Kaleidotransformation

－リンク機構による可変構造の体系化と Kaleidocycle の研究－

－ Systematization of Variable Structure System by Link Mechanisms and Deployment of Kaleidocycle －

桑原遼介

Ryohsuke KUWABARA

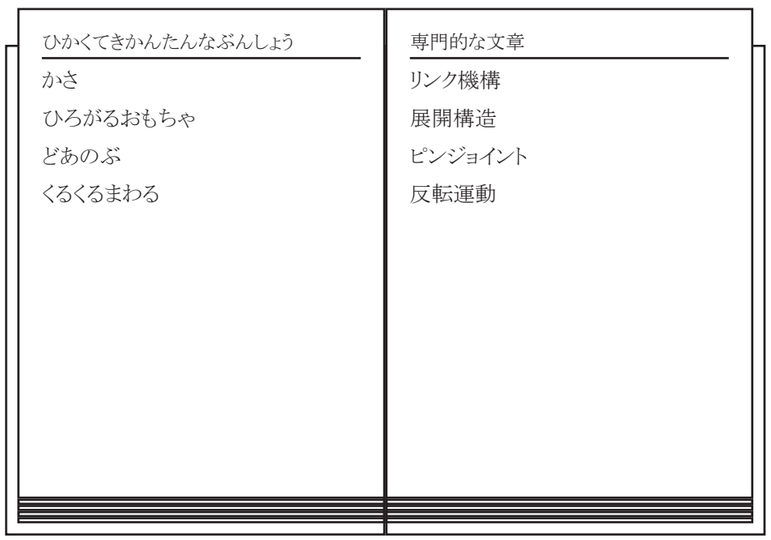
How to Read

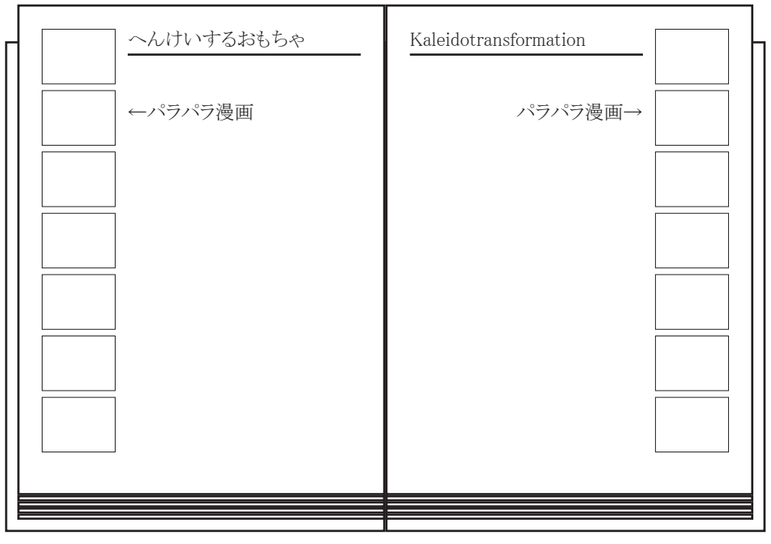
本の書籍は、左右のページでそれぞれ「出来るだけ一般的に知られている言葉を使った文章」と「出来るだけ専門的な用語を使った文章」の二種類で構成されている。左側が日本語、右側が英語という同時通訳本ではない。

「リンク機構」「展開構造」「ピンジョイント」「反転運動」等の単語が分かる方は右のページを読んでいただき、分からない方は左のページを読みながら、右のページを眺めていただけたらと思う。

著者の研究を少しでも多くの方に理解いただけたらと思い、こういった構成をとることにした。

また後半から、ページ両端部に Kaleidotransformation の変形をパラパラ漫画式で載せている。解説と共にその動き方を見ていただけたらと思う。





目次

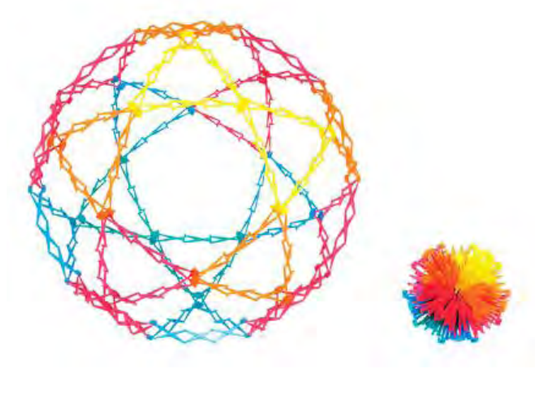
Contents

1．はじめに

リンク機構という言葉では一般には聞きなれない言葉であるが、身近にはたくさんのリンク機構であふれている。例えば傘であったり、パイプ椅子、車のワイパーなどの簡単なものから、より複雑なものだとおもちゃのスイッチピッチやホバーマンスフィア、テオ・ヤンセンのミニビーストなどは何処かで一度は目にしたものがあるのではないだろうか。

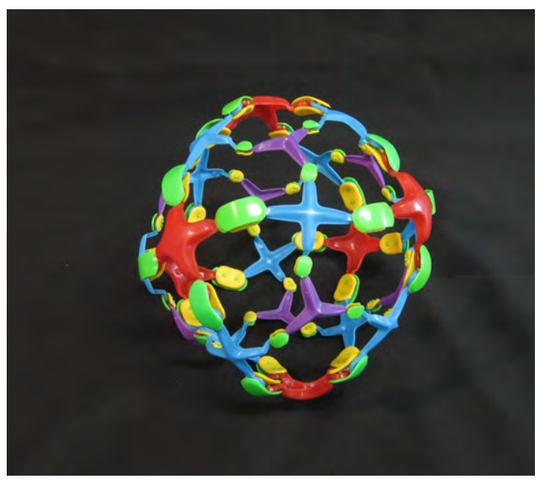
本研究は後者の、より洗練された仕組みを読み解くことと、カライドサクルと呼ばれるくるくる回って絵柄が変わっていく玩具について調べ、発展、展開させたものをまとめたものである。

スイッチピッチ



ホバーマンスフィア

テオ・ヤンセンのミニビースト



ひろがる玩具

1．Introduction

リンク機構による可変構造とは、シザーズ型展開構造[1)]を代表とする形態変化を可能とする構造である。あえて「展開構造[2)]」ではなく「可変構造」としたのは、その体系化を図るときに伸縮だけではなく形態変化という視点から見ることによって、より包括的に展開構造を取り上げそこからの発展が期待出来ると考えたからである。

展開構造に関する研究はE. P. Pineroの"Three Dimensional Reticular Structure[3)]"を初めとしてRichard Buckminster Fullerの"Vector Equilibrium[4)]"(Jitter bug)であったり、現在ではKoryo MIURAの"MIURA-ORI[5)]"を使った建築事例やChuck Hobermanの"Iris Dome[6)]"など様々な形で数多くなされている。近年ではドームの施工方法として用いられる、Mamoru KAWAGUCHIの"Pantadome erection[7)]"も注目を集めている。それらを、構造的視点から総括的にまとめられているものはS. Pellegrinoによる"Deployable Structures[8)]"やYan Chenによる論文"Design of Structural Mechanisms[9)]"などがあるが、その構成方法を体系的にまとめられたものは見られない。

また、Paul Schatzが発明した"Invertible Cube[10)]"あるいはDoris Schattschneideによる"M. C. Escher Kaleidocycles[11)]"で紹介される"Kaleidocycle"は現在そこからの大きな発展、展開が見られない。

本研究は、これらの対象について可動部分（ジョイント部分）の相互関係を分析し、分類することによって体系的にまとめると同時に"Kaleidocycle"について、発展、展開の可能性を探ることである。

S. Pellegrino, "Deployable Structures", 2001.

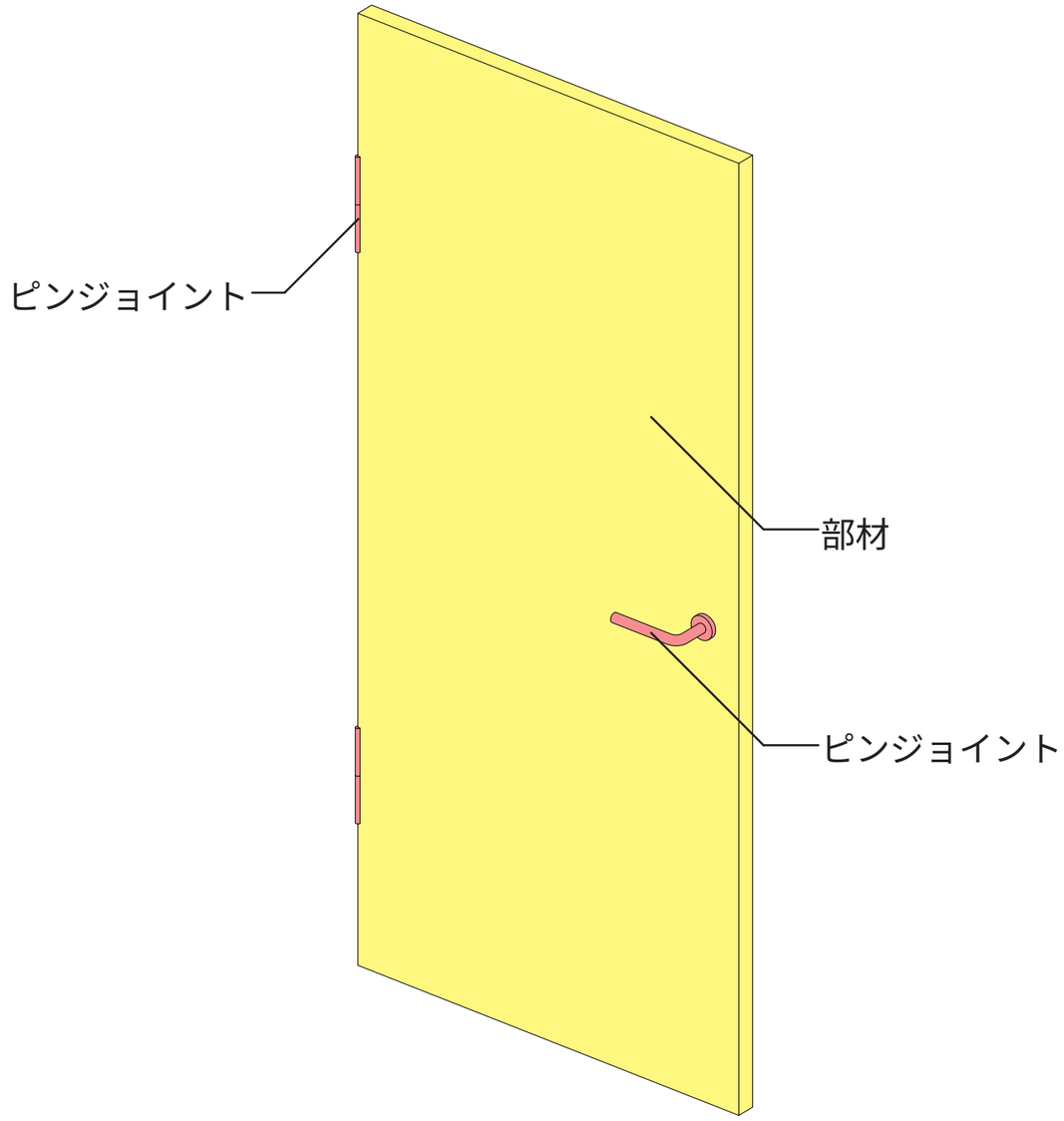
Doris Schattschneide, "M. C. Escher Kaleidocycles", 1977.

動くものの仕組みの分類

Link Mechanisms

２．動くものの仕組みの分類

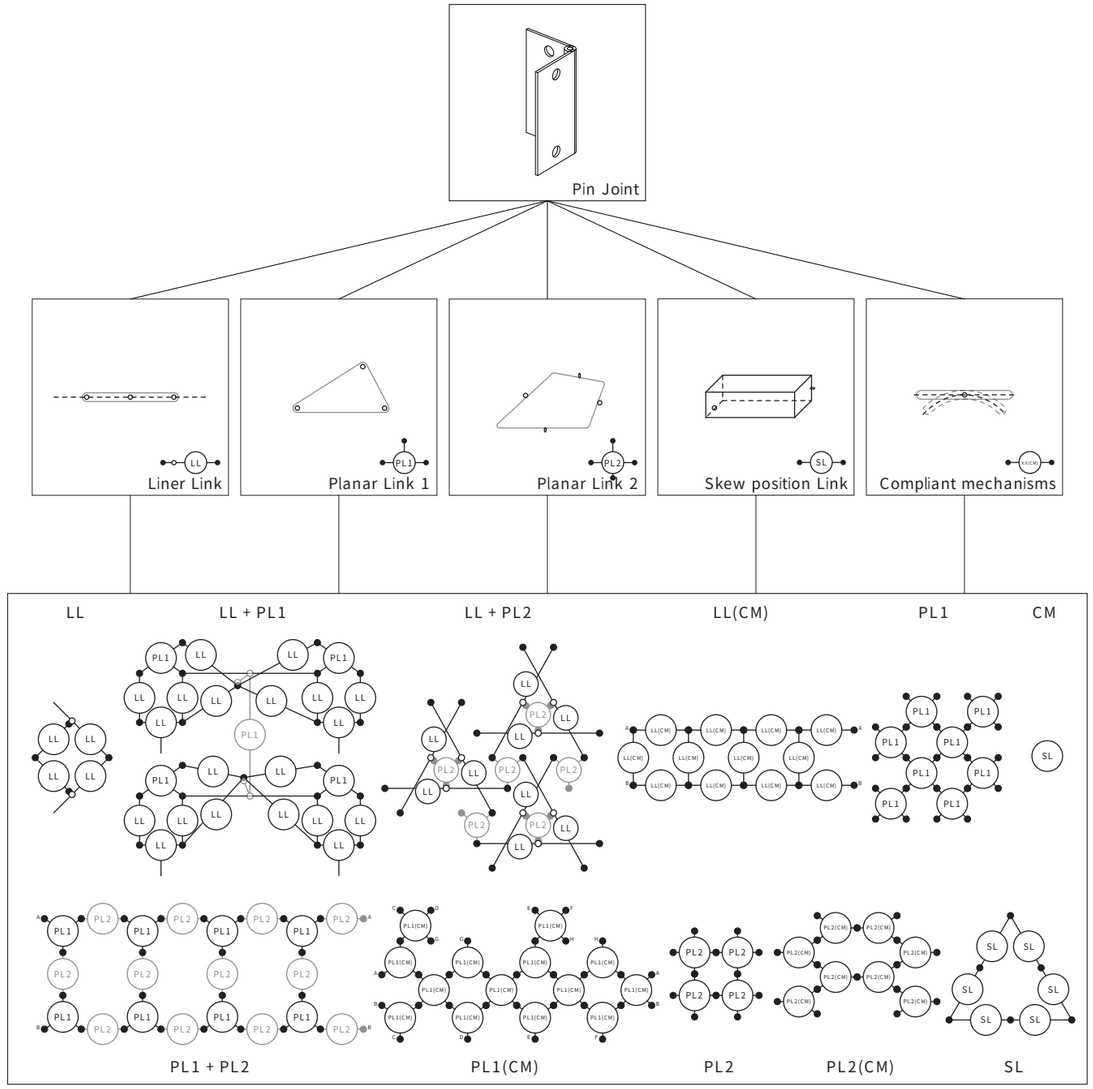
リンク機構の「ピンジョイント」と言われても聞きなれないかもしれないが、身近なものとしては蝶番（ドアの可動部）やドアノブなどの動きがそれにあたる。その部材を種類分けし、その組み合わせ方でリンク機構による動く仕組みを分類した。



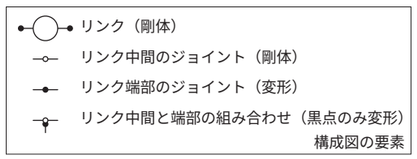
2．Link Mechanisms

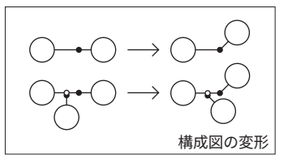
本研究ではリンク機構のジョイントの種類としてピン、スライダー、ボールという三つ存在する中で、ピンジョイントのみにフォーカスを当ててまとめている。それは、ボールジョイントについてはピンジョイントを複数組み合わせたものとして解釈可能なこと、スライダーについては一般的に力の伝達の方向を変えるもの、あるいは変形に安定性を与えるものとして使われることが多く構成としてあまり使われていないことからである。

リンク機構による可変構造を分類する方法としてまずリンク単体における可動部の構成を5つに分類し、それらを組み合わせることによって11個の事例に分類した。



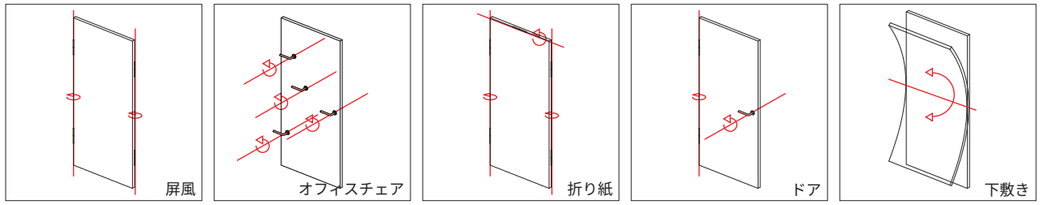
	LL	PL1	PL2	SL	CM
Liner Link	①	②	③		④
Planar Link 1		⑤	⑥		⑦
Planar Link 2			⑧		⑨
Skew position Link				⑩	
Compliant mechanisms					⑪





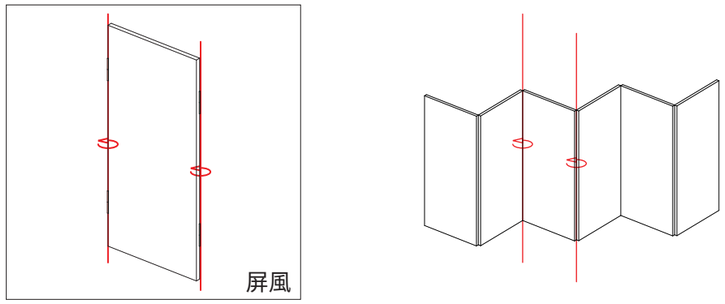
２－１．部材の種類

リンク部材を、1．屏風（扉の両端に蝶番が付いたもの）2．オフィスチェア（扉の面にドアノブが複数付いたもの）3．折り紙（扉の複数の辺に蝶番が付いたもの）4．ドア（扉にドアノブと蝶番が付いたもの）5．下敷き（扉がやわらかくて変形するもの）の5種類に分類する。



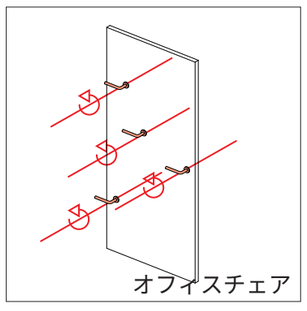
１．屏風

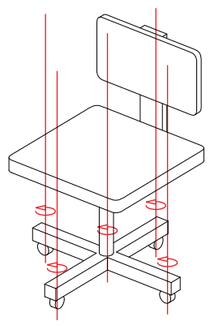
扉の両端に蝶番が付いた、屏風、あるいはマジックハンドのような状態。



２．オフィスチェア

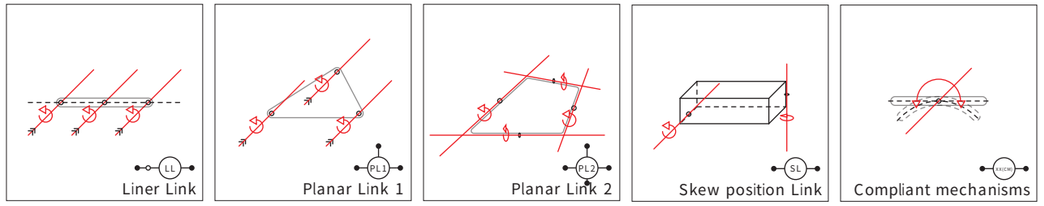
扉の面にドアノブが複数付いたオフィスチェア回転部分の関係のような状態。





2－1．Type of Link

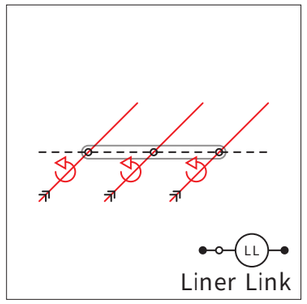
「可変構造」のリンク単体における可動部分の構成を大きく分けて、1．Liner Link（線のピンジョイント）2．Planar Link 1（面Ⅰのピンジョイント）3．Planar Link 2（面Ⅱのピンジョイント）4．Skew position Link（ねじれのピンジョイント）5．Compliant mechanisms（リンクの弾性変形）の5種類に分類する。これらの組み合わせによって「可変構造」は分類出来ると考えた。

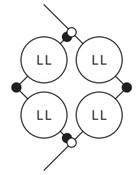


1．Liner Link (LL)

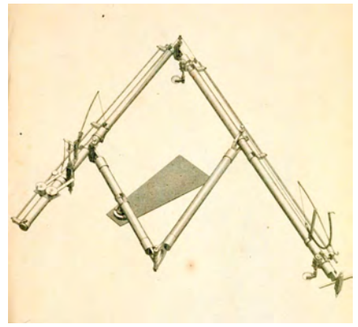
1つのリンク内に2つ以上のピンジョイントが一直線上に並び、そのジョイントがすべて平行な状態。あるいは、1つのリンク内に2つ以上のピンジョイントが同一平面上に存在し、全て平行な状態。

例：Pantograph[12],





Pantograph

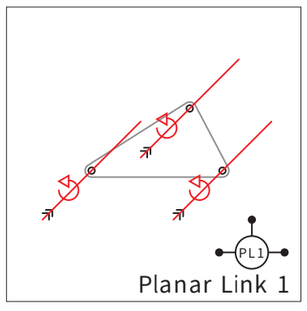


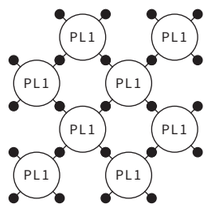
"Pantograph"

2．Planar Link 1 (PL1)

1つのリンク内に3つ以上のピンジョイントが直線状ではない位置に存在し、そのジョイントがすべて平行な状態。

例：Tile magic[13]



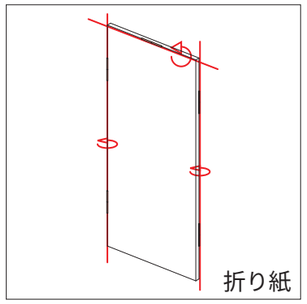


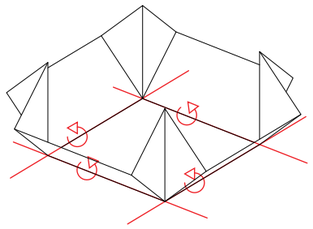
Tile magic

Akira Nishihara, "Tile magic", 1996.

3．折り紙

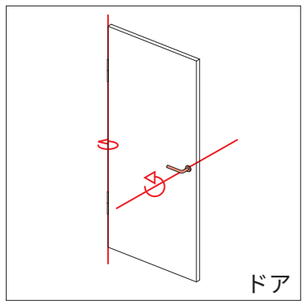
扉の複数の辺に蝶番が付いた折り紙のような状態。

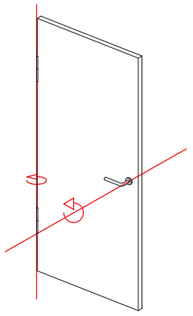




4．ドア

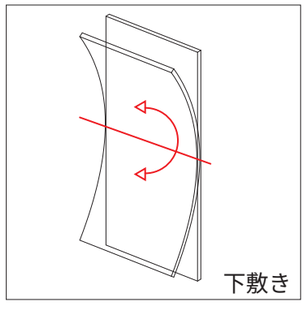
扉にドアノブと蝶番が付いたドアのような状態。

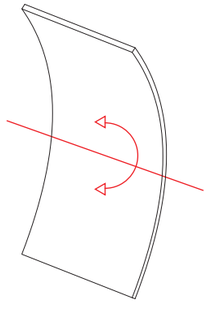




5．下敷き

扉がやわらかくて変形する下敷きのような状態。

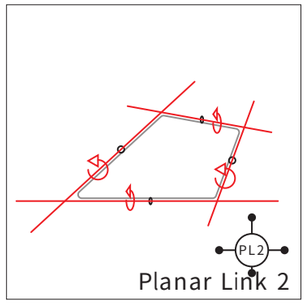


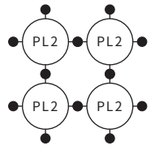


3．Planar Link 2 (PL2)

1つのリンク内に2つ以上のピンジョイントが同一平面状に存在し、そのジョイントがすべて平面に対して平行な状態。

例：MIURA-ORI





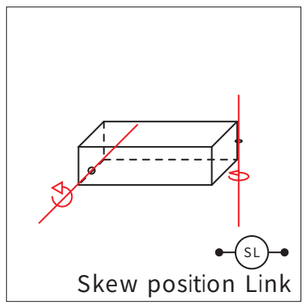
MIURA-ORI

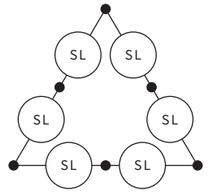
Koryo Miura,"MIURA-ORI", 1970.

4．Skew position Link (SL)

1つのリンク内に2つ以上のピンジョイントがねじれの位置で存在している状態。

例：Kaleidocycle, Invertible Cube





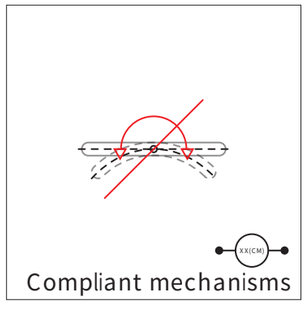
Invertible Cube

Paul Schatz, "Invertible Cube", 1929.

5．Compliant mechanisms (CM)

上記4つはピンジョイントの分類であり、これのみ例外的ではあるがリンクそのものが弾性変形によって可動部となる状態。

例：コンプライアントグリッパー [14)]

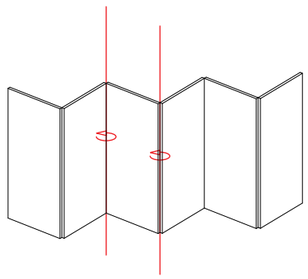


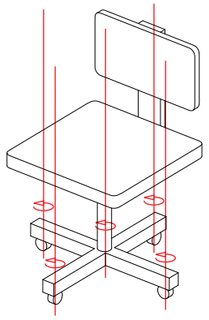
コンプライアントグリッパー

京都大学大学院 ,"コンプライアントグリッパー"

２－２．部材の組み合わせ

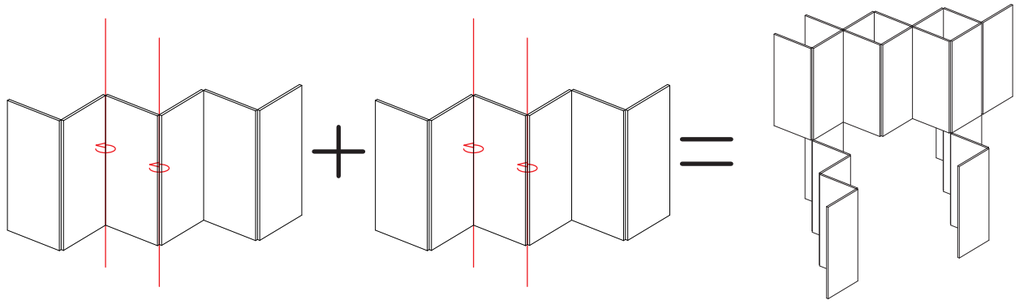
５種類に分類された部材の組み合わせ方でリンク機構による動く仕組みを11個に分類した。





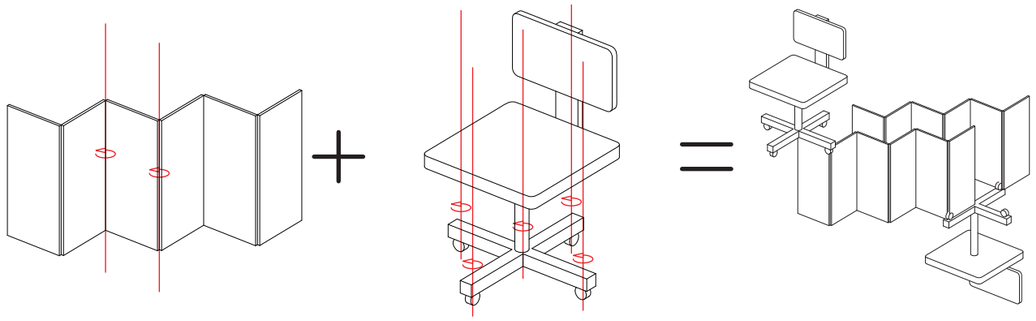
１．屏風

屏風の仕組みのみで構成されている事例。



２．屏風とオフィスチェア

屏風の仕組みとオフィスチェアの仕組みの組み合わせで構成されている事例。

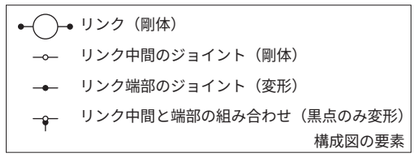


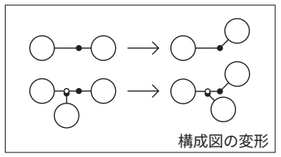
2－2．Variable Structure System

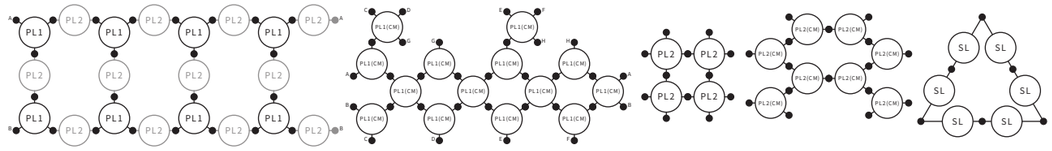
5種類に分類されたリンクの組み合わせにより現在発明されている「可変構造」を11個の事例に分類した。

しかしながら、ここで取り上げる以外の組み合わせ次第で（3つ以上の組み合わせ、あるいは Skew position Link とその他の組み合わせなど）いくらでも新しい「可変構造」は考えられうるであろう。

	LL	PL1	PL2	SL	CM
Liner Link	①	②	③		④
Planar Link 1		⑤	⑥		⑦
Planar Link 2			⑧		⑨
Skew position Link				⑩	
Compliant mechanisms					⑪



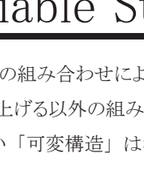


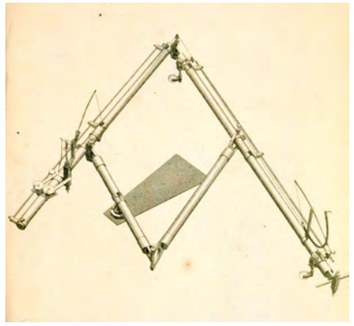


1．Liner Link

Liner Link のみ「LL-LL」で構成されている事例。

例：Pantorgraph



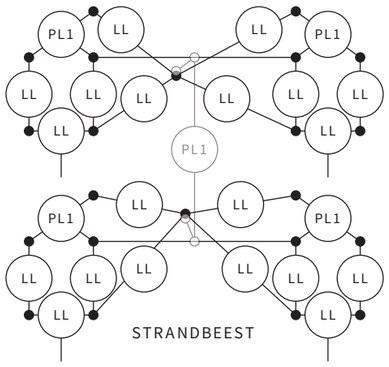


"Pantograph"

2．Liner Link and Planar Link 1

x,y 平面の Liner Link と Planar Link 1 の組み合わせを、z 軸方向で Planar Link 1 で繋いで構成されている事例。見え方としては Liner Link と Planar Link 1 で構成されており、その間に Planar Link 1 が存在する。

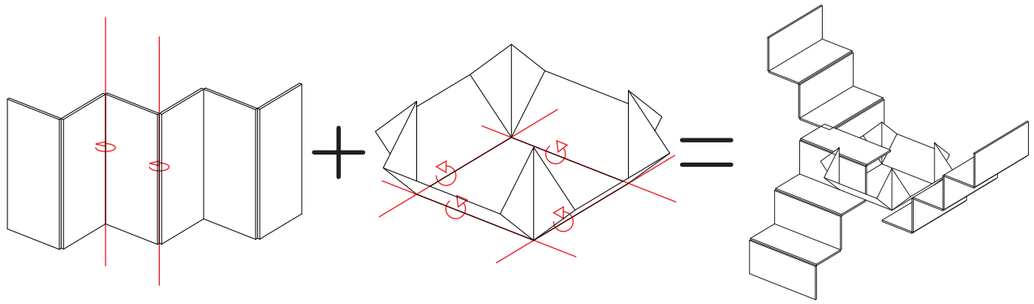
例：STRANDBEEST[15)]



Theo Jansen, "STRANDBEEST", 1990.

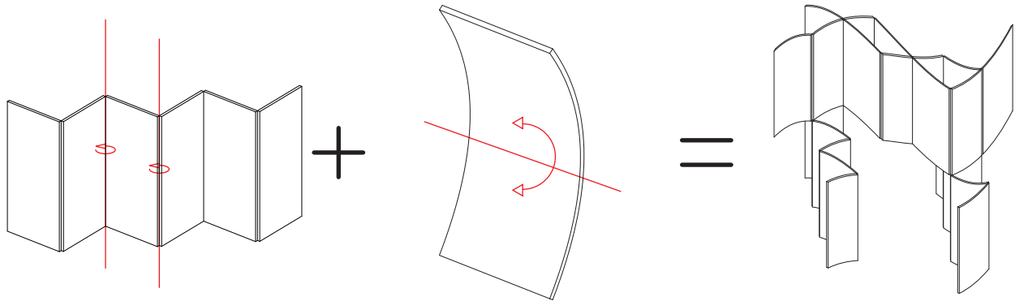
3．屏風と折り紙

屏風の仕組みと折り紙の仕組みで構成されている事例。



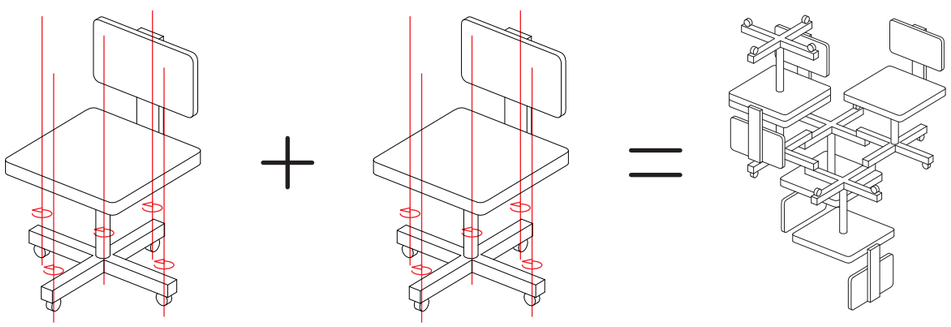
4．屏風が下敷き

屏風の仕組みが下敷きのようにゆがむもので構成されている事例。



5．オフィスチェア

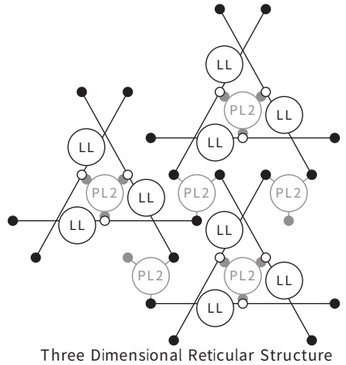
オフィスチェアの仕組みのみで構成されている事例。



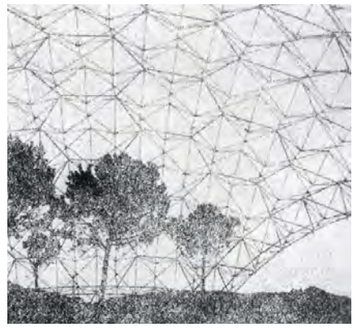
3．Liner Link and Planar Link 2

線と面の組み合わせで出来ており「LL-PL2-LL」という連続で構成されている事例。あたけぼねは例外的で、その部品としてはLLとSLを持ち構成としては一見「LL-SL-SL-LL」をとっているが、形態を構成した時点で「LL-PL2-LL」となる。

例：Three Dimensional Reticular Structure, Switch Pitch[16], Knirps[17] , あたけぼね [18]



Three Dimensional Reticular Structure

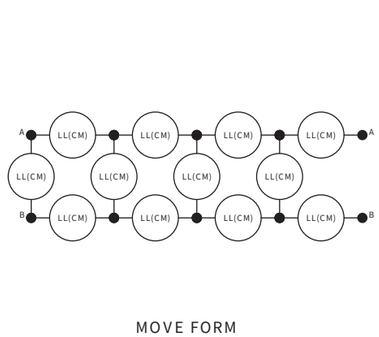


Emilio Perez Pinero, ″Three Dimensional Reticular Structure″,1960s

4．Liner Link is Compliant mechanisms

Liner Linkで構成され、リンクが弾性変形を起こす「LL(CM)-LL(CM)」事例。尚、MOVE FORMについては「LL-SL-SL-LL」あるいは「SL-SL-SL-SL」に翻訳することが可能である。

例：MOVE FORM[19]



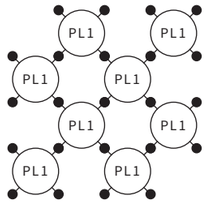
MOVE FORM

戸村浩 ,″MOVE FORM″, 1964.

5．Planar Link 1

Planar Link 1のみ「PL1-PL1」で構成されている事例。

例：Tile magic

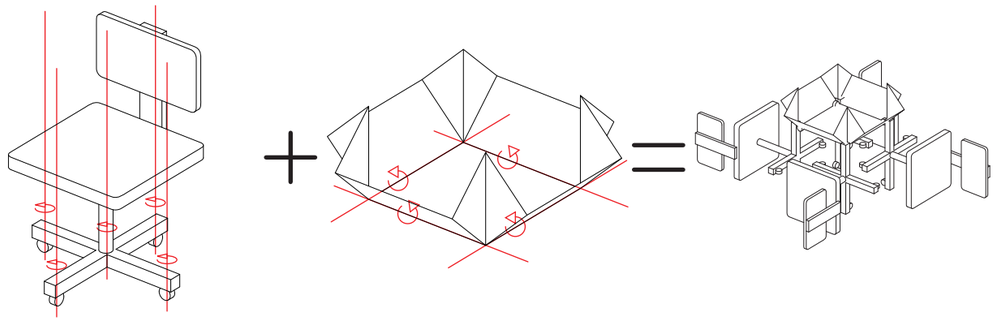


Tile magic

Akira Nishihara, ″Tile magic″, 1996.

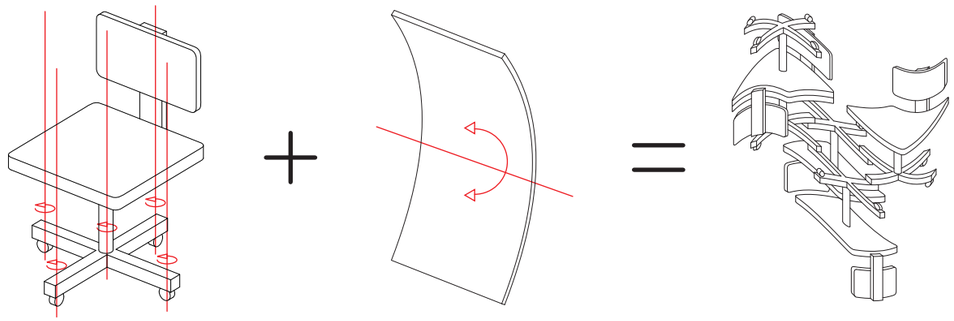
6．オフィスチェアと折り紙

オフィスチェアの仕組みと折り紙の仕組みで構成されている事例。



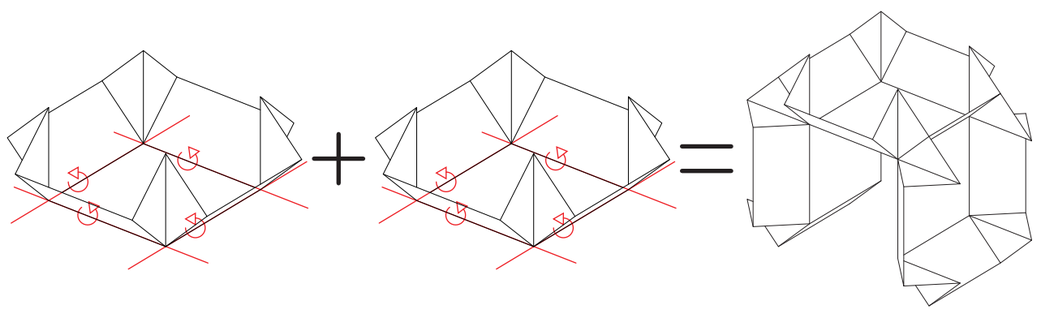
7．オフィスチェアが下敷き

オフィスチェアの仕組みが下敷きのようにゆがむもので構成されている事例。



8．折り紙

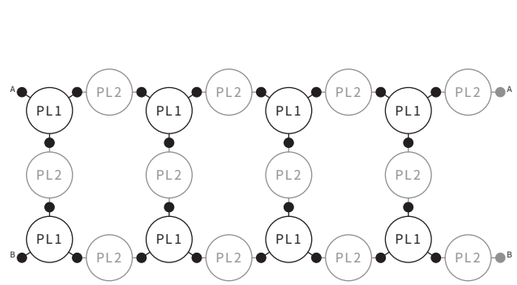
折り紙の仕組みのみで構成されている事例。



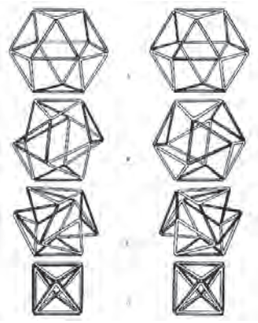
6．Planar Link 1 and Planar Link 2

面の組み合わせで出来ており「PL1-PL2-PL1」という連続で構成されている事例。見え方としては Planar Link 1 で構成されており、Planar Link 2 はその間に存在する。尚、Jitter Bug については「PL2-PL2-PL2-PL2」に翻訳することが可能であることが分かっている。

例：Jitter Bug, Flip-Flop Ball[20], ひろがる玩具 [21], Hoberman Sphere[22]



Jitter Bug

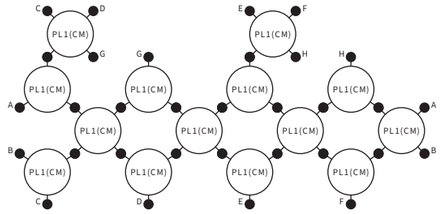


R. Buckminster Fuller "Vector Equilibrium", 1948

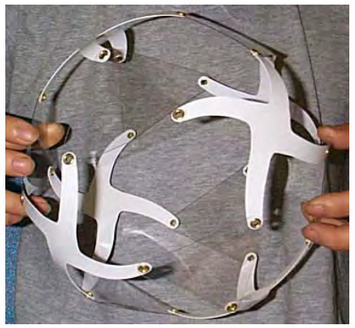
7．Planar Link 1 is Compliant mechanisms

面のピンジョイントで構成され、リンクが弾性変形を起こす「PL1(CM) -PL1(CM)」事例。尚、Juno's Spinner については「PL2-SL-SL-PL2」に翻訳することが可能である。

例：Juno's Spinner[23]



Juno ' s Spiner

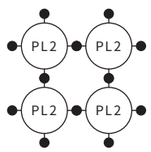


柳瀬順一，「Juno's Spinner」，1988.

8．Planar Link 2

Planar Link 2 のみ「PL2-PL2」で構成されている事例。

例：MIURA-ORI,GOKO-ORI[24]

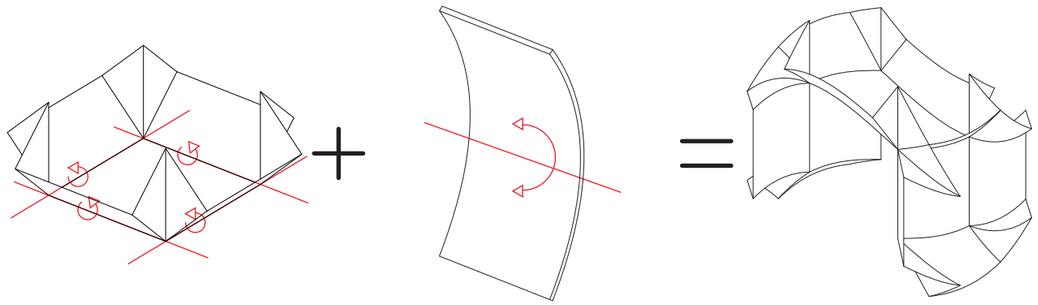


MIURA-ORI

Koryo Miura,"MIURA-ORI", 1970.

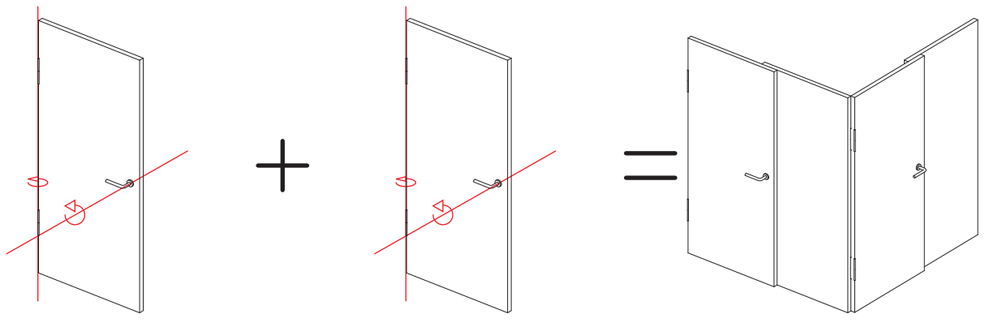
９．折り紙が下敷き

折り紙の仕組みが下敷きのようにゆがむもので構成されている事例。



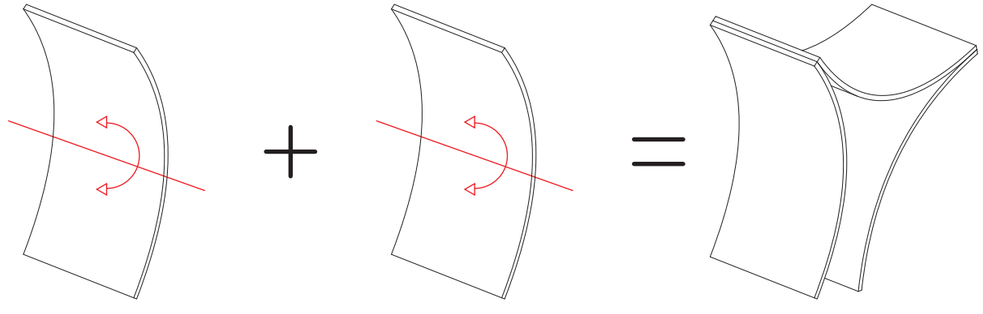
１０．ドア

ドアの仕組みのみで構成されている事例。



１１．下敷き

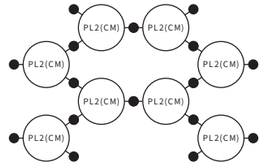
下敷きのようなゆがみのみで出来ている事例。



9．Planar Link 2 is Compliant mechanisms

面のピンジョイントで構成され、リンクが弾性変形を起こす「PL2(CM) -PL2(CM)」事例。

例：Origami Spiral[25)]



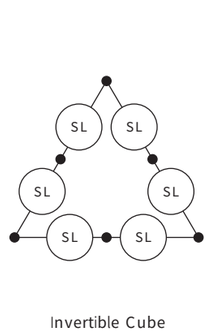
Origami Spiral

Jeff Beynon, ″Origami Spiral″

10．Skew position Link

Skew position Link のみ「SL-SL」で構成されている事例。より細分化すると単体のリンク内にその他の関係性が含まれているか、その含まれ方で分けることが可能。ミウラ折りに厚みを持たせた実験的なものを著者は制作した。[26)]

例：Kaleidocycle, Invertible Cube, Bennett Linkage[27)]



Invertible Cube

Paul Schatz, ″Invertible Cube″, 1929.

11．Compliant mechanisms

Compliant mechanisms のみで出来ている事例。

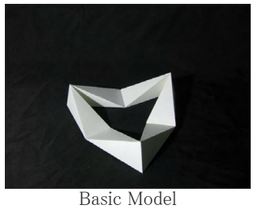
例：コンプライアントグリッパー

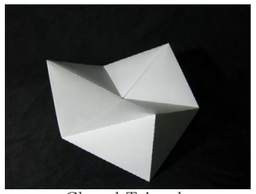
コンプライアントグリッパー

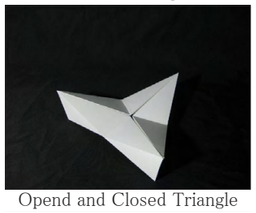
京都大学大学院 ,″コンプライアントグリッパー ″

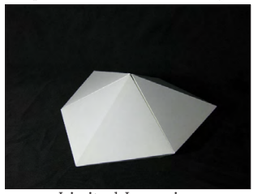
カレイドトランスフォーメーション

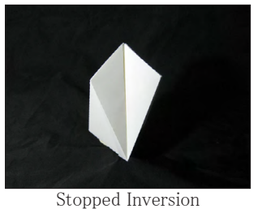
Kaleidotransformation

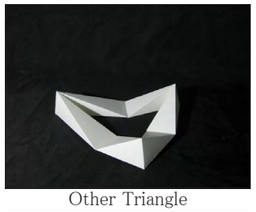

Basic Model


Closed Triangle

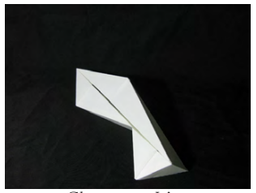

Opend and Closed Triangle


Limited Inversion


Stopped Inversion

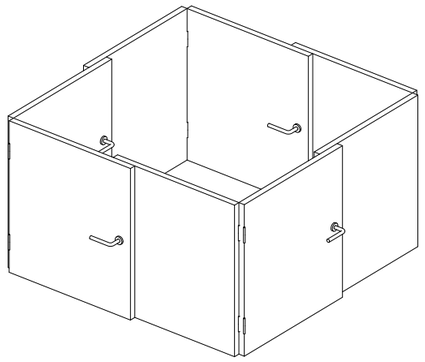

Other Triangle

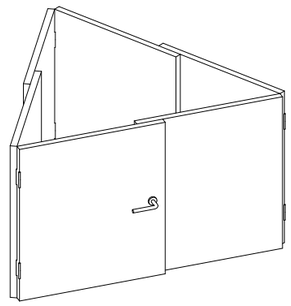

Change to Triangle

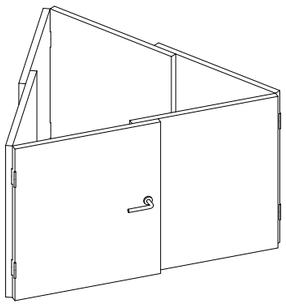

Change to Line

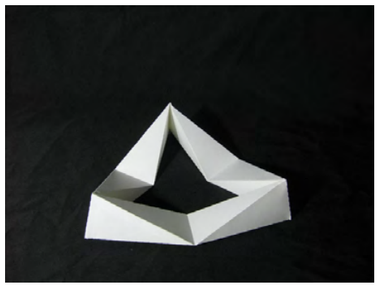
3．カレイドトランスフォーメイション

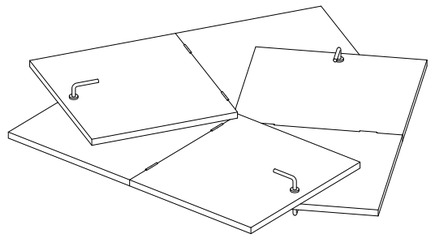
ドアの組み合わせで構成していくものは現在、単純な形状のものを環状につなげただけで終わらせており、発展展開が見られない。そこで、それを網目状に繋いでいくことや、ドア単体の変形方法について研究することによって、新たな可変構造の可能性を広げることが出来ると考えた。

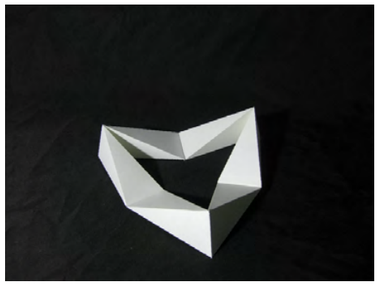








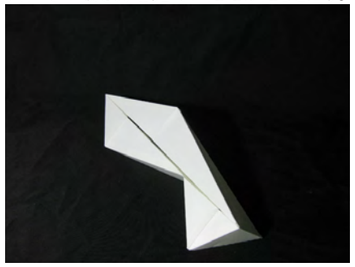


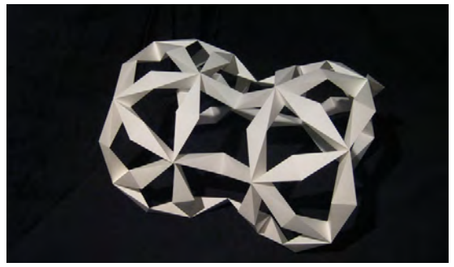


3．Kaleidotransformation

11個の分類のなかでとりわけ Kaleidocycle つまり Skew position Link のみで構成された「可変構造」は現在のところ、四面体を環状に繋いだだけの構成でとどまっている[28]。これについて「″Metamorphosis″（Kaleidocycle の形態を変化させる）」「″Increase″（Kaleidocycle の数を増やす）」の2つ手法によって、今までにない「可変構造」の新たな方向性を見出せるのではないかと考えた。

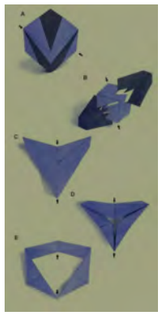
このページより先、両端に Kaleidotransformation の変形をパラパラ漫画式で載せている。解説と共にその動き方を見ていただけたらと思う。





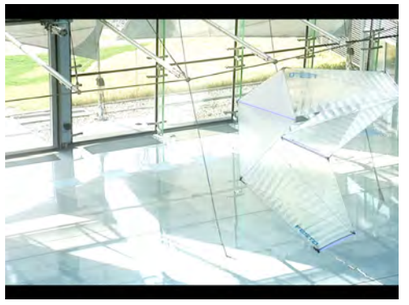
Kaleidocycle

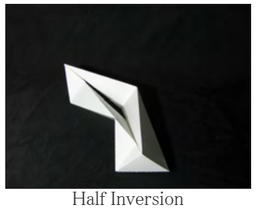
1929 年に Paul Schatz が発明した ″Invertible Cube″ を基とした反転運動を起こす可変構造体。後の 1977 年に Doris Schattschneide が ″M. C. Escher Kaleidocycles″ にて Kaleidocycle という名で機構はそのままに形態を変化させた状態で取り上げて以来、カライドサイクルという名で研究、紹介されている。この可変構造の最大の特徴は、Inversion（反転運動）と呼ばれる動きをすることで、Casper Schwabe の「ジオメトリック・アート[29]」では「従来から、多くの運動が、①（並進運動）と②（回転運動）の組み合わせで理解できると考えられてきました。らせん運動やひねり、ねじりなどに、③反転（反転運動）という要素が関わっていることを、科学や工学が公式に認めるようになったのは、それほど昔のことではありません。(p.158)」と紹介されている。本研究は、この ″Inversion″ を使ったリンク機構をより発展、展開させる試みともいえる。

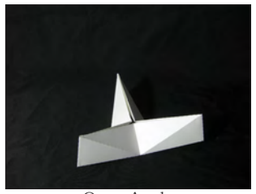


Paul Shatz(1898-1979)

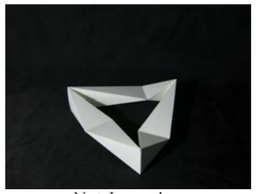
Kaleidocycle の基となる ″The Invertible Cube″ を発明した彫刻家。その他に ″Polysomatic Forms″ や ″The Geometry of Space″、″The Oloid″、″Further Developments″ などの発明をした。彼の死後も Paul Schatz Stiftung として彼の発明を使った研究や制作等が企業や大学と連携し行われているようである。

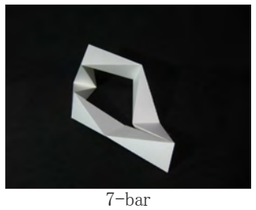


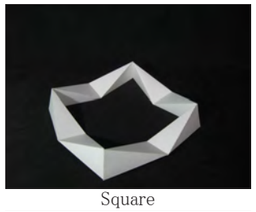

Half Inversion


Over Angle

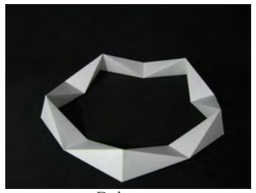

Change to Minimum


Not Inversion


7-bar


Square


9-bar

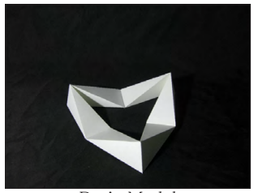

Polygon

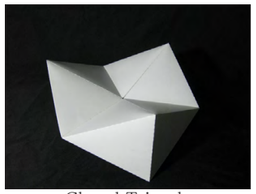
変形

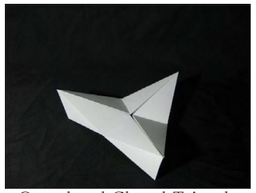
Metamorphosis

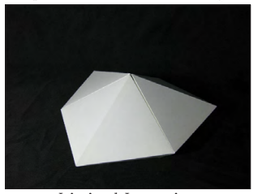
4． 変形

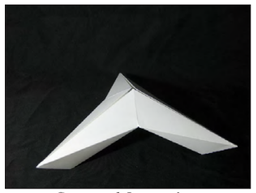
Kaleidocycle の変形方法は「ドアの形状」「蝶番の距離」「蝶番とドアノブの距離と角度関係」「ドアの数」の四つによって整理させられると考えた。

