三次元のメッシュ図形における体積計算方法の確立

ソリューション本部 第二本部 開発二部

齊田　和隆

1. はじめに

建築関連のアプリケーション開発において体積計算をできないかという要望があった。三次元モデルの体積計算であれば数学的な手法を用いて実現できるであろうという見通しから検討を進めることとなった。検証にあたっては、三次元ライブラリに Three.js を使用した TypeScript により WEB アプリケーションを開発した。

2. 体積計算アルゴリズムの選定

まずはアルゴリズムの選定から行う事にした。簡単な図形の体積を求めるだけであれば、立方体や多角錐の体積計算の公式はあるが、今回の場合は図形の形状が特定できず、どんな形状でも処理できる必要があった。また対応できる形状を順次追加して機能の拡張を行っていくのではなく 1 つの手法でどんな形状の体積でも計算できる汎用性の高い手法を選択する必要性があった。

この条件に合うアルゴリズムを選定してプログラムを構築することとなった。

3. ベクトルによる体積計算と問題点

今回対象にしている三次元モデルは三角形の集合体であるため、三次元空間でそれぞれの三角形と原点からなる三角錐の体積を計算したものを合計して、三次元モデルの体積を求める方法を採用した。（※計算方法の詳細な仕様は文末の参考ページ 1 参照）

これであれば比較的容易に三次元モデルの体積計算が可能であると考えられたが、プログラムを実装してサンプルの三次元データを使用して計算したところいくつかの問題点が発覚した。

3．1　計算時間の問題

1つ目は予想以上に計算対象の三次元モデルを構成している三角形の数が多いものがあることであった。当初は建築部材の体積という事だったため、大半は柱などの単純な形状であると想定していた。ところが実際には柱部材も凹凸などがあり、また床などはかなり複雑な形状が多くあった。本来はこういったことも想定してどんな形状でも計算できるアルゴリズムを検討していたため、形状が複雑であるという点は問題なかったが、計算時間がかかりすぎるという点においては想定外であった。計算速度について改良が必要であった。

3．2　不正な形状の問題

2つ目は、不正な形状が存在するということであった。三次元モデルは三角形の集まりであり、体積計算できるためにはその三角形が隙間なく定義されている必要がある。しかし実際のサンプルでは不正な形状も多く、正しい計算結果が出ないものが無視で

きない程度存在した。不正な形状があるデータ側に問題があるのだが、それで「計算できません」とエラーを出して終わりというのではこの体積計算のライブラリは使えないと判断されてしまう。

修復可能な不正モデルについては修復した上で体積計算するように改良する方向に進めた。

3. 3　形状の位置関係の問題

3つ目は1つの三次元モデルが１つの形状で無い場合があり、しかもその図形が干渉している場合があった。パターンとしては「干渉なし」、「包含」、「交差」の３パターンであった。

これらについてはそれぞれ別の形状として取り扱う必要があったが、三次元モデルは三角形の集まりであり、即ち座標点の集まりでしかないためにそれぞれ別の形状であることを判別する手法から検討する必要があった。

