画像処理による粉粒体高速流動の画像情報の統計的処理

吉渡 匠汰

概要

直観的な視覚情報において「ふうあい」は重要な要素の一つである。しかし、そのふうあいを言語で説明することは難しく、「暗黙知の領域」とされてきた。だが、近年では 0.1 ms レベルでの撮影を手軽に行えるようになり、この領域の基礎研究が進められてきた。以前までの研究では粉や水のハイスピード映像を撮影し、目視によってその特徴を調査してきた。本研究では二値化画像処理を中心としたコンピュータによる解析を行うことで粉粒体落下運動の映像の自己相関関数 $G(\Delta t)$ を求めた。テクスチャ状の特徴と比較し、粉粒体の落下運動の映像を数値で説明することを試みた。

目次

1　物体の運動評価法

物体の視覚情報を数値化する方法として、視覚情報の自己相関関数を計算する方法を考えた。自己相関関数とは 1 つのムービーについて、ある瞬間の映像が時間シフトした映像とどれだけ一致しているかという尺度である。自己相関関数を様々な粉粒体の落下映像について求めることとで、映像の変動を解析した。

1.1　画像処理について

物体の動きを正確に追うため本研究では二値化画像処理を用いている。二値化とは各画素について一定の閾値以下ならば白、そうでないならば黒にするという処理を施し、二色の画像を作成する処理である。運動する対象の色とギャップの大きい色を背景にして撮影し、二値化処理によって対象を白または黒で抜き出すことができる。

二値化処理を施した画像の例として図 1 を示す。

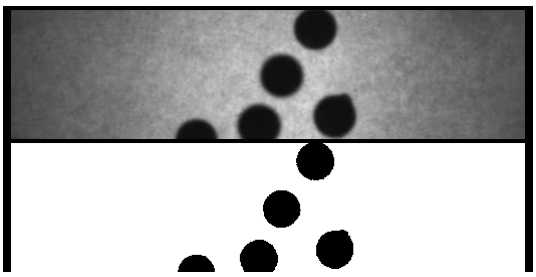

図 1　二値化画像の例 上：撮影された映像の 1 フレーム、下：二値化画像

本研究では解析の開発も行った。用いた言語は C++ と Python である。画像処理に関わるソースコードを Code 1 に示す。画像処理には OpenCV ライブラリを用いている。Code 1 は粉粒体の落下運動の動画からフレームを切り抜き、トリミングおよび二値化処理を行うプログラムである。そして、トリミング画像・二値化画像・2 つを並べた画像の 3 種類を各フレームごとにナンバリングして保存する。二値化を行う threshold 関数は閾値を引数に持つ (ここでは 33 となっている)。これはピクセルを黒に変換する境界で、閾値を低くすればするほど二値化画像は黒の割合が高くなる。

Code 1　動画のフレーム切り抜きと二値化

```
1  import cv2
2  import os
3
4  def video_to_frames(video, path_output_dir):
5    #Get video (mp4)
6    vidcap = cv2.VideoCapture(video)
7    count = 0
8    while vidcap.isOpened():
9      #Get a frame from video
10     success, image = vidcap.read()
11     if success:
12       #Get info with frame
13       height, width = image.shape[:2]
14       #Trimming frame & Save
15       tri = image[int(height/4)+18:height,0:
             width]
16       cv2.imwrite(path_output_dir+"/tri/"+
             str(count)+".png",tri)
17       #Binarization frame & Save
18       gray = cv2.cvtColor(tri,cv2.
             COLOR_BGR2GRAY)
19       ret, dst = cv2.threshold(gray,33,255,cv2
             .THRESH_BINARY)
20       cv2.imwrite(path_output_dir+"/two/"+
             str(count)+".png",dst)
21       #Union above two image & Save
22       fuse = cv2.vconcat([gray,dst])
23       cv2.imwrite(path_output_dir+"/"+str(
             count)+".png",fuse)
24       print(count)
25       count += 1
26     else:
27       break
28   cv2.destroyAllWindows()
29   vidcap.release()
30
31 video_to_frames('penguine.mp4', 'out')
```