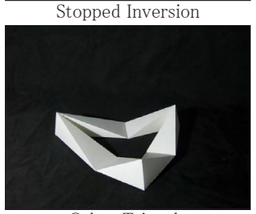

Basic Model

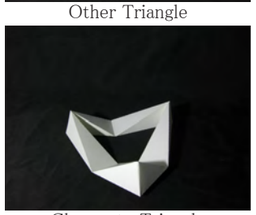

Closed Triangle

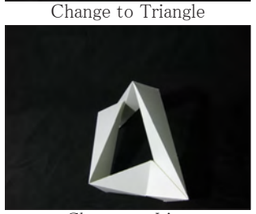

Opend and Closed Triangle

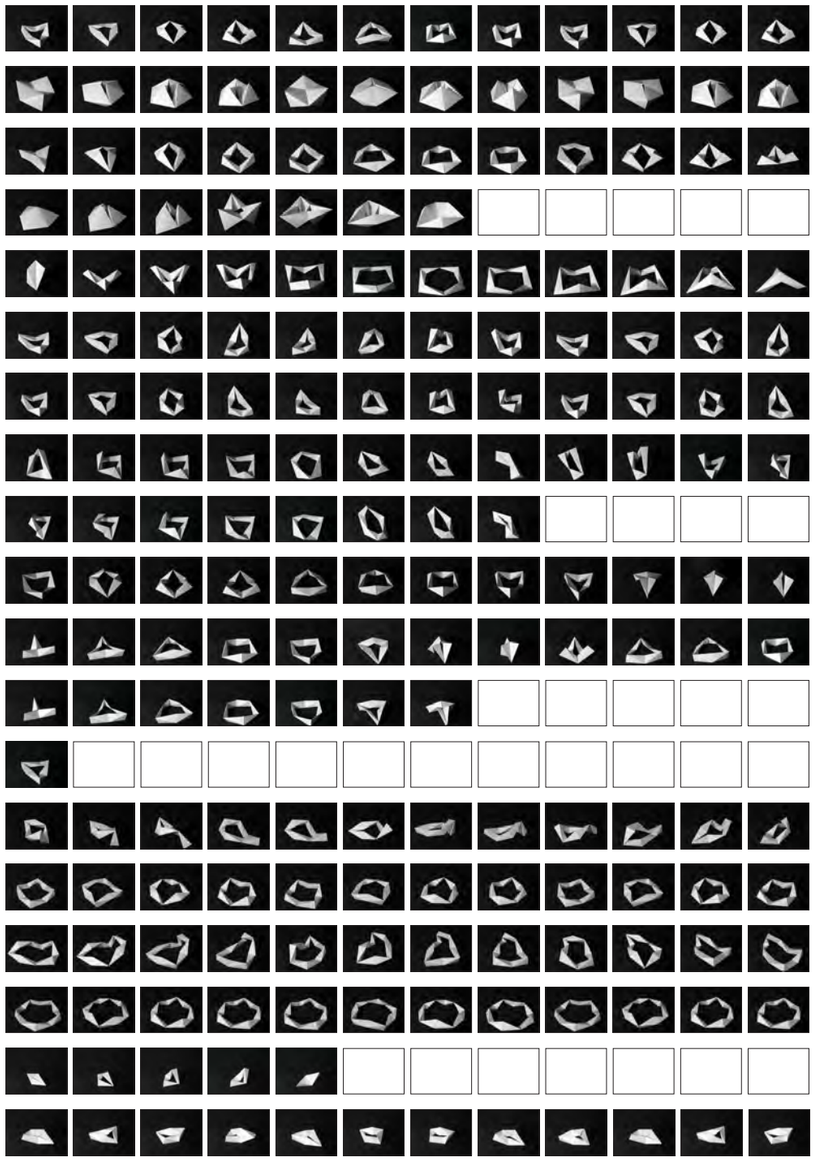

Limited Inversion


Stopped Inversion


Other Triangle


Change to Triangle

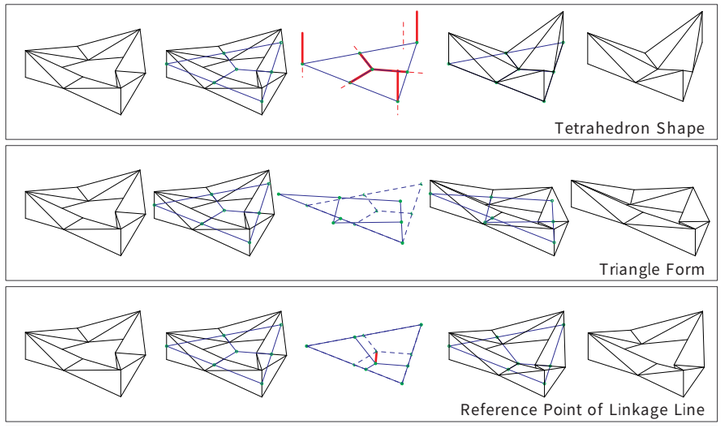

Change to Line

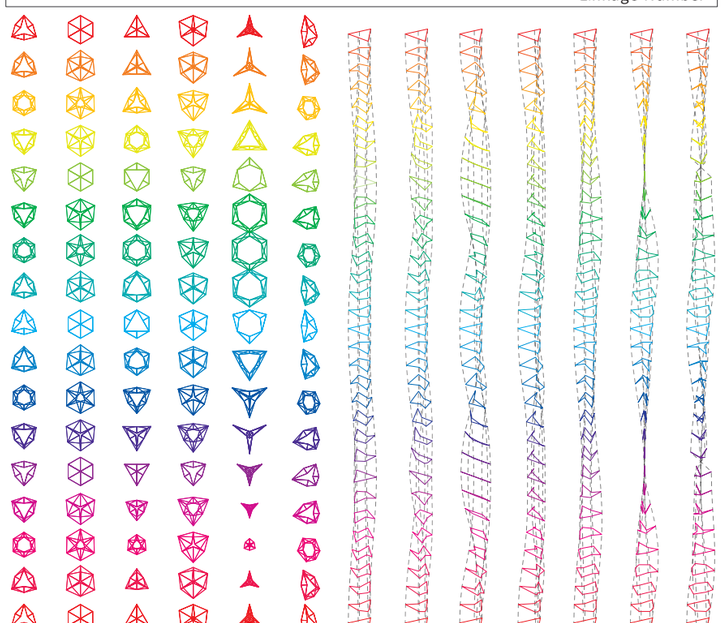


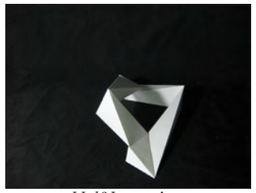
4． Metamorphosis

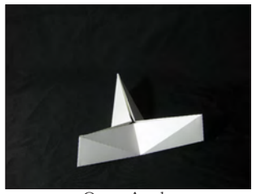
まず、Kaleidocycle 1ユニットにおける変形方法について述べる。大きく"Tetrahedron Shape"（リンク単体四面体の形の変化）、"Triangle Form"（基準三角形の変化）、"Reference Point of Linkage Line"（ピンジョイントの位置と角度関係の変化）、"Linkage Number"（リンク数の変化）の4つの方法で1ユニットは"Metamorphosis"させることが出来ると考えた。

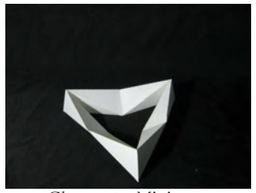
Kaleidocycle の反転運動は、ジョイント同士を結ぶ基準ラインを見ると1回の反転運動中に2回または4回、平面（三角形）の状態をとる。

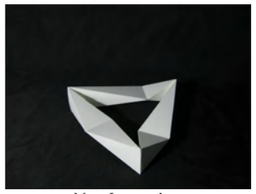


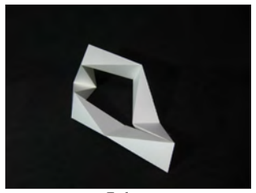


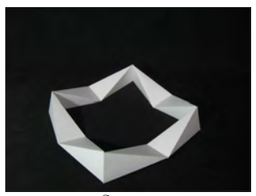

Half Inversion

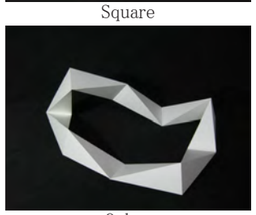

Over Angle

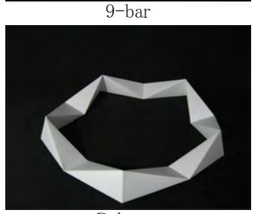

Change to Minimum


Not Inversion


7-bar

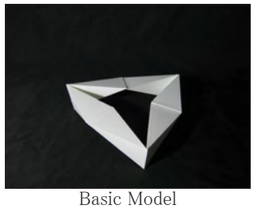

Square

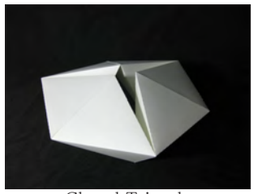

9-bar

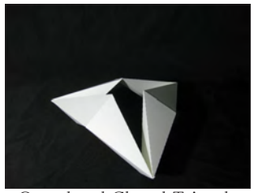

Polygon

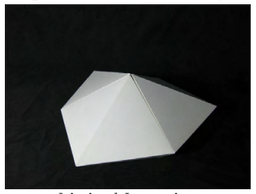
４－１．ドアの形状

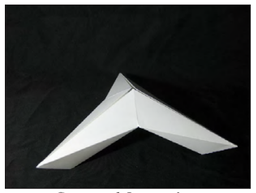
ドアの形状を変化させることによって、くるくる回転する量と環状の中の隙間が変化する。

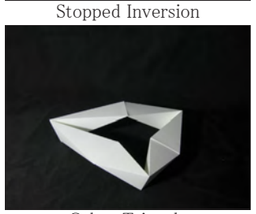

Basic Model

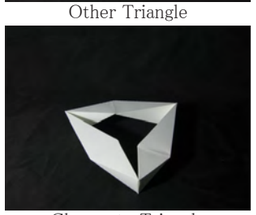

Closed Triangle

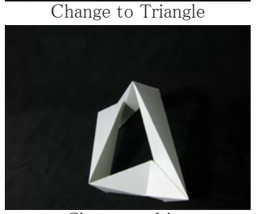

Opend and Closed Triangle

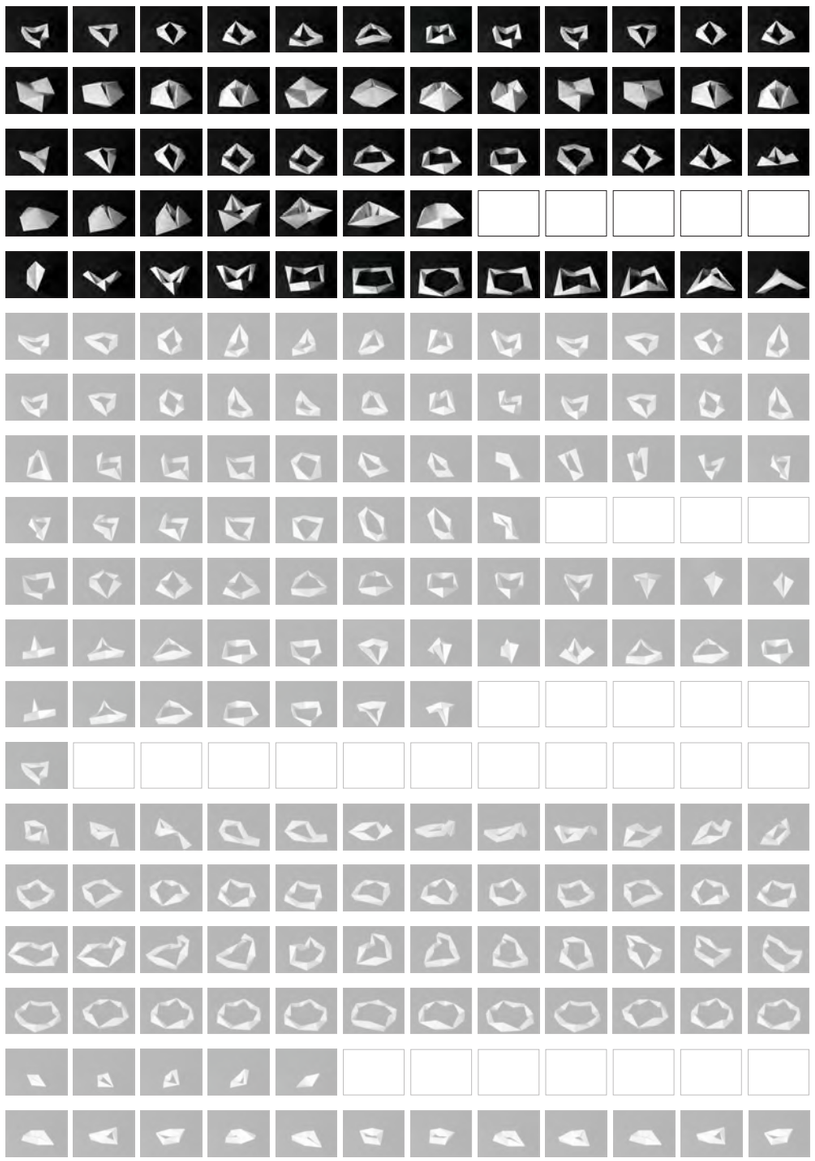

Limited Inversion


Stopped Inversion


Other Triangle


Change to Triangle

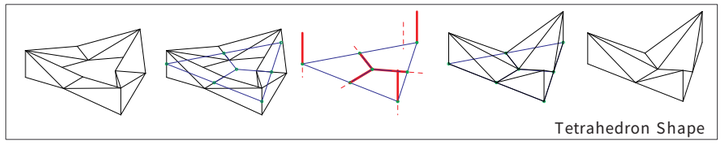

Change to Line

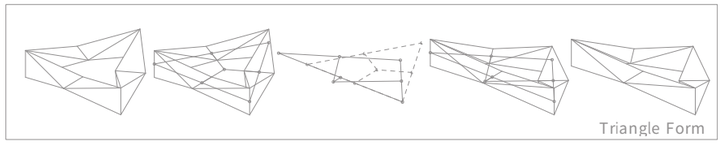


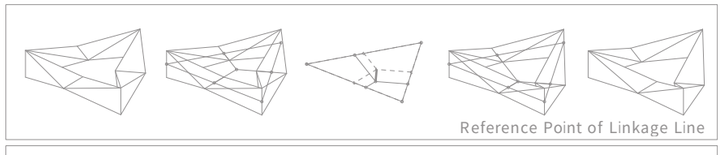
4－1．Tetrahedron Shape

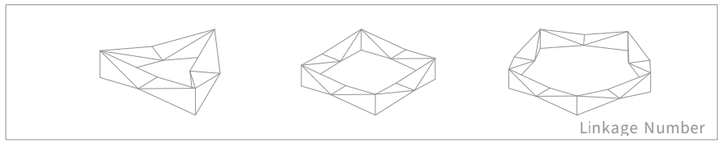
リンク単体四面体の形を変化させることによって、反転運動量と環状内部の空白量が変化する。

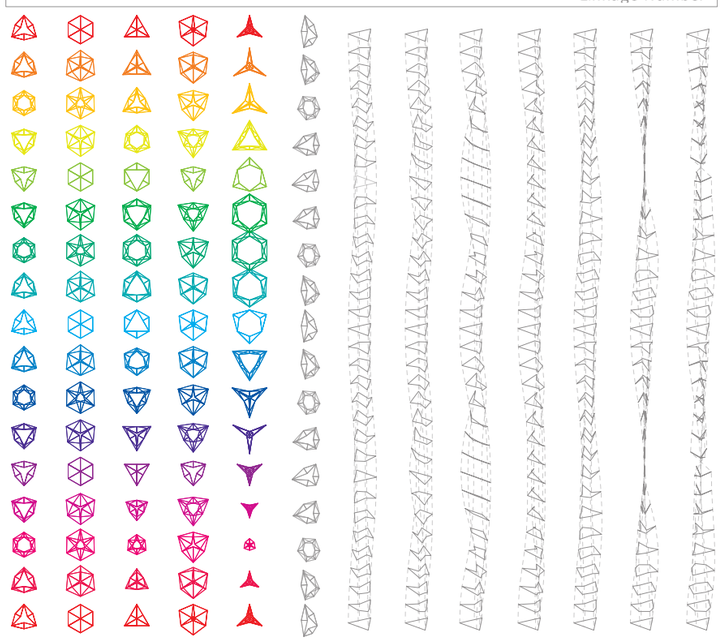
4－1－A．Basic Model: Opened Triangle
4－1－B．Application Model: Closed Triangle
4－1－C．Application Model: Opened and Closed Triangle
4－1－D．Application Model: Limited Inversion
4－1－E．Application Model: Stopped Inversion



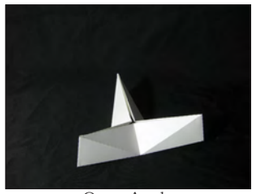


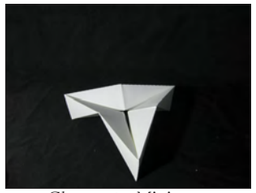


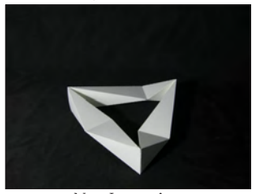




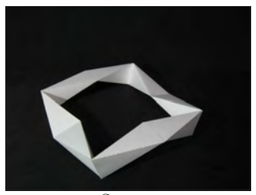

Half Inversion

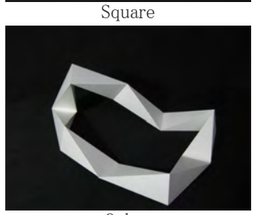

Over Angle

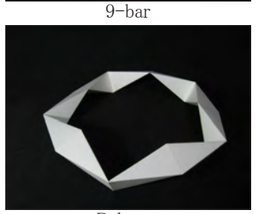

Change to Minimum


Not Inversion

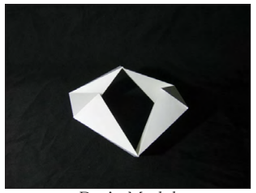

7-bar

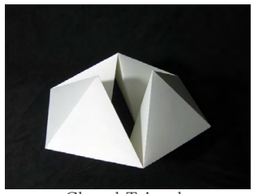

Square

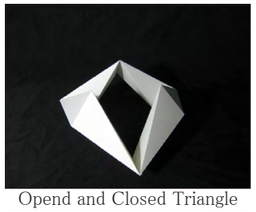

9-bar

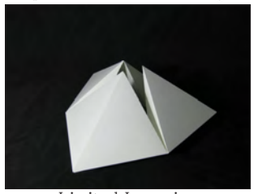

Polygon

４－１－Ａ．基本モデル： 開いた三角形

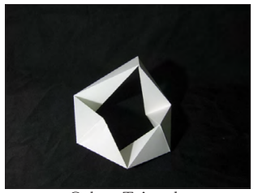

Basic Model

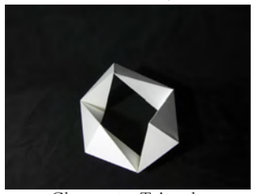

Closed Triangle

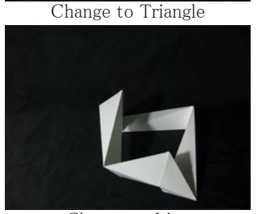

Opend and Closed Triangle

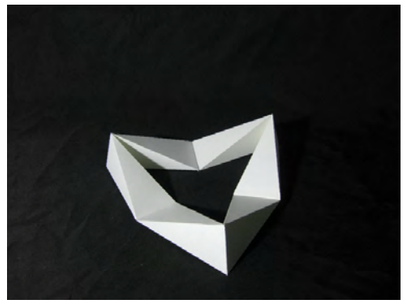

Limited Inversion

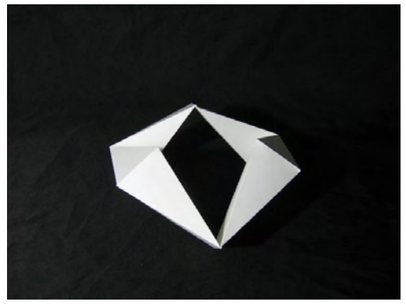

Stopped Inversion

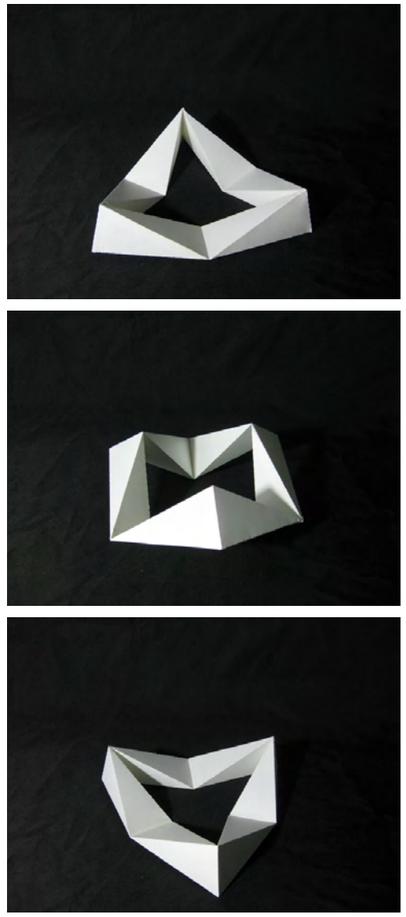

Other Triangle


Change to Triangle

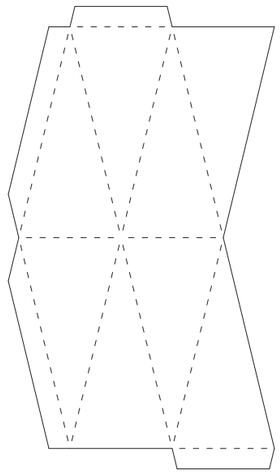

Change to Line

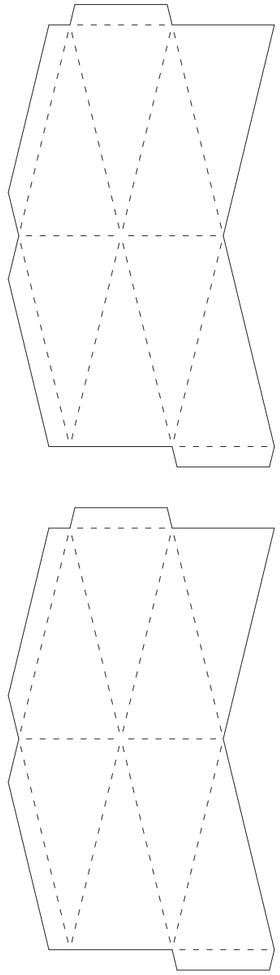






回転写真

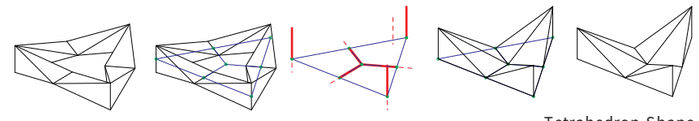


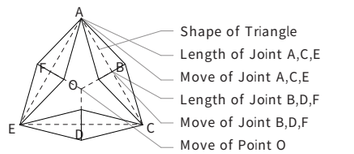


展開図

4－1－A．Basic Model: Opend Triangle

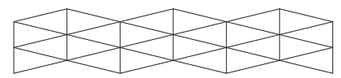
基準三角形の状態で隙間が生まれているもの。



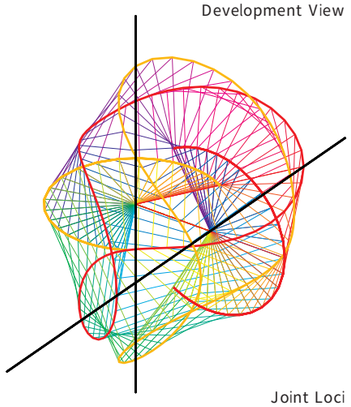


Shape of Triangle : AC:CE:EA = 1:1:1
Length of Joint A,C,E : AC * 1 / 4
Z Move of Joint A,C,E : 0
Length of Joint B,D,F : AC * 1 / 4
XY Move of Joint B,D,F : 0
Z Move of Point O : 0
XY Move of Point O : AC:AO:CO = √3:1:1

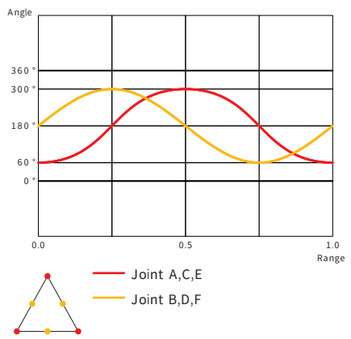
Metamorphosis Parameters



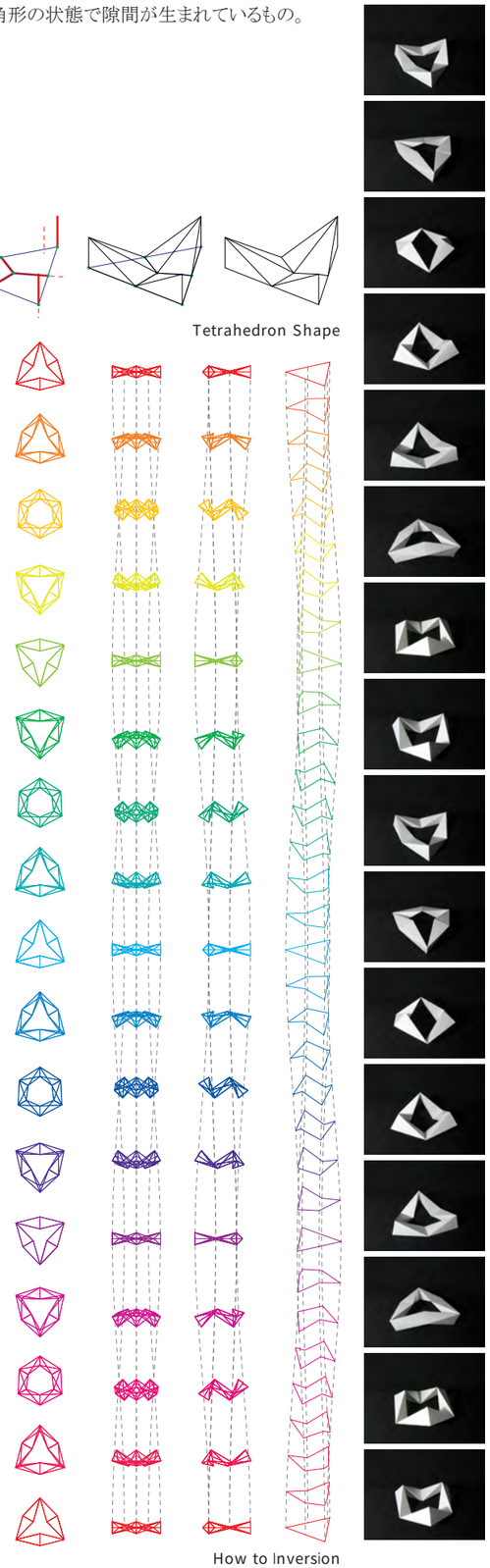
Development View



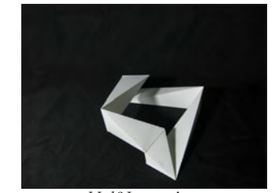
Joint Loci

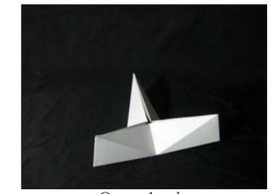


Joint Angles



How to Inversion

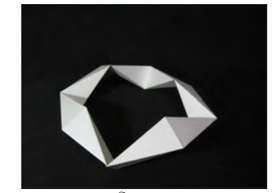

Half Inversion


Over Angle

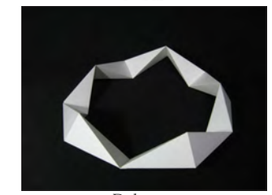

Change to Minimum


Not Inversion

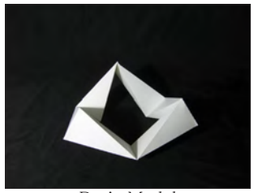

7-bar

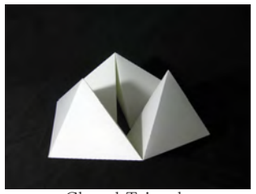

Square

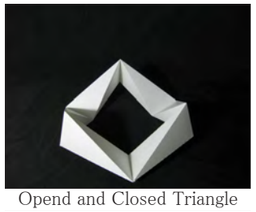

9-bar

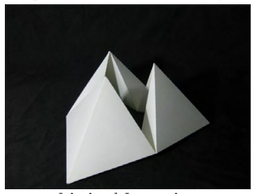

Polygon

４－１－Ｂ．発展モデル： 閉じた三角形


Basic Model

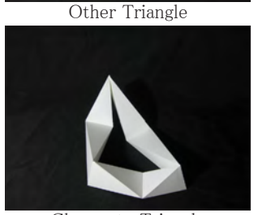

Closed Triangle

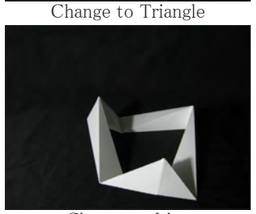

Opend and Closed Triangle

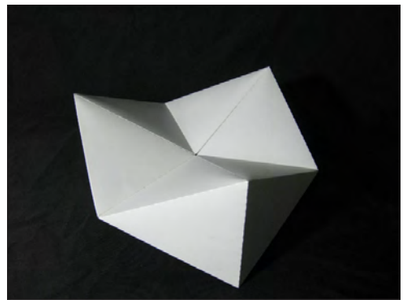

Limited Inversion

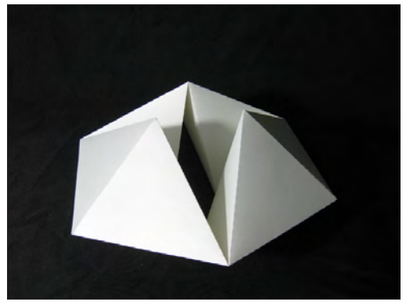

Stopped Inversion

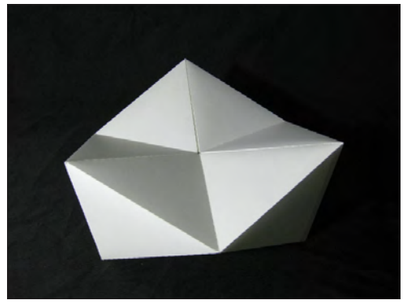

Other Triangle

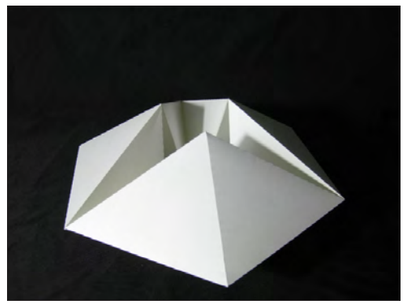

Change to Triangle

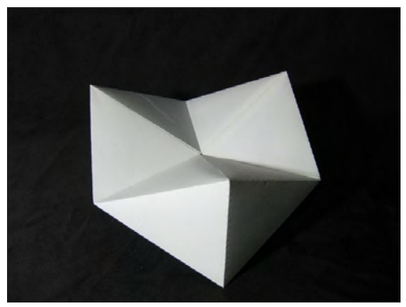

Change to Line



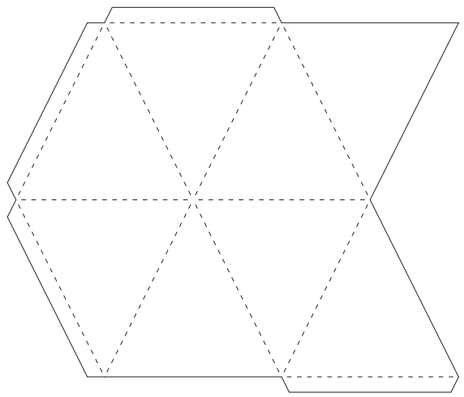


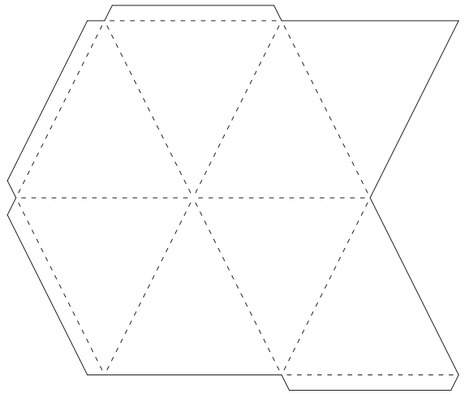


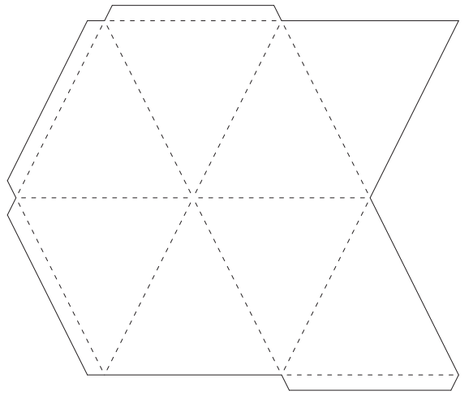




回転写真



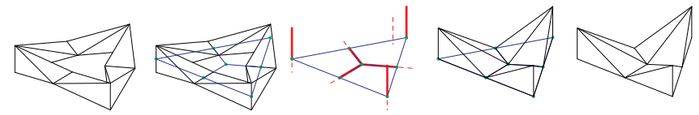


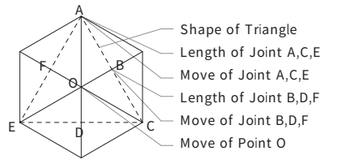


展開図

4－1－B．Application Model: Closed Triangle

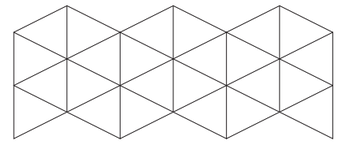
基準三角形の状態で隙間が完全にないもの。一般的に知られる Kaleidocycle はこの構成をとる。

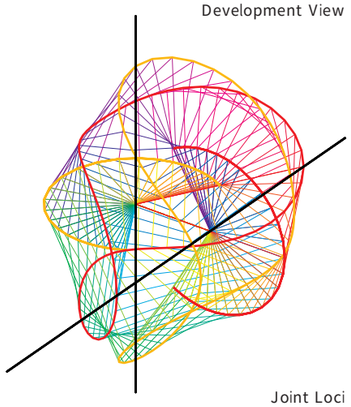

Tetrahedron Shape

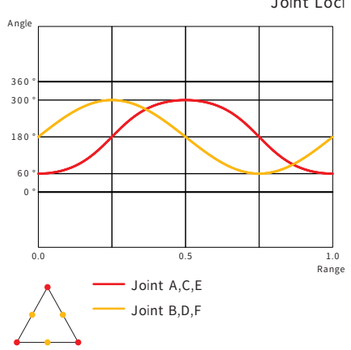


Shape of Triangle : AC:CE:EA = 1:1:1
Length of Joint A,C,E : AC * √3 / 3
Z Move of Joint A,C,E : 0
Length of Joint B,D,F : AC * √3 / 3
XY Move of Joint B,D,F : 0
Z Move of Point O : 0
XY Move of Point O : AC:AO:CO = √3:1:1

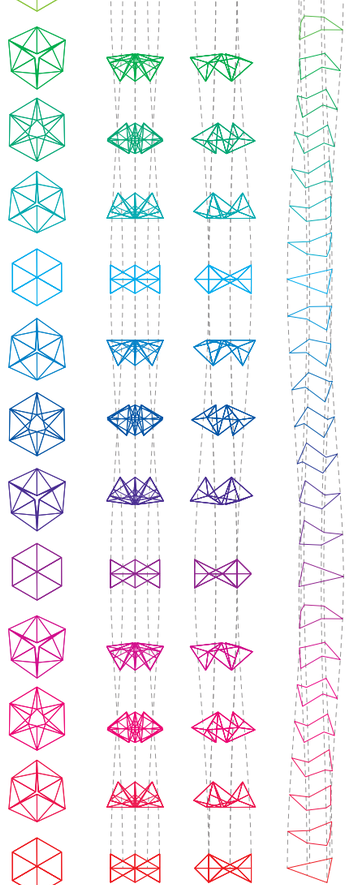
Metamorphosis Parameters

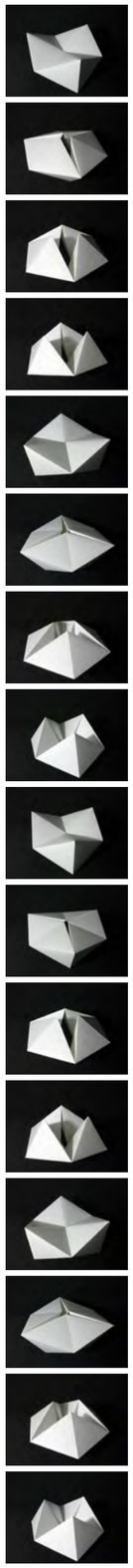

Development View

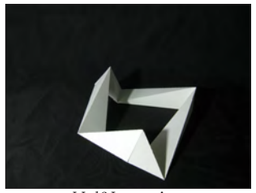

Joint Loci

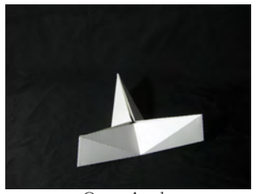


Joint Angles

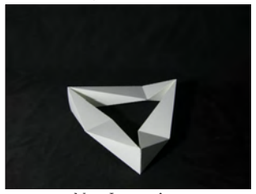

How to Inversion

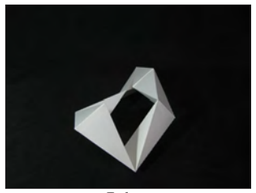


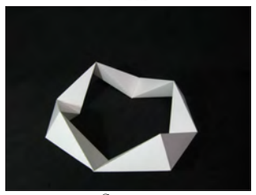

Half Inversion

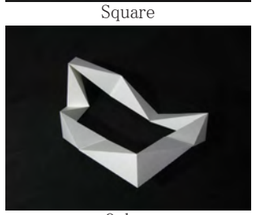

Over Angle

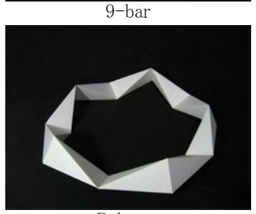

Change to Minimum


Not Inversion

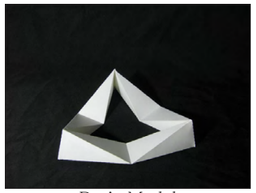

7-bar

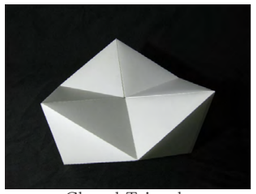

Square

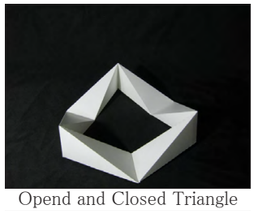

9-bar

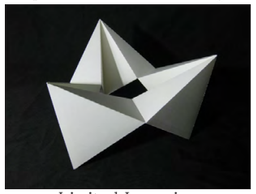

Polygon

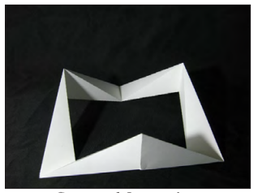
４－１－C．発展モデル：開いたり閉じたりの三角形


Basic Model

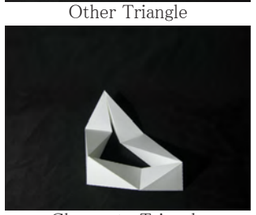

Closed Triangle

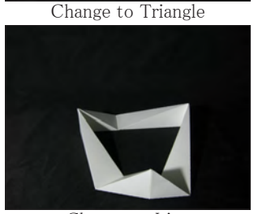

Opend and Closed Triangle

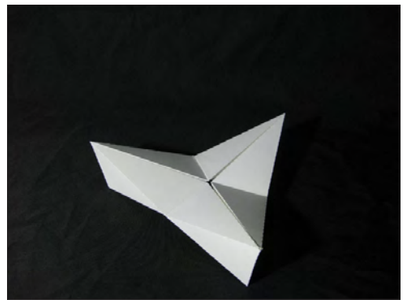

Limited Inversion

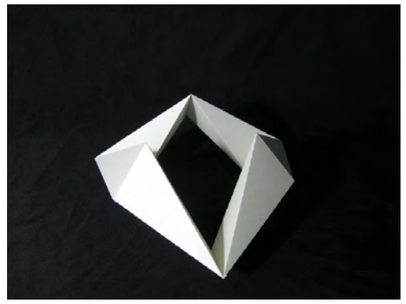

Stopped Inversion

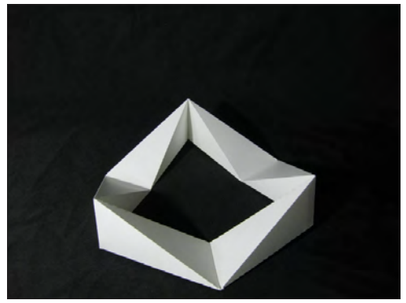

Other Triangle

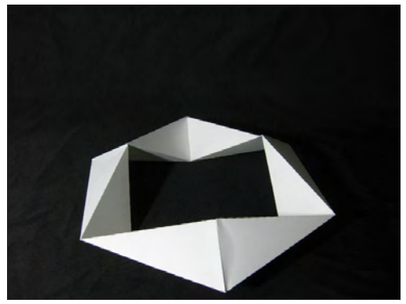

Change to Triangle

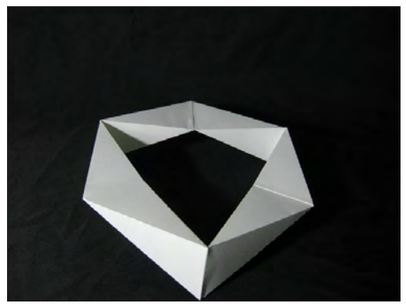

Change to Line



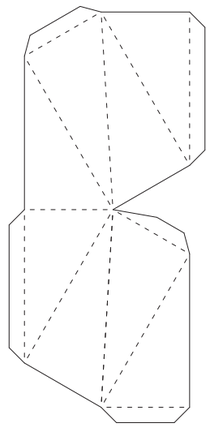


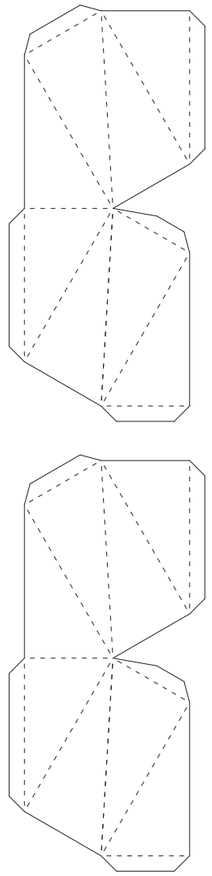






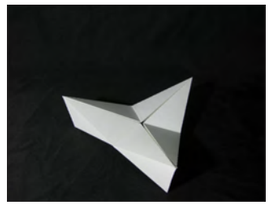
回転写真



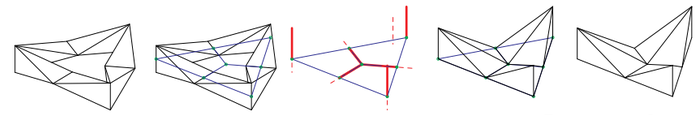


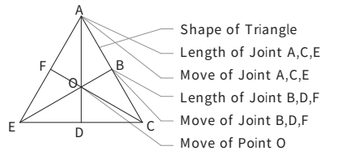
展開図

4－1－C．Application Model: Opened and Closed Triangle



基準三角形の状態で隙間がある状態とない状態を交互に繰り返すもの。Invertible Cube はこの構成をとる。

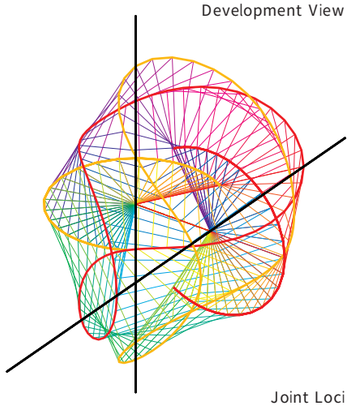

Tetrahedron Shape

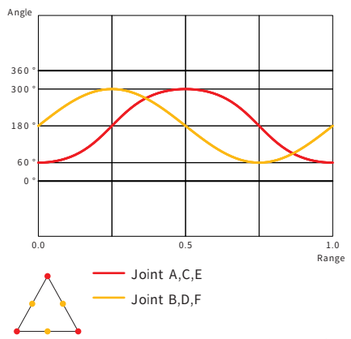


Shape of Triangle : AC:CE:EA = 1:1:1
Length of Joint A,C,E : AC * √3 / 6
Z Move of Joint A,C,E : AC * √3 / 12
Length of Joint B,D,F : AC * √3 / 6
XY Move of Joint B,D,F: AC * √3 / 12
Z Move of Point O : 0
XY Move of Point O : AC:AO:CO = √3:1:1

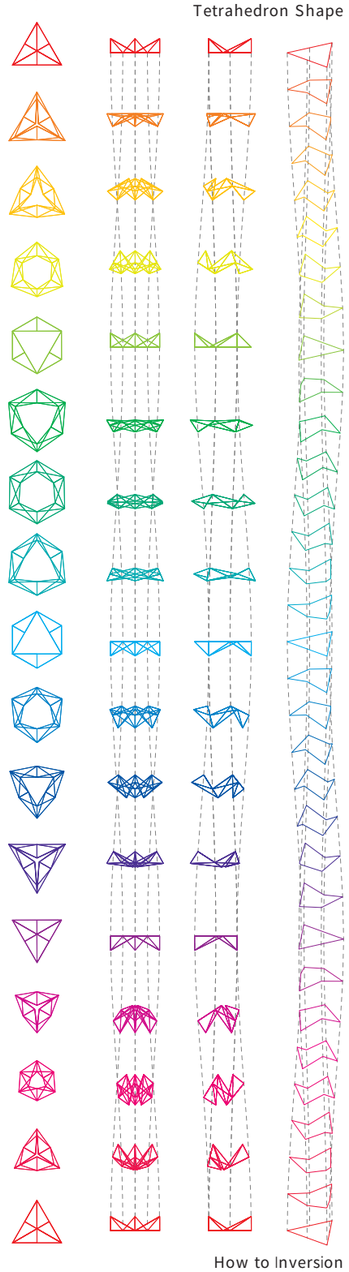
Metamorphosis Parameters

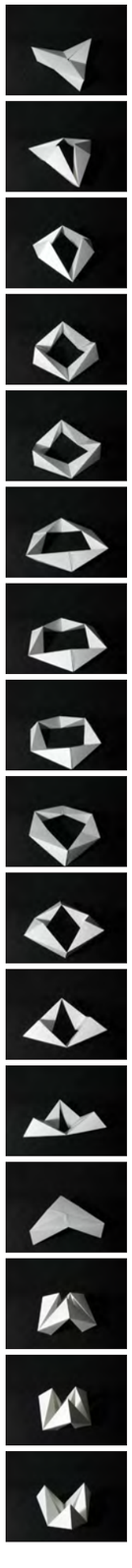

Development View

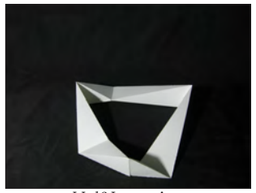

Joint Loci

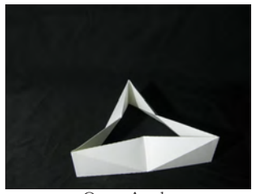


Joint Angles

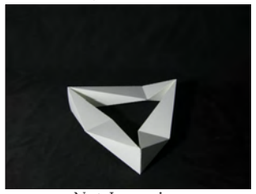

How to Inversion



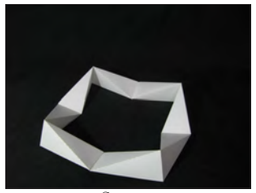

Half Inversion

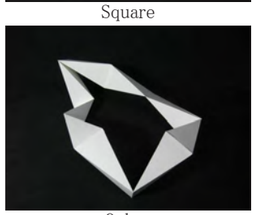

Over Angle


Change to Minimum


Not Inversion

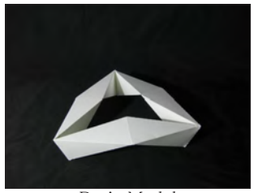

7-bar

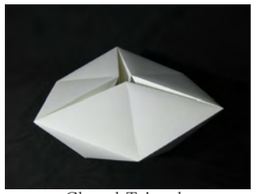

Square

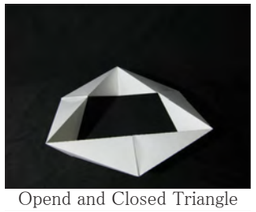

9-bar

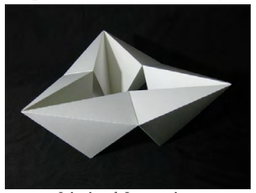

Polygon

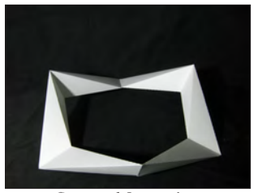
４－１－Ｄ．発展モデル： 制限された回転


Basic Model

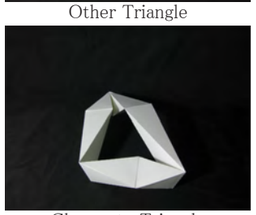

Closed Triangle

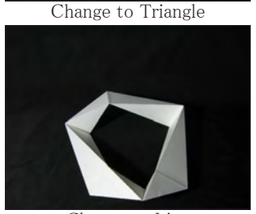

Opend and Closed Triangle

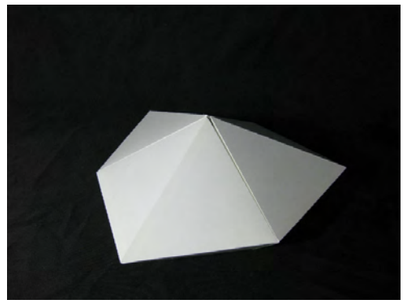

Limited Inversion

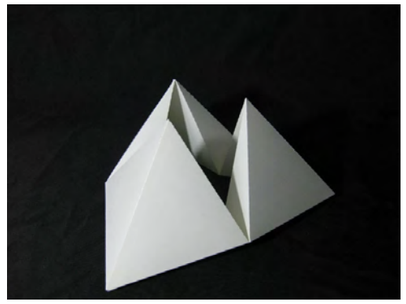

Stopped Inversion

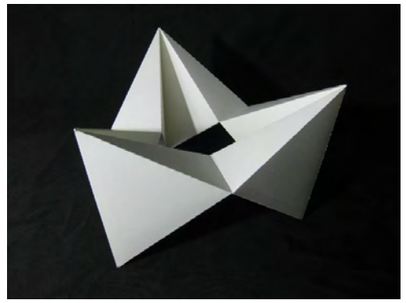

Other Triangle

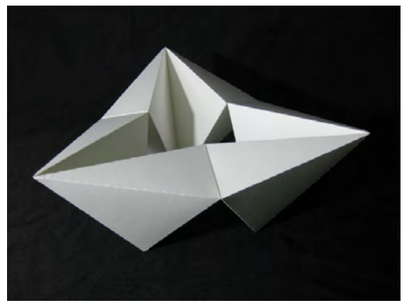

Change to Triangle

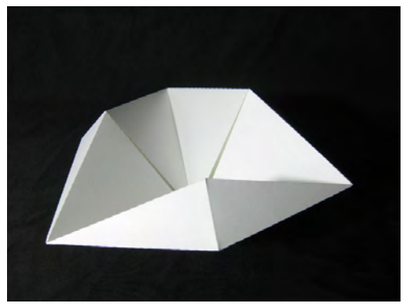

Change to Line



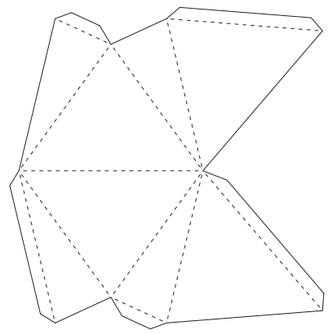


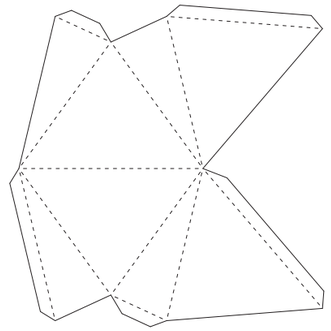


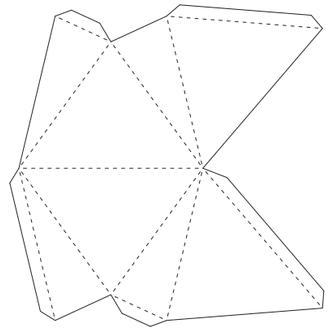




回転写真

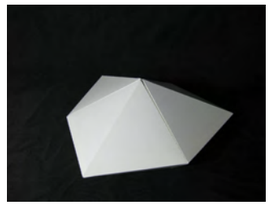




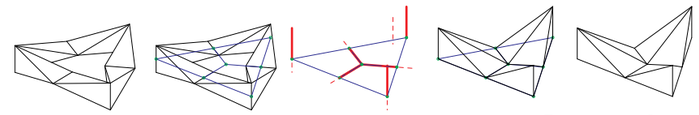


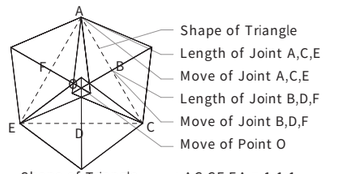
展開図

4－1－D．Application Model: Limited Inversion



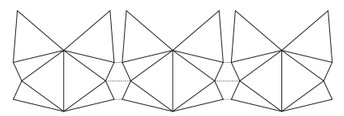
隙間を埋めたときに、結果として四面体同士がぶつかりそこで反転運動がストップしてしまうもの。

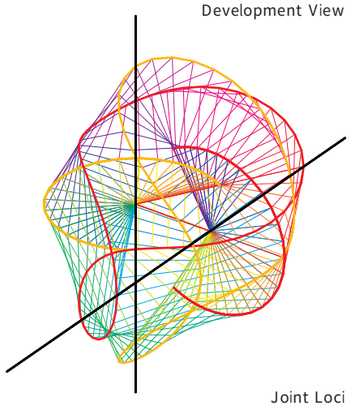

Tetrahedron Shape

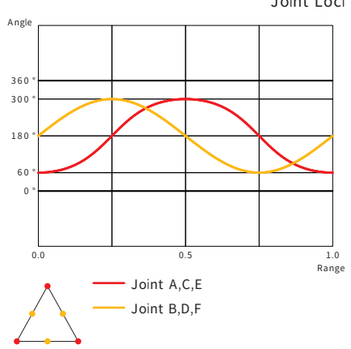


Shape of Triangle : AC:CE:EA = 1:1:1
Length of Joint A,C,E : AC * 1 / 4
Z Move of Joint A,C,E : 0
Length of Joint B,D,F : AC * 3 / 4
XY Move of Joint B,D,F : 0
Z Move of Point O : 0
XY Move of Point O : AC:AO:CO = √3:1:1

