MPEG-7 시각 정보 기술자의 특성을 반영한 효율적인 멀티미디어 데이타 비트맵 인덱싱 방법

(An Efficient Bitmap Indexing Method for Multimedia Data Reflecting the Characteristics of MPEG-7 Visual Descriptors)

정 진 국 † 낭 종 호 ††
(Jinguk Jeong) (Jongho Nang)

요 약 최근 멀티미디어 정보를 기술하기 위한 표준인 MPEG-7이 제안되어 이미지/동영상 검색 시스템과 같은 응용분야에서 사용되기 시작하였다. 그러나 MPEG-7 시각 정보 기술자들은 대부분 고차원으로 표현이 되고, 고차원에서 발생되는 문제인 "Curse of dimensionality" 때문에 기존의 인덱싱 방법(예를 들면 트리 구조를 이용하는 다차원 인덱싱 방법, 차원을 줄이는 방법, 양자화 등의 압축 기법을 이용하는 방법 등)으로는 효율적인 검색을 할 수 없다. 본 논문에서는 MPEG-7 시각 정보 기술자들의 특징을 반영한 효율적인 인덱싱 방법을 제안한다. 제안된 방법에서는 기술자를 속성 히스토그램으로 변형하고 히스토그램의 각 빈 값을 이진 형태로 표현하여 비트열을 생성하며, 이러한 비트열들을 이용하여 비트맵 인덱스를 구성한다. 질의 오브젝트가 입력되면 비트맵 인덱스를 이용하여 결과에 포함될 가능성이 있는 후보 오브젝트 리스트를 생성하게 되는데 즉, 각 오브젝트의 인덱스와 질의 오브젝트의 비트열에 대한 XOR (Exclusive OR) 연산을 수행하여서 후보 오브젝트 리스트를 생성한다. 그리고 이 리스트에 있는 오브젝트들에 대해서만 L1-norm과 같은, 기술자를 위해 사용되는 비교 연산식을 수행하여 최종 결과 오브젝트들을 사용자에게 보여주게 된다. 본 논문에서 제안하는 알고리즘은 단순한 비트 연산을 통해 검색 결과에 포함될 가능성이 있는 오브젝트들을 추출해낼 수 있기 때문에 빠른 시간 내에 검색을 마칠 수 있도록 해준다. 실험에 의하면 제안한 방법을 이용하는 경우, 90% 이상의 정확도를 유지하면서 검색 시간에서는 순차 검색에 비해 15배 이상의 속도 향상을 보임을 알 수 있었다.

키워드 : 멀티미디어 인덱싱, 멀티미디어 내용기반 검색, 비트맵 인덱싱

Abstract Recently, the MPEG-7 standard a multimedia content description standard is widely used for content based image/video retrieval systems. However, since the descriptors standardized in MPEG-7 are usually multidimensional and the problem called "curse of dimensionality", previously proposed indexing methods(for example, multidimensional indexing methods, dimensionality reduction methods, filtering methods, and so on) could not be used to effectively index the multimedia database represented in MPEG-7. This paper proposes an efficient multimedia data indexing mechanism reflecting the characteristics of MPEG-7 visual descriptors. In the proposed indexing mechanism, the descriptor is transformed into a histogram of some attributes. By representing the value of each bin as a binary number, the histogram itself that is a visual descriptor for the object in multimedia database could be represented as a bit string. Bit strings for all objects in multimedia database are collected to form an index file, bitmap index, in the proposed indexing mechanism. By XORing them with the descriptors for query object, the candidate solutions for similarity search could be computed easily and they are checked again with query object to precisely compute the similarity with exact metric such as L1-norm. These indexing and searching mechanisms are efficient because the filtering process is performed by simple bit-operation and it reduces the search space dramatically. Upon experimental results with more than 100,000 real images, the proposed indexing and searching mechanisms are about 15 times faster than the sequential searching with more than 90% accuracy.

Key words : Multimedia Indexing, Contents-based Multimedia Retrieval, Bitmap Indexing

· 본 연구는 서강대학교 산업기술연구소 연구비 지원으로 이루어졌음
† 비 회 원 : 삼성종합기술원 전문연구원
jjguk@mlneptune.sogang.ac.kr
†† 종신회원 : 서강대학교 컴퓨터학과 교수
jhnang@ccs.sogang.ac.kr
논문접수 : 2003년 8월 29일
심사완료 : 2004년 11월 2일

1. 서 론

데이타 압축 및 컴퓨터 처리 기술이 발달함에 따라 컴퓨터 상에서 멀티미디어 데이타 사용이 일반화되었고 이로 인해 내용 기반 이미지 검색 시스템과 같은 응용 분야의 연구가 활발히 이루어졌다. 내용 기반 이미지 검색 시스템에서 수행되는 검색 형태 중에서는 질의 이미지와 비슷한 k개의 이미지를 결과로 보여주는 k-NN (Nearest Neighbor) 검색이 가장 일반적인 검색 형태이다. 일반 텍스트 문서 검색과는 달리 이미지를 검색하는 k-NN 검색에서는 아주 많은 특성(예를 들면 색 정보, 텍스처 정보, 외곽선 정보 등)들을 이용하게 되고, 각 특성들이 고차원으로 표현되기 때문에 이미지의 특성을 읽고 비교하는 데 많은 시간이 요구된다. 이를 위해 다양한 멀티미디어 데이타 인덱싱 방법이 연구되었다. 또한 시스템마다 사용되는 특성 및 이를 기술하기 위한 방법이 다양하기 때문에 정보를 검색하거나 교환, 공유를 편리하게 하기 위한 메타 데이타 표준의 필요성이 생겨나게 되었다. MPEG-7[1-3]은 이러한 요구를 반영하기 위해 나온 메타 데이타 기술 표준이다.

멀티미디어 데이타 인덱싱 방법에 대한 기존의 연구는 트리 형태를 이용하여 데이타 공간을 나누는 다차원 인덱싱 방법[4-10], 차원을 줄여 검색 속도를 빠르게 하는 방법[11], 양자화 등의 압축 기법을 이용하는 방법[12] 등으로 나눌 수 있다. 하지만 기존의 방법들은 고차원 데이타 공간(100개 이상)을 고려하지 않았기 때문에 고차원에서는 검색 속도가 느려져서 순차 검색보다도 검색 시간이 오래 걸린다는 단점이 있다[13]. 그 이유는 차원의 개수가 많아짐에 따라 데이타 공간상의 오브젝트가 너무 산재하게 되어 데이타 공간을 나눈다는 것 자체가 큰 의미가 없게 되기 때문이다. MPEG-7 시각 정보 기술자는 고차원의 데이타로 표현이 되기 때문에 기존의 인덱싱 방법을 이용하여 검색을 하게 되면 검색 속도가 느려지게 된다. 그리고 MPEG-7 시각 정보 기술자는 유사도 측정을 하는 데 있어 각 차원의 데이타가 서로 다른 중요도를 가지고 비교된다는 특징이 있다. 대부분의 인덱싱 방법은 각 차원의 중요도를 고려하지 않기 때문에 MPEG-7 시각 정보 기술자의 데이타를 그대로 이용하게 되면 검색의 정확도가 떨어지게 된다. 그러므로 각 차원의 중요도가 같도록 데이타를 변형하여야 한다.

본 논문에서는 MPEG-7 시각 정보 기술자들과 같은 고차원 데이타들에 대하여 빠른 검색 속도를 유지하면서 정확도를 높일 수 있는 새로운 멀티미디어 인덱싱 방법을 제안한다. 본 논문의 방법에서는 기술자의 각 속성들을 동일한 중요도를 갖는 빈으로 변형한 후, 속성 히스토그램을 만들고, 히스토그램의 각 빈 값을 이진 형태로 표현하여 비트열을 생성한다. 그리고 이러한 비트열들을 이용하여 비트맵 인덱스를 구성한다. 각 속성들을 동일한 중요도를 갖는 빈으로 변형하기 위해서 기술자 비교 연산식의 특징을 이용하였다. Dominant Color의 비교 연산식은 가우스 혼합 모델을 따르고 있기 때문에, 속성들에 가우스 혼합 모델을 적용시킨 후 새로운 빈들을 생성하도록 하였다. 그리고 Homogeneous Texture의 비교 연산식은 하나의 속성이 여러 개의 속성과 비교되고, Edge Component Histogram의 비교 연산식은 가중치가 적용된 L1-norm을 사용하기 때문에 빈의 개수를 속성의 개수보다 많은 수로 확장시켜 각 빈이 독립적으로 비교될 수 있게 하였다. 인덱스는 검색 결과에 포함될 가능성이 있는 오브젝트들을 추출하기 위해서 이용이 된다. 질의 오브젝트가 입력되면 각 오브젝트의 인덱스와 질의 오브젝트의 비트열에 대한 XOR 연산을 수행하여서 후보 오브젝트 리스트를 생성한다. 그리고 이 리스트에 있는 오브젝트들에 대해서만 L1-norm과 같은, 기술자를 위해 사용되는 연산을 수행하여 최종 결과 오브젝트들을 사용자에게 보여주게 된다. 제안하는 알고리즘은 단순한 비트 연산을 통해 검색 결과에 포함될 가능성이 있는 오브젝트들을 추출해낼 수 있기 때문에 빠른 시간 내에 검색을 마칠 수 있도록 해준다. 본 논문에서 제안한 인덱싱 방법의 효율성을 증명하기 위해서 실제 100,000개 이상의 이미지 오브젝트들에 대한 비트맵 파일을 생성하였고, 이를 이용한 검색 시간과 MPEG-7 시각 정보 기술자내의 정보를 압축하지 않고 저장한 숫자 데이타를 이용한 검색 시간, VAFile[16]를 이용한 검색 시간을 비교하였다. 또한 검색 알고리즘의 정확도를 증명하기 위해서 Recall, Precision을 측정하였다. 실험을 통해 본 논문의 알고리즘은 90%이상의 정확도를 가지고 있고, 15배 이상의 속도 향상을 얻을 수 있음을 알 수 있었다. 본 논문의 알고리즘은 VOD (Video On Demand)나 DVL(Digital Video Library)과 같은 다양한 응용 분야에 유용하게 이용할 수 있을 것이다.

