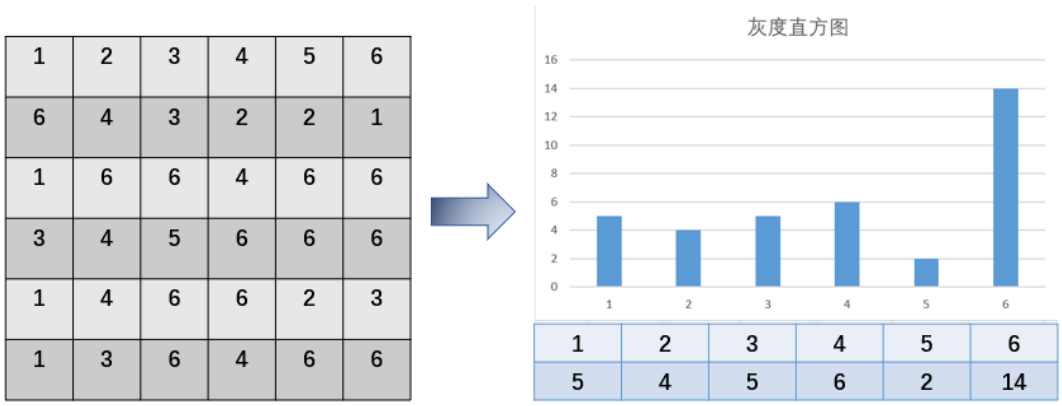
题目: 数字图像处理-灰度直方图

一、概述

灰度直方图是图像处理中用于统计图像中每个灰度级像素数量的图形表示方法。在数字图像中，每个像素点通常用一个灰度值来表示，这个值反映了该像素点的亮度。灰度值的范围通常是从 0 到 255。灰度直方图的横坐标表示灰度级，纵坐标表示该灰度级的像素数量。通过直方图，我们可以直观地了解图像的亮度分布情况，比如图像是偏暗还是偏亮，以及图像中不同亮度区域的分布情况。



1	2	3	4	5	6
6	4	3	2	2	1
1	6	6	4	6	6
3	4	5	6	6	6
1	4	6	6	2	3
1	3	6	4	6	6

1	2	3	4	5	6
5	4	5	6	2	14

图 1 灰度直方图绘制过程示意图

对于一张已经经过预处理的图像而言，其中每一个像素经过处理可以成为一个值，所有像素的组合可以视为一个二维矩阵，统计二维矩阵的每一个值的出现次数，统计为表，就可以通过表绘制出灰度直方图。

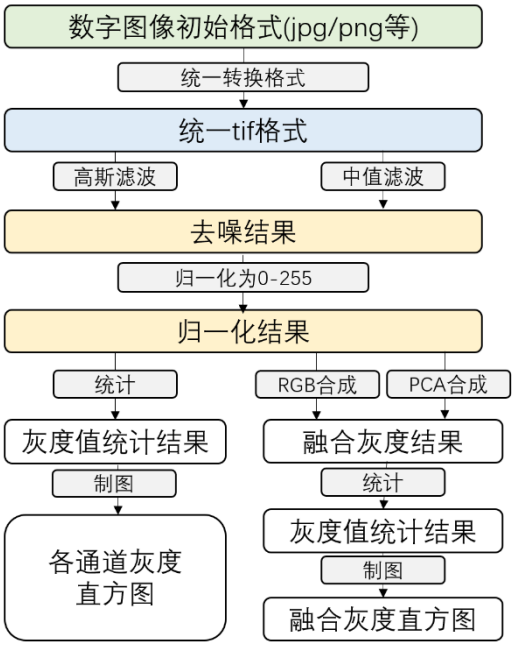


图 2 作业流程图

在本次作业中，我们通过一定的流程来绘制灰度直方图。首先，我们选取了四张初始图片作为最开始的数据输入。这四张图片的格式不一，有的是 jpg，有的是 png，因此，我们首先统一转换格式，全部使用 ArcGIS pro 工具转为 tif 格式再输出。随后，我们分别采用高斯滤波和中值滤波进行处理，并同时以无滤波处理作为对照组，分析不同滤波处理的效果。在得到去噪结果之后，我们进行归一化处理，将像素值全部转换为 0-255 之间的离散值。最后，我们会分别得到各通道灰度直方图和融合灰度直方图。其中，各通道灰度直方图是不同频段单独计算的结果，每一个频段对应一个柱状直方图结果，但是我们为了方便，使用不同颜色表示不同的柱状直方图，显示在一张图里。而对于融合灰度直方图而言，我们分别采用 RGB 合成灰度化方法和 PCA 方法进行融合，并进行与各通道灰度直方图一样的统计和制图流程，得到融合成一个频段的融合灰度直方图，最后进行对比分析。本作业会从灰度直方图及绘制流程图中去比对其中各个细节，发现不同处理方式的区别和规律。

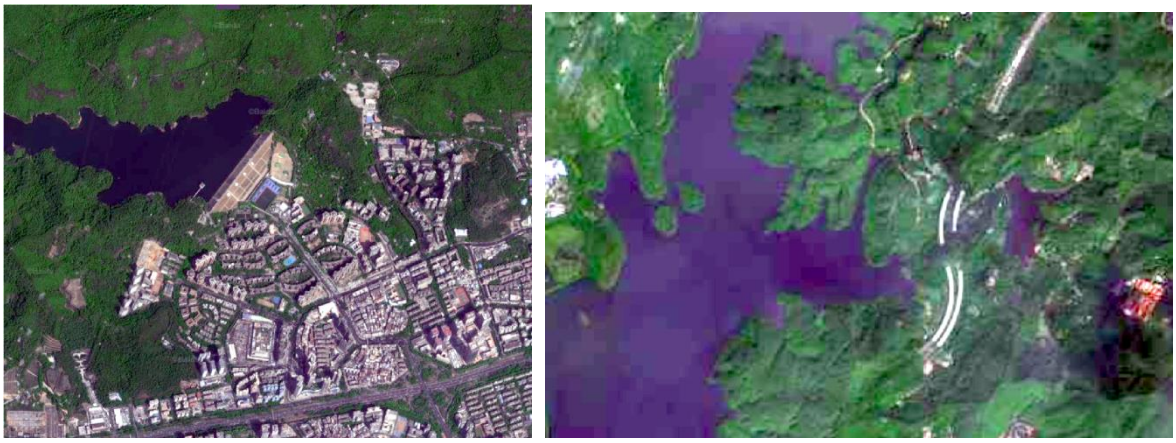


图 3：第一张和第二张初始遥感图，分别记为 001 和 002

图 4：第三张和第四张初始遥感图，分别记为 003 和 004

本作业综合采用两种语言，在数据预处理及数据整合主要使用 C++语言，制图主要使用 python 语言，分别采用 visual studio 及 visual studio code 环境进行。对于 C++环境而言,使用了系统库及 gdal 库;对于 python 环境而言,使用了 pandas、matplotlib 等库。

