小山工業高等専門学校　電気電子創造工学科
平成 30 年度　卒業論文

凸多面体近似による肝臓部分切除領域推定の探索効率化に関する研究

A Study on the estimation of optimal partial excision area in liver surgery based on the convex hull approximation method

電気電子創造工学科
麦倉　柊太
Department of Innovative Electrical Electronic Engineering
Shuta MUGIKURA

指導教員　小林　康浩
Yasuhiro KOBAYASHI

-目次-

第 1 章　はじめに

肝臓外科手術の手術計画では、臓器・腫瘍の形状や大きさを確認し、実際に手術で切除する領域を推定する。特に部分切除を行う場合には、腫瘍からある程度余裕を持たせた領域を切除することが一般的であり、切除領域は医師の裁量に任されていた。そのため腫瘍に栄養を与えている血管を完全に取り切れないことがあり、再発のリスクが高くなる可能性があった。われわれは「100%切除領域」を含む放物曲面で部分切除することを想定し、切除体積が最小となる最適な部分切除領域を推定する手法について研究している。そのための手法として、最適切除領域推定に関する先行研究[1][2]、またそれを改善した放物面をモデルとした部分切除領域推定[3]が提案されている。

しかしながら、従来手法[3]では探索に膨大な時間が必要であり、また、探索によって得られた解の妥当性について十分な検証が行えていない。そのため本研究では、「100%切除領域」を凸多面体に変換して探索の効率化を行うことで、これらの問題点の改善を図る。

本論文の構成は以下のようになっている。
第 1 章にて本研究の背景、目的及び概要について述べる。
第 2 章では、従来手法では、肝臓部分切除をどのような形状で、かつどのようなアルゴリズムを用いて行っていたかについて述べる。
第 3 章では、本研究で導入したアルゴリズムについて述べる。
第 4 章では、探索結果とアルゴリズムの評価を行う。
第 5 章では、本研究で生じた問題点を改善するための提案手法について述べる。
第 6 章では、本論文の結論を述べる。

第 2 章　肝臓部分切除面のモデル化と従来手法

2.1　まえがき

本章では、従来手法[3]で使用されている 100%切除領域の切除形状とその探索方法について述べる。本研究では、従来手法[3]を改良し別の手法を追加することによって、従来手法より探索時間を短縮する。そのため、本研究においてもこの従来手法における切除形状とアルゴリズムが基本となっている。

2.2　切除領域形状

部分切除手法では図 2.1 のような手法が主流となっている。これは腫瘍から 1~2cm ほど間隔を開けて切除するという手法である。本来、肝臓の部分切除手法は腫瘍に栄養を与えている血管を含む領域（以下、100%切除領域）を完全に切除する必要がある。しかしこの手法では切除領域をとる腫瘍からの間隔が医師の裁量次第のため、再発のリスクが高くなる。

それを改善するため従来手法[3]では図 2.2 のような切除領域を考える。これは腫瘍を囲む 100%切除領域をスプーンですくうことのできるような放物体である。このように 100%切除領域を設定し切除することで、再発のリスクが低くなる。

実際に用いる画像に適用したときの切除曲面断面は、図 2.3 のようになる。

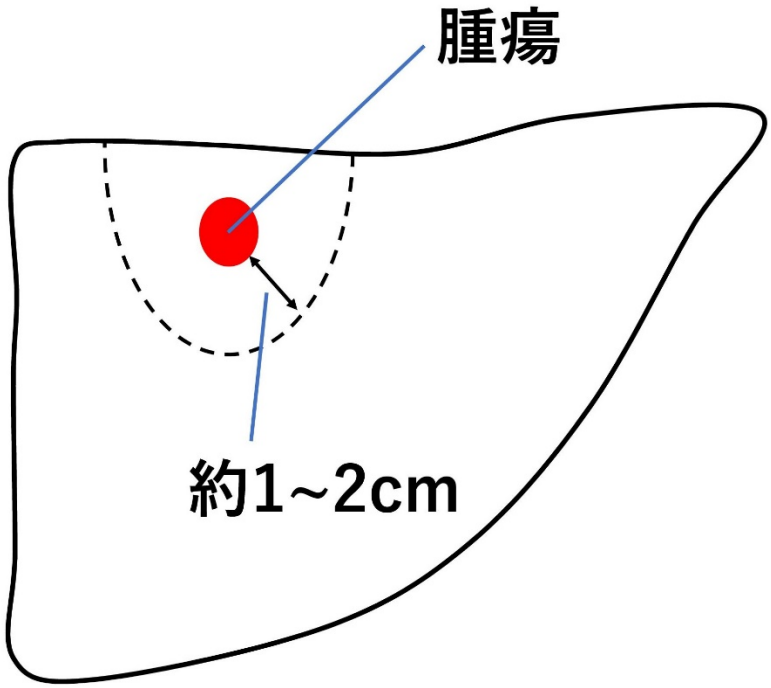


図 2.1　主流の切除手法

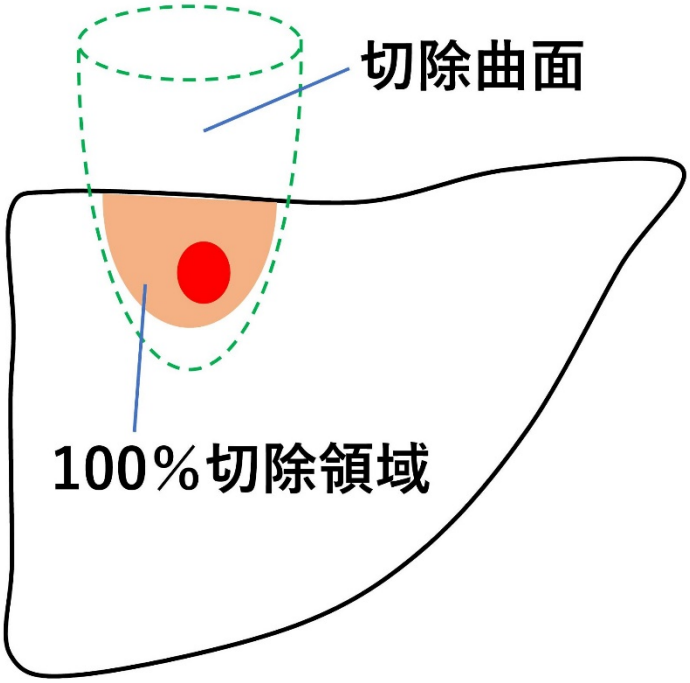


図 2.2　従来手法の切除手法

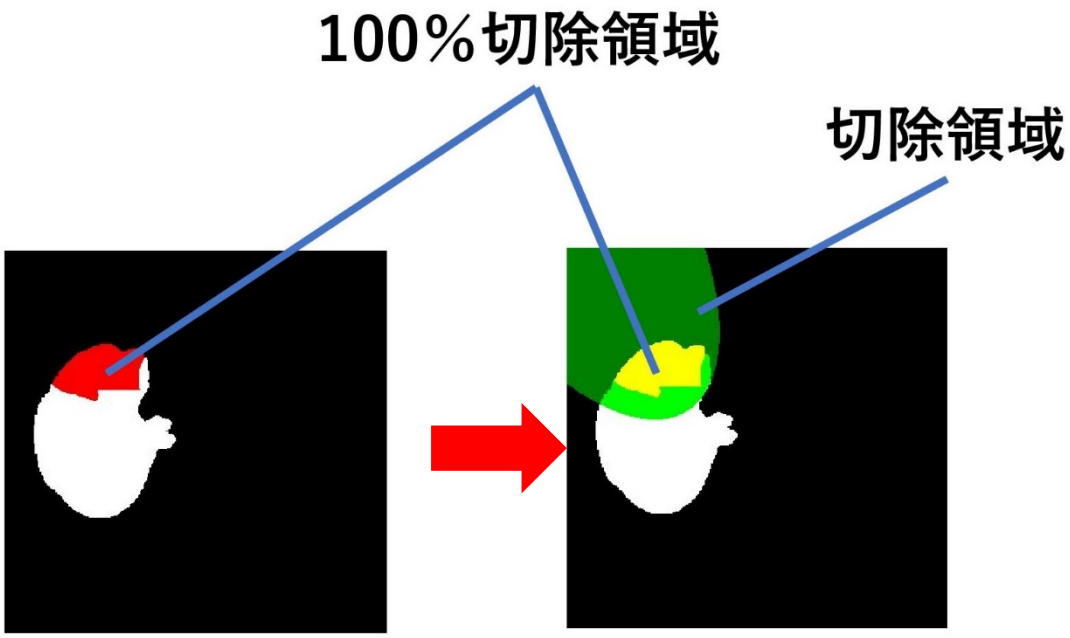


図 2.3　実際に用いる画像における切除曲面

2.3　切除曲面の表現方法

従来手法では、スプーンですくえるような曲面、つまり放物体で100%切除領域を切除することを想定している。

放物面の形状､位置､傾きは以下の 8 個のパラメータに依存する。

・回転(ロール・ピッチ・ヨー)　　: α、β、γ
・平衡移動　　: Tx、Ty、Tz
・焦点移動　　: Ay、Az

回転 α、β、γ は図 2.4 に示すとおり、それぞれ x、y、z 軸に対する回転角を表している。平行移動 Tx、Ty、Tz は図 2.5 に示すとおり､放物面の頂点が原点から移動する量を表している｡焦点距離 Ay、Az は図 2.6 に示すとおり放物面の y 軸、z 軸方向に対する広がりを表している。

なお、回転角は作用させる順番によって結果が変化してくる。今回は x-y-z 系のオイラー角を用いて回転させる。

そのようにすることで、回転変換前の座標を(X,Y,Z)とし、x、y、z に対する回転ベクトルをそれぞれ Rx、Ry、Rz とすると、式(2.1)によって回転変換後の座標(X',Y',Z')が与えられる。また、(X',Y',Z')および平衡移動と焦点距離を用いることで、式(2.2)により放物面を演算することが出来る。

$$\begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} = RzRyRx \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