2. 연구 배경

인덱싱이란 검색 속도를 향상시키기 위해서 데이타 공간상의 특징을 미리 계산하여 검색에 사용하기 위한 데이타를 만들어내는 방법이다. 본 장에서는 기존의 멀티미디어 인덱싱 방법에 대하여 설명하고, 본 논문의 중요 대상이 되는 MPEG-7 기술자의 특징에 대하여 설명하도록 한다.

2.1 기존의 멀티미디어 인덱싱 방법

내용 기반 이미지 검색 시스템에서 수행되는 검색 형

태 중에서는 질의 이미지와 비슷한 k개의 이미지를 결과로 보여주는 k-NN검색이 가장 일반적인 검색 형태이다. 데이타베이스를 DB, 검색 결과 집합을 R, 질의 오브젝트를 q라고 정의하고, 검색 결과 집합내의 오브젝트들을 a, 데이타베이스내의 오브젝트들 중에서 검색 결과 집합 내의 오브젝트들을 제외한 나머지 오브젝트들을 b라고 정의하는 경우, k-NN 검색은 결과 집합으로 다음과 같은 조건을 만족하는 오브젝트들을 포함하게 된다[13].

$$\forall a \in R \ , \forall b \in DB - R \ , |q-a| \le |q-b|$$

(비교 연산식은 L1-norm이라고 가정한다.)

위에서 설명된 k-NN검색을 위해 가장 많이 연구된 알고리즘 형태는 트리 형태로 대표되는 다차원 인덱싱 방법(multidimensional indexing methods)으로 T-tree[4], KDB-tree[5], Ann-tree[6], MB+-tree[7], R*-tree[8], X-tree[9], SR-tree[10] 등이 이와 같은 형태의 대표적인 알고리즘들이다. 이와 같은 방법에서는 기본적으로 데이타 공간을 나눈 후, 각 오브젝트들을 해당되는 공간으로 클러스터링을 한다. 질의가 오면 이 클러스터링 정보에 의해서 검색 공간을 줄이게 되는 것이다. 하지만 다차원 인덱싱 방법은 차원의 개수가 많아지면서 오브젝트들이 데이타 공간상에 너무 산재되기 때문에 차원의 개수가 적은 경우에는 빠른 시간 내에 검색이 가능하지만, 차원의 개수가 많은 경우에는 오히려 순차 검색보다 느려지게 된다. 기존의 연구[13]에 따르면 오브젝트의 공간을 256차원으로 가정하는 경우, 각 차원의 크기가 0.95(1로 정규화된 차원)인 초입방체 안에 오브젝트가 들어있는 확률이 0.002%이다. 이러한 실험 결과를 통해 고차원의 경우 데이타 공간을 나눈다고 하더라도 나누어진 공간상에 오브젝트가 하나도 없거나 혹은 하나만 존재할 확률이 높다는 것을 알 수 있다. 그러므로 오히려 인덱싱 정보를 읽는 시간 및 계산하는 시간만 늘어나게 되어 순차 검색보다 많은 검색 시간이 요구되는 것이다. 다차원 인덱싱 방법 외에도 차원을 줄이는 방법[11], 데이타를 압축하는 방법[12] 등 여러 가지 방법이 연구되었으나 데이타 손실 등의 문제로 인해 고차원 공간상에서 일반 순차 검색에 비해 정확도 및 속도에서 많은 향상을 보이지는 않았다.

IGrid 인덱스 알고리즘[14]은 위에서 설명한 고차원 데이타 공간상에서 발생하는 문제점을 해결하고자 제안된 알고리즘이다. 먼저 이 알고리즘에서는 각 차원을 똑같은 개수의 오브젝트들이 포함되도록 영역을 나눈 후 나누어진 영역 정보를 이용하여 검색을 하게 된다. 비교되는 두 개 오브젝트의 차원들 중 같은 영역 안에 포함되는 차원들을 쉽게 추출할 수 있게 되는데 이러한 차원들의 집합을 proximity 집합이라고 한다. 실제 유사도 측정은 proximity 집합 안의 차원에 대해서만 이루어진다. 이 알고리즘은 각 차원에 대해서 독립적인 계산이 적용되기 때문에 오브젝트가 산재되어 생기는 문제점은 발생되지 않는다. 하지만 똑같은 개수의 오브젝트를 갖도록 영역을 나누게 됨으로써 비슷한 오브젝트들이 서로 다른 영역으로 나뉘게 되는 경우가 생기게 된다. 예를 들어 그림 1과 같이 6개의 오브젝트를 2개의 영역으로 나눈다고 가정하는 경우, c는 d에 가깝지만 a, b와 한 영역으로 묶이게 된다. 이와 같은 오차들이 고차원에 대해서 누적이 된다면 전체 성능에 큰 영향을 주게 되어 정확도가 떨어지게 되는 것이다.

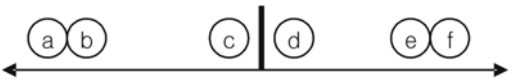


그림 1 IGrid 인덱스 알고리즘의 문제점

VAFile을 이용한 인덱싱 알고리즘[16]은 위에서 설명한 고차원 데이타 공간상에서 발생하는 문제점을 해결하고자 제안된 알고리즘이다. 이 알고리즘의 기본 아이디어는 오브젝트들로 구성된 집합의 대표값을 이용하여 결과 집합에 포함될 가능성이 있는 오브젝트들을 추출하여 후보 리스트를 생성하고, 이 후보 리스트 내의 오브젝트들에 대해서만 실제 비교 연산식을 수행하도록 하는 것이다. 먼저 데이타 공간을 미리 정의된 개수로 나눈 후 각 오브젝트에 대해서 포함되는 공간의 인덱스만 저장을 하게 된다. 예를 들어, 2차원 데이타 공간에서, 1차원, 2차원을 4개로 나누고 각 차원의 인덱스를 00,01,10,11이라고 정의하는 경우, 데이타가 각 1,2차원 2번째 공간에 위치한다면 데이타의 인덱스는 0101이 되는 것이다. 질의가 입력되면, 공간 정보를 이용하여 최소, 최대 거리의 근사치를 구할 수 있기 때문에 이를 이용하여 결과 집합에 포함될 가능성이 있는 오브젝트들을 추출할 수 있게 되는 것이다. 이 알고리즘은 저장되는 인덱스의 크기가 크지 않기 때문에 인덱스를 읽는 시간은 줄일 수 있지만, 후보 리스트를 추출하기 위해 유클리드 거리를 계산하는 계산량이 많아져서 검색하는데 시간이 많이 걸리게 된다.

2.2 MPEG-7 시각 정보 기술자 특징

MPEG-7[1-3]은 멀티미디어 메타 데이타 기술 표준이다. 현재 동영상 검색, 이미지 검색 등의 다양한 응용분야에 적용이 되고 있고, 6개의 분야에 걸쳐 표준화 연구가 진행되고 있다. 이 중에서 특히, MPEG-7 시각 정보 부분은 예제 기반 동영상/이미지 검색 시스템 등에서 많이 이용이 되고 있다. MPEG-7 시각 정보 부분에

는 색 정보, 텍스처 정보, 움직임 정보, 형태 정보 등을 기술하는 방법이 정의가 되었는데, 이 중에서 Dominant Color, Color Structure, Homogeneous Texture, Edge Histogram 등은 이미지 검색을 하는 데 기본적으로 이용이 될 수 있는 기술자들이다. 물론 이 외에도 움직임 정보, 형태 정보에 관한 기술자들이 있지만, 이러한 기술자들은 오브젝트 추출 등의 다른 부가적인 알고리즘을 적용시켜야 이용할 수 있다. 그렇기 때문에 위에서 제시된 4개의 기술자는 다른 기술자에 비해 쉽게 추출될 수 있어 널리 사용되고 있고, 이 절에서는 이 4개 기술자의 특징을 분석하도록 한다.