1.2　相関関数の評価方法

まず画像間の類似性というものを考える。ここで類似性とは同一時系列上にある 2 つの画像がどれだけ似ているかという意味で用いているが、まずは類似性の数値化を試みた。そのためには類似性の具体的な定義を行う必要がある。本研究では次のように定義した、「画像間で色が一致しているピクセルの数 [px]」。イメージ画像を図 2 に示す。比較する際にまずベースフレーム (t_0 時点) を一つ決める。次に $t_0 + \Delta t$ 後の画像と比較する。色が一致するピクセル数を $S(\Delta t)$ と表すことにする。$S(0)$ は同じ画像同士の比較なので完全に一致するため、$S(0)$ は

比較画像のピクセル数に等しい。Δt が少しずれて、例えば $S(\frac{1}{5000})$ のようになると $S(0)$ よりも少し小さくなる。これを $0 \leq \Delta t \leq 0.1[s]$ の範囲で計算する。なお落下運動は定常なものであるため、t_0 を適当に 20 点選び平均をとっている。

$S(0 \leq \Delta t \leq 0.1)$ を $S(0)$ の値で割り、規格化した値を自己相関関数 $G(\Delta t)$ とする。$S(\Delta t)$ の範囲が $0 \leq S(\Delta t) \leq S(0)$ であるから、相関関数 $G(\Delta t)$ の値域は $0 \leq G(\Delta t) \leq 1$ となる。自己相

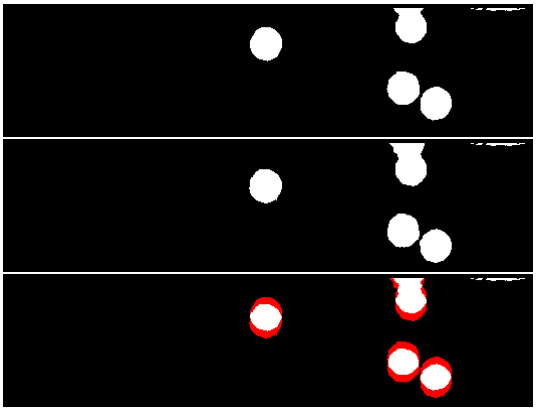

図 2　画像比較のイメージ (上 : t=0s, 中央 : $\frac{3}{5000}s$, 下 : 比較)
黒、白 · · · 色が一致, 赤 · · · 色が不一致

関関数を計算するために 2 つのプログラムを作成した。まず、このプログラム群を以下に示す。Code 2 は二値化画像を元にテキストファイルを作成するプログラムである。このプログラムは解析上必須ではないが、画像ファイルへのアクセスは時間がかかるプロセスなため一旦テキストに書き出すことで今後の解析時間の短縮を行っている。なお、二値化画像は 0:黒、255:白の 2 種類のピクセルで構成されている。テキスト化に当たってはファイルの可読性を考慮して 1:白に変換している。

Code 2　二値化画像のテキスト書き出し

```
#include<iostream>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/core/core.hpp>
#include<opencv2/features2d.hpp>
#include<opencv2/opencv.hpp>
#include<opencv2/imgproc.hpp>
#include<sys/stat.h>
#include<string>
#include<fstream>
#include<iostream>
#include<vector>
#include<sstream>

using namespace std;
using namespace cv;

void img_to_bw(string,string);
string tostr(int);

int main(){
  img_to_bw("out/two/","out/two/txt/");
  return 0;
}

void img_to_bw(string path_input,string
    path_output){
  int i = 0;
  struct stat st;

  //image information
  Mat img = imread(path_input+"0.png",
      CV_LOAD_IMAGE_GRAYSCALE);
  int height = img.size().height;
  int width = img.size().width;

  while(1){
    string path = path_input+tostr(i)+".png
        ";
    if(stat(path.c_str(),&st) != 0){
      //no File
      break;
    }
    ofstream ofs(path_output+tostr(i)+".txt
        ");
    Mat img = imread(path,
        CV_LOAD_IMAGE_GRAYSCALE);
    int height = img.size().height;
    int width = img.size().width;
    for(int h=0;h<height;h++){
      string oneline = "";
      Vec3b *pixs = img.ptr<Vec3b>(h);
      for(int w=0;w<width;w++){
        int pix = static_cast<int>(img.at<
            unsigned char>(h,w));
        /*Binarization image have 0 and 255.
         *255 to 1 for simplicity */
        if(pix == 255){
          pix = 1;
        }
        //Writing pixel color(0 or 1)
        oneline += tostr(pix);
        if(w < width){
          oneline += ","; //comma
              separated
        }
```

```
59          }
60          ofs<<oneline<<endl;
61        }
62        cout<<i<<endl;
63        i++;
64      }
65      destroyAllWindows();
66  }
67
68  string tostr(int num){
69      stringstream ss ;
70      ss << num;
71      return ss.str();
72  }
```

