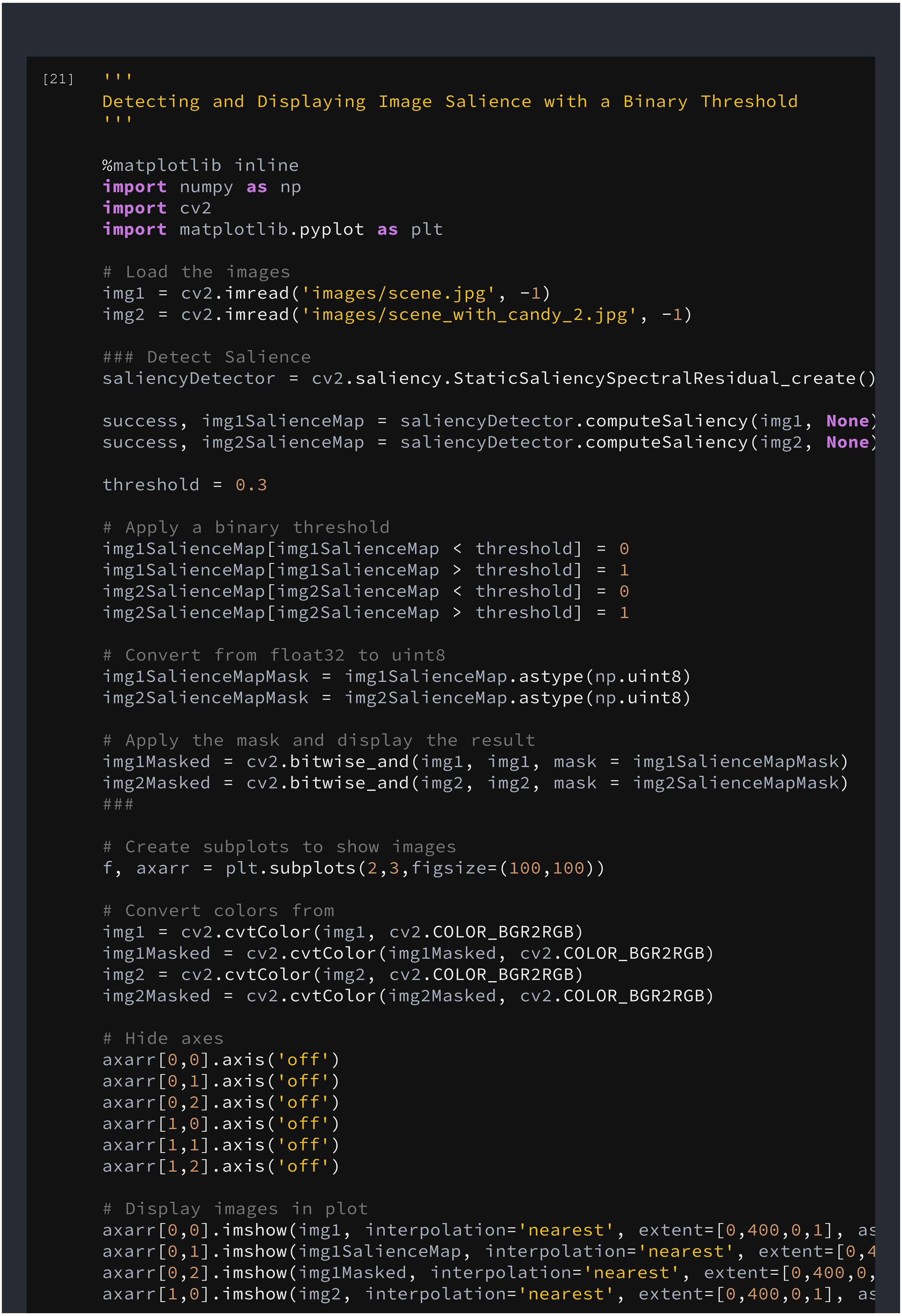
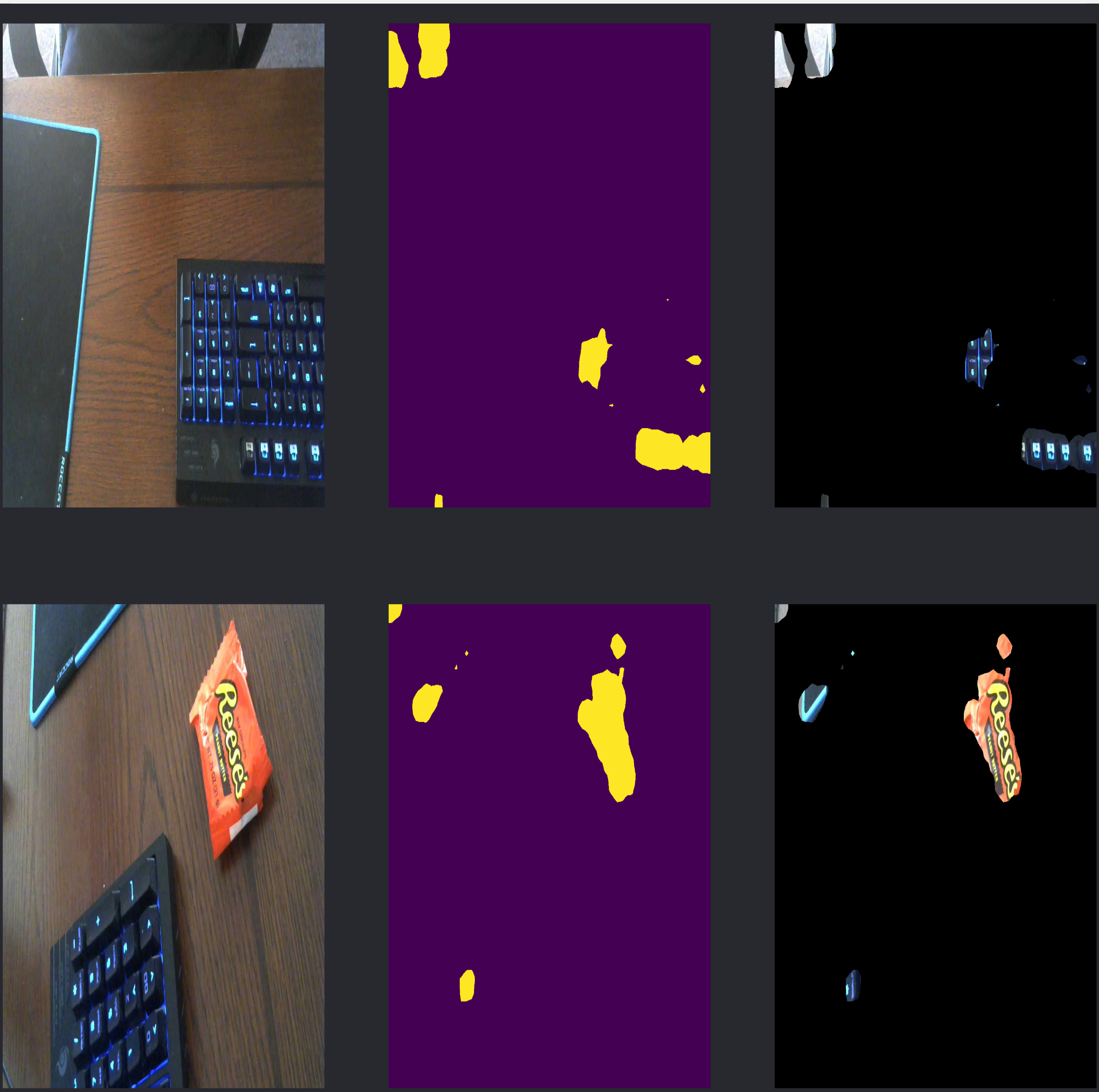
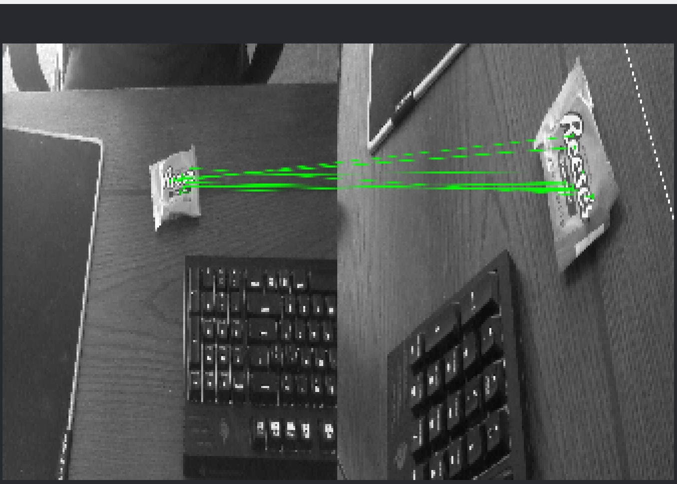


```
[21]  '''
      Detecting and Displaying Image Salience with a Binary Threshold
      '''

      %matplotlib inline
      import numpy as np
      import cv2
      import matplotlib.pyplot as plt

      # Load the images
      img1 = cv2.imread('images/scene.jpg', -1)
      img2 = cv2.imread('images/scene_with_candy_2.jpg', -1)

      ### Detect Salience
      saliencyDetector = cv2.saliency.StaticSaliencySpectralResidual_create()

      success, img1SalienceMap = saliencyDetector.computeSaliency(img1, None)
      success, img2SalienceMap = saliencyDetector.computeSaliency(img2, None)

      threshold = 0.3

      # Apply a binary threshold
      img1SalienceMap[img1SalienceMap < threshold] = 0
      img1SalienceMap[img1SalienceMap > threshold] = 1
      img2SalienceMap[img2SalienceMap < threshold] = 0
      img2SalienceMap[img2SalienceMap > threshold] = 1

      # Convert from float32 to uint8
      img1SalienceMapMask = img1SalienceMap.astype(np.uint8)
      img2SalienceMapMask = img2SalienceMap.astype(np.uint8)

      # Apply the mask and display the result
      img1Masked = cv2.bitwise_and(img1, img1, mask = img1SalienceMapMask)
      img2Masked = cv2.bitwise_and(img2, img2, mask = img2SalienceMapMask)
      ###

      # Create subplots to show images
      f, axarr = plt.subplots(2,3,figsize=(100,100))

      # Convert colors from
      img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
      img1Masked = cv2.cvtColor(img1Masked, cv2.COLOR_BGR2RGB)
      img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
      img2Masked = cv2.cvtColor(img2Masked, cv2.COLOR_BGR2RGB)

      # Hide axes
      axarr[0,0].axis('off')
      axarr[0,1].axis('off')
      axarr[0,2].axis('off')
      axarr[1,0].axis('off')
      axarr[1,1].axis('off')
      axarr[1,2].axis('off')

      # Display images in plot
      axarr[0,0].imshow(img1, interpolation='nearest', extent=[0,400,0,1], as
      axarr[0,1].imshow(img1SalienceMap, interpolation='nearest', extent=[0,4
      axarr[0,2].imshow(img1Masked, interpolation='nearest', extent=[0,400,0,
      axarr[1,0].imshow(img2, interpolation='nearest', extent=[0,400,0,1], as
