node for using IPython parallel computing
- 3D Visualization: Vision interface to DejaVu
- Symserv: working with point symmetry operators
- Web services: Vision nodes for Opal web services servers
- Adt: Vision interface to AutoDockTools (ADT)
Loading Libraries

Using the Menu

Using the button

Static list of libraries

Dynamic discovery of libraries
Exercise: locate a node by name

Task: Find the Dial node in the Standard library

Solution:
1 – left click in the search box (A)
2 – type dial <enter>.
   notice how the Dial node is temporarily highlighted in the library
if you missed it, type <enter again>

Alternative solution:
1 – left click in the Input category of the Standard Node library
2 – type the letter “d”
   notice how the category scrolls to show the Dial node
Exercise: drag and drop a node

Task: Drag and drop a Dial node on the programming canvas

1 – left click on the Dial node in the Input category of the Standard library
2 – drag the mouse to the canvas (without releasing the button)
3 – release the button where you want to place the node
- Sometimes ports with different data types can be connected.
  ◊ is the None data type, i.e., any Python object is accepted.
  Such ports will “morph” to the incoming data type.
- Examples of datatypes:
  ◇ string, ◇ int, ◇ image, ◇ list, ◇ boolean, ...
- Input ports can be “required” or “optional”. Valid date is necessary on “required” ports for a node to run.
File browser Widget is Bound to the `filename` Input port

Widget can be unbound

- Unbound widget can be rebound
- Widgets are for interactive user input

The `filename` Input port Becomes exposed at the Top of the node, allowing A name computed in the Network to be passed to The node
def doit(self, filename):
    import Image
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im)
Exercise

Task: Drag and drop a Print node on the programming canvas

1 – left click on the Output category of the Standard library and type “p”
2 – click in the print Node and drag it to the canvas and release the button
Exercise

Task: Connect the output of Dial to the input port of Print

1 – left click on the Output Port of the Dial node (A)
2 – drag the cursor (without releasing mouse button). A green line is drawn.
3 – move cursor close to “print” node’s input port. When you are close the line will snap to the port
4 – release the mouse button to create the connection

Port type morphed to float
Exercise: modifying parameters

Task: Modify the Dial value

1 – click on the handle and drag the cursor OR with the cursor over the dial type numbers

Note that:
- values are printed to the shell

Task: Prevent network execution on new data

1 – Toggle the *Toggle Immediate* icon in the toolbar

Modify the Dial value and note that values no longer print to the shell
Notes: node outlines

Node outlines:
- red: running
- orange: tried to run but missing data
- sick green: node failed

Turn off outlines using Edit-> Flash nodes when run
Exercise: Run Network

Task: Run the whole network, independently of data status

1 – Click on the Run button in the tool bar

Note that dial value is printed to the shell each time you run

Task: Make minimal run, i.e. run only nodes that have new input data (and their children)

1 – Click on the Minimal Run button in the tool bar

Note that dial value is printed when the dial has a new value but nothing is printed if the value of the dial is unchanged
Exercise: selecting

Task: Select the dial node

1 – Left click on the canvas background
2 – drag the cursor to draw a box around the Dial node
3 – release the mouse button to toggle nodes between selected and deselected mode

- Left click on background to clear the selection
- Ctrl-A selects all

Left click on background

Drag red box around nodes to select

Release to toggle Dial node from deselected to selected
Exercise: move nodes and connections

Task: move the Dial and print nodes together

1 – select nodes to move
2 – middle click on canvas background and drag cursor
3 – release the mouse button to toggle nodes between selected and deselected mode

Note: using the middle mouse button on canvas background with no selection scrolls the canvas
Exercise: move a single node

Task: move the Dial node alone

1 – select both nodes
2 – middle click on the node background. This node becomes temporarily selected.
3 – drag cursor, the node moves along.
4 – release the mouse button. The previous selection is restored.
Exercise: copy-paste parts

Task: duplicate the dial-print network

1 – select both nodes. Note the copy button in the toolbar becomes active.

2 – click on the copy button. Note the paste button becomes active.