Code 3 は Code 2 によって作成されたテキストファイルを元にして重複ピクセルをカウント、自己相関関数の計算を行うプログラムである。ベースフレームと比較するフレームの各ピクセル情報を比較し一致すればカウントを +1 するといった処理を行っている。基準とするベースフレームの数は 20 個、1 つのベースフレームに対して 501 フレーム (0.1 秒間分) の比較を行っている。保存する値は比較するフレーム数と同じ 501 個である。つまり、1 回目のベースフレームとの比較により 501 個分のカウント ($S(\Delta t) : S(0)$〜$S(500)$) ができるが、2 回目以降はこのカウントに累積させていくことになる。全てのフレームについて比較とカウントを行った後に正規化処理を行っている。正規化時点で 1 つのカウントには比較 20 回分の値が保存されている。これら 501 個分のカウントをベースフレーム同士の比較 $S(0)$ で除する。これにより $S(0) = 1$ 以降 1 以下の値が 500 個続く。これらの値を自己相関関数として出力する。

Code 3　重複ピクセルカウント

```
1   import numpy as np
2   import cv2
3   import os
4   import os.path
5   import csv
6   import sys
7   import datetime
8
9   #Two data -> 1(white) or 0(black)
10  def countone(path_input_dir,path_output_dir):
11      num_base = 20 #number of base file (
            Number of executions)
12      num_con = 500 #number of com
13      count_table = [] #count table
14
15      #Make count table
16      for tmp_counter in range(num_con):
17          count_table.append(0)
18
19      args = sys.argv
20
21      f_write = open(path_output_dir+args[1],'w'
            )
22
23      for base_count in range(num_base):
24          f_base = open(path_input_dir+str(
                base_count*30)+'.txt','r')
25          csvBase = csv.reader(f_base)
26          data_base = [v for v in csvBase]
27          i = 0
28          while(True):
29              path = path_input_dir+str(i+int(
                    base_count*30))+'.txt'
30              if(not os.path.exists(path) or i
                    >=500):
31                  #no File
32                  break
33              f_targ = open(path,'r')
34              csvRender = csv.reader(f_targ)
35              data_targ = [v for v in csvRender]
36              j = 0
37              count = 0
38              for row in data_targ:
39                  row_base = data_base[j]
40                  k = 0
41                  #One line
42                  for pix in row:
43                      #Count up if they match
44                      if(pix == args[2] and pix ==
                            row_base[k]):
45                          count = count + 1
46                      k = k + 1
47                  j = j+1
48              count_table[i] = count_table[i] + count
49              i = i+1
50          print(base_count)
51
52      for l in range(num_con):
53          #Normalization
54          result = str(l/5000) + '␣' + str(
                count_table[l]/count_table[0])+'\n'
55          f_write.write(result)
56
57      f_base.close()
58      f_targ.close()
59      f_write.close()
60
61  countone('out/two/txt/','number/')
62  cv2.destroyAllWindows()
```

1.3　相関関数の評価方法

2　測定方法

測定に用いるのは落下装置とハイスピードカメラである。落下装置はダンボール箱を加工したもので、光を通したり撮影したりするための穴や物体を充填する漏斗とストッパーで構成されている。装置とカメラの位置関係を図??に示す。2 つの距離は 70 cm とし、カメラに映っている範囲で物体の落下速度は約 2 m/s となるようにしている。

2.1　物体の種類

本研究で測定した物体は表 1 に示す 7 種類である。

表 1　実験に使用した物体

種類	直径 [mm]
金属球	6.2
BB 弾	6.0
発泡ポリスチレン	1.54
砂 (15〜20 メッシュ)	1
砂 (20〜30 メッシュ)	0.7
海砂 (150〜200 メッシュ)	0.08
ガラスビーズ	0.02-0.08
水 (白絵の具で着色)	-

