機器視覺 HW5 資工三 109590041 范遠皓

環境

Visual Studio 2019、C++ 、openCV2.4.13.6

image Edge Detection

轉灰階-> Gaussian 過濾->邊緣偵測

灰階、 Gaussian 過濾方式同之前作業。

```
370     // 讀取圖片
371     Mat sourceImage = imread(imageInfo.Path());
372
373     // 灰階、Gaussian 過濾
374     Mat grayImage = library.ConvertToGray(sourceImage);
375     Mat filterImage = library.FilterBy(grayImage, ImageLibrary::FilterType::Gaussian, 3);
376
377     // 邊緣偵測
378     map<ImageLibrary::EdgeType, Mat> sobelResult = library.Sobel(filterImage, imageInfo._sobelThreshold);
379     map<ImageLibrary::EdgeType, Mat> prewittResult = library.Prewitt(filterImage, imageInfo._prewittThreshold);
380     Mat laplacianResult1 = library.Laplacian(filterImage, laplacianKernel1, imageInfo._laplacian1Threshold);
381     Mat laplacianResult2 = library.Laplacian(filterImage, laplacianKernel2, imageInfo._laplacian2Threshold);
```

不同類型的 Threshold (Sobel、 Prewitt、 Laplacian1、 Laplacian2)

```
354     // 圖片檔名設定
355     vector<ImageInfo> images;
356     images.push_back(ImageInfo(IMAGE_FOLDER + "House512.png", 32, 32, 10, 12));
357     images.push_back(ImageInfo(IMAGE_FOLDER + "Lena.png", 40, 40, 10, 12));
358     images.push_back(ImageInfo(IMAGE_FOLDER + "Mandril1.png", 45, 45, 12, 15));
359
360     ImageLibrary library = ImageLibrary();
361
362     // 定義 Laplacian Kernel
363     const Kernel<int> laplacianKernel1 = { {0, 1, 0}, {1, -4, 1}, {0, 1, 0} };
364     const Kernel<int> laplacianKernel2 = { {1, 1, 1}, {1, -8, 1}, {1, 1, 1} };
365
```

使用 2 個 Kernel 進行捲積計算得出 gx、gy

經公式 G = |gx| + |gy| 得到最終邊緣強度。

```cpp
// 進行邊緣梯度計算
map<EdgeType, Mat> DetectEdgeBy2Kernel(const Mat& sourceImage, const Kernel<int>& kernelX, const Kernel<int>& kernelY, uchar threshold = 128) {
    map<EdgeType, Mat> resultMap;
    Mat& verticalImage = resultMap[EdgeType::Vertical] = Mat(sourceImage.size(), CV_8UC3);
    Mat& horizonImage = resultMap[EdgeType::Horizon] = Mat(sourceImage.size(), CV_8UC3);
    Mat& bothImage = resultMap[EdgeType::Both] = Mat(sourceImage.size(), CV_8UC3);

    // padding
    Mat padded = this->PadByZero(sourceImage, kernelX.size() / 2);

    // 計算 gx, gy, G = |gx| + |gy|
    int max = 0;
    Kernel<int> tempG(sourceImage.rows, vector<int>(sourceImage.cols));
    for (int i = 0; i < sourceImage.rows; i++) {
        for (int j = 0; j < sourceImage.cols; j++) {
            int gx = 0;
            for (int m = 0; m < kernelX.size(); m++)
                for (int n = 0; n < kernelX.size(); n++)
                    gx += padded.at<Vec3b>(i + m, j + n)[0] * kernelX[m][n];
            verticalImage.at<Vec3b>(i, j) = Vec3b((gx >> 16) & 255, (gx >> 8) & 255, gx & 255);

            int gy = 0;
            for (int m = 0; m < kernelX.size(); m++)
                for (int n = 0; n < kernelX.size(); n++)
                    gy += padded.at<Vec3b>(i + m, j + n)[0] * kernelY[m][n];
            horizonImage.at<Vec3b>(i, j) = Vec3b((gy >> 16) & 255, (gy >> 8) & 255, gy & 255);

            int G = abs(gx) + abs(gy);
            tempG[i][j] = G;
            max = max < G ? G : max;
        }
    }

