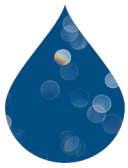
AssetKit Documentation

Release 0.3.2

Recep Aslantas

Aug 01, 2022

Getting Started:



AssetKit

COLLADA glTF .obj .stl .ply

generate normals bounding box flexible simple

triangulate convert UP axis Physically Based

morph utils skin utils small and fast

... more ...

AssetKit is brand-new 2D/3D asset importer, exporter and util library that is written in C language. C++ wrappers or other language bindings can be written in the future. This library will include common 3D util funcs.

- [x] COLLADA 1.4 and COLLADA 1.4.1
- [x] COLLADA 1.5
- [x] glTF 2.0 (Embedded or Separated (.gltf), Binary (.glb), Extensions...)
- [x] Wavefront Obj (.obj + .mtl)
- [x] STL (ASCII, Binary)
- [x] PLY (ASCII, Binary)
- [ ] USD and friends (License?)
- [ ] Alembic (License?)
- [ ] Draco
- [ ] X3D
- [x] in progress for next...

CHAPTER 1

Features

- Very very small, very fast and flexible library
- Single interface for glTF 2.0 (with extensions), COLLADA 1.4/1.4.1/1.5, Wavefront Obj and others...
- Javascript-like API to get URL or ID *obj = ak_getObjectById(doc, objectId)*...
- Options to Generate Mesh Normals *(Default: enabled)*
- Option to Triangulate Polygons *(Default: enabled)*
- Option to change Coordinate System *(Default: enabled)*
- Option to calculate Bounding Boxes *(Default: enabled)*
- Unique and Flexible Coordinate System * Support multiple coordinate system * Can convert any coordinate system to another with adding transform or with changing transform, vertex data...
- **Unique and Flexible Memory Management System**
  - Hierarchical unique memory management
  - When a node is freed then all sub memories will be freed
  - COLLADA's **sid** and **ID** values are mapped to memory nodes itself to reduce memory size and make it easy to manage things.
  - Allow attach ID, sid or user data to a memory node
- Object-based Asset support; resolve asset element for any element
- Bugfix some DAE files
- Will be optimized to be fastest, smallest and most flexible, extendible Asset loader.
- and others...

1.1 Build AssetKit

AssetKit does not have any external dependencies.

1.1.1 CMake (All platforms):

```
$ mkdir build
$ cd build
$ cmake .. # [Optional] -DAK_SHARED=ON
$ make
$ sudo make install # [Optional]
```

make will build **AssetKit** into **build** folder. If you don't want to install **AssetKit** to your system's folder you can get static and dynamic libs in this folder.

CMake Options:

```
option(AK_SHARED "Shared build" ON)
option(AK_STATIC "Static build" OFF)
option(AK_USE_C99 "" OFF) # C11
option(AK_USE_TEST "Enable Tests" OFF) # for make check - make test
```

Use with your CMake project example

```
cmake_minimum_required(VERSION 3.8.2)

project(<Your Project Name>)

add_executable(${PROJECT_NAME} src/main.c)
target_link_libraries(${LIBRARY_NAME} PRIVATE assetkit)

add_subdirectory(external/assetkit/)
```

1.1.2 Unix (Autotools):

```
$ sh autogen.sh
$ ./configure
$ make
$ make check            # run tests (optional)
$ [sudo] make install   # install to system (optional)
```

make will build **AssetKit** to **.libs** sub folder in project folder. If you don't want to install **AssetKit** to your system's folder you can get static and dynamic libs in this folder.

NOTE:: Change install name if required; after build is finished, **make** automatically runs *sh ./post-build.sh* script. It changes install names. You may want to edit build scripts and *post-build.sh* script if you want to build AssetKit with existing libraries. Default behavior is that AssetKit will look up sub libraries inside *.libs* folder, if you only need to change *.libs* name then change it in *post-build.sh* script file.

1.1.3 Windows (MSBuild):

Windows related build files, project files are located in *win* folder, make sure you are inside in assetkit/win folder.

Code Analysis are enabled, it may take awhile to build.

```
$ cd win
$ .\build.bat
```

if *msbuild* is not worked (because of multi versions of Visual Studio) then try to build with *devenv*:

```
$ devenv assetkit.sln /Build Release
```

Currently tests are not available on Windows.