• Dominant Color

이 기술자는 임의 영역에서의 주된 색 정보를 표현하고 있다. 미리 정의된 개수만큼 색을 추출하고, 각 색을 나타내는 양자화 정보, 칼라 공간 정보, 인덱스, 비율 등을 이용하여 기술한다. Dominant Color의 비교 연산식에서는 LUV 칼라 공간을 사용하고 있기 때문에 다른 칼라 공간에 있는 값을 LUV 칼라 공간으로 변형하여야 한다. 이 기술자를 이용하여 이미지 혹은 비디오를 검색하는 경우 사용되는 비교 연산식은 일반적으로 식 (1)과 같으며, 식 (1)은 각 색이 가우스 분포 형태로 분포된다고 가정하고 구성된 가우스 혼합 모델을 기반으로 만들어졌다. 이 식에서 F_1, F_2는 두 개의 기술자를 나타내고 있고, m_1, m_2는 F_1, F_2에서 사용되는 색의 개수, p_{1i}, p_{2i}는 F_1, F_2의 i번째 Dominant Color의 비율을 나타낸다. $c^{(l)}_{xi}$, $c^{(u)}_{xi}$, $c^{(v)}_{xi}$는 각 x번째 기술자, i번째 Dominant Color의 L값, U값, V값을 나타내고, $v^{(l)}_{xi}$, $v^{(u)}_{xi}$, $v^{(v)}_{xi}$는 x번째 기술자, i번째 Dominant Color의 L값, U값, V값의 분산을 나타낸다. 식 (1)을 보면 알 수 있듯이, Dominant Color에서 사용되는 비교 연산식은 단순한 1차 함수가 아니고, 각 변수의 중요도가 서로 다르기 때문에 일반적인 인덱싱 방법을 적용시킬 수 없다. 예를 들어 IGrid에서 색 정보, 비율 정보를 각 하나의 축으로 사용하고, 색 정보 축이 Proximity Set에 들어가지 않는 경우 이 축은 전혀 무시되고, 비율에 관한 축만을 이용하여 검색을 하게 될 것이다. 그렇게 된다면 단순히 비율이 비슷한 오브젝트들이 검색 결과로 사용자에게 보내지기 때문에 원하는 결과를 얻을 수 없게 된다.

$$D(F_1,F_2),=\sum_{i=1}^{m_1}\sum_{j=1}^{m_2}p_{1i}p_{1j}f_{1i,1j}$$

$$+\sum_{i=1}^{m_1}\sum_{j=1}^{m_2}p_{2i}p_{2j}f_{2i,2j}-\sum_{i=1}^{m_1}\sum_{j=1}^{m_2}2p_{1i}p_{2j}f_{1i,2j} \quad (1)$$

where

$$f_{xi,yj}=\frac{1}{2\pi\sqrt{v^{(l)}_{xi,yj}v^{(u)}_{xi,yj}v^{(v)}_{xi,yj}}}\times\exp[-(\frac{c^{(l)}_{xi,yj}}{v^{(l)}_{xi,yj}}+\frac{c^{(u)}_{xi,yj}}{v^{(u)}_{xi,yj}}+\frac{c^{(v)}_{xi,yj}}{v^{(v)}_{xi,yj}})/2]$$

$$c^{(l)}_{xi,yj}=(c^{(l)}_{xi}-c^{(l)}_{yj})^2, c^{(u)}_{xi,yj}=(c^{(u)}_{xi}-c^{(u)}_{yj})^2, c^{(v)}_{xi,yj}=(c^{(v)}_{xi}-c^{(v)}_{yj})^2$$

$$v^{(l)}_{xi,yj}=(v^{(l)}_{xi}+v^{(l)}_{yj}), v^{(u)}_{xi,yj}=(v^{(u)}_{xi}+v^{(u)}_{yj}), v^{(v)}_{xi,yj}=(v^{(v)}_{xi}+v^{(v)}_{yj})$$

• Color Structure

Dominant Color가 단순히 주된 색 정보만을 표시한 데 반해, Color Structure는 이미지 내에서 색 정보의 분포를 표현하는 데 사용된다. 이 기술자를 표현하기 위해 이용되는 것은 칼라 히스토그램으로 윈도우를 이동하면서 특정한 색이 있는 경우 그 색에 해당되는 빈의 값을 누적시켜 칼라 히스토그램을 구하게 된다. 보통 칼라 히스토그램으로 이용되는 빈의 개수는 256을 기본으로 하고 있으나 양자화하여 128, 64 등의 빈 개수도 이용할 수 있다. 이 기술자를 이용하는 경우에는 식 (2)와 같은 L1-norm을 비교 연산식으로 사용하게 된다. 식에서 F_1, F_2는 두 개의 기술자를 나타내고 있고, m은 F_1, F_2 에서 사용되는 빈의 개수, $h_1(i)$, $h_2(i)$는 F_1, F_2 의 i번째 빈 값을 나타내고 있다. 식을 보면 알 수 있듯이 각 빈이 같은 중요도로 비교되기 때문에 일반적인 인덱싱 방법을 이용할 수 있으나, 이 경우 차원의 개수가 너무 많기 때문에 빠른 검색을 기대할 수 없다.

$$D(F_1,F_2)=\sum_{i=1}^{m}|h_1(i)-h_2(i)| \quad (2)$$

• HomogeneousTexture

이 기술자는 평균 에너지와 에너지의 편향을 이용하여 임의 영역의 텍스처 정보를 표현하기 위한 기술자이다. 이 기술자에서 사용되는 비교 연산식은 식 (3)과 같은 가중치를 적용한 L1-norm이다. F_1, F_2는 두 개의 기술자를 나타내고 있으며 $TD_1(k)$, $TD_2(k)$는 F_1, F_2의 에너지와 편향을 나타내는 값이고, m은 값의 개수, $\alpha(k)$는 k번째 가중치를 나타내는 값이다. 즉, 각 가중치를 1로 둔다면 일반 L1-norm과 같게 되고 일반적인 인덱싱 방법을 적용할 수 있다. 하지만 만약 회전 혹은 축소/확대에 이 기술자를 사용한다면 하나의 값이 여러 개의 값과 비교가 되어야 한다. 기본적으로 이 기술자에서는 평균 에너지를 위해서 30개의 값을 기술하고, 에너지의 편향을 표현하기 위해서 또한 30개의 값을 기술하기 때문에 고차원이라고는 할 수 없으나 회전 혹은 축소/확대된 이미지를 찾는데도 사용하기 위해서는 차원을 늘린 후 인덱싱을 적용해야 하기 때문에 고차원이 되고, Color Structure와 마찬가지로 인덱싱 방법을 적용하더라도 빠른 검색을 기대할 수 없게 된다.

$$D(F_1,F_2)=\sum_{k=1}^{m}\left|\frac{TD_1(k)-TD_2(k)}{\alpha(k)}\right| \quad (3)$$

• Edge Component Histogram

Edge Component Histogram 기술자는 에지의 분포

를 기술하기 위한 기술자이다. 4*4로 이미지를 나눈 후 각 영역의 에지 분포를 기술하게 되는데, 다섯 가지 에지의 분포를 기술하게 되어서 총 80개의 빈이 된다. 일반적으로 사용되는 비교 연산식은 식 (4)와 같다. F_1, F_2는 두 개의 기술자를 나타내고 있으며 $h_1(i)$, $h_2(i)$는 F_1, F_2 기술자내의 정보 중에서 각 영역별 에지 히스토그램의 i번째 빈 값을 나타내고, $h^g_1(i)$, $h^g_2(i)$는 전체 이미지 히스토그램의 i번째 빈 값을 나타낸다. 그리고 $h^s_1(i)$, $h^s_2(i)$는 서로 인접한 영역을 합친 후 만들어진 새로운 영역에 대한 에지 히스토그램의 i번째 빈 값을 나타낸다. 식을 보면 알 수 있듯이 전체 이미지의 에지 히스토그램을 제외하면 똑 같은 가중치가 곱해진다. 하지만 전체 이미지의 히스토그램 값은 5가 곱해지기 때문에 다른 빈의 값과 동일한 차원으로 인덱싱을 하게 되면 성능이 나빠지게 된다. 이 기술자 또한 150개의 차원이 필요하게 되므로 고차원이라고 할 수 있다.

$$D(F_1,F_2)=\sum_{i=0}^{79}|h_1(i)-h_2(i)|+5*\sum_{i=0}^{4}|h_1^g(i)-h_2^g(i)|+\sum_{i=0}^{64}|h_1^s(i)-h_2^s(i)| \quad (4)$$

위에서 살펴본 바와 같이 MPEG-7 기술자는 고차원 데이타이고, 각 차원의 중요도가 다른 경우도 있다. 일반적으로 다차원 데이타를 위한 인덱싱 알고리즘 대부분은 100차원이 넘는 고차원에서는 검색 속도의 향상을 기대할 수가 없고, 고차원을 위한 기존의 인덱싱 알고리즘들은 recall, precision과 같은 성능이 좋지 않다는 문제점이 있다. 물론, 피드백을 통해 성능 향상을 기대할 수 있으나 기본적인 성능이 좋지 않다면, 사용자가 선택할 수 있는 오브젝트가 적어질 수 밖에 없기 때문에 전체적인 성능 향상을 기대할 수 없게 된다. 그러므로 MPEG-7 시각 정보 기술자로 구성된 데이타베이스를 사용하는 경우, 기존의 인덱싱 방법으로는 효율적인 검색을 할 수 없다. 또한, 일반적으로 인덱싱 알고리즘에서는 각 차원의 중요도를 다르게 이용하지 않기 때문에 MPEG-7 기술자를 변형한 후 인덱싱 알고리즘을 적용시켜야 한다.

