

東京大学大学院新領域創成科学研究科

社会文化環境学専攻

2018 年度

修 士 論 文

Fuzzy Node で接合される小片集積体の機構解析における

行列表現による解法

Matrix Solution to Mechanism Analysis Equation
for Aggregated Structure Connected with Fuzzy-Nodes

2019 年 1 月 21 日提出

指導教員 佐藤 淳 准教授

河村 京介

Kawamura, Kyosuke

内容

第1章 序	1
1.1 研究背景	2
1.2 Fuzzy Node の実例	5
1.3 Fuzzy Node の実現化	8
1.4 既往研究	10
1.5 本論文の目的	11
1.6 本論の構成	12
第2章 座標系の設定	15
2.1 平面座標系の設定	16
2.2 空間座標系の設定	19
第3章 Fuzzy Node で接合された2部材の解法	21
3.1 解を一度の計算で求める方法	22
3.2 解を反復計算で求める方法	44
第4章 複数の Fuzzy Node を持つ構造体への応用	47
4.1 解を一度の計算で求める解法の応用	48
4.2 解を反復計算で求める解法の応用	57
第5章 部材の挙動の可視化	63
5.1 制約近似法	64
5.2 重心追従法	72
5.3 複数の Fuzzy Node で接合された集積体の可視化	79

第6章 立体への拡張.....	87
6.1 重心追従法の立体への対応.....	88
6.2 Fuzzy Node で直鎖状に接合された部材の挙動の可視化.....	90
第7章 結.....	91
7.1 本論文での成果.....	92
7.2 今後の課題.....	94
参考文献.....	95

第1章 序

1.1 研究背景

以下のような小片集積体（例：図 1.1.1, 1.1.2）は、小片同士の接合具にわずかな自由度を与えることで構成されている。わずかな自由度の存在が複雑な全体形を可能にする一方、その複雑さからコンピュータでの管理が難しい。実際、これらを制作したときの図面は存在しない。はじめに目標とする形を決め、その目標形と同じになるように新しい小片を適宜取り付けながら作成している。これらの集積体は図面がないため作成前に構造解析を行うことが現状では不可能なので、実際の建築への応用が困難になっている。



図 1.1.1 i-cube の全体像 ¹⁾ (<http://world-architects.blogspot.com/2013/04/i-cube.html> より抜粋)



図 1.1.1 超薄板強化ガラスのインスタレーションの全体像 ²⁾ (<http://a-plus-e.blogspot.com/2015/06/agc-studio.html> より抜粋)

わずかな自由度の存在が複雑な全体形を可能にし、設計の多様性を生じる理由について説明する。このとき、単純化のため小片は正方形の二次元モデルとし、その小片によって構成される集積体も二次元モデルであるものとする（図 1.1.3, 1.1.4）。

図 1.1.3 のように小片で集積体を設計する場合、新しい小片を集積体に付け加えるとき、接合具にわずかな自由度がない状態では、小片を接合したい2点の距離が加える小片の一辺の長さと同しくないと付け加えることができない。

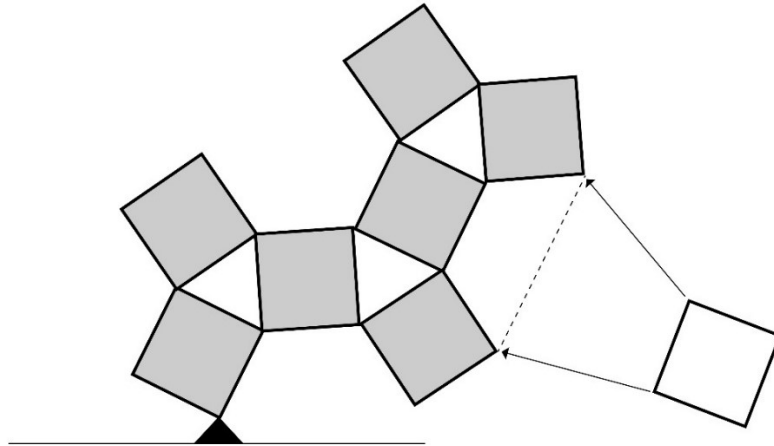


図 1.1.3 小片を取り付けられない様子

しかし、小片の接合具の形状にわずかな自由度を持たせると、各々の小片がわずかに動くことができるため、集積体全体の自由度を大きく増すことが可能になる。このとき、集積体が少し変形し、先ほどまではつけることができなかった小片を図 1.1.4 のようにつけられるようになる。

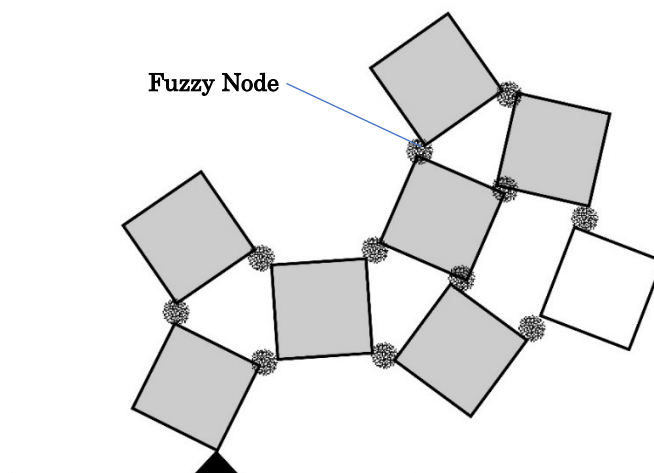


図 1.1.4 小片を取り付けた後の集積体の様子

本論ではこのような少しの自由度を与えた節点のことを Fuzzy Node と呼ぶ。

Fuzzy Node が存在するのは全体形的设计時にのみである。全体形的设计が終了し、小片の配置が確定したら、小片をその位置に固定させるような接合具を设计する。设计された接合部は全て異なるが、3D プリンターやロボットアームなどの近年の技術の発達を使用すれば施工性を損なわずに実現できる。図 1.1.5 のように小片の接合の形態によって使用している接合具が異なっていることが確認できる。

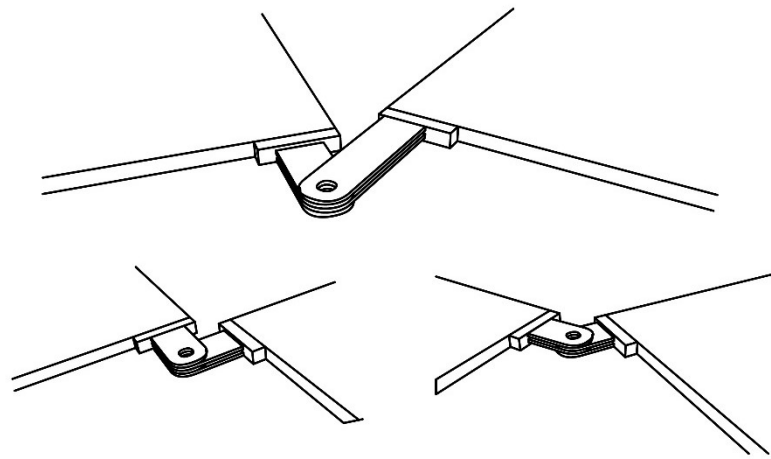


図 1.1.5 Fuzzy Node の異なる接合部の様子

Fuzzy Node の存在によって集積体の自由度が増えるため同一の小片だけでもより複雑な形態を構築することが可能になる。そのさい、Fuzzy Node があることで小片同士の接合箇所数が増加し、より安定的な構築物になる。Fuzzy Node の機構を単純な行列で表現し、その解を求めることができれば、集積体のとりうる形状を容易に知ることが可能になる。さらに、コンピュータで解を求める手法を確立できると、集積体をコンピュータ上で設計することが可能となり、施工前の段階で構造計算を施すことができるようになる。これにより、今まではパビリオンでしか実現可能でなかった Fuzzy Node を持つ集積体をより大きなスケールの建築に応用できるようになると推測される。

1.2 Fuzzy Node の実例

Fuzzy Node を使用する構築物の接合具の形状が異なるものと同一なもの2つの例を紹介する。

1つは接合具の形状を同一なものにし、Fuzzy Node を実現した例です。2013年日本大学の日大横河研究室によって構築された「*i*-cube (アイ・キューブ)」である。*i*-cube は木製のキューブの8つの頂点にヒートンと呼ばれるフック状の金具をつけ、ヒートン同士を引っ掛けることでキューブの集積体を形成したものである。キューブ一片の長さの1倍、 $\sqrt{2}$ 倍、 $\sqrt{3}$ 倍のモジュールで形成された立体トラスとみなすことも可能である。集積体に新たなキューブを取り付ける際は上記の3つの距離のどれかだけ丁度離れている2点を探さなければならないが、実際にそうなっている2点はほとんど存在しない。しかし、この構築物ではヒートン同士の接合部には少しの自由度があるため、小片を接合したい2点を引き寄せると集積体の節点が少しずつ調整され、新しいキューブをつけることができる距離に2点が近づくことが体感できる。また、取り付けたキューブがその位置に固定されることも体感できた。この構築物ではヒートン同士の隙間が Fuzzy Node の役割を果たしている。



図 1.2.1 *i*-cube の全体像^[1] (<http://world-architects.blogspot.com/2013/04/i-cube.html> より抜粋)

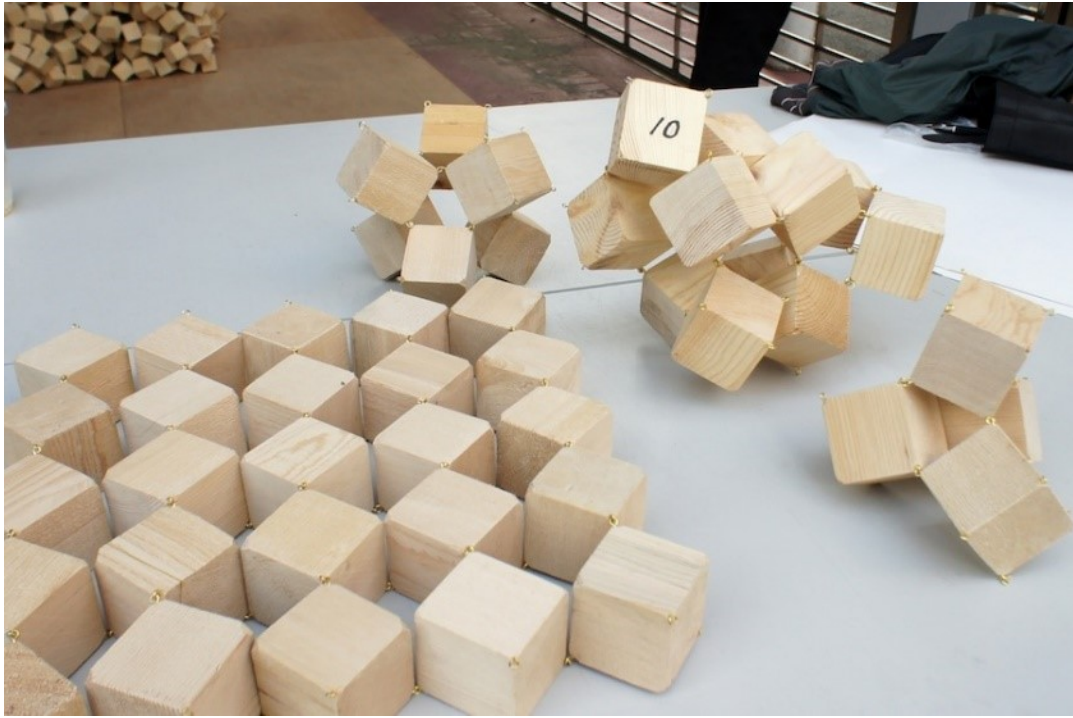


図 1.2.2 ζ -cube の接合部 ^[1] (<http://world-architects.blogspot.com/2013/04/i-cube.html> より抜粋)

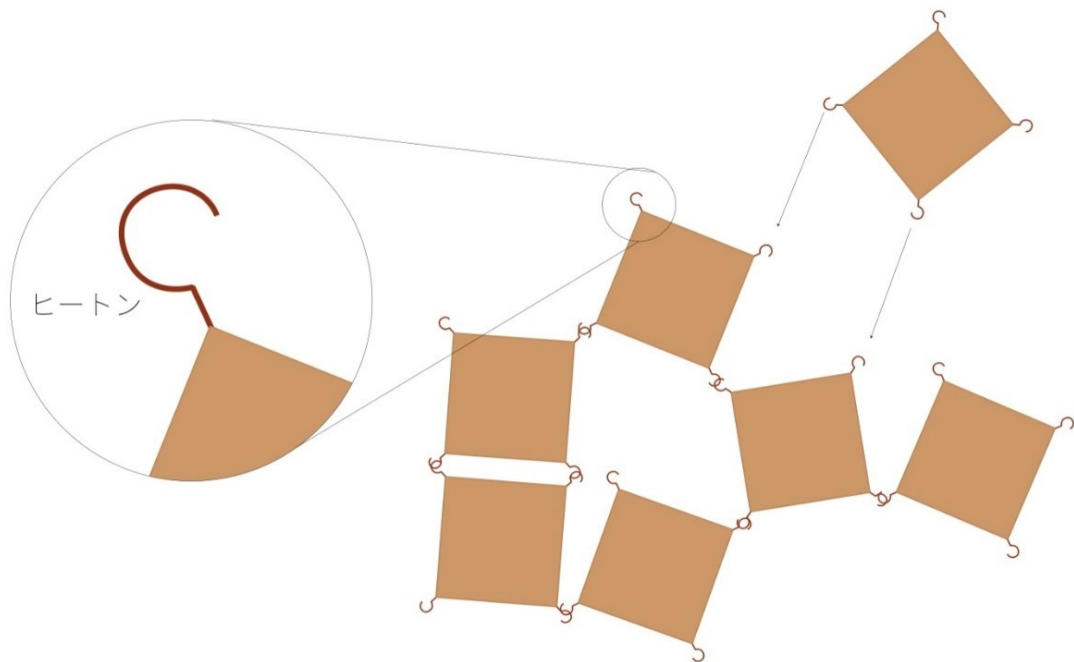


図 1.2.3 ζ -cube の模式図

もう1つは接合具の形状が接合箇所によって異なる例である。2015年に東京建築構成材デザイン工学 (AGC 旭硝子) 寄付講座インストール「構造の透過性から生み出される浸透空間」で構築された超薄板強化ガラスによるインストールである。これは周囲に穴の開いた四角形のガラスパネルをアルミの帯で接合して構成されている。アルミの帯は人の力で容易に変形できるほど十分に薄くなっている。新しいガラス板をつける際は3点以上で固定する必要がありますが、アルミの帯を適切な形に曲げることで3点を容易に確保できることを体感しました。このインストールにおいてはアルミが Fuzzy Node の役割を果たしているといえる。



図 1.2.4 超薄板強化ガラスのインストールの全体像 図 (<http://a-plus-e.blogspot.com/2015/06/agc-studio.html> より抜粋)



図 1.2.5 超薄板強化ガラスのインストールの接合部 図 (<http://a-plus-e.blogspot.com/2015/06/agc-studio.html> より抜粋)

1.3 Fuzzy Node の実現化

Fuzzy Node が実現化し、実際の建築に使用されたときの、イメージを示す。

この図 1.3.1 は正方形の小片を Fuzzy Node で接合し、設計されたファサードを使用している。二階は遮蔽物がなく日差しが強いので全面がファサードに覆われている。一方、一階は入り口があるため、開放感を演出するため入り口付近はファサードで覆っていない。一階から二階まで曲線に沿って滑らかにファサードが繋がっているのは Fuzzy Node によるものである。また、小片の配置が不均一なので光の抜け方にばらつきが生まれ、木漏れ日のような効果を生み出している。



図 1.3.1 Fuzzy Node をファサードに使用した参考例（作成：小島慎平）

研究背景で Fuzzy Node の存在によって構築物の強度が上がると、述べているがその理由について説明を行う。

同じデザインのファサードを Fuzzy Node を使用するものと使用しないものの 2 種類で比較する(図 1.3.2)。このモデルは二次元であり、正方形の小片から構成される。上のモデルは Fuzzy Node を使用していないもの、下のモデルは Fuzzy Node を使用しているものである。モデルの赤い丸は接合していない節点を表している。両方の全体モデルにおいて外周上に存在する節点には赤い丸がついており接合していないことが分かる。全体モデルの内部に存在する節点を比較する。白い点線で囲まれた部分に着目すると Fuzzy Node が無いときには接合できなかった節点同士が Fuzzy Node の存在によって接合していることが確認できる。上モデルでは内部にある接合していない節点が 10 個であるのに対し、Fuzzy Node がある下のモデルの場合では 4 つと半分以上減っている。このように、Fuzzy Node によって接合箇所が増え全体の強度が上がる。

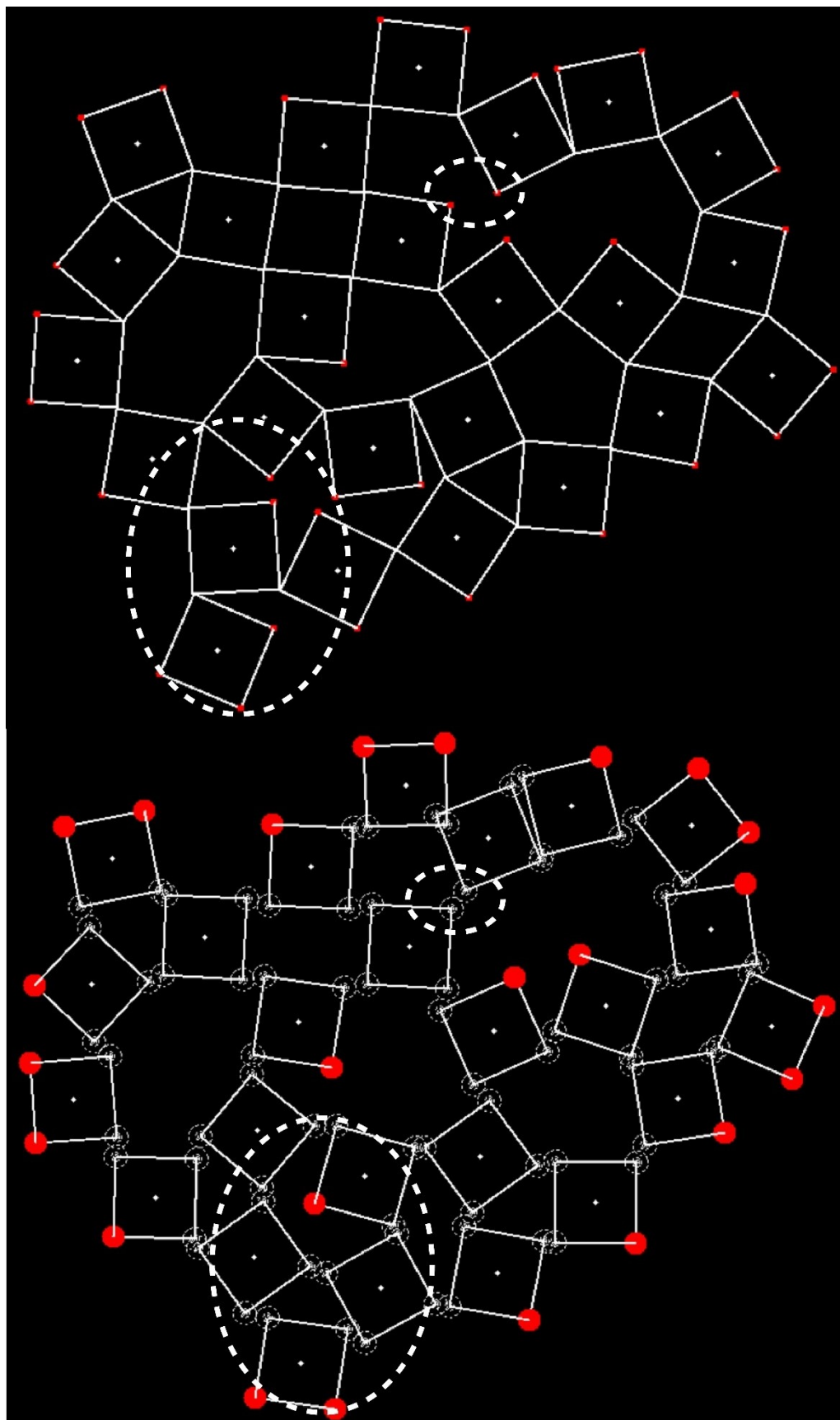


図 1.3.2 (上) Fuzzy Node なしの平面モデル、(下) Fuzzy Node ありの平面モデル

1.4 既往研究

「Fuzzy Node 接合形式で形成される集積形態の制御方法の基礎的研究」 [3]

この研究では多数の正方形剛体が三角形を形成しながら連結したモデルにおいて、2 節点間の距離の最大値・最小値を求める手法が示されている。

「Fuzzy Node で接合される小片集積体の機構解析における行列表現」 [4]

この先行研究では正方形剛体が2つ接合されたモデルにおいて Fuzzy Node がその行列表現に与える影響とその Fuzzy Node によって生じる可動域の可視化が提示されている。また、剛体が直鎖状に Fuzzy Node を介さずに接合されているときの行列表現の方法も明らかにしている。

1.5 本論文の目的

本論の目的は小片で集積された全体形を接合が Fuzzy (曖昧) なままに設計できるようにすることである。

本論で扱う Fuzzy Node には、わずかな自由度を与えるため、長さや角度を可変とし、自身の最大長を小片一辺の長さの $1/10$ 程度とする。また、1つの節点に最大で2つの小片までしか接合できないものとする。

本論では議論を単純化するため幾何学的な検討段階までを考慮するものとし、重力や風荷重等の力学的な作用は無視する。必要ならば、これらの外力が集積体に及ぼす影響は別途検討すればよい。

どのような形状の小片であっても同一なものであれば Fuzzy Node を持つ集積体として成立しますが、本論で扱う小片は正方形および立方体とする。また、正方形および立方体の頂点を節点とする。

平面において、小片2つが Fuzzy Node で接合されているときの可動域の可視化は既往研究で扱われている。本論では、2つの小片が Fuzzy Node で連結されている図 1.5.1 のようなモデルにおいて、上にある小片の節点に変位を与えたとき、各小片の変形を求める方法を提案する。

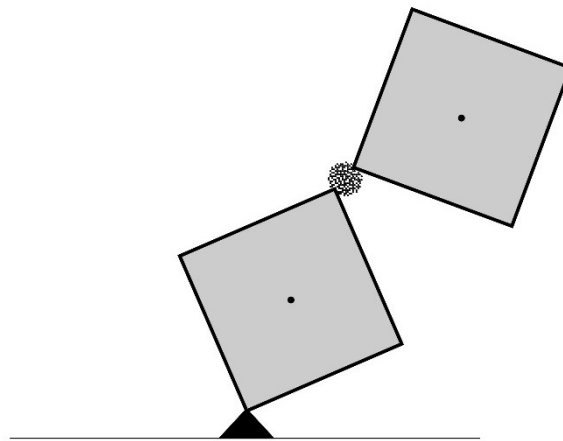


図 1.5.1 2つの剛体が Fuzzy Node で連結されているモデル

その後、小片が2つのときに提案した解法を複数の小片で構成されるモデルへ適用する方法の提案、及び三次元への拡張法を可能な解法では提案する。

1.6 本論の構成

図 1.6.1 に本論の研究の流れを表す。既往研究^[4]で扱われた Fuzzy Node で接合された 2 部材の行列表現の解法、および適用可能性を示す。

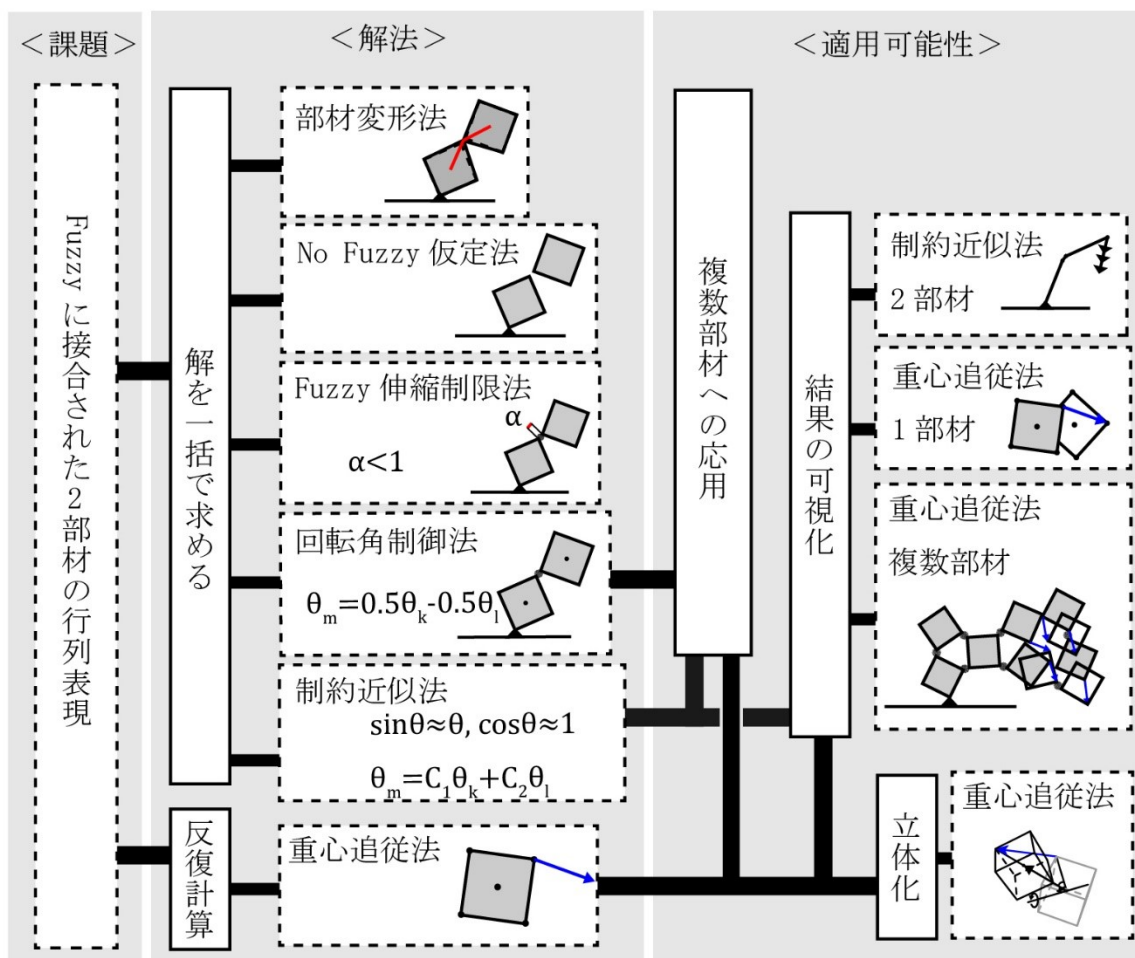


図 1.6.1 本論の研究の流れ

上記の図を参考にしながら本論の構成について述べる。

第 1 章 Fuzzy Node の特徴を、実際に使用されているパビリオンを用いながら説明し、Fuzzy Node を実際の建築へ適用するメリットを考察する。また、本論で扱う Fuzzy Node の特徴を明確にする。

第 2 章 本論で使用する平面座標と空間座標について説明する。平面座標については既往研究^[4]での定義を説明し、さらに追加で定義した関数を述べる。その後、空間座標の設定方法を説明する。

第 3 章 Fuzzy Node で接合された 2 部材の挙動の解法について考察する。図 1.6.1 に記載されている 6 種類の解法を検証し、それぞれの解法の特徴を比較し考察する。

第4章 第3章で提案した解法の中から解を求められたものを複数の部材にも適用する方法を検討する。その結果、複数部材への拡張が可能なものはその方法を示す。

第5章 第3章で提案された解法の描画を試みて、解法の有効性を確認し、考察する。また描画するにあたり使用しているソフトウェアやコードについて記す。

第6章 これまで二次元で扱ってきた解法を三次元に拡張する方法について考察する。また、三次元化したものを描画し、解法の有効性を確認する。

第7章 本論文の結論に加え、今後の課題と展望を述べる。

第2章座標系の設定

2.1 平面座標系の設定

2.1.1 一要素の平面座標系

本論では計算を単純化するため、まずは二次元で扱うものとする。本論で使用する座標系を説明する。絶対座標系と相対座標系の2つを使用する。絶対座標系はすべての部材に共通する座標であり、全ての部材に対して同じ原点と座標軸を持つものである。一方、相対座標系はそれぞれの部材で固有な座標系である。図 2.1.1.1 のように絶対座標系の座標軸を X 、 Y とし、部材 n の相対座標系 n の座標軸を ${}_n x$ 、 ${}_n y$ とし、相対座標系 n の原点 O_n を部材の重心の絶対座標 $({}_n X_0, {}_n Y_0)$ とする。このとき、2つの座標軸 ${}_n x$ と X のなす角度を θ_n とする。文字の左下に書いてある数字はその座標が所属する座標系を表す。文字の右下に書いてある数字はその座標が所属する座標系の何番目の点であるのかをあらわし、0 となっているときはその座標系の原点を表すものとする。 X と Y が大文字のときは絶対座標、一方、 x と y が小文字のときは相対座標を示す。また、相対座標系の回転を示す θ_n は $-180 \leq \theta_n < 180$ を満たすものとする。

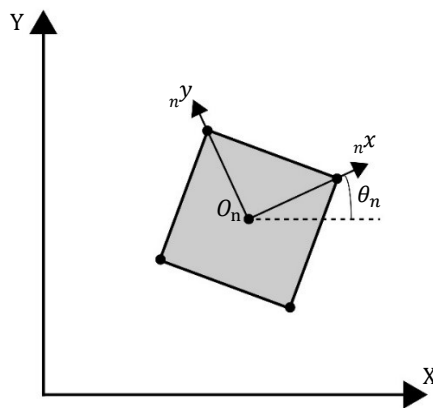


図 2.1.1.1 平面座標系の設定

次に平面座標系の特徴を説明する。正方形の薄片において4つの節点 $V_1({}_n x_1, {}_n y_1)$ 、 $V_2({}_n x_2, {}_n y_2)$ 、 $V_3({}_n x_3, {}_n y_3)$ 、 $V_4({}_n x_4, {}_n y_4)$ をとる (図 2.1.1.2)。

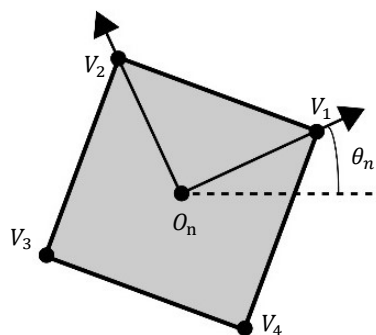


図 2.1.1.2 平面部材の相対座標

まず相対座標系と絶対座標系の関係性を示す。先ほど定義した点 $V_1({}_n x_1, {}_n y_1)$ を絶対座標 $V_1({}_n X_1, {}_n Y_1)$ を以下のように変換できる。

$$V_1 \begin{pmatrix} {}_n X_1 \\ {}_n Y_1 \end{pmatrix} = \begin{pmatrix} {}_n X_0 \\ {}_n Y_0 \end{pmatrix} + \begin{bmatrix} \cos \theta_n & -\sin \theta_n \\ \sin \theta_n & \cos \theta_n \end{bmatrix} \begin{pmatrix} {}_n x_1 \\ {}_n y_1 \end{pmatrix} \quad (2.1)$$

このように相対座標系 n の絶対座標系のなす角 θ_n で変換することができる。

次に部材が回転や移動をしたときの座標の変化を示す。部材が回転することはその部材の相対座標系が回転することに等しいので、 θ_n が $\Delta\theta_n$ 増加することと同等である。このとき、式(2.1)を変形して式(2.2)のように表す。頂点の変形後の座標にはダブルプライムをつける。

$$V''_1 \begin{pmatrix} {}_n X''_1 \\ {}_n Y''_1 \end{pmatrix} = \begin{pmatrix} {}_n X_0 \\ {}_n Y_0 \end{pmatrix} + \begin{bmatrix} \cos(\theta_n + \Delta\theta_n) & -\sin(\theta_n + \Delta\theta_n) \\ \sin(\theta_n + \Delta\theta_n) & \cos(\theta_n + \Delta\theta_n) \end{bmatrix} \begin{pmatrix} {}_n x_1 \\ {}_n y_1 \end{pmatrix} \quad (2.2)$$

部材が移動し、重心の絶対座標が $(\Delta_n X_0, \Delta_n Y_0)$ 変化したとき、部材節点の相対座標の変化はないので先ほどと同様に式(2.1)を変形して式(2.3)のように表せる。

$$V''_1 \begin{pmatrix} {}_n X''_1 \\ {}_n Y''_1 \end{pmatrix} = \begin{pmatrix} {}_n X_0 + \Delta_n X_0 \\ {}_n Y_0 + \Delta_n Y_0 \end{pmatrix} + \begin{bmatrix} \cos \theta_n & -\sin \theta_n \\ \sin \theta_n & \cos \theta_n \end{bmatrix} \begin{pmatrix} {}_n x_1 \\ {}_n y_1 \end{pmatrix} \quad (2.3)$$

以上のことから部材が $\Delta\theta_n$ 回転して、重心が $(\Delta_n X_0, \Delta_n Y_0)$ 移動した後の座標は式(2.2)と式(2.3)を用いて式(2.4)のように求められる。

$$V''_1 \begin{pmatrix} {}_n X''_1 \\ {}_n Y''_1 \end{pmatrix} = \begin{pmatrix} {}_n X_0 + \Delta_n X_0 \\ {}_n Y_0 + \Delta_n Y_0 \end{pmatrix} + \begin{bmatrix} \cos(\theta_n + \Delta\theta_n) & -\sin(\theta_n + \Delta\theta_n) \\ \sin(\theta_n + \Delta\theta_n) & \cos(\theta_n + \Delta\theta_n) \end{bmatrix} \begin{pmatrix} {}_n x_1 \\ {}_n y_1 \end{pmatrix} \quad (2.4)$$

回転行列を分配し、式(2.4)を以下のように変形する。

$$\begin{pmatrix} {}_n X''_1 \\ {}_n Y''_1 \end{pmatrix} = \begin{pmatrix} {}_n X_0 + \Delta_n X_0 \\ {}_n Y_0 + \Delta_n Y_0 \end{pmatrix} + \begin{bmatrix} \cos \Delta\theta_n & -\sin \Delta\theta_n \\ \sin \Delta\theta_n & \cos \Delta\theta_n \end{bmatrix} \begin{bmatrix} \cos \theta_n & -\sin \theta_n \\ \sin \theta_n & \cos \theta_n \end{bmatrix} \begin{pmatrix} {}_n x_1 \\ {}_n y_1 \end{pmatrix}$$

式(2.1)を代入する。

$$\begin{pmatrix} {}_n X''_1 \\ {}_n Y''_1 \end{pmatrix} = \begin{pmatrix} {}_n X_0 + \Delta_n X_0 \\ {}_n Y_0 + \Delta_n Y_0 \end{pmatrix} + \begin{bmatrix} \cos \Delta\theta_n & -\sin \Delta\theta_n \\ \sin \Delta\theta_n & \cos \Delta\theta_n \end{bmatrix} \left\{ \begin{pmatrix} {}_n X_1 \\ {}_n Y_1 \end{pmatrix} - \begin{pmatrix} {}_n X_0 \\ {}_n Y_0 \end{pmatrix} \right\}$$

定数 ${}_n X_0$ 、 ${}_n Y_0$ を左辺に移行する。

$$\begin{pmatrix} {}_n X''_1 - {}_n X_0 \\ {}_n Y''_1 - {}_n Y_0 \end{pmatrix} = \begin{pmatrix} \Delta_n X_0 \\ \Delta_n Y_0 \end{pmatrix} + \begin{bmatrix} \cos \Delta\theta_n & -\sin \Delta\theta_n \\ \sin \Delta\theta_n & \cos \Delta\theta_n \end{bmatrix} \left\{ \begin{pmatrix} {}_n X_1 \\ {}_n Y_1 \end{pmatrix} - \begin{pmatrix} {}_n X_0 \\ {}_n Y_0 \end{pmatrix} \right\}$$

定数と変数を分離するようにさらに変形する。

$$\begin{pmatrix} {}_nX''_1 - {}_nX_0 \\ {}_nY''_1 - {}_nY_0 \end{pmatrix} = \begin{bmatrix} 1 & 0 & {}_nX_1 - {}_nX_0 & -({}_nY_1 - {}_nY_0) \\ 0 & 1 & {}_nY_1 - {}_nY_0 & {}_nX_1 - {}_nX_0 \end{bmatrix} \begin{Bmatrix} \Delta {}_nX_0 \\ \Delta {}_nY_0 \\ \cos \Delta\theta_n \\ \sin \Delta\theta_n \end{Bmatrix} \quad (2.5)$$

式(2.5)のようになる。求め方は既往研究^[4]と異なるが同じ式が得られることが分かった。以後、この定数のみでなる行列(2.6)を変換行列と呼ぶ。

$$\begin{bmatrix} 1 & 0 & {}_nX_1 - {}_nX_0 & -({}_nY_1 - {}_nY_0) \\ 0 & 1 & {}_nY_1 - {}_nY_0 & {}_nX_1 - {}_nX_0 \end{bmatrix} \quad (2.6)$$

同様に部材が $\Delta\theta_n$ 回転して、重心が $(\Delta {}_nX_0, \Delta {}_nY_0)$ 移動した後の頂点 V_2 の座標を求める。

$$\begin{pmatrix} {}_nX''_2 - {}_nX_0 \\ {}_nY''_2 - {}_nY_0 \end{pmatrix} = \begin{bmatrix} 1 & 0 & {}_nX_2 - {}_nX_0 & -({}_nY_2 - {}_nY_0) \\ 0 & 1 & {}_nY_2 - {}_nY_0 & {}_nX_2 - {}_nX_0 \end{bmatrix} \begin{Bmatrix} \Delta {}_nX_0 \\ \Delta {}_nY_0 \\ \cos \Delta\theta_n \\ \sin \Delta\theta_n \end{Bmatrix} \quad (2.7)$$

式(2.7)から式(2.5)を減算した結果を式(2.8)に示す。

$$\begin{pmatrix} {}_nX''_2 - {}_nX''_1 \\ {}_nY''_2 - {}_nY''_1 \end{pmatrix} = \begin{bmatrix} {}_nX_2 - {}_nX_1 & -({}_nY_2 - {}_nY_1) \\ {}_nY_2 - {}_nY_1 & {}_nX_2 - {}_nX_1 \end{bmatrix} \begin{Bmatrix} \cos \Delta\theta_n \\ \sin \Delta\theta_n \end{Bmatrix} \quad (2.8)$$

このように同じ座標系内にある2点の変化後のベクトルは相対座標系の絶対座標系とのなす角度の変化 $\Delta\theta_n$ を用いて表すことができる。式(2.8)の右辺の三角関数で構成された行列にかけられている行列を以下のように定義する。

$$V_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}, V_j = \begin{pmatrix} x_j \\ y_j \end{pmatrix} \quad \text{のとき} \quad \text{Trans}(V_i, V_j) \equiv \begin{bmatrix} x_j - x_i & -(y_j - y_i) \\ y_j - y_i & x_j - x_i \end{bmatrix}$$

また本論では2点 $P_1(x_1, y_1)$ 、 $P_2(x_2, y_2)$ の距離を $L(P_1, P_2) \equiv \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ と表現する。

2.2 空間座標系の設定

2.2.1 一要素の空間座標系

第6章で Fuzzy Node を空間へ拡張するので、そのときに使用する空間座標系を説明する。平面のときと同じように絶対座標系と相対座標系を使用する。図 2.2.2.1 のように絶対座標系の座標軸を X 、 Y 、 Z として部材 n の相対座標系 n の座標軸を ${}_n x$ 、 ${}_n y$ 、 ${}_n z$ とし、相対座標系 n の原点 O_n を部材の重心の絶対座標 $({}_n X_0, {}_n Y_0, {}_n Z_0)$ とする。また、絶対座標系と相対座標系のそれぞれの座標軸が常に並行でかつ向きが同じになるように維持する。これは平面座標系の設定とは異なる。

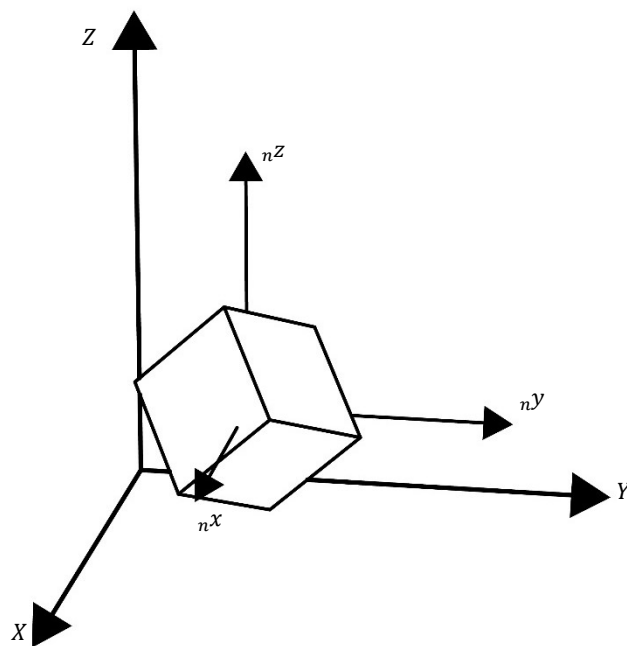


図 2.2.2.1 空間座標系の設定

次に空間座標系の特徴を説明する。立方体型の小片の8つの節点 $V_k ({}_n x_k, {}_n y_k, z_k)$ ただし $(1 \leq k \leq 8)$ をとる (図 2.4)。座標の添え字は平面座標系のときと同じように扱う。

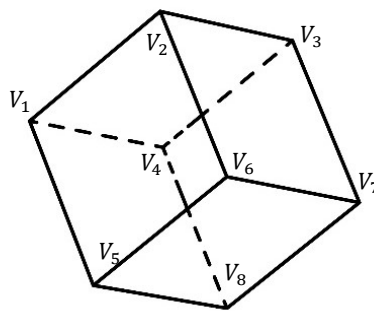


図 2.4 立体部材の相対座標

まず、平面座標系と同じように相対座標系と絶対座標系の関係性を示す。先ほど定義した点 $V_k(n^x_k, n^y_k, n^z_k)$ を絶対座標 $V_k(n^X_k, n^Y_k, n^Z_k)$ に変換する。系同士の座標軸が平行に保たれているので座標を加えるだけで求められる。式(2.9)のように表せる。

$$V_k \begin{pmatrix} n^X_k \\ n^Y_k \\ n^Z_k \end{pmatrix} = \begin{pmatrix} n^X_0 \\ n^Y_0 \\ n^Z_0 \end{pmatrix} + \begin{pmatrix} n^x_k \\ n^y_k \\ n^z_k \end{pmatrix} \quad (2.9)$$

次に部材の回転について考える。空間上の点 (x, y, z) を単位ベクトル $\vec{n} = (n_x, n_y, n_z)$ を軸にして θ 回転したときの座標は空間の回転行列 $R(n, \theta)$ を用いて以下のように求められる。

$$\begin{pmatrix} x'' \\ y'' \\ z'' \end{pmatrix} = R(n, \theta) \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$R(n, \theta) \equiv \begin{bmatrix} \cos \theta + n_x^2(1 - \cos \theta) & n_x n_y(1 - \cos \theta) - n_z \sin \theta & n_x n_z(1 - \cos \theta) + n_y \sin \theta \\ n_y n_x(1 - \cos \theta) + n_z \sin \theta & \cos \theta + n_y^2(1 - \cos \theta) & n_y n_z(1 - \cos \theta) - n_x \sin \theta \\ n_z n_x(1 - \cos \theta) - n_y \sin \theta & n_z n_y(1 - \cos \theta) + n_x \sin \theta & \cos \theta + n_z^2(1 - \cos \theta) \end{bmatrix}$$

