# 개선된 간단한 경계선 추적자 알고리즘
## (Improved Simple Boundary Following Algorithm)

정 철 호 † 한 탁 돈 ††
(Cheolho Cheong) (Tack-Don Han)

**요 약** 간단한 경계선 추적자 알고리즘(SBF: Simple Boundary Follower)은 영상의 인식과 표현을 위하여 사용되는 경계선 추적 알고리즘 중에서 널리 사용되는 것이다. SBF는 주로 이진화된 영상에서 물체의 경계선을 추적할 때 매우 간단하면서도 효율적인 성능을 제공한다. 그러나 추적자의 위치와 방향조건에 따라 인너코너(inner corner)와 인너-아우터 코너(inner-outer corner)의 일부 영상을 추적하지 못하는 비일관성(inconsistency)을 가지고 있고, 경계선 이외의 픽셀들에 대한 부가적인 이동연산을 요구한다는 단점이 있다. 이러한 제약성들 중 인너-아우터 코너의 비일관성을 해결하기 위하여 수정된 간단한 경계선 추적자 알고리즘(MSBF: Modified Simple Boundary Follower)이 연구되었지만 여전히 인너코너 추적 비일관성과 부가 연산 필요성에 대한 해결책은 제시되지 않았다. 본 논문에서는 이를 개선하기 위하여 비일관성이 일어나지 않도록 처리하고, 경계선 주위 픽셀에 대한 이동 연산을 줄일 수 있는 개선된 간단한 경계선 추적자 알고리즘(ISBF: Improved Simple Boundary Follower)을 제안하였다. 또한 경계선상의 픽셀들의 배치와 이동 경로 방식에 대해 분류를 함으로써 세 가지 알고리즘의 연산복잡도를 계산하였다. 제안된 알고리즘은 MSBF에 비하여 연산량이 감소하였고, 비일관성의 문제가 없는 것으로 분석되었다.

**키워드** : 경계선 추적 알고리즘, Papert의 터틀 알고리즘, 간단한 경계선 추적자, 수정된 간단한 경계선 추적자, 개선된 간단한 경계선 추적자

***Abstract*** The SBF (Simple Boundary Follower) is a boundary-following algorithm, and is used mainly for image recognition and presentation. The SBF is very popular because of its simplicity and efficiency in tracing the boundary of an object from an acquired binary image; however, it does have two drawbacks. First, the SBF cannot consistently process inner or inner-outer corners according to the follower's position and direction. Second, the SBF requires movement operations for the non-boundary pixels that are connected to boundary pixels. The MSBF (Modified Simple Boundary Follower) has a diagonal detour step for preventing inner-outer corner inconsistency, but is still inconsistent with inner-corners and still requires extra movement operations on non-boundary pixels. In this paper, we propose the ISBF (Improved Simple Boundary Follower), which solves the inconsistencies and reduces the extra operations. In addition, we have classified the tour maps by paths from a current boundary pixel to the next boundary pixel and have analyzed SBF, MSBF, and ISBF. We have determined that the ISBF has no inconsistency issues and reduces the overall number of operations.

**Key words** : boundary following algorithm, Papert's Turtle procedure, SBF: Simple Boundary Follower, MSBF: Modified Simple Boundary Follower, ISBF: Improved Simple Boundary Follower

## 1. 서 론

경계선 추적 알고리즘(boundary following algorithm)은 영상에서 물체의 형태를 추적하기 위한 매우 유용한 기법이다[1]. 이 기법은 2차원 혹은 3차원 이미지 상의 물체들의 외곽선이나 경계선의 모양을 기술하기 위한 것으로 물체 인식, 이미지 인식, 패턴 인식 등의 영상 인식과 2차원, 3차원 물체를 표현하기 위하여 사용

· 본 연구는 2005년도 한국과학재단 특정기초연구(R01-2005-000-10898-0)의 연구비 지원으로 수행하였습니다.
† 학생회원 : 연세대학교 컴퓨터과학과
bright@kurene.yonsei.ac.kr
†† 종신회원 : 연세대학교 컴퓨터과학과 교수
hantack@kurene.yonsei.ac.kr
논문접수 : 2005년 10월 24일
심사완료 : 2006년 2월 20일

된다. 즉, 이미지상의 물체의 외곽선이나 경계선을 따라 추적함으로써 물체의 특징점이나 모양을 판별할 수도 있고, 물체의 종류와 크기 등을 알아낼 수도 있는 기법이다.

이들 중 간단한 경계선 추적자 알고리즘(SBF: Simple Boundary Follower) 혹은 Papert의 터틀 프로시저(Papert's Turtle Procedure)이라고도 불리는 기법은 영상처리, 특히 영상인식 분야에서 널리 사용되고 있다[2-4]. 이 기법은 영상처리 기법에서 경계선을 추적하거나 물체의 모양을 판별하기 위해 사용되는 일반화된 사용 방법으로, 이미지나 특정한 정보들을 관심영역과 그렇지 않은 영역으로 나누고 이를 토대로 관심 영역을 추출할 때 이용한다[5]. 따라서 SBF는 주로 이진화된 영상을 기반으로 적용되는데, 컬러영상이나 그레이 영상 등에서도 관심 색상이나 밝기 중 특정한 범위의 정보를 관심 영역으로 설정하면 쉽게 적용할 수 있다. 또한 SBF는 물체의 외곽선을 구성하는 픽셀들을 추적할 때 매우 간단한 규칙을 이용하므로 구현이 용이하고 연산량이 적으므로 자원이 제약된 모바일 컴퓨팅 환경이나 임베디드 컴퓨팅 환경에서도 유용하다. 그러나 픽셀의 배치 형태와 추적자의 조건에 따라 코너의 픽셀, 특히 인너코너(inner corner)와 인너-아우터 코너(inner-outer corner)의 픽셀에 대한 추적여부가 달라지는 비일관성이 존재한다. 또한 경계선을 추적하기 위해 필요한 픽셀뿐만 아니라 경계선 주위의 픽셀도 추적하므로 부가적인 연산이 요구된다.

이런 비일관성으로 인해 일부 인너코너나 인너-아우터 코너에 포함되어 있는 물체나 영역은 추적할 수가 없다. 따라서 물체 전체의 외곽선을 모두 탐지하지 못하므로 물체를 인식할 때 매우 큰 문제점이 된다. 특히 근래에는 휴대전화를 통해 2차원 이미지 코드 인식, 얼굴인식, 문자인식 등의 서비스를 제공하고 있는데, 카메라의 성능이 일반적으로 낮아서 빛이나 조명에 대한 영향을 많이 받은 이미지를 얻게 된다. 이러한 이미지들은 스캐너나 고정식 카메라에서 얻은 이미지보다 품질이 많이 저하되며, 이진화시에는 픽셀들의 손실도 상대적으로 크다. 또한 전체적으로 이미지가 밝은 경우에 흔히 발생하는 문제이기도 하다. 따라서 이미지로부터 물체를 인식할 때에 이러한 비일관성의 문제는 매우 중요한 해결과제이다.

SBF에 기반한 수정된 간단한 경계선 추적자 알고리즘(MSBF: Modified Simple Boundary Follower)은 비일관성 문제를 해결하기 위한 방법이다[6]. 이 알고리즘은 인너-아우터 코너에 대한 비일관성 문제를 해결하였으나 인너코너의 픽셀의 비일관성 문제를 가지고 있으며 흰색 픽셀에 대한 부가적인 연산을 필요로 한다.

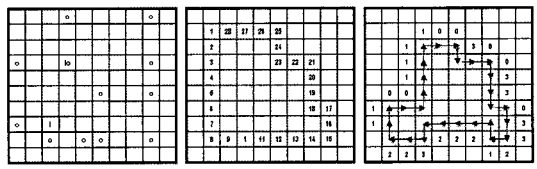
본 논문에서는 이러한 문제점들을 해결하기 위하여 MSBF를 기반으로 한 개선된 간단한 경계선 추적자 알고리즘(ISBF: Improved Simple Boundary Follower)을 제안하고자 한다. ISBF는 경계선상의 픽셀들만을 대상으로 하여 추적을 수행하고, 주변부 픽셀들에 대해서는 단순히 감지연산만 함으로써 연산량을 줄이고자 하였고, 인너코너에 대한 감지가 가능하도록 하였다. 본 논문에서는 먼저 SBF와 MSBF를 소개하고, 개선한 알고리즘의 규칙을 제안한 후 세 가지 알고리즘에 대해 연산량을 분석한 결과를 제시하고자 한다.

## 2. 관련 연구

### 2.1 경계선 추적 알고리즘

경계선 추적 알고리즘은 영상 처리 기술 중 간단하면서 유용한 것으로 영상의 분석과 표현에 사용된다. 영상의 물체의 외곽선이나 경계선을 추적함으로써 물체의 형태, 물체들의 연결성, 물체의 크기나 둘레길이, 물체 간의 관계 등의 정보도 알 수가 있다. 한 예로 그림 1(a)는 물체의 분할 정보를 표현함으로써 물체의 개수를 세기 위한 것이다. 물체의 외곽선을 추적한 후 외부 코너는 'o', 안쪽 코너는 'i', 그리고 두 개가 접한 인너-아우터 코너는 'io'로 표시하였다. 이 때 io를 하나의 물체로 판단할지 아니면 두 개의 물체로 판단할지에 따라 이 영상에는 두 개 혹은 세 개의 물체가 존재한다고 할 수 있다.

그림 1(b)는 물체의 둘레길이를 표현한 것으로 픽셀의 개수는 28이지만, 인너코너의 길이를 대각선으로 계산할 경우 $24+2\sqrt{2}$ = 26.8이다. 그림 1(c)는 프리만(Freeman)의 체인코드기법을 적용한 것으로 외곽선을 추적하면서 나타나는 픽셀들 간의 상대적인 방향을 알 수 있다.



