FreeBody Visualiser
Developer Guide

v0.1

Justas Medeisis
jm4711@ic.ac.uk

September 8, 2015

Introduction

The standalone FreeBody Visualiser (FBV) presents musculoskeletal model data generated by the FreeBody software package[1, 3] using interactive, animated three-dimensional graphics. FreeBody outputs that can be viewed with FBV include the movement of bones and muscles, muscle activations and joint contact forces. Use FBV to:

- find errors in the model;
- quickly and effectively analyse the model;
- demonstrate the model to colleagues and / or the public.

This source code manual contains detailed information about the technology and software architecture behind FBV. The document is split into multiple sections, each of which presents a different perspective into the project and its files.

Contents

1 Overview

The FreeBody Visualiser (FBV) is powered by Unity,[6] a cross-platform engine and development system for interactive 3D applications. This document is not an exhaustive reference on Unity-specific features. For up-to-date details on Unity's APIs and editor, refer to Unity's official documentation and tutorials.[7] FBV was developed with Unity 5.1 and Microsoft Visual Studio 2013 with the Visual Studio 2013 Tools for Unity extension.

Like for most Unity projects, everything of interest in FBV can be found under the *Assets* folder. *MainScene* defines all the `GameObject`s and the layout for the UI (see Section 1.1). The *Scripts* folder contains all the logic for FBV in the form of C# scripts (see Section 1.2). The *Prefabs* folder contains templates (known as prefabs in Unity terminology) for `GameObject`s, primarily used in FBV's UI; modifying a template updates all `GameObject`s that derive from it. The *Fonts* folder contains the files for the Roboto font[5] which is used throughout FBV. The *Cardboard* folder contains scripts and resources used by the Google Cardboard plugin (see Section 5.3). The remaining folders contain more esoteric files.

1.1 GameObjects and Components

As a Unity project, FBV consists of a number of entities known as `GameObject`s (see Fig. 1). Each `GameObject` is nothing more than a unique bundle of components, which necessarily contains the `Transform` and optionally contains any number of additional components. A `GameObject` may also contain subordinate (known as *child*) `GameObject`s. The most common components found in FBV's `GameObject`s are:

- `Transform` - a mandatory component. Defines the `GameObject`'s position in 3D space.
- `Script` - a component that refers to a script (i.e. source code) file that operates on its parent `GameObject` and, potentially, other `GameObject`s. Also known as a *behaviour*. Most of FBV's logic and rendering takes place in scripts.
- `Mesh Renderer` - a component responsible for rendering (drawing) a mesh (see Section 7 for definition), specifically used to render the 3D bone models.
- `Canvas` and `Rect Transform` - components used to define the FBV user interface (UI) using Unity's "New UI" system (introduced in

Unity 4.6).

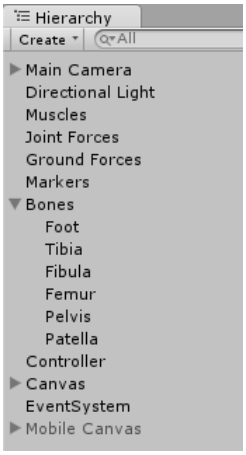


Figure 1: FreeBody Visualiser's `GameObject` hierarchy as seen in the Unity editor.

Below is a list of all the major FBV `GameObject`s, with a short summary of each:

- `Main Camera` - represents the main view into the 3D virtual space in which FBV renders all graphics.
- `Directional Light` - through its `Light` component, defines the direction, strength and other properties of the lighting for all rendered meshes.
- `Muscles` - renders FreeBody model muscles.
- `Joint Forces` - renders FreeBody model joint contact forces.
- `Ground Forces` - renders FreeBody model ground reaction forces.
- `Markers` - renders FreeBody model markers.
- `Bones` - a group of `GameObject`s representing bones in the lower limb model, each of which renders its corresponding mesh.
- `Controller` - responsible for all FBV logic.
- `Canvas` - a group of `GameObject`s that define the FBV UI for PC.

