SIFT精简流程

SIFT，又称尺度不变特征变换，在整个图像拼接的过程中所扮演的地位主要是寻找关键点和为关键点构建描述子。

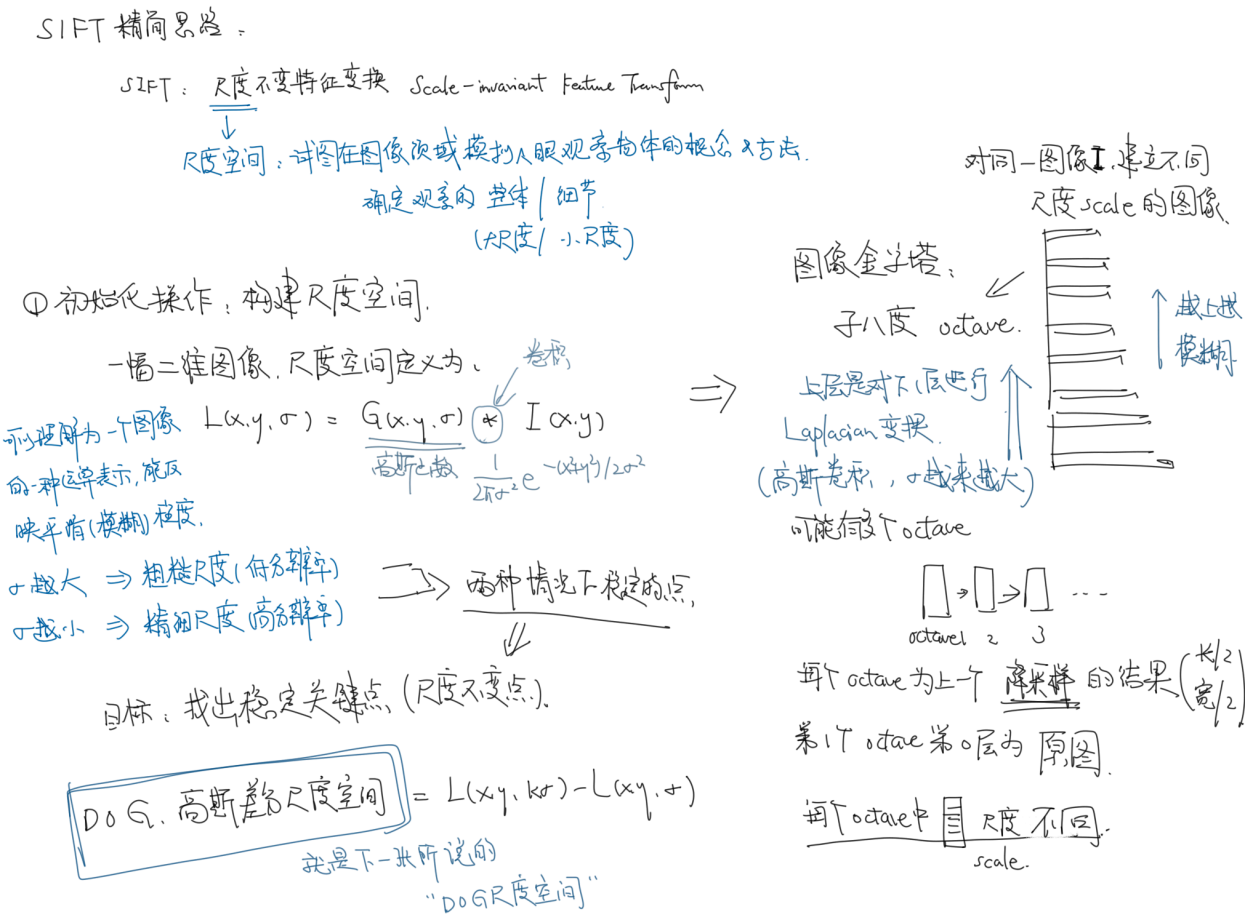
在课堂上，整体的流程和思路老师大概已经说的差不多了，我会稍微精简一些的用我的理解来概括一下SIFT包含的流程。

