Methods developed for the implementation of a new version of BioBlender

Pablo Enmanuel Ramos-Bermúdez[1], Monica Zoppè[2], Tiziana Loni[3], Mario Pupo-Meriño[4] and Edisel Navas-Conyedo[5]

[1] *Department of Bioinformatics, University of Informatics Sciences (UCI), San Antonio Highway Km 2 ½, Reparto Torrens, La Lisa, Havana, 0000-0002-8439-948X, Cuba*
[2] *Department of Biosciences, University of Milan, Institute of Biophysics, CNR of Italy, Via Celoria 26, 20133 Milan, 0000-0003-4590-0718, Italy*
[3] *Independent Artist/Developer, Dundee, Scotland, 0000-0003-1629-5731, United Kingdom*
[4] *Department of Bioinformatics, University of Informatics Sciences (UCI), San Antonio Highway Km 2 ½, Reparto Torrens, La Lisa, Havana, 0000-0001-9130-0620, Cuba*
[5] *Computational Mathematics Study Center, University of Informatics Sciences (UCI), San Antonio Highway Km 2 ½, Reparto Torrens, La Lisa, Havana, 0000-0002-4648-7622, Cuba*

Abstract

Knowing the three-dimensional structure of macromolecules, their movements, and the way in which they interact with each other and with the environment contributes to a better understanding of the functioning of the cellular machinery. Within scientific visualization is the art and science of 3D animation, a technique used by various programs, such as the open-source 3D modeling software Blender. Based on this, the BioBlender software package was developed, a Blender module that allows the intuitive representation of surface properties of biomolecules such as Electrostatic Potential and Molecular Lipophilic Potential in a photorealistic way. BioBlender was developed and maintained by the Scientific Visualization Unit of the National Research Council of Italy, for an already expired version of Blender, so it is necessary to develop a new version that integrates the current changes in Blender, which constitutes the research objective. To do this, models extracted from the Protein Data Bank biological database, development tools such as the PyCharm integrated development environment and the exclusive use of the Python programming language have been used. Despite significant progress, work is still ongoing, to develop new methods and techniques to extend the tool and obtain a reasonable sequence of protein movements.

Keywords

BioBlender, Python programming, Blender, molecular visualization

1. Introduction

The field of scientific animation can be exploited to facilitate the study of complex phenomena, combining the tools of animation and infographics, with scientific information, and including concepts associated with art, which facilitate their understanding. Among the most widely used tools is the art and science of 3D animation, which consists of creating and animating objects in three-dimensional space (with simulated surfaces, skeletons and physical properties) in a virtual world, which can be "filmed". using virtual cameras and lights [1]. Several programs are available for this, including the commercial Maya/Autodesk, 3D Studio Max, and Softimage XSI packages (all from Autodesk) [2] and the open-source 3D software Blender [3]. The latter is one of the most powerful computer graphics

HCI 2022 - VIII Iberoamerican Conference of Human Computer Interaction, October 13–15, 2022, La Habana, Cuba
EMAIL: pabloenmanuel99@gmail.com (P. E. Ramos-Bermúdez); monica.zoppe@cnr.it (M. Zoppé); tialo@tiscali.it (T. Loni); mpupom@uci.cu (M. Pupo-Meriño); enavas@uci.cu (E. Navas-Conyedo)
ORCID: 0000-0002-8439-948X (P. E. Ramos-Bermúdez); 0000-0003-4590-0718 (M. Zoppé); 0000-0003-1629-5731 (T. Loni); 0000-0001-9130-0620 (M. Pupo-Meriño); 0000-0002-4648-7622 (E. Navas-Conyedo)

(CG) engines for managing 3D content, that is, creating, animating, texturing and rendering visual objects and scenes, whose main advantage over other similar tools such as those mentioned above is that it is a completely free application, which, being cross-platform and open source, allows users to introduce new functions, such as the introduction of new add-ons.

Based on the latter, BioBlender was developed, an open-source Add-On that is free to distribute and constitute a complete instrument to elaborate protein movement and achieve a photorealistic representation of complex properties of protein surfaces such as Electrostatic Potential (EP) and Molecular Lipophilic Potential (MLP) through a visual code, based on textures and particle effects [1] [4]. Several investigations demonstrate the potential of this module, "Intuitive representation of surface properties of biomolecules using BioBlender" [5] and "The representation of electrostatics for biological molecules" [6], are examples of to them.

One of the main problems of BioBlender is that, due to the constant changes made to the platform on which it is based, the method needs constant updating to keep pace with the principal program. BioBlender was developed by the SciVis group for an outdated version of Blender, and has been updated only to some extent, up to Blender version 2.6. Therefore, this research aims to develop a new version of BioBlender functional in current versions of Blender, that is version 3.1.