$$= \begin{pmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (2.1)$$

$$(\mathrm{X}' - Tx) = Ay(Y' - Ty)^2 + Az(Z' - Tz)^2 \quad (2.2)$$

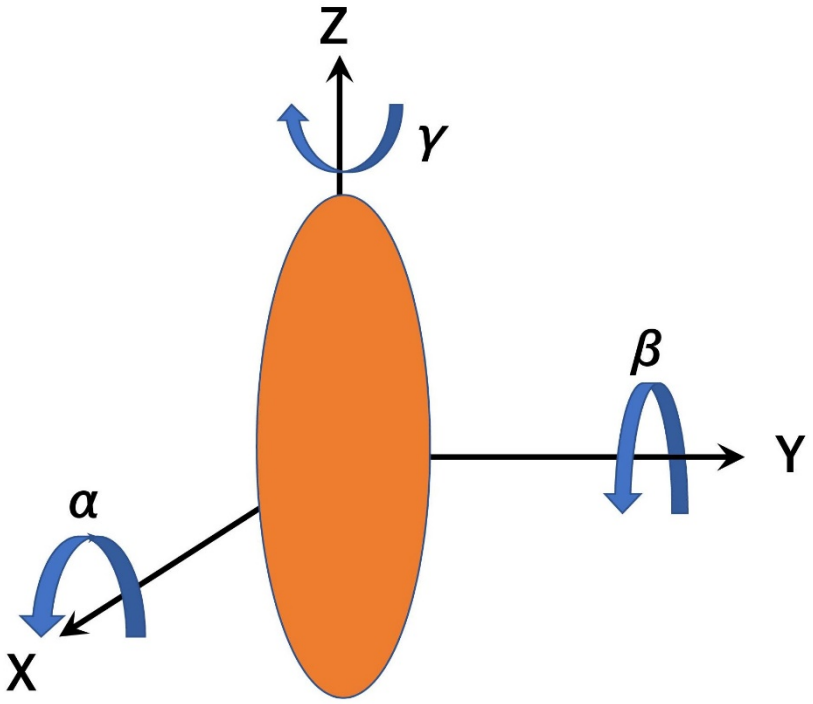


図 2.4　オイラー角による回転イメージ

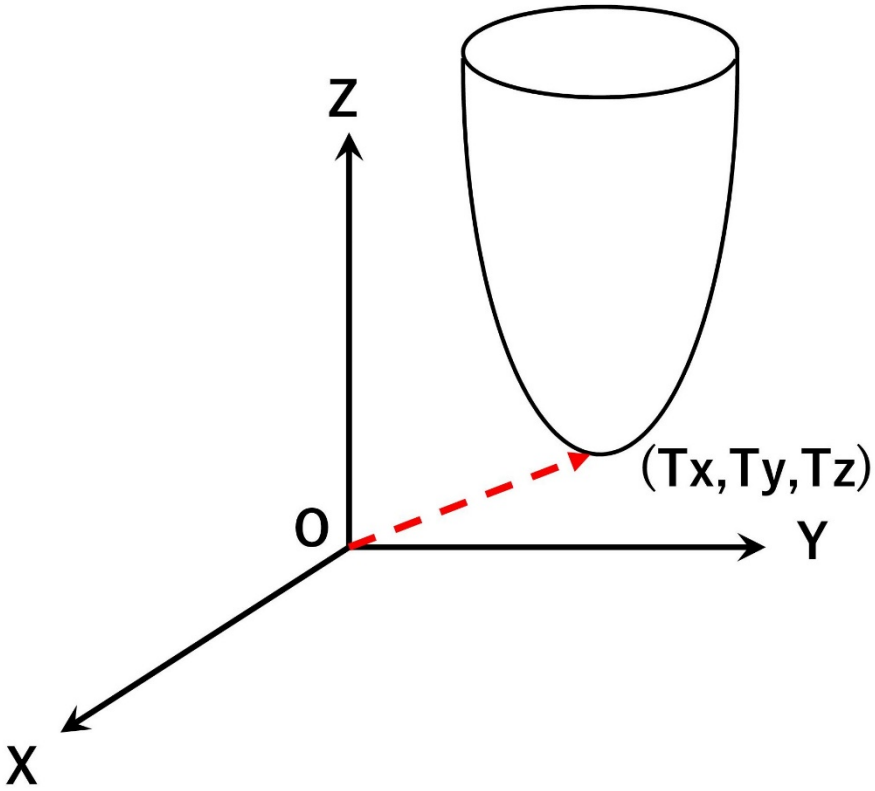
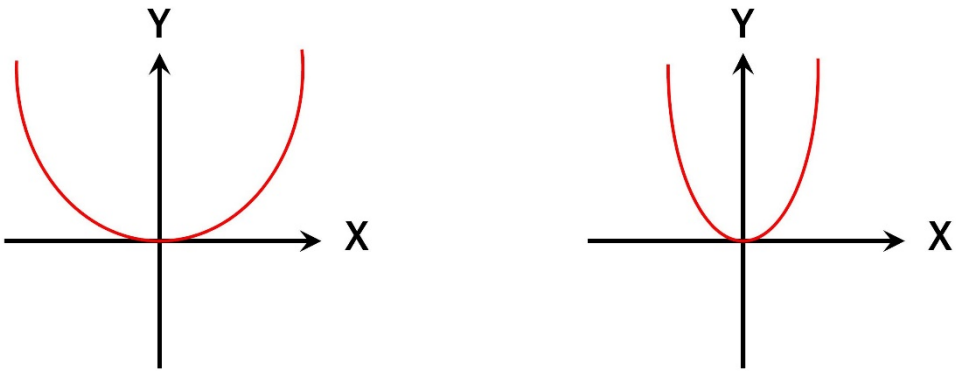


図 2.5　平衡移動のイメージ



(a)焦点距離大　　(b)焦点距離小

図 2.6　焦点距離のイメージ

2.4 従来の探索アルゴリズム

前項で述べた最適な放物面を探索するために用いることを検討した全探索とは、特定の探索範囲内のすべてのパラメータの組み合わせをすべて試すことで、最も良い結果のパラメータを求める探索方法である。しかしながら全探索では、取り得るパラメータすべてに対し演算するため、組み合わせ数に比例した探索時間を要する。探索の精度を要する部分切除領域の推定においては、その探索時間は膨大なものになる。参考文献[3]において 149,299,200,00 秒（約 3734 秒）が探索に要すると提示されており、それでは実用的ではない。

そのため、「求める必要のある切除曲面は、100%切除領域を完全に包含していなければならない」という前提条件を元に、図 2.7 中の判定条件「100%切除領域⊂放物面」に導入することで、無駄な処理を省略し高速化を図っている。処理を省略するための条件は以下の 2 つの場合である。また判定処理の流れを図 2.8 に示す。

・放物面の頂点が 100%切除領域内部にあるとき

図 2.9 に示すように、放物面の頂点が 100%領域内部にあるときは必ず 100%領域を完全に包含出来ない。放物面は頂点から最大 180° 方向しか包含することができないため、必然的に反対方向の 100%切除領域は残ってしまうためである。

・放物面の頂点は 100%切除領域の外部にあるが完全に包含出来ないとき

図 2.9 に該当しない、放物面の頂点が 100%切除領域外部にあるときもパラメータによっては処理を中止する。従来手法では大きな放物面から小さな放物面になるように焦点距離を変化させる。このとき他のパラメータは変化させない。図 2.10 のようにもし大きな放物面で 100%切除領域が完全に包含しない場合、それより小さな放物面では 100%切除領域を包含することが出来ない。

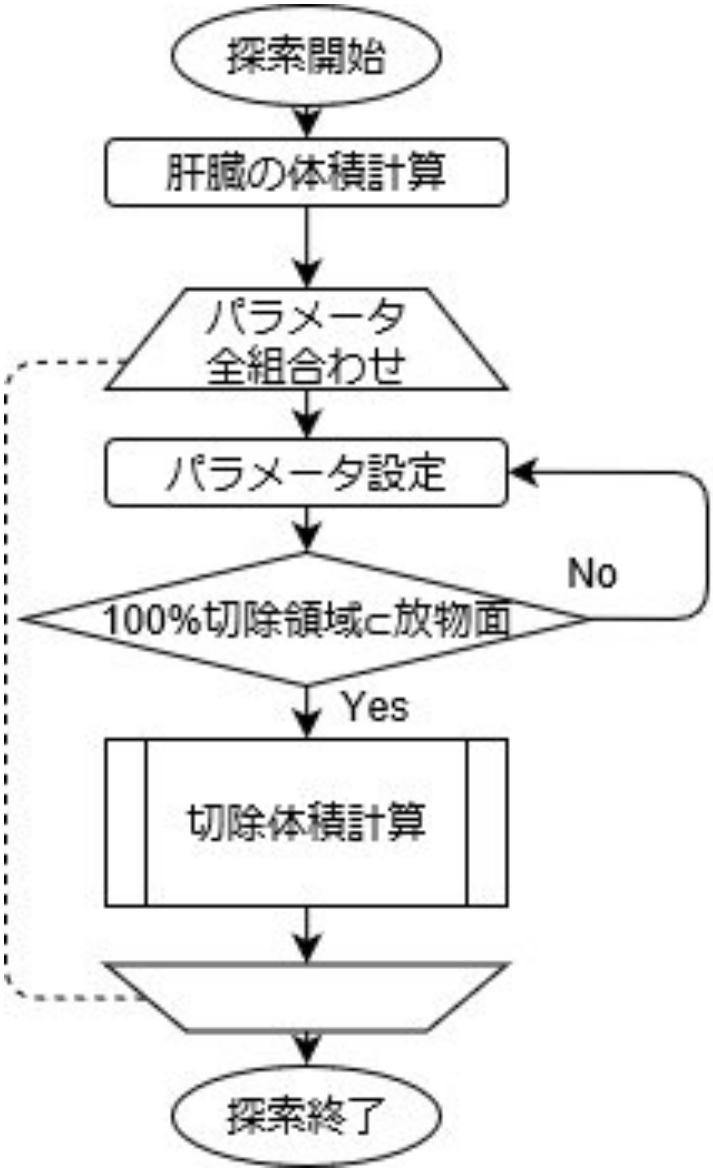

探索開始
肝臓の体積計算
パラメータ
全組合わせ
パラメータ設定
No
100%切除領域に放物面
Yes
切除体積計算
探索終了

図 2.7　処理の流れ

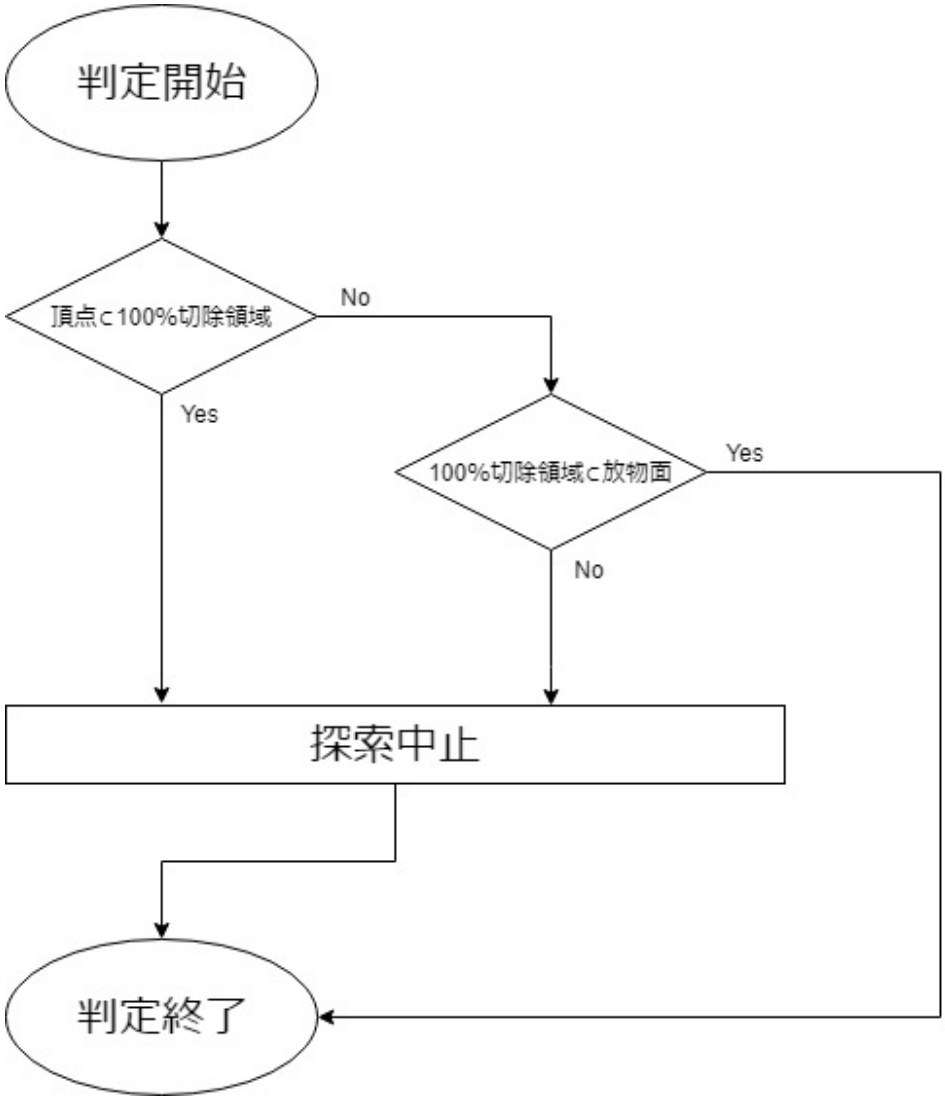


図 2.8　判定の流れ

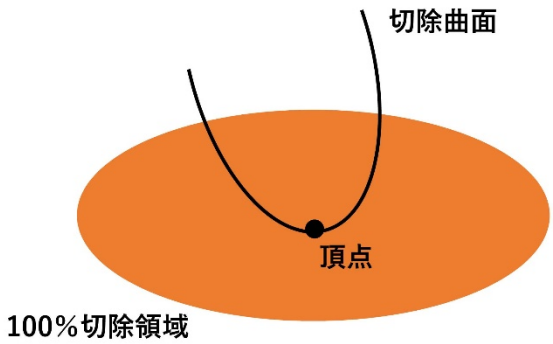


図 2.9　頂点が 100%切除領域内部にある場合

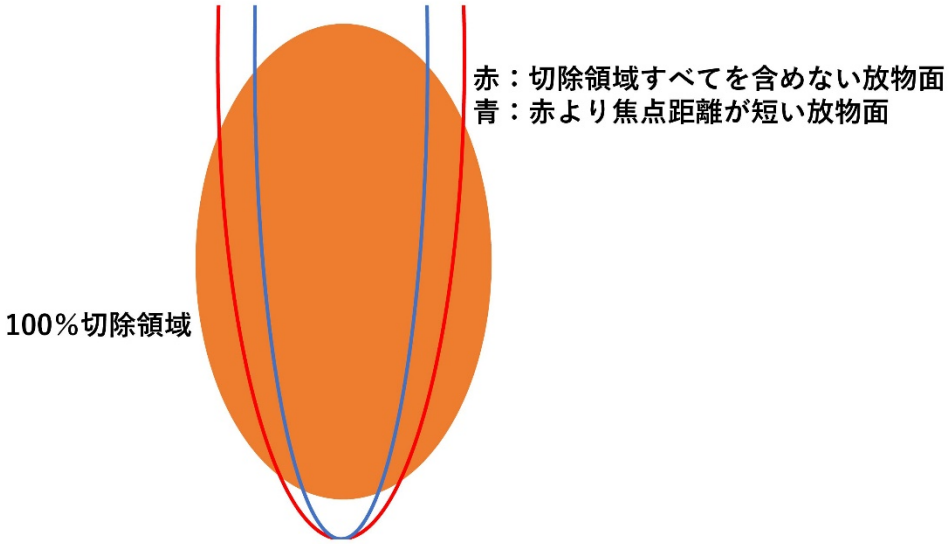


図 2.10　焦点距離による処理中断

2.5 むすび

本章では、従来手法[3]で用いている切除形状とその探索方法について述べた。切除形状としては放物面を採用し、探索方法は全探索を採用している。しかし全パラメータを全部計算する全探索では、膨大な時間を要する。そのため、探索時間を削減するため従来手法で採用していたアルゴリズムについても述べた。しかしこのアルゴリズムを用いても探索時間が膨大となってしまう。そのため、その探索時間をさらに削減するため本研究で採用した手法について次章で述べる。