      axarr[1,1].imshow(img2SalienceMap, interpolation='nearest', extent=[0,4
      axarr[1,2].imshow(img2Masked, interpolation='nearest', extent=[0,400,0,

      plt.show()
```



```
[2]  '''
     Feature Matching using ORB Feature Detector and Binary Descriptor

     ORB feature detection using an oriented FAST detection method and rota
     '''

     %matplotlib inline
     import numpy as np
     import cv2
     import matplotlib.pyplot as plt
     from PIL import Image

     # Load the images in gray scale
     img1 = cv2.imread('images/scene_with_candy.jpg', 0)
     img2 = cv2.imread('images/scene_with_candy_2.jpg', 0)

     ### Detect Salience
```

```
saliencyDetector = cv2.saliency.StaticSaliencySpectralResidual_create()

success, img1SalienceMap = saliencyDetector.computeSaliency(img1, None)
success, img2SalienceMap = saliencyDetector.computeSaliency(img2, None)

threshold = 0.3

# Apply a binary threshold
img1SalienceMap[img1SalienceMap < threshold] = 0
img1SalienceMap[img1SalienceMap > threshold] = 1
img2SalienceMap[img2SalienceMap < threshold] = 0
img2SalienceMap[img2SalienceMap > threshold] = 1

# Convert from float32 to uint8
img1SalienceMapMask = img1SalienceMap.astype(np.uint8)
img2SalienceMapMask = img2SalienceMap.astype(np.uint8)

# Apply the mask and display the result
img1Masked = cv2.bitwise_and(img1, img1, mask = img1SalienceMapMask)
img2Masked = cv2.bitwise_and(img2, img2, mask = img2SalienceMapMask)
###

# Detect the SIFT key points and compute the descriptors for the two
sift = cv2.xfeatures2d.SIFT_create()
keyPoints1, descriptors1 = sift.detectAndCompute(img1Masked, None)
keyPoints2, descriptors2 = sift.detectAndCompute(img2Masked, None)

# Create brute-force matcher object
bf = cv2.BFMatcher()

# Match the descriptors
matches = bf.knnMatch(descriptors1, descriptors2, k=2)

# Select the good matches using the ratio test
goodMatches = []

for m, n in matches:
    if m.distance < 0.7 * n.distance:
        goodMatches.append(m)

# Apply the homography transformation if we have enough good matches
MIN_MATCH_COUNT = 10

if len(goodMatches) > MIN_MATCH_COUNT:
    # Get the good key points positions
    sourcePoints = np.float32([ keyPoints1[m.queryIdx].pt for m in goo
    destinationPoints = np.float32([ keyPoints2[m.trainIdx].pt for m i

