GOM（Game Oriented Model Format)草案

GOMフォーマット仕様書

Game Oriented Model Format (.gom) Version 1.0 (Complete Specification)

---

1. 目的

GOMは、FBX等のDCCデータをゲーム実行向けに最適化し、 オフライン変換して保存するためのフォーマットである。

設計目標

- スキニングはモデル（geometry）空間で完結
- インスタンスごとに別Transform・別アニメ時間を安全に扱える
- FBX差（ノード階層・geometry transform）を吸収
- ワールド直スキニング事故を防止
- 実行時にFBXSDKやufbxなどのライブラリに依存しない
- 同一FBXから常に同一GOMが生成される（決定性保証）

---

2. 空間設計ポリシー（不変条件）

Invariant A: 頂点空間

- 頂点は常に geometry（=モデルローカル）空間
- スキニング結果も geometry 空間
- instanceWorld は描画直前に1回のみ適用

禁止事項

- スキニング途中で world に焼く
- geometry_to_world を途中適用する

---

3. 行列規約（最終固定）

GOM v1では以下を固定する。

- 数値型：float32
- メモリ並び：row-major
- 演算規約：row-vector 右掛け（p' = p * M）
- 行列は常に 4x4

- 座標系：Left-handed / Y-up / meters

4. ファイル形式

GOMはテキスト形式（GOMT0001）とバイナリ形式（GOMB0001）を持つ。

マジックナンバー（先頭8バイト）

- GOMT0001
- GOMB0001

5. テキスト形式（GOMT0001）

UTF-8（BOMなし）、構造リテラル形式。

5.1 Skeleton

```
Skeleton = {
  boneCount = <int>,
  bones = [
    {
      parent = <int>,
      name   = "<string>",
      path   = "<string>",
      bindLocal = [[...],[...],[...],[...]]
    }
  ]
}
```

- bones配列順がboneIndex
- pathは同名衝突対策

5.2 MeshPart

```
Meshes = [
  {
    name = "<string>",
    meshBindGeoToModel = [[...],[...],[...],[...]],
    vertices = [ { p,n,t,uv,bi,bw } ],
    indices = [ ... ],
    submeshes = [ { materialIndex, indexStart, indexCount } ],
    inverseBind = [ [[...]] ]
  }
]
```

ウェイト規約（必須）

- 最大4影響
- 合計1.0正規化
- weight降順、同値はboneIndex昇順タイブレーク

5.3 Animation

```
Animations = [
  {
```

```
    name = "<string>",
    duration = <float>,
    rootBoneIndex = <int>,
    flags = ["HasRootMotion"],
    tracks = [ { bone, keys[] } ]
  }
]
```

補間：T/S=linear、R=slerp

---

6. MeshPart規約

Rule M1

- FBXメッシュノード単位でMeshPart生成
- マテリアル分割はSUB0で表現

Rule M2

以下の場合はMeshPartを分割する：

- 別スケルトン参照
- inverseBindが異なる
- 頂点レイアウトが異なる

---

7. バイナリ形式（GOMB0001）

7.1 固定ヘッダ（512 bytes）

Little Endian固定。

7.1.1 固定値

- `magic` は常に `GOMB0001`（8 bytes ASCII）
- `headerSize` は常に `512`
- `formatVersion` は常に `1`

7.1.2 列挙値（enum）定義

本仕様の enum はすべて `uint32` とする。 未知の enum 値を受け取った場合、読み込みはエラーとする（v1）。

endianness（将来用）

v1ではファイルはLittle Endian固定だが、将来拡張のためヘッダに enum を持つ。

- `ENDIAN_LE = 1`（v1は常にこれ）
- `ENDIAN_BE = 2`（予約）

coordinateSystem

- `CS_LH_YUP_METERS = 1`

matrixConvention

行列の「格納順＋演算規約」を表す。

- `MC_ROW_MAJOR_ROW_VECTOR_RIGHT_MUL = 1`

定義：

- 行列は row-major で格納される（m00,m01,m02,m03,m10,...）
- 位置は row-vector として扱い `p' = p * M`（p=[x y z 1]）

7.1.3 validationFlags（bitfield）

`validationFlags` は `uint32`。

- bit0（1<<0）: bindPosePass
- bit1（1<<1）: weightsPass
- bit2（1<<2）: submeshMaterialsPass

