LUALCAD

～レーザー加工機用図面変換プログラム～

レーザー加工機のための 3D モデルデータの 2D 図面変換プログラムの開発
Development of a Program to Convert 3D Model Data Into 2D Drawings for Laser Processing Machines

クラークソン　到漣

1. 要旨

3DCAD で設計した立体物を，レーザー加工機で切断加工した板材で製作するためには，3D モデルデータを 2D 図面に変換する作業が必要となるが，この作業は煩雑な作業を含む。本研究は，3DCAD データをレーザー加工機で扱える 2D 図面に変換する作業を効率化するため，STL データをベクトル解析でスライスするアルゴリズムをもつ変換ソフト LUALCAD を開発し，その効果を検証した。

本研究では，主に 2 つのステップで，3D データを 2D 図面に変換する方法を発展させた。第 1 段階は，3D データを層ごとに分割して 2D 化した。この方法で作られた 2D 図面をレーザー加工で切り出せば，積層する形で 3DCAD で設計した立体物を再現できる。ただし，この方法では，各層の接着位置を人の手で決めるため，組み立て精度を確保できない。第 2 段階は，各層を柱で固定する形で 2D 図面を出力する方法である。これにより，第 1 の方法の弱点が解消された。さらに，現在，第 3 段階として，組み立てが容易で，部品をかみ合わせることで接着剤を不要にし，様々な方向からの組み合わせで立体物を構築する形で 2D 図面を出力する機能を実装中である。

2　研究目的・問題提起

2.1　デジタルファブリケーションを用いた立体製品の製作

本研究は、レーザー加工機がある製作現場，工場などの作業の効率化を目指したものである。レー

ザー加工機は、デジタルファブリケーションの一種で、CAD/CAM システムで作られたデジタルデータをもとに、木材、金属等の板材を高速かつ精密に加工することが可能である。レーザー加工機は、同じデジタルファブリケーションの一種である 3D プリンタ等と比較して、使い方次第では、他の技術以上に素早く精度のよい部品を製造し、立体物を製造できる技術である。

しかし、一部の高度なレーザー加工機を除くと 2 次元の形しか切り出せない。レーザー加工機で製作できる部品は、基本的には一定の厚みのある板状の部品である。レーザー加工機を用いて 3 次元の立体物を設計・製作するには、3D プリンタとは違い、板状の複数の部品を組み立てることを前提として考える必要がある。

2.2　レーザー加工機を用いて立体物を設計・製作する工程

レーザー加工機で 3 次元の物体を組み立てることは可能だが，Fusion360 などの 3DCAD ソフトウェアを使って手動で 3 次元 CAD をレーザー加工機が理解できる 2DCAD に変換する工程は、複数のソフトウェアを行き来するので効率が悪く，全体の製作時間の大半を占める。この工程には以下のような手順がある：

1. 立体物を Fusion360 で設計する際、レーザー加工機で製作することを前提に、使用する材料の厚みを意識して 3D図面を制作する。
2. 加工機で加工・製作できる板状の部品を、3D 図面の中から選ぶ。
3. 3D 図面中にある、レーザー加工機で切断できる板状の部品を投影できる面で「スケッチを開始」する。
4. 必要な部品の外形線がスケッチ平面に投影されていることを確認し、「スケッチを終了」する。

5. 5.で作成した新規図面を「.DXF ファイルとして保存」する。
6. AutoCAD で 5 で保存した.DXF ファイルを開き、レーザー加工機で実際に加工する部品図を制作する。このとき、材料の無駄をなくすために、部品と部品の間隔をできる限り狭めて配置する。
7. 2~6 の手順を 1 の手順で求めた必要部品一つ一つに行う。

上記で示した 3D 図面から 2D 図面に変換する工程は、煩雑かつ面倒で、時間がかかってしまう。本研究は、この工程をできる限り少ない操作と短い時間で効率化することを目指す。また、上記の手順 7 において、手動で位置調整を行っているので必ずしも材料面積が最小になるとは限らない。各必要部品の図面データを一つの 2D 図面にすべて落とし込む工程をも自動化することで使う材料面積を最小にできる，この自動化プログラムを LUALCAD（ルアルキャド）と呼称する。LUALCAD は、Laser cUtting ALgorithm for Converting 3d to 2d CAD の略である。

2.3　本研究の目的

本研究は，3DCAD データをレーザー加工機で扱える 2D 図面に変換する作業を効率化するため，STL データをベクトル解析でスライスするアルゴリズムをもつ変換ソフト LUALCAD を開発した。

3　研究方法

LUALCAD は Python 言語を用いたソフトウェアとなる。入力するデータは、3 次元の CAD データを STL データに変換したもの，レーザー加工機にセットする材料の厚さ，使用する自動化の機能の種類の選択である。出力はレーザー加工機が読み取ることができる DWG や DXF などの 2 次元

CAD 図面データである。

LUALCAD の基本となる機能は STL データのスライシングである。スライシングとは 3D 図面を複数の 2D 断面（層）に分割する工程を示す。これは STL データ中の三角形とスライス面との位置関係をベクトル演算を用いて行う。この機能を使用した 3 次元データから 2 次元図面の変換の効率化，組み立てる際の効率化を目指す機能を開発する。

本研究で使用したレーザー加工機は EpilogLaser 社の Fusion Pro 48 である。各自の PC で 3D モデリング、2D 図面変換、レーザー加工用 2D 部品図面を制作したうえで、レーザー加工機の制御 PC にデータを移行する。レーザー加工用 PC には CorelDRAW がインストールされている。同ソフトで DXF もしくは DWG 図面を開き、「印刷」ボタンを押すことで、レーザー加工機が使用できる状態になる。LUALCAD は、以上の工程を前提としたプログラムである。

4　開発結果：LUALCAD のアルゴリズムと機能

4.1　STL データとその読み取り

LUALCAD が入力するデータに使用する 3 次元 CAD の STL（メッシュ）データとは、3 次元の物体の表面を三角形で表した 3 次元データ形式であり，3D プリンターに入力するデータ形式と同様である。これは Fusion360 などの 3 次元 CAD ソフトウェアで 3 次元の物体を設計し，STL データとして出力することでできる。

Python で STL データを読み込むには mikedh による Trimesh というメッシュ操作の Python のツールを使う。(https://github.com/mikedh/trimesh) Trimesh とグラフ表示用のモジュール Plotly を使って図 1 のようなプログラムをたて，簡単な STL データを読み込ませると図 2 のように表示される。これを図 3 のプログラムに入力すると，表 1 のように図 2 の物体の表面を構成する三角形メ