以上の式より、点 $V_k(n^x_k, n^y_k, n^z_k)$ を単位ベクトル $\vec{n} = (n_x, n_y, n_z)$ を軸にして θ 回転したときの絶対座標点 $V''_k(n^X''_k, n^Y''_k, n^Z''_k)$ は次のように求められる。

$$V''_k \begin{pmatrix} n^X''_k \\ n^Y''_k \\ n^Z''_k \end{pmatrix} = \begin{pmatrix} n^X_0 \\ n^Y_0 \\ n^Z_0 \end{pmatrix} + R(n, \theta) \begin{pmatrix} n^x_k \\ n^y_k \\ n^z_k \end{pmatrix} \quad (2.10)$$

最後に部材の移動を示す。部材が $(\Delta n^X_0, \Delta n^Y_0, \Delta n^Z_0)$ 移動したとき、部材における相対座標の変化はないので、点 $V_k(n^x_k, n^y_k, n^z_k)$ の移動後の点 $V''_k(n^X''_k, n^Y''_k, n^Z''_k)$ は平面の平行移動の式と同様に式(2.9)を変形して式(2.11)のように表せる。

$$V''_k \begin{pmatrix} n^X''_k \\ n^Y''_k \\ n^Z''_k \end{pmatrix} = \begin{pmatrix} n^X_0 + \Delta n^X_0 \\ n^Y_0 + \Delta n^Y_0 \\ n^Z_0 + \Delta n^Z_0 \end{pmatrix} + \begin{pmatrix} n^x_k \\ n^y_k \\ n^z_k \end{pmatrix} \quad (2.11)$$

式(2.10)と式(2.11)から部材を回転し平行移動させたときの点 $V_k(n^x_k, n^y_k, n^z_k)$ の移動後の点 $V''_k(n^X''_k, n^Y''_k, n^Z''_k)$ は次のように求められる。

$$V_k \begin{pmatrix} n^X_k \\ n^Y_k \\ n^Z_k \end{pmatrix} = \begin{pmatrix} n^X_0 + \Delta n^X_0 \\ n^Y_0 + \Delta n^Y_0 \\ n^Z_0 + \Delta n^Z_0 \end{pmatrix} + R(n, \theta) \begin{pmatrix} n^x_k \\ n^y_k \\ n^z_k \end{pmatrix} \quad (2.12)$$

第3章 Fuzzy Node で接合された2部材の解法

3.1 解を一度の計算で求める方法

部材の位置を一度の計算で求める解法を提案する。

3.1.1 Fuzzy Node で接合された2部材の行列表現

Fuzzy Node を含む変換行列の解法について考察する。河村らの既往研究^[4]で求められている変換行列をもとに考察するので、はじめに、それを紹介する。

図 3.1.1.1 のように2つの正方形型の小片が Fuzzy Node で接合される二次元モデルについて考察する。4点 $V_1(x_1, y_1)$ 、 $V_2(x_2, y_2)$ 、 $V_3(x_3, y_3)$ 、 $V_4(x_4, y_4)$ をとり、部材1、FN (Fuzzy Node) 2、部材3それぞれの重心の座標を $O_1(ox_1, oy_1)$ 、 $O_2(ox_2, oy_2)$ 、 $O_3(ox_3, oy_3)$ をおく。

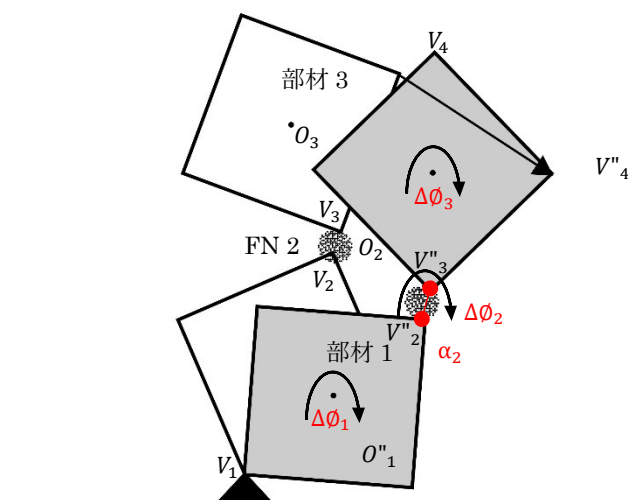


図 3.1.1.1 Fuzzy Node で接合された2部材に変位を加えている様子

節点 $V_4(x_4, y_4)$ に強制変位を与え $V''_4(x''_4, y''_4)$ に移動させたときの変換行列は式(3.1)のようになる。右辺にある未知数 α_2 は Fuzzy Node の長さを示すものであり、この式では FN 2 の長さを表している。また $\Delta\theta_1, \Delta\theta_2, \Delta\theta_3$ はそれぞれの回転の変化を表している。

$$\begin{Bmatrix} x''_2 - x''_1 \\ y''_2 - y''_1 \\ x''_3 - x''_2 \\ y''_3 - y''_2 \\ x''_4 - x''_3 \\ y''_4 - y''_3 \end{Bmatrix} = \begin{bmatrix} Trans(V_1, V_2) & 0 & 0 \\ 0 & Trans(V_2, V_3) & 0 \\ 0 & 0 & Trans(V_3, V_4) \end{bmatrix} \begin{Bmatrix} \cos \Delta\theta_1 \\ \sin \Delta\theta_1 \\ \alpha_2 \times \cos \Delta\theta_2 \\ \alpha_2 \times \sin \Delta\theta_2 \\ \cos \Delta\theta_3 \\ \sin \Delta\theta_3 \end{Bmatrix} \quad (3.1)$$

この行列を縮約すると次の式(3.2)になることが知られている^[4]。

$$\begin{cases} x''_4 - x''_1 \\ y''_4 - y''_1 \end{cases} = [Trans(V_1, V_2) \quad Trans(V_2, V_3) \quad Trans(V_3, V_4)] \begin{cases} \cos \Delta\phi_1 \\ \sin \Delta\phi_1 \\ \alpha_2 \times \cos \Delta\phi_2 \\ \alpha_2 \times \sin \Delta\phi_2 \\ \cos \Delta\phi_3 \\ \sin \Delta\phi_3 \end{cases} \quad (3.2)$$

この行列の解法を複数提案し、検討する。

3.1.2 No Fuzzy 仮定法

この解法は Fuzzy Node があるにもかかわらず、Fuzzy Node がないものと想定して解く方法である。

図 3.1.2.1 のように Fuzzy Node で接合された 2 つの正方形型の部材の挙動は式(3.2)と同様に表現することができる。この二次元モデルを図 3.1.2.2 のように Fuzzy Node がないものとして連立方程式を解くことを試みる。

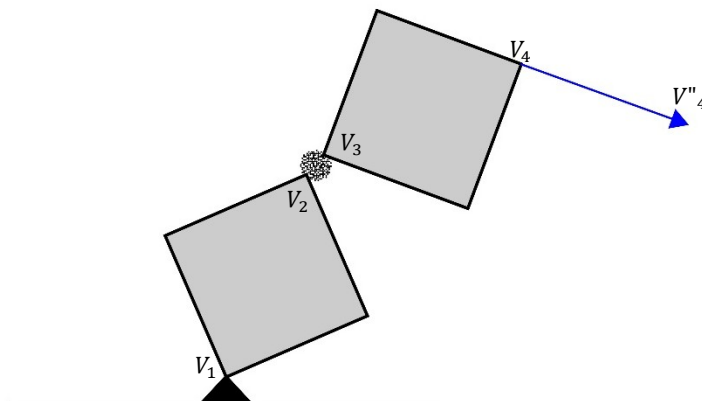


図 3.1.2.1 Fuzzy Node で接合された 2 部材に変位を加える様子

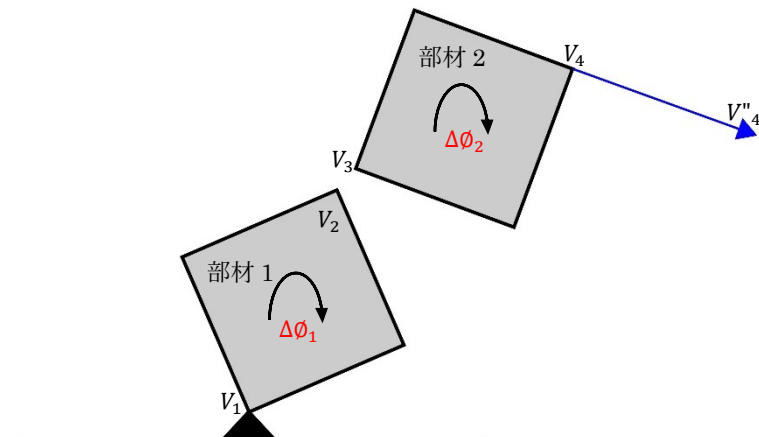


図 3.2.2.2 図 3.1.2.1 から Fuzzy Node をなくしたモデルに変位を加える様子

図 3.3 のように 4 点 $V_1(x_1, y_1)$ 、 $V_2(x_2, y_2)$ 、 $V_3(x_3, y_3)$ 、 $V_4(x_4, y_4)$ とそれぞれの部材の回転を $\Delta\theta_1$ 、 $\Delta\theta_2$ とすると、節点 $V_4(x_4, y_4)$ に強制変位を与え $V''_4(x''_4, y''_4)$ に移動させたときの変換行列を縮約した式は式(3.3)のようになる。

$$\begin{Bmatrix} x''_2 - x''_1 \\ y''_2 - y''_1 \\ x''_4 - x''_3 \\ y''_4 - y''_3 \end{Bmatrix} = \begin{bmatrix} Trans(V_1, V_2) & 0 \\ 0 & Trans(V_3, V_4) \end{bmatrix} \begin{Bmatrix} \cos \Delta\theta_1 \\ \sin \Delta\theta_1 \\ \cos \Delta\theta_2 \\ \sin \Delta\theta_2 \end{Bmatrix} \quad (3.3)$$

この連立方程式の解を求める。未知数と条件式の数の差に注目する。左辺にある未知数は x''_2 、 y''_2 、 x''_3 、 y''_3 の 4 つ、右辺には $\Delta\theta_1$ 、 $\Delta\theta_2$ の 2 つの合計 6 個存在する。一方、4 つの式しかないのでこの連立方程式の自由度は 2 となり、解が定まらない。また、式を見ると上の 2 式と下の 2 式は互いに独立していることがわかる。そのため、式(3.3)の解は連立方程式(3.4)と同値になる。

$$\begin{Bmatrix} x''_2 - x''_1 \\ y''_2 - y''_1 \end{Bmatrix} = [Trans(V_1, V_2)] \begin{Bmatrix} \cos \Delta\theta_1 \\ \sin \Delta\theta_1 \end{Bmatrix} \quad (3.4.1)$$

$$\begin{Bmatrix} x''_4 - x''_3 \\ y''_4 - y''_3 \end{Bmatrix} = [Trans(V_3, V_4)] \begin{Bmatrix} \cos \Delta\theta_2 \\ \sin \Delta\theta_2 \end{Bmatrix} \quad (3.4.2)$$

式(3.4.1)を以下のように変形できる。

$$\begin{Bmatrix} x''_2 - x''_1 \\ y''_2 - y''_1 \end{Bmatrix} = [Trans(V_1, V_2)] \begin{Bmatrix} \cos \Delta\theta_1 \\ \sin \Delta\theta_1 \end{Bmatrix}$$

$$\begin{Bmatrix} x''_2 - x''_1 \\ y''_2 - y''_1 \end{Bmatrix} = \begin{Bmatrix} (x_2 - x_1) \cos \Delta\theta_1 - (y_2 - y_1) \sin \Delta\theta_1 \\ (y_2 - y_1) \cos \Delta\theta_1 + (x_2 - x_1) \sin \Delta\theta_1 \end{Bmatrix}$$

ここで三角関数の合成を用いて変形する。

$$\begin{Bmatrix} x''_2 - x''_1 \\ y''_2 - y''_1 \end{Bmatrix} = \begin{Bmatrix} L(V_1, V_2) \cos \alpha \cos \Delta\theta_1 - L(V_1, V_2) \sin \alpha \sin \Delta\theta_1 \\ L(V_1, V_2) \sin \alpha \cos \Delta\theta_1 + L(V_1, V_2) \cos \alpha \sin \Delta\theta_1 \end{Bmatrix}$$

$$\begin{Bmatrix} x''_2 - x''_1 \\ y''_2 - y''_1 \end{Bmatrix} = \begin{Bmatrix} L(V_1, V_2) \cos(\Delta\theta_1 + \alpha) \\ L(V_1, V_2) \sin(\Delta\theta_1 + \alpha) \end{Bmatrix} \text{ただし、} \cos \alpha = \frac{x_2 - x_1}{L(V_1, V_2)}, \sin \alpha = \frac{y_2 - y_1}{L(V_1, V_2)}$$

3 つの未知数 x''_2 、 y''_2 、 $\Delta\theta_1$ の関係を示すことができた。式(3.4.2)でも同様に変形する。

$$\begin{Bmatrix} x''_4 - x''_3 \\ y''_4 - y''_3 \end{Bmatrix} = \begin{Bmatrix} L(V_3, V_4) \cos(\Delta\theta_2 + \beta) \\ L(V_3, V_4) \sin(\Delta\theta_2 + \beta) \end{Bmatrix} \text{ただし、} \cos \beta = \frac{x_4 - x_3}{L(V_3, V_4)}, \sin \beta = \frac{y_4 - y_3}{L(V_3, V_4)}$$

この連立方程式でも同様に 3 つの未知数 x''_3 、 y''_3 、 $\Delta\theta_2$ の関係を示すことができる。

これよりすべての変数の単純化した関係性は求められたが、条件が足りないために値までは求めることができないことが示された。今後の課題として、どのような条件を与えることで、連立方程式(3.4)が解けるようになるのか、また2つ以上の複数の部材への対応の可否を検討する必要がある。本論ではこれ以上、No Fuzzy 仮定法について検証しない。

3.1.3 部材変形法

この解法は Fuzzy Node が伸縮・回転するのでなく、部材の形が少し変形して Fuzzy Node の持つ役割を代わりに果たすという方法である。

No Fuzzy 仮定法と同様に図 3.1.2.1 のモデルと比較しつつ、変化した縮約行列の解法を考察する。図 3.1.3.1 のように Fuzzy Node で接合されている正方形型の部材の節点とその部材の重心との距離が2つの係数 α 、 β を用いて表現されるものとする。部材1、部材2それぞれの重心を $O_1(ox_1, oy_1)$ 、 $O_2(ox_2, oy_2)$ 、また4つの節点を $V_1(x_1, y_1)$ 、 $V_2(x_2, y_2)$ 、 $V_3(x_3, y_3)$ 、 $V_4(x_4, y_4)$ とおく。 V_2 、 V_3 は変形前の座標、 $V_5(x_5, y_5)$ は節点の座標を表す。

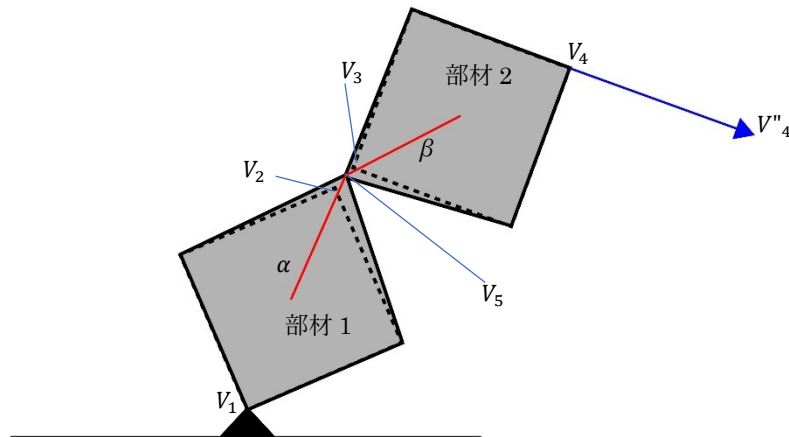


図 3.1.3.1 部材が少し変形している様子

節点 $V_4(x_4, y_4)$ に強制変位を与え $V_4''(x_4'', y_4'')$ に移動させたとき、全体の変換行列は次のように表せる。 α と β は $\alpha \times L(O_1, V_2) = L(O_1, V_5)$ と $\beta \times L(O_2, V_3) = L(O_2, V_5)$ を満たす。

$$\begin{pmatrix} x_1'' - ox_1 \\ y_1'' - oy_1 \\ x_5'' - ox_1 \\ y_5'' - oy_1 \\ x_5'' - ox_2 \\ y_5'' - oy_2 \\ x_4'' - ox_2 \\ y_4'' - oy_2 \end{pmatrix} = \begin{bmatrix} 1 & 0 & x_1 - ox_1 & -(y_1 - oy_1) & 0 & 0 & 0 & 0 \\ 0 & 1 & y_1 - oy_1 & x_1 - ox_1 & 0 & 0 & 0 & 0 \\ 1 & 0 & \alpha(x_2 - ox_1) & -\alpha(y_2 - oy_1) & 0 & 0 & 0 & 0 \\ 0 & 1 & \alpha(y_2 - oy_1) & \alpha(x_2 - ox_1) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & \beta(x_3 - ox_2) & -\beta(y_3 - oy_2) \\ 0 & 0 & 0 & 0 & 0 & 1 & \beta(y_3 - oy_2) & \beta(x_3 - ox_2) \\ 0 & 0 & 0 & 0 & 1 & 0 & x_4 - ox_2 & -(y_4 - oy_2) \\ 0 & 0 & 0 & 0 & 0 & 1 & y_4 - oy_2 & x_4 - ox_2 \end{bmatrix} \begin{pmatrix} \Delta ox_1 \\ \Delta oy_1 \\ \cos\Delta\theta_1 \\ \sin\Delta\theta_1 \\ \Delta ox_2 \\ \Delta oy_2 \\ \cos\Delta\theta_2 \\ \sin\Delta\theta_2 \end{pmatrix}$$

この行列を縮約すると次のようになる。

$$\begin{cases} x''_1 - x''_4 \\ y''_1 - y''_4 \end{cases} = \begin{bmatrix} Ax & -Ay & Bx & -By \\ Ay & Ax & By & Bx \end{bmatrix} \begin{cases} \cos\Delta\theta_1 \\ \sin\Delta\theta_1 \\ \cos\Delta\theta_2 \\ \sin\Delta\theta_2 \end{cases}$$

ただし、 $Ax = x_1 - ox_1 - \alpha(x_2 - ox_1), Ay = y_1 - oy_1 - \alpha(y_2 - oy_1)$
 $Bx = x_2 - ox_2 - \alpha(x_4 - ox_2), By = y_2 - oy_2 - \alpha(y_4 - oy_2)$

この形式は部材2つを Fuzzy Node なしで接合したときに得られる行列と同じなので先行研究^[4]から三角関数の合成を使用すると計算しやすい形に変換することができる。一方で、No Fuzzy 仮定法と同様、条件が足りないがためにすべての変数の値までは求めることができない。この解法においても、これ以上の考察を本論では行わない。

3.1.4 Fuzzy Node 伸縮制限法

No Fuzzy 仮定法と部材変形法ではどちらも自由度が高すぎたがために変数同士の関係式を示すことしかできなかつた。そこで、この項では今までの解法よりも厳しい条件を加えることでそれぞれの変数の値を求めることができるのか、また、その条件がどれほどの制約を持つものである必要があるのかを検証する。

この解法は名前の通り、Fuzzy Node の伸縮度合いに制限を設ける方法である。複数の制限の設け方を試す。今までは、正方形型の小片で考察していたがここでは点 D の軌跡を求める必要があるだけであり全体の挙動については考察しないものとし、図 3.1.2.1 のモデルを図 3.1.4.1 の線モデルで置き換えて考察する。

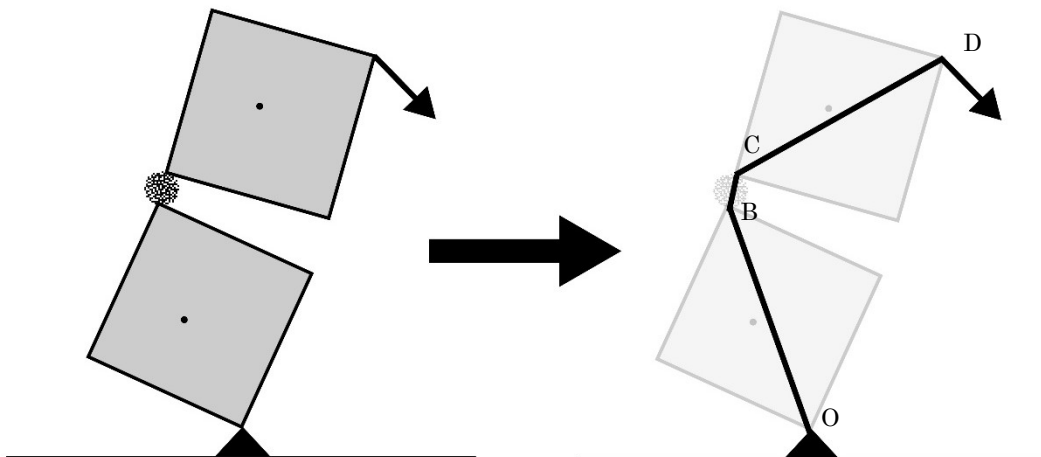


図 3.1.4.1 Fuzzy Node で接合された部材の線モデル化

図 3.1.4.1 の 4 つの節点を $O(0,0)$ 、 $B(b_x, b_y)$ 、 $C(c_x, c_y)$ 、 $D(d_x, d_y)$ とおく。このとき、線分 BC が Fuzzy Node を表している。線分 BC に制限を設けることによる D の軌跡の変化を考察する。このとき、節点 O は固定する。

本論では Fuzzy Node の最大長が部材一辺の 1/10 程度としているので、この条件を考慮したうえで以下の 3 種類の条件の下での D の軌跡を図示する。

$$OB = 10, CD = 10, BC = 1 \quad (3.5)$$

$$OB = 10, CD = 10, BC \leq 1 \quad (3.6)$$

$$OB = 10, CD = 10, BC \leq 0.5 \quad (3.7)$$

Fuzzy Node の長さが変化しないのが条件式(3.5)、Fuzzy Node の伸縮に上限を設けたのが条件式(3.6)と(3.7)である。

はじめに、どの条件においても共通である点 B の挙動について整理する。 $OB = 10$ より B は O を中心とする半径 10 の円周上に存在する。そのため、点 B を固定したときの点 D の軌跡を求めた後に、その求めた軌跡を点 O を中心に 1 回転させれば点 D の全体の軌跡が求められるため、いったん $B(b_x, b_y) = (10, 0)$ と定める。

条件式(3.5)のときの D の軌跡を図示する。

点 C は点 B を中心とする半径 1 の円周上なので $B = (10, 0)$ より

$$C(c_x, c_y) = (10 + \cos \theta, \sin \theta) \quad \text{ただし } 0 \leq \theta \leq 2\pi \quad \text{と表すことができる。}$$

さらに点 D は点 C を中心とする半径 10 の円周上にあるので

$$D(d_x, d_y) = (10 + \cos \theta + 10 \cos \phi, \sin \theta + 10 \sin \phi) \quad \text{ただし } 0 \leq \phi \leq 2\pi \quad \text{と表すことができる。}$$

ここで、ある点 C における点 D の軌跡について考える。図 3.1.4.2 のように表すことができる。小さい円は C の軌跡、大きい円が D の軌跡を表している。

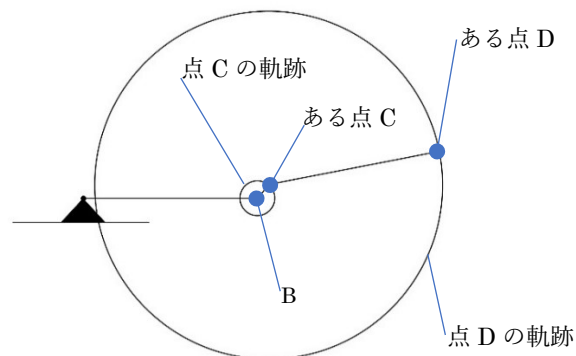


図 3.1.4.2 ある点 C における点 D の軌跡

$\overrightarrow{BD} = (d_x - b_x, d_y - b_y) = (10 + \cos \theta + 10 \cos \phi - 10, \sin \theta + 10 \sin \phi)$ となるので

$\overrightarrow{BD} = (\cos \theta + 10 \cos \phi, \sin \theta + 10 \sin \phi)$ と変形できる。これより

$|\overrightarrow{BD}|^2 = (\cos \theta + 10 \cos \phi)^2 + (\sin \theta + 10 \sin \phi)^2$ となる。さらに変形する。

$|\overrightarrow{BD}|^2 = 101 + 20 \cos \phi \cos \theta + 20 \sin \phi \sin \theta$ ここで加法定理の逆より

$|\overrightarrow{BD}|^2 = 101 + 20 \cos(\phi - \theta)$ と簡潔に表すことができ、ベクトル BD の大きさは次のように求まる。

$81 \leq |\overrightarrow{BD}|^2 \leq 121$ 両辺の平方根をとる。

$$\therefore 9 \leq |\overrightarrow{BD}| \leq 11$$

これよりベクトル BD の取りうる大きさの範囲が求められた。

$0 \leq \theta \leq 2\pi$ なので

点 B を固定したときの点 D の軌跡は図 3.1.4.2 のある点 C における点 D の軌跡を点 C の軌跡に沿って 1 回転させたものである。

点 D の軌跡は $9 \leq |\overrightarrow{BD}| \leq 11$ より点 B を中心とした半径 11 の円の円周上を含む内部かつ点 B を中心とした半径 9 の円の円周上を含む外部と等しい。

図示すると以下の図 3.1.4.3 のようになる。

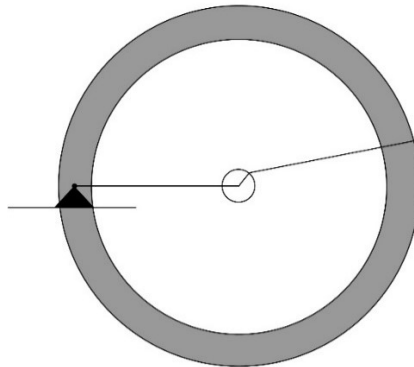


図 3.1.4.3 条件(3.5)のもとある点 B における点 D の軌跡

このとき、 $\overrightarrow{BD} = (r \cos \phi, r \sin \phi)$ $9 \leq r \leq 11, 0 \leq \phi \leq 2\pi$ と表すことができる。

$$\therefore \overrightarrow{OD} = (10 + r \cos \phi, r \sin \phi)$$

$$|\overrightarrow{OD}|^2 = (10 + r \cos \phi)^2 + (r \sin \phi)^2$$

$$|\overrightarrow{OD}|^2 = 100 + 20r \cos \phi + r^2$$

$$0 \leq \varphi \leq 2\pi \text{ より } 100 - 20r + r^2 \leq |\overrightarrow{OD}|^2 \leq 100 + 20r + r^2 \text{ となる。}$$

$$9 \leq r \leq 11 \text{ より}$$

$$0 \leq (r - 10)^2 \leq |\overrightarrow{OD}|^2 \leq (r + 10)^2 \leq 21^2 \text{ と変形できる。}$$

点 B は固定しているが実際は点 O を中心とする円周上に存在する、そのため点 D の軌跡は図 3.1.4.3 のとき求めた軌跡を点 O 中心に回転したものとなる。

$$0 \leq |\overrightarrow{OD}| \leq 21 \text{ となるので}$$

D の軌跡は点 O を中心とする半径 21 の円の内部および円周上になる。

次に条件式(3.6)のときの D の軌跡を図示する。

条件式(3.5)のときと同様に C を求める。

$$BC \leq 1 \text{ より}$$

$$C(c_x, c_y) = (10 + f * \cos \theta, f * \sin \theta) \text{ ただし } 0 \leq \theta \leq 2\pi, 0 \leq f \leq 1 \text{ となる}$$

条件式(3.5)のときと同様な計算を行う。

$$|\overrightarrow{BD}|^2 = 100 + f * 20 \cos(\varphi - \theta) + f^2 \text{ となる。}$$

$$100 - f * 20 + f^2 \leq |\overrightarrow{BD}|^2 \leq 100 + f * 20 + f^2 \text{ と変形できる。}$$

$$0 \leq f \leq 1 \text{ より}$$

$$9^2 \leq (f - 10)^2 \leq |\overrightarrow{BD}|^2 \leq (f + 10)^2 \leq 11^2 \text{ となる。}$$

これは条件式(3.5)と同じ結果である。

そのため、この条件における点 D の軌跡は条件式(3.5)のときと同じになる。

最後に条件式(3.7)のときの点 D の軌跡を図示する。

条件が不等式なので条件式(3.6)のときと同様に考える

ただしこのとき、 $BC \leq 0.5$ なので

$$0 \leq f \leq 0.5 \text{ となり、}$$

$$9.5^2 \leq (f - 10)^2 \leq |\overrightarrow{BD}|^2 \leq (f + 10)^2 \leq 10.5^2 \text{ となる。}$$

点 D の軌跡は $9.5 \leq |\overrightarrow{BD}| \leq 10.5$ より点 B を中心とした半径 10.5 の円の円周上を含む内部かつ点 B を中心とした半径 5.9 の円の円周上を含む外部と等しい。

図示すると以下の図 3.1.4.4 のようになる。

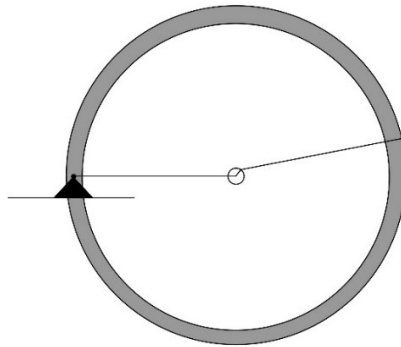


図 3.1.4.4 条件(3.7)のもとある点 B における点 D の軌跡

以後の計算は条件(3.5)のときと同様に行う。

$$|\overline{OD}|^2 = 100 + 20r \cos \phi + r^2 \quad \text{となる。}$$

$$0 \leq \phi \leq 2\pi \quad \text{より} \quad 100 - 20r + r^2 \leq |\overline{OD}|^2 \leq 100 + 20r + r^2 \quad \text{となる。}$$

$$9.5 \leq r \leq 10.5 \quad \text{より}$$

$$0 \leq (r - 10)^2 \leq |\overline{OD}|^2 \leq (r + 10)^2 \leq 20.5^2 \quad \text{と範囲が求まる。}$$

これを点 O 中心に回転させたものが D の軌跡になるので

D の軌跡は点 O を中心とする半径 20.5 の円の内部になる。

以上よりそれぞれの条件における D の軌跡を求めることができた。結果を以下の表 3.1.4.5 にまとめる。

	条件 : $BC = 1$	条件 : $BC \leq 1$	条件 : $BC \leq 0.5$
点 B 固定	$9 \leq \overline{BD} \leq 11$	$9 \leq \overline{BD} \leq 11$	$9.5 \leq \overline{BD} \leq 10.5$
点 B 自由	$0 \leq \overline{OD} \leq 21$	$0 \leq \overline{OD} \leq 21$	$0 \leq \overline{OD} \leq 20.5$

表 3.1.4.5 条件別の結果まとめ

条件 1 と条件 2 から Fuzzy Node の最大長が同じであれば Fuzzy Node の長さを可変にしても点 D の取りうる軌跡は変わらないことがわかった。また、条件 2 と条件 3 から Fuzzy Node の最大長を短くすると D の軌跡も全体的に小さくなることがわかった。

しかし、この結果は部材 2 つを Fuzzy Node で接合したときの軌跡を示しているだけで、部材が増えたときでも同様な傾向がみられる保証はない。それを求めることは Fuzzy Node の研究の今後の課題の 1 つである。

3.1.5 回転角制御法

No Fuzzy 仮定法と部材変形法では変数が変換行列内で登場する位置を調節し、解を求めやすくし、解を求めることを試みましたが、条件が足りず解を求めるまでには至りませんでした。Fuzzy Node 伸縮制限法では Fuzzy Node の長さに条件を与えるだけでは全体形の挙動を制御できないことがわかった。そこで次の解法では Fuzzy Node の回転具合に条件を設けて縮約行列を解くことを試みる。この解法を回転角制御法と呼ぶ。

ここでは線材の小片を Fuzzy Node で接合したモデル (図 3.1.5.1) で考察する。2つの線材の両端を節点とし、4点 $V_1(x_1, y_1)$ 、 $V_2(x_2, y_2)$ 、 $V_3(x_3, y_3)$ 、 $V_4(x_4, y_4)$ をとる。それぞれを線材 1、FN (Fuzzy Node) 2、線材 3 と呼ぶ。

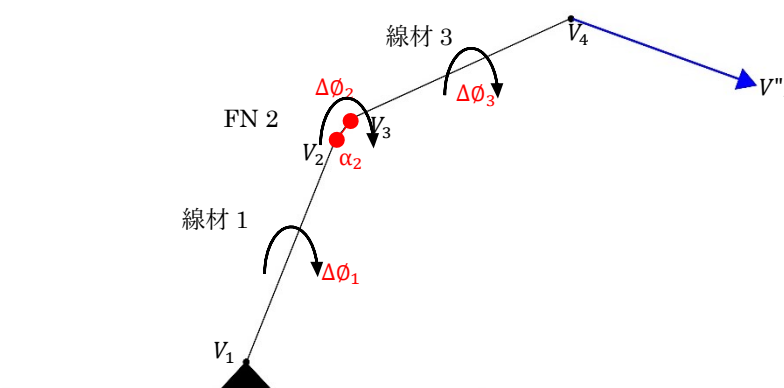


図 3.1.5.1 Fuzzy Node で接合された線モデル

節点 $V_4(x_4, y_4)$ に強制変位を与え $V''_4(x''_4, y''_4)$ に移動させたときの変換行列を縮約したものは式(3.2)から次のように求められる。右辺にある未知数 α_2 は Fuzzy Node の長さを示すものであり、この式では FN 2 の長さを表している。また $\Delta\phi_1, \Delta\phi_2, \Delta\phi_3$ はそれぞれの回転の変化を表している。

$$\begin{Bmatrix} x''_4 - x''_1 \\ y''_4 - y''_1 \end{Bmatrix} = [Trans(V_1, V_2) \quad Trans(V_2, V_3) \quad Trans(V_3, V_4)] \begin{Bmatrix} \cos \Delta\phi_1 \\ \sin \Delta\phi_1 \\ \alpha_2 \times \cos \Delta\phi_2 \\ \alpha_2 \times \sin \Delta\phi_2 \\ \cos \Delta\phi_3 \\ \sin \Delta\phi_3 \end{Bmatrix}$$

次のような変形を考える。

$$[Trans(V_1, V_2)] \begin{Bmatrix} \cos \Delta\phi_1 \\ \sin \Delta\phi_1 \end{Bmatrix} = \begin{Bmatrix} (x_2 - x_1) \cos \Delta\phi_1 - (y_2 - y_1) \sin \Delta\phi_1 \\ (y_2 - y_1) \cos \Delta\phi_1 + (x_2 - x_1) \sin \Delta\phi_1 \end{Bmatrix}$$

三角関数の合成を用いて変形する。

$$\begin{cases} x''_2 - x''_1 \\ y''_2 - y''_1 \end{cases} = \begin{cases} L(V_1, V_2) \cos \alpha \cos \Delta\theta_1 - L(V_1, V_2) \sin \alpha \sin \Delta\theta_1 \\ L(V_1, V_2) \sin \alpha \cos \Delta\theta_1 + L(V_1, V_2) \cos \alpha \sin \Delta\theta_1 \end{cases}$$

$$\begin{cases} x''_2 - x''_1 \\ y''_2 - y''_1 \end{cases} = \begin{cases} L(V_1, V_2) \cos(\Delta\theta_1 + \alpha) \\ L(V_1, V_2) \sin(\Delta\theta_1 + \alpha) \end{cases} \text{ただし、} \cos \alpha = \frac{x_2 - x_1}{L(V_1, V_2)}, \sin \alpha = \frac{y_2 - y_1}{L(V_1, V_2)}$$

$$\cos \alpha = \frac{x_2 - x_1}{L(V_1, V_2)}, \sin \alpha = \frac{y_2 - y_1}{L(V_1, V_2)} \text{に着目すると}$$

$\alpha = \theta_1$ であることがわかる。

これより、この式をさらに変形して

$$[Trans(V_1, V_2)] \begin{cases} \cos \Delta\theta_1 \\ \sin \Delta\theta_1 \end{cases} = \begin{cases} L(V_1, V_2) \cos(\theta_1 + \Delta\theta_1) \\ L(V_1, V_2) \sin(\theta_1 + \Delta\theta_1) \end{cases}$$

$$[Trans(V_1, V_2)] \begin{cases} \cos \Delta\theta_1 \\ \sin \Delta\theta_1 \end{cases} = \begin{bmatrix} L(V_1, V_2) & 0 \\ 0 & L(V_1, V_2) \end{bmatrix} \begin{cases} \cos(\theta_1 + \Delta\theta_1) \\ \sin(\theta_1 + \Delta\theta_1) \end{cases}$$

また、 $\theta_1 + \Delta\theta_1 = \theta''_1$ なので、次のようになる。

$$[Trans(V_1, V_2)] \begin{cases} \cos \Delta\theta_1 \\ \sin \Delta\theta_1 \end{cases} = \begin{bmatrix} L(V_1, V_2) & 0 \\ 0 & L(V_1, V_2) \end{bmatrix} \begin{cases} \cos \theta''_1 \\ \sin \theta''_1 \end{cases}$$

この変形を使用して変換行列を縮約したものを以下のように書き換える。

$$\begin{cases} x''_4 - x''_1 \\ y''_4 - y''_1 \end{cases} = \begin{bmatrix} L(V_1, V_2) & 0 & L(V_2, V_3) & 0 & L(V_3, V_4) & 0 \\ 0 & L(V_1, V_2) & 0 & L(V_2, V_3) & 0 & L(V_3, V_4) \end{bmatrix} \begin{cases} \cos \theta''_1 \\ \sin \theta''_1 \\ \alpha_2 \times \cos \theta''_2 \\ \alpha_2 \times \sin \theta''_2 \\ \cos \theta''_3 \\ \sin \theta''_3 \end{cases}$$

線材 1 と線材 2 は同一形状の部材であるため

$$L(V_1, V_2) = L(V_3, V_4) = 1 \text{ とできる。1 としたのは基準化のためである。}$$

これより変換行列は次のように表せる。

$$\begin{cases} x''_4 - x''_1 \\ y''_4 - y''_1 \end{cases} = \begin{bmatrix} 1 & 0 & L(V_2, V_3) & 0 & 1 & 0 \\ 0 & 1 & 0 & L(V_2, V_3) & 0 & 1 \end{bmatrix} \begin{cases} \cos \theta''_1 \\ \sin \theta''_1 \\ \alpha_2 \times \cos \theta''_2 \\ \alpha_2 \times \sin \theta''_2 \\ \cos \theta''_3 \\ \sin \theta''_3 \end{cases}$$

これから先はこの行列を変形して解を求める。

上記の行列を計算する。

$$\begin{cases} x''_4 - x''_1 = \cos \varphi''_1 + \alpha_2 L(V_2, V_3) \cos \varphi''_2 + \cos \varphi''_3 \\ y''_4 - y''_1 = \sin \varphi''_1 + \alpha_2 L(V_2, V_3) \sin \varphi''_2 + \sin \varphi''_3 \end{cases}$$

ここで、 φ''_1 と φ''_3 において、三角関数の和積の公式より変形する。

$$\begin{cases} x''_4 - x''_1 = 2\cos\left(\frac{\varphi''_1 + \varphi''_3}{2}\right)\cos\left(\frac{\varphi''_1 - \varphi''_3}{2}\right) + \alpha_2 L(V_2, V_3) \cos \varphi''_2 \\ y''_4 - y''_1 = 2\sin\left(\frac{\varphi''_1 + \varphi''_3}{2}\right)\cos\left(\frac{\varphi''_1 - \varphi''_3}{2}\right) + \alpha_2 L(V_2, V_3) \sin \varphi''_2 \end{cases}$$

連立方程式から $\cos\left(\frac{\varphi''_1 - \varphi''_3}{2}\right)$ を消去する。

$$\begin{aligned} ((x''_4 - x''_1) - \alpha_2 L(V_2, V_3) \cos \varphi''_2) \sin\left(\frac{\varphi''_1 + \varphi''_3}{2}\right) \\ = ((y''_4 - y''_1) - \alpha_2 L(V_2, V_3) \sin \varphi''_2) \cos\left(\frac{\varphi''_1 + \varphi''_3}{2}\right) \end{aligned}$$

ここで、

$$\cos \varphi''_2 \sin\left(\frac{\varphi''_1 + \varphi''_3}{2}\right) - \sin \varphi''_2 \cos\left(\frac{\varphi''_1 + \varphi''_3}{2}\right) = \sin\left(\frac{\varphi''_1 + \varphi''_3}{2} - \varphi''_2\right) \text{ より}$$

$$\begin{aligned} (x''_4 - x''_1) \sin\left(\frac{\varphi''_1 + \varphi''_3}{2}\right) - (y''_4 - y''_1) \cos\left(\frac{\varphi''_1 + \varphi''_3}{2}\right) \\ = \alpha_2 L(V''_1, V''_4) \sin\left(\frac{\varphi''_1 + \varphi''_3}{2} - C\right) \end{aligned}$$

$$\text{ただし } \sin C = \frac{x''_4 - x''_1}{L(V''_1, V''_4)}, \cos C = \frac{y''_4 - y''_1}{L(V''_1, V''_4)}$$

これより、計算中の連立方程式を以下のように変形できる。

$$\sin\left(\frac{\varphi''_1 + \varphi''_3}{2} - \varphi''_2\right) = \alpha_2 L(V''_1, V''_4) \sin\left(\frac{\varphi''_1 + \varphi''_3}{2} - C\right)$$

$$\text{ただし } \sin C = \frac{x''_4 - x''_1}{L(V''_1, V''_4)}, \cos C = \frac{y''_4 - y''_1}{L(V''_1, V''_4)}$$

このとき、 $\frac{\Delta\varphi_1 + \Delta\varphi_3}{2} = \Delta\varphi_2$ と仮定すると

$$\varphi_1 + \Delta\varphi_1 = \varphi''_1, \varphi_2 + \Delta\varphi_2 = \varphi''_2, \varphi_3 + \Delta\varphi_3 = \varphi''_3 \text{ なので}$$

$$\frac{\phi''_1}{2} + \frac{\phi''_3}{2} - \phi''_2 = \frac{\phi_1 + \Delta\phi_1}{2} + \frac{\phi_3 + \Delta\phi_3}{2} - \phi_2 + \Delta\phi_2$$

$$\frac{\phi''_1}{2} + \frac{\phi''_3}{2} - \phi''_2 = \frac{\phi_1 + \Delta\phi_1}{2} + \frac{\phi_3 + \Delta\phi_3}{2} - \phi_2 - \frac{\Delta\phi_1 + \Delta\phi_3}{2}$$

$$\frac{\phi''_1}{2} + \frac{\phi''_3}{2} - \phi''_2 = \frac{\phi_3 + \Delta\phi_3}{2} - \phi_2 \text{と変形でき、左辺を定数のみで表せる。}$$

