

重 庆 交 通 大 学

学生实验报告

课 程 名 称： 数 字 图 像 处 理 .

开 课 实 验 室： 软 件 实 验 中 心 .

学 院： 信息学院 年级 物联网工程专业 2 班

学 生 姓 名： 李骏飞 学 号 632109160602

指 导 教 师： 蓝 章 礼 .

开 课 时 间： 2023 至 2024 学 年 第 二 学 期

成 绩	
教师签名	

实验项目名称		伪彩色处理			
姓名	李骏飞	学号	632109160602	实验日期	2024.4.29
<p>教师评阅：</p> <p>1:实验目的明确<input type="checkbox"/>A<input type="checkbox"/>B<input type="checkbox"/>C<input type="checkbox"/>D</p> <p>2:内容与原理<input type="checkbox"/>A<input type="checkbox"/>B<input type="checkbox"/>C<input type="checkbox"/>D</p> <p>3:实验报告规范<input type="checkbox"/>A<input type="checkbox"/>B<input type="checkbox"/>C<input type="checkbox"/>D</p> <p>4:实验主要代码与效果展示<input type="checkbox"/>A<input type="checkbox"/>B<input type="checkbox"/>C<input type="checkbox"/>D</p> <p>5:实验分析总结全面<input type="checkbox"/>A<input type="checkbox"/>B<input type="checkbox"/>C<input type="checkbox"/>D</p>					
实验记录					
<p>一、实验目的</p> <p>基本要求：采用一种自己定义的算法，对灰度图像进行伪彩色处理 答案要有算法描述，核心代码，完成图片的效果。</p> <p>二、实验主要内容及原理</p> <p>伪彩色处理：</p> <p>伪彩色处理是指将灰度图像转换成彩色图像。因为人眼对于彩色的分辨能力远高于对灰度图像的分辨能力，所以将灰度图像转换成彩色可以提高人眼对图像细节的别能力。但是可以看出伪彩色并不能真实的反映图像的彩色情况。</p> <p>伪彩色的应用最广泛的就是在红外图像中，在设计红外热成像系统的时候，需要将温度数据绘制在 UI 界面上，我们大致都见过红外图像的样子，它就是通过将温度数据转化为灰度图像后，再转化成伪彩色图得到的，从而让人们能够对图像中物体的温度有了更清晰的别能力（相比于灰度图的展示）。</p> <p>(1) 色带映射法：</p>					

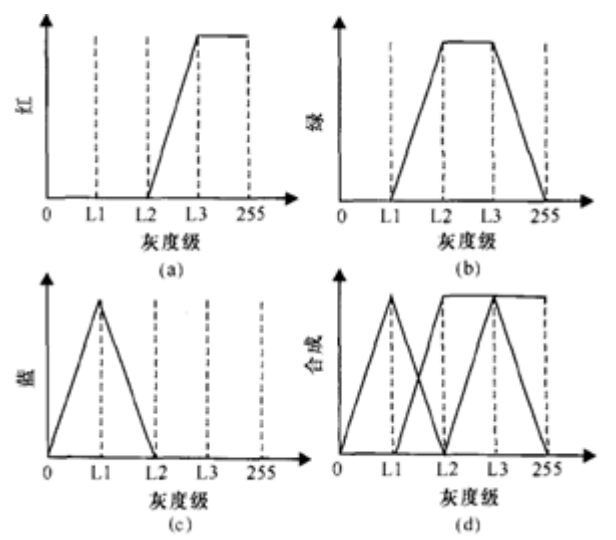
伪彩色处理其实就是灰度空间到 RGB 彩色空间的简单映射，不同的映射就对应了不同的伪彩色化方法。最主流的映射方法的色带对应如下：



它的主要功能是使灰度图中亮度越高的像素点，在伪彩色图中对应的点越趋向于红色；亮度越低，则对应的伪彩色越趋向于蓝色；总体上按照灰度值高低，由红渐变至蓝，中间色为绿色。由于它符合人们对于颜色与温度联系的认识（即红色代表高温警示色，蓝色代表低温），这种映射方法也是红外图像中最主流的映射方法（还有两种常用映射，它们的色带分别称为彩虹色带和铁红色带）。在本次实验中，我将通过查找彩虹色带和铁红色带的映射表，提前将每个灰度值对应的映射结果算好，根据这个映射规则，使用 C# 实现灰度图像的伪彩色处理。

(2) 灰度级-彩色变换法

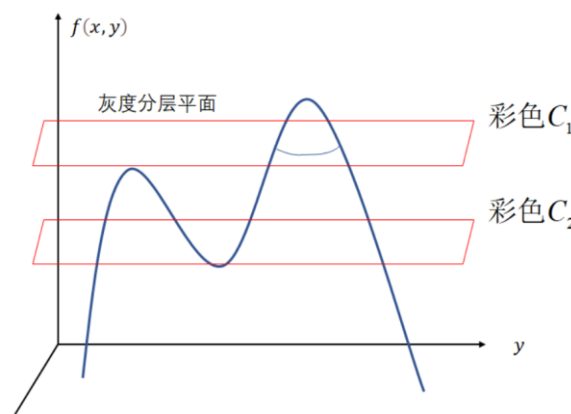
灰度级-彩色变换法可以将灰度图像变为具有多种颜色渐变的连续彩色图像。主要就是将图像通过不同变换特性的红、绿、蓝 3 个变换器，然后将三个颜色通道的输出合成某种颜色。由于三种颜色变换的不同，使得不同大小灰度级可以合成不同的颜色。一组典型的变换传递函数如下图。