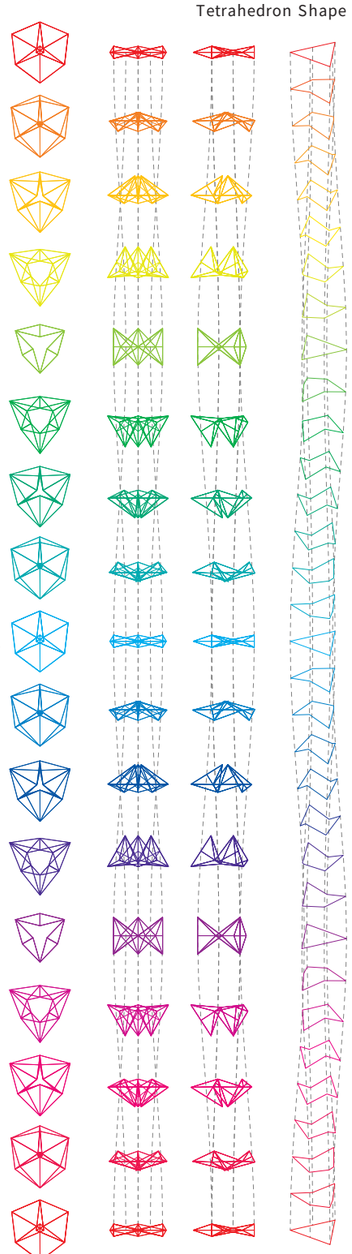
Metamorphosis Parameters

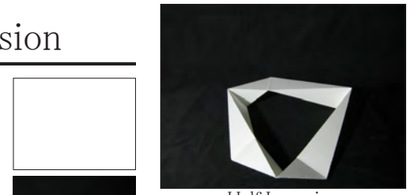

Development View

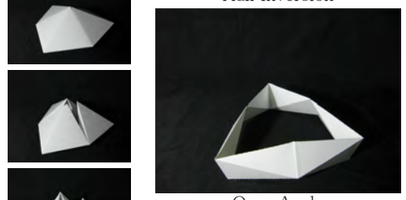

Joint Loci

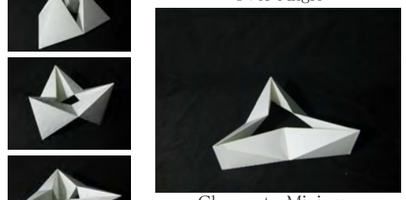


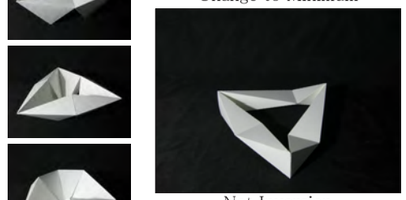
Joint Angles

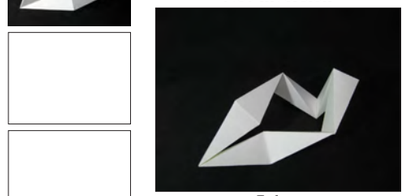

How to Inversion

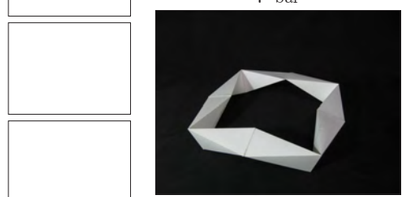

Half Inversion

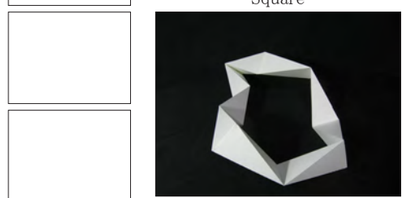

Over Angle

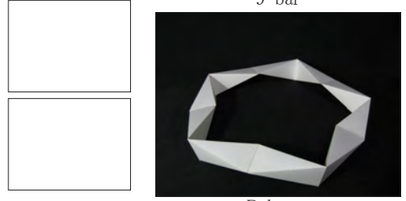

Change to Minimum


Not Inversion

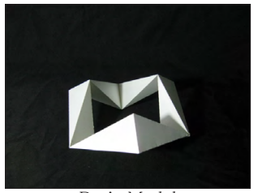

7-bar

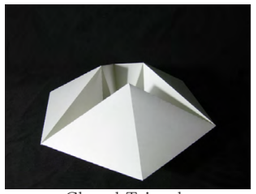

Square

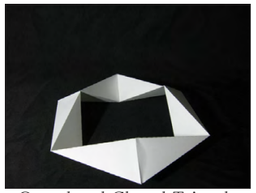

9-bar

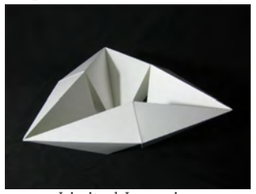

Polygon

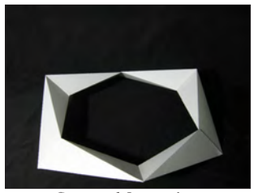
４－１－Ｅ．特殊モデル： 止められた回転

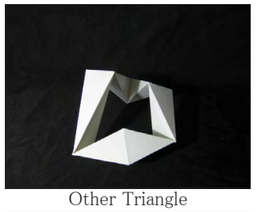

Basic Model

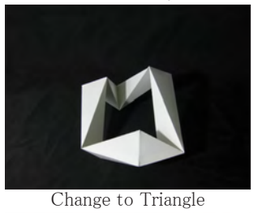

Closed Triangle


Opend and Closed Triangle

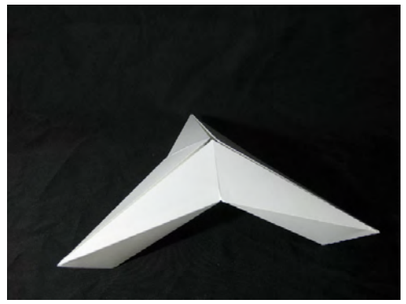

Limited Inversion

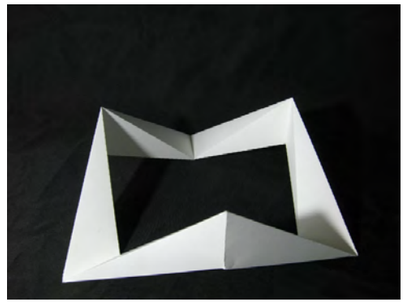

Stopped Inversion

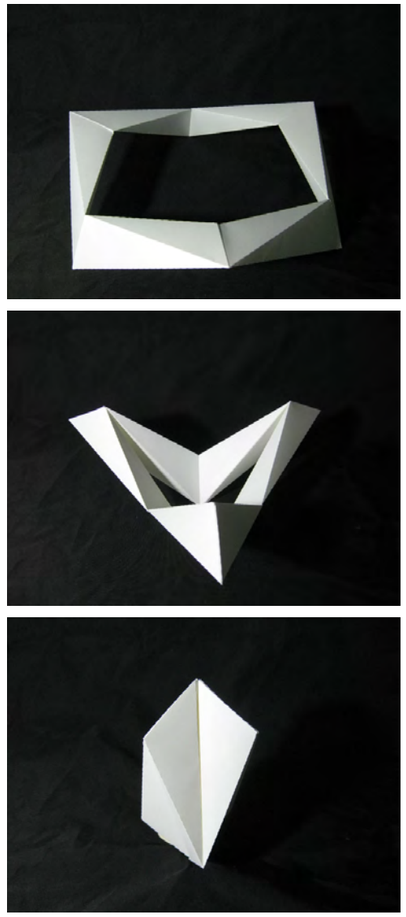

Other Triangle


Change to Triangle

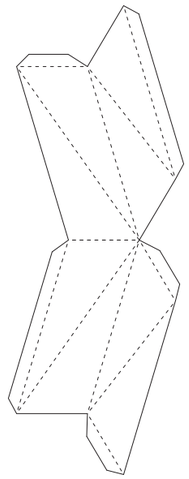

Change to Line

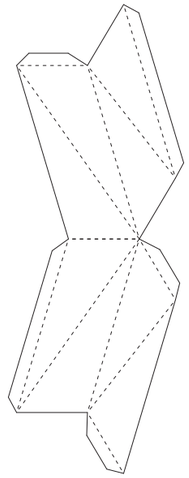


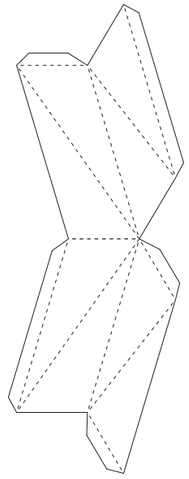




回転写真



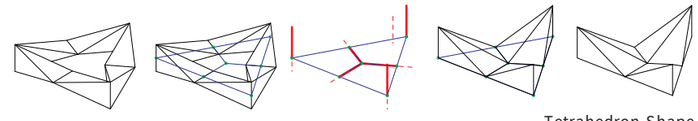


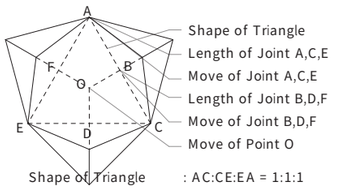


展開図

4－1－E．Another Model: Stopped Inversion

隙間を埋めたときに、結果として四面体同士がぶつかりそこで反転運動がストップしてしまうものの中で、さらに特殊な事例。一連の変形のほかに形態が成り立つ変形部分が存在するもの。

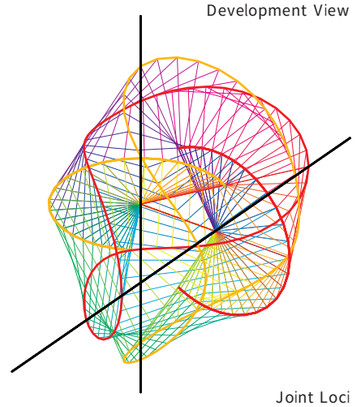

Tetrahedron Shape



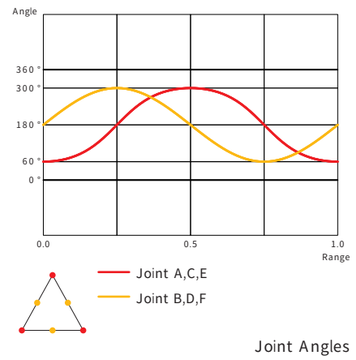
Shape of Triangle : AC:CE:EA = 1:1:1
Length of Joint A,C,E : AC * √3 / 6
Z Move of Joint A,C,E : AC * ((6 - √3) /12)
Length of Joint B,D,F : AC * √3 / 6
XY Move of Joint B,D,F : AC * ((6 - √3) /12)
Z Move of Point O : 0
XY Move of Point O : AC:AO:CO = √3:1:1

Metamorphosis Parameters

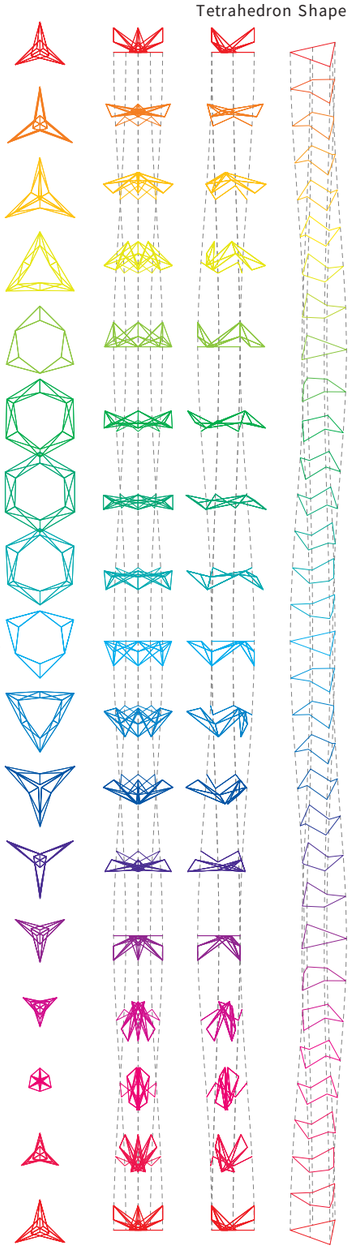
Development View



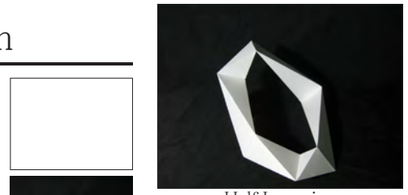
Joint Loci

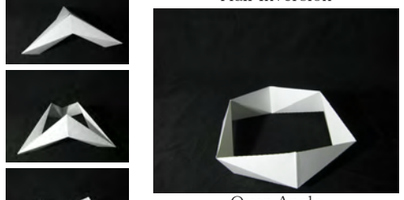


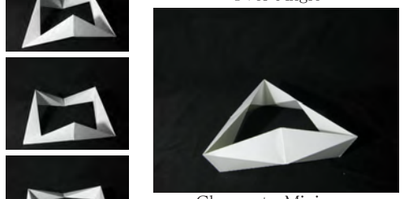
Joint Angles

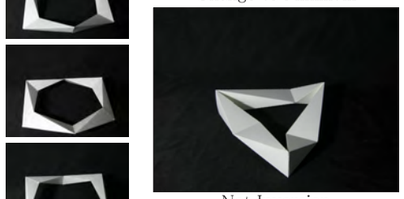


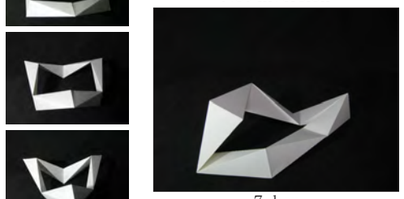
How to Inversion

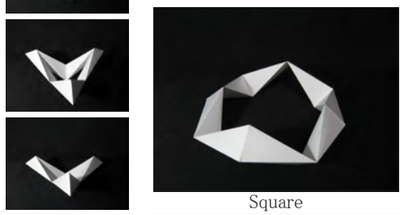

Half Inversion

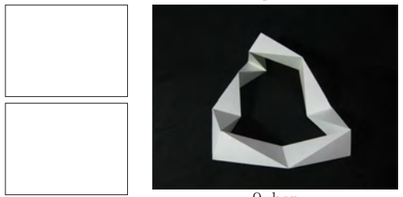

Over Angle

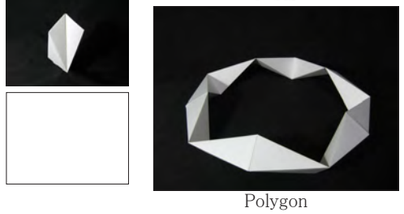

Change to Minimum


Not Inversion


7-bar

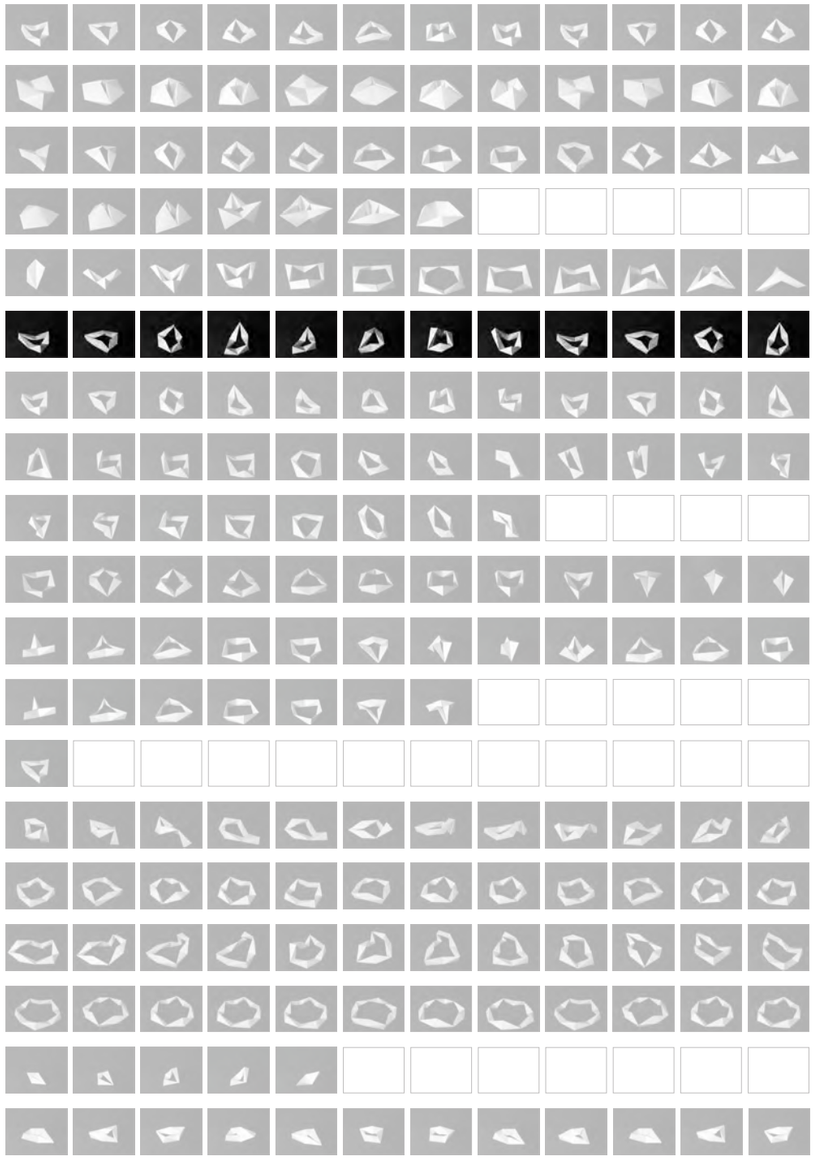

Square

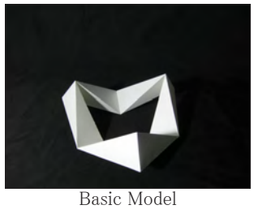

9-bar

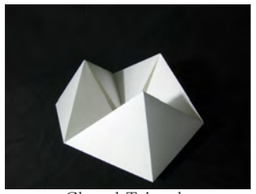

Polygon

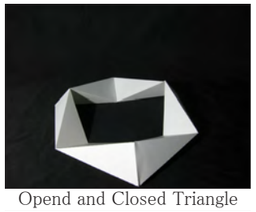
４－２．蝶番の距離

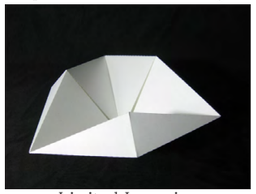
ドア2つを1組の辺としたとき、基準の三角形を得られるがその辺の長さ、つまり蝶番同士の距離を変える。

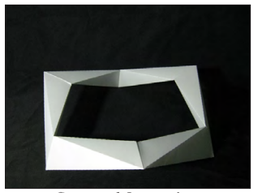



Basic Model

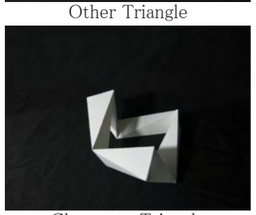

Closed Triangle

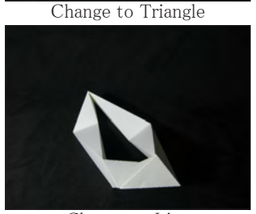

Opend and Closed Triangle


Limited Inversion


Stopped Inversion


Other Triangle


Change to Triangle

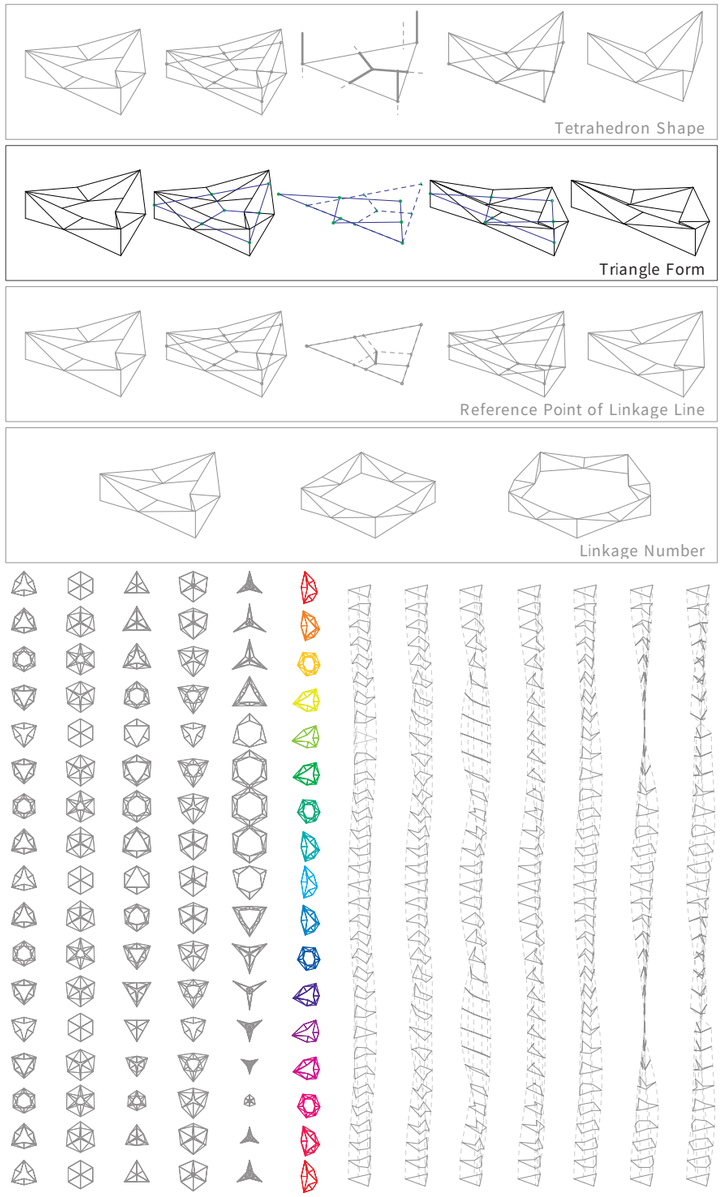

Change to Line

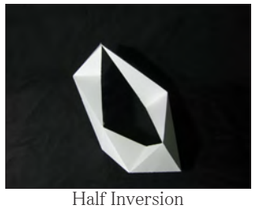
4－2．Triangle Form

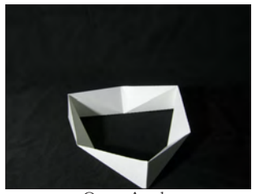
基準三角形を変形させる。

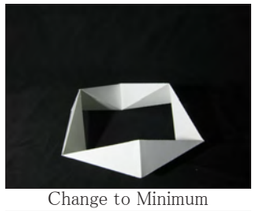
4－2－A．Basic Model: Equilateral Triangle

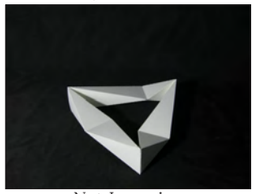
4－2－B．Application Model: Other Triangle




Half Inversion

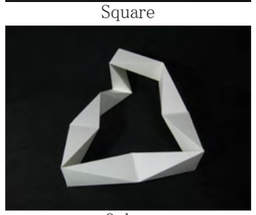

Over Angle

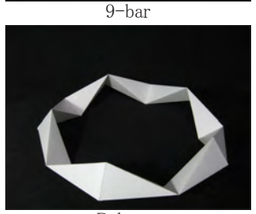

Change to Minimum


Not Inversion

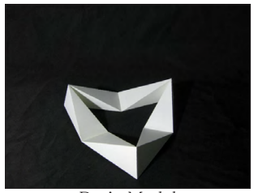

7-bar

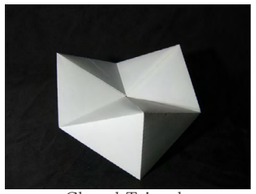

Square

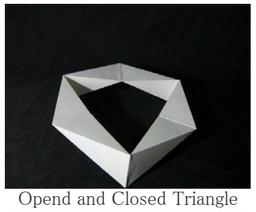

9-bar

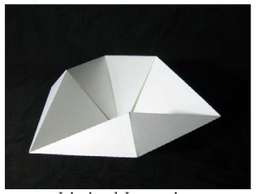

Polygon

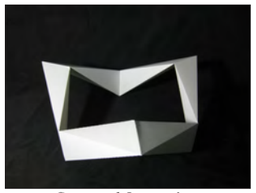
４－２－Ａ．基本モデル： 正三角形


Basic Model

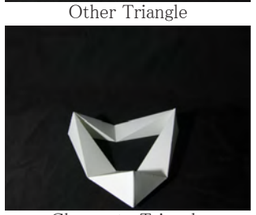

Closed Triangle

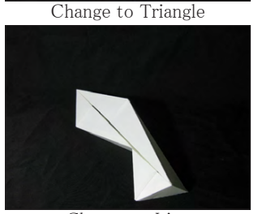

Opend and Closed Triangle

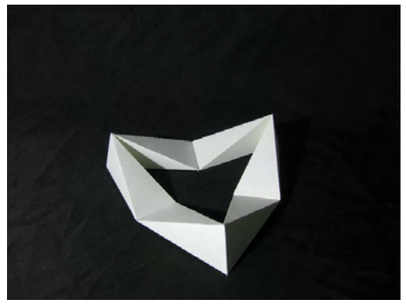

Limited Inversion

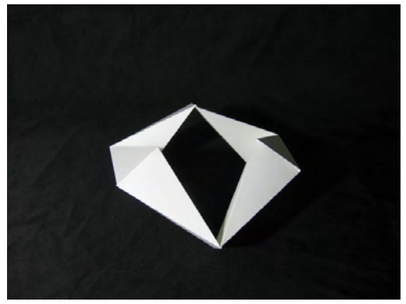

Stopped Inversion

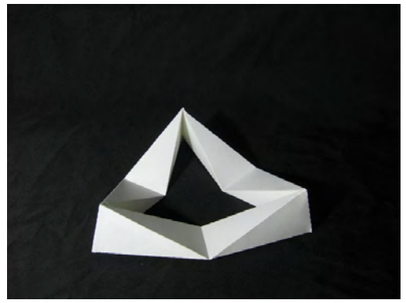

Other Triangle

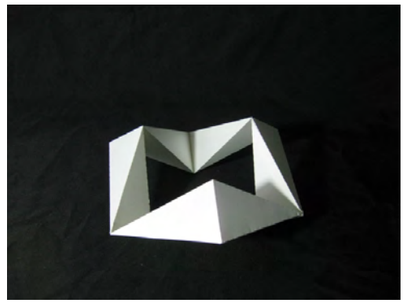

Change to Triangle

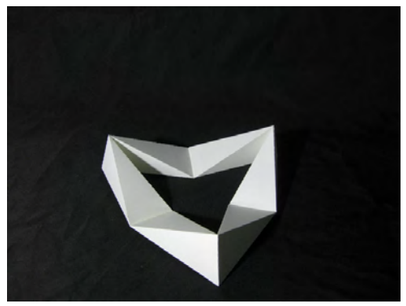

Change to Line



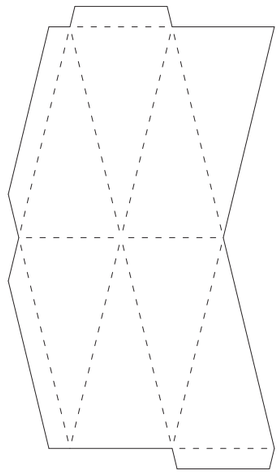


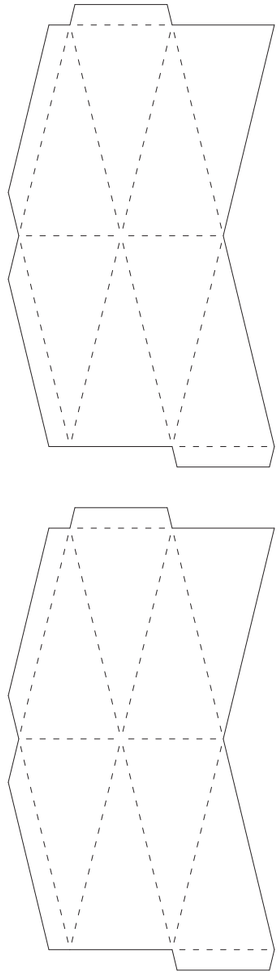






回転写真

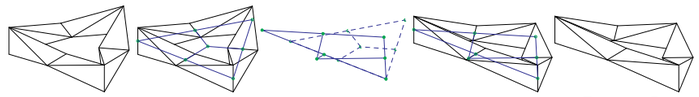


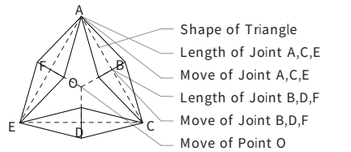


展開図

4－2－A．Basic Model: Equilateral Triangle

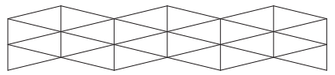
基準三角形が正三角形のときは、反転運動中に現れる4つの平面の三角形は全て同じ形をとる。

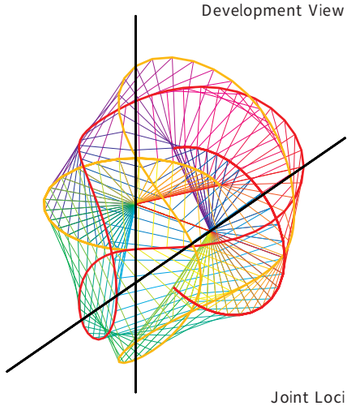

Triangle Form

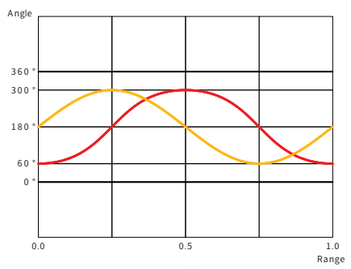


Shape of Triangle : AC:CE:EA = 1:1:1
Length of Joint A,C,E : AC * 1 / 4
Z Move of Joint A,C,E : 0
Length of Joint B,D,F : AC * 1 / 4
XY Move of Joint B,D,F : 0
Z Move of Point O : 0
XY Move of Point O : AC:AO:CO = √3:1:1

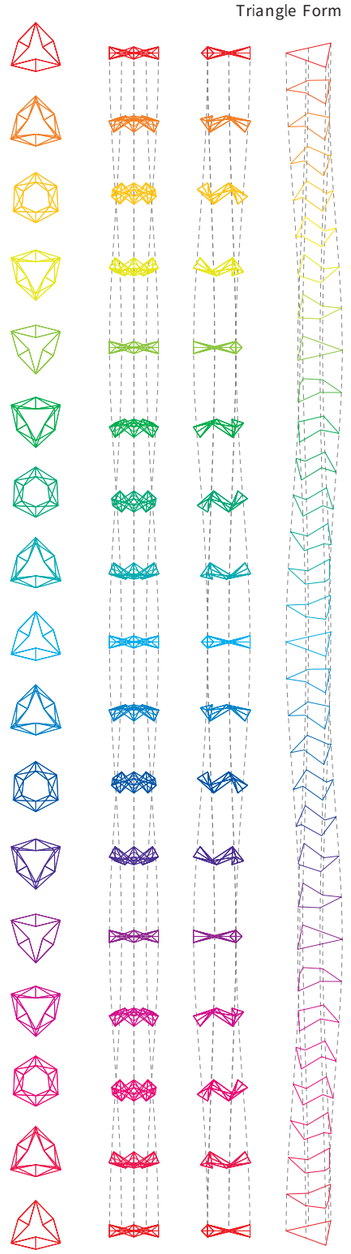
Metamorphosis Parameters

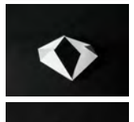

Development View

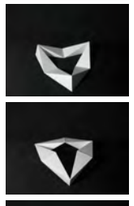

Joint Loci

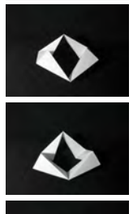


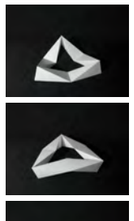
Joint Angles

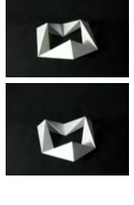

How to Inversion

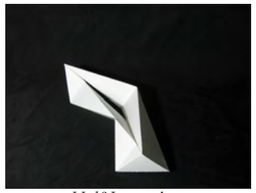


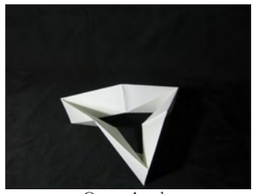









Half Inversion


Over Angle

Change to Minimum

Not Inversion

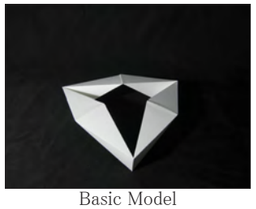
7-bar

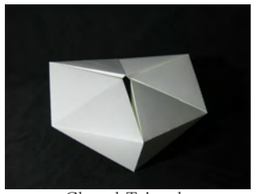
Square

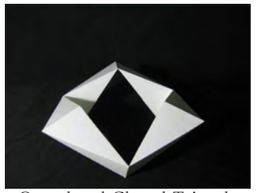
9-bar

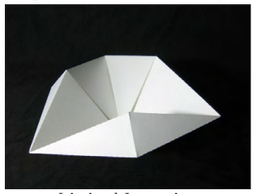
Polygon

４－２－Ｂ．発展モデル： その他の三角形

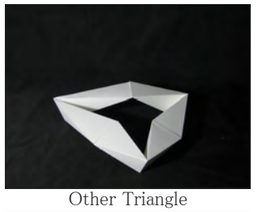

Basic Model

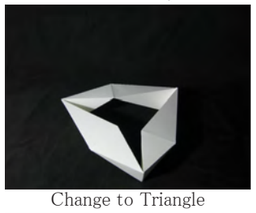

Closed Triangle

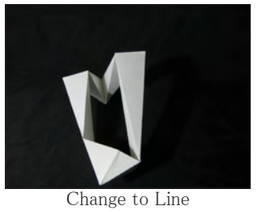

Opend and Closed Triangle

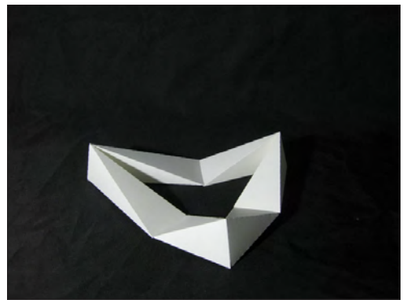

Limited Inversion

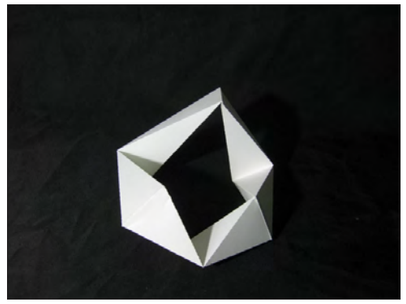

Stopped Inversion

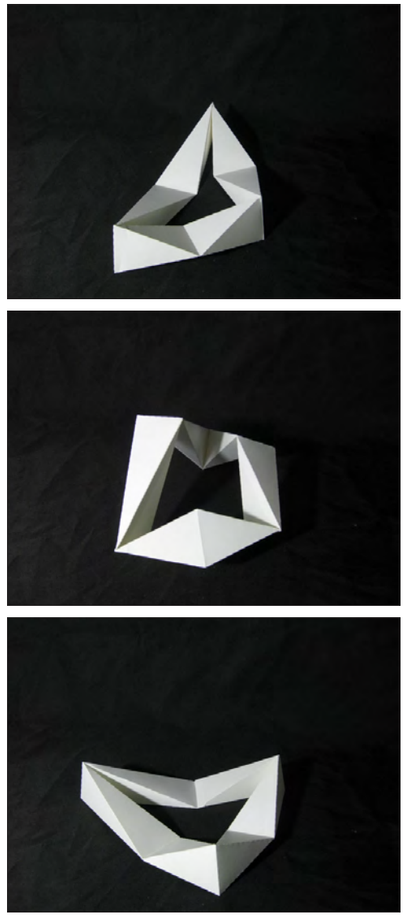

Other Triangle


Change to Triangle

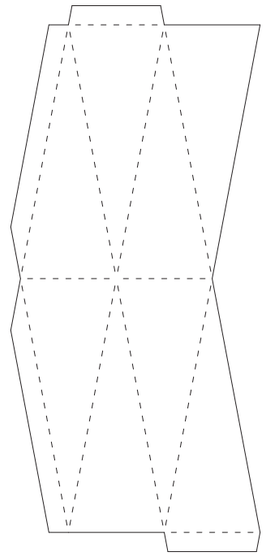

Change to Line

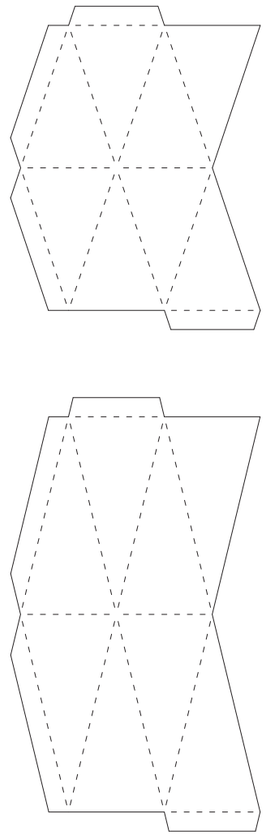






回転写真

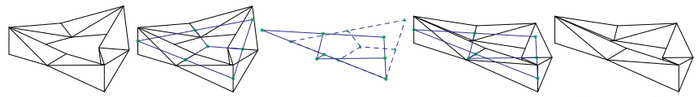


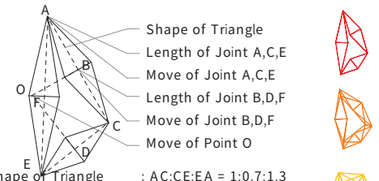


展開図

4－2－B．Application Model: Other Triangle

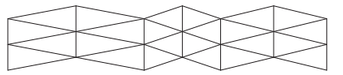
基準三角形を正三角形以外にすると、反転運動中に現れる4つの平面の三角形は、2種類の形をとり、それぞれが交互に現れる。また、2組の三角形はそれぞれで平面状で反転した形をとる。

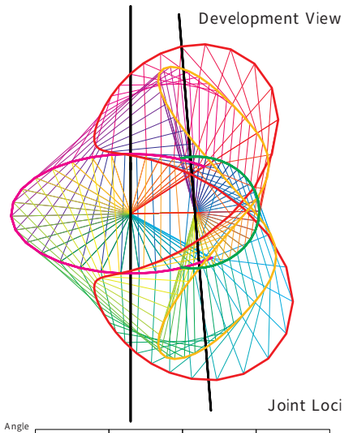

Triangle Form

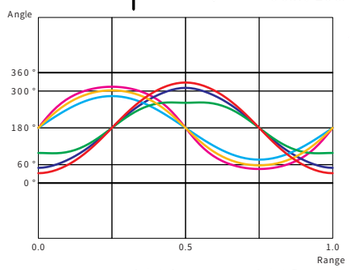


Shape of Triangle : AC:CE:EA = 1:0.7:1.3
Length of Joint A,C,E : AC * 1 / 4
Z Move of Joint A,C,E : 0
Length of Joint B,D,F : AC * 1 / 4
XY Move of Joint B,D,F : 0
Z Move of Point O : 0
XY Move of Point O : Circumcenter Point

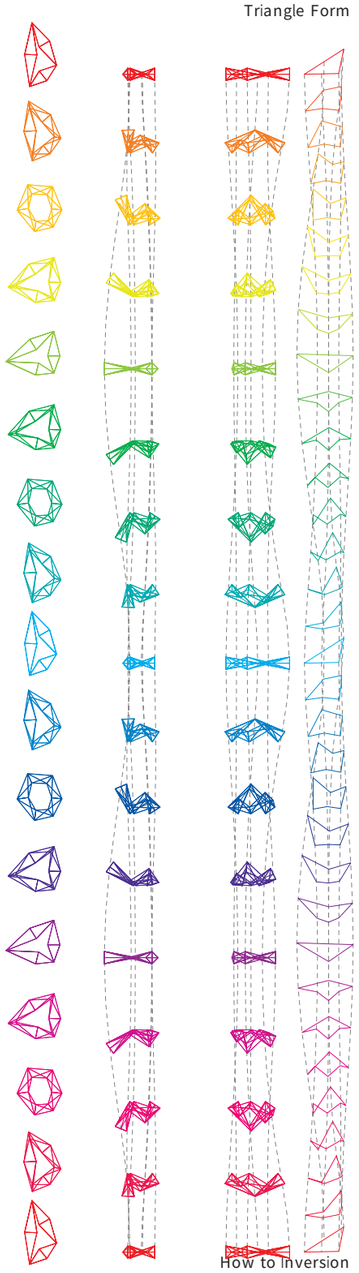
Metamorphosis Parameters

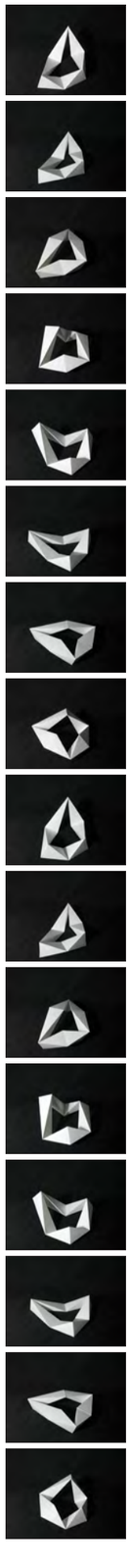


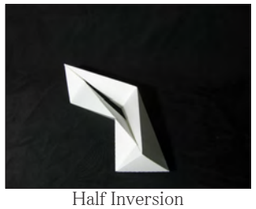




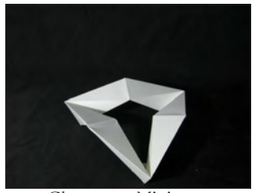
Joint Angles

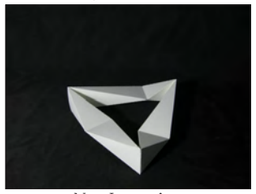

How to Inversion



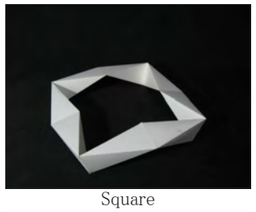

Half Inversion

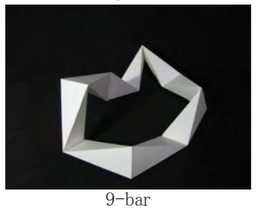

Over Angle

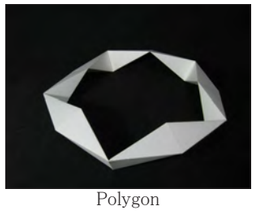

Change to Minimum

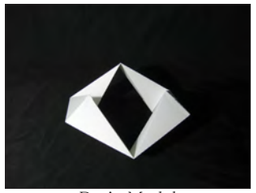

Not Inversion

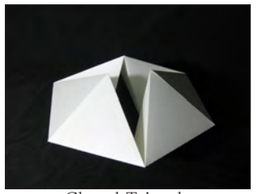

7-bar

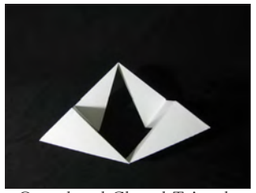

Square

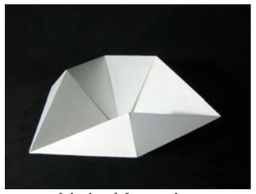

9-bar


Polygon


Basic Model

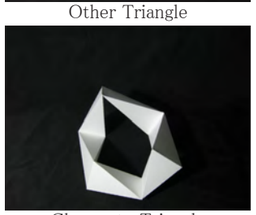

Closed Triangle

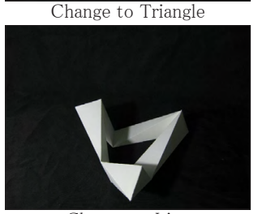

Opend and Closed Triangle


Limited Inversion


Stopped Inversion

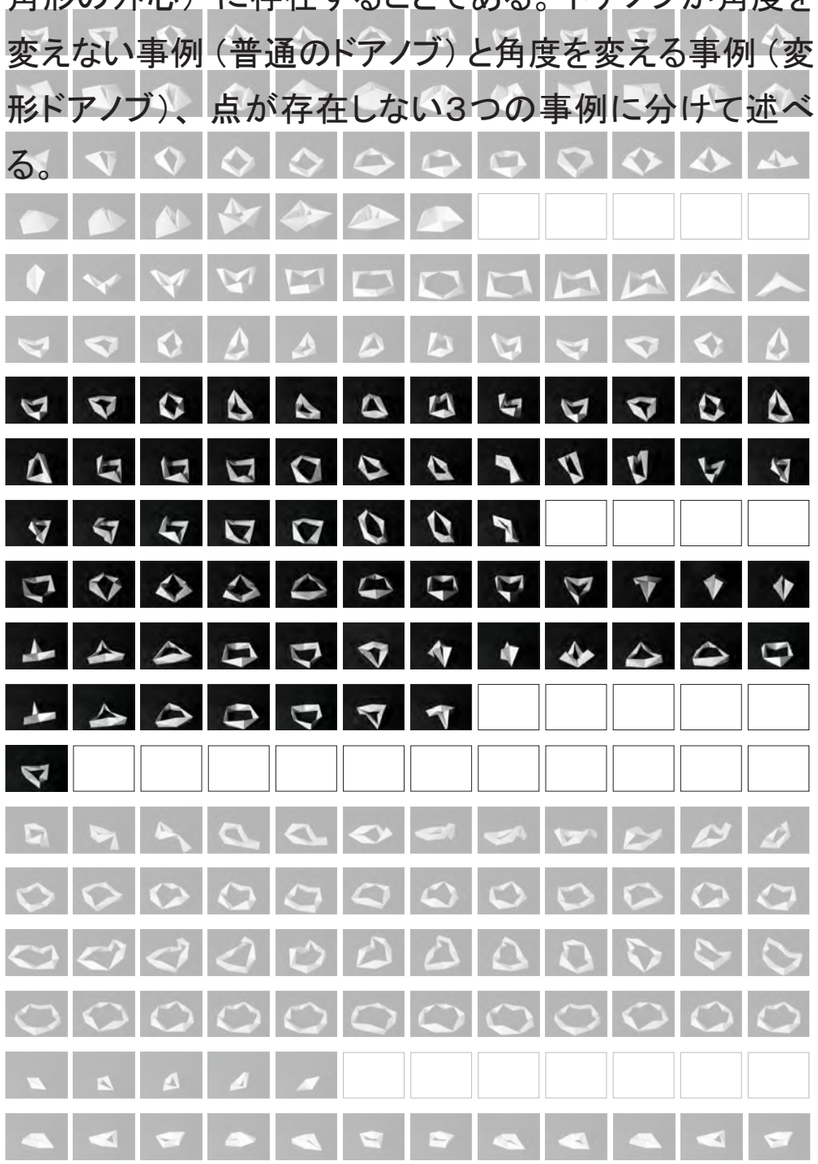

Other Triangle


Change to Triangle


Change to Line

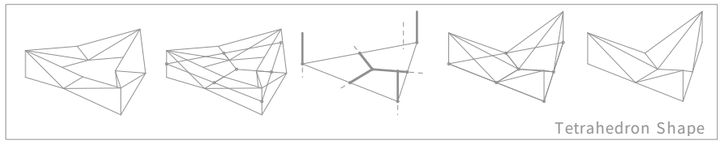
４－３．蝶番とドアノブの距離と角度関係

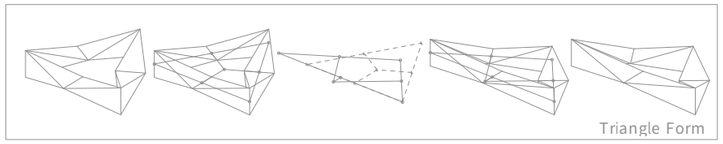
蝶番とドアノブの距離と角度関係を変化させることによって、回転の仕方が変わる。その際、回転を起こすための条件があり、ひとつはドアノブの回転ライン3つが1点で交わること、もうひとつはドアノブがドアに対して角度を変えない状態か、ドアノブが角度を変えてその回転ラインの一点がドアの作る三角柱の中心ライン（三角形の外心）に存在することである。ドアノブが角度を変えない事例（普通のドアノブ）と角度を変える事例（変形ドアノブ）、点が存在しない3つの事例に分けて述べる。

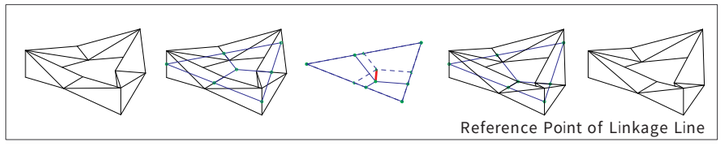


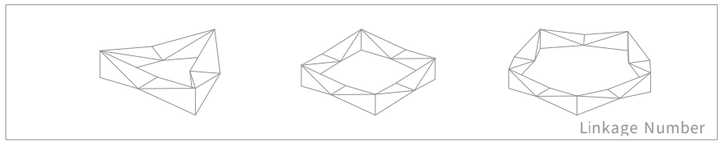
４－３．Reference Point of Linkage Line

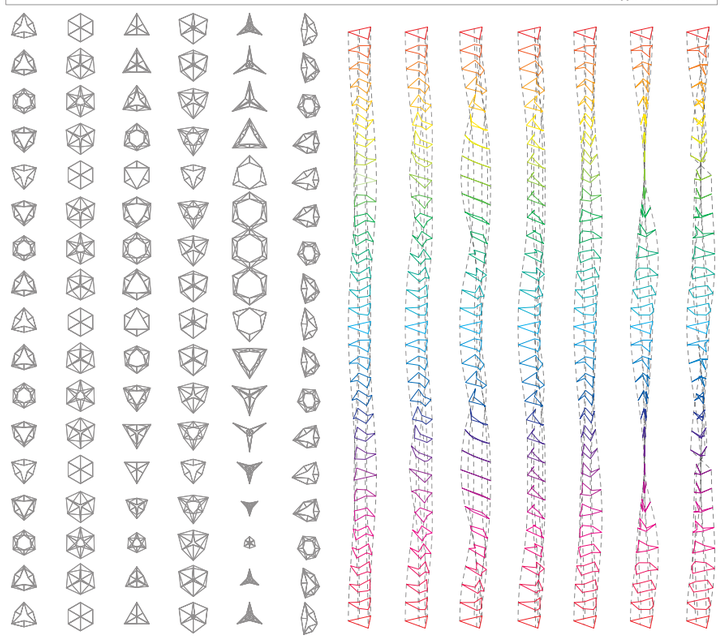
基準三角形で辺上に存在するピンジョイントの位置とその角度関係を変化させることによって、反転運動が変わる。ピンジョイントの位置と角度を決める際、大きく3つの事例に分けることが出来る。基準三角形を見たときに各頂点のピンジョイントのラインは三角形が作る平面に直交するものとおく時、残りの辺上に存在する3つのピンジョイントのラインは1点で交わる状態でないと反転運動が起こらないことがわかっている。さらにもうひとつの条件として、その点が三角形が作る平面状に存在する状態（Point on Triangle Plane）か、三角形の外心線上（三角形の外心からその三角形が作る平面に直行する線）に存在する状態（Point on Triangle Circumcenter Line）でないと反転運動が起こらないと思われる。以上の "Point on Triangle Plane"、"Point on Triangle Circumcenter Line" に1点で交わらない "Not Exist Point" の3つの事例に分けて述べる。

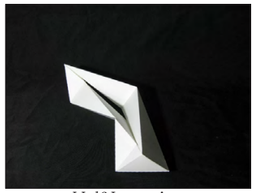




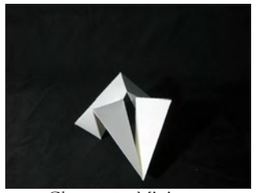


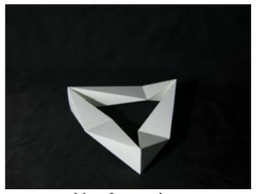


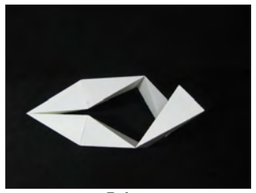



Half Inversion

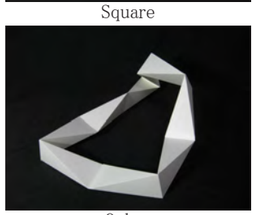

Over Angle

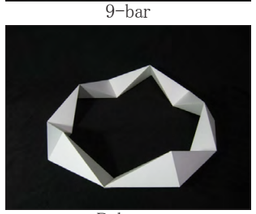

Change to Minimum

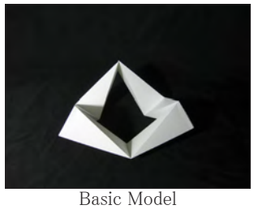

Not Inversion

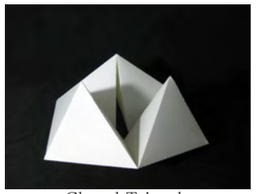

7-bar


Square

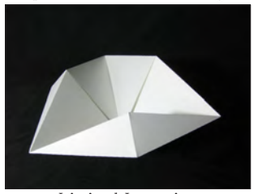

9-bar


Polygon

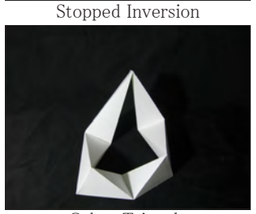

Basic Model

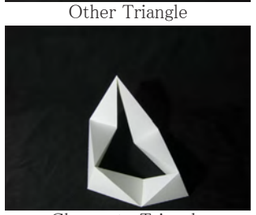

Closed Triangle

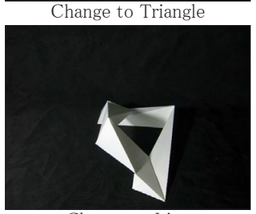

Opend and Closed Triangle


Limited Inversion


Stopped Inversion


Other Triangle


Change to Triangle


Change to Line

４－３－１．普通のドアノブ

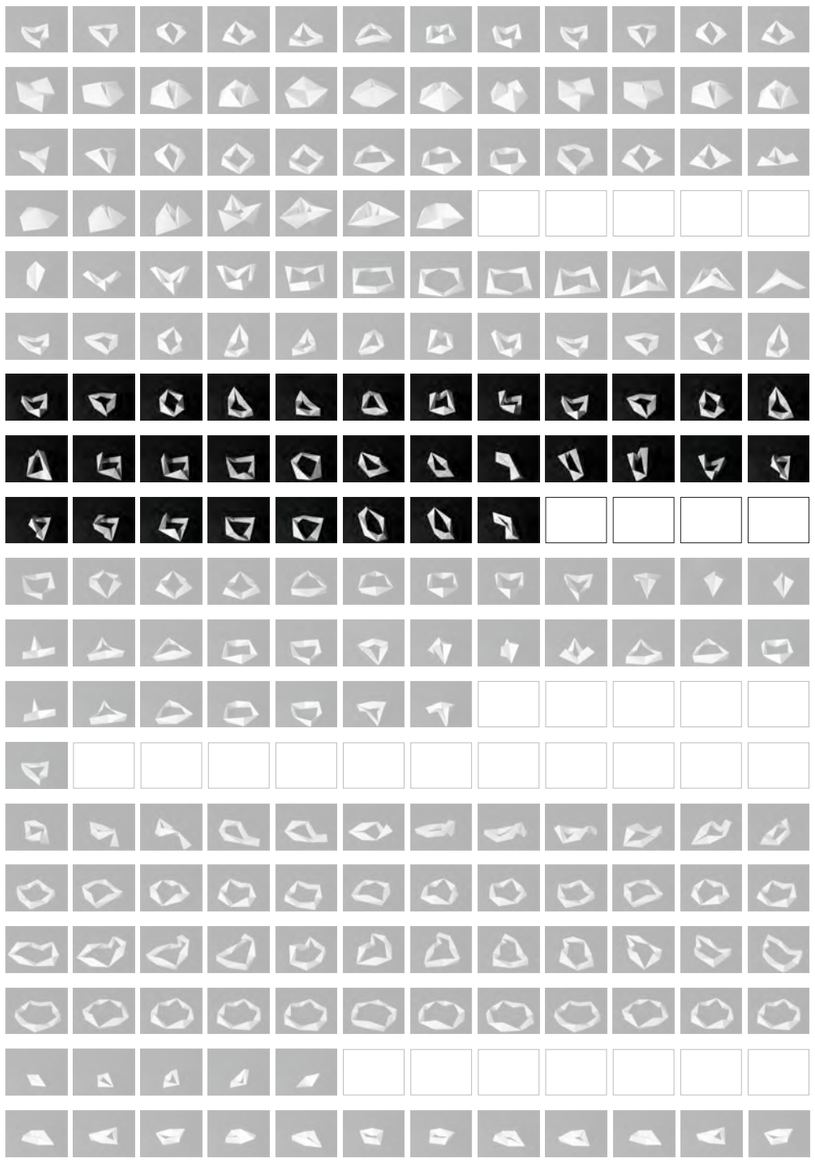
ドアノブが角度を変えない事例。

４－３－１－A．基本モデル：外心

４－３－１－B．発展モデル：三角形から三角形

４－３－１－C．発展モデル：三角形から直線

４－３－１－D．発展モデル：半分の回転



4－3－1．Point on Triangle Plane

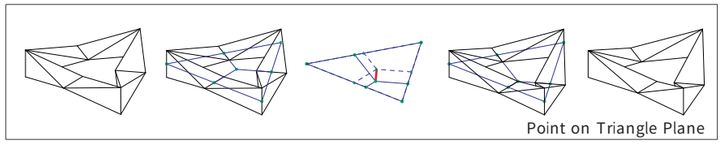
基準点が三角形の平面上に存在する事例。反転運動中に4回、平面（三角形）状態をとる。

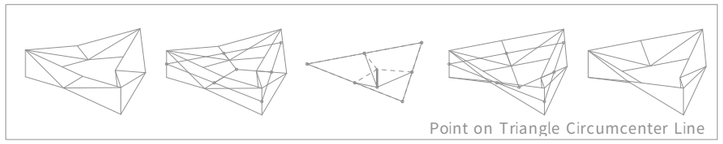
4－3－1－A．Basic Model: Circumcenter Point

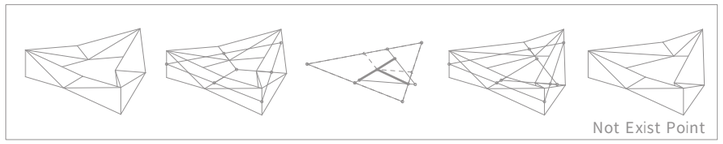
4－3－1－B．Application Model: Change to Triangle

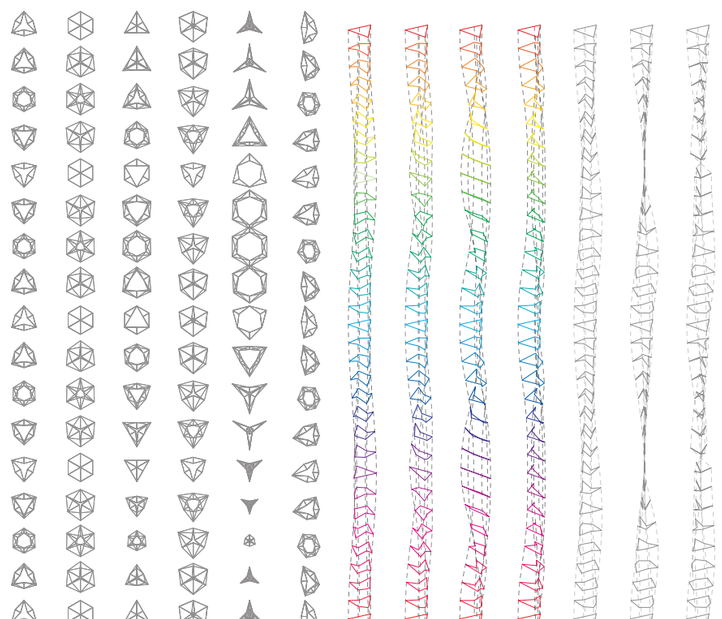
4－3－1－C．Application Model: Change to Line

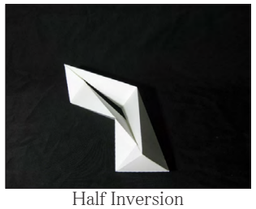
4－3－1－D．Application Model: Half Inversion