表 1 で金属球、BB 弾、発泡ポリスチレンは実際に測定したものだが、砂と海砂については表記されたメッシュ値を参考におおよその直径を決めている。

カメラのシャッタースピードは 5000 fps($\frac{1}{5000}$秒に 1 回)、露光時間は $\frac{1}{80000}$秒 である。

本研究においてほとんどの物体は黒背景にして白色で抜き出している。これは物体にライトを当てることによって物体が白っぽく映るためであるが、金属球については反射によって色のむらができてしまうため、白背景にして後ろから光を当てることにより黒で抜き出している。

3　各物体の評価

3.1　相関関数

実験で得られた各物体の相関関数を下記にまとめる。なお、グラフ中の略称については表 2 を参照のこと。図 3 に 0.1 秒間の自己相関関数の変化を示す。砂や水といった物質は比較的相関が減少していないことがわかる。これらの物体は人の目から見ても帯状に連なって落ちていることがわかる。すなわち、粒ひとつひとつが独立して動くことがなく、なだらかな動きを見せるのである。このような場合撮影したどのフレームを見ても形は大きく変わっていない。

表 2　粉粒体の略称

種類	略称
金属球	Metal
BB 弾	BB
発泡ポリスチレン	EPS
砂 (15〜20 メッシュ)	sand1520
砂 (20〜30 メッシュ)	sand2030
海砂 (150〜200 メッシュ)	seasand
ガラスビーズ	glass
水 (白絵の具で着色)	water

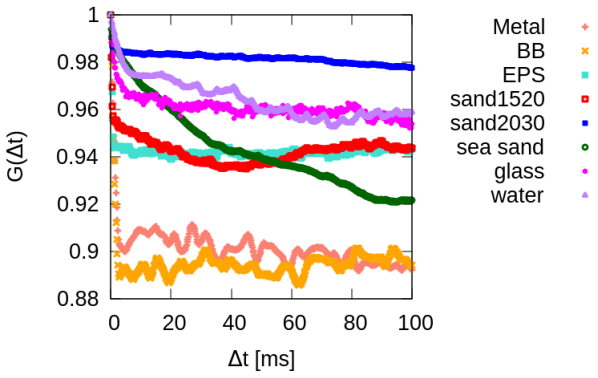

図 3　各物体の相関関数

多くの場合において初期状態 ($G(0) = 1$, 完全一致) から急激な減少が起こっていることがわかる。

$0 \leq \Delta t \leq 4[ms]$ を拡大したグラフが図 4 である。このグラフの結果から各粉粒体を 3 つのグループに分類した。第 1 グループは粒径 6 mm を超えるものである。金属球 (Metal) と BB 弾 (BB) がこれに当たる。グループ 1 は図 4 の範囲において 2 つの期間に分かれている。1 つは $0 \leq \Delta t \leq 2.5$ にある相関が減少する区間である。これを区間 I と呼ぶ。もう 1 つはほぼ相関が一定になる区間 II である。2 つの区間には明確な境界があり、2.5〜2.8 ms と見られる。また区間 II については他のグループよりも振動が大きい。振動が大きい理由としては球が離散体として運動するため、確率的にベースフレームとほとんど一致しないフレームがあるからである。

これよりも小さく粒径 1 mm 以上のものをグループ 2 とする。発泡ポリスチレン (EPS) と砂 15-20 メッシュ (sand1520) が含まれる。このグループについてもグループ 1 と同様 2 つの区間が見られるが、区間 I の時間がグループ 1 よりも短くなっている。境界は 1 ms あたりである。

最後に残りをグループ 3 とする。これらは粒径が 1 mm に満たない粒子径もしくは液体である。グループ 3 は特に海砂以下については区間 I が消失しているように見える。砂 (20-30 メッシュ) に関しては 0.5 ms 付近に境界があることが確認できるので、境界が見られない粉粒体については、より短い間隔で撮影を行うことで区間 I を発見することができると思われる。

