The Binary Static Mesh format (.BSM) Version 1 DRAFT

zvxryb@gmail.com

April 2, 2012

THIS DOCUMENT AND THE FORMAT IT DESCRIBES ARE NOT YET FINALIZED

1 Rationale:

The vast majority of modern model formats are designed primarily to act as generic interchange formats between modelling suites and do not accommodate the needs of modern computer games. For example, the Wavefront OBJ format, which is nearly-ubiquitous as a static mesh format, has the following problems:

1. It does not natively support per-vertex tangent-space attributes
2. Vertex coordinates, UV coordinates, and normals do not necessarily have a one-to-one mapping (when they are provided)
3. Use of smoothing groups rather than a fixed, pre-determined and optimized list of primitives

This means that a Wavefront OBJ loader must, at minimum:

1. Emit an entirely new list of vertex attributes with the one-to-one mapping required by modern graphics APIs.
2. Fill in missing vertex attributes.
3. Calculate per-vertex tangent space bases.
4. Split primitives along mirrored UVs, where tangent-space orientation changes abruptly.
5. Split primitives between smoothing groups.
6. Sort primitives into groups based on their material.
7. Optimize the resulting mesh.

While OBJ is addressed specifically, many of these issues are typical of other common formats. By offloading these tasks to export scripts for modelling suites, or to external converters/compilers, we can simplify the process of importing models and rely on a few well-tested tools instead of forcing game developers to choose between developing their own importers or using a bulky import library to do the work for them (which clumsily attempt to reconcile the inherent structural differences between formats, with mixed results). It is also considerably more likely that an export script will have the necessary mesh-processing facilities at its disposal, whereas an importer into a typical application would have to provide these routines for importing, specifically.

In designing a format which is constrained to only those features which are relevant to every-day game design, we aim to create a static mesh format which may be readily imported into any modern renderer with minimal parsing and processing.

(Also, COLLADA. Lol.)

2 Guiding design principles:

Our goal is to have one unifying static mesh format that any casual programmer can comprehend in its entirety and integrate in an afternoon; In short, the mesh analog of the ubiquitous TGA raster graphics format. It is also designed to compliment the IQM dynamic model format, and is partially inspired by its design.

BSM is designed to be simple and straightforward. It a raw binary representation of mesh data. It does not attempt to support every feature and format available, but defines one single, consistent format. Where the core features are insufficient, a flexible extension mechanism is provided which enables users to add auxiliary data while still complying with the core specification.

Numerical data is represented using 32-bit integers and floating-point values, such that endianness conversions, where necessary, can be performed in 4-byte chunks on most structures without knowledge of the layout of the structure (the necessary exception to this rule is the material name string in the `mesh` structure). Use of structures with four-byte components also simplifies packing and alignment rules on common platforms.

3 Features:

3.1 Current:

1. Simple bounding geometry (bbox/bsphere)
2. Typical vertex attributes for modern static meshes:
   (a) World-space coordinates
   (b) Texture coordinates

    (c) Normal vectors
    (d) Tangent vectors

3. Multiple meshes with unique material identifiers

4. Multiple convex hulls for collision detection and physics simulation

5. Occlusion mesh for dynamic visibility

3.2 Maybe in the future:

1. More sophisticated bounding geometry

    (a) Oriented bounding boxes
    (b) Convex hull
    (c) Per-mesh bounding geometry

2. Triangle adjacency data.

3.3 Probably never:

1. Animation – Try IQM for dynamic models, it's pretty good.

2. Complex physical attributes or materials – This is better left to a dedicated material fomat.

3. Entities such as cameras, lights, etc. – This is better left to a dedicated map or scene graph format.

4. 'Optional' fields and attributes – Users should know what to expect from a BSM model of any given version/extension.

5. Compression – This is unnecessary because meshes tend to be relatively small and it is common for game assets to be packaged in a compressed archive. In cases where you absolutely need compression, you may use external compression utilities (.bsm.bz2!)

6. Anything uncommon or excessively complex.

4 Maintenance

This is a public-domain format, which I hope will be community-maintained. For most people, an extension should be sufficient. Popular features may be adopted for inclusion in the core format. I'll keep track of changes, if you're interested in updating the format, send me an email at zvxryb@gmail.com.
[link to forum or IRC channel or some such thing?]

5 Extension/derivative guidelines:

Individuals may extend the format to include other attributes, and still use the BSM magic number and extension, provided that:

1. The version number must match the version number of the core format which it extends.
2. The extension ID may be any number which uniquely identifies your extended format, other than zero. Extension IDs of zero are reserved exclusively for the core format.
3. The format should be backwards-compatible with the core format which it modifies. This means that all extensions of the version 1 format should include a *full version 1 header* and all of the other required fields. Your extended format may include additional 'extension header' or 'secondary header' which follows the full version 1 header.

