Bio Visualisation with Blender and MembraneEditor Part 6
Blender Special Topics

Konstanz Research School Chemical Biology (KoRS-CB) Workshop
Björn Sommer, University of Konstanz & Mehmood Ghaffar, IPK Gatersleben
Version 25.02.2019

Forum:
http://www.cellmicrocosmos.org/Cmforum/viewforum.php?f=63

Actual Version of Blender:
http://www.blender.org

Actual Version of CELLmicrocosmos 2.2 MembraneEditor:
http://Cm2.CELLmicrocosmos.org

Actual Version of Blender:
http://www.blender.org

Here, Blender 2.79 is used.

Target

This tutorial describes how molecular data can be imported into Blender. This tutorial is based on the tutorial “Bio Visualisation with Blender and MembraneEditor Part 3 CELLmicrocosmos MembraneEditor”. It introduces

- User Interface Python Scripting,
- Atomic Blender and how to import atomic structures from the MembraneEditor to Blender (Repetition: also found in the MembraneEditor tutorial).

Abbreviation

RMB Right Mouse Button

LMB Left Mouse Button

! For using most of the shortcuts discussed in this tutorial, you have to be sure that the mouse cursor is WITHIN the view port of the 3D View !

User Interface Python scripting

In this tutorial, you will learn:

How to create a simple Panel at different locations in the blender environment.

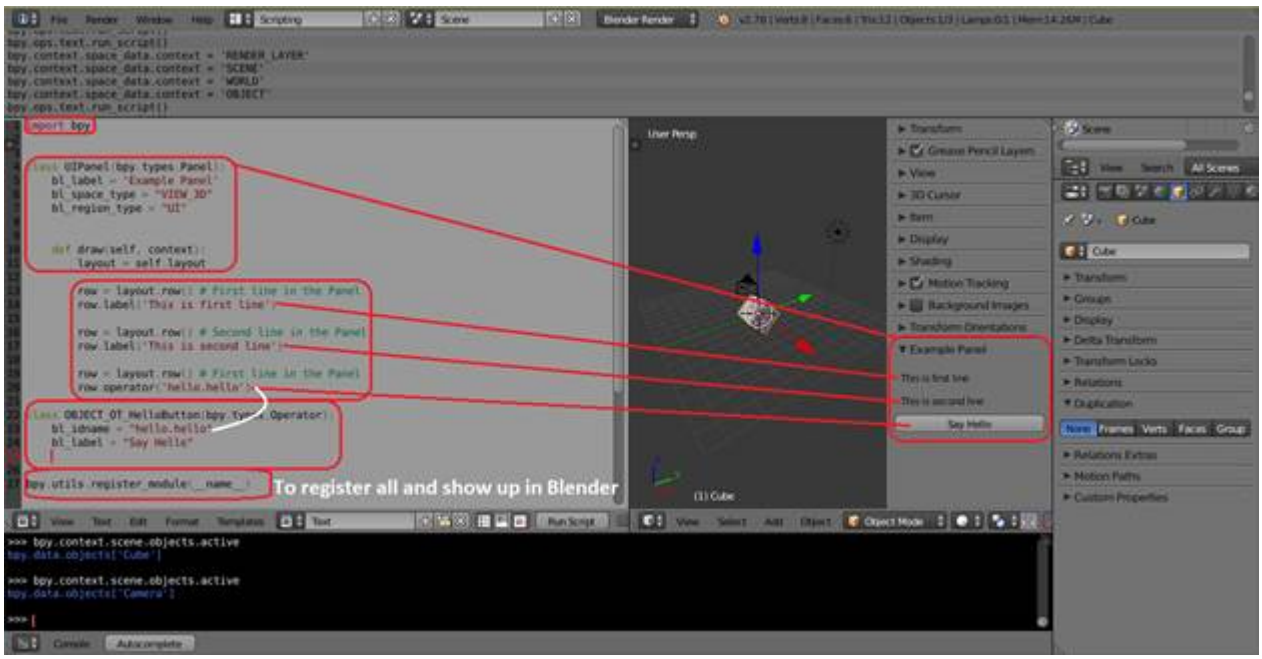
Fill the Panel with buttons, labels and other properties.

