More experimenting with color based segmentation

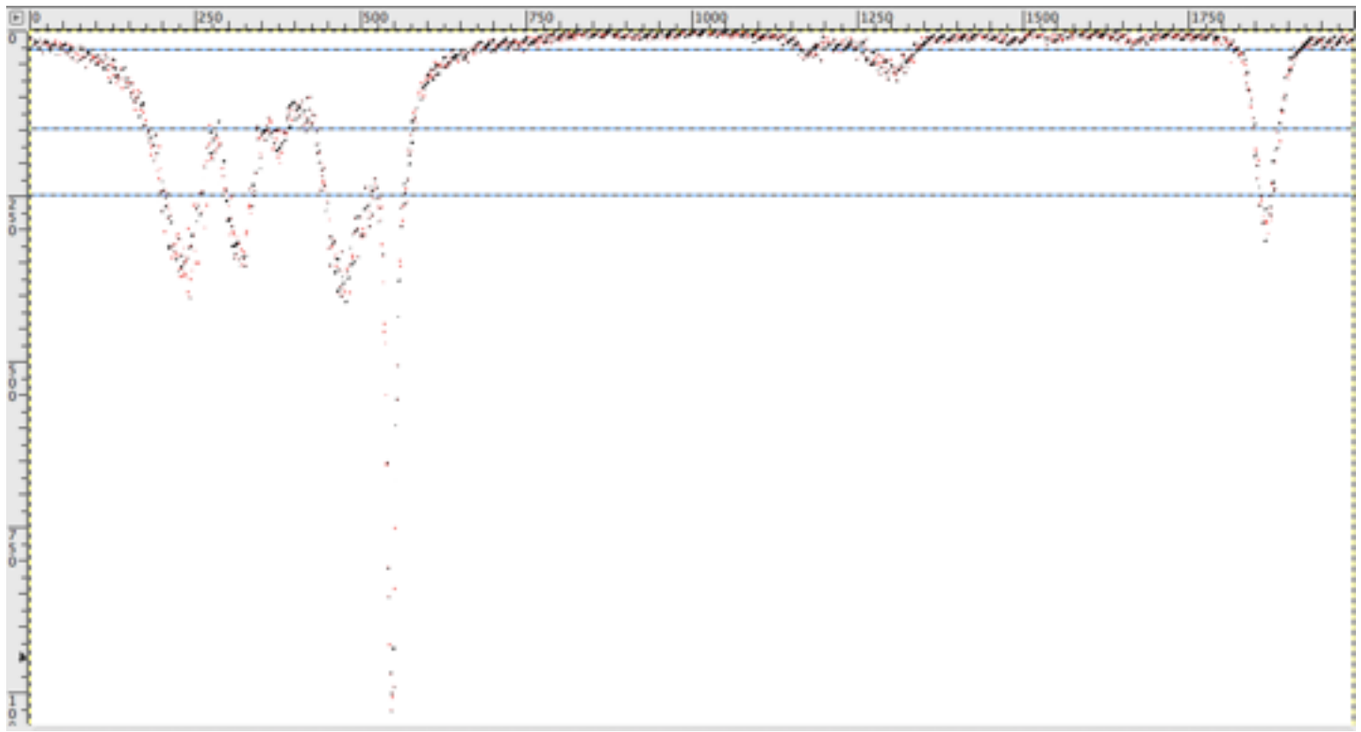
Wrote a clustering algorithm that does not need number of partitions specified (see project two-point-correlation). Then converted the colors to CIE XY Lab color space, and then polar angle of the color point (excluded black colors which are not defined in that space and white). Then made maps of CIEXY polar theta and the number of points with that polar theta (=frequency). The single map provides better segmentation when partitioned by frequency so
made 4 partitions, then found clusters within those 4 maps separately.

Each map got rescaled so the limits of its data fill a certain boundary. (The clustering algorithm uses dynamic programming so a smaller range of data has a faster runtime).

These are the results of partitions fixed to 0.03, 0.15, 0.25, 1.0 X max frequency and varying range sizes (which is similar to binning in that reference frame). The first frequency range, 0 to 0.03 X max frequency are the least frequent colors so those are left out of clustering and become a single set of pixels, that is a cluster).

Each resulting cluster is assigned a rotating color out of 28 possible colors and that's what is presented in the segmented images. The colors don't have value except delineating different clusters.

The original picture is below and the CIEXY Polar Theta vs frequency scaled to 2000 x 1000 before partitioning



Notes:

haven't yet implemented a correction to shift the theta range so that an existing gap is placed at 360, resulting in the original 360 and 0 being adjacent

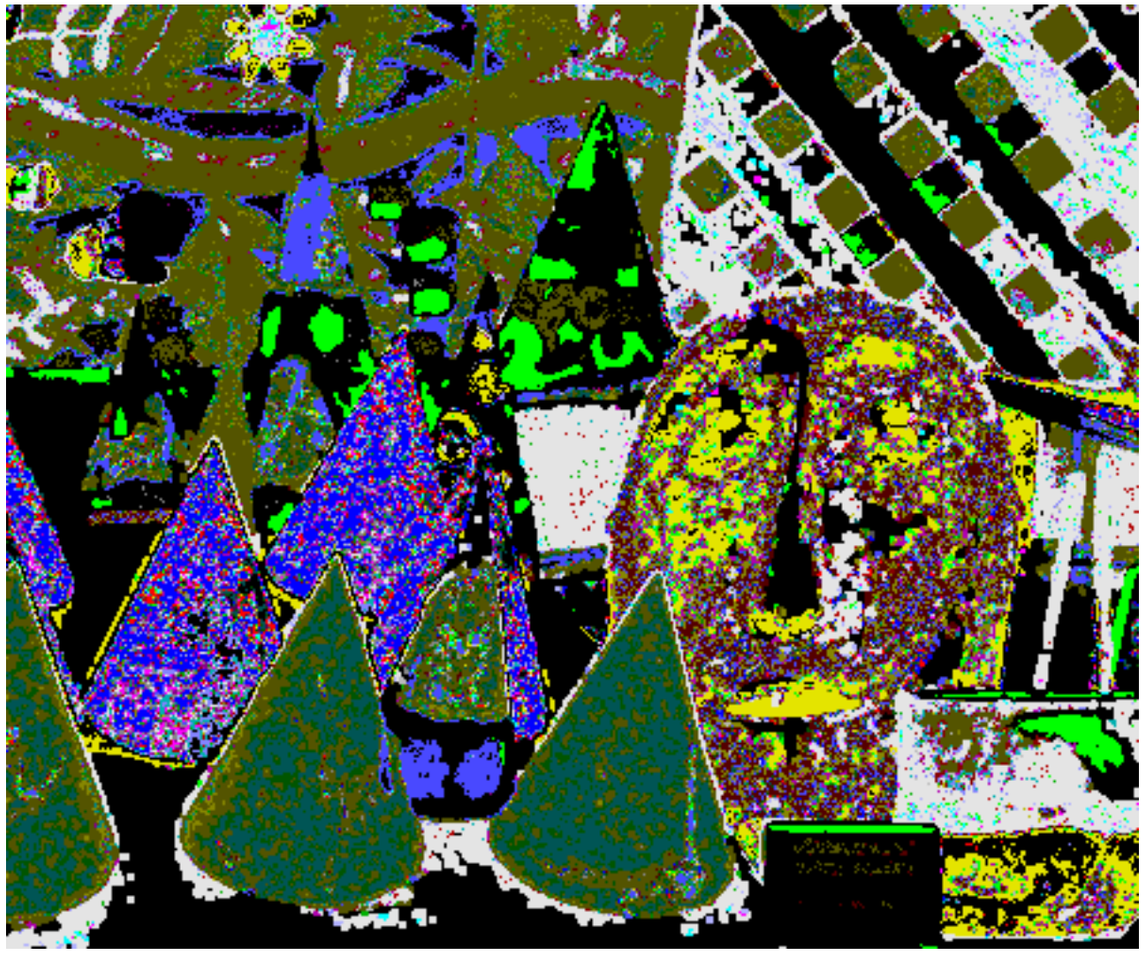
can see that one could apply merging rules easily to the 3000 x 1000 for embedded small clusters to result in large cluster.

The two color circles (next to last page) shows that the pixels left out of clustering (seen in black) should be merged into the closest adjacent color pixel's cluster.

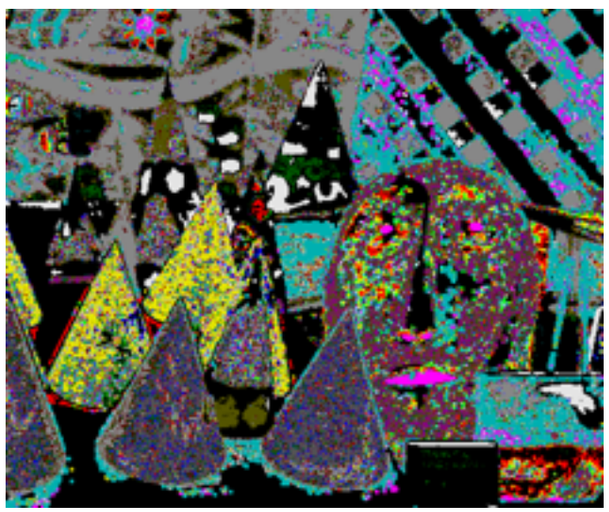
Will make a different algorithm that takes the peaks of the theta frequency map and partitions the pixels in between those peaks, theta-wise, into the peak clusters. Can see that the results should look similar to the k=8, 16 segmentation algorithm summarized in doc/colorSegmentation.pdf, but this should be faster than using kmeans++ and more stable (my kmeans++ uses random numbers).
(***Implemented now, see last page***)

theta range scaled to 3000 and freq range scaled to 3000

theta range scaled to 3000 and freq range scaled to 1000



theta range scaled to 2000 and freq range scaled to 1000



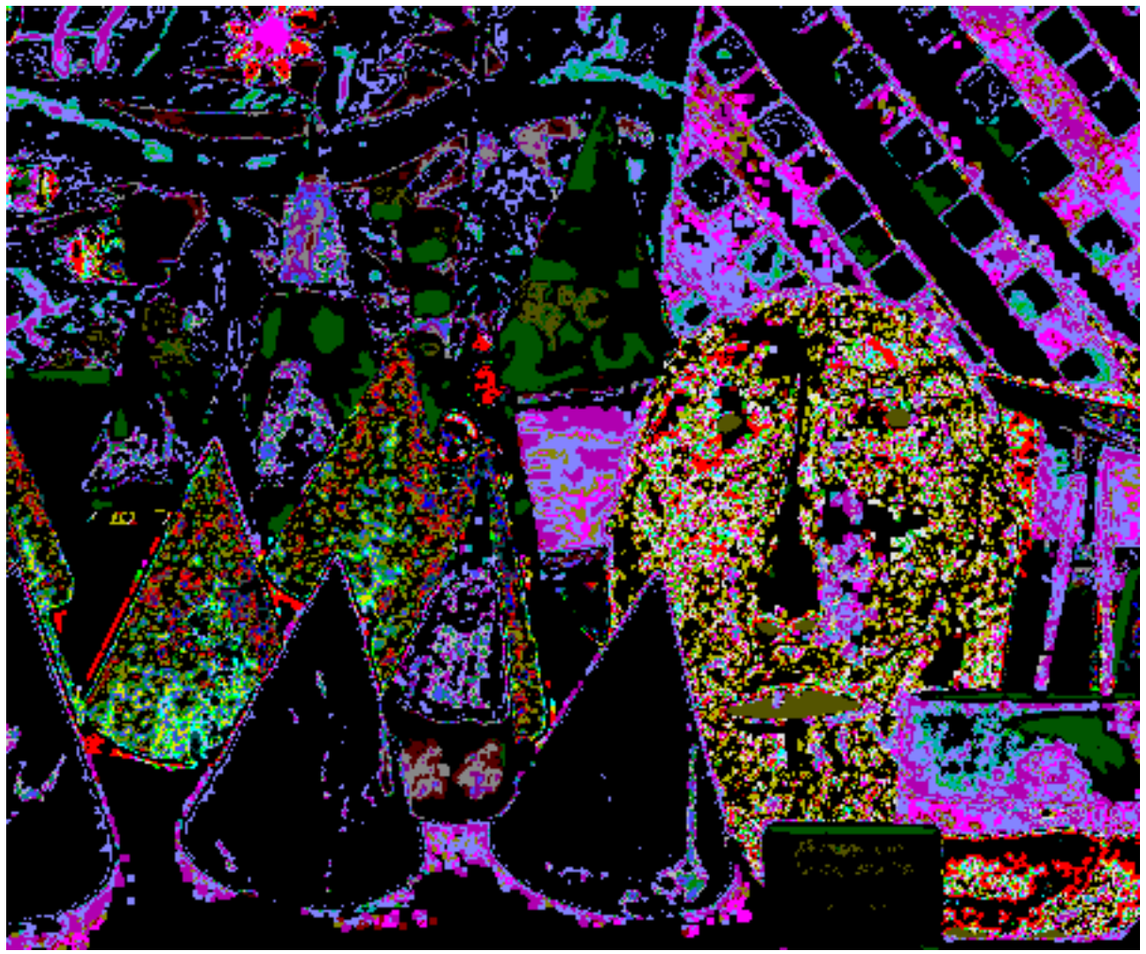
theta range scaled to 2000 and freq range scaled to 2000

theta range scaled to 1000 and freq range scaled to 1000

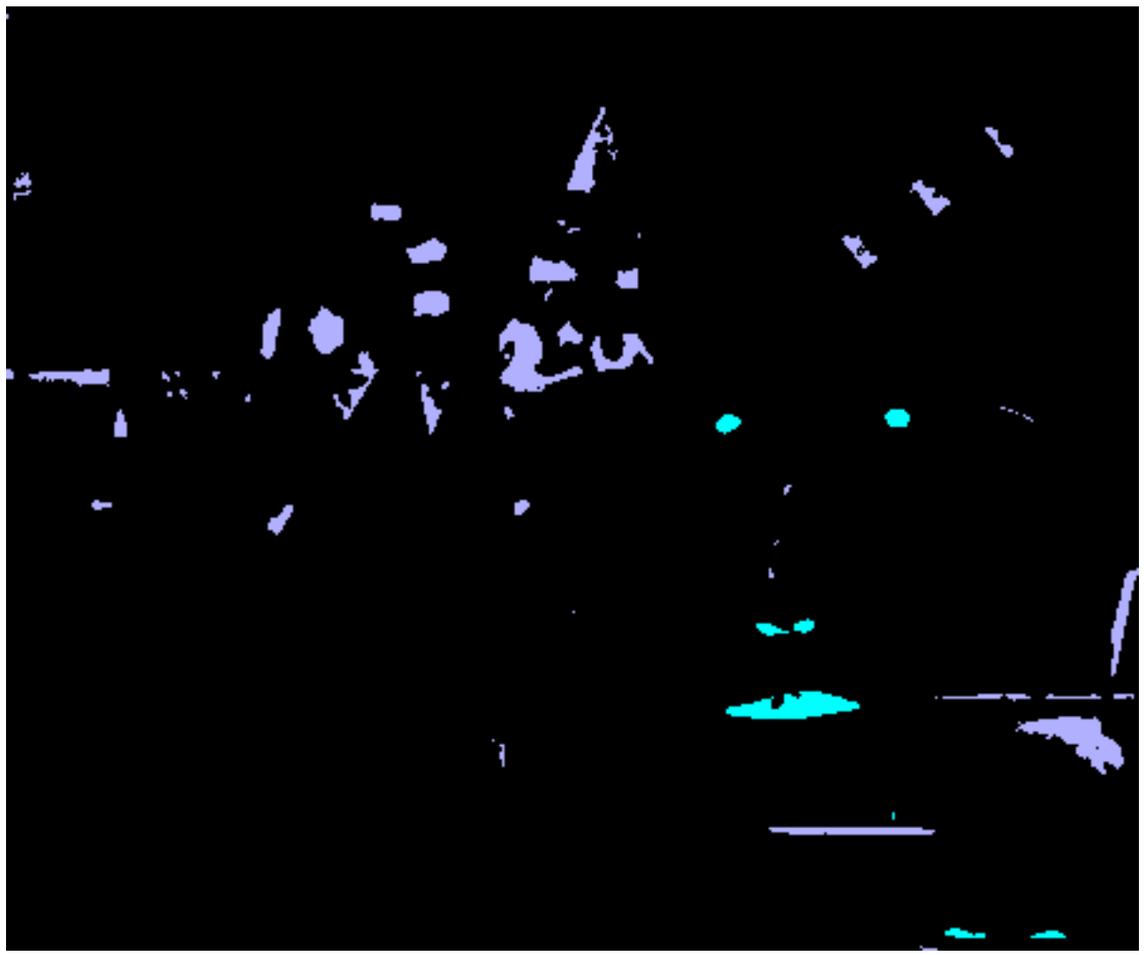
There is a blur of sigma=1 applied before. Note that the edges which are due to the blur and aliasing tend to be large frequency color groups.

Note that when a blur isn't applied, the results look more granulated as expected.

theta range scaled to 500 and freq range scaled to 500



theta range scaled to 200 and freq range scaled to 200



intensity and color ranges have collapsed, leaving only black and white pixel groups.

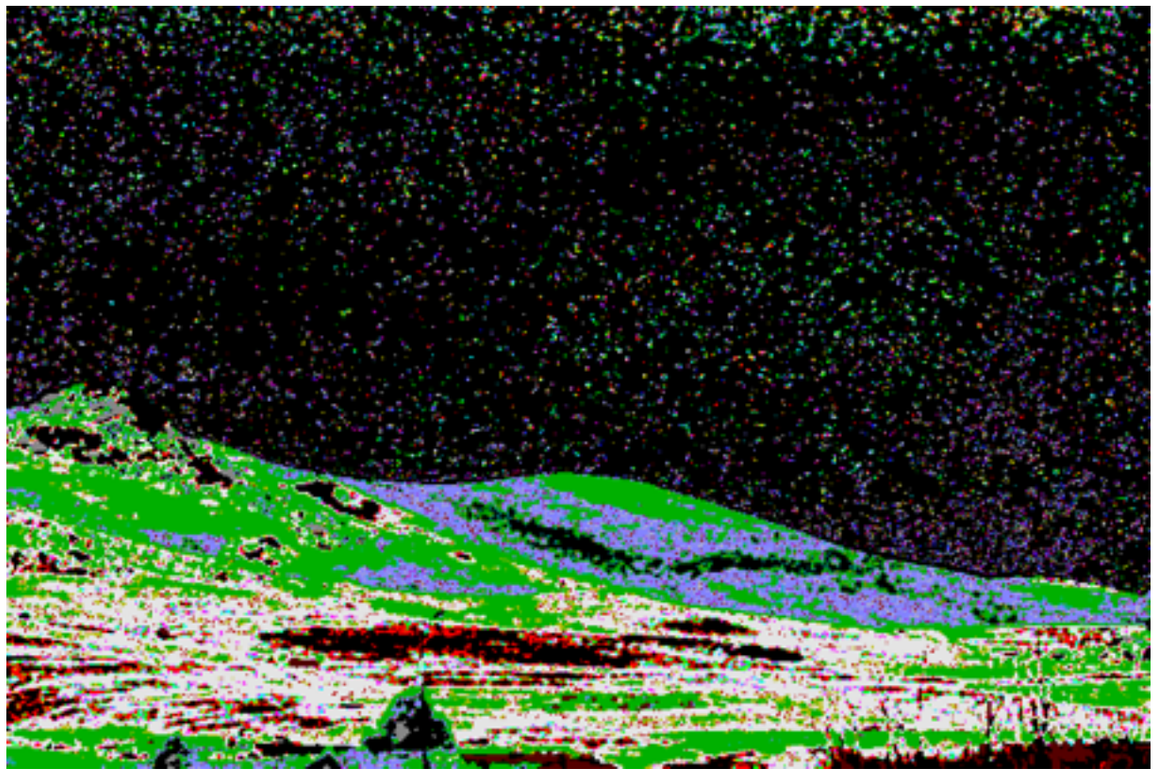
Same is presented for the cloudy san jose image.

These are the results of partitions fixed to 0.03, 0.15, 0.25, 1.0 X max frequency and varying range sizes (which is similar to binning in that reference frame). The first frequency range, 0 to 0.03 X max frequency are the least frequent colors so those are left out of clustering and become a single set of pixels, that is a cluster).

