Lezione 2

Bioinformatica

Mauro Ceccanti[‡] e Alberto Paoluzzi[†]

[†] Dip. Informatica e Automazione – Università “Roma Tre”
[‡] Dip. Medicina Clinica – Università “La Sapienza”

Protein structure modeling

- Molecule models
- Drawing small molecules from PDB files
- Structure chaining
- Geometric modeling of proteins

Sommario

Chemical Component Dictionary (CCD)

The CCD is as an external reference file describing all residue and small molecule components found in PDB entries

Search and browse the CCD using resources such as PDBeChem and Ligand Expo.

- contains detailed chemical descriptions for standard and modified amino acids / nucleotides, small molecule ligands, and solvent molecules
- includes stereochemical assignments, aromatic bond assignments, idealized coordinates, chemical descriptors (SMILES & InChI), and systematic chemical names.
- is organized by the 3-character alphanumeric code that PDB assigns to each chemical component
- is updated with each weekly PDB release.

Sommario

Drawing small molecules from PDB files

Include the Bio.PDB module from Biopython (and read this tutorial :o)

```
from Bio.PDB import *

def myprint (string):
    print "\n" + string + " ->", eval(string)
```

The `myprint()` function is used to show both an expression and the value produced by its evaluation

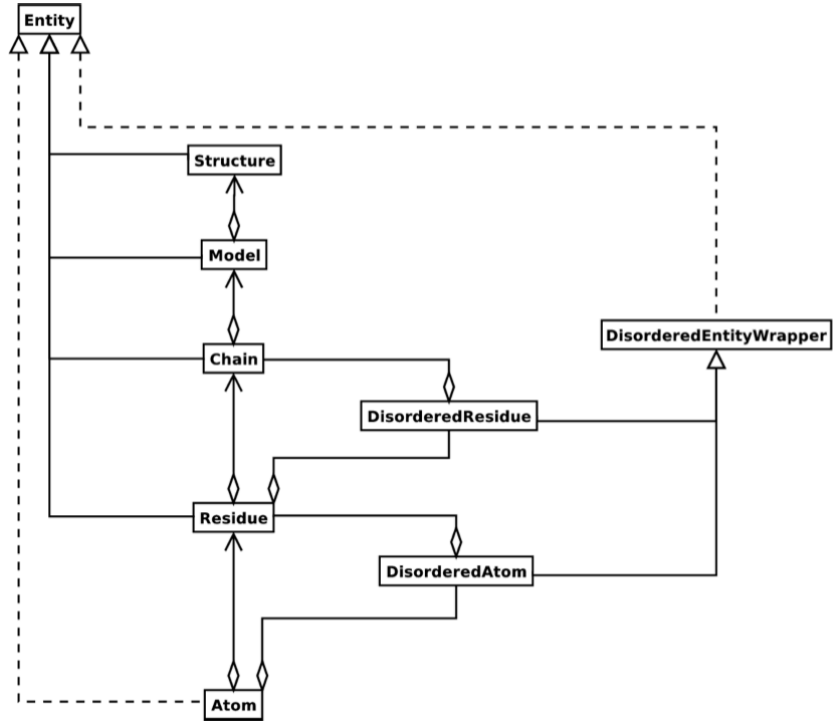
Complete source of the following programming example can be found in bio-pdb-example

Drawing small molecules from PDB files

The overall layout of a Structure object

The Bio.PDB package follows the Structure/Model/Chain/Residue/Atom architecture

- A structure consists of models
- A model consists of chains
- A chain consists of residues
- A residue consists of atoms



UML diagram of SMCRA architecture of the Structure object. Full lines with diamonds denote aggregation, full lines with arrows denote composition, full lines with triangles denote inheritance and dashed lines with triangles denote interface realization

Get the input PDB file

One (of many) possible path ...

Search and browse the Chemical Component Dictionary (CCD) using resources such as PDBeChem

- Ligand Dictionary
- Ligand Expo

1. look at the short Tutorial
2. and
   - 2.1 search
   - 2.2 browse
   - 2.3 download

The input file BTC.pdb

```
HEADER    NONAME 22-Aug-09
TITLE     Produced by PDBeChem
COMPND    BTC
AUTHOR    EBI-PDBe Generated
REVDAT   1  22-Aug-09     0
ATOM      1  N   BTC     0       1.585   0.483  -0.081  1.00 20.00           N+0
ATOM      2  CA  BTC     0       0.141   0.450   0.186  1.00 20.00           C+0
ATOM      3  CB  BTC     0      -0.533  -0.530  -0.774  1.00 20.00           C+0
ATOM      4  SG  BTC     0      -0.247   0.004  -2.484  1.00 20.00           S+0
ATOM      5  C   BTC     0      -0.095   0.006   1.606  1.00 20.00           C+0
ATOM      6  O   BTC     0       0.685  -0.742   2.143  1.00 20.00           O+0
ATOM      7  OXT BTC     0      -1.174   0.443   2.275  1.00 20.00           O+0
ATOM      8  H   BTC     0       1.693   0.682  -1.065  1.00 20.00           H+0
ATOM      9  H2  BTC     0       1.928  -0.454   0.063  1.00 20.00           H+0
ATOM     10  HA  BTC     0      -0.277   1.446   0.042  1.00 20.00           H+0
ATOM     11  HB2 BTC     0      -0.114  -1.526  -0.630  1.00 20.00           H+0
ATOM     12  HB3 BTC     0      -1.604  -0.554  -0.575  1.00 20.00           H+0
ATOM     13  HG  BTC     0      -0.904  -0.965  -3.145  1.00 20.00           H+0
ATOM     14  HXT BTC     0      -1.326   0.158   3.186  1.00 20.00           H+0
CONECT    1    2    8    9
CONECT    2    5    3   10    1
CONECT    3    2   11   12    4
CONECT    4    3   13
CONECT    5    2    6    7
CONECT    6    5
CONECT    7    5   14
CONECT    8    1
CONECT    9    1
CONECT   10    2
CONECT   11    3
CONECT   12    3
CONECT   13    4
CONECT   14    7
END
```

Drawing small molecules from PDB files

BTC (cysteine) is a small molecule, with only 1 model, 1 “chain”, 1 “residue”

`chain = model[' ']`, since there is an empty field for chain code in the file `BTC.pdb`

```
parser=PDBParser()
structure=parser.get_structure('cysteine', 'BTC.pdb')
myprint("structure")

model = structure[0]
myprint("model")

chain = model[' ']
myprint("chain")

residue = chain[0]
myprint("residue")
```

```
structure -> <Structure id=cysteine>

model -> <Model id=0>

chain -> <Chain id= >

residue -> <Residue BTC het=  resseq=0 icode= >
```

Drawing small molecules from PDB files

Get the ordered list of atom names in structure

```
for atom in residue:
        print atom.get_serial_number(), atom.get_coord()

myprint("[atom.get_name() for atom in residue]")
```

equivalently:

```
myprint("[atom.get_name() for atom in structure[0][' '][0]]")
```

```
[atom.get_name() for atom in residue] -> ['N', 'CA', 'CB', 'SG', 'C', 'O', 'OXT
', 'H', 'H2', 'HA', 'HB2', 'HB3', 'HG', 'HXT']
```

Drawing small molecules from PDB files

`CONECT` records are not parsed by `Bio.PDB` module — we go doing :o)

```
def getPdbConnect (filename):
    myfile = open(filename,'r')
    for record in myfile:
        terms = record.split()
        if terms[0] == "CONECT": print terms
    myfile.close()

## units = Angstrom: 1 x 10^(-10)


myprint("getPdbConnect('BTC.pdb')")
```

```
getPdbConnect('BTC.pdb') -> ['CONECT', '1', '2', '8', '9']
['CONECT', '2', '5', '3', '10', '1']
['CONECT', '3', '2', '11', '12', '4']
['CONECT', '4', '3', '13']
['CONECT', '5', '2', '6', '7']
['CONECT', '6', '5']
['CONECT', '7', '5', '14']
['CONECT', '8', '1']
['CONECT', '9', '1']
['CONECT', '10', '2']
['CONECT', '11', '3']
['CONECT', '12', '3']
['CONECT', '13', '4']
['CONECT', '14', '7']
None
```

Drawing small molecules from PDB files

Select the useful information

```
from pyplasm import *
def getPdbConnect (filename):
    myfile = open(filename,'r')
    for record in myfile:
        terms = record.split()
        if terms[0] == "CONECT":
            terms = AA(eval)(terms[1:])
            print terms
    myfile.close()

myprint("getPdbConnect('BTC.pdb')")
```

```
getPdbConnect('BTC.pdb') -> [1, 2, 8, 9]
[2, 5, 3, 10, 1]
[3, 2, 11, 12, 4]
[4, 3, 13]
[5, 2, 6, 7]
[6, 5]
[7, 5, 14]
[8, 1]
[9, 1]
[10, 2]
[11, 3]
[12, 3]
[13, 4]
[14, 7]
None
```

Drawing small molecules from PDB files

Compute the list of adjacent nodes (`terms`) to each `node`

```
from pyplasm import *
def getPdbConnect (filename):
    myfile = open(filename,'r')
    for record in myfile:
        terms = record.split()
        if terms[0] == "CONECT":
            node,terms = eval(terms[1]),AA(eval)(terms[2:])
            print node,terms
    myfile.close()

myprint("getPdbConnect('BTC.pdb')")
```

```
getPdbConnect('BTC.pdb') -> 1 [2, 8, 9]
2 [5, 3, 10, 1]
3 [2, 11, 12, 4]
4 [3, 13]
5 [2, 6, 7]
6 [5]
7 [5, 14]
8 [1]
9 [1]
10 [2]
11 [3]
12 [3]
13 [4]
14 [7]
None
```

Drawing small molecules from PDB files

Compute the directed arcs outgoing from each node

```
from pyplasm import *
def getPdbConnect (filename):
    myfile = open(filename,'r')
    for record in myfile:
        terms = record.split()
        if terms[0] == "CONECT":
            arcs = DISTL([ eval(terms[1]),AA(eval)(terms[2:]) ])
            print arcs
    myfile.close()

myprint("getPdbConnect('BTC.pdb')")
```

```
getPdbConnect('BTC.pdb') -> [[1, 2], [1, 8], [1, 9]]
[[2, 5], [2, 3], [2, 10], [2, 1]]
[[3, 2], [3, 11], [3, 12], [3, 4]]
[[4, 3], [4, 13]]
[[5, 2], [5, 6], [5, 7]]
[[6, 5]]
[[7, 5], [7, 14]]
[[8, 1]]
[[9, 1]]
[[10, 2]]
[[11, 3]]
[[12, 3]]
[[13, 4]]
[[14, 7]]
None
```

