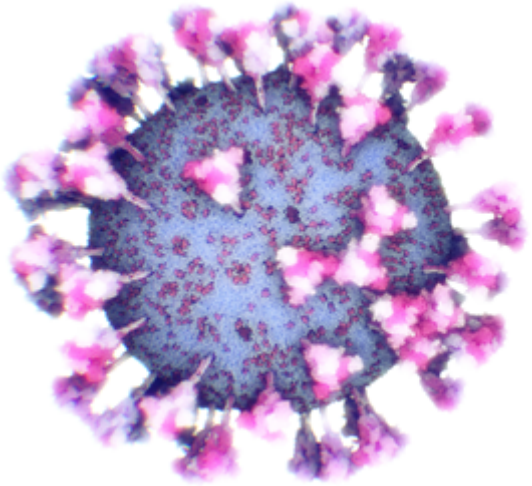
MesoCraft



This implementation was created as part of our paper published at IEEE VIS 2020.

Ngan Nguyen, Ondrej Strnad, Tobias Klein, Deng Luo, Ruwayda Alharbi, Peter Wonka, Martina Maritan, Peter Mindek, Ludovic Autin, David Goodsell, Ivan Viola. Modeling in the Time of COVID-19: Statistical and Rule-based Mesoscale Models. IEEE Transactions on Visualization and Computer Graphics, 2020.

Important part is the submission video that can be used as a tutorial available here: https://cemse.kaust.edu.sa/nanovis/news/modeling-time-covid-19-statistical-and-rule-based-mesoscale-models

Prerequisites

This software is built on top of Qt 5.14.1 library (https://download.qt.io/archive/qt/5.14/5.14.1/). All the libraries required for running are included. However, on a few systems the full installation of Qt 5.14.1 was necessary (this issue is being solved).

When working with large models (>100000 instances) the demand on resources can be significant. We advise to use both high-end NVidia's GPU and Intel's CPU with enough memory (>16 GB). However, testing some basic rules can be performed on a regular notebook.

Installation

Unzip the archive and run MesoCraft.exe.

Files naming convention

PDB names are supposed to be named "<name>.pdb".

Model files are supposed to be named "<custom-name>.json". This is important, the algorithm decides based on the name how to process every element.

Manipulation with objects

Left mouse button click in the scene to select an object.

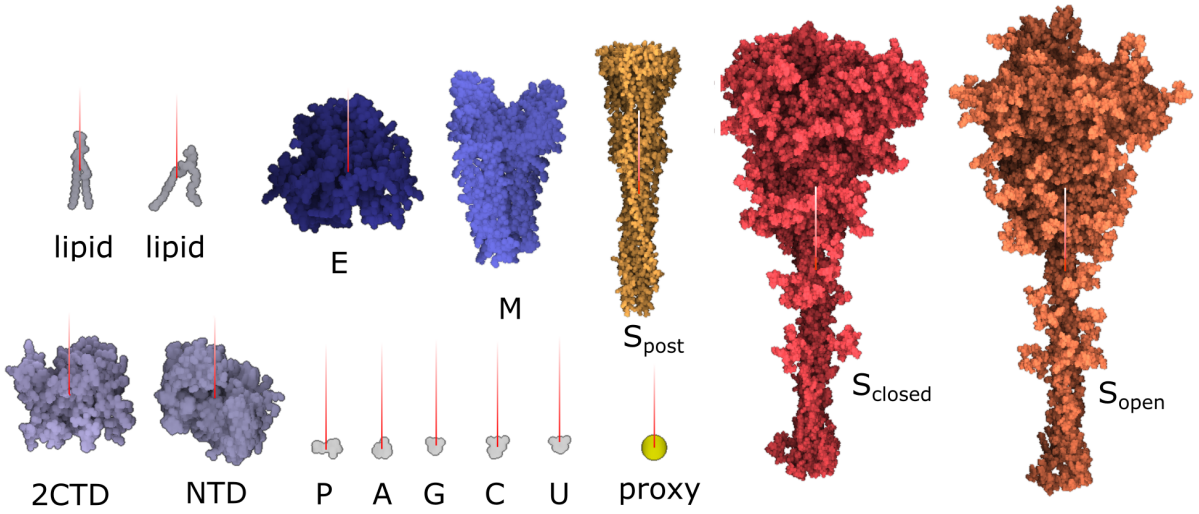
Holding **left mouse** button and moving with the mouse rotates the camera.

Mouse wheel performs zoom. The amount of zoom can by multiplied by holding Ctrl and/or Shift,

Using the gizmo to move (key **T**)|rotate (key **R**)|*scale (key **E**)* the object.

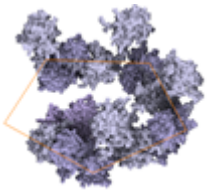
Building Blocks

- **Protein instances** (in the form of PDB files)

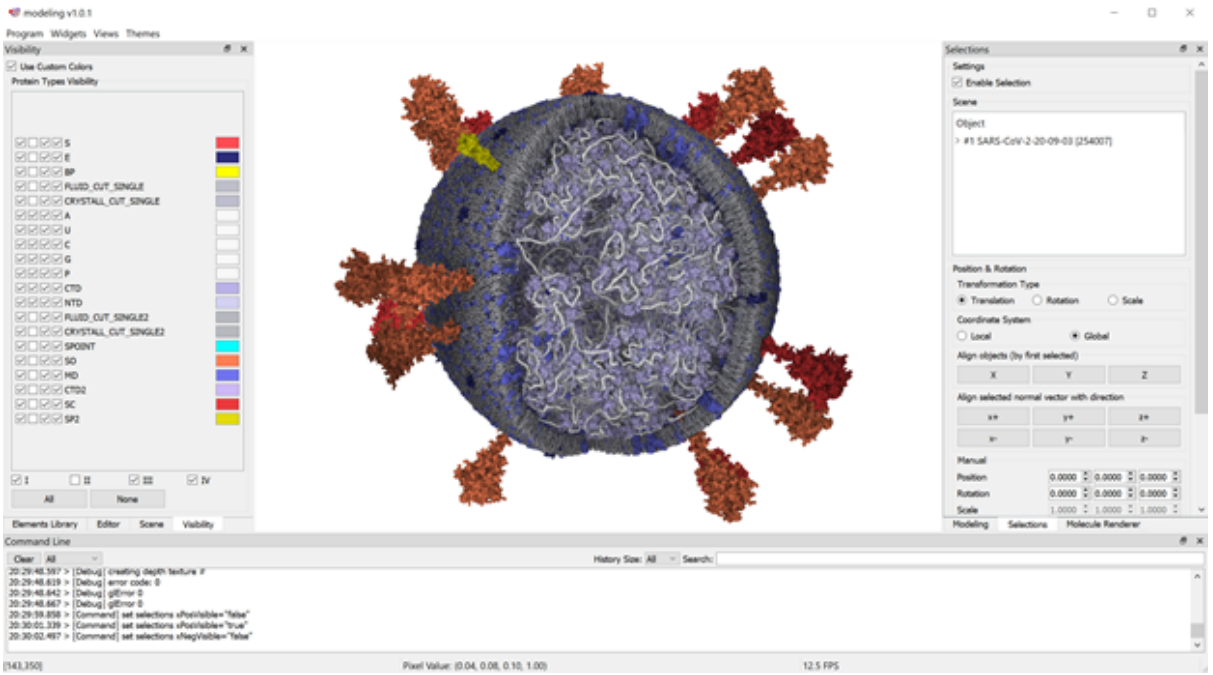


- **Skeletons**

- **Skeletal Model** (set of elements generated by a rule)



Environment



The environment is based on Qt GUI library. Individual windows can be relocated or hidden/shown.