3. 비트맵을 이용한 인덱싱 알고리즘

이 장에서는 본 논문에서 제안한 비트맵을 이용한 인덱싱 알고리즘에 대해서 설명하도록 한다. 기존의 인덱싱 알고리즘 대부분은 고차원에서 "Curse of dimensionality" 때문에 빠른 검색 속도를 보이지 못하고, MPEG-7 기술자들 대부분은 고차원 형태이기 때문에 고차원에서 빠른 검색 속도를 유지할 수 있는 새로운 인덱싱 알고리즘이 필요하다. 본 논문에서 제안한 알고리즘은 비트맵을 이용함으로 해서 인덱싱 데이타의 크기를 줄이고, 전체 오브젝트를 최소한 한번씩은 비교함으로 해서 정확도를 높이도록 하였다. 3.1절에서는 본 논문에서 사용된 비트맵 인덱스 생성 알고리즘에 대해서 설명하도록 하고, 3.2절에서는 제안된 비트맵 인덱스를 이용한 검색 알고리즘에 대해 설명하도록 한다. 마지막으로 3.3절에서는 2장에서 설명했던 MPEG-7 기술자들을 본 논문의 인덱싱 알고리즘에 맞도록 변형하기 위한 방법에 대해서 설명하도록 한다.

3.1 비트맵 인덱스 생성 알고리즘

데이타베이스에서 사용되는 비트맵 인덱스[15]는 각 오브젝트들이 가지고 있는 특성(차원)들을 이진형태로 나타낸 인덱스로, 어떤 오브젝트에 대한 인덱스의 각 비트는 이 오브젝트가 특정 조건을 만족시키는 지를 나타낸 것이다. 비트맵 인덱스의 특징은 크기가 작고, 하나의 비트에서 참,거짓 두 가지만을 표현할 수 있다는 것이다. 그림 2는 비트맵 인덱스의 예를 보여주는 그림이다. 그림에서 비트맵의 조건은 0.5보다 크다는 것이고, 테이블의 마지막 열에 조건에 맞는 비트맵들이 나열이 되어 있다. 각 비트맵의 비트들은 "0.5보다 크다" 혹은 "작다"의 두 가지 의미만 표현하고 있다. 결국 표현하고자 하는 데이타와 데이타의 조건만 정의할 수 있다면 원래 표현하는 데이타보다 훨씬 작은 수의 비트만으로 표현이 가능하게 된다.

오브젝트 \ 차원	1	2	3	4	비트맵
O_1	0.3	0.8	0.1	0.95	0101
O_2	0.2	0.4	0.9	0.1	0010
O_3	0.2	0.5	0.75	0.77	0111
O_4	0.7	0.6	0.0	0.8	1101

비트맵 조건: (>0.5)

그림 2 비트맵 인덱스의 예

이미지 오브젝트 인덱스에서 표현하고자 하는 데이타는 이미지의 특징을 나타내는 값들이다. 본 논문에서는 이러한 데이타를 Color Structure와 같은 히스토그램 형태라고 가정한다. 즉, 각 빈의 값은 동등하게 하나의 차원을 이룰 수 있다. 본 논문의 비트맵 인덱스에서 사용되는 조건은 각 빈이 차지하는 중요도이다. 히스토그램에서 중요하다는 의미는 빈의 값이 크다는 것과 같다. 즉, 빈의 값이 크다는 것은 그 빈에 의해 특징지어지는 특성이 이미지에 많다는 뜻이기 때문에 이미지에서 차지하는 중요도가 높아지게 된다.

각 빈이 차지하는 중요도를 이용하여 인덱스를 생성하는 과정은 다음과 같다. n개의 이미지 오브젝트 기술자들로 이루어진 데이타베이스를 Λ라고 하고, 데이타베이스내의 k번째 이미지 오브젝트의 기술자를 $\overrightarrow{O_k}$라고

정의하면 $\Lambda = \{ \overrightarrow{O_k} | \ 1 \le k \le n \}$과 같이 정의할 수 있다. 각 이미지 오브젝트의 기술자들은 히스토그램 형태라고 가정하기 때문에 전체 빈의 개수를 m개라고 하고, k번째 오브젝트 기술자의 i번째 히스토그램 빈값을 h_k^i라고 정의하면 $\overrightarrow{O_k} = (h_k^1, h_k^2, \cdots, h_k^m)$와 같이 나타낼 수 있다. 인덱스는 이와 같은 이미지 오브젝트의 기술자 $\overrightarrow{O_k}$마다 하나씩 생성된다. $\overrightarrow{O_k}$에 대한 인덱스를 생성하는 방법은 먼저 각 빈의 값을 0~1 사이의 값으로 정규화하여, 다음과 같이 정규화된 빈으로 구성된 벡터인 $\overrightarrow{N_k}$를 생성한다. 여기서 $h_{\max}^i$는 i번째 빈의 최대값을 나타낸다.

$$\overrightarrow{N_k} = (n_k^1, n_k^2, \cdots, n_k^m), \quad n_k^i = h_k^1 / h_{\max}^i$$

빈 값을 이진으로 표현하기 위해서 임계값을 사용하게 되는데, 임계값은 각 오브젝트의 기술자마다 상대적인 값으로 설정하기 위해 오브젝트 기술자의 빈 값을 이용한다. 즉, 정규화된 빈 값을 높은 순서로 정렬하여 T번째 값을 $\overrightarrow{O_k}$의 임계값(ε_k)으로 설정한다. 결국 k번째 오브젝트 기술자 인덱스의 i번째 비트는 n_k^i가 ε_k보다 크다면 1로 설정되고, 크지 않다면 0으로 설정되어 인덱스는 B_k와 같은 비트열이 된다.

$$B_k = b_k^1, b_k^2, \cdots, b_k^m,$$

$$\begin{cases} if \ n_k^i \ge \varepsilon_k, then \ b_k^i = 1 \\ else, b_k^i = 0, \ for \ \forall i, 1 \le i \le m \end{cases}$$

n개의 오브젝트 기술자들로 구성된 데이타베이스를 표현하기 위해서 필요한 비트맵 인덱스의 전체 크기는 각 오브젝트의 기술자가 m개의 빈, 즉 m차원으로 구성된 히스토그램이라고 가정하면 $\left\lceil \frac{n \times m}{8} \right\rceil$바이트가 된다.

그림 3은 본 논문에서 제안한 방법을 이용하여 만들어낸 비트맵 인덱스의 예를 보여주는 그림이다. 각 오브젝트 기술자의 차원은 8차원이라고 가정하고, T는 3으로 설정하였다. 첫 번째 오브젝트(O_1)에서는 5번째 차원의 값인 0.7이 임계값이 되며, 2번째 차원의 값, 4번째 차원의 값, 5번째 차원의 값이 중요도가 높은 값으로 설정이 된다. 이와 같은 방법으로 인덱스를 표현하면, 8차원의 오브젝트 하나를 표현하기 위해 단지 1 바이트만이 필요하게 되므로 오브젝트의 정보를 효율적으로 표현할 수 있게 된다.