Drawing small molecules from PDB files

Compute the undirected arcs

```
from pyplasm import *
def getPdbConnect (filename):
    myfile = open(filename,'r')
    for record in myfile:
        terms = record.split()
        if terms[0] == "CONECT":
            arcs = DISTL([ eval(terms[1]),AA(eval)(terms[2:]) ])
            arcs = [arc for arc in arcs if arc[0] < arc[1]]
            print arcs
    myfile.close()

myprint("getPdbConnect('BTC.pdb')")
```

```
getPdbConnect('BTC.pdb') -> [[1, 2], [1, 8], [1, 9]]
[[2, 5], [2, 3], [2, 10]]
[[3, 11], [3, 12], [3, 4]]
[[4, 13]]
[[5, 6], [5, 7]]
[]
[[7, 14]]
[]
[]
[]
[]
[]
[]
[]
None
```

Drawing small molecules from PDB files

Finally returns the list of undirected arcs

```
from pyplasm import *
def getPDBconnect (filename):
    myfile = open(filename,'r')
    arcs = []
    for record in myfile:
        terms = record.split()
        if terms[0] == "CONECT":
            pairs = DISTL([ eval(terms[1]), AA(eval)(terms[2:]) ])
            arcs += [arc for arc in pairs if arc[0] < arc[1]]
    myfile.close()
    return arcs

myprint("getPDBconnect('BTC.pdb')")
```

```
getPDBconnect('BTC.pdb') -> [[1, 2], [1, 8], [1, 9], [2, 5], [2, 3], [2, 10], [
3, 11], [3, 12], [3, 4], [4, 13], [5, 6], [5, 7], [7, 14]]
```

Drawing small molecules from PDB files

Extraction of atom coordinates (3D points), as a list of array

```
myprint("[atom.get_coord() for atom in residue]")
```

```
[atom.get_coord() for atom in residue] -> [array([ 1.58500004,  0.48300001, -0.
081     ], dtype=float32), array([ 0.141     ,  0.44999999,  0.186     ], dtype
=float32), array([-0.53299999, -0.52999997, -0.77399999], dtype=float32), array
([-0.24699999,  0.004     , -2.48399997], dtype=float32), array([-0.095     ,
0.006     ,  1.60599995], dtype=float32), array([ 0.685     , -0.74199998,  2.1
4299989], dtype=float32), array([-1.17400002,  0.44299999,  2.2750001 ], dtype=
float32), array([ 1.69299996,  0.68199998, -1.06500006], dtype=float32), array(
[ 1.92799997, -0.454     ,  0.063     ], dtype=float32), array([-0.27700001,  1
.44599998,  0.042     ], dtype=float32), array([-0.114     , -1.52600002, -0.63
     ], dtype=float32), array([-1.60399997, -0.55400002, -0.57499999], dtype=fl
oat32), array([-0.90399998, -0.96499997, -3.14499998], dtype=float32), array([-
1.32599998,  0.15800001,  3.18600011], dtype=float32)]
```

Drawing small molecules from PDB files

Extraction of atom coordinates (3D points), as a list of lists

```
myprint("[atom.get_coord().tolist() for atom in residue]")
```

```
[atom.get_coord().tolist() for atom in residue] -> [[1.5850000381469727, 0.4830
000102519989, -0.081000000238418579], [0.14100000262260437, 0.44999998807907104
, 0.18600000441074371], [-0.53299999237060547, -0.52999997138977051, -0.7739999
8903274536], [-0.24699999392032623, 0.0040000001899898052, -2.4839999675750732]
, [-0.094999998807907104, 0.0060000000521540642, 1.6059999465942383], [0.685000
00238418579, -0.74199998378753662, 2.1429998874664307], [-1.1740000247955322, 0
.44299998879432678, 2.2750000953674316], [1.6929999589920044, 0.681999981403350
83, -1.065000057220459], [1.9279999732971191, -0.45399999618530273, 0.063000001
013278961], [-0.27700001001358032, 1.4459999799728394, 0.041999999433755875], [
-0.11400000005960464, -1.5260000228881836, -0.62999999523162842], [-1.603999972
3434448, -0.55400002002716064, -0.57499998807907104], [-0.90399998426437378, -0
.9649999737739563, -3.1449999809265137], [-1.3259999752044678, 0.15800000727176
666, 3.1860001087188721]]
```

Drawing small molecules from PDB files

Prepare the graph data

```
def graph (filename):
    parser=PDBParser()
    structure=parser.get_structure('molecule', filename)
    model = structure[0]
    chain = model[' ']
    residue = chain[0]

    nodes = [atom.get_coord().tolist() for atom in residue]
    edges = getPDBconnect('BTC.pdb')
    return nodes,edges

myprint("graph('BTC.pdb')")
```

```
graph('BTC.pdb') -> ([[1.5850000381469727, 0.4830000102519989, -0.0810000002384
18579], [0.14100000262260437, 0.44999998807907104, 0.18600000441074371], [-0.53
299999237060547, -0.52999997138977051, -0.77399998903274536], [-0.2469999939203
2623, 0.0040000001899898052, -2.4839999675750732], [-0.094999998807907104, 0.00
60000000521540642, 1.6059999465942383], [0.68500000238418579, -0.74199998378753
662, 2.1429998874664307], [-1.1740000247955322, 0.442999988879432678, 2.27500009
53674316], [1.6929999589920044, 0.68199998140335083, -1.065000057220459], [1.92
79999732971191, -0.45399999618530273, 0.063000001013278961], [-0.27700001001358
032, 1.4459999799728394, 0.041999999433755875], [-0.11400000005960464, -1.52600
00228881836, -0.62999999523162842], [-1.6039999723434448, -0.55400002002716064,
-0.574999988807907104], [-0.903999998426437378, -0.9649999737739563, -3.144999980
9265137], [-1.3259999752044678, 0.158000000727176666, 3.1860001087188721]], [[1,
2], [1, 8], [1, 9], [2, 5], [2, 3], [2, 10], [3, 11], [3, 12], [3, 4], [4, 13],
[5, 6], [5, 7], [7, 14]])
```

Drawing small molecules from PDB files

Return a geometric value — i.e. a `<pyplasm.xge.xgepy.Hpc>` object

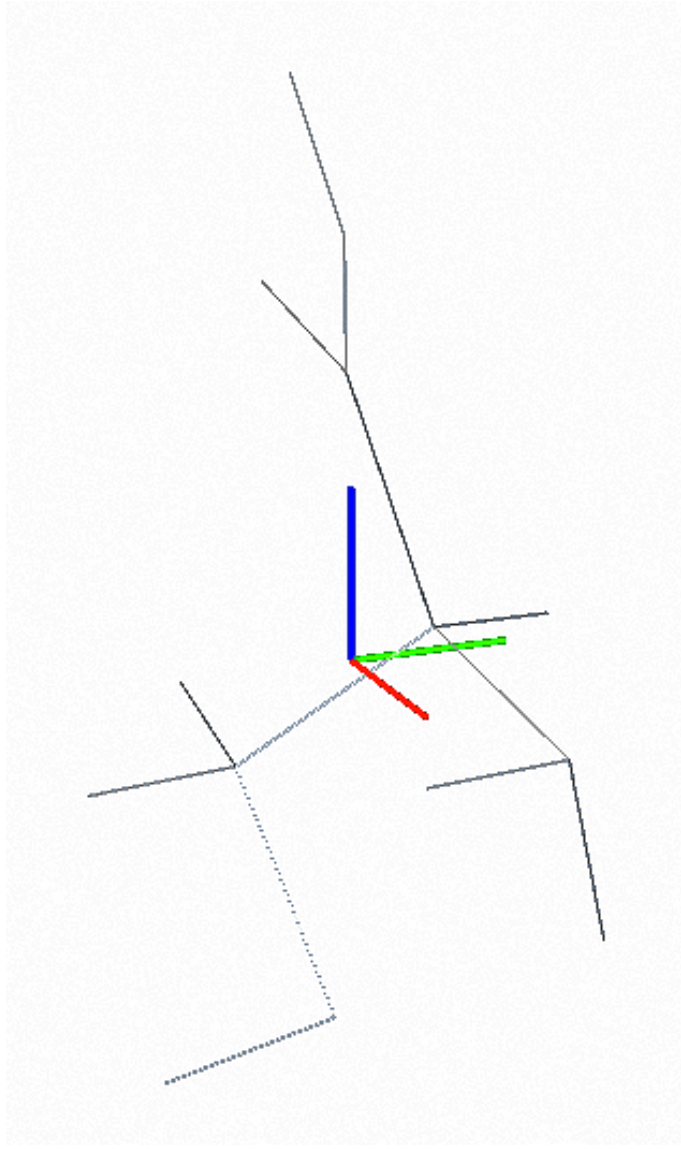
```
def graph (filename):
    parser=PDBParser()
    structure=parser.get_structure('molecule', filename)
    model = structure[0]
    chain = model[' ']
    residue = chain[0]

    nodes = [atom.get_coord().tolist() for atom in residue]
    edges = getPDBconnect('BTC.pdb')
    return MKPOL([nodes,edges,None])

myprint("graph('BTC.pdb')")
```

```
graph('BTC.pdb') -> <pyplasm.xge.xgepy.Hpc; proxy of <Swig Object of type 'std:
:tr1::shared_ptr< Hpc > *' at 0x326c50> >
```

Drawing small molecules from PDB files

View the 1-complex — embedded in 3D — of the molecule from `BTC.pdb`

```
VIEW(graph('BTC.pdb'))
```



Drawing small molecules from PDB files

Get the atomic radiuses from `atomic_radius.py`

```
## http://en.wikipedia.org/wiki/Atomic_radius#Calculated_atomic_radii
## http://en.wikipedia.org/wiki/Atomic_radius#Empirically_measured_atomic_radius
## http://en.wikipedia.org/wiki/Van_der_Waals_radius
## http://en.wikipedia.org/wiki/Covalent_radius

## 0 => No data available
## units = picometers: 1.0 x 10^(-12)

RADIUS_TYPE = 3 # van der Waals

## symbol:(name, empirical, Calculated, van der Waals, covalent)
atomic_radius = {
     "H":("hydrogen",35,53,120,38),
     "He":("helium",0,31,140,32),
     "Li":("lithium",145,167,182,134),
     "Be":("beryllium",105,112,153,90),
     "B":("boron",85,87,192,82),
     "C":("carbon",70,67,170,77),
     "N":("nitrogen",65,56,155,75),
     "O":("oxygen",60,48,152,73),
     "F":("fluorine",50,42,147,71),
     "Ne":("neon",0,38,154,69),
     "Na":("sodium",180,190,227,154),
     "Mg":("magnesium",150,145,173,130),
     "Al":("aluminium",125,118,184,118),
     "Si":("silicon",110,111,210,111),
     "P":("phosphorus",100,98,180,106),
     "S":("sulfur",100,88,180,102),
     "Cl":("chlorine",100,79,175,99),
```

Drawing small molecules from PDB files

Extract atom data (radiuses and type)