3 – click on the paste button. Note the pasted nodes are now selected and can be moved.
Exercise

Task: delete a connection

1 – right click on the connection between the pasted Dial and print nodes
2 – choose delete in the menu
Exercise: deleting

Task: delete a single node
1 – right click on the selected print node
2 – choose “Delete” in the menu

Task: delete multiple nodes and connections
1 – select all nodes
2 – click on the cut icon in the tool bar

Task: delete all nodes
1 – use Ctrl-A to select everything in the network
2 – click on the cut icon in the tool bar
Exercise: loading a library

Task: load the Python Imaging library (PIL)

Using the button bar
1 – click on the “load library” icon
2 – select “imagelib”
Exercise: Image viewer network

Task: create a network to display an image

1 – locate and instantiate a “Read Image” node
2 – locate and instantiate a “Show Image” node
   Note that a new window is created. It will hold the image.
3 – Connect the output of Read Image to the input of Show Image
4 – click on the file browser icon I the node

5 – navigate to Desktop/TutorialData/frames/ and select frame0000.png
Exercise: scale image

Task: modify the network to allow scaling the image

1 – locate and instantiate the scale node
2 – delete the connection between Read Image and Show Image
3 – connect Read Image to input of Scale and output of Scale to Show Image
4 – modify scale value
Exercise: rotate image

Task: modify the network to allow rotating the image

Note how sharp edges become jagged
Exercise: rotate image

Task: fix image edges using rotation interpolation filter

1. Read documentation string of the Rotate node. Notice the mention of a filter. Since the filter port is not visible it must be bound to a widget. Since the widget for filter is not visible in the node it must be in the node’s Parameter Panel.
2. Right-click on the rotate node and select Parameter Panel.
3. In the parameter panel choose BICUBIC.
Network Items Menus

Right click on network items to display their menus
Node Menu

- Run this node and its children
- Freeze this node. Frozen nodes do not run
- Start the node editor
- Displays a window containing widgets associated with this node
- Copy, Cut, Delete this node
- Replace node with a fresh copy
- Displays a run port on the right side of the node and a trigger port on the left side
Widget Menu

Display option panel
Only for Dial and Thumbwheel
Start widget and port editor
Show port at top of node
Move widget to node or panel
Connection Menu

Delete connection
Hide connection
Toggle connection’s blocking flag. Non blocking means a child can execute before the parent has
Network Menu
Task: Build an image browser to look at the images located in the movieFrame directory on the desktop

1 – locate and instantiate a NumberedName node
2 – set the directory to Desktop/TutorialData/frames
3 – set the base name to ‘frame’
4 – set padding to 4
5 – locate and instantiate a Thumbwheel node outputting an integer
6 – connect the output of Thumbwheel to the number input port of NumberedName
7 – locate and instantiate a Read Image node form the imaging library
8 – unbind the file browser widget from the filename input port in Read Image
9 – connect the filename output of the NumberedName node to the filename input port of the read Image node
10 – add the show Image node and connect the output of Read Image to the output of Show Image
Exercise: browser1

Task: Build an image browser to look at the images located in the tutorialDat/frames directory on the desktop.

Use Edit -> Color node by library to get node colors.
Exercise: save network

Task: Save the network

1. Click on the save button on the tool bar
2. Replace Network0_net.py by browser1_net.py in the browser
3. Click on save

The network name changed to browser1
Exercise: iterate over images

Task: make the network iterate over the file numbers

1 – delete the thumbwheel node
2 – locate the `range` node, read its documentation and instantiate one to create a list of numbers ranging from 0 to 100
3 – locate and instantiate an `iterate` node
4 – send the list created by `range` into the `iterate` node
5 – connect the `oneltem` output port of `iterate` to the `number` input port for the `NumberedName` node
6 – Run the `iterate` node

Task:
Save the network as “browser2_net.py”
Exercise: save MPEG

Task: save the sequence of images as an MPEG movie

1 – locate and instantiate Record MPEG Movie
2 – feed the image to the first input port
3 – connect the begin and end output ports of iterate to the begin and end input ports of Record MPEG Movie
4 – run the iterate node

Task:
Save the network as “saveMPEG_net.py”
Exercise: Play back movie

Task: play the MPEG movie

1 – locate and instantiate Play MPEG node
2 – unbind the widget from its movieFileName port
3 – feed the movie name from the Record MPEG node into the Play MPEG node
4 – kill the MPEG player to end the Play MPEG node’s execution
Day 2 Overview

