Pipeline

Pipeline....1
Pipeline....1
BAM Asset Manager....1
  Installing BAM....2
  bam pack....2
    Examples....3
  bam remap....3
    Subcommands....4
      remap start....4
      remap finish....4
      remap reset....4
Alembic....5
  Exporting to Alembic files....5
    Manual Transform....6
    Scene Options....7
    Object Options....7
    Particle Systems....7

Pipeline

This section of the manual focuses on the integration of Blender into a production pipeline. This is a vast topic that covers many areas of the software, but here we will focus on file/asset management and data I/O.

Note

The tools and workflows documented here require familiarity with working with a command line interface and are mostly aimed at TDs and technical users.

BAM Asset Manager

Refactoring linked .blend files is a common practice in a production environment. While some basic operations can be accomplished within Blender, sometimes it is more practical to perform them on the command line or via a script. During the production of Cosmos Laundromat (Gooseberry Open Movie Project) the *BAM Asset Manager* (BAM) was developed. The original scope of BAM included client-server asset management tools going beyond Blender, but it was later refocused on core utilities to perform two operations:

- blendfile packing
- automatic dependencies remapping

The following section of the manual focuses on how to use BAM.

Installing BAM

BAM is a standalone Python package, that can be run on any system without any particular configuration. The only requirement is Python 3 (and pip, the Python package manager, to easily install BAM).

Windows, Linux and macOS provide different ways to install Python 3 and pip. Check out the online docs to learn more about a specific platform.

Once Python 3 and pip are available, BAM can be installed via command line by typing:

```
pip3 install blender-bam
```

After a successful installation, the *bam* command will be available. By typing it and pressing the Enter key, all the available subcommands will be displayed.

bam pack

This command is used for packing a `.blend` file and *all* its dependencies into a `.zip` file for redistribution.

```
usage: bam pack [-h] [-o FILE] [-m MODE] [-e PATTERNS] [-a] [-q] [-c LEVEL]
             paths [paths ...]
```

You can simply pack a blend file like this to create a zip-file of the same name.

```
bam pack /path/to/scene.blend
```

You may also want to give an explicit output directory. The example shows how to pack a blend with maximum compression for online downloads

```
bam pack /path/to/scene.blend --output my_scene.zip --compress=best
```

The command provides several options to adapt to different workflows (final distribution, partial extraction, rendering).

-o, `--output` `<FILE>`
: Output file or a directory when multiple inputs are passed

-m, `--mode` `<MODE>`
: Output file or a directory when multiple inputs are passed. Possible choices: `ZIP`, `FILE`

-e, `--exclude` `<PATTERN(S)>`
: Optionally exclude files from the pack.

  `--exclude="*.png"`
  : Using Unix shell-style wildcards *(case insensitive)*.

  `--exclude="*.txt;*.avi;*.wav"`
  : Multiple patterns can be passed using the `;` separator.

-a, `--all-deps`
: Follow all dependencies (unused indirect dependencies too)

-q, `--quiet`
: Suppress status output

-c, `--compress` `<LEVEL>`
: Compression level for resulting archive Possible choices: `default`, `fast`, `best`, `store`

`--repo` `<DIR PATH>`
: Specify a “root” path from where to pack the selected file. This allows for the creation of a sparse copy of the production tree, without any remapping.

`--warn-external`

Report external libraries errors (missing paths)

Examples

Consider the following directory layout, and in particular the file *01_01_A.lighting.blend* with its linked libraries.

```
~/agent327/
└─ lib/
   ├─ chars/
   |  ├─ agent.blend  ------------->|
   |  ├─ boris.blend  ------------->|
   |  └─ barber.blend               |
   └─ scenes/                       |
      ├─ 01-opening                 |
      ├─ 01_01_A.lighting.blend  <--|  < BAM pack this file
      └─ 01_01_A.anim.blend  ------>|
```

Once we run `bam pack /scenes/01-opening/01_01_A.lighting.blend` we obtain a *01_01_A.lighting.zip* inside of which we find the following structure.

```
~/01_01_A.lighting
   ├─ 01_01_A.lighting.blend
   └─ __/
      ├─ 01_01_A.anim.blend
      └─ __/
         └─ lib/
            └─ chars/
               ├─ agent.blend
               └─ boris.blend
```