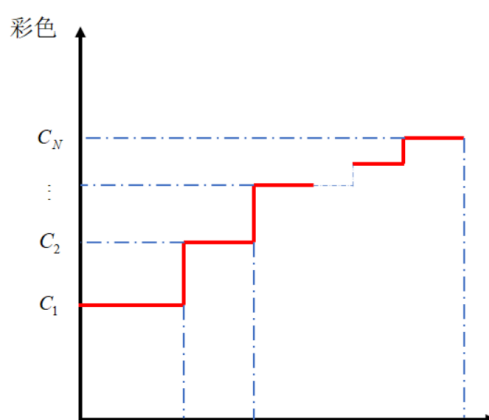
这里面需要注意的地方，代码只能是处理 JPG 格式的灰度图像，因为 JPG 图像的颜色深度是 24 位表示 (R,G,B)，每像素由 3 个字节表示即可，然而 PNG 图像的颜色深度是 32 位表示 (R,G,B,A)。

(3) 灰度分层法

将一幅灰度图像按灰度级别映射到彩色图像



灰度分层法又称灰度分割法或密度分层法，是伪彩色处理技术中最基本、最简单的方法。设一幅灰度图像 $f(x,y)$ 可以看作坐 (x,y) 的一个密度函数。把此图像的灰度分成若干等级，即相当于用一些和坐标平面即 $x-y$ 平面平行的平面在相交区域切割此密度函数。如上图所示，将这种映射用图像表示便是如下：



三、实验环境

Windows11

Visual Studio2021

C#语言

四、实验主要代码与效果展示

色带映射法

➤ 算法描述:

PGrayToPseudoColor2 函数接受两个参数:src 是输入的灰度图像,type 是转换类型,可以选择使用铁红(1)或彩虹(2)色带。

函数创建一个与输入图像大小相同的新位图 a,然后获取该位图的图像数据。通过调用 LockBits 方法,可以获取位图数据的指针,并指定以可读写的方式访问图像数据。

接下来,函数通过遍历图像的每个像素点来进行转换。对于每个像素点,首先将其灰度值除以 2,得到一个临时变量 temp。然后根据转换类型选择相应的颜色表进行颜色映射。

如果 type 为 1,即选择了铁红色带,函数使用名为 ironTable 的表进行映射。表中存储了每个灰度值对应的红、绿、蓝三个通道的颜色值。通过查表得到相应的颜色值后,将它们分别赋给像素点的蓝、绿、红通道。

如果 type 为 2,即选择了彩虹色带,函数使用名为 rainTable 的表进行映射。操作方式与铁红色带类似。

当完成所有像素点的颜色映射后,函数释放位图数据并返回转换后的位图。如果 type 参数不合法,函数则抛出一个异常并显示相应的错误消息。

该函数通过查表的方式将灰度图像转换为伪彩色图像,用户通过选择不同的色带可以实现不同的颜色效果。

```
1. // 铁红色带映射表
2. // 每一行代表一个彩色分类,存放顺序是 RGB
3. public static byte[,] ironTable = new byte[128, 3] {
4.     {0, 0, 0},
5.     {0, 0, 0},
6.     {0, 0, 36},
7.     {0, 0, 51},
8.     {0, 0, 66},
9.     -----// 部分铁红色带映射表
10.
11. // 彩虹色带映射表
12. public static byte[,] rainTable = new byte[128, 3] {
13.     {0, 0, 0},
14.     {0, 0, 0},
15.     {15, 0, 15},
16.     {31, 0, 31},
17.     {47, 0, 47},
```

```

18. -----// 彩虹色带映射表
19.
20. public static Bitmap PGrayToPseudoColor2(Bitmap src, int type)
21. {
22.     try
23.     {
24.         if (type == 1)
25.         {
26.             Bitmap a = new Bitmap(src); // 创建一个与源图像大小相同的新位图
27.             Rectangle rect = new Rectangle(0, 0, a.Width, a.Height);
28.             System.Drawing.Imaging.BitmapData bmpData = a.LockBits(rect, System.Dr
                awing.Imaging.ImageLockMode.ReadWrite, System.Drawing.Imaging.PixelFormat.Format24
                bppRgb);
29.             int stride = bmpData.Stride; // 位图每行的字节数
30.             unsafe
31.             {
32.                 byte* pIn = (byte*)bmpData.Scan0.ToPointer(); // 指向位图数据的指
                针
33.                 int temp;
34.                 byte R, G, B;
35.
36.                 for (int y = 0; y < a.Height; y++)
37.                 {
38.                     for (int x = 0; x < a.Width; x++)
39.                     {
40.                         temp = pIn[0] / 2; // 计算颜色查找表的索引
41.
42.                         R = ironTable[temp, 0]; // 从铁红色带表中查找红色分量
43.                         G = ironTable[temp, 1]; // 从铁红色带表中查找绿色分量
44.                         B = ironTable[temp, 2]; // 从铁红色带表中查找蓝色分量
45.
46.                         pIn[0] = B; // 设置像素的蓝色分量
47.                         pIn[1] = G; // 设置像素的绿色分量
48.                         pIn[2] = R; // 设置像素的红色分量
49.
50.                         pIn += 3; // 移动指针到下一个像素
51.                     }
52.                     pIn += stride - a.Width * 3; // 移动指针到下一行的起始位置
53.                 }
54.             }
55.             a.UnlockBits(bmpData); // 释放对位图数据的锁定
56.             return a; // 返回转换后的位图
57.         }
58.         else if (type == 2)
59.         {
60.             Bitmap a = new Bitmap(src); // 创建一个与源图像大小相同的新位图
61.             Rectangle rect = new Rectangle(0, 0, a.Width, a.Height);
62.             System.Drawing.Imaging.BitmapData bmpData = a.LockBits(rect, System.Dr
                awing.Imaging.ImageLockMode.ReadWrite, System.Drawing.Imaging.PixelFormat.Format24
                bppRgb);
63.             int stride = bmpData.Stride; // 位图每行的字节数
64.             unsafe
65.             {
66.                 byte* pIn = (byte*)bmpData.Scan0.ToPointer(); // 指向位图数据的指
                针
67.                 int temp;

