ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 230 за 2018 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Филиппов С.В.

Программная платформа Blender как среда моделирования объектов и процессов естественно-научных дисциплин

Рекомендуемая форма библиографической ссылки: Филиппов С.В. Программная платформа Blender как среда моделирования объектов и процессов естественно-научных дисциплин // Препринты ИПМ им. М.В.Келдыша. 2018. № 230. 42 с. doi:10.20948/prepr-2018-230

URL: http://library.keldysh.ru/preprint.asp?id=2018-230

Ордена Ленина
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В. Келдыша
Российской академии наук

С.В. Филиппов

Программная платформа Blender как среда моделирования объектов и процессов естественно-научных дисциплин

Москва — 2018

Филиппов С.В.

Программная платформа Blender как среда моделирования объектов и процессов естественно-научных дисциплин

Дано вводное описание Blender Python API, необходимое для численного моделирования в среде открытой программной платформы трёхмерного моделирования, анимации и рендеринга — Blender. Рассмотрены базовые методы 3D-моделирования объектов исследований на примере построения динамических молекулярных моделей и моделей фрагмента дна Северного Ледовитого океана по расчётным и внешним (экспериментальным) данным, представленным как численно, так и в форме графических изображений. Приведены примеры программ на языке Python, демонстрирующие основные методы работы с базовыми данными. Описана комплексная задача по созданию аналитической визуализации динамической молекулярной модели бета-2 адренорецептора по данным молекулярно-динамического исследования.

Ключевые слова: 3D, моделирование, аналитическая визуализация, Blender, Python, API, бета-2 адренорецептор, молекулярная динамика.

Blender software platform as an environment for modeling objects and processes of science disciplines

An introductory description of the Blender Python API is given, which is necessary for numerical simulation in the open source software platform of three-dimensional modeling, animation and rendering — Blender. The basic methods of 3D-modeling of research objects are considered on the example of building dynamic molecular models and models of the ocean bed fragment of the Arctic Ocean using calculated and external (experimental) data presented both numerically and in the form of graphic images. Examples of programs in the Python language are given, demonstrating the basic methods of working with basic data. The complex task of creating analytical visualization of the dynamic molecular model of the beta-2 adrenoreceptor according to the data of molecular dynamics research is described.

Key words: 3D, modeling, analytical visualization, Blender, Python, API, beta-2 adrenoreceptor, beta-2 adrenergic receptor, molecular dynamics.

Работа выполнена при поддержке Российского фонда фундаментальных исследований, проекты 18-07-00354 и 18-07-00223.

Оглавление

Введение

Blender представляет собой программный комплекс для трёхмерного моделирования объектов и процессов, а также рендеринга, визуализирующего смоделированные сцены. Помимо открытого программного кода и, как следствие, бесплатности, данный пакет отличается большой универсальностью и самодостаточностью, поскольку содержит практически исчерпывающий набор программных инструментов, необходимых для обеспечения всей технологической цепочки динамической компьютерной визуализации любого уровня сложности, начиная от инструментов полигонального моделирования, средств имитации материалов и заканчивая средствами для композитинга, видеомонтажа и даже построения автономных интерактивных программ [1].

Таким образом, благодаря вышеприведённому набору характеристик Blender может быть использован в качестве универсальной программной среды для моделирования объектов и процессов естественно-научных исследований. На уровень полноценного средства моделирования, сопоставимого по возможностям с MathLab и ему подобными, Blender выводит поддержка им интерпретируемого языка программирования Python [2], который помимо обеспечения полного контроля над всем инструментарием данной платформы обеспечивает и доступ к обширной библиотеке модулей, написанной научным сообществом [3-6].

При этом Blender выгодно отличается от традиционных сред математического моделирования не только перечнем встроенных инструментов и средств, но и удобным пользовательским графическим интерфейсом к ним, существенно эко-

номящим время исследователя, а также снижающим порог вхождения в данную сферу деятельности.

Широкому применению программной платформы Blender в качестве среды математического моделирования, на наш взгляд, препятствует лишь отсутствие систематизированной документации и учебных материалов по Blender Python API, что вообще свойственно свободным проектам с открытым исходным кодом. Этот недостаток и призвана хотя бы отчасти устранить данная публикация.

Основные сведения о Blender Python API

Внутренним скриптовым языком программирования Blender[1] является Python 3.x. Его интерпретатор встроен в пакет и не нуждается в дополнительной инсталляции. Сам пакет при этом выполняет роль интегрированной среды разработки — IDE (Integrated Development Environment), предоставляя разработчику такой базовый инструментарий, как текстовый редактор с подсветкой синтаксиса и интерактивная Python-консоль. Оба эти средства поддерживают функцию автодополнения/автозавершения (**Ctrl+Пробел**), обеспечивающую не только удобство, скорость и точность ввода, но и выполняющую функцию справочника доступных методов и параметров API.

Гибко настраиваемый графический интерфейс Blender обеспечивает комфортный рабочий процесс (рис. 1), не нуждающийся в дополнительном стороннем инструментарии.

Работа с Python API возможна как в интерактивной Python-консоли (Python Console), так и с помощью текстового редактора (Text Editor) [7].

К настоящему времени можно говорить о том, что API пакета Blender является устоявшимся[2], несмотря на продолжающееся интенсивное развитие пакета[3]. Однако некоторые его части всё ещё не задокументированы именно благодаря последнему обстоятельству.

Все элементы моделируемой сцены хранятся во внутренних структурах данных, организованных в виде именованных кортежей (коллекций), что позволяет получать к ним доступ с помощью строкового ключа, который более надёжен, нежели индекс [7, 8].

Доступ к атрибутам внутренних структур данных 3D-сцены осуществляется с помощью Python-модулей, среди которых наиболее часто употребимыми являются **bpy.data**, **bpy.ops**, **bpy.props**, **bpy.types**, **bpy.context**, **bpy.utils**, **bgl**, **blf**, **mathutils**.

1) Справедливо в отношении актуальной на момент данной публикации версии Blender — 2.79b.

2) При переходе от версии 2.49 к версии 2.5 и более поздним архитектура Blender была радикальным образом пересмотрена, что нашло своё отражение в сильно изменившемся Python API.

3) На момент данной публикации готовится к выходу версия 2.8.

Например, следующая команда, набранная в интерактивной Python-консоли, выводит информацию о размере коллекции объектов:

```
>>> bpy.data.objects                        # консольная команда
<bpy_collection[3], BlendDataObjects> # вывод интерпретатора в консоль
```

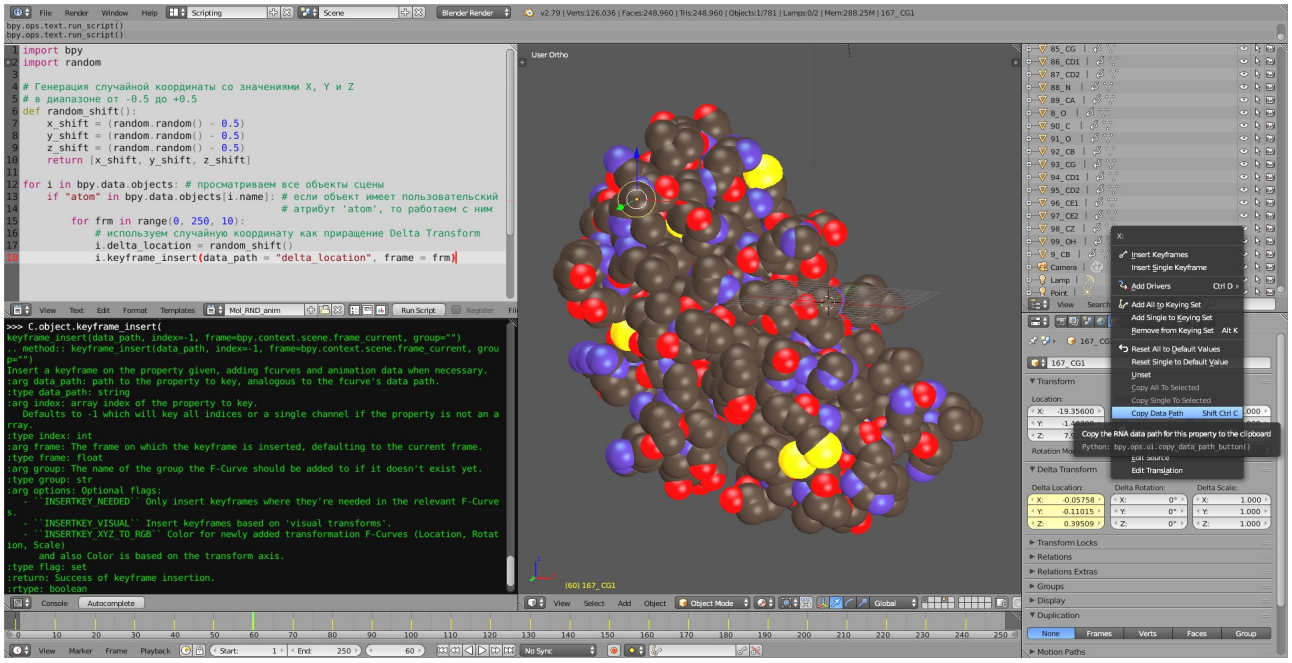


Рис. 1: Графический интерфейс программного пакета 3D-моделирования, анимации и рендеринга — Blender. Слева направо, сверху вниз показаны основные инструментальные панели среды моделирования:

1. Script Editor — текстовый редактор с подсветкой синтаксиса (вверху слева).
2. Видовое окно, отображающее проекцию трёхмерного объекта (в центре).
3. Outliner – иерархический реестр всех объектов текущей сцены (вверху справа).
4. Интерактивная Python-консоль (внизу слева).
5. Панель свойств объектов (внизу справа) и контекстное меню, дающее доступ к константам и ссылкам на справочные материалы Blender Python API.
6. Временная шкала (внизу)

Для **bpy.data** в Python-консоли зарезервирован псевдоним — переменная **D**, поэтому получить список имён существующих объектов можно следующим образом:

```
>>> list(D.objects)
[bpy.data.objects['Camera'], bpy.data.objects['Cube'],
bpy.data.objects['Point']]
```

Здесь «*Camera*» представляет объект сцены — виртуальную камеру, «*Cube*» — геометрический объект — куб, а «*Point*» — точечный источник света.

Используя имя объекта, можно получить доступ к таким его атрибутам, как положение в пространстве — **location.**

Команда, выводящая в консоль значение координат объекта «*Cube*», может быть такой:

```
>>> bpy.data.objects['Cube'].location
Vector((-16.3, 0.0, 0.0))
```

Как несложно догадаться, изменить положение объекта можно, выполнив прямую запись новых координат объекта «*Cube*» в поле **location**:

```
>>> bpy.data.objects['Cube'].location = -5, 1, 1.7
```

После ввода вышеприведённой команды позиция объекта «*Cube*» немедленно изменится, что будет отражено в видовом окне как перемещение данного объекта в точку пространства с заданными координатами. Но более точно подтвердить новые значения координат лучше с помощью ввода следующей команды в интерактивной Python-консоли Blender:

```
>>> bpy.data.objects['Cube'].location
Vector((-5.0, 1.0, 1.7000000476837158))
```

Низкоуровневое построение 3D-объектов посредством Blender Python API

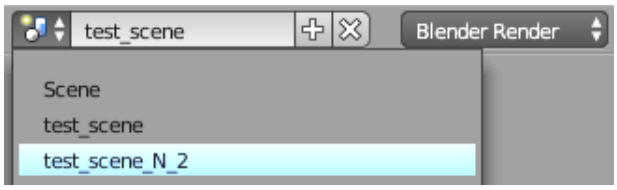
Поскольку объекты в Blender представляют собой достаточно сложные структуры данных, то само ядро программы Blender наделено исключительными правами по управлению ими. Таким образом, создание и удаление объектов возможно лишь с помощью специальных методов. Например, метода **.new()**, имеющегося для каждого типа объектов.

Новую сцену можно создать, вызвав следующую команду:

```
>>> bpy.data.scenes.new(name='test_scene_N_2')
bpy.data.scenes['test_scene_N_2']
```

Список сцен, включая только созданную, может быть выведен в Python-консоль:

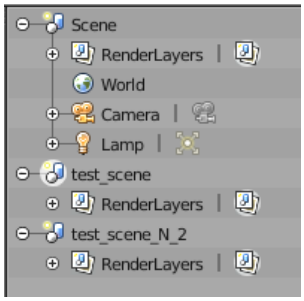
```
>>> list(bpy.data.scenes)
[bpy.data.scenes['Scene'], bpy.data.scenes['test_scene'],
bpy.data.scenes['test_scene_N_2']]
```



Ниспадающий список сцен

Каждая созданная структура данных, описывающая сцену, отображается в ниспадающем перечне сцен и в иерархическом реестре объектов Blender — Outliner.

Удаление объектов осуществляется с помощью соответствующего типу удаляемого объекта метода **.remove()**. Однако для уничтожения структур данных, описывающих объект, недостаточно указать его имя и придётся использовать сам указатель на объект:



Outliner

```
>>>
bpy.data.scenes.remove(bpy.data.scenes['test_scene'])
```

Подтвердить успешность операции можно, выведя в консоль список существующих сцен, в котором сцена с именем '*test_scene*' не значится.

```
>>> list(bpy.data.scenes)
[bpy.data.scenes['Scene'], bpy.data.scenes['test_scene_N_2']]
```

Наиболее естественным представлением моделируемых в среде Blender объектов являются геометрические фигуры, чаще всего представляющие из себя полигональные решётки. Для создания структуры данных полигональной сетки типа meshes также используется метод **.new()**:

```
>>> bpy.data.meshes.new(name='NEW_MESH_OBJECT')
bpy.data.meshes['NEW_MESH_OBJECT']
```

После выполнения вышеприведённой команды в списке mesh-объектов сцены можно обнаружить созданную полигональную сетку с указанным при её создании именем «*NEW_MESH_OBJECT*»:

```
>>> list(bpy.data.meshes)
[bpy.data.meshes['NEW_MESH_OBJECT']]
```

Однако в окне Outliner и тем более в видовом окне никаких новых геометрических объектов после успешного создания полигональной сетки при этом не возникает. Для того, чтобы создать реальный геометрический объект, нужно создать структуру данных типа objects, используя соответствующий метод **.new()**, передав ему в качестве параметров имя создаваемого объекта и ссылку на ранее созданную структуру данных типа mesh, а затем привязать этот объект к сцене:

```
# Получение ссылки на созданную ранее структуру данных типа meshes
>>> mesh = bpy.data.meshes['NEW_MESH_OBJECT']
# Создание объекта NEW_MESH_OBJECT, с использованием ссылки на
# структуру данных типа meshes
>>> mesh_obj = bpy.data.objects.new('NEW_MESH_OBJECT', mesh)
# Связывание объекта 'NEW_MESH_OBJECT' со сценой 'Scene'
>>> bpy.data.scenes['Scene'].objects.link(mesh_obj)
bpy.data.scenes['Scene']...ObjectBase
```

Сразу после связывания объекта '*NEW_MESH_OBJECT*' со сценой, его видимые атрибуты появляются в видовом окне в форме осей, обозначающих геометрический центр объекта, и соответствующей записи в иерархическом реестре — Outliner (рис. 2).

