谭树杰 11849060

Lab3 Report

1. Image translation

```
// xt, yt is the translation pixels along horizontal and vertical axis respectively,
// positive value means down and right direction respectively
cv::Mat Translation(cv::Mat srcImage, int xt, int yt){
    CV_Assert(srcImage.data != NULL);
    cv::Mat dstImage(srcImage.size(), srcImage.type(), cv::Scalar(0));
    for (int i = 0; i < srcImage.rows; i += 1){
        for (int j = 0; j < srcImage.cols; j += 1){
            int x = i - xt;
            int y = j - yt;
            if (x >= 0 && y >= 0 && x < srcImage.rows && y < srcImage.cols){
                dstImage.ptr<cv::Vec3b>(i)[j] = srcImage.ptr<cv::Vec3b>(x)[y];
            }
        }
    }
    return  dstImage;
}
```

lena.jpg，goldhill.jpg 经 Translation 处理后图像（向右平移 70，向下 50，左为原图，右为处理后图像，下同）




2. Image rotation

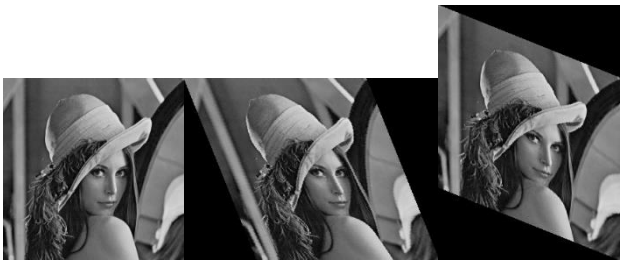
```
cv::Mat Rotate(cv::Mat srcImage, int angle)
{
    float alpha = angle * CV_PI / 180;
    float cx = srcImage.cols / 2.0;
    float cy = srcImage.rows / 2.0;
    float ru_x = srcImage.cols * cos(alpha);        // 右上角
    float ru_y = srcImage.cols * (-sin(alpha));
    float ld_x = srcImage.rows * sin(alpha);        // 左下角
    float ld_y = srcImage.rows * cos(alpha);
    float rd_x = ru_x + ld_x;          // 右下角
    float rd_y = ru_y + ld_y;
    float xMin = min(min(min(0.0f, ru_x), ld_x), rd_x);
    float xMax = max(max(max(0.0f, ru_x), ld_x), rd_x);
    float yMin = min(min(min(0.0f, ru_y), ld_y), rd_y);
    float yMax = max(max(max(0.0f, ru_y), ld_y), rd_y);
    int cols = xMax - xMin;
    int rows = yMax - yMin;
    float dx = cols / 2;
    float dy = rows / 2;
    cv::Mat dstImage(rows, cols, srcImage.type(), cv::Scalar(0));

    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols; ++j)
        {
            int y = (i - dy) * cos(alpha) - (j - dx) * sin(alpha) + cy;
            int x = (i - dy) * sin(alpha) + (j - dx) * cos(alpha) + cx;
            //cout << "x = " << x << "   y = " << y << endl;
            if (x >= 0 && y >= 0 && x < srcImage.cols && y < srcImage.rows){
                //cout << i << "   " << j << "  ";
                dstImage.at<cv::Vec3b>(i, j) = srcImage.at<cv::Vec3b>(y, x);
            }
        }
    }
    return dstImage;
}
```

lena.jpg, goldhill.jpg 顺时钟旋转 30 度后图像

3. Shear operations (vertical and horizontal) respectively

```
// axis = 0 for horizontal shear, 1 for vertical shear
// ratio is width displacement to height if axis = 0.
// and is height displacement to width if axis = 1.
cv::Mat Shear(cv::Mat srcImage, int axis, float ratio){
    CV_Assert(srcImage.data != NULL);
    if (axis == 0) {
        cv::Mat dstImage(srcImage.rows, int(srcImage.cols + srcImage.rows * ratio),
srcImage.type(), cv::Scalar(0));
        for (int i = 0; i < dstImage.rows; i += 1){
            for (int j = 0; j < dstImage.cols; j += 1){
                int x = i;
                int y = j - i * ratio;
                if (x >= 0 && y >= 0 && x < srcImage.rows && y < srcImage.cols){
                    dstImage.ptr<cv::Vec3b>(i)[j] = srcImage.ptr<cv::Vec3b>(x)[y];
                }
            }
        }
        return  dstImage;
    }
    else {
        cv::Mat dstImage(int(srcImage.rows + srcImage.cols * ratio), srcImage.cols,
srcImage.type(), cv::Scalar(0));
        for (int i = 0; i < dstImage.rows; i += 1){
            for (int j = 0; j < dstImage.cols; j += 1){
                int y = j;
                int x = i - y * ratio;
                if (x >= 0 && y >= 0 && x < srcImage.rows && y < srcImage.cols){
                    dstImage.ptr<cv::Vec3b>(i)[j] = srcImage.ptr<cv::Vec3b>(x)[y];
                }
            }
        }
        return  dstImage;
    }
}
```