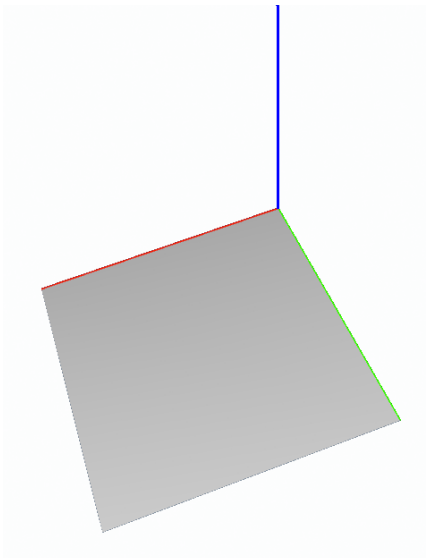
```
from atomic_radius import *


myprint("atomic_radius['Mg']")
myprint("atomic_radius['O'][0:2]")



myprint("[atom.get_id() for atom in residue]")
myprint("[atom.get_id()[0] for atom in residue]")
myprint("set([atom.get_id()[0] for atom in residue])")
myprint("list(set([atom.get_id()[0] for atom in residue]))")
```

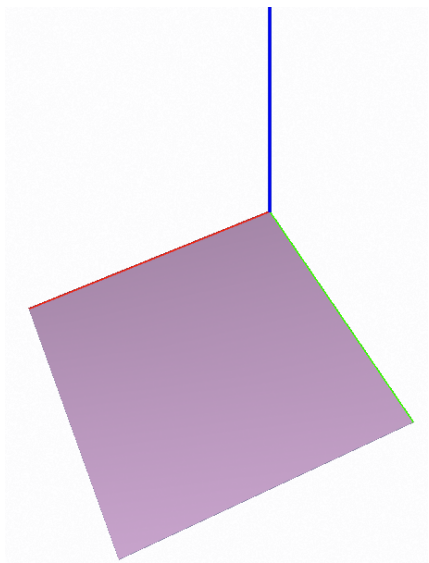
```
atomic_radius['Mg'] -> ('magnesium', 150, 145, 173, 130)

atomic_radius['O'][0:2] -> ('oxygen', 60)

[atom.get_id() for atom in residue] -> ['N', 'CA', 'CB', 'SG', 'C', 'O', 'OXT',
'H', 'H2', 'HA', 'HB2', 'HB3', 'HG', 'HXT']

[atom.get_id()[0] for atom in residue] -> ['N', 'C', 'C', 'S', 'C', 'O', 'O', '
H', 'H', 'H', 'H', 'H', 'H', 'H']

set([atom.get_id()[0] for atom in residue]) -> set(['H', 'C', 'S', 'O', 'N'])

list(set([atom.get_id()[0] for atom in residue])) -> ['H', 'C', 'S', 'O', 'N']
```

Drawing small molecules from PDB files

Atom color definition, according to practice (for biomolecules)

```
atom_color = {
     'H': Color4f([0.8, 0.8, 0.8, 1.0]), # ligth gray
     'C': Color4f([0.3, 0.3, 0.3, 1.0]), # dark gray (quite black)
     'N': BLUE,
     'O': RED,
     'F': Color4f([0.0, 0.75, 1.0, 1.0]), # ligth blue
     'P': ORANGE,
     'S': YELLOW,
     'Cl': GREEN,
     'K': Color4f([200./255, 162./255, 200./255, 1.0]) # lilac
}

myprint("VIEW(COLOR(atom_color['H'])(CUBOID([1,1])))")
myprint("VIEW(COLOR(atom_color['K'])(CUBOID([1,1])))")
```

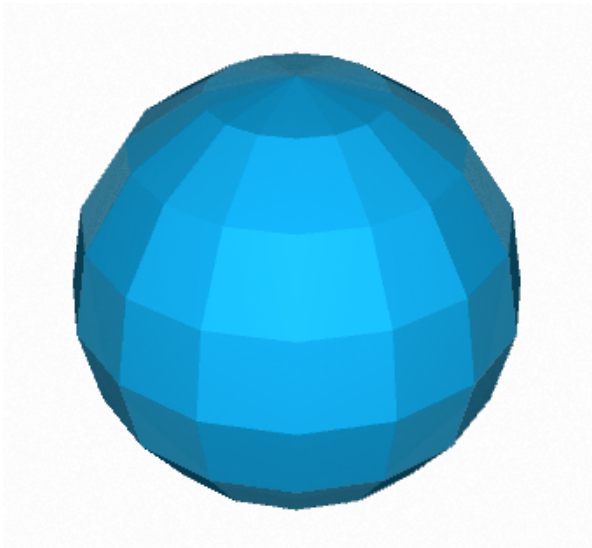


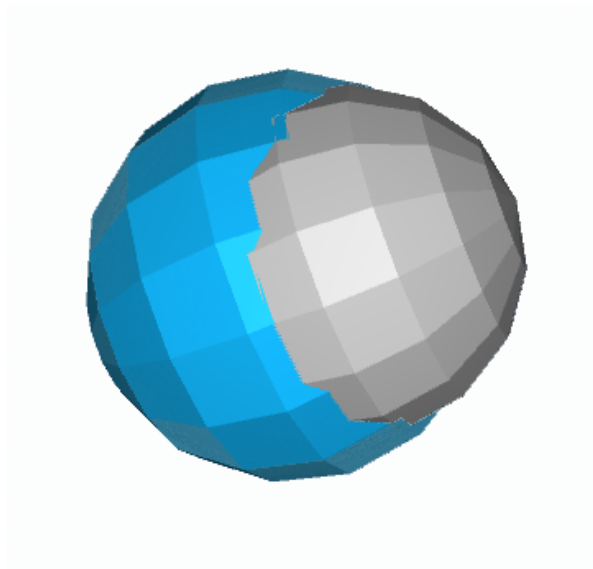


Drawing small molecules from PDB files

Drawing colored spheres — notice the conversion: picometers → Angstrom

```
def sphere(atom_code):
    return COLOR(atom_color[atom_code])(
        SPHERE(atomic_radius[atom_code][RADIUS_TYPE]/100.)([8,16]))

myprint("VIEW(sphere('F'))")


VIEW(STRUCT([ sphere('F'), T([1,2,3])([0.,.3,.5]), sphere('H') ]))
```

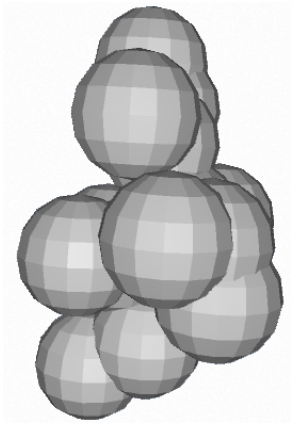


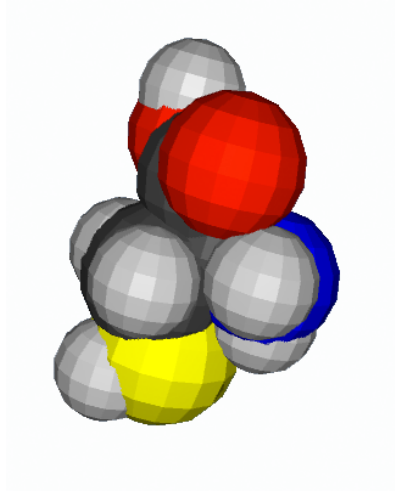


Drawing small molecules from PDB files

aaaaaaaa

```
def graph2 (filename):
    parser=PDBParser()
    structure=parser.get_structure('molecule', filename)
    model = structure[0]
    chain = model[' ']
    residue = chain[0]

    nodes = [atom.get_coord().tolist() for atom in residue]
    transls = AA(T([1,2,3]))(nodes)
    return STRUCT(CONS(transls)(sphere('H')))

myprint("graph2('BTC.pdb')")

VIEW(STRUCT([ graph('BTC.pdb'), graph2('BTC.pdb') ]))
```





Drawing small molecules from PDB files

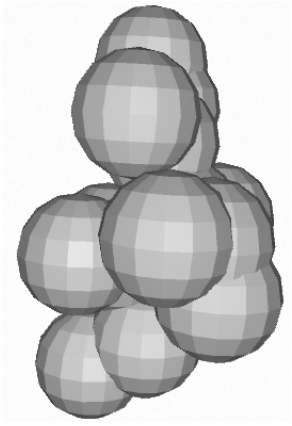
aaaaaaaa

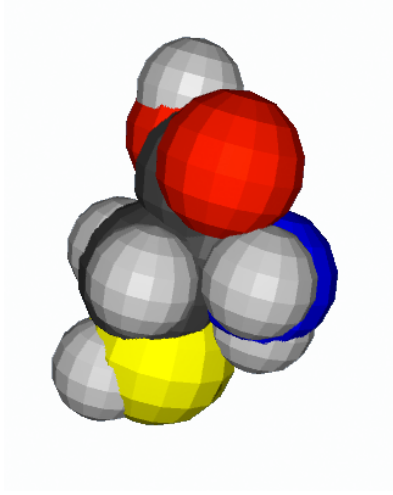
```
atom_types = [atom.get_id()[0] for atom in residue]
myprint("atom_types")

AA(sphere)(atom_types)
```

Drawing small molecules from PDB files

aaaaaaaa

```
def graph3 (filename):
    parser = PDBParser()
    structure = parser.get_structure('molecule', filename)
    residue = structure[0][' '][0]
    nodes = [atom.get_coord().tolist() for atom in residue]
    transls = AA(T([1,2,3]))(nodes)
    atom_types = [atom.get_id()[0] for atom in residue]
    atoms = AA(sphere)(atom_types)
    return STRUCT(AA(STRUCT)(TRANS([transls,atoms])))

VIEW(graph3('BTC.pdb'))
VIEW(STRUCT([ graph('BTC.pdb'), graph3('BTC.pdb') ]))
```