```

對邊緣強度進行正規化，並將結果二值化。

```cpp
        // 正規化、二值化
        for (int i = 0; i < sourceImage.rows; i++)
            for (int j = 0; j < sourceImage.cols; j++)
            {
                int value = tempG[i][j] * 255 / max > threshold ? 255 : 0;
                bothImage.at<Vec3b>(i, j) = Vec3b(value, value, value);
            }
        return resultMap;
    }

```

Sobel、 Prewitt 只差在 Kernel 不同。

Laplacian 每次只有一個 Kernel 另一個帶全零的 Kernel 使其不影響結果。

```
223     // Sobel Edge Detect
224     map<EdgeType, Mat> Sobel(const Mat& sourceImage, uchar threshold = 128) {
225         // 定義 Sobel Kernel
226         const Kernel<int> sobelKernelX = { {-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1} };
227         const Kernel<int> sobelKernelY = { {-1, -2, -1}, {0, 0, 0}, {1, 2, 1} };
228         return DetectEdgeBy2Kernel(sourceImage, sobelKernelX, sobelKernelY, threshold);
229     }
230
231     // Prewitt Edge Detect
232     map<EdgeType, Mat> Prewitt(const Mat& sourceImage, uchar threshold = 128) {
233         // 定義 Prewitt Kernel
234         const Kernel<int> prewittKernelX = { {-1, 0, 1}, {-1, 0, 1}, {-1, 0, 1} };
235         const Kernel<int> prewittKernelY = { {-1, -1, -1}, {0, 0, 0}, {1, 1, 1} };
236         return DetectEdgeBy2Kernel(sourceImage, prewittKernelX, prewittKernelY, threshold);
237     }
238
239     // Laplacian Edge Detect
240     Mat Laplacian(const Mat& sourceImage, const Kernel<int>& kernel, uchar threshold = 128) {
241         const Kernel<int> zeroKernelX = { {0, 0, 0}, {0, 0, 0}, {0, 0, 0} };
242         return DetectEdgeBy2Kernel(sourceImage, kernel, zeroKernelX, threshold)[EdgeType::Both];
243     }
244
```

• Discuss horizontal edge、vertical edge and both edge differences.

horizontal edge : 使用水平方向的捲積 Kernel，表示圖像中從左到右變化明顯的區域。

vertical edge: 使用垂直方向的捲積 Kernel，表示圖像中從上到下變化明顯的區域。

both edge: 同時檢測水平和垂直邊緣，可以獲得更全面的圖像描述和更準確的邊緣檢測結果。

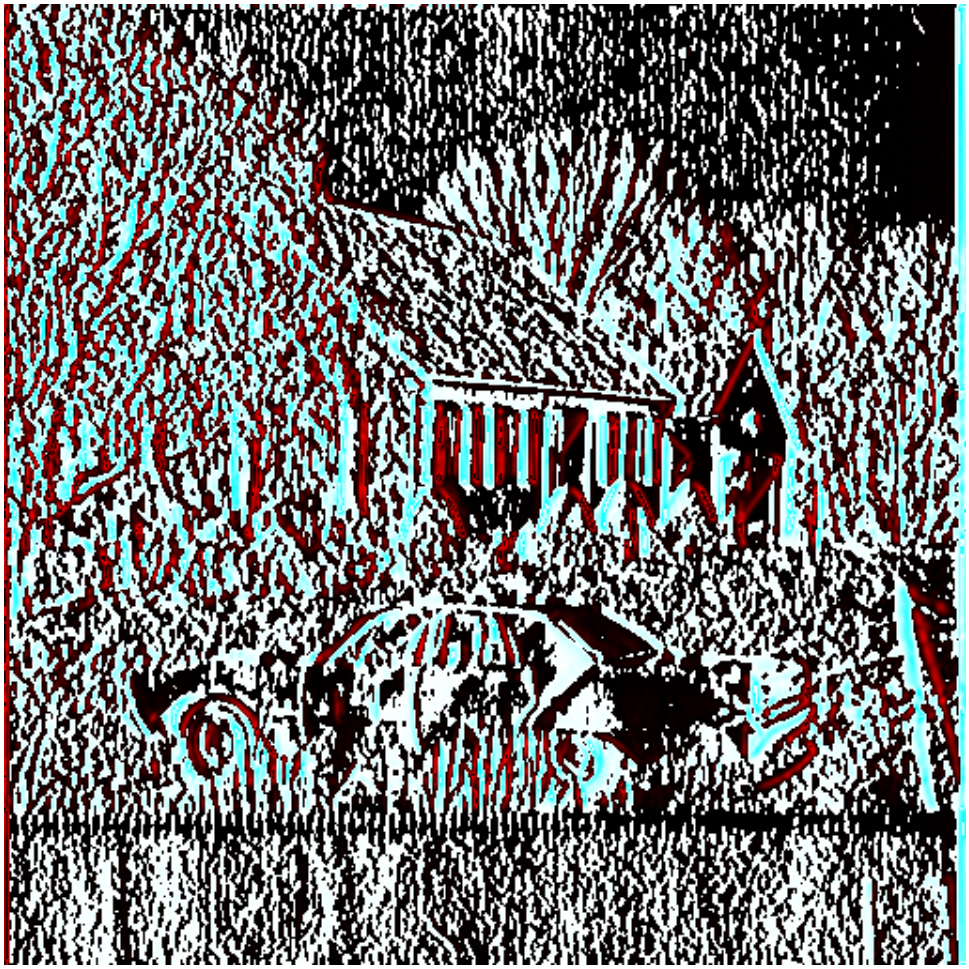
• Discuss Laplacian kernels differences.

