내용에 기반한 이미지 인덱싱 방법에 관한 연구

유 원 경† 정 을 윤††

요 약

대부분의 데이타베이스 시스템에서, 이미지는 캡션(caption), 주석(annotation),속성(attribute)과 같이 그 이미지와 관련된 텍스트를 이용하여 간접적으로 인덱스 되었다. 그러나, 이미지에 포함된 정보를 직접적으로 사용하여 내용에 기반한 이미지의 저장과 검색을 지원하는 이미지 데이타베이스 시스템의 요구가 점점 증가하고 있다. 내용에 기반한 몇몇 인덱싱 방법들이 있는데 그중에서 Petrakis는 이미지를 구성하는 오브젝트들의 공간관계와 속성을 고려한 이미지 인덱싱 방법을 제안했다. 이것은 '2-D string'에 기반한 인덱싱 연구의 확장인데, 이 방법은 많은 저장공간을 필요로 하며 융통성이 부족하다. 본 논문은 페이징 기법을 사용하는 kd-tree를 이용한 인덱스 화일구조를 제안한다. 그리고 정규화 과정을 사용해서 실제 이미지로부터 키를 추출하는 예를 보이고 시뮬레이션을 통해 비교하였다. 실험 결과는 제안된 방법이 훨씬 적은 저장공간을 요구하고, 융통성면에서 개선이 되었음을 보여준다.

A Study on Image Indexing Method based on Content

Weon Kyung Yoo† Eul Yun Chung††

ABSTRACT

In most database systems, images have been indexed indirectly using related texts such as captions, annotations and image attributes. But there has been an increasing requirement for the image database system supporting the storage and retrieval of images directly by content using the information contained in the images. There has been a few indexing methods based on contents. Among them, Petrakis proposed an image indexing method considering spatial relationships and properties of objects forming the images. This is the expansion of the other studies based on '2-D string'. But this method needs too much storage space and lacks flexibility. In this pager, we propose a more flexible index structure based on kd-tree using paging techniques. We show an example of extracting keys using normalization from the raw image. Simulation results show that our method improves in flexibility and needs much less storage space.

1. 서 론

기존의 데이타베이스 시스템은 대부분 일차원상의 숫자와 텍스트 데이타를 다루어 왔다. 기존의 데이타베이스 시스템에서 키워드에 기반한 인덱싱 기법들은 숫자와 텍스트 데이타를 다루는 사용자의 요구를 지원하는데 충분하다. 그러나 공간 데이타와 이미지 데이타는 이차원 혹은 삼차원이고 선형적인 관계들의 집합으로 직접적으로 저장될 수 없는 많은 정보를 포함한다. 이미지 정보는 지도, 컴퓨터 비젼에서 생성된 이미지, 기계 설계 도안, 구조 계획, shop-floor layout diagram의 이미지 범주를 포함한다[16].

최근의 이미지 프로세싱 기술과 이미지 저장 기술의 발전은 매우 큰 이미지 데이타베이스의 생성을 가능하게 만들었다. 이미지 데이타베이스 시스템은 이미지 정보 시스템의 핵심 부분이다. 이미지 정보 시스템은 GIS's(geographical infor-

* 이 논문은 1994년도 재단법인 운정재단 학술연구조성비 지원에 의하여 연구되었음
† 종신회원 : 성신여자대학교 전자계산학과 교수
†† 정 회 원 : NDS 경영지원실 근무
논문접수 : 1995년 7월 12일, 심사완료 : 1995년 11월 22일.

mation systems), OA(office automation), PACS's(medical picture archiving and communication systems), CAD(computer-aided design), CAM(computer-aided manufacturing), CAE (computer-aided engineering), SD(scientific databases)등의 많은 응용(application)에서 사용된다. 이미지 정보 시스템에서는 키워드에 기반한 기법들에 의해 적당하게 지원될 수 없는 많은 응용들이 있다[6]. '[6]에서는 전형적인 이미지 정보 시스템들의 분류를 보여주고 이미지 데이타베이스의 일반적인 경향을 다루고 있다.

이미지 데이타베이스 시스템은 다양한 이미지 표현(representation)과 이미지 프로세싱 기능(이미지 특징 추출), 내용에 기반한 검색, 그리고 효율적인 메카니즘을 제공해야만 한다[18]. 이미지 데이타를 효율적으로 검색하고 저장할 수 있는 이미지 데이타베이스를 이용함으로써 이미지 정보 시스템은 실시간에 많은 양의 데이타를 처리할 수 있다. 저장되는 데이타양, 데이타 처리, 통신 부담 등은 간결한 이미지 표현을 이용함으로써 줄여질 수 있다[14].

내용에 의한 이미지 검색에 있어서의 주요 문제점은 이미지의 내용을 정확하게 정의하고 해석하는데 어려움이 있다는 것이다. 이미지의 내용은 인간의 견지나 응용 범위에 따라 다른 해석을 가질 수 있다. 또한 이미지에 포함된 오브젝트들을 인식하기 어렵고 이들 오브젝트들의 상호관계(mutual relationship)는 표현되기 어렵다. 따라서 대부분의 시스템에서의 이미지 검색은 이미지에 포함된 정보에 의한 직접적인 검색이 아니라 캡션(caption), 주석(annotation), 이미지 속성(예 : 이름)들과 같이 이미지와 관련된 텍스트를 통한 간접적인 검색이다[17,18]. 그러나, 이미지에 포함된 정보에 의한 직접적인 검색, 즉 내용에 기반한 이미지 검색은 이미지 데이타베이스 시스템의 기본요소로 점점 중요하게 되었다[19].

내용에 기반한 이미지 검색 기법 연구로는 [8, 11, 18]이 있다. [8]에서는 오브젝트의 윤곽선으로부터, [11]에서는 오브젝트를 구성하는 사각형들로부터 이미지 정보를 추출하여 이 정보를 가지고 다차원 인덱스 구조(multidimensional index structure)를 사용하여 질의 이미지와 같거나 비슷한 이미지를 검색하는 모양 유사성에 기반한 이미지 검색방법이 제안되었다. [8, 11]에서 제안된 방법들은 이미지안에 검색 대상이 되는 오브젝트를 하나만 고려하고, 모양이나 크기와 같은 속성만을 인덱싱하는데 사용하고 있다. [18]에서는 이미지내 오브젝트들의 존재를 검증이론(the theory of evidence)을 사용해서 해석하여 이미지 내용의 해석을 얻는 유사 검색 방법과 오브젝트 속성와 오브젝트들의 상대적 위치를 사용하는 이미지 검색방법이 제안되었다. 국내에서도 광운대학교에서trie 구조를 이용한 인덱스기법을 제안하였다[20].

이미지 오브젝트들은 원래의 이미지(raw image)를 분석하여 인식되는데, 일반적으로 오브젝트들은 좌표(coordinates)와 공간 속성(spatial properties)을 가지며 데이타베이스에서 이미지를 검색할 때 이미지들을 구별하는 가장 중요한 방법 중의 하나는 이미지에 있는 오브젝트들과 이 오브젝트들 사이에 존재하는 공간관계(spatial relationship)를 인식하는 것이다.

이미지 프로세싱의 전반적인 처리 효율에 중요한 역할을 하는 것은 이미지 지식구조로써 이미지의 내용을 간결하게 표현하고 효율적으로 처리될 수 있는 구조여야 한다. 또한 지식구조는 객체 지향적(object oriented)이어야 하고 특히 공간 관계등과 같은 이미지에서 나타나는 지식을 가능한 많이 포함할 수 있어야 한다[5]. 이러한 목적을 위해서 Chang[4]은 이미지 안의 오브젝트들의 속성들과 공간관계에 기반한 이미지 인덱싱을 위하여 '2-D 스트링'이라 하는 공간 지식구조(spatial knowledge structure)의 사용을 제안하였다. 원래부터 이미지 데이타베이스 환경에서의 사용을 위해 2-D 스트링은 고안되었고 아이콘 인덱싱과 공간 추론을 위한 효율적인 수단을 제공한다[5].

'2-D 스트링'에 기반하여 이미지 검색을 하고자 하는 여러 연구[3, 5, 9, 10, 13, 14, 15, 18]을 살펴보면 다음과 같다. [5]에서는 '2-D 스트링'을 기반으로하여 효율적인 인덱싱, 융통성 있는 정보 검색, 공간 추론과 시각화(visualization)를 위한 기법이 제시되었다. 공간 관계는 퍼지 개념이기 때문에 부이미지(subpicture)에 의한 검색

을 위해서 유사 검색(similarity retrieval)의 능력이 필수적이다[17]. Lee[9]는 이'2-D 스트링'을 이용하여 이미지 데이타베이스에서 유사한 이미지를 검색하기 위한 세가지 종류의 이미지 질의를 제시하였다. 여기에서의 유사성 측도(measure)는 비교되는 두개의 '2-D 스트링'이 공통적으로 갖는 부분이 어느 정도인가이다. [10]에서는 복잡한 모양을 가진 중첩되는 오브젝트들로 구성된 이미지인 경우도 다룰 수 있도록 이미지 내의 공간 정보를 더욱 자세히 표현하는 '2D C-스트링'이라는 새로운 공간 지식 표현이 제안되었고 이에 기반한 유사 이미지 검색의 알고리즘이 또한 제안되었다.

