円検出の流れ確認
現在までの撮影状況

2019/11/13 谷

円検出の流れ

- 今の段階ではあまりgood とbad を満足に識別できているとはいえない。

- まだ円検出の精度が十分でないと感じたので、細かな修正を行った。

- 結果として、検出の精度は向上したが、それによるgood/bad 識別能力の向上には至らなかった。

穴検出の流れ

① グレースケールで画像を読み込む

② 誤検出を防ぐため、ガウシアンフィルタをかける

③ ret, th = cv2.threshold(gray, pixth=150, 255, cv2.THRESH_BINARY) で画像を二値化

④ edges = cv2.Canny(th, minVal, maxVal) でエッジ検出

⑤ circles = cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, 1, 500, param1=15, param2=vote, minRadius, maxRadius) で円検出

流れ③画像の二値化

ret, th = cv2.threshold(gray, pixth=150, 255, cv2.THRESH_BINARY)

- pixth: ピクセルの明るさに関するスレッショルド (0~255)

- pixth 以上の明るさのピクセルをすべて 255（白）にする処理 → pixth の決定に自由度あり

流れ④エッジ検出

edges = cv2.Canny(th, minVal, maxVal) でエッジ検出

- minVal、maxVal: エッジと判断するためのスレッショルド（今は固定）

- 本来はエッジ化処理も含めて円検出のメソッドがやってくれるが、実際に得られるエッジ画像を確認するため、前処理としてエッジ検出を行う。

流れ⑤円検出

circles = cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, 1, 500, param1=15, param2=vote, minRadius, maxRadius) で円検出

- 500：隣接する円との最小距離。大きい値にしておけば画像中に少ない数の円を検出する。小さい値だと、一つの円に対し複数の検出候補を出力してしまう。

- param1: エッジ検出のためのパラメータ（大きい方）。今回は予めエッジ化処理をしているので、適当。

- param2: ハフ変換の際の最低投票数。この値が小さいほど検出しやすくなる（〜誤検出が多くなる）。param2 の決定に自由度あり

- minRadius、maxRadius: 検出円半径の領域

vote 数を変化させる

- 画像の二値化に関するpixth
- 円検出の vote 数

に自由度があると考える（今までは定数としていた）。

まずは pixth =150 と固定し、vote 数を変化させた。

vote 数の処理

- vote 数が大きすぎると、円検出できない。

- vote 数が小さいと複数の候補が発生する。この場合、隣接する円を検出するモードでは非常に多くの円を検出してしまう。一つの円検出モードでは、ハフ変換の際の画像のスキャン方向に依存して、一番はやく見つけた円が出力される。

- → vote 数をまずは大きい値に設定し、そこから徐々に下げて検出できる vote 数をさがす。

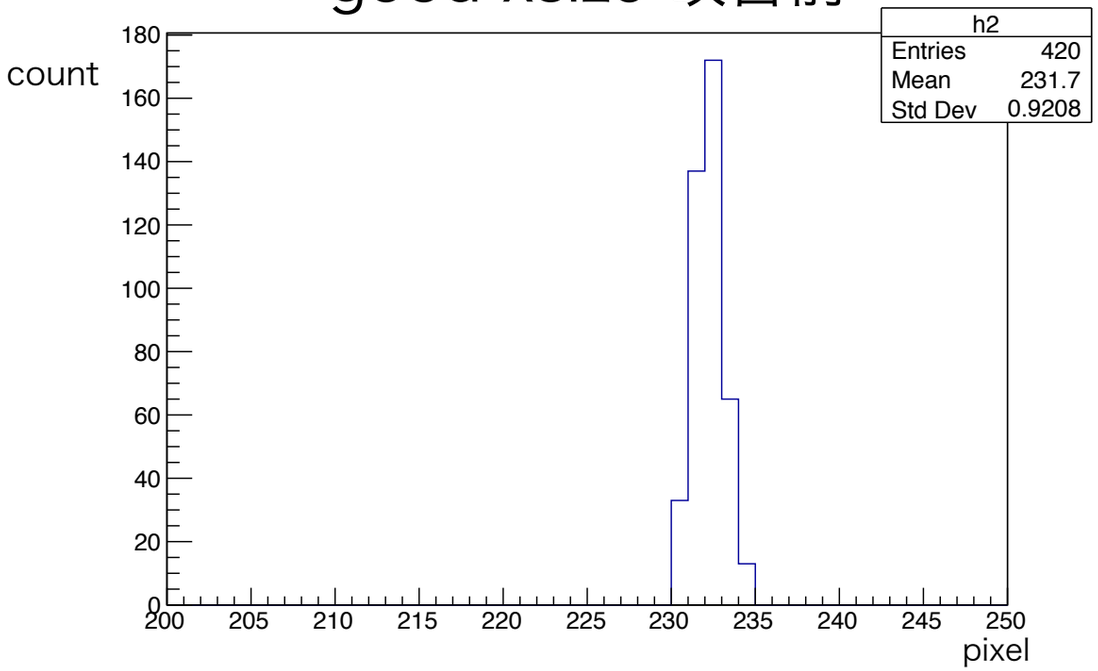
- good キューブの円検出成功例をみて、尤もらしい値に半径を制限することで、更に誤検出の割合を減らす。

改善前後の比較

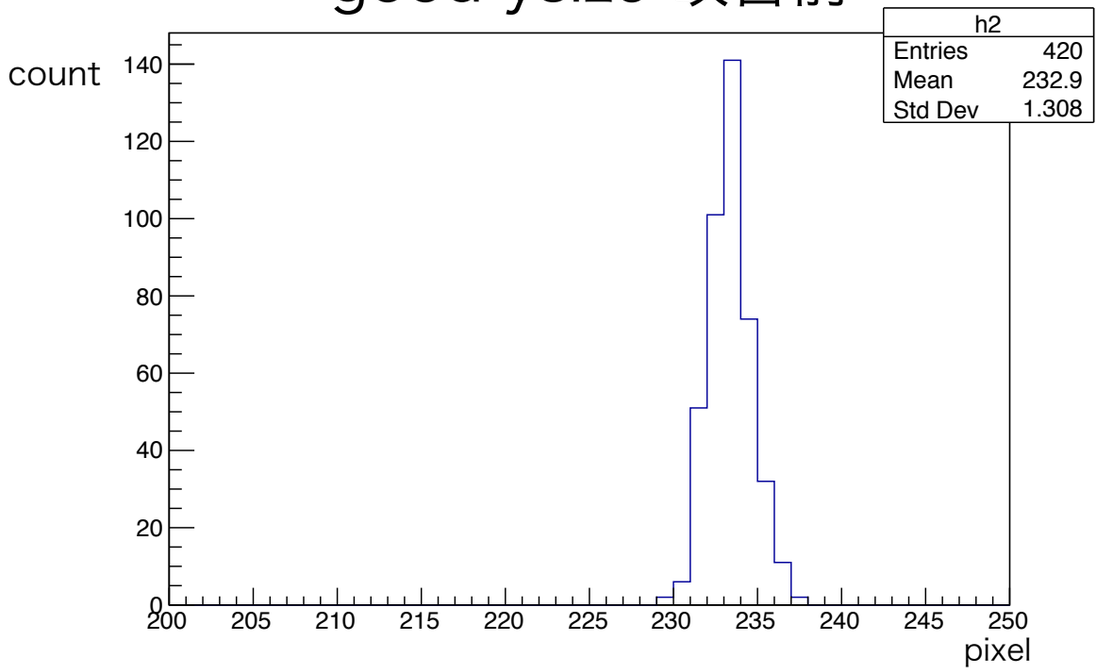
- 以前撮影した同じ画像ファイルを用いた。
- カメラ軸 = 穴中心、レンズ–キューブ間隔 35 cm
- good、 bad それぞれ７０個ずつ撮影
- 室内照明off、卓上ライト利用