- `Mobile Canvas` - a group of `GameObject`s that define the FBV UI for mobile devices, i.e. for FreeBody Visualiser Mobile.
- `EventSystem` - responsible for dispatching keyboard, mouse and orientation input events to all other objects and scripts.

1.2 Scripts

Scripts drive most of the interesting parts of FBV. Every `GameObject` has at least one script; some are generic Unity scripts, but most are unique to FBV.

Unity supports multiple programming languages for writing scripts. FBV's scripts are all written in C#. Below is a list of all the custom scripts in FBV, with a short summary of each:

- **Data**
  - `ModelController.cs` - delegates most logic to other scripts. Responsible for:
    1. loading a FreeBody model,
    2. updating the UI and other scripts after a model has loaded, and
    3. setting up UI controls that directly control visualisation.
  - `FrameController.cs` - keeps track of the current animation frame.
  - `FreeBodyModel.cs` - holder for FreeBody 1.1 XML parameter file data with fields relevant to visualisation.
  - `ModelParameterLoader.cs` - reads FreeBody 1.1 XML parameter files, and returns a populated `FreeBodyModel`.
  - `DataPathUtils.cs` - contains list of all relevant file paths for a given `FreeBodyModel`.
  - `MusclePart.cs` - contains list of all lower limb muscle parts.
  - `BoneData.cs` - holder for loaded bone position and orientation data.
  - `MuscleDataLoader.cs` - reads and parses FreeBody files related to muscle visualisation.
  - `MarkerDataLoader.cs` - reads and parses FreeBody files related to marker visualisation.

- `JointForceDataLoader.cs` - reads and parses FreeBody files related to joint contact force visualisation.
- `GroundForceDataLoader.cs` - reads and parses FreeBody files related to ground contact force visualisation.
- `BoneDataLoader.cs` - reads and parses FreeBody files related to bone visualisation.

- `FloatCsvFileReader.cs` - common logic for reading .CSV files containing lists of floats, vectors and / or quaternions into the respective data structures for use within FBV.
- `PrimitiveUtils.cs` - generates primitive meshes.
- `StlFileReader.cs` - parses .STL 3D model files into meshes.
- `BinaryReaderExtension.cs` - allows for "binary" data files to be interpreted and read as "ASCII" files. Used to enable more elegant code in `StlFileReader.cs`.

- `FrameMismatchException.cs` - used to report frame count mismatch during loading.

- **Graphics**
  - `MuscleMesh.cs` - renders muscle parts as lines.
  - `MarkerMesh.cs` - renders markers as spheres.
  - `JointForceMesh.cs` - renders joint contact force vectors as 3D arrow pairs.
  - `GroundForceMesh.cs` - renders ground contact force vectors as 3D arrow pairs.
  - `BoneMesh.cs` - renders a bone mesh.
- **UI**
  - `UIController.cs` - manages interactions strictly between different UI elements, e.g. toggles that control visibility of UI panels.
  - `CameraOrbit.cs` - allows for camera (user's view) control that orbits the visualised model.
  - `DragPanelHotspot.cs` - allows for the main control panel to be dragged around the window.

- `FPSDisplay.cs` - updates a text field with the program's current frames per second.
- `VRController.cs` - manages UI in the context of the FBV Mobile 3D VR mode.
- `ToggleWrapper.cs`, `ToggleUtils.cs`, `ToggleSprite.cs` - used by toggle elements in the UI.

2 Modules

The FBV project can be thought of as a collection of *modules*, each dedicated to loading and rendering a specific FreeBody model feature. The modules are all linked and managed by a global *controller*.

2.1 Controller

The controller is responsible for loading FreeBody model data relevant to all modules. It load the FreeBody 1.1 XML parameter file[2] to parse the location of files necessary to load all modules. The controller updates all modules once a new XML parameter file has been parsed and a FreeBody model is ready to be loaded.