질의 처리는 질의로 들어온 이미지를 '2-D 스트링'으로 변환해서 데이타베이스에 이미 저장되어 있는 이미지들의 모든 '2-D 스트링'과 비교하여 같으면 질의 이미지와 같은 이미지라 하여 검색하는 방식으로 진행된다. 이렇게 이미지 검색을 위해 픽셀 이미지를 모두 비교하는 것과는 달리 스트링 매칭으로 하여 좀 더 빠르게 검색할 수 있다. 만약 이미지 데이타베이스에 저장된 이미지의 수가 굉장히 많다면 하나의 이미지 질의를 처리할 때 데이타베이스에 있는 전체 '2-D 스트링'을 다 비교하는 저장된 이미지 데이타 수에 따른 선형적 검색(exhaustive search)이 되어 문제가 된다. 따라서 탐색공간을 줄이고 검색속도를 높일 수 있는 효율적인 인덱싱 방법이 필요하게 되었다. [18]에서는 '2-D 스트링'과 오브젝트 속성들을 함께 사용하는 'attributed 2-D 스트링' 방법을 사용하여 이미지 내용 표현을 강화했으며 '2-D 스트링' 매칭전에 오브젝트 속성에 대한 조건을 처리함으로써 탐색범위를 줄여 효율성을 개선하고 있다.

'2-D 스트링'에 기반하여 내용(content)에 의해 검색을 지원하는 인덱싱 방법 연구로는[3, 13, 14, 15]가 있다. [8, 11]에서 사용한 방법은 검색대상이 되는 오브젝트가 하나인 반면에 위의 인덱싱 방법들은 여러 오브젝트들을 포함하는 이미지를 다룰 수 있다. 그러나 위의 인덱싱 방법들은 저장공간을 많이 필요로하며, 새로운 종류의 이미지가 삽입되는 환경에서는 적합하지 않다.

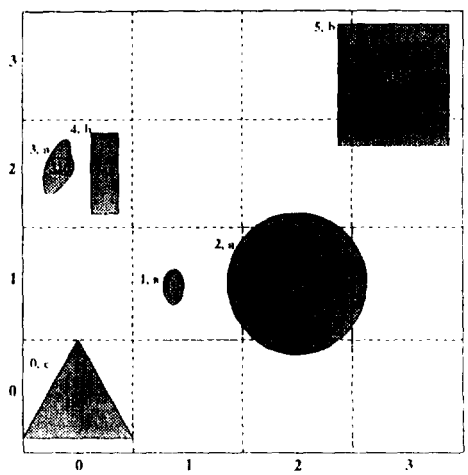
본 논문에서 이러한 문제점을 해결하기 위한 방법으로 kd-tree를 이용한 인덱스 화일구조의 사용을 제안하고, 이미지의 정규화 방법과 실제 이미지를 가지고 인덱스를 추출한 결과와 제시된 구조를 구현하여 시뮬레이션한 결과를 보인다.

2. 이미지 표현

2.1 '2-D 스트링' 표현

주어진 이미지의'2-D 스트링' 표현을 생성하기 위해서 먼저 이미지를 세그먼트화하고 모든 오브젝트들의 위치(center of mass)를 얻는다. x축으로는 왼쪽에서 오른쪽으로, y축으로는 아래에서 위로 오브젝트들을 취함으로써 그 이미지의 '2-D 스트링' 표현을 구성한다. 오브젝트들 자체는 그 오브젝트가 속한 클래스 혹은 이름에 의해 표현된다.

(그림 1)에서 오브젝트들의 왼쪽에 있는 숫자와 문자는 각각 오브젝트의 이름과 클래스이고 가운데에 있는 숫자는 오브젝트들의 픽셀수이다. (그림 1) 이미지 안에 포함된 오브젝트들은 세 개의 클래스 a,b,c중의 하나에 속하고, 0에서 5까지 이름이 붙여져 있다. 주어진 이미지의 '2-D 스트링' 표현은 'normal', 'absolute', 'augmented' 등의 다양한 형태를 가질 수 있다[4]. (그림 1)에 대한 'augmented 2-D 스트링' 표현



(그림 1) 예제 이미지
(Fig. 1) An example image

은 다음과 같다.

$$(u, v) = (0\,3\,4 < 1 < 2 < 5,\ 0 < 1\,2 < 3\,4 < 5)$$

기호 '<'는 스트링 u에서는 '왼쪽/오른쪽' 관계를 나타내고 스트링 v에서는 '아래/위'의 관계를 나타낸다. [15]에서는 'expanded 2-D 스트링'의 형태를 이용하였는데 이것은 'augmented 2-D 스트링' 표현 (u, v)를 세개의 일차원 스트링들의 형태 (x,r1,r2)로 확장한 것이다. 여기서 x는 u에 나타난 것과 같은 순서로 모든 오브젝트들의 클래스 혹은 이름을 포함하고, 스트링 r1과 r2는 x에 나타난 오브젝트들에 대한 x축, y축으로의 좌표를 각각 포함한다. (그림 1)의 'expanded 2-D 스트링'은 다음과 같다.

$$(x, r_1, r_2) = (0\,3\,4 < 1 < 2 < 5,\ 0\,0\,0\,1\,2\,3,\ 0\,2\,2\,1\,1\,3)$$

n개의 오브젝트를 포함하는 한 이미지로부터 이미지 부분집합(image subset)들의 집합이 생성된다. 각 집합에서 부분집합들은 똑같이 k개의 오브젝트들을 포함한다. k는 2에서부터 미리 정의된 수 K_{max}까지이다. 즉 k[2, min(n, K_{max})]이다. k개의 오브젝트를 가지는 이미지 질의가 들어온다면 k개의 오브젝트를 가지는 이미지 부분집합들의 집합이 질의에 대한 탐색 대상이 된다. 그러므로 Kmax는 미리 알 수 있다면 질의에 허용되는 최대 오브젝트 수로 정의할 수 있다. 일반적으로 여러 응용에 따라 K_{max}의 값이 달라질 수 있다. n개의 오브젝트들을 포함하는 한 이미지로부터 생성되는 모든 부분집합의 갯수는 다음과 같다.

$$\sum_{k=2}^{\min(n,\,K_{max})} \binom{n}{k} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2.1)$$

인덱싱에 앞서 이미지 부분집합에 있는 오브젝트들은 순서화 되어야 한다. 그런 다음 각 오브젝트들의 공간관계와 속성들의 값이 계산된다. 이 중에서 공간관계, 즉 오브젝트들의 위치는 오브젝트들을 구별하는 중요한 기준이 되기 때문에 인덱싱하는데 기본키가 될 수 있다[14]. 인덱싱을 하기 위해 사용되는 순서기준(ordering criterion)에는 일차 순서기준 (First Ordering Criterion)과 이차 순서기준 (Second Ordering Criterion) 두가지가 있다.

(a[0], a[1], ···, a[k-1])를 크기 k의 이미지 부분집합이고, 각각 (cg_x[i], cg_y[i])와 (cg_x[j], cg_y[j])를 중심좌표(centers of mass)로 가지는 두 오브젝트 a[i], a[j] (i,j∈[0, k-1])에 대해 a[i] < a[j]는 a[i]가 a[j]의 선행자(predecessor)임을 뜻하고 a[i] > a[j]는 a[i]가 a[j]의 후행자(successor)임을 뜻한다고 하자.

이 때, 일차 순서기준은 다음과 같다.

$$\forall i, j \in [0,\ k-1] \begin{cases} a[i] < a[j], \text{ if } cg_x[i] < cg_x[j] \text{ OR } cg_x[i] < cg_x[j] \text{ if } cg_x[i] = cg_x[j]; \\ a[i] > a[j], \text{ otherwise.} \end{cases}$$

이미지 부분집합에 포함된 오브젝트들에 이 순서기준을 적용함으로써 첫번째 순열 스트링(permutation string) p가 얻어진다. p는 오브젝트들에 해당하는 인덱스의 순서열이다.

두번째 순열 스트링 p′는 아래의 순서기준에 의해 정해진다.

$$\forall i, j \in [0,\ k-1] \begin{cases} a[i] < a[j], \text{ if } cg_y[i] < cg_y[j] \text{ OR } cg_y[i] < cg_y[j] \text{ if } cg_y[i] = cg_y[j]; \\ a[i] > a[j], \text{ otherwise.} \end{cases}$$

또한 랭크(rank)가 정의될 수 있다. p에서 i번째 오브젝트 p[i]의 랭크 r[i]를 스트링 p'에서 p[i]오브젝트를 선행하는 오브젝트들의 수로 정의한다.

$$r[i] = j \leftrightarrow p[i] = p'[j],\ 0 \le i,\ j < k$$

랭크 스트링 r은 만약 같은 중심좌표를 가지는 오브젝트가 없다면 집합 {0, 1, ···, k-1}에 대한 순열로서 k!개의 값을 가질 수 있다. 따라서 순서화된 이미지 부분집합 p는 랭크 스트링 r의 순위를 계산함으로써 D_k=k!크기의 주소 공간상의 유일한 주소 I_k로 매핑이 된다.

다음으로 이차 순서기준을 살펴 보겠다. 이미지 영역(area)을 M x N rectangular grid로 나눈다. 각 cell은 0에서 (M×N−1)까지 인덱스되며 각 오브젝트의 랭크 값 r[i]는 그 오브젝트의 중심좌표 (cg_x[i], cg_y[i])을 포함하는 grid cell의 인덱스이다. 여기서 X와 Y는 각각 이미지의 x축, y축의 grid의 실제 픽셀 수이다.

$$r[i]=Scg_x[i]\cdot M+Scg_y[i] \quad \cdots (2.2)$$

$$Scg_x[i]=\left(\frac{cg_x[i]}{X}\cdot N\right),\ Scg_y[i]=\left(\frac{cg_y[i]}{Y}\cdot M\right)$$

각각 r[i], r[j]의 값을 가진 두 오브젝트 a[i], a[j], i, j∈[0, k-1]에 대해 이차 순서기준은 다음과 같다.

$$\forall i,j\in[0,k-1]\begin{cases} a[i]<a[j], \text{ if } r[i]<r[j] \text{ OR } x[i]<x[j] \text{ if } r[i]=r[j]; \\ a[i]>a[j], \text{ otherwise.}\end{cases}$$

같은 위치를 갖는 오브젝트들은 이름에 의해 순서화된다. 한 이미지 부분집합에 포함된 오브젝트들을 이차 순서기준에 적용함으로써 순열 스트링 p(오브젝트 인덱스들의 순서열)와 랭크 스트링 r(p에 있는 순서화된 오브젝트들에 대해 계산된 위치 r[i])을 얻는다. 이차 순서기준을 사용할 때 순서화된 이미지 부분집합에 포함된 오브젝트들 사이의 공간 관계는 스트링 r로 표현된다.

전체 grid의 수가 t=M×N인 곳에 k개의 오브젝트들이 놓일 수 있는 경우의 수는 t개의 위치에서 순서에 관계없이 중복을 허락하여 k개를 뽑는 조합의 수 $_tH_k$와 같다. 그러므로 스트링 r이 가질 수 있는 서로 다른 값은 $_{k+t-1}C_k$이다. r스트링의 순위를 계산함으로써 순서화된 이미지 부분집합 p는 $D_k^t=_{k+t-1}C_k$ 크기의 주소 공간의 유일한 주소 I_k^t에 매핑된다.

이미지내에 있는 오브젝트들 사이의 공간관계(오브젝트들의 위치)는 이미지를 검색할 때 이미지들을 구별하는 가장 중요한 방법 중의 하나이다. 즉, 이미지내에 있는 오브젝트들의 위치로도 이미지를 충분히 구별할 수 있다. 그러나 같은 위치를 가지는 오브젝트들이 있는 경우에는 모호성이 있게 된다. 이러한 경우 이미지 부분집합을 순서화 하기 위해 위치 속성외에 각각의 오브젝트의 특징들을 설명하는 오브젝트들의 속성(색깔, 크기, roundness, orientation, perimeter…)들을 고려할 수 있다. 많은 속성들이 고려되어진다면 탐색범위도 줄어들 뿐 아니라 표현의 정확성으로 더욱 정확한 이미지 검색을 할 수 있다. 순서화된 이미지 부분집합 p가 주어지면 각각의 속성로부터 개별적인 속성 스트링(attribute string) $(u_0, u_1, \cdots, u_{k-1})$을 얻을 수 있다.

$$\forall i\in[0,k-1],\ u_i=\left(\frac{p[i]\text{의 속성값}}{\text{최대 속성값}}\times q\right)$$
$$\text{if } (p[i]\text{의 속성값}=\text{최대 속성값}),\ u_i=q-1 \quad \cdots\cdots (2.3)$$

최대 속성값은 각각의 속성들에 대해 다른 값을 가진다. 예를들면 roundness는 최대값으로 1(원일 경우)을, orientation의 경우 최대값으로 π= 3.141을 가질 수 있다. q는 'quantization value'라 불리는 정수로서 한 속성에 대해 한 오브젝트가 속할 수 있는 범주의 수이다.

크기가 k이고 오브젝트가 그 속성에 대해 가질 수 있는 값이 q개인 속성 스트링 x는 q개에서 순서를 생각하면서 중복을 허락하여 k개를 뽑는 조합의 수의 하나에 매핑될 수 있다. 즉 주소공간의 크기는 $D_k^x=_q\prod_k$이다. 순서화된 이미지 부분집합 p는 속성 스트링 x의 순위를 계산함으로써 D_k^x 크기의 주소공간 내의 유일한 주소 I_k^x에 매핑된다. 인덱싱을 하기위해 사용되는 이미지의 속성들은 이미지를 잘 특성짓는 속성이어야 하고 고려되는 응용에 따라 달라질 수 있다[14].

2.2 이미지 인덱싱과 저장

각 이미지의 순서화된 부분집합 p는 r, s, o, c, p … 형태의 스트링들로써 표현된다. 여기서 r, s, o, c, p, …는 각각의 오브젝트들의 특정한 속성에 해당하는 속성 스트링이다. 예를들어 r은 오브젝트들 사이의 '왼쪽/오른쪽', '아래/위' 관계를 나타내는 랭크 스트링, s는 '크기' 스트링, o는 'orientation' 스트링, c는 오브젝트들이 속하는 '클래스' 스트링, p는 '주변길이(perimeter)' 스트링이다.

이미지들은 자신으로부터 유도된 모든 부분집합들의 표현에 의해서 인덱스된다. 이미지로부터 'expanded 2-D 스트링'을 얻고 n개의 오브젝트를 포함하고 있는 하나의 이미지는 $\{p_k^l \mid 2\le k\le \min(n, K_{max}),\ 0\le l<{}_nC_k\}$개의 순서화된 이미지 부분집합으로 나누어진다. 속성 스트링의 갯수를 v라 하고 i번째 속성 스트링에 대해 계산된 순위를 $d_k^{l,i}$ $(0\le i\le v)$라 하자. [13, 14, 15]에서 크기 k의 l번째 이미지 부분집합 p_k^l의 키값 I_k^l 은 각 속성 스트링의 순위로부터 계산된다.

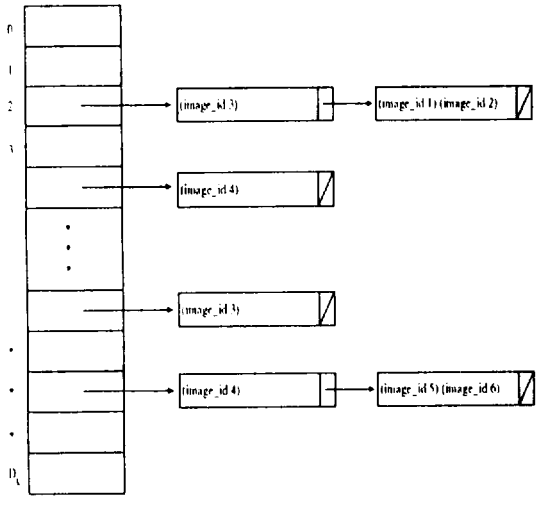
$$I_k = d_k^{0} + d_k^{1} \cdot D_k^{0} + d_k^{2} \cdot D_k^{0} \cdot D_k^{1} + \cdots + d_k^{l-1} \prod_{i=0}^{l-2} D_k^{i} \quad (2 \le k \le \min(n, K_{max}),\ 0 \le l < r, C_k \quad \cdots\cdots (2.4)$$

위의 식에서 명시된 주소 공간의 크기는

$$D_k = \prod_{i=0}^{k-1} D_k^{i} \quad 2 \le k \le K_{max} \text{ 이다.}$$

기존의 방법에서는 인덱싱을 위해 해쉬 화일 구조를 사용하였다. 크기 k개의 모든 이미지 부분집합은 H_k화일 안에 저장된다. H_k화일 구조는 (그림 2)와 같다. 화일 H_k는 고정된 크기의 데이타 페이지들의 집합으로 구성된다. 데이타 페이지의 연결된 리스트(linked list)로 오버플로우(overflow)를 처리한다. IDB(Image DataBase)는 각각 2, 3, …, K_{max}개의 오브젝트들로 구성된 부분집합들의 인덱스를 저장한 H_2, H_3, …, H_{Kmax}의 화일구조의 집합으로 구성된다. 이와 같은 구성으로 인하여 부분질의도 가능하다.

즉, 이미지의 일부분이 질의로 들어올 때 그 일부분을 포함하는 모든 이미지가 검색될 수 있다. 이미지 식별자(image id)는 물리적 데이타베이스에 저장된 원래 이미지에 대한 포인터가 된다. 2-D 스트링 표현을 포함하여 원래 이미지에 관련된 정보들이 이미지 식별자와 함께 저장될 수 있다.



(그림 2) D_k 크기의 주소공간을 가지는 H_k 화일 구조
(Fig. 2) File structure H_k with address space size of D_k

2.3 이미지 검색

이미지 저장과 검색과정은 다음과 같다. 새로운 이미지가 삽입되면 이미지 분석과정을 거쳐 '2-D 스트링'과 속성 스트링이 추출되고 이것으로부터 키값이 계산되어 인덱스 화일이 생성된다. 생성된 인덱스 화일들은 논리적 데이타베이스(logical database)에 저장되고 원 이미지는 물리적 데이타베이스(physical database)에 저장된다. 그리고, 논리적 데이타베이스와 물리적 데이타베이스에 저장되어 있는 이미지를 포인터로 연결한다.

같은 분석과정을 거쳐서 질의로 들어온 이미지의 키값을 계산하고 이 키값으로 논리적 데이타베이스에 저장된 인덱스 화일을 탐색한다. 다음, 포인터를 따라 키값을 가지는 이미지가 물리적 데이타베이스에서 검색된다(hypothesis generation). 최종적으로 사용자는 이들 검색된 질의 후보 이미지들에서 원하는 이미지를 선택할 수 있다(verification).

질의는 직접접근질의(direct access query)와 간접접근질의(indirect access query)로 구별될 수 있다[14].

직접접근 질의는, 질의 오브젝트의 수를 m이라 할때 m의 범위가 $2 \le m \le K_{max}$인 질의이다. 데이타 페이지에 '2-D 스트링'이 저장되어 있다면 '2-D 스트링 매칭 알고리즘'을 이용하여 질의 이미지의 '2-D 스트링'과 데이타 페이지에 저장된'2-D 스트링'을 비교한다.

간접접근 질의는 $m > K_{max}$인 질의이다. 질의 이미지는 K_{max} 크기의 부분집합으로 분해된다. 이것은 $\rho = {}_mC_{Kmax}$개의 개별적인 직접접근질의가 된다. 이 질의에 대해 검색된 이미지 집합 S_0, S_1, …, $S_{\rho-1}$가 얻어지고 이들의 교집합 $S = S_0 \cap S_1$, …, $S_{\rho-1}$는 원래 질의 이미지와 매칭되는 후보 이미지들의 식별자를 포함한다.

3. K-d tree를 이용한 인덱싱 방법

3.1 기존 인덱싱 방법의 문제점

Chang이 [3]에서 제안한 인덱싱 방법은 Cook과 Oldehoeft가 제안한 알고리즘[7]을 사용하여 순서화된 이미지 부분집합에 대한 키값을 계산한

다. 그런데 키값을 부여하기 위해 기준이 되는 순서를 오브젝트들의 빈도수로 계산하기 때문에 데이타베이스에 저장되는 이미지들을 미리 알아야 한다는 문제점을 가지고 있다. 따라서 이미지가 데이타베이스에 새로이 들어오게 된다면 이 알고리즘은 같은 속성을 가지지 않는 순서화된 이미지 부분집합이 같은 키값을 가지도록 하여 적합치 않다. 또한 이 순서화된 이미지 부분집합은 단지 두개의 오브젝트들로만 구성되기 때문에 중간 질의 결과의 초과 처리 때문에 검색 성능의 큰 오버헤드를 가진다. 예를 들면 만약 질의 이미지가 5개의 오브젝트들로 구성되어 있으면 $5\times4/2!=10$, 6개이면 $6\times5/2!=15$개의 순서화된 이미지 부분 집합이 만들어진다. 이 순서화된 이미지 부분 집합 각각이 이미지 집합을 검색하기 위한 구별된 질의가 된다. 이때 검색된 이미지 집합의 교집합은 원래 질의와 비교될 이미지를 포함한다. 따라서 질의 이미지에 포함된 오브젝트들의 수가 커지면 중간 질의도 더욱 많아지므로 중간 처리에 많은 오버헤드가 생긴다. 그러나 이 알고즘은 해쉬 테이블의 빈 슬롯(slot)이 적다는 장점이 있다.

[13, 14, 15]에서 제안된 방법은 [3]의 방법을 확장하여 두개 이상의 오브젝트들로 이루어진 그룹에 대해서도 인덱싱을 한다. 질의 이미지에 포함된 오브젝트 갯수의 크기를 가진 부분집합으로 이루어진 화일을 직접 찾아 검색하도록 함으로써 중간 처리 오버헤드를 줄여서 검색시간을 개선하고 있다. 그러나 각 그룹에 대한 인덱스 화일의 저장으로 저장 비용은 더 커지게 된다. 오브젝트들의 공간 관계에 대해서 '2-D 스트링'으로 표현한 것을 확장하여 객체들 사이의 포함관계, orientation등 많은 수의 오브젝트 속성(property)을 고려하고 있다. 즉 질의 이미지에 보다 비슷한 이미지를 검색하도록 한다.

[3]에서의 방법은 데이타베이스에 이미 저장된 오브젝트들을 포함하는 이미지가 들어와도 해쉥이 완전하게 되지 않는 반면에 [13, 14, 15]의 방법은 이미 저장된 오브젝트들로 구성된 이미지가 들어오면 주소 공간이 확보되어 있기 때문에 문제가 없다. 그렇지만 속성의 새로운 범주에 속하는 오브젝트를 가진 이미지가 들어온다면 이 방법도 주소 공간이 변하여서 다시 해쉬 테이블을 구축해야 하는 문제점을 가지고 있다. 또한 만약 실제로 저장되는 이미지가 적고 같은 위치를 가지는 오브젝트들로 구성된 이미지가 많이 들어 온다면 오브젝트들의 모든 위치의 경우를 생각하여 주소공간을 확보해 놓는 것은 문제가 된다.

[13, 14, 15]에서의 방법을 따르면 이차 순서기준을 사용하고 인덱싱을 하기 위해 사용되는 속성으로 오브젝트들의 공간관계와 클래스, 그리고 크기가 고려된다고 할 때 (그림 1)의 이미지의 크기가 4인 부분집합의 세번째 순서화된 부분집합 {0, 1, 2, 5}는 (0, 5, 9, 15)의 랭크 스트링과 (c a a b)의 클래스 속성 스트링, 그리고 (1 0 2 2)의 크기 속성 스트링을 갖는다. 그러므로 부분집합 {0, 1, 2, 5}의 $Ik_i'=3295$, $I_k'=55$,$I_k'=35$, 그리고 $I_k'=3295+55\times3876+35\times3876\times81=11204935$이다. 화일 H_4의 가능한 키 값의 범위, 즉 주소공간 $D_4=3876\times81\times81=25430436$이다. 그런데 quantization값 q가 늘어나는 경우를 생각해보자. 여기서는 quantization 값이 늘어나는 속성을 클래스 속성이라 하자. 새로운 클래스 d에 속하는 오브젝트가 들어오는 경우 새로운 클래스가 고려되어야 함으로써 I_k'의 크기가 4'로 커져 주소 공간 $D_4=3876\times256\times81=80372736$이 된다. 주소공간이 약 3.1604배 커진다. 그리고 부분집합 {0, 1, 2, 5}는 새로운 키값 $I_k'=35232259$을 가진다. 따라서 해쉬 테이블은 새롭게 구성되어야 하며 그 공간도 q가 커짐에 따라 지수적으로 증가하게 된다. 그러나, q가 커짐에 따라 주소공간의 크기는 지수적으로 증가하나, 표현은 더 정확해지므로, q가 커지더라도 실제적인 저장공간의 증가없이 큰 주소공간을 표현할 수 있는 인덱스 구조가 필요하다.

〈표 1〉은 오브젝트의 공간관계와 하나의 속성이 고려될 때의 주소공간의 변화이다. 〈표 1〉에서 보듯이 rectangular grid가 3−5 , q가 3−5라고 했을 경우에도 주소 공간의 크기가 매우 큼을 알 수 있다. D_k는 k와 v의 값이 커지면 상대적으로 매우 크게된다.(예 : $k>4$와 $v>2$일때, D_k는 10^9을 가진다[14].)

만약 이미지를 표현하는 오브젝트들의 속성 스트링의 갯수가 많아진다면 기존의 해쉬 구조를 이용한 방법은 문제를 가지게 된다. 이렇게 큰 주소공간을 처리하기 위해 [14]에서는 해결방안으로

collison을 허락하였다. collison을 허락하는 경우 다른 표현을 가지는 이미지 부분집합들이 같은 키 값을 가지게 되어 질의에 대한 검색 결과가 부정확하게 된다. 즉 검색 결과에 질의에 대한 응답으로 적합하지 않은 이미지가 자주 검색될 수 있다.

〈표 1〉 주소공간의 변화
〈table 1〉 Changes in address space

k	q	3 × 3 rectangular grid D_k^0	D_k^1	$D_k(D_k^0 \times D_k^1)$	4 × 4 rectangular grid D_k^0	D_k^1	$D_k(D_k^0 \times D_k^1)$	5 × 5 rectangular grid D_k^0	D_k^1	$D_k(D_k^0 \times D_k^1)$
2	3	$_9H_2$	3^2	405	$_{16}H_2$	3^2	1224	$_{25}H_2$	3^2	2925
	4		4^2	720		4^2	2176		4^2	5200
	5		5^2	1125		5^2	3400		5^2	8125
3	3	$_9H_3$	3^3	4455	$_{16}H_3$	3^3	22032	$_{25}H_3$	3^3	78975
	4		4^3	10560		4^3	52224		4^3	187200
	5		5^3	20625		5^3	102000		5^3	365625
4	3	$_9H_4$	3^4	40095	$_{16}H_4$	3^4	313956	$_{25}H_4$	3^4	1658475
	4		4^4	126720		4^4	992256		4^4	5241600
	5		5^4	309375		5^4	2422500		5^4	12796875
5	3	$_9H_5$	3^5	312741	$_{16}H_5$	3^5	3767472	$_{25}H_5$	3^5	28857465
	4		4^5	1317888		4^5	15876096		4^5	121605120
	5		5^5	4021875		5^5	48450000		5^5	371109375
6	3	$_9H_6$	3^6	2189187	$_{16}H_6$	3^6	39558456	$_{25}H_6$	3^6	432861975
	4		4^6	12300288		4^6	222265344		4^6	2432102400
	5		5^6	46921875		5^6	847875000		5^6	9277734375

3.2 kd-tree를 이용한 인덱싱 방법

부분집합의 크기 k, primary attribute의 수, quantization 값 q , grid size가 커짐에 따라 키 값(인덱스)이 있을 수 있는 주소공간은 엄청나게 커지게 된다. 주소 공간이 엄청나게 커지는데 비해서 어느 특정한 응용에 관련된 이미지들은 거의 비슷한 특징을 가지기 때문에 산출되는 주소는 일부분에 해당할 것이다. 그러므로 모든 상황을 고려한 기존의 방법 [13, 14, 15]들은 적당하지 않게 된다. [13, 14, 15]에서 사용한 키 계산법과 해쉬를 사용한 화일 구조는 주소 표현의 문제와 필요한 주소공간을 확보해야만 하는 인덱스를 위한 저장 공간의 오버헤드 문제를 가지고 있다.

주소 표현의 문제는 [13, 14, 15]에서의 키값 계산을 위해 이미지의 각각의 속성에서 계산된 순위들을 모두 곱하여 하나의 키로 구하였기 때문에 발생함으로 이것은 각각의 특성에서 나온 순위를 곱하지 않고 개별적인 키로 삼는다면 해결될 수 있다. 그리고 저장 공간의 오버헤드는 주소에 해당하는 이미지가 있을때만 인덱스 공간을 마련한다면 해결될 수 있다. 그러므로 하나의 키를 가지는 해쉬 테이블을 이용한 방법은 위의 문제점을 가지게 된다. 따라서 본 논문에서는 보다 큰 주소 공간을 표현할 수 있게 하기 위해서 각각의 오브젝트 속성에서 나온 순위를 곱하지 않고 각각을 키로 삼는 다중키 인덱스 방법인 kd-tree를 사용한다. 즉 오브젝트의 속성 스트링에서 계산된 순위가 각각의 키가 된다. 또한 그 키값에 해당하는 이미지가 있어야만 인덱스 공간이 확보되기 때문에 quatitization 값 q에 의해 변하게 되는 상황을 다룰 수 있게 된다.

그러나 kd-tree도 속성의 새로운 범주에 속하는 오브젝트가 들어오면 [13, 14, 15]에서의 속성 순위 계산 방법을 따르기 때문에 데이타베이스내에 있는 이미지에 대한 키값이 달라지게 된다. 그렇지만 [13, 14, 15]방법은 주소공간의 크기와 키값 계산의 값이 직접적으로 해쉬 테이블의 구조에 영향을 미치기 때문에 해쉬 테이블이 다시 구축되어야 하지만 kd-tree인 경우에 변수들이 kd-tree의 구조에 영향을 전혀 미치지 않으므로(단지 들어올 데이타의 갯수에만 의존) 처음부터 가능한 속성의 범주의 수(q)를 크게 잡고 키값 계산을 해 준다면 아무런 문제가 없다.

kd-tree[1]는 보조키와 기본키를 구별하지 않고 대칭적으로 취급하여 같은 수준의 검색을 요구하는 경우 여러개의 키를 사용하여 데이타의 저장과 조작을 하기 위한 효과적인 다중키 검색기법이다 [12]. kd-tree는 높이 균형 트리(height-balanced tree)가 아니고 이진 트리(binary tree)이므로 구성면에서 간단하다. 인덱싱을 하기위해 사용되는 속성들에서 산출된 각각의 순위가 키가 되며 삽입되는 데이타는 각각의 키들과 이미지에 대한 정보로 구성되어진다. kd-tree의 각 노드에서는 탐색공간을 두개의 부공간 HISON과 LOSON으로 나누기 위해 키들 중 하나를 선택한다. 두 부공간 HISON과 LOSON은 거의 같은 수의 데이타를 가진다. kd-tree는 들어오는 데이타의 키 중 탐색되는 노드의 LOSON과 HI-SON을 나누는 키의 값(discriminator value)을 비교하여 데이타가 저장될 부공간을 선택한다.

본 논문에서 사용한 kd-tree의 중간 노드와 리프 노드의 구조는 다음과 같고 (그림 3)은 kd-tree의 구조를 보이고 있다.

■ 중간노드의 구조

(*disc, loson-ptr, disc-value, hison-ptr*)

- *disc* loson과 hison을 나누는 키
- *loson-ptr* loson을 가리키는 포인터
- *disc-value* loson과 hison을 나누는 키의 값
- *hison-ptr* loson을 가리키는 포인터

■ 리프노드의 구조

($k_1, k_2, \cdots, k_n, data\text{-}ptr$)

- $k_1, k_2, \cdots, k_n$ 이미지 속성들에 대한 각각의 키값
- *data-ptr* 해당하는 키값을 가지는 이미지에 대한 정보를 가지고 있는 페이지에 대한 포인터

본 논문에서는 메모리 낭비와 입출력 시간을 줄이기 위하여 Cesarini와 Soda가 제시한 페이징 기법[2, 12]을 사용하였다. 이 기법은 B-tree의 구축과 같이 bottom-up방식으로 구성되고 트리가 부분적으로 균형 트리가 되도록 하여 선형 리스트로 되는 것을 방지한다. 한 I/O 페이지안에 노드를 삽입하여 트리를 구축하는데, 페이지가 꽉차면 그 페이지를 분할하여 이 페이지내의 루트 노드를 부모 페이지에 삽입하고 루트 이외의 나머지 노드들을 두 페이지로 나누어 구성하는 기법이다. 한 페이지안의 키들의 이진트리 구조는 대수적 탐색 시간(logarithmic search time)을 유지하고 빠른 데이타 갱신을 허용한다. 페이징 기법을 사용한 이진트리의 저장 공간 구조는 (그림 4)와 같다.

본 논문에서의 kd-tree 구조는 세가지 페이지 구조를 갖는다. 중간노드로 이루어진 중간노드 페이지, 리프노드로 이루어진 리프노드 페이지와 이미지의 물리적 주소와 정보를 가지고 있는 데이타 페이지가 있다. 여기서 Cesarini와 Soda의 페이징 기법은 중간 노드로 이루어진 페이지에 적용된다. 다음은 중간노드 페이지, 리프노드 페이지와 데이타 페이지 구조의 정의이다.

■ 중간노드 페이지

```
typedef struct {
        int                 num_internal;
        INTERNAL_NODE internal_node[num_internal];
INTER_PAGE;
```

■ 리프노드 페이지

```
typedef struct {
        int             num_leaf;
        LEAF_NODE   leaf_node[num_leaf];
} LEAF_PAGE;
```

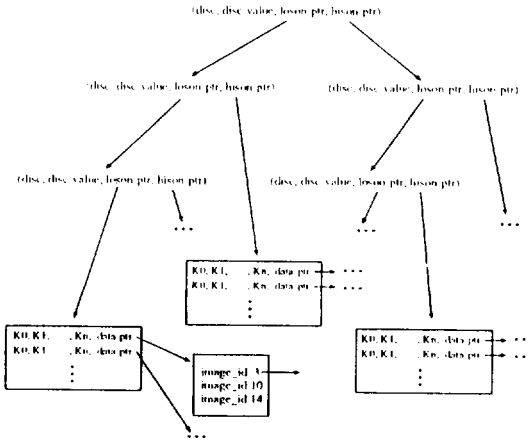
■ 데이타 페이지

```
typedef struct {
        int             image_id[info_num];
} DATA_PAGE;
```

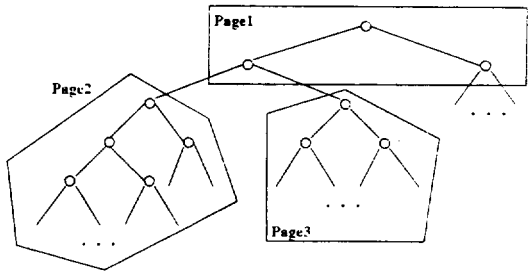
num internal과 num leaf는 한 페이지 안에 저장된 중간노드와 리프노드의 수이다. 정수는 4byte이므로 중간 노드는 총 13byte를 필요로 하고, 리프 노드는 $(v+1)\times 4$byte를 필요로 한다. v는 인덱싱에 사용되는 속성 스트링의 수이다. 예를 들면 한 I/O 페이지의 크기가 1024(1K)byte일 경우에 중간 노드는 78개까지, 리프노드는 $\left(\frac{1024-4}{(v+1)\times 4}\right)$ 까지 들어갈 수 있다. 데이타 페이지의 구조는 [3]에서와 같다.

3.3 정규화 과정

이미지의 scale과 translation 정도에 따라서 같은 이미지라도 다른 이미지 표현이 생성될 수 있는



(그림 3) kd-tree의 구조
(Fig. 3) kd-tree structure



(그림 4) 페이징 기법을 사용한 이진트리의 저장구조
(Fig. 4) Binary tree storage structure using paging method

문제가 있다. 따라서 정규화 과정이 필요하다. 본 논문의 정규화 과정은 다음과 같다. 이미지안에 있는 모든 오브젝트들을 포함하는 최소 사각형 MBR(Minimum Bounding Rectangle)을 구하고 이 MBR의 왼쪽 하단좌표를 (x_1, y_1), 오른쪽 상단좌표를 (x_2, y_2)라 하면 식 (2. 2)는 다음과 같이 수정된다.

$$r[i]=Scg_x[i]\cdot M+Scg_y[i]$$

$$Scg_x[i]=\left(\frac{cg_x[i]-x_1}{x_2-x_1}\cdot N\right),\ Scg_y[i]=\left(\frac{cg_y[i]-y_1}{y_2-y_1}\cdot M\right)$$

또한 몇몇 속성 스트링은 정규화 되어야 하는데 그 속성에는 오브젝트의 크기, 주변길이(perimeter)등이 해당된다. 식 (2.3)에서의 최대 속성값을 해당하는 이미지에 포함된 오브젝트가 그 속성에 대해 가질 수 있는 최대값으로 한다.

$$u=\left(\frac{\text{해당하는 오브젝트의 속성값}}{\text{고려되는 속성의 이미지내 오브젝트의 최대값}}\times q\right)$$

4. 실험결과 및 분석

본 장에서는 시뮬레이트된 이미지로 구성된 IDB를 사용하여 기존의 인덱스 화일구조와 본 논문에서 제시하는 화일구조를 구현하여 성능을 비교 분석하였다. 프로그램은 SUN SPARC 10(Model 41)에서 C언어로 구현하였다.