The following code will create a simple panel in view_3d space and in the user interface section.

```
class UIPanel(bpy.types.Panel):
  bl_label = 'Example Panel'
  bl_space_type = "VIEW_3D" # Defines the location of Panel
  bl_region_type = "UI"
#space_type_VIEW_3D: TOOLS, TOOLS_PROPS, UI
#space_type_PROPERTIES: type:WINDOW, context: object,
#material,etc

def draw(self, context):
layout = self.layout

# To register UIPanel
bpy.utils.register_module(__name__)
```



```
class OBJECT_OT_HelloButton(bpy.types.Operator):
  bl_idname = "hello.hello" #reference in code
  bl_label = "Say Hello" # text appear on the button

  def execute(self, context): #This block of code will be executed when the button is clicked
    return{'FINISHED'}
```

At this point you learn how to create a panel. And how to insert a button in it.

Now, we put some action to these buttons.

We define some buttons and assign each a translation operation to move active scene object along XYZ axes.

The code for these individual buttons is as follows:

```
import bpy
import sys

# store scene objects to a variable object
object = bpy.context.scene.objects

def translate_XYZ(*argv): # Translate along xyz axes
  if object.active:
    bpy.ops.transform.translate(value=(argv))


class UIPanel(bpy.types.Panel):
  bl_label = 'Example Panel'
  bl_space_type = "VIEW_3D"
  bl_region_type = "UI"

  def draw(self, context):
    layout = self.layout

    row = layout.row() # First line in the Panel
    row.label ('Move LEFT and RIGHT')
    row = layout.row()
    row.operator('translate.p_x')
```

```
    row.operator('translate.n_x')

    row = layout.row() # Second line in the Panel
    row.label('Move UP and Down')

    row = layout.row() # First line in the Panel
    row.operator('translate.p_y')
    row.operator('translate.n_y')


class Translate_PX(bpy.types.Operator): # move along +X Axis
  bl_idname = "translate.p_x"
  bl_label = "RIGHT"
  name = bpy.props.StringProperty()
  def execute(self, context):
    translate_XYZ(1,0,0)
    return{'FINISHED'}

class Translate_PY(bpy.types.Operator): # move along +Y Axis
  bl_idname = "translate.p_y"
  bl_label = "UP"
  name = bpy.props.StringProperty()
  def execute(self, context):
    translate_XYZ(0,1,0)
    return{'FINISHED'}

class Translate_NX(bpy.types.Operator): # move along -X Axis
  bl_idname = "translate.n_x"
  bl_label = "LEFT"
  name = bpy.props.StringProperty()
  def execute(self, context):
    translate_XYZ(-1,0,0)
    return{'FINISHED'}

class Translate_NY(bpy.types.Operator): # move along -Y Axis
```

```
    bl_idname = "translate.n_y"
    bl_label = "DOWN"
    name = bpy.props.StringProperty()
    def execute(self, context):
        translate_XYZ(0,-1,0)
        return{'FINISHED'}
bpy.utils.register_module(__name__)
```

➔ Code for individual buttons

➔ Code of library import

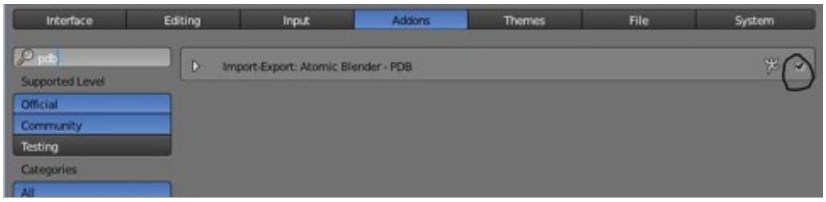
➔ Code for functions

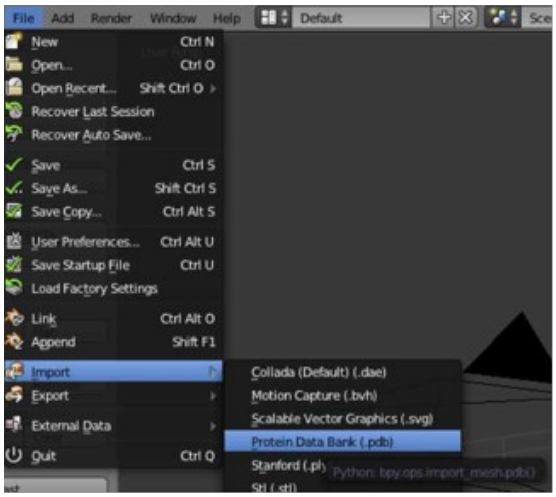
➔ Code for Panel layout

Atomic Blender: The Standard PDB Importer for Blender

Now, we want to import the PDB file into Blender. For this purpose, Blender comes with a special plug-in which has to be activated.

