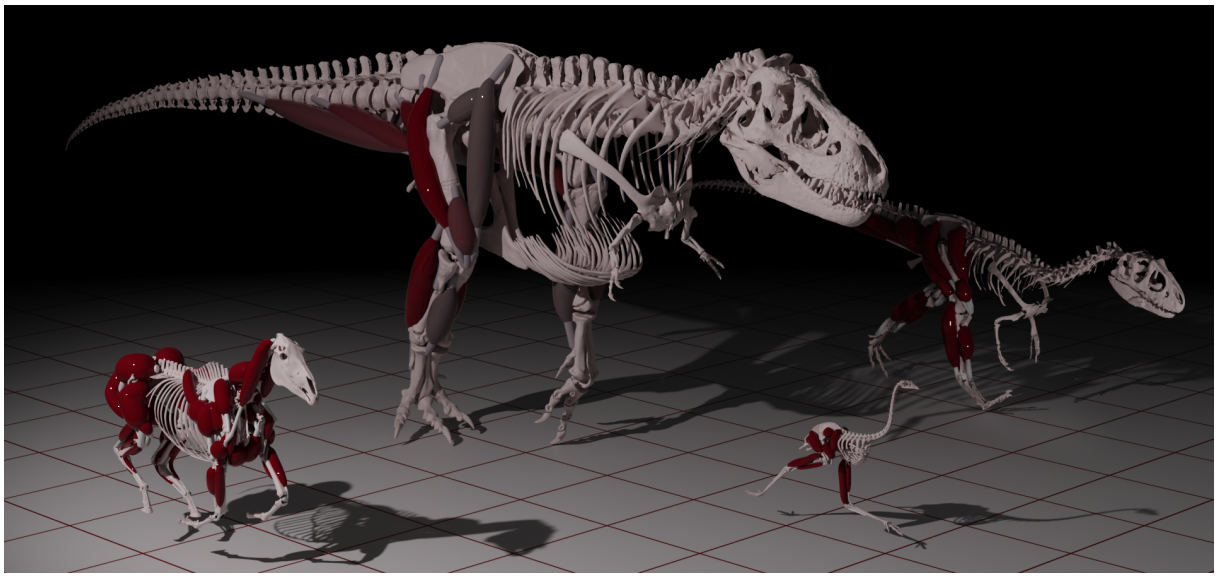
MuSkeMo Manual



Pasha van Bijlert

pashavanbijlert@gmail.com
Github Repository: Github
Bug Reports and Feature Requests: Submit an issue on Github

Version 0.8.8
November 28, 2024

Contents

1 Introduction

MuSkeMo is a tool for musculoskeletal modeling in Blender. MuSkeMo allows you to translate biological 3D scans to useful biomechanical information, in the form of full models or model components. These can then be used for further analysis within Blender, or exported for simulations in other biomechanical software. Simulation trajectories can be imported back into MuSkeMo to make publication-ready stills and videos, using Blender's built-in raytracing rendering. MuSkeMo can also be used for 3D landmarking and for simply calculating inertia tensors from 3D scans.

This MuSkeMo documentation file is meant to complement the video tutorial series on YouTube. This documentation is focused on features specific to MuSkeMo, and where necessary provides suggestions regarding Blender settings and features. Because Blender can be overwhelming at first, new users are recommended to follow the "Donut tutorial" series by BlenderGuru on Youtube. For Blender-specific questions, the reader is referred to the extensive Blender documentation. MuSkeMo currently officially supports Blender versions 3.0-4.1, but will offer 4.2+ support in the future.

2 Installation instructions

To download: Go to the releases page on the Github repository, and download the most recent version of "MuSkeMo.zip". The .zip file is used to install the plugin, but also contains a folder with utility functions (e.g., for OpenSim conversion).

To install (Fig. 1): In Blender v3-v4.1, go to Edit → preferences → Add-ons and then click "Install...", and then select your downloaded zip file. Then type in "MuSkeMo" in the search bar of the addon window, and enable the plugin by pressing the check mark. If using Blender 4.2+, you may need to use the legacy installer, but MuSkeMo has currently not been tested above v4.1.

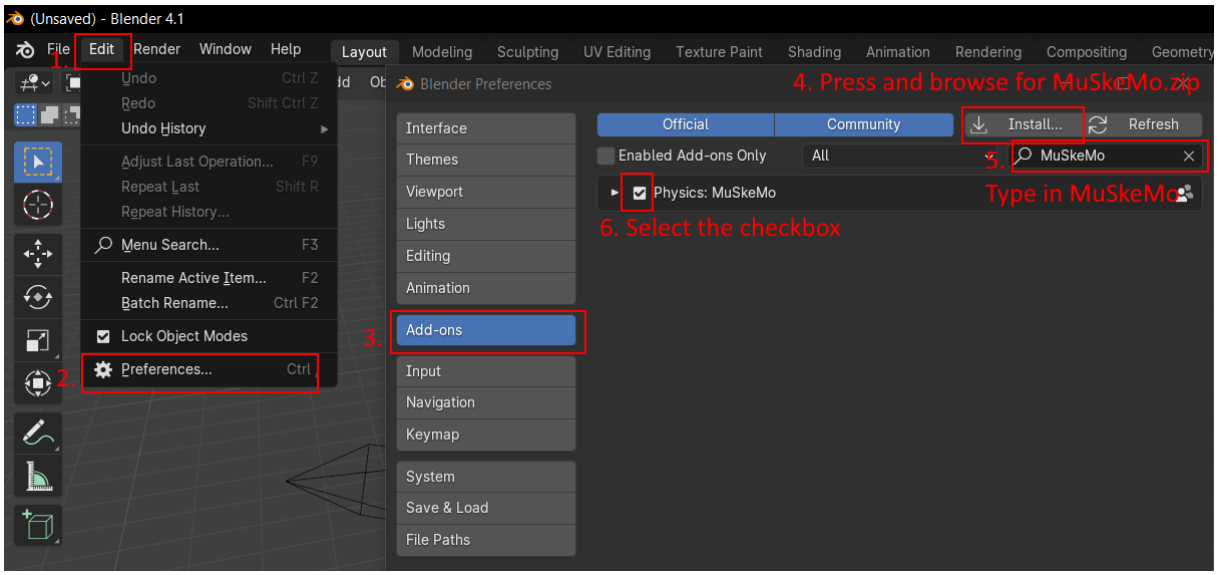


Figure 1: Installing MuSkeMo in Blender versions 3.0-4.1

3 Units

MuSkeMo uses the following units:

- **Spatial distance/position:** metres (m)
- **Mass:** kilogram (kg)
- **Moment of inertia:** kilogram metre squared ($\mathrm{kg\,m^2}$)
- **Force:** Newton (N)

- **Angle:** Degrees (°, only for pennation angle). Internally MuSkeMo uses radians during computations.

