西安交通大学并行计算架构与模式项目

基于内容的多线程视频分析与检索系统

Content-based multi-thread video analysis and retrieval system

学 院（系）：软件学院

专　　业：软件工程

学 生 姓 名：王 洁　王亚斌

学　　号：3118311021 3118311106

指 导 教 师：朱 利

评 阅 教 师：朱 利

完 成 日 期：2018.11.20

西安交通大学

Xi'an Jiaotong University

摘　要

在互联网迅速发展的今天，人们获取信息的方式从以往传统的纸质报纸、杂志、期刊到广播、电视再到现在最主流的从网络获取，已经发生了巨大变化。同时，信息大爆炸也给大众带来了诸多问题，例如，监控摄像头已遍布中国大地的每个街头，昼夜不停地监视和录像。在改善社会治安的同时，产生海量视频信息，对成千上万个监控平台进行监控将耗费大量的人力、物力和时间。在海量的视频中查找我们需要的信息，无疑是大海捞针，也给视频监控带来巨大的挑战。在此背景下，论文设计实现基于内容的多线程视频分析与检索系统，用来将用户感兴趣的视频分析处理，从数据库中获得用户感兴趣的相似片段来源等信息，为用户提供更加高效、精确的视频数据信息。

论文设计并实现了基于内容的多线程视频分析与检索系统。主要是通过 SSE4 指令写的多线程提取程序从视频中抽取关键帧保存到数据库中，然后用 SIFT 算法提取这些图片的特征点，针对这些特征点使用基于 OpenMP 编写的 Kmeans 程序进行高维聚类生成词典，将所有数据库中图片针对词典多线程生成对应的单词，最后用 KD-tree 进行快速检索。整个项目包括 OpenMP 多线程技术、机器学习算法、SIFT 特征提取等相关技术。深入分析了 OpenMP 编程语言，着重探讨了如何用 OpenMP 进行多线程 Kmeans 聚类，大大提高了生成词典所用的时间。在特征提取方面，使用的 VLFeat 开源机器视觉库本身已经是多线程和 SSE4 优化的提取速度很快，同时又用 Python 脚本来进行任务分配多进程并行提取特征。KD-tree 的构建是最后通过 Python 中 Numpy、SKLearn 等库来实现，能够满足实时增加节点，实时检索的要求。对于图片视觉单词生成，注意到每张图片的 SIFT 特征点数量是不同的，因此进行了归一化操作，能够提高检索的准确度。最后采用可视化方法对结果进行展示说明，并对系统工作进行了总结。

关键词：视频检索；Sift 算法；Kmeans 算法；KDTree 算法；OpenMP；多线程模型

Python-based multi-thread news crawl and analysis system

Abstract

Today, with the rapid development of the Internet, people's access to information has changed dramatically from traditional paper newspapers, magazines, periodicals, radio and television to the most mainstream online access. At the same time, the information explosion has also brought many problems to the public. For example, surveillance cameras have spread all over the streets of China, monitoring and recording around the clock. While improving social security, generating massive amounts of video information, monitoring thousands of monitoring platforms will consume a lot of manpower, material resources and time. Finding the information we need in a huge amount of video is undoubtedly a needle in a haystack, and it also poses a huge challenge to video surveillance. In this context, the thesis design implements a content-based multi-threaded video analysis and retrieval system, which is used to analyze and process the video of interest to users, obtain information such as similar fragment sources of interest from the database, and provide users with more efficient, Accurate video data information.

The paper designs and implements a content-based multi-threaded video analysis and retrieval system. The multi-thread extraction program written by the SSE4 instruction is used to extract key frames from the video and save them to the database. Then, the SIFT algorithm is used to extract the feature points of these images. For these feature points, the Kmeans program based on OpenMP is used to generate a high-dimensional clustering dictionary. The images in all the databases are generated for the dictionary multi-threaded, and finally the KD-tree is used for quick retrieval. The entire project includes OpenMP multi-threading technology, machine learning algorithms, SIFT feature extraction and other related technologies. In-depth analysis of the OpenMP programming language, focusing on how to use OpenMP for multi-threaded Kmeans clustering, greatly improving the time used to generate the dictionary. In terms of feature extraction, the VLFeat open source machine vision library itself is already multi-threaded and SSE4 optimized for fast extraction, while using Python scripts for task-distribution multi-process parallel extraction features. The construction of KD-tree is finally realized by Numpy, SKLearn and other libraries in Python, which can meet the requirements of real-time adding nodes and real-time retrieval. For the picture visual word generation, it is noted that the number of SIFT feature points of each picture is different, so the normalization operation is performed, and the accuracy of the search can be improved. Finally, the results are presented by visual methods, and the system work is summarized.

Keywords: Video Retrieval; SIFT; Kmeans algorithm; KDtree algorithm; OpenMP; Multi-threaded model;

目　　录

1 引言

1.1 本论文研究的背景和意义

互联网快速发展，并且已经逐步渗透到社会生活的各个角落，社会生活的基本方式已经发生了巨大变化。过去人们用诸如报纸、杂志等基于物质的信息传播、交换方式已经逐步被淘汰，这类方式俨然成为新时代的弃儿，互联网逐步取代这些方式，成为生活的必需品，网络必将成为社会的万能胶，深深地将每一个人、团体连接到一起。在网络时代里，人们的生活变得前所未有的方便快捷。随着发展需要，数以万计的网站不断被建立，信息也随之呈爆炸式发展，尤其是随着 Flickr、Facebook 等社交网站的流行，图像、视频、音频、文本等异构数据每天都在以惊人的速度增长。

同时近年来随着多媒体技术的进一步发展，越来越多的信息以视频形式储存、传输和表现。例如 Youtube 是全球最大的视频分享平台，用户量高达 10 亿+，每天上传的 UGC 和 PGC 都是百万级别，然而这使得人们在浩如烟海的视频信息中快速容易地获得自己感兴趣的内容变得更加困难，同时视频网站也需要基于视频内容精准向用户推送感兴趣的内容。目前，监控摄像头已遍布中国大地的每个街头，昼夜不停地监视和录像。在改善社会治安的同时，产生海量视频信息，对成千上万个监控平台进行监控将耗费大量的人力、物力和时间。在海量的视频中查找我们需要的信息，无疑是大海捞针，也给视频监控带来巨大的挑战。传统的人海战术，因效率低下以及容易错过关键目标，容易使视频监控处于“监而不控”的状态。如何化解这一危机，是现代安防的热点和难点。解决这些问题视频检索是其中的关键技术。

基于文本的图像检索方法始于上世纪 70 年代，它利用文本标注的方式对图像中的内容进行描述，从而为每幅图像形成描述这幅图像内容的关键词，比如图像中的物体、场景等，这种方式可以是人工标注方式，也可以通过图像识别技术进行半自动标注。在进行检索时，用户可以根据自己的兴趣提供查询关键字，检索系统根据用户提供的查询关键字找出那些标注有该查询关键字对应的图片，最后将查询的结果返回给用户。这种基于文本描述的图像检索方式由于易于实现，且在标注时有人工介入，所以其查准率也相对较高。但是这种基于文本描述的方式需要人工介入标注过程，使得它只适用于小规模的图像数据，在大规模图像数据上要完成这一过程需要耗费大量的人力与财力。

基于内容的图像检索技术便逐步建立起来，并在近十多年里得到了迅速的发展。典型的基于内容的图像检索基本框架如上图 1.1 所示，它利用计算机对图像进行分析，建立图像特征矢量描述并存入图像特征库，当用户输入一张查询图像时，用相同的特征提取方法提取查询图像的特征得到查询向量，然后在某种相似性度量准则下计算查询向量

到特征库中各个特征的相似性大小，最后按相似性大小进行排序并顺序输出对应的图片。基于内容的图像检索技术将图像内容的表达和相似性度量交给计算机进行自动的处理，克服了采用文本进行图像检索所面临的缺陷，并且充分发挥了计算机长于计算的优势，大大提高了检索的效率，从而为海量图像库的检索开启了新的大门。