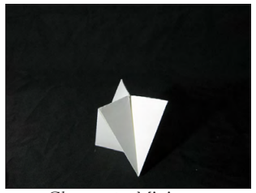


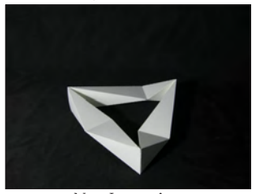




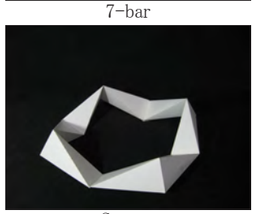

Half Inversion

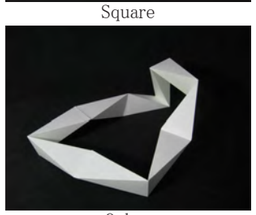

Over Angle

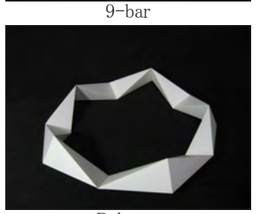

Change to Minimum


Not Inversion

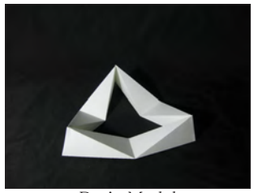

7-bar

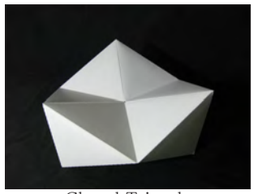

Square

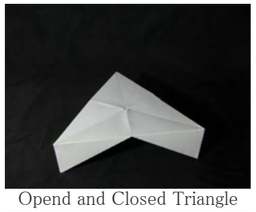

9-bar

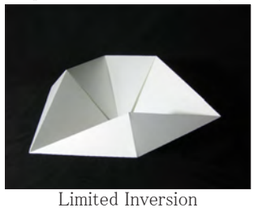

Polygon

４－３－１－Ａ．基本モデル： 外心


Basic Model

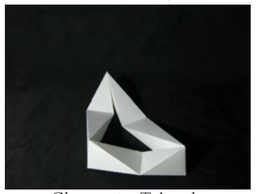

Closed Triangle

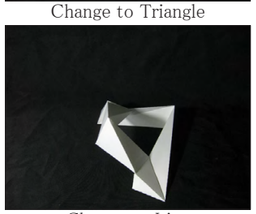

Opend and Closed Triangle

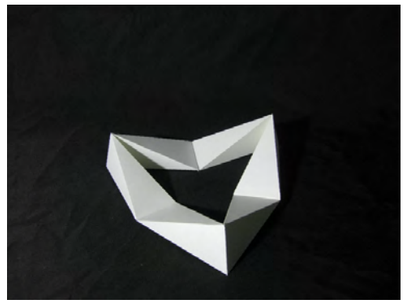

Limited Inversion

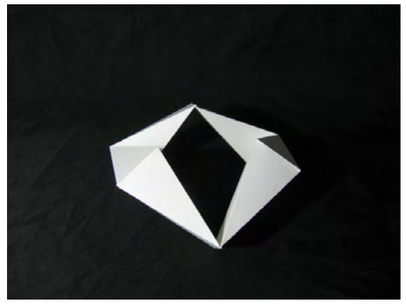

Stopped Inversion

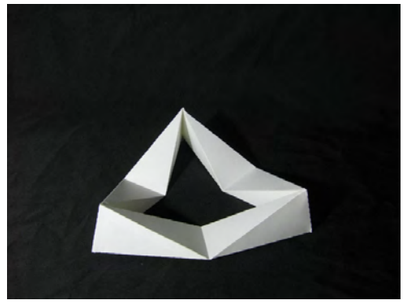

Other Triangle

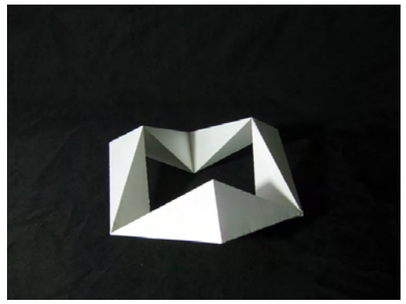

Change to Triangle

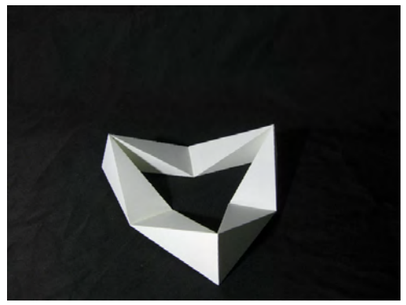

Change to Line



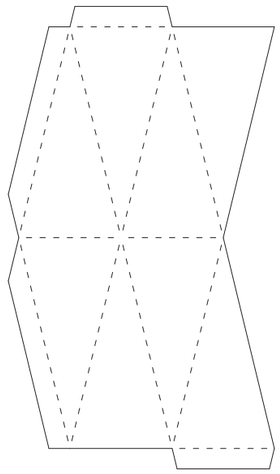


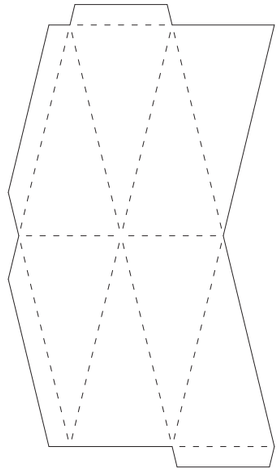


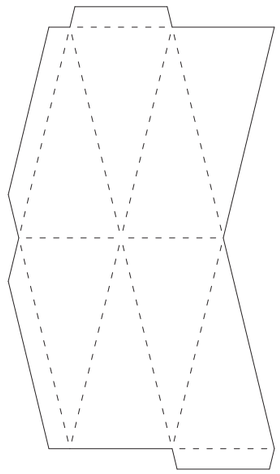




回転写真



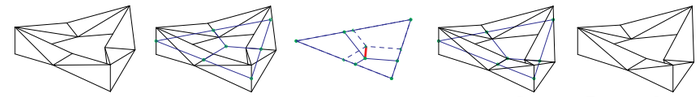


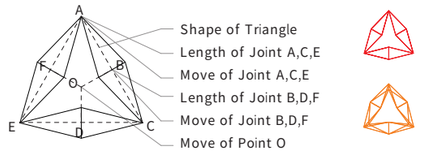


展開図

4－3－1－A．Basic Model: Circumcenter Point

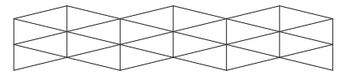
"Point on Triangle Plane"、"Point on Triangle Circumcenter Line" 両方に該当する事例。変形としては "Point on Triangle Plane" の変形が起こる。正三角形の場合、反転運動中に現れる4つの三角形は全て正三角形となる。

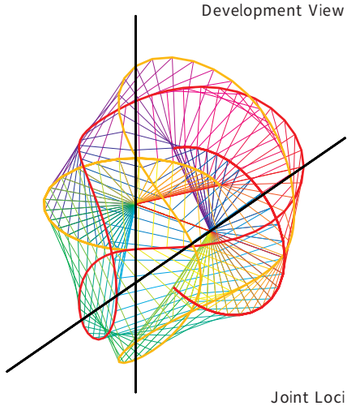

Point on Triangle Plane

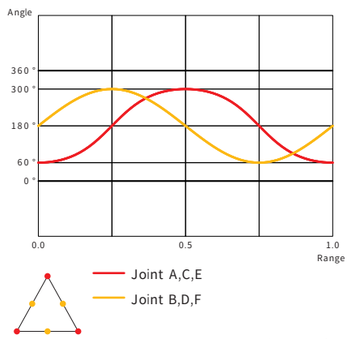


Shape of Triangle : AC:CE:EA = 1:1:1
Length of Joint A,C,E : AC * 1 / 4
Z Move of Joint A,C,E : 0
Length of Joint B,D,F : AC * 1 / 4
XY Move of Joint B,D,F : 0
Z Move of Point O : 0
XY Move of Point O : AC:AO:CO = √3:1:1

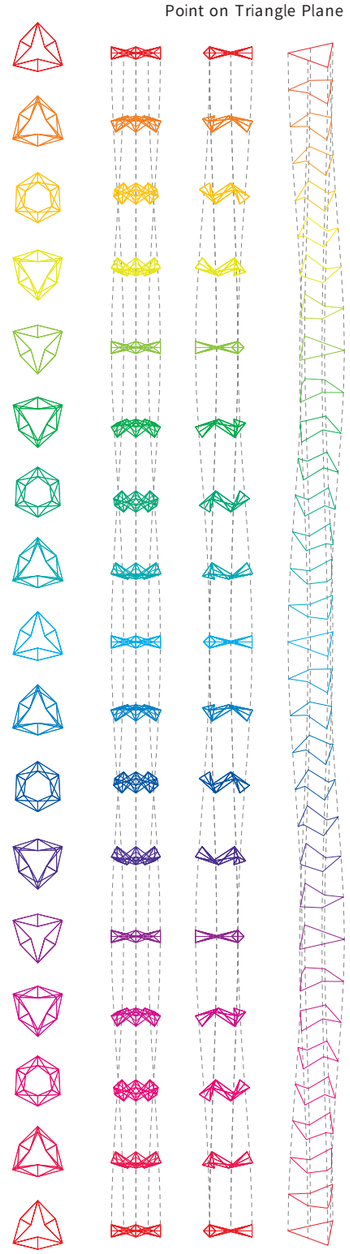
Metamorphosis Parameters


Development View


Joint Loci

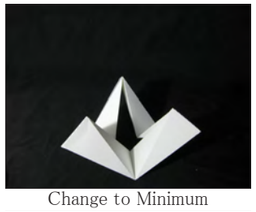


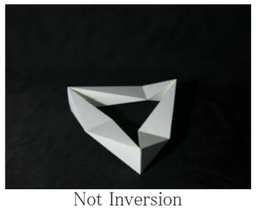
Joint Angles

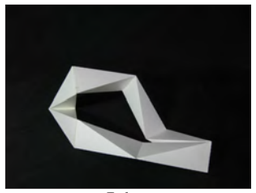

How to Inversion

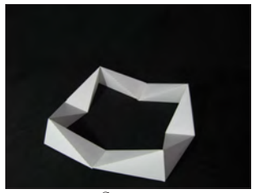

Half Inversion

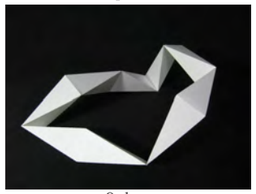

Over Angle

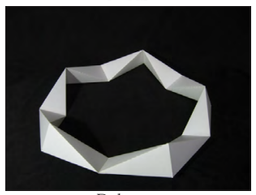

Change to Minimum


Not Inversion

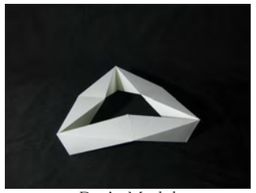

7-bar

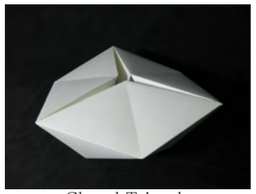

Square


9-bar

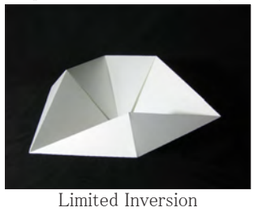

Polygon

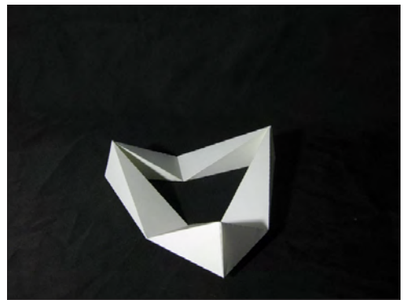
４－３－１－Ｂ．発展モデル：三角形から三角形

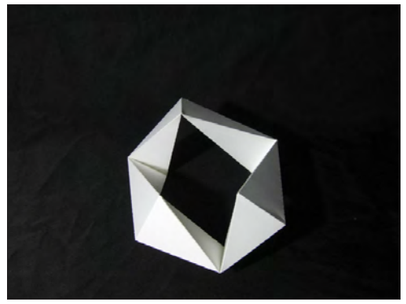

Basic Model

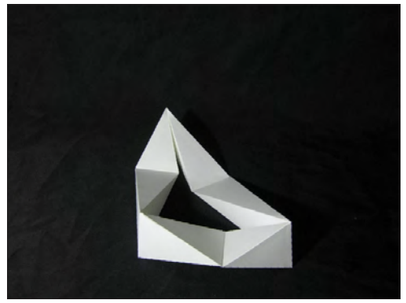

Closed Triangle

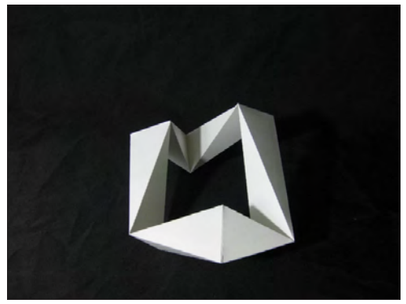

Opend and Closed Triangle

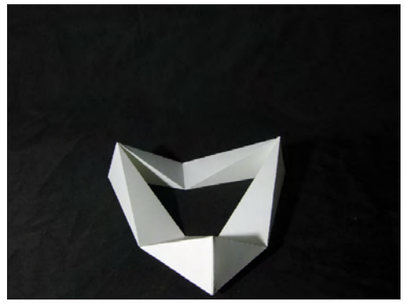

Limited Inversion

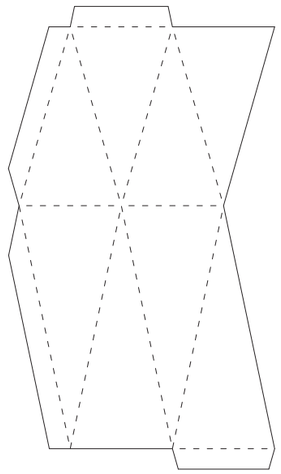


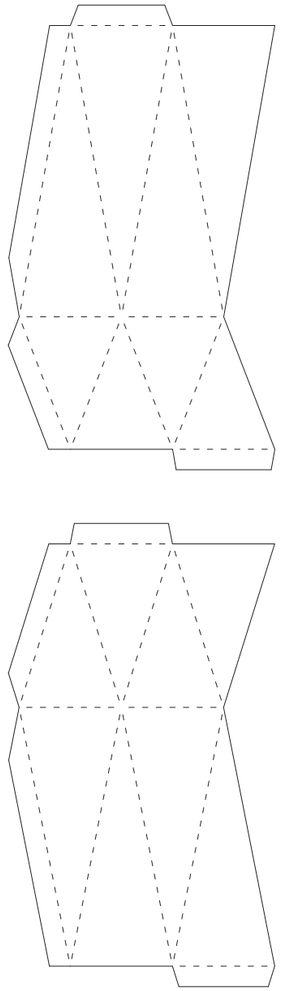








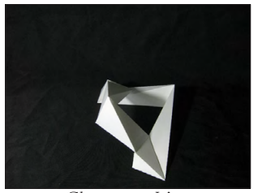




Stopped Inversion


Other Triangle

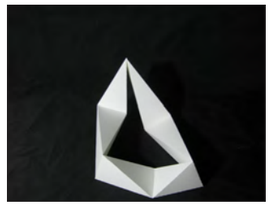

Change to Triangle


Change to Line

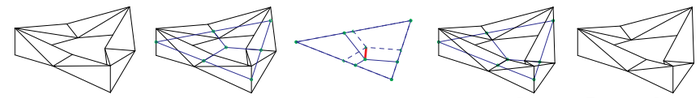
回転写真

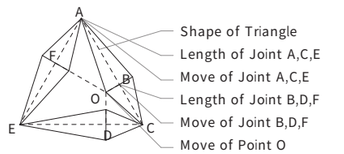
展開図

4－3－1－B. Application Model: Change to Triangle



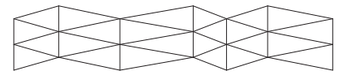
基準点が外心から見て一定以内の距離の場合に起こる。正三角形の場合、基準点が三角形内に存在する。基準三角形を正三角形以外にすると、反転運動中に現れる4つの平面の三角形は、2種類の形をとり、それぞれが交互に現れる。また、2組の三角形はそれぞれで平面状で反転した形をとる。


Point on Triangle Plane

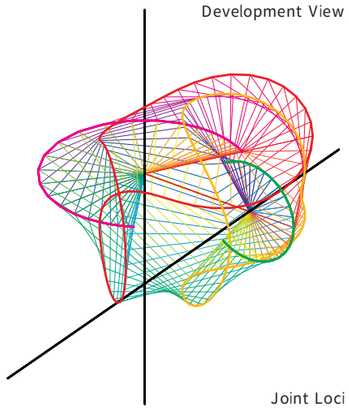


Shape of Triangle : AC:CE:EA = 1:1:1
Length of Joint A,C,E : AC * 1 / 4
Z Move of Joint A,C,E : 0
Length of Joint B,D,F : AC * 1 / 4
XY Move of Joint B,D,F : 0
Z Move of Point O : 0
XY Move of Point O : AC:CO:EO = 1:0.4:0.8

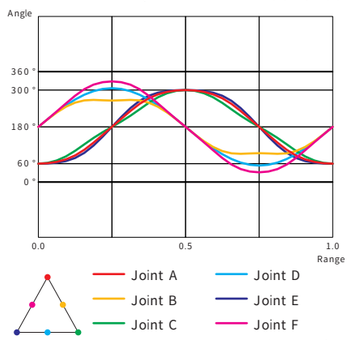
Metamorphosis Parameters



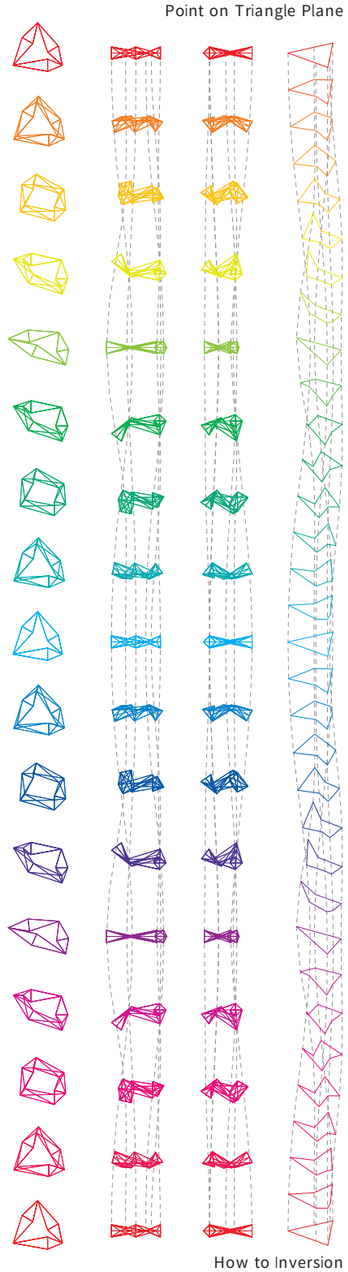
Development View



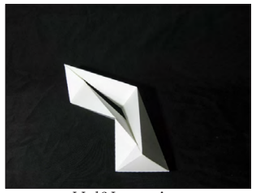
Joint Loci



Joint Angles

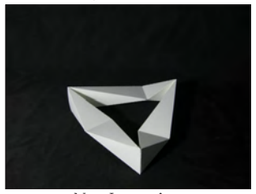


How to Inversion


Half Inversion


Over Angle

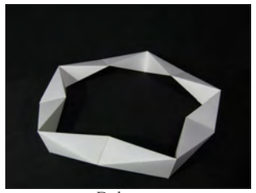

Change to Minimum


Not Inversion

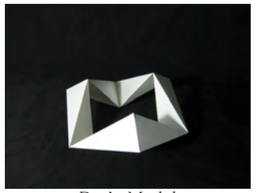

7-bar

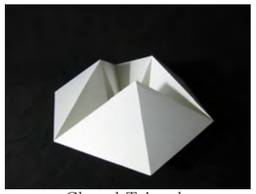

Square

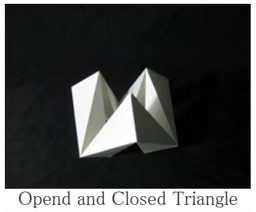

9-bar

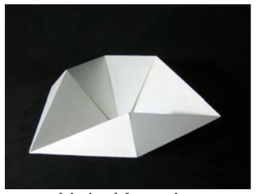

Polygon

４－３－１－Ｃ．発展モデル：三角形から直線


Basic Model

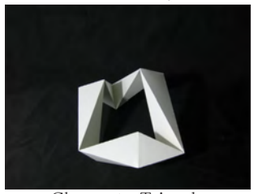

Closed Triangle

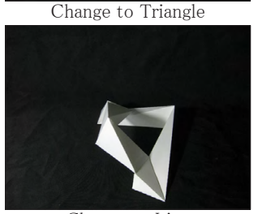

Opend and Closed Triangle

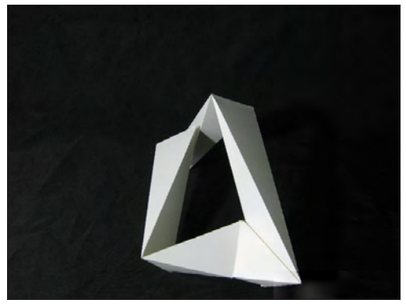

Limited Inversion

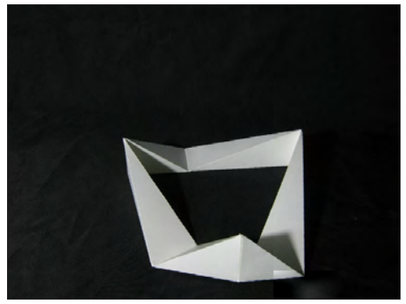

Stopped Inversion

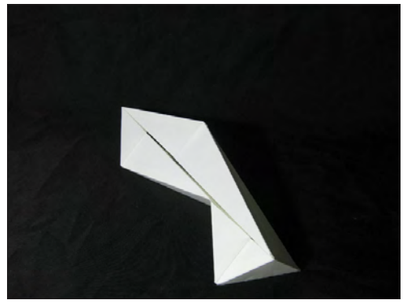

Other Triangle

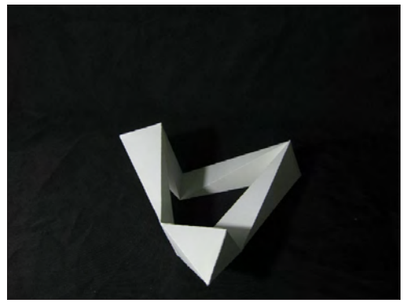

Change to Triangle

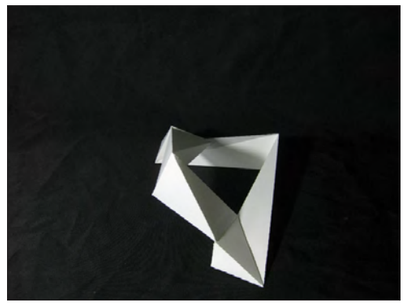

Change to Line



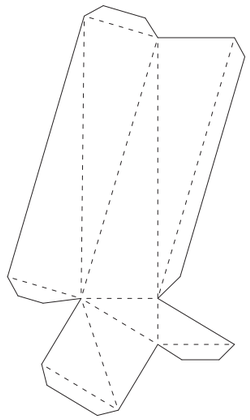


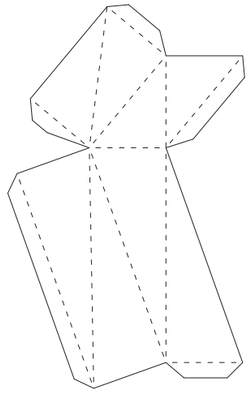


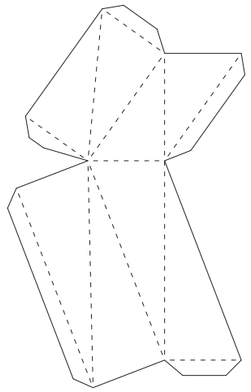




回転写真

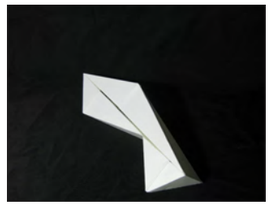




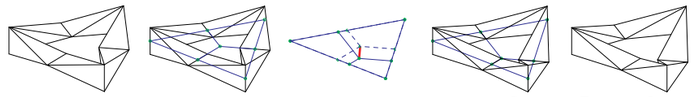


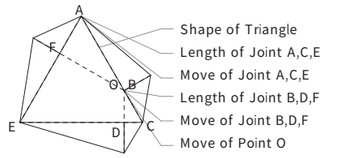
展開図

4－3－1－C．Application Model: Change to Line



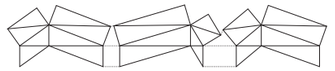
基準点が外心から見て一定以上の距離の場合に起こる。正三角形の場合、基準点が辺上に存在する。反転運動中に現れる三角形4つのうち2つが直線状になる。

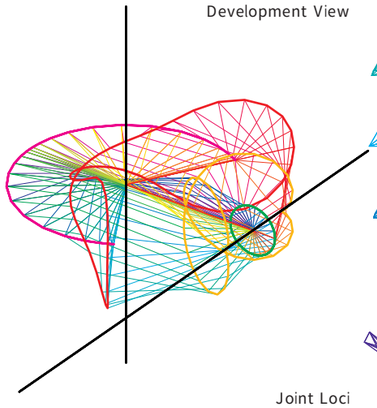

Point on Triangle Plane

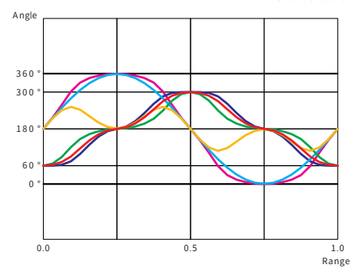


Shape of Triangle : AC:CE:EA = 1:1:1
Length of Joint A,C,E : AC * 1 / 4
Z Move of Joint A,C,E : AC * 1 / 8
Length of Joint B,D,F : AC * 1 / 4
XY Move of Joint B,D,F: AC * -1 / 8
Z Move of Point O : 0
XY Move of Point O : AC:AO:CO = 1:0.7:0.3

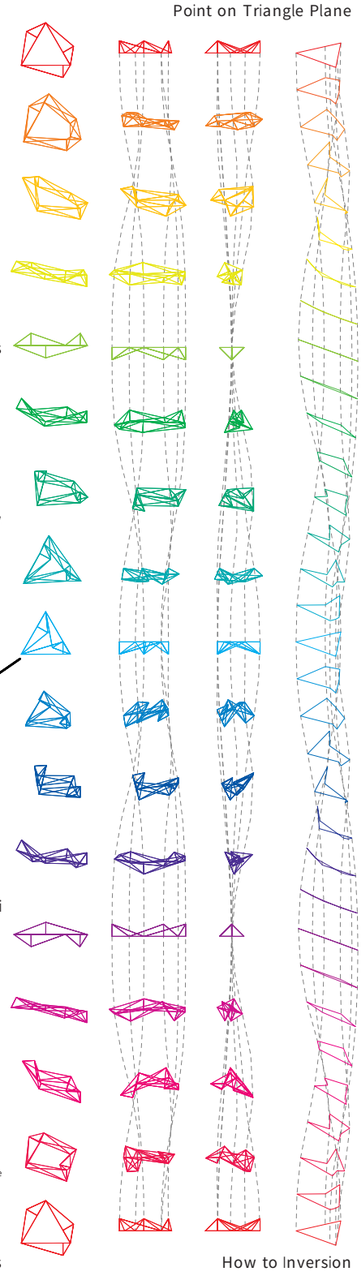
Metamorphosis Parameters

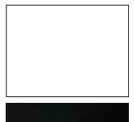


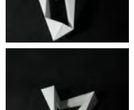

Development View

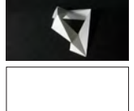


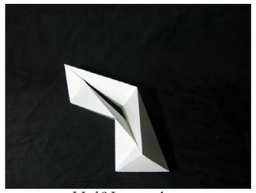
Joint Angles


How to Inversion








Half Inversion


Over Angle


Change to Minimum

Not Inversion

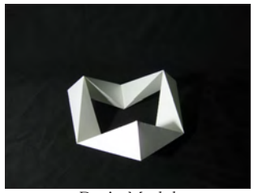
7-bar

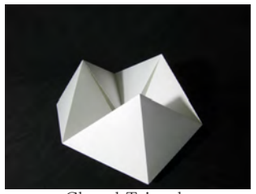
Square

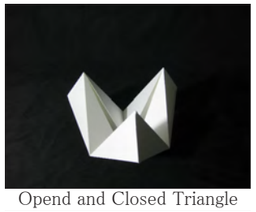
9-bar

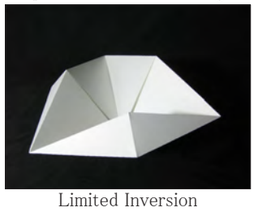
Polygon

４－３－１－Ｄ．発展モデル： 半分の回転

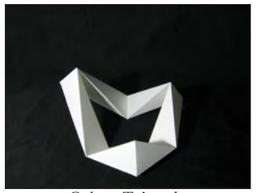

Basic Model


Closed Triangle

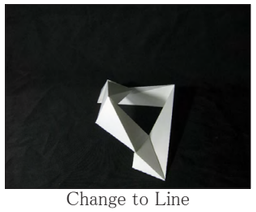

Opend and Closed Triangle

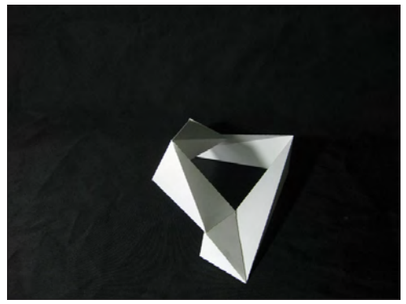

Limited Inversion

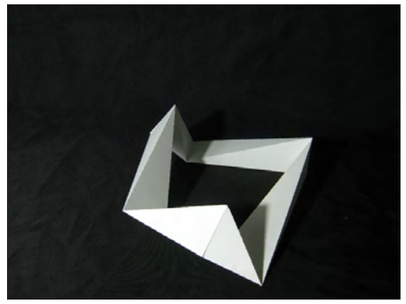
Stopped Inversion

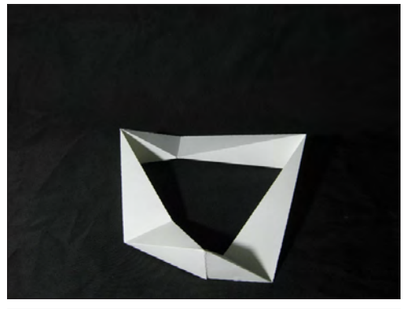

Other Triangle

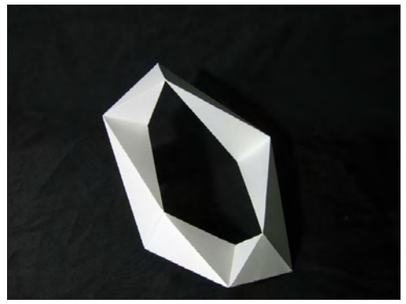

Change to Triangle

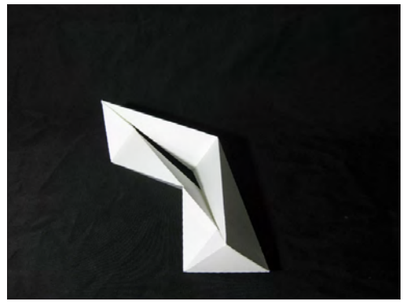

Change to Line



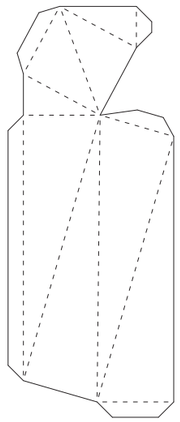


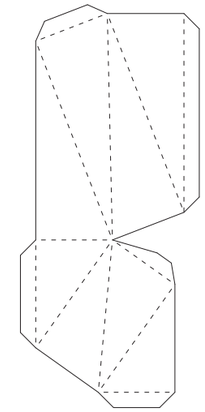


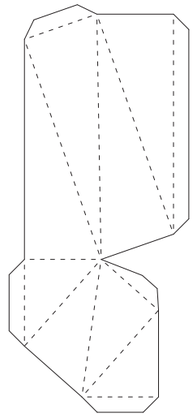




回転写真



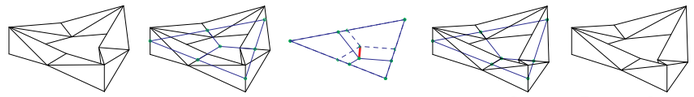


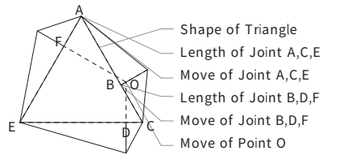


展開図

4－3－1－D．Application Model: Half Inversion

基準点が外心から見てＣの事例よりも距離を持つ場合に起こる。正三角形の場合、基準点が三角形外に存在する。反転運動量が半分になる。

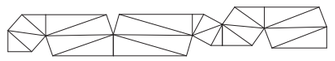

Point on Triangle Plane



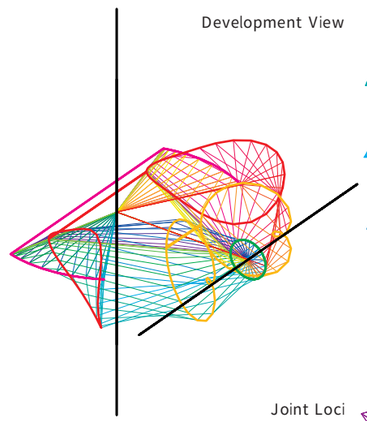
Shape of Triangle
Length of Joint A,C,E
Move of Joint A,C,E
Length of Joint B,D,F
Move of Joint B,D,F
Move of Point O

Shape of Triangle : AC:CE:EA = 1:1:1
Length of Joint A,C,E : AC * 1 / 4
Z Move of Joint A,C,E : AC * 1 / 8
Length of Joint B,D,F : AC * 1 / 4
XY Move of Joint B,D,F : AC * -1 / 8
Z Move of Point O : 0
XY Move of Point O : AC:AO:CO = 1:0.7:0.4

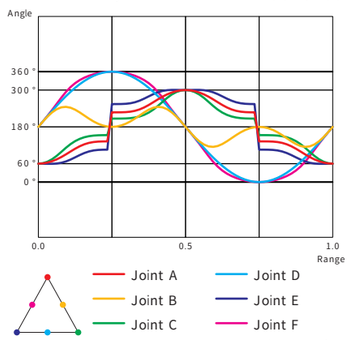
Metamorphosis Parameters



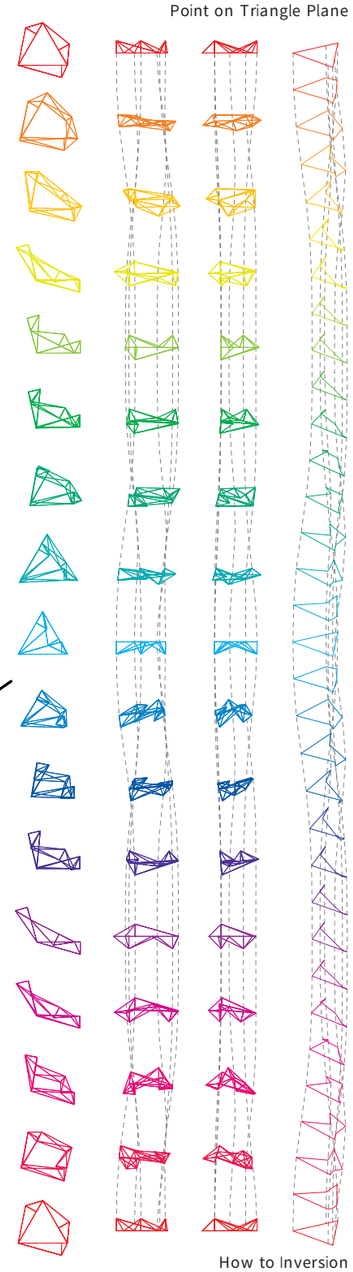
Development View



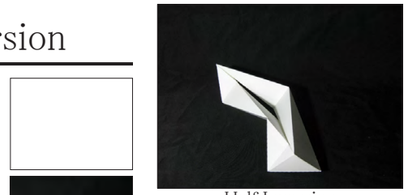
Joint Loci



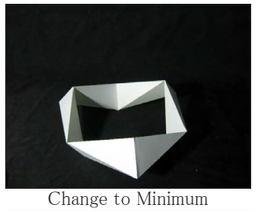
Joint Angles

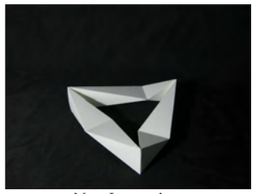


How to Inversion

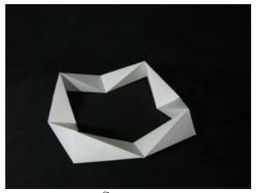

Half Inversion


Over Angle


Change to Minimum


Not Inversion


7-bar


Square


9-bar


Polygon

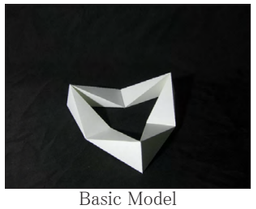
４－３－２．変形ドアノブ

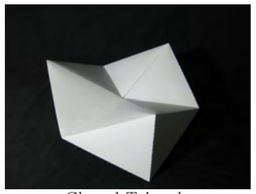
ドアノブが角度を変える事例。

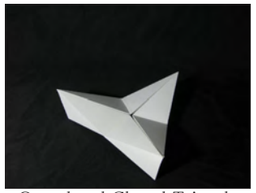
４－３－２－E．発展モデル：三角形から小さくなる

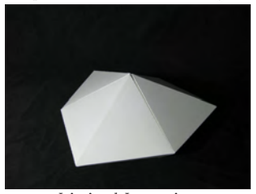
４－３－２－F．発展モデル：三角形から最小形

４－３－２－G．発展モデル：360度を超えた回転


Basic Model

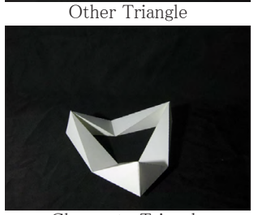

Closed Triangle

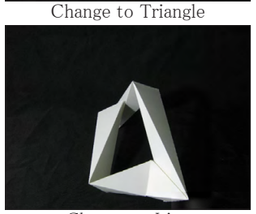

Opend and Closed Triangle

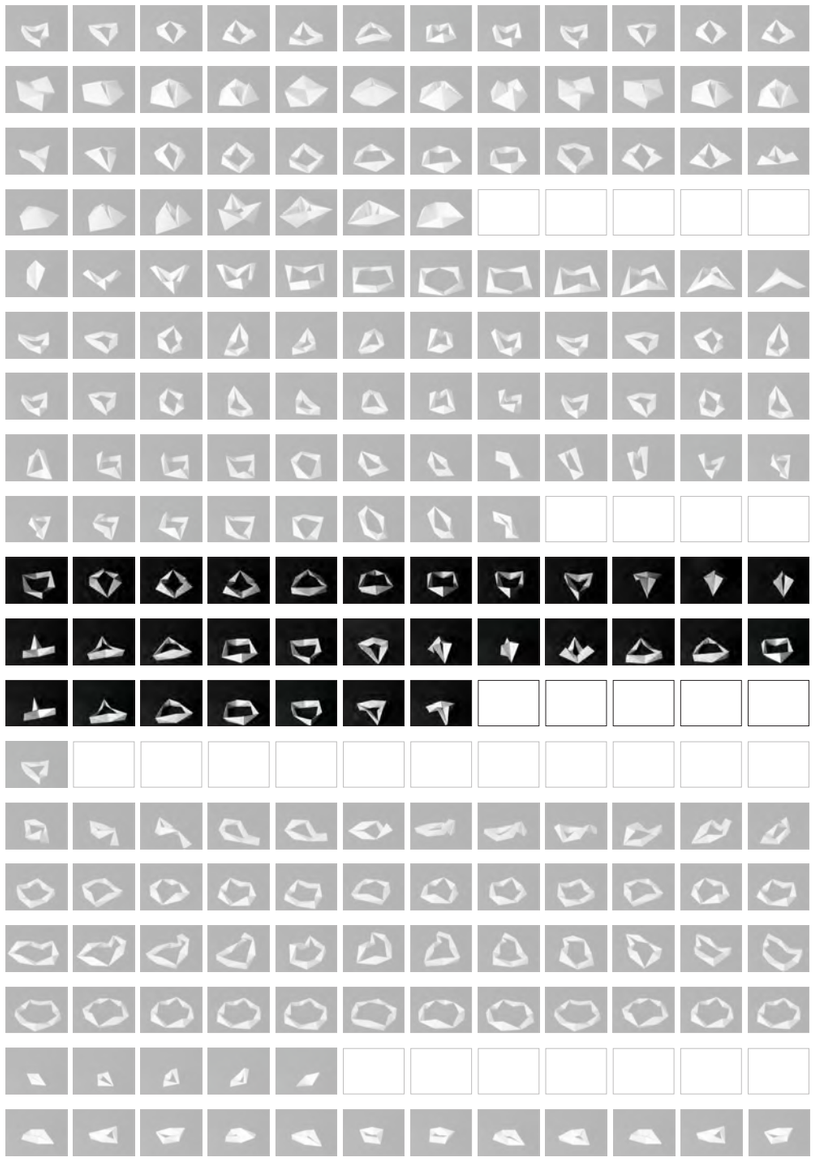

Limited Inversion


Stopped Inversion


Other Triangle


Change to Triangle


Change to Line



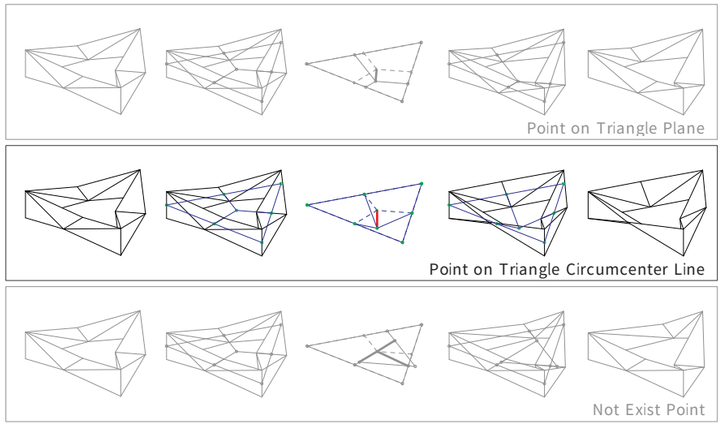
4−3−2.　Point on Triangle Circumcenter Line

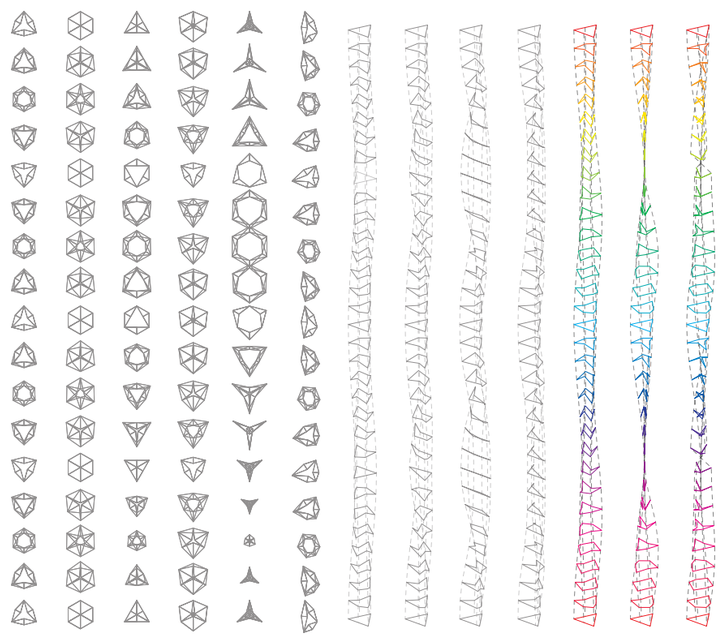
基準点が三角形の外心線上に存在する事例。この事例では、反転運動中に三角形は2つしか現れない。

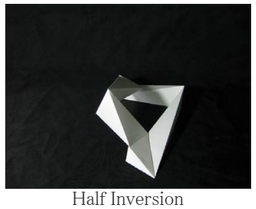
4−3−2−E.　Application Model: Change to Small

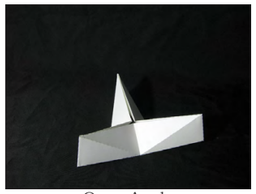
4−3−2−F.　Application Model: Change to Minimun

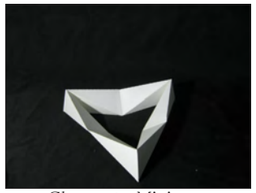
4−3−2−G.　Application Model: Over Angle

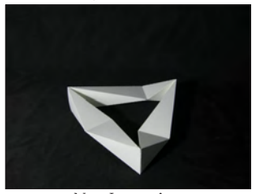


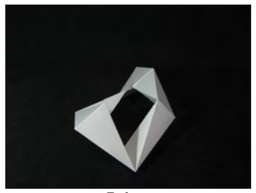



Half Inversion

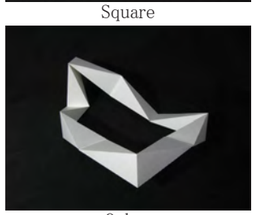

Over Angle

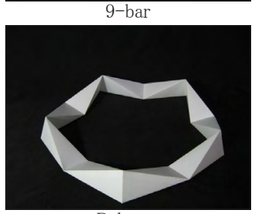

Change to Minimum


Not Inversion

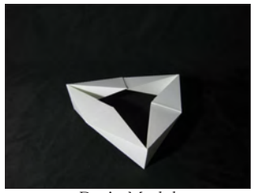

7−bar

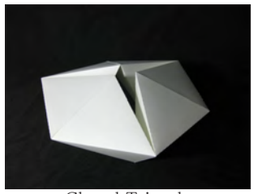

Square

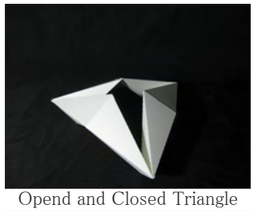

9−bar

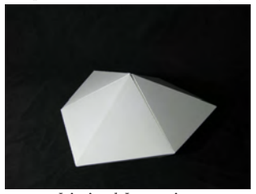

Polygon

４－３－２－Ｅ．発展モデル：三角形から小さくなる

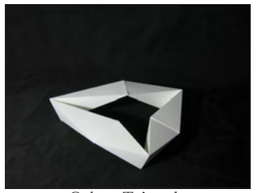

Basic Model

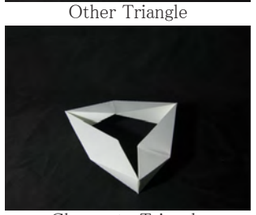

Closed Triangle

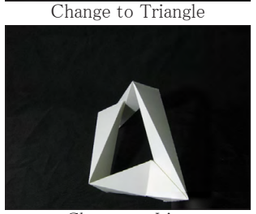

Opend and Closed Triangle

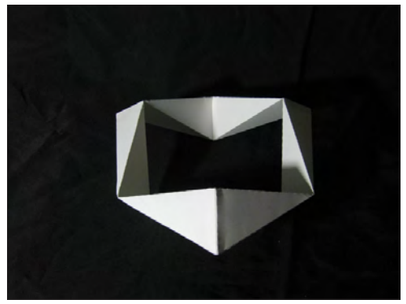

Limited Inversion

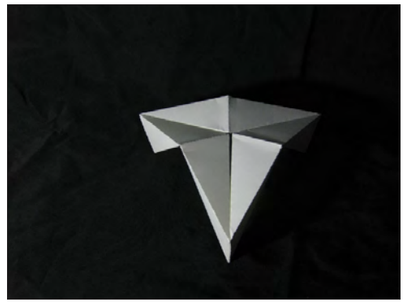

Stopped Inversion

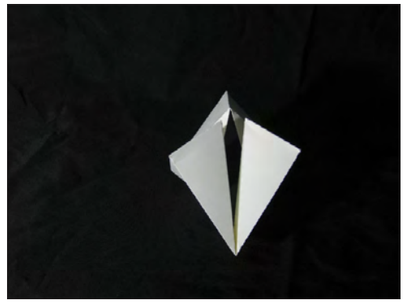

Other Triangle

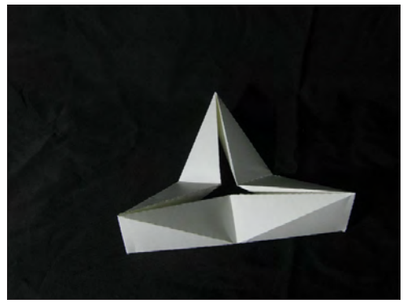

Change to Triangle

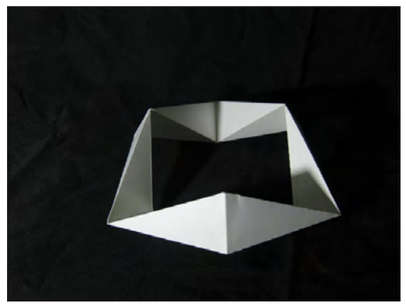

Change to Line



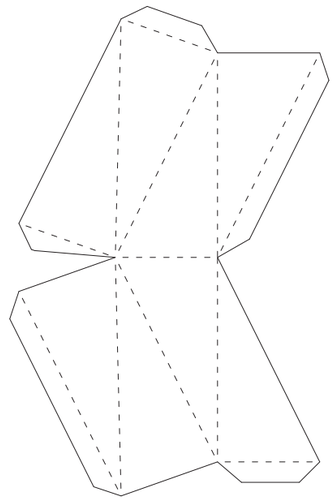


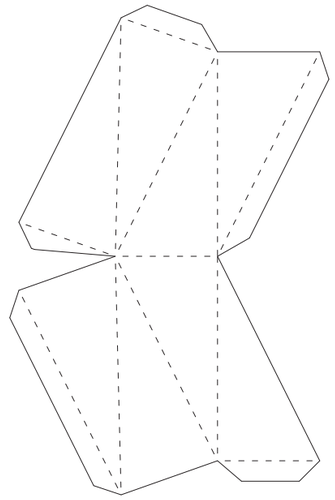


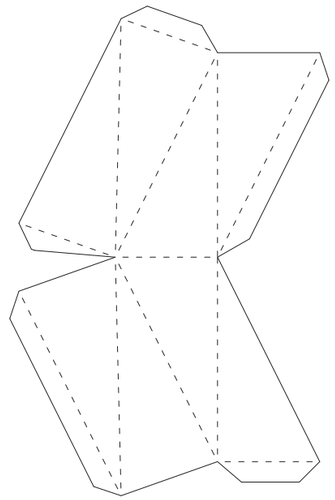




回転写真

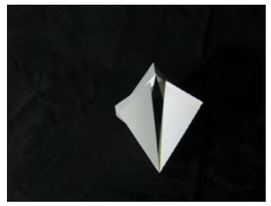




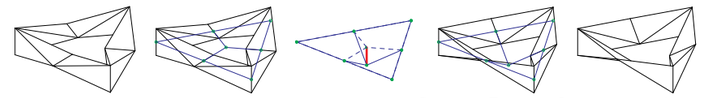


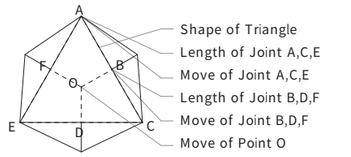
展開図

4－3－2－E．Application Model: Change to Small



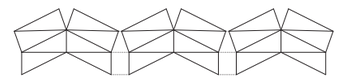
Point on Triangle Circumcenter Line の事例の中で、2点の間に存在する、反転運動の際の各ピンジョイントの変化量が数式的に見て0～360度より狭い値を推移し、反転運動をし続けられる事例。

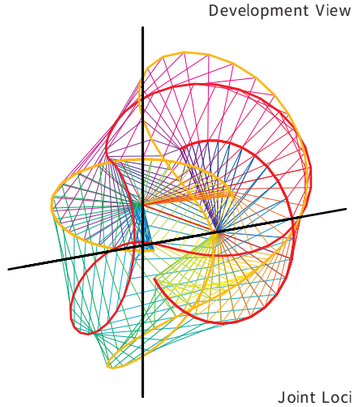

Point on Triangle Circumcenter Line

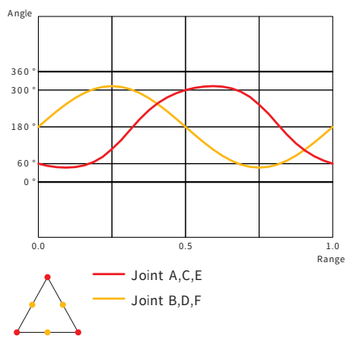


Shape of Triangle : AC:CE:EA = 1:1:1
Length of Joint A,C,E : AC * 1 / 4
Z Move of Joint A,C,E : AC * 1 / 8
Length of Joint B,D,F : AC * 1 / 4
XY Move of Joint B,D,F: AC * -1 / 8
Z Move of Point O : AC
XY Move of Point O : AC:AO:CO = √3:1:1

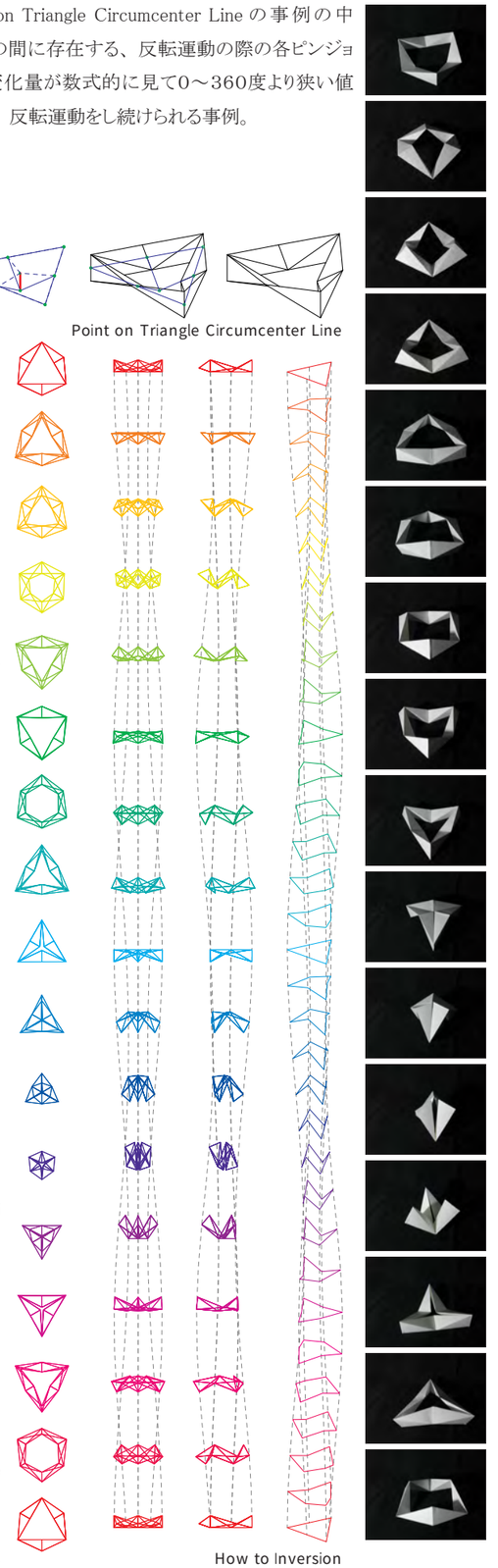
Metamorphosis Parameters


Development View

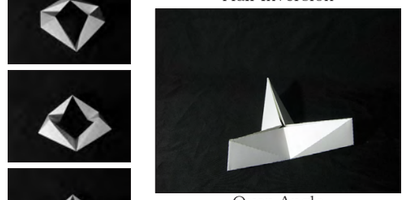

Joint Loci

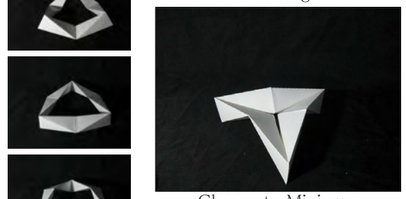


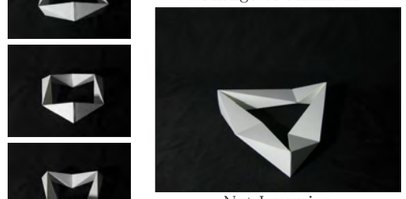
Joint Angles

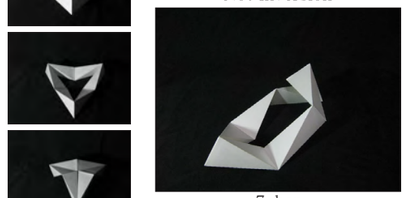

How to Inversion

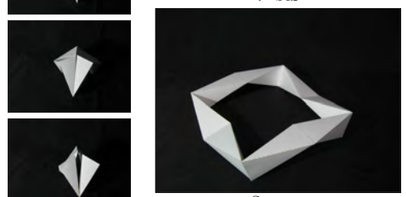

Half Inversion

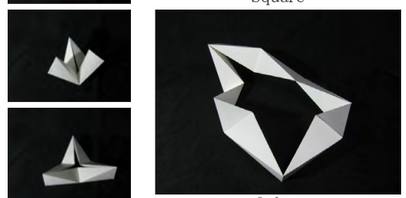

Over Angle

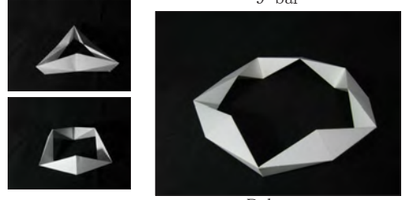

Change to Minimum


Not Inversion

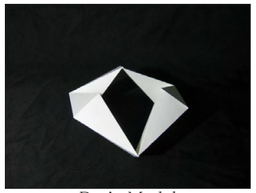

7-bar

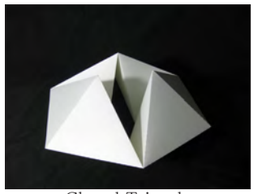

Square

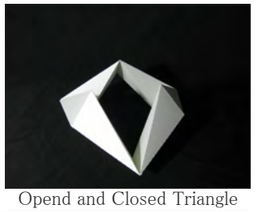

9-bar

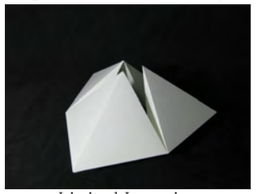

Polygon

４－３－２－Ｆ．発展モデル：三角形から最小形

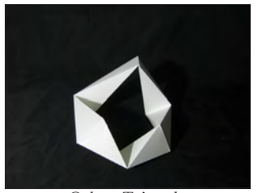

Basic Model

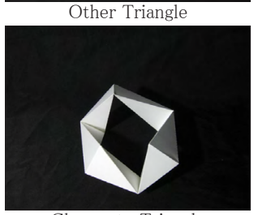

Closed Triangle

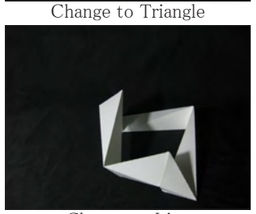

Opend and Closed Triangle

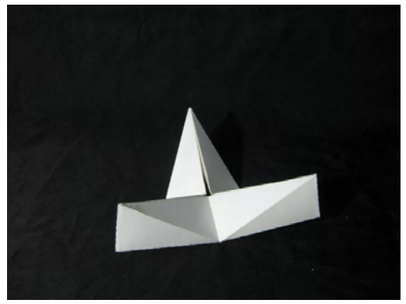

Limited Inversion

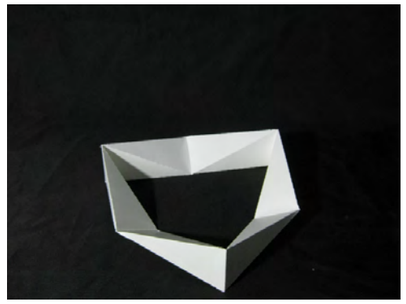

Stopped Inversion

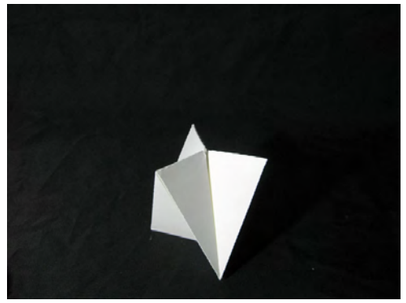

Other Triangle

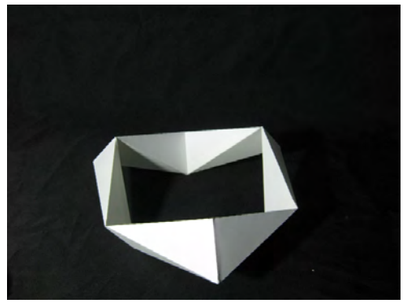

Change to Triangle

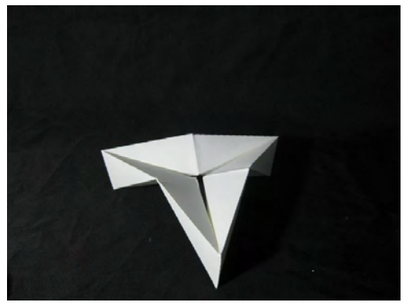

Change to Line



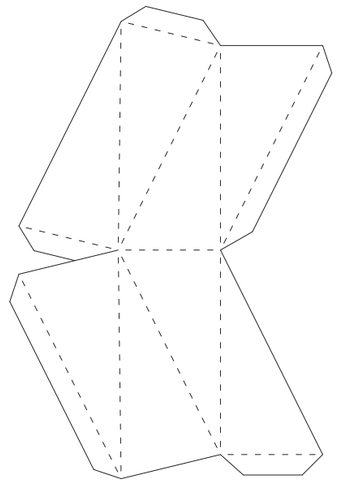


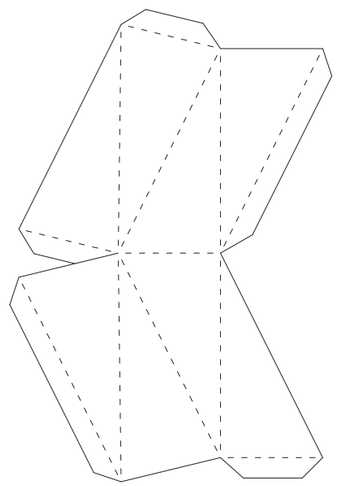


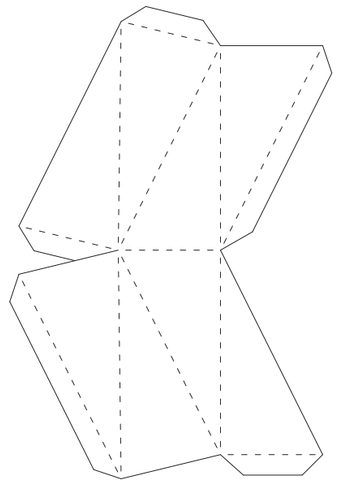




回転写真

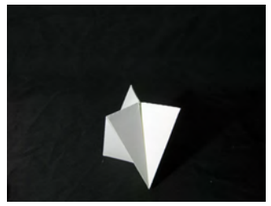




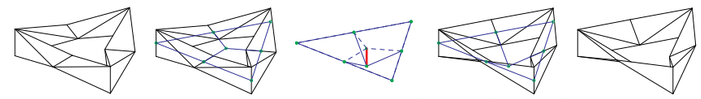


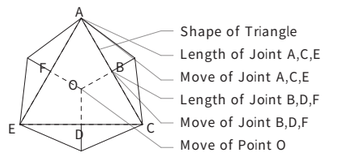
展開図

4－3－2－F．Application Model: Change to Minimun



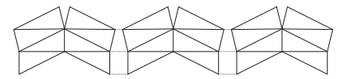
Point on Triangle Circumcenter Line の事例の中で、2点だけ存在する、反転運動の際の各ピンジョイントの変化量が数式的に見て0～360度内を推移し、反転運動をし続けられる事例。0度もしくは360度のときに、最も縮小した形を取れる事例。