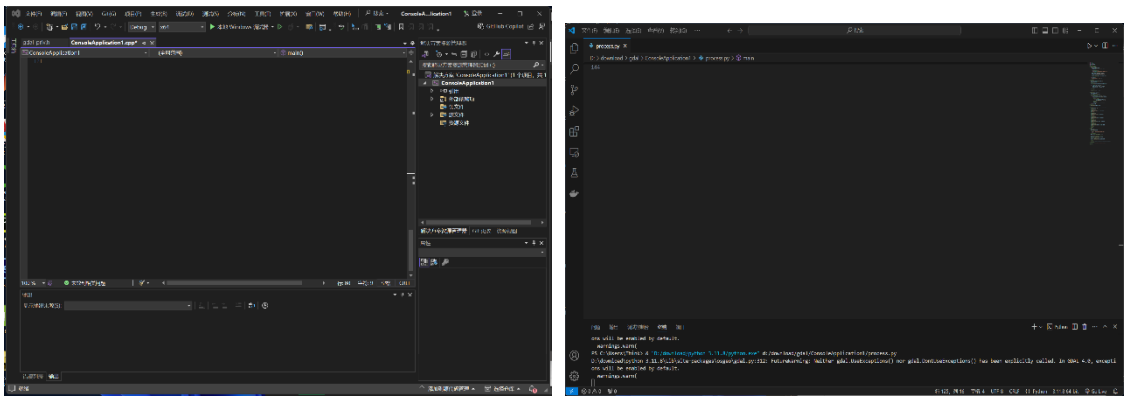


图 5：visual studio 和 visual studio code 的调试界面

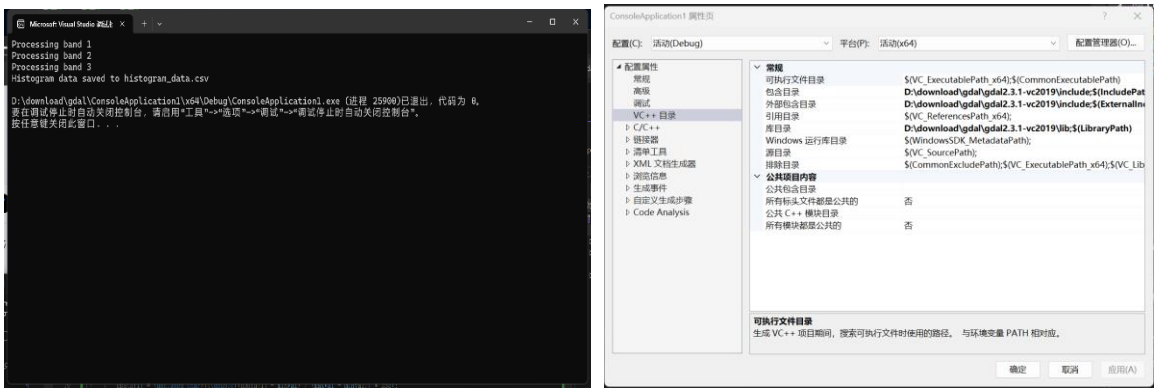


图 6：控制台输出界面及 gdal 库配置截图

二、预处理及生成各通道灰度统计数据理论基础

（1）预处理及滤波去噪

由于采用了不同的照片进行处理，因此首先需要把所有的照片都处理为同一种格式。Gdal 代码对于 tif 格式的文件比较友好，因此，将 tif 作为统一最终格式，这里是使用 ArcGIS pro 的自动数据处理功能实现的，这里不做过多赘述。

高斯滤波和中值滤波是图像处理中的两种常用的平滑技术，能够有效去除噪声。

高斯滤波是通过对图像应用高斯核进行加权平均的一种线性滤波器。它根据像素的空间距离，用高斯分布函数为每个像素赋予权重，距离中心点越远，权重越小，从而实现平滑图像并减少噪声。高斯滤波使用的是一个二维的高斯函数生成的核：

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

其中，x 和 y 是距离核中心的坐标，σ 是标准差，控制高斯核的扩展程度。更大的 σ 会导致图像的模糊效果更强。

在代码中，首先计算了高斯核的大小（kernelSize）并生成高斯权重矩阵。在遍历每个像素时，通过权重矩阵将邻域内的像素值加权平均，最终得到滤波后的图像。关键部分是高斯核的生成（利用高斯函数计算每个权重），以及对所有权重进行归一化，使其和为 1。

中值滤波是一种非线性滤波器，它通过对一个像素周围的邻域内的所有像素值进行排序，并取其中的中值来代替中心像素值，这种方法不会像均值滤波那样模糊边缘。中值滤波会首先为每个像素选定一个邻域矩阵，随后将邻域内的像素值排序，取排序后的中间值作为中心像素的新值。中值滤波能够保留图像的边缘细节，同时有效去除孤立的噪声点。

在代码中，首先在每个像素周围构建一个窗口（大小为 kernelSize），并将窗口内的所有像素值放入窗口向量中，然后对窗口中的值进行排序，取中间的值作为滤波后的像素值。

（2）归一化

在滤波处理之后对图像进行归一化处理的目的是将像素值的范围调整到一个标准的区间（通常是 0-255），从而增强对比度，使图像在显示或后续处理时具有更好的效果。归一化是一种线性变换，它将数据值缩放到特定的范围。图像处理中的归一化通常用于将像素值从原有的范围（例如，滤波后可能不再是 0-255）重新映射到 [0, 255] 的范围。

归一化的数学表达式为：

$$P_{norm} = \frac{\mathrm{p} - \min(\mathrm{Val})}{max(Val) - min(Val)} \times 255$$

其中，p 是当前像素值，$min(Val)$ 是图像中所有像素的最小值，$max(Val)$ 是图像中所有像素的最大值，P_{norm} 是归一化后的像素值，范围在 0 到 255 之间。这个公式将图像中每个像素的值按比例映射到新的区间中，使得整个图像的对比度被拉伸，特别是对于滤波后像素值范围较小的图像，可以显著提高视觉效果。

在代码中，首先遍历图像的每个像素，记录下当前图像中的最小值和最大值。最小值和最大值决定了归一化的上下限。随后，对图像中的每个像素应用归一化公式，将其缩放到 [0, 255] 范围。