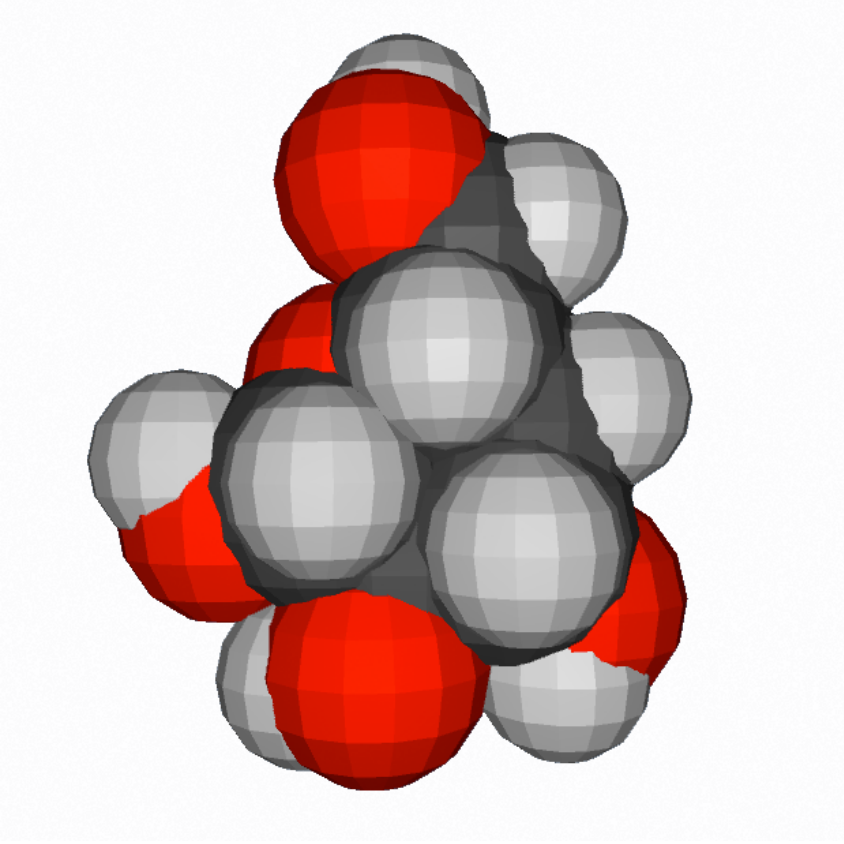




Drawing small molecules from PDB files

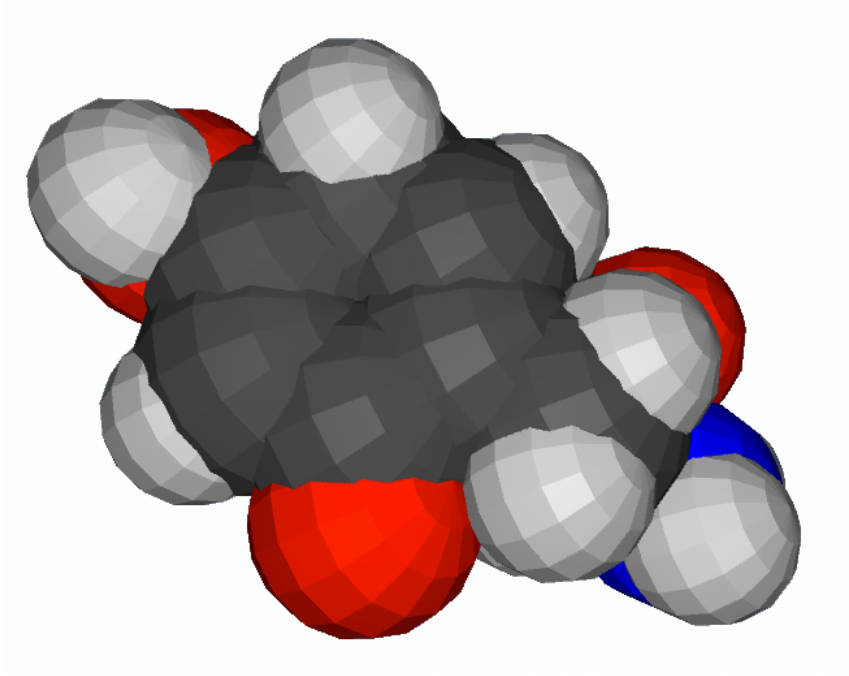
Other examples

BETA-D-GALACTOSE



```
VIEW(graph3('GAL.pdb'))
```

2-AMINO-3-(4-HYDROXY-6-OXOCYCLOHEXA-1,4-DIENYL)PROPANOIC ACID



```
VIEW(graph3('OTY.pdb'))
```

Sommario

Local to global coordinates

Main mechanism to attach substructures

- each substructure (geometric value) is given in a local coordinate frame
- a pair $[a, b]$ of substructures are joined in a common (global) coordinate frame
- the resulting structure is obtained as

```
c = STRUCT([a,b])
```

- to relocate b within the system of a (and c) we do:

```
c = STRUCT([a, Q, b])
```

where Q is an affine transformation (translation, rotation, scaling) applied to b

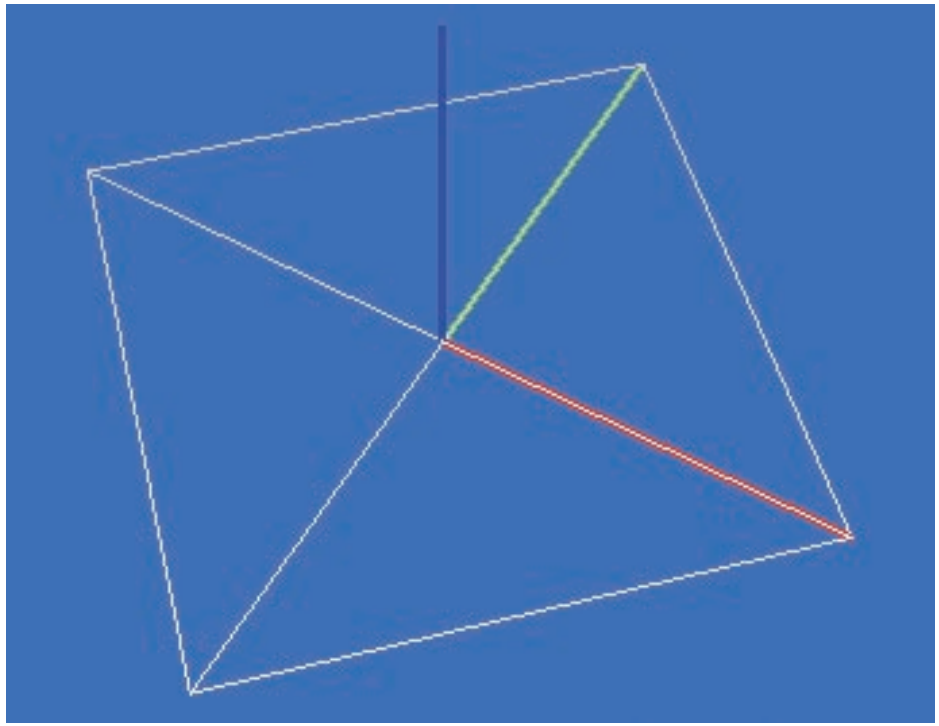
Local to global coordinates

STRUCT applies to sequences of objects and transformations

```
a = SKELETON(1)(SIMPLEX(2))
Q = R([1,2])(PI/2)
VIEW(STRUCT([a,Q,a,Q,a,Q,a]))
```

```
a1 = Q(a)
a2 = Q(Q(a))
a3 = Q(Q(Q(a)))
VIEW(STRUCT([a,a1,a2,a3]))
```

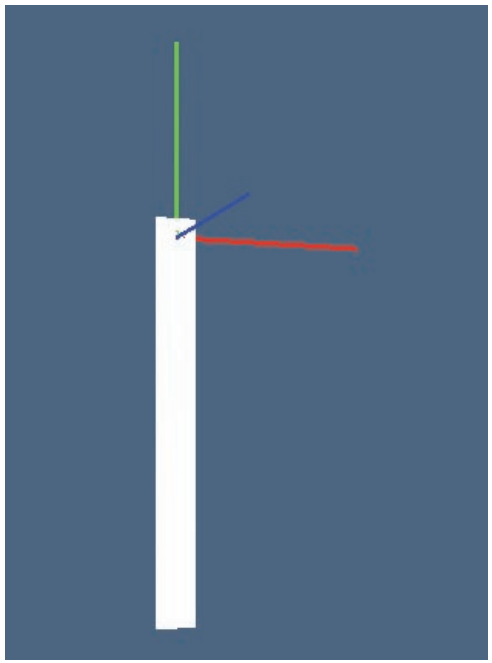
the above ⊂ objects are equivalent

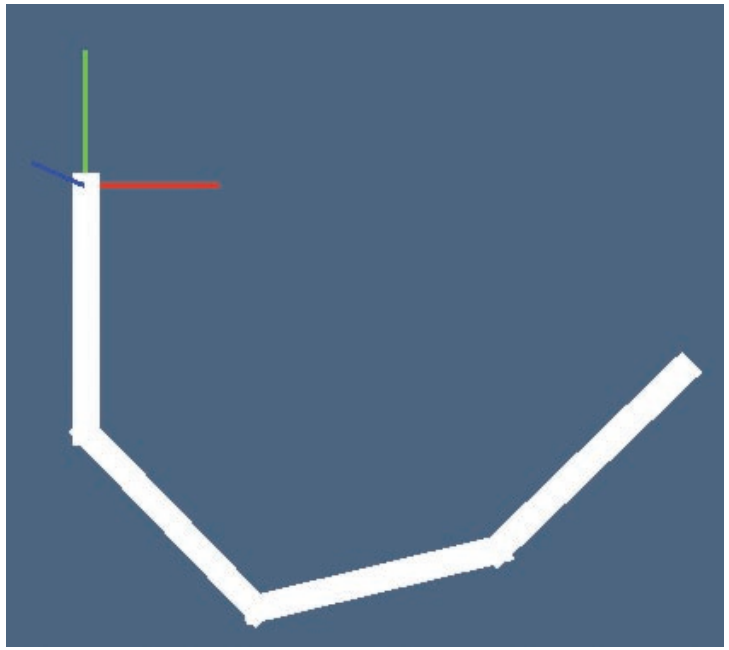


Local to global coordinates

2D chain of objects

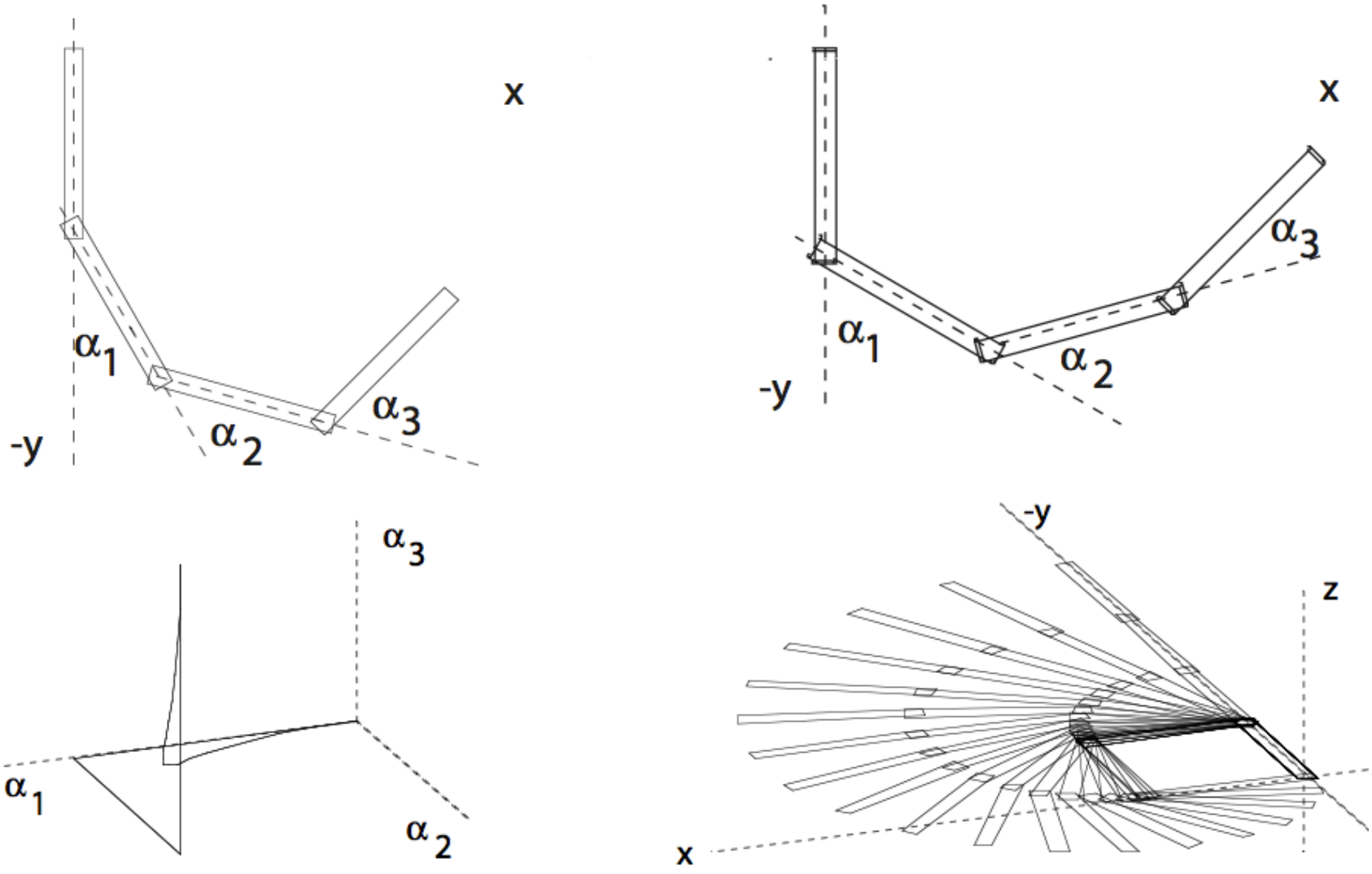
```
link = T([1,2])([-.1,-1.9])(CUBOID([.2,2.]))
def arm (a1,a2,a3):
    return STRUCT([link,
                   T(2)(-1.8), R([1,2])(a1), link,
                   T(2)(-1.8), R([1,2])(a2), link,
                   T(2)(-1.8), R([1,2])(a3), link])
VIEW(link)
VIEW(arm(PI/4,PI/3,PI/6))
```