WARNING: Blender's default units are in metres. Many of the calculations in MuSkeMo are derived from the spatial position of individual points (vertices) of the 3D meshes. If you change the base units of Blender, these calculations are also implicitly scaled, and the units will no longer be correct (e.g., if you change Blender's base units to centimetres, linear dimensions will be off by a factor of 100, but volumetrics and inertial properties will be off by a factor of 100^3). MuSkeMo currently only officially supports metres as the base unit, correct scaling of the outputs if using other units is up to the user.

4 A Note on Precision

MuSkeMo is built using the Blender Python API, which internally uses double precision numbers (64-bit floating point, $\sim$ 16 significant digits). However, Blender itself stores numbers using single precision (32-bit floating point, $\sim$ 8 significant digits), including all 3D point and mesh data. This means that precision beyond 8 significant digits cannot be expected when using MuSkeMo. Single precision floats nevertheless have a very large range of numbers that can be represented—the smallest number that can be accurately reported is in the order of 1×10^{-38}, approximately 1 million times smaller than the mass of an electron in kilograms.

5 Orientations

Orientations are represented as XYZ body-fixed Euler angles (see Appendix A) and as unit quaternions (see Appendix B). Internally, MuSkeMo avoids Euler angles where possible because they are prone to gimbal lock, but popular simulators often use Euler angles by default (e.g., OpenSim, SCONE).

6 Using MuSkeMo

MuSkeMo lives in Blender panels (Fig. 2A). Panels can be interactively resized and collapsed, which can be useful depending on your screen size. Although Blender does support using a trackpad, it is substantially more intuitive to navigate with a mouse, and a keyboard with a separate numberpad. There are also certain settings that make navigation easier (see Section 6.3).

WARNING: Ensure that 3D meshes (visual geometry and tissue outlines) have all their transformations applied before you start constructing the model. To do this, select the object, press `Control + A`, and select "All transforms". This accounts for the way that Blender stores local transformations. When you move, rotate, and/or scale 3D models in Blender, these transformations are initially stored only locally in the object's data (an extra storage layer in Blender). This means that the object's position has not changed in 3D space, you have defined an extra local transformation that can easily be undone. Some of the functions in MuSkeMo may behave unpredictably unless transformations have first been applied.

6.1 Where MuSkeMo stores model data

All the model components and other user-created objects are stored in Blender Collections (Fig. 2B), which are essentially just folders. The collection names all have sensible defaults in MuSkeMo (e.g., "Bodies" for bodies), but can be changed if desired. The collection names are also used as the default filenames during import/export, but this is also customizable. By default, child objects aren't always visible in the collection that they're actually in (see Section 6.3).

For each model component, data created by MuSkeMo are stored in Blender as Custom Properties assigned to each object in question (Fig. 2C, see also Section 7 on MuSkeMo Data Types).

6.2 Blender Auto Save

Blender is quite stable, but it is possible for it to crash. Fortunately, by default, Blender has an Auto Save function. After a crash, restart Blender, and go to File, Recover, Auto Save. Remember to save the recovered file.

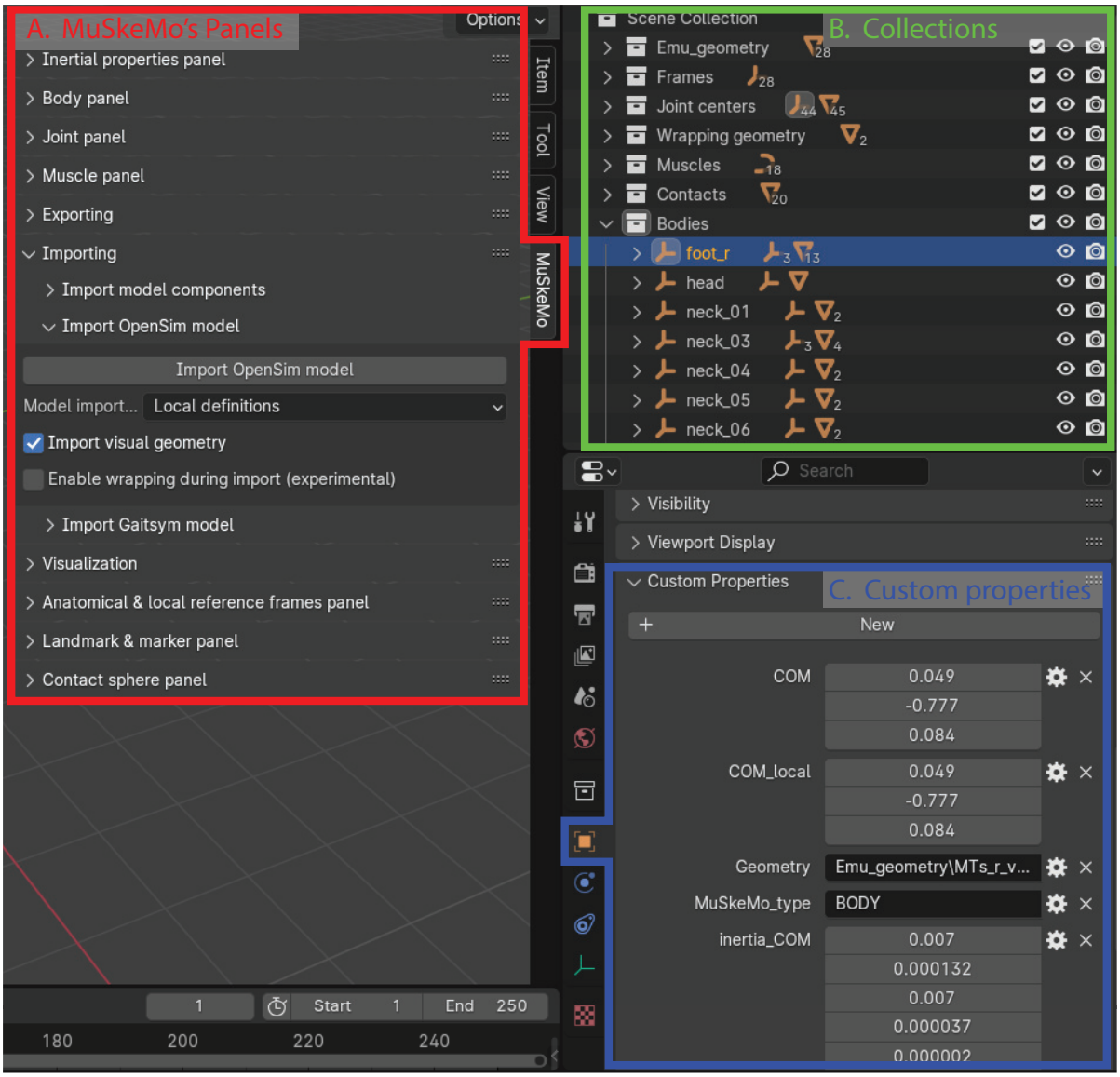


Figure 2: A. All of MuskeMo's functionality can be accessed via these panels. The panels can be interactively resized and collapsed. B. After creating or importing model components, they are stored in Blender collections. If child objects are not visible, turn off object children in the outliner filter (see Section 6.3 C. A component's data are stored as custom properties. Press the yellow square, scroll all the way down, and open the Custom Properties dropdown

6.3 MuSkeMo Global Settings Panel

The Blender user interface has several default settings that do not work well with MuSkeMo. By default, Blender forces the Z-axis as being the "up" direction in the viewport. The International Society of Biomechanics assumes Y as the up direction. To achieve this, the user should set the view rotation setting to "Trackball", and use "orbit around selection". The "MuSkeMo Global Settings" panel provides a button that does this automatically: "Set recommended Blender settings". This button also toggles off "Object Children" in the outliner. By default, objects are placed under their parents in the outliner, but this would for instance result in a body being placed under the parent joint in the "Joint collection", instead of in the "Body collection". Turning off "Object children" avoids this behavior. However, this is a project file setting, so this needs to be reset with each new project, unlike the navigation settings.

