SEM

Первая Группа

March 29, 2014

Сканирующая электронная микроскопия

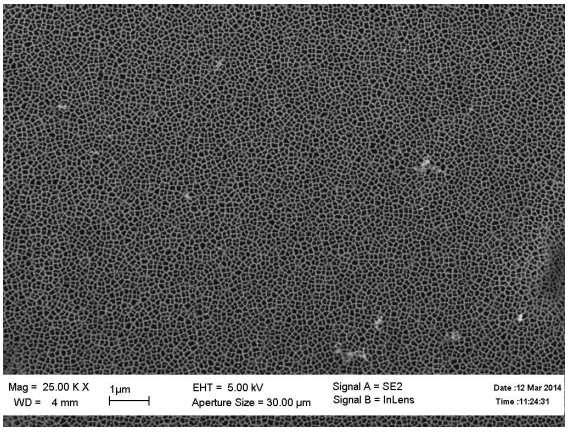
В работе были исследованы мембраны компании Whatman Anodisc (анодированный алюминий). С заявленными производителем были сопоставлены следующие параметры: толщина мембраны, диаметр пор, плотность пор. Зафиксированы также отклонение пор от цилиндричности и отклонение каналов от нормали к поверхности мембраны.

Введение

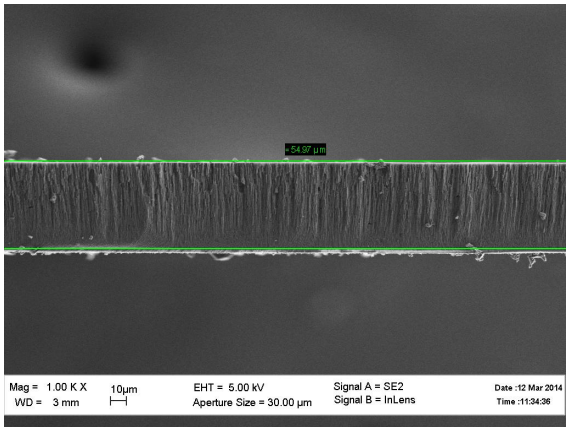
Мембраны анодированного алюминия могут использоваться в качестве матриц для формирования металлических нанопроволок и нанопроводов. При таком использовании являются важными особенности их геометрии, такие как форма, размер пор, наклон каналов и количество дефектов. В данной работе предметом исследования является мембрана компании Whatman Anodisc, предназначенная для использования как фильтра, со следующими заявленными усредненными параметрами:

- толщина мембраны 22 мкм
- диаметр пор 0.1 мкм
- плотность пор $2 \cdot 10^8$ см$^{-2}$

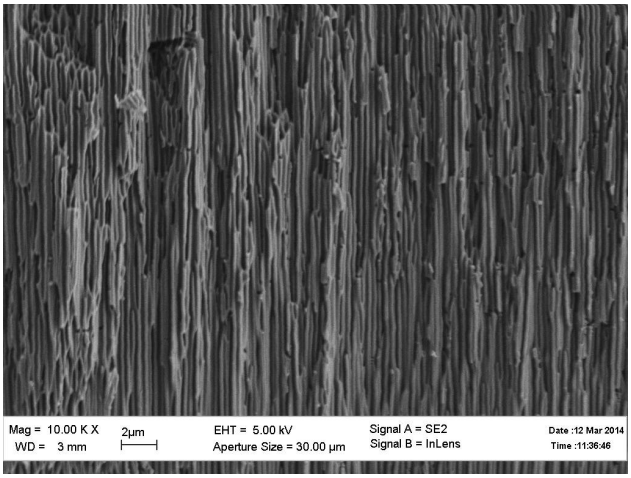
Результатом нашей работы является сравнение реальных характеристик мембраны с заявленными, а также определения качества полученных каналов для установления пригодности их к использованию в качестве матриц для выращивания металлических нанопроводов.