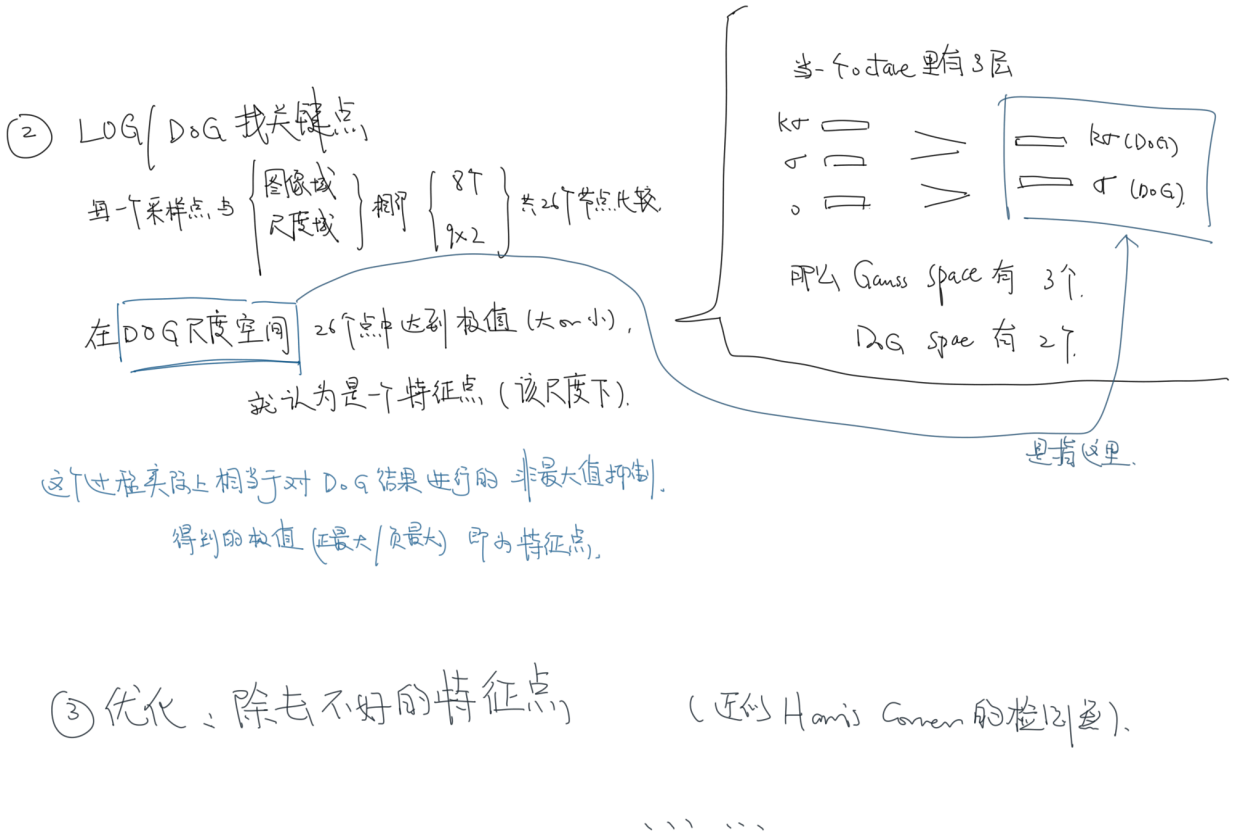
SIFT主要分为三个部分。

1. 尺度空间和金字塔构建。主要包括了生成图像在不同清晰度（sigma）下的尺度空间，以及相对应的DoG尺度空间图像，目的是为了求得关键点（不同模糊状态下变化大的点往往是边角）
2. 极值寻找和细化。主要是通过寻找极值求得关键点，并对计算的结果进行一定筛选的过程。
3. 生成方向和描述子。在寻找到的关键点的基础上，构建进一步的匹配信息。

接下来是稍微详细一些的介绍。

尺度空间和金字塔构建





实际实现的流程：

- `generateBaseImage()`，将输入图像放大一倍并应用高斯模糊。构建用于生成金字塔的初始图像。
- `computeNumberOfOctaves()`，将图像重复减半直到变得太小的次数。计算octave的总数。最终图像的边长至少应为1像素。

  > 将基础图像的`numOctaves - 1`时间减半，以得到`numOctaves`包括基础图像在内的图层。这样可以确保最大八度音阶的图像（最小图像）的边长至少为3。稍后我们将在每个DoG图像中搜索最小值和最大值，这意味着我们需要考虑3-by -3像素邻域。

- `generateGaussianKernels()`为特定图层中的每个图像创建一个模糊量列表。请注意，图像金字塔有`numOctaves`图层，但是每个图层（octave）本身都有`numIntervals + 3`图像。同一层中的所有图像都具有相同的宽度和高度，但是模糊量会逐渐增加。