```

```

68.         byte R, G, B;
69.
70.         for (int y = 0; y < a.Height; y++)
71.         {
72.             for (int x = 0; x < a.Width; x++)
73.             {
74.                 temp = pIn[0] / 2; // 计算颜色查找表的索引
75.
76.                 R = rainTable[temp, 0]; // 从彩虹色带表中查找红色分量
77.                 G = rainTable[temp, 1]; // 从彩虹色带表中查找绿色分量
78.                 B = rainTable[temp, 2]; // 从彩虹色带表中查找蓝色分量
79.
80.                 pIn[0] = B; // 设置像素的蓝色分量
81.                 pIn[1] = G; // 设置像素的绿色分量
82.                 pIn[2] = R; // 设置像素的红色分量
83.
84.                 pIn += 3; // 移动指针到下一个像素
85.             }
86.             pIn += stride - a.Width * 3; // 移动指针到下一行的起始位置
87.         }
88.     }
89.     a.UnlockBits(bmpData); // 释放对位图数据的锁定
90.     return a; // 返回转换后的位图
91. }
92. else
93. {
94.     throw new Exception("type 参数不合法!"); // 对于无效的 type 参数，抛出异常
95. }
96. }
97. catch (Exception e)
98. {
99.     MessageBox.Show(e.Message.ToString()); // 显示包含异常消息的错误消息框
100.    return null; // 如果发生异常，返回 null
101. }
102. }

```

➤ 演示效果:

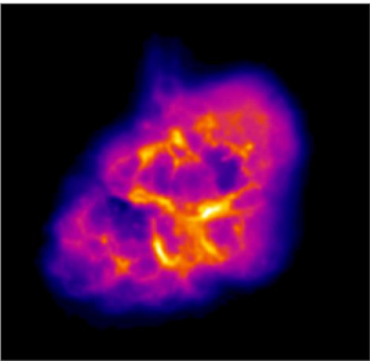
Ui 界面设计:

Form1

打开

还原

保存



高斯噪声 均值 标准差

确定

椒盐噪声

确定

随机噪声

确定

图像平滑 四邻域平均

确定

HSI

灰度化

图像缩放

二值化阈值

确定

形态学变换 膨胀

确定

SE大小 SE模板 正方形 类型 二值图

确定

亮度调节

确定

对比度调节

确定

图像旋转

确定

通道提取 红

确定

图像边缘提取 水平垂直差分法

确定

边缘叠加原图(锐化)

