



Visualization for Computational Science

Course Handouts for Lect. 15, March 13, 2002

Wednesday, 3/13/02 Detail

3/13/02	LEC15	<p>Module 2 -- Data Sources and Structures Data Structures 3 (Redmond) Scientific Data Conversion Example 1: Conversion of PDB Molecule Data to VisAD Example 2: Conversion of Analyze FMRI Data to VisAD Use of VisAD spreadsheet for general conversion (ALL LINKS TO SUBSEQUENT TOPICS CAN BE FOUND AT) http://courses.engr.wisc.edu/ecow/get/need/602/cramergeor/links/conversion/</p> <p>General Background and Utilities NetCDF Conversion MINC (NetCDF) FMRI Conversion Analyze FMRI Conversion (.img/.hdr) HDF4 and HDF5 Conversion (incl NetCDF) PDB (Molecular) Conversion</p>
---------	-------	--

DNA Molecule Data stripped to Atom Definitions Only:

```

-----
ATOM      1  P   UNK      1      5.973  -6.848  27.430  1.00  0.00
ATOM      2  O1P UNK      1      6.257  -8.279  27.210  1.00  0.00
ATOM      3  O2P UNK      1      6.791  -5.896  26.640  1.00  0.00
ATOM      4  O5' UNK      1      4.427  -6.552  27.180  1.00  0.00
ATOM      5  C5' UNK      1      3.455  -7.076  28.120  1.00  0.00
ATOM      6  C4' UNK      1      2.183  -7.444  27.390  1.00  0.00
ATOM      7  O4' UNK      1      1.397  -6.228  27.180  1.00  0.00
ATOM      8  C1' UNK      1      1.462  -5.844  25.820  1.00  0.00
ATOM      9  C2' UNK      1      2.419  -6.792  25.110  1.00  0.00
ATOM     10  C3' UNK      1      2.350  -8.030  25.990  1.00  0.00
ATOM     11  O3' UNK      1      1.231  -8.821  25.600  1.00  0.00
ATOM     12  N9   UNK      1      1.883  -4.418  25.770  1.00  0.00
ATOM     13  C8   UNK      1      3.150  -3.905  25.850  1.00  0.00
ATOM     14  N7   UNK      1      3.191  -2.608  25.780  1.00  0.00
ATOM     15  C5   UNK      1      1.865  -2.243  25.630  1.00  0.00
. . .
ATOM     645 C2' UNK      1     -5.504   4.112  25.590  1.00  0.00
ATOM     646 C3' UNK      1     -6.702   4.430  24.710  1.00  0.00
ATOM     647 O3' UNK      1     -7.800   3.609  25.100  1.00  0.00
ATOM     648 N1   UNK      1     -3.411   2.869  24.930  1.00  0.00
ATOM     649 C6   UNK      1     -2.676   4.024  24.830  1.00  0.00
ATOM     650 C5M UNK      1     -0.479   5.224  24.770  1.00  0.00
ATOM     651 C5   UNK      1     -1.329   3.995  24.880  1.00  0.00
ATOM     652 O4   UNK      1      0.608   2.631  25.080  1.00  0.00
ATOM     653 C4   UNK      1     -0.622   2.746  25.030  1.00  0.00
ATOM     654 N3   UNK      1     -1.438   1.643  25.120  1.00  0.00
ATOM     655 O2   UNK      1     -3.459   0.600  25.170  1.00  0.00
ATOM     656 C2   UNK      1     -2.818   1.633  25.080  1.00  0.00
    
```