(a) 물체의 분할 (b) 경계선 길이 측정 (c) 체인코드

그림 1 경계선 추적 알고리즘을 이용한 영상처리의 예 ([4,6]의 그림을 일부 수정)

### 2.2 SBF(Simple Boundary Follower)

SBF는 Papert가 1973년에 발표한 것으로 터틀 알고리즘으로도 알려져 있으며, 이진화된 물체의 외곽선을 쉽게 추적할 수 있다[3,7]. 이진화된 영상은 0과 1로 판별하게 되는데, 관심있는 물체의 영역을 검은색으로 하

고 값은 1로 지정하는 것이 통상적이다. 물론 컬러 이미지에서도 관심있는 영상을 1로 지정하고 그렇지 않은 컬러 영상들은 0으로 처리할 수 있다. SBF는 자신과 주위의 4개 인접픽셀(왼쪽, 오른쪽, 위, 아래)을 대상으로 하여 추적을 수행하게 되므로 4-방향 체인코드를 이용하면 쉽게 구현할 수 있다.

2.2.1 규칙

SBF에서 추적자는 자신의 위치 정보와 자신의 방향 정보를 가지며 이를 이용하여 경계선을 추적한다. SBF의 규칙은 다음과 같다.

(1) 조건

- 이미지는 이진화되어 0과 1로 표현되는 픽셀로 표현되며, 관심이 있는 물체의 경계선은 1로 이루어진다.
- 이미지상의 물체는 안이 채워진 도형으로 가정한다.
- 추적자의 시작 위치는 경계선 상의 임의의 한점이며 추적자의 방향성을 적절하게 설정한다.

(2) 프로시저

- Step 1: 추적자는 경계선 상의 한 점에서 임의의 방향성을 가지고 시작한다.
- Step 2: 현재 추적자가 위치한 픽셀 정보가 1이면 좌측 픽셀로 추적자를 이동시키고, 방향성을 현재의 좌향으로 지정한다. 그렇지 않으면 추적자를 현재의 우측 픽셀로 이동하고, 방향성도 현재의 우향으로 바꾼다.
- Step 3: 추적자의 변경된 픽셀 좌표와 방향성이 시작할 때의 좌표 및 방향성과 일치하는 지 검사하여 일치하면 종료하고 그렇지 않으면 step 2를 재수행한다.

그림 2에서는 추적자가 시작점인 S픽셀에서 E방향을 가지고 시작하여 경계선을 따라 이동하다가 시작점으로 다시 돌아오는 모습을 보이고 있다. 시작점이 검은색 픽셀이므로 추적자는 자신의 좌측 방향(N방향)으로 돌아서 인접픽셀로 진행한다. 따라서 추적자는 시작점의 N방향 인접픽셀에서 N방향을 바라보게 된다. 두 번째 이동에서는 현재 추적자의 위치픽셀이 흰색이므로 우측인 E방향의 인접픽셀로 이동한 후 E방향을 가지게 된다. 이런 식으로 경계선을 따라 이동하던 추적자는 시작점에 돌아와서 시작방향이었던 E를 가리키게 되면 추적을 종료한다.

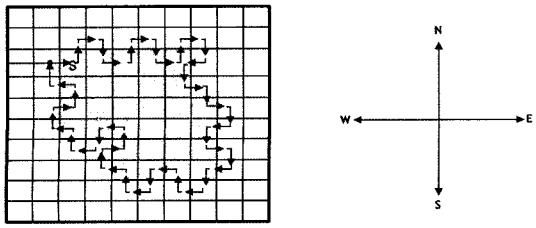
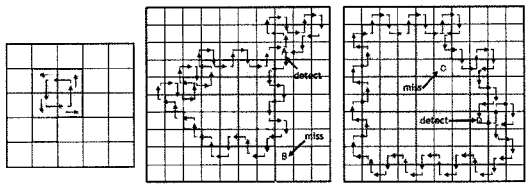



그림 2 SBF기반 물체의 경계선 추적과정의 예

2.2.2 제약점

SBF는 매우 간편해서 많이 사용하지만 맴돌이 현상, 비일관성 문제 그리고 경계선 외부의 부가적인 연산이 필요하다는 제약점을 가지고 있다. 이중 맴돌이 문제는 SBF의 초기 조건에 따라 일어나는 현상으로 시작점과 시작점의 좌측, 좌하측, 하측에 위치한 픽셀들이 모두 검은색일 경우 이 네 픽셀들을 맴돌고 추적이 종료되는 현상이다. 그림 3(a)는 맴돌이 현상의 예를 나타낸 것이다. 맴돌이 문제는 상기 네 픽셀의 값을 판별하여 해당 조건이 발생하지 않는 초기 방향과 위치를 설정함으로써 해결할 수 있으므로 본 논문에서는 분석하지 않았다.

비일관성이란 조건에 따라 경계선 추적 여부가 달라지는 것을 의미한다. 즉 현재 추적자의 위치에 따라 특정한 픽셀을 추적할 수도 있고, 추적할 수 없는 경우도 발생하는 것이다. SBF에는 두 가지 비일관성을 가지고 있는데, 인너-아우터 코너 추적 비일관성과 인너코너 추적 비일관성이 그것이다. 그림 3(b)는 인너-아우터 코너 추적 비일관성의 예제로서 A와 B 모두 인너-아우터 코너 픽셀이지만, A는 추적하는 반면, B는 추적하지 못하는 것을 보이고 있다. 이에 따라 이러한 코너와 연결된 영역이 있다면 이를 인식하지 못하게 된다. 그림 3(c)에서는 인너코너 추적 비일관성을 보이고 있는데 C와 D 모두 인너코너 픽셀이지만, 추적 가능 여부가 달라진다[6].




(a) 맴돌이 현상 (b) 인너-아우터코너 추적 비일관성
(c) 인너코너 추적 비일관성

그림 3 SBF의 제약성 ((b), (c)는 [6]의 자료를 수정함)

SBF의 또 다른 제약점은 경계선뿐만이 아니라 주변부의 픽셀도 추적해야한다는 것이다. 즉 관심이 있는 물체의 외곽선뿐만 아니라 외곽선 바깥 픽셀들도 함께 추적해야 하므로 연산량이 증대한다.

### 2.3 MSBF(Modified Simple Boundary Follower)

MSBF는 SBF의 문제점 중의 하나인 인너-아우터 코너 추적 비일관성을 해결하기 위하여 제안된 것이다. 인너-아우터 코너 중 추적하지 못하는 픽셀은 그림 4(a)와 같이 현재 추적자의 위치를 기준으로 좌하측에 인너-아우터 코너 픽셀이 위치하는 경우이다. 즉, A에 추적자가 N 방향성을 가지고 위치하는 경우, B를 탐색하지 못하게 된다. MSBF에서는 좌하측 대각선 검색 단계를 두어서 인너-아우터 코너를 추적할 수 있도록 하였다[6].

2.3.1 규칙

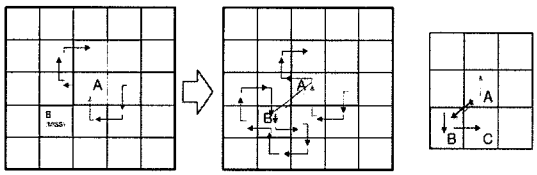
MSBF는 추적자가 검은색 픽셀에 존재하는 경우 좌하측의 대각선 방향에 인너-아우터 코너 픽셀 여부를 감지하고, 만약 존재한다면 이를 처리하는 기법을 추가하였다. MSBF는 대각선 방향의 픽셀 처리가 필요하기 때문에 8방향의 인접픽셀 연산이 필요하다.

(1) 조건

- 기본적으로 SBF의 조건과 같으며, MSBF의 시작시 추적자의 위치는 좌하측 인너-아우터 코너 픽셀을 피해야 한다. 그렇지 않으면 코너 처리 후 다시 시작점으로 돌아오면서 종료된다.

(2) 프로시저

- Step 1: 추적자는 경계선 상의 한 점에서 임의의 방향성을 가지고 시작한다. 그리고 대각선 추적 방지태그(이하 태그)를 0으로 설정한다.
- Step 2: 아래의 Case에 해당하는 부분을 수행한다.
  - Case A: 현재 추적자가 위치한 픽셀값이 1, 좌측과 하측의 픽셀이 0이고, 좌하측의 픽셀이 1, 그리고 태그가 0이면 추적자를 좌하측 픽셀로 이동시키고 방향성을 현재의 반대로 지정한다. 이후 태그를 1로 설정한다(대각선 추적). 이후 Step 3을 수행한다.
  - Case B: Case A가 아니면 현재 추적자의 픽셀값에 따라 SBF의 알고리즘을 따른 후, 태그를 0으로 설정하고 Step 3을 수행한다.
- Step 3: 추적자의 변경된 픽셀 좌표와 방향성이 시작할 때의 좌표 및 방향성과 일치하는 지 검사한 후 일치하면 종료하고 그렇지 않으면 Step 2를 재수행한다.