2. Materials and Methods

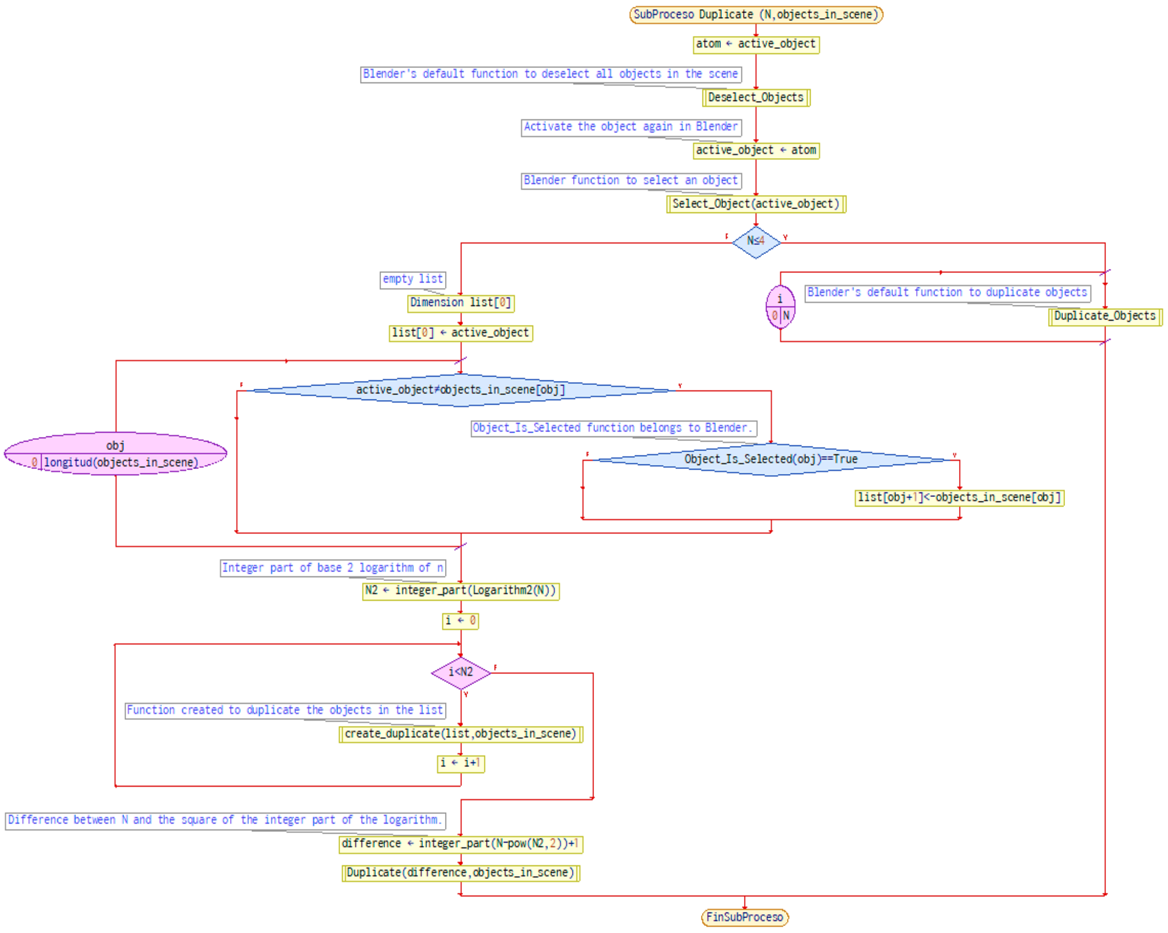
2.1. Migration to Python v3.10.2

The first step for the development of a new version of BioBlender functional in the version 3.1 of Blender, is the migration of the code from Python v2.6 to Python v3.10.2 onwards, obtaining at the same time, a restructuring of the script that couples all classes and functions. To achieve this, the PyCharm Community v2021.1.3 Integrated Development Environment (IDE) was used, where the objective of the research was carried out, where the source code was analyzed, and then divided into different modules, following the Python PEPs rules, new Python Improvement Proposals for each version, and new property and method changes exposed in the Blender v3.1 Python API documentation [7].

The incorporation of a Version Control System (VCS) to PyCharm allows developers to keep a strict control of their versions and to be able to safely return to each one of them when later problems arise in the development of the system [8]. This tool has been used for BioBlender version control, which has facilitated the modification and update of the code, since it has allowed having a history of changes or versions for reuse in case they are needed. The use of a VCS such as GitHub, the cloud-based service that implements Git [9], has allowed the collaboration of developers from different parts of the world and, in addition, has made it possible for researchers to test BioBlender and mention their concerns or errors obtained, facilitating the work of correcting errors in the new updates.

2.2. Import algorithm of atoms in the scene

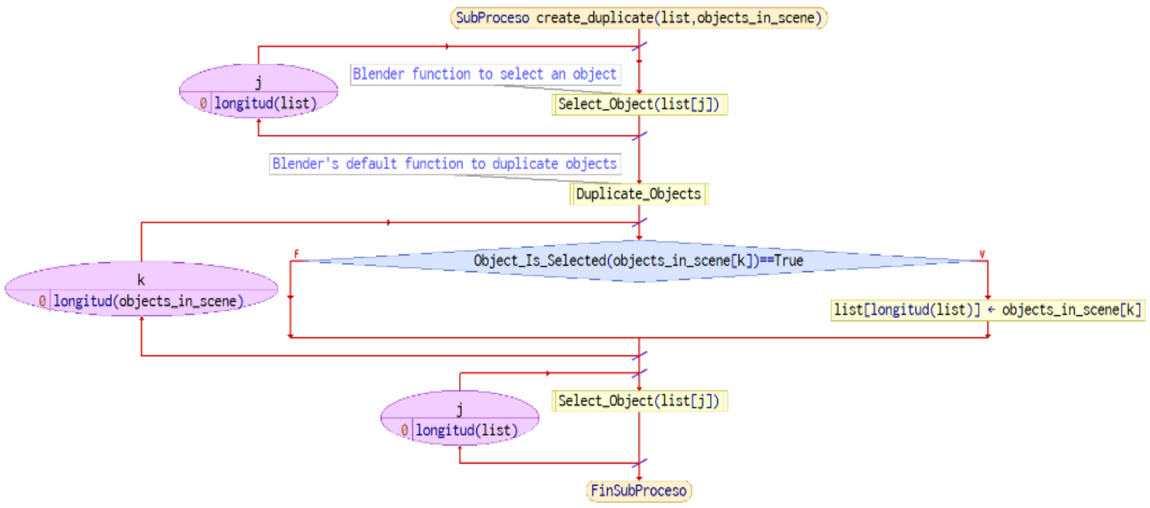
The process of importing the atoms from the PDB file into BioBlender depends on the duplication of objects provided by Blender, and this process requires an amount of time proportional to the number of atoms of the molecule, behaving exponentially. The default Blender function for such an action does not have a parameter that allows the duplication of a certain number N of objects. For this reason, an algorithm was developed so that the atom duplication process did not occur sequentially, one by one, which caused the waiting time to increase, but is achieved by doubling the existing atoms, thus speeding the process. The Figure 1 shows the flowchart of the generated algorithm.



