三角形から四面体へのコンバートに関して2
中川展男 **2022年8月23日**

目的

この文章では、三角形サーフェイスメッシュから四面体ボリュームメッシュへのコンバートに関して説明します。特に Matthias Müller さんの web サイト “Ten Minutes Physics”上で公開されている Blender アドオン(Python 実装)を C++ で移植してスタンドアロンのコマンドラインアプリケーションで動くように追実装しましたのでその過程で得た知見を共有します。オリジナル実装では、オプション詳細に関する情報がないためこの文章では、細かいオプション指定の違いに関してまとめる事も目的とします。

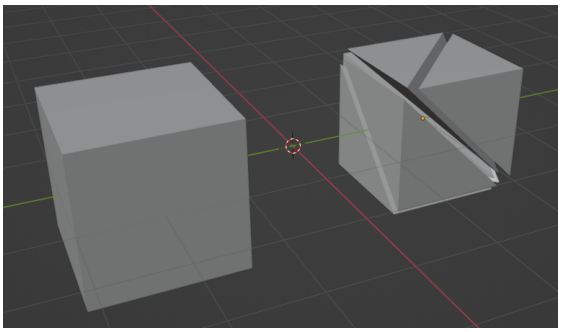
配布物

tetrahedralizer.zip

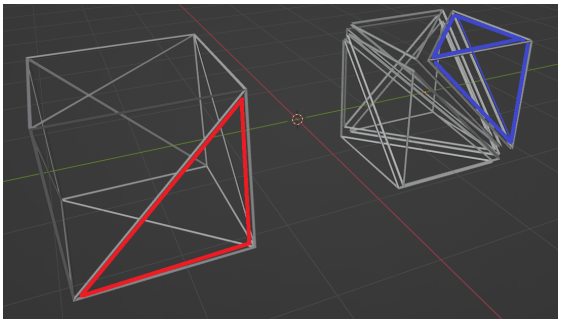
windows(x64) の実行形式を用意しています。テスト用に入力用の .obj ファイルも同梱しました。

概要

下図に示す通り三角形メッシュ(左)から四面体ボリュームメッシュ(右)の生成を行ないます。



ワイヤフレーム表示にすると左は三角形で構成され右は四面体で構成されていることがわかります。



わかりやすく見せるため、右側の箱は四面体と四面体との間に少し隙間を開けて表示していますが、実際には隙間はありません。

なぜ四面体メッシュが必要なのか

弾性体シミュレーションなどで、メッシュの内部に働く力をきちんと計算したいことがあります。その際に、三角形のサーフェイスメッシュ(つまり表面だけのメッシュ)では、中身が詰まっていないので風船のような弾性体としてしか計算できません。それに対し、四面体ボリュームメッシュ(つまり体積がきちんとある)では、弾性体の内部の変形でも体積が一定になるような計算が可能となるため、より詳細なシミュレーションが可能となります。

CG としての見た目のみが求められる場合は、内部のメッシュ構造を四面体で持つと冗長なのでメッシュは三角形で構成すると思います。その一方で四面体メッシュの情報を持つことで、メッシュ内部の詳細な力学計算が可能になるというわけです。

四面体メッシュは、Maya, Houdini でモデリングすることも可能でしょうが、モデリングが大変な場合は、三角形メッシュから四面体メッシュを自動生成する研究があり、近年では Fast Tetrahedral Meshing in the Wild などがあります。

シミュレーション用の四面体メッシュに関して

Fast Tetrahedral Meshing in the Wild の実装はロバストですし、入力メッシュの頂点を維持するなどの長所があり大変素晴らしいのですが、用途によっては機能が過剰です。その一方で Matthias Müller さんの実装は、ロバストさはないものの、シンプルで、なおかつ四面体の品質制御が可能であるという長所があります。

ここでの四面体の品質とは、四面体の体積が小さくなりすぎないことであったり、四面体が正四面体に近い状態を維持することであったりを指しています。このような品質は、シミュレーションアルゴリズムにもよりますが、体積の勾配を求めるようなアルゴリズムの場合有利に働きます。

コマンドライン引数

```
tetrahedralizer input_tri.obj output_tet.obj [resolution]
[min_quality_exp] [one_face_per_tet] [scale]
```

コンソールから、引数2つ以上をこの順で指定して実行する必要があります。引数が足りない場合は usage 表示を行い、再入力を促します。

```
引数1: 三角形メッシュ (例: input.obj)
Wavefront (*.obj) 形式

引数2: 四面体メッシュ (例: output.obj)
Wavefront (*.obj) 形式

引数3: 解像度 (デフォルト値:10 取りうる範囲: [0, 100] )

引数4: 最小品質指数 (デフォルト値:-3 取りうる範囲: [-4, 0] )

引数5: 四面体ごとに1 face 指定 (デフォルト値: true取りうる値: true
or false)
```