サイズに関する比較

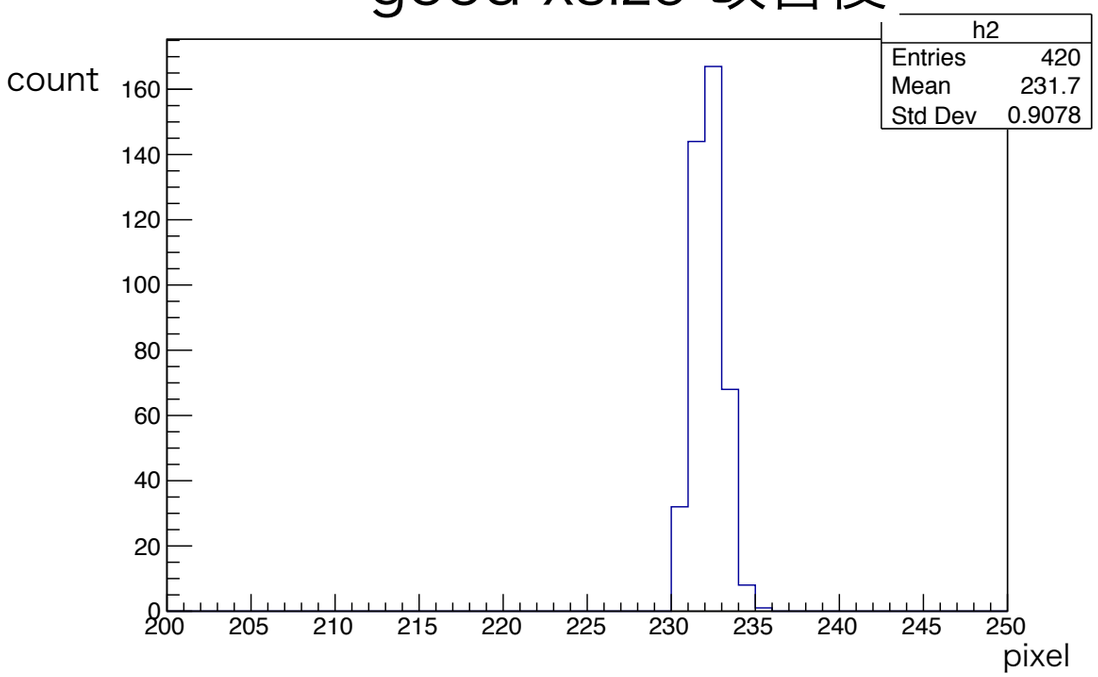
good xsize 改善前



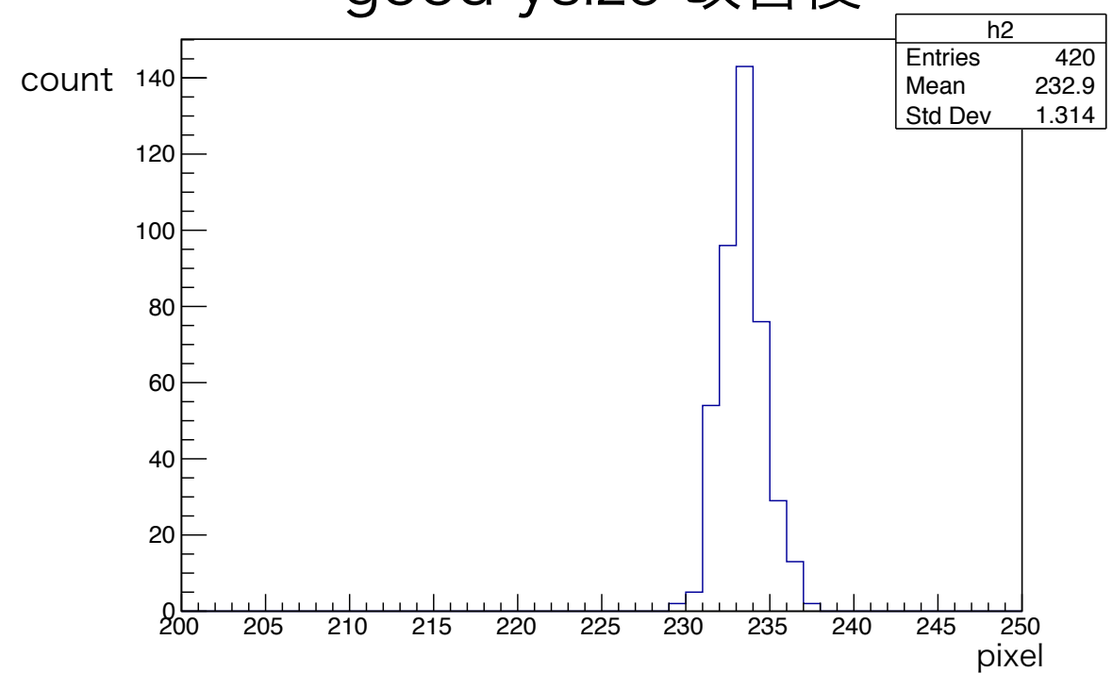
good ysize 改善前



good xsize 改善後

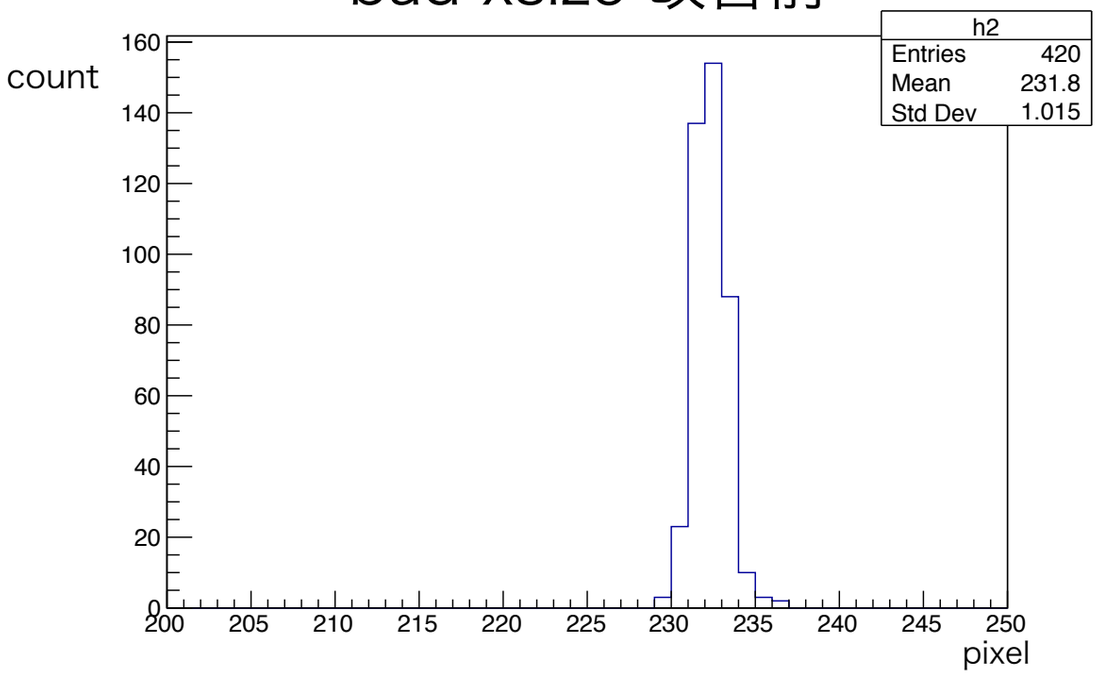


good ysize 改善後

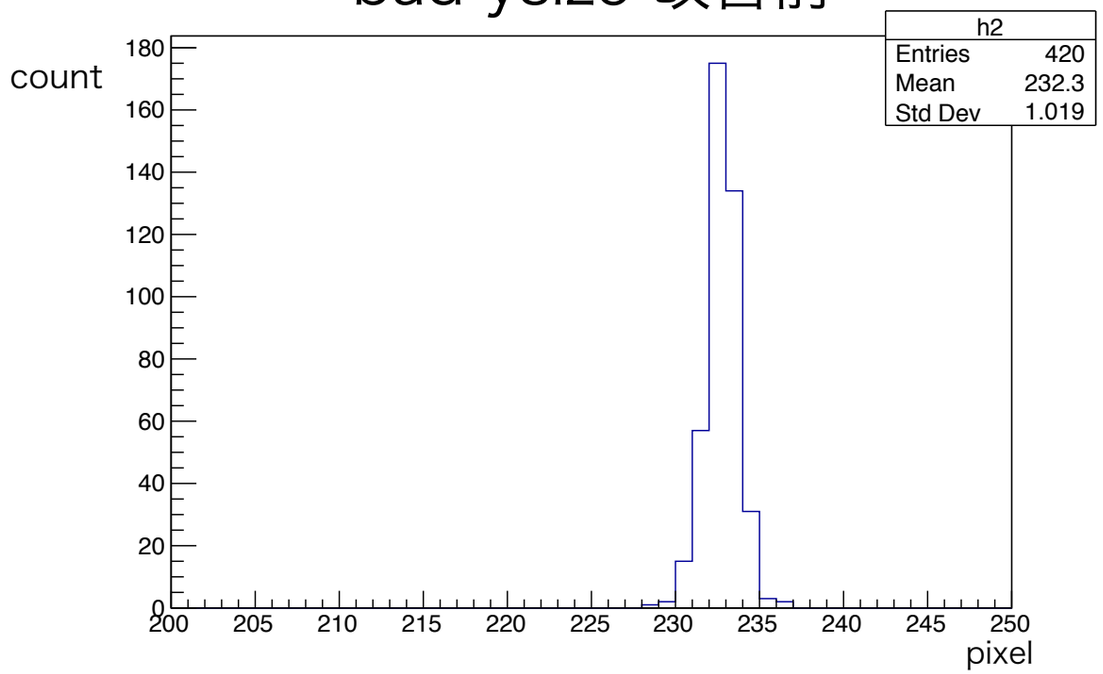


当然だが、サイズに関してはほとんど変化なし

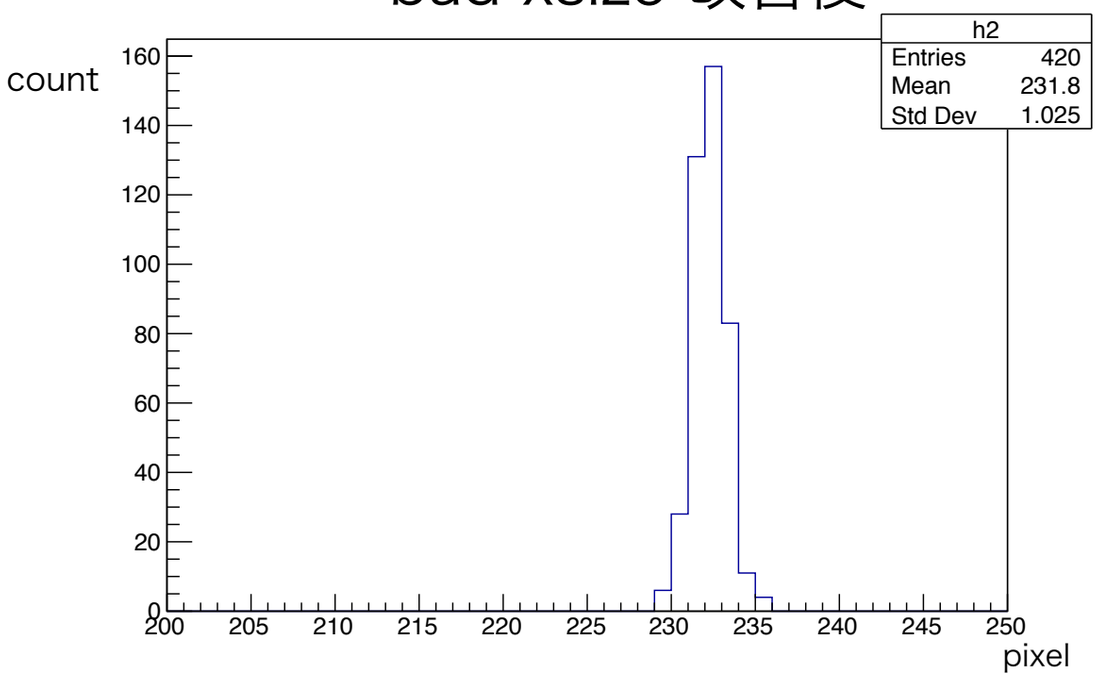
bad xsize 改善前



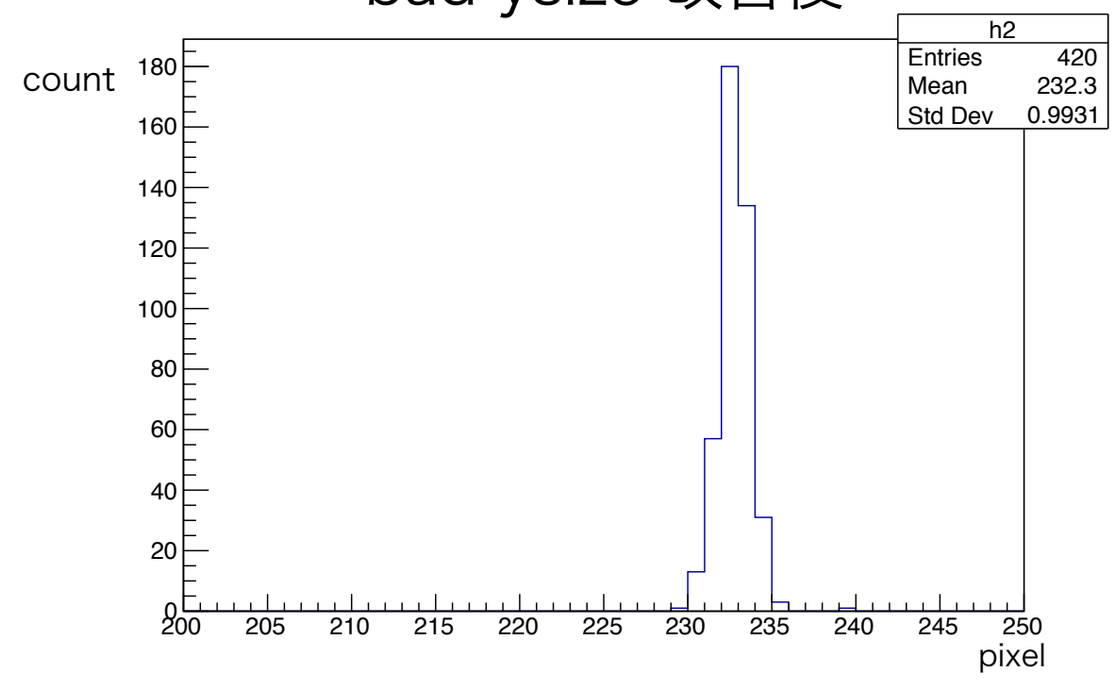
bad ysize 改善前



bad xsize 改善後

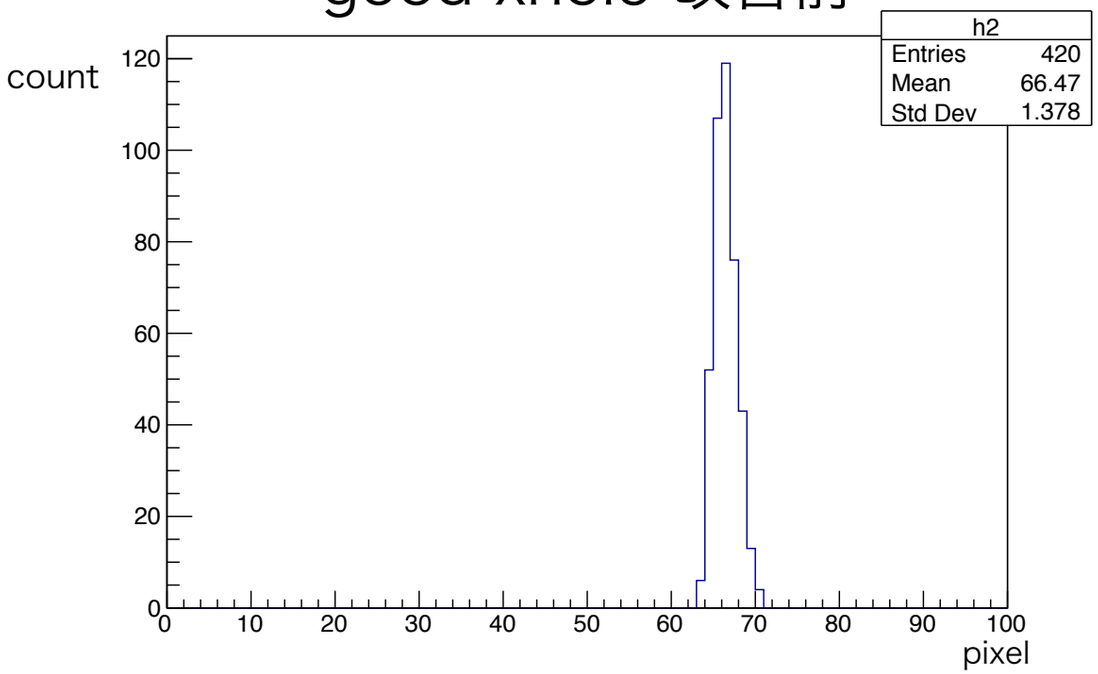


bad ysize 改善後

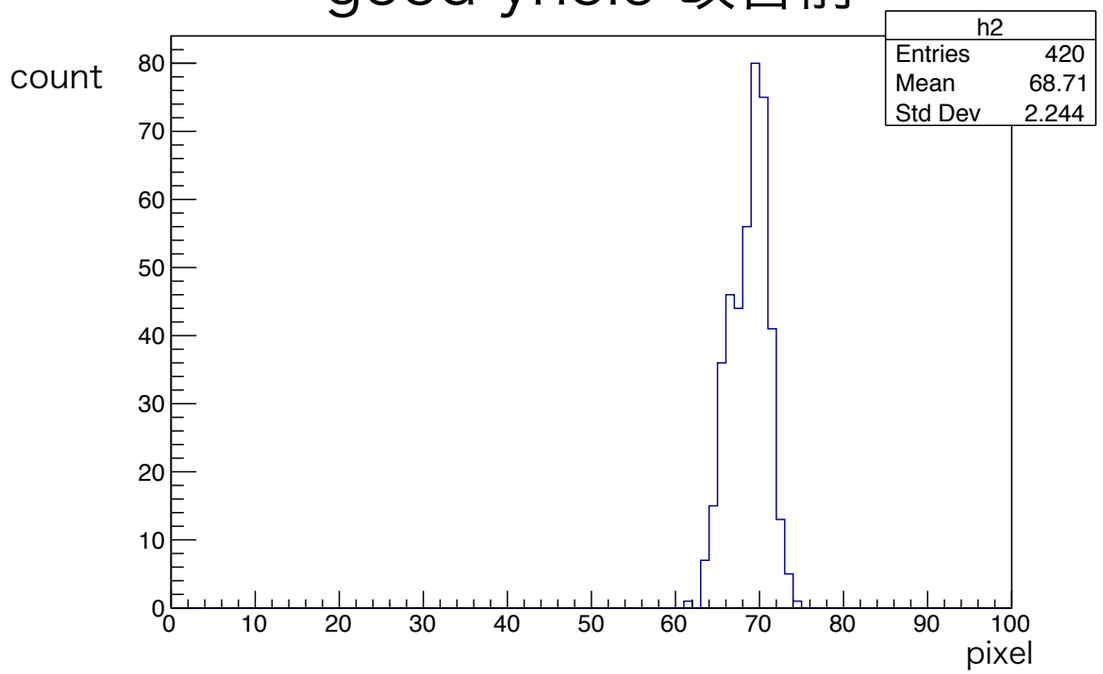


穴位置に関する比較

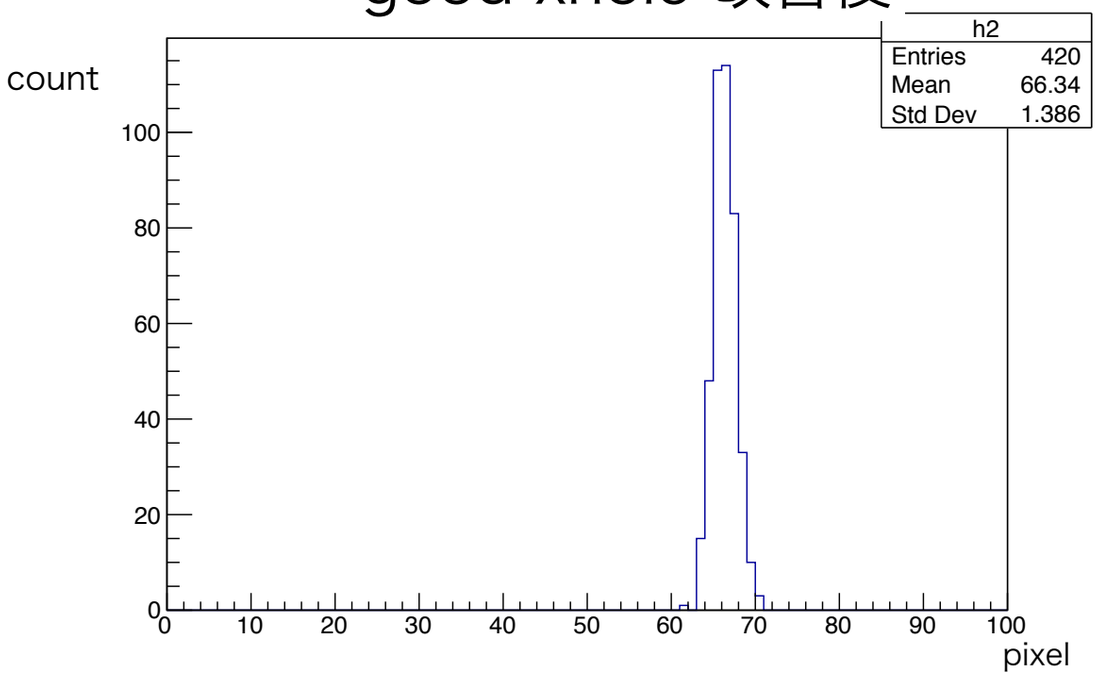