Kernel2 相比於 Kernel1 具有更強的邊緣響應和更好的細節捕捉能力。然而由於較大的權重，雜訊的影響也較大。

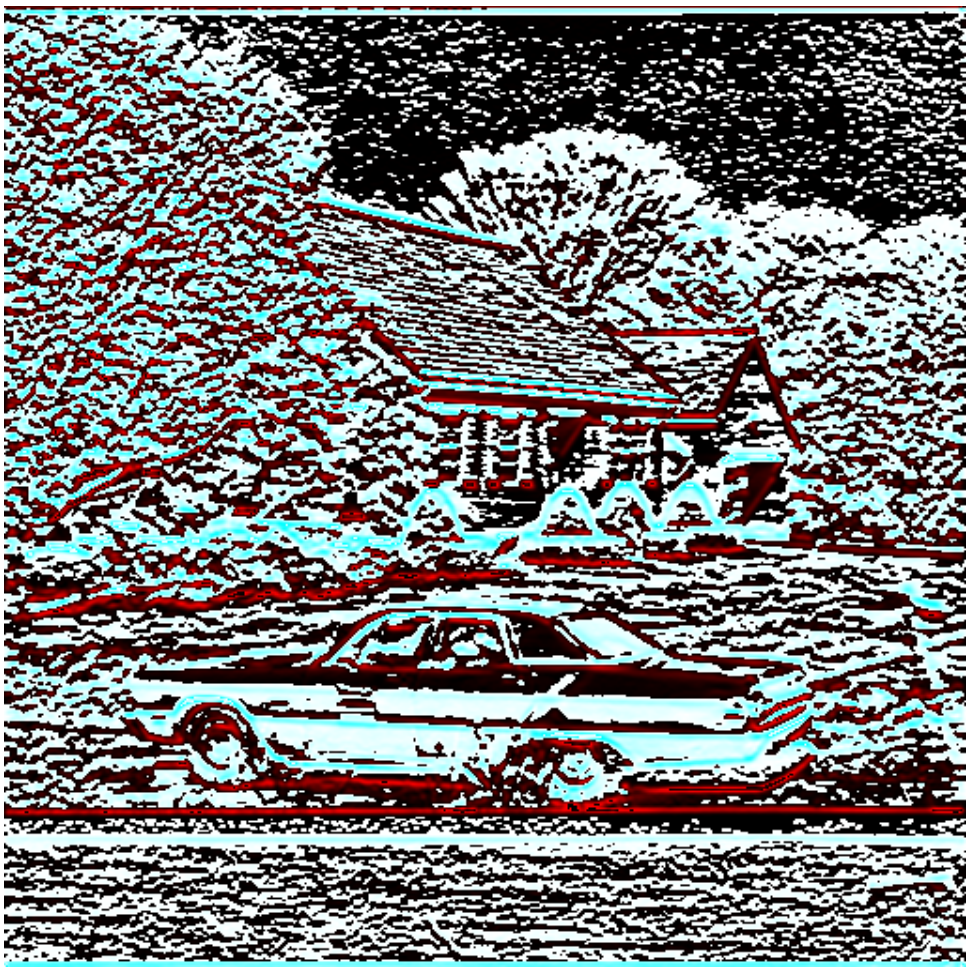
Result images

House512

Sobel Vertical