(a) Вид мембраны сверху



(b) Вид скола при меньшем увеличении



(c) Вид части скола в толще мембраны при большем увеличении

Рис. 1: Полученные снимки

1 Экспериментальная часть

1.1 Физический эксперимент

Так как в процессе сканирующей электронной микроскопии образец заряжается нам нужен проводящий слой на поверхности. В PVD-процессе был напылен тонкий слой золота на поверхность мембраны. Затем образец был помещен в микроскоп и была производена съёмка поверхностей. Фотографии можно видеть на Рис. 1

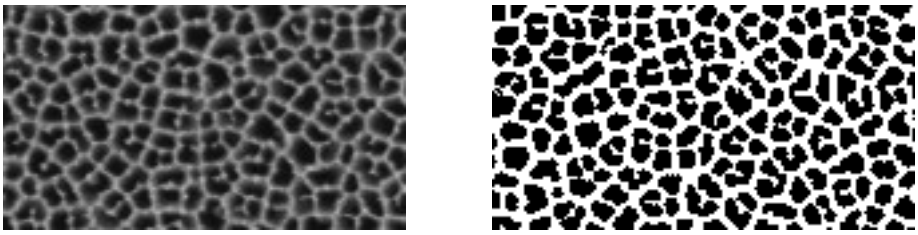
1.2 Данные

Результатом эксперимента являются полученные изображения мембраны. Для определения среднего диаметра пор и среднего угла их наклона в мембране с помощью этих картин были написаны программы на языке Python. Ниже приведены алгоритмы для каждой из задач. Для обеих задач необходима предварительная обработка изображения. Сначала выделяется область картинки, которая не содержит в себе данных о параметрах съемки и масштабе, имеющиеся внизу изображения. Затем происходит перебор пикселей. Те, значение которых больше некоторого порогового, закрашиваются белым цетом,

а остальные черным. Оптимальное пороговое значение подбирается опытным путём. Таким образом происходит выделение границы, и картинка становится контрастней.

Размеры пор

После обработки у нас имеется набор черных пикселей, которые являются внутренними по отношению к порам, и белые, которые принадлежат их границе. Начинаем последовательный перебор пикселей. Если находим чёрный пиксель, это значит, что мы нашли пору. Теперь находим все пиксели, принадлежащие этой же поре, и добавляем их в множество, соотвествующее этой поре. Это делается следующим образом: для первого черного пикселя проверяется, все ли его ближайшие соседи являются черными. Если да, то они добавляются в множество поры и закрашиваются серым, если нет, то вновь продолжается поиск черных пикселей, т.е новых пор, причем уже приписанные найденной поре пиксели уже точно выходят из повторного рассмотрения, т.к. мы закрасили их серым). В итоге мы получаем множество координат пикселей, принадлежащих каждой из найденным пор. Далее мы находим размер пор как диаметр множества или как корень из площади поры. Площадь поры вычисляется, как количество пикселей, принадлежащих ей, умноженное на площадь одного пикселя (размер пикселя известен).



(a) Элемент исходного изображения (b) Тот же элемент после установки порога

Рис. 2: Установка порога

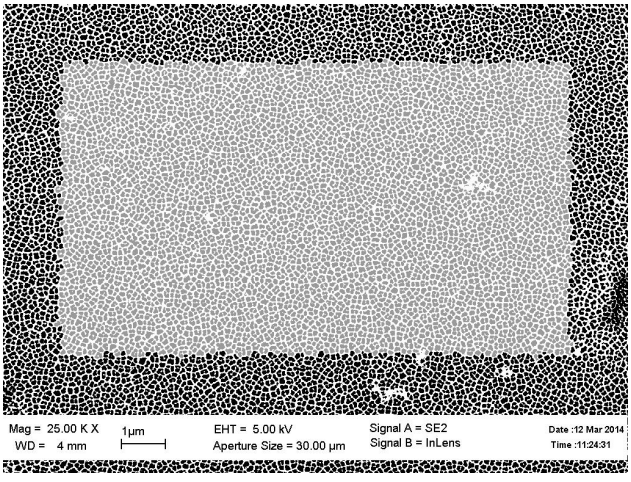


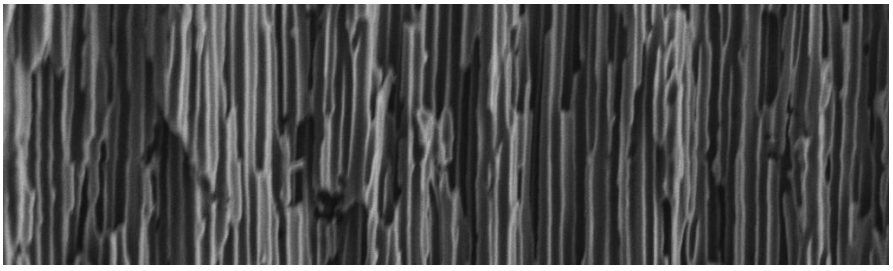
Рис. 3: Результирующее изображение

После работы программы остается массив объектов-клеток со всеми необходимыми данными. Результаты обработки в разделе “Результаты и обсуждение”