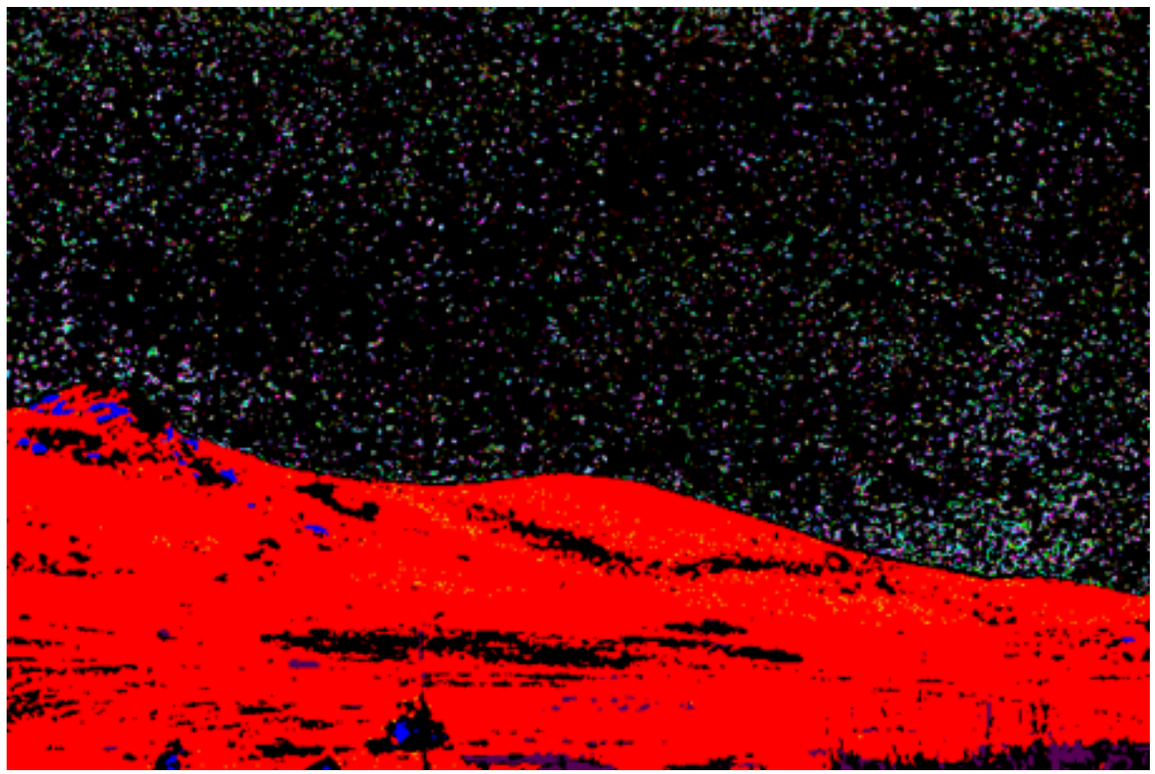
Each resulting cluster is assigned a rotating color out of 28 possible colors and that's what is presented in the segmented images. The colors don't have value except delineating different clusters.

The original picture is below and the CIEXY Polar Theta vs frequency scaled to 2000 x 1000 before partitioning

theta range scaled to 3000 and freq range scaled to 3000

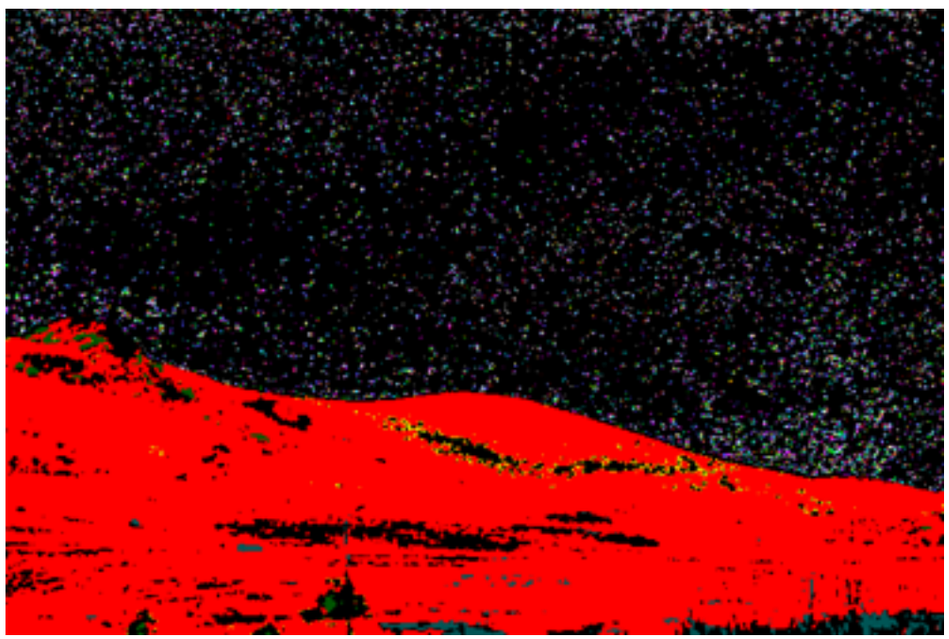


theta range scaled to 3000 and freq range scaled to 1000

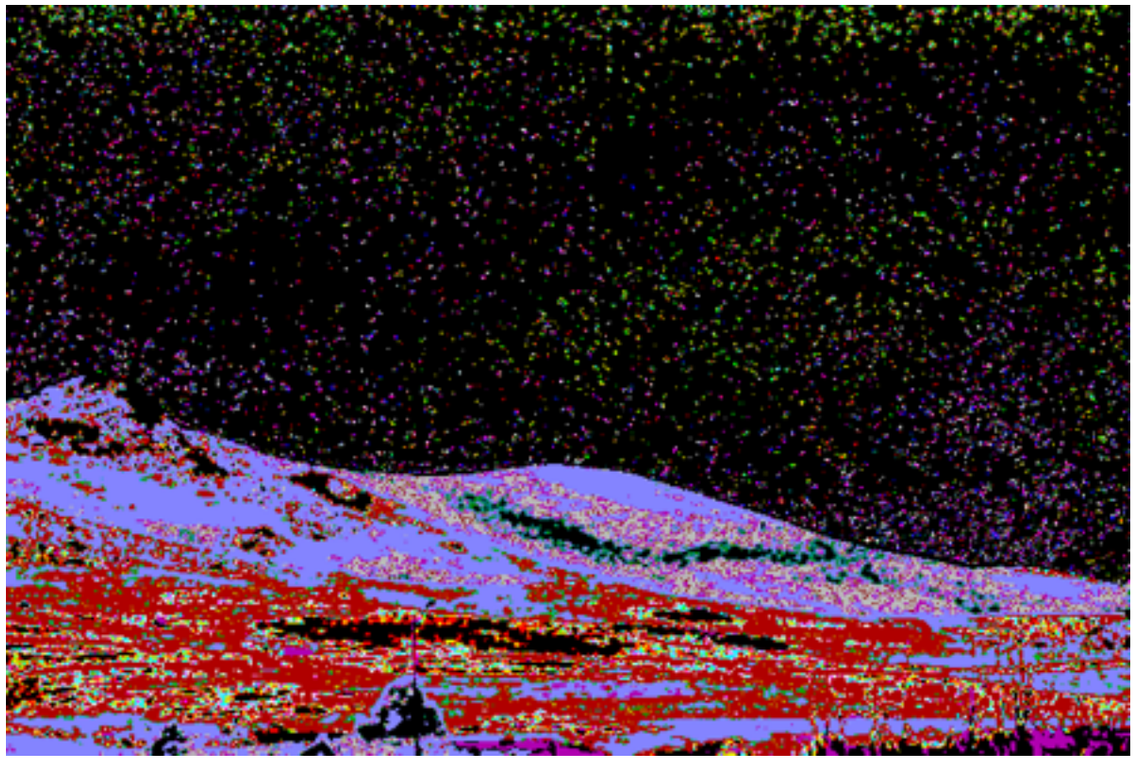


theta range scaled to 3000 and freq range scaled to 500

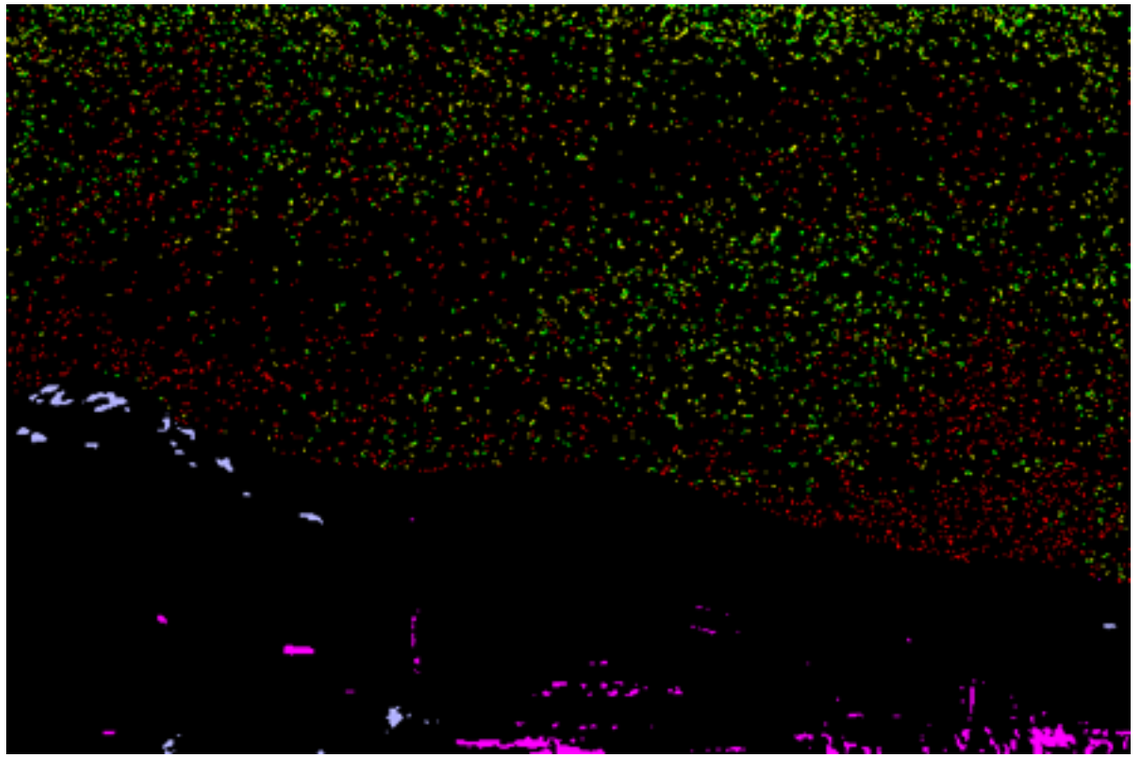
theta range scaled to 2000 and freq range scaled to 1000



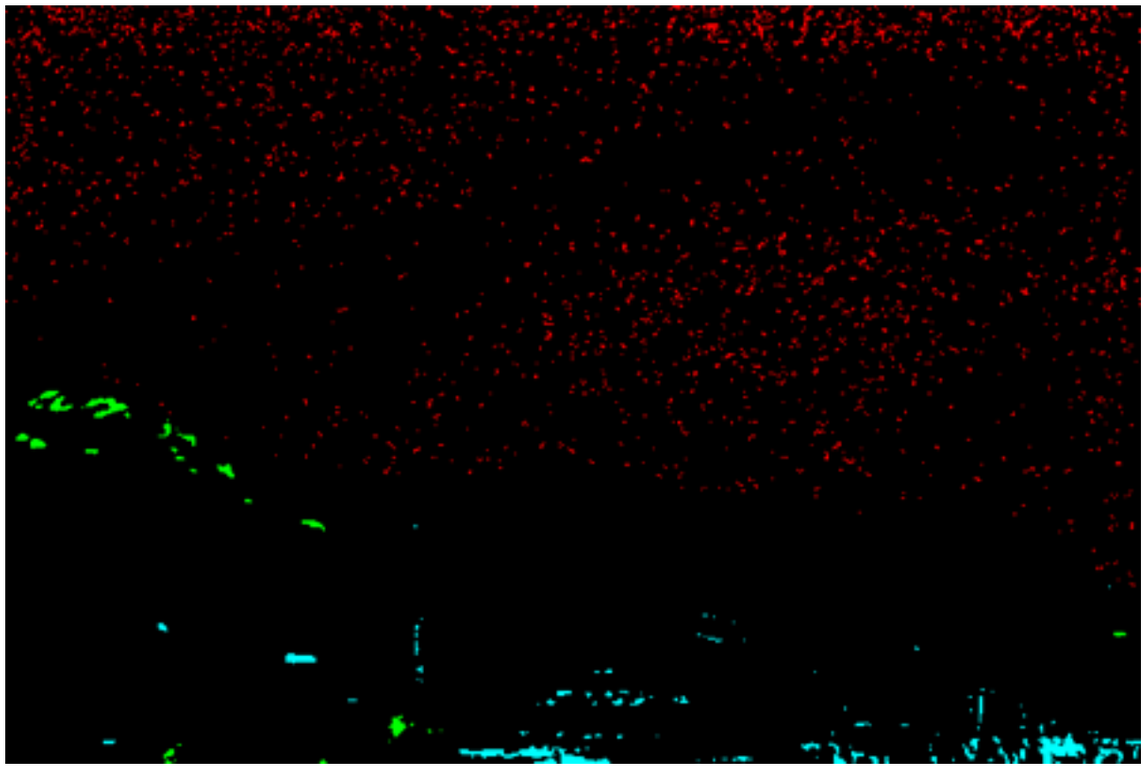
theta range scaled to 2000 and freq range scaled to 2000



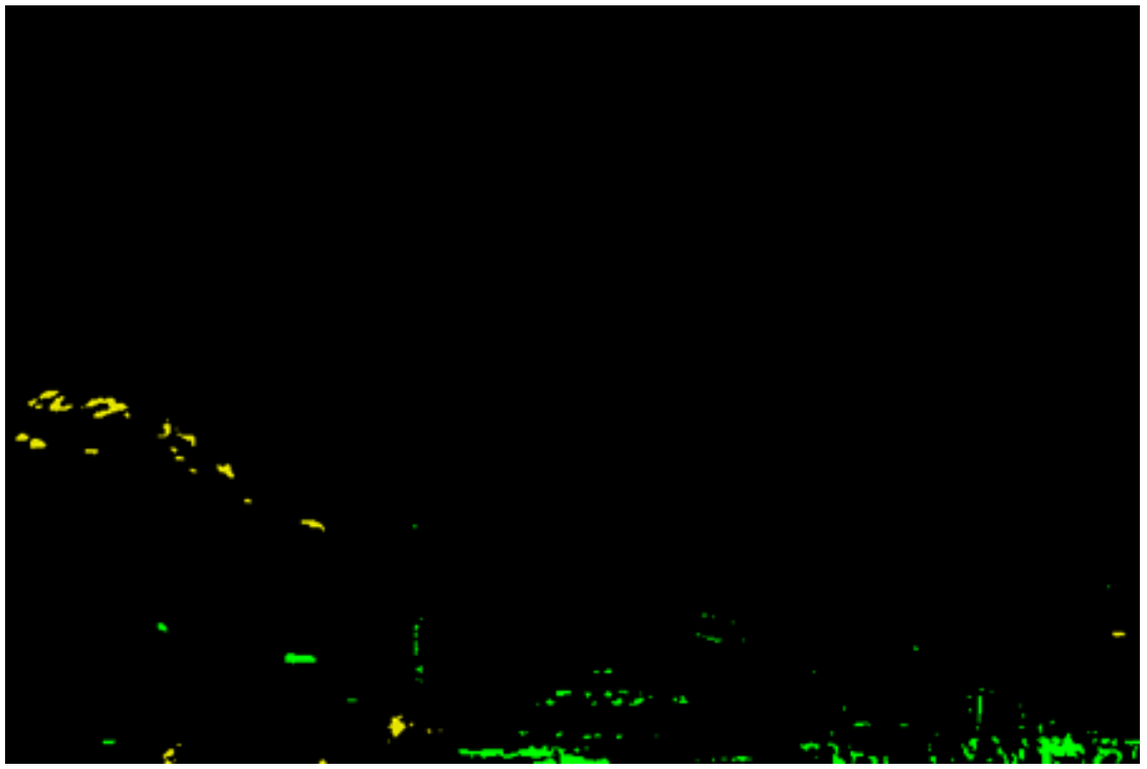
theta range scaled to 1000 and freq range scaled to 1000