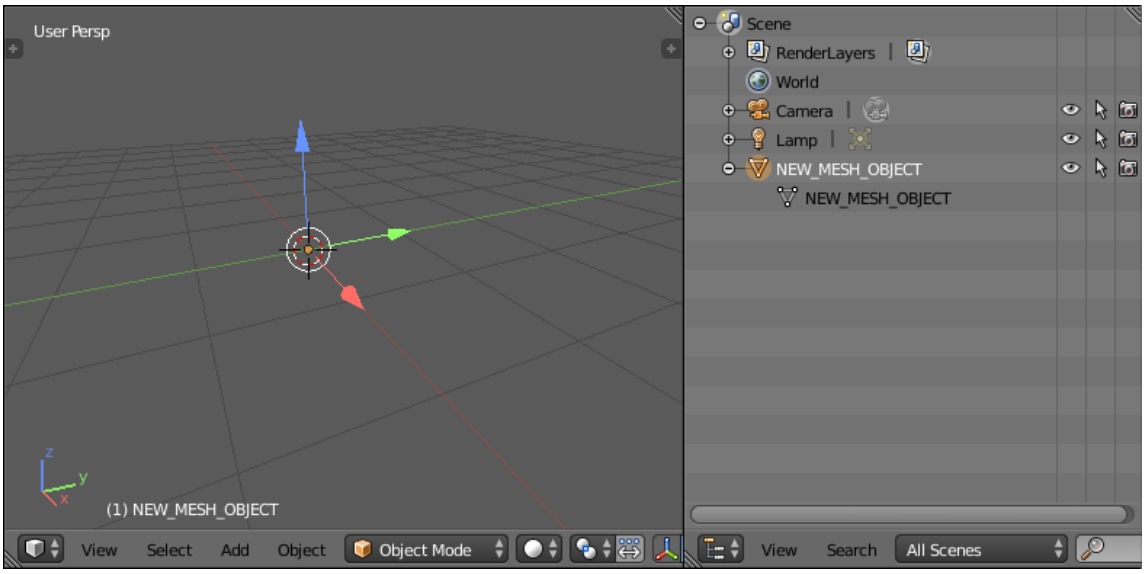


Рис. 2. Отображение созданного объекта в видовом окне и в окне иерархического представления сцены — Outliner

Несмотря на то, что созданный объект '*NEW_MESH_OBJECT*' пока не имеет реального геометрического наполнения, его положением в пространстве можно управлять путём записи координат в поле **.location**:

```
>>> bpy.data.objects['NEW_MESH_OBJECT'].location = 1.0, 1.0, 0.0
```

Для того чтобы определить геометрию полигонального объекта типа meshes, нужно подготовить два списка — вершин и граней (или рёбер).

Каждая вершина задаётся как трёхмерный вектор — список или кортеж из трёх чисел с плавающей запятой, соответствующих координатам X, Y и Z в трёхмерном пространстве. Массив вершин также должен быть организован в

виде списка или кортежа, но уже трёхкомпонентных векторов. Например, вот такого:

```
((0, 0, 1.86), (1, 0, 1.89), (2, 0, 1.22), (3, 0, 2.07), (4, 0, 2.3), (5, 0, 2.2), (0, 1, 3.22))
```

Аналогичным образом должен быть организован и список индексов вершин, образующих грани полигональной сетки. Только список/кортеж, определяющий грань, должен содержать от 3 и более индексов в списке вершин.

Например, вот так выглядит список[4] индексов вершин четырёхугольных граней для решётки, составленной из 6x6 вершин или 5x5 граней:

```
((0, 1, 7, 6), (1, 2, 8, 7) ..... (27, 28, 34, 33), (28, 29, 35, 34))
```

Ключевой операцией в создании геометрического объекта на основе пользовательских геометрических данных является использование метода **mesh.from_pydata()**, которому в качестве параметров передаются ранее подготовленные списки вершин, рёбер (в виде пары индексов) и граней, описанных как кортеж трёх и более индексов.

В листинге, доступном по ссылке, приведён пример программы (Python-скрипта), создающего прямоугольную регулярную полигональную решётку размером 81x81 вершин, Z-координаты которых рассчитаны по формуле **sin(√(x² + y²))**.

Результат работы этого программного кода, отображаемый в видовом окне Blender, представлен на рис. 3.

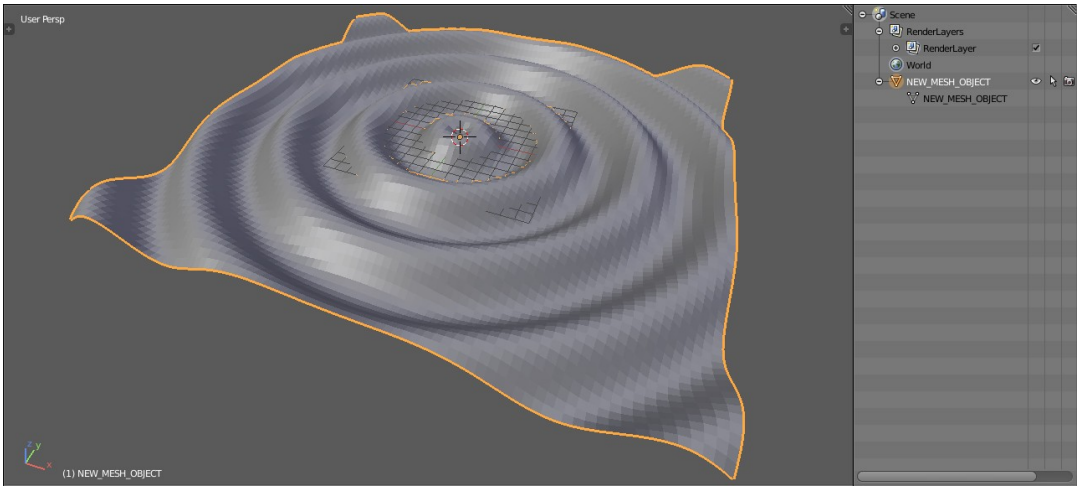


Рис. 3. Созданный с помощью Python-скрипта полигональный mesh-объект '*NEW_MESH_OBJECT*' и его представление в иерархическом реестре объектов — Outliner (справа)

4) Показаны только начальные и конечные элементы списка.

Вышеприведённый пример построения геометрического объекта может быть полезен при моделировании объектов, форма которых рассчитывается Python-программой непосредственно в среде Blender, во внешней программе или описана с помощью данных, полученных экспериментально. В Python-скрипт эти данные могут быть загружены обычными средствами языка Python из текстового файла, содержащего как минимум список координат вершин моделируемого объекта.

В качестве примера таких данных мы использовали трёхмерные координаты участка дна Северного Ледовитого океана [10, 11]. Небольшой фрагмент файла с данными показан на рис. 4.

```
1380000.00  1440000.00  -15.235161
1395000.00  1440000.00  -15.637782
1410000.00  1440000.00  -15.235161
1425000.00  1440000.00  -12.618121
1440000.00  1440000.00  -9.799770
```

Рис. 4. Фрагмент текстового файла *IBCAO_small_crop.map*, содержащий в каждой строке X, Y и Z координаты регулярной полигональной решётки, описывающей рельеф фрагмента дна Северного Ледовитого океана. Размерность описываемой файлом решётки — 97x97 вершин. Шаг решётки — 15 км

Считывание, интерпретацию содержимого текстового координатного файла (*IBCAO_small_crop.map*), его первичную обработку и формирование списка координат вершин полигональной сетки выполняет нижеследующий фрагмент Pytnon-скрипта:

```
import bpy    # Импорт основных модулей Blender API
import os
VERTEXES = () # список кортежей координат вершин

# --- Получение списка координат вершин из текстового файла ---
# Составление пути к текстовому файлу с заданным именем,
# находящемуся в одном каталоге с blend-файлом сцены
data_file_name = os.path.join(os.path.dirname(bpy.data.filepath),
                              "IBCAO_small_crop.map")
data_file = open(data_file_name, 'r') # открываем файл

# Построчное чтение координатного файла в цикле
for line in data_file:
    vertex = line.split() # выделение трёх строк, в каждой из которых
                          # записана X, Y или Z координаты
```

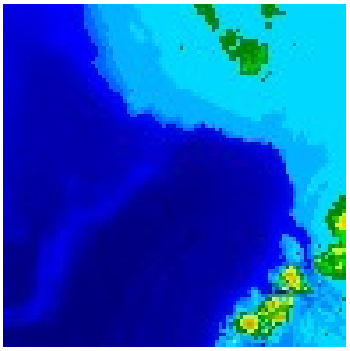
```
# преобразование строк в число и масштабирование координат
    vertex[0] = float(vertex[0]) / 2000  # X
    vertex[1] = float(vertex[1]) / 2000  # Y
    vertex[2] = float(vertex[2]) / 50    # Z
    VERTEXES += vertex, # наращивание кортежа из списков координат вершин
                        # Завершающая строку запятая важна!
data_file.close()       # закрываем файл
```

Первичная обработка заключается в преобразовании текстовых полей в числовой формат и масштабировании координат, чтобы исключить дополнительные шаги по настройке сцены, необходимые для корректного отображения больших геометрических объектов в видовом окне Blender.

При этом координата Z, определяющая высоту каждой точки поверхности, увеличена в 40 раз относительно координат X и Y, для того чтобы усилить визуальное восприятие рельефа.

Остальные фрагменты Python-скрипта, выполняющие генерацию списка четырёхугольных граней и создание mesh-объекта почти не отличаются от таковых у программы, строящей трёхмерный график функции $\mathbf{z = \sin(\sqrt{(x^2 + y^2)})}$.

Результат работы итогового скрипта, строящего рельеф фрагмента дна Северного Ледовитого океана представлен на рис. 5.



Фрагмент физической карты, размерностью 97 x 97 пикселей, соответствующий моделируемому участку поверхности дна Северного Ледовитого океана

Поскольку Blender является в первую очередь трёхмерным редактором, то, находясь в его среде, можно выполнить проецирование фрагмента физической карты на поверхность моделируемого рельефа дна с помощью текстурирования в штатном редакторе Blender - UV/Image Editor, как это показано на рис. 6 и получить таким образом принципиально иной уровень восприятия совмещённых данных.

С использованием вышеописанных методов моделирования в среде Blender была построена цифровая модель рельефа (ЦМР) дна Северного Ледовитого океана [15] низкого разрешения, выделенная из International Bathymetric Chart of the Arctic Ocean (IBCAO) Version 3.0 (модель IBCAO_V3_500m_RR) [16], описывающая территорию с размерами около 5800 км × 5800 км, в пределах которой расположен бассейн Северного Ледовитого океана в обрамлении северных областей Евразии и Северной Америки. ЦМР построена на квадратной сетке с шагом 10 км, представляет собой матрицу 581 × 581 и включает 337561 точку [17, 18].

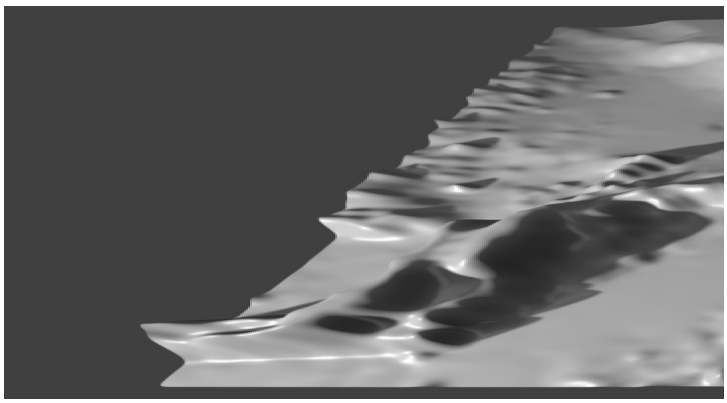


Рис. 5. Трёхмерная поверхность, моделирующая рельеф фрагмента дна Северного Ледовитого океана, построенная в среде Blender 3D с помощью Python-скрипта, строящего полигональную решётку по координатам, считываемым из текстового файла (рис. 4).
Полигональная решётка, имеющая размерность 96x96 граней, была сглажена с помощью алгоритма подразделяемых поверхностей Catmull-Clark [12] с количеством разбиений 2x и с помощью модели освещения Фонга, базирующейся на интерполяции нормалей вершин [13, 14]

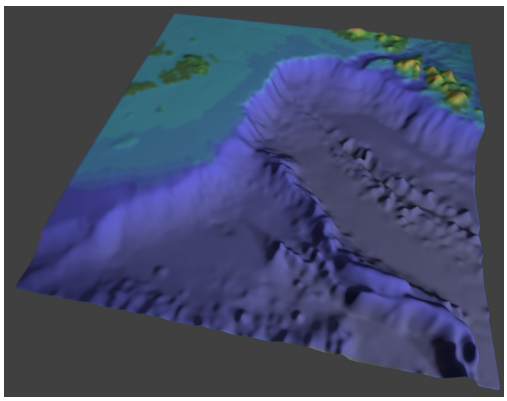


Рис. 6. Трёхмерная поверхность, моделирующая рельеф фрагмента дна Северного Ледовитого океана, текстурированная фрагментом физической карты, соответствующим рельефу. Коэффициент влияния текстуры на диффузный канал материала равен 0.8

При моделировании трёхмерных поверхностей, представляемых в виде регулярной полигональной решётки, имеет смысл рассмотреть случай, когда исходные данные представляются в виде обычных двумерных изображений, у которых яркость каждого пикселя кодирует значение Z-координаты вершины, представленной пикселем изображения. Такой вариант сохранения экспериментальных данных используется очень часто.
Хотя в стандартном языке Python и нет встроенных средств для работы с изображениями, в среде Blender, благодаря имеющимся инструментам, отсутствует необходимость использования сторонних библиотек для манипуляций с графическими файлами. Будучи графическим редактором, Blender 3D содержит широкий набор встроенных средств для обработки изображений, доступ к большинству из которых возможен посредством его API.
Загрузить графический файл в среду Blender можно при помощи окна UV/Image Editor (рис. 7), уже используемого выше для текстурирования фрагмента рельефа дна.

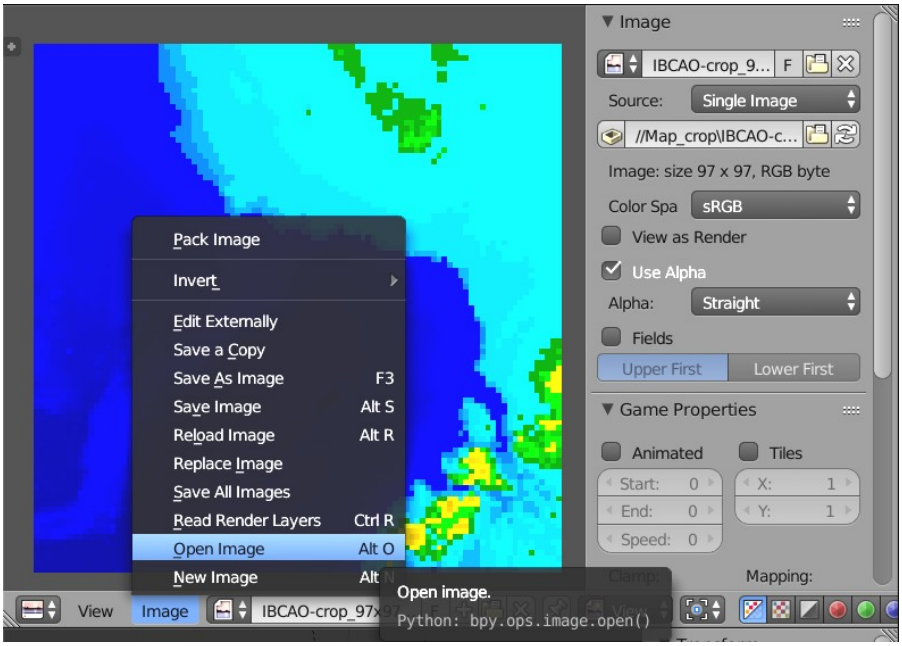


Рис. 7. Окно редактора UV/Image Editor с загруженной в него текстурой, представляющей собой фрагмент физической карты моделируемого участка дна Северного Ледовитого океана. Обратите внимание на всплывающую подсказку, отображающую функцию Blender API, позволяющую выполнить загрузку графического файла с помощью Python-скрипта

Структуры данных, описывающие загруженные в Blender изображения, хранятся в **bpy.data.images**.
Вывести список загруженных изображений можно с помощью уже известной команды **list()**. Например, так:

```
>>> list(bpy.data.images)
[bpy.data.images['IBCAO-crop.tif'], bpy.data.images['IBCAO-
crop_97x97.png']]
```

Несложно догадаться, что '*IBCAO-crop.tif*' и '*IBCAO-crop_97x97.png*' представляют собой имена графических файлов, загруженных в Blender посредством UV/Image Editor.
Как и для любого другого из объектов, с которыми работает Blender, данная среда предлагает набор базовых методов и для манипуляций с объектами-изображениями. Вывести список методов и полей данных, доступных для работы с загруженными в Blender графическими файлами, можно с помощью команды **dir()**, передав ей качестве аргумента ссылку на объект:

```
dir(bpy.data.images['IBCAO-crop_97x97.png'])
```

Из обширного списка имеет смысл рассмотреть два наиболее часто употребляемых поля рассматриваемой структуры данных — **size** и **pixels**.