Input/Output

Program->Save Model

Exports the current content of the scene into TXT file, where on every row one element instance is stored. No rules are exported. This model can be loaded using ***Program->Load Model*** function. For correct loading of the stored model all the protein instances that the model contains have to be in the library.

Program->Export->Rules

Exports into selected directory rules currently presented in the scene in the form of .json files. These files after copying into *work/data/modeling/models* folder can be after program restart used.

Program->Load Model

Loads previously stored model using ***Program->Save Model***.

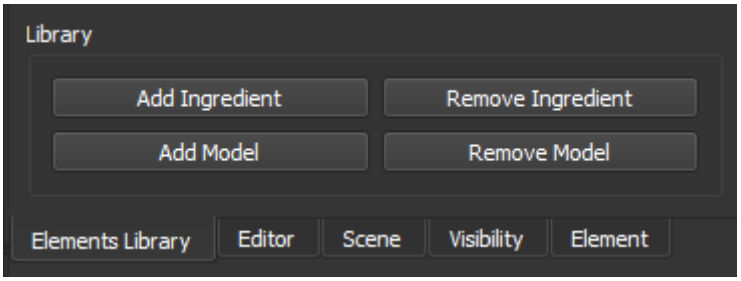
Program->Export->Protein Colors

Exports the currently set color scheme. For more details, see **Visibility** section.

Program->Import->Protein Colors

Load the previously stored color scheme from the file.

Adding additional items into the library



In the bottom part of the *Elements Library* panel.

Adding Ingredients

Add Ingredient opens a dialog where the user can select one protein file (PDB, mmCIF, …). After the confirmation the item is added into the library.

Important! After adding an ingredient, the software needs to be restarted in order to rebuild internal data structure. Several ingredients can be added within one session. The software can be restarted only once after the all desired ingredients are added.

Remove Ingredient removes the currently selected ingredient from the database. The software needs to be restarted to apply changes as well.

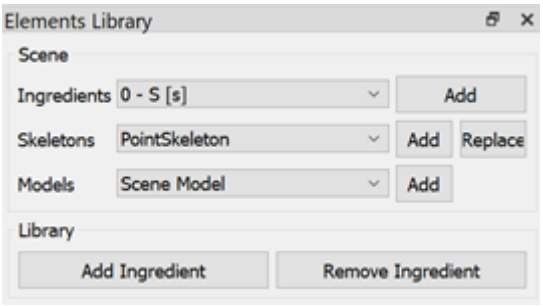
Adding rules

Add Model opens a dialog where the user can select previously exported rules (.json files). These will be listed in the *Models* combobox. The software restart is not required. Alternatively, the rule files can be manually copied into “work/data/modeling/models” directory (restart is required).

Remove Model removes the selected rule from the Model combobox. No restart is required.

Notice! When importing a rule, the user has to be sure that the elements used in the rule are available in the Ingredients library. Otherwise, the rule will not work!

Elements Library



Add elements into the scene

Ingredient – add the selected ingredient into the scene. Ingredients can be added/removed to/from the library.

Skeletons – adds the selected skeleton into the scene. Only presented skeletons are implemented.

Models – adds selected model (including rules) into the scene. All models (*.json) files that are presented in “data/modeling/models” directory are listed. This models can be create using **Save Rules** option from the main menu.

CustomTriangleSkeleton loads OBJ mesh from *work/data/models/model.obj* file.

Replacing the skeleton

In several cases it is easier to prepare rules on a single triangle skeleton. Once these rules are ready, the triangular skeleton can be for example replaced by CustomTriangleMesh skeleton. These rules are then applied on every triangle of the mesh. In order to perform this action, a skeletal model has to be selected. In the next step, the target skeleton is select

Revert – reverts all elements created by the selected rule from the scene.

Edit – opens rule edit dialog where all the parameters related to the rule can be set (see Rule Settings section).

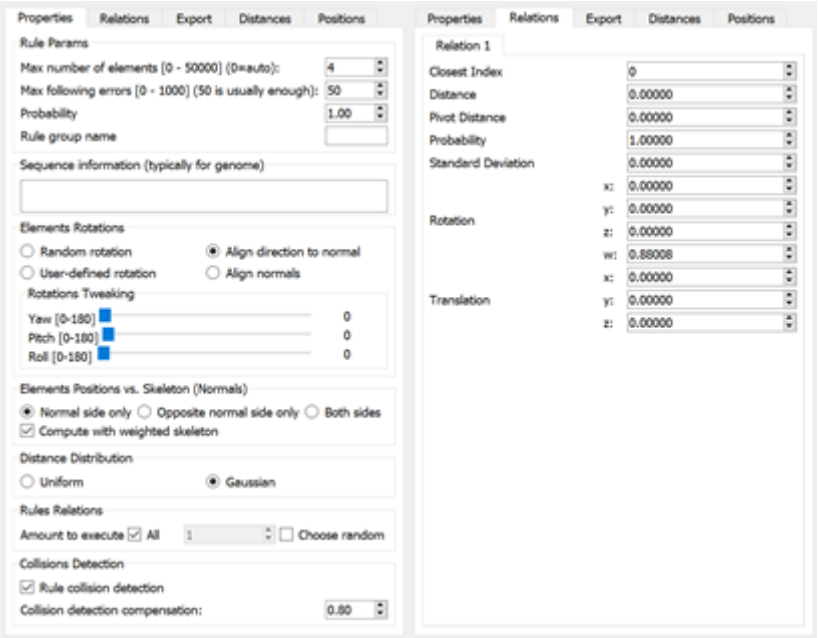
Remove – removes the rule from the scene (performs Revert first).

Selected to Up – rotates the selected object so their normal vector is oriented in the up direction.

Set Normal – The instance is rotated in the way that direction up will be the new normal vector. Then, clicking Set Normal button updates the normal vector of the selected instance within the library.

Set Bounding Sphere – every instance can have customized bounding sphere. This bounding sphere can be made visible in the *Scene* menu. Once bounding spheres are visible, the gizmo updates this bounding sphere. Using mouse the position can be changed and using -/+ keys it's size can be updated.

Rule Settings



These forms are to be able to update the settings of every rule. Based on the type of the rule, only selected parameters can influence the resulting model. For more details please see the paper.