6.4 Inertial properties panel

The main goal of this panel is to compute inertial properties from 3D volumetric meshes, eg. from CT-segmentations or surface scans. Inertial properties are **not dynamic**, if you move the 3D meshes or change their densities, you must recompute their inertial properties, otherwise COM, mass, and or inertia can be outdated. To compute the volumetric inertia tensor (with elements m^5) of a triangular mesh, MuSkeMo implements the solution derived and presented in [3]. This gives an exact solution for the volumetric moments of inertia of a closed, triangulated mehs, based on the Divergence Theorem. The volumetric tensor is multiplied by density to acquire the inertial tensor elements. This algorithm requires the mesh to be triangulated and watertight to provide meaningful results, and MuSkeMo alerts the user if this is not the case. It is possible to change the density property of an object, after which you'll have to select the object and rerun "Compute for selected meshes".

6.4.1 Generate minimal convex hulls

Several studies have used convex hulls as the starting point for estimating inertial properties of (extinct) animals directly from full skeletons In this subpanel, you can automatically generate convex hulls around (skeletal) meshes in a collection, and then apply methods from the literature to reconstruct the inertial properties.

You have to designate the "Skeletal mesh collection" where your skeletal meshes are located. This defaults to the "Geometry" collection, but it can also be a separate collection. Each separate object in the collection will receive its own convex hull, so the meshes in the collection should represent functional body segments. See [8, 1, 7] for examples. Convex hulls are placed in a new collection.

6.4.2 Expand convex hulls

Both the expansion panels (arithmetic and logarithmic) work with segment names and corresponding scale factors or logarithmic parameters. If only "whole_body" is typed in as Segment name 1 (with no other segment names defined), all the objects in the target collection will be treated as one - in arithmetic mode, all objects are scaled by a single factor, in logarithmic mode, all object volumes are summed before determining the whole-body scale factor.

If segment 1 is not "whole_body", or if you have more than one segment name defined in the panel, MuSkeMo tries to match whatever segment names you have typed into the panel to the names of the objects in your target collection. E.g., if you've typed "neck" as one of the segment names in the panel, with a corresponding scale factor, all the objects in Convex hull collection that have the case-sensitive word "neck" in their name will be expanded by the scale factor (e.g., "neck_1", "neck_prox.obj", but not "Neck_1" and also not "torso"). The "Expansion Template" dropdown menu gives several presets from the literature, and you can customize them if desired.

MuSkeMo assumes you want bilaterally symmetric models, and therefore when using the expansion functions, MuSkeMo generates bilaterally symmetric expanded hulls for the following segments: "head", "neck", "torso", "tail". Furthermore, MuSkeMo also applies directional scaling. The following segments are scaled about the Y and Z axes (and thus not the X axis): "head", "neck", "torso", "tail", "forearm", "hand", "toe". All other segments are scaled about the X and Z axes. The resultant scaled shapes may not be very biologically realistic. You could add a "Maintain volume" constraint and change the shape in Blender (this was a suggestion by Matt Dempsey). Apply the constraint after you are satisfied.

Under **Expand convex hulls - arithmetic**, you can use arithmetic expansion factors (i.e., linear scale factors) to scale your hulls. If a segment initially has a volume of 1 m^3 and a scale factor of 1.2, the resultant segment will have a volume of 1.2 m^3.

- **Custom:** Type in the segment names, and the desired scale factor.
- **Macaulay 2023 Bird:** The per-segment "Bird" average expansions from Macaulay et al. 2023 [7], supplementary data S6.
- **Macaulay 2023 Non-Avian Sauropsid:** The per-segment "Croc_Lizard" average expansions from Macaulay et al. 2023 [7], supplementary data S6.
- **Macaulay 2023 Average (Bird and Non-Avian Sauropsid):** The per-segment "Av." average expansions from Macaulay et al. 2023 [7], supplementary data S6.

- **Sellers 2012 Large Mammals:** This is a "whole_body" scale factor of 1.206 as described in [8]

Under **Expand convex hulls - logarithmic**, it will be possible to use log-transformed regression equations from the literature to scale your hulls. This functionality is currently disabled.

6.5 Body panel

Define rigid bodies, assign precomputed inertial properties, or compute directly by selecting one or several volumetric meshes. Inertial properties are **not dynamic**, if you move the source objects that the rigid bodies were based on, or change their densities, you must recompute all inertial properties of the body. Otherwise COM, mass, and or inertia can be outdated.

In this panel, you can also attach visualization geometry (eg., bone meshes) to bodies.

6.6 Joint panel

Define joints, and assign (and remove) parent and child bodies. If you want to change a joint's position or orientation, detach the parent and child bodies first. If a parent or child body has an anatomical (local) reference frame assigned, MuSkeMo automatically computes the relative positions and orientations in these frames as well. Orientations are stored as body-fixed, XYZ-Euler angles and as quaternions. All data that are created are included during export (if local frames are not assigned, these values will be nan).

It is also possible to define coordinate names in the joint panel. After exporting from MuSkeMo, the model conversion scripts (e.g., MuSkeMo_to_OpenSim) will only add DOFs to model if they are named (e.g. hip_angle_r). If no coordinates are named for a joint, the joint is turned into an immobilized joint (e.g., WeldJoint in OpenSim).

In the joint panel (under joint utilities), you can also mirror right side joints, fit geometric primitives (sphere, cylinder, ellipsoid, plane), and match the transformations of a joint to the fitted geometry. By default, MuSkeMo ensures that child objects and parent objects are not transformed with the joint. Instead, only the joint's position or orientation is changed, and related data (e.g., pos_in_child) are recomputed automatically.

6.7 Muscle panel

Define path-point muscles. Muscle points are added to the 3D cursor location, and parented to the selected Body (so you have to define bodies before creating muscles). Muscles points are added to whatever muscle name is currently active in the panel, and muscles are added to the user-designated "Muscles" collection.

6.7.1 Adding muscle points

To add a point to the end of the muscle (or to create a new muscle), type in the muscle name, and select the target body. Press shift + right mouse button to position the 3D cursor at the desired location in 3D space, and then press "Add muscle point". Muscle points are added to the 3D cursor location. You can change the locations of the path points by selecting the muscle in edit mode (select the muscle and press "TAB", then select the relevant point and press "G" to move it around). If the parent bodies are repositioned, you must delete and redraw the muscles.