SubProceso Duplicate (N,objects_in_scene)
atom ← active_object
Blender's default function to deselect all objects in the scene
Deselect_Objects
Activate the object again in Blender
active_object ← atom
Blender function to select an object
Select_Object(active_object)
N≤4
empty list
Dimension list[0]
list[0] ← active_object
Blender's default function to duplicate objects
Duplicate_Objects
i
0 N
obj
0 longitud(objects_in_scene)
active_object≠objects_in_scene[obj]
Object_Is_Selected function belongs to Blender.
Object_Is_Selected(obj)==True
list[obj+1]<-objects_in_scene[obj]
Integer part of base 2 logarithm of n
N2 ← integer_part(Logarithm2(N))
i ← 0
i<N2
Function created to duplicate the objects in the list
create_duplicate(list,objects_in_scene)
i ← i+1
Difference between N and the square of the integer part of the logarithm.
difference ← integer_part(N-pow(N2,2))+1
Duplicate(difference,objects_in_scene)
FinSubProceso

Figure 1: Atom duplication algorithm

Next, the Figure 2 shows the flowchart of function developed for the duplication of the objects in a list, called in the algorithm shown in Figure 1.



SubProceso create_duplicate(list,objects_in_scene)
j
0 longitud(list)
Blender function to select an object
Select_Object(list[j])
Blender's default function to duplicate objects
Duplicate_Objects
k
0 longitud(objects_in_scene)
Object_Is_Selected(objects_in_scene[k])==True
list[longitud(list)] ← objects_in_scene[k]
j
0 longitud(list)
Select_Object(list[j])
FinSubProceso

Figure 2: Flowchart of the "create_duplicate" function

2.3. Software Testing

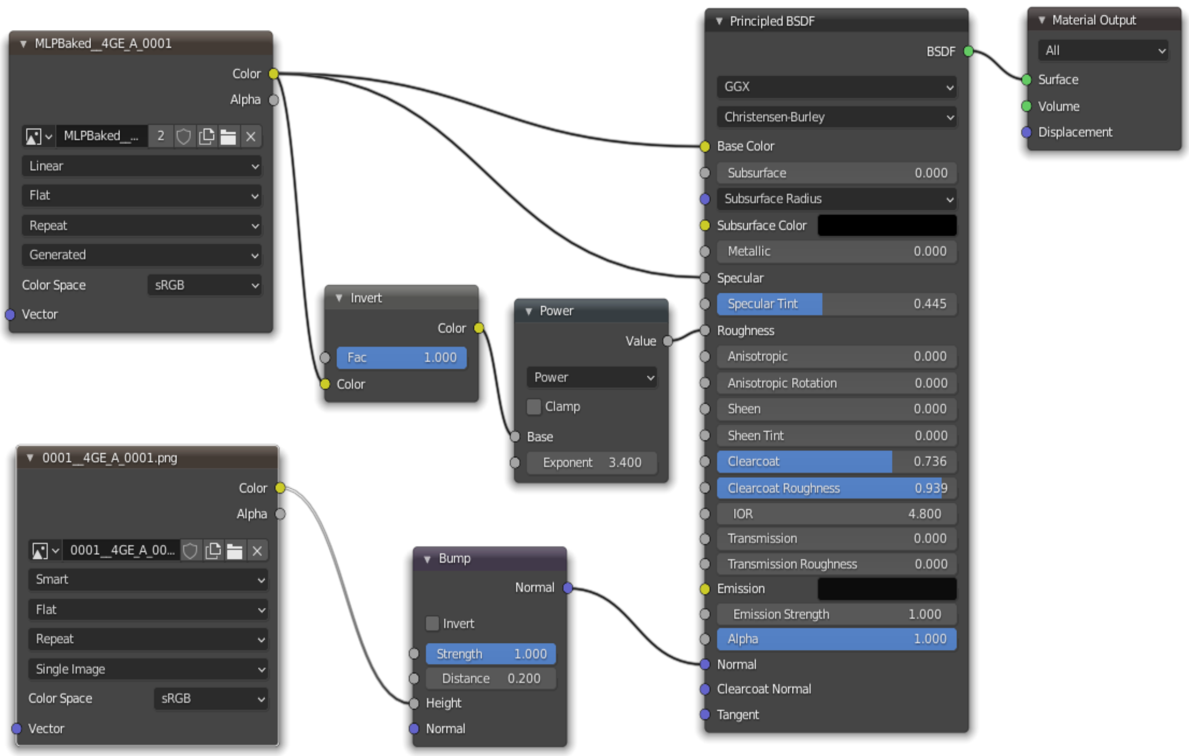
The software testing process is one of the fundamental aspects to measure the quality status of a computer system. In the case of BioBlender, functional tests were applied, using White Box methods for the evaluation of internal functionalities, and Black Box methods for the evaluation of external functionalities. As support for the functional tests, through the Black Box method, Usability tests were applied through the Inspection method.

The equivalence partitioning technique of the Black Box method was used to divide the input data into different types, in order to determine various test cases and check specific functions with each of them. In addition to this, the Heuristic technique of the Inspection method was used to evaluate usability and thus verify previously established usable design principles (heuristic principles). These principles are guidelines that establish requirements that the design must meet in order to facilitate its understanding and use by the end user. As a support tool for this type of test, the checklist established by the Department of Quality of the University of Informatics Sciences (UCI), defined as a list of questions, in the form of a questionnaire, used to verify the degree of compliance with certain heuristic rules established a priori for a given purpose.