theta range scaled to 500 and freq range scaled to 500

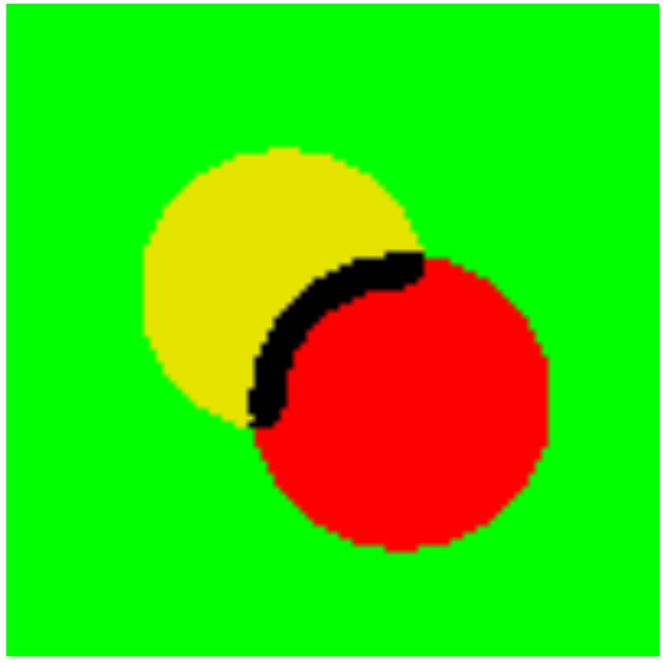


theta range scaled to 200 and freq range scaled to 200

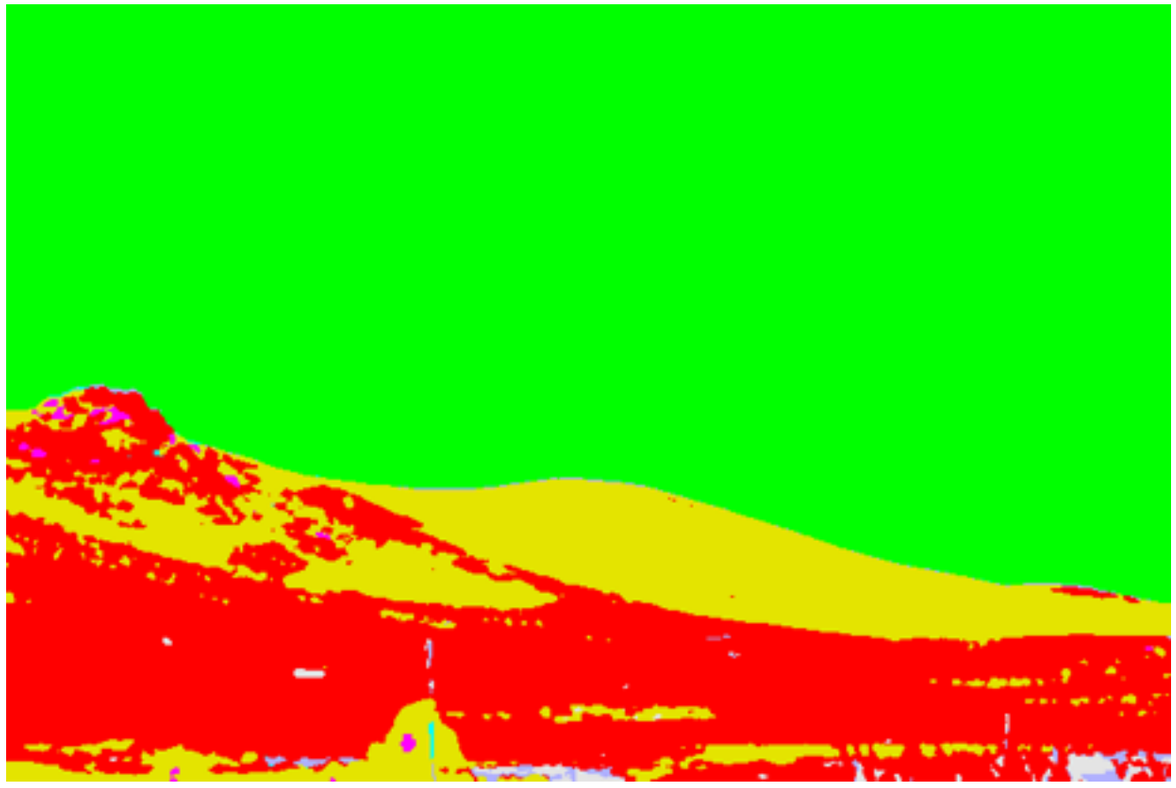


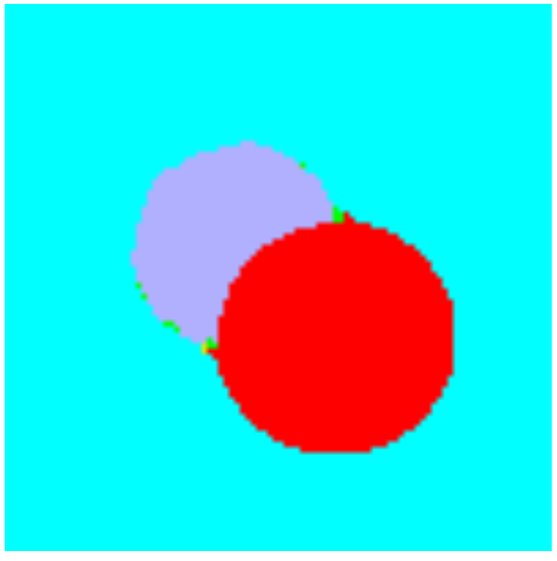
original image

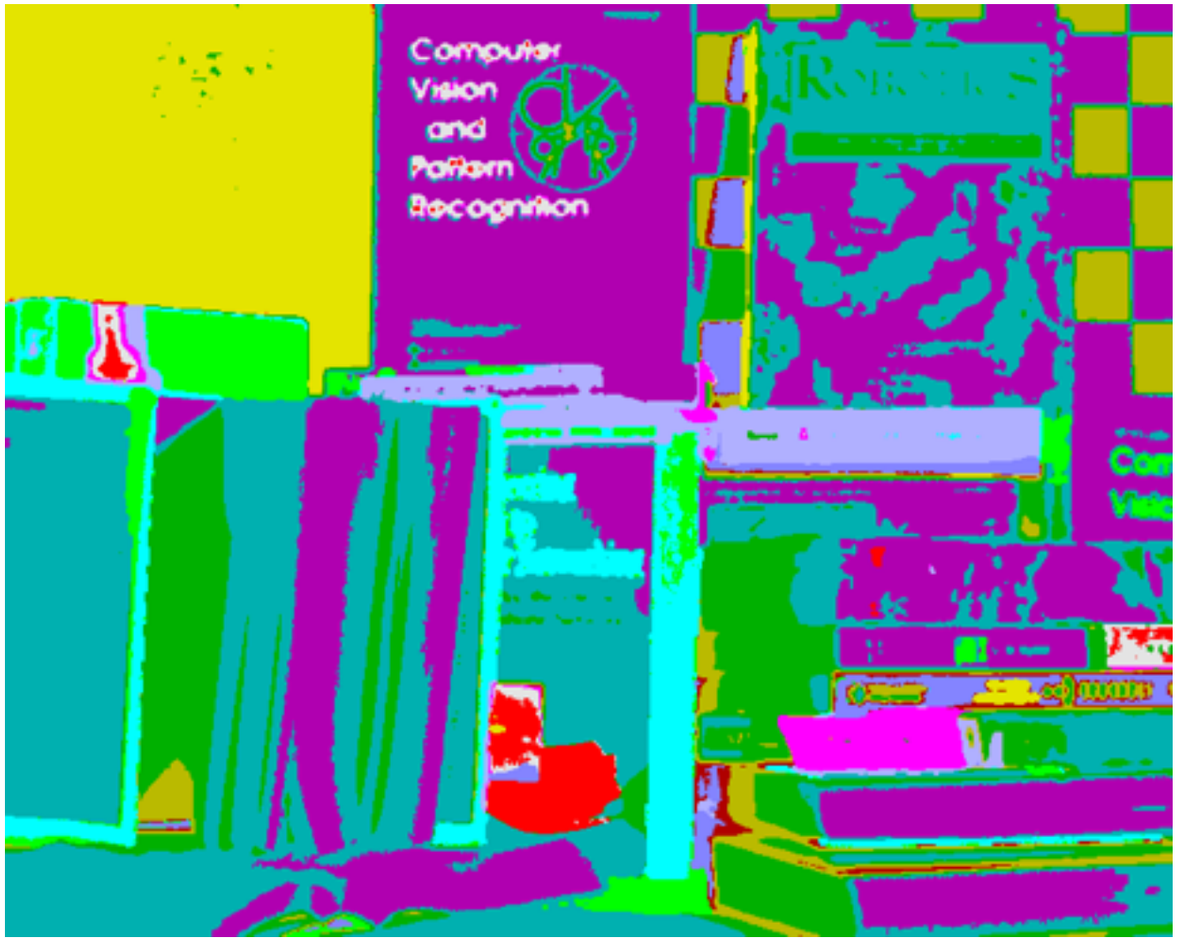
segmented: theta range scaled to 3000 and freq range scaled to 1000

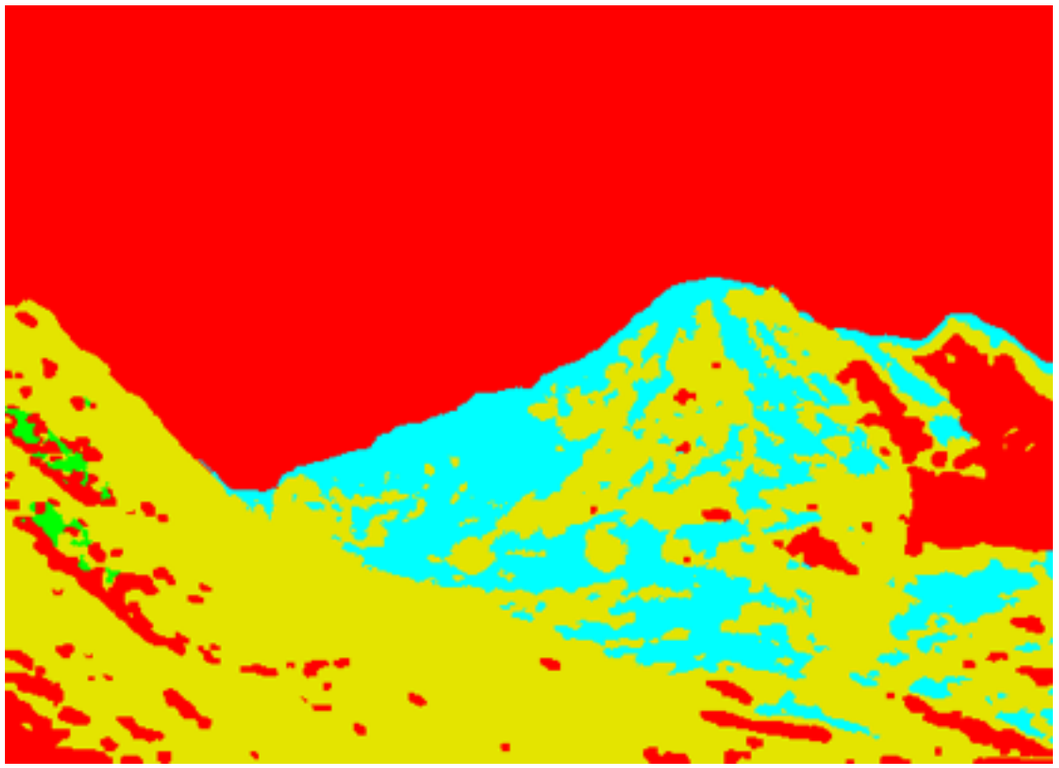


New algorithm that converts the color to polar theta CIEXY and defines the frequency as the number of points with a given theta. Uses a binning of 3 for theta currently.
Finds the peaks in frequency in theta vs frequency above a *limit frequency.* (using frequency limit of 0.03 X maxFreq) and then gathers points to the nearest peaks in theta.
The segmentation results (with false colors) are below.






Computer
Vision
and
Pattern
Recognition



all ImageSegmentation.java methods applied to a few pairs of images that are panoramic sets or stereo image sets with the goal of finding best segmentation for finding blobs to make matchable contours.

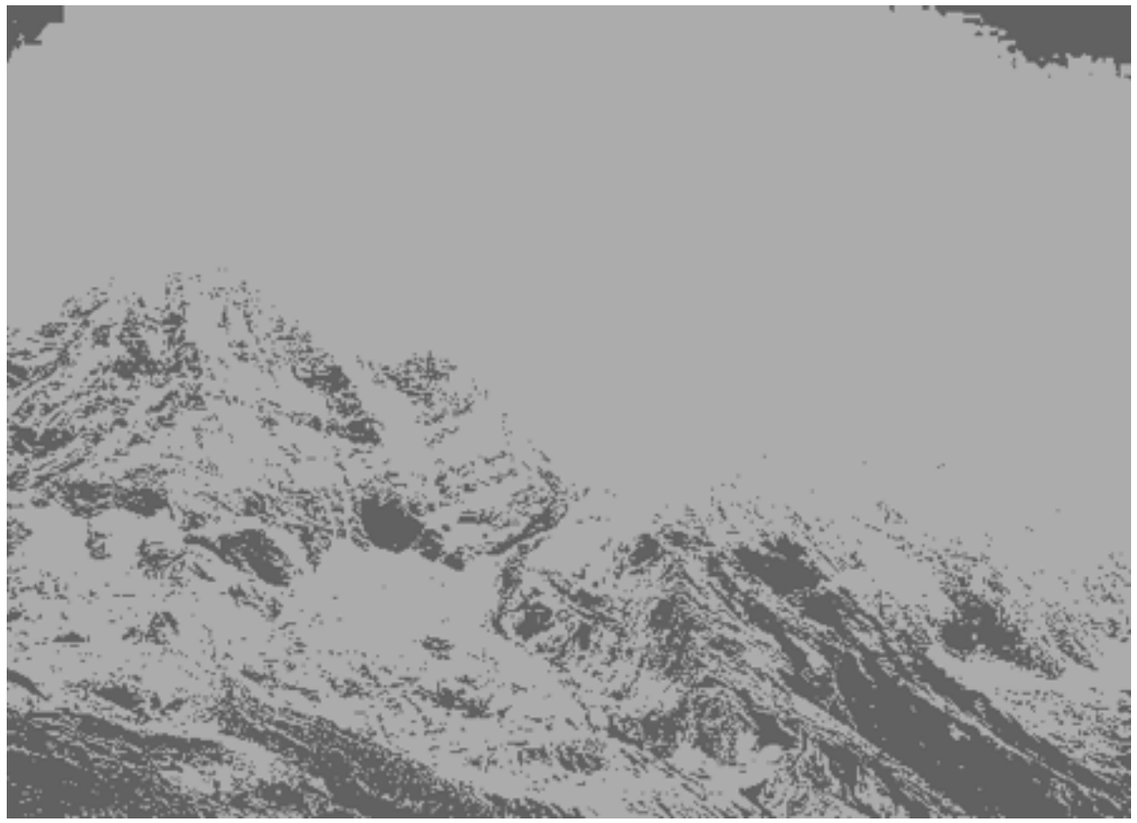
summary of next pages:

for brown & lowe 2003, best was KMPP w/ k=2
for Venturi, best was PolarCIEXYAndFrequency
for books, best was KMPP w/ k=2, 3 or 8

Venturi has lots of texture, so attempting a low resolution segmentation first is faster (the polar ciexy and frequency segmentation is O(N)… check that).

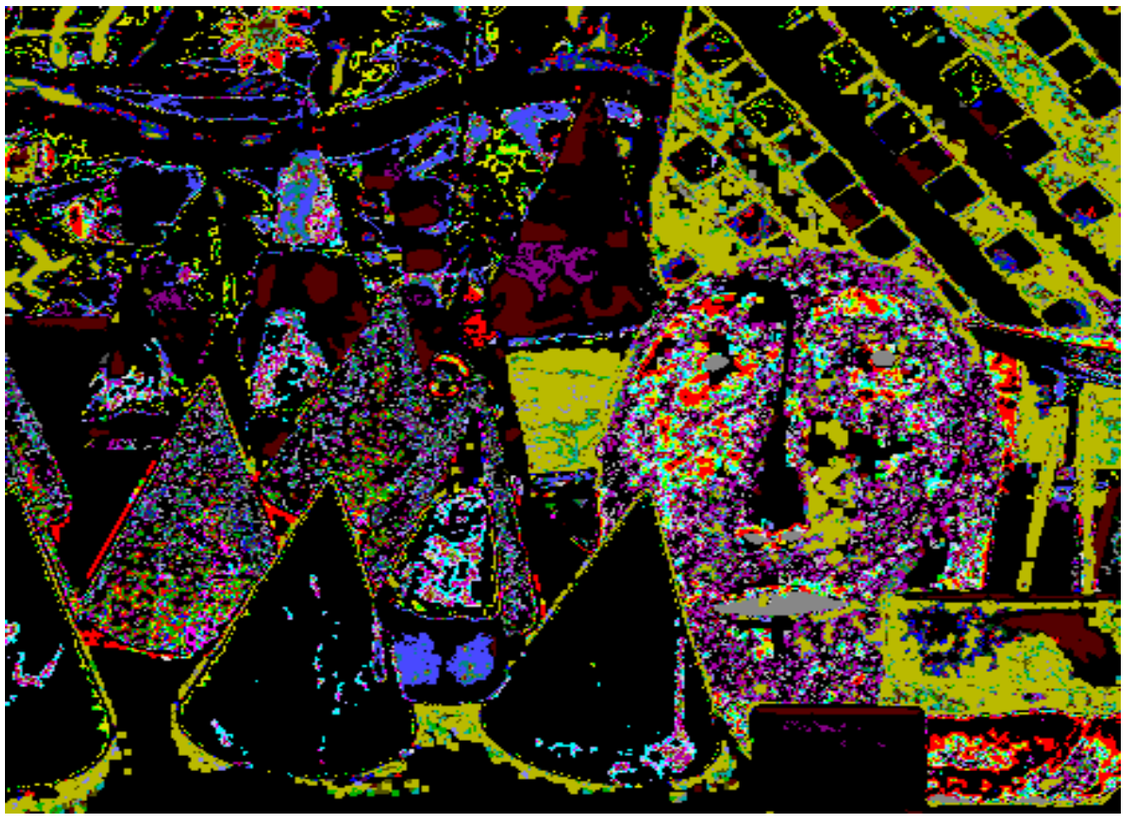
Books best matchable features w/o illumination and projection differences are the text. Text need further processing such as adaptive mean thresholding and may also need a method for the text to aggregate pixels in some letters.

```
int kBands = 2; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```

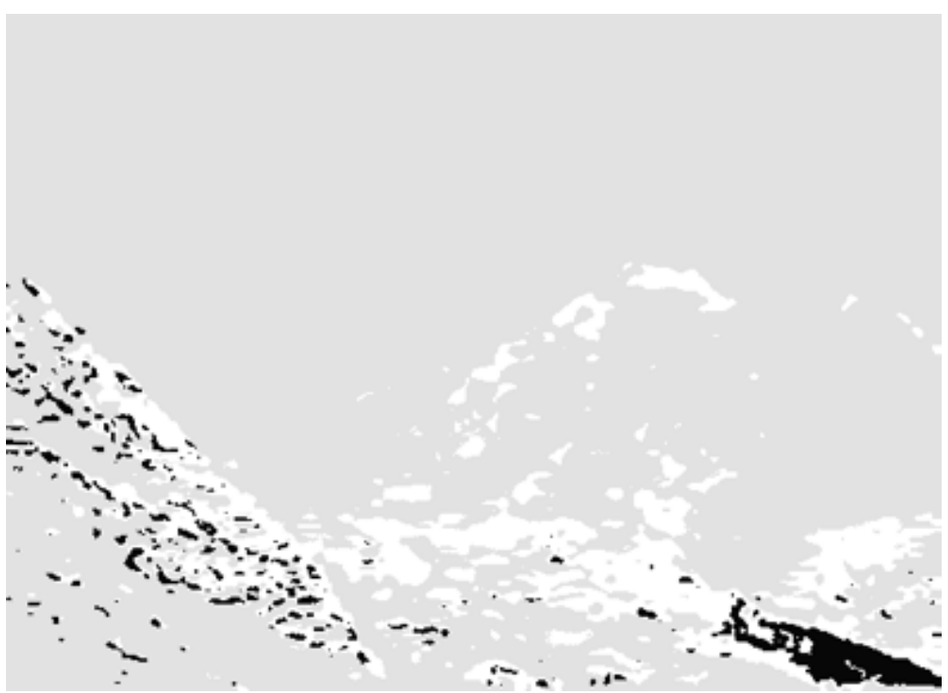


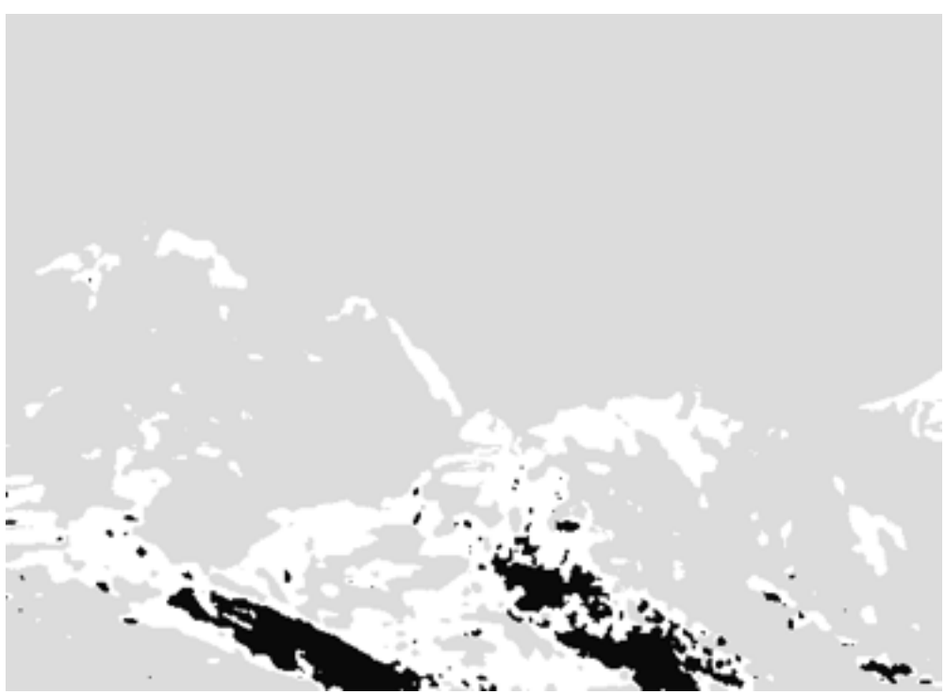
```
int kBands = 3; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```

int kBands = 8; imageSegmentation.applyUsingKMPP(gsImg1, kBands);

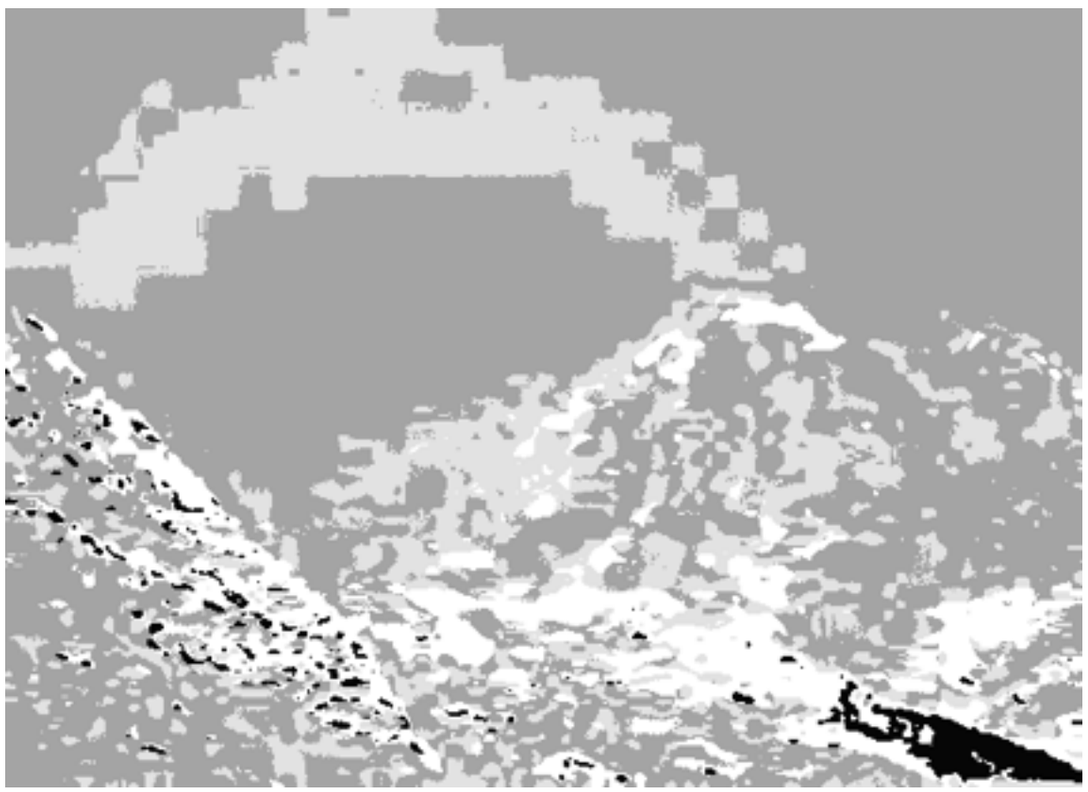


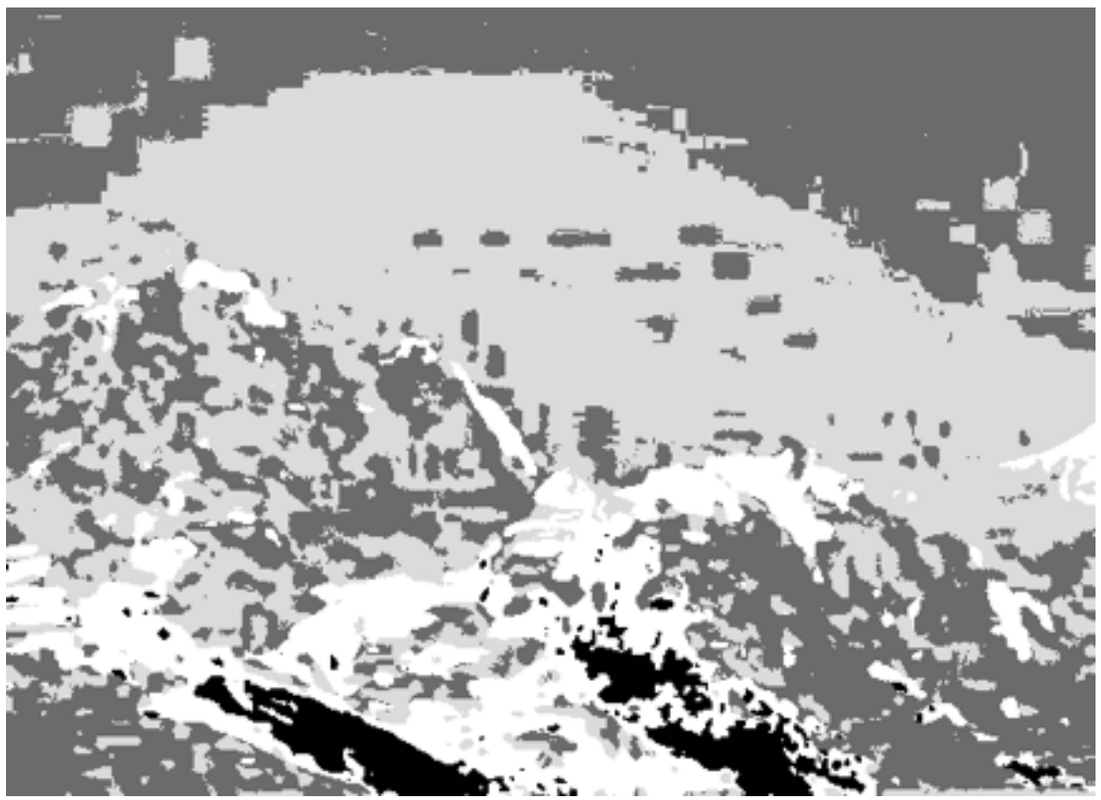
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);