ッシュの各頂点が直交座標系で表わされる。図 2 の物体は計 108 個の三角形がある。LUALCAD は表 1 のデータを使って変換の自動化を行う。

4.2　積層型 3 次元物体の組み立て機能

LUALCAD の最も基本的な機能は、この積層型 3 次元物体の組み立てである。この機能は入力された STL 物体を等間隔にある一定の方向でスライシングを行い，各スライシングから得られた断面図をレーザー加工機が切り出す。そうすると元の 3 次元の物体が層ごとに出力され，接着剤で順番に積層することでレーザー加工機が出力した 2 次元の物体を 3 次元にすることができる。

4.2.1　STL データのスライシング演算

入力された STL データをスライスするには STL 物体全体を考えるのではなく，STL 物体を形作るメッシュの三角形一つ一つをスライスし，それぞれの断面図を統合することで STL 物体全体の断面図が完成する。各三角形の断面図を求めるには三角形の 3 つの辺とスライス面の交点を調べなければならない。

STL 物体中の各三角形の各辺は図 4 の通り，線分ABとして定義できる。さらに，線分ABの方向ベクトルを$\vec{D}$とする。スライス面も同様に図 4 の通り，面の法線ベクトルを$\vec{N}$，面上の点をPとする。線分ABとスライス面との交点をIとする。点A,B,P,Iの位置ベクトルをそれぞれ$\vec{A}, \vec{B}, \vec{P}, \vec{I}$とする。

最初に直線ABとスライス面の交点I'を調べる。点I'の位置ベクトルを$\vec{I'}$とする。

直線AB上の任意の点をL，その位置ベクトルを$\vec{L}$とする。tを実数とすると，直線ABは以下の 1)式のように表わせる：

$$\vec{L} = \vec{A} + t\vec{D} \qquad \cdots 1)$$

スライス面は面上の任意の点をX，その位置ベクトルを$\vec{X}$とするとスライス面は以下の 2)式のように表わせる：

$$\vec{N}\cdot\left(\vec{X}-\vec{P}\right)=0 \qquad \cdots 2)$$

直線ABとスライス面が交わるかどうかを 3)式で確認する：

$$\vec{N}\cdot\vec{D}\neq 0 \qquad \cdots 3)$$

3)式はスライス面の法線ベクトル$\vec{N}$と直線ABの方向ベクトル$\vec{D}$が垂直に交わらない，つまり直線ABとスライス面が平行でないことを確認する。3）式が成り立つ，つまり直線ABとスライス面は平行でないとき，直線ABとスライス面は交点が存在することになる。

1，2)式において直線ABとスライス面の交点は$\vec{X}=\vec{L}$を条件として持つので 2)式の$\vec{X}$に 1 式の$\vec{L}$を代入する：

$$\vec{N}\cdot\left(\vec{A}+t\vec{D}-\vec{P}\right)=0 \qquad \cdots 4)$$

4)式の内積を展開し，tについて解く：

$$\vec{N}\cdot\left(\vec{A}-\vec{P}\right)+t\vec{N}\cdot\vec{D}=0$$

$$t\vec{N}\cdot\vec{D}=-\vec{N}\cdot\left(\vec{A}-\vec{P}\right) \qquad \cdots 5)$$

ここで，3)式で直線ABとスライス面が交点を持つか確認をしたので 3)式が成り立っている前提で 5)式を変形する：

$$t=\frac{-\vec{N}\cdot\left(\vec{A}-\vec{P}\right)}{\vec{N}\cdot\vec{D}} \qquad \cdots 6)$$

6)式から 3)式が成り立つときの 1)式を用いた交点の位置を示す実数tが求まる。これを 1)式に代入すると$\vec{L}$は交点I'の位置ベクトル$\vec{I'}$を示す：

$$\vec{I'}=\vec{A}-\frac{\vec{N}\cdot\left(\vec{A}-\vec{P}\right)}{\vec{N}\cdot\vec{D}}\vec{D} \qquad \cdots 7)$$

交点I'は直線ABとスライス面であり，求めたいのは線分ABとスライス面の交点Iである。点I'が点Iとなる条件は点I'が線分AB上に存在することであり，この条件を式として表すと実数sを用いて以下の 8)式のようになる：

$$\overrightarrow{AI'} = s\overrightarrow{AB} \quad (0 \leq s \leq 1) \qquad \cdots 8)$$

これを満たす点I'の条件を求める

$$\overrightarrow{AI'} \cdot \overrightarrow{AB} = s\overrightarrow{AB} \cdot \overrightarrow{AB} \quad (0 \leq s \leq 1)$$

$\overrightarrow{AB}$は STL 物体中の三角形の辺を示すベクトルなので$\overrightarrow{AB} \cdot \overrightarrow{AB} \neq 0$より，

$$s = \frac{\overrightarrow{AI'} \cdot \overrightarrow{AB}}{\overrightarrow{AB} \cdot \overrightarrow{AB}}$$

sの定義範囲より

$$0 \leq \frac{\overrightarrow{AI'} \cdot \overrightarrow{AB}}{\overrightarrow{AB} \cdot \overrightarrow{AB}} \leq 1 \qquad \cdots 9)$$

7)式で求めた$\overrightarrow{I'}$が 9)式を満たすとき，点I'は線分ABとスライス面の交点Iを表す。三角形の 3 つの辺に選定の計算を行うと 3 通りの三角形の断面図が出力される。三角形の 3 つの辺のうち，スライス面との交点をもつ辺が 1 つ以下の場合，断面図には存在しない，または点として存在するので三角形とスライス面は交わらない，つまり断面図はないと判断をする。スライス面との交点をもつ辺が 2 つの場合，三角形とスライス面は交差しているので断面図に載る情報は三角形とスライス面の交差線となる。この交差線の始点と終点はそれぞれスライス面と交差する 2 つの辺との交点を結んだ線分となる。三角形の 3 辺がスライス面と交差する，つまり三角形の方向ベクトルとスライス面の方向ベクトルが一致する場合，三角形はスライス面と同様の面上にくるので断面図にすべて載せる。

以上の工程を STL データ中の各三角形にスライス面を一定にして行い，各三角形とスライス面との断面図を一つの図面に組み合わせると STL データ全体のスライスが出力される。結果は図 6 の通りになる。

