基于GPU加速的快速图像相似区域查找

汤 颖 肖廷哲 范 菁
(浙江工业大学计算机科学与技术学院 杭州 310023)

摘 要 图像相似区域查找是很多图形图像应用中的关键问题,也是计算瓶颈。传统加速方法如ANN(Approximate Nearest Neighbor)处理较大图像区域时速度较慢,而且在非度量空间下不支持精确查找。提出基于GPU加速的图像相似区域并行查找的通用计算框架,该框架可以扩展,以支持任意距离函数。特别针对在图像处理中应用广泛的欧氏距离(度量空间)和Chamfer距离(非度量空间)分别提出了基于CUDA的高效相似区域查找算法,比较完备地给出了相似性计算在不同度量空间下的实现。进一步,在设计具体的CUDA加速算法中,结合不同距离计算的特点对并行计算过程进行优化。该方法采用穷举的查找策略,在欧氏距离和Chamfer距离下都能实现精确查找,且大大提高了计算效率。实验结果表明,加速算法在准确查找的基础上执行速度比传统加速方法提升了一至二个数量级,且应用于纹理合成的实例表明,算法可以快速合成高质量的纹理。

关键词 度量空间,图像相似区域,GPU,Chamfer距离,纹理合成
中图法分类号 TP391 **文献标识码** A

GPU-based Fast Search of Similar Patches in Images

TANG Ying XIAO Ting-zhe FAN Jing
(Computer Science and Technology College, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract Many image processing or computer graphic applications involve similar patches search in images, which is a very computational-expensive operation. Traditional acceleration methods such as ANN (Approximate Nearest Neighbor) do not support exact search in non-metric space, and the searching time becomes longer for high-dimensional space. In this paper the general GPU-based framework to compute the similar patches was proposed which can be extended to incorporate any distance functions. Specifically, both Euclidean distance and non-metric Chamfer distance are adopted to compute the similarity between two patches. We proposed the GPU-based algorithm for fast search of similar patches in images under these two distances. Besides, the distances computation is optimized to achieve more efficient CUDA implementation. This algorithm supports exact search since exhaustive search strategy is adopted. The distances between patches are computed in parallel on GPU to greatly improve the computational efficiency. Experimental results show that our method has one or two orders of magnitude speedup compared with traditional acceleration methods and the results of applications in texture synthesis show that our method is capable to synthesize high-quality textures with fast speed.

Keywords Metric space, Similar patches, GPU, Chamfer distance, Texture synthesis

1 引言

数字图像由像素组成,多个相邻的像素组成的局部区域能表达丰富的信息。许多与图像有关的应用需要利用区域间的关联,即区域之间的相似关系,比如纹理合成[1]、图像编辑[20]、基于内容重用的图像压缩[3]等。图像相似区域查找是处理单幅图像内或多幅图像间各个区域之间的相似关系的方法,即从图像空间中找出与某个区域的距离小于给定阈值的所有区域。本质上,我们可以认为这是一个高维空间内的相似点查询问题,如图1所示,局部区域可以表示成一个高维向量,那么图像中所有的局部区域组成了一个高维数据集。查找图像内相似的区域,相当于在高维数据集内找到与某点距离近的点。区域之间的相似程度取决于如何定义距离函数,距离越小表示越相似。最常用的距离函数是欧氏距离,即区域间所有对应像素颜色值差的平方和再开平方根。除了欧氏距离,非度量的Chamfer距离对形状和边缘有很好的区分性,可对形状匹配进行有效的应用[4,5],所以图像中物体形状或结构的相似性常使用Chamfer距离来表示。Bonneel等人在文献[6]中使用Chamfer距离定义区域的相似性,相较于欧氏距离可以更好地捕获区域内的结构信息。

到稿日期:2013-04-13 返修日期:2013-07-12 本文受国家自然科学基金(61003265)资助。
汤 颖(1977—),女,博士,副教授,硕士生导师,CCF会员,主要研究方向为计算机图形图像、虚拟现实和信息可视化,E-mail:ytang@zjut.edu.cn;肖廷哲(1988—),男,硕士生,CCF学生会员,主要研究方向为计算机图形图像和高性能计算;范 菁(1969—),女,博士,教授,硕士生导师,CCF理事,主要研究方向为虚拟现实、软件工程和人工智能(通信作者)。

实际应用中的图像区域所含像素通常有几十个到上百个，甚至更多，距离计算量大且复杂，导致图像相似区域查找成为计算瓶颈。针对图像相似区域查找加速的方法有很多，但是现有的加速方法大部分仅仅考虑了度量空间下的距离。针对非度量空间查找的加速方法只能进行近似查找。

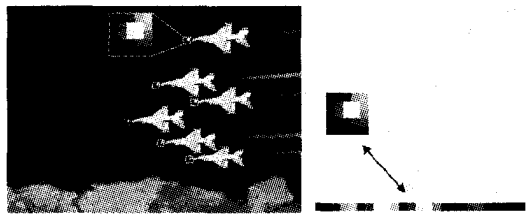


图 1　右图表示图像相似区域查找，局部区域是由多个像素组成；左图表示将局部区域对应到高维向量，向量的每个分量为像素的颜色值

本文基于距离计算的并行性提出了基于 GPU 加速的图像相似区域并行查找的通用计算框架，该框架与具体的距离函数无关，可以扩展到任意距离度量。在该通用框架的基础上，本文同时考虑了度量空间和非度量空间内的相似区域查找的设计和实现。具体来讲，我们针对在图像处理中应用广泛的欧氏距离和 Chamfer 距离分别设计了基于 CUDA 的高效相似区域查找算法，较完整地给出了不同距离空间下的加速计算。此外，在设计具体的加速算法时，我们对不同距离的特点进行分析，给出了基于距离计算优化的高效并行算法。总体来说，本文方法采用穷举查找策略，在不同距离空间内实现了精确的快速图像相似区域查找，速度上较传统的方法有很大的提高。

