7

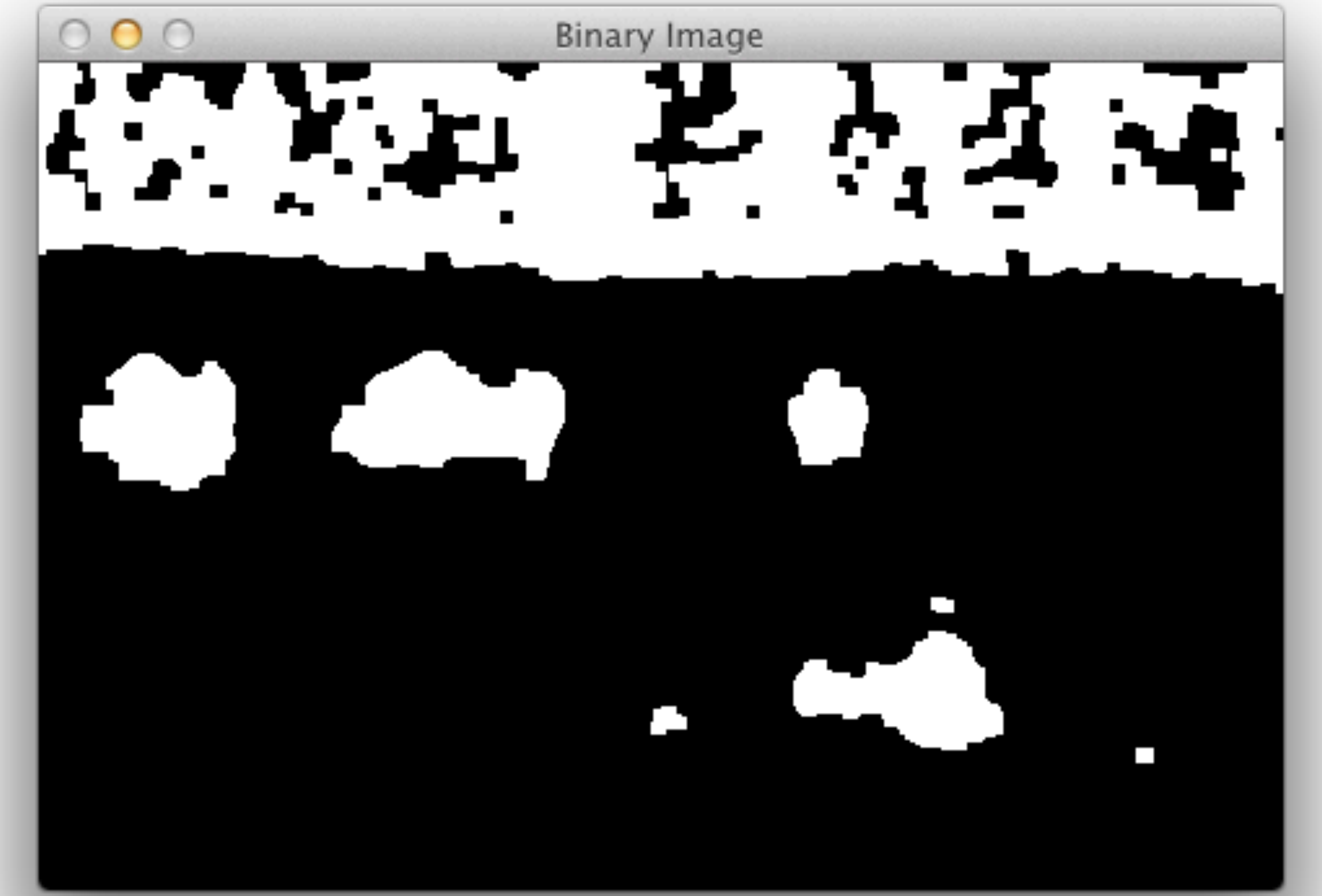
Extracting Lines, Contours, and Components

In this chapter, we will cover:

- Detecting image contours with the Canny operator
- Detecting lines in images with the Hough transform
- Fitting a line to a set of points
- Extracting the components' contours
- Computing components' shape descriptors