4.2.2　余分な線の除去

STL 物体を真ん中でスライスする場合は 4.2.1 で述べた断面図の演算方法は図 6A のように使用できるが，図 6B のように STL 物体の面のスライスを出力させるとレーザー加工機に入力したい枠線以外の余分な線が見られる。問題の発生所を各三角形とスライス面の断面図の詳細を調べて検出した。STL データの一つの面の向きと同様のスライス面を定義するとその STL データの面を覆う三角形の向きもスライス面と同様になる。このことからスライス面と一致する三角形がすべて断面図に載ってしまい，余分な線はその三角形の斜辺からなるものである。

これを解消するにはスライスを行う際にスライス面と一致する三角形を断面図に載せないようにする。スライス面と一致する三角形を無視することでスライス面と交差し，スライス面と平行でない三角形だけから枠線を抽出する（図 7 参照)。これをもとのスライシングプログラムに導入すると STL 物体を面でスライスしても図 8 のように枠線部分だけが残る。

4.2.3　連続的なスライシング，パッキングと組み立て

これまでのスライシングプログラムを等間隔で与えられた STL 物体をある方向に切り刻む，つまり連続的にスライシングを行う。プログラムは図 9 の通りになる。図 9 中の「slice_stl()」関数がスライシングを行うプログラムである。このプログラムを用い，図 2 の STL 物体を Z 軸方面，つまり下から上へと切り刻む，連続的にスライシングを行うと複数の層が図面となって別々にでてくる。各図面をグリッド・パッキングアルゴリズムに通して一つの図面にグリッド状に並べると図 10 のように出力される。

図 10 で出力された図面はレーザー加工機に読み込むことができ，実際にベクター加工をこの図面で MDF 材料に施した。のちに手動で層を接着剤で組み合わせた（図 11 参照)。

4.3　ガイダンス付き積層型 3 次元物体の組み立て機能

4.2 で紹介した積層型の組み立て機能はレーザー加工機で層を切り出したのちに手動で層一つ一つを接着剤で組み立てなければならないので層がずれてしまう（図 11 左上あたり）。そこで，手動でも組み立てる際に層をすべて固定してから接着剤でつなげ合わせることを可能にするプログラムである。

このプログラムは積層型の組み立て機能で出力した各層の図面の回りに四角い枠を足し，その 4 つ辺にそれぞれ穴をあける。この付け足しに加え，プログラムは 4 本の柱を同時にレーザー加工機で出力するように図面の下側に挿入する。図 2 の STL 物体をこのプログラムに通すと出力される図面は図 12 の通りになる。

図 12 をレーザー加工機で切り出し後，接着剤を使わずに簡単に積み重ねる。次に枠の 4 つの穴にレーザー加工機で出力された 4 つの柱を挿入し，すべての層をその位置に固定する。固定された層をすべて接着剤で合わせる。このように柱で層を固定することで手動でも精度の高い 3 次元物体がレーザー加工機が出力した 2 次元積層部品からできる。

4.4　より効率的な組み立て方法の自動化

図 11 のように一方向にレーザー加工機が出力した部品を接着剤で組み合わせるより様々な方向から部品を組み合わせて元の 3 次元の物体を製作する方が効率が良い。これは組み立てる際に必要な部品数が減少する上に様々な方向から部品を組み合わせているので向きの違う 2 つの部品の接合部分をレーザー加工で凹凸にすると嵌め合わせることができ，接着剤が必要なくなる。この機能はこの組み立て方に必要な部品を STL 物体から抽出するのが難点となる。

4.4.1　組み立てに必要な部品

様々な方向から部品を組み合わせる，つまり組み立て部品は元の 3 次元物体中には様々な方向で存在している。ここでは角ばった形だけ入力されるものとするので X,Y,Z 方向に向く組み立てに必要な部品を考える。この新しい組み立て方のメリットの一つである「部品数が少ない」ことに着目する。3 次元の物体を組み立てる際に必要な部品の数を最小にすることを目的とする。

ここで，連続的なスライシングを行う際に各断面図中の「パーツ」を定義する。パーツとは実際にレーザー加工機で出力されたときに材料部分になる箇所を指す。LUALCAD がパーツを理解するには各断面図の「材料部分」と「空白部分」を把握しなければならない。これを達成するには各断面図を幾何学的領域分類等の知識を用いた内外判定プログラムに通さなければならない。これを用いて図 14 のように各断面図のパーツを数える。この数値を「パーツ数」と以後表す。

最初に X,Y,Z 方向でありうるすべての部品を抽出するために全方向で連続的なスライシングを行う。図 2 の STL 物体を X,Y,Z 方向に連続的にスライシングを行うと図 13 のようになる。ここで，ある方向で抽出されたパーツ数が 2 以上の断面図はその部分だけが宙に浮いていない限り，必ず別の方向のスライシングで抽出されたパーツ数 1 の断面図と同様になる。これを用いるとパーツ数が 2 以上，つまり断面図に組み立て部品が 2 つ以上あったところが別の方向でのスライシングで見つけられるパーツ数 1 の断面図，つまり組み立て部品 1 つの図面を使用することで部品数を減らすことができる（図 15 参照)。この動作を X,Y,Z 方向でスライスして得たすべての断面図に行えば最終的に残る断面図はパーツ数 1 のものとなる。つまり，パーツ数 1 の断面図だけを抽出することで組み立てに必要な部品がでてくる。現時点ではこの組み立てに必要な部品を自動的に選択するプログラムが完成している。実際に図 2 の STL 物体を通すと，図 15 のようにパーツが 1 つの断面図だけが残る。

5　考察

本研究により，従来の手動での図面生成やレイアウト配置の煩雑さが大幅に軽減され，3D 設計から製造への工程が簡素化できることが分かった。これにより，製造コストの削減および時間短縮が可能となり，プロトタイピングや小規模な工房，研究機関でも利用しやすくなる。

6　結論・今後の展望

LUALCAD を用いて，3D モデルの STL データを 2D の DWG 形式に変換するプログラムの開発に成功した。STL データから自動で 2D 図面を生成し，パッキングアルゴリズムを用いた材料の効率的な配置が行えることが確認された。さらに，3D プリンターより短時間での加工が可能で，レーザー加工機で高精度な切断ができるため，完成品の組み立てがスムーズに進むが積層する際にズレが生じてしまう問題点がある。これを改善するにあたって，柱で各層を固定することで，積層構造が安定し，組み立て精度も向上すると考えられる。