본 논문에서 제안한 비트맵 인덱스를 사용하는 경우 각 인덱스가 오브젝트마다 하나씩 존재하기 때문에 동적으로 인덱스를 추가 삭제하는 것이 간단하게 수행될 수 있다. 그림 4(a)는 삽입 알고리즘에 관한 의사 코드를 보여주는 그림이고, 그림 4(b)는 삭제 알고리즘에 관

오브젝트 \ 차원	1	2	3	4	5	6	7	8	ε_k	비트맵
O_1	0.1	0.9	0	0.8	0.7	0	0.2	0.3	0.7	01011000
O_2	0.4	0.6	0.9	0	0	0.5	0.3	0.1	0.5	01100100
O_3	0.2	0	0.5	0	0	0.7	0	0.9	0.5	00100101
O_4	0	0.1	0	0.9	0.5	0.8	0.7	0	0.7	00010110

그림 3 제안한 알고리즘에 의한 비트맵 인덱스

```
// O[i] = 비트맵 인덱스 리스트
// { id: 아이디, index: 비트맵 인덱스}
// Bindex=현재 비트맵 인덱스의 개수
// newBI = 삽입하고자 하는 인덱스

Procedure Insert(Object newBI)
{
  O[Bindex].id = newBI.id;
  O[Bindex].index = newBI.index;
  Bindex = Bindex + 1;
}
```

(a) 삽입 알고리즘

```
// O[i] = 비트맵 인덱스 리스트
// { id: 아이디, index: 비트맵 인덱스}
// Bindex=현재 비트맵 인덱스의 개수
// offset = 삭제하고자 하는 인덱스 위치

Procedure Delete(int searchID)
{
  for ( int i = 0; i < Bindex ; i++ ) {
    if ( searchID == O[i].id ) {
      offset = &O[i];
      break;
    }
  }

  Delete(offset);

  for ( int j = 1 ; j < Bindex ; j++ ) {
    O[j] = O[j+1];
  }
  Bindex = Bindex - 1;
}
```

(b) 삭제 알고리즘

그림 4 인덱스 삽입 및 삭제 알고리즘

한 의사 코드를 보여주는 그림이다. 삽입하는 경우에는 단순히 마지막에 삽입하고자 하는 인덱스를 추가하면 되고, 삭제하는 경우에는 삭제하고자 하는 인덱스의 위치(offset)를 구한 후, 그 위치에 있는 인덱스를 삭제하면 된다. 삭제 후에는 그 인덱스에 해당하는 메모리 혹은 디스크를 다시 사용하기 위해 다른 인덱스의 위치를 재조정해주게 된다. 기존의 여러 알고리즘에서는 오브젝트의 상관성을 살펴보기 때문에 새로운 인덱스가 들어오거나 삭제될 때 그 상관성이 어떻게 변하는 지 살펴보고 인덱스를 재구성해야 하지만, 본 논문에서 제안한 인덱스의 경우에는 각 오브젝트가 독립적으로 처리될 수 있기 때문에 단순히 하나의 오브젝트에 대한 비트열만 추가하거나 삭제해주면 되는 것이다.

3.2 비트맵 인덱스를 이용한 검색 알고리즘

비트맵 인덱스는 오브젝트의 특징을 표현한 히스토그램내의 중요한 빈들을 1로 설정하여 이진화함으로써 근사화한 데이타이다. 본 논문의 검색 알고리즘(그림 5)에서는 이러한 비트맵 인덱스의 특징을 이용하여 검색 결과 리스트에 포함될 가능성이 높은 후보 오브젝트의 리스트를 생성한 후, 실제 L1-norm과 같은 비교 연산식은 후보 리스트에 대해서만 수행하도록 하였다. 후보 오브젝트의 리스트를 생성하기 위해서 본 논문의 검색 알고리즘에서는 일반적인 비교 연산식의 특징을 이용하게 된다. 일반적으로 사용되는 비교 연산식인 유클리드 연산법, L1-norm 등을 고려할 때 서로 차이가 많이 나는 빈들이 결과에 많은 영향을 끼치게 되는 것을 알 수 있다. 서로 높은 값을 가지거나 낮은 값을 가지게 되면 뺄셈에 의해 값이 없어지게 되므로 결과에 영향을 크게 끼치지는 않게 된다. 이와 같은 일반 연산식의 특징을 이용하기 위해 Exclusive OR 연산을 이용하였다. Exclusive OR 연산은 서로 다른 비트, 즉 0과 1 혹은 1과 0인 경우에만 1이 되는 특징이 있기 때문에 차이가 많이 나는 빈들을 추출(그림 5의 Filtering()함수)할 수 있게 된다. 결국 Exclusive OR 연산을 통해 나온 비트열의 1의 개수를 계산(그림 5의 NumberOfSetting()함수)하여 후보 리스트를 생성하게 된다. 그림 6은 질의 오브젝트가 입력될 때, L1-norm에 의한 결과값(그림 6(a))과 Exclusive OR연산에 의한 결과 비트열에 있는 1의 개수(그림 6(b))를 비교한 그림이다. 사용되는 오브젝트는 COIL-100(Columbia Object Image Library)과 MPEG-7 테스트 이미지들 중에서 12,861개를 추출하여 이용하였고, 최대 1의 개수는 40개로 제한하였다. 그림에서 A, B를 보면 알 수 있듯이, L1-norm에 의한 결과값이 작은 부분과 1의 개수가 적은 부분이 어느 정도 일치함을 볼 수 있고, 큰 부분도 1의 개수가 많은 부분과 일치함을 볼 수 있다. 즉, Exclusive OR 연산을 이용하여 비슷한 특성을 갖는 오브젝트를 추출할 수 있기 때문에 검색 영역을 줄일 수 있다는 것을 확인할 수 있다. 이와 같이 후보 리스트를 구한 후에는 마지막으로 줄어든 검색 영역에 대해 실제 비교 연산식(MPEG-7 시각 정보 기술자 비교 연산식)을 수행(그림 5의 FinalSearch()함수)하여 비슷한 k개의 오브젝트를 추출한 후 클라이언트에게 보내주게 된다.

근사화 방법을 이용하여 인덱스를 생성하는 경우에는 데이타 손실이 있기 때문에 100% 정확하게 비슷한 오브젝트를 찾는 것은 불가능하다. 그러므로 결과에 대한 사용자의 피드백을 처리하는 과정이 필요하게 되는데 본 논문의 알고리즘을 이용하는 경우에는 간단한 연산을 통해 피드백을 수행할 수 있게 된다. 예를 들어 k개의 결과 중 t개의 결과에 대해서만 사용자가 만족하는 경우($k \geq t$), 이 t개의 결과를 다시 질의로 이용하게 될 것이다. N개의 결과를 $R_1, \cdots, R_t$이라고 하는 경우, 새로운 질의 Q_{new}는 R_1 AND R_2 ⋯ AND R_t이 되고, 이를 이용하여 검색을 하게 된다. 여러 개의 인덱스에 대한 AND 연산을 수행하게 되면 공통적으로 중요한 빈들만 결과 비트열에 1의 값으로 남게 되어 t개에 대한 검색의 효과를 얻을 수 있게 된다.

3.3 MPEG-7 기술자의 변환 방법

2장에서 설명한 바와 같이 MPEG-7 기술자들 중에서는 히스토그램 형태가 아닌 기술자들이 있다. 이와 같은 기술자들은 본 논문에서 제안한 알고리즘에 히스토그램 형태로 변형하여 적용하여야 한다. Color Structure는 일반 히스토그램 형태이고, L1-norm 형태의 연산식을 사용하기 때문에 그대로 적용해도 된다. 이 절에서는 2장에서 설명된 기술자들 중에서 Color Structure 기술자를 제외한 나머지 기술자들을 히스토그램 형태로 바꾸는 방법에 대해 설명하도록 한다.

