선형 사진트리로 표현된 이진 영상의 면적과 둘레 길이를 계산하기 위한 상수시간 RMESH 알고리즘

김 기 원† · 우 진 운††

요 약

계층적 자료구조인 사진트리는 이진 영상을 표현하는데 매우 중요한 자료구조이다. 사진트리를 메모리에 저장하는 방법 중 선형 사진트리 표현 방법은 다른 표현 방법과 비교할 때 저장 공간을 매우 효율적으로 절약할 수 있는 이점이 있기 때문에 사진트리와 관련된 연산의 수행을 위해 선형 사진트리를 사용하는 효율적인 알고리즘 개발에 많은 연구가 진행되어 왔다. 본 논문에서는 RMESH(Reconfigurable MESH) 구조에서 3-차원 $n\times n\times n$ 프로세서를 사용하여 선형 사진트리로 표현된 이진 영상의 면적과 둘레 길이를 계산하는 알고리즘을 제안한다. 이 알고리즘은 $O(1)$ 시간 복잡도를 갖는다.

Constant Time RMESH Algorithm for Computing Area and Perimeter of Binary Image Represented by Linear Quadtrees

Gi Won Kim† · Jin Woon Woo††

ABSTRACT

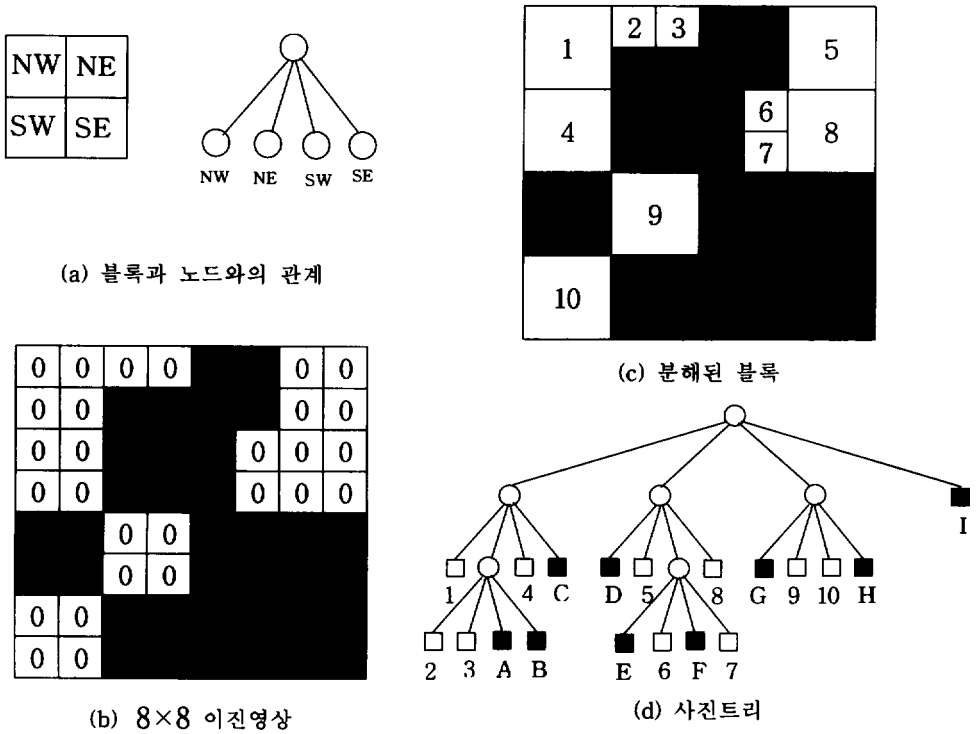
Quadtree, which is hierarchical data structure, is a very important data structure to represent binary images. A linear quadtree representation as a way to store a quadtree is efficient to save space compared with other representations. It, therefore, has been widely studied to develop efficient algorithms to execute operations related with quadtrees. In this paper, we present algorithms to compute the area and perimeter of binary images represented by quadtrees, using three-dimensional $n\times n\times n$ processors on RMESH(Reconfigurable MESH). Our algorithms have $O(1)$ time complexity.

1. 서 론

계층적 자료구조는 컴퓨터 그래픽, 영상처리, 지형처리, 패턴 인식 및 로보트 공학분야 등의 자료를 표현하는데 매우 적합한 기법이다. 특히 계층적 자료구조 중의 하나인 사진트리(quadtree)는 디지털 영상을 규칙적으로 분해(decomposition)하기 때문에 이진영상을 표현하는데 매우 유용한 자료구조이다[1,2].

$n\times n$ 이진 영상, ($n=2^k$, k는 양의 정수)에 대한 사진트리는 다음과 같이 정의된다. 사진트리의 루트(root) 노드는 전체 영상을 표현하는 것으로, 만약 영상의 모든 픽셀(pixel)들이 같은 색을 가진다면 루트 노드는 자식 노드를 갖지 않지만, 서로 다른 색을 가진다면 루트 노드는 4 개의 자식 노드를 갖는다. 자식 노드는 왼쪽부터 각각 영상의 NW, NE, SW 및 SE 블록(block)의 색을 표현한다(그림 1(a) 참조). 이와 같

※ 본 연구는 단국대학교의 1997년도 교내 연구비 지원에 의한 결과입니다.
† 준 회 원 : 단국대학교 전산통계학과
†† 종신회원 : 단국대학교 전산통계학과 교수
논문접수 : 1998년 4월 23일, 심사완료 : 1998년 6월 11일



(a) 블록과 노드와의 관계

(b) 8×8 이진영상

(c) 분해된 블록

(d) 사진트리

(그림 1) 이진 영상과 사진트리와의 관계
(Fig. 1) Relationship between a binary image and its quadtree

은 분해 과정은 노드가 표현하는 블록이 단지 하나의 공통된 색을 가지게 될 때까지 4 개의 자식 노드에 대해 순환적으로 적용된다.

예를 들면, 8×8 이진 영상을 사진트리로 표현해 보자. 일반적으로 이진 영상에서는, 그림 1(b)와 같이 WHITE는 0으로 BLACK은 1로 표현한다. 그림 1(c)는 그림 1(b)를 분해한 최종 결과를 블록으로 나타낸 것이고, 그림 1(d)는 그림 1(c)의 블록에 대해 사진트리로 표현한 것이다. 그림 1(c)와 (d)에서 WHITE 블록은 숫자, BLACK 블록은 영문자로 구별하였다. 그림 1(d)에서 사각형 BLACK 노드는 블록 전체가 1로 구성되어 있음을 의미하며, 사각형 WHITE 노드는 블록 전체가 0으로 구성되어 있음을 의미한다. 그리고 원형 노드는 내부 노드로서 GRAY 노드라 한다.