Through the White Box method, the internal structure of the code was deepened, monitoring the source code as the test cases are executed, specifically determining the instructions that have been executed by them. Logical coverage tests such as path coverage and Condition/Decision coverage were used, in addition to local data structure. Through these types of tests, a study was made of all the variables of the module, avoiding the declaration of variables with the same name, declared locally or globally, analyzing their behavior. By means of path coverage, enough test cases were assigned to execute all the BioBlender instructions, checking each of its functions and obtaining from them the errors produced during the executions, in order to later determine their possible solution.

2.4. Visualization of the Lipophilic Potential of the Molecule (MLP)

A restructuring of the code of the module for the calculation of the MLP was carried out to bake the texture using vertex colors, by mapping the molecular surface, projecting a three-dimensional mesh (x, y, z) in a two-dimensional image (x, y), where x and y represent the coordinates of the texture space, U and V respectively. The generated texture is merged with a default noise texture to be used in Blender's Shader Editor tool to create materials, using the shader nodes, which generate values, vectors and colors. In Figure 3 you can see the nodes that generate a photorealistic surface.



MLPBaked_4GE_A_0001
Color
Alpha
MLPBaked_... 2
Linear
Flat
Repeat
Generated
Color Space sRGB
Vector

0001_4GE_A_0001.png
Color
Alpha
0001_4GE_A_00...
Smart
Flat
Repeat
Single Image
Color Space sRGB
Vector

Invert
Color
Fac 1.000
Color

Power
Value
Power
Clamp
Base
Exponent 3.400

Bump
Normal
Invert
Strength 1.000
Distance 0.200
Height
Normal

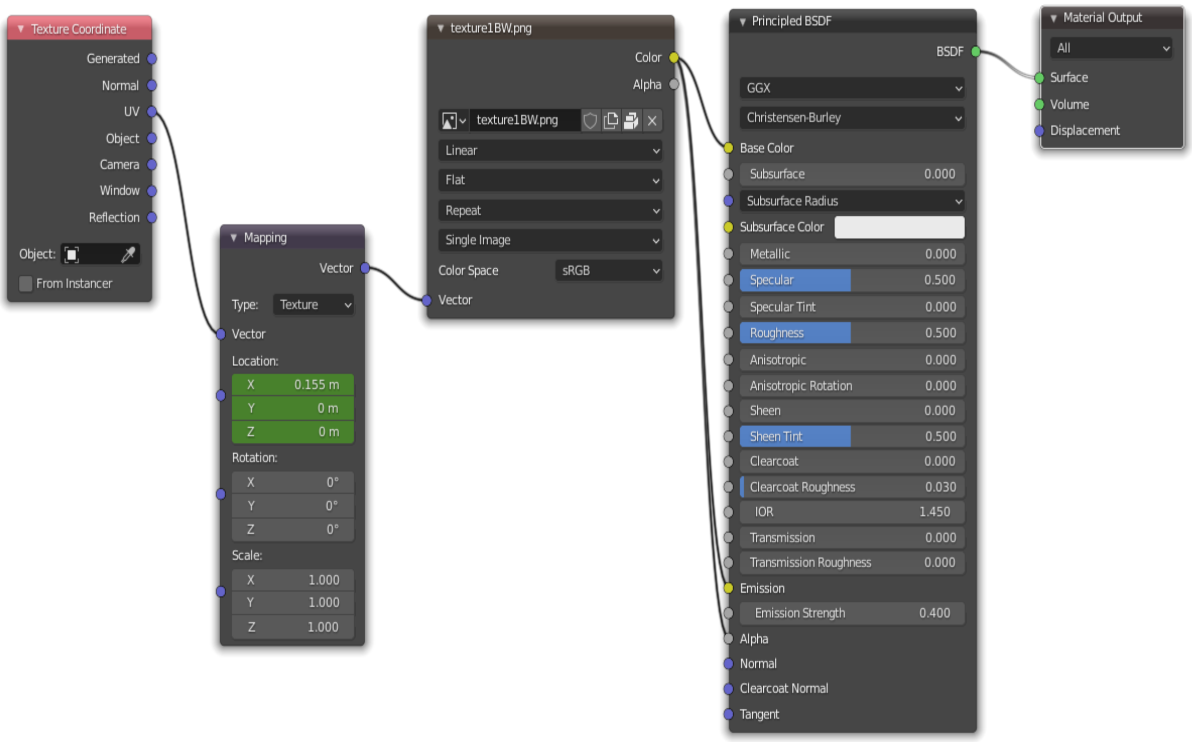
Principled BSDF
BSDF
GGX
Christensen-Burley
Base Color
Subsurface 0.000
Subsurface Radius
Subsurface Color
Metallic 0.000
Specular
Specular Tint 0.445
Roughness
Anisotropic 0.000
Anisotropic Rotation 0.000
Sheen 0.000
Sheen Tint 0.000
Clearcoat 0.736
Clearcoat Roughness 0.939
IOR 4.800
Transmission 0.000
Transmission Roughness 0.000
Emission
Emission Strength 1.000
Alpha 1.000
Normal
Clearcoat Normal
Tangent

Material Output
All
Surface
Volume
Displacement

Figure 3: Generated material to represent the MLP on the molecular surface

2.5. Representation of Electrostatic Potential (EP)

As of Blender 2.8, the particles coming from an emitter are not rendered, therefore, the curves along which the particles traveled have been converted to an object, and have been given a material, created in the aforementioned Shadow Editor. A black and white image texture (texture1BW in the Figure 4) was designed and assigned to the curve mesh. The Texture Coordinate node was used to obtain the UV texture coordinates of the curve, used as the input vector of the Mapping node. In this node, the value of the location on the X axis varies because it is animated on the basis of time (green color in the Figure 4), allowing the created texture to move along the curve and simulate particles by passing over them (from positive to negative). The Figure 4 shows the nodes that produce the particles effect.