1.1.4 Documentation (Sphinx):

AssetKit uses sphinx framework for documentation, it allows lot of formats for documentation. To see all options see sphinx build page:

https://www.sphinx-doc.org/en/master/man/sphinx-build.html

Example build:

```
$ cd assetkit/docs
$ sphinx-build source build
```

1.2 Getting Started

There are lot of file formats out there. It is not easy to implement each of them individually in an engine or software.

AssetKit provides single extensible interface for all file formats that is supported. **AssetKit** tries to full support all major file formats.

1.3 Quick Implementation

Assuming you already followed Build instructions and Getting Started sections. Also assuming you already linked **AssetKit** to your project and set include paths.

1.3.1 1. Include

AssetKit uses **ak** prefix for all functions and type names. To include **AssetKit**

```
#include <ak/assetkit.h>

/* other headers */
#include <ak/options.h>
...
```

1.3.2 2. Preparing

You may want to prepare loader before call `ak_load()` load function.

a. Setting Image loader

There was image loader inside **AssetKit** but it has been dropped to make it more generic. Becasue you may already have an image loader e.g. stb_image ...

AssetKit can trigger images and cache them for you, if it is already loaded than it will return loaded contents.

You need to set loader as below. The example used stb_image but you mat use another image loader...

```
void*
imageLoadFromFile(const char * __restrict path,
                  int        * __restrict width,
                  int        * __restrict height,
                  int        * __restrict components) {
  return stbi_load(path, width, height, components, 0);
}

void*
imageLoadFromMemory(const char * __restrict data,
                    size_t                  len,
                    int        * __restrict width,
                    int        * __restrict height,
                    int        * __restrict components) {
  return stbi_load_from_memory((stbi_uc const*)data, (int)len, width, height,␣
↪components, 0);
}

void
imageFlipVerticallyOnLoad(bool flip) {
  stbi_set_flip_vertically_on_load(flip);
}


/* Call this before loading document e.g. ak_load() or images */
ak_imageInitLoader(imageLoadFromFile, imageLoadFromMemory, imageFlipVerticallyOnLoad);
```

Just ensure that you set image loader before loading images. It is good to set it once before `ak_load()`.

b. Setting Options if Needed

Make sure that you set UP axis if you want to different one from **Y_UP**. For instance if you want to load contents as **Z_UP**

```
ak_opt_set(AK_OPT_COORD, (uintptr_t)AK_ZUP);
```

see options in the documentation. If the default values don't work for you then just change them before `ak_load()`.

1.3.3 3. Load Document

Use `ak_load()` to load a 3D file. It returns `AkDoc` which contains everything you need. You must check the result/return, it must be AK_OK otherwise, document is not loaded or falied at some point.

```
AkDoc   *doc;
AkResult ret;

if ((ret = ak_load(&doc, "[Path to a file e.g ./sample.gltf]", NULL) != AK_OK) {
   printf("Document couldn't be loaded");
   return;
}
```

or

```
AkDoc   *doc;
AkResult ret;

ret = ak_load(&doc, "[Path to a file e.g ./sample.gltf]", NULL);
if (ret != AK_OK) {
   printf("Document couldn't be loaded");
   return;
}
```

doc is passed as reference, if the result is success than the document will be set that reference parameter.

AssetKit will try to load referenced textures, images, binary files... so you must only pass original file, not folder.

Now you loaded the document you want. See next step.

There are two ways to load geometries from loaded document.

a. Load scene[s], nodes than load referenced geometries

b. Load all geometries in the document

The second way may cause to load unused geometries, because a geometry may not be referenced in scenes. It is better to follow scene > node > instance geometry > geometry path.

1.3.4 4. Load Scene[s]

AssetKit can load scenes, nodes, geometries and so on. If the file you loaded doesn't support scenes e.g Wavefront Obj. AssetKit creates a default scene for that file formats and adds reference of geometries to that scene.

There are **scene library** and **scene** in AssetKit **document**. The **scene** is the active scene for rendering, it references a scene from library.

```
AkInstanceBase *instScene;
AkVisualScene  *scene;

if ((instScene = doc->scene.visualScene)) {
  scene = (AkVisualScene *)ak_instanceObject(doc->scene.visualScene);
}
```

scene.visualScene is instance reference (`AkInstanceBase`), any scene may be instanced with this link/object. Another instance objects may have different types e.g. instance geometry (inherited from `AkInstanceBase`).