good xhole 改善前



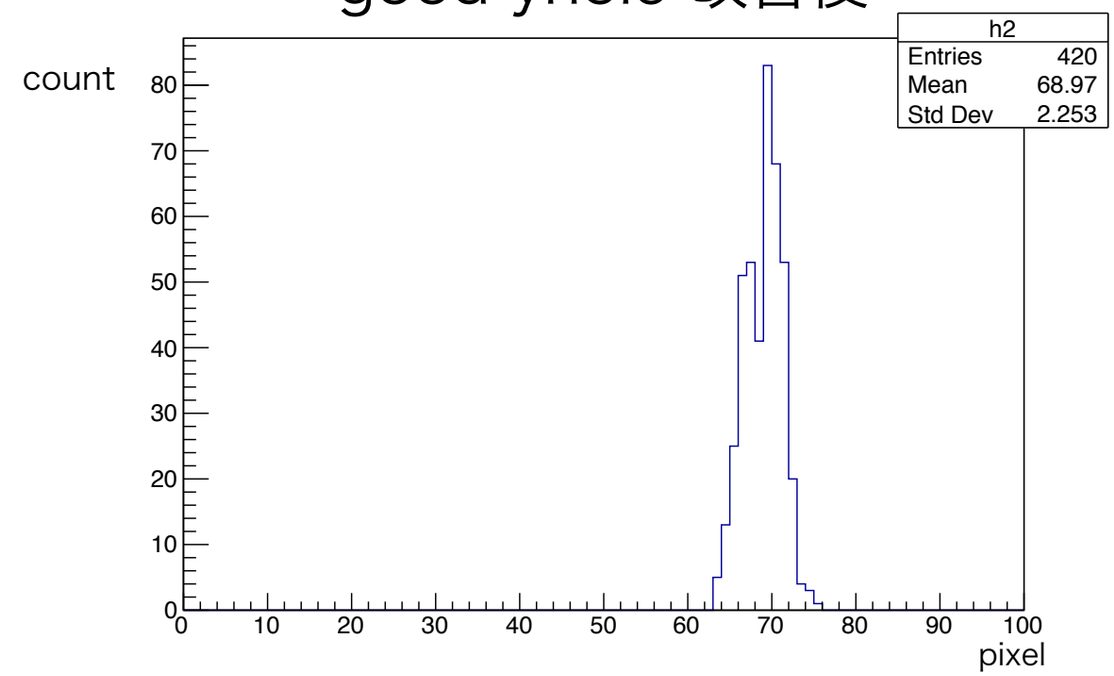
good yhole 改善前



good xhole 改善後

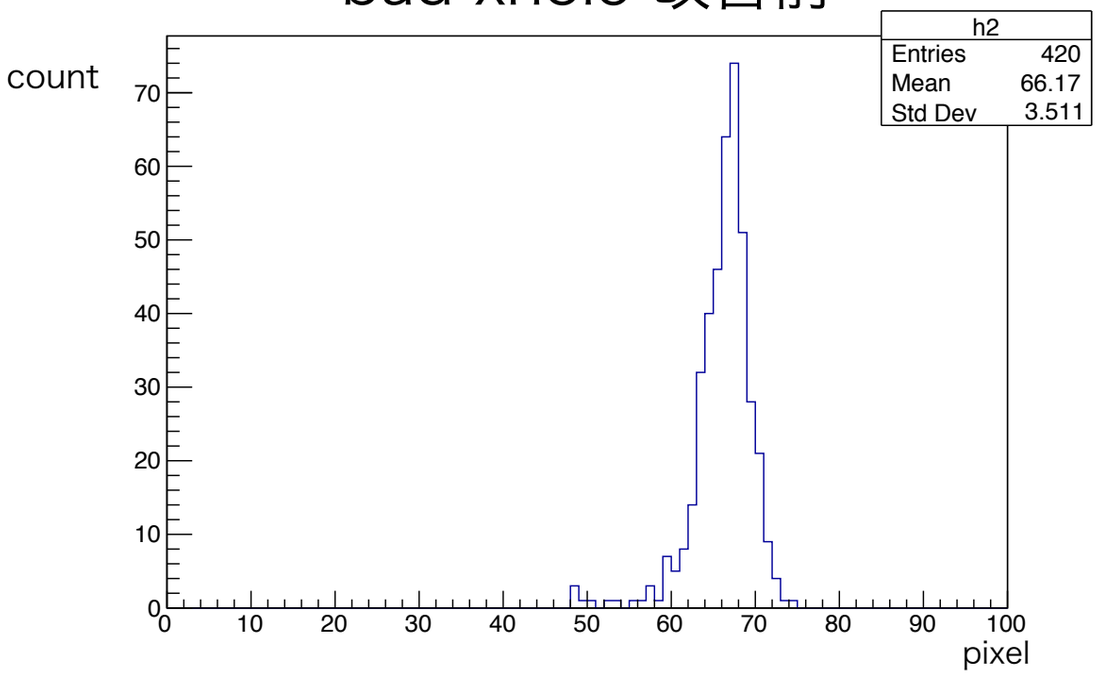


good yhole 改善後

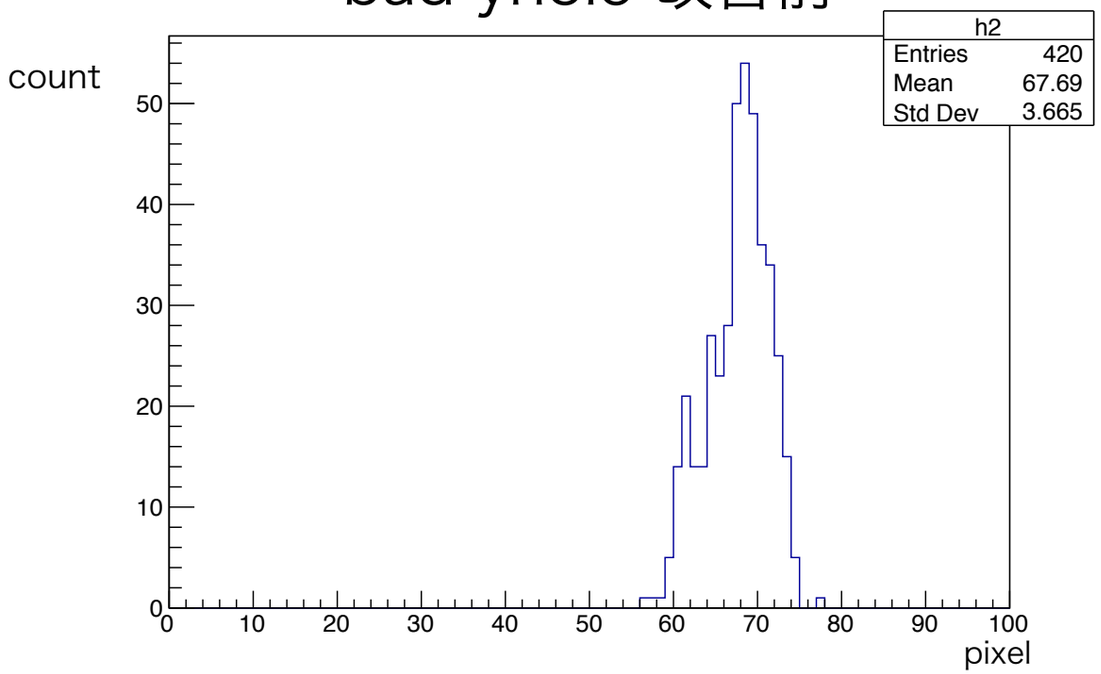


穴位置に関する比較:こちらもほとんど変化なし

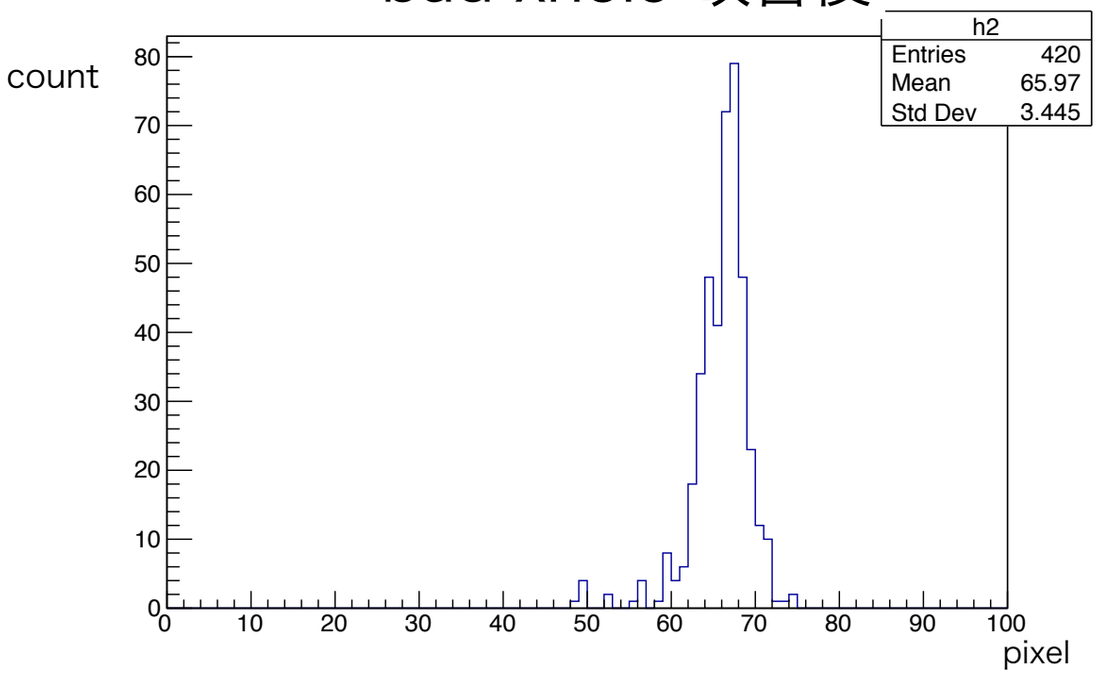
bad xhole 改善前



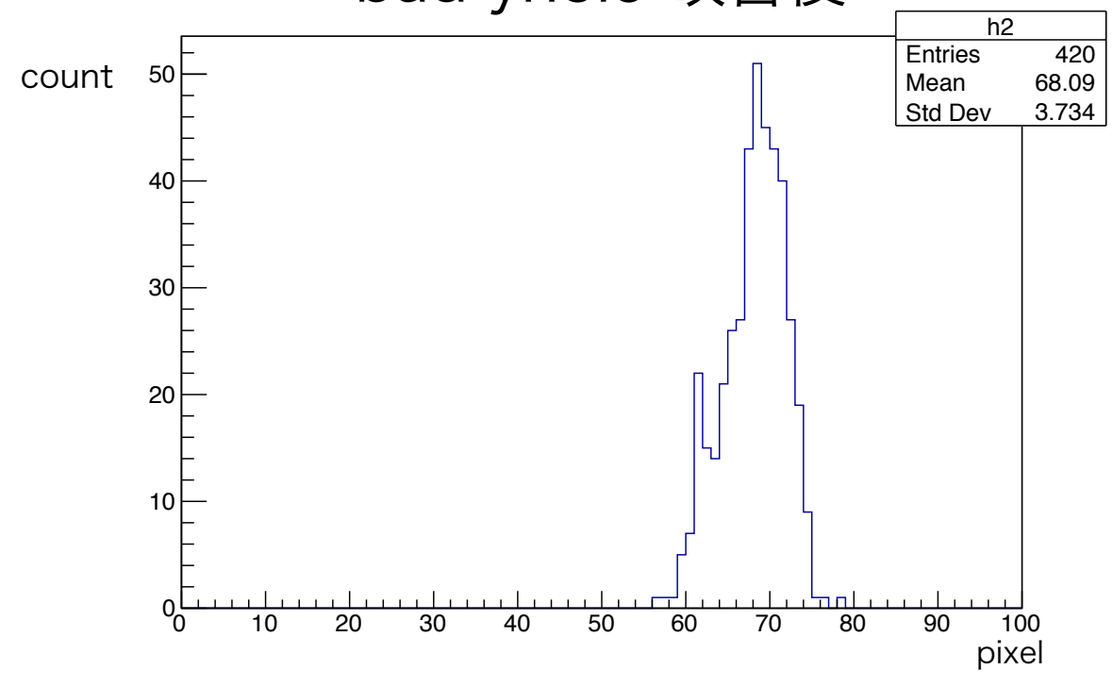
bad yhole 改善前



bad xhole 改善後



bad yhole 改善後



穴内部の明るさの情報を使う

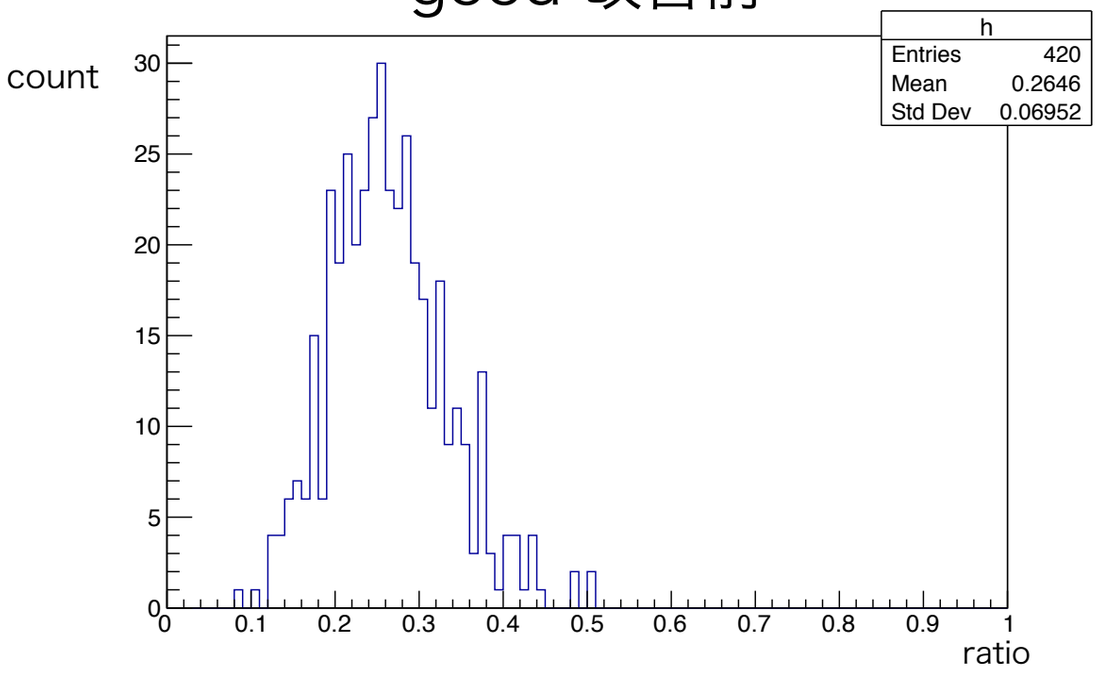
- 新たなパラメータを導入する。

- 穴を検出、その内部のピクセルについて明るさに関してヒストグラムを得る（0~255）

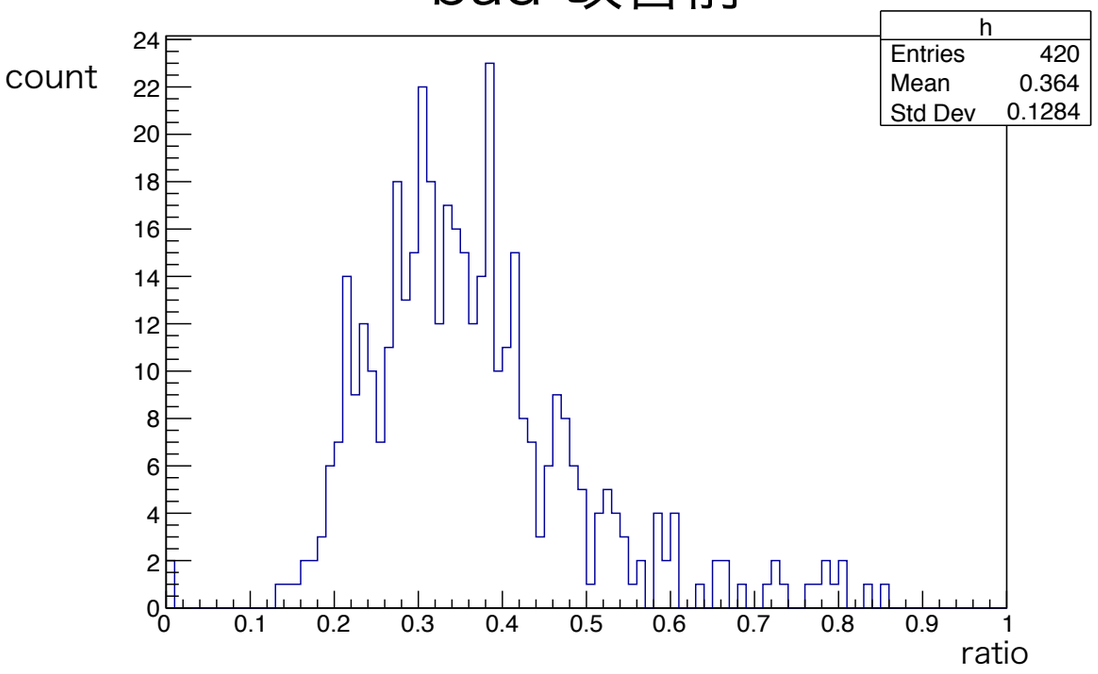