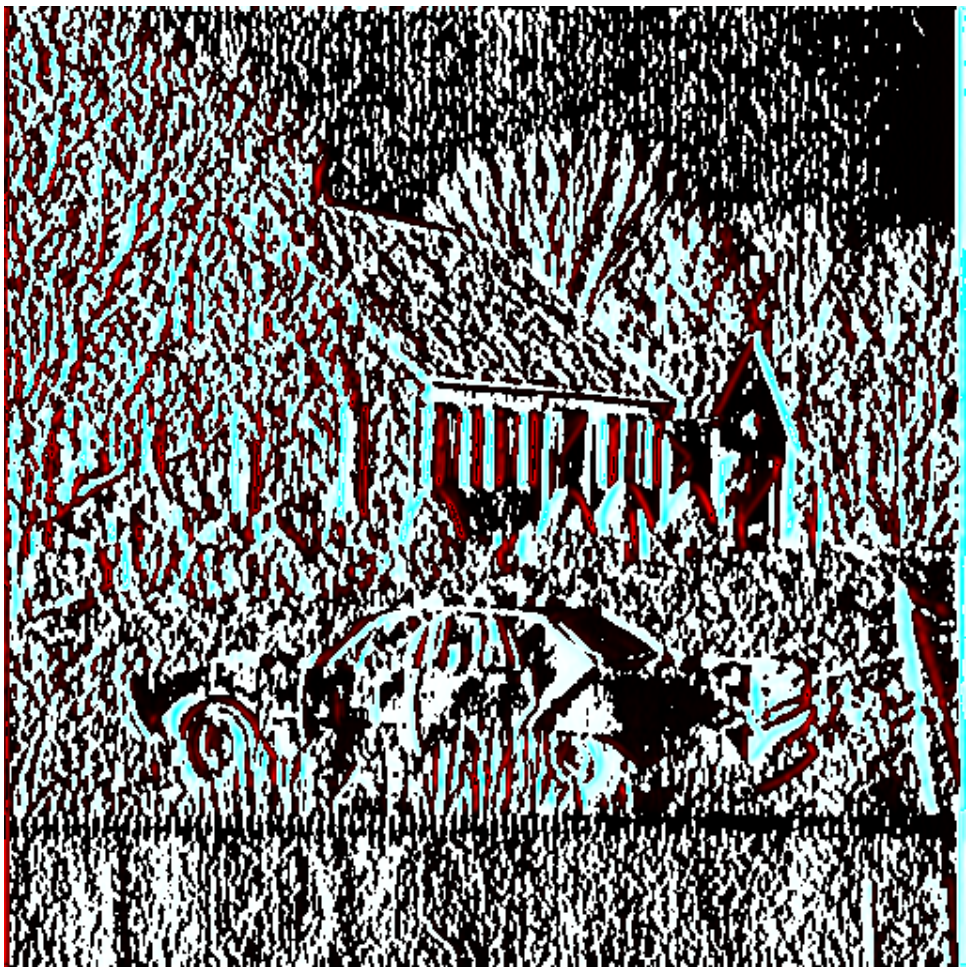


Sobel Horizon

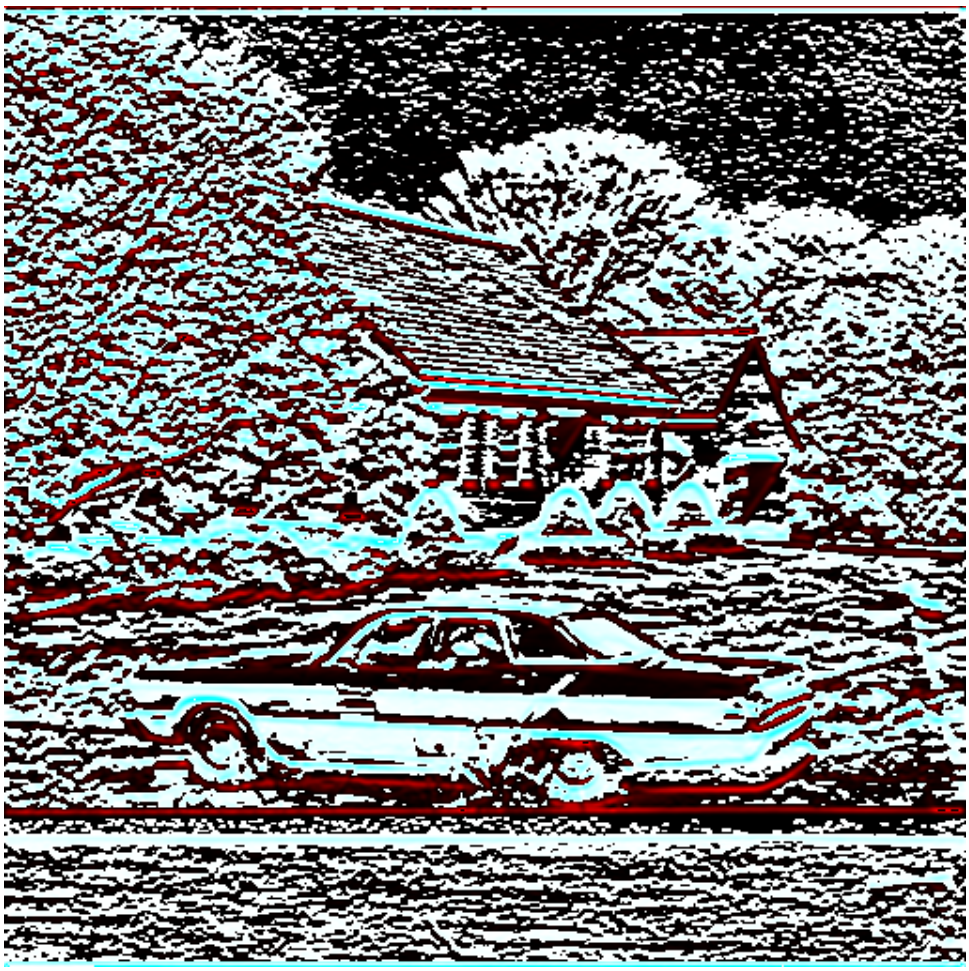


Sobel Both.

Prewitt Vertical



Prewitt Horizon



Prewitt Both.