（3）生成各通道灰度统计数据（main 函数）

波段遍历： 代码使用一个循环遍历图像的每个波段（即每个颜色通道）。这意味着对于每个波段，都会单独进行灰度值的统计。

读取波段数据： 在循环内部，首先通过 GetRasterBand(band) 获取当前波段的指针，然后分配内存以存储波段的像素数据，使用 RasterIO() 方法将该波段的像素值读入到 pData 数组中。

归一化处理： 读取完波段数据后，调用 normalize() 函数对 pData 进行归一化。这一步是为了确保每个波段的像素值在计算直方图之前都处于相同的标准范围（0 到 255），从而提高后续统计的准确性。

直方图初始化： 代码为直方图创建一个数组，长度为 256（对应灰度值的范围 0-255），并将所有元素初始化为 0。这个数组将用来统计每个灰度值出现的频率。

统计灰度值： 接下来，使用两个嵌套循环遍历整个波段的像素值。在每个像素位置上，代码读取其灰度值，并更新直方图数组中对应灰度值的计数。

保存统计数据： 在统计完成后，代码将直方图的数据写入到 CSV 文件中。每个波段的灰度值和对应的计数都会被写入一行，使得每个波段的统计结果易于后续分析。

三、不同滤波情形下各通道灰度直方图展示

（1）关于高斯滤波和中值滤波

在前面的理论中提到了比较通用的两种滤波，一种为高斯滤波，另外一种是中值滤波。高斯滤波能够较好地去除图像采集中的随机噪声，在过滤的同时保持图像整体结构，避免图像过度模糊，且具有较好的数学性质（如可分离性）。但是高斯滤波对于椒盐噪声（即随机出现的白点或黑点）效果较差，因为高斯滤波会对极端值敏感，另外，可能会导致图像边缘模糊，影响边缘检测的效果。中值滤波则特别适合去

除椒盐噪声，在图像降噪和细节保留方面表现良好，通过中值计算避免了极端值的影响，能够较好地保留边缘信息，但在速度和细节保持方面可能不如高斯滤波。

为了更好地体现两种滤波的不同之处，以 001 图像为例进行了两种滤波及不用滤波情况下的呈现情况。

原始图像001　　高斯滤波处理结果　　中值滤波处理结果

图 7：001 图像及两种滤波处理后的结果

可以发现，在滤波处理之后，图像都发生了较明显的变化。在经过高斯滤波处理后的图像更像是在原始图像上进行了模糊化处理，颜色到颜色之间的渐变更加柔和，而中值滤波处理之后的图像虽然看似也是进行了模糊化，但是颜色与颜色之间的边缘更加明显了。这些发现仅仅只是观察图像对比的结果。

（2）不同滤波情形下的各通道灰度直方图展示

在研究了高斯滤波及中值滤波的区别之后，现在采用无滤波、高斯滤波、中值滤波分别处理每张图像，并且展示每个频段的结果。在下图中，第一张为原始图像，第二张为无滤波处理的灰度直方图，第三张为高斯滤波处理的灰度直方图，第四张为中值滤波处理的灰度直方图。在图中，蓝色代表第一频段，橙色代表第二频段，绿色代表第三频段。为了方便起见，不是每一个频段都出一张直方图，而是通过不同颜色搭配的形式，同时展示三个频段的灰度直方图，这样可以更好地对三个频段的灰度直方图进行对比，也同时可以对不同情形滤波处理下的灰度直方图进行处理。

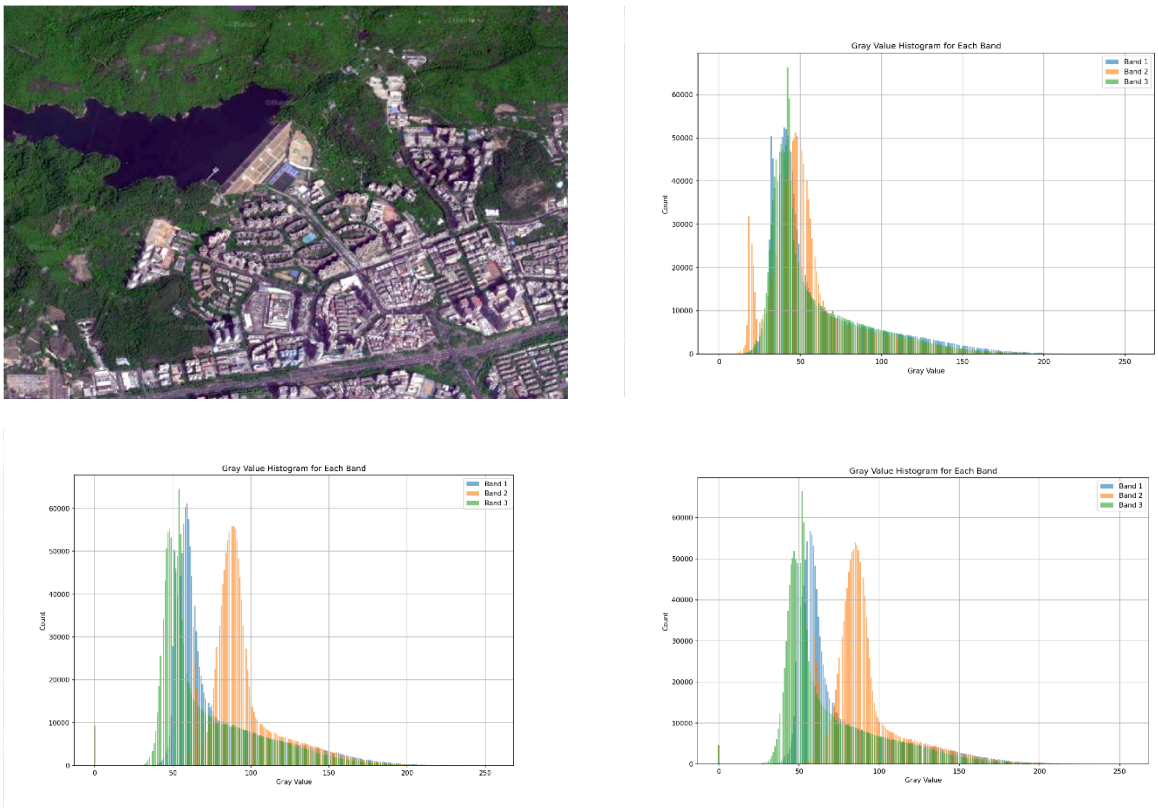
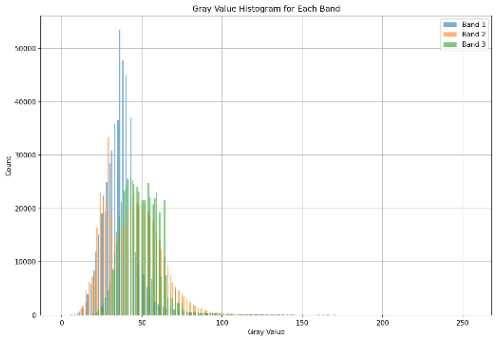
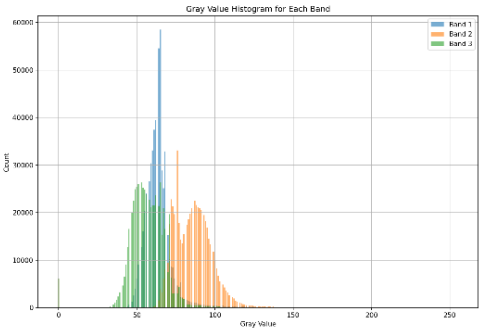