6.7.2 Adding muscle points

It is possible to insert muscle points (instead of adding them at the end). To insert a point, select the point index after which you would like to insert a point (starting at 1), and press "Insert muscle point".

6.7.3 Visualization radius

By default, muscles are visualized using a Geometry node modifier named "musc_name_SimpleMuscleViz" (see section 7.3.1). If this visualization option is used, you can change the visualization radius of all muscles simultaneously by using the "Update visualization radius" button after selecting a desired radius. Alternatively, you can manually tune individual radii of muscles by changing the input in the "SimpleMuscleViz" modifier.

If the volumetric visualization option is used for the muscles, visualization parameters can be tuned in the "VolumetricMuscleViz" modifier (see section 7.3.2 for details).

6.7.4 Reflect unilateral muscles

This feature reflects left- or right-sided muscles if currently only one of the two sides exists in the "Muscles" collection. You can choose what plane across which to reflect (default = 'XY'), and change the strings with which you designate the left and right sides (default is '_l' and '_r', respectively). The script then checks for all muscles whether its other-side counterpart exists, and if not, creates it. **Warning:** This function uses a string search to check whether the side string is in the object's name. This can potentially cause conflicts (e.g., the name "dorsal_rib_l" contains both '_l' and '_r').

6.8 Anatomical & local reference frames panel

Construct anatomical and local reference frames, by assigning landmarks or markers as the reference points to construct the axes directions. Local reference frames have a position and an orientation with respect to the global reference frame. If the global frame is denoted with the letter $\mathcal{G}$, then the position of an arbitrary frame in the global reference frame can be written as the following vector: $\vec{v}_{\mathcal{G}}$. Internally, orientations are stored through rotation matrices. The rotation matrix that rotates a vector from the local $\mathcal{B}$-frame to the global $\mathcal{G}$-frame can be written as: ${}^{\mathcal{G}}\mathbf{R}_{\mathcal{B}}$.

Orientations are exported as rotation (unit) quaternions (w, x, y, z), and also as body-fixed (intrinsic, active) XYZ-Euler angles (phi_x, phi_y, phi_z, in rad) (see Appendices A & B). The latter is prone to gimbal lock.

If anatomical / local frames are assigned to a body, MuSkeMo also computes inertial properties, joint positions and orientations, contact positions, and muscle path points with respect to these frames. This requires the transpose of matrix ${}^{\mathcal{G}}\mathbf{R}_{\mathcal{B}}$, namely: ${}^{\mathcal{B}}\mathbf{R}_{\mathcal{G}}$. This rotates a vector from the global $\mathcal{G}$-frame to the local $\mathcal{B}$-frame.

For an arbitrary point p expressed in $\mathcal{G}$, MuSkeMo computes the transformation to $\mathcal{B}$ as follows:

$$\vec{p}_{\mathcal{B}} = {}^{\mathcal{B}}\mathbf{R}_{\mathcal{G}} \, (\vec{p}_{\mathcal{G}} - \vec{v}_{\mathcal{G}}) \tag{1}$$

For an arbitrary matrix $\mathbf{I}$ expressed in $\mathcal{G}$ (e.g., an inertial tensor with respect to the body COM), MuSkeMo computes the transformation to $\mathcal{B}$ as follows [10]:

$$\mathbf{I}_{\mathcal{B}} = {}^{\mathcal{B}}\mathbf{R}_{\mathcal{G}} \, \mathbf{I}_{\mathcal{G}} \, {}^{\mathcal{G}}\mathbf{R}_{\mathcal{B}} \tag{2}$$

Readers familiar with linear algebra will recognize this as a similarity transformation, that defines a change of basis.

6.9 Landmark & marker panel

Similar to muscle points, landmarks are added to the 3D cursor location.

6.10 Contact panel

Similar to muscle points, contacts are added to the 3D cursor location. Contacts can also be assigned a parent body.

6.11 Export panel

You can export all the user-created datatypes via this panel. The individual exporters export all the data types from the user-designated collections (folders) in Blender. It is possible to export all the visual geometry to a subfolder.

MuSkeMo exports all the data with respect to both the global reference frame (origin), and body-fixed local reference frames. Orientations are exported as XYZ-Euler angle decompositions, and as quaternion decompositions.

Under export options, it is possible to configure other text-based filetypes for export (e.g., txt, bat), configure custom delimiters, and choose the number formatting in the exported files.

6.12 Import panel

You can currently import bodies, joints, muscles, frames, and contacts, if they are MuSkeMo-created CSV files.

6.12.1 OpenSim model import

MuSkeMo provides an OpenSim importer. This can import most of the components of an OpenSim model. Although the default behavior is to import models using local definitions, it is also possible to import models using global definitions (i.e., all the component positions are defined as their position in the global frame, and for joints specifically, transformations in parent and child are both set to the global position and orientation). Global definition import is mostly useful for models that were created using MuSkeMo's conversion script

6.12.2 Gaitsym 2019 model import

MuSkeMo includes a Gaitsym (2019) importer. It currently imports bodies, joints, muscles (DampedSpring elements are treated as muscles), contacts, and markers (as frames). Muscles that include wrapping are currently not supported, but limited wrapping support is planned in a future update. Visual geometry can be imported, but requires the user to type the name of the containing folder in "Gaitsym geometry folder". The geometries must be in a subdirectory of the model directory, and the name of this subdirectory must currently be manually typed into the panel.

It is possible to automatically apply a rotation to the entire model during import. This can be convenient because Gaitsym is generally geared towards the Z-axis being the "up" axis, whereas ISB recommends Y-up. To rotate a model from Z-up to Y-up, apply a -90 °rotation about the x-axis. Points simply get rotated, MOI gets transformed according to eq. 2 (although technically the transformation is now from one global frame to another). The same change-of-basis transformation is also applied to orientions (of joints, frames, etc.). The result is that the old Z-axis becomes the new Y-axis, and the old -Y-axis becomes the new Z-axis.

6.12.3 Other simulators

Future updates will include Hyfydy and MuJoCo model support.

6.13 Visualization panel

Rendering in Blender can be a complicated process. It is capable of professional level video graphics rendering, and there are a lot of settings that the user can modify to achieve this. This panel provides some ease of use functions that preset the rendering settings that the author finds visually appealing, while also providing adequate performance. There exist thousands of video tutorials for creating renders in Blender. Until a MuSkeMo-specific rendering tutorial is recorded, it is recommended that you follow one of the many out there (e.g., the Donut tutorial referenced at the top of this document). **If your computer does not have a powerful graphics card (GPU), it may be necessary to tweak the recommended settings.**

6.13.1 Trajectory import

MuSkeMo enables you to import simulated trajectories back into Blender to create high-quality animations with complex camera movements. Currently, it is only possible to import trajectories using OpenSim .sto files. If you are importing a periodic stride, it is possible to automatically loop these in sequence to create a video using multiple strides, while progressing the (selectable) forward translation coordinate. In this case, you have to define the root joint (default = groundPelvis) and the forward progression coordinate (default = coordinate_Tx).