2　相关工作

图像相似区域查找是纹理合成[1,10-13]的一个重要计算过程。纹理合成根据输入纹理样图，通过相似区域查找合成任意大小的视觉上与样图相似但又不重复的图。除了纹理合成，图像相似区域查找还是其他很多有关图像或视频处理方法中的核心方法。一些技术使用局部区域去匹配图像或视频中的物体[14-16]；文献[17]使用相似区域找出具有相似形状的物体；文献[18]则使用相似区域的加权平均对图片降噪。相似区域的查找还可以用于图像超分辨率合成[19]、图像修改[20]、图像压缩[2]和图像的约束合成[2,5,21]等。上述应用大部分使用欧氏距离来表示区域相似性，除了欧氏距离，还有马氏距离[15]、曼哈顿距离[17]和 Chamfer 距离[5]。

对图像相似区域查找的加速已有很多的方法，较为常用的方法是基于树形结构的查找方法，该方法预先对查找域进行划分以减少查找范围来进行加速，如 TSVQ[10]，kd-trees[1]和 VP-trees[7]。这些方法都支持精确查找和近似查找。图像的每个局部区域通常会包含几十个或上百个像素，属于高维数据，上述加速方法对于高维数据的精确查找一般来说是非常耗时的，因此在实际应用中常使用近似查找来加速。除了基于树形结构的近似查找外，还有针对高维数据的降维技术，如 PCA[1,11,12]，其通过降低数据维度来进行查找加速，其但这会降低区域匹配的精度。在纹理合成中，Ashikhmin[13]提出了 coherence 搜索策略进行邻域匹配加速，利用局部连贯性(local coherence)减小搜索范围，提高了相似区域的查找速度。Tong 在文献[22]中提出了 k-coherence 搜索方法，它和文献[13]中的方法类似，但对搜索范围进行了扩展。

树形结构的查找加速方法中定义的距离函数必须满足对称、非负和三角不等式，即距离函数需要在度量空间内定义[7]。而 Chamfer 距离不满足三角不等式，所以文献[7]中提到的树形结构加速方法不适用对于 Chamfer 距离的加速。ANN[23]和 FLANN[24]等基于树形结构加速的类库没有对 Chamfer 距离提供支持。Chamfer 距离各个维度对距离的影响是不独立的，无法使用 PCA 降维来加速。Chamfer 距离比欧氏距离计算复杂，为了加速计算 Chamfer 距离，文献[9,25]中将 Chamfer 距离近似映射到向量空间，使其可以使用欧氏距离来近似表示，采用这种方法只能得到近似的查找结果。k-coherence 搜索方法对于 Chamfer 距离是适合的[5]，但它也是一种近似的方法且仅适用于纹理合成。Athitso 等人在文献[26]提出一种适合任意距离的近似查找方法。就目前的一些加速方法来看，适合非度量的加速方法都是近似查找。

随着计算机图形技术的发展，可编程图形处理器(GPU)提供了越来越强大的并行计算能力。Lefebvre 和 Hoppe 提出了基于 GPU 的并行可控纹理合成算法[11]，其利用 GPU 编程语言 HLSL 第一次完全在 GPU 上实现了纹理合成，大大加快了合成速度。文献[21]给出了基于 CUDA 加速的邻域匹配框架，这个框架结合了基于全局搜索的精确查找和基于 k-coherence 搜索的近似查找。虽然文献[21]中使用 CUDA 对图像相似区域查找进行了很好的加速，但也仅针对欧氏距离。文献[28,29]采用 GPU 对运动估计中的相似块计算进行加速，方法设计简单，没有针对具体距离特点进行优化，且也没有考虑非度量空间下的距离加速。本文专门针对图像相似区域查找的加速进行讨论，在 GPU 上完成度量空间与非度量空间的并行加速。

3　可扩展的并行加速框架

设有图 A 和图 B，A 中有 n 个区域 $\{R_1,R_2,\cdots,R_n\}$，B 中有 m 个区域 $\{Q_1,Q_2,\cdots,Q_m\}$。图像相似区域查找是为 B 中每一个区域从 A 中找出前 k 个与其最相似的区域。图 2 表示纹理合成中的图像相似区域查找[10]，样图对应图 A，合成图对应图 B。区域间的相似性由区域间的距离来定义，距离越小表示越相似。那么最直接的方法是全局搜索的图像相似区域查找，表 1 给出了该方法的伪代码。如伪代码所示，该方法有两层循环。内层循环是计算 B 中一个区域 Q_j 与 A 中所有区域的距离 $d(Q_j,R_i)$，$i=1\cdots n$，并保存在距离集合 D_j 中；然后排序这些距离 $sort(D_j)$，取前 k 个最小距离对应的 A 中的区域 $P_{1\cdots k}(j)$，公式表示为 $P_{1\cdots k}(j)\leftarrow sort(D_j,k)$。外层循环表示的是逐个对 B 中的所有区域进行上述操作。全局搜索的图像相似区域查找，在查找相似区域时，会比较搜索范围内的所有区域，所以无论使用什么距离度量都可以得到最精确