Texture Coordinate
Generated
Normal
UV
Object
Camera
Window
Reflection
Object:
From Instancer

Mapping
Vector
Type: Texture
Vector
Location:
X 0.155 m
Y 0 m
Z 0 m
Rotation:
X 0°
Y 0°
Z 0°
Scale:
X 1.000
Y 1.000
Z 1.000

texture1BW.png
Color
Alpha
texture1BW.png
Linear
Flat
Repeat
Single Image
Color Space sRGB
Vector

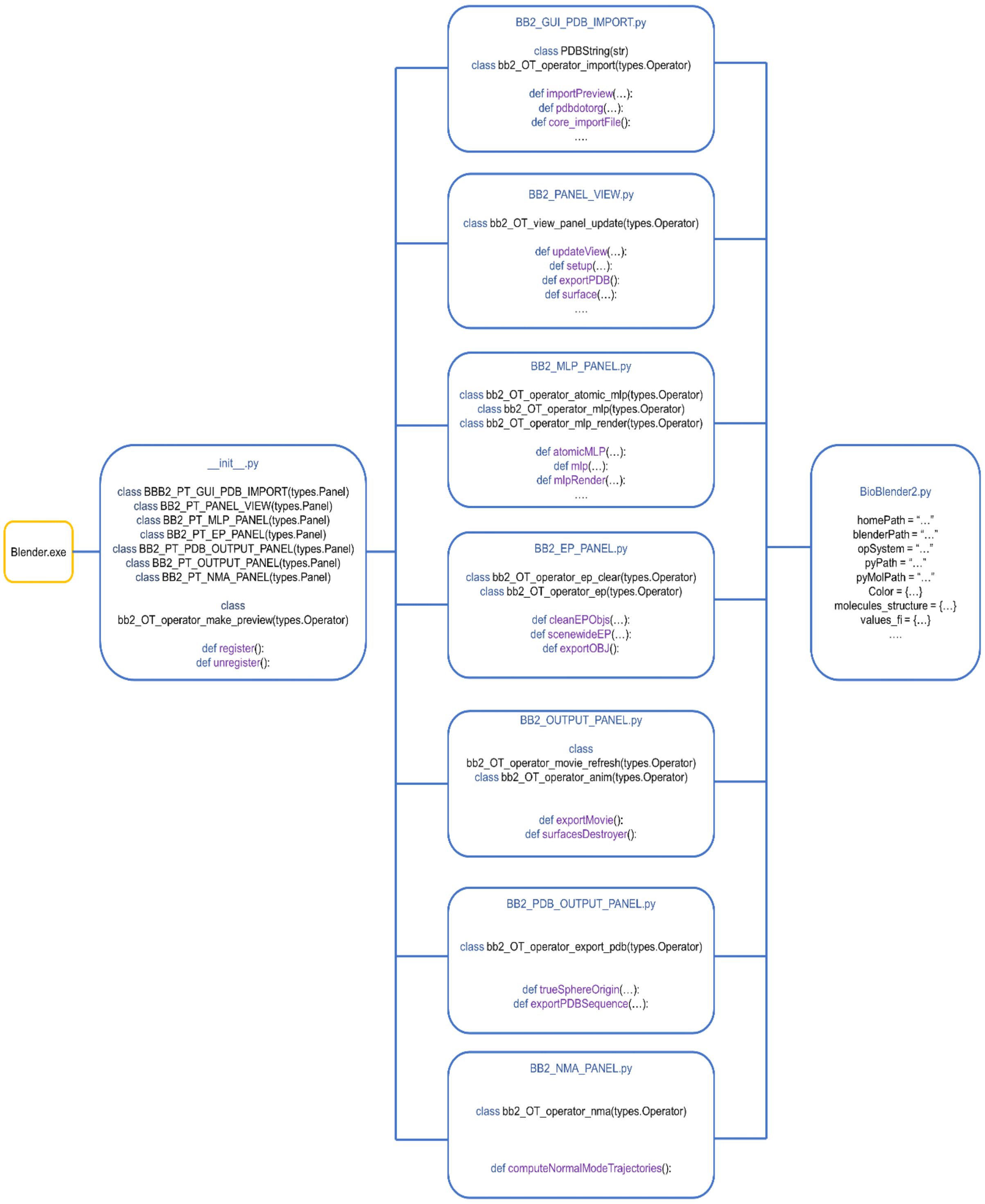
Principled BSDF
BSDF
GGX
Christensen-Burley
Base Color
Subsurface 0.000
Subsurface Radius
Subsurface Color
Metallic 0.000
Specular 0.500
Specular Tint 0.000
Roughness 0.500
Anisotropic 0.000
Anisotropic Rotation 0.000
Sheen 0.000
Sheen Tint 0.500
Clearcoat 0.000
Clearcoat Roughness 0.030
IOR 1.450
Transmission 0.000
Transmission Roughness 0.000
Emission
Emission Strength 0.400
Alpha
Normal
Clearcoat Normal
Tangent

Material Output
All
Surface
Volume
Displacement

Figure 4: Material generated to simulate particles that make their trajectory along the curves resulting from the calculation of the EP