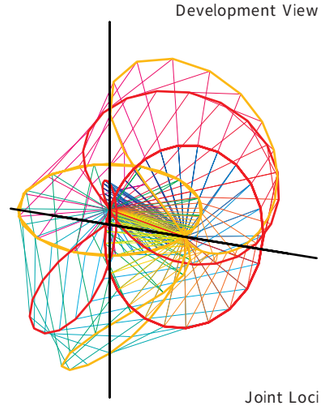

Point on Triangle Circumcenter Line

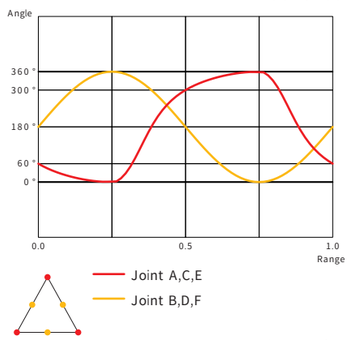


Shape of Triangle : AC:CE:EA = 1:1:1
Length of Joint A,C,E : AC * 1 / 4
Z Move of Joint A,C,E : AC * 1 / 8
Length of Joint B,D,F : AC * 1 / 4
XY Move of Joint B,D,F: AC * -1 / 8
Z Move of Point O : AC * 5 / 3
XY Move of Point O : AC:AO:CO = √3:1:1

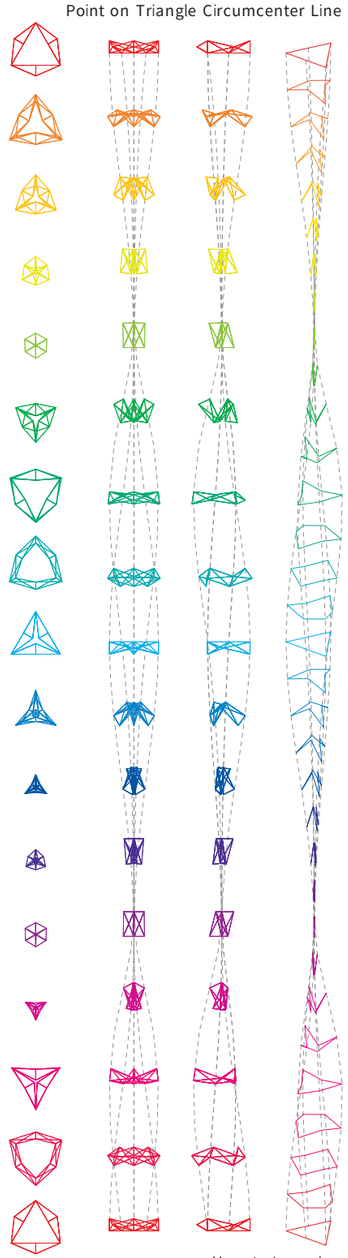
Metamorphosis Parameters

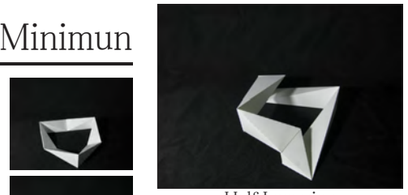

Development View

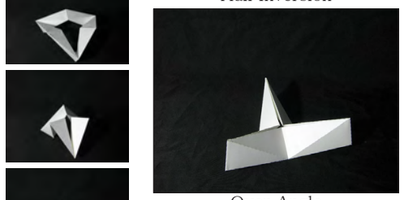

Joint Loci

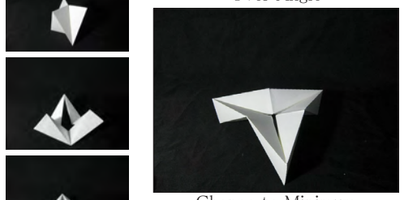


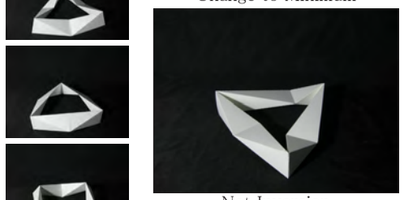
Joint Angles

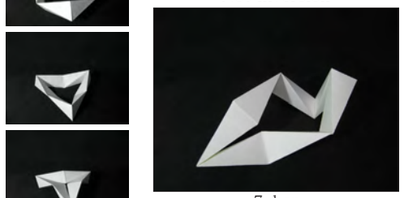

How to Inversion

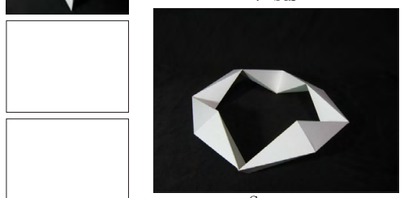

Half Inversion

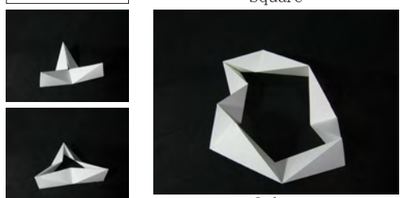

Over Angle

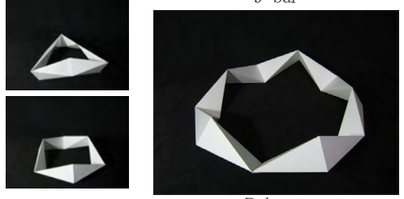

Change to Minimum


Not Inversion

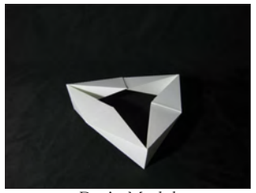

7-bar

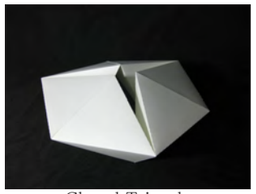

Square

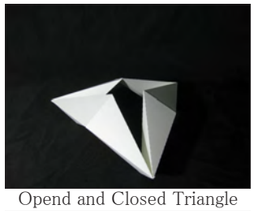

9-bar

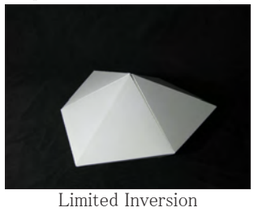

Polygon

４－３－２－Ｇ．発展モデル：３６０度を超えた回転

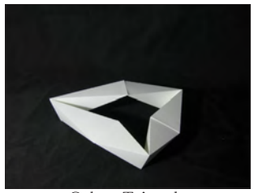

Basic Model

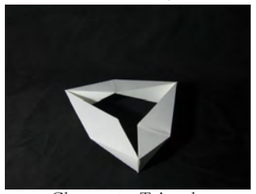

Closed Triangle

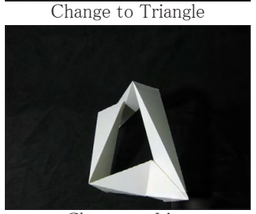

Opend and Closed Triangle


Limited Inversion

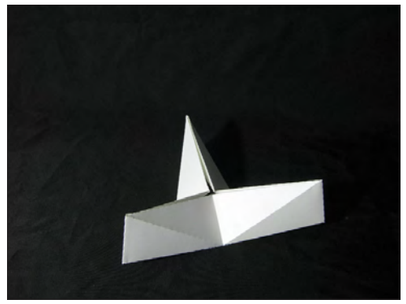

Stopped Inversion

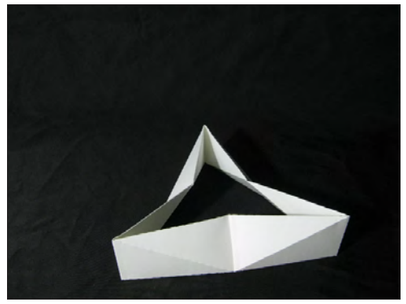

Other Triangle

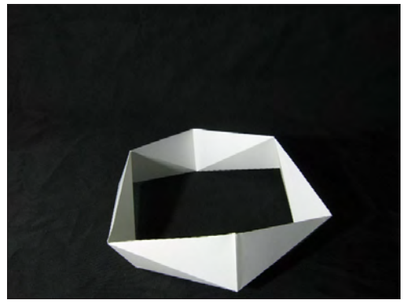

Change to Triangle

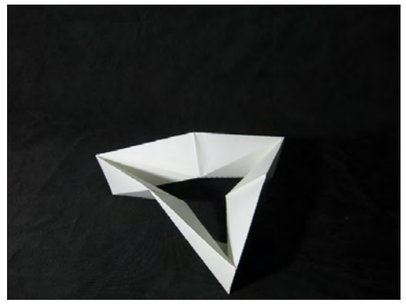

Change to Line

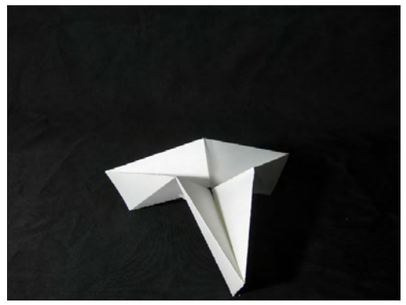
回転写真

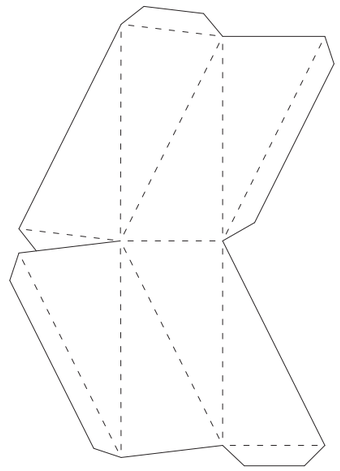


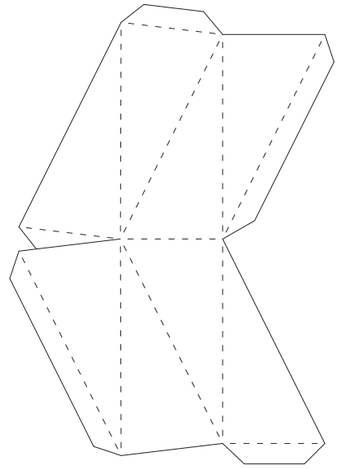


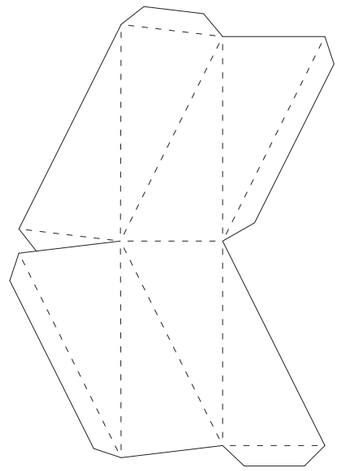








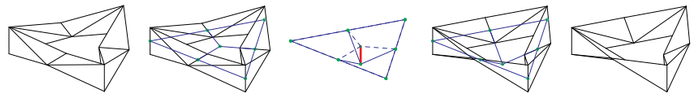


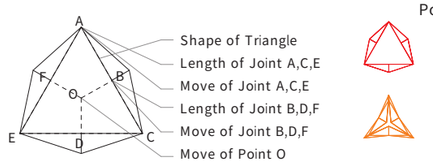


展開図

４－３－２－Ｇ．Application Model: Over Angle

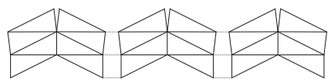
Point on Triangle Circumcenter Line の事例の中で、反転運動をし続けられない事例。反転運動の際、各ピンジョイントの角度の変化量が数式的に見て360度を超えるため反転運動量が制限される事例。

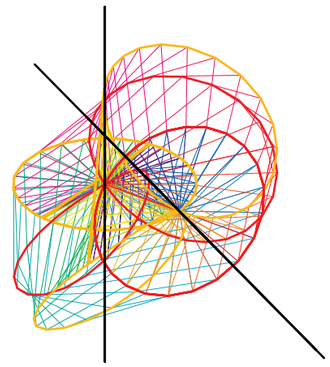

Point on Triangle Circumcenter Line

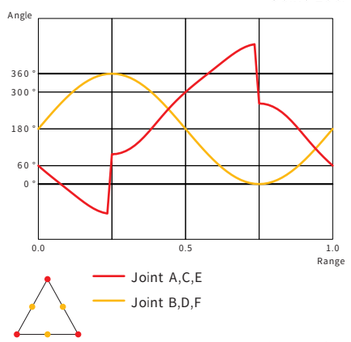


Shape of Triangle : AC:CE:EA = 1:1:1
Length of Joint A,C,E : AC * 1 / 4
Z Move of Joint A,C,E : AC * 1 / 8
Length of Joint B,D,F : AC * 1 / 4
XY Move of Joint B,D,F : AC * -1 / 8
Z Move of Point O : AC * 5 / 6
XY Move of Point O : AC:AO:CO = √3:1:1

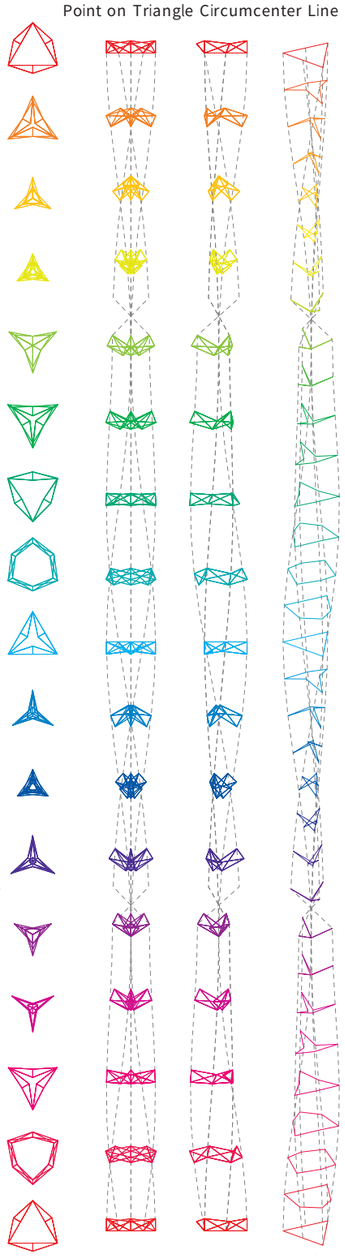
Metamorphosis Parameters

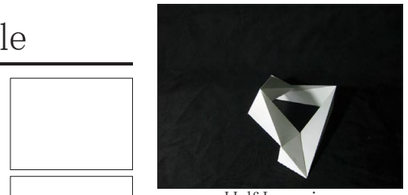

Development View

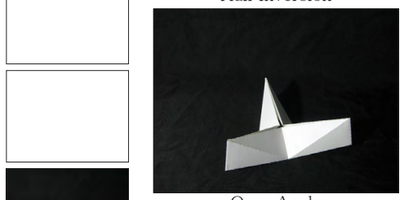

Joint Loci

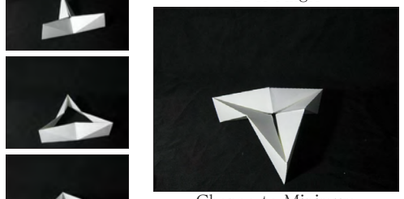


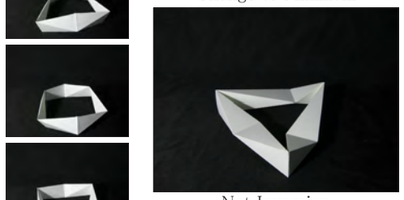
Joint Angles

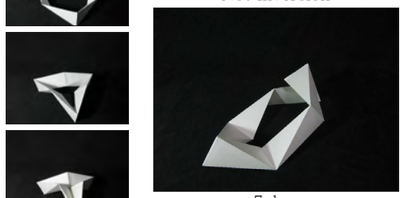

How to Inversion

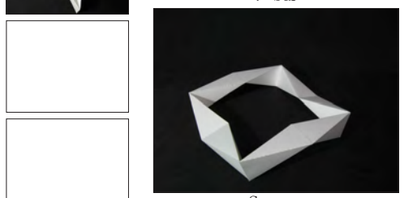

Half Inversion

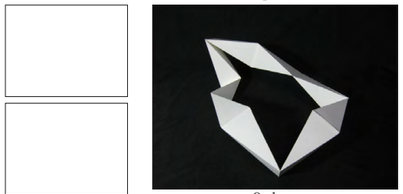

Over Angle

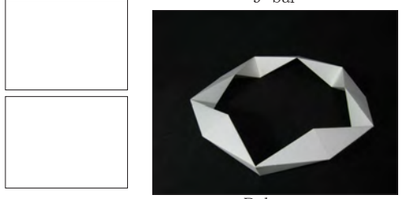

Change to Minimum


Not Inversion


7-bar


Square

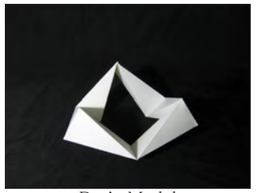

9-bar

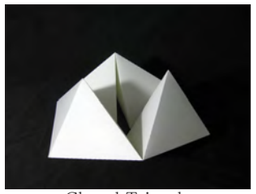

Polygon

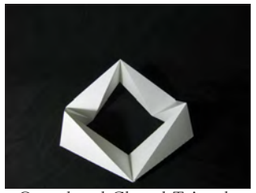
４－３－３．点が存在しない

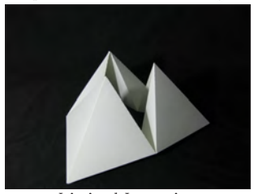
ドアノブの回転ラインが１点で交わらない事例。

４－３－３－ G.　特殊モデル：回転しない


Basic Model

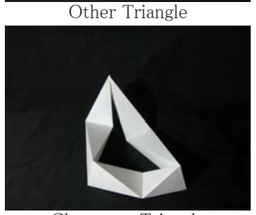

Closed Triangle

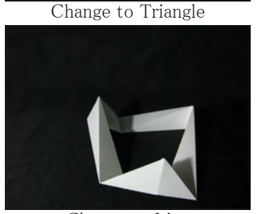

Opend and Closed Triangle

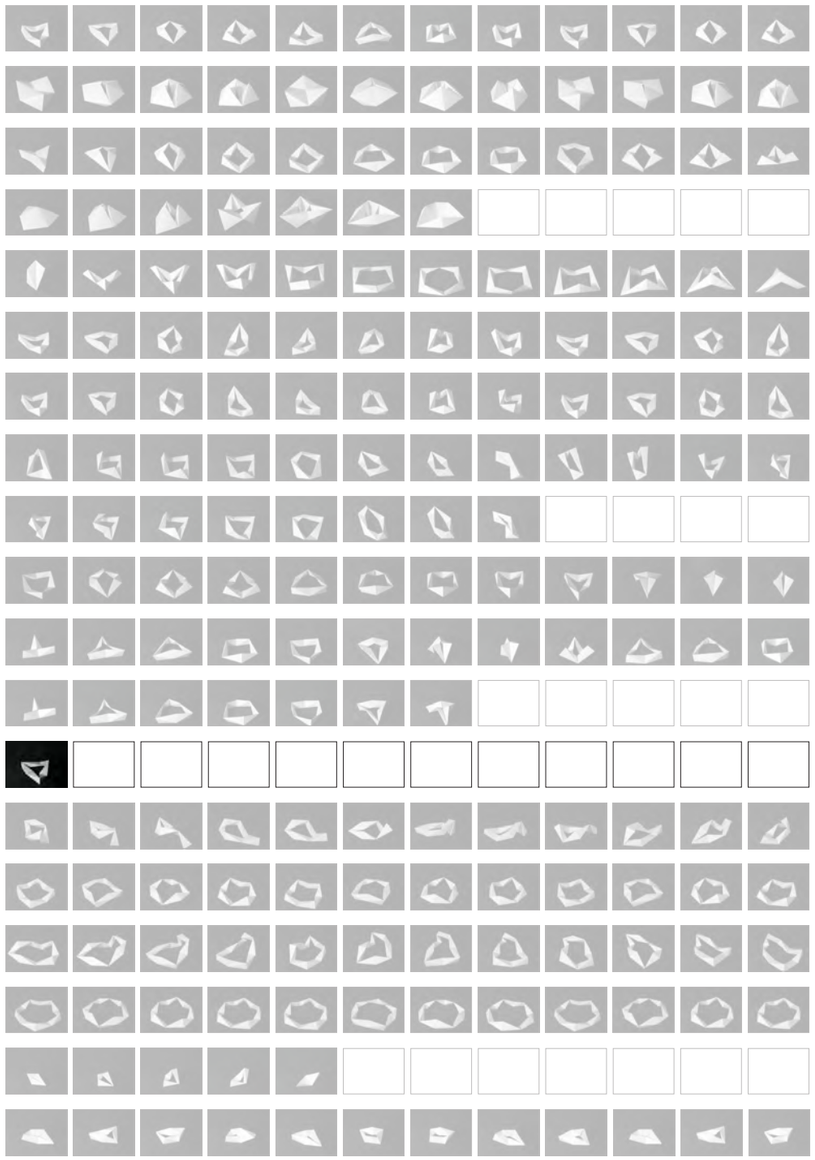

Limited Inversion


Stopped Inversion


Other Triangle


Change to Triangle

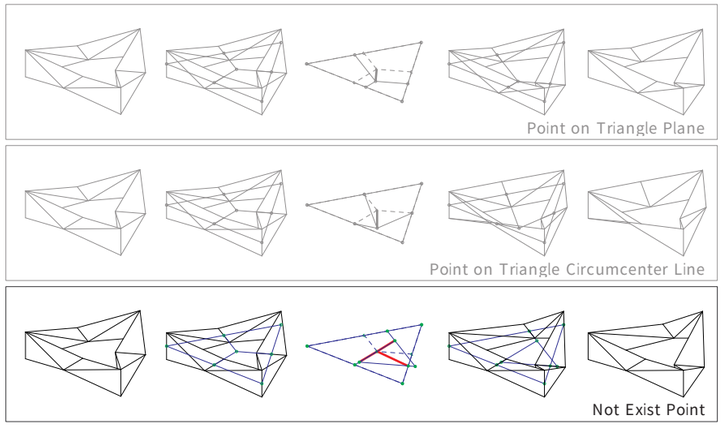

Change to Line

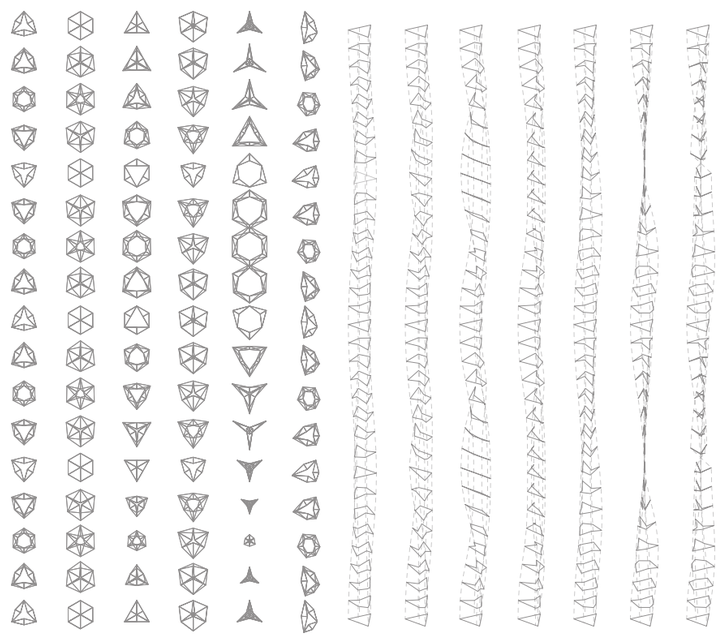


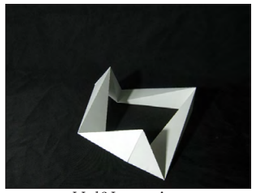
4－3－3．Not Exist Point

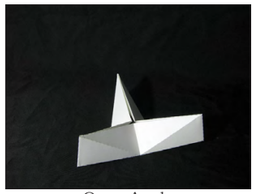
各ピンジョイントのラインが1点で交わらず、基準点が存在しない事例。

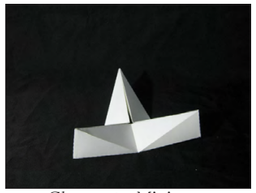
4－3－3－ G.　Another Model: Not Inversion

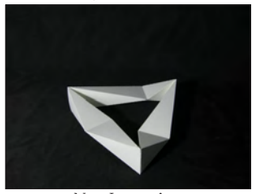




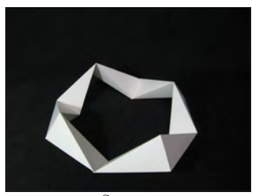

Half Inversion

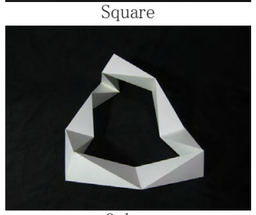

Over Angle

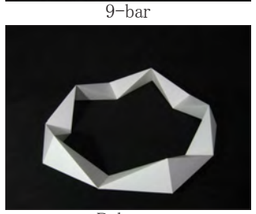

Change to Minimum

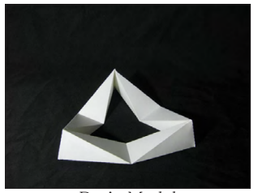

Not Inversion

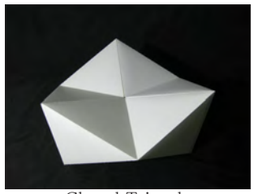

7-bar

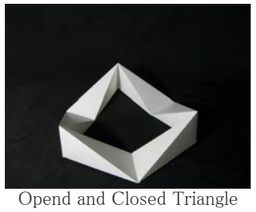

Square

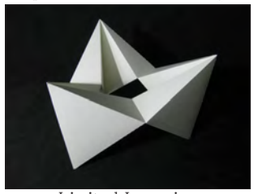

9-bar

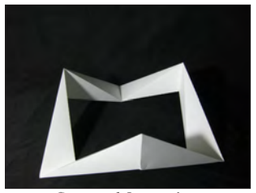

Polygon


Basic Model

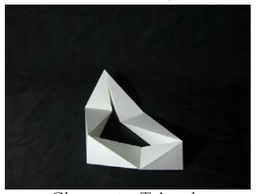

Closed Triangle

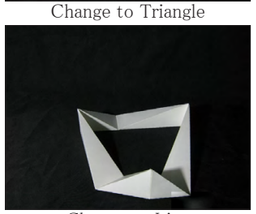

Opend and Closed Triangle


Limited Inversion

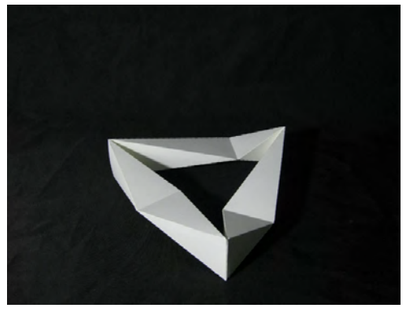

Stopped Inversion

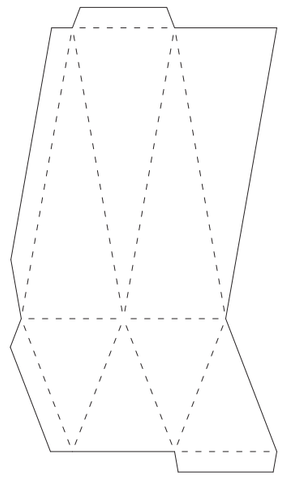

Other Triangle

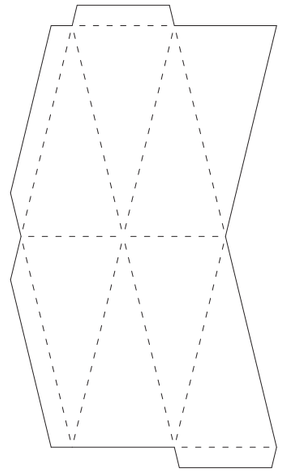

Change to Triangle

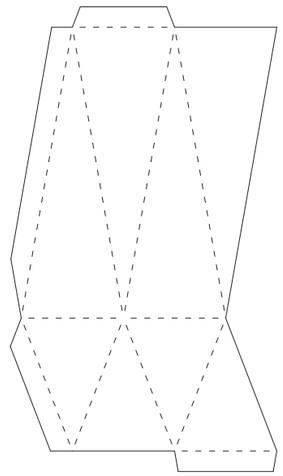

Change to Line

４－３－３－Ｇ．特殊モデル：回転しない





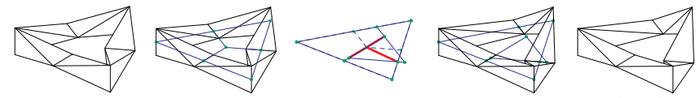


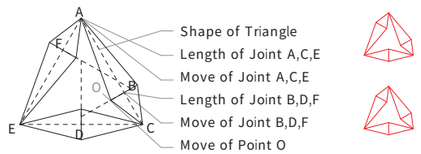


回転写真　　展開図

4－3－3－ G．Another Model: Not Inversion

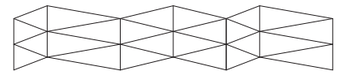
反転運動が起きない。



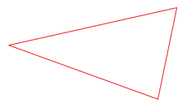


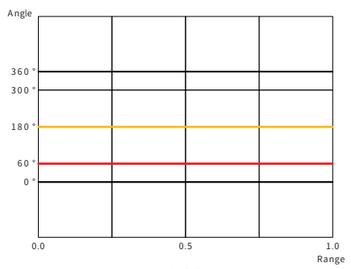
Shape of Triangle : AC:CE:EA = 1:1:1
Length of Joint A,C,E : AC * 1 / 4
Z Move of Joint A,C,E : 0
Length of Joint B,D,F : AC * 1 / 4
XY Move of Joint B,D,F : 0
Z Move of Point O : No Data
XY Move of Point O : No Data

Metamorphosis Parameters



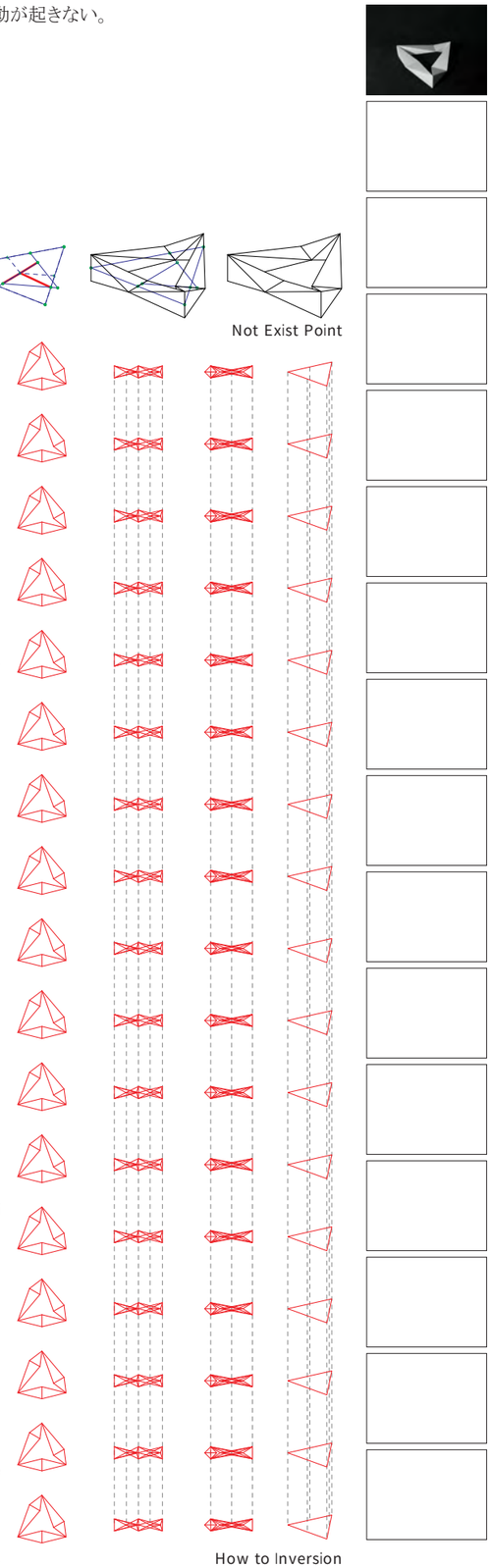
Development View



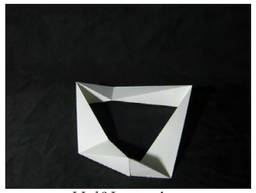


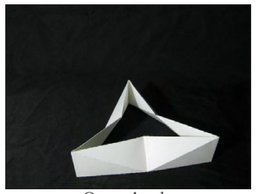
Joint Loci

Joint Angles

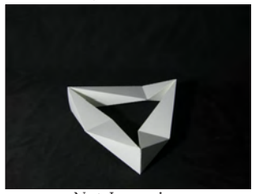


How to Inversion

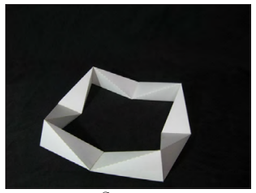

Half Inversion


Over Angle

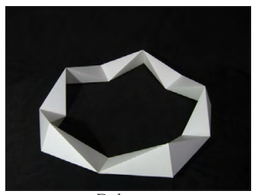

Change to Minimum

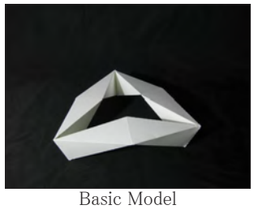

Not Inversion

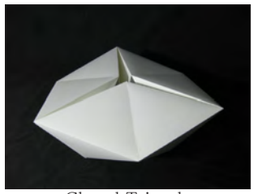

7-bar

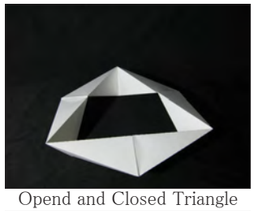

Square

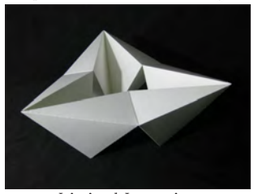

9-bar

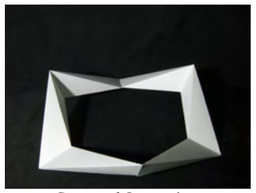

Polygon


Basic Model

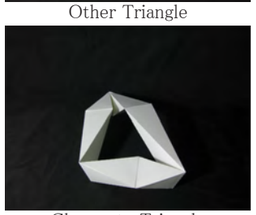

Closed Triangle

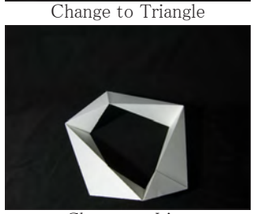

Opend and Closed Triangle


Limited Inversion


Stopped Inversion


Other Triangle


Change to Triangle


Change to Line

４－４．ドアの数

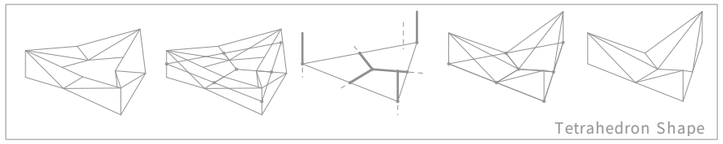
ドアの数を変える。ドアの数を変えることによって、変形の仕方が一定なものからいろいろなかたちへと変形するものになる。

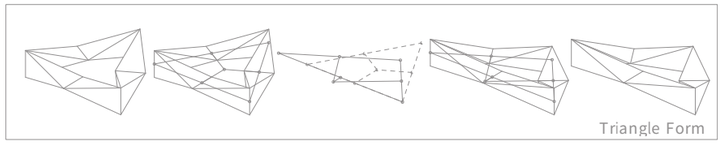
４－４－A．基本モデル：三角形（６つのボリューム）
４－４－B．発展モデル：３．５角形（７つのボリューム）
４－４－C．発展モデル：四角形（８つのボリューム）
４－４－D．発展モデル：４．５角形（９つのボリューム）
４－４－E．発展モデル：多角形（１０個以上のボリューム）
４－４－F．特殊モデル：二角形（４つのボリューム）
４－４－G．特殊モデル：２．５角形（５つのボリューム）

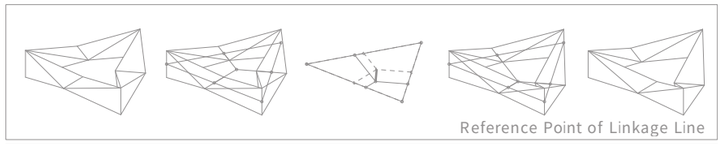
4－4．Linkage Number

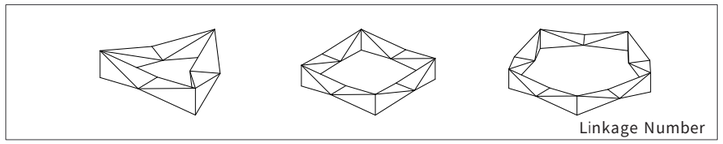
リンクの数を変える。リンクの数を変えることによって、自由度に変化が現れる。本研究ではこの項目については、より単純化されたものだけを扱っており、ジョイントの角度関係が Kaleidocycle と異なるものは "Another Model" として取り上げた。より複雑な事例は既に Marcus Engel や Toshihiro Onishi により研究されている。[30)]

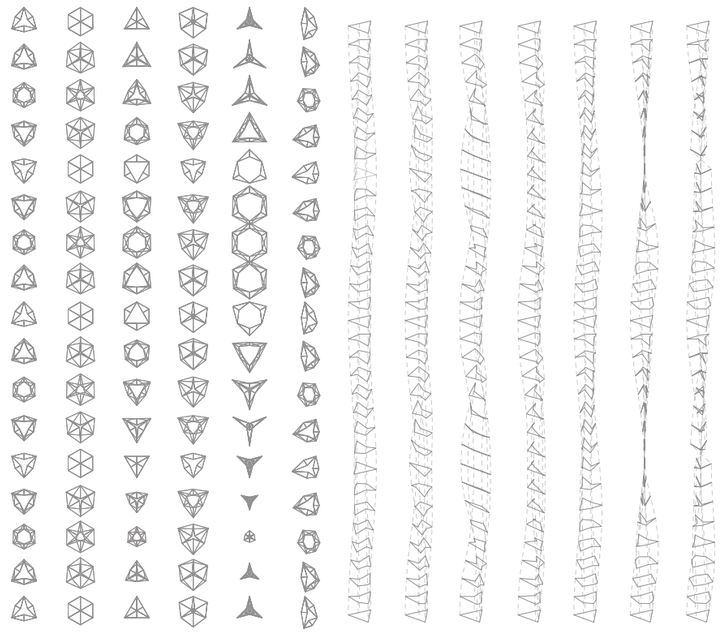
4－4－A．Basic Model: Triangle(6-bar Linkage)
4－4－B．Application Model: 7-bar(7-bar Linkage)
4－4－C．Application Model: Square(8-bar Linkage)
4－4－D．Application Model: 9-bar(9-bar Linkage)
4－4－E．Application Model: Polygon(over 10-bar)
4－4－F．Another Model: Bennett(4-bar Linkage)
4－4－G．Another Model: 5-bar(5-bar Linkage)

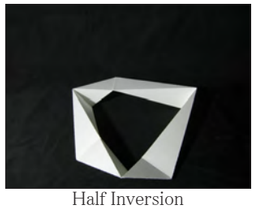






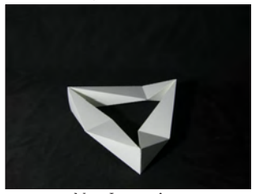





Half Inversion

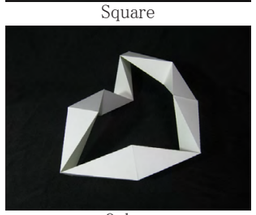

Over Angle

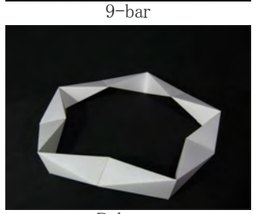

Change to Minimum


Not Inversion

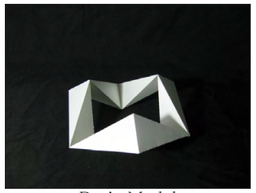

7-bar

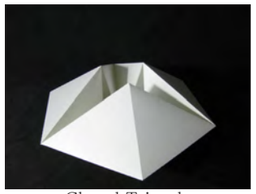

Square

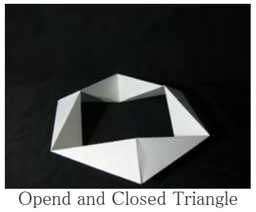

9-bar

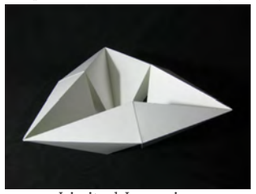

Polygon

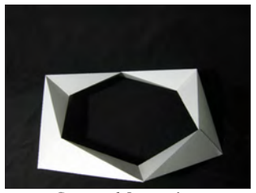
４－４－Ａ. 基本モデル： 三角形（６つのボリューム）

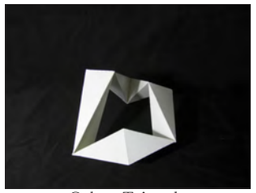

Basic Model

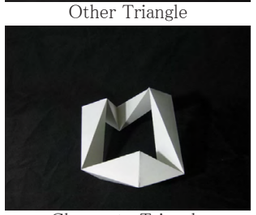

Closed Triangle

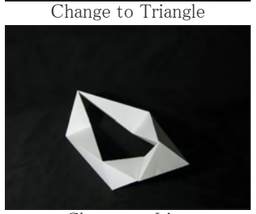

Opend and Closed Triangle

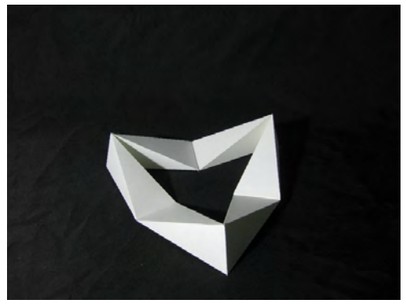

Limited Inversion

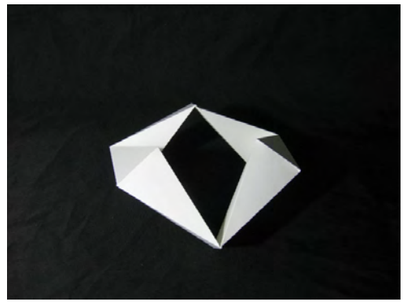

Stopped Inversion

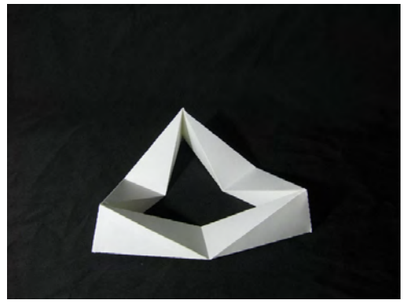

Other Triangle

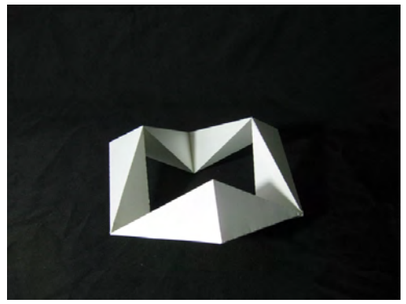

Change to Triangle

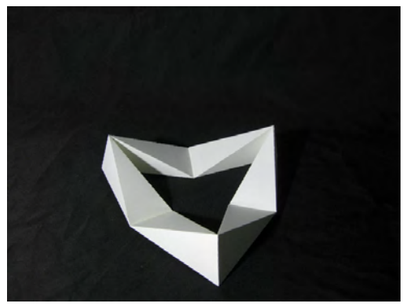

Change to Line



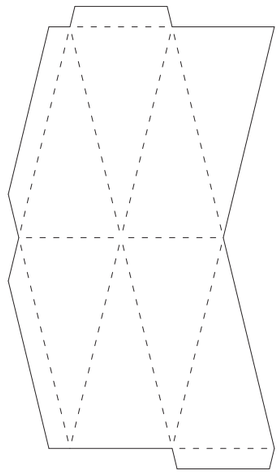


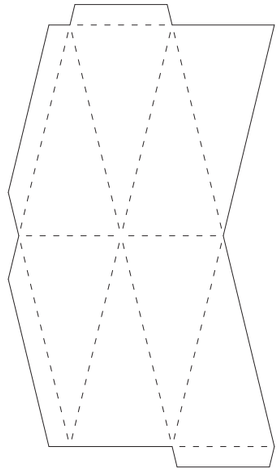


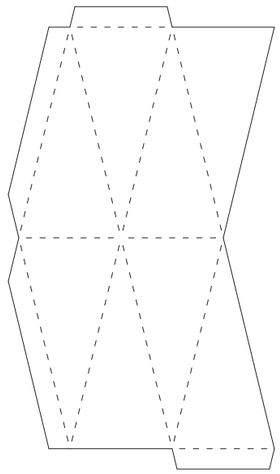




回転写真

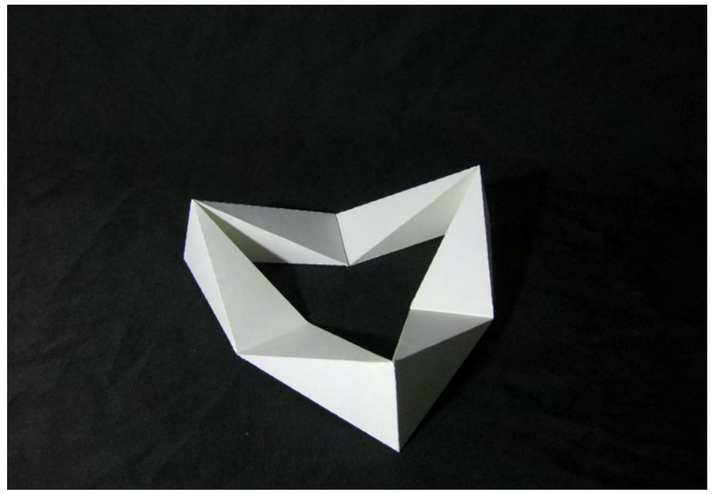




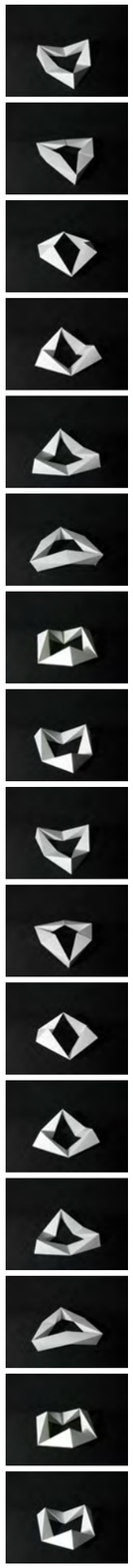


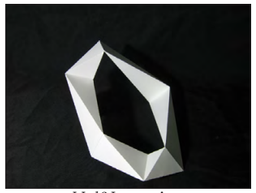
展開図

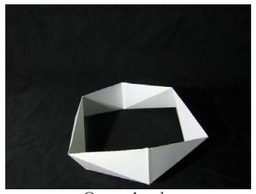
4－4－A.　Basic Model: Triangle(6-bar Linkage)



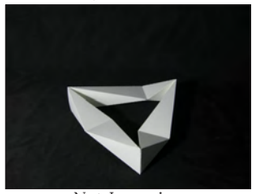
リンクの数が6つの状態。自由度1の変形をする。




Half Inversion

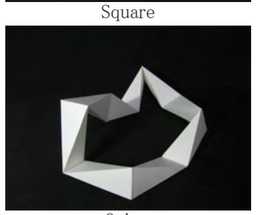

Over Angle


Change to Minimum


Not Inversion


7-bar

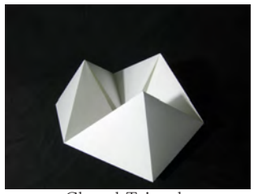

Square

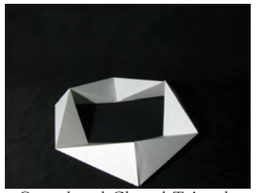

9-bar

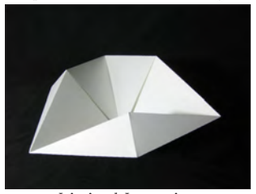

Polygon

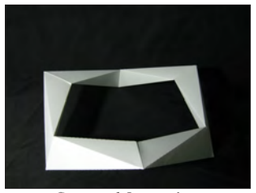
４－４－B. 発展モデル：３.５角形（７つのボリューム）


Basic Model


Closed Triangle

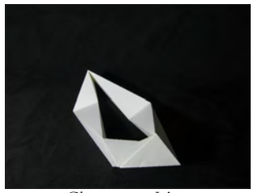

Opend and Closed Triangle

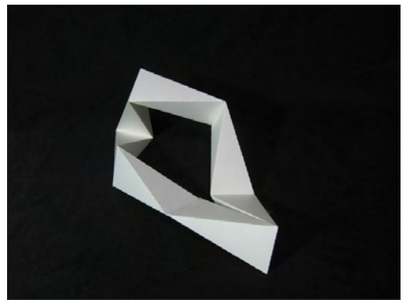

Limited Inversion

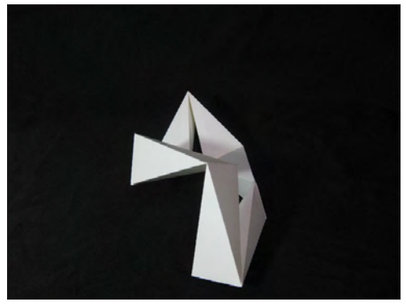

Stopped Inversion

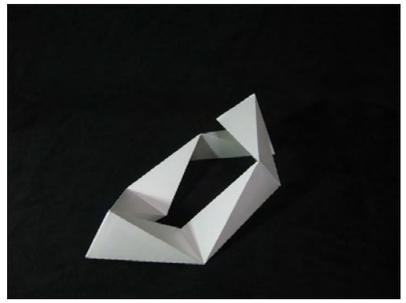

Other Triangle

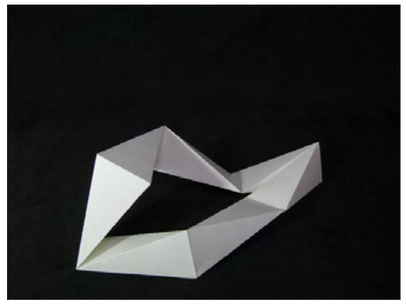

Change to Triangle

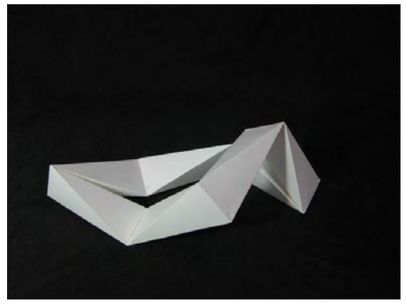

Change to Line



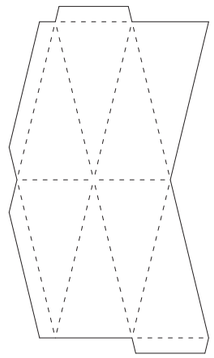


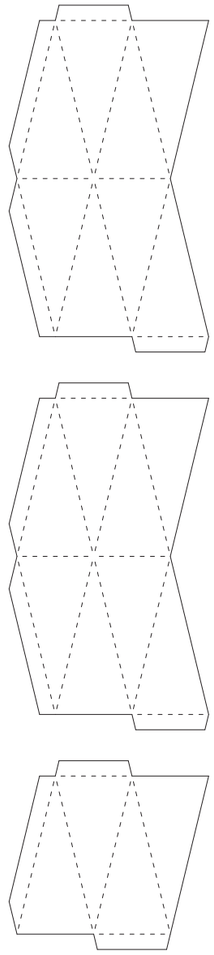






回転写真

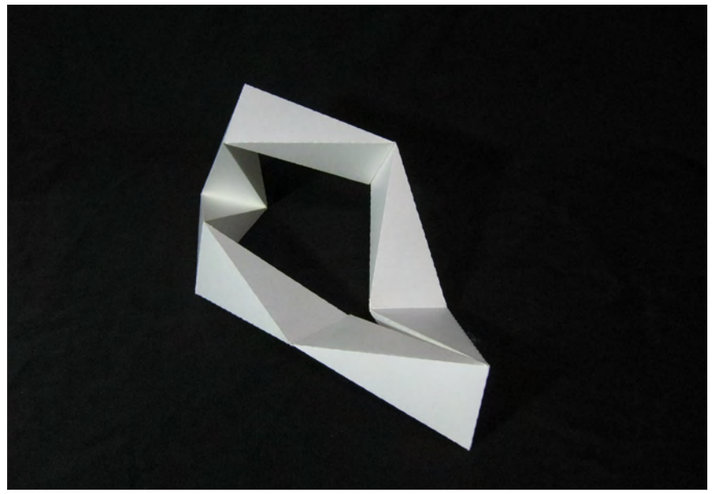




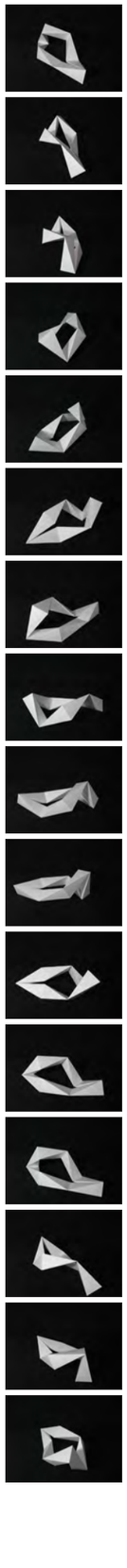
展開図

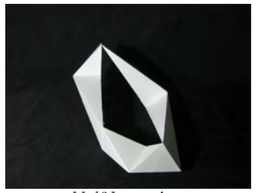
078

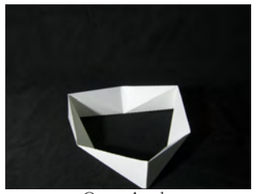
4－4－B. Application Model: 7-bar(7-bar Linkage)