论文的主要工作是基于内容的多线程视频分析与检索系统的设计和实现。本系统使用多线程技术从海量视频中快速抽取 SIFT 图像特征。并编写 OpenMP 优化过的 Kmeans 来对特征进行聚类获得单词向量。最终基于相似度匹配的 k 近邻(KD-tree)来快速检索数据库。

总的来说，视频检索需要有高性能的多线程图像处理技术、数据处理技术、海量数据存储技术、检索技术的支持，对图像处理、特征提取、数据分析等相关技术的研究可以满足用户获取特定视频内容的需求，比如高效、全面、重点突出等。

1.2 主要研究内容与工作

基于内容的多线程视频分析与检索系统的研究内容主要包括以下几个方面：

（1）对视频检索系统进行了介绍，包括发展和使用的相关算法及技术；

（2）设计并行预处理流程，从视频流中获取关键帧，并对图像数据进行预处理（增强，旋转，滤波，切分等）；

（3）设计并行特征提取模块，对图像数据进行高效稳定可重复的特征提取（SIFT 算法）；

（4）对图像数据库建立图像特征数据库，对数据处理模块进行规划、设计，包括数据清洗、数据处理、属性规整等功能；

（5）设计并行词袋模型提取模块，抽取检索图像特征，使用基于 OpenMP 编写的 Kmeans 构建字典，生成特征向量；

（6）设计检索模块，包含相似性度量准则，排序，搜索，使用的是 KD-tree 算法；

1.3 论文的组织结构

论文整体成文结构可分为三章，具体各章内容如下：

第一章引言。本章介绍了论文的研究背景、目的和意义，概述了图像检索等技术的发展。并且对本论文研究内容进行了说明，简单论述了论文的组织结构。

第二章基于内容的多线程视频分析与检索系统的设计。本章主要对系统总体架构、数据提取模块、数据处理模块、数据检索模块进行了设计，为实现系统提供了设计基础。

第三章基于内容的多线程视频分析与检索系统的实现。本章实现了数据提取系统，包括多进程视频提取实现、多线程特征提取、多线程 KMeans、多线程图片处理。对关键词 KMeans 算法做了重点研究，采纳基于 SIFT 算法的特征提取技术，利用基于 KMeans 建立的词典对图片生成描述向量并利用最近邻算法来测算相似度进行检索。

最后总结了论文工作。总结了系统在设计实现过程中遇到的困难，对未来可继续进行的研究工作进行了展望。

1.4 本章小结

本章介绍论文的研究背景、目的和意义，并对图像检索等技术的发展进行了概述。同时，对本论文研究内容进行了说明，简单论述了论文的组织结构。

2　多线程视频分析与检索系统的设计

2.1　系统总体架构设计

系统基本环境配置为 Windows 操作系统，安装 MinGW 并且具有 OpenMP 库函数。根据系统的功能需求，设计系统的系统组织架构如图 2.1 所示。下文将具体介绍各模块的设计工作。