- Look inside a node and modify it
- User library of nodes
- Writing a new node
- User panels and moving widgets
- Command line execution
- Detached execution
- Macro nodes
Exercise: Node editor

Task: start the node editor on the *Read Image* node

1 – Load the imaging library
2 – locate and instantiate a *Read Image* node
3 – Right click on the node’s background and select “Edit”
Notes: Node editor

- Edit node name
- Add/delete/edit Input Ports
- Add/delete/edit Output ports
- Edit widget gridding
- Edit node’s function
- Add port
- Delete port
- Start widget editor
- Start port editor
- Start widget placement editor
- Start code editor
Notes: Node editor

- The node name is Read Image
- 1 input port called filename
- 1 output port called image
- Note the input port is not visible at the top of the node because it gets its data from a widget which is placed inside the node.
- Double clicking on the node toggles showing/hiding widgets in the node
Exercise: Code editor

Task: start the code editor on the Read Image node

1. check the Edit ... check button in the node editor

```python
def doit(self, filename):
    if filename:
        im = Image.open(filename)
    if im:
        self.outputData(image=im)
```

- The arguments to the node’s function are named after the Input ports
- Data is output using the `self.outputData(portName=value, ...)`
Exercise: Node editor

Task: add an output port that will provide the image size

1 – click on “Add Output Port”

- An output port called “out1” is created
- The port appears on the node
- Code is added to the function showing how to output data on the port

```python
def doit(self, filename):
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im)

## to output data on port out1 use
## self.outputData(out1=data)
```
Exercise: Node editor

Task: add an output port that will provide the image size

2 – modify the function to output “im.size” on the new output port

```python
def doit(self, filename):
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im, out1=im.size)

## to output data on port out1 use
## self.outputData(out1=data)
```
Exercise: Node editor

Task: add an output port that will provide the image size

3 – Click Apply to set the function

```python
def doit(self, filename):
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im, out1=im.size)
    # to output data on port out1 use
    # self.outputData(out1=data)
```

OK does the same as Apply and hides the Code editor
Cancel hides the code editor without applying changes
Exercise: Node editor

Task: read an image and verify that the port outputs the dimensions

1 – read an image
2 – use the output port tooltip to look at the size

Note that the data type is None
Exercise: Node editor

Task: rename the new port ‘size and change its data type to ‘list’

1 – Click ‘edit port’ checkbutton for ‘out1’
2 – change the name in the port editor
3 – set the type to ‘list’
4 – click OK
The port was renamed

The port’s icon has changed reflecting the list data type

Our modification is unchanged

The example code was modified

```
def doit(self, filename):
    if filename:
        im = Image.open(filename)
    if im:
        self.outputData(image=im, out1=im.size)
## to output data on port size use
## self.outputData(size=data)
```
Exercise: Node editor

Task: fix the function after renaming the ‘out1’ port

1 – replace ‘out1’ by ‘size’ in the function
2 – click Apply
3 – use port’s tooltip to verify that the node name has changed

```python
def doit(self, filename):
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im, size=im.size)

### to ouput data on port size use
### self.outputData(size=data)
```
Exercise: Node editor

Task: add the ability to output a thumbnail of the image upon request

1 - Add an input port allowing to specify whether a thumbnail is wanted or not:
1.a – click on ‘Add Input Port’ (Note the new argument in the code editor, and the new port on the node’s icon)
1.b – click on ‘edit port’ for the newly added port
1.c – change the name to ‘thumbnail’ and the data type to ‘boolean’ in the port editor

```python
def doit(self, filename, thumbnail):
    if filename:
        im = Image.open(filename)
    if im:
        self.outputData(image=im, size=im.size)
    # to output data on port size use
    # self.outputData(size=data)
```
Exercise: Node editor

Task: add the ability to output a thumbnail of the image upon request

2 – Bind a check button widget to the new input port
2.a – click on ‘edit port’ for the ‘thumbnail’ input port
2.b – select NECheckButton for the type of widget
2.c – select ‘node’ for where to place the widget
2.d – click OK
Exercise: Node editor