- ピクセルの総数と指定した明るさ以上（今回は 20）のピクセル数を求め、明るいピクセルの割合を求める。

明るいピクセルの割合に関するヒストグラム、ほとんど変化なし

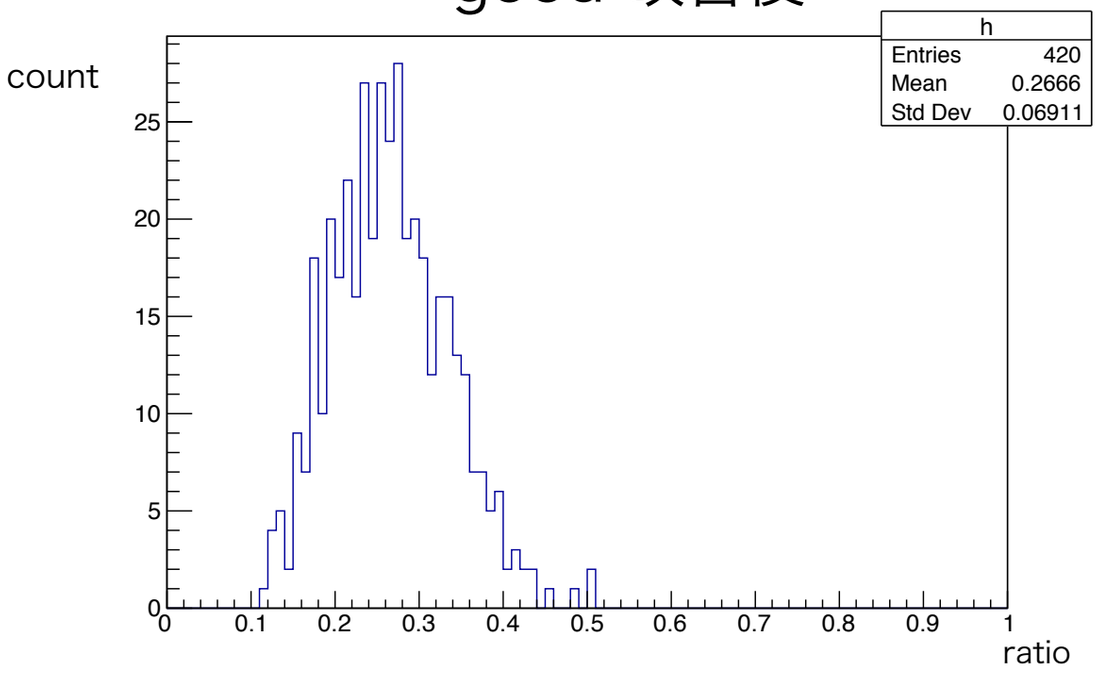
good 改善前



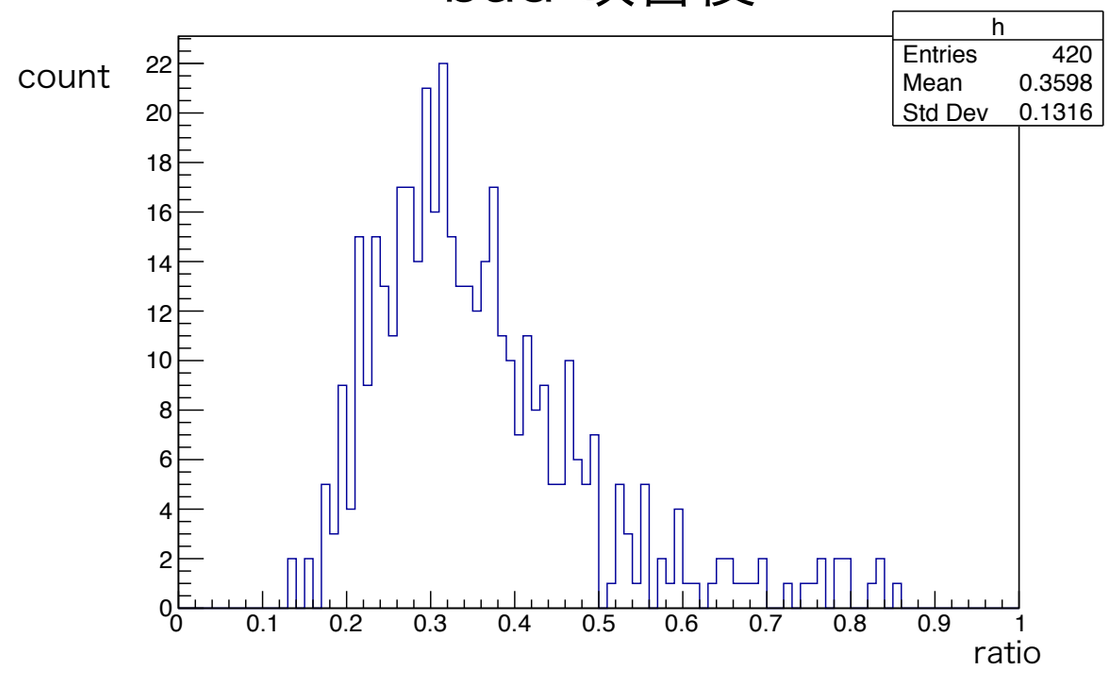
bad 改善前



good 改善後

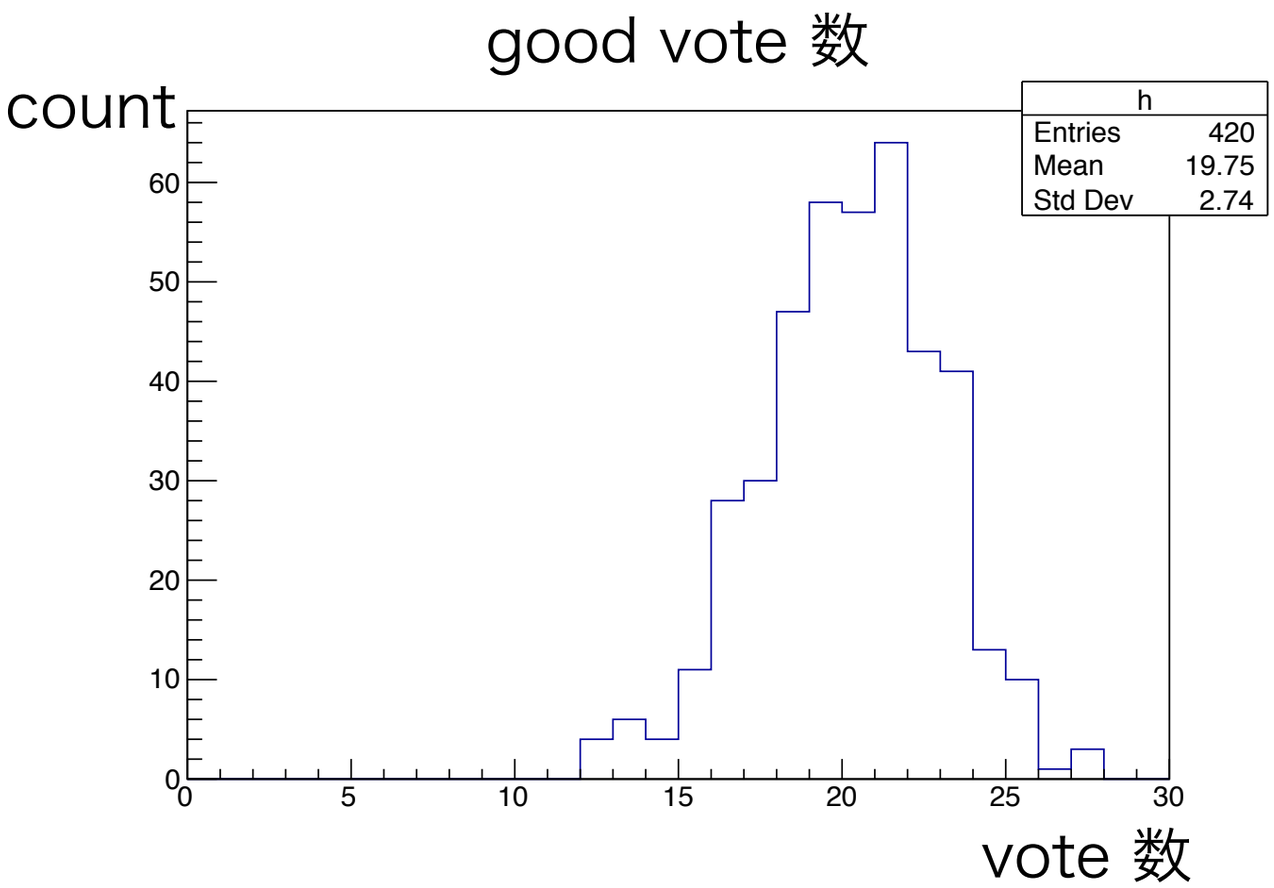


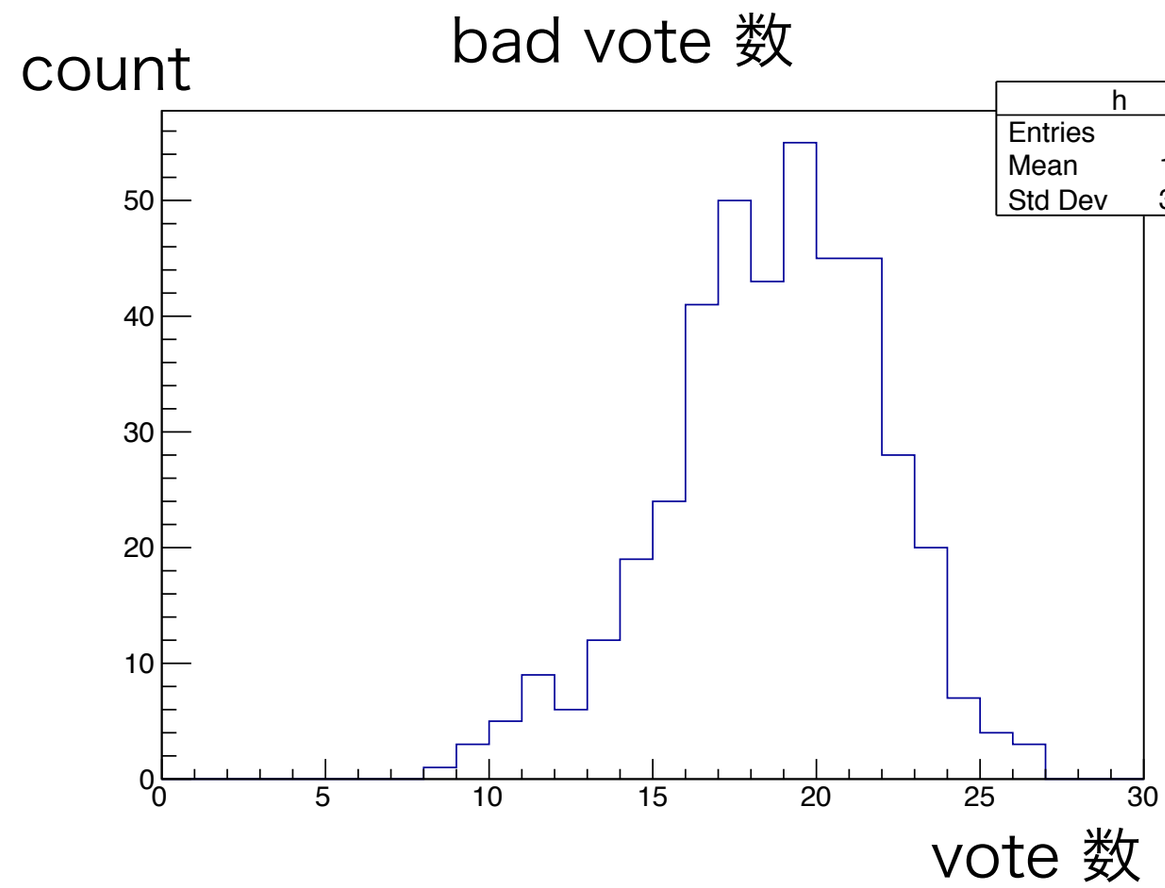
bad 改善後



vote 数に関するまとめ

- 結局、変化は見られなかった。





vote 数分布自体の幅は、good とbad で異なる。

以前は vote 数は１５で固定して行っていたが、実際には
それより高いvote 数で検出できることがわかった。

現在までの撮影状況

- 以前はカメラ軸-穴中心、距離３５cm で撮影を行った。
- 今回、カメラ軸-キューブ面中心、距離３５cm で撮影した。
- さらに、カメラ軸-キューブ面中心、近距離(５cm) での撮影も行った（これは未解析）