これより、さらに変形する。

$$\frac{\phi''_1}{2} + \frac{\phi''_3}{2} - C = \text{Arcsin} \left(\frac{\sin \left(\frac{\phi''_1}{2} + \frac{\phi''_3}{2} - \phi''_2 \right)}{\alpha_2 L(V''_1, V''_4)} \right)$$

$$\frac{\phi_1 + \Delta\phi_1}{2} + \frac{\phi_3 + \Delta\phi_3}{2} = \text{Arcsin} \left(\frac{\sin \left(\frac{\phi''_1}{2} + \frac{\phi''_3}{2} - \phi''_2 \right)}{\alpha_2 L(V''_1, V''_4)} \right) + C$$

$$\Delta\phi_2 = \text{Arcsin} \left(\frac{\sin \left(\frac{\phi''_1}{2} + \frac{\phi''_3}{2} - \phi''_2 \right)}{\alpha_2 L(V''_1, V''_4)} \right) + C - \frac{\phi_1 + \phi_3}{2}$$

Fuzzy Node 伸縮制限法から Fuzzy Node の伸縮度合いが全体形に与える影響が少ないことが分かったので

$\alpha_2 = 1$ とおくと、つぎのようになる。

$$\Delta\phi_2 = \text{Arcsin} \left(\frac{\sin \left(\frac{\phi''_1}{2} + \frac{\phi''_3}{2} - \phi''_2 \right)}{L(V''_1, V''_4)} \right) + C - \frac{\phi_1 + \phi_3}{2}$$

$$\text{ただし } \sin C = \frac{x''_4 - x''_1}{L(V''_1, V''_4)}, \cos C = \frac{x''_4 - x''_1}{L(V''_1, V''_4)}$$

$\Delta\phi_2$ を定数のみで求めることができることがわかった。

この解を得るために、この連立方程式に与えた条件について整理すると、

$$L(V_1, V_2) = L(V_3, V_4) = 1 \quad (3.8)$$

$$\frac{\Delta\phi_1 + \Delta\phi_3}{2} = \Delta\phi_2 \quad (3.9)$$

$$\alpha_2 = 1 \quad (3.10)$$

以上の3つの条件を与えたことがわかる。条件(3.8)は辺の長さが同じという条件である。現在のモデルでは線材モデルを使用しているので、この条件を使用できるが、小片が線材以外になると2頂点間の距離が複数通りになる場合もありうる。例として小片が正方形のとき図3.1.5.2のような状態では $L(V_1, V_2) = L(V_3, V_4)$ という条件が使えない状況になることがある。

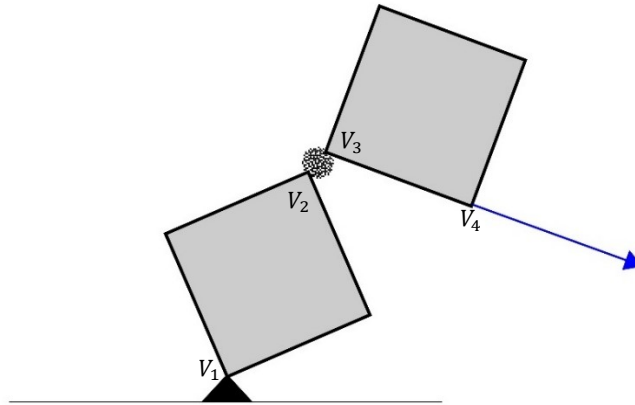


図 3.1.5.2 Fuzzy Node で接合された部材に変位を加える様子2

条件(3.9)は Fuzzy Node の変形角はそれを挟む2部材の変形角の平均に等しいという条件である。この条件が与える影響は描画したときの考察で述べる。

条件(3.10)は Fuzzy Node が伸縮しないという条件である。前項で述べたように、Fuzzy Node の伸縮度合いが全体形に与える影響はほとんどないので、この条件を与えた。 α の値を制御することで全体系の変形を制御できると予測される。今後は、 α の与える影響を検証していく必要がある。

次に、同じ回転制御法であるが線材モデル以外でも使うことのできるものを提案する。図3.1.1.1のモデルにおいて考察する。変換行列の式は式(3.2)から始めるものとする。以下にその式を記す。

$$\begin{Bmatrix} x_4'' - x_1'' \\ y_4'' - y_1'' \end{Bmatrix} = [Trans(V_1, V_2) \quad Trans(V_2, V_3) \quad Trans(V_3, V_4)] \begin{Bmatrix} \cos \Delta\theta_1 \\ \sin \Delta\theta_1 \\ \alpha_2 \times \cos \Delta\theta_2 \\ \alpha_2 \times \sin \Delta\theta_2 \\ \cos \Delta\theta_3 \\ \sin \Delta\theta_3 \end{Bmatrix}$$

これを線材モデルで行った変形と同じような変形を行う。

$$\begin{cases} x''_4 - x''_1 \\ y''_4 - y''_1 \end{cases} = \begin{bmatrix} L(V_1, V_2) & 0 & L(V_2, V_3) & 0 & L(V_3, V_4) & 0 \\ 0 & L(V_1, V_2) & 0 & L(V_2, V_3) & 0 & L(V_3, V_4) \end{bmatrix} \begin{cases} \cos \theta''_1 \\ \sin \theta''_1 \\ \alpha_2 \times \cos \theta''_2 \\ \alpha_2 \times \sin \theta''_2 \\ \cos \theta''_3 \\ \sin \theta''_3 \end{cases}$$

ここでは $L(V_1, V_2) \neq L(V_3, V_4)$ と仮定して解く。

これから先はこの行列を変形して解を求める。

上記の行列を計算する。

$$\begin{cases} x''_4 - x''_1 = L(V_1, V_2) \cos \theta''_1 + \alpha_2 L(V_2, V_3) \cos \theta''_2 + L(V_3, V_4) \cos \theta''_3 \\ y''_4 - y''_1 = L(V_1, V_2) \sin \theta''_1 + \alpha_2 L(V_2, V_3) \sin \theta''_2 + L(V_3, V_4) \sin \theta''_3 \end{cases}$$

θ''_2 の項を左辺に移動すし、少し変形する。

$$\begin{cases} x''_4 - x''_1 - \alpha_2 L(V_2, V_3) \cos \theta''_2 = L(V_1, V_2) \cos \theta''_1 + L(V_3, V_4) \cos \theta''_3 \\ y''_4 - y''_1 - \alpha_2 L(V_2, V_3) \sin \theta''_2 = L(V_1, V_2) \sin \theta''_1 + L(V_3, V_4) \sin \theta''_3 \end{cases}$$

ここで

$$A = \frac{\theta''_1 + \theta''_3}{2}, B = \frac{\theta''_1 - \theta''_3}{2} \quad \text{とおく}$$

連立方程式の上の式の右辺において

$$\begin{aligned} & L(V_1, V_2) \cos \theta''_1 + L(V_3, V_4) \cos \theta''_3 \\ &= L(V_1, V_2) \cos(A + B) + L(V_3, V_4) \cos(A - B) \\ &= (L(V_1, V_2) + L(V_3, V_4)) \cos B \cos A - (L(V_1, V_2) - L(V_3, V_4)) \sin B \sin A \end{aligned}$$

連立方程式の下式の右辺においても同様に計算して

$$\begin{aligned} & L(V_1, V_2) \sin \theta''_1 + L(V_3, V_4) \sin \theta''_3 \\ &= L(V_1, V_2) \sin(A + B) + L(V_3, V_4) \sin(A - B) \\ &= (L(V_1, V_2) + L(V_3, V_4)) \cos B \sin A - (L(V_1, V_2) - L(V_3, V_4)) \sin B \cos A \end{aligned}$$

と変形できる。

さらに $C^2 = ((L(V_1, V_2) + L(V_3, V_4)) \cos B)^2 + ((L(V_1, V_2) - L(V_3, V_4)) \sin B)^2$ とおくと

連立方程式は次のように変形することができる。

$$\begin{cases} x''_4 - x''_1 - \alpha_2 L(V_2, V_3) \cos \varnothing''_2 = C \cos(A + C_1) \\ y''_4 - y''_1 - \alpha_2 L(V_2, V_3) \sin \varnothing''_2 = C \sin(A + C_1) \end{cases}$$

$$\text{ただし、} \sin C_1 = \frac{(L(V_1, V_2) + L(V_3, V_4)) \cos B}{C}, \cos C_1 = \frac{(L(V_1, V_2) - L(V_3, V_4)) \sin B}{C}$$

連立方程式の2乗の和をとるので

まず、左辺の2乗の和を計算する。三角関数の合成を行う。

$$\begin{aligned} & (x''_4 - x''_1 - \alpha_2 L(V_2, V_3) \cos \varnothing''_2)^2 + (y''_4 - y''_1 - \alpha_2 L(V_2, V_3) \sin \varnothing''_2)^2 \\ &= L^2(V''_1, V''_4) + \{\alpha_2 L(V_2, V_3)\}^2 - 2\alpha_2 L(V''_1, V''_4) L(V_2, V_3) \cos(\varnothing''_2 - D_1) \end{aligned}$$

ここで式を整理するために定数の項を $const_n$ でまとめる (α_2 を定数とみなす)

$$= const_1 - 2\alpha_2 L(V''_1, V''_4) L(V_2, V_3) \cos(\varnothing''_2 - D_1)$$

$$\text{ただし、} \sin D_1 = \frac{y''_4 - y''_1}{L(V''_1, V''_4)}, \cos D_1 = \frac{x''_4 - x''_1}{L(V''_1, V''_4)},$$

$$const_1 = L^2(V''_1, V''_4) + \{\alpha_2 L(V_2, V_3)\}^2$$

次に右辺の2乗の和を計算する。

$$(C \cos(A + C_1))^2 + (C \sin(A + C_1))^2$$

$$= C^2$$

$$= ((L(V_1, V_2) + L(V_3, V_4)) \cos B)^2 + ((L(V_1, V_2) - L(V_3, V_4)) \sin B)^2$$

展開し、三角関数の合成を行う。

$$= L^2(V_1, V_2) + L^2(V_3, V_4) + 2L(V_1, V_2) L(V_3, V_4) (\cos B - \sin B)$$

$$= const_2 + 2\sqrt{2} L(V_1, V_2) L(V_3, V_4) \cos(B + \frac{\pi}{4})$$

$$\text{ただし } const_2 = L^2(V_1, V_2) + L^2(V_3, V_4)$$

これより、連立方程式の2乗の和は

$$const_3 = 2\alpha_2 L(V''_1, V''_4) L(V_2, V_3) \cos(\varnothing''_2 - D_1) + 2\sqrt{2} L(V_1, V_2) L(V_3, V_4) \cos(B + \frac{\pi}{4})$$

となる。ただし、 $const_3 = const_1 - const_2$

三角関数の中を変数と定数にわけると、

$$\phi''_2 - D_1 = \phi_2 + \Delta\phi_2 - D_1 = \Delta\phi_2 - \text{const}_4$$

$$B + \frac{\pi}{4} = \frac{\phi''_1 - \phi''_3}{2} + \frac{\pi}{4} = \frac{\Delta\phi_1 - \Delta\phi_3}{2} - \text{const}_5 \text{ と置き換えることができる。}$$

$$\text{このとき } \text{const}_4 = \phi_2 - D_1, \text{const}_5 = \frac{\phi''_1 - \phi''_3}{2} + \frac{\pi}{4} \text{ と定数になっている。}$$

$$\text{ここで、} \Delta\phi_2 = \frac{\Delta\phi_1 - \Delta\phi_3}{2} \text{ とすると、式は以下のように。}$$

$$\text{const}_3 = 2\alpha_2 L(V''_1, V''_4) L(V_2, V_3) \cos(\Delta\phi_2 - \text{const}_4) + 2\sqrt{2} L(V_1, V_2) L(V_3, V_4) \cos(\Delta\phi_2 - \text{const}_5)$$

$$\text{ここで、} \text{const}_6 = 2\alpha_2 L(V''_1, V''_4) L(V_2, V_3), \text{const}_6 = 2\sqrt{2} L(V_1, V_2) L(V_3, V_4) \text{ とすると}$$

$$\text{const}_6 \cos(\Delta\phi_2 - \text{const}_4) + \text{const}_7 \cos(\Delta\phi_2 - \text{const}_5) - \text{const}_3 = 0 \text{ となる。}$$

加法定理を用いて展開し、まとめると次のようになる。

$$\text{const}_3 = (\text{const}_6 \cos \text{const}_4 + \text{const}_7 \cos \text{const}_5) \cos \Delta\phi_2 + (\text{const}_6 \sin \text{const}_4 + \text{const}_7 \sin \text{const}_5) \sin \Delta\phi_2$$

ここで、三角関数の合成を使用する。

$$\text{const}_3 = \text{const}_7 \cos(\Delta\phi_2 - \text{const}_8)$$

$$\text{ただし、} \sin \text{const}_8 = \frac{(\text{const}_6 \sin \text{const}_4 + \text{const}_7 \sin \text{const}_5)}{\text{const}_7}$$

$$\cos = \frac{(\text{const}_6 \cos \text{const}_4 + \text{const}_7 \cos \text{const}_5)}{\text{const}_7}$$

$$\text{const}_7 = \sqrt{(\text{const}_6 \cos \text{const}_4 + \text{const}_7 \cos \text{const}_5)^2 + (\text{const}_6 \sin \text{const}_4 + \text{const}_7 \sin \text{const}_5)^2}$$

これより、 \cos の逆関数を用いると

$$\Delta\phi_2 = \text{Arccos} \left(\frac{\text{const}_3}{\text{const}_7} \right) + \text{const}_8 \text{ となる。}$$

回転角を求められる。

以上の計算より $\Delta\phi_2$ の値を求められることが示された。全部を文字のまま計算すると式が冗長になってしまうのでここでは行わないが、計算の途中で定数であるものをまとめている個所に計算した値を代入すれば最後まで値を数値解析により算出できることが確認できた。

3.1.6 制約近似法

回転角制御法では三角関数の出現により膨大な計算量が必要とされた。そこで、今回はその解決法の1つとして制約近似法を提案する。この方法は部材に変位を与えた際、その変位を微小な大きさに分割し増分計算を行う方法である。微小な計算にすることで三角関数の近似を可能にし、計算の単純化が可能になっている。

図3.1のモデルにおいて考察する。変換行列は式(3.2)からはじめるものとする。以下にその式を記す。

$$\begin{Bmatrix} x''_4 - x''_1 \\ y''_4 - y''_1 \end{Bmatrix} = [Trans(V_1, V_2) \quad Trans(V_2, V_3) \quad Trans(V_3, V_4)] \begin{Bmatrix} \cos \Delta\theta_1 \\ \sin \Delta\theta_1 \\ \alpha_2 \times \cos \Delta\theta_2 \\ \alpha_2 \times \sin \Delta\theta_2 \\ \cos \Delta\theta_3 \\ \sin \Delta\theta_3 \end{Bmatrix}$$

これより制約近似法を示す。

まず、制約近似法の近似の部分を示す。

分割した変位を微小なものとし、 $\sin \Delta\theta_n \approx \Delta\theta_n, \cos \Delta\theta_n \approx 1$ と置換する

次に、制約について考察する。

Fuzzy Node の回転角の変化が両端の部材の回転角の和で表されるとする。

任意の定数 C_1 と C_3 を用いて次のような条件を与える。

$$\Delta\theta_2 = C_1\Delta\theta_1 + C_3\Delta\theta_3 \quad (3.11)$$

また、Fuzzy Node の長さを表す変数 α_2 を1とする。

変換行列は以下のように書き換えることが可能となる。

$$\begin{Bmatrix} x''_4 - x''_1 \\ y''_4 - y''_1 \end{Bmatrix} = [Trans(V_1, V_2) \quad Trans(V_2, V_3) \quad Trans(V_3, V_4)] \begin{Bmatrix} 1 \\ \Delta\theta_1 \\ 1 \\ C_1\Delta\theta_1 + C_3\Delta\theta_3 \\ 1 \\ \Delta\theta_3 \end{Bmatrix}$$

$[Trans(V_1, V_2)] = [Trans_1(V_1, V_2) \quad Trans_2(V_1, V_2)]$ と置き、行列を計算する。

$$\begin{aligned} \begin{Bmatrix} x''_4 - x''_1 \\ y''_4 - y''_1 \end{Bmatrix} &= Trans_1(V_1, V_2) + Trans_1(V_2, V_3) + Trans_1(V_3, V_4) + \Delta\theta_1 Trans_2(V_1, V_2) \\ &\quad + (C_1\Delta\theta_1 + C_3\Delta\theta_3)Trans_2(V_2, V_3) + \Delta\theta_3 Trans_2(V_3, V_4) \end{aligned}$$

$$\begin{aligned} \begin{Bmatrix} x''_4 - x''_1 \\ y''_4 - y''_1 \end{Bmatrix} &= Trans_1(V_1, V_4) + (C_1Trans_2(V_2, V_3) + Trans_2(V_1, V_2))\Delta\theta_1 \\ &\quad + (C_3Trans_2(V_2, V_3) + Trans_2(V_3, V_4))\Delta\theta_3 \end{aligned}$$

定数を左辺、変数を右辺に分けると次の党になる。

$$\begin{aligned} \begin{Bmatrix} x''_4 - x''_1 \\ y''_4 - y''_1 \end{Bmatrix} - Trans_1(V_1, V_4) \\ = [C_1Trans_2(V_2, V_3) + Trans_2(V_1, V_2) \quad C_3Trans_2(V_2, V_3) + Trans_2(V_3, V_4)] \begin{Bmatrix} \Delta\theta_1 \\ \Delta\theta_3 \end{Bmatrix} \end{aligned}$$

ここで、以下のように行列を[A]、[B]とおく。

$$[A] = \begin{Bmatrix} x''_4 - x''_1 \\ y''_4 - y''_1 \end{Bmatrix} - Trans_1(V_1, V_4)$$

$$[B] = [C_1Trans_2(V_2, V_3) + Trans_2(V_1, V_2) \quad C_3Trans_2(V_2, V_3) + Trans_2(V_3, V_4)]$$

すると、次のような連立一次方程式で表すことが可能となる。

$$[A] = [B] \begin{Bmatrix} \Delta\theta_1 \\ \Delta\theta_3 \end{Bmatrix}$$

$\det([B]) \neq 0$ のとき、

$$\therefore \begin{Bmatrix} \Delta\theta_1 \\ \Delta\theta_3 \end{Bmatrix} = [B]^{-1}[A] \text{ と解を求められることがわかる。}$$

回転角制御法と比較すると、制約近似法のほうが容易に解を求められる。しかし、制約近似法では近似による誤差を処理する必要があるが生じる。ここでは2通りの方法を示す。

誤差の解決方法の1つが Fuzzy Node の伸縮にすべての誤差を吸収させる方法である。図 3.1.1.1 において V''_1 と V''_4 は既知なので、さきほど示した解法から $\Delta\theta_1$ と $\Delta\theta_3$ が求まり、部材1と部材3がどのような角度でどの位置に存在するのかが求められる。それぞれの部材の位置を求めた後に、2つの節点 V''_2 と V''_3 をつなぐ Fuzzy Node が最大長を超えない範囲でなら V''_4 を移動できると定義するのが Fuzzy Node を持つ構造体の機構解析の近似法を用いた1つの解だといえる。

もう1つの方法は条件を満たす結果を収束計算で求める方法である。元のモデルに微小な変位を与えた際の $\Delta\theta_1$ と $\Delta\theta_3$ を求め、固定された V''_1 から部材1、FN2、部材3の位置を順に計算する。このとき、近似の誤差によって実際に与えた強制変位とは異なる位置に節点 V''_4 が求まる。近似の誤差によってずれた節点を dV''_4 とする。新たに求まった節点 dV''_4 を目標の点に向かってまた微小な変位を与える。これを繰り返す行いことで誤差によって修正された dV''_4 が次第に目標としている点に近づく。 dV''_4 が目標としている点に十分に接近したときに計算を終る方法も1つの解決策である。

以上の2つの方法があげられる。前者を制約近似増分法、後者を制約近似収束法と呼ぶ。それぞれの解法の説明を図を用いて行う。結果を見やすくするために小片は細い線材のような形状なものと仮定し図示する。はじめに、図3.1.6.1のように線材に変位を加える。

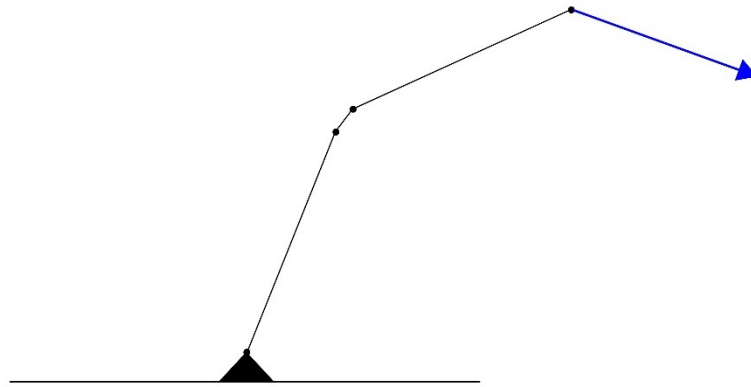


図 3.1.6.1 Fuzzy Node で接合されたモデルに変位を加える様子

回転角を近似できるようにするため、変位を図3.1.6.2のように微小な変位に分割する。

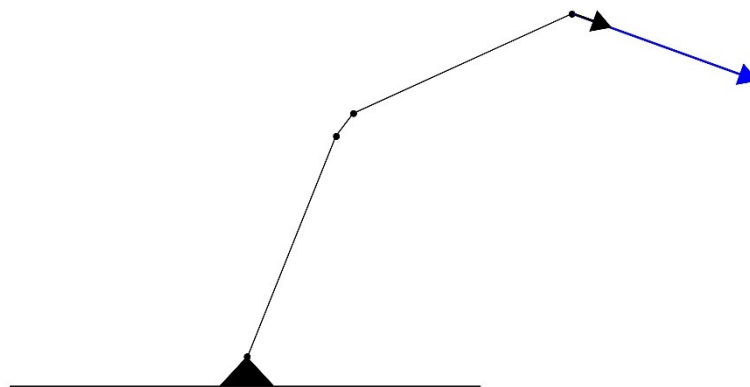


図 3.1.6.2 変位を微小なサイズに分割する

制約近似増分法を適用し、線材の位置を算出すると図3.1.6.3のように求まる。

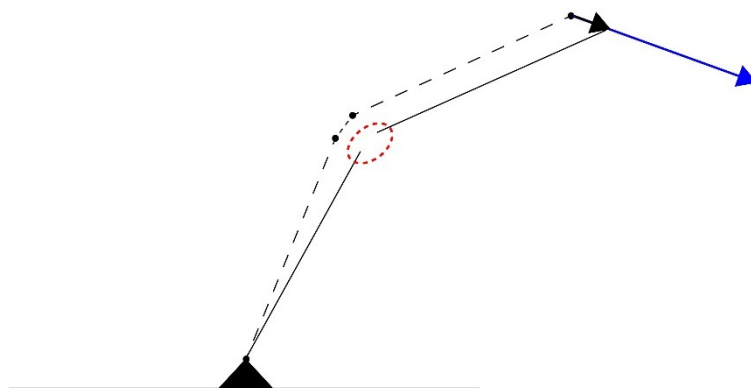


図 3.1.6.3 微小変形後の部材の位置を求める様子

このとき、赤い点線で囲まれた部分が Fuzzy Node で接合されている部分である。2点間の距離が Fuzzy Node の最大長以下なら次の増分計算に移行し、もし最大長を超えていたら最初に与えた変位が不適切であるとし、計算を終了する。次の増分計算に移行するときは、図 3.1.6.4 のようになる。

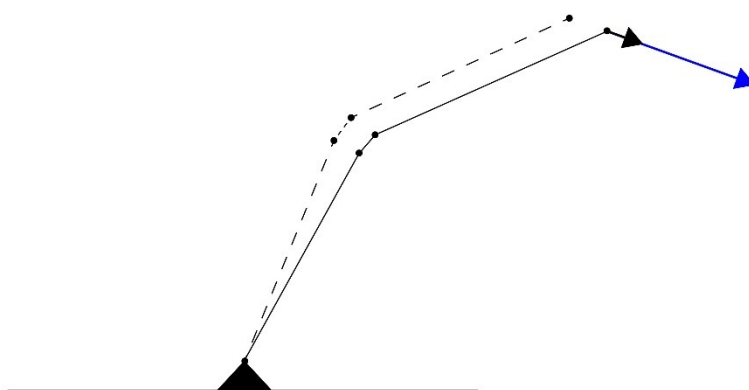


図 3.1.6.4 制約近似増分法で次の微小変位の計算を行う様子

このように変位を分割した分だけ増分計算をし、変位を与えた後の形状を求めているので制約近似増分法と呼んでいる。

次に制約近似収束法について説明する。

図 3.1.6.2 の状態から制約近似収束法を適用すると図 3.1.6.5 のようになる。

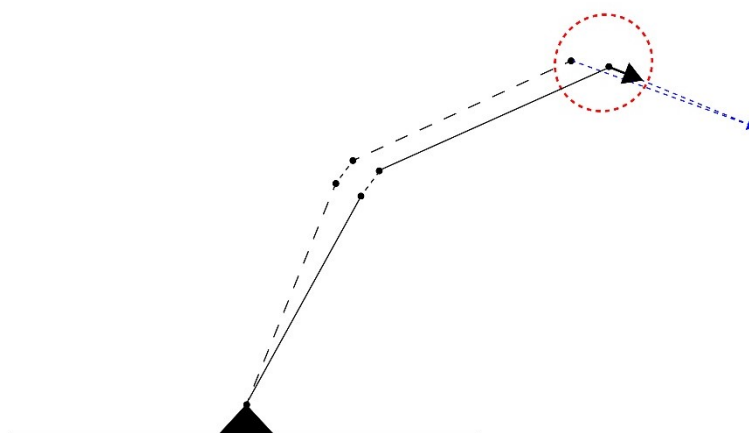


図 3.15 制約近似収束法を適用した様子

このとき、赤い点線で囲まれた部分に注目する。与えた変位から幾何学的に求まる節点と計算から求めた節点の位置がわずかにずれている。この近似で生じた誤差を次の計算に引き継ぎながら、計算するのがこの方法の特徴である。引き継いだ結果、次の変位は図 3.1.6.6 のようになる。

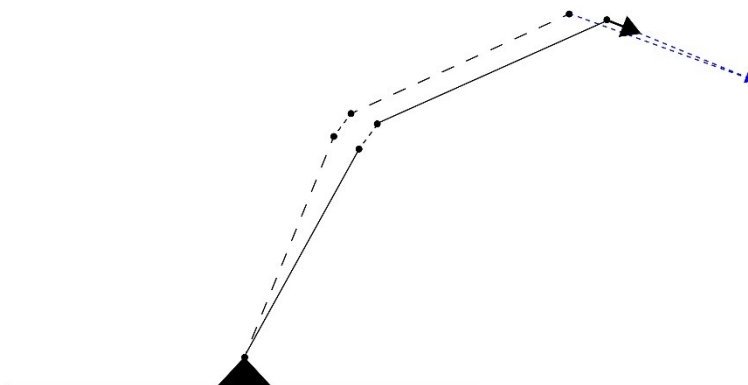


図 3.1.6.6 制約近似収束法で次の微小変位の計算を行う様子

この計算は制約近似増分法とは異なり、一回の微小計算で目標とする点にどの程度近づくのか予測できないため、何回繰り返し計算を行えばよいのか明らかでない。多少の誤差は避けることができないため目標の点との距離あらかじめ設定した値より小さくなった時点で計算を終了させる必要がある。これが収束計算となっているので解法の名前に収束と入れている。この解法は収束しない可能性があり、現状どのような場合に収束しないのか明らかにできていないので本論ではまだ推奨しないものとする。

以降、本論で扱う制約近似法は制約近似増分法をさす。

3.2 解を反復計算で求める方法

部材の位置を反復計算により求める方法を示す。

3.2.1 重心追従法

解を徐々に計算する方法として、重心追従法を提案する。この方法は部材に変位を与えたときその部材の重心の移動が最小となるように移動させるという方法である。

図 3.17 のように1つの正方形の部材の節点に変位を与えたときの挙動について考察する。部材1の重心 $O_1(ox_1, oy_1)$ とおき頂点 $V_1(x_1, y_1)$ をとり、 $V_1(x_1, y_1)$ に変位を与え $V''_1(x''_1, y''_1)$ に移動させる。

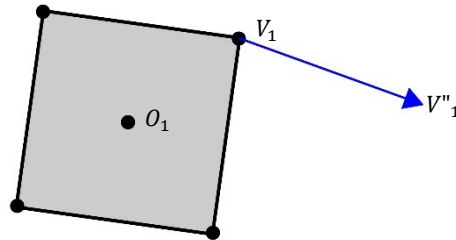


図 3.17 部材に変位を与える様子

変位を加えた後の部材の中心の位置を求める。

2点 O_1 と V_1 の距離 $L(O_1, V_1)$ は一定である。

それゆえ O''_1 は V''_1 を中心とする半径 $L(O_1, V_1)$ の円周上にある。

$$O''_1 = \begin{pmatrix} ox''_1 \\ oy''_1 \end{pmatrix} = \begin{pmatrix} x''_1 + L(O_1, V_1) \cos \theta \\ y''_1 + L(O_1, V_1) \sin \theta \end{pmatrix} \quad \text{ただし } 0 \leq \theta \leq 2\pi \text{ となる。}$$

これより、2点 O_1, O''_1 の距離の二乗は次のように計算できる。

$$L^2(O_1, O''_1) = (ox''_1 - ox_1)^2 + (oy''_1 - oy_1)^2$$

$$L^2(O_1, O''_1) = (x''_1 + L(O_1, V_1) \cos \theta - ox_1)^2 + (y''_1 + L(O_1, V_1) \sin \theta - oy_1)^2$$

$$L^2(O_1, O''_1) = L^2(O_1, V''_1) + L^2(O_1, V_1) + 2L(O_1, V''_1)L(O_1, V_1) \cos(\theta - \alpha)$$

$$\text{ただし } \sin \alpha = \frac{y''_1 - oy_1}{L(O_1, V''_1)}, \cos \alpha = \frac{x''_1 - ox_1}{L(O_1, V''_1)}$$

2点 O_1, O''_1 の距離の最小値を求める。

$$L^2(O_1, V''_1) + L^2(O_1, V_1) - 2L(O_1, V''_1)L(O_1, V_1) \leq L^2(O_1, O''_1)$$

$$(L^2(O_1, V''_1) - L(O_1, V_1))^2 \leq L^2(O_1, O''_1)$$

最小となるときの等号成立条件は $\cos(\theta - \alpha) = -1$ である。

$$\therefore \theta - \alpha = -\pi$$

$$\theta = \alpha - \pi$$

α は直線 $O_1V''_1$ とx軸のなす角度であるので

$\alpha - \pi$ はベクトル V''_1O_1 のx軸とのなす角度になる。

そのため、移動後の部材の中心 O''_1 は直線 $O_1V''_1$ 上に存在することがわかる。

このように部材を引っ張った際に移動後の重心が元の重心と変位を与えた点を結ぶ直線上に来る。重心が変位に追従しているように動くことから重心追従法と名付けられている。部材に重心追従法を適用すると図 3.2.1.1 のようになる。

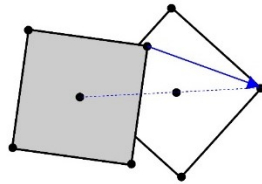


図 3.2.1.1 部材追従法における部材に変位を与えた際の部材の移動

次に部材が2つ以上あるときの変位の伝達を考える。図 3.2.1.2 のように Fuzzy Node で接合された2つの正方形部材で考察する。図 3.2.1.2 (左) のように部材に変位を与えると、部材が移動し、接合している Fuzzy Node の長さが長くなることもある。Fuzzy Node の長さが自身の最大長を超えるときはその超過分を図 3.2.1.2 (中) のように次の部材に与える変位とみなすように決める。新しい変位を部材に与えて、重心追従法を適用し位置を求める。結果として図 3.2.1.2 (右) のように移動する。

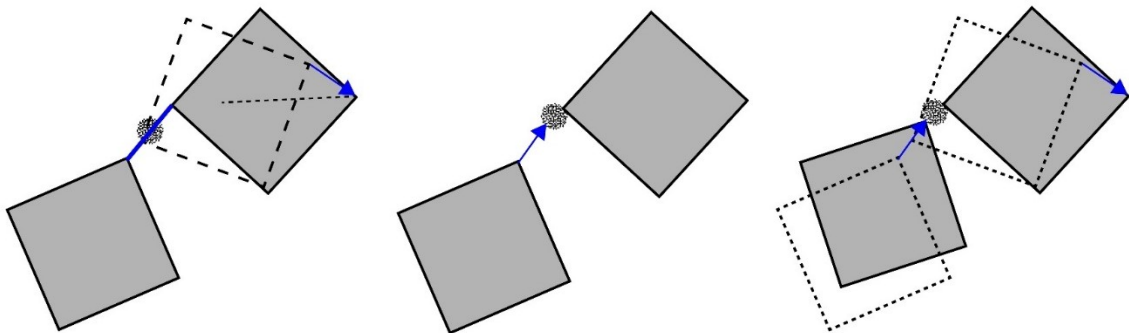


図 3.19 重心追従法の2部材への適用

第4章 複数の Fuzzy Node を持つ構造体への応用

4.1 解を一度の計算で求める解法の実用

4.1.1 Fuzzy Node を持たない環状モデルの考察

複数の Fuzzy Node を持つ集積体への応用を考える。これにあたりまずは Fuzzy Node を持たない環状の集積体について考察する。以下の図 4.1.1.1 の正方形の部材で構成された平面の環状モデルにおける変換行列とその解法について述べる。

図 4.1.1.1 の環状モデルにおいて 5 点 $V_1(x_1, y_1)$ 、 $V_2(x_2, y_2)$ 、 $V_3(x_3, y_3)$ 、 $V_4(x_4, y_4)$ 、 $V_5(x_5, y_5)$ をとる。また部材 1、部材 2、部材 3 それぞれの重心の座標を $O_1(ox_1, oy_1)$ 、 $O_2(ox_2, oy_2)$ 、 $O_3(ox_3, oy_3)$ をおく。点 V''_5 については移動後の x 座標の x''_5 のみ既知であるものとする。

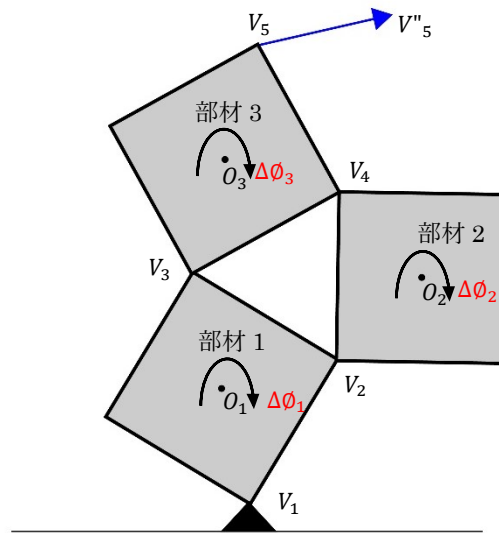


図 4.1.1.1 No Fuzzy の環状モデル

変換行列は次のように表すことができる。このとき、 $\Delta\phi_1, \Delta\phi_2, \Delta\phi_3$ は部材 1、部材 2、部材 3 の回転角を表しており、未知数となっている。

$$\begin{pmatrix} x''_1 - ox''_1 \\ y''_1 - oy''_1 \\ ox''_1 - ox''_2 \\ oy''_1 - oy''_2 \\ ox''_3 - ox''_1 \\ oy''_3 - oy''_1 \\ ox''_2 - ox''_3 \\ oy''_2 - oy''_3 \\ ox''_3 - x''_5 \\ oy''_3 - y''_5 \end{pmatrix} = \begin{bmatrix} Trans(O_1, V_1) & 0 & 0 \\ Trans(V_2, O_1) & Trans(O_2, V_2) & 0 \\ Trans(O_1, V_3) & 0 & Trans(V_3, O_2) \\ 0 & Trans(V_4, O_2) & Trans(O_3, V_4) \\ 0 & 0 & Trans(V_5, O_3) \end{bmatrix} \begin{pmatrix} \cos \Delta\phi_1 \\ \sin \Delta\phi_1 \\ \cos \Delta\phi_2 \\ \sin \Delta\phi_2 \\ \cos \Delta\phi_3 \\ \sin \Delta\phi_3 \end{pmatrix}$$

左辺より ox''_1, oy''_1 を消去する。

$$\begin{cases} x''_1 - ox''_1 \\ y''_1 - oy''_1 \\ x''_1 - ox''_2 \\ x''_1 - oy''_2 \\ ox''_3 - x''_1 \\ oy''_3 - x''_1 \\ ox''_2 - ox''_3 \\ oy''_2 - oy''_3 \\ ox''_3 - x''_5 \\ oy''_3 - y''_5 \end{cases} = \begin{bmatrix} \text{Trans}(O_1, V_1) & 0 & 0 \\ \text{Trans}(V_2, V_1) & \text{Trans}(O_2, V_2) & 0 \\ \text{Trans}(V_1, V_3) & 0 & \text{Trans}(V_3, O_2) \\ 0 & \text{Trans}(V_4, O_2) & \text{Trans}(O_3, V_4) \\ 0 & 0 & \text{Trans}(V_5, O_3) \end{bmatrix} \begin{cases} \cos \Delta\phi_1 \\ \sin \Delta\phi_1 \\ \cos \Delta\phi_2 \\ \sin \Delta\phi_2 \\ \cos \Delta\phi_3 \\ \sin \Delta\phi_3 \end{cases}$$

左辺より ox''_3 、 oy''_3 を消去する。

$$\begin{cases} x''_1 - ox''_1 \\ y''_1 - oy''_1 \\ x''_1 - ox''_2 \\ x''_1 - oy''_2 \\ x''_5 - x''_1 \\ y''_5 - x''_1 \\ ox''_2 - x''_5 \\ oy''_2 - y''_5 \\ ox''_3 - x''_5 \\ oy''_3 - y''_5 \end{cases} = \begin{bmatrix} \text{Trans}(O_1, V_1) & 0 & 0 \\ \text{Trans}(V_2, V_1) & \text{Trans}(O_2, V_2) & 0 \\ \text{Trans}(V_1, V_3) & 0 & \text{Trans}(V_3, V_5) \\ 0 & \text{Trans}(V_4, O_2) & \text{Trans}(V_5, V_4) \\ 0 & 0 & \text{Trans}(V_5, O_3) \end{bmatrix} \begin{cases} \cos \Delta\phi_1 \\ \sin \Delta\phi_1 \\ \cos \Delta\phi_2 \\ \sin \Delta\phi_2 \\ \cos \Delta\phi_3 \\ \sin \Delta\phi_3 \end{cases}$$

左辺より ox''_2 、 oy''_2 を消去する。

$$\begin{cases} x''_1 - ox''_1 \\ y''_1 - oy''_1 \\ x''_1 - x''_5 \\ y''_1 - y''_5 \\ x''_5 - x''_1 \\ y''_5 - y''_1 \\ ox''_2 - x''_5 \\ oy''_2 - y''_5 \\ ox''_3 - x''_5 \\ oy''_3 - y''_5 \end{cases} = \begin{bmatrix} \text{Trans}(O_1, V_1) & 0 & 0 \\ \text{Trans}(V_2, V_1) & \text{Trans}(V_4, V_2) & \text{Trans}(V_5, V_4) \\ \text{Trans}(V_1, V_3) & 0 & \text{Trans}(V_3, V_5) \\ 0 & \text{Trans}(V_4, O_2) & \text{Trans}(V_5, V_4) \\ 0 & 0 & \text{Trans}(V_5, O_3) \end{bmatrix} \begin{cases} \cos \Delta\phi_1 \\ \sin \Delta\phi_1 \\ \cos \Delta\phi_2 \\ \sin \Delta\phi_2 \\ \cos \Delta\phi_3 \\ \sin \Delta\phi_3 \end{cases}$$

3行目から6行目までに注目する。

$$\begin{cases} x''_1 - x''_5 \\ y''_1 - y''_5 \\ x''_5 - x''_1 \\ y''_5 - y''_1 \end{cases} = \begin{bmatrix} x_1 - x_2 & -(y_1 - y_2) & x_2 - x_4 & -(y_2 - y_4) & x_4 - x_5 & -(y_4 - y_5) \\ y_1 - y_2 & x_1 - x_2 & y_2 - y_4 & x_2 - x_4 & y_4 - y_5 & x_4 - x_5 \\ x_3 - x_1 & -(y_3 - y_1) & 0 & 0 & x_5 - x_3 & -(y_5 - y_3) \\ y_3 - y_1 & x_3 - x_1 & 0 & 0 & y_5 - y_3 & x_5 - x_3 \end{bmatrix} \begin{cases} \cos \Delta\phi_1 \\ \sin \Delta\phi_1 \\ \cos \Delta\phi_2 \\ \sin \Delta\phi_2 \\ \cos \Delta\phi_3 \\ \sin \Delta\phi_3 \end{cases}$$

y''_5 は未知数なのでこれを縮約する。

$$\begin{pmatrix} x''_1 - x''_5 \\ 0 \\ x''_5 - x''_1 \\ y''_5 - y''_1 \end{pmatrix} = \begin{bmatrix} x_1 - x_2 & -(y_1 - y_2) & x_2 - x_4 & -(y_2 - y_4) & x_4 - x_5 & -(y_4 - y_5) \\ y_3 - y_2 & x_3 - x_2 & y_2 - y_4 & x_2 - x_4 & y_4 - y_3 & x_4 - x_3 \\ x_3 - x_1 & -(y_3 - y_1) & 0 & 0 & x_5 - x_3 & -(y_5 - y_3) \\ y_3 - y_1 & x_3 - x_1 & 0 & 0 & y_5 - y_3 & x_5 - x_3 \end{bmatrix} \begin{pmatrix} \cos \Delta\theta_1 \\ \sin \Delta\theta_1 \\ \cos \Delta\theta_2 \\ \sin \Delta\theta_2 \\ \cos \Delta\theta_3 \\ \sin \Delta\theta_3 \end{pmatrix}$$

4つの変数 $\Delta\theta_1, \Delta\theta_2, \Delta\theta_3, y''_5$ に対して4つの式なので解が求められることがわかる。また、1～3行には3つの変数 $\Delta\theta_1, \Delta\theta_2, \Delta\theta_3$ しか現れないので4行目以外を考えることで3つの変数 $\Delta\theta_1, \Delta\theta_2, \Delta\theta_3$ を求めることができる。次のように4行目をなくして考える。

$$\begin{pmatrix} x''_1 - x''_5 \\ 0 \\ 0 \end{pmatrix} = \begin{bmatrix} x_1 - x_2 & -(y_1 - y_2) & x_2 - x_4 & -(y_2 - y_4) & x_4 - x_5 & -(y_4 - y_5) \\ x_3 - x_2 & -(y_3 - y_2) & x_2 - x_4 & -(y_2 - y_4) & x_4 - x_3 & -(y_4 - y_3) \\ y_3 - y_2 & x_3 - x_2 & y_2 - y_4 & x_2 - x_4 & y_4 - y_3 & x_4 - x_3 \end{bmatrix} \begin{pmatrix} \cos \Delta\theta_1 \\ \sin \Delta\theta_1 \\ \cos \Delta\theta_2 \\ \sin \Delta\theta_2 \\ \cos \Delta\theta_3 \\ \sin \Delta\theta_3 \end{pmatrix}$$

ここで、平面の回転行列を以下のように示す。

$$R(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

連立方程式の下2行を取り出す。

$$R(\Delta\theta_1) \begin{pmatrix} x_3 - x_2 \\ y_3 - y_2 \end{pmatrix} + R(\Delta\theta_2) \begin{pmatrix} x_2 - x_4 \\ y_2 - y_4 \end{pmatrix} + R(\Delta\theta_3) \begin{pmatrix} x_4 - x_3 \\ y_4 - y_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad (4.1)$$