    # Obtain the homography matrix
    M, mask = cv2.findHomography(sourcePoints, destinationPoints, meth
    matchesMask = mask.ravel().tolist()

    # Apply the perspective transformation to the source image corners
    h, w = img1.shape
    corners = np.float32([ [0, 0], [0, h - 1], [w - 1, h - 1], [w - 1
    transformedCorners = cv2.perspectiveTransform(corners, M)

    # Draw a polygon on the second image joining the transformed corne
    img2 = cv2.polylines(img2, [np.int32(transformedCorners)], True, (
else:
    print("Not enough matches are found - %d/%d" % (len(goodMatches),
    matchesMask = None

# Draw the matches
drawParameters = dict(matchColor=(0, 255, 0), singlePointColor=None, ma
result = cv2.drawMatches(img1, keyPoints1, img2, keyPoints2, goodMatche

# Display the results
plt.axis("off")
plt.imshow(result, extent=[0,400,0,1], aspect='auto')
plt.show()
```



[3]

```
'''
Change Detection based on
Feature Matching using ORB Feature Detector and Binary Descriptor
ORB feature detection using an oriented FAST detection method and rota
'''

%matplotlib inline
import numpy as np
import cv2
import matplotlib.pyplot as plt
from PIL import Image

# Load the images in gray scale
img1 = cv2.imread('images/scene.jpg', 0)
img2 = cv2.imread('images/scene_with_candy_2.jpg', 0)

### Detect Salience
saliencyDetector = cv2.saliency.StaticSaliencySpectralResidual_create()

success, img1SalienceMap = saliencyDetector.computeSaliency(img1, None)
success, img2SalienceMap = saliencyDetector.computeSaliency(img2, None)

threshold = 0.3

# Apply a binary threshold
img1SalienceMap[img1SalienceMap < threshold] = 0
img1SalienceMap[img1SalienceMap > threshold] = 1
img2SalienceMap[img2SalienceMap < threshold] = 0
img2SalienceMap[img2SalienceMap > threshold] = 1

# Convert from float32 to uint8
```

```
img1SalienceMapMask = img1SalienceMap.astype(np.uint8)
img2SalienceMapMask = img2SalienceMap.astype(np.uint8)

# Apply the mask and display the result
img1Masked = cv2.bitwise_and(img1, img1, mask = img1SalienceMapMask)
img2Masked = cv2.bitwise_and(img2, img2, mask = img2SalienceMapMask)
###

# Detect the SIFT key points and compute the descriptors for the two 
sift = cv2.xfeatures2d.SIFT_create()
keyPoints1, descriptors1 = sift.detectAndCompute(img1Masked, None)