All versions of the core format must be backwards-compatible in the same manner as any other extension, such that a BSM loader written for version 1 of the format can extract all of the version 1 properties from a version 1, 2, 3, ..., n file.

6 THE FORMAT

1. Data is stored in contiguous arrays, at locations specified within the header. Byte offsets are relative to the start of the file.
2. All values are stored in little-endian byte order.
3. "Float", where it appears, refers to 32-bit single-precision IEEE 754 floating-point values.
4. Byte offsets and item counts should always be ≥ 0.
5. Magic number:
   (a) As an ASCII string: `"BINARYSTATICMESH"`
   (b) As 32-bit integers: `0x414E4942, 0x54535952, 0x43495441, 0x4853454D` (little-endian)

6.1 Header:

Header

`0x00`	`char[16]`	MAGIC NUMBER
`0x10`	`int32`	Version
`0x14`	`int32`	Extension ID
`0x18`	`bsphere`	Bounding sphere
`0x28`	`bbox`	Bounding box
`0x40`	`int32`	Number of vertices
`0x44`	`int32`	Byte offset of world coordinate array
`0x48`	`int32`	Byte offset of texture coordinate array
`0x4C`	`int32`	Byte offset of normal vector array
`0x50`	`int32`	Byte offset of tangent vector array
`0x54`	`int32`	Number of triangles
`0x58`	`int32`	Byte offset of triangle vector array
`0x5C`	`int32`	Number of meshes
`0x60`	`int32`	Byte offset of mesh array
`0x64`	`int32`	Number of collision vertices
`0x68`	`int32`	Byte offset of collision vertex array
`0x6C`	`int32`	Number of convex hulls
`0x70`	`int32`	Byte offset of convex hull array
`0x74`	`int32`	Number of occlusion mesh vertices
`0x78`	`int32`	Byte offset of occlusion mesh vertex array
`0x7C`	`int32`	Number of occlusion mesh triangles
`0x80`	`int32`	Byte offset of occlusion mesh triangle array
132 bytes		

Bounding Sphere (bsphere)

`0x00`	`float`	X
`0x04`	`float`	Y
`0x08`	`float`	Z
`0x0C`	`float`	Radius
16 bytes		

Bounding Box (bbox)

`0x00`	`float`	Min. X
`0x04`	`float`	Min. Y
`0x08`	`float`	Min. Z
`0x0C`	`float`	Max. X
`0x10`	`float`	Max. Y
`0x14`	`float`	Max. Z
24 bytes		

6.2 Vertex Attributes:

World Coordinate

`0x00`	`float`	X
`0x04`	`float`	Y
`0x08`	`float`	Z
12 bytes		

Texture Coordinate

`0x00`	`float`	U
`0x04`	`float`	V
8 bytes		

Normal Vector

`0x00`	`float`	X
`0x04`	`float`	Y
`0x08`	`float`	Z
12 bytes		

Tangent Vector

`0x00`	`float`	X
`0x04`	`float`	Y
`0x08`	`float`	Z
`0x0C`	`float`	Tangent-space handedness
16 bytes		

(handedness should be 1.0 if $\vec{N} \times \vec{T} = \vec{B}$ or -1.0 if $\vec{N} \times \vec{T} = -\vec{B}$)

The tangent, bitangent, and normal of a given vertex are unit-length basis vectors for an orthogonal tangent space. Tangent and bi-tangent are defined with respect to the u- and v- texture-space coordinates. Bi-tangent is not explicitly stored, but may be recovered from the normal and tangent vectors. The normal and tangent vectors are *required to be normalized* before they are written to disk. A compliant BSM reader is not required to re-normalize normal or tangent vectors, however it may be done in order to support files from non-compliant writers.

6.3 Meshes:

Triangle

`0x00`	`int32[3]`	Three vertex indices
12 bytes		

Triangles are represented with counter-clockwise winding.

Mesh

`0x00`	`int32`	Index of first triangle in mesh
`0x04`	`int32`	Number of triangles in mesh
`0x08`	`char[256]`	Material name string (UTF-8, null-terminated)
264 bytes		

6.4 Collision:

Convex Hull Vertex

`0x00`	`float`	X
`0x04`	`float`	Y
`0x08`	`float`	Z
12 bytes		

Convex Hull

`0x00`	`int32`	Index of first collision vertex in convex hull
`0x04`	`int32`	Number of collision vertices in convex hull
8 bytes		

6.5 Occlusion Geometry:

A single low-resolution occlusion mesh which may be drawn to a depth buffer and tested against.

Occlusion Mesh Vertex

`0x00`	`float`	X
`0x04`	`float`	Y
`0x08`	`float`	Z
12 bytes		

Occlusion Mesh Triangle

`0x00`	`int32[3]`	Three vertex indices
12 bytes		