式(4.1)を $\Delta\theta_3$ で少し変形する。

$$R(\Delta\theta_1) \begin{pmatrix} x_3 - x_2 \\ y_3 - y_2 \end{pmatrix} + R(\Delta\theta_2) \begin{pmatrix} x_2 - x_4 \\ y_2 - y_4 \end{pmatrix} + R(\Delta\theta_3) \begin{pmatrix} x_4 - x_2 + x_2 - x_3 \\ y_4 - y_2 + y_2 - y_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$R(\Delta\theta_1) \begin{pmatrix} x_3 - x_2 \\ y_3 - y_2 \end{pmatrix} + R(\Delta\theta_3) \begin{pmatrix} x_2 - x_3 \\ y_2 - y_3 \end{pmatrix} + R(\Delta\theta_2) \begin{pmatrix} x_2 - x_4 \\ y_2 - y_4 \end{pmatrix} + R(\Delta\theta_3) \begin{pmatrix} x_4 - x_2 \\ y_4 - y_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\{R(\Delta\theta_1) - R(\Delta\theta_3)\} \begin{pmatrix} x_3 - x_2 \\ y_3 - y_2 \end{pmatrix} + \{R(\Delta\theta_3) - R(\Delta\theta_2)\} \begin{pmatrix} x_4 - x_2 \\ y_4 - y_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

回転行列の差を計算する。

$$R(\alpha) - R(\beta) = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix} - \begin{bmatrix} \cos \beta & -\sin \beta \\ \sin \beta & \cos \beta \end{bmatrix}$$

$$R(\alpha) - R(\beta) = \begin{bmatrix} \cos \alpha - \cos \beta & -\sin \alpha + \sin \beta \\ \sin \alpha - \sin \beta & \cos \alpha - \cos \beta \end{bmatrix}$$

$$R(\alpha) - R(\beta) = \begin{bmatrix} -2 \sin \frac{\alpha - \beta}{2} \sin \frac{\alpha + \beta}{2} & -2 \sin \frac{\alpha - \beta}{2} \cos \frac{\alpha + \beta}{2} \\ 2 \sin \frac{\alpha - \beta}{2} \cos \frac{\alpha + \beta}{2} & -2 \sin \frac{\alpha - \beta}{2} \sin \frac{\alpha + \beta}{2} \end{bmatrix}$$

$$R(\alpha) - R(\beta) = 2 \sin \frac{\alpha - \beta}{2} \begin{bmatrix} -\sin \frac{\alpha + \beta}{2} & -\cos \frac{\alpha + \beta}{2} \\ \cos \frac{\alpha + \beta}{2} & -\sin \frac{\alpha + \beta}{2} \end{bmatrix}$$

$$R(\alpha) - R(\beta) = 2 \sin \frac{\alpha - \beta}{2} R\left(\frac{\pi}{2} + \frac{\alpha + \beta}{2}\right) \text{となる。}$$

これより

$$\begin{aligned} & \sin \frac{\Delta\theta_1 - \Delta\theta_3}{2} R\left(\frac{\pi}{2} + \frac{\Delta\theta_1 + \Delta\theta_3}{2}\right) \begin{pmatrix} x_3 - x_2 \\ y_3 - y_2 \end{pmatrix} \\ & + \sin \frac{\Delta\theta_3 - \Delta\theta_2}{2} R\left(\frac{\pi}{2} + \frac{\Delta\theta_3 + \Delta\theta_2}{2}\right) \begin{pmatrix} x_4 - x_2 \\ y_4 - y_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \end{aligned}$$

ベクトル V_2V_3 、ベクトル V_2V_4 それぞれx軸とのなす角を θ_1 、 θ_2 とおくと

$$\begin{aligned} & \sin \frac{\Delta\theta_1 - \Delta\theta_3}{2} R\left(\frac{\Delta\theta_1 + \Delta\theta_3}{2}\right) R(\theta_1) \begin{pmatrix} L(V_3, V_2) \\ 0 \end{pmatrix} \\ & + \sin \frac{\Delta\theta_3 - \Delta\theta_2}{2} R\left(\frac{\Delta\theta_3 + \Delta\theta_2}{2}\right) R(\theta_2) \begin{pmatrix} L(V_4, V_2) \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \end{aligned}$$

と変形できる。

この方程式が成立するための条件を考える。

$$1. L(V_3, V_2) \sin \frac{\Delta\theta_1 - \Delta\theta_3}{2} = L(V_4, V_2) \sin \frac{\Delta\theta_3 - \Delta\theta_2}{2} \neq 0 \text{かつ整数} n \text{を用いて}$$

$$\frac{\Delta\theta_1 + \Delta\theta_3}{2} + \theta_1 = \frac{\Delta\theta_3 + \Delta\theta_2}{2} + \theta_2 + (2n + 1)\pi \text{となる場合}$$

2. $L(V_3, V_2) \sin \frac{\Delta\phi_1 - \Delta\phi_3}{2} = -L(V_4, V_2) \sin \frac{\Delta\phi_3 - \Delta\phi_2}{2} \neq 0$ かつ整数 n を用いて

$$\frac{\Delta\phi_1 + \Delta\phi_3}{2} + \theta_1 = \frac{\Delta\phi_3 + \Delta\phi_2}{2} + \theta_2 + 2n\pi \text{となる場合}$$

3. $L(V_3, V_2) \sin \frac{\Delta\phi_1 - \Delta\phi_3}{2} = L(V_4, V_2) \sin \frac{\Delta\phi_3 - \Delta\phi_2}{2} = 0$ となる場合

以上の3通りが考えられる。

条件1について考える。

式(4.1)は $\Delta\phi_1, \Delta\phi_2, \Delta\phi_3$ の対象式である。同様に $\Delta\phi_2$ と $\Delta\phi_1$ で少し変形すると2つの整数 l, m を用いて次のような結果が得られる。

$$\theta_2 + \frac{\Delta\phi_1 + \Delta\phi_2}{2} = \theta_3 + \frac{\Delta\phi_3 + \Delta\phi_2}{2} + (2l + 1)\pi$$

$$\theta_3 + \frac{\Delta\phi_2 + \Delta\phi_3}{2} = \theta_1 + \frac{\Delta\phi_1 + \Delta\phi_2}{2} + (2m + 1)\pi \text{となる。}$$

$\Delta\phi_1, \Delta\phi_2, \Delta\phi_3, \theta_1, \theta_2, \theta_3$ は $-\pi$ 以上 π 以下なので

3つの整数 k, l, m は0または ± 1 に限定される。

この2式と $\frac{\Delta\phi_1 + \Delta\phi_3}{2} + \theta_1 = \frac{\Delta\phi_3 + \Delta\phi_2}{2} + \theta_2 + (2n + 1)\pi$ の3つの式の和をとる。

$$0 = (2k + 2l + 2m + 3)\pi \text{となる。}$$

右辺の合計は必ず奇数なのでゼロとなることはなく、この条件は成立しない。

条件2について考える。

先ほどと同様に式(4.1)は $\Delta\phi_1, \Delta\phi_2, \Delta\phi_3$ の対象式を利用し、同様に $\Delta\phi_2$ と $\Delta\phi_1$ で少し変形する。

$$\theta_3 + \frac{\Delta\phi_2 + \Delta\phi_3}{2} = \theta_1 + \frac{\Delta\phi_1 + \Delta\phi_2}{2} + 2m\pi \text{となる。}$$

$$L(V_4, V_3) \sin \frac{\Delta\phi_2 - \Delta\phi_1}{2} = -L(V_2, V_3) \sin \frac{\Delta\phi_1 - \Delta\phi_3}{2}$$

$L(V_2, V_4) \sin \frac{\Delta\phi_3 - \Delta\phi_2}{2} = -L(V_3, V_4) \sin \frac{\Delta\phi_2 - \Delta\phi_1}{2}$ が得られる。

この2式と $L(V_3, V_2) \sin \frac{\Delta\phi_1 - \Delta\phi_3}{2} = -L(V_4, V_2) \sin \frac{\Delta\phi_3 - \Delta\phi_2}{2}$ の3つ積をとる。

$1 = -1$ となるのでこの条件でも成立しない。

条件3では

$L(V_3, V_2) \neq 0$ 、 $L(V_4, V_2) \neq 0$ なので

$$\sin \frac{\Delta\phi_1 - \Delta\phi_3}{2} = \sin \frac{\Delta\phi_3 - \Delta\phi_2}{2} = 0$$

これより

$\Delta\phi_1 = \Delta\phi_2 = \Delta\phi_3$ が導き出される。

これを連立方程式に代入すると値が求められる。

これより環状のモデルにおいて Fuzzy Node がない場合は解が求められる。次は、Fuzzy Node がある場合の環状モデルでも解を導くことができるのか検討する。

4.1.2 Fuzzy Node を含む環状モデル

先ほどは Fuzzy Node をもたない環状モデルを考察したので、次は Fuzzy Node を持つ環状モデルの変換行列の解法について検討する。Fuzzy Node がないときのモデルと同様に3つの正方形部材で構成される平面モデルで考察する。ただし、このとき部材同士は3つの Fuzzy Node で接合しているものとする (図 4.1.1.2)。

図 4.1.1.1 と同じ作りではあるがすべてが Fuzzy Node で接合されている図 4.1.2.1 のモデルを使用する。図のように8点 $V_1(x_1, y_1)$ 、 $V_2(x_2, y_2)$ 、 $V_3(x_3, y_3)$ 、 $V_4(x_4, y_4)$ 、 $V_5(x_5, y_5)$ 、 $V_6(x_6, y_6)$ 、 $V_7(x_7, y_7)$ 、 $V_8(x_8, y_8)$ をとり、それぞれを部材1、部材2、部材3、FN(Fuzzy Node)4、FN5、FN6と名付け、各重心を $O_1(ox_1, oy_1)$ 、 $O_2(ox_2, oy_2)$ 、 $O_3(ox_3, oy_3)$ 、 $O_4(ox_4, oy_4)$ 、 $O_5(ox_5, oy_5)$ 、 $O_6(ox_6, oy_6)$ とおく。点 V_8 の変位については Fuzzy Node がないモデルのときと同様に x 座標の変位先である x_8 のみ既知であるものとする。

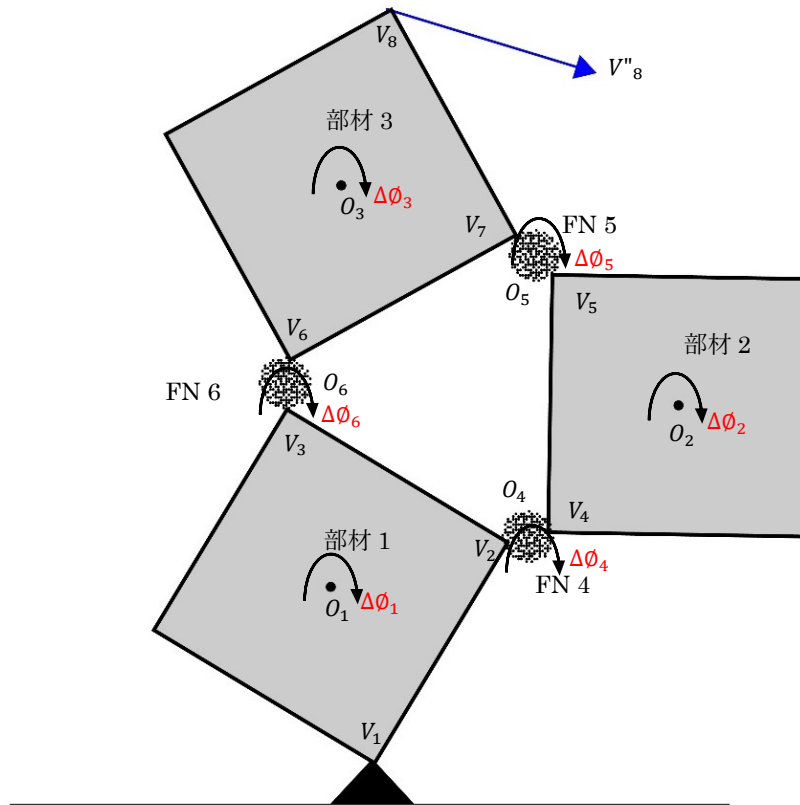


図 4.1.2.1 Fuzzy Node を持つ環状のモデル

このモデルの変換行列は次のようになる。Fuzzy Node 長さを表す係数は全て 1 であると仮定している。また、 $\Delta\phi_1, \Delta\phi_2, \Delta\phi_3, \Delta\phi_4, \Delta\phi_5, \Delta\phi_6$ は部材 1、部材 2、部材 3、FN(Fuzzy Node) 4、FN 5、FN 6 の回転角を表しており、未知数となっている。

$$\begin{cases} x''_1 - ox''_1 \\ y''_1 - oy''_1 \\ ox''_1 - ox''_2 \\ oy''_1 - oy''_2 \\ ox''_3 - ox''_1 \\ oy''_3 - oy''_1 \\ ox''_2 - ox''_3 \\ oy''_2 - oy''_3 \\ ox''_3 - x''_8 \\ oy''_3 - y''_8 \end{cases} = \begin{bmatrix} \text{Trans}(O_1, V_1) & 0 & 0 & 0 & 0 & 0 \\ \text{Trans}(V_2, O_1) & \text{Trans}(O_2, V_4) & 0 & \text{Trans}(V_4, V_2) & 0 & 0 \\ \text{Trans}(O_1, V_3) & 0 & \text{Trans}(V_6, O_3) & 0 & 0 & \text{Trans}(V_3, V_6) \\ 0 & \text{Trans}(V_5, O_2) & \text{Trans}(O_3, V_7) & 0 & \text{Trans}(V_7, V_5) & 0 \\ 0 & 0 & \text{Trans}(V_8, O_3) & 0 & 0 & 0 \end{bmatrix} \begin{cases} \cos \Delta\phi_1 \\ \sin \Delta\phi_1 \\ \cos \Delta\phi_2 \\ \sin \Delta\phi_2 \\ \cos \Delta\phi_3 \\ \sin \Delta\phi_3 \\ \cos \Delta\phi_4 \\ \sin \Delta\phi_4 \\ \cos \Delta\phi_5 \\ \sin \Delta\phi_5 \\ \cos \Delta\phi_6 \\ \sin \Delta\phi_6 \end{cases} \quad (4.2)$$

4.1.3 回転角制御法

Fuzzy Node で接合される 2 つの部材の行列表現のときと同様に、回転角制御法を用いて式 (4.2) を解くことを試みたが、本論では解を求めることができなかった、今後の課題とする。

4.1.4 制約近似法

制約近似法を使用して式(4.2)を解く。

制約近似法の制約条件を与える。

Fuzzy Node の回転角が両端の部材の回転角で表現するので、

6つの任意の定数 $F_{4,1}, F_{4,2}, F_{5,1}, F_{5,2}, F_{6,1}, F_{6,2}$ を用いて以下のように表す。

$$\Delta\theta_4 = F_{4,1}\Delta\theta_1 + F_{4,2}\Delta\theta_2$$

$$\Delta\theta_5 = F_{5,2}\Delta\theta_2 + F_{5,3}\Delta\theta_3$$

$$\Delta\theta_6 = F_{6,1}\Delta\theta_1 + F_{6,3}\Delta\theta_3 \text{ の 3 つ の 条件 を 加 え る 。}$$

また、3.1.5 節と同様に変位を分割し、回転角を微小なものとして三角関数を近似する。

$$\sin \Delta\theta_n \approx \Delta\theta_n, \cos \Delta\theta_n \approx 1 \text{ と 置 換 す る 。}$$

このとき、右辺はこのように表される。

$$\begin{pmatrix} \cos \Delta\theta_1 \\ \sin \Delta\theta_1 \\ \cos \Delta\theta_2 \\ \sin \Delta\theta_2 \\ \cos \Delta\theta_3 \\ \sin \Delta\theta_3 \\ \cos \Delta\theta_4 \\ \sin \Delta\theta_4 \\ \cos \Delta\theta_5 \\ \sin \Delta\theta_5 \\ \cos \Delta\theta_6 \\ \sin \Delta\theta_6 \end{pmatrix} = \begin{pmatrix} 1 \\ \Delta\theta_1 \\ 1 \\ \Delta\theta_2 \\ 1 \\ \Delta\theta_3 \\ 1 \\ F_{4,1}\Delta\theta_1 + F_{4,2}\Delta\theta_2 \\ 1 \\ F_{5,2}\Delta\theta_2 + F_{5,3}\Delta\theta_3 \\ 1 \\ F_{6,1}\Delta\theta_1 + F_{6,3}\Delta\theta_3 \end{pmatrix}$$

変換行列を次のように変形できる。

$$\begin{pmatrix} x''_1 - ox''_1 \\ y''_1 - oy''_1 \\ ox''_1 - ox''_2 \\ oy''_1 - oy''_2 \\ ox''_2 - ox''_3 \\ oy''_2 - oy''_3 \\ ox''_3 - x''_8 \\ oy''_3 - y''_8 \end{pmatrix} = \begin{bmatrix} Trans(O_1, V_1) & 0 & 0 \\ Trans(V_2, O_1) + F_{4,1}Trans(V_4, V_2) & Trans(O_2, V_4) + F_{4,2}Trans(V_4, V_2) & 0 \\ Trans(O_1, V_3) + F_{6,1}Trans(V_3, V_6) & 0 & Trans(V_6, O_3) + F_{6,3}Trans(V_3, V_6) \\ 0 & Trans(V_5, O_2) + F_{5,2}Trans(V_7, V_5) & Trans(O_3, V_7) + F_{5,3}Trans(V_7, V_5) \\ 0 & 0 & Trans(V_8, O_3) \end{bmatrix} \begin{pmatrix} 1 \\ \Delta\theta_1 \\ 1 \\ \Delta\theta_2 \\ 1 \\ \Delta\theta_3 \\ 1 \\ \Delta\theta_3 \\ 1 \end{pmatrix}$$

部材 2 つのときと同様の手法で、すべての回転角を求められる。部材が 2 つのときの制約近似法では問題にはならなかったが、部材が増えることで Fuzzy Node のみによって接合されている部材(例：図 4.1.2.1 の部材 2)が生じ、その位置を決定することができなくなる。このような場合、部材 2 の位置を定めるために、さらに条件を加える必要がある。

このように部材の位置を一意に決定できないときは「変位を加える部材に最も近い Fuzzy Node 以外は全て長さが一定である」という条件を設ける。例として、図 4.1.2.1 では FN4 の長さを一定とし FN4 の回転角から V_4 の位置を算出し、さらにそこから部材 2 の位置を求める。このとき、FN4 の長さを一定としているので FN5 と FN6 の長さだけは変化する。そのため、FN5 と FN6 が最大長を超えているか判定することでその変位の可否を決めることができる。このような条件を設けることで、複数の部材においても制約近似法を適用できる可能性が生まれる。

4.2 解を反復計算で求める解法の応用

反復計算で解を求める解法は重心追従法しかないので、ここでは重心追従法の応用法を示す。

4.2.1 重心追従法の応用

繰り返し計算を行い、影響のある部材だけに計算を施し、その部材の位置を求めることが可能になっているため、重心法は現状、複数の部材への適用が最も有効だと考えている。

第3章で Fuzzy Node で処理しきれなかった変位を次の部材に反映させる方法を簡素に提示しているがこの方法だけでは複数の部材には対応できない。対応できない状況とその解決方法を示す。

第3章で示した重心追従法は図4.2.1.1のように部材に1つの変位を与えたときの部材の挙動の計算の方法しか求めている。同一部材に2つ以上の変位を与えることはないから一見すると問題ないように見えるが、反復計算をしているとき、場合によっては同一部材に2つ以上の変位が加わる状態になる可能性がある。以下、その状態になる条件を示す。このときのモデルは正方形部材で構成された平面モデルであるとする。

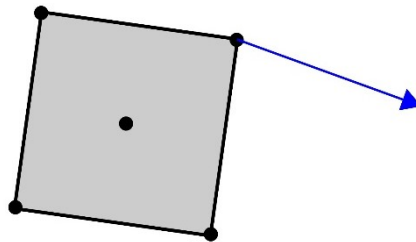


図 4.2.1.1 部材に1つの変位を与える様子

はじめに、1つの部材に変位を与える。(図4.2.1.2)

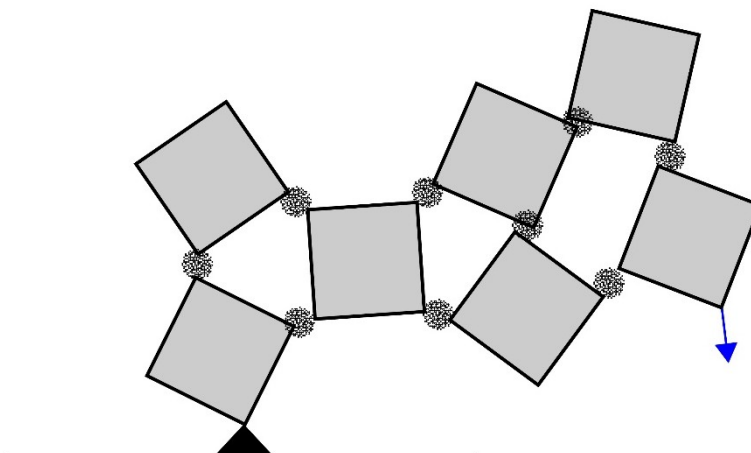


図 4.2.1.2 1つの部材に変位を与える

このように変位を与えた際、与えた変位が大きすぎると隣の部材にも影響を与えることがある。図 4.2.1.3 のように変位を直接加えてない部材が動いているのがわかる。

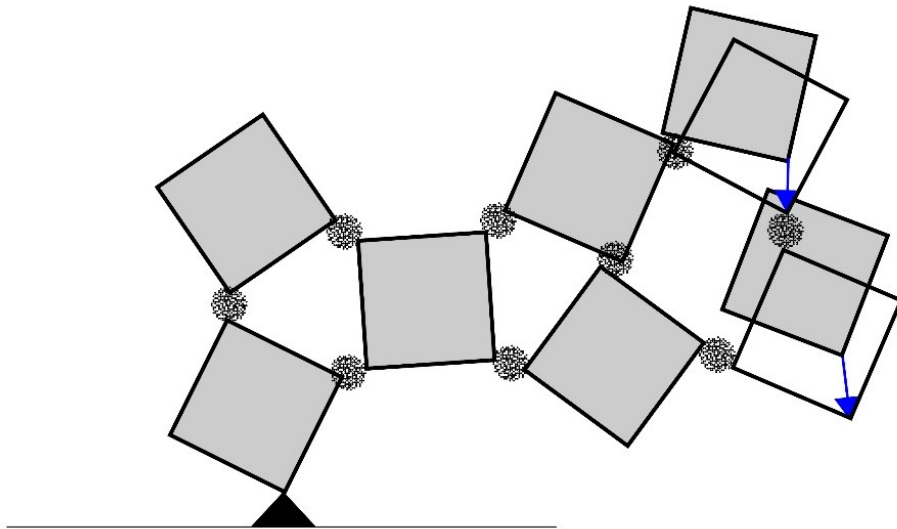


図 4.2.1.3 隣の部材に変位を伝播させる様子

この状態のまま、さらに変位を与え続けると以下の図のように変位が伝わり、同一部材に複数の変位が加わる状態となる（図 4.2.1.4）。

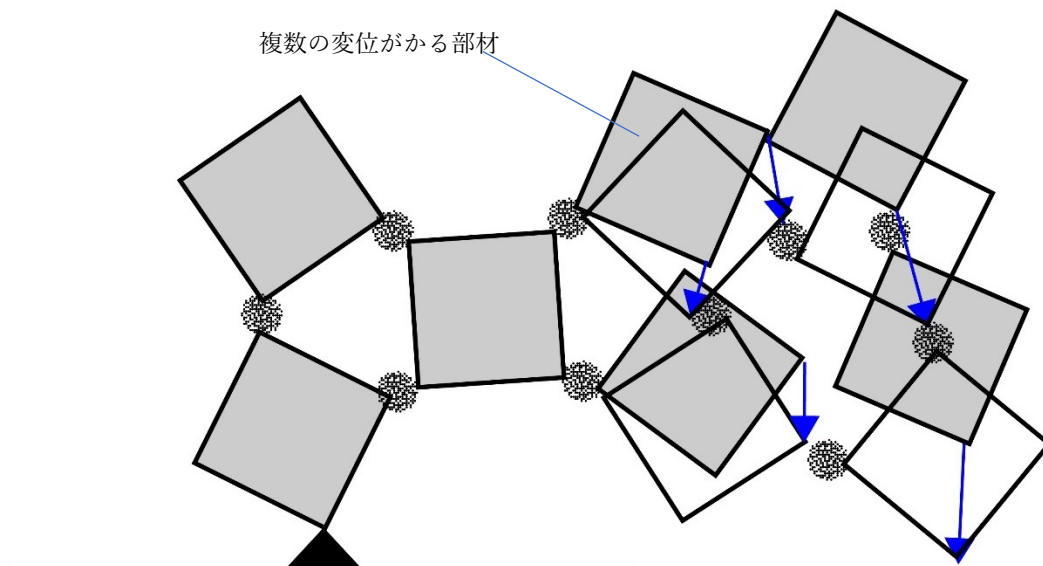


図 4.2.1.4 1つの部材に複数の変位が加わる様子

そこで、1つの部材に複数の変位を加えたときの重心追従法の適用法を提案する。

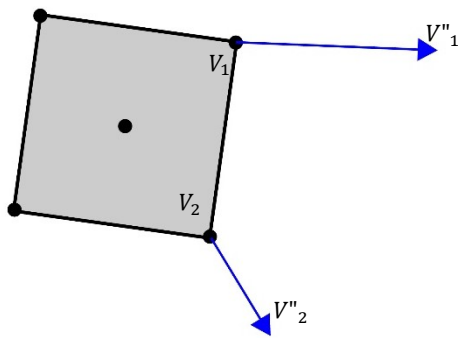


図 4.2.1.5 正方形の部材に2つの変位を加えた様子

上図 4.2.1.5 のように1つの部材に複数の変位が加わったときの解法を以下の2つの手順で行う。小片の形状を正方形で表示しているが、どのような形状でも解けるような方法になっている。

1. 変位を合成する。
2. 合成した変位を部材に与える。

1つ目の手順である変位の合成をする。変位の合成をするとき、図 4.2.1.5 では2つの変位の合成であるが、2つ以上の変位でも適用できるような解決策を示す。

この手順では合成変位の始点と合成変位のベクトルを求める。図 4.2.1.5 のように2点 $V_1(x_1, y_1)$ 、 $V_2(x_2, y_2)$ をとり、それぞれが $V''_1(x''_1, y''_1)$ 、 $V''_2(x''_2, y''_2)$ に引っ張られたときの合成変位を求めながら一般解へと拡張する。

はじめに、合成変位の始点を求める。

合成変位の始点は変位を与えられた点にそれぞれの変位を重さとして与えたときの重心である、と仮定する。

図 4.2.1.5 の変位の合成ベクトルの始点 D_s は図 4.2.1.6 のようになり、計算では次のように求まる。

$$D_s = \frac{L(V_1, V''_1)}{L(V_1, V''_1) + L(V_2, V''_2)} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \frac{L(V_2, V''_2)}{L(V_1, V''_1) + L(V_2, V''_2)} \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$$

次に一般式を記す。

節点 $V_k(x_k, y_k)$ が $V''_k(x''_k, y''_k)$ に引っ張られるとする。

変数の条件は $1 \leq k \leq n$ かつ $1 < n$

合成変位の始点 D_s は次のように求められる。

$$D_s = \frac{1}{\sum_{k=1}^n L(V_k, V''_k)} \sum_{k=1}^n L(V_k, V''_k) \begin{pmatrix} x_k \\ y_k \end{pmatrix} \quad (4.3)$$

と表せる。

次に合成変位のベクトルを求める。

求める変位の大きさは全ての変位ベクトルの長さの比をかけた和とする。

このとき、図 4.2.1.5 におけるベクトルの和は

$$\begin{aligned} \vec{D}_s &= \frac{L(V_1, V''_1)}{L(V_1, V''_1) + L(V_2, V''_2)} \vec{V_1 V''_1} + \frac{L(V_2, V''_2)}{L(V_1, V''_1) + L(V_2, V''_2)} \vec{V_2 V''_2} \\ \vec{D}_s &= \frac{L(V_1, V''_1)}{L(V_1, V''_1) + L(V_2, V''_2)} \begin{pmatrix} x''_1 - x_1 \\ y''_1 - y_1 \end{pmatrix} + \frac{L(V_2, V''_2)}{L(V_1, V''_1) + L(V_2, V''_2)} \begin{pmatrix} x''_2 - x_2 \\ y''_2 - y_2 \end{pmatrix} \end{aligned}$$

と表せる。

同様に以下、一般解を記す。

節点 $V_k(x_k, y_k)$ が $V''_k(x''_k, y''_k)$ に引っ張られるとする

変数の条件は $1 \leq k \leq n$ かつ $1 < n$

合成変位ベクトル \vec{D}_s は次のように求められる。

$$\vec{D}_s = \frac{1}{\sum_{k=1}^n L(V_k, V''_k)} \sum_{k=1}^n L(V_k, V''_k) \begin{pmatrix} x''_k - x_k \\ y''_k - y_k \end{pmatrix} \quad (4.4)$$

これより、手順 1 の変位の合成が示された。これは一般解も示しており 1 つ部材に対して 2 つ以上の変位が与えられたときは求められる。この一般解の式に $n = 1$ を代入すると変位を 1 つ与えたときの状況となり、第 3 章の 1 つの部材に変位を加えたときの重心追従法の式と一致するので、この一般式は $n = 1$ を満たす。変位の数によらず、先ほど定義した一般式を適用できることがわかった。

次に手順 2 の合成した変位を部材に作用させることを考える。図 4.2.1.5 において変位の合成を終えた状態を図 4.2.1.6 に示す。

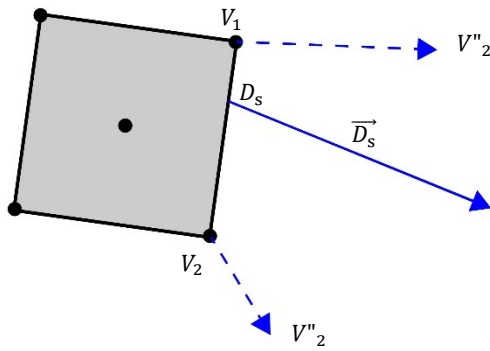


図 4.2.1.6 2つの変位を合成した様子

合成元となった変位はもう無視してよいので、図 4.8 のように部材 1 つに対して 1 つの変位として計算できる。3 章で重心追従法を行ったときは 1 つの部材の節点に 1 つの変位を与えていたが、変位を与える対象が節点でなくても同様の計算が可能なので、移動後の部材の位置を求めることができる。

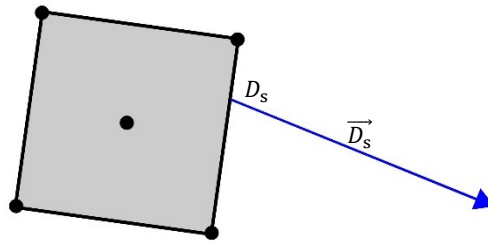


図 4.2.1.7 部材の任意の点に変位を与える様子

図 4.2.1.7 のような条件で重心追従法を適用すると図 4.2.1.8 のようになる。

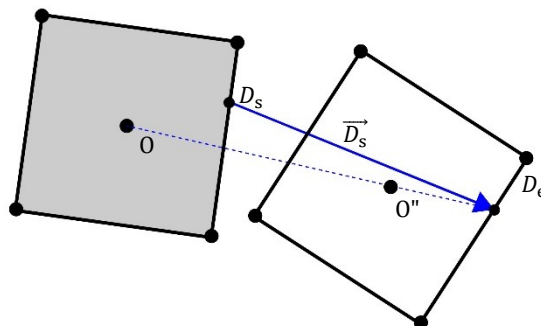


図 4.2.1.8 図 4.2.1.7 移動後の様子

合成変位の始点 D_s と終点 D_e が部際の相対座標系上で同じ値であることが図 42.1.8 からわかる。また、元の部材の重心 O と移動後の部材の重心 O'' 、合成変位ベクトルの終点 D_e が同じ直線上にあることが確認できる。

しかし、変位を合成することによって、 $D_s = O$ となる可能性が生じる。このとき、 $D_e = O''$ となり D_e, O, O'' の3点を通る直線を一意に決めることができない。そのため、合成した変位の始点が部材の重心と一致するときは、部材を合成変位の方に平行移動するものと定義する。

本節で説明した重心追従法を応用することで平面ならば、どのような形状の集積体であっても1つの部材に変位を与えた際の全体の変形を求めることが可能になる。第5章でプログラミングを用いた描画を行い。第6章で重心追従法の立体への応用について考察する。

第5章部材の挙動の可視化

5.1 制約近似法

本節では制約近似法の可視化を行う。第3章では Fuzzy Node の回転角を両端の部材の回転角を組み合わせて表現していたのでその組み合わせ方が変形後の形に与える影響を比較し、検証する。

制約近似法の全体形の変形を描画するにあたり「Rhinceros5 | ライノセララス | 3次元 CAD ソフトウェア version5 SR11 64-bit」、「Grasshopper (グラスホッパー) version August-27 2014 Build 0.9.0076」を windows10 のコンピュータで行った。図 5.1.1 のような正方形の小片で作成された二次元モデルを線材で置き換えたモデルの挙動を比較・検証する。それぞれを部材 1、FN2、部材 3 とする。

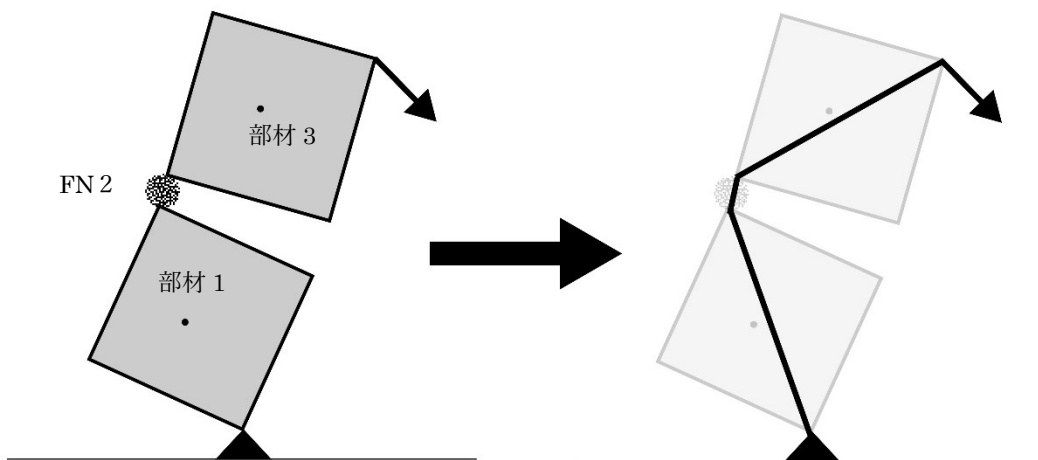


図 5.1.1 剛体モデルを線モデルに変換する

Grasshopper を用いて以下のようなプログラムを作成した (図 5.1.2)。

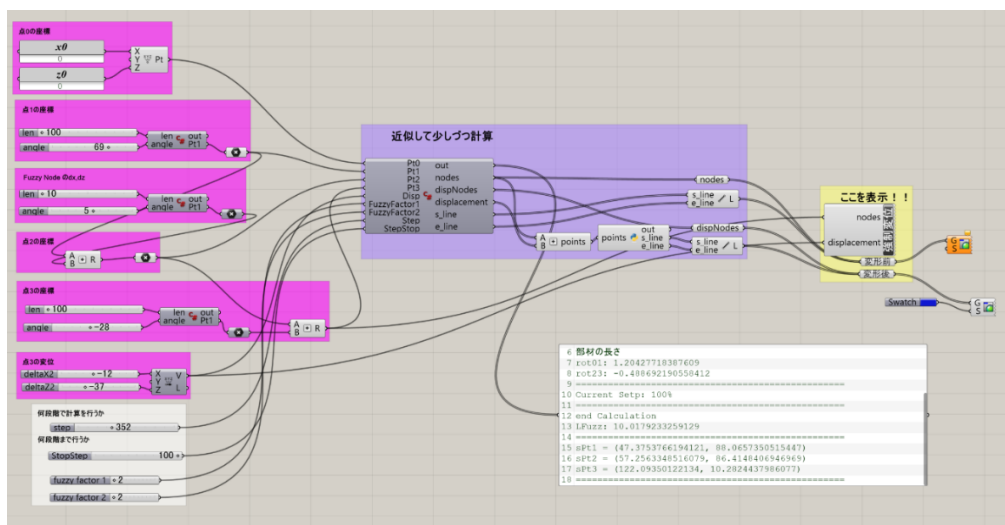


図 5.1.2 近似法のプログラムの全体図

図 5.1.3 では近似法のパラメータを設定できるようになっている。それぞれの部材の長さや初期の角度が入力できる。図 5.1.3 で例えると部材 1 は長さ 100 で X 軸となす角度は 107 度、FN 2 は長さ 10 で X 軸となす角度は 56 度、部材 3 は長さ 100 で X 軸となす角度は 32 度となっている。図内にある数字が示されているものを以後、スライダーと呼ぶ。

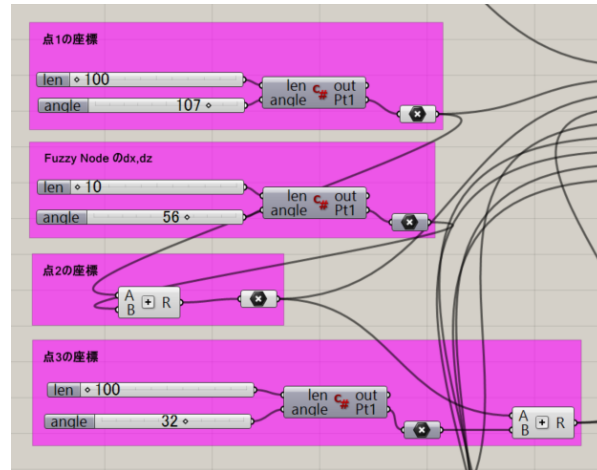


図 5.1.3 近似法のパラメータを制御するパネル

図 5.1.4 は部材に与える変位を調節する部分である。節点に与える変位を、x変位とz変位で入力する。図 5.1.3 で例えると、この場合では部材 2 に Z 方向に -35 の変位がかかっている。

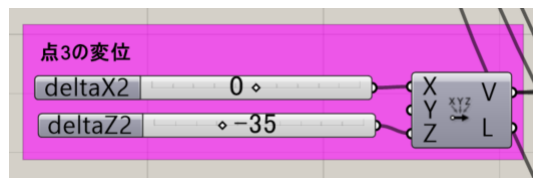


図 5.1.4 強制変位のパラメータを制御するパネル

図 5.1.5 では近似法の計算の詳細の設定と Fuzzy Node の回転角の制御が可能となっている。画像内一番上にある step と書いてあるスライダーは与えた変位を何分割して計算を行うかを表している。その下にある stop step というスライダーは増分計算を全体の何パーセントまで行うか制御するものである。残りの二つである fuzzy factor 1 と fuzzy factor 2 は Fuzzy Node の回転角を制御するパラメータとなっている。

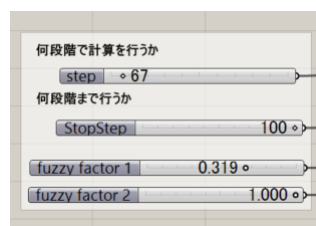
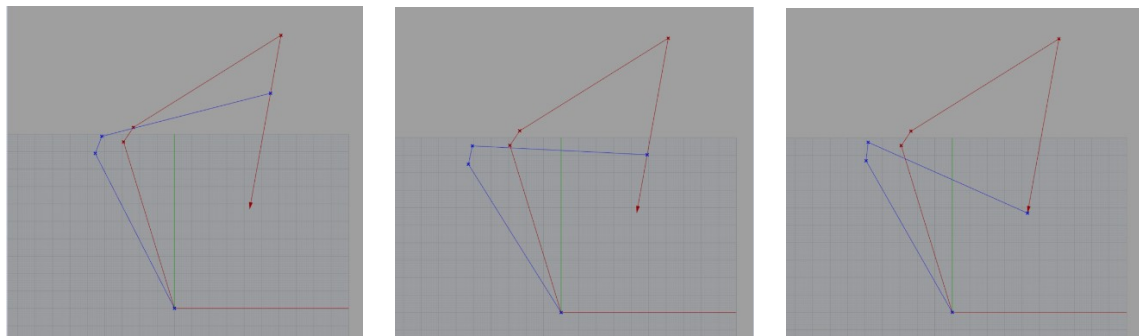


図 5.1.5 パラメータを制御するパネル

本論では部材の長さを 100、Fuzzy Node の長さを 10 として考察を行った。赤い線が変形前の形、青い線が変形後の形を示している。また矢印のついた赤い線は与えた変位となっている。

はじめに、パラメータ：step step の影響をみる。図 5.1.5 の step step と書いてあるスライダーで調節することが可能となっている。図 5.1.6 は step step を 35、70、100 と徐々に増加させたときのモデルの挙動を表示している。変位に対して約 1/3 ずつ移動しているのがわかる。



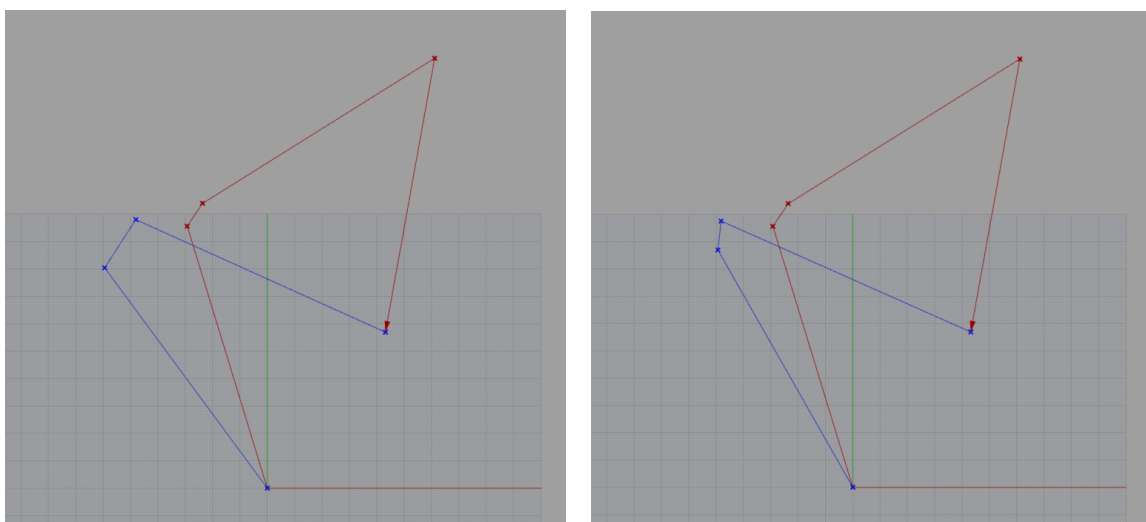
step step:35

step step:70

step step:100

図 5.1.6 近似法 step step の与える影響

次に増分計算の回数の違いの影響を考察する。図 5.1.5 の step と書いてあるスライダーで調節することが可能となっている。増分計算の回数を増やすと一回当たりの変位が小さくなるので近似したときの誤差が少なくなる。一方であまりにも多くの回数の計算を行うと計算に時間が必要となる。適切な増分回数や一回の計算での適切な変位を今後求めていく必要がある。次の図 5.1.7 は変位を 3 分割と 30 分割して計算したときの結果である。



step :3

step :30

図 5.1.7 近似法 step の与える影響

このように増分計算回数が3回と少ないと近似によって生じた誤差が大きく、それを吸収した Fuzzy Node の長さが大幅に変わってしまう。一方、増分計算回数を増やしていくが、30回を超えたあたりから差が見受けられなくなった。30回以上計算したものはほぼ変わらず、差がわからないのでここでは示さない。