The controller also acts as the conduit for UI control of all modules. It hooks into events from toggles, buttons and sliders to toggle the visibility of and configure the visualisation settings of modules.

Relevant `GameObject`s:

- `Controller`

Relevant scripts:

- `ModelController.cs`
- `FreeBodyModel.cs`
- `ModelParameterLoader.cs`
- `DataPathUtils.cs`
- `FrameController.cs`

2.2 Muscle Module

The muscle module displays muscle part positions and, optionally, activations, per frame. Data is loaded from multiple files. Muscles are rendered as a mesh consisting of screen-aligned rectangles that approximate lines. Width and color of these rectangles (muscle parts) is modulated according to normalised activation values, or, if activation display is disabled, width is fixed and color set according to muscle part membership to pre-defined muscle groups.

Relevant `GameObject`s:

- `Muscles`

Relevant scripts:

- `MusclePart.cs`
- `MuscleDataLoader.cs`
- `MuscleMesh.cs`
- `FloatCsvFileReader.cs`

2.3 Force Module

Forces include joint contact forces and ground reaction forces. Data necessary for force display - position, direction, and magnitude, per frame - is loaded from multiple files. Forces are rendered as 3D arrows: the arrow tip denotes the position; the arrow orientation denotes the direction; and the arrow length denotes the magnitude of the corresponding force.

Relevant `GameObjects`:

- `Joint Forces`
- `Ground Forces`

Relevant scripts:

- `JointForceDataLoader.cs`
- `GroundForceDataLoader.cs`
- `JointForceMesh.cs`
- `GroundForceMesh.cs`
- `FloatCsvFileReader.cs`
- `PrimitiveUtils.cs`

2.4 Marker Module

Markers are the simplest feature to visualise. They are loaded in as positions, per frame, and are rendered as simple spheres.

Relevant `GameObjects`:

- `Markers`

Relevant scripts:

- `MarkerDataLoader.cs`
- `MarkerMesh.cs`
- `FloatCsvFileReader.cs`
- `PrimitiveUtils.cs`

2.5 Bone Module

The bone module loads in bone positions and rotations, per frame, as well as position and rotation offsets. Bone models are dynamically loaded from .STL model files and are rendered as meshes. Performance may suffer if highly detailed .STL models are used. Note that due to a limit on mesh size, large .STL models need to be rendered using more than one mesh.

Relevant `GameObjects`:

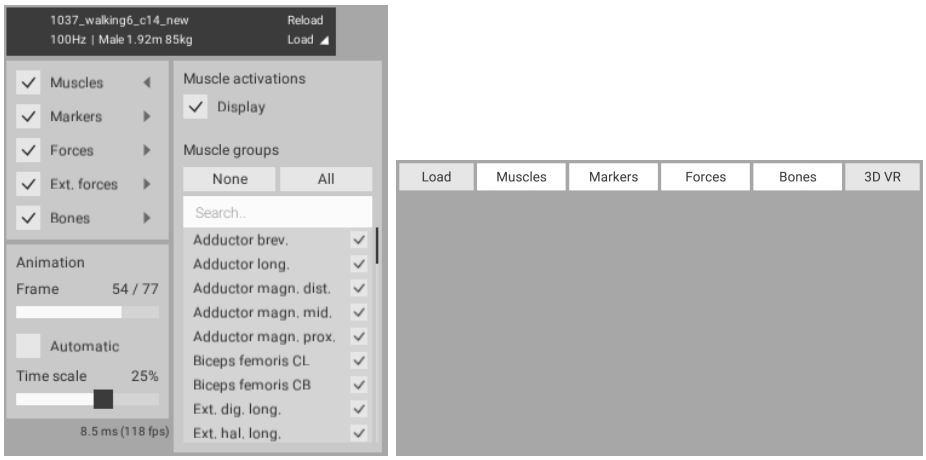
- `Bones`
- `Foot`
- `Tibia`
- `Fibula`
- `Femur`
- `Pelvis`
- `Patella`

Relevant scripts:

- `BoneData.cs`
- `BoneDataLoader.cs`
- `BoneMesh.cs`
- `FloatCsvFileReader.cs`
- `StlFileReader.cs`

3 User Interface

The user interface is constructed using Unity's "New UI" system within the `Canvas` (see Fig. 2a) and `Mobile Canvas` (see Fig. 2b) `GameObject`s to provide two modes, depending on the platform that FBV is running on (see Section 5 for details on FBV Mobile).