的结果；但是计算量庞大且很耗时。

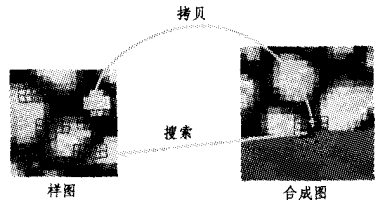


图 2　纹理合成：从样图的 L 形区域集合中找出与合成图中红色 L 形区域最相似的区域

表 1　顺序执行的全局搜索的图像相似区域查找

```
for each patch Qj in B
  distance set Dj
  patch set P1…k(j)
  for each patch Ri in A
    put d(Qj,Ri) in Dj
  end
  P1…k(j)←sort(Dj,k)
end
```

全局搜索的图像相似区域查找有很好的并行性。本文的通用并行加速计算框架基于算法本身的并行性使用 GPU 加速，不仅减少了执行时间，还保证了结果的精确性。GPU 并行加速的粒度较小，一般来说将任务分割成尽量小而多的并行执行的子任务，会取得更好的加速效果。在全局搜索的相似区域查找中，对于 B 中的区域查找相似区域是独立的，即伪代码中的外层循环是可以并行执行的；任意区域间的距离计算都是独立的，则伪代码中的内部循环也是可以并行的。

基于上述并行性分析，图像相似区域查找可分成两步进行 GPU 加速，如表 2 伪代码所示。第一步内外两层循环同时并行，计算 A 中所有区域到 B 中所有区域的距离，得到距离集合 D，公式表示为 $D=\{d(R_i,Q_j)\mid i=1,\cdots,n,j=1,\cdots,m\}$。这里将距离的计算分成 $n\times m$ 个并行执行的子任务，每个子任务计算一对 AB 间区域的距离 $d(R_i,Q_j)$。第二步外层循环平行，通过查找距离集合 D，为 B 中的每个区域在 A 中找到前 k 个与其距离最小的区域。D 分成 m 个子集，这些子集为 $D_j=\{d(R_i,Q_j)\mid i=1,\cdots,n,j=1,\cdots,m\}$，分别对应 B 中一个区域 Q_i。本步骤分成 m 个并行执行的子任务，每一个子任务查找一个子集 D_j，为相应的区域 Q_i 找到前 k 个相似的区域。

表 2　并行执行的全局搜索的图像相似区域查找

```
//第一步计算距离
distance set D
for each patch Qj in B do in parallel
  for each patch Ri in A do in parallel
    put d(Qj,Ri) in D
  end
end
//第二步查找前 k 个距离最小的区域
similar patch set P1…k
for j from 1 to m do in parallel
  P1…k(j)←sort(Dj,k)
end
```

上述并行的方法只需修改度量函数 $d(Q_j,R_i)$ 就可以扩展到任意的距离，可以在没有损失精确度的情况下提高计算速度。由此可见，此并行加速框架不同于一些只适用于度量空间的基于剪枝思想的加速方法[7]，它适用于任意的距离函数。

在距离的定义中，欧氏距离是图像/视频处理中最常用的距离度量函数，它被广泛用于纹理合成、视频编码和视频特征跟踪和匹配中；此外，非度量空间的 Chamfer 距离也在图像处理中有着重要的作用，它经常用于图像边缘、结构特征等的匹配。下面两节我们分别给出在这两种距离函数下结合距离计算过程的具体 CUDA 加速算法的设计和实现。

4　欧氏距离加速

将区域 R 和 Q 看成向量 $R=(r_1,r_2,\cdots,r_d)$ 和 $Q=(q_1,q_2,\cdots,q_d)$，向量的分量就是区域中所含像素的颜色值。区域间的欧氏距离计算如式(1)所示，即像素颜色差的平方和再开根号。使用基于欧氏距离的图像相似区域查找可以得到颜色相似的区域。

$$d(R,Q)=\|R-Q\|=\sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\cdots+(p_d-q_d)^2} \tag{1}$$

图像相似区域查找是先对距离排序，然后取距离最小的前 k 个结果，所以使用欧氏距离的平方对结果没有影响。本文区域的相似性使用欧氏距离的平方来定义。那么式(2)表示，对于图 B 中的每个区域 Q，根据欧氏距离的平方找到 k 个与其最相似区域的集合 $C_{1\cdots k}(Q)$。

$$C_{1\cdots k}(Q)=\underset{R\in A}{Sort}\|R-Q\|^2 \quad (R\in A,Q\in B) \tag{2}$$

根据式(2)分解该运算，得到式(3)：

$$\begin{aligned}C_{1..k}(Q)&=\underset{R\in A}{Sort}(\|Q\|^2+\|R\|^2-2R\cdot Q)\\&=\underset{R\in A}{Sort}(\|R\|^2-2R\cdot Q) \quad (R\in A,Q\in B)\end{aligned} \tag{3}$$

从式(3)中可以看出，图 A 中所有区域到图 B 中某一区域 Q 的距离计算公式都含有 $\|Q\|^2$ 这一项，那么这些距离的大小关系只依赖于 $\|R\|^2-2R\cdot Q$。所以通过对距离排序从图 A 中查找出前 k 个与区域 Q 最相似的区域，只需要计算 $\|R\|^2-2R\cdot Q$ 部分即可。