lena.jpg, goldhill.jpg 经水平剪切和竖直错切后图像



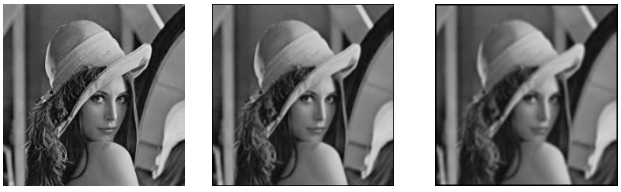
goldhill.jpg 经水平剪切和竖直错切后图像

4. Smoothing with 3x3, 5x5 windows using averaging and median and binarization filters respectively

（1）averaging filter

```
// the filter size would be (2 * size + 1) by (2 * size + 1)
cv::Mat averageFileter(cv::Mat srcImage, int size){
    CV_Assert(srcImage.data != NULL);
    cv::Mat dstImage(srcImage.size(), srcImage.type(), cv::Scalar(0));
    int tmp;
    for (int i = size; i < srcImage.rows - size; i += 1){
        for (int j = size; j < srcImage.cols - size; j += 1){
            tmp = 0;
            for (int k = i - size; k <= i + size; k++){
                for (int m = j - size; m <= j + size; m++){
                    tmp += int(srcImage.at<uchar>(k, m));
                }
            }
            dstImage.ptr<uchar>(i)[j] = cv::saturate_cast<uchar>( tmp / (2 * size + 1)
/ (2 * size + 1) );
        }
    }
    return  dstImage;
}
```

lena.jpg、goldhill.jpg 经 average filter 处理后图像（3x3, 5x5 windows）





（2）median filter

```
cv::Mat medianFilter(cv::Mat srcImage, int size){
    CV_Assert(srcImage.data != NULL);
    cv::Mat dstImage(srcImage.size(), srcImage.type(), cv::Scalar(0));
    for (int i = size; i < srcImage.rows - size; i++){
        for (int j = size; j < srcImage.cols - size; j++){
                vector<uchar> tmp;
                for (int k = i - size; k <= i + size; k++){
                    for (int m = j - size; m <= j + size; m++){
                        tmp.push_back( int(srcImage.at<uchar>(k,m)) );
                    }
                }
                sort(tmp.begin(), tmp.end());
                dstImage.ptr<uchar>(i)[j] =cv::saturate_cast<uchar>( tmp[(2 * size +
1)*(2 * size + 1) / 2] );
        }
    }
    return  dstImage;
}
```

lena.jpg、goldhill.jpg 经 median filter 处理后图像（3x3, 5x5 windows）





（3）binarization filter

```
cv::Mat binarizationFileter(cv::Mat srcImage, int thresh){
    CV_Assert(srcImage.data != NULL);
    cv::Mat dstImage(srcImage.size(), srcImage.type(), cv::Scalar(0));
    for (int i = 1; i < srcImage.rows - 1; i += 1){
        for (int j = 1; j < srcImage.cols - 1; j += 1){
            if (srcImage.at<uchar>(i, j) > thresh){
                dstImage.at<uchar>(i, j) = 255;
            }
            else {
                    dstImage.at<uchar>(i, j) = 0;
            }
        }
    }
    return dstImage;
}
```

lena.jpg、goldhill.jpg 经 binarization filter 处理后图像（threshold = 100）




5. Sharpen images with Laplacian and Sobel operators respectively through masking process

(1) Laplacian

```
cv::Mat laplace(cv::Mat srcImage){
    CV_Assert(srcImage.data != NULL);
    cv::Mat dstImage(srcImage.size(), srcImage.type(), cv::Scalar(0));
    cv::Mat lapMat = (cv::Mat_<float>(3, 3) << 0, 1, 0,
        1, -4, 1,
        0, 1, 0);
    float g;
    for (int i = 1; i < srcImage.rows - 1; i += 1){
        for (int j = 1; j < srcImage.cols - 1; j += 1){
            g  = 0;
            for (int r = -1; r < 1; r++){
                for (int c = -1; c <= 1; c ++){
                    g += float(srcImage.ptr<uchar>(i + r)[j + c]) *
lapMat.ptr<float>(r+1)[c+1];
                }
            }
            dstImage.at<uchar>(i,j) = cv::saturate_cast<uchar>(g);
        }
    }
    dstImage.convertTo(dstImage, CV_8UC1);
    return  dstImage;
}
```