次に Fuzzy Node の回転角を制御するときに用いる変数が移動後の全体形に及ぼす影響を検証する。式(3.11)の2つの定数 C_1 と C_3 を fuzzy facot1(ff1)と fuzzy factor2(ff2)とする。それぞれの値を変化させその結果を比較する。

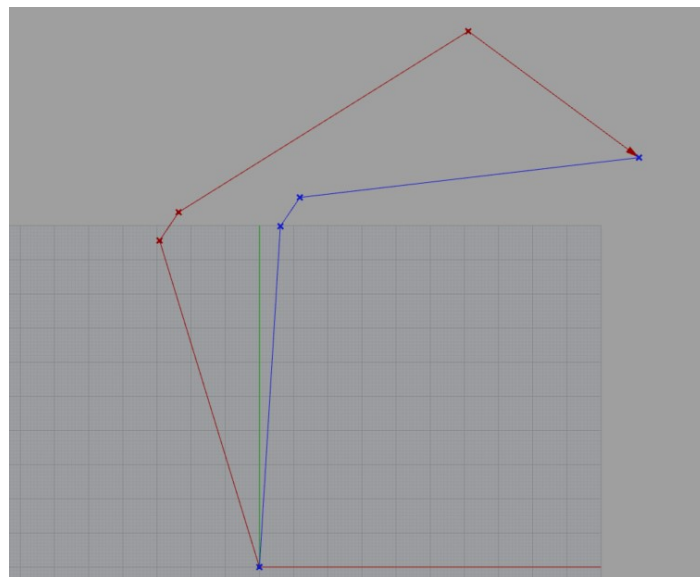


図 5.1.8 近似法 fuzzy factor1 = 0, fuzzy factor2 = 0

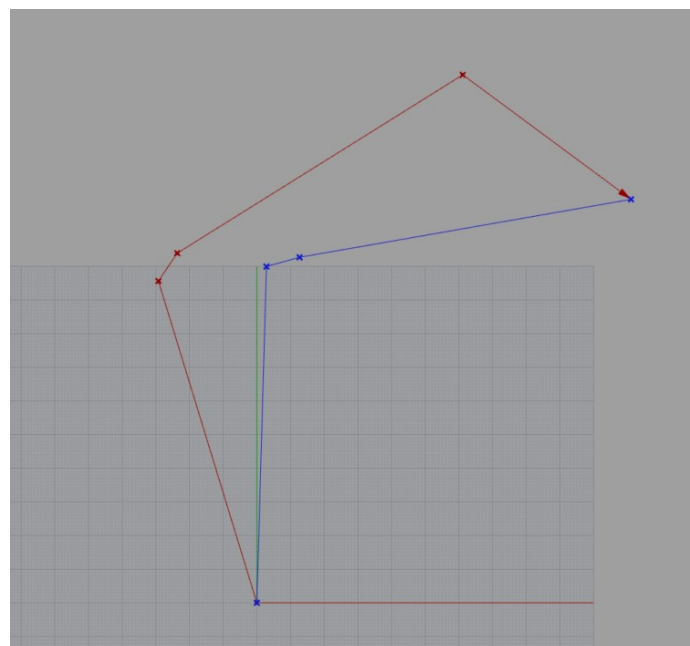


図 5.1.9 近似法 fuzzy factor1 = 1, fuzzy factor2 = 1

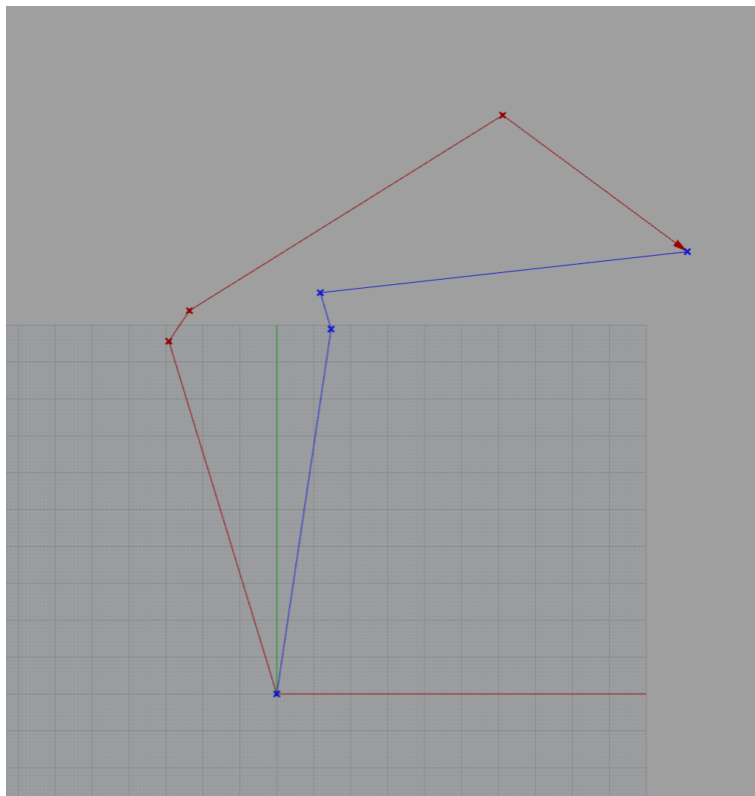


图 5.1.10 近似法 fuzzy factor1 = -1, fuzzy factor2 = -1

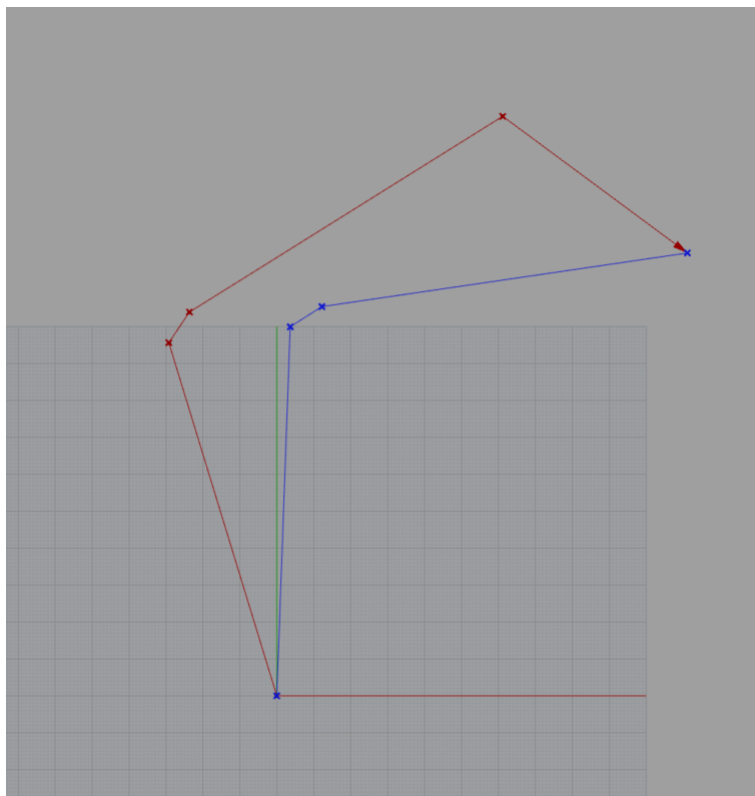


图 5.1.11 近似法 fuzzy factor1 = 1, fuzzy factor2 = 0

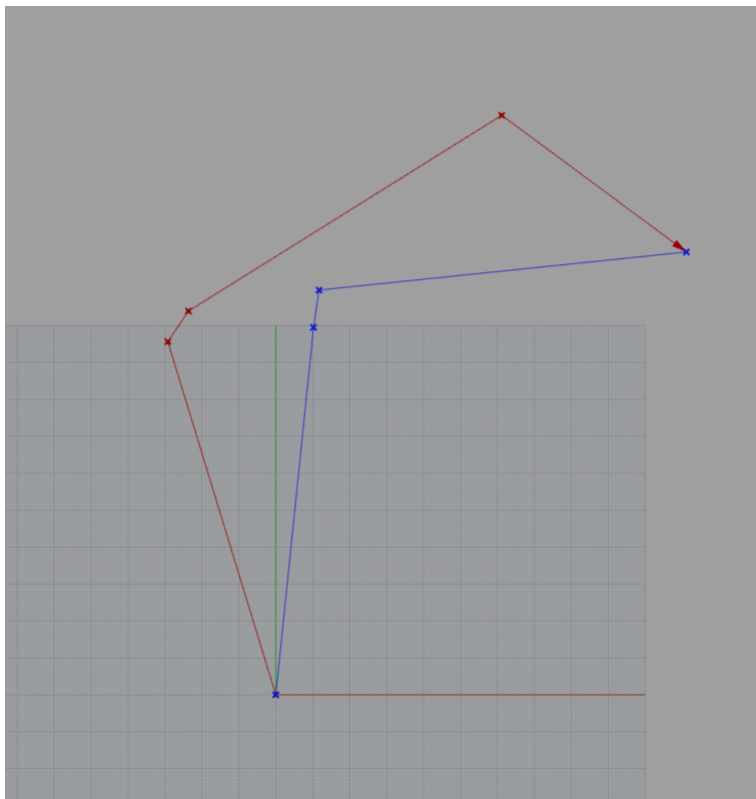


图 5.1.12 近似法 fuzzy factor1 = -1, fuzzy factor2 = 0

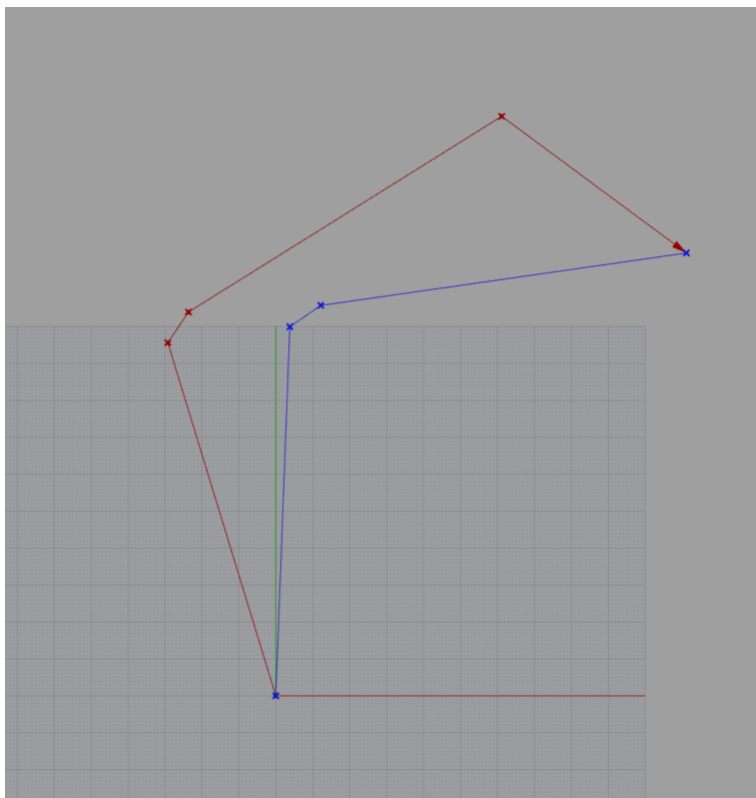


图 5.1.13 近似法 fuzzy factor1 = 0.5, fuzzy factor2 = 0.5

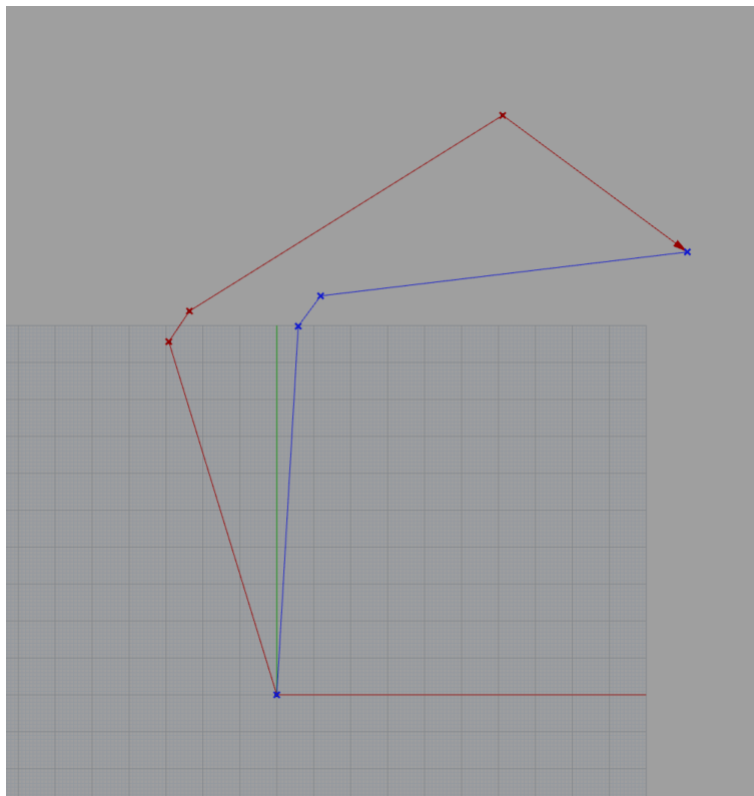


图 5.1.14 近似法 fuzzy factor1 = 0.5, fuzzy factor2 = -0.5

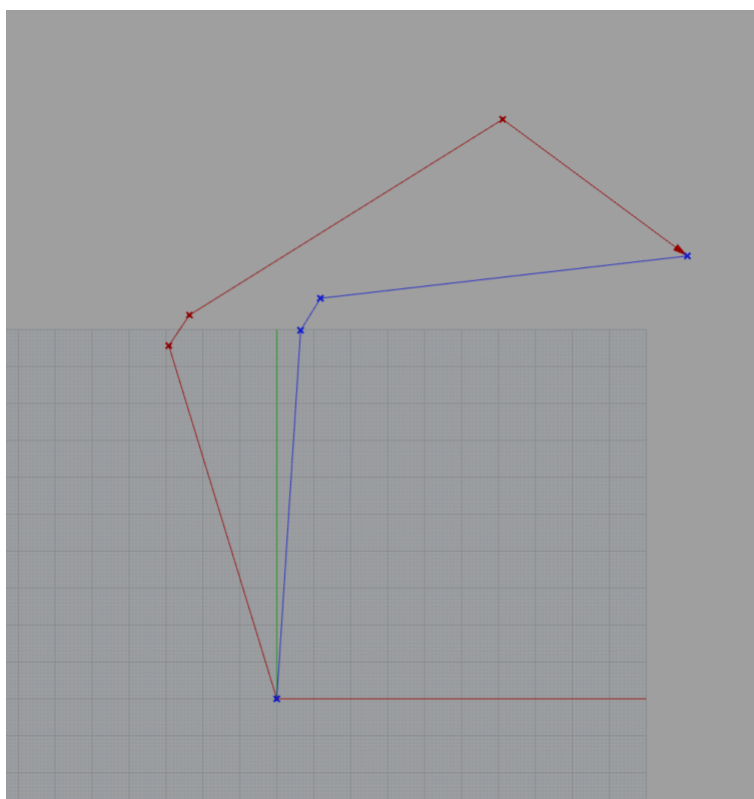


图 5.1.15 近似法 fuzzy factor1 = -0.5, fuzzy factor2 = 0.5

以上の図 5.1.8～図 5.1.15 は fuzzy factor の全体形への影響をわかりやすく図示したものである。

ff1 と ff2 がともに 0 のとき、Fuzzy Node の回転角は 0 なので同じ角度のままになると計算でき、図 5.1.8 で実際にそのようになっているのが確認できる。図 5.1.9 と図 5.1.10 では両方の係数が等しく前者では 1 後者では -1 となっている。係数の値が 1 となっている図 5.1.9 では、Fuzzy Node がほかのものと比較して一番回転しているのが確認でき、-1 となっている図 5.1.13 では、Fuzzy Node が反対方向に最も回転しているのが確認できる。

図 5.1.11 と図 5.1.12 のように片方の係数を 0 にすると複数部材のとき、どちらの係数を 0 にするのかなどの判定が増えるため、好ましくない。また、描画してみても変形後の形が他と大差ないので使用する必要がないことが確認できた。残りの図 5.1.13～図 5.1.15 では係数を 0.5 にして、和の平均や差の平均をとってみた。係数が両方とも負のものは図 5.1.10 のように部材とは逆のほうに回転してしまうのが見えた。回転角制御法で制限を設けたのと同様に、差の平均をとってみたものの特に幾何学的な意味もなく差がないので使用する必要がないと推測される。

複数の組み合わせの係数で比較してきた結果、両部材の平均をとる (ff1=ff2=0.5) 場合が複数部材に対応できる可能性が最も高く、最も適切であると本論では結論づける。

5.2 重心追従法

制約近似法のとおりコンピュータとソフトウェアを用いて描画を行った。重心追従法では1つの部材に複数の変位を与えたときの挙動を求められれば複数部材のときの挙動をも求められる。そのため、単体の部材に複数の変位を与えたときの挙動のみを示す。

図 5.2.1 に示されているのが重心追従法の全体の構成である。これは1つの正方形型の小片に変位を与えたときの挙動を重心追従法で求めるプログラムである。

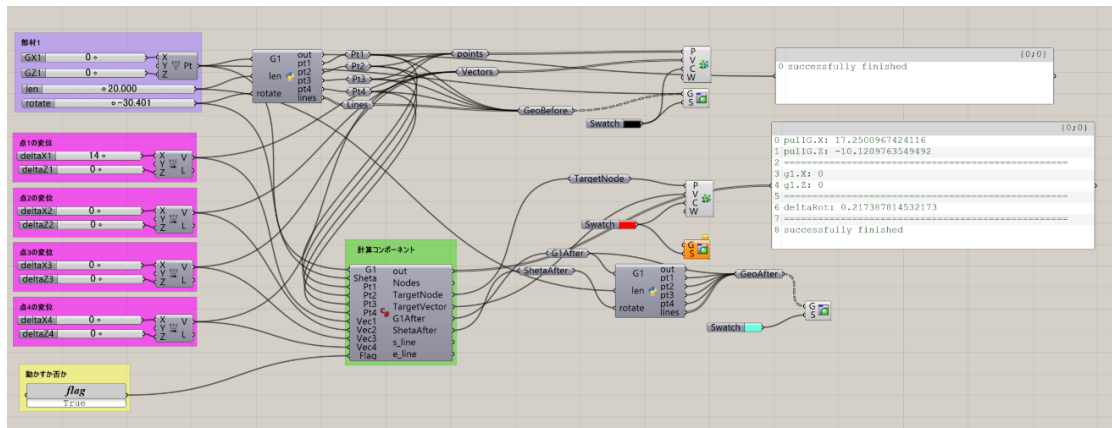


図 5.2.1 重心追従法のプログラムの全体構成

図 5.2.2 の部分で正方形部材の初期の状態を設定できるようになっている。GX1 と GX2 は部材の中心の座標、len が部材の中心から各節点までの距離そして rotate が部材の回転を表している。

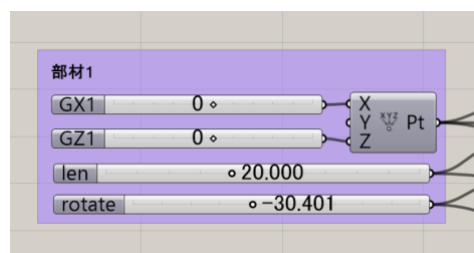


図 5.2.2 部材のパラメータ

図 5.2.3 の部分で部材の各節点に変位を与えられる。それぞれの頂点の水平と鉛直方向に独立して変位を入れることができる。スライダーに示してある点 1 から 4 が正方形部材の各頂点を示している。同じ頂点に対して「水平方向の変位のみ」などの水平と鉛直の片方だけに変位を与えることはできない構成となっている。また、両方の値がゼロのときは、変位がないものとして計算するようになっている。

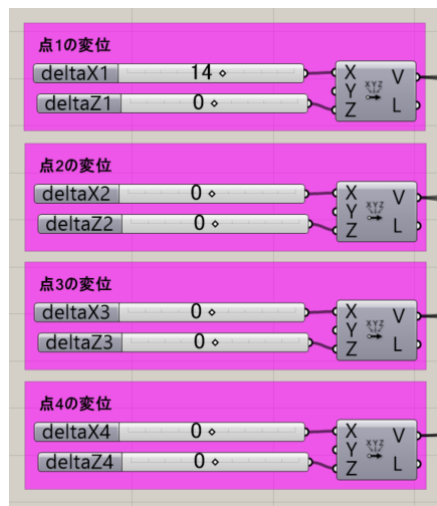


図 5.2.3 部材の変位パラメータ

図 5.2.4 の部分が重心追従法の計算を行っている。これは部材の初期の重心と頂点の座標、回転角そしてそれぞれの頂点に与えられた変位を引数として持っている。引数を上から順 (G1 から Vec4) に説明すると、それぞれは正方形部材の重心、部材の角度、それぞれの頂点の位置、それぞれの頂点にかかる変位を表している。第 4 章の重心追従法の複数部材への対応で求めた一般解を用いて計算するプログラムが書かれている。

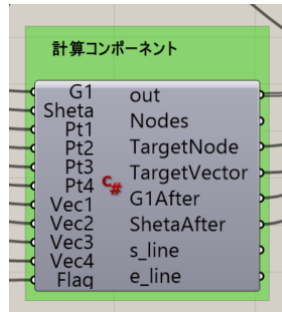


図 5.2.4 部材移動関数

1つの正方形部材に加える変位を1つ、2つ、3つと増やしていく。それぞれの個数の変位において、特徴的な変位を入力することでプログラムの有効性を確認する。黒い正方形が移動前の状態、黒の矢印が入力した変位を表している。また、赤い矢印が合成後の変位を表し、水色の正方形が移動後の状態を示している。

次の2つの図 5.2.5 と図 5.2.6 は部材に対して1つの変位を加えている。

1つ目は部材を平行移動させるため、パラメータを調節している。図 5.2.5 を見ると水平方向に平行移動していることが確認できる。2つ目の図 5.2.6 では一般的な移動を表している。移動前の重心と移動後の重心、変位先の点が同一直線状に存在することが確認できる。

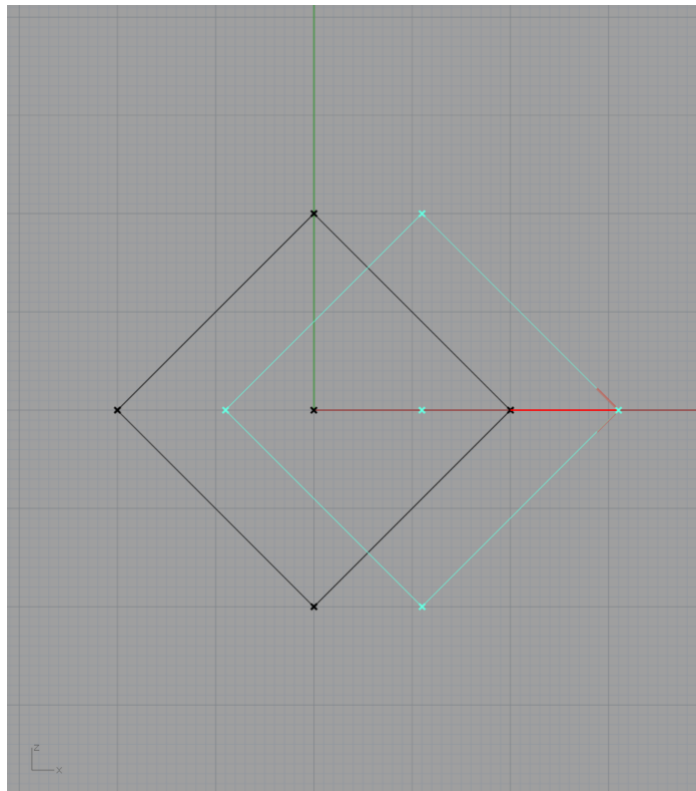


図 5.2.5 部材に変位を与えた様子 1

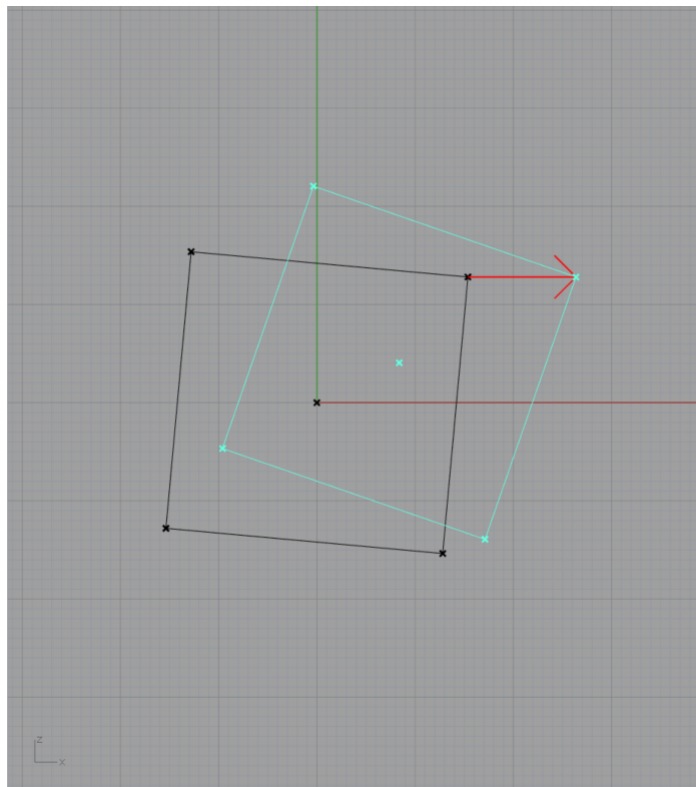


図 5.2.6 部材に変位を与えた様子 2

次の3つの図は部材に2つの変位を与えた際の部材の挙動を示している。

1つ目は合成した変位の始点が部材の重心一致するようにパラメータを入力している。対角線上の節点に同じ大きさの変位を与えることで合成変位を表す矢印の始点が元の部材の重心と一致していることが確認できる。第4章で述べたようにこの状態のときは部材を矢印の方向に平行移動させると定義しており、実際にそのような挙動となっていることが確認できる（図5.2.7）。

2つ目は同じ大きさの変位を逆方向に入力することで、合成した変位の大きさを0にしている。変位が0なので部材には何も影響がない。図5.2.8を見ると部材の状態が何も変わっていないことが確認できる。

3つ目の図5.2.9では一般的なものを表している。第4章の図4.9のような挙動が実際の計算で求められることが確認できた。

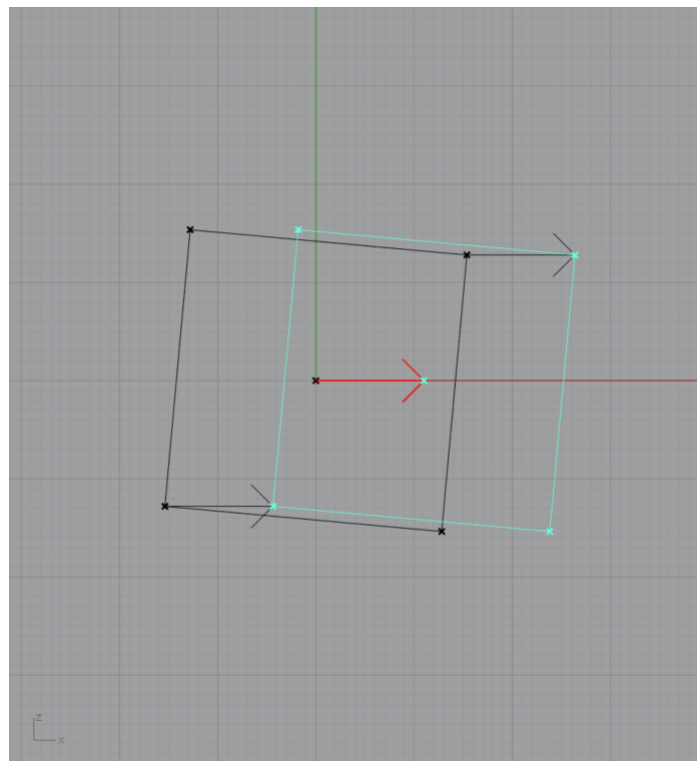


図5.2.7 2つの変位を与えた様子1

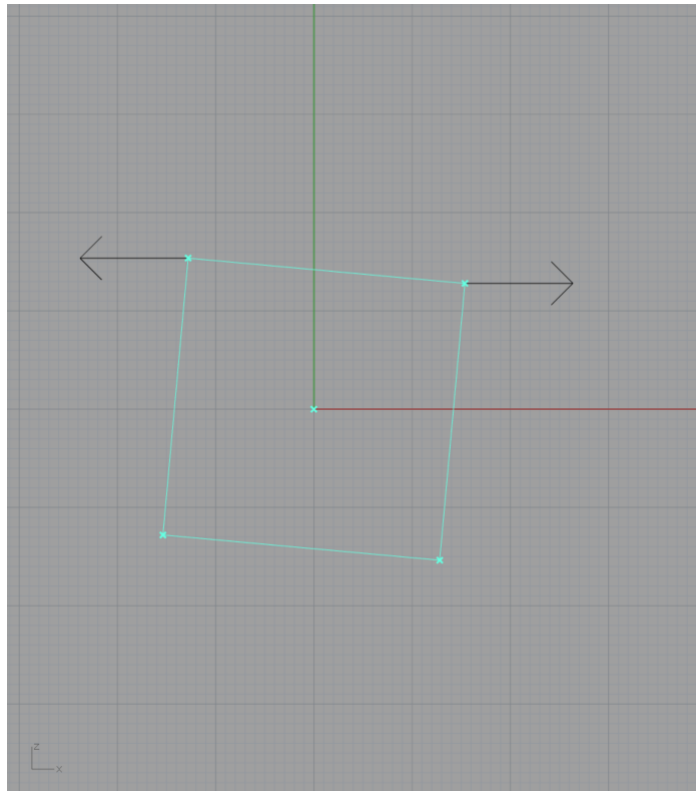


図 5.2.8 部材に 2 つの変位を与えた様子 2

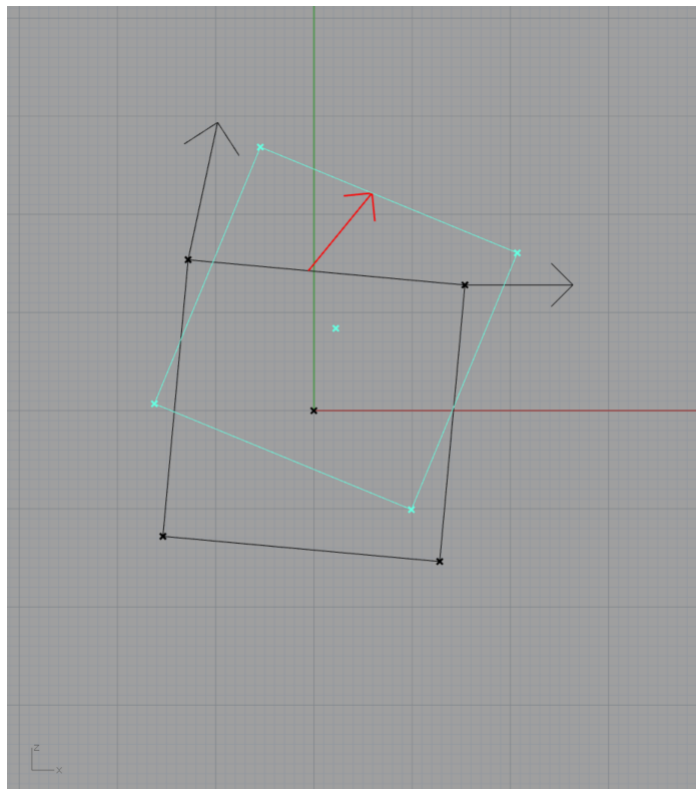


図 5.2.9 部材に 2 つの変位を与えた様子 3

次の2つの図は部材に3つの変位を与えた際の部材の挙動を示している。

1つ目は3つの変位の内2つが互いに打ち消し合うようにパラメータが設定されている。もう1つの変位は見やすいように水平なものにしている。図5.2.10の合成した変位を見ると、上下方向の変位がしっかり互いに打ち消しあっているため合成した変位には水平方向のベクトルしか含まれていないのがみてわかる。またその大きさも与えられた水平ベクトルよりも小さくなっており、上下に移動しようとする変位が左右の動きを制限していることが確認できる。

2つ目の図5.2.11は一般的なものを表している。このようにどのような変位が複数個あっても移動後の部材が求められることが確認できた。

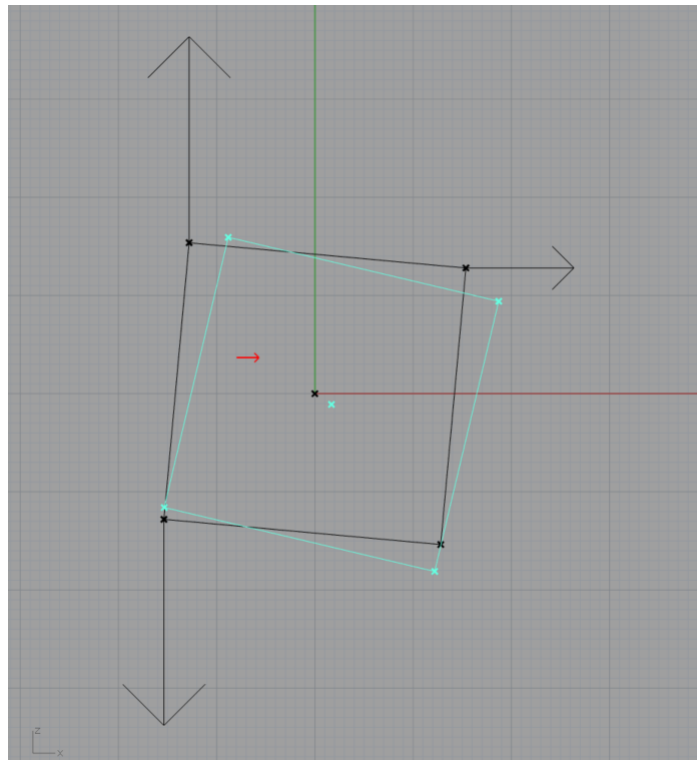


図 5.2.10 部材に3つの変位を与えた様子1

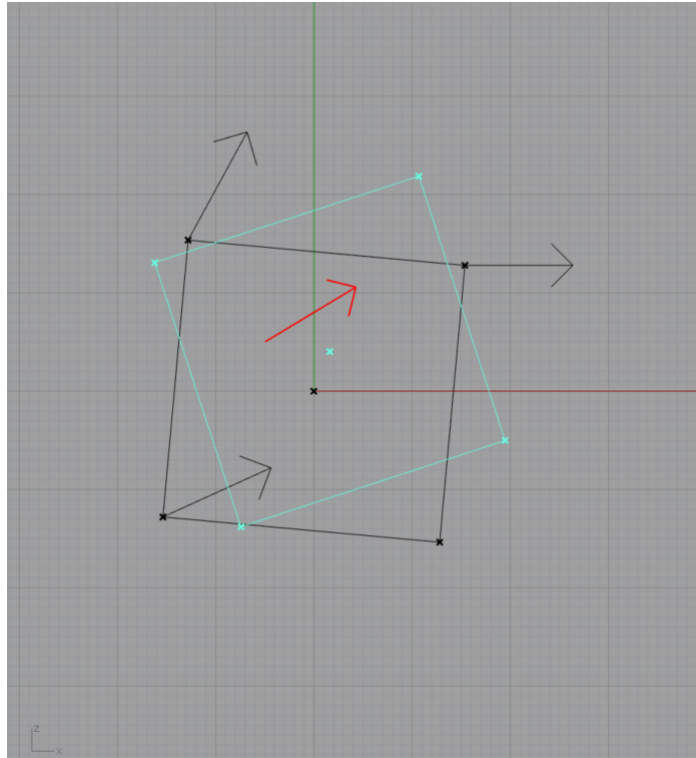


図 5.2.11 部材に 3 つの変位を与えた様子 2

5.3 複数の Fuzzy Node で接合された集積体の可視化

5.3.1 ソフトウェアの構造

複数の Fuzzy Node で接合された集積体の描画にあたり重心追従法を適用した。ソフトウェア開発に使用したコンピュータは第 2 節で述べたものと同じである。このソフトウェアは C# で書かれており、「Visual Studio 2017」を使用してコンパイルされている。ここでは、ソフトウェア開発で作成したコードの構造について説明する。

データの保存用に以下の 6 つのクラスを作成した。上から順番にそれぞれのクラスの構成について説明する。各クラスには値を保存する変数とそのクラスに関連する計算を行う関数が格納されている。関数については詳しくは本論では触れないが付録にすべてのソースコードを記載する。

1 PointD

このクラスは座標を保存している。以下に記されている情報を変数として格納しており、立体でも使用するために Z 座標を保存している。また、座標の移動や指定した点を中心に回転するための関数も用意されている。

型名 変数名 変数の役割の説明 という順序で記載する。ほかのクラスでも同様に示す。

double X 空間座標の X 座標を格納する変数

double Y 空間座標の Y 座標を格納する変数

double Z 空間座標の Z 座標を格納する変数

double Len 空間座標の原点との距離を格納する変数

2 Node

このクラスは PointD に情報を付加し、節点情報を保存している。以下の情報を格納する。

PointD Coord 空間座標を格納する変数

int ConnectedPartId この節点が接合している部材 id を格納する変数

int ConnectedNodeId この節点が接合している部材の節点 id を格納する変数

double R 部材の重心との距離を格納する変数

double A 部材の重心を原点としてこの点が水平線とのなす角を格納する変数

NodeState NState 節点が既に接合されているか格納する変数

ObjectState OState 節点の選択の有無を格納する変数

3 Part2D

このクラスは各部材の情報を保存している。重心の座標や自身の回転、節点の情報が以下に記されている形式で保存されている。節点の情報は連想配列で保持され、そのキーが節点特有の id となっている。変位を与えたり、接合を行ったりする対象を示すときにこの id を使用する。

Node Center 部材の重心座標を格納する変数

double Rotation 部材の回転量を格納する変数

ObjectState OState 節点の選択の有無を格納する変数

Dictionary<int, Node> PointFolder 部材の節点情報を連想配列にて格納する変数

4 MassObject

このクラスは集積体を保存している。集積体を構成する基本の部材情報や現在配置されている部材の情報、Fuzzy Node の最大長などの情報が格納されている。配置されている部材は連想配列で保存されており、そのキーが部材固有の id となっている。

Part2D MasterPiece 部材の基本形状を格納する変数

Dictionary<int, Part2D > Parts すべての部材を格納する変数

int FuzzyLen Fuzzy Node の最大長を格納する変数

DateTime SavedDate 最後に保存した時間を格納する変数

5 Force

このクラスは1つの部材にかかる変位を保存している。複数の変位を格納できるように変位は連想配列で保持され、そのキーが変位のかかる節点の id になっている。

Dictionary<int, PointD> Vectors 部材の節点情報を連想配列にて格納する変数

6 Forces

このクラスは全部材に対する変位を保存している。変位を与えている各部材それぞれに対して Force をそれぞれ作成し、作成したものを連想配列にて格納している。この連想配列のキーはここでは部材固有の id となっている。

Dictionary<int, Force> すべての変位を連想配列にて格納する変数

int counter 計算回数を格納する変数

5.3.2 ソフトウェアの構築

前節で述べたクラスをどのように構築してソフトウェアを作成したのか説明を行う。作成したクラスの関係性を整理する。

集積体を構成するクラスの関係性を以下のように表せる。図 5.3.2.1 は k 個の節点を持つ小片が n 個存在する集積体を表している。MassObject クラスの中には部材の基本形となるもの (Masterpiece) や Fuzzy Node の最大長 (uzzyLen) が格納されている。また、Parts の中には n 個の part が含まれており、それぞれの part には k 個の節点が格納されているのが確認できる。それぞれの部材は Parts2D-○と番号で識別できるようになっている。この○には 1 から n の固有の整数が割り当てられ一意なものとなっている。また、それぞれの Part2D-○内にはそれぞれの部材の節点情報を表す Node と命名したクラスが格納されている。この Node にも Part2D 同様に 1 から k までの整数が識別番号として割り当てられている。このように設定すると、Part2D と Node に割り当てられた番号を組み合わせることですべての部材 (Part2D) と節点 (Node) を区別することが可能になる。

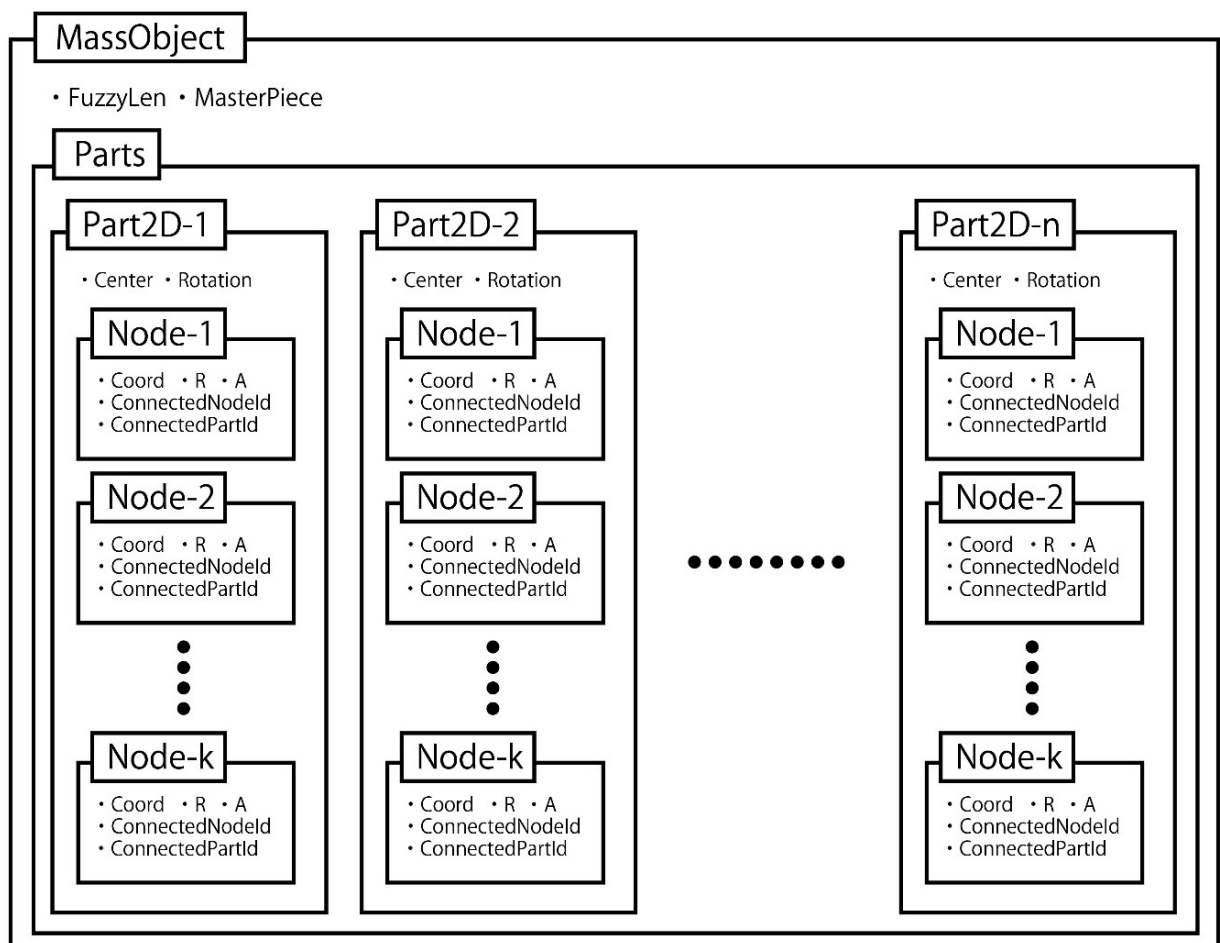


図 5.3.2.1 集積体のクラスの関係性

次は変位のデータ構成について詳しく見てみる。

変位を構成するクラスは以下の図 5.3.2.2 のように表せる。この Forces というクラスも先ほどと同様に二重の入れ子の構造になっている。それぞれの変位が Force-○と番号で識別できるようになっている。また、それぞれの Force-○クラス内には変位 Vector-○が格納されており、これも番号で識別できるようになっている。Force の番号がそれぞれの部材を示しており、その中にある Vector の番号がそれぞれの節点の変位を示している。各部材の節点に必ずしも変位が加わっているとは限らないので、先ほどの集積体のクラスとは異なり、識別番号が連番にはならない。また、部材ごとに作用する変位の数も異なるので Force 内の変位の総数が不均一になっていることがわかる。