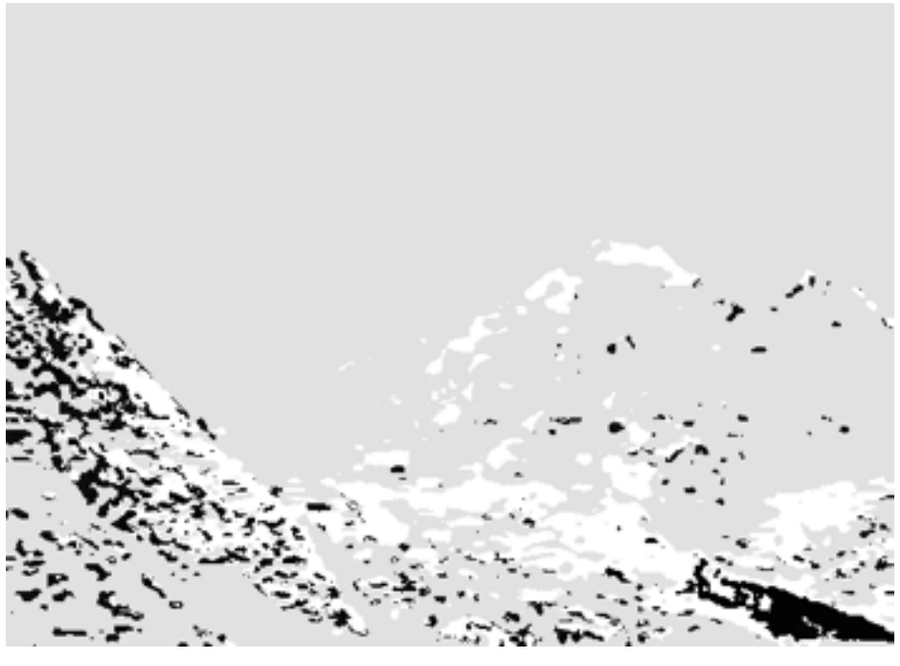


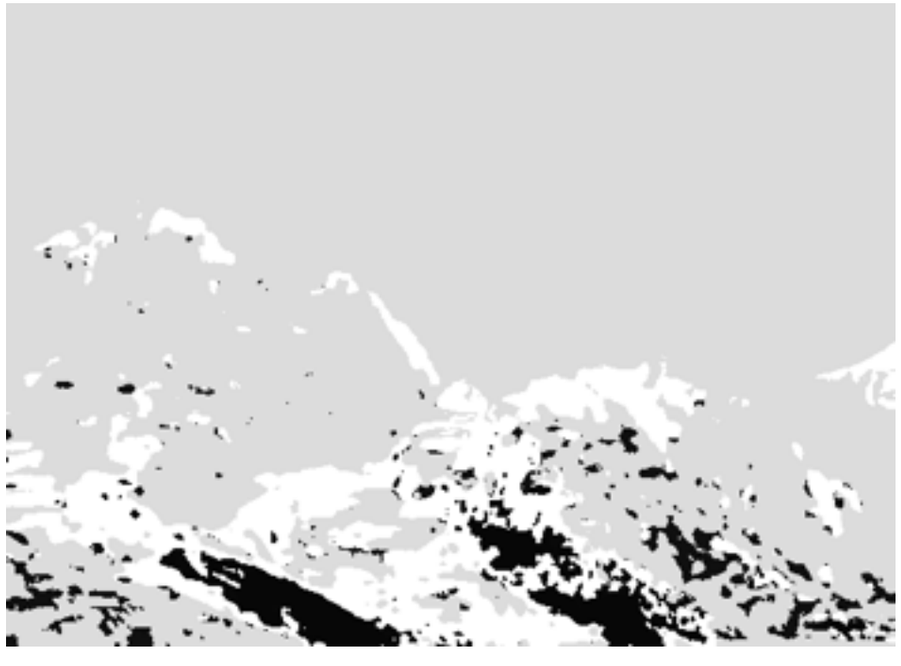
int kBands = 3; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);



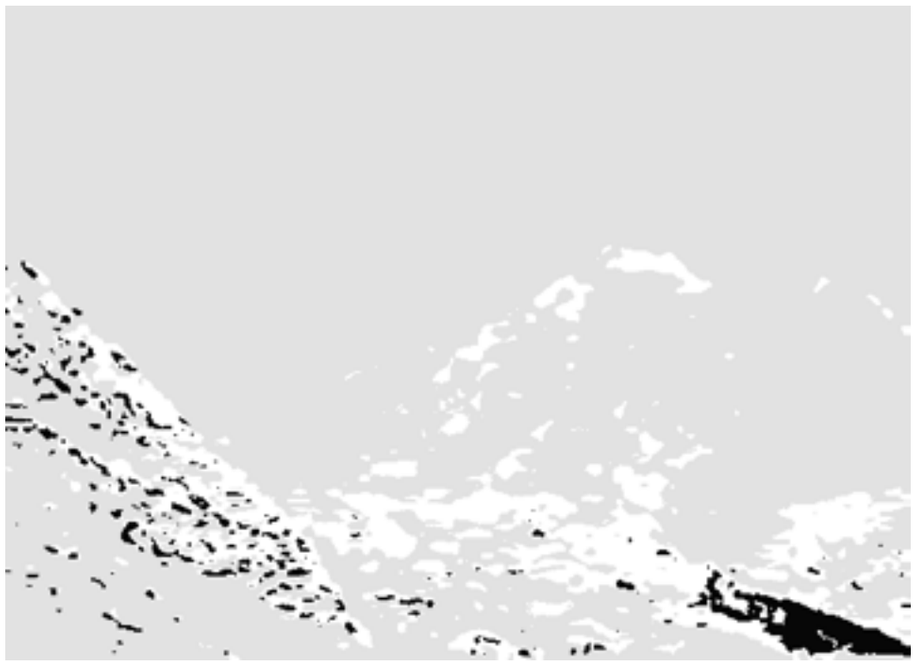


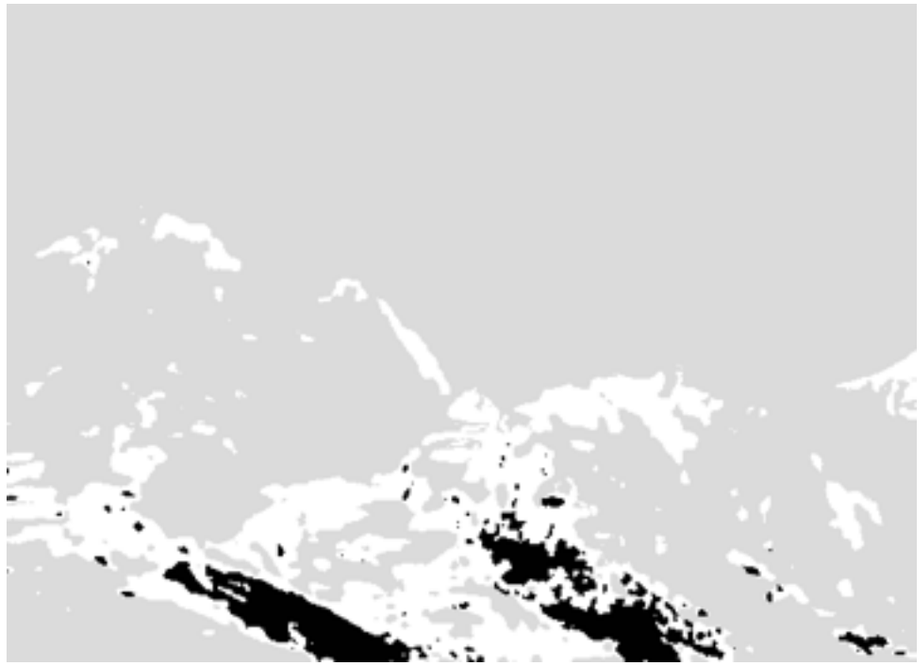
int kBands =8; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);



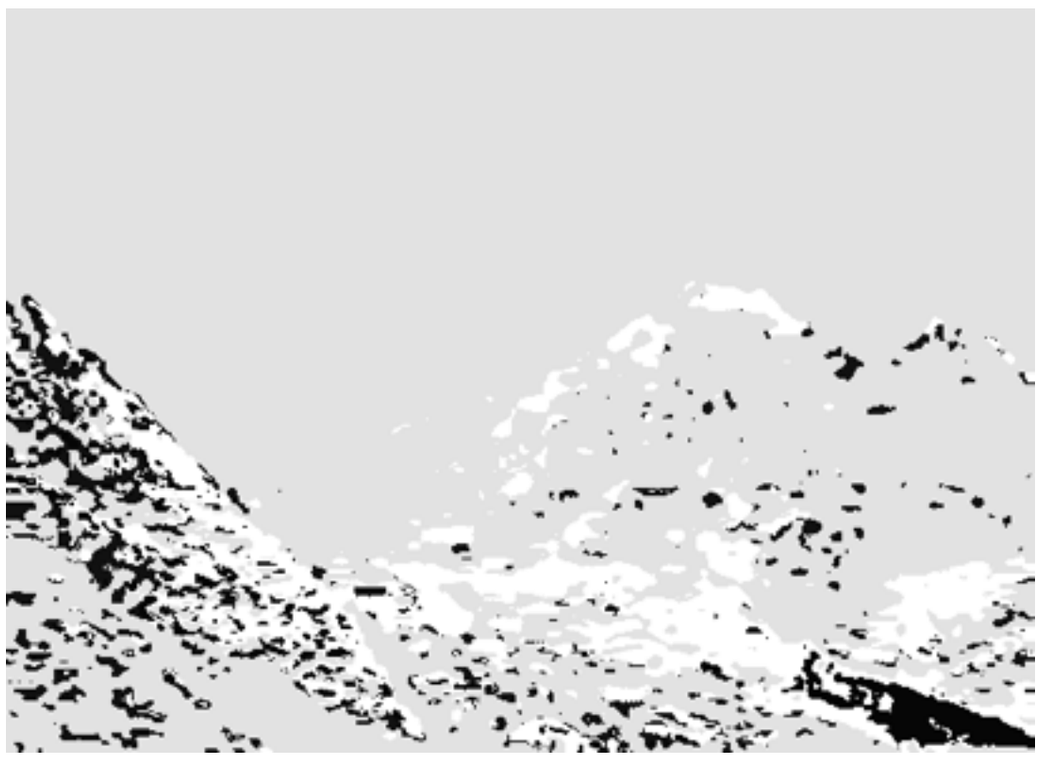


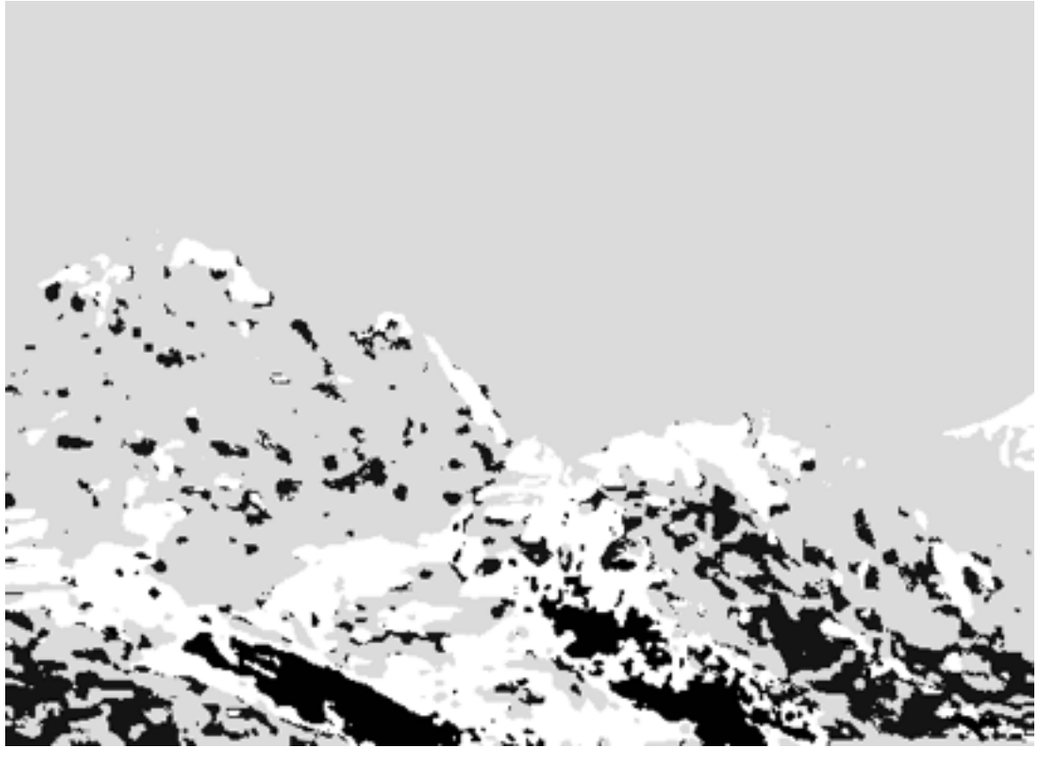
```
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);
```



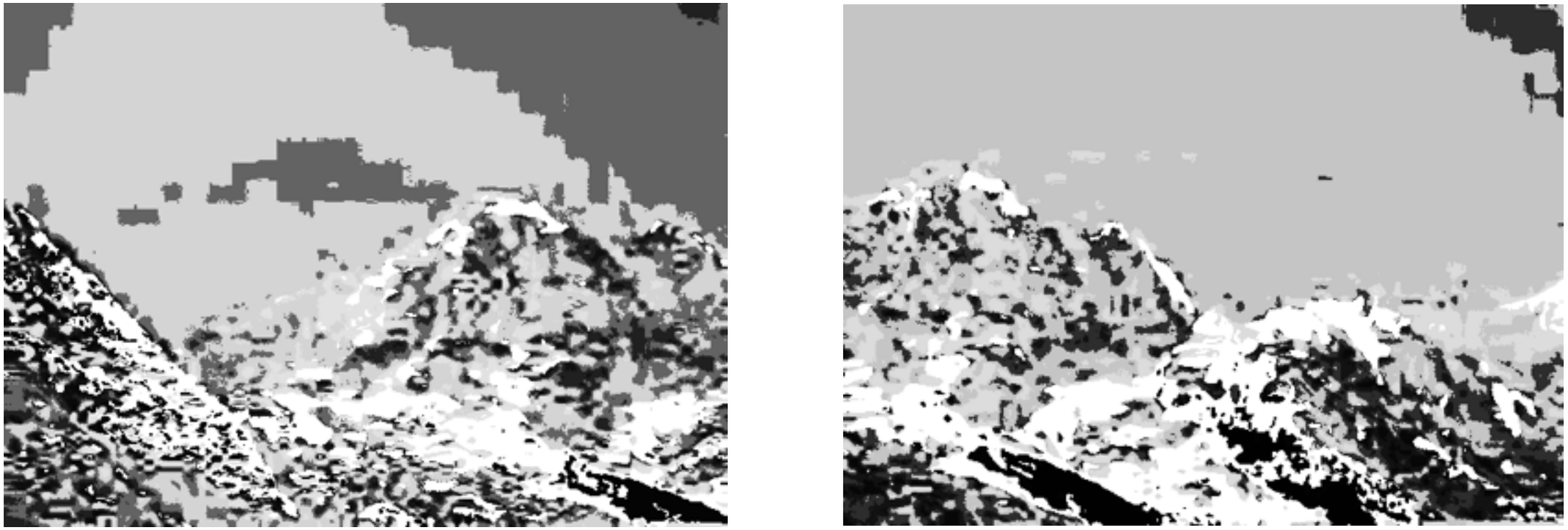


```
int kBands = 3; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);
```



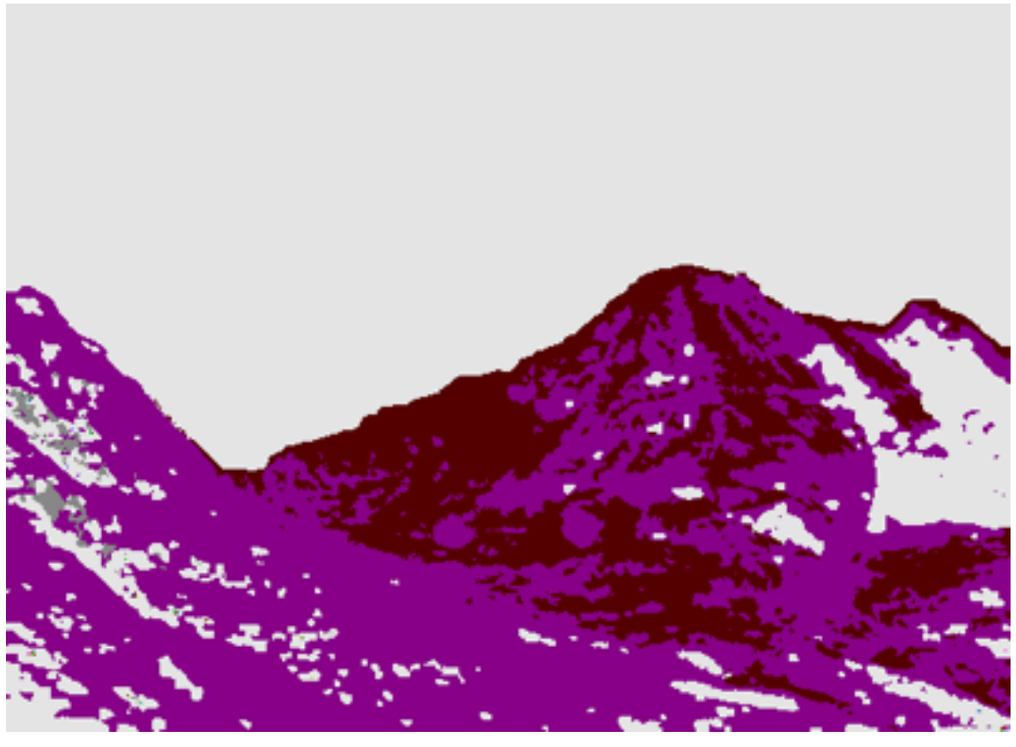


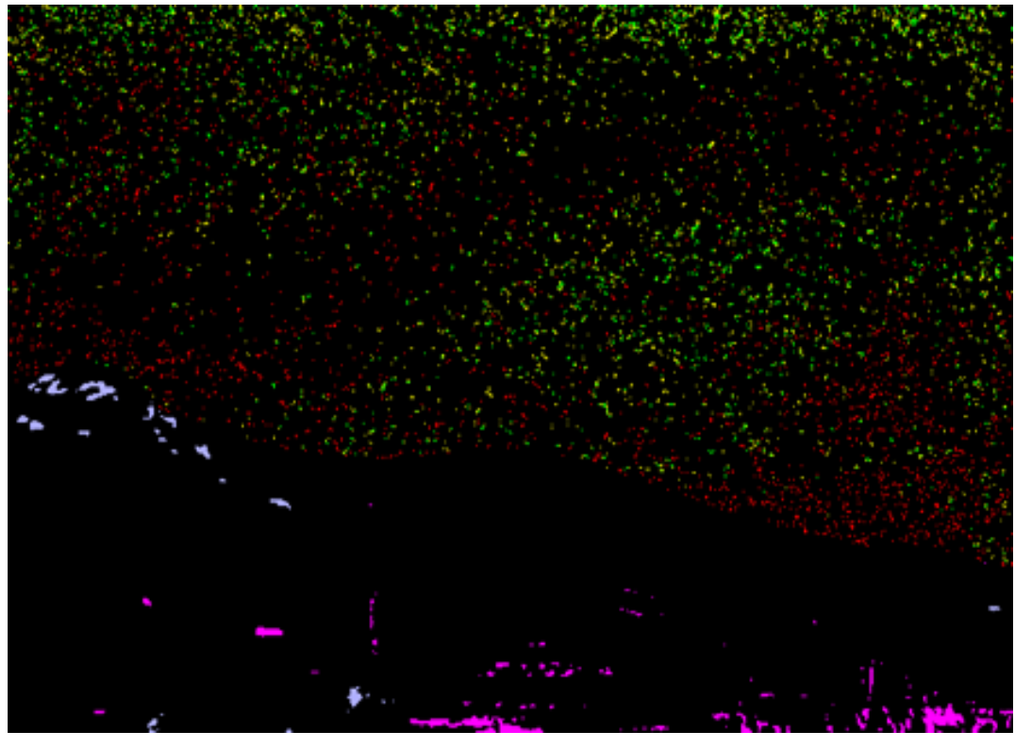
int kBands = 8; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);