(a) SBF의 인너-아우터 코너 처리
(b)와 (c) MSBF의 인너-아우터 코너 처리 개념도
그림 4 MSBF의 인너-아우터 코너 처리 기법

그림 4(b)는 MSBF가 인너-아우터 코너 정보를 처리하는 방법을 나타내고 있다. 추적자가 N의 방향성을 가지고 A에 위치하였을 때 인너-아우터 코너 픽셀인 B를 찾은 후, 추적자는 B로 이동하고, 방향성을 본래의 반대 방향인 S로 설정한다. 이렇게 대각선 방향을 처리하는 단계를 Step 2의 Case A와 같이 미리 설정해두면 그림 4(c)의 C 위치로 추적을 계속할 수 있게 된다. 다만, 인너-아우터 코너 처리가 끝난 직후 다시 인너-아우터 처리가 일어날 경우 A와 B사이에서 무한루프가 발생하게 된다. 이를 방지하기 위해서는 현재의 추적자와 전단계의 추적자 정보를 유지하고 인너-아우터 코너 처리 직전 두개의 추적자 정보를 상호 비교하여 서로가 인너-아우터 코너 픽셀이고 반대 방향성을 가지고 있으면, 인너-아우터 코너 처리를 하지 않고 다음 단계를 수행하면 된다. 다른 방법으로는 대각선 추적 방지 태그를 이용하여 무한루프를 방지할 수 있다. 본 논문에서는 연산량이 상대적으로 적은 태그 방식을 제안하여 사용하였으며 상기 프로시저에 제시하였다.

2.3.2 제약점

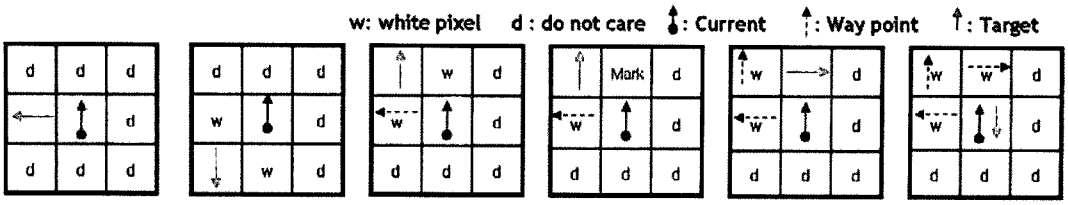
MSBF는 인너-아우터 코너 처리를 용이하게 해결한 기법이지만, SBF의 다른 문제인 인너코너의 비일관성 문제 그리고 경계선 외부의 부가적인 연산이 필요하다는 제약점을 여전히 가지고 있다.

## 3. ISBF(Improved Simple Boundary Follower)

ISBF는 MSBF를 기반으로 비일관성 문제를 해결하고 경계선외의 부가적인 연산량을 줄임으로써 연산 속도를 빠르게 하기 위하여 제안되었다. SBF와 MSBF의 경우 추적자는 물체의 경계선과 그 주위의 픽셀들을 모두 추적하게 된다. 즉, 물체가 검은색이고 주변부가 흰색이라고 한다면, 물체의 경계선 픽셀뿐만 아니라 주위의 흰색 픽셀을 모두 거치는 것이다. 또한, MSBF로는 검은색 픽셀들의 배치 형태에 따른 비일관성을 해결할 수 없다. 제안된 ISBF 알고리즘은 검은색 픽셀에서 다음의 검은색 픽셀까지 경로, 즉 배치 형태를 설정하고 이에 따라 흰색 픽셀에 대해서는 이동이 아니라 단순히 탐지만 한 후 검은색 픽셀로 바로 이동하도록 설계하였다. 또한 검은색 픽셀간의 배치 형태를 분류함에 있어서 인너코너와 인너-아우터 코너를 함께 고려하였다.

### 3.1 물체 경계선 픽셀의 배치 형태 및 이동 경로의 분류

ISBF는 추적자가 경계선상의 픽셀, 즉 검은색 픽셀만을 따라 이동해야 하므로 이에 따라 현재의 검은색 픽셀에서 다음의 검은색 픽셀까지 이동할 수 있는 최단 경로를 구하여야 한다. 그러한 경로의 경우의 수는 모두 6가지이며, 각각의 경로들의 종류를 그림 5에서 픽셀과 화살표로 표시하였다. 그림에서 회색 픽셀은 이진화된 물체의 경계선 픽셀, 'w(white)'로 표시된 픽셀은 물체 외부의 픽셀, 그리고 'd'는 고려할 필요가 없는 픽셀을 의미한다. 그림의 가운데 픽셀은 경계선 상의 픽셀로서 이곳에 위치한 동그라미가 달린 화살표는 현재 추적자의 위치와 방향성을 의미한다. 또한 점선 화살표는 다음 경계선 픽셀까지 이동하기 위한 경로를 나타내고, 실선 화살표는 목적지인 다음 경계선 픽셀에서의 추적자의 위치와 방향성을 의미한다. 그림의 의미는 다음 표 1과 같다.




(a) 좌측 픽셀 (b) 좌하측 인너-아우터 코너 (c) 좌상측 인너-아우터 코너 (d) 상측의 인너코너 (e) 상측의 픽셀 (f) 아우터 코너

그림 5 물체 경계선 픽셀의 배치 및 경계선 픽셀간 이동 경로의 종류

표 1 경계선 픽셀 배치 종류와 추적자의 연산 후 정보

| Case | 경계선 픽셀 배치 형태 | 특징 | 이동 후 추적자의 정보 | |
|---|---|---|---|---|
| | | | 위치 | 방향성 |
| A | 좌측 인접 | 일반적인 형태로 코드의 좌측에 인접함. 때로 인너코너를 구성하는 일부일 수 있음 | 좌측 | 좌향 |
| B | 좌하측 인너-아우터 코너 | SBF에서 추적하지 못하는 인너-아우터 코너의 배치 형태 | 좌하측 | 역방향 |
| C | 좌상측 인너-아우터 코너 | SBF에서 추적되는 인너-아우터 코너의 배치 형태 | 좌상측 | 순방향 |
| D | 상측의 인너코너 | SBF, MSBF에서 추적 못하는 인너코너 형태 | 좌상측 | 순방향 |
| | | | 인너코너 추적을 위해 상측 픽셀의 위치를 마크 표시함 | |
| E | 상측의 인접 픽셀 | 일반적인 인접 픽셀 형태 | 상측 | 우향 |
| F | 아우터 코너 | 일반적인 아우터 픽셀 형태 | 현재 픽셀 | 역방향 |

그림 5의 (a)와 (e)의 배치도는 일반적인 인접 경계선의 경우이며, 때로는 SBF와 MSBF에서 추적가능한 인너코너의 일부일 수 있다. 그림 5(f)는 일반적인 아우터 코너일 때 나타나며, (c)는 SBF와 MSBF에서 추적이 가능한 인너-아우터 코너의 배치도이다. 그림 5(b)는 SBF에서 추적하지 못하고, MSBF에서 추적이 가능한 인너-아우터 코너의 배치도이며, (d)는 SBF, MSBF 모두 추적이 불가능한 인너코너의 배치의 경우이다.

이동경로를 6가지로 분류한 이유는, 현재의 경계선 픽셀에서 MSBF 기법으로 추적할 수 있는 다음 경계선 픽셀의 배치 관계가 (a),(b),(c),(e),(f)의 5가지이고 이에 인너코너 비일관성이 일어나는 (d)의 경우를 추가하였기 때문이다. 물론 경계선 픽셀이 우측, 우상, 우하, 그리고 하측에도 배치될 수 있으나 이런 경우는 모두 현재 경계선 픽셀이 아우터 코너인 경우로서, 먼저 (f) 단계를 거친 이후 다음 단계에서 추적이 가능하다.

### 3.2 규칙

ISBF에서는 이진화된 영상에서 현재 추적자의 위치를 기준으로 앞에서 분류한 6가지로 경계선상 픽셀 배치 형태 중 어떠한 것인지를 판별한 후, 이를 기반으로 추적을 하여야 한다. SBF와 MSBF에서는 경로상에 배치된 w 픽셀 부분도 연산을 하여야 하는데, 각각의 위치에서 픽셀의 값을 판별하는 감지연산과 다음 픽셀로의 이동연산을 포함하고 있다. 그러나 ISBF에서 고려할 것은 다음 경계선 픽셀의 위치와 방향성이기 때문에 이동 경로상의 픽셀에서는 이동연산을 수행하지 않는다. 다만 6가지 분류를 하기 위해서는 추적자 주위의 픽셀 값 감지연산을 최소화할 수 있도록 연산순서를 분석하여 설계하여야 한다. 예를 들어 추적자의 좌측 픽셀 감지연산, 즉 좌측 픽셀값을 판별하는 과정은 6가지 이동경로 판별에 모두 필요한 연산이므로 가장 먼저 수행하는 것이 효율적이다. 이때 좌측 픽셀이 경계선상의 픽셀이면 추적자는 그림 5(a)에서 보이듯이 마찬가지로 왼쪽으로 이동한다. 그렇지 않고 w픽셀이면 Case A를 제외한 나머지 경우를 고려하여야 하는데 Case B의 경우를 먼저 판별한 후 나머지 경우를 판별하여야 한다. 그렇지 않으면 좌하측 인너-아우터 코너를 추적하지 못하는 비일관성의 경우가 발생하기 때문이다. 이런 방식에 따라 경로의 판별은 Case A부터 Case F의 순서로 진행하는 것이 효율적이다. ISBF의 프로시저는 다음과 같다.

(1) 조건

- MSBF와 같은 조건을 가진다.

(2) 프로시저

- Step 1: 추적자는 경계선 상의 한 점에서 임의의 방향성을 가지고 시작한다. 그리고 대각선 추적 방지태그(이하 태그)를 0으로 설정한다.
- Step 2: 다음 중 조건을 만족하는 Case를 수행한 후 Step 3을 수행한다.

  Case A: 좌측 픽셀이 1이면 추적자를 좌측 픽셀로 이동시키고, 방향성도 좌향으로 지정한다.

  Case B: 좌측과 하측의 픽셀이 0이고, 좌하측의 픽셀이 1, 그리고 태그가 0이면 추적자를 좌하측 픽셀

로 이동시키고 방향성을 반대로 지정한다.