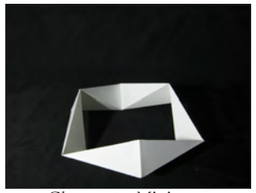


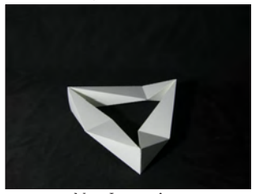
リンクの数が7つの状態。自由度1の変形をする。リンクの数が奇数のため、ジョイントの角度関係が Kalaidocycle と同じ状態では平面上の多角形にはならない。

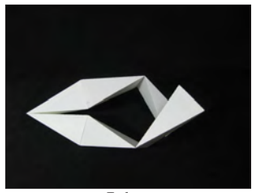



Half Inversion


Over Angle

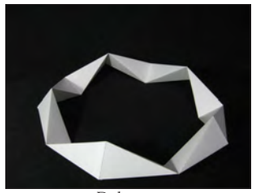

Change to Minimum


Not Inversion


7-bar

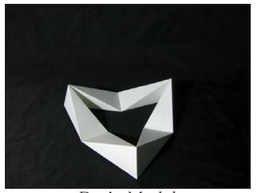

Square

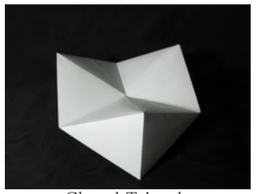

9-bar

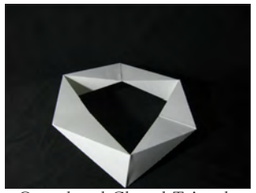

Polygon

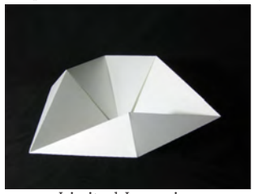
079

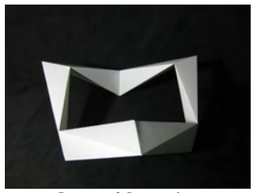
４－４－C. 発展モデル：四角形（８つのボリューム）


Basic Model


Closed Triangle

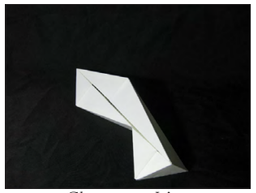

Opend and Closed Triangle


Limited Inversion

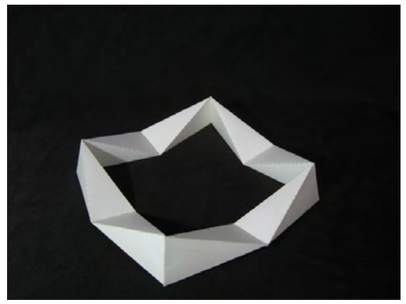

Stopped Inversion

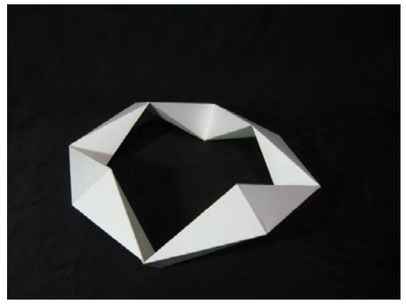

Other Triangle

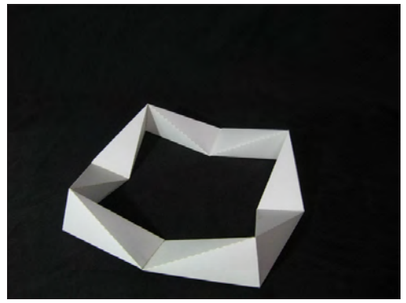

Change to Triangle

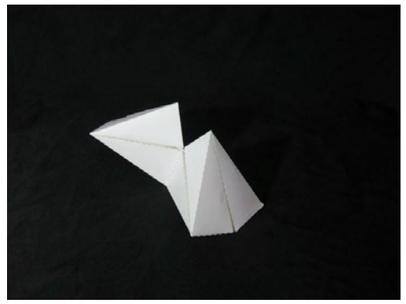

Change to Line

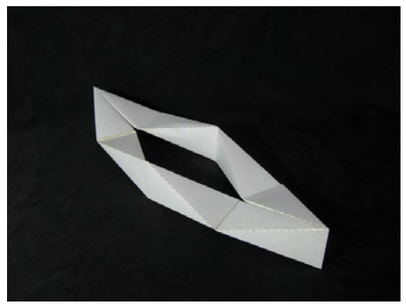
回転写真



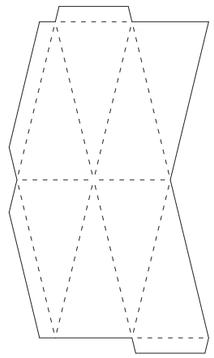


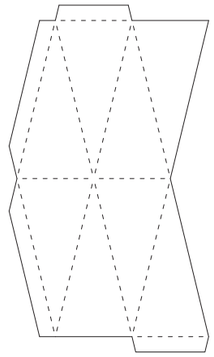


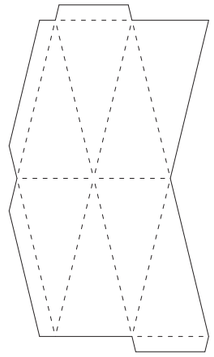


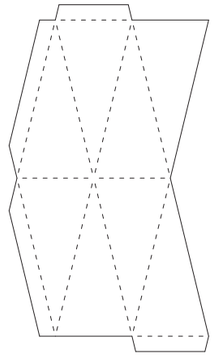


展開図

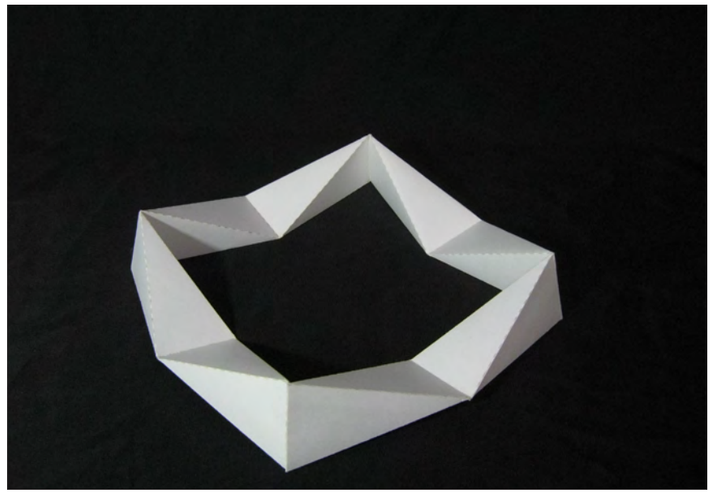




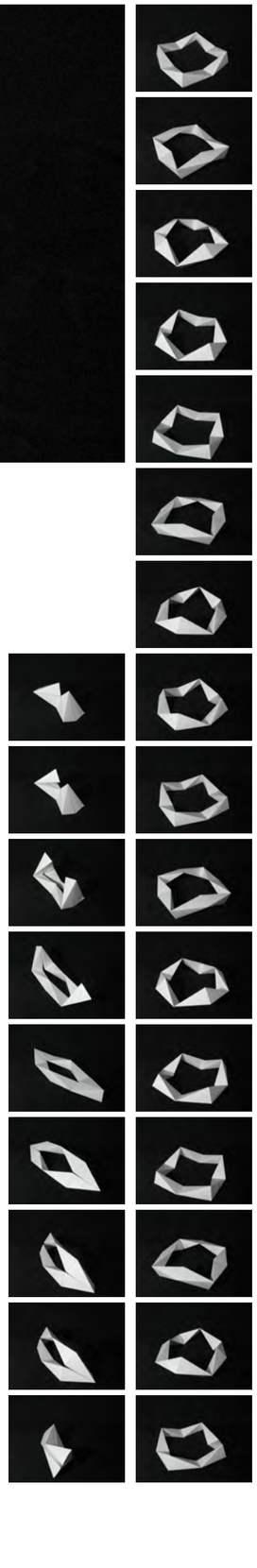


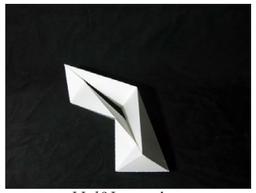


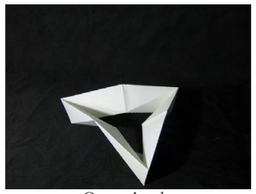
4－4－C. Application Model: Square(8-bar Linkage)

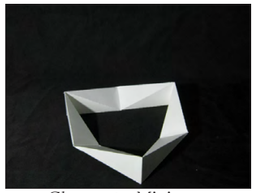


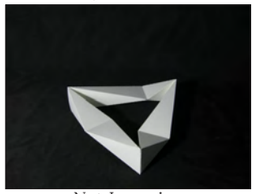
リンクの数が8つの状態。自由度1ではないが、自由度は低い。



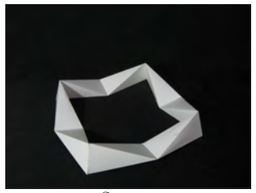

Half Inversion


Over Angle

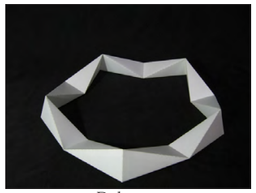

Change to Minimum


Not Inversion

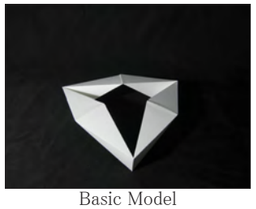

7-bar

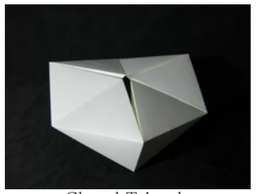

Square

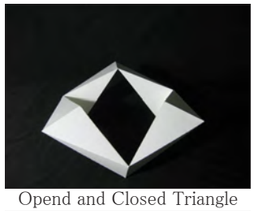

9-bar

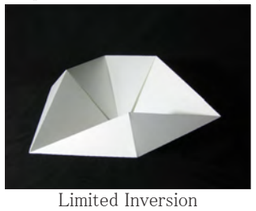

Polygon

４－４－D. 発展モデル：４. ５角形（９つのボリューム）


Basic Model

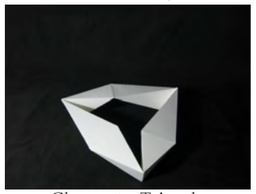

Closed Triangle

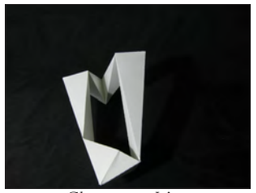

Opend and Closed Triangle


Limited Inversion

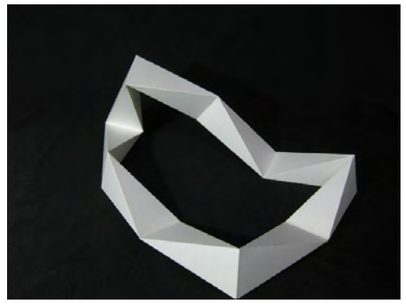

Stopped Inversion

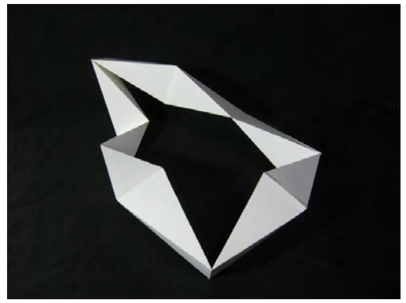

Other Triangle

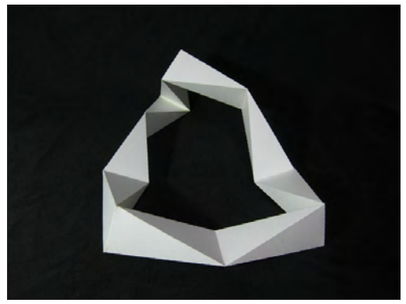

Change to Triangle

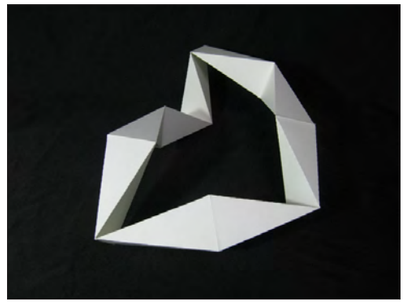

Change to Line

回転写真

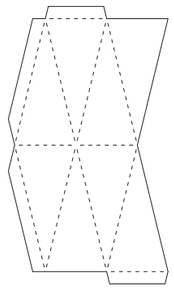


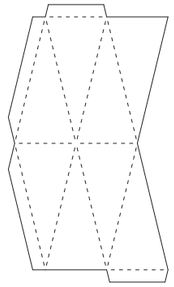


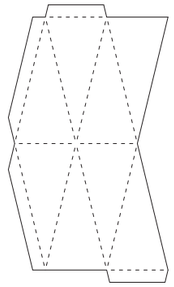


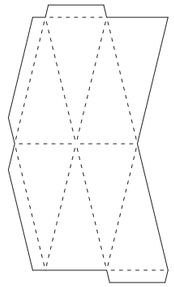


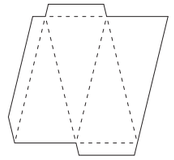
展開図



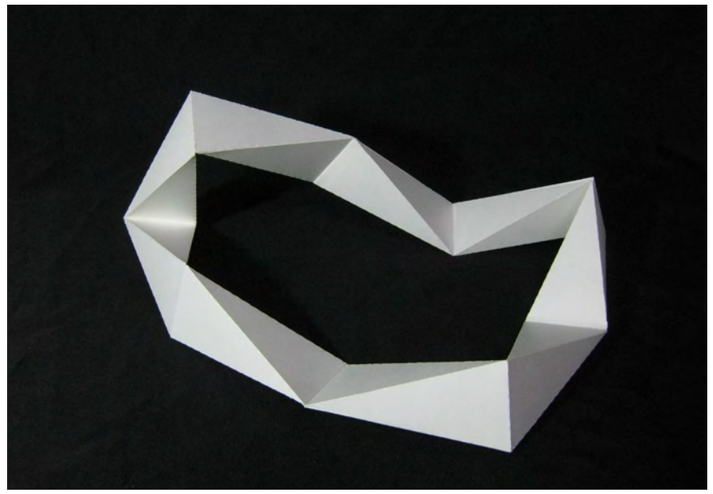




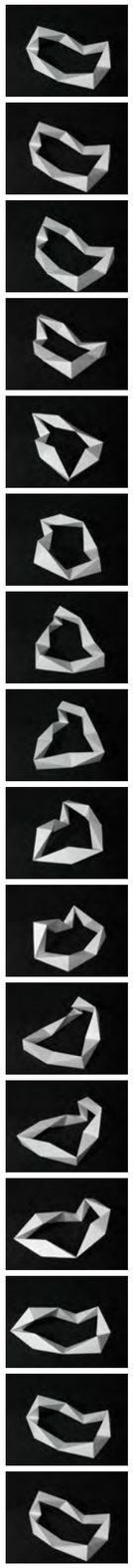


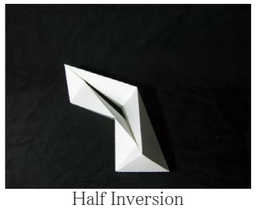


4－4－D.　Application Model: 9-bar(9-bar Linkage)



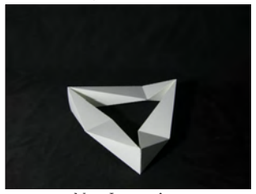
リンクの数が9つの状態。自由度1ではないが、自由度は低い。リンクの数が奇数のため、ジョイントの角度関係が Kalaidocycle と同じ状態では平面上の多角形にはならない。

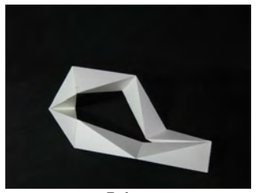


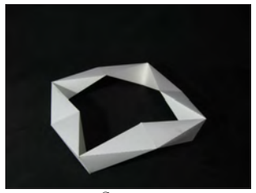

Half Inversion

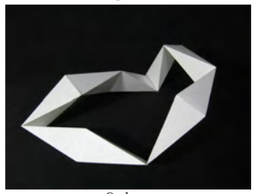

Over Angle


Change to Minimum


Not Inversion

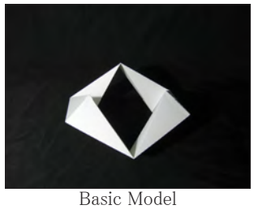

7-bar

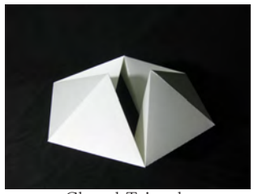

Square

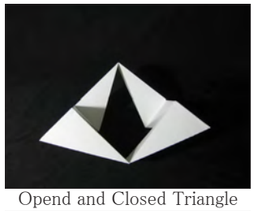

9-bar

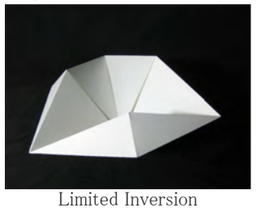

Polygon

４－４－E. 発展モデル：多角形（１０個以上のボリューム）

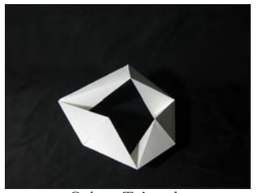

Basic Model

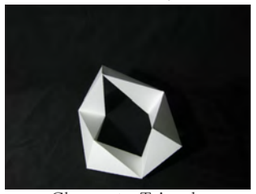

Closed Triangle

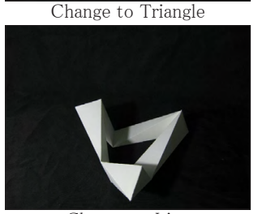

Opend and Closed Triangle

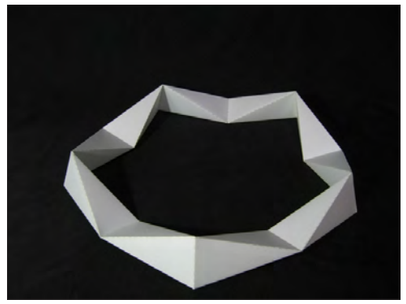

Limited Inversion

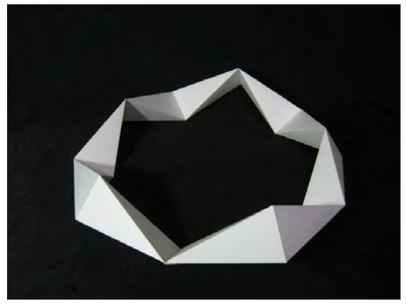

Stopped Inversion

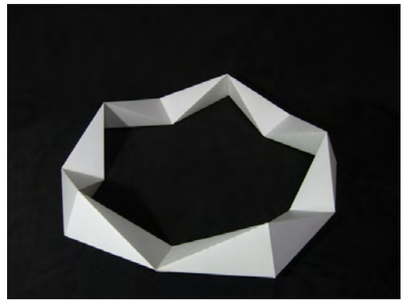

Other Triangle

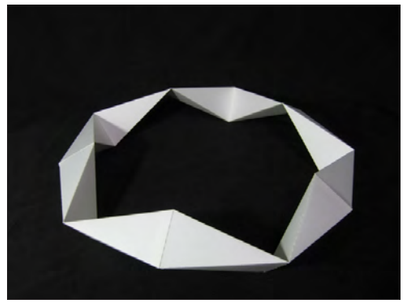

Change to Triangle

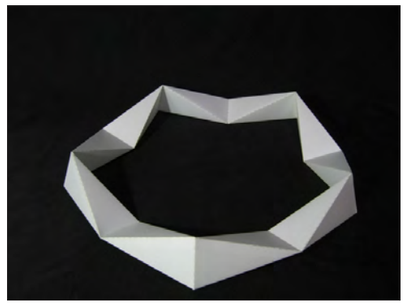

Change to Line



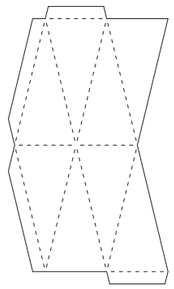


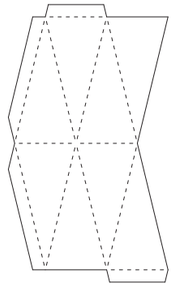


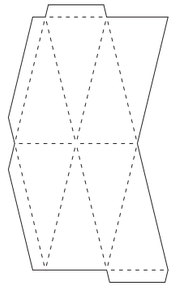


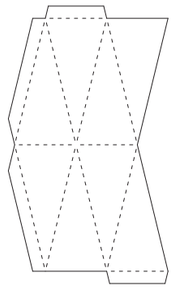


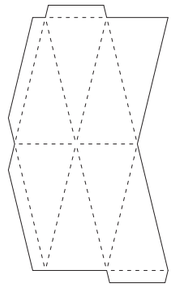
回転写真





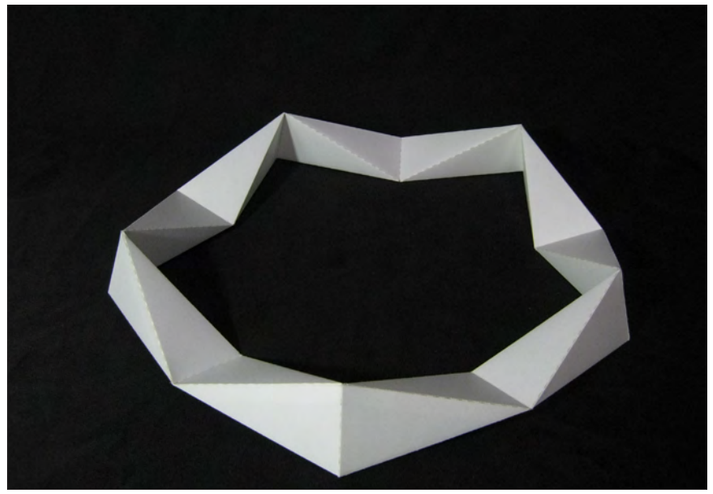




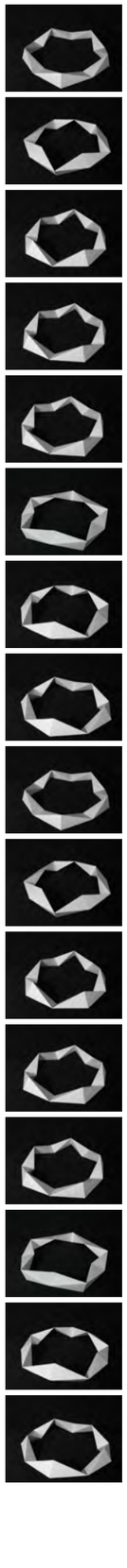


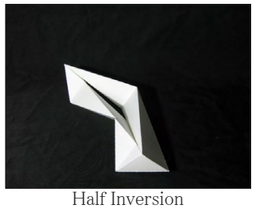
展開図

4－4－E.　Application Model: Polygon(over 10-bar)



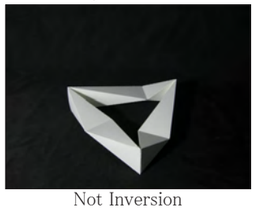
リンクの数が10個以上の状態。多自由度の変形をする。



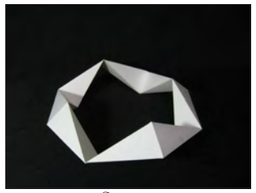

Half Inversion


Over Angle

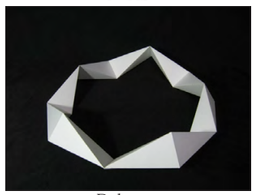

Change to Minimum


Not Inversion

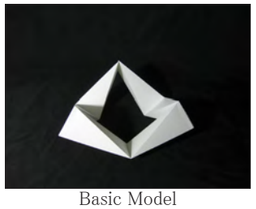

7-bar

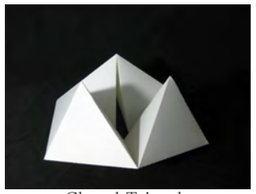

Square

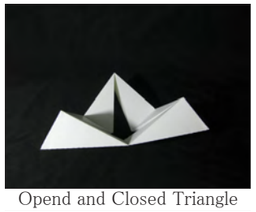

9-bar

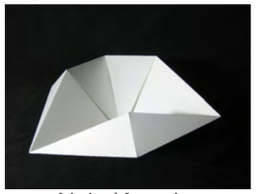

Polygon

４－４－Ｆ. 特殊モデル： 二角形（４つのボリューム）

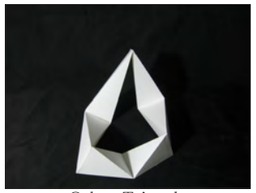

Basic Model

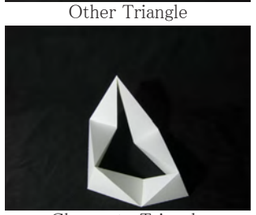

Closed Triangle

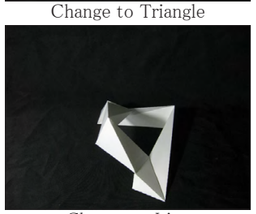

Opend and Closed Triangle

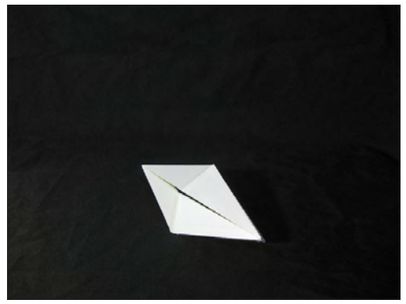

Limited Inversion

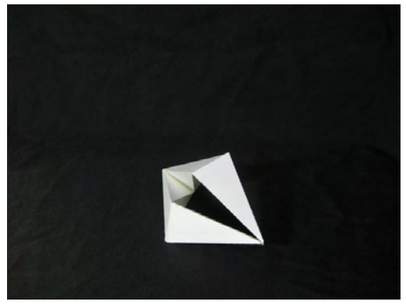

Stopped Inversion

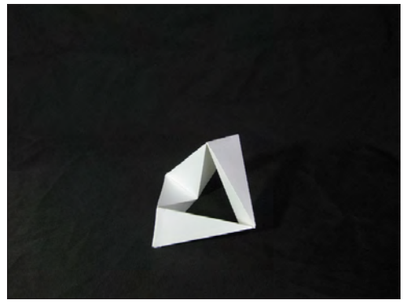

Other Triangle

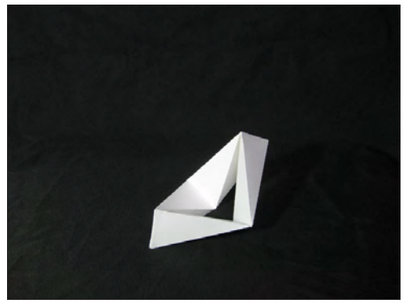

Change to Triangle

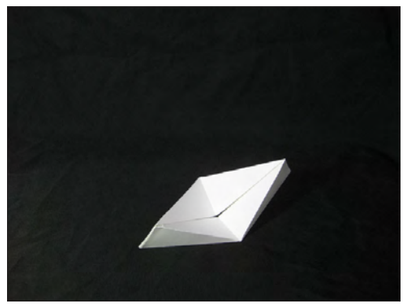

Change to Line



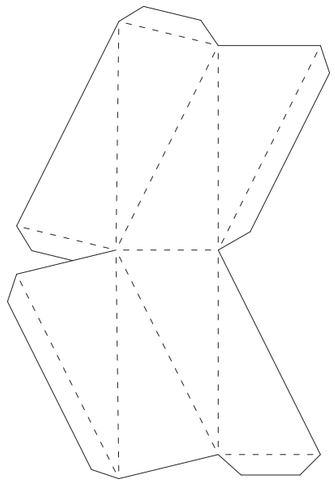


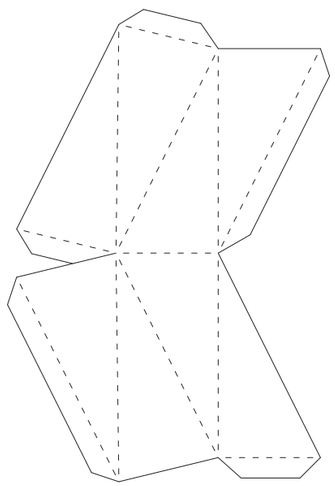






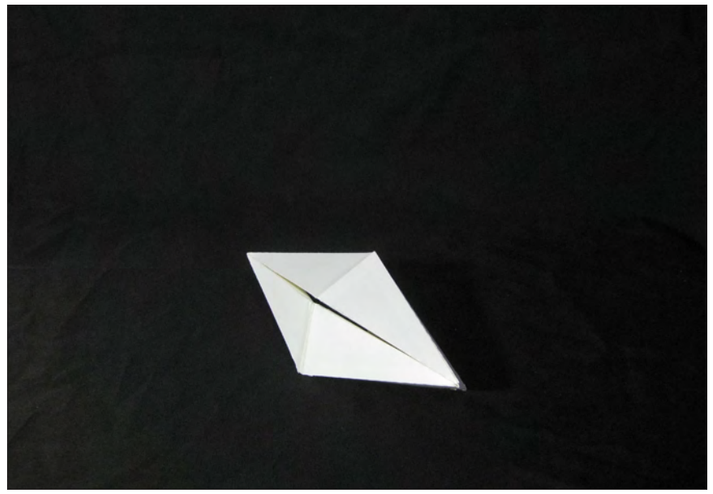
回転写真



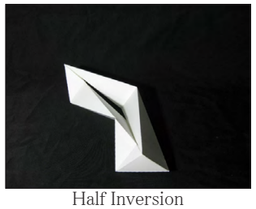


展開図

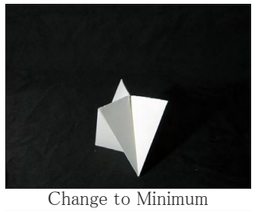
4－4－F． Another Model: Bennett(4-bar Linkage)

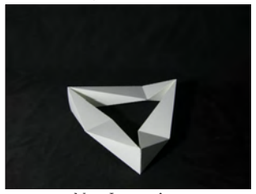


リンクの数が4つの状態。基本形のピンジョイントの角度関係と関係が異なる状態でないと成立しないため、存在はするが例外として取り扱う。Bennett Linkage と呼ばれる。

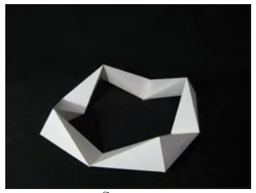

Half Inversion

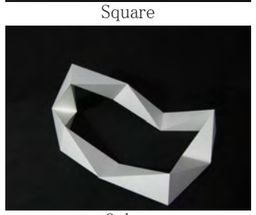

Over Angle

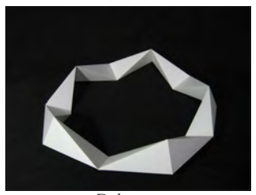

Change to Minimum


Not Inversion


7-bar

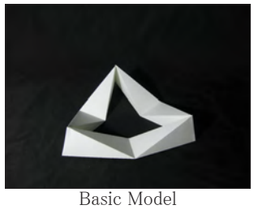

Square

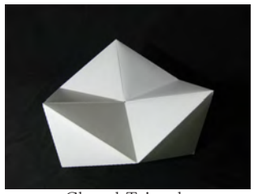

9-bar

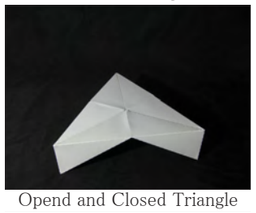

Polygon

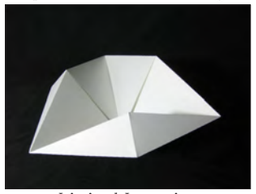
４－４－G. 特殊モデル：２.５角形（５つのボリューム）

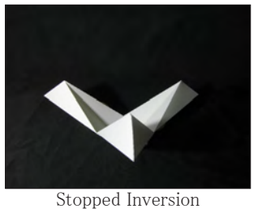
Ｎｏ　Ｄａｔａ

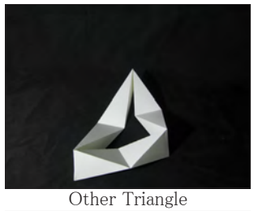

Basic Model

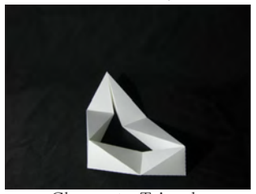

Closed Triangle

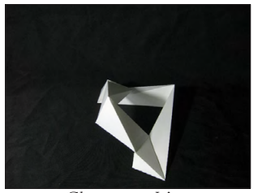

Opend and Closed Triangle


Limited Inversion

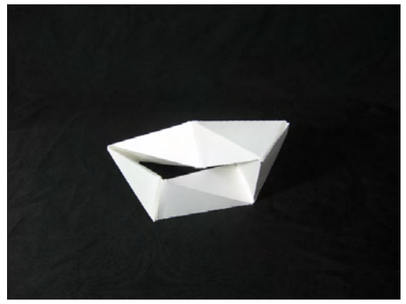

Stopped Inversion

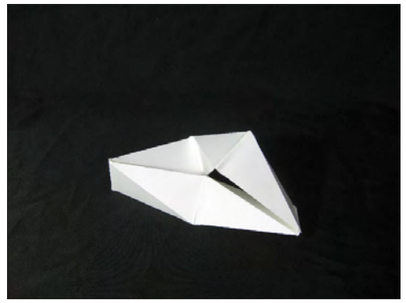

Other Triangle

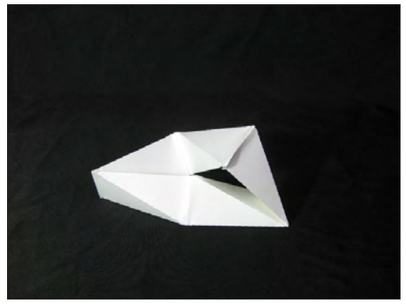

Change to Triangle


Change to Line

回転写真

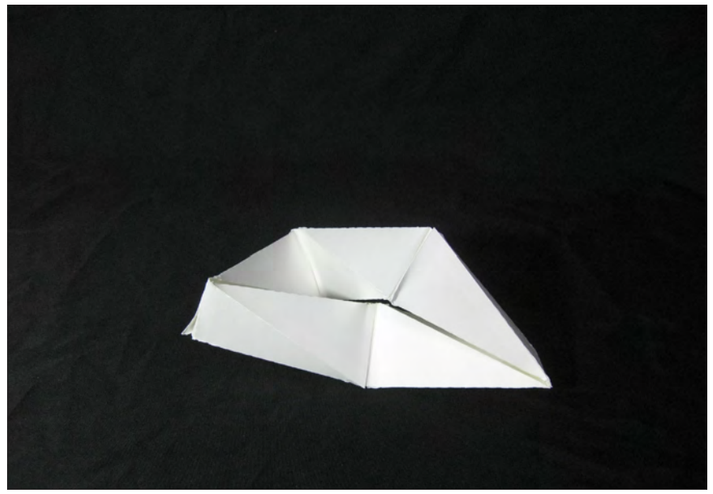




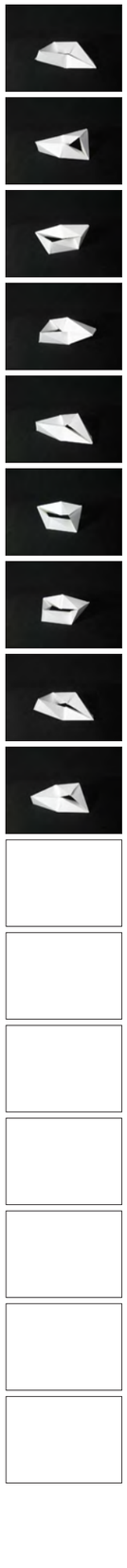


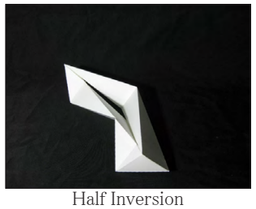
展開図

４－４－G.　Another Model: 5-bar(5-bar Linkage)



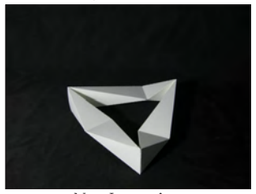
リンクの数が5つの状態。恐らく成り立つ。基本形のピンジョイントの角度関係と関係が異なる状態でないと成立しないため、存在はするが例外として取り扱う。

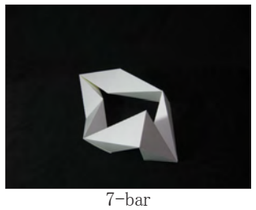


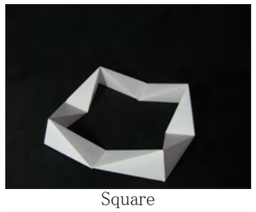

Half Inversion


Over Angle

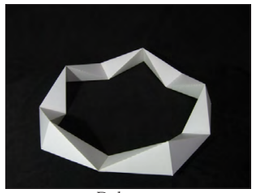

Change to Minimum


Not Inversion


7-bar


Square


9-bar


Polygon

增加

Increse

５．増加

ドア2つを１組とした辺として扱い、それらを平面的に構成、あるいは立体的に構成することによって起こる変形について述べる。環状の構成でなくなると、くるくる回転し続けることが出来ず、変形を起こすものとなる。


Dome

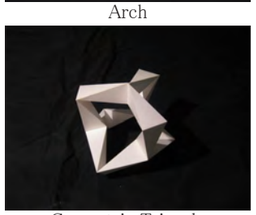

Dome

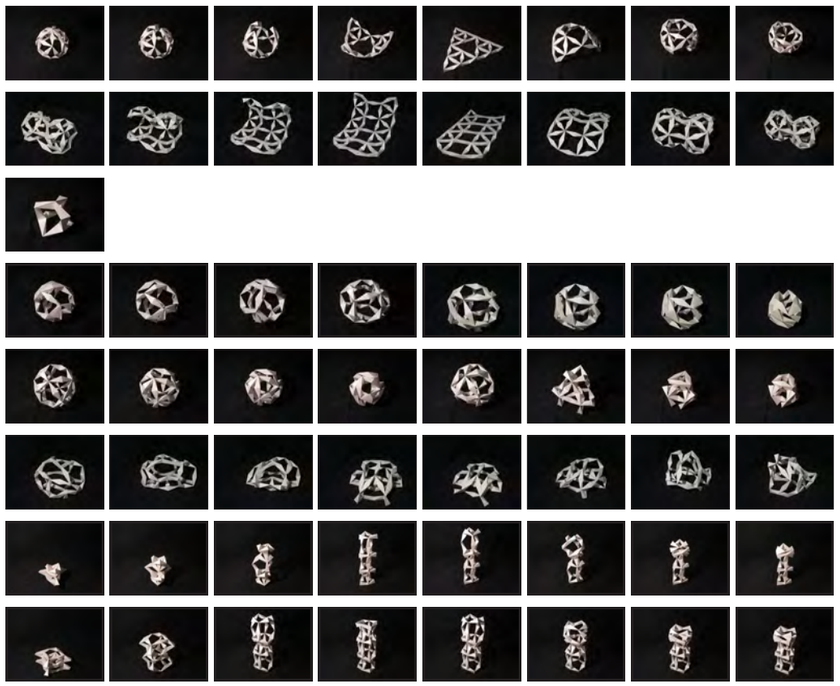

Dome


Dome


Dome


Arch

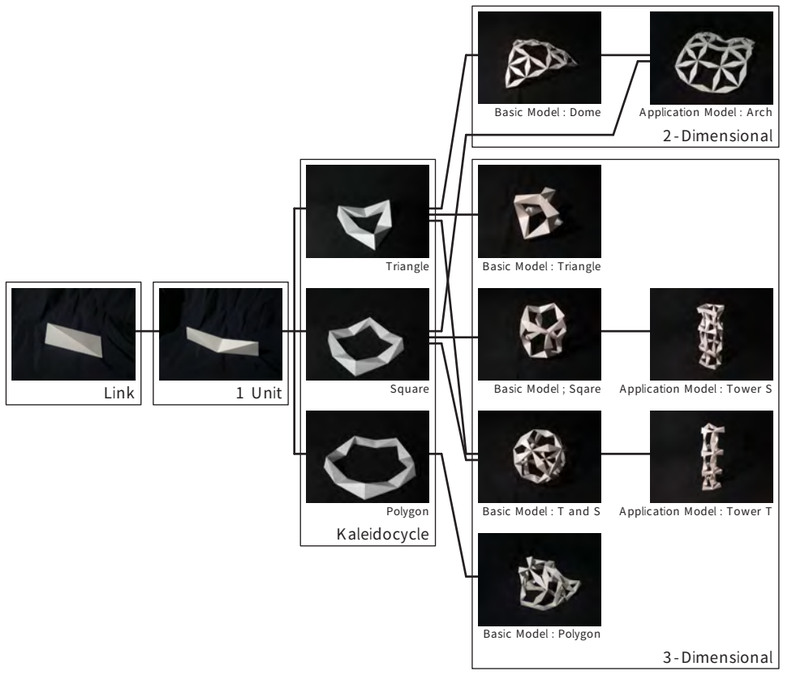

Geometric Triangle



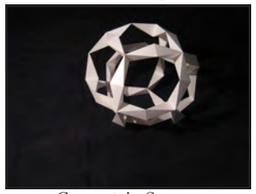
5．Increse

次に Kaleidocycle を組み合わせることによる展開方法について述べる。本研究では四面体を2つで1組とした構成をとり、Kaleidocycle 単体は3辺の構成による三角形と捉える。環状の構成を越えた時点で反転運動は途中で止まり、この展開方法は反転運動ではなく、変形を起こすものとなる。

平面で構成し変形を起こす事例と、立体で構成し変形を起こす事例の二つに分けて述べる。



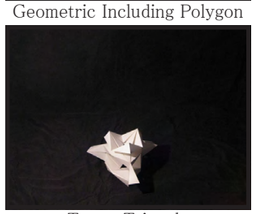

Geometric Square

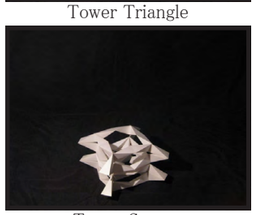

Geometric Square


Geometric Square


Geometric Square and Triangle


Geometric Including Polygon


Tower Triangle


Tower Square

５－１．平面的構成

ドアを2次元的に配置する。曲がる様に変形する。

５－１－A．基本モデル：ドーム

５－１－B．発展モデル：アーチ


Dome

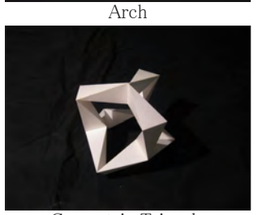

Dome

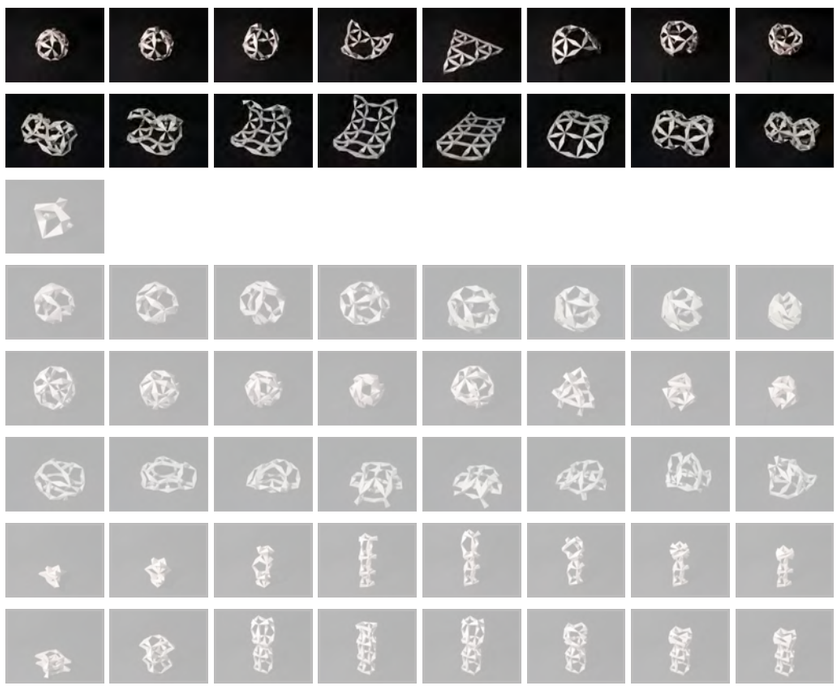

Dome


Dome


Dome


Arch


Geometric Triangle

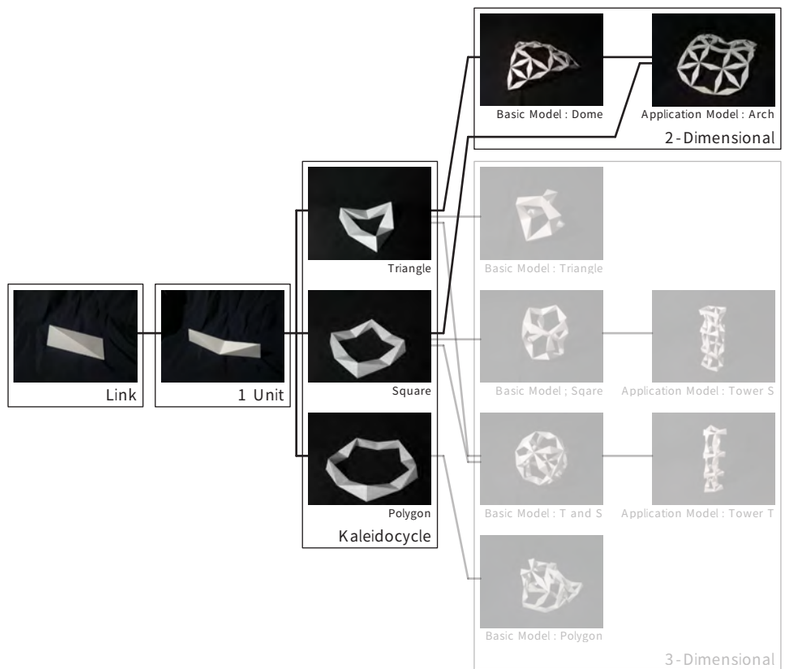


5－1．2-Dimensional

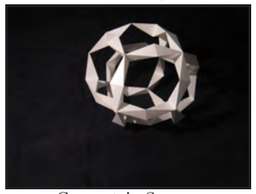
2次元上における構成。平面状態から立ち上がるように変形する。

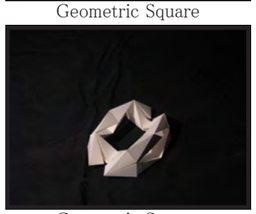
5－1－A．Basic Model: Dome

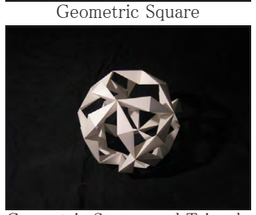
5－1－B．Application Model: Arch

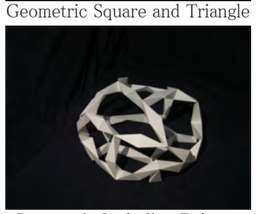



Geometric Square


Geometric Square


Geometric Square


Geometric Square and Triangle


Geometric Including Polygon

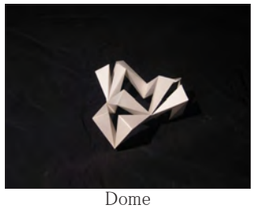

Tower Triangle


Tower Square

５－１－Ａ．基本モデル： ドーム


Dome

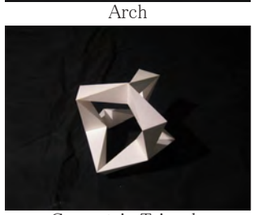

Dome

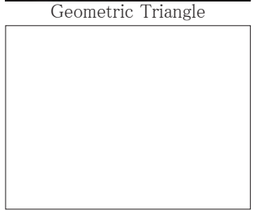

Dome

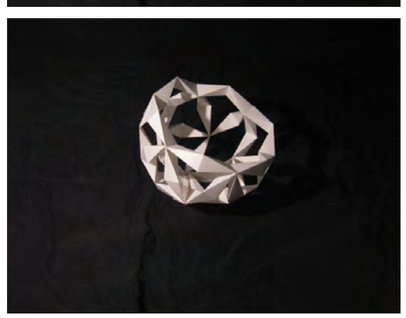

Dome


Dome

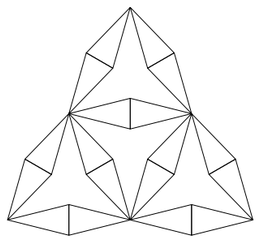

Arch

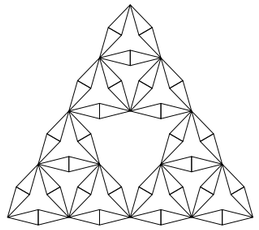

Geometric Triangle

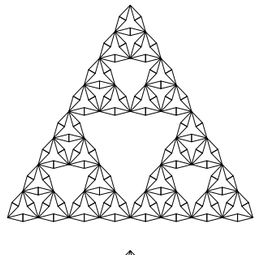


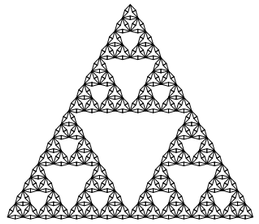


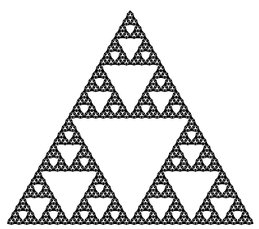
変形写真

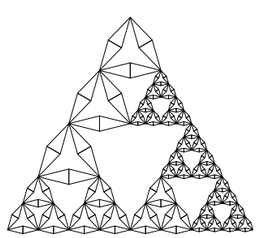






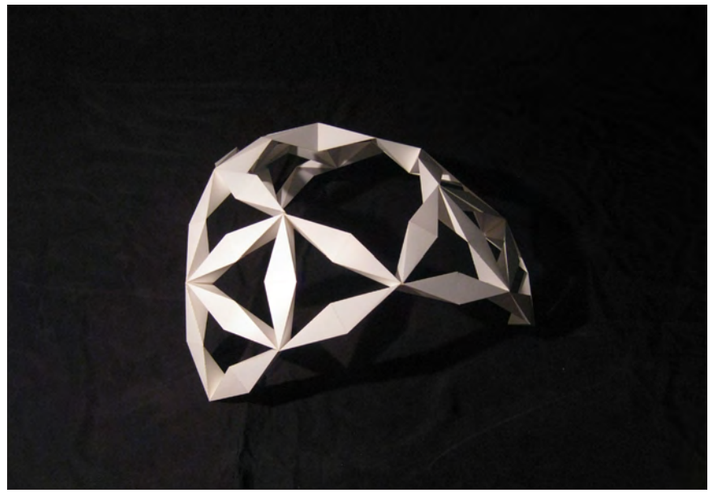




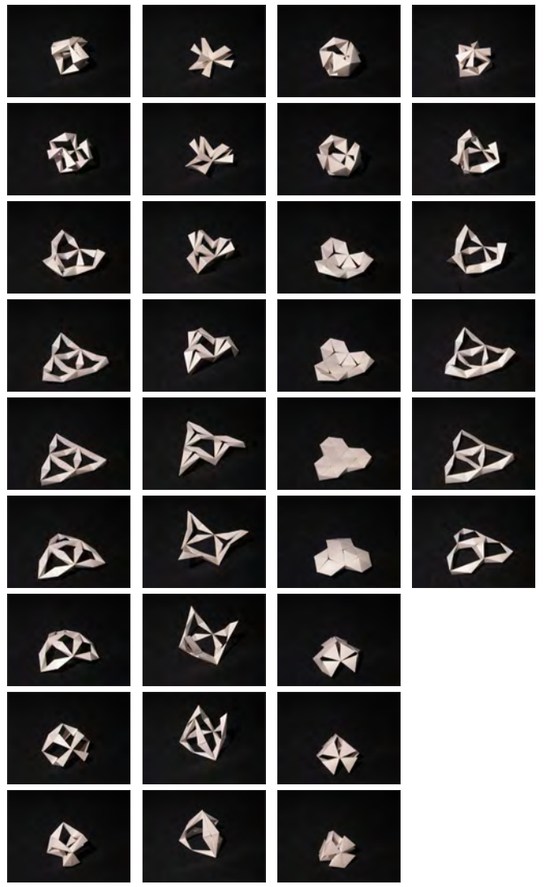


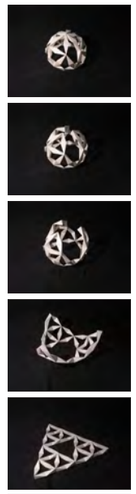
展開図

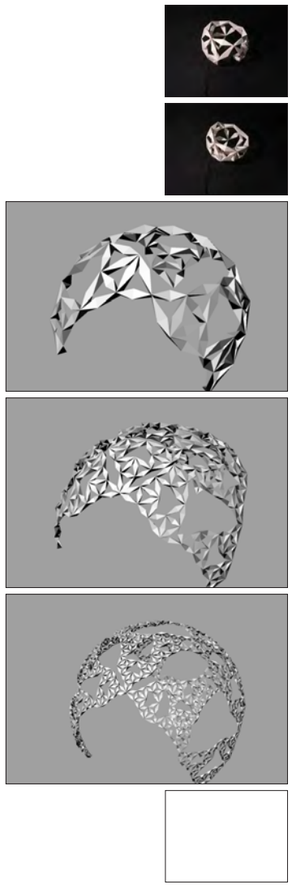
5－1－A．Basic Model: Dome



単純に三角形を連続させたもの。平面からドーム状、球体状へと自由度1で変形する。Fractal のひとつである Sierpinski gasket のように増やしていくことによって、細分化、巨大化が可能。細分化していくことによって、変形はより球体状になる。また、平面状で充填された状態から開くように変形することが可能。

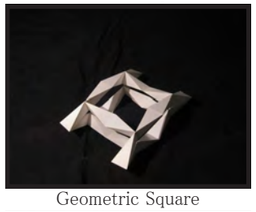






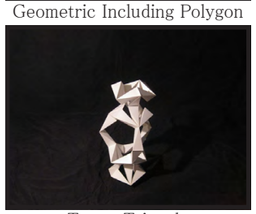

Geometric Square


Geometric Square

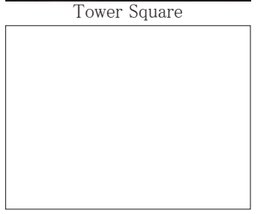

Geometric Square


Geometric Square and Triangle


Geometric Including Polygon


Tower Triangle


Tower Square



５－１－Ｂ．発展モデル： アーチ


Dome


Dome


Dome

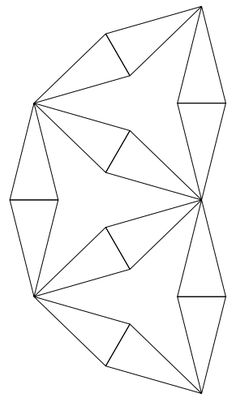

Dome

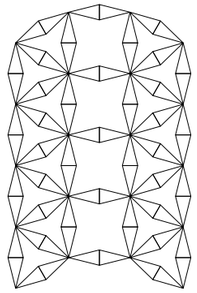

Dome

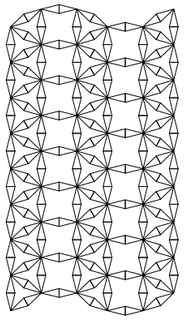

Arch

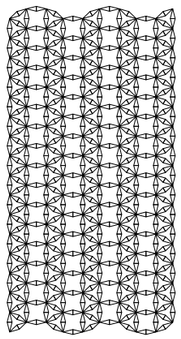

Geometric Triangle

変形写真



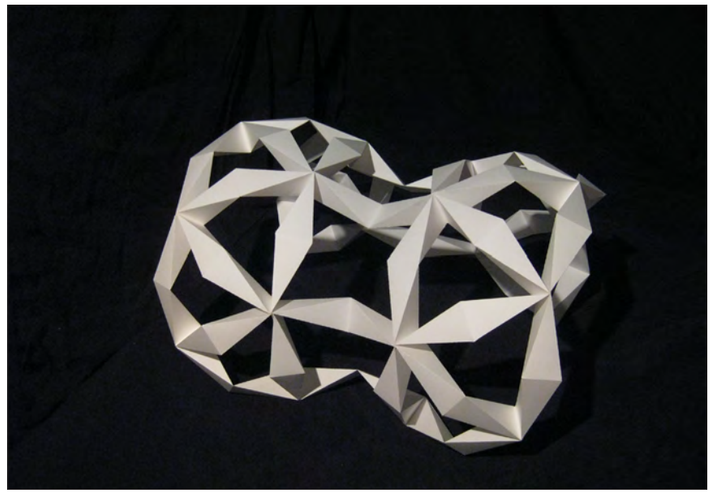






展開図

5－1－B．Application Model: Arch

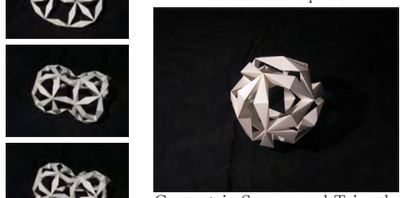


三角形を直線状に連続させたものを四角形で繋いだもの。平面からアーチ状、筒状へとほぼ自由度1で変形する。三角形の連続を増やすことによって、変形はより円状になり、四角形で繋ぐ量を増やすことによって、アーチの長さを変えることが可能。