因为任意两区域的距离计算是独立的，很适合并行计算，所以利用 GPU 加速先计算出图 B 中每个区域与图 A 中所有区域的距离，得到矩阵 $DMatrix$，矩阵的列数为图 B 中区域的个数 m，行数为图 A 中区域的个数 n。为了方便 GPU 加速，把图像中的区域进行整合，用矩阵的形式存储。图 A 的区域整合成矩阵 $RMatrix=|R_1,R_2,\cdots,R_n|$，大小为 $d\times n$，每一列表示图 A 中的一个区域。同样图 B 的区域整合成矩阵 $QMatrix=|Q_1,Q_2,\cdots,Q_m|$，大小为 $d\times m$，每一列表示图 B 中的一个区域。计算 $\|R\|^2-2R\cdot Q(R\in A,Q\in B)$ 等同于先计算矩阵 $RMatrix$ 列向量的模方 $\|RMatrix\|^2$，然后再减去两个矩阵乘的两倍即 $-2*RMatrix^{\mathrm{T}}*QMatrix$，得到的结果为矩阵 $DMatrix$。

基础线性代数程序集(BLAS)是高度优化的线性代数的程序库[27]，提供向量和矩阵运算，并且有各种系统平台上的实现。CUBLAS 是 NVIDIA 提供的使用 CUDA 实现的并行

的 BLAS，较传统的 BLAS 函数有更好的性能。所以我们调用 CUBLAS 提供的矩阵乘法函数来计算 $-2 * RMatrix^{T} * QMatrix$。在 CUDA 框架下，程序被分为主机端和设备端，在主机端执行 CPU 上的操作，在设备端执行 GPU 上的操作。内核函数(Kernel)为 CUDA 运行在设备端的并行计算函数。内核函数只能对显存的数据进行操作。所以使用 GPU 计算距离，需要将 *RMatrix* 和 *QMatrix* 拷贝到显存，得到的 *DMatrix* 也存储在显存。考虑到当区域的数量很多时，*DMatrix* 会很大，显存将无法容纳，因此分批进行图片相似区域查找，一次只处理图 B 中的部分区域。算法根据现有显存大小，动态确定当前可处理的区域个数。对于每一次处理，计算图 B 中的部分区域和图 A 中所有区域的距离，然后采用插入排序法找到对应前 k 个最小距离的区域。具体计算步骤如下：

Step1　整合区域得到 *RMatrix* 和 *QMatrix*，在设备端分配存储空间，把 *RMatrix*、*QMatrix* 拷贝给设备端；在主机端分配空间 *indexMatrix*，存储计算得到的最相似区域的标示。

Step2　查找显存中剩余空间，确定每次可同时处理图 B 中区域的数量 max *QueryNum*。

Step3　根据 max *QueryNum*，对 *QMatrix* 分块，对 *QMatrix* 的每一个分块 *subQMatrix*：

Step3.1　计算 *RMatrix* 矩阵列向量的模方 $\| RMatrix \|^2$，得矩阵 *Matrix*1；

Step3.2　使用 CUBLAS 的矩阵乘法函数计算 $-2 * RMatrix^{T} * subQMatrix$，得矩阵 *Matrix*2；

Step3.3　求和得到图 B 中部分区域的距离 *subDMatrix* = *Matrix*1 + *Matrix*2；

Step3.4　对 *subDMatrix* 的每一列进行插入排序 *Sort*(*subDMatrix*)，列间并行加速，取每一列前 k 个距离 $C_{1\cdots k}(Q)$；

Step3.5　将 $C_{1\cdots k}(Q)$ 中区域的标示拷贝到 *indexMatrix* 对应位置的内存中。

图像相似区域查找的 GPU 加速关键在于区域间距离的并行计算，本节通过分解欧氏距离平方的公式把距离计算转换成矩阵乘等运算，充分利用了 GPU 的并行计算能力。我们已将上述的欧氏距离加速应用于图像类比[21]，在速度和效果上都有明显的提高。

5　Chamfer 距离加速

在对图像中物体的查找、定位、识别等应用中，物体的形状和边缘往往比颜色更重要。Chamfer 距离对形状和边缘有很好的区分性，可以有效应用于形状匹配中，相较于欧氏距离可以更好地捕获区域内的结构信息。图 3 的第一行和第二行分别给出了两组图像区域，通过观察可知，第一行的两个图像区域明显比第二行更相似，即距离更近。但是图 3 中的计算结果显示这两组图像区域在欧氏距离得到的比较结果与直觉相反，第二行的欧氏距离小于第一行；而采用 Chamfer 距离可以得到符合我们观察的结果，即第一组的 Chamfer 距离更小，更相似。