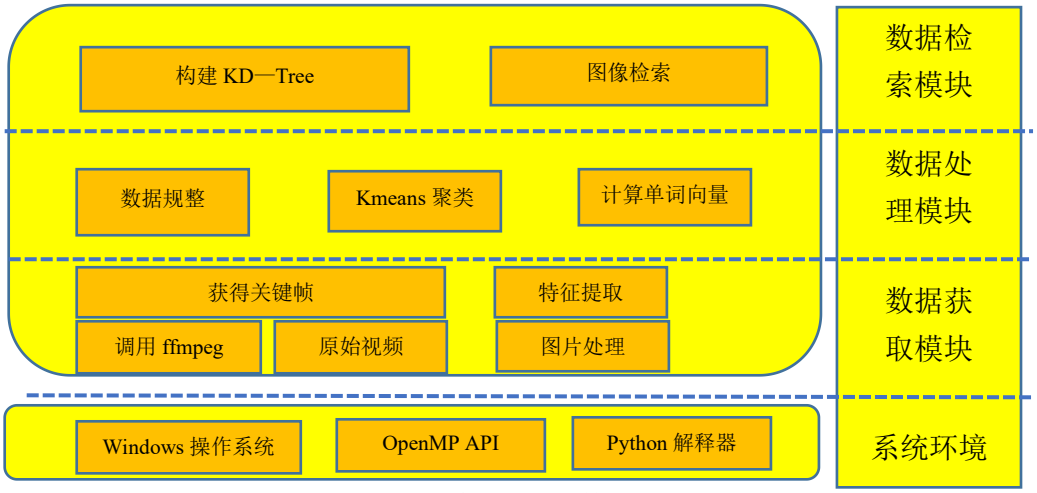

图 2.1　系统组织架构图

图 2.2 表示的是系统流程图：

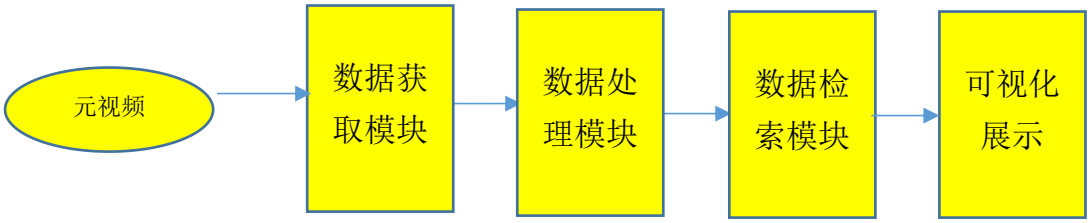

图 2.2　系统流程图

2.2　特征提取设计

2.2.1　提取策略设计

SIFT 特征提取使用多进程并行提取策略，处理流程如图 2.3 所示。

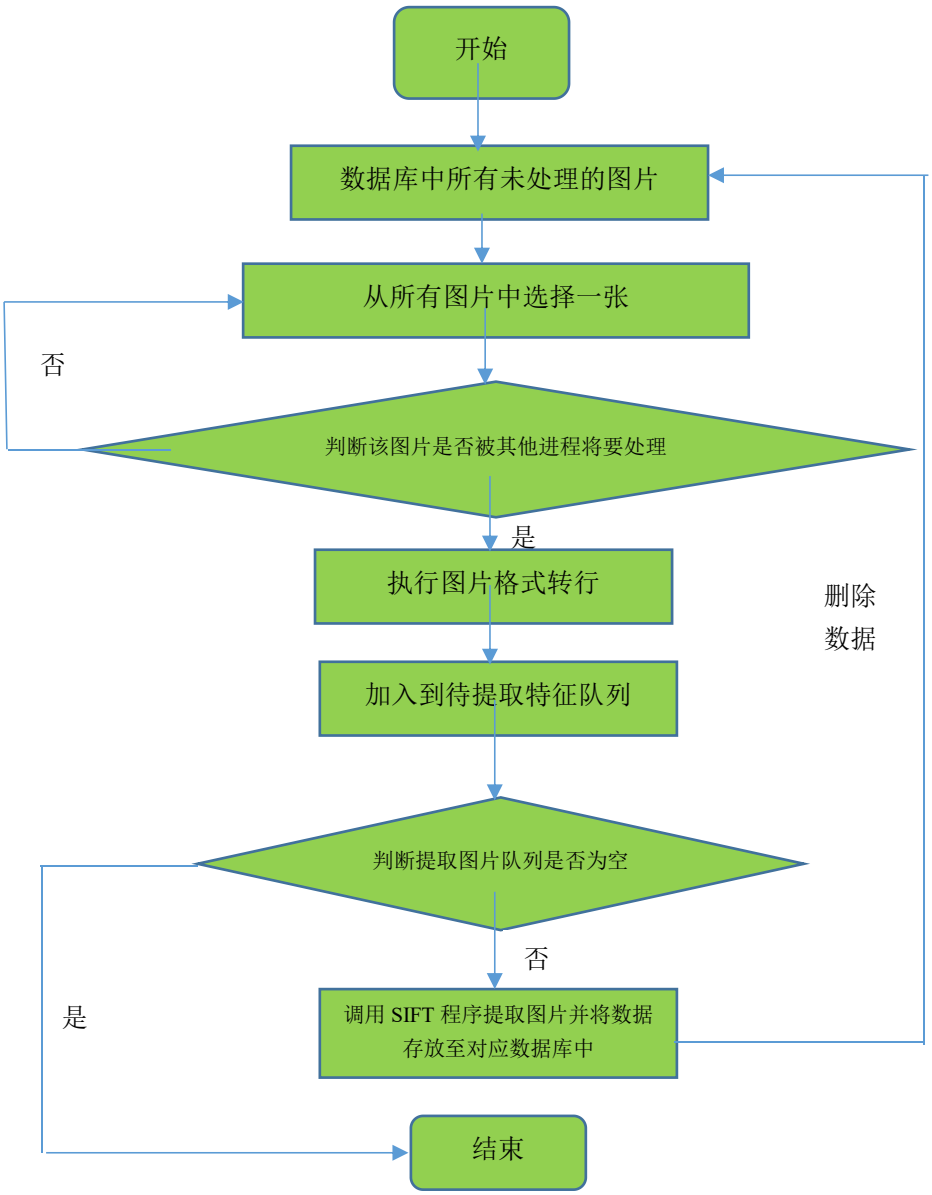

图 2.3　特征提取处理流程

一开始，将视频数据存放至 VideoFolder 文件夹中，一个循环调用的 Python 程序会将所有该目录下文件记录在 TXT 目录表中，该程序同时维护另外两个目录，分别是已经处理的视频和待处理的视频。之后用 C 语言编写的程序会从未处理视频的 TXT 目录中取出文件交给 FFmpeg 程序去并行提取关键帧，并且将提取过的文件从未处理目录中删除。处理过的视频都放在 DataSet 文件夹中对应的位置，这里同样有 Python 程序维护着三个目录。也是同样的方式调用 SIFT 程序并行提取特征。

4.2.2 特征聚合生成字典

系统主要是使用KMeans算法来生成SIFT描述子的字典，KMeans本身是用OpenMP优化过的多线程并行聚类的，但是只需要当数据库中数据变化巨大后才需要使用。

4.2.3 生成图片单词表达

将数据库中图片的 SIFT 描述子们全都转换成一个 K 维的向量，以后就用这个 K 维向量来表示这张图片，将所有数据存放至一个文本中。

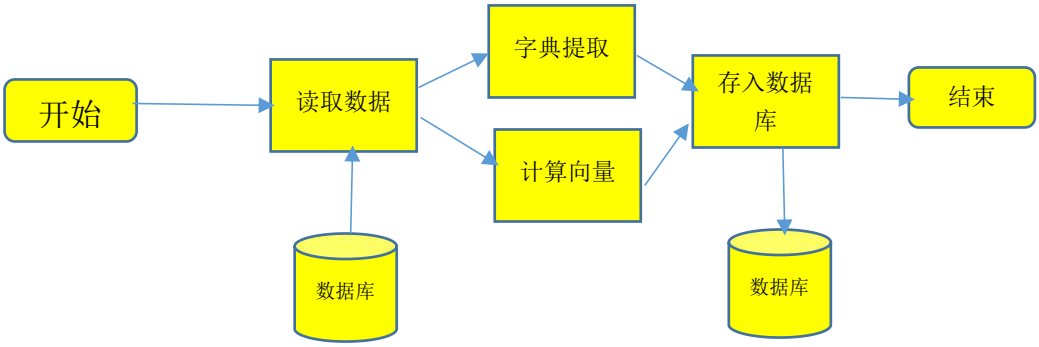

图 2.4 单词建立流程图

2.3 数据检索设计

数据检索就是通过构建最近邻 KD-tree 来计算数据库中同被检索图片最相似的 K 个图片来源，由此可以定位到视频来源。

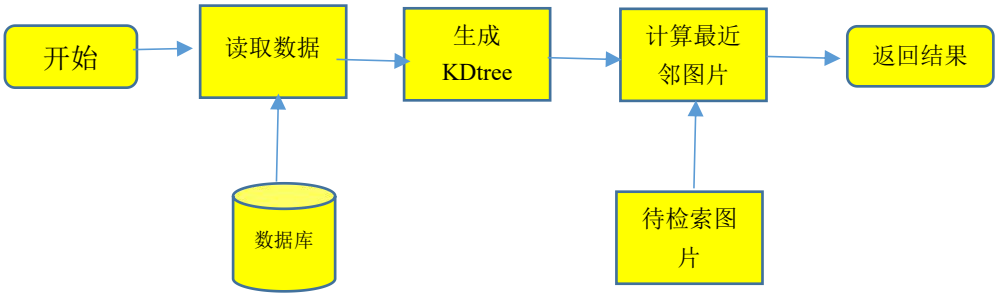

图 2.5 检索流程图

2.4 本章小结

本章进行了基于内容的多线程视频分析与检索系统设计，包括数据提取模块、数据处理模块和数据检索模块的设计，为之后数据提取和数据处理、分析工作提供了设计基础。

3　视频分析与检索系统的实现

3.1　数据获取模块实现

本节将根据上节视频检索系统设计，结合程序代码详细介绍系统的实现过程。

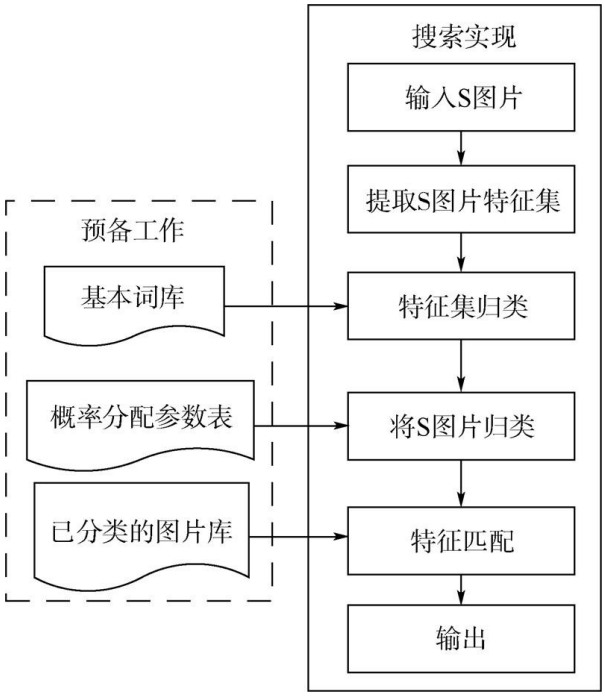

图 3.1　提取过程流程图

3.1.1　关键帧的获取

关键帧，是指动画中一个绘图，定义任何的起点和终点平滑过渡，一系列关键帧定义了观看者将看到的运动，而关键帧在电影，视频或动画上的位置定义了运动的时间。

整个关键帧获取其实是使用 FFmpeg 软件来处理的，FFmpeg 是一个自由软件，可以运行音频和视频多种格式的录影、转换、流功能，同时该软件是当今各大主流播放器的内核。在编译过程通过配置，很方便地支持多线程和 MMX、SSE 和 AVX 等优化。

在本项目中通过一个 C 语言函数来调用 FFmpeg 对视频进行提取，主要目的是能自动进行任务分配并且能够同时启用多个 FFmpeg 进程来处理数据。关键代码如下：

```
for(int j=0; j<index; ++j){
   sprintf(name[j], "ffmpeg -i %s\\%s.mp4 -r 1 %s",input_filename, file[j],
           final_out[j]);
}
#pragma omp parallel for
```

```
for(int j=0; j<index; ++j){
    system(name[j]);
    printf("\n");
}
```

3.1.2 关键帧的处理

获得的关键帧被按来源存放在数据库中等待预处理操作。进行特征提取之前需要对图片进行处理，主要是获得灰度图像以及尺寸变换，同时要备份一个文件列表记录需要提取的图片标识符（也就是地址），方便后面程序操作。