第 3 章　探索における提案手法

3.1　まえがき

本章では、本研究の探索環境や、従来手法[3]からさらに探索時間を減少させるため追加した手法について述べる。本研究では解の妥当性検証が目的の 1 つとなっているため、探索方法は従来手法[3]と同様の全探索を用いた。

3.2　探索環境

本研究では、実際の医療機関に導入することができるシステムの構築を目的としている。以下の 3 つの理由によりフリーの画像処理ソフトウェア ImageJ を用いて探索を行う。

1. CT 画像で主流のプロトコル DICOM が使用できる
2. 生物学では、デファクト・スタンダードのソフトウェアである
3. フリーかつパブリックドメインなソフトウェアな為導入が容易

ImageJ は図 3.1 のようにプラグインをインストールすることで、機能を追加する。なお、ImageJ は Java と Python のプラグインに対応しているが、従来手法[3]が Java でプラグインを作成していたことを踏まえ、本研究においても Java での開発を行った。Java のプラグイン作成は Eclipse で行った。

表 3.1　環境構成

画像ソフトウェア	ImageJ 1.51k
言語	Java 1.8.0._66
IDE	Eclipse Neon.3 Release (4.6.3)
OS	Windows 8.1
CPU	AMD A10-7700K 3.4GHz
メモリ	8GB

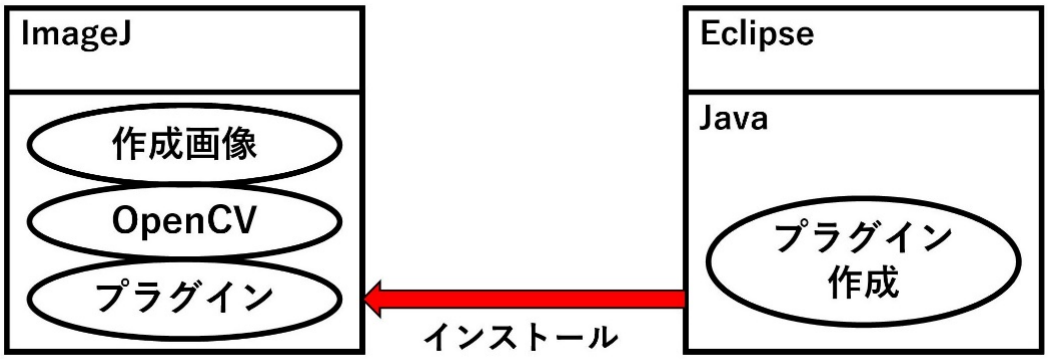


図 3.1　環境構築の構成

3.3 探索に使用する画像の作成

本研究で使用する画像は、下記の画像サンプルから作成した画像である。

・肝臓本体　(a)
・腫瘍　(b)
・血管　(c)
・100%切除領域　(d)

探索を行う前に画像を作成することで、作成画像を用いて探索を行った。画像作成の流れを図 3.2 に示す。まず、画像(a)(b)(c)の論理和を演算することで肝臓全体の画像を作成する。しかしこのままでは肝臓内部に穴が存在する画像となるため、本来の肝臓の画像を復元できていないと考えられる。そのためこの穴を埋める演算を行う。さらにその画像に対し 100%切除領域を赤くして先ほど作成した肝臓画像と合成することで、すべての情報を 1 つの画像にまとめる。そして本研究ではこの最終的な画像を用いる。

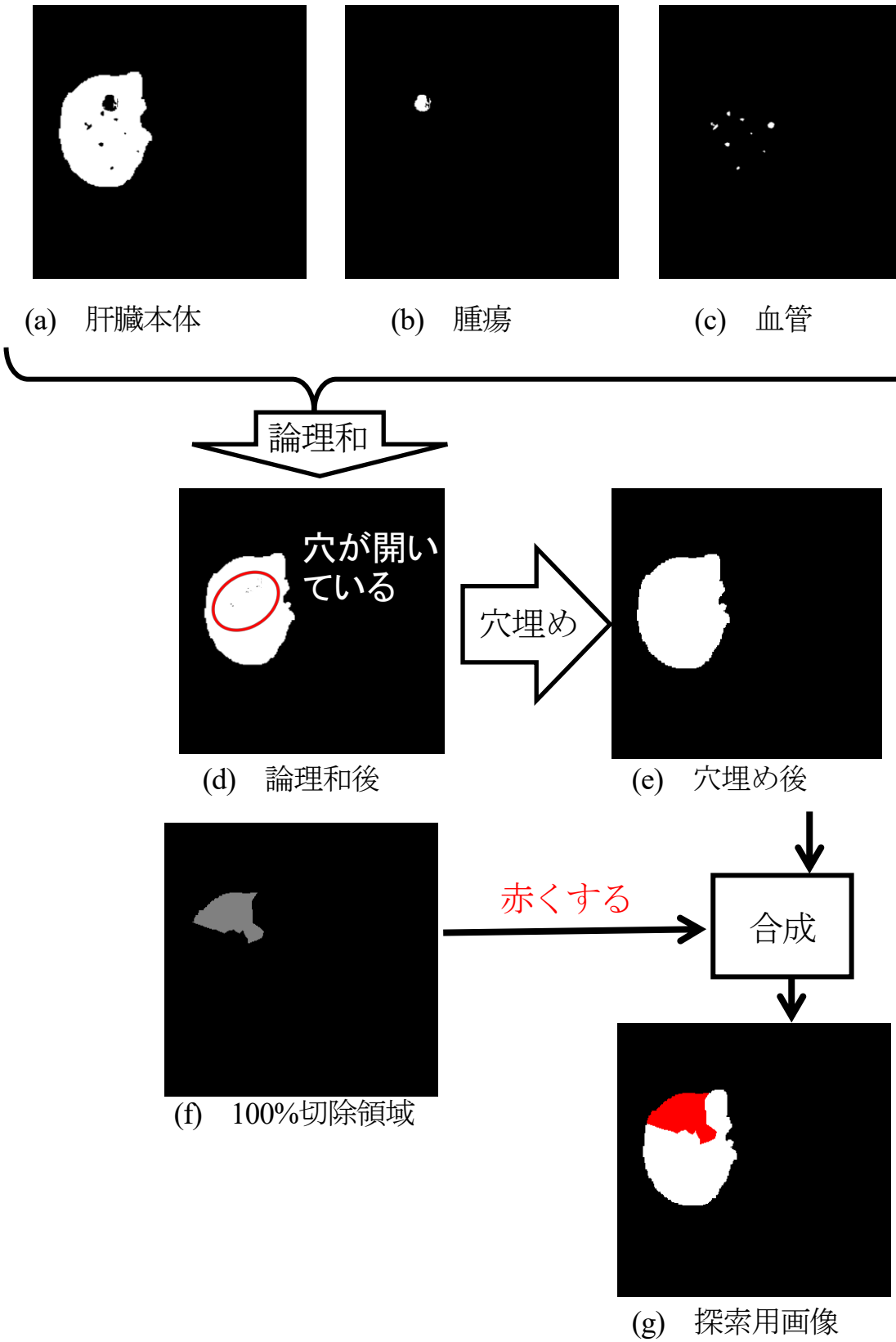


図 3.2　探索用画像の作成

3.4 追加した探索アルゴリズム

本研究で追加した探索アルゴリズムは下記の 3 つである。

手法①　切除体積算出処理の改善
手法②　包含判定高速化
手法③　肝臓と 100%切除領域との位置関係による判定基準の導入

・手法①　切除体積算出処理の改善

本処理は、切除体積算出処理における不必要な処理を減少させることで探索時間を短くするというものである。具体的には 2 つの処理に分けられる。

第 1 に、算出処理の途中中断である。従来の手法では、各パラメータの組み合わせにおける切除体積を正確に算出してから暫定値との大小比較を行っていた。本手法では、切除体積算出中でも暫定値を超えた段階で算出処理を終了させる。

第 2 に、切除領域の判定必要数を減少させることである。従来手法は、100%切除領域、肝臓部分、肝臓でない部分のすべてに対し判定を行い、切除領域の体積を計算している。しかし体積を計算する必要がある場所というのは、本来肝臓部分のみである。なぜなら 100%切除領域は必ずすべて包含している必要があるため、すべての妥当パラメータで含まれている領域であるためである。また肝臓でない部分は、肝臓でないため、放物面に含まれていたとしても、肝臓切除体積には含まれないため、演算を行う必要が無いためだ。

これらの処理の流れを図 3.3 に示す。

・手法②　包含判定高速化

本処理は、100%切除領域を凸多面体に近似することで 100%切除領域がすべて内包しているかの判定を高速化するというものである。

凸包点とは、任意の数の点が与えられた時にその点をすべて内包することが出来る点集合のことである。点数を 8 個とした場合の凸包化の例を図 3.4 に示す。凸包化のアルゴリズムとしては、本研究では参考文献[4]の QuickHull を用いた。

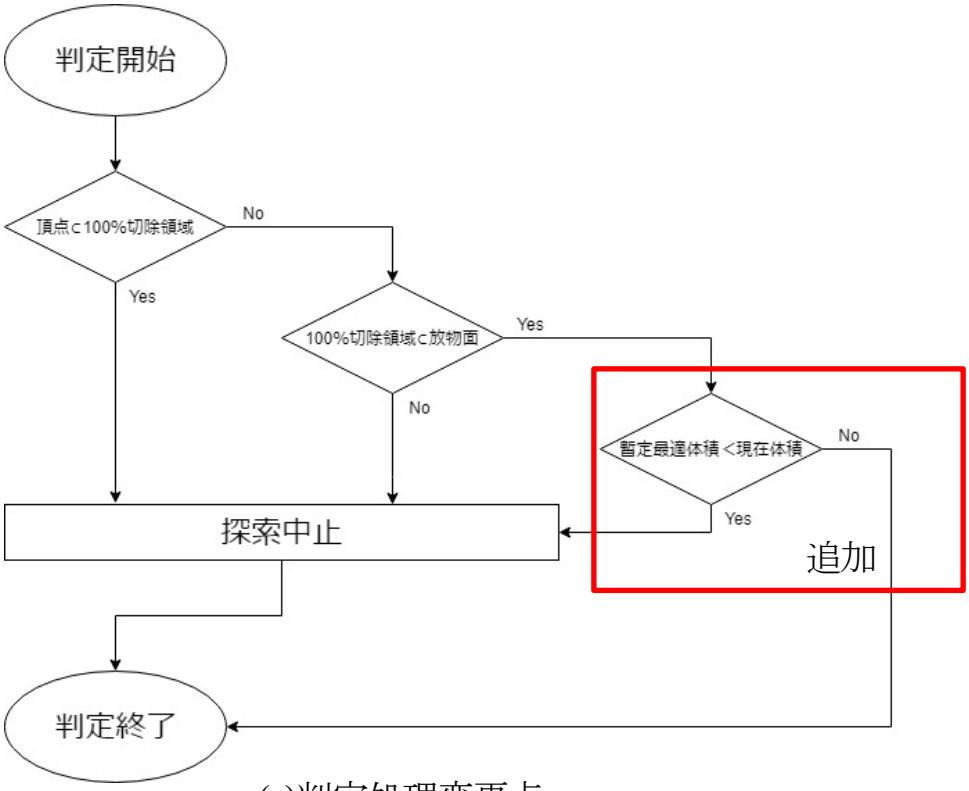
従来手法では、100%切除領域が放物面に完全に包含されているかの判定は、100%切除領域の境界面の点すべてを用いている。提案

手法では、その境界面を凸多面体に変換し判定に用いる点数を減らすことで探索時間短縮を図る。

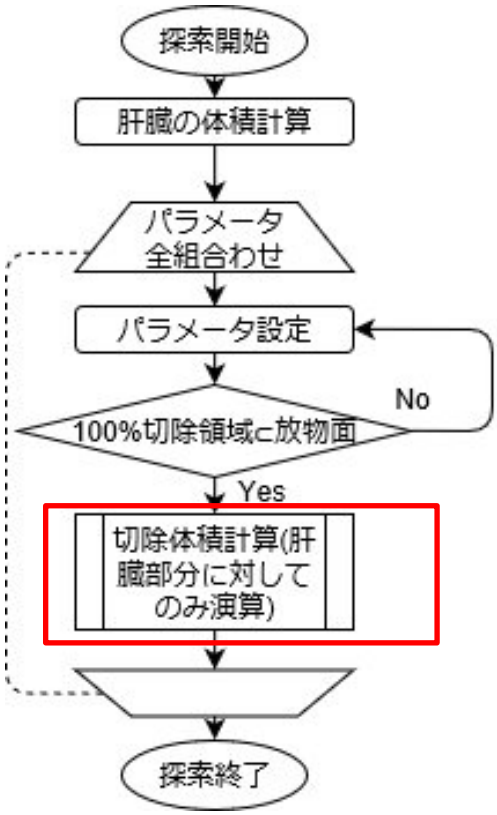
・手法③　肝臓と 100%切除領域の位置関係による判定基準の導入

本処理は、肝臓と 100%切除領域の位置関係から推測出来る、最適解となりやすいパラメータを優先計算することで、探索時間を短くする手法である。

100%切除領域と肝臓の重心の位置関係により、最適な放物面のパラメータをおおよそ推定することが出来る。図3.5の例では、100%切除領域の重心より、肝臓の重心の方が右下にある場合を考える。この場合は、左上が開いた放物面を考えた方が肝臓の切除体積は小さくなる可能性が高くなる。よってそのパラメータを優先的に計算させることで、探索時間を減少させる。