Activate it
here: Atomic
Blender





Select now File → Import → Protein Data Bank (.pdb)

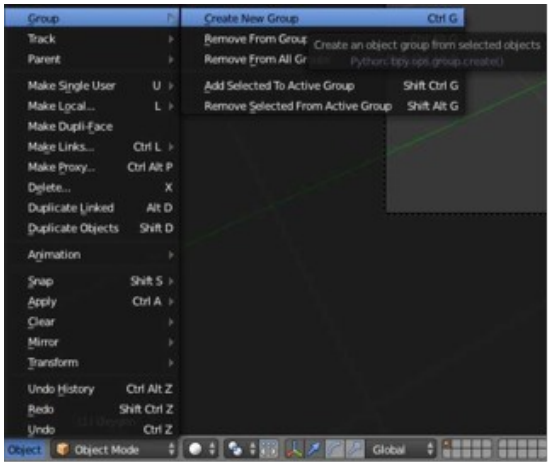
Okay, guess now which PDB file you will import? Yes, take the one from the MembraneEditor. In addition, in the appearing window you will see a lot of options, but use the basic options now:

You will see now that Blender will need some time to import this structure. So if you generated the 100 x 100 membrane, it is a good idea to do it on a fast computer, such as an i7 with 8 GB Ram. If you find out that your computer is not fast enough, just generate a smaller membrane, let

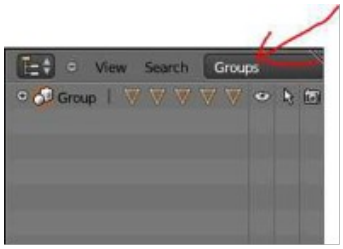
us say 50 x 50. This should solve the problem.

Directly after importing the structure it is a good idea to scale the membrane done to a size fitting to your environment. Just press “S” and scale it down.

In the Outliner, you see the different parts of the molecule. It might be a good idea to group them, so that it is no problem to select the whole molecule with one click. Select all segments part of the membrane by holding shift and LM all segment parts:

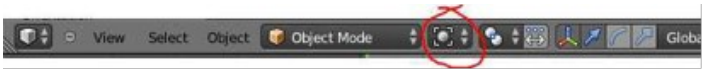


Then select Object → Group → Create New Group



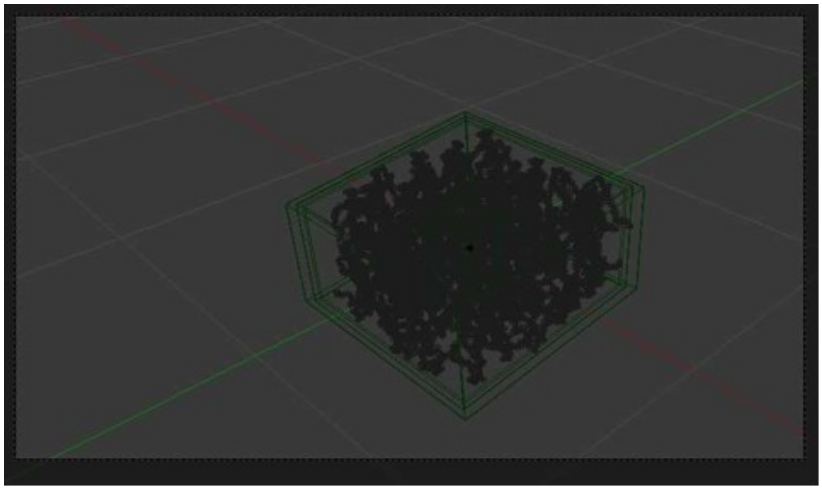
If you select now “Groups” in the Outliner:

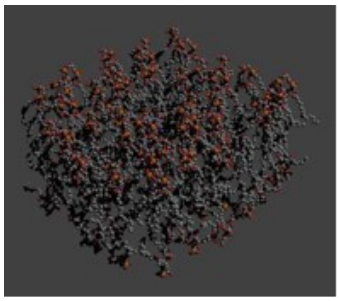
You can select the whole membrane with one click.



Because the visualization of this many spheres needs a lot of performance, it would be a good idea to switch to Bounding Box mode.

It will look like this and the navigation will be much faster.





Now, do the rendering by pressing F12.

In the previous tutorials you have learnt a lot of ways to improve the visualization, do it, if you want to make a good visualization.