Task: add the ability to output a thumbnail of the image upon request

3 - Name the widget
3.a – Click on ‘widget grid config’ in the node editor
3.b – Type “make Thumbnail” in the name field
3.c – click OK
Exercise: Node editor

Task: add the ability to output a thumbnail of the image upon request

4 – add an output port for the thumbnail image
4.a – Click on ‘Add Output Port’ (Note new output port on the node’s icon)
4.b – Click on ‘edit port’ for the newly added port
4.c – change the name to ‘thumbnail’ and the data type to ‘image’ in the port editor
Exercise: Node editor

Task: add the ability to output a thumbnail of the image upon request

5 - Edit the function to use the thumbnail argument

```python
def doit(self, filename, thumbnail):
    if filename:
        im = Image.open(filename)
        if im:
            th = None
            if thumbnail:
                th = im.copy()
                th.thumbnail((64, 64), Image.ANTIALIAS)
            self.outputData(image=im, size=im.size, thumbnail=th)
```
Exercise: Node editor

Task: save network with modified node

1 - use File -> Save ...
2 - save as ‘modif_net.py’

Task: reload network and verify all modifications

1 – click on the load network icon in the tool bar
2 – load the network called modif_net.py
3 – verify that all added input and output ports are restored
4 – verify the node’s function has the modifications

The loaded network as a trailing ‘1’ in its name to make
Exercise: Node editor

Task: make the thumbnail size a parameter controlled by a thumbwheel widget

1 – add input port with good name and type integer
2 – bind a thumbwheel and place it in the node
3 – modify the function to use the value provided by the port
Exercise: User library

Task: save the modified Read Image node in our own library

1 – right-click on the node’s background and select “save as customized node”
2 – navigate the file browser to the Input folder of “MyDefaultLib”
3 – edit the file name to be “MyReadImage.py”
4 – click on Save
Notes: User library

The library is automatically loaded

The node appears in the Input category
Exercise: Writing a node

Task: write the Read Image node starting from a node template

1 – locate and instantiate the Generic node
2 – use the node editor to re-create the Read Image node
   - add an input port called ‘filename’ of type string
   - bind an NEEntryWithFileBrowser widget and place it in the node
   - add the name ‘filename’ to the widget
   - add an output port called ‘image’ of type image
   - complete the node’s function
3 – save node in MyDefaultLib - Input

The original Read Image node uses Image.open
Image is imported in the file defining the node
In your node you will have to import Image

```
def doit(self, filename):
    import Image
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im)
```
Exercise: Looking at the source code

Task: study a source code of the node we saved in the user library

1 – navigate to HOME\mgltools\1.5.6\Vision\UserLibs\MyDefaultLib\Input
2 – right-click on MyReadImage.py and select open with IDLE
```python
class ReadImage(NetworkNode):
    """based on the Image.open function. Reads an image file
Input:  filename (string)
Output: Image"""

def __init__(self, name='Read Image', **kw):
    kw['name'] = name
    apply(NetworkNode.__init__, (self,), kw)

    self.inputPortsDescr.append(datatype='str', name='filename')
    self.outputPortsDescr.append(datatype='image', name='image')

    fileTypes = [('all', '*'), ('jpeg', '*.jpg'), ('tiff', '*.tif'),
                 ('png', '*.png'), ('bmp', '*.bmp')]

    self.widgetDescr['filename'] = {
        'class': 'WNEEntryWithFileBrowser', 'master': 'node',
        'filetypes': fileTypes, 'title': 'read image', 'width': 10,
        'labelCfg': {'text': 'image file:'}}

code = """def doit(self, filename):
if filename:
    im = Image.open(filename)
if im:
    self.outputData(image=im)
"""

    self.setFunction(code)
```
Exercise: User Panel

Task: create a panel that provides selected widgets from a network

1 – load the saved network “saveMPEG_net.py”
2 – create a user Panel using Edit -> Create user panel
   (name the panel FramesToMovie)
3 – used the widget menu to move the following widgets to the panel
   directory from NumberedName node
   MPEG file name from the Record MPEG movie node
4 – middle-click on widget in panel and drag to move it around
Notes: User Panel

The panels allows:
- running and stopping the execution
- showing/hiding the network
Exercise: command line execution

Task: run the network from the command line using Vision

1 – save the network with the panel as “saveMPEGPanel_net.py”
2 – start a DOS command
3 – In the DOS window type:
"c:\Program Files\MGLTools 1.5.6\vision.bat" ..\saveMPEGPanel_net.py

- Use the Tab key for automatic completion
- Note that the network comes up with the Vision GUI
Exercise: Network Execution

Task: run a network from the command line as a program

1 – In the DOS window type:
c:\Users\rctraining\saveMPEGPanel_net.py –help or –h

The help message display info about command line options including parameters that can be set from the command line.

