エディタでつくる glTF ファイル

ソリューション本部　開発二部

中島　太朗

1. はじめに

3D モデルのファイル形式には様々なものがあるが、その中でも glTF は無料で誰でも利用でき、インターネットやモバイルでの効果的なデータ送受信を考慮に入れた次世代のファイル形式仕様である。

本稿では、3D ファイルとはどのようなものかを馴染みのない方に向けて、glTF ファイルを元にチュートリアル形式で説明していく。

2.　3D 形状データを構成する要素

glTF ファイルフォーマットの説明の前に、3D 形状データとは何かについて説明する。

アプリケーション上で三角形を表示することを考えた場合、最低限には以下の要素が必要となる。

- 頂点位置
- 面を構成する頂点インデックスの列
- 面の裏表情報

3.　glTF ファイルとは？

glTF（GL Transmission Format）とは、The Khronos Group が策定した、ネットワーク上で 3D コンテンツを効果的に送受信可能なロイヤリティフリーのファイルフォーマット仕様である。glTF は、テキストとバイナリ形式の両方があり、テキスト形式の場合は、JSON 形式で記述する。

ただし、glTF ファイルは、テキスト形式とは言っても、座標値などのデータは、バイナリファイルあるいはバイナリをテキスト化した BASE64 形式で格納する。

4. 3D 形状データを作成する

3D 形状データに具体的な数値を与えて、glTF ファイルを作っていく。以下のような四角形を作成する。

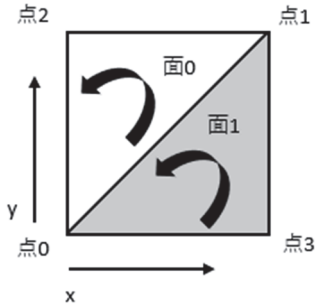

図 1　本稿で作成する四角形

頂点位置

点 0 (0.0, 0.0, 0.0),
点 1 (1.0, 1.0, 0.0),
点 2 (0.0, 1.0, 0.0),
点 3 (1.0, 0.0, 0.0)

面構成の頂点インデックスの列と順序

面 0（点 0，点 1，点 2）
面 1（点 1，点 2，点 3）

glTF では、四角形も三角形の面の集まりとして表現するため、面を二つ作成することになる。

面構成の頂点順序によって、面のウラオモテを表現する。視線方向から面を眺めた時に、点の並びが反時計周りになっている場合に、面をオモテとみなして表示する。図 1 では、面0が点 0→点 1→点 2、面 1 が点 1→点 2→点 3 の並びが反時計周りになっているため、オモテになり、表示される。

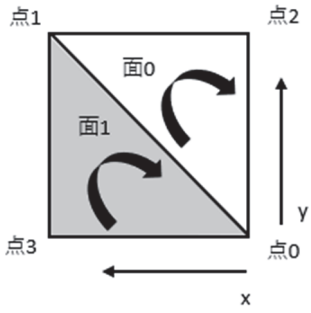

図 2 図 1を裏側から眺めた図

図 1 を裏側から眺めた場合の図 2 は、点の並びが時計周りとなるため、ウラと判定され、面は表示されない。

5. バッファ用ファイルを作成する

バッファ用ファイルはバイナリなので、バイナリエディタを使って作成する。

前章の頂点位置と頂点インデックス列をバイナリに落とし込むと以下のようになる。

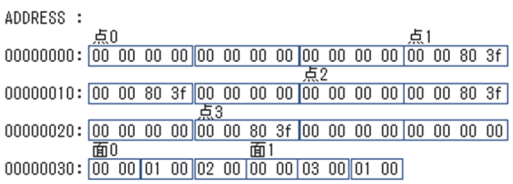

図 3 バイナリエディタ上での四角形データ

数値 2 つで 00～FF の値が入り、1 バイトを表す。四角の枠で囲んだ 4 バイトで float 値（単精度浮動小数点）を表す。一つの頂点位置は、X,Y,Z の 3 つの数値が必要なため、12 バイトを占める。位置 0x00000000～0x0000002F（3行目まで）が、48 バイトとなり、頂点位置 4 つの領域となる。