Поле **size** представляет собой список, содержащий в первом своём элементе (индекс 0) ширину изображения файла в пикселях, а во втором элементе (индекс 1) — его высоту. При вводе команды в Python-консоли это выглядит так:

```
>>> bpy.data.images['IBCAO-crop.tif'].size[0]
376
>>> bpy.data.images['IBCAO-crop.tif'].size[1]
670
```

где 367 — ширина изображения, загруженного из файла '*IBCAO-crop.tif*' [16], а 670 — его высота в пикселях.

Для удобства манипуляций с данными, содержащимися в загруженном графическом файле, можно получить ссылку на объект-изображение, указав его имя:

```
img = bpy.data.images['IBCAO-crop_97x97.png']
```

Доступ к содержимому загруженного в Blender графического файла (пикселям) предоставляет одноимённое поле pixels. Его содержимое можно вывести на экран следующей командой:

```
>>> bpy.data.images['test_img.png'].pixels[:]
(0.0, 0.0, 1.0, 1.0,
 0.5, 0.5, 0.5, 1.0,
 1.0, 0.0, 0.0, 1.0,
 0.0, 1.0, 0.0, 1.0,
 1.0, 1.0, 1.0, 1.0,
 0.0, 0.0, 0.0, 1.0)[5]
```

Из рисунка 8 видно, что цвет каждого пикселя описывается четырьмя нормированными значениями с плавающей запятой, где первое число определяет интенсивность красной компоненты (**Red**), второе — зелёной (**Green**), третье — синей (**Blue**), а четвёртое определяет прозрачность пикселя (интенсивность в **Alpha**-канале).

Кроме того, сопоставив графическое и числовое представление тестового изображения, можно сделать вывод и о порядке записи пикселей — слева направо, снизу вверх.

Стоит отметить, что такой формат хранения пиксельной информации справедлив для практически всех загружаемых в Blender растровых графических файлов.

5) Консольный вывод был переформатирован для удобства восприятия групп из 4-х чисел, каждая из которых описывает один пиксель загруженного в Blender изображения.

Как и в случае с координатами геометрических объектов, пользователю доступно прямое изменение значений цветовых компонент списка пикселей (рис. 9), чем можно воспользоваться для алгоритмической попиксельной обработки изображений средствами Python, выгрузив результат в том же UV/Image Editor как графический файл (меню: Image→Save As Image или горячей клавишей данного редактора — F3).

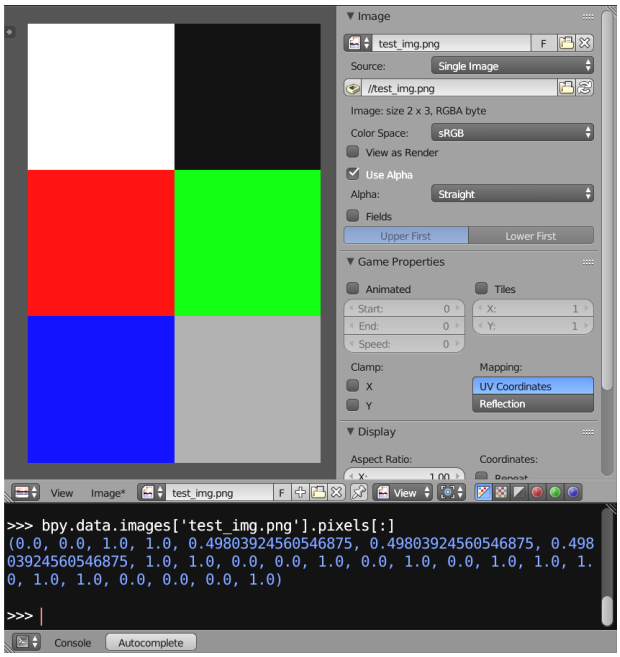


Рис. 8. Отображение тестового графического файла с размерностью 2x3 пикселя в окне UV/Image Editor и в Python-консоли в виде списка нормированных числовых значений, описывающих цвет пикселей в формате RGBA

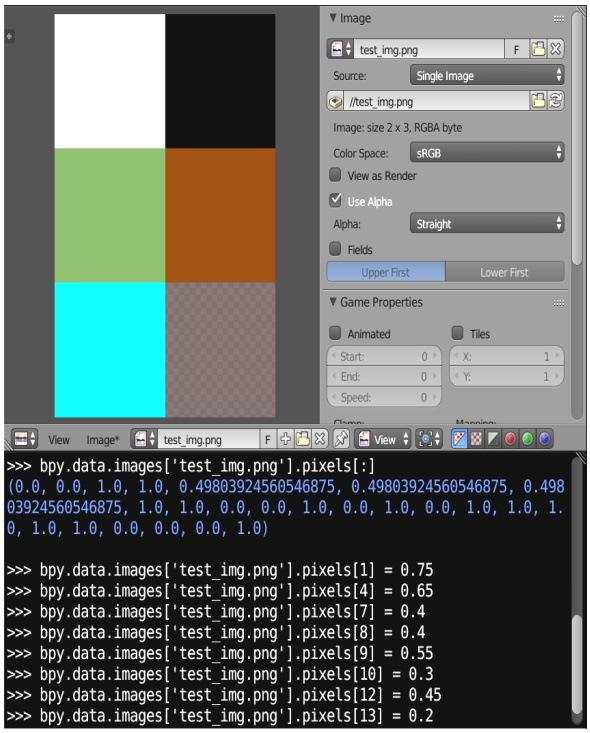


Рис. 9. Прямая запись новых значений цветовых компонент пикселей и отображение её результата в окне UV/Image Editor

При использовании вышеописанных методов работы с загруженными в Blender изображениями был написан скрипт[6], выполняющий достаточно типичную задачу — построение рельефа поверхности, где X и Y координаты представляют собой регулярную решётку и потому не определяются внешними по отношению

6) При написании данного программного кода преследовалась цель обеспечить его лучшую читаемость, пусть и в ущерб оптимизации по объёму и скорости исполнения. Кроме того, из тех же соображений из данного примера практически полностью исключены все средства обработки ошибок.

к программе данными, а Z-координата определяется яркостью соответствующих пикселей в графическом файле *IBCAO-crop_97x97.png* [16].
Результат работы программы, выполняющей построение рельефной поверхности по карте высот, сохранённой в виде растрового графического файла, представлен на рисунках 10 и 11.

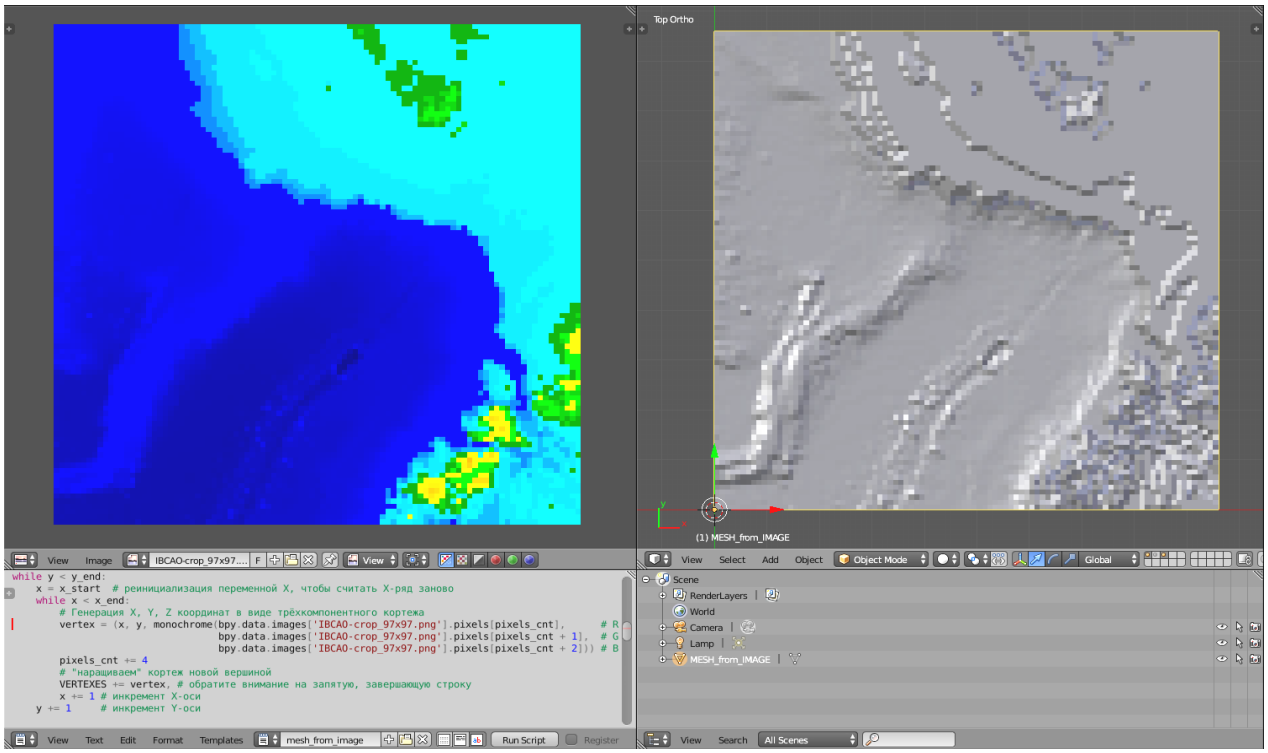


Рис. 10. Пример построения рельефной поверхности (справа) по карте высот, представленной в виде растрового графического файла (слева)

Приведённый пример нельзя считать картографически корректным, однако он наглядно показывает, как можно загружать графические файлы в среду программной платформы Blender и использовать содержащуюся в них информацию для построения моделей. При этом Blender Python API позволяет и создавать графические файлы, и сохранять их, что открывает для рассматриваемой программной платформы широкие возможности по Python-обработке изображений, а также для вовлечения в рабочий процесс широкого класса внешних специализированных программ, рассчитанных на получение информации в форме растровых графических файлов.

Создать структуру данных, отвечающих за хранение графического файла, можно с помощью метода **.new**, которому в качестве параметров необходимо передать как минимум три параметра: текстовое имя, ширину и высоту изображения в пикселях. Например, так:

```
>>> bpy.data.images.new("test_img", width = 128, height = 64)
bpy.data.images['test_img']
```

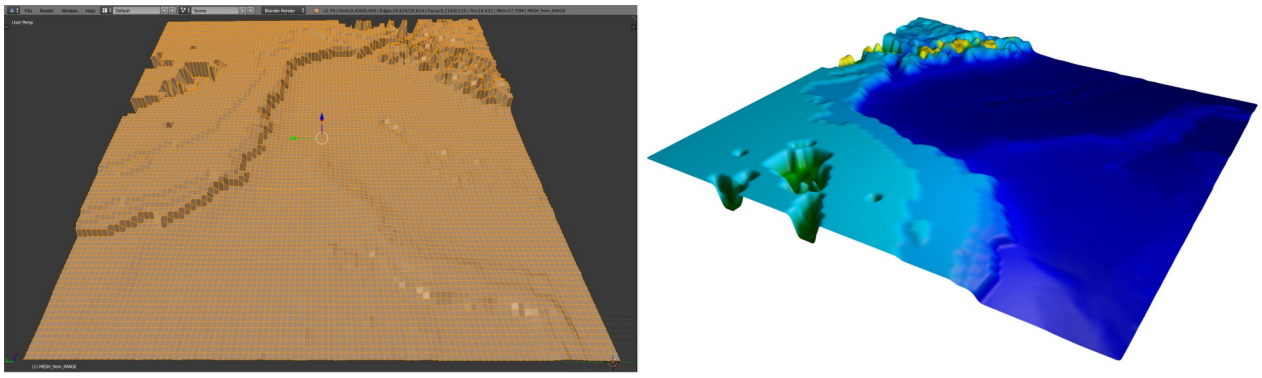


Рис. 11. Трёхмерное представление рельефной поверхности, построенной с использованием графического файла *IBCAO-crop_97x97.png* [16] в качестве карты высот. В правой части показан результат рендеринга 3D-поверхности с наложенной на неё исходной картой высот в виде текстуры

Сразу после создания описывающей графический файл структуры данных, можно работать с ней, записывая в предварительно инициализированное ядром Blender поле **pixels** данные[7] об интенсивности каждого пикселя, что ранее уже было показано (рис. 9).

Для сохранения изображения, созданного в среде Blender, необходимо в поле **.file_format** указать тип графического файла, например, "PNG", "JPEG" или любой другой из числа поддерживаемых Blender [2]:

```
>>> bpy.data.images['test_img'].file_format = "PNG"
```

А также указать путь к сохраняемому файлу в файловой системе:

```
>>> bpy.data.images['test_img'].filepath_raw = "d:\\saved_name.png"
```

После выполнения вышеперечисленных операций сохранение графического файла осуществляется вызовом метода **.save()** без передачи параметров:

```
>>> bpy.data.images['test_img'].save()
```

Стоит отметить, что созданные в среде Blender изображения сохраняются внутри blend-файла (так же, как и другие объекты) независимо от того, были ли они сохранены в виде отдельных графических файлов или нет. Для того, чтобы изображение не включалось в blend-файл, у него не должно быть "пользова-

7) Напомним, порядок расположения субпикселей в массиве (списке) — RGBA, а значение, описывающее интенсивность каждой из цветовых компонент, находится в диапазоне от 0 до 1.

телей". То есть изображение не должно использоваться другими объектами. Обнулить счётчик числа пользователей можно с помощью метода **.user_clear()**:

```
>>> bpy.data.images['test_img'].user_clear()
```

Python-контроль операторов Blender

В начале статьи было сказано, что платформа Blender может быть интересна научному сообществу в качестве среды для моделирования благодаря очень обширному и подчас уникальному набору инструментов, применимость и гибкость которого многократно усиливается при контроле посредством Blender Python API.

Доступ к встроенным инструментам Blender, которые в терминах данной программной среды называются ***операторами***, может быть осуществлён с помощью объекта **bpy.ops**.

Список операторов, доступных на заданном иерархическом уровне, можно вывести в Python-консоль с помощью уже упоминавшейся функции **dir()**. Однако иерархическая структура инструментов Blender столь обширна и сложна, что в сам пакет была интегрирована система помощи в виде всплывающих подсказок — Python Tooltips (рис. 12).