引数6: スケール (デフォルト値: 0.8 取りうる値[0.1, 1.0]) スケールは、引数5 が false のときのみ反映されます

例1: 最小指定
tetrahedralizer input_tri.obj output_tet.obj

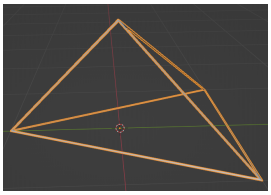
例2: 内部分割解像度指定のみ
tetrahedralizer input_tri.obj output_tet.obj 3

例3: 四面体品質も指定
tetrahedralizer input_tri.obj output_tet.obj 3 0

例4: デバッグ用に四面体ではなく三角形を出したいとき。隙間を強調するため 0.5倍スケールもかけている
tetrahedralizer input_tri.obj output_tet.obj 3 -4 false 0.5

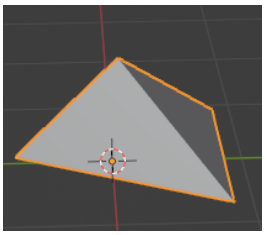
オプション詳細

引数1: 入力メッシュとして三角形メッシュを指定します。下記例は4頂点、4三角形の入力です。ファイル名をフルパス指定します。



```
v -1.000000 0.000000 -1.000000
v  1.000000 0.000000 -1.000000
v  0.000000 0.000000  1.000000
v  0.000000 1.000000  0.000000
f 1 2 3
f 3 4 1
f 4 3 2
f 4 2 1
```

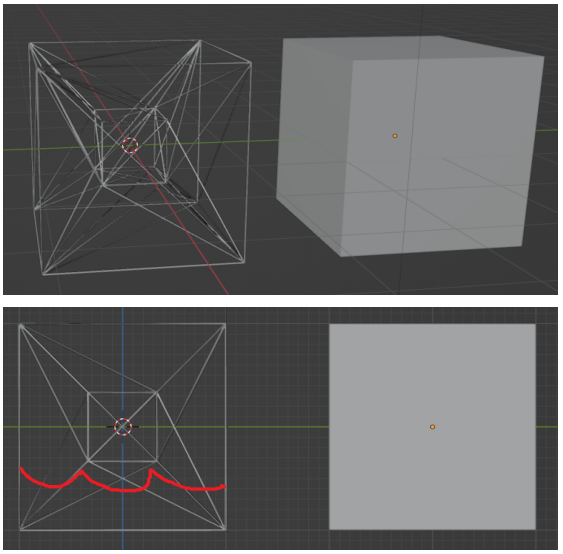
引数2: 出力メッシュとして、四面体メッシュのファイル名をフルパスで指定します。下記は4頂点、1四面体の例です。(このような .obj は四面体として正しくレンダリングできないソフトが多いので注意してください。)



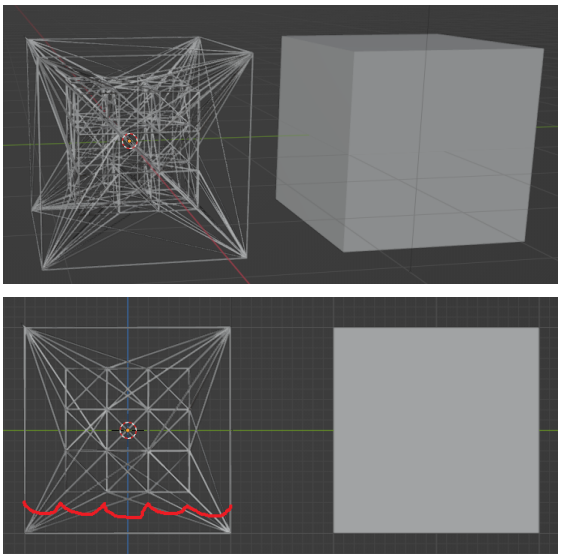
```
v -1.000061 0.000013 -1.000100
v 0.999996 0.000017 -0.999938
v 0.000065 0.000079 0.999970
v 0.000072 0.999935 0.000049
f 3 2 1 4
```

引数3: 内部解像度指定。整数指定で、0～100の範囲で指定します。下図のように、三角形サーフェイスメッシュの内部にこの解像度で分割し頂点を追加して、四面体を生成します。下図は内部を3等分および5等分した四角形です。