사진트리에서 레벨(level)은 루트 노드에서 임의의 노드까지의 거리로 정의하며, 루트 노드의 레벨은 0으로 한다. 그리고 사진트리의 높이는 $\log_4(n\times n)$ 값으로 정의한다. 사진트리의 높이가 h일 때 레벨이 l인 노드의 블록 크기를 $2^{(h-l)}\times 2^{(h-l)}$로 계산할 수 있다. 다시 말해서 이 블록은 $4^{(h-l)}$개의 픽셀들의 모임을 표현한다. 예를 들면, 그림 1(d)에서 노드 I는 4×4, 노드 D는 2×2, 노드 E는 1×1의 블록 크기를 갖는다.

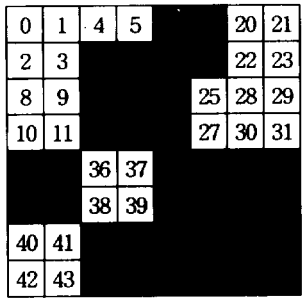
지금까지 사진트리를 메모리에 저장하기 위한 여러 가지 방법들이 제안되었다. 그 중 트리 구조를 사용하는 방법은 각 노드가 자신의 자식 노드를 가르키는 포인터 값을 저장하는 공간을 필요로 하므로 사진트리를 구성하는 노드들의 수가 많을 경우 포인터를 기억하기 위한 많은 저장 공간을 필요로 하는 단점이 있다. 이러한 단점을 보완하기 위하여 선형 사진트리(linear quadtree) 표현 방법을 사용한다[1].

선형 사진트리 표현 방법은 사진트리의 BLACK 노

드에 해당되는 블록의 위치와 크기에 관한 정보, 즉 (*Index*, *Level*)만을 저장하는 것이다. 이때 (*Index*, *Level*)을 위치 코드(locational code)라 한다. 여기에서 *Index*는 사진트리 노드에 해당하는 블록의 맨위 왼쪽에 있는 픽셀의 shuffled row-major 인덱스이다.

$n=2^i$, $i>0$인 $n\times n$ 이진 영상의 픽셀에 인덱스를 부여하는 방법은 여러 가지가 있으나, 그 중 가장 널리 사용되는 방법은 row-major 인덱스, column-major 인덱스, shuffled row-major 인덱스이다. Row-major 인덱스 방법은 r 행과 c 열의 픽셀에 $r\times n+c$의 인덱스를 부여하고, column-major 인덱스 방법은 r 행과 c 열의 픽셀에 $c\times n+r$의 인덱스를 부여하며, shuffled row-major 인덱스 방법은 r과 c의 이진 표현이 $r_{i-1}\cdots r_1 r_0$과 $c_{i-1}\cdots c_1 c_0$, $(i=\log_2 n)$일 때, 해당 픽셀에 이진수 표현으로 $r_{i-1}\,c_{i-1}\cdots r_1\,c_1\,r_0\,c_0$의 인덱스를 부여한다. 따라서 이러한 2가지 인덱스 사이의 상호 변환이 가능하며, 인덱스의 위치를 나타내는 행과 열 번호도 쉽게 구할 수 있다.

그림 1(b)의 8×8 이진 영상에 shuffled row-major 인덱스를 부여하면 그림 2(a)와 같고, 그림 1(d)의 BLACK 노드들을 위치 코드를 이용하여 선형 사진트리 표현으로 나타내면 그림 2(b)와 같다.



(a) 픽셀에 부여된 shuffled-row major 인덱스

BLACK

노드	A	B	C	D	E	F	G	H	I
Index	6	7	12	16	24	26	32	44	48
Level	3	3	2	2	3	3	2	2	1

(b) 선형사진트리

(그림 2) 위치 코드를 이용한 선형 사진트리 표현
(Fig. 2) The linear quadtree representation using locational code

그림 2(b)와 같이 선형 사진트리의 표현에서 WHITE 노드에 대한 정보는 저장하지 않고 BLACK 노드에 대한 정보만을 저장하는 이유는 사진트리를 다시 구축하지 않고도 BLACK 노드에 대한 정보를 이용하여 WHITE 노드에 대한 정보를 쉽게 구할 수 있으며, 또한 BLACK 노드에 대한 정보만을 저장하므로써 저장 공간을 최소화할 수 있는 장점이 있기 때문이다.

동일한 컴포넌트의 면적과 둘레 길이는 영상 처리의 응용에서 중요하게 사용되는 기하학적 성질에 속한다. 따라서 현재까지 단일 프로세서 뿐만 아니라 다양한 병렬 컴퓨터 구조에서 $n\times n$ 이진 영상의 면적과 둘레 길이를 계산하는 알고리즘들이 제안되었다[1,2,3,4,5]. 특히 병렬 컴퓨터와 관련하여 Hung과 Rosenfeld[4]는 $n\times n$ mesh 구조에서 이미지 컴포넌트의 이웃 찾기를 찾기 위한 $O(n)$ 시간 알고리즘을 제안하였으며, Jenq와 Sahni[5]는 $n\times n$ RMESH 구조에서 이미지 컴포넌트들의 면적과 둘레 길이를 각각 $O(\log n)$시간에 계산하는 알고리즘을 제안하였다.

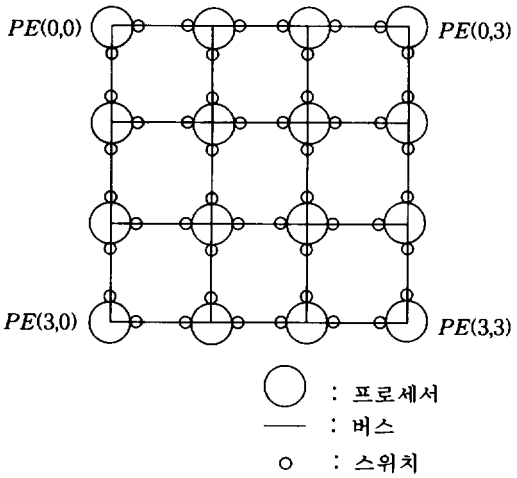
본 논문에서는 이진 영상의 면적과 둘레 길이를 계산하는 상수 시간 알고리즘들을 제안하며, 이 알고리즘들은 $n\times n\times n$ RMESH에서 $O(1)$ 시간 복잡도를 갖는다. 지금까지 $n\times n\times n$ RMESH 상에서 상수 시간을 갖는 많은 이진 영상 알고리즘들이 개발되었는데, 이진 영상과 선형사진트리 사이의 상호 변환 알고리즘[6,7], 선형사진트리의 집합 연산[8] 등을 들 수 있다. 본 논문에서 제안하는 알고리즘은 [5]의 알고리즘에 비하여 더 많은 프로세서들을 사용하지만 상수 시간의 시간 복잡도를 가지므로 시간적인 면에서 매우 효율적이라 할 수 있다.

2. RMESH 구조

RMESH는 Reconfigurable MESH의 약어로서 기존의 메쉬(mesh) 구조에 동적으로 재구성 가능한 버스 시스템을 결합한 구조로서 Miller, Prasanna-Kumar, Reisis, Stout에 의하여 제안되었으며[9], 구조적인 장점 때문에 다양한 분야에서 연구되었고 효율적인 알고리즘들이 개발되었다[10, 11, 12]. 또한 버스 시스템의 재구성 방법 면에서 서로 차이를 갖는 PARBUS 구조와 MRN 구조가 제안되었다[13, 14].