Наклоны каналов

В основе метода для решения данной задачи лежит преобразование Хафа (Hough Transform). Но предварительная подготовка в этом случае содержит в себе, кроме описанного выше, использование встро-

енного в Python Image Library фильтра поиска границ, иначе алгоритм будет работать неверно. Преобразование Хафа позволяет находить прямые в изображении и получать значения их параметров. После обработки на входе программы мы имеем набор черных пикселей, которые имеют значение интенсивности 0, и белых пикселей, большинство из которых лежат на неидеальных прямых, коэффициенты наклона которых мы хотим определить. Для любых двух белых точек картинки мы знаем их координаты в декартовой системе координат: (X1,Y1) и (X2,Y2). В этом пространстве любая прямая задается двум коэффициентами: углом наклона A и коэффициентом B, т.е. Y=AX+B. Но если в новом пространстве в качестве параметров, которыми задается прямая, мы возьмем не A и B, а X и Y, т.е B=-XA-Y. то в соответствие каждой из двух точек мы можем поставить по одной прямой в новом пространстве, называемом пространство Хафа (Hough space). Пересечение этих двух прямых в пространстве Хафа дает нам точку, координатами которой являются коэффициенты прямой, на которой лежат две данные точки в исходном пространстве. После последовательного перебора всех пар точек в пространстве Хафа мы получим множество прямых и их точек пересечения. Если нарисовать это множество точек пересечения, причем если для двух разных пар точка пересечения совпадает, закрашивать ее интенсивнее, то в итоге мы получим картину, на которой есть локальные максимумы интенсивности, причем самые яркие точки соответствуют искомым прямым. Интенсивность этих точек тем больше, чем лучше исходные точки лежат на одной прямой. Т.к. точки с левой и правой сторон картинки соответствуют разным прямым, то для оптимизации программы вся картинка разделяется на несколько частей ширины, в несколько раз превышающей максимальное расстояние между прямыми на картинке. Также учтено, что преимущественное направление прямых на фотографии вертикально, поэтому чтобы избежать обращения коэффициента A в бесконечность, предполагается поворот картины на 90 градусов. Также в целях оптимизации было решено отсекать те точки пересечения, которые соответствуют большим углам наклона искомых прямых (порог выбирается также эмпирически). Определение края и порога, а также разбиение на части, дали следующие результаты:



(a) Элемент исходной картинки Рис.1 (c)

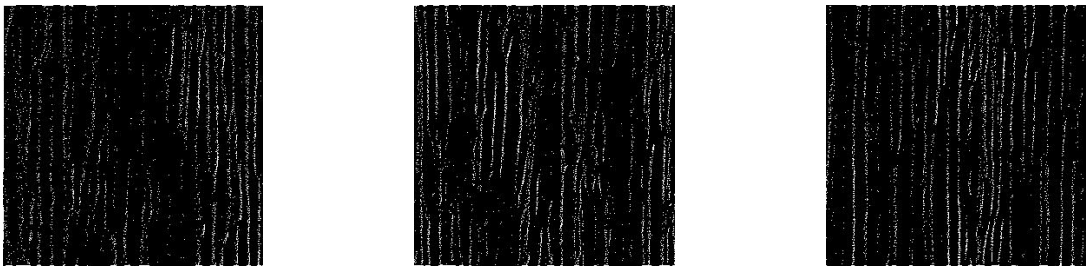


Рис. 4: Разбиение (на целое число отрезов заданной ширины) и предварительная обработка изображения для Hough transform

В пространстве Хафа при анализе второго (среднего) отрезка фотографии получено следующее распределение интенсивностей:

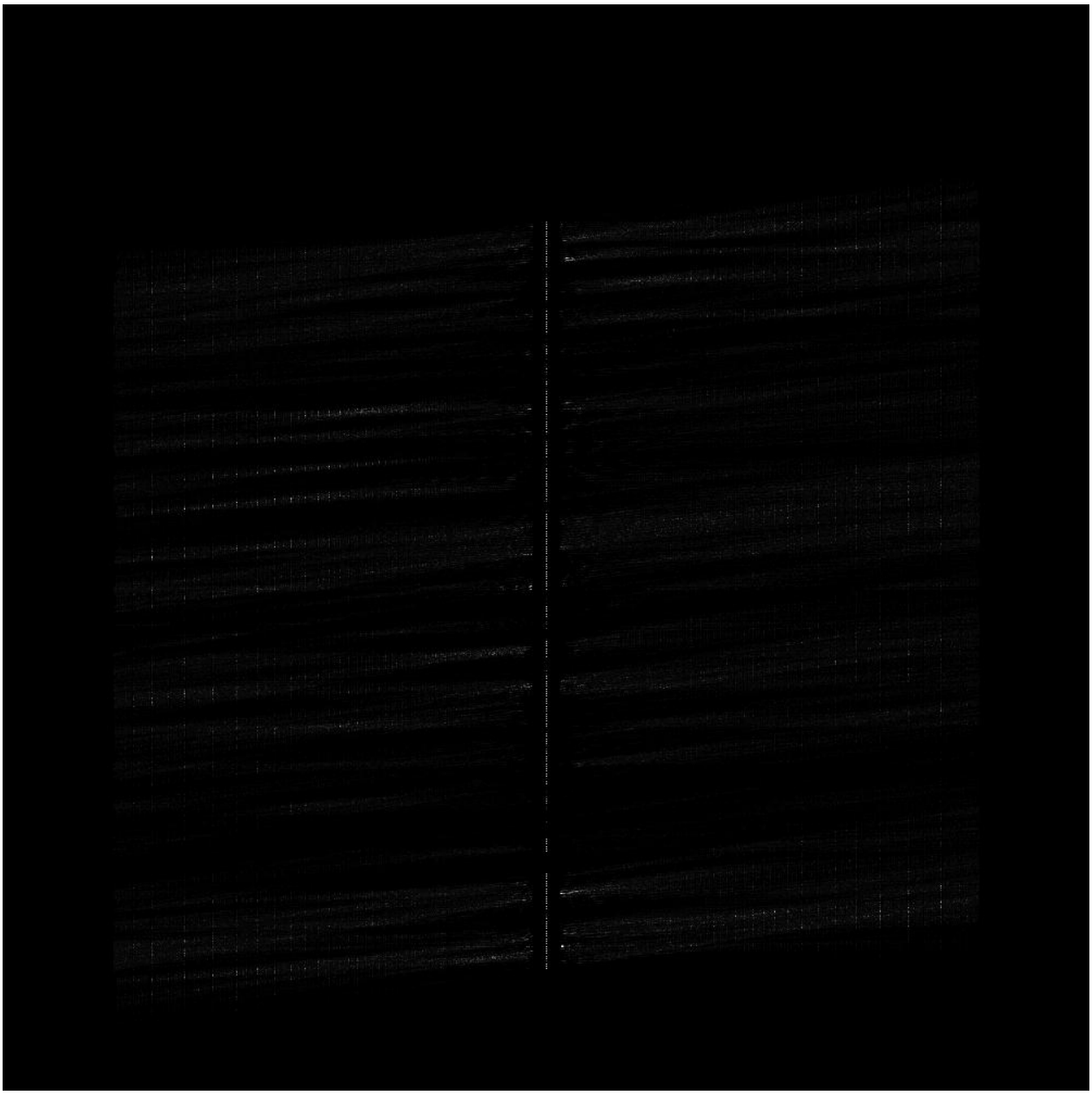
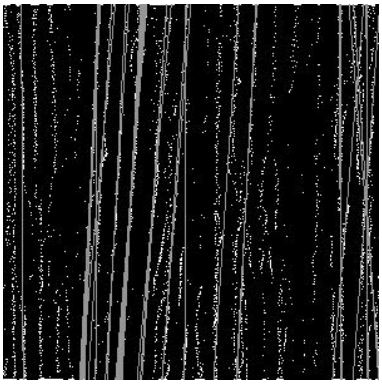
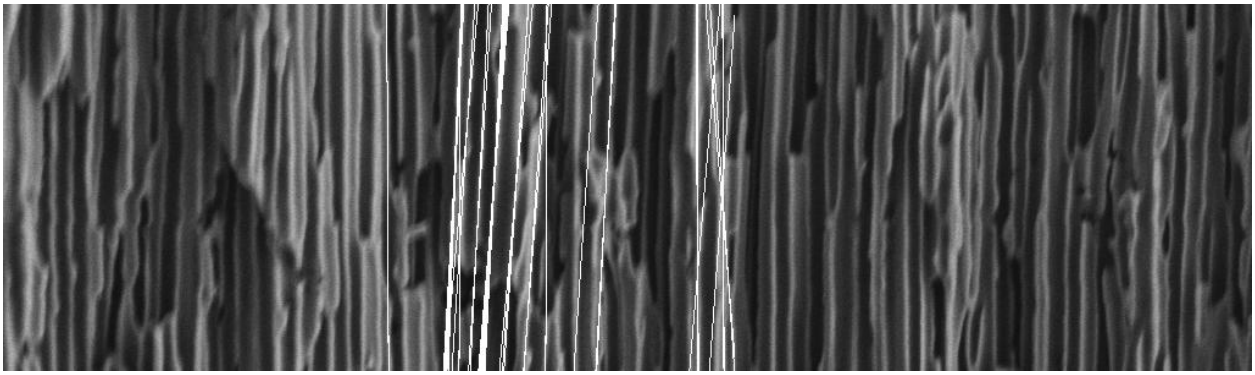