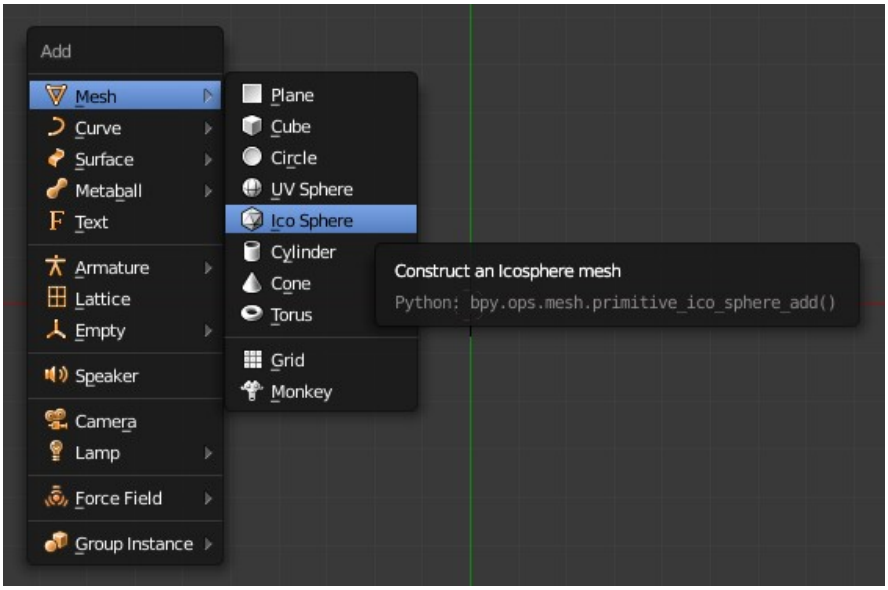


Рис. 12. Всплывающая подсказка Python Tooltips, демонстрирующая синтаксис Python-команды (вторая строка, отображаемая серым цветом), создающей геометрический объект *IcoSphere*

Получив ссылку на нужный оператор, можно с помощью функции автодополнения (**Ctrl+Пробел**) в Python-консоли получить справку и по доступным для данного оператора параметрам и их значениям по умолчанию, как это было

сделано для геометрического объекта — сферы (*Ico Sphere*), строящейся путём подразделения граней икосаэдра:

```
bpy.ops.mesh.primitive_ico_sphere_add(subdivisions = 2,
                                      size = 1,
                                      calc_uvs = False,
                                      view_align = False,
                                      enter_editmode = False,
                                      location = (0, 0, 0),
                                      rotation = (0, 0, 0),
layers = (False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False,
False))
```

Формат и синтаксис Blender-команд, создающих другие геометрические примитивы, аналогичен представленному.

Как видим, для большинства базовых геометрических форм наиболее часто используемыми можно назвать три параметра: *subdivisions*, *size* и *location*, которые в нашем случае позволяют определить при создании генерируемой из икосаэдра сферы количество подразделений её базовой геометрии (от 1 до 10), её размер (радиус) и положение в пространстве.

Конкретный список параметров, актуальных для объектов каждого типа, и диапазоны доступных для них значений можно выяснить в онлайн-справочнике, ссылку на соответствующий раздел которого также предоставляет встроенная в Blender система помощи путём всплывающих подсказок (Python Tooltips, рис. 12). Для этого нужно выбрать в контекстном меню, вызываемом по нажатию правой клавиши мыши, пункт **Online Python Reference**.

Значение возможностей по созданию и контролю простых геометрических фигур в среде Blender сложно переоценить. Даже одной вышеприведённой команды по созданию сферы в трёхмерном пространстве уже достаточно для написания простого скрипта, строящего Ван-дер-Ваальсовы молекулярные модели [19, 20].

Рассмотрим пример такого скрипта[8].

На рис. 13 представлена часть Python-программы, выполняющая ряд таких подготовительных операций, как импорт необходимых для работы скрипта модулей, объявление и инициализация переменных.

В строке 1 выполняется подключение основного модуля Blender Python API.

Импорт модуля **OS** (строка 2) не является обязательным и нужен лишь для удобного составления полного пути к PDB-файлу [21] путём извлечения пути к blend-файлу и присоединения к нему имени искомого PDB-файла (*2ins.pdb*)

8) При написании данного программного кода преследовалась цель обеспечить его лучшую читаемость, пусть и в ущерб оптимизации по объёму и скорости исполнения. Кроме того, из тех же соображений из данного примера практически полностью исключены все средства обработки ошибок.

[22]. То есть скрипт полагает, что файл молекулярной модели находится в том же каталоге, что и blend-файл.

```
 1 import bpy
+2 import os
 3
 4 # Переменные, используемые в качестве индексов кортежей
 5 ATOM_NUM  = 0
 6 ATOM_NAME = 1
 7 ATOM_XYZ  = 2
 8 # Переменные для хранения текущих значений параметров
 9 atm_num = 0
10 atm_name = ""
11 atm_x = 0.0
12 atm_y = 0.0
13 atm_z = 0.0
14
15 sphr_sdiv = 3 # гладкость сферы (количество подразделений икосаэдра)
16
17 # Словарь - база данных ван-дер-ваальсовых радиусов атомов (Ангстр.)
18 # и материалов в виде кортежа цветовых RGB-компонентов
19 ADB = {'C' :(1.71, (0.016, 0.006, 0.0)),   # тёмно-коричневый
20        'H' :(1.16, (0.25, 0.417, 0.52)),   # светло-голубой
21        'O' :(1.29, (0.7, 0.0, 0.0)),       # ярко-красный
22        'N' :(1.5,  (0.043, 0.017, 0.193)), # сине-фиолетовый
23        'S' :(1.84, (0.778, 0.425, 0.0)),   # жёлтый
24        'CL':(1.9,  (0.54, 0.778, 0.097))}  # жёлто-зелёный
25
26 # Определяем кортежи для единичного атома и молекулы (списка атомов)
27 Atm = () # атом
28 Mol = () # молекула
29
30 # Составление пути к PDB-файлу с заданным именем, находящемуся
31 # в одном каталоге с blend-файлом сцены
32 data_file_name = os.path.join(os.path.dirname(bpy.data.filepath),
33                               "2ins.pdb")
```

Рис. 13. Часть скрипта, строящего простую Ван-дер-Ваальсову молекулярную модель инсулина [22], выполняющая ряд подготовительных операций

Переменные, имена которых начинаются с «*ATOM_*», определяют индексы полей в кортеже, хранящем записи об атомах молекулярной модели. Этот приём позволяет обойтись без использования таких структур данных, как классы, но при этом сохранить читаемость программного кода и снизить вероятность ошибок.

Блок объявления переменных в строках 8 — 15 не нуждается в дополнительных комментариях, кроме тех, которые есть в самой программе.

В строках 17 — 24 объявляется и инициализируется Python-словарь, выполняющий роль простейшей базы данных атомов (химических элементов). Ключом в записях служит строка, обозначающая химический элемент. Сама запись представляет собой кортеж, первый элемент которого (индекс 0) хранит Ван-дер-Ваальсов радиус атома [23 - 25], а второй — кортеж из трёх чисел в формате с плавающей запятой в диапазоне от 0 до 1, которые определяют условный цвет атома посредством указания интенсивности цветовых RGB-компонент [26, 27].

Фрагмент скрипта, выполняющего считывание и первичную интерпретацию PDB-файла, представлен на рис. 14.

```
data_file = open(data_file_name, 'r') # открываем файл

# Построчное чтение и интерпретация координатного файла в цикле
for line in data_file:
    line = line.upper() # преобразуем все символы к верхнему регистру
                        # для упрощения интерпретации
    # Рассматриваем только строки, начинающиеся с ключевых слов
    # 'ATOM', описывающие атомы полипептидных цепей
    if "ATOM" in line[:6]:
        # Выделяем из PDB-строки значащие подстроки и
        # выполняем преобразование типов
        atm_num = int(line[ 6:11])
        atm_name =    line[12:16]  # [13] - тип атома (хим.эл.)
        atm_x = float(line[30:38]) # X
        atm_y = float(line[38:46]) # Y
        atm_z = float(line[46:54]) # Z
        # составляем кортеж - запись инф.об атоме
        Atm = (atm_num, atm_name, (atm_x, atm_y, atm_z))
        # составляем кортеж атомов, описывающий молекулу
        Mol += Atm, #
# закрываем файл
data_file.close()
```

Рис. 14. Фрагмент скрипта, выполняющего считывание и первичную интерпретацию PDB-файла. Позиции значащих полей в пределах строки PDB-файла взяты согласно спецификации этого формата [21]

В цикле рассматриваются лишь строки, начинающиеся с ключевого слова *ATOM*. Все другие (в том числе и *HETATM*) отбрасываются. Однако в реальных задачах моделирования объектов и процессов молекулярной биологии [28 - 30] на этапе считывания и интерпретации исходных данных имеет смысл выполнять иерархическую сегментацию макромолекулярной модели на группы гетерогенных атомов, субъединицы, цепочки и другие более мелкие структурные единицы, такие как нуклеотиды и аминокислоты.

Цикл, создающий материалы для атомов каждого типа, перечисленных в словаре, приведён на рис. 15.

```
# Создаём материалы для каждой пары в словаре
for mtr_name in ADB.keys():
    bpy.data.materials.new(mtr_name) # создаём материал с именем ключа
    # назначаем созданному материалу диффузный цвет из базы данных
    # (второй элемент кортежа)
    bpy.data.materials[mtr_name].diffuse_color = ADB[mtr_name][1]
```

Рис. 15. Фрагмент кода, создающий материалы для каждого из элементов в базе данных атомов

В качестве имён материалов используются ключи, извлечённые из словаря *ADB* (строка 59). Если материал с таким именем уже существует, то Blender создаст новый материал с именем ключа, к которому будет добавлен числовой суффикс. Однако диффузный цвет из базы данных (строка 63) будет назначен материалу, имя которого в точности соответствует ключу из словаря *ADB*, то есть уже существующему. И в дальнейшем будет использоваться материал с этим именем, поскольку все обращения к объектам в данном скрипте ведутся по ключу — имени химического элемента.

Вся работа по построению геометрии Ван-дер-Ваальсовой молекулярной модели [19, 20] ведётся в единственном цикле (рис. 16, строка 66), последовательно перебирающем элементы кортежа *Mol*, то есть считанные из PDB-файла данные об атомах.

```
65 # Создаём сферы атомов молекулярной модели
66 for Atm in Mol:
67     # создаём сферу
68     bpy.ops.mesh.primitive_ico_sphere_add(
69         subdivisions = sphr_sdiv,          # количество подразделений
70         size = ADB[Atm[ATOM_NAME][1]][0], # ван-дер-ваальсов радиус
71         location = Atm[ATOM_XYZ])          # положение в пространстве
72 
73     bpy.ops.object.shade_smooth() # определяем для сферы сглаженное отображение граней
74                                   # с использованием интерполяции нормалей их вершин
75     # формируем составное имя атома из его номер и обозначения хим.элемента
76     atm_name = str(Atm[ATOM_NUM]) + "_" + Atm[ATOM_NAME]
77     # назначаем сформированное имя только что созданному объекту - сфере
78     bpy.context.active_object.name = atm_name
79     # назначаем созданному объекту-сфере материал согласно обозначению его хим.элемента
80     bpy.data.objects[atm_name].data.materials.append(bpy.data.materials[Atm[ATOM_NAME][1]])
81     # определяем для созданного атома произвольное пользовательское свойство "atom"...
82     bpy.data.objects[atm_name]["atom"] = Atm[ATOM_NAME][1]
```

Рис. 16. Финальная часть Python-скрипта, создающая молекулярную модель из сфер Ван-дер-Ваальсова радиуса, обозначающих атомы

Сфера, обозначающая атом, создаётся командой в строках 68 — 71. Значение её радиуса берётся из первого элемента (индекс 0), получаемого из базы данных (Python-словаря *ADB*) по ключу *[Atm[ATOM_NAME][1]*, который представляет собой один символ (то есть, второй по счёту, с индексом 1) из второго элемента кортежа, определяемого константой[9] *ATOM_NAME*.

В текущем примере не используется параметр *layers*, отвечающий за размещение созданного объекта на одном или на нескольких из 10 имеющихся в Blender слоёв. Однако в реальной задаче может оказаться очень полезным распределение различных групп атомов по разным слоям, которое позволяет быстро включать и выключать их отображение и выбирать нужную группу для удобной работы с ней [28 - 30].

9) На самом деле, конечно же, переменной, поскольку объявление констант в языке Python не предусмотрено.

Операция (строка 73), включающая сглаженное отображение граней полигонального объекта, не является обязательной (рис. 17). К тому же её можно выполнить и позже.

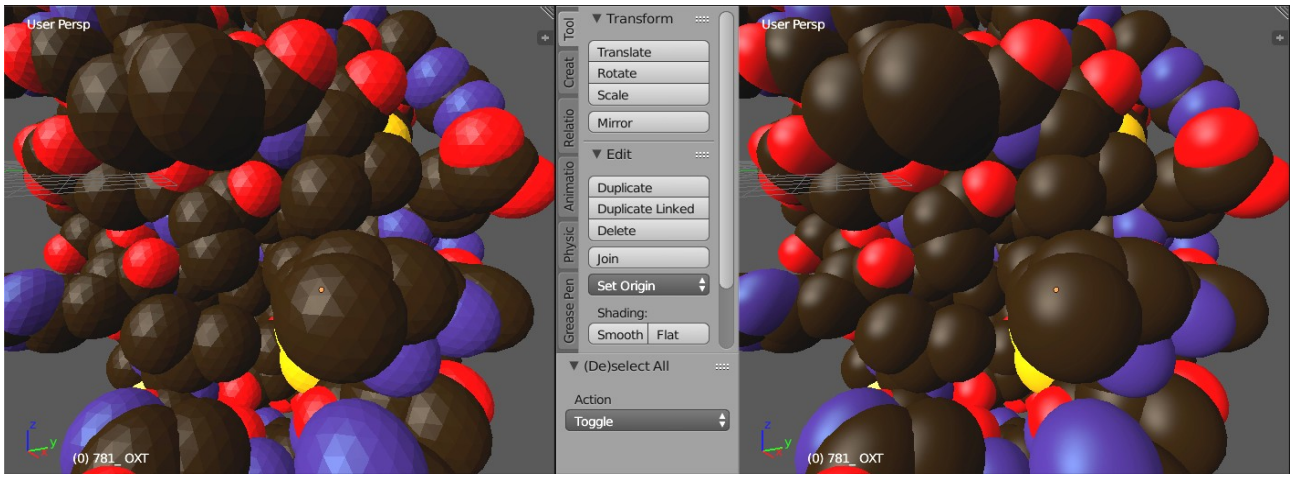


Рис. 17. Различие между Flat (плоским, слева) и сглаженным по Фонгу [13, 14] (Smooth, справа) отображением полигональных объектов

В строке 76 (рис. 16) из номера атома и его имени, полученных из PDB-файла [21], составляется имя объекта-сферы, обозначающей атом. А в строке 78 оно назначается только что созданному объекту с использованием контекста выделения, который как раз и указывает на последний созданный объект-сферу.

В строке 80 созданной сфере назначается материал ("цвет атома") [26, 27]. При этом обращение к объекту-сфере ведётся уже по назначенному ему имени. Ссылка на подготовленный ранее материал получается также по его имени *Atm[ATOM_NAME][1]*, которое представляет собой односимвольную строку, извлекаемую из кортежа *Atm*, полученного по индексу *ATOM_NAME* в цикле.

В последней, 82-й строке Python-скрипта, показанного на рис. 16, выполняется операция по созданию пользовательского (Custom) атрибута с произвольным именем (в нашем примере это "*atom*") и его инициализация односимвольной строкой, взятой из строки — имени текущего атома по индексу 1. Количество таких произвольных атрибутов у любого объекта Blender неограниченно. Размещать в них можно любые данные, относящиеся к стандартным типам: числа целые и с плавающей запятой, строки, списки, а также словари, у которых ключ — строка, а значение — константа одного из стандартных типов.

Пользовательские атрибуты копируются при дублировании родительского объекта, могут изменяться во времени (быть анимированными) и, что важно, сохраняются в blend-файле.