区間 I と区間 II の境界を「相関持続時間 t_c」とする。目視の範囲でわかる t_c を表 3 にまとめる。

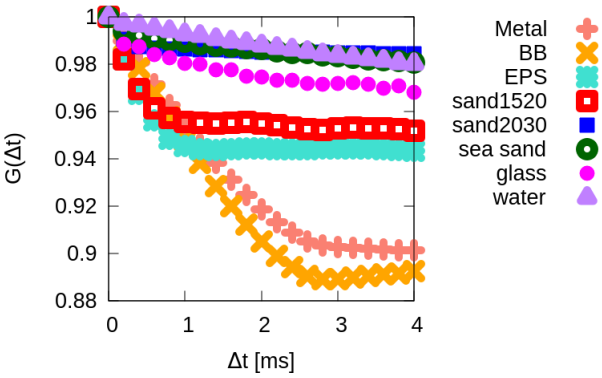

図 4　各物体の相関関数 (0.004 秒まで)

表 3　相関持続時間 (目視)

グループ	$t_c[ms]$
グループ 1	2.5-2.8
グループ 2	1
グループ 3(砂 (20-30 メッシュ))	0.5
グループ 3(その他)	-

3.2　微分を用いた相関持続時間 t_c の算出

前述したように相関持続時間 t_c は目視により大体の検討をつけることができるが、軸の縮尺等によって見え方は変わってくる。そこで、見た目という定性的な分析法ではなく、微分を用いて定量的に t_c を求めた。図 5 は自己相関関数を時間差で微分したグラフである。

自己相関関数は区間 I では負の傾きを持つが、区間 II は 0 に近い傾きで振動する。そのため微分 $\frac{dG(\Delta t)}{\Delta t}$ で 0 付近になるときの時間差 Δt が t_c と一致すると考えられる。

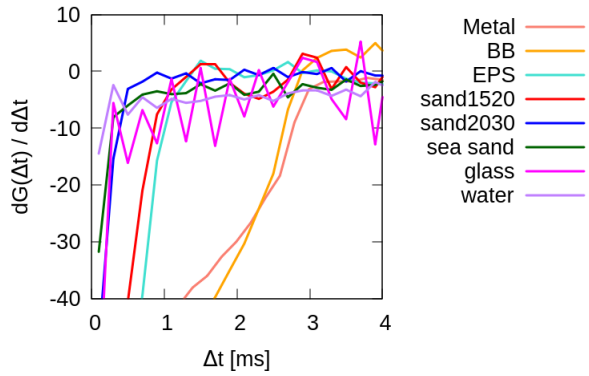

図 5　各物体の相関関数の微分

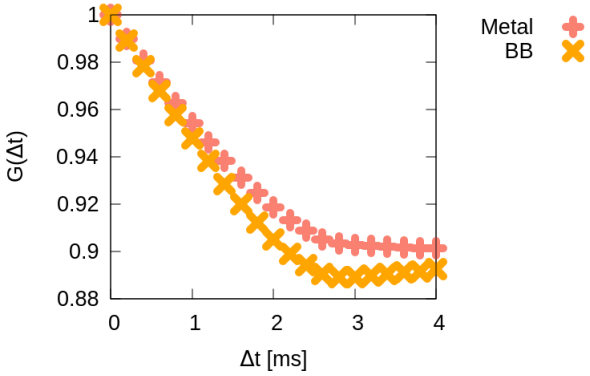

図 6　グループ 1 の自己相関関数

微分を用いる場合、傾きが-10 になるときの Δt を調べることで目視とおおよそ一致することがわかった。表 4 に各グループの微分値が-10 になるおおよその時間を示す。

表 4　相関持続時間

グループ	$t_c[ms]$(微分)
グループ 1	2.7
グループ 2	0.8
グループ 3(砂 (20-30 メッシュ))	0.4
グループ 3(その他)	0.2

3.3　グループ 1 の比較

グループ 1 は相関減少区間 (区間 I) と一定になる区間 (区間 II) が明確に分かれていることが特徴である。グループ 1 の自己相関関数を図 6 に示す。グループ 1 に属する金属球 (Metal, 6.2 mm) と BB 弾 (BB, 6.0 mm) で大きな差はみられない。これは粒径のさがあまりないためであると考えられる。測定した粉粒体の種類が少ないので完全に傾向を予想することはできないが、自己相関関数の概形は粒径にのみに影響され、種類にはよらないものと思われる。

3.4　グループ 2 の比較

グループ 2 の発泡ポリスチレンと砂 (15-20 メッシュ) は微分後の形を見ても明らかなように粒径が細かい砂の方が相関減少時間が短い。その結果砂のほうが区間 II 時点の値は大きい。このグループのように落下粉粒体を連続体として捉えることができる場合、相関減少は連続体の帯のヘリとその周辺の変化を表している。すなわち相関減少がより少ない砂は発泡ポリスチレンよりもヘリの変化が少ないということが言える。

