
AssimpKit Documentation

Release 1.0

Deepak Surti

January 02, 2017

1	AssimpKit at a glance	3
1.1	Introduction	3
1.2	Why AssimpKit	3
2	Getting Started	5
2.1	Requirements	5
2.2	Installation	5
2.3	API Overview	5
3	Tutorial	7
3.1	Load a 3D model	7
3.2	Load Skeletal Animations	8
3.3	Serialization and integrating with asset pipeline	10
3.4	Using .scn archives exported from AssimpKit in your app	10
4	Example Apps	13
4.1	Build Instructions for the example apps	13
4.2	About the iOS example app	13
4.3	About the macOS example app	15
5	FAQ	19
6	Installation	21
6.1	Assimp Dependency	21
7	XCode Project Layout	23
7.1	Common Code	23
7.2	Targets	24
7.3	Example Apps	24
8	Design	25
8.1	Classes	25
8.2	Generating the scene kit scene graph	26
8.3	Pass 1: Generating the graph with geometry	26
8.4	Pass 2: Generating Skeletal Animations	28
8.5	Pass 3: Generating native SCNScene instances	33
8.6	Loading Animations	33
9	Contributing	35

10 API	37
11 Release Notes	39
11.1 Version 1.0	39

AssimpKit is an open source library [hosted on Github](#) that converts 30 3D file formats using [Assimp](#) to [Scene Kit](#) scenes.

The main documentation for this library is split into:

- *User Documentation*
- *Developer Documentation*
- *Dessert*

AssimpKit at a glance

1.1 Introduction

AssimpKit is a cross platform library (macOS, iOS) that converts the files supported by Assimp to Scene Kit scenes.

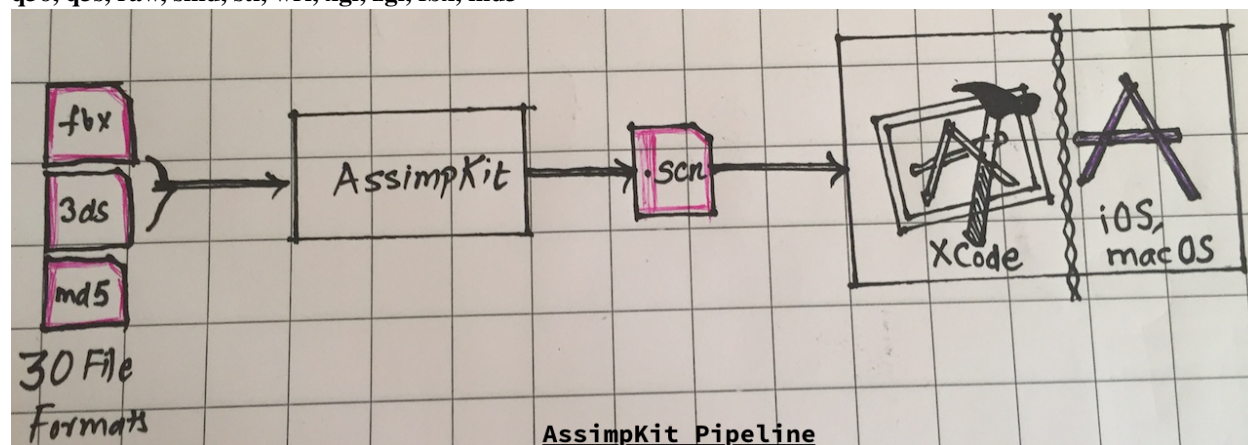
1.2 Why AssimpKit

AssimpKit currently supports 30 file formats that allows you to use these files directly in SceneKit without having to convert these to any of the files that SceneKit or Model IO supports thereby saving an extra step in your asset pipeline.

1.2.1 File formats supported

Currently AssimpKit supports the following file formats:

3d, 3ds, ac, b3d, bvh, cob, dae, dxf, hmp, ifc, irr, md2, md5mesh, md5anim, m3sd, nff, obj, off, mesh.xml, ply, q3o, q3s, raw, smd, stl, wrl, xgl, zgl, fbx, md3



Getting Started

2.1 Requirements

- Xcode 8.0 or later
- ObjC 2.0
- iOS 10.0 or later
- macOS 10.11 or later

2.2 Installation

AssimpKit is [Carthage](#) compatible.

To install with Carthage, follow the instructions on Carthage.

Your application Cartfile should have the following entry for AssimpKit:

```
github "dmsurti/AssimpKit"
```

After carthage update, add the appropriate platform framework (iOS, macOS) to your project. The frameworks are placed in iOS and Mac subdirectories under the `Carthage/Build` directory of your project.

2.2.1 Important Build Setting for iOS applications only

If you are developing an iOS application, set the `Enable Bitcode` under `Build Settings->Build Options` of your target to `NO`.

2.3 API Overview

Table below lists the important classes in AssimpKit.

Class/Category	Description
<code>SCNScene(AssimpImport)</code>	The container for all SceneKit content, loaded with assimp.
<code>SCNNode(AssimpImport)</code>	The node category to add animation to a node.

You can use the `AssimpImport` category defined on `SCNScene` to load scenes. The post processing steps that the assimp library can apply to the imported data are listed at `AssimpKitPostProcessSteps`.

The imported `SCNAssimpScene` contains a model `SCNScene` which represents the 3D model and the skeleton if it contains one, in addition to the array of animations each represented by an `SCNScene` object. The `SCNAssimpScene` also contains the key names for the animations which can be used when adding, removing animations.

The `AssimpImport` category defined on `SCNScene` and `SCNNode` also contain a method to add a skeltal animation.

For more information, refer to the Tutorial.

Install the `AssimpKit.framework` following the [Installation](#) guide.

It is recommended to go through the [API Overview](#), before working through the tutorial.

3.1 Load a 3D model

3.1.1 Load a Scene which is a part of your app bundle

You can load a scene which is a part of your app bundle, as in Listing I-1 below.

Listing I-1: Load a scene which is part of your app bundle:

```
#import <AssimpKit/PostProcessing.h>
#import <AssimpKit/SCNScene+AssimpImport.h>

NSString *spider = @"spider.obj";

// Start the import on the given file with some example postprocessing
// Usually - if speed is not the most important aspect for you - you'll
// probably request more postprocessing than we do in this example.
SCNAssimpScene* scene =
    [SCNScene sceneNamed:spider
     postProcessFlags:AssimpKit_Process_FlipUVs |
                     AssimpKit_Process_Triangulate]];

// retrieve the SCNView
SCNView *scnView = (SCNView *)self.view;

// set the model scene to the view
scnView.scene = scene.modelScene;
```

3.1.2 Load a scene by specifying a file URL

You can load a scene by specifying a file URL, as in Listing I-2 below.

Listing I-2: Load a scene with a file URL:

```
#import <AssimpKit/PostProcessing.h>
#import <AssimpKit/SCNScene+AssimpImport.h>
```

```
// The path to the file path must not be a relative path
NSString *soldierPath = @"/assets/apple/attack.dae";

// Start the import on the given file with some example postprocessing
// Usually - if speed is not the most important aspect for you - you'll
// probably request more postprocessing than we do in this example.
SCNAssimpScene *scene =
    [SCNScene assimpSceneWithURL:[NSURL URLWithString:soldierPath]
     postProcessFlags:AssimpKit_Process_FlipUVs |
     AssimpKit_Process_Triangulate]];

// retrieve the SCNView
SCNView *scnView = (SCNView *)self.view;

// set the model scene to the view
scnView.scene = scene.modelScene;
```

3.2 Load Skeletal Animations

AssimpKit builds on top of the skeletal animation support provided by SceneKit. For any scene that contains skeletal animation data, it creates a skinner and sets it to the node whose geometry the skinner deforms. The animated scene after importing will contain a set of animations each with a unique animation key. You only have to add the animation to the scene to play it, without even worrying about which node to add the animation.

AssimpKit supports skeletal animations irrespective of whether they are defined in one animation file or multiple animation files.

3.2.1 Load an animation which is defined in the same file

You can load an animation which is defined in the same file as the model you are animating, using the listing I-3 below.

Listing I-3: Load and play an animation which is defined in the same file:

```
#import <AssimpKit/PostProcessing.h>
#import <AssimpKit/SCNScene+AssimpImport.h>

// The path to the file path must not be a relative path
NSString *boyPath = @"/of/assets/astroBoy_walk.dae";

// Start the import on the given file with some example postprocessing
// Usually - if speed is not the most important aspect for you - you'll
// probably request more postprocessing than we do in this example.
SCNAssimpScene *scene =
    [SCNScene assimpSceneWithURL:[NSURL URLWithString:boyPath];
     postProcessFlags:AssimpKit_Process_FlipUVs |
     AssimpKit_Process_Triangulate]];

// get the animation which is defined in the same file
NSString *walkID = @"astroBoy_walk-1";
SCNScene *walkAnim = [scene animationSceneForKey:walkAnim];

// add the walk animation to the boy model scene
[scene.modelScene addAnimation:attackAnim];

// retrieve the SCNView
```

```
SCNView *scnView = (SCNView *)self.view;

// set the model scene to the view
scnView.scene = scene.modelScene;
```

3.2.2 Load an animation which is defined in a separate file

You can load an animation which is defined in a separate file from the model you are animating, using the listing I-5 below.