Вышеописанная операция не имеет прямого отношения к построению молекулярной модели. Однако использование произвольных атрибутов может быть

крайне полезно при реализации достаточно сложных расчётных алгоритмов, поскольку позволяет хранить такие данные, как заряды, параметры, характеризующие взаимодействие с растворителем, и т. п. [28 - 30].

Набор пользовательских данных (атрибутов) может быть привязан к любому уровню структур данных с идентификатором (ID). Так, например, в Python-консоли Blender объекту *554_ CG1*, представляющему атом углерода с номером 554 назначается атрибут '*hydro*', инициализированный значением **True**:

```
>>> bpy.data.objects['554_ CG1']['hydro'] = True
```

Проверить значение атрибута можно также с помощью Python-консоли:

```
>>> bpy.data.objects['554_ CG1']['hydro']
1
```

После создания пользовательского атрибута его имя и значение можно видеть на соответствующей вкладке (в данном случае Object) редактора свойств — Properties (рис. 18). Там же его можно и отредактировать во всплывающем контекстном окне.

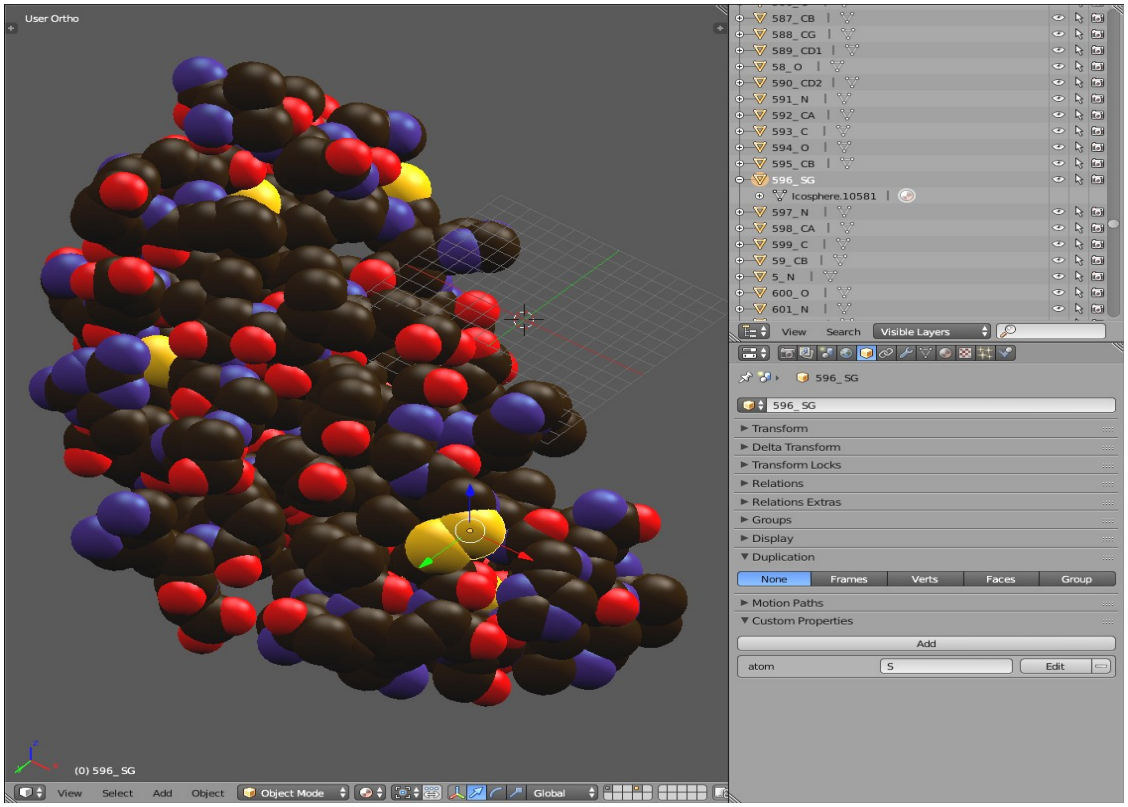
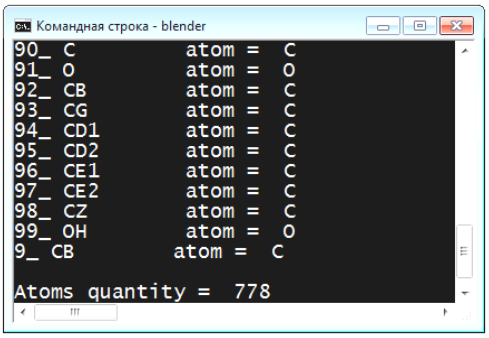


Рис. 18. Визуальное представление пользовательских (Custom) атрибутов атома серы (S) с номером 596 в графическом интерфейсе Blender (нижний правый угол)

Внутри Python-скрипта привязанный к каждому созданному объекту-сфере атрибут «atom» может быть использован не только для хранения сопутствующих объекту данных, но прежде всего как индикатор искомого объекта, отличающий его от всех прочих объектов сцены, таких как источники света, камеры и пр. Так, например, в приведённом на рис. 19 примере среди всех объектов сцены отбираются лишь те, у которых присутствует атрибут «atom» [31].

```
1  import bpy
2
3
4  counter = 0
5
6  for i in bpy.data.objects: # просматриваем все объекты сцены
7      if "atom" in bpy.data.objects[i.name]: # если объект имеет пользовательский
8                                              # атрибут 'atom', то работаем с ним
9          print(i.name, "      atom = ", bpy.data.objects[i.name]['atom'])
10         counter += 1
11
12 print('\nAtoms quantity = ', counter)
```

Рис. 19. Фрагмент Python-скрипта, выбирающий из всех объектов сцены те, у которых установлен атрибут "atom", подсчитывающий их число и выводящий имена объектов-сфер и значение атрибута "atom" в консоль Blender



Вывод отладочной информации в программную консоль Blender

Имена таких объектов вместе со значением пользовательского атрибута выводятся в программную консоль Blender.

Справедливости ради стоит отметить, что результат Blender-визуализации (рис. 20) молекулы инсулина [22], построенной вышеприведённым скриптом (рис. 13-16), нельзя назвать строго оригинальным. Построение молекулярных моделей и ранее можно было выполнять в большом числе 3D-пакетов. Например, ещё для Kinetix 3D Studio Max [32] был написан простой подключаемый модуль (plug-in) PDBImp.

Высочайшего уровня в качестве визуализации молекулярных моделей удалось достичь Jyrki Hokkanen в разработанном им модуле, расширяющем 3D-пакет Realsoft 3D [33], и разработчикам программного дополнения Molecular Maya (mMaya) [34] для Autodesk Maya [35]. При этом первое решение утратило свою актуальность в связи с неподдерживаемым статусом как самого plug-in модуля, так и 3D-пакета, для которого оно было разработано[10]. Второе решение

10) Разработка пакета компанией Realsoft Oy была прекращена приблизительно в 2011 году. Последний же релиз Realsoft 3D v.7 для Windows датируется и вовсе 2009 годом.

(Molecular Maya), несмотря на бесплатный статус базового плагина, является откровенно коммерческим продуктом, предлагающим пользователю покупать дополнительные варианты представления молекулярных моделей. К тому же, сам коммерческий пакет Autodesk Maya обладает крайне низкой доступностью для конечного пользователя из-за его высокой стоимости[11], а также высокой сложности освоения и применения даже опытными пользователями [35].

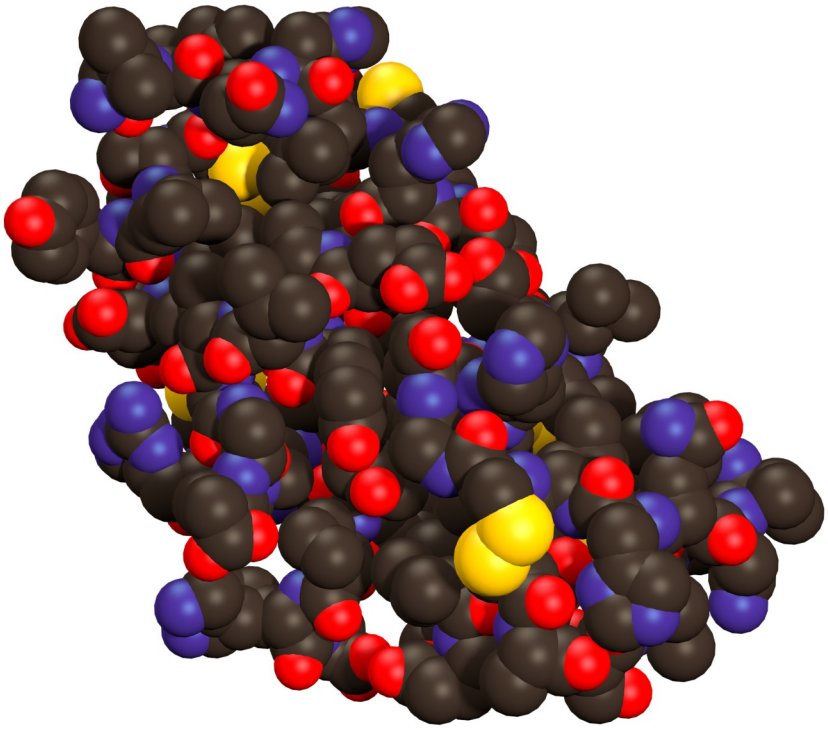


Рис. 20. Визуализированная в Blender Ван-дер-Ваальсова модель молекулы инсулина, построенная Python-скриптом, представленным на рисунках 13 — 16

И, наконец, в дистрибутив Blender входит Python-дополнение (Add-on) для импорта/экспорта PDB-файлов — Atomic Blender PDB [36], работу которого, к сожалению, нельзя назвать удовлетворительной (рис. 21), несмотря на большой объём кода данной Python-программы.

Из приведённого сравнительного рисунка (рис. 21) очевидно, что Atomic Blender PDB не использует Ван-дер-Ваальсовы радиусы для сфер, обозначающих атомы (несмотря на то, что именно такой вариант молекулярной модели был выбран в настройках этого программного дополнения, и не позволяет отсе-

11) На момент данной публикации предлагается на условиях подписки.

кать ненужные атомы кислорода, описываемые в PDB-файле ключевым словом HETATM[12] и относящиеся, судя по всему, к растворителю.

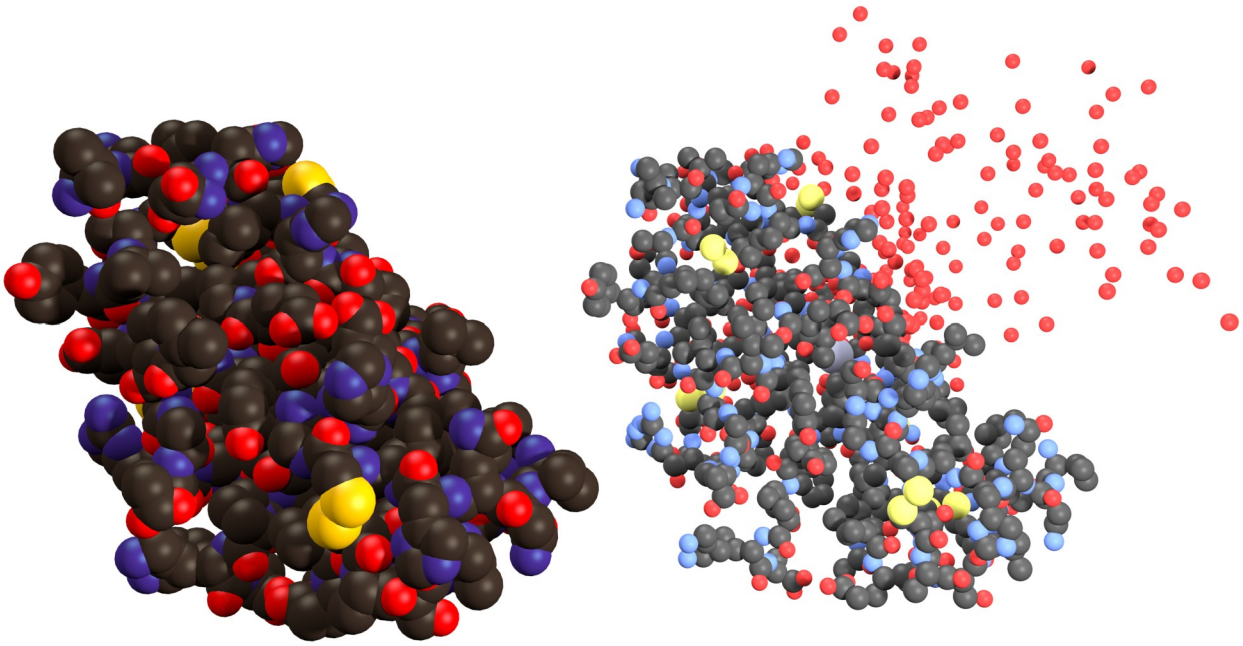


Рис. 21. Визуализированная в Blender Ван-дер-Ваальсова модель молекулы инсулина [22], построенная Python-скриптом, представленным на рисунках 13 — 16 (слева), и построенная штатным дополнением 3D-редактора — Atomic Blender PDB [36]

Более того, структура 3D-модели инсулина (рис. 22), построенной Atomic Blender PDB, такова, что не позволяет безболезненно удалить ненужные группы атомов, ибо они созданы как экземпляры-ссылки на базовую геометрию сферы, расположенную в центре координат (по одному реальному объекту для каждого типа атомов), и составляют, по сути, единый с ней объект.

Вышеописанный метод характеризуется заметно меньшим потреблением оперативной памяти (рис. 23), что нельзя игнорировать, когда речь идёт о моделях биологических макромолекул, число атомов у которых может составлять десятки и даже сотни тысяч, но подходит он лишь для визуализации неизменяемых во времени моделей.

12) Справедливости ради нужно заметить, что «ненужные» строки PDB-файла, начинающиеся с ключевого слова HETATM, можно изъять из него с помощью простого текстового редактора.
Таким же образом можно и сегментировать молекулярную модель, разбив участки, описывающие разные структурные единицы молекулярной модели, на разные PDB-файлы, загружаемые в Blender по отдельности, но собираемые в рамках одной сцены.

Тем не менее снизить потребление памяти можно при помощи ряда оптимизаций. Прежде всего, оптимизацией геометрии, как это было сделано в приводимом нами в качестве примера Python-скрипте на рисунках рис. 13 — 16. Так, геометрия объекта Icosphere с числом подразделений 4x содержит 642 вершины против 994 у сферы типа UV Sphere, которые предлагает использовать Atomic Blender PDB, давая при этом сопоставимое визуальное качество. Мы же использовали число подразделений 3x, при котором на одну сферу затрачивается лишь 162 вершины, которых достаточно для корректного представления геометрии сфер, обозначающих атомы в молекулярных моделях.

