Intuitive visualization of surface properties of proteins

Andrei R[1,2], M Callieri[3], Zini MF[1], Loni T[4], Maraziti G[4] and Zoppè M[1]

Scientific Visualization Unit

[1]Institute of Clinical Physiology, CNR; [2]Scuola Normale Superiore; [3]Big Bang Solutions (Navacchio); [4]ISTI – CNR. Pisa.

www.scivis.ifc.cnr.it

Introduction

"A picture is worth a thousand words" goes the old adage. We have engaged in the elaboration of motion and in the study of a visualization system that permits the observation of animated proteins with the simultaneous representation of molecular surface (determined by all atoms), with its Lipophilic and Electrostatic potentials. Our aim is to show proteins' internal motion obtained from structural data, visualizing molecules in a directly informative way. This task is done using Blender[1], an open source software for 3D animation, visual effects and video games, in conjunction with several scientific programs.

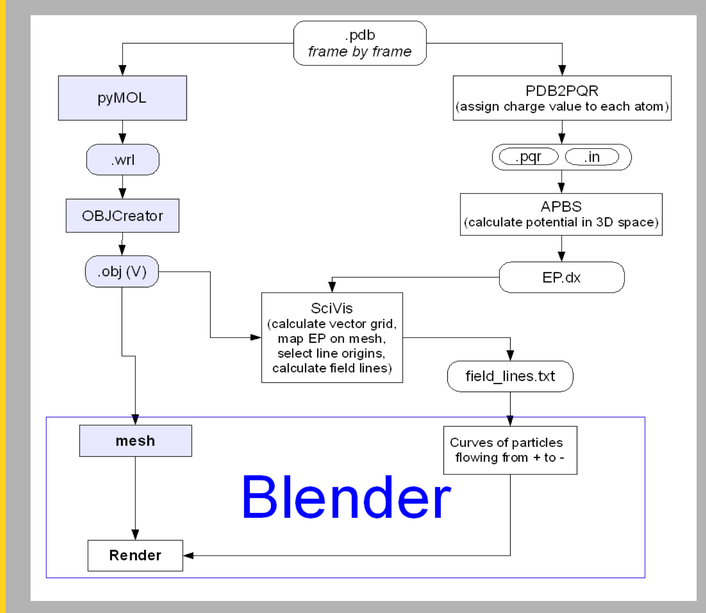
Proteins' motion

Using custom scripts written in Python[2], .pdb files are imported in Blender. The conformations are set in the time line of Blender and its Game Engine interpolates between them considering both the links between atoms (rigid body joints, built using an aa library) and collisions between all atoms. In this way we find new conformations of proteins as transition forms between a set of given experimental models. Interpolated conformations are exported as pseudo .pdb files and analysed using GROMOS[3] force field in SwissPDB-Viewer[4], and adjusted if necessary[5].

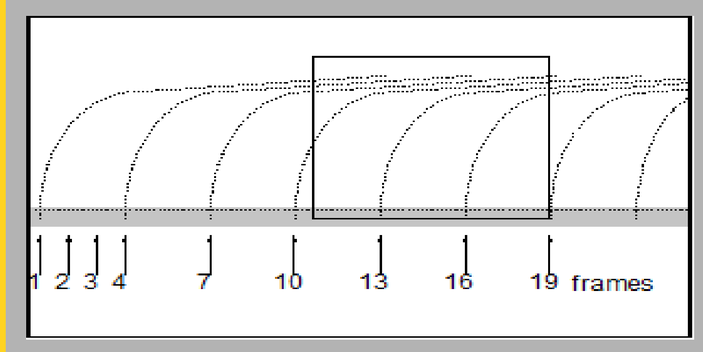
Blender

A complete package for 3D modeling, animation, special effects and game development. Objects are geometric entities created either by modeling or by importing them. They are given a surface appearance by the use of material shaders and textures. These two elements define the behavior of the surface when illuminated, by specifying local informations like color, reflectance (dull or shiny) and micro-structure (roughness or smoothness). Once the animation and texturing is defined, the scene is equipped with other assets such as a background, lights and cameras and the process concludes with the 'filming' (rendering of all frames which are assembled to generate a video).