Geometric Square

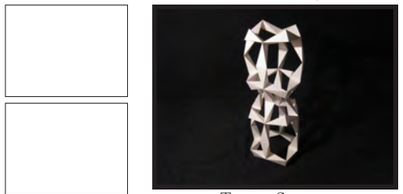

Geometric Square


Geometric Square


Geometric Square and Triangle


Geometric Including Polygon

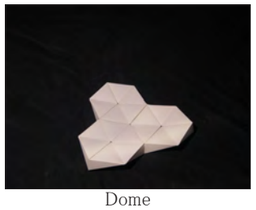

Tower Triangle


Tower Square


Dome

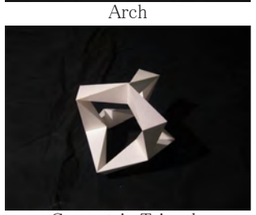

Dome


Dome


Dome


Dome


Arch


Geometric Triangle

５－２．立体的構成

ドアを3次元的に組む。伸縮するように変形する。

５－２－A．基本モデル：多面体

1. 三角形
2. 四角形
3. 三角形と四角形
4. 多面体を含む

５－２－B．発展モデル：タワー

1. 三角形
2. 四角形

5－2．3-Dimensional

3次元上における構成。伸縮するかたちで変形する。

5－2－A．Basic Model: Geometric

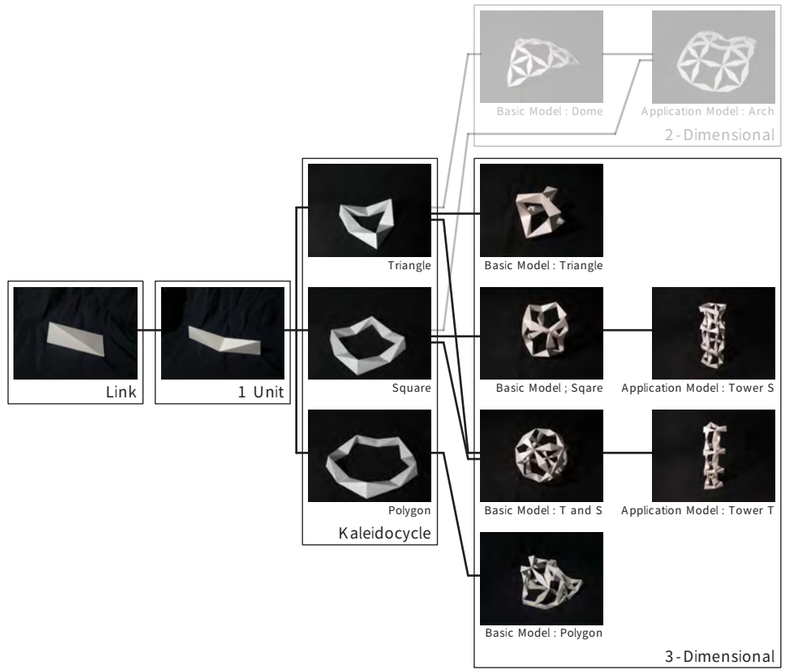
多面体的に構成したもの。

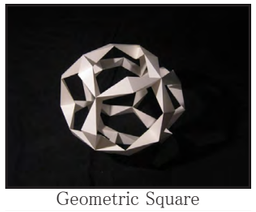
1．Triangle
2．Square
3．Triangle and Square
4．Including Polygon

5－2－B．Application Model: Tower

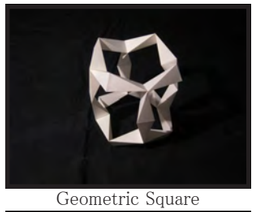
多面体を積み上げる構成をしたもの。変形の流れがいくつか存在し、ひとつのモデルでそれを組み合わせることによって部分で拡張、収縮した状態で静止させることが可能。

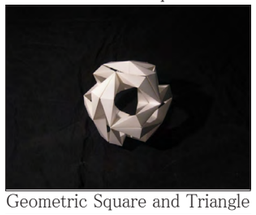
1．Triangle
2．Square



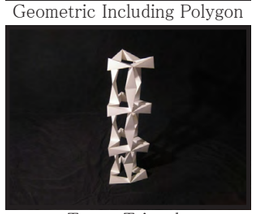

Geometric Square

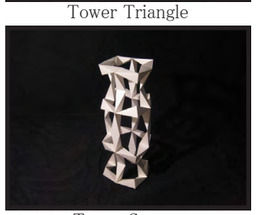

Geometric Square


Geometric Square


Geometric Square and Triangle


Geometric Including Polygon


Tower Triangle

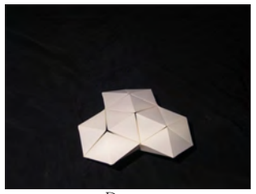

Tower Square

５－２－Ａ．基本モデル：多面体　三角形


Dome


Dome

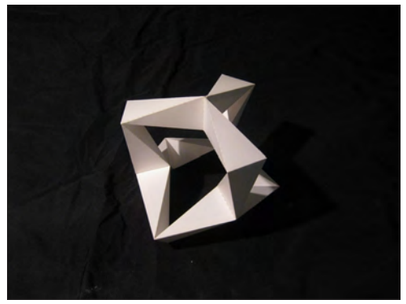

Dome

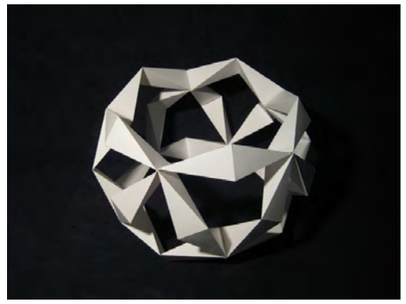

Dome

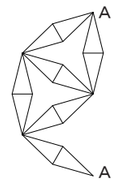

Dome


Arch

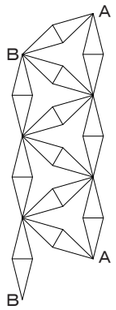

Geometric Triangle



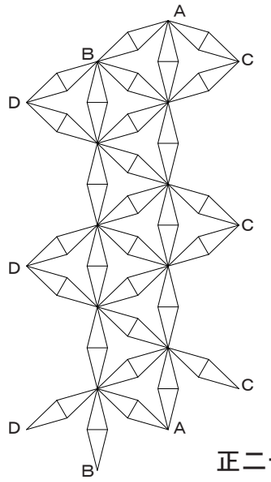




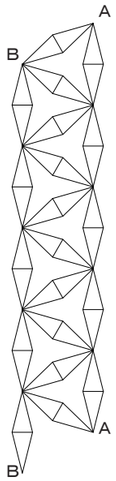
正四面体



正八面体



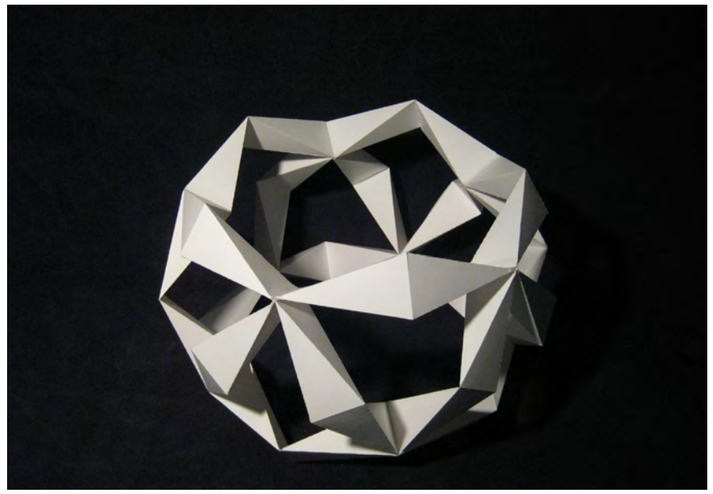
正二十面体



正反五角柱

変形写真　　展開図

5－2－A．Basic Model: Geometric Triangle



三角形のみで構成された多面体、正四面体、正八面体、正二十面体は、変形しない。また、三角形の帯で構成される反角柱も同様に変形を起こさない。


Geometric Square


Geometric Square


Geometric Square


Geometric Square and Triangle


Geometric Including Polygon


Tower Triangle

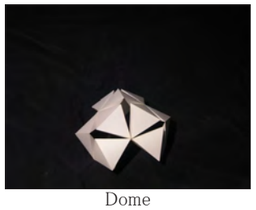

Tower Square

５－２－Ａ．基本モデル： 多面体　四角形


Dome


Dome


Dome

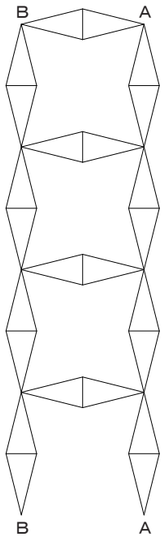

Dome


Dome

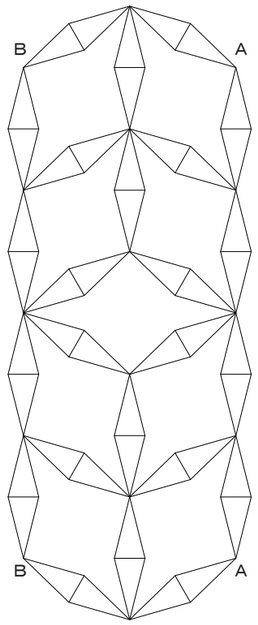

Arch


Geometric Triangle

変形写真



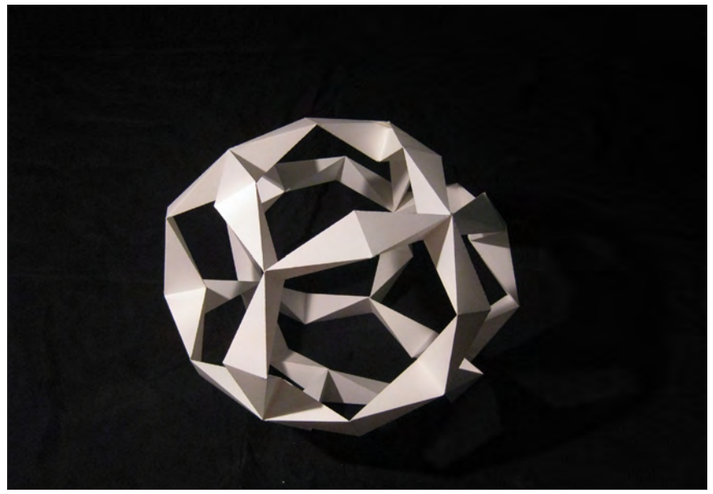
立方体



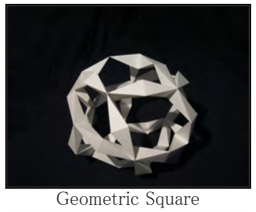
菱形十二面体

展開図

5－2－A．Application Model: Geometric Square

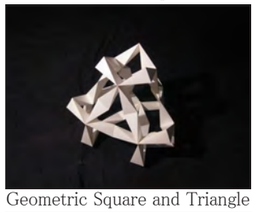


四角形のみで構成された多面体は（恐らく）変形させると四角形が重なった状態まで縮小することが出来る。


Geometric Square


Geometric Square


Geometric Square


Geometric Square and Triangle


Geometric Including Polygon


Tower Triangle


Tower Square

５－２－Ａ．基本モデル：多面体　三角形と四角形


Dome

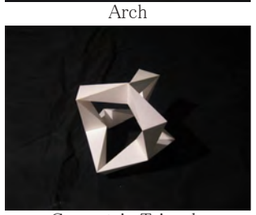

Dome


Dome

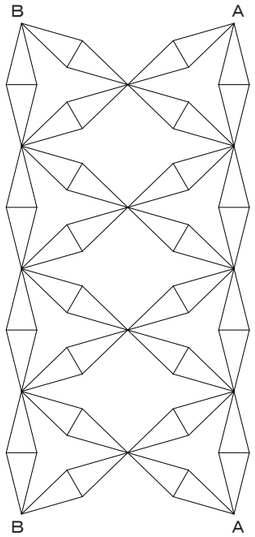

Dome


Dome


Arch


Geometric Triangle

変形写真



立方八面体

展開図

5－2－A. Application Model: Geometric Triangle and Square

三角形と四角形で構成された多面体は、全体が伸縮するようなかたちで変形する。


Geometric Square


Geometric Square


Geometric Square


Geometric Square and Triangle


Geometric Including Polygon


Tower Triangle


Tower Square

５－２－Ａ．基本モデル：多面体　多面体を含む


Dome

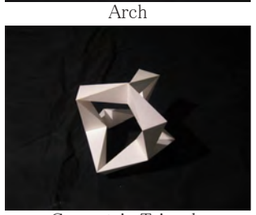

Dome


Dome

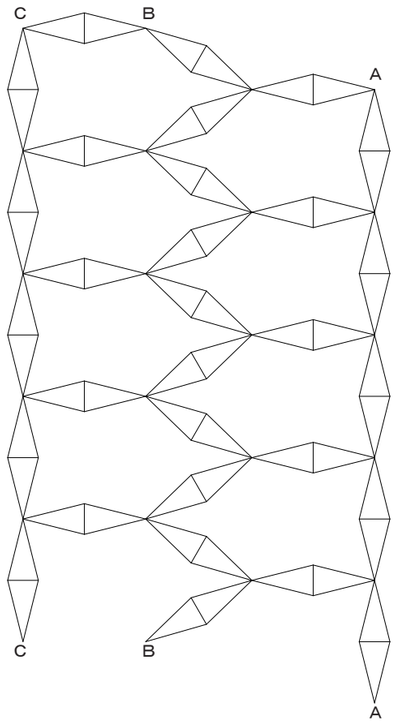

Dome


Dome


Arch


Geometric Triangle

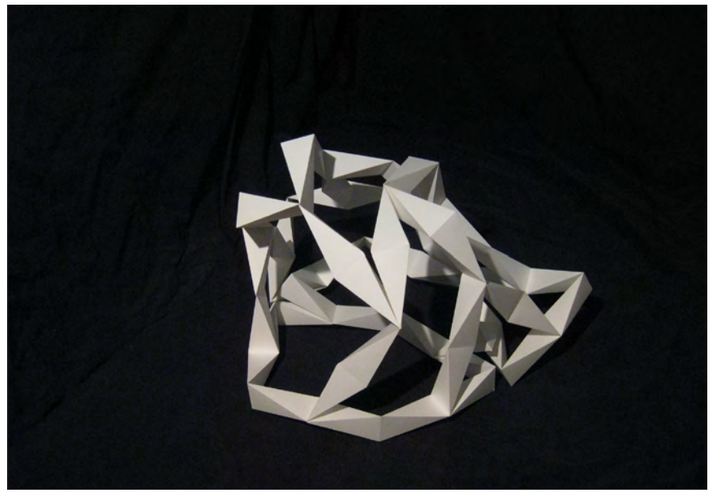
変形写真



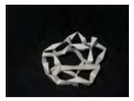
正十二面体

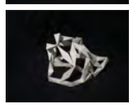
展開図

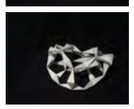
5－2－A.　Application Model: Geometric Including Polygon

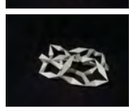


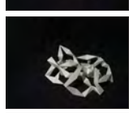
五角形以上の多角形を含む構成の多面体は、辺中のピンジョイントの角度が多面体の内側から見たとき180度を越える変形をし、多自由度の変形をする。

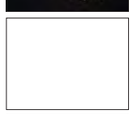


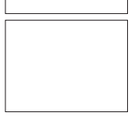


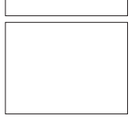













Geometric Square


Geometric Square


Geometric Square


Geometric Square and Triangle


Geometric Including Polygon


Tower Triangle


Tower Square

５－２－Ｂ．発展モデル：タワー　三角形


Dome

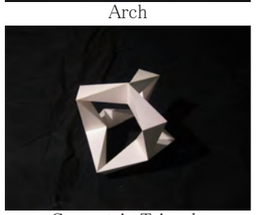

Dome

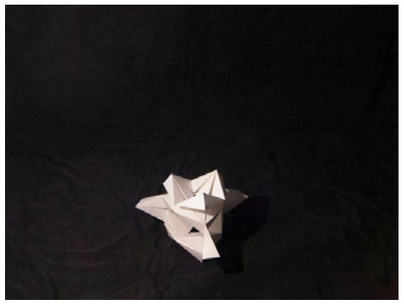

Dome


Dome

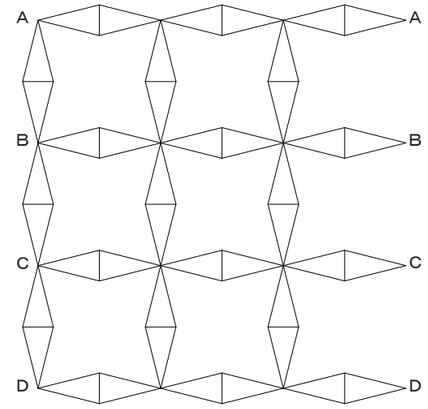

Dome


Arch


Geometric Triangle

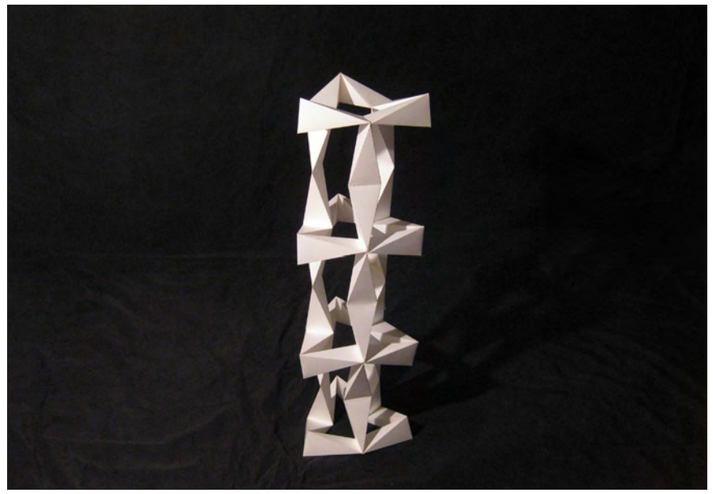


変形写真

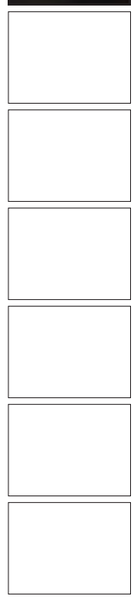


展開図

5－2－B．Application Model: Tower Triangle



三角柱を積み上げた構成。低い自由度で変形し、大きな伸縮を得られる。



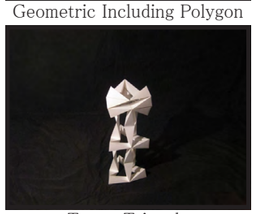

Geometric Square


Geometric Square


Geometric Square


Geometric Square and Triangle


Geometric Including Polygon

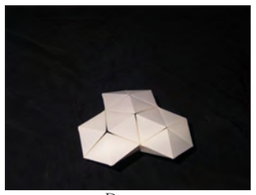

Tower Triangle


Tower Square


Dome

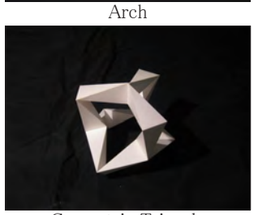

Dome

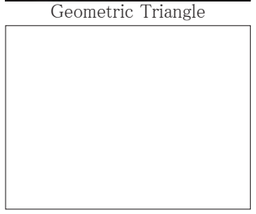

Dome


Dome


Dome

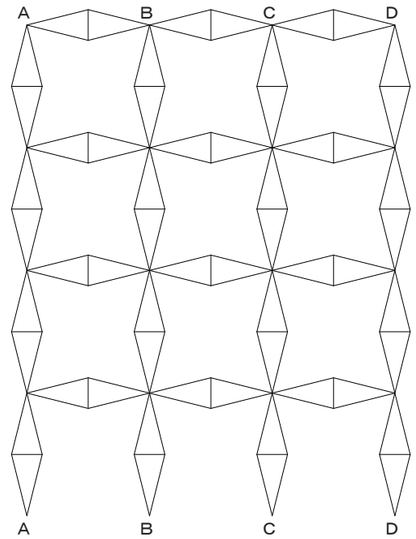

Arch


Geometric Triangle



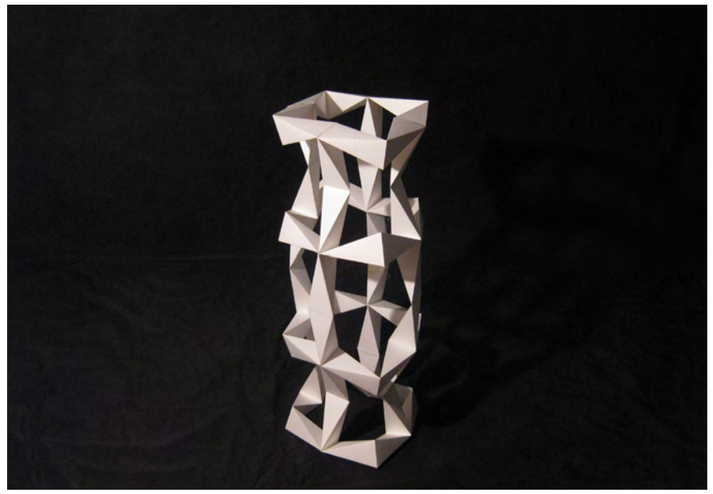
５－２－Ｂ．発展モデル：タワー　四角形

変形写真

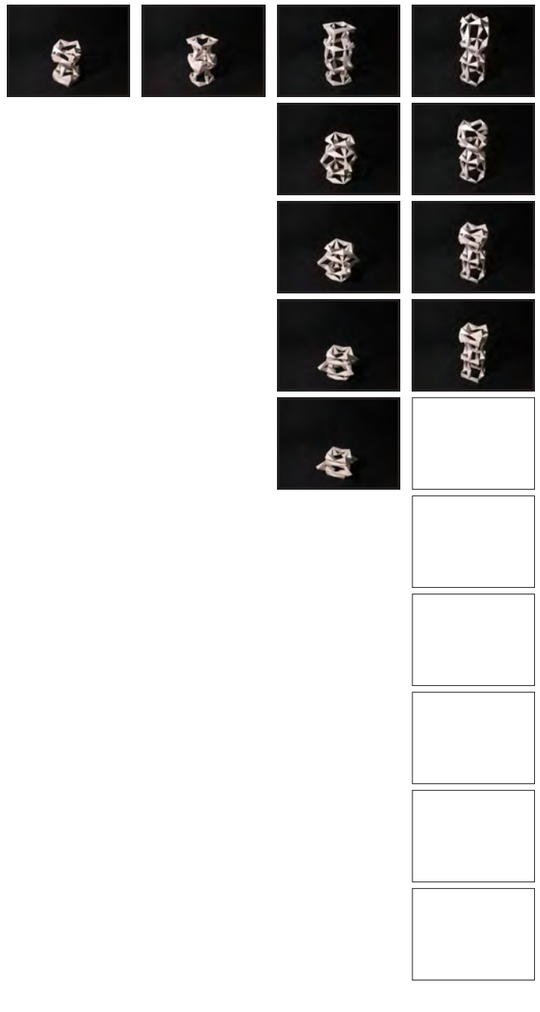


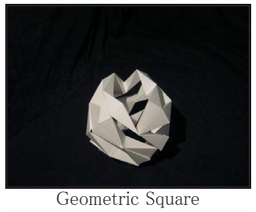
展開図

5－2－B．Application Model: Tower Square

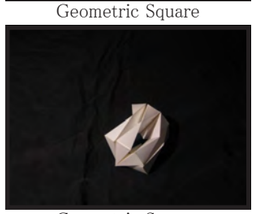


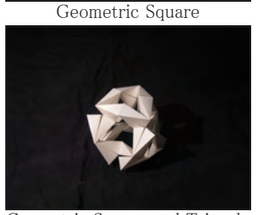
立方体を積み上げた構成。Triangle よりやや高い自由度で変形し、大きな伸縮を得られる。



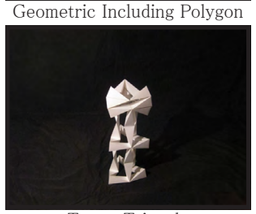

Geometric Square


Geometric Square

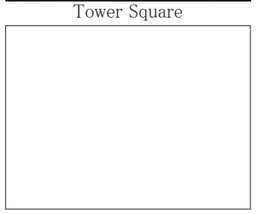

Geometric Square


Geometric Square and Triangle


Geometric Including Polygon


Tower Triangle


Tower Square

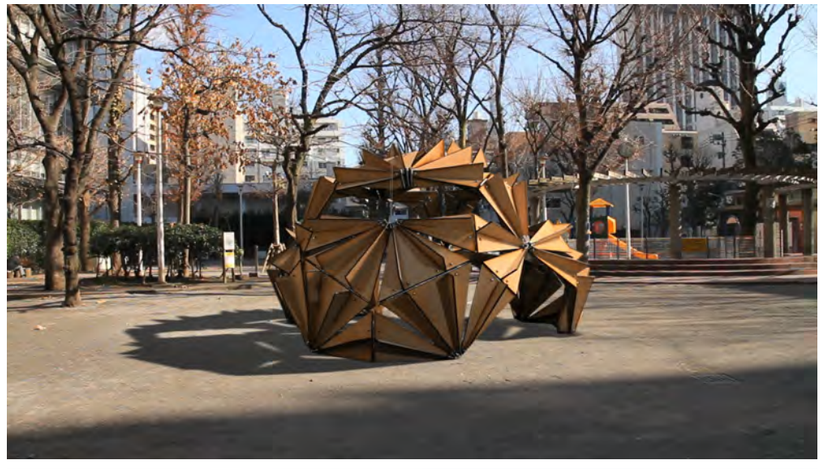


おわりに

Afterword

６．おわりに

研究の結果として、増加による展開のドーム・タワー・多面体の3つのモデルを制作した。



6－1．Afterword

研究の結果として、3つのモデル〝Dome〟、〝Geometric Square〟、〝Tower Square〟について制作を行った。それぞれ、〝Dome〟は平面形で充填を行えること、〝Geometric Square〟は四角形の状態まで収縮可能なこと、〝Tower Square〟は大きな伸縮を得られることが特徴で、実スケールでの可能性を感じた。

この研究のきっかけは大学の先輩の御幸さんと喫煙所で話したことからである。御幸さんの制作のお手伝いをしている中で、自分はいったい何を修士制作でするべきか悩んでいたところ、御幸さんからひとつのアイデアをいただいた。

『立体的なリンク機構に興味は持っているけど、今やるほど時間がない』

その言葉をきっかけに本制作は始まった。

まずは既存の面白い玩具を徹底的に調べた。そこでテオヤンセンやチャックホバーマンに出会った。彼らの洗練された玩具を眺めながら果たして自分はいったい何が出来るのだろうかと想いを馳せた。

その他にも、とにかくリンク機構と呼ばれる仕組みを少しでも使っているであろう物を調べ続けた。リンク機構は玩具になって世間に浸透されている。

リサーチを進めていく中でひとつの玩具に出会った。『Invertible Cube』である。それが実は純粋に立体的なリンク機構のみを使った唯一の事例だということを知った。さらに面白いことに、この Invertible Cube は 1929 年に発明されたにもかかわらず、その先に発展展開がほぼ行われずにいることが分かったのだ。フラーが Jitter bug を発明する 20 年も昔に発明されてから、そのかたちをほぼ変えずにいるのだ。後の 1977 年に『M.C. Escher Kaleidocycles』という書籍にて Kaleidocycle という名で登場し、少し発展が見られたが。Invertible Cube から Kaleidocycle と名を変えたそれは、よりその仕組みについて読み解かれるようになったようである。

そんなことを知らずに、修士制作の中でこの Invertible Cube を複数つけてみたらどうか、と試しに複数くっつけてみたら、かなり面白い動きをすることが分かった。そのあたりからこの Invertible Cube を深く調べていくことになった。

1年間という短い期間の中で、かなり多くの既往研究を調べたつもりだが、その中ではこのような展開をさせたものは見つからなかった。そこから勢いにのって発展展開を試み続け、気づけば年を越していた感じである。

物理学や工学の専攻ではない自分のこの発見を誰かが拾い、世に広まっていくことを願っている。

建築とはかなりかけ離れた方向性で進めていくことを推奨してくれた、研究室の先生である北川原先生には大変感謝しています。また、建築科というアカデミックの機関でありながらこんな研究を認め、吉田五十八賞という賞まで下さった芸大の教授の方々、様々な意見を下さった助手さんや周りのみんな、ほとんど笑いをとる為だけの映像制作に協力してくれたみんな、なによりきっかけをくれた御幸さん、ポールシャッツさん、ありがとうございました。

最後に、自分の意思を尊重し遠くで見守り大学院まで好きなことをさせてくれた家族、父、母、兄に感謝しています。ありがとう。

PS. こんな自分を雇ってくれる企業や事務所の方、絶賛募集中です。

2013.01.25 くわばらりょうすけ

6－2.　Reference

1)　シザーズ型展開構造とは、E. P. Pinero による展開構造をはじめとした、製図道具の Pantograph をいくつも連続的に組み合わせたような展開構造。

2)　通常、構造内部に回転するジョイントやスライドするメカニズムをもち、収縮、拡張する構造。

3)　Emilio Perez Pinero, ″Three Dimensional Reticular Structure″, United States Patent 3,185,164, 1965.

4)　Richard Buckminster Fuller, ″Synergetics″, 1982, 190-215.

5)　三浦公亮が考案した折り畳み方。日本名はミウラ折り。

6)　Chuck Hoberman, ″Iris Dome″, 2000, Hanover, Germany.

7)　川口衛氏が考案した構法、パンタドーム構法。1995 年、なみはやドームの施工の際に初めて試みられた。非鉛直方向のリフトアップを行った世界で初めての事例。他に、なら百年会館なども同様の方法で施工が行われた。

8)　S. Pellegrino, ″Deployable Structures″, 2001. ここで取り上げられているものは、リンク機構に限らない視点でその動きを数学的、物理学的にまとめているものだと思われる。

9)　Yan Chen,″Design of Structural Mechanisms″, 2003. リンク機構の発明の中の一部を数式によってまとめている。本研究の視点である Kaleidocycle についても ″Shatz linkage″ という名で登場し単純な形状についてのみ、まとめられている。

10)　Paul Schatz, ″Invertible Cube″, 1929.

11)　Doris Schattschneide, ″M. C. Escher Kaleidocycles″, 1977.

12)　製図道具。

13)　Akira Nishihara, ″Tile magic″, 1996,.
URL: http://www1.ttcn.ne.jp/a-nishi/

14)　京都大学大学院工学研究科機械理工学専攻 ,「コンプライアントグリッパー」

15)　Theo Jansen, ″STRANDBEEST″, 1990.

16)　Chuck Hoberman, ″Switch Pitch″, 2004.

17)　Hans Haupt, ″Knirps″, 1928.

18)　阿竹克人 ,「あたけぽね」, 2000s.

19)　戸村浩 ,「MOVE FORM」, 1964.

20)　Akira Nishihara, ″Flip-Flop Ball″, 2000.
URL: http://www1.ttcn.ne.jp/a-nishi/

21)　ひろがる玩具という名を勝手につけたが、Flip-Flop Ball や Jitter Bug などに似た構造で、美術館の売店などに時々売っている名称不明のもの。

22)　Chuck Hoberman, ″Hoberman Sphere″, 1995.

23)　柳瀬順一 ,「Juno's Spinner」, 1988.
URL: http://www.polyhedra.jp/

24)　御幸朋寿 ,「GOKO-ORI」, 2008.

25)　Jeff Beynon, ″Origami Spiral″

26)　ミウラ折りに厚みを持たせたもの、あるいはそれを積層させた物などを制作した。

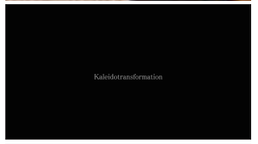
27)　Bennett, ″Bennett Linkage″, 1903.

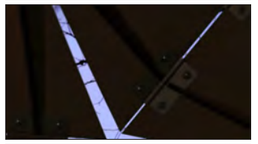
28)　6つの四面体の構成による形態変化については論文 ″Design of Structural Mechanisms″ で取り上げられている。ここで、その発展事例として Skew position Link を3つのピンジョイントにて展開させる事例を出しているが、本研究では 2 つのピンジョイントのみに絞って進めることにした。

29)　カスパー・シュワーベ＋石黒敦彦 ,「ジオメトリック・アート」, 2006.

30)　海外ではプログラミングによる CG アニメーション（Marcus Engel, ″M. C. Escher Kaleidocycles″, http://www.kaleidocycles.de/index.shtml）であったり、日本でも大西俊弘氏がいくつかの論文を出している。

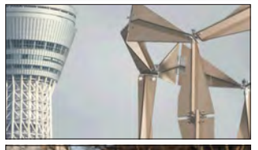






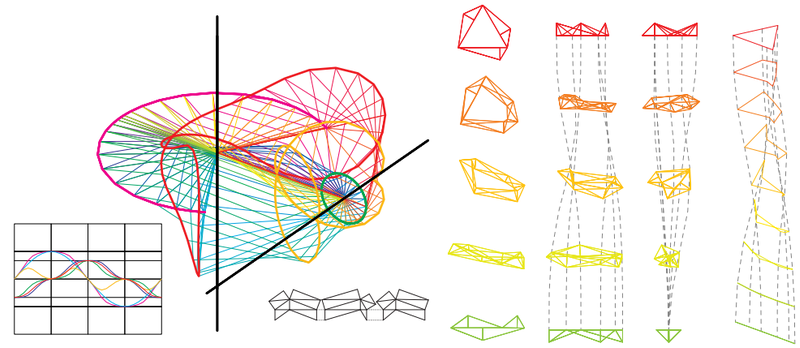






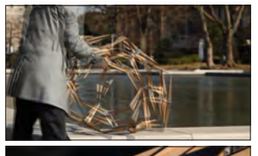
6－3. Script

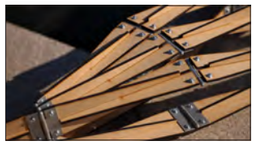
研究の際に Grasshopper の C#Script にて Kaleidotransformation の反転運動のプログラム化を行った。″Metamorphosis″で起こりうる形態を関数化し、変数を変えることによってその反転運動の仕方をビジュアル化した。



```
private void RunScript(
    Point3d P0b, Point3d P1a, Point3d P2a, Point3d P3a, double Range, double DistA, double DistB, int Case,
double LA, double MA, double LB, double MB, bool Thick, bool onPlane, double x, ref object Kaleidocycle,
ref object KaleidocycleMir, ref object KaleidoPoint, ref object KaleidoPointMir, ref object KaleidoVector, ref
 object KaleidoVectorMir, ref object KaleidoLineA, ref object KaleidoLineAMir, ref object KaleidoLineB, ref
object KaleidoLineBMir, ref object KaleidoAngleA, ref object KaleidoAngleB, ref object KaleidoLength)
 {

  /* ■■■■■■■■■■ MAKE KALEIDOCYCLE CASE ■■■■■■■■■■ */

  /* ■ set up X : Case1, Case2, Case3 ■ */
  //using data
  Point3d P0a = new Point3d();
  //    Range = Range % 1;//you can use Range over 1;
  //value of using data

  if(Case == 1){
    Print("Case1:□Infinite,Plane|");
    if(P0b == nullPoint){
      P0a = InfPoint;
      P0b = circumcenterPoint(P1a, P2a, P3a);
    } else {
      P0a = InfPoint;
      P0b = PLPclosestPoint(P0b, P1a, ABCnormalVector(P1a, P2a, P3a));
    }
  } else if(Case == 2){
    Print("Case2:□Infinite,Circumcenter|");
    P0a = InfPoint;
    P0b = movePoint(circumcenterPoint(P1a, P2a, P3a),
      ABCnormalVector(P1a, P2a, P3a), DistB);
    if(DistB > 0){
      Print(" ■■ error[111] in Case2 :this case has error.");
    }
  } else if(Case == 3){
    Print("Case3:□Circumecenter,Circumecenter|");
    Print(" ■■ error[123] in Case3 : this case has so many error...");
    P0a = movePoint(circumcenterPoint(P1a, P2a, P3a),
      ABCnormalVector(P1a, P2a, P3a), DistA);
    P0b = movePoint(circumcenterPoint(P1a, P2a, P3a),
      ABCnormalVector(P1a, P2a, P3a), DistB);
    if(DistB > 0){
      Print(" ■■ error[121] in Case3 :this case has error.");
    }
  } else {
    P0a = InfPoint;
    P0b = InfPoint;
    Print(" ■■ error[133] in Case number");
  }

  if(DistB == 0 && (Case == 2 || Case == 3)){
    Case = 1;
  }

  if(Case == 2 && (Range == 0.25 || Range == 0.75)){
    Range += 0.005;
  }

  /* ■■■■■■■■■■ MAKE KALEIDOCYCLE PRESET DATA ■■■■■■■■■■ */

  /* ■ set up1 : Using Data 1 "not change in code line" ■ */

  Point3d P1b = new Point3d();
  Point3d P2b = new Point3d();
  Point3d P3b = new Point3d();

  Vector3d R1a = new Vector3d();
  Vector3d R1b = new Vector3d();
  Vector3d R2a = new Vector3d();
  Vector3d R2b = new Vector3d();
  Vector3d R3a = new Vector3d();
  Vector3d R3b = new Vector3d();

  double D1a = new Double();
  double D1b = new Double();
  double D2a = new Double();
  double D2b = new Double();
  double D3a = new Double();
  double D3b = new Double();

  double D1 = new double();
  double D2 = new double();
  double D3 = new double();
  double D4 = new double();
  double D5 = new double();
  double D6 = new double();

  /* ■ set up3 : Value of Using Data 1 ■ */

  P1b = closestPoint(P1a, P2a, P0b);
  P2b = closestPoint(P2a, P3a, P0b);
  P3b = closestPoint(P3a, P1a, P0b);

  if(Case == 3){
    R1a = normalVector(P1a, P0a);
    R2a = normalVector(P2a, P0a);
    R3a = normalVector(P3a, P0a);
  } else {
    R1a = ABCnormalVector(P1a, P2a, P3a);
    R2a = ABCnormalVector(P1a, P2a, P3a);
    R3a = ABCnormalVector(P1a, P2a, P3a);
  }

  R1b = normalVector(P1b, P0b);
  R2b = normalVector(P2b, P0b);
  R3b = normalVector(P3b, P0b);

  D1a = PointDistance(P1a, P1b);
  D1b = PointDistance(P1b, P2a);
  D2a = PointDistance(P2a, P2b);
  D2b = PointDistance(P2b, P3a);
  D3a = PointDistance(P3a, P3b);
  D3b = PointDistance(P3b, P1a);

  D1 = PointDistance(P1a, P2a);
  D2 = PointDistance(P2a, P3a);
  D3 = PointDistance(P3a, P1a);
  D4 = D1a + D3b;
  D5 = D2a + D1b;
  D6 = D3a + D2b;

  IntVector = ABCnormalVector(P1a, P2a, P3a);

  /* ■■■■■■■■■■ MAKE KALEIDOCYCLE CASE1 ■■■■■■■■■■ */

  int Case1 = new int();

  if(Case == 1){

    if((D1a > D1 || D1b > D1) || (D2a > D2 || D2b > D2) || (D3a > D3 || D3b > D3)){
      Case1 = 7;
      Print("Case1-X-3,4:□triangleA is not triangle|");
      Print(" ■■ error[205] in Case1-X-3,4: maybe exist but could not make.");
    } else if(D4 > D5 + D6){
      Case1 = 4;
      Print("Case1-1-2:□triangleB is not triangle|");
      if(ABCCos(D1, D2, D3) < 0){
        //this error cause if P0b is out of triangle[A]
        //maybe this error is link on "Case1-3".
        Print(" ■■ error[210] in Case1-1-2: exist but could not make.");
      }
    } else if(D5 > D4 + D6){
      Case1 = 5;
      Print("Case1-2-2:□triangleB is not triangle|");
    } else if(D6 > D4 + D5){
      Case1 = 6;
      Print("Case1-3-2:□triangleB is not triangle|");
      Print(" ■■ error[215] in Case1-3-2 : exist but can not make.");
    } else if(D6 > D4 + D5){


    } else if(D4 > D5 && D4 > D6) {
      Case1 = 1;
      Print("Case1-1-1:□triangleB is triangle|");
    } else if(D5 > D4 && D5 > D6) {
      Case1 = 2;
      Print("Case1-2-1:□triangleB is triangle|");
    } else if(D6 > D4 && D6 > D5) {
      Case1 = 3;
      Print("Case1-3-1:□triangleB is triangle|");
      Print(" ■■ error[231] in case1-3-1 : invert but can not make.");
    }
  }

  /* ■■■■■■■■■■ MAKE KALEIDOCYCLE LINE ■■■■■■■■■■ */

  /* ■ set up2 : Using Data 2 "change in code line" ■ */

  double Q1b = new double();
  double minAngle = new double();
  double maxAngle = new double();
  double RangeSin = new double();//sinCurve of Range
  double Q1bE = new DoubleQ;//Exterior Angle of Q1b
```

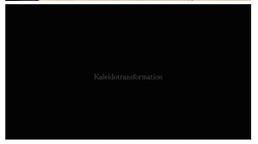
```
Point3d P2a1 = new Point3d();
Vector3d R2a1 = new Vector3d();

Point3d P2b1 = new Point3d();
Point3d P2b2 = new Point3d();
Point3d P3a1 = new Point3d();
Point3d P3a2 = new Point3d();
Point3d P3b1 = new Point3d();
Point3d P3b2 = new Point3d();

Point3d P0b1 = new Point3d();
Point3d P0b2 = new Point3d();

double D0a1a = new Double();
double D0a2a = new Double();
double D0a3a = new Double();
double D0a2b = new Double();
double D0a3b = new Double();

double Q2b = new double();
double Q3b = new double();

double D1a3a = new Double();
double D2a3a = new Double();

double Q1a1 = new double();
double Q2a1 = new double();
double Q3a1 = new double();
double Q1a2 = new double();
double Q2a2 = new double();
double Q3a2 = new double();

Vector3d R2b1 = new Vector3d();
Vector3d R3a1 = new Vector3d();
Vector3d R3b1 = new Vector3d();
Vector3d R2b2 = new Vector3d();
Vector3d R3a2 = new Vector3d();
Vector3d R3b2 = new Vector3d();

/* ■ set upX : Range of Q1b ■ */

if(Case == 1){
  if(Case1 <= 3){
    minAngle = Math.Acos(ABCCos(D6, D4, D5));
    maxAngle = Math.PI * 2 - Math.Acos(ABCCos(D6, D4, D5));
  } else {
    minAngle = 0;
    maxAngle = Math.PI * 2;
  }
} else {
  minAngle = 0 - x;
  maxAngle = Math.PI * 2 + x;
}

RangeSin = Math.Round(Math.Sin(Range * Math.PI * 2), 4);
Q1b = (RangeSin + 1) / 2 * (maxAngle - minAngle) + minAngle;

Q1bE = Q1b - Math.PI;//Exterior Angle

/* ■ set upX : Point[P2a] ■ */

P2a1 = rotatePoint(P2a, P1b, R1b, Q1bE);
R2a1 = rotateNVector(R2a, R1b, Q1bE);

/* ■ set up4 : Point Distance of Point[P0a] ■ */

//if(Q1b[degree] == 180 )
if(Q1b == Math.PI && (Case == 1 || Case == 2)){

  P0a = InfPoint;
  D0a1a = double.PositiveInfinity;
  D0a2a = double.PositiveInfinity;
  D0a3a = double.PositiveInfinity;
  D0a2b = double.PositiveInfinity;
  D0a3b = double.PositiveInfinity;

} else if(Q1b == 0 || Q1b == Math.PI * 2){

  if(Case == 1){
    P0a = InfPoint;
    D0a1a = double.PositiveInfinity;
    D0a2a = double.PositiveInfinity;
    D0a3a = double.PositiveInfinity;
    D0a2b = double.PositiveInfinity;
    D0a3b = double.PositiveInfinity;
  } else {
    P0a = P1a;
    D0a1a = 0;
    D0a2a = 0;
    D0a3a = 0;
    D0a2b = 0;
    D0a3b = 0;
  }

} else {

  //LLIntersection(P1a,R1a,P2a1,R2a)
  P0a = LLIntersection(P1a, R1a, P2a1, R2a1);

  //PointDistance(P0a, P1a)
  D0a1a = PointDistance(P0a, P1a);

  //PointDistance(P0a, P2a1)
  D0a2a = PointDistance(P0a, P2a1);

  //PointDistance(P0a, P2b)
  //Hypotenuse(D0a2a, D2a)
  D0a2b = Hypotenuse(PointDistance(P0a, P2a1), D2a);

  //PointDistance(P0a, P3a)
  //Opposite(D0a2b, D2b)
  D0a3a = Opposite(Hypotenuse(PointDistance(P0a, P2a1), D2a), D2b);

  //PointDistance(P0a, P3b)
  //Hypotenuse(D0a1a, D3b)
  D0a3b = Hypotenuse(PointDistance(P0a, P1a), D3b);

}

/* ■ set up5 : Radian[Q2b] and Radian[Q3b] ■ */

//if(Q1b[degree] == 180)
if(Q1b == Math.PI){

  Q2b = Math.PI;
  Q3b = Math.PI;

} else if(Q1b == 0 || Q1b == Math.PI * 2){
  if(Case == 1){
    if(Case1 == 4){
      Q2b = Math.PI;
      Q3b = Q1b;
    } else if(Case1 == 5){
      Q2b = Q1b;
      Q3b = Math.PI;
    }
  } else {
    Q2b = Q1b;
    Q3b = Q1b;
  }

} else if(Q1b < Math.PI){

  //Math.PI - P3Angle(P2b,P2a,P3a)
  Q2b = Math.PI - P3Angle(
    PLPclosestPoint(P2b, movePoint(P2a, R2a, D0a2a), R2b),
    movePoint(P2a, R2a, D0a2a),
    movePoint(P3a, R3a, D0a3a)
    );

  //Math.PI - P3Angle(P3b,P1a,P3a)
  Q3b = Math.PI - P3Angle(
    PLPclosestPoint(P3b, movePoint(P1a, R1a, D0a1a), R3b),
    movePoint(P1a, R1a, D0a1a),
    movePoint(P3a, R3a, D0a3a)
    );

  if(Case == 1){
    if(D0a3b < D3a){
      Print(" ■ ■ error[330] in case1-3 : exist? but could not create.");
    } else {

      if(D1a > D1b){

        //Math.PI - P3Angle(P3b,P1a,P3a)
        Q3b = Math.PI - P3Angle(
          P3b,
          movePoint(P1a, normalVector(P1a, P0a), D0a1a),
          movePoint(P3a, R3a, D0a3a)
          );

      } else if(D1a < D1b) {
        if(P3Angle(P1b, P1a, P0a) >= P3Angle(P1b, P1a, P2a1)){
          Q2b = Math.PI - P3Angle(
            P2b,
            movePoint(P2a, -R2a, D0a2a),
            movePoint(P3a, R3a, D0a3a)
            );
        }
      }
    }
  }

} else if(Q1b > Math.PI) {

  //Math.PI + P3Angle(P2b,P2a,P3a)
  Q2b = Math.PI + P3Angle(
    PLPclosestPoint(P2b, movePoint(P2a, R2a, D0a2a), R2b),
    movePoint(P2a, R2a, D0a2a),
    movePoint(P3a, R3a, D0a3a)
    );

  //Math.PI + P3Angle(P3b,P1a,P3a)
  Q3b = Math.PI + P3Angle(
    PLPclosestPoint(P3b, movePoint(P1a, R1a, D0a1a), R3b),
    movePoint(P1a, R1a, D0a1a),
    movePoint(P3a, R3a, D0a3a)
    );
  if(Case == 1){
    if(D0a3b < D3a){
      Print(" ■ ■ error[365] in case1-3 : exist? but could not create.");
    } else {

      if(D1a > D1b){

        //Math.PI + P3Angle(P3b,P1a,P3a)
        Q3b = Math.PI + P3Angle(
          P3b,
          movePoint(P1a, -normalVector(P1a, P0a), D0a1a),
          movePoint(P3a, R3a, D0a3a)
          );

      } else if(D1a < D1b) {
        if(P3Angle(P1b, P1a, P0a) >= P3Angle(P1b, P1a, P2a1)){
          Q2b = Math.PI + P3Angle(
            P2b,
            movePoint(P2a, -R2a, D0a2a),
            movePoint(P3a, R3a, D0a3a)
            );
        }
      }
    }
  }

}

/* ■ set up6 : Point Distance of Point[1a] and Point[2a] and Point[3a] ■ */

//PointDistance(P2a1,P3a)
D2a3a = Math.Sqrt(
  Math.Pow(D2a + D2b * Math.Cos(Q2b - Math.PI), 2)
  + Math.Pow(D2b * Math.Sin(Q2b - Math.PI), 2)
  );

//PointDistance(P1a,P3a)
D1a3a = Math.Sqrt(
  Math.Pow(D3b + D3a * Math.Cos(Q3b - Math.PI), 2)
  + Math.Pow(D3a * Math.Sin(Q3b - Math.PI), 2)
  );

/* ■ set up7 : Point[P3a1] and Point[P3a2] ■ */

if(Q1b == Math.PI){

  P3a1 = P3a;
  P3a2 = PLPMirror(
    P3a1,
    P1a,
    normalizeVector(CrossProduct(R1a, normalVector(P1a, P2a1)))
    );

} else if(Q1b == 0 || Q1b == Math.PI * 2){

  if(Case == 1){
    if(Case1 == 4){
```





```
      P3a1 = CCIntersectionPlus(P1a, P2a1, (D3b - D3a), D2, normalVector(P1a, P3a));
      P3a2 = CCIntersectionMinus(P1a, P2a1, (D3b - D3a), D2, normalVector(P1a, P3a));
    } else if(Case1 == 5){
      P3a1 = CCIntersectionPlus(P1a, P2a1, D3, (D2a - D2b), normalVector(P1a, P3a));
      P3a2 = CCIntersectionMinus(P1a, P2a1, D3, (D2a - D2b), normalVector(P1a, P3a));
    }
  } else {
    P3a1 = P1a;
    P3a2 = P1a;
  }

} else if(Q1b < Math.PI){

  //PLPLDIntersectionPlus(movePoint[P1a],R1a,movePoint[P2a],R2a,ABCHeight[P3a,P0a,P1a])
  P3a1 = PLPLDIntersectionPlus(
    movePoint(P1a, normalVector(P1a, P0a), D1a3a * ABCCos(D0a3a, D0a1a, D1a3a)),
    R1a,
    movePoint(P2a1, normalVector(P2a1, P0a), D2a3a * ABCCos(D0a3a, D0a2a, D2a3a)),
    R2a1,
    ABCHeight(D0a1a, D0a3a, D1a3a)
    );
  P3a2 = PLPMirror(
    P3a1,
    P1a,
    normalizeVector(CrossProduct(normalVector(P0a, P1a), normalVector(P0a, P2a1)))
    );

  if(P3Angle(P1b, P1a, P2a1) <= Math.Acos(ABCCos(D6, D4, D5))){
    if(PointDistance(P3a1, P2b) < PointDistance(P3a2, P2b)){
      Point3d TempPoint = P3a1;
      P3a1 = P3a2;
      P3a2 = TempPoint;
    }
  }

} else if(Q1b > Math.PI){

  //PLPLDIntersectionPlus(movePoint[P1a],R1a,movePoint[P2a],R2a,ABCHeight[P3a,P0a,P1a])
  P3a1 = PLPLDIntersectionMinus(
    movePoint(P1a, normalVector(P1a, P0a), D1a3a * ABCCos(D0a3a, D0a1a, D1a3a)),
    R1a,
    movePoint(P2a1, normalVector(P2a1, P0a), D2a3a * ABCCos(D0a3a, D0a2a, D2a3a)),
    R2a1,
    ABCHeight(D0a1a, D0a3a, D1a3a)
    );
  P3a2 = PLPMirror(
    P3a1,
    P1a,
    normalizeVector(CrossProduct(normalVector(P0a, P1a), normalVector(P0a, P2a1)))
    );

  if(PointDistance(P3a1, P2b) > PointDistance(P3a2, P2b)){
    Point3d TempPoint = P3a1;
    P3a1 = P3a2;
    P3a2 = TempPoint;
  }
}

if(Case == 1){
  if(PointDistance(P3a1, P3a) > PointDistance(P3a2, P3a)){
    Point3d TempPoint = P3a1;
    P3a1 = P3a2;
    P3a2 = TempPoint;
  }
}

if(Range >= 0.25 && Range < 0.75){
  Point3d TempPoint = P3a1;
  P3a1 = P3a2;
  P3a2 = TempPoint;
}

/* ■ set up8 : Point[P3b1] and Point[P3b2] and Point[P2b1] and Point[P2b2] ■ */

if(Q1b == Math.PI){

  P2b1 = P2b;
  P3b1 = P3b;
  P2b2 = PLPMirror(
    P2b1,
    P1a,
    normalizeVector(CrossProduct(R1a, normalVector(P1a, P2a1)))
    );
  P3b2 = PLPMirror(
    P3b1,
    P1a,
    normalizeVector(CrossProduct(R1a, normalVector(P1a, P2a1)))
    );

  if(Range > 0.25 && Range < 0.75){
    Point3d TempPoint = P2b1;
    P2b1 = P2b2;
    P2b2 = TempPoint;
    TempPoint = P3b1;
    P3b1 = P3b2;
    P3b2 = TempPoint;
  }

} else if(Q1b == 0 || Q1b == Math.PI * 2){

  if(Case == 1){
    if(Case1 == 4 || Case1 == 5){
      P2b1 = movePoint(P2a1, normalVector(P2a1, P3a1), D2a);
      P2b2 = movePoint(P2a1, normalVector(P2a1, P3a2), D2a);
      P3b1 = movePoint(P1a, normalVector(P1a, P3a1), D3b);
      P3b2 = movePoint(P1a, normalVector(P1a, P3a2), D3b);
    }
  } else {
    P2b1 = movePoint(P1a, normalVector(P1a, P1b), D2b);
    P2b2 = movePoint(P1a, normalVector(P1a, P1b), D2b);
    P3b1 = movePoint(P1a, normalVector(P1a, P1b), D3b);
    P3b2 = movePoint(P1a, normalVector(P1a, P1b), D3b);
  }

} else {
  if(Case == 1){

    //PPLPLclosestPoint(P1a,R1a,P3a1,R3a1)
    P3b1 = PPLPLclosestPoint(P1a, R1a, P3a1, normalVector(P3a1, P0a));

    //PPLPLclosestPoint(P1a,R1a,P3a2,R3a2)
    P3b2 = PPLPLclosestPoint(P1a, R1a, P3a2, normalVector(P3a2, P0a));

    //PPLPLclosestPoint(P2a,R2a,P3a1,R3a1)
    P2b1 = PPLPLclosestPoint(P2a1, R2a1, P3a1, normalVector(P3a1, P0a));

    //PPLPLclosestPoint(P2a,R2a,P3a2,R3a2)
    P2b2 = PPLPLclosestPoint(P2a1, R2a1, P3a2, normalVector(P3a2, P0a));

  } else {

    if(D3b <
      PointDistance(P1a, PPLPLclosestPoint(P1a, R1a, P3a1, normalVector(P3a1, P0a)))
    ){
      if(Case == 3){
        Print(" ■■ error[575] in case3 : maybe not exist... ");
        P3b1.Z = Math.Sqrt(-1); //null data
        P3b2.Z = Math.Sqrt(-1); //null data
        P2b1.Z = Math.Sqrt(-1); //null data
        P2b2.Z = Math.Sqrt(-1); //null data
      }
    } else {
      if(Case != 1){

        if(Q1b < Math.PI){

          //PLPLDIntersectionPlus(P1a,R1a,P3a1,R3a1,D3b)
          P3b1 = PLPLDIntersectionPlus(P1a, R1a, P3a1, normalVector(P3a1, P0a), D3b);

          //PLPLDIntersectionPlus(P1a,R1a,P3a2,R3a2,D3b)
          P3b2 = PLPLDIntersectionPlus(P1a, R1a, P3a2, normalVector(P3a2, P0a), D3b);

          //PLPLDIntersectionMinus(P2a,R2a,P3a1,R3a1,D2a)
          P2b1 = PLPLDIntersectionMinus(P2a1, R2a1, P3a1, normalVector(P3a1, P0a), D2a);

          //PLPLDIntersectionMinus(P2a,R2a,P3a2,R3a2,D2a)
          P2b2 = PLPLDIntersectionMinus(P2a1, R2a1, P3a2, normalVector(P3a2, P0a), D2a);
        } else if(Q1b > Math.PI){

          //PLPLDIntersectionPlus(P1a,R1a,P3a1,R3a1,D3b)
          P3b1 = PLPLDIntersectionPlus(P1a, R1a, P3a1, normalVector(P3a1, P0a), D3b);

          //PLPLDIntersectionPlus(P1a,R1a,P3a2,R3a2,D3b)
          P3b2 = PLPLDIntersectionPlus(P1a, R1a, P3a2, normalVector(P3a2, P0a), D3b);

          //PLPLDIntersectionMinus(P2a,R2a,P3a1,R3a1,D2a)
          P2b1 = PLPLDIntersectionMinus(P2a1, R2a1, P3a1, normalVector(P3a1, P0a), D2a);

          //PLPLDIntersectionMinus(P2a,R2a,P3a2,R3a2,D2a)
          P2b2 = PLPLDIntersectionMinus(P2a1, R2a1, P3a2, normalVector(P3a2, P0a), D2a);

        }
      }
    }
  }
}

/* ■ set upX : Q1a and Q2a and Q3a ■ */

if(Case == 1){

  if(Math.Round(Range, 5) < 0.25 || Math.Round(Range, 5) >= 0.75){
    Q1a1 = P3Angle(P1a, P1b, P3b1);
    Q2a1 = P3Angle(P2a1, P2b1, P1b);
    Q3a1 = P3Angle(P3a1, P3b1, P2b1);
    Q1a2 = Math.PI * 2 - P3Angle(P1a, P1b, P3b2);
    Q2a2 = Math.PI * 2 - P3Angle(P2a1, P2b2, P1b);
    Q3a2 = Math.PI * 2 - P3Angle(P3a2, P3b2, P2b2);
  } else {
    Q1a1 = Math.PI * 2 - P3Angle(P1a, P1b, P3b1);
    Q2a1 = Math.PI * 2 - P3Angle(P2a1, P2b1, P1b);
    Q3a1 = Math.PI * 2 - P3Angle(P3a1, P3b1, P2b1);
    Q1a2 = P3Angle(P1a, P1b, P3b2);
    Q2a2 = P3Angle(P2a1, P2b2, P1b);
    Q3a2 = P3Angle(P3a2, P3b2, P2b2);
  }

} else {

  if(Range < 0.25){

    if(PointOverPL(P0a, P1a, ABCnormalVector(P1a, P1b, P3b1)) == true){
      Q1a1 = -P3Angle(P1a, P1b, P3b1);
    } else {
      Q1a1 = P3Angle(P1a, P1b, P3b1);
    }
    if(PointOverPL(P0a, P2a1, ABCnormalVector(P2a1, P2b1, P1b)) == true){
      Q2a1 = -P3Angle(P2a1, P2b1, P1b);
    } else {
      Q2a1 = P3Angle(P2a1, P2b1, P1b);
    }
    if(PointOverPL(P0a, P3a1, ABCnormalVector(P3a1, P3b1, P2b1)) == true){
      Q3a1 = -P3Angle(P3a1, P3b1, P2b1);
    } else {
      Q3a1 = P3Angle(P3a1, P3b1, P2b1);
    }

    if(PointOverPL(P1a, P1b, ABCnormalVector(P1b, P2b2, P3b2)) == true){
      Q1a2 = Math.PI * 2 - P3Angle(P1a, P1b, P3b2);
    } else {
      Q1a2 = P3Angle(P1a, P1b, P3b2);
    }
    if(PointOverPL(P2a1, P1b, ABCnormalVector(P1b, P2b2, P3b2)) == true){
      Q2a2 = Math.PI * 2 - P3Angle(P2a1, P2b2, P1b);
    } else {
      Q2a2 = P3Angle(P2a1, P2b2, P1b);
    }
    if(PointOverPL(P3a2, P1b, ABCnormalVector(P1b, P2b2, P3b2)) == true){
      Q3a2 = Math.PI * 2 - P3Angle(P3a2, P3b2, P2b2);
    } else {
      Q3a2 = P3Angle(P3a2, P3b2, P2b2);
    }

  } else if(Range >= 0.25 && Range < 0.5){

    if(PointOverPL(P1a, P1b, ABCnormalVector(P1b, P2b1, P3b1)) == true){
      Q1a1 = Math.PI * 2 - P3Angle(P1a, P1b, P3b1);
    } else {
      Q1a1 = P3Angle(P1a, P1b, P3b1);
    }
    if(PointOverPL(P2a1, P1b, ABCnormalVector(P1b, P2b1, P3b1)) == true){
      Q2a1 = Math.PI * 2 - P3Angle(P2a1, P2b1, P1b);
    } else {
      Q2a1 = P3Angle(P2a1, P2b1, P1b);
    }
    if(PointOverPL(P3a1, P1b, ABCnormalVector(P1b, P2b1, P3b1)) == true){
      Q3a1 = Math.PI * 2 - P3Angle(P3a1, P3b1, P2b1);
    } else {
      Q3a1 = P3Angle(P3a1, P3b1, P2b1);
    }

    if(PointOverPL(P0a, P1a, ABCnormalVector(P1a, P1b, P3b2)) == true){
      Q1a2 = -P3Angle(P1a, P1b, P3b2);
    } else {
      Q1a2 = P3Angle(P1a, P1b, P3b2);
    }
```