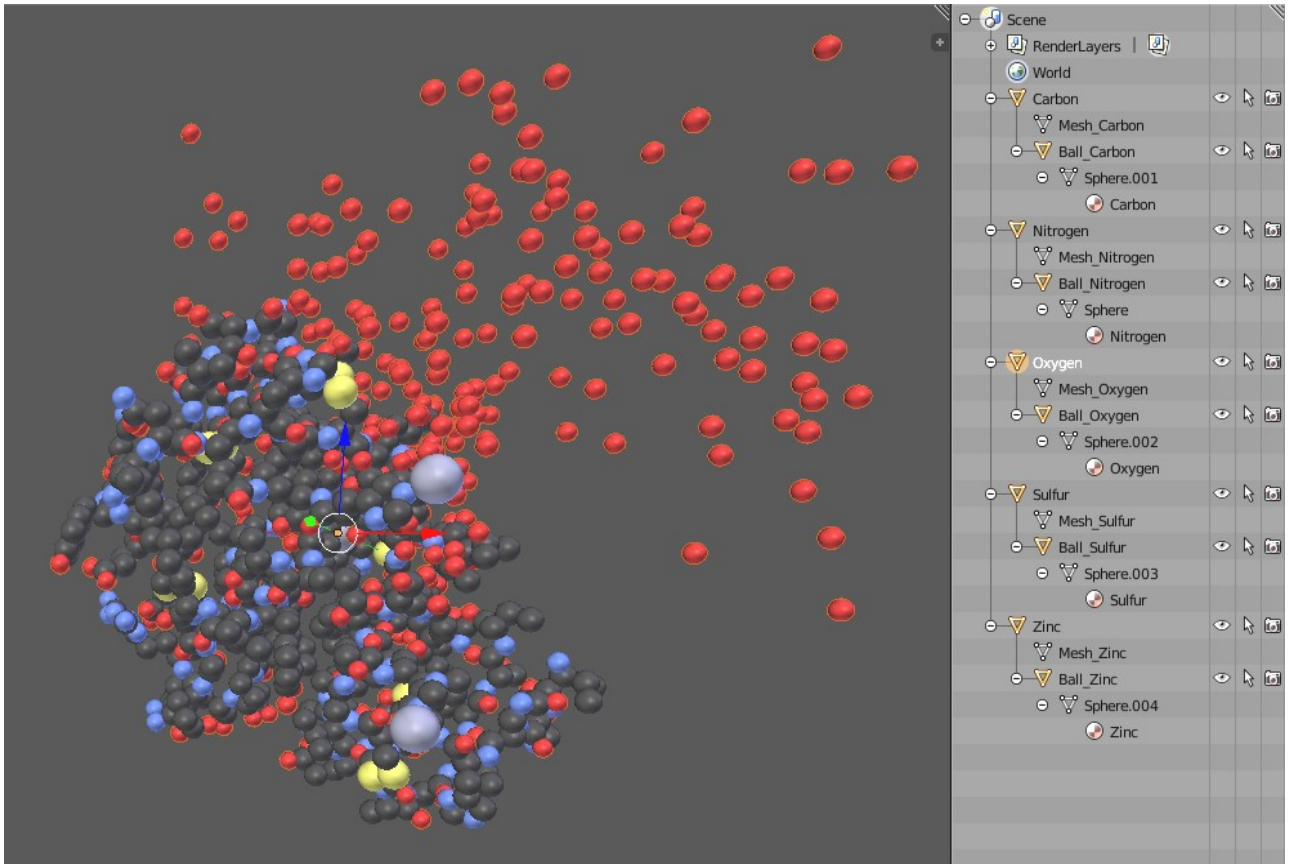


Рис. 22. 3D-модель молекулы инсулина [22], построенная программным модулем Atomic Blender PDB [36].
Слева — представление модели в видовом окне Blender.
Справа — иерархическое представление структуры построенной модели в Outliner

Но самое главное препятствие для рассмотрения всех вышеперечисленных средств визуализации молекулярных моделей в качестве систем моделирования процессов, происходящих на молекулярном уровне, заключается в том, что основной упор в них сделан именно на визуализации, а вовсе не на интерактив-

ной работе с молекулярными моделями и математическом моделировании их динамики. Именно поэтому нами в работах [28, 29] была предпринята попытка создать программы Maya-K-PDB [30] и VOXEL [29], строящие «активные» молекулярные модели в среде 3D-пакетов Autodesk Maya [35] и Maxon Cinema 4D [37] соответственно, главным назначением и отличием которых была возможность динамического моделирования процессов, происходящих на молекулярном уровне.

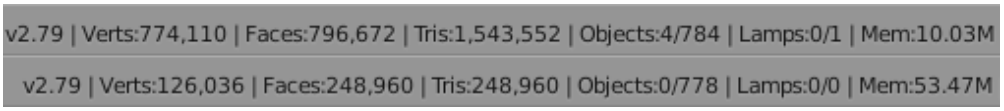


Рис. 23. Статистика сцен с молекулярной моделью инсулина, построенной Atomic Blender PDB (вверху) и построенной предложенным нами на рисунках 13 — 16 Python-скриптом (внизу)

Однако серьёзного развития эти программы не получили прежде всего из-за низкой доступности «родительских сред» - Maya и Cinema 4D в основном по части освоения и использования их инструментария и API.

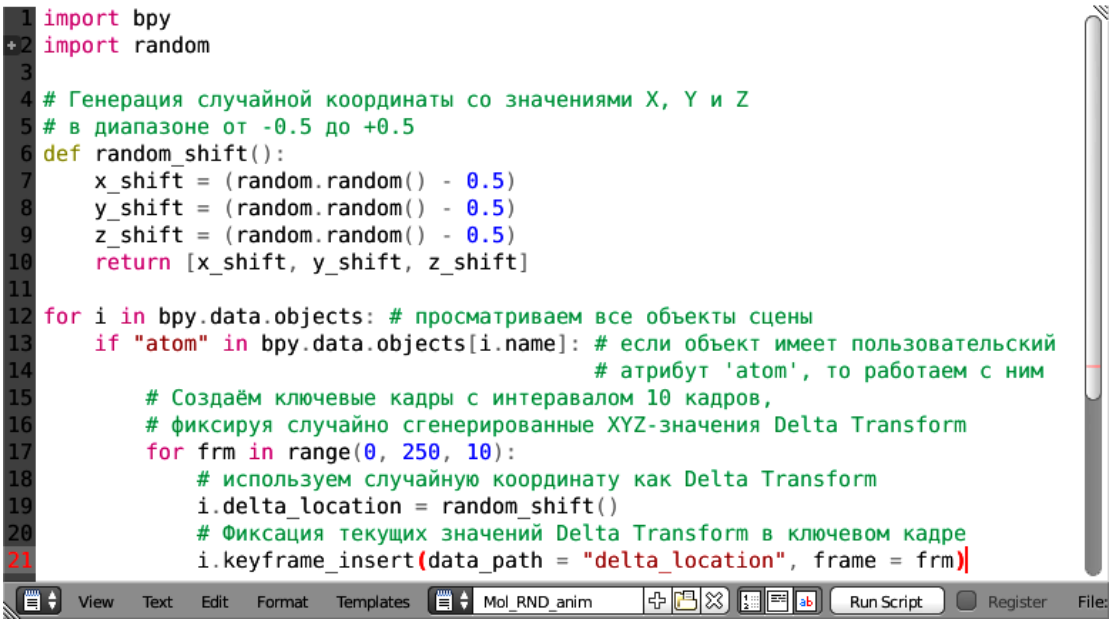
В настоящее время нам кажется весьма перспективным развитие ранее предложенных подходов к молекулярному моделированию [28 - 30] именно на базе открытой платформы Blender [1] благодаря тому характеру развития, которое приобрел данный пакет сейчас.

Воспроизведение динамики посредством Blender Python API

Приняв во внимание всё вышеизложенное, рассмотрим базовые возможности моделирования динамики биологических макромолекул в среде Blender на наглядном примере.

Достаточно очевидным для моделирования изменяющихся во времени процессов в среде любого редактора 3D-графики будет использование их средств для анимации объектов. Наиболее же часто используемым в 3D-редакторах приёмом анимации является анимация по ключевым кадрам.

Blender Python API позволяет изменять во времени практически любой параметр объекта. Для этого анимируемому параметру необходимо присвоить нужное значение, а затем «зафиксировать» его во времени, связав с номером кадра, в котором этот параметр и примет ранее «зафиксированное» значение. Пример использования вышеописанного подхода продемонстрирован в скрипте на рис. 24, который, естественно, полагает, что молекулярная модель была построена ранее с помощью Python-программы, представленной на рис. 13 — 16.



```
1  import bpy
2  import random
3
4  # Генерация случайной координаты со значениями X, Y и Z
5  # в диапазоне от -0.5 до +0.5
6  def random_shift():
7      x_shift = (random.random() - 0.5)
8      y_shift = (random.random() - 0.5)
9      z_shift = (random.random() - 0.5)
10     return [x_shift, y_shift, z_shift]
11
12 for i in bpy.data.objects: # просматриваем все объекты сцены
13     if "atom" in bpy.data.objects[i.name]: # если объект имеет пользовательский
14                                            # атрибут 'atom', то работаем с ним
15         # Создаём ключевые кадры с интеравалом 10 кадров,
16         # фиксируя случайно сгенерированные XYZ-значения Delta Transform
17         for frm in range(0, 250, 10):
18             # используем случайную координату как Delta Transform
19             i.delta_location = random_shift()
20             # Фиксация текущих значений Delta Transform в ключевом кадре
21             i.keyframe_insert(data_path = "delta_location", frame = frm)
```


Рис. 24. Python-скрипт, создающий для каждого атома молекулярной модели анимацию изменений его положения в пространстве на основе случайно сгенерированных координат

Функция *random_shift()*, объявленная в строках 6 — 10 (рис. 24), генерирует трёхкомпонентный список-вектор со случайными значениями X, Y и Z, каждое из которых может принимать значения в диапазоне от -0.5 до +0.5.

Строки 12 и 13 повторяют аналогичные строки из приведённого на рис. 19 скрипта и, соответственно, отбирают для анимации только те объекты, у которых есть пользовательский атрибут «atom».

В цикле *for* (строка 17 на рис. 24) для каждого атома создаётся анимация длительностью 240 кадров с интервалом 10. Для этого сначала в строке 19 в поля **Delta Location** (**X**, **Y** и **Z**) текущего атома (на рис. 25 выделены светло-жёлтым цветом, обозначающим анимированные параметры) записывается случайно сгенерированный функцией ***random_shift()*** сдвиг координат. А затем в строке 21 с помощью метода ***keyframe_insert()*** вставляется ключевой кадр. Аргумент ***data_path*** указывает на то, какие параметры должны быть «зафиксированы» в ключевом кадре. Нужное значение ***data_path*** можно получить (скопировать в буфер обмена) при помощи контекстного меню в графическом интерфейсе Blender (на рис. 25, выделено голубым цветом). В нашем случае это строка «**delta_location**».

В пределах группы параметров **Delta Location** доступ к отдельным её полям осуществляется по индексу, указываемому в аргументе «**index**». Так, для поля **X** значение «**index**» должно быть 0, для **Y** = 1, а для **Z**, соответственно, 2. По умолчанию метод вызывается со значением «**index**», равным -1, что предписы-

вает ядру Blender «зафиксировать» в ключевом кадре новые значения всех трёх полей параметра **Delta Location**.

После исполнения вышеописанного Python-скрипта (рис. 24) все ключевые кадры для всех анимированных объектов-атомов можно увидеть в специальном редакторе Blender — Dope Sheet (рис. 26).

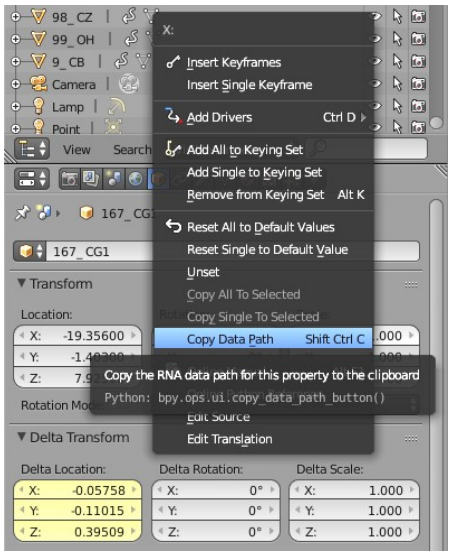


Рис. 25. Поле **Delta Location** в графическом интерфейсе Blender и контекстное меню, указывающее путь к анимируемым данным

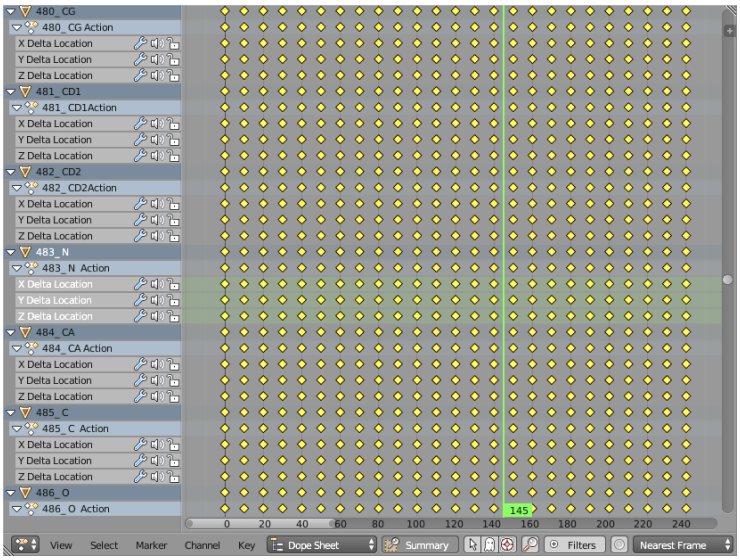


Рис. 26. Окно редактора Blender — Dope Sheet, отображающее все "ключи анимации" (жёлтые ромбики)

Как известно, одним из главных принципов анимации по ключевым кадрам в 3D-редакторах является интерполяция промежуточных значений анимированных параметров во всех кадрах, расположенных между ключевыми (рис. 27). Изменение параметров при этом может интерполироваться сплайном Безье [38, 39] (по умолчанию) и линейно, но может изменяться и ступенчато. Наглядно это продемонстрировано в результирующем видеоролике.

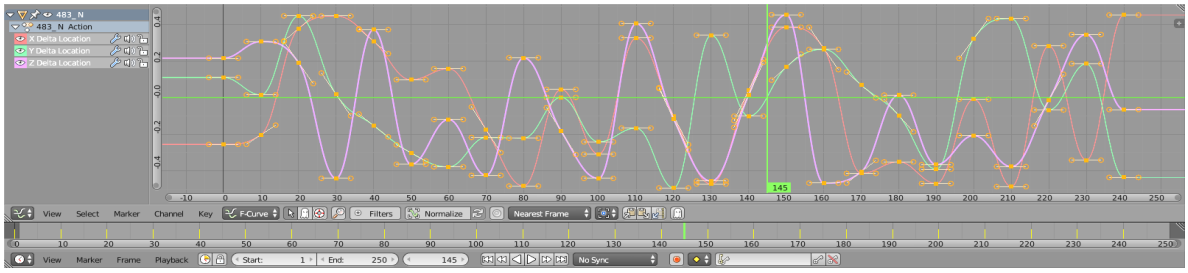


Рис. 27. Редактор анимационных кривых (сверху) и временная линия (снизу)

Опыт использования среды моделирования Blender в построении молекулярно-динамической 3D-модели β_2-адренорецептора и аналитической визуализации динамики водородной связи Ser74-Trp15

Данная работа была выполнена с использованием большинства из вышеописанных методов моделирования в среде программной платформы Blender.

Прежде всего, с помощью модифицированной[13] версии представленного здесь Python-скрипта [42] для построения динамических молекулярных моделей, из 425 PDB-файлов, представляющих собой снимки траекторий атомов, рассчитанных методом молекулярно-динамического моделирования, была построена соответствующая модель свободной (АПО) формы β_2-адренорецептора (β_2-АР). Исходные данные были получены Т.В. Богдан и Е.С. Алексеевым в работе о роли в конформационной стабильности β_2-АР консервативной межспиральной водородной связи Ser74-Trp158 в сайте связывания холестерина [41] по исходным данным рентгеноструктурного анализа с наилучшим разрешением PDB ID: 2RH1 [42, 43].

Построенная динамическая молекулярная модель воспроизводит 4 666-атомную белковую молекулу рецептора и не включает молекулы ионов солей и воды. Объём геометрических данных в сцене при этом составил 755 892 вершины и 1 493 120 треугольных граней. Динамика макромолекулы была представлена в виде траекторий всех атомов β_2-адренорецептора, позиции которых зафиксированы в виде ключевых кадров с интервалом 20 кадров. С учётом того, что длина молекулярно-динамического исследования равнялась 8.5 нс, а протяжённость анимации — 8500 кадров, «масштаб» временной шкалы составил 1 к:1 пс [40].