2 – In the Dos window type:
c:\Users\rctraining\saveMPEGPanel_net.py

Only the parameter panels comes up
Exercise: endless loop

Task: build a network with an endless loop that will print the value of a dial if the value is positive

1 – create a new network using the button bar
2 – turn off immediate mode using the button bar
3 – locate and instantiate a while node and set the condition to 1
4 – locate and instantiate a pass node
5 – connect the output of while to the pass node
6 – right click on the pass node and select “show special ports”
7 – locate and instantiate a Dial node
8 – right click on the Dial node and select “show special ports”
9 – connect the “trigger” special port of Pass to the “run” special port of Dial
8 – locate and instantiate an If node and set the condition to “value > 0.0”
10 – connect the dial output to the value in put port of the If node
11 – locate and instantiate a print node
12 – connect the _if output port of the If node to the print node
13 – turn immediate mode back on
14 – save network as endlessPrint_net.py
Exercise: endless loop

Task: run the network in Vision

1 – click on the run network button in the tool bar
2 – make the dial positive to print to the shell or negative to stop printing
Exercise: endless loop

Special ports:
- Left side of the node (run) receives signal
- Right side (trigger) sends a signal after running

Allow to trigger Dial execution each Time Pass runs without actually Passing data from Pass to Dial

Task: run the while node

When the dial is < 0.0 the print node does not flash, when you move the dial to a positive value it flashes
Exercise: detached execution

Task: run the network in a separate process

1 – click on the run detached button in the tool bar
2 – make the dial positive to print to the shell or negative to stop printing
Notes: detached execution

Python 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)]
Type "help", "copyright", "credits" or "license" for more information.
>>> import socket
>>> s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> err = s.connect(('localhost', 50001))

>>> # retrieve value of Dial
>>> code = """val = Dial_4.inputPorts[0].widget.get()
...    clientSocket.send(str(val))""
>>> s.send(code)
>>> data = s.recv(1024)
>>> print data
0.11
>>>
Macro nodes

- Macro nodes represent a network as a single node in a parent network
- Data can be passed into the macro and come out of it
- Macros can be nested
- Macros can be added to libraries of Vision nodes

Task: load the MolKit library
Macro nodes

Task: use a Lines Macro to display a molecule

1 – locate and instantiate a Read Molecule node
2 – locate and instantiate a Lines Macro node

The macro output data of type geometry defined in the 3D Visualization library which is pulled in automatically

3 – locate and instantiate a Viewer node

A window associate to the Viewer node (black window) is created. It can be used to display 3D geometry.

4 – connect the Read molecule output to the macro input
5 – connect the macro output to the Viewer
6 – read the molecule TutorialData/2plv.pdb from the desktop
7 – click in the 3D viewer (black window) and type the letters r n c d
8 – use middle mouse button to rotate the molecule
Macro nodes

Task: open the macro node

1 – double click on the macro node

A new network called “Lines Macro” is available. Click on the tab to view the macro network.

2 – click on all expanded nodes to collapse them (hide widgets)

Macro networks have 2 special nodes “input Ports” and “output Ports”. These nodes allow data to enter and exit the macro.

