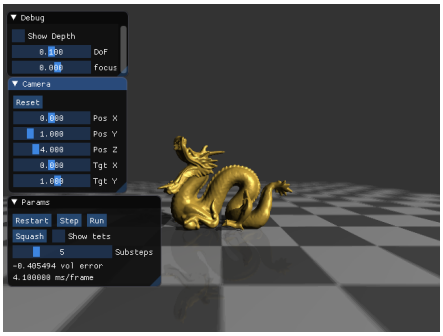
実装: A Constraint-based Formulation of Stable Neo-Hookean Materials

2022年8月23日更新 中川展男



目的

この文章では、論文: A Constraint-based Formulation of Stable Neo-Hookean Materials (2021) に関して、実際に動作するアプリケーションを用いて素早く理解することを目的としています。

配布物

- neohookean

windows(x64), macos(x64,arm64), ubuntu(x64) 版の実行形式を用意しています。

起動方法

./neohookean input.txt

引数にモデルを指定すると、外部モデルを読み込みます。引数なしの場合、オリジナルで提供されている dragon モデルを表示します。

操作方法

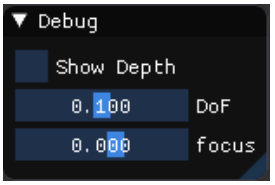
キーボード:
r: リスタート
s: スクリーンショット取得
d: デプスバッファ取得
esc: アプリケーション終了

マウス:
左ボタン: オブジェクト選択
中ボタン: カメラズーム
右ボタン: カメラ操作

パラメータ操作

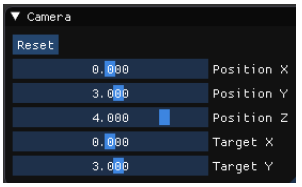
Restart: リスタート(r キー同様)
Step: ステップ実行、押すごとにフレーム進める
Run: 通常実行に切り替える
Squash: メッシュをランダムに潰す
Show tets: シミュレーション四面体メッシュ表示
Show tri: 描画用三角形メッシュ表示
Substep: Substep XPBD での物理シミュレーション全体の分割数を指定
use friction: 摩擦を使うかどうか
friction: 摩擦の大きさ
vol error: 体積の定常状態からの値からの誤差
ms/frame: フレームごとの経過時間(シミュレーション時間のみ)
xxx tet, xxx tri: シミュレーション四面体数、描画用三角形メッシュ数
dev_comp.: 偏位コンプライアンス係数
vol_comp.: 体積コンプライアンス係数

デバッグ機能



Show Depth: デプスバッファ表示
DoF: 被写界深度強度
focus: 被写界深度フォーカス位置

カメラ操作



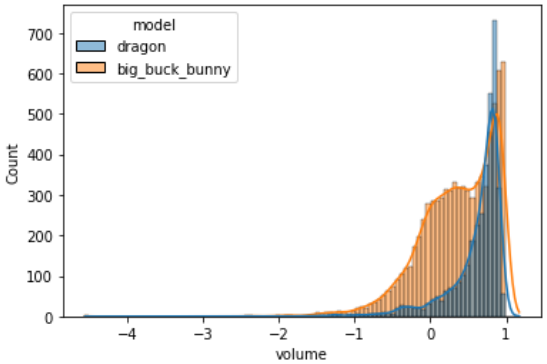
Reset: カメラ位置初期
その他はカメラ位置と注視点を指定できます。

実験

追実装
ひとまず筆者提供の js 版の完全な移植を試みた。先行研究と同等の機能が無条件安定を維持しながら極めて高速に実現されていることがわかる。数学・物理的な背景は難しいが最終的に極めてシンプルなコードで実装できることも理解した。

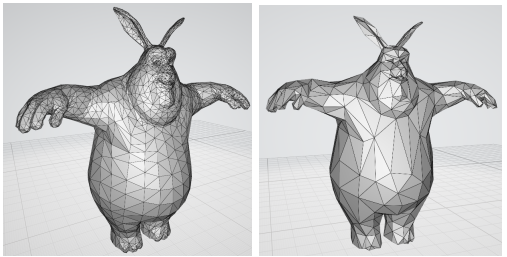
メッシュ読み込み
fTetWild を用い、任意の三角形メッシュから、四面体メッシュの自動生成を行い、更に自前のコンバータで、四面体を描画用メッシュのエンベロープとみなして描画用の三角形メッシュとの対応をウエイトとして書き出し、更にその情報を読み込めるようにすることで任意のメッシュの読み込み対応を行った。メッシュの形状によって、安定するまでの substep に違いが出ることがわかった。