Rule window

Max number of elements – maximum number of elements that can be created by this rule.

Max following errors – the algorithm is stopped after this specified amount of consecutive errors (collisions) is generated.

Probability – the probability of selecting this rule once applying the rules in the same group.

Rule group name – customizable string that specifies the group name into this rule belongs to.

Sequence information – specific item for creating RNA. (description: TBD)ed in the menu and the button **Replace** is clicked.

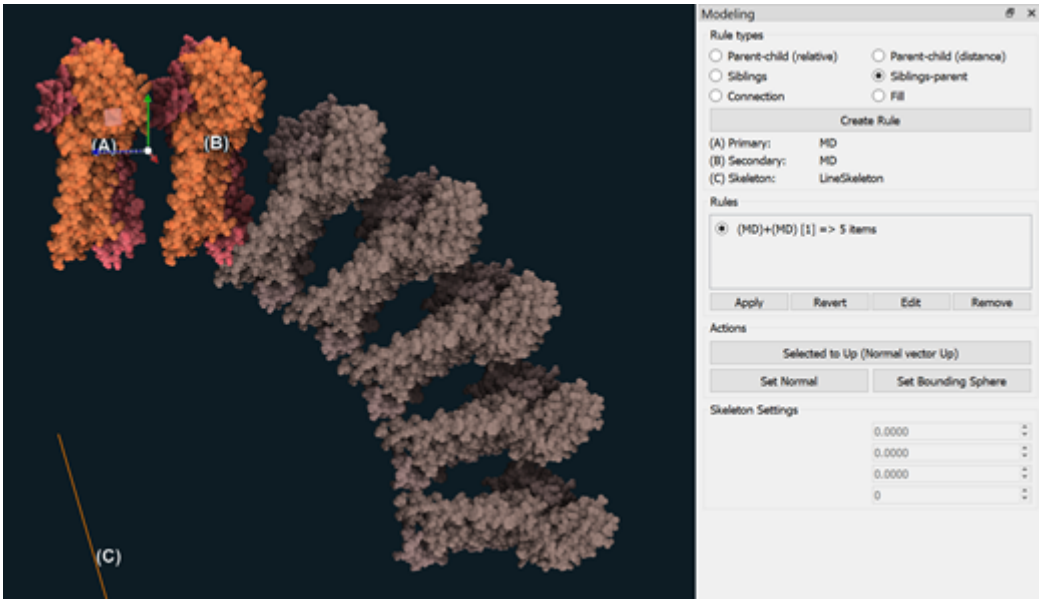
Add elements into the library

Add Ingredient – adds ingredient (user selected PDB file) into the library. Updates all necessary dependencies. Restart is required in order to update cache file.

Remove Ingredient – removes the selected ingredient from the library. Restart is required in order to update cache file.

The elements can be added manually by modifying library (*modeling-elements-library.json)* file in *work/data/modeling* directory. Every time the library is updated, cache file (*work/data/modeling/cache-file/ingredients-cache.bin*) has to be deleted and the software needs to be restarted. The cache file is created automatically during the start up.

Modeling



Rules types – which rule type is about to be created. For the difference, please see the paper. Note, that the order of selected elements matters.

Create rule – when element(s) and rule type is selected, this creates the rules itself.

(A), (B), (C) – helper identifiers that visualize what elements are expected in order to create the selected type of rule.

Rules – currently loaded rules in the scene. To select a rule, click on the respective radiobutton.

Apply – apply the selected rule on all the respective elements in the scene IF no element is selected. If a set of elements is selected, the application is performed only on those.

Elements rotation – type of rotation applied on the newly generated element in the scene. For more info, see ***Elements rotation*** subsection.

Elements positions vs. Skeleton – where the element is about to be placed with the respect to the skeleton.

Compute with weighted skeleton – if the rule is applied on triangular mesh with triangles evaluated by different probabilities, this probability is taken into account.

Distance Distribution – type of distance distribution used

Rules relations – typically sibling rule. How many relations from the set of defined relations will be applied.

Collision detection – whether collision detection is enabled

Collision detection compensation – the ratio that is applied to the bounding sphere to allow overlapping.

Relations window

Closest index – the closest index of the skeleton when the relation was created (parent-child rule)

Distance – distance to the skeleton the relation was created (parent-child rule)

Pivot distance – distance to the closest point belonging to the skeleton

Probability – probability to select this relation when only a subset of all relations is selected

Standard Deviation – allowed standard deviation for the distance

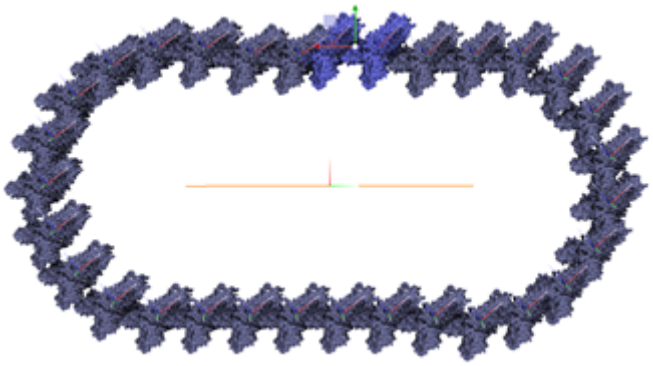
Rotation – stored rotation of the element

Translation – stored translation of the element

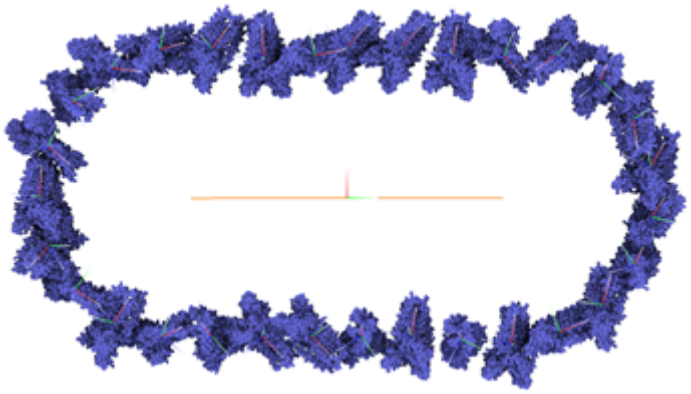
Elements rotation

Once a position of the newly generated element is estimated, the rotation is computed. This computation can be modified by four different criteria:

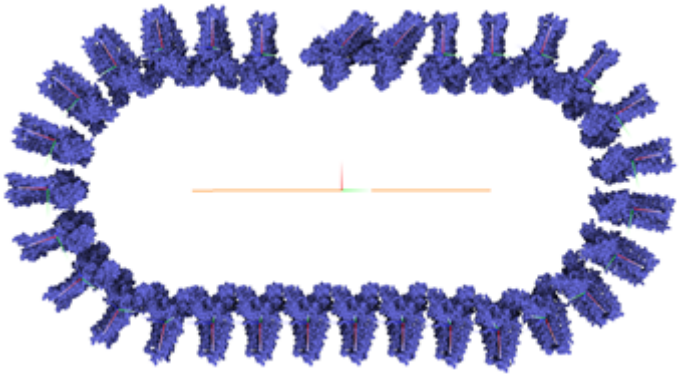
1) ***User-defined rotation*** – all instances have the same rotation as the template instances (highlighted).