3. Results and Discussion

BioBlender code was made modular, dividing the original script into several modules. Each of them contains a specific function, providing greater control and better access to classes and functions. The Figure 5 shows a general perspective of the source code, where each of the resulting modules can be seen.



```
__init__.py

class BBB2_PT_GUI_PDB_IMPORT(types.Panel)
class BB2_PT_PANEL_VIEW(types.Panel)
class BB2_PT_MLP_PANEL(types.Panel)
class BB2_PT_EP_PANEL(types.Panel)
class BB2_PT_PDB_OUTPUT_PANEL(types.Panel)
class BB2_PT_OUTPUT_PANEL(types.Panel)
class BB2_PT_NMA_PANEL(types.Panel)

class
bb2_OT_operator_make_preview(types.Operator)

def register():
def unregister():
```

```
BB2_GUI_PDB_IMPORT.py

class PDBString(str)
class bb2_OT_operator_import(types.Operator)

def importPreview(...):
def pdbdotorg(...):
def core_importFile():
....
```

```
BB2_PANEL_VIEW.py

class bb2_OT_view_panel_update(types.Operator)

def updateView(...):
def setup(...):
def exportPDB():
def surface(...):
....
```

```
BB2_MLP_PANEL.py

class bb2_OT_operator_atomic_mlp(types.Operator)
class bb2_OT_operator_mlp(types.Operator)
class bb2_OT_operator_mlp_render(types.Operator)

def atomicMLP(...):
def mlp(...):
def mlpRender(...):
....
```

```
BB2_EP_PANEL.py

class bb2_OT_operator_ep_clear(types.Operator)
class bb2_OT_operator_ep(types.Operator)

def cleanEPObjs(...):
def scenewideEP(...):
def exportOBJ():
```

```
BB2_OUTPUT_PANEL.py

class
bb2_OT_operator_movie_refresh(types.Operator)
class bb2_OT_operator_anim(types.Operator)

def exportMovie():
def surfacesDestroyer():
```

```
BB2_PDB_OUTPUT_PANEL.py

class bb2_OT_operator_export_pdb(types.Operator)

def trueSphereOrigin(...):
def exportPDBSequence(...):
```

```
BB2_NMA_PANEL.py

class bb2_OT_operator_nma(types.Operator)

def computeNormalModeTrajectories():
```

```
BioBlender2.py

homePath = "..."
blenderPath = "..."
opSystem = "..."
pyPath = "..."
pyMolPath = "..."
Color = {...}
molecules_structure = {...}
values_fi = {...}
....
```

Figure 5: Code Architecture: Overview

Once the above was achieved, various tests were carried out to measure the import time of the atoms, with the algorithm mentioned in previous sections, and without the algorithm. Remarkable results were obtained in terms of the time required for the import of a large molecule, which was previously very high using a common desktop computer. In Figure 6, it is possible to observe how this process behaves.

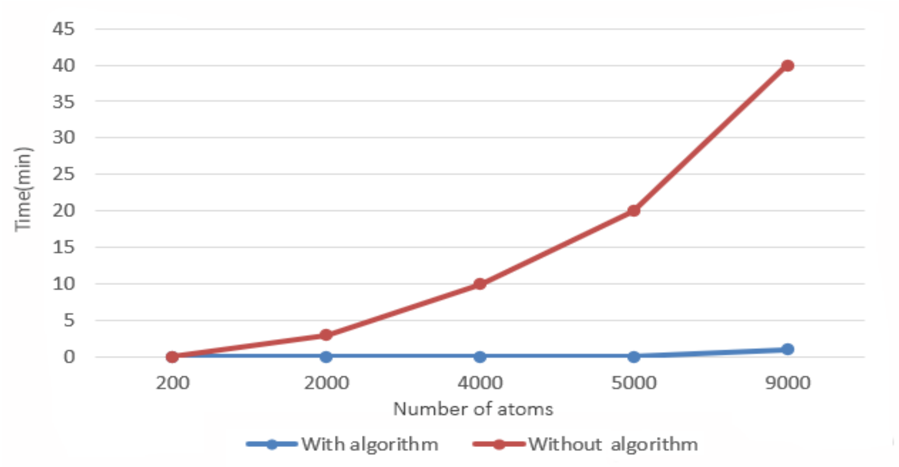


Figure 6: Comparison of the duration time of the import of the atoms of a molecule

BioBlender offers the possibility of viewing only what the user wants to see, in a simple and fast way, without the need for prior knowledge of the platform. The Figure 7 shows an example of the different ways of visualizing a protein in 3D space, depending on the option selected in BioBlender, based on the PDB format file from the Protein Data Bank biological database [10] and on surface calculation performed by PyMol [11].

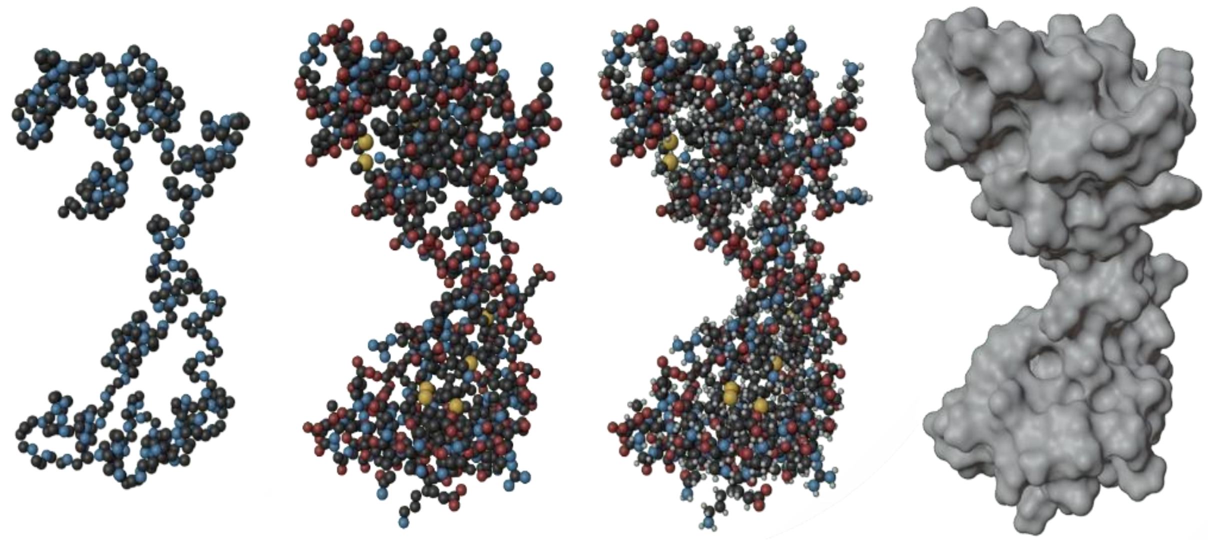


Figure 7: Calcium-free Calmodulin protein (CaM, PDB 1CFC) in 3D space. Main chain → + Side chains → +Hydrogens (only if imported) → Protein surface

The MLP depends on the properties of the atoms immediately below the surface. These are integrated through several steps [1], and finally rendered, using the method exposed in the previous section. The results obtained are shown in Figure 8.