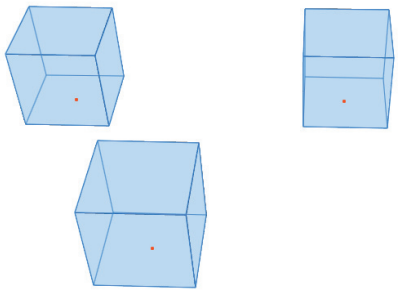

図 1 「干渉なし」の例
（3つの独立した立方体で1つの形状となる）

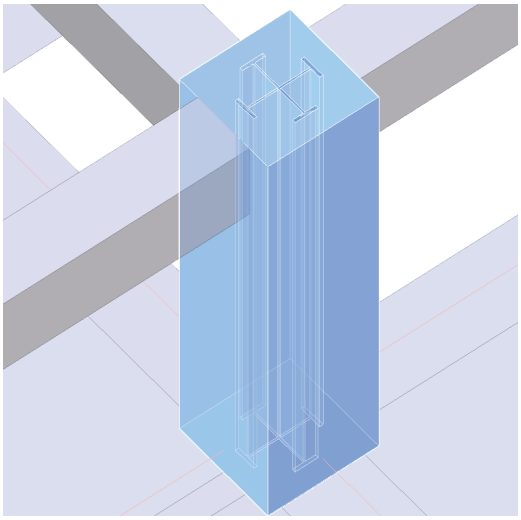

図 2 「包含」の例
（外側の四角柱と内側の十字形の2つの図形で1つの形状）

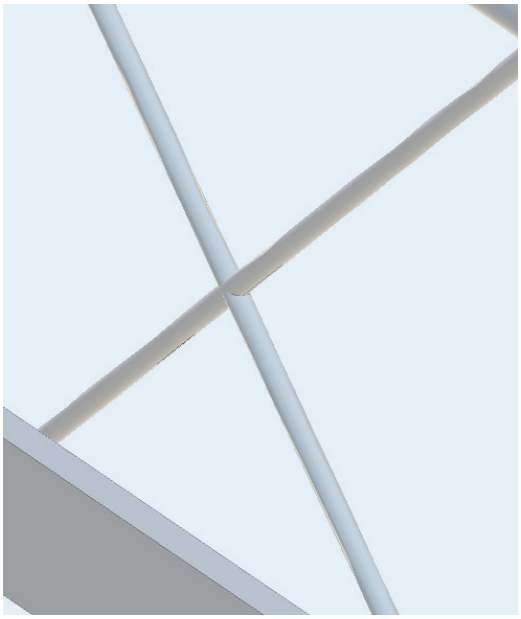

図 3 「交差」の例
（中央十字で交差している2つの多角柱で1つの形状）

4. 問題点の解決方法

それぞれの問題点の解決方法として次の対応を実施した。

4. 1　計算時間の短縮

まず単純に計算ロジックの見直しを行い、無駄な計算や計算順の見直し、逆引き用の配列を用意するなどして高速化を図った。これによってある程度の高速化はできたが、TypeScript言語ではそれでも限界があり、WebAssanbly を使用してさらに高速化を行う事とした。この対応により計算時間はほぼ問題ないレベルまで高速化することができたが、その代償としてメンテナンス性の低下が発生した。バイナリコードにすることによって高速化しているためエラーが発生した場合の不具合特定に時間がかかるというものである。エラー箇所は中間言語で提示されるため、バイナリコードのままよりはましであったが、WebAssanbly化する場合は小さいモジュールの集合体にするなど1つ1つのプログラムサイズを小さくしてエラー時の解析を容易にする必要性があった。

4. 2　不正な形状の修復

三次元モデルの基本的なルールとして構成される三角形の面には表裏が存在する。その判別方法は三角形を構成する三頂点を結ぶ順番になる。三角形を構成している頂点を a,b,c とすると(a,b,c)と定義するのと(a,c,b)と定義するのとで三角形のループの向きが逆になり、頂点は同じだが向きが逆の三角形となる。

また体積計算ができない三次元モデルには下記のものがある。(詳しくは文末の参考ページ 2 参照)