keyPoints2, descriptors2 = sift.detectAndCompute(img2Masked, None)

# Create brute-force matcher object
bf = cv2.BFMatcher()

# Match the descriptors
matches = bf.knnMatch(descriptors1, descriptors2, k=2)

# Select the bad matches using the ratio test
badMatches = []

for m, n in matches:
    if m.distance > 0.7 * n.distance:
        badMatches.append(m)

# Get the bad key points positions
changePoints = np.float32([ keyPoints1[m.queryIdx].pt for m in badMatc

# This contains the coordinates of points in the region of a salient 
# found in the second image and not found (or very poorly matched) in 
print(changePoints)

# TODO: Find a way to represent this visually
```

```
[[[ 229.53855896    7.375844  ]]

 [[ 230.5677948    19.32374191]]

 [[ 211.94612122   22.69588852]]

 [[ 230.76502991   25.47104836]]

 [[ 205.4480896    34.30665207]]

 [[ 198.32571411   50.86026764]]

 [[ 224.84307861   57.03881836]]

 [[ 171.69572449   64.88794708]]

 [[ 211.47601318   74.22544098]]

 [[ 144.74691772   77.30522156]]

 [[  27.23398209   81.1124115 ]]
 [[  837.20483398   461.73580933]]

 [[  844.39379883   462.046875  ]]

 [[  809.42150879   469.54943848]]

 [[  809.42150879   469.54943848]]

 [[  809.42150879   469.54943848]]

 [[  785.21478271   480.4564209 ]]

 [[  865.60321045   494.3069458 ]]

 [[  794.39093018   499.17404175]]

 [[  825.34661865   501.14175415]]

 [[  762.39251709   502.56716919]]

 [[ 1200.03161621   503.56106567]]

 [[  835.85119629   504.16522217]]

 [[  759.03863525   509.13619995]]

 [[  756.3394165    518.31365967]]

 [[  782.64758301   523.02661133]]

 [[  787.76922607   524.25268555]]

 [[  812.66717529   537.37817383]]

 [[  812.66717529   537.37817383]]

 [[ 1257.91381836   545.40692139]]

 [[  985.06292725   624.45648193]]

 [[ 1170.05957031   628.21533203]]

 [[  986.49438477   629.95153809]]

 [[ 1078.64672852   632.94445801]]

 [[ 1171.37634277   632.74133301]]

 [[  986.52301025   635.66766357]]

 [[ 1228.36694336   638.60046387]]
 [[ 1051.58117676   639.08148193]]

 [[  994.56677246   641.58862305]]

 [[ 1098.0324707    641.97485352]]

 [[ 1229.57189941   641.8805542 ]]

 [[ 1237.5838623    650.18792725]]

 [[ 1002.21112061   651.09130859]]

 [[ 1066.57617188   660.05749512]]