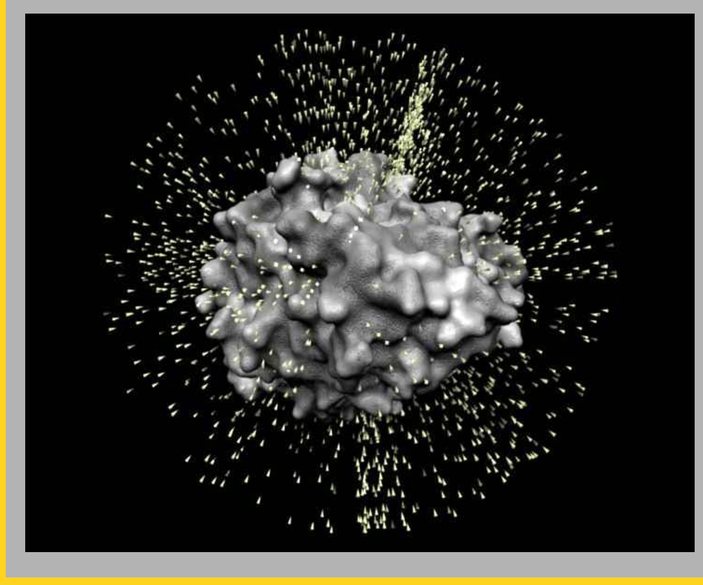
Electrostatic potential



EP calculus and representation pipeline
Starting from the same .pdb file used for MLP calculation, PDB2PQR[9] adds atomic charge to each atom, then APBS[10] calculates the EP values and stores them in a .dx file; SciVis (**d**) uses the information about the mesh (previously calculated for MLP – blue squares) and the .dx file to calculate the field lines: the EP values are mapped on the surface and the field lines are built following to the gradient of EP; the total number of lines will be proportional to the global EP of the molecule (Number of lines x eV/Å^2). The lines are saved as sequences of points in a ASCII and imported in Blender as curves along which travel small line particles from positive to negative, respecting the field lines convention in physics (**e**).



Particle generation and representation
Field lines are imported as curves every 3 frames. Particles emitted from their positive end have a life-time of 12 frames. After the tenth frame the system is in ready-state (marked with a square).

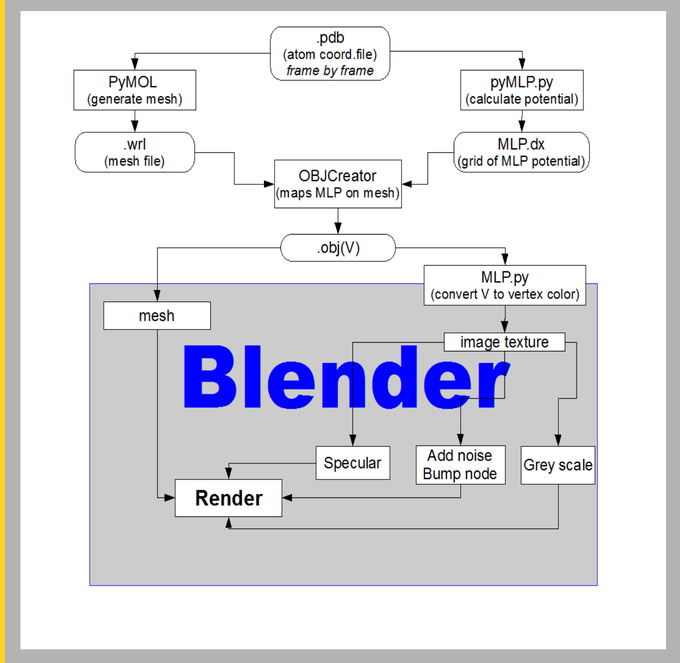


EP representation for still images or pdf with embedded 3D
Our main focus is the development of tool to visualize proteins' motions; however this is only possible where sufficient data are available from the literature. In still images, particles cannot transmit the polarity of the charged areas of molecules. To overcome this limitation, we propose the use of oriented objects, such as pyramids which are built along the field lines in scivis.exe with the tip oriented towards the negative end, maintaining the convention.
For the Interactive 3D representation we used 3DNP (3D-No Plugins), a 3D viewer[11], which does not need browser plugin, except JavaScript. The output is delivered as a collection of images taken from different points on a sphere around the protein. These images provide a 3D vision of the protein controlled by the viewer using the mouse.