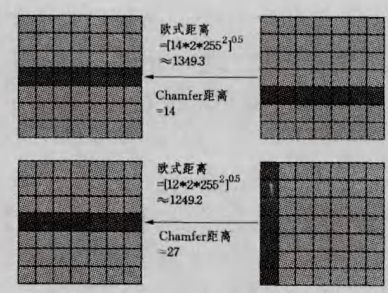


图 3　分别计算每层区域间的欧氏距离和 Chamfer 距离，根据欧氏距离判定下层的两个区域更相似，根据 Chamfer 距离判定上层的两个区域更相似

为了计算图像区域间结构匹配的相似性，文献[5]定义 Chamfer 距离为区域 Q 所有像素与区域 R 中标签值相同的像素之间最小的 L_∞ 距离的总和。如果 R 中没有相同标签值的像素，这个像素的距离为区域大小的两倍。根据定义来看，Chamfer 距离是不对称的。如果要使 Chamfer 距离对称，Q 到 R 的单向距离加上 R 到 Q 的单向距离，得到双向距离。设区域的向量表示：$R=\{r_{00}, r_{01}, \cdots, r_{xy} \cdots, r_{dd}\}$，$Q=\{q_{00}, q_{01}, \cdots, q_{ij} \cdots, q_{dd}\}$，区域为方形，大小为 $(d+1)\times(d+1)$，ij 和 xy 表示像素在区域中的坐标，q_{ij} 和 p_{xy} 表示对应像素的标签值(程序实现中标签值就是像素的颜色值)，则 Q 到 R 的 Chamfer 距离如式(4)表示：

$$d_{QR}=\sum_{i=0}^{d}\sum_{j=0}^{d}\{d_{ij} \mid (d_{ij}=\max(|i-x|,|j-y|)\cap q_{ij}=r_{xy}) \cup (d_{ij}=2\times d \cap q_{ij}\neq r_{xy}), 0\leqslant x\leqslant d, 0\leqslant y\leqslant d\}_{\min} \tag{4}$$

$\{\cdot\}_{\min}$ 表示集合中最小的元素。表 3 为 Chamfer 距离计算的伪代码。从伪代码可以看出，Chamfer 距离计算非常复杂，有多重嵌套循环。计算所有区域间的距离，还需要加上对区域遍历的循环嵌套，这样的循环嵌套耗时将是巨大的。在纹理合成中，对 64×64 分辨率的图片进行基于全局搜索的 Chamfer 距离下的相似区域查找(区域大小为 5×5)将有百亿次循环嵌套。

表 3　Chamfer 距离计算的伪代码

```
function ChamferDistance(Q,R)
  dQR=0;
  for i=0:1:d do
    for j=0:1:d do
      d=2×d;
      for x=0:1:d do
        for y=0:1:d do
          if qij==rxy then temp=max(|i-x|,|j-y|);
          if temp < d then d=temp;
        end
      end
      dQR+=d;
    end
  end
return dQR
```

Chamfer 距离不是度量距离，不符合三角不等式，不适合使用传统的树形结构的加速，GPU 加速是一种很好的解决途径。Chamfer 距离的加速与欧氏距离的加速大体一致，使用 GPU 并行计算每个区域间的距离，然后排序，得到相似的区

域。但是由式(4)可以看出,Chamfer 距离的计算比欧氏距离更复杂,它的各维度不是独立影响距离大小的,对于它的计算没有类似欧氏距离那样的分解方法,这里我们用一个内核函数来计算。

GPU 并行计算区域间的距离前,先要进行数据组织。将图片中的区域整合成矩阵,矩阵的列数为区域的个数,行数为区域所含像素的个数,即维度。如式(4)所示,Chamfer 距离的计算不仅与像素的颜色值有关,还与像素在区域中的相对位置有关。所以用矩阵表示区域集合时,矩阵的每个元素要包含两个信息,即像素颜色值和像素在区域中的相对坐标。与欧氏距离加速一样,将图 A 和图 B 的区域分别整合成矩阵 $RMatrix=|R_1,R_2,\cdots,R_n|$ 和矩阵 $QMatrix=|Q_1,Q_2,\cdots,Q_m|$。

CUDA 编程模型把线程组织成线程、线程块、线程块网格的层次结构,每个线程有一个局部存储器(local memory),每个线程块有一个共享存储器(share memory),能被该线程块中所有线程使用。所有线程都能访问全局存储器(global memory)。CUDA 访问全局存储器需要 400～600 个时钟周期,但访问共享存储器仅需 4 个时钟周期;每个线程块包含线程数也是有限制的。因此在算法设计中,应对线程块、线程块网格进行合理划分,采用共享内存来保存每个线程块计算所需的信息,减少访问全局存储器的次数。GPU 中除了上述 3 种存储器,还有纹理储存器(texture memory)。纹理存储器是只读存储器,但它有纹理缓存,访问速度比全局存储器高。所以将区域矩阵加载到纹理存储器中,可以提高访存的速度。

矩阵 *RMatrix* 和 *QMatrix* 中元素的类型采用 uchar4,uchar4 的 x 和 y 表示像素相对坐标,z 表示像素的颜色值。*RMatrix* 和 *QMatrix* 绑定到纹理,存储在纹理存储器中。因为各个区域间的距离计算是无关联的,所以我们设计一个线程计算一对区域间的距离。那么计算距离的内核函数有 $n\times m$ 个线程,使得每个线程块有 16×16 个线程,则有 $n/16\times m/16$ 个线程块。每个线程块声明一个在共享存储器的数组来保存所要使用的区域数据。每个线程执行上述伪代码所示的距离计算,实现距离的并行计算。具体计算步骤如下:

Step1　整合区域得到 *RMatrix* 和 *QMatrix*,把 *RMatrix*、*QMatrix* 绑定到纹理,存储在纹理存储器,即拷贝到设备端;在主机端分配空间 *indexMatrix*,存储计算得到的最相似区域的标示。