  > 哪里`+ 3`来的？我们的`numIntervals + 1`图像涵盖了`numIntervals`从一个模糊值到该值两倍的步骤。我们还有另一个`+ 2`在图层中第一个图像之前的一个模糊步骤，以及图层中最后一个图像之后的另一个模糊步骤。最后我们需要这两个额外的图像，因为我们要减去相邻的高斯图像以创建DoG图像金字塔。

  > `gaussian_kernels`这个数组的第一个元素只是我们的开始`sigma`，但之后每个元素都是额外的尺度，我们需要与先前的尺度进行卷积。

- `generateGaussianImages()`从基础图像开始，然后根据我们的图像进行模糊处理`gaussian_kernels`。

我们跳过第一个元素，`gaussian_kernels`因为我们从已经具有该模糊值的图像开始。我们将倒数第二张图像减半，因为它具有我们想要的适当模糊效果，并使用它开始下一层（下一个octave）。

- 计算DoG尺度空间。`dog_images[2][i] = gaussian_images[2][i + 1] – gaussian_images[2][i]`，并注意DoG图像看起来像边缘贴图。

极值寻找和细化

- 遍历每一层，一次拍摄三个连续的图像。26个点的极大值（正或负）为关键点。
- **优化【待研究】：借助二次插值将这些关键点位置定位到亚像素精度。**

  https://www.cnblogs.com/fcfc940503/p/11484789.html

- **优化【待研究】：**为关键点附近的像素创建渐变的直方图。计算该邻域中每个像素处2D梯度的大小和方向。我们放置在该bin中的实际值是具有高斯加权的像素的梯度大小。这使得距离关键点较远的像素对直方图的影响较小。我们对附近的所有像素重复此过程，将结果累加到相同的36 bin直方图中。完成后，我们将平滑直方图。**为每个峰创建一个单独的关键点**，这些关键点除了它们的方向属性外都是相同的。如SIFT论文所述，这些**附加关键点**在实际应用中显着有助于检测稳定性。（位置相同，方向不同）

  https://www.cnblogs.com/fcfc940503/p/11492540.html

- 使用排序并删除重复项`removeDuplicateKeypoints()`。

生成方向和描述子

④ 方向计算.

1. 旋转不变性，确定特征点方向.

$$m(x,y)=\sqrt{(L(x+1,y)-L(x-1,y))^2+(L(x,y+1)-L(x,y-1))^2}$$ 模.

$$\theta(x,y)=\operatorname{atan2}\left(\frac{L(x,y+1)-L(x,y-1)}{L(x+1,y)-L(x-1,y)}\right)$$ 方向.

最终结果.

至此，每个关键点有3个信息：{位置 (x,y) / 尺度 $(k\sigma)$ / 方向 $(\theta(x,y))$}

实际计算时，在邻域采样，用直方图统计像素梯度方向.

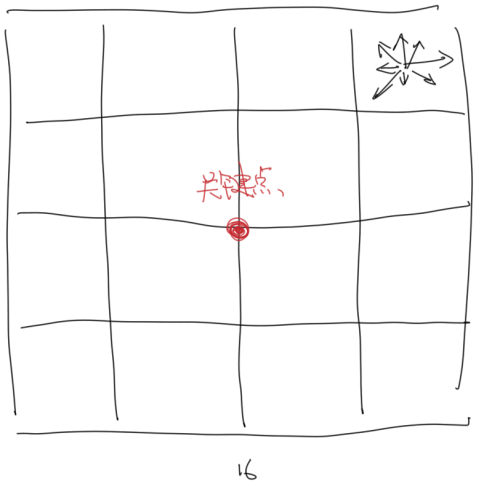
（45°一个柱，共8个柱 

）

用高斯函数对直方图进行平滑，减少突变影响

直方图峰值代表主方向.

⑤ 关键点描述子.



- 坐标轴为关键点方向.（旋转不变性）
- 取16×16的窗口.
  （关键点在对应尺度空间中的邻域）.
- 用公式求得每个点的梯度幅值&方向
- 高斯窗口加权运算

⇓

- 每4×4的区域进行一个累加形成种子点
  （方向按值累加，形成8个方向信息）

一共 4×4×8=128个信息值.

128维描述子.