① 頂点が重複している
② 孤立した三角形がある
③ 三角形に隙間がある
④ 三角形のループが逆のものがある

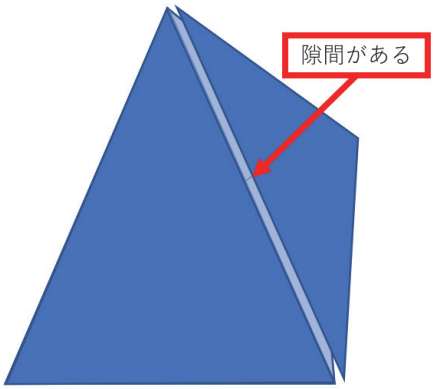

図 4　隙間のある不正図形の例

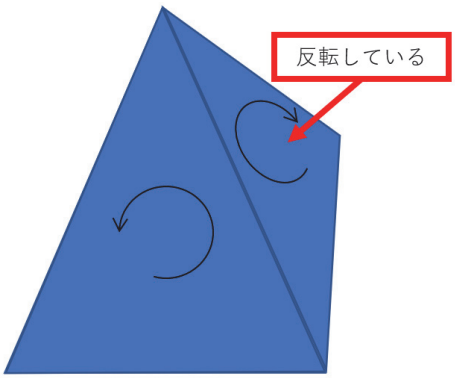

図 5　面が反転している不正図形の例

以下に、不正な状態をそれぞれ解消していく過程について述べる。

(1)　頂点が重複しているもの

重複した頂点については全く同じ頂点の場合は統合するだけで良いが、それだけではあまり効率的に形状を修復できなかった。倍精度小

数点のデータのため、わずかな誤差のために別の頂点として扱われてしまうケースがあるためである。閾値を設定して閾値未満の差の頂点は同じ頂点として頂点の統合を行った。

また上記の頂点の統合を行ったことにより発生する不正な三角形（例えば下記のようなもの）が存在する。

① 三角形の頂点で同じものがある
② 三角形の全ての頂点が同じ直線上にある

これらはつぶれた状態の三角形でそれらの三角形は削除する。

(2) 孤立した三角形があるもの

孤立した三角形については、完全に孤立したものと三角形の一辺のみ他の三角形と共有しているものは不要な三角形として削除した。二辺を共有している三角形の場合は、残り一辺の三角形部分が抜けている場合が考えられるため残した。

次に隙間のある三角形の場合はその隙間を埋めるために三角形を追加する。ここでの三角形の隙間の修復は、まず三角形の三辺の内の一辺がほかのどの三角形とも共有していない辺を抽出する。次にその辺をつなぎ合わせたループを抽出する。これは複数見つかることがある。そのループの中が抜けている面部分となるためここに三角形を追加して隙間を埋める。

(3) 三角形のループが逆のものがあるもの

三角形のループが逆のものについては正しい向きにする。三角形のループが逆である判別方法は正しい閉ループになっている場合、三角形の辺を a-b とした場合、a-b の向きを持つ三角形と b-a の向きを持つ三角形が一組ある状態になる。これが a-b が二組や b-a が二組になっているとどちらかが逆ループになっているとなる。三角形は三辺あるので残りの辺も判定して三辺とも逆ループを持っている場合にその三角形が逆ループということになる。ただし複数の三角形でまとめて逆ループで登録されていると単純に一つの三角形の比較だけでは判断がつかない場合もある。

4.3 形状の位置関係の解消

1つの三次元モデルに複数の形状がある場合の対処として、まずは複数の形状に分かれているのかどうか判別する必要がある。これは不正形状の修復の際に使用した頂点の共有と辺の共有からいくつかのグループに分類でき、それがそのまま別形状として扱える（実は例外もあるが、ここでは割愛する。）。

次に「干渉なし」、「包含」、「交差」のどれにあたるか判別する必要がある。これに関しては構成面と三角形の辺の交差判定を全てに対して実施してその結果から取得する。交差判定で交差なしとなった場合が「干渉なし」と「包含」のどちらかであり、「包含」の場合は内側の形状はどの方向に対してラインを伸ばしても必ず外側の形状と奇数回交差する。「干渉なし」の場合は偶数回になる。

それぞれのパターンが判明すれば、「干渉なし」の場合は別々の形状として扱うだけで良く、「包含」の場合は、内側の形状を無視するか内側の形状分だけ体積を除く（内側の形状は空洞にしたい場合）かする必要があるが、それは業務仕様によるのでライブラリのオプションで設定すればよいと考えている。