現段階での
good/bad 選別状況

- カメラ軸-穴中心での撮影
    - 以前撮影した画像（35cm の距離）を用いる
    - 書き直したコードでの再解析を行った
- カメラ軸-キューブ面中心での撮影

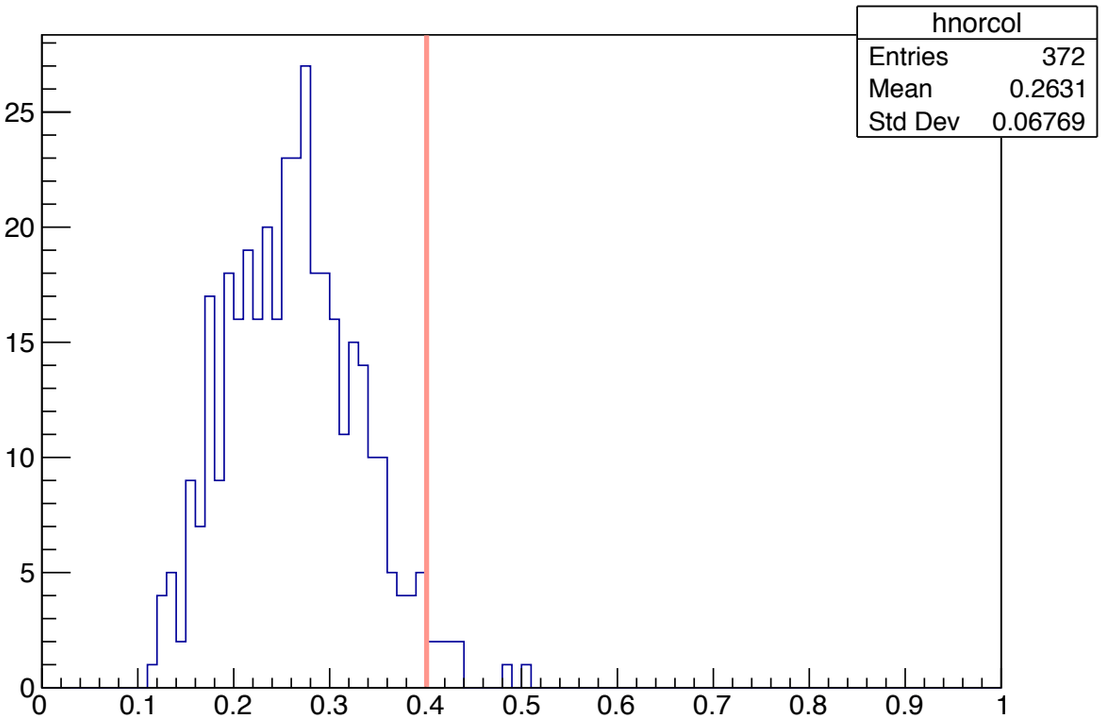
カメラ軸-穴中心

- 以前と同様に、サイズ、穴位置でのカットによりgood: 63キューブ、bad: 14キューブとなった。

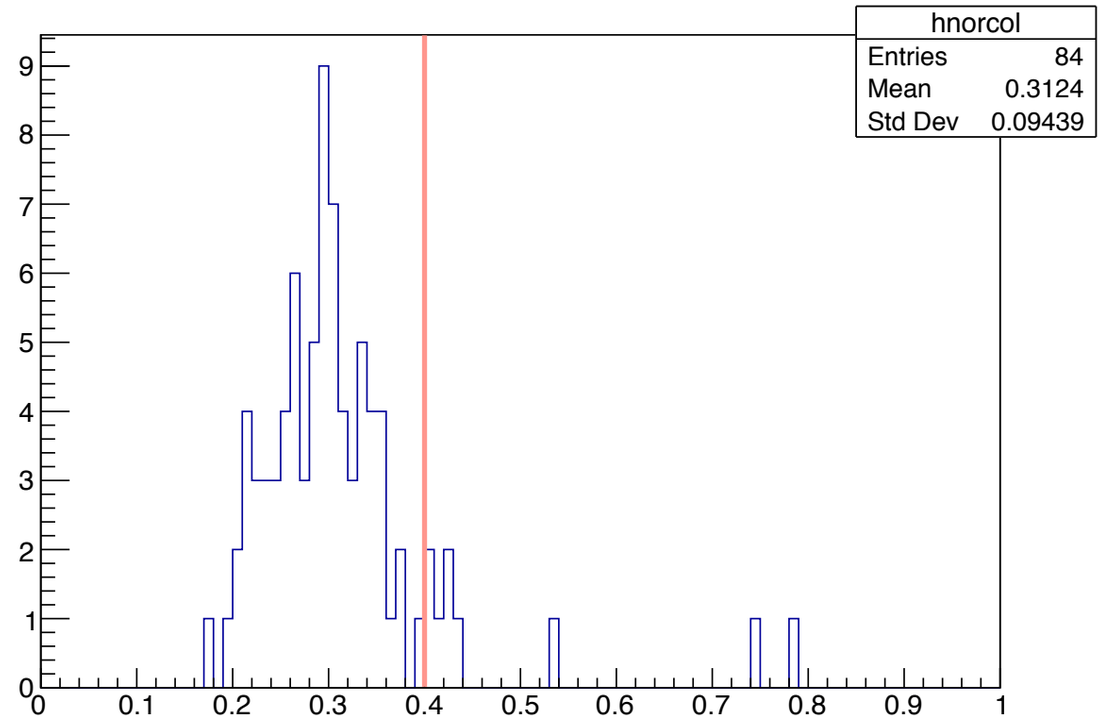
- 円の半径については、今回は検出段階でかなり制限をかけているので、特にカットはかけない。