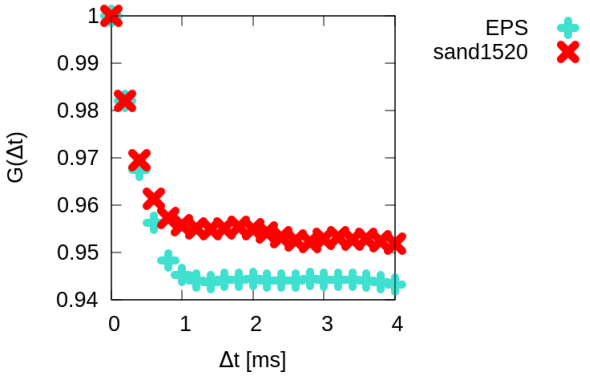

図 7　グループ 2 の自己相関関数

3.5　グループ 3 の比較

グループ 3 の自己相関関数の形はそれぞれ個性的な形をしている。まず、砂 (20-30 メッシュ) はグループ 1,2 の傾向を引き継ぎ、相関減少時間を 0.5 ms と短くし、以降相関をほぼ一定に保っている。この区間分けができるのは砂 (20-30 メッシュ) まで

となる。まず 1 つ細かい海砂には区間 Ⅱ が見られず、解析限界の 100 ms, G=0.92 まで緩やかに減少を続けている。60 ms 付近でグループ 2 の相関を追い抜いている。ただし減少速度は 8 つの中でも遅いため 4 ms という短い間隔で見れば典型的なグループ 3 として分類できる。ガラスビーズや水の相関には再び 2 つの区間が現れる。しかも 5〜6 ms というグループ 1 よりも長い時間減少する。ガラスビーズと水についても海砂と同じく相関は 0.95 以上という高い値を保っている。

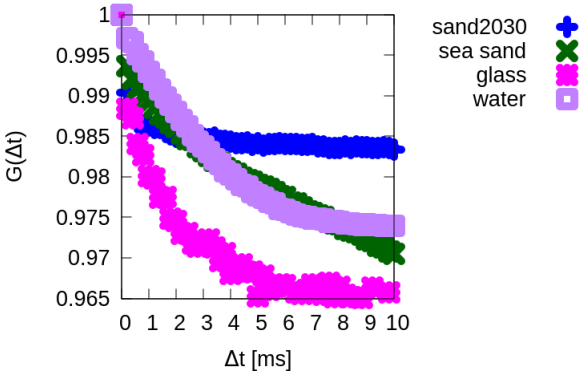

図 8　グループ 3 の自己相関関数

3.6　球 (離散体) と粉 (連続体) の相関減少の相違

グループ 3 における相関減少時間と相関減少速度から海砂以下について次の予測が立てられる。粒径が小さくなると一旦相関減少時間は長くなるということである。そもそもグループ 1 に代表される離散体の相関減少区間は、初期位置からの球 1 つ分の移動を表していた。これはグループ 2 から少し意味が変わってくる。グループ 2 より粒径が細かくなると映像に帯、すなわち連続体の部分が現れるのである。連続体部分の相関減少はヘリの小さな変化に起因している。つまりグループ 2 の相関減少は離散体と連続体両方の要素を含んでいる。そしてグループ 3 には連続体のみが写っている。連続体の変化は離散体よりもずっと小さいため相関減少速度も小さくなる。そのためグループ 1 よりも減少時間が長くなるものと考えられる。ただし、減少が緩やかな分区間 Ⅱ との区別がつきづらくなる。本研究では、減少後の相関値が高いため、全体的な傾向としてグループ 3 は相関減少時間が 0.5 ms 以下で曖昧または消失したものとしてまとめている。

4　透過な物体の測定方法検討

自己相関関数の計算には二値化画像処理が不可欠である。二値化する条件は被写体と背景の色のギャップが大きいことである。しかし、透明な物体は背景を透過するため二値化に十分なギャップを作ることができない。そこで、別な方法により二値化に似た処理を提案する。