予約：bit3以降は将来用。v1では0。

7.1.4 ChunkTableの指定

- `chunkTableOffset == 0` の場合：ChunkTableなし
- `chunkTableOffset != 0` の場合：ファイル先頭からのバイトオフセットで ChunkTable 先頭を指す
- `chunkTableCount` はエントリ数

```
char   magic[8]            // GOMB0001
uint32 headerSize          // 512
uint32 formatVersion       // 1
uint32 coordinateSystem    // enum
uint32 matrixConvention    // enum
uint8  sourceHash[32]      // SHA-256 raw
uint32 converterVersionStr // STR0 index
uint32 validationFlags     // bitfield
uint32 chunkTableOffset
uint32 chunkTableCount
uint8  reserved[...]       // 0埋め
```

---

8. バイナリ全体の堅牢ルール（バイナリ専用）

8.1 Endianness

- Little Endian固定

8.2 アライメント

- チャンクは16バイト境界にアライン

8.3 チャンク基本構造

```
uint32 fourcc
uint32 size
uint32 version
uint32 flags
byte payload[size]
pad to 16
```

8.4 チャンクサイズ上限チェック（必須）

- sizeがファイル末尾を超える場合は即エラー
- sizeが実装上限を超える場合はエラー

8.5 未知チャンク

- 未対応fourccはsize分スキップ可能

8.6 ChunkTable（任意）

- chunkTableOffset!=0 の場合、ChunkEntry配列を保持

8.7 MSH0.flags（任意）

- bit0 = indices16
- bit1 = hasGBTM
- 予約ビットは0

---

9. バイナリチャンク厳密レイアウト

STR0

```
uint32 stringCount
uint32 offsets[stringCount]
uint32 dataSize
uint8  data[dataSize]
```

SKEL

```
uint32 boneCount
int32 parent[boneCount]
uint32 nameId[boneCount]
uint32 pathId[boneCount]
float32 bindLocal[boneCount][16]
```

MSH0

```
uint32 meshNameId
uint32 vertexFormat
uint32 vertexCount
uint32 indexCount
uint32 boneCount
uint32 flags
uint32 reserved[2]
```

VB0

```
uint32 vertexCount
uint32 stride
uint32 vertexFormat
uint32 reserved
uint8  data[vertexCount*stride]
```

IB0

```
uint32 indexCount
uint32 indexSizeBytes
uint32 reserved[2]
uint8 data[indexCount*indexSizeBytes]
```

SUB0

```
uint32 submeshCount
struct { int32 materialIndex; uint32 indexStart; uint32 indexCount; }
```

IBND

```
uint32 boneCount
float32 inverseBind[boneCount][16]
```

BND0

```
float32 aabbMin[3]
float32 aabbMax[3]
float32 sphereCenter[3]
float32 sphereRadius
```

GBTM（任意）

```
float32 meshBindGeoToModel[16]
```

ANIM

```
uint32 clipCount
```

CLP0

```
uint32 clipNameId
float32 duration
float32 sampleRate
int32 rootBoneIndex
uint32 flags
uint32 trackCount
```

Track構造：

```
int32 boneIndex
uint32 keyCount
for each key:
  float32 time
  float32 tr[3]
  float32 rot[4]
  float32 sc[3]
```

---

10. 変換ツール検証（必須）

- Bind整合
- ウェイト合計
- Submesh完全被覆

- boneIndex範囲チェック
- index範囲チェック

---

11. 実装安全規約（v1推奨）

11.1 読み込み上限値（推奨）

実装は以下の上限を持つことを推奨する（超過時はエラー）。

- maxBoneCount = 4096
- maxVertexCount = 10,000,000
- maxIndexCount = 30,000,000
- maxClipCount = 1024
- maxTrackCount = 8192
- maxKeyCount = 1,000,000（クリップ単位）

※ 実装側で変更可能だが、無制限読み込みは禁止。

11.2 必須チャンク条件（バイナリ）

各 MSH0 は以下の子チャンクを必須とする：

- VB0
- IB0
- SUB0
- IBND
- BND0

GBTM は任意。

IB0.indexSizeBytes は 2 または 4 のみ許可。 それ以外は読み込みエラー。

11.3 アニメキー時刻規約

- keys[].time は昇順必須
- 0 <= time <= duration
- ループ再生時は time = fmod(time, duration) を推奨

11.4 inverseBind の定義

inverseBind[i] は、bind姿勢における boneWorld[i] の逆行列に相当する。

スキニングパレットは以下で構成される：

palette[i] = boneWorld_current[i] * inverseBind[i]

これにより、geometry 空間内でスキニングが完結する。

End of GOM Specification v1.0

Revision #1
★Created 2 March 2026 23:42:41 by youe2
Updated 9 June 2026 12:18:20 by youe2