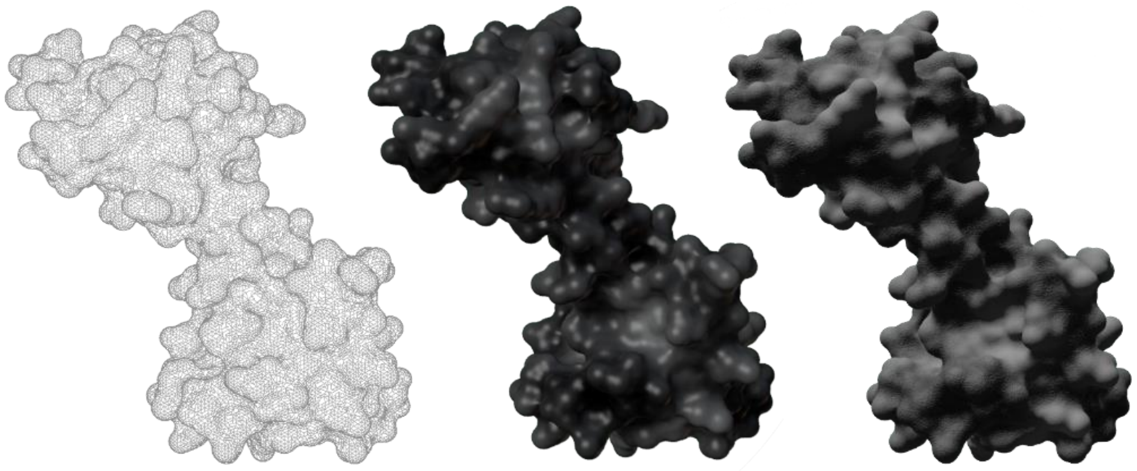


Figure 8: CaM protein. Mesh → MLP display as gray levels (white is very hydrophobic, black is very hydrophilic) → MLP display using BioBlender texture

The representation of the EP as particles moving along a trajectory, reproduced in time, is easily interpreted and conveys the idea of polarity of the charged areas of a biomolecule. In still images, linear particles cannot convey the polarity of charged areas of molecules, and should be replaced by shapes that convey information about direction, such as small comets or arrows. In Figure 9, you can see the example of Alcohol Dehydrogenase from Drosophila Lebanonensis (PDB 1A4U) on the left side and Actin (PDB 1ATN) on the right, as a single frame of an animated representation.

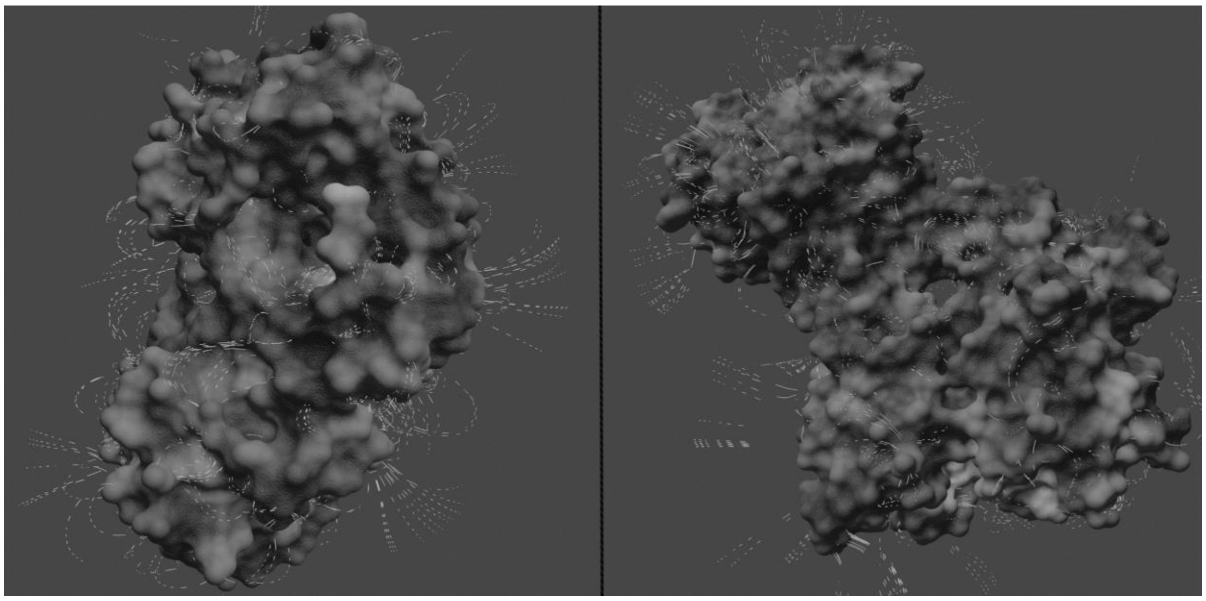


Figure 9: Single-frame representation of the animation

Visually representing the surface features of proteins in motion allows immediate insight into the behavior of their molecular features. When proteins move, they change the location of their atoms, according to physic-chemical constraints. When a PDB file contains several “Models” or conformations for a protein, using BioBlender it is possible to obtain a sequence of the movement of the protein. The molecules can be rendered according to the different forms of representation previously exposed. Next, in Figure 10 some of the models of NMR Structure of Trp-Cage Miniprotein Construct TC5b (PDB 1L2Y) are visualized.