一方で，3D 形状が複雑になるほど，スライシング処理に時間がかかることが課題として浮上した。この処理時間の短縮には，King, B., Rennie, A. & Bennett, G.による論文「An efficient triangle mesh slicing algorithm for all topologies in additive manufacturing.」で述べられている 2 つの方法,「ECC アルゴリズム」と「エッジ・マッチング」を活用できる。

今後の展望として積層型組み立て機能とそのガイダンス付きの機能にどんなユーザでも使えるような UI を作成する。LUALCAD へのアクセス方法は Github が提供するウェブアプリで製作して行う。さらに，積層型の組み立て機能は現時点ではベクター加工のみ出力されるのでどの部品がどこにくるかを示してくれるラスター加工を含めることで組み立てる際に生じる混乱を防ぐことができる。

組み立てに必要な部品の抽出に成功したが図 15 のままでははみ出てしまう部分があり，組み立てることは物理的に不可能となっている。これを解消するにはプログラムが抽出した部品を使って元の 3 次元物体を最適に組み立てる方法を検索する必要がある。これははみ出る部分の図面を除去することで組み立てを可能にする考えから基づいているが，「はみ出る部分の除去」は「はみ出る部分」が組み立て方によって変わってくるので一つに定まらないことからありうる組み立て方から最適のを選択しなければならない。ここで一番最適な組み立て方法は部品数が一番少ないことである。

7　参考文献

King, B., et. al. An efficient triangle mesh slicing algorithm for all topologies in additive manufacturing. Int J Adv Manuf Technol 112, 1023–1033 (2021). URL: https://link.springer.com/article/10.1007/s00170-020-06396-2

8　謝辞

本研究を行うにあたって，指導担当科目とは少々違った趣旨の研究であるにも関わらず、多大な指導・アドバイス、そして時間を費やしてくださった、東京科学大学附属科学技術高等学校機械システム分野の柴沼俊輔先生に感謝します。

9　図表・画像

```python
def view_stl(stl_path:str):

    data = m.Mesh.from_file(stl_path)
    # reshape Nx3x3 to N*3x3 to convert polygons to points
    points = data.vectors.reshape(-1, 3)
    faces_index = np.arange(len(points))  # get indexes of points

    N=3
    rows = (np.arange(points.shape[0])%N == (N-1)) + 1
    triangles = np.repeat(points, rows, axis=0)
    triangles = np.insert(triangles, [*range(4, len(triangles), 4)], [np.nan, np.nan, np.nan], axis=0)

    lines = [
            go.Mesh3d(
            x = points[:,0],  # pass first column of points array
            y = points[:,1],  # .. second column
            z = points[:,2],  # .. third column
            # i, j and k give the vertices of triangles
            i = faces_index[0::3],  # indexes of the points. k::3 means pass every third element
            j = faces_index[1::3],  # starting from k element
            k = faces_index[2::3],  # for example 2::3 in arange would be 2,5,8,...
            opacity = 1,
            color='lightpink'
        ),
        go.Scatter3d(
            x=triangles[:,0],  # pass first column of points array
            y=triangles[:,1],  # .. second column
            z=triangles[:,2],  # .. third column
            mode='lines',
            name='',
            line=dict(color='rgb(25,25,25)', width=7)
        )]


    fig = go.Figure(data=lines)
    fig.update_layout(width=1000, height=1000)

    fig.show()
```

図 1：STL データを開き，表示する

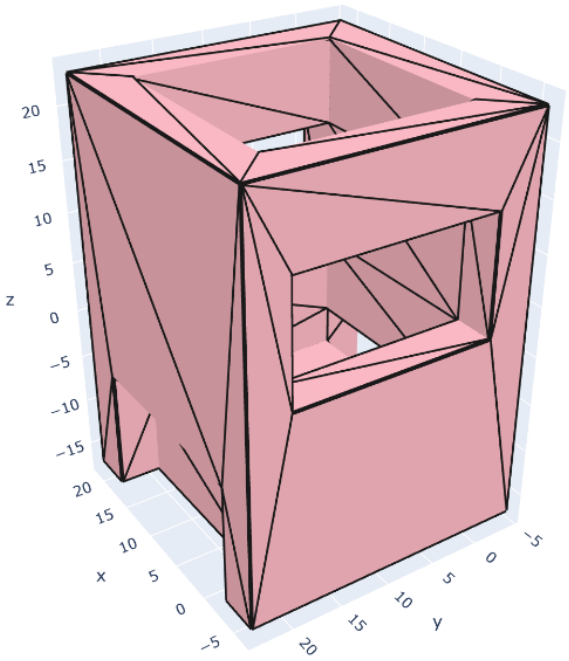


図 2：STL 物体の例

```python
def stl_to_xls(stl_file, output_xls):
    mesh = trimesh.load_mesh(stl_file)

    if not mesh.is_volume:
        raise ValueError("The input STL file does not contain a valid mesh with triangles.")

    workbook = xlsw.Workbook(output_xls)
    worksheet = workbook.add_worksheet()

    headers = ['Triangle', 'Vertex 1', 'Vertex 2', 'Vertex 3']
    worksheet.write_row(0, 0, headers)

    for i, face in enumerate(mesh.faces):
        v1, v2, v3 = mesh.vertices[face]

        row = [i+1, str(tuple(v1)), str(tuple(v2)), str(tuple(v3))]

        worksheet.write_row(i + 1, 0, row)

    workbook.close()

    print(f"STL data has been written to {output_xls}")
```