 [[ 1064.14331055   662.51190186]]

 [[  227.28561401     5.05622292]]

 [[  233.40689087     6.73576546]]

 [[  227.53100586    10.52461338]]

 [[  234.34568787    12.20573997]]

 [[  227.58129883    17.30895233]]

 [[   27.54380989    18.62885094]]

 [[  233.89118958    19.22638893]]

 [[  228.01742554    22.67527771]]

 [[  231.35072327    31.8449173 ]]

 [[  231.35072327    31.8449173 ]]

 [[  141.87113953    32.4144783 ]]

 [[  220.15280151    43.66368103]]

 [[  221.83921814    56.61723709]]

 [[  178.82362366    75.53662872]]

 [[  142.6370697     80.20145416]]

 [[  152.93693542    80.09256744]]

 [[   33.50662994    81.14445496]]

 [[  858.67950439   456.60464478]]
 [[  857.18927002   460.50054932]]

 [[  837.3659668    461.58920288]]

 [[  806.05041504   467.61535645]]

 [[  859.17370605   482.09909058]]

 [[  865.33630371   484.03552246]]

 [[  859.98516846   494.29193115]]

 [[  795.9519043    501.62234497]]

 [[  849.90228271   501.58514404]]

 [[  849.90228271   501.58514404]]

 [[  815.71917725   534.27813721]]

 [[ 1103.17797852   623.11181641]]

 [[ 1044.08984375   625.55700684]]

 [[ 1042.2668457    629.02044678]]

 [[ 1042.2668457    629.02044678]]

 [[ 1112.29785156   629.43383789]]

 [[ 1099.51171875   629.98754883]]

 [[ 1099.51171875   629.98754883]]

 [[ 1081.51049805   630.74963379]]

 [[ 1158.12219238   631.64611816]]

 [[ 1158.12219238   631.64611816]]

 [[ 1158.12219238   631.64611816]]

 [[ 1113.38452148   633.54022217]]

 [[  993.9307251    638.16870117]]

 [[ 1021.65161133   639.11315918]]

 [[ 1030.55688477   639.27514648]]

 [[ 1030.55688477   639.27514648]]
 [[ 1052.32177734   639.48443604]]

 [[ 1052.32177734   639.48443604]]

 [[ 1034.3104248    642.51226807]]

 [[ 1139.69104004   642.4788208 ]]

 [[ 1039.63647461   643.39440918]]

 [[ 1256.27075195   644.20031738]]

 [[ 1256.27075195   644.20031738]]

 [[ 1090.00219727   644.91564941]]

 [[ 1090.00219727   644.91564941]]

 [[ 1037.20629883   646.60021973]]

 [[ 1205.83740234   662.35717773]]

 [[  214.53674316    29.80657959]]

 [[  229.646698      39.92161942]]

 [[  226.37136841    42.43355942]]

 [[  201.33580017    51.56173706]]

 [[   44.92500305    63.19553757]]

 [[   61.89330673    72.92066956]]

 [[ 1109.59362793   410.81500244]]

 [[ 1109.59362793   410.81500244]]