(a) Desktop UI. (b) Mobile UI.

Figure 2: FBV supports two different UI modalities.

Relevant `GameObject`s:

- `Canvas`
- `Mobile Canvas`
- `EventSystem`

Relevant scripts:

- `CameraOrbit.cs`
- `UIController.cs`
- `DragPanelHotspot.cs`
- `FPSDisplay.cs`

4 Data Loading

4.1 Finding Files

FBV loads a model by reading in the paths of all its relevant files from a FreeBody 1.1 XML parameter file. This is done by the `ModelParameterLoader.cs` script. Refer to the FreeBody 1.1 User's Guide for an overview of the schema.[2]

The XML elements and attributes parsed and used by FBV are listed below:

- `study_level_parameters`
  - `study_name`
  - `responsible_person`
  - `output_directory_path_for_visualisation`
  - `output_directory_path_for_optimisation`
- `universal_physical_parameters`
  - `frames_per_second`
  - `radius_per_marker_metres`
- `subject`
  - `subject_sex`
  - `subject_height_metres`
  - `subject_mass_kg`
  - `subject_anatomy_dataset_path`
  - `subject_anatomy_dataset_file_name`
- `dynamic_trial_parameters`
  - `start_frame_number`
  - `end_frame_number`

An example XML parameter file can be found on the following page. Please note that this file does not strictly adhere to the full original FreeBody 1.1 XML parameter file schema[2]; instead, it demonstrates use of the subset of elements and attributes parsed by FBV. The name of the file is arbitrary.

study_params.xml

```
<?xml version="1.0" encoding="utf-8"?>
<study_level_parameters
  study_name="1037_walking6_c14_new"
  responsible_person="Ziyun Ding (z.ding@imperial.ac.uk)"
  output_directory_path_for_visualisation=
  "C:\example\1037_C14\walking6\Outputs\Muscle_geometry"
  output_directory_path_for_optimisation=
  "C:\example\1037_C14\walking6\Outputs\Optimisation">

  <universal_physical_parameters
    frames_per_second="100"
    radius_per_marker_metres="0.007"/>

  <subject
    subject_sex="Male"
    subject_height_metres="1.92"
    subject_mass_kg="85"
    subject_anatomy_dataset_path="C:\example\Anatomy_dataset"
    subject_anatomy_dataset_file_name="Zhan303_C14_dataset.xml">

    <dynamic_trial_parameters
      start_frame_number="1"
      end_frame_number="77" />

  </subject>

</study_level_parameters>
```

Once the XML parameter file has been parsed, the `DataPathUtils.cs` script derives paths to all files needed for model visualisation. The file paths in the list in Section 4.1.1 use the following abbreviations that refer to paths obtained from the XML parameter file attributes (see Section 4.1):

- `VIS` = value of `output_directory_path_for_visualisation`.
- `OPT` = value of `output_directory_path_for_optimisation`.
- `STUDY` = value of `study_name`.
- `AN_PATH` = value of `subject_anatomy_dataset_path`.
- `AN_PREFIX` = value of `subject_anatomy_dataset_file_name`, with the suffix trimmed; e.g. if the value is `Zhan303_C01_dataset.xml`, `AN_PREFIX` = `Zhan303_C01`.

4.1.1 Files Used by FBV

The following is a list of all files used by FBV to display the model, sorted by model feature. All abbreviations (capitalised) are explained in the previous section.