2.1 2-차원 RMESH

크기가 $n \times n$인 2-차원 RMESH의 기본 구조는 메쉬이며 프로세서들 사이의 통신을 위하여 브로드캐스트 버스(broadcast bus)가 존재한다. 예를 들어, 그림 3은 4×4 RMESH 구조를 보여준다. 프로세서들을 식별하기 위해 각 프로세서에게 $PE(i,j)$ 를 부여한다. 이때 $0 \le i,j < n$, i는 행의 인덱스이고, j는 열의 인덱스이다.



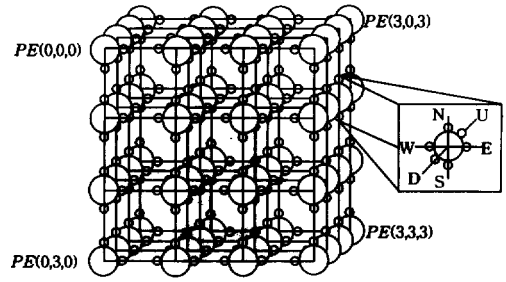
(그림 3) 4×4 RMESH 구조
(Fig. 3) 4×4 RMESH structure

브로드캐스트 버스상의 통신 제어를 위하여 버스 스위치가 있다. 버스 스위치들은 각 프로세서의 상, 하, 좌, 우에 하나씩 존재하는데, 이를 각각 N(north), S(south), W(west), E(east)라 한다. 버스 스위치는 각 프로세서의 소프트웨어에 의하여 $O(1)$ 시간에 조작되며, 스위치의 개폐 여부에 따라 브로드캐스트 버스를 다수의 서브버스(subbus)들로 재구성이 가능하다. 예를 들어, 각 프로세서가 자신의 S와 N 스위치를 끊고 E와 W 스위치를 연결한다면 여러 개의 서브버스가 형성되는데, 이를 행 버스(row bus)라 하고, 자신의 E와 W 스위치를 끊고 S와 N 스위치를 연결한다면 여러 개의 서브버스가 형성되는데, 이를 열 버스(column bus)라 한다.

두 개의 프로세서들은 충돌이 없는 한 공통된 하나의 특정 스위치를 동시에 개폐할 수 있다. 버스상에는 특정 시간에 단 하나의 프로세서만이 데이터를 실을 수 있으며, 서브버스 위에 실린 데이터는 단위 시간에 그 버스에 연결된 모든 프로세서에게 전달될 수 있다. 만약 한 프로세서가 서브버스상에 있는 모든 프로세서에게 레지스터(register) X의 값을 브로드캐스트하려면 broadcast(X) 명령을 사용하고, 브로드캐스트 버스의 내용을 읽어 레지스터 R에 저장하려면 R := content(broadcast bus) 명령을 사용한다. 따라서 데이터 브로드캐스트는 $O(1)$ 시간에 수행된다.

2.2 3-차원 RMESH

2-차원 RMESH를 확장하여 3-차원 RMESH를 구성할 수 있다. 3-차원 RMESH에서는 각 프로세서에게 $PE(l,i,j)$를 부여한다. 이때 $0 \le l,i,j < n$, l은 각 프로세서가 위치한 계층(layer)이고, i와 j는 계층 l에서의 행과 열의 인덱스이다. 예를 들어, 그림 4는 4×4×4 RMESH를 보여준다. 버스 스위치들은 기본적으로 2-차원 RMESH와 같이 N, S, W, E 스위치가 존재하며, 추가적으로 각 프로세서마다 계층을 연결하는 U(up)와 D(down) 스위치가 존재한다. 그리고 모든 프로세서의 N, S, W, E 스위치를 끊고 U와 D 스위치를 연결하면 여러 개의 서브버스가 형성되는데, 이를 UD 버스라 한다.



(그림 4) 4×4×4 RMESH 구조
(Fig. 4) 4×4×4 RMESH structure

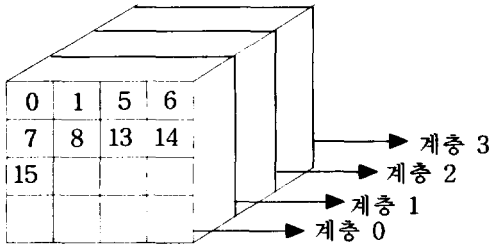
3. 기본적인 연산

여기서는 3차원 RMESH 구조에서 기하학적 성질들을 계산하기 위해 필요한 기본적인 연산들을 알아 본다.

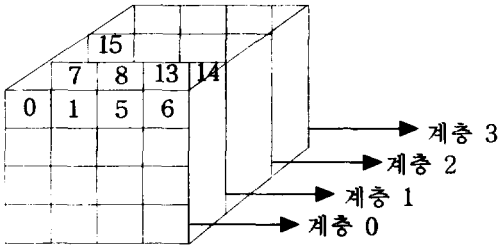
3.1 3차원 RMESH의 재구성

3차원 $n\times n\times n$ RMESH 구조를 $n\times n^2$ RMESH의 개념을 가진 구조가 되도록 재구성한다. 이 연산은 계층 0에 row-major 순서로 저장된 위치 코드들을 일련의 연속된 위치 코드들로 재구성하기 위해 사용되며, 계층 0의 행 i에 있는 위치 코드들을 계층 i의 행 0으로 이동시킨 후, 홀수 번째 계층에 있는 위치 코드를 역순으로 permute함으로써 만들어진다.

예를 들어, 4×4의 영상에서 9개의 위치 코드들이 존재할 때, 이 위치 코드들은 그림 5와 같이 4×4×4 RMESH의 계층 0에 속하는 프로세서에 row-major 순서로 하나씩 저장된다. 그림 5에서는 편의상 *Index* 값만을 보여준다.



(그림 5) 4×4×4 RMESH상의 계층 0의 초기 상태
(Fig. 5) The initial status of layer 0 on 4×4×4 RMESH



(그림 6) 계층 0의 행 i의 위치 코드를 계층 i의 행 0으로 이동
(Fig. 6) Moving the locational code in row i of layer 0 to row 0 of the i^{th} layer

이 연산을 위해서 세가지 작업이 차례로 일어난다. 첫째, UD 버스를 이용하여 행 i에 있는 위치 코드들을 계층 i로 이동시킨 후, 각 계층에서 열 버스를 이용하여 행 0으로 이동시킨다. 이 경우 데이터 이동에 걸리는 시간은 $O(1)$이며, 그림 8은 단계 1 후에 그림 6과 같게 된다.

둘째, 홀수 계층에 있는 위치 코드를 역순으로 permute시키는데, 그 과정은 다음과 같이 $O(1)$ 시간에 수행된다.