Task: close the macro node

1 – double click on the “input Port” node
Macro nodes

Task: create a new macro node

1 – Edit -> create macro or Ctrl-m
2 – name the macro “MyMacro”

You are automatically taken inside the macro network

3 – locate and instantiate a pass node inside the macro
4 – connect the first output port of the Macro input Port node to the input of pass
5 – connect the output of pass to the first input port of the Macro output Port node
Macro nodes

Task: create a new macro node

In the parent network:
1 – instantiate a Dial node and send the value into the macro
2 – instantiate a print node and connect the output of the macro
Day 3 Overview

- Introduction to Pmv
- Building a viral capsids
- Running PMV commands in a Vision network
- Creating new PMV commands
The installer placed 3 icons on the desktop. To start Vision double click on the Vision Icon.
Exercise: start Pmv

Task: load a protein into PMV

1 – start Pmv
2 – go to page 8 of the PMV tutorial and follow the procedure to load the molecule
   the molecule hsg1.pdbqs is located in Desktop/TutorialData
Exercise: Pmv mouse

Task: learn PMV mouse bindings

1 – go to page 10 of the PMV tutorial and read section 3. Mouse Bindings

<table>
<thead>
<tr>
<th>Button</th>
<th>Mod</th>
<th>Left</th>
<th>Middle</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>None</td>
<td><strong>Pick</strong></td>
<td><strong>Rotate</strong></td>
<td>Translate left/right (X) and up/down (Y)</td>
</tr>
<tr>
<td></td>
<td>Shift</td>
<td><strong>Select</strong></td>
<td><strong>Scale or Zoom</strong></td>
<td>Translate in/out (Z)</td>
</tr>
</tbody>
</table>
**Exercise: bindingsPmv key bindings**

**Task: learn PMV viewer keystrokes**

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td><strong>Reset</strong> view</td>
</tr>
<tr>
<td>N</td>
<td><strong>Normalize</strong> – scale so all visible molecules fit in the Viewer</td>
</tr>
<tr>
<td>C</td>
<td><strong>Center</strong> on the center of gravity of all the molecules</td>
</tr>
<tr>
<td>D</td>
<td>Toggle on/off <strong>Depth-cueing</strong> (blends molecule into background farther away)</td>
</tr>
<tr>
<td>T</td>
<td>Toggle between <strong>transform root</strong> (i.e. scene) and transform the Viewer’s current object</td>
</tr>
<tr>
<td>A</td>
<td>Toggle on/off <strong>Auto Depth-cueing</strong> (turn on/off setting bounding box of fog to the current object)</td>
</tr>
<tr>
<td>L</td>
<td>Toggle on/off <strong>Opengl Lighting</strong> (turns on/off photorealistic lighting)</td>
</tr>
</tbody>
</table>
Exercise: Pmv dashboard

Task: learn undock/dock dashboard

1 – click on the dock/undock dashboard button in Pmv/s toolbar

- The dashboard detached and appears in its own window

- The toolbar button is depressed while the dashboard is undocked
- To re-dock the dashboard click on the button again
Notes: dashboard
Notes: dashboard

- show/hide molecule
- display lines
- display spheres
- display sticks & balls
- display ribbon
- display molecular surface

representation commands

color commands

color commands
Notes: dashboard

display spheres
display sticks & balls
display ribbon
display molecular surface

show/hide molecule
select

representation commands
color commands
Exercise: Pmv dashboard

Task: Display chain A as CPK

1 – left-click in dashboard on the circle at the intersection of chain ‘A’ and CPK
Exercise: dashboard

Task: display chain B as scaled down CPK

1 – right-click in dashboard on the circle at the intersection of chain ‘B’ and CPK

The option panel of the display CPK command appears

2 – Set scale Factor to 0.3
3 – click OK
Exercise: dashboard

Task: un display CPK spheres for chain
un display lines for chain A and B
display ribbon for chain B

1 – left-click at the intersection of chain ‘B’ and CPK
2 – left-click at the intersection of chain ‘hsg1’ and Lines
3 – left-click at the intersection of chain ‘B’ and ‘Rib.’
Exercise: dashboard

Task: color hsg1 by atom type

1 – left-click at the intersection of ‘hsg1’ and Atom

- Coloring is applied to all representations
- The ribbon is blue because the N atom info is used
- The color button NOT check buttons (i.e. no state)
Exercise: dashboard

Task: color chain B by secondary structure element

1 – right-click at the intersection of chain ‘B’ and Sec.Str.