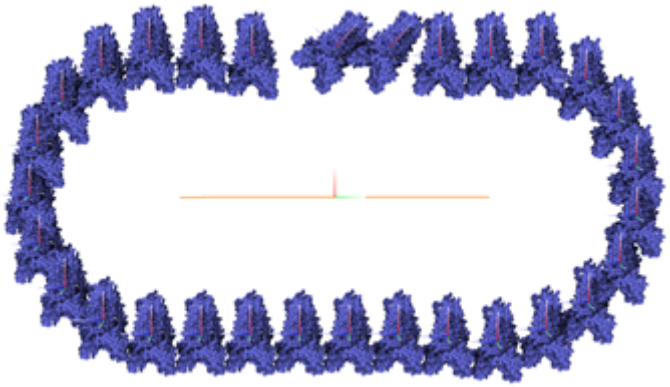
2) ***Random rotation*** – all instances have random rotation.



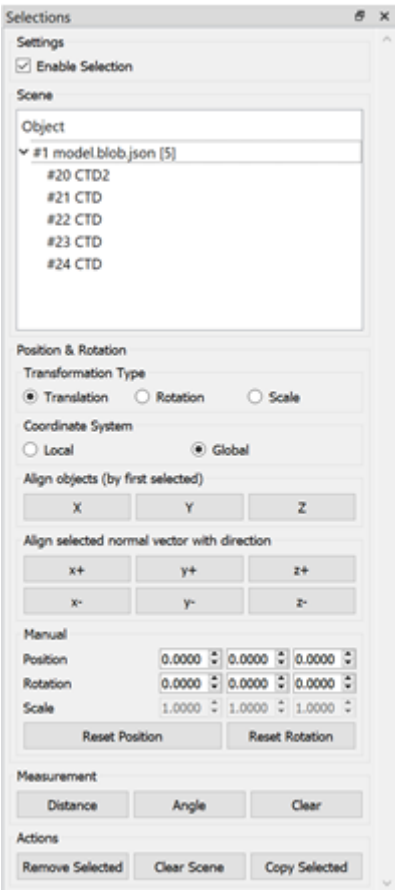
3) ***Align to skeleton*** –normal vector of an instance is aligned with the direction into the skeleton.



4) ***Align normals*** – normal vector of all instances is aligned with the normal vector of the skeleton.



Selections



Enable selection – enables the selection of objects.

Scene – list of elements in the scene. Clicking in the tree elements can be selected.

Position & Rotation – switching between controlling mode of selected elements.

Coordinate System – which coordinate system to use while controlling the selected elements.

Align objects – aligns all selected objects to the very first selected object in the given axis.

Align selected normal vector – rotates the selected objects in the way that the normal vector is aligned with the selected direction.

Manual – manual settings of the position and rotation

Measurement Distance – measures a distance between first two selected objects

Measurement Angle – measures the angle between first three selected objects

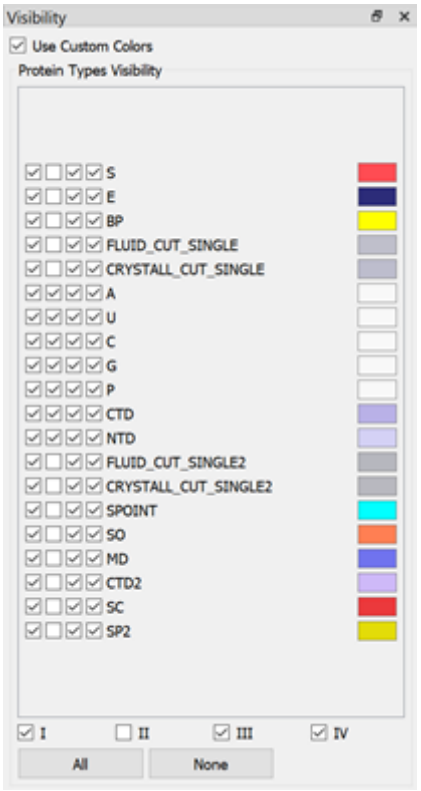
Measurement Clear – removes all measurement objects from the scene

Remove Selected – removes selected elements from the scene

Clear Scene – removes all elements from the scene

Copy Selected – creates a copy of selected elements

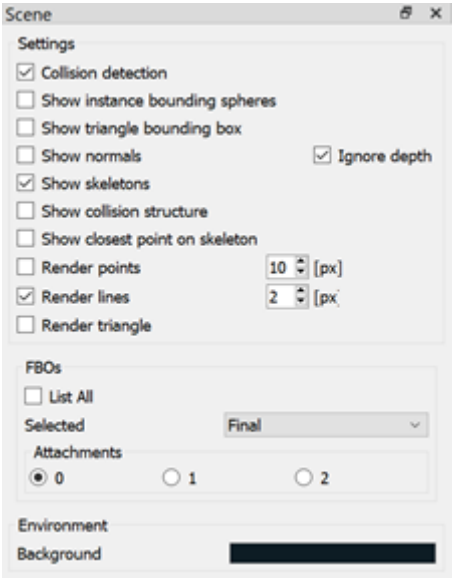
Visibility



Use Custom Colors – there is some color scheme in the software by default. In order to use colors stored in the library this checkbox needs to be selected. The colors can be redefined by clicking on the color rectangle. The color scheme can be export using the ***Program->Export->Colors*** menu.

Quadrant selections (I, II, III, IV) – in which quadrant of xz plane the respective types are visible.Using this functionality different cuts can be created. Please note, that in order to create a cut, the model has to be positioned in the origin.