• Dominant Color의 변환

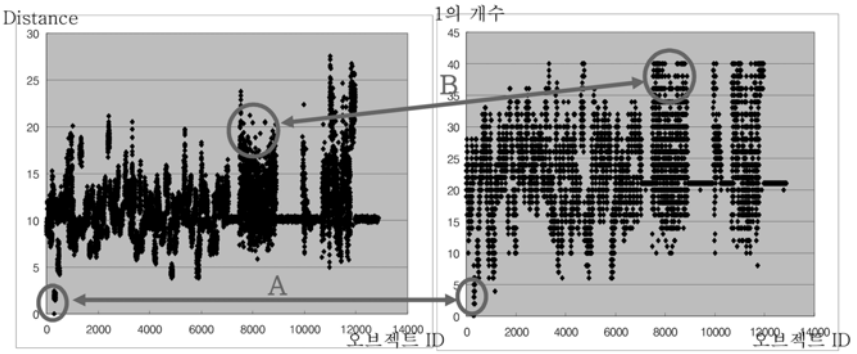
Dominant Color는 지정된 개수의 색으로 세그멘테이션 한 결과를 기술한다. 이 결과를 히스토그램 형태로 만들기 위해서는 세그멘테이션 하기 전의 이미지가 필요하게 된다. 그러나 실제 인덱싱을 위해 사용될 수 있는 데이타는 Dominant Color 기술자 하나이기 때문에 원래 이미지에서 얻을 수 있는 데이타에 근사하도록 데이타를 변형하여야 한다. Dominant Color 기술자를 기술하기 위해 이용되는 특성은 색 정보 인덱스와 비율 정보이다. 이 중에서 색 정보 인덱스는 빈의 인덱스로 변형이 가능하고, 비율 정보는 빈의 값으로 변형이 가능하다. 그러나 단순히 이와 같이 변형한다면, 몇 개의 색으로만 표현이 되어 1의 개수에 의한 구분이 불가능하게 될 것이다. 이를 해결하기 위해서는 다른 정보를 같이 표현해야 한다. Dominant Color의 비교 연산식은

```
// QueryBit : 질의 비트맵 인덱스
// numObject : 데이터베이스내에 있는 오브젝트의 개수
// resultBuffer : 질의 비트맵과 데이터베이스내 인덱스의 XOR 결과 비트맵
// Database[i] : 데이터베이스내 i번째 비트맵 인덱스
// resultNumber : resultBuffer의 1의 개수
// numResult : 후보 오브젝트 리스트의 크기
// maxnumberofone : 현재 후보 오브젝트 리스트내 비트맵들의 1의 개수 중 가장 큰 1의 개수
// maxindex : maxnumberofone을 갖는 오브젝트의 인덱스
// ResultSet :결과 후보 리스트{numberofone:1의 개수,sourceindex:오브젝트의 인덱스}
// numbit: 비트맵의 전체 비트수
// Query : 질의
// DatabaseVal[i] : 데이터베이스내 i번째 오브젝트

Procedure BitMapSearch(BitMap QueryBit) {
  Filtering(QueryBit);          // 비트맵 인덱스를 이용하여 결과 후보 리스트를 생성
  FinalSearch(QueryBit);        // 줄어든 검색 영역에 대한 유사도 측정
}
Procedure Filtering(BitMap QueryBit) {      // 결과 후보 리스트를 생성하기 위한 함수
  for ( int i = 0 ; i < numobject ; i++ ) {
    BitMap resultBuffer   = QueryBit ^ Database[i];     // XOR 계산
    int resultNumber   = NumberOfSetting(resultBuffer);  // 1의 개수 계산
    if ( i < numResult ) {       // 결과 집합 초기화
      ResultSet[i].numberofone = resultNumber; ResultSet[i].index = i;
    }
    else {
      // maxnumberofone, maxindex 초기화
      if ( i == numResult)
        UpdateMax();
      // 새로운 오브젝트가 후보 리스트에 추가되는 경우
      if ( resultNumber < maxnumberofone ) {
        ResultSet[maxindex].numberofone = resultNumber; ResultSet[maxindex].index = i;
        UpdateMax();
      }
    }
  }
}
Procedure NumberOfSetting(BitMap Buffer) {   // 비트맵에 있는 1의 개수를 구하는 함수
  int resultNum = 0;
  for ( int i = 0 ; i < numbit ; i++ ) {
    if ( (Buffer[i] & 1) != 0 ) {
      resultNum++;
    }
  }
  return resultNum;
}
Procedure UpdateMax() {     // 후보 리스트에 있는 비트맵중 1의 개수가 가장 많은 것을 찾아냄
  for ( int i = 0 ; i < numResult ; i++ ) {
    if ( ResultSet[i].numberofone > maxnumberofone ) {
      maxindex = i; maxnumberofone=  ResultSet[i].numberofone;
    }
  }
}
Procedure FinalSearch(BitMap QueryBit) {   // 결과 후보 리스트를 대상으로 비교 연산식 수행
  for ( int i = 0 ; i < numResult ; i++ ) {
    Distance(Query, DatabaseVal[ResultSet[i].index]);   // 실제 비교 연산식 수행
  }
}
```

그림 5 검색 알고리즘

가우스 혼합 모델을 기반으로 만들어진 식이다. 식 (1)을 보면 알 수 있듯이 가우스 혼합 모델을 적용하기 위해 단순히 비율의 차이만 이용되는 것이 아니라 색 정보 인덱스의 차이와 분산도 전체 유사도에 중요하게 이용되는 것을 알 수 있다. 그러므로 본 논문에서는 이와 같은 특징을 적용하기 위해 가우스 필터를 적용한 후 각 곡선을 혼합하여 히스토그램 형태의 데이타로 변형하였다. 그림 7은 Dominant Color의 개수가 5개이고, 각 빈의 값은 0~1사이의 값으로 정규화되었으며, 빈은 0~128의 값으로 정규화된 경우, 히스토그램 형태로 변형된 Dominant Color의 예를 보여주는 그림이다. 그림을 보면 알 수 있듯이, 본 논문에서 사용한 방법을 적용하면 좌측 테이블과 같이 여러 색으로 구성된 Dominant Color는 우측 그래프와 같이 각 인덱스 주변으로



(a) L1 norm에 의한 결과 값 (b) Exclusive OR 결과에 대한 1의 개수

그림 6 일반 비교 연산식과 XOR를 이용한 연산식의 비교

색 정보 인덱스	빈값
1.57	0.1
19.34	0.033
37.3	0.06
57.49	0.1
80.73	0.66

Dominant Color(5개의 색)

Ω
(가우스 혼합 모델)

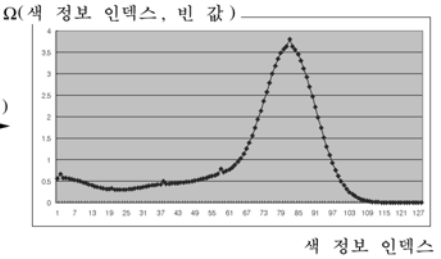


그림 7 Dominant Color 기술자의 변형

가우스 함수의 형태를 띄게 되고, 모든 가우스 함수의 값들이 더해져서 만들어진 그래프의 형태로 바뀌는 것을 알 수 있다.

• Homogeneous Texture의 변환

2장에서 설명했던 바와 같이 Homogeneous Texture 기술자는 유사도 측정을 위해 일반 l1-norm을 연산식으로 사용한다. 그러나 회전 혹은 축소/확대된 이미지도 찾아내기 위해서는 하나의 차원이 서로 다른 여러 차원과 비교되게 된다. 그러므로 본 논문의 인덱싱 알고리즘을 그대로 적용한다면 많은 검색 오류를 발생하게 된다. 이와 같은 문제점을 해결하기 위해서 전체 차원을 늘리는 방법을 이용한다. 예를 들어 두 개의 Homogeneous Texture 기술자 벡터 $\vec{A}$ = {a_1,a_2,a_3,a_4,a_5}와 $\vec{B}$ = {b_1, b_2,b_3,b_4,b_5}가 비교되는 경우, $\vec{A}$의 첫 번째 차원 값(a_1)이 $\vec{B}$의 첫 번째(b_1), 네 번째(b_4), 다섯 번째 차원 값(b_5)과 비교되고, 나머지 값들은 상대되는 차원 값과 비교된다고 가정하자. 이와 같은 경우 하나의 차원 값이 3개의 차원 값과 비교가 되기 때문에 5차원인 각 벡터를 2차원씩 늘려서 7차원으로 만든 후 각 차원이 독립적으로 비교되도록 한다. 즉, $\vec{A}$는 $\vec{A'}$ = {a_1,a_2,a_3,a_4,a_5, a_1,a_1}이 되고, $\vec{B}$는 $\vec{B'}$ = {b_1,b_2,b_3,b_4,b_5,b_4,b_5}이 된다. 차원을 늘린 후에는 일반 히스토그램과 같이 본 논문의 인덱싱 알고리즘을 그대로 적용시켜 검색에 이용하도록 한다.

• Edge Component Histogram의 변환

Edge Component Histogram 기술자에서 유사도 측정을 위해 사용되는 연산식은 식 (4)와 같다. 식을 보면 알 수 있듯이, 전체 에지 히스토그램과 지역적인 에지 히스토그램 모두를 연산식에 이용하고 있다. 그러나 실제로 기술자를 통해 기술되는 것은 지역적인 에지 히스토그램뿐이기 때문에 유사도를 측정하기 위해서는 전체 에지 히스토그램에 관한 정보를 미리 계산하여 인덱싱 하여야 한다. 전체 에지 히스토그램은 지역적인 에지 히스토그램을 이용하여 구할 수 있기 때문에 미리 계산하는 것이 가능하다. 연산식에 사용되는 모든 정보를 구한 후에는 각 차원을 동일하게 처리할 수 있도록 변형하여야 한다. 식 (4)를 보면 다른 차원들은 동일하게 처리되는데 전체 에지 히스토그램에 대한 차원은 5가 곱해지는 것을 볼 수 있다. 이를 해결하기 위해서 전체 히스토그램에 관한 차원만 5개씩 확장하였다. 결국 5차원으로 표현되는 전체 히스토그램이 25차원으로 늘어나게 되어 식 (4)는 식 (5)와 같이 변형된다.