四面体メッシュの体積**(volume)**の評価

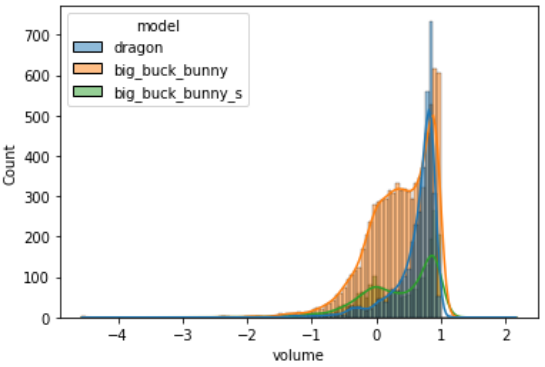


筆者提供の dragon モデルに比較して、自前の fTetWild を用いたワークフローで生成した big-buck_bunny モデルの四面体は収束までに必要な substep が大きい事がわかった。原因を調べるために四面体の体積のヒストグラムを上記グラフで比較したところ big_buck_bunny は体積(volume)が0に近いメッシュ数の数が多い事がわかった。

荒い四面体での実験



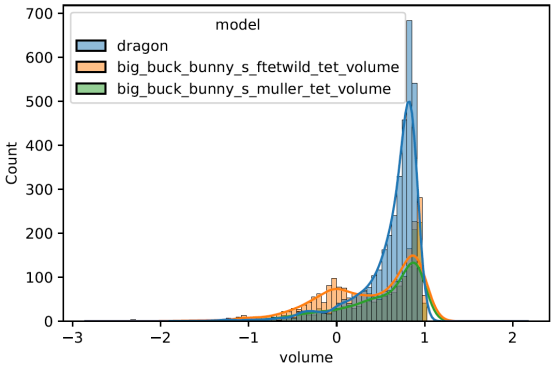
Blender の Decimate モディファイヤで、5278 triangles → 1054 triangles で荒い入力用三角形メッシュを用いて比較した。



同様に四面体の体積を比較したがモディファイヤのアルゴリズムの都合もあり、big_buck_bunny(オリジナル) と big_buck_bunny_s(荒くしたもの) ヒストグラムの形は変わらず、そのままメッシュ数が減るような結果になっている。結果的に荒くすることで収束までに必要な substep 数は減っているが計算品質が下がっただけなので効率としては dragon メッシュのように体積の少ない四面体を極力作らない方向のリメッシュが好ましいだろう。

Müller さん提供 **Tetrahedrizer**
論文公開から半年近く後(2022年3月19日) に last author の Matthias Müller さんの web サイト上で、Blender のアドオンとして、独自の Tetrahedrizer の実装が公開された。tri2tet の実装を拡張し、このアドオンで生成した四面体メッシュを Blender 上から .obj でエクスポートした四面体を入力として受け取れるようにし、比較実験を行った。

四面体の体積のヒストグラム



dragon: 公開デモ実装で公開されているモデル。
big_buck_bunny_s_muller_tet_volume: Müller さん提供 Tetrahedrizer で作られたモデル。
big_buck_bunny_s_muller_tet_volume: fTetWild で生成された四面体モデル。
fTetWild で生成されたモデルと比較して、Müller さん提供 Tetrahedrizer でエクスポートされた他の2モデルは、体積0近辺の四面体数が少ない様子がグラフから読み取れる。これは、Müller さん提供

Tetrahedrizer では、四面体の体積と辺の長さから四面体品質を求め、品質が低いメッシュを取り除く処理が入っているためであると考えられる。この処理は今回の用途には向いていると考えられる。なぜなら、荒い四面体メッシュを用いて、描画用三角形メッシュを変形させる処理(Tetrahedral Skinning)が用いられているため、四面体のメッシュは荒いほうが高速に計算できるからである。