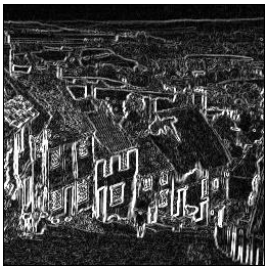
lena.jpg、goldhill.jpg 经 Laplacian operator 处理后图像

(2) Sobel

```
cv::Mat soble(cv::Mat srcImage){
    CV_Assert(srcImage.data != NULL);
    cv::Mat dstImage(srcImage.size(), srcImage.type(), cv::Scalar(0));
    cv::Mat gxMat = (cv::Mat_<float>(3, 3) << -1, 0, 1,
                                              -2, 0, 2,
                                              -1, 0, 1);
    cv::Mat gyMat = (cv::Mat_<float>(3, 3) << -1, -2, 1,
        -0, 0, 0,
        -1, 2, 1);
    float gx, gy;
    for (int i = 1; i < srcImage.rows - 1; i += 1){
        for (int j = 1; j < srcImage.cols - 1; j += 1){
            gx = gy = 0;
            for (int r = -1; r <= 1; r++){
                for (int c = -1; c <= 1; c ++){
                    gx += srcImage.ptr<uchar>(i + r)[j + c] *
gxMat.ptr<float>(r+1)[c+1];
                    gy += srcImage.ptr<uchar>(i + r)[j + c] *
gyMat.ptr<float>(r+1)[c+1];
                }
            }
            dstImage.ptr<uchar>(i)[j] =cv::saturate_cast<uchar>( sqrt(gx * gx + gy *
gy) );
        }
    }
    dstImage.convertTo(dstImage, CV_8UC1);
    return  dstImage;
}
```

lena.jpg、goldhill.jpg 经 Sobel operator 处理后图像



6. Gamma correction using gamma value 0.1, 0.4, 0.6, 0.8 and compute the variances of the resulted images

```
cv::Mat gamma(cv::Mat srcImage, double g){
    CV_Assert(srcImage.data != NULL);
    cv::Mat dstImage(srcImage.size(), srcImage.type(), cv::Scalar(0));
    for (int i = 0; i < srcImage.rows; i += 1){
        for (int j = 0; j < srcImage.cols; j += 1){
                dstImage.at<uchar>(i, j) =
cv::saturate_cast<uchar>( pow(float(srcImage.at<uchar>(i, j)) / 255.0, g) * 255.0 );
        }
    }
    double variance = 0;
    for (int i = 0; i < srcImage.rows; i++){
        for (int j = 0; j < srcImage.cols; j++){
            variance += float(dstImage.at<uchar>(i, j) );
        }
    }
    variance /= (dstImage.rows * dstImage.cols );
    cout << "variance is " << variance << endl;
    return dstImage;
}
```

lena.jpg、goldhill.jpg 经 gamma correction 处理后图像，gamma value 分别为 0.1,0.4,0.6,0.8; lena.jpg 处理后的方差分别为 228.013, 167.276, 138.573, 116.36，goldhill.jpg 处理后的方差分别为 232.773, 179.35, 152.192, 130.267

7.Write histogram enhancement function to perform global and local image enhancement

```
cv::Mat hist(cv::Mat srcImage){
  CV_Assert(srcImage.data != NULL);
  cv::Mat dstImage(srcImage.size(), srcImage.type(), cv::Scalar(0));
  int histImage[256] = { 0 };
  for (int i = 0; i < srcImage.rows; i += 1){
       for (int j = 0; j < srcImage.cols; j += 1){
            histImage[srcImage.at<uchar>(i, j)] ++;
       }
  }
  int cumulate[256] = { 0 };
  cumulate[0] = histImage[0];
  for (int i = 1; i < 256; i++){
       cumulate[i] += cumulate[i - 1] + histImage[i];
  }
  int s[256] = { 0 };
  for (int i = 0; i < 256; i++){
       s[i] = cvFloor(255.0 * cumulate[i] / double(srcImage.rows) /
srcImage.cols);
  }
  for (int i = 0; i < srcImage.rows; i += 1){
       for (int j = 0; j < srcImage.cols; j += 1){
            dstImage.at<uchar>(i, j) = s[ srcImage.at<uchar>(i, j) ];
       }
  }
  return dstImage;
}
```

lena.jpg、goldhill.jpg 经直方图均值化处理后图像