影像處理報告

1. 實驗用圖片

影像來源皆為網路上下載，512x512 像素。一張是英文字母 A 的圖片，另一張是包含河流的風景圖片。

2. 演算法與程式碼

旋轉部分演算法：

以原點為中心，順時針轉 30 度

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

反推，從新圖片找原圖的像素(逆時針轉 30 度)

$$x = x'\cos\theta + y'\sin\theta$$
$$y = -x'\sin\theta + y'\cos\theta$$

為了以圖片中心旋轉，須找到圖片中心點

$$\mathrm{ox} = \mathrm{width} \div 2$$
$$\mathrm{oy} = \mathrm{height} \div 2$$

因此，先將新圖像素平移以方便計算，旋轉後再平移回原位

$$\mathrm{nx} = x' - \mathrm{ox}$$
$$\mathrm{ny} = y' - \mathrm{oy}$$
$$x = \mathrm{n}x'\cos\theta + ny'\sin\theta + ox$$
$$y = -nx'\sin\theta + ny'\cos\theta + \mathrm{oy}$$

為了顯示所有旋轉後的像素點，必須放大旋轉後的圖片

新的圖片大小

$$\mathrm{nw} = \mathrm{w}\cos\theta + h\sin\theta$$
$$\mathrm{nh} = \mathrm{w}\sin\theta + h\cos\theta$$

因為我們的圖片都是正方形，因此可以簡化成

$$\text{nedge} = \text{edge}(\sin\theta + \cos\theta)$$

算放大後的圖片像素點位移

$$\text{ox} = -\text{nedge}\cos\theta - \text{nedge}\sin\theta + \text{width} \div 2$$
$$\text{oy} = \text{nedge}\sin\theta - \text{nedge}\cos\theta + \text{height} \div 2$$

最終帶回原式，旋轉後平移

$$x = \text{n}x'\cos\theta + ny'\sin\theta + ox$$
$$y = -nx'\sin\theta + ny'\cos\theta + \text{oy}$$

插值演算法：

Nearest-neighbor interpolation

其演算法為找最接近的點，因此推回的點四捨五入後即可找到

```
for y in range(length):
    for x in range(length):
        row = round(-x*csin + y*ccos + oy)
        col = round(x*ccos + y*csin + ox)
        if row >=0 and row < height and col >= 0 and col < width:
            tmp[y][x] = img[row][col]

return tmp
```

Bilinear interpolation

其演算法為附近的點加權計算，依照點的遠近進行加權

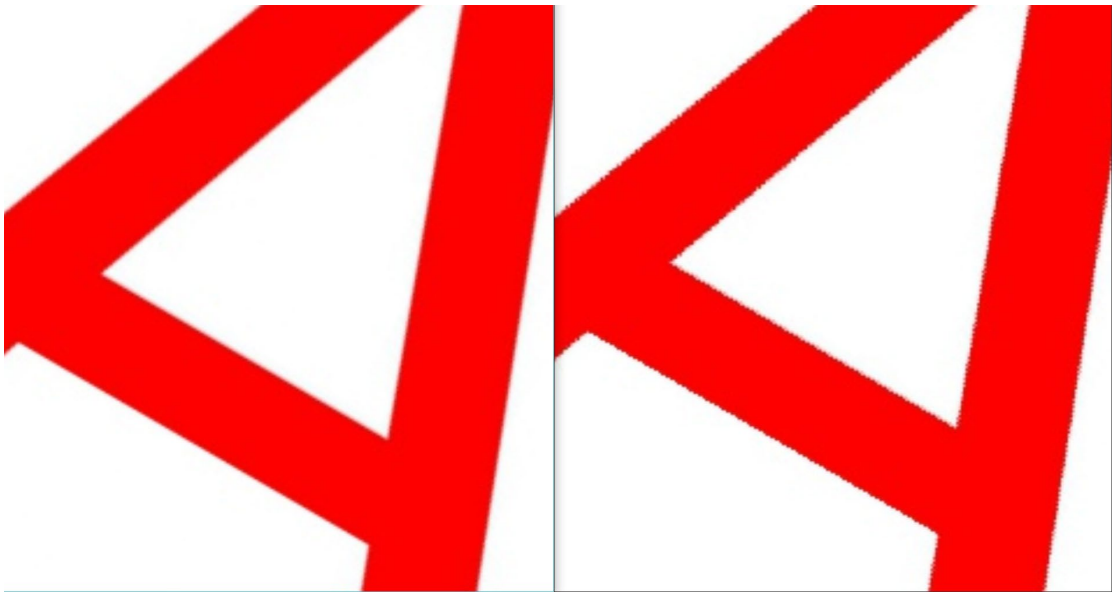
```
for y in range(length):
    for x in range(length):
        srcy = -x*csin + y*ccos + oy
        srcx = x*ccos + y*csin + ox
        if srcy >=0 and srcy < height and srcx >= 0 and srcx < width:
            u = srcx - int(srcx)
            v = srcy - int(srcy)
            tmp[y][x] = ((1 - u) * (1 - v) * img[yvalue(srcy, height)][xvalue(srcx, width)] +
                    u * (1 - v) * img[yvalue(srcy, height)][xvalue(srcx + 1, width)] +
                    (1 - u) * v * img[yvalue(srcy + 1, height)][xvalue(srcx, width)] +
                    u * v * img[yvalue(srcy + 1, height)][xvalue(srcx + 1, width)])

return tmp
```

3. 實驗結果與討論

文字比較

Bilinear interpolation 左 右 Nearest-neighbor interpolation



風景比較

Bilinear interpolation 左 右 Nearest-neighbor interpolation

從實驗結果的圖像來看，nni 的鋸齒比起 bi 來更為明顯。

但 bi 在圖像清晰度上，有些部分會較為模糊，例如：顏色較多元的部分，因為參照了附近所有的像素點來混合運算。

再速度方面很明顯感受的到 nni 的運算速度是比 bi 快上不少的，因此需要快速處理時 nni 是較佳的插值方式。