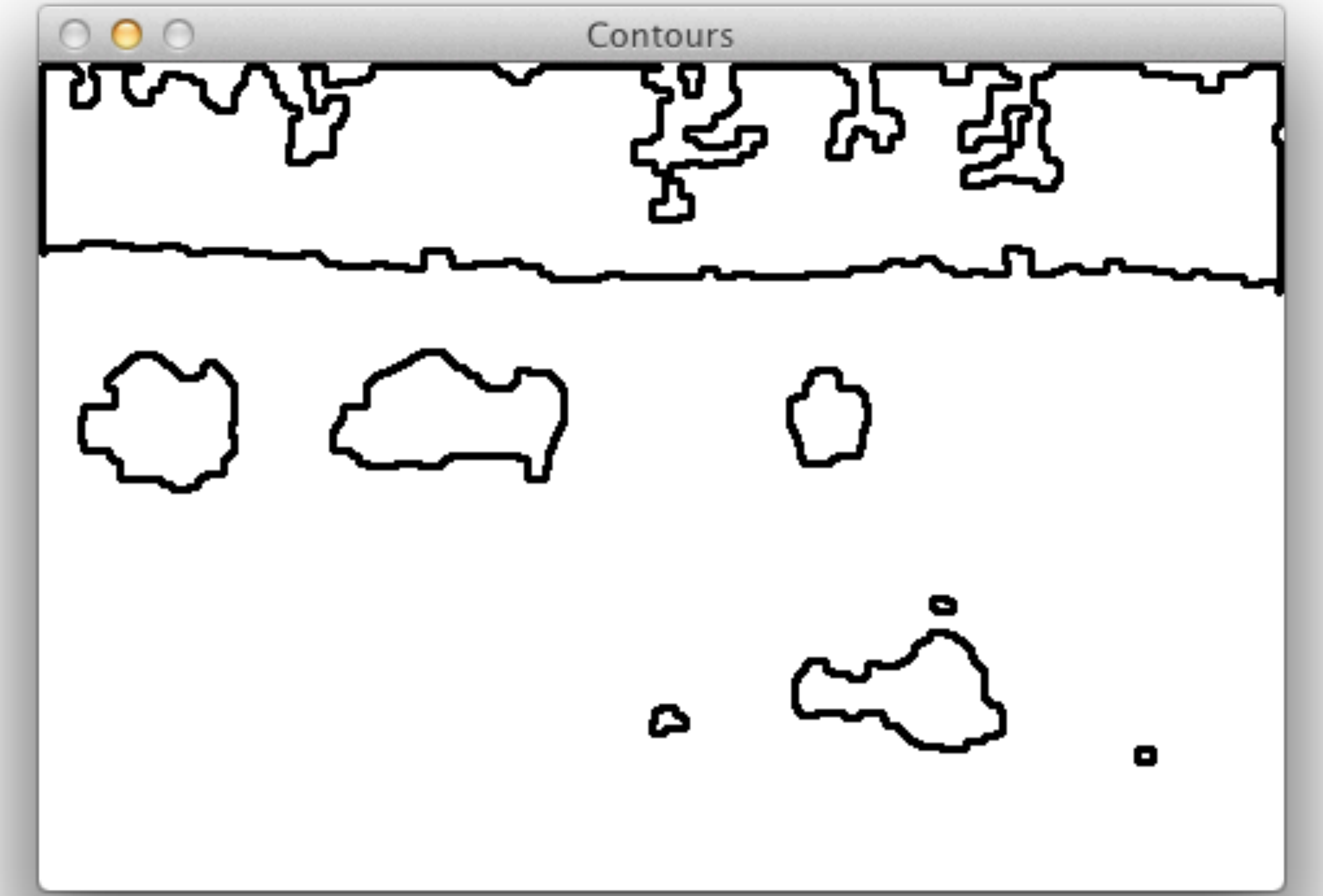
Extracting the components' contours

```
// blobs.cpp
//
int main()
{
   // Read input binary image
   cv::Mat image= cv::imread("../images/binaryGroup.bmp",0);
   if (!image.data)
       return 0;

   cv::namedWindow("Binary Image");
   cv::imshow("Binary Image",image);
```