对于每个关键点，我们的第一步是创建另一个梯度方向的直方图。我们考虑围绕每个关键点的正方形邻域（这次不同边长），但是现在我们将这个邻域旋转关键点的角度。这就是SIFT不变于旋转的原因。在这个旋转的社区中，我们做了一些特别的事情。我们计算行和列仓，它们只是邻域局部的索引，表示每个像素所在的位置。我们还计算每个像素的梯度大小和方向，就像计算关键点方向时一样。但是，我们实际上并不存储直方图并累积值，而只是存储每个像素的直方图bin索引和bin值。请注意，此处的直方图只有8个bin，可以覆盖360度，而不是以前的36 bin。

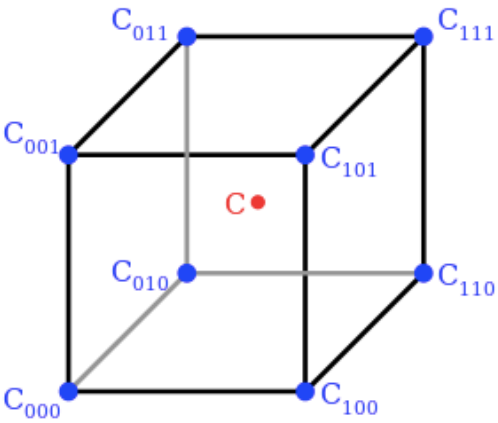
现在是棘手的部分。想象一下，假设我们采用正方形邻域，一个2D数组，并将每个像素替换为长度为36的向量。与每个像素关联的方向框将索引为其36长度的向量，并且在此位置，我们将存储加权梯度此像素的大小。这将形成大小的3D阵列`(window_width, window_width, num_bins)`，其评估对`(4, 4, 36)`。我们将展平这个3D数组作为描述符向量。但是，在此之前，最好进行一些平滑处理。

> 应该是（4，4，8）吧？4*4*8 = 128刚好是展开之后的128的长度

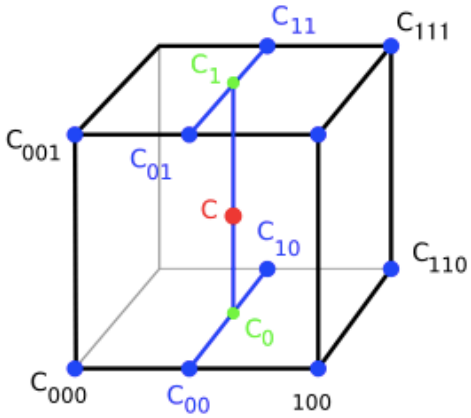
> 接下来是 归一化处理，将特征向量长度进行归一化处理，进一步去除光照的影响。？？

什么样的平滑？*提示：*它涉及更多有限的差异。我们将通过在三个维度（行箱，列箱和方向箱）中将其分布在八个邻域中来平滑每个邻域像素的加权梯度大小。我们将使用一种称为**三线性插值**的方法，或更准确地说，我们将使用其反函数。维基百科提供了公式和方法的良好可视化。简而言之，每个邻域像素都有一个行bin索引，列bin索引和方向bin索引，我们希望将其直方图值按比例分配给它的八个相邻bin，同时确保分配的部分加起来等于原始值。

您可能想知道为什么我们不简单地将直方图值的八分之一分配给八个邻居。问题在于，邻域像素可能具有*分数* bin索引。想象每个邻域像素都由3D点表示`[row_bin, col_bin, orientation_bin]`，如下图的红色点所示。该3D点可能不位于由其八个整数相邻点（下面的蓝色点）形成的立方体的精确中心。如果我们想准确地将红点处的值分配给蓝点，则更接近红点的蓝点应获得更大比例的红点值。这恰恰是三线性插值的反函数。我们将红色点分成两个绿色点，将两个绿色点分成四个点，最后将四个点分成最后的八个点。



我们执行三线性插值的逆运算，获取可能偏心的红点，并将其分布在八个邻点之间（来自Wikipedia）。



我们首先将红点划分为两个可能不相等的绿点，然后递归直到达到蓝点（来自Wikipedia的图像）。

我们的最后一步是将平滑的3D数组展平为长度为128的描述符向量。然后，我们将应用阈值并进行归一化。在OpenCV实施中，然后将描述符缩放并饱和到0到255之间，以在以后比较描述符时提高效率。

我们重复此过程为每个关键点生成一个描述符向量。