1 Assignment 2 (100 points)

In class, we discussed how image gradients are used the measure the change in the values of the image. The intuition of basic edge detection is that object boundaries lead to large changes in the image values, implying that objects will be surrounded by edges and edges will enclose objects. In this assignment, you will build on ideas from Lab 2 in order to explore the extent to which this is true, and become familiar with some strategies for improving results when the real world is more messy than our algorithms assume. Edge detection and segmentation remains an active area of research in Computer Vision.

To submit your work, put answers to the written questions in a file(s) at the top level directory. Clearly label all of your work. For the math-oriented portion of the assignment, you may typeset in LaTeX or Word. If you don't know how to correctly typeset math equations, write your solution very neatly, with good labels, on a piece of paper and take a clear, high resolution photograph of the paper. Work that is not legible will not be graded. Clean your Visual Studio Solution, removing all temporary files, then zip up your code, output files, and written answers. Name the file `CS585_Assignment2_username.zip`. Submit the code with web-submit.

1.1 Learning Objectives

1. Become familiar with different strategies for performing edge detection
2. Explore practical issues with edge detection
3. Learn about expressing linear operations on images as convolutions
4. Become comfortable with calculating image gradients

1.2 Technical Task (30 points)

The first task in your assignment is to implement a rudimentary edge detector by filling in the function `findEdgesGradientMagnitude`. You will use the function from the lab to calculate the gradient magnitude, and then you will create a separate edge mask by thresholding. You should use OpenCV functions to do this as much as possible, instead of doing it one element at a time. Watch your types!

The skeleton code for the assignment already has event handling set up so that when you click on a point in the original image, it will call a function to attempt to fill in the region bounded by edges. You must fill in the body of the function `fillRegionBoundedByEdges` by using the OpenCV `floodFill` algorithm. (If you are so inclined, you can implement the recursive version of this function that we described in class and to see what happens when you try to fill large regions.)

The event handling is already in place to let you control the edge-finding interactively by smoothing the image before calculating the gradients and controlling the threshold using track bars on the window with the original image. Answer the questions below about the behavior of this edge detector on different types of images under different conditions.

The Canny edge detector is the most widely used edge detector you will encounter. (It was John Canny's Master's thesis at MIT.) It has many steps, including smoothing the image, calculating gradients, thresholding the gradients, refining the edges by looking for regions where the gradient magnitude is maximal, and then linking edge segments. It is implemented in OpenCV with the `Canny()` function, and you can find a tutorial about how to use it here: `http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html`. The skeleton code provided will allow you to switch between the gradient magnitude thresholding algorithm and the Canny algorithm by pressing "g" or "c". Fill in the body of the function `findEdgesCanny` to allow you to see how the results are different.

As you work with these edge detectors, you may find that it is difficult to achieve the results you expect. Morphological operators are a widely used post-processing step, and can be a big help in many practical applications. An OpenCV tutorial on how to use Morphological operators is here: `http://docs.opencv.org/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html`. Morphological operators are also described in Szeliski ch. 3.3.2, and Shapiro and Stockman 3.5. In your program, when you press "e" for "erode" and "d" for dilate, the program will call a function to attempt to perform a morphological operation on your edge image. You must fill in the function `refineEdgesMorphological` to perform these operations by calling the OpenCV functions.

You can find more information about gradients in Szeliski section 3.2.3 (more general information about image filtering in section 3.2.2). Edges are discussed in section 4.2.1. You will also find that the OpenCV tutorials that are linked from inside the comments of the code are also good references.

To summarize:

1. Implement edge detection by thresholding gradient magnitude `findEdgesGradientMagnitude`

2. Fill in the function to fill an image region bounded by edges `fillRegionBoundedByEdges`

3. Use the Canny edge detection `findEdgesCanny`

4. Work with morphological operators to perform post-processing `refineEdgesMorphological`

5. Answer the questions below to explore the behavior of the edge detectors

6. Test that your program mechanically works by working with the image "yellowEgg.jpg"

1.3 Programming Assignment Questions (28 points)

Your program will save the edge image and the region-filled image when you press "s". It will create a unique counter for each image, but the counter will reset each time you start the program, so be sure to rename your files as you work.

1. (4 pts) Run your program on the image yellowEgg.jpg both by thresholding the gradient magnitude and by using the Canny edge detector. Choose a threshold and smoothing level that leads to a nice contour and filled region of the egg. Save your edge and region-filled images as "SoloYellowEgg-Gradient-edges_[smoothing]_[threshold].png", "SoloYellowEgg-Gradient-result_[smoothing]_[threshold].png," and "SoloYellowEgg-Canny-edges_[smoothing]_[threshold].png" and "SoloYellowEgg-Canny-result_[smoothing]_[threshold].png."

2. (4 pts) Run your program on the image "eggsAndCube.jpg." Choose a smoothing level that you feel gives good contours in the gradient magnitude, and then for each of the red, green, blue, and yellow eggs, choose the highest threshold that gives a clean result. Name your files "redEgg-edges_[smoothing]_[threshold].png", "redEgg-result_[smoothing]_[threshold].png" (one for each color egg).

3. (4 pts) Are you able to find any parameters that wil allow you to identify the cube (or a face of the cube)? Choose your best try and name your file "cube-edges_[smoothing]_[threshold].png" and "cube-result_[smoothing]_[threshold].png"

4. (4 pts) Use your program on the image "camoEggsOnPaper.jpg." If you knew that you were looking for textured objects on a plain background, what steps could you combine in order to highlight an entire egg, even though there are lots of internal edges? What type of binary image analysis steps could you perform in order to segment the entire egg?