Listing I-4: Load and play an animation which is defined in a separate file:

```
#import <AssimpKit/PostProcessing.h>
#import <AssimpKit/SCNScene+AssimpImport.h>

// The path to the file path must not be a relative path
NSString *explorer = @"/assets/apple/explorer_skinned.dae";

// Start the import on the given file with some example postprocessing
// Usually - if speed is not the most important aspect for you - you'll
// probably request more postprocessing than we do in this example.
SCNAssimpScene *scene =
    [SCNScene assimpSceneWithURL:[NSURL URLWithString:explorer]
     postProcessFlags:AssimpKit_Process_FlipUVs |
                     AssimpKit_Process_Triangulate];

// load an animation which is defined in a separate file
NSString *jumpAnim = @"/explorer/jump_start.dae";
SCNAssimpScene *jumpStartScene =
    [SCNAssimpScene assimpSceneWithURL:[NSURL URLWithString:jumpAnim]
     postProcessFlags:AssimpKit_Process_FlipUVs |
                     AssimpKit_Process_Triangulate];

// get the animation with animation key
NSString *jumpId = @"jump_start-1";
SCNScene *jumpStartAnim = [jumpStartScene animationSceneForKey:jumpId];

// add the jump animation to the explorer scene
[scene.modelScene addAnimation:jumpStartAnim];

// retrieve the SCNView
SCNView *scnView = (SCNView *)self.view;

// set the model scene to the view
scnView.scene = scene.modelScene;
```

3.2.3 Adding an animation to a node

You can also add an animation to a node, using the `SCNNode(AssimpImport)` category.

Listing I-5: Load and play an animation added to SCNNode:

```
#import <AssimpKit/PostProcessing.h>
#import <AssimpKit/SCNScene+AssimpImport.h>

// Some node somewhere to which you add the animation
```

```

SCNNode *targetNode = ...

// load an animation which is defined in a separate file
NSString *jumpAnim = @"/explorer/jump_start.dae";
SCNAssimpScene *jumpStartScene =
    [SCNAssimpScene assimpSceneWithURL:[NSURL URLWithString:jumpAnim]
     postProcessFlags:AssimpKit_Process_FlipUVs |
                     AssimpKit_Process_Triangulate];

// get the animation with animation key
NSString *jumpId = @"jump_start-1";
SCNScene *jumpStartAnim = [jumpStartScene animationSceneForKey:jumpId];

// add the jump animation to the explorer scene
[targetNode addAnimation:jumpStartAnim];

```

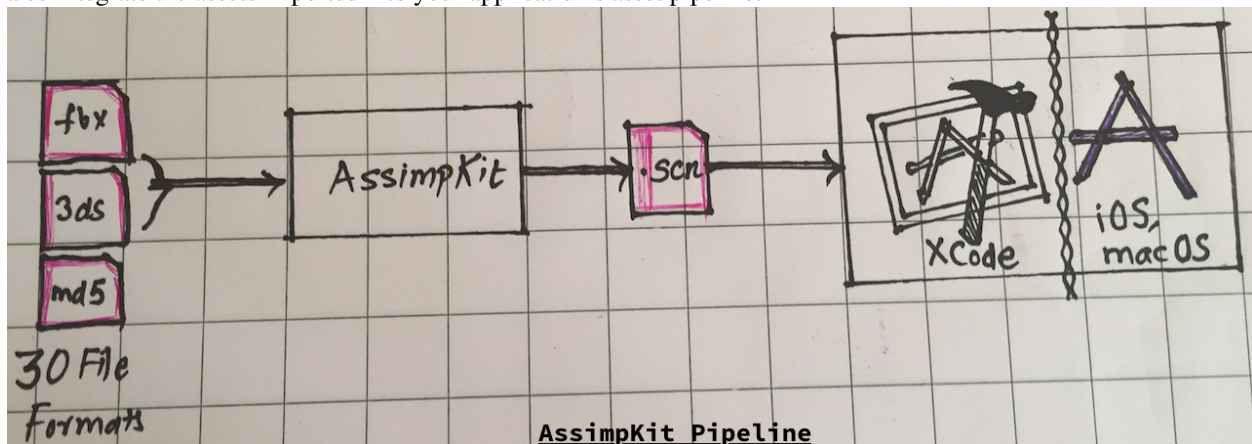
3.2.4 Removing Animations

You can use the `removeAllAnimations` method defined in `SCNAnimatable` to remove all animations attached to the object, using AssimpKit.

3.3 Serialization and integrating with asset pipeline

You can serialize the model and animation scenes in `SCNAssimpScene` using the `write` defined in `SCNScene` to export to either `.scn` or `.dae` file. See the discussion section of `write` for more details.

By exporting using the above serialization method, you can both edit the exported assets in XCode's scene editor and also integrate the assets imported into your application's asset pipeline.



3.4 Using .scn archives exported from AssimpKit in your app

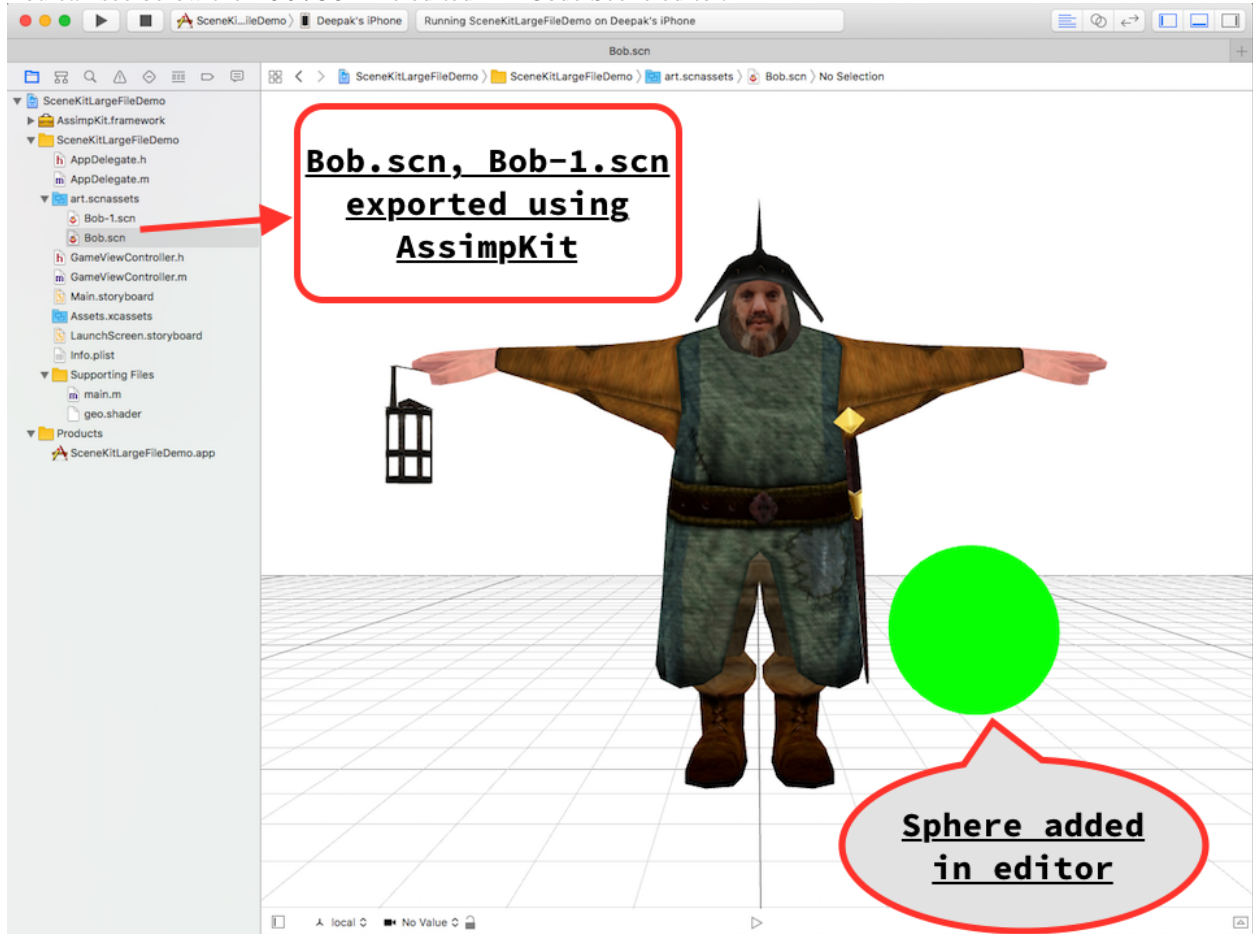
Assuming you have two files in the Quake `.md5` format, `Bob.md5mesh` which contains the 3D model data and `Bob.md5anim` which contains a skeletal animation. Using the API as explained above, you can load both the model `SCNScene` and animation `SCNScene` and then export these to the native `.scn` archive format.