- A geometry chooser is displayed allowing to select the representations that will be colored

2 – select secondarystructure
3 – click OK
Exercise: dashboard

Task: display residue Arg8 in chain B as sticks and balls and color the sticks and balls by DG

1 – expand chain B
2 – left-click at ARG8 and S&B
3 – right click at ARG8 and DG
4 – select ‘sticks’ and ‘balls’
5 – click OK
Exercise: Viral capsid

Task: build the viral capsid of Polio Virus

1 – right-click on hsg1 in the dashboard and select ‘Delete’
2 – right-click on PMV Molecules and load Desktop/TutorialData/2plv.pdb
3 – start Vision by clicking on the Vision button in the toolbar

The Pmv node library has Vision nodes specific to PMV
Exercise: Viral capsid

Task: build the viral capsid of Polio Virus

4 – load the symmetry server library of Vision nodes
Exercise: Viral capsid

Task: build the viral capsid of Polio Virus

5 – build the following network

6 – run the network
Exercise: Viral capsid

Task: change the representation from lines to a coarse molecular surface

1 – using the dashboard un-display the lines for 2plv
2 – execute the command Compute -> Coarse Molecular Surface from PMV menus using default parameters
3 - place cursor on 3D viewer window and type ‘r’, ‘n’, ‘c’

The compute coarse molecular surface command is implemented as a Vision networks that is loaded the first time the command runs
Exercise: Viral capsid

Task: color the surface by chain

1 – using the dashboard color 2plv by chains
Exercise: Viral capsid

Task: color the surface depth in the capsid
Exercise: Viral capsid

Task: expand the capsid by translating each 5-fold copy along its 5-fold axis

1 – double click on the Icosahedral1 node to expand the macro
2 – double click in the 5-fold node to display its parameter panel
3 – highlight the 5 fold axis and type Ctrl-c to copy the values 0.000 0.525 0.851
4 – locate and instantiate a translate node
5 – double click on the translate node to display its parameter panel
6 – highlight the translation vector and type Ctrl-v to paste 0.000 0.525 0.851
7 – delete the connection between the 5-fold and the 3-fold nodes
8 – insert the translate node between the 5-fold and 3-fold nodes
9 – modify the translation vector in the parameter panel of the translation node
   (right click on the thumbwheel to increase sensitivity to 10 for better results)
10 – witness the capsid expand
Exercise: Viral capsid

Task: expand the capsid by translating each 5-fold copy along its 5-fold axis
Exercise: Viral capsid

Task: add range and iterate node to automate expansion
Exercise: Viral capsid

Task: add range and iterate node to automate expansion

1 – unbind “vector length” widget in the parameter panel of the translate
2 – add range node from 0 to 100 in steps of 5
3 – add iterate node to iterate over range output
4 – feed value from iteration into “vector length” port
Exercise: Viral capsid

**Task: build the TMV capsid**

1. delete the 2plv molecule
2. load the Desktop/TutorialData/tmv/2tmv.pdb protein
3. hide the lines and display a coarse molecular surface
4. instantiate Pmv Viewer node
5. instantiate a Choose Geom node
6. connect the viewer to the choose Geom (the combo box will be populated)
7. select root | 2tmv | CoarseMolecularSurface in the combo box
8. instantiate a Helix node
9. Instantiate a Set Instances node (use the one from the 3D vis library)
10. connect Choose Geom and Helix outputs to Set Instances
11. double click on helic to display its parameter panel
12. set copies to 50
13. start changing the angle of the helix (values around 20 are good)
14. interact with the molecule in the viewer
15. start changing the rise of the helix
16. replace the lines by a coarse molecular surface
17. set the rise to 1.43 and the angle to 22.04
18. color by Instances (using the dashboard)
Exercise: Viral capsid

Task: build the TMV capsid
Exercise: use Pmv cmds in Vision

Task: build a network to run the computeMSMS cmd

1. Run the MSMS command in Pmv first to see possible arguments
2. In Pmv delete 2tmv and load tf_1.pdb
3. In Pmv use the Compute -> molecular surface command
   - Note the arguments that are possible include: surface name, probe radius, density, per molecule, etc...
4. Click Cancel
5. In Vision: create a new network
6. Instantiate a Pmv node and select computeMSMS for the cmd:
7. Instantiate a Run_command node and display its parameter panel
8. Connect the cmd port of the Pmv node to the Run_command (note what happens)
9. Instantiate a tf_1 node and connect the molecule to the new input port on Run computeMSMS
10. The surface gets computed after you connect
11. Vary the density probe dial to higher values
Exercise: use Pmv cmds in Vision