例として図 5.3.2.2 の Force-1 は図 5.3.2.1 の Part2D-1 にかかる変位を示しており、Force-1 内にある Vector-1、Vector-3、Vector-6 は Node-1 と Node-3 と Node-6 に変位が与えられていることを示し、その変位の大きさを保持している。

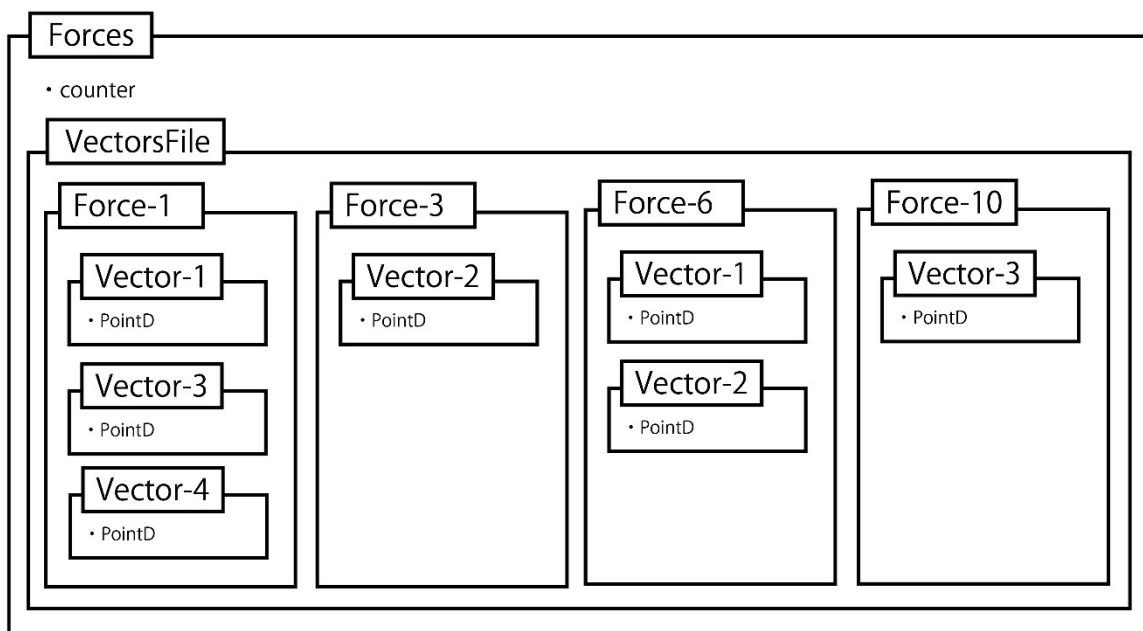


図 5.3.2.2 変位のクラスの関係性

ここからは重心追従法の構築について説明する。

以下は重心追従法の手順を簡潔に示したものである。

1. 特定の部材の節点をマウスで移動させ位置を調節させる。このとき、マウスで調節した距離が節点に与える強制変位となる。
2. 集積体のクラスと変位のクラスを掛け合わせ、すべての部材に関連する変位を作用させ、仮に変位を加えたときの部材のそれぞれの位置を計算する

3. 手順2で移動した部材に関してのみ接合している節点同士の接合距離が Fuzzy Node の最大長を超えてないか判定する。
4. Fuzzy Node の長さが規定値を超えるものがあるとき、その超過分を接合される節点に与える変位と定義し、Forces のクラスにその超過した分を換算した変位を新しく加えて、手順2に戻る。

手順1から順に操作を行い、手順3で Fuzzy Node の最大長を超えるものがなければ手順4には進まない。一方、最大長を超えるものがあれば手順4に進み、このサイクルを繰り返し行い部材の位置を計算する。

このサイクルを繰り返していると、計算で求めることができないような状況に陥る可能性がある。再計算回数に上限を設けておき、上限に達してしまったときは手順1で与えた変位が不適切であると判定して手順1を施す前の位置に部材を戻す。図 5.3.2.1 にある counter という変数が現在何回目の再計算をしているのか記録している。

5.3.3 ソフトウェアの活用

開発したソフトウェアの詳細について記す。

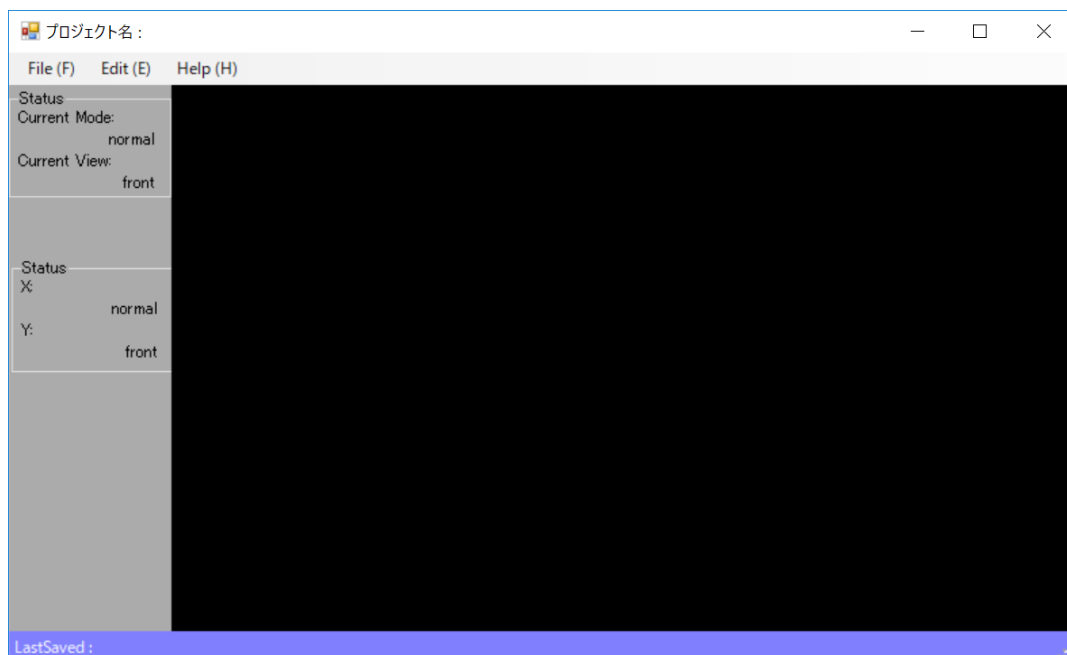


図 5.3.3.1 ソフトウェアのメイン画面

左側の灰色の帯が画面の情報を表しており、右側の黒い部分が作業画面となっている。黒い部分をマウスで右クリックをすると図 5.3.3.1 のようなクリックメニューが出てくる。

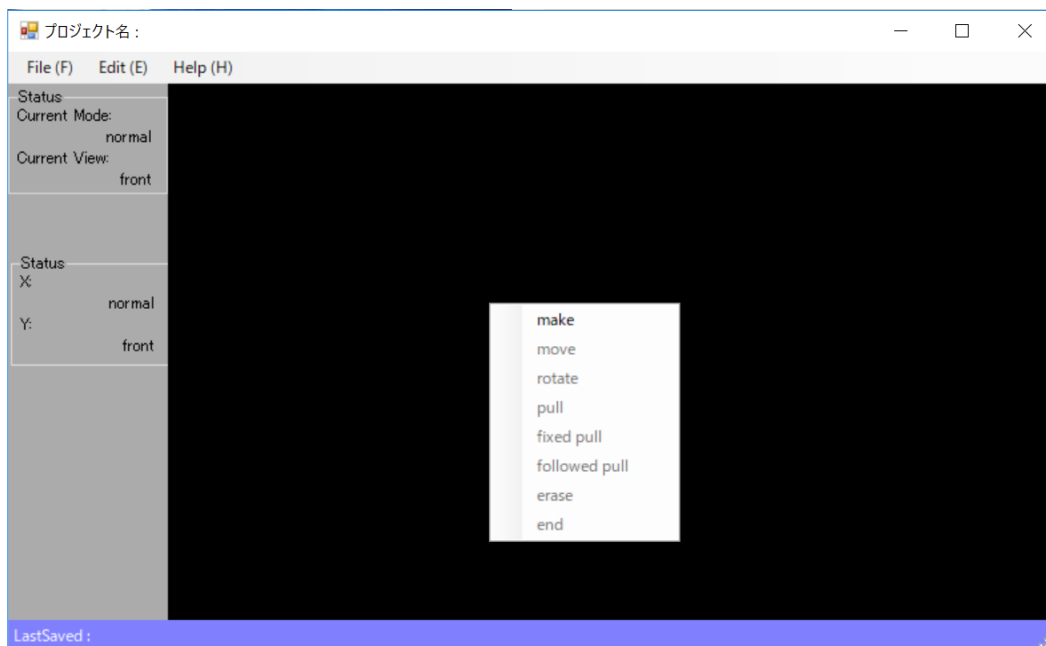


図 5.3.3.2 クリックメニューの表示

メニューには 8 つの操作が用意されている。

1. Make

部材を作成するコマンドである。クリックした位置を重心として部材が追加される。

2. Move

部材を移動するコマンドである。重心をマウスで調節できる。

3. Rotate

部材を回転するコマンドである。重心を中心に節点を回転させる。

4. Pull

1 つの部材に重心追従法を適用するコマンドである。節点をドラッグすることで変位を与えられる。

5. Fixed Pull

Pull と同じであるが、この操作を施す部材が他の部材と接合されているとき、その接合距離が Fuzzy Node の最大長を超えない範囲で部材に重心追従法を適用する。

6. Followed Pull

複数部材の重心追従法である。変位の与え方は Pull と同じになっている。

7. Erase

選択された部償を削除するコマンドである。

8. End

操作を終了するコマンドである。

これらの操作を活用し正方形の小片で設計したものが図 5.3.3.3 である。

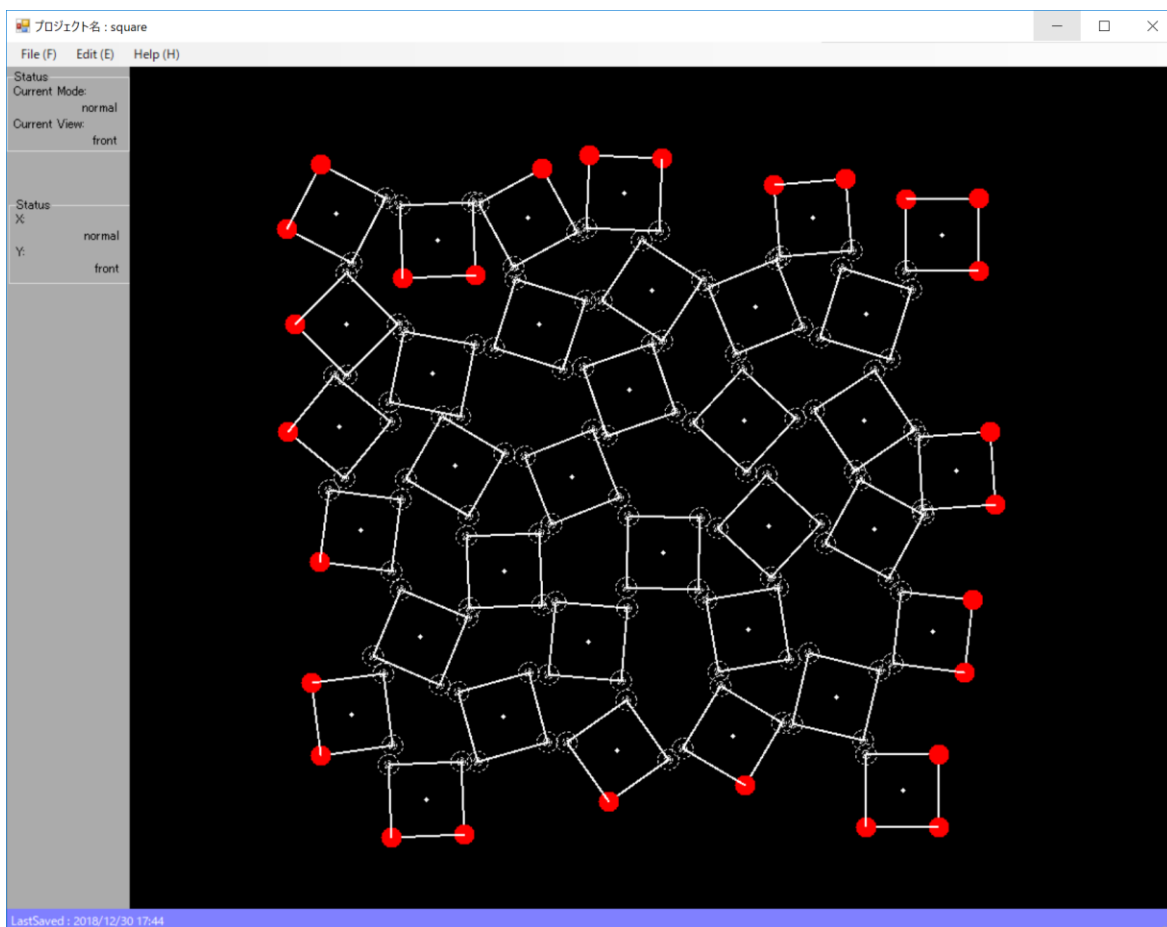


図 5.3.3.3 ソフトウェアの様子 1

上の図は正方形で構成したものである。接合可能な部材の節点を点には赤い円が描かれている。多くの空隙を作りながらもほぼすべての節点を接合することが可能となっているのがわかる。

このソフトウェアは正方形だけでなく、ほかの正多角形や自由な多角形にも対応している。次のように自分で形を作ることも可能となっている。

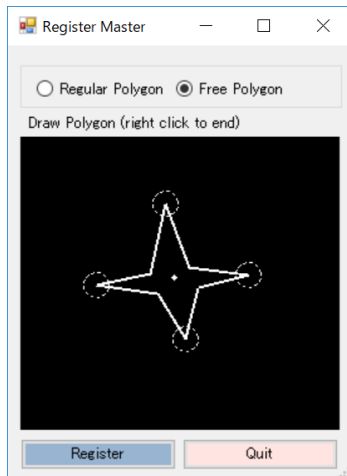


図 5.3.3.4 自由多角形の作成

各々作成した多角形でも同じ操作が可能となっている。図 5.3.3.4 の部材でソフトウェアを使用すると次のようなものを作成することが可能である。

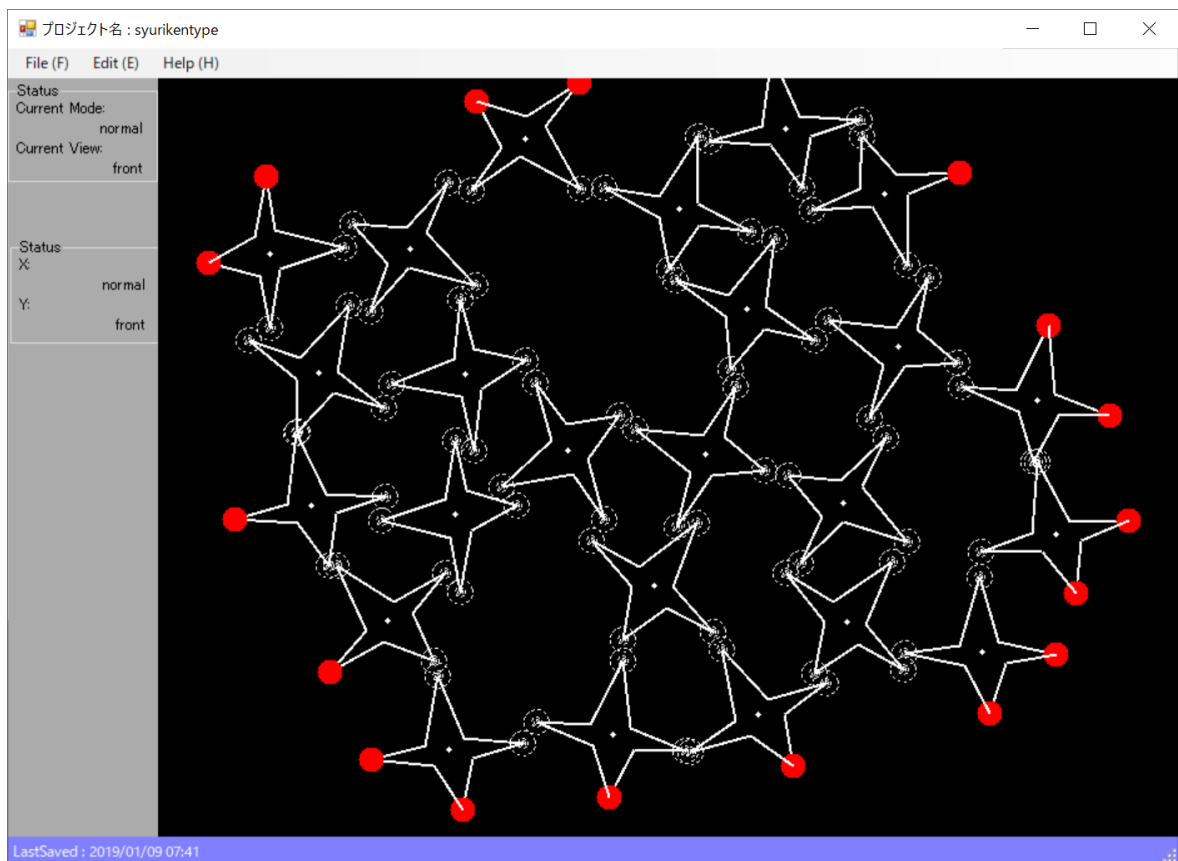


図 5.3.3.5 ソフトウェアの様子2

このソフトウェアの使用することで、多数の小片を並べる場合の挙動に異常がないこと、および、全体形的设计自由度が増すことが確認できた。重心追従法の有効性を示すことができた。

第6章 立体への拡張

6.1 重心追従法の立体への対応

重視追従法の立体への拡張を考察する。平面でのそれぞれの手順をどのように立体に適用させるか順に示す。

平面のときに示した手順を簡潔に以下に記載する。

1. 特定の小片の節点に強制変位を与える。
2. 変位を与えた小片の移動すべき位置を重心追従法で求める。
3. 手順2で移動した小片と接合する全ての Fuzzy Node の長さを判定する。
4. Fuzzy Node の長さが規定値を超えるものがあるとき、その超過分を接合される節点に与える変位と定義する。

それぞれの手順において、立体への適用方法を考える。

手順1は二次元であった強制変位を三次元に拡張することで解決できる。手順3は2点間の距離を空間座標で求めることで解決できる。また、手順4で超過分を変位に変えるのも三次元ベクトルを使用することで、手順1と同様に拡張できる。このように手順1と3、4は容易に拡張できるので、次は手順2の立体への拡張について考察する。

手順2をさらに以下の2つの工程に分け、それぞれを立体に拡張することを考える。

2. 1 変位を合成する。
2. 2 重心追従法で小片の移動後の位置を求める。

まず、2. 1の変位の合成の立体への適用を示す。次のように式(4.3)と式(4.4)に引数を1つ追加することで、立体にも適用できる。

$$D_s = \frac{1}{\sum_{k=1}^n L(V_k, V''_k)} \sum_{k=1}^n L(V_k, V''_k) \begin{pmatrix} x_k \\ y_k \\ z_k \end{pmatrix}$$
$$\vec{D}_s = \frac{1}{\sum_{k=1}^n L(V_k, V''_k)} \sum_{k=1}^n L(V_k, V''_k) \begin{pmatrix} x''_k - x_k \\ y''_k - y_k \\ z''_k - z_k \end{pmatrix}$$

次に、手順2. 2の立体の適用法を考える。立体に重心追従法を適用するとき、平面のときに定めた条件のままでは小片の位置を一意に決定することができない。そこで、以下の条件を加える。

条件：合成変位と小片の重心のなす平面の法線ベクトルを軸に回転する。

議論を簡潔にするため、立方体型の小片の1つの節点に強制変位を与え、移動させることを考える(図 6.1.1)。

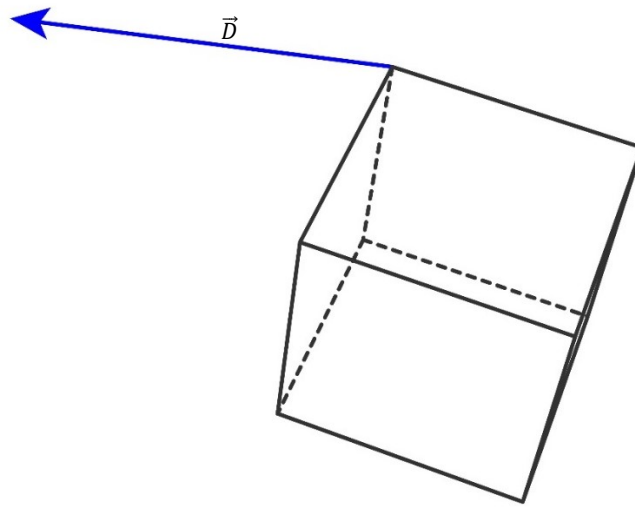


図 6.1.1 立体の小片に変位を与える様子

このとき、先ほど設けた条件から移動後の小片の位置は、変位ベクトル \vec{D} と重心 G のなす平面の法線ベクトル \vec{n} を軸に小片を $\Delta\theta$ 回転し、重心を ΔG 移動させることで求まる(図 6.1.2)。

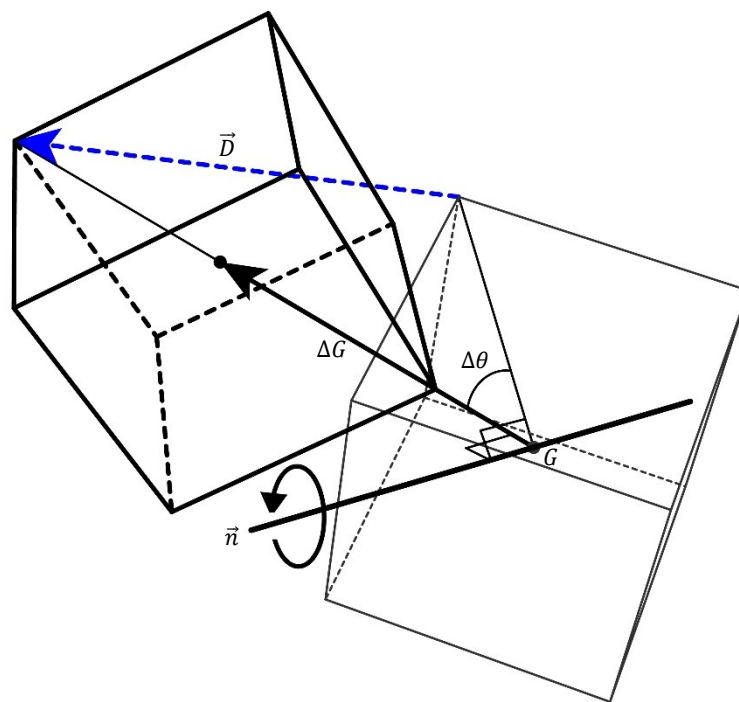


図 6.1.2 変位と小片の重心のなす平面と小片の様子

これは立体の回転と移動なので2章で扱った式(2.12)から、移動後のすべての頂点の座標を求めることができる。

6.2 Fuzzy Node で直鎖状に接合された部材の挙動の可視化

平面のときと同様の反復計算により、複数の小片を連動的に操作可能なソフトウェアを作成し、その解法の有効性を確認した。図 6.2.1 は Fuzzy Node で接合された3つの部材の○のついた節点に白い矢印の変位を与えたとき、変位を直接与えてない部材が連動して動いている様子である。

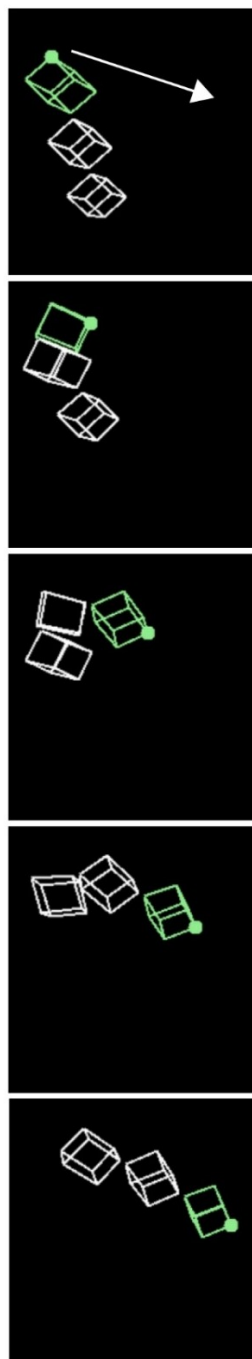


図 6.2.1 立体の重心追従法のソフトウェアの様子

第7章 結

7.1 本論文での成果

Fuzzy Node で接合される小片集積体の機構について特定の条件を与えることで、数値解析による解を得る方法をできた。

本論文での成果を以下にまとめる

1. Fuzzy Node を持たないモデル

Fuzzy Node を持たない環状モデルにおいて、その変換行列が解けることが示された。
(第4章・第1節)

2. 部材の位置を一度の計算で求める解法

・回転角制限法

Fuzzy Node で接合された2部材に変位を与えたときの行列を変形すると次の式(7.1)を得られる。式(7.1)に条件(7.2)を与えることで三角関数の引数に同じ変数が表れ、解が求まることが分かった。

$$C_1 = C_3 \cos(\Delta\theta_2 - C_3) + C_4 \cos\left(\frac{\Delta\theta_1}{2} - \frac{\Delta\theta_3}{2} + C_5\right) \quad (7.1)$$

$$\Delta\theta_2 = \Delta\theta_1/2 - \Delta\theta_3/2 \quad (7.2)$$

・制約近似法

Fuzzy Node の回転角 θ_f を両側の部材の回転角 θ_m, θ_n を用いて式(7.3)のように定義すると、Fuzzy Node で接合された2部材モデルや環状モデルに変位を与えたときの行列が三角関数を含まない連立一次方程式となるので解が求まることが分かった。

$$\Delta\theta_f = C_k \Delta\theta_m + C_l \Delta\theta_n \quad (C_k, C_l \text{は任意の定数}) \quad (7.3)$$

・その他の解法

Fuzzy Node で接合された2つの部材の行列表現において、No Fuzzy 仮定法、部材変形法では条件が足りず、解は求まらなかったが、変数同士の関係式だけを求めることができた。また、Fuzzy 伸縮制限法では、Fuzzy Node の長さが全体形の変形に与える影響を描画し比較検討することができた。

3. 部材の位置を反復計算で求める方法 (重心追従法)

重心追従法は平面、立体に関係なく解を求められることを示せた。部材に1つの変位を与えたときは重心の移動が最も少なくなるように、また、複数の変位が同じ部材にかかるときは式(7.4)で1つに変位を合成してから部材を移動させると定義した。

$$G = \frac{1}{\sum_{m=1}^n L(kV_m, kV''_m)} \sum_{m=1}^n L(kV_m, kV''_m) \begin{pmatrix} kX_m \\ kY_m \\ kZ_m \end{pmatrix} \quad (7.4)$$

$$\vec{G} = \frac{1}{\sum_{m=1}^n L(kV_m, kV''_m)} \sum_{m=1}^n L(kV_m, kV''_m) \begin{pmatrix} kX''_m - kX_m \\ kY''_m - kY_m \\ kZ''_m - kZ_m \end{pmatrix}$$

また、複数部材のときは、以下の4つの手順を定め、Aから順番に手順を行いDまで達したときはBに戻り、再びサイクルを繰り返すことで解が求まることが分かった。

- A. 特定の部材の節点に変位を加える。
- B. 変位を与えた部材が移動するべき位置を重心追従法で求める。
- C. Bで移動した部材と接合する全てのFuzzy Nodeの長さを判定する。
- D. Fuzzy Nodeの長さが規定値を超えるものがある時、その超過分を接合される頂点に与える変位と定義する。

重心追従法を組み込んだソフトウェアを開発し、部材の移動を表現することで解法の有効性を確認することができた。

7.2 今後の課題

1. Fuzzy Node を持たないモデル

より普遍的なモデルにおける解法を見つけることが最終的な目標であるが、次は環状部分が2つ存在するモデルの解法等、少しずつ部材を増やしながら検討していく必要がある。

2. 部材の位置を一括で求める解法

- ・回転角制御法

環状モデルでの解法を見つけ、複数の部材に適用できる方法を検討する必要がある。

- ・制約近似法

計算過程で逆行列をとるとき、その行列式が0でないと仮定して解を求めている。逆行列が存在しないときが、どのような形状のモデルであるのかを模索し、逆行列が存在しないときの解の求め方を提示する必要がある。また、制約を与える際に設ける定数が及ぼす影響を複数の部材のときにも検証し、より普遍的なモデルでも解を求められるように検証する必要がある。

3. 部材の位置を徐々に求める方法

重心追従法は立体まで拡張することができたので、今後の課題として、ソフトウェアの計算の高速化、ほかの解法との関係があげられる。

参照文献

- [1] “japan-architects ブログ,” 17 4 2013. [オンライン]. Available: <http://world-architects.blogspot.com/2013/04/i-cube.html>.
- [2] “a + e (建築と食べ物) ,” 25 6 2015. [オンライン]. Available: <http://a-plus-e.blogspot.com/2015/06/agc-studio.html>.
- [3] 西村, “曖昧な節点 Fuzzy Node で接合される集積形態の制御方法,” 日本建築学会大会学 学術講演梗概集, 2016.
- [4] 河村, “Fuzzy Node で接合される小片集積体の機構解析における行列表現,” 日本建築学会大会学 学術講演梗概集, 2017.

謝辞

本論文の執筆にあたり、多くの方々からのご指導、ご協力をいただきました。

はじめに、指導教員である佐藤淳先生には、卒業論文の際から延べ3年間ご指導いただきました。修士課程に進学し、卒論の Fuzzy Node の研究をさらに前進させ、ソフトウェアの開発にまで挑戦することができました。困ったときには、明確なアドバイスをいただき、研究としてまとめることができました。深く感謝申し上げます。

また、副指導教員である清家剛先生、本郷キャンパスでの合同ゼミなどで指導していただいた藤田香先生と権藤智之先生にもお世話になりました。同じ構法系の研究室ではありますが、専門が少しずつ異なり、別の視点からのご助言をしていただきました。誠にありがとうございました。

そして、佐藤事務所の職員の方、先輩方、同期、後輩の皆様にはご指導、ご協力をいただきました。誠にありがとうございました。

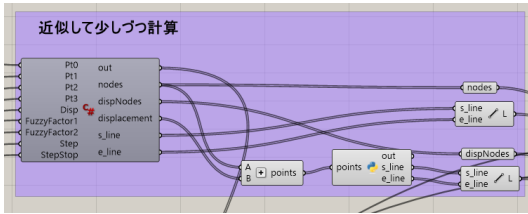
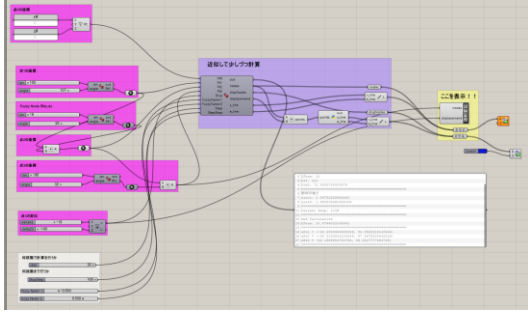
最後に、私のここまでの学生生活を支えていただいた家族に感謝し、謝辞に代えさせていただきます。

河村京介

資料（ソースコード）

Fuzzy Node 2 Parts.gh

制約近似法を Grass hopper で表示するために使用したファイル。

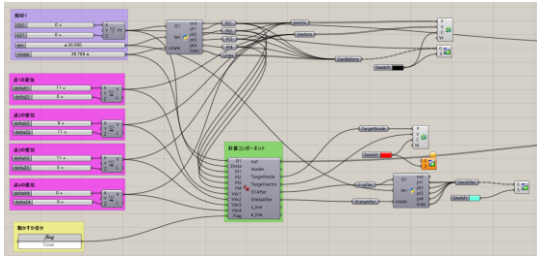


```
private void RunScript(Point3d Pt0, Point3d Pt1, Point3d Pt2,
Point3d Pt3, Vector3d Disp, double FuzzyFactor1, double
FuzzyFactor2, int Step, int StepStop, ref object nodes, ref object
dispNodes, ref object displacement, ref object s_line, ref object
e_line)
{
    // int nNode = 3;
    int nPart = 2;
    int nMaterial = 3;
    this.pt0 = Pt0; this.pt1 = Pt1; this.pt2 = Pt2; this.pt3 = Pt3;
    this.disp = Disp;
    nodes = new Point3d[] {pt0, pt1, pt2, pt3};
    s_line = new Point3d[] {pt0, pt1, pt2};
    e_line = new Point3d[] {pt1, pt2, pt3};
    double L01, LFuzz, L23, L0d2;
    L01 = Math.Sqrt(Math.Pow(pt1.X - pt0.X, 2) +
Math.Pow(pt1.Z - pt0.Z, 2));
    LFuzz = Math.Sqrt(Math.Pow(pt2.X - pt1.X, 2) +
Math.Pow(pt2.Z - pt1.Z, 2));
    L23 = Math.Sqrt(Math.Pow(pt3.X - pt2.X, 2) +
Math.Pow(pt3.Z - pt2.Z, 2));
    L0d2 = Math.Sqrt(Math.Pow(pt3.X + disp.X - pt0.X, 2) +
Math.Pow(pt3.Z + disp.Z - pt0.Z, 2));
    double rot01, rot23;
    rot01 = Math.Atan2(pt1.Z - pt0.Z, pt1.X - pt0.X);
    rot23 = Math.Atan2(pt3.Z - pt2.Z, pt3.X - pt2.X);
    Point3d sPt0, sPt1, sPt2, sPt3;
    double sr01, sr23;
    sPt0 = pt0; sPt1 = pt1; sPt2 = pt2; sPt3 = pt3;
    sr01 = rot01; sr23 = rot23;
    SS = (int) (StepStop * (Step / 100.0));
    for (int s = 1; s <= SS; s++) {
        Point3d tPt3 = new Point3d(pt3.X + ((double) s) * (disp.X /
(double) Step), 0, pt3.Z + ((double) s) * (disp.Z / (double) Step));
        Matrix leftMat = new Matrix(2, 1);
        leftMat[0, 0] = tPt3.X - pt0.X;
        leftMat[1, 0] = tPt3.Z - pt0.Z;
        // 変換行列の宣言
        Matrix transMat = new Matrix(2, 2 * nMaterial);
        transMat[0, 0] = sPt3.X - sPt2.X;
        transMat[1, 0] = sPt3.Z - sPt2.Z;
        transMat[0, 1] = -(sPt3.Z - sPt2.Z);
```

```
transMat[1, 1] = sPt3.X - sPt2.X;
transMat[0, 2] = sPt2.X - sPt1.X;
transMat[1, 2] = sPt2.Z - sPt1.Z;
transMat[0, 3] = -(sPt2.Z - sPt1.Z);
transMat[1, 3] = sPt2.X - sPt1.X;
transMat[0, 4] = sPt1.X - sPt0.X;
transMat[1, 4] = sPt1.Z - sPt0.Z;
transMat[0, 5] = -(sPt1.Z - sPt0.Z);
transMat[1, 5] = sPt1.X - sPt0.X;
Dictionary<string, Matrix> Dic;
Dic = SplitOnesMultipliedRows(transMat);
Matrix cMat, vMat;
cMat = Dic["const"];
vMat = Dic["var"];
Matrix ones = new Matrix(nMaterial, 1);
ones.Zero();
for (int r = 0; r < nMaterial; r++) {
    ones[r, 0] = 1;
}
// vMat を変換する
Matrix vTMat = new Matrix(2, 2 * nPart);
vTMat[0, 0] = vMat[0, 0] + vMat[0, 1] * FuzzyFactor1;
vTMat[1, 0] = vMat[1, 0] + vMat[1, 1] * FuzzyFactor1;
vTMat[0, 1] = vMat[0, 2] + vMat[0, 1] * FuzzyFactor2;
vTMat[1, 1] = vMat[1, 2] + vMat[1, 1] * FuzzyFactor2;
if (vTMat.Invert()) {
    // drs01, drs23 を求める
    double drs01, drs23;
    Matrix leftCalcMat = new Matrix(2, 1);
    leftCalcMat[0, 0] = leftMat[0, 0] - (cMat * ones)[0, 0];
    leftCalcMat[1, 0] = leftMat[1, 0] - (cMat * ones)[1, 0];
    drs01 = (vTMat * leftCalcMat)[1, 0];
    drs23 = (vTMat * leftCalcMat)[0, 0];
    sr01 = drs01 + sr01;
    sr23 = drs23 + sr23;
    sPt1.X = L01 * Math.Cos(sr01);
    sPt1.Z = L01 * Math.Sin(sr01);
    sPt2.X = tPt3.X - L23 * Math.Cos(sr23);
    sPt2.Z = tPt3.Z - L23 * Math.Sin(sr23);
    // 値の保存
    sPt3 = tPt3;
}
}
displacement = new Point3d(new Point3d(0, 0, 0),
new Point3d(sPt1.X - pt1.X, 0, sPt1.Z - pt1.Z),
new Point3d(sPt2.X - pt2.X, 0, sPt2.Z - pt2.Z),
new Point3d(sPt3.X - pt3.X, 0, sPt3.Z - pt3.Z));
dispNodes = new Point3d[] {sPt0, sPt1, sPt2, sPt3};
Print("end Calculation");
LFuzz = Math.Sqrt(Math.Pow(sPt2.X - sPt1.X, 2) +
Math.Pow(sPt2.Z - sPt1.Z, 2));
}
int SS;
Point3d pt0, pt1, pt2, pt3;
Vector3d disp;
private Dictionary<string, Matrix>
SplitOnesMultipliedRows(Matrix mt) {
    Dictionary<string, Matrix> dic = new Dictionary<string,
Matrix>();
    int col = mt.ColumnCount, row = mt.RowCount;
    Matrix varMat, constMat;
    varMat = new Matrix(row, col / 2);
    constMat = new Matrix(row, col / 2);
    for (int c = 0; c < col / 2; c++) {
        for (int r = 0; r < row; r++) {
            varMat[r, c] = mt[r, (2 * c) + 1];
            constMat[r, c] = mt[r, (2 * c)];
        }
    }
    dic.Add("var", varMat);
    dic.Add("const", constMat);
    return dic;
}
```

PullPart.gs

1つの部材に重心追従法を適応し表示したプログラム。



```
private void RunScript(Point3d G1, double Sheta, Point3d Pt1, Point3d Pt2, Point3d Pt3, Point3d Pt4, Vector3d Vec1, Vector3d Vec2, Vector3d Vec3, Vector3d Vec4, bool Flag, ref object Nodes, ref object TargetNode, ref object TargetVector, ref object G1After, ref object ShetaAfter, ref object s_line, ref object e_line)
{
    // 変数の定義
    =====
    // check if calculate
    this.flag = Flag;
    // 重心の座標を保存
    this.g1 = G1; this.sheta = Sheta;
    // 点 1~4 の座標を存
    this.pt1 = Pt1; this.pt2 = Pt2; this.pt3 = Pt3; this.pt4 = Pt4;
    // 点 3 の変位を保存
    this.vec1 = Vec1; this.vec2 = Vec2; this.vec3 = Vec3;
    this.vec4 = Vec4;
    // 座標の配列で保存
    this.nodes = new Point3d[] {pt1, pt2, pt3, pt4};
    // ベクトルを配列で保存
    this.vectors = new Vector3d[] {vec1, vec2, vec3, vec4};
    // 線を引くために点を保存
    s_line = new Point3d[] {pt1, pt2, pt3, pt4};
    e_line = new Point3d[] {pt2, pt3, pt4, pt1};
    // 何点で引っ張られているのか保存
    double pointsRatio = 0;
    bool[] chosen = new bool[] {false, false, false, false};
    // 引っ張られている点の重心を定義
    Point3d pullG = new Point3d(0, 0, 0);
    // 引っ張られている点のベクトルを定義
    Vector3d pullVector = new Vector3d(0, 0, 0);
    // 引っ張られている情報を加工
    for(int i = 0; i < nodes.Count(); i++){
        if (vectors[i].Length > 0){
            // 選択に変更
            chosen[i] = true;
            // 座標を追加
            pullG.X += vectors[i].Length * nodes[i].X;
            pullG.Z += vectors[i].Length * nodes[i].Z;
            // ベクトルを追加
            pullVector.X += vectors[i].Length * vectors[i].X;
            pullVector.Z += vectors[i].Length * vectors[i].Z;
            // 点の数を記録
            pointsRatio += vectors[i].Length;
        }
    }
    // 重心を求める
    pullG.X = pullG.X / pointsRatio;
    pullG.Z = pullG.Z / pointsRatio;
    // ベクトルを平均化
    pullVector.X = pullVector.X / pointsRatio;
    pullVector.Z = pullVector.Z / pointsRatio;
    // 表示用に出力
    TargetNode = pullG;
```

```
TargetVector = pullVector;
// target と 重心の角度
double targetLength = Math.Sqrt(Math.Pow(g1.X - pullG.X, 2) + Math.Pow(g1.Z - pullG.Z, 2));
// 角度の保存
double rotBefore, rotAfter, deltaRot;
rotBefore = Math.Atan2(pullG.Z - g1.Z, pullG.X - g1.X);
rotAfter = Math.Atan2(pullVector.Z + pullG.Z - g1.Z, pullVector.X + pullG.X - g1.X);
if (Math.Sqrt(Math.Pow(pullG.X - g1.X, 2) + Math.Pow(pullG.Z - g1.Z, 2)) < 0.001){
    deltaRot = 0;
} else {
    deltaRot = rotAfter - rotBefore;
}
G1After = new Point3d(pullVector.X + pullG.X + targetLength * Math.Cos(rotAfter + Math.PI), 0, pullVector.Z + pullG.Z + targetLength * Math.Sin(rotAfter + Math.PI));
ShetaAfter = sheta + deltaRot * 180.0 / Math.PI;
// // 直線 01 間の角度、直線 12 間の角度、直線 02"間の角度を求める
// double rot01, rot23;
// rot01 = Math.Atan2(pt1.Z - pt0.Z, pt1.X - pt0.X);
// rot23 = Math.Atan2(pt3.Z - pt2.Z, pt3.X - pt2.X);
Print("successfully finished");
}
// <Custom additional code>
// flag to calculate
bool flag;
// 部材の情報
Point3d g1;
double sheta;
// 点 1~4 の座標
Point3d pt1, pt2, pt3, pt4;
// 点 3 の強制変位
Vector3d vec1, vec2, vec3, vec4;
// 座標の配列で保存
Point3d[] nodes;
// ベクトルを配列で保存
Vector3d[] vectors;
// </Custom additional code>
```