В уже построенной модели β_2-АР были найдены, выделены и перенесены на отдельный слой атомы, принадлежащие искомым аминокислотам Ser74-Trp158.

Сделано это было при помощи следующей короткой Python-программы:

```
import bpy
for i in bpy.data.objects:  # просматриваем все объекты сцены
    if 'GRP_NUM' in bpy.data.objects[i.name]:
        if (bpy.data.objects[i.name]['GRP_NUM'] == 158 or
            bpy.data.objects[i.name]['GRP_NUM'] == 74):
            bpy.data.objects[i.name].select = True
```

Этот Python-скрипт, последовательно перебирает все объекты сцены, наделённые пользовательским свойством "GRP_NUM" и выделяет атомы, для которых значение поля "GRP_NUM" равно 158 или 74.

13) В основном, в данной версии программы были улучшены функции считывания и интерпретации содержимого PDB-файлов, реализована обработка ошибок и возможность выборочного импорта содержимого PDB-файлов согласно списку номеров и имён импортируемых атомов [31].

Важнейшим элементом любого молекулярно-динамического исследования является контроль межатомных расстояний. В нашем случае наибольший интерес представляла динамика межспиральной водородной связи, образованной атомами кислорода № 734 (Ser74) и азота № 2074 (Trp158), на протяжении всего молекулярно-динамического исследования.

Несмотря на то, что в среде Blender существуют интерактивные инструменты для измерения расстояний между геометрическими примитивами, нами было принято решение рассчитать длину водородной связи в каждый момент времени, для которого имеется снимок состояния молекулярной системы в процессе её молекулярно-динамического исследования, то есть с интервалом 20 пс, соответствующим шагу в 20 кадров. Что и было сделано с помощью нижеследующего Python-скрипта [31]:

```
# Создаём цилиндрический "столбик гистограммы" с именем "GRAPH"
# и высотой 1. Единичная высота важна для последующего масштабирования
bpy.ops.mesh.primitive_cylinder_add(vertices = 32, radius = 0.25,
                                    depth = 1, location = (0, 0, 0.5))
bpy.context.active_object.name = "GRAPH"
grph = bpy.data.objects['GRAPH'] # получим ссылку на объект-столбик
# установим центр объекта (origin) в центр нижней грани, совпадающий
# с центром координат, простейшим способом - через 3D-курсор
bpy.context.scene.cursor_location = (0.0, 0.0, 0.0)
bpy.ops.object.origin_set(type = 'ORIGIN_CURSOR')

# Двигаемся по timeline в диапазоне анимации 0 - 8500
# с шагом расстояния между ключевыми кадрами - 20 кадров
for frm in range(0, 8500, 20):
    bpy.data.scenes['Scene'].frame_set(frm)  # устанавливаем текущий кадр
    # вычисляем расстояние, соединящее два атома
    grph.scale[2] = distance(bpy.data.objects["734_OG"].location,
                             bpy.data.objects["2074_NE1"].location)
    # фиксируем новую высоту столбика гистограммы
    grph.keyframe_insert(data_path = "scale", index = 2, frame = frm)
```

Представленный фрагмент Python-скрипта[14], вычисляет длину водородной связи на протяжении всего молекулярно-динамического исследования и фиксирует её в форме высоты столбика гистограммы.

Результаты расчёта были зафиксированы в виде высоты столбика гистограммы, выступающего в роли индикатора длины водородной связи.

Анимационная кривая (рис. 28), построенная штатным редактором Blender – Graph Editor, представляет собой график изменения межатомного расстояния

14) Из соображений компактности публикации исходный Python-скрипт был упрощён. Ряд переменных заменены константами, функция distance(pA, pB), вычисляющая расстояние между двумя атомами, не показана.

между атомами кислорода № 734 (Ser74) и азота № 2074 (Trp158) на протяжении всего молекулярно-динамического исследования.

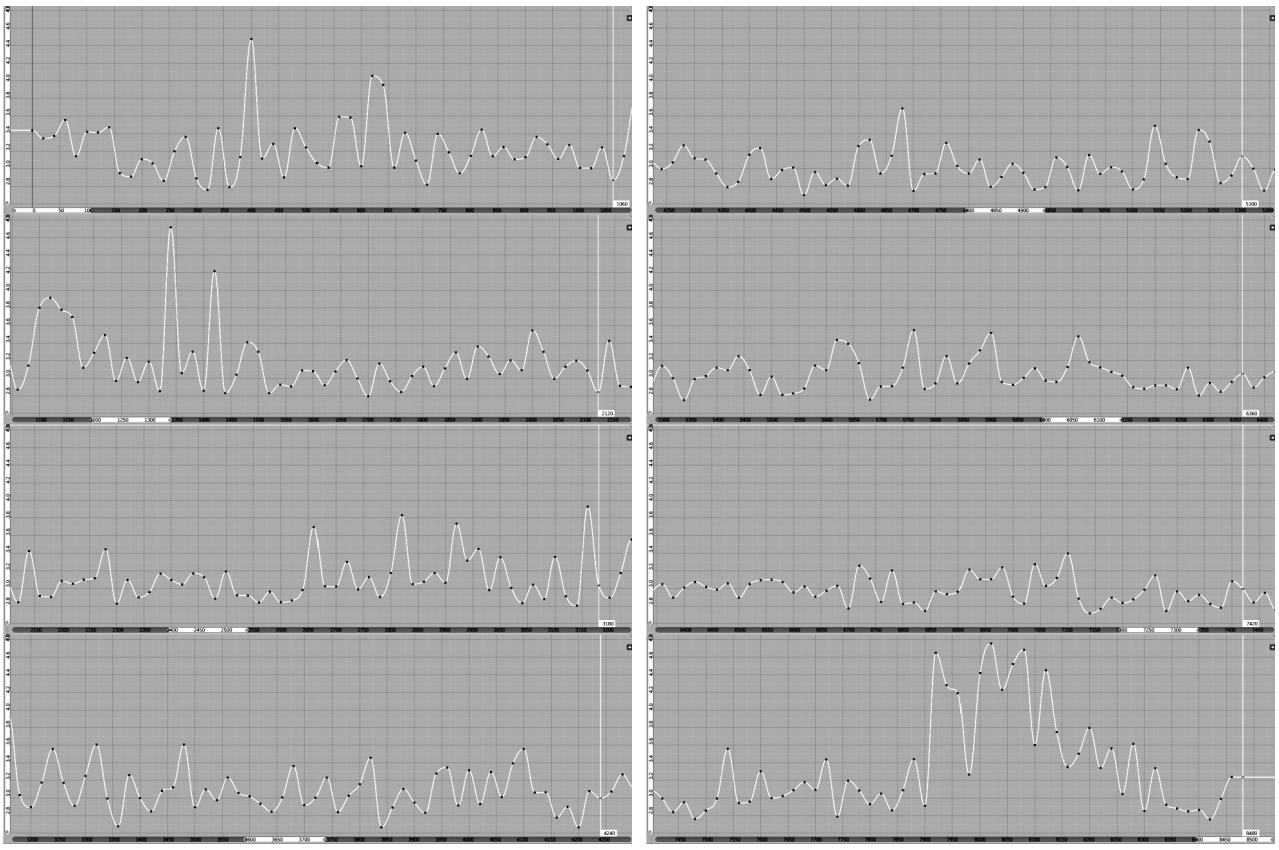


Рис. 28. Анимационная кривая длины столбика гистограммы — индикатора длины водородной связи между атомом кислорода № 734 (Ser74) и атомом азота № 2074 (Trp158). По оси X отложено время в кадрах, соответствующее времени молекулярно-динамического моделирования в пикосекундах. По оси Y — длина водородной связи в ангстремах

Использованный нами подход для фиксации рассчитанных значений в виде анимации одного из параметров геометрического объекта (столбика гистограммы) приводит к тому, что, наряду с изменением позиции каждого из атомов моделируемой молекулы, синхронно и в реальном масштабе времени изменяется и высота индикатора длины межспиральной водородной связи Ser74-Trp158. Данный «эффект» был использован при подготовке финального демонстрационного видеофрагмента [44]. При этом для лучшей наглядности на столбик гистограммы средствами Blender была наложена текстура с цветовым градиентом (рис. 29), который призван индицировать красно-жёлтыми цветовыми оттенками расстояния, при которых вероятен разрыв водородной связи.

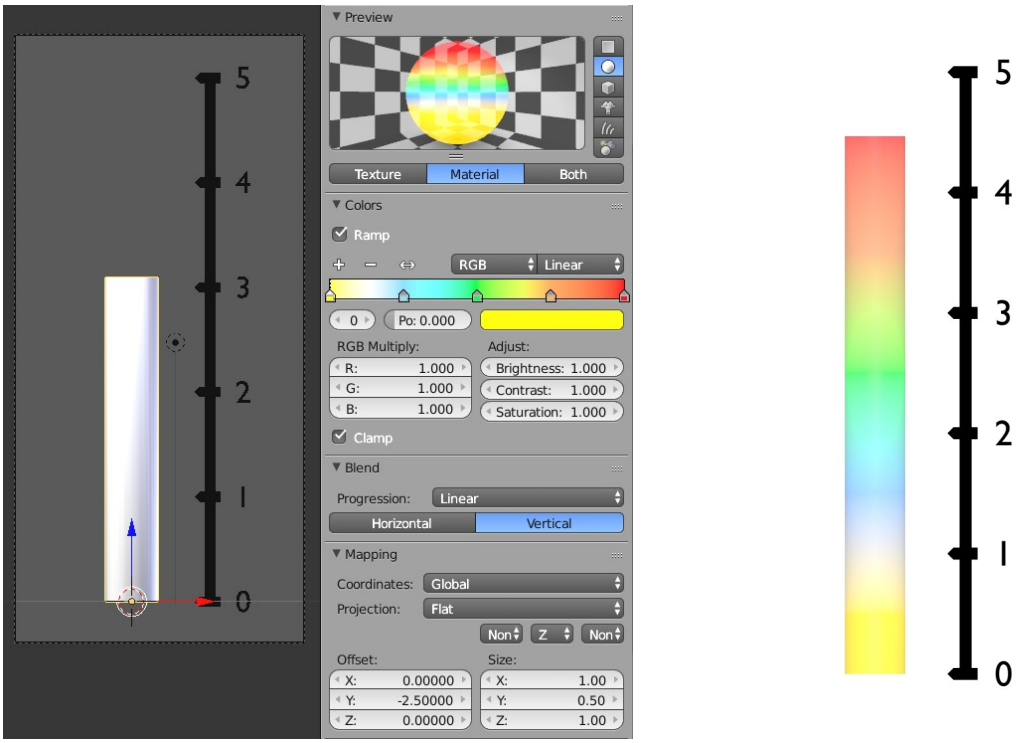


Рис. 29. Графический индикатор длины межспиральной водородной связи Ser74-Trp158. Слева направо: 3D-модель и панель настроек материала с цветовым градиентом, вид после рендеринга

Для детализированного представления динамики интересующих нас аминокислот — Ser74 и Trp158, вокруг воображаемой оси межспиральной водородной связи между их атомами кислорода № 734 (Ser74) и азота № 2074 (Trp158) был создан виртуальный аналог системы стабилизации съёмочных камер, призванный компенсировать непрерывные перемещения искомых атомов на протяжении всего молекулярно-динамического исследования [31]. Синхронные компенсирующие перемещения окружности (рис. 30), к которой были привязаны четыре перпендикулярно расположенные относительно друг друга камеры и точки их нацеливания, были также рас-

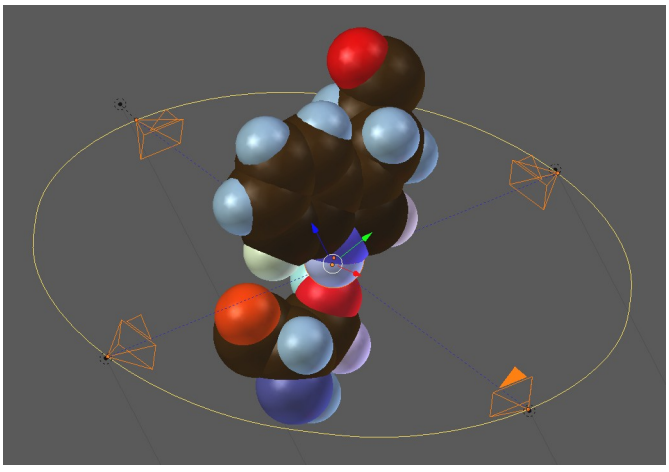


Рис. 30. Фрагмент сцены, воспроизводящей молекулярно-динамическое исследование, выполняющий "стабилизированную съёмку" области интересов — межспиральной водородной связи Ser74-Trp158

считаны и зафиксированы в виде анимации соответствующих объектов с помощью следующего Python-скрипта:

```
# Создаём "пустышку" для нацеливания камер с именем "TARGET"
bpy.ops.object.empty_add(type = 'PLAIN_AXES')
bpy.context.active_object.name = "TARGET"
# Создаём окружность - траекторию и родительский объект для камер "RIG"
bpy.ops.curve.primitive_nurbs_circle_add(radius = 9)
bpy.context.active_object.name = "RIG"

# Двигаемся по таймлайну в диапазоне анимации с шагом, равным
# расстоянию между ключевыми кадрами
for frm in range(0, 8500, 20):
    bpy.data.scenes['Scene'].frame_set(frm)  # устанавливаем текущий кадр
    # вычисляем середину отрезка, соединяющего два атома
    # и сразу перемещаем в эту позицию "пустышку"
    trgt = bpy.data.objects['TARGET']
    rig  = bpy.data.objects['RIG']
    trgt.location = midpnt(bpy.data.objects["734_OG"].location,
                           bpy.data.objects["2074_NE1"].location)
    # центр окружности - траектории для камер помещаем в середину
    # отрезка, "связывающего" атомы и совпадающего с "пустышкой"
    rig.location = trgt.location
    # фиксируем позицию "пустышки" в ключевом кадре
    trgt.keyframe_insert(data_path = "location", frame = frm)
    # фиксируем позицию траектории камер в ключевом кадре
    rig.keyframe_insert(data_path = "location", frame = frm)

    # поворачиваем плоскость окружности и "пустышку"
    # перпендикулярно оси, соединяющей атомы
    trgt.rotation_mode = 'QUATERNION'
    rig.rotation_mode  = 'QUATERNION'
    direction = (bpy.data.objects["2074_NE1"].location -
                 bpy.data.objects["734_OG"].location)
    trgt.rotation_quaternion = direction.to_track_quat('Z', 'Y')
    rig.rotation_quaternion  = direction.to_track_quat('Z', 'Y')
    # фиксируем вращение в ключевом кадре
    trgt.keyframe_insert(data_path = "rotation_quaternion", frame = frm)
    rig.keyframe_insert(data_path  = "rotation_quaternion", frame = frm)
```

Для облегчения визуальной идентификации ближайших к водородной связи атомов аминокислот Ser74-Trp158, на поверхность сфер, обозначающих эти атомы, были назначены индивидуальные материалы[15] с текстурами, содержащими

15) Данный этап построения сцены был выполнен исключительно штатными средствами Blender – его редактором материалов со сферическим типом проецирования на основании генерируемых для сферы UV-координат и значением влияния текстуры на параметры диффузного канала материала Intensity и Color – 0.2.

обозначения этих атомов в PDB-файле [21] в формате: «НомерАтома_ИмяАтома».