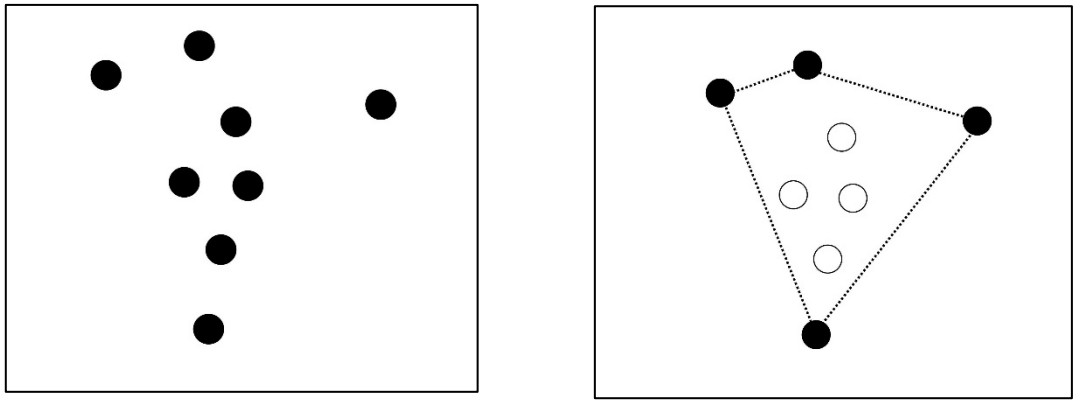


(a)判定処理変更点



(b)処理全体外観

図 3.3　切除体積算出処理の改善による処理の変化



(a)凸包化する前　　(b)凸包化した後

図 3.4　凸包化する前後における判定点の変化

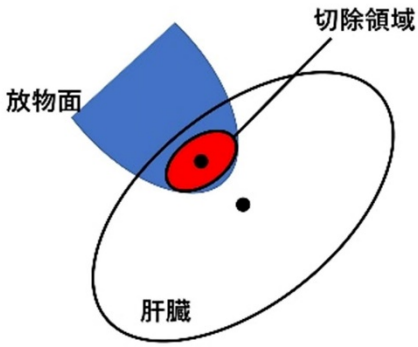


図 3.5　重心位置関係による放物面推定

3.5 むすび

本研究において用いている環境や、その環境における探索のために画像サンプルをどのように加工するかについて述べた。また、探索時間を削減するため、本研究で提案した手法について述べた。

次章では、それらアルゴリズムを用いた場合の探索結果や探索時間の変化について述べる。

第 4 章　探索結果とアルゴリズムの評価

4.1　まえがき

本章では、前章で述べた探索アルゴリズムを用いて実際に探索を行った結果について述べ、従来手法との比較を行う。また、その結果から今回用いた探索アルゴリズムの評価についても述べる。

4.2　探索条件

本研究では下記 3 つの画像を用いた。

・画像 1

比較的肝臓の領域に対する 100%切除領域の体積の割合が大きい画像である(肝臓に対する 100%切除領域の割合：約 17.76%)。画像サイズは 341[Pixel]×341[Pixel]×257[枚]である。なお、図 4.1 に形状を示す。

・画像 2

画像 1 と同様の画像を 170[Pixel]×170[Pixel]×128[枚]に縮小した画像である。

・画像 3

比較的肝臓の領域に対する 100%切除領域の体積の割合が小さい画像である(肝臓に対する 100%切除領域の割合：約 5.76%)。画像サイズは 341[Pixel]×341[Pixel]×257[枚]である。なお、図 4.2 に形状を示す。

探索パラメータは、大きいサイズの画像と小さいサイズの画像によって変化させた。

・小さい画像(画像 2)
　回転の間隔：30[°]
　シフトの間隔：10[Pixel]
　焦点距離の範囲：1/50～8/50

・小さい画像(画像 1 と画像 3)
　回転の間隔：10[°]
　シフトの間隔：5[Pixel]
　焦点距離の範囲：1/50～8/50

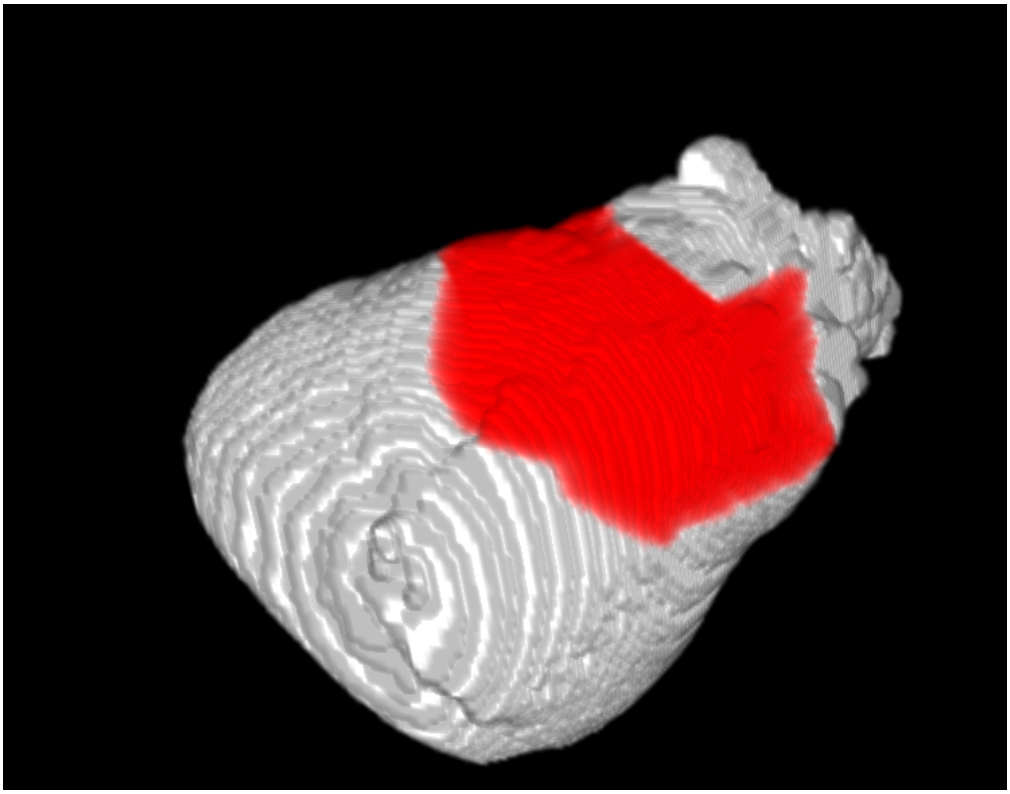


図 4.1　画像 1 と画像 2 における肝臓と 100%切除領域の形状

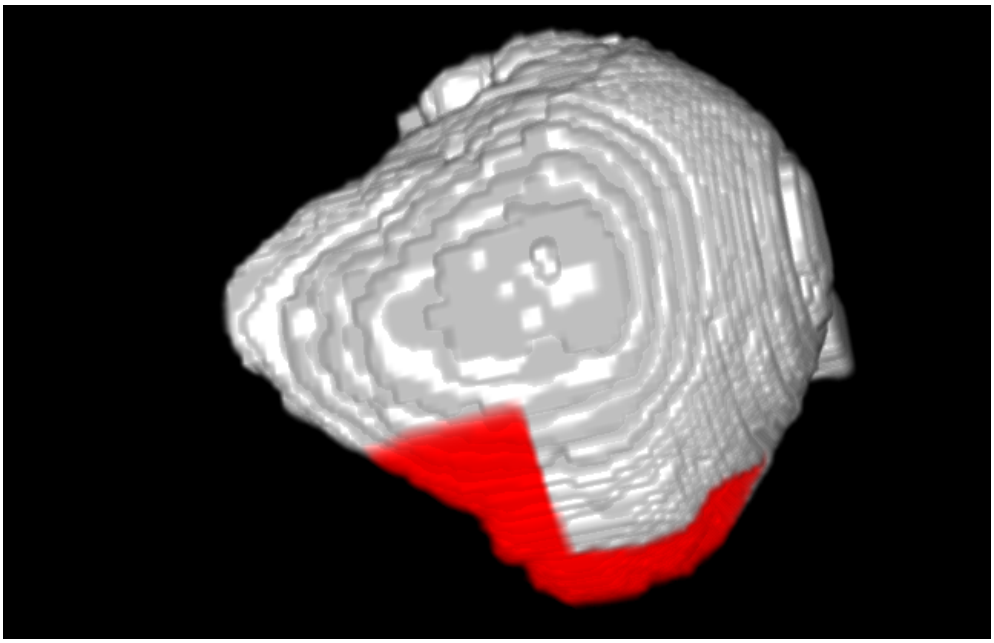


図 4.2　画像 3 における肝臓と 100%切除領域の形状

4.3 探索結果

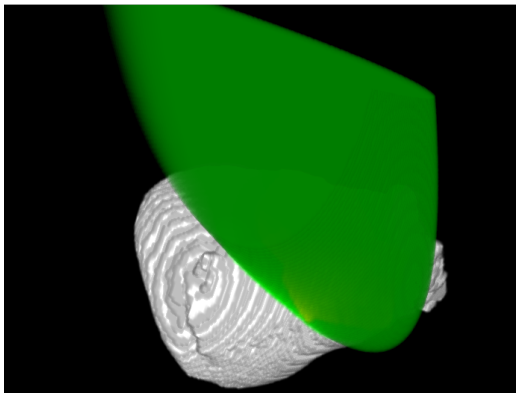
4.2 で述べた探索条件でそれぞれの画像に対し探索を行った結果を表 4.1 に示す。結果は、それぞれの画像に対し探索を開始してから結果が出力されるまでの時間を示している。図 4.3 にそれぞれの探索で得られた最適解の断面図を示す。

探索時間が手法①を加えることで約 98.58%、さらに手法②を加えることで約 99.34%、そしてさらに手法③を加えることで約 99.41%短縮した。つまり、それぞれの探索アルゴリズムが有意義に働いたことが示されている。また、本来の画像の大きさである画像 1 においては、約 1 時間にまで探索時間が短縮したことにより、実用的な探索時間になったといえるだろう。しかし画像 3 のように 100%切除領域が小さい場合、取り得るパラメータが大幅に増えることで探索時間が膨大となる。画像 3 においては探索時間に 3.98 日要しており、実用的ではない。よって、それらの画像に対しても探索時間を短縮する必要がある。

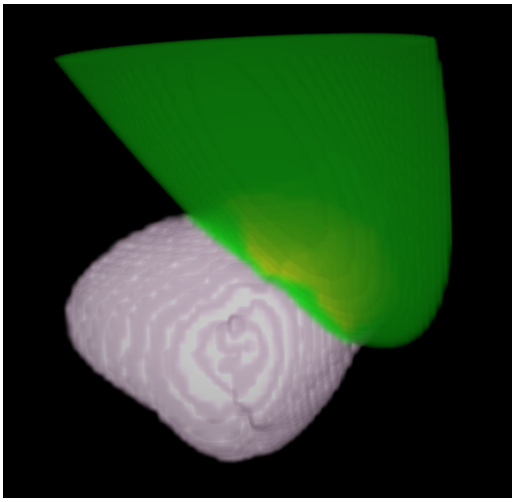
表 4.1　探索結果

	画像 1	画像 2	画像 3
従来手法	測定不可	15,165,211	測定不可
①有②無③無	68,355,783	215,877	測定不可
①有②有③無	4,310,836	100,697	測定不可
①有②有③有	4,089,351	89,672	343,727,004

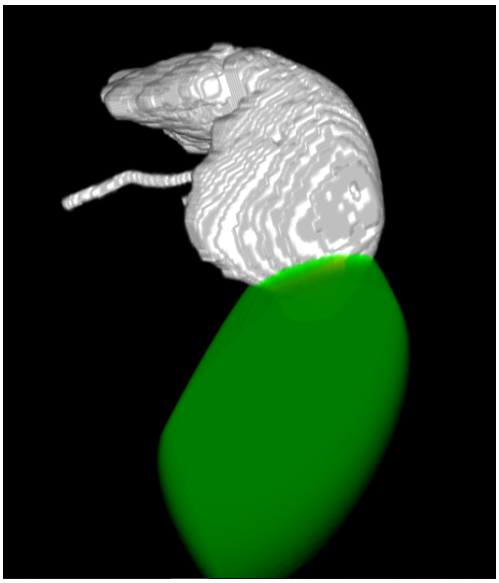
単位はすべて[ms]