预处理步骤是在 Python 脚本 convert.py 中实现的，主要是使用了 python 自带的图像处理库 PIL，因为单纯的 C/C++没有能处理图片的库，这个操作本身计算量也很小，所以简单用十几行的 Python 代码实现比较合适，convert.py 关键代码实现如下：

```
def process_image(imagename,outimg):
    # create a pgm file
    im = Image.open(imagename).convert('L') #.convert('L') 将 RGB 图像转为灰度模式
    im.save(outimg+'.pgm') #将灰度值图像信息保存在.pgm 文件中

if __name__ == "__main__":
    in_path = '..\\CBVideoSearch\\DataSets\\Imgs\\'
    out_path = '..\\CBVideoSearch\\DataSets\\processed_imgs\\'
    for dirname in os.listdir(in_path):
        if not os.path.exists(out_path+dirname):
            os.makedirs(out_path+dirname)
        for file in os.listdir(in_path+dirname):
    process_image(in_path+dirname+'\\'+file,str(out_path+dirname+'\\'+file.spli
t('.')[0]))
```

为方便后面处理，再转成灰度图后还使用 PGM 格式来保存数据，PGM 是黑白超声影像图像中经常用到的格式，最大特点是文件头部分用 ASCII 码来存储的，方便读取。

维护的目录有三个，所有图像、已经处理过的数据、当前未处理的数据，都有 TXT 格式的文档，由一个 Python 脚本 Distributer.py 维护，这样后续特征提取程序可以并行地从数据库中提取数据并且处理，不会产生等待，也减少了实现进程通讯的难度。

图 3.2　维护的目录示意图

3.1.3　特征提取

本研究使用的特征是 SIFT 算子，SIFT（Scale-invariant feature transform）是一种检测局部特征的算法，该算法通过求一幅图中的特征点（interest points,or corner points）及其有关 scale 和 orientation 的描述子得到特征并进行图像特征点匹配，获得了良好效果。该方法的主要步骤如下：

一、构建尺度空间

SIFT 算法是在不同的尺度空间上查找关键点，而尺度空间的获取需要使用高斯模糊来实现。高斯模糊是一种图像滤波器，它使用正态分布(高斯函数)计算模糊模板，并使用该模板与原图像做卷积运算，达到模糊图像的目的。N 维空间正态分布方程为：

$$G(r) = \frac{1}{(\sqrt{2\pi\sigma^2})^N} e^{-r^2/2\sigma^2}$$

其中，σ 是正态分布的标准差，σ 值越大，图像越模糊(平滑)。r 为模糊半径，模糊半径是指模板元素到模板中心的距离。只有通过高斯卷积核才能实现尺度变换。这个滤波之后的结果就是图象被模糊化了，高斯核代码如下：

```
if (self->gaussFilterSigma != sigma) {
  vl_uindex j ;
  vl_sift_pix acc = 0 ;
  if (self->gaussFilter) vl_free (self->gaussFilter) ;
```

```
            self->gaussFilterWidth = VL_MAX(ceil(4.0 * sigma), 1) ;
            self->gaussFilterSigma = sigma ;
            self->gaussFilter = vl_malloc (sizeof(vl_sift_pix) * (2 *
self->gaussFilterWidth + 1)) ;
            for (j = 0 ; j < 2 * self->gaussFilterWidth + 1 ; ++j) {
               vl_sift_pix d = ((vl_sift_pix)((signed)j
-(signed)self->gaussFilterWidth))/((vl_sift_pix)sigma) ;
               self->gaussFilter[j] = (vl_sift_pix) exp (- 0.5 * (d*d)) ;
               acc += self->gaussFilter[j] ;
            }
            for (j = 0 ; j < 2 * self->gaussFilterWidth + 1 ; ++j) {
               self->gaussFilter[j] /= acc ;
            }
         }
```

有了高斯滤波核，我们就能通过多尺度图像金字塔形成不同分辨率下的图像。多尺度图像金字塔通过对原始图像进行多尺度像素采样的方式，生成 N 个不同分辨率的图像。

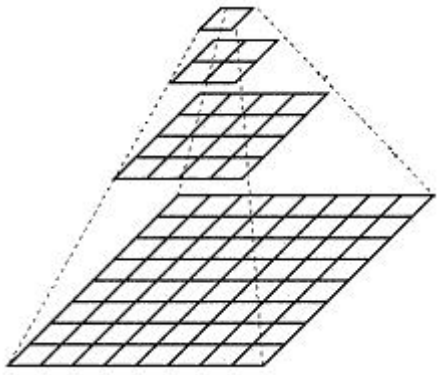

图 3.3　多尺度图像金字塔

高斯金字塔是在 SIFT 算子中提出来的概念，首先高斯金字塔并不是一个图像金字塔，而是有很多组（Octave）金字塔构成，并且每组金字塔都包含若干层（Interval）。

高斯金字塔构建过程：

对于参数 σ，在 Sift 算子中取的是固定值 1.6。

1．先将原图像扩大一倍之后作为高斯金字塔的第 1 组第 1 层，将第 1 组第 1 层图像经高斯卷积（其实就是高斯平滑或称高斯滤波）之后作为第 1 组金字塔的第 2 层。

2．将σ乘以一个比例系数k,等到一个新的平滑因子σ=k*σ，用它来平滑第1组第2层图像，结果图像作为第3层。

3．如此这般重复，最后得到L层图像，在同一组中，每一层图像的尺寸都是一样的，只是平滑系数不一样。它们对应的平滑系数分别为:0,

σ，$k^2\sigma$，$k^3\sigma$.....$k^{(L-2)}\sigma$。

4．将第1组倒数第三层（也就是从上往下数第三层）图像作比例因子为2的降采样，得到的图像作为第2组的第1层，然后对第2组的第1层图像做平滑因子为σ的高斯平滑，得到第2组的第2层，就像步骤2中一样，如此得到第2组的L层图像，同组内它们的尺寸是一样的，对应的平滑系数分别为：0,

σ，$k^2\sigma$，$k^3\sigma$.....$k^{(L-2)}\sigma$。但是在尺寸方面第2组是第1组图像的一半。

这样反复执行，就可以得到一共O组，每组L层，共计O* L个图像，这些图像一起就构成了高斯金字塔，结构如下：

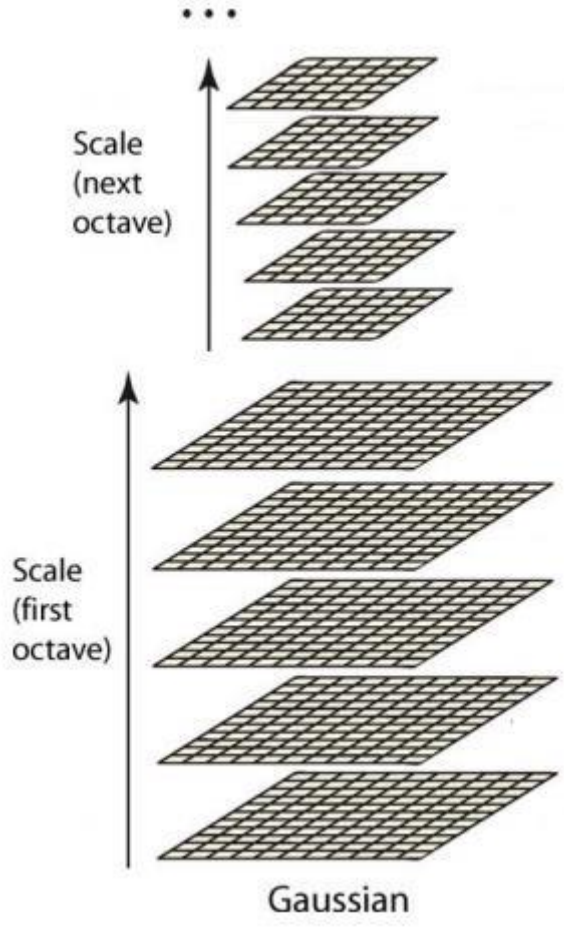

图3.4　高斯金字塔

降维采样的相关代码如下:

```
copy_and_downsample
(vl_sift_pix            *dst,
 vl_sift_pix const *src,
 int width, int height, int d)
{
   int x, y ;

   d = 1 << d ; /* d = 2^d */
   for(y = 0 ; y < height ; y+=d) {
      vl_sift_pix const * srcrowp = src + y * width ;
      for(x = 0 ; x < width - (d-1) ; x+=d) {
         *dst++ = *srcrowp ;
         srcrowp += d ;
      }
   }
}
```

