Alembic

Dexter Studios R&D
Wanho Choi

Introduction

- Alembic is an **open source** library developed by **SPI** and **ILM**.
- It was announced at **SIGGRAPH 2011**, and has been widely adopted across the industry.
- The Alembic is intended to facilitate **baked (cached) geometry** workflows and **application/platform-independent** geometry **interchange** and **sharing**.
- Alembic is focused on efficiently storing the computed results of complex procedural geometric constructions.
- It is specifically not concerned with storing the complex dependency graph of procedural tools used to create the computed results.

- For example, Alembic will efficiently store the **animated vertex positions** and **animated transforms** that result from an arbitrarily complex animation and simulation process, but will not attempt to store a representation of the network of computations (rigs, basically) which were required to produce the final, animated vertex positions and animated transforms.

- Alembic supports the **common geometric representations used in VFX and animation industries**, including polygon meshes, subdivision surface, parametric curves, NURBS patches and particles.

- Alembic also has support for transform hierarchies and cameras.

Layers

- Alembic is composed of **several layers** of libraries, each of which builds on the layer below it.

- **AbcCoreAbstract** is the lowest-level library, and is used for storing and managing Alembic data.

- AbcCoreAbstract is implemented concretely by **AbcCoreHDF5** or **AbcCoreOgawa**.

- In practice, AbcCoreAbstract would only be used by users who wish to implement their own custom data objects which are not part of the official Alembic release.

- **Abc** sits on the top of AbcCoreAbstract and is intended to be easily understood by a non-beginner user.

- **AbcGeom** is implemented mainly atop Abc.

- If you wish to deal with high-level, CG-specific data types such as poly meshes, subdivision meshes, transforms, etc., AbcGeom is the layer for that enables the storage and retrieval of geometric data in an intuitive way.

- Each library has a **namespace** which is the same as the name of it.

- You can find all of the layers at "**/usr/local/include/Alembic**".

HDF5 vs Ogawa

- **Hierarchical Data Format (HDF)** is an open source file format for storing huge amounts of numerical data.

- HDF5 was developed by National Center for Supercomputing Applications.

- HDF5 is designed to address some of the limitations of HDF4.

- HDF5 organizes with the data hierarchically and manages large amount of data very efficiently.

- **Ogawa** is a custom-made thread-safe and efficient data format library that is designed to replace the slow HDF5 system.

- Ogawa format is faster and has smaller file size than HDF5 format.
  - 4~25 times faster file reading
  - 5~15% smaller file size

- Currently, both data (HDF5 and Ogawa) **co-exist** so no backward compatibility will be lost.

- Just newer Alembic files (Ogawa) will be faster and smaller than old Alembic files (HDF5).

- Another reason why Alembic prefers Ogawa to HDF5 is to minimize library dependencies.

Hierarchical Data Structure

- The Alembic stores data in a **hierarchical parent/child tree**.

- A hierarchical tree is an **acyclic directed graph**, i.e. a descendant of an object cannot be the ancestor of that same object.

- Alembic uses forward **slash (/)** to separate each hierarchical level, which is similar to Linux file system.

  ex) /Parent/Child/Grandchild

- In the hierarchical space of Alembic, **/** means the **root**, which is the same as Linux file system.

Parallel Input/Output Abstraction

- Alembic has a completely **separate** but **parallel** class hierarchy for **input** and **output**.

- Classes related to read files has a prefix **'O'**, which represents **"output"**. ex) OArchive

- Classes related to write files has a prefix **'I'**, which represents **"input"**. ex) IArchive

- Conceptually, this is analogous to C++ iostream which has ostream for output and istream for input.

- In this document, however, **we will elide 'I' and 'O'** for the sake of clarity convenience and use them only if necessary.

Major Concept in Alembic

- An Archive consists of multiple **Objects**, which may or may not have **Properties**, in a hierarchical parent/child tree that forms an acyclic directed graph.

- Data are written as **Samples** at specific times, and are read back similarly.

Archive

- An Alembic file has the extension of **".abc"**.
- An Archive is for the actual file on disk (.abc), which contains all of the scene data.
- An Archive is the **top-level container** in a scene (a scene = a file).
- Therefore, an Archive is created by specifying a **file name**.
- Generally, an Archive may have **one or more Objects**, but we can create an empty Archive without any Objects under it.
- When OArchive **goes out of the scope**, an .abc file is created and the data is written to the disk.

Object

- An **Object** is the **main unit** in hierarchical data structure of Alembic.
- You can think of an Archive as a typical file system such as FAT32 or NTFS, and of Objects as directories in that file system.
- Each Object can be the **parent** to any number of other Objects.
- Every Object is the **child** of the other Object except for the Object of the Archive itself.
- In other words, an Archive is **the top-most Object** in an Alembic hierarchy.
- Generally, an Object may have one or more **Properties**, but we can create an empty Object without any Properties under it.

Property

- A **Property** contains **raw data**.
  (raw data: real data which we really wants to stored and extracted)

- There are three types of Properties:

  - **ScalarProperty**: a Property which stores a single scalar value

  - **ArrayProperty**: a Property which stores variable-length arrays

  - **CompoundProperty**: a property which does not contain data directly, but rahter contains other Properties.

- A CompoundProperty can have ScalarProperties, ArrayProperties, or other CoumpoundProperties.