図 3：STL データの各三角形のデータを抽出

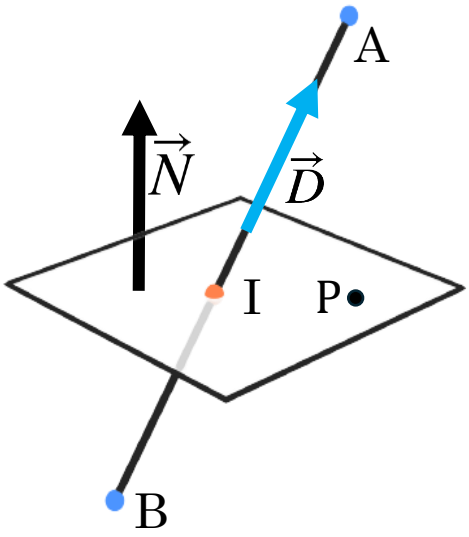
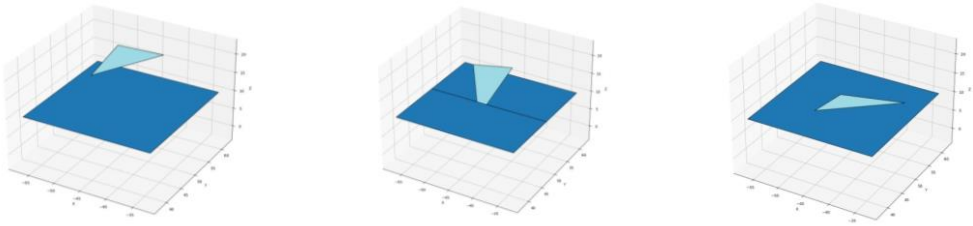