Note how all paths have been remapped relative to the placement of *01_01_A.lighting.blend* in the root of the output. If we run `bam pack /scenes/01-opening/01_01_A.lighting.blend --repo ~/agent327`, the output will be different.

```
~/01_01_A.lighting
   ├─ lib/
   |  └─ chars/
   |     ├─ agent.blend
   |     └─ boris.blend
   └─ scenes
      └─ 01-opening/
         ├─ 01_01_A.lighting.blend  < The BAM packed file
         └─ 01_01_A.anim.blend
```

In this case no path is remapped, and we simply strip out any file that is not referenced as a direct or indirect dependency of *01_01_A.lighting.blend*. This is effectively a sparse copy of the original production tree.

bam remap

Remap blend file paths

```
usage: bam remap [-h] {start,finish,reset} ...
```

This command is a 3 step process:

- first run `bam remap start .` which stores the current state of your project (recursively).
- then re-arrange the files on the filesystem (rename, relocate).
- finally run `bam remap finish` to apply the changes, updating the `.blend` files internal paths.

```
cd /my/project

bam remap start .
mv photos textures
mv barbershop_v14_library.blend barberhop_libraray.blend
bam remap finish
```

Note

Remapping creates a file called `bam_remap.data` in the current directory. You can relocate the entire project to a new location but on executing `finish`, this file must be accessible from the current directory.

Note

This command depends on files unique contents, take care not to modify the files once remap is started.

Subcommands

remap start

Start remapping the blend files

```
usage: bam remap start [-h] [-j] [paths [paths ...]]
```

`-j`, `--json`
: Generate JSON output

remap finish

Finish remapping the blend files

```
usage: bam remap finish [-h] [-r] [-d] [-j] [paths [paths ...]]
```

`-r`, `--force-relative`
: Make all remapped paths relative (even if they were originally absolute)

`-d`, `--dry-run`
: Just print output as if the paths are being run

`-j`, `--json`
: Generate JSON output

remap reset

Cancel path remapping

```
usage: bam remap reset [-h] [-j]
```

Alembic

From the Alembic home page:

> Alembic is an open computer graphics interchange framework. Alembic distills complex, animated scenes into a non-procedural, application-independent set of baked geometric results. This 'distillation' of scenes into baked geometry is exactly analogous to the distillation of lighting and rendering scenes into rendered image data.
>
> Alembic is focused on efficiently storing the computed results of complex procedural geometric constructions. It is very 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 which could involve enveloping, corrective shapes, volume-preserving simulations, cloth and flesh simulations, and so on. Alembic will not attempt to store a representation of the network of computations (rigs, basically) which are required to produce the final, animated vertex positions and animated transforms.

TL;DR: Alembic can be used to write an animated mesh to disk, and read it back quickly & efficiently. This means that a mesh can be animated with a very CPU-heavy rig, 'baked' to an Alembic file, and loaded into the shot file for shading and lighting with only moderate CPU usage.

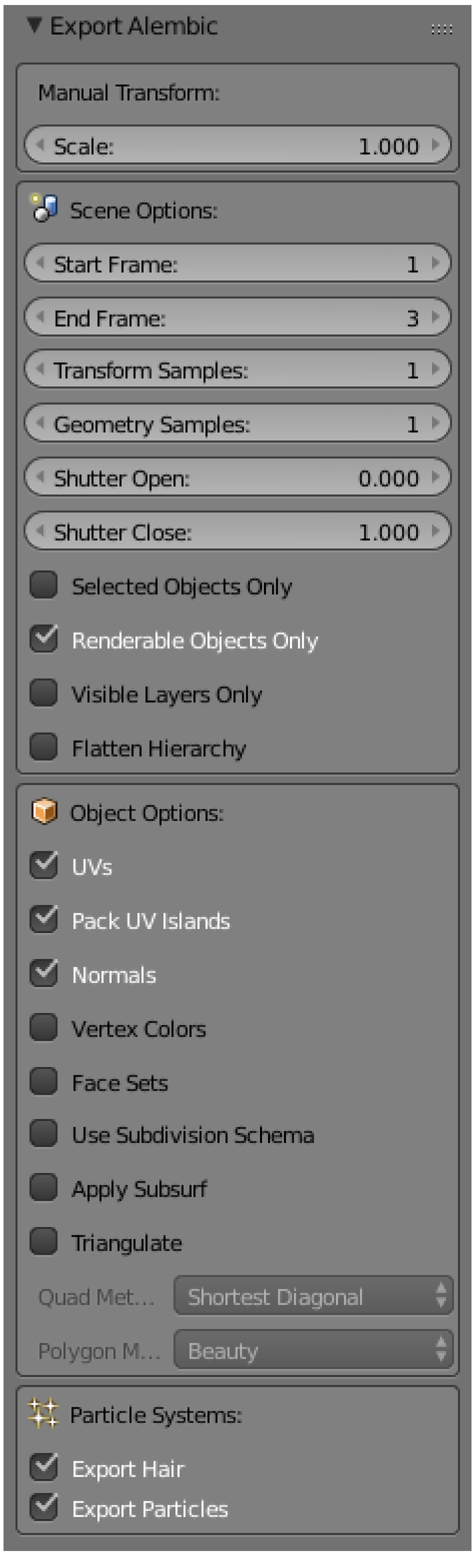
Support for the Alembic file format was introduced in Blender 2.78.