We need to get actual scene object from instance object. There are a few helpers for this task. But we will use `ak_instanceObject()` here.

1.3.5 5. Load Nodes[s]

After you get a scene, you can iterate through root nodes. There are also nodes in NodeLibrary in document but in this way you only get used nodes.

There are a few elements in nodes

- Node Transform
- One or more instance geometries
- One or more instance cameras

- One or more instance lights
- One or more instance nodes
- One or more child nodes
- Bounding Box as AABB of Node
- ...

5.1 Transforms

You must multiply node's transform with its parent to get transform in WORLD space for each node recursively.

A node can contain Matrix or individual Transform Elements like Rotation, Translation or Scaling. **AssetKit** also provides a util to combine these individual transforms into matrix with `ak_transformCombine()`.

AssetKit does not combines them automatically because they may be referenced to animated individually.

5.2 Instance Geometries

This is the one of critical sections to understand. Nodes uses `AkInstanceGeometry` type to reference a `AkGeometry`.

A `AkInstanceGeometry` object may store these informations:

- Instance to geometry
- Material to bind
- Instance to morpher
- Instance to skinner

5.2.1 Instance to geometry | Loading Geometry

A node may contain multiple geometries, so you must iterate each one and get the geometry with `ak_instanceObject()` function.

After you get the geometry you can load geeometry elements. A `AkGeometry` object can contain one of mesh, spline and brep.

```
AkObject *prim;
AkResult  ret;

/*
   return if the geometry is already loaded,
   you can use a RBTree or HasMap... (see https://github.com/recp/ds)
  */

prim = geom->gdata;
switch ((AkGeometryType)prim->type) {
  case AK_GEOMETRY_MESH:
     /* load mesh */
    ret = loadMesh(...);
    break;
  default:
    ret = AK_ERR;
}
```

(continues on next page)

(continued from previous page)

```
18
19 return ret;
```

Now it is time to load a mesh.

5.2.2 Loading mesh

This tutorial will only cover loading meshes, extra tutorials may be provided in the future for loading curves, nurbs...

A mesh object is packed as `AkObject` inside `AkGeometry`. In previous section you may see that we have a switch control to check whether we have a mesh inside geometry or not.

AssetKit provides unique design to store this kind of objects with `AkObject`. (Think `AkObject` as **Object** class in .NET or **NSObject** in ObjC.) Otherwise we would store additional pointers or inherits Mesh from Geometry and then cast it to mesh. This is another option of course, even **AssetKit** may change to this design in the future if needed. Currently we are not doing this because geometry object is top container.

AssetKit provides a helper to get object from `AkObject` with `ak_objGet()` macro.

We can get `AkMesh` object from `AkGeometry` as

```
1 AkMesh *mesh;
2
3 mesh = ak_objGet(geom->gdata);
```

Now we have mesh object. Let's inspect a mesh type.

A mesh contains one or more primitives (or submeshes) as `AkMeshPrimitive`. Each primitive contains AABB, the mesh also contains an AABB which is sum of all.

A mesh also contains default weights for morph targets but a Node in Scene object can override that.

5.2.2.1 Loading mesh primitives

A mesh primitive may be one of `AkLines`, `AkPolygon` or `AkTriangles`. `AkMeshPrimitive` is base type for all of them. You can cast them to `AkMeshPrimitive` or you can use **.base** member.

1.4 API documentation

1.4.1 version

Header: ak/version.h

AssetKit uses semantic versioning (http://semver.org) which is MAJOR.MINOR.PATCH

AK_VERSION_MAJOR is major number of the version.
AK_VERSION_MINOR is minor number of the version.
AK_VERSION_PATCH is patch number of the version.

every release increases these numbers. You can check existing version by including *ak/version.h*

1.5 Options

Currently **AssetKit** provides global options but in th future document based options may be supported.

1.5.1 Options / Preferences / Settings Design

To make things simplier and easy to manage, **AssetKit** provides options as **key** - **value** pair.

The key always is the `AkOption` enum type. The value's type is `uintptr_t`, so you can pass integers, enums and pointers (by casting to *uintptr_t* e.g. `(uintptr_t)(void *)myPointer`).