2D chain of objects

Configurations `arm([PI/6,PI/4,PI/3])` and `arm([PI/3,PI/4,PI/6])`



Configuration space path produced as a cubic Bézier curve

Math of rigid bodies chaining

degrees of freedom (DOF) are the set of independent parameters that specify the position and orientation of a rigid body

- 2D rigid body $B \subset \mathbb{E}^2$ has 3 DOFs:
  - $\mathbf{t} \in \mathbb{R}^2$ (traslation vector)
  - $\alpha \in [0, 2\pi]$ (angle)
- 3D rigid body $B \subset \mathbb{E}^3$ has 6 DOFs:
  - $\mathbf{t} \in \mathbb{R}^3$ (traslation vector)
  - $\alpha, \beta, \gamma \in [0, 2\pi]$ (3 angles or 3 spherical coords r, θ, ϕ)

Math of rigid bodies chaining

open kinematic chain

- A system with several bodies would have a combined DOF that is the sum of the DOFs of the bodies, less the internal constraints they have on relative motion

- A specific type of linkage is the open kinematic chain, where a set of rigid links are connected at joints

Sommario

EXAMPLE: "ALA.pdb" (Alanine PDB file)

```
HEADER    NONAME 22-Aug-09
TITLE     Produced by PDBeChem
COMPND    ALA
AUTHOR    EBI-PDBe Generated
REVDAT   1  22-Aug-09     0
ATOM      1  N   ALA     0      -0.966   0.493   1.500  1.00 20.00           N+0
ATOM      2  CA  ALA     0       0.257   0.418   0.692  1.00 20.00           C+0
ATOM      3  C   ALA     0      -0.094   0.017  -0.716  1.00 20.00           C+0
ATOM      4  O   ALA     0      -1.056  -0.682  -0.923  1.00 20.00           O+0
ATOM      5  CB  ALA     0       1.204  -0.620   1.296  1.00 20.00           C+0
ATOM      6  OXT ALA     0       0.661   0.439  -1.742  1.00 20.00           O+0
ATOM      7  H   ALA     0      -1.383  -0.425   1.482  1.00 20.00           H+0
ATOM      8  H2  ALA     0      -0.676   0.661   2.452  1.00 20.00           H+0
ATOM      9  HA  ALA     0       0.746   1.392   0.682  1.00 20.00           H+0
ATOM     10  HB1 ALA     0       1.459  -0.330   2.316  1.00 20.00           H+0
ATOM     11  HB2 ALA     0       0.715  -1.594   1.307  1.00 20.00           H+0
ATOM     12  HB3 ALA     0       2.113  -0.676   0.697  1.00 20.00           H+0
ATOM     13  HXT ALA     0       0.435   0.182  -2.647  1.00 20.00           H+0
CONECT    1    2    7    8
CONECT    2    3    5    9    1
CONECT    3    2    4    6
CONECT    4    3
CONECT    5    2   10   11   12
CONECT    6    3   13
CONECT    7    1
CONECT    8    1
CONECT    9    2
CONECT   10    5
CONECT   11    5
CONECT   12    5
CONECT   13    6
END
```

Aminoacid models

file input

```
def read(peptides):
    """ To produce the list of aminoacid models. """
    def moleculeFile(peptide):
        return 'aminoacids/'+ peptide +'.pdb'
    molecules = [moleculeGraph(moleculeFile(peptide)) for
        peptide in peptides]
    return molecules
```

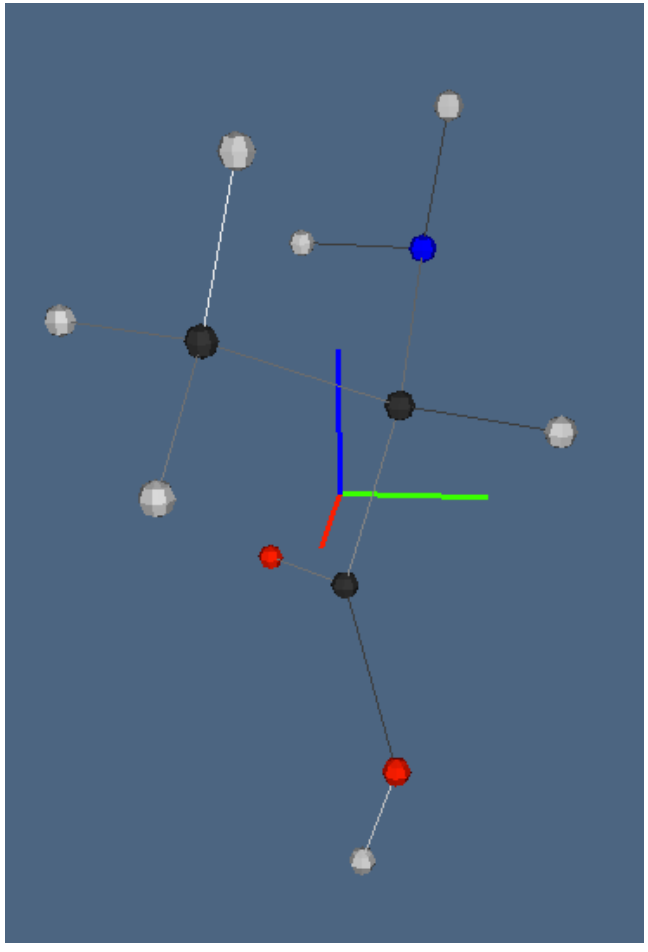
Aminoacid models

Reading from the aminoacids subdirectory

read : *listof*(*codes*) → *listof*(*moleculetriples*)

```
>>> read(["ALA"])
[[[[-0.966, 0.493, 1.5], [0.256,
    0.418, 0.691], [-0.093, 0.017,
    -0.716], [-1.055, -0.681,
    -0.922], [1.203, -0.62, 1.296],
     [0.661, 0.439, -1.7419],
    [-1.383, -0.425, 1.4819],
    [-0.6759, 0.661, 2.4519],
    [0.7459, 1.3919, 0.6819],
    [1.4589, -0.33, 2.3159],
    [0.7149, -1.5939, 1.307],
    [2.1129, -0.6759, 0.697],
    [0.435, 0.1819, -2.647]], [[1,
    2], [1, 7], [1, 8], [2, 3], [2,
     5], [2, 9], [3, 4], [3, 6],
    [5, 10], [5, 11], [5, 12], [6,
    13]], ['N', 'CA', 'C', 'O', 'CB
    ', 'OXT', 'H', 'H2', 'HA', 'HB1
    ', 'HB2', 'HB3', 'HXT']]]
```

```
>>> VIEW(model(CAT(read(["ALA"]))))
```



Aminoacid models

To generate the graph (node–labeled) of a CCD file

```
from ccdgl import *

def moleculeGraph (filename):
    """ To generate the graph (node–labeled) of a CCD file. """
    residues = CCDParser (filename)
    atoms = [atom for residue in residues for atom in residue]
    nodes = [atom.get_coord().tolist() for atom in atoms]
    arcs = getPDBconnect(filename)
    labels = [atom.get_id() for residue in residues for atom in
        residue]
    return [nodes,arcs,labels]
```

Aminoacid models

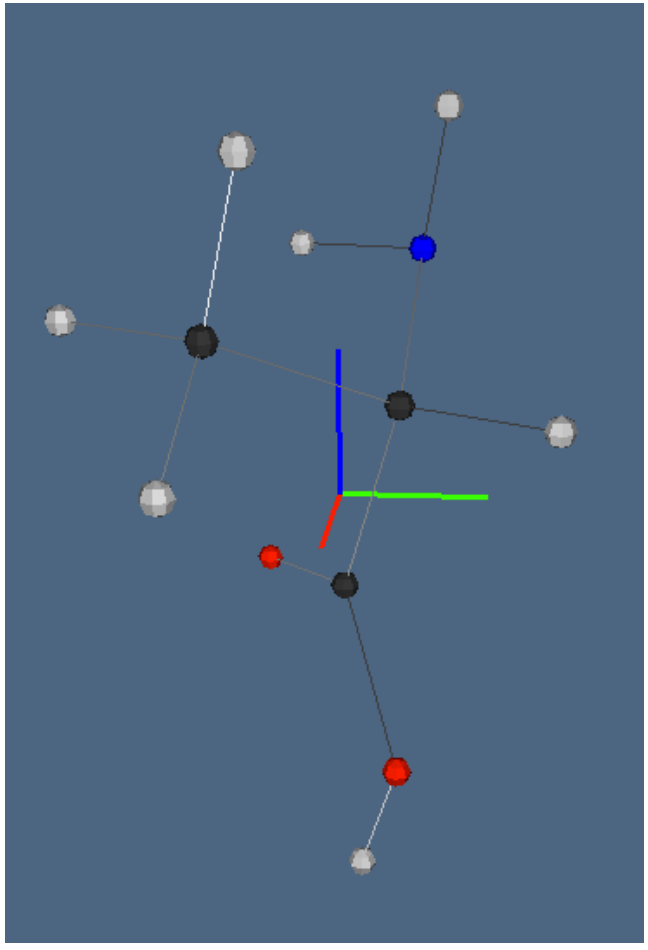
To generate the HPC geometry a molecule graph

```
def model(molecule):
    """ To generate the HPC geometry a molecule graph. """
    nodes,arcs,labels = molecule
    atoms = []
    for (node,label) in zip(nodes,labels):
        color = atom_color[label[0]]
        atom = T([1,2,3])(node)(SPHERE(0.1)([4,8]))
        atoms += [COLOR(color)(atom)]
    return STRUCT([STRUCT(atoms),MKPOL(molecule)])
```

