Supplementary Material for:

New Image Texture Analysis, and Application to Polymer Membrane Surface Morphologies and Roughness

Clifford S. Todd1\* and William A. Heeschen1§

1 The Dow Chemical Company, 1897 Building, Midland MI 48667 USA

§ Retired

|  |  |
| --- | --- |
|  |  |
|  |  |

Figure S1. Sample #1, Mixed morphology.

|  |  |
| --- | --- |
|  |  |
|  |  |

Figure S2. Sample #2, Ridge & Valley morphology.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Figure S3. Sample #3, Nodular morphology.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Figure S4. Sample #4, Mixed morphology. The right half of the bottom image was not used for image analysis.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Figure S5. Sample #5, Mixed morphology.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Figure S6. Sample #6, Nodular morphology.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Figure S7. Sample #7, Ridge & Valley morphology.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Figure S8. Sample #8, Ridge & Valley morphology.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Figure S9. Sample #9, Ridge & Valley morphology.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Figure S10. Gray level histograms of membrane SEM images. For Gray Level scale, black is to the left, white to the right.

### Surface Roughness Measurement

AFM image processing was performed using Image Metrology SPIP software v5.0.2. Second-order polynomial plane fitting, with mean z height set to zero, was performed on the height images prior to performing roughness calculations. Average Roughness (Sa) is defined as the arithmetic average of the absolute values of the surface height deviations measured from the mean data plane, expressed as , where: *zj* is the surface height deviation measured from the mean data plane for a given pixel; *n* is the number of pixels in the image. Root mean squared average roughness (Sq) is defined as the RMS average of height deviations taken from the mean data plane, expressed as. Surface area difference (Sdiff) is defined as the percentage difference between an image’s three-dimensional surface area and its two-dimensional footprint area.

|  |  |  |  |
| --- | --- | --- | --- |
| (a) Original | (b) Higher Brightness | | (c) Lower Contrast & Brightness |
| (d) | | (e) | |
| (f) | | (g) | |
| (h) | | (i) | |

Figure S11. (a) Original SEM image of #7. (b) Brightness increased. (c) Contrast lowered and brightness decreased. (d) Mean of the Standard Deviation gray level as a function of domain size raw calculations. (e) Normalized Mean of the Standard Deviation. (f) Standard Deviation of the Standard Deviations. (g) Normalized Standard Deviation of the Standard Deviations. (h) Standard Deviation of the Mean. (i) Normalized Standard Deviation of the Mean.

Table S1. K Nearest Neighbor analysis for K=3.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Actual | Predicted | |
| Nodular | Ridge & Valley |
| Training set | Nodular | 12 | 2 |
| Ridge & Valley | 1 | 25 |
| Validation set | Nodular | 5 | 0 |
| Ridge & Valley | 0 | 12 |

|  |  |  |  |
| --- | --- | --- | --- |
| 06-01 | 06-03 | 06-04 | 06-05 |
| 06-06 | 06-07 | 06-08 | 06-09 |
| 06-10 | 06-11 | 13-1 | 13-2 |
| 13-3 | 13-4 | 13-6 | 13-7 |

Figure S12. Example SEM images of the 16 samples used to calibrate surface roughness as a function of image texture.

Table S2. Surface roughness measurements.

|  |  |  |  |
| --- | --- | --- | --- |
| **Sample** | **Sa (nm)** | **Sq (nm)** | **Sdiff (%)** |
| 06-01 | 75.0 ± 5.8 | 96.4 ± 8.5 | 108.0 ± 5.4 |
| 06-03 | 78.0 ± 21.8 | 104.8 ± 30.7 | 91.0 ± 13.3 |
| 06-04 | 75.2 ± 8.4 | 103.6 ±15.3 | 82.7 ± 1.2 |
| 06-05 | 58.4 ± 12.1 | 84.0 ±19.7 | 67.5 ± 11.4 |
| 06-06 | 93.8 ± 35.1 | 133.6 ± 54.6 | 92.1 ± 15.9 |
| 06-07 | 82.4 ± 31.2 | 115.7 ± 36.8 | 62.3 ± 5.6 |
| 06-08 | 102.7 ± 37.5 | 137.7 ± 47.0 | 68.4 ± 9.0 |
| 06-09 | 105.9 ± 62.8 | 140.5 ± 79.1 | 62.3 ± 12.7 |
| 06-10 | 61.0 ± 7.4 | 82.1 ± 14.1 | 56.3 ± 1.1 |
| 06-11 | 69.7 ± 12.1 | 93.3 ± 20.1 | 60.9 ± 2.5 |
| 13-1 | 47.3 ± 3.7 | 60.2 ± 4.7 | 91.1 ± 2.6 |
| 13-2 | 7.4 ± 2.1 | 9.6 ± 2.9 | 11.5 ± 5.9 |
| 13-3 | 19.9 ± 1.8 | 25.4 ± 2.4 | 24.4 ± 2.1 |
| 13-4 | 24.5 ± 3.9 | 38.4 ± 10.2 | 45.9 ± 4.7 |
| 13-6 | 41.3 ± 1.2 | 52.6 ± 1.6 | 63.9 ± 1.7 |
| 13-7 | 42.8 ± 1.6 | 54.5 ± 2.3 | 61.3 ± 3.3 |

Table S3. Fit equations and statistics for surface roughness as a function of principal component scores.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Fit Equation | | | | RASE Training | RASE Validation |
| Metric | Intercept | PC1 | PC2 | Rsquare |
| Sa | 48.5 | 86.3 | -39.2 | 0.590 | 17.8 | 18.5 |
| Sq | 65.8 | 118.9 | -48.7 | 0.594 | 24.2 | 25.0 |
| Sdiff | 57.6 | 38.4 | -39.9 | 0.187 | 21.3 | 20.9 |

|  |  |
| --- | --- |
|  |  |

Figure S13. Sa and Sdiff predicted from texture analysis.

# Plugins and Macros for Image Analysis

## ImageJ Plugin to calculate gray-level statistics

The following text is a Plugin written for ImageJ to accomplish gray-level calculations.

/\*\* This plug-in implements the texture analysis described in Todd & Heeschen. \*/

import ij.\*;

import ij.process.\*;

import ij.gui.\*;

import java.awt.\*;

import ij.text.\*;

import ij.plugin.filter.\*;

public class Texture\_MeanSD implements PlugInFilter {

public static final String VERSION="2.00b005b\_02Feb09";

static final String CANCELLED="Cancelled";

static String arg;

static ImagePlus imp;

//These need to be class variables to be remembered through separate calls

static int boxMinDim=2;

static int numDecRpt=3;//optional dialog

static boolean doAutoAR=false;//optional dialog

//output values - public and static in case they need to get pulled out by another plugin

public static float[] sd\_mean,sd\_sd,mean\_sd;

static boxDef box;

static long[] sumZ,sumZ2,rawCount;//values for current box

static double meanZ,sdZ;//used for local calc in box

static double[] sumMeanZ,sumMeanZ2,sumSDZ,sumSDZ2;//values for level

public int setup(String arg, ImagePlus imp) {

this.arg = arg;

this.imp = imp;

if (arg.equals("about"))

{showAbout(); return DONE;}

return DOES\_8G+NO\_CHANGES;

}

public void run(ImageProcessor ip) {

IJ.write(textureStr(ip));

}

public static String doTexture(String paramString){

//code to get image processor based on paramString. If text, then find window title. If negative number, get window ID.

ImageProcessor myIP;

try{//assume it is an integer

int myID;

myID=Integer.parseInt(paramString);//throws exception if not an integer

IJ.selectWindow(myID);

imp = IJ.getImage();

myIP=imp.getProcessor();

}

catch(Exception e) {//now assume the string is valid

IJ.selectWindow(paramString);

imp = IJ.getImage();

myIP=imp.getProcessor();

}

return textureStr(myIP);

}

/\*\*textureStr is the heart of the method. It returns the texture report as a string.\*/

static String textureStr(ImageProcessor ip){

if (IJ.shiftKeyDown()) {if (!getParameters()) return CANCELLED;}

//get width, height and the region of interest

int tInt=0;

double tDouble=0.0;

int wide = ip.getWidth();//this is the offset in the pixel vector

int high = ip.getHeight();

box = new boxDef(boxMinDim,wide,high,doAutoAR);

sumZ = new long[box.levels];/\*\*sums for current box\*/

sumZ2 = new long[box.levels];

rawCount = new long[box.levels];/\*\*counts for current box\*/

meanZ=0.0; sdZ=0.0;

sumMeanZ = new double[box.levels];

sumMeanZ2 = new double[box.levels];

sumSDZ = new double[box.levels];

sumSDZ2 = new double[box.levels];/\*\*sums for level\*/

sd\_mean = new float[box.levels];

sd\_sd = new float[box.levels];

mean\_sd = new float[box.levels];

long measPix=box.xDim\*box.yDim\*(1<<(box.levels-1))\*(1<<(box.levels-1));

// create an array of the pixels from the original image

byte[] pixels = (byte[]) ip.getPixels();

long currZ,currZ2;

int x,y,pos;

long pCounter =0;

long boxBasePix = box.xDim\*box.yDim;

while (pCounter < measPix){

for (y=0; y<box.yDim; y++) {

pos=box.globXOff + (box.globYOff+y)\*wide;

for (x=0; x<box.xDim; x++) {

currZ = (long)(pixels[pos]&0xff);

sumZ[0] += currZ;

sumZ2[0] += currZ\*currZ;

pos++;

}

}

pCounter += boxBasePix;

rawCount[0] = boxBasePix;

meanZ = ((double)sumZ[0])/((double)boxBasePix);

tDouble = ((double)(((long)boxBasePix)\*sumZ2[0] - (sumZ[0]\*sumZ[0])))/(double)(boxBasePix\*(boxBasePix-1));

if (tDouble>0.0) sdZ = Math.sqrt(tDouble);

else sdZ = 0.0;

sumMeanZ[0] += meanZ;

sumMeanZ2[0] += meanZ\*meanZ;

sumSDZ[0] += sdZ;

sumSDZ2[0] += sdZ\*sdZ;

box.count[0]++;

sumZ[1] += sumZ[0];//transfer data up to next level

sumZ2[1] += sumZ2[0];

rawCount[1] += rawCount[0];

sumZ[0] = 0; sumZ2[0] = 0; rawCount[0] = 0;//prepare for next count

tInt = (int)(pCounter/((long)boxBasePix));

//=how many base boxes have been counted.

nextBox(tInt,1);

//level 1 (index=0) is handled as a special case, so start calling for level 2 (index=1)

}

//Have all raw data, now reduce to report

String space = " ";

String tab = "\t";

String newLine="\n";

String rptString=imp.getTitle()+newLine;

rptString=rptString+"MeasROI\_XxY"+tab+IJ.d2s(box.roiXDim,0)+tab+IJ.d2s(box.roiYDim,0)+tab+"StructElem\_XxY"+tab+IJ.d2s(box.xDim,0)+tab+IJ.d2s(box.yDim,0)+newLine;

rptString=rptString+"level"+tab+"boxCount"+tab+"mean\_sd"+tab+"sd\_sd"+tab+"sd\_mean";

for (int ii=0;ii<box.levels;ii++) {

if (box.count[ii]<2) {

float imageMean = (float)sumMeanZ[ii];

float imageSD = (float)sumSDZ[ii];

mean\_sd[ii] = imageSD;

sd\_sd[ii] = 0;

sd\_mean[ii] = 0;

rptString=rptString+newLine+IJ.d2s(ii+1,0)+tab+IJ.d2s(box.count[ii],0)+tab+"-1"+tab+IJ.d2s(imageSD,numDecRpt)+tab+IJ.d2s(imageMean,numDecRpt);