确定

伪彩色处理

铁红色带映射

彩虹色带映射


铁红色带映射

灰度级-彩色变换法


强度分层法

确定


确定



原图



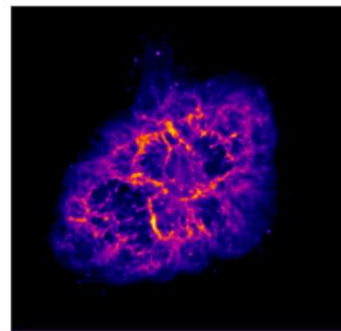
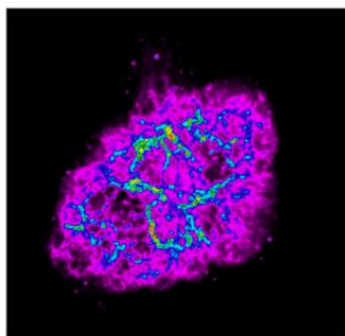
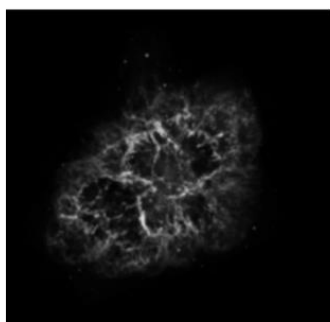
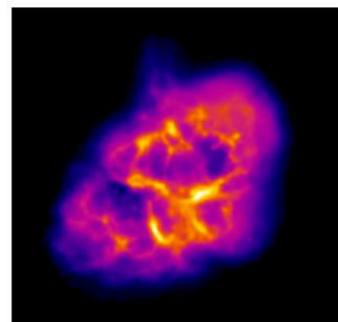
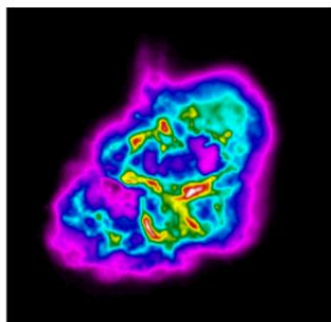
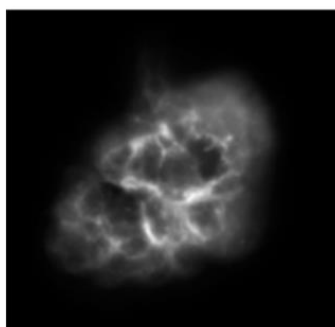
彩虹色带映射



铁红色带映射

比如可以使用不同色带增强拍摄的星云图片效果，明显看出铁红色带映射的效果更加震撼和具有冲击力：

- 8 -

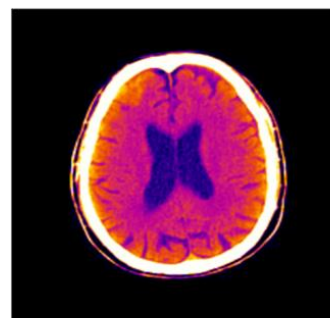
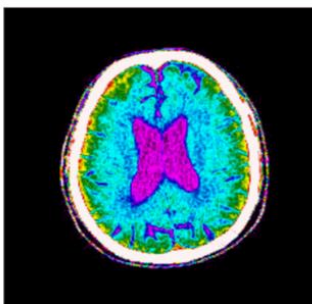
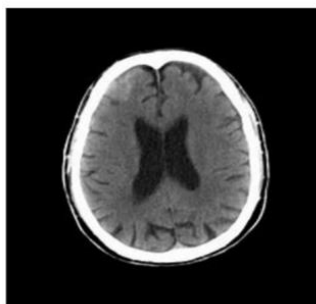
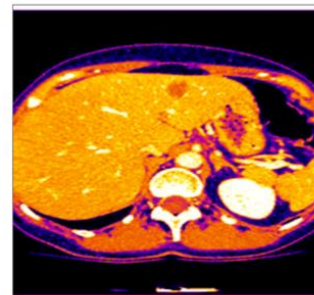
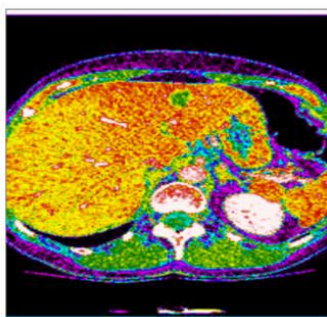
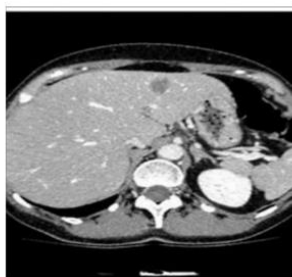


原图

彩虹色带映射

铁红色带映射

又比如在医学影像中，我们发现彩虹色带相对来说效果更显著，能帮助我们观察到更多的病理特征：



原图

彩虹色带映射

铁红色带映射

这里我只是简单了举例了两种常见的色带映射方式，在 `opencv` 中共有 20 多种色带映射，我们可以根据需求选择不同的对灰度图像进行伪彩色化，当然 `c#` 也提供了一些接口，帮助我们调用响应的色带映射，但这里

我为了深入理解伪彩色化的原理还是采用查表法，将映射表写了出来，然后根据映射表来进行处理操作。

灰度级-彩色变换法

➤ 算法描述：

循环遍历 bt2 的每个像素，对于每个像素，使用 bt1.GetPixel(i,j) 获取其颜色，并提取颜色的红色（R）、绿色（G）和蓝色（B）分量。使用经典的灰度转换公式得到每个像素的灰度范围，根据 temp 的值的范围，选择不同的伪彩色映射规则，并将对应的颜色赋给 show_bt 中相应位置的像素。