Laplacian Kernel1

Laplacian Kernel2

Lena

Sobel Vertical

Sobel Horizon

Sobel Both

Prewitt Vertical

Prewitt Horizon

Prewitt Both

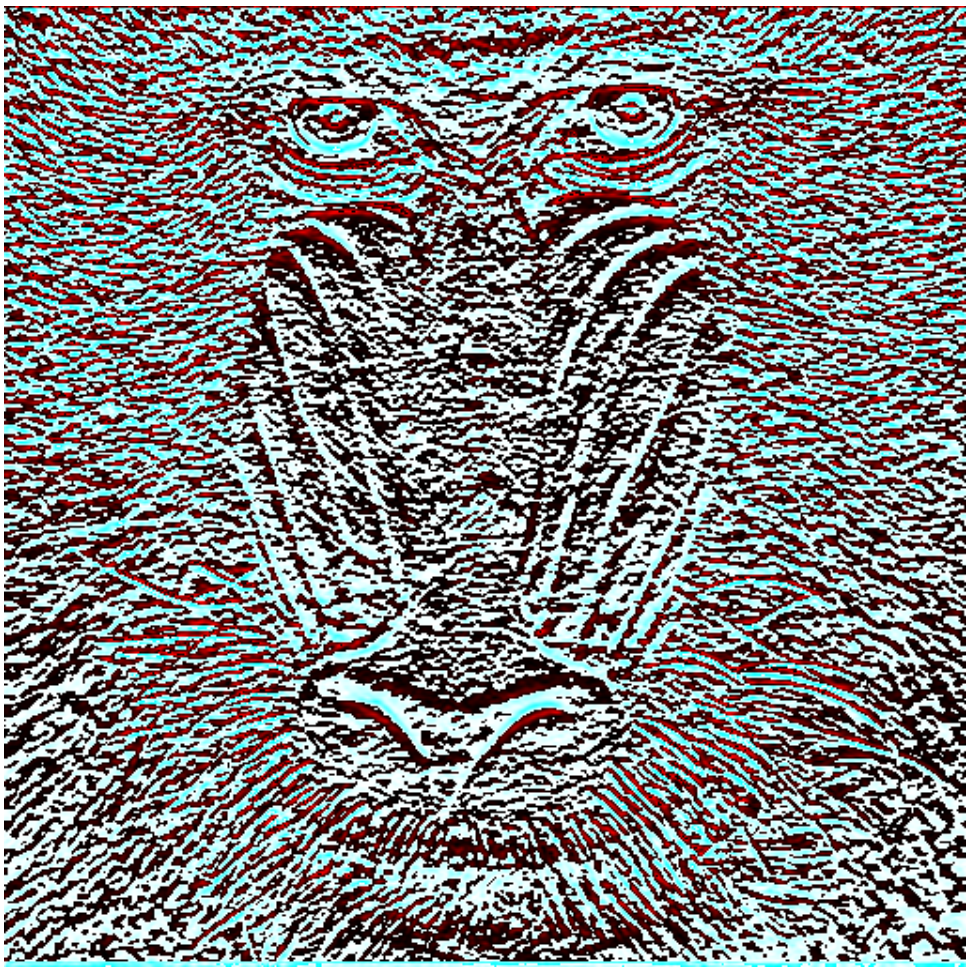
Laplacian1