Рис. 5: Распределение интенсивностей в пространстве Хафа

Это распределение позволило определить на фотографии следующие прямые:



(a) Обнаруженные прямые на обработанной части изображения



(b) Обнаруженные прямые на оригинале

Коэффициенты всех обнаруженных прямых после работы программы сохраняются в памяти компьютера, поэтому возможна любая дальнейшая статистическая обработка.

2 Результаты и обсуждение

2.1 Размеры пор

В результате анализа данных были получены: гистограмма распределения размеров пор, усредненные размеры пор двумя способами (корень из площади и диаметр), характерное отклонение от цилиндричности по разности вычисленных характерных средних размеров, плотность пор.

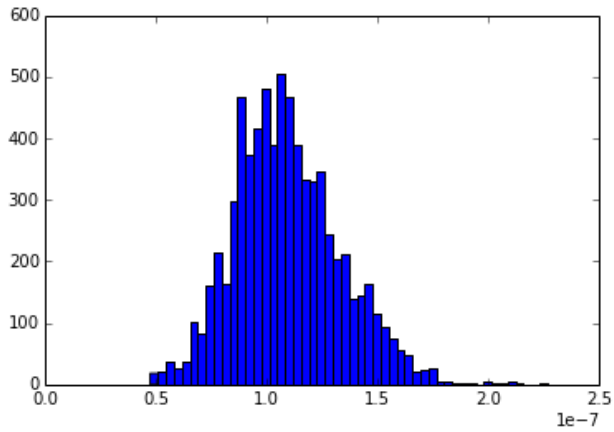


Рис. 7: Гистограмма распределеня пор по размерам

Заявленный средний размер хорошо совпадает с рассчитанным - 0.11 ± 0.05 мкм. В тоже время в плотности пор обнаружены существенные расхождения: заявленное значение в $2 \cdot 10^8$ см$^{-2}$ отличается от вычисленного $4 \cdot 10^9$ см$^{-2}$ в 20 раз. Характерное отклонение от цилиндричности составило $-2.85 \cdot 10^{-8}$ мкм, то есть в среднем преобладает вытянутая форма пор (поры вытянуты на 29%).

2.2 Толщина мембраны

Измеренная толщина мембраны составила 57 мкм, что почти в три раза отличается от заявленной 22 мкм.

2.3 Каналы

По результатам работы программы получено среднее по модулю отклонение от вертикали в 2° в толще мембраны.Имеется также тенденция к искривлению каналов относительно прямых линий. О количестве дефектов сложно судить в силу наличия большого количества механических повреждений на сколе.

2.4 Выводы

Имеются существенные отклонения от заявленных параметров. Ошибки с нашей стороны быть не может, возможно, измерялся фильтр другой модели нежели чем предполагалось.

Мембрана, скорее всего, непригодна для выращивания нанопроводов. Качество каналов как и в плане цилиндричности, так и в плане постоянства наклона, недостаточно, на наш взгляд, для формирования проводов длины, сравнимой с толщиной пластины.

3 Приложение

Здесь представлены процедуры работы с программами, написанными нами. Исодный код можно найти по адресу https://github.com/vdrhtc/CellDistinguisher

3.1 Cell distinguisher

In [1]:
```
%pylab inline
import sys
sys.path.append('src')
from PIL import Image, ImageFilter
from cell_distinguisher.distinguisher import *
from line_distinguisher.distinguisher import *
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [2]:
```
pixel_size = 1e-6/70
```

In [3]:
```
im = Image.open('1_01.tif')
im.show()
```

In [4]:
```
altered_im = get_cells_from_image(im, (100, 100) , (im.getbbox()[2]-100, im.getbbox()[3]-200), pixel_size, 80)
```

In [143]:
```
altered_im.show()
```

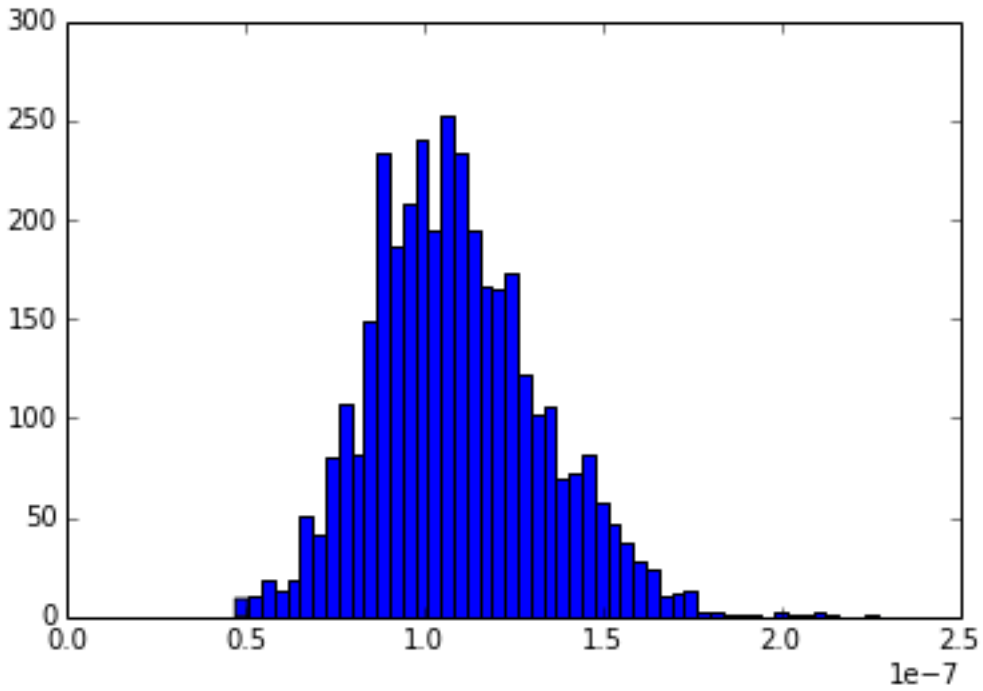
In [5]:
```
len(cells)
```

Out [5]: 3626

In [11]:
```
sizes = [cell.first_norm() for cell in cells]
sizes1 = [cell.second_norm() for cell in cells]
deltas = map(lambda x, y: x*math.sqrt(4/math.pi)-y, sizes, sizes1)
```

In [13]:
```
plt.hist(sizes, 50)
print("Средний рамер поры по корню из площади: %.2e"% mean(sizes))
print("Средний рамер поры по диаметру: %.2e"% mean(sizes1))
print("Среднее отклонение от цилиндричности: %.2e"% mean(list(deltas)))
print("Плотность пор: %.2e"%(len(cells)/((im.getbbox()[2]-200)*(im.getbbox()[3]-300)*(pixel_size*100)**2)))
```

Средний рамер поры по корню из площади: 1.09e-07
Средний рамер поры по диаметру: 1.52e-07
Среднее отклонение от цилиндричности: -2.85e-08
Плотность пор: 4.61e+09



In [124]:
```
import sys
sys.path.append('src')
from PIL import Image, ImageFilter
from cell_distinguisher.distinguisher import *
from line_distinguisher.distinguisher import *
%pylab inline
```

Populating the interactive namespace from numpy and matplotlib

WARNING: pylab import has clobbered these variables: ['copy']
'%pylab --no-import-all' prevents importing * from pylab and numpy

3.2 Line distinguisher

In [133]:
```
im1 = Image.open('./1_10.tif').crop((0, 300, 1024, 600))
im1.show()
```

In [47]:
```
slices = prepare_image(im1, threshold=80, slice_thickness=300)
```

In [56]:
```
slices[2].show()
```

In [67]:
```
coeffs = get_lines(slices[1], tolerance_a_b=(4, 1), max_a=0.1)
```

Getting coefficients...
Processing 4575 points.
485186, 4574

In [68]:
```
max_coeff = max(list(coeffs.values()))
```

In [109]:
```
image = Image.new("L", (1000, 1000))
interesting_points = []

column = 0
for point in coeffs.keys():
    level = max_coeff

    if coeffs[point] > max_coeff/130 and point[0]!=0:
        interesting_points.append(point)

        if column != 2:
            print(point, end = '        ')
            column+=1
        else:
            print(point)
            column = 0

    image.putpixel((int(point[0]*4e3)+500, int(point[1]*2.3)+200), int(coeffs[point]/level*255*90))
# image.show()
```

(-0.0625, 111.6) (0.0037, 268.9) (-0.0714, 300.6)
(-0.0833, 287.8) (-0.05, 77.5) (-0.0036, 146.0)
(0.0039, 268.9) (0.0068, 268.0) (-0.0417, 76.1)
(-0.0714, 113.4) (-0.0667, 130.7) (0.0038, 290.0)
(0.0035, 290.0) (-0.0667, 188.0) (-0.04, 75.8)
(-0.0667, 147.1) (0.0667, 279.7) (-0.0714, 130.9)
(-0.0769, 149.8) (-0.0038, 75.1) (-0.0039, 75.1)
(-0.0667, 112.5) (0.0041, 268.9) (-0.0455, 76.9)

```
(-0.0435, 76.5)      (-0.0909, 89.5)      (-0.0435, 77.0)
(-0.05, 77.0)        (-0.0455, 76.7)      (-0.0714, 113.1)
(-0.0476, 76.8)      (-0.05, 201.7)       (-0.05, 110.0)
(-0.0455, 76.4)      (-0.0357, 75.7)      (-0.0526, 77.5)
(0.004, 289.9)       (-0.0714, 112.9)     (0.0036, 268.9)
(-0.0556, 110.7)     (0.0037, 289.9)      (-0.0909, 134.2)
(-0.05, 109.8)       (0.0035, 269.0)      (0.0833, 276.0)
(-0.0769, 113.8)     (-0.0588, 110.9)     (-0.0417, 75.8)
(-0.0417, 76.2)      (-0.0037, 146.0)     (-0.0625, 100.4)
(0.0034, 269.0)      (-0.05, 77.2)        (-0.0625, 86.8)
(0.004, 268.9)       (-0.0833, 114.8)     (-0.0556, 77.8)
(0.0039, 289.9)      (0.0047, 14.7)       (0.0036, 269.0)
(0.0048, 14.7)       (0.0038, 268.9)      (0.0037, 290.0)
(-0.0556, 202.6)     (0.0037, 269.0)      (-0.05, 98.0)
(0.0036, 290.0)      (0.0038, 289.9)      (-0.0625, 111.3)
(-0.0714, 112.6)
```

In [117]:
```
copy = slices[1].copy()
```

In [113]:
```
for interesting_point in interesting_points:
    for y in range(0, copy.getbbox()[3]):
#         print(interesting_point, int(interesting_point[0]*y+interesting_point[1]), y)
        if int(interesting_point[0]*y+interesting_point[1]) > 0 and
int(interesting_point[0]*y+interesting_point[1]) < copy.getbbox()[2]:
            copy.putpixel((int(interesting_point[0]*y+interesting_point[1]), y), 150)
copy.show()
```

In [121]:
```
copy_1 = im1.copy()
```

In [122]:
```
for interesting_point in interesting_points:
    for y in range(0, copy.getbbox()[3]):
#         print(interesting_point, int(interesting_point[0]*y+interesting_point[1]), y)
        if int(interesting_point[0]*y+interesting_point[1]) > 0 and
int(interesting_point[0]*y+interesting_point[1]) < copy.getbbox()[2]:
            copy_1.putpixel((int(interesting_point[0]*y+interesting_point[1]+300), y), 255)
copy_1.show()
```