4.1절에서는 실험의 성능 평가의 기준과 실험에 사용된 여러 변수들에 대해 설명을 한다. 4.2절에서는 실험에 사용되는 데이타를 생성하기 위한 데이타 생성기에 대해서, 4.3절에서는 실제 이미지로부터 키값을 추출한 결과를 보이고 4.4절에서는 해쉬를 이용한 화일구조와 kd-tree를 이용한 화일구조의 각 변수들에 대한 실험결과를 보여주고 실험결과를 비교분석 하였다.

4.1 성능 평가의 기준 및 실험 변수의 선택

실험의 성능 평가 기준은 각 구조가 필요로 하는 저장공간이다. 따라서 해쉬구조와 kd-tree구조의 성능을 평가하기 위하여 다음과 같은 변수들을 선택하여 실험하였다.

- 이미지 데이타 갯수 : N
- 페이지의 크기 : 실험에서 사용된 페이지의 크기를 P라 한다.
- 이미지 부분집합의 크기 : IDB는 $2 \le k \le K_{max}$인 k크기의 오브젝트들을 가지는 화일들로 구성된다.
- quantization 값 : 각 오브젝트들의 속성은 quantization 값인 q개의 범주로 나누어지고여기에 속하게 된다.
- rectangular grid의 크기 : $R \times R$

4.2 데이타 생성기 구현

이미지를 이미지 오브젝트들로 세그멘트하였다고 가정한다. 키가 되는 오브젝트들의 속성은 오브젝트들의 공간관계와 오브젝트가 속하는 클래스 두개로 삼았다. 대부분의 응용에서 이미지에 포함되는 오브젝트들 수의 평균은 10보다 작으므로 [14] random number generator는 2 - 10개의 오브젝트를 포함하고 있는 이미지에 해당하는 2-D 스트링을 생성한다. 인덱싱을 하는데 이차 순서기준은 속성 속성들이 추가되도록 확장될 수 있고 일차 순서기준보다 더 정확하게 이미지를 표현할 수 있기 때문에 본 논문에서는 순서기준으로 이차 순서기준을 사용하였다.

4.3 실제 이미지로부터 키값 추출 결과의 예

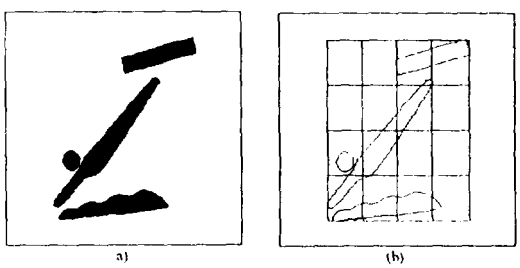
실험에 사용한 이미지는 binary 이미지이다. 순서기준으로 이차 순서기준을 사용하였으며 rectangular grid의 크기는 4×4이다. (그림 5)에서 (a)는 입력된 실제 이미지의 한 예이고 (b)는 입력된 이미지를 세그먼트하여 오브젝트들을 추출한 후 정규화 과정을 위해 MBR을 구하고 4×4 grid로 나눈 것이다. 2-D 스트링이 얻어지고, 추출된 오브젝트들로 구성된 순서화된 이미지 부분집합들의 속성 스트링들이 정규화 과정을 거쳐 얻어진다. 그런 다음 키값이 이들 스트링들로부터 계산된다. 고려한 속성 스트링은 공간관계의 스트링, 즉 랭크 스트링(r)과 크기 스트링(s), 방향 스트링(o), 그리고 주변길이 스트링(p)이다. 〈표 2〉는 그 결과이다.

4.4 실험결과 및 분석

이미지 데이타 갯수(N), 페이지 크기(P), qu-

antization 값(q), 그리고 rectangular grid의 크기(R×R)를 변화시키면서 각 구조의 실험결과를 보여주겠다. 〈표 3, 4, 5〉는 실험을 통해 얻어진 각 구조의 저장공간의 크기를 보여주고 있다. 표에서 "이미지 갯수"는 인덱스되는 이미지의 갯수이고 부분집합의 갯수는 이들 이미지로부터 생성된 $2 \le k \le K_{max}$인 k 크기의 부분집합의 집합에 속하는 부분집합의 갯수이다. 'Data'는 생성된 페이지 중에서 데이타 페이지의 수, 'L'은 데이타 페이지의 연결 리스트의 최대 길이(depth)이고 '페이지'는 해쉬 구조와 kd-tree 구조를 사용하여 IDB를 구축했을 때 생성되는 총 페이지 수이다. 'Ndx'는 인덱스 페이지 수, 'D'는 검색시 데이타 페이지에 접근할 때의 페이지 접근 횟수의 최대값(depth)을 나타낸다. 해쉬구조와 kd-tree구조에서 생성된 데이타 페이지의 수(Data)와 데이타 페이지의 연결 리스트의 최대 길이(L)는 같다.

〈표 3〉은 P=1024, R×R=3×3, q=3, IDB가 2≤K≤6까지의 크기를 갖는 이미지 부분집합들의 화일로 구성될 때, 이미지 데이타 갯수 N의 변화에 따른 총 저장공간의 크기와 페이지 접근횟수이다. 〈표 3〉의 변수들의 값은 작은 주소공간을 만드는 값으로 여기서 해쉬 구조를 사용했을 경우 적은 저장공간을 가질 경우이다. 이미지 데이타의 갯수가 10000개가 들어올 때까지 총 인덱스 페이지의 수(Ndx)를 비교해보면 해쉬구조가 kd-tree보다 더 많은 저장공간을 필요로 함을 볼 수 있다. kd-tree의 인덱스 페이지(Ndx)의 수가 커지는 경우는 주소공간의 대부분의 키를 가지도록 많은 이미지들이 들어올 경우로써 주소공간이 큰 경우에는 이런 경우가 일어나기는 어려울 것이다. 또한, 특정한 응용에서는 비슷한 이미지가 많이 들어와서 같은 키를 가지는 이미지가 많을 것이라 예상할 수 있다. 즉 주소공간이 클수록 kd-tree를 사용했을 경우가 해쉬구조를 이용했을 경우보다 효과가 점점 커지며 kd-tree가 더 많은 저장공간을 가지기 위해서는 대단히 많은 이마지가 들어와야 함을 알 수 있다. 그러므로 가능한 주소 공간에서 모든 주소공



〈그림 5〉 실제 이미지로부터 키값 추출
(a)입력 이미지 (b) 오브젝트 추출결과
(Fig. 5) Key value extraction from raw image
(a) input image (b) object extraction result

〈표 2〉 이미지의 키값 추출 결과
〈table 2〉 Key value extraction result from the image

이미지 부분집합	순서화된 이미지 부분집합	랭크 스트링(r)	키	크기 스트링(s)	키	orient 스트링(o)	키	주변길이 스트링(p)	키
{0,1}	{1,0}	(1,4)	11	(0,2)	2	(3,0)	12	(0,3)	3
{0,2}	{0,2}	(4,5)	19	(2,3)	11	(0,1)	1	(3,3)	15
{0,3}	{0,3}	(4,11)	70	(2,2)	10	(0,0)	0	(3,2)	14
{1,2}	{1,2}	(1,5)	16	(0,3)	3	(3,1)	13	(0,3)	3
{1,3}	{1,3}	(1,11)	67	(0,2)	2	(3,0)	12	(0,2)	2
{2,3}	{2,3}	(5,11)	71	(3,2)	14	(1,0)	4	(3,2)	14
{0,1,2}	{1,0,2}	(1,4,5)	46	(0,2,3)	11	(3,0,1)	49	(0,3,3)	15
{0,1,3}	{1,0,3}	(1,4,11)	297	(0,2,2)	10	(3,0,0)	48	(0,3,2)	14
{0,2,3}	{0,2,3}	(4,5,11)	305	(2,3,2)	46	(0,1,0)	4	(3,3,2)	62
{1,2,3}	{1,2,3}	(1,5,11)	302	(0,3,2)	14	(3,1,0)	52	(0,3,2)	14
{0,1,2,3}	{1,0,2,3}	(1,4,5,11)	1047	(0,2,3,2)	46	(3,0,1,0)	196	(0,3,3,2)	62

〈표 3〉 P=1024, 3×3 grid, q=3일때 페이지 생성수와 페이지 접근 횟수
〈table 3〉 Number of pages generated and page accesses when P=1024, 3×3 grid, and q=3