最初の四角の枠は、すべて 0 が入っていることから、0.0 の値を表すことは想像できるかと思われる。実際、0x00000000 は 0.0 を表す。点 0 の座標値は(0.0, 0.0, 0.0)でである。点 1 の座標値は（1.0, 1.0, 0.0）である。1.0 は単精度浮動小数点で表すと 0x3f800000 となる。図では、4 つめの四角の枠で囲まれた 00 00 80 3f の並びのところになる。glTF は、値に対するエンディアン（複数バイトの並べ方）が、桁の小さい方から並べるリトルエンディアンのため、0x3f800000 が、00 00 80 3f の並びとなっている。（なお、桁の大きい方から並べることをビッグエンディアンと呼び、その場合は 3f 80 00 00 の順に並ぶ。）

また、0x3f800000 が 1.0 なのは、何故かと疑問を持つ方もいるであろう。これについては、IEEE 754 という標準仕様が元になっているためである。https://float.exposed/0x3f800000 のサイトが比較的分かりやすいので、参考にして頂きたい。

次に面の並びについては、(0, 1, 2)、(0, 3, 1)と定義した。unsigned short（符号なし短整数型）の 2 バイトで整数を表現する。それぞれ(0x0000, 0x0001, 0x0002)、(0x0000, 0x0003, 0x0001)となる。これらを glTF のリトルエンディアン仕様で並べると、各数値のバイトが逆順になるので、00 00 01 00 02 00 00 00 03 00 01 00 というバイナリエディタ上での並びとなる。

作成したバッファ用ファイルは rect.bin という名前で保存する。

6. glTF ファイルに落とし込む

頂点位置とインデックス列のバイナリファイルが作成できたので、次に、テキストエディタを使って、glTF の JSON ファイルフォーマットに落とし込んだものを解説する。

glTF ファイルは、3D データを scene、node、mesh、accessor、bufferView、buffer に主に分けて管理する。

scene と node はシーンの基本構造について記述する。

mesh は 3D オブジェクトのジオメトリ(形状情報)について記述する。

accessor, bufferView, buffer はデータの参照先、レイアウトを記述する。

JSON フォーマットの場合、[ ]を配列、{ }をオブジェクトと呼びます。{ }は、キーと値をペアで持ち、配列と同様にカンマで複数保持する。

まずは、buffers について説明する。

```
54   "buffers": [
55     {
56       "byteLength": 60,
57       "uri": "rect.bin"
58     }
59   ],
60   "asset": {
61     "version": "2.0"
62   }
63 }
```

図 4 buffers の記述

buffers[0].uri には、バイナリファイル名が glTF ファイルを基準とした相対パスで入る。glTF と bin ファイルは同じ階層に置くので、ファイル名だけでよい。また、byteLength には rect.bin のファイルサイズ 60 バイトが入る。

次に buffers を参照する bufferViews を説明する。

```
42   "bufferViews": [
43     {
44       "buffer": 0,
45       "byteLength": 48,
46       "byteOffset": 0
47     },
48     {
49       "buffer": 0,
50       "byteLength": 12,
51       "byteOffset": 48
52     }
53   ],
```

図 5 bufferViews の記述

bufferViews には、バッファ領域情報が入る。

bufferViews[0]には頂点位置、bufferViews[1]には面の頂点インデックスの列を設定する。buffer キーは、先に設定した buffers のインデックス値 0 を持つ。bytreLength、byteOffset については、rect.bin における、頂点位置情報であるサイズ 48 バイト、開始位置 0 バイト、面の頂点インデックスの列情報であるサイズ 12 バイト、開始位置 48 バイトをそれぞれ設定する。