二、LoG 近似 DoG 找到关键点

使用高斯金字塔每组中相邻上下两层图像相减，得到高斯差分塔（Difference of Gaussian， DOG 算子），如下图所示，进行极值检测：

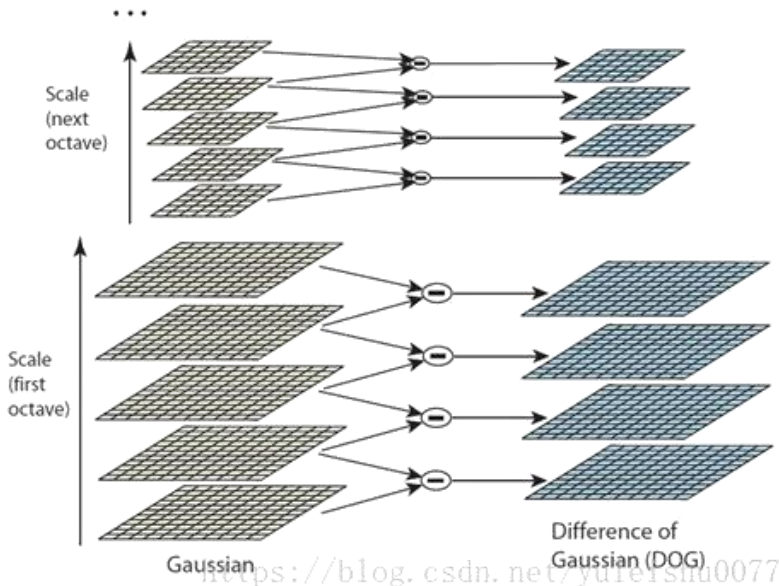

图 3.5 DOG 找关键点

为了寻找尺度空间的极值点，每一个采样点要和它所有的相邻点比较，看其是否比它的图像域和尺度域的相邻点大或者小。如下图所示，中间的检测点和它同尺度的 8 个相邻点和上下相邻尺度对应的 9×2 个点共 26 个点比较，以确保在尺度空间和二维图像空间都检测到极值点。一个点如果在 DOG 尺度空间本层以及上下两层的 26 个领域中是最大或最小值时，就认为该点是图像在该尺度下的一个特征点。

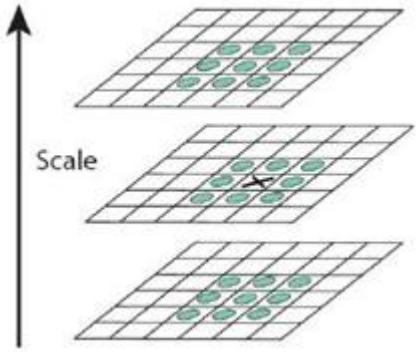

图 3.6　特征点的获得

上面检测到的极值点是离散空间的极值点，然后通过拟合三维二次函数来精确确定关键点的位置和尺度，同时去除低对比度的关键点和不稳定的边缘响应点(因为 DoG 算子会产生较强的边缘响应)，以增强匹配稳定性、提高抗噪声能力。为了使描述符具有旋转不变性，需要利用图像的局部特征(关键点邻域像素的梯度方向分布特性)给每一个关键点分配一个基准方向。

```
void
vl_sift_detect (VlSiftFilt * f)
{
/* clear current list */
f-> nkeys = 0 ;
/* compute difference of gaussian (DoG) */
pt = f-> dog ;
for (s = s_min ; s <= s_max - 1 ; ++s) {
vl_sift_pix* src_a = vl_sift_get_octave (f, s ) ;
vl_sift_pix* src_b = vl_sift_get_octave (f, s + 1) ;
vl_sift_pix* end_a = src_a + w * h ;
while (src_a != end_a) {
*pt++ = *src_b++ - *src_a++ ;
}
}
/* ---------------------------------------------------------------
* Find local maxima of DoG
```

```
* ------------------------------------------------------------- */
/* start from dog [1,1,s_min+1] */
pt = dog + xo + yo + so ;
for(s = s_min + 1 ; s <= s_max - 2 ; ++s) {
for(y = 1 ; y < h - 1 ; ++y) {
for(x = 1 ; x < w - 1 ; ++x) {
v = *pt ;
#define CHECK_NEIGHBORS(CMP,SGN) \
( v CMP ## = SGN 0.8 * tp && \
v CMP *(pt + xo) && \
v CMP *(pt - xo) && \
.....(太多省略不写).....
v CMP *(pt - yo + xo - so) && \
v CMP *(pt - yo - xo - so) )
if (CHECK_NEIGHBORS(>,+) ||
CHECK_NEIGHBORS(<,-) ) {
/* make room for more keypoints */
if (f->nkeys >= f->keys_res) {
f->keys_res += 500 ;
if (f->keys) {
f->keys = vl_realloc (f->keys,
f->keys_res *
sizeof(VlSiftKeypoint)) ;
} else {
f->keys = vl_malloc (f->keys_res *
sizeof(VlSiftKeypoint)) ;
}
}
k = f->keys + (f->nkeys ++) ;
k-> ix = x ;
k-> iy = y ;
k-> is = s ;
}
pt += 1 ;
}
pt += 2 ;
}
pt += 2 * yo ;
}
/* -------------------------------------------------------------------
* Refine local maxima
* ------------------------------------------------------------- */
/* this pointer is used to write the keypoints back */
k = f->keys ;
```

```
for (i = 0 ; i < f->nkeys ; ++i) {
int x = f-> keys [i] .ix ;
int y = f-> keys [i] .iy ;
int s = f-> keys [i]. is ;
double Dx=0,Dy=0,Ds=0,Dxx=0,Dyy=0,Dss=0,Dxy=0,Dxs=0,Dys=0 ;
double A [3*3], b [3] ;
int dx = 0 ;
int dy = 0 ;
int iter, i, j ;
for (iter = 0 ; iter < 5 ; ++iter) {
x += dx ;
y += dy ;
pt = dog
+ xo * x
+ yo * y
+ so * (s - s_min) ;
/** @brief Index GSS @internal */
#define at(dx,dy,ds) (*( pt + (dx)*xo + (dy)*yo + (ds)*so))
/** @brief Index matrix A @internal */
#define Aat(i,j) (A[(i)+(j)*3])
/* compute the gradient */
Dx = 0.5 * (at(+1,0,0) - at(-1,0,0)) ;
Dy = 0.5 * (at(0,+1,0) - at(0,-1,0));
Ds = 0.5 * (at(0,0,+1) - at(0,0,-1)) ;
/* compute the Hessian */
Dxx = (at(+1,0,0) + at(-1,0,0) - 2.0 * at(0,0,0)) ;
Dyy = (at(0,+1,0) + at(0,-1,0) - 2.0 * at(0,0,0)) ;
Dss = (at(0,0,+1) + at(0,0,-1) - 2.0 * at(0,0,0)) ;
Dxy = 0.25 * ( at(+1,+1,0) + at(-1,-1,0) - at(-1,+1,0) - at(+1,-1,0) ) ;
Dxs = 0.25 * ( at(+1,0,+1) + at(-1,0,-1) - at(-1,0,+1) - at(+1,0,-1) ) ;
Dys = 0.25 * ( at(0,+1,+1) + at(0,-1,-1) - at(0,-1,+1) - at(0,+1,-1) ) ;
/* solve linear system ..................................... */
Aat(0,0) = Dxx ;
Aat(1,1) = Dyy ;
Aat(2,2) = Dss ;
Aat(0,1) = Aat(1,0) = Dxy ;
Aat(0,2) = Aat(2,0) = Dxs ;
Aat(1,2) = Aat(2,1) = Dys ;
b[0] = - Dx ;
b[1] = - Dy ;
b[2] = - Ds ;
/* Gauss elimination */
for(j = 0 ; j < 3 ; ++j) {
double maxa = 0 ;
```

```
double maxabsa = 0 ;
int maxi = -1 ;
double tmp ;
/* look for the maximally stable pivot */
for (i = j ; i < 3 ; ++i) {
double a = Aat (i,j) ;
double absa = vl_abs_d (a) ;
if (absa > maxabsa) {
maxa = a ;
maxabsa = absa ;
maxi = i ;
}
}
/* if singular give up */
if (maxabsa < 1e-10f) {
b[0] = 0 ;
b[1] = 0 ;
b[2] = 0 ;
break ;
}
i = maxi ;
/* swap j-th row with i-th row and normalize j-th row */
for(jj = j ; jj < 3 ; ++jj) {
tmp = Aat(i,jj) ; Aat(i,jj) = Aat(j,jj) ; Aat(j,jj) = tmp ;
Aat(j,jj) /= maxa ;
}
tmp = b[j] ; b[j] = b[i] ; b[i] = tmp ;
b[j] /= maxa ;
/* elimination */
for (ii = j+1 ; ii < 3 ; ++ii) {
double x = Aat(ii,j) ;
for (jj = j ; jj < 3 ; ++jj) {
Aat(ii,jj) -= x * Aat(j,jj) ;
}
b[ii] -= x * b[j] ;
}
}
/* backward substitution */
for (i = 2 ; i > 0 ; --i) {
double x = b[i] ;
for (ii = i-1 ; ii >= 0 ; --ii) {
b[ii] -= x * Aat(ii,i) ;
}
}
```

```
/* ..................................................... */
/* If the translation of the keypoint is big, move the keypoint
* and re-iterate the computation. Otherwise we are all set.
*/
dx= ((b[0] > 0.6 && x < w - 2) ? 1 : 0)
+ ((b[0] < -0.6 && x > 1 ) ? -1 : 0) ;
dy= ((b[1] > 0.6 && y < h - 2) ? 1 : 0)
+ ((b[1] < -0.6 && y > 1 ) ? -1 : 0) ;
if (dx == 0 && dy == 0) break ;
}
/* check threshold and other conditions */
{
double val = at(0,0,0)
+ 0.5 * (Dx * b[0] + Dy * b[1] + Ds * b[2]) ;
double score = (Dxx+Dyy)*(Dxx+Dyy) / (Dxx*Dyy - Dxy*Dxy) ;
double xn = x + b[0] ;
double yn = y + b[1] ;
double sn = s + b[2] ;
vl_bool good =
vl_abs_d (val) > tp &&
score < (te+1)*(te+1)/te &&
score >= 0 &&
vl_abs_d (b[0]) < 1.5 &&
vl_abs_d (b[1]) < 1.5 &&
vl_abs_d (b[2]) < 1.5 &&
xn >= 0 &&
xn <= w - 1 &&
yn >= 0 &&
yn <= h - 1 &&
sn >= s_min &&
sn <= s_max ;
if (good) {
k-> o = f->o_cur ;
k-> ix = x ;
k-> iy = y ;
k-> is = s ;
k-> s = sn ;
k-> x = xn * xper ;
k-> y = yn * xper ;
k-> sigma = f->sigma0 * pow (2.0, sn/f->S) * xper ;
++ k ;
}
} /* done checking */
} /* next keypoint to refine */
```

```
/* update keypoint count */
f-> nkeys = (int)(k - f->keys) ;
}
```