- **Muscles**:
  - `VIS/STUDY_muscle_path{`i`}.csv` - position of each muscle's origin and insertion points, per frame. Note that this path refers to multiple files, where the $\{i\}$ is replaced by $\{i \in \mathbb{Z} \mid 0 \leq i < 163\}$, and 163 is the number of distinct muscle elements.
  - `OPT/STUDY_force_gcs.csv` - actual activation of each muscle, per frame.
  - `OPT/STUDY_force_ub.csv` - maximum activation of each muscle, per frame.
- **Joint contact forces**:
  - `VIS/STUDY_rot_centres_gcs.csv` - positions of ankle, knee and hip joints.
  - `VIS/STUDY_tf_contact_gcs.csv` - positions of lateral and medial tibiofemoral joints.
  - `OPT/STUDY_force_gcs.csv` - magnitudes and directions of joint contact forces. (note same file as for muscle activations)
- **External forces**:
  - `VIS/STUDY_external_forces.csv` - positions, magnitudes and directions of ground reaction forces.
- **Markers**:
  - `VIS/STUDY_dynamic_marker.csv` - positions of dynamic markers.
  - `VIS/STUDY_virtual_static_marker.csv` - positions of virtual static markers.
- **Bones**:
  - `VIS/STUDY_anatomy_model_origin.csv` - default position offset for each bone in model.
  - `VIS/STUDY_anatomy_model_orientation.csv` - default rotation offset for each bone in model.

- `VIS/STUDY_origins.csv` - position of each bone in model for each frame.
- `OPT/STUDY_lcs_quaternion.csv` - rotation of each bone in model for each frame.
- (optional) `AN_PATH/AN_PREFIX_bones/AN_PREFIX_Foot.stl`
- (optional) `AN_PATH/AN_PREFIX_bones/AN_PREFIX_Tibia.stl`
- (optional) `AN_PATH/AN_PREFIX_bones/AN_PREFIX_Fibula.stl`
- (optional) `AN_PATH/AN_PREFIX_bones/AN_PREFIX_Patella.stl`
- (optional) `AN_PATH/AN_PREFIX_bones/AN_PREFIX_Femur.stl`
- (optional) `AN_PATH/AN_PREFIX_bones/AN_PREFIX_Pelvis.stl`

4.2 Parsing CSV Files

CSV files are parsed by the `FloatCsvFileReader.csv` script, which makes use of the fact that all FreeBody output CSV files are tables of floats and reads the files one line at a time. Depending on the feature being loaded, delegate scripts parse the floats as either floats (for magnitudes, activations), vectors (for positions, forces) or quaternions (for orientations).

4.3 Parsing STL Files

STL model files are parsed by the `StlFileReader.cs` script, which supports both binary and ASCII STL files.

5 Support for Mobile Devices

The experimental FreeBody Visualiser Mobile (FBVM) is FBV for mobile devices (see Fig. 3). FBVM has most of the capabilities of FBV, with some additional features and benefits:

- view FreeBody models on the go with no need for a powerful workstation;
- view FreeBody models in interactive three-dimensional space;
- load FreeBody models from a single file with no need for manual configuration.

Figure 3: FreeBody Visualiser Mobile.

FBVM is designed to be intuitive and simple to use, with fewer configuration options but all the power of FBV. FBVM is a mobile application for smartphones and tablets running an Android OS. FBVM is classified as experimental because it has received only limited testing and uses a novel package format for distribution and loading of FreeBody data.

FBVM is, in fact, the same Unity project as FBV. All the logic and rendering is identical to FBV, with two key differences:

1. The `Canvas` UI container is disabled and the `Mobile Canvas` is enabled instead to display a simplified UI with larger targets that is more appropriate for touch screens.

2. A Unity plugin exists for each mobile platform supported by FBVM (currently, only Android), with logic to launch a system-specific file picker and allow the user to select a FreeBody model package (see Section 5.2) to load.