Scene



Collision detection – enables collision detection

Show instance bounding spheres – renders bounding spheres of elements

Show triangle bounding box – debug purpose

Show normals – renderes normal vectors of the elements in the scene

Ignore depth – the helper structures are rendered top most

Show skeletons – renders helper geometry of skeletons

Show collision structure – debug purpose

Show closest point in skeleton – debug purpose

Render points – enables rendering of points and what size

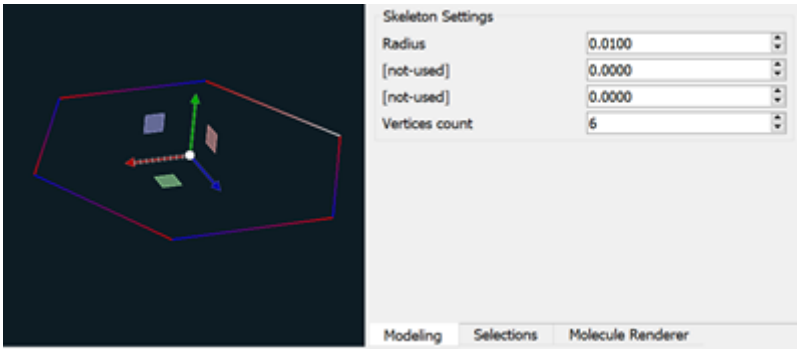
Render lines – enables rendering of lines and what size

Render triangle – enables rendering of triangles

FBOs – visualizing frame buffer objects and their attachments – debug purpose

Environment background – sets the color of the background

Skeletons



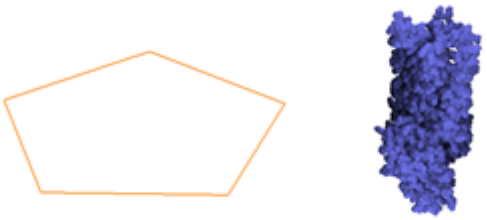
Skeletons are of the main part when modeling a complex structure. These proxy geometries can be added into the scene via ***Elements Library***. Once the skeleton is selected it's parameters can be adjusted in ***Modeling*** tab page.

Examples of rules

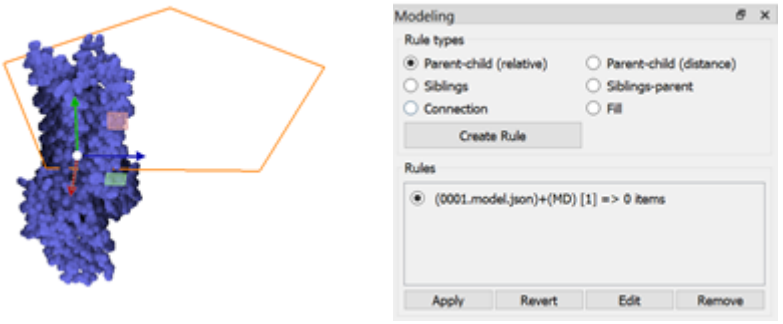
In this chapter several examples are presented. These examples are to present the capabilities of the system. It is important to understand that these rules can be combined (where it makes sense) as this forms the strength of the system.

Parent-child relative rule

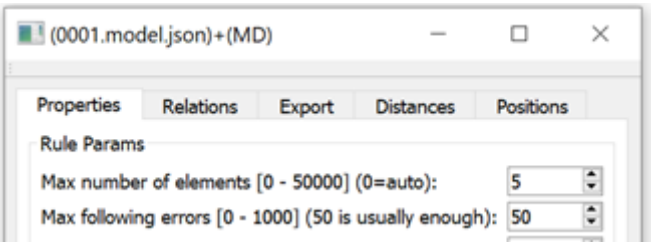
This rule is used the bind an element to a proxy geometry – skeleton. Firstly, a skeleton and an instance need to be inserted into the scene. In the following example, a pentagonal skeleton and a protein instance (md.pdb) are presented.



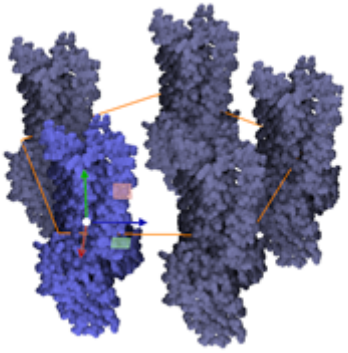
After selecting the protein instance by mouse a moving it to the desired position in the proximity of the skeleton, the rule can be created. To create ***Parent-child (relative)*** rule, select the respective option in the ***Modeling menu*** and click on the ***Create Rule*** button. The relative position of the protein instance to the closest skeleton vertex is stored. This position is in the later phase replicated for selected amount of vertices of the skeleton.



By default, the amount of elements when creating ***Parent-child (relative)*** rule is set to ***0***. To increase this amount go to the ***Edit dialog*** and change ***Max number of elements*** parameter.

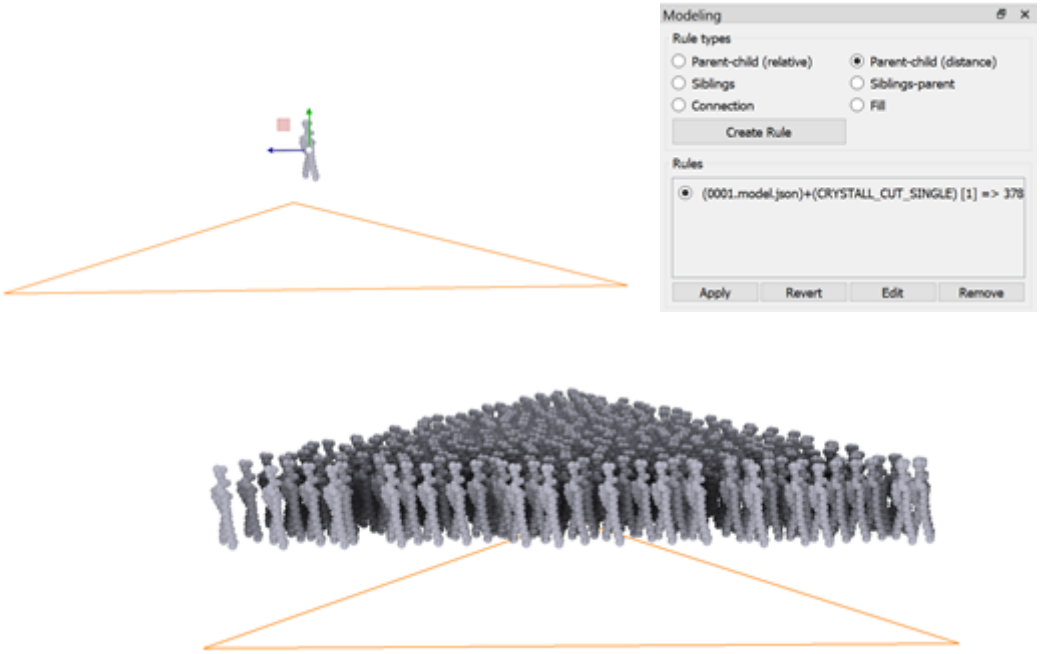


After applying the rule, the result should look like:



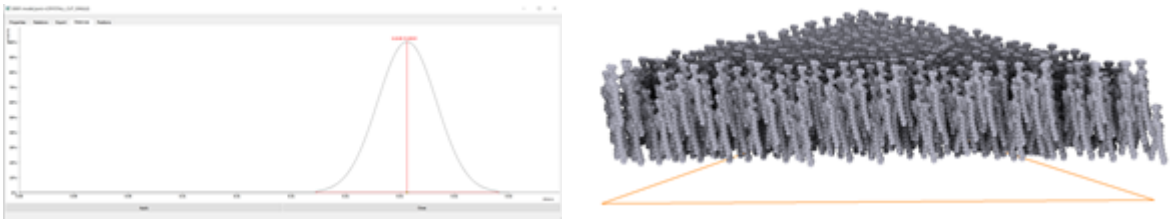
Parent-child distant rule

This rule is used for populating elements in the defined distance to the skeleton.