重心追従法ソフトウェア

以下 C#を用いて開発したソフトウェアの
ファイルの詳細です。

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0"
sku=".NETFramework,Version=v4.7.1" />
  </startup>
</configuration>
```

Packages.config

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Newtonsoft.Json" version="11.0.2"
targetFramework="net471" />
  <package id="System.ValueTuple" version="4.5.0"
targetFramework="net471" />
</packages>
```

Program.cs

```
namespace FuzzyNodeCS
{
  static class Program
  {
    [STAThread]
    static void Main() {
      Application.EnableVisualStyles();
      Application.SetCompatibleTextRenderingDefault(false);
      int go = MC.InitApps();
      if (go == 0) {
        Application.Run();
      }
    }
  }
}
```

CalcView.cs

```
namespace FuzzyNodeCS
{
  static class CalcView
  {
    static internal double XRot(PointD vector) {
      return Math.Atan2(vector.Y, vector.X);
    }
    static internal double ZRot(PointD vector) {
      double xyFlat = Math.Sqrt(Math.Pow(vector.X, 2) +
Math.Pow(vector.Y, 2));
      return Math.Atan2(vector.Z, xyFlat);
    }
    static internal PointD Vector(PointD pb, double x,
double y, double z) {
      return new PointD(x - pb.X, y - pb.Y, z - pb.Z);
    }
    static internal PointD Vector(PointD pb, PointD pt) {
      return new PointD(pt.X - pb.X, pt.Y - pb.Y, pt.Z -
pb.Z);
    }
  }
}
```

```
static internal PointD Rotate(PointD bp, double rot,
double x, double y, double z) {
  double L = Math.Sqrt(Math.Pow(x, 2) +
Math.Pow(y, 2) + Math.Pow(z, 2));
  if (L == 0) // 回転してないのと一緒に
  { return bp; }
  double ux = x / L;
  double uy = y / L;
  double uz = z / L;
  PointD vector1 = new PointD(Math.Cos(rot) +
Math.Pow(ux, 2) * (1 - Math.Cos(rot)),
Math.Cos(rot) - uz * Math.Sin(rot),
Math.Cos(rot) + uy * Math.Sin(rot));
  ux * uy * (1 -
ux * uz * (1 -
Math.Cos(rot) + uy * Math.Sin(rot));
  PointD vector2 = new PointD(uy * ux * (1 -
Math.Cos(rot) + uz * Math.Sin(rot),
Math.Cos(rot) +
Math.Pow(uy, 2) * (1 - Math.Cos(rot)),
Math.Cos(rot) -
Math.Cos(rot) - ux * Math.Sin(rot));
  uy * uz * (1 -
Math.Cos(rot) +
PointD vector3 = new PointD(uz * ux * (1 -
Math.Cos(rot) - uy * Math.Sin(rot),
Math.Cos(rot) +
Math.Pow(uz, 2) * (1 - Math.Cos(rot)));
  uz * ux * (1 -
Math.Cos(rot) +
ux * Math.Sin(rot),
Math.Cos(rot) +
Math.Pow(uz, 2) * (1 - Math.Cos(rot)));
  return new PointD(InnerProduct(bp, vector1),
InnerProduct(bp, vector2),
InnerProduct(bp, vector3));
}
static internal PointD Rotate(PointD bp, double rot,
PointD axis) {
  return Rotate(bp, rot, axis.X, axis.Y, axis.Z);
}
static internal double InnerProduct(PointD p1, PointD
p2) {
  return p1.X * p2.X + p1.Y * p2.Y + p1.Z * p2.Z;
}
static internal PointD Outerproduction(PointD p1,
PointD p2) {
  double s1, s2, s3;
  s1 = p1.Y * p2.Z - p1.Z * p2.Y;
  s2 = p1.Z * p2.X - p1.X * p2.Z;
  s3 = p1.X * p2.Y - p1.Y * p2.X;
  return new PointD(s1, s2, s3);
}
static internal PointD ChangeToPlane(Observer ob,
PointD pt) {
  PointD basePoint = ob.Position.Duplicate();
  basePoint.AddVector(1, ob.ViewPoint);
  PointD res = PlaneLineIntersection(basePoint,
ob.ViewPoint,
ob.Position,
pt);
  if (res != null)
  {
    res.AddVector(-1, basePoint);
    double x, y;
    x = InnerProduct(res, ob.ViewPointX) *
ob.Focus;
    y = InnerProduct(res, ob.ViewPointY) *
ob.Focus;
    return new PointD(x, y, 0);
  }
  else
  {
    return null;
  }
}
static private PointD PlaneLineIntersection(PointD
planePt, PointD planeV, PointD p1, PointD p2) {
  PointD res = null;
  double denominator, numerator;
  denominator = InnerProduct(planeV, Vector(p1,
p2));
  if (denominator != 0) {
```

```

        numerator = -InnerProduct(planeV,
Vector(planePt, p1));
        double k = numerator / denominator;
        if (k > 0) {
            res = p1.Duplicate();
            res.AddVector(k, Vector(p1, p2));
        }
    }
    return res;
}
}
}

```

Observer.cs

```

namespace FuzzyNodeCS.View
{
    class Observer
    {
        internal PointD Position;
        internal PointD ViewPoint { get => GetViewVector(); }
        internal PointD ViewPointX { get =>
GetViewXVector(); }
        internal PointD ViewPointY { get =>
GetViewYVector(); }
        internal double RadZ;
        internal double RadXY;
        internal double Focus = 300;
        internal Action _calcView;
        internal Observer(Action calcView) {
            ChangeToDefault();
            _calcView = calcView;
        }
        internal void ChangeToDefault() {
            // デフォルト視点の設定
            Position = new PointD(900, 0, 0);
            // 原点を見るように設定
            RadXY = Math.PI;
            // 水平を見る
            RadZ = 0;
        }
        private PointD GetViewVector(){
            return new PointD(Math.Cos(RadXY) *
Math.Cos(RadZ),
                Math.Sin(RadXY) *
Math.Cos(RadZ),
                Math.Sin(RadZ));
        }
        private PointD GetViewXVector(){
            return new PointD(Math.Cos(RadXY - Math.PI / 2),
                Math.Sin(RadXY - Math.PI / 2),
                0);
        }
        private PointD GetViewYVector(){
            return new PointD(Math.Cos(RadXY) *
Math.Cos(RadZ + Math.PI / 2),
                Math.Sin(RadXY) *
Math.Cos(RadZ + Math.PI / 2),
                Math.Sin(RadZ + Math.PI / 2));
        }
        internal void TransPosition(double x, double z) {
            double deltaX, deltaY;
            deltaX = Math.Cos(RadXY - Math.PI / 2) * x;
            deltaY = Math.Sin(RadXY - Math.PI / 2) * x;
            Position.Trans(deltaX, deltaY, z);
            _calcView?.Invoke();
        }
        internal void MovePosition(string way, double step) {
            double deltaX, deltaY, deltaZ;
            switch (way)
            {
                case "right":
                    deltaX = Math.Cos(RadXY - Math.PI / 2) *
step;
                    deltaY = Math.Sin(RadXY - Math.PI / 2) *
step;
                    deltaZ = 0;
                    break;

```

```

                case "left":
                    deltaX = - Math.Cos(RadXY - Math.PI / 2)
* step;
                    deltaY = - Math.Sin(RadXY - Math.PI / 2)
* step;
                    deltaZ = 0;
                    break;
                case "forward":
                    deltaX = Math.Cos(RadXY) * step;
                    deltaY = Math.Sin(RadXY) * step;
                    deltaZ = 0;
                    break;
                case "backward":
                    deltaX = -Math.Cos(RadXY) * step;
                    deltaY = -Math.Sin(RadXY) * step;
                    deltaZ = 0;
                    break;
                case "up":
                    deltaX = 0;
                    deltaY = 0;
                    deltaZ = - 1 * step;
                    break;
                case "down":
                    deltaX = 0;
                    deltaY = 0;
                    deltaZ = 1 * step;
                    break;
                default:
                    throw new NotImplementedException();
            }
            Position.Trans(deltaX, deltaY, deltaZ);
            _calcView?.Invoke();
        }
        internal void ChangeView(double x, double z) {
            RadXY += x / 100;
            RadZ += z / 100;
            _calcView?.Invoke();
        }
    }
}

```

ProjectInfo.cs

```

namespace FuzzyNodeCS
{
    static internal class ProjectInfo
    {
        internal static bool IsNew {
            get => (MC.projectFilePath == null);
        }

        internal static (Exception,int) OpenData(string filePath) {
            try
            {
                ;
                return (null,
GOpenFromFile.OpenFromFile(filePath));
            }
            catch (Exception ex)
            {
                return (ex, -1);
            }
        }
        internal static Exception SaveData(bool Save) {
            try
            {
                if (IsNew || !Save)
                {
                    var filePath = GOpenFromFile.SaveFile();

                    if (filePath != "")
                    {
                        MC.projectFilePath = filePath;
                    }
                }
            }
            else
            {

```



```

        pS =
part.PointFolder[part.PointFolder.Count() - 1].ToPointI0;
    }
    else
    {
        pS = part.PointFolder[i - 1].ToPointI0;
    }
    pE = part.PointFolder[i].ToPointI0;
    pen.Width = 2;
    pen.DashStyle = DashStyle.Solid;
    g.DrawLine(pen, pS, pE);
    if (part.PointFolder[i].NState ==
NodeState.FREE)
    {
        pen.Width = 1;
        pen.DashStyle = DashStyle.Dash;
        pen.DashPattern = new float[] { 3.0F,
2.0F };
        g.DrawEllipse(pen, pE.X -
MC.MO.FuzzyLen / 2, pE.Y - MC.MO.FuzzyLen / 2,
MC.MO.FuzzyLen, MC.MO.FuzzyLen);
    }
}
}
private void TestDrawPart(Graphics g, TestPart part,
double dx, double dy, Pen pen) {
    ChangePenColor(part, pen);
    foreach ((int, int) nodes in part.Lines)
    {
        try
        {
            if
(part.PointFolder[nodes.Item1].TransCoord != null &&
part.PointFolder[nodes.Item2].TransCoord != null)
                g.DrawLine(pen,
part.PointFolder[nodes.Item1].TransCoord.CTransI(dx, dy),
part.PointFolder[nodes.Item2].TransCoord.CTransI(dx, dy));
        }
        catch {}
    }
    foreach (TestNode node in part.PointFolder.Values)
    {
        if (node.OS == ObjectState.chosen)
        {
            try
            {
                Point pt =
node.TransCoord.CTransI(dx, dy);
                g.FillEllipse(GetBrushColor(node), pt.X
- 6, pt.Y - 6, 12, 12);
            }
            catch
            {
            }
        }
    }
}
internal void TestDrawParts(Bitmap bmp,
List<TestPart> parts)
{
    using (Graphics g = Graphics.FromImage(bmp))
    {
        g.InterpolationMode =
System.Drawing.Drawing2D.InterpolationMode.High;
        double dx = bmp.Size.Width / 2;
        double dy = bmp.Size.Height / 2;
        Pen pen = new Pen(Color.White, 2);
        foreach (TestPart part in parts)
        {
            TestDrawPart(g, part, dx, dy, pen);
        }
    }
}
internal void DrawParts(MassObject MObject, Bitmap
bmp) {
    using (Graphics g = Graphics.FromImage(bmp))

```

```

        g.InterpolationMode =
System.Drawing.Drawing2D.InterpolationMode.High;
        foreach (Part2D part in MObject.Parts.Values)
        {
            Pen pen = new Pen(Color.Yellow, 2);
            Point center =
part.Center.Coord.ToPointI_XY();
            ChangePenColor(part, pen);
            pen.Width = 2;
            g.DrawEllipse(pen, center.X - 1, center.Y -
1, 2, 2);
            for (int i = 0; i < part.PointFolder.Count();
i++)
            {
                // 始点と終点の宣言
                Point pS, pE;
                if (i == 0) {
                    pS =
part.PointFolder[part.PointFolder.Count() - 1].ToPointI0;
                }
                else{
                    pS = part.PointFolder[i -
1].ToPointI0;
                }
                pE = part.PointFolder[i].ToPointI0;
                // 色の設定
                ChangePenColor(part, pen);
                // 図形の描写
                pen.Width = 2;
                pen.DashStyle = DashStyle.Solid;
                g.DrawLine(pen, pS, pE);
                // 接合部の描写
                pen.Width = 1;
                pen.DashStyle = DashStyle.Dash;
                pen.DashPattern = new float[] { 3.0F,
2.0F };
                if (part.PointFolder[i].NState ==
NodeState.FREE)
                {
                    pen.Color = Color.Red;
                    g.DrawEllipse(pen, pE.X - 20 / 2, pE.Y - 20 / 2, 20, 20);
                    g.DrawEllipse(pen, pE.X - 2 / 2, pE.Y - 2 / 2, 2, 2);
                    g.DrawEllipse(pen, pE.X - 4 / 2, pE.Y - 4 / 2, 4, 4);
                    g.DrawEllipse(pen, pE.X - 8 / 2, pE.Y - 8 / 2, 8, 8);
                    g.DrawEllipse(pen, pE.X - 16 / 2, pE.Y - 16 / 2, 16, 16);
                }
                else if (part.PointFolder[i].NState == NodeState.FIXED) {
                    pen.Color = Color.White;
                    g.DrawEllipse(pen, pE.X -
MC.MO.FuzzyLen / 2, pE.Y - MC.MO.FuzzyLen / 2,
MC.MO.FuzzyLen, MC.MO.FuzzyLen);
                    bool flag = true;
                    double counter = 0;
                    while (flag) {
                        if (Math.Pow(2, counter) <=
MC.MO.FuzzyLen && MC.MO.FuzzyLen < Math.Pow(2,
counter + 1)) {
                            flag = false;
                        }
                    }
                    else{
                        counter += 1;
                    }
                }
                if (counter > 0) {
                    for (int k = 1; k < counter; k++){
                        int r = (int)Math.Pow(2, k);
                        g.DrawEllipse(pen, pE.X - r / 2, pE.Y - r / 2, r, r);
                    }
                }
            }
        }
    }
}
private void ChangePenColor(TestPart part, Pen pen) {
    if (part.OState == ObjectState.unchosen) {
        pen.Color = Color.White;
    }
    else if (part.OState == ObjectState.chosen) {
        pen.Color = Color.LightGreen;
    }
}

```

```

    }
    else if (part.OSState == ObjectState.candidate) {
        pen.Color = Color.Red;
    }
    else{
        MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
    }
}
private Brush GetBrushColor(TestNode node)
{
    Brush brush;
    if (node.OS == ObjectState.unchosen) {
        brush = new SolidBrush(Color.White);
    }
    else if (node.OS == ObjectState.chosen) {
        brush = new SolidBrush(Color.LightGreen);
    }
    else if (node.OS == ObjectState.candidate) {
        brush = new SolidBrush(Color.Red);
    }
    else{
        MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
        throw new NotImplementedException();
    }
    return brush;
}
private void ChangePenColor(Part2D part, Pen pen)
{
    if (part.OSState == ObjectState.unchosen) {
        pen.Color = Color.White;
    }
    else if (part.OSState == ObjectState.chosen) {
        pen.Color = Color.LightGreen;
    }
    else if (part.OSState == ObjectState.candidate)
        pen.Color = Color.Red;
    else
    {
        MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
    }
}
}
}

```

Geometry.cs

```

using FuzzyNodeCS.DataClass;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;

namespace FuzzyNodeCS
{
    class Geometry
    {
        internal (List<int>, bool) ClickCenter(MassObject mo,
PointD pt) {
            List<int> list = new List<int>();
            bool flag = false;
            foreach (int index in mo.Parts.Keys)
            {
                if
(pt.GetDistance(mo.Parts[index].Center.Coord) <
MC.EffectiveClick) {
                    flag = true;
                    list.Add(index);
                }
            }
            return (list, flag);
        }
        internal bool ClickCenter(Part2D part, PointD pt) {

```

```

            bool flag = false;
            if (pt.GetDistance(part.Center.Coord) <
MC.EffectiveClick)
            {
                flag = true;
            }
            return (flag);
        }
        internal (List<int>, bool) ClickVertex(Part2D part,
PointD pt){
            List<int> list = new List<int>();
            bool flag = false;
            foreach (int index in part.PointFolder.Keys {
                if
(pt.GetDistance(part.PointFolder[index].Coord) <
MC.EffectiveClick) {
                    flag = true;
                    list.Add(index);
                }
            }
            return (list, flag);
        }
        internal void Disconnect(MassObject mo, int index) {
            List<(int, int)> res =
mo.Parts[index].ClearConnection();

            for (int i = 0; i < res.Count(); i++)
            {
                mo.Parts[res[i].Item1].PointFolder[res[i].Item2].ClearConnectio
n();
            }
        }
        internal void FindConnection(MassObject mo, int
SelectedPart, bool disconnect) {
            if (disconnect) {
                Disconnect(mo, SelectedPart);
            }
            foreach (int key in mo.Parts.Keys) {
                if (key != SelectedPart)
                {
                    if
(mo.Parts[key].Center.GetDistance(mo.Parts[SelectedPart].Cent
er) < 3 * mo.MasterPiece.MaxLength()) {
                        foreach (int index0 in
mo.Parts[key].PointFolder.Keys) {
                            if
(mo.Parts[key].PointFolder[index0].NState ==
NodeState.FREE) {
                                foreach (int index1 in
mo.Parts[SelectedPart].PointFolder.Keys) {
                                    if
(mo.Parts[key].PointFolder[index0].GetDistance(mo.Parts[Selec
tedPart].PointFolder[index1]) < mo.FuzzyLen) {
                                        mo.Parts[key].PointFolder[index0].SetConnection(SelectedPart,
index1);
                                        mo.Parts[SelectedPart].PointFolder[index1].SetConnection(key,
index0);
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
        internal void FixedPull(MassObject mo, PointD pt, int
vertex) {
            Part2D savedPart =
mo.Parts[mo.IsSelected].Duplicate();
            mo.Parts[mo.IsSelected].Pull(pt, vertex);
            List<(int, int, int, PointD)> flag =
CheckFuzzy(mo, mo.IsSelected);
            if (flag.Count() != 0) {
                mo.Parts[mo.IsSelected] = savedPart;
            }
        }
        internal void FollowedPullTest(MassObject mo, PointD
pt, int vertex)
        {
            string savedMo = mo.ToJson();

```

```

        MassObject _mo =
Newtonsoft.Json.JsonConvert.DeserializeObject<MassObject>(s
avedMo);
        Debug.WriteLine("FollowedPullTest()");
        (Forces, int) res;
        res.Item1 = new Forces0;
        Force force = new Force0;
        PointD from =
_mo.Parts[_mo.IsSelected].PointFolder[vertex].Coord;
        force.AddForceForVertex(vertex, new PointD(pt.X -
from.X, pt.Y - from.Y));
        res.Item1.AddForce(MC.MO.IsSelected, force);
        res.Item2 = 0;
        while (res.Item2 == 0) {
            if (res.Item1.counter != 0) {
                _mo =
Newtonsoft.Json.JsonConvert.DeserializeObject<MassObject>(s
avedMo);
            }
            res = MovedByForces(_mo, res.Item1);
        }
        if (res.Item2 == -1){
            MC.MO =
Newtonsoft.Json.JsonConvert.DeserializeObject<MassObject>(s
avedMo);
        }
        else if (res.Item2 == 1) {
            MC.MO = _mo;
        }
    }
}
private (Forces, int) MovedByForces(MassObject mo,
Forces forces) {
    forces.counter += 1;
    (Forces, int) res;
    res.Item1 = forces.Duplicate0;
    if (forces.counter < 6) {
        res.Item2 = 1;
        foreach (KeyValuePair<int, Force> force in
forces.VectorsFile) {
            mo.Parts[force.Key].Pull(force.Value);
        }
        foreach (int moved in forces.VectorsFile.Keys) {
            List<(int, int, int, int, PointD)> limits =
CheckFuzzy(mo, moved);
            if (limits.Count() > 0) {
                res.Item2 = 1;
                foreach ((int, int, int, int, PointD)
limit in limits) {
                    if
(forces.VectorsFile.ContainsKey(limit.Item1) &&
forces.VectorsFile.ContainsKey(limit.Item3)) {
                        res.Item2 = -1;
                        return res;
                    }
                    else {
                        res.Item2 = 0;
                        if
(res.Item1.VectorsFile.ContainsKey(limit.Item3))
res.Item1.VectorsFile[limit.Item3].AddForceForVertex(limit.Ite
m4, limit.Item5);
                    }
                    else{
                        Force force = new Force0;
                        force.AddForceForVertex(limit.Item4, limit.Item5)
res.Item1.AddForce(limit.Item3, force);
                    }
                }
            }
        }
    }
    else{
        res.Item2 = -1;
    }
    return res;
}
internal List<(int, int, int, int, PointD)>
CheckFuzzy(MassObject mo, int partId) {
    List<(int, int, int, int, PointD)> res = new List<(int,
int, int, int, PointD)>0;

```

```

        foreach (int key in
mo.Parts[partId].PointFolder.Keys) {
            if (mo.Parts[partId].PointFolder[key].NState
== NodeState.FIXED) {
                double len =
mo.Parts[partId].PointFolder[key].
GetDistance(mo.Parts[mo.Parts[partId].PointFolder[key].Conne
ctedPartId].PointFolder[mo.Parts[partId].PointFolder[key].Con
nectedNodeId]);
                if (len > mo.FuzzyLen) {
                    PointD p0 =
mo.Parts[partId].PointFolder[key].Coord;
                    PointD p1 =
mo.Parts[mo.Parts[partId].PointFolder[key].ConnectedPartId].P
ointFolder[mo.Parts[partId].PointFolder[key].ConnectedNodeId]
.Coord;
                    PointD delta = new PointD((p0.X -
p1.X) * ((len - mo.FuzzyLen) * 1.1) / len, (p0.Y - p1.Y) * ((len -
mo.FuzzyLen) * 1.1) / len);
                    mo.Parts[partId].PointFolder[key].ConnectedPartId.ToString() +
" ";
                    res.Add((partId, key,
mo.Parts[partId].PointFolder[key].ConnectedPartId,
mo.Parts[partId].PointFolder[key].ConnectedNodeId, delta);
                }
            }
        }
        return res;
    }
    internal void FollowedPull3D(List<TestPart> parts,
PointD pt, int partId, int vertexId) {
        parts[0].Pull(pt, vertexId);
        double len1 =
parts[0].PointFolder[4].Coord.GetDistance(parts[1].PointFolder[
5].Coord);
        if (len1 > 20) {
            double ratio1 = 1 - 20/ Math.Ceiling(len1);
            PointD move1 = new
PointD((parts[0].PointFolder[4].Coord.X -
parts[1].PointFolder[5].Coord.X) * ratio1,
(parts[0].PointFolder[4].Coord.Y -
parts[1].PointFolder[5].Coord.Y) * ratio1,
(parts[0].PointFolder[4].Coord.Z -
parts[1].PointFolder[5].Coord.Z) * ratio1);
            parts[1].Pull(move1, 5);
            double len2 =
parts[1].PointFolder[4].Coord.GetDistance(parts[2].PointFolder[
5].Coord);
            if (len2 > 20) {
                double ratio2 = 1 - 20/ Math.Ceiling(len2);
                PointD move2 = new
PointD((parts[1].PointFolder[4].Coord.X -
parts[2].PointFolder[5].Coord.X) * ratio2,
(parts[1].PointFolder[4].Coord.Y -
parts[2].PointFolder[5].Coord.Y) * ratio2,
(parts[1].PointFolder[4].Coord.Z -
parts[2].PointFolder[5].Coord.Z) * ratio2);
                parts[2].Pull(move2, 5);
            }
        }
    }
}
#endregion
}
}

```

RedoUndo.cs

```

namespace FuzzyNodeCS
{
    class RedoUndo
    {
        internal List<string> History;
        internal int CurrentIndex;
        private const int MaxUndo = 30;
        internal RedoUndo()
        {

```



```

        CurrentIndex = -1;
        History = new List<string>();
    }
    internal void Add(MassObject mo)
    {
        if (CurrentIndex + 1 < History.Count()){
            History.RemoveRange(CurrentIndex + 1,
History.Count() - (CurrentIndex + 1));
        }
        if (History.Count() >= MaxUndo) {
            History.RemoveAt(0);
            History.Add(mo.ToJson());
        }
        else{
            History.Add(mo.ToJson());
            CurrentIndex += 1;
        }
    }
    internal void Init(MassObject mo) {
        History.Clear();
        History.Add(mo.ToJson());
        CurrentIndex = 0;
    }
    internal (bool, MassObject) Undo(){
        return ChangeIndex(-1);
    }
    internal (bool, MassObject) Redo(){
        return ChangeIndex(1);
    }
    private (bool, MassObject) ChangeIndex(int delta) {
        try
        {
            if (CurrentIndex + delta > -1 && CurrentIndex
+ delta < History.Count()){
                CurrentIndex += delta;
                return (true,
Newtonsoft.Json.JsonConvert.DeserializeObject<MassObject>(
History[CurrentIndex]));
            }
            else{
                return (false, null);
            }
        }
        catch{
            MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
            return (false, null);
        }
    }
}
}
}

```

MC.sc

```

namespace FuzzyNodeCS
{
    internal static class MC
    {
        internal static string ApplicationName = "FuzzyNode";
        internal static string projectFilePath;
        internal const string fileType = "fnode_file";
        internal const string fileExtension = ".fnode";
        internal static string ProjectName { get =>
Path.GetFileName(MC.projectFilePath); }
        internal static string ProjectNameNoExtension { get =>
Path.GetFileNameWithoutExtension(MC.ProjectName); }
        internal static bool OpeningNewExe = false;
        internal static string ArgPath { get; set; } = "";
        public static int EffectiveClick = 10;
        public static MassObject MO;
        private static Action SetStatus { get; set; }
        internal static int InitApps(){
            try
            {
                int flag = 0;
                MO = new MassObject();
                string[] args;

```

```

                if (ArgPath == ""){
                    args =
Environment.GetCommandLineArgs();
                }
                else{
                    args = new[] { "", ArgPath };
                }
                if (args.Count() > 1) {
                    var filePath = args[1];
                    Exception ex;
                    (ex, flag) =
ProjectInfo.OpenData(filePath);
                }
                if (flag != -1) {
                    if (args.Count() > 1){
                        var frm = new MainForm(false);
                        frm.SetStatus();
                        SetStatus = frm.SetStatus;
                        frm.Draw();
                        frm.StartPosition =
FormStartPosition.CenterScreen;
                        Show(frm);
                    }
                    else{
                        var frm = new MainForm(true);
                        frm.SetStatus();
                        SetStatus = frm.SetStatus;
                        frm.StartPosition =
FormStartPosition.CenterScreen;
                        Show(frm);
                    }
                }
                return flag;
            }
            catch (Exception ex) {
                MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
                return -1;
            }
        }
        internal static void Save(bool Save)
        {
            int selected = MC.MO.IsSelected;
            MC.MO.SetSelected(-1);
            ProjectInfo.SaveData(Save);
            MC.MO.SetSelected(selected);
            SetStatus();
        }
        internal static void Show(Form frm) {
            frm.Show();
        }
        internal static void Exit(bool exitFlg = false) {
            if (exitFlg) {
                Application.Exit();
            }
        }
    }
}
}
}

```

Force.cs

```

namespace FuzzyNodeCS.DataClass
{
    class Force
    {
        public Dictionary<int, PointD> Vectors;
        public Force(){
            Vectors = new Dictionary<int, PointD>();
        }
        public void AddForceForVertex(int vertexIndex, double
x, double y) {
            AddForceForVertex(vertexIndex, new PointD(x, y));
        }
        public void AddForceForVertex(int vertexIndex, PointD
point) {
            Vectors.Add(vertexIndex, point);
        }
    }
}
}
}

```

Forces.cs

```

namespace FuzzyNodeCS.DataClass
{
    class Forces
    {
        public Dictionary<int, Force> VectorsFile;
        public int counter;
        public Forces()
        {
            counter = 0;
            VectorsFile = new Dictionary<int, Force>();
        }
        public void AddForce(int Id, Force force)
        {
            VectorsFile.Add(Id, force);
        }
        public Forces Dupilcate()
        {
            string jsonStr =
Newtonsoft.Json.JsonConvert.SerializeObject(this);
            Forces copiedForces =
Newtonsoft.Json.JsonConvert.DeserializeObject<Forces>(jsonStr);
            return copiedForces;
        }
    }
}

```

MassObject.cs

```

namespace FuzzyNodeCS.DataClass
{
    class MassObject
    {
        public Part2D MasterPiec;
        public Dictionary<int, Part2D> Parts;
        public int FuzzyLen = 1;
        public DateTime SavedDate;
        public int IsSelected;
        public MassObject()
        {
            Parts = new Dictionary<int, Part2D>();
            IsSelected = -1;
        }
        public void Clear()
        {
            Parts = new Dictionary<int, Part2D>();
            IsSelected = -1;
        }
        public void SetSelected(int i)
        {
            IsSelected = i;
            foreach (int index in Parts.Keys)
            {
                if (index == i)
                {
                    Parts[index].OState = ObjectState.chosen;
                    IsSelected = index;
                }
                else
                {
                    Parts[index].OState =
ObjectState.unchosen;
                }
            }
        }
        public void SetCandidate(int i)
        {
            foreach (int index in Parts.Keys)
            {
                if (index == i)
                {
                    Parts[index].OState =
ObjectState.candidate;
                    IsSelected = index;
                }
                else
                {
                    Parts[index].OState =
ObjectState.unchosen;
                }
            }
        }
        public string ToJson()
        {
            return
Newtonsoft.Json.JsonConvert.SerializeObject(this);
        }
    }
}

```

Node.cs

```

namespace FuzzyNodeCS
{
    class Node
    {
        public PointD Coord;
        public NodeState NState;
        public ObjectState OState;
        public int ConnectedPartId;
        public int ConnectedNodeId;
        public double R;
        public double A;
        public Node()
        {
            Coord = new PointD(0, 0);
            ConnectedPartId = -1;
            ConnectedNodeId = -1;
            NState = NodeState.FREE;
            OState = ObjectState.unchosen;
            R = -1; A = -1;
        }
        public Node(PointD pt, NodeState ns =
NodeState.FREE)
        {
            Coord = pt;
            ConnectedPartId = -1;
            ConnectedNodeId = -1;
            NState = ns;
            OState = ObjectState.unchosen;
            R = -1; A = -1;
        }
        public Node(double x, double y, NodeState ns =
NodeState.FREE)
        {
            Coord = new PointD(x, y);
            ConnectedPartId = -1;
            ConnectedNodeId = -1;
            NState = ns;
            OState = ObjectState.unchosen;
            R = -1; A = -1;
        }
        public Node(PointD pt, double r, double a, NodeState ns
= NodeState.FREE)
        {
            double aa = a * Math.PI / 180.0;
            double x = r * Math.Cos(aa) + pt.X;
            double y = r * Math.Sin(aa) + pt.Y;
            Coord = new PointD(x, y);
            ConnectedPartId = -1;
            ConnectedNodeId = -1;
            NState = ns;
            OState = ObjectState.unchosen;
            R = r; A = a;
        }
        public void SetRA(PointD pt)
        {
            R = GetDistance(pt);
            A = GetAngle(pt);
        }
        public void Trans(PointD pt)
        {
            Trans(pt.X, pt.Y, pt.Z);
        }
        public void Trans(double x, double y, double z)
        {
            Coord.Trans(x, y, z);
        }
        public void MoveTo(PointD pt)
        {
            MoveTo(pt.X, pt.Y, pt.Z);
        }
        public void MoveTo(double x, double y, double z)
        {
            Coord.MoveTo(x, y, z);
        }
        public void Rotate(PointD pt, double rot)
        {
            Rotate(pt.X, pt.Y, rot);
        }
        public void Rotate(double x, double y, double rot)
        {
            Coord.RotateXY(x, y, rot);
        }
        public double GetDistance(Node node)
        {
            return GetDistance(node.Coord);
        }
    }
}

```

```

    }
    public double GetDistance(PointD pt) {
        return GetDistance(pt.X, pt.Y, pt.Z);
    }
    }
    public double GetDistance(double x, double y, double z)
    {
        return Coord.GetDistance(x, y, z);
    }
    }
    public double GetAngle(PointD pt) {
        return GetAngle(pt.X, pt.Y);
    }
    }
    public double GetAngle(double x, double y) {
        return Coord.GetAngleXY(x, y);
    }
    }
    public Point ToPointI() {
        return Coord.ToPointI_XY();
    }
    }
    public void SetSelection(ObjectState os) {
        OState = os;
    }
    }
    public void SetConnection(int partId, int nodeId) {
        ConnectedPartId = partId;
        ConnectedNodeId = nodeId;
        NState = NodeState.FIXED;
    }
    }
    public (int, int) ClearConnection() {
        int value1, value2;
        value1 = ConnectedPartId;
        value2 = ConnectedNodeId;
        ConnectedPartId = -1;
        ConnectedNodeId = -1;
        NState = NodeState.FREE;
        return (value1, value2);
    }
    }
    public override string ToString() {
        return Coord.ToString();
    }
    }
    class TestNode
    {
        public PointD Coord;
        public PointD RealativeCoord;
        public PointD TransCoord;
        public double R;
        public NodeState NState;
        public ObjectState OS = ObjectState.unchosen;
        public int ConnectedPartId;
        public int ConnectedNodeId;
        public TestNode() {
            public TestNode(double x, double y, double z, NodeState
ns = NodeState.FREE) {
                Coord = new PointD(x, y, z);
                RealativeCoord = new PointD(x, y, z);
                TransCoord = null;
                ConnectedPartId = -1;
                ConnectedNodeId = -1;
                NState = ns;
            }
        }
        public void SetR(PointD pt) {
            R = RealativeCoord.GetDistance(pt);
        }
        }
        public void Trans(PointD pt) {
            Trans(pt.X, pt.Y, pt.Z);
        }
        }
        public void Trans(double x, double y, double z) {
            Coord.Trans(x, y, z);
        }
        }
        public void MoveTo(PointD pt) {
            MoveTo(pt.X, pt.Y, pt.Z);
        }
        }
        public void MoveTo(double x, double y, double z) {
            Coord.MoveTo(x, y, z);
        }
        }
        internal (double, double) CalcRotation(double x, double
y, double z) {
            PointD tp = CalcView.Vector(Coord, x, y, z);
            double rotX = CalcView.XRot(tp);
            double rotZ = CalcView.ZRot(tp);
            return (rotX, rotZ);
        }
        }
        public void SetConnection(int partId, int nodeId) {
            ConnectedPartId = partId;

```

```

        ConnectedNodeId = nodeId;
        NState = NodeState.FIXED;
    }
    }
    public (int, int) ClearConnection() {
        int value1, value2;
        value1 = ConnectedPartId;
        value2 = ConnectedNodeId;
        ConnectedPartId = -1;
        ConnectedNodeId = -1;
        NState = NodeState.FREE;
        return (value1, value2);
    }
    }
    public override string ToString() {
        return Coord.ToString();
    }
    }
    }
    }

```

Part.cs

```

namespace FuzzyNodeCS
{
    class Part2D
    {
        public Node Center;
        public double Rotation;
        public ObjectState OState;
        public Dictionary<int, Node> PointFolder;
        public List<int> SelectedPoints;
        public Part2D() {
            OState = ObjectState.unchosen;
            SelectedPoints = new List<int>();
            PointFolder = new Dictionary<int, Node>();
        }
        public Part2D(PointD center, int numOfVertex, int
diagonalLength) {
            OState = ObjectState.unchosen;
            SelectedPoints = new List<int>();
            PointFolder = new Dictionary<int, Node>();
            Center = new Node(center);
            double unitAngle = 2 * Math.PI / numOfVertex;
            for (int i = 0; i < numOfVertex; i++) {
                Node node = new Node(center, diagonalLength,
unitAngle * i);
                PointFolder.Add(i, node);
            }
        }
        public Part2D(List<PointD> pL) {
            OState = ObjectState.unchosen;
            SelectedPoints = new List<int>();
            PointFolder = new Dictionary<int, Node>();
            double gX = 0, gY = 0;
            foreach (PointD pt in pL) {
                gX += pt.X;
                gY += pt.Y;
            }
            gX = gX / (double)pL.Count();
            gY = gY / (double)pL.Count();
            Center = new Node(gX, gY);
            for (int i = 0; i < pL.Count(); i++) {
                Node node = new Node(pL[i],
NodeState.CONNECTION);
                node.SetRA(Center.Coord);
                PointFolder.Add(i, node);
            }
        }
        public Part2D Duplicate() {
            string jsonStr =
Newtonsoft.Json.JsonConvert.SerializeObject(this);
            Part2D copiedPart =
Newtonsoft.Json.JsonConvert.DeserializeObject<Part2D>(jsonS
tr);
            return copiedPart;
        }
        }
        private void AddRotation(double deltaRot) {
            Rotation += deltaRot;
        }
        }
        private void SetRotation(double deltaRot) {

```

```

        Rotation = deltaRot;
    }
    private double CalcRotation(double x, double y, int
vertex) {
        double deltaRot = 0;
        double rotBefore =
Math.Atan2(PointFolder[vertex].Coord.Y - Center.Coord.Y,
PointFolder[vertex].Coord.X - Center.Coord.X);
        double rotAfter = Math.Atan2(y - Center.Coord.Y, x
- Center.Coord.X);
        deltaRot = rotAfter - rotBefore;
        return deltaRot;
    }
    public void SetVertex(double rot = 0) {
        Rotation = rot;
        foreach(KeyValuePair<int, Node> node in
PointFolder) {
            node.Value.Coord.X = node.Value.R *
Math.Cos(node.Value.A + Rotation) + this.Center.Coord.X;
            node.Value.Coord.Y = node.Value.R *
Math.Sin(node.Value.A + Rotation) + this.Center.Coord.Y;
        }
    }
    public void MoveTo(double x, double y, double z) {
        Center.MoveTo(x, y, z);
        SetVertex(Rotation);
    }
    public void MoveTo(PointD pt) {
        MoveTo(pt.X, pt.Y, pt.Z);
    }
    public void Rotate(double x, double y, int vertex) {
        AddRotation(CalcRotation(x, y, vertex));
        SetVertex(Rotation);
    }
    public void Rotate(PointD pt, int vertex) {
        Rotate(pt.X, pt.Y, vertex);
    }
    public void Pull(double x, double y, int vertex) {
        double centerX, centerY;
        AddRotation(CalcRotation(x, y, vertex));
        centerX = PointFolder[vertex].R *
Math.Cos(Rotation + PointFolder[vertex].A + Math.PI) + x;
        centerY = PointFolder[vertex].R *
Math.Sin(Rotation + PointFolder[vertex].A + Math.PI) + y;
        Center.MoveTo(centerX, centerY, 0);
        SetVertex(Rotation);
    }
    public void Pull(PointD pt, int vertex) {
        Pull(pt.X, pt.Y, vertex);
    }
    public void Pull(Force force) {
        PointD gPoint = new PointD(0, 0);
        PointD gPower = new PointD(0, 0);
        double denominator = 0;
        foreach (KeyValuePair<int, PointD> f in
force.Vectors) {
            gPoint.X += PointFolder[f.Key].Coord.X *
f.Value.Len;
            gPoint.Y += PointFolder[f.Key].Coord.Y *
f.Value.Len;
            gPower.X += f.Value.X * f.Value.Len;
            gPower.Y += f.Value.Y * f.Value.Len;
            denominator += f.Value.Len;
        }
        if (denominator != 0) {
            gPoint.X = gPoint.X / denominator;
            gPoint.Y = gPoint.Y / denominator;
            gPower.X = gPower.X / denominator;
            gPower.Y = gPower.Y / denominator;
            double rotBefore = Math.Atan2(gPoint.Y -
Center.Coord.Y, gPoint.X - Center.Coord.X);
            double rotAfter = Math.Atan2(gPoint.Y +
gPower.Y - Center.Coord.Y, gPoint.X + gPower.X -
Center.Coord.X);
            double deltaRot = rotAfter - rotBefore;
            double dis =
gPoint.GetDistance(Center.Coord);
            AddRotation(deltaRot);
            double centerX, centerY;
            centerX = dis * Math.Cos(rotAfter + Math.PI)
+ gPoint.X + gPower.X;

```

```

        centerY = dis * Math.Sin(rotAfter + Math.PI) +
gPoint.Y + gPower.Y;
        Center.MoveTo(centerX, centerY, 0);
        SetVertex(Rotation);
    }
    }
    public int FreeNum()
    {
        int counter = 0;
        foreach (Node node in PointFolder.Values) {
            if (node.NState == NodeState.FREE) {
                counter += 1;
            }
        }
        return counter;
    }
    public List<(int,int)> ClearConnection()
    {
        List<(int, int)> res = new List<(int, int)>(0);
        foreach (Node node in PointFolder.Values) {
            if (node.NState == NodeState.FIXED) {
                (int, int) item = node.ClearConnection();
                res.Add(item);
            }
        }
        return res;
    }
    public void SetConnection(int opponentPartId, int
opponentNodeId, int selfVertexNum) {
        PointFolder[selfVertexNum].SetConnection(opponentPartId,
opponentNodeId);
    }
    public double MaxLength()
    {
        double max = 0;
        foreach (Node node in PointFolder.Values) {
            if (node.R > max) {
                max = node.R;
            }
        }
        return max;
    }
    }
    class TestPart
    {
        public TestNode Center;
        public double ARot;
        public double XRot;
        public double ZRot;
        public PointD axisVector { get => new
PointD(Math.Cos(XRot) * Math.Cos(ZRot),
Math.Sin(XRot) * Math.Cos(ZRot),
Math.Sin(ZRot)); }
        public ObjectState OState;
        public Dictionary<int, TestNode> PointFolder;
        public List<(int, int)> Lines;
        public TestPart()
        {
            OState = ObjectState.unchosen;
            PointFolder = new Dictionary<int, TestNode>(0);
            Lines = new List<(int, int)>(0);
        }
        public TestPart Duplicate()
        {
            string jsonStr =
Newtonsoft.Json.JsonConvert.SerializeObject(this);
            TestPart copiedPart =
Newtonsoft.Json.JsonConvert.DeserializeObject<TestPart>(json
Str);
            return copiedPart;
        }
        public void SetVertexCoord()
        {
            foreach (TestNode node in PointFolder.Values) {
                PointD pt;
                pt = CalcView.Rotate(node.RelativeCoord, XRot, 0, 0, 1);
                PointD rotAxis = CalcView.Rotate(new PointD(1,0,0), XRot -
Math.PI/2, 0, 0, 1);
                pt = CalcView.Rotate(pt, ZRot, rotAxis);
                pt = CalcView.Rotate(pt,
ARot,
axisVector);
                pt.AddVector(1, Center.Coord);
                node.Coord = pt;
            }
        }
    }