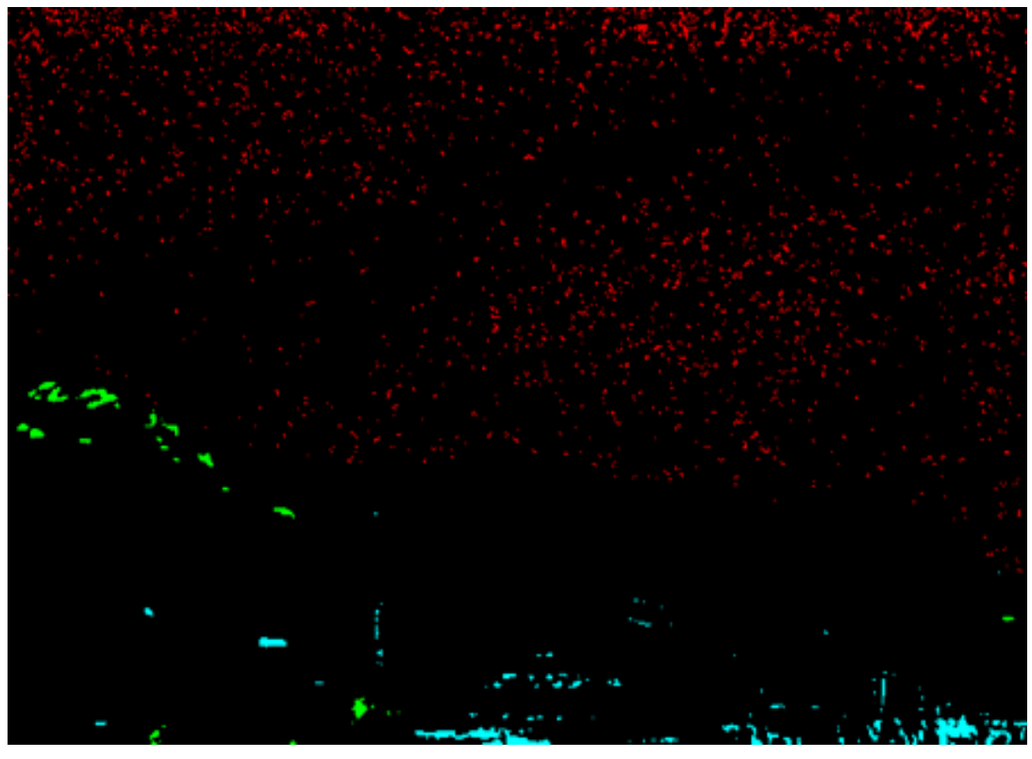
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenHistogram(gsImg1, kBands);

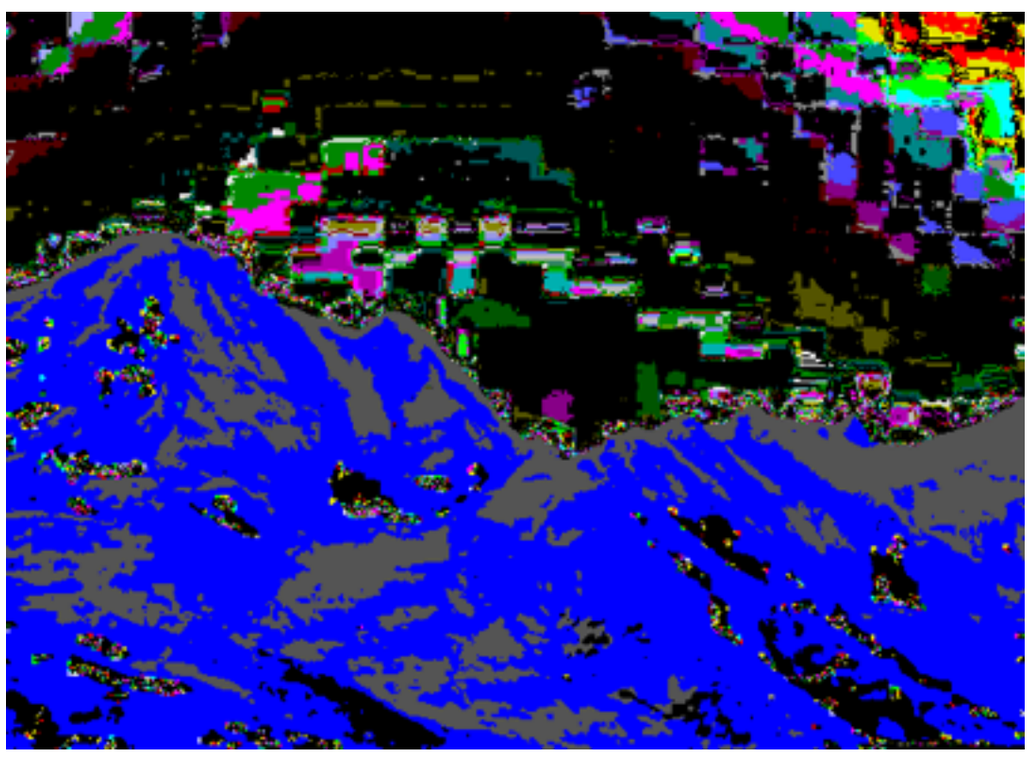
```
imageSegmentation.calculateUsingCIEXYAndClustering(gsImg1, true);
```



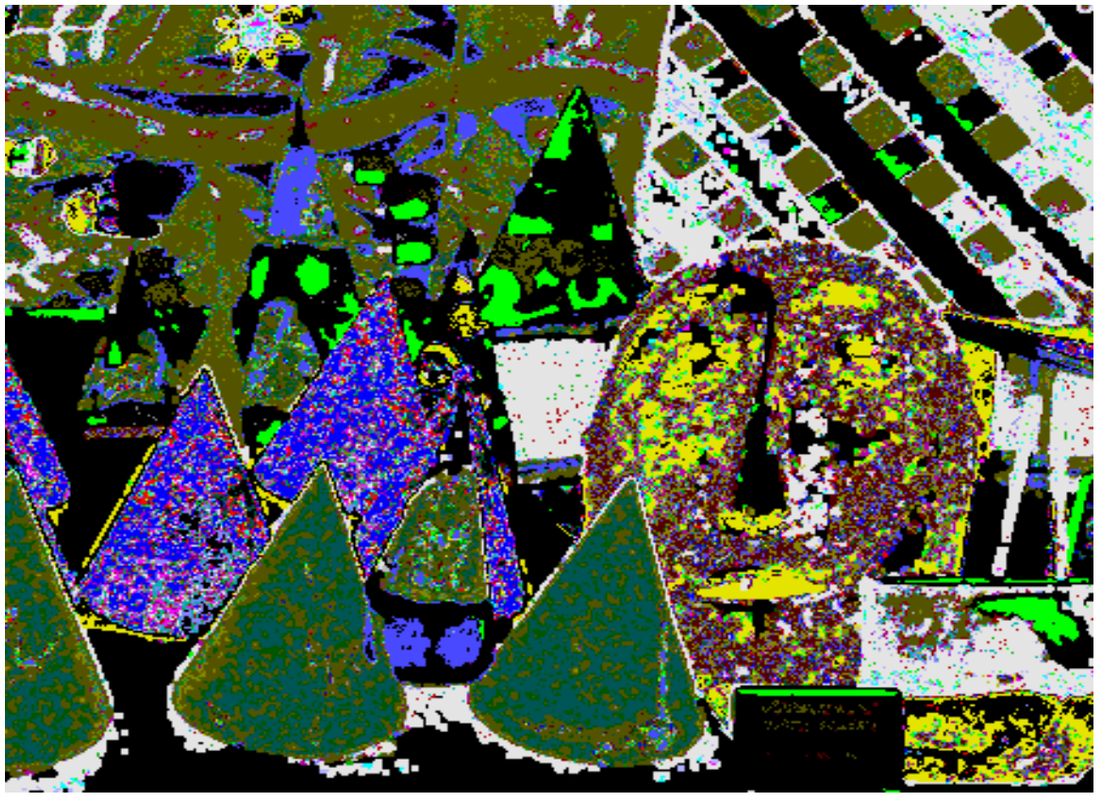


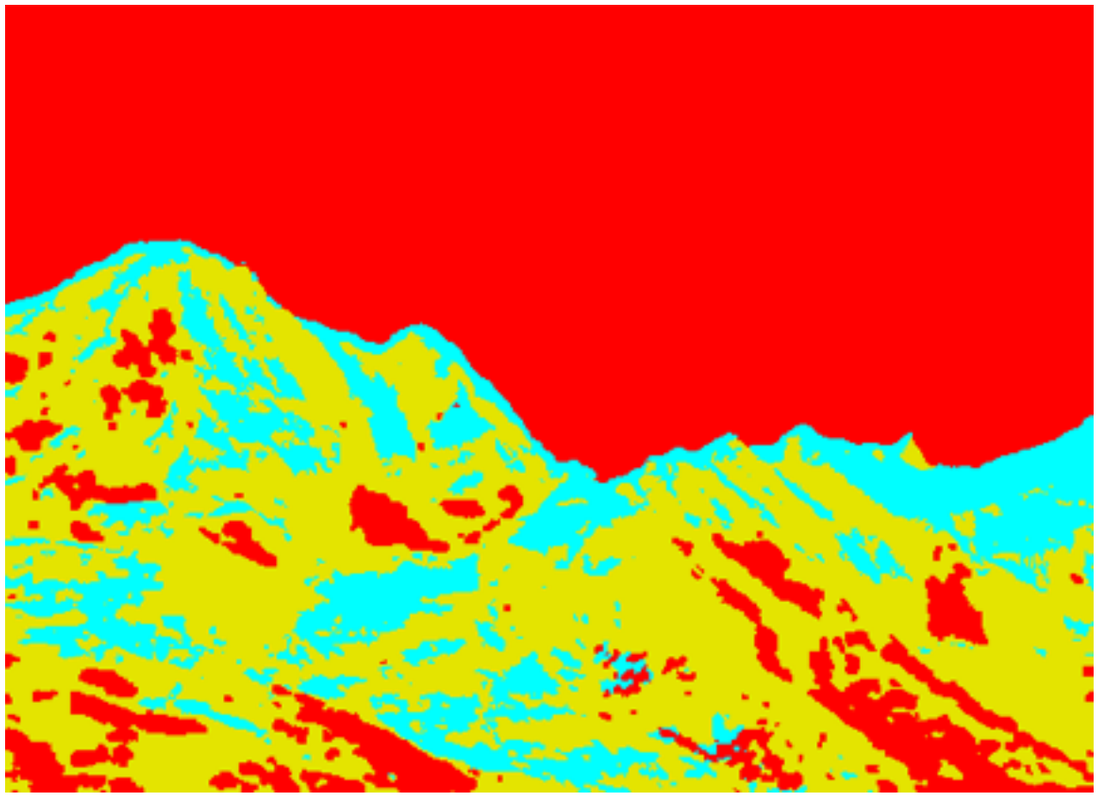
```
imageSegmentation.calculateUsingPolarCIEXYAndClustering(gsImg1, true);
```



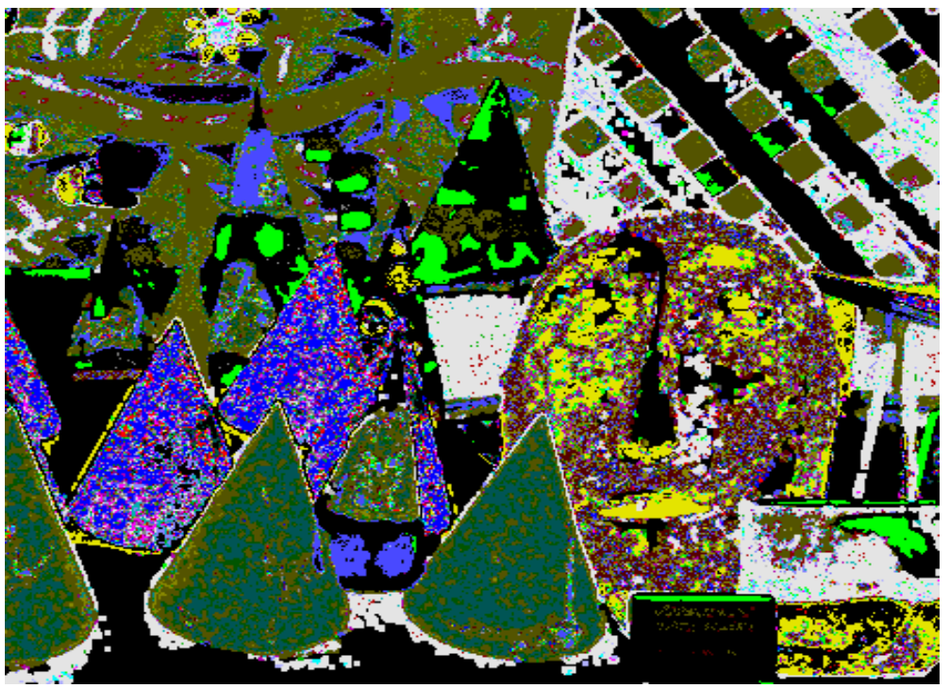


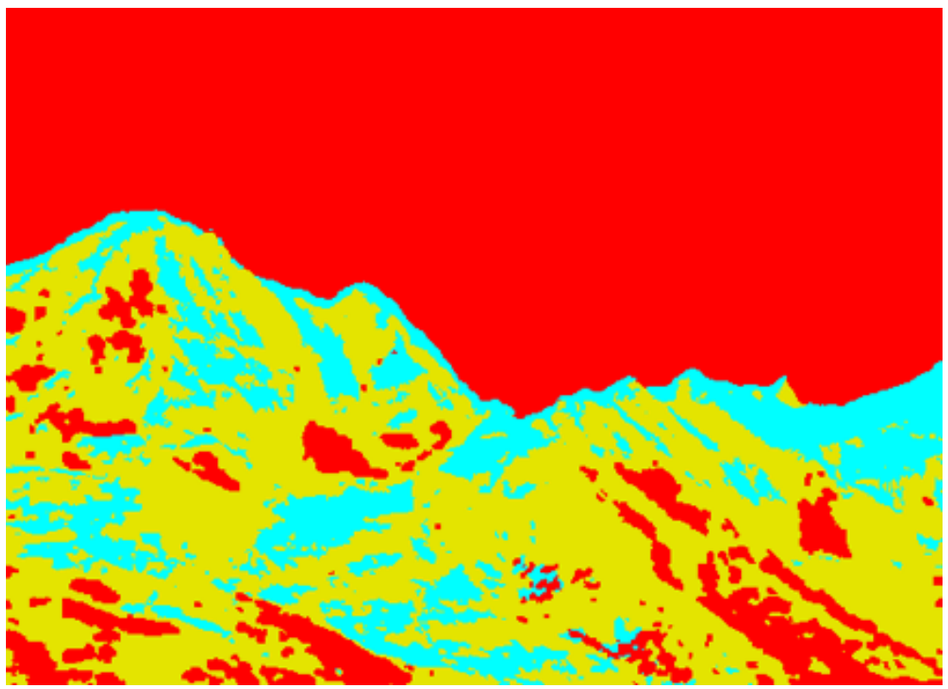
```
imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, true);
```





```
imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, 0.2f, true);
```



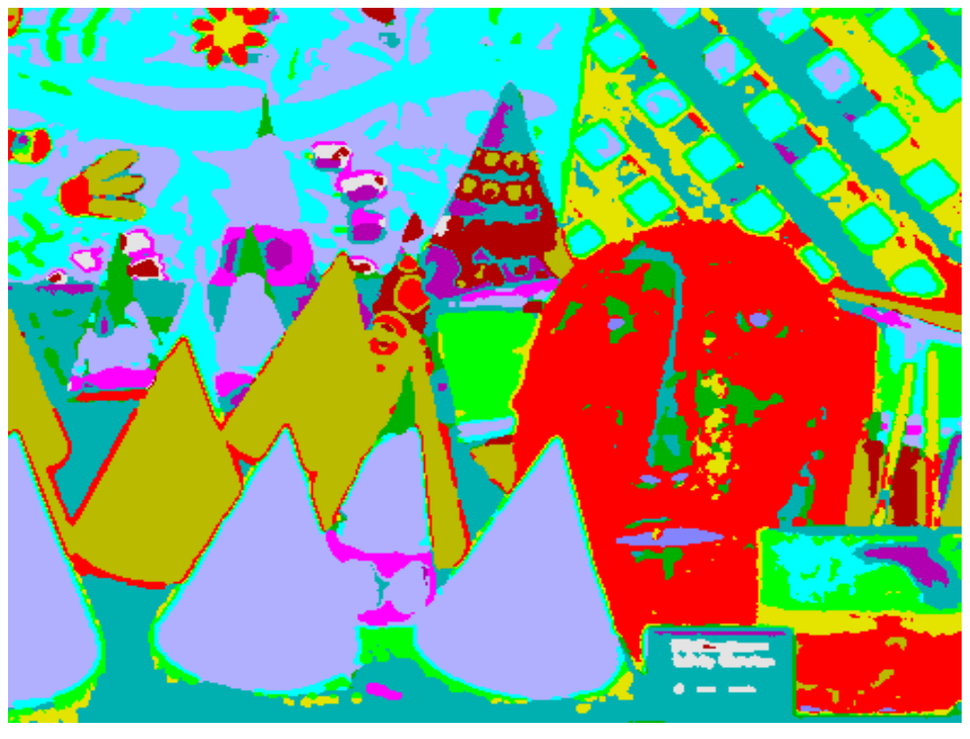


```
int kBands = 2; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```

```
int kBands = 3; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```

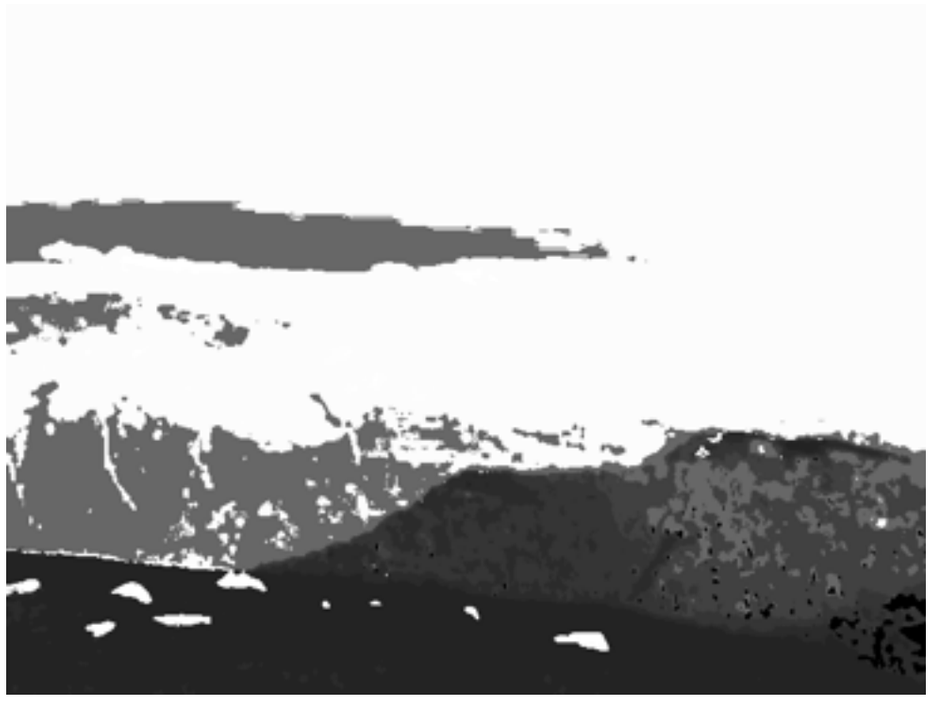
int kBands = 8; imageSegmentation.applyUsingKMPP(gsImg1, kBands);