問題は、「交差」の場合でこれは重複部分の体積を除く必要があるが、そのためには重複部分を判別

する必要がある。または交差した図形を 1 つに結合してから体積計算を行う方法もある。図形の結合などを行うためにはどうすればよいか、今回の検証ではブーリアン演算を使用することにした。

4. 4　ブーリアン演算

ブーリアン演算を行い 2 つの形状を結合することは比較的容易に実施することが出来た。しかし、この手法には致命的な問題が潜んでいた。それは面数の少ない形状同士でも結合を行って出来上がった形状の面数は爆発的に増えてしまうというものである。頂点数 100 程度の面同士を結合した場合でも数万の面数になることもあり、その場合は前章で実施した高速化を行った場合でも高速化前並みに計算時間がかかるようになってしまった。形状の大きな体積部分が交差している形状の場合はまだいいが、一部が少し交差しているだけの形状の場合は体積の差はほとんど発生しないのに対して計算時間だけが膨大になってしまうという結果になってしまった。

これに関しては現在も解決方法が無く検討を続けている問題になる。

5. 新たな問題点

それぞれ解決したと思っていた箇所でもまだ問題が残っていることが分かってきた。

5. 1　高速化の考慮漏れ

WebAssanbly を使用して高速化したが扱える配列数に上限があり、面数が異常に多い形状の場合に逆引き用のマッピング配列がサイズオーバーでエラーになるという問題が発覚した。これはブーリアン演算でできた結合形状の面数の多いもので発覚した。こちらに関しては逆引き用のマッピング配列を使用しない方法で回避は可能であったが、そうすると処理速度がかなり低下する結果となってしまった。大多数の形状の処理速度が犠牲になってしまうために面数が多い場合だけ逆引き用のマッピング配列を使用しない方法を使用することにより全体的なパフォーマンスを維持するようにした。

5. 2　形状の位置関係の考慮漏れ

形状の位置関係については「干渉なし」、「包含」、「交差」の 3 パターンで解決と思っていたが、「包含」について多重に発生するパターンの考慮が漏れていた。包含された形状の体積を無視するオプションの場合は問題ないが、体積を除く場合はその中にまた別の形状があるためにその形状については加算しないといけないとなると包含された順番の判別も必要になり、その中に「交差」も混じってくるとさらにパターンが増えていく結果となる。

このパターンが増える状況は最終的に対応しきれずに破綻するパターンになるので避ける方向での検討となる。

5. 3　三次元モデル同士の位置関係の考慮漏れ

今までは三次元モデル内での個々の形状での位置関係について検討していたが、実際にはデータとして読み込まれた全ての三次元モデルの位置関係について「干渉なし」、「包含」、「交差」の判定結果の体積計算が必要となる。こちらは三次元モデル内での位置関係を拡張して考えればよいが、その問題の解決方法が確立できていないままとなっている。

6. おわりに

三次元モデルの体積計算だけなら容易であろうと思っていたが、実際に検討すると、安易に不正形状をエラーとして体積計算しないのではなく、修復できる形状は修復しようとしたために難易度が上がってしまった。

この先の検討としてはエラーとなる形状はごく一部のものであるため、計算結果としてエラーとなったものを明確に認識できるようにしてその分はデータ修正してもらう。また修復できた形状についても修復結果が正しいものであるか判断が必要になるため修復後の形状を表示して判断を行えるようにして、完璧ではなくても一般使用レベルで使えるものとする。またエラーの出た箇所が容易に特定でき、その修正を促すことができるものを目指したいと考えている。

＜参考ページ＞

1) 「STL ファイルの体積計算」
https://recruit.cct-inc.co.jp/tecblog/com-geometry/stl_volume/

2) 「STL ファイルのエラーチェック」
https://recruit.cct-inc.co.jp/tecblog/com-geometry/stl_error_check/