图 8：001 图像、无滤波、高斯滤波、中值滤波处理后的灰度直方图结果

在 001 图像中，可以发现，无滤波处理下的灰度直方图和滤波处理后的灰度直方图峰值的区间有明显的不同。在无滤波处理下，三个频段的峰值都集中在 50 的灰度值左右，说明图像的灰度值比较统一。在 25 灰度值左右，第二频段有一个小范围的峰值，但可以明显发现在经过滤波处理之后，这种小范围的峰值都被大大削弱了，甚至直接消失了。此外，经过滤波的处理，有部分范围的灰度值分布更加集中且平滑了。在 001 图像中，高斯滤波和中值滤波处理结果相似，但是也可以发现一些区别，高斯滤波在第三频段处理得稍微差一些，在第一频段处理得更好，说明第一频段的高斯噪声较多，而第三频段的椒盐噪声较多，使得高斯滤波在处理时出现了一些差异。从整体上看，大部分频段的原始结果还是比较优良的，使得总体滤波的处理没有显示出特别大的差别。





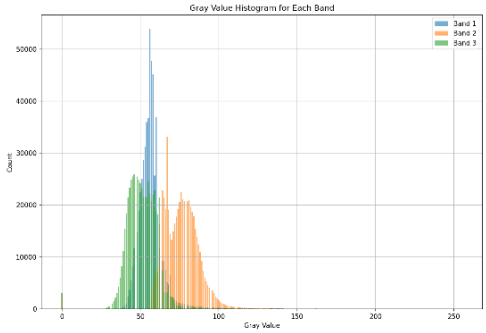
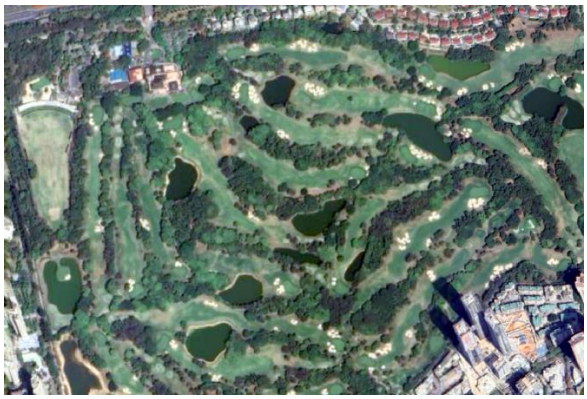
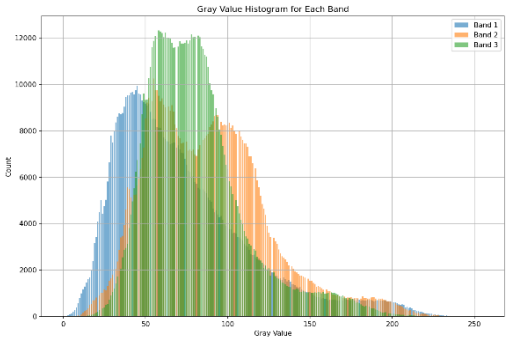
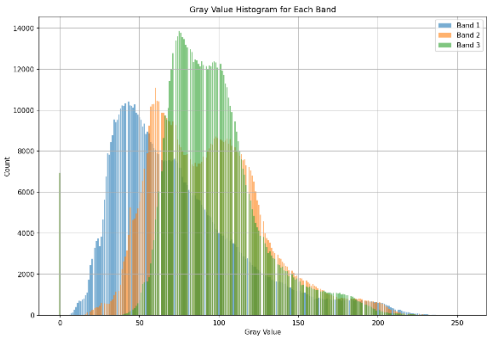


图 9：002 图像、无滤波、高斯滤波、中值滤波处理后的灰度直方图结果

在 002 图像中，无滤波和有滤波处理后的结果，其峰值没有像 001 图像中那样出现偏移的情况，同时可以明显看到在经过滤波处理后的直方图更加平滑且美观，而且效果是中值滤波在高斯滤波之上，说明椒盐噪声较多，而高斯噪声较少，同时各个频段的噪声分布都比较均匀。从整体上看，002 图像的直方图也也相对集中在 50 左右，但是和 001 图像不同的是，最大值会相对来说更小一些，说明 002 图像直方图虽然集中区间也和 001 图像相同，但是相对来说会比 001 图像分布得跟均匀一些。001 图像的极端值更多更明显，而 002 图像基本是只有第一频段在 50 到 100 灰度值之间出现了一个急剧上升的情况。







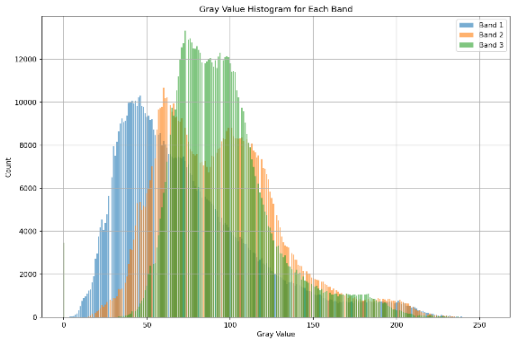


图 10：003 图像、无滤波、高斯滤波、中值滤波处理后的灰度直方图结果

在 003 图像中，无滤波和有滤波处理后的结果与 002 图像相似，峰值没有大范围偏移，但是不同在于对于 003 图像而言，有无滤波处理的差别极小，说明原始图像原本的质量较高，噪声较少。而且从整体上来看，第一频段、第二频段和第三频段分别集中在 40、55、75 左右的灰度值，而且第二频段和第三频段都在 100-150 之间的灰度值呈现了第二个小的峰值，同时从每个频段的分布来看，集中趋势不没有 001 和 002 图像那么明显，均匀的趋势更明显，说明该图像均衡化程度更高，且亮度分布不集中。