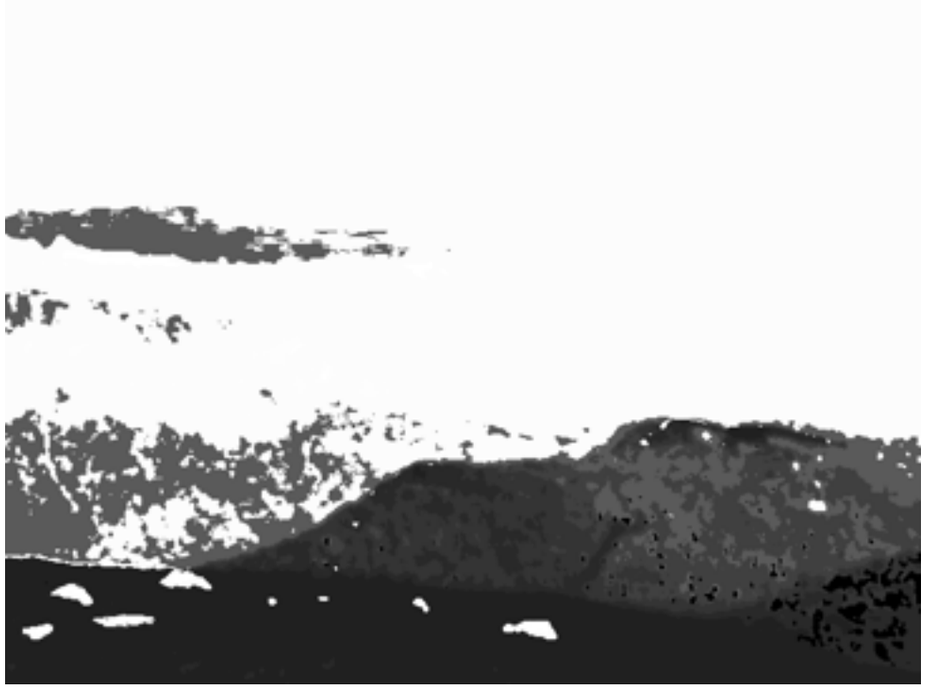
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);



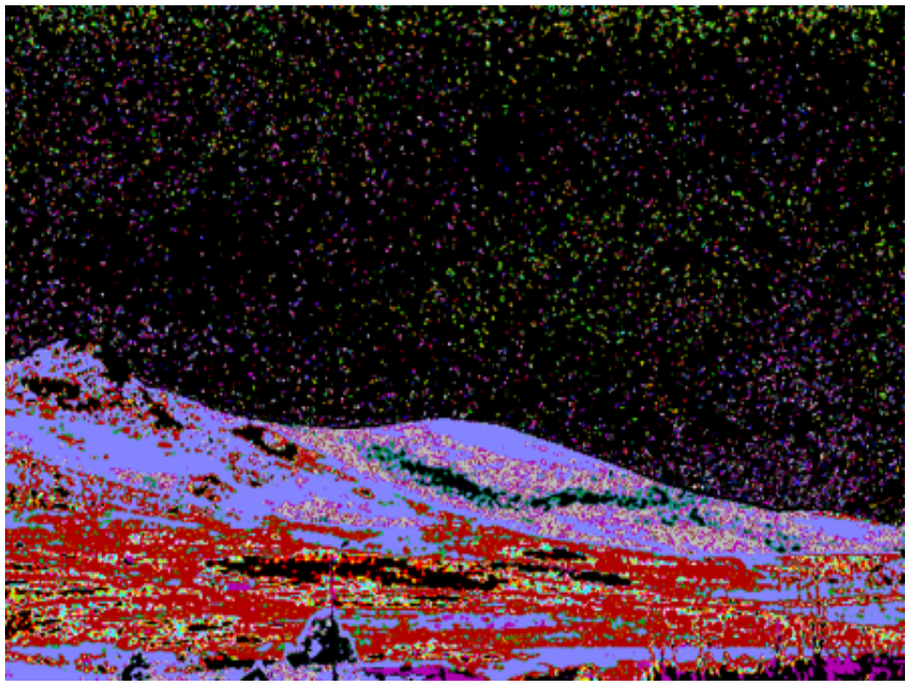
int kBands = 3; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

int kBands = 8; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

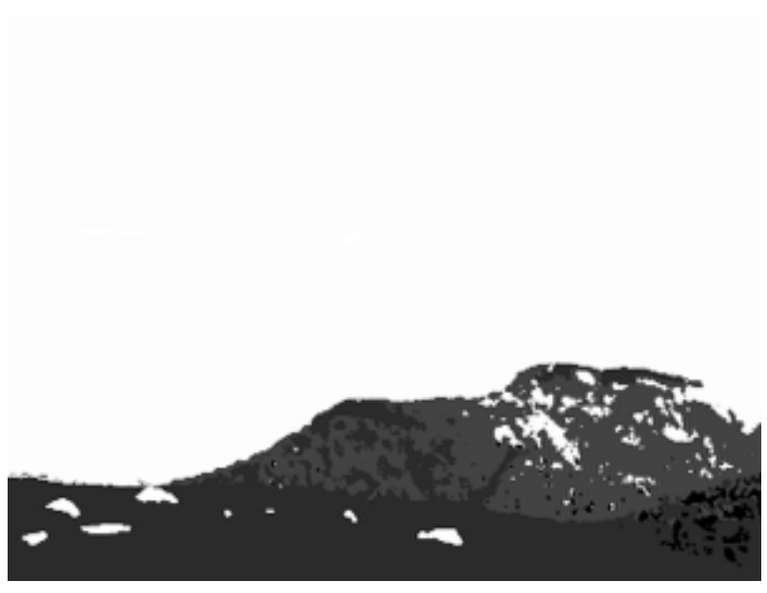




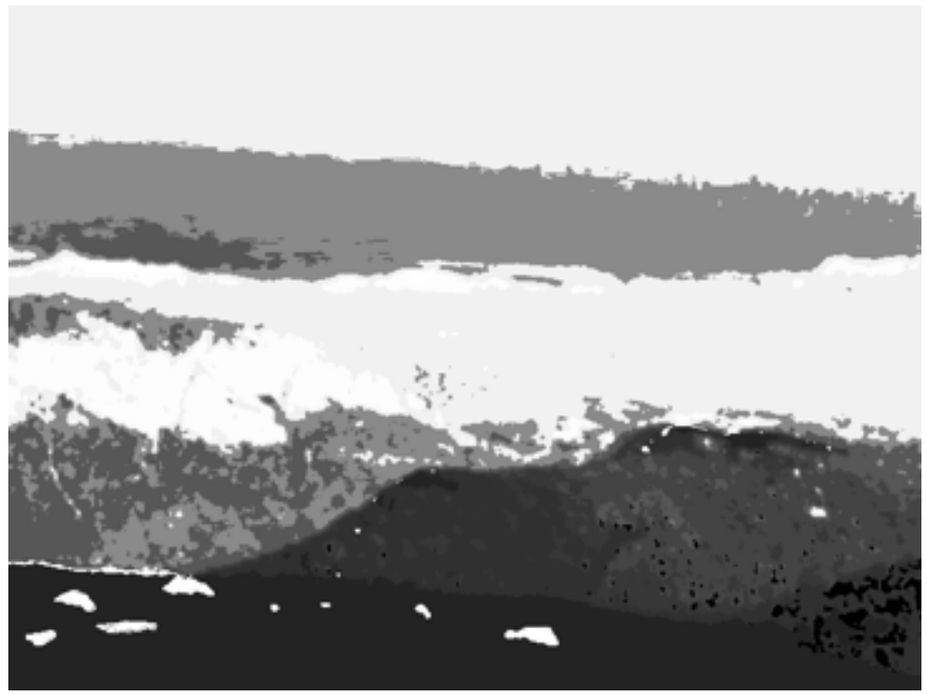
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);



int kBands =3; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);

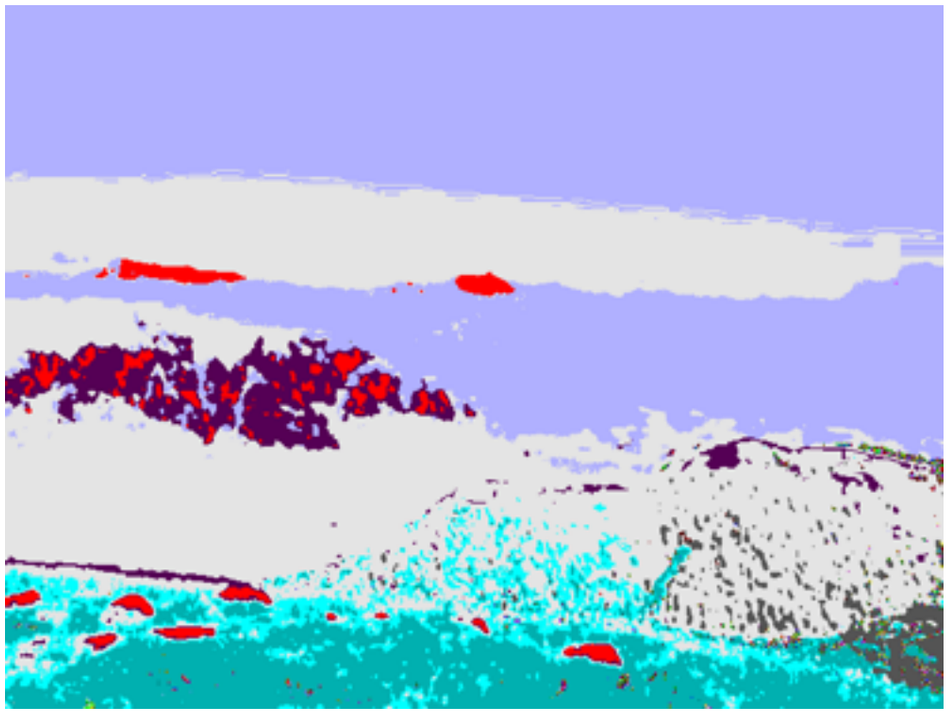


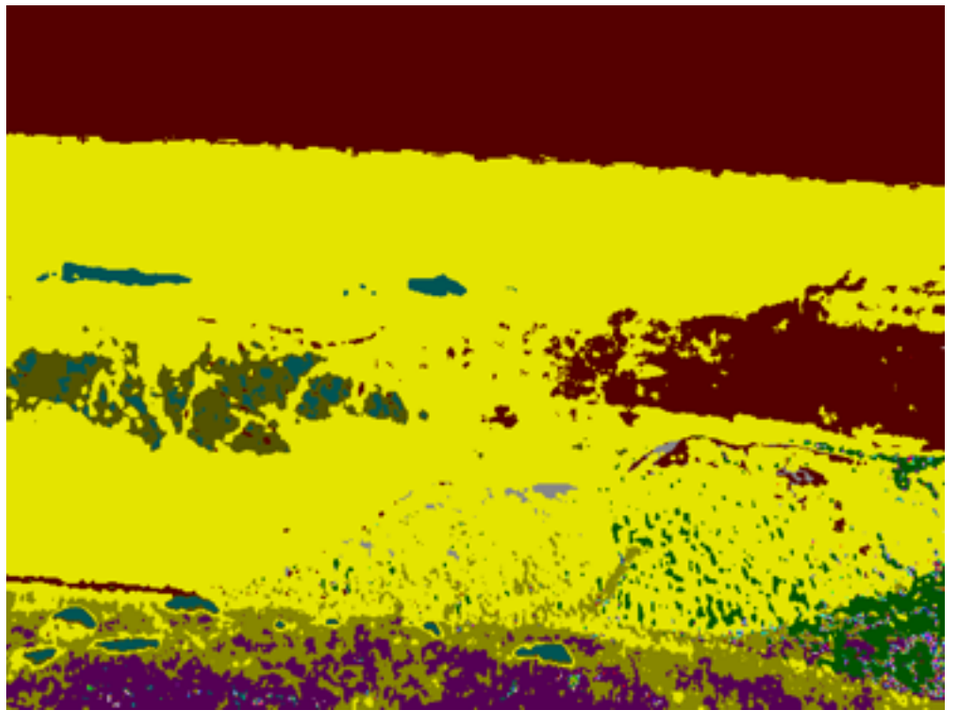
int kBands =8; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);



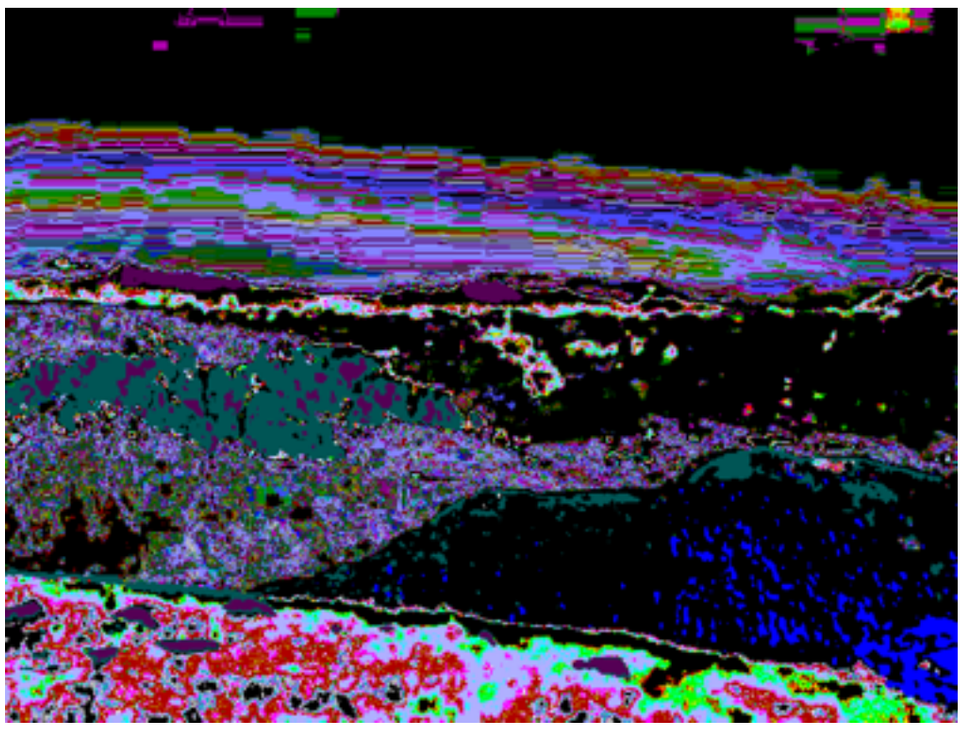
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenHistogram(gsImg1, kBands);

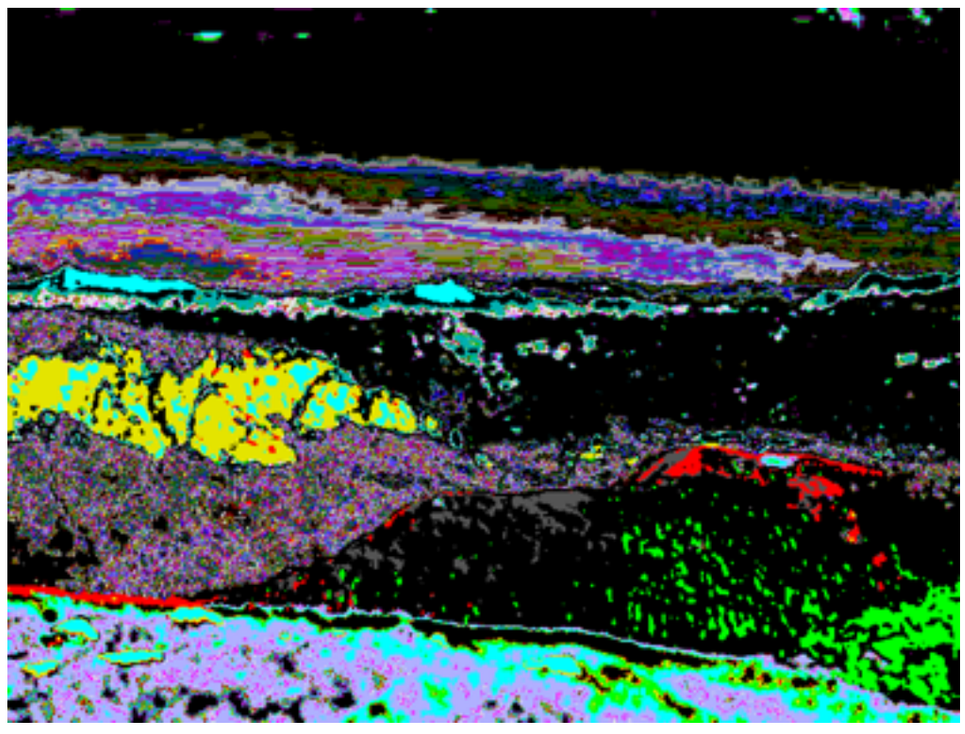
```
imageSegmentation.calculateUsingCIEXYAndClustering(gsImg1, true);
```



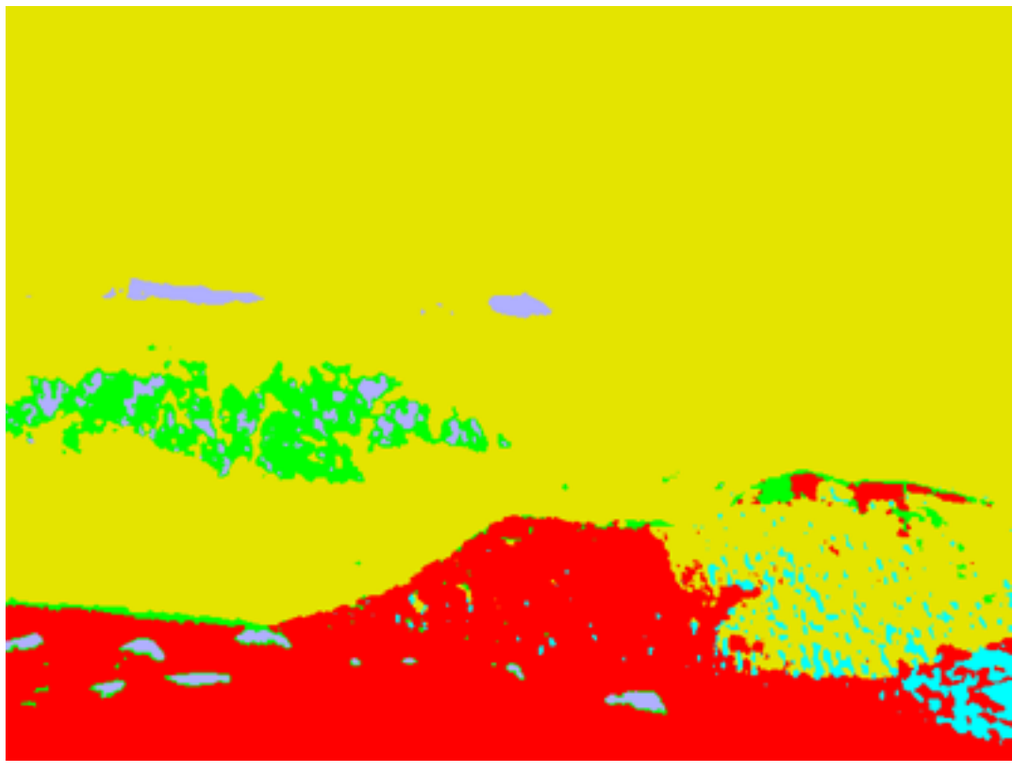


```
imageSegmentation.calculateUsingPolarCIEXYAndClustering(gsImg1, true);
```