6.13.2 Visualization options

This subpanel includes several convenience tools to aid users who are new to animations in Blender. These are:

- **Convert to volumetric muscles:** adds volumetric muscle visualization to each muscle in the scene. See section 7.3.2 for details.
- **Create a ground plane:** adds a ground plane to the scene
- **Set recommended render settings:** sets the rendering engine to Cycles (see below), the device to GPU, Max samples to 1000, turns on persistent data (under performance), and sets the "Look" to very high contrast (under Color Management). **WARNING:** the Cycles rendering engine is a path tracing (raytracing) rendering engine. If you do not have access to a computer with a powerful GPU, rendering performance will be incredibly poor. In that case, you should switch to the Eevee rendering engine.
- **Set black background gradient for renders:** creates a node setup in the compositor that de-emphasizes the background.

6.13.3 Default colors

In Blender, objects get assigned a color (and other surface rendering properties, such as roughness) by assigning a material. Before importing and/or model component creation, you can define the desired default colors in this panel for muscles, visual geometry (bones), joints, contacts, markers, geometric primitives, and wrapping geometry. If an instance of the object has been already created (e.g., you have already created a joint), you can change the colors by changing the object's material in the properties tab or in the shader editor. Unlike all other object types, each individual muscle in the model gets a unique material. This is to ensure that its activation can be independently animated when importing trajectories. This means that if you want a different default muscle color, you need to define it before model import/creation, or you must change all the muscle-materials individually.

Joints, contacts, wrap geometry, geometric primitives, and markers are provided with a transparent material by default (this is achieved with a mix shader node in the shader editor).

6.14 MuSkeMo utilities

The MuSkeMo.zip release contains a folder named MuSkeMo utlities, which includes several useful functions and scripts.

6.14.1 OpenSim conversion

The Matlab script "MuSkeMo_to_OpenSim.m" converts your MuSkeMo outputs to an OpenSim model. You must have the OpenSim Matlab API installed, and "CreateOpenSimModelFunc.m" must be in the same directory. The script provides a graphical user interface that lets you select MuSkeMo-created csv files of the model components. It is possible to create a model using global model definitions (optional local frames are ignored, and position/orientation in parent and child are both set to the global position and orientation). It also possible to construct a model using local model definitions. This requires the user to assign a `local_frame` to each body.

6.14.2 Fitting muscle lines of action

It can be useful to fit a curve to a 3D mesh of a muscle, for instance, acquired via DICE-CT scanning or surface scanning. "MuscleLineOfActionFitter.py" is a rudimentary fitting tool. It is not currently built into MuSkeMo yet, but the script can be opened in a Blender scene via the script editor. The user must fill in the target muscle name in the script, and ensure that two objects named 'origin' and 'insertion' are present in the scene. You can set the desired resolution.

The fitter works by slicing up the mesh into n-sections, whose heights are determined by the user-input resolution, and the origin-insertion distance. The slices are created by performing a boolean intersection between the target mesh, and a cuboid that progresses in n-steps from the origin position to the insertion position, aligned in this direction. The volumetric centroid is computed for each slice, and these centroids combined with the origin and insertion form the fitted curve.

High resolution results in slices with a smaller height and thus a smoother curve, with very high resolutions approximating thin cross-sections instead of volumetric slices. High resolutions can potentially result in clipping (ignoring) bits of the muscle during the fitting procedure: if due to the shape, sections of the muscle are proximal to the origin, or distal to the insertion, they are not included in the boolean intersection, and their volumes thus not represented in the line of action. It is up to the user to account for this, and this is why the default behavior is to display both the slice-cuboids, and the resultant slices, which can be compared to the input muscle.

Lower resolution gives a less smooth result, but in most cases will include the entire muscle during its computation.

The resultant objects are Blender curves. It is possible to resample these, if desired. By selecting a curve and right-clicking, it is possible to convert the curve to a mesh. This makes it possible to snap to the fitted curve when creating a MuSkeMo muscle based on the fitted curve.

6.14.3 Compute closest point between objects

The Python script "ComputeClosestPointBetweenMeshes.py" can be run in the Blender script editor. It outputs the closest point between two meshes (in meters). This can be useful to compute minimal joint spacing while articulating a skeleton, or for instance in a series of XROMM frames. An example of how to extend it to work with multiple objects, loop through frames, and export distances as a CSV can be seen in "ComputeClosestPointRealExample.py". This script was written for Voeten et al. (in prep). If that paper has come out by when using this script, please consider citing it as the source.

6.14.4 MuSkeMo version updater

The folder also contains an updater (Python) script to update older MuSkeMo scenes to v0.6.3 and up. To run this, open the python script in the Blender script editor and run it. Back up your work first. This script will be removed in a future update.

7 MuSkeMo Data Types

7.1 BODY

A rigid body. Rigid bodies have inertial properties, which can be computed during model creation from 3D scans:

- `mass` (kg)
- `COM` (center of mass (m) in the global reference frame)
- `inertia_COM` (moment of Inertia (kg m^2) about the COM, in the global reference frame)

Bodies can have one or more optional attached `Geometry` meshes for visualization (e.g., 3D meshes of bones). These are delimited with a '`;`' if present, and usually preceded with the name of the subdirectory (default = '`Geometry`') in which the meshes will be exported. For example:

'`Geometry/cranium.obj;Geometry/mandible.obj;`'

COM is always reported in the global reference frame, and MOI is always computed with respect to the COM in the global reference frame and orientation. It is possible to assign a `local_frame` to a body (see section 7.7). This automatically computes the following properties:

- `COM_local` (center of mass (m) in the body's local reference frame)
- `inertia_COM_local` (moment of Inertia (kg m^2) about the COM, in the body's local reference frame)

The underlying object type in Blender is an "Empty".

7.2 GEOMETRY

Visual geometry (e.g., bone meshes, or tissue outline meshes) can be attached to bodies for visualization purposes. In most simulators, these don't have a physical function, but are purely used for visualization

purposes and for determining where muscle attachments are relative to the bodies.

WARNING: OpenSim has a visualization bug that prevents models from loading correctly if the total file size of the attached geometry exceeds 50MB. You can decimate the meshes to reduce them to about 50MB or attempt to load the model into OpenSim Creator.

If Decimating the model doesn't work, the visualization error can sometimes also be triggered by meshes that are composites of several different parts, or that have many loose triangles. Using MuSkeMo, first detach the visual geometry via the Geometry panel. Then, select the mesh and press TAB for edit mode, and then press P (for seParate). This separates the model by loose parts. The resulting meshes should be parented to the body individually. This will require you to re-export the geometries, re-export the bodies CSV, and regenerate the OpenSim model. However, it is usually the total filesize that triggers this issue, not composite meshes.

