# FBXConverter and CompileFbxArtPackage.bat Documentation

Jibran Syed – Winter 2016

## Introduction

This document was meant to explain the general implementation and usage of the FBXConverter command line program.

## Pipeline

From the FBX file to the final runtime format used by the engine, the FBX art data goes through the following process:

1. The FBX data is loaded by `FBXConverter.exe` and optimized into runtime formats of 4 different types
    a. Vertex Data (Reduced)
    b. Triangle Data (Corrected)
    c. Texture Buffer (Formatted)
    d. Bounding Sphere Data (Calculated)
2. The 4 mentioned formats get saved into 4 files each
    a. `.vrt` for vertex data
    b. `.tri` for triangle data
    c. `.tex` for texture buffer
    d. `.sph` for bounding sphere
3. `MetaAppender.exe` adds metadata to these exported formats like name and type-of-resource, so that they can be quickly looked up on the hard disk. It creates a new file, appending `.lge` to the extension.
4. `DataPackager.exe` takes every `.lge` file in the current directory and combines all the binary data into one package file with the extension `.lgepkg`. The default name used in the engine is "`game_resources.lgepkg`".

To streamline the process, a batch script called `CompileFbxArtPackage.bat` exists to take the FBX files specified in the script and do the proper conversions. Unfortunately adding new FBX files to the pipeline would require manually tweaking of the script file.

The data in the final package becomes the source for loading all the data that originally existed in the FBX files. The generated package is copied into the game engine's project directory in `/res/packages/`.

## Runtime Formats

The runtime format of the vertex data starts with a header struct called `VboHeader` followed by an array of `VertexData` objects. `VboHeader` contains general information about the vertices, such as raw size in bytes, size of each element, and number of elements.

To optimize vertices, they are sorted and reduced to prevent redundant data. This also affects the triangle data, and is corrected accordingly.

It must be noted that the converter turns 1 `VertexData` into 8 `Floats`. That's because `VertexData` is composed of 8 `floats`: positions X-Y-Z, texture coordinates U-V, and vertex normal components $N_X$-$N_Y$-$N_Z$. These are the vertex attributes used by the engine in that order. It's all converted into floats so that OpenGL can just take in the data as raw floats.

----

The runtime format of the triangle data starts with a header called `EboHeader` followed by an array of `TriangleData` objects. `EboHeader` contains similar information to `VboHeader`, but instead it has information about the triangle data of the FBX mesh. `TriangleData` composes of three integer indices representing the different vertices needed to draw one triangle. The engine takes in this triangle data as raw integers.

----

The runtime format of texture data starts with a header called `TextureHeader` and is followed by a raw buffer of texture data. The texture data is of type `unsigned char` and can support either RGB or RGBA color data formats, depending on the depth of the texture. The `TextureHeader` contains the following information:

- The width of the texture
- The height of the texture
- The depth of the texture
    - Usually 24 or 32 bits
- The color format to be used in the GPU
- The color format originally used in the texture file

---

The runtime format of the bounding sphere only composes of a single struct called `BoundingSphere`. This struct contains the center of the bounding sphere in model space, and the radius of the sphere, also in model space. The spheres are translated and scaled in real-time in the engine, so that the original bounding sphere data isn't tampered with.

Bounding spheres are calculated by examining all the vertices in the FBX mesh and calculating the centroid and bounding volume using Ritter's Method. The results are accurate enough to ensure a good center and radius, with a tiny margin of error.

## Usage

The program `FBXConverter` has the following command line usage:

```
FbxConverter [FBX_FILE_NAME] --no-verbose --reverse-winding
```

The `FBX_FILE_NAME` is the name of the file to load model data from. It must be in the FBX file format.

`--no-verbose` is an optional parameter that disables verbose printing of the vertex buffers in the process of being reduced. It's recommend if using in a batch script.

`--reverse-winding` is an optional parameter that changes the vertex winding of the exported mesh. This is important for front-face-culling engines that define the "front" of a polygon by the direction of the winding of its vertices.

This program will output 4 files with the same name as the FBX file, but be of different extensions: `.vrt`, `.tri`, `.tex`, and `.sph`.

-----------

The program `MetaAppender` has the following command line usage:

```
MetaAppender data.bin -t [RESOURCE_TYPE] -n [NAME]
```

The `data.bin` can be any file of any extension. The engine will only work with `.vrt`, `.tri`, `.tex`, and `.sph` files.

`RESOURCE_TYPE` is a required parameter that must come after a `-t` tag that indicated the type of data being packaged. The following are valid inputs:

- `Vertices`
- `Triangles`
- `Texture`
- `BoundingSphere`

`NAME` is a required parameter that must come after an `-n` tag that indicates the name of this data as it will be referenced by the engine and its unpacking utility class.

This program will output one file that is the original input file's name but with `.lge` appended at the end, so like `data.bin.lge` for example.

The program DataPackager has the following command line usage:

```
DataPackager package.lgepkg -n [NAME] -v [VERSION]
```

The `package.lgepkg` is the data package of all the processed FBX files to be used in the engine It can technically have any name or any extension, but the engine is currently hard-coded to only look for a package called `game_resources.lgepkg`. This file is the final output.

`NAME` is a required parameter that must come after an `-n` tag that indicated the name of the data package. Currently, the engine doesn't make use of this information.

`VERSION` is a required parameter that must come after a `-v` tag that indicates the version of this data package. The version can be any string such as "1.0" or "v1.0" or "v1.0.0", etc. Currently, the engine doesn't make use of this information.