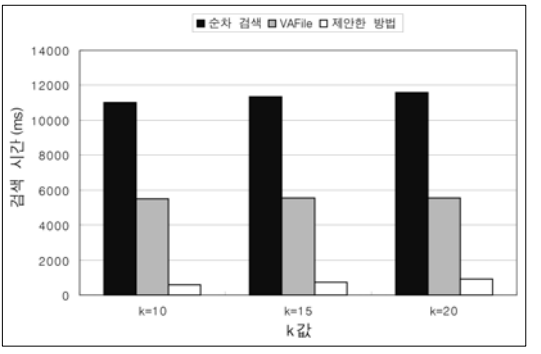
$$D(F_1,F_2)=\sum_{i=0}^{79}|h_1(i)-h_2(i)|+\sum_{i=0}^{24}|h^g(i)-h_2^g(i)|+\sum_{i=0}^{64}|h^s(i)-h_2^s(i)| \quad (5)$$

4. 실험 및 분석

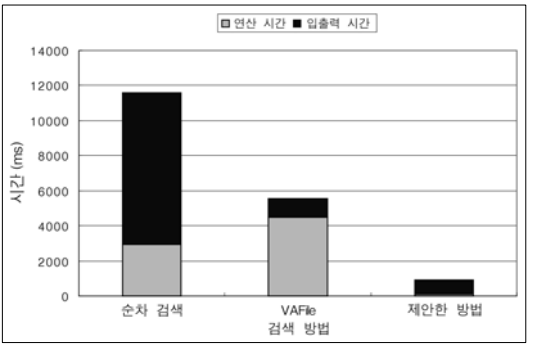
기본적으로 MPEG-7은 기술 언어인 DDL(Description Definition Language)을 이용하여 기술한다. 그러나 DDL은 XML기반의 텍스트 데이타이기 때문에 전체 용량이 커지고, 파싱하는 오버헤드가 있다. 이러한 오버헤드를 줄이기 위해서 DDL의 태그들은 제외하고 필요한 값들만 저장한 숫자 데이타를 이용할 수 있다. 본 논문에서 제안한 알고리즘의 효용성을 증명하기 위해 숫자 데이타를 이용한 순차 검색, VAFile[16]을 이용한 검색과 본 논문의 인덱스 데이타를 이용한 검색을 비교하였다. 본 논문에서 제안한 비트맵 인덱싱의 효용성 및 효율성을 증명하기 위해서 100,000 이미지 이상의 이미지 집합(12,861 COIL 이미지 and 99,074 COREL DRAW 이미지)에 대해 실험하였고, VA-File[16]을 이용한 검색 및 순차 검색과 비교하였다. 사용되는 저급 수준 내용 정보는 Color Structure 기술자를 이용하였고, 차원의 개수는 256으로 제한하였다. 본 논문에서 제시된 실험 결과는 1.5GHz CPU, 512MB 메모리, Microsoft Windows XP에서 수행된 결과이다.

그림 8은 검색 시간의 비교 결과를 보여주는 그림이다. 그림 8(a)는 111,935개의 오브젝트, 후보 오브젝트 리스트의 크기가 k*10인 경우, 찾는 개수인 k에 따른 검색 시간의 변화를 보여주는 그림이고, 그림 8(b)는 k가 15일 때의 프로파일을 보여주는 그림이다. 순차 검색의 경우에는 검색을 위해 모든 오브젝트의 저급 수준 내용 정보를 읽어야 하기 때문에 많은 입출력 시간이 필요하다. 반면에 VA-File[16]을 이용하는 검색의 경우에는 후보 오브젝트 리스트에 대해서만 저급 수준 내용 정보를 읽기 때문에 입출력 시간을 줄일 수는 있지만, 후보 오브젝트 리스트를 구하기 위해서 유클리드 연산법을 수행하기 때문에 많은 연산 시간이 필요하다. 그러나, 본 논문에서 제안한 비트맵 인덱싱의 경우에는 후보 오브젝트 리스트의 저급 수준 내용 정보만을 읽기 때문에 입출력 시간을 줄일 수 있고, 간단한 비트 연산자인 XOR를 이용하여 후보 오브젝트 리스트를 구하기 때문에 전체 연산 시간을 줄일 수 있다. 이와 같은 이유로 인해 본 논문에서 제안하는 비트맵 인덱싱 알고리즘은 VA-File[15]을 이용하는 검색 보다 7.5배 빠른 검색 시간을 보였고, 순차 검색 보다는 검색 시간이 15.3배 빨라졌다. 그림 8(c)는 제안하는 알고리즘의 확장성을 보기 위한 실험 결과로, 데이타베이스의 크기에 따른 각 검색 알고리즘의 검색 시간의 변화를 보여주는 그림이다. k는 15이다. 그림을 보면 알 수 있듯이, 입출력 시간과 연산 시간의 증가로 인해 순차 검색 및 VA-File [16]을 이용한 검색의 검색 시간은 비례적으로 증가하는 것을 볼 수 있지만, 제안하는 검색 알고리즘은 단순히 몇 번의 XOR만 추가적으로 필요하게 되기 때문에 많은 검색 시간의 증가를 보이지는 않았다.

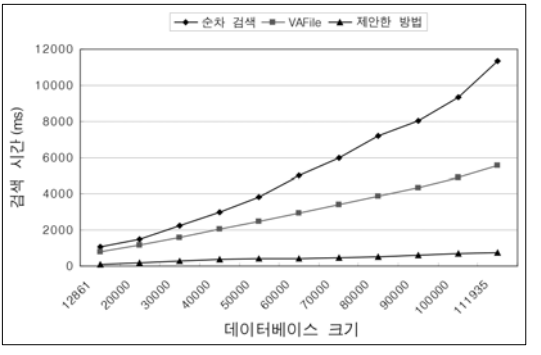
정확도를 측정하기 위해 Recall, Precision 을 계산하였다. VAFile[16]의 경우에는 정확성을 보증하는 인덱싱 알고리즘이기 때문에 환경에 상관 없이 100%의 Recall/Precision을 보여주는 반면, 제안하는 알고리즘은



(a) k에 따른 검색 시간 변화 (111,935 이미지)



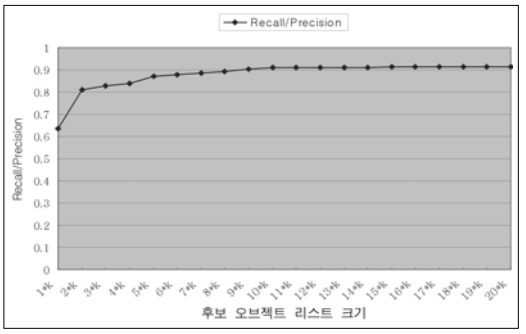
(b) 검색 시간 프로파일 (111, 935 이미지, k=15)



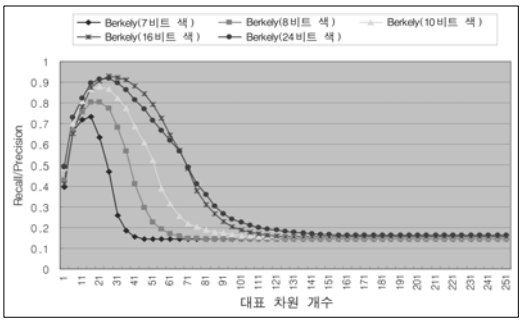
(c) 데이타 베이스 크기에 따른 검색 시간 변화 (k=15)

그림 8 순차 검색, VAFile을 이용한 검색 제안한 검색 방법의 검색 시간 비교

상황에 따라 정확도가 달라질 수 있다. 제안하는 알고리즘의 정확도는 후보 오브젝트 리스트의 크기와 대표 차원의 개수에 따라 변할 수 있다. 후보 오브젝트 리스트의 크기가 커지는 경우에는 보다 많은 오브젝트의 저급 수준 내용 정보를 읽고 비교하기 때문에 그림 9(a)에 나타나는 것처럼 Recall, Precision이 커지게 된다. 또한 그림 9(a)를 보면 알 수 있듯이, 후보 오브젝트 리스트의 크기가 어느 정도 이상인 경우에는 정확도가 많이 변하지 않는 것을 볼 수 있다. 물론, 후보 오브젝트 리스트의 크기가 계속 커지게 되면, 조금씩 정확도가 좋아져 결국 1.0이 되겠지만, 그만큼 저급 수준 내용 정보를 읽고, 비교 연산식을 수행해야 하는 오브젝트의 개수가 많아지기 때문에 검색 속도가 느려진다. 실험을 통해 10*k개의 후보 오브젝트 리스트인 경우에 0.9의 Recall/Precision 값을 보였고, 그보다 커지는 경우에는 큰 변화가 없음을 알 수 있었다. 일반적으로 멀티미디어 데이타의 검색은 텍스트 검색과는 다르게 정확하게 같은 데이타를 찾는 것이 아니라 비슷한 데이타를 찾는 것이다. 그렇기 때문에 한번의 검색으로 원하는 데이타를 찾는 방법보다는 연관성 피드백과 같은 방법을 이용하여 반복할수록 사용자가 원하는 형태의 검색을 할 수 있도록 해주는 방법이 대부분이다. 그러므로, 시스템을 구성하는 데 있어서 알고리즘의 속도가 중요한 역할을 하게 되고, 정확도는 연관성 피드백을 이용하여 높여줄 수 있게 된다.