7.3 MUSCLE

A path-point muscle. Each muscle point is automatically parented to a body. Muscles have the following user-definable contractile properties:

- `F_max` (maximal contractile force of the muscle fibers, in Newtons)
- `optimal_fiber_length` (in meters)
- `pennation_angle` (in degrees)
- `tendon_slack_length` (in meters)

The underlying object type in Blender is a poly-curve. Each point in the curve is attached to a body using a hook-modifier.

7.3.1 Simple muscle visualization

Simple muscle visualizations are achieved by adding a simple Geometry node setup to each muscle (curve in Blender). This node setup essentially lofts a curve with the user-specified radius (default = 0.015 m) across the entire length of the curve. The radius can be changed without going into the Geometry nodes setup, by selecting the correct modifier in the modifier stack.

Under "Geometry nodes", with the "SimpleMuscleViz" node of a muscle selected, the node setup is visible. The visualization setup itself is specified by the "SimpleMuscleNode" node group (select it and press 'TAB' to modify, this node group is shared by all muscles and thus modifications are applied to all muscles). The node setup also applies a material to each individual muscle, so that their colors can be animated invidually.

7.3.2 Volumetric muscle visualization

Similar to the simple muscle visualizations, volumetric muscle visualizations are also achieved by adding a Geometry node to each muscle. The node-group itself is shared across muscles, but muscle visualizations are individualized through four parameters, which can be accessed directly in the "VolumetricMuscleViz" modifier. These are:

- MuscleVolume: Computed using each muscle's F_max, optimal_fiber_length, and specific_tension (set in the Visualization panel, see section 6.13). The node setup adjusts the muscle's radius to keep the volume constant irrespective of the length.
- MuscleTendonLengthRatio: This decides the ratio of the muscle belly to the total musculotendon complex length. If set to 1, the muscle belly is stretched across the entire muscle's length.
- TendonMuscleRadiusRatio: This sets the relative ratio of the tendon with respect to the muscle's radius. Set to 0 if you want no tendon visualization.
- ProxToDistMuscleBellyBias: By default, the muscle belly is in the middle of the curve. If MuscleTendonLengthRatio is less than 1, you can shift the muscle belly more proximally or more distally using "ProxToDistMuscleBellyBias".

7.4 JOINT

A joint in MuSkeMo represents the connection between two rigid bodies, allowing them to articulate relative to each other. The joint position and orientation can be expressed in XYZ Euler angles or quaternions.

`parent_body` and `child_body`: these are the two bodies connected by the joint. These must be defined by the user. When defined by the user, `pos_in_global` and `or_in_global` are both computed.

- `pos_in_global`: The position of the joint center in the global reference system (in meters).
- `or_in_global_XYZeuler`: The orientation of the joint in the global reference system (in radians) using an XYZ Euler decomposition.
- `or_in_global_quat`: The orientation of the joint in the global reference system, expressed as a quaternion (w, x, y, z).

These define the joint's position and orientation in the global reference frame. The position is given as a list of three floats (x, y, z) in meters. The orientation is expressed in both XYZ Euler angles (radians) and as a quaternion (w, x, y, z).

If a `parent` or `child` body also has a local `FRAME` defined, MuSkeMo automatically computes the transformations with respect to that frame, resulting in six more paremeters:

- `pos_in_parent_frame`: The position of the joint center in the local reference frame attached to the parent body (in meters).
- `or_in_parent_frame_XYZeuler`: The orientation of the joint in the local reference frame attached to the parent body (in radians) using an XYZ Euler decomposition.
- `or_in_parent_frame_quat`: The orientation of the joint center in the local reference frame attached to the parent body , expressed as a quaternion (w, x, y, z).
- `pos_in_child_frame`: The position of the joint center in the local reference frame attached to the child body (in meters).
- `or_in_child_frame_XYZeuler`: The orientation of the joint in the local reference frame attached to the child body (in radians) using an XYZ Euler decomposition.
- `or_in_child_frame_quat`: The orientation of the joint center in the local reference frame attached to the child body , expressed as a quaternion (w, x, y, z).

Each joint also has six coordinates which can be named. These are meant to represent the generalized coordinates, or degrees of freedom, of the model. If they are named, this coordinate becomes a degree of freedom when using one of the provided scripts to convert a MuSkeMo model to a simulator (e.g., if you want `coordinate_Rz` to be a degree of freedom in your OpenSim model, give `coordinate_Rz` a name such as 'hip_flexion_r').

- `coordinate_Tx`: Translation along the x-axis
- `coordinate_Ty`: Translation along the y-axis
- `coordinate_Tz`: Translation along the z-axis
- `coordinate_Rx`: Rotation along the x-axis
- `coordinate_Ry`: Rotation along the y-axis
- `coordinate_Rz`: Rotation along the z-axis

The underlying object type in Blender is a UV sphere mesh. This object is essentially only for visualization purposes, but a future version of MuSkeMo may add the option to add axes as well.

7.5 LANDMARK

A `LANDMARK` in MuSkeMo refers to a user-specified point on a rigid body or geometry. Landmarks are currently immediately parented to visual geometry, and can be used to help define (anatomical or local) reference frames. Future updates will include support for parenting to bodies, and exporting/importing global and local positions.

The underlying Blender object type is a UV sphere mesh.

7.6 CONTACT

Contact geometry defines areas or regions where external forces might act on a rigid body. These points represent places where a body or object interacts with another object or the environment during simulations. MuSkeMo does not compute any contact forces, but the user can define contact positions if the model will be used for simulations. MuSkeMo currently only supports contact spheres. Contact the developer if you need more geometry types. Like joints, contact spheres can have a `pos_in_global` and a `pos_in_parent_frame`.

The underlying object type in Blender is a UV sphere mesh.

7.7 FRAME

In MuSkeMo, a `FRAME` is a local coordinate system that can be assigned to any rigid body. It defines a reference position and orientation relative to which other properties of the body (such as the center of mass or moment of inertia) can be computed. Each rigid body can be assigned one (optional) local `FRAME`, which can represent an anatomical reference frame. If a local frame is assigned to a body, body segment parameters with respect to that frame are automatically computed, as are the local transformations of any parent and/or child joints. These are removed if the frame is detached from the body.

7.8 GEOM_PRIMITIVE

In MuSkeMo, it is possible to fit geometric primitive shapes (spheres, cylinders, ellipsoids, and planes) to 3D meshes. This can be useful for defining joint centers of rotation using the skeletal geometry. The geometric details of the shapes are stored for reuse. These are:

- Sphere: `sphere_radius`
- Cylinder: `cylinder_height`
- Ellipsoid: `ellipsoid_radii` (x, y, and z components)
- Plane: `plane_dimensions` (x and y components)

The underlying object types in Blender are regular meshes (UV Sphere, cylinder, ellipsoid).

MuSkeMo implements two different sphere fitting algorithms, described in [5, 11]. The cylinder fit algorithm was adapted from [2]. The ellipsoid fit algorithm [9] was provided courtesy of Mark Semple, who converted a Matlab implementation written by Yuri Petrov to Python. I modified it further to ensure right-handed coordinate systems.