(a)画像 1 出力結果



(b)画像 2 出力結果



(c)画像 3 出力結果

図 4.3　探索結果画像

4.4 今後の課題

今後の課題として、探索 3 のように 100%切除領域が小さい画像においては探索時間が膨大になり実用不可能な値になってしまうことが挙げられる。また、解の妥当性について十分な検証が本研究を通しても未だ行えていない。これらを解決する手法について述べる。

まず探索時間の改善手法である。探索時間は、さらに下記の手法を追加することで改善が見込めると考えられる。

・段階的な探索精度の向上

現在探索パラメータを固定し探索を行っている。この処理を、荒いパラメータで探索を行った後にその結果によってパラメータを絞っていき、徐々に細かい探索をさせていく。これにより総じた探索時間を減少させることが期待できる。しかしこの処理により真の最適解を探索することが出来なくなる可能性があり、それに対する処理を検討する必要がある。

・機械学習の利用

今回の研究の目的として、「得られた解の妥当性の検証」が挙げられる。今回のプログラムにおいては、探索の精度パラメータが小さいほどより最適な解が出力され、その解というのは精度パラメータを小さくするほど収束していく。

本研究において、出力された結果を検討し、最適解として、その精度パラメータを用いた解が適切か(収束する画像に近いか)を検討する。そして適切である場合にはそのパラメータを用いて他の画像群に対する 100%切除領域を作成し、それを機械学習させ解を出力することで探索時間をより減少させることが出来る。

・楕円体近似

100%切除領域に対し楕円体近似を行う。そしてその際に得られる楕円体パラメータ(回転・焦点距離・頂点)を用いることで、探索時間の減少を図る。具体的手法や検討事項は次章で述べる。

また、解の妥当性について検証法について述べる。今回の全探索においては、探索の精度を上げるほど精度の良い、つまり妥当性の高い解が出る。しかしどの値まで上げれば妥当性が高くかつ実用的な探索時間が出るか判明していない。

そのため、時間と切除体積を変数とした評価関数を作成し、パラ

メータを変化させていった上でのそれぞれのパラメータにおける探索の評価を行うことを提案する。探索結果は探索精度をあげていくほどある結果に収束していくはずであるから、それを検出するためである。そしてその収束結果を妥当性のある解として用いることが出来るであろう。

4.5 むすび

実際に、本研究で作成した探索アルゴリズムを用いて探索を行った。その結果探索時間を最大約 99.41%減少と、大幅に削減することが出来た。しかし100%切除領域が小さい場合には探索時間に約3.98日かかることもあり、未だ実用的なシステムにはなっていないと考えられる。

今後の課題は本章で述べたとおり、探索時間を削減しかつ解の妥当性の検証手法を検討する必要であることである。

次章では、その課題の前者である「探索時間の削減」に関して提案した手法の 1 つである 100%切除領域の楕円体近似について述べる。

第 5 章　放物面推定効率化のための楕円体近似

5.1　はじめに

本章では、楕円体近似の手法について検討を行う。本研究では、従来手法の放物面 8 個のパラメータでの全探索に対する判定処理を改善した。しかしながら、本研究で使用した第 4 章で示している画像 3 のように 100%切除領域が小さいほど、判定処理対象となるパラメータの組み合わせ数は増大する。得られた凸多面体から最適解に近い放物面を推定することができれば、推定された放物面のパラメータの周辺を中心に探索することで探索範囲を限定することが期待でき、探索時間も減少できる。しかしながら凸多面体から直接放物面を近似することが難しいため、本稿では前段階として凸多面体から楕円体を近似することを検討する。

5.2　楕円体近似の流れ

処理手順は、「軸の選定処理」と、「切除領域を xyz 軸上に置き換えた場合の楕円体近似処理」の 2 つに分けられる。軸の選定処理により近似楕円体の長軸・短軸を演算し、その情報を利用して xyz 軸上での楕円体近似処理を行う。

5.3　軸の選定処理

軸の選定処理では、主成分分析を用いて凸多面体から楕円体の軸を求める。主成分分析とは、n 次元(本研究の場合は 3 次元)のデータ群に対し以下の手順を行うことで、特徴的な軸を求める処理である。

1. データ群のうち、分散が最大となる軸を求める。
2. これまで出力された計算結果の軸すべてに直交し、かつ分散が最大となる軸を求める。
3. 2 の処理を繰り返すことで、n 個の軸を計算する。

出力される軸の順に、第 1 主成分、第 2 主成分、第 3 主成分と呼ばれ、n 次元のデータ群においては第 n 主成分まで出力される。今回の CT 画像は 3 次元となるため、主成分分析の解としては第 3 主

成分までが出力される。第 1 主成分はデータ群の中で最も分散が大きい軸を表している為、楕円体近似の楕円体における長軸として考えることが出来る。第 2 主成分は第 1 主成分に、第 3 主成分は第 1 主成分と第 2 主成分に直交しながら、分散が最も大きくなる軸のことである。つまりこれらのデータを用いることで、楕円体における短軸を考えることが出来る。

この処理を OpenCV[5]を用いて画像 1 に対し実装した結果、下記のような結果が出力された。

第 1 主成分　(-0.292 , -0.547 , 0.784)

第 2 主成分　(0.951 , -0.251 , 0.179)

第 3 主成分　(0.0987 , 0.798 , 0.594)

これら値の整合性検証は今後行っていく必要があるが、さきほど述べたとおり、これらの値が長軸・短軸を示しているといえる。

5.4 軸上における楕円体近似

切除領域を xyz 軸上に置き換えた場合の楕円体近似処理について述べる。5.3 で述べた手法を用いて取得された長軸・短軸の情報と重心の情報を利用することで、軸を基準として回転している 100%切除領域を軸上に置き換えることが出来る(図 5.1)。なお、今回は z 軸上に長軸、x 軸と y 軸上に短軸が当てはまるように置き換える。その場合モデル関数は式(5.1)のようになる。

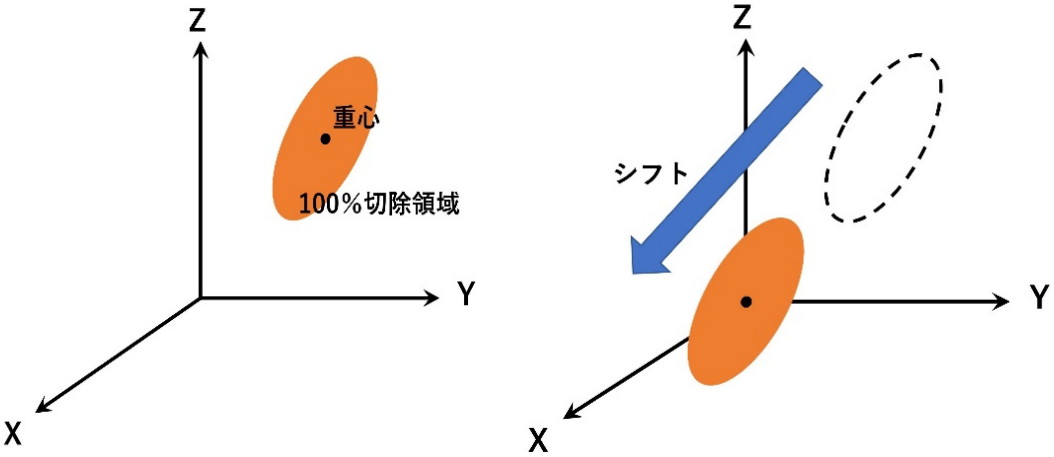
$$\mathrm{A}(\mathrm{x}^2 + y^2) + Bz^2 = 1 \qquad (5.1)$$

この軸上に置き換えた 100%切除領域において限定した近似処理を行う。近似の手法としては最小二乗法を用いる。最小二乗法とは、ある誤差を含む測定値に対し特定のモデル関数を仮定したとき、その誤差の二乗の和を最小にしすることで最も確からしい関係式を求める演算である。

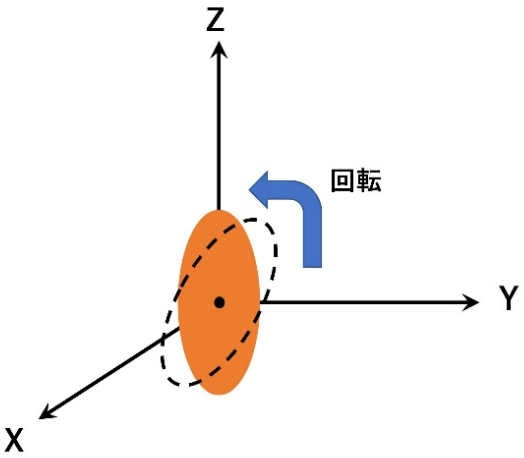
今回の場合においては、式(5.2)が最小となる A と B、つまり式(5.3)を満たす A と B を求めれば良い。

$$\sum_{i=1}^{n}\{A({x_i}^2+{y_i}^2)+B{z_i}^2-1\}^2=F(A,B) \qquad (5.2)$$

$$\frac{\partial F(A,B)}{\partial A}=0 \text{ かつ } \frac{\partial F(A,B)}{\partial B}=0 \qquad (5.3)$$



(a)元の物体位置　　(b)シフト後の物体位置



(c)回転後の物体位置

図 5.1　主成分分析による軸上への置き換え

5.5 楕円体近似により取得できるパラメータ

前項までで述べてきたことから、100%切除領域の下記パラメータが取得できている。

・回転角

・重心

・xyz 軸上に移動させたときにおける、楕円体の関数

またこれらのパラメータより、モデル関数式(5.1)の A と B を用いることで焦点距離は式(5.4)のように表せる。

$$\text{焦点距離} = \sqrt{A^2 - B^2} \qquad (5.4)$$

5.6 楕円体パラメータの応用

本項では、前項で述べた楕円体パラメータを実際の探索でどのように応用することが出来るかについて述べる。

まず、焦点距離を利用した探索短縮方法についてである。100%切除領域より小さい焦点距離をもつ放物面は最適解として適さない可能性が高い。なぜなら図 5.2 のようにそのような焦点距離をもつ放物面は、100%切除領域すべてを包含することが出来ない、もし出来たとしても肝臓の別除領域が大きくなることが想定される為である。

そして、軸情報を用いた探索短縮方法についてである。このためには下記 3 つの情報を用いる。

①肝臓の長軸

②100%切除領域の長軸

③肝臓と 100%切除領域それぞれの重心をつなぐ線分のベクトル

これらのベクトルは放物面推定を行う上で下記の様に 4 つに場合分けを行う。

[1] ベクトル①と②が垂直かつベクトル①と③が垂直

この場合は図 5.3(a)のように、100%切除領域の長軸と水平の軸を持った放物面が最適解をもつ可能性が高い。なぜなら、それ以外の放物面の場合はそれより切除体積が大きくなってしまうことが想定される為である。

[2] ベクトル①と②が垂直かつベクトル①と③が水平

この場合は図 5.3(b)のように、100%切除領域の短軸と水平の軸を持った放物面が最適解をもつ可能性が高い。なぜなら、それ以外の放物面の場合はそれより切除体積が大きくなってしまうことが想定される為である。

[3] ベクトル①と②が水平かつベクトル①と③が垂直

この場合は図 5.3(c)のように、100%切除領域の短軸と水平の軸を持った放物面が最適解をもつ可能性が高い。なぜなら、それ以外の放物面の場合はそれより切除体積が大きくなってしまうことが想定される為である。

[4] ベクトル①と②が水平かつベクトル①と③が水平

この場合は図 5.3(d)のように、100%切除領域の長軸と水平の軸を持った放物面が最適解をもつ可能性が高い。なぜなら、それ以外の放物面の場合はそれより切除体積が大きくなってしまうことが想定される為である。