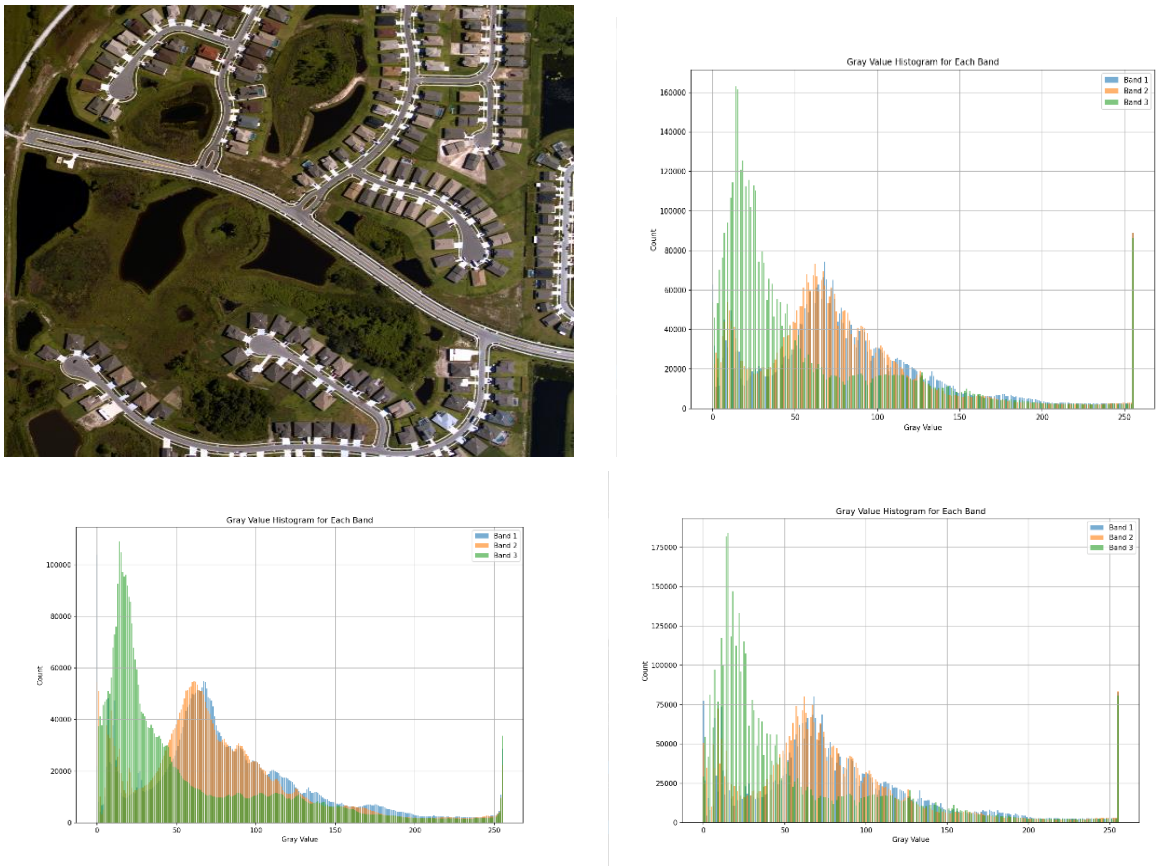


图 11：004 图像、无滤波、高斯滤波、中值滤波处理后的灰度直方图结果

在 004 图像中，无滤波和有滤波处理后的结果显示，峰值没有大范围偏移，但是可以明显看到该图像在高斯滤波之后的效果远比中值滤波要好。中值滤波显示出的结果和原图像结果没有太大差异，这也证明了该图像较少椒盐噪声，而更多为连续的、分布均匀的随机噪声，使得经过高斯滤波处理后的结果直方图表现得更平滑，其中更少有区间突兀空缺的情形。从整体上看，该结果相对于其它三个图像而言更加不同，其中第一频段和第二频段的峰值在 50-100 灰度值之间，但是第三频段的峰值却集中在 25 左右，这样的错峰分布使得整体在 0-100 灰度值呈现更加的不稳定化。另外，可以发现的是，在接近最大值 255 时，也出现了一个突兀的翘尾现象，说明图像中纯白色的区块相对前几张更多更集中。

四、融合灰度直方图及相关理论

（1）融合灰度直方图相关理论

在遥感图像处理中，RGB 灰度合成和主成分分析（PCA）合成是将多波段图像合成为单波段灰度图像的两种常用方法。每种方法都有其独特的优势，用于不同的场景，便于进一步分析和处理图像数据，特别是在进行灰度直方图分析时。

RGB 灰度合成是一种将彩色图像转换为灰度图像的常用方法。在遥感图像中，常见的波段组合如红（Red）、绿（Green）、蓝（Blue）波段可以表示为 RGB 彩色图像。灰度图像表示亮度信息，没有颜色，合成的结果是以人眼的感知亮度为基础。

灰度值通常是根据红、绿、蓝波段的不同比重加权合成的，因为人眼对不同颜色的敏感度不同。一般来说，绿色波段对人眼影响较大，其次是红色，蓝色影响最小。因此，合成公式常为：

$$Gray = 0.2989 \times \mathrm{R} + 0.5870 \times \mathrm{G} + 0.1140 \times \mathrm{B}$$