(1)如果 temp 在范围 [0, 63] 内，将像素设置为蓝色渐变。

(2)如果 temp 在范围 [64, 127] 内，将像素设置为蓝色到绿色的渐变

(3)如果 temp 在范围 [128, 191] 内，将像素设置为绿色到红色的渐变

(4)如果 temp 在范围 [192, 255] 内，将像素设置为红色到黄色的渐变。

这些范围和颜色选择是基于经验和视觉效果而确定的。

该算法的**基本思路**就是根据每个像素的灰度值选择相应的颜色进行映射，从而实现将灰度图像转换为伪彩色图像的效果。不同的灰度值范围对应不同的颜色，以增强图像的视觉效果和对比度。

```
1. //灰度级-彩色变换法
2. private Bitmap weicaise_chuli(Bitmap src)
3. {
4.     Bitmap bt2 = new Bitmap(pictureBox_show.Image);
5.     Bitmap bt1 = new Bitmap(pictureBox_show.Image);
6.     Bitmap show_bt = new Bitmap(pictureBox_show.Image);
7.     for (int i = 0; i < bt2.Width; i++)
8.     {
9.         for (int j = 0; j < bt2.Height; j++)
10.        {
11.            Color c = bt1.GetPixel(i, j);
12.            int R = c.R; int G = c.G; int B = c.B;
13.            byte temp = (byte)(B * 0.114 + G * 0.587 + R * 0.299);
14.            if (temp >= 0 && temp <= 63)
15.            {
16.                show_bt.SetPixel(i, j, Color.FromArgb(0, (byte)(4 * temp), (byte)2
17.                55));
18.            }
19.            if (temp >= 64 && temp <= 127)
20.            {
21.                show_bt.SetPixel(i, j, Color.FromArgb(0, 255, (byte)(510 - 4 * tem
22.                p)));
23.            }
24.        }
25.    }
26. }
```

```

22.         if (temp >= 128 && temp <= 191)
23.         {
24.             show_bt.SetPixel(i, j, Color.FromArgb((byte)(4 * temp - 510), 255,
25.             0));
26.         }
27.         if (temp >= 192 && temp <= 255)
28.         {
29.             show_bt.SetPixel(i, j, Color.FromArgb(255, (byte)(1022 - 4 * temp)
30.             , 0));
31.         }
32.     }
33.     return show_bt;
34. }

```

➤ 实现效果:



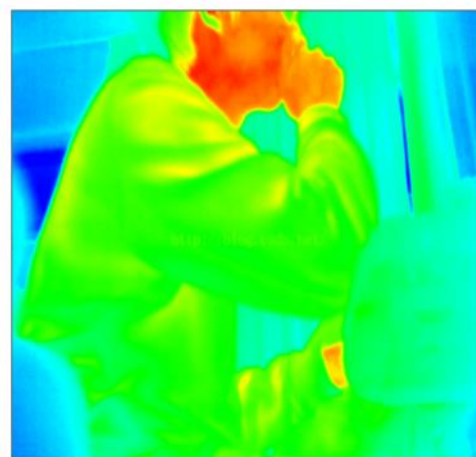
原图



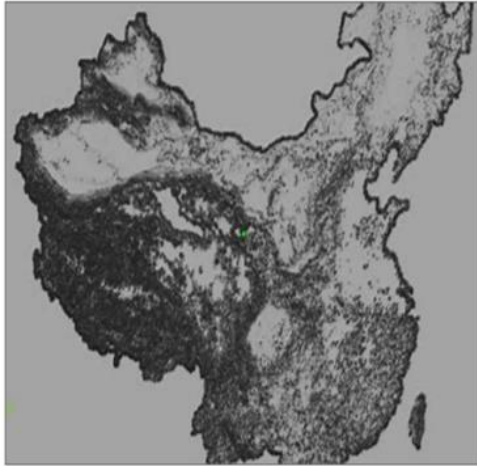
伪彩色变换



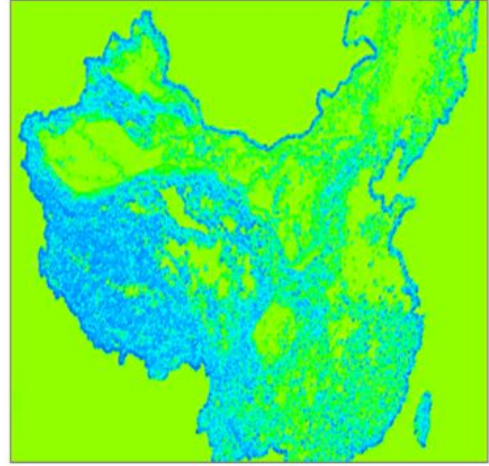
原图



伪彩色变换



原图



伪彩色变换

✚ 灰度分层法:

➤ 算法描述:

根据输入的分层数 `numLevels` 计算每个灰度层的宽度 `levelWidth`，即将灰度级别从 0 到 255 均分成 `numLevels` 个层数，然后，根据每个灰度级别所在的层数和在该层内的偏移量，计算对应的伪彩色。

创建一个颜色映射表 `colorMap`，来存储每个灰度级别对应的伪彩色。遍历 0 到 255 的灰度级别，对每个灰度级别计算其所在的层数 `level` 和层内的灰度偏移量 `offset`。根据层数和偏移量计算对应的伪彩色，其中红色和绿色的值根据层数进行插值，蓝色的值根据偏移量变化。通过遍历图像的每个像素，计算其灰度级别并根据该级别选择对应的伪彩色，并将其赋值给新的图像 `pseudoColorImage`。

```
1. private void btn_GrayLevel_Click(object sender, EventArgs e)
2. {
3.     Bitmap bt1 = new Bitmap(original_image.Image);
4.     int grayLevelValue = trackBar_grayLevel.Value; //灰度分层数
5.     Bitmap change = WeicaiseChuli(bt1, grayLevelValue);
6.     pictureBox_show.Refresh();
7.     pictureBox_show.Image = change;
8. }
9.
10. //灰度分层法
11. private Bitmap WeicaiseChuli(Bitmap src, int numLevels)
12. {
13.     if (numLevels <= 0 || numLevels > 256)
14.     {
```

```

15.         throw new ArgumentException("Invalid number of levels. Must be between 1 a
16.         nd 256.");
17.     }
18.     Bitmap pseudoColorImage = new Bitmap(src.Width, src.Height);
19.
20.     // 计算每个灰度层的宽度
21.     int levelWidth = 256 / numLevels;
22.
23.     // 定义颜色映射表
24.     Color[] colorMap = new Color[256];
25.
26.     // 设置颜色映射表
27.     for (int i = 0; i < 256; i++)
28.     {
29.         // 计算当前灰度级别所在的层数
30.         int level = i / levelWidth;
31.
32.         // 计算当前层内的灰度偏移量
33.         int offset = (levelWidth / (numLevels - 1)) * (i % levelWidth);
34.
35.         // 根据层数和偏移量计算伪彩色
36.         byte red = (byte)(level * (255 / (numLevels - 1)));
37.         byte green = (byte)((numLevels - 1 - level) * (255 / (numLevels - 1)));
38.         byte blue = (byte)offset;
39.
40.         colorMap[i] = Color.FromArgb(red, green, blue);
41.     }
42.
43.     // 遍历灰度图像的每个像素
44.     for (int x = 0; x < src.Width; x++)
45.     {
46.         for (int y = 0; y < src.Height; y++)
47.         {
48.             Color pixelColor = src.GetPixel(x, y);
49.             int grayLevel = (int)(pixelColor.R * 0.299 + pixelColor.G * 0.587 + pi
50.             xelColor.B * 0.114);
51.
52.             // 根据灰度级别选择对应的伪彩色
53.             pseudoColorImage.SetPixel(x, y, colorMap[grayLevel]);
54.         }
55.     }
56.     return pseudoColorImage;
57. }
58.

```

➤ 实现效果:

Gui 界面设计:

Form1

打开

还原

保存

高斯噪声

均值

标准差

确定

椒盐噪声

确定

随机噪声

确定

图像平滑

四邻域平均

确定

HSI

灰度化

图像缩放

亮度调节

确定

对比度调节

确定

图像旋转

确定

通道提取

红

确定

二值化阈值

确定

形态学变换

膨胀

确定

SE大小

SE模板

正方形

类型

二值图

图像边缘提取

水平垂直差分法

确定

边缘叠加原图（锐化）

确定

伪彩色处理

彩虹色带映射

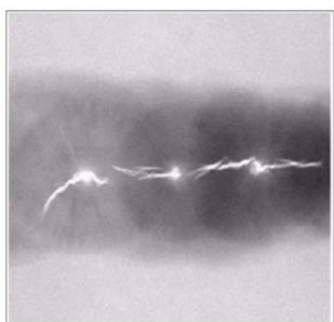
确定

灰度分层法

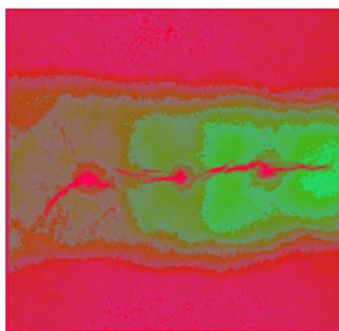
34

确定

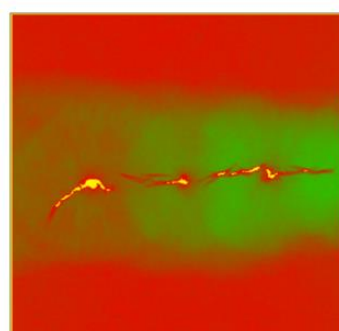
灰度分层法可以用于焊点问题检测，比如给焊点有问题的图片拍一张照片，下面的原灰度图像就是焊点有问题，我们就可以通过不同的灰度分层来识别，从而突出有缺陷的焊点，从而简化我们的工作，降低误差率。例如下图中我们发现当灰度分层数为 81 时，能够较好的鉴别出有误的焊点。



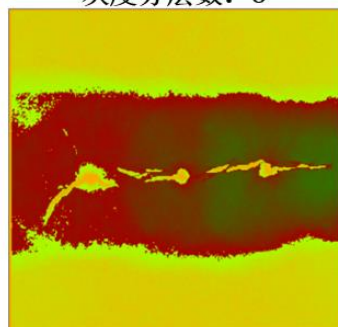
原图



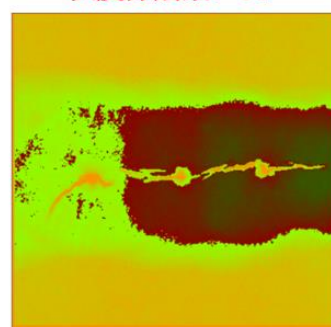
灰度分层数: 8



灰度分层数: 81

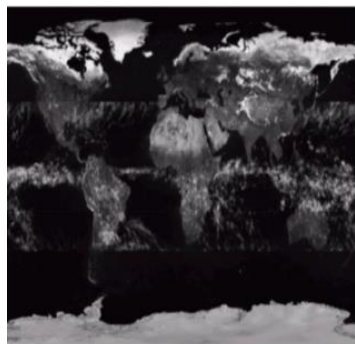


灰度分层数: 165

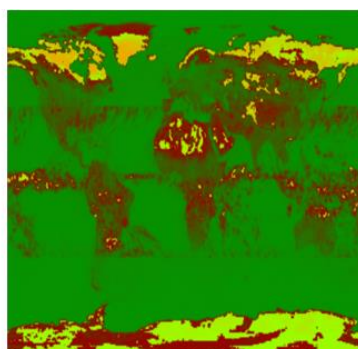


灰度分层数: 143

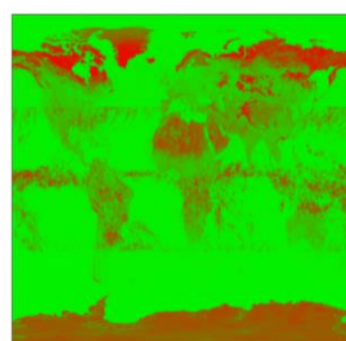
又比如我们可以用颜色来突出**降雨水平**，图像的强度值直接与降雨相对，相较于不太清楚的原图，显然经过分层之后的图像更直观，更清楚，我们只需要制定一个标准，比如颜色越黄降雨量越多，即可完成降雨量的可视化判断。