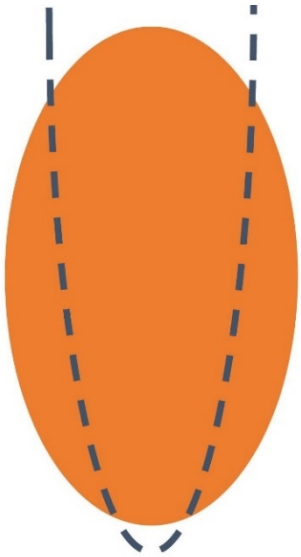
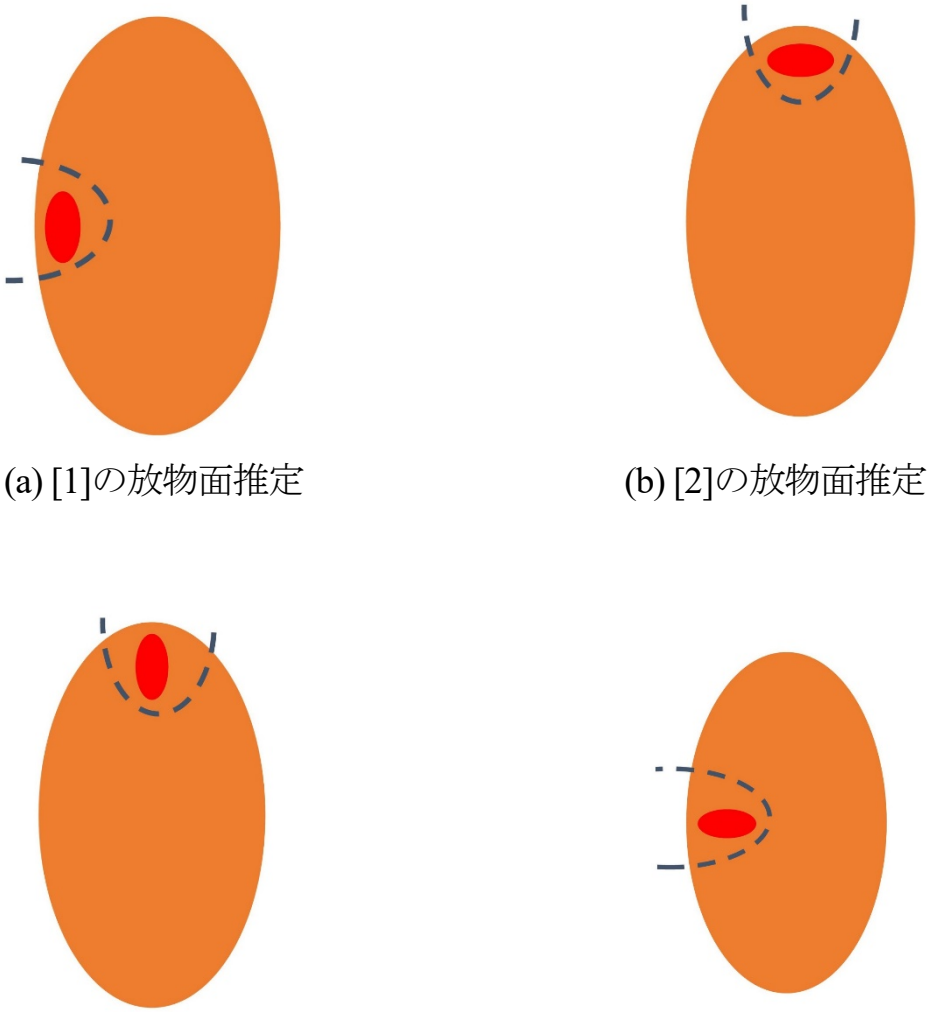


図 5.2　焦点距離による探索短縮方法



(a) [1]の放物面推定　　(b) [2]の放物面推定

(c) [3]の放物面推定　　(d) [4]の放物面推定

図 5.3　軸情報による探索短縮方法

5.7 むすび

本章では、前章までで問題に上がった「100%切除領域が小さい画像に対しては、探索時間が膨大になってしまう」という問題について解決する手法として、楕円体近似やその具体的利用法について述べた。しかし実装を行い検証することは今後の課題である。

第 6 章　おわりに

本研究では、肝臓外科手術における最適部分切除領域推定のための全探索について、従来手法[3]の課題であった探索時間を減少させるためのアルゴリズムを提案し探索を行った。その結果、探索時間が約 99.41%減少するなど大幅に探索時間が減少した。しかし、100%切除領域が小さい画像に対し探索を行うと探索時間が約 3.98 日要することもあり、未だに医療機関で実用出来るシステムにはなっていないと考えられる。

今後の課題は前章までに述べてきたとおり、実際に医療機関で使用できる精度を保った上でどのような画像でも探索時間を実用的な物にしていくことや、解の妥当性検証についての手法を検討していくことである。

参考文献

[1]鈴木健明, 張山昌論, 亀山充隆, 下田貢, 窪田敬一, “肝臓手術における実用的な制約条件を考慮した最適切除領域推定”, 計測自動制御学会東北支部第 295 回研究集会, 295-1(2015)
[2]岡田萌, 張山昌論, 亀山充隆, 下田貢, “腫瘍領域情報に基づく肝臓切除容量最小化のための門脈切除点計算”, 計測自動制御学会東北支部第 279 回研究集会, 279-3(2013)
[3]小野翔平：平成 29 年度卒業論文「肝臓外科手術における最適部分切除領域推定に関する研究」
[4] A Robust 3D Convex Hull Algorithm in Java, https://github.com/Quickhull3d/quickhull3d (2018/10/15)
[5] OpenCV library, https://opencv.org (2019/1/11)

謝辞

本研究の遂行をはじめとして、研究発表の資料作成、ないしは大学入試の面接練習、そして本論文をまとめるにあたって、指導教員の小山工業高等専門学校電気電子創造工学科小林准教授に熱心でかつ丁寧なご指導を承りました。1 年間のご指導で多大な知識を得ることができ、実践的な経験をすることができました。ここに感謝の意を表します。

また、研究発表の時をはじめその他多くの機会で大変有意義なご意見やご提案をいただいた、電気電子創造工学科の教員皆様や学生たちに感謝いたします。

付録

(1)ImageJ のクラス

ImageJ には独自のクラスが多く用いられている。本章ではそのクラスのうちから本プログラムで用いているものについて解説する。

IJ.log(" ")　ログを表示する。

ImagePlus　ImageJ 上に表示されている画像について情報を取得する。

- getSize　枚数を取得
- getWidth　横幅を取得
- getHeight 高さを取得
- updateAndDraw　画像を新たなデータに基づいて描写し直す

他の事項に関しては付録(2)で述べている。

(2)4 章の探索におけるプログラムコードとその解説

①必要なライブラリの導入

・java.util.ArrayList　　可変配列を作成するため

・org.opencv.core.Core　主成分分析を行うため
org.opencv.core.CvType
org.opencv.core.Mat

・ij.ImagePlus　　ImageJ を使用するため
ij.ImageStack
ij.WindowManager
ij.gui.GenericDialog
ij.plugin.filter.PlugInFilter
ij.process.ImageProcessor

② OpenCV の展開

OpenCV を用いる際に必須の文章。この文により OpenCV を展開する。

③ ImageJ セットアップメソッド

プログラム実行時に 1 回のみ実行される。ImageJ に入力した画像をどのような形式で扱うかを定義する。

今回は DOES_RGB のため、RGB 各 8bit のカラー画像を扱うということが定義されている。RGB であるため取得したピクセルの色情報は 24bit であるが、プログラム上では int 型に格納している。このようにすると上 8bit が余るが、その部分には全ビットに対し 1 が格納される。そのため、例えば黒ピクセルの色情報は”11111111_00000000_00000000_00000000”と表現される。

④ ダイアログ表示

この部分では、ダイアログを表示しユーザから数値を取得し探索の精度値を変更する。ImageJ のクラスである GenericDialog を用いている。

⑤画像情報取得と体積計算

ImageJ のクラスである ImageProcessor を用いることで画像情報を 1 枚ずつ取り出す。また、その処理と並行して肝臓部分や 100%切除領域の体積を計算する。また、切除体積処理に必要な部分(白い肝臓部分)を配列(cal_vol)に格納する。

⑥shutu 配列情報

本配列は、最適な解が更新され次第その解に相当するパラメータを格納しておくための配列である。つまりあるパラメータにおいて shutu[8]よりも小さい切除体積が出力された場合、そのパラメータを shutu[0]から shutu[7]へ格納する。

⑦凸包化

この部分では、100%切除領域に対し凸包化を行っている。この処理は、100%切除領域の配列への格納、100%切除領域の凸包化、本プログラム様式に合致する様に凸包化した後の点を格納、という 3 手順に分けられる。

(7.1) 配列への格納

今回用いたクラスの仕様に合致する形で配列へ座標情報を格納する必要がある。今回用いるクラス(参考文献[4])は、Point3d という独自の型を用いている。最終的には Point3d の固定配列の形で QuickHull3D の build 関数の中へ入れる必要がある。そのため、最初に ArrayList で Point3d という型の points_before を作成した。その中に値を格納し、その後その ArrayList を通常の配列へ変更する。

(7.2)100%切除領域の凸包化

先ほど作成した通常の配列を QuickHull3D の中の build 関数へ入れることで凸包化を行っている。

本文での hull.build(points)がこれに当たる。

(7.3)凸包化した後の点を格納

QuickHull3D の中の getVertices を用いることで、凸包化後の点情報を入手することが出来る。本プログラムの仕様に合致する様に配列 setujo に対し x→y→z の順番で凸包化後の座標を格納した。

⑧主成分分析

OpenCV を用いて主成分を行うためには、OpenCV の Mat 型変数へ値を格納する必要がある。また、結果もすべて Mat 型で帰ってくるため Mat 型の配列を多数作成している。Mat 型の変数を作成する際には、配列の大きさとデータタイプを指定する必要がある。今回は入出力データの仕様を考え、データタイプは CV_16UC1 を採用した。これは、配列内それぞれの値が 16bit 符号なし 1 チャンネルのデータであることを意味する。また、格納する際は Mat 型の put というクラスを使用した。

Core.PCACompute2(cal_part,mean_cal, vector_cal,eigen_cal)の部分が主成分分析を行っている文章である。mean_cal の位置に入力データそれぞれの平均値、vector_cal の部分に主成分分析の結果の軸、eigen_cal の部分に主成分分析の際に必要な固有値が Mat 型で格納される。

分析後のデータ格納では 3 次元配列を用いている。これは、主成分分析の Mat 型の get 関数を用いると double[]型の値が返ってくるため、配列に 1 つ冗長性が必要であるためである。

⑨重心の関係性格納

ここでは、「肝臓と 100%切除領域との位置関係による判定基準の導入」で用いる、肝臓と 100%切除領域の重心関係を演算している。演算結果は part という int 型の変数へ格納する。変数 part は初期値が 0 でありこの演算を通すと 0 以外の値になるはずなので、この演算後の値が 0 であった場合は何かしらエラーが発生したと考えられる。そのエラーを検出するための文も入れている。

⑩最適切除領域探索

この部分では最適な切除領域を探索する。本研究で「探索時間を削減」と言っていたことはつまり「この部分の処理時間を削減する」ということと同義である。探索は for 文を用いることで全探索を実装している。

提案手法③は青枠の部分である。青枠の部分を下記の様に変更することで探索③をなくすことが出来る。

```
for(shix=jux-50;shix<=jux+50;shix+=shiftpos){
        for(shiy=juy-50;shiy<=juy+50;shiy+=shiftpos){
                for(shiz=juz-50;shiz<=juz+50;shiz+=shiftpos){
```

なお、シフトであるが大きい画像の時は 100%切除領域の重心座標から±50[Pixel]、小さい画像の時は±25[Pixel]の範囲に頂点をもつ放物面に対して全探索を行っている。

より良い解が検出されるたび shutu の値を更新することによって探索を行う。また、更新されたときのみそのパラメータの値をすべて表示する。

⑪最適解描写

最適解に相当する放物面を描写する部分である。⑩での探索結果であるパラメータを用いることで放物面を作成し、「放物面かつ 100%切除領域」の部分は黄色、「それ以外の放物面の部分」は緑色で描写を行う。

```
import java.util.ArrayList;
import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import ij.IJ;
import ij.ImagePlus;
import ij.ImageStack;
import ij.WindowManager;
import ij.gui.GenericDialog;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;

public class My_Plugin implements PlugInFilter {
    static {
        System.loadLibrary("opencv_java400");
    }

    protected ImagePlus image;
    public static final int black=0xff000000;          //色の宣言
    public static final int white=0xffffffff;
    public static final int red=0xffff0000;
    public static final int lightgreen=0x00ff00;
    public static final int deepgreen=0x008000;
    public double[] shutu = new double[9];//出力用 parameter
    public boolean setujof=true;
    public double shoteny=1/5;//曲率
    public double shotenz=5/5;//曲率
    public int[][][] ori;//すべてのピクセルをコピーする用

    //回転角
    public double kakux=60*(Math.PI)/180;
    public double kakuy=60*(Math.PI)/180;
    public double kakuz=0*(Math.PI)/180;
    //public static final double enhosei=0.01;
    //腫瘍中心からの最短距離
```

①

②

```
public double dist=0;
public int[] point=new int[3];
public static int kakupos=10;//角度の間隔
public static int shiftpos=5;//平行移動の間隔
public static int shotenrate=50;//焦点の倍率

public int setup(String arg, ImagePlus imp) {
        image = imp;
        return DOES_ALL;
}                                                    ③

boolean showdialog() {
        int[] wList = WindowManager.getIDList();
        if (wList==null) {
                error();
                return false;
        }
        String[] titles = new String[wList.length];
        for (int i=0; i<wList.length; i++) {
        ImagePlus imp = WindowManager.getImage(wList[i]);
                titles[i] = imp!=null?imp.getTitle():"";
        }
GenericDialog gd = new GenericDialog("RegionGrowingSphere");
        gd.addChoice("OrgStack:", titles, titles[0]);
        gd.addNumericField("角度の精度:", kakupos, 0 );
        gd.addNumericField("平衡移動の精度:", shiftpos, 0 );
        gd.addNumericField("焦点の倍率:", shotenrate, 0 );
        gd.showDialog();
        if (gd.wasCanceled()) return false;

        int index1 = gd.getNextChoiceIndex();
        image = WindowManager.getImage(wList[index1]);
        kakupos = (int)gd.getNextNumber();
        shiftpos = (int)gd.getNextNumber();
        shotenrate = (int)gd.getNextNumber();
```

```
            IJ.log("凸包なし、すぐ抜けあり");
            IJ.log("初期条件");
            IJ.log("角度の精度は："+kakupos);
            IJ.log("平衡移動の精度は："+shiftpos);
            IJ.log("焦点の倍率は："+shotenrate);
            IJ.log(" ");
            return true;
    }                                                    ④

    void error() {    //エラー表示
            IJ.showMessage("RegionGrowingSphere", "This command
requires one stack.¥n");
    }

    public void run(ImageProcessor ip) {
            IJ.log("最新版");

            if(!showdialog()) {
                    return;
            }

            ImagePlus imp=IJ.getImage();
            ImageStack ist=imp.getStack();
            int slice=ist.getSize();
            int width =imp.getWidth();
            int height = imp.getHeight();

            ori=new int[slice][height][width];        //初期化
            int[] vector=new int[3];

            int shix=59;
            int shiy=75;
            int shiz=56;
            int shiftx=0;
            int shifty=0;
```

```
int shiftz=0;
int shix_start=0;
int shiy_start=0;
int shiz_start=0;

int cnt=0;//切除領域の体積（肝臓と緑の楕円体が重なった部分）
int cntvol=0;//肝臓の体積
//肝臓の腫瘍中心座標
int jux=0;
int juy=0;
int juz=0;
int jux_cal=0;
int juy_cal=0;
int juz_cal=0;
double calx=0;
double caly=0;
double calz=0;
double shux=0;
double shuy=0;
double shuz=0;
int cntshu=0;//肝臓の腫瘍の体積
double kakudox=0;
double kakudoy=0;
double kakudoz=0;

int chotenx=50;
int choteny=50;
int chotenz=50;
int shiftposx=shiftpos;
int shiftposy=shiftpos;
int shiftposz=shiftpos;

int part=0;
```

```
int kakuposx=kakupos;
int kakuposy=kakupos;
int kakuposz=kakupos;
double kakudo=0;
double kakudo_kankei=0;

ArrayList<Double> setujox=new ArrayList<Double>();
ArrayList<Double> setujoy=new ArrayList<Double>();
ArrayList<Double> setujoz=new ArrayList<Double>();
ArrayList<Integer> setujo=new ArrayList<Integer>();
//肝臓の領域を格納
ArrayList<Integer> cal_vol = new ArrayList<Integer>();
int point1=0;

long start_time=System.currentTimeMillis();

//体積計算
for(int i=1;i<=slice;i++){
        ImageProcessor ipa=ist.getProcessor(i);

        int[] pix1=(int[])ipa.getPixels();
        for(int pixnum=0;pixnum<pix1.length;pixnum++){
                int col1=pix1[pixnum];
                int enx=pixnum%width;
                int eny=(int)(pixnum/width);
                int enz=i-1;

                //腫瘍の体積と中心座標を求める
                ori[i-1][eny][enx]=col1;

                if(col1!=black){
                        cntvol++;
                        jux_cal+=enx;
                        juy_cal+=eny;
                        juz_cal+=enz;
```

```
                                    if(col1==white) {//白のとき
                                        cal_vol.add(enx);
                                        cal_vol.add(eny);
                                        cal_vol.add(enz);
                                    }
                                    if(col1==red){//赤のとき
                                        jux+=enx;
                                        juy+=eny;
                                        juz+=enz;
                                        cntshu++;

                                        if(col1!=pix1[pixnum-
1]||col1!=pix1[pixnum+1]||col1!=pix1[pixnum-
width]||col1!=pix1[pixnum+width]){
                                        point1++;
                                        }

                                    }
                                }
                            }
                        }
                        long end_time=System.currentTimeMillis();
                        IJ.log("jurai houkatu POINT: "+point1);                    ⑤
                        IJ.log("jurai houkatu TIME: "+(-(start_time -
end_time))+"ms");



                        jux=jux/cntshu;
                        juy=juy/cntshu;
                        juz=juz/cntshu;
                        jux_cal=jux_cal/cntvol;
                        juy_cal=juy_cal/cntvol;
                        juz_cal=juz_cal/cntvol;
                        IJ.log("Liver Volume"+cntvol);
```

```
                IJ.log("PIXELS: "+width*height*slice);
                IJ.log("Tumor Volume"+cntshu);
                IJ.log("腫瘍 Center x:"+jux+" y:"+juy+" z:"+juz);
                IJ.log("肝臓 Center x : "+jux_cal+" y : "+juy_cal+" z :
"+juz_cal);
                vector[0]=jux-jux_cal;
                vector[1]=juy-juy_cal;
                vector[2]=juz-juz_cal;

                shutu[0]=shix;
                shutu[1]=shiy;
                shutu[2]=shiz;
                shutu[3]=kakux;
                shutu[4]=kakuy;
                shutu[5]=kakuz;
                shutu[6]=shoteny;
                shutu[7]=shotenz;
                shutu[8]=cntvol;                                    ⑥

                double progress=0;
                long start_prog=System.currentTimeMillis();
                long end_prog=System.currentTimeMillis();
                long start_judge=System.currentTimeMillis();
                long end_judge=System.currentTimeMillis();
                long judgetime=0;
                long start_vol=System.currentTimeMillis();
                long end_vol=System.currentTimeMillis();
                long vol_time=0;
                int pixnum=0;

                //convex hull start
                for(int i=1;i<slice-1;i++){        //境界面取得
                        for(int j=1;j<height-1;j++) {
                                for(int k=1;k<width-1;k++) {
                                        pixnum=j*width+k;
```

```
                        if((ori[i][j][k]==red)){
                            setujoz.add((double)i);
                            setujoy.add((double)j);
                            setujox.add((double)k);
                        }
                    }
                }
            }
```

⑦

```
            ArrayList<Point3d> points_before =new ArrayList<Point3d>();
            Mat cal_part=new Mat(setujox.size(),3,CvType.CV_16UC1);
            Mat vector_cal=new Mat();
            Mat eigen_cal=new Mat();
            Mat mean_cal=new Mat();
            Mat shu_part=new Mat(setujox.size()+(cal_vol.size()/3),3,CvType.CV_16UC1);
            Mat vector_shu=new Mat();
            Mat eigen_shu=new Mat();
            Mat mean_shu=new Mat();

            for(int i=0;i<setujox.size();i++) {
                cal_part.put(i, 0, setujox.get(i));
                cal_part.put(i, 1, setujoy.get(i));
                cal_part.put(i, 2, setujoz.get(i));
                shu_part.put(i, 0, setujox.get(i));
                shu_part.put(i, 1, setujoy.get(i));
                shu_part.put(i, 2, setujoz.get(i));
            }

            for(int i=0;i<(cal_vol.size()/3);i++) {
                cal_part.put(i+setujox.size(), 0, cal_vol.get(3*i));
                cal_part.put(i+setujox.size(), 1, cal_vol.get(3*i+1));
                cal_part.put(i+setujox.size(), 2, cal_vol.get(3*i+2));
```

```
                }

                Core.PCACompute2(cal_part,mean_cal,
vector_cal,eigen_cal);  //主成分分析
                Core.PCACompute2(shu_part,mean_shu,
vector_shu,eigen_shu); //主成分分析



                //分析後のデータ格納
                double[][][]                    meandata_cal=new
double[(int)mean_cal.rows()][(int)mean_cal.cols()][(int)mean_cal.channels
()];
                for(int i=0;i<mean_cal.rows();i++) {    //平均値の格納
                        for(int j=0;j<mean_cal.cols();j++) {
                                meandata_cal[i][j]=mean_cal.get(i, j);
                        }
                }
                double[][][]                    vectordata_cal=new
double[(int)vector_cal.rows()][(int)vector_cal.cols()][(int)vector_cal.chann
els()];
                for(int i=0;i<vector_cal.rows();i++) {
                        for(int j=0;j<vector_cal.cols();j++) {
                                vectordata_cal[i][j]=vector_cal.get(i, j);
                        }
                }
                double[][][]                    eigendata_cal=new
double[(int)eigen_cal.rows()][(int)eigen_cal.cols()][(int)eigen_cal.channels
()];
                for(int i=0;i<eigen_cal.rows();i++) {
                        for(int j=0;j<eigen_cal.cols();j++) {
                                eigendata_cal[i][j]=eigen_cal.get(i, j);
                        }
                }

                double[][][]                    meandata_shu=new
```

```
double[(int)mean_shu.rows()][(int)mean_shu.cols()][(int)mean_shu.channe
ls()];
                for(int i=0;i<mean_shu.rows();i++) {    //平均値の格納
                        for(int j=0;j<mean_shu.cols();j++) {
                                meandata_shu[i][j]=mean_shu.get(i, j);
                        }
                }
                double[][][]                            vectordata_shu=new
double[(int)vector_shu.rows()][(int)vector_shu.cols()][(int)vector_shu.chan
nels()];
                for(int i=0;i<vector_shu.rows();i++) {
                        for(int j=0;j<vector_shu.cols();j++) {
                                vectordata_shu[i][j]=vector_shu.get(i, j);
                        }
                }
                double[][][]                            eigendata_shu=new
double[(int)eigen_shu.rows()][(int)eigen_shu.cols()][(int)eigen_shu.channe
ls()];
                for(int i=0;i<eigen_shu.rows();i++) {
                        for(int j=0;j<eigen_shu.cols();j++) {
                                eigendata_shu[i][j]=eigen_shu.get(i, j);
                        }
                }
                //分析後のデータ格納完了

                IJ.log("腫瘍(cal)の PCA 結果");
                for(int i=0;i<vector_cal.rows();i++) {
                        for(int j=0;j<vector_cal.cols();j++) {
                                IJ.log(i+","+j+" "+vectordata_cal[i][j][0]);
                        }
                }


        calx=Math.atan(vectordata_cal[0][1][0]/vectordata_cal[0][0][0]);
```

```
caly=Math.atan(Math.sqrt(vectordata_cal[2][0][0]*vectordata_cal[
2][0][0]+vectordata_cal[2][1][0]*vectordata_cal[2][1][0])/vectordata_cal[2
][2][0]);

calz=Math.atan(vectordata_cal[2][1][0]/vectordata_cal[2][0][0]);
        calx=calx*180/Math.PI;
        caly=caly*180/Math.PI;
        calz=calz*180/Math.PI;


        IJ.log("肝臓(shu)の PCA 結果");
        for(int i=0;i<vector_shu.rows();i++) {
                for(int j=0;j<vector_shu.cols();j++) {
                        IJ.log(i+","+j+" "+vectordata_shu[i][j][0]);
                }
        }

shux=Math.atan(vectordata_shu[0][1][0]/vectordata_shu[0][0][0]);

shuy=Math.atan(Math.sqrt(vectordata_shu[2][0][0]*vectordata_sh
u[2][0][0]+vectordata_shu[2][1][0]*vectordata_shu[2][1][0])/vectordata_sh
u[2][2][0]);

shuz=Math.atan(vectordata_shu[2][1][0]/vectordata_shu[2][0][0]);
        shux=shux*180/Math.PI;
        shuy=shuy*180/Math.PI;
        shuz=shuz*180/Math.PI;                                    ⑧

        //腫瘍と肝臓の関係性を計算
        if(jux>jux_cal) {
                chotenx=chotenx*(-1);
                shiftposx=shiftposx*(-1);
                if(juy>juy_cal) {
                        choteny=choteny*(-1);
                        shiftposy=shiftposy*(-1);
```

```
                    if(juz>juz_cal) {
                            chotenz=chotenz*(-1);
                            shiftposz=shiftposz*(-1);
                            part=1;
                    }else if(juz<=juz_cal) {
                            chotenz=chotenz*1;
                            shiftposz=shiftposz*1;
                            part=2;
                    }
            }else if(juy<=juy_cal) {
                    choteny=choteny*1;
                    shiftposy=shiftposy*1;
                    if(juz>juz_cal) {
                            chotenz=chotenz*(-1);
                            shiftposz=shiftposz*(-1);
                            part=3;
                    }else if(juz<=juz_cal) {
                            chotenz=chotenz*1;
                            shiftposz=shiftposz*1;
                            part=4;
                    }
            }
    }else if(jux<=jux_cal) {
            chotenx=chotenx*1;
            shiftposx=shiftposx*1;
            if(juy>juy_cal) {
                    choteny=choteny*(-1);
                    shiftposy=shiftposy*(-1);
                    if(juz>juz_cal) {
                            chotenz=chotenz*(-1);
                            shiftposz=shiftposz*(-1);
                            part=5;
                    }else if(juz<=juz_cal) {
                            chotenz=chotenz*1;
                            shiftposz=shiftposz*1;
```

```
                                        part=6;
                                }
                        }else if(juy<=juy_cal)  {
                                choteny=choteny*1;
                                shiftposy=shiftposy*1;
                                if(juz>juz_cal) {
                                        chotenz=chotenz*(-1);
                                        shiftposz=shiftposz*(-1);
                                        part=7;
                                }else if(juz<=juz_cal) {
                                        chotenz=chotenz*1;
                                        shiftposz=shiftposz*1;
                                        part=8;
                                }
                        }
                }

                IJ.log("シフトするピクセル数は  x:"+chotenx+"
y:"+choteny+" z:"+chotenz);

                if(part==0) {
                        IJ.log("関係性が計算できていません");
                        return;
                }
                IJ.log("関係性は"+part+"です");
```