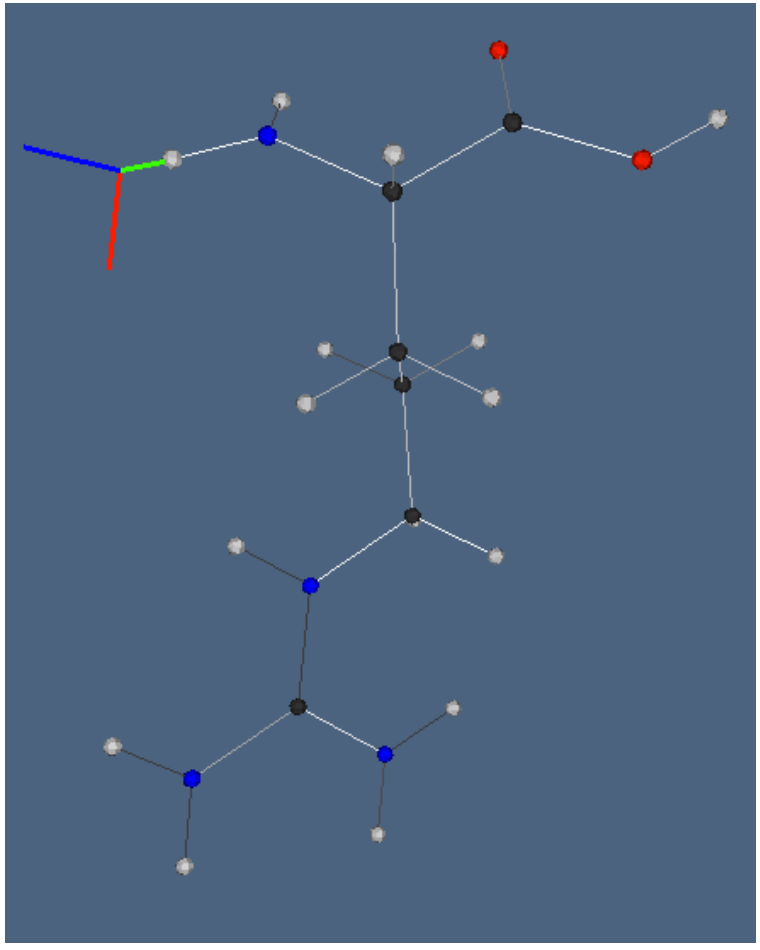
Aminoacid models

Reading from the aminoacids subdirectory

```
>>> VIEW(model(CAT(read(["ALA"]))))
```



```
>>> VIEW(model(CAT(read(["ARG"]))))
```



Each peptide is stored in the PDB file in a local coordinate system

Aminoacid models

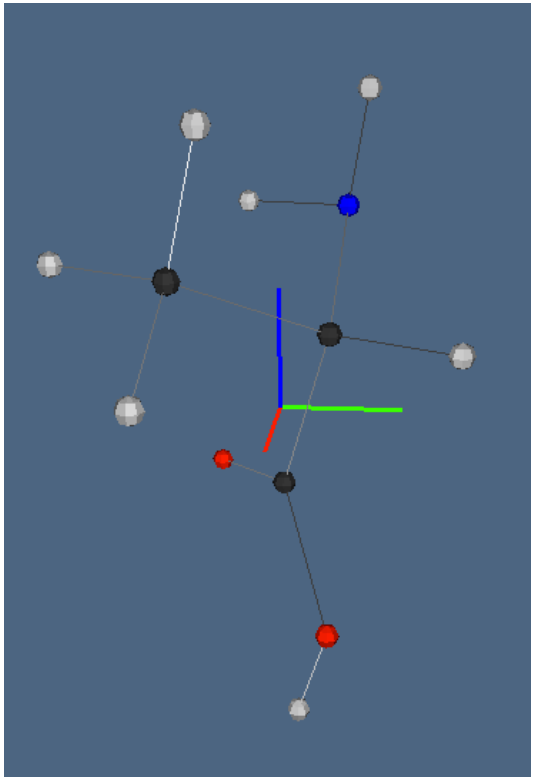
Returns the list of rotation, translation, HPC model and molecule's graph triple

```
def peptideTransform(hook,pivot,molecule):
    """ Affinely transforms a molecule.
        Returns the list of rotation, translation,
        HPC model and molecule's graph triple.
    """
    def transl(point):
        return T([1,2,3])([-coord for coord in point])
    vect = VECTDIFF([pivot,hook])
    axis = UNITVECT(VECTPROD([vect,[1,0,0]]))
    angle = ACOS(INNERPROD([UNITVECT(vect),[1,0,0]]))
    return [ROTN([angle,axis]),transl(hook),model(molecule),
        molecule]
```

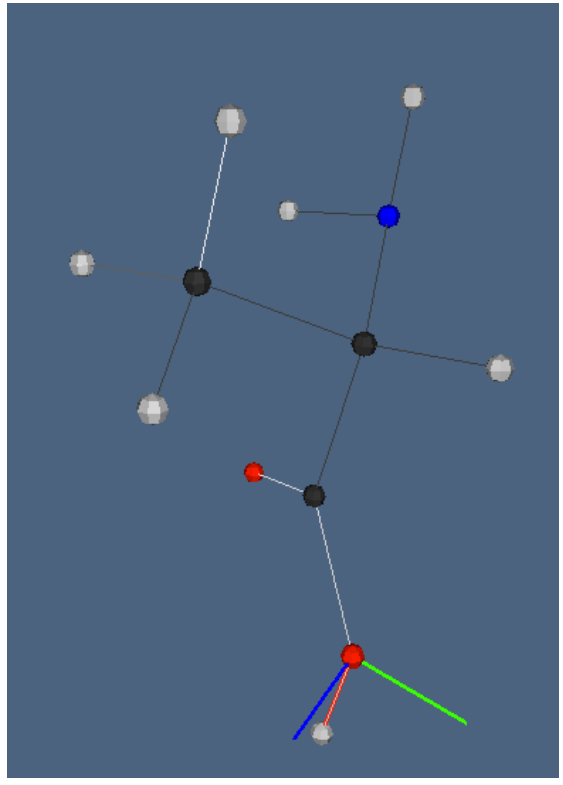
Aminoacid models

Affine transformation of the molecule

```
>>> molecule = CAT(read(["ALA"]))
>>> VIEW(model(molecule))
```



```
>>> hook  = [0.661,0.439,-1.742]
>>> pivot = [0.435,0.182,-2.647]
>>> VIEW(STRUCT(peptideTransform(hook
    ,pivot,molecule)[:-1]))
```



The (rigid) transformation moves the hook to the origin, and the pivot to the *x*-axis

Interface to affine transformation of the molecule

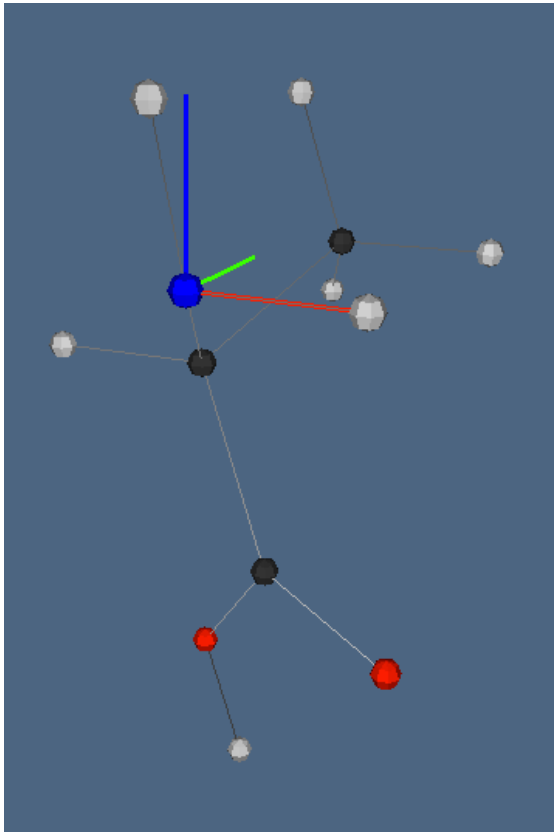
```
def link(molecule):
    """ Second order function to transform a molecule. """
    nodes,arcs,labels = molecule
    atom_codes = [label[0] for label in labels]
    def link0(fun0,code0,fun1,code1):
        def first(code):
            return atom_codes.index(code)
        def last(code):
            return len(atom_codes) - 1 - atom_codes[::-1].index(
                code)
        a0,a1 = eval(fun0)(code0),eval(fun1)(code1)
        return peptideTransform( nodes[a0],nodes[a1], molecule )
    return link0
```

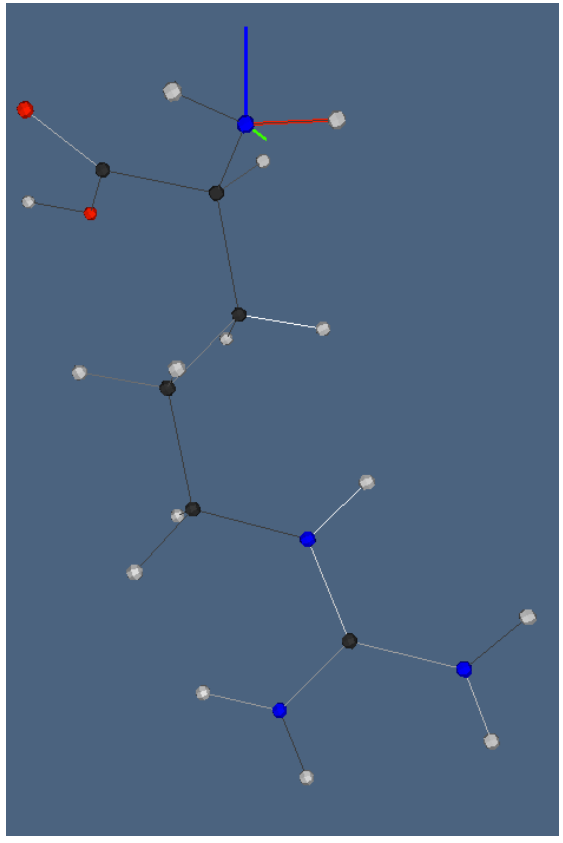
```
>>> molecule = moleculeGraph("aminoacids/ALA.pdb")
>>> link (molecule)
<function link0 at 0x16f5d8f0>
>>> link (molecule)('first','N','first','H')
[<function h at 0x16f5d0f0>, <function <lambda> at 0x16f5d470>,
    <pyplasm.xge.xgepy.Hpc; proxy of <Swig Object of type 'std
    ::tr1::shared_ptr<␣Hpc␣>␣*' at 0x33bef0> >, [[[-0.966,
    0.493, 1.5], [0.257, 0.418, 0.692], [-0.094, 0.017,
    -0.716], [-1.056, -0.682, -0.923], [1.204, -0.620, 1.296],
    [0.661, 0.439, -1.742], [-1.383, -0.425, 1.482], [-0.676,
    0.661, 2.452], [0.746, 1.392, 0.682], [1.459, -0.330,
    2.316], [0.715, -1.594, 1.307], [2.113, -0.676, 0.697],
    [0.435, 0.182, -2.647]], [[1, 2], [1, 7], [1, 8], [2, 3],
    [2, 5], [2, 9], [3, 4], [3, 6], [5, 10], [5, 11], [5, 12],
    [6, 13]], ['N', 'CA', 'C', 'O', 'CB', 'OXT', 'H', 'H2', 'HA
    ', 'HB1', 'HB2', 'HB3', 'HXT']]]
>>> link (molecule)('first','N','first','H')[:-1]
[<function h at 0x16f5c0f0>, <function <lambda> at 0x16f5d630>,
    <pyplasm.xge.xgepy.Hpc; proxy of <Swig Object of type 'std
    ::tr1::shared_ptr<␣Hpc␣>␣*' at 0x33bea8> >]
>>> STRUCT(link (molecule)('first','N','first','H')[:-1])
<pyplasm.xge.xgepy.Hpc; proxy of <Swig Object of type 'std::tr1
    ::shared_ptr<␣Hpc␣>␣*' at 0x33bf08> >
>>> VIEW(STRUCT(link (molecule)('first','N','first','H')[:-1]))
```