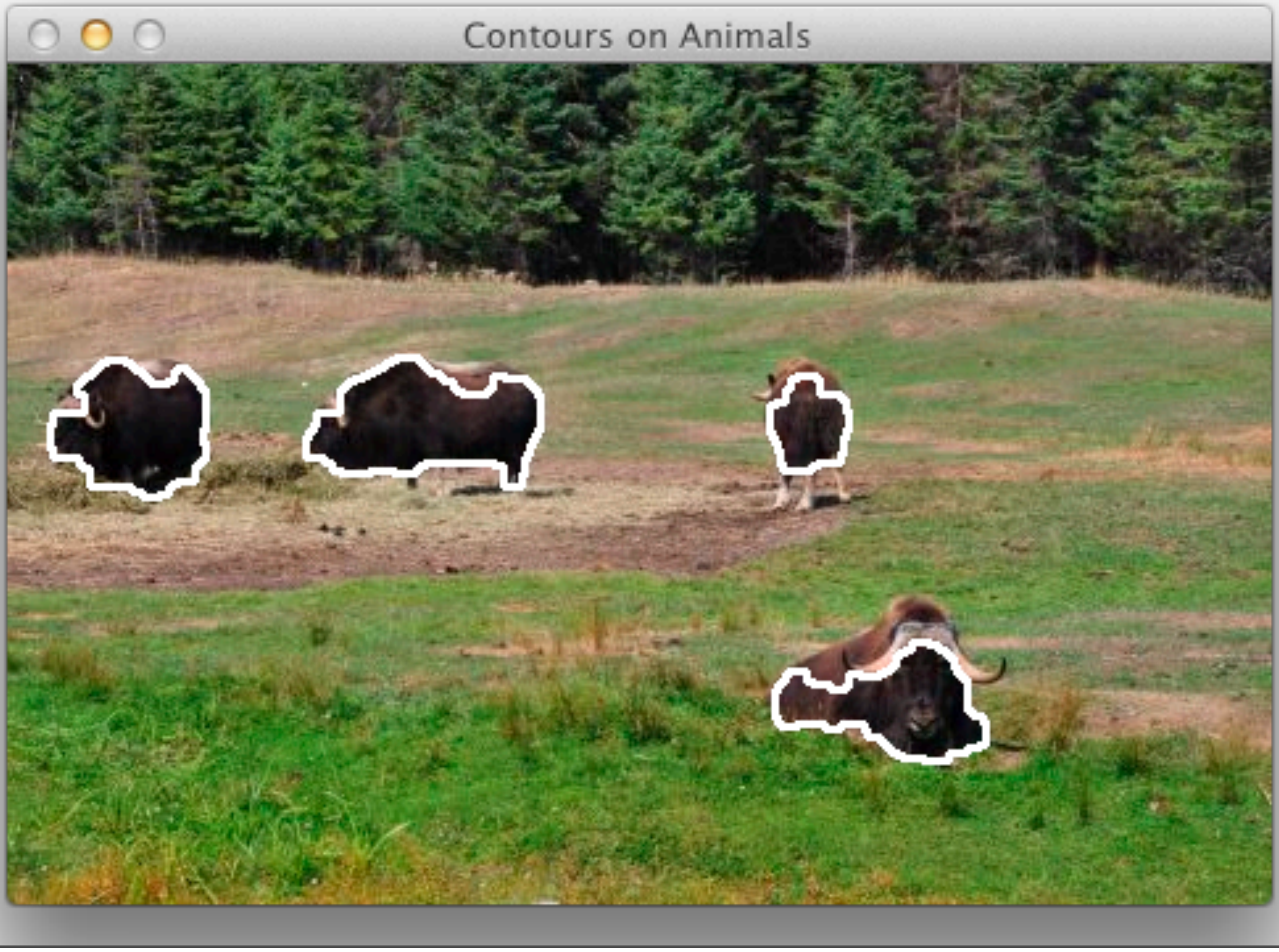


```
// Get the contours of the connected components
std::vector< std::vector<cv::Point> > contours;
cv::findContours(image,
                  contours, // a vector of contours
                  CV_RETR_EXTERNAL, // retrieve the external contours
                  CV_CHAIN_APPROX_NONE); // retrieve all pixels of each contours

// Print contours' length
std::cout << "Contours: " << contours.size() << std::endl;
std::vector< std::vector<cv::Point> >::iterator itContours= contours.begin();
for ( ; itContours!=contours.end(); ++itContours)
 {
    std::cout << "Size: " << itContours->size() << std::endl;
}

// draw black contours on white image
cv::Mat result(image.size(),CV_8U,cv::Scalar(255));
cv::drawContours(result,contours,
                  -1, // draw all contours
                  cv::Scalar(0), // in black
                  2); // with a thickness of 2

cv::namedWindow("Contours");
cv::imshow("Contours",result);
```

```
Contours: 9
Size: 22
Size: 41
Size: 220
Size: 24
Size: 111
Size: 197
Size: 245
Size: 36
Size: 1947
Polygon size: 12
```