Case C/D: Case A, B가 아니고, 좌상측 픽셀이 1이면 추적자를 좌상측 픽셀로 이동만 한다(Case C). 이때 이동전 상측 픽셀이 1이면 상측 픽셀 위치를 마크처리한 후 수행한다(Case D).

Case E/F: Case A~D가 아니고 상측이 1이면 추적자를 상측 픽셀로 이동시키고 방향성을 우향으로 지정한다(Case E). 그렇지 않으면 추적자의 위치를 변경하지 않고 방향만 역방향으로 바꾼다(Case F).

- Step 3: Case B를 수행한 경우에 태그를 1, 그 외의 경우이면 태그를 0으로 설정한 후, 추적자의 변경된 픽셀 좌표와 방향성이 시작할 때의 좌표 및 방향성과 일치하는 지 검사한다. 일치하면 종료하고 그렇지 않으면 step 2를 재수행한다.

## 4. 알고리즘 분석 및 비교

제안된 알고리즘을 기존의 알고리즘과 비교하여 분석하기 위해서 먼저 알고리즘에 필요한 연산들의 종류와 연산량을 정의하였다. 이후 각 알고리즘이 탐색할 수 있는 경로의 종류와 그에 따른 복잡도를 상호 분석하였다.

### 4.1 연산자의 종류

세 가지 알고리즘에서 사용하는 연산의 종류는 크게 픽셀의 값을 알아내는 감지연산, 추적자를 이동시키는 이동연산, 어떠한 조건을 만족하는 지 점검하는 체크연산, 어떠한 조건을 설정해주는 설정연산, 그리고 픽셀의 특정 위치를 알려주는 마크연산이 있다. 이러한 연산들을 정의하기 전에 필요한 세 가지 기본연산들을 먼저 정의하도록 한다.

4.1.1 기본 연산

영상을 이진화하여 얻어진 이미지를 I라고 할 때, 이 이미지는 N×N개의 픽셀로 이루어져 있다고 가정하자. 이때 어떠한 픽셀의 2차원 좌표 P는 다음 식과 같이 표현할 수 있다.

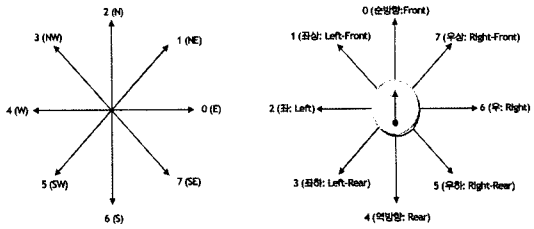
$$p=(x,y),\quad 0 \leqq x,y \leqq N-1,\ i.e.\ p \in I$$

현재 추적자가 위치한 좌표를 $p_c=(x_c,y_c)$, 추적자의 절대방향 정보를 $d_c$라고 하고, 추적자 정보는 $F=(p_c,d_c)$라고 하자. 이때 절대방향 정보 $d_a$와 상대방향 정보 $d_r$은 그림 6과 같이 8방향 코드로 표현하였다.

(1) 위치결정 연산(AP: Assigning Position)

위치결정 연산은 현재 추적자로부터 특정한 방향에 있는 인접 픽셀의 위치 정보를 구하는 연산을 의미한다. 이때 추적자($p_c$)로부터 절대방향 $d_a$에 위치한 픽셀의 좌표 $p_n$을 구하는 위치결정 연산은 다음과 같다.

$$AP(p_c,d_a)=(x_n,y_n)=p_n\ ,\ \ p_c \in I, p_n \in I$$




(a) 절대방향 정보 $d_a$ (b) 상대방향 정보 $d_r$

그림 6 절대방향 정보와 상대방향 정보의 8방향 코드

이 때 위치결정연산의 복잡도 $T(AP)$ = 1이라고 하자.

(2) 방향결정 연산(AD: Assigning Direction)

방향결정 연산은 어떠한 절대 방향성 $d_a$에 대해 상대적인 방향 $d_r$만큼 회전하여 구해지는 새로운 절대 방향성 $d_n$을 구하는 연산이다. 8방향 코드를 사용하므로 회전 연산은 절대방향값과 상대방향값을 더한 후 8로 나눈 나머지를 구하면 얻을 수 있다.

$$AD(d_a,d_r)=(d_a+d_r)\ MOD\ \ 8=d_n$$

$$T(AD)=1$$

(3) 픽셀감지 연산(DP: Detection Pixel)

픽셀감지 연산은 특정 좌표에 있는 픽셀값이 0인지 1인지 판별하는 연산이다. 픽셀 $p$가 관심이 있는 물체 O 위에 있는 경우는 1이고, 그렇지 않으면 0의 값을 가진다.

$$DP(p)=\begin{cases}1\ where & p \in O,\ O \subset I\\ 0\ otherwise\end{cases}$$

$$T(DP)=1$$

4.1.2 연산

세 가지 알고리즘을 위한 감지연산, 추적자를 이동시키는 이동연산, 어떠한 조건을 만족하는 지 점검하는 체크연산, 어떠한 조건을 설정해주는 설정연산, 그리고 픽셀의 특정 위치를 알려주는 마크연산들은 다음과 같이 분류할 수 있다.

(1) 감지 연산(Detection)

감지 연산은 추적자 혹은 추적자 인접픽셀의 값을 알려주는 연산으로 특정한 픽셀의 값을 구하는 연산이다. 픽셀감지 연산과 마찬가지로 특정한 좌표가 주어지면 그에 대한 픽셀값을 알려준다. 감지연산은 추적자의 현재위치 픽셀값을 알려주는 추적자감지연산 DC(Detecting the Current pixel)와, 추적자 주위의 인접픽셀값을 알려주는 DN(Detecting a Neighbor pixel) 연산으로 나뉜다.

$$DC(p_c)=DP(p_c)$$

$$DN(F_c,d_r)=DP(AP(p_c,AD(d_c,d_r)))$$

추적자 위치의 픽셀값을 구하는 경우는 DP의 연산량과 같고, 추적자 주위의 픽셀값을 판별하는 경우에는 먼저 방향결정 연산 후 위치결정 연산을 수행하고 DP 연

산을 해야 하므로 연산복잡도는 다음과 같이 구할 수 있다.

$$T(DC)=1,\qquad T(DN)=3$$

(2) 이동 연산(Move operation)

이동 연산은 추적자를 특정한 픽셀로 이동해주는 연산으로서 경로정보의 종류에 따라 다음 픽셀로 이동할 때 사용한다. 만약 $d_{r1}$이 추적자로부터 인접 좌표의 위치를 얻기 위한 상대방향정보이고, $d_{r2}$가 추적자의 방향정보를 바꾸기 위한 상대방향정보라고 하면 이동연산은 다음 식과 같이 표현한다.

$$M(F_c,d_{r1},d_{r2})=(AP(p_{c,}AD(d_c,d_{r1})),AD(d_c,d_{r2}))=F_n$$

이때 만약 $d_{r1}=d_{r2}=d_r$ 이라면 다음과 같이 식을 바꿀 수 있다.

$$M(F_c,d_r)=(AP(p_{c,}d_n)),d_n)=F_n\ ,\quad d_n=AD(d_c,d_r)$$

따라서 연산복잡도는 다음과 같이 두 가지 경우가 있다.

$$T(M)=\begin{cases}2\,where & d_{r1}=d_{r2}\\ 3\,otherwise\end{cases}$$

여기서는 편의상 $T(M)$=2인 이동연산을 $M_2$라고 하고, $T(M)$=3인 연산을 $M_3$라고 하자.

(3) 대각선 추적방지 태그설정 연산 (STD: Setting a tag for preventing a Diagonal Detour)

MSBF와 ISBF에서는 Case B처럼 좌하측 인너-아우터 코너를 탐지하여 이동한 이후 대각선방지 태그를 1로 설정하고, 이후 추적에서 태그가 1이면 인너-아우터 코너 추적 단계를 건너뜀으로써 무한루프를 방지할 수 있다. 대각선추적방지 태그 연산은 Case B 직후 태그를 1로 설정하고, 다른 경우의 경우에는 0으로 설정해줌으로써 효과적인 Case B처리를 가능토록 해준다.

$$STD(tag,case)=\begin{cases}tag=1\,where\quad Case\,B\\ tag=0\,otherwise\end{cases}$$

$$T(STD)=1$$

(4) 조건비교 연산(Checking operation)

조건비교 연산에는 종료조건 연산(CE: Checking End condition)과 대각선추적조건 연산(CTD: Checking Tag condition for Diagonal detour)이 있다. 세 가지 알고리즘은 공히 추적을 시작할 때의 추적자의 정보 $Fs$를 기억해두었다가 현재의 추적자 $F_c$가 물체 위에 있을 때마다 서로 동일한지 검사하는 종료조건 연산을 수행하고, 그 결과가 1이면 종료하게 된다. 대각선추적조건 연산은 대각선추적방지 태그의 상태를 읽고 이에 따라 Case B처리를 하기 위한 것이다.

$$CE(F_s,F_c)=\begin{cases}1\,where\quad F_s=F_c\\ 0\,otherwise\end{cases}$$

$$CTD(tag)=\begin{cases}1\,where\qquad tag=0\\ 0\,otherwise\end{cases}$$

$$T(CE)=1\ \ T(CTD)=1$$

이때 종료조건 연산은 추적자의 좌표 비교와 방향정보 비교를 하므로 복잡도가 최대 2이지만, 물체의 경계선 픽셀의 개수 M이 충분히 클 때, 두 가지 연산이 모두 일어나는 경우는 시작점에 도착했을 때뿐이고, 대부분의 경우 좌표 비교만으로 이 연산은 종료되므로 복잡도는 1이라고 할 수 있다.