Assume `Bob.md5mesh` is exported to `Bob.scn` and `Bob.md5anim` is exported to `Bob-1.scn`, then in some iOS/macOS app, you can load these and play the animation as such.:

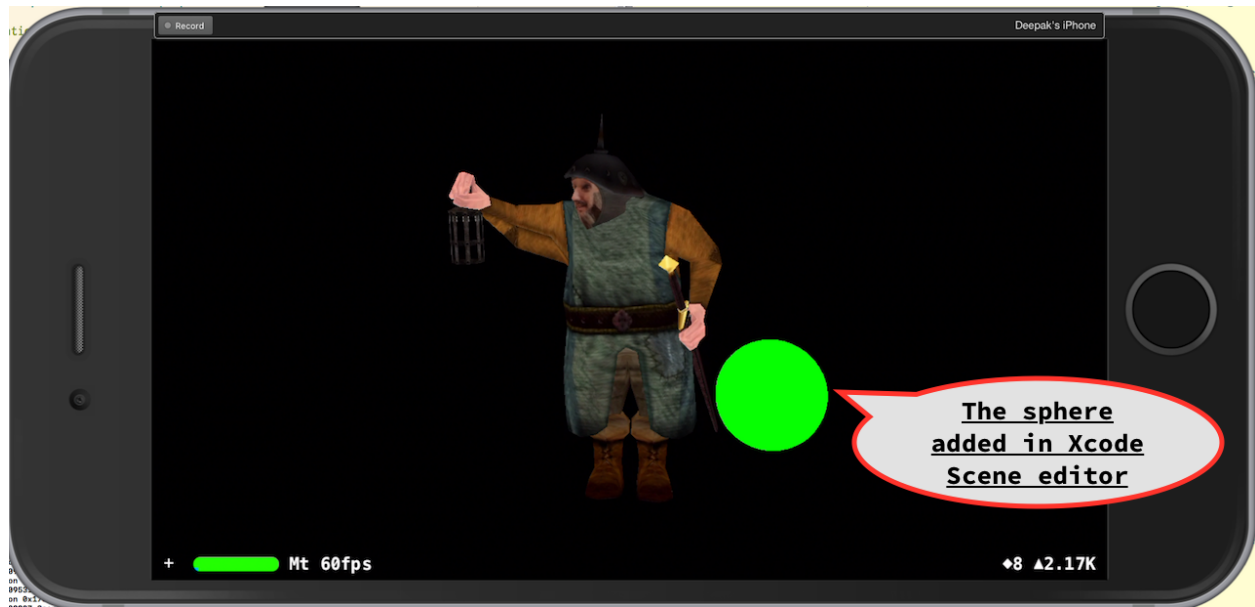
```
#import <AssimpKit/SCNScene+AssimpImport.h>

SCNScene *scene = [SCNScene sceneNamed:@"art.scnassets/Bob.scn"];
SCNScene *animScene = [SCNScene sceneNamed:@"art.scnassets/Bob-1.scn"];
[scene addAnimationScene:animScene];
```

You can see below the Bob.scn file edited in XCode Scene editor.



The edited Bob.scn with animation rendered.



Example Apps

The library source code on [GitHub repo](#) ships with two example apps, for iOS and macOS platforms, which demonstrate the use of this API.

4.1 Build Instructions for the example apps

- Checkout the source code from [GitHub repo](#).
- Open the `iOS-Example.xcodeproj` for the iOS example app.
- Open the `OSX-Example.xcodeproj` for the macOS example app.
- Clean, Build in XCode.

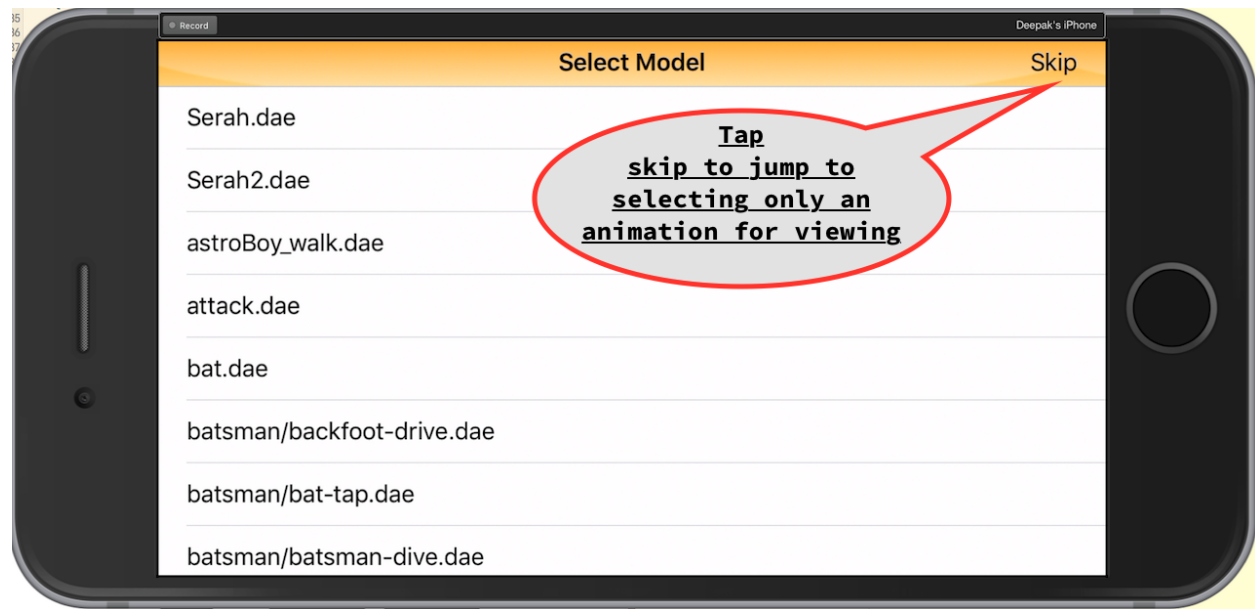
4.2 About the iOS example app

This example app has `Application supports iTunes file sharing` property enabled in its `plist` file, which allows you to use iTunes to upload your 3D models to your device.

Once you have uploaded the 3D models, you can view the models as such.

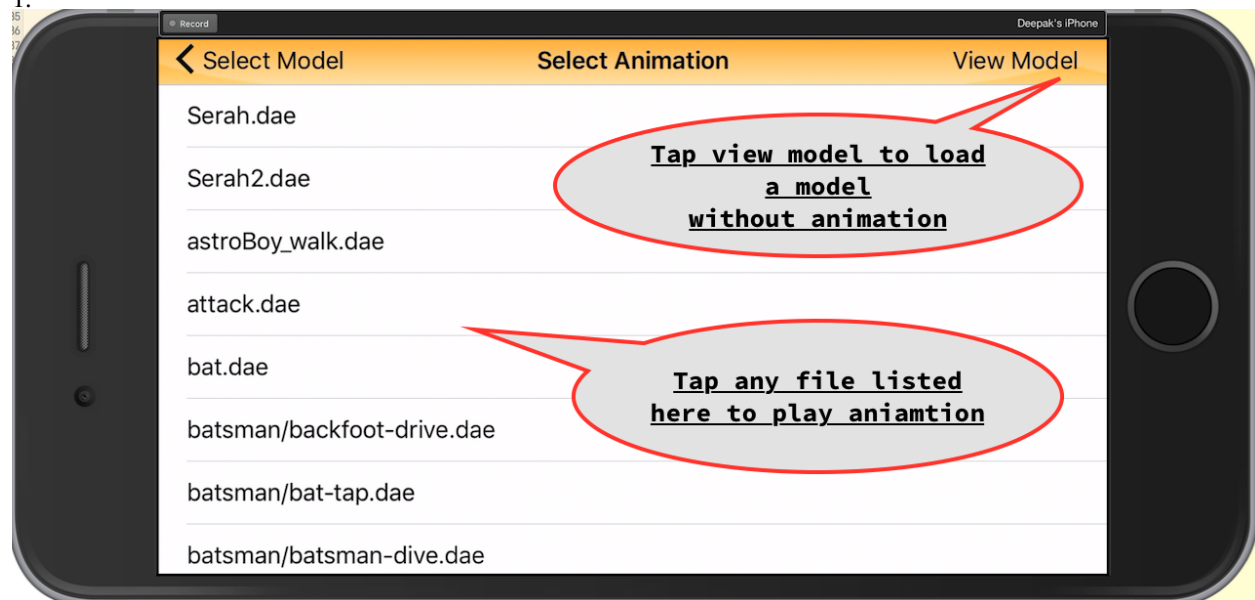
4.2.1 Step 1

You can pick the model to view. You can also skip picking a model and navigate to picking only a 3D skeletal animation.



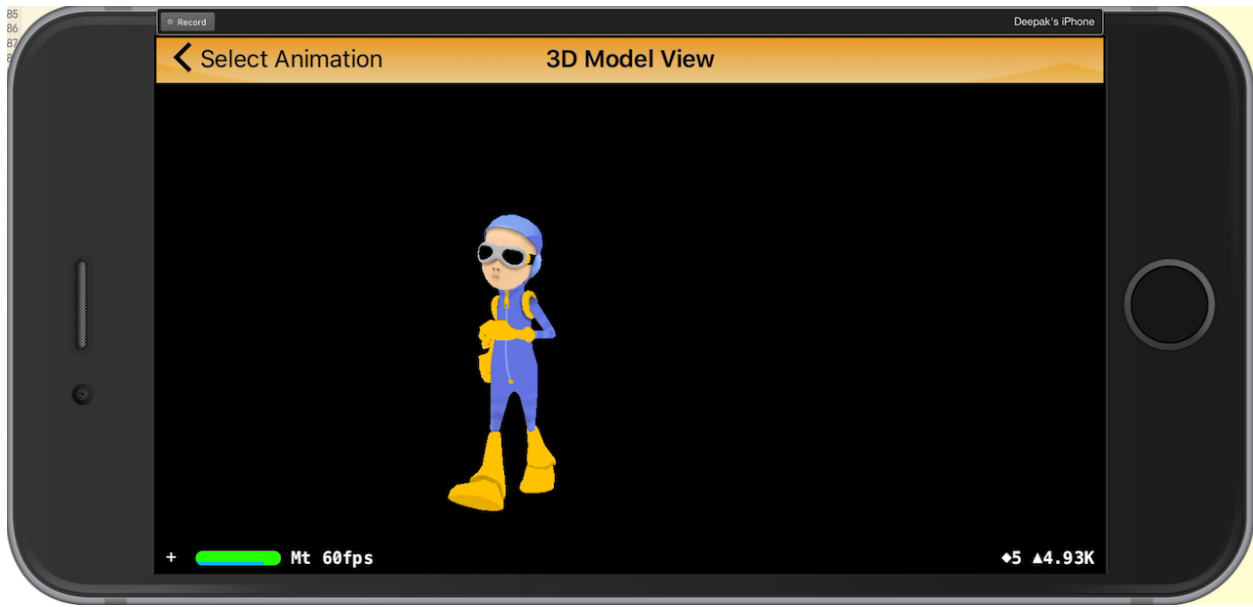
4.2.2 Step 2

You can pick a 3D animation in this step. You can skip this step if you just want to view the 3D model selected in step 1.



4.2.3 Step 3

Based on your selection, you can view only the model, the skeltal animation or both.

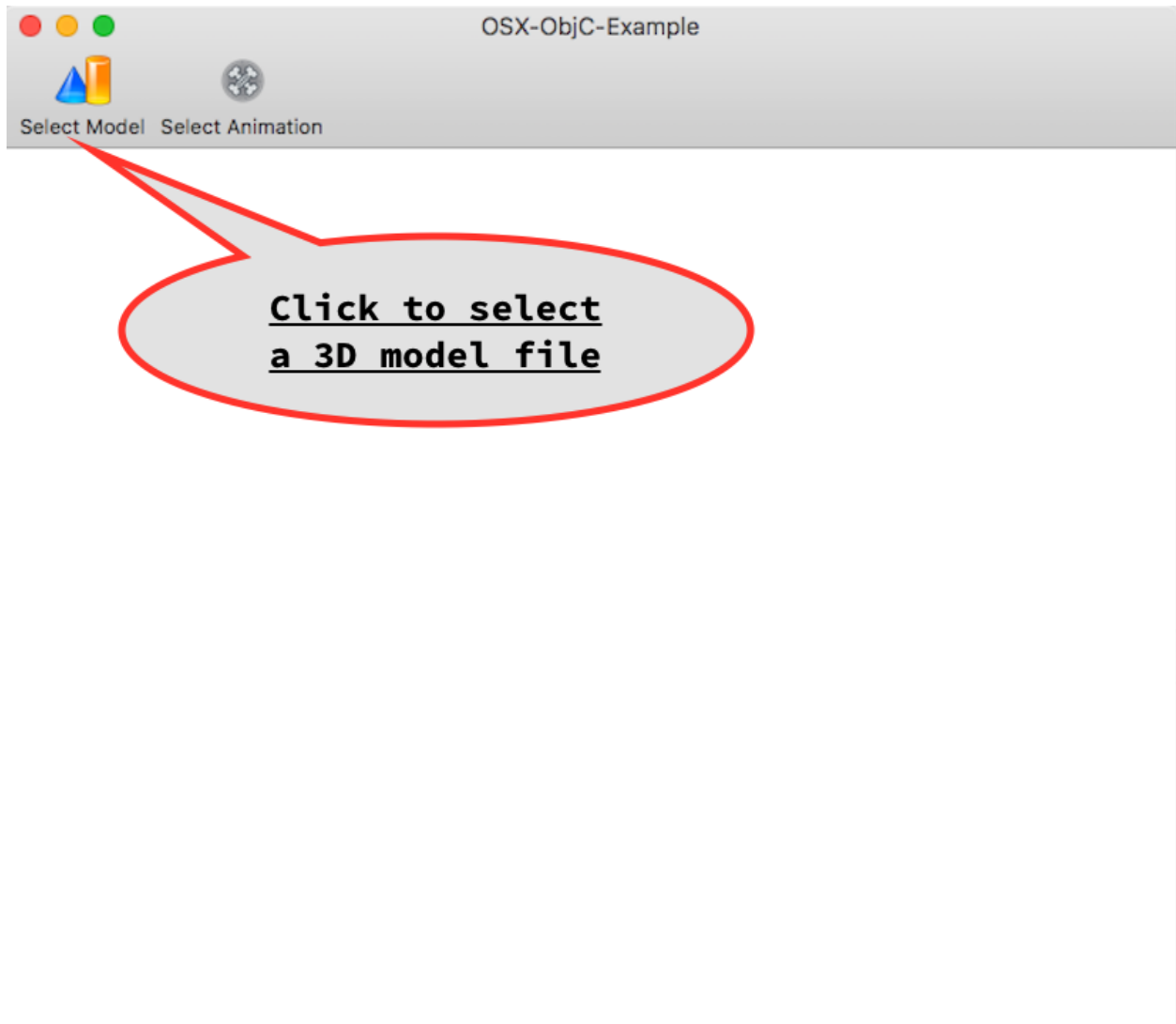


4.3 About the macOS example app

This example app allows you to select the 3D model and animation files using the file picker.

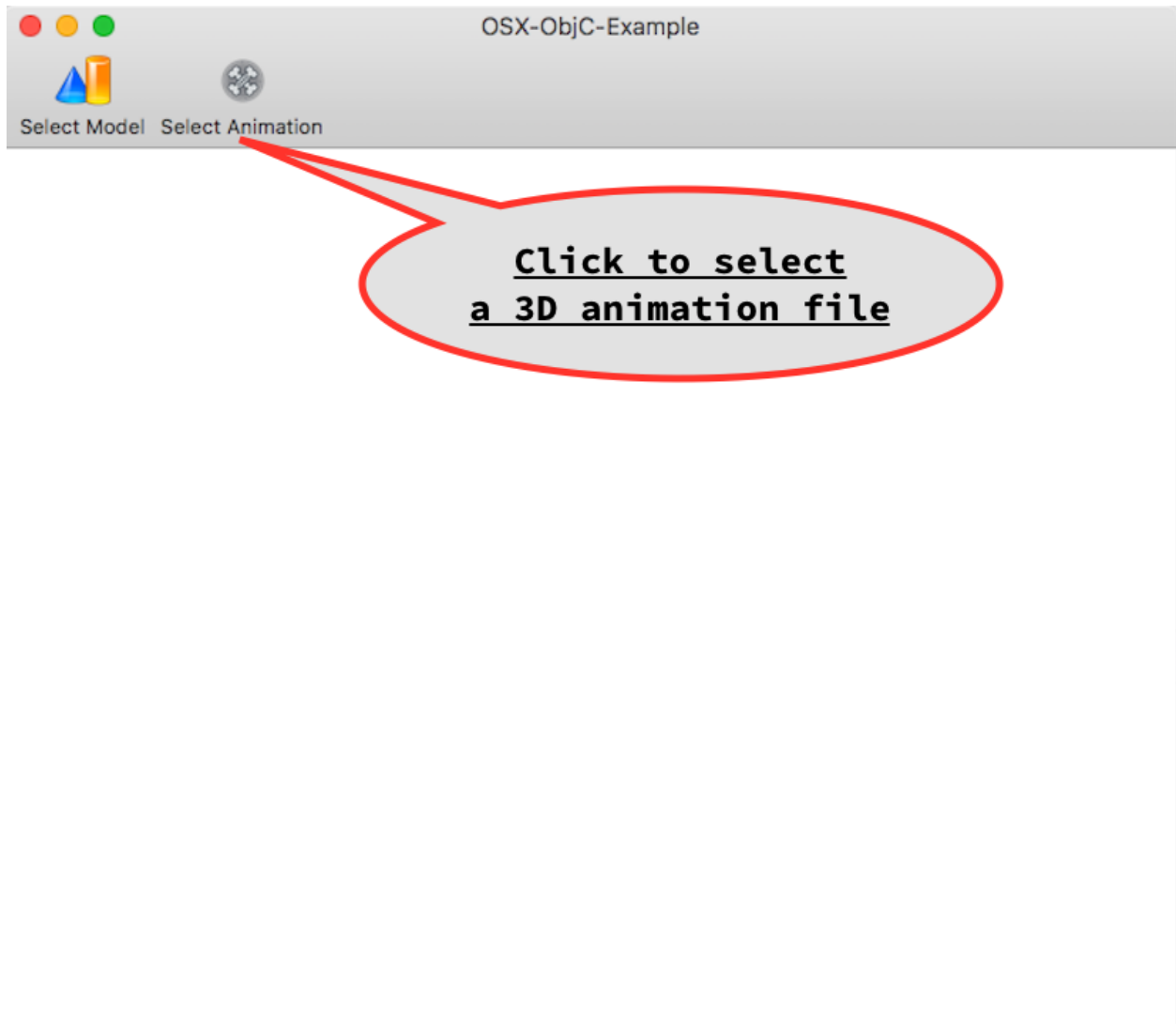
4.3.1 Step 1

You can pick the model to view. You can also skip picking a model and instead pick only a 3D skeletal animation.