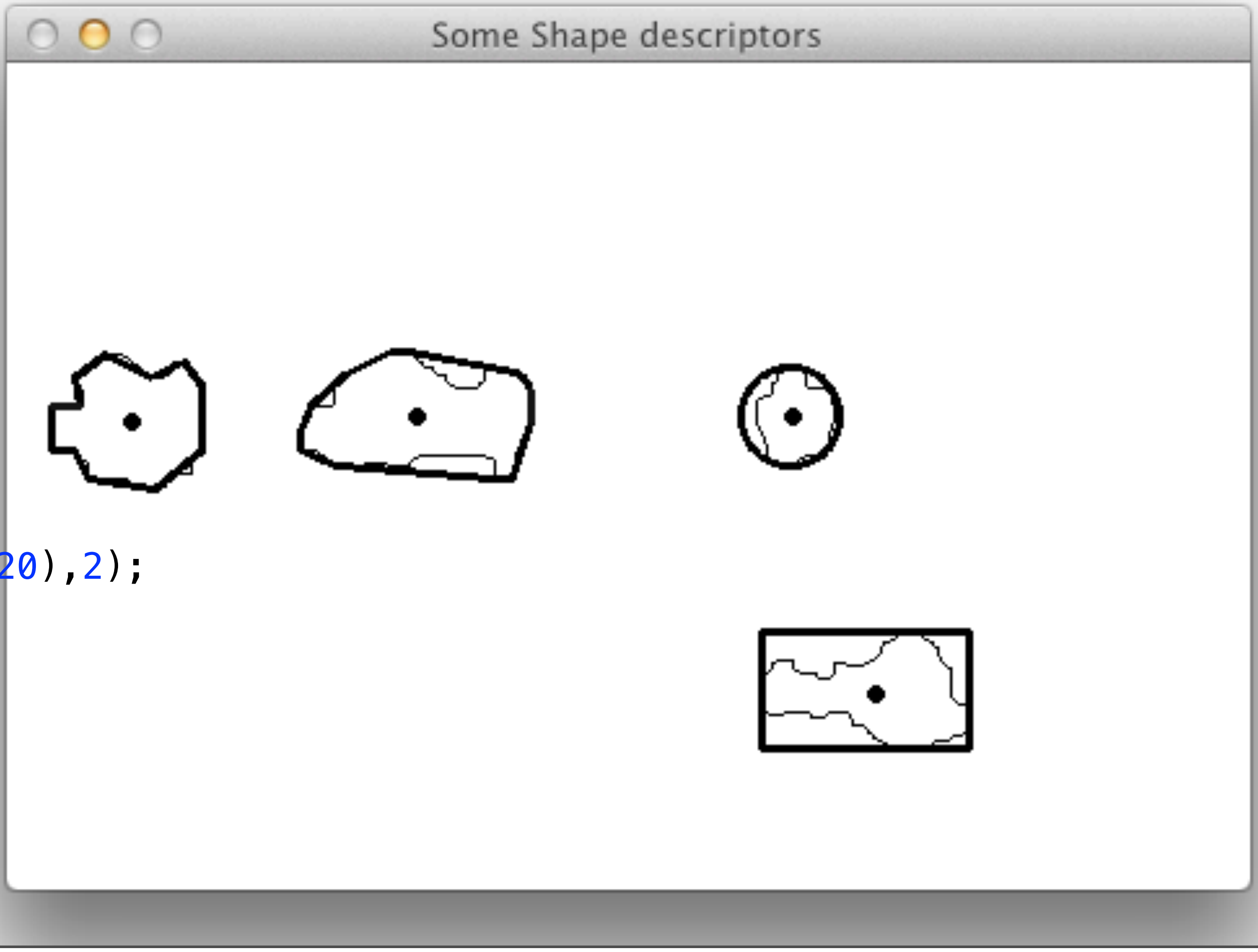


```
// Eliminate too short or too long contours
int cmin= 100;  // minimum contour length
int cmax= 1000; // maximum contour length
std::vector< std::vector<cv::Point> >::iterator itc= contours.begin();
while (itc!=contours.end()) {

    if (itc->size() < cmin || itc->size() > cmax)
        itc= contours.erase(itc);
    else
        ++itc;
}

// draw contours on the original image
cv::Mat original= cv::imread("../images/group.jpg");
cv::drawContours(original,contours,
                 -1, // draw all contours
                 cv::Scalar(255,255,255), // in white
                 2); // with a thickness of 2

cv::namedWindow("Contours on Animals");
cv::imshow("Contours on Animals",original);
```