}

else {

mean\_sd[ii] = (float)(sumSDZ[ii]/(double)box.count[ii]);

tDouble = (((double)box.count[ii])\*sumSDZ2[ii] - (sumSDZ[ii]\*sumSDZ[ii]))/(double)(box.count[ii]\*(box.count[ii]-1));

if (tDouble>0.0) sd\_sd[ii] = (float)Math.sqrt(tDouble);

else sd\_sd[ii] = 0;

tDouble = (((double)box.count[ii])\*sumMeanZ2[ii] - (sumMeanZ[ii]\*sumMeanZ[ii]))/(double)(box.count[ii]\*(box.count[ii]-1));

if (tDouble>0.0) sd\_mean[ii] = (float)Math.sqrt(tDouble);

else sd\_mean[ii] = 0;

rptString=rptString+newLine+IJ.d2s(ii+1,0)+tab+IJ.d2s(box.count[ii],0)+tab+IJ.d2s(mean\_sd[ii],numDecRpt)+tab+IJ.d2s(sd\_sd[ii],numDecRpt)+tab+IJ.d2s(sd\_mean[ii],numDecRpt);

}

}

return rptString;

}

/\*\* nextBox is called recursively to build the upper level data sets as they become valid \*/

static void nextBox(int boxNum,int level){

double tDouble = 0.0;

int nextLevel = level+1;

if (level>1) boxNum = boxNum>>2;//assume recursive calls for higher levels

//adjust global offsets for next box

switch (boxNum & 0x0000000003){

case 1: box.globXOff += box.xOffset[level]; break;

case 2: box.globYOff += box.yOffset[level]; break;

case 3: box.globXOff -= box.xOffset[level]; break;

case 0:{

box.globYOff -= box.yOffset[level];// get back to starting point

//summarize data for the current box at this level

meanZ = (double)sumZ[level]/(double)rawCount[level];

tDouble = (double)(rawCount[level]\*sumZ2[level] - sumZ[level]\*sumZ[level])/(double)(rawCount[level]\*(rawCount[level]-1));

if (tDouble>0.0) sdZ = Math.sqrt(tDouble);

else sdZ = 0.0;

sumMeanZ[level] += meanZ;

sumMeanZ2[level] += meanZ\*meanZ;

sumSDZ[level] += sdZ;

sumSDZ2[level] += sdZ\*sdZ;

box.count[level]++;

if (nextLevel<box.levels){

sumZ[nextLevel] += sumZ[level];//transfer data up to next level

sumZ2[nextLevel] += sumZ2[level];

rawCount[nextLevel] += rawCount[level];

nextBox(boxNum,nextLevel);

}

//reset for next box at this level

sumZ[level] = 0;sumZ2[level] = 0;rawCount[level] = 0;

break;

}

}

}

static boolean getParameters(){

boolean plotResults =false;

GenericDialog gd = new GenericDialog("Processing Setup", IJ.getInstance());

gd.addNumericField("Minimum Cell Dimension (pixels):",boxMinDim,0);

gd.addCheckbox("Auto-Adjust Aspect Ratio", doAutoAR);

gd.addCheckbox("Plot Results", plotResults);

gd.addNumericField("Number of Decimals to Report:",numDecRpt,0);

gd.showDialog();

if (gd.wasCanceled())

return false;

int tInt=(int)gd.getNextNumber();

if ((tInt < 2)||(tInt > 64)) {

IJ.error("Minimum Box Size out of range (2 <= size <= 64");

return false;

}

boxMinDim = tInt;

//auto aspect ratio flag

doAutoAR = gd.getNextBoolean();