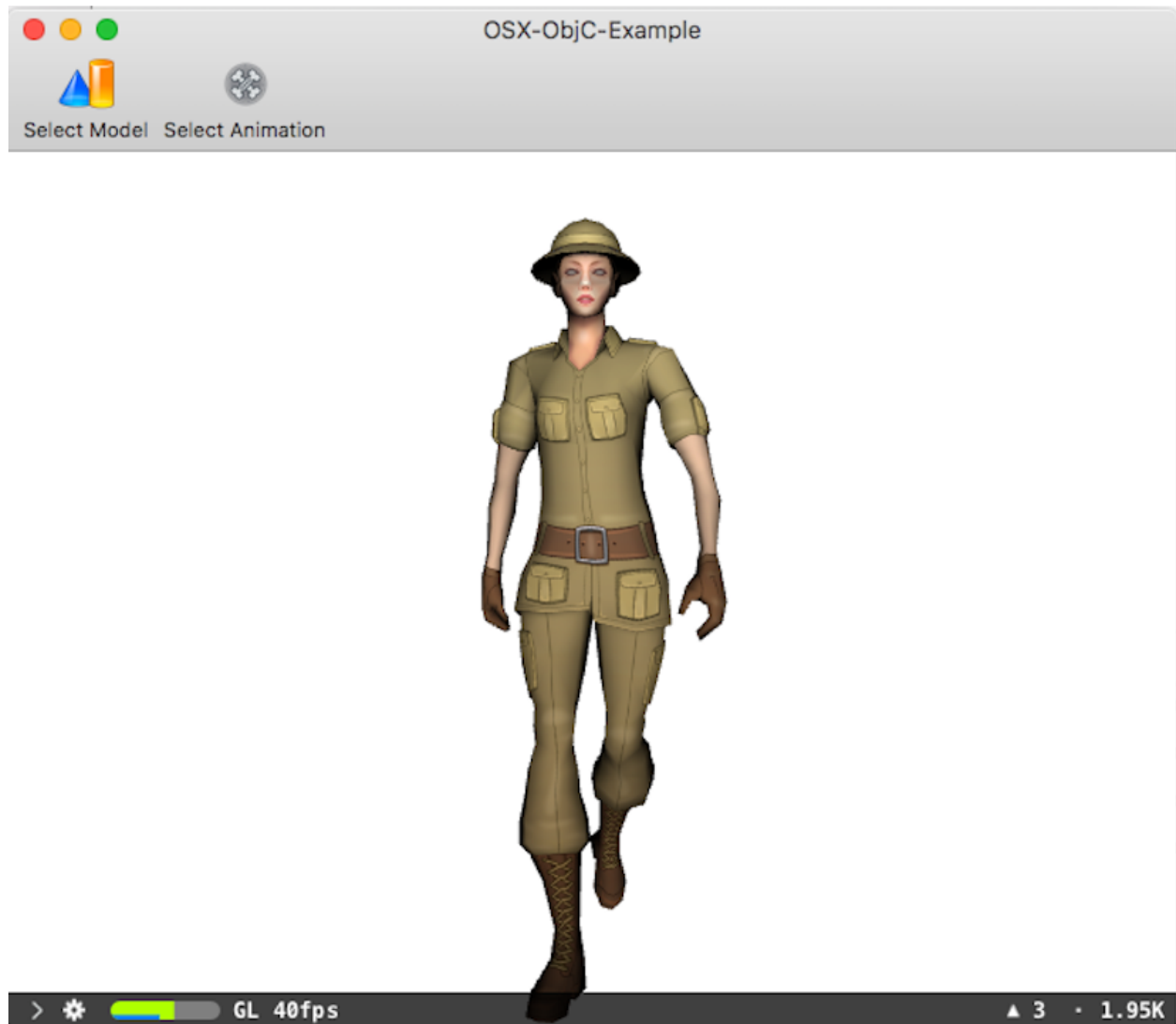
4.3.2 Step 2

You can pick a 3D animation in this step. You can skip this step if you just want to view the 3D model selected in step 1.



4.3.3 Step 3

Based on your selection, you can view only the model, the skeltal animation or both.



FAQ

1. Which are the file formats supported?
 1. See *File formats supported*.
2. Why AssimpKit?
 1. See *Why AssimpKit*.
3. What is the preferred use case for using AssimpKit?
 1. The preferred use case is when you are using [SceneKit](#) and want to use files that are not supported by [SceneKit](#) or [Model I/O](#). If your target app is a real time app such as a game, it is recommended that you export the scenes generated by AssimpKit into a native SceneKit archive and then use these archives. See *Using .scn archives exported from AssimpKit in your app*.

Installation

This library has a simple 2 step installation process:

- Fork the [repository](#).
- Clone your forked repository.

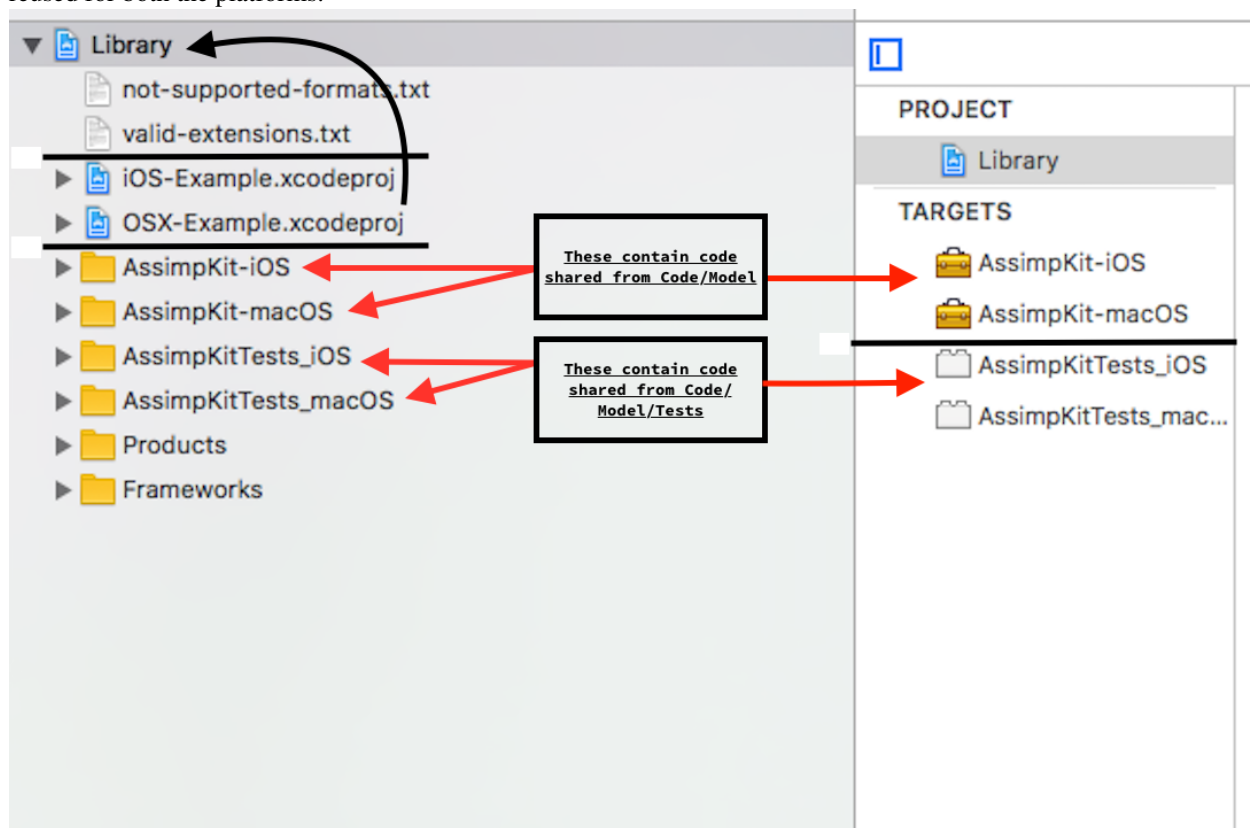
6.1 Assimp Dependency

As AssimpKit depends on [Assimp](#), AssimpKit ships with the fat libraries for both iOS and macOS platform. The xcodeproj is also configured for Assimp dependency usage, so you only need to checkout and start editing!!!