内部分割数3の場合



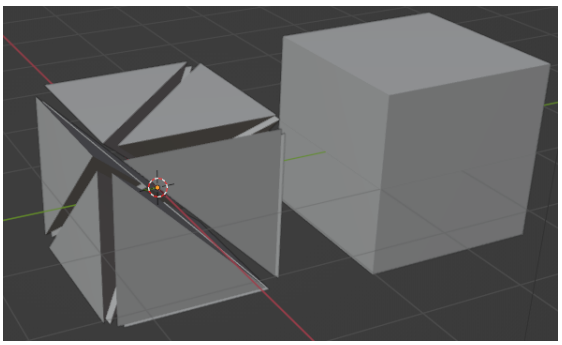
内部分割数5の場合



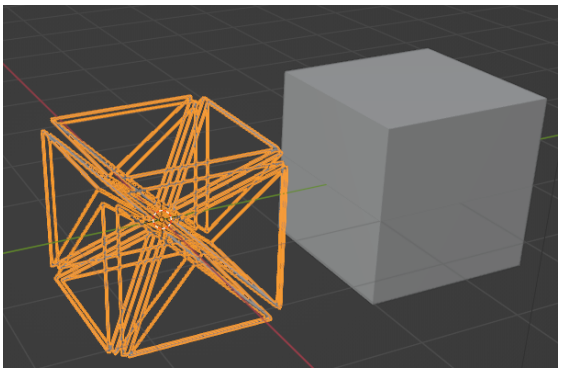
引数4: 最小品質指数を指定します。指数なので -4を指定した場合は、四面体品質として 10^{-4} の値以下の四面体が生成されません。四面体品質 $q_{tetrahedron}$ は、下記で計算されます。簡単に説明すると四面体の体積が小さいほど低くなり、正四面体に近いほど大きくなります。つまり体積 0 に近い四面体や、細長い四面体では $q_{tetrahedron}$ の値が小さくなります。参考:Air Meshes for Robust Collision Handling

$$q_{\text{tetrehedron}} = \frac{12}{\sqrt{2}} \frac{V}{l_{\text{rms}}^3}, \quad l_{\text{rms}} = \sqrt{\frac{l_1^2 + l_2^2 + l_3^2 + l_4^2 + l_5^2 + l_6^2}{6}},$$

引数5および引数6: 基本この2つはセットで指定します。引数5に false を指定した場合はデバッグ用に、四面体ではなく三角形を出力し、なおかつ引数6で指定したスケールでスケールをかけた四面体を出力します。下図で隙間が空いて見えるのはスケール 0.8 をかけて四面体を少し小さくしているからです。主にデバッグ用に可視化する用途です、



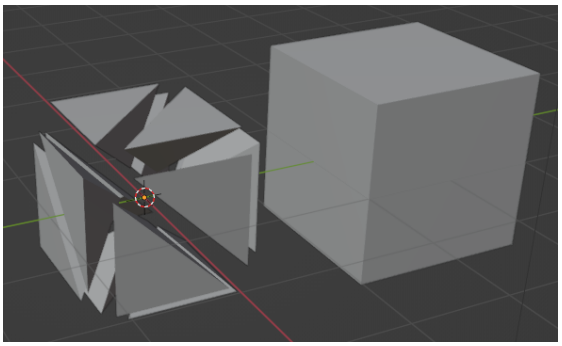
非常にわかりにくいのですが下図は三角形4つで構成された四面体形状です。



出力は三角形になっているので注意してください。4三角形で四面体形状を表現しています。あくまでデバッグ用の可視化用途です。

```
v -0.699968 0.499955 -0.900013
...
f 1 2 3
...
```

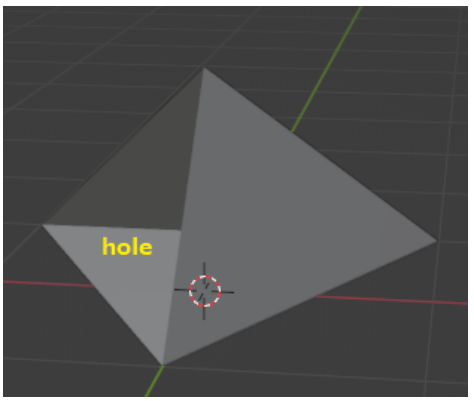
スケールに 0.6 を指定すると下図のように隙間が増えます



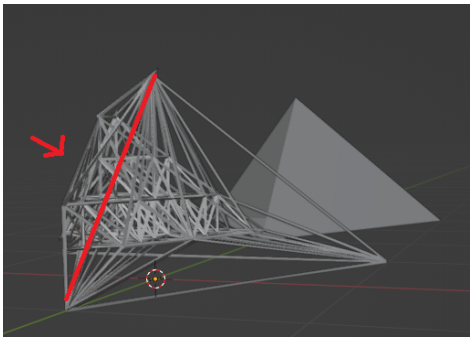
オリジナル実装の不具合と思われる箇所

追実装の過程で、Müller さんによるオリジナル実装に不具合と思われる箇所があったので、調査後に修正した。

問題になるケースは、下記の用にメッシュに穴が開いているケースのみで



メッシュの内外処理が-x軸方向は正しく行われていないため、不具合修正前では下記図のように、穴の外まで追加の頂点が追加され、その頂点を用いた四面体が生成されてしまい不自然な形状となってしまう。



メッシュの内外判定の不具合修正後は、穴の空いたメッシュでも穴の形状を残しながら、内部を詰めた四面体生成に成功するようになりました。