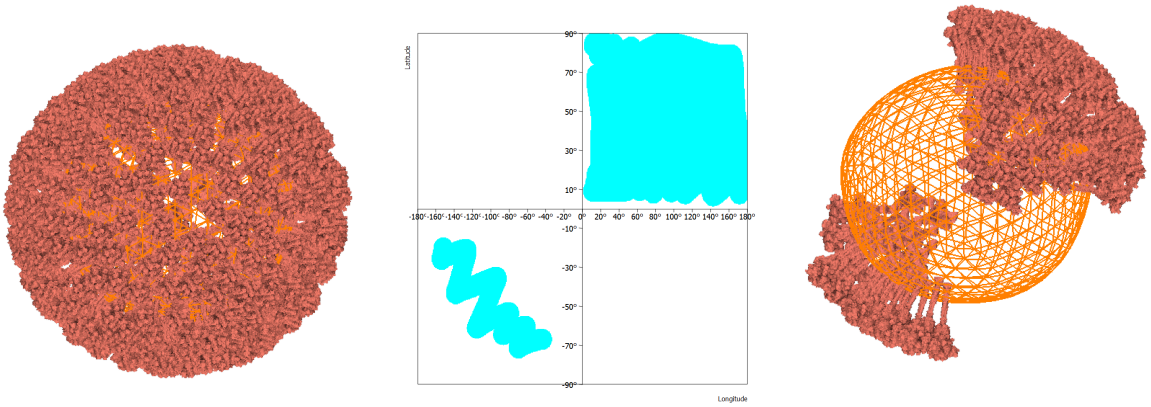
Distances

At the rule definition phase, the exact distance to the skeleton is stored. This distance can be customized (by adding a standard deviation) in the ***Edit Rule*** dialog in the ***Distance Tab***. By selecting the green point with (*mouse left button* and moving) in the ***Distance Tab***, the user can adjust the distance. Using the r*ight mouse button* and moving the user can adjust the standard deviation.



Angle distribution

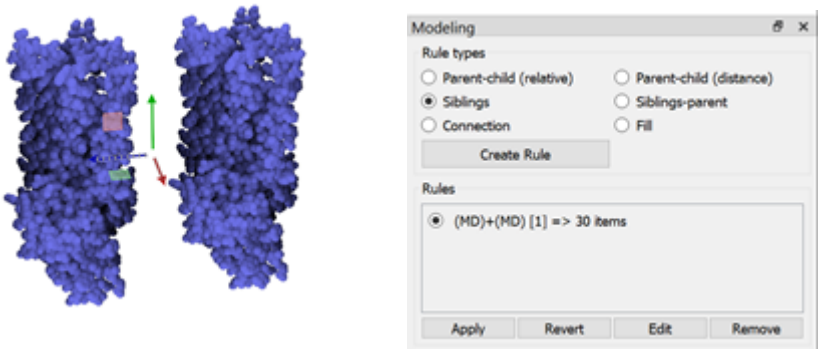
The user can specify (constrain) the areas where elements are being populated using the Angle distribution feature. In the ***Positions tabpage***, using the *left mouse button*, the user can draw in which regions the elements will be populated. The map represents spherical coordinates (longitude, latitude). Using the *right mouse button*, a portion of the painted path can be removed.



Notice: Angle distribution is implemented so far only for the Distance rule!

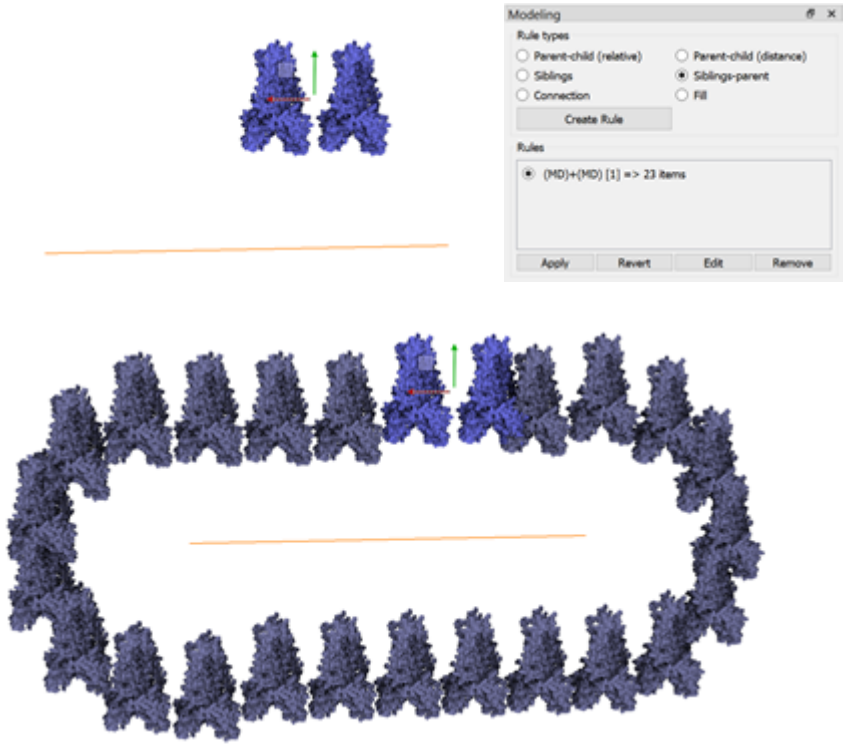
Siblings rule

Siblings rule is created to model the relationship between two elements. Rotation and translation are stored. The rule finds all relevant elements in the scene on which it can be applied. If some instances are selected, the rule is applied only on those.



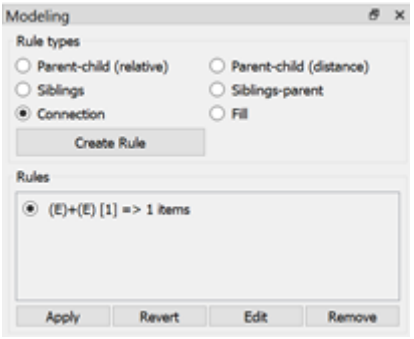
The example of application the very same rule on two elements. First, they are in a row. Second, they are placed randomly in the scene.

Siblings-parent rule



Connection rule

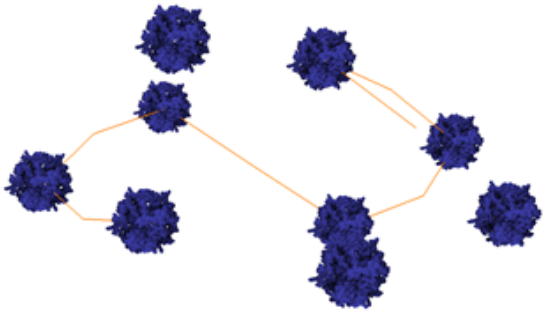
Connection rule is used to create a polyline. There are currently two modes of the connection rule.