이미지 갯수	부분집합의 갯수		Data	L	해쉬 페이지	해쉬 Ndx	해쉬 D	Kd-tree 페이지	Kd-tree Ndx	Kd-tree D
2000	Subset2	37623	405	1	407	2	1	413	8	2
	Subset3	75840	4445	1	4463	18	1	4526	81	2
	Subset4	106825	6214	1	6371	157	1	6323	109	3
	Subset5	107338	62712	1	63934	1222	1	63800	1088	3
	Subset6	76928	53132	1	61684	8552	1	54045	913	3
	total	404554	126908		136859	9951		129107	2199	
4000	Subset2	72618	405	1	407	2	1	413	8	2
	Subset3	145001	4455	1	4473	18	1	4532	77	2
	Subset4	202946	6496	1	6653	157	1	6611	115	3
	Subset5	203011	106112	1	107334	1222	1	107954	1842	3
	Subset6	145092	99910	1	108462	8552	1	101627	1717	3
	total	768668	217378		227329	9951		221137	3759	
6000	Subset2	110370	472	2	474	2	1	481	9	2
	Subset3	220840	4455	1	4473	18	1	4532	77	2
	Subset4	308878	6666	1	6823	157	1	6787	121	3
	Subset5	308320	143921	1	145143	1222	1	146442	2521	3
	Subset6	219795	149239	1	157791	8552	1	151822	2583	3
	total	1168203	304753		314704	9951		310064	5311	
8000	Subset2	145004	729	2	731	2	1	738	9	2
	Subset3	288928	4455	1	4473	18	1	4530	75	2
	Subset4	403631	6859	1	7016	157	1	6979	120	3
	Subset5	403205	168594	1	169816	1222	1	171528	2934	3
	Subset6	287905	191116	1	199668	8552	1	194433	3317	3
	total	1528673	371753		381704	9951		378208	6455	
10000	Subset2	184295	729	2	731	2	1	738	9	2
	Subset3	369682	4455	1	4473	18	1	4539	84	3
	Subset4	518298	7062	1	7219	157	1	7188	126	3
	Subset5	518227	193614	1	194836	1222	1	197008	3394	3
	Subset6	369662	239904	1	248456	8552	1	244078	4174	3
	total	1960164	445764		455715	9951		453551	7787	

간을 확보한 해쉬보다 주소에 해당하는 키값을 가진 이미지가 들어올 경우에만 인덱스가 생성되도록 하는 구조인 kd-tree를 구축하는데 필요한 페이지 수는 적다고 할 수 있다.

〈표 4〉와 〈표 5〉는 P=1024, 3×3 rectangular grid상에서 각각 q=4, q=5일 때의 이미지 갯수 N의 변화에 따른 총 저장공간의 크기와 페이지 접근횟수이다. 〈표 3, 4, 5〉를 보면 해쉬구조와 kd-tree의 인덱스 페이지 수가 점점 더 큰 차이가 남을 알 수 있다. 이미지 데이타 갯수(N)이 10000일 경우 차이는 q=3일 때 약 21%, q=4일 때 약 79%, q=5일 때 약 93%이다. (그림 6)은 〈표 3, 4, 5〉의 인덱스 페이지 수의 비교이다. 즉, q가 3에서 5로 변화하였을 경우의 kd-tree와 해쉬 구조의 인덱스 페이지 수의 비교이다.

(그림 7)은 P=1024, 4×4 rectangular grid, Q=3일 때 이미지 데이타가 50000개까지 들어왔을 경우의 인덱스 페이지 수의 비교이다. 증가폭으로 보아 인덱스 페이지의 수가 더 커지려면 많은 이미지가 들어와야 한다.

해쉬 구조의 장점은 주어진 키를 기반으로 매우 빠른 직접 접근(direct access)을 제공한다는 것이다. 해쉬 구조에서 처음의 데이타 페이지에 접근할 때의 페이지 접근 횟수의 최대값(D)은 1이다. 그러나 실험에서 kd-tree의 데이타 페이지 접근 횟수의 최대값(D)은 2-4임을 보이지만, 페이징 기법으로 한 페이지안의 키들의 kd-tree 구조는 빠른 데이타 검색을 허용함으로 검색 성능에서 크게 차이가 나지 않으리라 예상된다. 그리고 페이지의 크기가 커지면 데이타 페이지 검색 횟수의 차이는 더욱 줄어들 것이다. 주소공간의 크기는 k, 인덱싱하는데 필요한 속성 수, quantization 값 q, rectangular grid의 크기가 커질수록 증가한다. 반응시간(response time)은 질의에 대한 응답으로 검색되어지는 이미지 집합과 밀접한 관계가 있으며 검색되는 이미지 집합의 크기가 커질수록 증가한다[14]. 따라서 한 인덱스에 함께 저장되는 이미지 집합의 크기가 적을수록 반응시간은 줄어든다. 검

〈표 4〉 P=1024, 3×3 grid, q=4일때 페이지 생성수와 페이지 접근 횟수
〈table 4〉 Number of pages generated and page accesses when P=1024, 3×3 grid, and q=4

이미지 갯수(N)	부분집합의 갯수		Data	L	해쉬			Kd-tree		
					페이지	Ndx	D	페이지	Ndx	D
2000	Subset2	35945	720	1	723	3	1	733	13	2
	Subset3	71459	10312	1	10351	42	1	10494	182	3
	Subset4	99358	17041	1	17536	495	1	17343	301	3
	Subset5	98413	72537	1	77685	5148	1	73804	1267	3
	Subset6	69403	54561	1	102609	48048	1	55523	962	3
	total	374578	155171		208907	53736		157897	2725	
4000	Subset2	73004	720	1	723	3	1	734	14	2
	Subset3	145066	10519	1	10561	42	1	10695	176	3
	Subset4	201152	19077	1	19572	495	1	19426	348	3
	Subset5	199597	140641	1	145789	5148	1	143080	2439	3
	Subset6	141166	109533	1	157581	48048	1	111454	1921	3
	total	760285	280490		334226	53736		285389	4898	
6000	Subset2	109855	720	1	723	3	1	733	13	2
	Subset3	219323	10553	1	10595	42	1	10736	183	3
	Subset4	306356	19618	1	20113	495	1	19961	343	3
	Subset5	305643	206534	1	211682	5148	1	210121	3587	3
	Subset6	217887	167769	1	215817	48048	1	170682	2913	3
	total	1159064	405194		458930	53736		412233	7039	
8000	Subset2	143727	720	1	723	3	1	734	14	2
	Subset3	285277	10557	1	10599	42	1	10741	184	3
	Subset4	396802	19887	1	20382	495	1	20244	357	3
	Subset5	394607	259560	1	264708	5148	1	264108	4548	3
	Subset6	280602	507462	1	264786	48048	1	220486	3748	3
	total	1501015	507462		561198	53736		516318	8851	
10000	Subset2	183921	816	2	819	3	1	830	14	2
	Subset3	368789	10559	1	10601	42	1	10743	184	3
	Subset4	517345	20517	1	21012	495	1	20511	354	3
	Subset5	518020	326641	1	331789	5148	1	332282	5641	4
	Subset6	370276	284671	1	332719	48048	1	289584	4913	4
	total	1958351	643204		696940	53736		653950	11106	

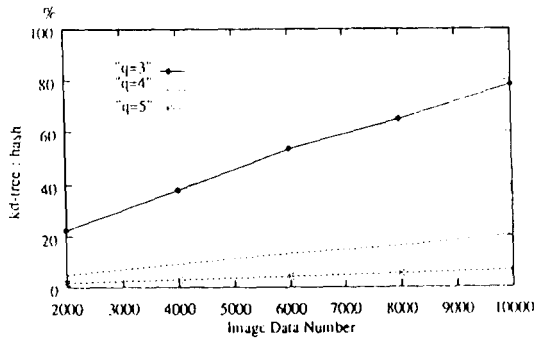
〈표 5〉 P=1024, 3×3 grid, q=5일때 페이지 생성수와 페이지 접근 횟수
〈table 5〉 Number of pages generated and page accesses when P=1024, 3×3 grid, and q=5

이미지 갯수(N)	부분집합의 갯수		Data	L	해쉬			Kd-tree		
					페이지	Ndx	D	페이지	Ndx	D
2000	Subset2	35450	1125	1	1130	5	1	1147	22	2
	Subset3	70036	18598	1	18679	81	1	18925	327	3
	Subset4	97117	31769	1	32978	1209	1	32356	578	3
	Subset5	96338	76949	1	92660	15711	1	78293	1344	3
	Subset6	68306	56463	1	239752	183289	1	57444	981	3
	total	367247	184904		385199	200295		188165	3232	
4000	Subset2	74891	1125	1	1130	5	1	1147	22	2
	Subset3	150505	20232	1	20313	81	1	20575	343	3
	Subset4	211464	41523	1	42732	1209	1	42286	754	3
	Subset5	212152	165902	1	181613	15711	1	168772	2870	3
	Subset6	151971	125032	1	308321	183289	1	127205	2173	3
	total	800983	353814		554109	200295		359985	6162	
6000	Subset2	109806	1125	1	1130	5	1	1145	20	2
	Subset3	218210	20498	1	20579	81	1	20850	352	3
	Subset4	302824	44218	1	45427	1209	1	45016	783	3
	Subset5	299951	233049	1	248760	15711	1	237094	4045	3
	Subset6	212275	175021	1	358310	183289	1	178056	3035	3
	total	1143066	473911		674206	200295		482161	8235	
8000	Subset2	147604	1125	1	1130	5	1	1146	21	2
	Subset3	296614	20565	1	20646	81	1	20919	354	3
	Subset4	417612	45957	1	47166	1209	1	46824	845	3
	Subset5	420283	320439	1	336150	15711	1	326037	5598	4
	Subset6	302331	248043	1	431332	183289	1	252323	4280	4
	total	1584444	636129		836424	200295		647249	11098	
10000	Subset2	183880	1125	1	1130	5	1	1147	22	2
	Subset3	367844	20609	1	20690	81	1	20962	353	3
	Subset4	514455	46862	1	48071	1209	1	47727	851	3
	Subset5	513540	387165	1	402876	15711	1	393881	6716	4
	Subset6	366100	301149	1	484438	183289	1	306374	5225	4
	total	1945819	756910		957205	200295		770091	13167	

색되는 이미지 집합의 크기는 주소공간의 크기 D_k에 반비례하여 D_k가 커짐에 따라 저장되는 이미지 집합은 더 큰 주소공간에 분포할 수 있음으로 인덱스당 저장되는 이미지의 집합의 크기는 감소한다. 그러므로 k, 인덱싱하는데 필요한 속성 수, q, rectangular grid의 크기가 커질수록 이미지 표현의 정확성을 높일 뿐 아니라 검색의 성능도 좋아진다. 해쉬구조는 주소공간에 영향을 미치는 이들 변수에 민감하여 이들의 크기가 커질수록 지수적인 저장공간의 증가를 보이나 kd-tree는 단지 이미지 데이타의 갯수와 키값의 분포에만 영향을 받는다.

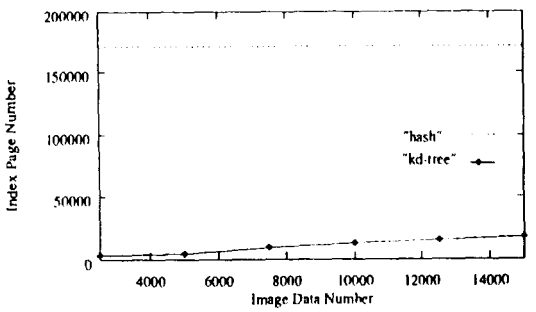
5. 결 론

본 논문은 내용에 기반한 이미지 인덱싱에 관한



(그림 6) P=1024, 3×3 grid에서 q의 변화에 따른 인덱스 페이지 수의 비교

(Fig. 6) Comparison of the number of index pages changing q when P=1023, 3×3 grid



(그림 7) P=1024, 4×4 grid, q=3일 때의 인덱스 페이지 수

(Fig. 7) Number of index pages when P=1024, 4×4 grid and q=3

연구이다. 내용에 기반한 이미지 검색은 이미지 데이타베이스 시스템의 기본 요소로 되어오고 있고 점점 중요성이 커지고 있다. 내용에 기반한 이미지 검색을 위해서, 이미지의 내용을 간결하게 표현하고 효율적으로 처리될 수 있는 구조로 알려진 2-D 스트링에 기반한 인덱싱 연구가 있어왔다. 그 중에서 Petrakis[14]의 방법은 기존의 2-D 스트링에 기반한 인덱싱 방법을 확장하고, 검색 대상이 될 수 있는 여러 오브젝트를 가지는 이미지에도 적용될 수 있으며 오브젝트들의 공간관계와 오브젝트들의 속성을 고려한 효과적인 인덱싱 방법이다. 그렇지만 Petrakis의 방법은 저장공간을 많이 필요로 하며, 새로운 종류의 이미지가 삽입되는 환경에서는 적합하지 않다. 이러한 문제점은 오브젝트들의 각각의 속성들로부터 나온 순위들을 모두 곱하여 하나의 키로 만들고 인덱스 화일구조로 해쉬 구조를 사용함으로써 발생한 것이다.

본 논문은, 각각의 속성들로부터 나온 키값들을 개별적인 키로 하여, 키의 값에 따라 저장공간이 확보되므로 적은 저장공간을 필요로하고,융통성과 확장성이 있는, 동적 다중키 인덱스 구조인 페이징된 kd-tree의 사용을 제안하였다. 그리고 이미지의 scale과 translation 정도에 영향이 없이 검색될 수 있도록 정규화 과정을 추가하였고, 실제 이미지로부터 각 스트링들과 그것들의 순위, 그리고 키값이 추출되는 과정을 보였다. 그리고 제안된 인덱스 화일구조와 기존의 인덱스 화일구조와의 성능을 시뮬레이트된 이미지 데이타베이스를 이용한 실험을 통해서 비교하였다. 이미지 부분집합의 크기(k), 인덱싱하는데 필요한 속성 수, 오브젝트들의 속성의 범주(q), rectangular grid의 크기가 커질수록 저장공간의 크기는 증가하지만 이미지 표현의 정확성을 높일 뿐 아니라 검색의 성능도 좋아진다. 실험결과로, 본 논문에서 제시한 방법이 융통성, 확장성, 그리고 저장공간면에서 더 효과적임을 알 수 있다.

참 고 문 헌

[1] J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," Comm. of the ACM, Vol. 18, No. 9,

pp. 509-517, 1975.

[2] F. Cesarini and G. Soda, "Binary Trees Paging," Information Systems, 7(4), pp. 337-344, 1982.

[3] Chin-Chen Chang and Suh-Yin Lee, "Retrieval of Similar Pictures on Pictorial Databases," Pattern Recognition, 24(7) : 675-680, 1991.

[4] Shi-Kuo Chang, Qing-Yun Shi, and Cheng-Wen Yan, "Iconic Indexing by 2-D Strings," IEEE Trans. on Pattern Analysis and Machine Intelligence, 9(3) : 413-428, May, 1987.

[5] Shi-Kuo Chang, C. W. Yan, T. Arndt, and D. Dimitroff, "An Intelligent Image Database System," IEEE Trans. Software Eng., 14(5) pp.681-688, May, 1988.

[6] Shi-Kuo Chang and Arding Hsu, "Image Information Systems : Where Do We Go FromWhere?," IEEE Transactions on Knowledge and Data Engineering, 4(5) : 431-442, 1992.

[7] C. R. Cook and R.Oldehoeft, "A Letter-Oriented Minimal Perfect Hashing Function," ACM SIGPlan Notices, 17, pp.18-27, September, 1982.

[8] H.V. Jagadish, "A Retrieval Technique for Similar Shapes," Proceedings of the ACM SIGMOD International Conference on the Management of Data, pp. 208-217, Denver Colorado, May 1991.

[9] Suh-Yin Lee, Man-Kwan Shan, and Wei-Pang Yang, "Similarity Retrieval of Iconic Image Databases," Pattern Recognition, 22(6) : 675-682, 1989.

[10] Suh-Yin Lee and Fang-Jung Hsu,"Spatial Reasoning and Similarity Retrieval of Images using 2D C-String Knowledge Representation," Pattern Recognition, 25 (3) : 305-318, 1992.

[11] Rajiv Mehrotra and James E. Gary, "Feature-Based Retrieval of Similar Shapes", 9th International Conference on Data Engineering, pp.108-115, 1993.

[12] B. C. Ooi, Efficient Query Processing in Geographic Information Systems, Springer-Verlag, Berlin, 1990.

[13] Euripides G.M. Petrakis, "Image Representation, Indexing and Retrieval based on Spatial Relationships and Properties of Objects," Technical Report No. 75, ICS-FORTH, Heraklion, Crete, Greece, March 1993.

[14] Euripides G.M. Petrakis and Stelios C. Orphanoudakis, "Methodology for the Representation, Indexing and Retrieval of Images by Content," Image and Vision Computing, 11(8) : 504-521, October, 1993.

[15] Euripides G.M. Petrakis and Stelios C. Orphanoudakis, "A Generalized Approachfor Image Indexing and Retrieval Based on 2-D Strings," Technical Report No. 103, ICS-FORTH, Heraklio, Crete, Greece, December, 1993.

[16] S.S.Iyengar and R.L.Kashyap,"Guest Editors' Introduction to Image Databases," IEEE Transactions on Software Engineering, Vol. 14. No. 5, May, 1988.

[17] Hideyuki Tamura and Naokazu Yokoya, "Image Database Systems : A Survey.," Pattern Recognition, 17(1) : 29-49, 1984.

[18] C. Thonas, Multimedia Office Filing : The MULTOS Approach, Elsevier Science Publishers B.V., North-Holland, 1990

[19] V.Gudivana and V.Raghavan, "Content-Based Image Retrieval Systems," Computer, vol.28, no.9, pp.18-22,September,1995/

[20] 최기호 외 4인, "내용을 기반으로 한 이미지 검색 데이타베이스 시스템," 정보과학회지, 제 13 권, 제 1호, pp.8-17, 1995년 1 월

유 원 경

1979년 서울대학교 계산통계학과 졸업 (이학사)
1981년 서울대학교 계산통계학과 대학원 전산학 전공(이학석사)
1987년 서울대학교 계산통계학과 대학원 전산학 전공(이학박사)
1981년~86년 한남대학교 전산학과 전임강사, 조교수
1986년~현재 성신여대 전자계산학과 교수
관심분야 : 데이타베이스, 컴퓨터 시스템 분석

정 을 윤

1993년 성신여자 대학교 전자계산학과 졸업(이학사)
1995년 성신여자 대학교 전자계산학과 대학원(이학석사)
1995년~현재 NDS 경영지원실 근무
관심분야 : 데이타베이스, 이미지 프로세싱