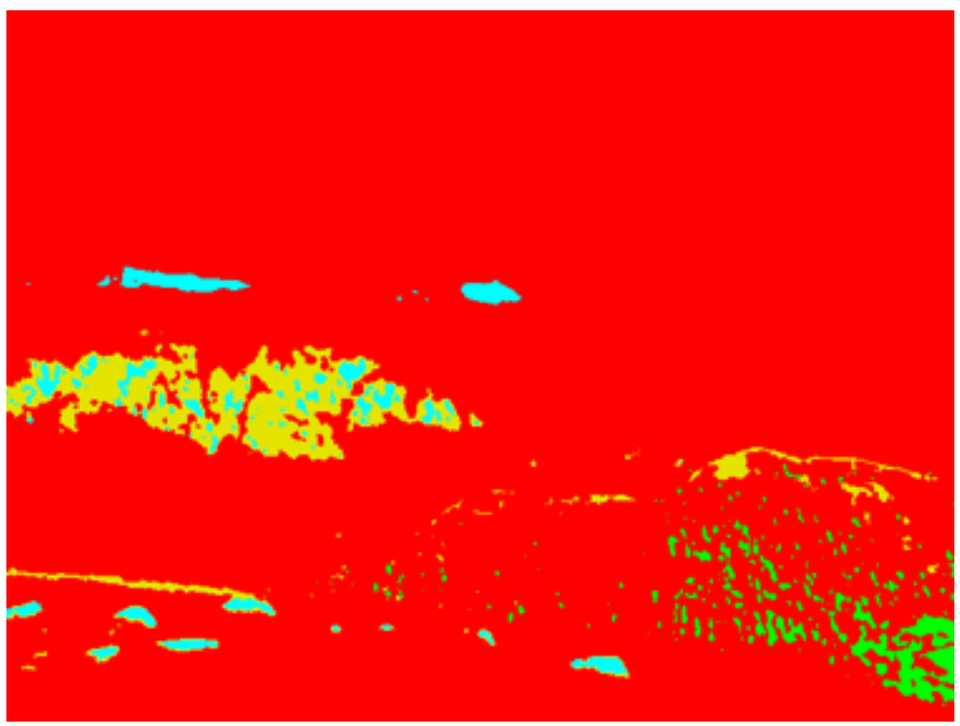


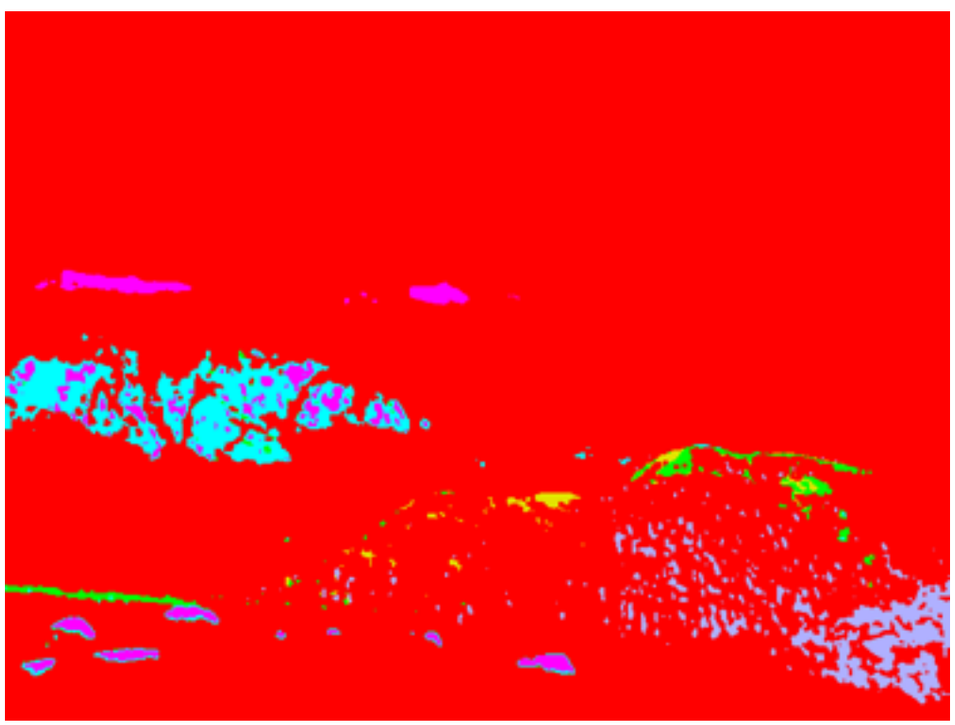


```
imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, true);
```

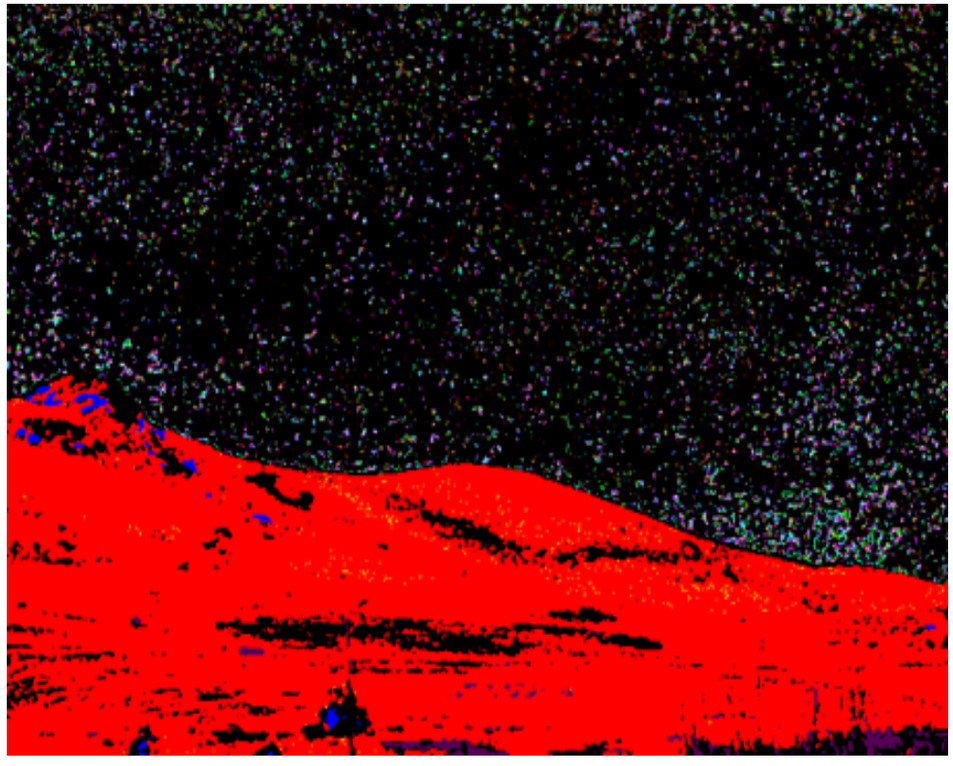


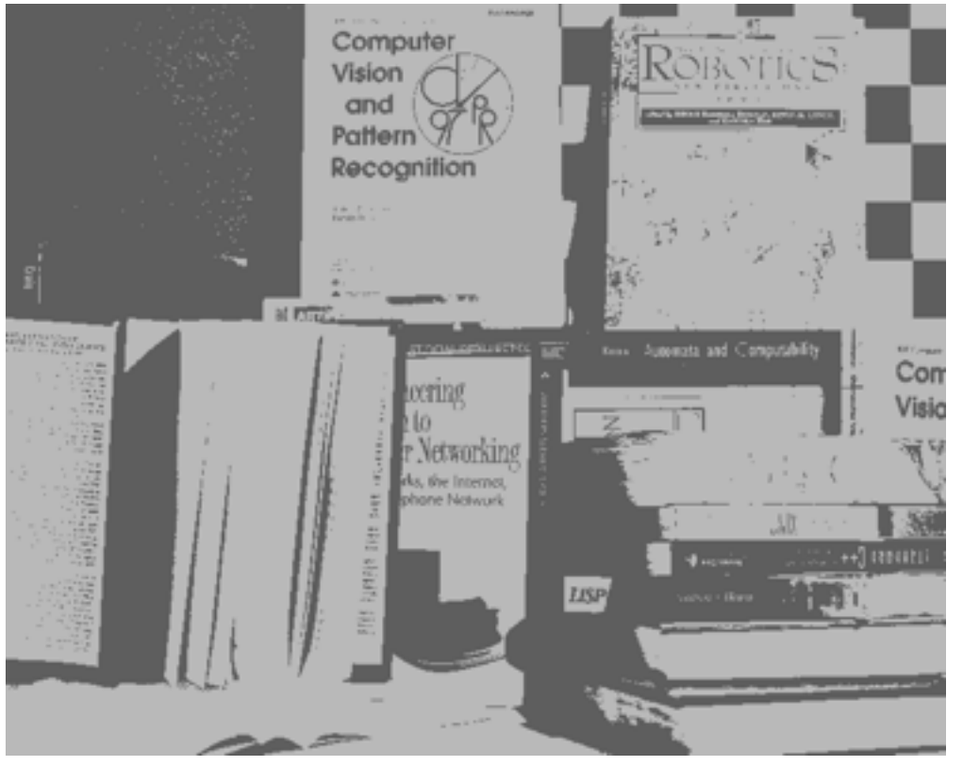
```
imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, 0.2f, true);
```



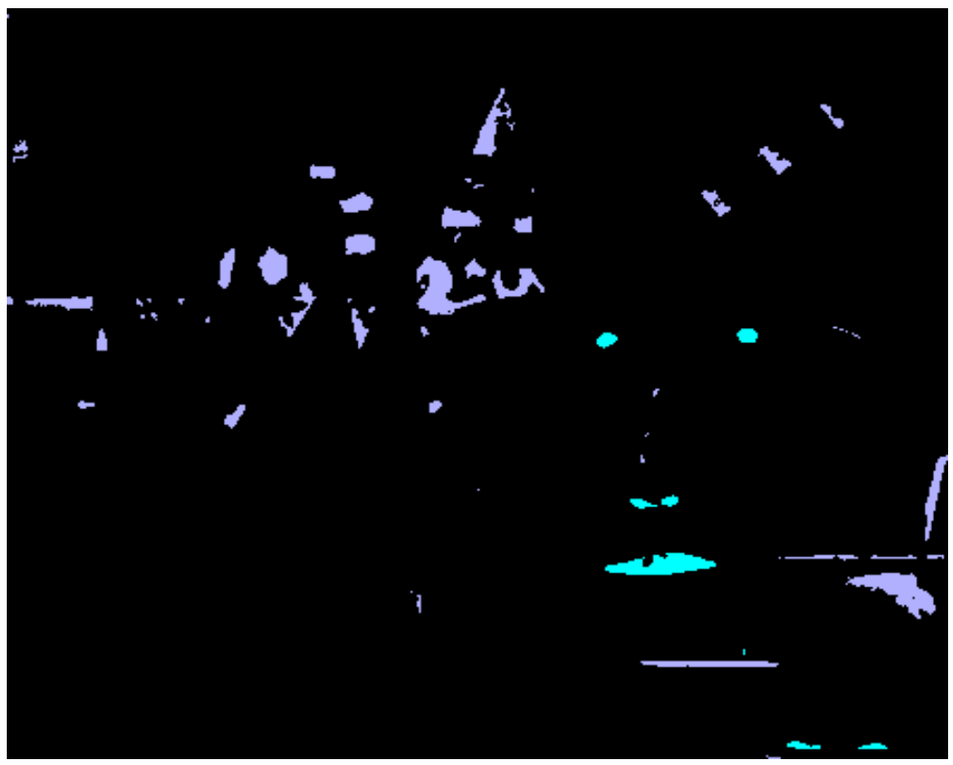


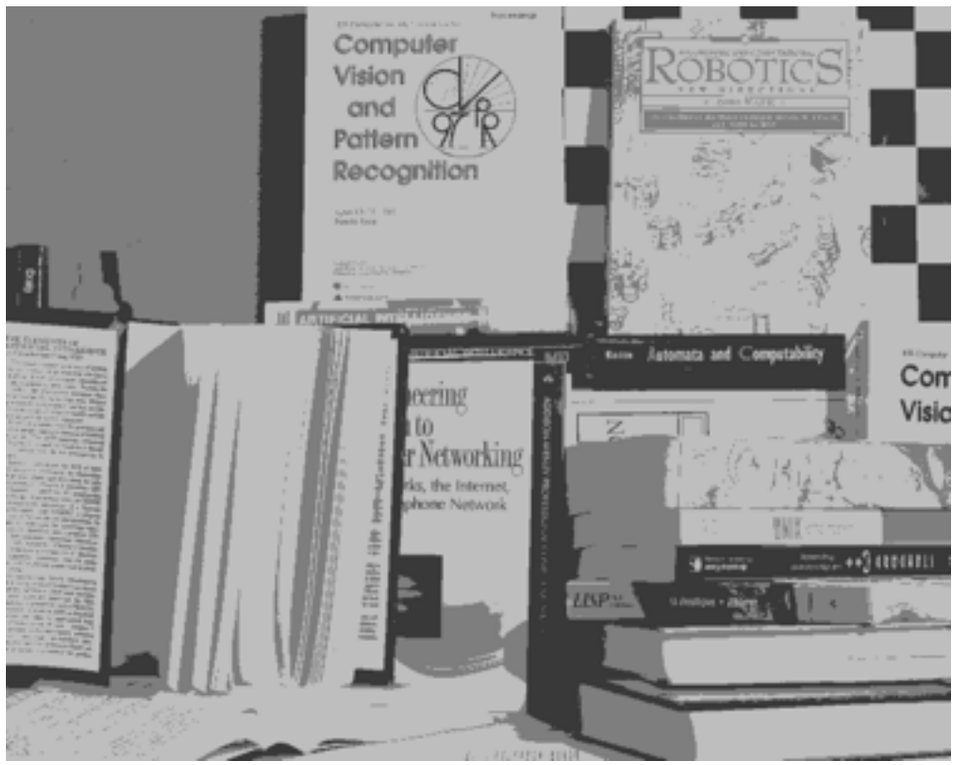
```
int kBands = 2; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```



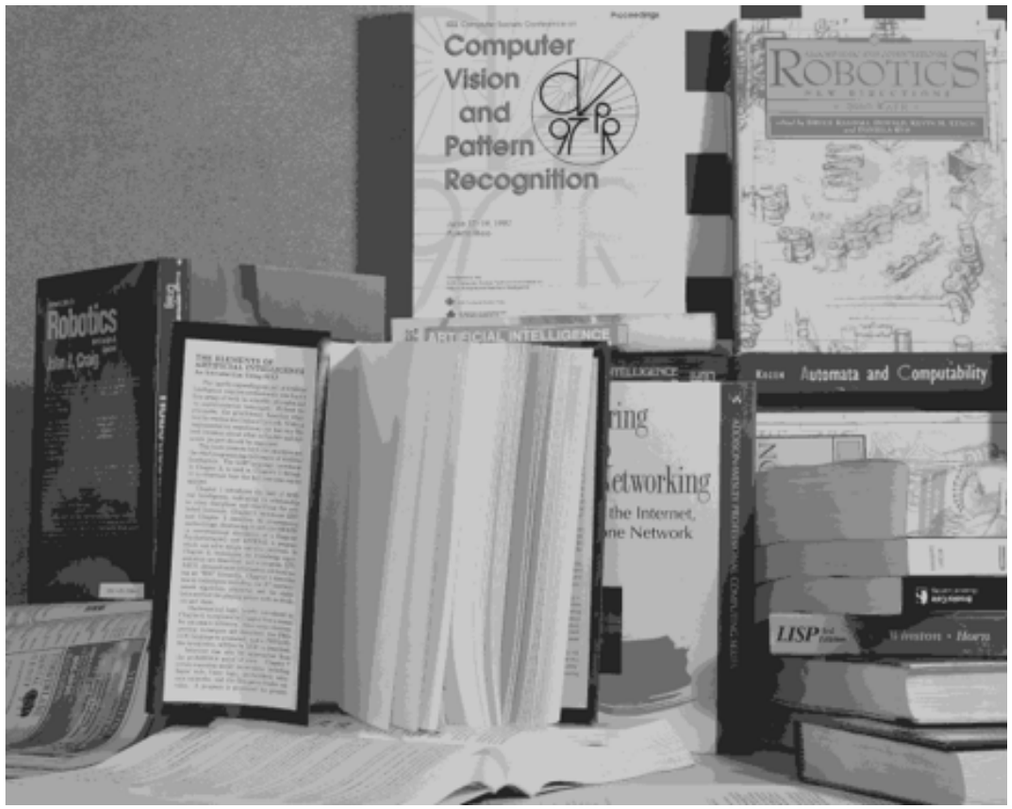


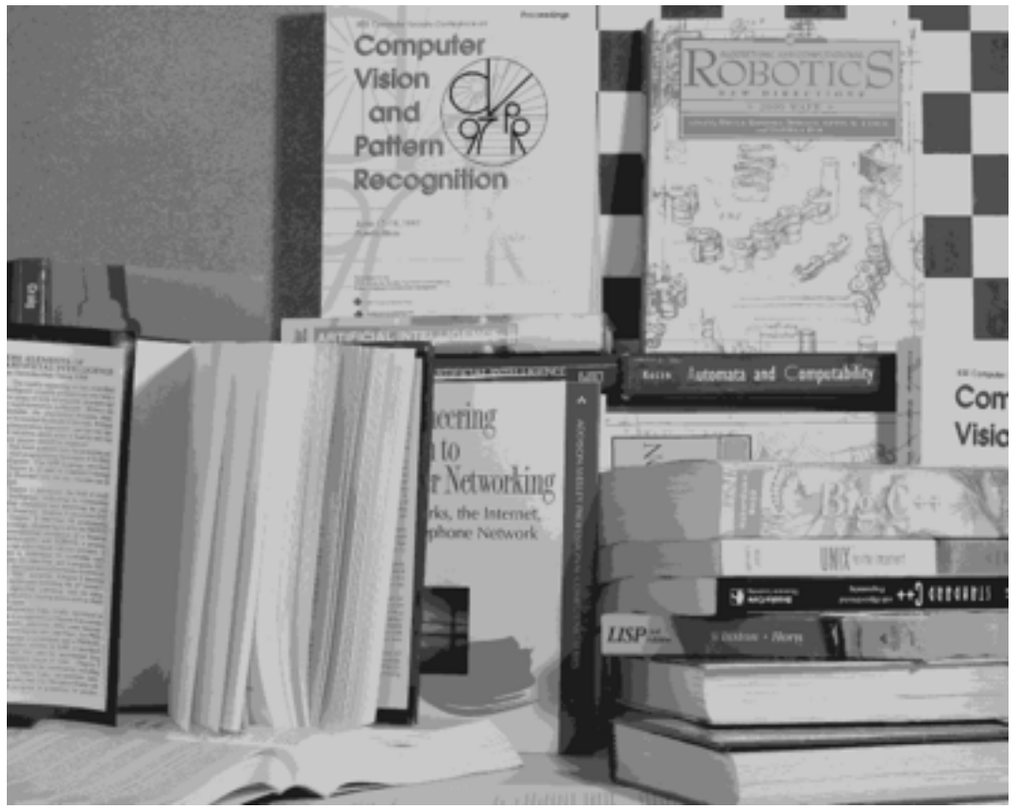

```
int kBands = 3; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```



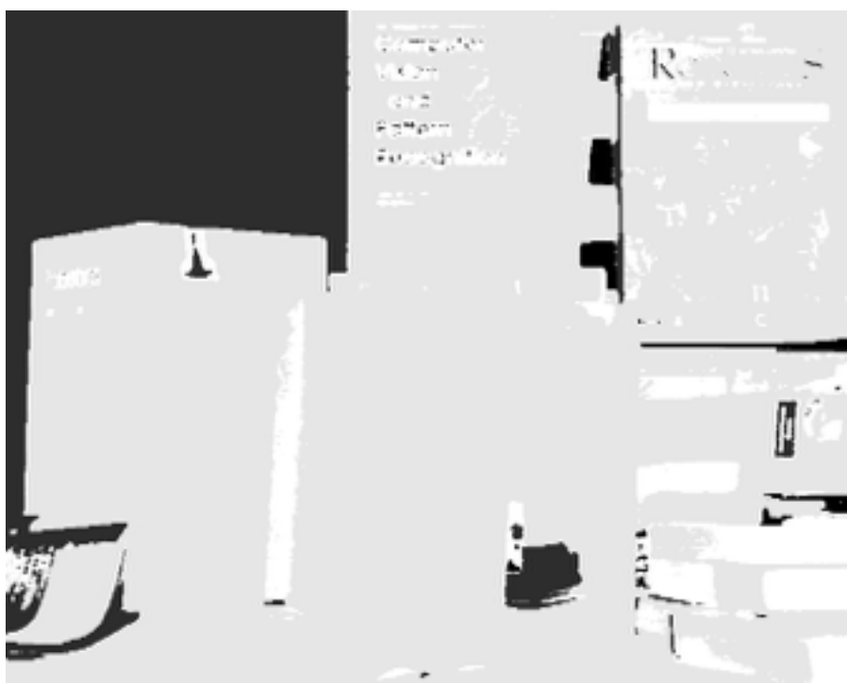


int kBands = 8; imageSegmentation.applyUsingKMPP(gsImg1, kBands);



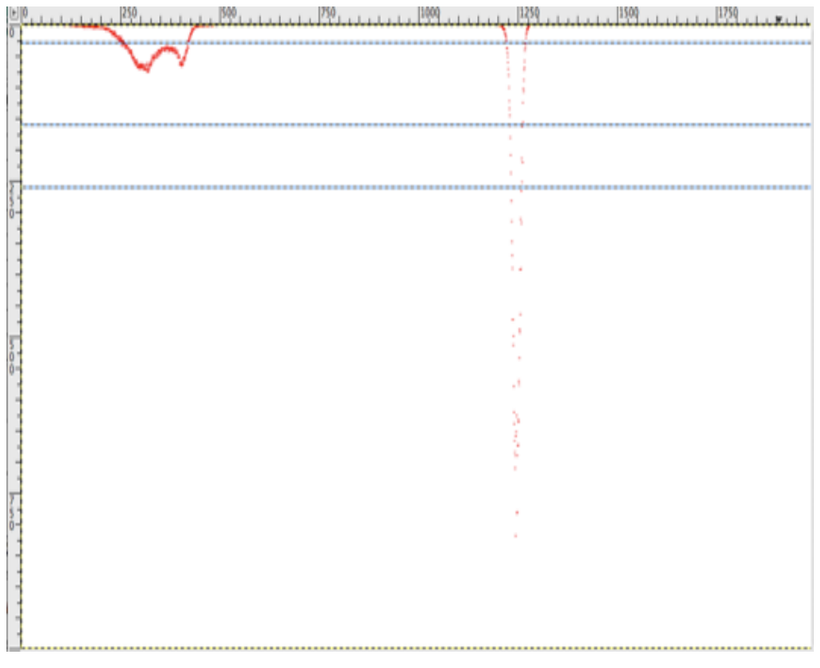


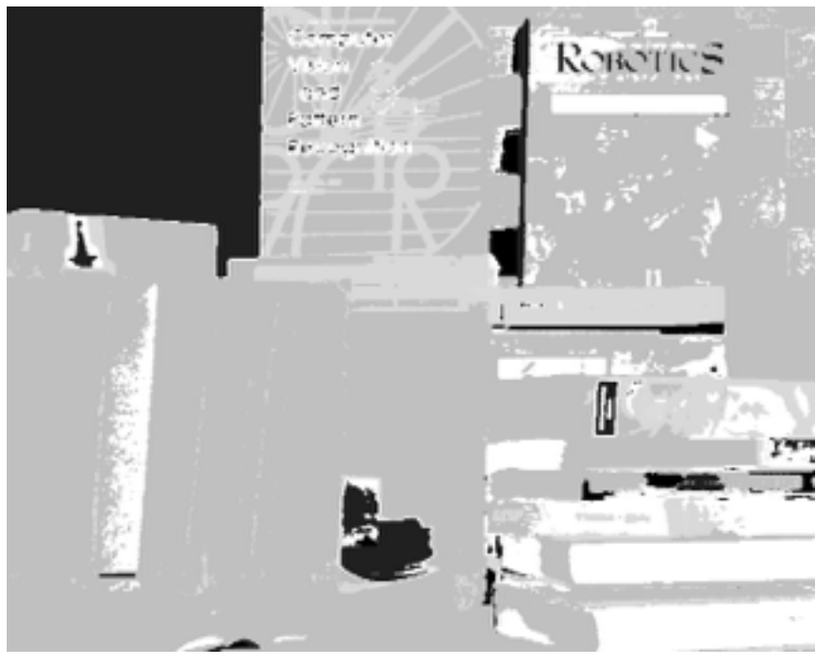
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);