The platform-specific plugins can be found in the *Assets/Plugins* folder.

5.1 Android Plugin

The Android plugin is a .JAR library file, developed as an Android Studio project. The Android library project consists of only three files of interest:

1. `MainActivity.java` - extends from `GoogleUnityActivity` provided by the Cardboard SDK (see Section 5.3); hosts the Unity player and can launch the native Android file picker.

2. `FreeBodyFileLoader.java` - unpacks a FreeBody model package .ZIP file (see Section 5.2) after it has been selected by the user to local storage, such that it can be treated by FBVM just like FBV treats

3. `AndroidManifest.xml` - declares various app configuration values.

5.2 FreeBody Model Package

FBVM uses an alternative data storage mechanism to that of FBV. To greatly simplify distribution and loading on mobile devices, all FreeBody files are consolidated into a single directory, here referred to as a FreeBody *model package*. A model package is an entirely self-contained directory that contains all the files of a single FreeBody model. The directory is compressed to become a .ZIP file archive, which offers two benefits:

1. FreeBody model packages can be distributed as a single file, and
2. the compressed file, by definition, is of a smaller size.

The layout of the model package can be seen below. The package contains a standard FreeBody 1.1 XML parameter file, which must be named `parameters.xml`, at its root path. The one primary requirement for the XML parameter file unique to this model package format is that **all file paths must be relative and contained within the package**. A relative path in this context is a file path with its root as the root of the model package directory (i.e. the location of the `parameters.xml` file). The name of the model package .ZIP archive is arbitrary.

Model package layout

```
study_name.zip
| parameters.xml
| <... files>
```

An example XML parameter file that conforms to this relative path requirement for a FreeBody model package can be found on the following page (compare to a standard XML parameter file in Section 4.1). Note the relative path values for the attributes

- `output_directory_path_for_visualisation`,
- `output_directory_path_for_optimisation`, and
- `subject_anatomy_dataset_path`.

Note also that paths are defined with forward-slashes (/) instead of back-slashes (\) to better conform to the major mobile OS (derived from Unix) conventions.

parameters.xml

```
<?xml version="1.0" encoding="utf-8"?>
<study_level_parameters
  study_name="1037_walking6_c14_new"
  responsible_person="Ziyun Ding (z.ding@imperial.ac.uk)"
  output_directory_path_for_visualisation=
  "./Outputs/Muscle_geometry"
  output_directory_path_for_optimisation=
  "./Outputs/Optimisation">

  <universal_physical_parameters
    frames_per_second="100"
    radius_per_marker_metres="0.007"/>

  <subject
    subject_sex="Male"
    subject_height_metres="1.92"
    subject_mass_kg="85"
    subject_anatomy_dataset_path="./Anatomy_dataset"
    subject_anatomy_dataset_file_name="Zhan303_C14_dataset.xml">

    <dynamic_trial_parameters
      start_frame_number="1"
      end_frame_number="77" />

  </subject>

</study_level_parameters>
```

The corresponding file structure of the model package can be seen below.

Example model package layout

```
C14_walking6.zip
| parameters.xml
--Anatomy_dataset
  | Zhan303_C14_dataset.xml
  --Zhan303_C14_bones
    | <... files>
--Outputs
  --Muscle_geometry
    | <... files>
  --Optimisation
    | <... files>
```

5.3 Virtual Reality Mode

FBVM has a 3D virtual reality (VR) mode which visualises the model in stereoscopic 3D (see Fig. 4). This is achieved by use of Google's Cardboard SDK for Unity v0.5.0.[4]