図 4：スライス面と三角形の辺の定義



1 点以下で切断面と交差する：
断面図にはない

2 点で切断面と交差する：
断面図には 2 点を結んだ線がある

3 点で切断面と交差する：
断面図には三角形すべ

図 5：断面図に載るデータ

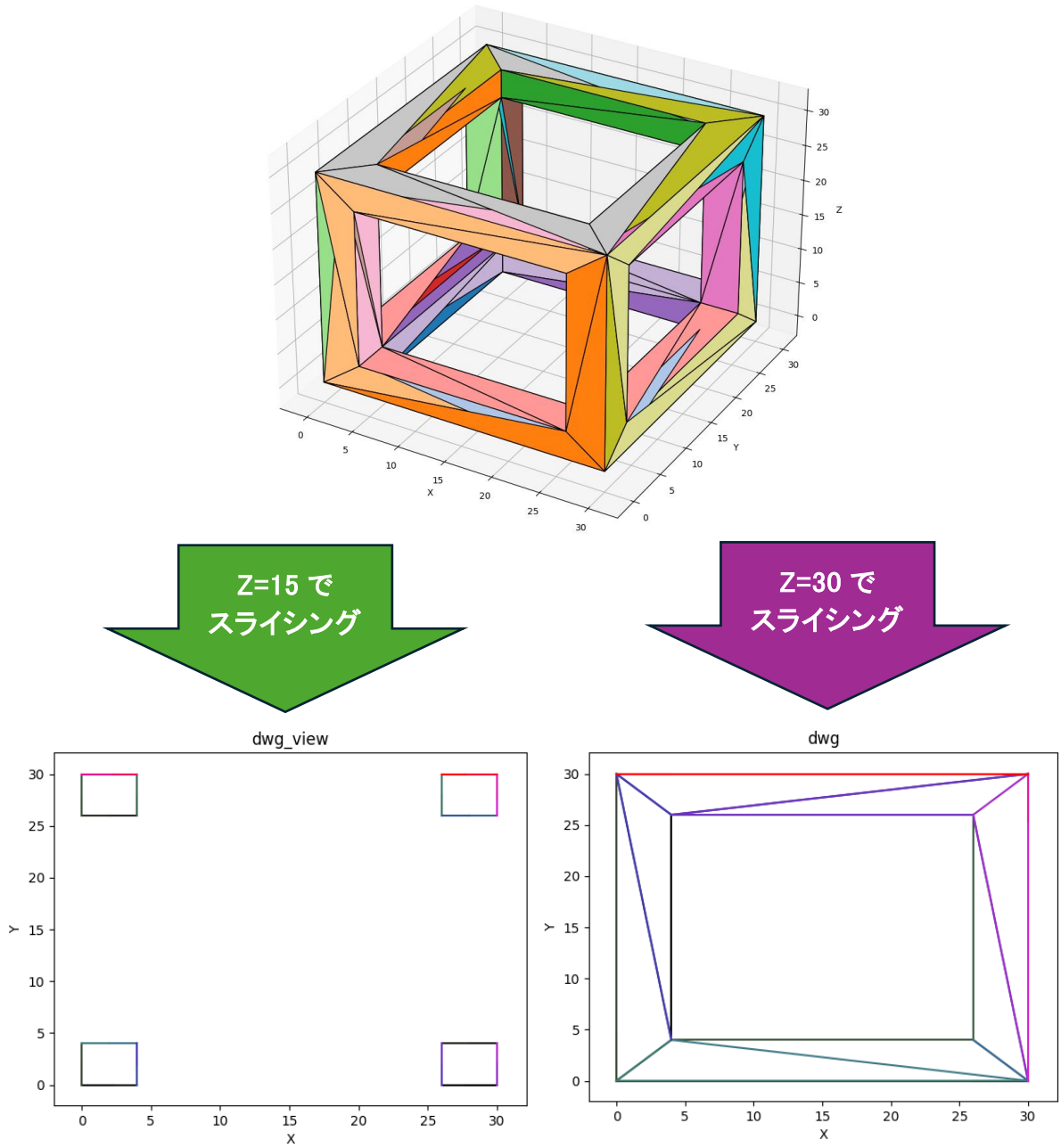


図 6：ベクトル演算のスライシング結果（左：A，右：B）

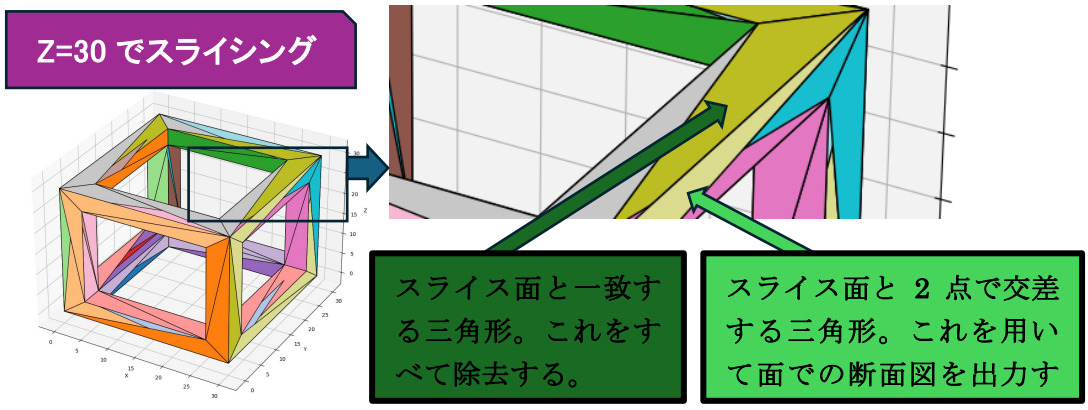


図 7 ：余分な線の除去，余分な三角形の除去

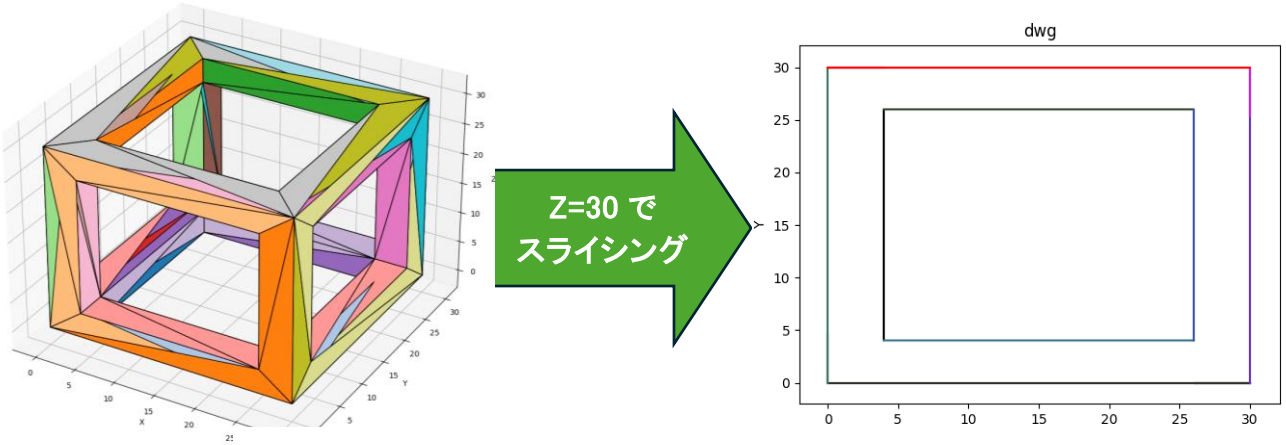


図８：余分な線の除去後の面のスライシング

```python
def width_slice_stl(input_trimesh:trimesh.Trimesh,sliceWidth:float,slicePlaneNormal:list = [0,0,1],printDeets:bool = False):
    docList = []
    points = np.unique(input_trimesh.vertices.reshape([-1, 3]), axis=0)
    slicePlaneNormal = np.array(slicePlaneNormal)/np.linalg.norm(slicePlaneNormal)
    out = find_signed_min_max_distances(points,slicePlaneNormal)
    minPoint = out[0]
    maxPoint = out[2]
    printIF(printDeets,f"min point:{minPoint} max point:{maxPoint}","width_slice_stl")
    a = []
    b = []
    for i in range(len(slicePlaneNormal)):
        if float(slicePlaneNormal[i]) == 0:
            a.append(0)
            b.append(0)
        else:
            a.append(maxPoint[i])
            b.append(minPoint[i])
    a, b = np.array(a), np.array(b)
    sliceNumbers = math.ceil(np.linalg.norm(a-b)/sliceWidth)
    slicePlanePoints = [b + slicePlaneNormal*i*sliceWidth+slicePlaneNormal*sliceWidth/2 for i in range(sliceNumbers)]
    for i in range(len(slicePlanePoints)):
        doc = slice_stl(input_trimesh,list(slicePlaneNormal),list(slicePlanePoints[i]),printDeets)
        docList.append(doc)
        printIF(printDeets,f"{i+1}/{len(slicePlanePoints)}: sliced mesh with plane point: {slicePlanePoints[i]}","width_slice_stl")

    return docList
```