Creating one curve

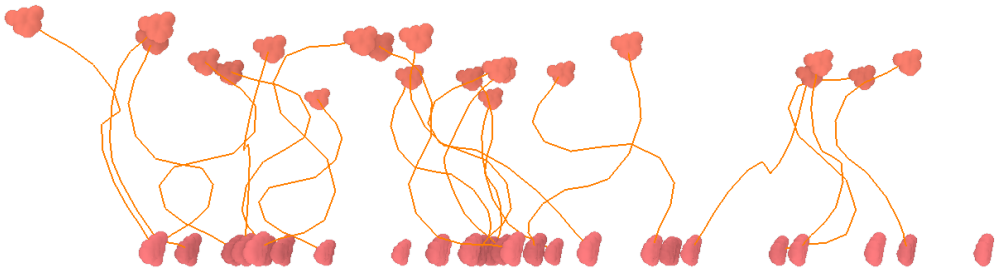
By selecting an element, the user specifies a type. Then, all elements in the scene of this type are selected. The resulting curve (polyline) connects these points.

Remark: the algorithm that searches for the connection does not necessarily uses all the points (< 10 can be not connected).



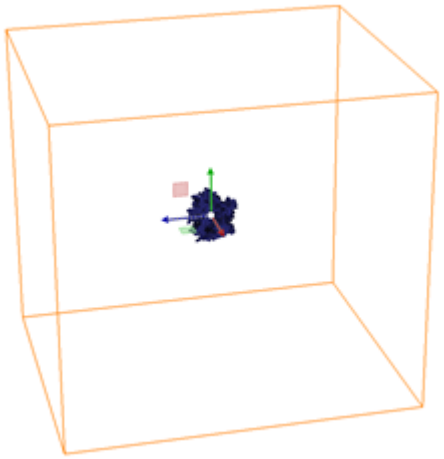
Creating set of connections

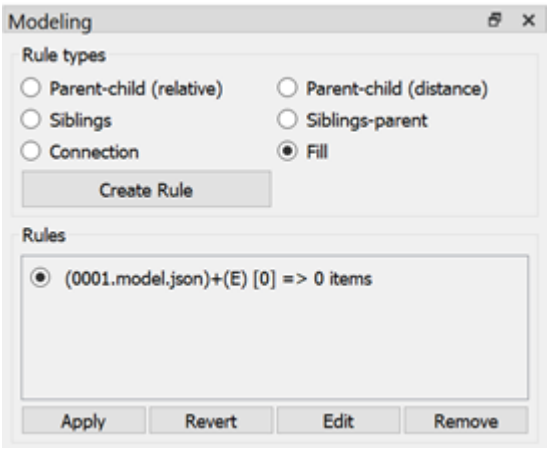
By selecting two elements, the user specifies two types. Then, pairs of these types are identified in the scene and connected. Every type is on the different end of the curve (polyline). The length (0 for default) and type (Line, PolyLine, Curve) of the curve can be specified.

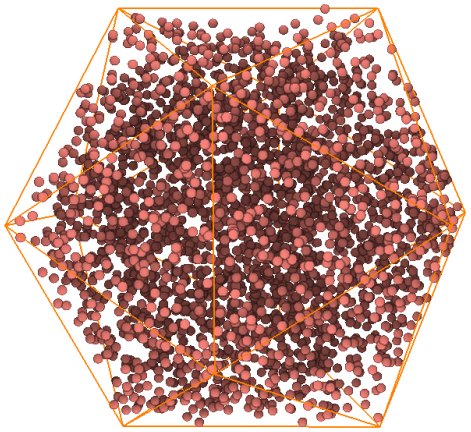


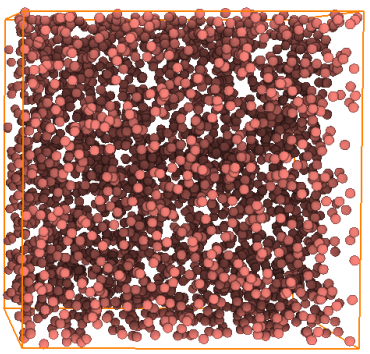
Fill rule

Using Fill rule elements can be populated inside the 3D skeleton (box, icosahedron, custom triangle mesh). The number of elements can be modified in the ***Edit Rule*** dialog (30 by default)



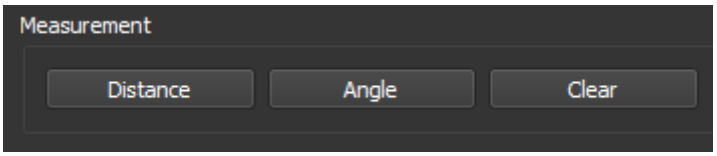


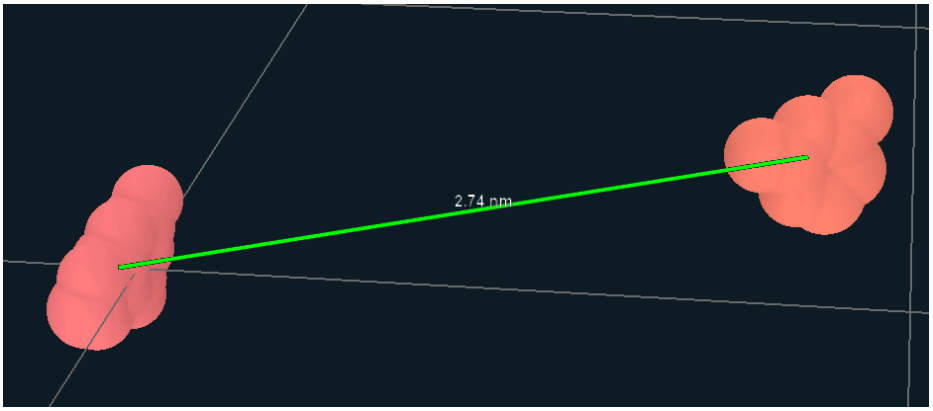


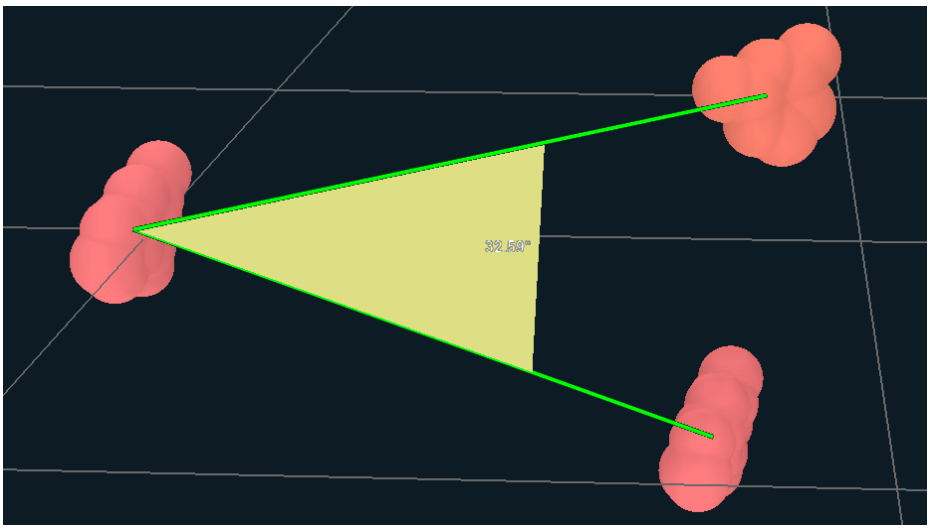


Measurement

The distance between two elements can be measured by selecting both of the elements and clicking the ***Distance*** button in ***Alignment->Measurement*** groupbox. Additional measuring line tool is added into the scene. The tool is updated every time a respective element is moved. To measure the angle, three elements need to be selected following hitting the ***Angle*** button. The all measurement tools can be removed by hitting the ***Clear*** button.