The repository also contains `assets` which have their own licenses and are meant to be used only for testing. Please refer to the [License](#) and [Licenses](#) for more information.

XCode Project Layout

The XCode project is laid out such that the common cross platform code related to reading using Assimp and transforming it to a Scene Kit scene graph is reused for both the iOS and macOS platforms. The testing code is similarly reused for both the platforms.



7.1 Common Code

The common code, including both sources and tests, is placed under `Code/Model`. This common code is then used in 4 targets, 2 of which ship the iOS and macOS frameworks while the other two are testing targets.

7.2 Targets

The project contains the following 4 targets.

7.2.1 AssimpKit-iOS

This target builds the `AssimpKit.framework` for the iOS platform using the common code.

7.2.2 AssimpKit-macOS

This target builds the `AssimpKit.framework` for the macOS platform using the common code.

7.2.3 AssimpKitTests_iOS

This target tests the common code using the test `assets` for the iOS platform.

7.2.4 AssimpKitTests_macOS

This target tests the common code using the test `assets` for the macOS platform.

7.3 Example Apps

The library also contains 2 example apps `iOS-Example.xcodeproj` and `OSX-Example.xcodeproj` which are configured for the `AssimpKit.framework` dependency. You can read more about the `:ref:example-apps` here.

7.3.1 Changing Code

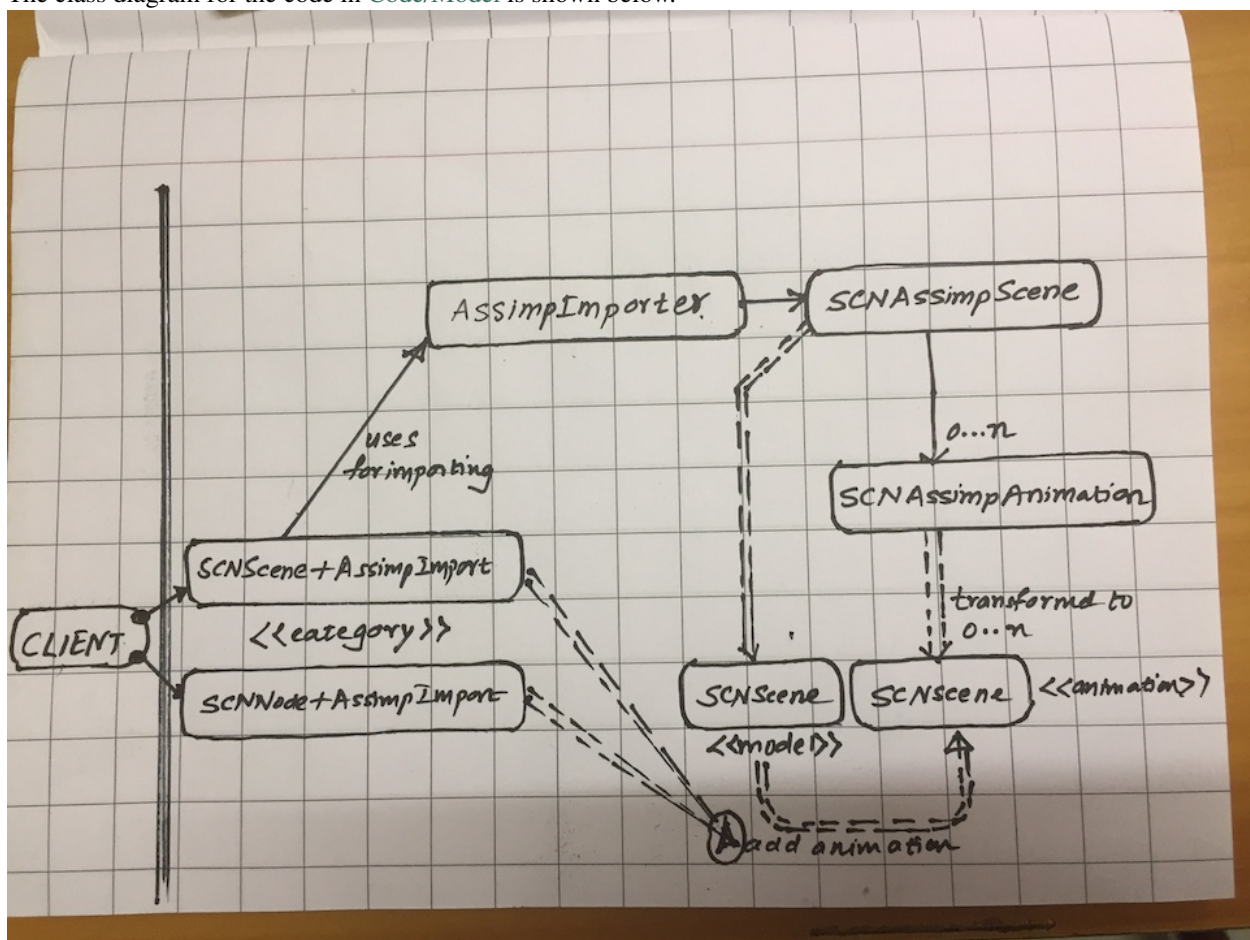
Any code change either for fixing a bug or adding a new feature, should ideally result in updates to the test code as well as example apps.

Design

The design of this library is based on the transformation of the [Assimp](#) scene graph to a [Scene Kit](#) scene graph.

8.1 Classes

The class diagram for the code in [Code/Model](#) is shown below.



[AssimpImporter](#) is the most important class which transforms the assimp scene graph to the scene kit scene graph. It does the transformation by doing a depth first traversal of the assimp scene graph and for each [assimp node](#) visited, it generates a [scene kit node](#).

`SCNAssimpScene` contains all the transformed data, excluding the animation data, for which `SCNAssimpAnimation` is the container. The `SCNAssimpScene` generates the model `SCNScene` and animation `SCNScene` instances. The `SCNScene+AssimpImport` category also contains method to add the animation data in `SCNScene` to another `SCNScene` or `'SCNNode'`

8.2 Generating the scene kit scene graph

The scene kit scene graph is generated in a 3 pass process:

- Pass 1: Generate the scene graph with geometry, materials and camera. In this pass, we also collect the bone names if the `aiMesh` has bones.
- Pass 2: This pass is executed only if the file has skeletal animation data, in which case, we infer all the skeleton info so that we can make a `SCNSkinner` and then generate the animation data using `CAAnimation` objects.
- Pass 3: Finally we transform the generated scene graph to a `SCNScene` and each animation generated in pass 2 to `SCNScene`. This transformation is important as it makes it easy to both serialize to native `.scn` format and integrate into asset pipelines and/or applications.

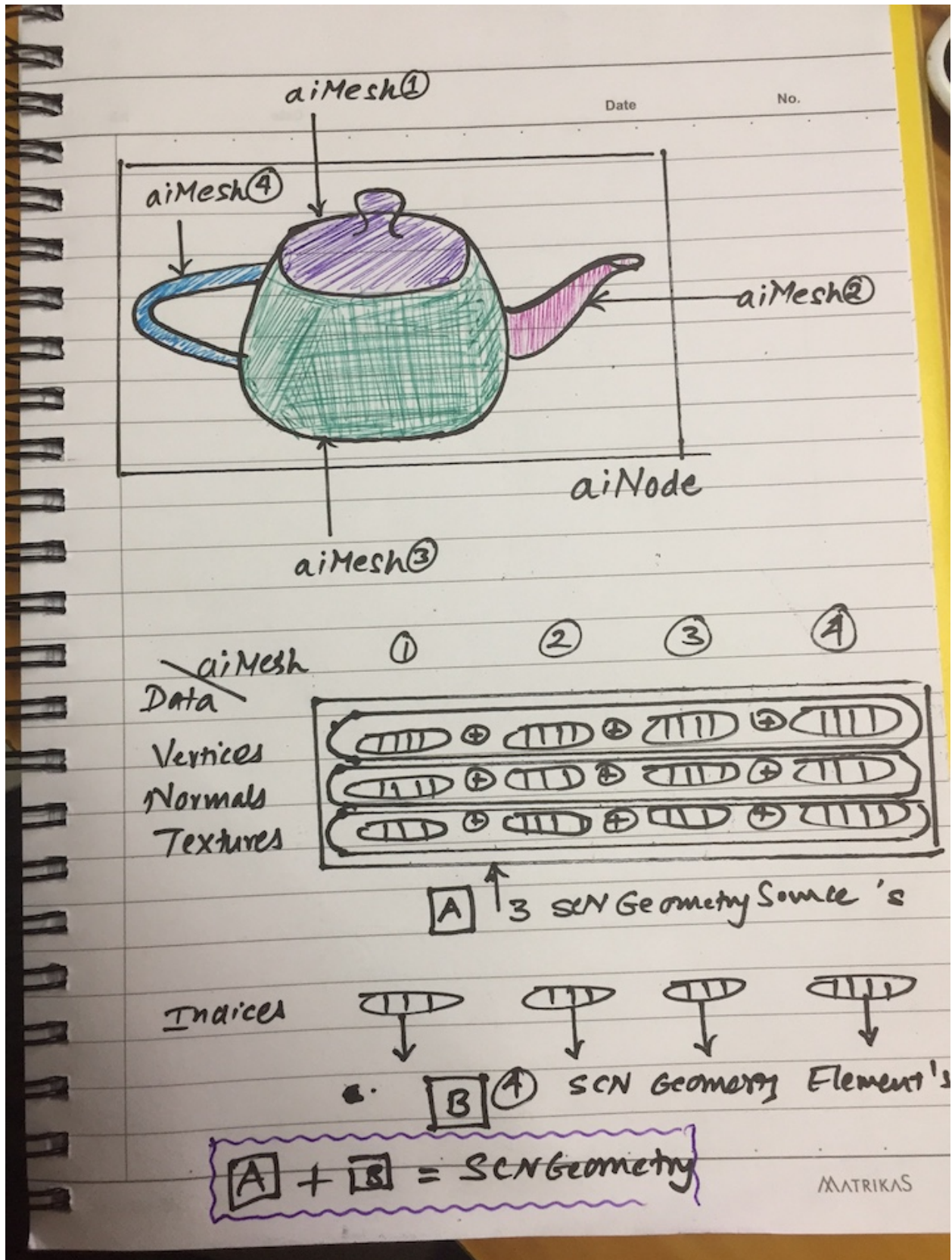
Each pass is further described in detail next.