Interface to affine transformation of the molecule

```
>>> molecule = moleculeGraph("
    aminoacids/ALA.pdb")
>>> VIEW(STRUCT(link (molecule)('
    first','N','first','H')[:-1]))
```

```
>>> molecule = moleculeGraph("
    aminoacids/ARG.pdb")
>>> VIEW(STRUCT(link (molecule)('
    first','N','first','H')[:-1]))
```





The (rigid) transformation moves the hook ('first','N') to the origin, and the pivot ('first','H') to the *x*-axis

Aminoacid models

To transform a molecule structure, where *structure* := (*rotation*, *translation*, *graph*), and *graph* := (*vertices*, *edges*, *labels*), to a *newgraph* := (*newvertices*, *edges*, *labels*)

```
def transform(structure):
    """ To transform a molecule structure according to the
        translation and rotation in its graph triple. """
    nodes,arcs,labels = structure[-1]
    pol = STRUCT(CAT([ structure[0:2], AA(MK)(nodes) ]))
    return [UKPOL(pol)[0],arcs,labels]
```

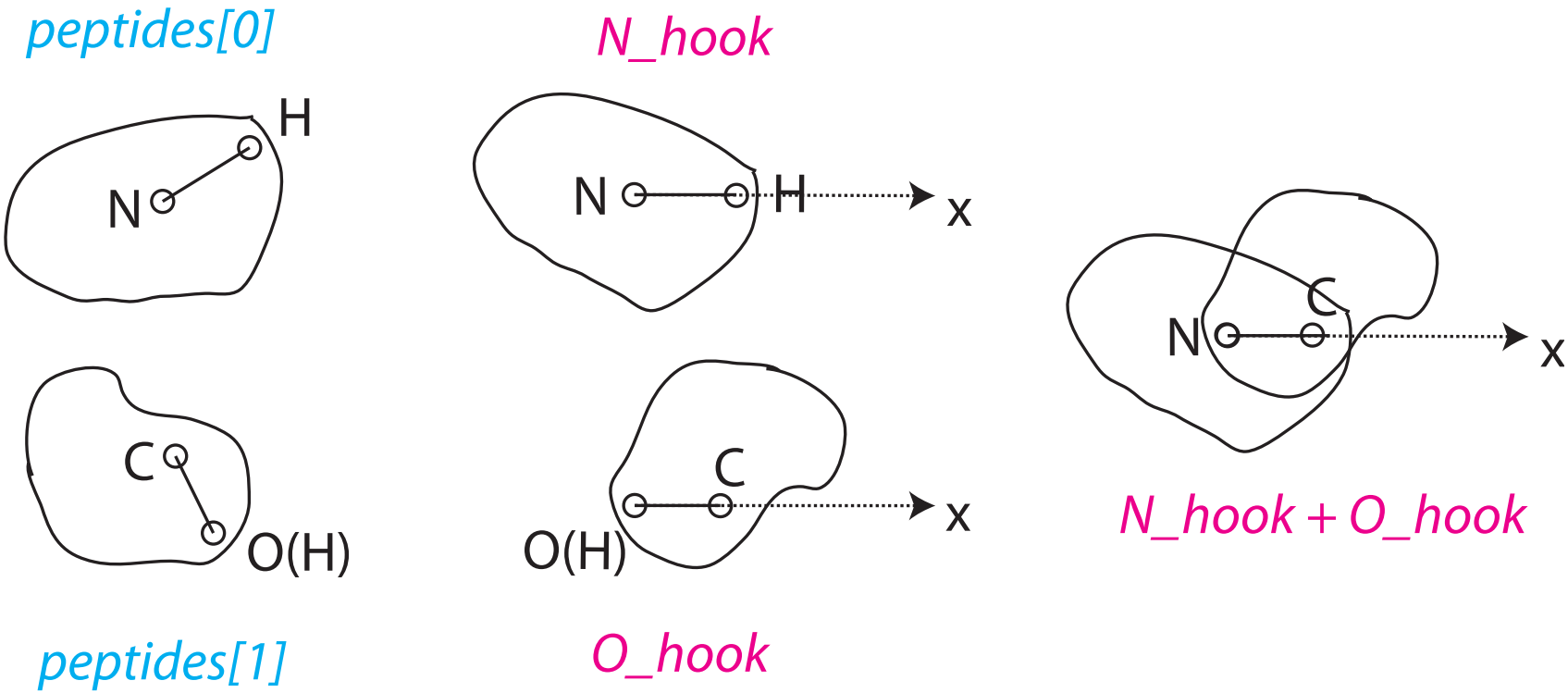
Peptide bond

Definitions from Wikipedia

- A peptide bond (amide bond) is a covalent chemical bond formed between two molecules when the carboxyl group of one molecule reacts with the amino group of the other molecule, thereby releasing a molecule of water (H_2O).
- This is a dehydration synthesis (also known as a condensation reaction), and usually occurs between amino acids.
- The resulting C(O)NH bond is called a peptide bond, and the resulting molecule is an amide.
- The four-atom functional group -C(=O)NH- is called a peptide link.
- Polypeptides and proteins are chains of amino acids held together by peptide bonds, as is the backbone of PNA (Peptide Nucleic Acid).

Peptide bond

Implementation via translation of *N_hook*, *O_hook* to the origin and rotation of the *N_pivot*, *O_pivot* to the x axis

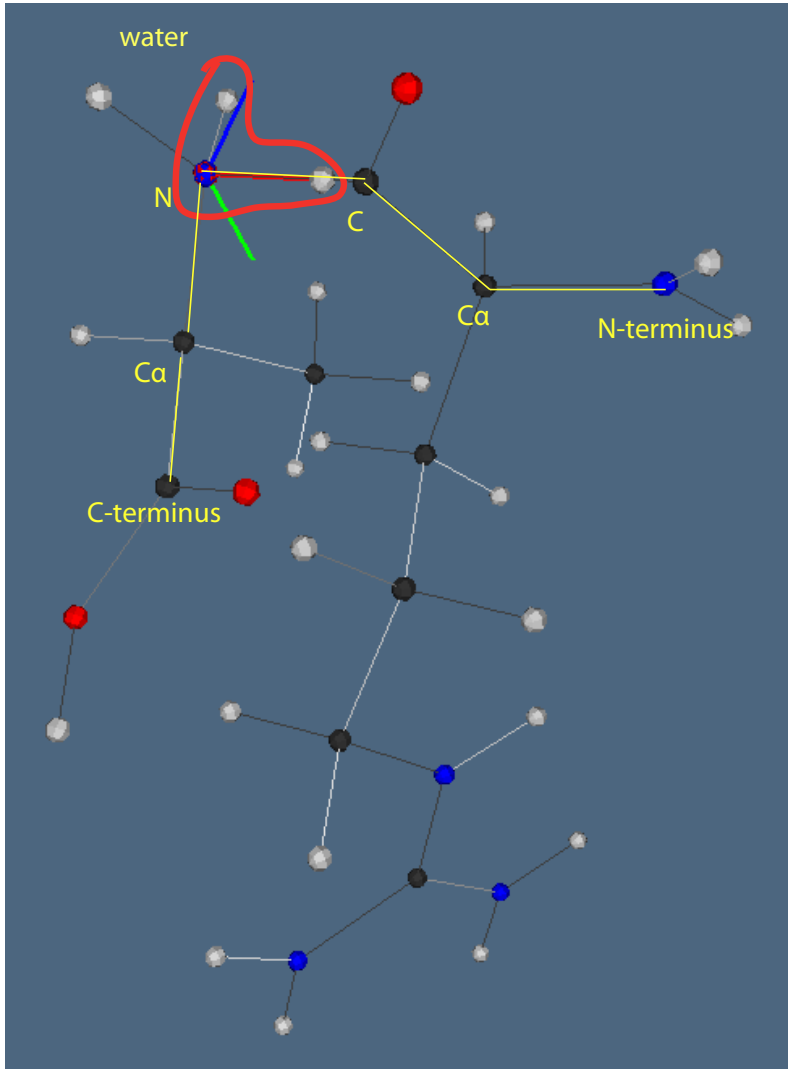


Example

Peptide bond (ALA — ARG)

Two peptide molecules are put in the same coordinate system

```
>>> peptides = ["ALA", "ARG"]
>>> molecules = read(peptides)
>>> O_hook, N_hook = firstPair(molecules)
>>> VIEW(STRUCT([STRUCT(O_hook[:3]),
    STRUCT(N_hook[:3])]))
```



Polypeptide construction

linkPeptides(molecules) is used to generate the data needed to chain a peptide list

```
def firstPair(molecules):
    """ To link the first pair of a list of peptides. """
    N_hook = link(molecules[0])('first','N','first','H')
    O_hook = link(molecules[1])('last','O','ID',2)
    return [O_hook,N_hook]
```

```
def nextPeptides(molecule,previous):
    """ To chain a peptide to the previous in a polypeptide. """
    N_hook = link(transform(previous))('first','N','first','H')
    O_hook = link(molecule)('last','O','ID',2)
    return [O_hook,N_hook]
```

```
def linkPeptides(molecules):
    """ To generate the data needed to chain a peptide list. """
    hooks = [firstPair(molecules)]
    for k in range(2,len(molecules)):
        transformed = nextPeptides(molecules[k],hooks[0][0])
        hooks.insert(0,transformed)
    return hooks
```