Laplacian2

Mandrill

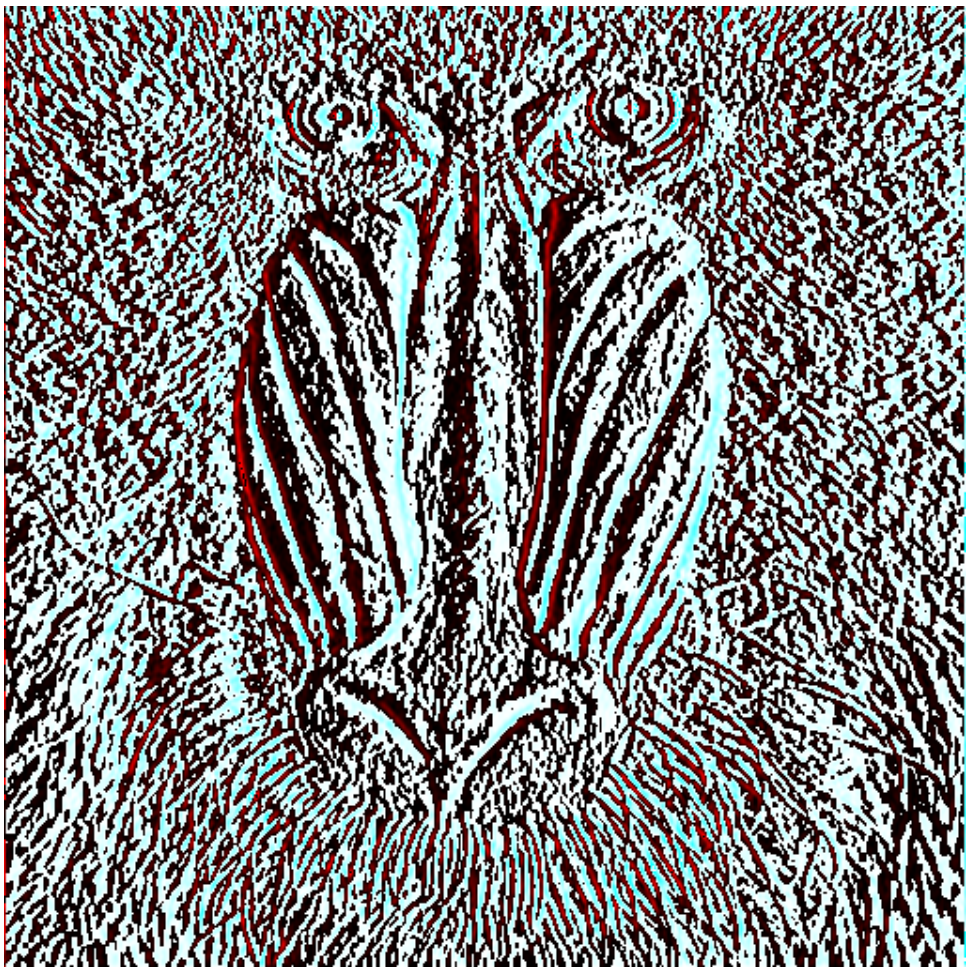
Sobel Vertical

Sobel Horizon



Sobel Both

Prewitt Vertical



Prewitt Horizon

Prewitt Both

Laplacian1

Laplacian2