Step2　查找显存中的剩余空间,确定每次可同时处理 B 中区域的数量 max *QueryNum*。

Step3　根据 max *QueryNum*,对 *QMatrix* 分块,对 *QMatrix* 的每一个分块 *subQMatrix*:

Step3.1　计算距离 *subDMatrix* = *chamferDisance*(*RMatrix*,*subQMatrix*);

Step3.2　对 *subDMatrix* 的每一列进行插入排序 *Sort*(*subDMatrix*),列间并行加速,取每一列前 k 个距离 $C_{1\cdots k}(Q)$;

Step3.3　将 $C_{1\cdots k}(Q)$ 中区域的标示拷贝到 *indexMatrix* 对应位置的内存中。

6　实验结果

6.1　实验环境与测试数据

实验硬件配置如下:CPU:Intel Xeon E5506 2.13GHz,内存:4G,显卡:NVIDIA Quadro 600。算法程序的软件平台采用 Visual Studio2008 及 CUDA4.0。

测试数据为一组图片,图片的像素分辨率分别为 16×16、32×32、64×64、128×128 和 256×256。在测试时,图 A 和图 B 是同一张图片,区域为图片中每个像素的邻域,那么图像相似区域查找就是对于图片内每个像素,在自身图片中找到与它的邻域最相似的前 k 个邻域。这个过程与基于 k-coherence纹理合成算法的预处理过程相似,可以为纹理合成提供搜索候选集。基于欧氏距离的区域相似性查找使用真彩色 RGB 图,那么每个区域向量的维度为其所含像素的个数的 3 倍。基于 Chamfer 距离的区域相似性查找使用灰度图,像素灰度值是用来作为标签的,其中区域向量的维度为所含像素的个数。

6.2　时间性能比较

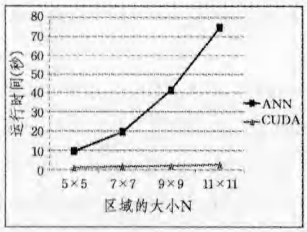
本小节通过实验比较了本文方法与传统的 CPU 加速方法的计算性能。基于欧氏距离的图像相似区域查找的传统加速方法很多,这里我们采用了 ANN 加速方法。基于 Chamfer 距离的图像相似区域查找采用文献[5]中提到的超球面加速方法。实验结果显示,本文方法比传统加速方法提升了 1 至 2 个数量级。

表 4 给出了 ANN 方法和 CUDA 方法处理不同大小图片的运行时间,待查找的最相似邻域个数 k 为 12;邻域大小 N 为 11×11。从表 4 可看出,在图片分辨率较小时,区域数量较少,CUDA 方法与 ANN 相比优势较小,因为采用 GPU 搜索时,计算距离及搜索比较的时间较短,大量时间耗费在 CPU 内存和 GPU 内存拷贝数据中,而 ANN 不需要进行数据拷贝。当图片分辨率增大时,区域数量也明显增多,计算距离的时间远远大于内存拷贝所需时间,CUDA 方法与 ANN 相比优势体现得较为明显。测试结果表明,CUDA 方法加速最明显时相比 ANN 方法有 20 多倍的加速比。图 4 给出了随着区域大小的变化,CUDA 方法和 ANN 运行时间的变化;图 5 给出了 CUDA 方法和 ANN 运行时间与最相似邻域个数 k 的关系。从图 4 和图 5 看出,区域大小 N 和最相似区域个数 k 的变化对 CUDA 方法的计算速度影响较小,CUDA 方法的运行效率在变化的参数设置下更加稳定。

表 4　基于欧氏距离的图像相似区域查找时间比较

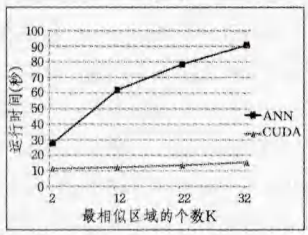
图片分辨率	16×16	32×32	64×64	128×128	256×256
ANN 运行时间(s)	0.311	1.529	10.896	74.645	412.339
CUDA 运行时间(s)	0.301	0.329	0.523	2.729	37.011

针对基于 Chamfer 距离的区域相似性计算,表 5 给出了采用 C++蛮力方法、超球面加速方法和本文 CUDA 方法的运行时间。其中邻域 N 大小为 5×5,待查找的最相似区域的个数 k 为 10。基于超球面的加速方法是很保守的,能够得到较精确的结果。



11×11 大小的区域，对应 11×11×3=363 维的数据，最相似邻域个数 k 为 12，图片大小为 128×128

图 4　区域大小 N 对查找时间的影响



区域大小 N 为 5×5，图片大小为 256×256

图 5　最相似邻域个数 K 对查找的影响

表 5　基于 Chamfer 距离的区域相似性查找时间比较

图片分辨率	16×16	32×32	64×64	128×128	256×256
C++蛮力 (s)	9.985	163.005	2585.951	42486.652	—
超球面加速 (s)	1.872	21.715	447.364	8594.688	—
CUDA (s)	0.127	0.236	2.148	31.346	488.076