7.9 WRAP

Wrapping objects defined in OpenSim models (but currently not yet Gaitsym models) are included in the model import. The underlying object types in Blender are regular meshes (UV Sphere, cylinder, ellipsoid).

MuSkeMo currently does not yet support physically accurate wrapping, and only provides a very experimental setup as an option during import for model visualizations. When the option is selected during import, muscles get one extra geometry node modifier per wrapping interaction. The "wrapping" is currently achieved by a node setup that projects the section of the muscle inside the wrapping object in a 180 degree arc onto the wrapping surface. This is not a minimal-distance wrapping approach, and is thus not physically realistic, but could potentially be useful for visualizations. The direction of the wrapping can be changed by changing the projection angle - accessible in the node setup. 0°means the wrap is projected around the positive x-direction of the wrapping object's local coordinate system, and the angle is the local z-angle.

Implementing physically accurate realtime wrapping is a challenge in Blender, because it seems that creating custom curve types is not currently possible. The most promising avenue appears to be using Geometry Nodes, but this comes with its own set of challenges because all of the logic needs to be implemented with nodes, and currently Blender does not have something equivalent to a "Scripted Expression"

node. Thus, even a simple equation such as $\sqrt{0.5x^2 + 3xy + \cos(x)}$ requires 8 nodes, and thus things can get very complicated rather quickly.

An implementation of [4] for wrapping cylinders and spheres is currently under development, but is not yet part of the main release.

A Euler angles

A.1 Conventions used by MuSkeMo

Euler angles define rotation matrices that transform between reference frames. Conceptually, they are relatively straightforward: 3D rotations are defined by three successive rotations about different axes in space. However, Euler angles are ambiguous without explicitly defining what convention is being used. It is possible to choose twelve different sets of axes about which we perform the successive rotations (e.g., XYZ, XYX, etc). It is furthermore possible to define these axes with respect to the rotating body (body-fixed, intrinsic), or the global frame (space-fixed, extrinsic). This determines the order of the application of the individual rotations. The rotations can also be defined to rotate the body itself (active rotations), or rotate the frame around a stationary vector (passive rotations). Active and passive rotation matrices are related by being each others' transpose.

To enable cross-compatibility with OpenSim, MuSkeMo uses the same convention as OpenSim. This convention uses a body-fixed (intrinsic) XYZ-decomposition using active rotations (so the object is being rotated, not the reference frame). OpenSim is built on Simbody, and the Simbody Documentation cites page 423 of [6] as their source.

The full rotation matrix ${}^{\mathcal{G}}\mathbf{R}_{\mathcal{B}}$ that MuSkeMo uses to rotate a vector from a body-fixed frame (the $\mathcal{B}$-frame) to the global reference frame (the $\mathcal{G}$-frame) is presented here to prevent ambiguity:

$$ {}^{\mathcal{G}}\mathbf{R}_{\mathcal{B}} = \begin{bmatrix} \cos\phi_y \cos\phi_z & -\cos\phi_y \sin\phi_z & \sin\phi_y \\ \cos\phi_x \sin\phi_z + \cos\phi_z \sin\phi_x \sin\phi_y & \cos\phi_x \cos\phi_z - \sin\phi_x \sin\phi_y \sin\phi_z & -\cos\phi_y \sin\phi_x \\ \sin\phi_x \sin\phi_z - \cos\phi_x \cos\phi_z \sin\phi_y & \cos\phi_z \sin\phi_x + \cos\phi_x \sin\phi_y \sin\phi_z & \cos\phi_x \cos\phi_y \end{bmatrix} \tag{3} $$

Here, ϕ_x, ϕ_y, and ϕ_z represent rotations about the body-fixed, right-handed x-, y-, and z-axes. Brackets are omitted because each cosine and sine only has one term inside it. A vector ${}^{\mathcal{B}}\vec{v}$ expressed in the $\mathcal{B}$-frame (hence the ${}^{\mathcal{B}}$-prefix), can be rotated to the $\mathcal{G}$-frame by pre-multiplying it with ${}^{\mathcal{G}}\mathbf{R}_{\mathcal{B}}$. In other words:

$$ {}^{\mathcal{G}}\vec{v} = {}^{\mathcal{G}}\mathbf{R}_{\mathcal{B}} {}^{\mathcal{B}}\vec{v} \tag{4} $$

Blender itself also implements Euler angles, but uses the convention of space-fixed (extrinsic) rotations. **As a result, what Blender refers to as ZYX-Euler angles is the same as the body-fixed XYZ-Euler angles used by MuSkeMo, OpenSim, SCONE, etc.**

A.2 Understanding decomposed successive rotations

The different Euler-angle conventions in use were the source of much confusion for the author. A step-by-step explanation is provided in the next sections of this Appendix. For further reading, the reader is referred to [6, 10].

Rotation About the x-axis

To rotate by ϕ_x about the x-axis, the rotation matrix $\mathbf{R}_x$ is given by:

$$ \mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi_x & -\sin\phi_x \\ 0 & \sin\phi_x & \cos\phi_x \end{pmatrix} \tag{5} $$

For example, applying this to a unit vector initially pointing in the y-axis direction:

$$\mathbf{R}_x \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \tag{6}$$

If $\phi_x = \frac{\pi}{4}$, this results in:

$$\begin{pmatrix} 0 \\ \cos\frac{\pi}{4} \\ \sin\frac{\pi}{4} \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \tag{7}$$

Thus, the second and third columns give the new y- and z-axes after rotating by ϕ_x in the yz-plane.

Rotation About the y-axis

To rotate by ϕ_y about the y-axis, the rotation matrix $\mathbf{R}_y$ is:

$$\mathbf{R}_y = \begin{pmatrix} \cos\phi_y & 0 & \sin\phi_y \\ 0 & 1 & 0 \\ -\sin\phi_y & 0 & \cos\phi_y \end{pmatrix} \tag{8}$$

Similarly, applying this to a unit vector initially pointing in the x-axis direction:

$$\mathbf{R}_y \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \tag{9}$$

If $\phi_y = \frac{\pi}{4}$, this results in:

$$\begin{pmatrix} \cos\frac{\pi}{4} \\ 0 \\ -\sin\frac{\pi}{4} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{10}$$

Thus, the first and third columns of $\mathbf{R}_y$ give the x- and z-axes directions after the rotation.

Rotation About the z-axis

To rotate by ϕ_z about the z-axis, the rotation matrix $\mathbf{R}_z$ is:

$$\mathbf{R}_z = \begin{pmatrix} \cos\phi_z & -\sin\phi_z & 0 \\ \sin\phi_z & \cos\phi_z & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{11}$$

Similarly, the first and second columns of $\mathbf{R}_z$ give the x- and y-axes directions after rotating about the z-axis.

Three Successive Rotations about Body-Fixed X, Y, and then Z

If we decompose the full rotation into three successive rotations about the body-fixed X-axis, Y-axis, and Z-axis (denoted as $\mathbf{R}_x$, $\mathbf{R}_y$, and $\mathbf{R}_z$ respectively), the combined rotation matrix is:

$$^{\mathcal{G}}\mathbf{R}_{\mathcal{B}} = \mathbf{R}_x\mathbf{R}_y\mathbf{R}_z$$

We said that we were starting with a rotation about X, but the vector will first be multiplied by Rz Why is this?