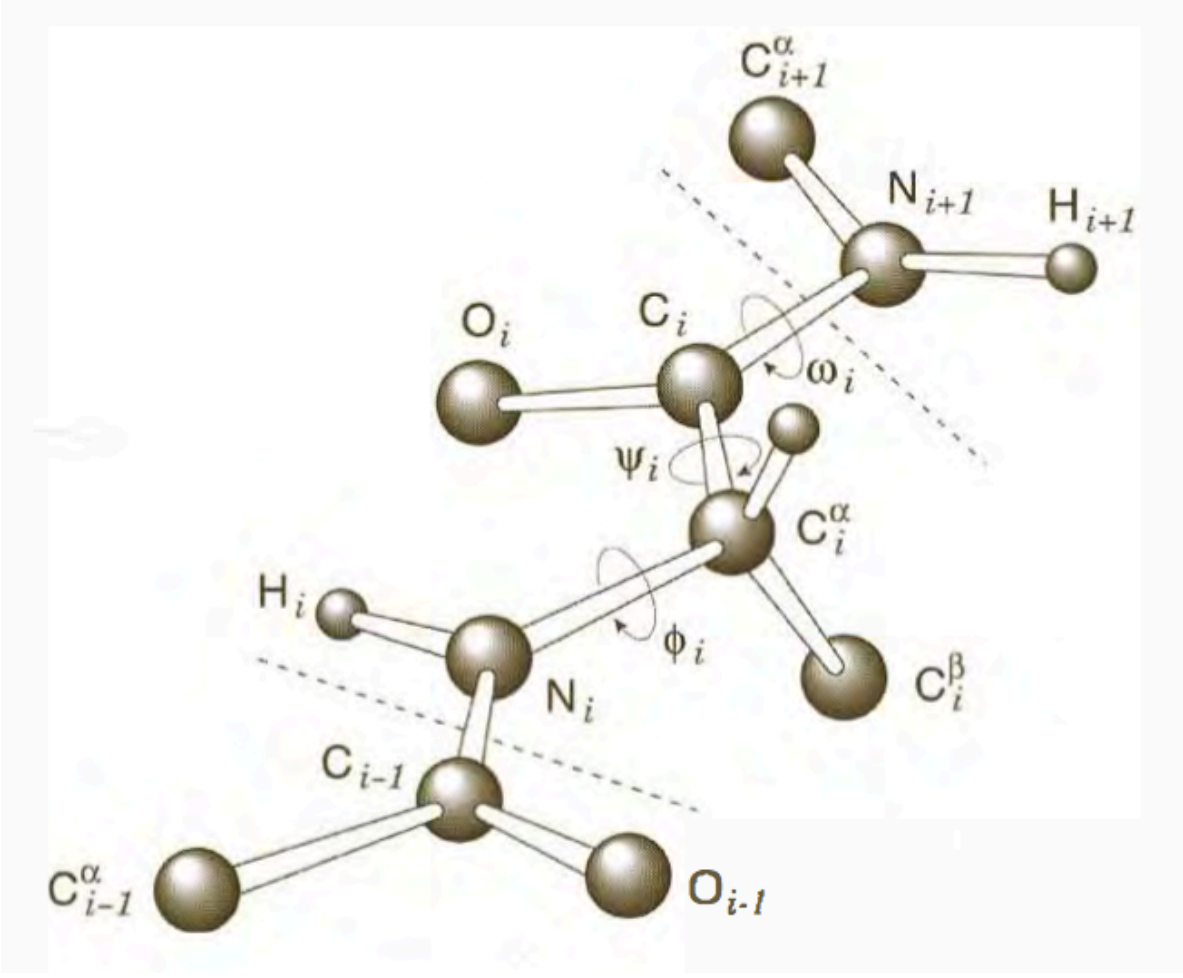
Make (fake) protein conformations

A constant torsional rotation of π angle around each peptide bond axis is given here

```
torsion = R([1,2,3])(PI)
def polypeptide(peptides):
    """ To chain peptide models in local coords and
        transformations.
        Returns an HPC model """
    molecules = read(peptides)
    structures = linkPeptides(molecules)
    out = [[STRUCT(structures[-1][0][:3])],structures
        [-1][1][:-1]]]
    structures = structures[:-2]
    for structure in structures[::-1]:
        out.insert(0,[[STRUCT(structure[0][:3])],[torsion],
            structure[1][:2]])
    return STRUCT(CAT(CAT(out)))
```

Protein conformations

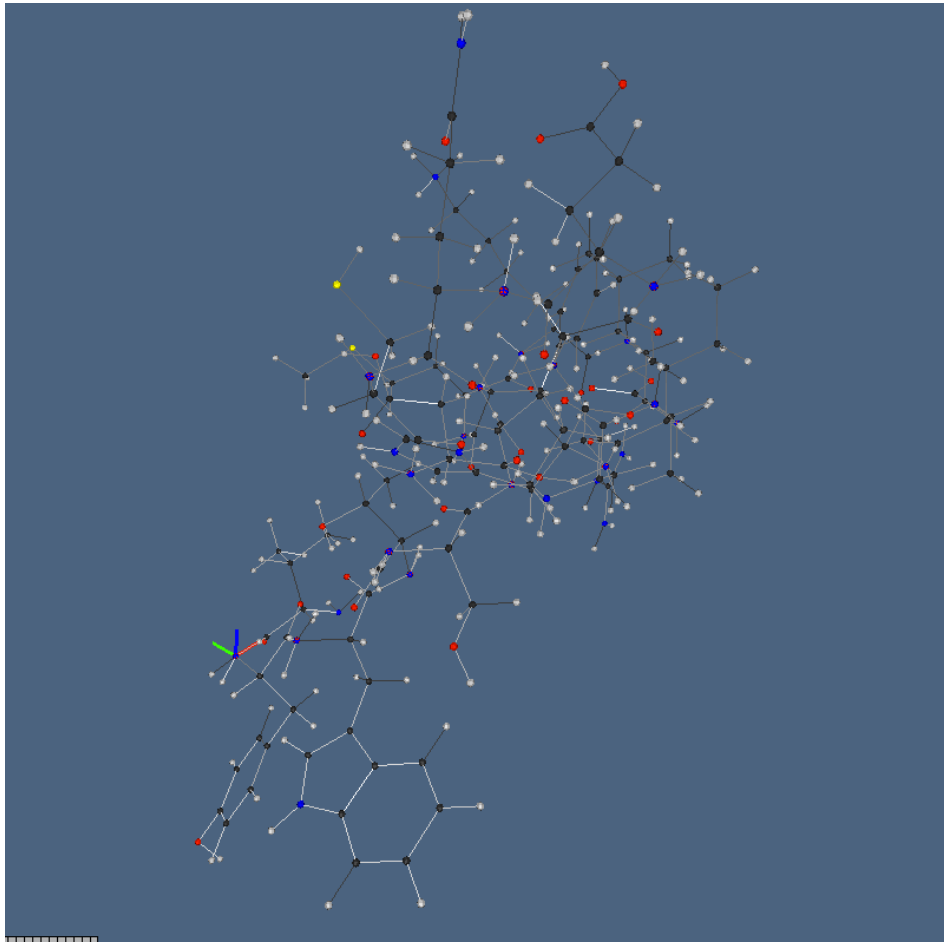
The protein conformation may be described by $3N_{peptides}$ torsional angles ϕ_i, ψ_i, ω_i (generalized coordinates), instead than by $3N_{atoms}$ cartesian coordinates x_k, y_k, z_k, where $3N_{peptides} \approx N_{atoms}/3$

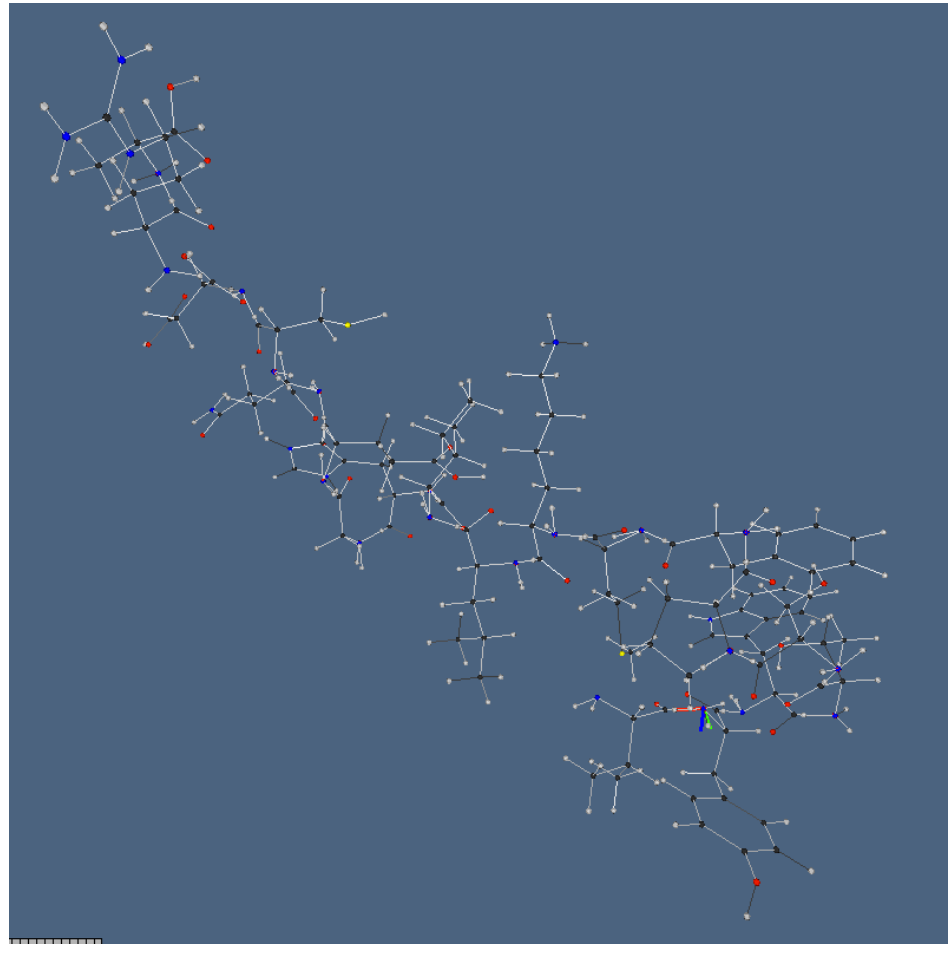


[from Lesk, Introduction to Bioinformatics, Oxford Univ. Press, 2008]

Examples — (fake) polypeptide(peptides)

Conformations depending only on ONE generalized coordinate





```
>>> torsion = R([2,3])(0)
>>> VIEW(polypeptide(peptides))
```

```
>>> torsion = R([2,3])(PI)
>>> VIEW(polypeptide(peptides))
```

EXAMPLE — (fake) polypeptide models

The chain of peptides is given by amino acids in lexicographical order

```
if __name__ == "__main__":

    peptides = ["ARG", "ASN","ASP","CYS", "GLN", "GLU", "GLY",
                "HIS", "ILE", "LEU", "LYS", "MET", "PHE", "PRO",
                "SER", "THR", "TRP", "TYR", "VAL"]

    protein = polypeptide(peptides)
    VIEW(protein)
```

Parameterization of chains by angles

bbbbbbbbb

- aaaaaaa
- aaaaaaa
- aaaaaaa