(5) 마크연산(Marking operation)

마크 연산은 Case D와 같이 인너코너에 위치한 경우 이를 표시해두는 연산이다. 마크 연산은 실질적으로는 추적자의 누적된 추적경로에 포함을 시키거나 별도로 정보를 저장해둔 후 추후 다른 작업을 수행할 수 있다. 즉, 안이 비어있는 물체의 경우에는 재귀호출기법을 사용하여 이 지점부터 안쪽의 별도의 물체를 탐색할 수도 있는 것이다. 마크연산의 경우에는 인너코너의 좌표위치와 추적자의 현재 절대방향 정보를 저장한다.

$$Mark(F_c)=(AP(p_{c,}AD(d_c,d_r)),d_c)=F_M$$

$$T(Mark)=2$$

이러한 연산들 이외에 추적자의 처음 시작 정보를 설정하는 연산도 필요하지만, 초기에 한 번만 발생하므로 연산복잡도 계산에는 의미를 가지지 않는다. 또한 추적자의 지난 경로들을 기억하기 위한 연산도 제외하였다. 표 2는 연산들의 종류와 연산복잡도를 정리한 것이다.

표 2 연산의 종류

| 연산종류 | 연산 | 내용 | 연산복잡도(T) |
|---|---|---|---|
| 감지연산 | DC | Detecting the Current pixel | 1 |
| | DN | Detecting a Neighbor pixel | 3 |
| 이동연산 | M | Move operation | 2 or 3 |
| 설정연산 | STD | Setting a tag for preventing a Diagonal detour | 1 |
| 조건비교 연산 | CE | Checking End condition | 1 |
| | CTD | Checking Tag condition for Diagonal detour | 1 |
| 마크연산 | Mark | Marking an inner-corner pixel | 2 |

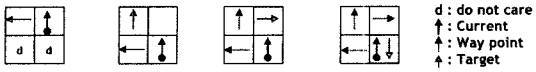
### 4.2 SBF의 알고리즘 분석

SBF 알고리즘과 다른 알고리즘을 비교하기 위하여 먼저 SBF 알고리즘에서 추적자가 이동할 수 있는 물체 경계선상의 픽셀 배치와 이동경로의 종류와 SBF 알고리즘의 의사코드(pseudo code)를 제시하였다. 그리고 이를 바탕으로 연산의 복잡도를 분석한 결과를 제시하였다.

4.2.1 SBF 알고리즘의 픽셀 배치와 이동경로의 종류

SBF에서는 물체경계선상의 픽셀과 다음 픽셀간 배치의 경우로 네 가지 종류가 존재하며 그 종류는 그림 7과 같다. SBF에서는 인너-코너 비일관성과 아우터-코

너 비일관성이 있으므로 ISBF와 달리 2가지 경우를 처리할 수 없다. 따라서 그림 5와 표 1에 제시된 경우들 중에서 처리가능한 경우는 Case A, C, E, F이다.




(a) Case A (b) Case C (c) Case E (d) Case F

그림 7 SBF의 경계선 픽셀의 배치 및 경계선 픽셀간 이동 경로의 종류

4.2.2 SBF의 연산복잡도

(1) SBF의 의사코드

```
SBF(F_S, I, O) // F_S : 시작시 추적자(좌표,방향), I: Image ,
                                    // O: object
   F_C ← F_S ;                      // F_C : 추적자의 정보
   Append F_C to path;
   Do
       if (p_C ∈ O)                 // DC
           MOVE(F_C, Left) ;  //M_2: Fc turns left and goes ahead
           Append F_C to path;
       else
           MOVE(F_C, Right) ;       //M_2 turn right and go ahead
       if ( F_C == F_S ) return path;   //CE
   END DO
END SBF

MOVE (F, d_r) //M_2 (F , d_r), F=((x,y), d), d :절대방향, d_r :상대방향
   d ← AD(d , d_r);
   p_c ← AP(p , d );
END MOVE
```

(2) 연산복잡도

SBF에서는 물체경계선상의 픽셀과 그렇지 않은 픽셀을 함께 추적하여 처리해야 한다. 이미지상의 어떠한 물체 O의 경계를 추적하는 연산량은 각 픽셀의 배치 방식들, 즉 추적자의 경로를 이루는 각각 Case들의 개수($n_i$)와 각 Case들을 구성하는 연산량들과 곱한 후 모두 합산하면 구할 수 있다. 이때 물체경계선상 픽셀 하나에서 이루어지는 연산의 복잡도를 T(B), 그렇지 않은 픽셀에서 이루어지는 연산복잡도를 T(W)라고 하고, 해당 Case i에서 연산에 필요한 B픽셀의 개수를 $n_{iB}$, W픽셀의 개수를 $n_{iW}$라고 하면 물체를 추적하는 연산복잡도는 다음과 같이 표현할 수 있다.

$$T(O) = \sum_i n_i T(case_i) = \sum_i n_i (n_{iB} T(B) + n_{iW} T(W)),$$
$$n_i = n_{iB} + n_{iW}, i = A, C, E, F$$

분류된 이동경로의 종류들 중 Case A의 경우에는 현재 추적자가 위치한 B픽셀에서만 연산이 일어나는데, 의사코드에서 보듯이 현재위치 픽셀값을 추출하는 추적자감지연산(DC), 왼쪽의 픽셀로 이동시키는 이동연산($M_2$), 그리고 종료조건을 판단하는 종료조건 연산(CE)이 1번씩 발생한다. Case C의 경우에는 B픽셀 1개와 W픽셀 1개에서 연산이 일어나는데 DC연산 2번, $M_2$연산 2번, 그리고 CE가 1번 발생한다. 따라서 각 경우별로 연산복잡도를 구하면 다음과 같다.

$$T(Case\ A) = 1T(B) + 0T(W)$$
$$= T(DC) + T(M_2) + T(CE) = 4$$

$$T(Case\ C) = 1T(B) + 1T(W)$$
$$= 2T(DC) + 2T(M_2) + T(CE) = 7$$

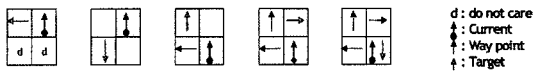
$$T(Case\ E) = 1T(B) + 2T(W)$$
$$= 3T(DC) + 3T(M_2) + T(CE) = 10$$

$$T(Case\ F) = 1T(B) + 3T(W)$$
$$= 4(DC) + 4T(M_2) + T(CE) = 13$$

### 4.3 MSBF의 알고리즘 분석

4.3.1 MSBF 알고리즘의 픽셀 배치와 이동경로의 종류

MSBF에서는 물체경계선상의 픽셀과 다음 픽셀간 배치의 경우로 다섯 가지 종류가 존재하며 그 종류는 그림 8과 같다. MSBF에서는 Case D의 경우 인너코너 비일관성이 있어서 추적하지 못하므로 그림 5에서 Case A, B, C, E, F의 경우를 처리할 수 있다.




(a) Case A (b) Case B (c) Case C (d) Case E (e) Case F

그림 8 MSBF의 경계선 픽셀 배치 및 경계선 픽셀간 이동 경로의 종류

4.3.2 MSBF의 연산복잡도

(1) MSBF의 의사코드

```
MSBF(F_S, I, O)  // F_S:시작시 추적자의 정보(좌표,방향), I:Image ,O:object
    F_C ← F_S ;  // F_S : 추적자의 정보
    tag ← false;
    Do
        if (p_C ∈ O)                                        // DC
            Append F_C to path;
            if (tag == false) and (p_Left ∉ O) and (p_Rear ∉ O)
               and (p_Left-Rear ∈ O)                        // CTD, 3DN
                MOVE_NEW(F_C, Left − Rear, Rear);
                                  //M_3 좌하측 인너-아우터 코너 이동
                tag ← true;                                 // STD
            else
                MOVE(F_C, Left) ;  // M_2: turn left and go ahead
                tag ← false;                                // STD
        else  if ( F_C == F_S ) return path;  //CE
            MOVE(F_C , Right) ;     // M_2: turn right and go ahead
            tag ← false;            //STD
    END DO
END MSBF

MOVE_NEW (F, d_r1, d_r2)      //M_3 (F , d_r1, d_r2), F=((x,y), d),
                              // d : 절대 방향정보, d_r1, d_r2 :상대방향정보
    MOVE(F, d_r1);
    d ← AD(d , d_r2);
END MOVE_NEW
```

(2) 연산복잡도

MSBF도 SBF와 마찬가지로 B픽셀과 W픽셀 모두를

추적하여야 한다. 다만 인너-아우터 코너 처리를 위한 이동연산시에는 새로운 좌표 위치를 설정하기 위한 연산 AP와 새로운 추적자의 절대방향 정보를 설정하기 위한 AD연산으로 구성되므로 이 경우에는 $T(M_3)$=3이 요구된다. 따라서 이동연산이 두 가지가 있게 되므로 각 경우마다 B픽셀에서 이루어지는 연산량이 달라진다. 따라서 물체의 경계선(O)를 추적할 때의 연산량은 아래 식과 같이 표현할 수 있다.

$$T(O)=\sum_i n_i T(case_i)=\sum_i (n_{iB}T_i(B)+n_{iW}T(W)),$$
$$n_i=n_{iB}+n_{iW}, i=A,B,C,E,F$$

MSBF에서는 인너-아우터 처리를 우선적으로 해야 한다. 따라서 현재의 위치가 B픽셀인 경우, 좌측, 좌하측, 그리고 하측의 픽셀값을 판별해야 하며, 좌측과 하측이 W픽셀이고 좌하측이 B픽셀이어야 한다. 또한 대각선추적 방지태그값이 거짓(false)이어야 한다. 따라서 이들 중 하나만 해당이 되지 않아도 인너-아우터 처리 연산을 종료시킬 수 있으나 최악의 경우에는 5개의 조건을 검사해야 하므로 Case A가 가지는 최악의 경우의 연산복잡도는 15가 된다.