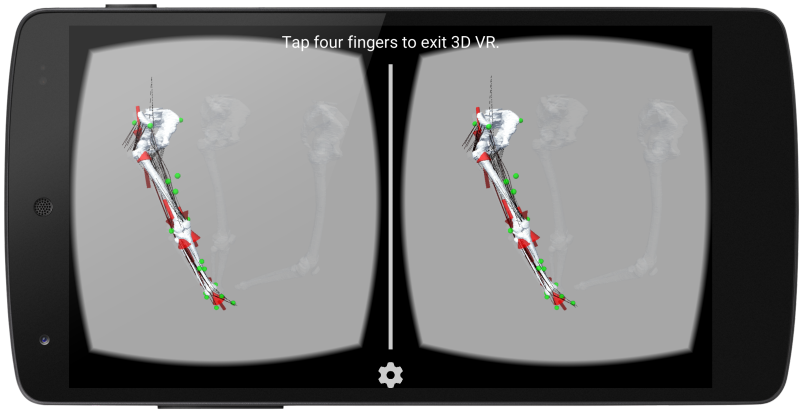


Figure 4: FreeBody model in stereoscopic 3D.

Relevant `GameObjects`:

- `Main Camera`
- `Controller`

Relevant scripts:

- `VRController.cs`
- `UIController.cs`

6 Future Work

Suggested future work:

- Create standalone script to automatically convert an existing well-formed XML parameter file to a self-contained FreeBody model package.
- Add support for loading FreeBody model package format archives to the desktop version of FBV, to enable easier distribution and cross-platform support.
- Deploy and test FBV for Mac and Linux.
- Deploy and tweak FBVM to support iOS.
- Add tests.
- Devise a data schema to dynamically determine relevant bone models to load, expanding visualisation support beyond the lower limb models currently defined by FreeBody 1.1. The fixed bone `GameObject`s are currently the primary lower limb specific part of the project.
- Devise a data schema to dynamically determine number of muscle parts to load, for same reasons as above.
- On completion of the two above items, add support for display of any animated anatomic biomechanical model adhering to the FreeBody model data format.

7 Glossary

Common terms used throughout this document:

- **FreeBody model** - a set of files conforming to the FreeBody data structure, including input and output files of the FreeBody Lower Limb Model and the FreeBody Matlab Optimisation and Visualisation applications and (optionally) 3D bone model files, *for a single study*. In FreeBody 1.1, the locations of these files are declared in an XML parameter file.
- **FreeBody model package** - a custom schema for a self-contained .ZIP file archive that contains a valid FreeBody model XML parameter file and all associated files needed for visualisation.
- **Mesh** - (as used in a computer graphics context) short for polygon mesh, which is a collection of vertices (points in 3D space) and faces (planes in 3D space) connecting them that together define a solid 3D shape.

References

[1] D. J. Cleather and A. M. J. Bull. "The development of a segment-based musculoskeletal model of the lower limb: introducing FREEBODY". en. In: *Royal Society Open Science* 2.6 (June 2015), pp. 140449–140449. ISSN: 2054-5703. DOI: 10.1098/rsos.140449. URL: http://rsos.royalsocietypublishing.org/content/2/6/140449.abstract.

[2] Ziyun Ding. *XML-based interactive lower limb musculoskeletal modelling software FreeBody 1.1 Users Guide*. 2015. URL: http://www.msksoftware.org.uk/software/freebody/.

[3] Ziyun Ding, Daniel Cleather, and A.M.J. Bull. *FreeBody*. 2015. URL: http://www.msksoftware.org.uk/software/freebody/ (visited on July 10, 2015).

[4] Google Inc. *Cardboard SDK for Unity*. 2015. URL: https://developers.google.com/cardboard/unity/ (visited on Dec. 26, 2014).

[5] Christian Robertson. *Roboto - Google Fonts*. 2014. URL: https://www.google.com/fonts/specimen/Roboto (visited on Sept. 8, 2015).

[6] Unity Technologies. *Unity - Game Engine*. 2015. URL: http://unity3d.com/ (visited on Sept. 3, 2015).

[7] Unity Technologies. *Unity - Learn*. URL: http://unity3d.com/learn (visited on Sept. 3, 2015).