穴内部の明るさの情報を用いたカット

good ratio



bad ratio



- カット後、残った面について穴内部の明るいピクセルの数を調べる。< 0.4 で更にカットをかけると

good: 53 キューブ、bad: 10 キューブ となる

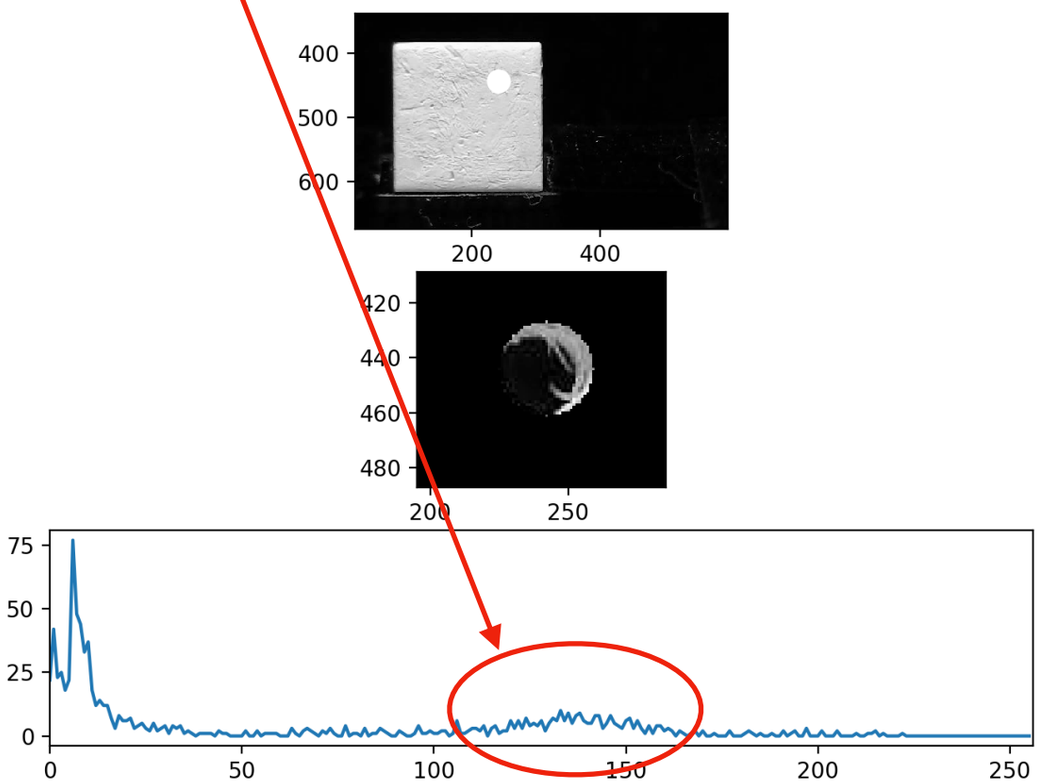
残ったキューブの画像：pic1010bad/selected1108/

重み付け明るさ（提案）

- 明るさ情報を抽出する際に、明らかに穴の内側が見えている場合、それを示す特徴的なヒストグラムになる。

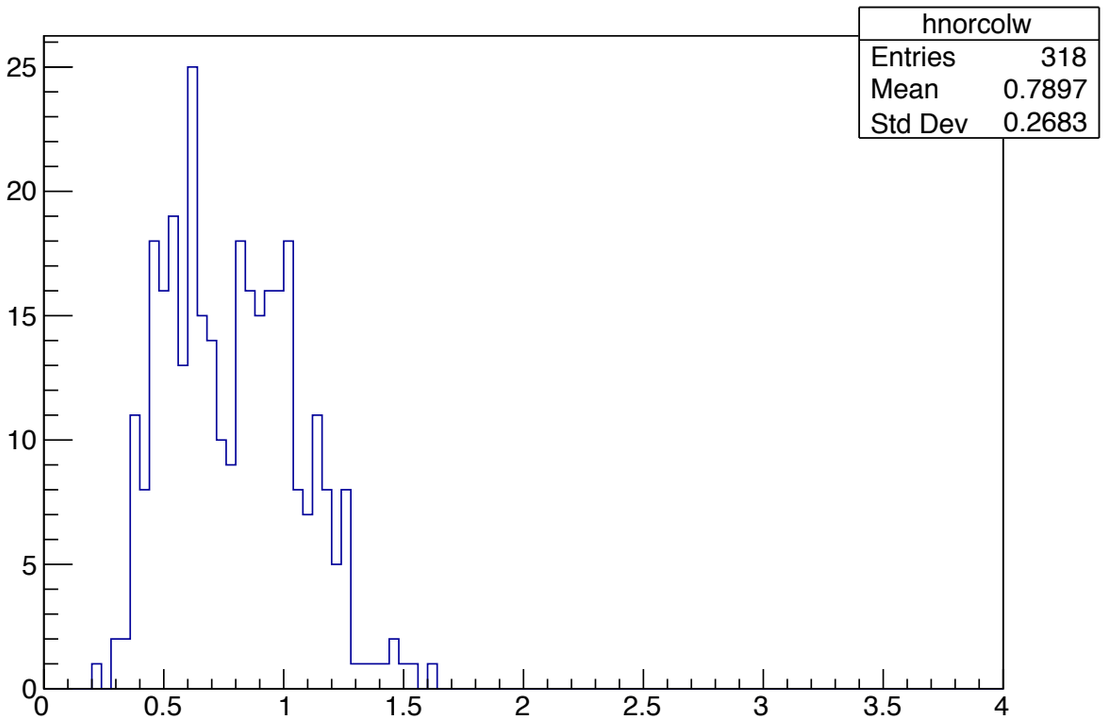
- なんとかしてこれを数値化したい。

- 該当箇所のピクセル数をカウントする際に、重みをつける？

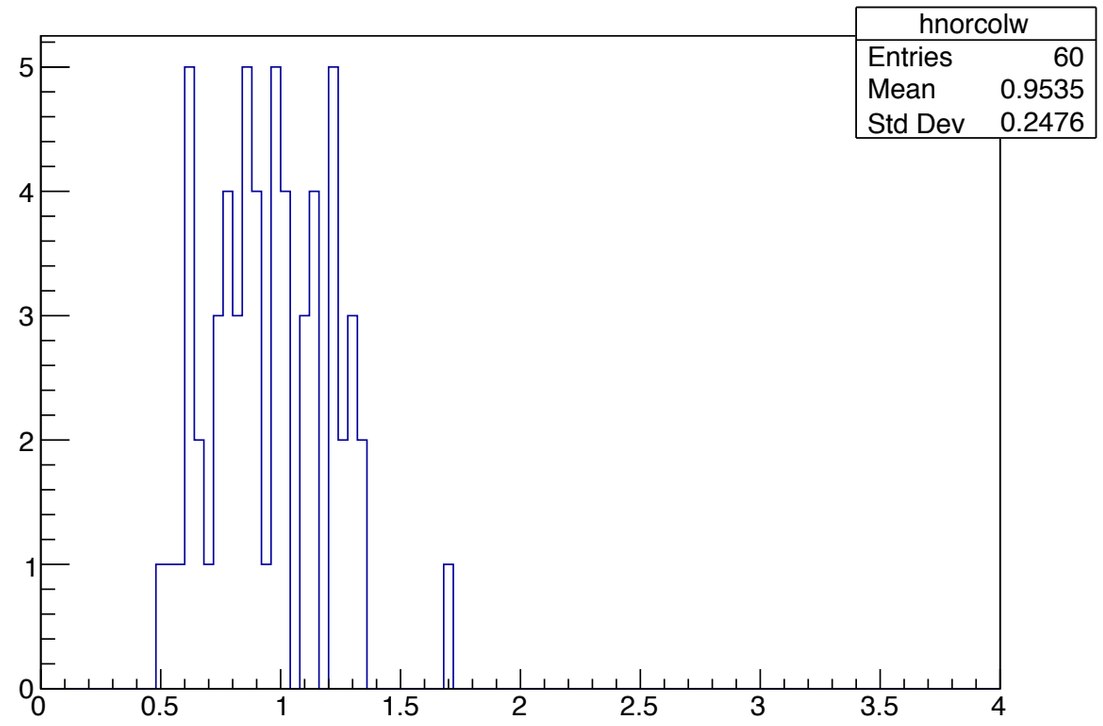


重み付け明るさの情報を用いたカット

good ratio



bad ratio



- 残ったキューブについて、明るさ60以上のピクセルに対して５倍の重さをつけたヒストグラム。

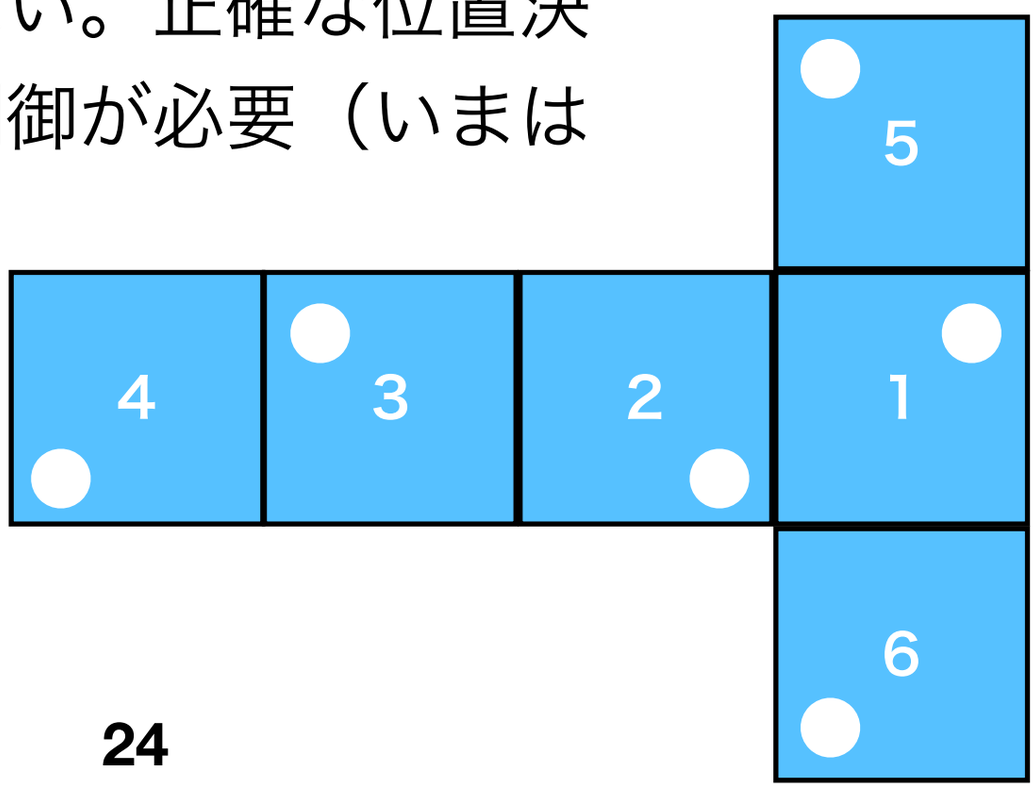
- この情報からは有意なカットはできなさそう

残ったキューブの画像：pic1010bad/selected1108/

カメラ軸-キューブ面中心

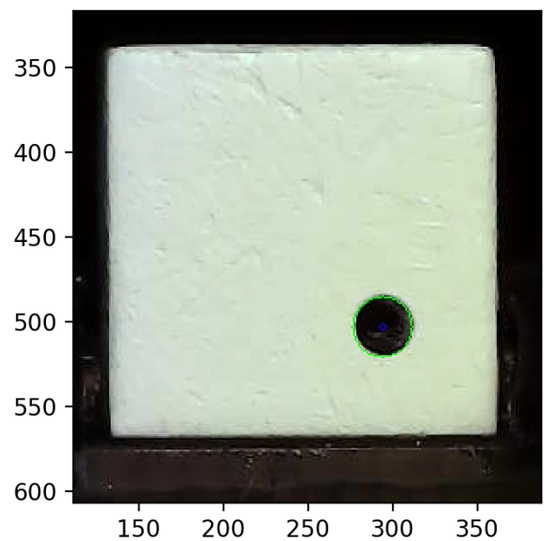
- good、 bad それぞれ７０個下のような面番号、向きで撮影。キューブIDは以前の撮影時と同じ。

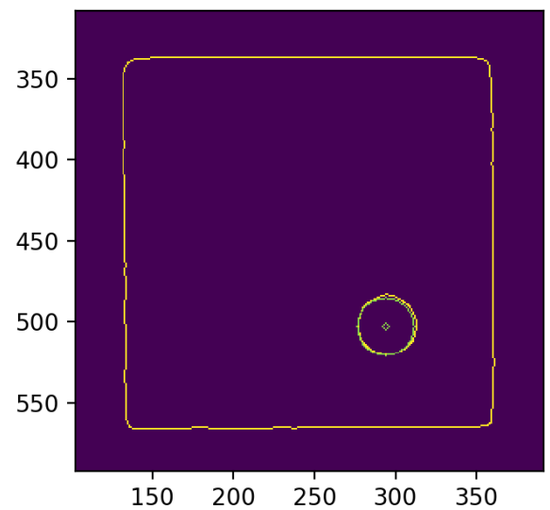
- 以前の撮影もそうであるが、カメラとキューブの相対位置の決定が非常に難しい。正確な位置での撮影は（今の所）できない。正確な位置決めのためにはコンピュータ制御が必要（いまはまだ要らない？）



- 穴の内部の明るさのヒストグラムを得る際に、穴を大きく見積もってしまっていると、左下図のようにキューブ表面の白い部分が入ってしまう。大きく見積もってしまった場合は右下図のように検出半径よりも１小さい円を用いる。