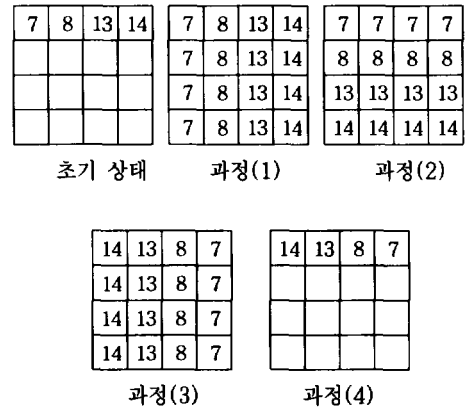
(1) 홀수 계층의 행 0에 있는 위치 코드를 열 버스를 이용하여 브로드캐스트한다.

(2) $PE(l,i,i)$, l=홀수, $0\le i<n$에 있는 위치 코드를 행 버스를 이용하여 브로드캐스트한다.

(3) $PE(l,i,j)$, l=홀수, $i+j=n-1$, $0\le i,j<n$에 있는 위치 코드를 열 버스를 이용하여 브로드캐스트한다.

(4) 행 0을 제외한 나머지 프로세서에 있는 위치 코드를 제거한다.

예를 들어, 그림 6의 계층 1에서는 그림 7과 같은 과정을 통하여 permute된다.



(그림 7) RMESH상에서 permute하는 과정
(Fig. 7) The process of permuting on RMESH

셋째, $n\times n\times n$ RMESH를 $n\times n^2$ RMESH의 개념으로 재구성하는 것으로, 그 과정은 다음과 같이 관련된 프로세서들 사이의 스위치만을 조작하므로 $O(1)$ 시간에 수행된다.

(1) $PE(l,i,0)$와 $PE(l,i,n-1)$, $0\le l,i<n$, 의 UD 버스만을 연결한다.

(2) $PE(l,i,n-1)$, l=짝수,의 D 버스를 끊고, $PE(l,i,n-1)$, l=홀수,의 U 버스를 끊는다.

(3) $PE(l,i,0)$, l=짝수,의 D 버스를 끊고, PE

$(l, i, 0)$, l=홀수,의 U 버스를 끊는다.

이 과정을 통하여 각 계층의 프로세서들은 인접한 계층의 프로세서들과 맨 좌측 혹은 맨 우측에서 연결된 형태로 재구성된다. 이러한 구성은 n^2 개의 프로세서들이 n 번 연속적으로 연결된 형태이다.

이상과 같이 이 연산은 $O(1)$ 시간에 수행되며, 그림 5의 초기 상태가 단계 1의 수행 후, 프로세서의 연결에 따라 계층들을 펼쳐 놓으면 그림 8과 같게 된다.

(그림 8) $n \times n^2$ RMESH

(Fig. 8) $n \times n^2$ RMESH

3.2 위치 코드의 분해

이 연산은 크기가 $s \times s$ $(1 < s \leq n)$인 블록을 나타내는 위치 코드를 그 블록에 포함된 1×1 블록을 나타내는 위치 코드들로 분해하는 것이다. 예를 들어, 높이가 2인 사진 트리에서 위치 코드 (12, 1)은 2×2 크기의 블럭이므로 이 위치 코드는 (12, 2), (13, 2), (14, 2), (15, 2) 라는 4개의 위치 코드들로 분해되어야 한다.

먼저 이 연산이 수행되기 전에, 분해될 위치 코드 (a, b)는 계층 0의 프로세서 $PE(0, i, j)$, $i = a'$ div n, $j = a'$ mod n 에 저장되어 있는 것으로 가정한다. 여기서 i와 j는 각각 shuffled row-major 순서 a에 대응하는 row major 순서 a'의 위치를 나타내는 행 번호와 열 번호에 해당한다.

이 연산은 다음과 같이 3단계로 수행될 수 있다.

(1) UD 버스를 이용하여 계층 0에 있는 위치 코드를 계층 b로 이동한다. 위치 코드 (12, 1)의 예를 들어보자. 먼저 이 위치 코드의 인덱스 12는 shuffled row-major 순서이므로 대응하는 row major 순서를 구하면 10이 된다. 따라서 이 위치 코드는 $PE(0,2,2)$에 존재하게 되면 이 단계에서 $PE(1,2,2)$로 이동한다.

(2) 각 계층에서 위치 코드를 받은 $PE(l, i, j)$는 N, S, W, E 버스를 이용하여 $PE(l, i+u, j+v)$, $0 \leq u < 2^{h-l}, 0 \leq v < 2^{h-l}$의 프로세서들과 연결되는 블록을 형성한다. 여기서 형성되는 프로세서 블록은 해당 위치 코드의 크기와 일치한다. 그리고 하나의 신호를 블록내의 프로세서들에게 브로드캐스트하며, 신호를 받은 프로세서들은 자신의 행 번호와 열 번호를 이용하여 새로운 위치 코드를 만든다. 즉 $PE(l, i, j)$는 위치 코드 $(n \times i + j, h)$을 만들어 낸다. 예를 들어, 위치 코드 (12, 1)을 전달받은 $PE(1,2,2)$는 $PE(1,2,3)$, $PE(1,3,2)$, $PE(1,3,3)$와 함께 블록을 형성하게 되고 각각 (12, 2), (13, 2), (14, 2), (15, 2)의 위치 코드들을 생성하게 된다.

(3) UD 버스를 이용하여 새로 생성된 위치 코드들을 계층 0로 이동한다. 이 과정에서 모든 위치 코드들이 계층 0의 프로세서들로 이동하게 되며, 사진 트리의 위치 코드들은 서로 중복되는 부분이 없기 때문에 이 과정에서 중복된 위치 코드들이 생성될 수 없다.

3.3 위치 코드의 이동

위치코드들이 $n \times n \times n$ RMESH의 계층 0에 속하는 프로세서에 하나씩 저장되어 있을 때, 위치 코드가 나타내는 *Index* 값을 row-major 인덱스의 행과 열 번호 (i, j)로 변환한 후, 이 위치 코드를 계층 0의 프로세서 *PE(0, i, j)*로 이동한다.

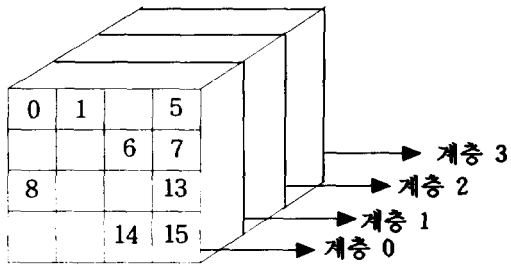
예를 들어, 그림 5에 주어진 위치 코드의 *Index*에 대해 row-major 인덱스의 행과 열 번호를 구하면 그림 9와 같게 된다.