Task: modify the network to compute a surface for each amino acid

1 – instantiate a select MolFrag node to get a list of residues
2 – send the tf_1 molecule into this node and select Residue for level
3 – add an iterate node to iterate over the list of residues output by this node
4 – connect the oneltem output port of iterate to the compute MSMS node

Surface is computed for the whole molecule but only the patch for the last residue is displayed
Exercise: use Pmv cmds in Vision

Task: modify the network to compute a surface for each amino acid

5 – un-check the “perMol” check button in the computeMSMS parameter panel

Surface is now computed for the set of atoms in the residue

6 – run the iterate node and watch the surface walk along the chain
Exercise: scale CPK radii by charge

Task: display CPK spheres scales y the atomic charge

1 – delete all molecules and load indinavir
2 – un-display lines, display CPK and color by atom types
3 – create a new vision network
4 – instantiate an indinavir node, Select MolFrag and Extract Atom Property
5 – send the molecule into select MolFrag with level set to Atom
6 – send the resulting AtomSet into Extract Atom Property and set prop. Name to ‘charge’
7 – instantiate a Dial node and set it to 3.0 and an op2 node
8 – in op2 select operator to be ‘mul’ and check apply to elements
9 – connect the list of charges to the first input port of op2 and the dial to the second
10 – instantiate a Pmv Viewer node and a Choose Geom
11 – connect the viewer output to the Choose Geom node and select root|indinavir|cpk
12 – instantiate a Call method node and set the signature to “Set radii <enter>”
13 – connect the output of Choose Geom to the first port of call method
14 – connect the list of scaled charges coming out of mul to the second port
Exercise: scale CPK radii by charge

Task: display CPK spheres scales by the atomic charge
Exercise: new PMV command

Task: run the 3res.py script from the command line using Pmv

1 – save the network with the panel as “saveMPEGPanel_net.py”
2 – start a DOS command
3 – In the DOS window type:
   "c:\Program Files\MGLTools 1.5.6\pmv.bat" 3res.py 1BVN_4.0_1.0_-17.0.txt

- Use the Tab key for automatic completion
- Note that the network comes up with the Vision GUI
Exercise: new PMV command

Modify the parameters of ShowSolutions to browse results
Day 4 Overview
Day 5 Overview

• The Volume library
• The vizlib library
• The matplotlib library
• Student problems
• Wrap up
Working with volumetric data
Working with volumetric data

Boolean operation can be performed on masks to create complex masks. Here a spherical slab mask is created by XOR’ing 2 spheres.
2D plotting

http://mgltools.scripps.edu/packages/vision/matplotlib/vision-networks-for-matplotlib
3D Visualization

- Bind Geometry To Molecule
- Set Cartoon Outlines
- Set Antialiasing
- Set Background Color
- Set Camera Size
- Video Recorder
- Render Large Image
- Spin - Bounce - Oscillate
3D Visualization
3D Visualization: GUI overview

- **Bind Mouse to transform**
- **Operations assigned to mouse buttons** (changes with modifiers)
- **When checked 3D Xforms apply to root**
- **Geometry objects hierarchy**
- **Reset Xform of current object**
- **Root geom parent of all geometries and current object**
- **Master geom for all geoms of a given molecule**
- **Geoms created by Pmv cmds for that molecule**
- **Set rotation center to center of the scene**
- **Fit the scene in the view**
3D Visualization: GUI overview

Select property panel to show

Object property panel
3D Visualization: GUI overview

Camera property panel

Object | Camera | Clip | Light | Bookmarks
--- | --- | --- | --- | ---
Bounding Box
Background Color
Auto Depthcue
Video Recorder
Projection
Scene Antialiasing
Cartoon Outlines
Ambient Occlusion
Selection Settings

- Depthcueing
- Thumbnail
- Overall Lighting

clipZ Fog
3D Visualization: GUI overview

Clipping planes property panel

Lights property panel
2D plotting

http://mgltools.scripps.edu/packages/vision/matplotlib/vision-networks-for-matplotlib
Matplotlib in DejaVu