8.3 Pass 1: Generating the graph with geometry

8.3.1 Generating Geometry

The assimp node can contain multiple meshes where each mesh maps to the `SCNGeometryElement`. The importer generates a single `SCNGeometrySource` for each of vertex, normal and texture data for all the meshes in the nodes. Next it generates separate `SCNGeometryElement` for each mesh in the data, ensuring the vertex indices are offset correctly for the combined geometry source.

A visual representation of this transformation is as shown.



8.3.2 Generating Materials

As seen in the visual for geometry transformation, the importer now generates a material for each mesh in the node. The material in assimp maps to `SCNMaterial` in scene kit. The importer generates a texture if available for the following material properties: diffuse, specular, ambient, reflective, opacity, normals and light map. The material property in assimp maps to `SCNMaterialProperty`.

8.3.3 Generating Cameras

The camera in assimp maps to `SCNCamera` in scene kit. For an assimp node with a camera, the importer generates a node with a camera which has `xFov`, `zNear`, `zFar` configured.

8.3.4 Generating Lights

As of now the generation of lights has been disabled, due to a problem in serialization of light nodes in scene kit. See [Issue #46](#).

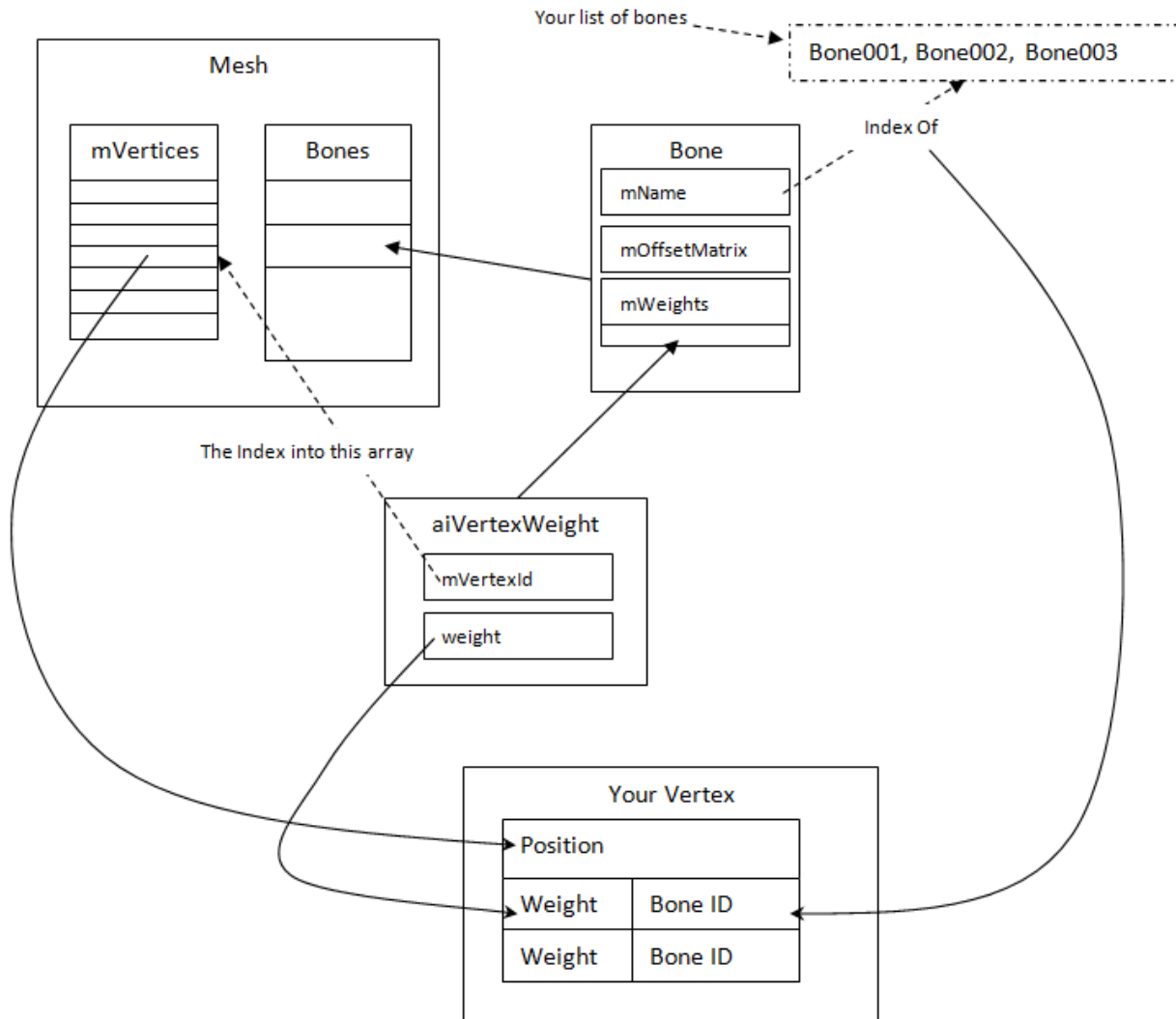
8.4 Pass 2: Generating Skeletal Animations

The skeletal animation data is generated in a 3 step process which consists of:

- Making a skeleton database
- Making a scene kit skinner
- Making the core animation objects

8.4.1 Making a skeleton database

The assimp scene graph does not contain a unique list of bones or the root of the skeleton which have to be inferred from the assimp data structures.