Index:	0	1	5	6	7	8	13	14	15
(*i*, *j*):	(0,0)	(0,1)	(0,3)	(1,2)	(1,3)	(2,0)	(2,3)	(3,2)	(3,3)

(그림 9) 위치 코드의 *Index*를 행과 열번호로 바꾼 예

(Fig. 9) An example converting the *Index* into the row and column numbers

이와 같은 위치 코드의 이동은 [6]에 주어진 알고리즘에 의해 O(1) 시간에 수행될 수 있으며, 수행된 결과는 그림 10과 같이 이동된다.



(그림 10) 위치코드의 이동 결과
(Fig. 10) The result of the locational code moves

3.4 정 렬

정렬은 컴퓨터와 관련된 응용에서 매우 중요한 알고리즘이므로 재구성 가능한 메쉬 구조에서도 효율적인 알고리즘의 개발에 많은 연구가 이루어져 왔다.

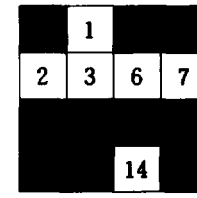
n 개의 데이터를 $O(1)$ 시간에 정렬하는 알고리즘을 개발하기 위해 초기에는 count sort 방법이 3-차원 $n\times n\times n$ RMESH[10, 11], PARBUS[15], MRN[14] 구조에 적용되었다. 그 후 사용되는 프로세서의 수를 줄이기 위한 노력이 계속되었는데, Jang과 Prasanna[16]는 $n\times n$ PARBUS에서 column sort 방법을 사용하여 $O(1)$ 시간에 정렬하는 알고리즘을 제안하였고, Nigam과 Sahni[17]는 column sort와 rotate sort 알고리즘을 각각 $n\times n$ RMESH에 적용하여 n 개의 데이터를 $O(1)$ 시간에 정렬할 수 있는 알고리즘을 제안하였다.

특히 Nigam과 Sahni는 rotate sort 알고리즘을 3-차원 $n\times n\times n$ RMESH에 적용하여 n^2 개의 데이터를 $O(1)$ 시간에 정렬할 수 있는 알고리즘을 제안하였다. 이 알고리즘에서 초기의 n^2 개의 데이터는 계층 0에 속하는 $PE(0,i,j)$ ($0\le i,j<n$)에 존재하며, 정렬된 결과는 계층 0에 속하는 프로세서에 row-major 순서로 하나씩 저장된다. 즉, $PE(0,0,0)$, $PE(0,0,1)$, $\cdots$, $PE(0,1,0)$, $PE(0,1,1)$, $\cdots$, $PE(0,n-1,n-1)$의 순서로 저장된다.

4. 면 적

면적(area)은 이진 영상의 동일한 컴포넌트에 속하는 픽셀들의 수를 의미하는 것으로, 면적을 계산하기 이전에 동일한 컴포넌트들을 찾기 위한 레이블링 알고리즘이 선행되어야 한다[4,5]. 일반적으로 레이블링 알고리즘은 선형사진트리로 표현된 영상에 대해서는 동일한 컴포넌트의 위치 코드에 같은 레이블을 부여한다. 따라서 컴포넌트의 레이블을 저장하기 위한 *Comp* 필드를 위치 코드에 추가해야만 한다.

예를 들어, 그림 11(a)의 이진영상에 대하여 그림 11(b)와 같이 위치코드와 *Comp* 필드가 주어졌다고 가정하자. 이때 *Comp* 필드는 선행된 레이블링 알고리즘에 의해 결정된 것으로 면적 계산에서는 입력으로 주어진다. 이와 같은 입력에 대해 같은 *Comp* 값을 갖는 위치 코드들의 블록 크기의 합이 해당 컴포넌트의 면적에 해당하며, 계산된 결과는 그림 11(b)의 *Area*에 주어진다.



(a) 이진영상의 예

Index	:	0	4	5	8	12	13	15
Level	:	2	2	2	1	2	2	2
Comp	:	1	2	2	3	3	3	3
Area	:	1	2	2	7	7	7	7

(b) 계산된 결과

(그림 11) 면적 구하기의 예
(Fig. 11) An example of computing area

면적 계산에서 k ($0<k\le n^2$) 개의 위치코드는 초기에 계층 0에 속하는 k 개의 프로세서에 row-major 순서로 하나씩 저장되어 있다고 가정한다. 면적을 계산하는 RMESH 알고리즘은 알고리즘 1과 같이 10 단계로 구성된다.

[단계 1]: 위치 코드 〈*Index*, *Level*, *Comp*〉를 갖는 계층 0의 프로세서는 *Index* 값의 row-major 인덱스의 행과 열 번호 (i, j)를 계산한다.

[단계 2]: 위치 코드를 계층 0의 프로세서 $PE(0,i,j)$로 이동한다.

[단계 3]: *Level* 값에 따라 위치 코드를 분해한다. 분해되는 위치 코드를 갖는 프로세서는 분해되기전의 위치 코드를 그대로 가지며 *New* = 0 의 값을 추가한다. 분해 과정에서 새로 생성되는 위치 코드를 갖는 프로세서는 *New* = 1 의 값을 위치 코드에 추가한다.

[단계 4]: 위치 코드의 *Comp* 필드값을 기준으로 정렬한다.

[단계 5]: 위치 코드를 갖는 계층 0의 프로세서 $PE(0,i,j)$는 $i\times n+j$ 의 값을 *RM_Id* 필드에 저장하여 위치 코드에 추가한다.

[단계 6]: 계층 0의 행 i에 있는 위치코드들을 계층 i의 행 0으로 이동시킨 후, 홀수번째 계층에 있는 위치코드를 역순으로 permute한다. 그리고 $n\times n\times n$ RMESH를 $n\times n^2$ RMESH의 개념을 가진 구조가 되도록 재구성한다.

[단계 7]: 같은 *Comp* 필드값을 갖는 프로세서들을 세그먼트로 분리한다.

[단계 8]: 각 세그먼트내의 프로세서는 *RM_ID* 값을 이용하여 자신이 속한 세그먼트의 길이를 계산하여 *Area* 필드에 저장하고, 위치 코드에 추가한다.

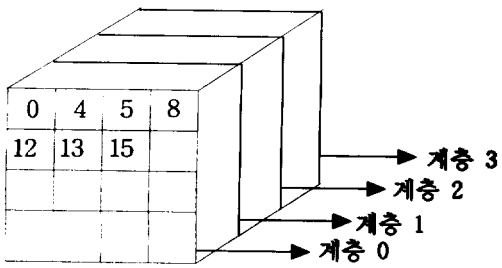
[단계 9]: 위치 코드들을 계층 0의 프로세서에 row-major 순서로 하나씩 저장한다.

[단계 10]: 위치 코드의 *New* 값이 1이면 *Index*을 ∞로 변경한다. 그리고 *Index* 값을 기준으로 하여 위치 코드를 정렬한다.

알고리즘 1. 면적을 계산하는 RMESH 알고리즘

이진 영상의 컴포넌트에 대한 면적을 계산하는 RMESH 알고리즘이 수행되는 과정을 단계별로 살펴보자. $n\times n$ 이진영상에 대하여 3-차원 $n\times n\times n$ RMESH 구조를 사용한다.