PDB to VisAD conversion and display program for DNA Molecule (tricky):

```
from visad.python.JPythonMethods import *
from visad import *
from visad.util import Delay
import string
from subs import *
import math

# number of atoms in input file
# determined beforehand
# there should be a better way to derive through data read
n_atoms = 656

# make type for x,y,z positions of atoms of specified id
# independent variable is index id_num
atom_type = makeType("(index -> (x_in_A, y_in_A, z_in_A, id))");
atom_range = atom_type.getRange()

# Set range for atoms as total number read in
atom_dom = makeDomain("index", 1, n_atoms, n_atoms)

# open dna molecule location array with identifiers
# molecule=open("dna_molecule.txt","r")
molecule=open("c:\\transfer\\vis course\\pdb files\\raw_molecule.txt","r")

# string positions of x, y and z atom locations
# determined by looking at incoming data set
# there should be a better way to read data with separators
xpos=5
ypos=6
zpos=7
idpos = 2

# create a temporary value list to put atom coordinates in
dna_x=[]
dna_y=[]
dna_z=[]
dna_id=[]

print "start"
# loop from 0 to number of atoms - 1
for i in range(n_atoms):

    # create a temporary expandable list to put string data in
    values = []
    # get one line at a time
    values=string.split(molecule.readline())
    # print values

    # convert the x, y and z coordinates for atom "i"
    # append x, y and z values to value lists
    temp=float(values[xpos])
    dna_x.append(temp)

    temp=float(values[ypos])
    dna_y.append(temp)

    temp=float(values[zpos])
    dna_z.append(temp)

    # set id to value based on character found
    id_str = values[idpos][0]
    if id_str == "O":
        id_type = 0
    elif id_str == "N":
        id_type = 1
    elif id_str == "C":
        id_type = 2
    elif id_str == "P":
```

```
        id_type = 3
    else:
        id_type = -1
    # append id to id list
    dna_id.append(id_type)

# set up locations object based on index mapping
locs = FlatField(atom_type, atom_dom)

# fill locations object with location and id data
locs.setSamples([dna_x, dna_y, dna_z, dna_id])

# map display to accept atom locations and use shape widget
maps = makeMaps(atom_range[0], "x", atom_range[1], "y", atom_range[2], "z",
                atom_range[3], "shape")

# use equalized ranges based on knowledge of original data
# so aspect ratio will look correct
maps[0].setRange(-30,30)
maps[1].setRange(-30,30)
maps[2].setRange(-30, 30)

# display with preset mappings
plot (locs, maps)

#tricks to build balls for molecules
control = maps[3].getControl()
normals = [0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0,
           0.0, 0.0, 1.0, 0.0, 1.0, 0.0, -1.0, 0.0, 0.0,
           0.0, 0.0, 1.0, -1.0, 0.0, 0.0, 0.0, -1.0, 0.0,
           0.0, 0.0, 1.0, 0.0, -1.0, 0.0, 1.0, 0.0, 0.0,
           0.0, 0.0, -1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
           0.0, 0.0, -1.0, 0.0, 1.0, 0.0, -1.0, 0.0, 0.0,
           0.0, 0.0, -1.0, -1.0, 0.0, 0.0, 0.0, -1.0, 0.0,
           0.0, 0.0, -1.0, 0.0, -1.0, 0.0, 1.0, 0.0, 0.0]
coords = []
for i in range(72):
    coords.append(0.05 * normals[i])

cols = [1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.25, 0.75, 0.75, 0.5, 0.5, 0.2]

atoms = []
for i in range(4):
    # set ball shapes into atom indexes
    atoms.append(VisADTriangleArray())
    atoms[i].vertexCount = 24
    atoms[i].coordinates = coords
    atoms[i].normals = normals
    colors = []
    # set colors for the different ids
    for j in range(72):
        colors.append(int(255.0 * cols[3*i + j%3]))
    atoms[i].colors = colors

# put in shapes for atoms in display
control.setShapeSet(Integer1DSet(4))
control.setShapes(atoms)

# close the file for other uses
molecule.close()

print "done"
```

```

VisAD/Python Conversion Program for binary FRMI Analyze/.img data to VisAD
-----
from visad.python.JPythonMethods import *
from visad.python.JPythonMethods import *
from visad.python.JPythonMethods import *
from visad import *
import math
from visad.data.visad import VisADForm
import struct
# n time steps of an n x n x n grid
nx=64
ny=64
nz=36
nt=110
# units/lengths of grid dimensions per information from header
lx=3.125*63
ly=3.125*63
lz=4.5*35
lt=3.00*nt
# get two frames per boxcar group of 10
step_size=5
# start at 1st then 6th frames to
start_time=0
# open file for binary read
fin=open("c:\\fmri_visual_stim_EPI_sag.img","rb")
reduce_fmri_file="C:\\Transfer\\Vis Course\\MRI
files\\Mystery_data_2\\fmri_reduced.vad"
full_fmri_file="C:\\Transfer\\Vis Course\\MRI files\\Mystery_data_2\\fmri_full.vad"
# set sizes
nvol=nx*ny*nz
# 2 bytes per binary value
nbytes=nvol*2
# set up file for VisAD formatted output
fout = VisADForm()
# make the type (i.e., schema) for the time sequence of grids
ftype = makeType(" (time_in_sec -> ( (x_fmri_in_mm, y_fmri_in_mm, z_fmri_in_mm) ->
value_fmri_16bit) )")
# make the (0,...,total time) sampling for the time domain
fdom = makeDomain("time_in_sec", 0, lt, nt+1)
# set up resample domain for fewer frames
fdom2 = makeDomain("time_in_sec", start_time, lt, nt/step_size+1)
# make the (1,...,n) x (1,...,n) x (1,...,n) sampling for the
# grid domain but in measured units
gdom = makeDomain("(x_fmri_in_mm, y_fmri_in_mm, z_fmri_in_mm)", 0, lx, nx, 0, ly,
ny, 0, lz, nz)
# create the time sequence data object (a VisAD FieldImpl)
seq = FieldImpl(ftype, fdom)
# loop for each time step in the sequence
for i in range(0, nt):
    # track progress
    print i
    # create an array to hold the grid values for time step i
    v = []
    # binary read a full frame of bytes
    g=fin.read(nbytes)
    # convert all binary bytes to short integer tuples at once
    v=struct.unpack("147456h",g)
    # create a grid field as the i-th sample of the time sequence
    seq.setSample(i, field(gdom, "value_fmri_16bit", v) )
# resample the result down to manageable number of frames
seq_reduced=resample(seq,fdom2)
# plot the time sequence of grids
plot(seq)
# save the full file
fout.save(full_fmri_file, seq, 1)
# save the reduced file
fout.save(reduce_fmri_file, seq_reduced, 1)

```