(a) 후보 오브젝트 크기에 따른 정확도 변화(111,935 이미지)



(b) 대표 차원 개수에 따른 정확도의 변화

그림 9 Recall/Precision 실험

대표 차원의 개수가 너무 적거나 많으면, 각 오브젝트를 근사화하는 것이 의미가 없어지게 되기 때문에 대표 차원의 개수는 검색의 정확도에 큰 영향을 미친다. 그렇기 때문에 대표 차원 개수에 따라 정확도가 얼마나 변하는 지 실험을 통해 확인해 보았다. 실험에 사용한 이미지 집합은 버클리 대학에서 제공한 25,170개의 이미지(192x128, 24비트 색)로 이루어진 집합으로, 각 이미지의 색 비트수를 조정하면서 실험하였다. 그림 9(b)에서 보는 바와 마찬가지로, 제안한 알고리즘의 Recall/Precision 값은 대표 차원의 개수가 25개가 넘는 경우에는 대표 차원의 개수가 많아질수록 정확도가 떨어지는 것을 알 수 있다. 또한 색 비트가 줄어들수록 각 이미지 내에서 사용되는 색의 수가 적어지기 때문에 가장 높은 정확도를 가질 때의 대표 차원의 개수 또한 줄어드는 것을 알 수 있다. 이 실험을 통해 가장 효율적인 대표 차원의 개수는 이미지의 색 비트수와 연관이 되는 것을 알 수 있고, 그 변화의 정도는 크지 않는 것을 알 수 있다.

5. 결 론

데이타 압축 및 컴퓨터 처리 기술이 발달함에 따라 컴퓨터 상에서 멀티미디어 데이타 사용이 일반화되었고, 내용 기반 이미지 검색 시스템과 같은 응용 분야의 연구가 활발히 이루어졌다. 일반 텍스트 검색과는 달리 이미지 검색 시스템에서는 아주 많은 특성들을 이용하게 되고, 각 특성들이 고차원으로 표현되기 때문에 이미지의 특성을 읽고 비교하는 시간을 줄이는 것이 중요한 연구 과제가 되었다. 이를 위해 다양한 인덱싱 방법이 연구되었으나, 고차원에서는 검색 속도가 느려지는 문제점이 있다. 특히 멀티미디어 정보를 기술하기 위한 표준인 MPEG-7 시각 정보 기술자들은 대부분 고차원으로 표현되기 때문에 기존의 인덱싱 방법으로는 효율적인 검색을 할 수 없다.

본 논문에서는 MPEG-7 시각 정보 기술자들과 같은 고차원 데이타들에 대하여 빠른 검색 속도를 유지하면서 정확도를 높일 수 있는 새로운 멀티미디어 인덱싱 알고리즘을 제안하였고 MPEG-7시각 정보 기술자들을 본 논문의 알고리즘에 맞도록 변형하는 방법을 제시하였다. 기존의 여러 알고리즘들이 비슷한 오브젝트들을 하나의 인덱스로 클러스터링하여 검색을 하는 데 이용한 반면 본 논문의 알고리즘에서는 오브젝트별로 인덱스를 유지하도록 하였다. 즉, 세부적인 오브젝트의 특성을 유지 하였기 때문에 검색 성능(Recall, Precision)을

높일 수 있었다. 또한 각 차원마다 하나의 비트만을 사용하였기 때문에 인덱스 파일의 크기를 줄일 수 있었고, 검색을 위해 단순한 비트 연산을 이용하여 검색 속도를 향상시킬 수 있었다. 실험을 통해 본 논문의 알고리즘을 적용하는 경우, 90%이상의 정확도를 유지하면서, 15배 이상의 속도 향상 효과를 얻을 수 있음을 알 수 있었다.

오브젝트의 개수가 많은 경우, Exclusive OR연산 이후에 똑 같은 1의 개수를 갖는 데이타들이 많았다. 그로 인해 잘못 찾는 경우가 발생했는데, 보다 자세한 연산을 적용하여 검색 성능을 높일 수 있는 방법을 고안해야 할 것이다. 본 논문의 알고리즘은 VOD(Video On Demand)나 DVL(Digital Video Library)과 같은 다양한 응용 분야에 유용하게 이용될 수 있을 것이다.

참 고 문 헌

[1] M.Martinez, *Overview of the MPEG-7 Standard (version 5.0),* ISO/IEC JTC1/ SC29/WG11/ N4031, March 2001.

[2] A. Yamada, M. Pickering, S. Jeannin, L. Cieplinski, and Jens, *MPEG-7 Visual part of eXperimentation Model Verision 9.0*, ISO/IEC JTC1/ SC29/WG11/ N3914, January 2001.

[3] B.S. Manjunath, P. Salembier, and T. Sikora, *Introduction to MPEG-7*, John Wiley & Sons LTD, 2002.

[4] H. Lu, Y. Yeung Ng, and Z. Tian, "T-tree or B-tree : Main Memory Database Index Structure," *Proceedings of 12th Australasian Database Conference*, pp. 65-73, 2001.

[5] R. Orlandic, and B. Yu, "Implementing KDB-trees to Support High-Dimensional Data," *Proceedings of 2001 International Symposium on Database Engineering & Applications*, pp. 58-67, 2001.

[6] K. Lin, and C. Yang, "The Ann-tree: an Index for Efficient Approximate Nearest Neighbor Search," *Proceedings of 7th International Conference on Database Systems for Advanced Applications*, pp. 174-181, 2001.

[7] Q. Yang, A. Vellaikal, and S. Dao, "MB+-tree: A New Index Structure for Multimedia Databases," *Proceedings of International Workshop on Multimedia Database Management Systems*, pp. 151-158, 1995.

[8] N. Beckmann, H.P, Kriegal, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," *Proceedings of ACM SIGMOD Conference*, pp. 321-331, 1990.

[9] S. Berchtold, D.A. Keim, and H.P. Kriegel, "The X-tree: An Index Structure for High-Dimensional Data," *Proceedings of 22nd International Conference Very Large Databases*, pp. 28-39, 1996.

[10] N. Katayama, and S. Satoh, "The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries," *Proceedings of ACM SIGMOD Conference*, pp. 369-380, 1997.

[11] K.V.R. Kanth, D. Agrawal, and A. Singh, "Dimensionality Reduction for Similarity Searching in Dynamic Databases," *Proceedings of ACM SIGMOD Conference*, pp. 166-177, 1998.

[12] E. Tuncel, H. Ferhatosmanoglu, and K. Rose, "VQ-Index: An Index Structure for Similarity Searching in Multimedia Databases," *Proceedings of ACM International Conference on Multimedia*, pp. 543-552, 2002.

[13] G.H. Cha, X. Zhu, D. Petkovic, and C.W. Chung, "An Efficient Indexing Method for Nearest Neighbor Searches in High-Dimensional Image Databases," *IEEE Transaction on Multimedia*, Vol. 4, No.1, pp. 76-87, 2002.

[14] C.C. Aggarwal, and P.S. Yu, "The IGrid Index: Reversing the Dimensionality Curse for Similarity Indexing in High Dimensional Space," *Proceedings of ACM SIGKDD International Conference*, pp. 119-129, 2000.

[15] P.O. Neil, and D. Quass, "Improved Query Performance with Variant Indexes," *Proceedings of ACM SIGMOD Conference,* pp. 38-49, 1997.

[16] P. Weber, H. J. Scheck, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity Search Methods in High Dimensional Space," *Proceedings of VLDB Conference*, pp. 194-205, 1998.

정 진 국

1998년 2월 서강대학교 컴퓨터학과(공학사) 2000년 2월 서강대학교 컴퓨터학과(공학석사). 2004년 8월 서강대학교 컴퓨터학과(공학박사). 2004년 8월~ 삼성종합기술원 전문연구원. 관심분야는 멀티미디어 시스템, 동영상 분석, 동영상 인덱싱, MPEG-7

낭 종 호

1986년 2월 서강대학교 전자계산학과 졸업(학사). 1988년 2월 한국과학기술원 전산과 졸업(석사). 1992년 2월 한국과학기술원 전산과 졸업(박사). 1992년 3월~1992년 8월 한국과학기술원 정보전자 연구소(연구원). 1992년 9월~1993년 8월 일본 Fujitsu 연구소(방문연구원). 1993년 9월~현재 서강대학교 컴퓨터학과 교수. 관심분야는 멀티미디어 시스템, 동영상 검색, 동영상 분석