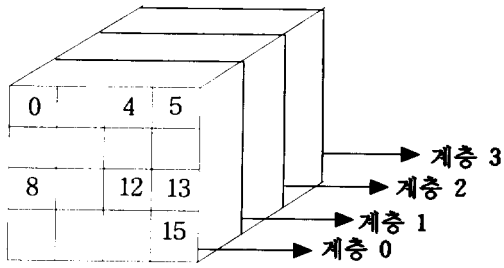
초기에 선형 사진트리로 표현된 위치코드들이 $n\times n\times n$ RMESH의 계층 0에 속하는 프로세서에 row-major 순서로 하나씩 저장되어 있다고 가정한다. 그림 11(b)의 위치코드에 대한 초기상태를 나타내면 그림 12와 같다.



(그림 12) 4×4×4 RMESH의 계층 0의 초기 상태
(Fig. 12) The initial state of 4×4×4 RMESH

[단계 1]에서 위치 코드의 *Index* 값은 shuffled row-major 인덱스이므로 row-major 인덱스의 행과 열 번호 (i, j) 를 단순 계산에 의해 바꿀 수 있음을 앞 절에서 보았다. 따라서 이 단계는 $O(1)$ 시간에 수행 가능하다.

[단계 2]에서 위치 코드는 단계 1에서 계산된 i, j에 따라 계층 0의 프로세서 $PE(0,i,j)$로 이동한다. 이 과정은 3.3절의 기본적인 연산에 해당하며 $O(1)$ 시간에 수행된다. 단계 2가 수행되면 그림 12의 위치 코드는 그림 13과 같이 이동된다.



(그림 13) 계층 0의 위치 코드를 $PE(0,i,j)$로 이동
(Fig. 13) Moving the locational codes to $PE(0,i,j)$

[단계 3]에서는 위치 코드를 분해한다. 이 과정은 3.2절의 기본적인 연산에 의해 $O(1)$시간에 수행 가능하다. 분해 과정에서 새로 생성되는 위치 코드의 *Comp* 값은 원래 위치 코드의 *Comp* 값을 그대로 적용한다. 분해 과정의 마지막 과정에서 생성된 위치 코드들이 계층 0의 프로세서들에 저장된다. 이때 분해되는 위치 코드를 가진 프로세서는 분해되기 전의 위치 코드를 그대로 저장하며 *New* = 0의 값을 추가하며, 새로 생성된 위치 코드를 가진 프로세서는 *New* = 1의 값을 추가한다.

[단계 4]에서는 위치 코드의 *Comp* 필드값을 기준으로 정렬한다. 이 정렬 과정은 3.4절의 정렬 알고리즘에 의해 $O(1)$ 시간에 수행된다.

[단계 5]에서 위치 코드를 갖는 계층 0의 프로세서 $PE(0,i,j)$는 $i\times n+j$ 의 값을 *RM_Id* 필드에 저장하여 위치 코드에 추가한다. 이 단계는 단순 계산만이 요구되므로 $O(1)$ 시간에 수행 가능하다.

[단계 6]은 3.1절의 기본적인 연산으로 $n\times n\times n$ RMESH를 $n\times n^2$ RMESH의 개념을 가진 구조가 되

도록 재구성한다. 이 과정을 통하여 각 계층의 프로세서들은 인접한 계층의 프로세서들과 맨 좌측, 혹은 맨 우측에서 연결된 형태로 재구성된다. 이러한 구성은 n^2 개의 프로세서들이 n 번 연속적으로 연결된 형태이며, 이 단계는 $O(1)$시간에 수행 가능하다.

〔단계 7〕에서 각 프로세서는 자신의 *Comp*값과 왼쪽 프로세서의 *Comp*값을 비교하여 다르면 *Segment* 레지스터에 1을 저장하고, 같으면 0을 저장한다. 그리고 맨 앞 프로세서의 *Segment*에는 무조건 1로 저장한다. 여기서 *Segment*가 1인 프로세서는 하나의 세그먼트의 시작을 의미한다. 그러므로 세그먼트를 분리하기 위해 *Segment*의 값이 1인 프로세서의 W 스위치를 끊는다. 이 단계에서는 프로세서들이 왼쪽 프로세서만을 접근하여 계산을 하기 때문에 소요 시간은 $O(1)$ 이다.

〔단계 8〕에서는 *Segment*가 1인 프로세서는 세그먼트 내의 프로세서들에게 자신의 *RM_ID*를 전달하며, 마지막에 위치한 프로세서가 (자신의 *RM_ID*값 − 전달받은 *RM_ID*값 + 1)을 계산함으로써 해당 세그먼트의 길이를 구할 수 있다. 이 길이를 세그먼트 내의 프로세서들에게 브로드캐스트하면 각 프로세서들은 이 값을 *Area* 필드에 저장한 후 위치 코드에 추가한다. 이 단계에서는 같은 세그먼트에 속하는 프로세서들 사이의 전달만이 필요하고 전달받은 값을 *Area* 필드에 저장만 하므로 소요 시간은 $O(1)$이다.

〔단계 9〕에서는 위치코드들을 계층 0의 프로세서에 row-major 순서로 저장하는 단계로서, 먼저 계층 i의 행 0에 있는 위치코드들을 계층 0의 행 i로 이동시킨다. 이 과정은 단계 6의 과정을 역순으로 수행할 수 있으며, 소요 시간은 $O(1)$이다.

〔단계 10〕는 초기에 입력된 위치 코드들만을 남기기 위한 작업을 수행하는 단계로서, 이것은 단계 3에서 새로 생성된 위치 코드를 제거함으로써 가능하다. 따라서 위치 코드의 New 값이 1이면 *Index*을 ∞로 변경한 후, *Index* 값을 키로 하여 위치 코드를 정렬한다. 이 단계의 주된 부분은 정렬이므로 $O(1)$ 시간에 수행 가능하다.

지금까지 알고리즘 2의 각 단계가 모두 $O(1)$ 시간에 수행될 수 있음을 보았으며, 그림 14는 그림 11의 면적을 계산하는 과정을 단계별로 보여준다.

〔단계 1〕*Index* : 0 4 5 8 12 13 15
Level : 2 2 2 1 2 2 2
Comp : 1 2 2 3 3 3 3
〔단계 2〕 위치 코드의 이동
〔단계 3〕*Index* : 0 4 5 8 9 10 11 12 13 15
Level : 2 2 2 1 2 2 2 2 2 2
New : 0 0 0 0 1 1 1 0 0 0
〔단계 4〕*Comp* : 1 2 2 3 3 3 3 3 3 3

〔단계 5〕*RM_ID* : 0 1 2 3 4 5 6 7 8 9
〔단계 6〕 위치 코드의 이동
〔단계 7〕*Segment* : 1 1 0 1 0 0 0 0 0 0
〔단계 8〕*Area* : 1 2 2 7 7 7 7 7 7 7
〔단계 9〕 위치 코드의 이동

〔단계 10〕*Index* : 0 4 5 8 12 13 15
Level : 2 2 2 1 2 2 2
Comp : 1 2 2 3 3 3 3
Area : 1 2 2 7 7 7 7

(그림 14) 알고리즘 1의 단계별 수행 과정
(Fig. 14) The process of each step of algorithm 1

지금까지 알고리즘 1의 각 단계가 모두 $O(1)$ 시간에 수행될 수 있음을 설명하였으며, 정리 1과 같이 요약할 수 있다.