```

```

    public void RotateSetVertexCoord(PointD axis, double
rot) {
        foreach (TestNode node in PointFolder.Values) {
            PointD pt;
            node.RelativeCoord =
CalcView.Rotate(node.RelativeCoord, rot, axis);
            pt = node.RelativeCoord.Duplicate();
            pt = CalcView.Rotate(pt,
                                ARot,
                                axisVector);
            pt.AddVector(1, Center.Coord);
            node.Coord = pt;
        }
        public void MoveTo(double x, double y, double z) {
            PointD sub = CalcView.Vector(Center.Coord, x, y,
z);
            Center.MoveTo(x, y, z);
            foreach (TestNode node in PointFolder.Values) {
                node.Coord.AddVector(1, sub);
            }
        }
        private void AddRotation(double k, (double, double)
deltaRot) {
            XRot += k * deltaRot.Item1;
            ZRot += k * deltaRot.Item2;
        }
        public void MoveTo(PointD pt) {
            MoveTo(pt.X, pt.Y, pt.Z);
        }
        public void MakeCube(double x, double y, double z) {
            Center = new TestNode(0, 0, 0);
            ARot = 0; XRot = 0; ZRot = 0;
            OState = ObjectState.unchosen;
            for (int i = 1; i <= 8; i++) {
                double xc, yc, zc;
                if(i <= 4) {xc = x/2; }
                else {xc = -x/2; }
                if(i % 2 == 0) {yc = y/2; }
                else {yc = -y/2; }
                if((i + 1) % 4 <= 1) {zc = z/2; }
                else {zc = -z/2; }
                TestNode tn = new TestNode(xc, yc, zc);
                tn.SetR(Center.Coord);
                PointFolder.Add(i, tn);
            }
            SetVertexCoord();
            //線情報の追加
            Lines.Add((1, 2));
            Lines.Add((1, 3));
            Lines.Add((1, 5));
            Lines.Add((2, 4));
            Lines.Add((2, 6));
            Lines.Add((3, 4));
            Lines.Add((3, 7));
            Lines.Add((4, 8));
            Lines.Add((5, 6));
            Lines.Add((5, 7));
            Lines.Add((6, 8));
            Lines.Add((7, 8));
        }
        private (double, double) CalcRotation(double x, double
y, double z, int vertex) {
            double deltaXRot = 0;
            double rotXBefore =
Math.Atan2(PointFolder[vertex].Coord.Y - Center.Coord.Y,
PointFolder[vertex].Coord.X - Center.Coord.X);
            double rotXAfter = Math.Atan2(y - Center.Coord.Y,
x - Center.Coord.X);
            deltaXRot = rotXAfter - rotXBefore;
            double deltaZRot = 0;
            double rotZBefore =
Math.Atan2(PointFolder[vertex].Coord.Y - Center.Coord.Y,
PointFolder[vertex].Coord.X - Center.Coord.X);
            double rotZAfter = Math.Atan2(y - Center.Coord.Y,
x - Center.Coord.X);
            deltaZRot = rotZAfter - rotZBefore;
            return (deltaXRot, deltaZRot);
        }
    }

```

```

        public void Pull(double x, double y, double z, int vertex) {
            PointD bVector = CalcView.Vector(Center.Coord,
PointFolder[vertex].Coord);
            PointD aVector = new
PointD(PointFolder[vertex].Coord.X - Center.Coord.X + x,
PointFolder[vertex].Coord.Y - Center.Coord.Y + y,
PointFolder[vertex].Coord.Z - Center.Coord.Z + z);
            PointD outerVector =
CalcView.Outerproduction(bVector, aVector);
            double Sin = outerVector.Len / (aVector.Len *
bVector.Len);
            double theta = Math.Asin(Sin);
            PointD newVec = CalcView.Rotate(axisVector,
theta, outerVector);
            XRot = CalcView.XRot(newVec);
            ZRot = CalcView.ZRot(newVec);
            PointD vec = CalcView.Vector(Center.Coord,
x + PointFolder[vertex].Coord.X,
y + PointFolder[vertex].Coord.Y,
z + PointFolder[vertex].Coord.Z);
            double Lall = vec.GetDistance(0, 0, 0);
            double k = (Lall -
PointFolder[vertex].RelativeCoord.Len) / Lall;
            Center.Coord.AddVector(k, vec);
            RotateSetVertexCoord(outerVector, theta);
        }
        private void SetRotation((double, double) deltaRot) {
            XRot = deltaRot.Item1;
            ZRot = deltaRot.Item2;
        }
        public void Pull(PointD pt, int vertex) {
            Pull(pt.X, pt.Y, pt.Z, vertex);
        }
        public List<(int, int)> ClearConnection() {
            List<(int, int)> res = new List<(int, int)>();
            foreach (TestNode node in PointFolder.Values) {
                if (node.NState == NodeState.FIXED) {
                    (int, int) item = node.ClearConnection();
                    res.Add(item);
                }
            }
            return res;
        }
        public void SetConnection(int opponentPartId, int
opponentNodeId, int selfVertexNum) {
            PointFolder[selfVertexNum].SetConnection(opponentPartId,
opponentNodeId);
        }
        public double MaxLength() {
            double max = 0;
            foreach (TestNode node in PointFolder.Values) {
                if (node.R > max) {
                    max = node.R;
                }
            }
            return max;
        }
    }
}

```

PointD.cs

```

namespace FuzzyNodeCS
{
    public class PointD
    {
        public double X;
        public double Y;
        public double Z;
        public double Len { get => GetDistance(0, 0, 0); }
        public PointD(double x = 0, double y = 0, double z = 0) {
            this.X = x;
            this.Y = y;
            this.Z = z;
        }
        public PointD Duplicate() {
            string jsonStr =
Newtonsoft.Json.JsonConvert.SerializeObject(this);

```

```

        PointD res =
JsonConvert.DeserializeObject<PointD>(jsonStr);
        return res;
    }
    public void Trans(PointD pt) {
        Trans(pt.X, pt.Y, pt.Z);
    }
    public void Trans(double x, double y, double z) {
        this.X += x;
        this.Y += y;
        this.Z += z;
    }
    public Point CTransI(double x, double y) {
        int ix, iy;
        ix = (int)Math.Round(this.X + x);
        iy = (int)Math.Round(this.Y + y);
        if (this.Z != 0) {
            throw new NotImplementedException();
        }
        return new Point(ix, iy);
    }
    public void MoveTo(PointD pt) {
        MoveTo(pt.X, pt.Y, pt.Z);
    }
    public void MoveTo(double x, double y, double z) {
        this.X = x;
        this.Y = y;
        this.Z = z;
    }
    public void AddVector(double k, PointD pt) {
        this.X += k * pt.X;
        this.Y += k * pt.Y;
        this.Z += k * pt.Z;
    }
    public void Rotate(PointD pt, double rot) {
        RotateXY(pt.X, pt.Y, rot);
    }
    public void RotateXY(double x, double y, double rot) {
        double x1 = this.X - x;
        double y1 = this.Y - y;
        double rad = rot * Math.PI / 180.0;
        X = Math.Cos(rad) * x1 - Math.Sin(rad) * y1 + x;
        Y = Math.Sin(rad) * x1 + Math.Cos(rad) * y1 + y;
    }
    public double GetDistance(PointD pt) {
        double len = GetDistance(pt.X, pt.Y, pt.Z);
        return len;
    }
    public double GetDistance(double x, double y, double z) {
        double len = Math.Sqrt(Math.Pow(x - this.X, 2)
            + Math.Pow(y - this.Y, 2)
            + Math.Pow(z - this.Z, 2));
        return len;
    }
    public double GetAngleXY(PointD pt) {
        return GetAngleXY(pt.X, pt.Y);
    }
    public double GetAngleXY(double x, double y) {
        double angle = Math.Atan2(this.Y - y, this.X - x);
        return angle;
    }
    public Point ToPointI_XY(){
        int x = (int)Math.Round(X);
        int y = (int)Math.Round(Y);
        return new Point(x, y);
    }
    public override string ToString(){
        return "X:" + X.ToString() + " Y:" + Y.ToString() + "
Z:" + Z.ToString();
    }
}
}

```

EmunAssist.cs

```

namespace FuzzyNodeCS
{
    internal static class EnumAssist

```

```

    {
        internal static string Name(this ModeType tgt) {
            string[] names;
            names = new string[] { "normal", "make", "move",
"rotate", "pull", "fixed pull", "followed pull", "erase" };
            return names[(int)tgt];
        }
        internal static string Name(this ViewPoint tgt) {
            string[] names;
            names = new string[] { "right", "front", "top",
"perspective" };
            return names[(int)tgt];
        }
    }
}

```

ModeType.cs

```

enum ModeType
{
    NEUTRAL,
    MAKE,
    MOVE,
    ROTATE,
    PULL,
    FIXEDPULL,
    FOLLOWEDPULL,
    ERASE,
};

```

MouseAction.cs

```

enum MouseAction
{
    None,
    RightMoving,
    LeftMoving
}

```

NodeState.cs

```

enum NodeState
{
    FREE,
    FIXED,
    CONNECTION,
};

```

ObjectState.cs

```

enum ObjectState
{
    unchosen,
    chosen,
    candidate,
};

```

PointType.cs

```

enum PointType
{
    CENTER,
    VERTEX,
}

```

ViewPoint.cs

```

enum ViewPoint
{
    RIGHT,
    FRONT,
    TOP,
    PERSPECTIVE,
};

```

MainForm.cs

```

namespace FuzzyNodeCS
{
    public partial class MainForm : Form
    {
        private Drawing _Drawing = new Drawing();
        private Geometry _Geometry = new Geometry();
        private RedoUndo _RedoUndo = new RedoUndo();
        ModeType _modeType;
        ViewPoint _viewPoint;
        bool _IsDragging;
        PointD _StartPoint;
        int _PulledVertex;
        bool _MasterPiece;
        public MainForm(bool fresh) {
            InitializeComponent();
            InitForm();
            InitVar();
            _MasterPiece = fresh;
            if (!fresh) {
                _RedoUndo.Init(MC.MO);
                SetRedoUndoButtons();
            }
        }
        private void InitForm()
        {
            _modeType = ModeType.NEUTRAL;
            _viewPoint = ViewPoint.FRONT;
        }
        private void InitVar()
        {
            _IsDragging = false;
            _StartPoint = new PointD(-1, -1);
            _PulledVertex = -1;
        }
        private void SetMasterPiece()
        {
            MasterPieceForm frm = new MasterPieceForm();
            frm.StartPosition =
FormStartPosition.CenterParent;
            frm.ShowDialog();
            if (frm.DialogResult == DialogResult.OK) {
                MC.MO.MasterPiece = frm.part;
            }
            else if (frm.DialogResult == DialogResult.Abort) {
                MC.Exit(true);
            }
            else {
                MessageBox.Show("You have to register a
pice.¥n¥rTo end this app press the quit button", "Caution");
                SetMasterPiece();
            }
        }
        internal void SetProjectName()
        {
            Text = string.Format("プロジェクト名 : {0}",
MC.ProjectNameNoExtension);
        }
        internal void SetSavedDate()
        {
            if (ProjectInfo.IsNew) {
                toolStripStatusLabelDate.Text =
string.Format("LastSaved : ");
            }
            else {
                toolStripStatusLabelDate.Text =
string.Format("LastSaved : {0}",
MC.MO.SavedDate.ToString("yyyy/MM/dd HH:mm"));
            }
        }
        internal void SetStatus()
        {
            SetProjectName();

```

```

                SetSavedDate();
            }
            private void MainForm_Shown(object sender,
EventArgs e) {
                if (_MasterPiece)
                {
                    SetMasterPiece();
                    _RedoUndo.Init(MC.MO);
                    SetRedoUndoButtons();
                }
                _MasterPiece = false;
            }
            private void MainForm_FormClosed(object sender,
FormClosedEventArgs e) {
                MC.Exit(true);
            }
            private void MainForm_FormClosing(object sender,
FormClosingEventArgs e)
            {
            }
            private void toolStripMenuItemNew_Click(object
sender, EventArgs e) {
                try
                {
                    NewFile();
                }
                catch
                {
                    MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
                }
            }
            private void toolStripMenuItemOpen_Click(object
sender, EventArgs e) {
                try
                {
                    OpenFile();
                }
                catch
                {
                    MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
                }
            }
            private void toolStripMenuItemSave_Click(object
sender, EventArgs e) {
                try
                {
                    MC.Save(true);
                }
                catch
                {
                    MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
                }
            }
            private void toolStripMenuItemSaveAs_Click(object
sender, EventArgs e) {
                try
                {
                    MC.Save(false);
                }
                catch
                {
                    MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
                }
            }
            private void toolStripMenuItemUndo_Click(object sender,
EventArgs e) {
                try
                {
                    using (null) {
                        (bool flag, MassObject mo) =
_Redundo.Undo();
                        if (flag) {
                            MC.MO = mo;

```

```

        if (MC.MO.IsSelected == -1) {
ChangeMode(ModeType.NEUTRAL);
        }
        Draw0;
        SetRedoUndoButtons();
    }
}
catch
{
    MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
}

private void toolStripMenuRedo_Click(object sender,
EventArgs e) {
    try
    {
        using (null) {
            (bool flag, MassObject mo) =
_Redundo.Redo0;
            if (flag) {
                MC.MO = mo;
                if (MC.MO.IsSelected == -1)
ChangeMode(ModeType.NEUTRAL);
            }
            Draw0;
            SetRedoUndoButtons();
        }
    }
    catch
    {
        MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
    }
}

private void toolStripMenuMake_Click(object sender,
EventArgs e) {
    try
    {
        ChangeMode(ModeType.MAKE);
        MC.MO.SetSelected(-1);
        Draw0;
    }
    catch
    {
        MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
    }
}

private void toolStripMenuMove_Click(object sender,
EventArgs e) {
    try
    {
        ChangeMode(ModeType.MOVE);
    }
    catch
    {
        MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
    }
}

private void toolStripMenuRotate_Click(object sender,
EventArgs e) {
    try
    {
        ChangeMode(ModeType.ROTATE);
    }
    catch
    {
        MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
    }
}

```

```

private void toolStripMenuPull_Click(object sender,
EventArgs e) {
    try
    {
        ChangeMode(ModeType.PULL);
    }
    catch
    {
        MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
    }
}

private void toolStripMenuFixedPull_Click(object
sender, EventArgs e) {
    try
    {
        ChangeMode(ModeType.FIXEDPULL);
    }
    catch
    {
        MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
    }
}

private void toolStripMenuFollowedPull_Click(object
sender, EventArgs e) {
    try
    {
        ChangeMode(ModeType.FOLLOWEDPULL);
    }
    catch
    {
        MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
    }
}

private void toolStripMenuErase_Click(object sender,
EventArgs e) {
    try
    {
        if (ErasePart0 == 0) {
            Draw0;
        }
    }
    catch
    {
        MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
    }
}

private void toolStripMenuNeutral_Click(object sender,
EventArgs e) {
    try
    {
        ChangeMode(ModeType.NEUTRAL);
        MC.MO.SetSelected(-1);
        Draw0;
    }
    catch
    {
        MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
    }
}

private void MainForm_Resize(object sender,
EventArgs e) {
    Draw0;
}

private void MainPictureBox_MouseDown(object
sender, MouseEventArgs e) {
    switch (e.Button)
    {
        case MouseButton.Right:
            break;
        case MouseButton.Left:

```



```

        MainPictureBox_MouseDownLeft(e.X,
e.Y);
        break;
    default:
        break;
    }
}
private void MainPictureBox_MouseMove(object
sender, MouseEventArgs e) {
    switch (e.Button)
    {
        case MouseButton.Right:
            break;
        case MouseButton.Left:
            MainPictureBox_MouseMoveLeft(e.X,
e.Y);
            break;
    default:
        break;
    }
}
private void MainPictureBox_MouseUp(object sender,
MouseEventArgs e) {
    switch (e.Button)
    {
        case MouseButton.Right:
            break;
        case MouseButton.Left:
            MainPictureBox_MouseUpLeft(e.X, e.Y);
            break;
    default:
        break;
    }
}
private void MainPictureBox_MouseClick(object sender,
MouseEventArgs e) {
    switch (e.Button) {
        case MouseButton.Right:
            EditContextMenuMode();
            contextMenuMode.Show(Cursor.Position.X,
Cursor.Position.Y);
            break;
        case MouseButton.Left:
            MainPictureBox_MouseClickLeft(e.X, e.Y);
            break;
    default:
        break;
    }
}
private void MainForm_Resize(object sender,
EventArgs e) {
    Draw();
}
private void MainPictureBox_MouseDown(object
sender, MouseEventArgs e)
{
    switch (e.Button)
    {
        case MouseButton.Right:
            break;
        case MouseButton.Left:
            MainPictureBox_MouseDownLeft(e.X,
e.Y);
            break;
    default:
        break;
    }
}
private void MainPictureBox_MouseMove(object
sender, MouseEventArgs e)
{
    switch (e.Button)
    {
        case MouseButton.Right:
            break;
        case MouseButton.Left:
            MainPictureBox_MouseMoveLeft(e.X,
e.Y);
            break;
    default:
        break;
    }
}

```

```

        break;
    }
}
private void MainPictureBox_MouseUp(object sender,
MouseEventArgs e) {
    switch (e.Button)
    {
        case MouseButton.Right:
            break;
        case MouseButton.Left:
            MainPictureBox_MouseUpLeft(e.X, e.Y);
            break;
    default:
        break;
    }
}
private void MainPictureBox_MouseClick(object sender,
MouseEventArgs e) {
    switch (e.Button) {
        case MouseButton.Right:
            EditContextMenuMode();
            contextMenuMode.Show(Cursor.Position.X,
Cursor.Position.Y);
            break;
        case MouseButton.Left:
            MainPictureBox_MouseClickLeft(e.X, e.Y);
            break;
    default:
        break;
    }
}
private void MainPictureBox_MouseClickLeft(double x,
double y) {
    PointD ClickedPoint = new PointD(x, y);
    switch (_modeType)
    {
        case ModeType.NEUTRAL:
            ChosePart(ClickedPoint);
            break;
        case ModeType.MAKE:
            MakePart(ClickedPoint);
            Draw();
            break;
    default:
        break;
    }
}
private void MainPictureBox_MouseDownLeft(double
x, double y) {
    PointD ClickedPoint = new PointD(x, y);
    switch (_modeType)
    {
        case ModeType.MOVE:
            if
(_Geometry.ClickCenter(MC.MO.Parts[MC.MO.IsSelected],
ClickedPoint)) {
                _IsDragging = true;
            }
            break;
        case ModeType.ROTATE:
        case ModeType.PULL:
        case ModeType.FIXEDPULL:
        case ModeType.FOLLOWEDPULL:
            (List<int> chosen, bool flag) =
_Geometry.ClickVertex(MC.MO.Parts[MC.MO.IsSelected],
ClickedPoint);
            if (flag) {
                _IsDragging = true;
                _PulledVertex = chosen[0];
            }
            break;
    default:
        break;
    }
}
private void MainPictureBox_MouseMoveLeft(double x,
double y) {
    PointD MovedPoint = new PointD(x, y);
    if (_IsDragging) {
        switch (_modeType)

```

```

    {
        case ModeType.MOVE:
            MovePart(MovedPoint, false);
            break;
        case ModeType.ROTATE:
            RotatePart(MovedPoint,
                _PulledVertex, false);
            break;
        case ModeType.PULL:
            PullPart(MovedPoint, _PulledVertex,
                false);
            break;
        case ModeType.FIXEDPULL:
            FixedPull(MovedPoint, _PulledVertex,
                false);
            break;
        case ModeType.FOLLOWEDPULL:
            FollowedPull(MovedPoint,
                _PulledVertex, false);
            break;
        default: break;
    }
}
private void MainPictureBox_MouseUpLeft(double x,
double y){
    PointD MovedPoint = new PointD(x, y);
    if (!_IsDragging) {
        switch (_modeType)
        {
            case ModeType.MOVE:
                MovePart(MovedPoint, true);
                break;
            case ModeType.ROTATE:
                RotatePart(MovedPoint,
                    _PulledVertex, true);
                break;
            case ModeType.PULL:
                PullPart(MovedPoint, _PulledVertex,
                    true);
                break;
            case ModeType.FIXEDPULL:
                FixedPull(MovedPoint, _PulledVertex,
                    true);
                break;
            case ModeType.FOLLOWEDPULL:
                FollowedPull(MovedPoint,
                    _PulledVertex, true);
                break;
            default: break;
        }
    }
    InitVar();
}
private void MakePart(PointD pt) {
    Part2D part = MC.MO.MasterPiece.Duplicate();
    part.MoveTo(pt);
    int newIndex;
    if (MC.MO.Parts.Keys.Count() > 0) {
        newIndex = MC.MO.Parts.Keys.Max() + 1;
    }
    else{
        newIndex = 0;
    }
    MC.MO.Parts.Add(newIndex, part);
    _RedoUndo.Add(MC.MO);
    SetRedoUndoButtons();
}
private void MovePart(PointD pt, bool end = false) {
    MC.MO.Parts[MC.MO.IsSelected].MoveTo(pt);
    _Geometry.FindConnection(MC.MO,
MC.MO.IsSelected, true);
    Draw();
    if (end) {
        _RedoUndo.Add(MC.MO);
        SetRedoUndoButtons();
    }
}
private void RotatePart(PointD pt,int vertex, bool end =
false) {

```

```

        MC.MO.Parts[MC.MO.IsSelected].Rotate(pt,
vertex);
        _Geometry.FindConnection(MC.MO,
MC.MO.IsSelected, true);
        Draw();
        if (end) {
            _RedoUndo.Add(MC.MO);
            SetRedoUndoButtons();
        }
    }
    private void PullPart(PointD pt, int vertex, bool end =
false) {
        MC.MO.Parts[MC.MO.IsSelected].Pull(pt, vertex);
        _Geometry.FindConnection(MC.MO,
MC.MO.IsSelected, true);
        Draw();
        if (end) {
            _RedoUndo.Add(MC.MO);
            SetRedoUndoButtons();
        }
    }
    private void FixedPull(PointD pt, int vertex, bool end =
false)
    {
        _Geometry.FixedPull(MC.MO, pt, vertex);
        _Geometry.FindConnection(MC.MO,
MC.MO.IsSelected, false);
        Draw();
        if (end)
        {
            _RedoUndo.Add(MC.MO);
            SetRedoUndoButtons();
        }
    }
    private void FollowedPull(PointD pt, int vertex, bool
end = false)
    {
        Debug.WriteLine("FollowedPull");
        _Geometry.FollowedPullTest(MC.MO, pt, vertex);
        _Geometry.FindConnection(MC.MO,
MC.MO.IsSelected, false);
        Draw();
        if (end)
        {
            _RedoUndo.Add(MC.MO);
            SetRedoUndoButtons();
        }
    }
    private int ErasePart(){
        int res = -1;
        if (MC.MO.IsSelected != -1) {
            res = 0;
            _Geometry.Disconnect(MC.MO,
MC.MO.IsSelected);
            MC.MO.Parts.Remove(MC.MO.IsSelected);
            MC.MO.SetSelected(-1);
            _RedoUndo.Add(MC.MO);
            SetRedoUndoButtons();
            ChangeMode(ModeType.NEUTRAL); /
        }
        return res;
    }
    internal void Draw() {
        MainPictureBox.Image = null;
        GC.Collect();
        if (MainPictureBox.Size.Width *
MainPictureBox.Height > 0) {
            var ni = new
Bitmap(MainPictureBox.Size.Width, MainPictureBox.Height);
            _Drawing.DrawParts(MC.MO, ni);
            SuspendLayout();
            MainPictureBox.Image = ni;
            ResumeLayout();
        }
    }
    private void NewFile(){
        MC.OpenningNewExe = true;
        OpenFuzzyNodeApp();
    }
    private void OpenFile(){
        MC.OpenningNewExe = true;
    }
}

```

```

        OpenFileDialog ofd = new OpenFileDialog
        {
            Filter = "fnode(.fnode) | *.fnode | All Files
(*.*)|*.*",
            FilterIndex = 1,
            Title = "fnode ファイルを選択してください",
            RestoreDirectory = true
        };
        if (ofd.ShowDialog() == DialogResult.OK) {
            if (Path.GetExtension(ofd.FileName) ==
MC.fileExtension) {
                OpenFuzzyNodeApp(" ¥" + ofd.FileName
+ " ¥");
            }
            else {
                MessageBox.Show("正しい形式のファイル
を選択してください。");
            }
        }
        private int OpenFuzzyNodeApp(string path = "") {
            MC.Exit(true);
            if (MC.OpenningNewExe) {
                string ExePath =
Environment.GetCommandLineArgs()[0];
                System.Diagnostics.Process.Start(ExePath,
path);
                MC.OpenningNewExe = false;
                return 0;
            }
            else {
                return -1;
            }
        }
        private void SetRedoUndoButtons() {
            toolStripMenuUndo.Enabled =
(!_RedoUndo.CurrentIndex > 0);
            toolStripMenuRedo.Enabled =
(!_RedoUndo.CurrentIndex + 1 < _RedoUndo.History.Count());
        }
        internal void ChangeMode(ModeType mt) {
            _modeType = mt;
            currentModeLabel.Text =
EnumAssist.Name(_modeType);
        }
        private void EditContextMenuMode() {
            toolStripMenuMake.Enabled = (_modeType !=
ModeType.MAKE);
            toolStripMenuMove.Enabled =
(MC.MO.IsSelected != -1);
            toolStripMenuRotate.Enabled =
(MC.MO.IsSelected != -1);
            toolStripMenuPull.Enabled =
(MC.MO.IsSelected != -1);
            toolStripMenuFixedPull.Enabled =
(MC.MO.IsSelected != -1);
            toolStripMenuFollowedPull.Enabled =
(MC.MO.IsSelected != -1);
            toolStripMenuErase.Enabled =
(MC.MO.IsSelected != -1);
            toolStripMenuNeutral.Enabled = (_modeType !=
ModeType.NEUTRAL);
        }
        private void ChosePart(PointD pt) {
            List<int> selectedParts;
            bool flag;
            (selectedParts, flag) =
_Geometry.ClickCenter(MC.MO, pt);
            MC.MO.SetSelected(-1);
            Draw();
            if (flag) {
                ChosePartFromMenu(selectedParts);
            }
        }
        private void ChosePartFromMenu(List<int> sList) {
            if (sList.Count() > 1) {
                ContextMenuStrip ChoseMenu = new
ContextMenuStrip();
                foreach (int index in sList) {
                    ToolStripMenuItem stripMenuItem = new
ToolStripMenuItem();

```

```

                    stripMenuItem.Text = "part " +
index.ToString();
                    stripMenuItem.Tag = index;
                    stripMenuItem.Click +=
MenuItem_Clicked;
                    stripMenuItem.MouseEnter +=
MenuItem_MouseEnter;
                    stripMenuItem.MouseLeave +=
MenuItem_MouseLeave;
                    ChoseMenu.Items.Add(stripMenuItem);
                }
                ChoseMenu.Show(Cursor.Position.X,
Cursor.Position.Y);
            }
            else {
                MC.MO.SetSelected(sList[0]);
                Draw();
            }
        }
        private void MenuItem_Clicked(object sender,
EventArgs e) {
            try
            {
                ((ToolStripMenuItem)sender).MouseEnter -=
MenuItem_MouseEnter;
                ((ToolStripMenuItem)sender).MouseLeave -=
MenuItem_MouseLeave;
                MC.MO.SetSelected(((ToolStripMenuItem)sender).Tag);
                Draw();
            }
            catch
            {
                MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
            }
        }
        private void MenuItem_MouseEnter(object sender,
EventArgs e) {
            try
            {
                MC.MO.SetCandidate(((ToolStripMenuItem)sender).Tag);
                Draw();
            }
            catch
            {
                MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
            }
        }
        private void MenuItem_MouseLeave(object sender,
EventArgs e) {
            try
            {
                MC.MO.SetCandidate(-1);
                Draw();
            }
            catch
            {
                MessageBox.Show(string.Format("{0}",
MethodBase.GetCurrentMethod().Name), "ERROR");
            }
        }
        internal void testestest(string str1, string str2) {
            testLabel1.Text = str1;
            testLabel2.Text = str2;
        }
        private void toolStripMenuIHelp_Click(object sender,
EventArgs e) {
            TestForm frm = new TestForm();
            frm.ShowDialog();
        }
    }
}

```

MasterPiece.cs

```

namespace FuzzyNodeCS.Frm
{
    public partial class MasterPieceForm : Form
    {
        /// </summary>
        private int vertexNum;
        private List<PointD> ptList;
        private Part2D fPart;
        internal Part2D part;
        private Drawing _Drawing = new Drawing();
        private Geometry _Geometry = new Geometry();
        private bool freeRegister, regRegister;
        int mode = 0;
        public MasterPieceForm()
        {
            InitializeComponent();
            ptList = new List<PointD>();
            radioButtonRgr.Checked = true;
            freeRegister = false;
            regRegister = false;
        }
        private void rbtn_CheckedChanged(object sender,
        EventArgs e) {
            if (radioButtonFree.Checked) {
                panelFree.Visible = true;
                panelReg.Visible = false;
                buttonRegister.Enabled = freeRegister;
            }
            else if (radioButtonRgr.Checked) {
                panelFree.Visible = false;
                panelReg.Visible = true;
                buttonRegister.Enabled = regRegister;
            }
        }
        private void textBoxRegVertex_TextChanged(object
        sender, EventArgs e) {
            if (textBoxRegVertex.Text == "") {
                regRegister = false;
            }
            else {
                if (int.TryParse(textBoxRegVertex.Text, out int
                i)) {
                    if (i <= 2 || i >= 101) {
                        textBoxRegVertex.Text = "";
                        regRegister = false;
                    }
                    else {
                        vertexNum = i;
                        regRegister = true;
                    }
                }
                else {
                    textBoxRegVertex.Text = "";
                    regRegister = false;
                }
            }
            buttonRegister.Enabled = regRegister;
        }
        private void buttonRegister_Click(object sender,
        EventArgs e) {
            if (radioButtonRgr.Checked) {
                part = new Part2D(new PointD(200, 200),
                vertexNum, 50);
            }
            else if (radioButtonFree.Checked) {
                part = fPart;
            }
        }
        private void partBox_MouseClick(object sender,
        MouseEventArgs e) {
            switch (e.Button) {
                case MouseButtons.Right:
                    if (mode == 0 && ptList.Count() >= 3)
                    {
                        contextMenuStripJoin.Show(Cursor.Position.X,
                        Cursor.Position.Y);
                    }
                    break;
                case MouseButtons.Left:
                    PointD ClickedPoint = new PointD(e.X,
                    e.Y);
                    if (mode == 0) {

```

```

                        ptList.Add(ClickedPoint);
                        DrawPoints();
                    }
                }
            }
            else if (mode == 100) {
                List<int> selectedVertexes;
                bool flag;
                (selectedVertexes, flag) =
                _Geometry.ClickVertex(fPart, ClickedPoint);
                if (flag) {
                    if (fPart.PointFolder[selectedVertexes[0]].NState ==
                    NodeState.FREE) {
                        fPart.PointFolder[selectedVertexes[0]].NState =
                        NodeState.CONNECTION;
                    }
                    else if
                    (fPart.PointFolder[selectedVertexes[0]].NState ==
                    NodeState.CONNECTION) {
                        fPart.PointFolder[selectedVertexes[0]].NState =
                        NodeState.FREE;
                    }
                }
                if (fPart.FreeNum() > 2) {
                    freeRegister = true;
                }
                else {
                    freeRegister = false;
                }
                buttonRegister.Enabled =
                freeRegister;
                DrawPart();
            }
        }
        break;
        default:
            break;
        }
    }
    private void ToolStripMenuJoin_Click(object sender,
    EventArgs e) {
        mode = 100;
        fPart = new Part2D(ptList);
        DrawPart();
    }
    private void DrawPoints() {
        partBox.Image = null;
        GC.Collect();
        if (partBox.Size.Width * partBox.Height > 0) {
            var ni = new Bitmap(partBox.Size.Width,
            partBox.Height);
            ni;
            SuspendLayout();
            partBox.Image = ni;
            ResumeLayout();
        }
    }
    private void DrawPart() {
        partBox.Image = null;
        GC.Collect();
        if (partBox.Size.Width * partBox.Height > 0) {
            var ni = new Bitmap(partBox.Size.Width,
            partBox.Height);
            _Drawing.DrawPart(fPart, ni);
            SuspendLayout();
            partBox.Image = ni;
            ResumeLayout();
        }
    }
}

```

TestForm.cs

```

namespace FuzzyNodeCS.Frm
{
    public partial class TestForm : Form
    {

```

```

private Drawing _Drawing = new Drawing();
private Observer _Observer;
private List<TestPart> _parts;
private bool _isLeftPressed;
private bool _isRightPressed;
private bool _isForwardPressed;
private bool _isBackwardPressed;
private bool _isUpPressed;
private bool _isDownPressed;
private bool _isShiftPressed;
private double _Speed = 1;
private bool _isAltPressed;
private MouseAction _mouseAction =
MouseAction.None;
private int _chosenNode;
private int _chosenIndex = -1;
private Point _startLoc;
public TestForm() {
    InitializeComponent();
    Init();
}
private void Init()
{
    movePanel.Visible = false;
    _Observer = new Observer(CalcNewView);
    _parts = new List<TestPart>();
    TestPart cube1 = new TestPart();
    cube1.MakeCube(100, 100, 100);
    cube1.MoveTo(0, 0, 0);
    cube1.PointFolder[4].ConnectedPartId = 1;
    cube1.PointFolder[4].ConnectedNodeId = 5;
    _parts.Add(cube1);
    TestPart cube2 = new TestPart();
    cube2.MakeCube(100, 100, 100);
    cube2.MoveTo(105, 105, 105);
    cube1.PointFolder[5].ConnectedPartId = 0;
    cube1.PointFolder[5].ConnectedNodeId = 4;
    cube1.PointFolder[4].ConnectedPartId = 2;
    cube1.PointFolder[4].ConnectedNodeId = 5;
    _parts.Add(cube2);
    TestPart cube3 = new TestPart();
    cube3.MakeCube(100, 100, 100);
    cube3.MoveTo(210, 210, 210);
    cube1.PointFolder[5].ConnectedPartId = 1;
    cube1.PointFolder[5].ConnectedNodeId = 4;
    _parts.Add(cube3);
CalcNewView();
}
#region PictureBox Events
private void testPictureBox_MouseDown(object sender,
MouseEventArgs e) {
    if (_mouseAction == MouseAction.None) {
        switch (e.Button)
        {
            case MouseButtons.Right:
                MouseDownRight(sender, e);
                break;
            case MouseButtons.Left: break;
            default: break;
        }
    }
}
private void MouseDownRight(object sender,
MouseEventArgs e) {
    if (_isShiftPressed || _isAltPressed) {
        _mouseAction = MouseAction.RightMoving;
        _startLoc = new Point(e.X, e.Y);
    }
}
private void testPictureBox_MouseMove(object sender,
MouseEventArgs e) {
    switch (e.Button)
    {
        case MouseButtons.Right:
            if (_mouseAction ==
MouseAction.RightMoving) {
                MouseMoveRight(sender, e);
            }
            break;
        case MouseButtons.Left: break;
        default: break;
    }
}
private void MouseMoveRight(object sender,
MouseEventArgs e)
{
    if (_isShiftPressed) {
        _Observer.TransPosition(-e.X + _startLoc.X, -
e.Y + _startLoc.Y);
        _startLoc = new Point(e.X, e.Y);
        Draw();
    }
    else if (_isAltPressed) {
        _Observer.ChangeView(-e.X + _startLoc.X, -e.Y
+ _startLoc.Y);
        _startLoc = new Point(e.X, e.Y);
        Draw();
    }
}
}
private void testPictureBox_MouseUp(object sender,
MouseEventArgs e) {
    switch (e.Button)
    {
        case MouseButtons.Right:
            if (_mouseAction ==
MouseAction.RightMoving) {
                MouseUpRight(sender, e);
            }
            break;
        case MouseButtons.Left: break;
        default: break;
    }
}
private void MouseUpRight(object sender,
MouseEventArgs e) {
    _mouseAction = MouseAction.None;
}
private void TestForm_KeyDown(object sender,
KeyEventArgs e) {
    try
    {
        if (!_isShiftPressed && !_isAltPressed) {
            if (e.Shift) {
                _isShiftPressed = true;
                _Speed = 3;
            }
            if (e.Alt) {
                _isAltPressed = true;
            }
        }
        if (_isRightPressed) {
            if (e.KeyCode == Keys.D) {
                _isRightPressed = true;
            }
        }
        if (_isLeftPressed) {
            if (e.KeyCode == Keys.A) {
                _isLeftPressed = true;
            }
        }
        if (_isForwardPressed) {
            if (e.KeyCode == Keys.W) {
                _isForwardPressed = true;
            }
        }
        if (_isBackwardPressed) {
            if (e.KeyCode == Keys.S) {
                _isBackwardPressed = true;
            }
        }
        if (_isUpPressed) {
            if (e.KeyCode == Keys.E) {
                _isUpPressed = true;
            }
        }
        if (_isDownPressed) {
            if (e.KeyCode == Keys.C) {
                _isDownPressed = true;
            }
        }
    }
}

```

```

    }
    if (_isRightPressed)
    {
        _Observer.MovePosition("right", _Speed);
    }
    if (_isLeftPressed) {
        _Observer.MovePosition("left", _Speed);
    }
    if (_isForwardPressed) {
        _Observer.MovePosition("forward",
_Speed);
    }
    if (_isBackwardPressed) {
        _Observer.MovePosition("backward",
_Speed);
    }
    if (_isUpPressed) {
        _Observer.MovePosition("up", _Speed);
    }
    if (_isDownPressed) {
        _Observer.MovePosition("down", _Speed);
    }
}
catch {
    Debug.WriteLine(string.Format("{0}: ERROR",
MethodBase.GetCurrentMethod().Name));
}
}
private void TestForm_KeyUp(object sender,
KeyEventArgs e) {
    try{
        if (e.Shift)
        {
            _isShiftPressed = false;
            _Speed = 1;
        }
        if (e.Alt) {
            _isAltPressed = false;
        }
        if (e.KeyCode == Keys.D) {
            _isRightPressed = false;
        }
        if (e.KeyCode == Keys.A) {
            _isLeftPressed = false;
        }
        if (e.KeyCode == Keys.W) {
            _isForwardPressed = false;
        }
        if (e.KeyCode == Keys.S) {
            _isBackwardPressed = false;
        }
        if (e.KeyCode == Keys.E) {
            _isUpPressed = false;
        }
        if (e.KeyCode == Keys.C) {
            _isDownPressed = false;
        }
    }
    catch
    {
        Debug.WriteLine(string.Format("{0}: ERROR",
MethodBase.GetCurrentMethod().Name));
    }
}
private void TestForm_Deactivate(object sender,
EventArgs e)
{
    try
    {
        if (_isShiftPressed) {
            _isShiftPressed = false;
            _Speed = 1;
        }
        if (_isAltPressed) {
            _isAltPressed = false;
        }
        if (_isRightPressed)
        {
            _isRightPressed = false;

```

```

    }
    if (_isLeftPressed) {
        _isLeftPressed = false;
    }
    if (_isForwardPressed) {
        _isForwardPressed = false;
    }
    if (_isBackwardPressed) {
        _isBackwardPressed = false;
    }
    if (_isUpPressed) {
        _isUpPressed = false;
    }
    if (_isDownPressed) {
        _isDownPressed = false;
    }
}
catch {
    Debug.WriteLine(string.Format("{0}: ERROR",
MethodBase.GetCurrentMethod().Name));
}
}
private void TestForm_Resize(object sender, EventArgs e) {
    if (_parts != null) {
        Draw();
    }
}
private void CalcNewView()
{
    foreach (TestPart part in _parts) {
        foreach (TestNode node in
part.PointFolder.Values) {
            node.TransCoord =
CalcView.ChangeToPlane(_Observer, node.Coord);
        }
    }
    Draw();
}
private void Draw()
{
    try
    {
        testPictureBox.Image = null;
        GC.Collect();
        if (testPictureBox.Size.Width *
testPictureBox.Height > 0) {
            var ni = new
Bitmap(testPictureBox.Size.Width, testPictureBox.Height);
            _Drawing.TestDrawParts(ni, _parts);
            SuspendLayout();
            testPictureBox.Image = ni;
            ResumeLayout();
        }
    }
    catch
    {
        Debug.WriteLine(string.Format("{0}: ERROR",
MethodBase.GetCurrentMethod().Name));
    }
}
private void testPictureBox_MouseClick(object sender,
MouseEventArgs e) {
    switch (e.Button)
    {
        case MouseButtons.Left:
            double x = e.X -
testPictureBox.Image.Size.Width / 2;
            double y = e.Y -
testPictureBox.Image.Size.Height / 2;
            List<int> res;
            res = ModelHit(_parts, x, y);
            Draw();
            break;
        case MouseButtons.Right: break;
        case MouseButtons.Middle: break;
    }
}
private List<int> ModelHit(List<TestPart> parts,
double x, double y) {
    List<int> res = new List<int>();
    if (cPullBtn.Text == "start pull") {
        chosenPartLbl.Text = "";

```

```

        chosenVertexLbl.Text = "";
        foreach (TestPart part in parts) {
            part.OSState = ObjectState.unchosen;
            foreach (KeyValuePair<int, TestNode>
pair in part.PointFolder) {
                TestNode node = pair.Value;
                node.OS = ObjectState.unchosen;
            }
            foreach (TestPart part in parts) {
                foreach (KeyValuePair<int, TestNode>
pair in part.PointFolder) {
                    TestNode node = pair.Value;
                    if (node.TransCoord.GetDistance(x, y,
0) < MC.EffectiveClick) {
                        node.OS = ObjectState.chosen;
                        _chosenNode = pair.Key;
                        _chosenIndex = parts.IndexOf(part);
                        chosenVertexLbl.Text = pair.Key.ToString();

                        chosenPartLbl.Text = _chosenIndex.ToString();
                        res.Add(parts.IndexOf(part));
                        part.OSState = ObjectState.chosen;
                        break;
                    }
                }
            }
        }
        return res;
    }
    private void cPullBtn_Click(object sender, EventArgs e) {
        if (cPullBtn.Text == "start pull" && _chosenIndex > -1) {
            movePanel.Visible = true;
            cPullBtn.Text = "end pull";
        }
        else if (cPullBtn.Text == "end pull"){
            movePanel.Visible = false;
            cPullBtn.Text = "start pull";
        }
    }

    private void moveBtn_Click(object sender, EventArgs e)
    {
        _parts[_chosenIndex].Pull(10, 10, 10,
        _chosenNode);
        CalcNewView();
    }

    private async void button1_Click(object sender,
EventArgs e) {
        Geometry geo = new Geometry();
        for (int k = 0; k < 3020 * 2; k++)
        {
            PointD pt;
            int i = k % 3020;
            if (i < 100){
                pt = new PointD(0, -4, 4);
            }
            else if (i < 170) {
                pt = new PointD(-3, -5, 0);
            }
            else if (i < 270) {
                pt = new PointD(-2, 0, 0);
            }
            else if (i < 470){
                pt = new PointD(2, 0, -4);
            }
            else if (i < 720) {
                pt = new PointD(-2, 6, 4);
            }
            else if (i < 820) {
                pt = new PointD(2, 0, 0);
            }
            else if (i < 1095) {
                pt = new PointD(2, 0, -4);
            }
            else if (i < 1220) {
                pt = new PointD(-2, -6, 4);
            }
            else if (i < 1320) {
                pt = new PointD(0, -6, 1);
            }
            else if (i < 1920) {
                double loop = 600;
                double r = i - 1320;
                double t = 2 * Math.PI / loop;
                pt = new PointD(4 * Math.Cos(t * r +
Math.PI / 2), 10 * Math.Sin(t * r), 4 * Math.Cos(t * r));
            }
            else if (i < 2020){
                pt = new PointD(0, 6, -1);
            }
            else if (i < 2220) {
                pt = new PointD(0, 3, -3);
            }
            else if (i < 2820)
            {
                double r = i - 2220;
                if (Math.Floor(r / 100) % 2 == 0) {
                    pt = new PointD(-1, -2, 12);
                }
                else {
                    pt = new PointD(1, -2, -12);
                }
            }
            else if (i < 3020) {
                pt = new PointD(0, 3, 3);
            }
        }
        else {
            throw new NotImplementedException();
        }
        geo.FollowedPull3D(_parts, pt, 0, 5);
        CalcNewView();
        await Task.Delay(1);
    }
}
}
}
}
}
}

```