Superpixel Segmentation

Import Modules

In [2]:

```
import vigra
from vigra import graphs
import numpy
import opengm
import matplotlib
import pylab
try:
    from sklearn.cluster import MiniBatchKMeans, KMeans
    from sklearn import mixture
except:
    raise RuntimeError("this examples nees sklearn")
```

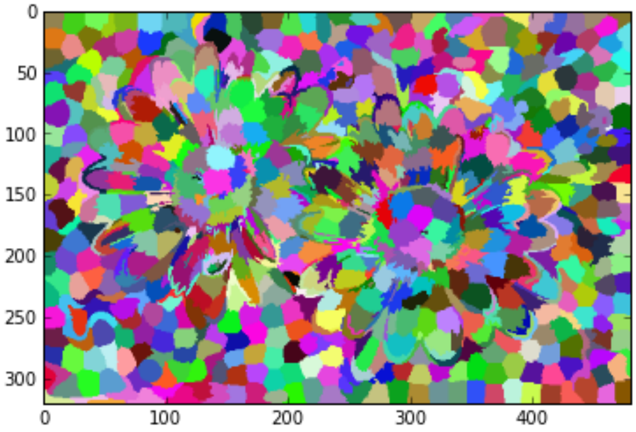
Load Image

In [3]:

```
# parameter:
filepath = '124084.jpg'   # input image path
# load image and convert to LAB
img = vigra.impex.readImage(filepath)
# get super-pixels with slic on LAB image
imgLab = vigra.colors.transform_RGB2Lab(img)

vigra.imshow(img)
vigra.show()
```

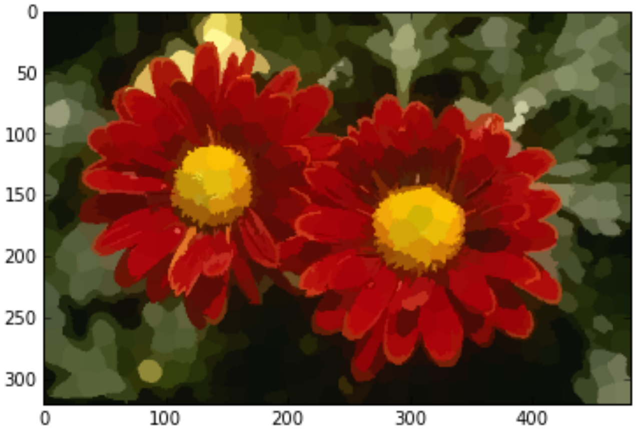
Superpixel Segmentation and RAG

In [45]:
```
superpixelDiameter = 15   # super-pixel size
slicWeight = 15.0        # SLIC color - spatial weight
labels, nseg = vigra.analysis.slicSuperpixels(imgLab, slicWeight,
                                              superpixelDiameter)
labels = vigra.analysis.labelImage(labels)-1

# get 2D grid graph and RAG
gridGraph = graphs.gridGraph(img.shape[0:2])
rag = graphs.regionAdjacencyGraph(gridGraph, labels)

# plot superpixels
cmap = matplotlib.colors.ListedColormap ( numpy.random.rand ( nseg,3))
pylab.imshow ( labels.swapaxes(0,1).squeeze(), cmap = cmap)
pylab.show()
```



Node Features

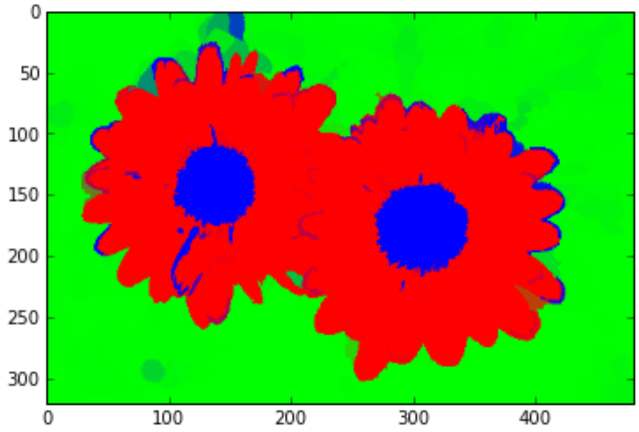
In [46]:
```
# accumulate node features from grid graph node map
# which is just a plain image (with channels)
nodeFeatures = rag.accumulateNodeFeatures(imgLab)
nodeFeaturesImg = rag.projectNodeFeaturesToGridGraph(nodeFeatures)
nodeFeaturesImg = vigra.taggedView(nodeFeaturesImg, "xyc")
nodeFeaturesImgRgb = vigra.colors.transform_Lab2RGB(nodeFeaturesImg)
vigra.imshow(nodeFeaturesImgRgb)
```