 [[  881.00878906   478.52713013]]

 [[  881.00878906   478.52713013]]

 [[  806.91778564   492.94665527]]

 [[  830.11773682   500.90710449]]

 [[  835.80291748   501.19000244]]

 [[ 1231.90905762   505.55044556]]

 [[ 1223.06140137   506.02078247]]
```

```
 [[  815.70916748   521.38165283]]

 [[ 1248.46362305   626.00292969]]

 [[ 1161.12268066   628.42767334]]

 [[ 1241.04821777   629.67785645]]

 [[ 1244.13220215   633.26513672]]

 [[ 1028.6809082    634.31994629]]

 [[ 1028.6809082    634.31994629]]

 [[  987.96539307   637.37548828]]

 [[ 1094.4597168    637.7911377 ]]

 [[ 1085.56359863   639.90997314]]

 [[ 1085.56359863   639.90997314]]

 [[ 1256.54260254   644.28436279]]

 [[ 1256.54260254   644.28436279]]

 [[ 1095.59020996   648.8918457 ]]

 [[ 1157.12756348   651.11706543]]

 [[  240.98355103    16.1060524 ]]

 [[  120.1446228     16.8957901 ]]

 [[  244.87963867    20.84566879]]

 [[  208.8500824     24.48658371]]

 [[  208.8500824     24.48658371]]

 [[  214.8861084     29.1664257 ]]

 [[  139.85588074    36.70783234]]

 [[  215.11701965    39.2916069 ]]

 [[  197.31500244    50.15283203]]

 [[  176.24282837    67.6316452 ]]

 [[   49.91285706    69.12474823]]
```

```
[[  190.79133606    75.98268127]]

[[   63.92201233    82.3267746 ]]

[[   42.34514618    96.85280609]]

[[  856.03259277   452.15533447]]

[[  799.72802734   472.19680786]]

[[  778.90740967   501.46884155]]

[[  788.64050293   501.30813599]]

[[  764.97515869   504.19619751]]

[[  765.1394043    519.02502441]]

[[ 1043.58154297   638.28961182]]

[[ 1112.55712891   640.71820068]]

[[ 1160.04589844   641.14477539]]

[[ 1047.9119873    643.17169189]]

[[ 1146.74584961   643.40979004]]

[[ 1247.07702637   642.51501465]]

[[ 1233.9543457    644.31945801]]

[[ 1233.9543457    644.31945801]]

[[ 1107.13085938   644.98303223]]

[[  123.36654663    26.9052124 ]]

[[  151.60946655    42.26722717]]

[[   33.02424622    45.03487778]]

[[  217.97053528    53.4055748 ]]

[[  224.07052612    64.01406097]]

[[  785.71575928   475.19967651]]

[[  770.95947266   477.36889648]]

[[  810.89990234   498.24307251]]
```

```
        [[  810.89990234   498.24307251]]

        [[  810.89990234   498.24307251]]

        [[  863.12988281   498.95019531]]