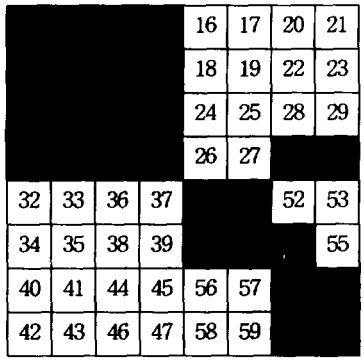
정리 1 $k(0<k\leq n^2)$ 개의 위치 코드가 $n\times n\times n$ RMESH의 계층 0에 row-major 순서로 각 프로세서에 하나씩 저장되어 있을 때, 면적 계산 알고리즘은 $O(1)$ 시간에 수행된다.

5. 둘레 길이

둘레 길이(perimeter)는 이진 영상의 동일한 컴포넌트에 속하는 BLACK 픽셀들 중 WHITE 픽셀과 인접한 픽셀들의 수를 의미하는 것으로, 둘레 길이를 계산하기 이전에 면적 계산과 마찬가지로 동일한 컴포넌트들을 찾기 위한 레이블링 알고리즘이 선행되어야 한다〔4,5〕. 따라서 컴포넌트의 레이블을 저장하기 위한 *Comp* 필드가 위치 코드에 추가된다.

예를 들어, 그림 15(a)의 이진 영상에 대하여 그림 15(b)와 같이 위치 코드와 *Comp* 필드가 주어졌다고 가정하자. 이때 *Comp* 필드는 선행된 레이블링 알고리

즘에 의해 결정된 것으로 둘레 길이의 계산에서는 입력으로 주어진다. 이와 같은 입력에 대해 같은 *Comp* 값을 갖는 위치 코드들에서 WHITE 픽셀과 인접한 BLACK 픽셀들의 수가 해당 컴포넌트의 둘레 길이에 해당하며, 계산된 결과는 그림 15(b)의 *Perimeter*에 주어진다. 이 예에서 보는 바와 같이 전체 영상의 경계 부분에 있는 픽셀들(예: 0, 1, 31, 63 등)은 모두 WHITE 픽셀에 인접한 것으로 간주된다.



(a) 이진 영상의 예

Index	:	0	30	31	48	54	60
Level	:	1	3	3	2	3	2
Comp	:	1	2	2	3	3	3
Perimeter	:	12	2	2	9	9	9

(b) 계산된 결과

(그림 15) 둘레 길이를 구하는 예
(Fig. 15) An example of computing perimeter

면적 계산에서와 마찬가지로 둘레 길이의 계산에서도 $k(0<k\leq n^2)$ 개의 위치코드가 초기에 계층 0에 속하는 k 개의 프로세서에 row-major 순서로 하나씩 저장된다. 둘레 길이를 계산하는 RMESH 알고리즘은 알고리즘 2와 같이 10 단계로 구성되며, 면적 계산 알고리즘과 유사하다.

〔단계 1〕: 위치 코드 〈*Index*, *Level*, *Comp*〉를 갖는 계층 0의 프로세서는 *Index* 값의 row-major 인덱스의 행과 열 번호 (i, j)를 계산한다.

〔단계 2〕: 위치 코드를 계층 0의 프로세서 $PE(0,i,j)$로 이동한다.

〔단계 3〕: *Level* 값에 따라 위치 코드를 분해한다. 분해되는 위치 코드를 갖는 프로세서는 분해되기전의 위치 코드를 그대로 가지며 $New = 0$ 의 값을 추가한다. 분해 과정에서 새로 생성되는 위치 코드를 갖는 프로세서는 $New = 1$ 의 값을 위치 코드에 추가한다.

〔단계 4〕: 위치 코드를 갖는 계층 0의 프로세서는 자신의 *Comp* 값과 인접한 프로세서의 *Comp* 값을 비교하여 모두 같으면 *Comp* 필드값에 ∞을 저장한다. 그리고 위치 코드의 *Comp* 필드값을 기준으로 정렬한다.

〔단계 5〕: 위치 코드를 갖는 계층 0의 프로세서 $PE(0,i,j)$는 $i\times n+j$ 의 값을 *RM_Id* 필드에 저장하여 위치 코드에 추가한다.

〔단계 6〕: 계층 0의 행 i에 있는 위치코드들을 계층 i의 행 0으로 이동시킨 후, 홀수번째 계층에 있는 위치코드를 역순으로 permute한다. 그리고 $n\times n\times n$ RMESH를 $n\times n^2$ RMESH의 개념을 가진 구조가 되도록 재구성한다.

〔단계 7〕: 같은 *Comp* 필드값을 갖는 프로세서들을 세그먼트로 분리한다.

〔단계 8〕: 각 세그먼트내의 프로세서는 *RM_ID* 값을 이용하여 자신이 속한 세그먼트의 길이를 계산하여 *Perimeter* 필드에 저장하고, 위치 코드에 추가한다.

〔단계 9〕: 위치 코드들을 계층 0의 프로세서에 row-major 순서로 하나씩 저장한다.

〔단계 10〕: 위치 코드의 *New* 값이 1이면 *Index*를 ∞로 변경한다. 그리고 *Index* 값을 키로 하여 위치 코드를 정렬한다.

알고리즘 2. 둘레 길이를 계산하는 RMESH 알고리즘

이진 영상의 컴포넌트에 대한 둘레 길이를 계산하는 RMESH 알고리즘이 수행되는 과정은 단계 4를 제외하고는 면적을 계산하는 알고리즘 1과 유사하다. 따라서 단계 4의 과정만을 살펴보기로 한다.

〔단계 4〕에서 위치 코드를 갖는 계층 0의 프로세서는 자신 *Comp* 값과 인접한 프로세서의 *Comp* 값을 비교하게 되는데, 모두 같은 값을 갖는다면 이 프로세서가 가진 픽셀은 둘레에 속하지 않게 되므로 둘레 길이의 계산에서 제외되어야 한다. 따라서 *Comp* 필드값에 ∞을 저장한다. 그리고 위치 코드의 *Comp* 필드값을 기준으로 정렬한다. 여기서 *Comp* 값을 비교하는 것은 W, E, N, S의 순서대로 인접 프로세서 간에 이루어질 수 있으므로 $O(1)$ 시간에 수행 가능하다. 그리고 정렬 과정은 3.4절에 따라 $O(1)$ 시간에 수행될 수 있으므로 이 단계는 전체적으로 $O(1)$ 시간에 수행 가능하게 된다.