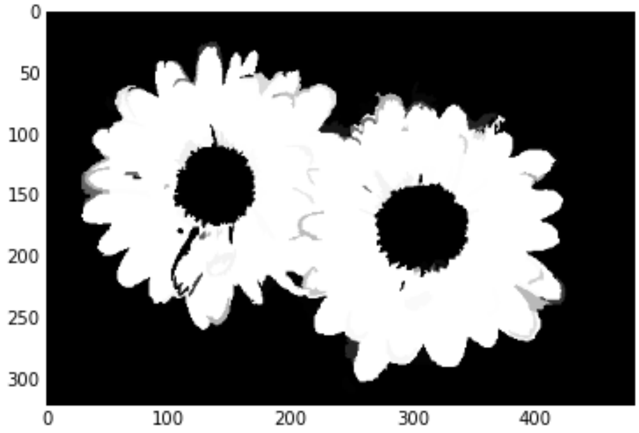


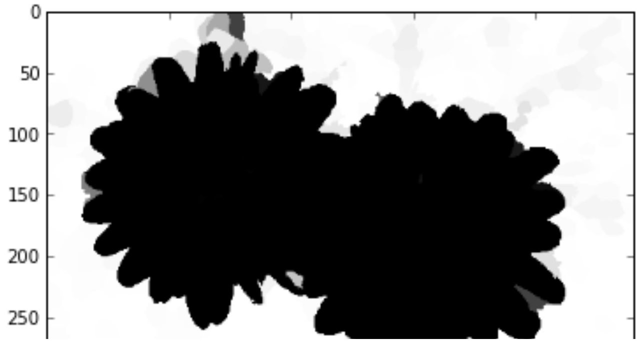
Out[46]:
```
<matplotlib.image.AxesImage at 0xf49f9d0>
```

GMM clustering to get cluster probabilities

In [53]:
```
nCluster   = 3

g = mixture.GMM(n_components=nCluster)
g.fit(nodeFeatures[:,:])
clusterProb = g.predict_proba(nodeFeatures)



clusterProbImg = rag.projectNodeFeaturesToGridGraph(clusterProb.astype(numpy.float32
clusterProbImg = vigra.taggedView(clusterProbImg, "xyc")
vigra.imshow(clusterProbImg)
vigra.show()

for c in range(nCluster):
    clusterProbImg = rag.projectNodeFeaturesToGridGraph(clusterProb[:,c].astype(nump
    clusterProbImg = vigra.taggedView(clusterProbImg, "xy")
    vigra.imshow(clusterProbImg)
    vigra.show()
```







Potts regularization

In [50]:

```
# strength of potts regularizer
beta = 10.0

# graphical model with as many variables
# as superpixels, each has 3 states
gm = opengm.gm(numpy.ones(rag.nodeNum,dtype=opengm.label_type)*nCluster)

# convert probabilites to energies
probs = numpy.clip(clusterProb, 0.00001, 0.99999)
costs = -1.0*numpy.log(probs)

# add ALL unaries AT ONCE
fids = gm.addFunctions(costs)
gm.addFactors(fids,numpy.arange(rag.nodeNum))

# add a potts function
regularizer = opengm.pottsFunction([nCluster]*2,0.0,beta)
fid = gm.addFunction(regularizer)

# get variable indices of adjacent superpixels
# - or "u" and "v" node id's for edges
uvIds = rag.uvIds()
uvIds = numpy.sort(uvIds,axis=1)

# add all second order factors at once
gm.addFactors(fid,uvIds)
```

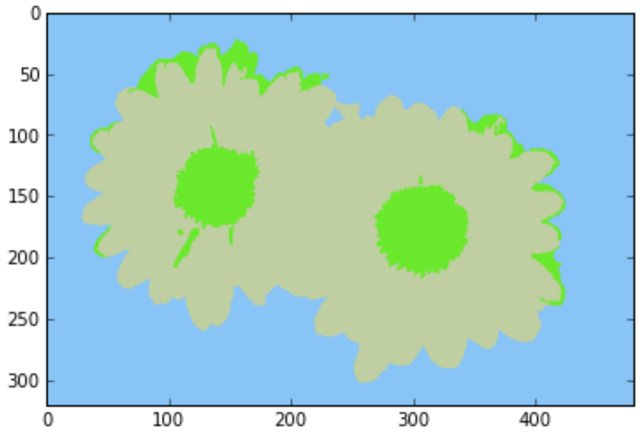
Out[50]:

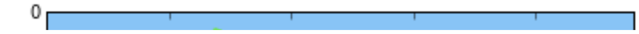
```
3184
```

In [51]:

```
Inf  = opengm.inference.BeliefPropagation
parameter = opengm.InfParam(steps=10,damping=0.5,convergenceBound=0.001)
inf = Inf(gm,parameter=parameter)


class PyCallback(object):
    def __init__(self,):
        self.labels=[]
    def begin(self,inference):
        print "begin of inference"
    def end(self,inference):
        self.labels.append(inference.arg())
    def visit(self,inference):
        gm=inference.gm()
        labelVector=inference.arg()
        print "energy  ",gm.evaluate(labelVector)
        self.labels.append(labelVector)


callback=PyCallback()
visitor=inf.pythonVisitor(callback,visitNth=1)


inf.infer(visitor)

cmap = matplotlib.colors.ListedColormap ( numpy.random.rand ( nCluster,3))
for arg in callback.labels:
    arg = vigra.taggedView(arg, "n")
    argImg = rag.projectNodeFeaturesToGridGraph(arg.astype(numpy.uint32))
    argImg = vigra.taggedView(argImg, "xy")
    # plot superpixels
    pylab.imshow ( argImg.swapaxes(0,1).squeeze(), cmap = cmap)
    pylab.show()
```

```
begin of inference
energy   3068.16256623
energy   2699.42655288
energy   2491.21469108
energy   2431.5037594
energy   2317.53743441
energy   2274.84824561
energy   2204.10503333
energy   2181.56009123
energy   2183.74267258
energy   2183.74267258
```





In [7]: