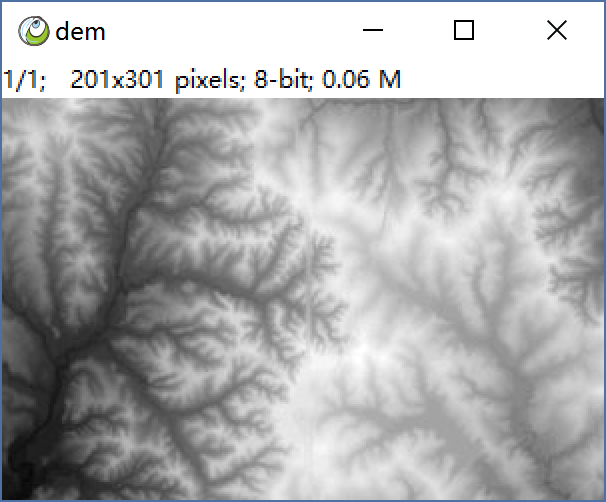
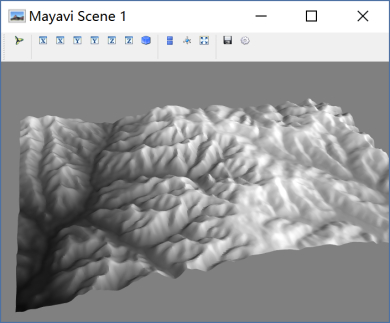
1. **高程分析**

前面两章我们讨论了图像的像素运算与滤波器运算，像素运算是对原图每一个像素进行一个数学映射，得到新的图像。而滤波器的作用是通过卷积对图像上某些信息的抑制或增强。凸显一些信息，而弱化其他的。而以上方法都是针对整张图像进行的，而本章我们将讨论一些基于高程的算法，这些算法能在图像上得到特定的点或线，这些点或线往往具有地理与水文意义。

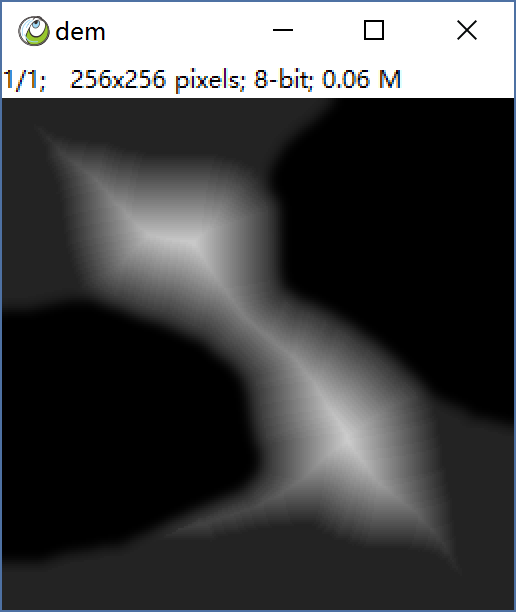
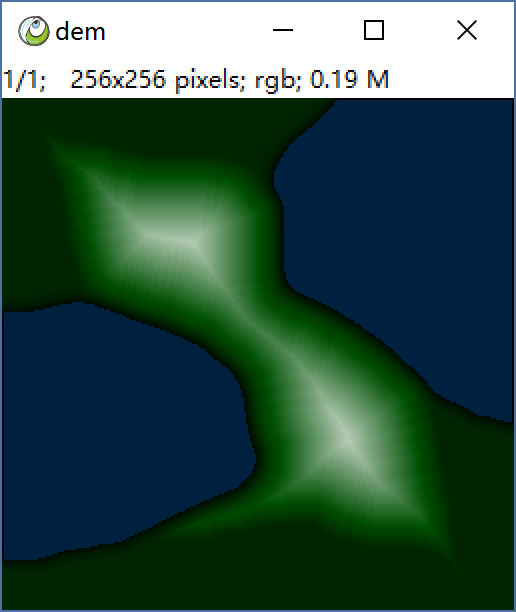
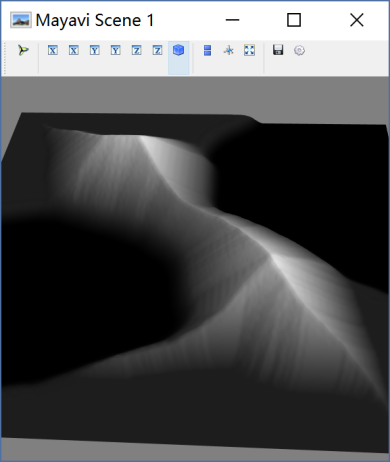
**DEM数字高程模型**

此前我们已经认识到图像是由排列在规则网格上的像素构成，并且像素的亮度可以映射为高程，而本节中我们将进一步加强这种概念，（Digital Elevation Model）简称DEM，意思是是按照一定的尺度采样，将地表的高度数字化，以亮度的方式存储在图像中，其实它与普通的灰度图像别无二致，只是它来源于现实地表，另外由于地表是起伏的，但很少突变（除非峡谷，断裂带等）因而视觉上这类图像是明暗变化，但一般没有明显的边缘。

DEM 三维表面重建 丘陵地带

**观察描述：**DEM图像中黑色的地方代表凹地，河谷，而亮色部位代表丘陵，山峰。

DEM 山丘湖泊 三维地表

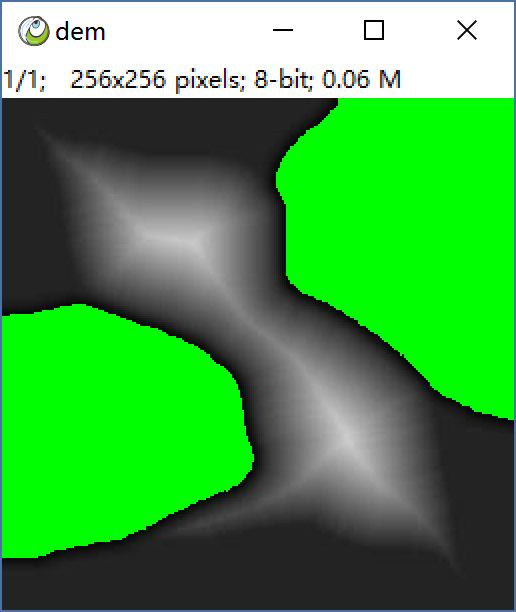
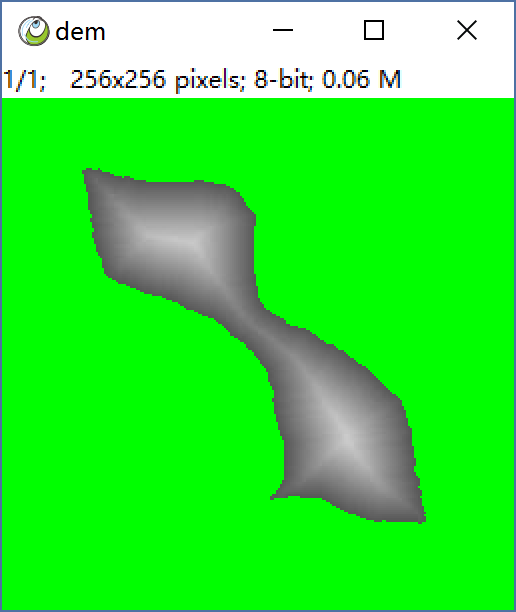
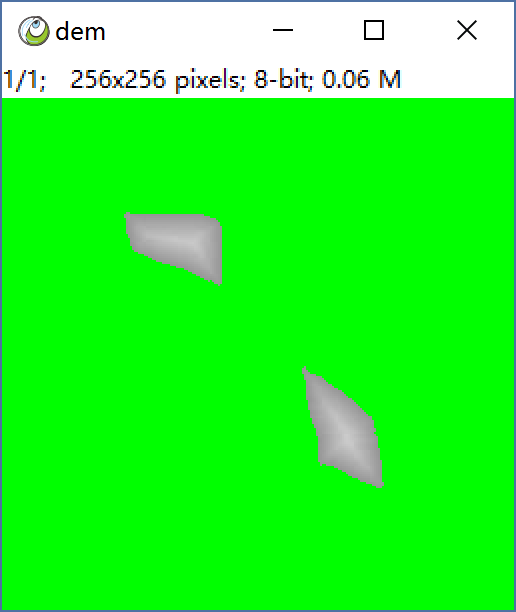
**观察描述：**观察上图的DEM，两座山丘，中间的亮线是山脊，山脊两侧是两个湖泊。这个DEM并不是某处的天然地形，而是特意制作的，目的是为了更好的展示问题和算法原理。

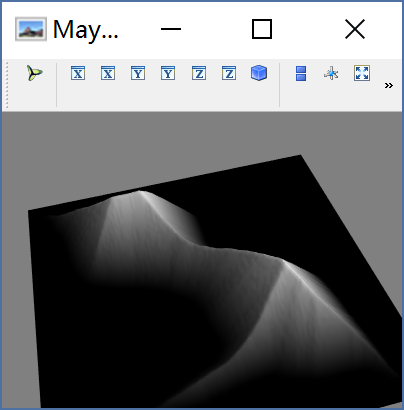
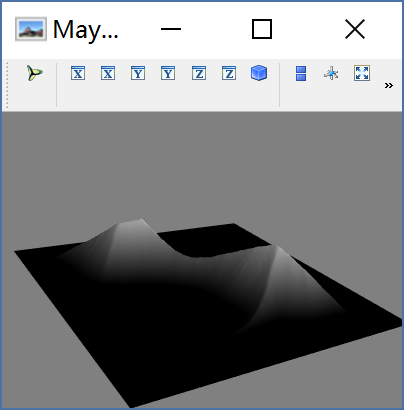
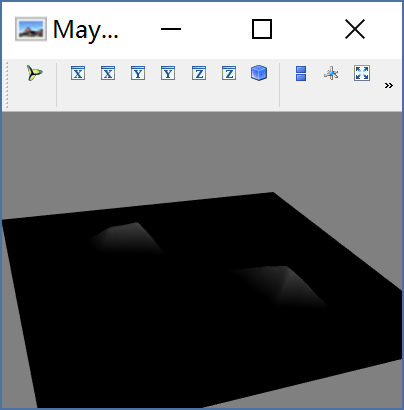
**阈值化**

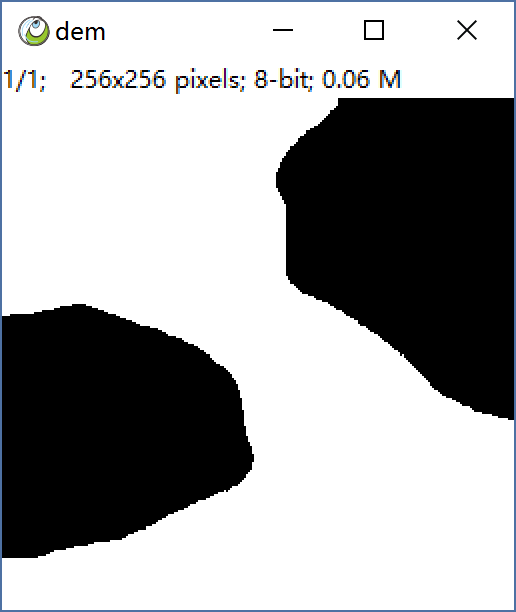
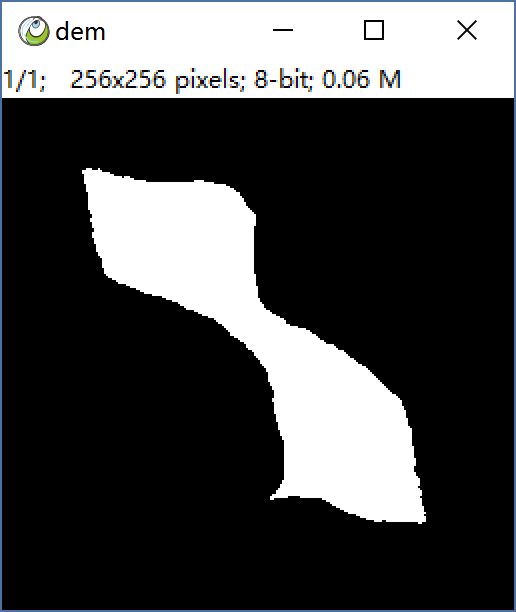
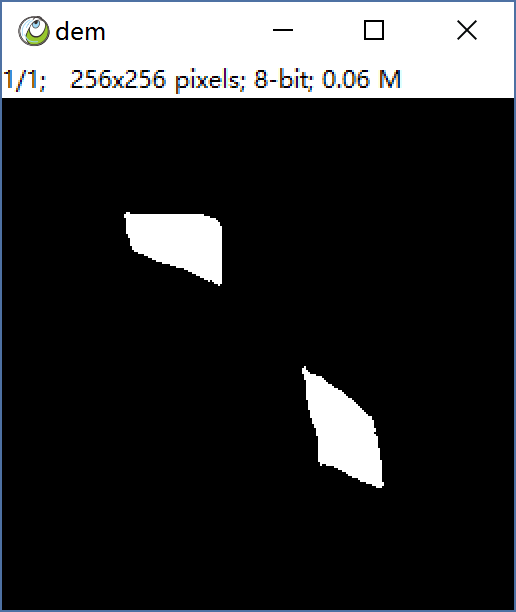
在像素运算以章我们已经了解过阈值，但并没有正式介绍，在此我们正式介绍阈值的概念。阈值即对高度进行一个非此即彼的判别，大于则属于，小于则不属于，正可谓成王败寇。用DEM理解阈值再好不过了。

**$ Image > Adjust > Threshold**

**$ IBook > Chapter5 Elevation-Analysis > Show Do Threshold**

阈值thr = 20 阈值thr = 120 阈值 thr = 200

**观察描述：**阈值的结果是二值的，我们可以假象这是一个大洪水渐渐淹没山丘的过程，绿色部分是水覆盖的区域，随着水的上涨，两座山之间的鞍部也被浸没，最后成为了两个独立的小岛（注意这里是）

**计算方法**

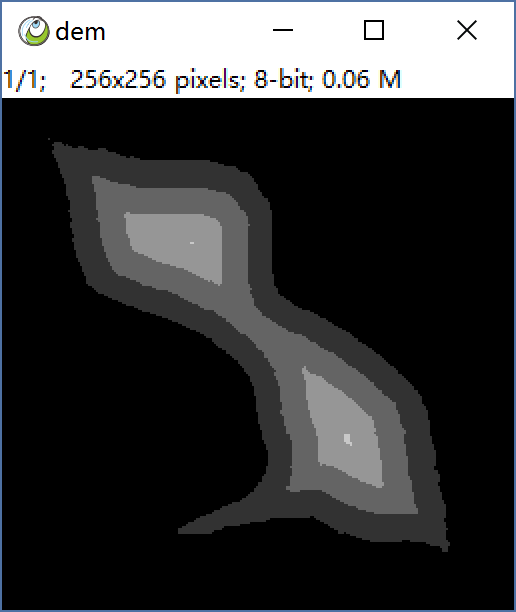
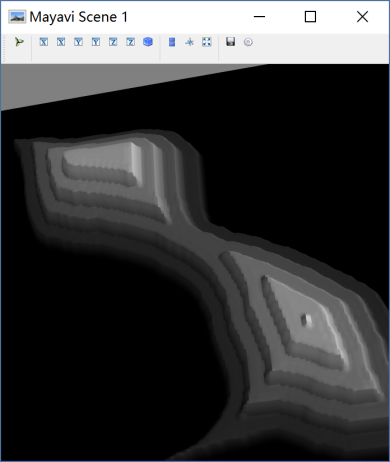
阈值的计算方法非常简单，一个生活中的例子就是对百分制的成绩进行是否几个的评判，大于等于60分的就几个，小于的不及格，这就是阈值。

**等级化**

前面我们已经介绍了阈值化方法，阈值是对亮度是否高于某个值进行判断，而这里我们介绍等级化，等级化的意思是将原本渐变的亮度聚类为几个等，进而可以得到等值线，等值线上的点，具有相同的高程。

**$ Process > Hydrology > Find Isoline**

**$ IBook > Chapter5 Elevation-Analysis > Show Find Isoline**

等级化 三维效果 等值线叠加

**观察描述：**等级化之后，原本平滑的图片看到了比较明显的边缘，这是因为像素等级化之后，差异变明显了，而对应的三维模型可以看到，类似人工的梯田，对等级图像做一个拉普拉斯滤波，就得到了变化的部分，也就是等值线，等值线在地理上是一个非常重要的概念。

**计算方法**

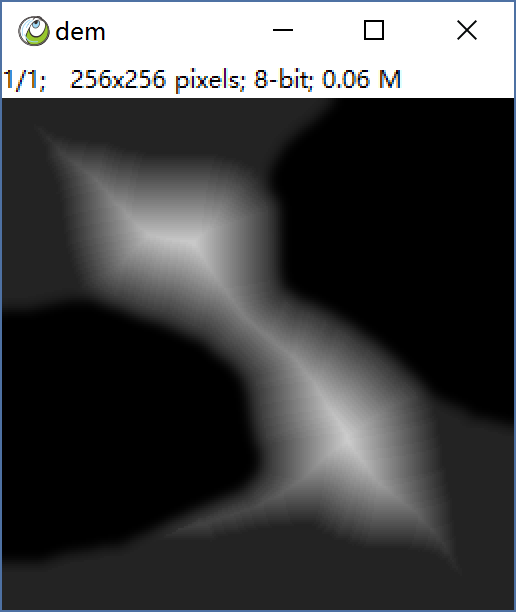
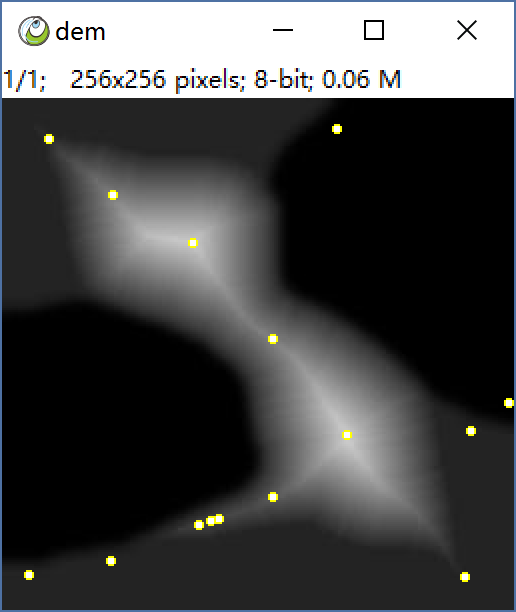
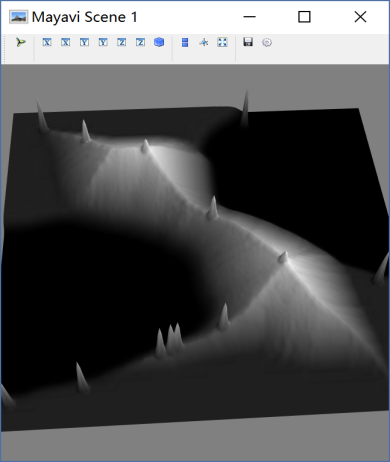
其实某种意义上阈值化可以理解为等级化的特例，即只有两个等级，等级化是多个等级，比如对百分制的成绩分优，良，及格，不及格等，我们知道及格线是60分，而70-90之间都属于良，90分以上是优秀，只是在地理中往往我们的等值线是等间距的，除了特殊强调的重要标识。

**局部最大值与制高点**

**$ Process > Hydrology > Find Maximum**

**$ IBook > Chapter5 Elevation-Analysis > Show Find Maximum**

局部最大值顾名思义，不同于全局，局部最大值只是局部最大，可能存在多个。

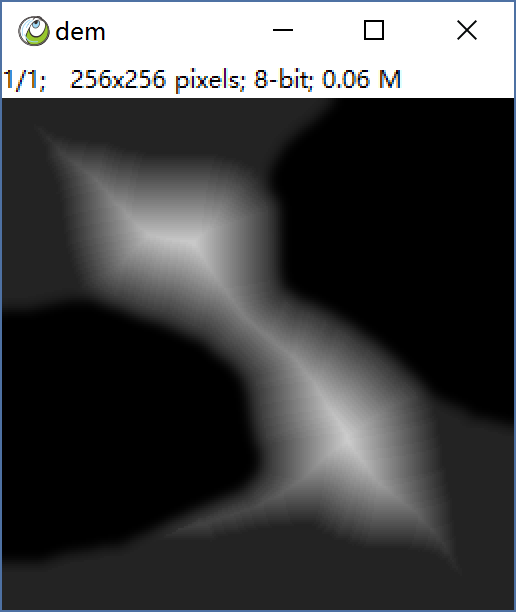
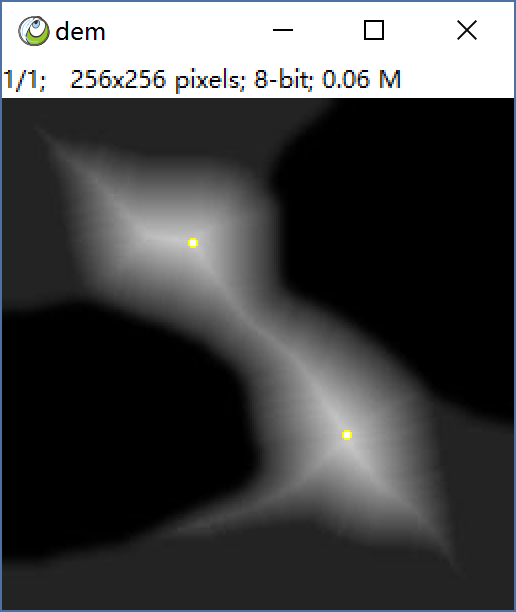
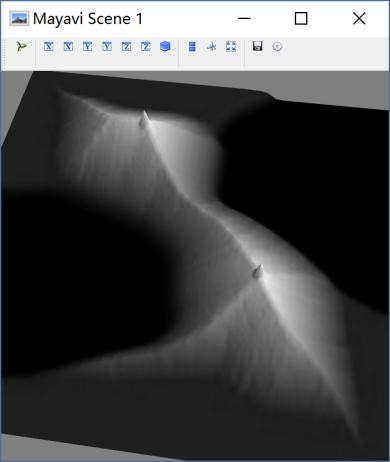
  

两个山丘 制高点检测 三维效果

**思考：**为什么出现了多个我们预期之外的点？因为局部最大值收噪声影响敏感，任何的像素波动都有可能会使其超过周围，因而得到了很多个，甚至是因为图像存储过程中的压缩损失，比如jpg格式。

**计算方法：**在一个3x3小范围内，如果中心像素不小于周围任何一个，那么它就是一个局部最大值。

制高点在战略上是有非常重要的意义的，占据制高点，往往意味着掌握了附近区域的开火权，然而刚才的局部最大值并不能很好的代表制高点，原因我们也已经分析了，下面我们对其进行改进。

两个山丘 制高点检测 三维效果

**思考：**如何可以称之为制高点？首先在局部最大，并且在一定范围内可以火力覆盖，并且不能处于附近一个更高的制高点的火力覆盖范围内，满足这样的要求，才具备战略意义。

**计算方法**

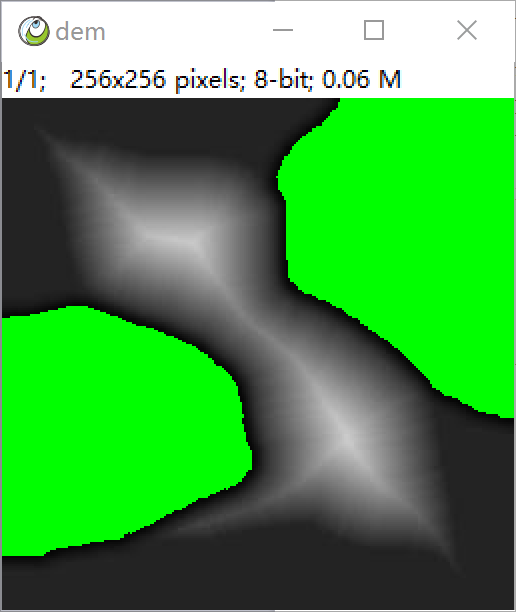
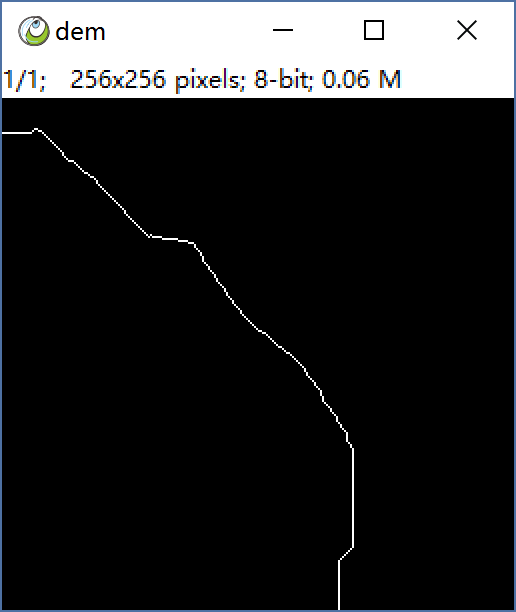
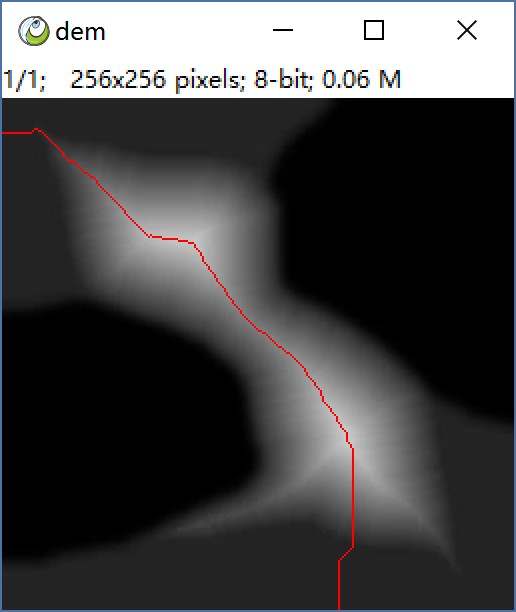
其实制高点计算是在上一步局部最大值计算的基础上进行了筛选，一般来说，从一个山头到达另一个山头都需要先下山，再上山，如果两个高点，之间没有足够凹的山谷隔离，那么这两个点中只能保留较高的，试想更高点的人直冲下来，甚至不需要爬山，就可以占领你的山头，那自然是不安全的。具体实现其实并不简单，但读者只需要领悟大概意思。

**分水岭和山脊线**

上面我们讨论了关于特征点的提取，接下来我们来看分水岭问题，分水岭是一个水文学上的概念，分水岭两侧的降水汇集到不同的流域。

**$ Process > Hydrology > Find Watershed**

**$ IBook > Chapter5 Elevation-Analysis > Show Find Watershed**

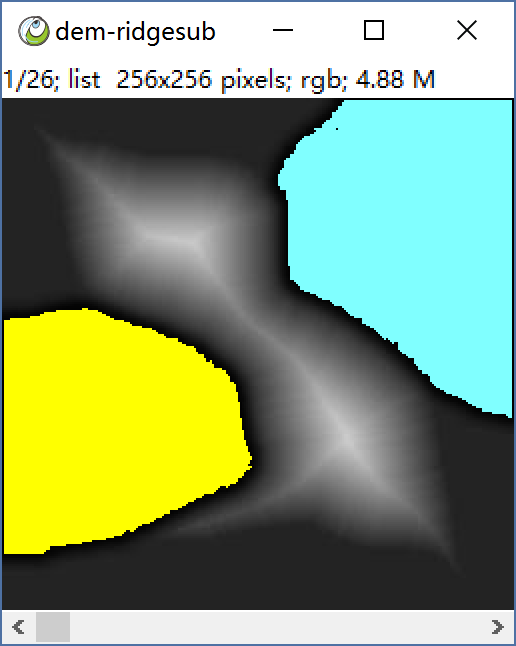
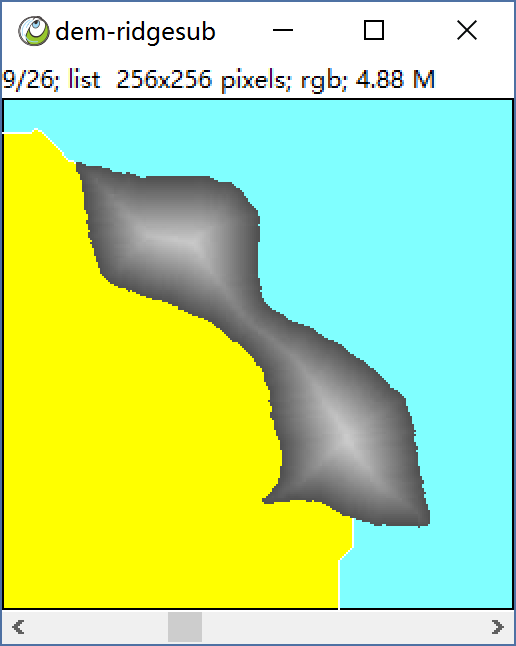
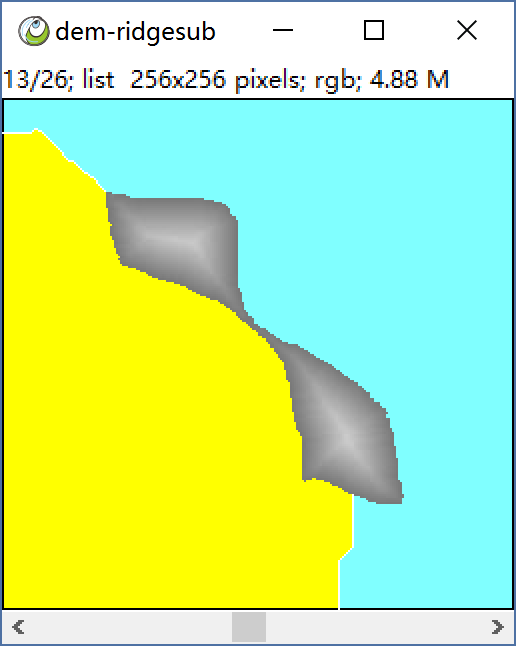
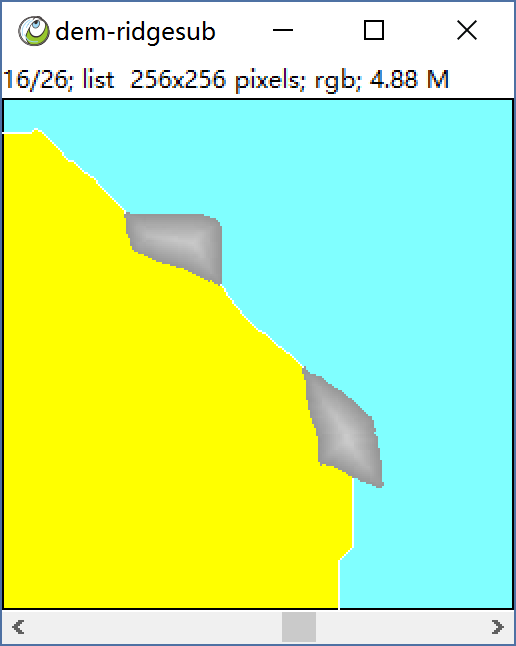
集水区 分水岭 叠加效果

观察描述：当叠加分水岭到原图之后，可以看到分水岭算法基本是沿着山脊线的，这容易理解，山脊两侧的降水会流向两个不同的集水区。

**计算方法**

分水岭需要确定两个集水区，这里把最初的两个湖泊用两种不同颜色标记，然后模拟水位渐渐上升的过程，在两个水域交汇的地方停止，最后的分割线就是分水岭。

**$ IBook > Chapter5 Elevation-Analysis > Show Watershed Sub**

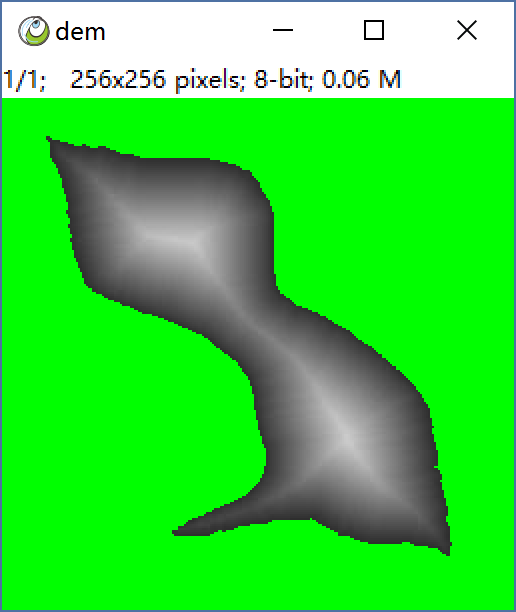
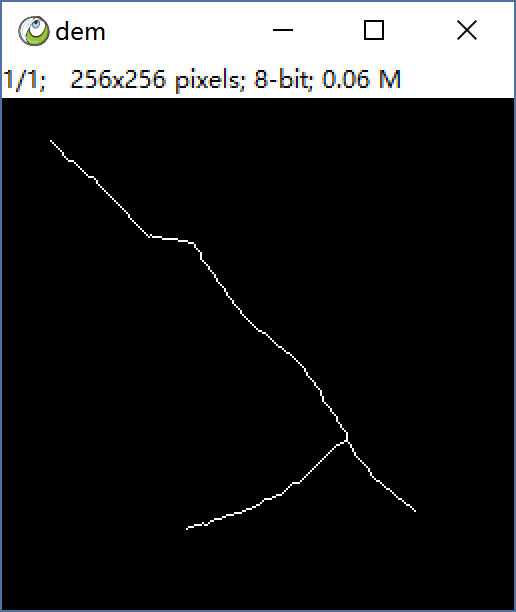
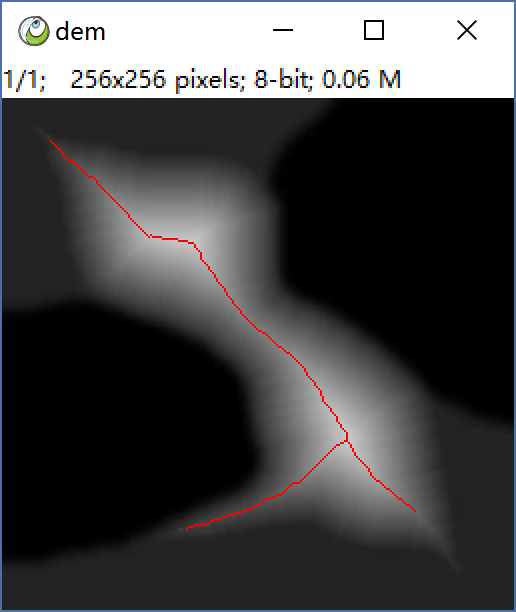
确定集水区 水位上升 淹没鞍部 继续上升

**观察描述：**分水岭需要确定初始的集水区，是指是模拟两个集水区上涨，最终完全淹没山丘的过程，而不同流域的最终界限就是分水岭。

以上是计算分水岭，另一个与之相似的问题是提取山脊线，分水岭的计算需要独立的集水区，最终结果是集水区的分割线，而山脊线略有不同，我们先来看图。

**$ Process > Hydrology > Find Ridge**

**$ IBook > Chapter5 Elevation-Analysis > Show Find Ridge**

水平面 山脊线 叠加效果

**观察描述：**与分水岭不同，山脊线可以形成孤立的线，线段基本是沿着我们地理意义上的山脊行走，除了主脉，也有余脉。

**计算方法**

试想如果用分水岭方法计算山脊线，相同的集水区最终将不会形成分割线，所以无法得到理想结果，这里我们亦然采用涨水发，但是只用一个集水区，并且制定新的终止规则。

1. 不能淹没终端像素
2. 不能将原本相联的区域分成两个孤岛。

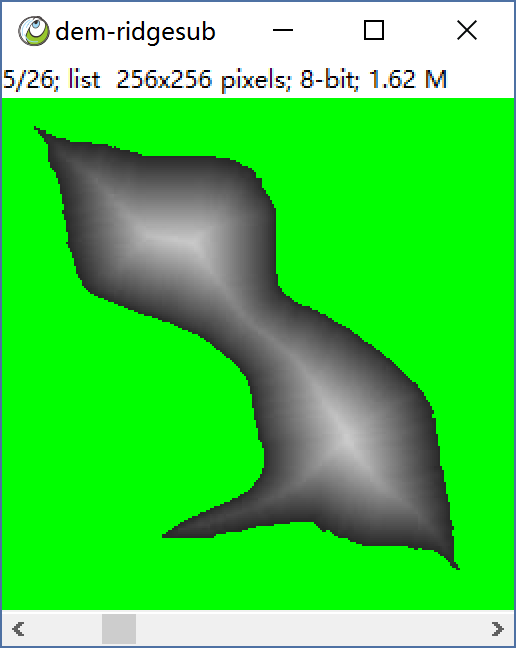
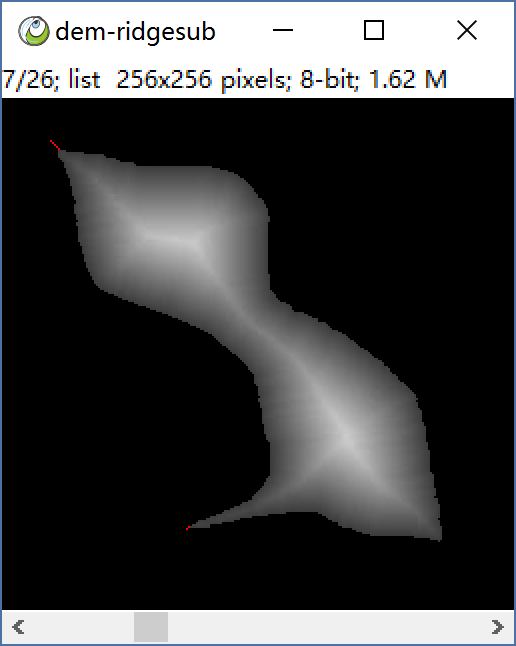
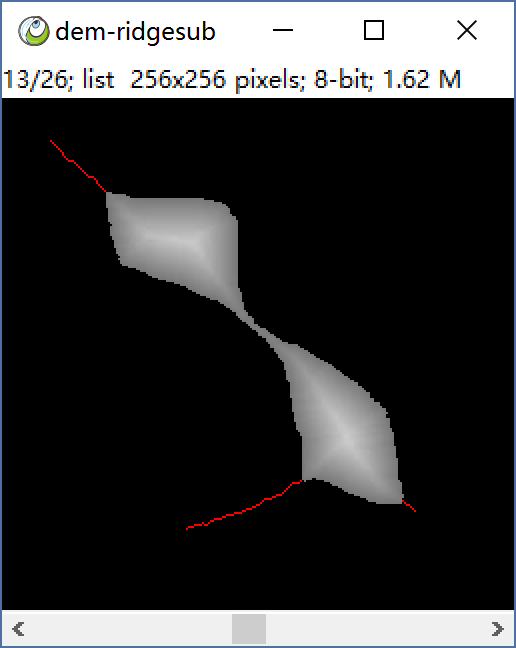
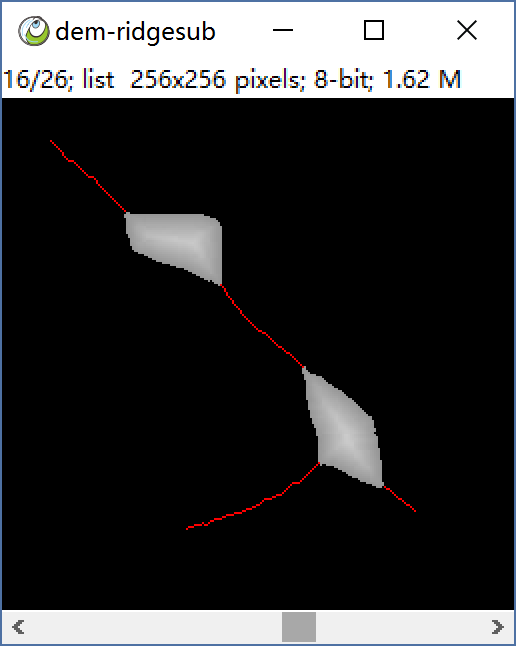
判断上述原则是否成立，可以简单的用周围的3x3像素进行判别。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | 1 | 2 | 3 | | 8 | 0 | 4 | | 7 | 6 | 5 | | |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | |

编号方式 可以去除 可以去除 末端原则 末端原则 切断原则 切断原则

**观察描述：**我们依旧采用涨水法，只是涨水过程中来确定当前的点是否可以淹没，哪些不能。与分水岭不同，山脊线采用区域自我收敛过程中的拓扑检查来实现提取。上图绿色的代表可以去除，红色的由于成为末端，或者导致切断，因而不可去除，我们看下面的例子来体验这种规则是如何发挥作用的。以上情况只是举了几个具体的例子，实际一共8个邻居，每个格子有两个状态，理论上一共有2^8=256中情况，如果我们对邻居进行编号，那么判别可以通过查表来完成。

**$ IBook > Chapter5 Elevation-Analysis > Show Find Ridge Sub**

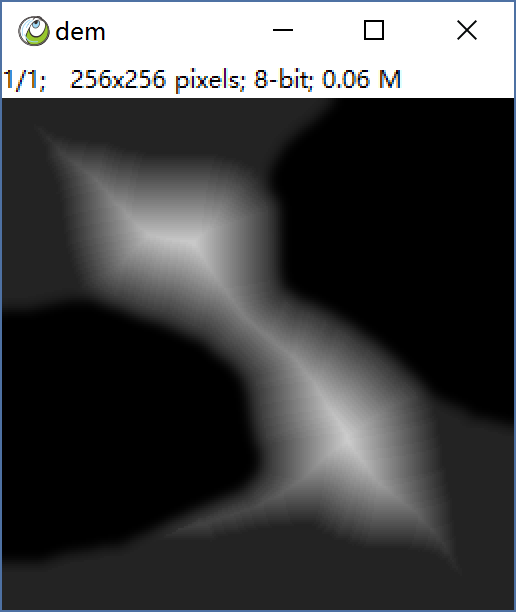
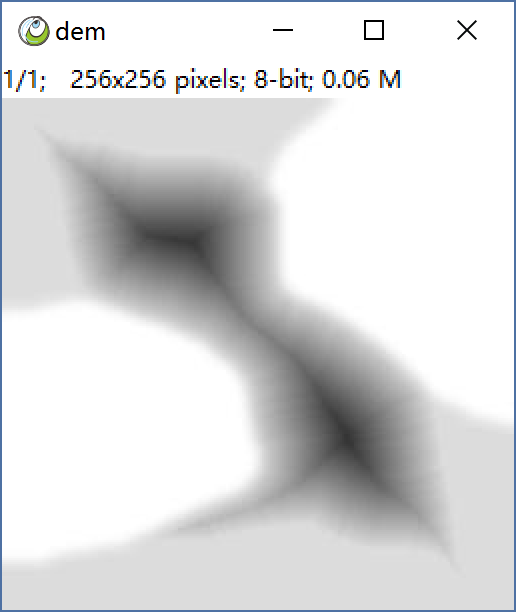
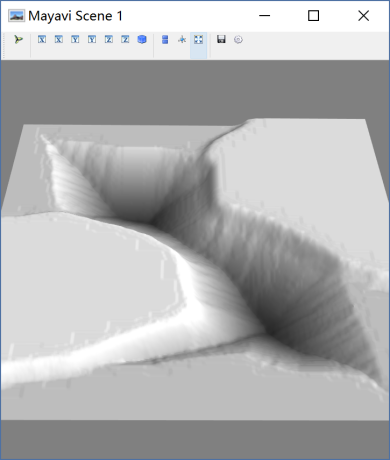
开始漫水 尖端保留原则 不切断原则 渐渐成形

**观察描述：**涨水过程中，山脊末端是尖锐的，当3x3像素内只剩下一个邻居时，就不能再去除了，同样随着水位上涨，鞍部率先被淹没，但由于鞍部的淹没会让物体分成两个孤立部分，这样就违背不切断原则，我们看到，在这种方法下，得到了比较理想的山脊线。

**注：**在编写本书时，查找OpenCV，scikit-image等开源库，并没有发现一个很好的山脊线提取方法，因此以上介绍的山脊线提取是作者本人编写的，以上介绍的各种算法都可以从感性角度去理解，但真正编程实现，并不容易，你需要处理很多情况，但算法实现不属于本书讨论范畴，有兴趣的可以查阅相关资料，或阅读ImagePy源码。

**高程分析中的对偶思想**

高和低是一个相对的概念，以上我们了解了如何提取制高点，分水岭，山脊线，事实上我们只要将图像求反，就可以用相同的方法求积水点（洼地），河谷线。

原图 求反 三维地表

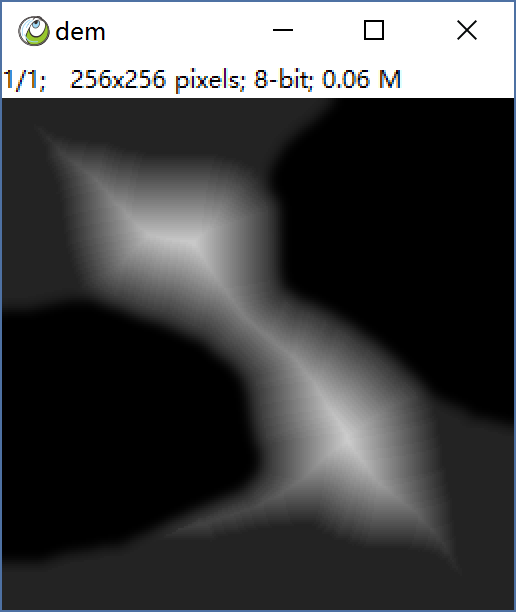
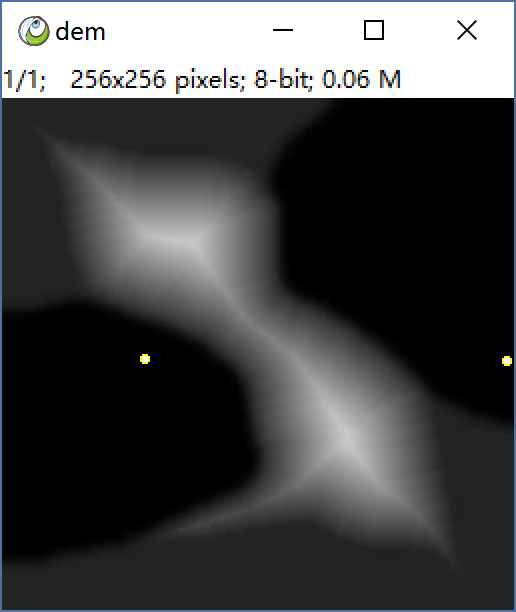
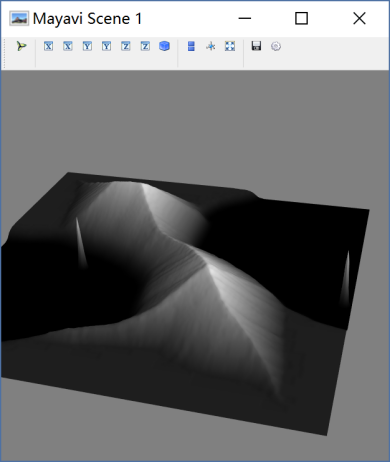
**观察描述：**我们看到原本两个山峰现在编程了洼地，而湖泊编程了高原。

**制高点与积水点**

制高点是对周围具有一定控制力的点，而积水点是附近一定区域内最低的点，我们的例子中两个积水点会出现在湖泊中。

**$ Process > Hydrology > Find Minimum**

**$ IBook > Chapter5 Elevation-Analysis > Show Find Minimum**

DEM 积水点 三维地表

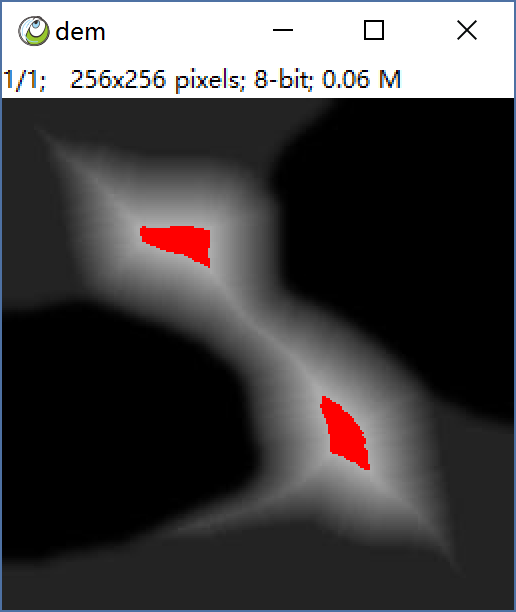
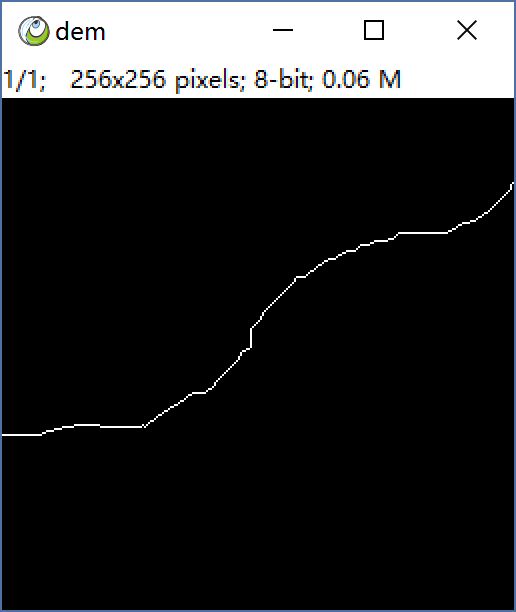
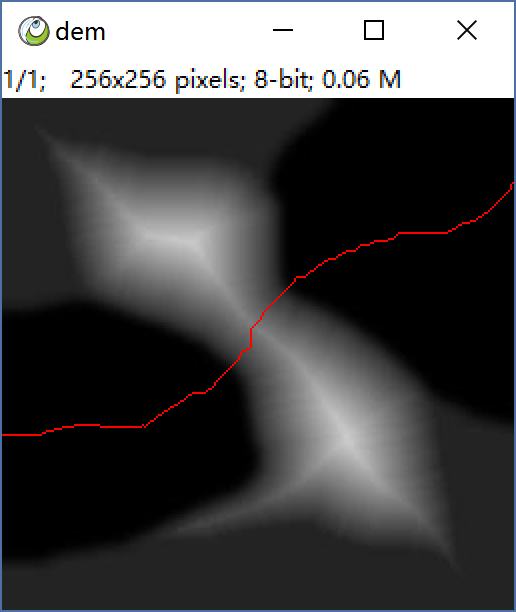
**观察描述：**如我们想象，积水点出现在两个湖泊内。

**山脊与河谷**

用两个山峰作为种子，对图像求分水岭。（实际是在反向图像上完成的）

**$ Process > Hydrology > Find Watershed**

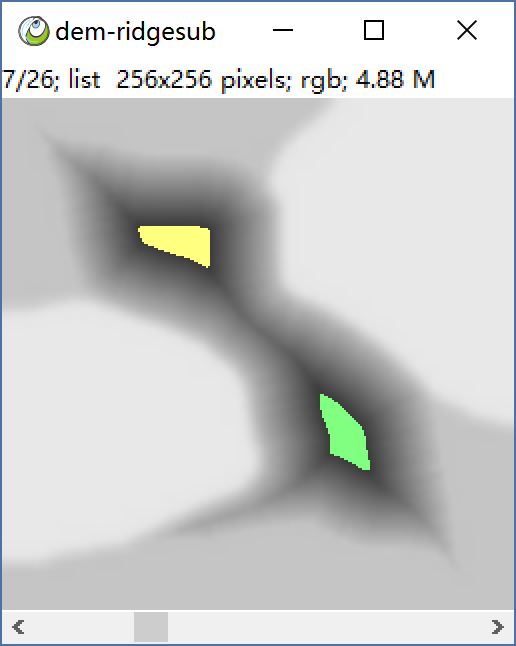
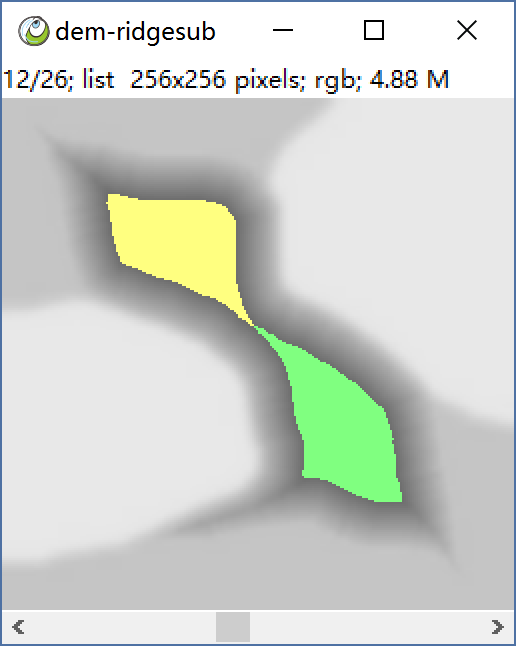
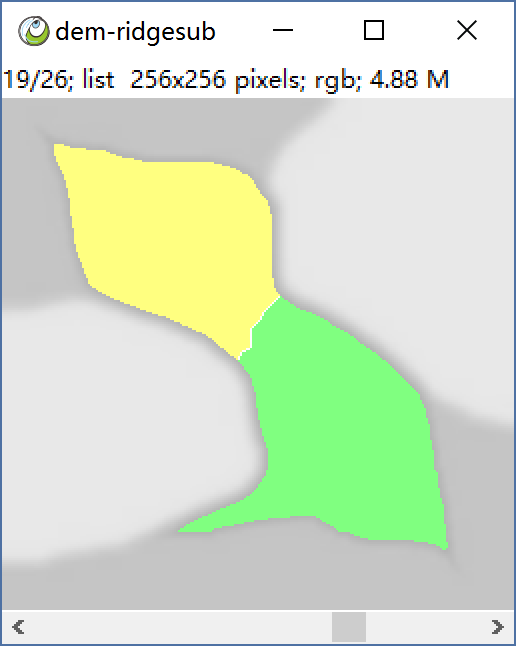
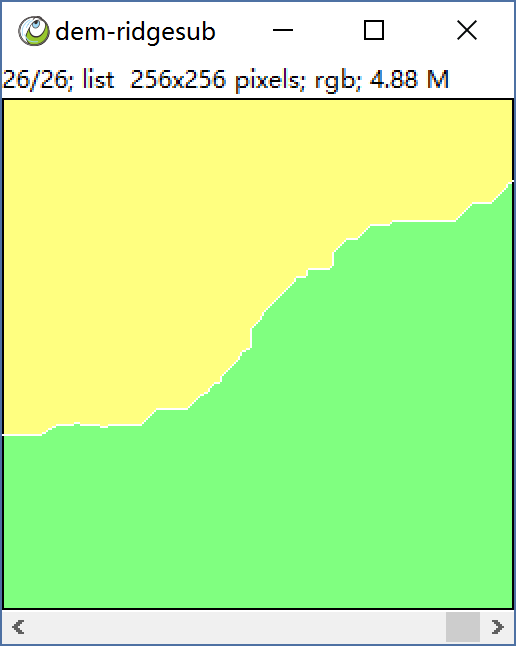
**$ IBook > Chapter5 Elevation-Analysis > Show Find Watershed Inv**

山丘做集水区 分水岭 叠加效果

**观察描述：**用山丘的反向做分水岭，可想而知，鞍部成为了山脊，而得到的分水岭线很好的将两座山分开。

**$ IBook > Chapter5 Elevation-Analysis > Show Find Watershed Inv Sub**

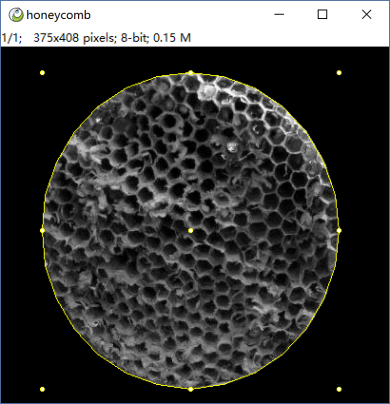
   

集水区 在鞍部交汇 继续涨水 完成灌溉

**观察描述：**水汇集在两个洼地，渐渐上涨，直到在鞍部汇集，继续上涨，汇集线随之延伸，直到水漫上高原，完成整个过程，每个像素都被染色。

**一个图像分割的例子**

上面的例子我们看到，分水岭可以把山沿着山脊分开，也可以从鞍部把两座山峰分开，事实上分水岭是图像分割中的一个经典算法，简单而有效。选取种子点，然后模拟涨水，就可以得到分割结果，结合本章的知识，我们来看一个综合性问题，对一个蜂巢进行分割。

蜂巢照片 转灰度 构造选取，清除无关区域

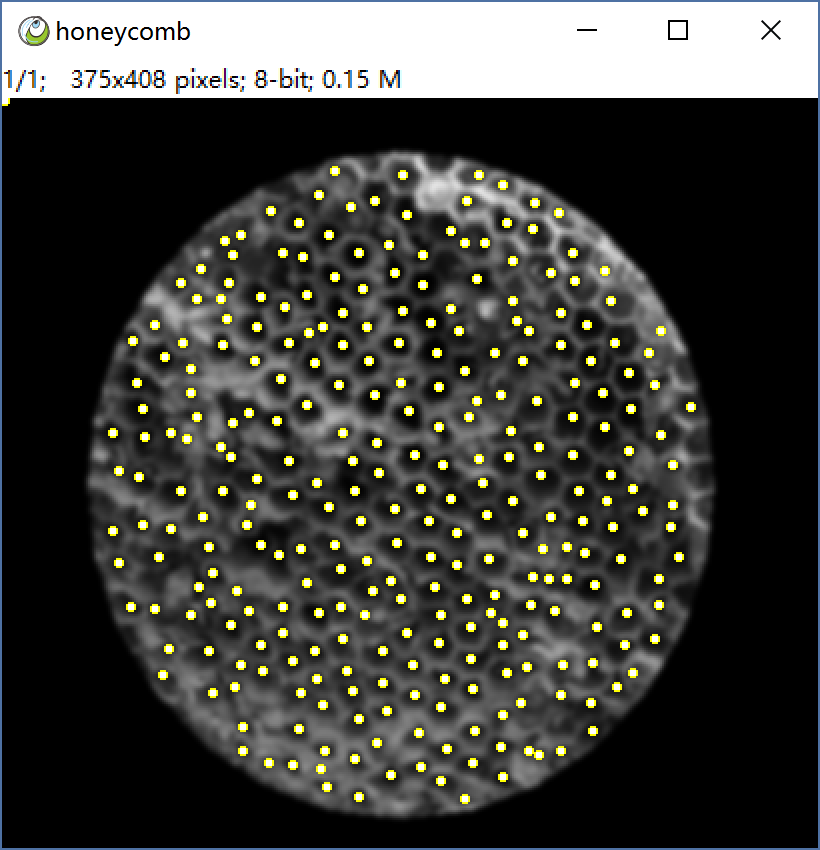
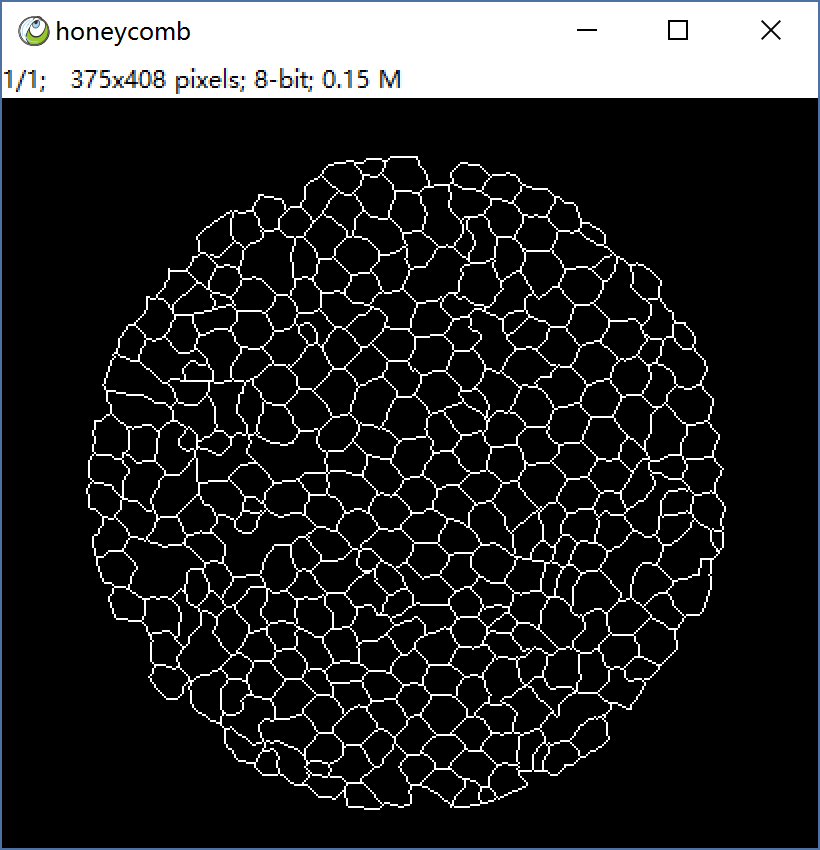
**观察描述：**对蜂巢进行拍照，尽可能正对相机，我们不需要色彩信息，因而转成灰度图像，之后构造椭圆ROI，将外部区域清除为黑色。