从表 5 可以看出，CUDA 方法的性能优势明显，图片大小为 64×64 时，比 C++蛮力方法快上千倍，比超球面加速方法也有 200 多倍的加速。测试图片的大小是以 4 倍的方式增加的，每个像素的邻域为一个局部区域，所以区域的数量也是以 4 倍的方式增加。查找时需要计算两两区域之间的距离及对两两区域进行比较，计算量随着图片的增大以 16 倍的方式增大，如表 5 所列，BF-C++的计算时间大致符合这个规律。对于 CUDA 的时间在 GPU 使用率饱和(64×64)之后也是以约 16 倍的方式增加，但是 CUDA 方法的起步时间小(对于 64×64 的图片只需 2.148s)，所以 CUDA 方法的效率肯定大大高于 C++蛮力方法。超球面加速方法是通过裁剪搜索空间加速的，为了保证结果的精确性，裁剪的效率不高，加速效果没有 CUDA 方法好。对于 256×256 图片的处理，C++蛮力方法和超球面加速方法的耗时都是巨大的，按照表 5 中已测时间来估计它们的处理时间可能需要几天，这么长的处理时间意义已不大。图 6 给出了 CUDA 和超球面方法的加速比随着图片大小的变化情况，可以看出，CUDA 方法的加速优势随着区域数量的增加变得更为明显，数据量的增加可以更好地发挥 GPU 并行的能力，掩盖数据拷贝的延迟。

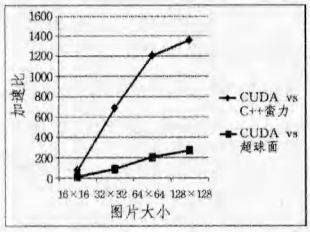


图 6　CUDA 方法分别与 C++蛮力方法和超球面加速方法的加速比

6.3　图像相似区域查找的应用实例——纹理合成

图像相似区域查找是纹理合成的核心算法，根据区域的相似性从样图选取像素或块来组成任意大小的纹理图，区域的相似性保证了合成的纹理图与样图的相似。图像相似区域的查找计算很耗时，这决定了纹理合成也是一个非常耗时的应用。将 CUDA 加速的快速图像相似区域查找方法应用于纹理合成，比 ANN 加速方法有明显的提高。本文对使用上述两种方法加速纹理合成进行了性能的测试。测试数据为 64×64 大小的样图，邻域大小为 11×11，合成大小分别为 64×64、128×128、192×192 和 256×256 的纹理图。实验结果如表 6 所列，当合成 256×256 的纹理图时 CUDA 方法计算效率比 ANN 方法将近提高了 20 倍。图 7 展示了 CUDA 方法与 ANN 方法的加速比，合成纹理图越大，CUDA 方法优势越明显；而且 CUDA 加速的纹理合成效果理想，如图 8 所示。

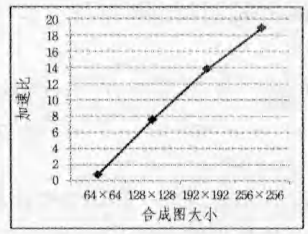


图 7　纹理合成应用中 CUDA 方法与 ANN 方法的加速比

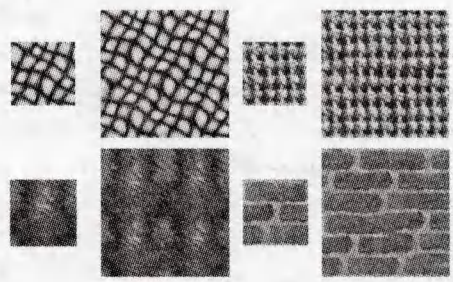


图 8　纹理合成实例

表 6　纹理合成的时间比较

合成图大小	64×64	128×128	192×192	256×256
ANN(s)	0.799	12.066	33.887	71.361
CUDA(s)	1.067	1.594	2.452	3.782

结束语　本文提出了基于 GPU 的通用图像相似区域查找加速框架，并分别针对度量空间的欧氏距离和非度量空间 Chamfer 距离进行了分析和优化，给出了基于 CUDA 的图像相似区域查找的设计和实现。文中对 CUDA 加速方法和 CPU 上传统的基于树形结构的加速方法进行了性能比较，对于基于欧氏距离的图像相似区域查找，CUDA 方法相比 ANN 方法有 20 多倍的加速；对于基于 Chamfer 距离的图像相似区域查找，CUDA 方法相比超球面方法有 200 多倍的加速，相比于 C++的蛮力实现更有上千倍的加速，速度方面有明显的优势。本文的 CUDA 加速方法可扩展至任意距离度量，且得到精确的结果。本文的方法虽然在精确度和速度上比传统的方法要好，但是受 GPU 显存的限制无法处理大图片和大区域。这是因为本文方法采用的是穷举搜索策略，需要将图 A 中所有的区域拷贝到显存，当图 A 中的区域过多过大时，显存将无法容纳。如果图 A 大小为 512×512，每个像素 33×33 大小的邻域为一个区域，那么需要超过 1G 的显存空间，本文实验使用的 NVIDIA Quadro 600 将无法处理。在

未来的工作中，我们需要考虑如何减少显存的消耗。在很多应用中，较精确的近似查找能得到比较满意的结果，如何将传统方法中裁剪搜索空间或降维的思想与 GPU 的并行加速结合起来从而得到更大规模图片的快速相似区域查找是未来的研究方向。

参 考 文 献

[1] Efros A A, Leung T K. Texture synthesis by non-parametric sampling[C]// Proceedings of the Seventh IEEE International Conference on Computer Vision. Greece, 1999: 1033-1038