⑨

```
                for(int i=0;i<setujox.size();i++) {
points_before.add(new Point3d(setujox.get(i),setujoy.get(i),setujoz.get(i)));
                }

                //配列の変更(可変から普通に)
        int size=points_before.size();
        Point3d[] points = points_before.toArray(new Point3d[size]);
                QuickHull3D hull = new QuickHull3D();
```

```
            hull.build(points);      //凸包化

            setujox.clear();         //不要配列削除
            setujoy.clear();
            setujoz.clear();
            points_before.clear();
            Point3d[] vertices = hull.getVertices();
            double x;
            double y;
            double z;
            for (int i = 0; i < vertices.length; i++) {//本 program 様式へ変更
                Point3d pnt = vertices[i];
                x=pnt.x;
                y=pnt.y;
                z=pnt.z;
                setujo.add((int)x);
                setujo.add((int)y);
                setujo.add((int)z);
            }

                end_time=System.currentTimeMillis();
                IJ.log("CONVEX POINT: "+setujo.size()/3);
        IJ.log("CONVEX TIME: "+(-(start_time - end_time))+"ms");
                //凸包化終了                                          ⑦

                IJ.log("肝臓の角度は  x:"+calx+" y:"+caly+" z:"+calz);
                IJ.log("腫瘍の角度は  x:"+shux+" y:"+shuy+" z:"+shuz);
                IJ.log("探索する頂点は  x:"+chotenx+"  y:"+choteny+"
z:"+chotenz+"から始まります");

                start_time=System.currentTimeMillis();
                shix_start=jux+chotenx+shiftposx;
                shiy_start=juy+choteny+shiftposy;
                shiz_start=juz+chotenz+shiftposz;
```

```
		IJ.log("この地点から探索を始めます x:"+shix_start+"
y:"+shiy_start+" z:"+shiz_start);
		IJ.log("大きさは 3 次元：:"+ori[0][0].length+" 2 次元
"+ori[0].length+" 1 次元:"+ori.length);
		IJ.log("　　探　　索　　回　　数　　は
z:"+Math.abs(chotenz)*2/Math.abs(shiftposz)+"
y:"+Math.abs(choteny)*2/Math.abs(shiftposy)+"
x:"+Math.abs(chotenx)*2/Math.abs(shiftposx));

		//定数宣言
		long progsize = 3456649728L;

		shix=shix_start;
	for(shiftx=0;shiftx<=Math.abs(chotenx)*2/Math.abs(shiftposx);shif
tx++) {
			shix-=shiftposx;
			shiy=shiy_start;

	for(shifty=0;shifty<=Math.abs(choteny)*2/Math.abs(shiftposy);shif
ty++) {
				shiy-=shiftposy;
				shiz=shiz_start;

	for(shiftz=0;shiftz<=Math.abs(chotenz)*2/Math.abs(shiftposz);shif
tz++) {
					shiz-=shiftposz;

	for(kakudox=0;kakudox<180;kakudox+=kakupos){
		kakux=kakudox*(Math.PI)/180;
	for(kakudoy=0;kakudoy<360;kakudoy+=kakupos){
		kakuy=kakudoy*(Math.PI)/180;
	for(kakudoz=0;kakudoz<360;kakudoz+=kakupos){
		kakuz=kakudoz*(Math.PI)/180;

	//頂点が切除領域内部にないか確認
```

```
		if(ori[shiz][shiy][shix]==red){
			setujof=false;
			progress+=16;
		}else{
			setujof=true;
		}

		if(setujof==true){//頂点が切除領域内部にない場合
		for(int shoy=1;shoy<=10;shoy*=2){
			shoteny=(double)shoy/shotenrate;
		for(int shoz=1;shoz<=10;shoz*=2){
			shotenz=(double)shoz/shotenrate;
				progress++;

			boolean houkatuf=true;
			start_judge=System.currentTimeMillis();

			for(int houkatu=0;houkatu<setujo.size();houkatu+=3){
				double enx=setujo.get(houkatu);
				double eny=setujo.get(houkatu+1);
				double enz=setujo.get(houkatu+2);
				enx-=shix;
				eny-=shiy;
				enz-=shiz;
				double tranx=enx;
				double trany=eny;
				double tranz=enz;

			enx=tranx*Math.cos(kakuz)*Math.cos(kakuy)+trany*(-
Math.cos(kakux)*Math.sin(kakuz)+Math.sin(kakux)*Math.sin(kakuy)*Ma
th.cos(kakuz))+tranz*(Math.sin(kakux)*Math.sin(kakuz)+Math.cos(kakux)
*Math.sin(kakuy)*Math.cos(kakuz));
		eny=tranx*Math.cos(kakuy)*Math.sin(kakuz)+trany*(Math.cos(ka
kux)*Math.cos(kakuz)+Math.sin(kakux)*Math.sin(kakuy)*Math.sin(kakuz
))+tranz*(-
```

```
Math.sin(kakux)*Math.cos(kakuz)+Math.cos(kakux)*Math.sin(kakuy)*M
ath.sin(kakuz));
        enz=-
tranx*Math.sin(kakuy)+trany*Math.sin(kakux)*Math.cos(kakuy)+tranz*
Math.cos(kakux)*Math.cos(kakuy);

        if((double)(-(eny*eny*shoteny+enz*enz*shotenz))<enx){
                houkatuf=false;
                break;
            }
        }
            end_judge=System.currentTimeMillis();

            judgetime+=(end_judge - start_judge);
             start_vol=System.currentTimeMillis();

            if(houkatuf==true){
                int counter=0;
                boolean cntfrag=true;

                for(int cal_vol_num=0;
cal_vol_num<cal_vol.size();cal_vol_num+=3){
                  double enx=(double)cal_vol.get(cal_vol_num);
                  double eny=(double)cal_vol.get(cal_vol_num+1);
                  double enz=(double)cal_vol.get(cal_vol_num+2);
                  enx-=shix;
                  eny-=shiy;
                  enz-=shiz;
                  tranx=enx;
                  double trany=eny;
                  double tranz=enz;
        enx=tranx*Math.cos(kakuz)*Math.cos(kakuy)+trany*(-
Math.cos(kakux)*Math.sin(kakuz)+Math.sin(kakux)*Math.sin(kakuy)*Ma
th.cos(kakuz))+tranz*(Math.sin(kakux)*Math.sin(kakuz)+Math.cos(kakux)
*Math.sin(kakuy)*Math.cos(kakuz));
```

```
        eny=tranx*Math.cos(kakuy)*Math.sin(kakuz)+trany*(Math.cos(ka
kux)*Math.cos(kakuz)+Math.sin(kakux)*Math.sin(kakuy)*Math.sin(kakuz
))+tranz*(-
Math.sin(kakux)*Math.cos(kakuz)+Math.cos(kakux)*Math.sin(kakuy)*M
ath.sin(kakuz));
        enz=-
tranx*Math.sin(kakuy)+trany*Math.sin(kakux)*Math.cos(kakuy)+tranz*
Math.cos(kakux)*Math.cos(kakuy);


        if((double)(-(eny*eny*shoteny+enz*enz*shotenz))>=(enx)){
                                counter++;
                                if(counter>shutu[8]) {
                                        cntfrag=false;
                                }
                        }
                        if(!cntfrag) {
                                        break;
                        }
                }
                        if(counter<shutu[8]){
                                shutu[0]=shix;
                                shutu[1]=shiy;
                                shutu[2]=shiz;
                                shutu[3]=kakudox;
                                shutu[4]=kakudoy;
                                shutu[5]=kakudoz;
                                shutu[6]=shoy;
                                shutu[7]=shoz;
                                shutu[8]=counter;

                                IJ.log("progress:"+(progress/progsize));
                                end_prog=System.currentTimeMillis();
                        IJ.log("between time:"+(end_prog - start_prog));
```

```
                        start_prog=System.currentTimeMillis();
                        IJ.log("shiftx :"+shutu[0]);//x2 ついてる
                        IJ.log("shifty :"+shutu[1]);
                        IJ.log("shiftz :"+shutu[2]);
                        IJ.log("kakudox :"+shutu[3]);
                        IJ.log("kakudoy :"+shutu[4]);
                        IJ.log("kakudoz :"+shutu[5])
                    }
                }else{
                    break;
                }
                end_vol=System.currentTimeMillis();
                vol_time+=(end_vol - start_vol);
            }
                                                                    }
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }                                                    ⑩

                kakux=shutu[3]*(Math.PI)/180;
                kakuy=shutu[4]*(Math.PI)/180;
                kakuz=shutu[5]*(Math.PI)/180;

                shoteny=shutu[6]/shotenrate;
                shotenz=shutu[7]/shotenrate;

                for(int i=1;i<=slice;i++){
                    ImageProcessor ipa1=ist.getProcessor(i);

                    int[] pix1=(int[])ipa1.getPixels();
```

```
for(pixnum=0;pixnum<pix1.length;pixnum++){
//一次元配列の要素数から x,y 座標を計算
        double enx=pixnum%width;
        double eny=(int)(pixnum/width);
        double enz=i-1;
        int conx=pixnum%width;
        int cony=(int)(pixnum/width);
        int conz=i-1;

        int col1=pix1[pixnum];
        //シフト
        enx-=shutu[0];
        eny-=shutu[1];
        enz-=shutu[2];

        double tranx=enx;
        double trany=eny;
        double tranz=enz;

        //ロールピッチヨーで回転

    enx=tranx*Math.cos(kakuz)*Math.cos(kakuy)+trany*(-
Math.cos(kakux)*Math.sin(kakuz)+Math.sin(kakux)*Math.sin(kakuy)*Ma
th.cos(kakuz))+tranz*(Math.sin(kakux)*Math.sin(kakuz)+Math.cos(kakux)
*Math.sin(kakuy)*Math.cos(kakuz));

    eny=tranx*Math.cos(kakuy)*Math.sin(kakuz)+trany*(Math.cos(ka
kux)*Math.cos(kakuz)+Math.sin(kakux)*Math.sin(kakuy)*Math.sin(kakuz
))+tranz*(-
Math.sin(kakux)*Math.cos(kakuz)+Math.cos(kakux)*Math.sin(kakuy)*M
ath.sin(kakuz));
        enz=-
tranx*Math.sin(kakuy)+trany*Math.sin(kakux)*Math.cos(kakuy)+tranz*
Math.cos(kakux)*Math.cos(kakuy);
```

```
                                //放物線の回転体を作成する
                                if((double)(-
(eny*eny*shoteny+enz*enz*shotenz))>=(enx)){
                                        if(col1!=black){
                                                if(col1==0xffff0000){
        pix1[pixnum]+=0x00ff00;
                                                }else if(col1==white){
        pix1[pixnum]=0x00ff00;
                                                }else {
                                        IJ.log("NOT white or red position");
                                        IJ.log("x:"+conx+" y:"+cony+" z:"+conz);
                                                }
                                                cnt+=1;
                                        }else{
                                                pix1[pixnum]+=0x008000;
                                        }
                                }
    ipa1.set((int)(pixnum%width), (int)(pixnum/width), (int)pix1[pixnum]);
                        }
                }                                                    ⑪

                //IJ.log("CUT Volume: "+cnt);
                IJ.log("shiftx :"+shutu[0]);//x2 ついてる
                IJ.log("shifty :"+shutu[1]);
                IJ.log("shiftz :"+shutu[2]);
                IJ.log("kakudox :"+shutu[3]);
                IJ.log("kakudoy :"+shutu[4]);
                IJ.log("kakudoz :"+shutu[5]);
                IJ.log("shoteny :"+shutu[6]+"/"+shotenrate);
                IJ.log("shotenz :"+shutu[7]+"/"+shotenrate);
                IJ.log("cut volme :"+(shutu[8]+(double)cntshu));
                IJ.log("cnt :"+cnt);
                end_time=System.currentTimeMillis();
                IJ.log("search time:"+(end_time - start_time)+"ms");
                IJ.log("judge time:"+judgetime+"ms");
```

```
                IJ.log("Calculation time"+vol_time+"ms");

                imp.updateAndDraw();
        }
}
```