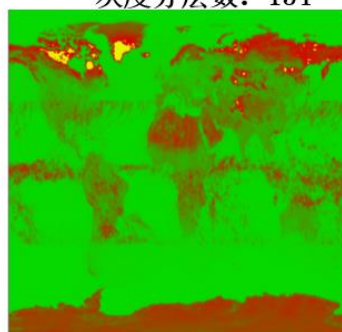
原图



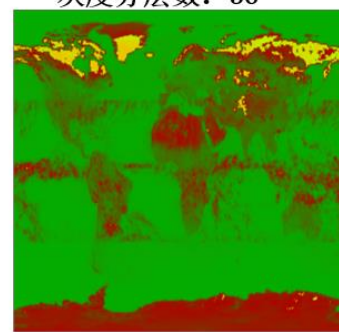
灰度分层数: 154



灰度分层数: 36

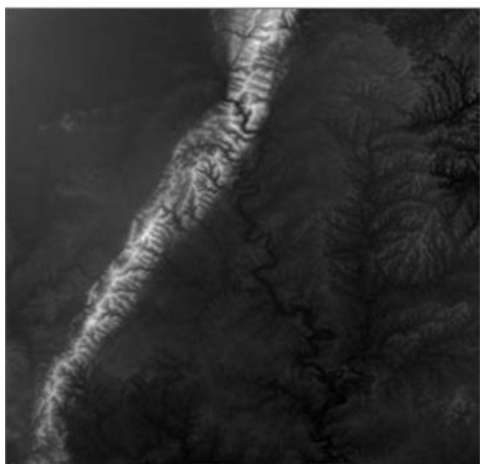


灰度分层数: 224

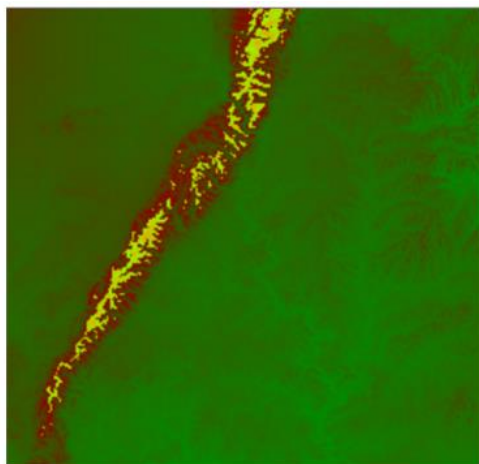


灰度分层数: 91

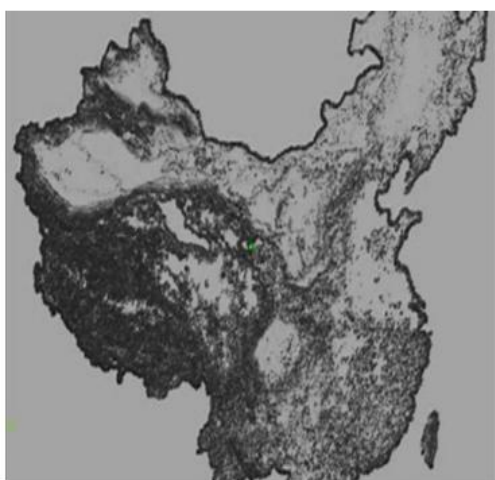
又比如识别**地形海拔**，同样能够很直观的突出其特征：



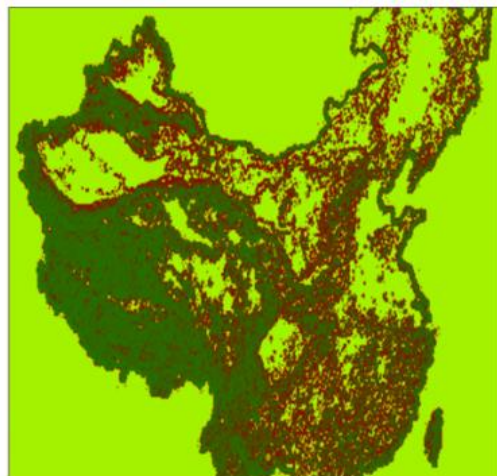
原图



灰度分层数: 143



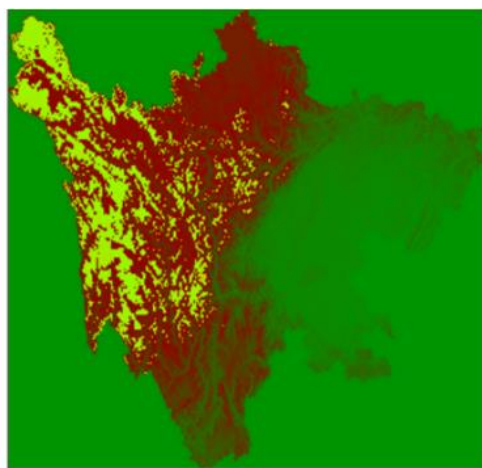
原图



灰度分层数: 151



原图



灰度分层数: 149

五、实验结果及分析(包括心得体会, 本部分为重点, 不能抄袭复制)

➤ 完成情况:

完成了实验全部的基本要求, 最终的结果基本达到了我的预期

➤ 实验心得

在本次实验中, 我使用了色带映射法、灰度级-彩色变换法和自定义的灰度分层法对灰度图像进行了伪彩色处理。并对它们的优缺点进行了整理, 以下是详细描述:

(1) 色带映射法:

色带映射法是一种常见的伪彩色化方法, 它通过将灰度级别与特定的色带进行映射, 将灰度图像转换为彩色图像。在本次实验中, 我选择了**彩虹色带**和**铁红色带**两种经典色带进行映射。色带映射法的优点是简单易实现且具有直观的效果, 能够使图像呈现出丰富的色彩。然而, 它的缺点是色带的选择和灰度级别与色彩之间的映射关系可能对图像结果产生较大影响, 且不同色带对图像的表达效果有所差异。

(2) 灰度级-彩色变换法:

灰度级-彩色变换法是一种基于灰度级别与彩色之间的映射关系进行伪彩色处理的方法。在实验中, 我根据灰度级别选择了不同的彩色值, 例如低灰度级别对应较暗的颜色, 高灰度级别对应较亮的颜色。这种方法的优点是可以根据需求自定义灰度级别和对应的彩色值, 使图像具有一定的表达能力和艺术效果。然而, 缺点是对灰度级别和彩色之间的映射关系要求较高, 需要根据具体情况进行调整, 否则可能导致图像的失真或信息丢失。

(3) 自定义的灰度分层法:

自定义的灰度分层法是我根据实验要求自行设计的一种方法。该方法通过将灰度级别均匀分层, 并为每个灰度级别分配一个伪彩色, 将灰度图像转换为伪彩色图像。优点是简单易实现, 通过调整分层数可以控制伪彩色的丰富程度, 适用于突出或区分不同的图像特征或灰度值范围。缺点是分层算法的效果可能相对较简单, 不同灰度级别之间的颜色变化相对较少。

在实验过程中, 我也查阅资料了解到更多关于灰度图像伪彩色化的应用案例, 比如经常用于我们的医学领域, 卫星云图以及地形图等等, 因此在实验过程中我也主动去尝试用自己编写的算法对这些图像进行处理, 看是否能达到我的预期或是接近标准图。通过这种实践, 也让我更深入的理

解了为什么对灰度图像会有一个伪彩色化的过程。

通过本次实验，我发现不同的伪彩色处理方法具有各自的优缺点。色带映射法简单易用，但对色带选择和映射关系的敏感性较高；灰度级-彩色变换法具有较高的自定义性，但需要根据具体情况进行调整；自定义的灰度分层法简单有效，但可能对颜色变化的表达能力有一定限制。因此，在实际应用中，可以根据需求和图像特点选择合适的伪彩色处理方法，或者结合不同方法进行实验和调整，以获得最佳的伪彩色效果。