[2] Hertzmann A, Jacobs C E, Oliver N, et al. Image analogies[C]// Proceedings of the 28th annual conference on Computer graphics and interactive techniques. New York, NY, USA: ACM, 2001: 327-340

[3] Wang H, Wexler Y, Ofek E, et al. Factoring repeated content within and among images [J]. ACM Trans. Graphics (SIGGRAPH), 2008, 27: 3

[4] Barrow H G, Tenenbaum J M, Bolles R C, et al. Parametric correspondence and Chamfer matching: two new techniques for image matching [C] // Proceedings of the 5th International Joint Conference on Artificial Intelligence-Volume 2. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1977: 659-663

[5] Borgefors G. Hierarchical Chamfer matching: a parametric edge matching algorithm[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1988, 10(6): 849-865

[6] Bonneel N, Van De Panne M, Lefebvre S, et al. Proxy-Guided Texture Synthesis for Rendering Natural Scenes[C]// Vision Modeling and Visualization Workshop (VMV 2010). Siegen, Allemagne: Eurographics Association, 2010: 87-95

[7] Kumar N, Zhang L, Nayar S. What Is a Good Nearest Neighbors Algorithm for Finding Similar Patches in Images? [J]. Computer Vision-ECCV 2008, 5303: 364-378

[8] Samet H. 多维与度量数据结构基础[M]. 周立柱，王宏，邓俊辉，等译. 北京：清华大学出版社，2011：461-698

[9] Athitsos V, Sclaroff S. Database Indexing Methods for 3D Hand Pose Estimation[C]// Gesture-Based Communication in Human-Computer Interaction: 5th International Gesture Workshop, GW 2003, Genova, Italy, April 15-17, 2003, Selected Revised Papers. Springer, 2004, 2915: 288

[10] Wei L Y, Levoy M. Fast texture synthesis using tree-structured vector quantization[C]// Proceedings of the 27th annual conference on Computer graphics and interactive techniques. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000: 479-488

[11] Lefebvre S, Hoppe H. Parallel controllable texture synthesis [J]. ACM Transactions on Graphics, 2005, 24(3): 777-786

[12] Lefebvre S, Hoppe H. Appearance-space texture synthesis [J]. ACM Transactions on Graphics, 2006, 25(3): 541-548

[13] Ashikhmin M. Synthesizing natural textures[C]// Proceedings of the 2001 symposium on Interactive 3D graphics. New York, NY, USA: ACM, 2001: 217-226

[14] Lowe D G. Distinctive Image Features from Scale-Invariant Keypoints[J]. International Journal of Computer Vision, 2004, 60(2): 91-110

[15] Sivic J, Zisserman A. Video Google: a text retrieval approach to object matching in videos[C]// Proceedings of the Ninth IEEE International Conference on Computer Vision. 2003: 1470-1477

[16] Nister D, Stewenius H. Scalable Recognition with a Vocabulary Tree[C]// Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2006, 2: 2161-2168

[17] Shechtman E, Irani M. Matching Local Self-Similarities across Images and Videos[C]// Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. 2007: 1-8

[18] Buades A, Coll B, Morel J-M. A non-local algorithm for image denoising[C]// Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2005, 2: 60-65

[19] Freeman W T, Jones T R, Pasztor E C. Example-based super-resolution[J]. IEEE Computer Graphics and Applications, 2002, 22(2): 56-65

[20] Barnes C, Shechtman E, Finkelstein A, et al. PatchMatch: a randomized correspondence algorithm for structural image editing [J]. ACM Transactions on Graphics-TOG, 2009, 28(3): 24

[21] Tang Y, Shi X, Xiao T, et al. An improved image analogy method based on adaptive CUDA-accelerated neighborhood matching framework[J]. The Visual Computer, 2012, 28(6): 743-753

[22] Tong X, Zhang J, Liu L, et al. Synthesis of bidirectional texture functions on arbitrary surfaces [J]. ACM Transactions on Graphics, 2002, 21(3): 665-672

[23] Mount D M, Arya S. ANN: Approximate Nearest Neighbor Library[EB/OL]. http://www.cs.umd.edu/~mount/ANN/, 2012-08-29

[24] Muja M, Lowe D G. Fast approximate nearest neighbors with automatic algorithm configuration[C]// Proceedings of International Conference on Computer Vision Theory and Applications (VISSAPP'09). Portugal: INSTICC Press, 2009: 331-340

[25] Athitsos V, Hadjieleftheriou M, Kollios G, et al. Query-sensitive embeddings[C]// Proceedings of the 2005 ACM SIGMOD international conference on Management of data. New York, NY, USA: ACM, 2005: 706-717

[26] Athitsos V, Potamias M, Papapetrou P, et al. Nearest Neighbor Retrieval Using Distance-Based Hashing [C] // Proceedings of IEEE 24th International Conference on Data Engineering. 2008: 327-336

[27] Basic Linear Algebra Subprograms[EB/OL]. Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms, 2012-09-11

[28] Monteiro E, Maule M, Sampaio F, et al. Real-time block matching motion estimation onto GPGPU[C]// 2012 19th IEEE International Conference on Image Processing (ICIP). IEEE, 2012: 1693-1696

[29] Massanes F, Cadennes M, Brankov J G. Compute-unified device architecture implementation of a block-matching algorithm for multiple graphical processing unit cards[J]. Journal of electronic imaging, 2011, 20(3)