Due to the Open Source nature of the Alembic standard, as well as the C++ library implementing that standard, **Blender can be used in a hybrid pipeline**. For example, other software, such as Houdini or Maya, can export files to Alembic, which can then be loaded, shaded, and rendered in Blender. It is also possible to animate characters (or other models) in Blender, export to Alembic, and load those files into other software for further processing.

Exporting to Alembic files

This section describes the effect of the different export options.

Bforartists Reference Manual - © Copyright - This page is under OCL license



▼ Export Alembic

Manual Transform:

Scale: 1.000

Scene Options:

Start Frame: 1

End Frame: 3

Transform Samples: 1

Geometry Samples: 1

Shutter Open: 0.000

Shutter Close: 1.000

☐ Selected Objects Only

☑ Renderable Objects Only

☐ Visible Layers Only

☐ Flatten Hierarchy

Object Options:

☑ UVs

☑ Pack UV Islands

☑ Normals

☐ Vertex Colors

☐ Face Sets

☐ Use Subdivision Schema

☐ Apply Subsurf

☐ Triangulate

Quad Met… Shortest Diagonal

Polygon M… Beauty

Particle Systems:

☑ Export Hair

☑ Export Particles

Alembic Export options.

Manual Transform

Scale

This sets the global scale of the Alembic file. Keep it at the default value of 1.0 to use Blender’s units.

Bforartists Reference Manual - © Copyright - This page is under OCL license

Scene Options

Start Frame and End Frame

Sets the frame range to export to Alembic. This defaults to the current scene frame range.

Sub-frame sampling: Transform & Geometry Samples, Shutter Open & Close

These options control the sub-frame sampling of animations. Transform Samples sets the number of times per frame at which animated transformations are sampled and written to Alembic. Geometry Samples sets the same, but then for animated geometry. Shutter Open & Close define the interval [open, close) over which those samples are taken. The valid range is -1 to 1, where -1 indicates the previous frame, 0 indicates the current frame, and 1 indicates the next frame. For example, if information for detailed mesh motion blur is desired, some subframes around the current frame can be written to Alembic by using a sample count of 5, Shutter Open at -0.25 and Shutter Close at 0.25. This mimicks a “180 degree” shutter, opening 90 degrees before the frame and closing 90 degrees after the frame.

Selected Objects Only

When enabled, exports only the selected objects. When disabled, all objects are exported.

Renderable Objects Only

This is useful to, for example, avoid exporting custom bone shapes.

Visible Layers Only

Limits the export to scene layers that are currently visible.

Flatten Hierarchy

When disabled, parent/child relations between objects are exported too. Any parent object that is not exported itself, but with children that *are* exported, is replaced by an Empty. When enabled, parent/child relations are not exported, and transformations are all written in world coordinates.

Object Options

UVs

When enabled, UV maps are exported. Although the Alembic standard only supports a single UV map, Blender exports all UV maps in a way that should be readable by other software.

Pack UV Islands

TODO: figure out & describe what this does

Normals

TODO: figure out & describe what this does

Vertex Colors

When enabled, exports vertex colours. At this moment, this only supports static vertex colors, and not dynamically animated vertex colors.

Face Sets

TODO: figure out & describe what this does

Use Subdivision Schema

When enabled, writes polygonal meshes using the “SubD” Alembic schema, rather than the “PolyMesh” schema.

Apply Subsurf

TODO: figure out & describe what this does

Triangulate

Triangulates the mesh before writing to Alembic.

Particle Systems

Alembic has no support for Particle Systems, in the same way that it does not support armatures. Hair is exported as animated zero-width curves. Particles are exported as animated points.