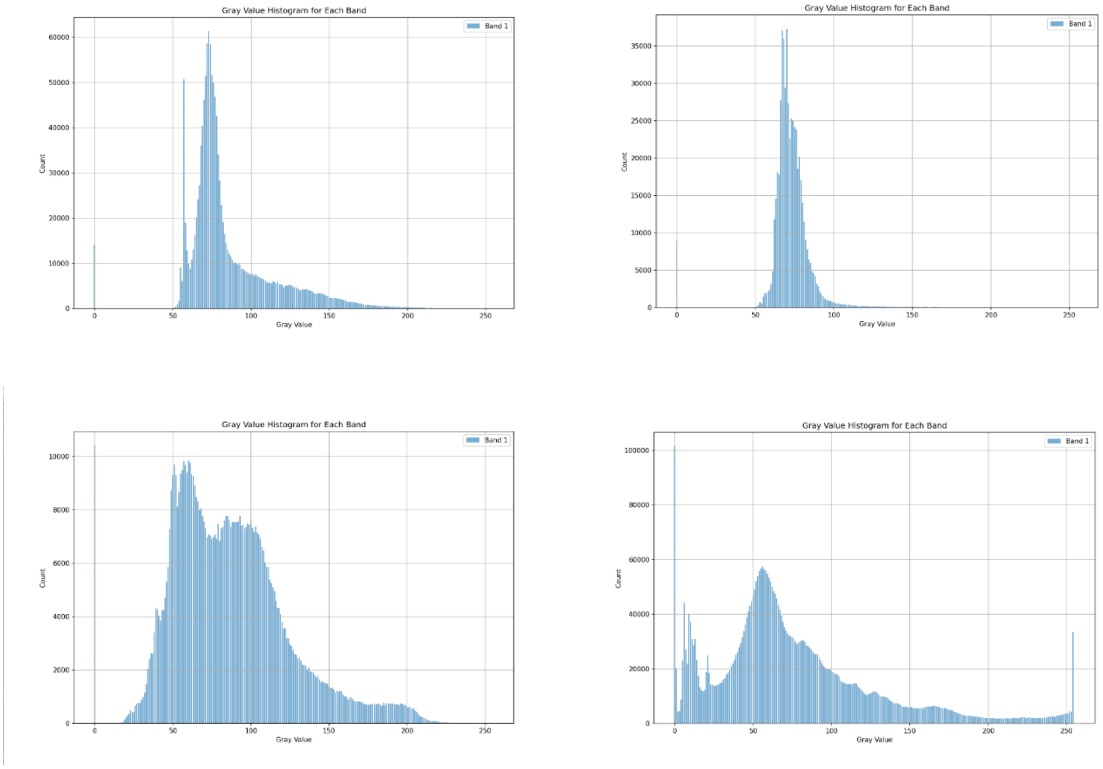
在代码中，对应选取三个波段作为输入数据，按公式将 RGB 波段的值按比例加权求和，得到每个像素的灰度值，再按照原来直方图数据统计的方法求解直方图。这样的方法比较适合于可见光波段的遥感图像处理。

主成分分析（PCA）是一种常用的降维技术，用于将多波段数据转化为少量的“主成分”。PCA 通过分析数据的协方差矩阵，找出数据的主要变化方向，将具有最大信息量的部分集中到几个主成分中。对于遥感图像，PCA 可以将多个波段（如多光谱或高光谱波段）综合为少量主成分，其中第一个主成分（PC1）通常包含了最多的光谱信息，因此可以作为单波段图像用于后续分析。

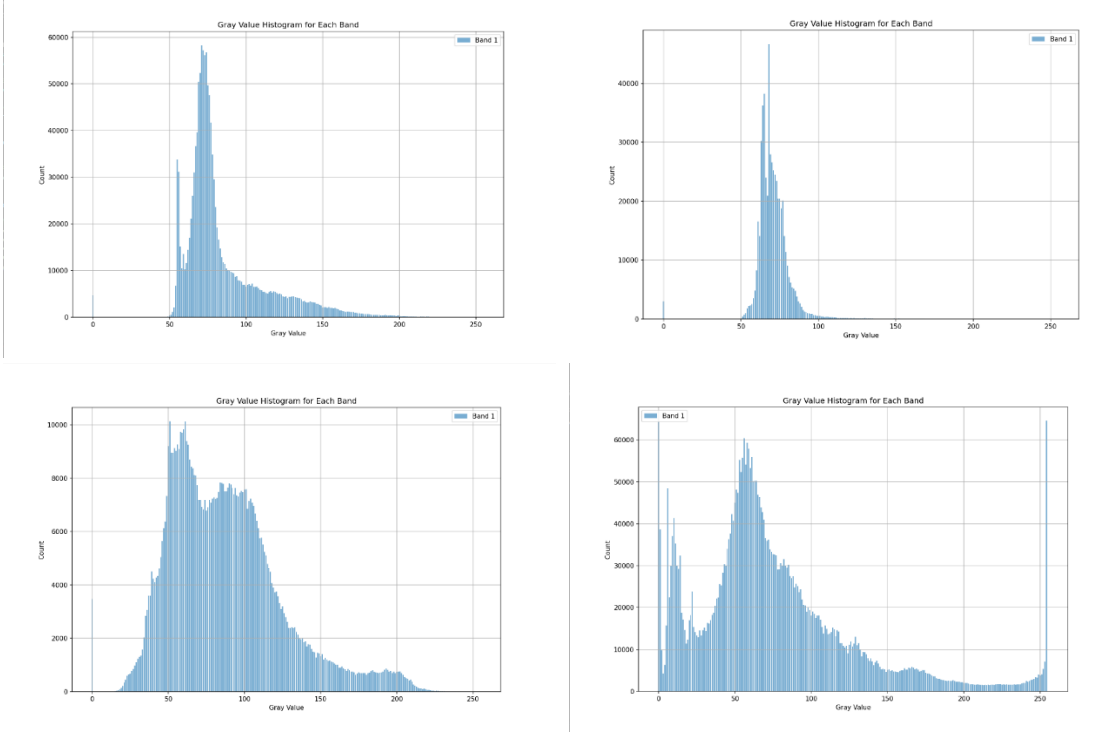
RGB 灰度合成是一种简单、基于人眼感知的合成方法，适用于可见光波段的遥感图像快速处理。而 PCA 合成则是一种更复杂的降维技术，适合高光谱或多波段数据，能够提取图像的主要信息并消除波段间的相关性。在需要对单波段灰度图像进行直方图分析时，这两种方法是遥感图像处理中最通用的合成方法。