$$T(Case\ A)=1T_A(B)$$
$$=\begin{cases} T(DC)+T(CE)+T(CTD)+3T(DN) \\ \quad +T(M_2)+T(STD)=15 & (Max.) \\ T(DC)+T(CE)+T(CTD)+T(M_2) \\ \quad +T(STD)=6 & (Min.) \end{cases}$$

$$T(Case\ B) = T(DC)+T(CE)+T(CTD) +3T(DN)+T(M_3)+T(STD)=16$$

$$T(Case\ C)=$$
$$\begin{cases} T(Case\ B)-T(M_{new})+T(M_2)+T(W)= \\ \quad 15+T(DC)+T(M_2)+T(STD)=19 & (Max.) \\ T(DC)+T(CE)+T(CTD)+T(M_2)+ \\ \quad T(STD)+T(W)=10 & (Min.) \end{cases}$$

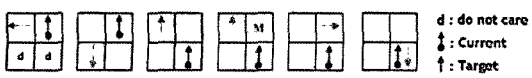
$$T(Case\ E)=\begin{cases} \max(T(Case\ C))+T(W) \\ \quad =19+4=23 & (Max.) \\ \min(T(Case\ C))+T(W) \\ \quad =10+4=14 & (Min.) \end{cases}$$

$$T(Case\ F)=\begin{cases} \max(T(Case\ D))+T(W) \\ \quad =23+4=27 & (Max.) \\ \min(T(Case\ D))+T(W) \\ \quad =14+4=18 & (Min.) \end{cases}$$

### 4.4 ISBF의 알고리즘 분석

4.4.1 ISBF 알고리즘의 픽셀 배치와 이동경로의 종류

ISBF는 그림 5의 경우를 모두 처리할 수 있으며, 이를 보다 간략화한 것은 그림 9와 같다.




(a) Case A (b) Case B (c) Case C (d) Case D (e) Case E (f) Case F

그림 9 ISBF의 경계선 픽셀의 배치 및 경계선 픽셀간 이동 경로의 종류

4.4.2 ISBF의 연산복잡도

(1) ISBF의 의사코드

```
ISBF(F_S, I, O)      // F_S : 시작시 추적자(좌표,방향), I: Image , O: object
  F_C ← F_S ;        // F_C : 추적자의 정보
  Append F_C to path;
  tag ← false;
  Do
    if (p_Left ∈ O)                          // Case A // DN
      MOVE(F_C, Left) ;                      // M_2: turn left and go ahead
      tag ← false;                           // STD
    else
      if (tag == false)and(p_Rear ∉ O)and(p_Left-Rear ∈ O)
                                             // Case B  //CTD, 2DN
        MOVE_NEW(F_C, Left − Rear, Rear) ;
                                             // M_3 좌하측 인너-아우터 코너 이동
        tag ← true;                          // STD
      else
        if (p_Front-Left ∈ O)                // Case C & Case D
          if (p_Front ∈ O)                   // DN
            F_M ← MARK(F_C, Front);          // MARK
          p_C ← AP(p_C, AD(d_C, Left − Front));
                                             // M_2: 좌상측 이동 및 순방향지정
        else if (p_Front ∈ O)                // Case E
          MOVE_NEW(F_C, Front, Right) ;
                                             // M_3: 상측 이동 및 우향 정보지정
        else
          d_C ← AD(d_C, Rear);               // Case F
        tag ← false;                         // STD
    Append F_C to path;
    if ( F_C == F_S ) return path;           //CE
  END DO
END ISBF

MARK (F, d_r)
  MARKS[] ← (AP(F, Front), d_r);
END MOVE
```

(2) 연산복잡도

ISBF는 MSBF와 달리 W픽셀에서 이동 연산이 없으며 인너코너 처리를 위한 MARK연산이 포함된다. 다만 B픽셀 연산에는 주변의 W픽셀을 감지하는 연산량이 포함되어 있다.

$$T(O)=\sum_{i=A}^{F} n_i T(case_i)=\sum_{i=A}^{F} n_i T_i(B)$$

연산복잡도는 아래와 같이 구할 수 있다.

$$T(Case\ A)=T(DN)+T(M_2)+T(STD)+T(CE)=7$$

$$T(Case\ B)=T(DN)+T(CTD)+2T(DN) +T(M_3)+T(STD)+T(CE)=15$$

$$T(Case\ C)=\begin{cases} T(Case\ B)-T(M_3)+T(AP) \\ \quad +T(AD)+2T(DN)=20 & (Max.) \\ T(DN)+T(CTD)+2T(DN)+T(AP) \\ \quad +T(AD)+T(STD)+T(CE)=14 & (Min.) \end{cases}$$

$$T(Case\ D)=\begin{cases} \max(T(Case\ C))+T(MARK) \\ \quad =20+2=22 \quad (Max.) \\ \min(T(Case\ C))+T(MARK) \\ \quad =14+2=16 \quad (Min.) \end{cases}$$

$$T(Case\ E)=\begin{cases} \max(T(Case\ C))-T(AP)-T(AD)-T(DN) \\ \quad +T(M_3)+T(DN)=15+6=21 \quad (Max.) \\ \min(T(Case\ C))-T(AP)-T(AD)-T(DN) \\ \quad +T(M_3)+T(DN)=9+6=15 \quad (Min.) \end{cases}$$

$$T(Case\ F)=\begin{cases} \max(T(Case\ E))-T(M_3)+T(AD) \\ \quad =21-3+1=19 & (Max.) \\ \min(T(Case\ E))-T(M_3)+T(AD) \\ \quad =15-2=13 & (Min.) \end{cases}$$

### 4.5 실험

SBF와 MSBF, 그리고 ISBF 알고리즘의 성능을 검증하기 위하여 인너코너와 인너-아우터 코너가 들어 있는 이미지를 대상으로 각기 알고리즘을 이용하여 추적자의 경로를 추적하였다. 실험 대상 이미지로는 그림 3의 (b)와 (c)를 이용하였는데, (c)에 있는 인너코너 탐지 능력을 먼저 비교하고, 이후 (b)의 인너-아우터 코너 탐지 가능 여부를 비교하였다.

4.5.1 인너-코너 추적 비일관성 처리

그림 10은 그림 3(c)를 대상으로 한 세 알고리즘의 추적 경로와 경계선 픽셀의 처리 연산량을 나타낸 것이다. 그림 10의 (a)와 (e)는 이미지에서 물체를 이진화 정보로 표현한 것으로 물체의 영역을 B와 1로 표현하였고, 그 외의 영역은 W와 0으로 표현하였다. 그림에서 (b)~(d)는 세 알고리즘이 추적한 경로를 나타낸 것으로 '*'표시는 추적한 B 픽셀을 의미하고 '^' 기호는 추적자가 위치했던 W 픽셀을 의미한다. (f)~(h)는 추적한 B 픽셀의 카운터를 표시하기 위한 것으로서 '-1'은 이동연산이 수행된 W 픽셀을 의미한다.

이때 추적자의 시작 위치와 종료 위치는 같기 때문에 가장 큰 숫자가 위치한 곳이 시작점이자 종료점이 되고, 그 숫자는 최종 수행된 B 픽셀의 카운터이다. 이해를 돕기 위해 추적자가 지나간 경계선 픽셀은 음영처리를 하였다. (f)~(h)에서 (5,1), (7,3), (8,6) 좌표의 경우에는 카운터가 2씩 증가하였는데, 이는 아우터-코너 연산을 통해 추적자가 두 번 추적했기 때문이다. 추적자의 시작 위치는 그림 (a)에서 S로 표시한 (1,1)이며, 시작시 E 방향 정보를 가지고 추적을 개시하였다. 알고리즘을 수행한 결과, SBF와 MSBF는 인너코너인 (5,3)의 경계선 픽셀을 추적하지 못하였으나 ISBF에서는 추적을 하였으므로 B픽셀에서 연산 횟수가 더 많았다. 그리고 SBF와 MSBF에서는 W픽셀들에서 이동 연산이 발생하였으나 ISBF에서는 이동연산이 없음을 보인다.

4.5.2 인너-아우터 코너 추적 비일관성 처리

그림 11은 그림 3의 (b)를 대상으로 인너-아우터 코너 처리에 대한 세 알고리즘의 추적 경로와 경계선 픽셀의 처리 개수를 나타낸 것이다. 이 실험에서 추적자의 시작 위치는 그림 11(a)에서 S로 표시한 (1,6)이며, 시작시 E방향 정보를 가지고 추적을 개시하였다. 알고리즘을 수행한 결과, 그림 (b)~(d)에서 보듯이 SBF는 이미지의 우상측쪽의 인너-아우터 코너를 추적하지 못한 반면 MSBF와 ISBF는 이 영역을 추적할 수 있었다.