5. (4 pts) Run your program on the image rainbowSpheres.jpg. You may find that some areas of the image are easier to segment than others. Try clicking on a few image regions (spots or stripes on the balls). Choose your favorite good result and name it "rainbowEasy-edges_[smoothing]_[threshold].png" and "rainbowEasy-result_[smoothing]_[threshold].png"

6. (4 pts) Identify an area of the image that you think should be reasonably easy to pick out, but that the program is having trouble with. (For example, the purple and blue stripes on top of the rainbow ball). What is happening that causes the entire background to be filled when you attempt to fill the region bounded by edges starting in one of these areas?

7. (4 pts) Use morphological operators to manipulate your edge map to help close gaps where the region-filling is leaking out so that you can fill in a difficult region. Create before and after images of your troublesome regions

named "rainbowHard-before-edges_[smoothing]_[threshold].png" and "rainbowHard-before-result_[smoothing]_[threshold].png" and "rainbowHard-after-edges_[smoothing]_[threshold].png" and "rainbowHard-after-result_[smoothing]_[threshold].png"

1.4 Lab Follow-up Questions (39 points)

1. Under the folder for Lab2, there is a folder called "gradients" containing several image where I have prepared several images using the formula $I(x, y) = x * \cos(\theta) + y \sin(\theta)$. Remember that the gradient of a function (written as $\nabla I = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$) is just a vector containing the partial derivatives of the function with respect to each variable. You take a partial derivative of a function using the normal rules of calculus, but you treat all variables other than the one you are interested in as constants.

   (a) (2 pts) Write an expression for the gradient of I, given the formula that I used to generate the images.

   (b) (2 pts) For each of three values of $\theta = (0°, 45°, 90°)$, calculate what you anticipate the values of the gradient will be

   (c) (3 pts) Using the program from the Lab, inspect the gradient of the images named "gradient_theta=0.png", "gradient_theta=45.png", "gradient_theta=90.png." Did you find any discrepancies?

   (d) (3 pts) Using the program from the Lab, inspect the gradient of the image named "gradient_mysteryTheta.png." Report the values that you find and use them to determine the angle I used to generate the image.

   (e) (3 pts) The image "gradient_mysteryThetaNoisy.png" has been corrupted with Gaussian noise similar to what you might encounter with a sensor in a camera. Inspect the calculated values of the gradient in order to estimate the value of theta that I used. (Hint: it is not the same as "gradient_mysteryTheta.png") Choose at least three locations and report the calculated values of θ.

   (f) (3 pts) Use the smoothing slider on the program from the lab to smooth the image before calculating the gradient. Choose three locations and report the calculated values of θ. Did it help?

   (g) (3 pts) A mathematical expression for a smoothed step edge is a sigmoid function. (`http://en.wikipedia.org/wiki/Sigmoid_function`). The image "gradient_sigmoid.png" has been generated with such a function (given below). By inspecting the gradient of the image at the center point, estimate the angle that I used to generate the image.

$$I(x, y) = \frac{1}{1 + e^{-(\cos(\theta)(x - x_c) + \sin(\theta)(y - y_c))}}$$

   (where $(x_c, y_c) = (128, 128)$), the center of the image).

2. (4 pts) Write down the kernel corresponding to the logic in the following loop (assume that we can use double array indexes into the image):

```
Image source;
Image result;
for (int r = 1; r < image.rows-1; r++)
  for (int c=1;  c < image.cols-1; c++)
    result = source[r-1][c-1] + 2*source[r-1][c] + source[r-1][c+1]
          + 2*source[r][c-1] + 4*source[r][c]     + 2*source[r][c]
          + source[r+1][c-1] + 2*source[r+1][c] + source[r+1][c+1];
```

3. (8 pts) (In lieu of implementing convolution) Perform the following convolution by hand. You may notice that the kernel doesn't fit in some portions of the image. What design decision do you need to make in order to obtain a result at element (1,2) (or any other boundary pixel)? (`http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/copyMakeBorder/copyMakeBorder.html`)

Source:

	1	2	3	4	5	6	7
1	9	10	1	8	2	7	8
2	1	10	4	9	7	3	2
3	9	2	9	7	0	10	5
4	6	10	8	8	3	0	4
5	1	10	10	7	0	4	6
6	3	5	7	4	1	4	7
7	5	8	0	7	8	8	8

Kernel:

1	2	1
1	-4	1
1	2	1

4. (8 pts) (In lieu of implementing morphological operators) Given the following image and structuring element, perform the erosion and dilation by hand

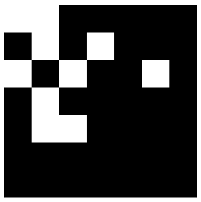
Source:

	1	2	3	4	5	6	7
1	1	1	0	0	0	0	0
2	0	1	0	1	0	0	0
3	1	0	1	0	0	1	0
4	0	1	0	0	0	0	0
5	0	1	1	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

Structuring Element:

0	1	0
1	1	1
0	1	0

The source matrix, visualized as an image looks like this:



2 Lecture preparation (3 points)

2.1 Preparation for Lecture 4

In Lecture 4, we will discuss face detection with the Viola and Jones face detector. This technique is popular because they use a neat computational trick, the ”integral image” to compute the response of the image to box filters very fast.

Please take a look at these two articles before class on Tuesday:

- `http://en.wikipedia.org/wiki/Viola-Jones_object_detection_framework`
- `http://en.wikipedia.org/wiki/Summed_area_table`

2.2 Preparation for Lecture 5 (3 pts)

In Lecture 5, we will discuss image pyramids for performing multi-scale feature detection.

Under the folder for Assignment 2, there is a folder called ”camoEggsPyramid.” Use your program from section 1.2 to perform edge detection on each of the images in the folder. What do you notice about the way the edge-finding and region-filling works on each of the images in the pyramid?