図９：連続的にスライシングを行うプログラム

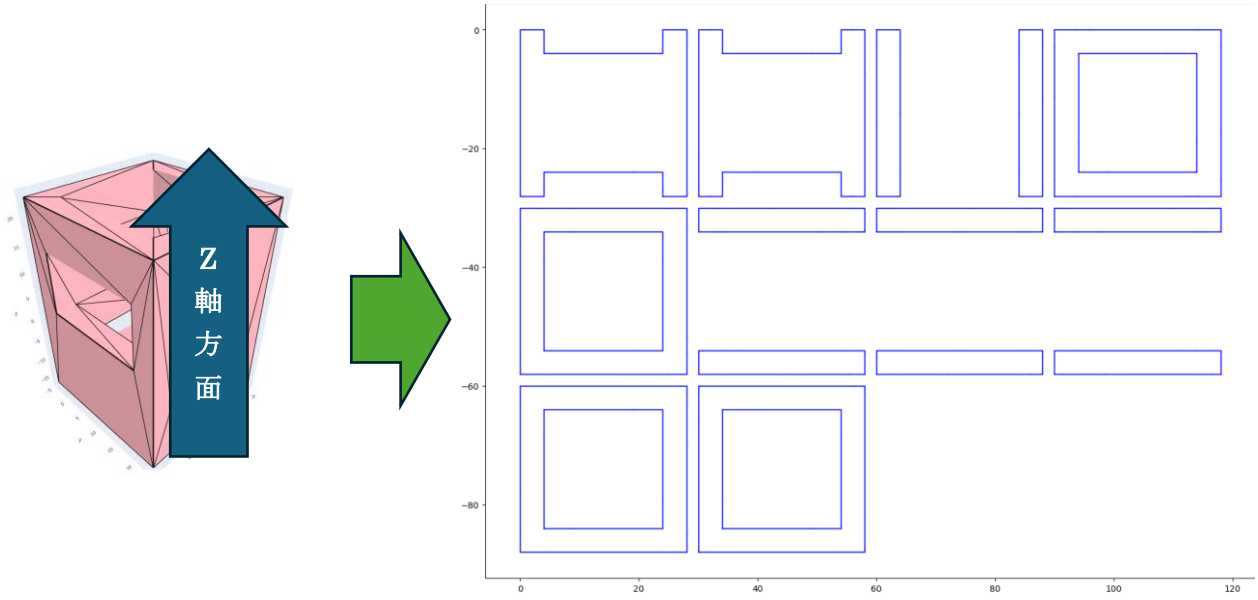


図１０：連続的なスライシングの出力

図 11：図面から元の 3 次元の物体へ

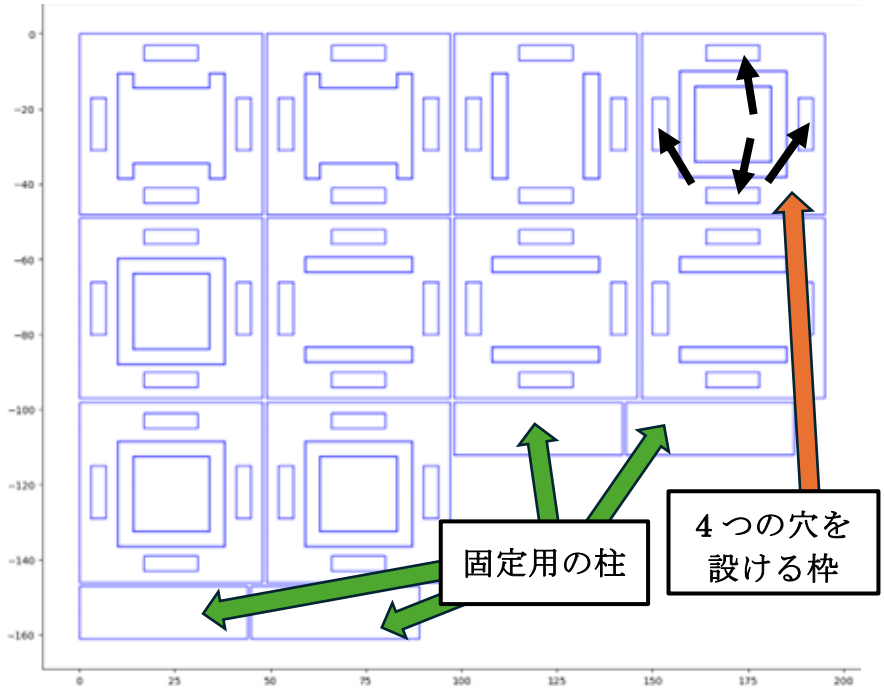


図 12：ガイダンスつきの積層組み立て機能の出力

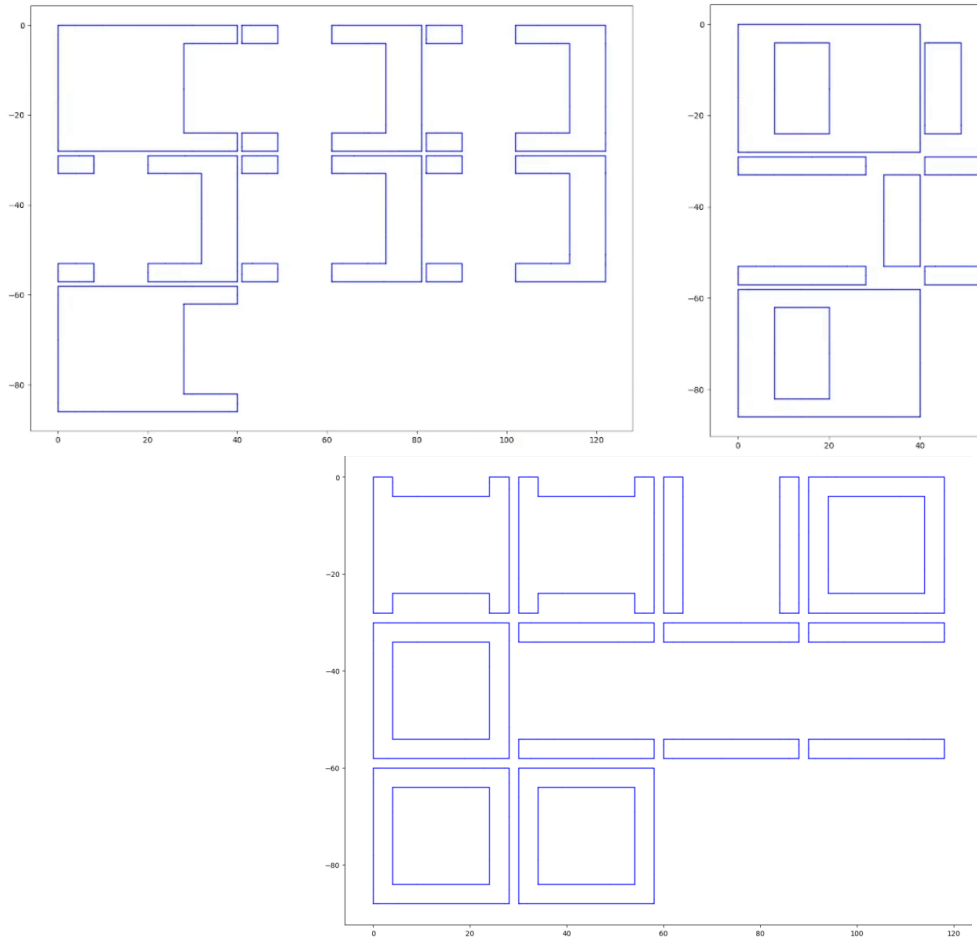


図 13：図 2 を X,Y,Z 方向で連続的にスライシング（左上：Y 方向，右上：X 方向，下：Z 方向）

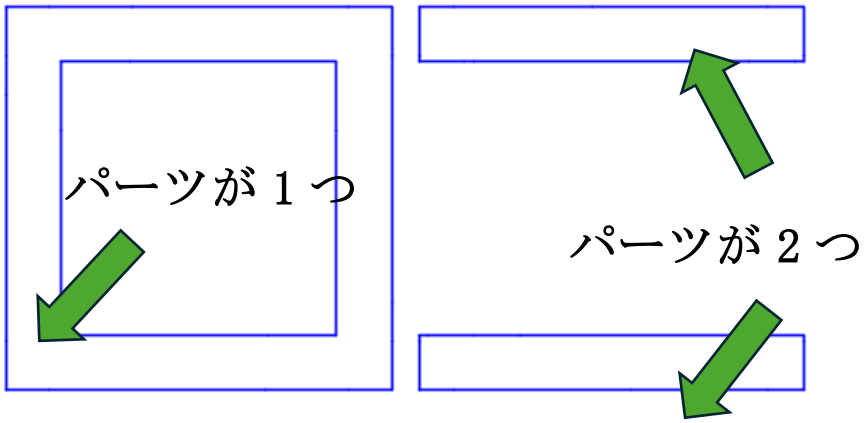


図 14：各断面図のパーツの定義と数え方

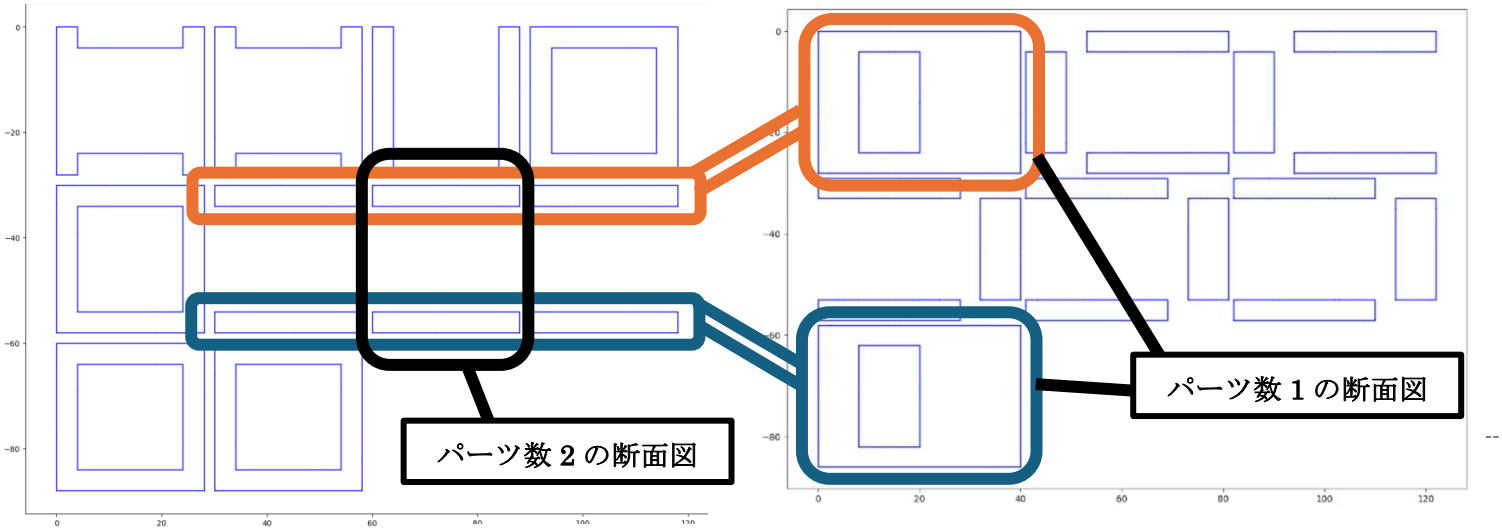


図 15：パーツ数と同じ部品を示す部品

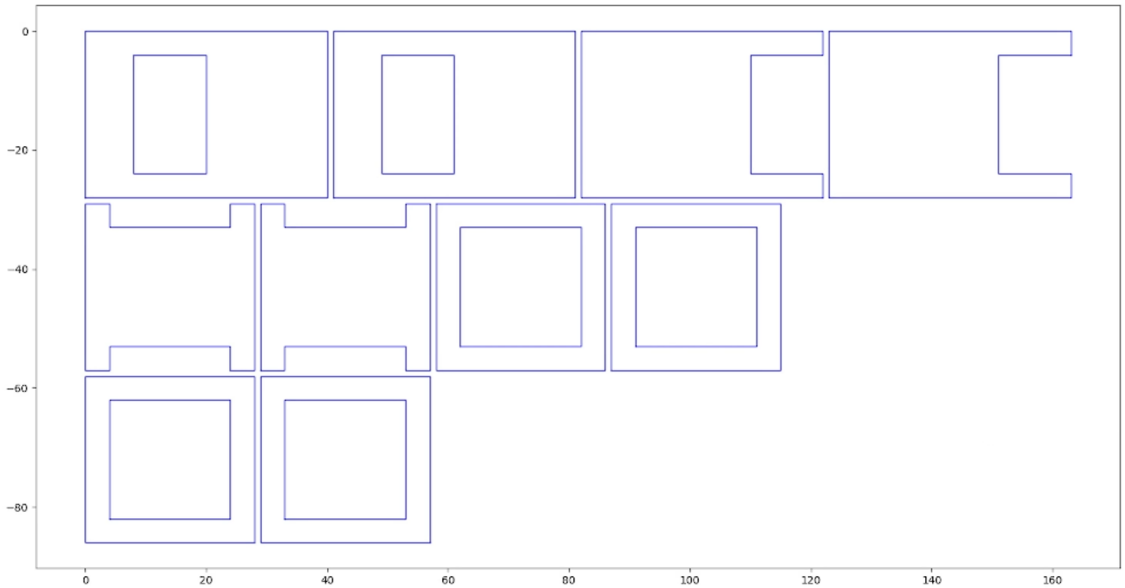


図 16：現時点での LUALCAD の最新機能の出力（組み立てに必要な部品）

Triangle	Vertex 1	Vertex 2	Vertex 3
1	(17.5, 23.5, -5.0)	(17.5, 19.5, -5.0)	(-2.5, 23.5, -5.0)
2	(-2.5, 23.5, -5.0)	(17.5, 19.5, -5.0)	(-2.5, 19.5, -5.0)
3	(-2.5, 19.5, -17.0)	(-2.5, 23.5, -17.0)	(-2.5, 19.5, -9.0)
4	(-2.5, 19.5, -9.0)	(-2.5, 23.5, -17.0)	(-2.5, 23.5, -5.0)
5	(-2.5, 19.5, -9.0)	(-2.5, 23.5, -5.0)	(-2.5, 19.5, -5.0)
6	(-2.5, 19.5, -9.0)	(-2.5, 19.5, -5.0)	(-2.5, -0.5000000596046448, -5.0)
7	(-2.5, -0.5000000596046448, -5.0)	(-2.5, 19.5, -5.0)	(-2.5, 19.5, 3.0)
8	(-2.5, -0.5000000596046448, -5.0)	(-2.5, 19.5, 3.0)	(-2.5, -0.5000000596046448, 3.0)
...	...	...	...
計108	（頂点１）	（頂点２）	（頂点３）

表 1：STL データの三角形データ

10　本研究のプログラム

LUALCAD は Github で最新版が見られます：

https://github.com/theplatecrafter/Larva