MSBF와 ISBF는 인너-아우터 코너를 추적함에도 불구하고 B 픽셀의 연산 횟수에서 2개의 차이를 보였는데 이는 이미지에 포함되어 있던 인너코너 픽셀 처리 때문에 발생한 것이다. 이미지에는 인너코너 픽셀들이 (2,5), (3,4), (3,6) 좌표에 모두 3개 존재하는데, MSBF와 ISBF는 이들 중 (2,5)와 (3,4)에 대해서 차이를 보였다. 즉, ISBF에서 추적자는 (2,5) 픽셀을 두 번 추적하는데, 경계선 추적 중 두 번째 B 픽셀과 35번째 B픽셀로서 추적하였음을 그림 11(h)에서 보여주고 있다. 또한 (3,4) 픽셀은 ISBF에서 네 번째 B픽셀로 처리되었다. 그러나 MSBF는 (2,5)를 33번째 B픽셀로 한 번 추적하고, (3,4)는 추적하지 못하였다. 이 때문에 ISBF와 MSBF 사이에서 2라는 연산 횟수 차이가 발생한 것이다. 세 알고리즘에 대해 비일관성에 대한 처리를 실험한 결과 ISBF

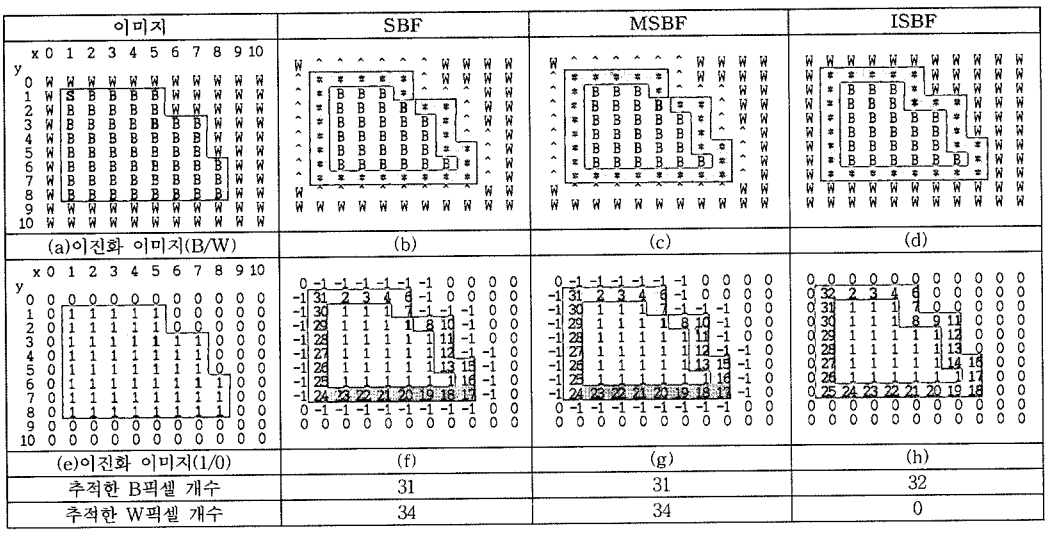



그림 10 인너-코너 비일관성에 대한 알고리즘 비교

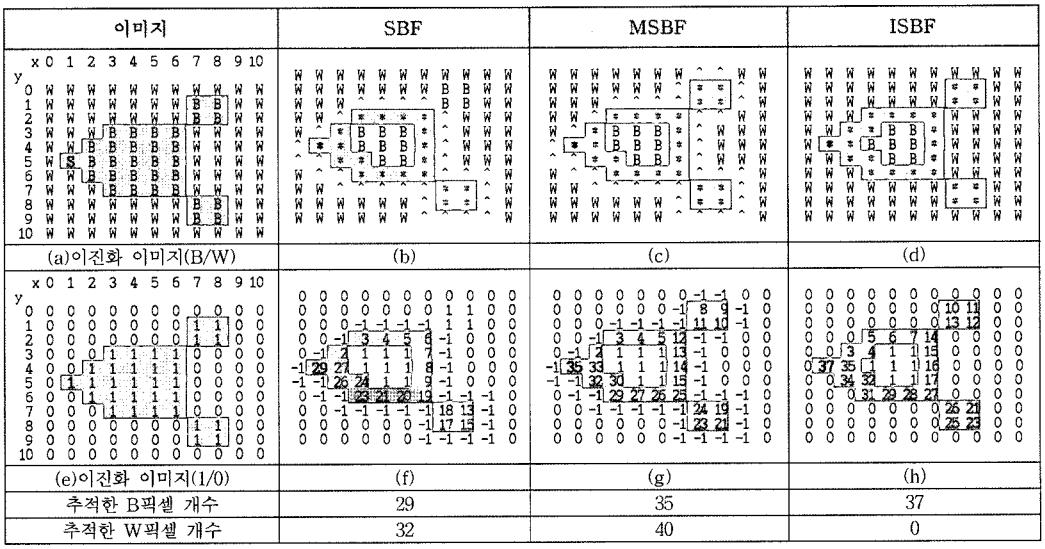



그림 11 인너-아우터 코너 비일관성에 대한 알고리즘 비교

는 비일관성 문제가 없으며 경계선 이외의 픽셀에서는 이동 연산이 없음을 알 수 있다.

4.5.3 연산량 비교

(1) 이론적 연산복잡도

SBF, MSBF, 그리고 제안된 알고리즘의 추적 가능한 경로의 종류와 연산복잡도는 표 3과 같다. MSBF에서 경우별로 최대값과 최소값이 발생하는 것은 인너-아우터 코너 연산 수행시 추적자의 인접픽셀의 값을 찾는 DN연산의 개수에 기인한 것이다. 추적자가 B픽셀에 있을 때, 좌하측 인너-아우터 코너 경우 판별시 태그가 true이면 다른 인접픽셀값을 조사할 필요없이 다른 케이스를 찾게 된다. 즉, CTD 연산시 true이면 Case B이후의 다른 경우를 찾게 되는 것이다. 그러나 태그가 false이면 좌측, 좌하측, 하측의 3개 인접 픽셀값을 조사해야 하므로 최대 3개의 DN 연산이 추가된다. 따라서 MSBF에서 Case C~F까지 최대와 최소의 연산복잡도는 9의 차이가 있게 되며 평균적으로 4.5의 차이가 발생한다. ISBF의 경우에는 좌측 픽셀 감지 연산을 미리 수행하므로 Case B이후 DN의 연산 개수 차이가 최대 2개이기 때문에 최대값과 최소값의 차이가 6이고, 평균적으로 연산복잡도는 3의 차이가 발생한다.

표 3 세 가지 알고리즘의 Case별 연산복잡도(∞: 추적 불능)

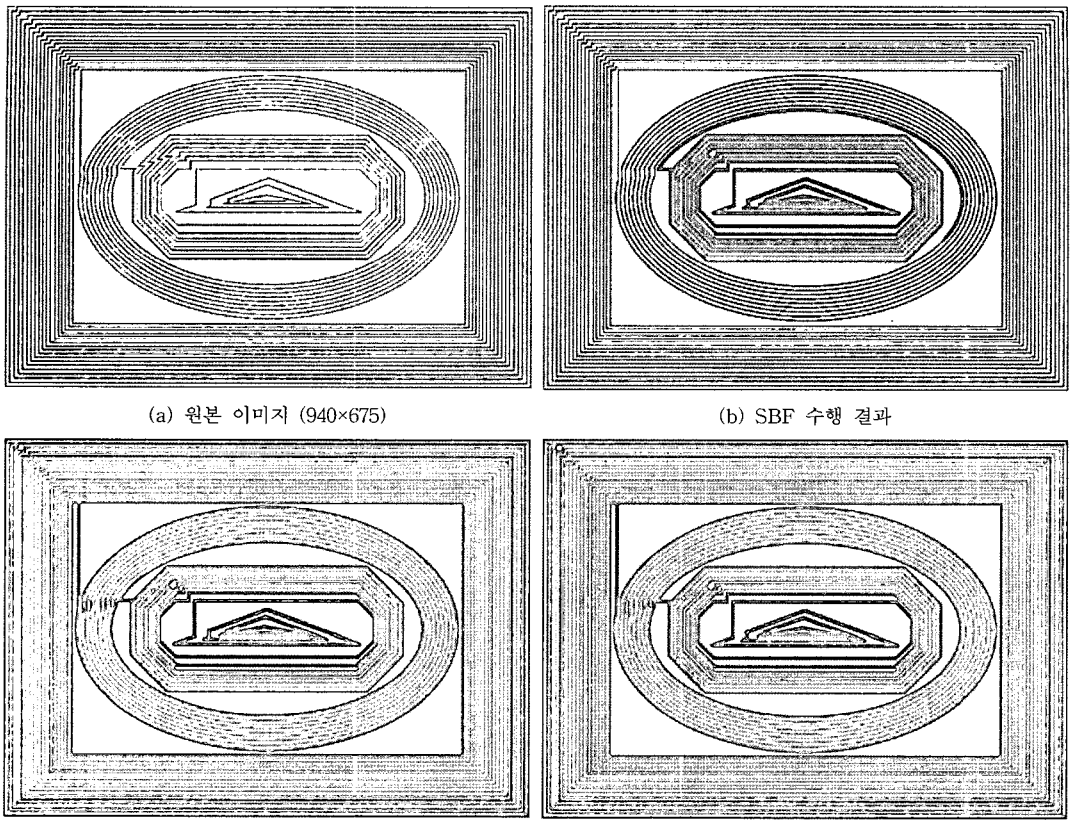
| Cases | SBF | MSBF | | | ISBF (Proposed) | | |
|---|---|---|---|---|---|---|---|
| | | Max. | Min. | Avg. | Max. | Min. | Avg. |
| Case A | 4 | 15 | 6 | 10.5 | 7 | | |
| Case B | ∞ | 16 | | | 15 | | |
| Case C | 7 | 19 | 10 | 14.5 | 20 | 14 | 17 |
| Case D | ∞ | ∞ | | | 22 | 16 | 19 |
| Case E | 10 | 23 | 14 | 18.5 | 21 | 15 | 18 |
| Case F | 13 | 27 | 18 | 22.5 | 19 | 13 | 16 |

표 3에서 제안된 알고리즘은 MSBF에 비해 평균적으로 연산복잡도가 낮으며 각 경우별로 최대 연산복잡도와 최소 연산복잡도의 차이가 적음을 보이고 있다.