Documented global options:

```
typedef enum AkOption {
  AK_OPT_INDICES_DEFAULT             = 0,  /* false    */
  AK_OPT_INDICES_SINGLE_INTERLEAVED  = 1,  /* false    */
  AK_OPT_INDICES_SINGLE_SEPARATE     = 2,  /* false    */
  AK_OPT_INDICES_SINGLE              = 3,  /* false    */
  AK_OPT_NOINDEX_INTERLEAVED         = 4,  /* true     */
  AK_OPT_NOINDEX_SEPARATE            = 5,  /* true     */
  AK_OPT_COORD                       = 6,  /* Y_UP     */
  AK_OPT_DEFAULT_ID_PREFIX           = 7,  /* id-      */
  AK_OPT_COMPUTE_BBOX                = 8,  /* false    */
  AK_OPT_TRIANGULATE                 = 9,  /* true     */
  AK_OPT_GEN_NORMALS_IF_NEEDED       = 10, /* true     */
  AK_OPT_DEFAULT_PROFILE             = 11, /* COMMON   */
  AK_OPT_EFFECT_PROFILE              = 12, /* true     */
  AK_OPT_TECHNIQUE                   = 13, /* "common" */
  AK_OPT_TECHNIQUE_FX                = 14, /* "common" */
  AK_OPT_ZERO_INDEXED_INPUT          = 15, /* false    */
  AK_OPT_IMAGE_LOAD_FLIP_VERTICALLY  = 16, /* true     */
  AK_OPT_ADD_DEFAULT_CAMERA          = 17, /* true     */
  AK_OPT_ADD_DEFAULT_LIGHT           = 18, /* false    */
  AK_OPT_COORD_CONVERT_TYPE          = 19, /* DEFAULT  */
  AK_OPT_BUGFIXES                    = 20, /* TRUE     */
  AK_OPT_GLTF_EXT_SPEC_GLOSS         = 21, /* TRUE     */
  AK_OPT_COMPUTE_EXACT_CENTER        = 22  /* FALSE    */
} AkOption;
```

The comment at right contains default value for that option. All options will be documented below. (**TODO**)

As you can see and imagine, not all options are integer or boolean. For instance to set `Y_UP` for `AK_OPT_COORD` option, you need to cast `Y_UP` opinter to `uintptr_t` type.

Sample options:

```
/* compute bounding box for mesh and for its primitives */
ak_opt_set(AK_OPT_COMPUTE_BBOX, true);

/* triangulate meshes that are not triangles e.g. polygons... */
ak_opt_set(AK_OPT_TRIANGULATE, true);

/* change UP axis to Y UP, see coordinate sys documentation. */
ak_opt_set(AK_OPT_TRIANGULATE, (uintptr_t)AK_YUP);
```

Functions:

1. *ak_opt_set()*

2. *ak_opt_get()*
3. *ak_opt_set_str()*

void **ak_opt_set**(AkOption *option*, uintptr_t *value*)

Sets an option by key and value. Pass integers, booleans or pointers as value. Value needs to be casted to **uintptr_t**. Pass only supported values for a key.

Parameters:

[in] **option** option (see AkOption enum)

[in] **value** option value

uintptr_t **ak_opt_get**(AkOption *option*)

Get value of option as **uintptr_t**. If the option is pointer than you need to cast it to pointer. For instance `(AkCoordSys *)ak_opt_get(AK_OPT_COORD)` or `(void *)ak_opt_get(AK_OPT_COORD)`. If there are no warnings then you don't need to cast result to Boolean or Integers for Boolean/Integer options. For instance `if (ak_opt_get(AK_OPT_TRIANGULATE)) ...`

Parameters:

[in] **option** option (see AkOption enum)

void **ak_opt_set_str**(AkOption *option*, const char **value*)

Similar to *ak_opt_set()* but it accepts null terminated string parameter and it uses `ak_strdup()` to duplicate string to keep it. Then it casts duplicated string to `uintptr_t`, so you can get value anytime with *ak_opt_get()*.

NOTE: When you set new value then the old value will be free-ed. If you need to keep old value then you must duplicate it yourself. Otherwise memory leaks would occoured...

Parameters:

[in] **option** option (see AkOption enum)

[in] **value** option value as null-terminated string

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index

A