在本次实验中，将分别就不同的融合方法求解直方图并进行展示。

（2）融合灰度直方图

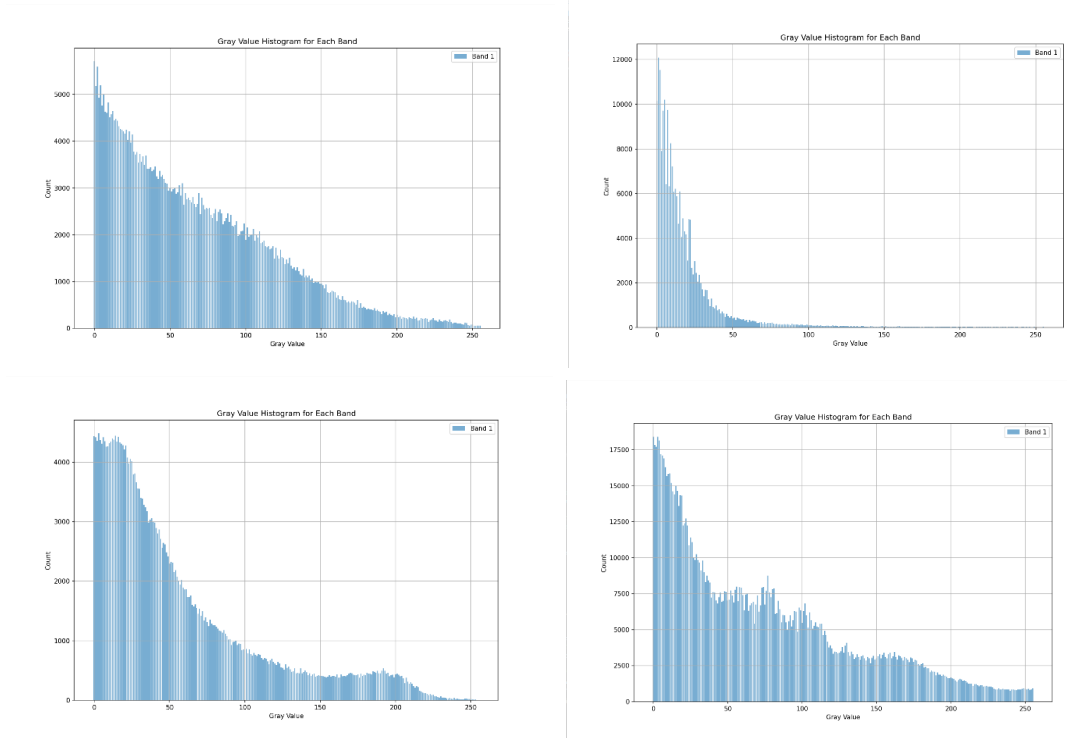


001-004 经过高斯滤波之后的 RGB 融合灰度直方图

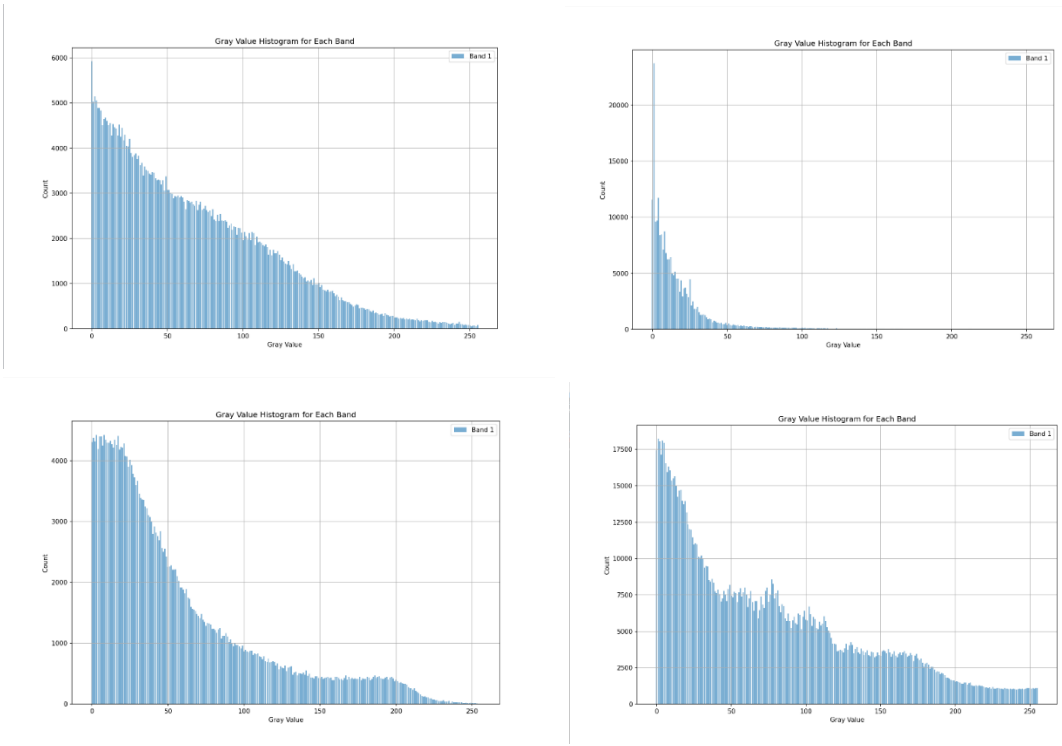


001-004 经过中值滤波之后的 RGB 融合灰度直方图

以上的图展示了 001-004 在经过高斯滤波及中值滤波之后的 RGB 融合灰度直方图。可以发现，两种滤波处理下的 RGB 融合灰度直方图除了在小部分细节有差异之外，基本上没有特别大的差异。从这些图，我们可以分析四张图的整体分布的不同。RGB 融合灰度之后，直方图非常的平滑且具有可比性，没有出现之前单一频段的大量区间空缺的情况。001 和 002 图像显示出整体分布比较集中，基本集中在 50-100 的区间。003 和 004 图像相对来说会均衡一些，003 出现了一个大的峰值和一个小的峰值，且两个峰值相隔较近，使得整体值都大部分分布在 50-100 之间，也可以观察出其整体均值也为四张图最大；004 这是双峰值，且两个峰值相隔较远，一个分布在 10 灰度值左右，另外一个分布在 70 灰度值左右，前者的覆盖宽度非常小，而后者的覆盖宽度非常大。相对来说不会像 003 那么均衡，但是仍然要比 001 和 002 图像要均衡。



001-004 经过中值滤波之后的 PCA 融合灰度直方图



001-004 经过高斯滤波之后的 PCA 融合灰度直方图

以上的图展示了 001-004 在经过高斯滤波及中值滤波之后的 PCA 融合灰度直方图。可以发现，两种滤波处理下的 PCA 融合灰度直方图也同样差别比较小，但是 PCA 融合灰度直方图和 RGB 融合灰度直方图显示出了很大的区别。PCA 图像基本上呈现一个从低灰度值到高灰度值递减的状态，这也与 PCA 的算法关系密切，使得 PCA 结果不能以灰度值为标准去看，而是要以整体递减的状态去研究。可以发现，001 图像呈现缓慢均匀下降的状态，而 002 图像呈现快速下降的状态，003 图像呈现较快下降的状态，而 004 图像呈现波动下降的状态。

五、总结

本作业围绕流程介绍、理论基础及实验结果展开，选取四种图像进行了直方图相关的充分分析，其中，滤波方法对比实验了无滤波、高斯滤波和中值滤波情形下的直方图差异；融合方法对比实验了 RGB 融合和 PCA 融合的直方图差异。从直方图本质出发，分析图像像素值整体和局部分布。由于对图像处理技术知识掌握有限，可能存在错漏分析的情况，未来会进一步在课程学习中不断改正和进步。