次に bufferViews を参照する accssors について説明する。

```
28   "accessors": [
29     {
30       "bufferView": 0,
31       "componentType": 5126,
32       "count": 4,
33       "type": "VEC3"
34     },
35     {
36       "bufferView": 1,
37       "componentType": 5123,
38       "count": 6,
39       "type": "SCALAR"
40     }
41   ],
```

図 6 accessors の記述

accessors には、bufferViews と同様に頂点位置と面の頂点インデックスの列のバッファ領域の型情報を格納する。

5126, 5123 はマジックナンバーで float(単精度浮動小数点)と unsigned short(符号なし短整数型)を表す。VEC3 は XYZ の座標値、SCALAR はスカラを表す。

accessors[0]は、頂点位置の型が float 値で、VEC3 を 4 つ持ち、accessors[1]は unsigned short 値で 6 つ持つことを示す。

次に accessors を参照する meshes について説明する。

```
16   "meshes": [
17     {
18       "primitives": [
19         {
20           "attributes": {
21             "POSITION": 0
22           },
23           "indices": 1
24         }
25       ]
26     }
27   ],
```

図 7 meshes の記述

meshes ではメッシュの属性を設定する。

POSITION キーは、頂点位置を表すので、頂点位置情報を持つ accessors[0]を参照するために 0 を、indices キーは面の頂点インデックス列を表すので、accessors[1]を参照するために 1 を設定する。meshes 内部はキー名によって何を示すかが変わってくるが、インデックスを表す数値は、POSITION・indices キーのように、accessor を参照することが多い。

最後にシーン全体を定義する。

```
1 {
2   "scene": 0,
3   "scenes":
4   [
5     {
6       "nodes": [
7         0
8       ]
9     }
10  ],
11  "nodes": [
12    {
13      "mesh": 0
14    }
15  ],
```

図 8 scene・scenes・nodes の記述

“scene”:0 で scenes[0]のシーンを使うことを示す。

3 行目の scenes 配列で全シーンを定義する。一つのシーンだけを設定するので、scenes の配列に、オブジェクトは一つになる。

6 行目で scenes[0]がどのノードを使うかを示す。0 の値は 11 行目からの nodes[0]を使うことを表す。

11 行目の nodes は、glTF ファイル内の全ノードを設定する。meshes[0]を参照するノードとなる。

JSON ファイルを眺めると、scenes[0]が使用する nodes[0]の指し示す mesh が accessor→bufferView →buffer→バイナリファイルの順に紐づいたデータを参照することが分かるかと思う。

glTF のファイル名は、rect.gltf とする。

7. glTF ファイルを表示する

glTF ファイルが作成できたので、表示する。Windows 10 では標準で glTF ビューワーが入っている。作成した rect.gltf ファイルをダブルクリックすると図 9 のように表示されることが確認できる。

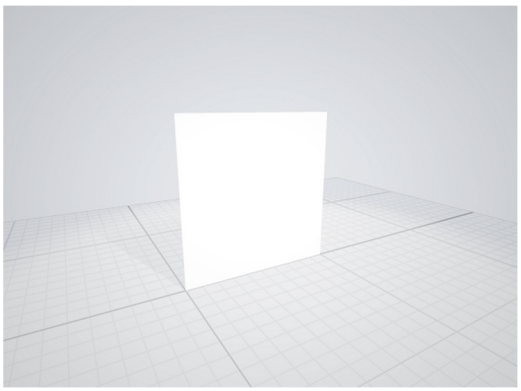

図 9 完成表示図

なお、下記 URL に、本稿で作成したファイルをアップロードしているので参考にして頂きたい。

https://github.com/tar-naka-app/gltf-tuto

8. おわりに

駆け足で、glTF ファイルについて説明してきたが、どのようなものか理解して頂けただろうか。3D ファイルを業務や趣味で扱う場合に、本稿を参照して頂き、理解の一役を少しでも担えれば幸いである。

＜参考文献＞

1） https://github.com/KhronosGroup/glTF/tree/master/specification/2.0

2） https://float.exposed/0x3f800000