        [[  863.12988281   498.95019531]]

        [[  817.02374268   504.93984985]]

        [[  801.13671875   523.06762695]]

        [[ 1102.13098145   639.90905762]]

        [[ 1154.26428223   645.77819824]]

        [[ 1240.99975586   646.71557617]]

        [[  202.79678345    55.61571121]]

        [[  219.0362854     75.67868805]]

        [[   49.78216934    91.8459549 ]]

        [[  815.49755859   465.34283447]]

        [[  841.52935791   468.16195679]]

        [[  837.98120117   475.99157715]]

        [[  837.98120117   475.99157715]]

        [[  837.98120117   475.99157715]]

        [[  835.80529785   509.09332275]]

        [[  786.49377441   518.54351807]]

        [[  838.94732666   518.62652588]]

        [[  190.358078     704.49346924]]

        [[  208.84550476    33.8540802 ]]

        [[  163.57302856    65.54798126]]

        [[  872.27624512   492.64758301]]

        [[ 1249.43811035   544.3850708 ]]

        [[  990.10369873   629.85559082]]
```

```
[[ 1049.3918457    630.64154053]]

[[ 1049.3918457    630.64154053]]

[[ 1106.41552734   631.43395996]]

[[ 1106.41552734   631.43395996]]

[[ 1234.04406738   634.27832031]]

[[  221.44618225    38.7995224 ]]

[[  164.21116638    50.73623657]]

[[  800.02148438   489.79278564]]

[[  800.02148438   489.79278564]]

[[ 1208.66601562   501.44384766]]

[[ 1208.66601562   501.44384766]]

[[ 1169.54492188   619.1159668 ]]

[[ 1032.4642334    630.11914062]]

[[ 1089.1171875    630.5402832 ]]

[[  993.20428467   655.9552002 ]]

[[  825.6776123    488.89175415]]

[[  825.6776123    488.89175415]]

[[ 1012.32873535   632.60369873]]

[[  131.75250244    68.50597382]]

[[  187.90661621    67.35686493]]

[[ 1068.53027344   634.29821777]]

[[ 1068.53027344   634.29821777]]

[[ 1127.21435547   637.89868164]]

[[ 1127.21435547   637.89868164]]

[[ 1235.71130371   607.07751465]]

[[ 1040.89868164   632.66876221]]
[[ 1040.89868164   632.66876221]]

[[ 1097.47314453   633.92510986]]

[[ 1157.41821289   636.00787354]]

[[ 1244.30834961   637.54974365]]

[[ 1167.45812988   664.23266602]]

[[ 1234.46264648   671.21270752]]

[[ 1124.21826172   598.73718262]]

[[ 1130.52233887   671.51184082]]

[[  810.0836792    501.42736816]]

[[  810.0836792    501.42736816]]

[[  811.26062012   501.1625061 ]]]
```