//plot results flag

plotResults = gd.getNextBoolean();

if (plotResults) IJ.showStatus("Results should be plotted!");

else IJ.showStatus("Results won't be plotted!");

//number of decimal places to keep in report

tInt = (int)gd.getNextNumber();

if ((tInt < 0)||(tInt > 10)) {

IJ.error("Reported decimals out of range (0 <= decimals <= 10");

return false;

}

numDecRpt = tInt;

return true;

}

void showAbout() {

IJ.showMessage("Mottle Analysis","Analyzes Mottle and Appearance");

}

}

class boxDef {//index for level 1 is 0 and level 12 is 11

double ar11=1.0;//convention is wide x high

double ar43=4.0/3.0;

double ar32=3.0/2.0;

int defMaxLevels=13;//allows full processing of a 4MB x 4MB image @ 2x2 elements

public int levels;

public int globXOff, globYOff;

public int xDim, yDim;

public long[] count;

public int[] xOffset, yOffset;

public int roiXDim, roiYDim;//actual analyzed area in pixels

public boxDef(int minDim, int xSide, int ySide, boolean autoAR, int maxLevels) {

int minSide=xSide, maxSide=minSide;

int maxDim=0;

xDim=yDim=maxDim=minDim;//for now, set these identical to minimum

if(maxLevels<1) maxLevels=defMaxLevels;

globXOff=globYOff=0;//re-initialize

if(autoAR){//use this option to optimize structuring element, otherwise it's a square

int type=0;//default type is square. 1==4x3,2==3x2

double ratio=((double)xSide)/((double)ySide);

double factor=ar11;//1:1

if((ratio>=ar32)||(ratio<=(1.0/ar32))) factor=ar32;//3:2

else if((ratio>=(ar43/1.0))||(ratio<=(1/ar43))) factor=ar43;//4:3

minDim--;//seed do/while with 1 pixel smaller

double tReal=0.0;

do {

minDim++;

tReal=(double)minDim\*factor;

} while ((tReal%1)>0);

maxDim=(int)tReal;

if(xSide<=ySide){

minSide=xSide;

xDim=minDim;

yDim=maxDim;

}

else {

minSide=ySide;

xDim=maxDim;

yDim=minDim;

}

}

else if(xSide<=ySide){minSide=xSide;}

else {minSide=ySide;}

int test = minSide/minDim;

int tInt=0;

while (test>0){

test = test>>1;

tInt++;

}

levels = tInt;

//adjust minDim so top level includes full image

if (levels > maxLevels) levels = maxLevels;

count = new long[levels];

roiXDim=xDim\*(1<<(levels-1));

roiYDim=yDim\*(1<<(levels-1));

xOffset = new int[levels];

yOffset = new int[levels];

xOffset[0]=yOffset[0] = 0;

xOffset[1]=xDim;

yOffset[1]=yDim;

for (int ii=2;ii<levels;ii++){

xOffset[ii] = 2\*xOffset[ii-1];

yOffset[ii] = 2\*yOffset[ii-1];

}

}

public boxDef(int minDim, int xSide, int ySide, boolean autoAR) {

this(minDim, xSide, ySide, autoAR, 0);

}

public boxDef(int minDim, int xSide, int ySide) {

this(minDim, xSide, ySide, false, 0);

}

public boxDef(int minDim, int xSide) {

this(minDim, xSide, xSide, false, 0);

}

}

## ImageJ Macro

The following macro written for ImageJ divides the SEM images into two non-overlapping square images and implements the gray-level calculations from the “Texture MeanSD” plugin.

currImgName=getTitle;

makeRectangle(0, 180, 512, 512);

run("Duplicate...", "title=1\_"+currImgName);

run("Texture MeanSD");

selectWindow(currImgName);

makeRectangle(512, 180, 512, 512);

run("Duplicate...", "title=2\_"+currImgName);

run("Texture MeanSD");