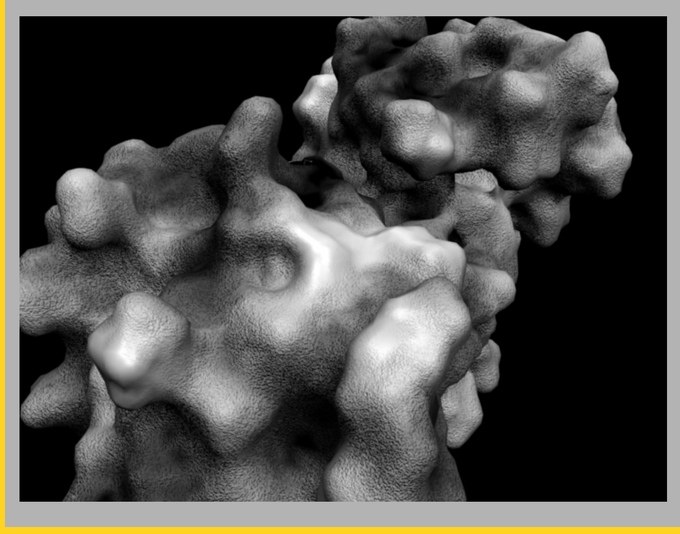
Surface properties visualization

We propose a new visual code for the representation of two important surface properties: electrostatic potential (EP) and molecular lipophilic potential (MLP). Using features different from color permits their simultaneous delivery in photorealistic images. The main idea of the proposed visual mapping is to exploit perceptual associations between the values to be mapped and visual characterization of real-world objects. Ideally, by using already established perceptual association, the viewer will be able to understand the provided information more naturally, without the use of explicit legends.

Lipophilic potential

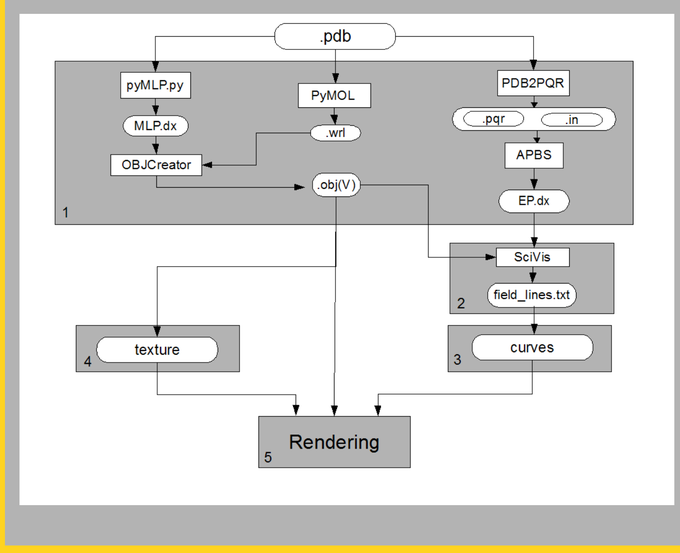


MLP calculus and representation pipeline
For each .pdb file exported from Blender, PyMOL[6] and pyMLP.py[7] calculate the surface and the MLP values (using Testa[8] formula), respectively. Then, OBJCreator (**a**) maps MLP (stored in a .dx file – MLP is calculated in every point of a grid in the protein space) on the surface using trilinear interpolation, and creates an .obj file. The .obj file is imported in Blender and the MLP values are converted to vertex colors (**b**). Each mesh is then unwrapped to generate a texture parametrization and the per-vertex color values are baked in a texture image. UV unwrapping is a procedure that consists in flattening a 3D object (e.g., the world globe) on a 2D plane (e.g., the world map), so that each vertex of the 3D mesh is assigned a correspondent 2D texture coordinate. These images are finally mapped on the material of the mesh using custom python scripts (**c**). In order to make the more hydrophilic areas rough the procedure involves the addition of a noise pattern of amplitude proportional to the degree of gray of the texture.



MLP representation
Final image at high resolution showing the gradient of MLP distribution over the molecular surface. The code for the representation of hydropathy that we propose is a range of optical features that go from light-smooth-shiny (hydrophobic) to dark-rough-dull (hydrophilic).