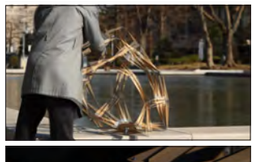
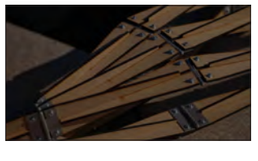










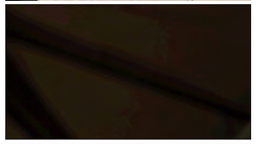




```
    if(PointOverPL(P0a, P2a1, ABCnormalVector(P2a1, P2b2, P1b)) == true){
      Q2a2 = -P3Angle(P2a1, P2b2, P1b);
    } else {
      Q2a2 = P3Angle(P2a1, P2b2, P1b);
    }
    if(PointOverPL(P0a, P3a2, ABCnormalVector(P3a2, P3b2, P2b2)) == true){
      Q3a2 = -P3Angle(P3a2, P3b2, P2b2);
    } else {
      Q3a2 = P3Angle(P3a2, P3b2, P2b2);
    }

  } else if(Range >= 0.5 && Range < 0.75){

    if(PointOverPL(P0a, P1a, ABCnormalVector(P1a, P1b, P3b1)) == true){
      Q1a1 = Math.PI * 2 + P3Angle(P1a, P1b, P3b1);
    } else {
      Q1a1 = Math.PI * 2 - P3Angle(P1a, P1b, P3b1);
    }
    if(PointOverPL(P0a, P2a1, ABCnormalVector(P2a1, P2b1, P1b)) == true){
      Q2a1 = Math.PI * 2 + P3Angle(P2a1, P2b1, P1b);
    } else {
      Q2a1 = Math.PI * 2 - P3Angle(P2a1, P2b1, P1b);
    }
    if(PointOverPL(P0a, P3a1, ABCnormalVector(P3a1, P3b1, P2b1)) == true){
      Q3a1 = Math.PI * 2 + P3Angle(P3a1, P3b1, P2b1);
    } else {
      Q3a1 = Math.PI * 2 - P3Angle(P3a1, P3b1, P2b1);
    }

    if(PointOverPL(P1a, P1b, ABCnormalVector(P1b, P2b2, P3b2)) == true){
      Q1a2 = P3Angle(P1a, P1b, P3b2);
    } else {
      Q1a2 = Math.PI * 2 - P3Angle(P1a, P1b, P3b2);
    }
    if(PointOverPL(P2a1, P1b, ABCnormalVector(P1b, P2b2, P3b2)) == true){
      Q2a2 = P3Angle(P2a1, P2b2, P1b);
    } else {
      Q2a2 = Math.PI * 2 - P3Angle(P2a1, P2b2, P1b);
    }
    if(PointOverPL(P3a2, P1b, ABCnormalVector(P1b, P2b2, P3b2)) == true){
      Q3a2 = P3Angle(P3a2, P3b2, P2b2);
    } else {
      Q3a2 = Math.PI * 2 - P3Angle(P3a2, P3b2, P2b2);
    }

  } else if(Range >= 0.75){

    if(PointOverPL(P0a, P1a, ABCnormalVector(P1a, P1b, P3b2)) == true){
      Q1a2 = Math.PI * 2 + P3Angle(P1a, P1b, P3b2);
    } else {
      Q1a2 = Math.PI * 2 - P3Angle(P1a, P1b, P3b2);
    }
    if(PointOverPL(P0a, P2a1, ABCnormalVector(P2a1, P2b2, P1b)) == true){
      Q2a2 = Math.PI * 2 + P3Angle(P2a1, P2b2, P1b);
    } else {
      Q2a2 = Math.PI * 2 - P3Angle(P2a1, P2b2, P1b);
    }
    if(PointOverPL(P0a, P3a2, ABCnormalVector(P3a2, P3b2, P2b2)) == true){
      Q3a2 = Math.PI * 2 + P3Angle(P3a2, P3b2, P2b2);
    } else {
      Q3a2 = Math.PI * 2 - P3Angle(P3a2, P3b2, P2b2);
    }

    if(PointOverPL(P1a, P1b, ABCnormalVector(P1b, P2b1, P3b1)) == true){
      Q1a1 = P3Angle(P1a, P1b, P3b1);
    } else {
      Q1a1 = Math.PI * 2 - P3Angle(P1a, P1b, P3b1);
    }
    if(PointOverPL(P2a1, P1b, ABCnormalVector(P1b, P2b1, P3b1)) == true){
      Q2a1 = P3Angle(P2a1, P2b1, P1b);
    } else {
      Q2a1 = Math.PI * 2 - P3Angle(P2a1, P2b1, P1b);
    }
    if(PointOverPL(P3a1, P1b, ABCnormalVector(P1b, P2b1, P3b1)) == true){
      Q3a1 = P3Angle(P3a1, P3b1, P2b1);
    } else {
      Q3a1 = Math.PI * 2 - P3Angle(P3a1, P3b1, P2b1);
    }
  }
}

/* ■ set upX : R3a ■ */

if(Case == 1){
  if(Math.Round(Range, 5) < 0.25 || Math.Round(Range, 5) >= 0.75){
    R3a1 = ABCnormalVector(P3a1, P3b1, P2b1);
    R3a2 = ABCnormalVector(P3a2, P2b2, P3b2);
  } else {
    R3a1 = -ABCnormalVector(P3a1, P3b1, P2b1);
    R3a2 = -ABCnormalVector(P3a2, P2b2, P3b2);
  }
} else {
  if(Q1b == Math.PI){
    if(Range == 0 || Math.Round(Range, 5) == 1.0){
      R3a1 = ABCnormalVector(P3a1, P3b1, P2b1);
      R3a2 = ABCnormalVector(P3a2, P2b2, P3b2);
    } else {
      R3a1 = -ABCnormalVector(P3a1, P3b1, P2b1);
      R3a2 = -ABCnormalVector(P3a2, P2b2, P3b2);
    }
  } else if(Q1b == 0 || Q1b == Math.PI * 2){
    R3a1 = ABCnormalVector(P3a1, P3b1, P2b1);
    R3a2 = ABCnormalVector(P3a2, P2b2, P3b2);
    R3a1 = rotateNVector(R1a, normalVector(P1a, P1b), V2Angle(R3a, R3b) * 2);
    R3a2 = rotateNVector(R1a, normalVector(P1a, P1b), V2Angle(R3a, R3b) * 2);
  } else if(Q1b < Math.PI){
    R3a1 = normalVector(P3a1, P0a);
    R3a2 = normalVector(P3a2, P0a);
  } else {
    R3a2 = -normalVector(P3a2, P0a);
    R3a1 = -normalVector(P3a1, P0a);
  }
}

/* ■ set upX : R2b and R3b ■ */

if(Q1b == 0){
  if(Case == 1){
    R2b1 = -CrossProduct(normalVector(P2b1, P2a1), R2a);
    R2b2 = -CrossProduct(normalVector(P2b2, P2a1), R2a);
    R3b1 = -CrossProduct(normalVector(P3b1, P1a), R1a);
    R3b2 = -CrossProduct(normalVector(P3b2, P1a), R1a);
  } else {
    R2b1 = rotateNVector(R1b, normalVector(P1a, P1b), V2Angle(R2b, R2a));
    R2b2 = rotateNVector(R1b, normalVector(P1a, P1b), V2Angle(R2b, R2a));
    R3b1 = rotateNVector(R1a, normalVector(P1a, P1b), V2Angle(R3b, R1a));
    R3b2 = rotateNVector(R1a, normalVector(P1a, P1b), V2Angle(R3b, R1a));
  }
} else if(Q1b == Math.PI * 2){
  if(Case == 1){
    R2b1 = -normalVector(P2b1, P0b1);
    R2b2 = normalVector(P2b2, P0b2);
    R3b1 = -normalVector(P3b1, P0b1);
    R3b2 = normalVector(P3b2, P0b2);
    R2b1 = -CrossProduct(normalVector(P2b1, P2a1), R2a);
    R2b2 = -CrossProduct(normalVector(P2b2, P2a1), R2a);
    R3b1 = -CrossProduct(normalVector(P3b1, P1a), R1a);
    R3b2 = -CrossProduct(normalVector(P3b2, P1a), R1a);
  } else {
    R2b1 = rotateNVector(R1b, normalVector(P1a, P1b), V2Angle(R2b, R2a));
    R2b2 = rotateNVector(R1b, normalVector(P1a, P1b), V2Angle(R2b, R2a));
    R3b1 = rotateNVector(R1a, normalVector(P1a, P1b), V2Angle(R3b, R1a));
    R3b2 = rotateNVector(R1a, normalVector(P1a, P1b), V2Angle(R3b, R1a));
  }
} else if(Q1b == Math.PI){
  if(Case == 1){
    R2b1 = normalizeVector(CrossProduct(R2a1, normalVector(P2a1, P2b1)));
    R2b2 = normalizeVector(CrossProduct(R2a1, normalVector(P2a1, P2b2)));
    R3b1 = normalizeVector(CrossProduct(R3a1, normalVector(P3a1, P3b1)));
    R3b2 = normalizeVector(CrossProduct(R3a2, normalVector(P3a2, P3b2)));
  } else {
    if(Range == 0 || Math.Round(Range, 5) == 1.0){
      R2b1 = normalVector(P2b1, P0b);
      R2b2 = -normalVector(P2b2, PLPMirror(P0b, P1b, normalVector(P1b, P0b)));
      R3b1 = normalVector(P3b1, P0b);
      R3b2 = -normalVector(P3b2, PLPMirror(P0b, P1b, normalVector(P1b, P0b)));
    } else {
      R2b1 = -normalVector(P2b1, PLPMirror(P0b, P1b, normalVector(P1b, P0b)));
      R2b2 = normalVector(P2b2, P0b);
      R3b1 = -normalVector(P3b1, PLPMirror(P0b, P1b, normalVector(P1b, P0b)));
      R3b2 = normalVector(P3b2, P0b);
    }
  }
} else if(Q1b > Math.PI){
  R2b1 = ABCnormalVector(P2b1, P3a1, P2a1);
  R2b2 = ABCnormalVector(P2b2, P3a2, P2a1);
  R3b1 = ABCnormalVector(P3b1, P1a, P3a1);
  R3b2 = ABCnormalVector(P3b2, P1a, P3a2);
} else {
  R2b1 = -ABCnormalVector(P2b1, P3a1, P2a1);
  R2b2 = -ABCnormalVector(P2b2, P3a2, P2a1);
  R3b1 = -ABCnormalVector(P3b1, P1a, P3a1);
  R3b2 = -ABCnormalVector(P3b2, P1a, P3a2);
}

/* ■ set upX : Point[P0b1] and Point[P0b2] ■ */

if(Q1b == Math.PI){
  if(Case == 3){
    Print(" ■■ error[603] in case3 : P0b2 is not Corrent Data");
  }
  P0b1 = P0b;
  P0b2 = PLPMirror(P0b1, P1b, normalVector(P1b, P0b1));
  if(Range > 0.25 && Range < 0.75){
    Point3d TempPoint = P0b1;
    P0b1 = P0b2;
    P0b2 = TempPoint;
  }

} else if(Q1b == minAngle || Q1b == maxAngle){

  if(Case == 1){
    if(Case1 <= 3 || Case1 == 6){
      P0b1 = InfPoint;
      P0b2 = InfPoint;
      if(Case1 == 4 || Case1 == 5){
      //LLIntersection(P3b1,R3b1,P1b,R1b)
      P0b1 = LLIntersection(
        P3b1,
        CrossProduct(normalVector(P3b1, P1a), R1a),
        P1b,
        R1b
        );

      //LLIntersection(P3b2,R3b2,P1b,R1b)
      P0b2 = LLIntersection(
        P3b2,
        CrossProduct(normalVector(P3b2, P1a), R1a),
        P1b,
        R1b
        );
      }
    } else {
      P0b1 = InfPoint;
      P0b2 = InfPoint;
    }

  } else {

    if(Case == 1){
      //LLIntersection(P3b1,R3b1,P1b,R1b)
      P0b1 = LLIntersection(
        P3b1,
        normalizeVector(
        CrossProduct(normalVector(P3b1, P1a), normalVector(P3b1, P3a1))),
        P1b,
        R1b
        );

      //LLIntersection(P3b2,R3b2,P1b,R1b)
      P0b2 = LLIntersection(
        P3b2,
        normalizeVector(
        CrossProduct(normalVector(P3b2, P1a), normalVector(P3b2, P3a2))),
        P1b,
        R1b
        );
    } else {

      P0b1 = L3closestPoint(P1b, R1b, P2b1, R2b1, P3b1, R3b1);
      P0b2 = L3closestPoint(P1b, R1b, P2b2, R2b2, P3b2, R3b2);

    }
  }
}

/* ■ from Radian to Degree ■ */

Q1b = Q1b / (Math.PI / 180);
Q2b = Q2b / (Math.PI / 180);
Q3b = Q3b / (Math.PI / 180);
Q1a1 = Q1a1 / (Math.PI / 180);
Q2a1 = Q2a1 / (Math.PI / 180);
Q3a1 = Q3a1 / (Math.PI / 180);
Q1a2 = Q1a2 / (Math.PI / 180);
Q2a2 = Q2a2 / (Math.PI / 180);
Q3a2 = Q3a2 / (Math.PI / 180);
```

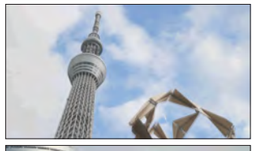
```
/* ■■■■■■■■■■ MAKE KALEIDOCYCLE LIST ■■■■■■■■■■ */

/* ■ set up0 : make List from Item ■ */

List<Point3d> KPointN = new List<Point3d>();
List<Point3d> KPointM = new List<Point3d>();
List<Vector3d> KVectorN = new List<Vector3d>();
List<Vector3d> KVectorM = new List<Vector3d>();
List<double> Qb = new List<double>();
List<double> Qa = new List<double>();
List<double> Dist = new List<double>();
List<double> DistPre = new List<double>();

/* ■ set up1 : make List Data ■ */

KPointN.Add(P0a); KPointN.Add(P0b1); KPointN.Add(P1a); KPointN.Add(P1b);
KPointN.Add(P2a1); KPointN.Add(P2b1); KPointN.Add(P3a1); KPointN.Add(P3b1);

KPointM.Add(P0a); KPointM.Add(P0b2); KPointM.Add(P1a); KPointM.Add(P1b);
KPointM.Add(P2a1); KPointM.Add(P2b2); KPointM.Add(P3a2); KPointM.Add(P3b2);

KVectorN.Add(R1a); KVectorN.Add(R1b); KVectorN.Add(R2a1);
KVectorN.Add(R2b1); KVectorN.Add(R3a1); KVectorN.Add(R3b1);

KVectorM.Add(R1a); KVectorM.Add(R1b); KVectorM.Add(R2a1);
KVectorM.Add(R2b2); KVectorM.Add(R3a2); KVectorM.Add(R3b2);

Qb.Add(Q1b); Qb.Add(Q2b); Qb.Add(Q3b);
Qa.Add(Q1a1); Qa.Add(Q2a1); Qa.Add(Q3a1);
Qa.Add(Q1a2); Qa.Add(Q2a2); Qa.Add(Q3a2);

for(int i = 2; i < KPointN.Count - 1; i++){
  Dist.Add(KPointN[i].DistanceTo(KPointN[i + 1]));
}
Dist.Add(KPointN[KPointN.Count - 1].DistanceTo(KPointN[2]));

/* ■■■■■■■■■■ ROTATE KALEIDOCYCLE POINT ■■■■■■■■■■ */

if(onPlane == true){
  double rotN2 = new double();
  double rotN3 = new double();
  Point3d cenP0 = new Point3d();
  Point3d cenP1 = new Point3d();
  Vector3d cenV0 = new Vector3d();
  Point3d rotP2 = new Point3d();
  Point3d rotP3 = new Point3d();
  Vector3d rotV3 = new Vector3d();
  Vector3d swiV3 = new Vector3d();
  Vector3d swiV4 = new Vector3d();

  cenP0 = innercenterPoint(P1a, P2a, P3a);
  cenV0 = normalizeVector(
    CrossProduct(normalVector(P1a, P2a), normalVector(P1a, P3a))
    );

  if(Case == 1 && Q1b == Math.PI){
    rotN2 = 0;
  } else {
    if(Range <= 0.5){
      rotN2 = -P3Angle(KPointN[2], KPointN[3], KPointN[4]);
    } else {
      rotN2 = P3Angle(KPointN[2], KPointN[3], KPointN[4]);
    }
  }

  for(int i = 0; i < KPointN.Count; i++){
    KPointN[i] = rotatePoint(KPointN[i], KPointN[2], KVectorN[1], rotN2);
  }
  for(int i = 0; i < KVectorN.Count; i++){
    KVectorN[i] = rotateNVector(KVectorN[i], KVectorN[1], rotN2);
  }

  rotP2 = closestPoint(KPointN[2], KPointN[4], KPointN[6]);
  rotP3 = PLPclosestPoint(KPointN[6], KPointN[2], cenV0);
  rotV3 = normalVector(P1a, P2a);
  swiV3 = addVector(
    cenV0, normalizeVector(
    CrossProduct(normalVector(rotP2, rotP3), normalVector(rotP2, KPointN[2]))
    )
    );
  swiV4 = addVector(
    rotV3, normalizeVector(
    CrossProduct(normalVector(rotP3, rotP2), normalVector(rotP3, KPointN[6]))
    )
    );

  if(Case == 1){

    if(Range == 0.0 || Range == 1.0){
      rotN3 = 0;
    } else if(Range == 0.5){
      rotN3 = Math.PI;
    } else if(Range < 0.25){
      rotN3 = P3Angle(rotP2, rotP3, KPointN[6]);
    } else if(Range >= 0.25 && Range < 0.5){
      rotN3 = Math.PI - P3Angle(rotP2, rotP3, KPointN[6]);
    } else if(Range > 0.5 && Range < 0.75){
      rotN3 = Math.PI + P3Angle(rotP2, rotP3, KPointN[6]);
    } else {
      rotN3 = Math.PI * 2 - P3Angle(rotP2, rotP3, KPointN[6]);
    }

  } else {
    if(Range == 0.0){
      rotN3 = 0;
    } else if(Range == 0.5){
      rotN3 = Math.PI;
    } else if(Range == 1.0){
      rotN3 = Math.PI * 2;
    } else if(Q1b == 0 || Q1b == Math.PI * 2){
      rotN3 = infinity;
    } else {
      if(Range <= 0.5){
        if(swiV3.Length >= 1){
          if(swiV4.Length >= 1){
            rotN3 = P3Angle(rotP2, rotP3, KPointN[6]);
          } else {
            rotN3 = -P3Angle(rotP2, rotP3, KPointN[6]);
          }
        } else {
          rotN3 = Math.PI - P3Angle(rotP2, rotP3, KPointN[6]);
        }
      } else {
        if(swiV3.Length >= 1){
          rotN3 = Math.PI * 2 - P3Angle(rotP2, rotP3, KPointN[6]);
        } else {
          rotN3 = Math.PI + P3Angle(rotP2, rotP3, KPointN[6]);
        }
      }
    }
  }

  for(int i = 0; i < KPointN.Count; i++){
    KPointN[i] = rotatePoint(KPointN[i], KPointN[2], rotV3, rotN3);
  }
  for(int i = 0; i < KVectorN.Count; i++){
    KVectorN[i] = rotateNVector(KVectorN[i], rotV3, rotN3);
  }

  cenP1 = innercenterPoint(KPointN[2], KPointN[4], KPointN[6]);

  if(Range != 0 && Range != 1.0){
    for(int i = 0; i < KPointN.Count; i++){
      KPointN[i] = movePoint(
        KPointN[i],
        normalVector(cenP1, cenP0),
        PointDistance(cenP1, cenP0)
        );
    }
  }
}

/* ■■■■■■■■■■ MAKE KALEIDOCYCLE LINE ■■■■■■■■■■ */

List<Polyline> KLine = new List<Polyline>();
List<List<Line>> KLineA = new List<List<Line>>();
List<List<Line>> KLineB = new List<List<Line>>();

KLine.Add(new Polyline());
KLine.Add(new Polyline());
KLineA.Add(new List<Line>());
KLineA.Add(new List<Line>());
KLineB.Add(new List<Line>());
KLineB.Add(new List<Line>());

for(int i = 2; i < KPointN.Count; i++){
  KLine[0].Add(KPointN[i]);
  KLine[1].Add(KPointM[i]);
}
KLine[0].Add(KPointN[2]);
KLine[1].Add(KPointM[2]);

KLineA[0].Add(new Line(KPointN[0], KPointN[2]));
KLineA[0].Add(new Line(KPointN[0], KPointN[4]));
KLineA[0].Add(new Line(KPointN[0], KPointN[6]));
KLineA[1].Add(new Line(KPointM[0], KPointM[2]));
KLineA[1].Add(new Line(KPointM[0], KPointM[4]));
KLineA[1].Add(new Line(KPointM[0], KPointM[6]));
KLineB[0].Add(new Line(KPointN[1], KPointN[3]));
KLineB[0].Add(new Line(KPointN[1], KPointN[5]));
KLineB[0].Add(new Line(KPointN[1], KPointN[7]));
KLineB[1].Add(new Line(KPointM[1], KPointM[3]));
KLineB[1].Add(new Line(KPointM[1], KPointM[5]));
KLineB[1].Add(new Line(KPointM[1], KPointM[7]));

/* ■■■■■■■■■■ MAKE KALEIDOCYCLE GEOMETRY ■■■■■■■■■■ */

KaleidocycleGeometry KGN = new KaleidocycleGeometry();
KGN.setupKaleido(KPointN, KVectorN, LA, MA, LB, MB);
KaleidocycleGeometry KGM = new KaleidocycleGeometry();
KGM.setupKaleido(KPointM, KVectorM, LA, MA, LB, MB);

/* ■■■■■■■■■■ OUT DATA ■■■■■■■■■■ */

if(Thick == true){
  Kaleidocycle = KGN.Lines;
  KaleidocycleMir = KGM.Lines;
} else {
  Kaleidocycle = KLine[0];
  KaleidocycleMir = KLine[1];
}
KaleidoPoint = KPointN;
KaleidoPointMir = KPointM;
KaleidoVector = KVectorN;
KaleidoVectorMir = KVectorM;
KaleidoLineA = KLineA[0];
KaleidoLineAMir = KLineA[1];
KaleidoLineB = KLineB[0];
KaleidoLineBMir = KLineB[1];
KaleidoAngleA = Qa;
KaleidoAngleB = Qb;
KaleidoLength = Dist;

 /* ■■■■■■■■■■ END OF MAIN CODE LINE ■■■■■■■■■■ */
}

//<Custom additional code>
/* ■■■■■■■■■■ USING DATA ■■■■■■■■■■ */

/* ■ set up X : InfinityPoint ■ */

public static double infinity = double.PositiveInfinity;
public static Point3d InfPoint = new Point3d(infinity, infinity, infinity);
public static Point3d nullPoint = new Point3d(0, 0, 0);
public static Vector3d InfVector = new Vector3d();

/* ■■■■■■■■■■ METHOD ■■■■■■■■■■ */
//MiddlePoint of Point[A] and Point[B]
Point3d ABMiddlePoint(Point3d A, Point3d B){
  Point3d P = new Point3d();
  P.X = (A.X + B.X) / 2;
  P.Y = (A.Y + B.Y) / 2;
  P.Z = (A.Z + B.Z) / 2;
  return P;
}

Point3d ABmovePoint(Point3d A, Point3d B, double DA, double DB){
  Point3d P = new Point3d();
  Vector3d V = new Vector3d();
  double D = new double();
  V = normalVector(A, B);
  D = PointDistance(A, B);
  P = movePoint(A, V, DA / (DA + DB) * D);
  return P;
}

//closest Point of Line[A,B] and Point[P]
Point3d closestPoint(Point3d A, Point3d B, Point3d P){
```

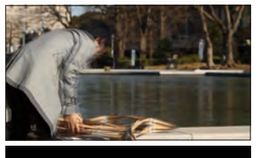
```
  Point3d CP = new Point3d();
  double k;

  k =
    ((P.X − A.X) * (B.X − A.X)
    + (P.Y − A.Y) * (B.Y − A.Y)
    + (P.Z − A.Z) * (B.Z − A.Z))
    / ((B.X − A.X) * (B.X − A.X)
    + (B.Y − A.Y) * (B.Y − A.Y)
    + (B.Z − A.Z) * (B.Z − A.Z));
  CP.X = k * B.X + (1 − k) * A.X;
  CP.Y = k * B.Y + (1 − k) * A.Y;
  CP.Z = k * B.Z + (1 − k) * A.Z;

  return CP;

}

//normalizeVector of Vector[A]
public static Vector3d normalizeVector(Vector3d A){
  Vector3d vec = new Vector3d();
  double s;

  s = 1 / Math.Sqrt(A.X * A.X + A.Y * A.Y + A.Z * A.Z);
  vec.X = A.X * s;
  vec.Y = A.Y * s;
  vec.Z = A.Z * s;

  return vec;
}

//unit Vector of Point[A] to Point[B]
public static Vector3d normalVector(Point3d A, Point3d B){
  Vector3d vec = new Vector3d();

  vec.X = B.X − A.X;
  vec.Y = B.Y − A.Y;
  vec.Z = B.Z − A.Z;
  vec = normalizeVector(vec);

  return vec;
}

//Vector of Point[A] to Point[B]
public static Vector3d ABVector(Point3d A, Point3d B){
  Vector3d vec = new Vector3d();
  vec.X = B.X − A.X;
  vec.Y = B.Y − A.Y;
  vec.Z = B.Z − A.Z;
  return vec;
}

//Vector of Vector[A] and Vector[B]
public static Vector3d addVector(Vector3d A, Vector3d B){
  Vector3d vec = new Vector3d();
  vec.X = B.X + A.X;
  vec.Y = B.Y + A.Y;
  vec.Z = B.Z + A.Z;
  return vec;
}

//normal unit Vector of Plane[A,B,C]
public static Vector3d ABCnormalVector(Point3d A, Point3d B, Point3d C){
  Vector3d V = new Vector3d();
  Vector3d VAB = new Vector3d();
  Vector3d VAC = new Vector3d();

  VAB = normalVector(A, B);
  VAC = normalVector(A, C);
  V = normalizeVector(CrossProduct(VAB, VAC));

  return V;
}

//rotate Point of Point[G] and originalPoint[P] and normalVector[R] and Angle[Q]
Point3d rotatePoint(Point3d G, Point3d P, Vector3d R, double Q){
  Point3d G0 = G;

  G.X =
    +(R.X * R.X * (1 − Math.Cos(Q)) + Math.Cos(Q)) * G0.X
    + (R.X * R.Y * (1 − Math.Cos(Q)) − R.Z * Math.Sin(Q)) * G0.Y
    + (R.X * R.Z * (1 − Math.Cos(Q)) + R.Y * Math.Sin(Q)) * G0.Z
    − P.X * (R.X * R.X * (1 − Math.Cos(Q)) + Math.Cos(Q) − 1)
    − P.Y * (R.X * R.Y * (1 − Math.Cos(Q)) − R.Z * Math.Sin(Q))
    − P.Z * (R.X * R.Z * (1 − Math.Cos(Q)) + R.Y * Math.Sin(Q))
    ;
  G.Y =
    +(R.Y * R.X * (1 − Math.Cos(Q)) + R.Z * Math.Sin(Q)) * G0.X
    + (R.Y * R.Y * (1 − Math.Cos(Q)) + Math.Cos(Q)) * G0.Y
    + (R.Y * R.Z * (1 − Math.Cos(Q)) − R.X * Math.Sin(Q)) * G0.Z
    − P.X * (R.Y * R.X * (1 − Math.Cos(Q)) + R.Z * Math.Sin(Q))
    − P.Y * (R.Y * R.Y * (1 − Math.Cos(Q)) + Math.Cos(Q) − 1)
    − P.Z * (R.Y * R.Z * (1 − Math.Cos(Q)) − R.X * Math.Sin(Q))
    ;
  G.Z =
    +(R.Z * R.X * (1 − Math.Cos(Q)) − R.Y * Math.Sin(Q)) * G0.X
    + (R.Z * R.Y * (1 − Math.Cos(Q)) + R.X * Math.Sin(Q)) * G0.Y
    + (R.Z * R.Z * (1 − Math.Cos(Q)) + Math.Cos(Q)) * G0.Z
    − P.X * (R.Z * R.X * (1 − Math.Cos(Q)) − R.Y * Math.Sin(Q))
    − P.Y * (R.Z * R.Y * (1 − Math.Cos(Q)) + R.X * Math.Sin(Q))
    − P.Z * (R.Z * R.Z * (1 − Math.Cos(Q)) + Math.Cos(Q) − 1)
    ;

  return G;
}

//rotate normalVector of Vector[N] and originalPoint[P] and normalVector[R] and Angle[Q]
Vector3d rotateNVector(Vector3d N, Vector3d R, double Q){
  Vector3d N0 = N;

  N.X =
    +(R.X * R.X * (1 − Math.Cos(Q)) + Math.Cos(Q)) * N0.X
    + (R.X * R.Y * (1 − Math.Cos(Q)) − R.Z * Math.Sin(Q)) * N0.Y
    + (R.X * R.Z * (1 − Math.Cos(Q)) + R.Y * Math.Sin(Q)) * N0.Z
    ;
  N.Y =
    +(R.Y * R.X * (1 − Math.Cos(Q)) + R.Z * Math.Sin(Q)) * N0.X
    + (R.Y * R.Y * (1 − Math.Cos(Q)) + Math.Cos(Q)) * N0.Y
    + (R.Y * R.Z * (1 − Math.Cos(Q)) − R.X * Math.Sin(Q)) * N0.Z
    ;
  N.Z =
    +(R.Z * R.X * (1 − Math.Cos(Q)) − R.Y * Math.Sin(Q)) * N0.X
    + (R.Z * R.Y * (1 − Math.Cos(Q)) + R.X * Math.Sin(Q)) * N0.Y
    + (R.Z * R.Z * (1 − Math.Cos(Q)) + Math.Cos(Q)) * N0.Z
    ;

  return N;
}
```

```
}

//Intersection of Point[AP] to Vector[AR] and Point[BP] to Vector[BR]
Point3d LLIntersection(Point3d AP, Vector3d AR, Point3d BP, Vector3d BR){
  Point3d CP = new Point3d();
  double x0 = AP.X;
  double y0 = AP.Y;
  double z0 = AP.Z;
  double x1 = BP.X;
  double y1 = BP.Y;
  double z1 = BP.Z;
  double s0 = AR.X;
  double t0 = AR.Y;
  double u0 = AR.Z;
  double s1 = BR.X;
  double t1 = BR.Y;
  double u1 = BR.Z;
  double n0 = new double();
  double n1 = new double();

  if(s0 == 0 && t0 == 0 && u0 == 0){
    CP.X = x0;
    CP.Y = y0;
    CP.Z = z0;
  } else if(
    (s0 != 0 && u1 == 0)
    || (t0 != 0 && t1 == 0)
  || (s0 != 0 && s1 == 0)){

    if(u0 != 0 && u1 == 0){
      n0 = (z1 − z0) / u0;
    } else if(t0 != 0 && t1 == 0) {
      n0 = (y1 − y0) / t0;
    } else if(s0 != 0 && s1 == 0){
      n0 = (x1 − x0) / s0;
    }
    CP.X = x0 + s0 * n0;
    CP.Y = y0 + t0 * n0;
    CP.Z = z0 + u0 * n0;
  } else {

    if(s1 == 0 && t1 == 0 && u1 == 0){
      CP.X = x1;
      CP.Y = y1;
      CP.Z = z1;
    } else if(u0 == 0 && u1 != 0){
      n1 = (z0 − z1) / u1;
    } else if(t0 == 0 && t1 != 0){
      n1 = (y0 − y1) / t1;
    } else if(s0 == 0 && s1 != 0){
      n1 = (x0 − x1) / s1;
    } else if((u0 != 0 && s1 != 0) || (u1 != 0 && s0 != 0)){
      n1 = ((x1 − x0) * u0 − (z1 − z0) * s0) / (u1 * s0 − s1 * u0);
    } else if((t0 != 0 && u1 != 0) || (t1 != 0 && u0 != 0)){
      n1 = ((z1 − z0) * t0 − (y1 − y0) * u0) / (t1 * u0 − u1 * t0);
    } else if((s0 != 0 && t1 != 0) || (s1 != 0 && t0 != 0)){
      n1 = ((y1 − y0) * s0 − (x1 − x0) * t0) / (s1 * t0 − t1 * s0);
    }
    CP.X = x1 + s1 * n1;
    CP.Y = y1 + t1 * n1;
    CP.Z = z1 + u1 * n1;
  }

  return CP;
}

//ClosestPoint of Point[AP] to Vector[AR] and Point[BP] to Vector[BR]
Point3d LLclosestPoint(Point3d AP, Vector3d AR, Point3d BP, Vector3d BR){
  Point3d ABP = new Point3d();
  Vector3d ABV = new Vector3d();
  Point3d CP = new Point3d();
  ABV = normalizeVector(CrossProduct(AR, BR));
  CP = PLPclosestPoint(BP, AP, ABV);
  ABP = LLIntersection(AP, AR, CP, BR);
  return ABP;
}

//ClosestPoint of Line[AP,AR] and Line[BP,BR] and Line[CP,CR]
Point3d LLclosestPoint(
Point3d AP, Vector3d AR, Point3d BP, Vector3d BR, Point3d CP, Vector3d CR){
  Point3d P = new Point3d();
  Point3d AB = new Point3d();
  Point3d AC = new Point3d();
  Point3d BA = new Point3d();
  Point3d BC = new Point3d();
  Point3d CA = new Point3d();
  Point3d CB = new Point3d();
  Vector3d VAP = new Vector3d();
  Vector3d VBP = new Vector3d();
  Vector3d VCP = new Vector3d();
  Vector3d VP = new Vector3d();
  Point3d OAB = new Point3d();
  Point3d OAC = new Point3d();
  Point3d OBA = new Point3d();
  Point3d OBC = new Point3d();
  Point3d OCA = new Point3d();
  Point3d OCB = new Point3d();
  Point3d OA = new Point3d();
  Point3d OB = new Point3d();
  Point3d OC = new Point3d();
  Point3d OP = new Point3d();
  Point3d OAP = new Point3d();
  Point3d OBP = new Point3d();
  Point3d OCP = new Point3d();
  Point3d PA = new Point3d();
  Point3d PB = new Point3d();
  Point3d PC = new Point3d();
  Point3d P1 = new Point3d();
  Point3d P2 = new Point3d();
  Point3d P3 = new Point3d();

  AB = LLclosestPoint(AP, AR, BP, BR);
  AC = LLclosestPoint(AP, AR, CP, CR);
  BA = LLclosestPoint(BP, BR, AP, AR);
  BC = LLclosestPoint(BP, BR, CP, CR);
  CA = LLclosestPoint(CP, CR, AP, AR);
  CB = LLclosestPoint(CP, CR, BP, BR);
  VAP = addVector(ABVector(AB, BA), ABVector(AC, CA));
  VBP = addVector(ABVector(BA, AB), ABVector(BC, CB));
  VCP = addVector(ABVector(CA, AC), ABVector(CB, BC));
  VP = normalizeVector(CrossProduct(VAP, VCP));
  OAB = AB;
  OAC = PLPclosestPoint(AC, AB, VP);
  OBA = PLPclosestPoint(BA, AB, VP);
  OBC = PLPclosestPoint(BC, AB, VP);
  OCA = PLPclosestPoint(CA, AB, VP);
  OCB = PLPclosestPoint(CB, AB, VP);
```

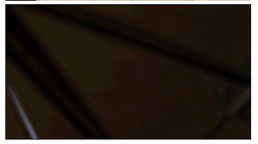




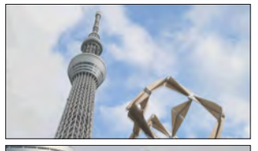
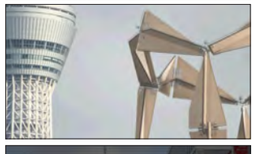



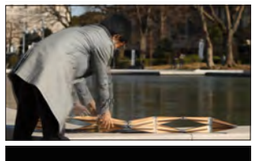

```
  OA = LLIntersection(
    OAB, normalVector(OAB, OAC),
    OBA, normalVector(OBA, OBC));
  OB = LLIntersection(
    OCA, normalVector(OCA, OCB),
    OBA, normalVector(OBA, OBC));
  OC = LLIntersection(
    OCA, normalVector(OCA, OCB),
    OAB, normalVector(OAB, OAC));
  OP = innercenterPoint(OA, OB, OC);
  OAP = PLclosestPoint(OA, normalVector(OA, OB), OP);
  OBP = PLclosestPoint(OB, normalVector(OB, OC), OP);
  OCP = PLclosestPoint(OC, normalVector(OC, OA), OP);
  PA = LLIntersection(BP, BR, OAP, VP);
  PB = LLIntersection(CP, CR, OBP, VP);
  PC = LLIntersection(AP, AR, OCP, VP);
  P1 = PLPclosestPoint(OP, PA, VP);
  P2 = PLPclosestPoint(OP, PB, VP);
  P3 = PLPclosestPoint(OP, PC, VP);
  if(PointDistance(P1, P2) > PointDistance(P2, P3)
  && PointDistance(P1, P2) > PointDistance(P1, P3)){
    P = P3;
  } else if(PointDistance(P1, P3) > PointDistance(P2, P3)
  && PointDistance(P1, P3) > PointDistance(P1, P2)){
    P = P2;
  } else {
    P = P1;
  }

  return P;
}

//distance of Point[A] and Point[B]
public static double PointDistance(Point3d A, Point3d B){
  double D = new Double();
  D = Math.Sqrt(
    (A.X - B.X) * (A.X - B.X)
    + (A.Y - B.Y) * (A.Y - B.Y)
    + (A.Z - B.Z) * (A.Z - B.Z));
  return D;
}

//Hypotenuse of Adjacent[A] and Opposite[B]
double Hypotenuse(double A, double B){
  double D = new double();
  D = Math.Sqrt((A * A + B * B));
  return D;
}

//Opposite of Hypotenuse[H] and Adjacent[B]
double Opposite(double H, double B){
  double D = new double();
  D = Math.Sqrt(H * H - B * B);
  if(B > H){ D = -Math.Sqrt(B * B - H * H); }
  return D;
}

//closestPoint of Point[A] and Plane[O,U]
public static Point3d PLPclosestPoint(Point3d A, Point3d O, Vector3d U){
  Point3d P = new Point3d();

  if(U.X == 0){
    P.X = A.X;
  } else {
    P.X = A.X
    + U.X * ((U.X * O.X + U.Y * O.Y + U.Z * O.Z)
    - (U.X * A.X + U.Y * A.Y + U.Z * A.Z));
  }
  if(U.Y == 0){
    P.Y = A.Y;
  } else {
    P.Y = A.Y
    + U.Y * ((U.X * O.X + U.Y * O.Y + U.Z * O.Z)
    - (U.X * A.X + U.Y * A.Y + U.Z * A.Z));
  }
  if(U.Z == 0){
    P.Z = A.Z;
  } else {
    P.Z = A.Z
    + U.Z * ((U.X * O.X + U.Y * O.Y + U.Z * O.Z)
    - (U.X * A.X + U.Y * A.Y + U.Z * A.Z));
  }
  return P;
}

//Angle of Line[O,A] and Line[O,B]
public static double P3Angle(Point3d O, Point3d A, Point3d B){
  double R = new Double();
  Vector3d OA = normalVector(O, A);
  Vector3d OB = normalVector(O, B);
  R = Math.Acos(
    (OA.X * OB.X + OA.Y * OB.Y + OA.Z * OB.Z)
    / (Math.Sqrt(OA.X * OA.X + OA.Y * OA.Y + OA.Z * OA.Z)
    * Math.Sqrt(OB.X * OB.X + OB.Y * OB.Y + OB.Z * OB.Z))
    );
  if(double.IsInfinity(R) == true){
    R = 0;
  }
  return R;
}

//Angle of Vecotr[A] and Vector[B]
double V2Angle(Vector3d A, Vector3d B){
  double R = new Double();
  R = Math.Acos(
    (A.X * B.X + A.Y * B.Y + A.Z * B.Z)
    / (Math.Sqrt(A.X * A.X + A.Y * A.Y + A.Z * A.Z)
    * Math.Sqrt(B.X * B.X + B.Y * B.Y + B.Z * B.Z))
    );
  if(double.IsInfinity(R) == true){
    R = 0;
  }
  return R;
}

//move Point of Point[A] and normalVector[V] and Length[L]
public static Point3d movePoint(Point3d A, Vector3d V, double L){
  A.X += V.X * L;
  A.Y += V.Y * L;
  A.Z += V.Z * L;
  return A;
}

//Cosines A of Length[A] and Length[B] and Length[C]
public static double ABCCos(double A, double B, double C){
  double Cos = new double();
  Cos = (B * B + C * C - A * A) / (2 * B * C);
  return Cos;
}
```

```
}

//Height AH of Length[A] and Length[B] and Length[C]
double ABCHeight(double A, double B, double C){
  double H = new double();
  H = Math.Sqrt(Math.Abs(C * C
    - (C * C - B * B + A * A) * (C * C - B * B + A * A) / ((2 * A) * (2 * A))));
  return H;
}

//closestPoint of Line[AP,AR] and Point[BP]
Point3d PLclosestPoint(Point3d AP, Vector3d AR, Point3d BP){
  Point3d IntP = new Point3d();
  double T = new Double();

  T = (AR.X * ( BP.X - AP.X ) + AR.Y * ( BP.Y - AP.Y ) + AR.Z * ( BP.Z - AP.Z ))
    / (AR.X * AR.X + AR.Y * AR.Y + AR.Z * AR.Z);
  IntP.X = AP.X + AR.X * T;
  IntP.Y = AP.Y + AR.Y * T;
  IntP.Z = AP.Z + AR.Z * T;

  return IntP;
}

//CrossProduct of Vector[VA] and Vector[VB]
public static Vector3d CrossProduct(Vector3d VA, Vector3d VB){
  Vector3d VC = new Vector3d();

  VC.X = VA.Y * VB.Z - VB.Y * VA.Z;
  VC.Y = VA.Z * VB.X - VB.Z * VA.X;
  VC.Z = VA.X * VB.Y - VB.X * VA.Y;

  // ■ for avoid errors
  if(Math.Abs(VC.Z) <= 0.000000001){ VC.Z = 0; }

  return VC;
}

//DotProduct of Vector[VA] and Vector[VB]
double DotProduct(Vector3d VA, Vector3d VB){
  double VD = new double();
  VD = VA.X * VB.X + VA.Y * VB.Y + VA.Z * VB.Z;
  return VD;
}

//Intersection of Plane[PA,VA] and Plane[PB,VB] and Distance[LA]
Point3d PLPLDIntersectionPlus(
Point3d PA, Vector3d VA, Point3d PB, Vector3d VB, double LA){
  Point3d IntP = new Point3d();
  Point3d IntPA = new Point3d();
  Point3d IntPB = new Point3d();
  Point3d IntPAB = new Point3d();
  Vector3d PAI = new Vector3d();
  Vector3d PBI = new Vector3d();
  Vector3d CPAB = new Vector3d();
  double DAC = new double();
  double DCP = new double();

  IntPA = PLPclosestPoint(PA, PB, VB);
  IntPB = PLPclosestPoint(PB, PA, VA);
  IntPAB = LLIntersection(
    PA, normalVector(PA, IntPB), PB, normalVector(PB, IntPA));
  PAI = normalVector(PA, IntPAB);
  PBI = normalVector(PB, IntPAB);

  CPAB = CrossProduct(PAI, PBI);

  CPAB = normalizeVector(CPAB);
  DAC = PointDistance(PA, IntPAB);
  DCP = Opposite(LA, DAC);
  IntP = movePoint(IntPAB, CPAB, DCP);


  return IntP;
}

//Intersection of Plane[PA,VA] and Plane[PB,VB] and Distance[LA]
Point3d PLPLDIntersectionMinus(
Point3d PA, Vector3d VA, Point3d PB, Vector3d VB, double LA){
  Point3d IntP = new Point3d();
  Point3d IntPA = new Point3d();
  Point3d IntPB = new Point3d();
  Point3d IntPAB = new Point3d();
  Vector3d PAI = new Vector3d();
  Vector3d PBI = new Vector3d();
  Vector3d CPAB = new Vector3d();
  double DAC = new double();
  double DCP = new double();

  IntPA = PLPclosestPoint(PA, PB, VB);
  IntPB = PLPclosestPoint(PB, PA, VA);
  IntPAB = LLIntersection(
    PA, normalVector(PA, IntPB), PB, normalVector(PB, IntPA));
  PAI = normalVector(PA, IntPAB);
  PBI = normalVector(PB, IntPAB);

  CPAB = CrossProduct(PBI, PAI);

  CPAB = normalizeVector(CPAB);
  DAC = PointDistance(PA, IntPAB);
  DCP = Opposite(LA, DAC);
  IntP = movePoint(IntPAB, CPAB, DCP);

  return IntP;
}

//closestPoint of Point[PA] on Plane[PA,VA] on Plane[PB,VB]
Point3d PPLPLclosestPoint(Point3d PA, Vector3d VA, Point3d PB, Vector3d VB){

  Point3d IntPA = new Point3d();
  Point3d IntPB = new Point3d();
  Point3d IntPAB = new Point3d();

  IntPA = PLPclosestPoint(PA, PB, VB);
  IntPB = PLPclosestPoint(PB, PA, VA);
  IntPAB = LLIntersection(
    PA, normalVector(PA, IntPB), PB, normalVector(PB, IntPA));

  return IntPAB;

}

//Mirroring Point on Plane[O,U] of Point[A]
Point3d PLPMirror(Point3d A, Point3d O, Vector3d U){
  Point3d P = new Point3d();
  Point3d B = new Point3d();
  Vector3d VAP = new Vector3d();
  double DAP = new double();
```






```
    P = PLPclosestPoint(A, O, U);
    VAP = normalVector(A, P);
    DAP = PointDistance(A, P);
    B = movePoint(A, VAP, DAP * 2);

    return B;

  }

  //circumcenterPoint of Point[A] and Point[B] and Point[C]
  Point3d circumcenterPoint(Point3d A, Point3d B, Point3d C){
    Point3d P = new Point3d();

    Point3d AB = new Point3d();
    Point3d AC = new Point3d();
    Vector3d VAB = new Vector3d();
    Vector3d VAC = new Vector3d();
    Vector3d V = new Vector3d();
    Vector3d VABP = new Vector3d();
    Vector3d VACP = new Vector3d();

    AB = ABMiddlePoint(A, B);
    AC = ABMiddlePoint(A, C);
    VAB = normalVector(A, B);
    VAC = normalVector(A, C);
    V = normalizeVector(CrossProduct(VAB, VAC));
    VABP = normalizeVector(CrossProduct(V, VAB));
    VACP = normalizeVector(CrossProduct(V, VAC));
    P = LLIntersection(AB, VABP, AC, VACP);

    return P;
  }

  //Intersection of Circle[A,DA] and Circle[B.DB] on Plan[A,B,V]
  Point3d CCIntersectionPlus(Point3d A, Point3d B, double DA, double DB, Vector3d V){
    Point3d P = new Point3d();
    Vector3d VAB = new Vector3d();
    Vector3d U = new Vector3d();
    double DAB = new double();

    VAB = normalVector(A, B);
    U = normalizeVector(CrossProduct(VAB, V));
    DAB = PointDistance(A, B);
    P = movePoint(A, VAB, DA);
    P = rotatePoint(P, A, U, Math.Acos(ABCCos(DB, DA, DAB)));

    return P;
  }

  //Intersection of Circle[A,DA] and Circle[B.DB] on Plan[A,B,V]
  Point3d CCIntersectionMinus(Point3d A, Point3d B, double DA, double DB, Vector3d V){
    Point3d P = new Point3d();
    Vector3d VAB = new Vector3d();
    Vector3d U = new Vector3d();
    double DAB = new double();

    VAB = normalVector(A, B);
    U = normalizeVector(CrossProduct(VAB, V));
    DAB = PointDistance(A, B);
    P = movePoint(A, VAB, DA);
    P = rotatePoint(P, A, U, -Math.Acos(ABCCos(DB, DA, DAB)));

    return P;
  }

  //area of Point[A] and Point[B] and Point[C]
  public static double Heron(Point3d A, Point3d B, Point3d C){
    double AB = new double();
    double BC = new double();
    double CA = new double();
    AB = PointDistance(A, B);
    BC = PointDistance(B, C);
    CA = PointDistance(C, A);
    double Heron = Math.Sqrt(
      (AB * AB + BC * BC + CA * CA) * (AB * AB + BC * BC + CA * CA)
      - 2 * (Math.Pow(AB, 4) + Math.Pow(BC, 4) + Math.Pow(CA, 4))) / 4;
    return Heron;
  }

  //bisector of Point[O] and Point[A] and Point[B]
  Vector3d bisector(Point3d O, Point3d A, Point3d B){
    Vector3d V = new Vector3d();

    Vector3d VOA = new Vector3d();
    Vector3d VOB = new Vector3d();
    VOA = normalVector(O, A);
    VOB = normalVector(O, B);
    V.X = (VOA.X + VOB.X) / 2;
    V.Y = (VOA.Y + VOB.Y) / 2;
    V.Z = (VOA.Z + VOB.Z) / 2;

    return V;
  }

  //innercenterPoint of Point[A] and Point[B] and Point[C]
  Point3d innercenterPoint(Point3d A, Point3d B, Point3d C){
    Point3d P = new Point3d();
    Vector3d VAP = new Vector3d();
    Vector3d VBP = new Vector3d();
    VAP = bisector(A, B, C);
    VBP = bisector(B, A, C);
    P = LLIntersection(A, VAP, B, VBP);
    return P;
  }

  //inscribedcircleRadius of Point[A] and Point[B] and Point[C]
  double inscribedcircleRadius(Point3d A, Point3d B, Point3d C){
    double R = new double();
    double S = new double();
    double AB = new double();
    double BC = new double();
    double CA = new double();
    AB = PointDistance(A, B);
    BC = PointDistance(B, C);
    CA = PointDistance(C, A);
    S = Heron(A, B, C);
    R = S * 2 / (AB + BC + CA);
    return R;
  }


  //boolean of Point[P] is in Inscribedcircle[A,B,C]
  bool onInscribedcircle(Point3d P, Point3d A, Point3d B, Point3d C){
    bool BL = new bool();
    Vector3d U = new Vector3d();
    Point3d O = new Point3d();
    Point3d Q = new Point3d();

    double R = new double();
    O = innercenterPoint(A, B, C);
    R = inscribedcircleRadius(A, B, C);
    U = normalizeVector(CrossProduct(normalVector(A, B), normalVector(A, C)));
    Q = PLPclosestPoint(P, A, U);

    if(PointDistance(O, Q) < R){
      BL = true;
    } else {
      BL = false;
    }

    return BL;
  }

  //boolean of Point[P] is over the Plane[O,U]
  bool PointOverPL(Point3d P, Point3d O, Vector3d U){
    bool BL = new bool();
    Point3d A = new Point3d();
    Point3d B = new Point3d();
    A = movePoint(O, U, PointDistance(O, P));
    B = movePoint(O, U, -PointDistance(O, P));
    if(PointDistance(A, P) <= PointDistance(B, P)){
      BL = true;
    } else {
      BL = false;
    }

    return BL;
  }

  /* ■■■■■■■■■■ CLASS ■■■■■■■■■■ */
  //KaleidocycleGeometry class
  public class KaleidocycleGeometry{

    public List<Point3d> Point = new List<Point3d>();
    public List<Point3d> PointT = new List<Point3d>();
    public List<Point3d> PointU = new List<Point3d>();
    public List<Vector3d> Vector = new List<Vector3d>();
    public List<Polyline> Lines = new List<Polyline>();
    public Point3d PointA = new Point3d();
    public Point3d PointB = new Point3d();

    public void setupKaleido(List<Point3d> P, List<Vector3d> V,
    double LA, double MA, double LB, double MB){
      PointA = P[0];
      PointB = P[1];

      for(int i = 2; i < P.Count; i++){
        Point.Add(P[i]);
        PointT.Add(P[i]);
        PointU.Add(P[i]);
      }
      for(int i = 0; i < V.Count; i++){
        Vector.Add(V[i]);
      }

      moveKaleido(LA, MA, LB, MB);
      lineKaleido();

    }

    public void moveKaleido(double LA, double MA, double LB, double MB){
      for(int i = 0; i < PointT.Count; i++){
        if( % 2 == 0){
          PointT[i] = movePoint(PointT[i], Vector[i], LA / 2);
          PointT[i] = movePoint(PointT[i], Vector[i], MA);
          PointU[i] = movePoint(PointU[i], Vector[i], -LA / 2);
          PointU[i] = movePoint(PointU[i], Vector[i], MA);
        } else {
          PointT[i] = movePoint(PointT[i], Vector[i], LB / 2);
          PointT[i] = movePoint(PointT[i], Vector[i], MB);
          PointU[i] = movePoint(PointU[i], Vector[i], -LB / 2);
          PointU[i] = movePoint(PointU[i], Vector[i], MB);
        }
      }
    }

    public void lineKaleido(){
      PointT.Add(PointT[0]);
      PointU.Add(PointU[0]);
      for(int i = 0; i < 4 * (PointT.Count - 1); i++){
        Lines.Add(new Polyline());
      }
      for(int i = 0; i < (PointT.Count - 1); i++){
        Lines[i * 4].Add(PointT[i]);
        Lines[i * 4].Add(PointU[i]);
        Lines[i * 4].Add(PointT[i + 1]);
        Lines[i * 4].Add(PointT[i]);

        Lines[i * 4 + 1].Add(PointT[i]);
        Lines[i * 4 + 1].Add(PointU[i]);
        Lines[i * 4 + 1].Add(PointU[i + 1]);
        Lines[i * 4 + 1].Add(PointT[i]);

        Lines[i * 4 + 2].Add(PointT[i]);
        Lines[i * 4 + 2].Add(PointT[i + 1]);
        Lines[i * 4 + 2].Add(PointU[i + 1]);
        Lines[i * 4 + 2].Add(PointT[i]);

        Lines[i * 4 + 3].Add(PointU[i]);
        Lines[i * 4 + 3].Add(PointT[i + 1]);
        Lines[i * 4 + 3].Add(PointU[i + 1]);
        Lines[i * 4 + 3].Add(PointU[i]);
      }
      PointT.RemoveAt(PointT.Count - 1);
      PointU.RemoveAt(PointU.Count - 1);
    }
  }

  //</Custom additional code>
```