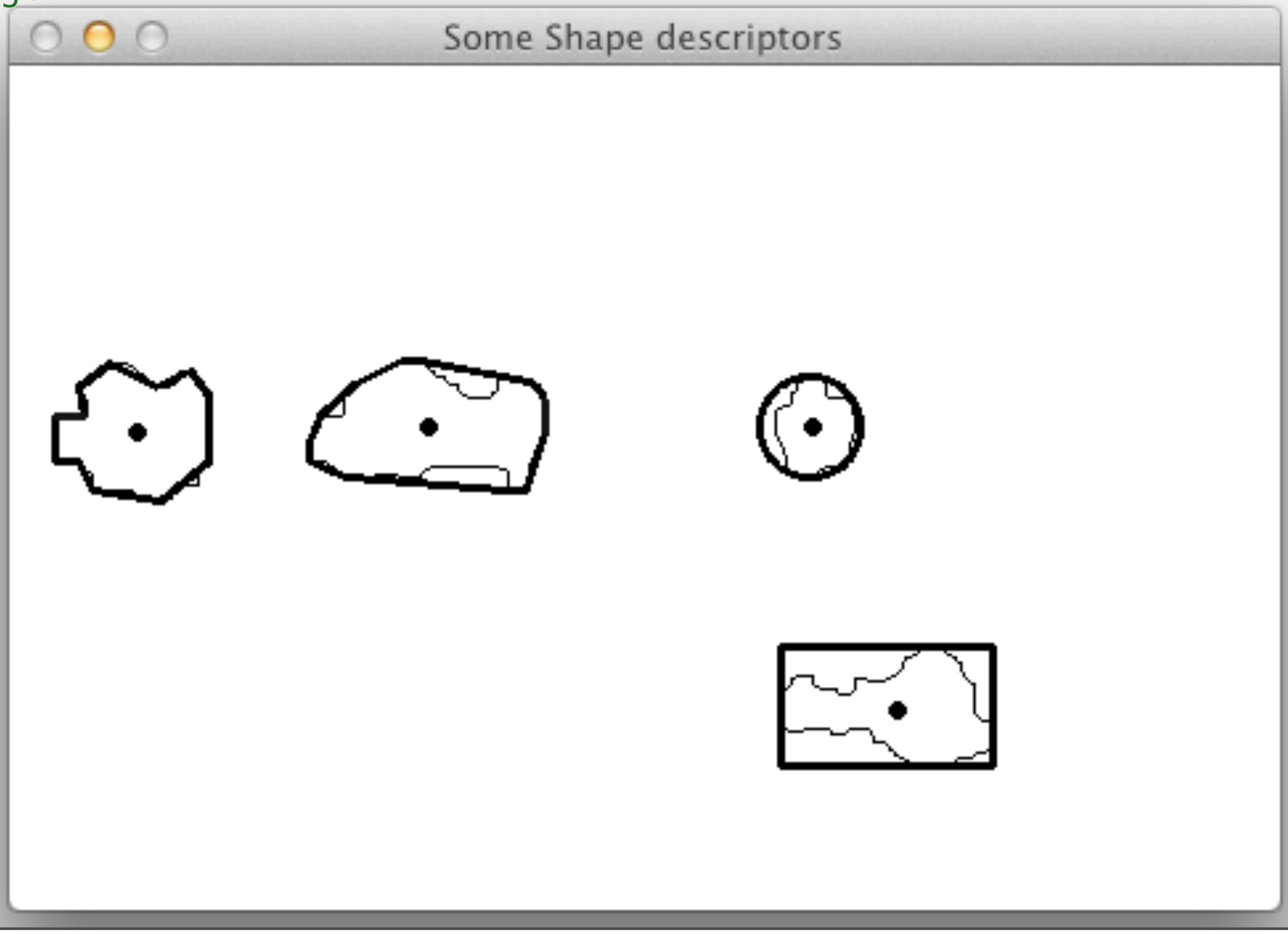
```
// Let's now draw black contours on white image
result.setTo(cv::Scalar(255));
cv::drawContours(result,contours,
                 -1, // draw all contours
                 cv::Scalar(0), // in black
                 1); // with a thickness of 1

// testing the bounding box
cv::Rect r0= cv::boundingRect(cv::Mat(contours[0]));
cv::rectangle(result,r0,cv::Scalar(0),2);

// testing the enclosing circle
float radius;
cv::Point2f center;
cv::minEnclosingCircle(cv::Mat(contours[1]),center,radius);
cv::circle(result,cv::Point(center.x,center.y),static_cast<int>(radius),cv::Scalar(0),2);

//   cv::RotatedRect rrect= cv::fitEllipse(cv::Mat(contours[1]));
//   cv::ellipse(result,rrect,cv::Scalar(0),2);

// testing the approximate polygon
std::vector<cv::Point> poly;
cv::approxPolyDP(cv::Mat(contours[2]),poly,5,true);

std::cout << "Polygon size: " << poly.size() << std::endl;

// Iterate over each segment and draw it
std::vector<cv::Point>::iterator itp= poly.begin();
while (itp!=(poly.end()-1)) {
     cv::line(result,*itp,*(itp+1),cv::Scalar(0),2);
     ++itp;
}
// last point linked to first point
cv::line(result,*(poly.begin()),*(poly.end()-1),cv::Scalar(20),2);
```

```
Polygon size: 12
```