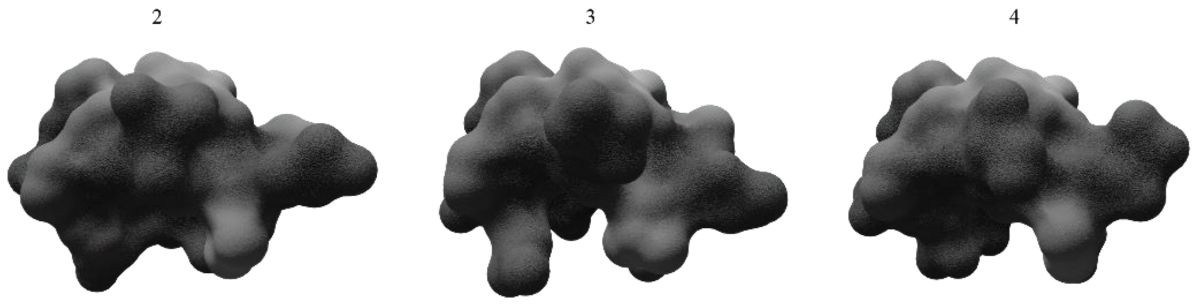


Figure 10: Conformations 2, 3 and 4 of NMR Structure of Trp-Cage Miniprotein Construct TC5b (PDB 1L2Y)

To evaluate the quality of the internal and external logic of BioBlender, 2 iterations of software tests were carried out, until satisfactory results were achieved, taking into account the correct behavior of the system in different situations. In summary:

- Functional tests applying the Black Box test technique: a total of 9 defects were obtained in the first iteration, 6 classified in the group of Options that do not work, 2 of Validation and 1 of Spelling.
- Usability tests using the checklist to apply the Heuristic technique: a total of 10 defects were obtained in the first iteration, 5 of these belong to Visibility of the system, 4 to Aesthetics and minimalist design, and 1 to Consistency and standards.
- Functional tests applying the White Box test technique: 8 defects were obtained in the first iteration, 5 belong to Error in logic variable and 3 are classified as Loop variables modified inappropriately.

In the three types of tests, the defects detected in the first iteration were corrected, which meant that in the second iteration no defects were detected in any of the tests carried out, thus demonstrating the quality of the proposed components and, in turn, meeting the system requirements.

Today, in summer 2022, the previously exposed functions are available, allowing their use in a recent version of Blender (version 3.1), based on Python version 3.10.2. Said result contributes to the continuous development of this Add-On package, enabling the creation and implementation of future functions that add to the interpretation in three-dimensional space of surface properties such as MLP and EP, important for understanding the behavior of molecules within their environment. This is an important contribution in the current scenario, where the advancement of biological technologies is essential in this era, marked by deaths and diseases caused by a wide variety of pathogens.

4. Conclusions

The evidence presented together with the results of the software tests carried out, demonstrate compliance with the research objectives. That said, it is possible to represent biomolecules in 3D space using the version 3.1 of Blender and to build surface properties such as Molecular Lipophilic Potential and Electrostatic Potential, providing a better understanding of important biochemical information such as hydropathy or charges. The migration of the code to Python v3.10.2 onwards, an important step to achieve the reincorporation of the module to Blender, constitutes, in turn, the basis for future implementations, that may expand the features available in BioBlender.

In addition to this, the research does not stop at these results, since the platform on which this software package is developed is constantly upgraded and expanded, which implies studying it continuously to keep pace with new animation technologies.

The development codes, installation manual and user guide can be found at https://github.com/PabloEnmanuelRamos/BioBlender21.git.

5. References

[1] Raluca Andrei, Marco Callieri, Maria Zini, Tiziana Loni, Giuseppe Maraziti, Mike Chen Pan, and Monica Zoppé. “Bioblender: a software for intuitive representation of surface properties of biomolecules”. Computing Research Repository - CORR, 01 (2010).

[2] Autodesk Homepage, 2021. URL: https://www.autodesk.es/.

[3] Blender Foundation Homepage, 2022. URL: https://www.blender.org.

[4] Maria Zini, Yuri Porozov, Tiziana Loni, Raluca Andrei, and Monica Zoppé. “Use of bioblender for all atom morphing of protein structures”. EMBnet.journal, 18:124 (2012). doi: 10.14806/ej.18.A.413.

[5] Raluca Andrei, Marco Callieri, Maria Zini, Tiziana Loni, Giuseppe Maraziti, Mike Chen Pan, and Monica Zoppé. “Intuitive representation of surface properties of biomolecules using bioblender”. BMC bioinformatics, 13 Suppl 4:S16 (2012). doi: 10.1186/1471-2105-13-S4-S16.

[6] Monica Zoppé and Tiziana Loni. “The Representation of Electrostatics for Biological Molecules”. pages 215– 225. ISBN 978-3-319-12210-6 (2015). doi: 10.1007/978-3-319-12211-3.

[7] Blender 3.1 Python API Documentation Homepage 2021. URL: https://docs.blender.org/api/3.1/.

[8] Luis Wanumen and Wanumen Silva. “Los sistemas de control de versiones the systems of control of versions”. Volumen 5 (2008) :63–72, 11.

[9] Software Freedom Conservancy, 2022. URL: Git Homepage, https://git-scm.com.

[10] Frances Bernstein, Thomas Koetzle, Graheme Williams, Edgar Meyer, Michael Brice, John Rodgers, Olga Kennard, Takehiko Shimanouchi, and Mitsuo Tasumi. “The protein data bank: A computer-based archival file for macromolecular structures”. European journal of biochemistry / FEBS, 80:319–24 (1977). doi: 10.1016/S0022-2836(77)80200-3.

[11] W.L. DeLano. “The pymol molecular graphics system. Proteins”, 30 (2002) :442–454, 01.