三、给特征点赋值一个 128 维方向参数

在完成关键点的梯度计算后，使用直方图统计邻域内像素的梯度和方向。梯度直方图的范围是 0～360 度，其中每 10 度一个柱，总共 36 个柱；或者每 45 度一个柱，总共 8 个柱。如下图所示，直方图的峰值则代表了该关键点处邻域梯度的主方向，即作为该关键点的方向。下图所示为八个方向的梯度直方图。

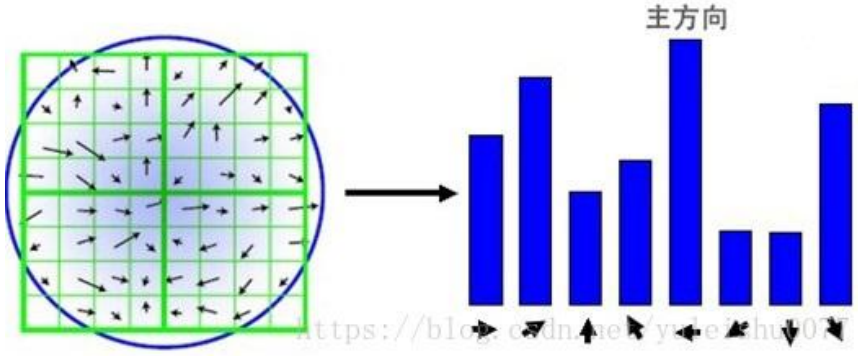

图 3.7　特征点周围主方向示意图

计算关键点方向的核心代码如下：

```
maxh = 0 ;
for (i = 0 ; i < nbins ; ++i)
   maxh = VL_MAX (maxh, hist [i]) ;
/* find peaks within 80% from max */
nangles = 0 ;
for(i = 0 ; i < nbins ; ++i) {
   double h0 = hist [i] ;
   double hm = hist [(i - 1 + nbins) % nbins] ;
   double hp = hist [(i + 1 + nbins) % nbins] ;
   /* is this a peak? */
   if (h0 > 0.8*maxh && h0 > hm && h0 > hp) {
      /* quadratic interpolation */
      double di = - 0.5 * (hp - hm) / (hp + hm - 2 * h0) ;
      double th = 2 * VL_PI * (i + di + 0.5) / nbins ;
      angles [ nangles++ ] = th ;
      if( nangles == 4 )
         goto enough_angles ;
```

```
        }
      }
    enough_angles:
      return nangles ;
```

四、关键点描述子的生成

对于每一个关键点，拥有三个信息：位置、尺度以及方向。接下来就是为每个关键点建立一个描述符，用一组向量将这个关键点描述出来，使其不随各种变化而改变，如光照变化、视角变化等。这个描述子不但包括关键点，也包含关键点周围对其有贡献的像素点，并且描述符应该有较高的独特性，以便于提高特征点正确匹配的概率。

其实这个过程就是在关键点周围一定的范围内重新统计周边像素的方向，如图所示，将周边区域划分为 4*4 的小区域，然后再统计小区域中 8 个方向的量，最后形成 128 维的向量经过归一化就是这个关键点的描述子。