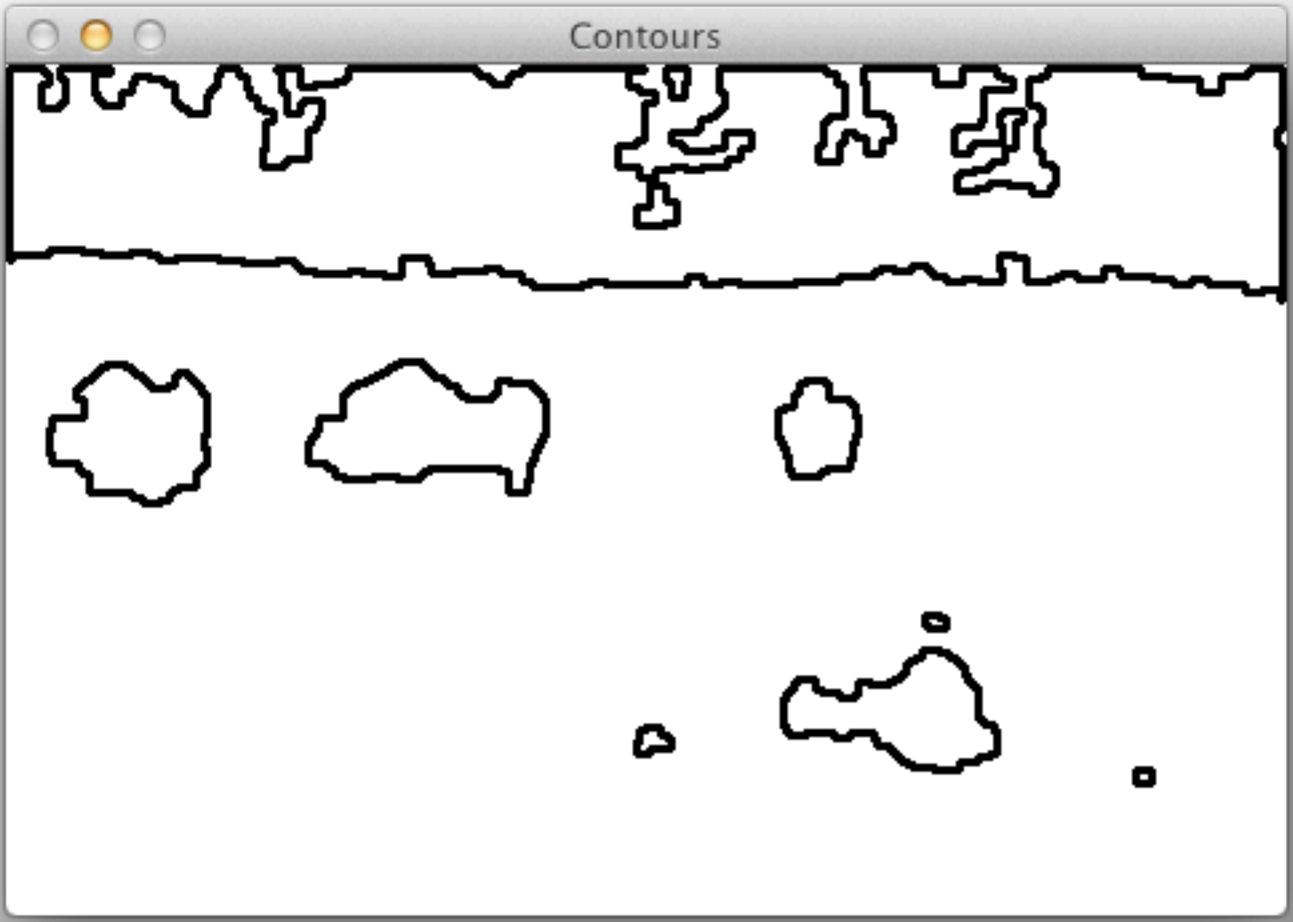


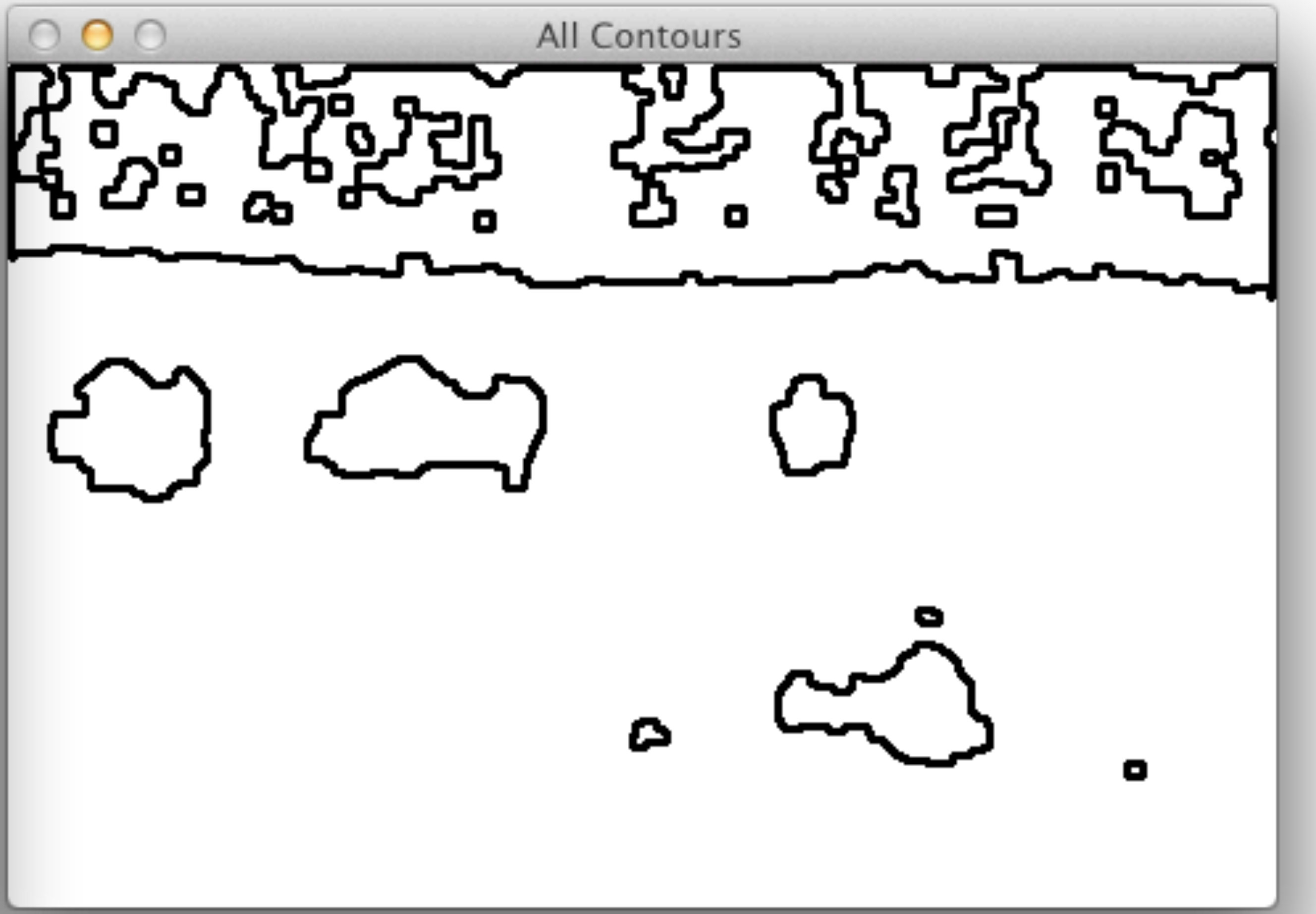
```
// testing the convex hull
std::vector<cv::Point> hull;
cv::convexHull(contours[3],hull);

// Iterate over each segment and draw it
std::vector<cv::Point>::iterator it= hull.begin();
while (it!=(hull.end()-1)) {
     cv::line(result,*it,*(it+1),cv::Scalar(0),2);
     ++it;
}
// last point linked to first point
cv::line(result,*(hull.begin()),*(hull.end()-1),cv::Scalar(20),2);

// testing the moments

// iterate over all contours
itc= contours.begin();
while (itc!=contours.end()) {

     // compute all moments
     cv::Moments mom= cv::moments(cv::Mat(*itc++));

     // draw mass center
     cv::circle(result,
               // position of mass center converted to integer
               cv::Point(mom.m10/mom.m00,mom.m01/mom.m00),
               2,cv::Scalar(0),2); // draw black dot
}

cv::namedWindow("Some Shape descriptors");
cv::imshow("Some Shape descriptors",result);
```