В конечном итоге созданная сцена была визуализирована встроенной системой рендеринга Blender в форме пяти раздельных видеопотоков (длительностью 8500 кадров), демонстрирующих точку зрения из четырёх виртуальных камер, расположенных на окружности вокруг искомой водородной связи, а также специальной ортогональной камеры, визуализирующей динамику столбика гистограммы — индикатора длины межспиральной водородной связи.

И, наконец, в той же среде Blender был выполнен финальный композитинг (рис. 31) пяти синхронизированных видеопотоков, представляющий собой аналитическую визуализацию динамики межспиральной водородной связи Ser74-Trp158 в ходе молекулярно-динамического исследования свободной (АПО) формы β_2-адренорецептора [44].

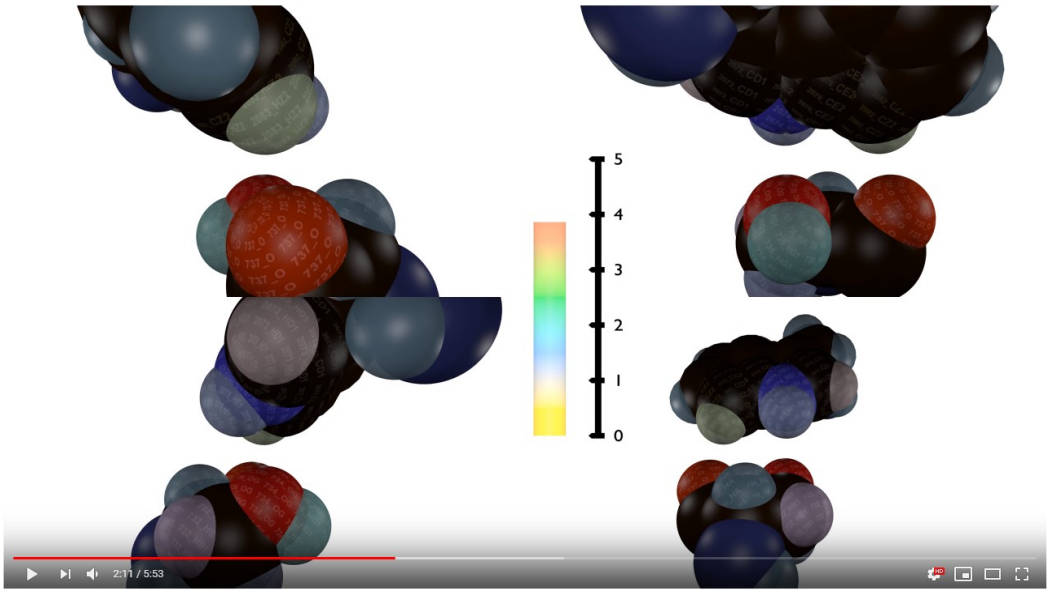


Рис. 31. Стоп-кадр 4K-видеофрагмента (2160p), размещённого на Интернет-видеохостинге [44]

Заключение

В ходе разработки методов моделирования объектов и процессов естественно-научных дисциплин в среде открытой программной платформы Blender была продемонстрирована её состоятельность в качестве инструмента, пригодного для решения задач по аналитической визуализации в вышеназванных областях исследований как с точки зрения скорости взаимодействия с построенными моделями, так и с точки зрения функционала, не уступающего таковому в узкоспециализированных программах [45, 46]. Так, например, воспроизведение

динамики 4 666-атомной молекулы β_2-адренорецептора и любые манипуляции с ней осуществлялись в реальном масштабе времени даже на устаревшем оборудовании, а при помощи простых Python-скриптов с использованием Blender API был выполнен ряд типичных для редакторов молекулярных моделей задач по автоматизированному поиску необходимых атомов (по заданными критериям), выделение необходимых групп атомов и перенос их на отдельные слои для выстраивания необходимого визуализирующего окружения и, наконец, автоматизированное выполнение вспомогательных расчётов межатомных расстояний, длин связей и углов между ними на протяжении всей длительности молекулярно-динамического исследования [31, 40].

Таким образом, платформа Blender позволяет относительно просто и быстро строить точные модели объектов, форма которых представлена в виде трёхмерных поверхностей, описанных аналитически или численно. При этом данные могут быть представлены как в виде текстовых таблиц, так и в виде графических файлов. Штатные средства Blender — редакторы анимации, материалов и текстур — позволяют достигать комплексного (многомерного) представления информации о моделируемом объекте или процессе, что было показано нами на примере моделирования поверхности участка дна Северного Ледовитого океана [10, 11, 17, 18] и аналитической визуализации молекулярно-динамического исследования β_2-адренорецептора [40, 31, 41 - 44].

В конечном итоге, в данной работе, как нам кажется, была продемонстрирована и состоятельность Blender Python API как средства математического моделирования, оснащённого обширным набором встроенных в данную программную среду функций и способного при необходимости привлечь гигантский потенциал библиотек языка Python [3 - 6].

Также в ходе решения тестовой задачи по построению аналитической визуализации была подтверждена стабильность межспиральной водородной связи Ser74-Trp158 в сайте связывания холестерина на протяжении молекулярно-динамического исследования длительностью 8.5 нс [40].

Библиографический список

1. Blender is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline — modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. [Электронный ресурс]. URL: https://www.blender.org (дата обращения: 29.10.2018).
2. Blender Documentation Contents: Blender 2.78.0 e8299c8 - API documentation. [Электронный ресурс]. URL: https://docs.blender.org/api/blender_python_api_current/contents.html (дата обращения: 29.10.2018).

3. Langtangen Hans Petter, Python Scripting for Computational Science (Texts in Computational Science and Engineering), Springer; 3rd edition, 2009, pp.756.
4. Доля П.Г., Введение в научный Python. Харьковский национальный университет. Факультет математики и информатики. 2016, с.265.
5. Программирование и научные вычисления на языке Python. [Электронный ресурс]. URL: https://ru.wikiversity.org/wiki/Программирование_и_научные_вычисления_на_языке_Python (дата обращения: 29.10.2018).
6. NumPy for Matlab users. [Электронный ресурс]. URL: https://docs.scipy.org/doc/numpy-1.15.0/user/numpy-for-matlab-users.html (дата обращения: 29.10.2018).
7. Blender 2.78.0 e8299c8 - API documentation: Quickstart Introduction. [Электронный ресурс]. URL: https://docs.blender.org/api/blender_python_api_current/info_quickstart.html (дата обращения: 29.10.2018).
8. Python 3.x в контексте 3D-редактора Blender. [Электронный ресурс]. URL: http://blender3dlove.blogspot.com/2013/05/1-python-blender.html (дата обращения: 29.10.2018).
9. Blender: Three ways to create objects. [Электронный ресурс]. URL: https://en.blender.org/index.php/Dev:Py/Scripts/Cookbook/Code_snippets/Three_ways_to_create_objects (дата обращения: 29.10.2018).
10. Флоринский И.В., Филиппов С.В. Трехмерное геоморфометрическое моделирование дна Северного Ледовитого океана: применение пакета Blender // И.К. Лурье (ред.), Национальная картографическая конференция – 2018: Сборник тезисов, Москва, 16–19 окт. 2018. М.: Географический факультет МГУ, 2018, с. 268-271.
11. Флоринский И.В., Филиппов С.В. Трехмерное моделирование рельефа: применение пакета Blender // ИнтерКарто/ИнтерГИС 24, Материалы Международной конференции, Петрозаводск, Бонн, Анкоридж, 19 июля – 1 августа 2018, Ч. 2. Петрозаводск: КарНЦ РАН, 2018, с. 250-261.
12. Catmull E., Clark J. Recursively generated B-spline surfaces on arbitrary topological meshes. Computer-Aided Design. 1978. V. 10, No 6. pp. 350–355. doi:10.1016/0010-4485(78)90110-0
13. Phong Bui Tuong. Illumination of Computer-Generated Images // Department of Computer Science, University of Utah, UTEC-CSs-73-129, July 1973.
14. Phong Bui Tuong. Illumination for Computer Generated Pictures // Comm. ACM, Vol. 18(6):311-317, June 1975.
15. Florinsky I.V., Filippov S.V., Abramova A.S., Zarayskaya Yu.A., Selezneva E.V. Towards geomorphometric modelling of the topography of the arctic ocean floor // Proceedings of the 7th International Conference on Cartography

& GIS, 18-23 June 2018, Sozopol, Bulgaria. Sofia: Bulgarian Cartographic Association, 2018, (in press).
16. Jakobsson M., Mayer L., Coakley B., Dowdeswell J.A., Forbes S., Fridman B., Hodnesdal H., Noormets R., Pedersen R., Rebesco M., Schenke H.W., Zarayskaya Y., Accettella D., Armstrong A., Anderson R.M., Bienhoff P., Camerlenghi A., Church I., Edwards M., Gardner J.V., Hall J.K., Hell B., Hestvik O., Kristoffersen Y., Marcussen C., Mohammad R., Mosher D., Nghiem S.V., Pedrosa M.T., Travaglini P.G., Weatherall P. The International Bathymetric Chart of the Arctic Ocean (IBCAO) Version 3.0 // Geophysical Research Letters. – 2012. – Vol. 39. – No. 12. – L12609.
17. Флоринский И.В., Филиппов С.В., Абрамова А.С., Зарайская Ю.А., Селезнева Е.В. Разработка системы для моделирования рельефа дна Северного Ледовитого океана // Тезисы докладов 12-й Международной конференции «Интеллектуализация обработки информации» ИОИ-2018, Гаэта, Италия, 8–13 окт. 2018. М.: Торус-Пресс, 2018, с. 170.
18. Флоринский И.В., Филиппов С.В. Трехмерное моделирование рельефа: применение пакета Blender // ИнтерКарто/ИнтерГИС 24, Материалы Международной конференции, Петрозаводск, Бонн, Анкоридж, 19 июля – 1 августа 2018, Ч. 2. Петрозаводск: КарНЦ РАН, 2018, с. 250-261.
19. Walter L. Koltun. (1965). Precision space-filling atomic models. *Biopolymers*. **3**, 665-679.
20. Levinthal C. (1966). Molecular model-building by computer. *Scientific American*. **214**, 42–52.
21. PDB File Format Documentation. [Электронный ресурс]. URL: http://www.wwpdb.org/documentation/file-format (дата обращения: 29.10.2018).
22. Smith, G.D., Duax, W.L., Dodson, E.J., Dodson, G.G., Degraaf, R.A.G., Reynolds, C.D. The Structure of Des-Phe B1 Bovine Insulin // Acta Crystallogr. 1982, Sect.B 38: 3028. [Электронный ресурс]. URL: https://www.rcsb.org/structure/2ins (дата обращения: 29.10.2018).
23. Зефиров Ю.В., Зоркий П.М. Журн. структур. химии. 1974. Т.15, No1. С.118-122.
24. Зефиров Ю.В., Зоркий П.М. Журн. структур. химии. 1976. Т.17, No 4. С.745-746.
25. Los Alamos National Laboratory: PERIODIC TABLE OF ELEMENTS. [Электронный ресурс]. URL: https://periodic.lanl.gov/list.shtml (дата обращения: 29.10.2018).
26. Robert B. Corey and Linus Pauling (1953): Molecular Models of Amino Acids, Peptides, and Proteins. Review of Scientific Instruments, Volume 24, Issue 8, pp. 621—627. doi:10.1063/1.1770803

27. The Corey-Pauling-Koltun Models to construct macromolecules soon will be available for research and teaching. Copyright, W. L. Koltun, July 1965.
28. Филиппов С.В., Соболев Е.В., Использование технологий профессиональной компьютерной графики для визуализации результатов научных исследований // Компьютеры и суперкомпьютеры в биологии / Под ред. Лахно В.Д., Устинина М.Н., Москва-Ижевск: Институт компьютерных исследований, 2002. стр.476-497.
29. Филиппов С.В. Методы и алгоритмы визуализации структурных и динамических данных, характеризующих макромолекулярные структуры. *Диссертация на соискание ученой степени к.ф.-м.н*. Пущино. 2014.
30. Филиппов С.В., Сивожелезов В.С., Ким В.Л., Сычев В.В., Устинин М.Н. Программа трехмерного моделирования и визуализации конформационной динамики биомакромолекул Maya-K-PDB // Математическая биология и биоинформатика, 2015, № 1, Том 10, с.260-282.
31. Филиппов С.В // Методы работы с динамическими молекулярными моделями, построенными в среде открытого 3D-редактора Blender // Доклады Международной конференции "Математическая биология и биоинформатика". Под ред. В.Д. Лахно. Том 7. Пущино: ИМПБ РАН, 2018, 14-19 октября. doi:10.17537/icmbb18.62
32. Программа 3ds Max для 3D-моделирования и визуализации. [Электронный ресурс]. URL: https://www.autodesk.ru/products/3ds-max/overview (дата обращения: 29.06.2018).
33. Realsoft Graphics. URL: http://www.realsoft.com (дата обращения: 29.06.2018).
34. Clarafi|tools Molecular Maya (mMaya) is a free plugin for Autodesk Maya that lets users import, model and animate molecular structures. [Электронный ресурс]. URL: https://clarafi.com/tools/mmaya/ (дата обращения: 29.06.2018).
35. Программа для 3D-анимации, моделирования и визуализации Maya. URL: https://www.autodesk.ru/products/maya/overview (дата обращения: 29.06.2018).
36. Atomic Blender PDB. [Электронный ресурс]. URL: http://development.root-1.de/Atomic_Blender_PDB.php (дата обращения: 29.06.2018).
37. Maxon Cinema 4D. [Электронный ресурс]. URL: https://www.maxon.net/ru/produkty/cinema-4d/obzor/ (дата обращения: 29.06.2018).
38. Piegl, L. Fundamental Developments of Computer Aided Geometric Design. San Diego, CA: Academic Press, 1993.

39. Bartels, R. H.; Beatty, J. C.; and Barsky, B. A. "Bézier Curves." Ch. 10 in An Introduction to Splines for Use in Computer Graphics and Geometric Modelling. San Francisco, CA: Morgan Kaufmann, pp. 211-245, 1998.
40. Филиппов С.В., Сивожелезов В.С. // Метод построения динамических молекулярных моделей в среде открытой 3D-платформы Blender на примере β2-адренорецептора // Доклады Международной конференции "Математическая биология и биоинформатика". Под ред. В.Д. Лахно. Том 7. Пущино: ИМПБ РАН, 2018, 14-19 октября. doi:10.17537/icmbb18.23
41. Богдан Т.В., Алексеев Е.С. Роль консервативной межспиральной водородной связи Ser74-Trp158 в сайте связывания холестерина в конформационной стабильности β2-адренорецептора (молекулярно-динамическое моделирование). Журнал структурной химии. 2017. Т. 58. № 2. С. 402–409. doi:10.1134/S002247661702024X
42. Cherezov V., Rosenbaum D.M., Hanson M.A., Rasmussen S.G., Thian F.S., Kobilka T.S., Choi H.J., Kuhn P., Weis W.I., Kobilka B.K., Stevens R.C. Science. 2007. V. 318. № 5854. P. 1258.
43. High resolution crystal structure of human B2-adrenergic G protein-coupled receptor. [Электронный ресурс]. URL: http://pdb.org/pdb/explore/explore.do?structureId=2rh1 doi:10.2210/pdb2RH1/pdb (дата обращения: 29.06.2018).
44. Визуализация (Blender) мол.динамики Ser74–Trp158 B2-адренорецептора APO. [Электронный ресурс]. URL: https://youtu.be/uQwWV1xm3xk (дата обращения: 29.06.2018).
45. Home Page for RasMol and OpenRasMol: Molecular Graphics Visualisation Tool. [Электронный ресурс]. URL: http://www.openrasmol.org (дата обращения: 29.06.2018).
46. PyMol: a molecular visualization system on an open source foundation, maintained and distributed by Schrödinger. [Электронный ресурс]. URL: http://www.pymol.org (дата обращения: 29.06.2018).