알고리즘 2에서 단계 4를 제외한 다른 단계들은 면적 알고리즘의 단계와 같은 과정을 수행하므로 전체 수행 시간은 $O(1)$ 시간에 수행될 수 있다. 그림 16은 그림 15의 위치코드들의 둘레 길이를 계산하는 과정을 단계별로 보여준다.

〔단계 1〕 *Index* : 0 30 31 48 54 60
Level : 1 3 3 2 3 2
Comp : 1 2 2 3 3 3
〔단계 2〕 위치 코드의 이동
〔단계 3〕 위치 코드의 분해
〔단계 4〕 *Index* : 0 1 2 4 5 7 8 10 11 13 14 15 30 31 48 49 50 54 60 61 62 63
Level : 1 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 2 3 3 3
Comp : 1 1 1 1 1 1 1 1 1 1 1 2 2 3 3 3 3 3 3 3
New : 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 0 1 1 1
〔단계 5〕 *RM_ID*: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
〔단계 6〕 위치 코드의 이동
〔단계 7〕 *Segment*: 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0
〔단계 8〕 *Perimeter*: 12 12 12 12 12 12 12 12 12 12 12 2 2 8 8 8 8 8 8 8
〔단계 9〕 위치 코드의 이동
〔단계 10〕 *Index* : 0 30 31 48 54 60
Level : 1 3 3 2 3 2
Comp : 1 2 2 3 3 3
Perimeter : 12 2 2 8 8 8

(그림 16) 알고리즘 2의 단계별 수행 과정
(Fig. 16) The process of each step of algorithm 2

지금까지 알고리즘 2의 각 단계가 모두 $O(1)$ 시간에 수행될 수 있음을 설명하였으며, 정리 2와 같이 요약할 수 있다.

정리 2 k ($0<k\leq n^2$) 개의 위치 코드가 $n\times n\times n$ RMESH의 계층 0에 row-major 순서로 각 프로세서에 하나씩 저장되어 있을 때, 둘레 길이 계산 알고리즘은 $O(1)$ 시간에 수행된다.

6. 결 론

본 논문에서는 3-차원 $n\times n\times n$ RMESH 구조에서 선형 사진트리로 표현된 이진 영상의 면적과 둘레 길이를 계산하기 위한 상수 시간 알고리즘을 제안하였다. 이 알고리즘은 3-차원 RMESH 구조에서 유용한 기본적인 연산들을 사용하였으며, 이 연산들은 모두 $O(1)$ 상수 시간 복잡도를 가진다.

특히 본 알고리즘에서는 위치 코드를 이진 영상으로 변환하지 않고 직접 위치 코드에 기본 연산들을 적용함으로써 시간 지체를 줄인다.

참 고 문 헌

[1] H. Samet, Application of Spatial Data Structures, Computer Graphics, Image Processing, and GIS. Addison-Wesley, 1990.
[2] H. Samet, The Design and Analysis of Spatial Data Structures, Addison-Wesley, 1990.

[3] H. Samet and M. Tamminen, "Computing Geometric Properties of Images Represented by Linear Quadtrees," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.PAMI-7, No.2, March 1985.

[4] Y. Hung and A. Rosenfeld, "Parallel Processing of Linear Quadtrees on a Mesh-Connected Computer," Journal of Parallel and Distributed Computing, Vol.7, pp.1-27, 1989.

[5] J. Jenq and S. Sahni, "Reconfigurable Mesh Algorithms for The Area And Perimeter Of Image Components," 1991 International Conference on Parallel Processing, Vol.Ⅲ, pp.280-281, 1991.

[6] 김 명, 장주욱, "재구성가능 매쉬에서 $O(1)$ 시간 복잡도를 갖는 이진영상/사진트리 변환 알고리즘," 정보과학회논문지(A), 제23권, 제5호, pp.454-466, 1996.

[7] 공헌택, 우진운, "RMESH 구조에서의 선형 사진트리 구축을 위한 상수 시간 알고리즘," 정보처리 논문지, 제4권, 제9호, pp.2247-2258, 1997.

[8] 공헌택, 우진운, "RMESH 구조에서 선형사진트리의 집합 연산을 위한 상수 시간 알고리즘," 정보과학회 논문지(A), 제24권, 제11호, pp.1218-1231, 1997.

[9] R. Miller, V. Prasanna-Kumar, D. Reisis, and Q. Stout, "Parallel Computation on Reconfigurable Meshes," IEEE Transactions on Computers, Vol.42, No.6, pp.678-692, 1993.

[10] J. Jenq and S. Sahni, "Reconfigurable Mesh Algorithms for The Hough Transform," Proceedings of International Conference on Parallel Processing, Vol.Ⅲ, pp.34-41, 1991.

[11] J. Jenq and S. Sahni, "Reconfigurable Mesh Algorithms for Image Shrinking, Expanding, Clustering, and Template Matching," Proceedings 5th International Parallel Processing Symposium, pp.208-215, 1991.

[12] 김 홍근, 조 유근, "단순다각형의 내부점 가시도를 위한 효율적인 RMESH 알고리즘," 정보학회 논문지, 제20권 11호, pp.1693-1701, 1993.

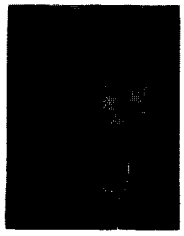
[13] J. Jang, H. Park, and V. Prasanna, "A Fast Algorithm for Computing Histogram on a Reconfigurable Mesh," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.17, No.2, pp.97-106, 1995.

[14] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, "The Power of Reconfiguration," Journal of Parallel and Distributed Computing, 13, pp.139-153, 1991.

[15] B. Wang, G. Chen, and F. Lin, "Constant Time Sorting on a Processor Array with a Reconfigurable Bus System," Information Processing Letters, 34, 4, pp.187-190, 1990.

[16] J. Jang and V. Prasanna, "An Optimal Sorting Algorithm on Reconfigurable Meshes," Proceedings 6th International Parallel Processing Symposium, 1992.

[17] M. Nigam and S. Sahni, "Sorting n Numbers On $n \times n$ Reconfigurable Meshes With Buses," Proceedings 7th International Parallel Processing Symposium, pp.174-181, 1993.



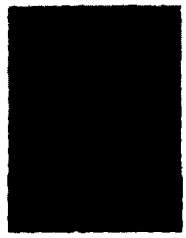
김 기 원

1988년 단국대학교 계산통계학과 (학사)

1992년 단국대학교 전산통계학과 (석사)

1994년~현재 단국대학교 전산통계학과 박사 과정(수료)

관심분야 : 알고리즘, 병렬 처리, 병렬 알고리즘



우 진 운

1980년 서울대학교 수학교육과(학사)

1989년 미국 Univ. of Minnesota 전산학과 (박사)

1980년~1983년 대한항공 및 국토개발연구원 전산실 근무

1989년~현재 단국대학교 전산통계학과 부교수

관심분야 : 알고리즘, 병렬 처리, 병렬 알고리즘