(2) 이미지를 통한 실험 결과

경계선 픽셀 개수가 적은 경우에는 세 가지 알고리즘의 상대적인 속도 차이를 측정하기 어렵다. 따라서 그림 12의 (a)와 같이 경계선이 많은 예제 이미지를 이용하여 연산량을 비교하였다. 원본 이미지는 940×675 크기이며, 두께가 1~3픽셀인 하나의 경계선으로 이루어져있다. 실험을 위해 이 경계선의 시작점은 그림 중앙의 삼각형 모양 안의 선의 끝인 (500,343)으로 설정하였고, 초기 방향은 E로 하였다. 알고리즘들이 성공적으로 수행될 경우, 추적자는 그림 12 (a)의 좌상단에 위치한 선의 끝까지 추적한 후 다시 시작점까지 돌아오게 된다. 또한 추적자가 이미지의 한계 바깥을 감지하거나 이동하는 경우 해당 위치는 흰색 픽셀로 처리하여 프로그램상 오류가 일어나지 않도록 하였다.

프로그램은 Visual Studio 6.0에서 C++로 구현되었고, 콘솔 모드에서 이미지 파일을 읽은 후 처리 후 결과를 이미지 파일과 콘솔로 출력하도록 하였다. 실험결과, 그림 12에서 보듯, SBF는 추적을 실패하였고, MSBF와 제안한 알고리즘은 경계선을 추적하는데 성공하였다. 표 4는 세 가지 알고리즘의 추적 결과를 제시한 것으로서 추적한 경계선 픽셀의 개수와 윈도즈용 C언어 함수인 GetTickCount()를 이용한 속도 측정치를 보여준다. 실험결과 제안한 ISBF 알고리즘이 경계선 픽셀을 가장



(a) 원본 이미지 (940×675) (b) SBF 수행 결과

(c) MSBF 수행 결과 (d) ISBF 수행 결과

그림 12 예제 이미지를 이용한 세 가지 알고리즘의 추적 결과 비교 (추적한 경로를 굵은 선으로 처리함)

표 4 알고리즘의 성능 비교

| | SBF | | MSBF | | ISBF | |
|---|---|---|---|---|---|---|
| 실험환경 | Notebook: Samsung SENS V20<br>CPU: Intel Pentium 4 1.8GHz / RAM: 248MB /<br>OS: Microsoft Windows XP professional | | | | | |
| 경계선 픽셀 추적횟수 | 24,111 개 | | 156,689 개 | | 163,098 개 | |
| 비경계선 픽셀 추적횟수 | 24,114 개 | | 156,772 개 | | 0 개 | |
| 시행 (차수) | 총 연산 시간 (초) | 경계선 픽셀당 연산시간 ($ms:10^{-3}$초) | 총 연산 시간 (초) | 경계선 픽셀당 연산시간 ($ms:10^{-3}$초) | 총 연산 시간 (초) | 경계선 픽셀당 연산시간 ($ms:10^{-3}$초) |
| 1 | 0.021 | 0.000871 | 0.190 | 0.001213 | 0.150 | 0.000920 |
| 2 | 0.030 | 0.001244 | 0.190 | 0.001213 | 0.140 | 0.000858 |
| 3 | 0.030 | 0.001244 | 0.200 | 0.001276 | 0.150 | 0.000920 |
| 4 | 0.030 | 0.001244 | 0.190 | 0.001213 | 0.150 | 0.000920 |
| 5 | 0.020 | 0.000829 | 0.191 | 0.001219 | 0.160 | 0.000981 |
| 6 | 0.020 | 0.000829 | 0.200 | 0.001276 | 0.150 | 0.000920 |
| 7 | 0.020 | 0.000829 | 0.190 | 0.001213 | 0.141 | 0.000865 |
| 8 | 0.021 | 0.000871 | 0.180 | 0.001149 | 0.150 | 0.000920 |
| 9 | 0.020 | 0.000829 | 0.191 | 0.001219 | 0.150 | 0.000920 |
| 10 | 0.020 | 0.000829 | 0.200 | 0.001276 | 0.161 | 0.000987 |
| 평균시간 | 0.023 | 0.000962 | 0.192 | 0.001227 | 0.150 | 0.000921 |

많이 추적하였음을 보여주고 있는데, 이런 차이는 제안한 알고리즘이 비일관성들을 해결하였기 때문이다.

추적자가 추적을 시작한 시점부터 종료시점까지의 시간을 측정한 결과, ISBF의 경계선 픽셀 하나당 평균처리 시간이 셋 중에 상대적으로 가장 적었다. 다만, SBF의 경우 추적한 픽셀의 개수가 매우 적고, 총 연산시간이 0.05초도 되지 않기 때문에 픽셀당 연산 시간값의 정확도가 상대적으로 낮다. 따라서 경계선 추적 경로가 유사한 MSBF와 ISBF 알고리즘만을 서로 상대적으로 비교하는 것이 타당하며, ISBF가 MSBF보다 상대적으로 픽셀당 처리시간이 적다고 결론내릴 수 있다.

## 5. 결 론

본 논문에서는 영상처리 기법에 있어서 영상의 분할, 물체 형태 인식, 영상 합성 등의 분야에 사용되고 있는 SBF와 MSBF 기법을 소개하고 이들의 제약점을 보완할 수 있는 ISBF 기법을 제안하였다. 이를 위하여 먼저 관심이 되는 경계선상의 픽셀들의 위치 관계를 고려하여 탐색 경로들을 여섯 가지로 분류하였고, 이를 통해

SBF에서 추적이 되지 않는 인너-아우터 코너와 인너코너 추적 비일관성은 추적자로부터 좌하측 방향의 인너-아우터 코너 혹은 상측의 인너코너가 존재하는 경우임을 알 수 있었다. 그리고 MSBF의 경우에는 인너-아우터 코너 추적의 비일관성은 해결되었으나 상측의 인너코너가 추적되지 않았다. 이에 따라 기존 알고리즘의 비일관성을 해소하는 동시에 비경계선 픽셀상의 부가적인 연산량을 감소시킬 수 있는 알고리즘을 제안하였다. 알고리즘들을 비교 분석하기 위해서는 공통된 연산을 정의할 필요가 있으므로, 추적자가 경로를 탐색할 때 요구되는 기본 연산으로서 위치결정, 방향결정 및 픽셀감지 연산을 정의하였고, 또한 감지, 이동, 설정, 조건비교, 마크 연산을 정의하였다. 그리고 이를 이용하여 세 알고리즘의 의사코드들을 분석하였고, 이미지를 통해 추적능력에 대한 실험을 하였다. 의사코드 분석 결과 제안된 알고리즘에는 SBF와 MSBF에서 발생하는 인너코너 및 인너-아우터 코너 추적 비일관성이 없었다. 그리고 MSBF에 비해서 상대적으로 연산량이 적으며, 각 경우별로 최대 연산량과 최소 연산량 차이도 상대적으로 적다는 결론을 내릴 수 있었다. 또한 경계선이 많은 이미지를 처리한 결과 경계선 추적 횟수가 많고, 경계선 픽셀당 처리 시간이 상대적으로 적었다.

본 논문에서는 물체의 내부가 채워져 있는 경우나 단선으로 구성된 물체를 대상으로 알고리즘을 제안하고 분석하였는데, 인너코너로 연결되어진 물체나 안의 일부가 채워져 있지 않은 경우 등에는 추적이 불가능한 경우가 있다. 이를 위해 트리구조 기법이나 재귀호출법등을 이용한 추가적인 연구가 필요하다.

## 참 고 문 헌


[1] S. Marchand-Maillet, Y.M. Sharaiha, Binary Digital Image Processing, pp.173-180, Academic Press, 2000.

[2] M. Das, M.J. Paulik, N.K. Loh, "A Bivariate Autoregressive Modeling Technique for Analysis and Classification of Planar Shapes," IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 12, No. 1, pp. 97-103, January 1990.

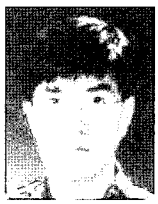
[3] S. Papert, "Uses of technology to enhance education," Technical report 298, AI Lab., MIT, 1973.

[4] I. Pitas, Digital Image Processing Algorithms and Applications, pp.275-329, John Wiley & Sons, 2000.

[5] N. Arica, F.T. Yarman-Vural, "An Overview of Character Recognition Focused on Off-Line Handwriting," IEEE Transaction on Systems, Man, and Cybernetics-PART C: Applications and Reviews, VOL. 31, NO. 2, pp.216-233, May 2001.

[6] E. Gose, R. Johnsonbaugh, S. Jost, Pattern recognition and Image analysis, pp.338-346, Prentice Hall, 1996.

[7] R.O. Duda, P.E. Hart, Pattern recognition and scene analysis, John Wiley & Sons, 1973.


정 철 호

1994년 연세대학교 통계학과 졸업(이학사). 1998년 연세대학교 컴퓨터과학과 졸업(공학사). 2001년 연세대학교 컴퓨터과학과 졸업(석사). 2000년~현재 (주)칼라짚미디어 연구팀장. 2002년~현재 연세대학교 컴퓨터과학과 박사과정. 관심분야는 HCI, 영상인식, 유비쿼터스 컴퓨팅



한 탁 돈

1978년 연세대학교 전자공학과 졸업(학사). 1983년 Wayne State University 컴퓨터공학(공학석사). 1987년 University of Massachusetts at Amherst 컴퓨터공학 전공(박사). 1987년~1989년 Cleveland 주립대학 조교수. 1999년~2004년 국가지정연구실(3D 가속기 분야) 책임교수. 1989년~현재 연세대학교 컴퓨터과학과 교수. 관심분야는 HCI, ASIC 설계, 고성능 컴퓨터구조, 3D 가속기 구조, 유비쿼터스 컴퓨팅