Consider we want to rotate a vector $\vec{v}_{\mathcal{B}}$ (expressed in the body-fixed frame $\mathcal{B}$) by arbitrary angles about body-fixed X, Y, and Z-axes, in that order. After the rotations, what will $\vec{v}_{\mathcal{G}}$ (the vector in the global frame $\mathcal{G}$) be?

When the two frames are aligned (i.e., when all rotation angles are zero), $\vec{v}_{\mathcal{B}} = \vec{v}_{\mathcal{G}}$.

We start by rotating about the body-fixed x-axis. Thus, we have:

$$\vec{v}_{\mathcal{G}} = \mathbf{R}_x \vec{v}_{\mathcal{B}}$$

At this point, the vector $\vec{v}_{\mathcal{B}}$ is rotated about the $x_{\mathcal{B}}$-axis, which was coincident with the global $x_{\mathcal{G}}$-axis.

Why Can't We Simply Add the Next Rotation by pre-multiplying the previous equation?

The next rotation becomes less intuitive. It may be tempting to add the rotation about the y-axis as follows:

$$\cancel{\mathbf{R}_y \mathbf{R}_x \vec{v}_{\mathcal{B}}}$$

However, this is incorrect! This formulation would result in a rotation about the global $y_{\mathcal{G}}$-axis, because the first rotation was about the global $x_{\mathcal{G}}$-axis (since $x_{\mathcal{B}}$ and $x_{\mathcal{G}}$ were coincident before the first rotation).

Given that we want to rotate about the local $y_{\mathcal{B}}$-axis (which coincides with $y_{\mathcal{G}}$ before the first rotation), we must instead insert $\mathbf{R}_y$ between $\mathbf{R}_x$ and $\vec{v}_{\mathcal{B}}$, as follows:

$$\vec{v}_{\mathcal{G}} = \mathbf{R}_x \mathbf{R}_y \vec{v}_{\mathcal{B}}$$

Note: This implies that successively rotating about body-fixed x- and y-axes is equivalent to successively rotating about the global y- and x-axes, but the order is reversed depending on the frame of reference! Rotations about body-fixed axes are called *intrinsic*, whereas rotations about global axes are called *extrinsic*.

Adding the Third Rotation

A similar argument applies to the third rotation about the local $z_{\mathcal{B}}$-axis. Therefore, the total equation for the vector in the global frame after all three rotations is:

$$\vec{v}_{\mathcal{G}} = \mathbf{R}_x \mathbf{R}_y \mathbf{R}_z \vec{v}_{\mathcal{B}}$$

Thus, a body-fixed XYZ-sequence of rotations (intrinsic) is identical to a global-frame ZYX-sequence of rotations (extrinsic), with the rotation order reversed!

In body-fixed notation, the successive rotation matrices are multiplied from left to right when rotating from the body-fixed frame $\mathcal{B}$ to the global frame $\mathcal{G}$. In the above example, using XYZ about the body-fixed axes gives:

$${}^{\mathcal{G}}\mathbf{R}_{\mathcal{B}} = \mathbf{R}_x \mathbf{R}_y \mathbf{R}_z$$

The inverse operation, which rotates from the global frame back to the body-fixed frame, is given by:

$${}^{\mathcal{B}}\mathbf{R}_{\mathcal{G}} = ({}^{\mathcal{G}}\mathbf{R}_{\mathcal{B}})^{-1} = ({}^{\mathcal{G}}\mathbf{R}_{\mathcal{B}})^T = \mathbf{R}_z^T \mathbf{R}_y^T \mathbf{R}_x^T$$

B Rotation quaternions

To be written. MuSkeMo uses algorithms based on [3].

References

[1] Samuel J. Coatham, William I. Sellers, and Thomas A. Püschel. "Convex Hull Estimation of Mammalian Body Segment Parameters". In: *Royal Society Open Science* 8.6 (June 2021), p. 210836. ISSN: 2054-5703. DOI: 10.1098/rsos.210836. URL: https://royalsocietypublishing.org/doi/10.1098/rsos.210836 (visited on 11/16/2021).

[2] David Eberly. *Least Squares Fitting of Data by Linear or Quadratic Structures*. Geometric Tools. URL: https://www.geometrictools.com/Documentation/LeastSquaresFitting.pdf.

[3] David H. Eberly and Ken Shoemake. *Game Physics*. 1st ed. The Morgan Kaufmann Series in Interactive 3D Technology. Amsterdam ; Boston: Elsevier/Morgan Kaufmann, 2004. 776 pp. ISBN: 978-1-55860-740-8.

[4] Brian A. Garner and Marcus G. Pandy. "The Obstacle-Set Method for Representing Muscle Paths in Musculoskeletal Models". In: *Computer Methods in Biomechanics and Biomedical Engineering* 3.1 (Jan. 2000), pp. 1–30. ISSN: 1025-5842, 1476-8259. DOI: 10.1080/10255840008915251. URL: http://www.tandfonline.com/doi/abs/10.1080/10255840008915251 (visited on 10/11/2024).

[5] Charles F Jekel. "Digital Image Correlation on Steel Ball". In: *Obtaining Non-Linear Orthotropic Material Models for Pvc-Coated Polyester via Inverse Bubble Inflation (MSc Thesis)*. Stellenbosch University, 2016, pp. 83–87. URL: https://hdl.handle.net/10019.1/98627.

[6] Thomas R. Kane, Peter W. Likins, and David A. Levinson. *Spacecraft Dynamics*. New York Hamburg: McGraw-Hill, 1983. 436 pp. ISBN: 978-0-07-037843-8.

[7] Sophie Macaulay et al. "Decoupling Body Shape and Mass Distribution in Birds and Their Dinosaurian Ancestors". In: *Nature Communications* 14.1 (Mar. 22, 2023), p. 1575. ISSN: 2041-1723. DOI: 10.1038/s41467-023-37317-y. URL: https://www.nature.com/articles/s41467-023-37317-y (visited on 04/03/2023).

[8] W. I. Sellers et al. "Minimum Convex Hull Mass Estimations of Complete Mounted Skeletons". In: *Biology Letters* 8.5 (2012), pp. 842–845. ISSN: 1744957X. DOI: 10.1098/rsbl.2012.0263. pmid: 22675141.

[9] Mark Semple. *pyEllipsoid_Fit*. URL: https://github.com/marksemple/pyEllipsoid_Fit.

[10] Heike Vallery and Arend L Schwab. *Advanced Dynamics*. Delft University of Technology, 2019. 481 pp. ISBN: 978-94-6186-948-7.

[11] Sumith Yesudasan. *Fast Geometric Fit Algorithm for Sphere Using Exact Solution*. Version 1. 2015. DOI: 10.48550/ARXIV.1506.02776. URL: https://arxiv.org/abs/1506.02776 (visited on 10/11/2024). Pre-published.