Troubleshooting

In this chapter several troubleshootings will be presented. For asking/suggesting of adding one, feel free to contact ondrej.strnad@kaust.edu.sa.

How to move MesoCraft data folder

When upgrading to a newer version of MesoCraft, it is wanted to move your "***work/data"*** folder as it might contain some custom data you have already changed/created. The following steps need to be performed:

- Copy "*work/data*" subfolder from the old version into the new version (the relative path has to be again "*work/data*")
- Delete file "*work/data/modeling/cache-files/ingredients-cache.bin*". This file will be recreated with the next MesoCraft session.

History revisions

10/24/2020 – [1.0.1] Initial version 1.0.1 of the build and this document.

11/02/2020 – [1.0.2] Examples of rules added, Fill rule added, minor bugs causing the application crash while creating rules fixed.

11/12/2020 – [1.0.3] Added helper identifiers (both to GUI and scene) for creating rules, scene scale revised, bugfixes.

15/06/2021 - [1.2.3] Connection rule re-implemented, several bugs leading to software crash fixed, rotations in rules unified

24/06/2021 - [1.2.4] Angle distribution feature added.

28/07/2021 - [1.3.0] Adding elements into the library.

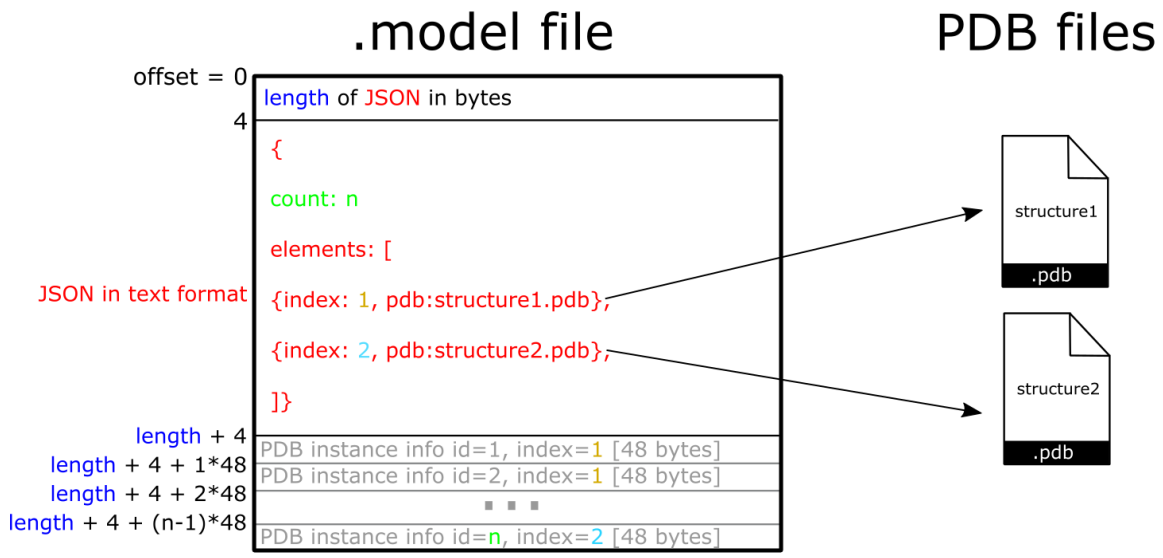
Known Issues

- A spike protein is shown while an asynchronous action is being executed. It disappears once the action is finished.
- Siblings-Parent rule does not work properly every time for Triangle and Rectangle skeletons

Model file format specification

One of the possible outputs of MesoCraft is .model file format. This file contains information about the current model and it's hierarchy. The description of the file format follows.

Model file format is a mixed JSON text header and binary data file. The first 4 bytes specify the length of the JSON information (metadata describing the model). Following these 4 bytes, JSON is stored, following by the binary data (12*4 bytes per instance). The instance binary data represents one instance of structure loaded from a PDB in the scene. PDB files are not part of the file and are expected to be parsed separately and centered to the origin before application of transformations from .model file. Overall file structure can be seen in the following schematic drawing.



JSON specification (v. 1)

JSON header contains information about the number of elements of the model, their referencing PDB etc. The description follows:

The root node contains the amount of element (count - parameter necessary for parsing the binary data), name of the model and further metadata about the elements.


```
{
    "boundingSphere":{ },
    "count":629,
    "elements":[ ],
    "file":"T4 Virus Version 3.1.model",
    "formatVersion":1
}
```

In the *elements* array, metadata regarding every type (not instance!) are included. There are two main types of elements - *protein* and *model*. In the case of *protein* type, there is an attribute *pdb* this type refers to.


```
{
   "baseRotation":{ },
   "boundingSphere":{ },
   "color":"#ff0000",
   "colorHCL":{ },
   "index":140,
   "label":"GP18",
   "name":"GP18",
   "pdb":"gp18.pdb",
   "type":"protein"
},
```

```
{
   "color":"#ff4c53",
   "colorHCL":{ },
   "index":109,
   "label":"0673.model",
   "name":"0673.model.json",
   "type":"model"
},
```


Binary instance info

Is formed by 48 bytes data chunk with the following layout:

[4 bytes][int] index
[4 bytes][float] position x
[4 bytes][float] position y
[4 bytes][float] position z
[4 bytes][float] rotation x
[4 bytes][float] rotation y
[4 bytes][float] rotation z
[4 bytes][float] rotation w
[4 bytes][int] id
[4 bytes][int] parent id
[4 bytes] <reserved 1>
[4 bytes] <reserved 2>

Parameters:

index - reference to the index field in the element node in the JSON metadata (can be understood as *type id*)

position - 3D vector

rotation - specified as a quaternion

id - unique identifier of the instance of the given type. Thus, proteins and models have both unique number sequence (i.e. there might exist a *protein* and a *model* with the similar id - the type needs to be taken into account while parsing the file)

parent id - id of a parent. Parent is always of type *model* (i.e. a parent cannot be a *protein*).