今回注目したのは透明な物体が背景を透過するとき、完全に背景を映し出さないという性質である。つまり透明な物体であっても多くの場合背景が歪んで見え、その物体を認識することができる。これを利用して背景の歪みを検出するプログラムを作成した。この測定法には背景に方眼紙 (図 9) を用いる。

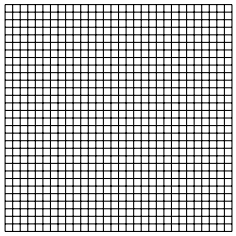

図 9　背景 (1 mm 方眼)

図 9 を背景にし、透明な物体を落下運動させる。この映像を二値化し、線分を検出する。この段階では背景や光の反射によって物体の色が安定しない。だが、物体がない場所はそのまま線が映っているため線が検出できるが、物体がある場所は線が見えなくなるため検出できなくなる。次に真っ白で大きさが映像のものと同じ画像を用意する。これに線分を検出した場所に沿って太い線を引く。すると背景部分の多くが黒く塗りつぶされるため擬似的に透明物体を白く抜き出すことができる。図 10 に透明物体の落下画像 (初期二値化済み) と疑似二値化画像を示す。

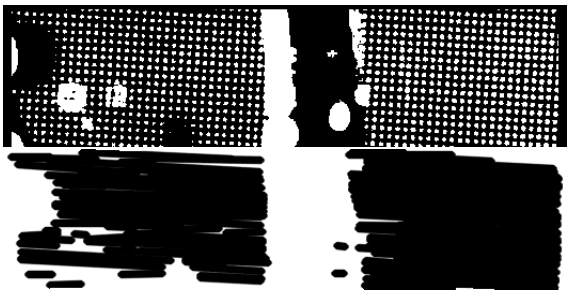

図 10　落下映像 (上) と疑似二値化画像 (下)

Code 4 に線分検出プログラムを示す。本プログラムは疑似二値化画像の作成を目的としている。関数 getline が実際に線分を検出している。getline ではまず対象画像を読み込んだ後、白色画像を作成している。次に線分を Hough 変換によって検出している。これにより線分の場所や長さを座標という形で得ることができる。この座標を元に白色画像に検出した線分の位置をなぞるように線分を描画する。この線分の太さを適当に太くすると線分の位置周辺が黒く塗りつぶされる。これを検出した線分全てに行う。線分が見えている周辺は透明物体が通っていない=背景なので、最終的に背景のおおよそが黒く塗りつぶされることになる。

Code 4　線分検出

```
#include<iostream>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/core/core.hpp>
#include<opencv2/features2d.hpp>
#include<opencv2/opencv.hpp>
#include<opencv2/imgproc.hpp>
#include<sys/stat.h>
#include<string>
#include<fstream>
#include<iostream>
#include<vector>
#include<sstream>

using namespace std;
using namespace cv;

void getline(string,string);
string tostr(int);

int main(){
  string twopath = "out/two/";
  string linepath = "out/line/";
  string path = "";
  int i = 0;
  struct stat st;

  while(!stat((path = twopath+tostr(i)+".png").c_str(),&st)){
    //Get line & Save
    getline(path,linepath+tostr(i)+".png");
    i++;
  }
  return 0;
}

void getline(string path_input,string path_output){
  //Original image
  Mat src_img = imread(path_input,1);
  Mat dst_img, work_img;
  dst_img = src_img.clone();
  int width = dst_img.cols;
  int height = dst_img.rows;
  //All white image
  Mat white(Size(dst_img.cols,dst_img.rows), CV_8UC3,Scalar::all(255));
  Mat chain(Size(dst_img.cols*2,dst_img.rows),CV_8UC3);
  cvtColor(src_img, work_img, CV_BGR2GRAY);
  //Get edge
  Canny(work_img, work_img ,80, 100, 3);

  vector<Vec4i> lines;
  //Get line
  HoughLinesP(work_img, lines, CV_HOUGH_PROBABILISTIC, CV_PI/180, 70, 6, 10);

  vector<Vec4i>::iterator it =lines.begin();
  for(;it != lines.end(); ++it){
    Vec4i l = *it;
    //Draw line in the black to all white image
    line(white, Point(l[0], l[1]), Point(l[2], l[3]),Scalar(0,0,0), 5, CV_AA);
  }

  //Save
  imwrite(path_output,white);
}

string tostr(int num){
  stringstream ss;
  ss << num;
  return ss.str();
}
```