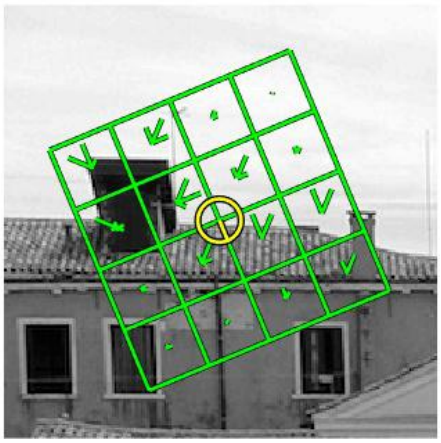

图 3.8　某一特征点的描述子

```
252.45 215.022 0.888115 4.70393 22 4 0 1 2 0 0 0 166 75 0 0 0 0 0 0 166 59 0 3 15 0 0 0 0 0 0 7 37 0 0 0 49 11 0 1 1 0 0 0
245.947 216.143 1.08433 4.585 12 0 0 0 6 1 0 17 139 0 0 0 0 0 0 139 95 0 0 7 11 1 0 111 0 0 0 6 19 4 0 0 1 0 0 0 10 2 0 2
399.711 220.946 1.11279 5.02478 25 0 0 0 9 10 13 8 203 14 0 0 0 1 1 59 114 2 0 0 13 18 4 30 2 0 0 0 2 3 2 7 53 1 0 7 31 2
337.481 223.515 1.03031 4.48059 13 5 1 0 0 0 0 0 166 100 0 0 0 0 0 0 166 70 0 0 7 3 0 0 0 0 0 0 20 9 0 0 41 10 1 0 0 0 0 1
220.301 226.066 0.980589 4.50881 48 0 0 0 0 0 0 0 59 167 4 0 0 0 0 0 167 56 5 0 7 11 4 1 34 0 0 0 2 4 1 0 0 49 7 0 0 0 0 0 6
218.021 228.528 1.03888 4.37451 7 0 0 0 1 1 0 0 151 12 0 0 0 0 0 66 151 10 0 0 9 6 1 89 0 0 0 3 18 10 0 1 2 0 0 0 0 0 0
446.044 229.126 1.05985 4.84801 76 0 0 0 6 3 0 18 198 25 3 7 1 0 0 21 27 14 5 54 27 0 0 0 9 3 0 7 6 0 0 0 83 0 0 0 11 2 0
57.1303 238.634 1.10507 5.51308 28 0 0 0 0 0 0 0 12 168 9 0 0 0 0 0 103 119 5 1 10 8 2 1 39 0 0 3 40 11 1 0 0 40 6 1 0 0 0 0
71.7282 250.454 1.07845 4.94735 2 3 0 3 7 1 0 0 127 140 0 0 0 0 0 0 141 141 0 0 0 2 4 2 19 12 0 0 0 8 19 5 28 10 0 1 2 3 1
14.3132 211.983 1.26126 5.66408 20 2 0 0 0 0 0 0 12 149 27 0 0 0 0 0 100 94 35 8 1 0 0 0 41 0 7 18 12 0 0 0 0 36 12 0 0 0 0
294.077 220.445 1.23586 4.92313 1 0 0 0 0 0 0 1 1 173 0 0 0 0 0 0 18 173 2 0 1 2 0 0 30 10 0 0 5 13 0 0 1 4 0 0 0 0 0 0 1 17
302.444 220.408 1.24969 5.00628 34 1 0 0 0 0 0 0 6 200 22 0 0 0 0 0 21 130 12 0 3 16 1 0 7 0 0 0 3 8 0 0 0 52 5 0 0 0 0 0 3
323.911 223.484 1.14804 4.76229 69 0 0 0 0 0 0 0 13 192 1 0 0 0 0 0 100 77 0 0 1 17 1 0 30 0 0 0 1 9 1 1 0 67 3 0 0 0 0 0 6
357.015 223.382 1.46925 4.66293 22 3 0 0 0 0 0 0 1 201 60 0 0 0 0 0 0 146 23 0 0 15 7 0 0 0 0 0 0 4 12 1 0 0 44 5 1 2 1 0 0 0
258.359 213.634 1.38784 4.5048 30 13 0 0 3 0 0 0 188 116 0 0 0 0 0 0 146 57 0 0 17 10 0 0 0 0 0 0 0 16 10 0 0 64 28 0 1 2 0 0
282.457 215.843 1.7631 4.99876 8 0 0 2 8 1 2 6 169 49 0 0 0 0 1 22 158 31 0 6 11 2 0 7 0 0 0 2 6 4 8 4 27 5 0 4 14 4 1 2 1
51.1971 232.415 1.7399 5.3494 68 0 0 0 0 0 0 0 28 177 4 2 11 1 0 0 73 15 3 4 59 9 0 0 1 0 0 1 32 3 0 0 0 62 0 0 0 0 0 0 10 1
341.409 225.724 1.80272 4.66524 2 0 0 1 0 0 0 0 171 8 0 0 0 0 0 4 171 14 0 0 1 0 0 13 17 0 0 0 12 4 0 1 2 0 0 2 1 0 0 0 17
51.3785 232.357 1.82492 5.39508 88 0 0 0 0 0 0 0 42 175 5 3 8 0 0 0 65 12 3 7 60 2 0 0 1 0 0 3 31 1 0 0 1 84 0 0 0 0 0 0 19
245.945 218.21 2.38153 4.49406 12 3 0 0 0 0 0 0 1 148 1 0 0 0 0 0 111 148 0 0 0 4 2 0 117 1 0 0 0 14 5 0 2 10 4 0 0 1 1 0 0
79.5527 235.93 12.8767 5.00042 0 2 8 25 4 0 0 0 117 81 7 3 0 0 0 0 125 48 1 0 0 0 0 48 110 0 0 0 0 0 0 0 103 11 7 6 33 7 0 0
318.304 207.622 14.5882 4.79108 28 0 0 8 43 2 0 23 148 10 0 1 1 1 1 106 90 5 0 4 3 1 1 34 148 0 0 0 0 0 0 74 37 6 1 24 47
```

图 3.9　提取的关键点描述子

3.2 KMeans 聚类生成词典

特征库可视为特征向量的集合，这些特征向量集对应图片的基本信息，为了简化生成的特征库，需要将这些特征向量进行归类.由于这些特征向量是基于欧式空间的 128 维向量,KMeans 算法作为一种基于距离和划分的聚类算法,可以在欧式空间中对这些特征向量进行归类。特征库中所有特征向量经过 K-Means 算法聚类后生成基本词库 H，由于单词组合文本替代了难以描述化的图片，系统便可依据此文本库将对应的图片识别分类，图（）就是这个过程的示意图。

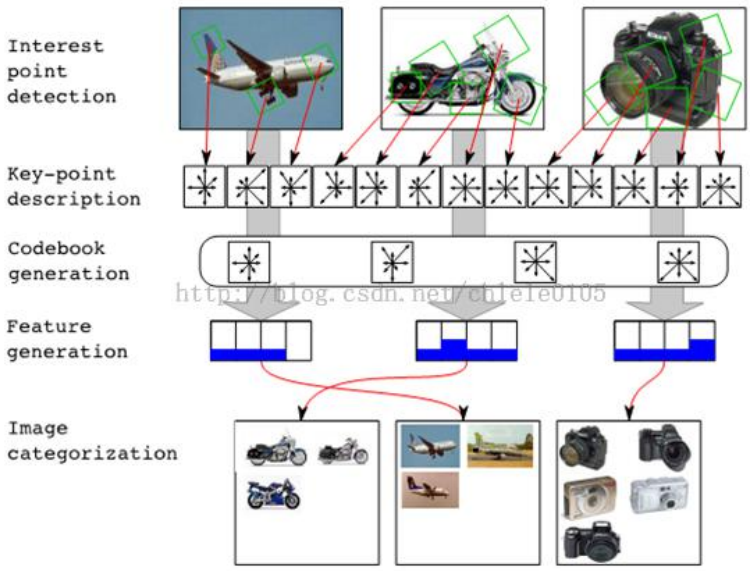

图 3.10　生成图像表示流程图

KMeans 就是把 n 个点（可以是样本的一次观察或一个实例）划分到 k 个聚类中，使得每个点都属于离他最近的均值（此即聚类中心）对应的聚类，以之作为聚类的标准。这个问题将归结为一个把数据空间划分为 Voronoi cells 的问题。这个过程依据分类精度和类别数量需要经过若干步，下面是具体实现细节。

在实现上使用 OpenMP 多线程并行计算，首先用一个线程控制启动，然后启动多个线程分别计算多个聚类点，最后把计算结果返回给第一个线程……如此反复运行。

启动 OpenMP 最大数量的线程：

```
nthreads = omp_get_max_threads();
```

然后开始用多线程对 K 个聚类点进行计算

```
#pragma omp parallel for \
        private(i,j,index) \
```

```
            firstprivate(numObjs,numClusters,numCoords) \
            shared(objects,clusters,membership,newClusters,newClusterSize) \
            schedule(static) \
            reduction(+:delta)
    for (i=0; i<numObjs; i++) {
        /*找到最近邻点  */
        index = find_nearest_cluster(numClusters, numCoords, objects[i],
                                                clusters);
        /*  记录族类发生改变  */
        if (membership[i] != index) delta += 1.0;
        /*  设定改变的族类  i */
        membership[i] = index;
        /*  更新族类的点  */
        #pragma omp atomic
        newClusterSize[index]++;
        for (j=0; j<numCoords; j++)
            #pragma omp atomic
            newClusters[index][j] += objects[i][j];
    }
```