**分析**

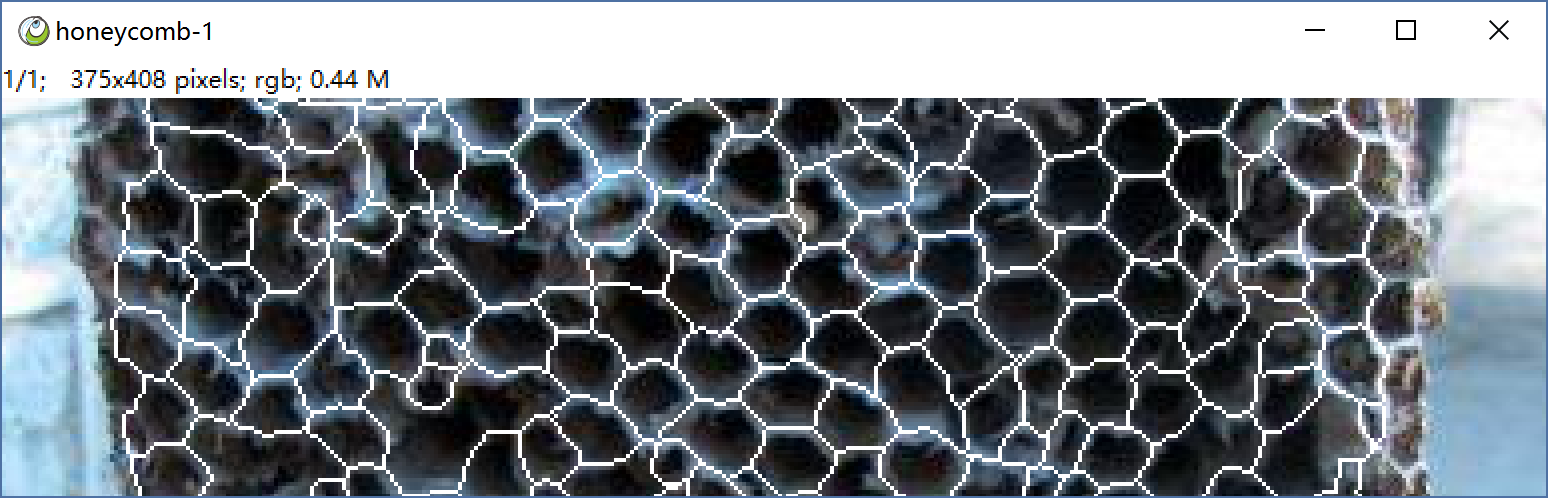
蜂巢的特点是每个孔大致为六边形，孔内亮度比六边形轮廓更暗，结合本章所学内容，我们用集水点检测，可以在每个孔内得到一个集水点，进而用集水点对整张图进行分水岭检测，得到结果。由于图像存在很多锐利的部分，对提取集水点造成干扰，因而首先对图像做了一个sigma为2的高斯滤波。

**$ Process > Hydrology > Find Maximum**

**$ Process > Hydrology > Watershed With ROI**

检测集水点 分水岭



叠加回原图

**观察描述：**平滑后，集水点基本上出现在每个孔的内部，用集水点进行分水岭计算，得到结果。为了更直观的查看分割的精准度，我们将分水岭叠加回原图，可以看到效果非常理想。

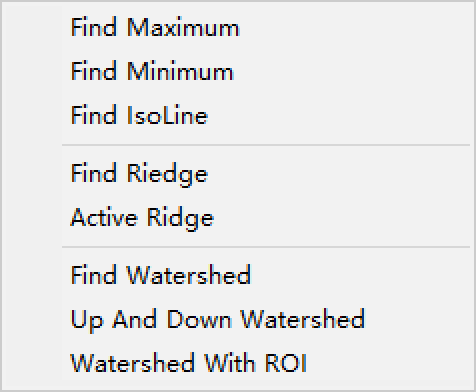
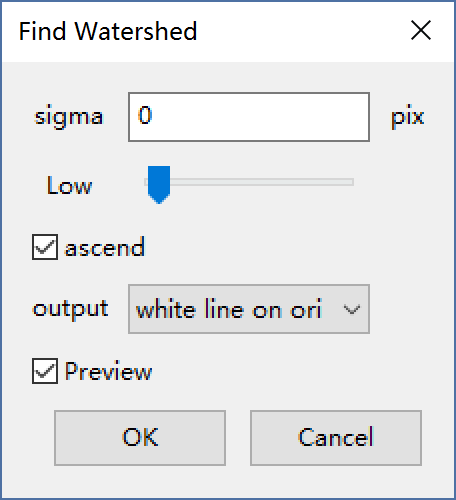
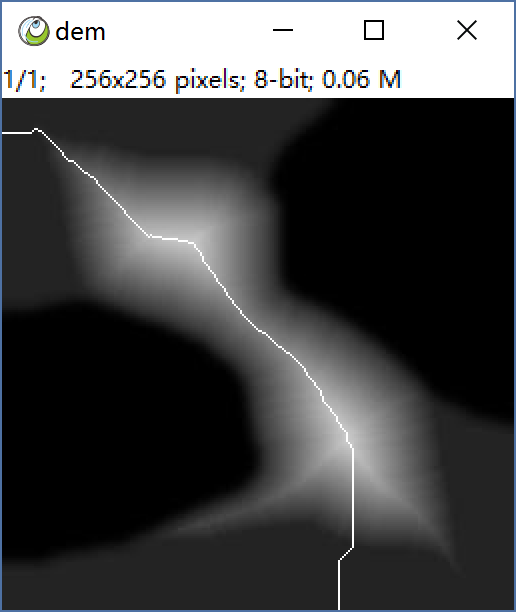
**分割的意义何在？**

分割在图像处理中是一个非常重要的课题，也是一个很困难的课题。分割意味着将图像划分成有具体意义的单元，为后续分析提供了可能性，随着学习深入，我们将陆续讨论计数，粒径分析等问题，这些工作都是以分割为前提的。

**ImagePy中的高程分析**

或许其他一些书籍和算法库不是这样分类的，因为一般分水岭回在图像分割中介绍，作者本人觉得制高点，集水点，等高线，分水岭，山脊线之间关系密切，并且都来源于地形分析。本章介绍的相关功能，都在ImagePy > Process > Hydrology下，意思是水文计算。我们选取分水岭工具进行讲解，其他功能类似。

**$ Imagepy > Process > Hydrology >**

菜单 对话框 分水岭

**参数讲解：**分水岭算法这里设定了几个参数，这些参数让算法变得更实用。

Sigma：在处理前对原图进行一个高斯模糊

Low：将小于Low值的区域作为种子点进行标记

Ascend：决定算法使用涨水法还是退水法法，退水是从最高处开始的，分割线穿过鞍部

Output：决定计算结果的形式，当填写white line的时候计算结果是白色的线，当output是gray line时，结果为灰度的分水线，灰度值等于原对应像素值，也可以选white line on ori，这种模式的结果是白色的分水岭线覆盖在原图之上。

**本章小结**

本章我们讨论了与高程分析有关的一些问题，这些问题往往与地理和水文有关。从最简单的阈值，等级化，到局部最大值，制高点提取，以及分水岭和山脊线提取。然后我们讨论了高程分析的对偶思想，制高点反过来可以变成集水点，高山反过来可以变成洼地，我们用反向的DEM进行分水岭计算，成功的得到了穿过鞍部的分水岭，将两座山峰分开了。最后我们用了一个提取蜂巢的例子讲解如何利用分水岭对图像进行分割。