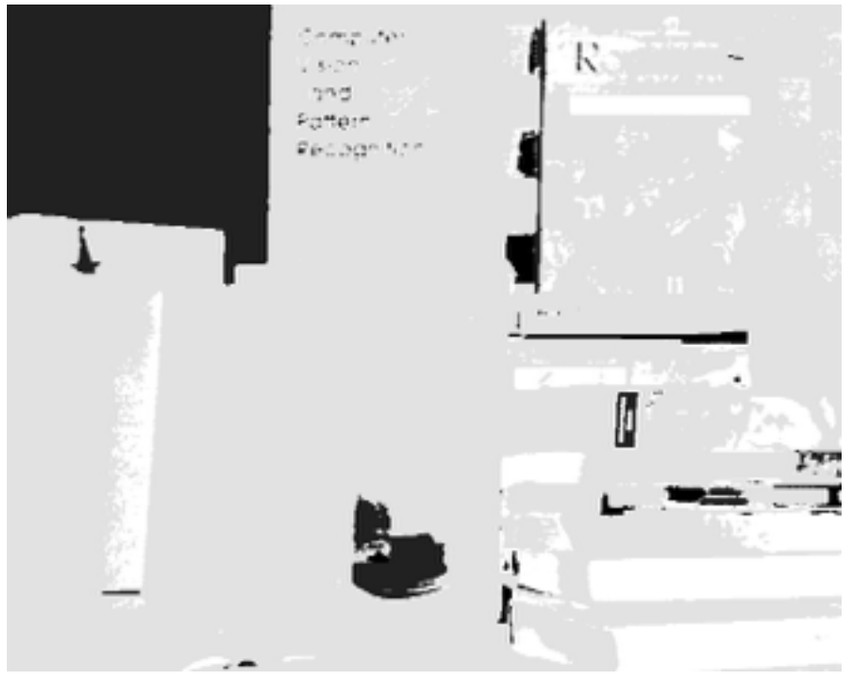
int kBands = 3; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

int kBands = 8; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

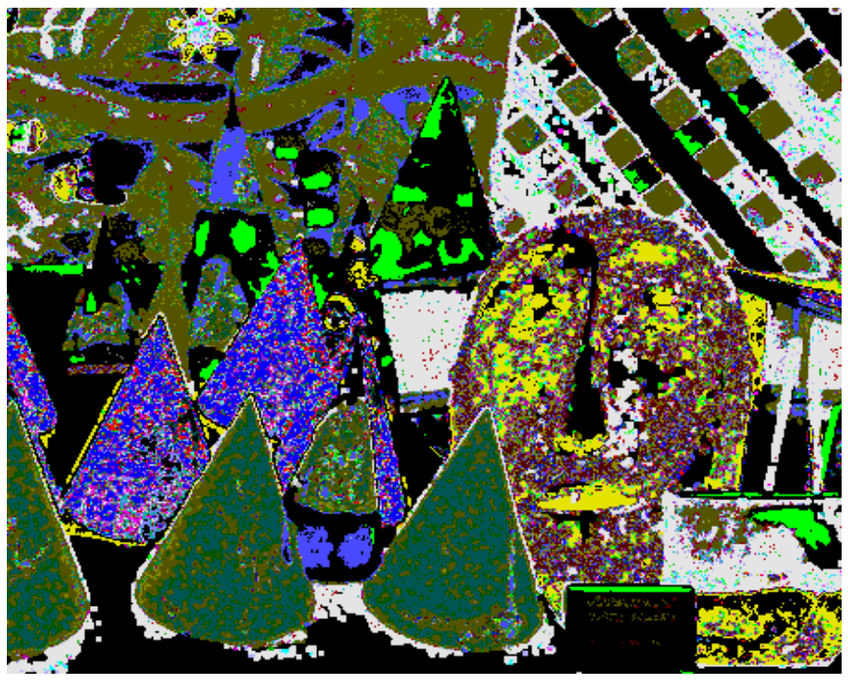




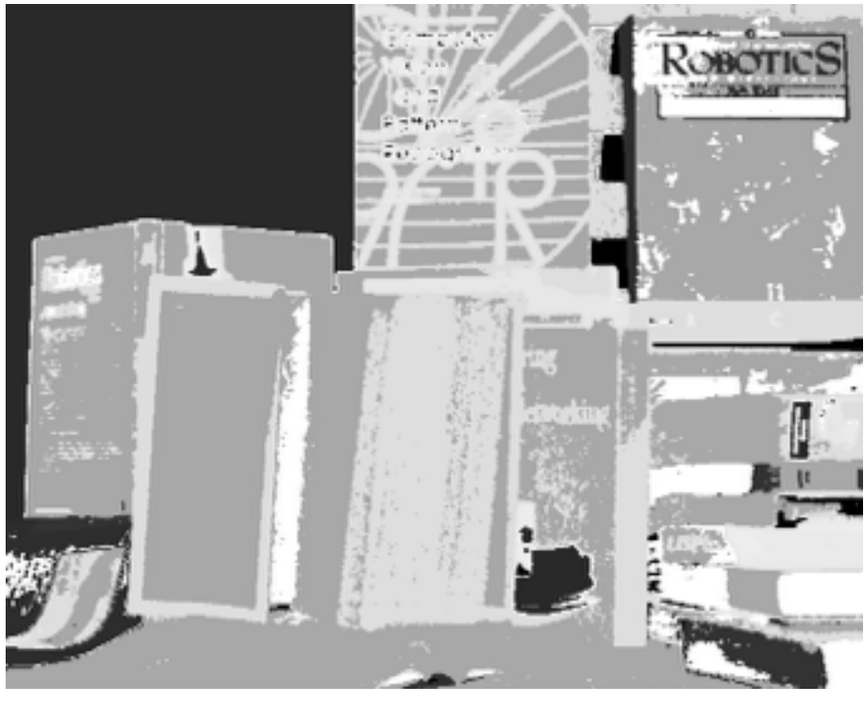
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);



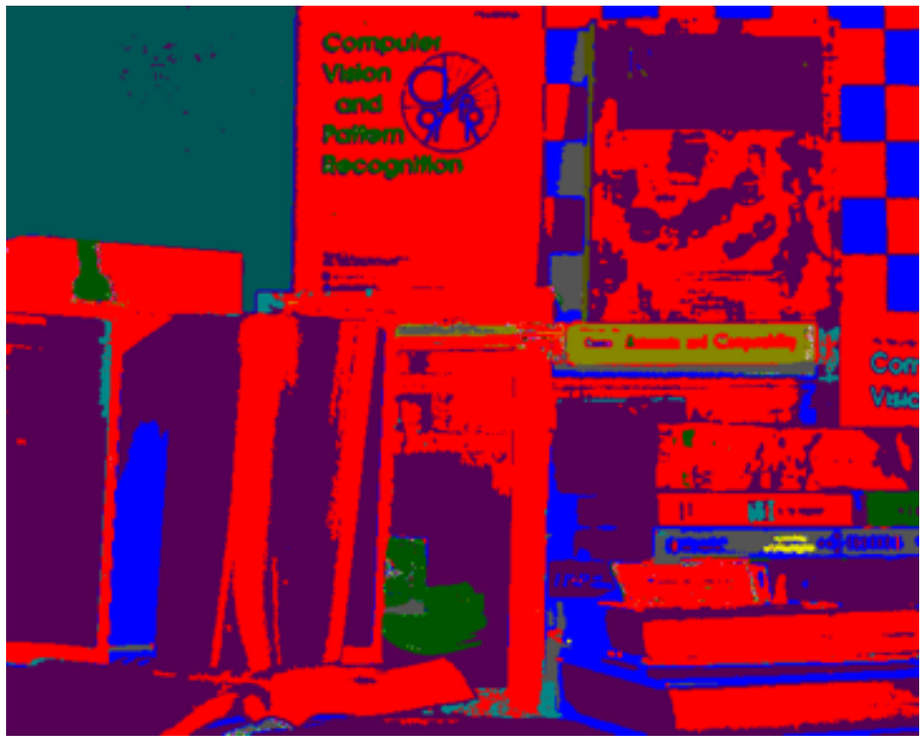
int kBands = 3; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);



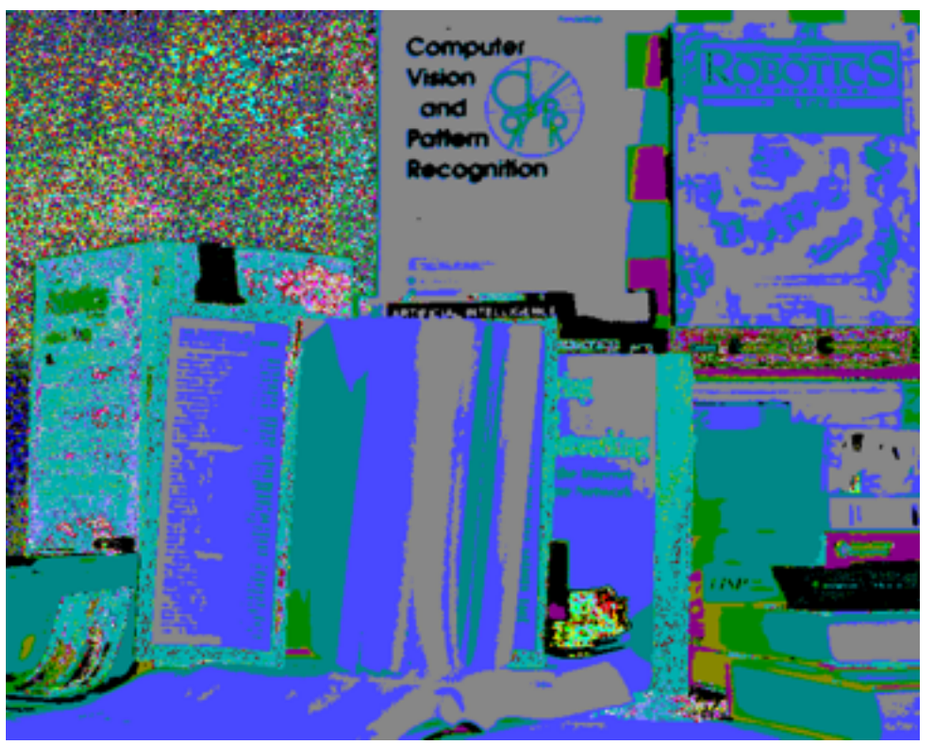
```
int kBands = 8; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);
```

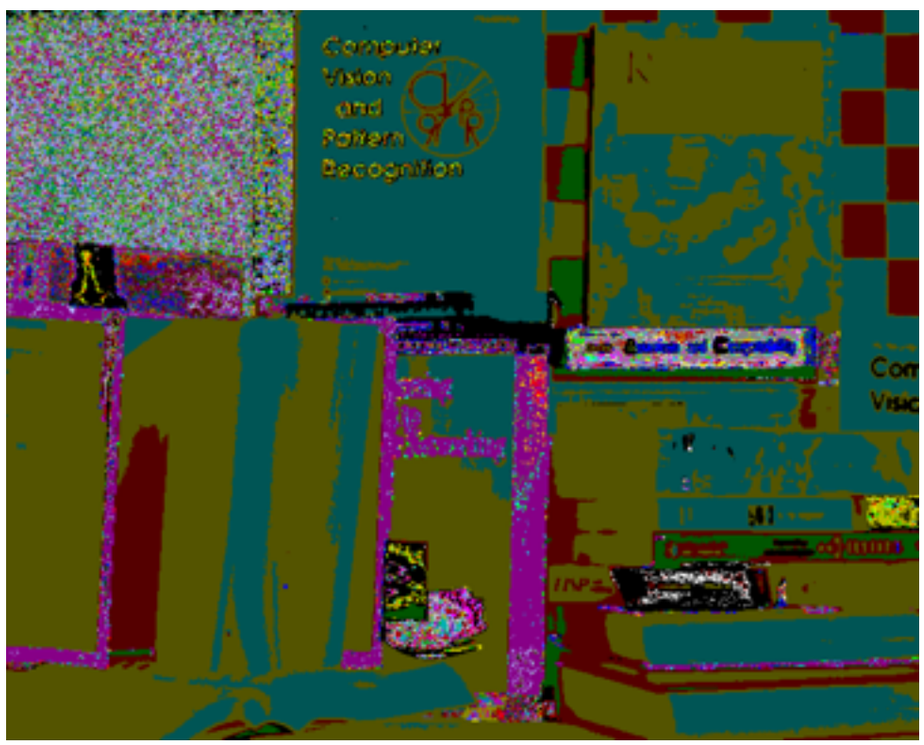


```
imageSegmentation.calculateUsingCIEXYAndClustering(gsImg1, true);
```

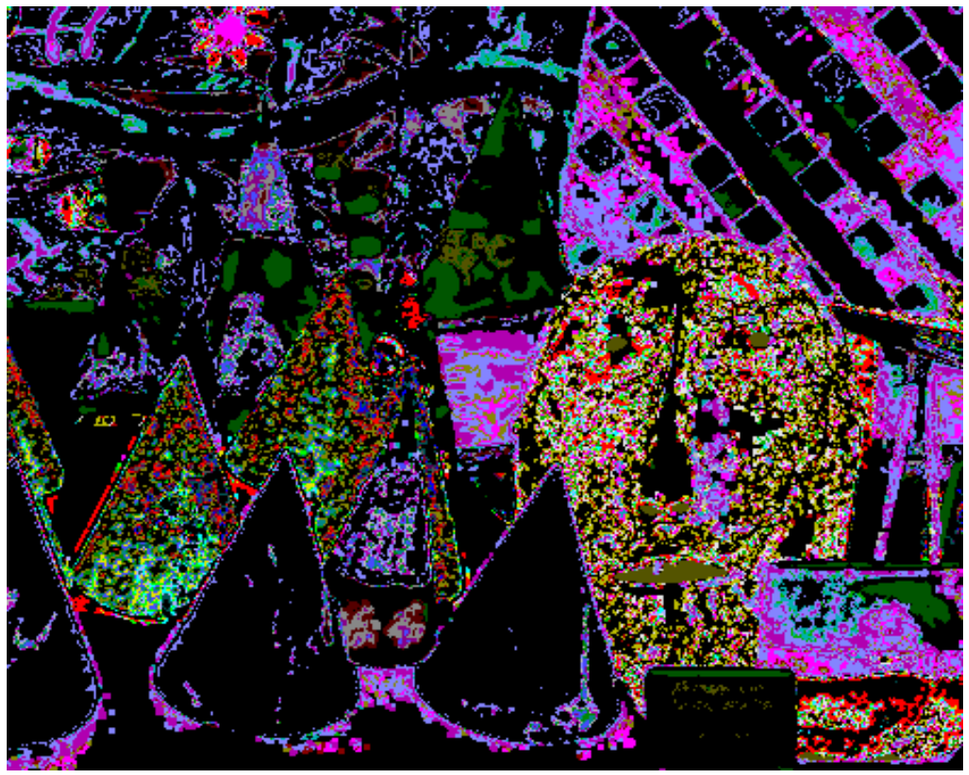


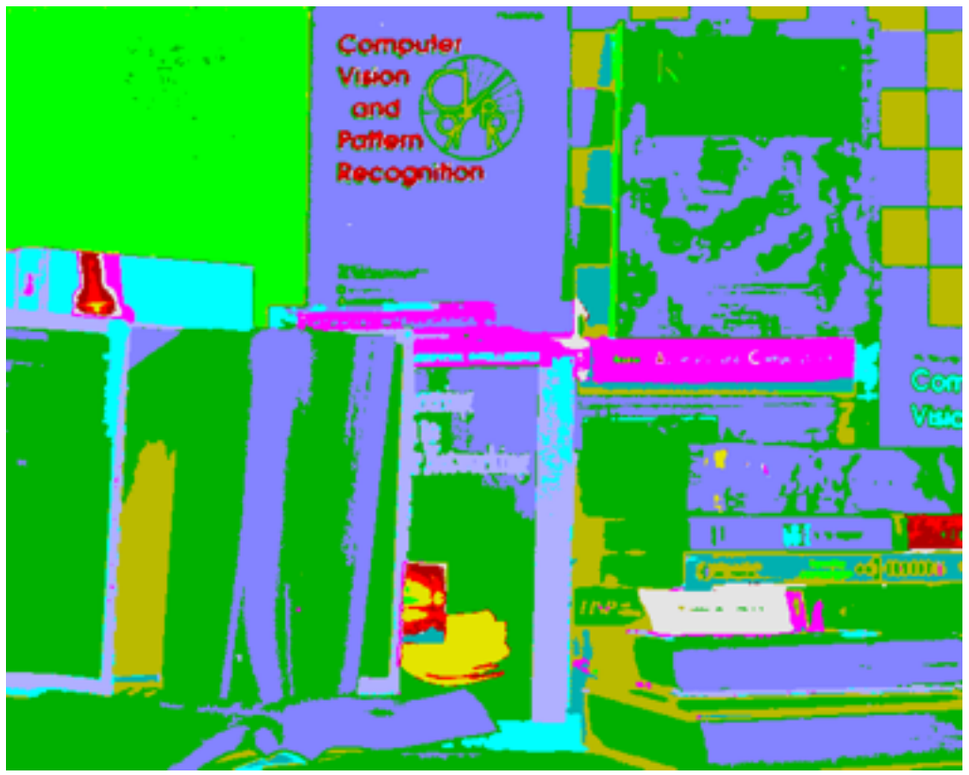

```
imageSegmentation.calculateUsingPolarCIEXYAndClustering(gsImg1, true);
```



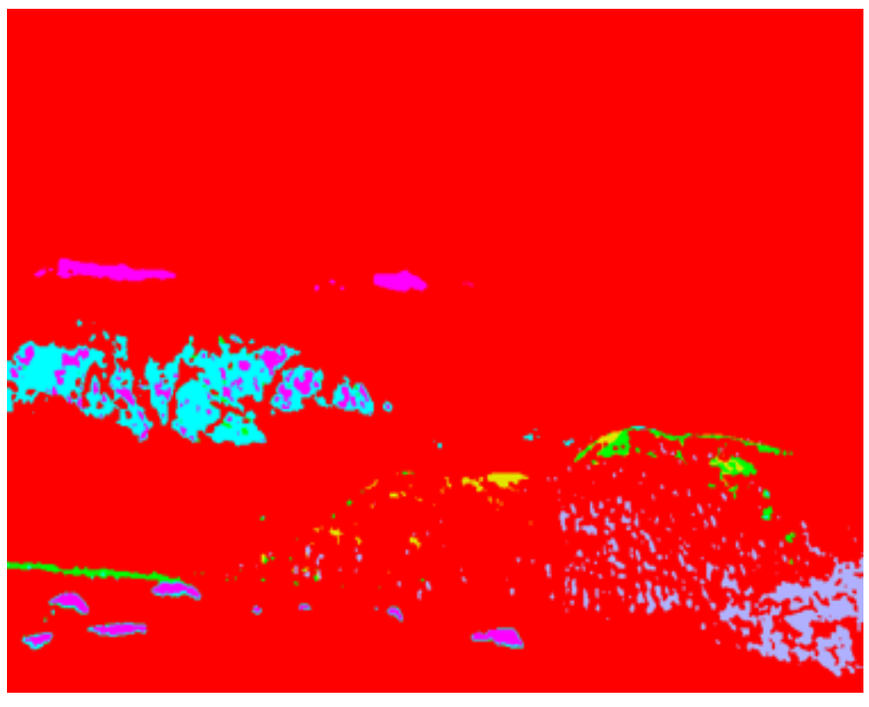


```
imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, true);
```






```
imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, 0.2f, true);
```



int kBands = 2; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
zoom in