$$m_{ji} = \sum_{x,y} \left(\mathrm{array}(x,y) \cdot x^j \cdot y^i \right)$$



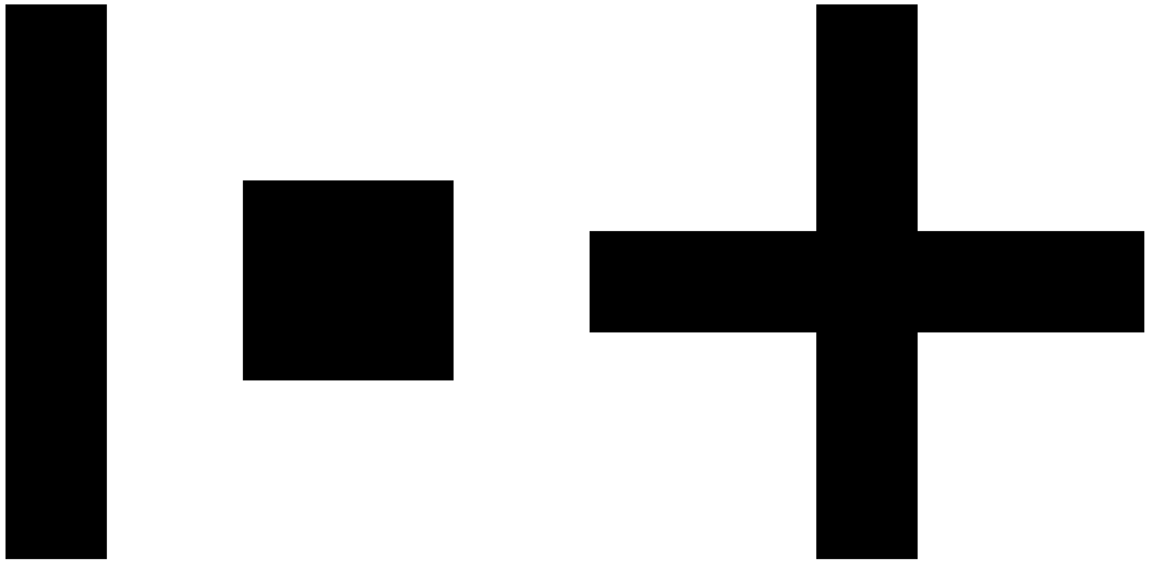
```
// New call to findContours but with CV_RETR_LIST flag
image= cv::imread("../images/binaryGroup.bmp",0);

// Get the contours of the connected components
cv::findContours(image,
                 contours, // a vector of contours
                 CV_RETR_LIST, // retrieve the external and internal contours
                 CV_CHAIN_APPROX_NONE); // retrieve all pixels of each contours

// draw black contours on white image
result.setTo(cv::Scalar(255));
cv::drawContours(result,contours,
                 -1, // draw all contours
                 cv::Scalar(0), // in black
                 2); // with a thickness of 2
cv::namedWindow("All Contours");
cv::imshow("All Contours",result);
```





2D Object Recognition

Object Modeling



- Objects are black (black colored)
- B/G is always white (A4 paper)
- A photo of each object is taken for modeling.

- Given an image of some of the objects, the task is to recognize the object.

2D Object Recognition

Input Image