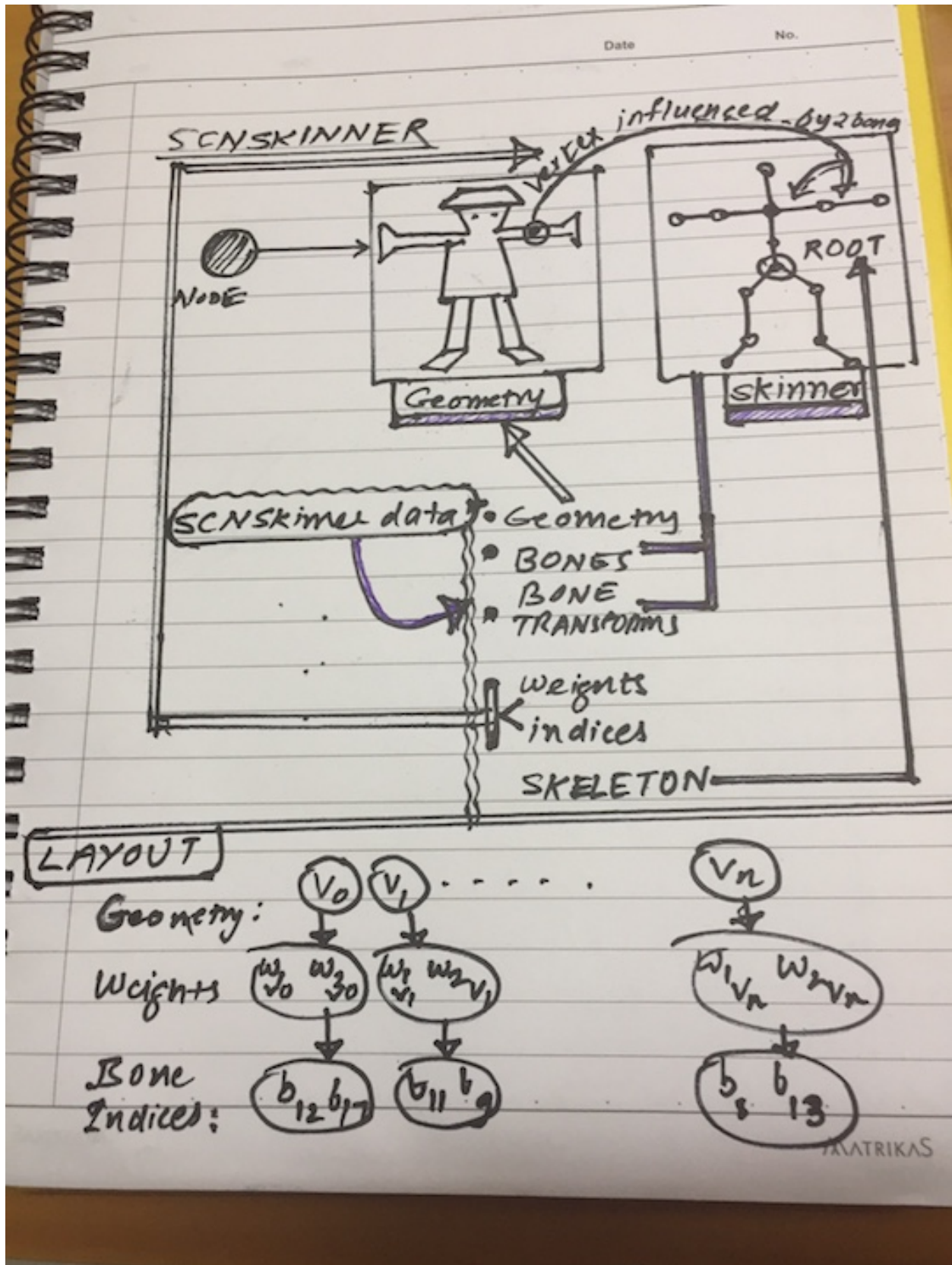


We parse the data structures above, so that we have a list of unique bone names, bone nodes and the bone inverse transforms. Once the unique bone nodes is known, the importer determines the root of the skeleton as that node which has the lowest depth from the parent!

8.4.2 Making a skinner

In order to make a skinner, we also need the vertex weights data in addition to the bone nodes and their inverse bind transforms which are available from [Making a skeleton database](#).

Assuming each vertex is influenced by 2 weights, the scene kit skinner data layout is as such.



The importer first finds the number of vertices at the node and the maximum weights. As a node may contain multiple

meshes, the weights information is generated for the combined meshes at that node and if a given node has less weights than the maximum weights, zero weights are added for the remaining weights.

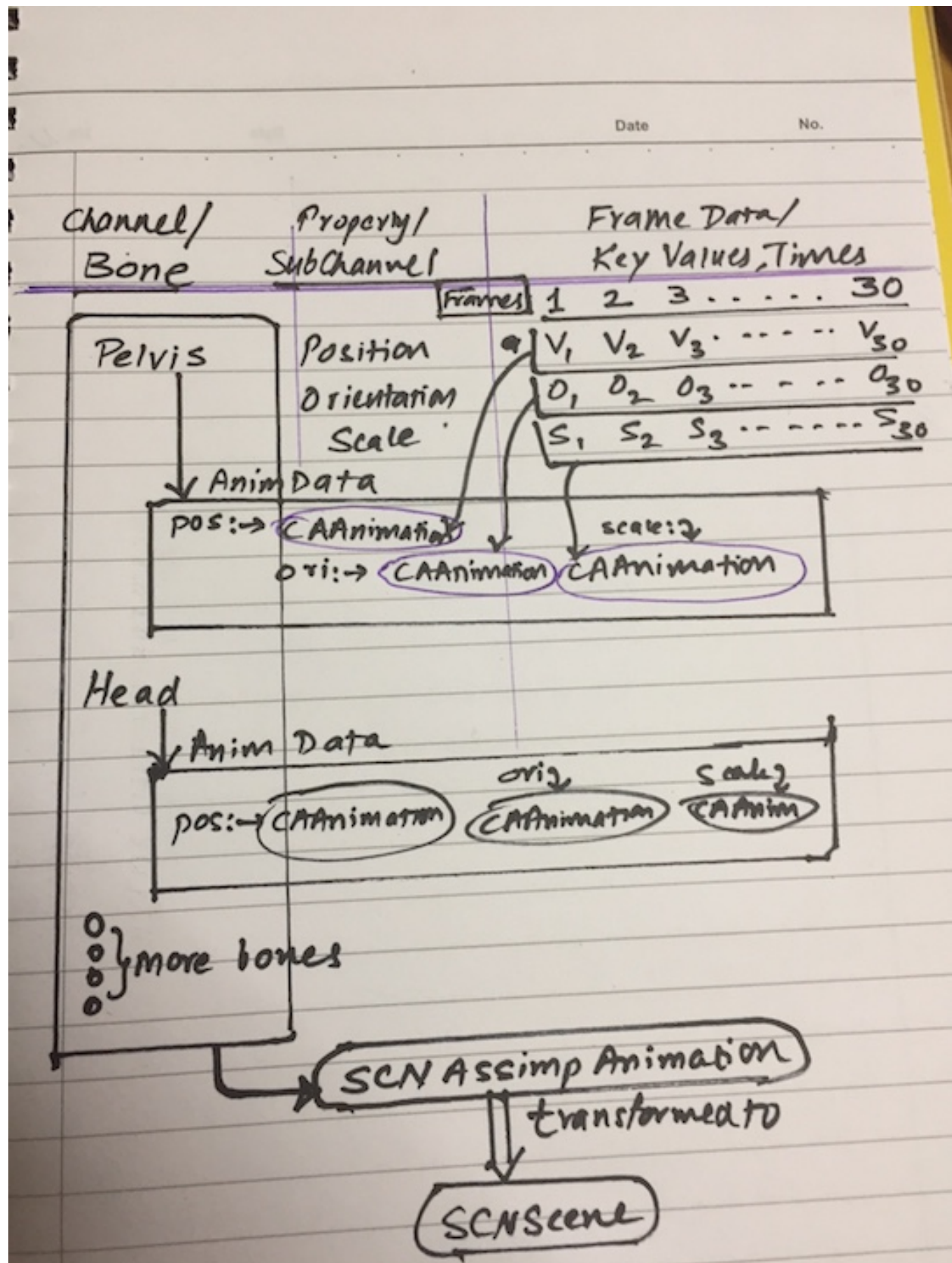
In assimp, each mesh's bones have vertex weights from which we have to calculate the inverse data of which vertices are influenced by which bones.

When calculating the bone indices for the corresponding bone weights, we pass the unique array of bone names which we will use when constructing the skinner so that the bone indices are as per skinner's bone indices layout. Again here, we translate from the assimp bone name to the index in the array of bone names generated when making the skeleton database.

If you combine the visuals of the assimp data structures and map them to the [SCNSkinner](#), and understand the skeletal animation concept of vertex deformation using bone weights, then the above will be easier to understand.

8.4.3 Making the animations

The animation data is stored in [aiAnimation](#) as shown.



Each channel represents a bone and contains the keys for position, orientation and scale. The position, orientation and

scale keys are then converted into a `CAAnimation` object. Each position and scale key value is represented by a `SCNVector3` while the orientation is represented by a `SCNVector4_` which is a quaternion. These core animation objects are stored in a dictionary keyed by `position`, `orientation`, `scale`, along with a generated animation name, gives us a `SCNAssimpAnimation` object.

If we have multiple animations in a file, we end up with multiple `SCNAssimpAnimation` instances.

At the end of pass 2, we end up with `SCNAssimpScene` instance with `SCNAssimpAnimation` objects if animation data exists.

8.5 Pass 3: Generating native SCNScene instances

The `SCNAssimpScene` instance is now transformed into a `SCNScene` instance. Each `SCNAssimpAnimation` instance is transformed into a `SCNScene` instance. By transforming these to `SCNScene` instances, both serialization and integration into existing asset pipelines and/or applications becomes trivial.

8.6 Loading Animations

The `SCNScene+AssimpImport` and `SCNNode+AssimpImport` category defines a method to add the animation data to a scene or node. As all the animation data is just `CAAnimation` objects, the animation `SCNScene` graph is traversed and the core animation objects are added to the corresponding bone node in the target scene or target nodes' subtree.

Contributing

To contribute to AssimpKit:

- Please open an issue describing the bug, enhancement or any other improvement.
- If valid, please supply the sample model file that can help demonstrate the issue.
- If the design involves a larger refactor, please open a issue to discuss the refactor.
- After discussion on the issue, you can submit a Pull Request by forking this project.
- Please accompany your Pull Request with updates to test code and example apps, the latter may not be required for every change.
- Please ensure you format the code using `clang-format`, there exists a `.clang-format` config for this project.

API

You can find the [API Docs](#) here.

Release Notes

11.1 Version 1.0

- Supports: geometry, materials, cameras and skeletal animations.
- Supports: serialization to native .scn format.
- Production Ready.