Automation



Automation of the entire procedure
Even for short videos of few seconds, hundreds of images are rendered (25frs/sec), creating the need of handling a number of different programs and managing very large data set. As shown in the scheme, every step of the calculi and of the rendering is automated in blocks. Steps of the automation are enclosed in squares. 1. execution of various programs outputs a .dx file containing the information on EP, and an .obj file with the surface of the protein with the MLP mapped on it (**f**); 2. field lines calculations and export as ASCII files; 3. field lines import in Blender; 4. image textures creation and their assignment to the material of each mesh; 5. rendering of the scene containing the textured meshes and field lines with flowing particles (**g**).

Conclusions

We have developed a computational instrument that allow the display of molecular surfaces of moving (or still) proteins and other macromolecules, putting special emphasis on their electrical and lipophilic properties. We consider that this representation will allow better (or at least more immediate and intuitive) understanding of the dynamical forces governing intermolecular interactions and thus facilitate new insights and discoveries.
Our system permits the fast morphing of proteins, elaborating transition between conformations in a fast and reliable way, using Blender Game Engine. Once obtained the sequence, we propose protein visualization as an animated video in which the molecule is shown as surface, rather than structure. EP and MLP are shown simultaneously for each intermediate conformation of moving proteins avoiding the use of color, which cannot be interpreted without a legend. Using real world tactile/sight feelings, the nanoscale world of proteins becomes more understandable, familiar to our everyday life, making it easier to introduce "un-seen" phenomena (concepts) such as hydropathy or charges, while leaving the utilization of color space for the description of other biochemical information.

References

1. www.blender.org
2. www.python.org
3. van Gunsteren, WF. et al. Vdf Hochschulverlag ETHZ,1996.
4. Guex, N. et al. Electrophoresis 1997, 18, 2714-2723.
5. Zini *et al.* Manuscript in preparation
6. DeLano WL: The PyMOL Molecular Graphics System. 2002
7. http://code.google.com/p/pymlp/source/browse/trunk/pyMLP.py
8. Testa B et al. *Pharm Res* 1996, 13:335-43
9. Dolinsky T J et al. *Nucleic Acids Res* 2004, 32:W665-7
10. Baker N A et al. *Proc Natl Acad Sci U S A* 2001, 98:10037-41
11. http://www.thoro.de/page/3dnp-introduction-en

Custom scripts and programs written in Python and C++ and available in open format from our website

a. OBJCreator – software application developed in ANSI C/C++. It maps MLP values on the mesh and exports an .obj file. It takes as inputs mesh data (.wrl) created by PyMOL and the values of MLP (.dx), created by pyMLP.py. For every vertex of the mesh, the correspondent grid-cell is identified and the local potential is calculated by trilinear interpolation. The .obj file stores information of the position of each vertex of the mesh and its correspondent MLP value.

b. MLP.py – a Python script to be used within the Blender integrated scripting engine to obtain image textures. It imports the .obj files into a Blender scene, and converts the MLP values to vertex colors (levels of gray) assigning a color to each vertex. The 3D surface is then unwrapped to obtain a UV texture parametrization. Finally, a texture map is built by baking the computed per-vertex color values in the texture image.

c. texture.py – a Python script used within Blender, that builds a basic material for the 3D mesh imported into the scene and assigns to this material the texture images generated by MLP.py.

d. SciVis.exe – a software application written in C++ used to calculate the field lines and to export them in an ASCII file to be imported in Blender. This tool uses the 3D surface (.obj) generated by OBJCreator and the EP grid (.dx) calculated by APBS. The computation of the field lines involves: EP values mapping on the 3D surface, gradient field calculation, automatic selection of areas with high values of EP and computation of the corresponding field lines using the gradient field.

e. import_curves.py – a Python script used in Blender to read the ASCII file generated by scivis.exe; it creates curves (spline curves, defined by control points) in the Blender scene and associates a particle emitter to the positive end of every curve.

f. cycle.sh – a shell script for Linux that reads the .pdb files and execute external programs and scripts (PyMOL, PDB2PQR, APBS, pyMLP.py and OBJCreator), saving the two .dx files for EP and MLP and the .obj file with the MLP values.

g. Render.py – another Python script used in Blender during the final step of rendering. For each frame it selects the corresponding mesh (imported by MLP.py), it assigns the appropriate image texture and renders the scene, according to the scene settings (camera, lights *etc.)*. It finally saves every rendered image as a .png file.