经过上面过程我们得到了单词字典，总共有 K 个聚类点，每行都是代表本聚类点所对应的 SIFT 描述子。

```
0 -0.097357 -0.076102 -0.126336 -0.171088 -0.121801 0.308518 0.326167 -0.063486 0.016237 -0.095293 -0.123690 -0.15
1 -0.166838 -0.165755 -0.085816 0.173944 0.173414 -0.051683 -0.131266 -0.154495 0.196135 -0.080857 -0.120879 -0.01
2 -0.091813 -0.148502 -0.140010 0.039359 0.291309 -0.020033 -0.134332 -0.123861 0.216537 0.073208 -0.070279 -0.065
3 0.015057 -0.044997 -0.113936 -0.137909 -0.055444 0.178434 0.283582 0.111992 -0.141056 -0.102473 -0.064311 -0.003
4 -0.076568 -0.121759 -0.116010 -0.083502 -0.068545 -0.065171 -0.082899 -0.059457 -0.044101 -0.034400 0.051822 0.0
5 -0.081359 -0.059276 -0.005679 -0.068801 -0.127948 -0.091859 -0.010354 -0.037754 -0.117027 -0.052511 0.041765 0.0
6 -0.081708 -0.139981 -0.162501 -0.147908 -0.076610 0.012604 0.042070 -0.019250 0.064054 -0.042774 -0.099726 -0.10
7 -0.103747 0.038681 0.315299 0.111178 -0.136806 -0.164911 -0.141244 -0.139580 -0.177549 -0.020981 0.483824 0.4035
8 -0.032340 -0.148862 -0.168274 -0.169196 -0.144932 -0.035140 0.192724 0.184979 0.050040 -0.130158 -0.178977 -0.19
9 -0.015620 -0.148863 -0.163370 -0.135019 -0.088074 -0.074402 -0.015543 0.095629 0.308775 -0.060886 -0.154674 -0.1
10 -0.079259 -0.177131 -0.173328 -0.143017 -0.104490 0.034284 0.250453 0.186465 0.025802 -0.161795 -0.182910 -0.17
11 -0.067256 -0.036734 -0.039504 -0.056479 -0.080268 -0.084062 -0.073066 -0.102093 -0.118674 -0.108785 -0.038613 0
12 0.067102 -0.122925 -0.170267 -0.161902 -0.102328 -0.074500 -0.048065 0.042346 0.589146 0.098491 -0.156289 -0.17
13 -0.071423 -0.108823 -0.122797 -0.001199 0.202541 -0.030899 -0.127638 -0.105979 0.064451 -0.138444 -0.155682 0.0
14 -0.046494 0.041409 -0.021529 -0.104582 -0.125029 -0.111608 -0.128607 -0.122485 -0.054681 0.088271 0.045751 -0.0
15 -0.076860 -0.047091 -0.086676 -0.127068 -0.101491 -0.039679 -0.082126 -0.107741 0.158429 0.199809 -0.004738 -0.1
16 -0.091100 -0.097186 -0.105220 -0.106371 -0.083514 -0.063548 -0.085194 -0.113332 -0.044833 -0.071111 -0.089604 -0
17 -0.170396 -0.157232 -0.148415 -0.058936 0.294044 0.285441 -0.076998 -0.167392 0.024270 -0.106861 -0.155040 -0.1
18 -0.091385 -0.115980 -0.135146 -0.115941 -0.078246 -0.075532 -0.086054 -0.097619 -0.021097 0.078407 0.020402 -0.0
19 -0.069356 -0.065446 -0.105542 -0.107816 -0.067636 -0.050393 -0.090029 -0.099550 0.333130 0.201574 -0.069849 -0.1
20 -0.035740 -0.018602 -0.081708 -0.114201 -0.077222 -0.063934 -0.110963 -0.098473 -0.115312 -0.094627 -0.109513 -0
21 -0.120376 -0.141829 -0.140557 -0.126416 -0.094550 -0.062607 -0.047587 -0.076777 -0.027131 0.052607 0.044944 -0.0
22 -0.085283 -0.045802 -0.106034 -0.166873 -0.124596 0.244158 0.257359 -0.064998 -0.178396 -0.150392 -0.137499 -0.1
23 0.010503 -0.049413 -0.124676 -0.128324 -0.111212 -0.106279 -0.118201 -0.087109 0.292655 0.430354 0.089065 -0.03
24 -0.069323 -0.095738 -0.139663 -0.089904 0.212760 0.170973 -0.096444 -0.128066 0.212243 -0.031050 -0.147202 -0.1
25 -0.094197 -0.123718 -0.133399 -0.126947 -0.074090 0.058238 0.071826 -0.031006 -0.082774 -0.046142 -0.030613 -0.0
26 -0.148052 -0.140572 -0.149575 -0.098346 0.249055 0.400212 -0.050845 -0.161448 0.053011 0.132691 -0.120172 -0.14
27 -0.089309 -0.113955 -0.123557 -0.106083 -0.034810 -0.018093 -0.051482 -0.069009 0.309553 -0.000523 -0.122820 -0
28 -0.158169 -0.157583 -0.129247 0.036810 0.403418 0.149333 -0.140428 -0.174157 -0.087700 -0.135371 -0.155147 -0.0
29 -0.133444 -0.141513 -0.114164 0.027693 0.207124 0.016652 -0.118393 -0.135005 -0.000128 -0.001700 -0.027828 -0.0
```

图 3.11　获得的部分字典聚类

3.3 KD-tree 最近邻检索模块实现

一种用于高维空间中的快速最近邻和近似最近邻查找技术——Kd-Tree（Kd 树）Kd-Tree，即 K-dimensional tree，是一种高维索引树形数据结构，常用于在大规模的高维数据空间进行最近邻查找(Nearest Neighbor)和近似最近邻查找(Approximate Nearest Neighbor)。

Kd-Tree，即 K-dimensional tree，是一棵二叉树，树中存储的是一些 K 维数据。在一个 K 维数据集合上构建一棵 Kd-Tree 代表了对该 K 维数据集合构成的 K 维空间的一个划分，即树中的每个结点就对应了一个 K 维的超矩形区域（Hyperrectangle）。

Kd-Tree 的构建算法：

（1） 在 K 维数据集合中选择具有最大方差的维度 k，然后在该维度上选择中值 m 为 pivot 对该数据集合进行划分，得到两个子集合；同时创建一个树结点 node，用于存储<k, m>；

（2）对两个子集合重复（1）步骤的过程，直至所有子集合都不能再划分为止；如果某个子集合不能再划分时，则将该子集合中的数据保存到叶子结点（leaf node）。

实现代码如下：

```
def KDTree(point_list,depth=0):
    try:
        k = len(point_list[0])
    except IndexError as e:
        return  None
    axis = depth%k
    point_list.sort(key = itemgetter(axis))
    median = len(point_list) //2
    _left_child = KDTree(point_list[:median], depth + 1)
    _right_child = KDTree(point_list[median + 1:], depth + 1)
    node = Node(axis, point_list[median], _left_child, _right_child,None)
    if node.left_child != None:
        node.left_child.parent = node;
    if node.right_child != None:
        node.right_child.parent = node
    return node
```

3.4 本章小结

本章主要目的是实现了关键帧的并行提取、特征点的并行提取以及构建字典和图片单词向量。对 KMeans 算法进行了实现；对 SIFT 特征点的提取进行了实现；利用 KDtree 对图像进行了快速检索。

结　论

论文介绍了视频检索的发展，阐明了其特点和原理，通过测试可以得到以下结论：

（1）使用 OpenMP，能有提高并行程度、加速计算。

（2）结合 SIFT 和 KMeans 生成图像描述向量，能够有效进行图像检索。

（3）KMeans 维度越多检索效果越好但是也更慢，需要更好的并行检索算法。

论文设计实现的基于内容的多线程视频检索系统、数据提取模块、数据分析模块能够达到本来目的要求，但还存在一些不足之处。首先，系统各功能模块结构松散，提取数据后不能够实现数据处理和数据分析的自动化；再次使用 SIFT 算法泛化能力不够好，如果数据库中视频变化巨大需要重新生成字典。

在未来的系统研究过程中，可以根据需求完善系统的相关功能，应用机器学习等人工智能研究领域的新技术，设计实现功能全面的智能视频检索系统，实现数据处理和分析自动化，使得系统更加智能，减少人工维护成本。当然论文需要改进的地方还有很多，希望可以不断改进、完善。

参 考 文 献

[1] https://github.com/brookicv
[2] https://zhuanlan.zhihu.com/p/34890676
[3] http://www.ece.northwestern.edu/~wkliao/Kmeans/index.html
[4] https://blog.csdn.net/yuleishu0077/article/details/81983827
[5] https://blog.csdn.net/steven_miao/article/details/51942343
[6] http://www.vlfeat.org/