六、核心代码展示

（1）高斯滤波

```
void gaussianFilter(unsigned char* pData, unsigned char* pFilteredData, int width, int height,
double sigma) {
    int kernelSize = 2 * static_cast<int>(std::ceil(3 * sigma)) + 1;
    int halfSize = kernelSize / 2;

    // 生成高斯核
    std::vector<double> kernel(kernelSize * kernelSize);
    double sum = 0.0;
    for (int y = -halfSize; y <= halfSize; y++) {
        for (int x = -halfSize; x <= halfSize; x++) {
            double value = std::exp(-(x * x + y * y) / (2 * sigma * sigma));
            kernel[(y + halfSize) * kernelSize + (x + halfSize)] = value;
            sum += value;
        }
    }

    // 归一化高斯核
    for (auto& val : kernel) {
        val /= sum;
    }

    // 应用高斯核
    for (int y = halfSize; y < height - halfSize; y++) {
        for (int x = halfSize; x < width - halfSize; x++) {
            double pixelValue = 0.0;
            for (int i = -halfSize; i <= halfSize; i++) {
                for (int j = -halfSize; j <= halfSize; j++) {
                    pixelValue += kernel[(i + halfSize) * kernelSize + (j + halfSize)] *
                        pData[(y + i) * width + (x + j)];
                }
            }
            pFilteredData[y * width + x] = static_cast<unsigned char>(pixelValue);
        }
    }
}
```

（2）中值滤波

```
void medianFilter(unsigned char* pData, unsigned char* pFilteredData, int width, int height,
int kernelSize) {
    int halfSize = kernelSize / 2;
    std::vector<unsigned char> window;
```

```
    for (int y = halfSize; y < height - halfSize; y++) {
        for (int x = halfSize; x < width - halfSize; x++) {
            window.clear();

            for (int i = -halfSize; i <= halfSize; i++) {
                for (int j = -halfSize; j <= halfSize; j++) {
                    window.push_back(pData[(y + i) * width + (x + j)]);
                }
            }

            std::sort(window.begin(), window.end());
            pFilteredData[y * width + x] = window[window.size() / 2];
        }
    }
}
```

（3）归一化

```
void normalize(unsigned char* pData, int width, int height) {
    unsigned char minVal = 255, maxVal = 0;

    // 找到最小值和最大值
    for (int i = 0; i < width * height; i++) {
        if (pData[i] < minVal) minVal = pData[i];
        if (pData[i] > maxVal) maxVal = pData[i];
    }

    // 归一化处理
    for (int i = 0; i < width * height; i++) {
        pData[i] = static_cast<unsigned char>(((double)(pData[i] - minVal) / (maxVal - minVal))
* 255);
    }
}
```

（4）主函数：生成各通道灰度统计数据

```
int main()
{
    GDALAllRegister();
    const char* filename = "D:\\download\\gdal\\001.tif";
    GDALDataset* pDatasetRead = (GDALDataset*)GDALOpen(filename, GA_ReadOnly);

    if (pDatasetRead == NULL)
    {
        std::cout << "Cannot open file." << std::endl;
        return -1;
    }
```

```
    int lWidth = pDatasetRead->GetRasterXSize();
    int lHeight = pDatasetRead->GetRasterYSize();
    int nBands = pDatasetRead->GetRasterCount();

    std::ofstream outputFile("histogram_data.csv");
    if (!outputFile.is_open())
    {
        std::cout << "Unable to open output file." << std::endl;
        return -1;
    }

    outputFile << "Band, Gray Value, Count\n";  // CSV header

    for (int band = 1; band <= nBands; band++)
    {
        std::cout << "Processing band " << band << std::endl;

        // 读取当前波段
        GDALRasterBand* pBand = pDatasetRead->GetRasterBand(band);
        unsigned char* pData = (unsigned char*)CPLMalloc(lWidth * lHeight * sizeof(unsigned
char));
        pBand->RasterIO(GF_Read, 0, 0, lWidth, lHeight, pData, lWidth, lHeight, GDT_Byte, 0,
0);

        // 归一化
        normalize(pData, lWidth, lHeight);

        // 计算直方图
        int histogram[256] = { 0 };
        for (int i = 0; i < lHeight; ++i)
        {
            for (int j = 0; j < lWidth; ++j)
            {
                int pixelValue = pData[i * lWidth + j];
                histogram[pixelValue]++;
            }
        }

        // 写入当前波段的直方图数据
        for (int i = 0; i < 256; ++i)
        {
            outputFile << band << "," << i << "," << histogram[i] << "\n";
        }

        CPLFree(pData);
    }

    outputFile.close();
```

```
    GDALClose(pDatasetRead);

    std::cout << "Histogram data saved to histogram_data.csv" << std::endl;

    return 0;
    }
```

（5）直方图制图

```
import pandas as pd
import matplotlib.pyplot as plt

# 读取 CSV 文件，不读取列名
data = pd.read_csv("D:\\download\\gdal\\ConsoleApplication1\\histogram_data001.csv", header =
None)

# 检查数据的前几行，确保格式正确
print(data.head())

# 确保波段列是字符串类型，然后提取唯一的波段
data[0] = data[0].astype(str)  # 将波段列转换为字符串
bands = data[0].unique()

# 设置绘图的大小
plt.figure(figsize = (12, 8))

# 遍历每个波段并绘制直方图
for band in bands :
# 筛选当前波段的数据
band_data = data[data[0] == band]

# 确保第二列和第三列的数据为数字
gray_values = pd.to_numeric(band_data[1], errors = 'coerce')  # 转换为数值，无法转换的将变为NaN
counts = pd.to_numeric(band_data[2], errors = 'coerce')

# 绘制柱状图
plt.bar(gray_values.dropna(), counts.dropna(), label = f'Band {band}', alpha = 0.6)

# 添加标题和标签
plt.title('Gray Value Histogram for Each Band')
plt.xlabel('Gray Value')
plt.ylabel('Count')
plt.legend()
plt.grid()

# 显示图形
plt.show()
```

（6）RGB灰度合成

```
// 合成灰度图
unsigned char* pGrayData = (unsigned char*)CPLMalloc(lWidth * lHeight * sizeof(unsigned char));

for (int i = 0; i < lWidth * lHeight; i++)
{
    // 灰度值 = 0.2989 * R + 0.5870 * G + 0.1140 * B
    pGrayData[i] = static_cast<unsigned char>(0.2989 * pFilteredR[i] + 0.5870 * pFilteredG[i] +
0.1140 * pFilteredB[i]);
}
```

（7）PCA合成

```
# 主成分分析（PCA）降维
def apply_pca(bands_data, num_components = 1) :
    pca = PCA(n_components = num_components)
    reshaped_data = np.stack(bands_data, axis = -1).reshape(-1, len(bands_data))
    pca_result = pca.fit_transform(reshaped_data)
    return pca_result.reshape(bands_data[0].shape)
```