Examples of **ScalarProperties**:

- The name of a rigid body: StringProperty (single string type with extent=1)
- The number of a rigid body: Int32Property (32-bit integer type with extent=1) or Int64Property (64-bit integer type with extent=1)
- The mass of a rigid body: FloatProperty (32-bit floating type with extent=1) or DoubleProperty (64-bit floating type with extent=1)
- The 4x4 transform matrix of a rigid body: M44fProperty (32-bit floating type with extent=16) or M44dProperty (64-bit floating type with extent=16)
- The velocity of a rigid body: V3fProperty (32-bit floating type with extent=3) or V3dProperty (64-bit floating type with extent=3)
- The bounding box of a rigid body: Box2fProperty (32-bit floating type with extent=6) or Box2dProperty (64-bit floating type with extent=6)

Examples of **ArrayProperties**:

- A list of the vertices of a polygonal mesh: V3fArrayProperty (32-bit floating type array with extent=3) or V3dArrayProperty (64-bit floating type array with extent=3)

- A list of the connections of a polygonal mesh: Int32ArrayProperty (32-bit integer type array with extent=1)

- If an Object has any Properties, it must have **a top-level CompoundProperty** which contains all other Properties.

- Therefore, you must access this top-level CompoundProperty first to manipulate all the Properties of an Object.

Sample

- A **Sample** is the container that composes **raw data and a time** into a single entity.
- As mentioned eariler, an Alembic file consists of a series of **Samples of Properties at different times**.
- Therefore, every Property has a notion of when it has been **“sampled”** (for both writing and reading files).
- Namely, data in Alembic is **stored in Samples at specific times**, which are then **stored in Properties**.
- For most cases, Samples and raw data can be considered as the same thing.
- A SimpleProperty (Scalar or Array) can be stored or extracted without a Sample, but if it is an animated (per frame) data, it must be **stored** or **extracted** using a Sample **with specifying its time**.
- A **TimeSampling** is the main interface to time information.
- A **TimeSamplingType** controls how Properties relate time values to their SampleIndices.

There are four types of time sampling that Alembic supports:

- **Uniform**: Time interval is uniform. One Sample per each discrete time. Only start time and the interval are recorded. If the start time is 1.0 and interval is 0.1, the time distribution is 1.0, 1.1, 1.2, 1.3, 1.4, … The most common time sampling type.

- **Identity**: Time interval is 1. Just a special case of uniform sampling. The default time sampling type.

- **Cyclic**: Time interval is non-uniform. Interval and sample array are recorded. If the interval is 1.0 and the sample array is [0.0,0.1,0.3], the time distribution is 0.0, 0.1, 0.3, 1.0, 1.1, 1.3, 2.0, 2.1, 2.3, ...

- **Acyclic**: Time interval is non-uniform. All samples are recored directrly as an array without time interval.

- All Samples must be written in strictly **increasing temporal order**.

- Each Property can have its own TimeSampling scheme, but it is not usual.

MetaData

- Objects and Properties can have **MetaData**.

- MetaData: a collection of string **key/value pairs** that allow users to specify information.

Object/Property Header

- An Object/Property has its **header**.

- An **ObjectHeader**/**PropertyHeader** contains the Object/Property's Name, FullName, and MetaData.

- Therefore, name or MetaData can be extracted through not only Object/Property itself but also ObjectHeader/PropertyHeader, but using ObjectHeader/PropertyHeader is a compact and **quick way to extract name or MetaData**.

- For example, if you wish to find the name of a specific Object, you don't need to (and shouldn't) load the entire Object into memory.

- If you want to know more Object header, see **“…/AbcCoreAbstract/ObjectHeader.h”**.

Summary on Alembic::Abc

- An **Alembic** has an **Archive** which is the top-level Object in the scene of it.
- An Archive consists of **multiple Objects**, which may or may not have **Properties**, in a **hierarchical parent/child tree**.
- An Object can have **raw data** as **Properties**.
- If an Object has raw data, it must have a **CompoundProperty** which is a **top-level container** of other Properties.
- A CompoundProperty can have **SimpleProperties** (either **ScalarProperties** or **ArrayProperties**) or other **CoumpoundProperties**.
- If a SimpleProperty is animated, then it has **Samples**.
- A Sample is to be stored or extracted by **specifying its time**.

Schema

- A **Schema** is a **minimal set** of expected Properties grouped together to represent a **behavior**.

- It is simply a **collection** of Properties that encapsulates some semantic understanding of an Object.

- A **SchemaObject** is simply an Object containing a single, specifically named Schema, and nothing more.

- There are several Schemas in **AbcGeom** layer: **Points**, **Curves**, **PolyMesh**, **NuPatch**, **Camera**, **Xform**, **Light**, etc.

Alembic::AbcGeom

- Let's begin to talk about **Alembic::AbcGeom**.
- Alembic::AbcGeom is based on Alembic::Abc.
- Alembic::AbcGeom is a **high-level API** intended to be used to easily store and retrieve Samples of common geometric data types.
- Using AbcGeom layer is conceptually very mush like using the Alembic::Abc layer but, it provides more **well-defined and higher-level functions**.
- Conceptually, a **PolyMeshSchema** (or simply PolyMesh) is a top-level CompoundProperty of a polygonal mesh Object.
- You can access the raw data of a PolyMesh more easily using Schema than using Property. In the case of a PolyMesh, these Properties would include a list of vertices (a V3fArray), a list of indices (an IntArray), and so on.

References

- https://code.google.com/archive/p/alembic/wikis/AlembicPoint9UsersGuide.wiki
- https://groups.google.com/forum/#!msg/alembic-discussion/FTG1HuuO_qA/jUadpxpk3IoJ
- http://jonmacey.blogspot.kr/2011/12/getting-started-with-alembic.html
- http://i-saint.hatenablog.com/entry/2016/02/09/215542
- http://docs.alembic.io/index.html