検出は悪くないように見えるが、
１小さい半径の円を用いる。

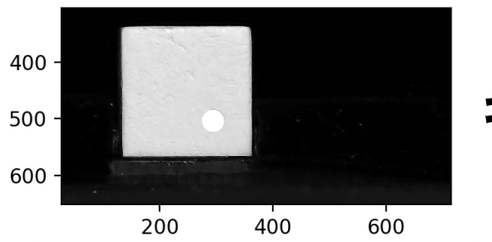




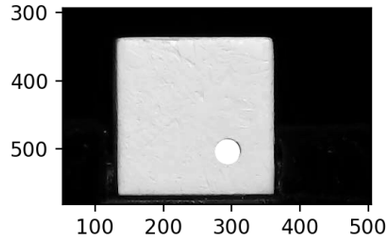
検出半径でのヒストグラム抽出

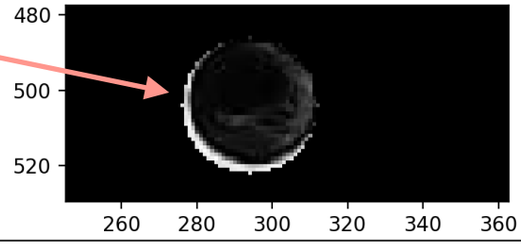
検出半径 -1 でのヒストグラム抽出

円の縁に白い
部分が入る

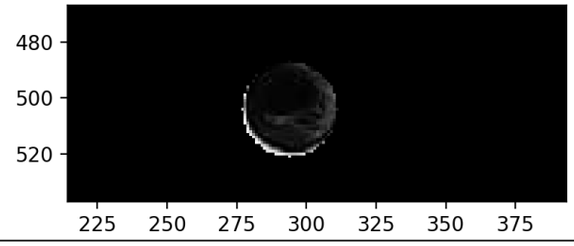


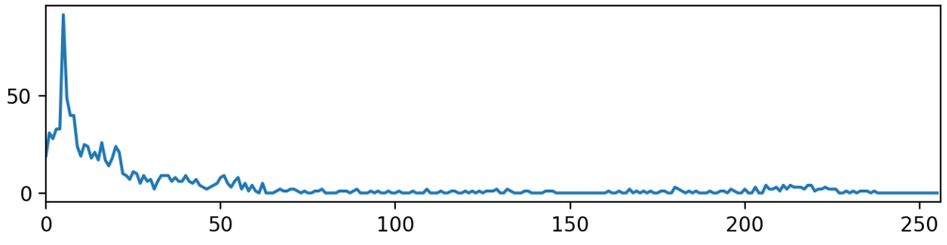
キューブの画像＋マスク

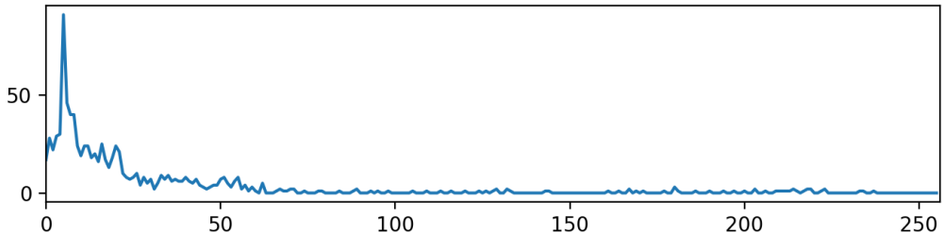




マスク部分（円内部）

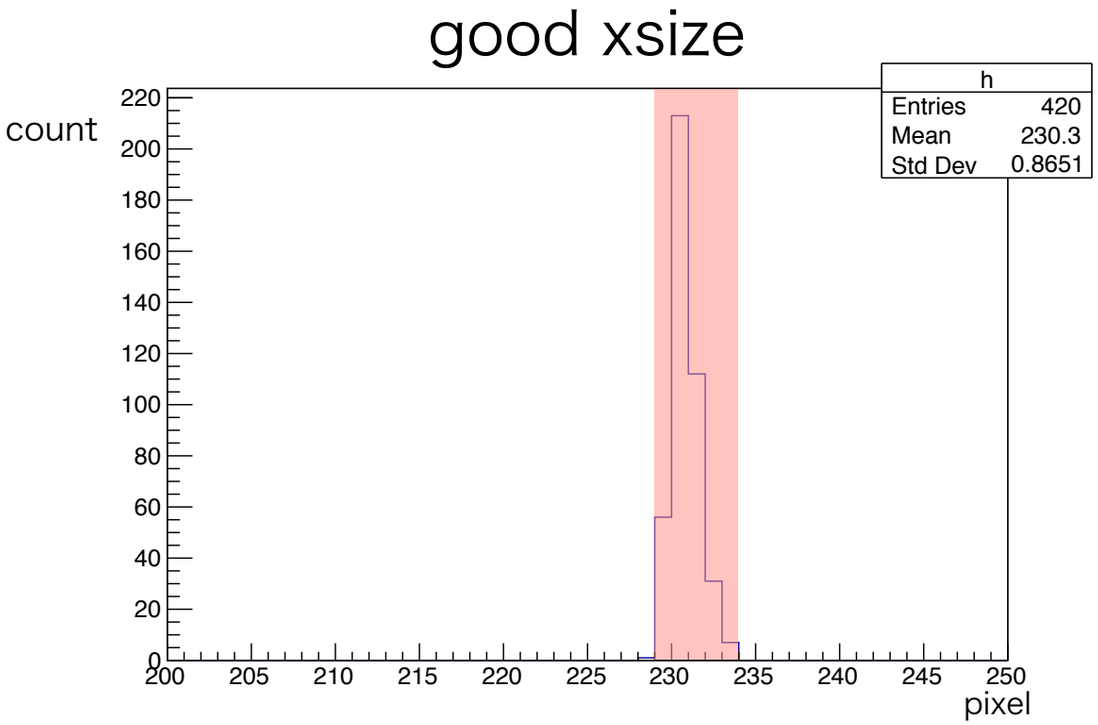


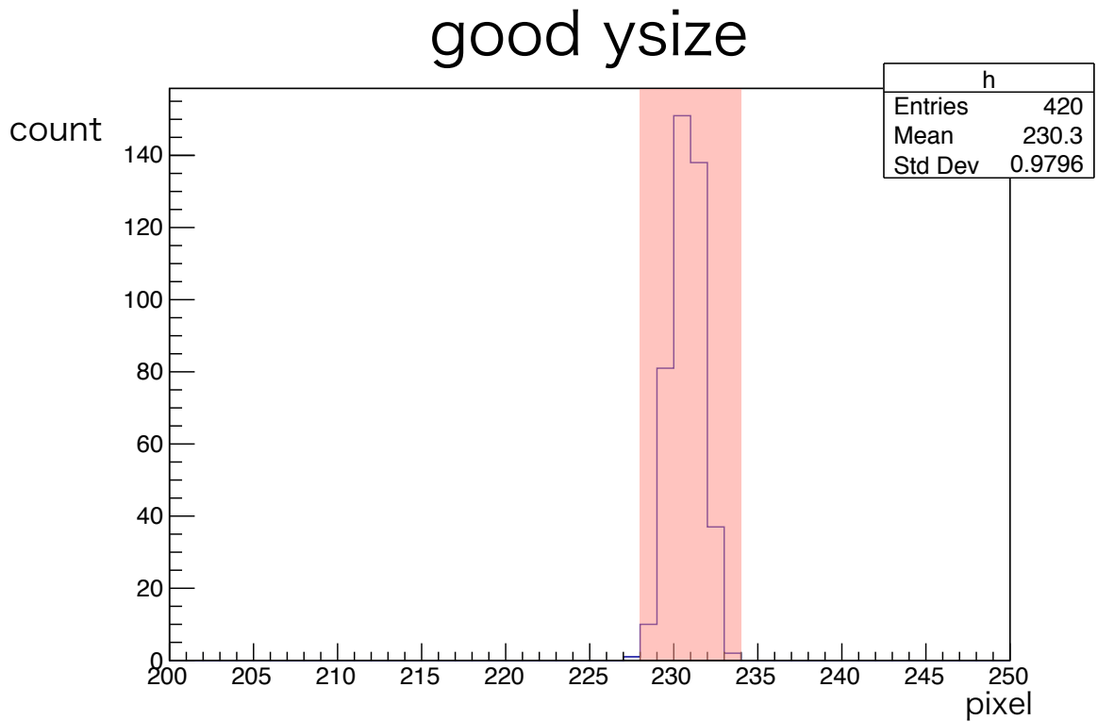


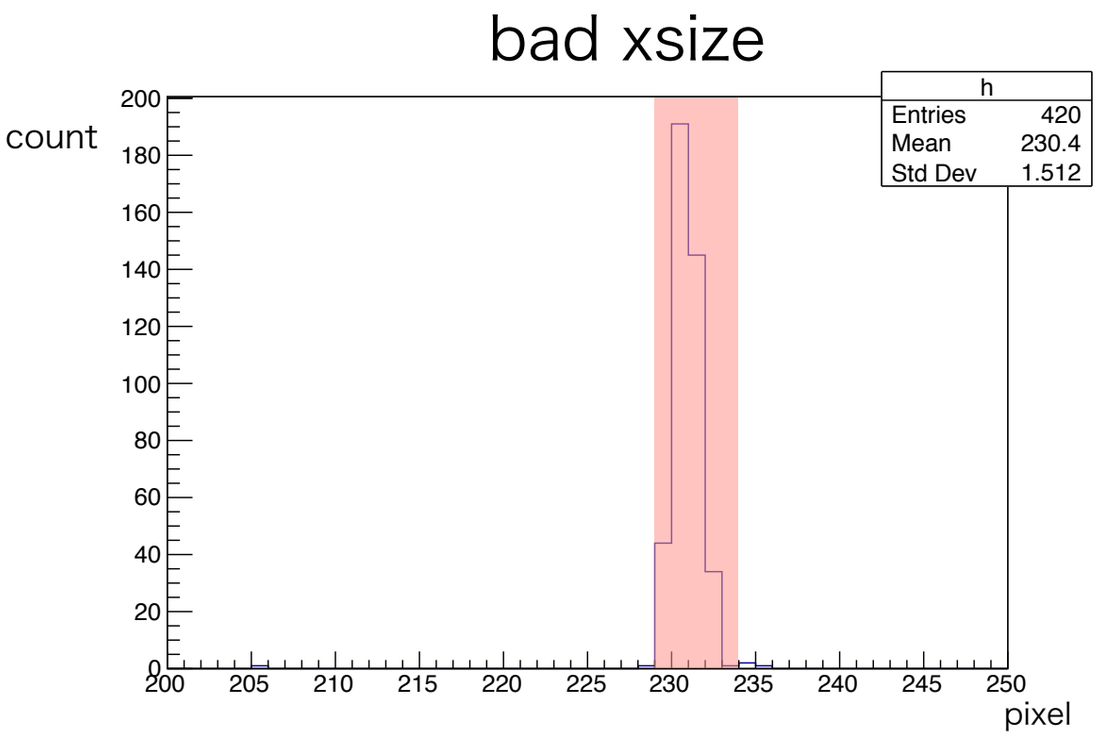


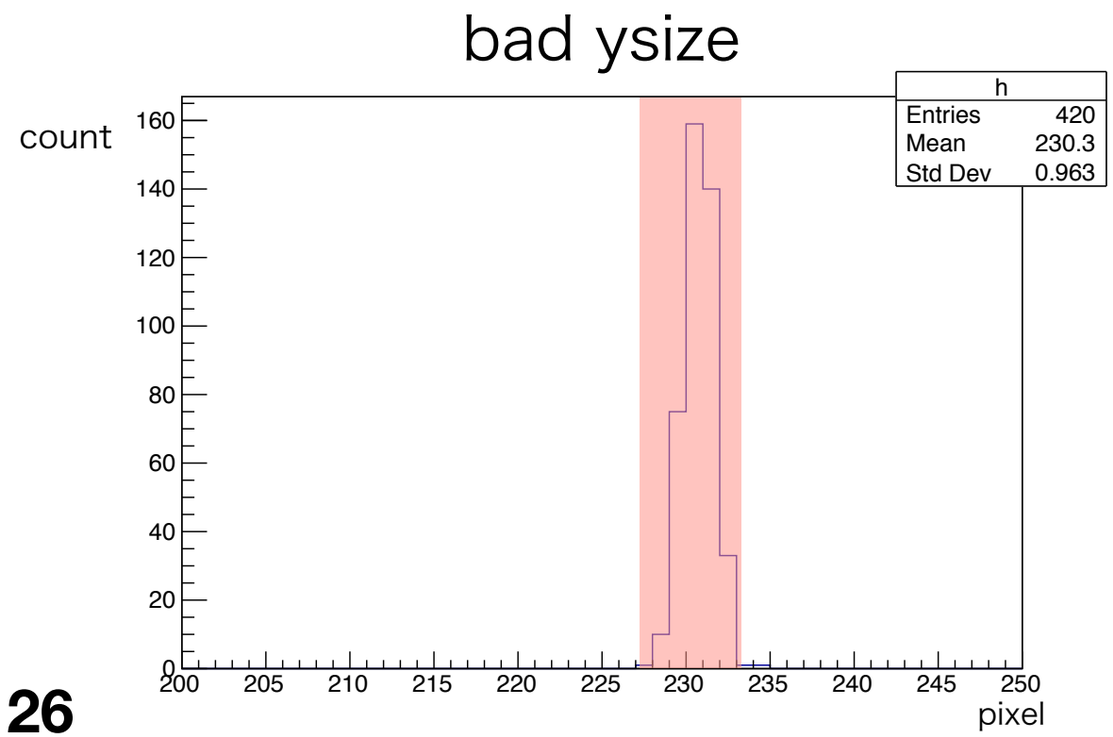
マスク部分のヒストグラム（横軸：明るさ）

good の分布でのカット(サイズ)

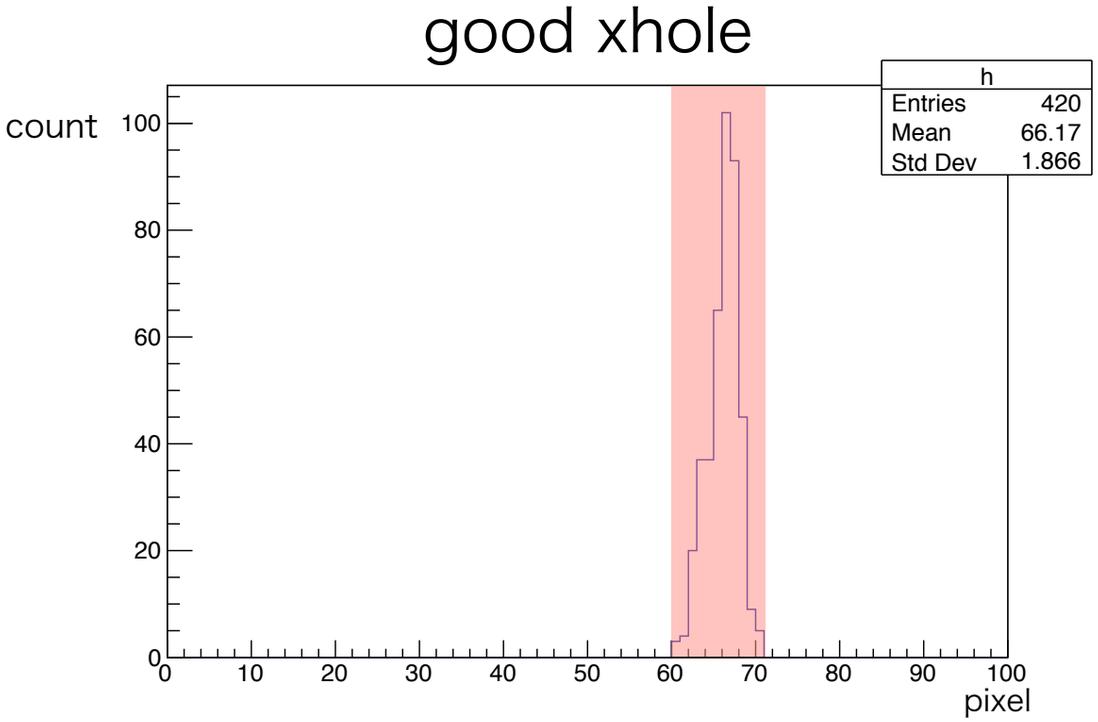


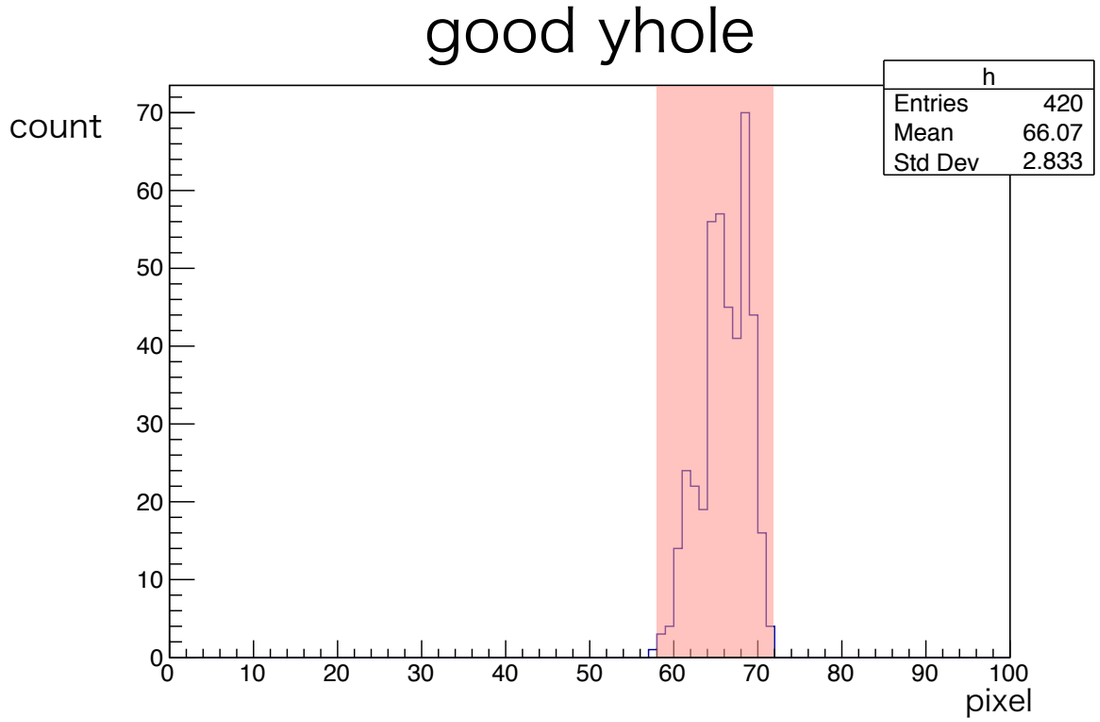


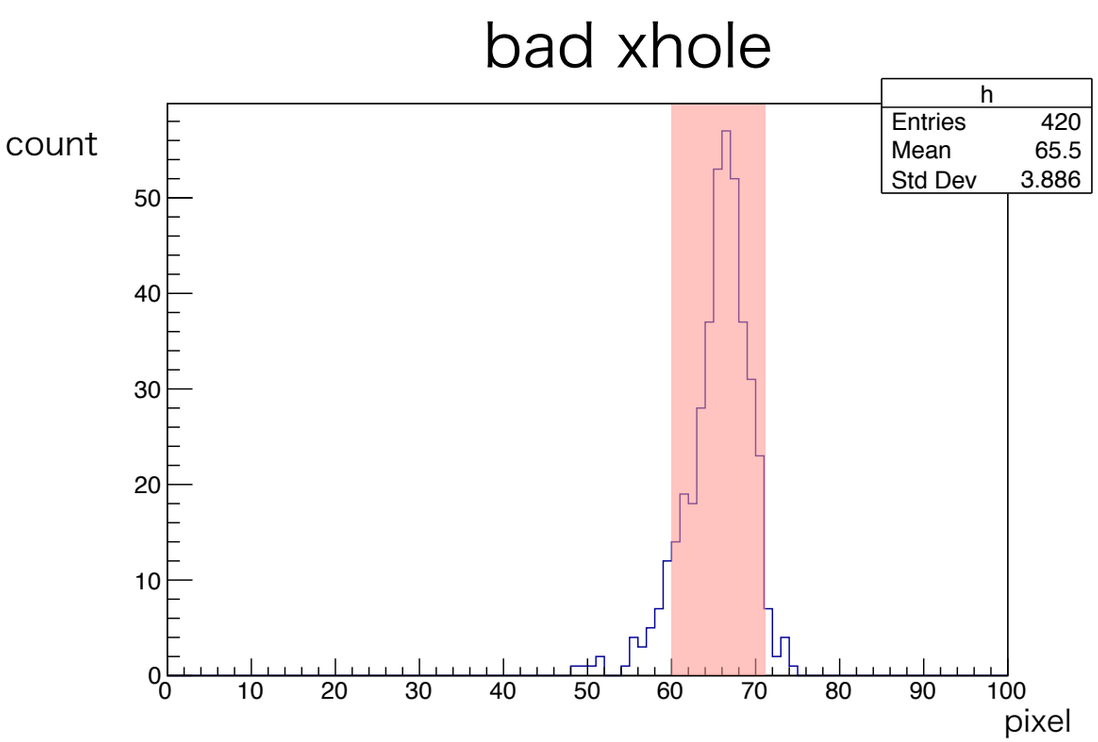


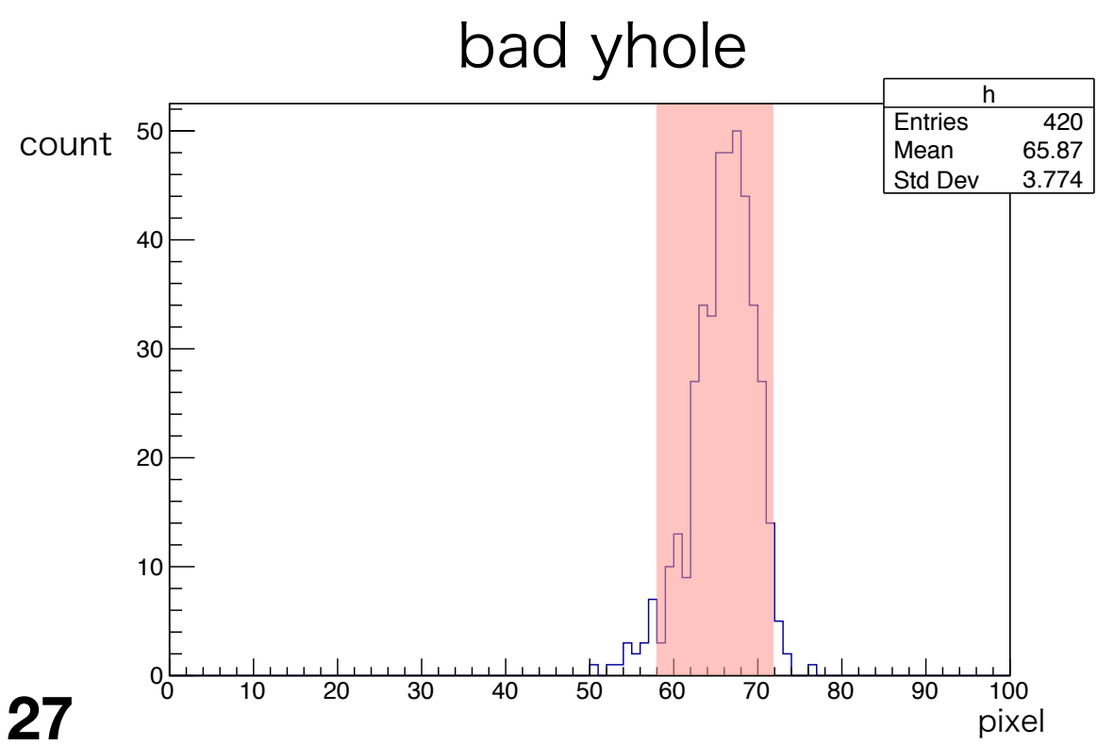


good の分布でのカット(穴位置)









- ここまでのカットで各７０個のキューブから、

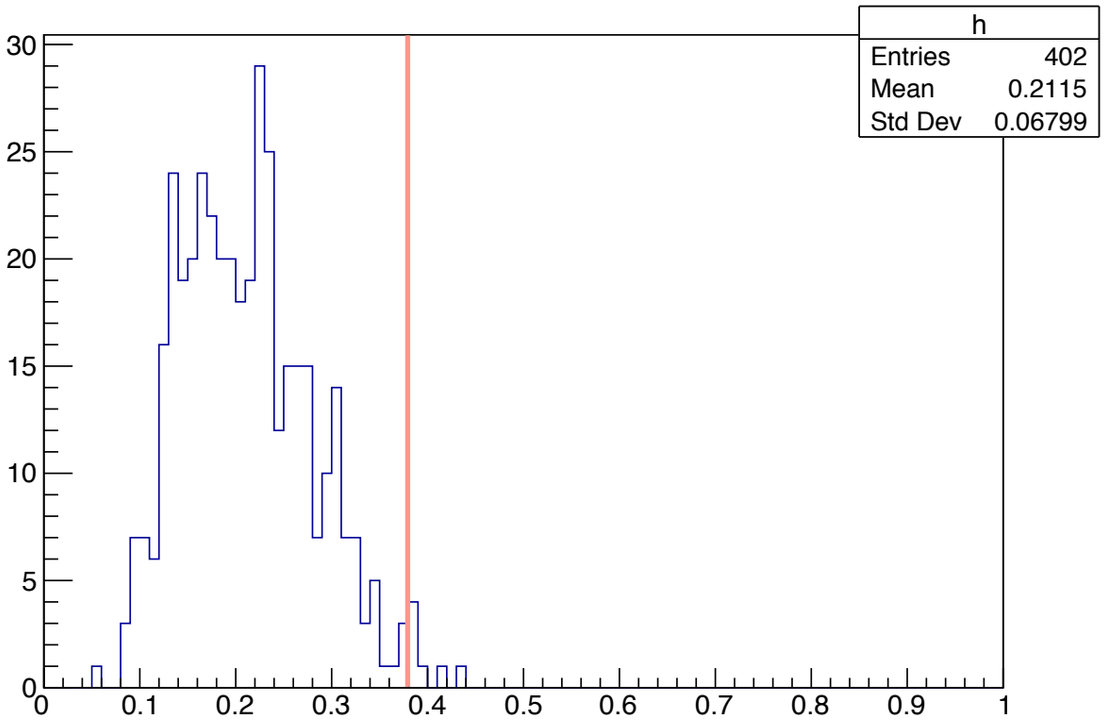
good: 67 キューブ、bad: 25 キューブ となった。

（あるいは good をもっと犠牲にすればもう少しbad を排除できる:

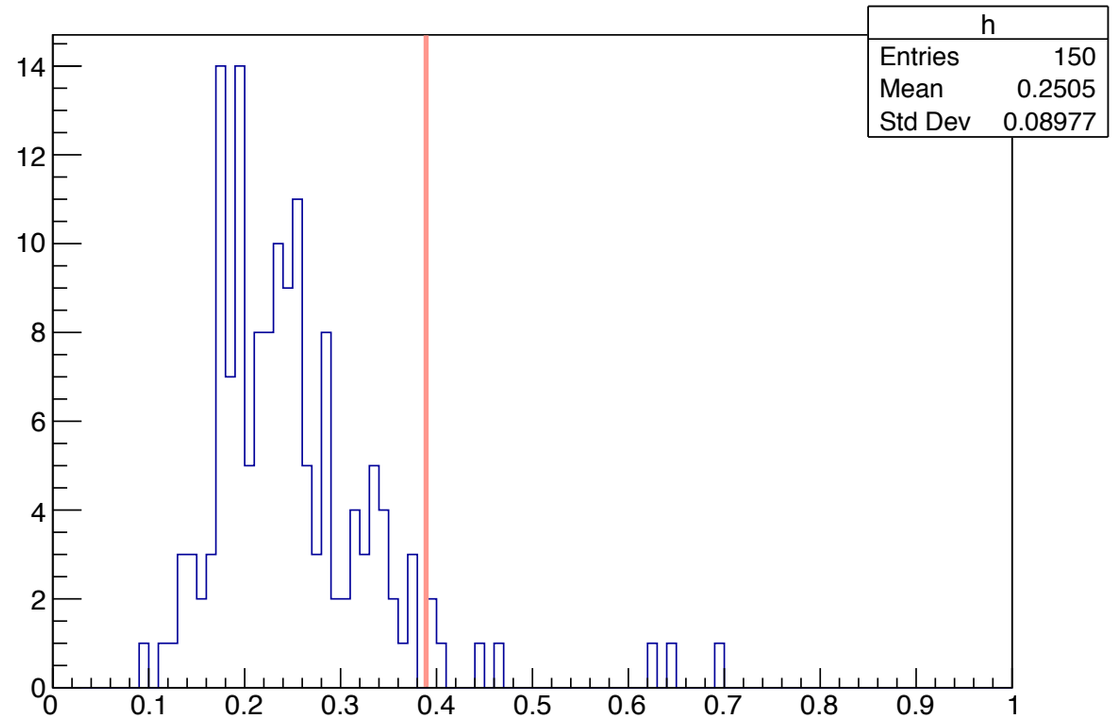
xhole の条件を厳しくして good: 64、bad: 21）

穴内部の明るさの情報を用いたカット

good ratio



bad ratio

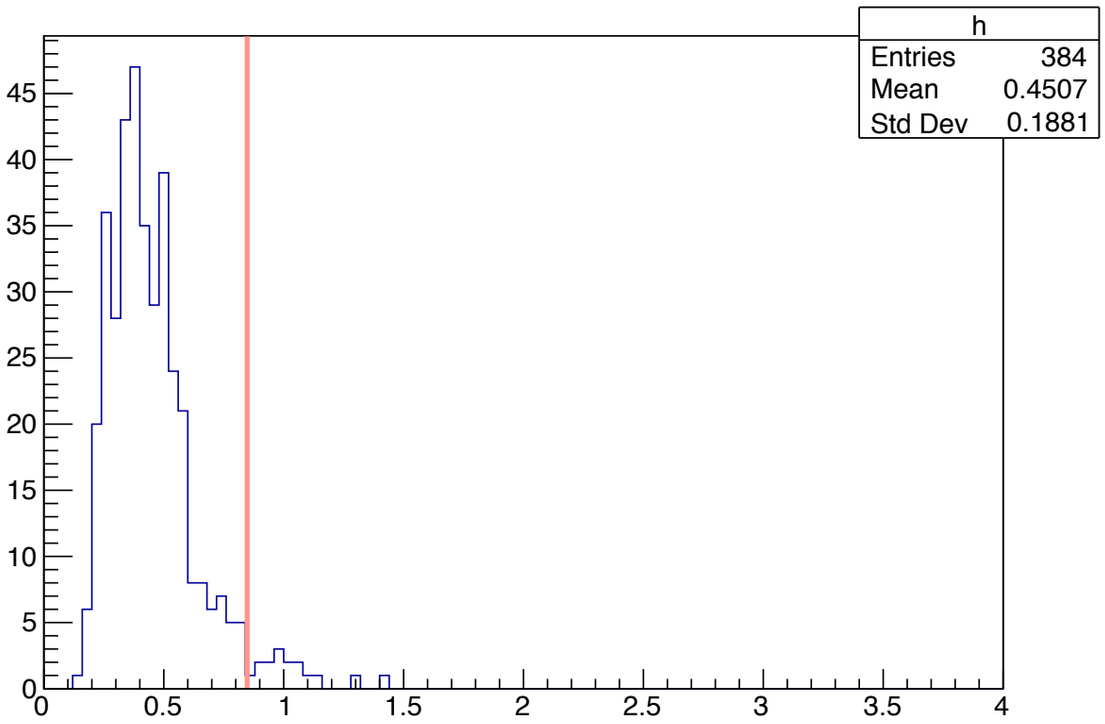


- カット後、残った面について穴内部の明るいピクセルの数を調べる。< 0.4 で更にカットをかけると

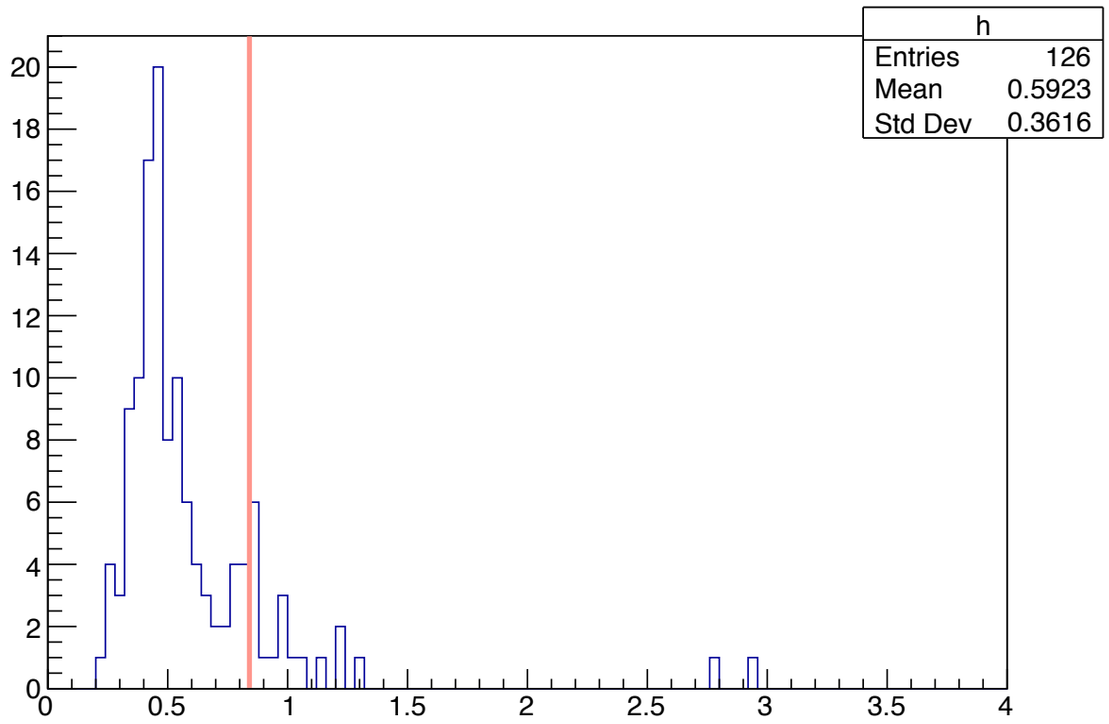
good: 60 キューブ、bad: 17 キューブ となる

穴内部の重み付き明るさでのカット

good ratio



bad ratio



- 20 ~ 199 の明るさのピクセルに対して５倍の重みをつけたヒストグラム。< 0.85 で更にカットをかけると

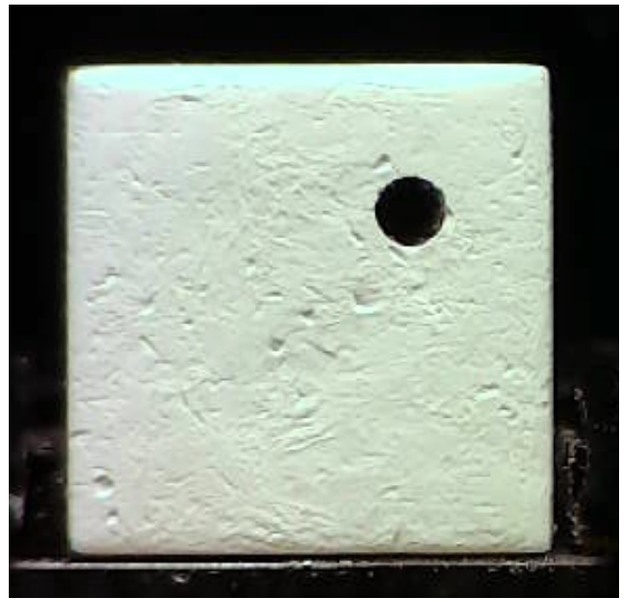
good: 54 キューブ、bad: 13 キューブ となる

残ったキューブの画像：pic1111bad/selected1113/

まとめ

- 以前にくらべて判別効率が向上したとは言えない。
  - 以前の残ったキューブ（穴中心）：good 62個、bad 15個
  - 今回の残ったキューブ（穴中心）：good 63個、bad 14個
  - （面中心）：good 64個、bad 21個
  - 明るさ情報を用いても特に改善はしていない。
- ここまでのカットで残ったキューブをすべて合わせて金属棒でマニュアル・クオリティチェックを行ってみる。
- すべてのキューブがマニュアルチェックをクリアすれば選別のアルゴリズムはうまく作動しているといえる。だめなら再考の必要あり。

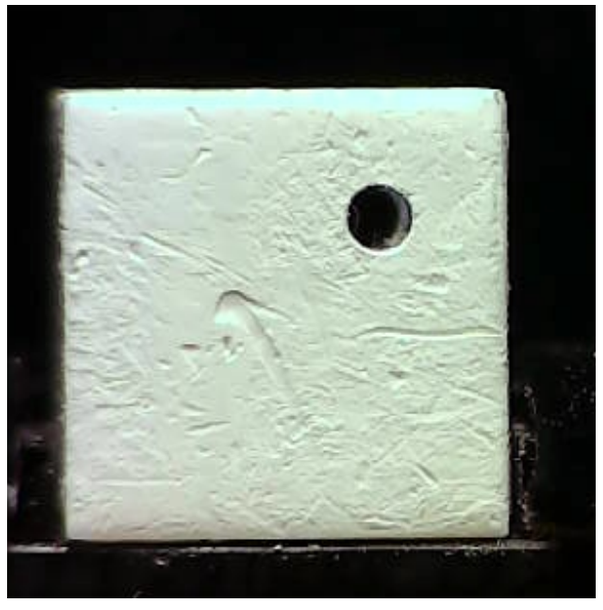
- 画像を見た感じだと、カメラ軸-穴中心だと内部のようすがよく見える。逆に、少しでも傾いた状態で撮影してしまうと穴まで傾いたように写ってしまう（右図）。



穴中心、まっすぐ撮影した例

- カメラ軸-キューブ面中心だと穴の内部が（少なくとも今の照明では）暗くなってしまう。ただし穴の内部の情報を用いない場合は４辺を等しく扱えるこちらが有利かもしれない。

キューブ面中心、右下図と同じキューブの同じ面。穴の中がよくわからない。



穴中心、傾いて撮影したと思われる例。キューブ自体が少し右を向いているように見える

近距離での撮影

- カメラ軸-キューブ面中心にて、距離５cm 程度で撮影して、good がどれだけばらつくか確認する。解析は後日

- 穴の内部の情報を使わない、カメラの視差等を気にせず表面の情報のみを使う場合は、結局この撮影が最もゆうりかもしれない。